香港城市大學
City University of Hong Kong

CS 6290

Privacy-enhancing Technologies

Department of Computer Science

Slides credit in part from E. Ben-Sasson, D. Dziembowski, I. Eyal, R. Geambasu, and F. Greenspan, A. Judmayer, A. Juels, A. Miller, J. Poon, V. Shmatikov, D. Song, and F. Zhang

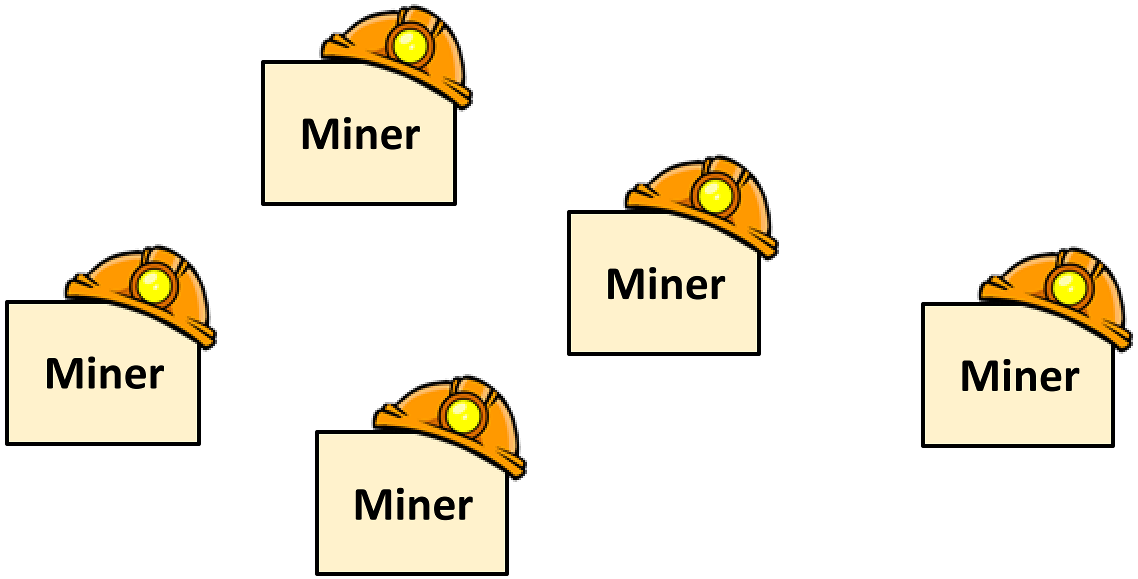
Lecture 3 – Energy-efficient Consensus Mechanisms

Prof. Cong WANG

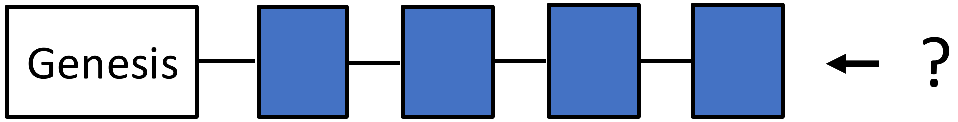
CS Department

City University of Hong Kong

Recall: PoW Mining

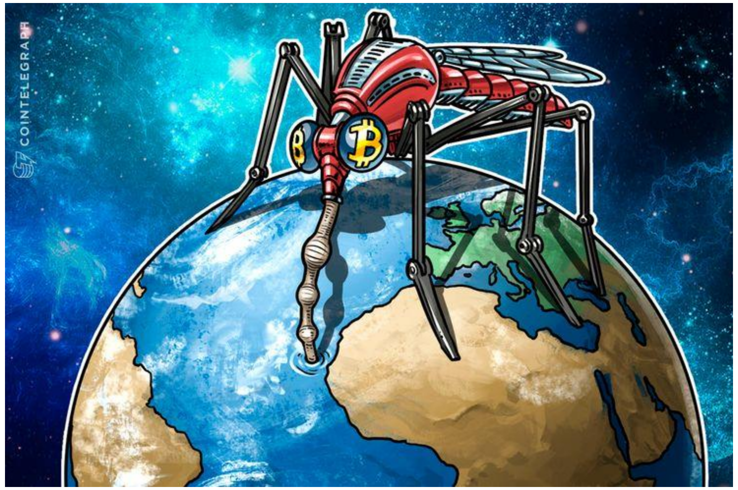


Pr [Success] proportional to computing power



Bitcoin PoW: Criticism

- Relies on an ongoing **computational race**
  - Honest majority of computing power is critical
- Burns energy (**Proof of Work = Proof of Waste?**)
  - 10+ GW (http://realtimebitcoin.info)
    - Ireland consumes 3.1 GW and Austria 8.2 GW



Challenge: Replace PoW with Alternate Resource Lottery

- Other physical resources, with different properties?
  - Disk space
  - Useful computation/storage
  - Etc …

- What about the coin itself?
  - “Virtual resource mining” --> <u>Proof of Stake (PoS)</u>

Challenge: Replace PoW with Alternate Resource Lottery

- Other physical resources, with different properties?
  - Disk space
  - Useful computation/storage
  - Etc …

- What about the coin itself?
  - “Virtual resource mining” --> Proof of Stake (PoS)

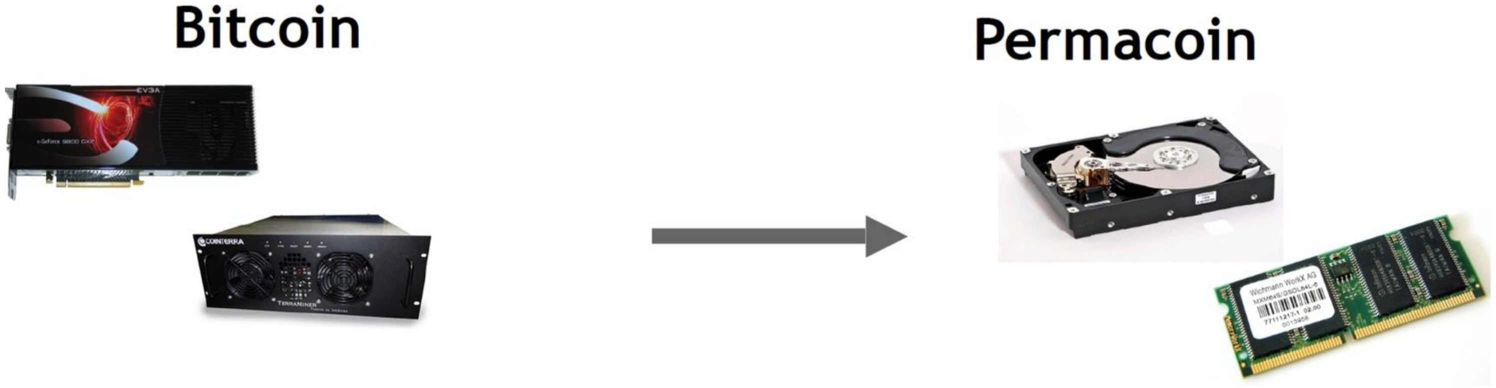
First question: can we recycle the waste of Bitcoin and do something useful?

Greening Bitcoin

Proof-of-useful-works

Repurposing Bitcoin Work

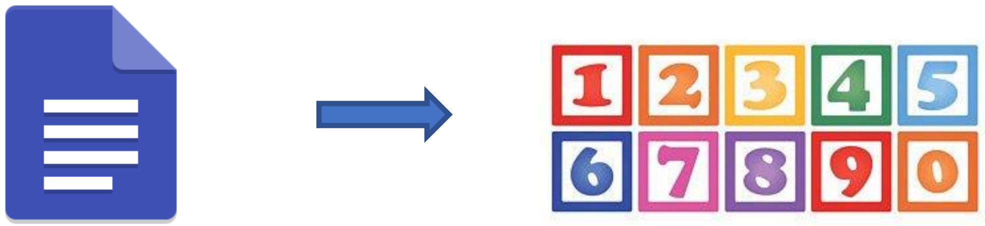
- <u>Permacoin:</u>
  - Idea: a puzzle where hardware investment is useful, even if the work is wasted?



- Side effect:
  - Massively distributed, replicated storage system

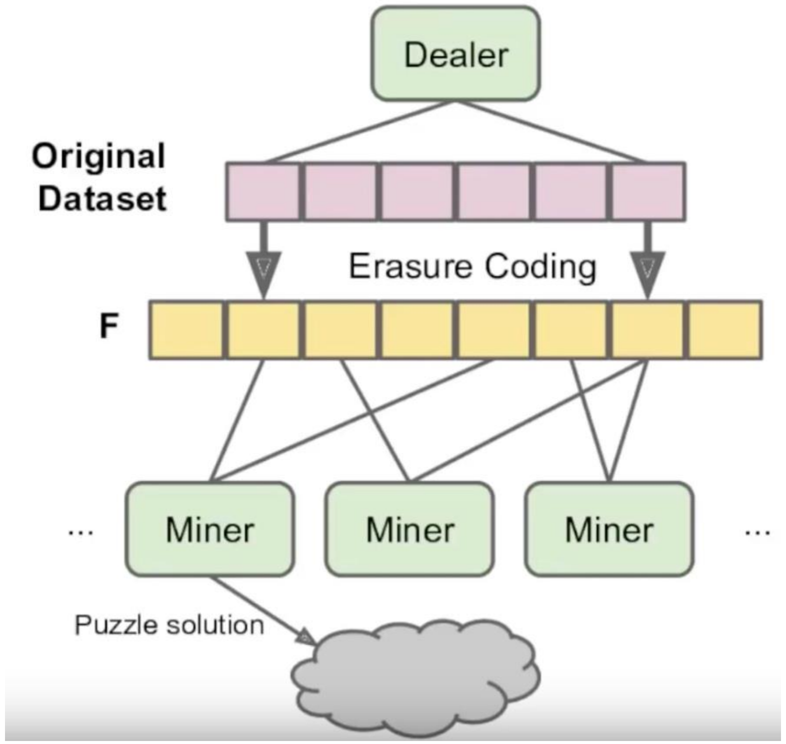
Permacoin: Setting

- Assume we have a large public file **F** to store
- For simplicity: **F** is chosen globally, at the beginning, by a **<u>trusted dealer</u>**
- Each user (miner) stores a random subset of the file

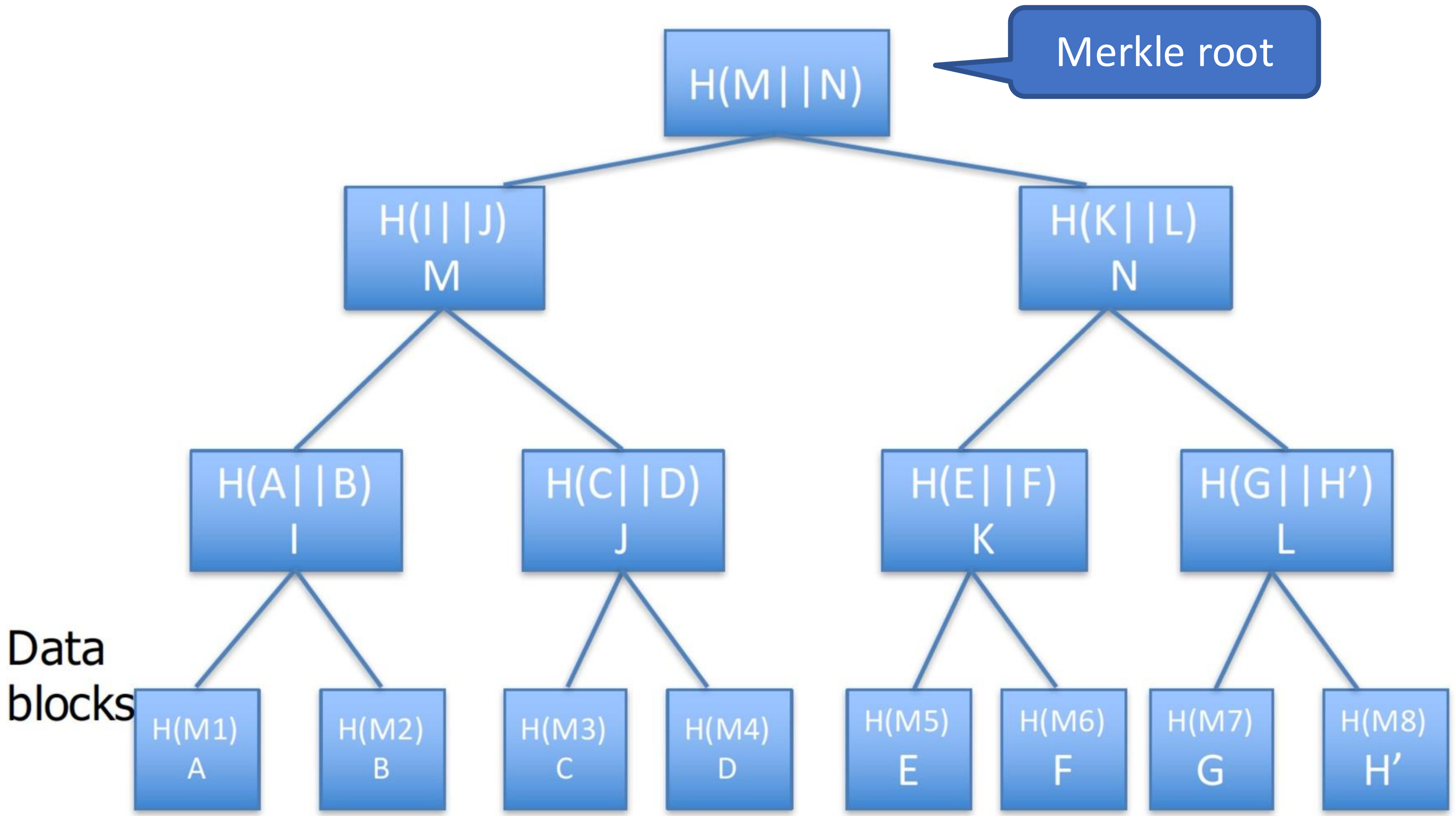


Permacoin: Features

- **Storage**: a large public dataset
  - Optionally, users can submit their own data to archive
- **Recoverability**: even after catastrophic failure
  - Thanks to Erasure Coding (will introduce later)
- **Diversity**: geographical, as well as administrative
  - Store at individual miners

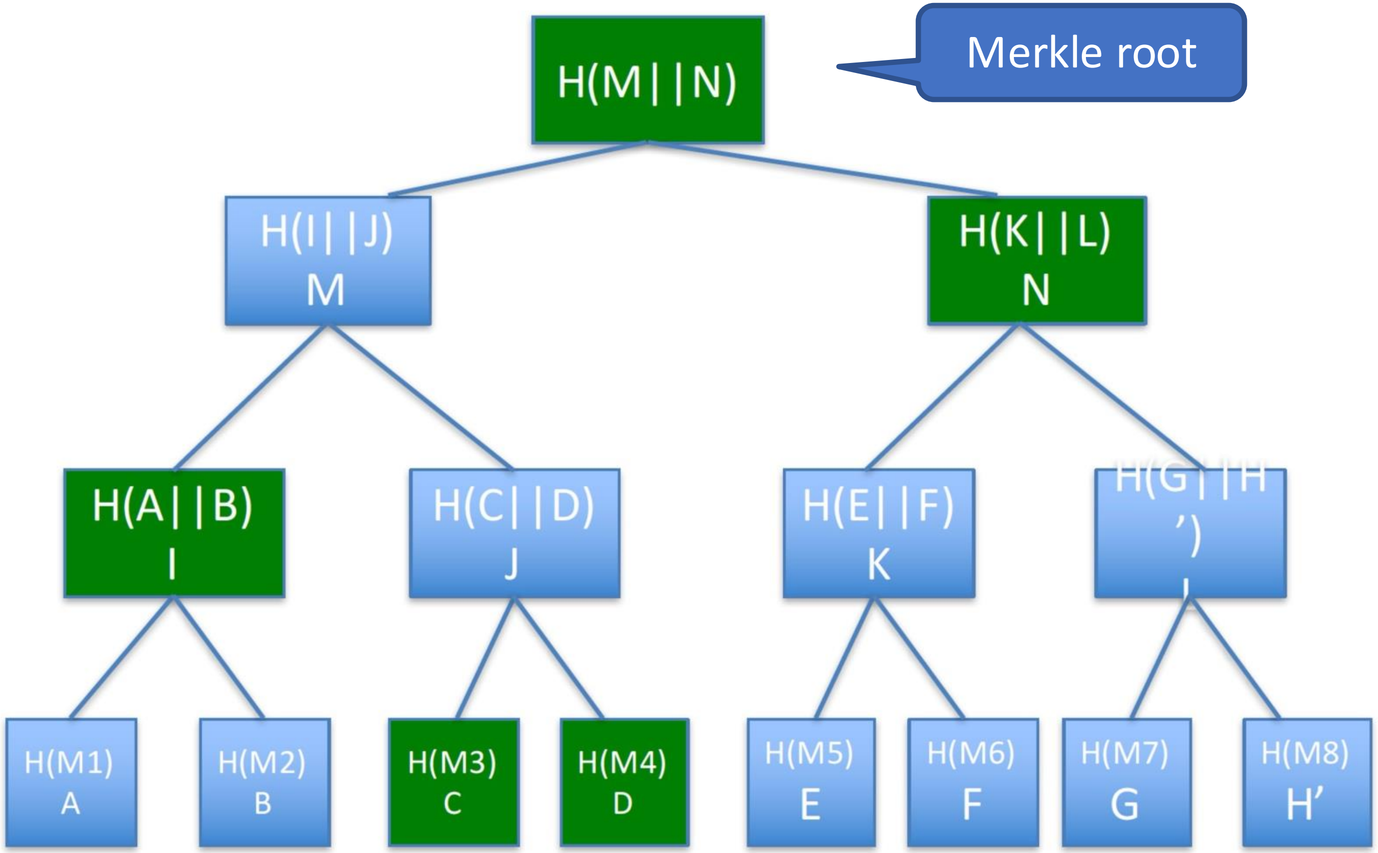


Background: Merkle Tree



To check D, Proof = <H(M4), H(M3), H(A||B), H(K||L), H(M||N)> should match with root

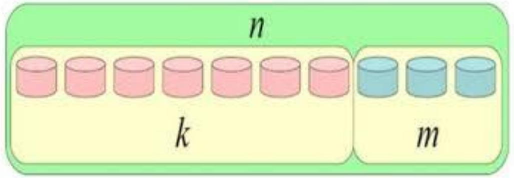
Background: Merkle Tree



To check D, Proof = <H(M4), H(M3), H(A||B), H(K||L), H(M||N)>
should match with root

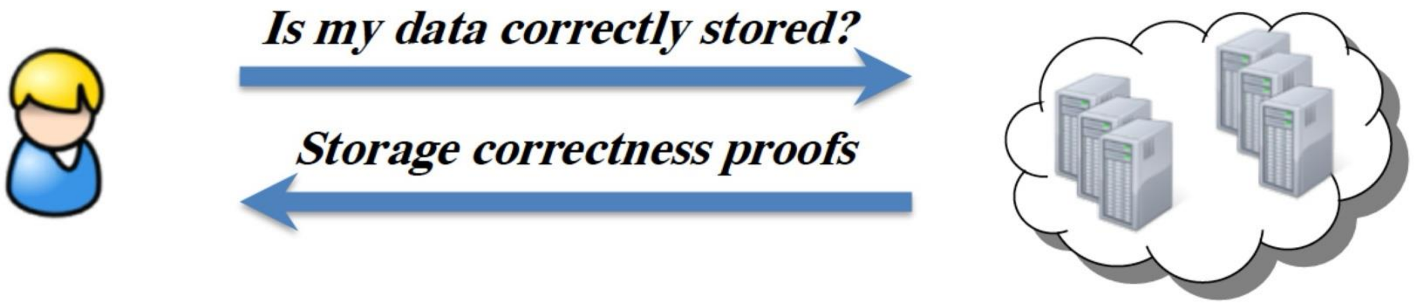
Background: Erasure Codes

- k data blocks $\xrightarrow{encode}$ $k + m$ blocks
  - Example: Reed-Solomon 6+3



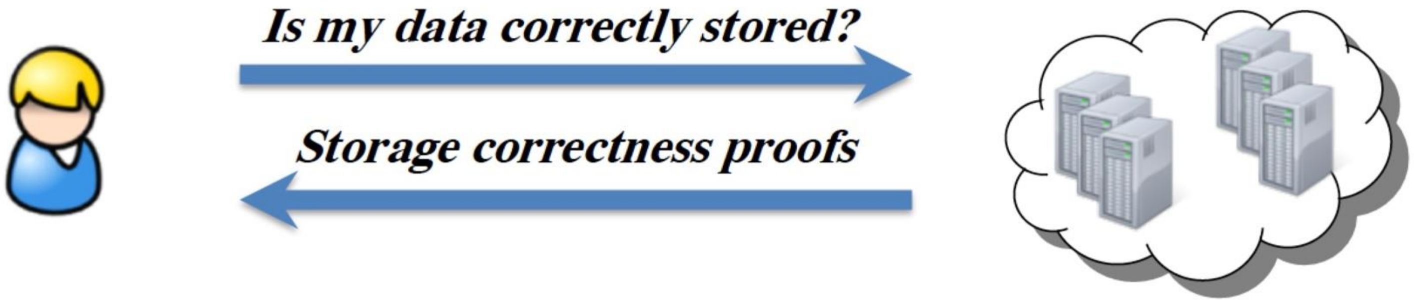
- Reliability: can tolerate m failures
- Efficiency:
  - Save disk space
    - Compared to previous effort: one data block replicate to many (say 3) replicas
  - Save I/O bandwidth on the write path

Background: Data Auditing



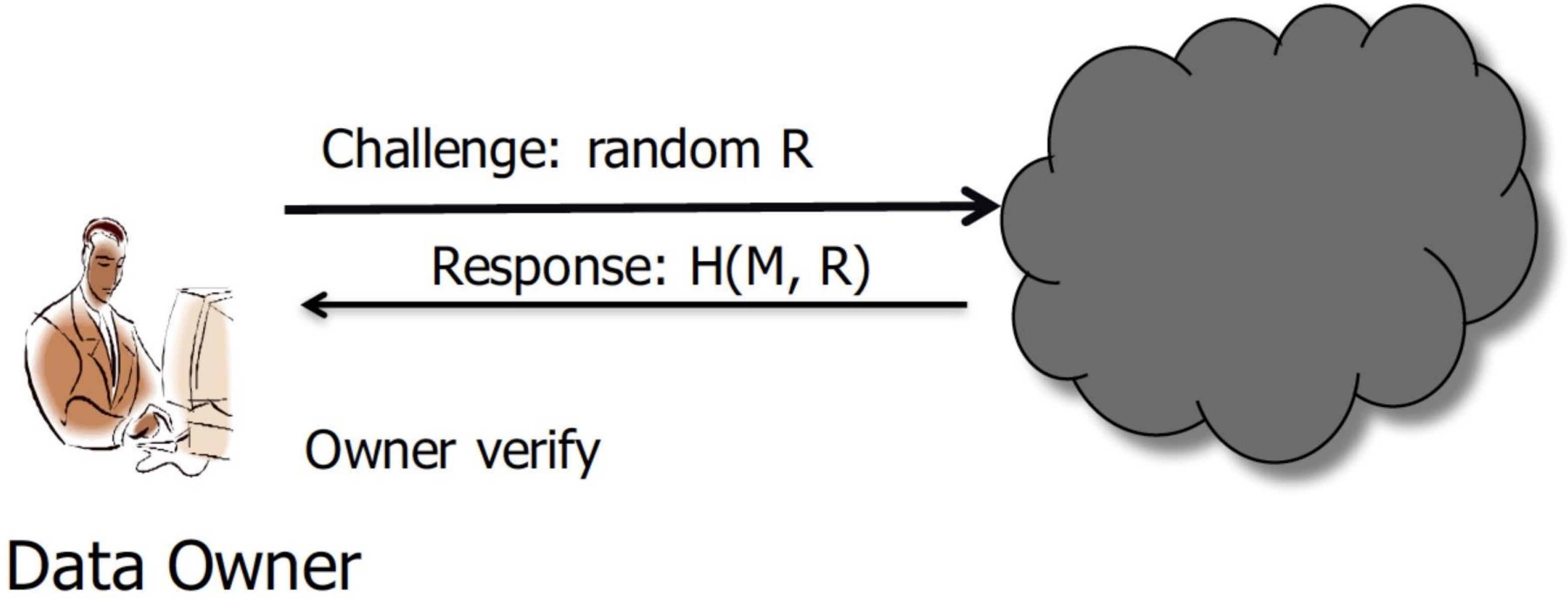
- Client stores data on the storage providers
- Providers can behave unfaithfully
  - Discard old data
  - Hide data loss
- <u>Client needs a guarantee that data is stored correctly</u>
  - To help extend data trust perimeter into the storage providers
  - To meet security, system, and performance requirements

Background: PoR



- Proofs of Retrievability (PoR):
  - The client (verifier) runs an efficient data audit proof in which the data storage server (prover) proves that **it still possesses the client's data <u>and client can recover entire file</u>**

How to Audit



Secure Storage Auditing

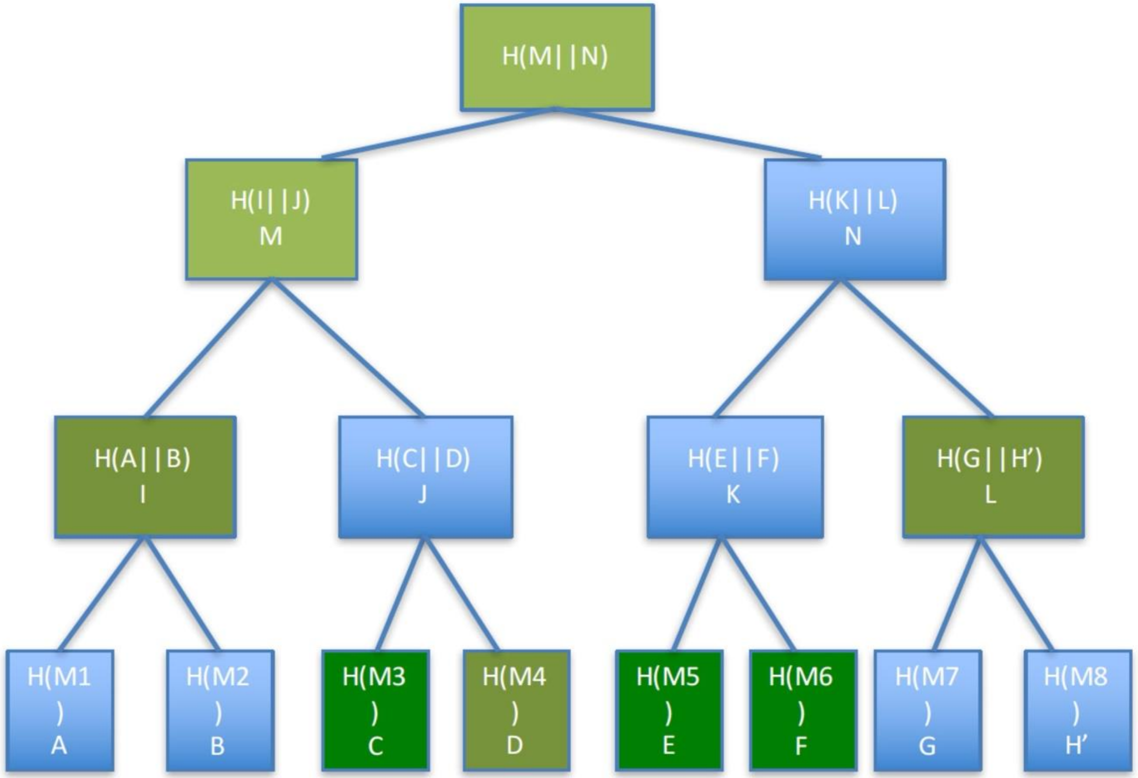
- Demand efficient storage correctness guarantee without requiring local data copies
  - Traditional methods for storage security can not be directly adopted
  - Retrieving massive data for checking is unpractical (large bandwidth)
- Allow meaningful tradeoffs between security and overhead
  - Communication and computation costs should be low
  - Auditing cost should not outweigh its benefits
- Cope with frequent data changing while ensuring continuous data auditing
  - Data may be frequently updated by owner for application purposes
  - Auditing mechanisms inherently need to support data dynamics

Secure Storage Auditing (Cont.)

- Enable public auditing for unified risk evaluation
  - Introduce a third-party auditor (**or miners in the Blockchain network**) assists in data auditing
  - Public auditing should not affect **owner's data privacy (optional)**
- Handle multiple auditing tasks simultaneously (batch auditing)
  - The individual auditing of each data file can be tedious and inefficient
  - Batch auditing improves efficiency and saves computation overhead

One Example: Merkle Tree for Data Auditing

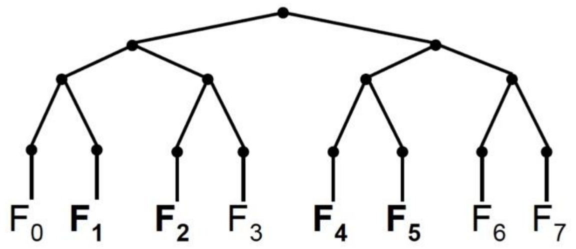
- **Challenge** blocks: C, E, F
- **Proof**: D, I, L, M Root
- **Verify** with stored root value



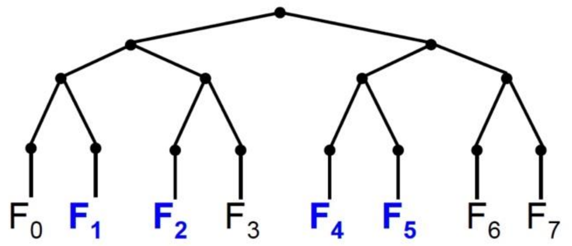
Data Auditing in Permacoin (1)

- Merkle-tree based
- The Merkle root **digest** of the file

  F is publicly accessible
  - Each leaf (e.g., F_0) is a segment of the file

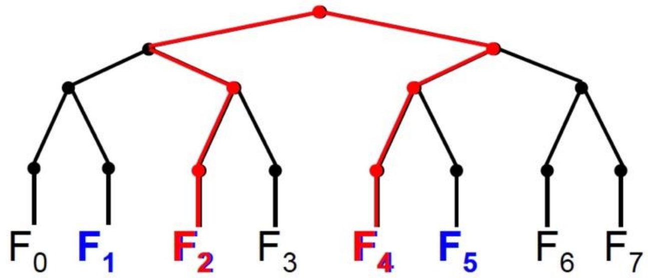


- Each miner (with public signing key **PK**) is mapped to a random subset of segments to store
  - Basing on the hash of **PK**
  - E.g., **PK** here maps to four segments
    - $F_{1,}$ $F_{2,}$ $F_{4,}$ F_5



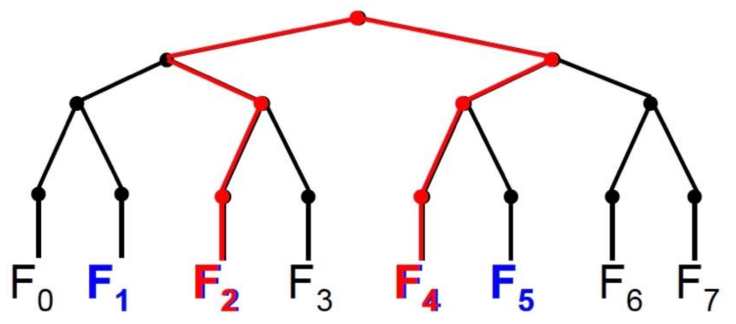
Data Auditing in Permacoin (2)

- Non-interactive challenge generation
  - Given the epoch-dependent puzzle **puz**
    - **puz** = v || B_l || $MR(x)$ || T, where v is the software version number, B_l is the previous mined block header, $MR(x)$ is the merkle-tree root over new transactions x, and T is the timestamp
  - The random challenge indices are selected
    a. h1 := H (**puz** || **PK** || s), where s is a random string chosen by the miner **PK**
    b. **h1 selects k segments from subset**
       – E.g., F_2, F_4 are selected



Data Auditing in Permacoin (3)

- The **ticket** for the puzzle **puz** is
  - **ticket** := (**PK**, s, {$F_{2,}\ \pi_2$}, {$F_{4,}\ \pi_4$})
    where π_i is the Merkle proof for segment F_i
- **Last step:**
  - Compute h2 := H **(puz || ticket)**
  - **Winner if** h2 < **TARGET**



Potential Benefits

- Reducing Bitcoin’s “honest” cost
- One example: UTXO (**unspent output from bitcoin transactions**) database
  - Miners in Bitcoin validate every transaction
  - Validation requires the UTXO database (GBs)
  - However, currently maintaining the UTXO database doesn’t pay
- **Idea: use Permacoin to reward UTXO storage**

Besides augmenting the storage of a public dataset, it seems that we can do more …

REM: Resource Efficient Mining for Blockchains

Fan Zhang, Ittay Eyal, Robert Escriva,
Ari Juels, Robbert van Renesse

CORNELL NYCTECH

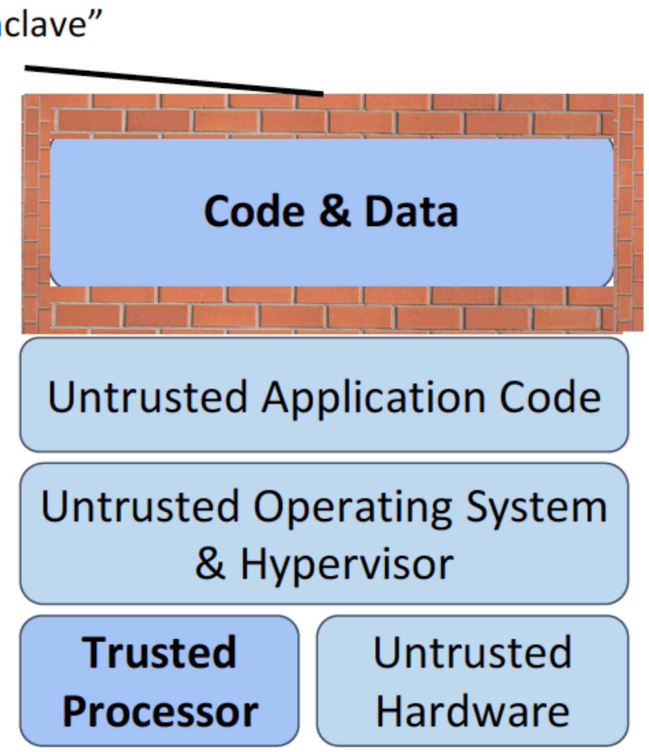
IC3 The Initiative For CryptoCurrencies & Contracts

Vancouver, Canada

13 September 2017 USENIX Security 2017

Software Guard eXtension (SGX)

Integrity

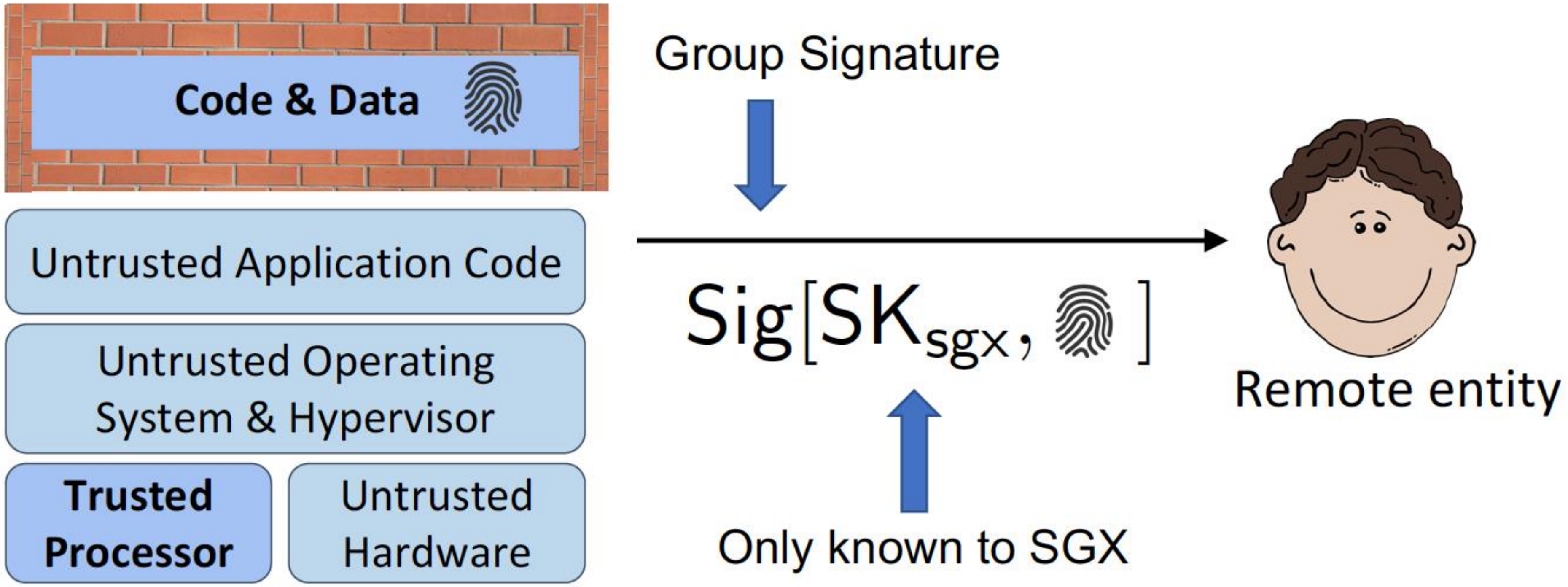


Other software and even OS cannot tamper with control flow.

Confidentiality

Other sofware and even OS can learn nothing about the interal state*.

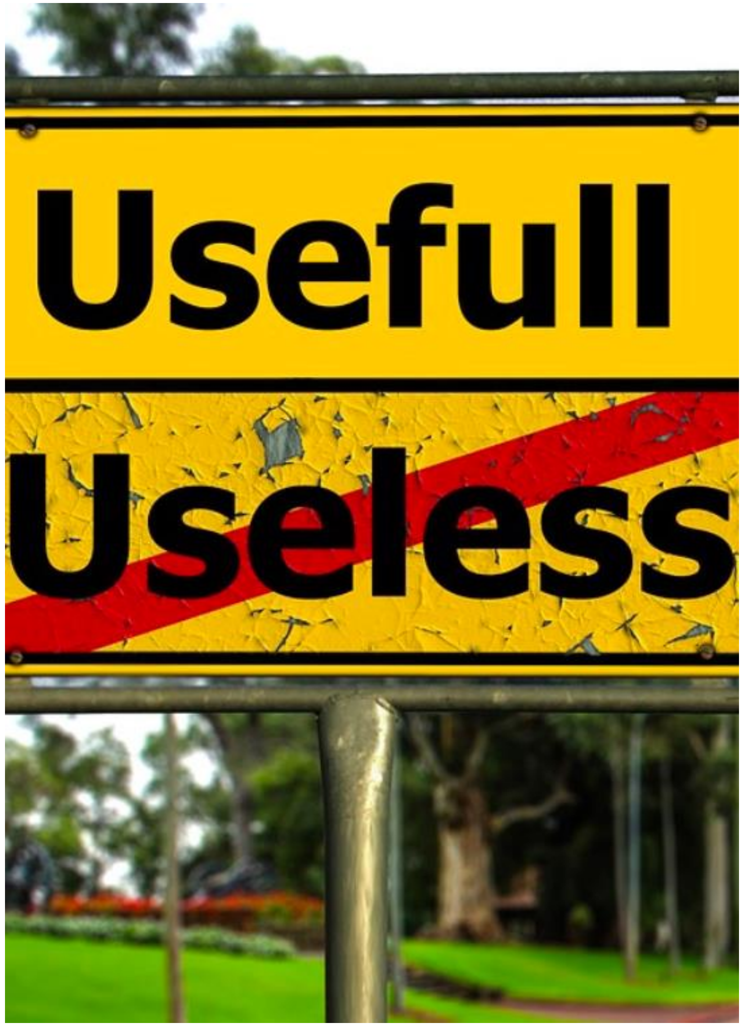
SGX: remote attestation



SGX-backed Blockchain: A New Security Model

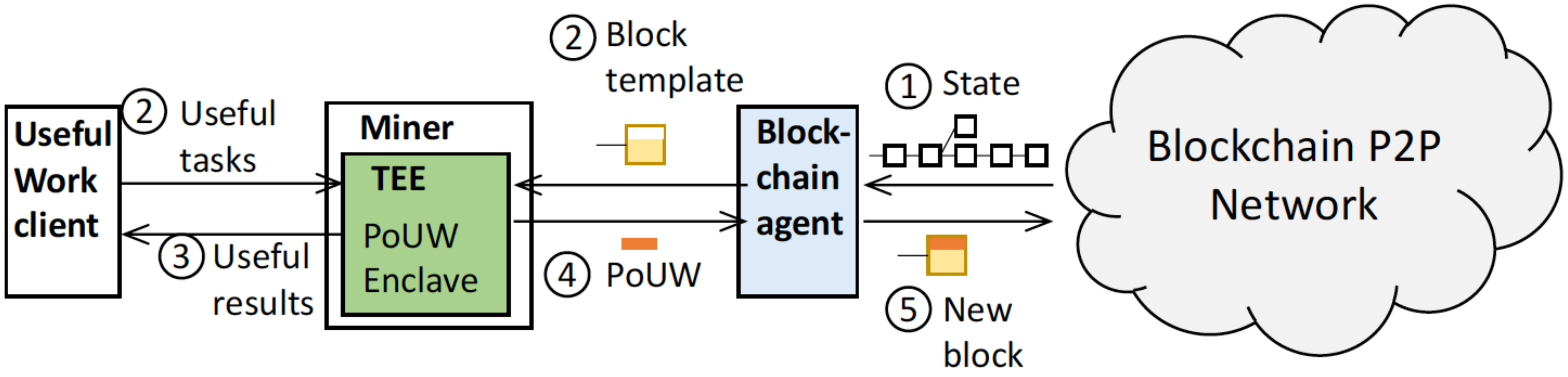
- Permission-less
  - Anyone with SGX equipped can join
- Partially decentralized
  - Assume SGX works as advertised
  - Intel correctly manages the group signature

Proof of Useful Work (PoUW)



- Replace the hash calculation in PoW with "useful" mining work
- Each unit of useful work grants a Bernoulli test
- Similar exponential block time

REM: Architecture Overview



- Blockchain agent:
  - Collect transactions and generate a block template (a block without PoUW)
  - Publish a mined block to the P2P network and receive reward
- Useful work client:
  - Generate PoUW tasks
- Miners
  - Return useful task results and **(possibly)** provide PoUW

Like Bitcoin, exponential block time is ensured, e.g., 10mins per PoUW among all miners. How?

Useful Work Metering

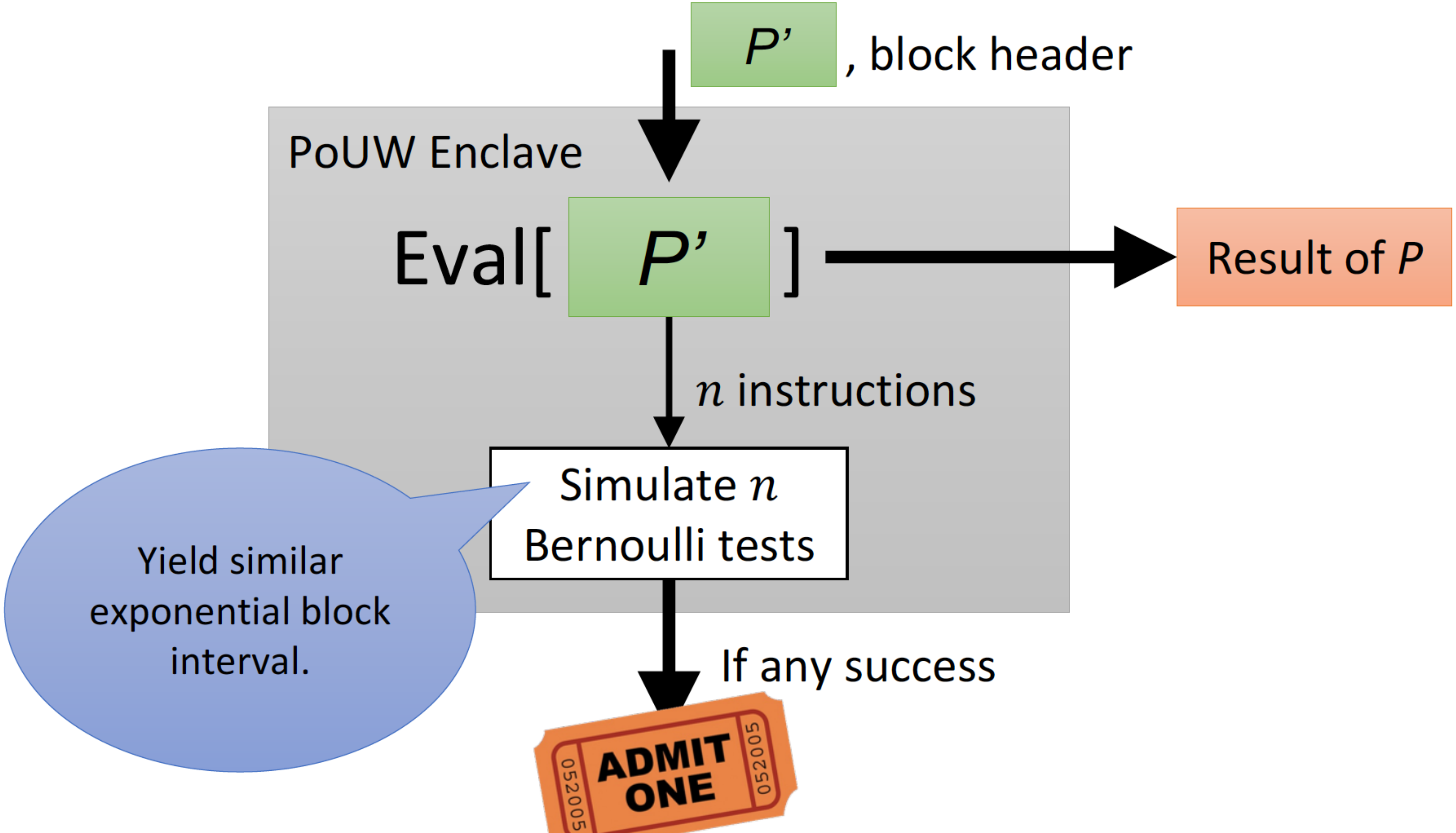
- **Key question 1**: how to meter the effort a miner has conducted?

- Proof of Work: perform two SHA256 hash operations
- REM: perform a CPU instruction that is defined in an PoUW task

Useful Work Metering (cont.)

- **Key question 2**: how to determine whether an effort is successful, i.e., resulting in a new block?
- Proof of Work: check if the computed hash value is *smaller* than a target value
  - Overall mining effort is measured in terms of the <u>number of executed hashes</u>
- REM: each performed CPU instruction wins a chance to conduct a **Bernoulli trial**
  - Mining times are distributed similar to PoW
  - Overall mining effort is measured in terms of the <u>number of executed useful-work instructions</u>

REM Miners



P', block header
PoUW Enclave
Eval[P']
Result of P
n instructions
Simulate n Bernoulli tests
Yield similar exponential block interval.
If any success
ADMIT ONE
052005
052005

Challenge: Replace PoW with Alternate Resource Lottery

- Other physical resources, with different properties?
  - Disk space
  - Useful computation/storage
  - Etc …

- What about the coin itself?
  - “Virtual resource mining” --> <u>Proof of Stake (PoS)</u>

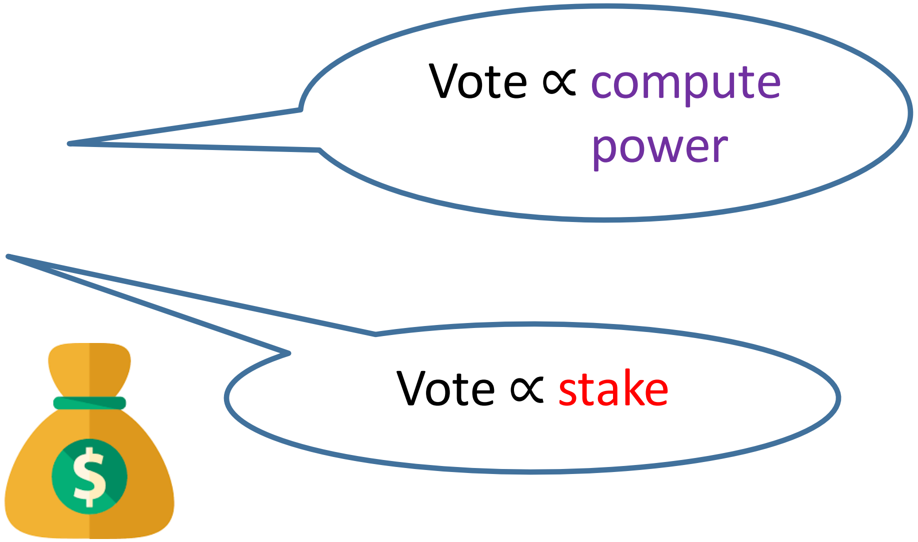
Challenge: Replace PoW with Alternate Resource Lottery

- Other physical resources, with different properties?
  - Disk space
  - Useful computation/storage
  - Etc …
- What about the coin itself?
  - “Virtual resource mining” --> Proof of Stake (PoS)

PoS: Stake-based Lottery

- Blockchain tracks ownership of coins among parties
- Idea: participants elected <u>proportionally to stake</u>

  $\Rightarrow$ No need for physical resources
- Different ideologies
  - Proof-Of-<u>Work</u> (POW)
  - Proof-Of-<u>Stake</u> (POS)



Ideal PoS Consensus

- Execute in epochs
  - An epoch consists of several time steps
- In each epoch:
  1. Configure a committee of stakeholders
  2. Randomly elect a leader in every time step (based on their stakes)
  - In each time step:
    a. Leader selects the longest available chain
    b. Leader generates a new block and broadcasts it to the network
- Reconfigure the committee at the end of each epoch
  - To reflect stake changes (due to money transfers)

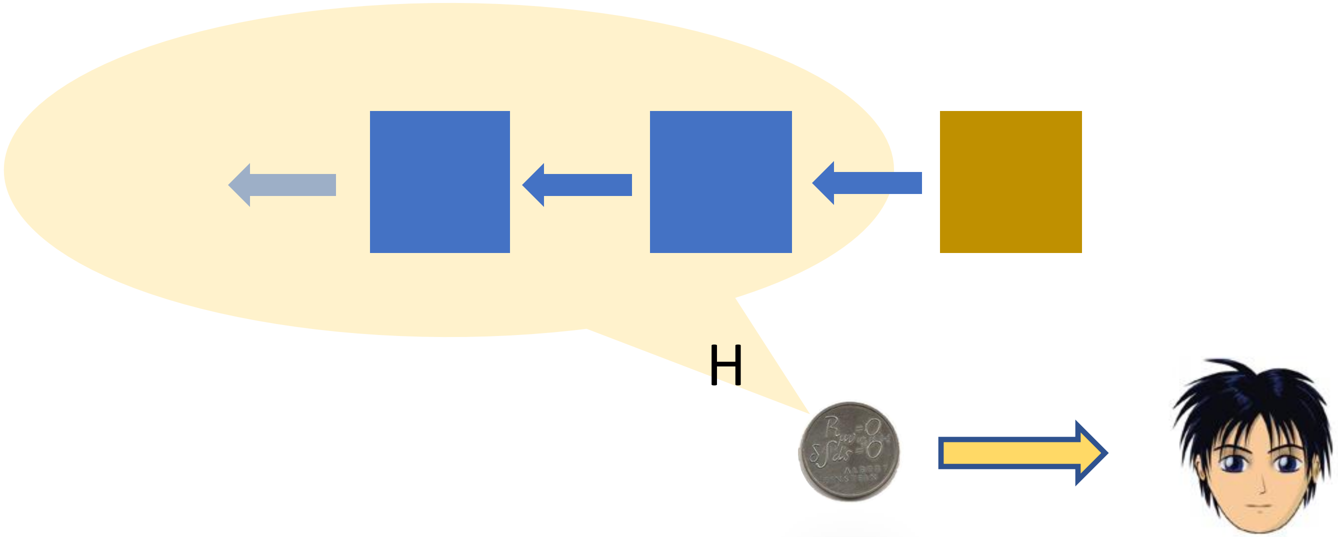
PoS: The Challenge

- Difficulty: Electing a coin (owned by a stakeholder) requires randomness
- If the adversary can bias the randomness, things get difficult

An Idea

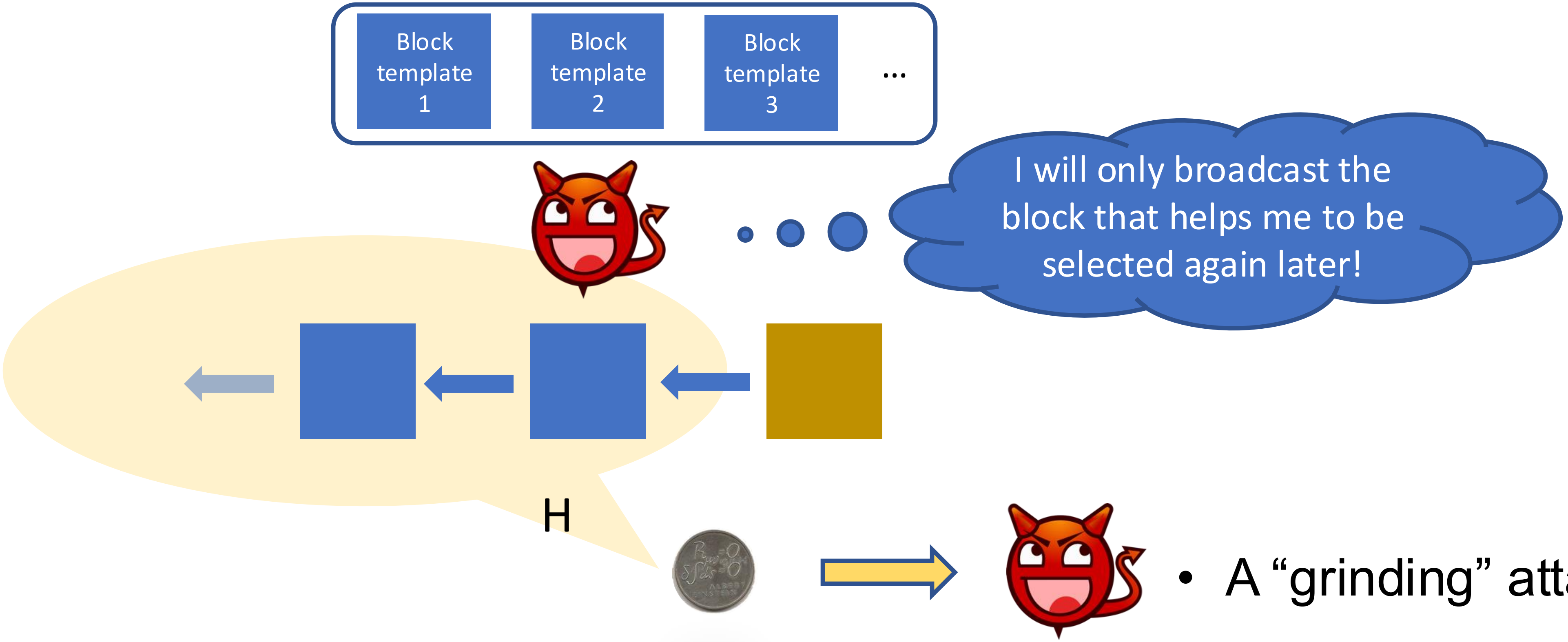
- Blockchain already contains effectively random data (e.g., block hashes)
- Obvious approach:

  Just hash the current blockchain!



An Idea … confounded!

- However, adversary can try to bias the randomness in their own favor



- A "grinding" attack

PoS Designs “in the wild”

- PoS blockchains have appeared “in the wild”:
  - NXT
  - Peercoin
  - DPoS (BitShares, Steem, EOS)
  - Casper (Ethereum)
  - Etc…

PoS Designs with Rigorous Guarantees

- Eventual (Nakamoto-style) Consensus:
  - Ouroboros
  - Ouroboros Praos
  - Ouroboros Genesis
  - Snow White

- Block-wise Byzantine Agreement:
  - Algorand

Here we will only elaborate two PoS designs with rigorous security guarantees:

Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol, Aggelos Kiayias et al., *in Proc. of CRYPTO*, 2017

&

Algorand: Scaling Byzantine Agreements for Cryptocurrencies, Yossi Gilad et al., *in Proc. of SOSP,* 2017

Ouroboros: 10000ft View

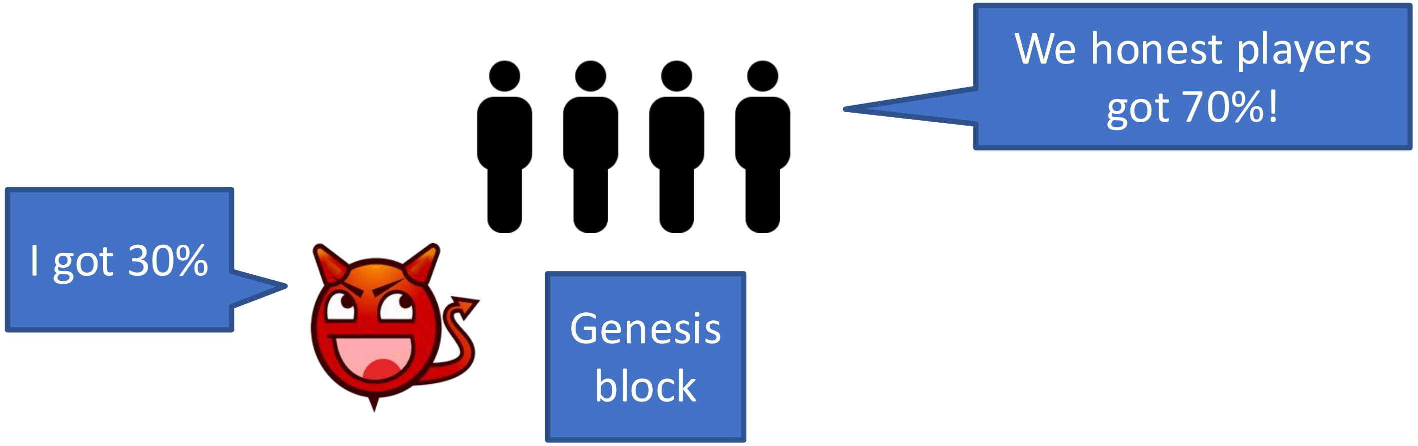
- Assumes **synchronous** time & communication
- Guarantees
  - <u>persistence</u>: stable transactions immutable
  - <u>liveness</u>: new transactions included eventually

IF

  - Adversary has minority stake throughout
  - Adversary subject to corruption delay
    - Corrupt an honest stakeholder after some delays

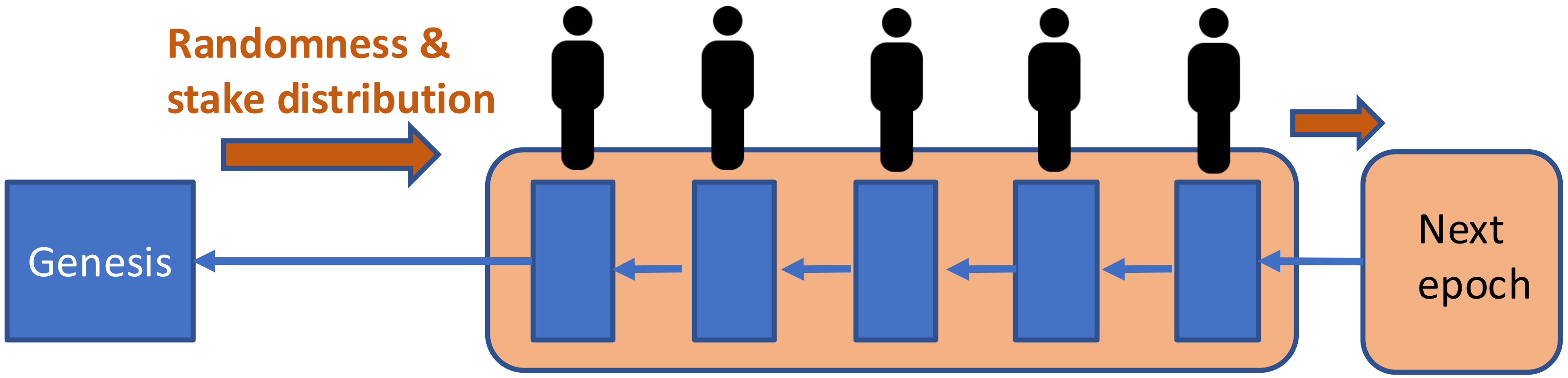
Ouroboros: Overview (1)

- Assuming root-of-trust
  - Initial stake distribution of stakeholders is hardcoded in the genesis block
  - Trusted to achieve <u>honest majority</u>



Ouroboros: Overview (2)

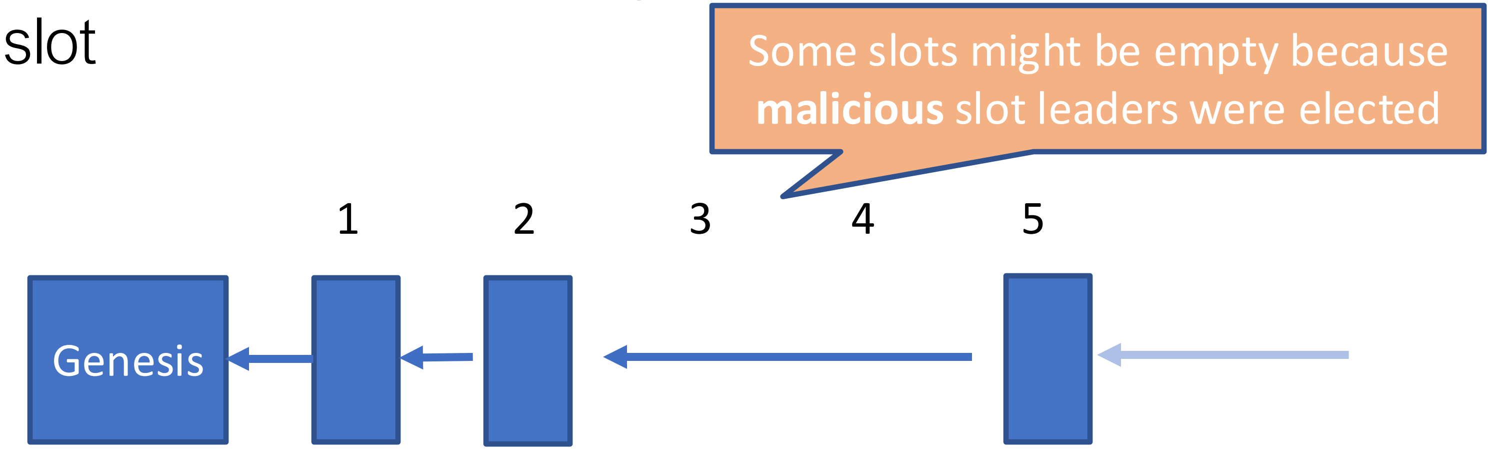
- Randomly elect leaders in proportional to their stake
  - Leveraging coin-tossing and verifiable secret-sharing to generate randomness for the next epoch (<u>will introduce later</u>)
  - **The generated randomness and stake distribution define R slot leaders in the next epoch**
    - Randomly **select coins** in the previous epoch, and elect the corresponding coin owners as slot leaders in the next epoch
      - – **More coins (money) you have, more chances to be elected!**



An epoch with R slots (e.g., R = 5)

Ouroboros: Overview (3)

- Blockchain validation:
  a. Starts with genesis block
  b. A sequence of blocks follow, associated with increasing slot numbers
  c. No conflicting transactions
  d. Each block signed by $\mathbf{L_i}$, the leader associated with that slot



Randomness Generation

- Recall: randomness is needed for next epoch's leader election
- Idea:
  - Use a secure <u>coin-tossing protocol</u> to generate it during the previous epoch
- Result:
  - If a majority of the leaders in the previous epoch are honest, we can get <u>clean (unbiased) randomness</u> for next epoch

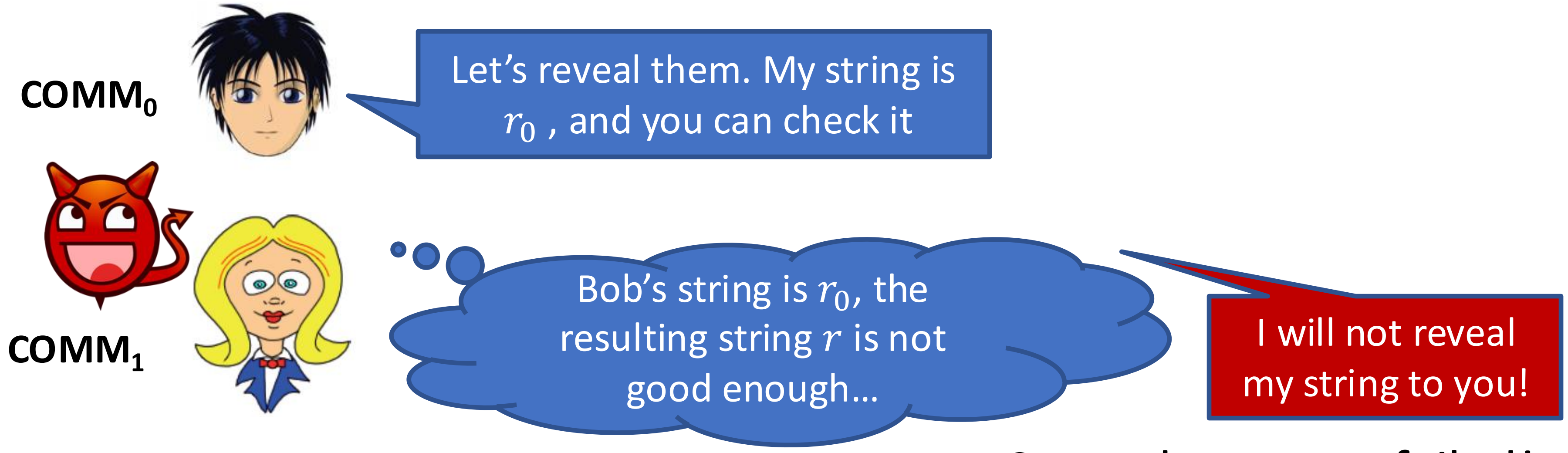
Coin-tossing Protocol

- Goal: generate an **unbiased** random string in a distributed setting
- **We start with strawman design ①:**
  - Simply outputs $r = \oplus_{i=0}^{n-1} r_i$
  - Here, we demonstrate the case that $n = 2$

What is the value of r now?

Second Attempt

- We can have each peer **commit to their chosen input** before seeing other inputs
  - <u>Commit-then-reveal approach</u>
  - Assume Bob has committed its string r_0 as COMM_0, and Alice has committed its string r_1 as COMM_1



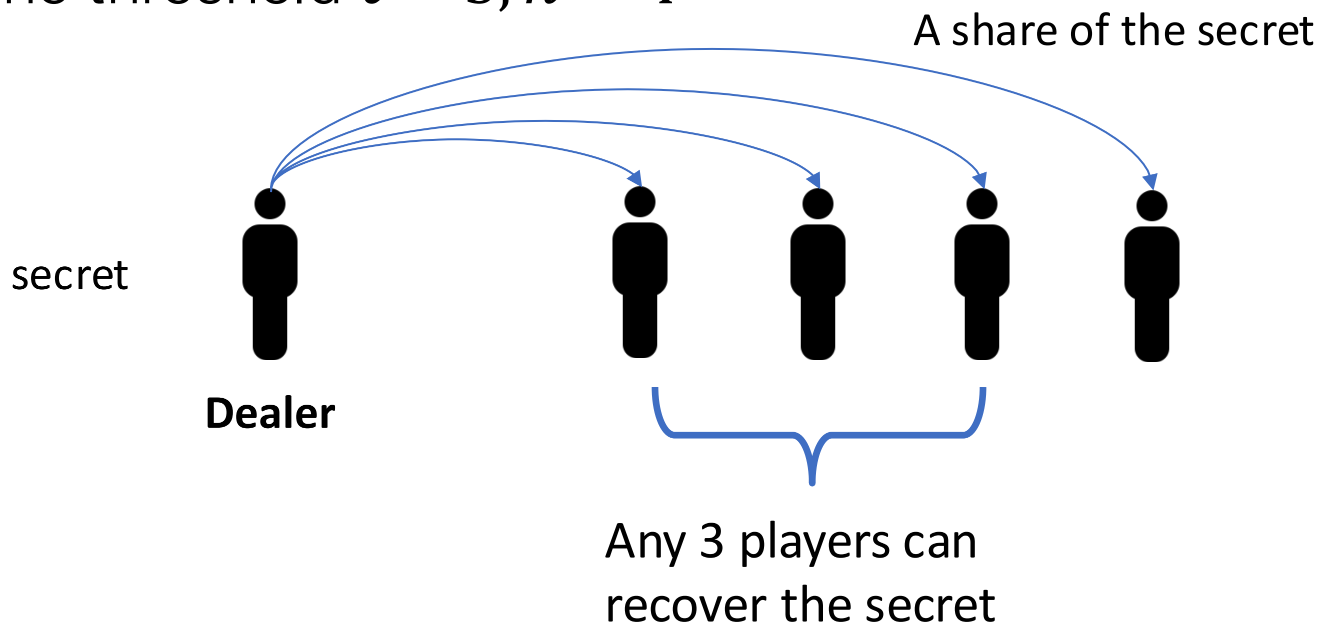
Second attempt, failed!

Third Attempt

- We wish to ensure that a **dishonest peer cannot force the protocol to abort** by refusing to participate
  - We need more than 2 players!
  - Leverage a (t, n)-secret sharing scheme
    - n is the number of players
- (t, n)-secret sharing
  - A secret is shared among n persons, and we only need t shares to recover it
  - E.g., Shamir's secret-sharing scheme
    - Can extend to achieve <u>public verifiable</u> to address bad shares that prevent correct recovery of the shared secret

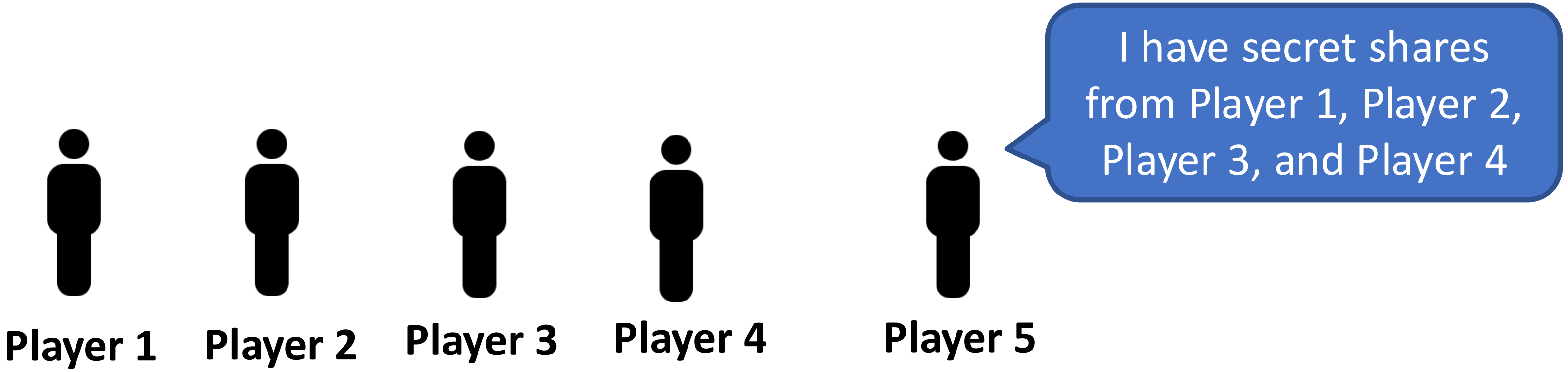
Simplified Demonstration

- Consider a $(3{,}4)$-secret sharing scheme for 5 players, with one player acted as the dealer (for sharing a secret)
  - The threshold $\mathrm{t} = 3, n = 4$



Simplified Demonstration (Cont.)

- Each player will act as the dealer once and share a secret to others
  - Thus in the case of 5 players, each player will eventually have 4 shares of secrets



Simplified Demonstration (Cont.)

- Reconstructing all secrets (i.e., 5 secrets in our previous example) and <u>XORing all secrets to generate the final random string</u>
- Analysis:
  - As long as the majority of players (e.g., 3) are honest, all secrets can be correctly recovered
    - Since we set $\mathrm{t} = 3$ as our threshold in the secret sharing scheme
  - A <u>random and unbiased</u> string is guaranteed!

Protocols for Augmenting Ouroboros

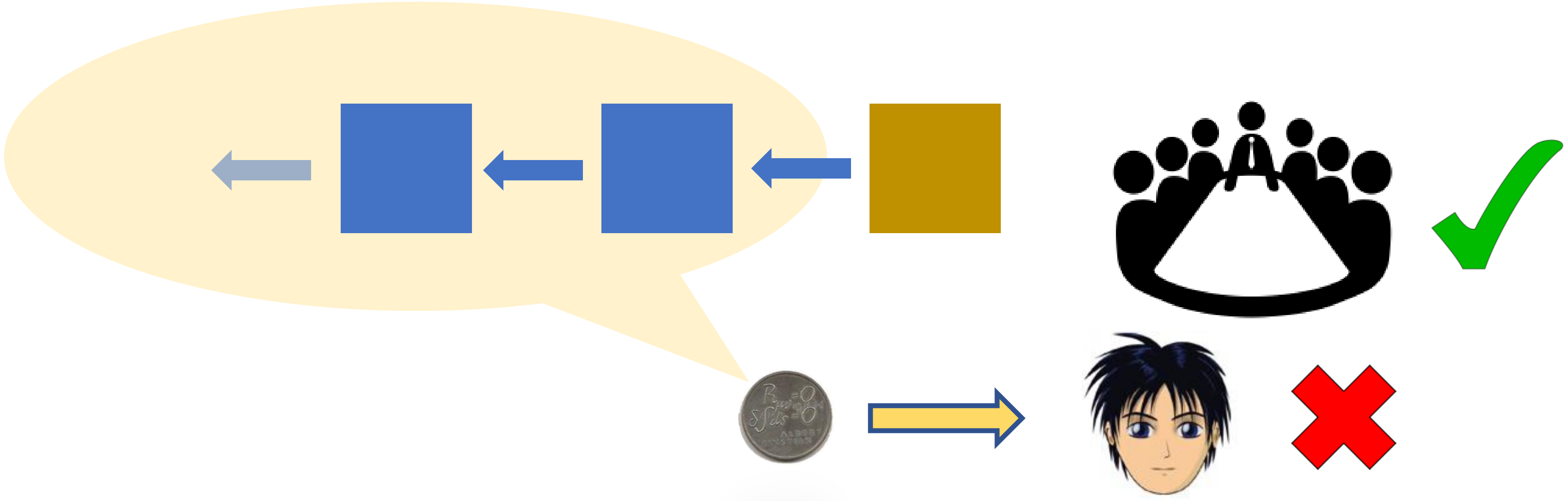
- **Ouroboros Praos** with improved assumptions:
  - In a **semi-synchronous communication** model
  - Despite fully adaptive corruptions
    - I.e., attacker now can corrupt honest player that **has been elected in previous epochs**
- **Ouroboros Genesis**:
  - Global UC formalization
  - Improved bootstrapping procedure
    - New chain selection rule (instead of longest chain only)
  - Dynamic availability
    - Achieves security with node join and leave

Algorand: Scaling Byzantine Agreements for Cryptocurrencies, Yossi Gilad et al., *in Proc. of SOSP,* 2017

(not to be included in final exam)

Algorand: The View from Mars

- Each new block in Ouroboros is generated by one stakeholder
- What if:
  - We replace that specific stakeholder with **a committee**? => Algorand



Algorand: Overview

- Ingredients:
  - **Committee formation:** Cryptographic Sortition
    - Verifiable random functions (VRF)
  - **Consensus:** new Byzantine Agreement protocol (BA*)
- Assumption:
  - Adversary can only control less than 1/3 of total money

Question 1: how can we securely elect a committee from the entire network?

=> Algorand uses a technique called
cryptographic sortition

Cryptographic Sortition

- Goal: Select a subset of users
- **Simple idea**: Get the users to interact and talk amongst themselves to elect a committee
  - Is it secure?

- However, the adversary could be listening to the election process
  - Might **target some users** before they can participate in the interactive protocol (denial of service attacks!)

Cryptographic Sortition in Algorand

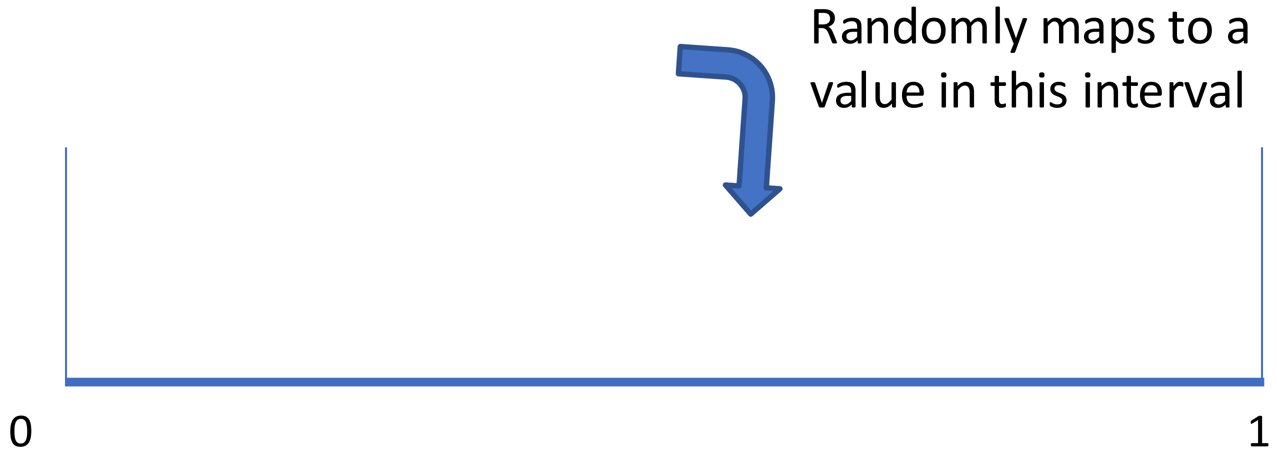
- Goal: <u>Local and non-interactive</u> way to select a subset of users (*forming a committee*)
  - Implemented using Verifiable Random Functions (VRF)

<PK, SK> key pair

Hash is **pseudo-random** and **distributed uniformly between [0, $2^{hashlen}$-1]**

Result: A Verifiable Hash Value

- Given <Hash, **PK**, proof>, anyone can verify the validity of Hash
- Since Hash is uniformly distributed between $[0, 2^{hashlen}-1]$
  - $T = Hash/2^{hashlen} \in [0,1)$



Since Hash is verifiable, T is verifiable too!

Next important property:

=> Making the chance that each stakeholder is selected proportional to its stake in the blockchain!

Background: Binominal Distribution

- $B\ (k; w, p)$: Probability of getting k successes in w trials, where probability of success in each trial is p

$$B\ (k; w, p) = \Pr\ (X = k) = \binom{w}{k}\ p^k (1 - p)^{w-k}$$

for $k\ =\ 0, 1, 2, \ldots, w,$ where

$$\binom{w}{k} = \frac{w!}{k!(w-k)!}$$

is the **binomial coefficient**

- One useful property:

$$\sum_{k=0}^{w} B\ (k; w, p) = 1$$

Notations in Algorand

- W: current total amount of money units in the system
- τ: threshold, denoting expected number of money units selected
- p: τ/W
- w: stake/money of a user

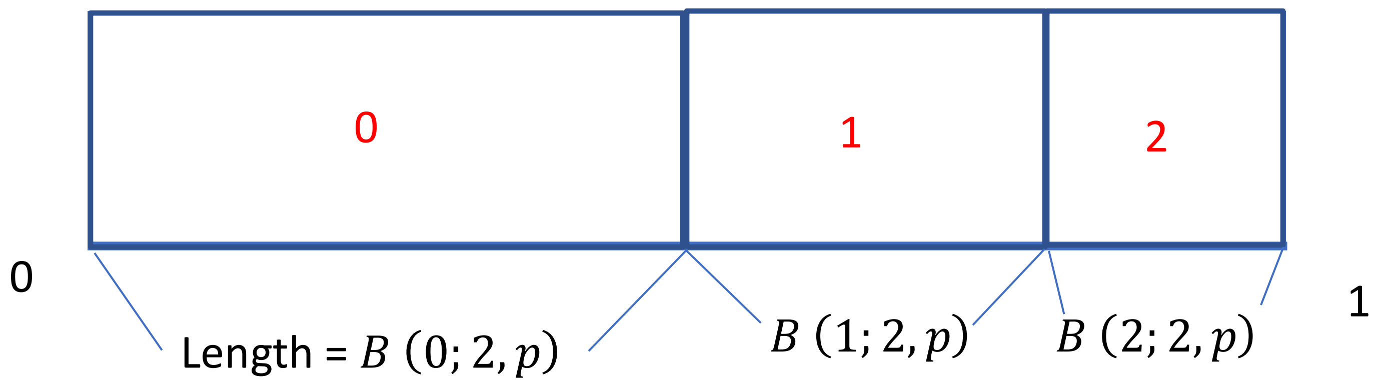
$B\ (k; w, p)$ Meaning in Algorand

- If p represents the probability that a coin is selected
  - $\Rightarrow B\ (k; w, p)$ defines the probability that exactly k coins are selected from the stakeholder's w coins
  - $\Rightarrow$ For example, Bob has 2 coins and the probability p is pre-defined
    - $B\ (0; 2, p)$ is the probability that no coin from Bob is selected
    - $B\ (1; 2, p)$ is the probability that one coin from Bob is selected
    - $B\ (2; 2, p)$ is the probability that two coins from Bob are selected

$$\sum_{k=0}^{2} B\ (k; 2, p) = 1$$

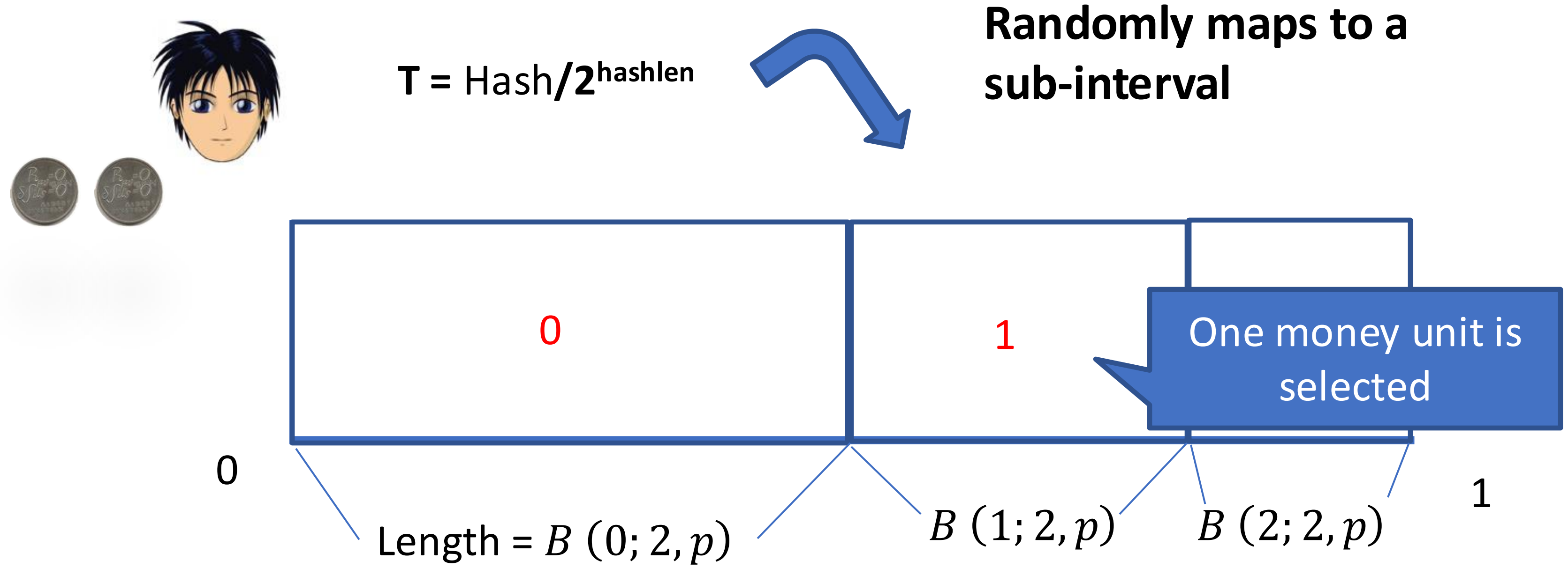
Achieving Stake-based Selection (1)

- Idea: for each stakeholder, we divide the interval $[0,1)$ into $w + 1$ sub-intervals, where w is the # of money units the stakeholder owns
  - Requirement: each sub-interval j still reflects the probability of $B\ (j; w, p)$
- With 2 coins (money uints)



Achieving Stake-based Selection (2)

- We have a verifiable value **T** = Hash/2^{hashlen} that uniformly distributed in interval $[0,1)$



Achieving Stake-based Selection (3)

- Observation: more money units might be selected at a **richer** stakeholder
- τ can be adjusted to control the # of (totally) selected money units
  - Since $p := \tau / W$
- Definition: IF <u>more than one money unit is selected</u>, the corresponding stakeholder is selected in the committee

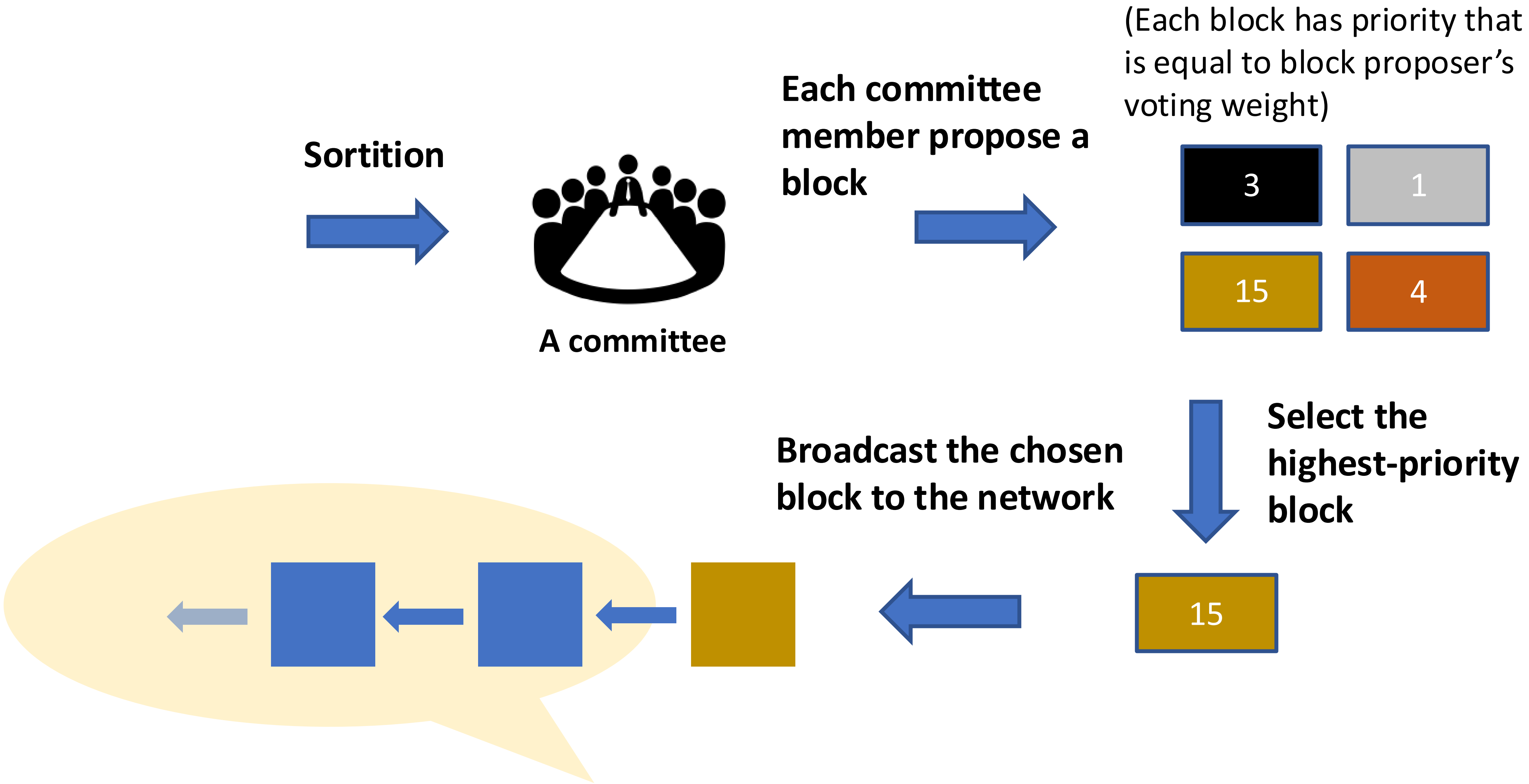
Result: A Committee with Different Voting Weights

Identity	# of selected money units
Alice	4
Bob	1
Carol	15
John	3
Peter	9

Vote weight

Next question: How to generate and commit new blocks with this committee?

Strawman Design



Actually, Algorand Consensus is More Complicated

- In each step, a different committee is elected through cryptographic sortition
  - Address attacker who can corrupt some honest committee members after the election
- Basic strategy: weighted majority voting:
  - Each committee member will broadcast their vote for their block
    - Vote for highest priority block
    - All users can see this message
  - Users that receive more than a threshold of votes for a block will hold onto that block

Simplified Demonstration (1)

(threshold=10)

Sortition

Committee 1

Identity	Vote weight
Alice	4
John	1
Carol	15

Broadcast proposed blocks

4

1

15

Three valid blocks have been proposed.

I will hold onto Carol's block since it has passed the threshold!

User bob
(hasn't been selected in committee 1)

Simplified Demonstration (2)

(threshold=10)

Sortition

Committee 2

Bob

I have been selected in committee 2!

I will broadcast the highest-priority block with my votes attached!

9

Broadcast

Identity	Vote weight
John	3
Peter	4
Bob	9

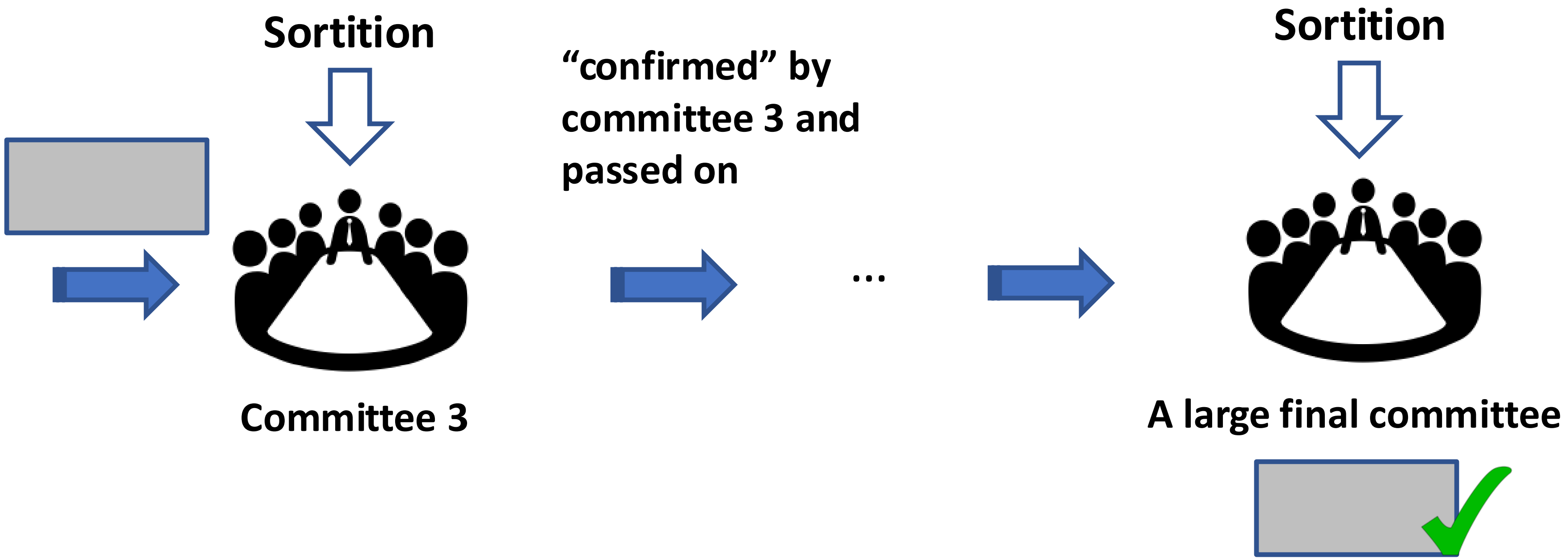
9 + 3 =12

Can pass the threshold!

“Accepts” it to be processed in the next step

Simplified Demonstration (3)

- So on and so forth …
- Finally, being confirmed by a final committee and committed to the network

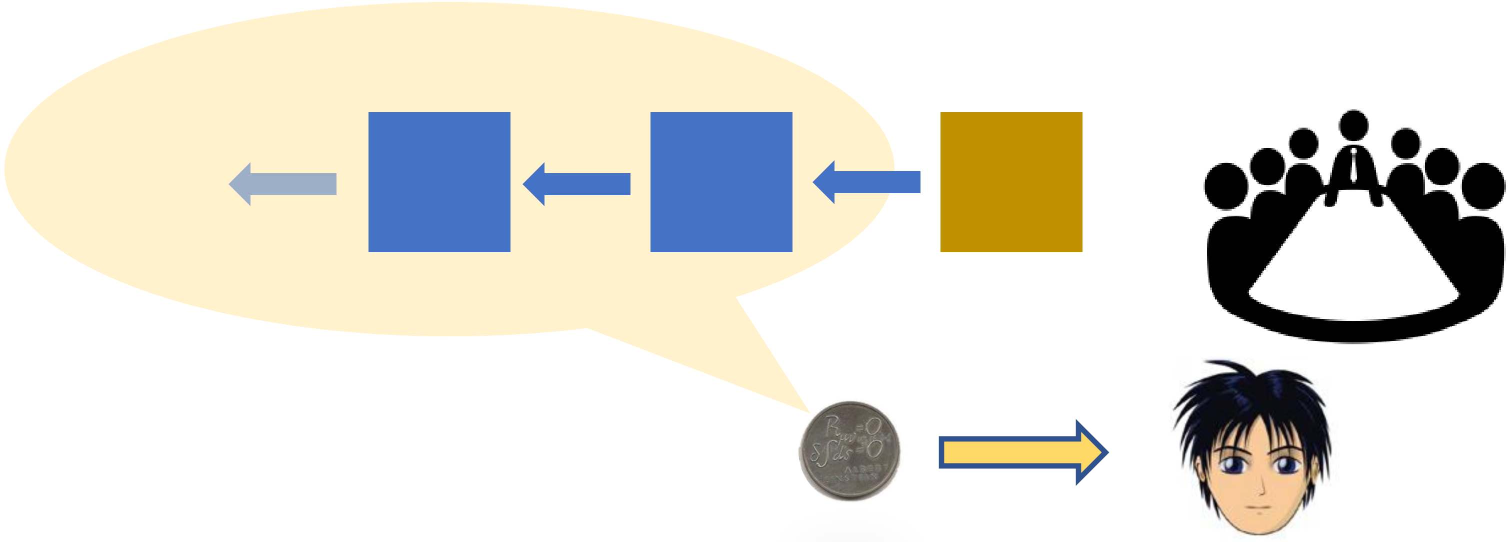


Performance Highlights

- BA* protocol in expectation terminates
  - In 4 steps
    - in the case where the block proposer with highest vote is "honest" and the network is strongly synchronous
  - Or in 13 steps
    - in "disaster" case
    - Might require additional algorithm for recovery
- ~1 min to confirm transactions vs an hour (6 blocks) in Bitcoin

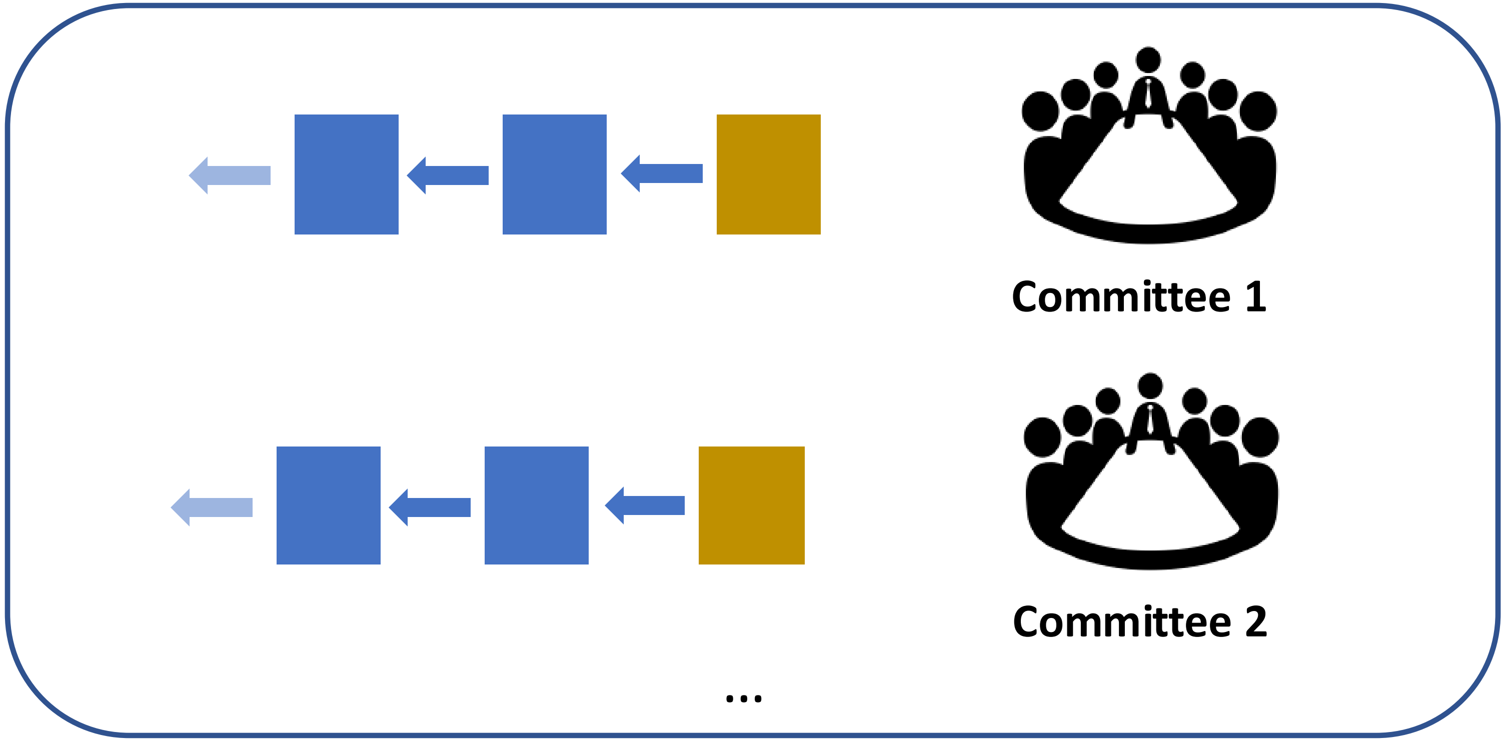
Wrapping Up

- Ouroboros uses one stakeholder for handling block proposal
- Algorand (in each step) maintains one committee for handling block proposal



Can we go further?
Say, with multiple committees working in parallel?

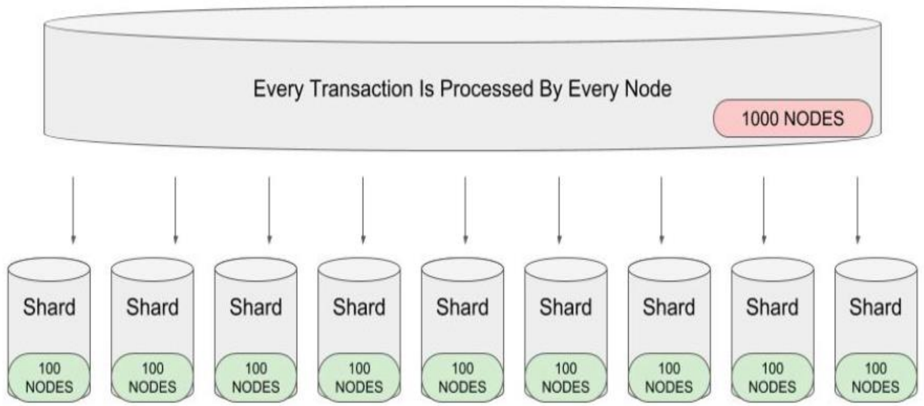
Sharding: A Glimpse

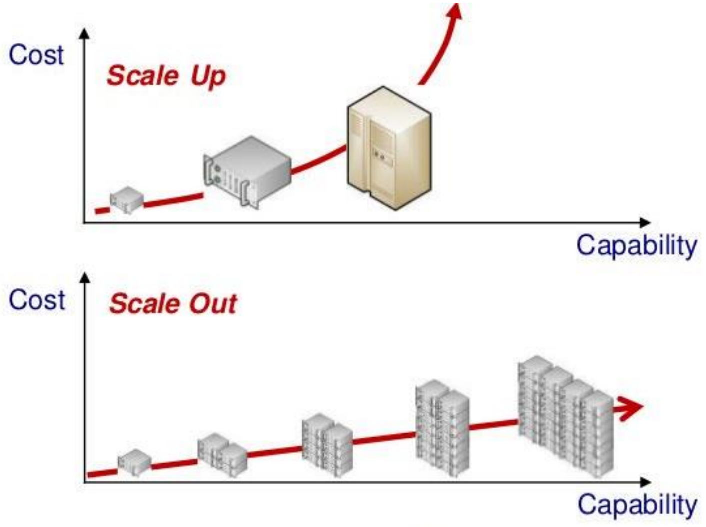


A blockchain system that scales with the number of players participated in the system

Blockchain Sharding Protocols

- Motivation: single committee improves performances, but it does not **scale out**
- Idea:
  - **splitting transactions** among multiple committees (shards) which process these transactions <u>in parallel</u>



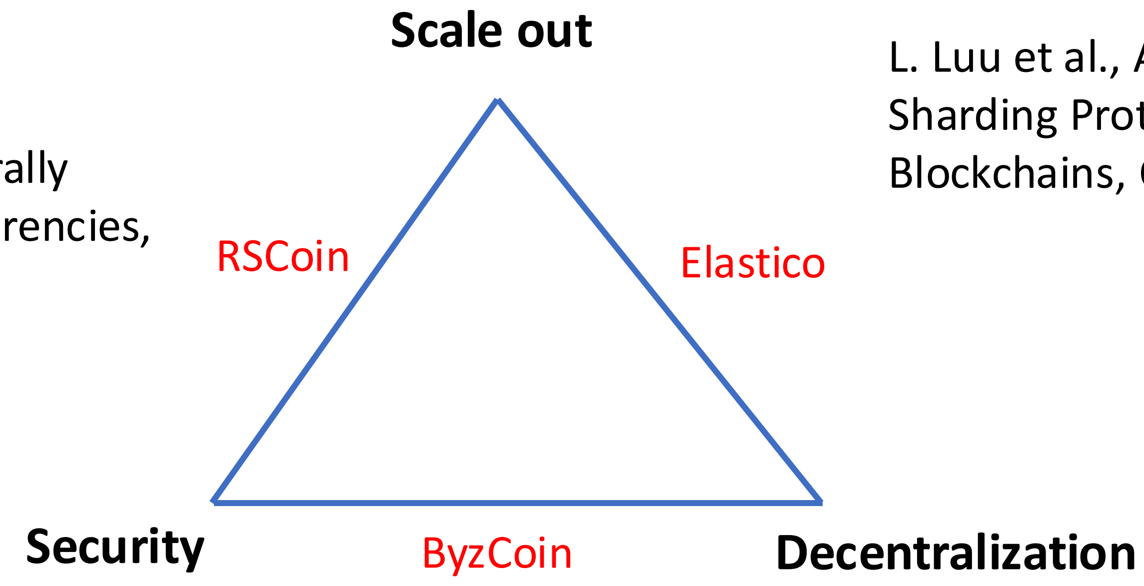


But Scaling out Blockchains is Not Easy

Figure credit to Omniledger

The Scalability Trilemma (Vitalik)

G. Danezis and S. Meiklejohn, Centrally Banked Cryptocurrencies, NDSS 2016



L. Luu et al., A Secure Sharding Protocol for Open Blockchains, CCS 2016

E. Kokoris Kogias et al., Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing, USENIX Security 2016

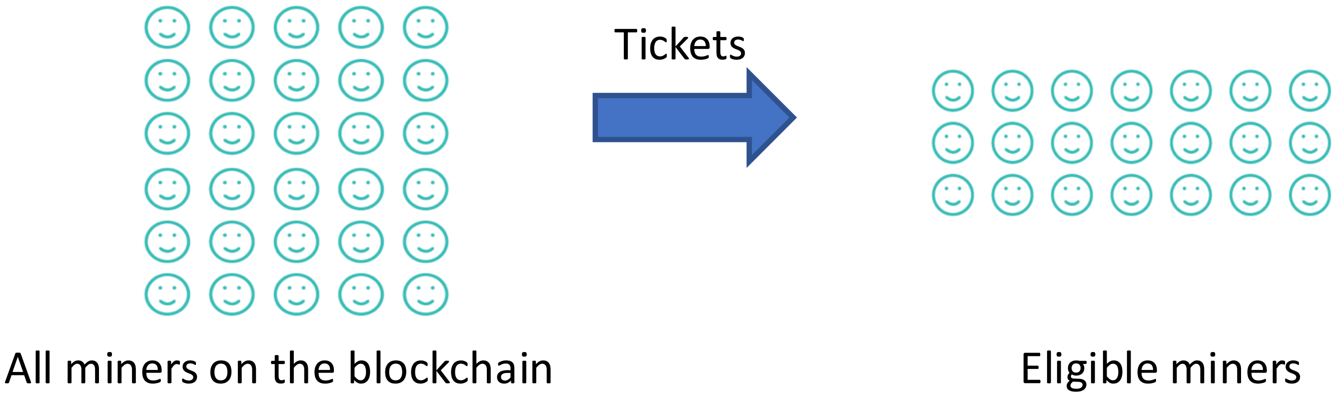
Figure credit to Omniledger

An Overview of Existing Blockchain Sharding Methods

- Main components:
  - Miner assignment + intra-shard consensus + security refinements
- UTXO-based mode:
  - Elastico CCS'16, Omniledger S&P'18, Rapidchain CCS'18, Monoxide NSDI'19
  - Projects: Zilliqa, Blockclique, …
- Account-based mode:
  - Chainspace NDSS'18, DANG et al. SIGMOD' 19

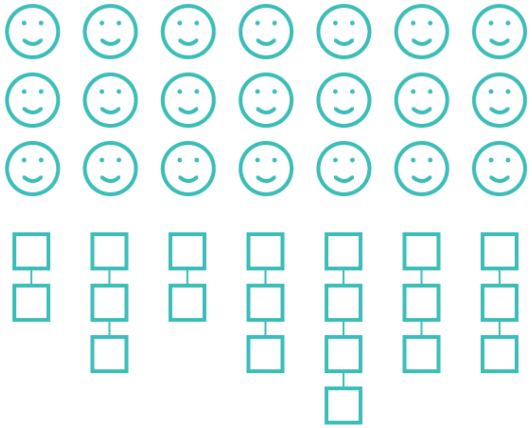
Miner Assignment

- "Buy ticket" before participating
  - Ticket here can be a Proof-of-work (Rapidchain, Omniledger) or a stake-based VRF proof
  - Mainly to address *Sybil attacks*



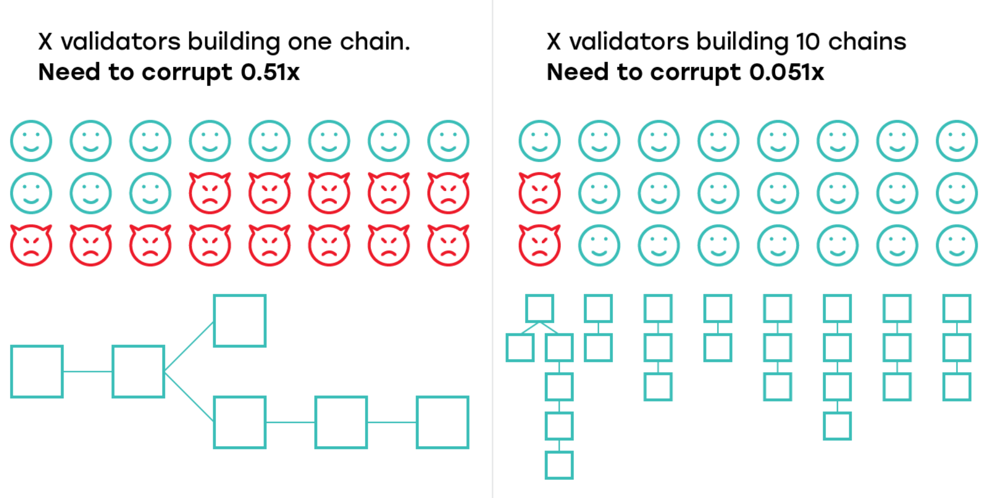
Shard Configurations

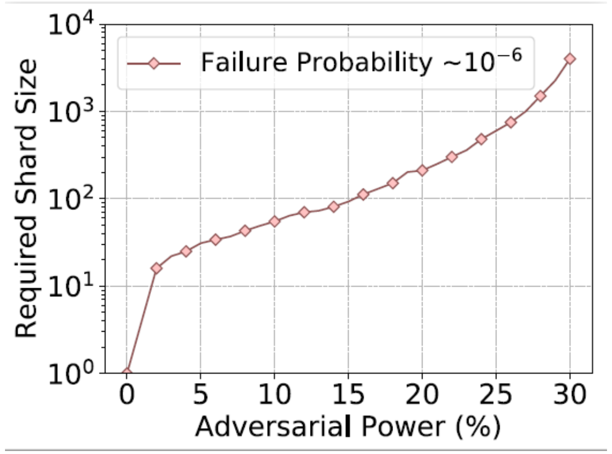
- The number of shards is defined in advance
- Each shard has a shard_ID (can be a number)



Which shard a miner should go?

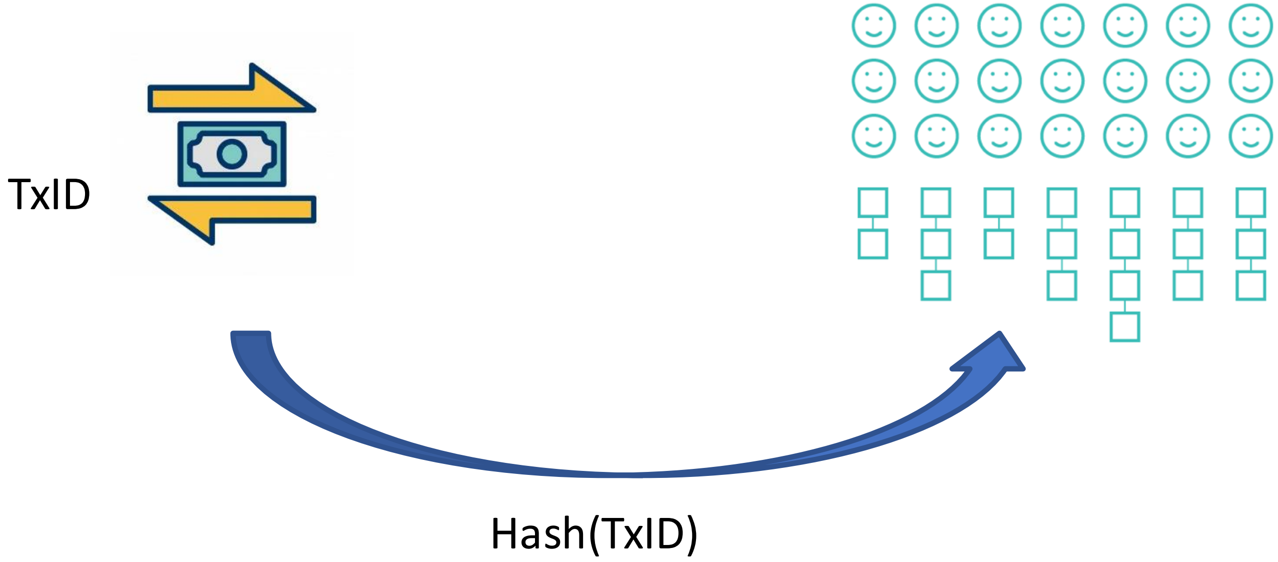
- Let miners choose?
  - All malicious miners can choose the same chain
- Randomly assign miners?
  - Preserve security for adequately large shard size





Transaction Assignment

- Create **disjoint transaction sets**
  - Hash the ID of the incoming transaction
  - See which shard it maps to and forward it to that corresponding shard



Main Confronted Challenges

- Intra-shard consensus
  - How to reach consensus in each shard
- Cross-chain verification
  - Inputs/outputs of a transaction might be maintained in different sharded blockchains

Intra-shard Consensus

- Random miner assignment guarantees each shard to have an upper of malicious miners
  - 1/3 in Omniledger, Chainspace and Elastico
  - 1/2 in Rapidchain
- As the miners in each shard are identified, all existing standard consensus protocols can be used
  - PBFT is used in Omniledger, Chainspace and Elastico
  - An enhanced consensus algorithm (with ½ resilient) is used in Rapidchain

Final Consensus?

- In ***Elastico***, at each epoch, the blocks generated by shards will be checked by a final committee
  - *For making each shard small (100 miners)*
  - Only if the final committee confirms the blocks, they become final at the global state (slower)

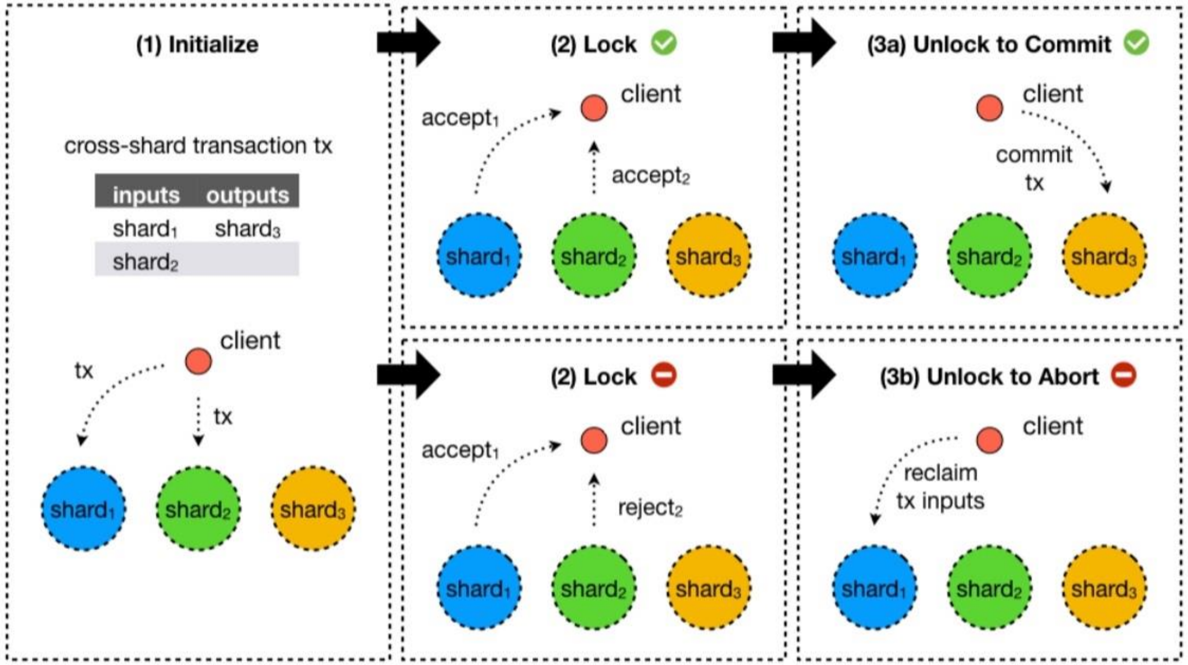
Subsequent works (e.g., Omniledger and Rapidchain) remove the final committee by <u>increasing the size of shards</u>

Cross-shard transaction

- The following example is courtesy of Andrew Miller:
  - A user wants to purchase a train ticket and reserve a hotel
  - The operation should be atomic: either both reservation succeed or neither do
- A (in shard_1) , B (in shard_2) -> C (in shard_3)
- Both A, B should be accepted before C happens

Cross-shard transaction

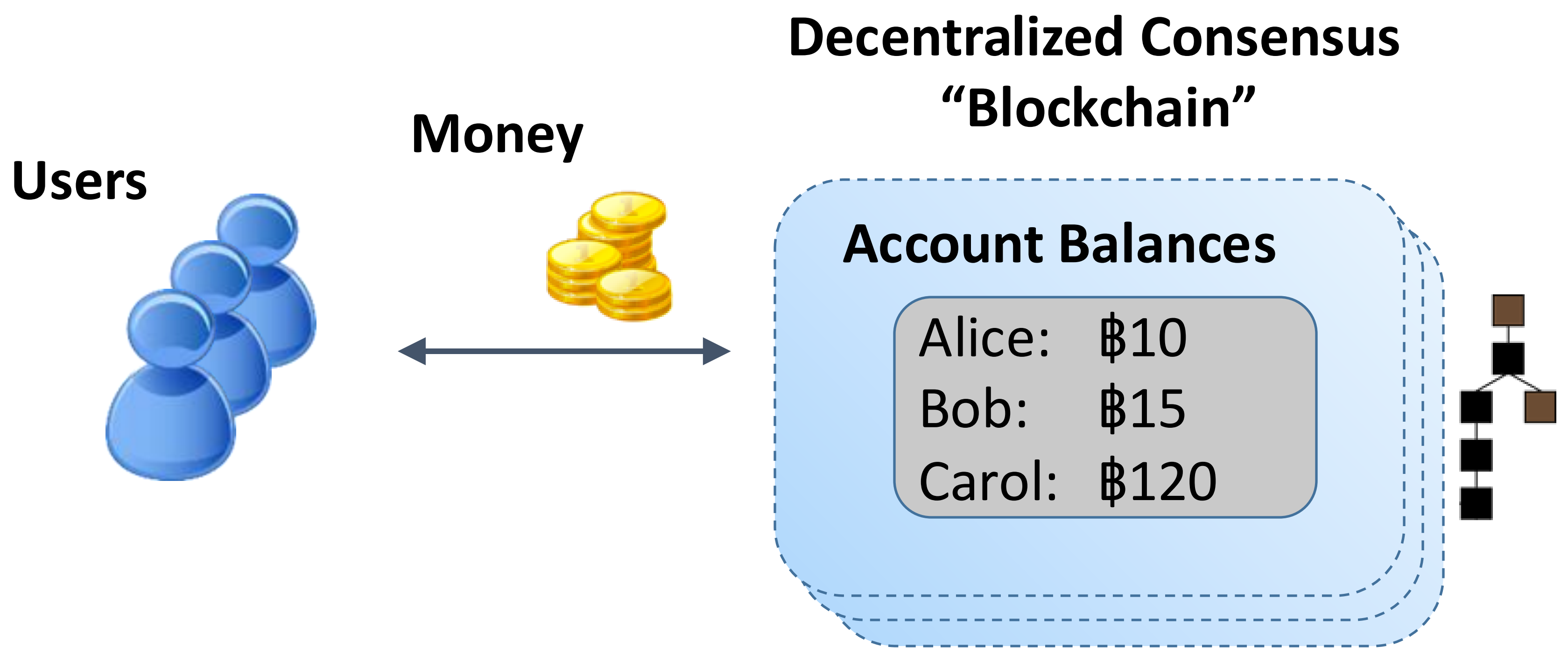
- If the two objects are situated in two different shards, cross-shard operation should be involved
  - **Atomic commit**: "lock-and-free", assuming synchronous communication



Challenge: what if the communication between two shards is ***asynchronous***; how to ***reduce the overhead***

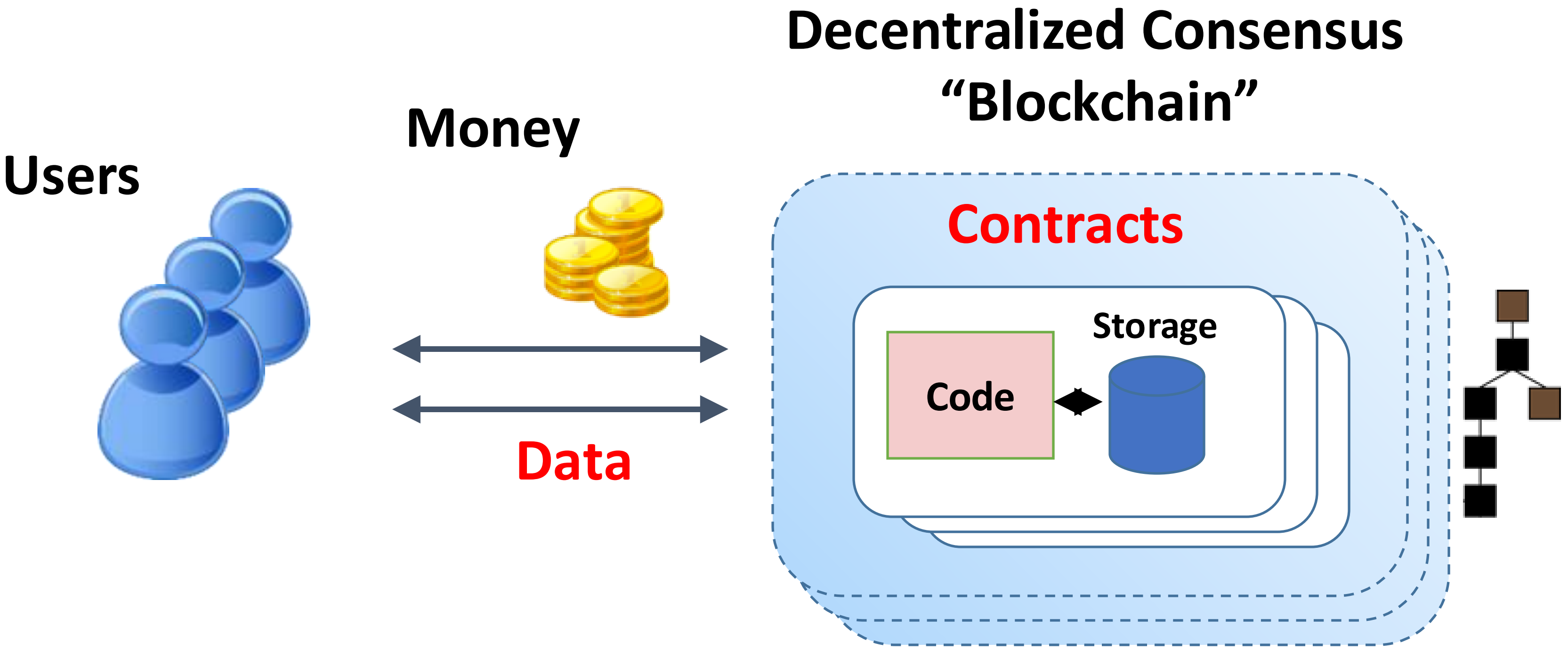
Recall: Bitcoin

- Just one application on top of a blockchain: token exchange



Smart Contracts

- User-defined programs running on top of a blockchain



What's a (decentralized) smart contract?

- Executable object on blockchain
- Scripted in Turing-complete language
  - Bitcoin has highly restricted language
- Code defines contract, e.g.,
  - Financial instrument
    - If GOOG rises to $1,000 by 30 June 2015, assign 10 shares from Alice to Bob and pay Alice $10,000
  - Szabo (1997): Smart contract reassigns physical access to your car from you to your bank if you don't make a payment
- Autonomous: *Enforced by network*

Can we build a blockchain that natively support smart contracts?

=>Ethereum

About Ethereum

- Founded by *Vitalik Buterin*, Gavin Wood and Jeffrey Wilcke in 2014
- Core: Ethereum Virtual Machine (“EVM”)
- “general purpose computation” programming language
- Every node of the network runs the EVM and executes the **same instructions on blockchain.**
- Sometimes described evocatively as a **“world computer”**

ETHER PRICE
$3,169.41 @ 0.034092 BTC (-2.10%)

MARKET CAP
$381,891,866,118.00

Assessed on Jan 2025 https://etherscan.io

Introducing the Ethereum platform

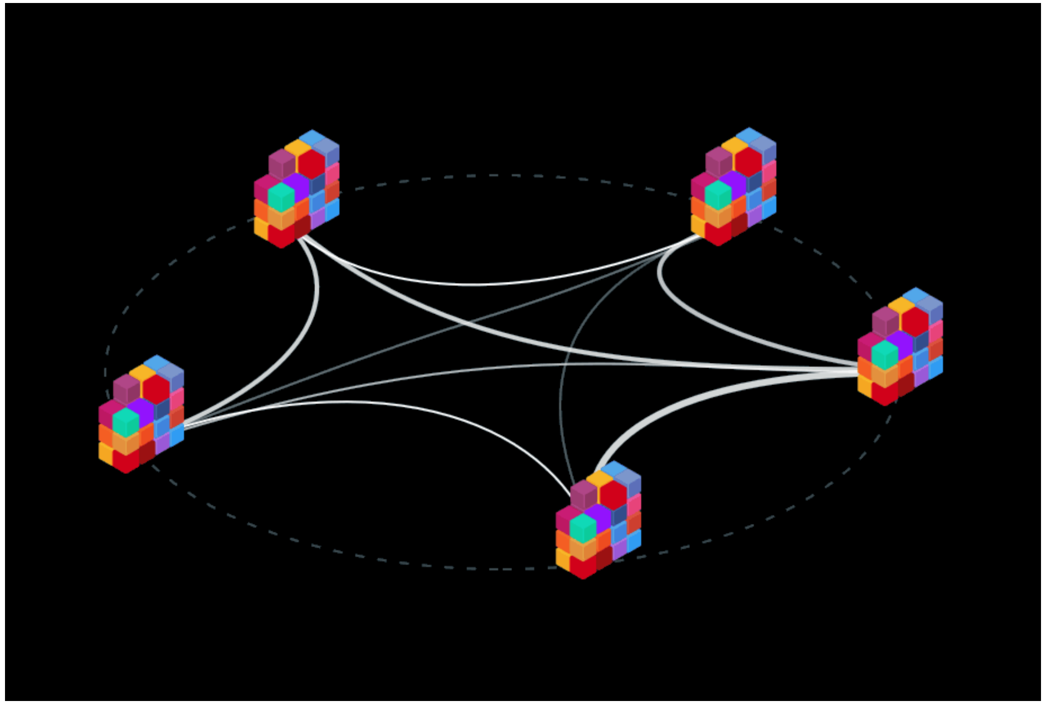


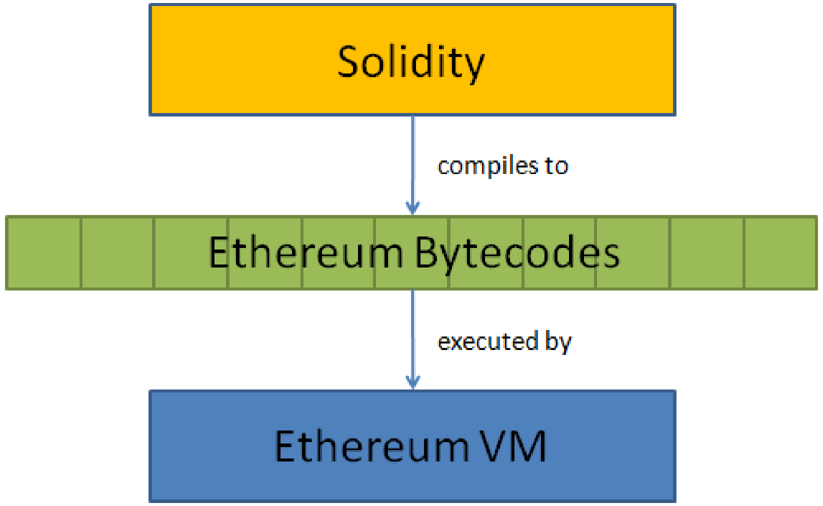
Figure by ethereum.org

In-built cryptocurrency

- Ether
  - Mined in a way similar to Bitcoin
  - Block reward: ~ 3 ETH
- Currently $3100+ per ether
- The crypto-fuel for the Ethereum network
  - Healthy ecosystem: a form of **payment** to compensate the machines for executing the requested operations
  - Also, an incentive ensuring that developers write quality applications (wasteful code costs more)

General purpose script language

- High-level languages : Serpent, *Solidity*, LLL
- Similar to Java script
- Written contracts are compiled to *EVM code* and deployed on EVM
- Each contract has unique address on EVM
- Execution?

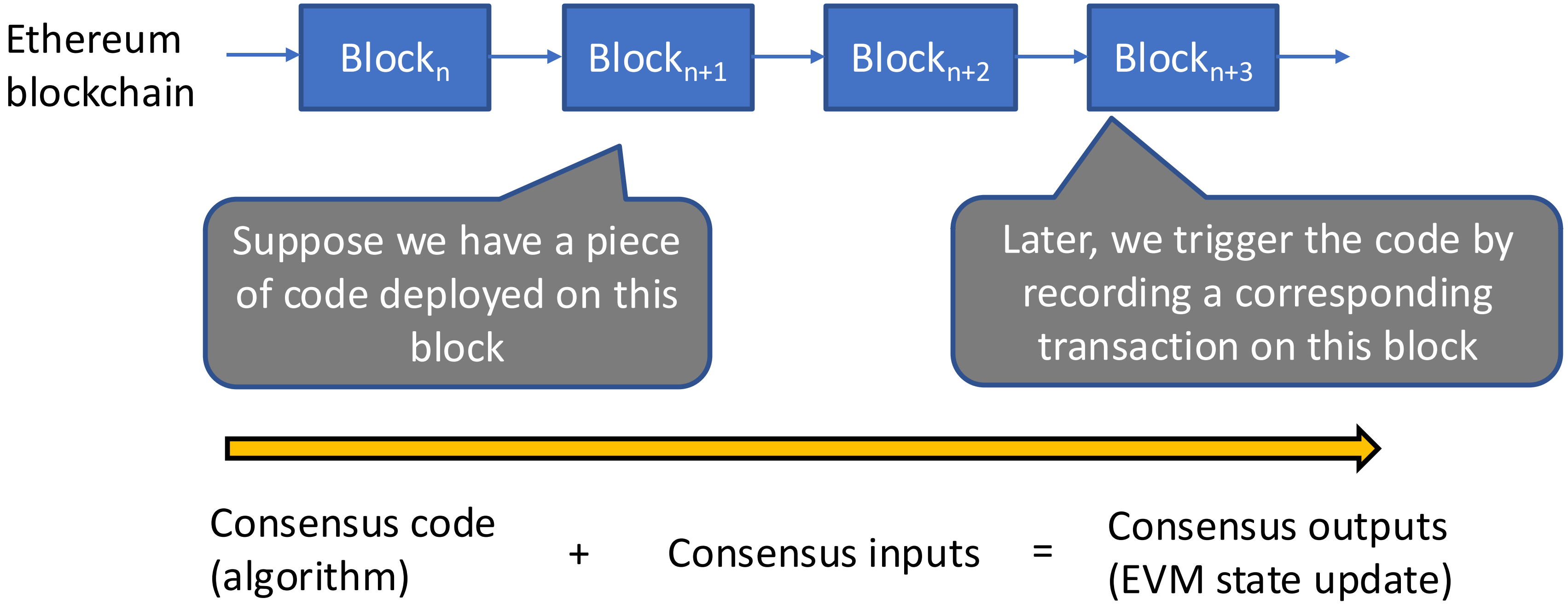


Code execution

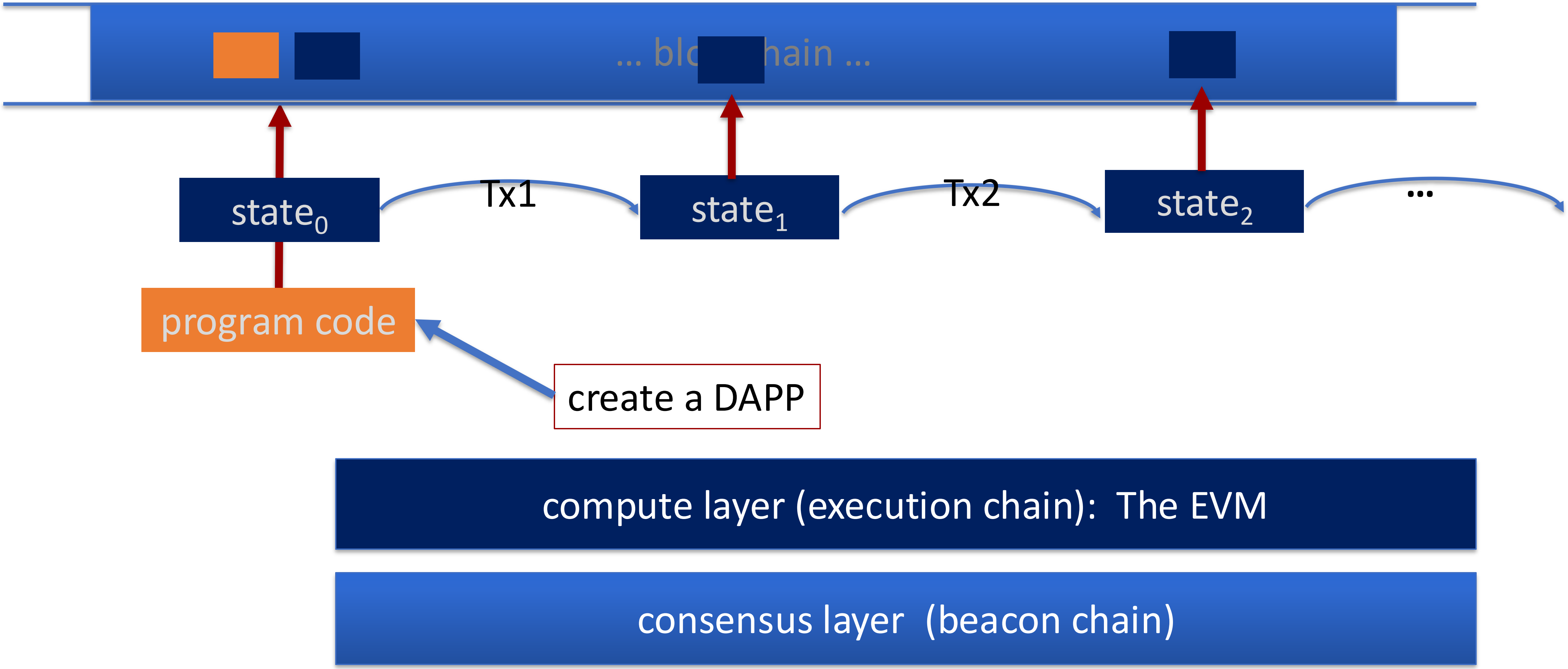
- Deployed contracts (code) are triggered by transactions
  - Transactions specify a TO contract address it sends to
  - Record on the blockchain
  - EVM executes it with the deployed code
- Code can:
  - Send ETH to other contracts / individuals
  - Read / write storage
  - Call (i.e., start execution in) other contracts

Code execution

- Executed by all nodes in the Ethereum network
  - Ask any node to learn the updated state of the EVM



Running a program on a blockchain (DAPP: decentralized applications)



Gas system

- We cannot tell whether or not a deployed code will run **infinitely**
  - Waste computing resources
  - Might damage system robustness
- Ethereum solution: charge fee per **computational step (gas)**
  - e.g. adding two numbers costs 3 gas, calculating a hash costs 30 gas, and sending a transaction costs 21000 gas
  - Special gas fees also apply to *storage* operations
  - **Currently, the gas price is around 2 * 10^{-9} ether (https://ethgasstation.info)**

Gas limit

- Specify the *maximum amount of gas* the sender is willing to buy in a transaction
  - The code processing stops If the gas used exceeds this limit during execution
  - The sender still has to pay for the performed computation
- Gas limit also applies to each block
  - Similar to the maximum block size in Bitcoin
  - Voting mechanism (can upvote/downvote gas limit by 0.0976% every block)

Writing Ethereum Smart Contract

- Programming language: Solidity
  - Contract-oriented
  - Syntax similar to Javascript
- A contract is similar to a class
  - State variables
  - Functions
  - Function Modifiers
  - Events
- Types: integer, string, array, mapping…

We will talk more about this in the tutorial session ☺

Simple smart contract: Lottery

Contract `Lottery` (pseudocode)

Init:

Tend := 28 February 2018, $ticket := 1, pool := {}, pot := 0

Function TicketPurchase():

On receiving $amt from some party **P**:

Assert $amt = $ticket, balance[**P**] ≥ $amt

balance[**P**] := balance[**P**] - $ticket

pot := pot + $ticket

pool := pool ∪ **P**

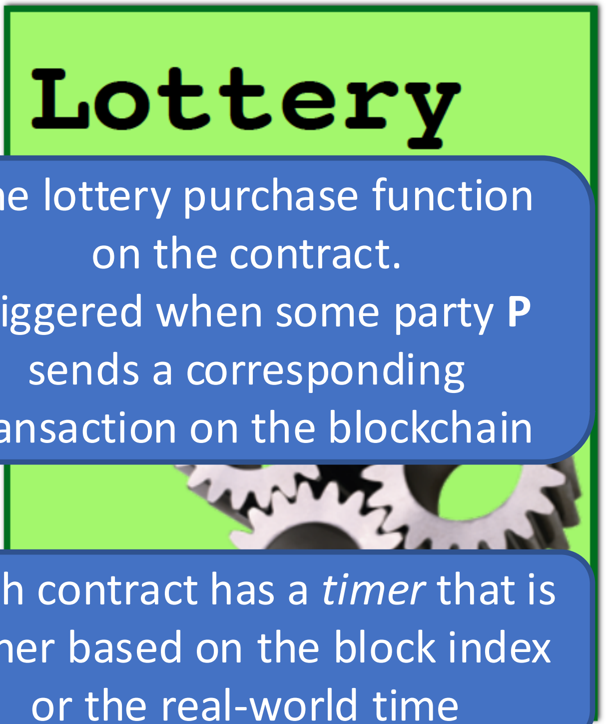
Contract timer: T

If T > Tend then:

$\mathrm{W} \overset{R}{\Leftarrow} \mathrm{pool}$

balance[W] := balance[W] + pot

Set up contract parameters and data structures



The lottery purchase function on the contract. Triggered when some party **P** sends a corresponding transaction on the blockchain

Each contract has a *timer* that is either based on the block index or the real-world time

When certain condition is satisfied, this part automatically executes

Solidity

- A *contract-oriented*, high-level language for implementing smart contracts
- Targets the Ethereum Virtual Machine (EVM)
- Syntax at https://solidity.readthedocs.io/

```
pragma solidity ^0.4.0;

contract SimpleStorage {
    uint storedData;

    function set(uint x) public {
        storedData = x;
    }

    function get() public constant returns (uint) {
        return storedData;
    }
}
```

Browser-solidity

- An useful integrated development environment (IDE) with solidity
  - Design and run smart contract on a Java VM environment
  - Debugging
- Easier for beginners
  - Will not consume “real” money
  - Need not to set up your local EVM
    - E.g., ethereumjs-testrpc https://github.com/ethereumjs/testrpc
- https://remix.ethereum.org/

Related References

- Andrew Miller and Ari Juels and Elaine Shi and Bryan Parno and Jonathan Katz. "Permacoin: Repurposing Bitcoin Work for Data Preservation", in Proc. of IEEE S&P 2014.
- F. Zhang, I. Eyal, R. Escriva, A. Juels and R. V. Renesse. "REM: Resource-Efficient Mining for Blockchains", in Proc. of USENIX Security, 2017.
- Aggelos Kiayias and Alexander Russell and Bernardo David and Roman Oliynykov. "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol", in Proc. of CRYPTO, 2017.
- Yossi Gilad and Rotem Hemo and Silvio Micali and Georgios Vlachos and Nickolai Zeldovich. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies", in Proc. of SOSP, 2017.
- Eleftherios Kokoris-Kogias and Philipp Jovanovic and Linus Gasser and Nicolas Gailly and Ewa Syta and Bryan Ford. "OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding", in Proc. of IEEE S&P, 2018.
- Mustafa Al-Bassam and Alberto Sonnino and Shehar Bano and Dave Hrycyszyn and George Danezis. "Chainspace: A Sharded Smart Contracts Platform". in Proc. of NDSS, 2018.
- Loi Luu and Viswesh Narayanan and Chaodong Zheng and Kunal Baweja and Seth Gilbert and Prateek Saxena. "A Secure Sharding Protocol For Open Blockchains". in Proc. of ACM CCS 2016.
- Mahdi Zamani and Mahnush Movahedi and Mariana Raykova. "RapidChain: A Fast Blockchain Protocol via Full Sharding". in Proc. of ACM CCS 2018.