SUBSPACE NETWORK

Technical Overview

subspace.network

Agenda

Stanford CBR Seminar

1. Introduction to Subspace
2. Honest Consensus Protocol
3. Arrive at a Secure Construction

Bonus – Storage & Execution Protocols

Background

How to tally votes in Nakamoto Consensus?

Proof of Work

one-CPU-one-vote

Proof of Stake

one-Coin-one-vote

Proof of Capacity

one-Disk-one-vote

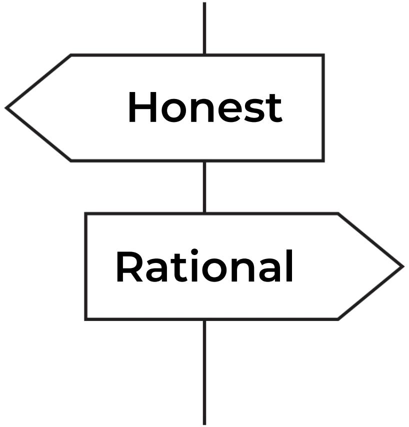
Background

Storage required to run a node

	Bitcoin	Ethereum	chia
Archival Node *full state + history*	403 GB	10.5 TB	~ 300 GB
Full Node *full state*	4.7 GB	633 GB	~ 123 GB
Light Client *minimal state*	58 MB	1 GB	233 KB

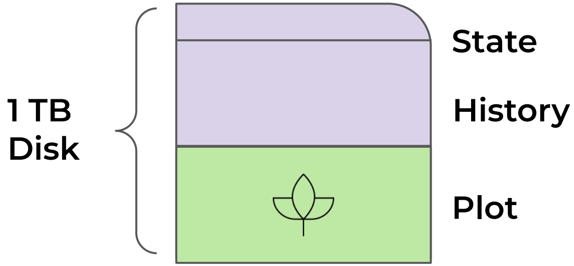
Problem

The Farmer's Dilemma

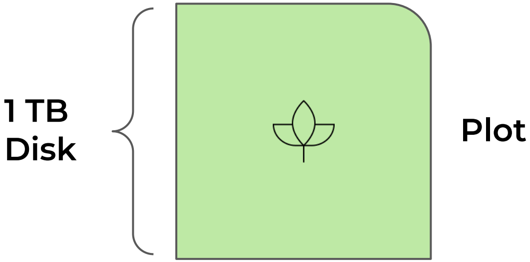


✓ Retain the history
✓ Maintain the state
✓ Verify new blocks

✗ Ignore the history
✗ Discard the state
✗ Trust new blocks



Higher RoI ($ / TB)



Solution: Subspace

How to make PoC Incentive-Compatible

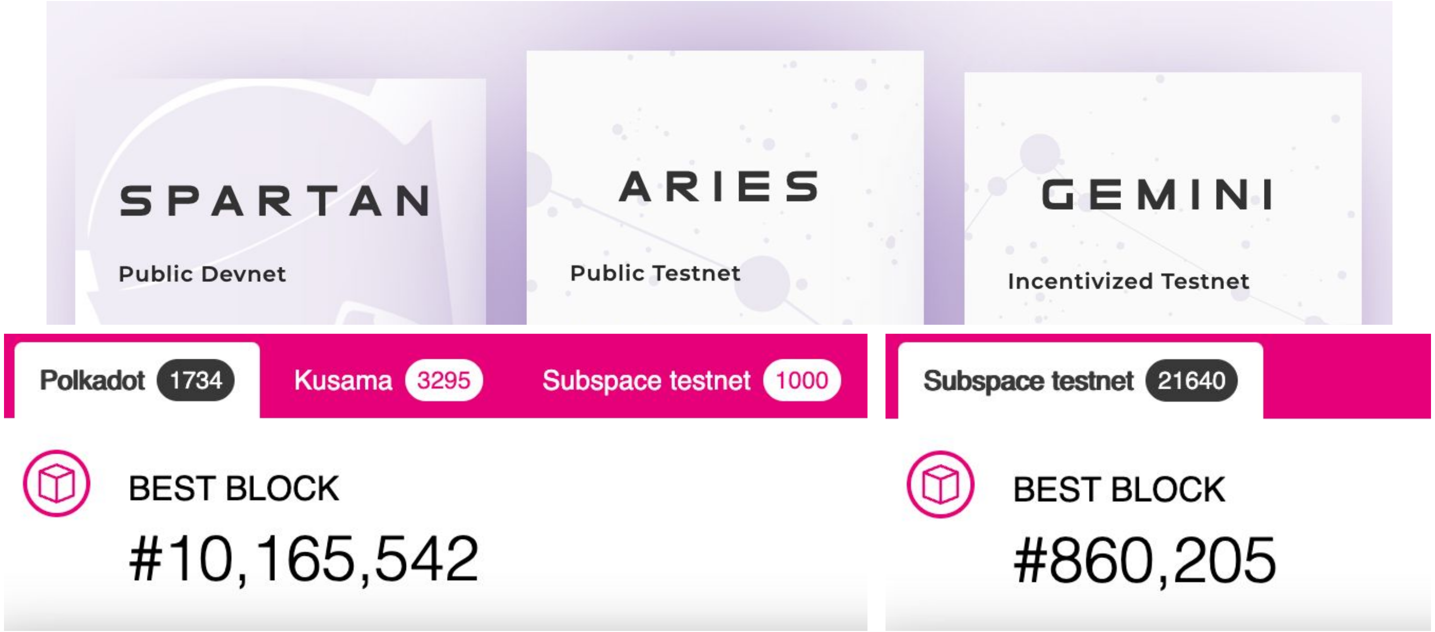
Incentivize nodes to store the history
Proof-of-Archival-Storage (PoAS) Consensus

Separate consensus and compute
Decoupled Execution (DecEx)

Spread the history across all nodes
Distributed Storage Network (DSN)

Public Testnet

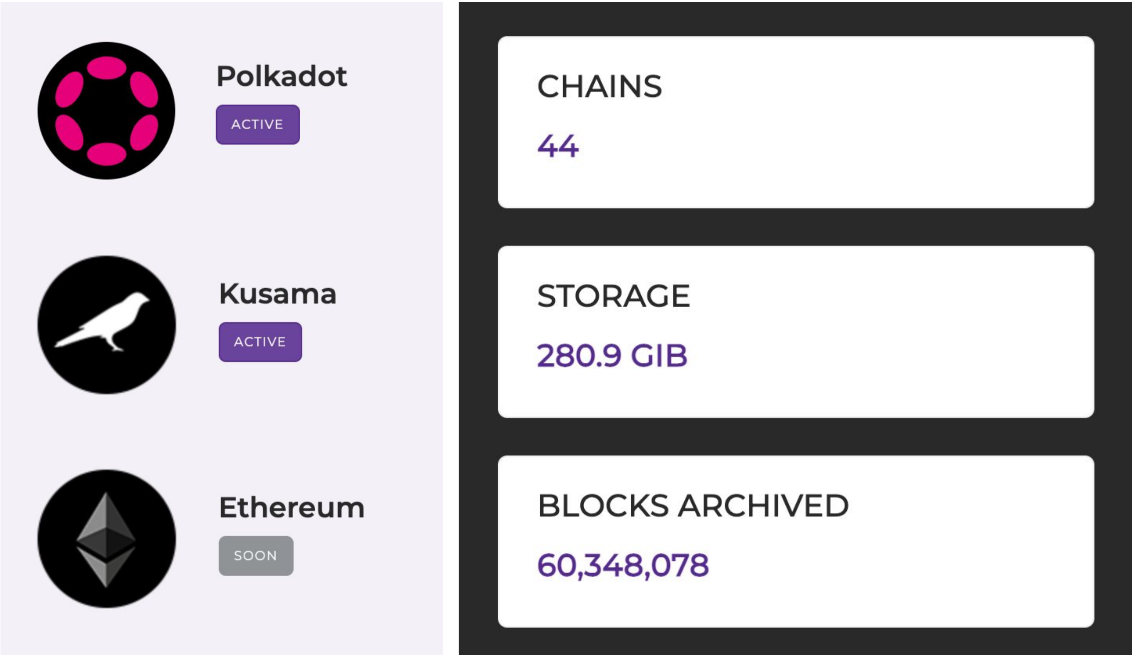
Open Farming for All



https://telemetry.subspace.network/

Permanent Archival Storage

Subspace Relayer



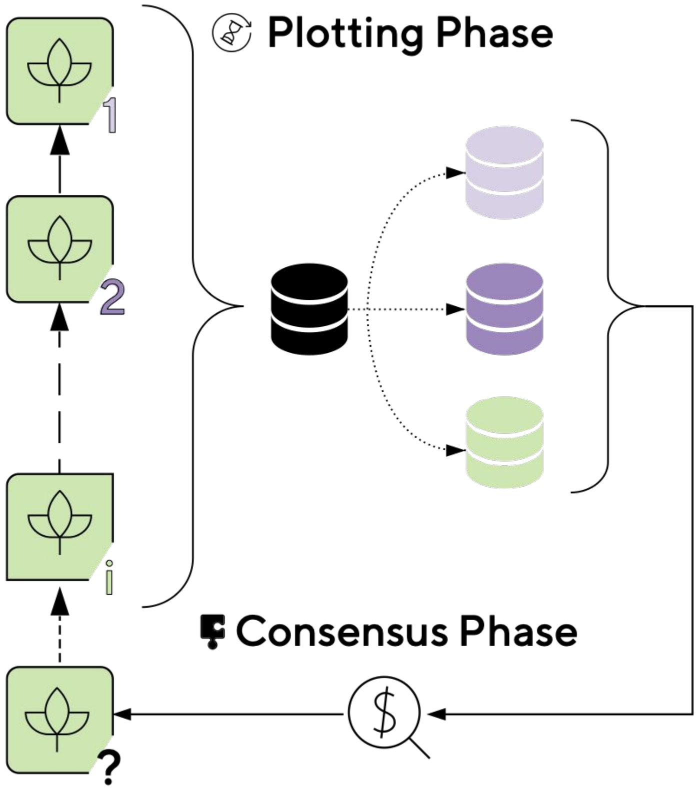
https://testnet-relayer.subspace.network/

Consensus Layer

How to agree on transaction ordering

PoAS Consensus Layer

Proof of Archival Storage



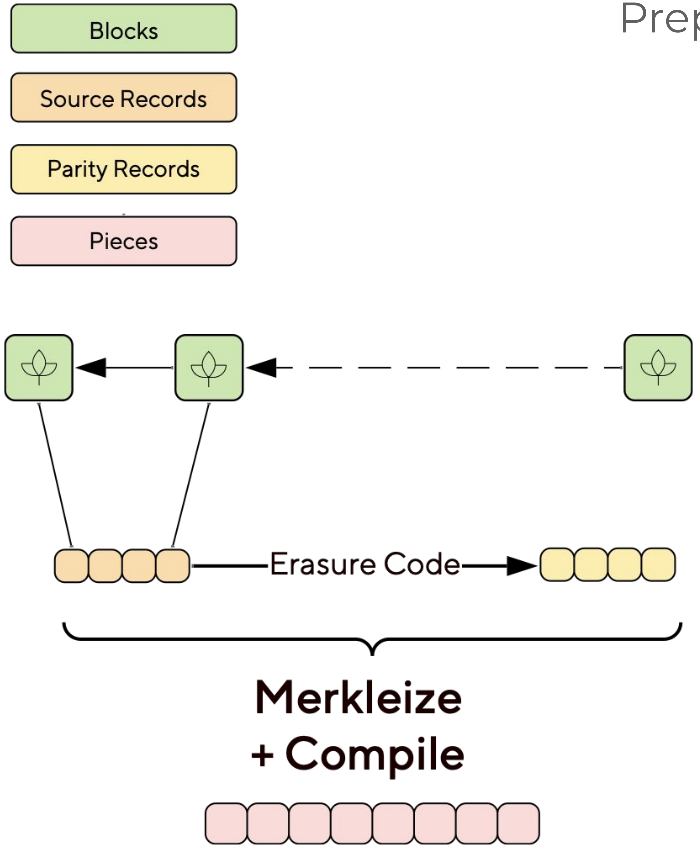
Two Phase Protocol

1. Initial Setup Phase (plotting) – create a unique copy of the history

2. Challenge-Response Phase (Consensus) – audit the history and produce blocks

Archiving

Preparing the history

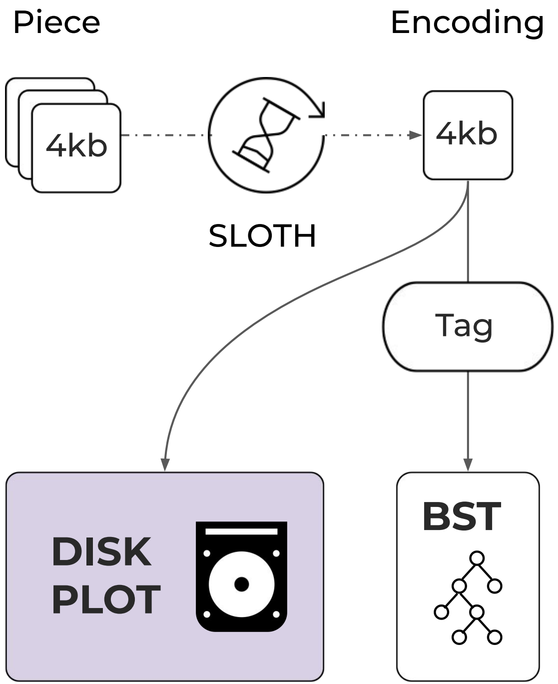


For each new block

1. Append to a buffer of some size
2. When buffer is full
   - Slice into a set of source records
   - Erasure code a set of parity records with some rate
   - Merkelize the entire record set
   - Append a merkle proof to each record, yielding a piece set
3. Commit to a root chain block

Plotting

Initial Setup Phase

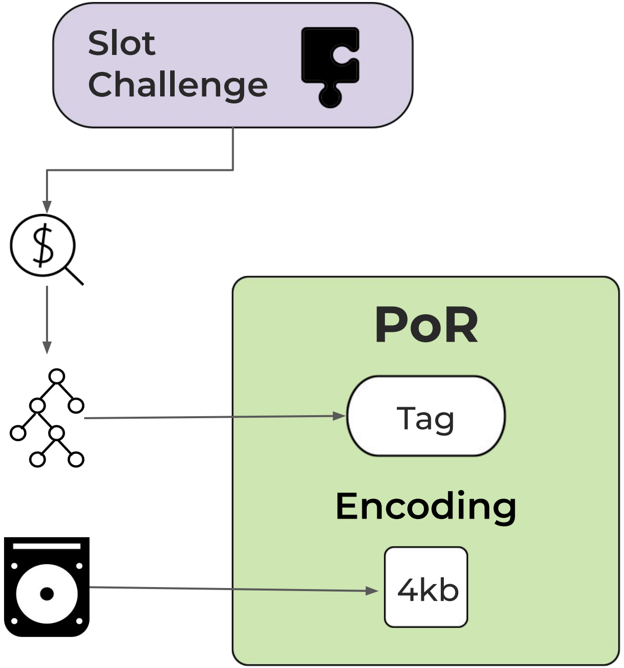


For each 4096 byte piece of history

1. Encode using SLOTH-256-189 where key is hash of public key
2. Create a tag (commitment) to the encoding
3. Write the encoding to disk (plot)
4. Store the tag prefix within a Binary Search Tree (BST)

Farming

Continuous Challenge-Response Phase



For each *timeslot*

1. Issue a random challenge
2. Query BST for nearest tag to the challenge by XOR distance
3. If within dynamic solution range: compile a Proof-of-Replication (PoR)
    a. Sign the tag and challenge
    b. Attach encoding and public key
    c. Broadcast to the network
4. All nodes verify the PoR
    a. Ensure tag w/in solution range
    b. Decode and verify witness
    c. Check the signature

How PoAS Compares

A *permissionless* proof-of-*useful*-storage

	Proof-of-Space ***useless*** *data*	**Proof-of-Storage** ***useful*** *data*
Permission**less** *dynamic availability*	chia	SUBSPACE NETWORK
Permission**ed** *farmer registration*	spacemesh	Filecoin

Securing Consensus

Against all known attack vectors

Security Properties

Goals & Assumptions

Simple Proofs of Space-Time and Rational Proofs of Storage

Tal Moran[*] Ilan Orlov[†]

Abstract

We introduce a new cryptographic primitive: Proofs of Space-Time (PoSTs) and construct an extremely simple, practical protocol for implementing these proofs. A PoST allows a prover to convince a verifier that she spent a "space-time" resource (storing data—space—over a period of time). Formally, we define the PoST resource as a trade-off between CPU work and space-time (under reasonable cost assumptions, a rational user will prefer to use the lower-cost space-time resource over CPU work).

Compared to a proof-of-work, a PoST requires less energy use, as the "difficulty" can be increased by extending the time period over which data is stored without increasing computation costs. Our definition is very similar to "Proofs of Space" [ePrint 2013/796, 2013/805] but, unlike the previous definitions, takes into account amortization attacks and storage duration. Moreover, our protocol uses a very different (and much simpler) technique, making use of the fact that we explicitly allow a space-time tradeoff, and doesn't require any non-standard assumptions (beyond random oracles). Unlike previous constructions, our protocol allows incremental difficulty adjustment, which can gracefully handle increases in the price of storage compared to CPU work. In addition, we show how, in a crypto-currency context, the parameters of the scheme can be adjusted using a market-based mechanism, similar in spirit to the difficulty adjustment for PoW protocols.

1 Introduction

A major problem in designing secure decentralized protocols for the internet is a lack of identity verification. It is often easy for an attacker to create many "fake" identities that cannot be distinguished from the real thing. Several strategies have been suggested for defending against such attacks (often referred to as "sybil attacks"); one of the most popular is to force users of the system to spend resources in order to participate. Creating multiple identities would require an attacker to spend a correspondingly larger amount of resources, making this attack much more expensive.

https://eprint.iacr.org/2016/035.pdf

Assuming 51% of the resources are controlled by economically rational nodes. Given the proper security parameters, the the protocol shall:

1. Maintains the *safety and liveness* properties of Nakamoto consensus
2. Maintains the *fairness* of *one-disk-one-vote*

Against *all* known attacks...

Security Outline

Categorizing Attacks

1 Secure Proof-of-Storage

- *Lazy Farming*
- *On-Demand Encoding*
- *Compression Attacks*

2 Secure Randomness

- *Grinding Attacks*
- *Public Simulation*
- *Private Simulation*

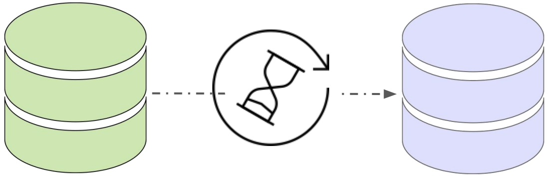
3 Secure Notion of Time

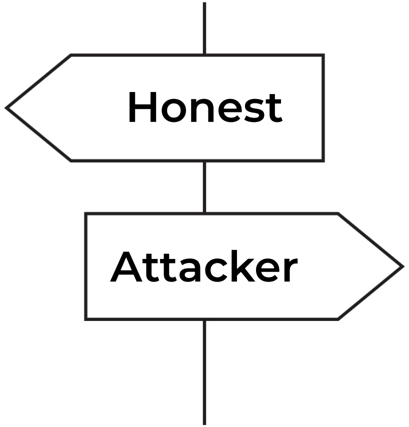
- *Prediction Attacks*
- *Long-Range Attacks*
- *Desync. Attacks*

Lazy Farming

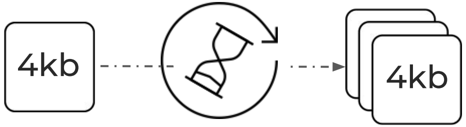
Breaking archival storage

Honest farmer downloads the full history and creates a single unique replica with one identity





Lazy farmer downloads a single piece of the history and encodes it many times under different identities



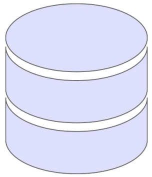
Solution

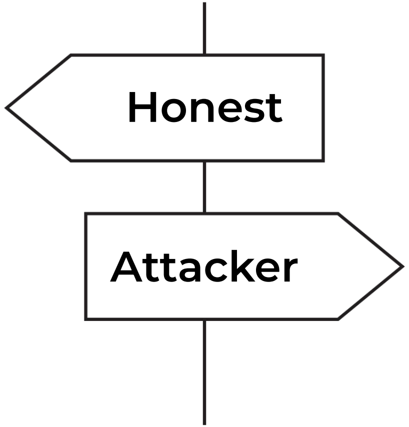
Salt challenge with identity s.t. each BST is unique

On-Demand Encoding

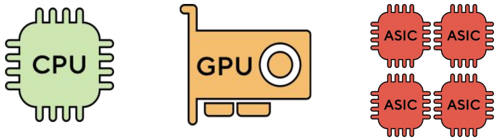
Mining PoAS

Honest farmer pre-encodes their plot to disk and queries BST on each challenge





Attacker attempts to create the same number of encodings on-demand within the window

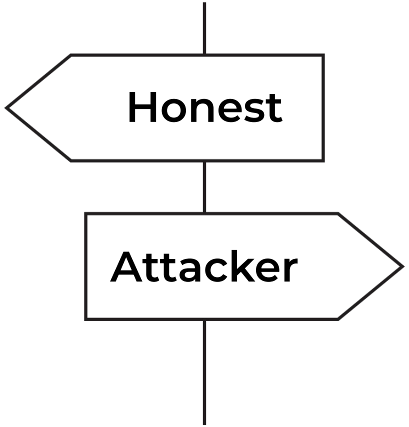


Solution

Tune encoding delay s.t. mining always uses more energy than farming

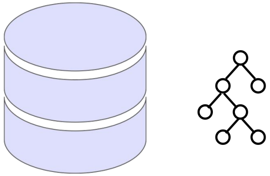
Compression Attack

Discarding the Plot



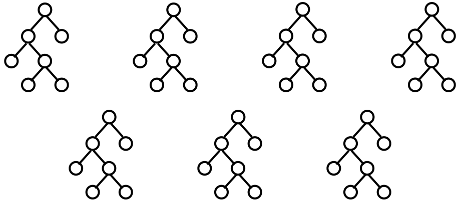
Honest farmers retains
a large (1TB) plot and a small
(8 GB) binary search tree (BST)

An attacker could "compress"
their plot by only storing
the BST, many times...



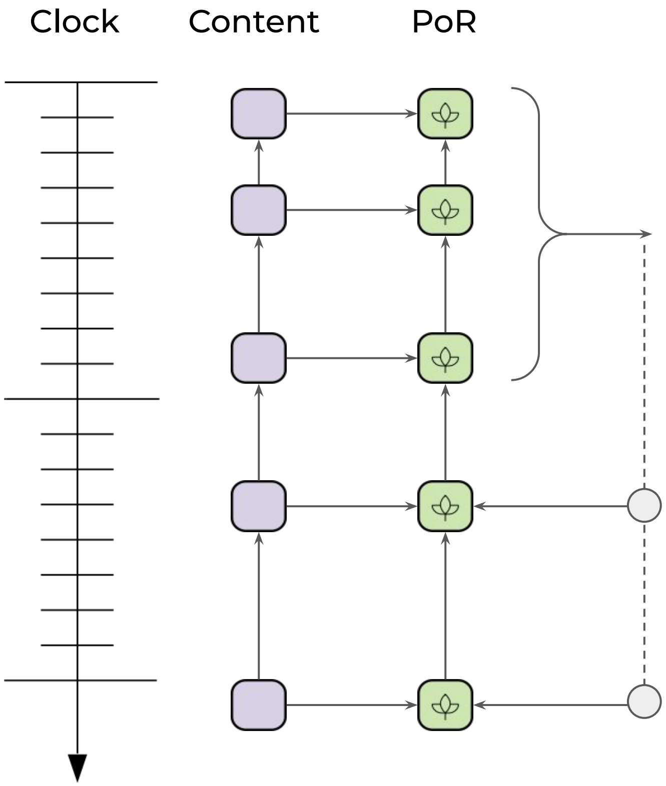
Solution

Require farmers to commit to
their plots at some interval
yielding a "salted" BST



Securing Randomness

Maintaining the chain structure



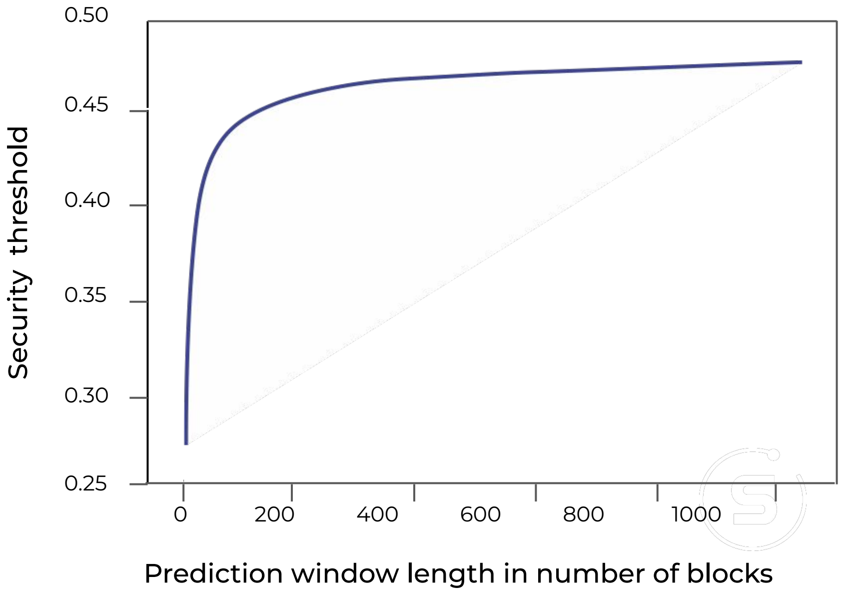
Content Grinding
Segregate the content and proof

Public Simulation (Equivocation)
Burn plots by blocklisting signing keys

Private Simulation
Recycle the challenge over many rounds

C-Correlation

Reducing Predictability



As the slot update interval (prediction window) is increased, the security threshold increases from 27% to 50%. At an window of 32 blocks (3 mins), the security is up to 42%. At a window of 256 blocks (26 mins), the security is up to 47%

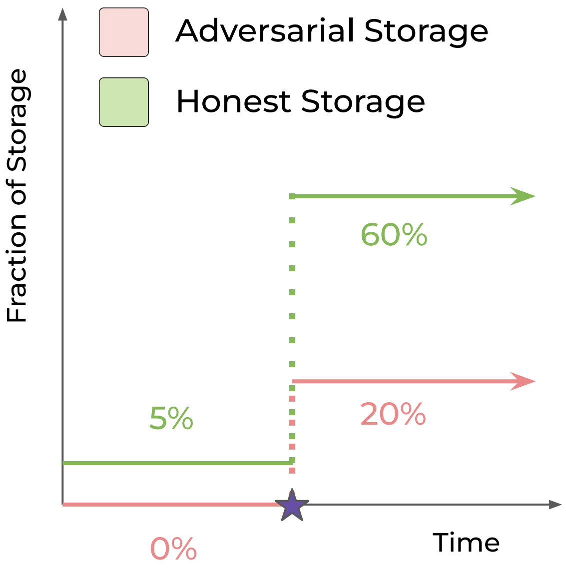
But prediction window allows for:

- Bribery Attacks
- Improved On-Demand Encoding

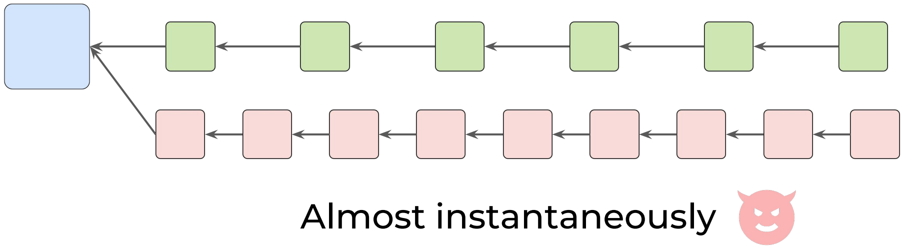
https://arxiv.org/abs/1910.02218

Long-Range Attack

Rewriting History from Genesis

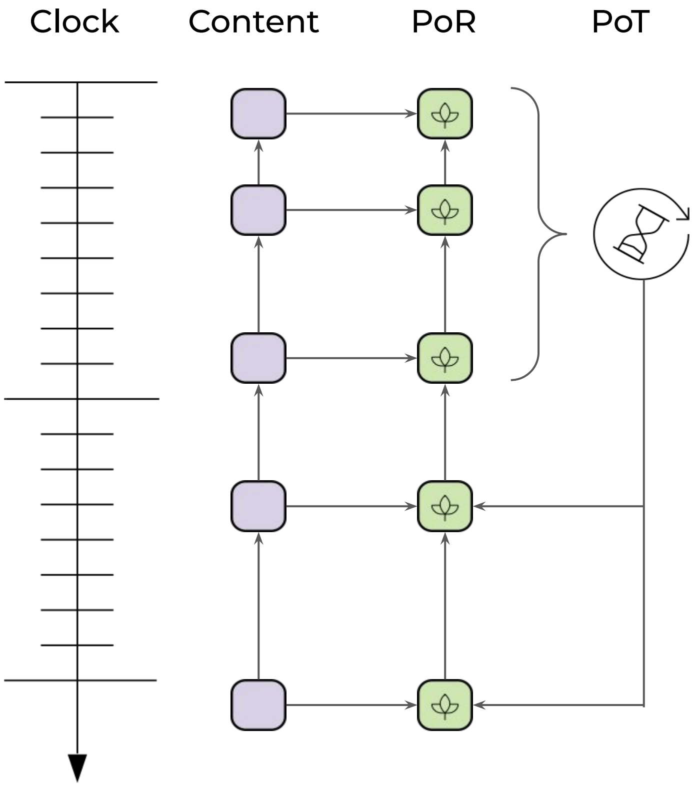


Originally considered hard for PoAS due to replotting time, but there are efficient forms of the attack...



Proof-of-Time

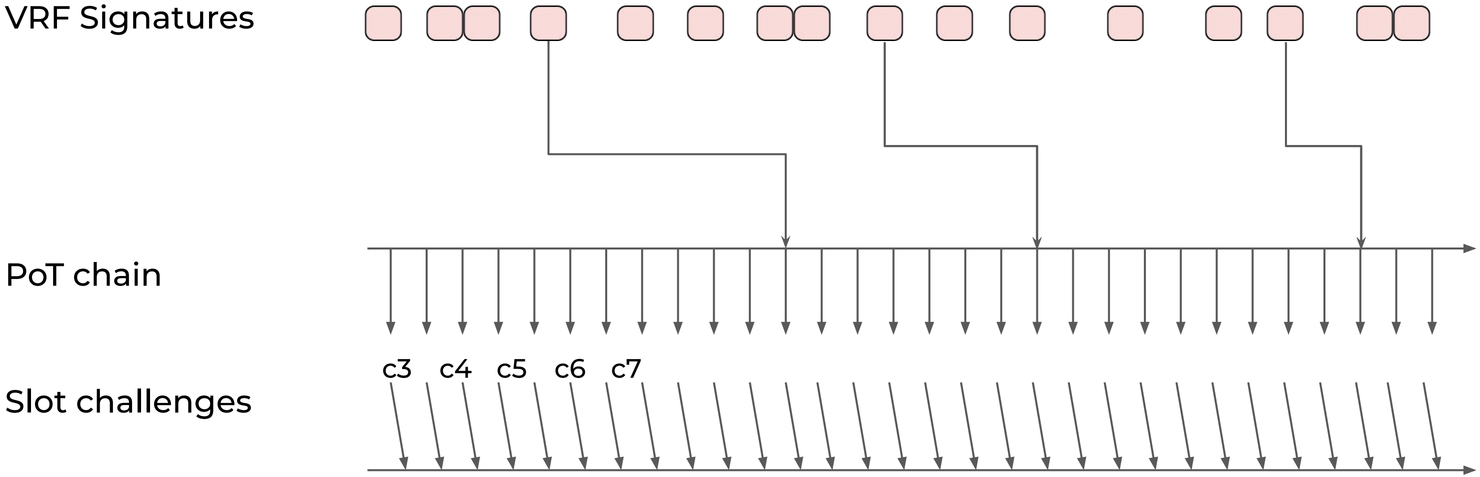
Sealing the History



1. Iterate a permutation over the random beacon
2. Periodically commit to the history
3. Use the output as the challenge
4. Inspired by PoS Arrow of Time (PoSaT)
5. Long range attack is much harder
6. Prediction window is much shorter
7. Use iterated AES to reduce Amax
8. No difficulty reset (soft fork)
9. Executors run the PoT Chain
10. Farmers optimistically follow
11. Anyone can prove invalid PoT

Proof-of-Time

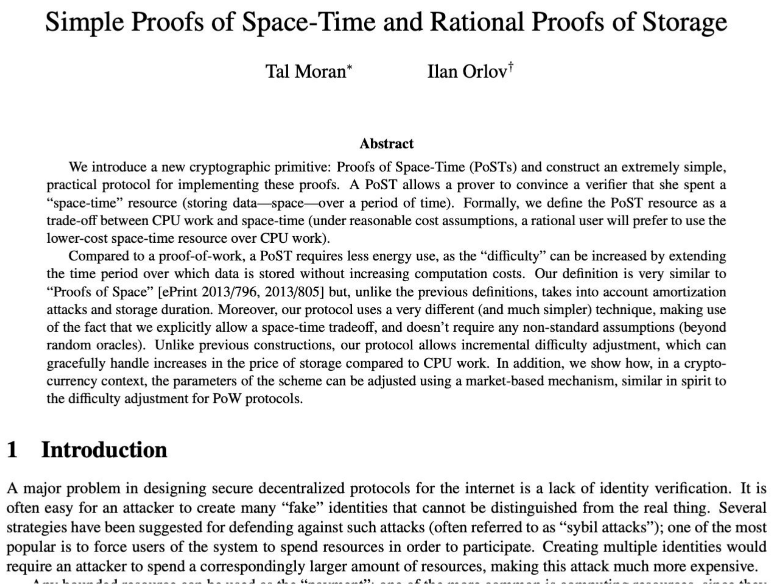
Maintaining the arrow of time



Inspired by PoSaT (PoS with Arrow of Time)
https://arxiv.org/abs/2010.08154

Security Properties

Next Steps



Simple Proofs of Space-Time and Rational Proofs of Storage

Tal Moran* Ilan Orlov†

Abstract

We introduce a new cryptographic primitive: Proofs of Space-Time (PoSTs) and construct an extremely simple, practical protocol for implementing these proofs. A PoST allows a prover to convince a verifier that she spent a "space-time" resource (storing data—space—over a period of time). Formally, we define the PoST resource as a trade-off between CPU work and space-time (under reasonable cost assumptions, a rational user will prefer to use the lower-cost space-time resource over CPU work).

Compared to a proof-of-work, a PoST requires less energy use, as the "difficulty" can be increased by extending the time period over which data is stored without increasing computation costs. Our definition is very similar to "Proofs of Space" [ePrint 2013/796, 2013/805] but, unlike the previous definitions, takes into account amortization attacks and storage duration. Moreover, our protocol uses a very different (and much simpler) technique, making use of the fact that we explicitly allow a space-time tradeoff, and doesn't require any non-standard assumptions (beyond random oracles). Unlike previous constructions, our protocol allows incremental difficulty adjustment, which can gracefully handle increases in the price of storage compared to CPU work. In addition, we show how, in a crypto-currency context, the parameters of the scheme can be adjusted using a market-based mechanism, similar in spirit to the difficulty adjustment for PoW protocols.

1 Introduction

A major problem in designing secure decentralized protocols for the internet is a lack of identity verification. It is often easy for an attacker to create many "fake" identities that cannot be distinguished from the real thing. Several strategies have been suggested for defending against such attacks (often referred to as "sybil attacks"); one of the most popular is to force users of the system to spend resources in order to participate. Creating multiple identities would require an attacker to spend a correspondingly larger amount of resources, making this attack much more expensive.

https://eprint.iacr.org/2016/035.pdf

Assuming 51% of the resources are controlled by economically rational nodes. Given the ***proper security parameters***, the the protocol shall:

1. Maintains the *safety and liveness* properties of Nakamoto consensus
2. Maintains the *fairness* of *one-disk-one-vote*

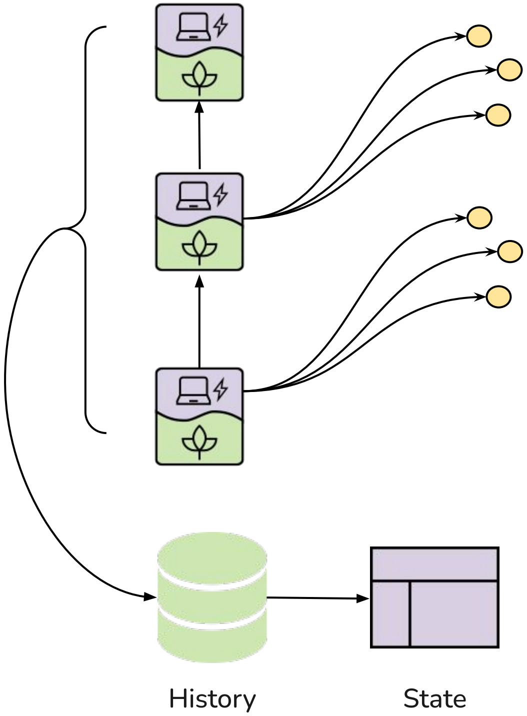
Against *all* known attacks...

Compute Layer

How to agree on the global state

Decoupled Execution

Separation of Concerns



In a standard blockchain, each full node will...

1. Propose new blocks
2. Verify new transactions
3. Maintain chain history
4. Maintain chain state

We separate these roles between two types of nodes

Storage Farmers

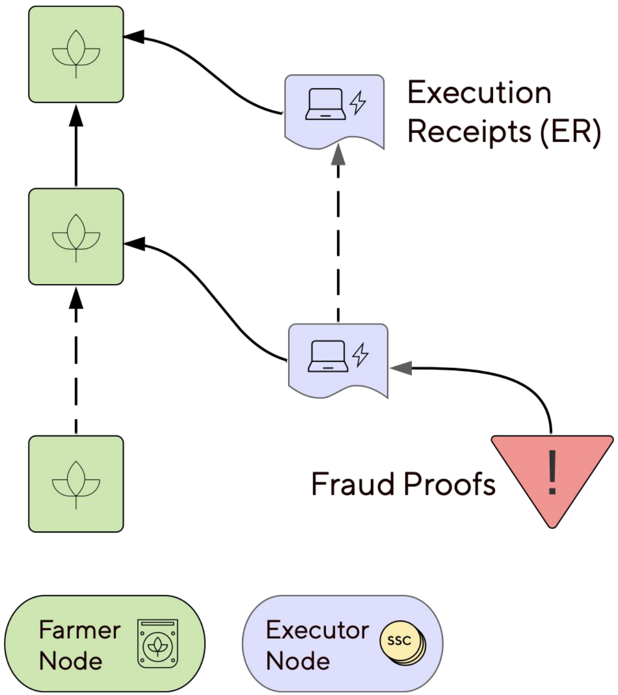
Prove they are storing the actual blockchain history

Staked Executors

Prove they are holding coins and tracking the latest state

Decoupled Execution

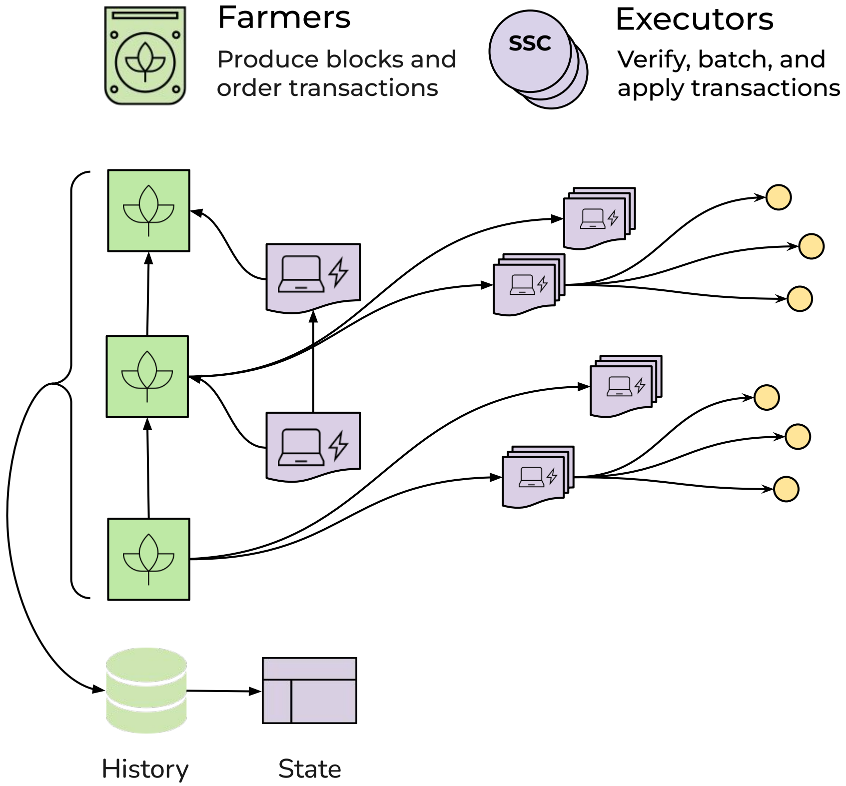
Maintaining Security



1. Farmers produce blocks
2. Executors apply blocks
3. Executors produce ERs proportional to their staked credits
4. Invalid ERs will lead to fraud proofs
5. Farmers can verify fraud proofs, leading to executors being slashed
6. Assumes at least one honest full node that is not under eclipse attack

Decoupled Execution

Vertical Scaling



1. User’s generate transactions and broadcast to executors
2. Executors verify the user has funds to cover the transaction fee
3. Executors produce transaction bundles proportional to their stake
4. Farmers include include bundles into blocks, providing an ordering
5. Executors apply the bundles and generate a new state root
6. Block size not limited by network delay, but farmer bandwidth
7. Data Availability Sampling (DAS) will allows scaling to executor bandwidth

How it Compares

Eventually Eager Execution

flow

OP

Optimism

CELESTIA

Eager Execution

Standard Model

Lazy Execution

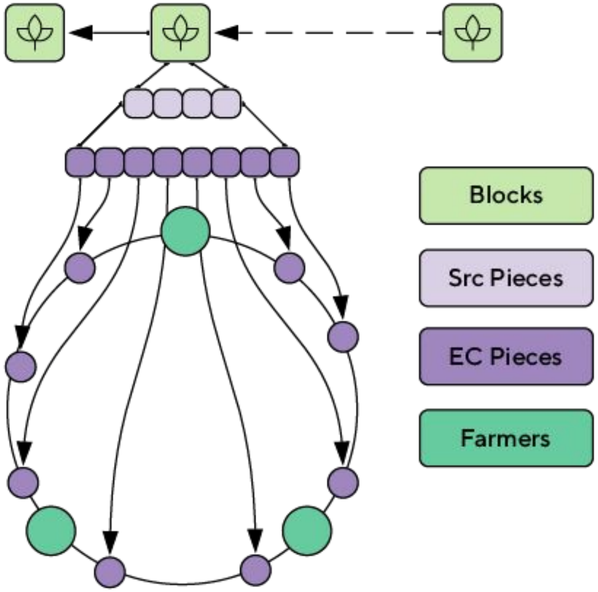
Client Side

Storage Layer

How to ensure data persists

Distributed Storage Of Subspace

How to store and retrieve our own history



Assuming a very large history, we must ensure it remains

- Durable
- Load Balanced
- Retrievable
- Efficient Sync

To achieve this we employ

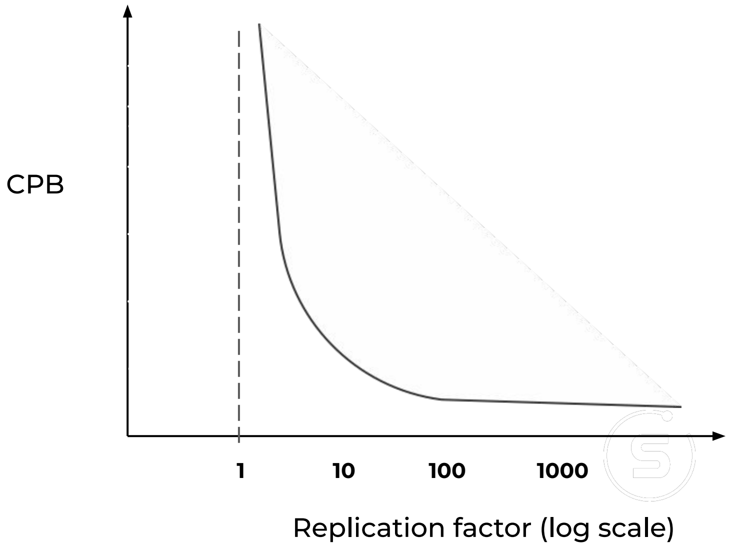
- Erasure coding
- Consistent Hashing
- Light Kademlia DHT
- Super Light Client

Storage Fee Pricing

Incentivizing Permanent Storage

Cost of Storage (CoS)
Subspace Credits / byte (CPB)

$$\rightarrow \frac{\sum \textit{Circulating Credit Supply}}{\sum \textit{Space Pledged} - \sum \textit{Space Reserved}}$$



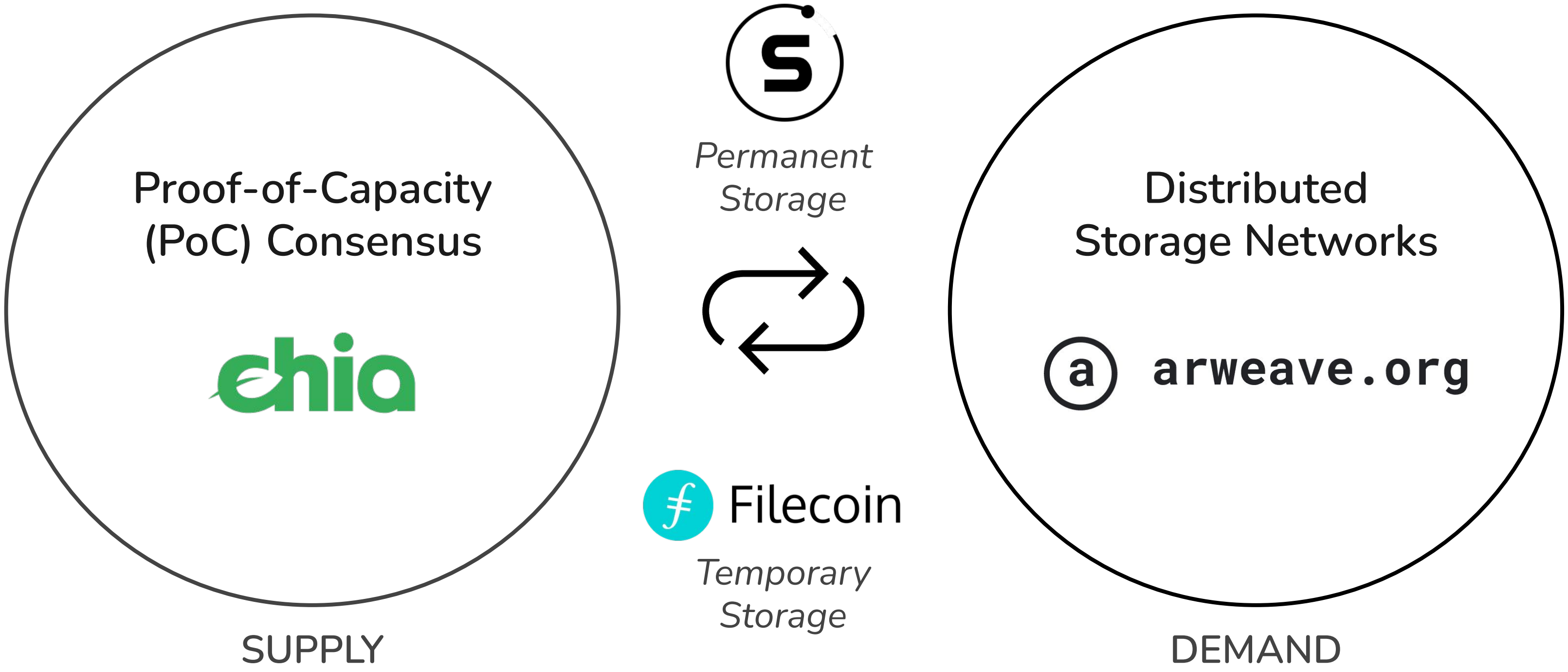
CoS is normally constant
- op_return → satoshis / byte
- call_data → gwei / byte

Our CoS (storage fee) is dynamic
- RF increases, fees get lower
- As RF decreases, fees get higher

Portion of fees are placed in an endowment and paid out gradually

How it compares?

AMM for On-Chain Storage

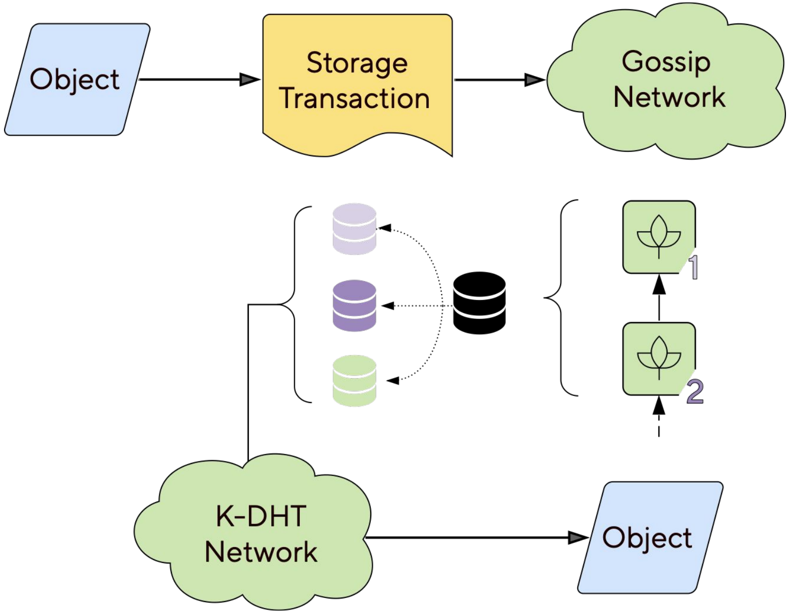


Application Layer

How Ethereum can benefit from Subspace

Storage API

How to store and retrieve any data



Subspace.js – Developer SDK

put(object) → object_id

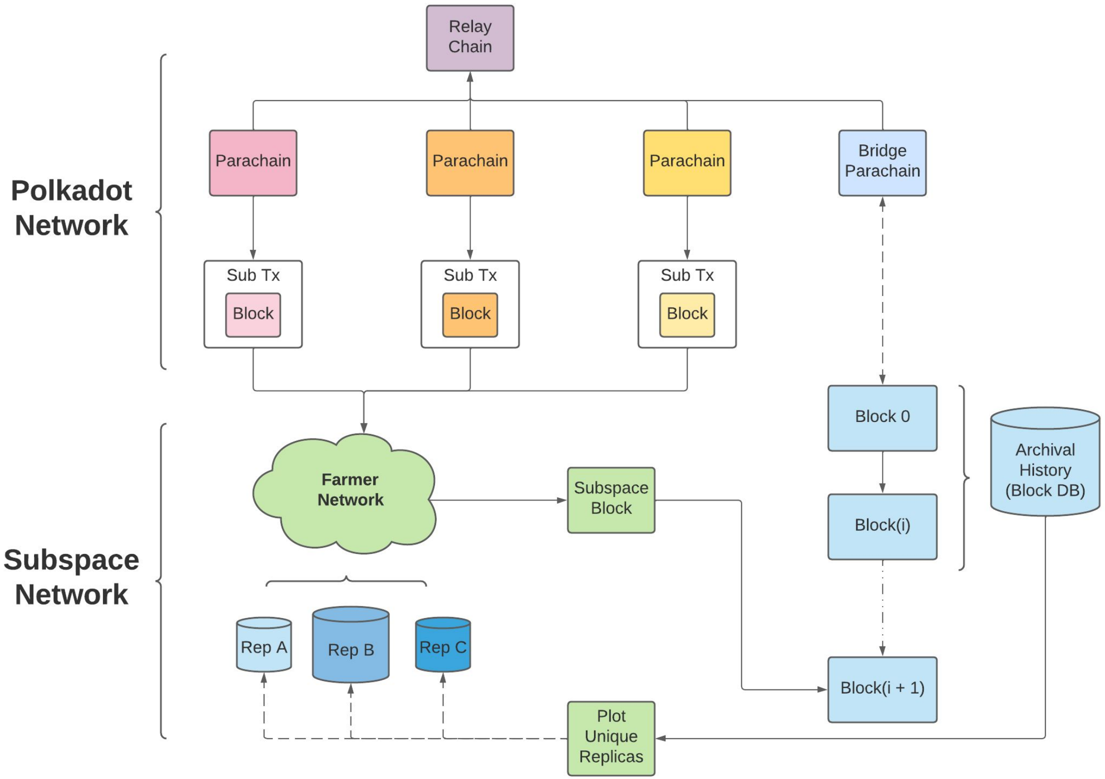
- wrap in subspace transaction
- include within a block
- store mapping on DHT
- pieces are spread across plots

get(object_id) → object

- retrieve mapping from DHT
- retrieve pieces from plots
- reconstruct object and verify

10k Foot Overview

Subspace 🤝 Dotsama



Integration Benefits

Subspace 💜 Everyone

Greater Scalability

A release valve for blockchain bloat

Off-Chain Storage

State Management

Higher Decentralization

Reduced reliance on traditional infra

Node Synchronization

Distributed Archival Nodes

Better Interoperability

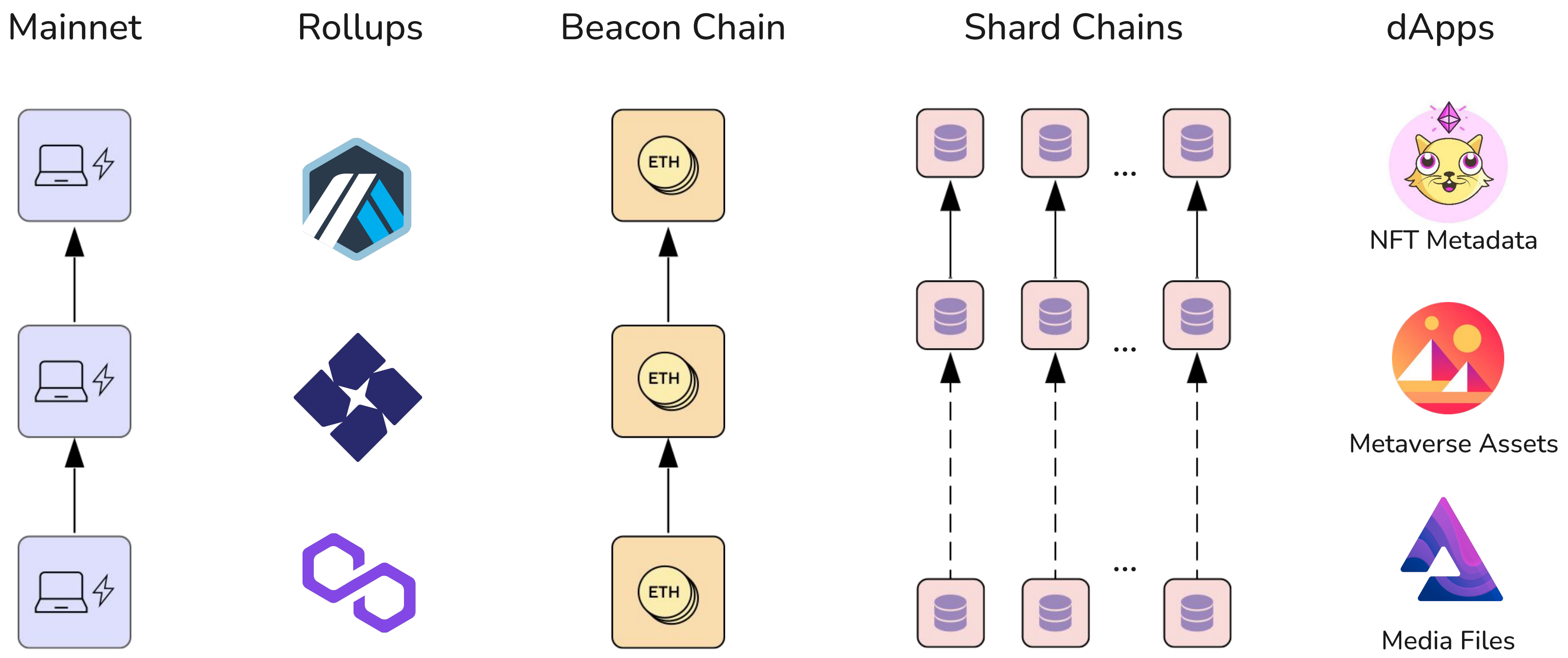
Validated archiving allows for trustless bridging

Common Query API

Cross-Chain Messaging

Ethereum Ecosystem

Where will all of this data live?



Thanks!

And we’re hiring security researchers ;-)