Filecoin

Karthik Inbasekar

Ingonyama

26 Jan 2023

Overview

Introduction

What is Filecoin? - A high level overview

Proof of Replication and Proof of spacetime

Proof of Replication - deep dive

Overview

Replica: Stacked Depth Robust Graph

Poseidon Hash Detour

What to commit and how?

Proof Mechanism

R1CS to Groth16

SnarkPack aggregation

What is Filecoin?

A decentralized storage network, powered by a blockchain and a token - **filecoin**

Different Blockchains use different systems such as

Eg: Bitcoin: Proof of Work, Ethereum: Proof of stake

Filecoin is a Blockchain built on a variation of **Proof of stake - Proof of space-time.**

Instead of tokens as stake, **proven storage determines probability of mining a block**.

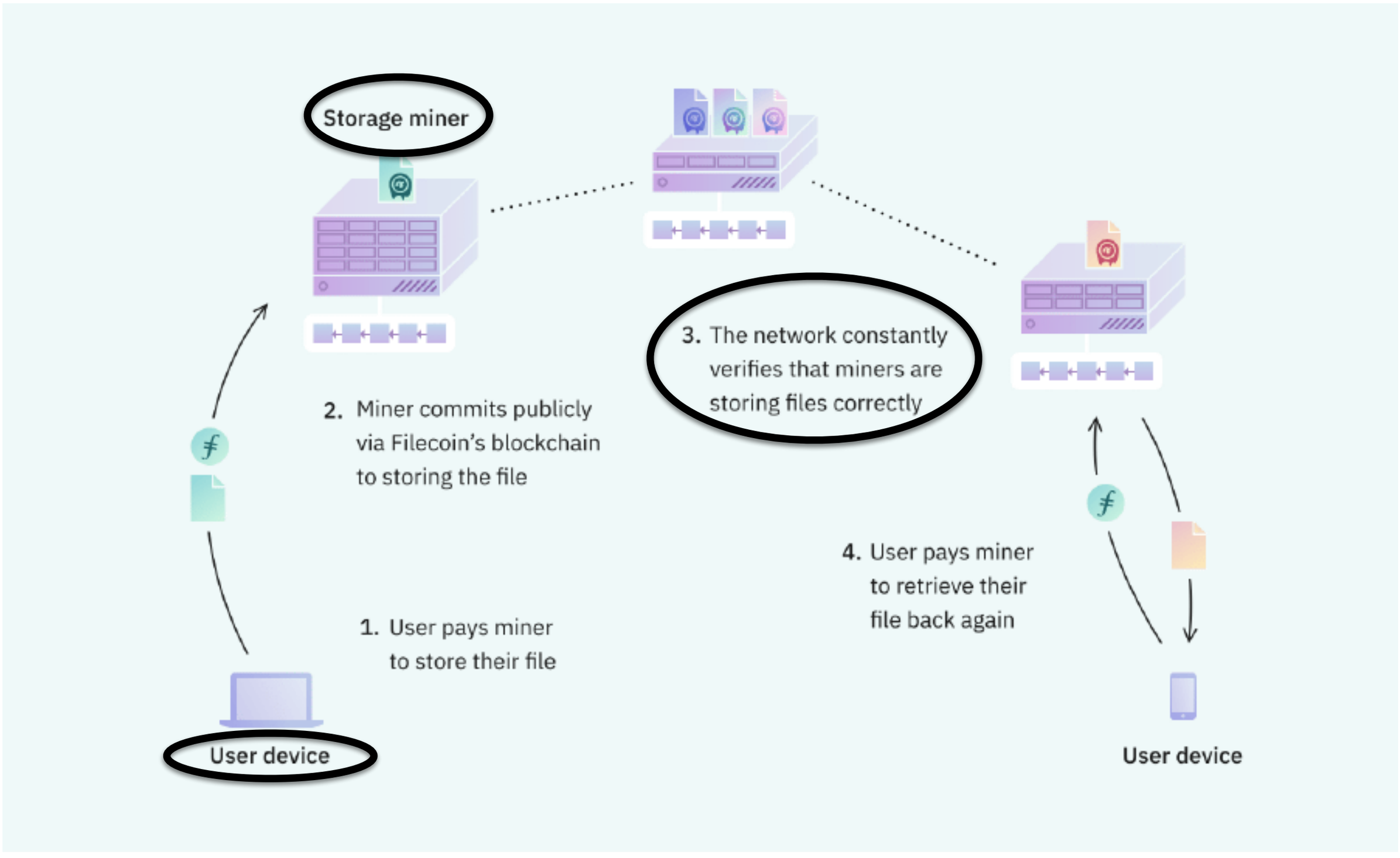
Miner's **power in consensus is proportional to amount of storage**

How does the economics work?

How does one prove to a network that you are indeed storing something?

High level mechanism

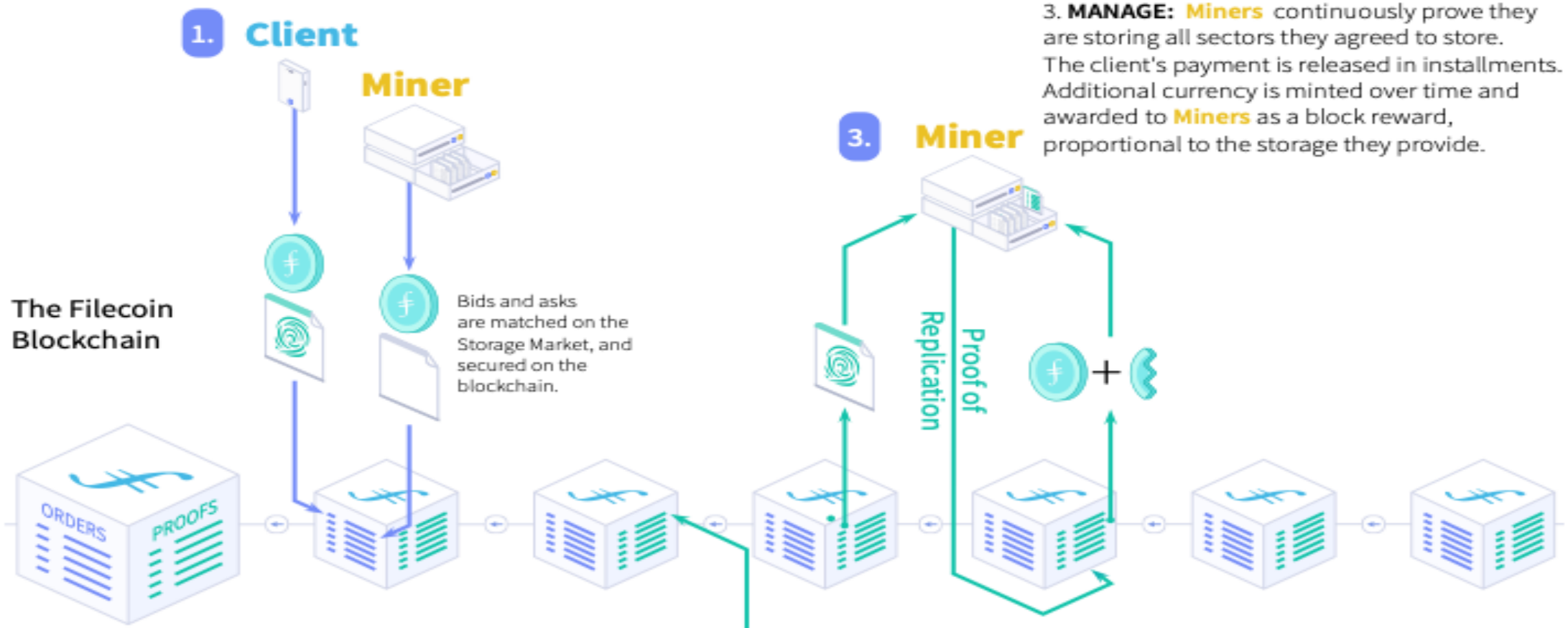
Filecoin functions like a **decentralized storage marketplace**.



Filecoin docs

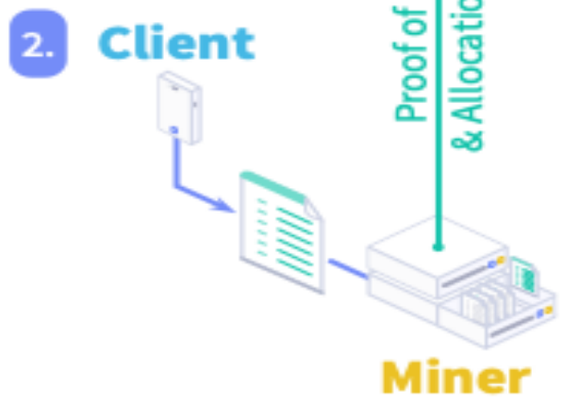
1. **PUT:** Clients send information about the file, storage duration, and a small amount of filecoin to the Storage Market as a bid. Simultaneously, Miners submit asks, competing to offer low cost storage. Deals are made in the Storage Market, on the blockchain.

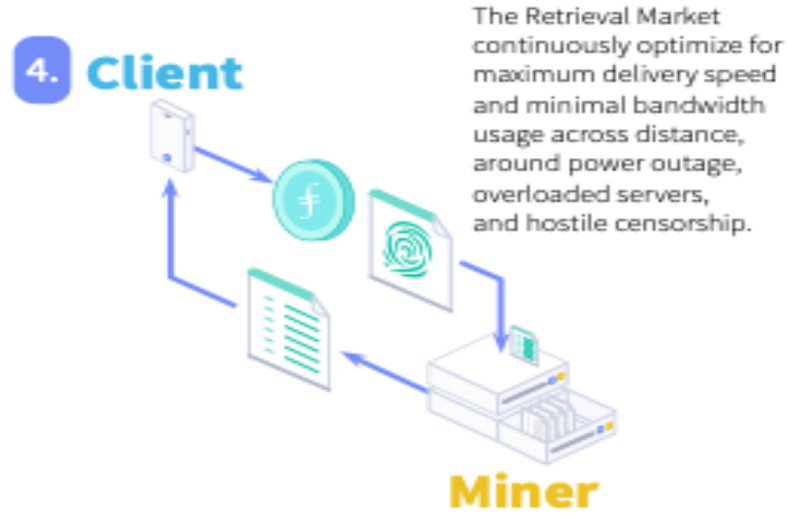
Filecoin docs



3. **MANAGE:** Miners continuously prove they are storing all sectors they agreed to store. The client's payment is released in installments. Additional currency is minted over time and awarded to Miners as a block reward, proportional to the storage they provide.

2. **SEND:** The Client then sends the file to the Miner, and the Miner adds the file to a sector. The sectors are cryptographically sealed, with verification sent to the blockchain.





Economics

4. **GET: A Client requests a file with some payment** in filecoin to the Retrieval Market (off chain); the first Miner to send the file is paid. Eventually, the contract expires and the storage is once again free.

How do you even prove that you store something

Filecoin *verifies* *storage of data*

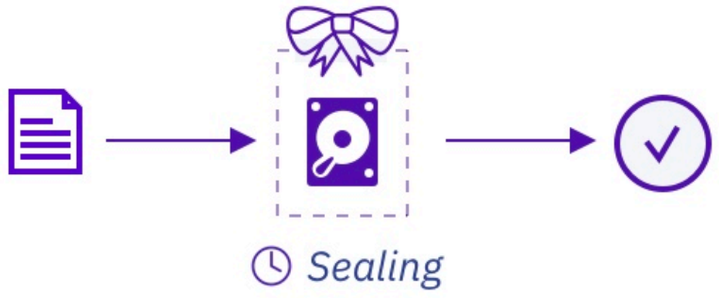
with 2 cryptographic Proofs of Storage

Each Storage Miner must submit proofs that they are continuing to store unique copies of Client data in order to receive payment and rewards.

Proof of Replication

(PoRep)

Based on **Sealing**, a gradual, sequential operation to generate an encoding of the data for each miner, a unique replica.



*Miners provide public **proof** that **a unique encoding of the data exists in physical storage**.*

Proof of Spacetime

(PoSt)

Over time, randomly selected miners have random sectors challenged, from which data is read for verifications and compressed into a PoSt proof.

*Miners provide public proof that a given encoding of the data **existed in physical storage continuously over a period of time.***

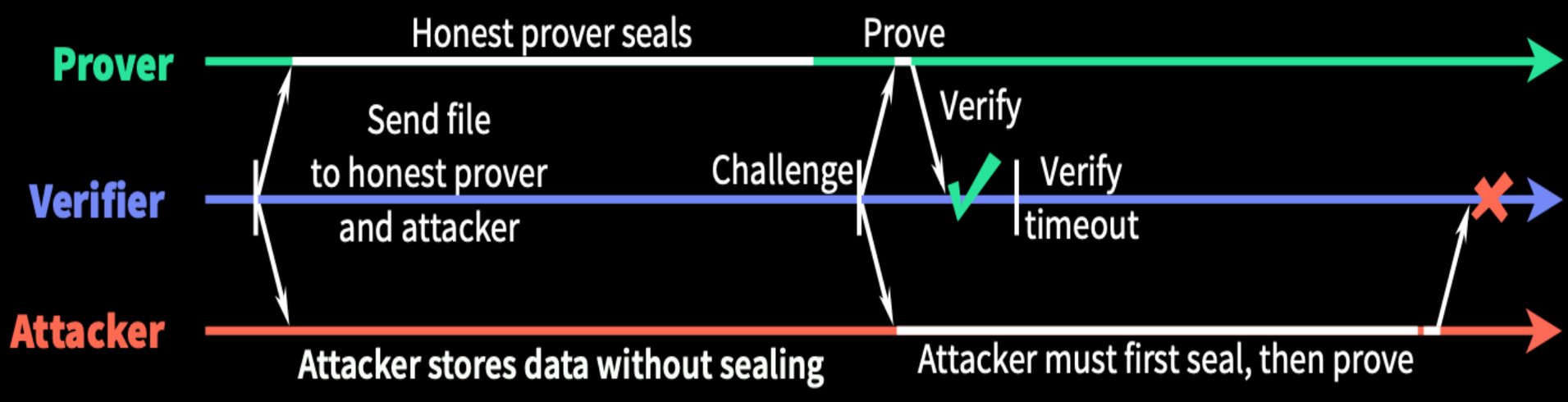
Filecoin docs

Proof of replication high level overview

The Proof-of-Replication

Filecoin docs

The *Proof-of-Replication* functions cryptographically verify that every copy is stored independently, and allow this fact to be checked periodically. Each unique replica is generated by cryptographically sealing the data at the time of storage, using a per-replica encryption key. The *Seal* and *Proving* functions are chosen such that generating a new *Seal* cannot be performed fast enough to compromise the proof. These proofs also create a publicly verifiable record that the miner correctly stored the data as promised, and was online to serve it. The Filecoin Blockchain uses *Proof-of-Replication* in its mining process, to reward miners for storing files, to punish miners who fail their contracts, and to detect and recover any missing pieces.



Sealing is an intense amount of work done on data provided by the client

The proof of replication, is actually a proof of work done on a client data

This proof is compressed using a zk-SNARK and published on chain

Proof of spacetime high level overview

Once the miner has completed the sealing and published a PoRep

Window PoSt

Compute a Merkle proof that sealed data is still stored periodically.

Every 24 hours (epoch) is divided into 30 min windows, and predefined sectors of sealed data are randomly assigned to these windows

In **each window, prover must submit a proof** that random parts of the assigned sealed sectors are present in storage.

If the sealed data is not present, the miner is penalized and their stake is reduced.

It is difficult to do the sealing process again to create another replica within the window period. (Even if it is done, it would be a new replica)

Overview

Introduction

What is Filecoin? - A high level overview

Proof of Replication and Proof of spacetime

Proof of Replication - deep dive

Overview

Replica: Stacked Depth Robust Graph

Poseidon Hash Detour

What to commit and how?

Proof Mechanism

R1CS to Groth16

SnarkPack aggregation

Proof of Replication Deep dive

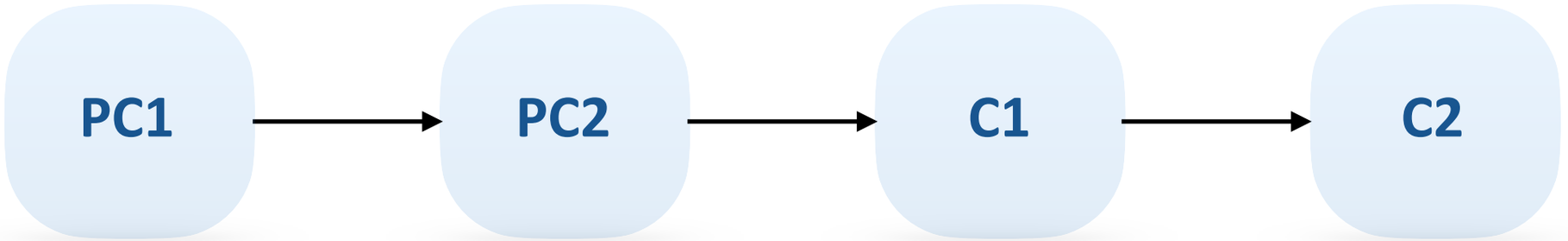
Sector: Unit of storage that Filecoin miner can commit to network

Data from multiple clients — 32 GiB or 64 GiB

Sealed sector: encoded data

Unsealed sector: raw data

Sequential **Processes**



Precommit1 (PC1): Encode and create a **unique** replica of data

Precommit2 (PC2): Generate Merkle tree and Merkle root of the replica

Commit1 (C1) : Choose random subset of Merkle leafs from PC2 and compute path to the root

Commit2 (C2): ZK-Snark proof (Groth16) for the chain.

Proof of Replication: PC1: overview

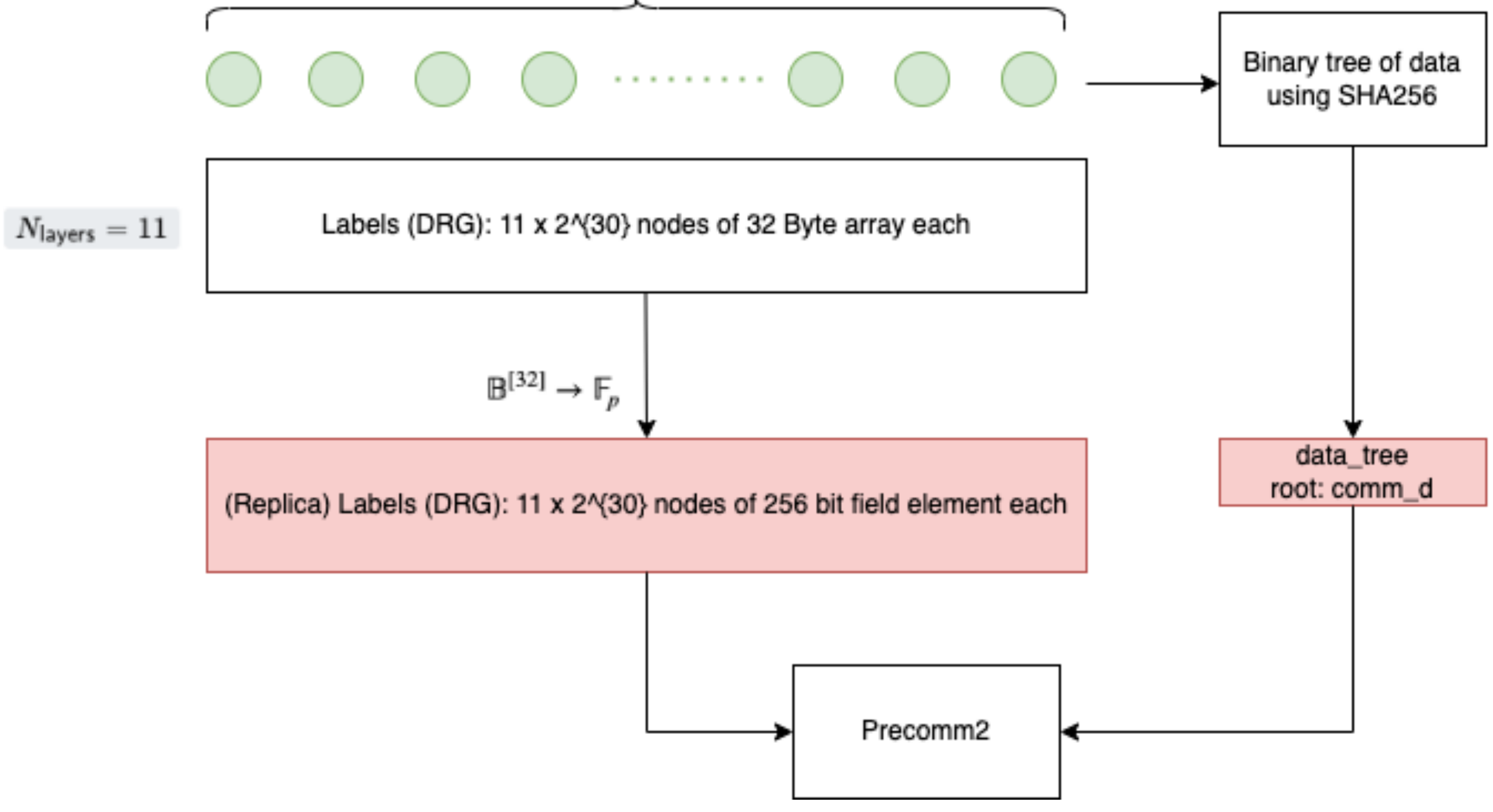
$\ell^{\text{byte}}_{\text{sector}} = 32\ \text{GiB} = 32 * 1024 * 1024 * 1024\ \text{Bytes}$

The byte length of a sector D .

$\ell^{\text{byte}}_{\text{node}} = \ell^{\text{byte}}_{\mathbb{F}_q} = 32\ \text{Bytes}$

32 GiB data sector split into
32 Byte array each

$N_{\text{nodes}} = \ell^{\text{byte}}_{\text{sector}} / \ell^{\text{byte}}_{\text{node}} = 2^{30}\ \text{Nodes}$



PoRep:PC1:Data tree

Sector Construction

A sector D is constructed from Filecoin client data where the aggregating of client data of has been preprocessed/bit-padded such that two zero bits are placed between each distinct 254-bit slice of client data. This padding process results in a sector D such that every 256-bit slice represents a valid 254-bit field element $\mathbb{B}^{[32]}_{\text{safe}}$.

A Merkle tree $\text{TreeD} : \text{BinTree}$ is constructed for sector D whose leaves are the 256-bit slices $D_i : \mathbb{B}^{[32]}_{\text{safe}} \in D$.

$$D_i : \mathbb{B}^{[32]}_{\text{safe}} = D[i * 32 \,..\, (i+1) * 32]$$
$$\text{TreeD} = \text{BinTree::new}([D_0, \ldots, D_{N_{\text{nodes}}-1}])$$
$$\text{CommD} : \mathbb{B}^{[32]}_{\text{safe}} = \text{TreeD.root}$$

Filecoin docs

Each TreeD is constructed over the preprocessed sector data D.

Tree D is a Merkle Tree made literally out of Data Byte arrays using SHA256

CommD is the **public commitment to the Data tree**

PoRep: PC1: SDRG

Main tool: SDRG: Stacked Depth Robust Graph

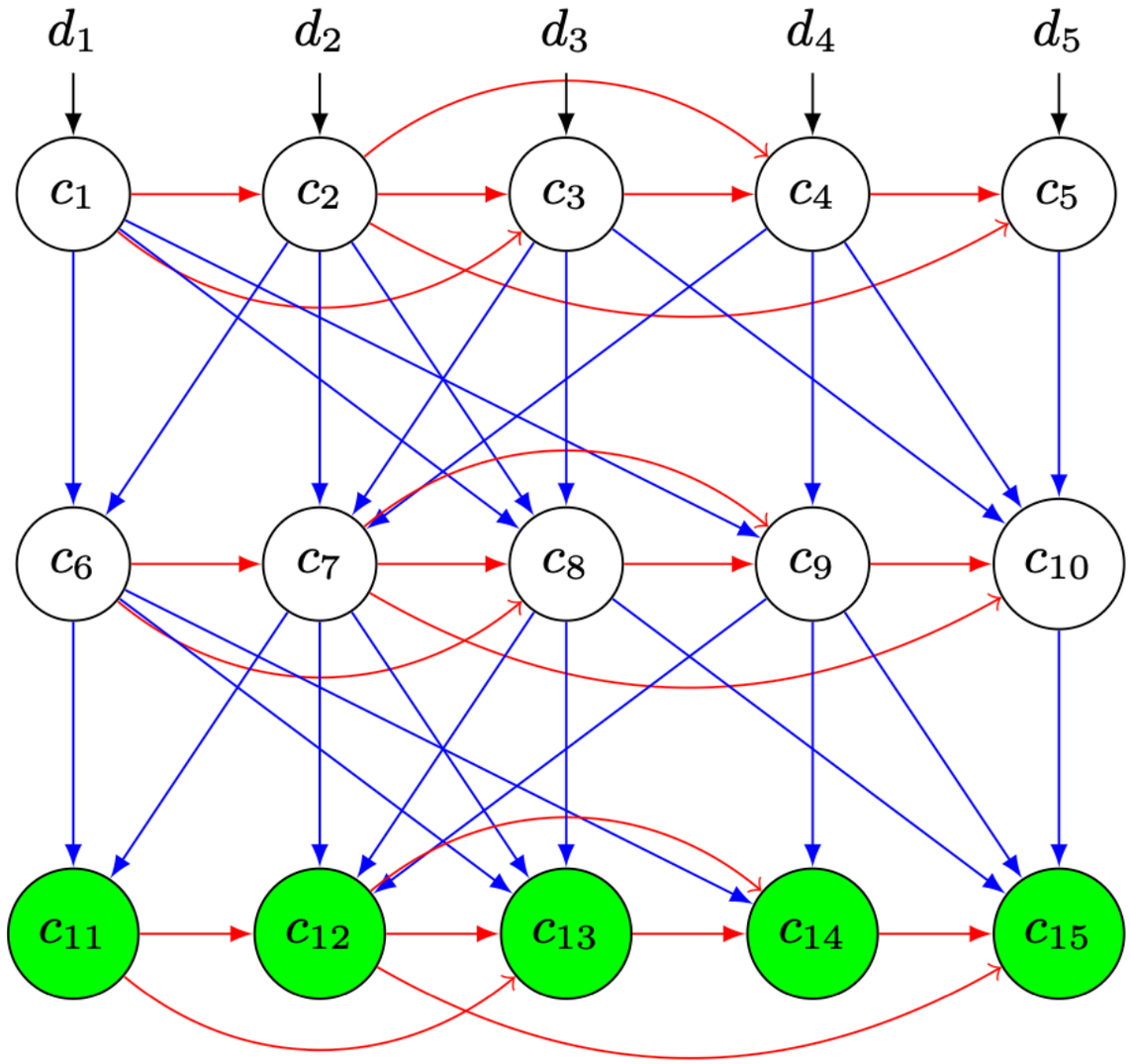
Slow, non-parallelizable, sequential process

DRG: Label C_i on a given node v_i in a row is the output of hash function on labels of all parent nodes of v_i

Computation of labels: SHA256

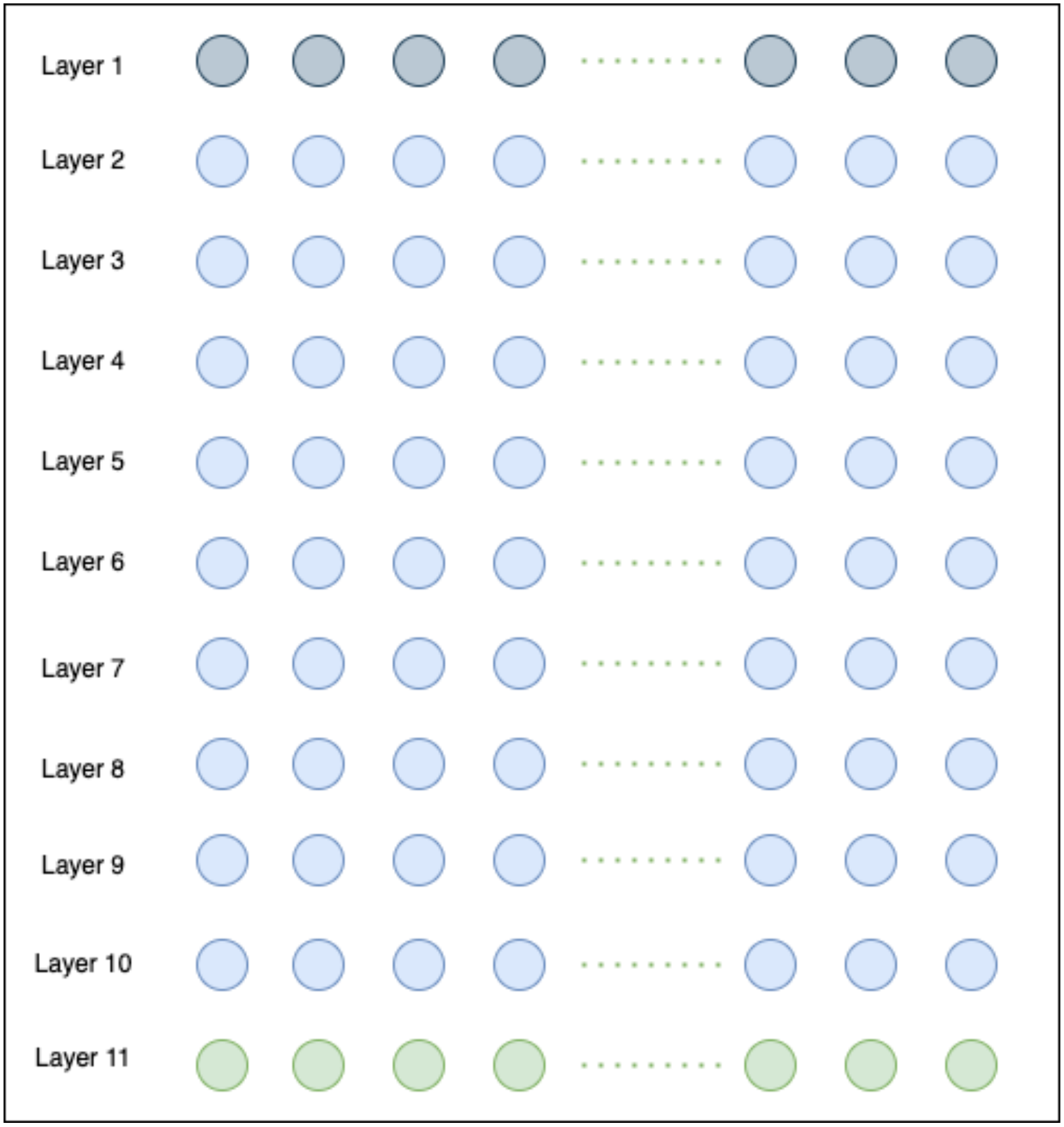
SDRG: Stack several DRG's into layers, nodes on each layer extend to one layer below using a bipartite graph. The labels in green are the final derived labels of the SDRG

Security: ϵ space gap: Even deletion of a small amount of nodes in the earlier layer, will require a cheating prover to rederive the entire graph.



Fisch

PoRep: PC1: output



Output: Replica

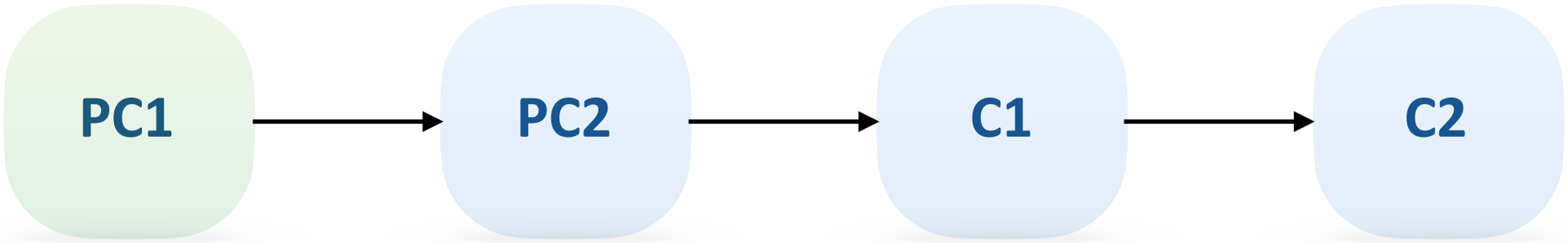
11 layers of labels, each layer is 32GiB

Each label is a 32 Byte array $\mathbb{B}^{[32]}$

The layers of SDRG are **derived sequentially** using SHA256

This referred to as a sealed sector and is huge.. how does one commit to this?

PoRep: PC1 → PC2



All 32 byte array elements are converted to field elements $\mathbb{B}^{[32]} \rightarrow \mathbb{F}_p$

Elements are in scalar field that arises from BLS12-381 subgroup of prime order p (256 bits)

The main computation in PC2 are **Poseidon hashes** and **Merkle trees** on the replica encoded in field elements.

Detour to Poseidon

Overview

Introduction

What is Filecoin? - A high level overview

Proof of Replication and Proof of spacetime

Proof of Replication - deep dive

Overview

Replica: Stacked Depth Robust Graph

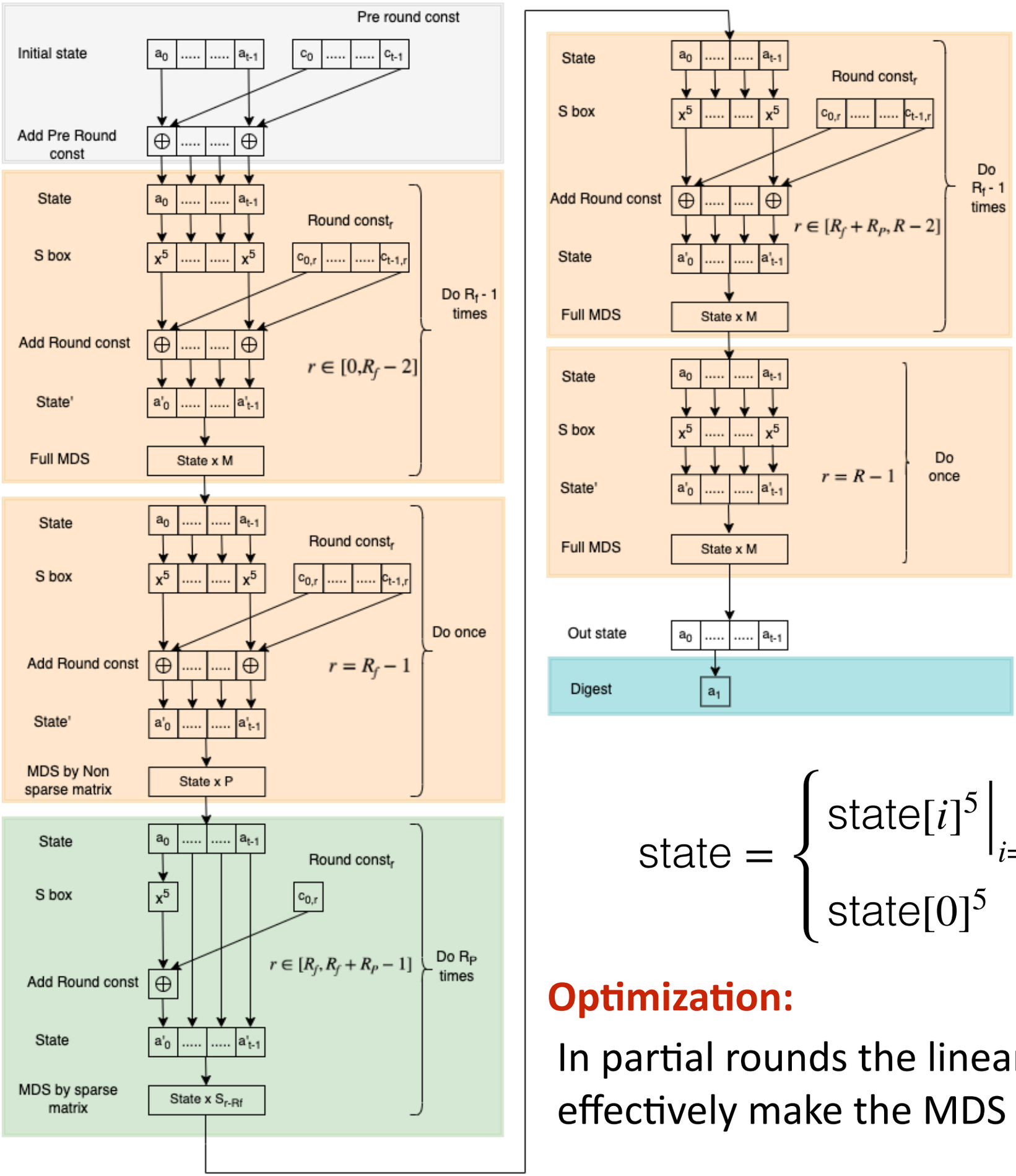
Poseidon Hash Detour

What to commit and how?

Proof Mechanism

R1CS to Groth16

SnarkPack aggregation



Optimized Poseidon

256 bit field elements in scalar field of BLS12-381

Red: Full rounds

Green: Partial rounds

Digest size : 1

128 bit security

$$\text{state} = \begin{cases} \text{state}[i]^5 \big|_{i=0,1,\ldots,t-1} & \text{Full round} \\ \text{state}[0]^5 & \text{Partial round} \end{cases}$$

Optimization:

In partial rounds the linear layers are combined to effectively make the MDS matrix sparse.

Overview

Introduction

- What is Filecoin? - A high level overview
- Proof of Replication and Proof of spacetime

Proof of Replication - deep dive

- Overview
- Replica: Stacked Depth Robust Graph
- Poseidon Hash Detour
- What to commit and how?

Proof Mechanism

- R1CS to Groth16
- SnarkPack aggregation

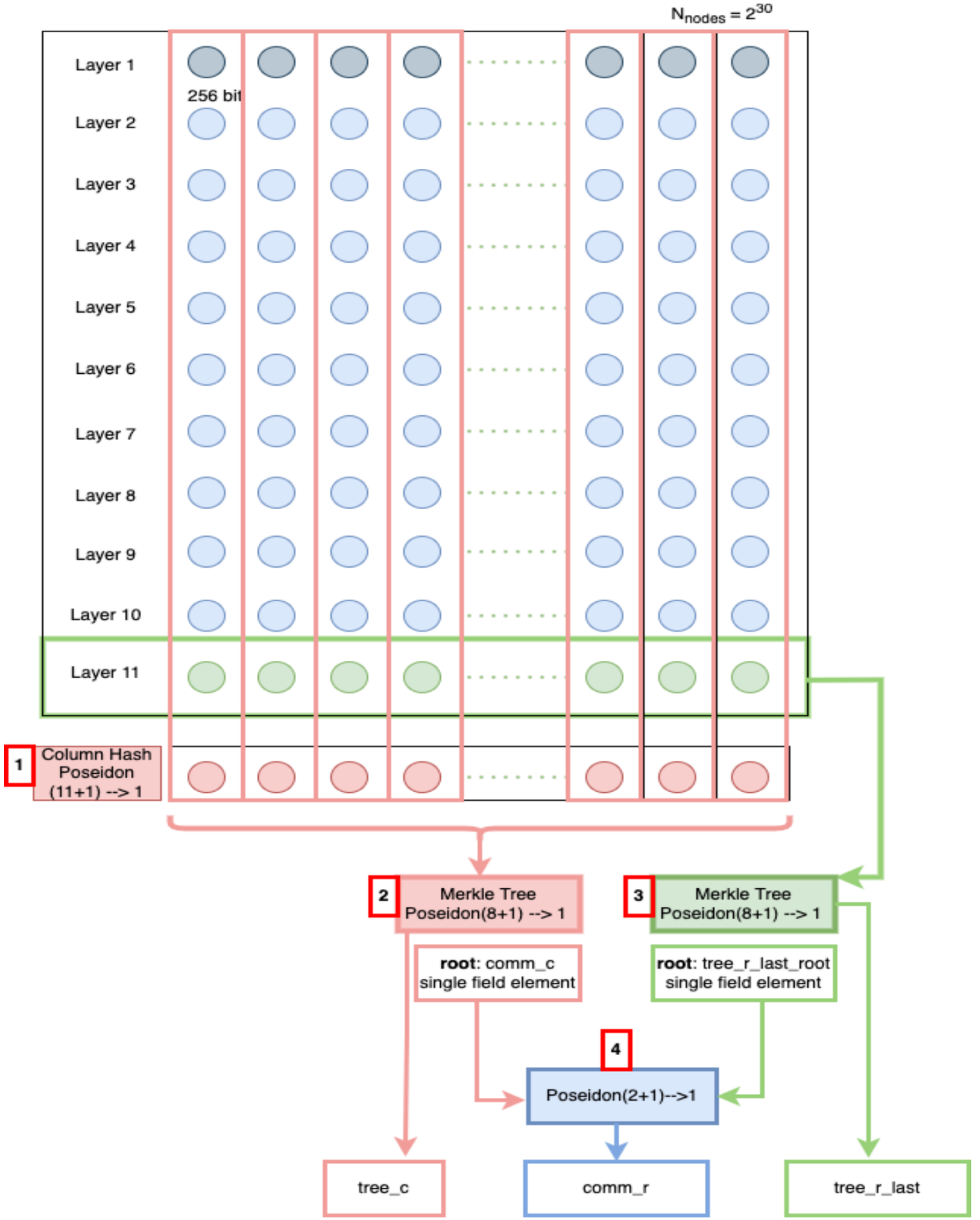
PoRep: PC2

What to commit?

The Last layer is the result of the SDRG - so let us make a Merkle tree out of it and commit the root.

But how do I know you just didn't make the last layer up?

Compute hashes of all column labels and make a Merkle tree out of the result, and commit the root.



PoRep: PC2

$\text{Poseidon}_8(\mathbb{F}_p^{[9]}) \rightarrow \mathbb{F}_p^{[1]}$

Octinary Merkle tree for SDRG labels. **tree_r_last_root**

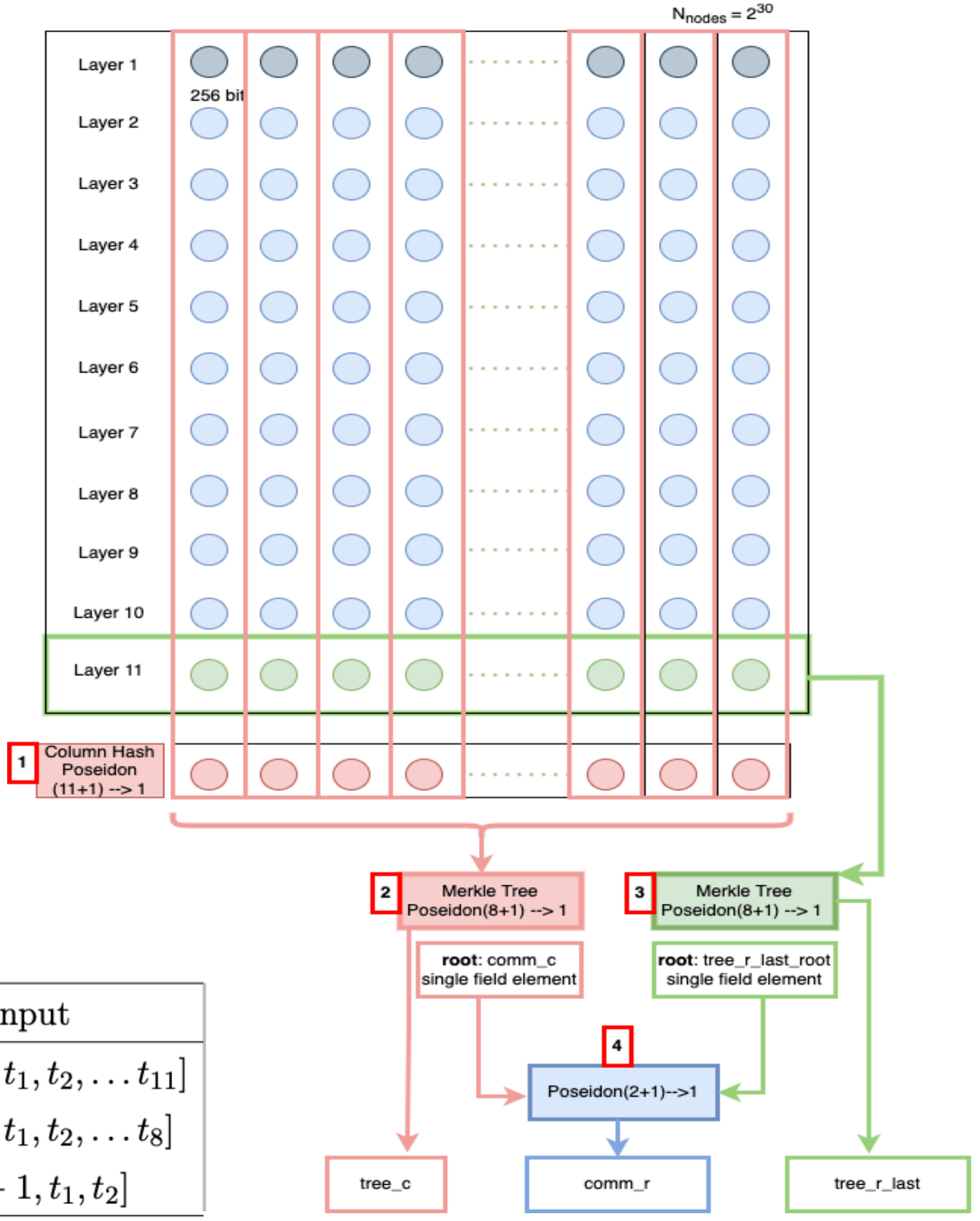
$\text{Poseidon}_{11}(\mathbb{F}_p^{[12]}) \rightarrow \mathbb{F}_p^{[1]}$

for each column in the array

$\text{Poseidon}_8(\mathbb{F}_p^{[9]}) \rightarrow \mathbb{F}_p^{[1]}$

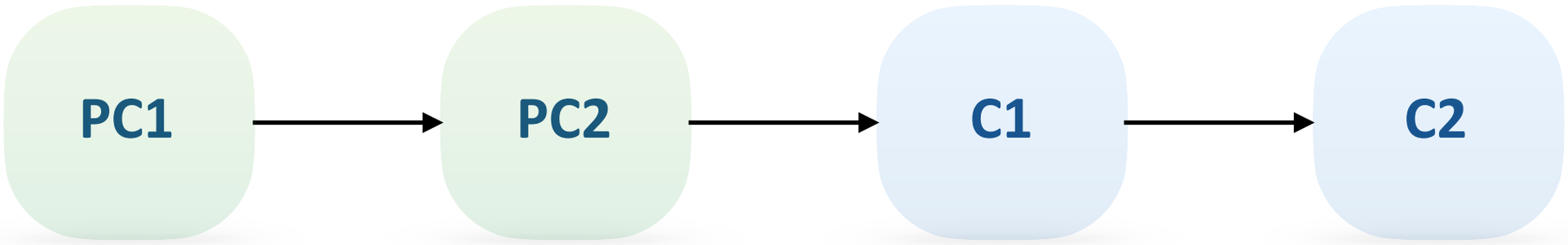
Octinary Merkle tree of column hash result **comm_c**

$\text{Poseidon}_2(\mathbb{F}_p^{[3]}) \rightarrow \mathbb{F}_p^{[1]}$ **comm_r**



Instantiation	t	input
$\texttt{Poseidon}_{11}(\mathbb{F}_p^{[12]}) \rightarrow \mathbb{F}_p^{[1]}$	12	$[2^{11}-1, t_1, t_2, \ldots t_{11}]$
$\texttt{Poseidon}_8(\mathbb{F}_p^{[9]}) \rightarrow \mathbb{F}_p^{[1]}$	9	$[2^8-1, t_1, t_2, \ldots t_8]$
$\texttt{Poseidon}_3(\mathbb{F}_p^{[3]}) \rightarrow \mathbb{F}_p^{[1]}$	3	$[2^2-1, t_1, t_2]$

PoRep: PC2 → C1



At the end of the day: The Merkle roots are publicly committed.

In addition to storing the blown up SDRG, also store two 10 depth Merkle tree labels. The storage overhead per sector is

32 GiB data + 11 x 32 GiB from SDRG + 128 GiB for Merkle trees :D

The C1 process, identifies random nodes for the challenges, in total it picks 176 challenge nodes in the data.

Overview

Introduction

- What is Filecoin? - A high level overview
- Proof of Replication and Proof of spacetime

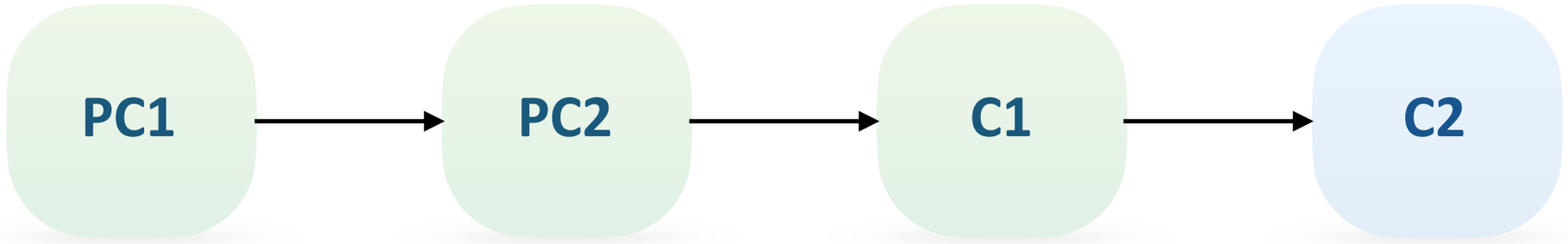
Proof of Replication - deep dive

- Overview
- Replica: Stacked Depth Robust Graph
- **Poseidon Hash Detour**
- What to commit and how?

Proof Mechanism

- R1CS to Groth16
- SnarkPack aggregation

PoRep: C1 → C2



There are four different proofs that make up PoRep

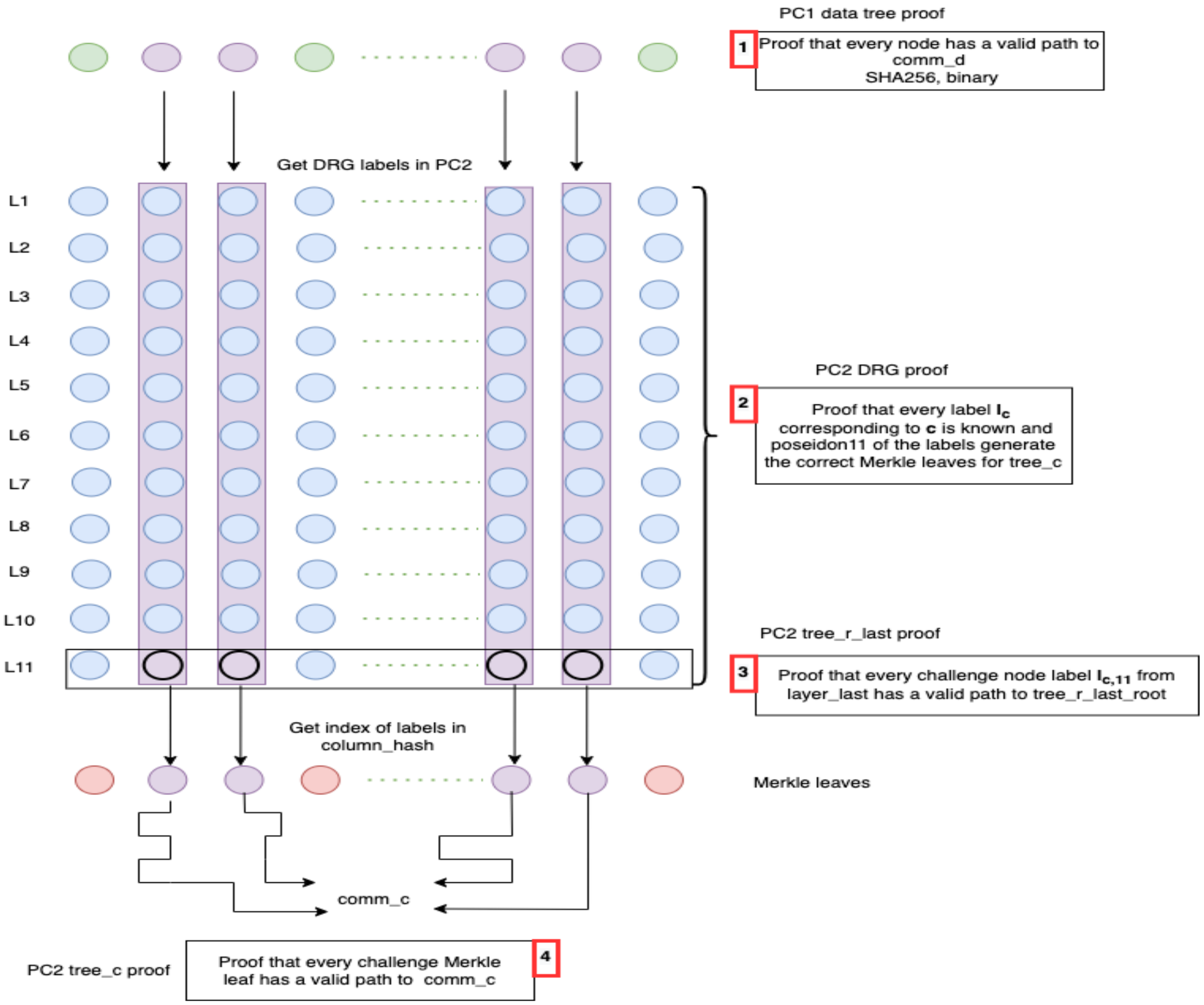
Receive challenge nodes from PC1.

Data: Merkle proof for Data tree

SDRG labels: Merkle Proof for derived Last layer of SDRG

Proof of computation SDRG: Column commitments of SDRG labels and Merkle proof of resulting nodes.

Proof of computation in PC2: All node labels in the Merkle path that follows from the challenge nodes to the public commitment.


PC1 data tree proof
1
Proof that every node has a valid path to comm_d
SHA256, binary
Get DRG labels in PC2
L1
L2
L3
L4
L5
L6
L7
L8
L9
L10
L11
PC2 DRG proof
2
Proof that every label l_c corresponding to c is known and poseidon11 of the labels generate the correct Merkle leaves for tree_c
PC2 tree_r_last proof
3
Proof that every challenge node label $l_{c,11}$ from layer_last has a valid path to tree_r_last_root
Get index of labels in column_hash
Merkle leaves
comm_c
PC2 tree_c proof
Proof that every challenge Merkle leaf has a valid path to comm_c
4

C2: Proving

If every miner posts such **crazy long path element proofs** then the blockchain would run out of space :D

Use a Zk SNARK to compress the path elements - Groth16.

Function: create_porep_circuit(
PorepPartitionProof$_{R,k}$,
k,
ReplicaID,
CommD,
CommC,
CommR,
CommCR,
$\mathcal{R}_{\text{porep\_challenges}} : \mathbb{B}^{[32]}$,
) $\rightarrow$ R1CS

Filecoin docs

Groth16 system has been implemented in multiple frameworks and programming languages and is the most-used SNARK system to this day. To give a sense of proportion, the Filecoin network verifies more than 2 million Groth16 SNARKs per day!

So they want to move to Nova :P

C2: are we done yet?

A single Groth16 proof is

- **Structure of the verification key** vk**:** The verification key is of the form: **BMMTV19**

$$\mathsf{vk} := (p = g^{\rho}, q = h^{\tau}, [s_j = g^{(\beta u_j(x) + \alpha v_j(x) - w_j(x))}]_{j=1}^{2\ell}, d = h^{\delta}) \ .$$

  Here $\rho, \tau, \delta, x \in \mathbb{F}$ are secrets generated (and discarded) during the generation of the proving and verification keys, and $u_i(X), v_i(X), w_i(X)$ are public polynomials that together define the QAP representation of the computation being checked.
- **Structure of the proof** π**:** The proof π is of the form $\pi := (A, B, C) \in \mathbb{G}_1 \times \mathbb{G}_2 \times \mathbb{G}_1$is of the form:
- **Verifier's checks:** On input a verification key vk, an NP instance $\mathbb{x} := (a_1, \ldots, a_m) \in \mathbb{F}^m$, and a proof $\pi = (A, B, C)$, the verifier checks that $e(A, B) = e(p, q) \cdot e(\prod_{j=1}^{2\ell} s_{1,j}^{a_j}, h) \cdot e(C, d)$.

The trusted set up is of length 2^{27} and the **PoRep SNARK is split into 10 smaller SNARKS**

Given 10 instances: **checking each SNARK individually takes 30 pairings and** $10 * l$ **multiexponentiations** which is expensive

Proof Aggregation

Given n Groth16 proof instances $\pi_i = (A_i, B_i, C_i) \quad D_i \equiv \mathsf{VK}$

$A_i, C_i \in \mathbb{G}_1$
$B_i, D_i \in \mathbb{G}_2$
$Y_i \in \mathbb{G}_T$

Instead of checking individually $e(A_i, B_i) = Y_i \cdot e(C_i, D_i)$

The aggregation will instead check a single randomized equation:

$$\prod_{i=0}^{n-1} e(A_i, B_i)^{r^i} = \prod_{i=0}^{n-1} Y_i^{r^i} \cdot e\Big(\prod_{i=0}^{n-1} C_i^{r^i}, D\Big).$$

We denote by $Y'_{prod} := \prod_{i=0}^{n-1} Y_i^{r^i}$ so this can be rewritten as:

SnarkPack

$$Z_{AB} = Y'_{prod} \cdot e(Z_C, D), \quad \text{where } Z_{AB} := \prod_{i=0}^{n-1} e(A_i, B_i)^{r^i} \text{ and } \quad Z_C := \prod_{i=0}^{n-1} C_i^{r^i}.$$

What is left after checking that this unified equation holds is to verify that the elements Z_{AB}, Z_C are consistent with the initial proof triples in the sense that they compute the required inner product. This is done by applying an argument that proves two different inner pairing product relations:

- TIPP: the target inner pairing product takes some initial committed vectors $\mathbf{A} \in \mathbb{G}_1, \mathbf{B} \in \mathbb{G}_2$ and shows that $Z_{AB} = \prod_{i=0}^{n-1} e(A_i, B_i)$;
- MIPP: the multi-exponentiation inner product takes a committed vector $\mathbf{C} \in \mathbb{G}_1$ and a vector $\mathbf{r} \in \mathbb{Z}_p$ and shows that $Z_C = \prod_{i=0}^{n-1} C_i^{r^i}$.


Thats all Folks!