Session 8

Navigator sessions

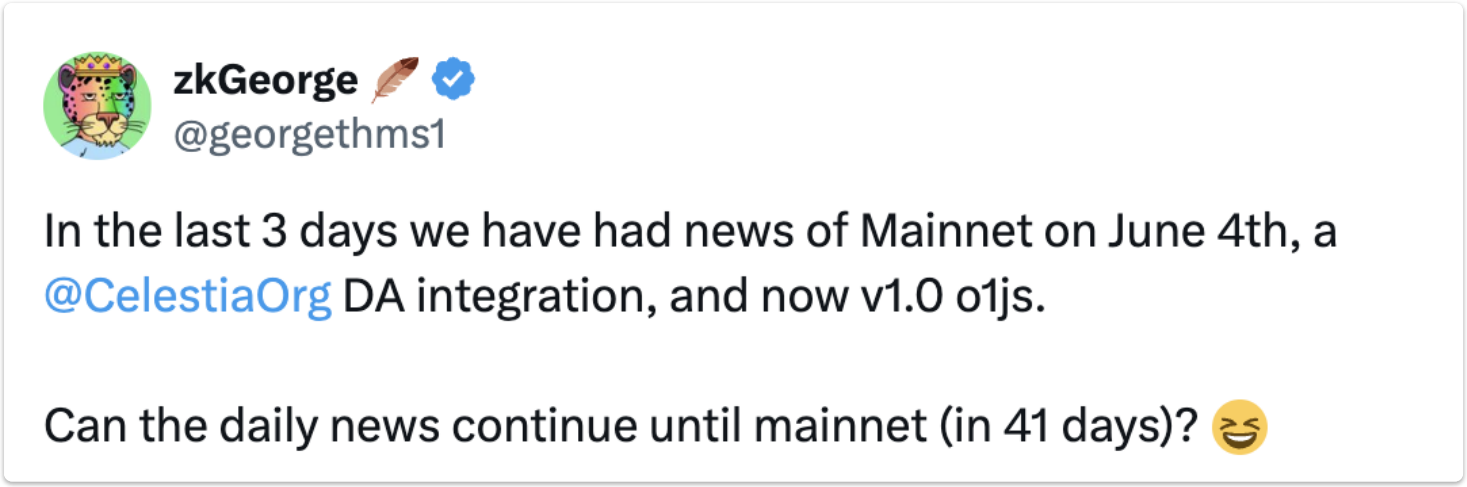
Date	Topics	
15/12/23	o1js review	
19/1/24	Development workflow, design approaches, techniques and useful patterns	
26/1/24	Recursion	
9/2/24	Application storage solutions	
1/3/24	Utilising decentralisation	
15/3/24	zkOracles and decentralised exchanges	
5/4/24	Ensuring security	
26/4/24	Review Session	

Today's topics

- What's new
- o1js changes
- Data availability
- Foreign Field Arithmetic
- Development tips
- An ultimate zk platform architecture
- Proof Markets

- Recursion revisited
- General Blockchain directions
- Privacy and regulation
- Research areas
- Course review
- Next Steps

What's new



Mainnet upgrade

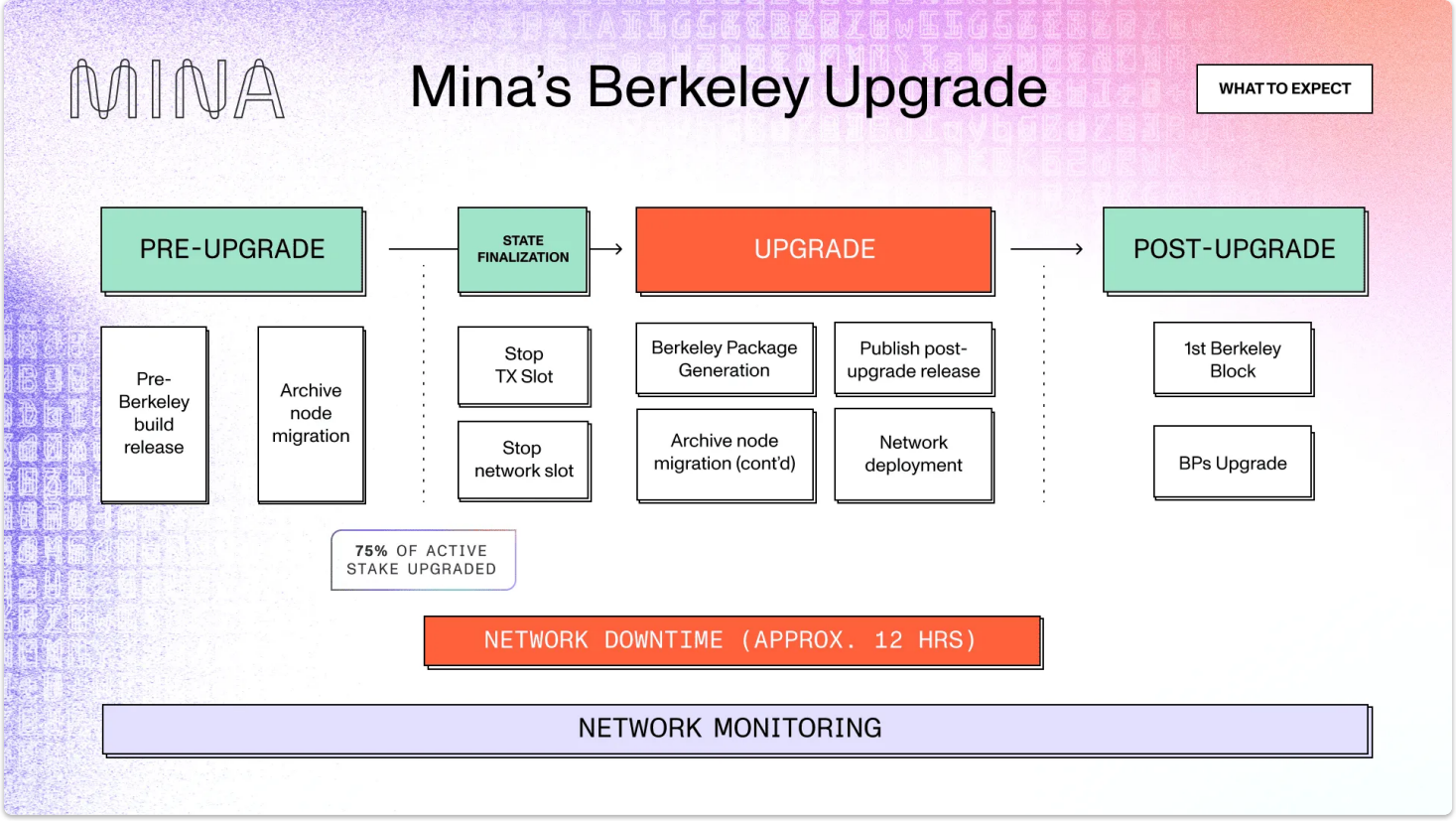
Mina Foundation and o1Labs together are excited to announce that Mina's Berkeley upgrade is scheduled for June 4, 2024

See post post

This major mainnet upgrade will bring three key features to the protocol, as voted on by the community:

- Easier zkApp programmability (MIP 4)
- Kimchi, a more powerful proof system (MIP 3)
- Removal of Supercharged Rewards (MIP 1)

What to expect - Docs



Countdown

https://zkok.io/

Zeko MVP

Zeko MVP is live for testing , see announcement

zkHack - Krakow

o1js changes

See change log

See announcement

Summary

- Removed Poseidon hashing bottleneck
- Enhanced performance on Apple silicon
- Introduced asynchronous circuits for faster, simpler zkApps
- Added recursion to reduce() method for processing limitless actions
- Removed top-level await which simplifies integration with UI frameworks

---

Data availability

Mina - Celestia Integration

See announcement
See blog

The data availability problem is relevant to increasing the size of blocks, sharding, and rollups, which aim to improve a blockchain's throughput. Data availability proofs are a solution to this problem, allowing clients to check with a high probability that all the data for a block has been published by only downloading a very small piece of that block.

This is achieved through the use of erasure codes, which allow the recovery of the entire block even if a significant portion of it is missing. By implementing data availability proofs, blockchain networks can ensure that all data is available, thereby enhancing the security and decentralisation of the network.

The integration of Celestia's modular DA layer with Mina's ZK native blockchain will enable

higher throughput for zkApps, particularly those that require large-scale off-chain data.

Also see this proposal from onurinanc who along with succinctlabs built the solution.
See Repo
Blobstream X's core contract is `BlobstreamX`, which stores commitments to ranges of data roots from Celestia blocks. Users can query for the validity of a data root of a specific block height via `verifyAttestation`, which proves that the data root is a leaf in the Merkle tree for the block range the specific block height is in.

Geometry research will verify Blobstream X proofs produced by succinctlabs, which are Groth16 proofs of Celestia's consensus, giving Mina zkApps access to Celestia block headers.

Data Availability overview

See EF Docs
In order to re create the state, transaction data is needed, the data availability question is where this data is stored and how to make sure it is available to the participants in the system.

	Validity Proofs		Fault Proofs
Data On-Chain	Volition	ZK-Rollup	Optimistic Rollup
Data Off-Chain		Validium	Plasma

STARKWARE

Validium solutions

Validium solutions use off-chain data availability and computation.
Validity proofs are used to show the correctness of execution.

Foreign Field Arithmetic

See Mina docs
repo PR

We do arithmetic based on the Field class that is native to our proof system. Other cryptographic systems however will have different finite fields so we cannot directly interact with them.
Foreign fields allows us to interface with other systems so that we can for example verify signatures created using a finite field different to the one we use in Mina.
This ability has been used to implement Keccak and ECDSA. See article

We create a Foreign Field class using the `createForeignField` class factory (taking the field modulus as parameter). The modulus needn't be a prime number and the maximum modulus value we can use is 259, this allows for most elliptic curves commonly used.

Taking the examples from the documentation :

```
import { createForeignField } from 'o1js';

class Field17 extends
createForeignField(17n) {}
```

Here we are creating a Field based on a finite field that is mod 17.

```
let x = Field17.from(16);
x.assertEquals(-1); // 16 = -1 (mod 17)
x.mul(x).assertEquals(1); // 16 * 16 = 15
* 17 + 1 = 1 (mod 17)
```

A useful size would have a modulus of 256 to match the basic datatype used in the EVM.

```
class UInt256 extends
createForeignField(1n << 256n) {}

// and now you can do arithmetic modulo
2^256!
let a = UInt256.from(1n << 255n);
let b = UInt256.from((1n << 255n) + 7n);
a.add(b).assertEquals(7);
```

The base type supplied is `ForeignField`, it supports the following operations

```
x.add(x); // addition
x.sub(2); // subtraction
x.neg(); // negation
x.mul(3); // multiplication
x.div(x); // division
x.inv(); // inverse
```

plus the following provable methods

```
x.assertEquals(y); // assert x == y
x.assertLessThan(2); // assert x < 2

let bits = x.toBits(); // convert to a
`Bool` array of size log2(modulus);
Field17.fromBits(bits); // convert back
```

and non provable methods

```
let y = SmallField.from(5n); // convert
from bigint or number
y.toBigInt() === 5n; // convert to bigint
```

Types of ForeignField

There are 3 types

- unreduced
- almost reduced and
- canonical

  You can get these via static methods :

```
let x = new Field17.Unreduced(0);
let y = new Field17.AlmostReduced(0);
let z = new Field17.Canonical(0);
```

Unreduced

In short, unreduced means that a value can be larger than the modulus, most arithmetic operations return unreduced fields.

```
import assert from 'assert';

let z = x.add(x);
assert(z instanceof Field17.Unreduced);
```

Restrictions

Unreduced fields can be added and subtracted, but not multiplied or divided.

Almost Reduced

These allow multiplication and division.

```
let zz = zAlmost.mul(zAlmost); //
zAlmost.mul() is defined

// but .mul() returns an unreduced field
again:
assert(zz instanceof
SmallField.Unreduced);

// zAlmost.inv() is defined, and returns
an almost reduced field:
assert(zAlmost.inv() instanceof
SmallField.AlmostReduced);
```

It is possible to enforce this type, for example if taking as an input, in the following way

```
class AlmostField17 extends
Field17.AlmostReduced {}

class MyContract extends SmartContract {
  @state(AlmostField17.provable) x =
State<AlmostField17>();

  @method async myMethod(y: AlmostField17)
{
    let x = y.mul(2);
    this.x.set(x.assertAlmostReduced());
```

```
  }
}
```

Canonical

This is the most restricted type, they are guaranteed to be smaller than the modulus. The only operation possible with a canonical field is the boolean equality check

```
let xCanonical = x.assertCanonical();
let yCanonical = y.assertCanonical();
let isEqual =
xCanonical.equals(yCanonical);
```

When you create fields from constants, they always get fully reduced. The type signature of `ForeignField.from()` reflects this and returns a canonical field:

```
let constant = Field17.from(16);
assert(constant instanceof
Field17.Canonical);

// these also work, because `from()` takes
the input mod 17:
Field17.from(100000000n) satisfies
CanonicalForeignField;
```

```
Field17.from(-1) satisfies
CanonicalForeignField;
```

Conversion

You can convert any field to canonical by calling `.assertCanonical()`:

```
let zCanonical = z.assertCanonical();
assert(zCanonical instanceof
Field17.Canonical);
```

A cheap way is to show it is equal to a constant

```
let zCanonical = z.assertEquals(3);
assert(zCanonical instanceof
Field17.Canonical);
```

For further tips when using ForeignField see Docs

BigInt implementation

See discussion

We only have built-in BigInts up to 256 bits right now, using ForeignField.

Useful tips

Problems / solutions from Discord

Question : Non inclusion proof / nullifiers, see post

Answer : MerkleMap is the best currently available way I know of to prove this. It's not ideal and it would be great to find something that is more "succinct" and therefore scalable.

Questions : Is it possible to prove contract has been deployed ? See post

Answer : Try to think in terms of preconditions and account updates instead of deploying of a smart contract. You can't prove that smart contract has been deployed by someone since it's not even defined what "deployed by someone" means here. To make sure an account is under specific conditions, set a precondition. To make sure the account contains specific verification key, prove an account update with the corresponding proving key.

Question : How to validate a public key post

Answer : `PublicKey.fromBase58()` will throw an exception if not valid.

Question :

Is it possible to prove array inclusion,

For example in an array as follows:

```
const hashes = new
Array(ATTEST_LENGTH).fill(Field(0));
```

See post

Answer :

```
assert(attestations.hashes.map(h =>
h.equals(attestation)).reduce(Bool.or));
```

Protokit Tips and tricks

See article

See the zkNoid Repo for details on how to start protokit in a docker container.

Ultimate zk platform

Talk at ZK Accelerate from Brandon about the ultimate zk platform

Properties

- Easy composability
- Privacy
- Decentralisation
- Computational Efficiency
- Scaling

Composability

- via recursion
- or contract / token composition
- using familiar languages
- with ability to add zkVM

Important that we prove client side
Race condition reconciliation is a problem

Need to system to run on everyday hardware
To help this we can break transactions down and then we need sequencing, which needs to be decentralised.
This needs to be verifiable, so what we want is

the whole system to be a proof with incremental verification. We can farm this out to marketplaces.

Decentralisation

- of participants
- of sequencing
- of proving
- of end user usage

Proof Market

A collaboration between Taceo and =nil;Foundation
Proof Market as the place where proof requesters and proof producers meet and create a free, open, and self-sustaining market.

Demo

There are three primary roles (parties) in the Proof Market:

- **Proof requesters** are applications that require zero-knowledge proofs, and make requests for them on the Proof Market.
- **Proof producers** are owners of computational infrastructure, who generate proofs for the requests made by proof requesters.
- **Circuit developers** make zero-knowledge circuits, that are used to generate requests and subsequent proofs.

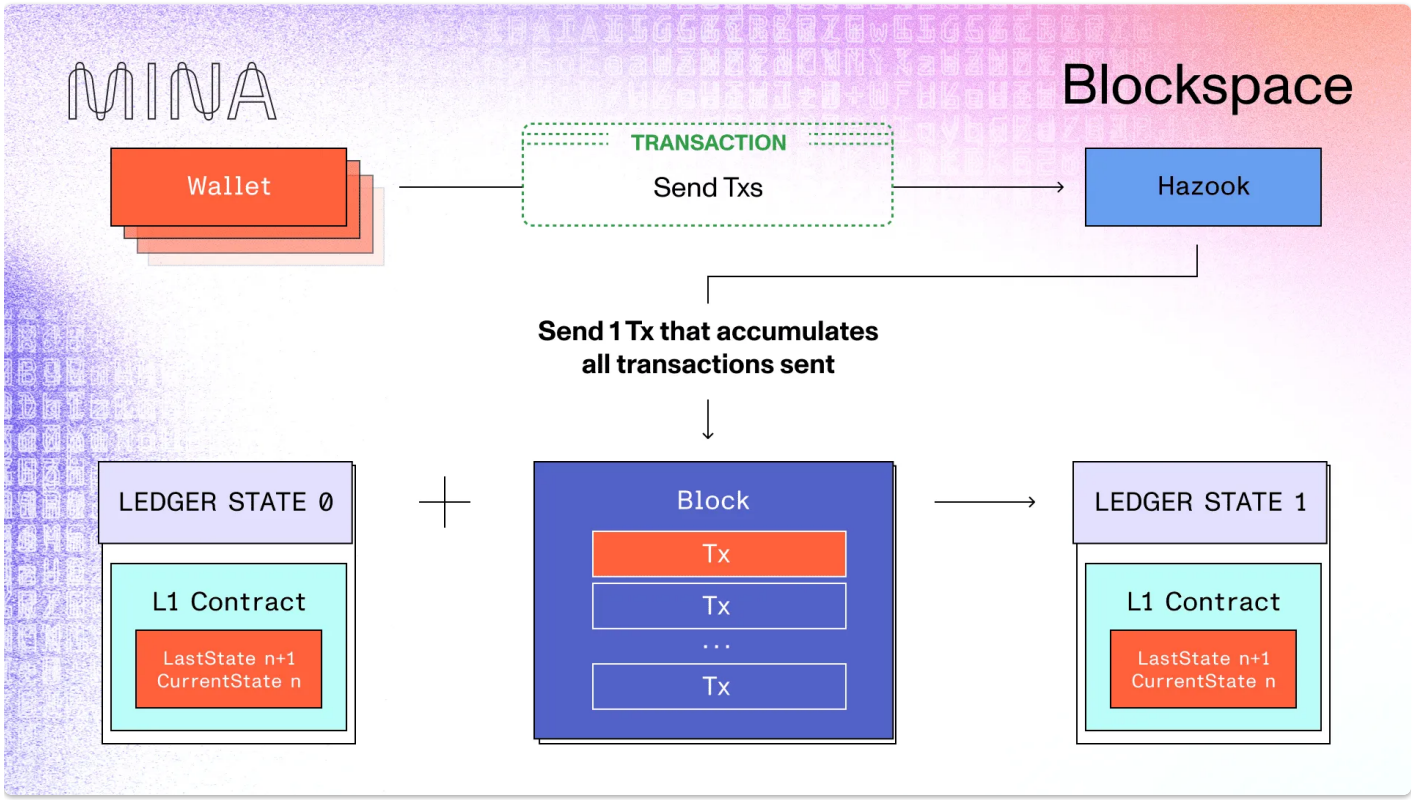
The Proof Market toolchain has tools for proof requesters and producers. To learn more about

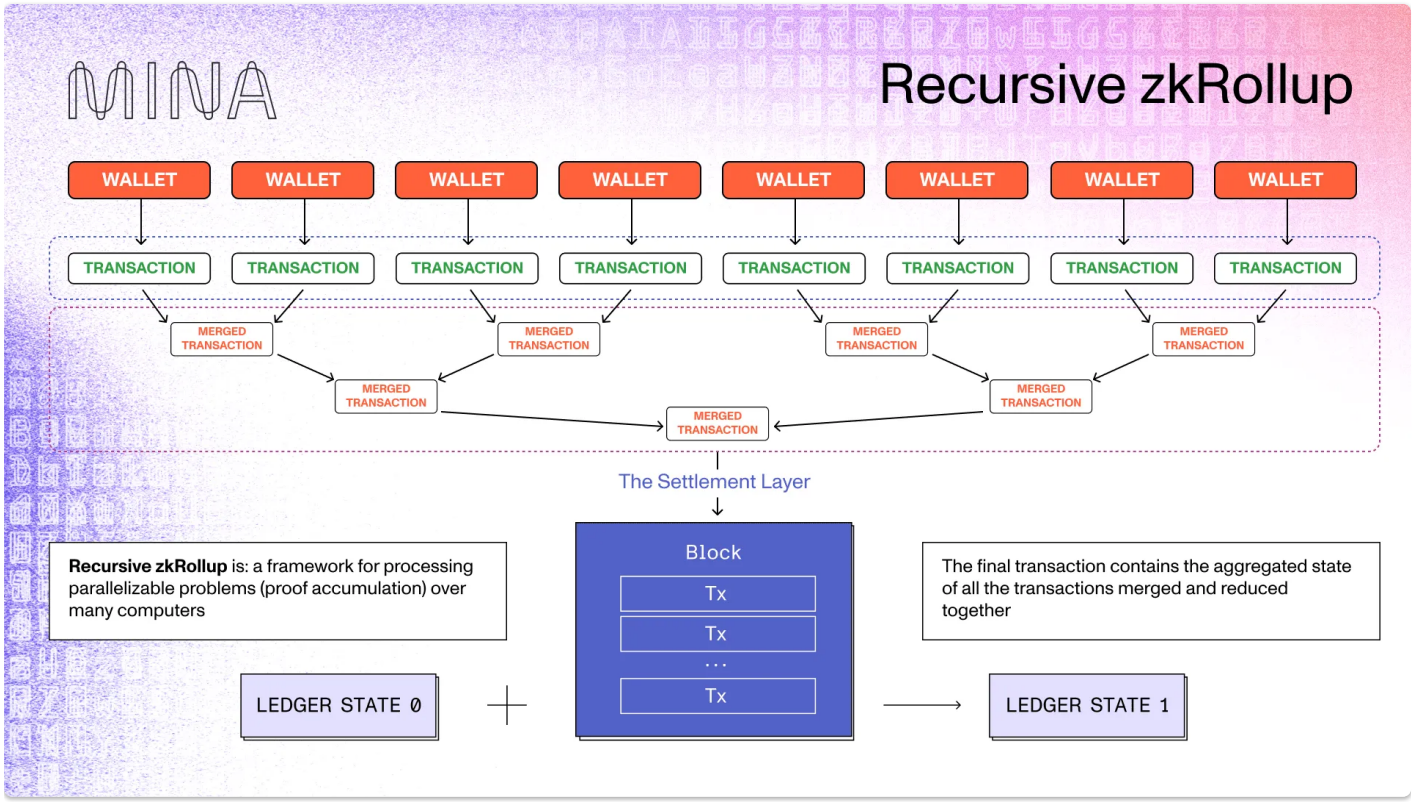
> the Proof Market and these roles, read the Proof Market documentation.
>
> If you're interested in circuit development, check out the zkLLVM compiler and zkLLVM template project.
>
> If you wish to provide proofs, a toolchain is available.

Recursion continued

[Blog]

Hazook [Repo]





For example, with Hazook running proofs for 4000 user transactions should cost about

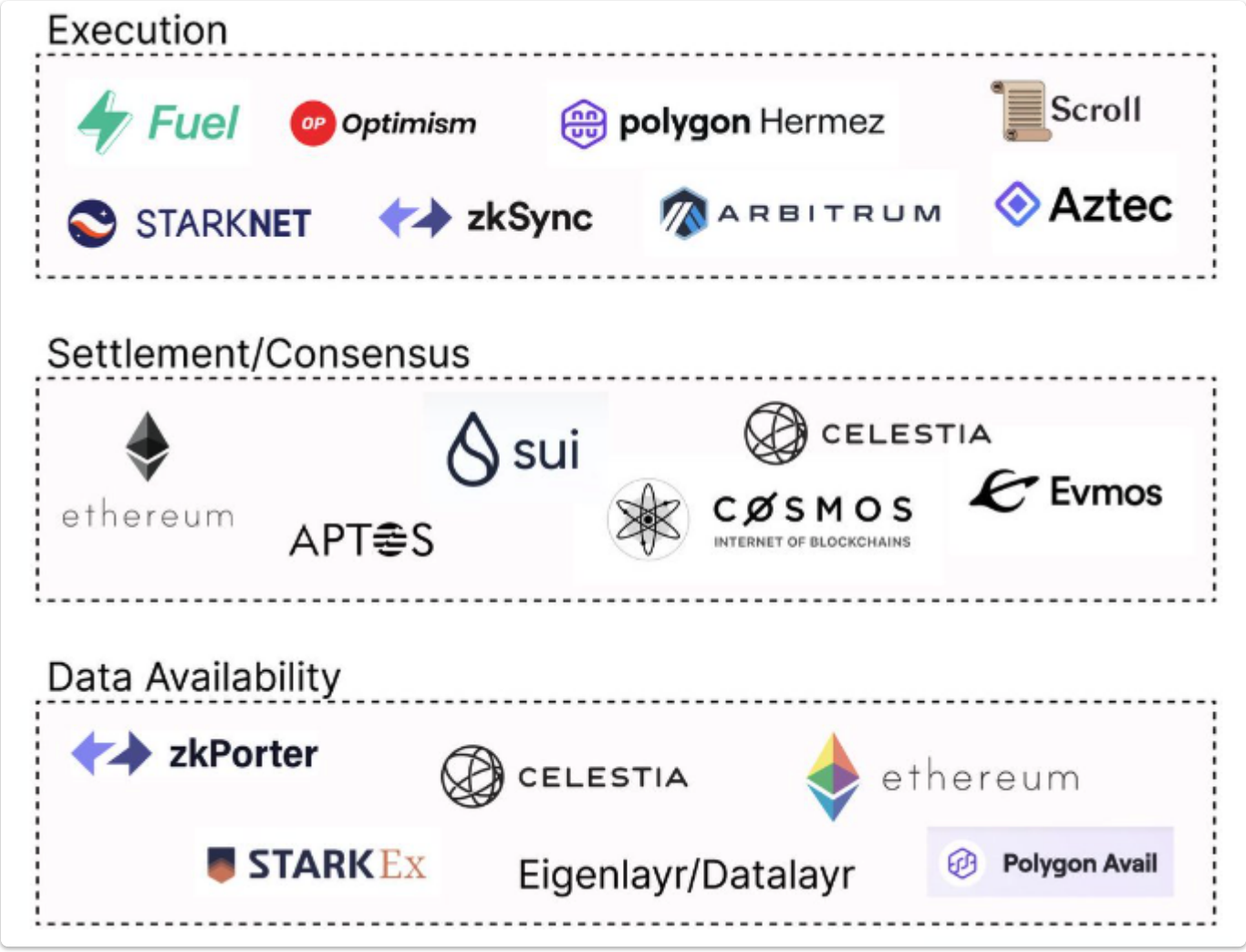
$245, or$ .06 per transaction.

One powerful use case is governance. For example, no matter how long a voting period may be, most users wait until the final hours before casting their votes. Implementations like Hazook are built to scale with the help of on-demand cloud services in order to meet the required computing power needed to meet a surge in user demand.

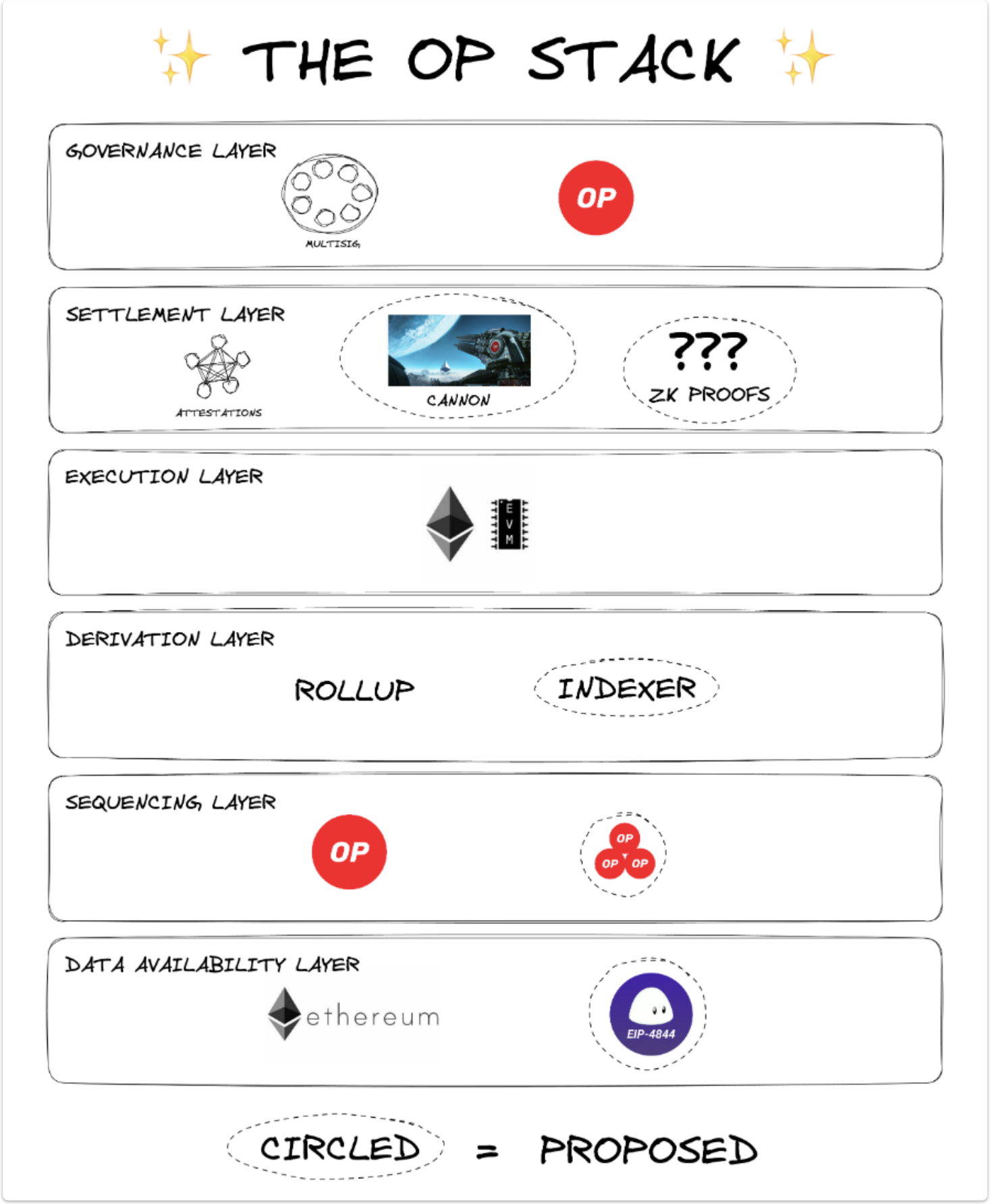
Difference with Protokit :
Protokit doesnt require cloud infrastructure.

General Blockchain directions

Modular Blockchains

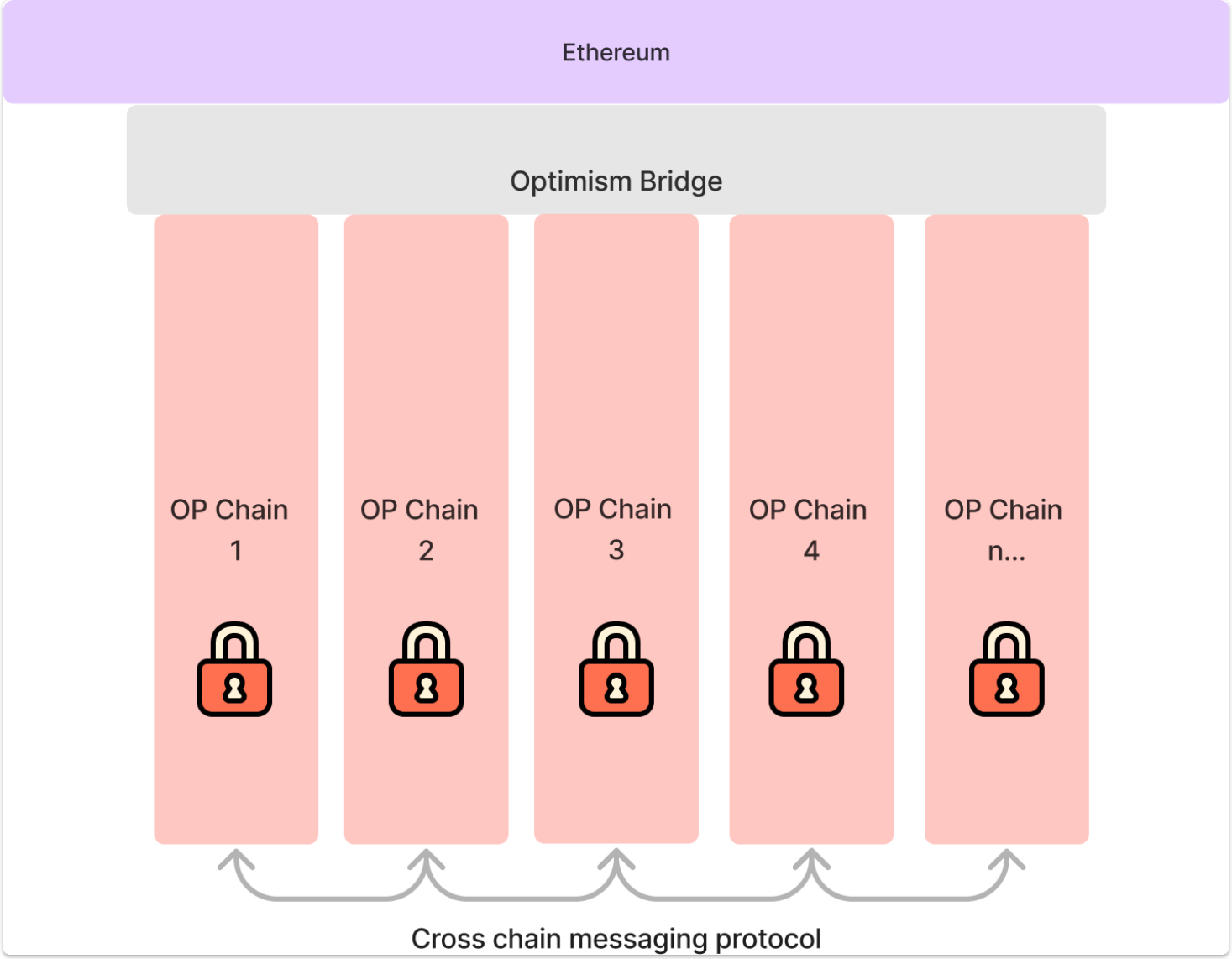


Rollup stacks



Optimism Superchain

This is a proposed network of L2s that share security, communication layers, and a common development stack - the OP Stack.



App chains

We have seen how we can use Protokit to build app chains, in addition to the L3 app chains we also see similar architectures with

Polkadot parachains :

These connect to Polkadot's main relay chain for security and are customisable for particular applications.

Cosmos Zones

These are blockchains connected to Cosmos Hub and use the IBC protocol for interoperability,

allowing them to interact and transfer assets.

Avalanche subnets

These are a set of validators working together to buid consensus on a set of blockchains, these chains can be application specific.

Regulation versus privacy

Governments / regulators are increasingly clamping down on privacy preserving (financial) applications.

- Tornado Cash
- Samourai Wallet developers arrested
- SEC / Consensys arguing about Metamask, whether it acts as a broker.
- Legislation in the UK makes seizure of crypto assets easier.

Geometry Research

Future Directions

See Post

The work done to integrate Mina-Celestia, involved verification of Groth16 proofs, which means execution of pairings inside zkApps. This opens up possibilities of

- zkBridges based on Groth16 or Plonk proofs
- Large rollups, utilizing a higher constraint limit
- BLS signatures, for consensus verification
- Interoperability with existing apps - such as private and collusion-resistant voting and hidden information games.
  The work and library for this will be open sourced.

Recent Papers

Reducing Foreign Field Arithmetic

See Paper

"We construct techniques for offloading foreign arithmetic from a zero-knowledge circuit including (i) equality of discrete logarithms across different groups; (ii) scalar multiplication without requiring elliptic curve operations; (iii) proving knowledge of an AES encryption."

Folding Schemes

Lattice Fold

See Paper

"it is as performant as Hypernova, while providing post-quantum security."

Security

Sok of vulnerabilities in SNARKS

See Paper

Course Review

Session 1 - Introduction

- Decentralisation
- Basic and advanced data types
- Permissions
- Bitwise operations
- Composability
- Action / Reducer

Session 2 - Mina Development

- Development workflow and tools
- Techniques and useful patterns

Session 3 - Recursion

- Recursion use case
- Recursion
- Custom Tokens
- ECDSA example

Session 4 - Application storage solutions

- Simple techniques
- Data structures
- Other techniques
- Concurrency Problems

    - Action Recucer
    - Alternatives
- The bigger picture
- Modular Blockchains
- L2 architecture
- L2 Projects on Mina
    - Anomix
    - Zeko
    - Protokit part 1

Session 5 - Utilising decentralisation

- Protokit continued
- Decentralisation

Session 6 - zkOracles and decentralised exchanges

- zkOracles
- Decentralised Exchanges

Session 7 - Ensuring security

- Security
- Identity
- zkML

Session 8 - Review session

- o1js changes

- Data availability
- Foreign Field Arithmetic
- Development tips
- An ultimate zk platform architecture
- Proof Markets
- Recursion revisited
- General Blockchain directions
- Privacy and regulation
- Research areas
- Course review
- Next Steps

Next Steps

Mina Community

See Docs

Recent additions

Geometry Research

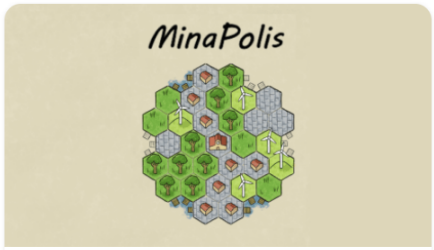
Geometry Research empowers protocols using cryptography

MiDNA

MiDNA (Mina for DNA Analysis) is a service that utilizes Mina and ZKApp. DNA sequence data are analyzed by the servi...

zkCloudWorker

A fast cloud proving service for zkApps.



MINAPolis

Site Links

MRLN

Rate Limiting Nullifier on Mina.

Sliced

Approach to Cooperative Group Purchases.

DRMina

Minager League

Trident

Grants programs

See article

In addition to the Navigator program, there are zkIgnite cohorts

Core grants

See RFPs , for example

- RFP for tokenomics in Mina

  See RFP

Above all

Celebrate the mainnet upgrade

Keep Building, you are pioneers

Keep Decentralising our (financial) systems