Lambdaclass Ethereum Consensus Client

An Elixir Ethereum Consensus Client

Tomás Arjovsky
Martín Paulucci
Tomás Grüner
Paul-Henry Kajfasz

Section 1

Introduction

Who are we?

fellowship program

Tomás Arjovsky
@arkenan

Paul-Henry Kajfasz
@phklive

Tomás Grüner
@megaredhand

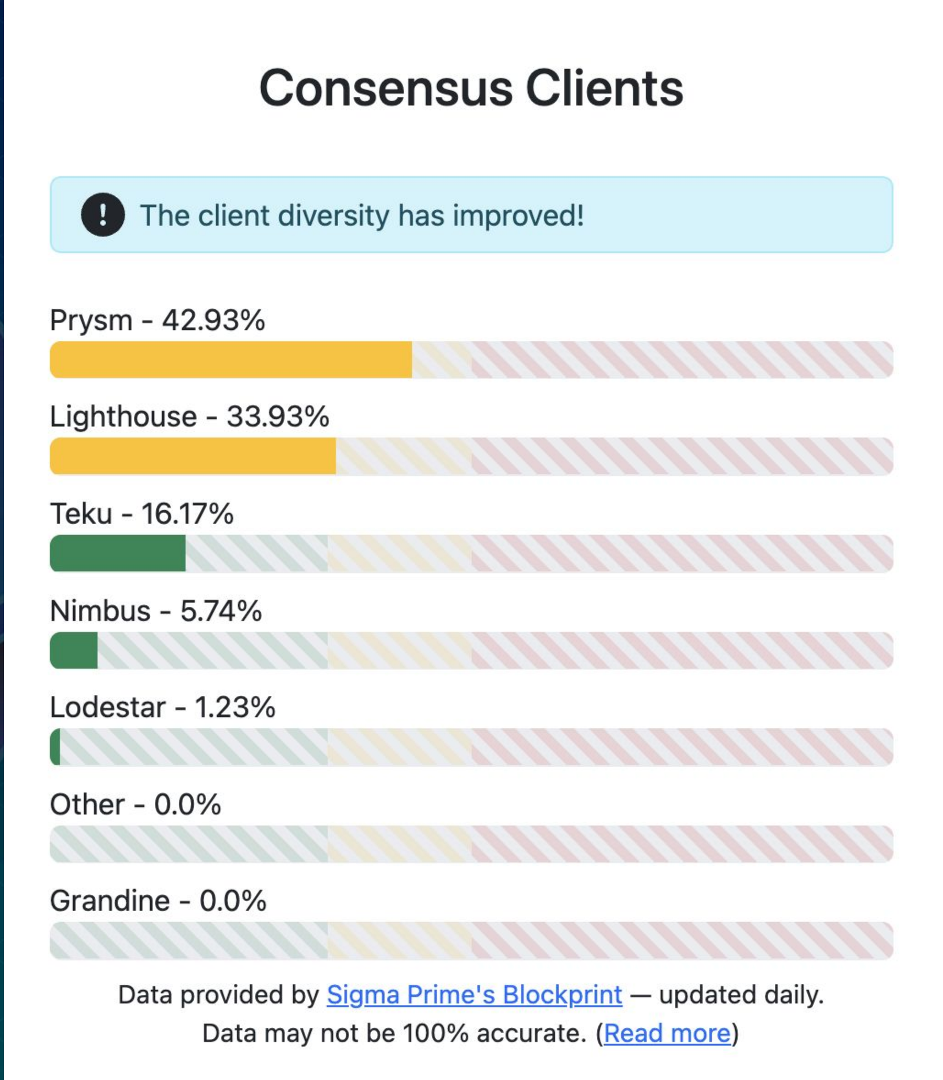
Martín Paulucci
@mpaulucci

Section 2

Vision

Client diversity

https://clientdiversity.org/



Consensus Clients

The client diversity has improved!

Prysm - 42.93%

Lighthouse - 33.93%

Teku - 16.17%

Nimbus - 5.74%

Lodestar - 1.23%

Other - 0.0%

Grandine - 0.0%

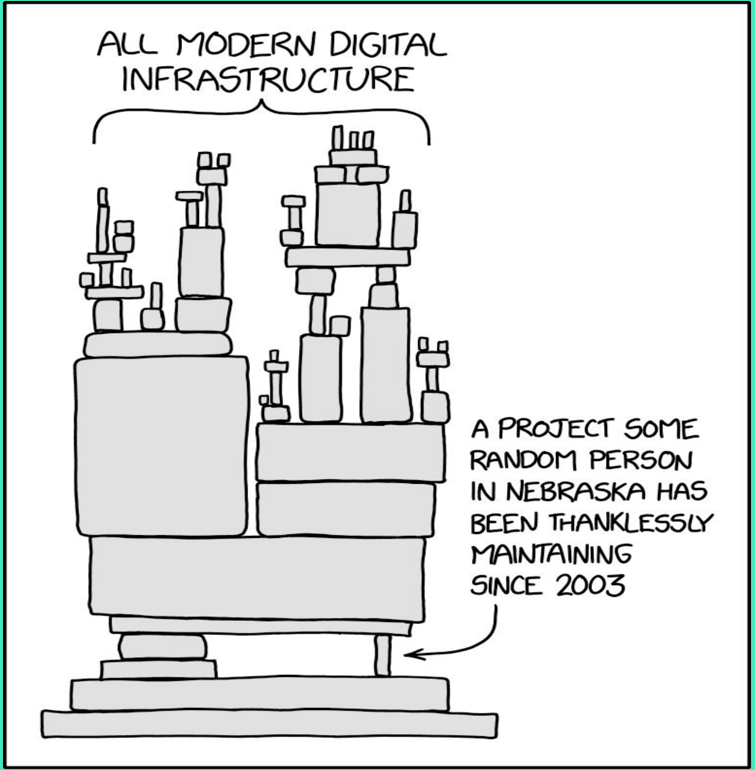
Data provided by Sigma Prime's Blockprint — updated daily.
Data may not be 100% accurate. (Read more)

Why is client diversity so important?

Why is client diversity so important?

As summarized by Danny Ryan, in a client with:

- <66.6%, a fault/bug in a single client cannot be finalized.
- <50%, a fault/bug in a single client's fork choice cannot dominate the head of the chain.
- <33.3%, a fault/bug in a single client cannot disrupt finality.


ALL MODERN DIGITAL INFRASTRUCTURE
A PROJECT SOME RANDOM PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

https://xkcd.com/2347/

Ways our client improves diversity

Functional language

Elixir is a programming language following the functional paradigm. It has a unique concurrency model and was designed to support distributed programming

Geographical diversity

The core team is mostly from South America, and most of the external contributions have been from countries of the global south

First post-Merge client

We're starting a client from the latest fork (Capella) without adding support for previous forks (at least initially). This should make the codebase simpler.

Why Choose Elixir for Ethereum?

Concurrency and Distribution

Elixir excels in handling numerous concurrent processes efficiently with its lightweight process model and message passing mechanism. This design is ideal for distributed systems, allowing for scalable and robust handling of many simultaneous operations.

Fault tolerance & Reliability

Elixir adopts a "let it crash" philosophy, where systems are designed to self-recover from failures. Supervision trees monitor and restart failed processes, ensuring high availability and resilience, crucial for systems where continuous operation is essential.

Functional Programming

Elixir is a functional programming language, which means it emphasizes functions and immutable data. This paradigm is beneficial for concurrent programming as it reduces side effects and makes the code more predictable.

Hot reloading & Observability

Unique among many programming languages, Elixir allows for code updates in live systems without downtime. This feature is invaluable for maintaining long-running, high-availability systems, enabling seamless updates and bug fixes in real-time. The VM also enables internal observability.

Section 3

What have we built so far?

Building Blocks

- SSZ (de)serialization
- Snappy (de)compression
- BLS signing and verifying

Networking

Req/Resp

Gossip

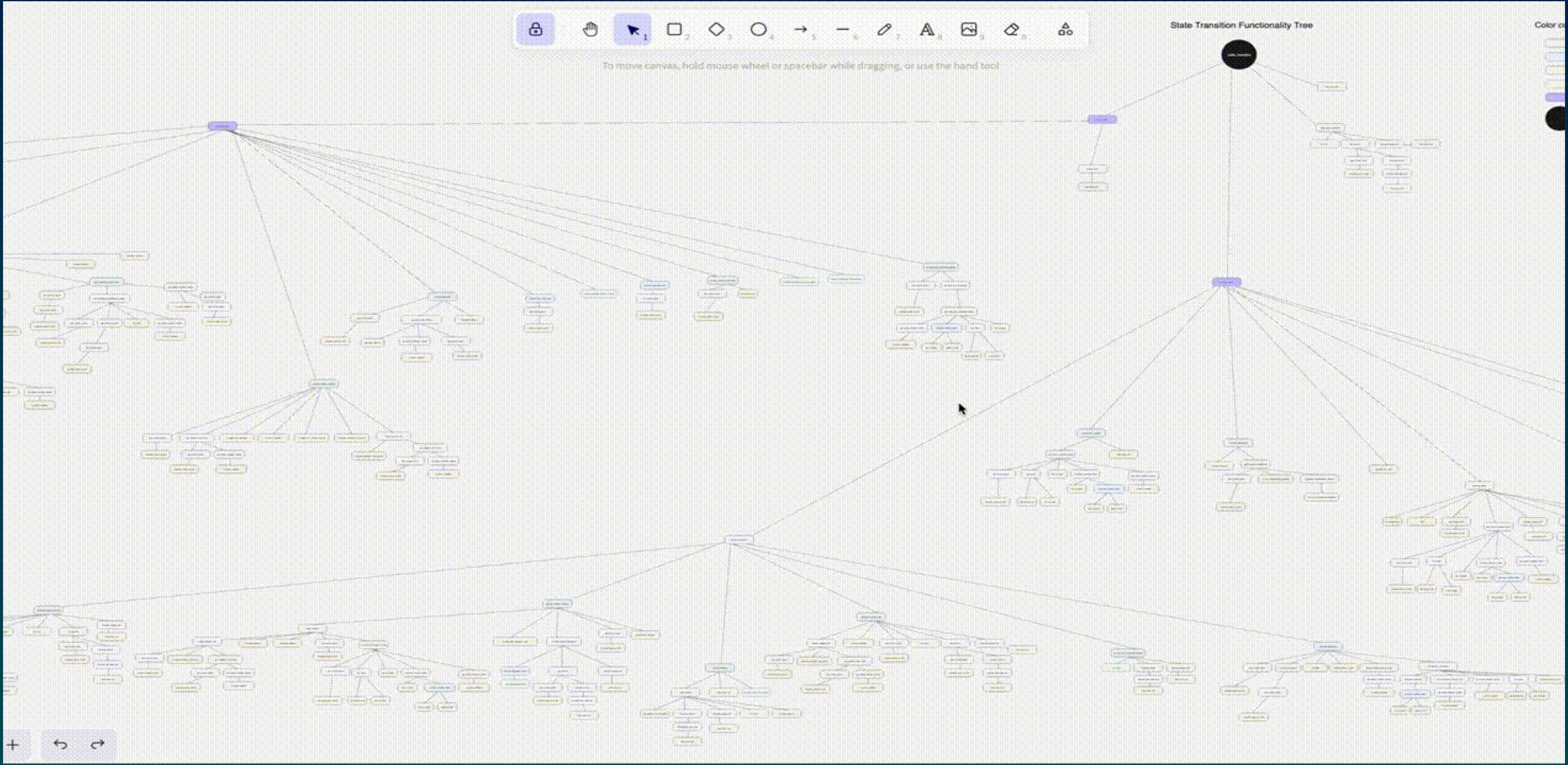
Discovery

- Peer discovery
- Topic subscription
- Serve Incoming Requests
- Send Outgoing Requests

State transition function

- Block processing
- Slot processing
- Epoch processing

State Transition


State Transition Functionality Tree
To move canvas, hold mouse wheel or spacebar while dragging, or use the hand tool
Color co

Syncing

- Checkpoint sync from URL
- Missing blocks fetching
- Attestations and blocks gossip hearing

Fork Choice

- Head calculation
- External input handlers:
  - On tick
  - On block
  - On attestation
  - On attester slashing

- consensus-spec-tests
- Continuous Integration
- Grafana & Prometheus
- Easy repo setup

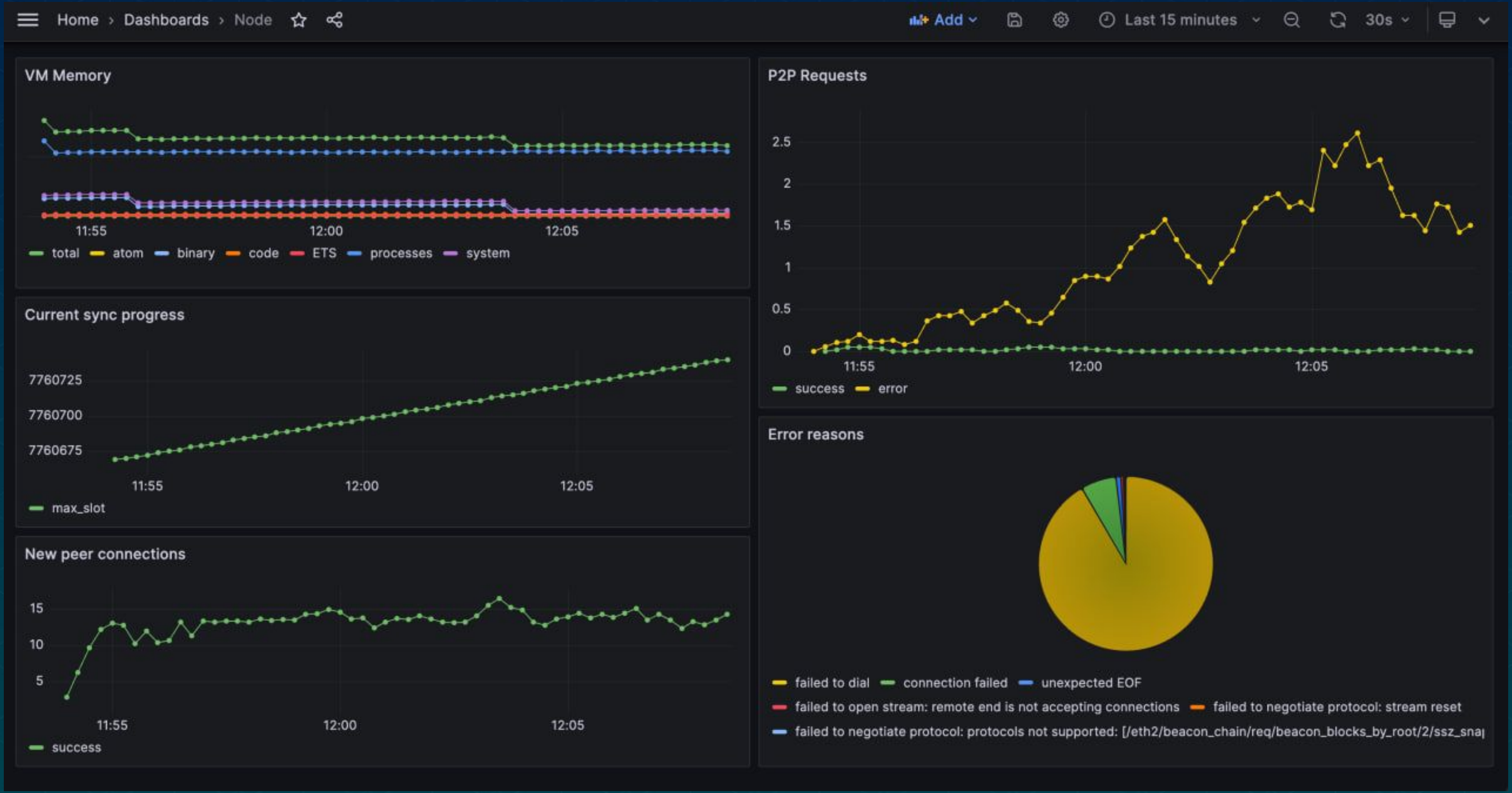
Others

CI/CD

DevX

Observability

Observability



failed to dial · connection failed · unexpected EOF
failed to open stream: remote end is not accepting connections · failed to negotiate protocol: stream reset
failed to negotiate protocol: protocols not supported: [/eth2/beacon_chain/req/beacon_blocks_by_root/2/ssz_snap

Where are we at?

This we have ✅

- 80% of the state transition
- Checkpoint sync
- Networking, decompressing, deserialization.
- Fork choice store. LMD Ghost. Casper FFG.
- On_tick, on_block, on_attestation handlers.
- Prometheus + Grafana for observability

This we don’t ❌

- Beacon and Engine APIs
- Validators Integrated/tested.
- Native Elixir libp2p, SSZ, and Snappy implementations.
- Local testnet for automated integration testing.
- End-to-end testing.

Community

Community

- We decided to build the client in the open and created a telegram group from day one. Today it has 400+ members

- LambdaClass decided to run a 4 month Bootcamp to reward our early contributors and encourage even more participation

- Overall, we got 13 contributors to the codebase and active participation in the Telegram group

Contributors

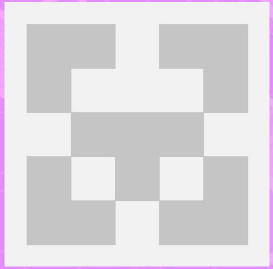
Godspower Eze

Akash

Fernando Ledesma

Mete Karasakal

Ayush


Berwingan

Seungmin Jeon

Naman Garg

Section 4

Next steps

Future roadmap

Q4 2023

Beacon Node fully compatible with Capella spec

Q1 2024

Validator implementation and deployment to testnets

Q2 2024

Optimizations and getting ready for mainnet

Section 5

Conclusion

Special thanks to ❤

- Ben Edgington (Eth2 book)
- Vitalik & EF Consensus-specs and consensus-spec-tests
- Core consensus devs answering questions and giving good advice
- Josh D. & Mario H. for the EPF

Let’s keep Ethereum diverse & resilient

Thank you!

Tomás Arjovsky
Martin Paulucci
Tomás Grüner
Paul-Henry Kajfasz

Our repo

LA
@LAMBDACONSENSUS

@lambdaclass @phklive @arkenan @megaredhand @mpaulucci