Mana

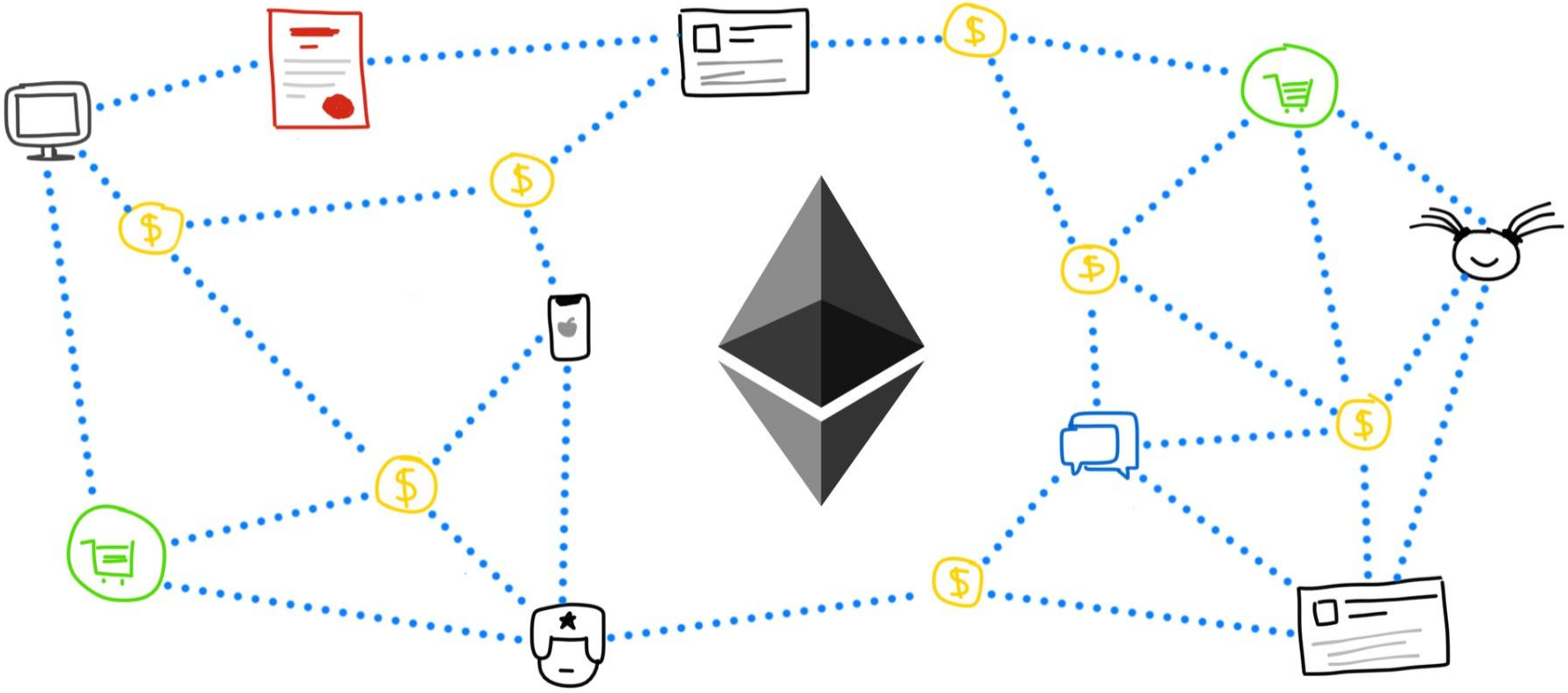
Ethereum client in Elixir

Ayrat Badykov


WTF IS THIS!
WTF WAS THAT?!

Acknowledgements


SAY BLOCKCHAIN
ONE MORE TIME...

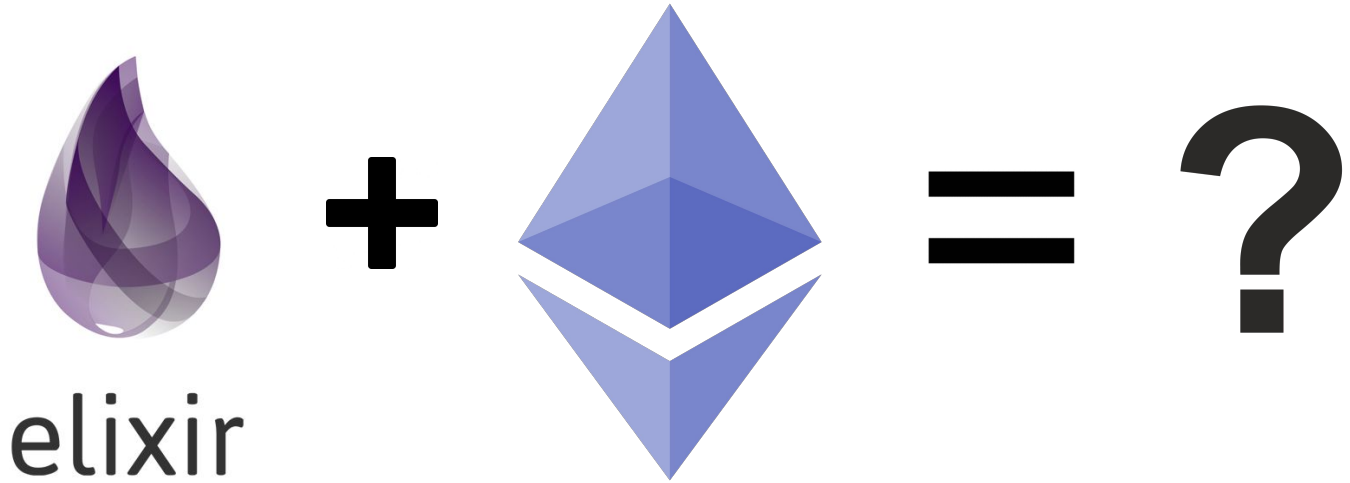


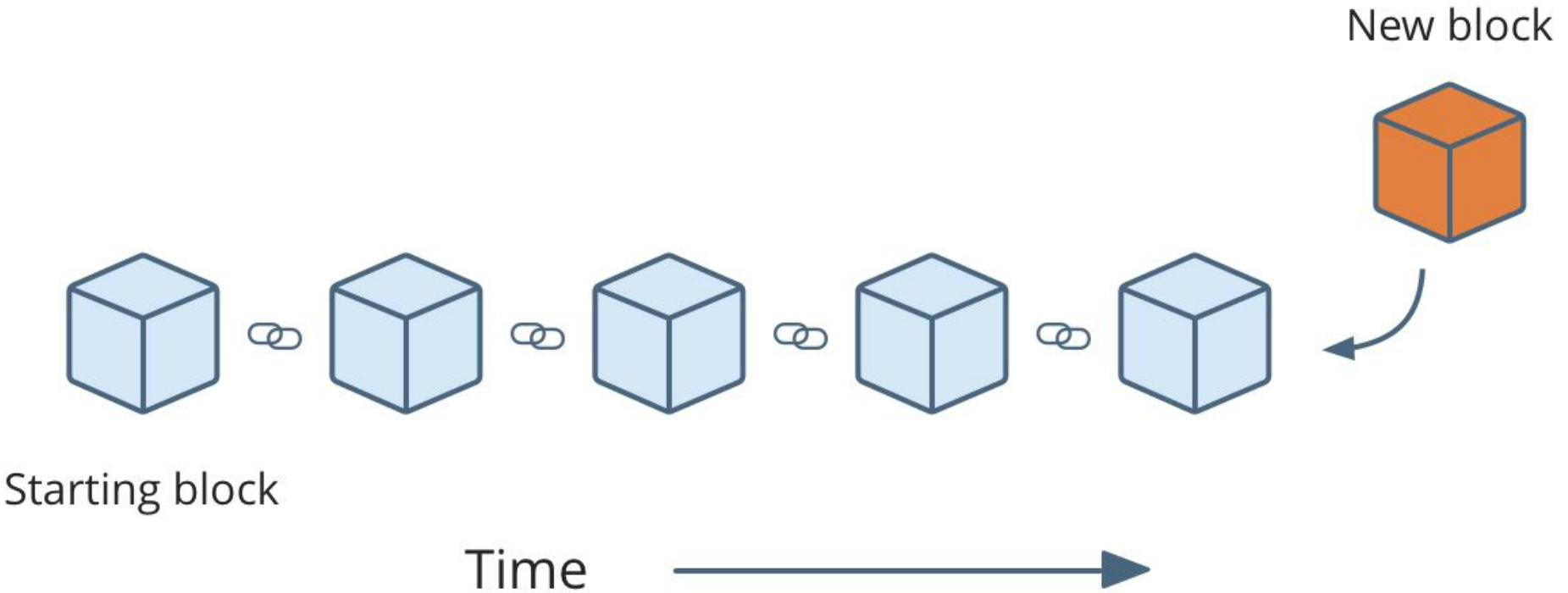
Ethereum clients

- Parity (Rust)
- Geth (Go)
- Pyethereum (Python)

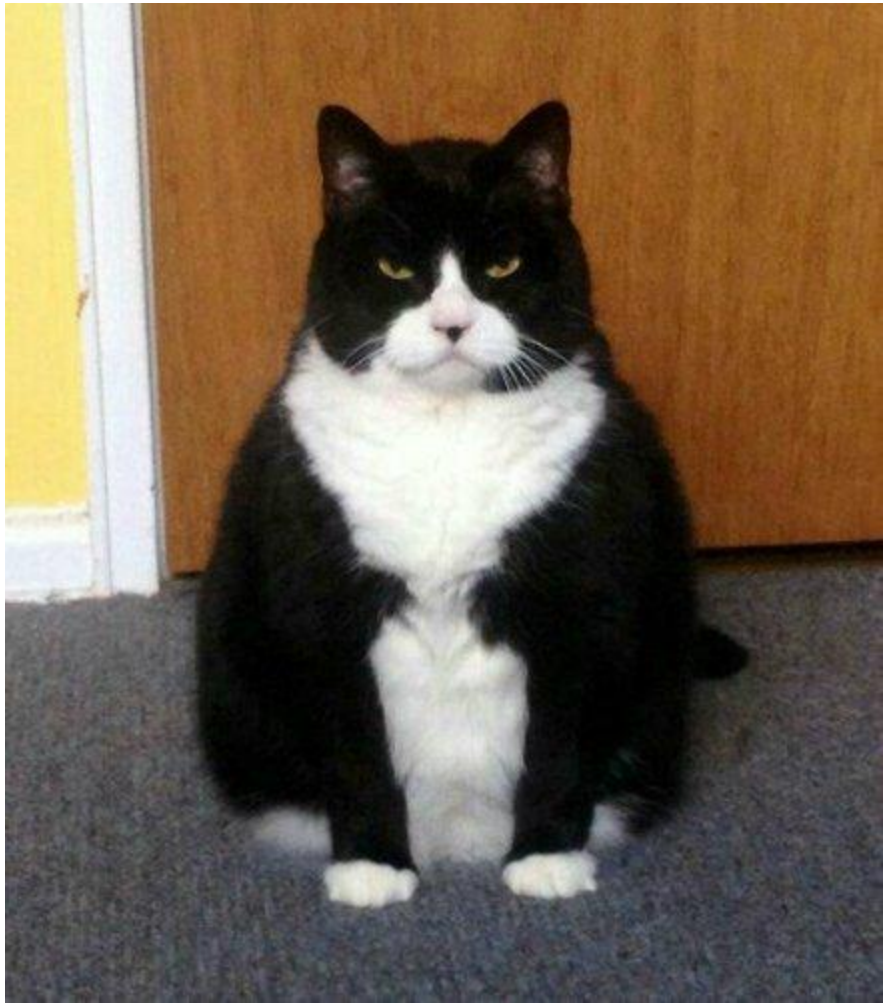
...

Is Elixir suitable for Ethereum?





Crypto Beginners



Consensus algorithms

- Proof of Work
- Proof of Stake
- Proof of Authority

PROOF OF WORK

Proof of Work

- hard, useless problem
- a lot of computational power
- a significant amount of energy


Proof of Stake

Proof of Stake

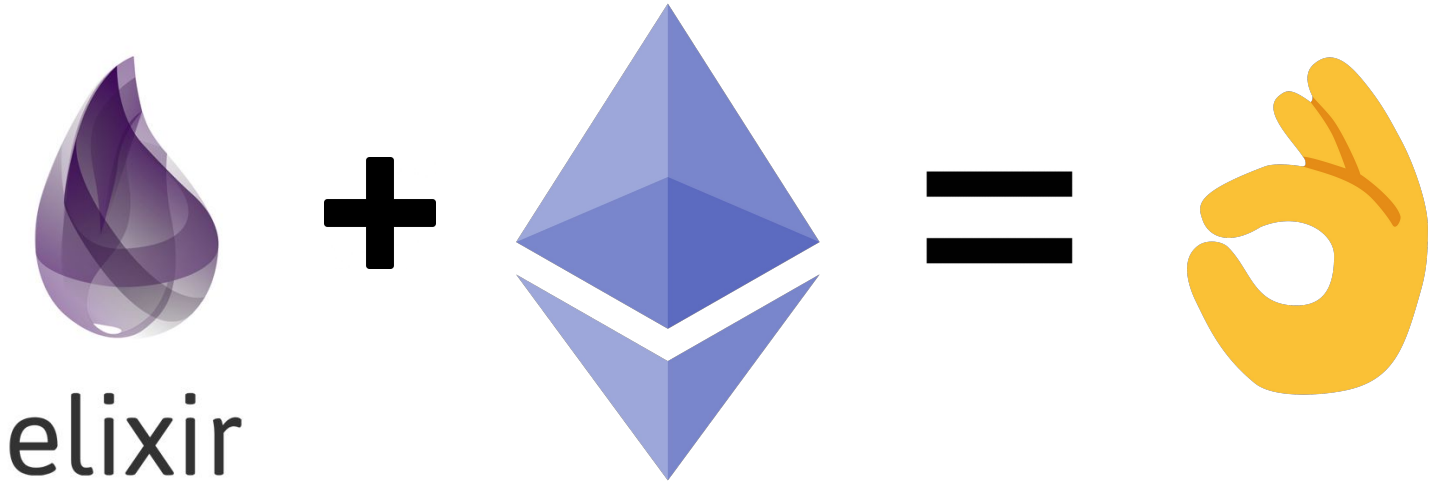
- depends on a validator’s economic stake
- number of tokens you own matter
- small numbers of people own the majority of stakes


Proof of Authority

Proof of Authority

- modification of Proof of Stake
- identity as a stake
- verified personal identities

Is Elixir suitable for Ethereum?



Mana

Project structure

- ExRLP
- MerklePatriciaTree
- EVM
- Blockchain
- ExWire

ExRLP

- Ethereum’s homebrew binary encoding
- simplicity of implementation
- guaranteed absolute byte-perfect consistency

ExRLP - recursive length prefix

```elixir
defprotocol ExRLP.Encode do
  def encode(value, options \\ [])
end

defimpl ExRLP.Encode, for: BitString do
...
end

defimpl ExRLP.Encode, for: Integer do
...
end

defimpl ExRLP.Encode, for: List do
...
end
```

```
iex> [[[]], []] |> ExRLP.Encode.encode
<<195, 193, 192, 192>>

iex> [42, "eth"] |> ExRLP.Encode.encode
<<197, 42, 131, 101, 116, 104>>

iex> [42, ["sun", "moon", 5]] |> ExRLP.Encode.encode
<<204, 42, 202, 131, 115, 117, 110, 132, 109, 111, 111, 110, 5>>
```

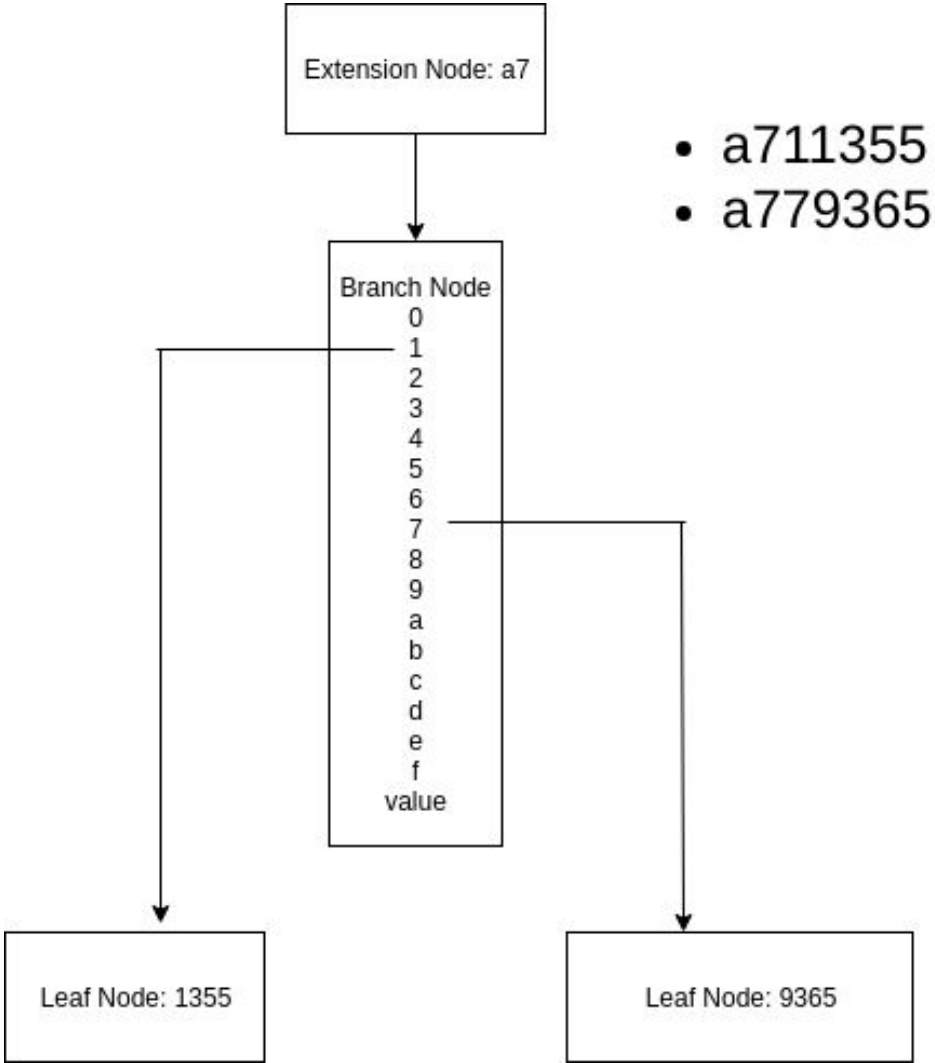
```elixir
@spec encode_item(binary()) :: binary()
defp encode_item(<<byte>> = item) when byte_size(item) == 1 and byte < 128 do
  item
end

defp encode_item(item) when is_binary(item) and byte_size(item) < 56 do
  prefix = 128 + byte_size(item)

  <<prefix>> <> item
end

defp encode_item(item) do
  be_size = Utils.big_endian_size(item)
  byte_size = byte_size(be_size)

  <<183 + byte_size>> <> be_size <> item
end
```

```
    if b0 < 128:  # single byte
        return (b'', bytes, 1, start)
    elif b0 < SHORT_STRING:  # short string
        if b0 - 128 == 1 and rlp[start + 1] < 128:
            raise DecodingError('Encoded as short string although single byte was possible',
rlp)
        return (rlp[start:start + 1], bytes, b0 - 128, start + 1)
    elif b0 < 192:  # long string
        ll = b0 - 183  # - (128 + 56 - 1)
        if rlp[start + 1:start + 2] == b'\x00':
            raise DecodingError('Length starts with zero bytes', rlp)
        len_prefix = rlp[start + 1:start + 1 + ll]
        l = big_endian_to_int(len_prefix)  # noqa: E741
        if l < 56:
            raise DecodingError('Long string prefix used for short string', rlp)
        return (rlp[start:start + 1] + len_prefix, bytes, l, start + 1 + ll)
    elif b0 < 192 + 56:  # short list
```

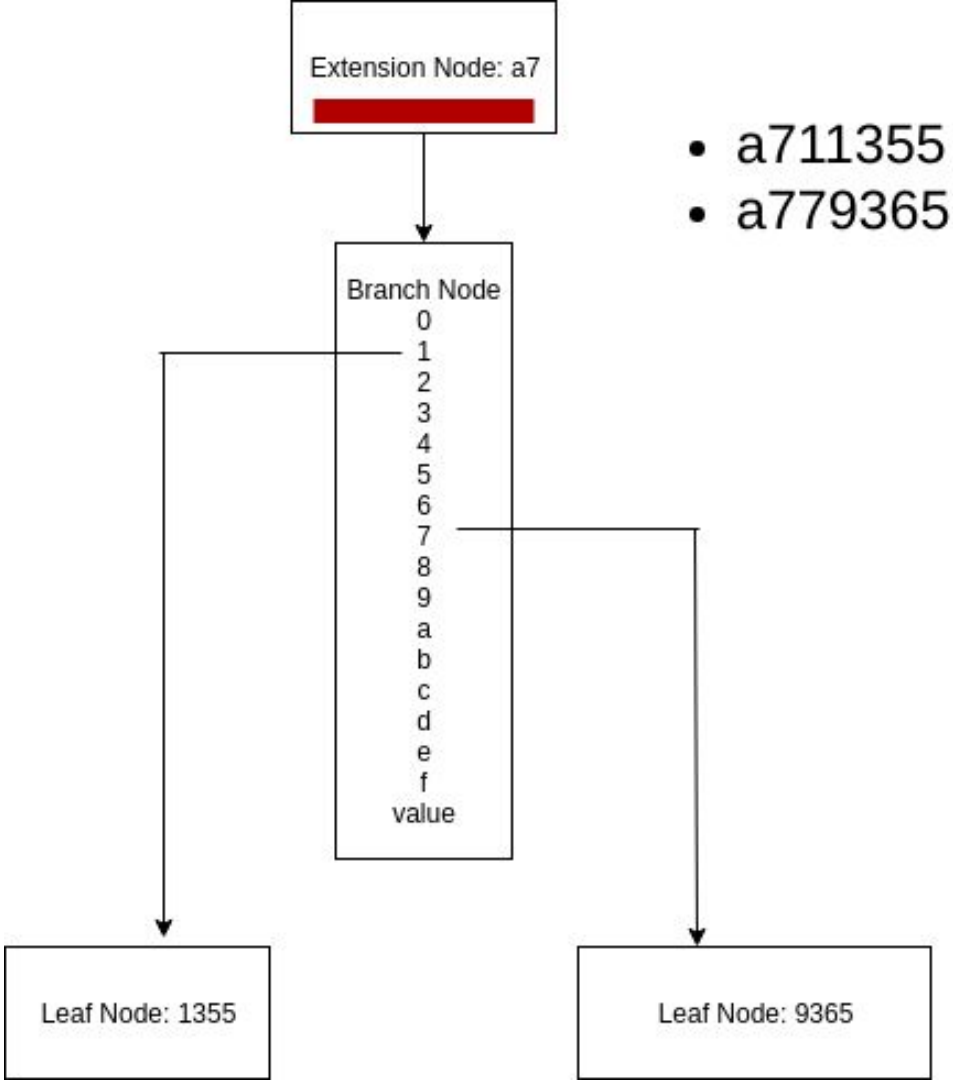
Merkle Patricia Tree (Trie)

- cryptographically authenticated data structure
- Key-value storage
- O(log(n)) efficiency for inserts, lookups and deletes

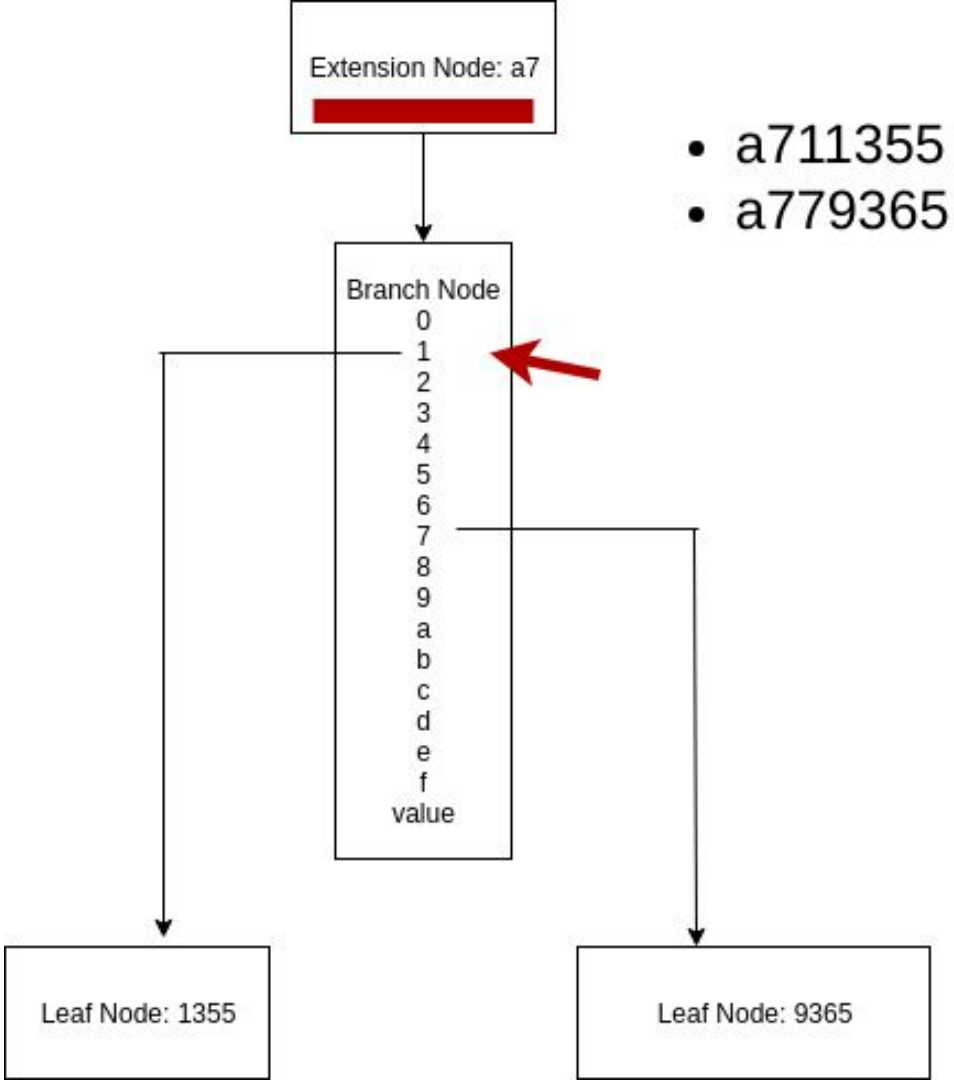
```elixir
defmodule MerklePatriciaTree.DB do

 @callback init(db_name) :: db
 @callback get(db_ref, MerklePatriciaTree.Trie.key()) :: {:ok, value} |
:not_found
 @callback put!(db_ref, MerklePatriciaTree.Trie.key(), value) :: :ok
 @callback delete!(db_ref(), MerklePatriciaTree.Trie.key()) :: :ok
 @callback batch_put!(db_ref, Enumerable.t(), integer()) :: :ok
```



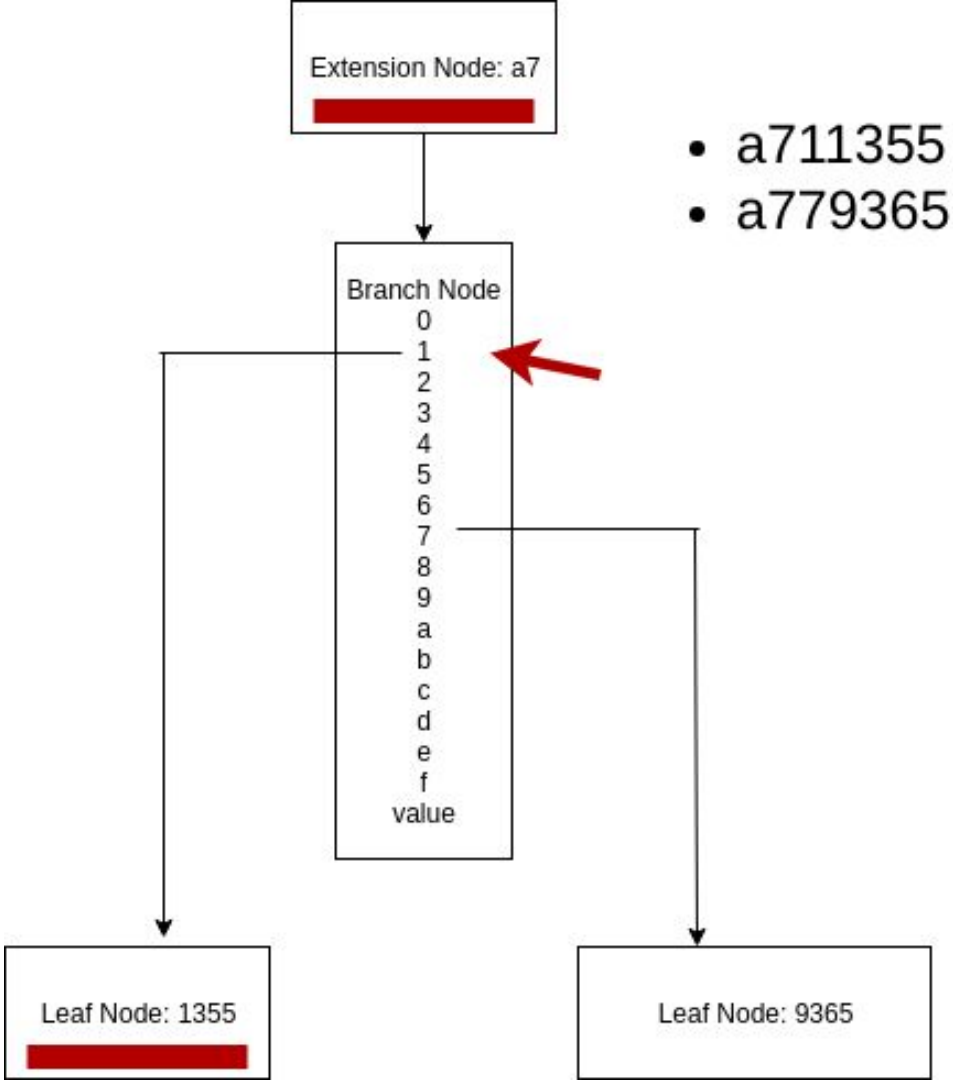
- a711355
- a779365



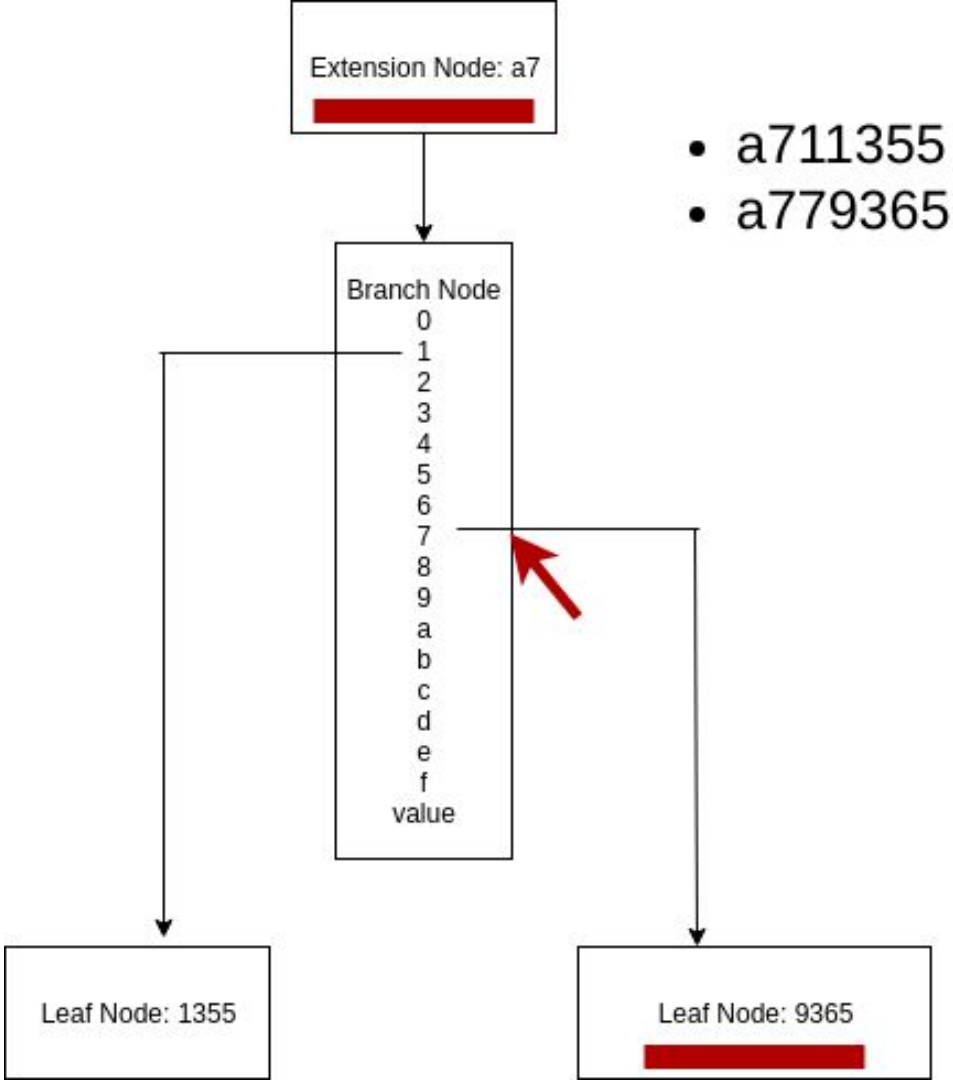
- a711355
- a779365



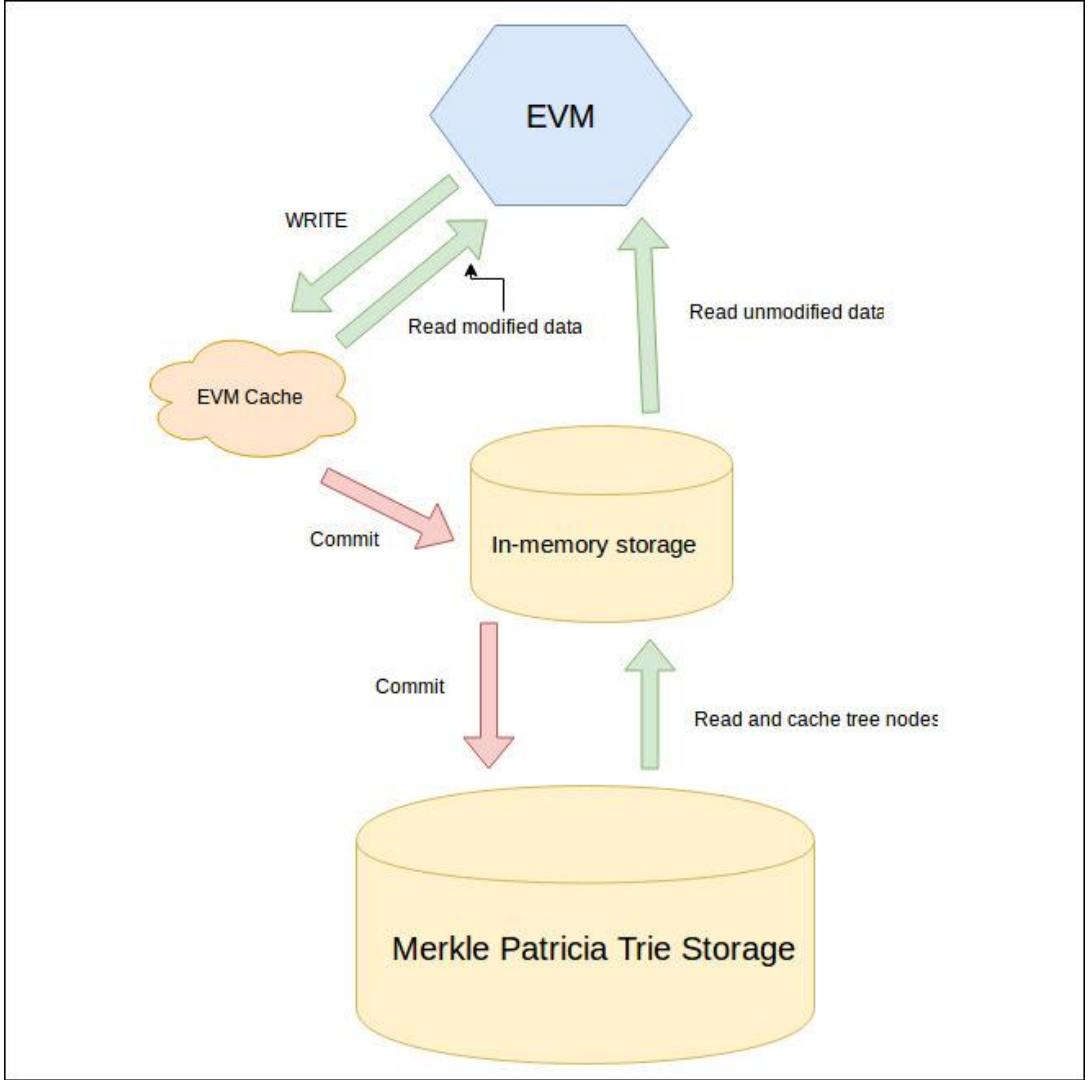
- a711355
- a779365



- a711355
- a779365



- a711355
- a779365

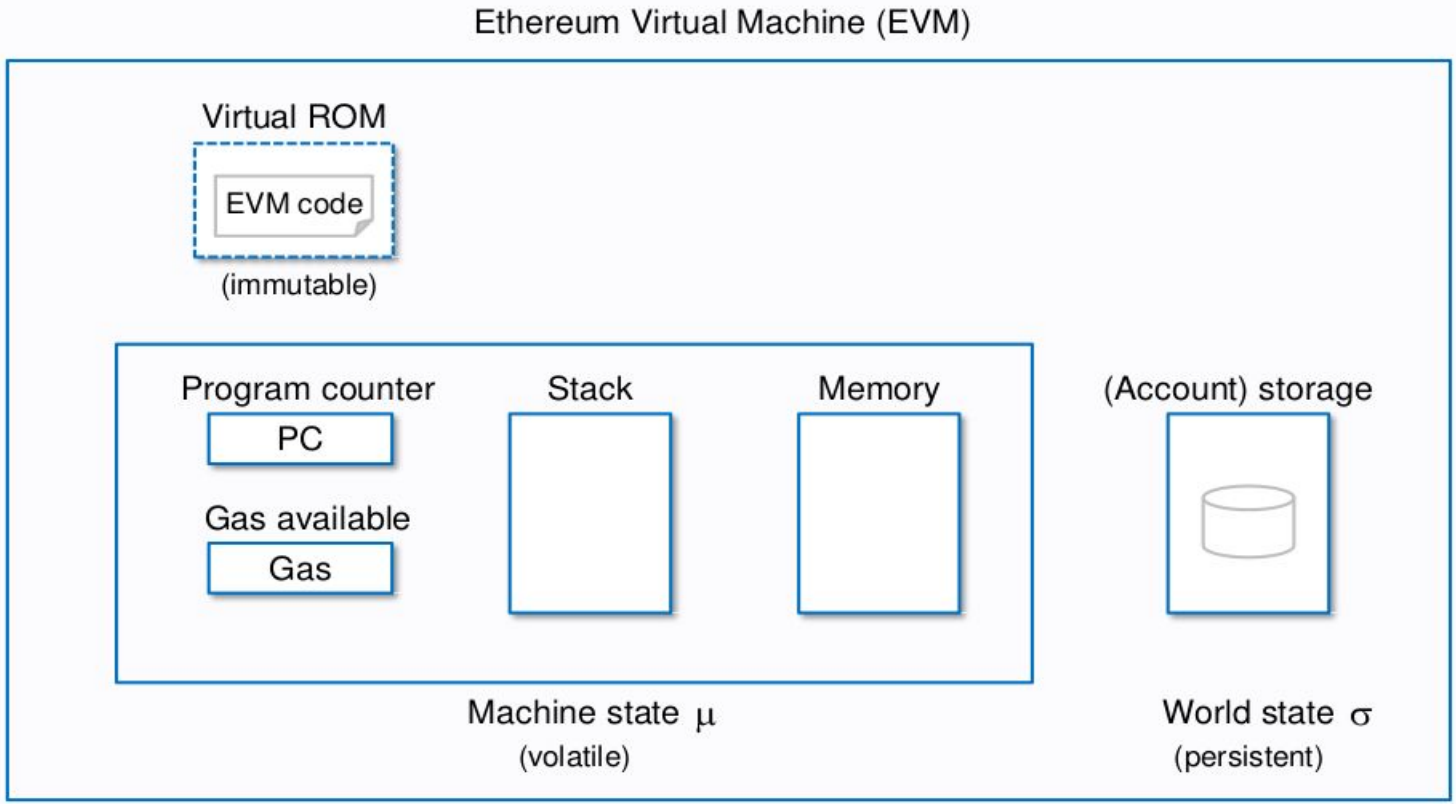

EVM
WRITE
Read modified data
Read unmodified data
EVM Cache
Commit
In-memory storage
Commit
Read and cache tree nodes
Merkle Patricia Trie Storage

EVM

- internal state and computation
- executes machine code compiled from Solidity, LLL etc
- stack machine, the stack has a maximum size of 1024


@merlinragdoll

EVM architecture



The EVM is a simple stack-based architecture.

```
def cycle(machine_state, sub_state, exec_env, cost_with_status) do
   operation = MachineCode.current_operation(machine_state, exec_env)
   inputs = Operation.inputs(operation, machine_state)

   machine_state = MachineState.subtract_gas(machine_state, cost_with_status)
   {updated_exec_env, sub_state} = SubState.add_refund(machine_state, sub_state, exec_env)

   {n_machine_state, n_sub_state, n_exec_env} =
     Operation.run(operation, machine_state, sub_state, updated_exec_env)

   final_machine_state =
     n_machine_state
     |> MachineState.move_program_counter(operation, inputs)
     |> MachineState.increment_step()

   {final_machine_state, n_sub_state, n_exec_env}
 end
```

EVM

```
iex> code = <<96, 1, 96, 0, 1, 96, 0, 85>>

iex> code |> EVM.MachineCode.decompile
[:push1, 1, :push1, 0, :add, :push1, 0, :sstore]
```

```
iex> EVM.run(code)
stack:
[]
operation: push1
stack:
[1]
operation: push1
stack:
[1, 0]
operation: add
stack:
[1]
operation: push1
stack:
[1, 0]
operation: sstore
stack:
[]
```

Blockchain

(1) Validate (or, if mining, determine) ommers;

(2) validate (or, if mining, determine) transactions;

(3) apply rewards;

(4) verify (or, if mining, compute a valid) state and block nonce

Blockchain

```
errors = []

|> check_state_root_validity(child_block, block)

|> check_ommers_hash_validity(child_block, block)

|> check_transactions_root_validity(child_block, block)

|> check_gas_used(child_block, block)

|> check_receipts_root_validity(child_block, block)

|> check_logs_bloom(child_block, block)
```

Blockchain hardfork configuration

- Upgrades in Ethereum
- Way to introduce new changes to the chain

Blockchain hardfork configuration

```
defmodule EVM.Configuration do
  @moduledoc """
  Behaviour for hardfork configurations.
  """

  @type t :: struct()

  # EIP2
  @callback contract_creation_cost(t) :: integer()

  # EIP150
  @callback extcodesize_cost(t) :: integer()
```

ExWire

- RLPx
- DevP2P
- Eth Wire

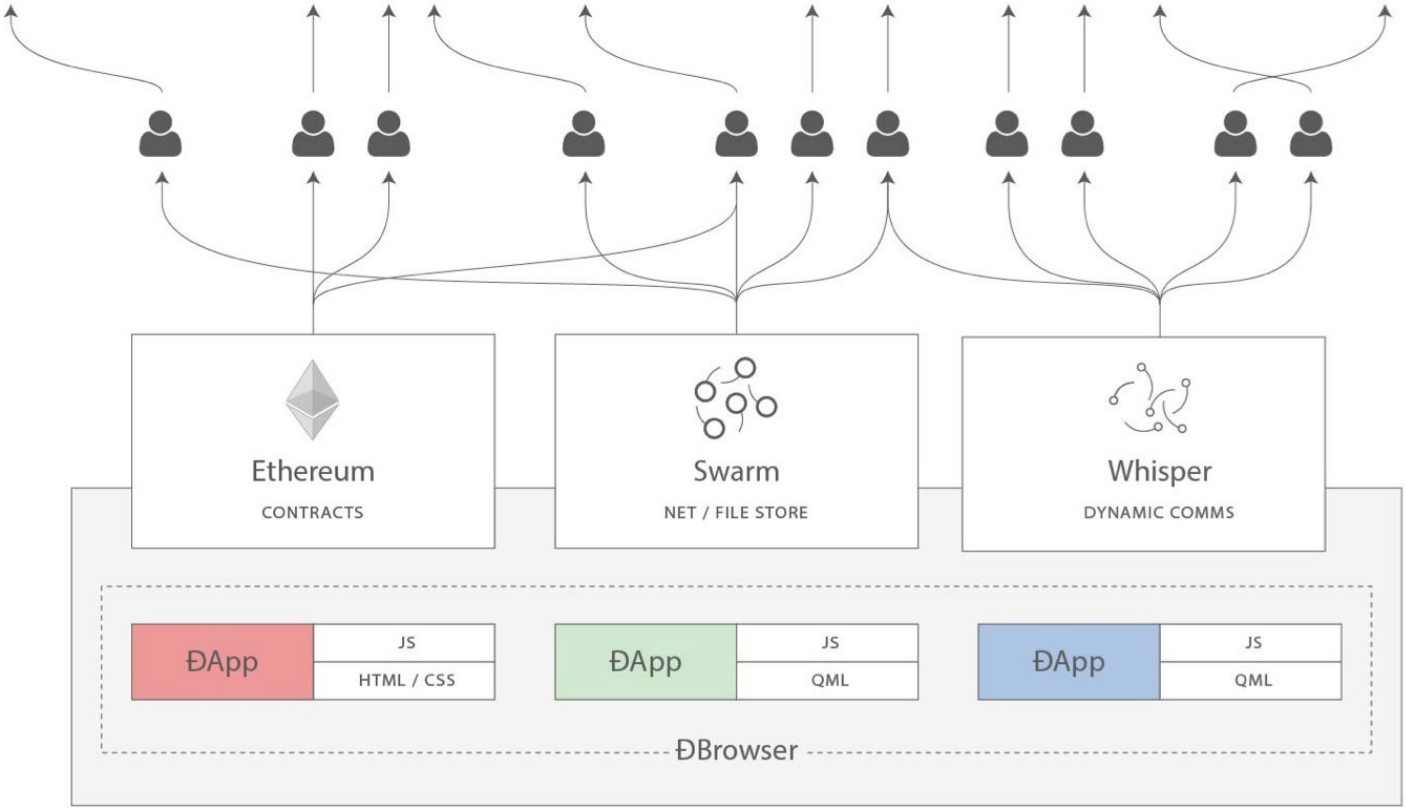
RLPx

- Node Discovery and Network Formation
- Encrypted handshake
- Encrypted transport
- Peer Reputation

DevP2P

- Hello
- Disconnect
- Ping
- Pong

Web3 protocols



Current state

- Passing all common tests
- Working p2p layer
- Working warp sync

Usage

```
git clone https://github.com/mana-ethereum/mana
```

```
mix sync --chain ropsten --provider-url
ipc://path/jsonrpc.ipc
```

Future directions

- JSON-RPC API
- Optimization
- Different consensus algorithms

Advantages of Elixir

- Concise syntax
- Concurrent execution
- Well-documented code

Things to improve for dev community

- Tests are not documented
- Backward compatability
- DevP2P documentation

More Libraries

- Ethereumex

- Ex_abi

- BN

Who are using our projects

- OmiseGO
- Consensys
- AgileAlpha

About me

Github: https://github.com/ayrat555

Blog: https://www.badykov.com/

Telegram: https://t.me/Ayrat555

Email: ayratin555@gmail.com

Thanks!

https://github.com/poanetwork/mana

https://forum.poa.network/t/elixir-developer/2047