Mana

Ethereum client in Elixir

Ayrat Badykov

Acknowledgements

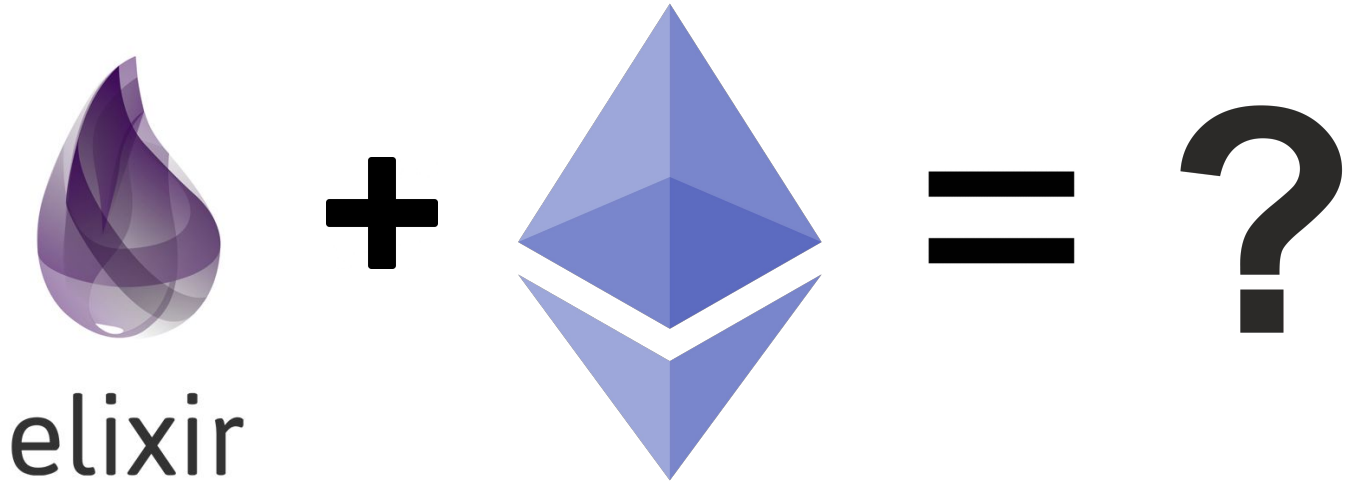
Ethereum

Ethereum clients

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

...

Is Elixir suitable for Ethereum?



Consensus algorithms

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

Proof of Work

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

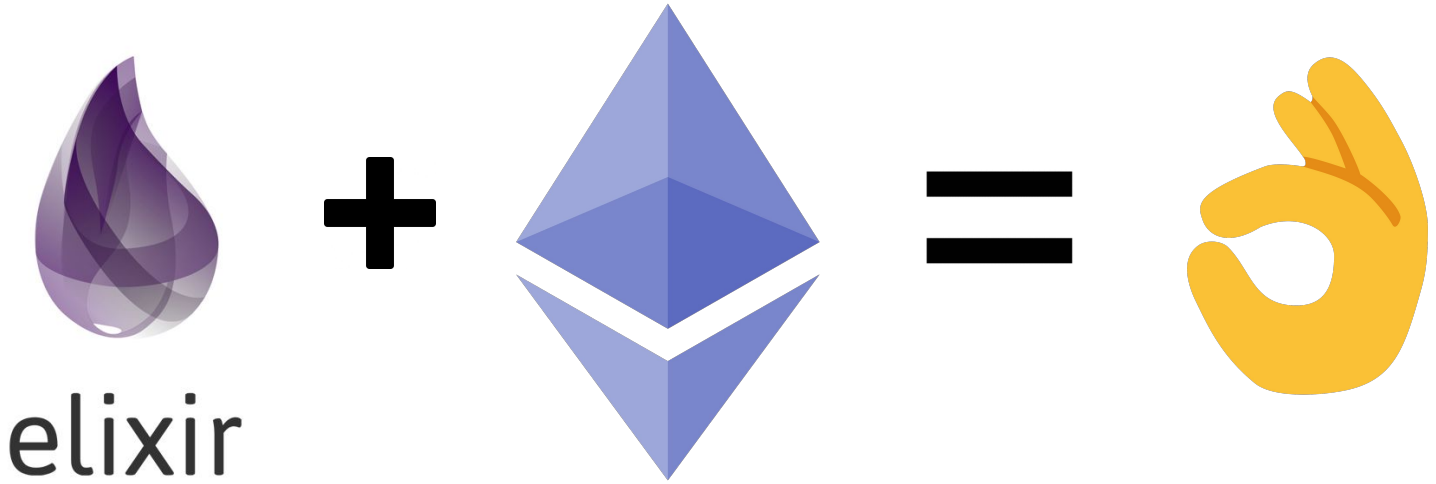
Proof of Stake

- 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

- 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>>
```

ExRLP

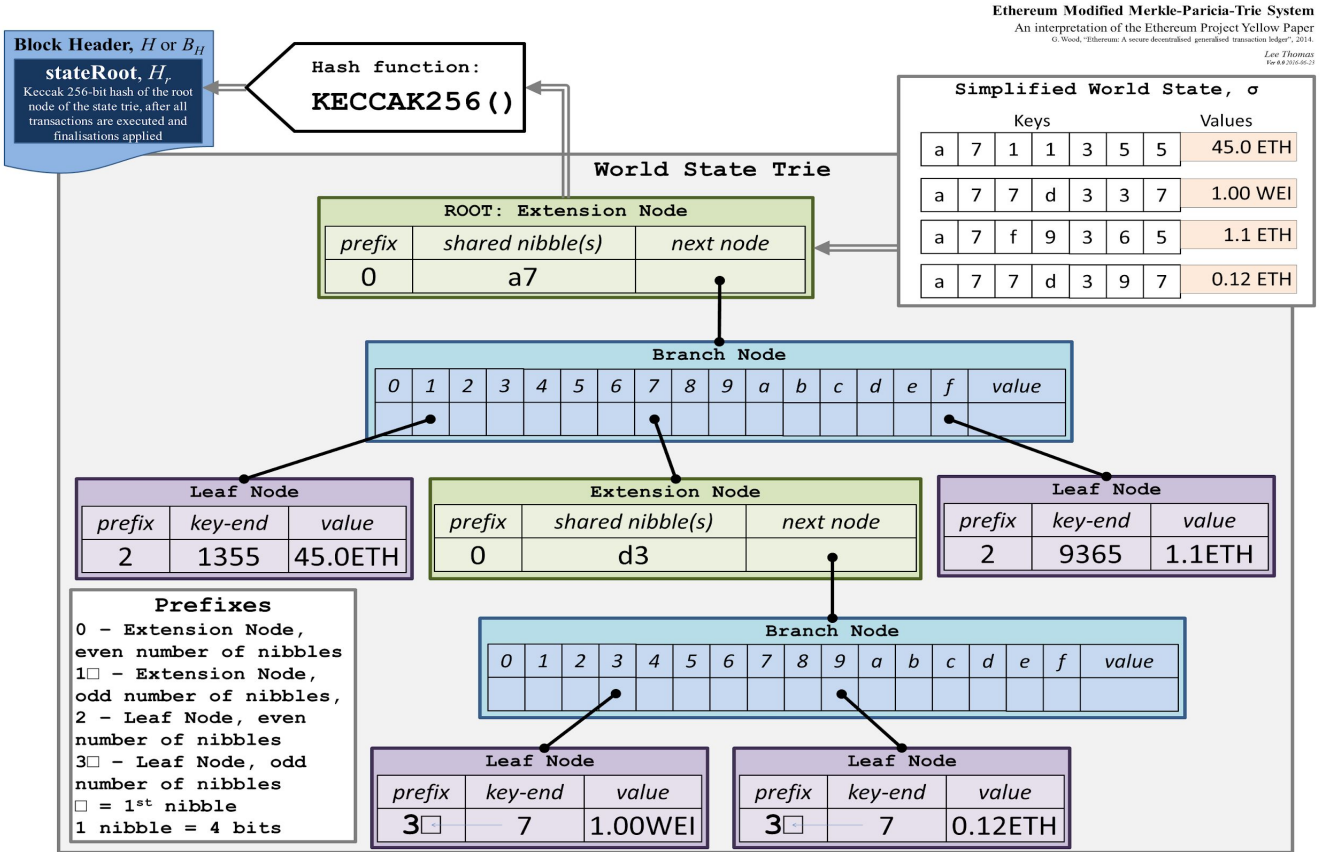
https://www.badykov.com/elixir/2018/05/06/rlp/

https://github.com/exthereum/ex_rlp

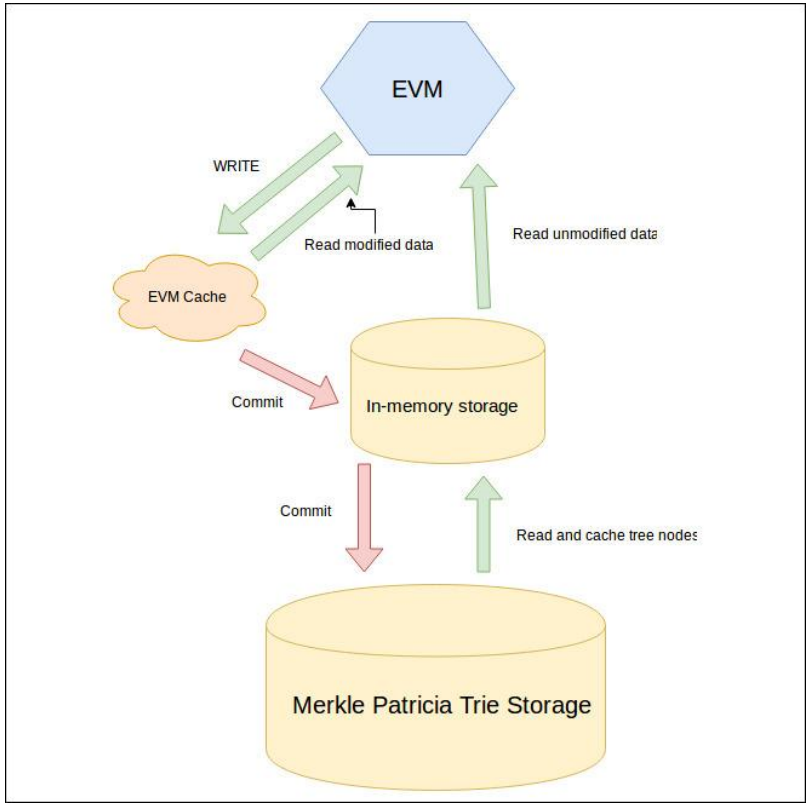
Merkle Patricia Tree (Trie)

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

Merkle Patricia Tree (Trie)



Storage overview



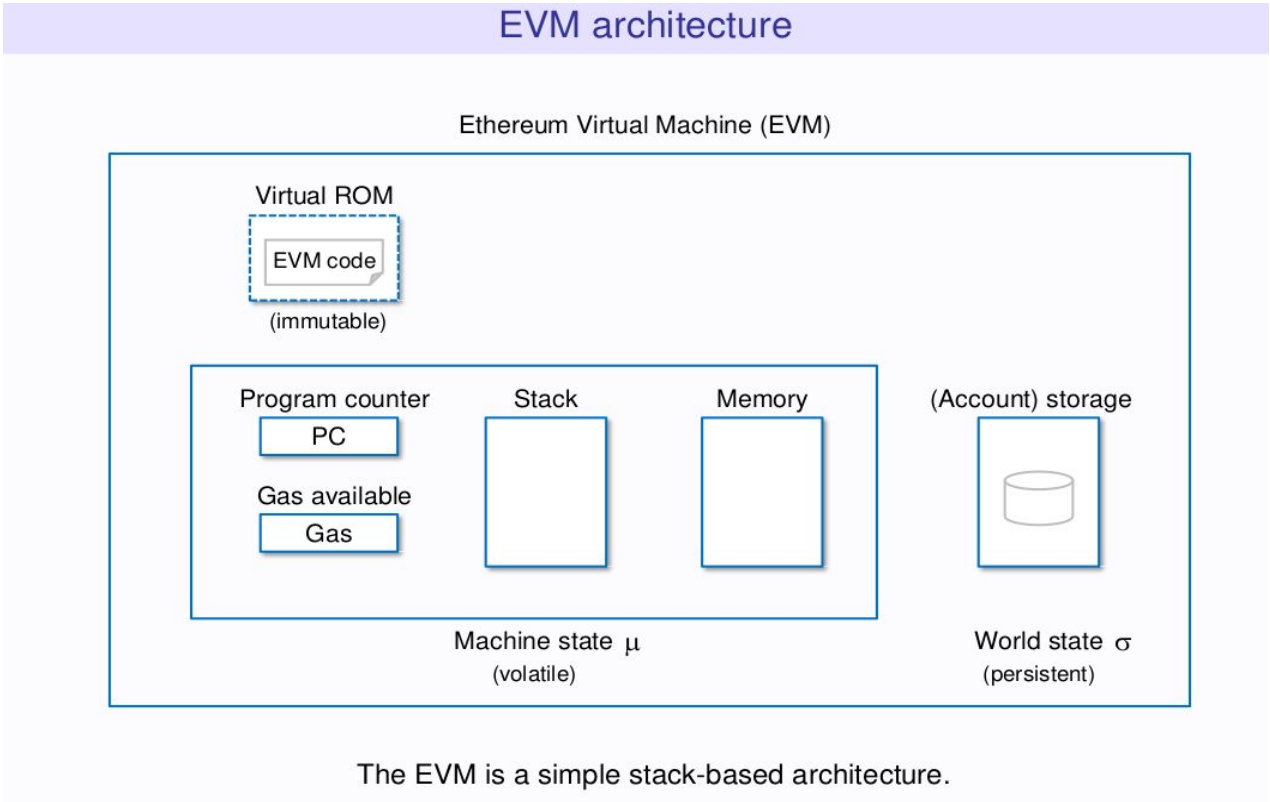
Storage in Ethereum

https://www.badykov.com/ethereum/2018/11/10/storage-in-ethereum/

EVM

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

EVM


EVM architecture
Ethereum Virtual Machine (EVM)
Virtual ROM
EVM code
(immutable)
Program counter
PC
Gas available
Gas
Stack
Memory
(Account) storage
Machine state μ
(volatile)
World state σ
(persistent)
The EVM is a simple stack-based architecture.

EVM

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

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

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

EVM

https://www.badykov.com/elixir/2018/04/29/evm-basics/

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

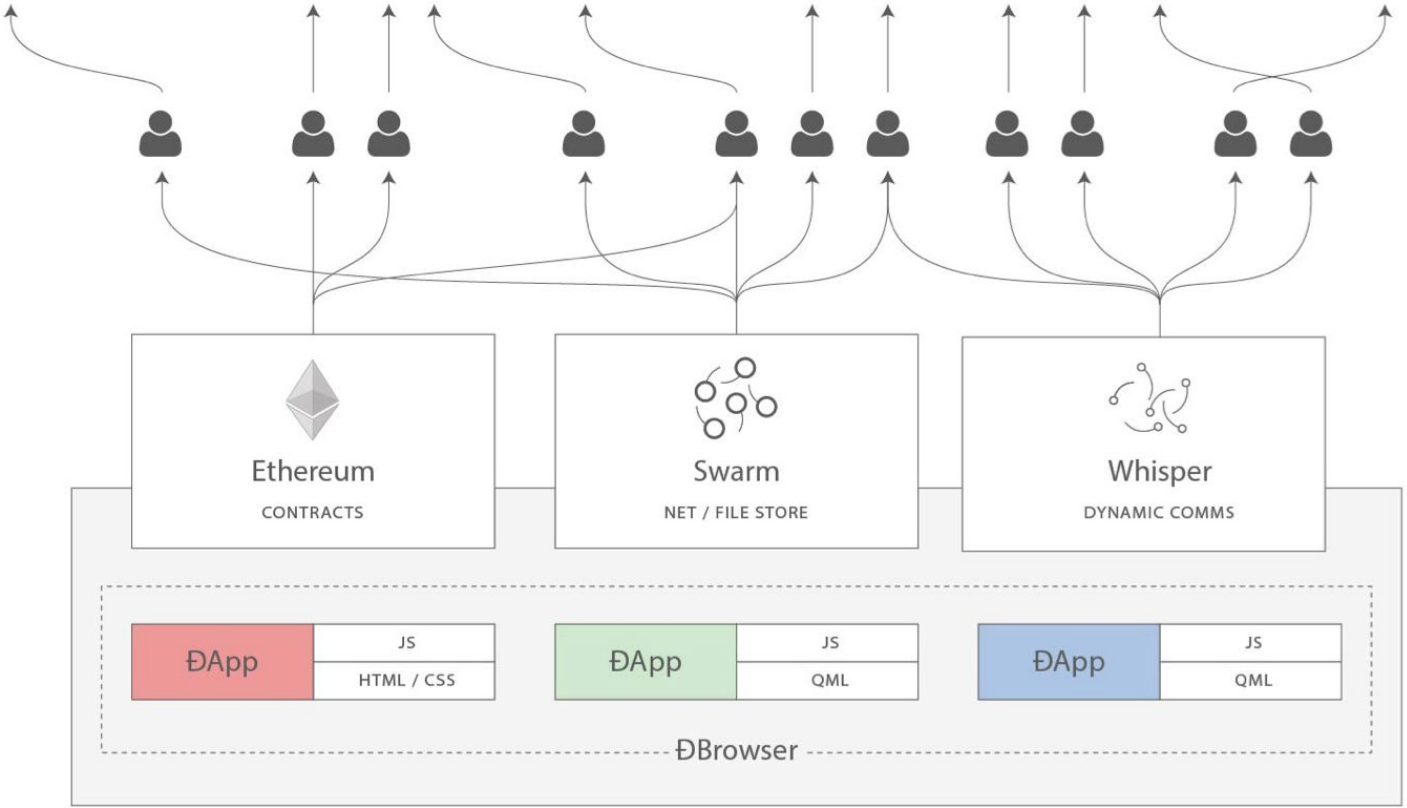
RLPx

- 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

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

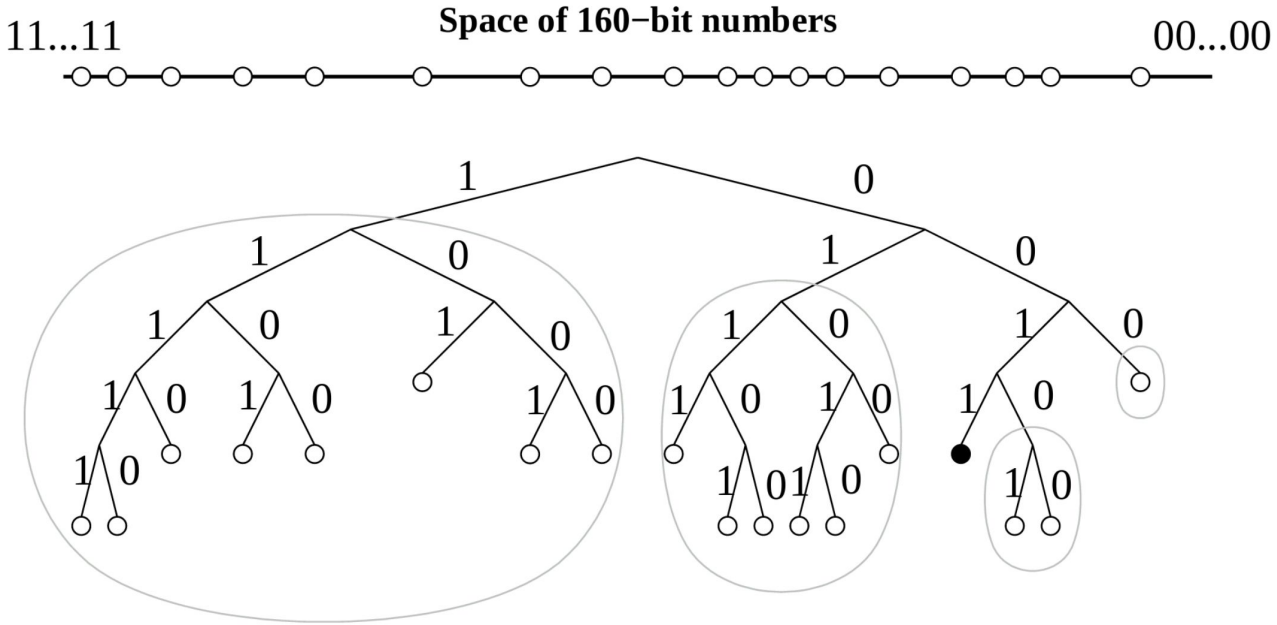
More Libraries

- Ethereumex
- Ex_abi
- BN

Who are using our projects

- OmiseGO
- Consensys
- AgileAlpha

Node Discovery. Kademlia



Node Discovery. Storage

- Nodes
- K-buckets
- Routing Table

Node Discovery. Protocol

- Ping
- Pong
- FindNeighbours
- Neighbours

- bucket size (k) = 16
- concurrenct (alpha) = 3
- number of buckets = 256

Node Discovery. Implementation

- ExWire.Network - sends and receives messages.
- ExWire.Kademlia.Server - encapsulates Kademlia algorithm’s state.

Node Discovery. Implementation

```
def init(params) do
  {udp_module, udp_process_name} = discovery_param(params, :network_adapter)
  kademlia_name = discovery_param(params, :kademlia_process_name)
  port = discovery_param(params, :port)
  bootnodes = (params[:nodes] || nodes()) |> Enum.map(&Node.new/1)

  children = [
   {KademliaServer,
    [
     name: kademlia_name,
     current_node: current_node(),
     network_client_name: udp_process_name,
     nodes: bootnodes
    ]},
   {udp_module,
    [
     name: udp_process_name,
     network_module: {Network, [kademlia_process_name: kademlia_name]},
     port: port
    ]}
  ]

  Supervisor.init(children, strategy: :rest_for_one)
 end
```

Node Discovery. Implementation

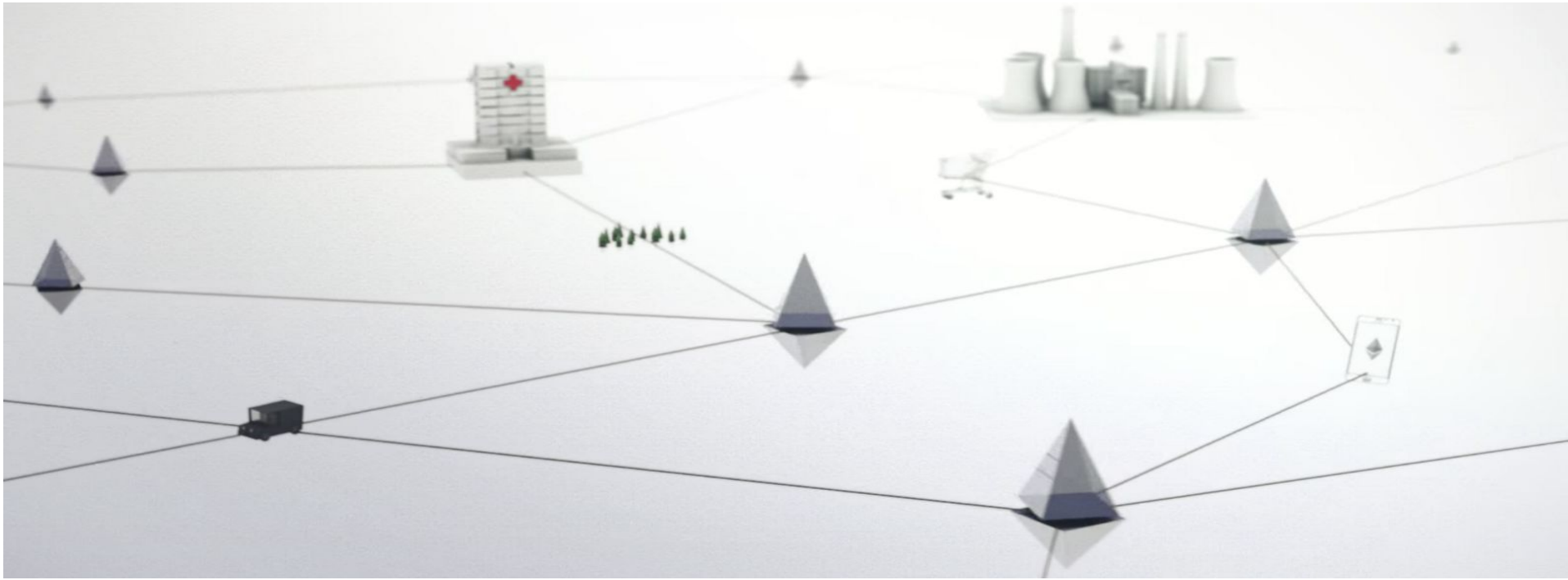
```
@spec refresh_node(t(), Node.t(), Keyword.t()) :: {atom, t()}
  def refresh_node(bucket = %__MODULE__{}, node, options \\ [time: :actual]) do
    cond do
      member?(bucket, node) -> {:reinsert_node, node, reinsert_node(bucket, node,
options)}
      full?(bucket) -> {:full_bucket, last(bucket), bucket}
      true -> {:insert_node, node, insert_node(bucket, node, options)}
    end
  end
```

Node Discovery. Implementation

```elixir
@spec start(RoutingTable.t(), [Node.t()]) :: RoutingTable.t()
 def start(table, bootnodes \\ []) do
  table = add_bootnodes(table, bootnodes)

  this_round_nodes = RoutingTable.discovery_nodes(table)

  Enum.each(this_round_nodes, fn node ->
   find_neighbours(table, node)
  end)

  %{
   table
   | discovery_nodes: table.discovery_nodes ++
        this_round_nodes,
     discovery_round: table.discovery_round + 1
  }
 end
```

```elixir
@spec add_bootnodes(RoutingTable.t(), [Node.t()]) :: RoutingTable.t()
 defp add_bootnodes(table, nodes) do
  Enum.reduce(nodes, table, fn node, acc ->
   RoutingTable.refresh_node(acc, node)
  end)
 end

 @spec find_neighbours(RoutingTable.t(), Node.t()) :: :ok
 defp find_neighbours(table, node) do
  find_neighbours = FindNeighbours.new(table.current_node.public_key)

  Network.send(find_neighbours, table.network_client_name, node.endpoint)
 end
```

Node Discovery. Implementation



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