æternity

WeAreDevelopers

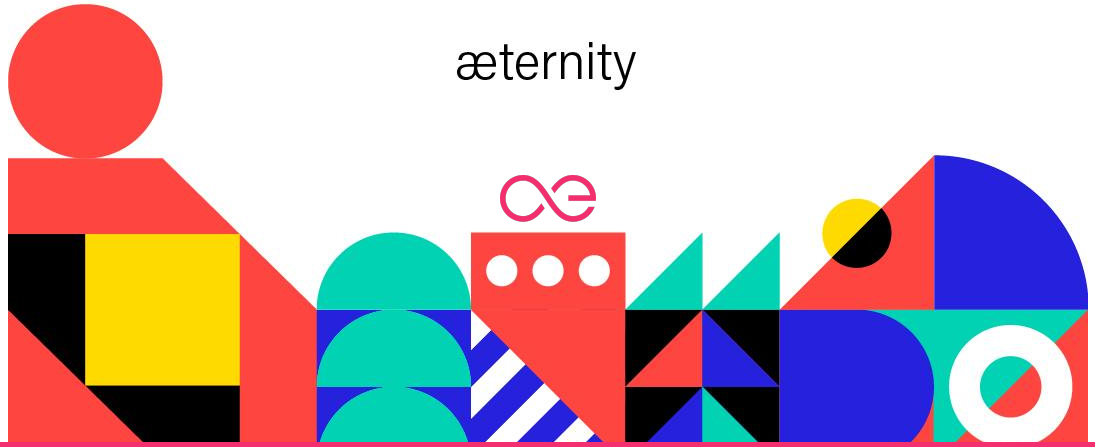
7th June
11:15–12:00
13:00–15:00

WORKSHOP

æternity Theoretics
Building on æternity

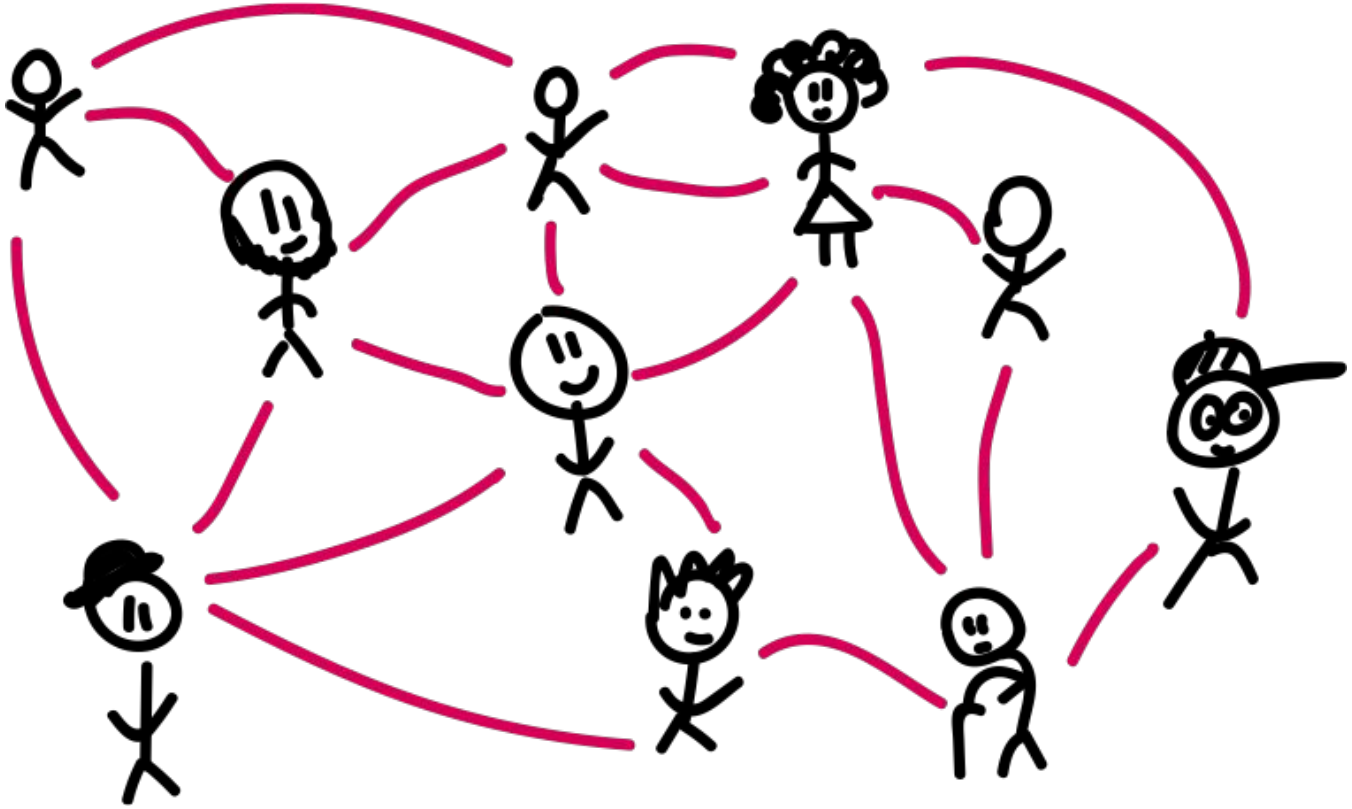
Milen Radkov
Phillipp Piwowarsky

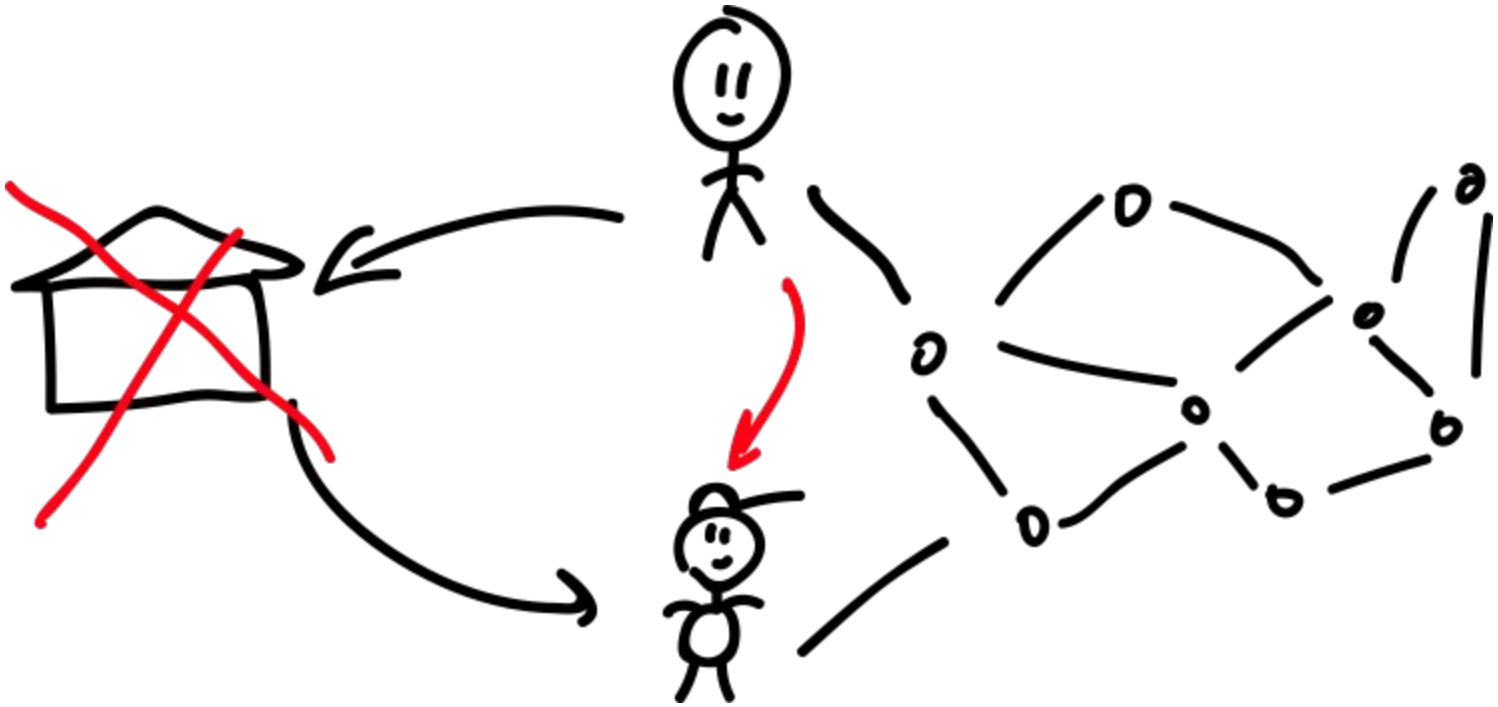
æternity

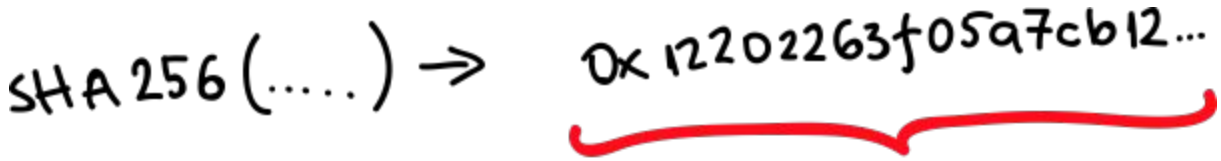


Blockchain

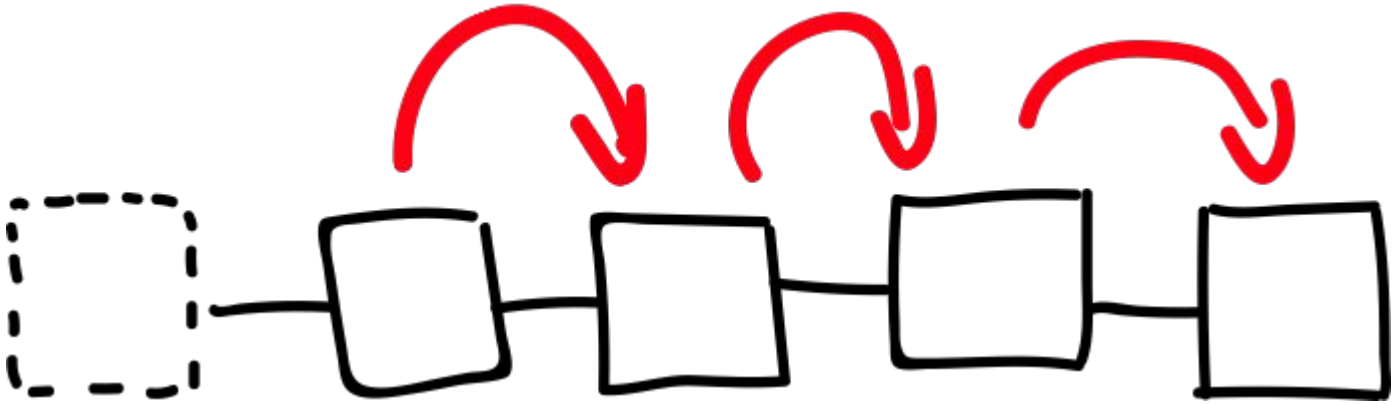
Satoshi Nakamoto

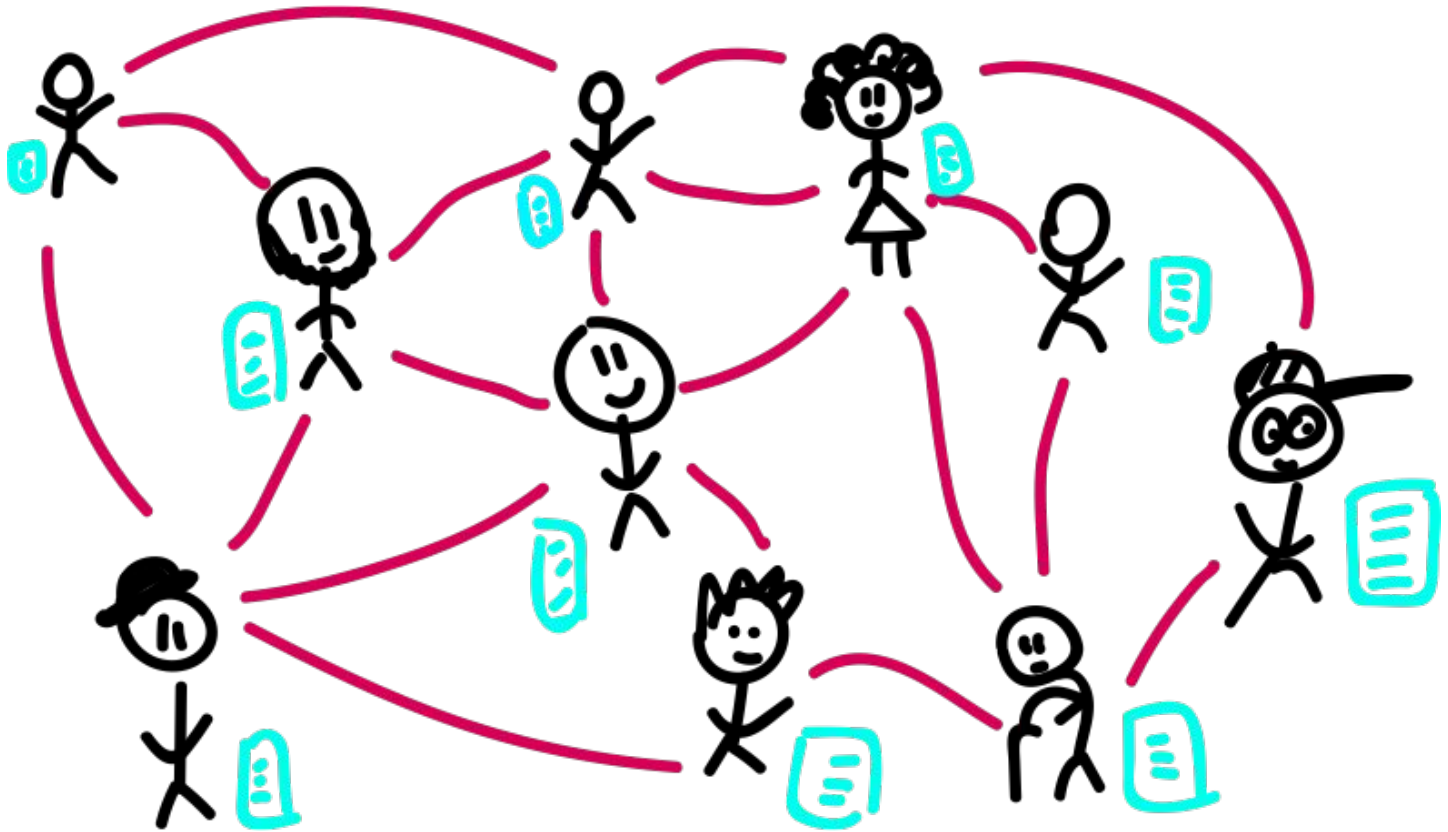






hash

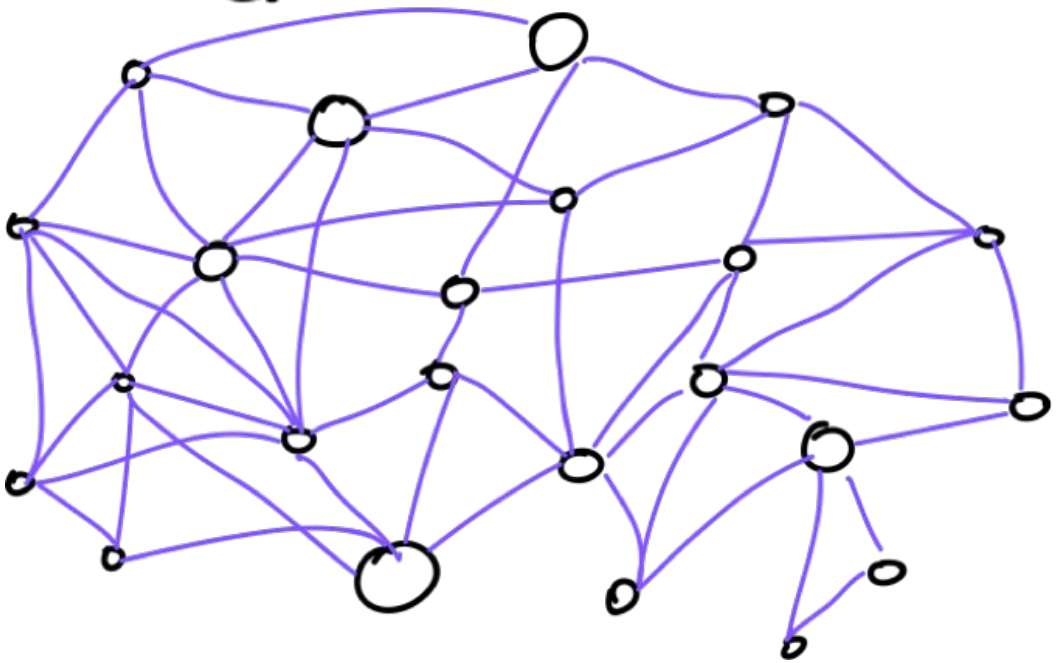




- [x] Trust
- [x] Double spending - Solved!
- [x] No Intermediaries
- [x] Transfer of value
- [x] Immutable store


Smart
Contract

DAO



æternity

Sophia Language

What is æternity?

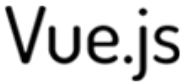
An open-source, blockchain 3.0 apps platform.

Aiming to address problems related to:

- scalability & cost
- governance
- usability & user-friendliness
- efficiency
- real world data
- smart-contract security

Technology

Some of the technology we use.



You can find a full list on GitHub.

The æternity team

- Erlang superstars
- Cryptocurrency veterans
- Blockchain enthusiasts & believers
- More than 200 years of total programming experience

Features:

Bitcoin-NG Consensus.

Developed in Cornell University. Based on the proven design of the Nakamoto Consensus. Based on key and micro blocks. **Improves on-chain transaction speed** allowing up to 6000 transactions per minute (100 tx/s).

Features:

Naming System (AENS).

æternity provides an **integrated naming system**. A name could point to an account or an oracle on the æternity blockchain. The naming system **increases usability, provides utility**, and allows for transfer of ownership.

Beta version implemented *name.test*. Name auctions -> *name.ans*.

Features:

Governance.

æternity's governance is implemented via [delegated] **voting, weighted by the amount of tokens the account holds**. Provides both technical tools to permit governance and frameworks for human interaction for **effective discussion**.

Liquid democracy on-chain. Non-binding outcomes. No staking. AE token users signal to the developers what they believe is the best approach. Anyone can post a question, anyone with AE tokens can reply.

Features:

Scalability via State Channels.

State channels provide a method for users to **privately communicate and transact off-chain**. æternity's method for **reducing the on-chain load**.

In-Depth:

State Channels - how do they work

- Two party encrypted communication
- Stateful and can have off-chain smart contracts
- Two phase protocol
- Part of the on-chain protocol

In-Depth:

State Channels - perks

1. Throughput limited by networking and CPU power.
2. Blazingly fast.
3. Cheap.

In-Depth:

State Channels - use cases

- Instant payments - no need for confirmations
  - Coffee
  - Offline-payments (developing countries)
- Micro payments - payments per second/minute/hour/day
  - Monthly payments can be broken down to regular payments
  - Services - pay what you consume
- Privacy - not there yet

Features:

Cuckoo Cycle PoW.

æternity uses Cuckoo Cycle (ASIC resistant) developed by John Tromp. GRIN uses a variation of the same algorithm.

Cuckoo Cycle is the first graph-theoretic Proof-of-Work, and **the most memory bound**, yet with instant verification.

Features:

Oracles.

Oracles are source of information which can be accessed on the blockchain. **Anyone can be an oracle provider**, their reputation determines whether or not they are seen as a reliable source.

Features:

Unique smart contract approach.

1. Contract execution should be safe.
2. Contract execution should be efficient and scale.
3. Contract execution should be cheap.
4. There should be a simple way to migrate from existing smart contracts.

Smart contract language – **Sophia [ML]**. Functional Ocaml-like language, syntax most resembles that of Reason.

Aeternity ecosystem

- Documentation Hub
- Developer Tools
- Wallets
- SDKs
- Middleware

Aeternity SDKs

- Javascript
- Python
- Go

Developer Tools

- Forgae
    - forgae init
    - forgae node
    - forgae test
    - forgae compile
    - forgae deploy

Sophia

Sophia is a ML-family language.

- developed to be used for creating smart contracts on Aeternity Blockchain
- strongly typed language
- restricted mutable state
- state

Sophia simple storage smart contract

```
contract SimpleStorage =

    record state = { stored_data: int }

    public function init() : state = { stored_data = 0 }

    public stateful function set(x: int) =
      put(state{ stored_data = x })

    public function get() : int = state.stored_data
```

Solidity vs Sophia comparison

```
pragma solidity >=0.4.0 <0.6.0;

contract SimpleStorage {
  uint storedData;

  function set(uint x) public {
    storedData = x;
  }

  function get() public view returns (uint) {
    return storedData;
  }
}
```

```
contract SimpleStorage =

    record state = { stored_data: int }

    public function init() : state = { stored_data = 0 }

    public stateful function set(x: int) =
      put(state{ stored_data = x })

    public function get() : int = state.stored_data
```

Sophia types

Type	Description	Example
int	A 256 bit 2-complement integer	`-1`
address	A 256 bit number given as a hex	`ff00`
bool	A Boolean	`true`
bits	A bit field (with 256 bits)	`Bits.none`
string	An array of bytes	`"Foo"`
list	A homogeneous immutable singly linked list.	`[1, 2, 3]`
tuple	An ordered heterogeneous array	`(42, "Foo", true)`
record	An immutable key value store with fixed key names and typed values	`record balance = { owner: address, value: int }`

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#types

Sophia types

map	An immutable key value store with dynamic mapping of keys of one type to values of one type	`type accounts = map(string, address)`
option('a)	An optional value either None or Some('a)	`Some(42)`
state	A record of blockstate key, value pairs	
transactions	An append only list of blockchain transactions	
events	An append only list of blockchain events (or log entries)	
signature	A signature - 64 bytes	
Chain.ttl	Time-to-live (fixed height or relative to current block)	`FixedTTL(1050)` `RelativeTTL(50)`
oracle('a, 'b)	And oracle answering questions of type 'a with answers of type 'b	`Oracle.register(acct, qfee, ttl)`
oracle_query('a, 'b)	A specific oracle query	`Oracle.query(o, q, qfee, qttl, rttl)`

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#types

Sophia Built-in functions

- Map.

Builtin functions on maps

The following builtin functions are defined on maps:

```
Map.lookup(k : 'k, m : map('k, 'v)) : option('v)
Map.lookup_default(k : 'k, m : map('k, 'v), v : 'v) : 'v
Map.member(k : 'k, m : map('k, 'v)) : bool
Map.delete(k : 'k, m : map('k, 'v)) : map('k, 'v)
Map.size(m : map('k, 'v)) : int
Map.to_list(m : map('k, 'v)) : list(('k, 'v))
Map.from_list(m : list(('k, 'v))) : map('k, 'v)
```

https://github.com/aeternity/protocol/blob/minerva/contracts/sophia.md#builtin-functions-on-maps

Sophia Built-in functions

- Map.
- String.

Builtin functions on strings

The following builtin functions are defined on strings:

```
String.length(s : string) : int
String.concat(s1 : string, s2 : string) : string
String.sha3(s : string) : hash
String.sha256(s : string) : hash
String.blake2b(s : string) : hash
```

https://github.com/aeternity/protocol/blob/minerva/contracts/sophia.md#strings

Sophia Built-in functions

- Map.
- String.
- Int.

Builtin functions on integers

The following builtin functions are defined on integers:

```
Int.to_str(i : int) : string
```

https://github.com/aeternity/protocol/blob/minerva/contracts/sophia.md#builtin-functions-on-integers

Sophia Built-in functions

- Map.
- String.
- Int.
- Crypto.

Cryptographic primitives

The following hash functions are supported:

```
Crypto.sha3(x : 'a) : hash
Crypto.sha256(x : 'a) : hash
Crypto.blake2b(x : 'a) : hash
String.sha3(s : string) : hash
String.sha256(s : string) : hash
String.blake2b(s : string) : hash
```

https://github.com/aeternity/protocol/blob/minerva/contracts/sophia.md#cryptographic-primitives

Sophia Built-in functions

- Map.
- String.
- Int.
- Crypto.
- Interface for Oracle.

An Oracle operator will use the functions:

- `Oracle.register`
- `Oracle.get_question`
- `Oracle.respond`
- `Oracle.extend`

An Oracle user will use the functions:

- `Oracle.query_fee`
- `Oracle.query`
- `Oracle.get_answer`

https://github.com/aeternity/protocol/blob/minerva/contracts/sophia.md#oracle-interface

Sophia Built-in functions

- Map.
- String.
- Int.
- Crypto.
- Interface for Oracle.
- Interface for Account.

To spend tokens from the contract account to the account "to" you call the **Chain.spend** function.

```
Chain.spend(to : address, amount : integer)
```

https://github.com/aeternity/protocol/blob/minerva/contracts/sophia.md#account-interface

Sophia Contract primitives

- Contract.

- `Contract.creator` is the address of the entity that signed the contract creation transaction.
- `Contract.address` is the address of the contract account.
- `Contract.balance` is the amount of coins currently in the contract account. Equivalent to `Chain.balance(Contract.address)`.

https://github.com/aeternity/protocol/blob/minerva/contracts/sophia.md#contract-primitives

Sophia Contract primitives

- Contract.
- Call.

  - `Call.origin` is the address of the account that signed the call transaction that led to this call.
  - `Call.caller` is the address of the entity (possibly another contract) calling the contract.
  - `Call.value` is the amount of coins transferred to the contract in the call.
  - `Call.gas_price` is the gas price of the current call.
  - `Call.gas_left()` is the amount of gas left for the current call.

https://github.com/aeternity/protocol/blob/minerva/contracts/sophia.md#contract-primitives

Sophia Contract primitives

- Contract.
- Call.
- Chain.

  - `Chain.balance(a : address)` returns the balance of account `a`.
  - `Chain.block_hash(h)` returns the hash of the block at height `h`.
  - `Chain.block_height` is the height of the current block (i.e. the block in which the current call will be included).
  - `Chain.coinbase` is the address of the account that mined the current block.
  - `Chain.timestamp` is the timestamp of the current block.
  - `Chain.difficulty` is the difficulty of the current block.
  - `Chain.gas_limit` is the gas limit of the current block.

https://github.com/aeternity/protocol/blob/minerva/contracts/sophia.md#contract-primitives

Sophia Arithmetic operations

- Safe arithmetic operations.
- Sophia values are 256-bit words.

  - addition (`x + y`)
  - subtraction (`x - y`)
  - multiplication (`x * y`)
  - division (`x / y`), truncated towards zero
  - remainder (`x mod y`), satisfying `y * (x / y) + x mod y == x` for non-zero `y`
  - exponentiation (`x ^ y`)

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#arithmetic

Lists

- Dynamically sized, homogenous, immutable, singly linked list

```
[1, 33, 2, 666]                                                    : list(int)
[(1, "aaa"), (10, "jjj"), (666, "the beast")]                      : list((int, string))
[{[1] = "aaa", [10] = "jjj"}, {[5] = "eee", [666] = "the beast"}] : list(map(int, string))
```

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#lists

Lists

- Prepend with ::
- Concat with ++

```
42 :: [1,2,3]  == [42,1,2,3]

[11,22,33] ++ [44,55,66] == [11,22,33,44,55,66]
```

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#lists

Maps and records

- Record is a fixed set of fields with associated possibly different types

```
record account = { name    : string,
                   balance : int,
                   history : list(transaction) }
```

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#maps-and-records

Maps and records

- Maps can contain an arbitrary number of key-value bindings, with fixed type

  map(‘k, ‘v)

- The type can be any type but function or map type.

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#maps-and-records

Maps and records

- Constructing with a function

- Record

```
function new_account(name) =
  {name = name, balance = 0, history = []}
```

- Map

```
function example_map() : map(string, int) =
  {["key1"] = 1, ["key2"] = 2}
```

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#maps-and-records

Maps and records

- Accessing values

- Record - r.f
- Map - m[k]

```
function get_balance(a : address, accounts : map(address, account)) =
  accounts[a].balance
```

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#maps-and-records

Maps and records

- Updating values

- Record - r{f = v}
- Map - m{[k] = v}

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#maps-and-records

Maps and records

- Maps in the VM are implemented as hash maps and support fast lookup and update

- Large map size does not increase the gas cost for reading or updating it

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#maps-and-records

Maps vs Mappings - Solidity

```
// Solidity

mapping [string => uint] public balances;

balances[“milen”] = 2

//

contractName.balances[“milen”];

//

function balances(string _account) returns (uint) {

        return balances[_account];

}
```

Maps - Sophia

```
// Sophia


function get_balance(a: address, accounts: map(address, account)) =

    account[a].balance
```

Sophia - Maps built-in functions

Builtin functions on maps

The following builtin functions are defined on maps:

```
Map.lookup(k : 'k, m : map('k, 'v)) : option('v)
Map.lookup_default(k : 'k, m : map('k, 'v), v : 'v) : 'v
Map.member(k : 'k, m : map('k, 'v)) : bool
Map.delete(k : 'k, m : map('k, 'v)) : map('k, 'v)
Map.size(m : map('k, 'v)) : int
Map.to_list(m : map('k, 'v)) : list(('k, 'v))
Map.from_list(m : list(('k, 'v))) : map('k, 'v)
```

for vs switch - Solidity

```
// there is not an obvious number of iterations

// every step is paid

// you can fairly easy hit the gas limit

for     (uint x = 0; x < refundAddressesList.length; x++) {

        refundAddressesList[x].transfer(SOME_AMOUNT);

}
```

Sophia - loops

```
private function map(f : ‘a => ‘b, l : list(‘a)) : list(‘b) =
     switch(l)
          [] => []
          e :: l’ => f(e) :: map(f, l’)
```

State changing - Solidity

```
uint public firstStageDuration = 8 days;
uint public firstStagePriceOfTokenInWei = 85005100306018 wei;    //0.00008500510030601840 ETH per Token // 1176

uint public firstStageEnd;

uint constant public secondStageDuration = 12 days;
uint constant public secondStagePriceOfTokenInWei = 90000900009000 wei;     //0.00009000090000900010 ETH per To

uint public secondStageEnd;

uint constant public thirdStageDuration = 41 days;
uint constant public thirdStagePriceOfTokenInWei = 106258633513973 wei;          //0.00010625863351397300 ETH p

uint constant public thirdStageDiscountPriceOfTokenInWei = 95002850085503 wei;  //0.00009500285008550260 ETH pe

uint public thirdStageEnd;

uint constant public TOKENS_HARD_CAP = 500000000000000000000000000; // 500 000 000 with 18 decimals

// 18 decimals
uint constant POW = 10 ** 18;

// Constants for Realase Three Hot Hours
```

State changing - Sophia

```
put(state{ latest_auction_slot_id = next_auction_slot_id,
           auction_slots[next_auction_slot_id] = new_auction_slot })
```

Crypto

- Signature verification

Crypto.ecverify(msg: hash, pubkey: address, sig: signature) : bool

Crypto.ecverify_secp256k1(msg: hash, pubkey: bytes(64), sig: bytes(64)) : bool

https://github.com/aeternity/protocol/blob/master/contracts/sophia.md#cryptographic-primitives

Oracles

An Oracle operator will use the functions:

- `Oracle.register`
- `Oracle.get_question`
- `Oracle.respond`
- `Oracle.extend`

An Oracle user will use the functions:

- `Oracle.query_fee`
- `Oracle.query`
- `Oracle.get_answer`

https://github.com/aeternity/protocol/blob/master/oracles/oracles.md#oracles

To-Do List Smart Contract

```
contract ToDoList =
  record state = {
    index: int,
    tasks : map(int, task)}

  record task = {
    name: string,
    completed: bool }

  public stateful function init() =
    { index = 0,
      tasks = {}}

  public function get_tasks_count() : int =
    Map.size(state.tasks)

  public stateful function add_task(task : string) =
    let new_task = {
      name = task,
      completed = false }
    put(state{tasks[state.index] = new_task})
    put(state{index = state.index + 1})

  public stateful function complete_task(index : int) : bool =
    put(state{tasks[index].completed = true})
    true
```

```
  public function get_task_by_index(index: int) : string =
    switch(Map.lookup(index, state.tasks))
      None  => "No such task."
      Some(x) => x.name

  public function is_task_completed(index : int) : bool =
    switch(Map.lookup(index, state.tasks))
      None  => false
      Some(x) => x.completed
```

Thank you!

Resources

- https://aeternity.com
- https://github.com/aeternity/aeternity
- http://aeternity.com/documentation-hub
- https://github.com/aeternity/aepp-forgae-js
- https://github.com/aeternity/aepp-sophia-examples
- https://forum.aeternity.com
- https://github.com/aeternity/aepp-sdk-js


ÆSOLUTIONS

// CURRENT BLOCKCHAIN USE-CASES

Dynamic registry
Fractional investing

Marketplaces
„Programmed trust“

Identity
Health records

Payments infrastructure
Automated Invoice

Smart contracts
Music settlements

Supply Chain
proof of ownership

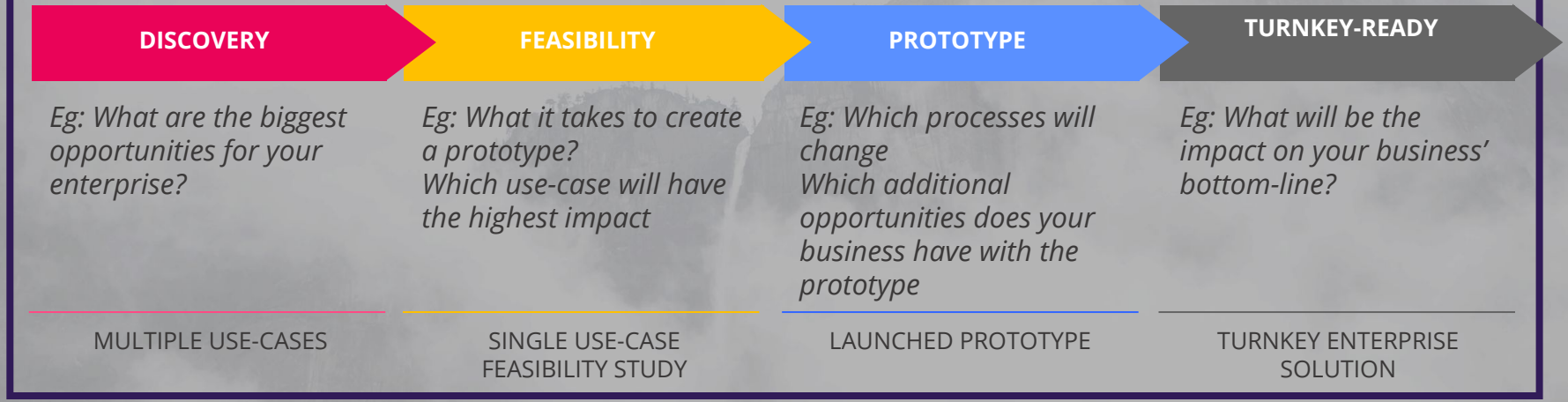
Automation
insurance claims

Blockchain technology has matured to distrupt business processes in many verticals.

ÆSOLUTIONS

- Key question based approach
- Expertise to handle any verticals
- Mrisk and increasing the business impact – step by step

CONSULTING-PROCESS



Coding Challenge

https://thepiwo.github.io/aepp-todolist