iExec

iExec - EthCC 2019

iExec ecosystem in Web3 perspective

Francois Branciard
fb@iex.ec
@fbranciard
www.iex.ec

Introduction

Substrate meets iExec: how to Build a Domain-Specific Token-Based Chain.

Abstract:
The iExec network needs a Domain-Specific chain to track deals between the different actors interacting on its marketplace for cloud resources (buyers and sellers of computing, datasets and applications). The chain would be used to record all contributions of these actors to trigger payments using PoCo (The iExec Proof-of-Contribution algorithm).
For private and consortium use of the iExec Stack, a PoA bridge and chain can be set up to offer an enterprise-ready solution.
For public usage, iExec has more complex requirements and is experimenting with Substrate to tackle: a token-based PoS consensus, governance, upgradability, the use of wasm for new features, and interoperability as a Polkadot parachain.

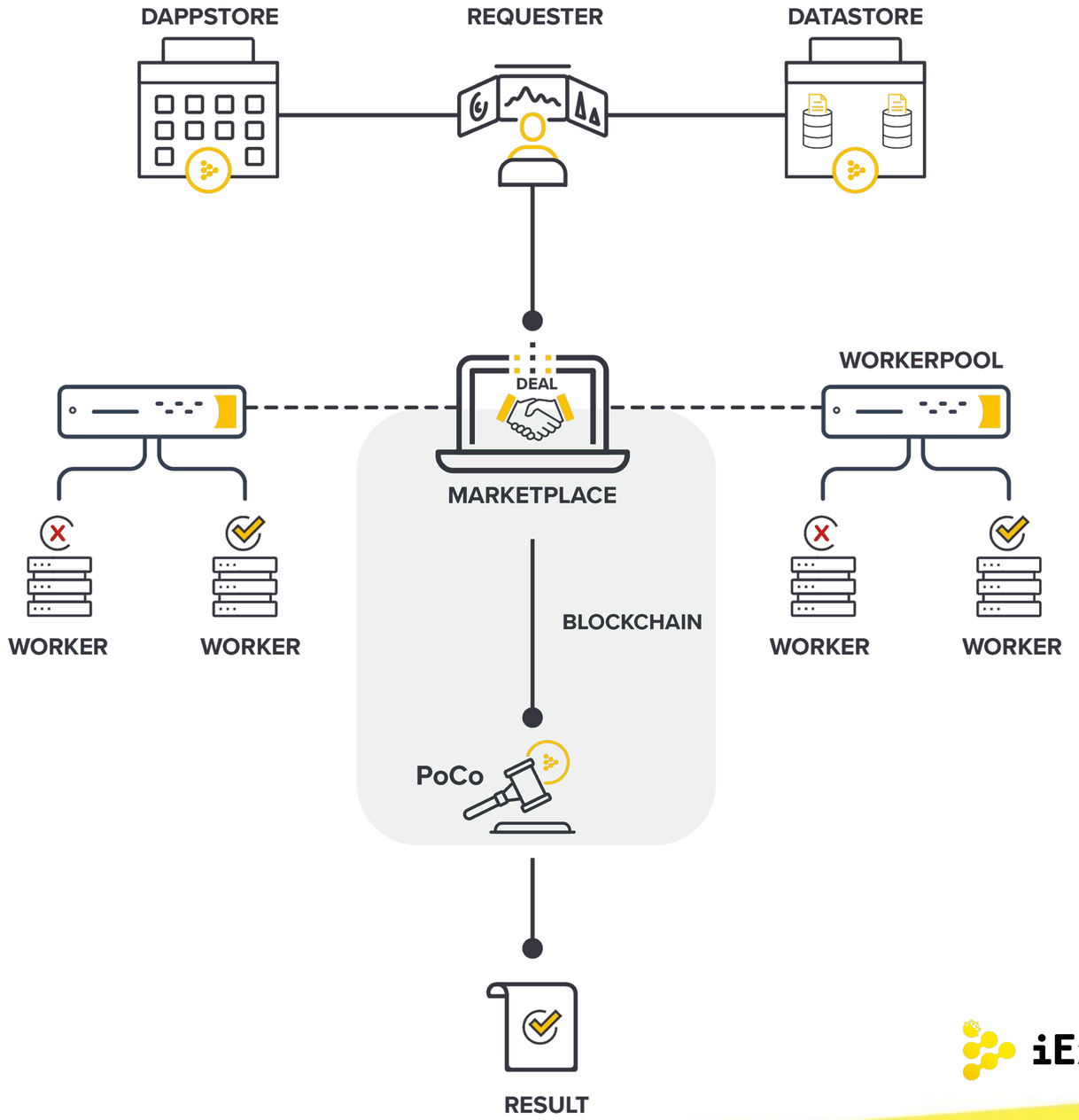
iExec

Plan

1. iExec ecosystem
2. iExec ecosystem in a web3 perspective
3. Delivery in progress : Consortium bridge
4. Research in progress : Substrate Domain-Specific Token-Base Chain

iExec

iExec ecosystem



iExec

Off chain compute

iExec ecosystem

- Only basic algorithms can be reasonably run in EVM
- Off chain compute to extend capacity of dapp
- Proof of Contribution : Economic games to preserve result trust *
- Marketplace resources : datasets, applications, servers providers

POCO

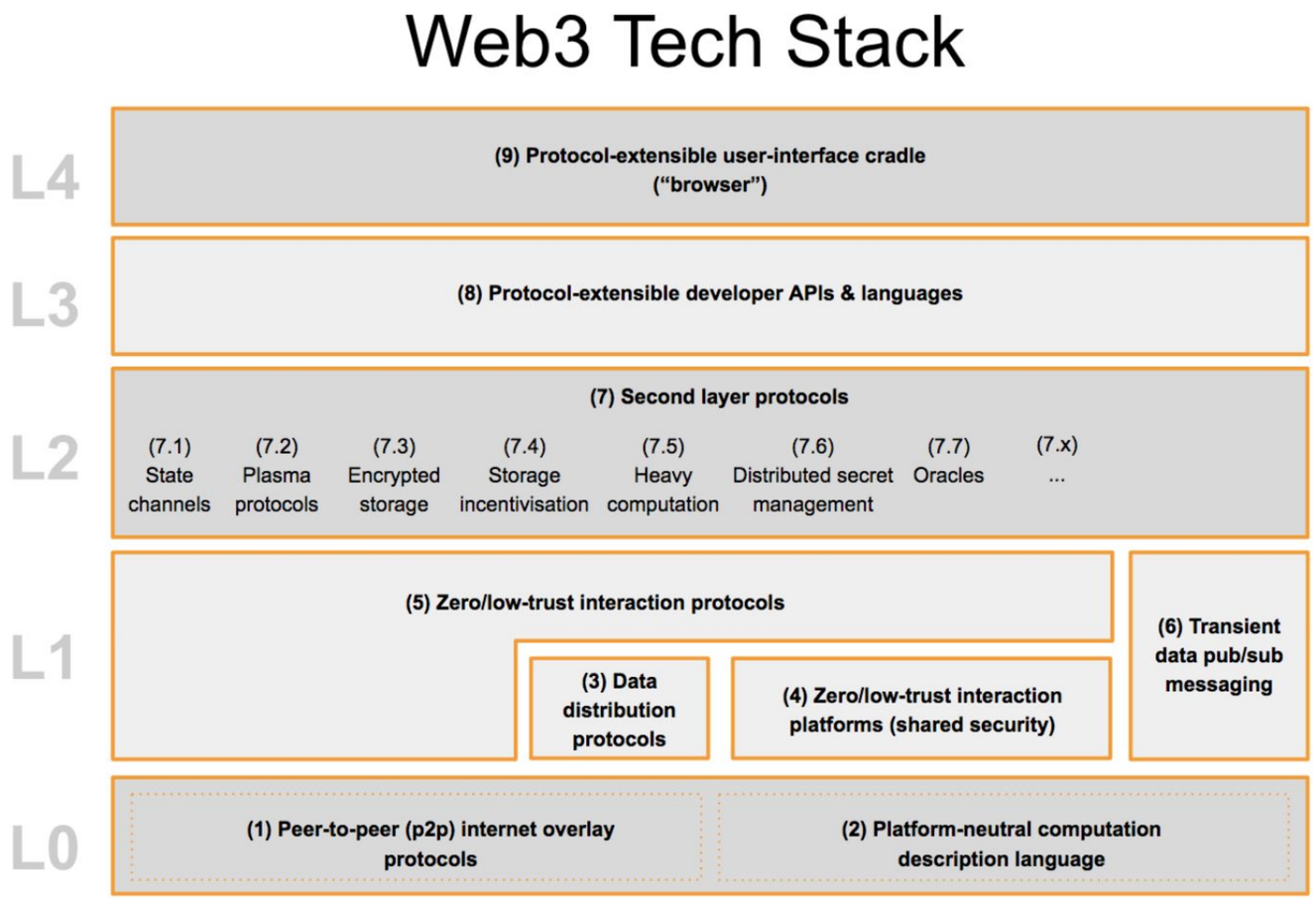
iExec

* https://docs.iex.ec/poco.html

Computation layer

computation layer for the decentralized web3 stack

iExec ecosystem in a web3 perspective



(7) Second layer protocols

(7.5) Heavy computation

iExec

Sidechain strategy

iExec ecosystem in a web3 perspective

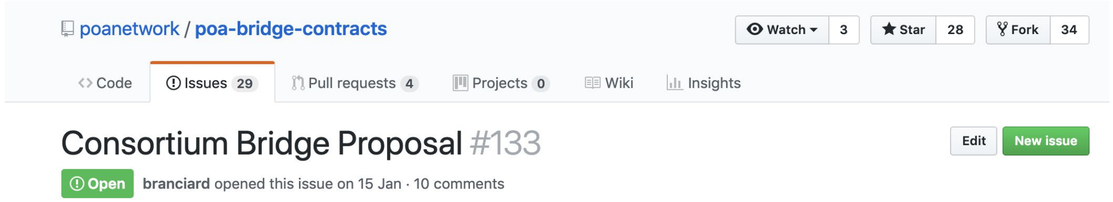
- **Deliver** pragmatic intermediary solution with existing technologies :
  - Mainnet EVM smart contract
  - PoA chain EVM smart contract
  - Bridges (EVM <-> EVM)
  - Mitigate governance with consortium
- **Research** on an optimal solution :
  - Replace bridges by dedicated relay-chain, with their own incentives, to link messages between chains. aka polkadot
  - Governance modules : block production, network upgrades. aka substrate modules
  - Owned autonomous domain chain incentive and governance

iExec

Delivery in progress : Consortium bridge

Consortium bridge

- ERC-20 ERC-20 Token bridge **Poa-network**
  Adding **Consortium Bridge feature** on Poa-network bridge
  - use Mainnet token asset in consortium
  - limit bridge responsibility for whitelisted addresses



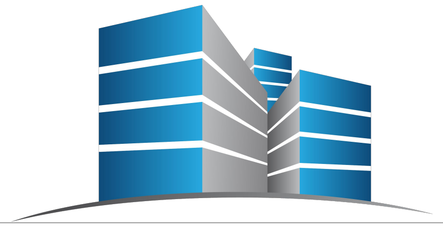
  - https://github.com/poanetwork/poa-bridge-contracts/issues/133
    https://forum.poa.network/t/consortium-bridge/1739

- Illustration of consortium bridge usage

Let’s build a consortium

Consortium bridge

Example of **Application verticals**



ROLE	Data Provider	App Provider	Resource Provider
Expertise	A company that have dataset that can be used to train AI model	A company that provide algorithm to trained model.	A company that has idle GPU resources
Authority			

iExec

Let’s build a consortium

Consortium bridge

- PoA **chain** under authority :

- Dataset Provider deploys **dataset** with iExec Stack

- Application Provider deploys **app** with iExec Stack

- Resource Provider creates **workerpool** with iExec Stack

- PoCo Transactions : shared **auditability** of **usage** between parties

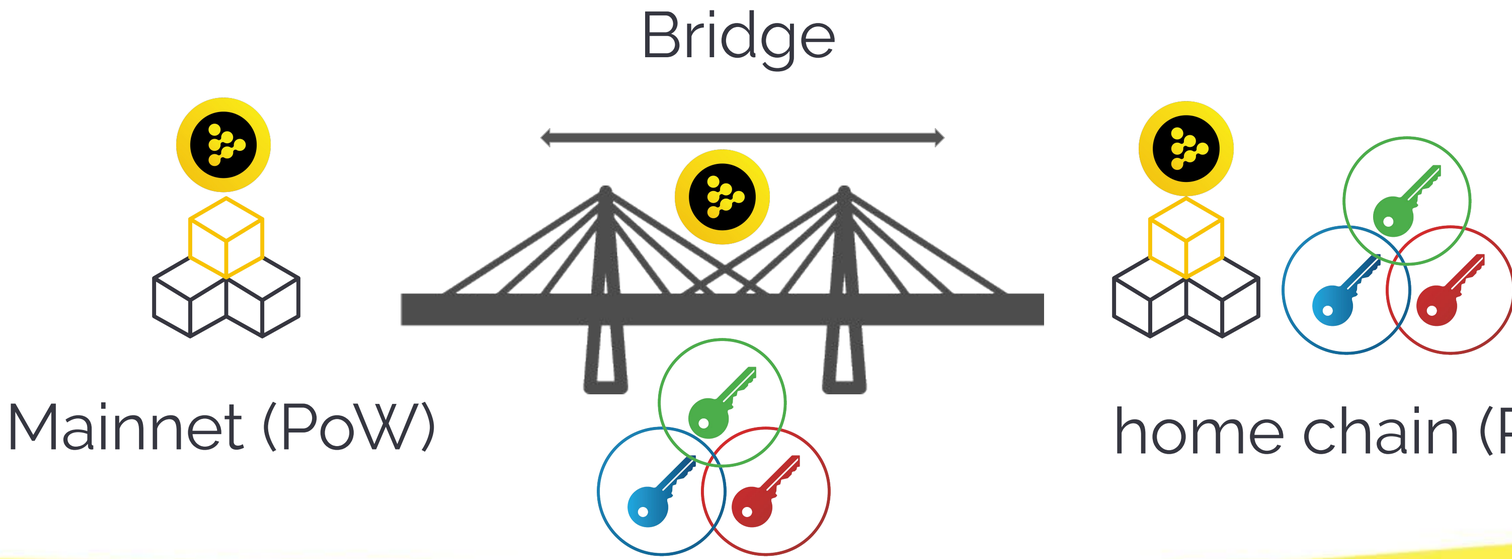
POCO

iExec

Let’s automate token payment between them

Consortium bridge

- add a **consortium bridge** under same authority than can **deposit** and **withdraw** between the mainnet and their “home” chain.
- use marketplace contract to set dataset, app, resources dynamic **prices usage between them**.

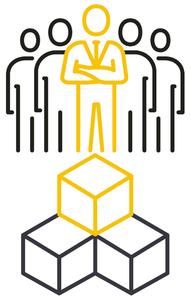


iExec

Let’s automate payment for their join customers too

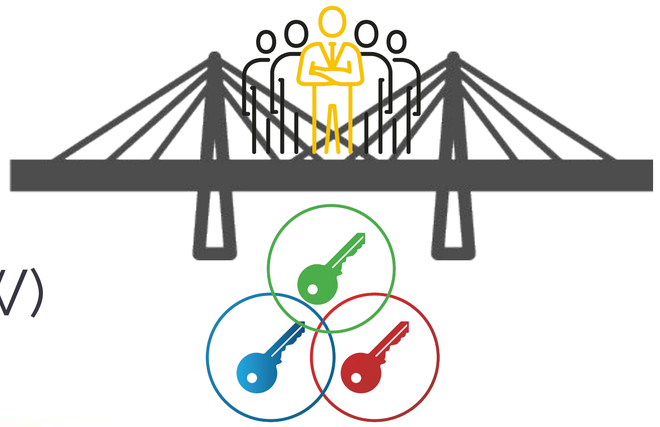
Consortium bridge

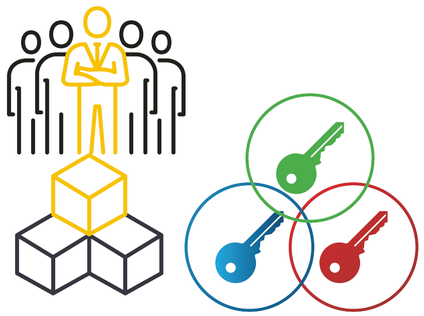
-  a potential customer of 
- **IF** ( **trust** ) **AND** ( **WHITHLIST** )



Mainnet (PoW)

Bridge





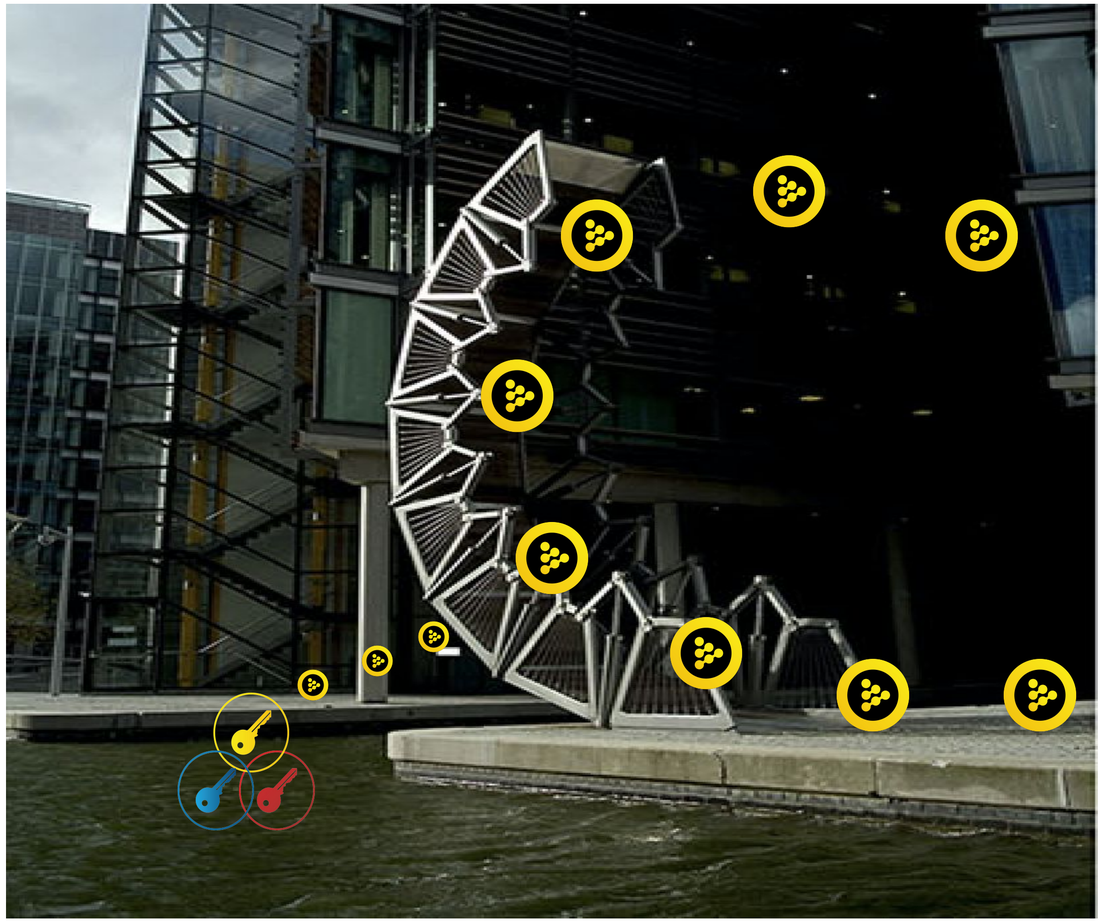
home chain (PoA)

iExec

What happens if you are not whitelisted to cross consortium bridge ?

Consortium bridge

consortium side



Mainnet side

asset return back
(minus bridge fees)

Research in progress : Substrate POC

Substrate POC

- From substrate-template-node providing base modules:
  - System, Timestamp, Consensus, Balances etc ...

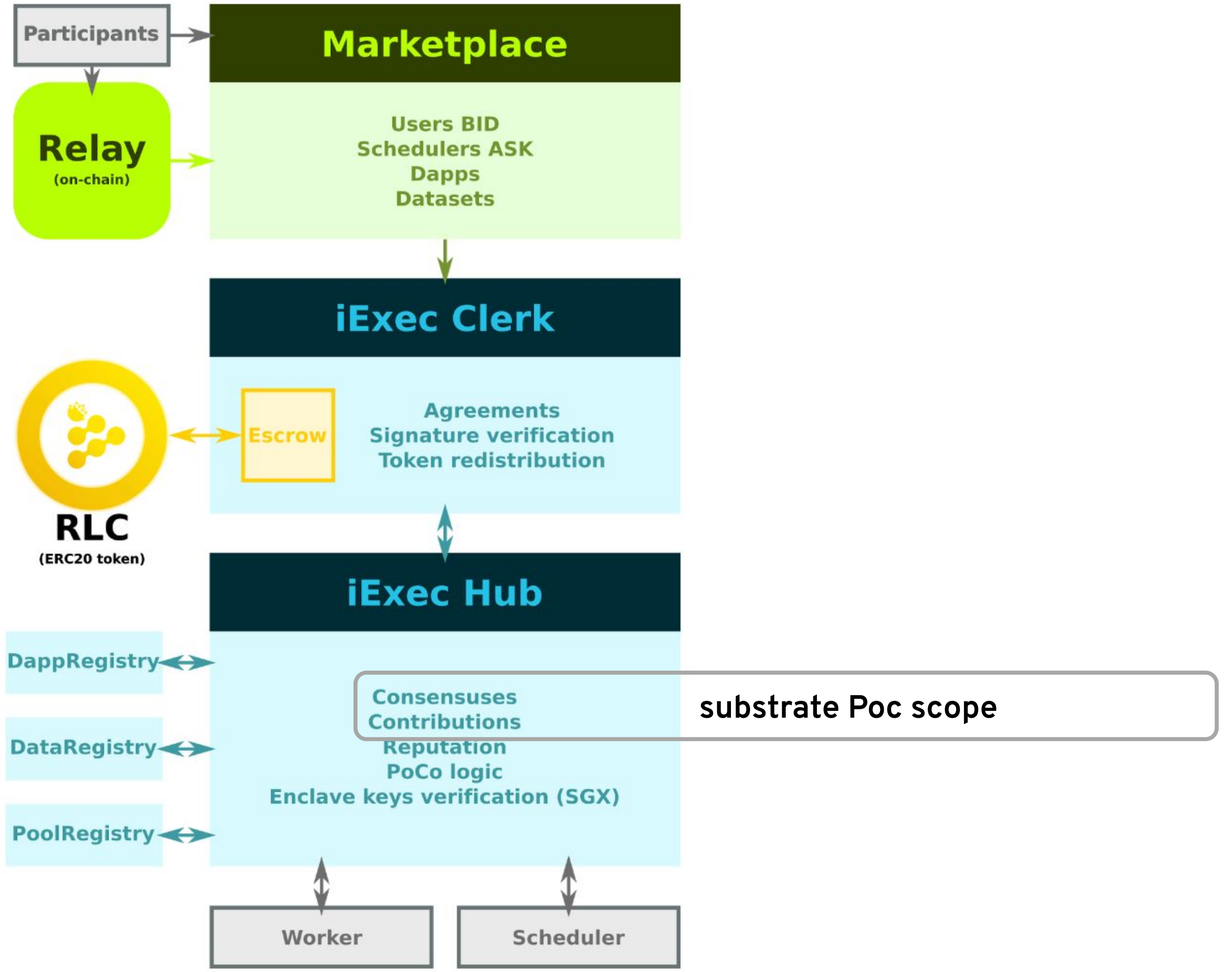
- Custom modules can be added, iExec first poc scope :
  - modelize Tasks and workers contributions
  - contribute, reveal schemas with economic games incentive.

- Objective :
  - familiarized with wasm (next blockchain standard to come ?)
  - runtimes vs smart contract feedback
  - explore what on-chain governance modules can offer

iExec

First poc scope

Substrate POC



iExec

Contribute / Reveal

Substrate POC

1. Off chain Task : echo “hello-world”
2. contribute :
   - **stake** to **contribute**
   - contribution 2 params :
     - result vote : vote on a result but do not reveal it yet.
     - contribution seal : needed to prove result knowledge later
3. consensus reached on vote
4. reveal
   - **reveal** and **reward/slash**
   - reveal 1 param :
     - contribution unseal

.

iExec

Contribute / Reveal

Substrate POC

1. Off chain Task : echo “hello-world”
2. contribute :
   - **stake** to **contribute**
   - contribution 2 params :
     - **result vote** : hash(hash(hello-world))
     - **contribution seal** : hash(@worker ^ hash(hello-world))
3. consensus reached on vote
4. reveal
   - **reveal** and **reward/slash**
   - reveal 1 param :
     - **contribution unseal** : hash(hello-world)
       - check result vote was valid
       - contribution seal was valid.

iExec

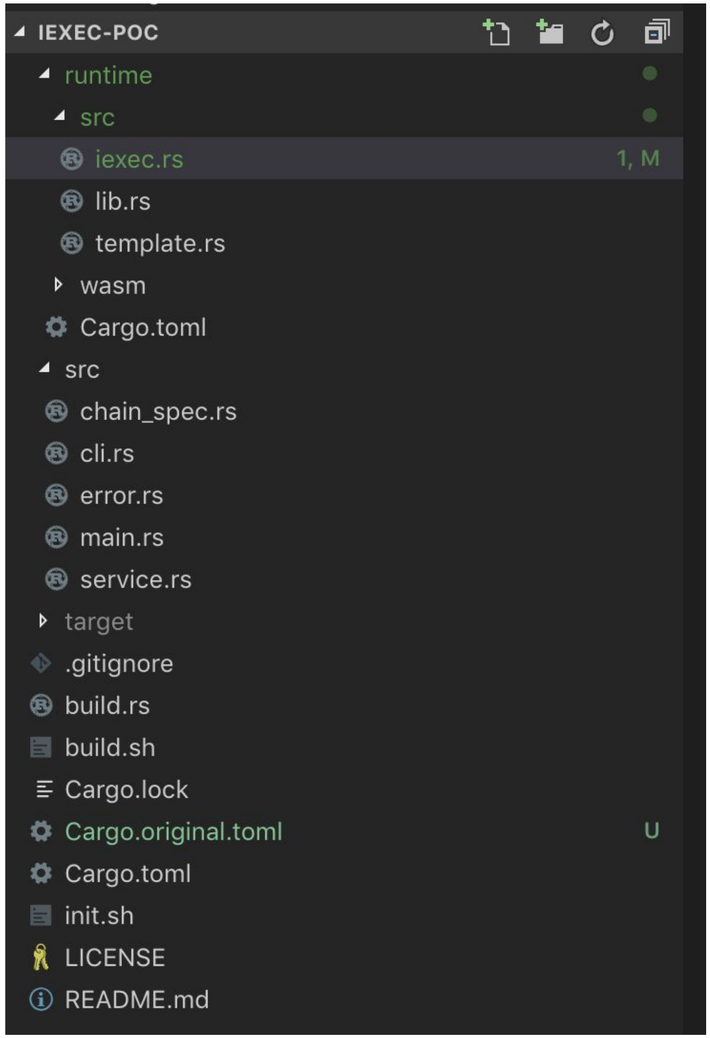
Demo

Substrate POC

iExec

From substrate node template

Substrate poc




```
construct_runtime!(
    pub enum Runtime with Log(InternalLog: DigestItem<Hash, Ed25519AuthorityId>) where
        Block = Block,
        NodeBlock = opaque::Block,
        UncheckedExtrinsic = UncheckedExtrinsic
    {
        System: system::{default, Log(ChangesTrieRoot)},
        Timestamp: timestamp::{Module, Call, Storage, Config<T>, Inherent},
        Consensus: consensus::{Module, Call, Storage, Config<T>, Log(AuthoritiesChange), Inherent},
        Aura: aura::{Module},
        Indices: indices,
        Balances: balances,
        Sudo: sudo,
        Fees: fees::{Module, Storage, Config<T>, Event<T>},
        // Used for the module template in `./template.rs`
        TemplateModule: template::{Module, Call, Storage, Event<T>},
        // Used for the module iexec in `./iexec.rs`
        IexecModule: iexec::{Module, Call, Storage, Event<T>},
    }
);
```

iExec

Struct Model

Substrate poc

```
#[derive(Encode, Decode, Default, Clone, PartialEq)]
pub struct Task<Hash> {
    id: Hash,
    threshold:u64,
    // simplify replication for this poc of https://docs.iex.ec/pocosrc/poco-trust.html#trust2018
    //https://github.com/iExecBlockchainComputing/iexec-doc/raw/master/techreport/iExec_PoCo_and_trustmanageme

}

#[derive(Encode, Decode, Default, Clone, PartialEq)]
pub struct Contribution<Hash> {
    id: Hash,
    task_id: Hash,
    result_vote: Hash,
    result_seal: Hash,
}
```

iExec

Define Storage

Substrate poc

```
/// This module's storage items.
decl_storage! {
    trait Store for Module<T: Trait> as IexecModule {
        // Just a dummy storage item.
        // Here we are declaring a StorageValue, `Something` as a Option<u32>
        // `get(something)` is the default getter which returns either the stored `u32` or `None` if nothing stored
        Something get(something): Option<u32>;
        Tasks get(task): map T::Hash => Task<T::Hash>;
        Contributions get(contribution): map T::Hash => Contribution<T::Hash>;
        ModuleSalt: u64;

        TasksConsensus get(task_consensus): map T::Hash => T::Hash;

        AllTasksCount get(all_tasks_count): u64;
        AllTasksArray get(task_by_index): map u64 => T::Hash;
        AllTasksIndex: map T::Hash => u64;

        ContributionsArray get(task_contributions_by_index): map (T::Hash, u64) => T::Hash;
        ContributionsCount get(task_contributions_count): map T::Hash => u64;
        ContributionsIndex: map T::Hash => u64;


        ContributionsResultVoteCount get(task_contributions_result_vote_count): map (T::Hash, T::Hash) => u64;


    }
}
```

iExec

Define functions

Substrate poc

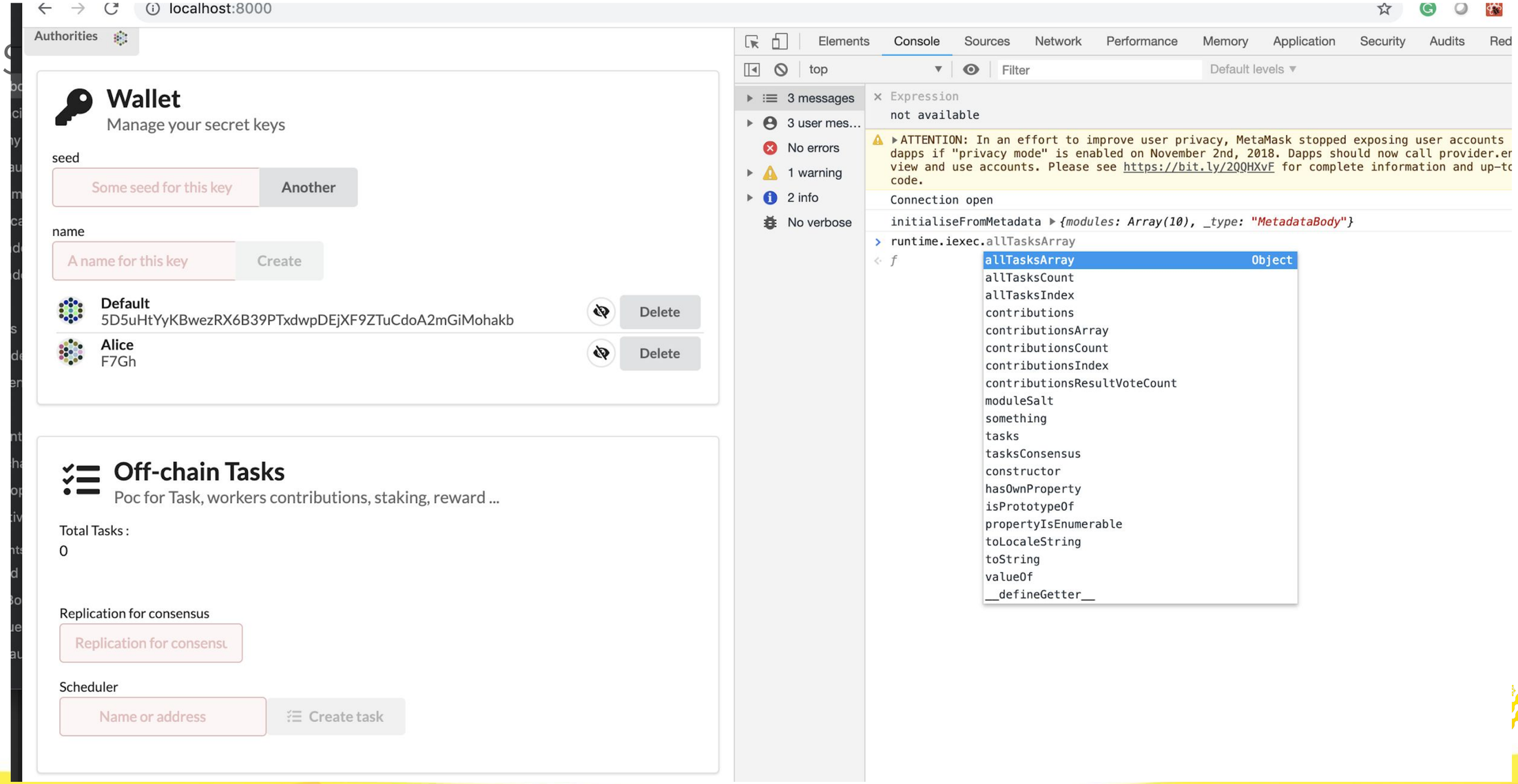
```
pub fn create_task(_origin,_threshold: u64) -> Result {
```

```
pub fn contribute(_origin,_task_id: T::Hash, _result_vote: T::Hash,_result_seal: T::Hash) -> Result {
```

```
pub fn reveal(_origin,_task_id: T::Hash,_result_unseal: T::Hash) -> Result {
```

iExec

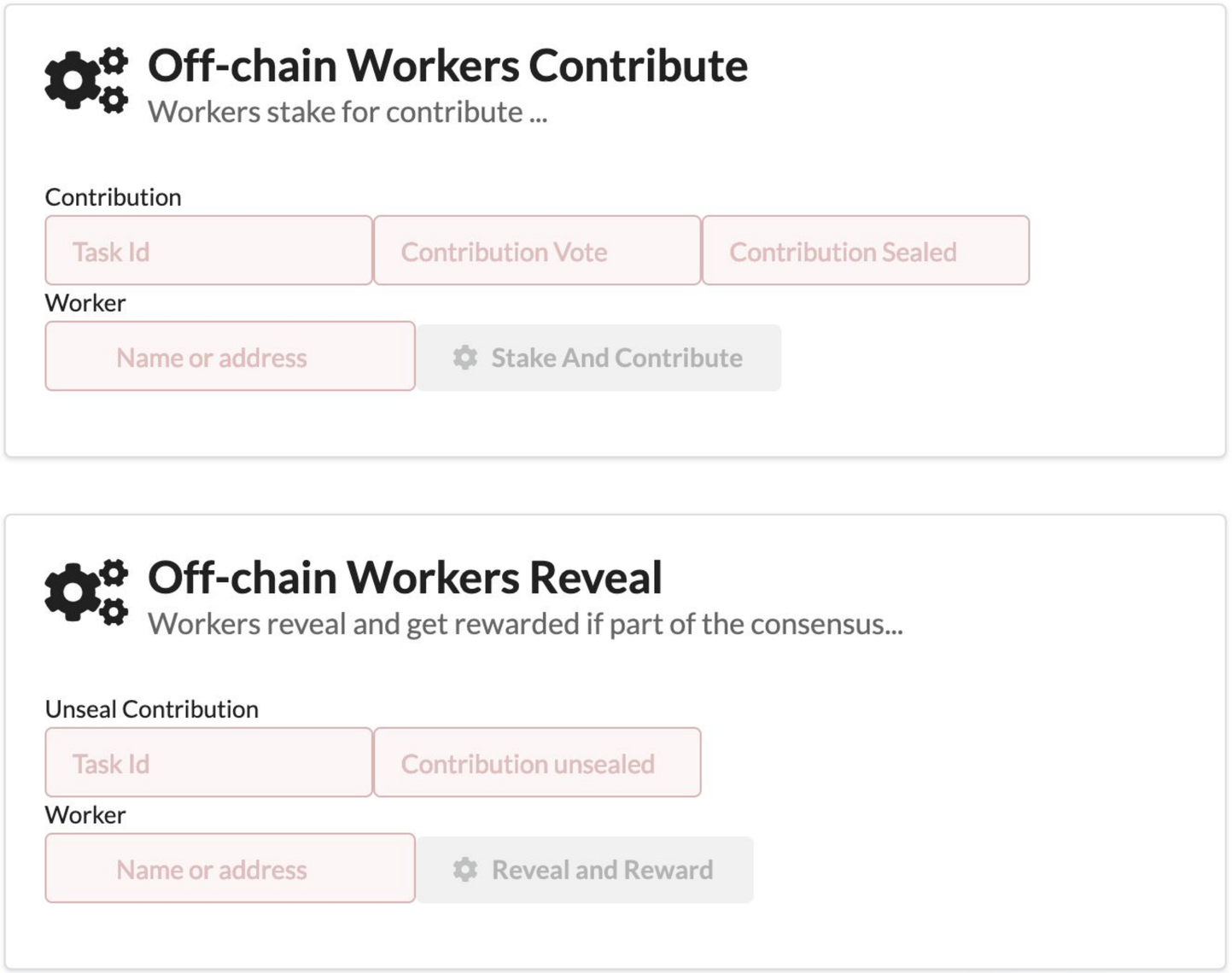
Substrate-ui



iExec

Contribution reveal

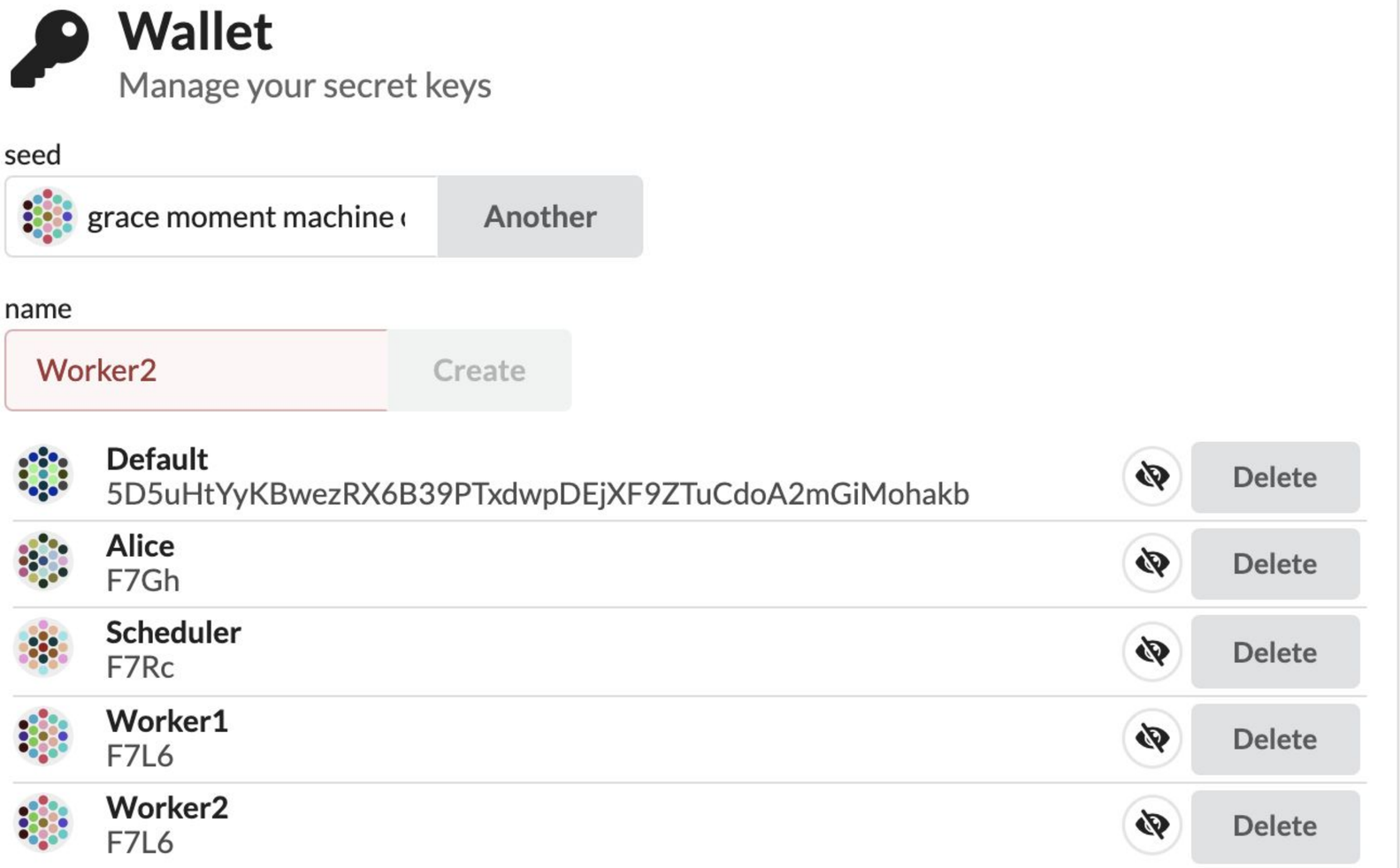
Substrate poc



iExec

Actors

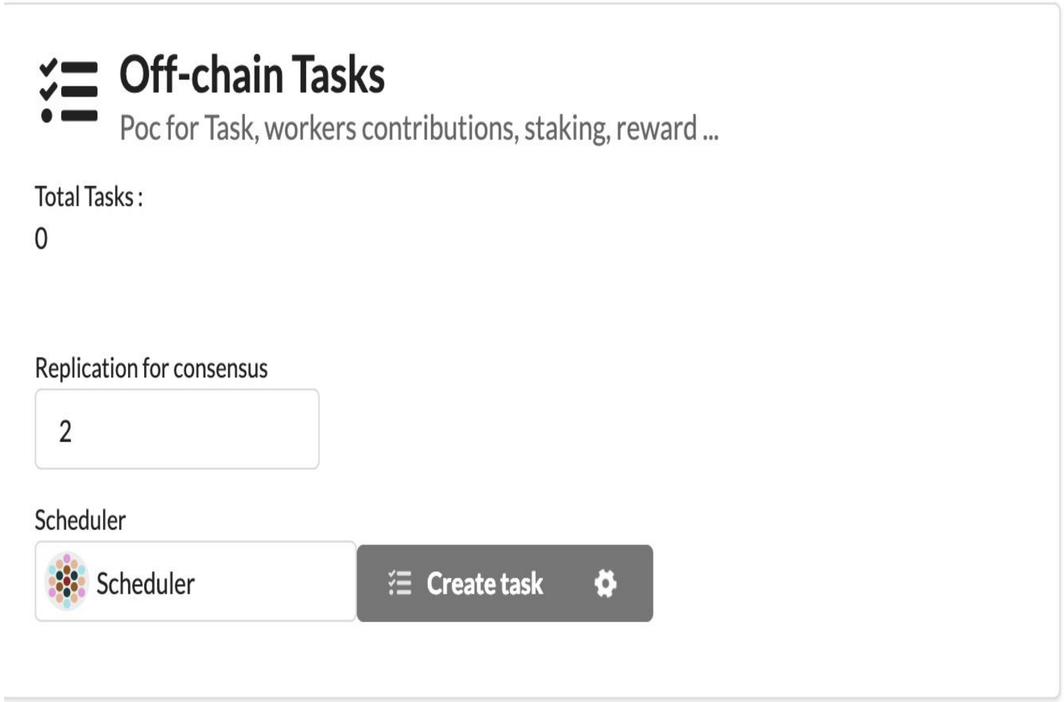
Substrate poc

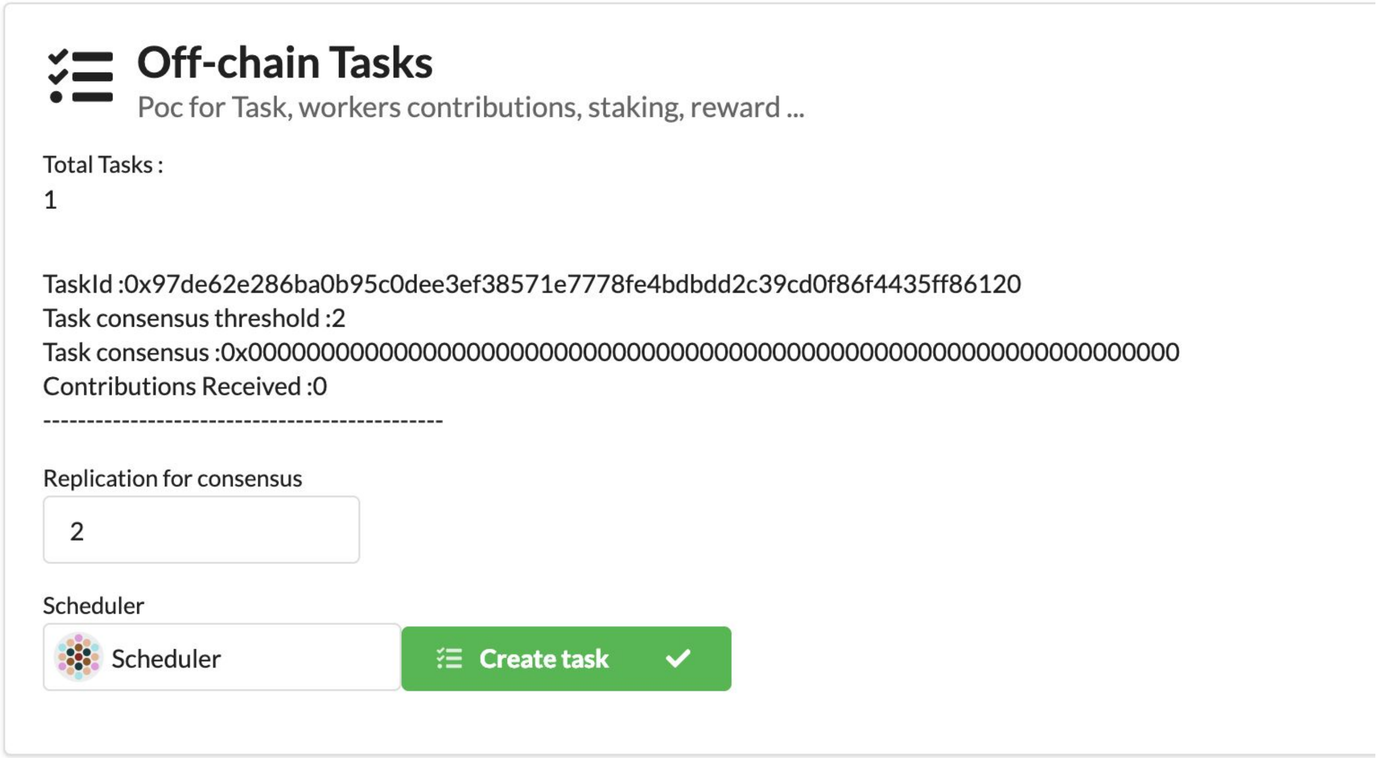


iExec

Task creation

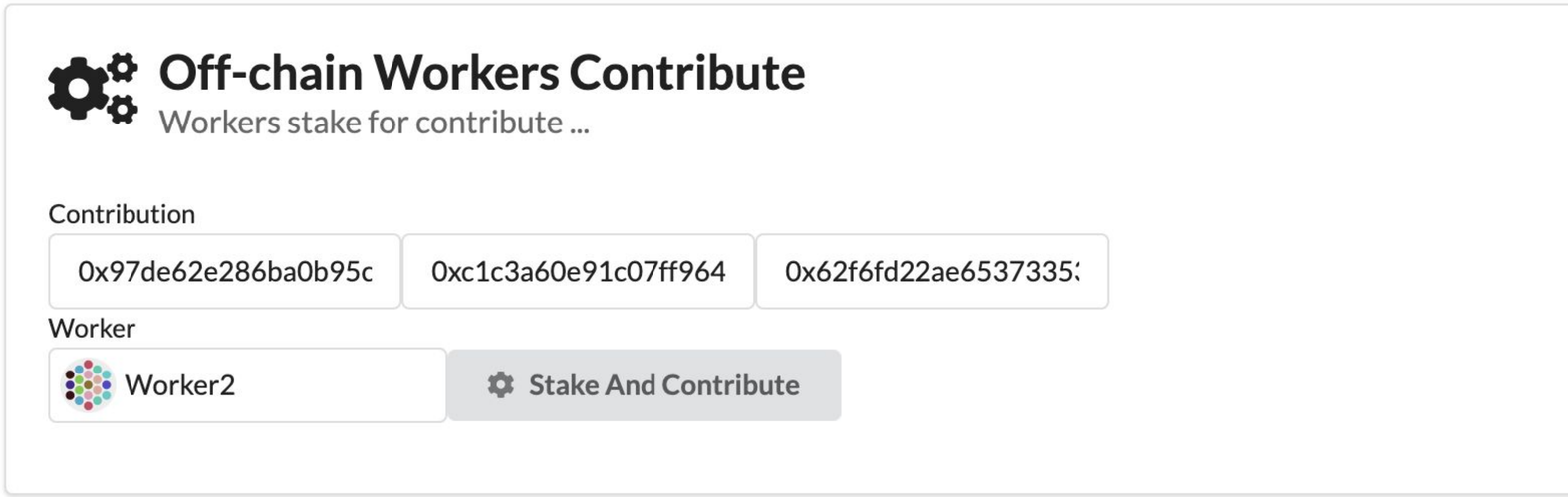
Substrate poc





Worker contribute (stake)

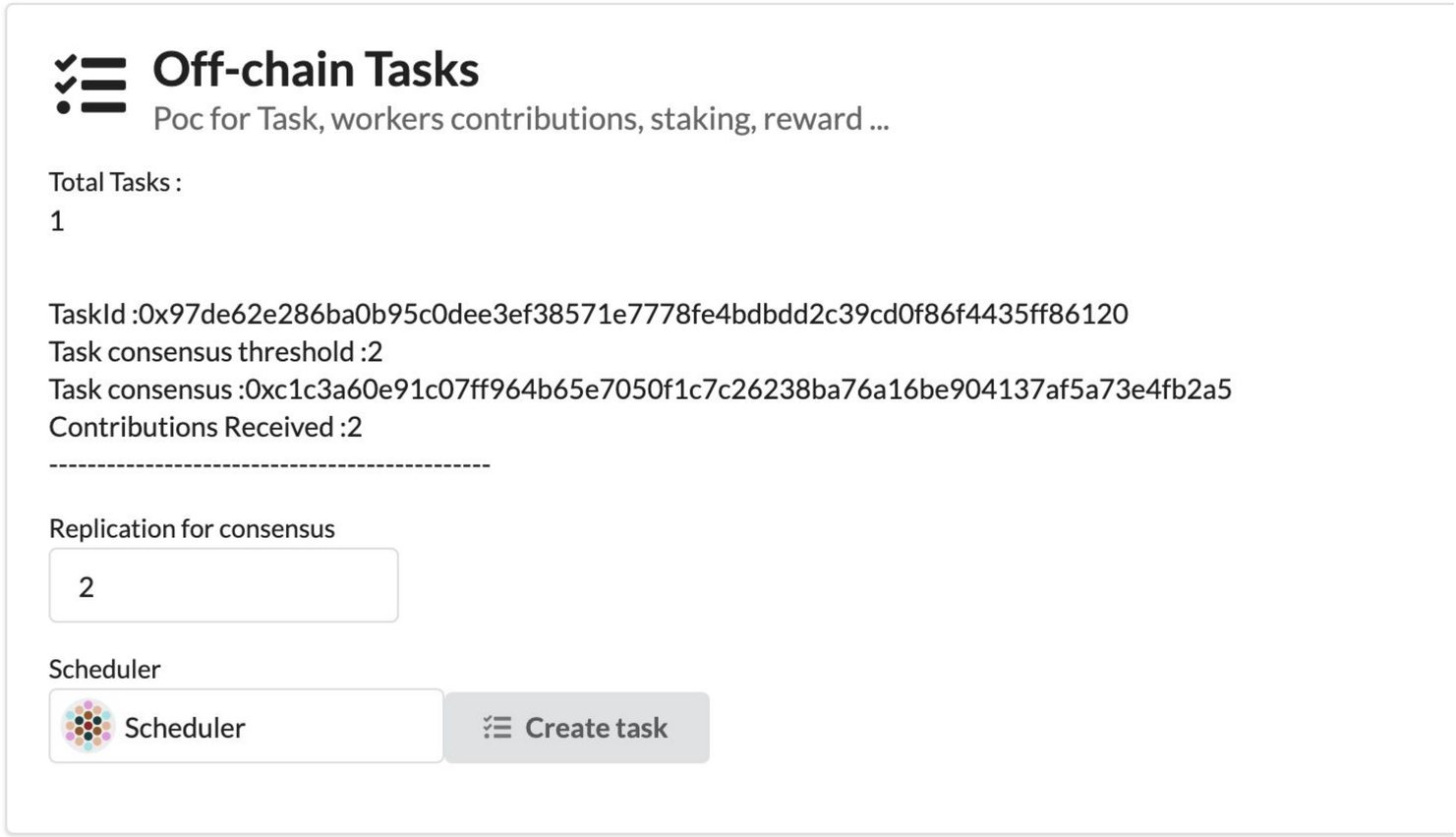
Substrate poc



iExec

Consensus Reached

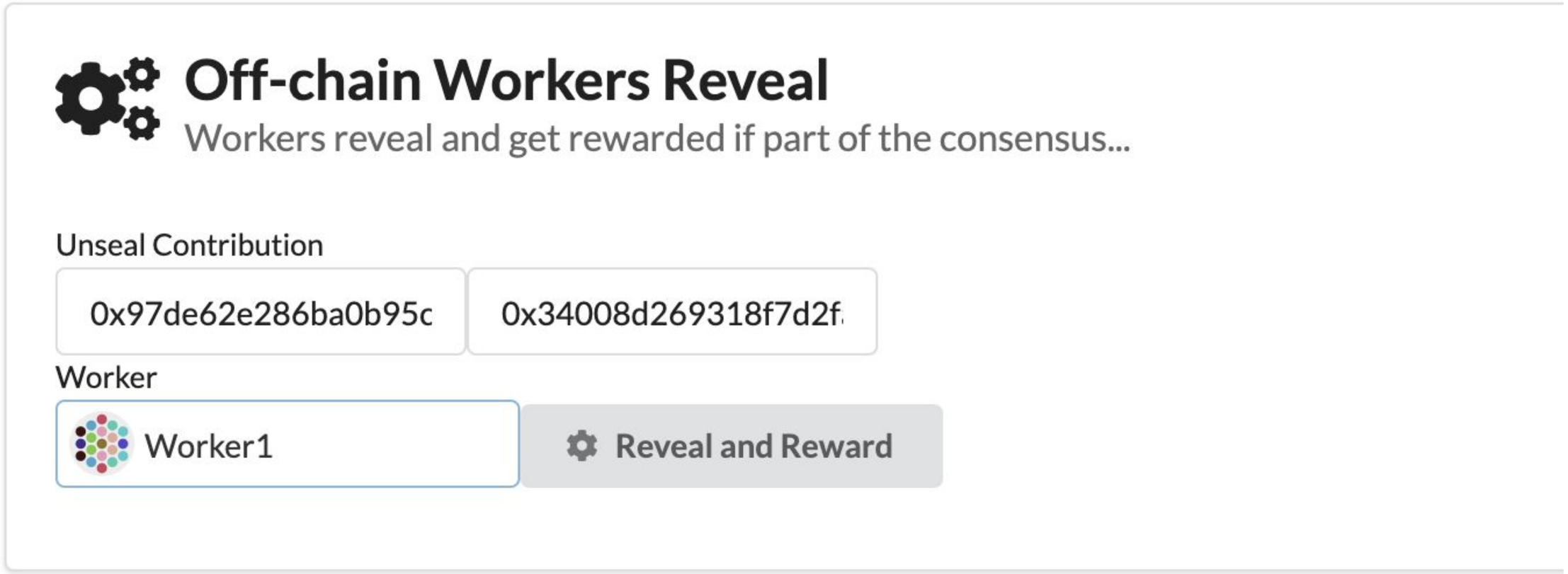
Substrate poc



iExec

Worker Reveal (reward or slashing)

Substrate poc

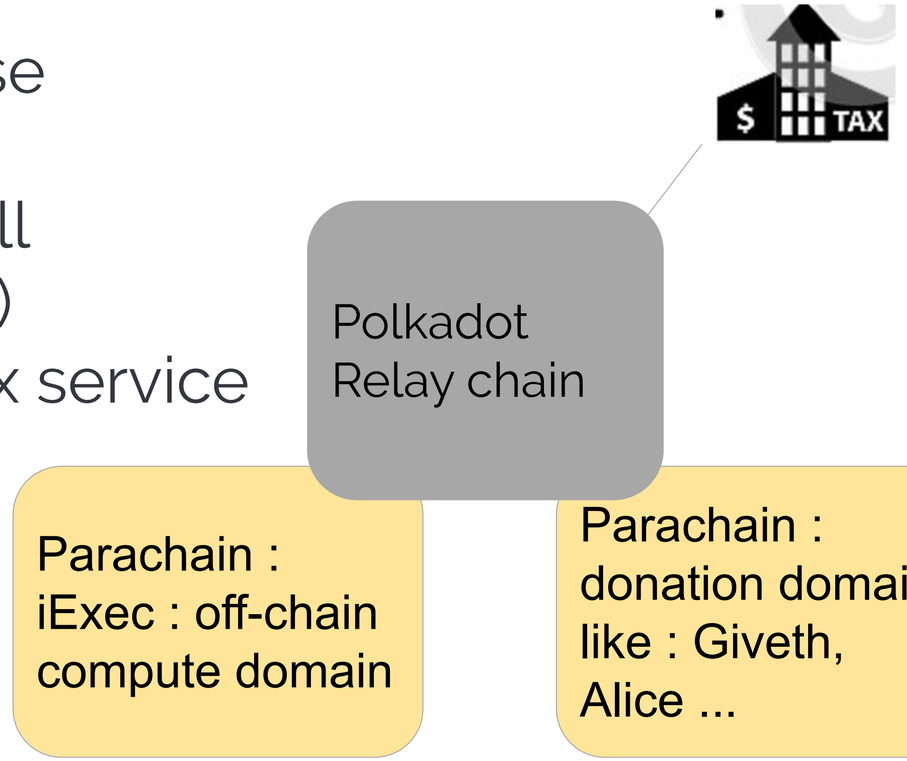


iExec

Speculative idea with interoperability

Substrate poc

- Different parachain typology (logic, consensus, security, governance, performances)
- But interoperability enable new use case
- DCO : Donate computation offering ?
  - Auditable proof that fund raised is well used (app medical compute research)
  - External proof in the relay chain for tax service to have tax deduction.
    (IDEN3 non-reusable proof like :))



iExec

Common vision and synergy

Substrate meets iExec | iExec meets Substrate

Substrate

Off-chain compute

Allows for off-loading jobs outside of the runtime

- Chain can schedule tasks to be executed by a recognised group
- Task definition stored as Wasm on-chain
- Additional APIs for this Wasm... URL fetch, local storage, IPFS
- Create economic games on top to ensure fidelity of result

POCO

iExec

Credit and extract from Gavin Wood talk at M-1 | Asset Management 3.0 Conference | Day 2
https://www.youtube.com/watch?v=Opj_5ORbyXA

iExec

Thank you !

Francois Branciard
fb@iex.ec
@fbranciard
www.iex.ec