SCEW: Programmable BFT-Consensus for Client-Centric P2P Web Applications

PaPoC ’21

Martijn Sauwens, Kristof Jannes, Bert Lagaisse, Wouter Joosen

26 April 2021

eLoyalty: Shared Loyalty Program

Integrated Loyalty programs
Redeem loyaly points at any participating store

! Decentralized: no central authority
Merchants do not fully trust eachother

! Double-spending problem
No customers may spend the same loyalty point

eShare: Sharing Economy

Tool Sharing Platform
Small communities share tools and track them

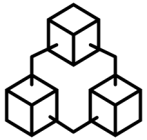
! Lack of trust between participants
Tools can be stolen, damaged, lost, …

! Decentralized Tracking
Whereabouts of tools must be tracked reliably

Application Challenges

Managing shared assets

Securing assets with real world value and consequences



Peer-to-Peer applications

Networks of mutually distrusting parties

Real-World interactions

Applications supplementary to interactions in the real world

Ease-of-Use

Non-expert target audience

State-of-the-art

Peer-to-Peer data synchronization frameworks
Automerge, Legion, OWebSync, Yjs

Blockchains
Bitcoin (PoW), Ethereum (PoW + Smart Contracts),
Hyperledger Fabric (BFT + Smart Contracts)

Consensus for the browser
“You Don’t Need a Ledger”

SCEW: Programmable BFT-Consensus with Smart Contracts for Client-Centric P2P Web Applications

A programming framework for lightweight consensus

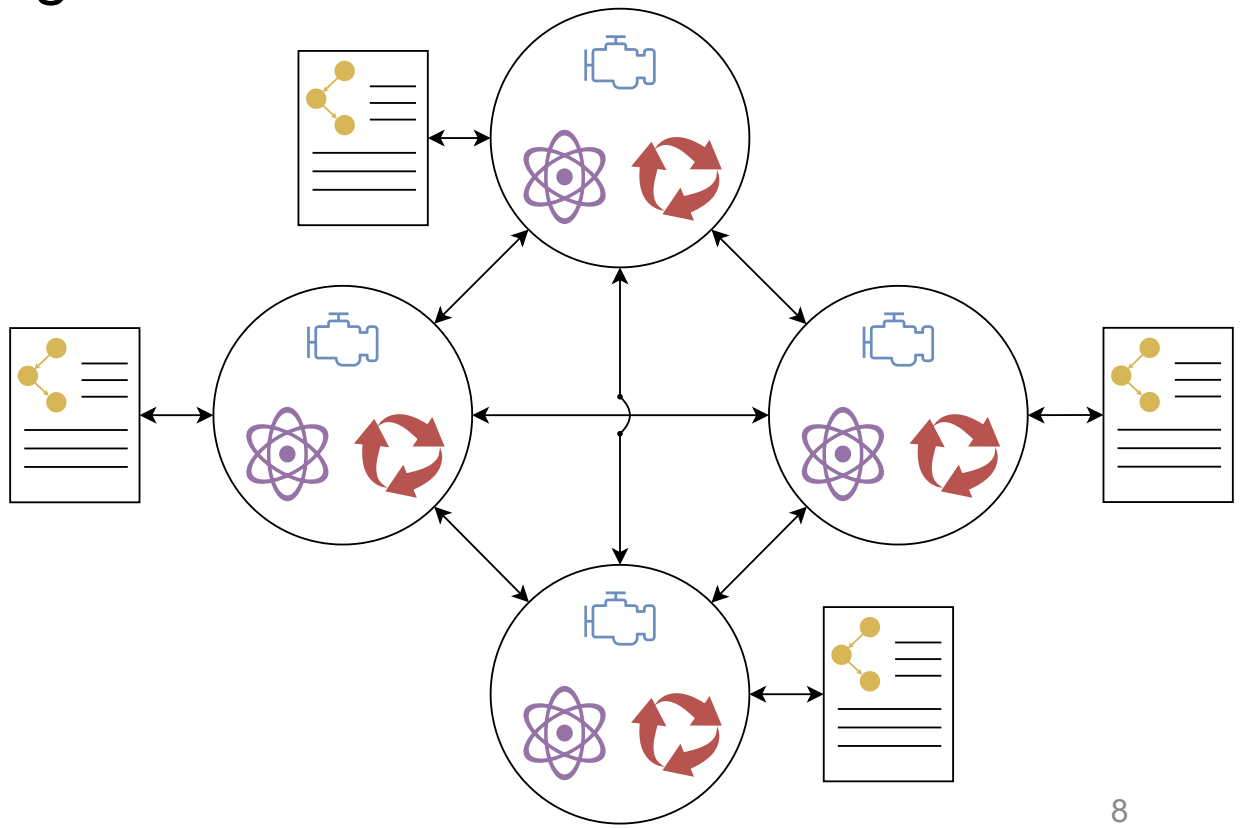
Architecture and programming interface

Evaluation

Performance and overhead analysis

Taking a step back

Future work and conclusion



SCEW: Programmable BFT-Consensus with Smart Contracts for Client-Centric P2P Web Applications

A programming framework for lightweight consensus

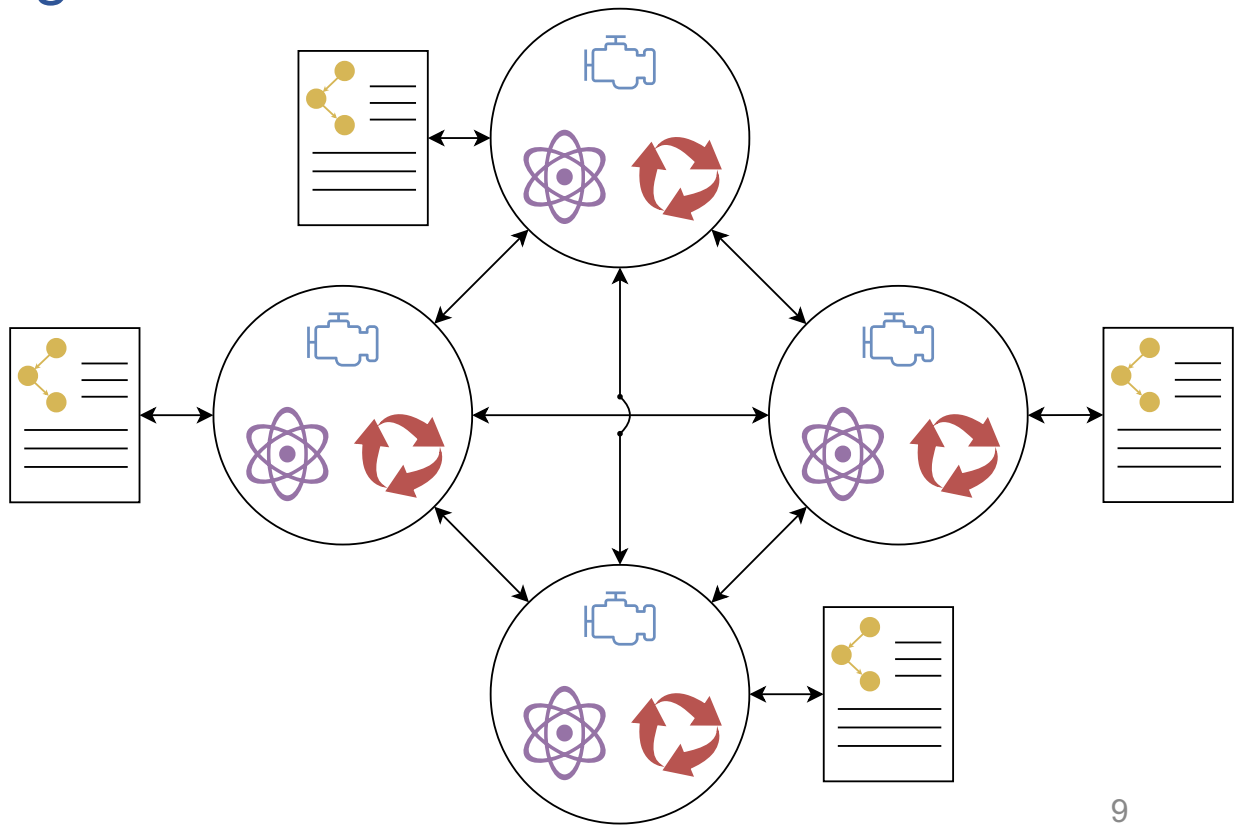
Architecture and programming interface

Evaluation

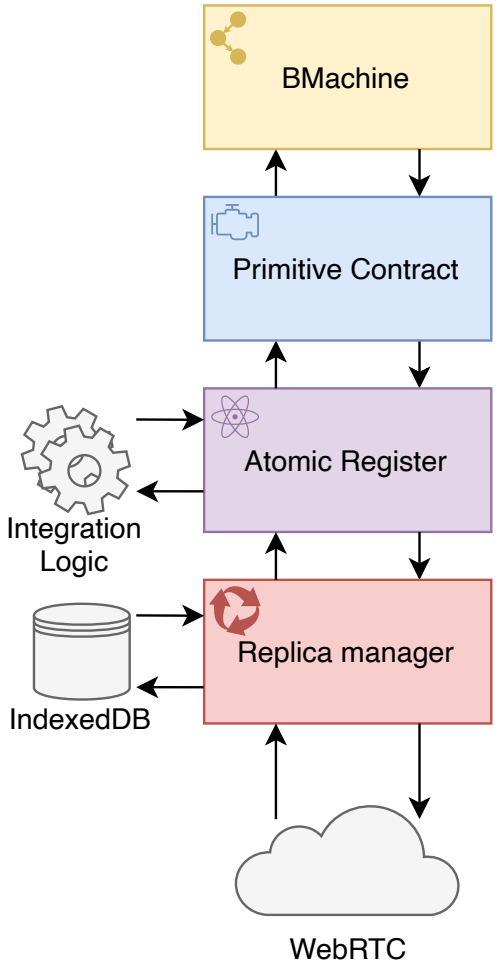
Performance and overhead analysis

Taking a step back

Future work and conclusion



SCEW: A Programming Framework For Lightweight Consensus



State-Based approach to asset management

Programmable, Byzantine Fault Tolerant, Lightweight

Smart Contracts

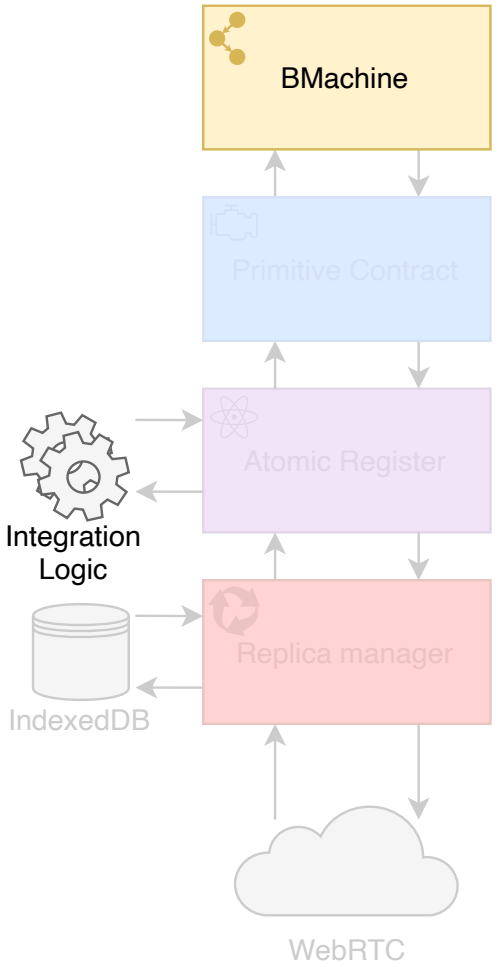
Model asset life-cycle

Atomic Registers

Own and represent a single asset

Protect against arbitrary and Byzantine faults

Developer Point of View



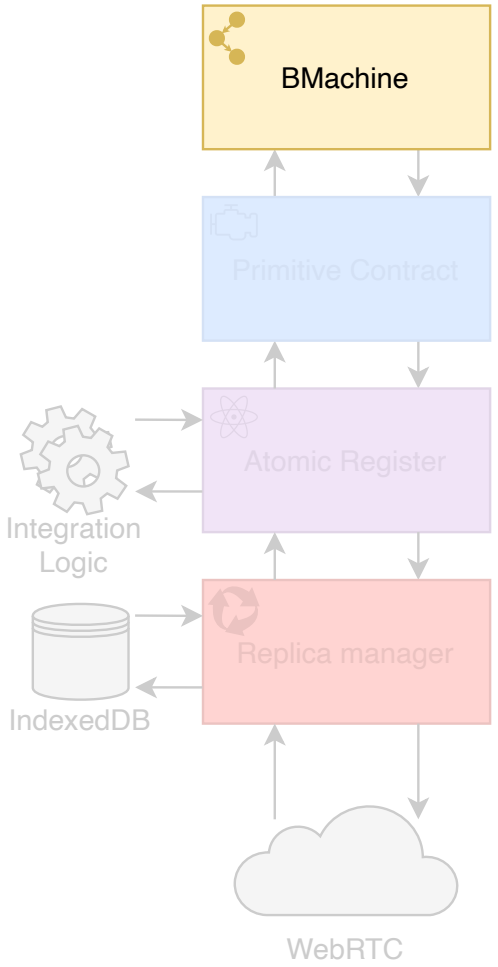
Write BMachine Smart Contract

Describe asset life-cycle as FSM

Write Integration Logic

- Initiating state transitions by calling contract
- Reading value of current state from register

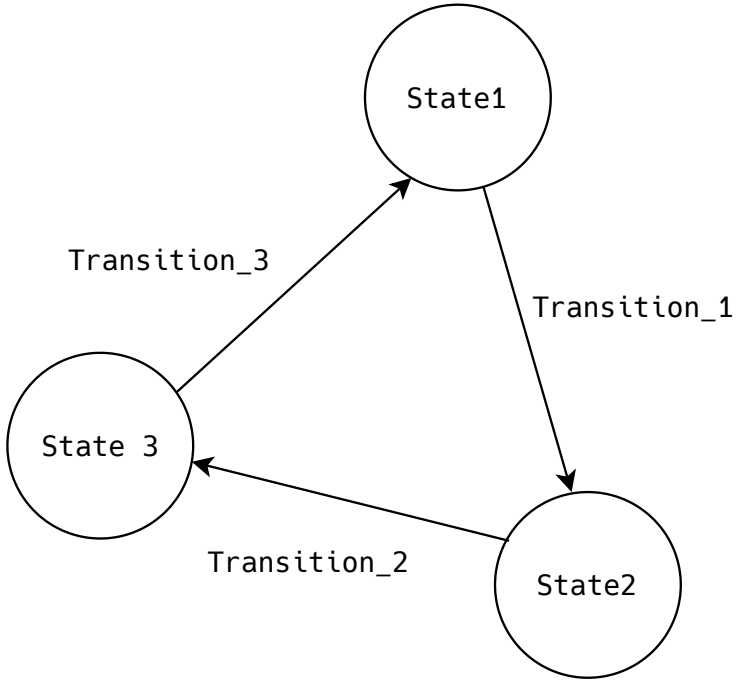
Smart Contracts



BMachine Smart Contract

Finite state machine modelling asset life-cycle

Smart Contracts




```
Transition = {
  guard: (State, Input, Ctx) =>  bool,
  effect: (State, Input, Ctx) => State
};

State = { name: string, value: Value };
```

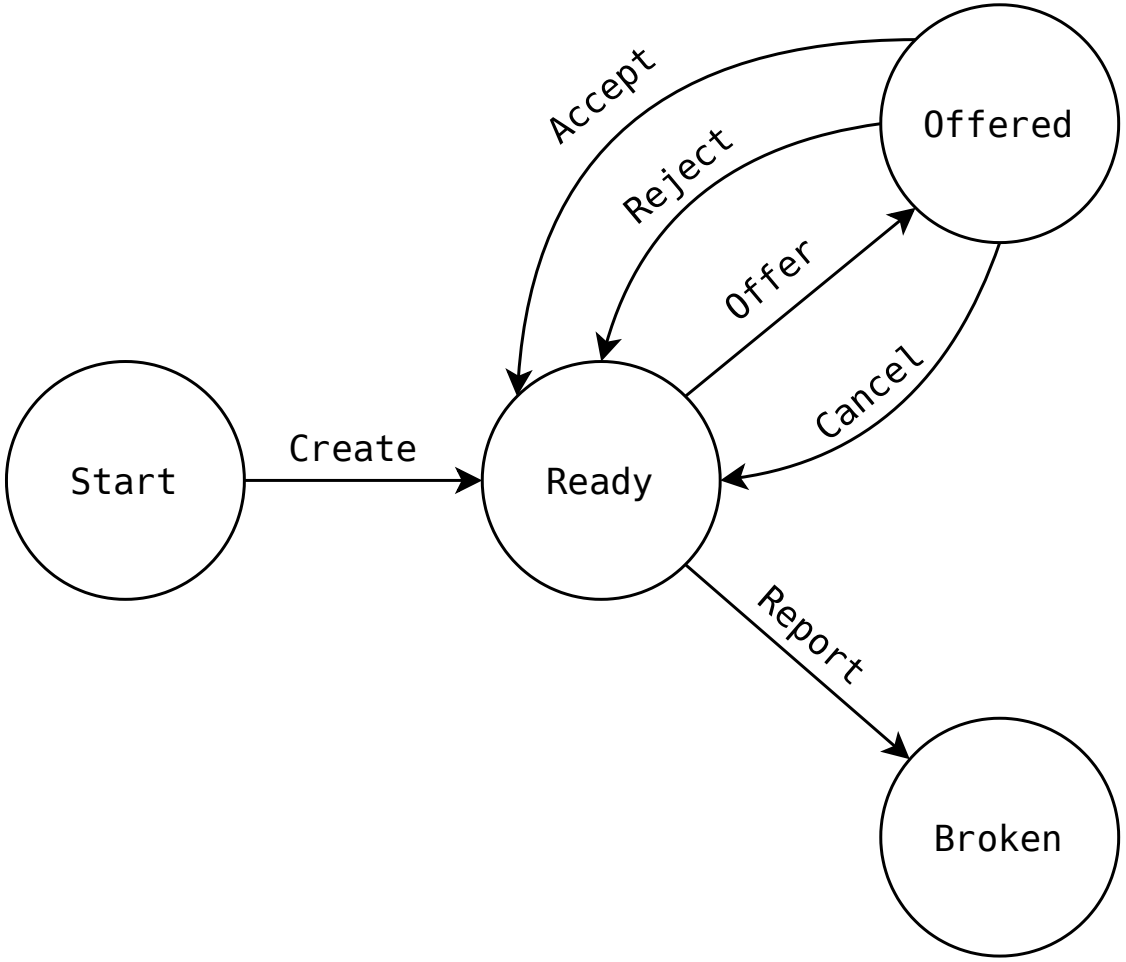
BMachine Smart Contract

Finite state machine modelling asset life-cycle

BMachine Definition:

- BMachine states
  Values and lifecycle phases
- BMachine transitions
  Transformations of asset value
  Precondition: guard
  Postcondition: effect

Smart Contracts: eShare



Model life-cycle of a tool

Creation, Tracking, Out-of-Order

Creation

Start → Ready

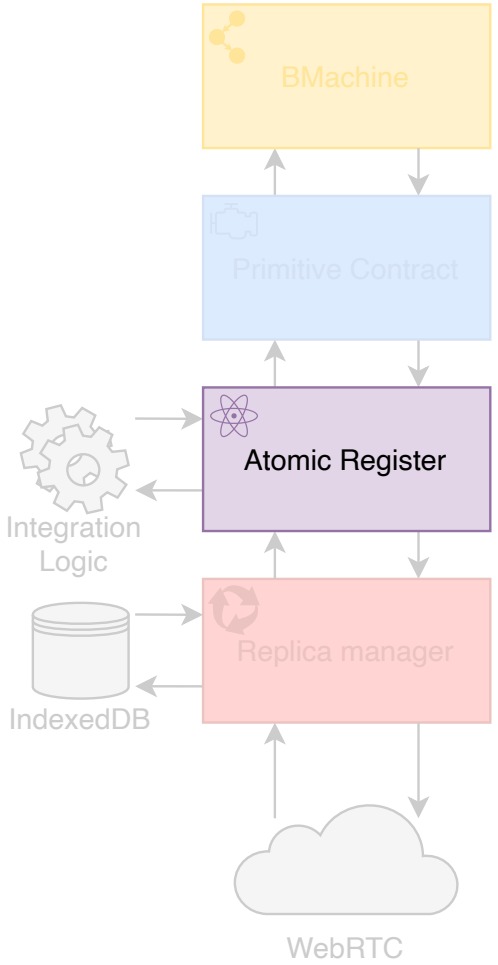
Tracking

Ready ↔ Offered

Out-of-Order

Ready → Broken

Atomic Registers



Stores individual assets

Synchronization and protection

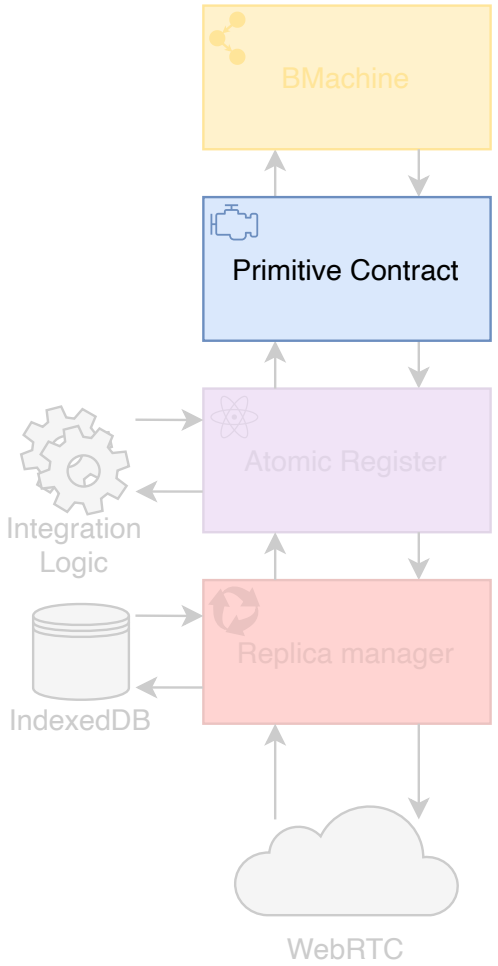
State-Based CRDT

- Robust synchronization of single assets
- Reduce communication with Merkle Trees

Shared Asset Protection

Through BFT-Consensus and signed proposals

Primitive Contract



Adaption Layer

High level State Machines vs Register

Encoding BMachine State

Retrieve state and call transitions

Handle BMachine Transitions

As proposals for Atomic Register

SCEW: Programmable BFT-Consensus with Smart Contracts for Client-Centric P2P Web Applications

A programming framework for lightweight consensus

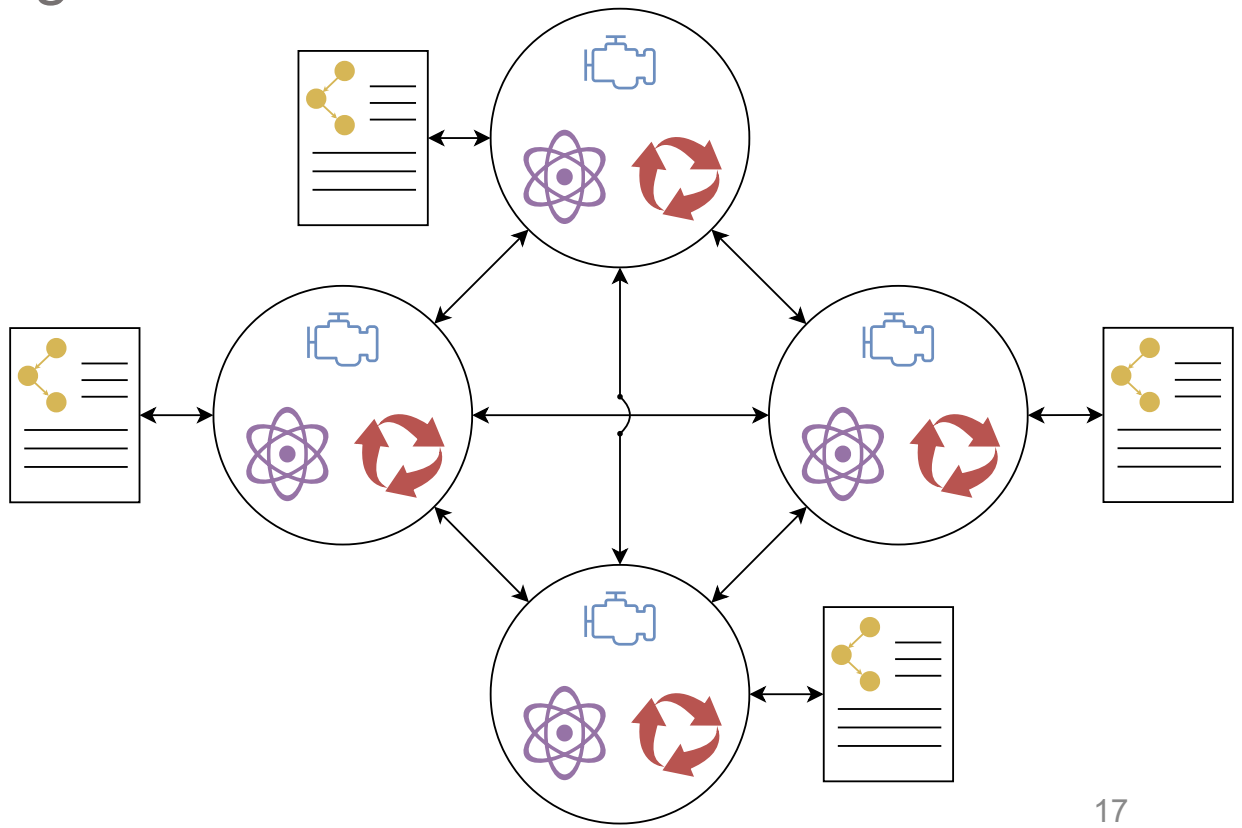
Architecture and programming interface

Evaluation

Performance and overhead analysis

Taking a step back

Future work and conclusion



Experimental Setup

eShare use-case

Users share tools at fixed transaction rate

Performance at Scale

Scale up to 100 browser instances

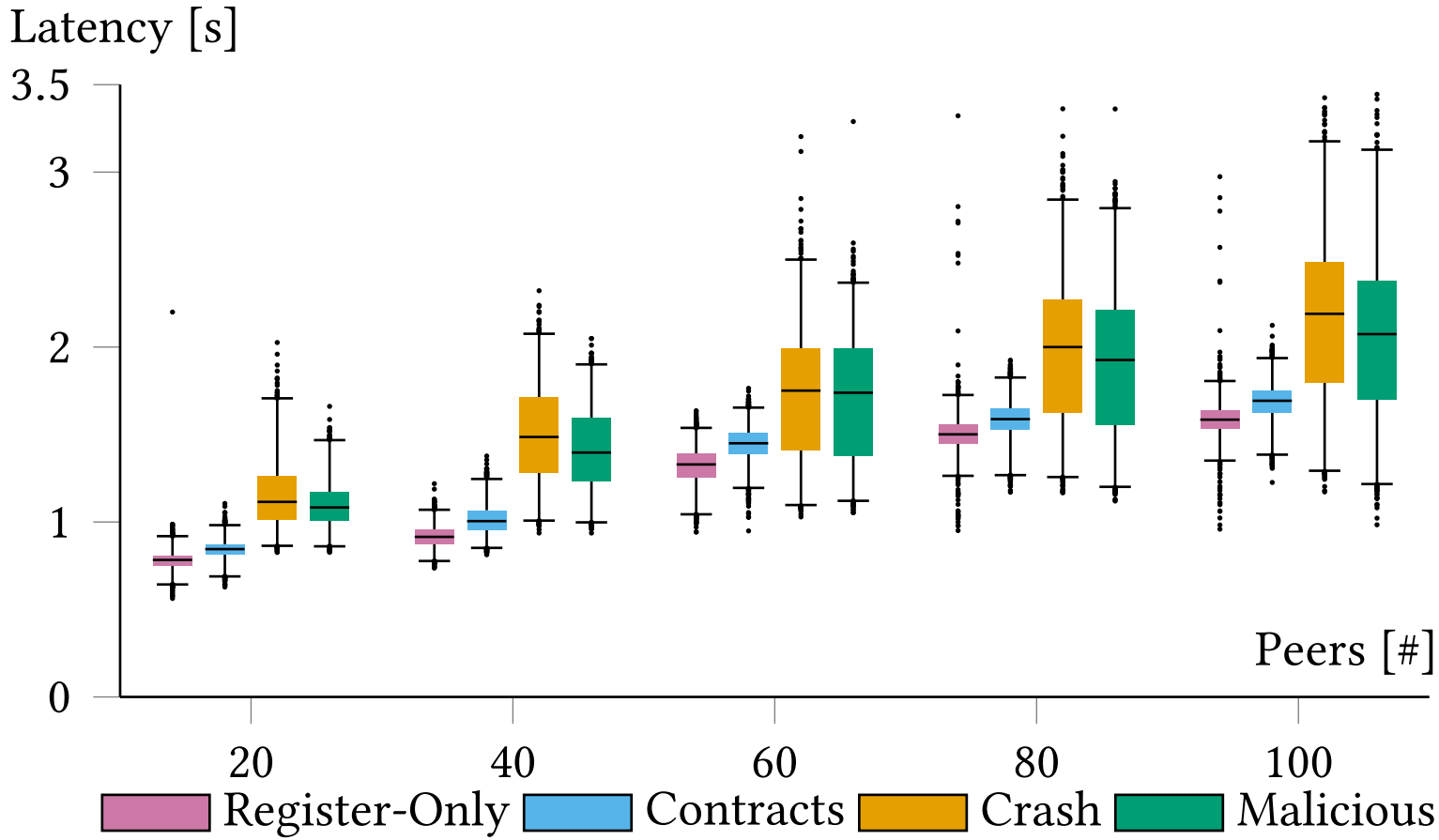
Overhead Analysis

eShare application with and without contracts

[Byzantine] Fault Tolerance at Scale

Crashes and invalid proposals

Evaluation Results



SCEW: Programmable BFT-Consensus with Smart Contracts for Client-Centric P2P Web Applications

A programming framework for lightweight consensus

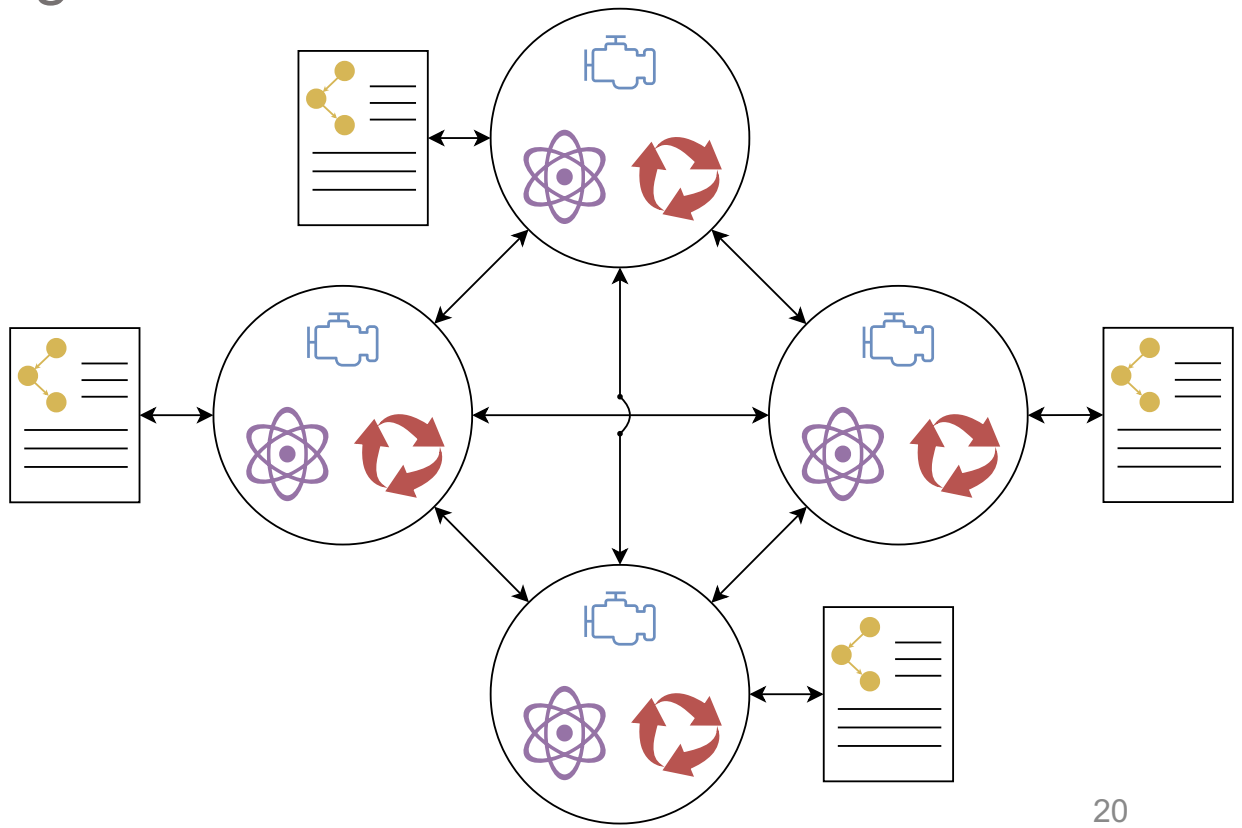
Architecture and programming interface

Evaluation

Performance and overhead analysis

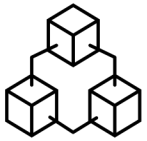
Taking a step back

Future work and conclusion



Challenges Revisited

Managing Shared Assets
Securing assets with real world value



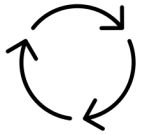
Peer-to-Peer applications
Networks of mutually distrusting parties

Real-World interactions
Supporting interactions in the real world

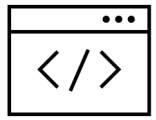
Ease-of-Use
Non-expert target audience

Smart Contracts

BFT-Consensus



CRDT’s



Web application

Future Work

Manage assets individually

Smart contracts cannot call eachother,
No support for transactions across multiple assets.

Comparison with blockchain solutions

Explore alternative Smart Contract formats

Beyond BMachines

SCEW Programming framework

Lightweight BFT-Consensus
Through state-based atomic registers

Smart Contracts
State machine representation of contract life-cycle

Client-Centric P2P Web Applications
Browser implementation

SCEW: Programmable BFT-Consensus for Client-Centric P2P Web Applications

PaPoC ’21

Martijn Sauwens, Kristof Jannes, Bert Lagaisse, Wouter Joosen
martijn.sauwens@kuleuven.be
26 April 2021

DistriNet
KU LEUVEN