HyperService: Interoperability and Programmability Across Heterogeneous Blockchains

Make Web3.0 Connected!

Zhuotao Liu, Yangxi Xiang, Jian Shi, Peng Gao, Haoyu Wang, Xusheng Xiao, Bihan Wen, Yih-Chun Hu

Blockchain Proliferation

Payment Network

Smart Contract Platform

2.3K

Total # of Projects Listed on CoinMarketCap

“Make Blockchains Great”

Sharding &
Layer-II Channels

Blockchain Y

Consensus Protocols

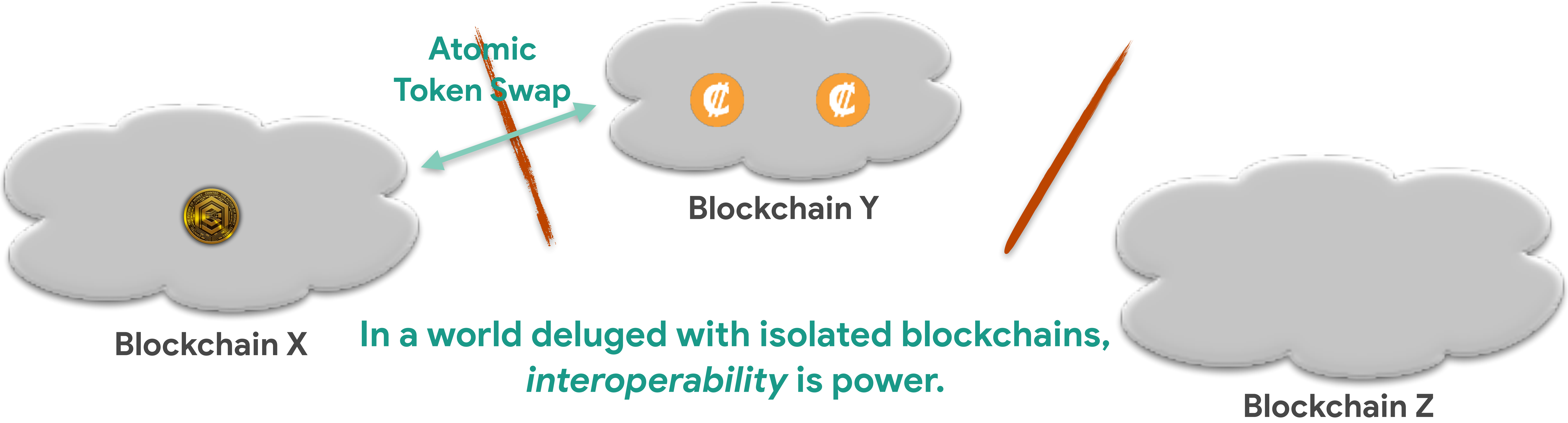
Blockchain X

Privacy &
Program Analysis

Blockchain Z

Privacy
Security
Scalability
Decentralization

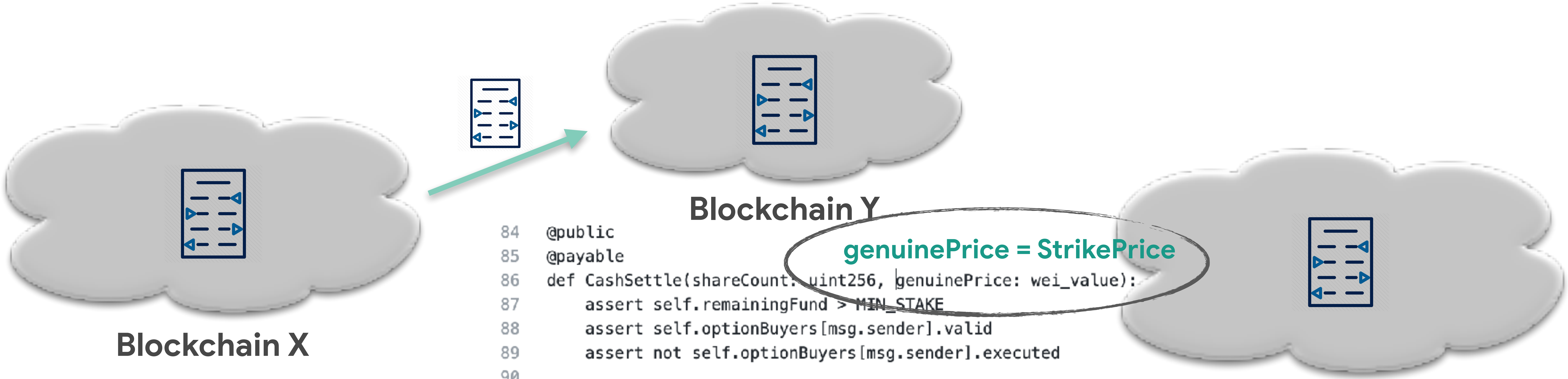
Atomic Token Swap is NOT the complete scope



In a world deluged with isolated blockchains, *interoperability* is power.

Blockchain interoperability is complete only with programmability ...

Passive Distributed Ledgers → Programmable State Machine



Blockchain X

```
pragma solidity 0.4.22;

contract Broker {
    uint constant public MAX_OWNER_COUNT = 50;
    uint constant public MAX_VALUE_PROPOSAL_COUNT = 5;

    // The authorative ouput provided by this Broker contracts.
    uint public StrikePrice;
```

StrikePrice = $10

Blockchain Y

```
@public
@payable
def CashSettle(shareCount: uint256, genuinePrice: wei_value):
    assert self.remainingFund > MIN_STAKE
    assert self.optionBuyers[msg.sender].valid
    assert not self.optionBuyers[msg.sender].executed

    if genuinePrice > self.strikePrice:
```

genuinePrice = StrikePrice

Blockchain Z

```
    function CashSettle(unit shareCount, uint genuinePrice)
        public
        optionAvaiable
    {
        require(optionBuyers[msg.sender].valid && !optionBuye
        // Only execute the option if it is profitable.
        if (genuinePrice > strikePrice) {
```

Challenge I: A virtualization layer to abstract away heterogeneity

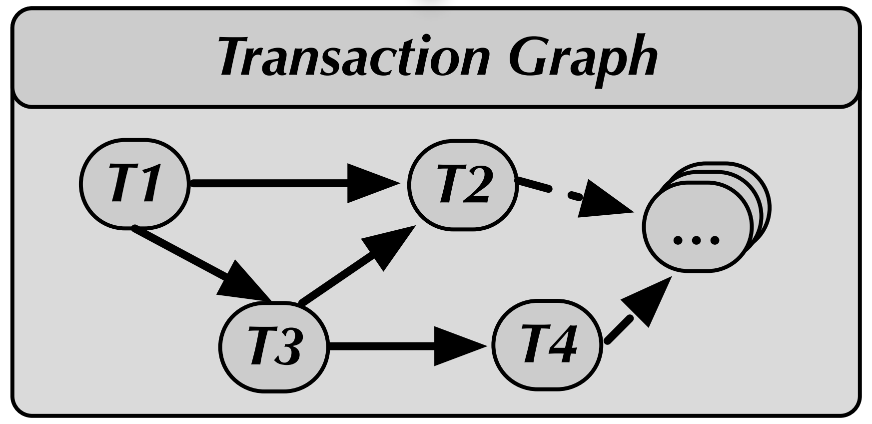
Cross-chain dApps: how to *uniformly define* operations among heterogeneous contracts and accounts ...

Challenge II: Cryptography protocols to realize cross-chain dApps

Cross-chain dApps in the era of Web3.0

Contain more complex operations than just token transfers

dApp Executables



- *Transactions on different Blockchains;*
- *Transactions in specific order;*
- *Downstream transactions depend on state resulted from upstream transactions;*

How to realize transactions via *decentralized protocols?*

Our Proposal — HyperService

- **A developer-facing programming framework**
  - **Universal State Model:** a blockchain-neutral model to describe dApps
  - **HyperService Language:** a high-level language to program dApps

- **A blockchain-facing cryptography protocol** to realize dApps on-chain
  - **Network Status Blockchain:** a decentralized trust anchor
  - **Insurance Smart Contract:** a trust-free code arbitrator

A universal platform for developing and executing dApps across heterogenous Blockchains

Programming Framework — Universal State Model

$$\mathscr{M} = \{\mathscr{E}, \mathscr{P}, \mathscr{C}\} = \{\text{Entities, Operations, Constraints}\}$$

Entities: objects extracted from underlying blockchains

Entities	Attributes
account	address, balance, unit
contract	***state variables[]***, ***interfaces[]***, source

Blockchain X

```
pragma solidity 0.4.22;

contract Broker {
    uint constant public MAX_OWNER_COUNT = 50;
    uint constant public MAX_VALUE_PROPOSAL_COUNT = 5;

    // The authorative ouput provided by this Broker contracts.
    uint public StrikePrice;
```

X::Broker.StrikePrice

Y::Option.CashSettle(uint256, wei_value)

```
@public
@payable
def CashSettle(shareCount: uint256, genuinePrice: wei_value):
    assert self.remainingFund > MIN_STAKE
    assert self.optionBuyers[msg.sender].valid
    assert not self.optionBuyers[msg.sender].executed

    if genuinePrice > self.strikePrice:
```

Blockchain Y

Programming Framework — Universal State Model

$$\mathscr{M} = \{\mathscr{E}, \mathscr{P}, \mathscr{C}\} = \{\text{Entities, Operations, Constraints}\}$$

Operations: computation performed over several entities

Operations	Attributes
payment	from, to, value, exchange rate
invocation	interface, ***parameters[]***, invoker

An example invocation operation:

Y::Option.CashSettle(10, X::Broker.StrikePrice)

Blockchain X

```
pragma solidity 0.4.22;

contract Broker {
    uint constant public MAX_OWNER_COUNT = 50;
    uint constant public MAX_VALUE_PROPOSAL_COUNT = 5;

    // The authorative ouput provided by this Broker contracts.
    uint public StrikePrice;
```

X::Broker.StrikePrice

Y::Option.CashSettle(uint256, wei_value)

```
@public
@payable
def CashSettle(shareCount: uint256, genuinePrice: wei_value):
    assert self.remainingFund > MIN_STAKE
    assert self.optionBuyers[msg.sender].valid
    assert not self.optionBuyers[msg.sender].executed

    if genuinePrice > self.strikePrice:
```

Blockchain Y

Programming Framework — Universal State Model

$$\mathscr{M} = \{\mathscr{E}, \mathscr{P}, \mathscr{C}\} = \{\text{Entities, Operations, Constraints}\}$$

Entities: objects extracted from underlying blockchains

Operations: computation performed over several entities

Constraints: dependencies among operations

Entities	Attributes	Operations	Attributes	Dependency
account	address, balance, unit	***payment***	from, to, value, exchange rate	***precondition***
contract	***state variables[]***, ***interfaces[]***, source	***invocation***	interface, ***parameters[]***, invoker	***deadline***

HyperService Language (HSL): A high-level programming language

import: include the source code of all contracts defined in the HSL program

account & contract: defining entities extracted from underlying blockchains

payment & invocation: defining operations among entities

before, after & deadline: defining dependencies among operations

```
# Import the source code of contracts written in different languages.
import ("broker.sol", "option.vy", "option.go")
# Entity definition.
# Attributes of a contract entity are implicit from its source code.
account a1 = ChainX::Account(0x7019..., 100, xcoin)
account a2 = ChainY::Account(0x47a1..., 0, ycoin)
account a3 = ChainZ::Account(0x61a2..., 50, zcoin)
contract c1 = ChainX::Broker(0xbba7...)
contract c2 = ChainY::Option(0x917f...)
contract c3 = ChainZ::Option(0xefed...)
# Operation definition.
op op1 invocation c1.GetStrikePrice() using a1
op op2 payment 50 xcoin from a1 to a2 with 1 xcoin as 0.5 ycoin
op op3 invocation c2.CashSettle(10, c1.StrikePrice) using a2
op op4 invocation c3.CashSettle(5, c1.StrikePrice) using a3
# Dependency definition.
op1 before op2, op4; op3 after op2
op1 deadline 10 blocks; op2, op3 deadline default; op4 deadline 20 mins
```

Figure 2: A cross-chain Option dApp written in HSL.

Unified Type	*Solidity*	*Vyper*	*Go*
Boolean	bool	bool	bool
Numeric	int, unit	int128, decimal, ...	int, float, ...
Array	array, bytes	array, bytes	array, slice

Programming Framework Core -- HSL Program Compilation

Extract state variables and interfaces from imported contracts

Unify different contract languages

- Compatibility: type check
- Verifiability: state variables
- Feasibility: no dep-loop

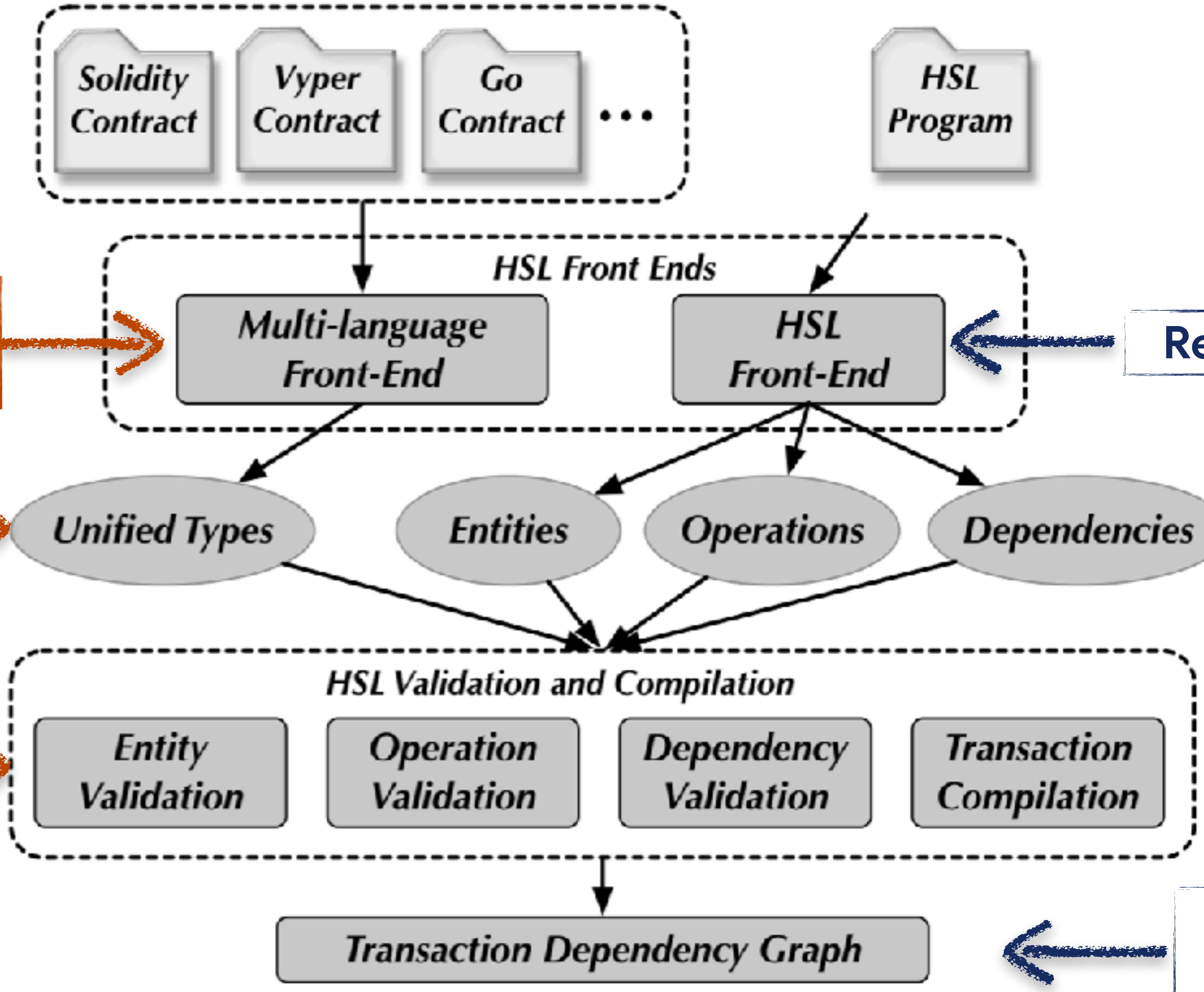


Figure 3: Workflow of HSL Compilation.

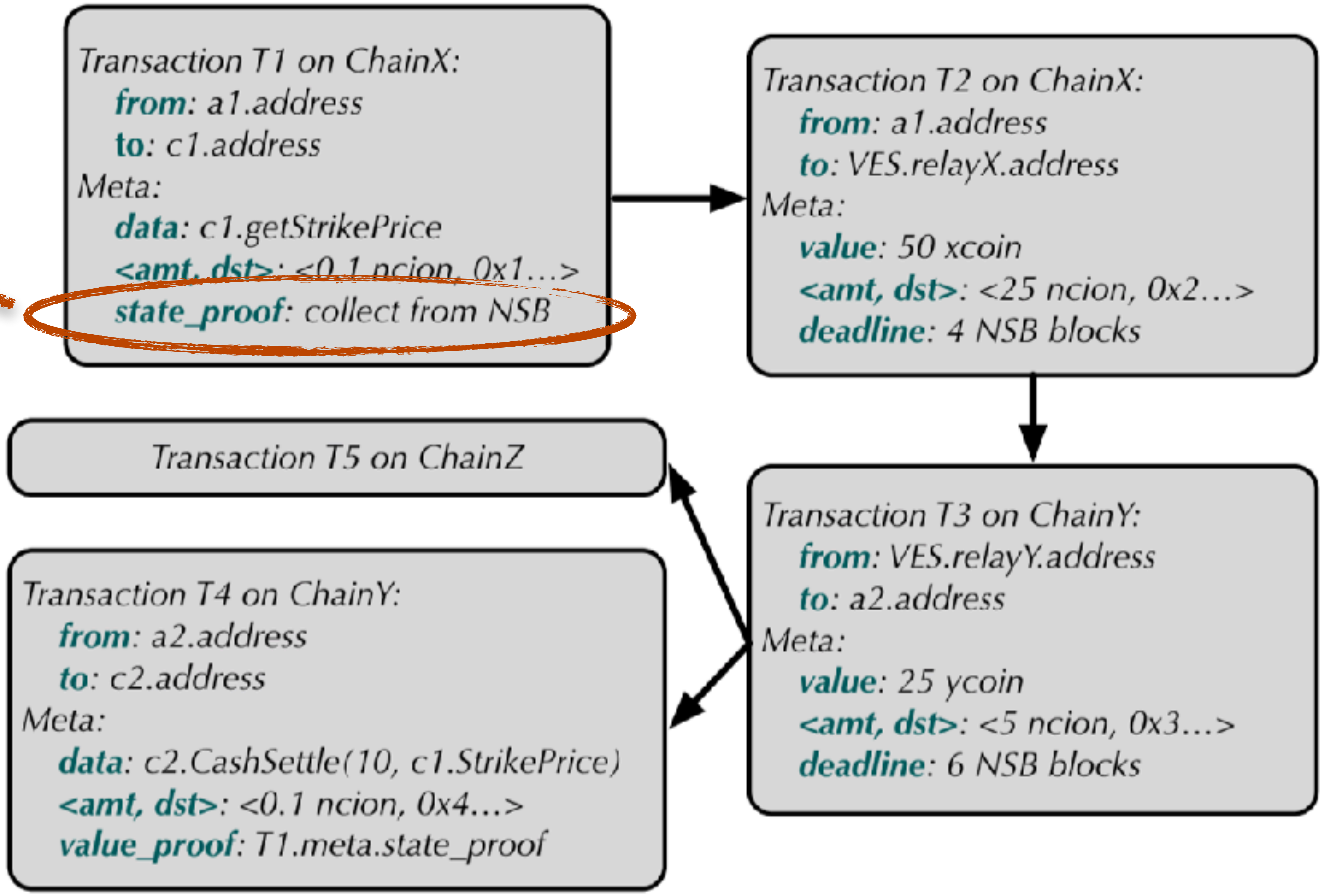
Read and parse HSL programs

Extract {E, O, D} from HSL program

HSL program executables

Transaction Dependency Graph (TDG) — HSL Program Executables

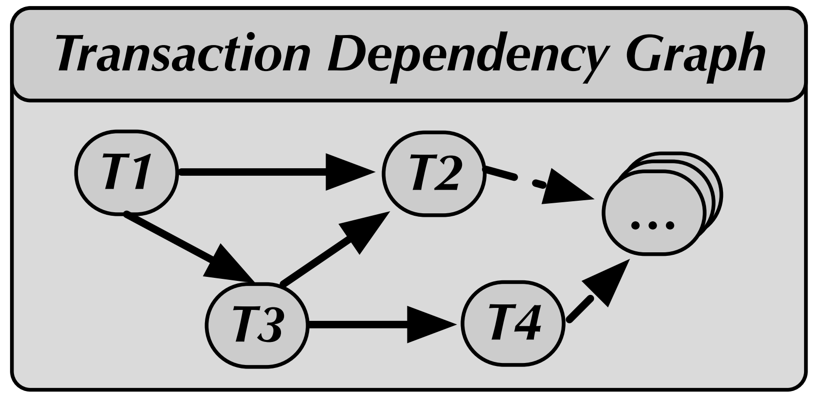
- Resulting state of T1 is used subsequently
- A state proof needs to be collected after T1 is finalized.



- **Each vertex defines:**
  - Full information for computing a blockchain-executable transaction
  - Metadata to ensure correct execution
- **Edges define the transaction order**

HyperService Architecture

Developer-facing Programming Framework



Universal Inter-Blockchain Protocol

Universal Inter-Blockchain Protocol (UIP) Overview

- A protocol spoken by all parties to co-execute cross-chain dApps
- Fully decentralized: no authorities and no mutual trust among parties

- Provable security properties
  - Correctness assurance, financial atomicity, and accountability

- Network Status Blockchain: a decentralized trust anchor
- Insurance Smart Contract: a trust-free code arbitrator

UIP Security Properties

TDG is realized as desired	• dApp execution either finishes correctly or being financially reverted	Financial Atomicity
Accountability	• Regardless of at which stage the execution fails, the misbehaved parties are held accountable for the failure	
Correctness Guarantee	• If blockchains are modeled with bounded transaction finality latency, dApps are guaranteed to finish correctly if all parties are honest	

Security properties of dApps executed by UIP
(Proved in UC-Framework)

NSB Design

- Consolidate transactions and state from underlying blockchains
- Provide unified representations for transaction status and state in form of verifiable Merkle proofs

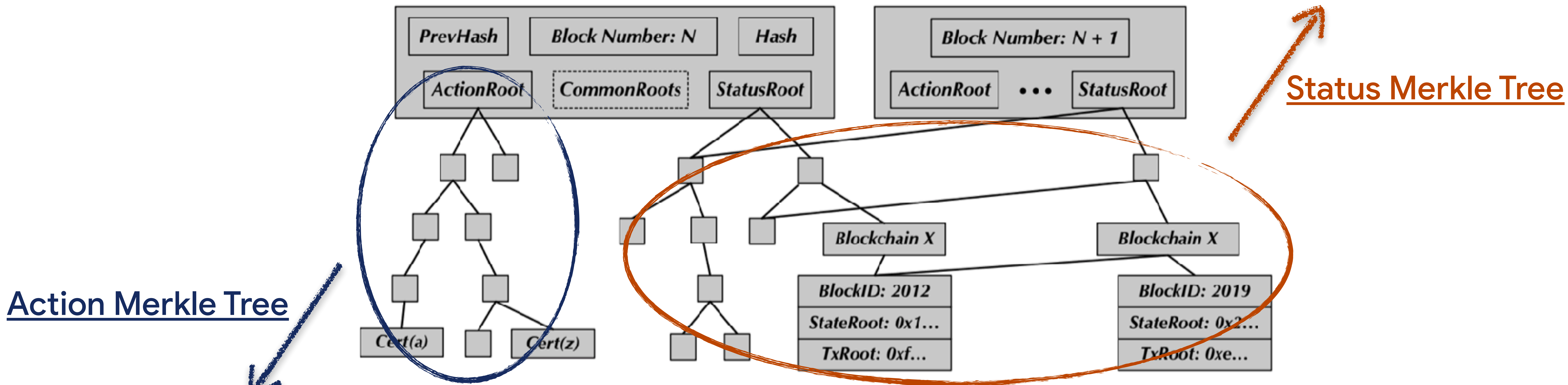


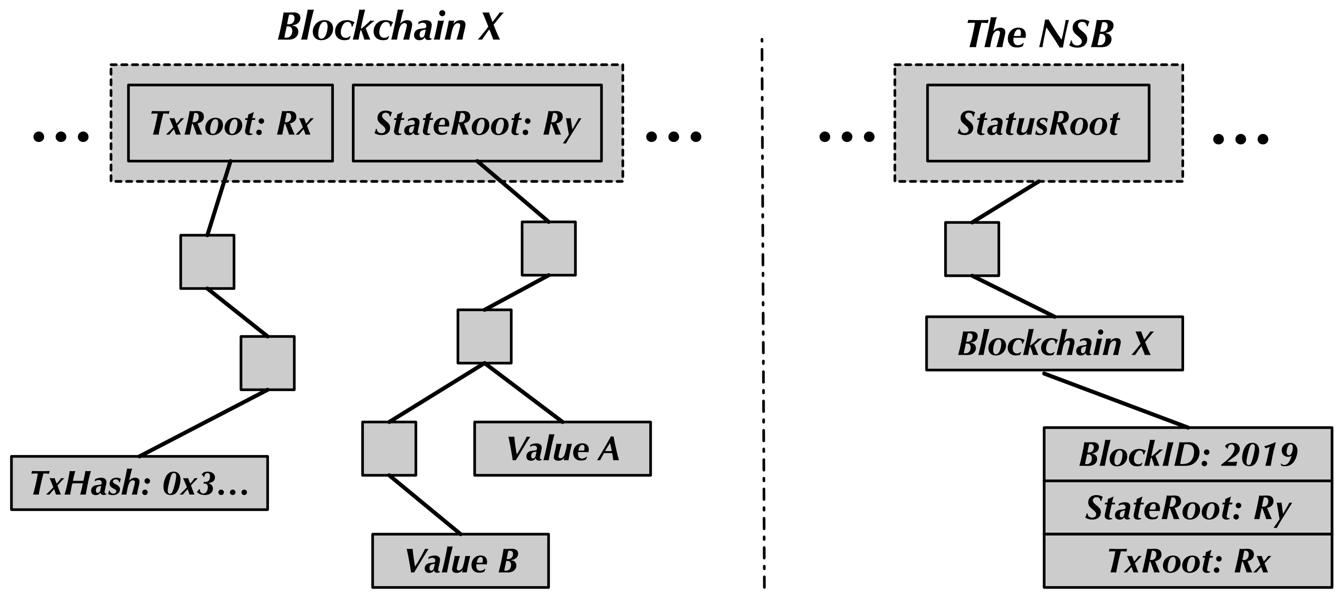
Figure 5: The architecture of NSB blocks.

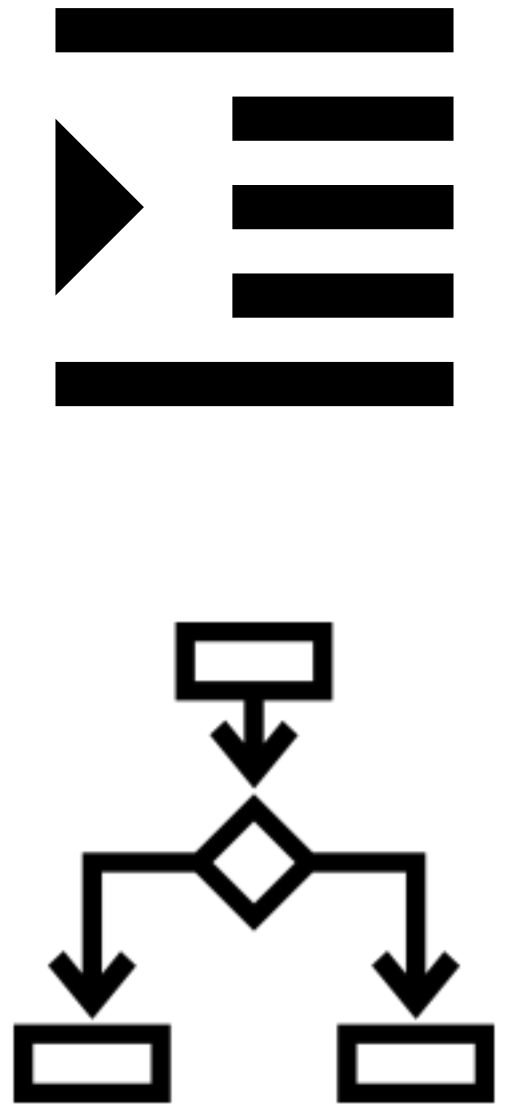
- Proof of Actions (PoAs): allow parties to construct proofs to certify their actions taken during executions

NSB: Provide unified and objective views on the status of dApp executions

Insurance Smart Contract (ISC)

Merkle Proofs





Decision Logic

if CorrectExecution:
Pay service fee
else:
Revert effective fund
Enforce accountability

Implementation and Source Code Release (as of March 2020)

60K

Lines of Code

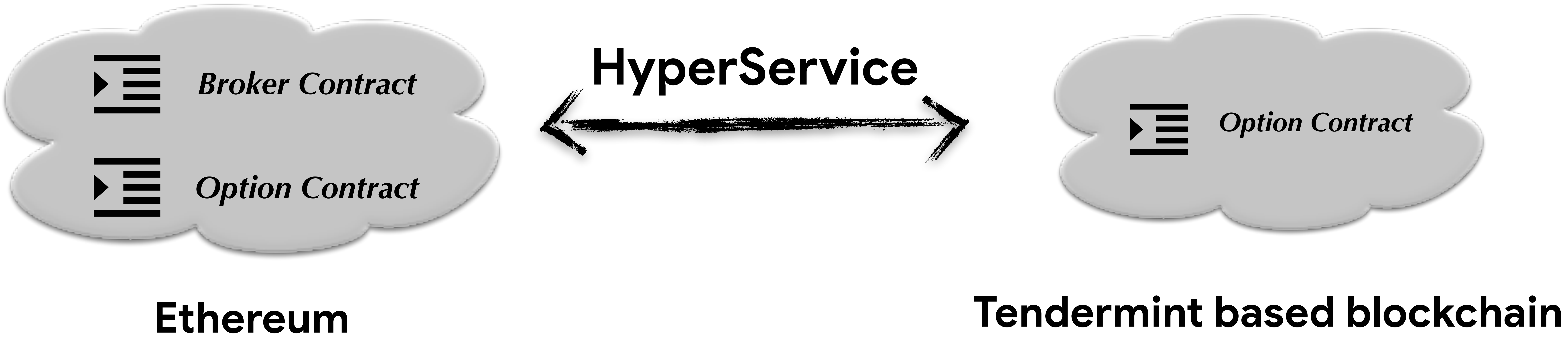
- Incorporate Ethereum and a permissioned blockchain built on Tendermint
  - Different consensus efficiency and transaction finality definition
  - Different contract languages: Solidity VS. Go

- Three categories of cross-chain dApps
  - Financial derivative, asset movement and federated computing

- Released source code: https://github.com/HyperService-Consortium

Demo: End-to-end executions on HyperService

1. *Invoke E::Broker.ComputeStrikePrice()*
2. *Invoke T::Option.cash_settle(E::Broker.StrikePrice)*
3. *Invoke E::Option.CashSettle(E::Broker.StrikePrice)*



HyperService: A *universal platform* for *developing* and *executing* dApps across *heterogenous* Blockchains

Q & A

Thank You

hyperservice.team@gmail.com