CHAINSPACE

A Scalable Smart Contracts Platform

Shehar Bano
Postdoc, InfoSec group
University College London

s.bano@ucl.ac.uk
@thatBano

https://github.com/chainspace

NDSS '18 SYMPOSIUM

The Team

Mustafa Al-Bassam
(UCL)

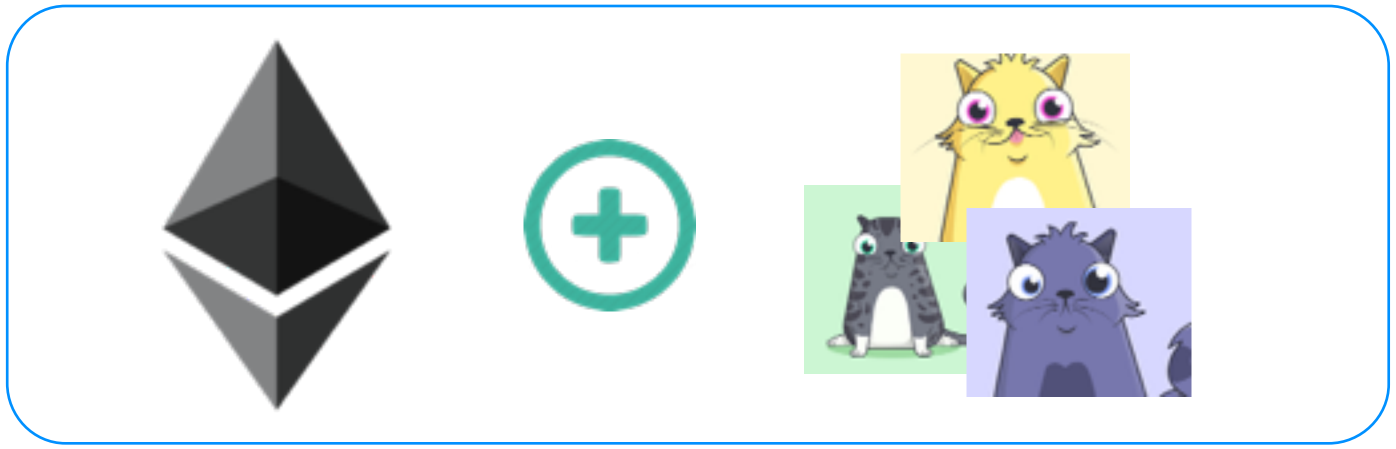
Alberto Sonnino
(UCL)

Bano
(UCL)

Dave Hrycyszyn
(constructiveproof)

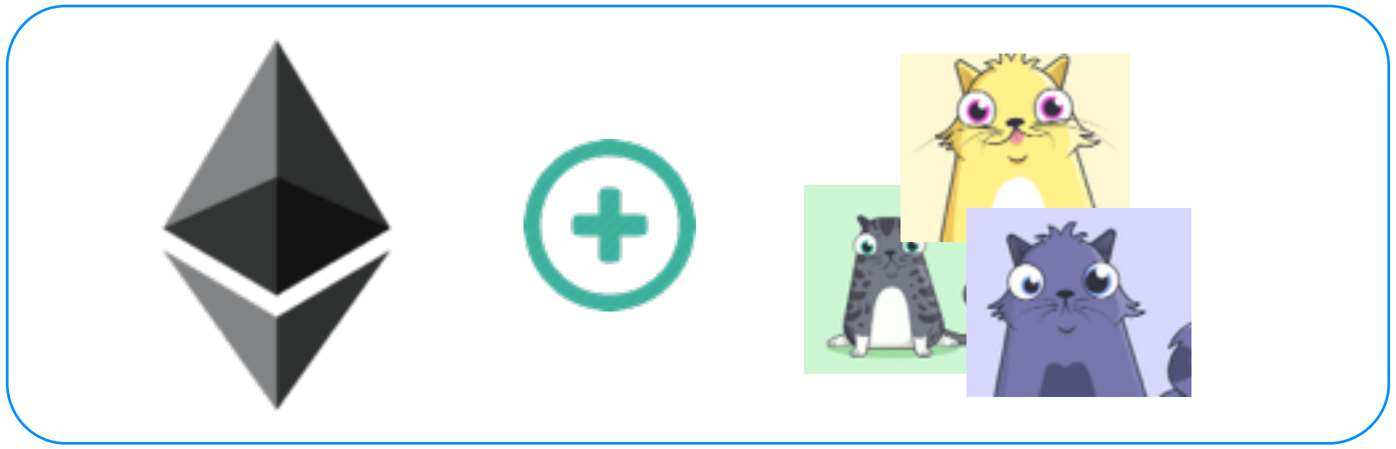
George Danezis
(UCL)

Why Chainspace?



- Blockchains are cool — but scale badly

Why Chainspace?



- Blockchains are cool — but scale badly

Transactions are recorded on chain

Inputs are therefore public

- Hard to operate on secret inputs

Why Chainspace?

	Smart Contract	Scalable	Privacy
Ethereum	✓	✗	✗
Hawk	✓	✗	✓
ZCash	✗	✗	✓
Omniledger	✗	✓	✗
RSCoin	✗	✓	✗

Why Chainspace?

contribution I

Scalable smart contract platform

Why Chainspace?

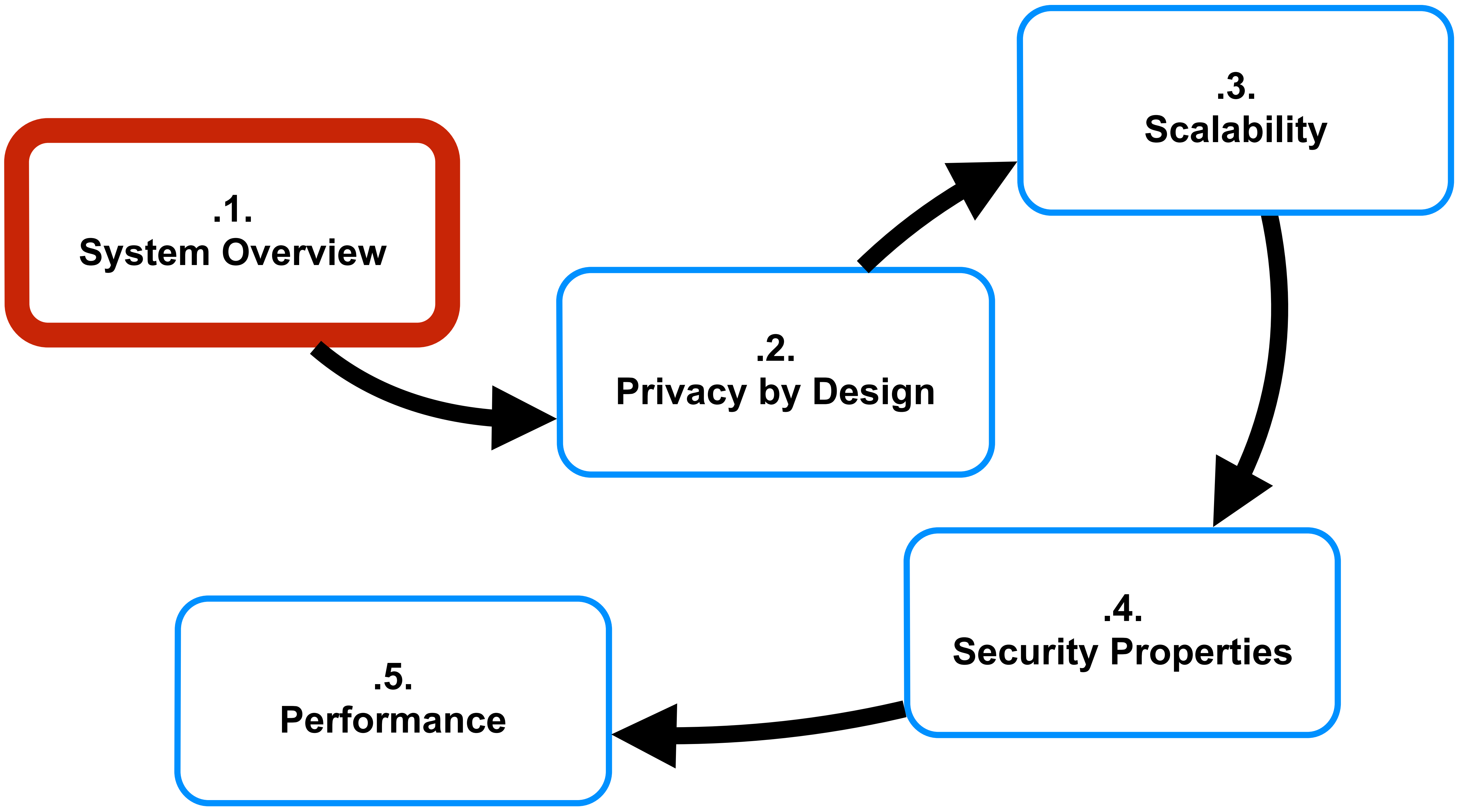
contribution I

Scalable smart contract platform

contribution II

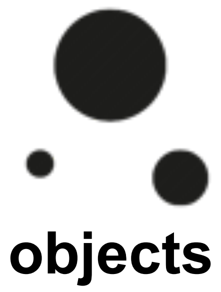
Supporting privacy

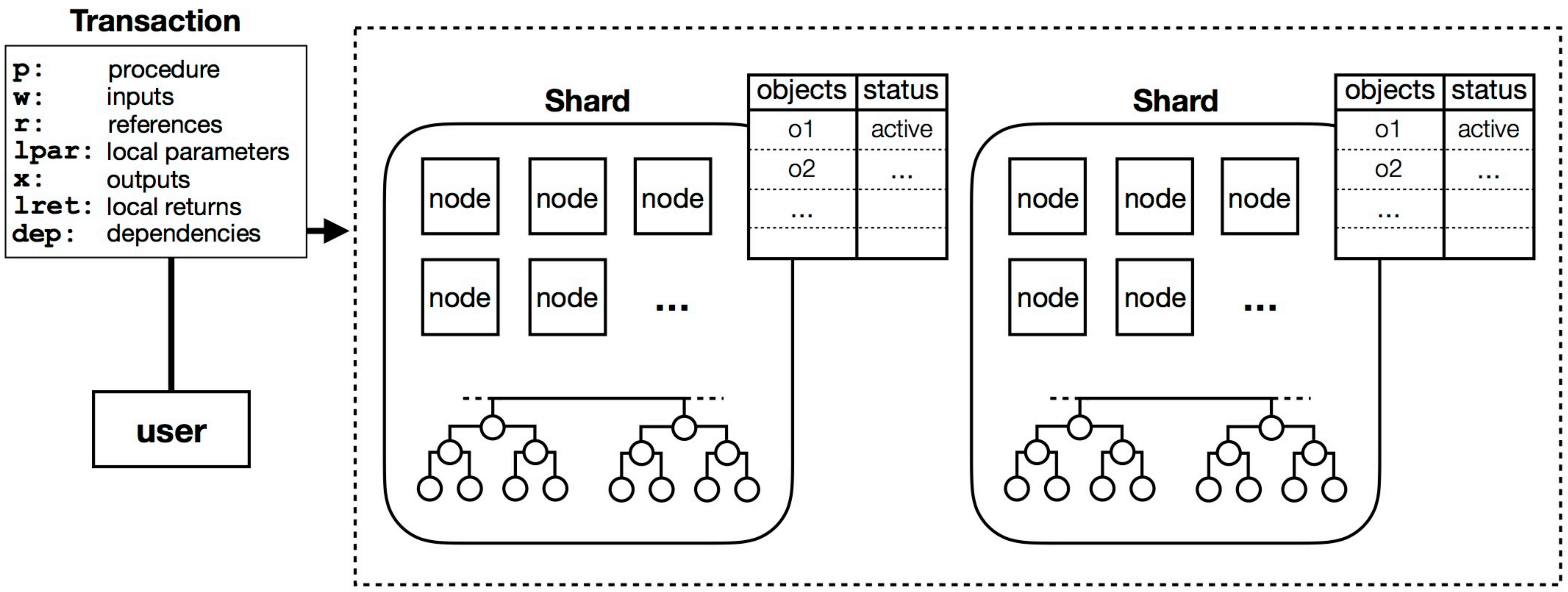
Contents


.1.
System Overview
.2.
Privacy by Design
.3.
Scalability
.4.
Security Properties
.5.
Performance

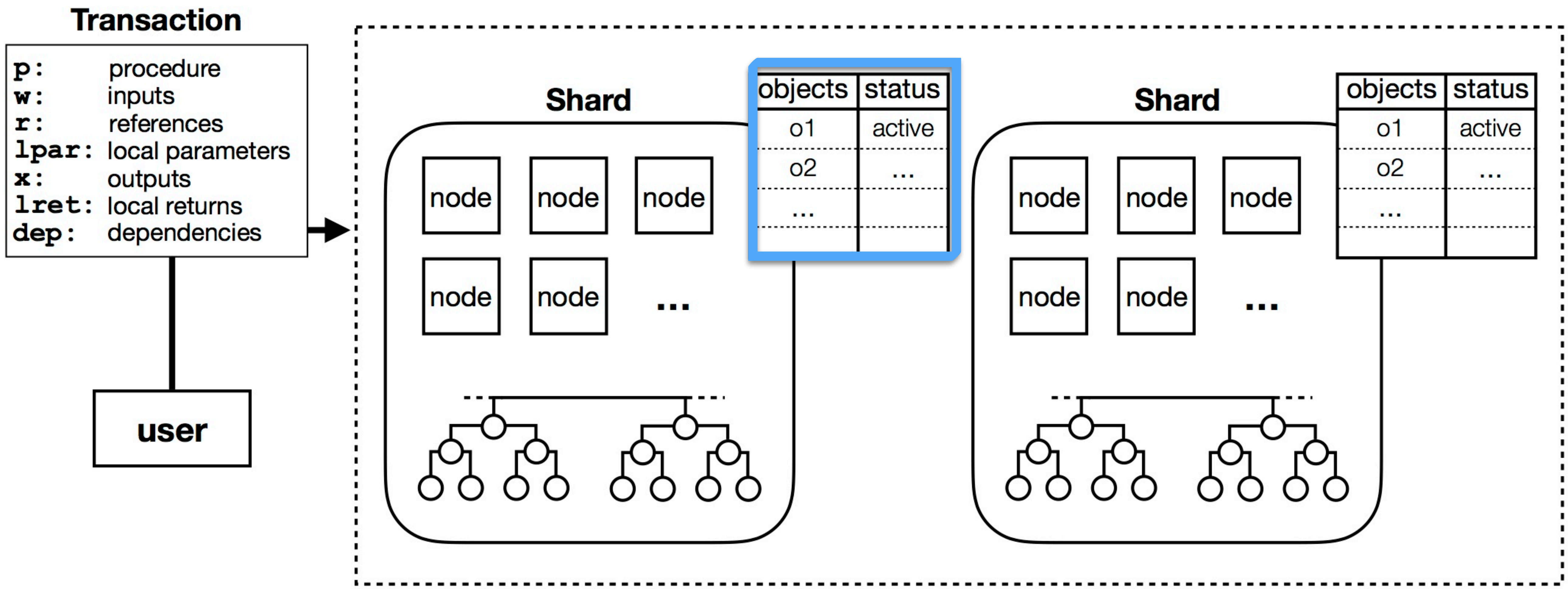
System Overview

- How Chainspace works?
  - Nodes are organised into **shards**
  - Shards manage **objects**





Objects


Transaction
p: procedure
w: inputs
r: references
lpar: local parameters
x: outputs
lret: local returns
dep: dependencies
user
Shard
node
...
objects
status
o1
active
o2
...
Shard

Objects

- Hold state in Chainspace (e.g. Bank Account, Train Seat, Hotel Room).
- Object state is immutable.
- Objects may be in two meta-states, either active or inactive.
  - **Active objects** are available to be operated on through smart contract procedures.
  - **Inactive ones** are retained for the purposes of audit only.

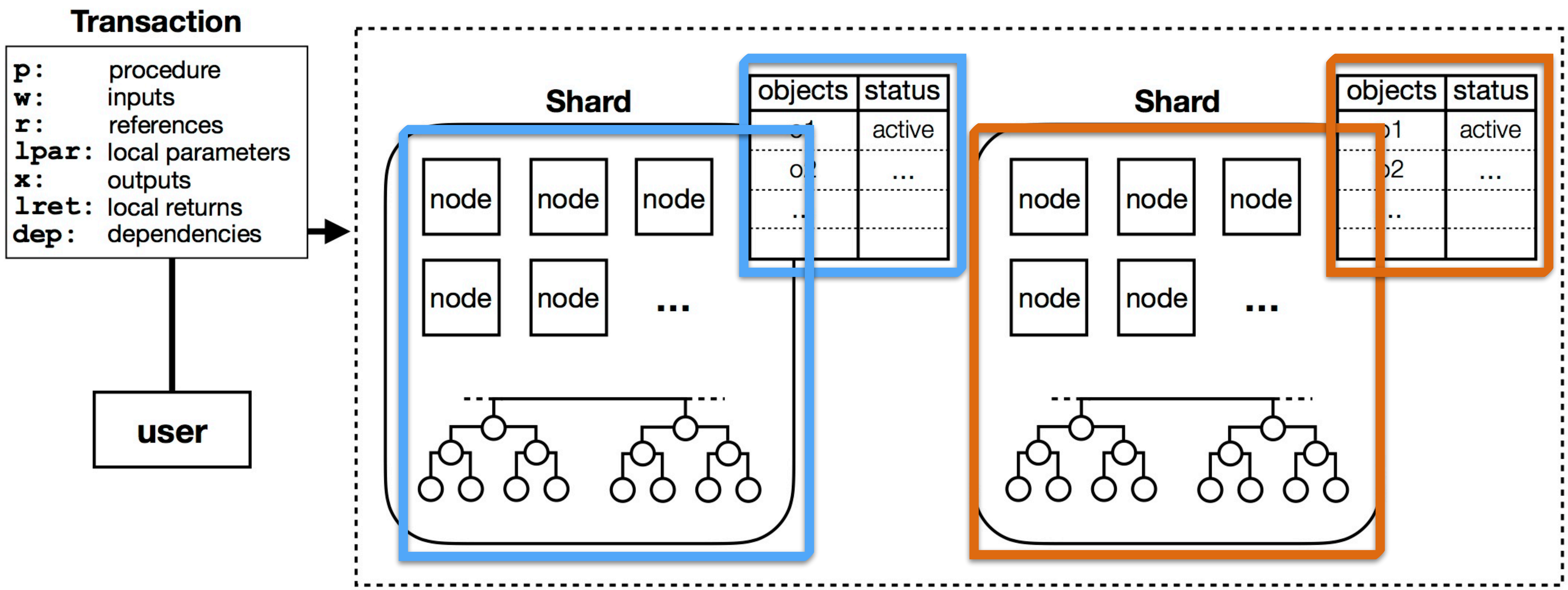
Smart Contracts

- Contracts are special types of objects
- Contain executable information on how other objects be manipulated
  - Contracts contain **procedures** that define the logic by which a number of objects are processed
  - Procedures do not have to be pure functions, and may be randomized, keep state or have side effects

Composition of Smart Contracts

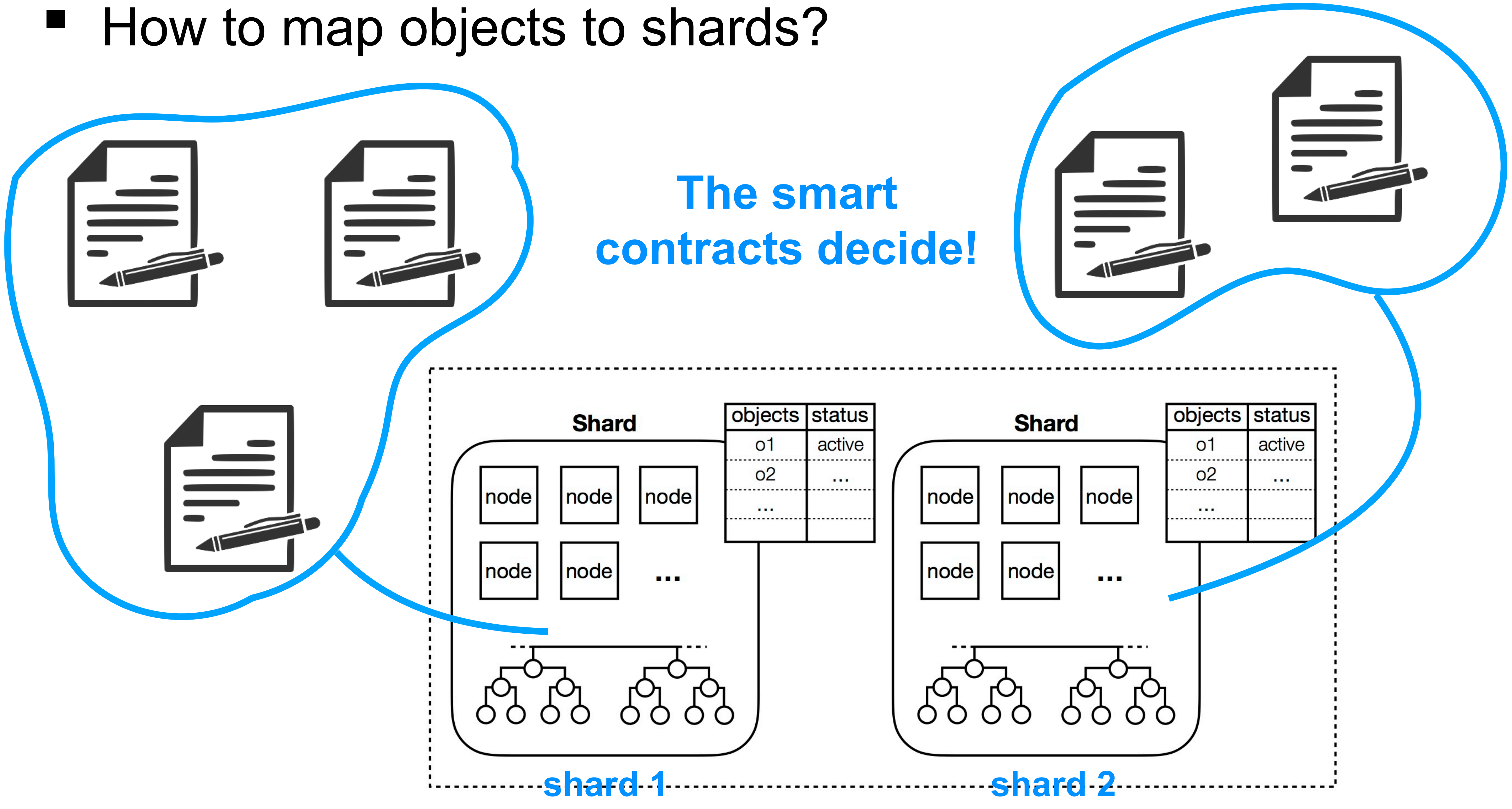
- A contract procedure may call a procedure of another smart contract
- Allows the creation of a library of smart contracts from different authors that act as utilities for other higher-level contracts

Object-to-Shard Mapping

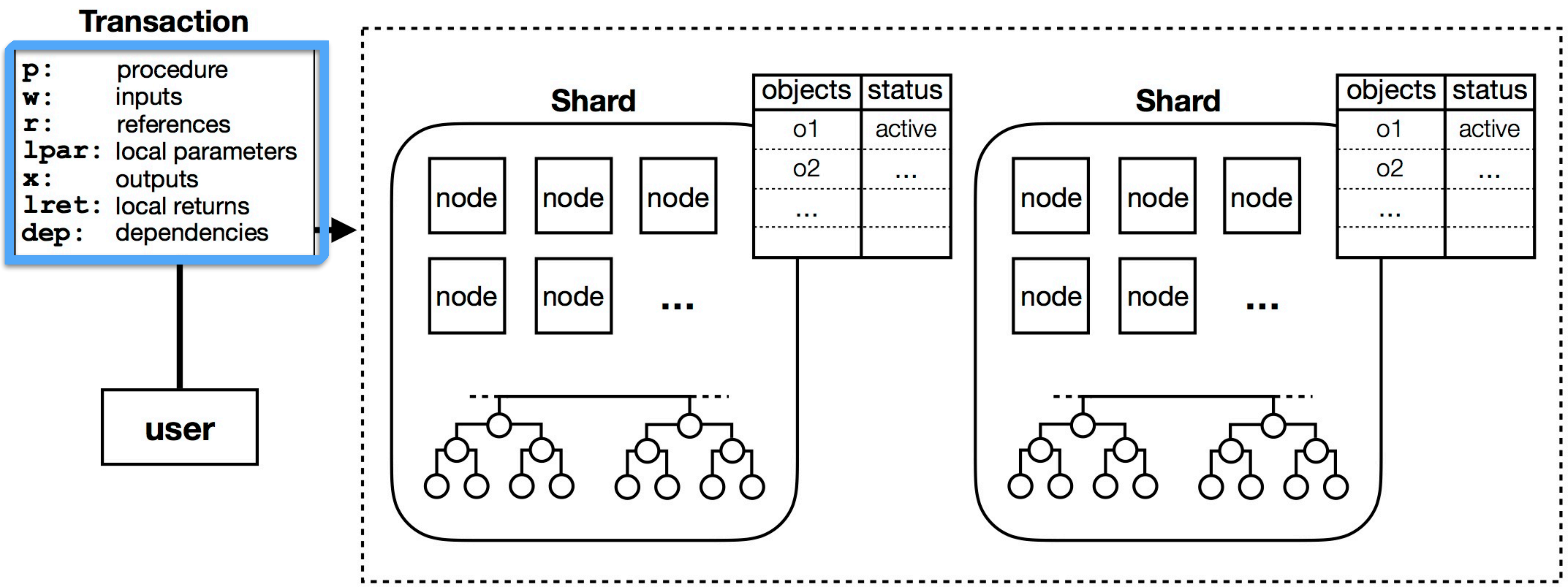


Smart Contracts map Objects to Shards

- How to map objects to shards?

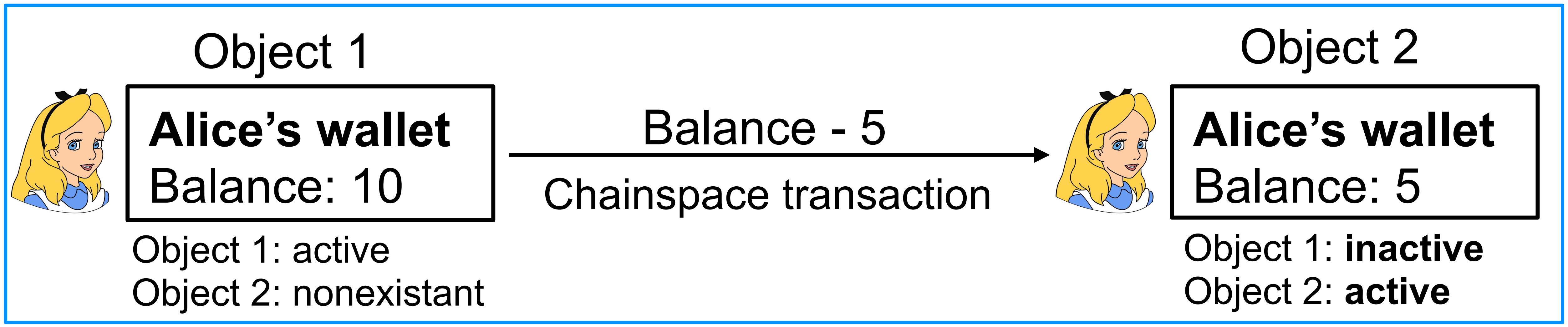


Transactions



Transactions

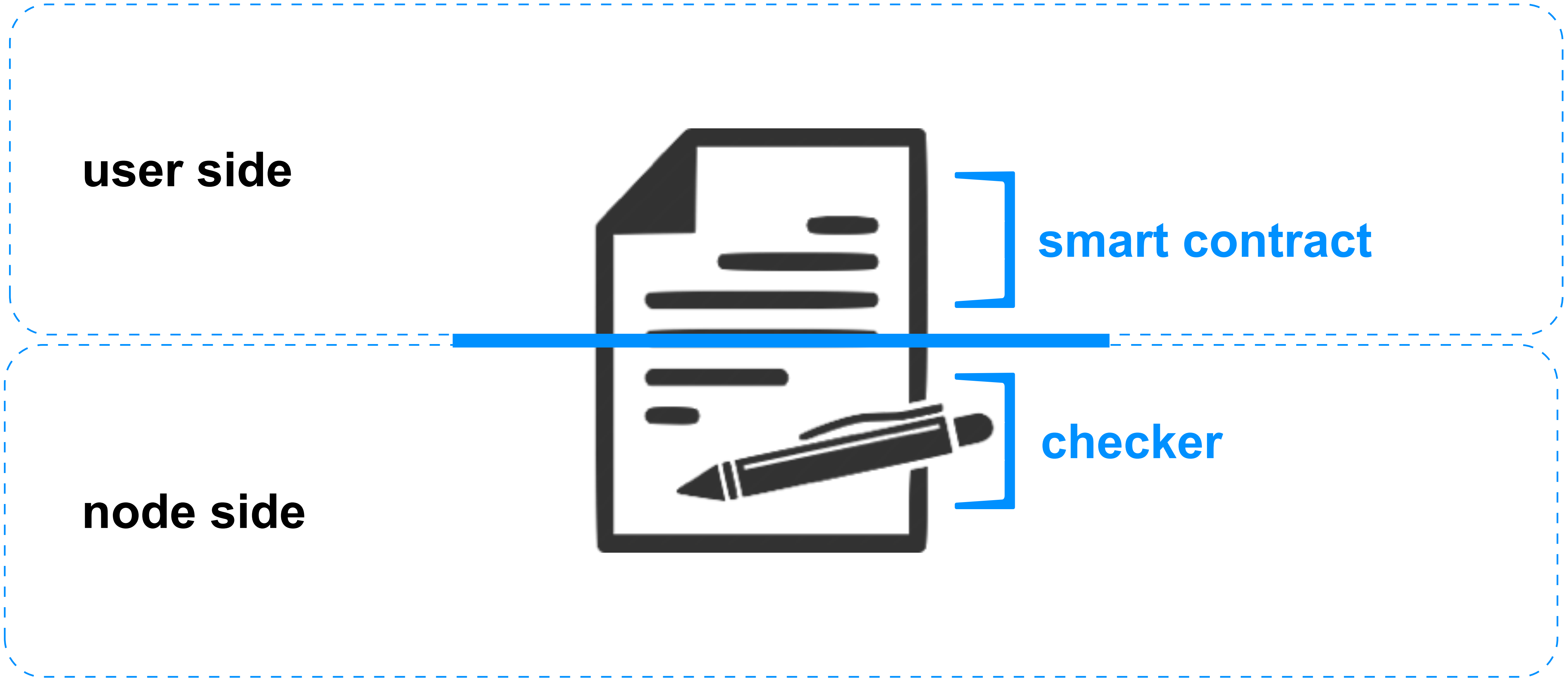
- Instantiation of smart contracts
- Once a transaction is accepted in Chainspace
  - all input objects 'die' (become inactive)
  - all output objects are 'born' (become active)



Contents

Checkers

- Every smart contract has a checker



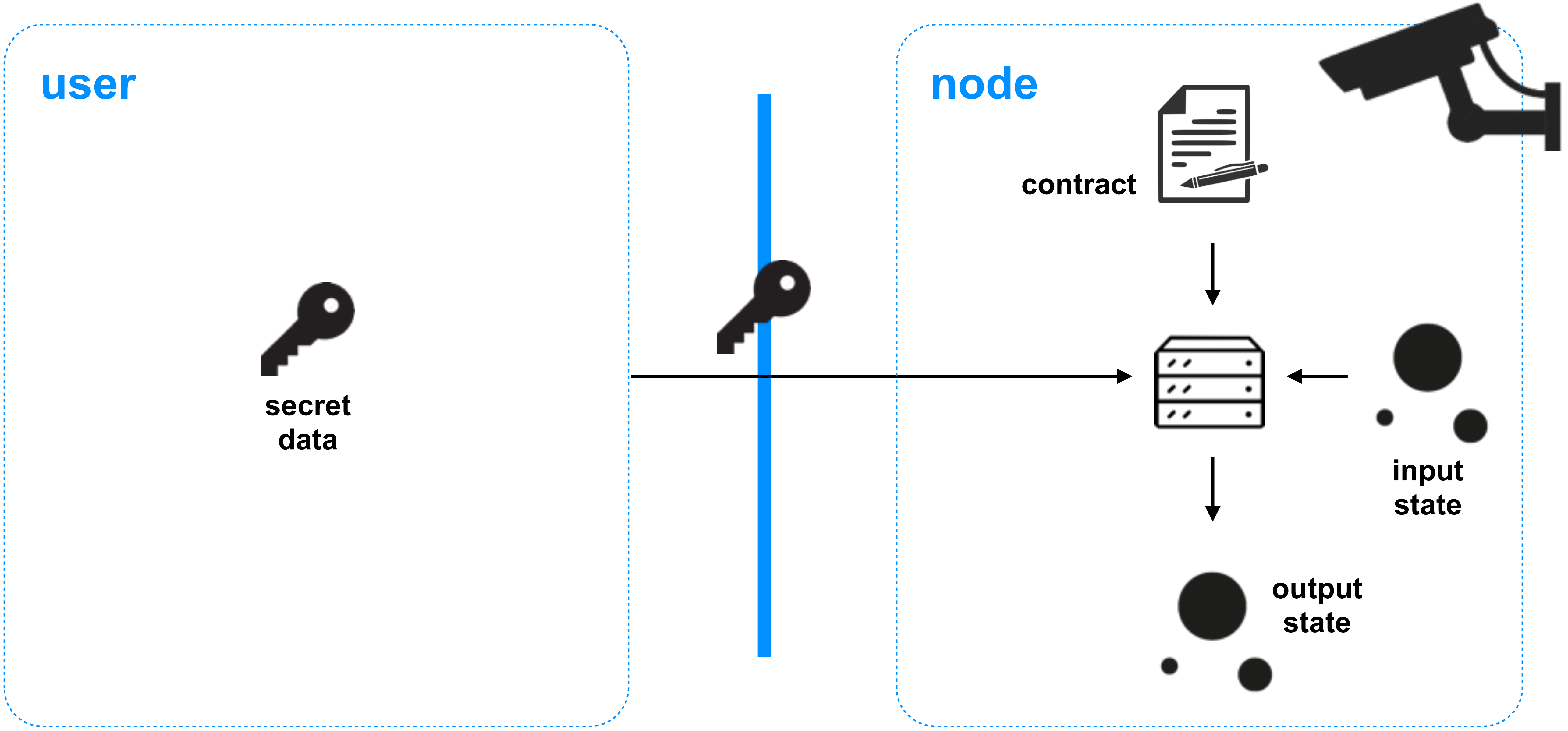
Checkers

- Checkers are pure functions (i.e., deterministic, and have no side-effects), and return a Boolean value.

- A checker requires **no secret inputs**
  - only has sufficient information to check transaction validity (e.g., a zero-knowledge proof)

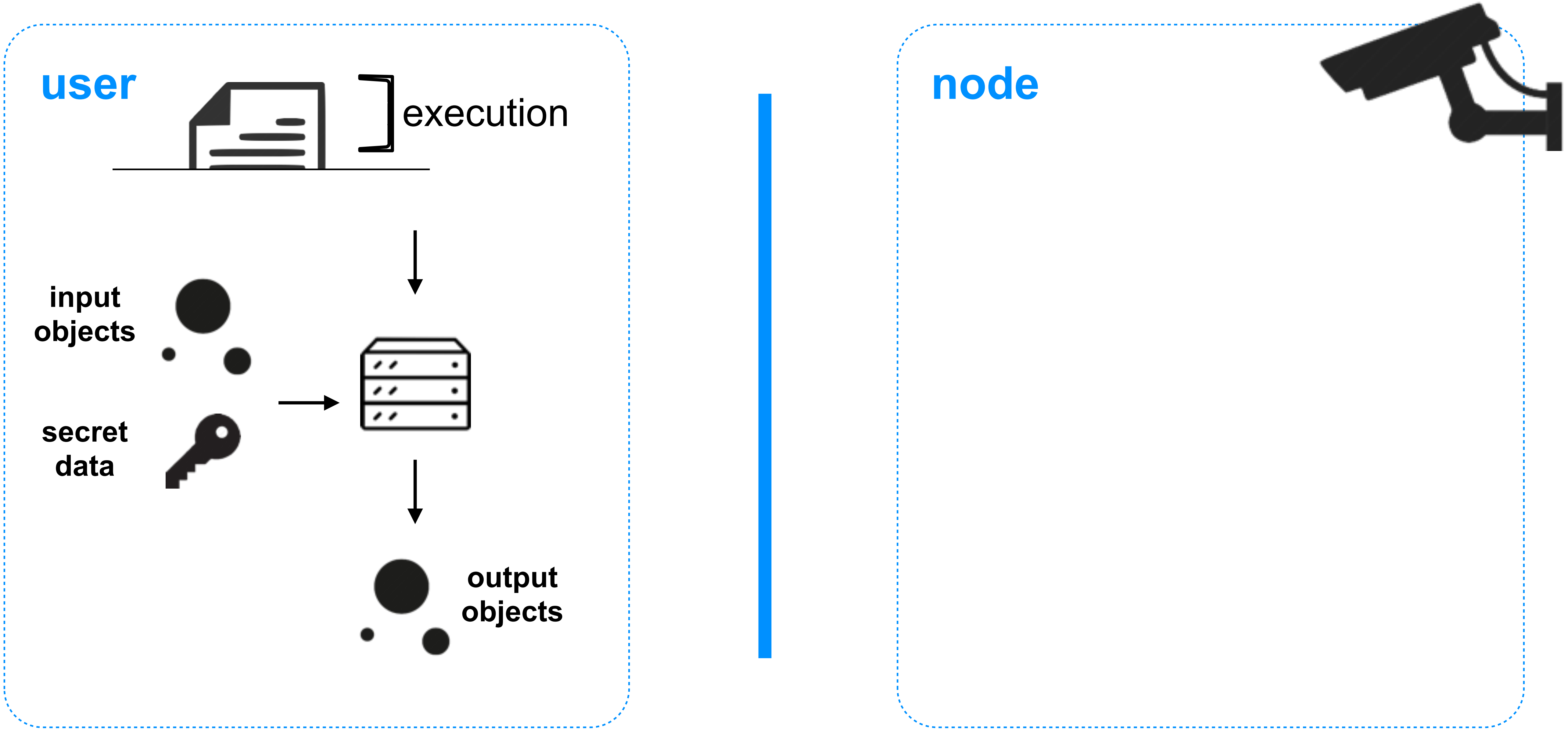
Privacy by Design

- Transaction in classic blockchains



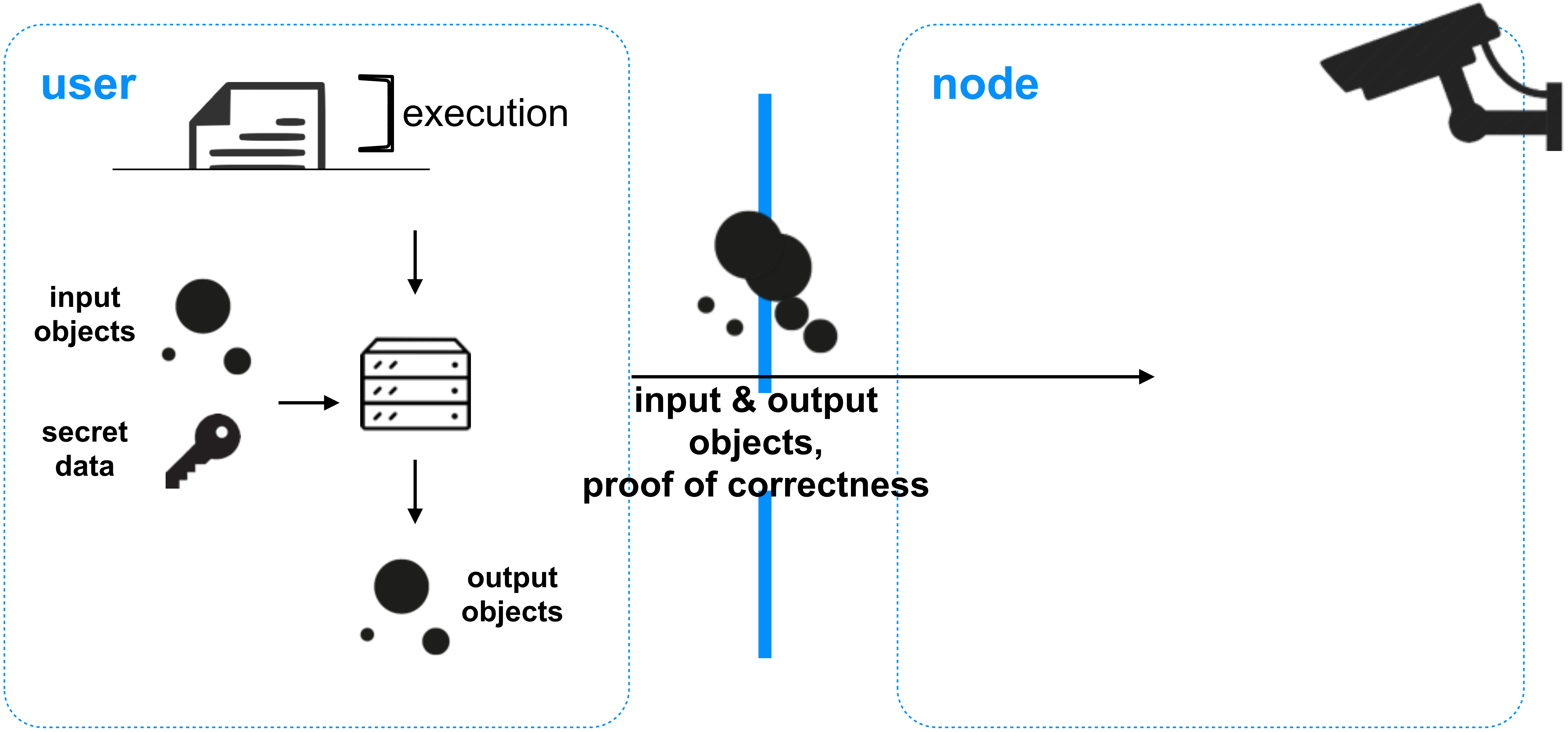
Privacy by Design

- Chainspace transaction



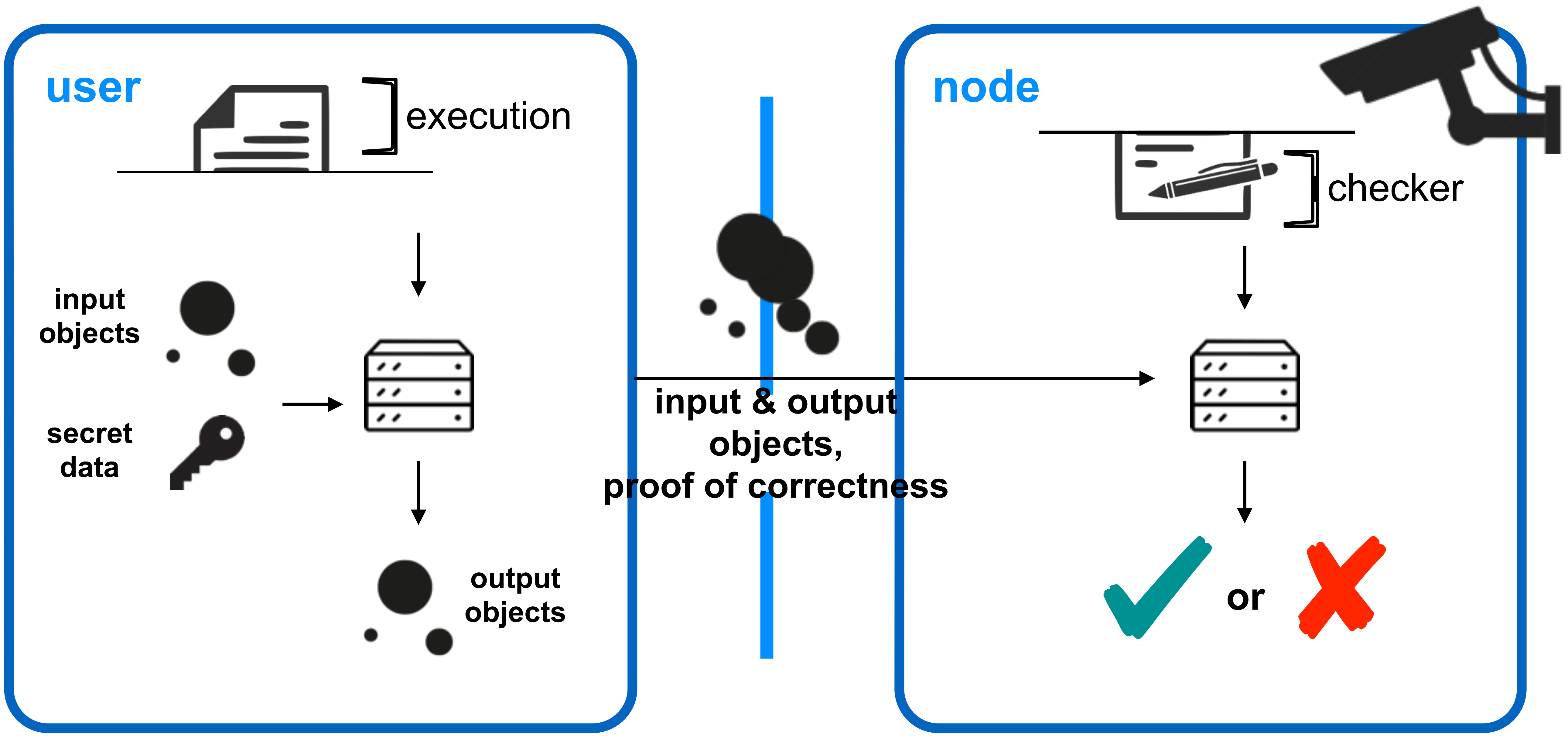
Privacy by Design

- Chainspace transaction



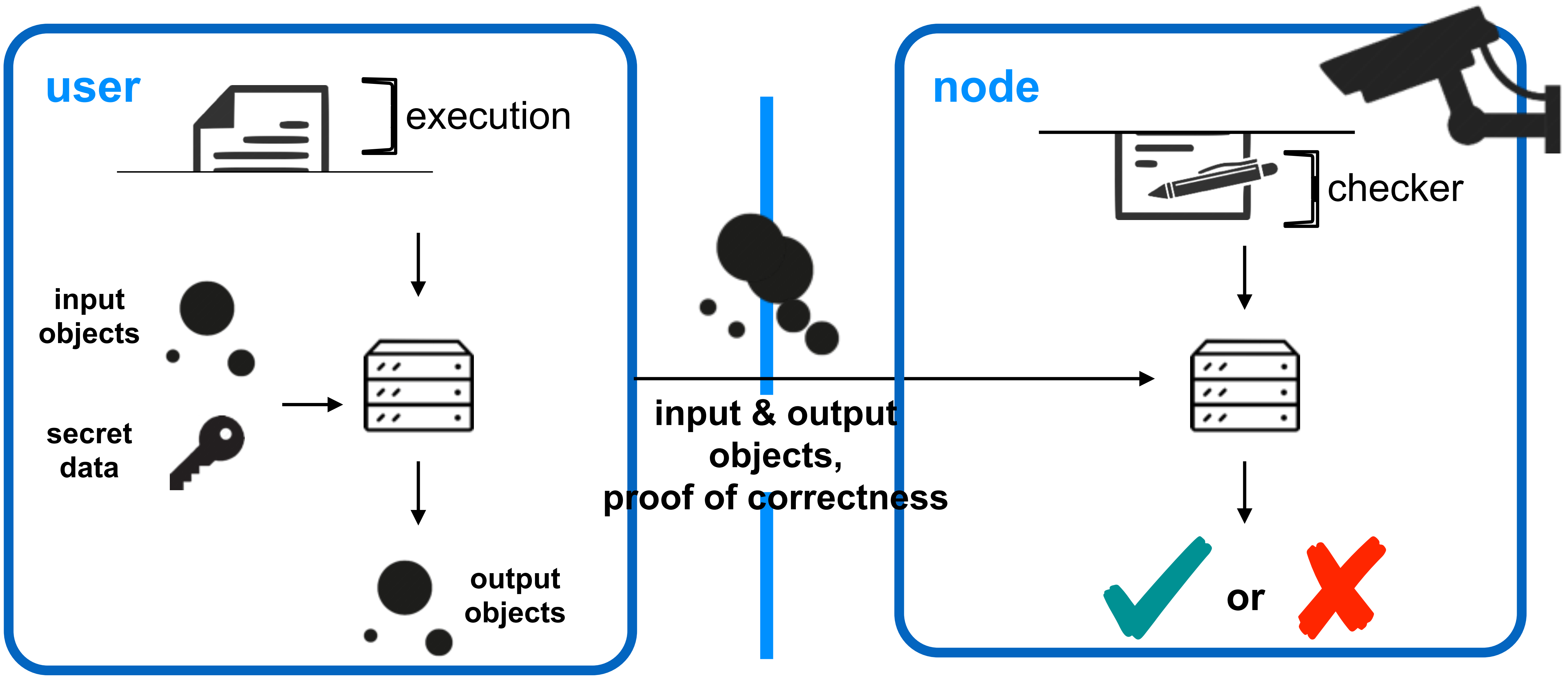
Privacy by Design

- Chainspace transaction



Example: Private E-petitions

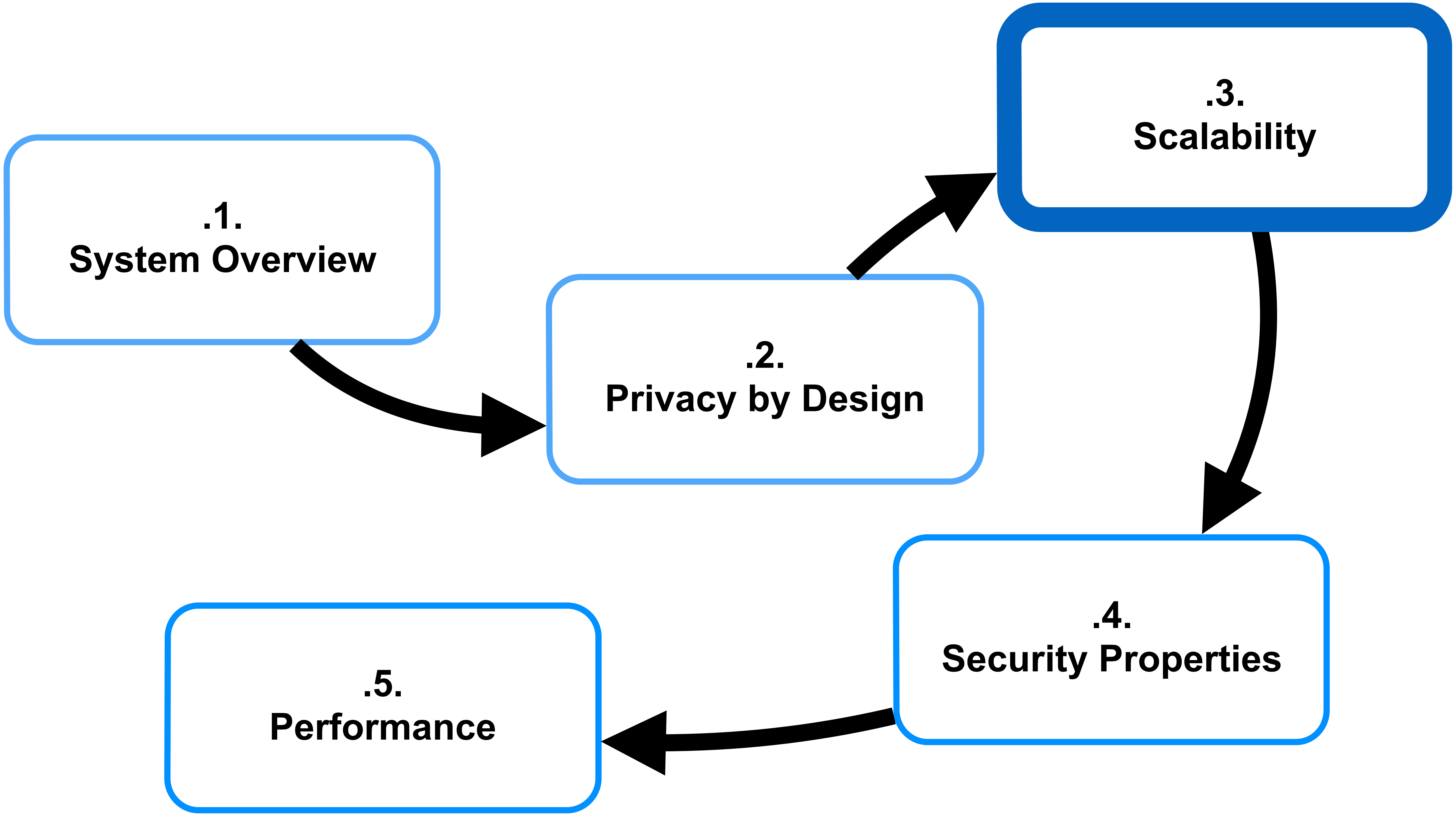
- Legitimate user casts a vote without identifying herself



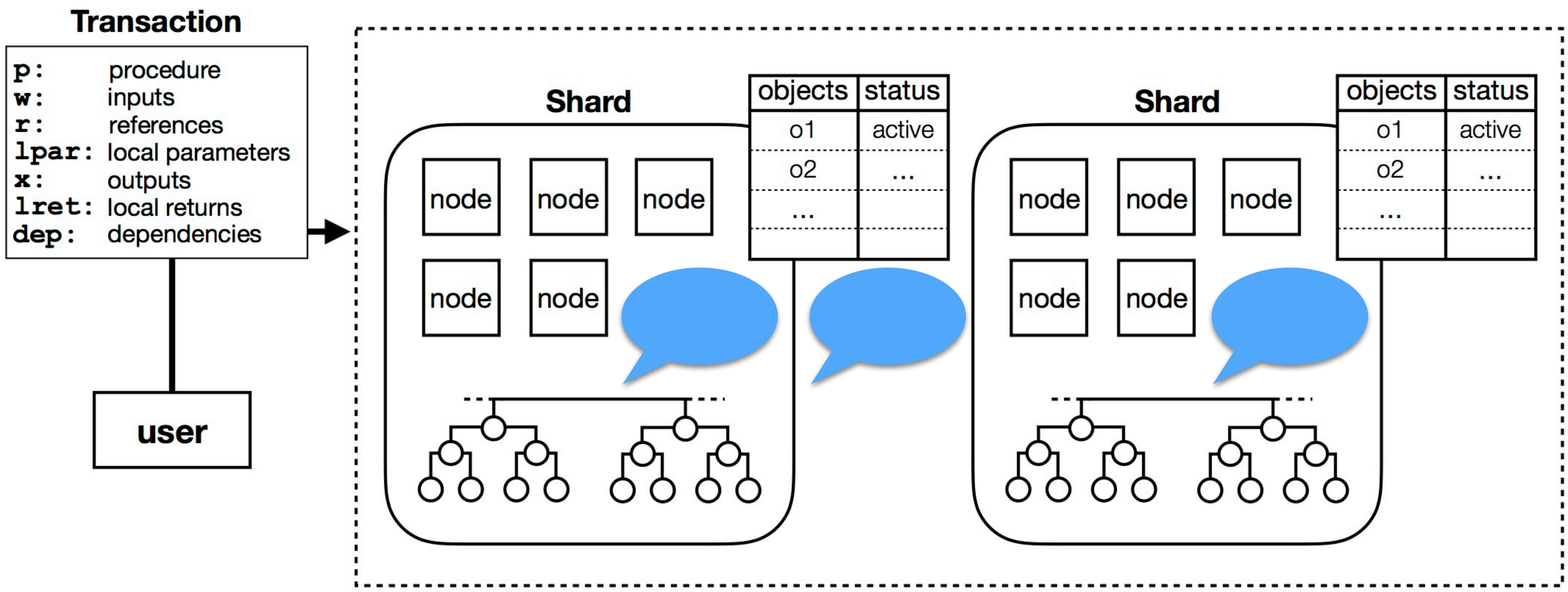
Cast a vote

Check zero-knowledge proof that the vote corresponds to an actual user who possesses a valid ID (secret data)

Contents

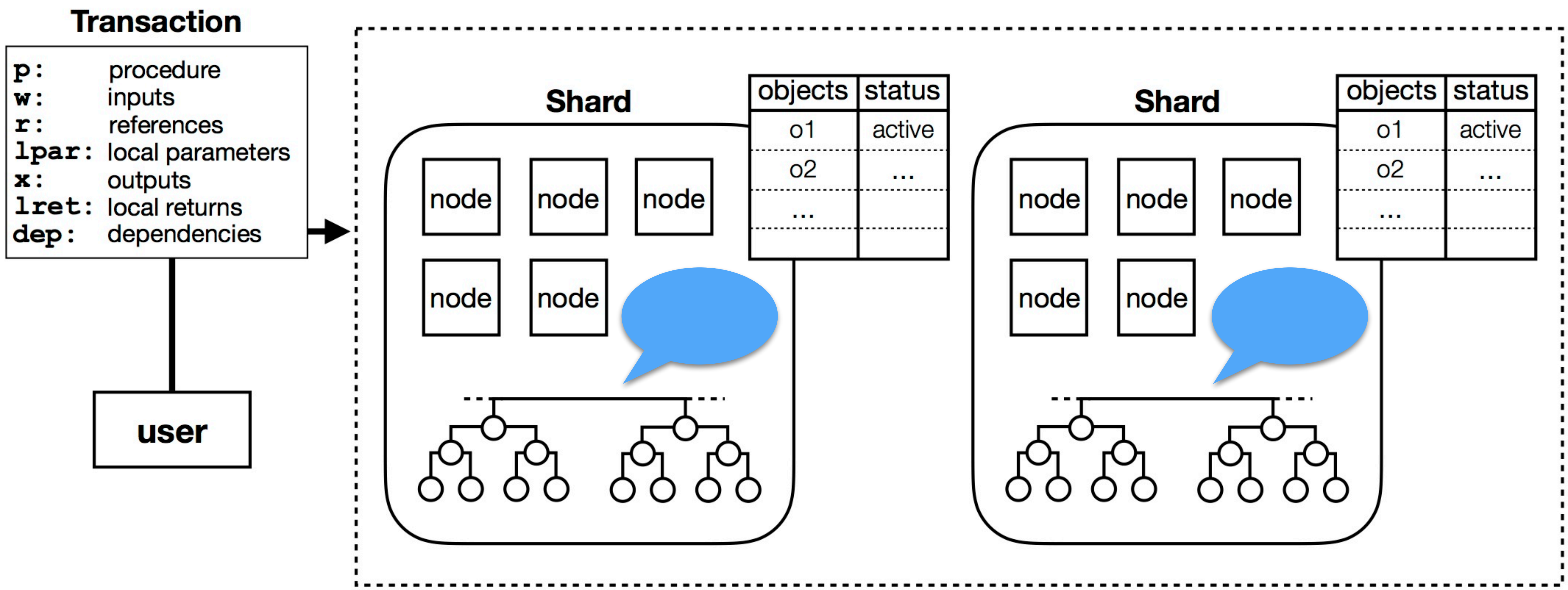

.1.
System Overview
.2.
Privacy by Design
.3.
Scalability
.4.
Security Properties
.5.
Performance

Consensus



- How do nodes agree whether to accept or reject a transaction?

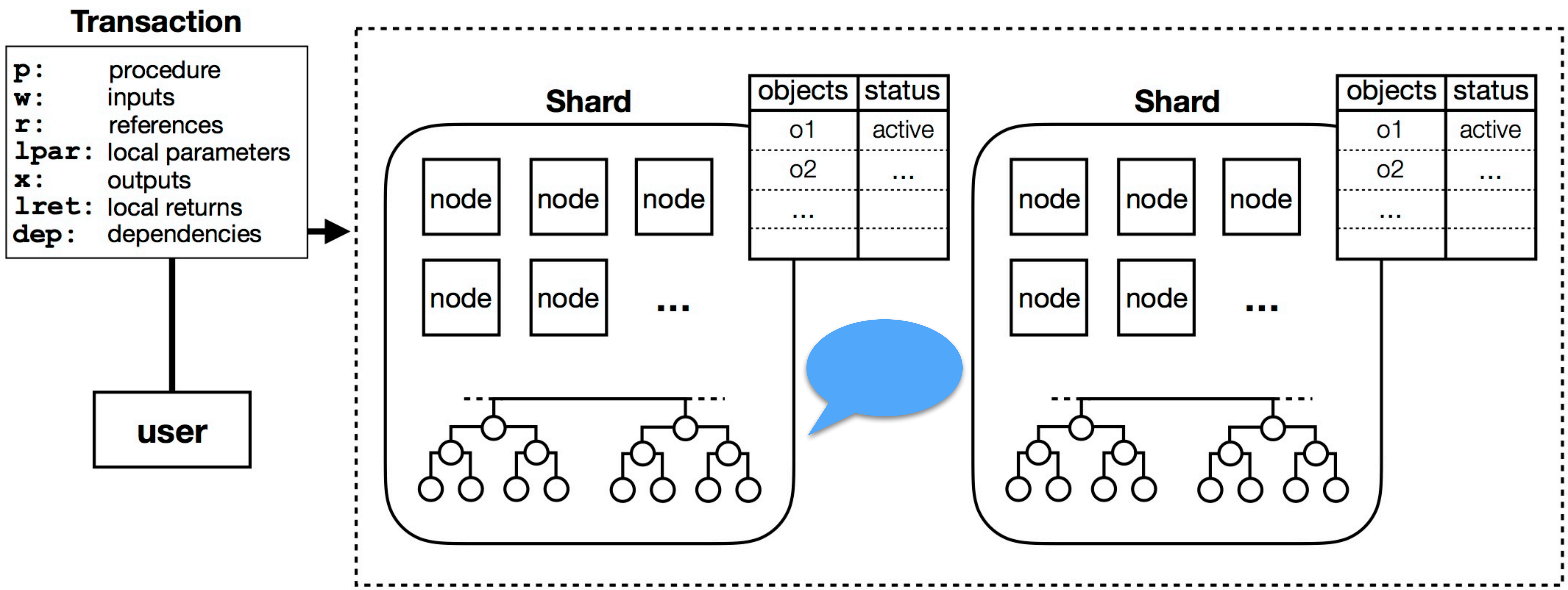
Inter-Shard Consensus



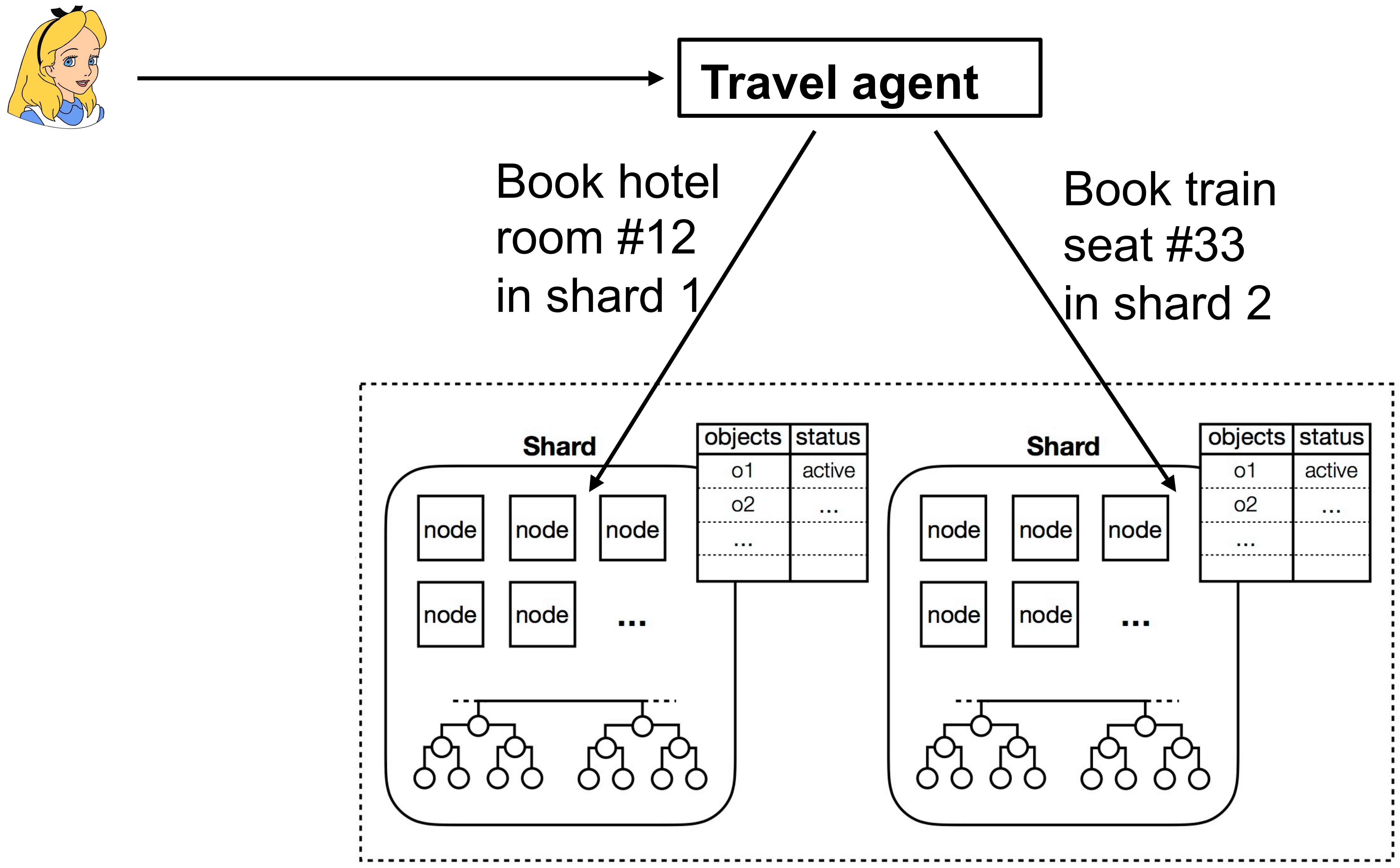
Inter-Shard Consensus

- **Byzantine Fault Tolerant (BFT)** protocol which grauantees:
  - **Safety:** All honest members of a shard of size 3f + 1, agree on a specific common sequence of actions, despite some f malicious nodes within the shard
  - **Liveness:** When agreement is sought, a decision or sequence will eventually be agreed upon

Intra-Shard Consensus



Intra-Shard Consensus


Travel agent
Book hotel room #12 in shard 1
Book train seat #33 in shard 2
Shard
node
...
Shard
node
...

objects	status
o1	active
o2	...
...	

objects	status
o1	active
o2	...
...	

Intra-shard Consensus

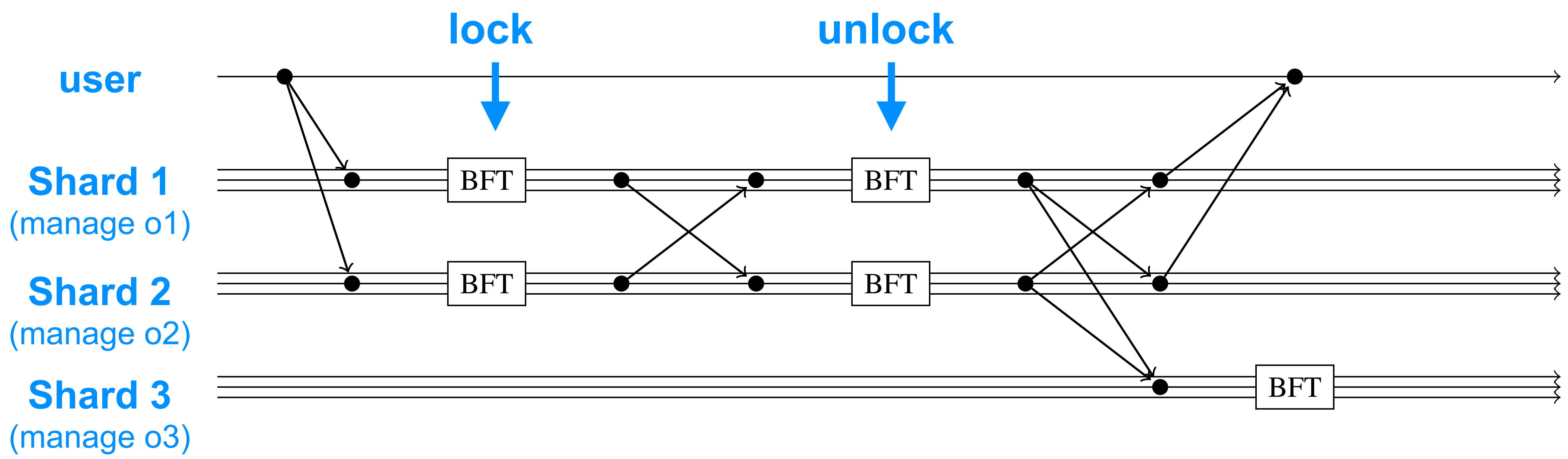
- Atomic commit protocol
  - A transaction is only accepted if all the concerned shards agree, otherwise it is rejected

S-BAC Consensus Protocol

- How nodes reach consensus?

The S-BAC Protocol

Byzantine Agreement + **Atomic Commit**

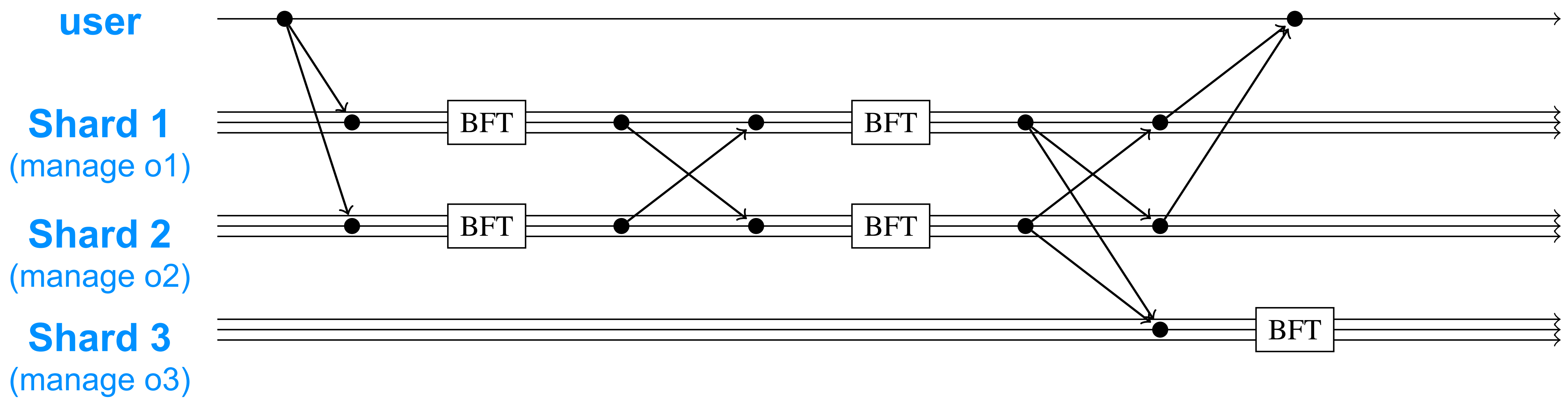


S-BAC enables Scalability

- The Wisdom behind S-BAC

Only shards managing *o1* and *o2* are reaching consensus

Shard 1 and Shard 2 can work in parallel



Contents

Security Properties

- What does Chainspace guarantee?
  - **Honest Shard:** among ***3f+1*** nodes, at most ***f*** are malicious.
  - **Malicious Shard:** over ***f*** dishonest nodes.
  - Chainspace properties:

Security Properties

- What does Chainspace guarantee?
  - **Honest Shard:** among ***3f+1*** nodes, at most ***f*** are malicious.
  - **Malicious Shard:** over ***f*** dishonest nodes.
  - Chainspace properties:

Transparency

Anyone can authenticate the history of transactions and objects that led to the creation of an object.

Encapsulation

A smart contract cannot interfere with objects created by another contract (except if defined by that contract).

Integrity
(Honest Shard)

Only valid & non-conflicting transactions will be executed.

Non-Repudiation

Misbehaviour is detectable: there are evidences of misbehaviour pointing to the faulty parties or shards.

Security Properties

- What does Chainspace guarantee?
  - **Honest Shard:** among ***3f+1*** nodes, at most ***f*** are malicious.
  - **Malicious Shard:** over ***f*** dishonest nodes.
  - Chainspace properties:

Transparency

Anyone can authenticate the history of transactions and objects that led to the creation of an object.

Encapsulation

A smart contract cannot interfere with objects created by another contract (except if defined by that contract).

Integrity
(Honest Shard)

Only valid & non-conflicting transactions will be executed.

Non-Repudiation

Misbehaviour is detectable: there are evidences of misbehaviour pointing to the faulty parties or shards.

Security Properties

- What does Chainspace guarantee?
  - **Honest Shard:** among ***3f+1*** nodes, at most ***f*** are malicious.
  - **Malicious Shard:** over ***f*** dishonest nodes.
  - Chainspace properties:

Transparency

Anyone can authenticate the history of transactions and objects that led to the creation of an object.

Encapsulation

A smart contract cannot interfere with objects created by another contract (except if defined by that contract).

Integrity
(Honest Shard)

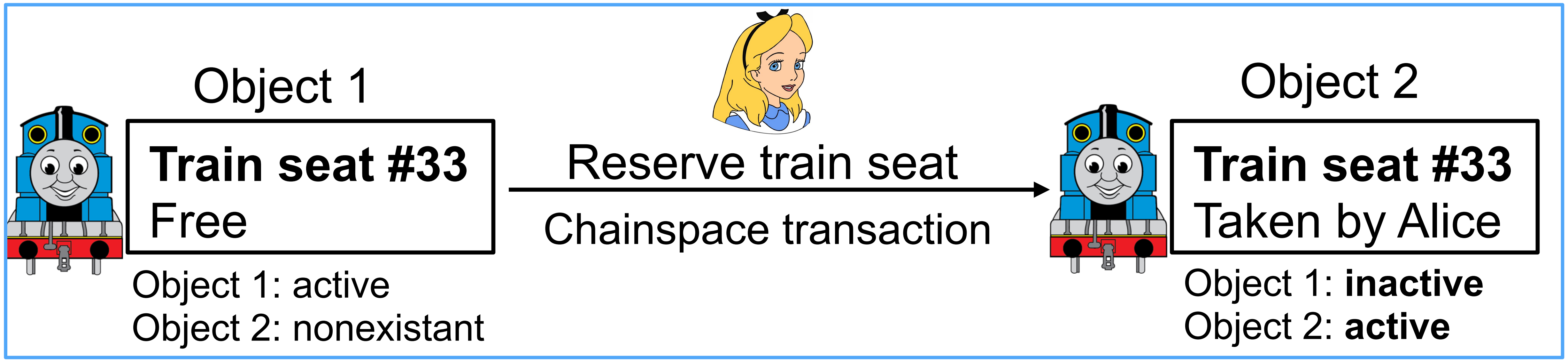
Only valid & non-conflicting transactions will be executed.

Non-Repudiation

Misbehaviour is detectable: there are evidences of misbehaviour pointing to the faulty parties or shards.

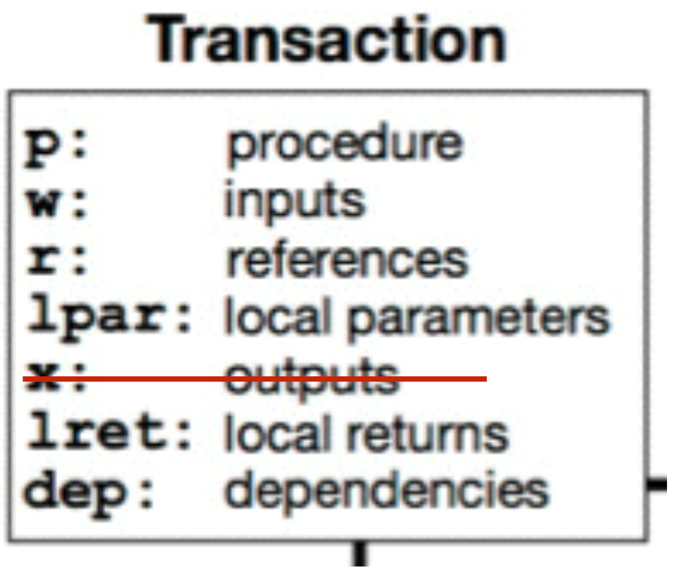
Hash-DAG Structure

- Objects and transactions naturally form a directed acyclic graph (DAG)
  - Directed graph: transactions take as input active objects, render them inactive, and create a new set of active output objects
  - No cycles: Each object may only be created by a single transaction



Hash-DAG Structure

- Every transaction T has an id
  - id_T is Hash(all input info except outputs)

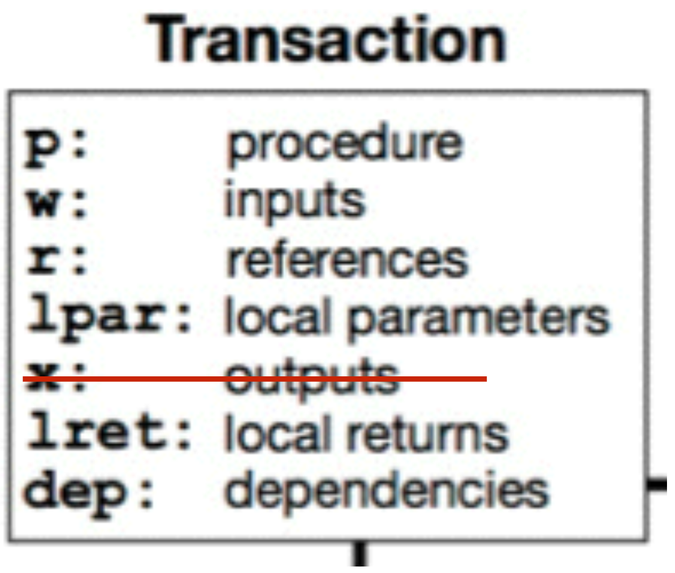


Transaction

`p:`	procedure
`w:`	inputs
`r:`	references
`lpar:`	local parameters
~~`x:`~~	~~outputs~~
`lret:`	local returns
`dep:`	dependencies

Hash-DAG Structure

- Every transaction T has an id
  - id_T is Hash(all input info except outputs)



Transaction

`p:`	procedure
`w:`	inputs
`r:`	references
`lpar:`	local parameters
~~`x:`~~	~~outputs~~
`lret:`	local returns
`dep:`	dependencies

- Every object O has an id
  - id_O is Hash(O || id_T)

Hash-DAG Structure

- Given O, and id_O, it is possible to verify all transactions and previous (now inactive) objects and references that contribute to the existence of O

Security Properties

- What does Chainspace guarantee?
  - **Honest Shard:** among ***3f+1*** nodes, at most ***f*** are malicious.
  - **Malicious Shard:** over ***f*** dishonest nodes.
  - Chainspace properties:

Transparency

Anyone can authenticate the history of transactions and objects that led to the creation of an object.

Encapsulation

A smart contract cannot interfere with objects created by another contract (except if defined by that contract).

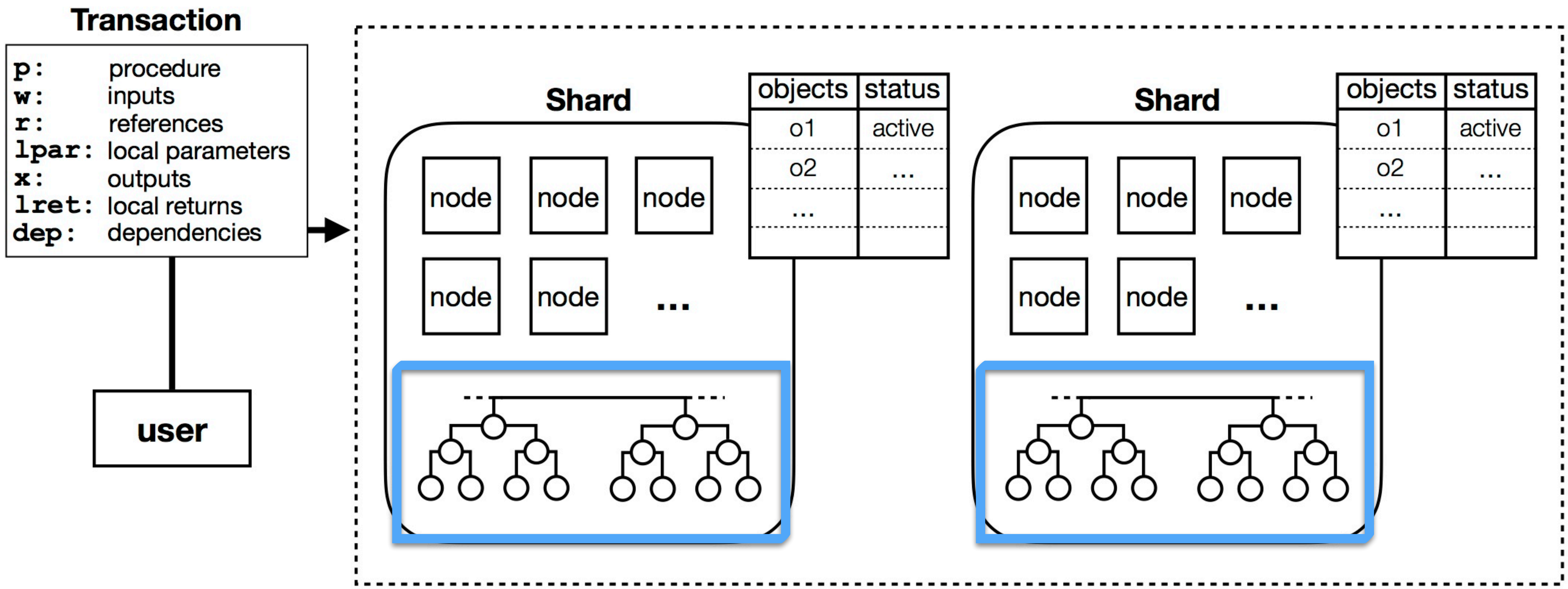
Integrity
(Honest Shard)

Only valid & non-conflicting transactions will be executed.

Non-Repudiation

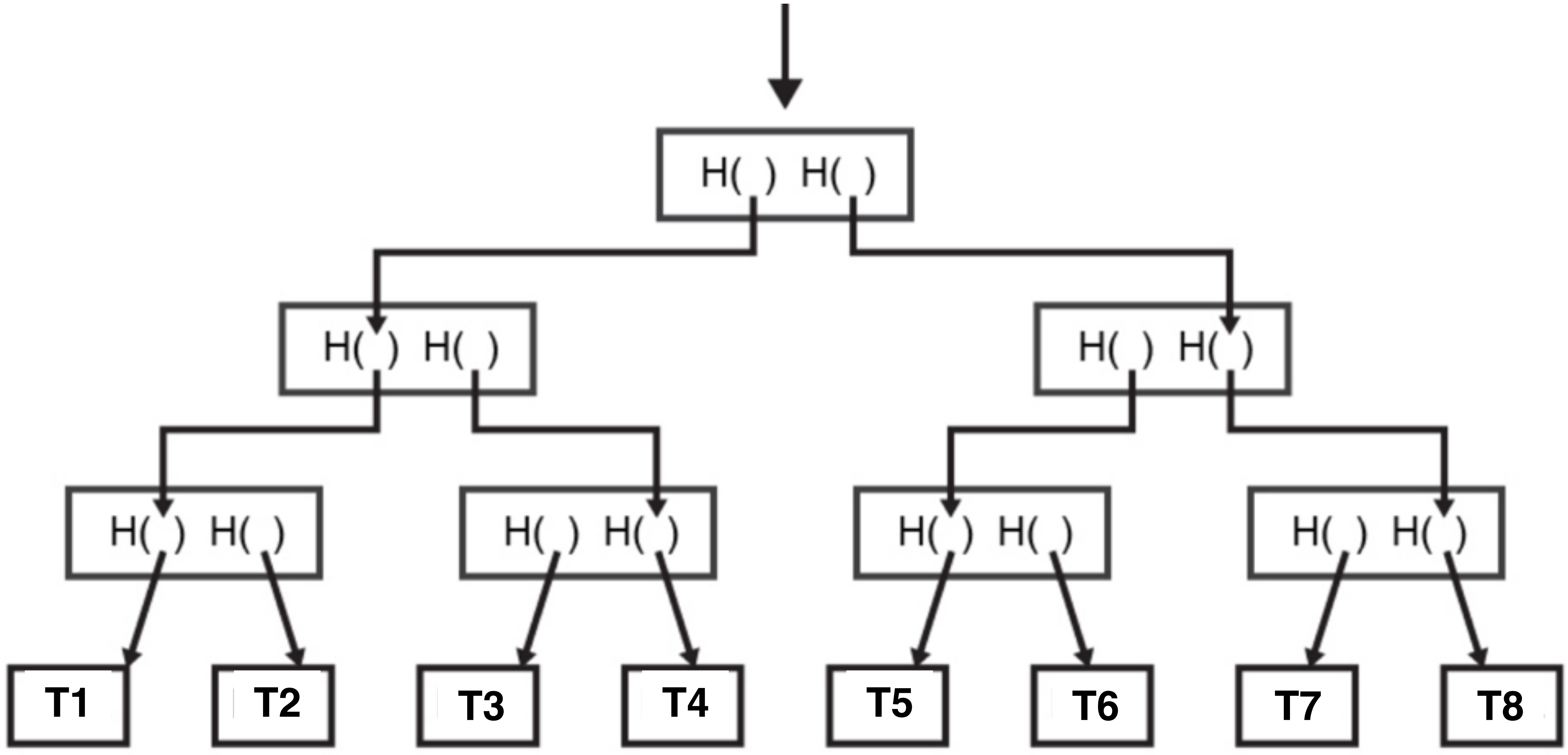
Misbehaviour is detectable: there are evidences of misbehaviour pointing to the faulty parties or shards.

Auditability



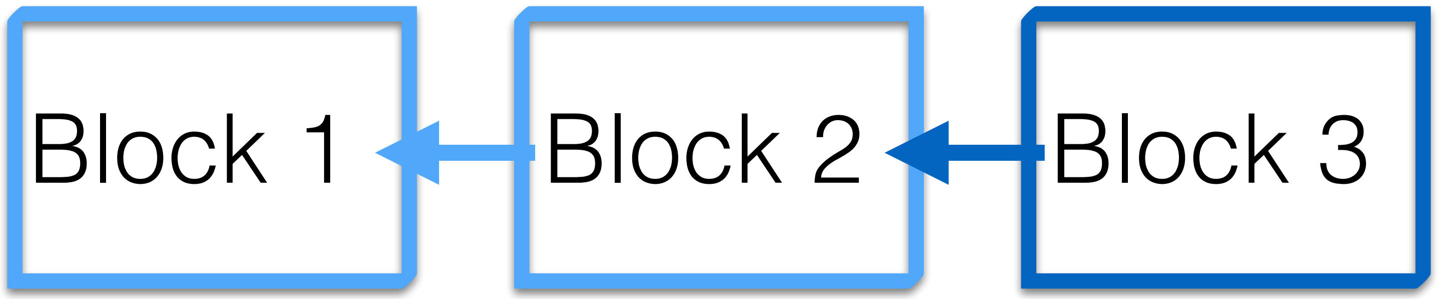
Node Hash-Chains

- Each node in a shard forms a Merkle tree containing all transactions that have been accepted or rejected



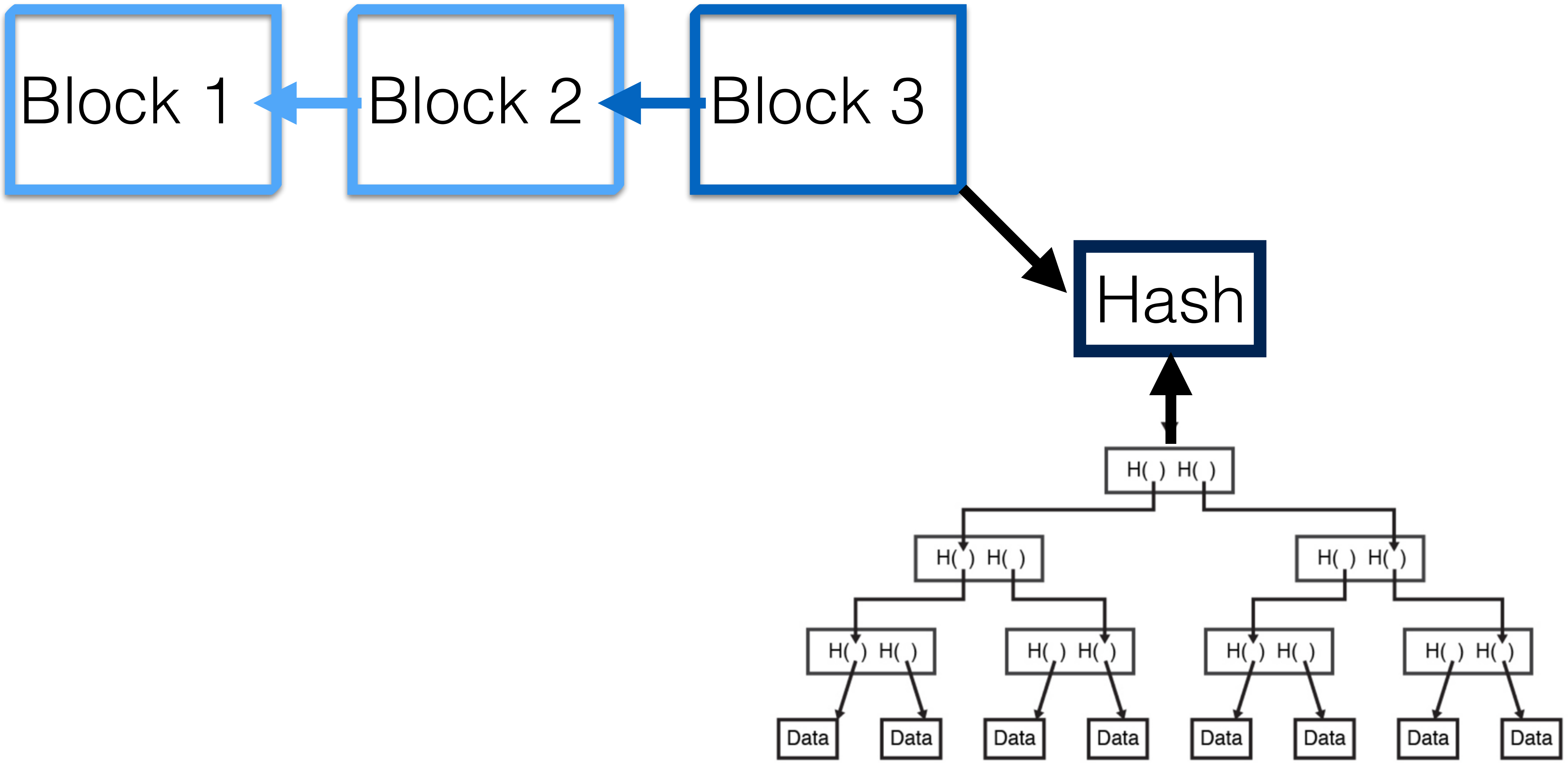
Node Hash-Chains

- Periodically, nodes within a shard consistently agree to seal a checkpoint, as a block of transactions into their hash chains



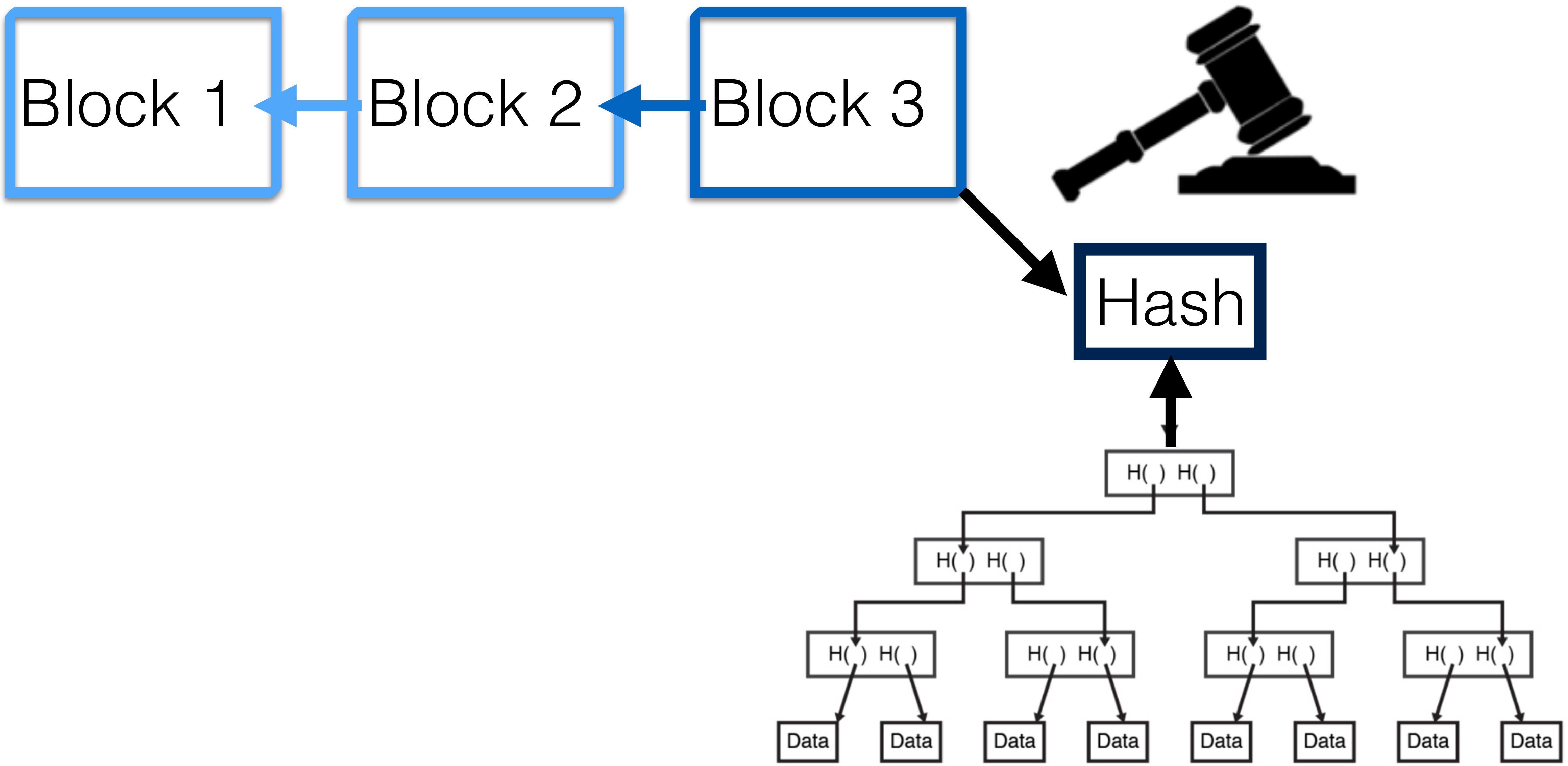
Node Hash-Chains

- Periodically, nodes within a shard consistently agree to seal a checkpoint, as a block of transactions into their hash chains



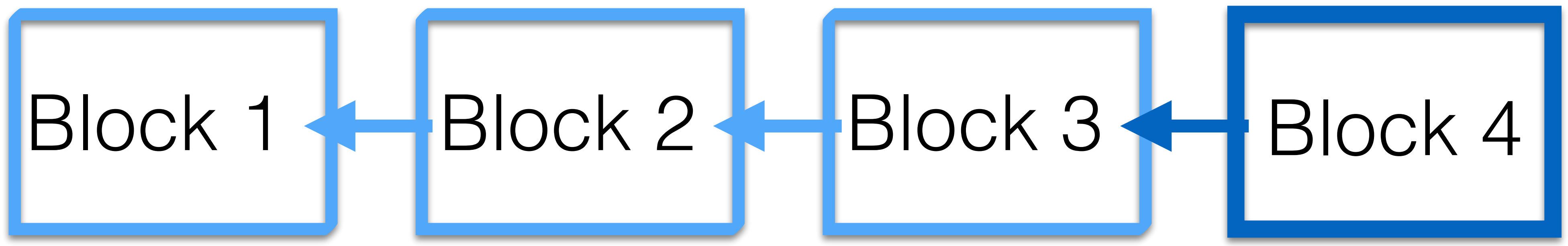
Node Hash-Chains

- Periodically, nodes within a shard consistently agree to seal a checkpoint, as a block of transactions into their hash chains



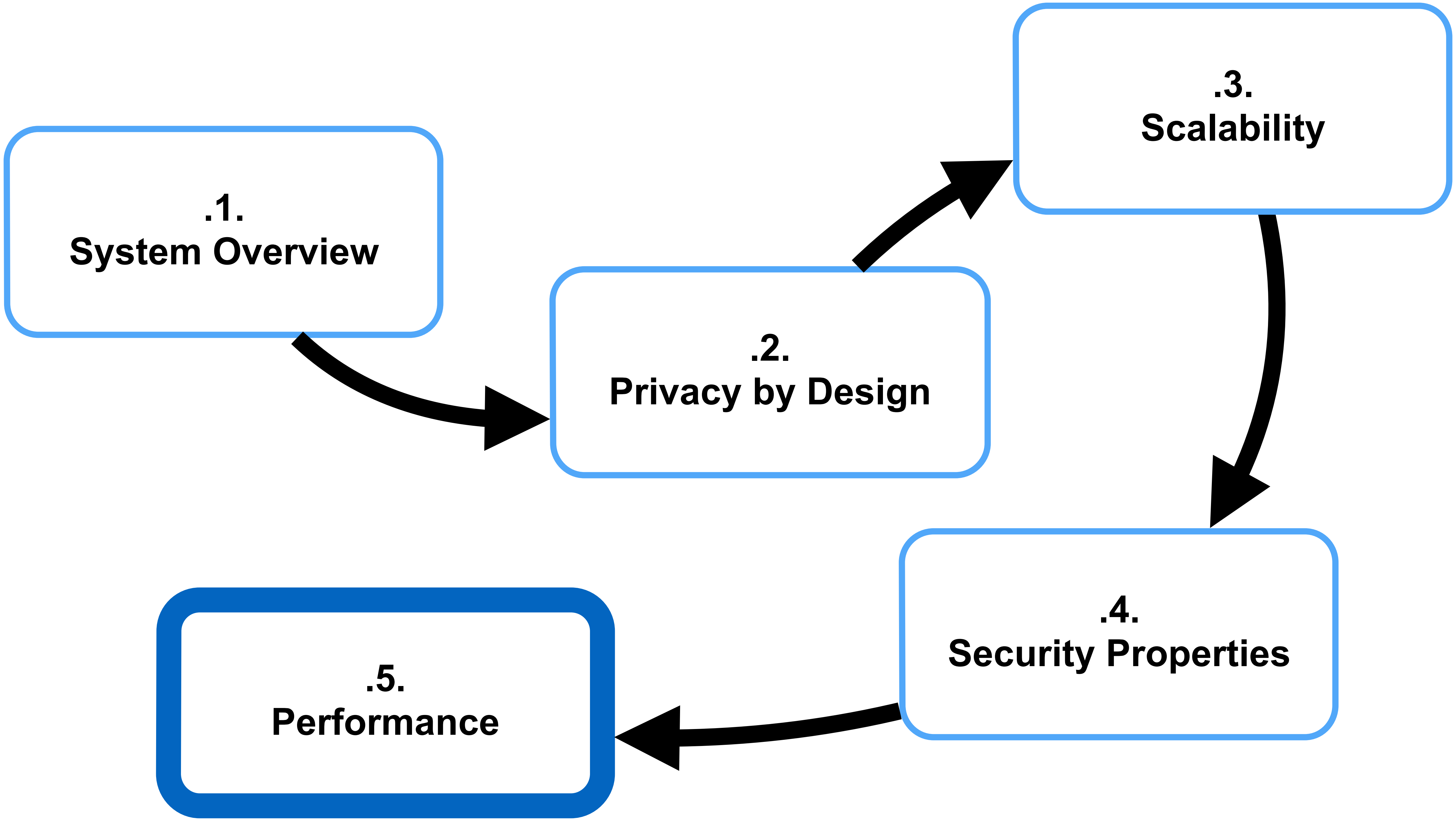
Node Hash-Chains

- Periodically, nodes within a shard consistently agree to seal a checkpoint, as a block of transactions into their hash chains



- Auditing involves re-executing transactions and comparing the result with the hash-chain

Contents


.1.
System Overview
.2.
Privacy by Design
.3.
Scalability
.4.
Security Properties
.5.
Performance

Performance

- What did we implement?

S-BAC protocol implemented in Java

Based on BFT-SMaRt

Performance

- What did we implement?

S-BAC protocol implemented in Java

Based on BFT-SMaRt

Python contract simulator

Helps developers
Simulation of the checker
No need for full deployment

Performance

- What did we implement?

S-BAC protocol implemented in Java

Based on BFT-SMaRt

Python contract simulator

Helps developers
Simulation of the checker
No need for full deployment

Everything is released as open source software

`https://github.com/chainspace`

Performance

- What did we implement?

Measured and tested on Amazon AWS

S-BAC protocol implemented in Java

Based on BFT-SMaRt

Python contract simulator

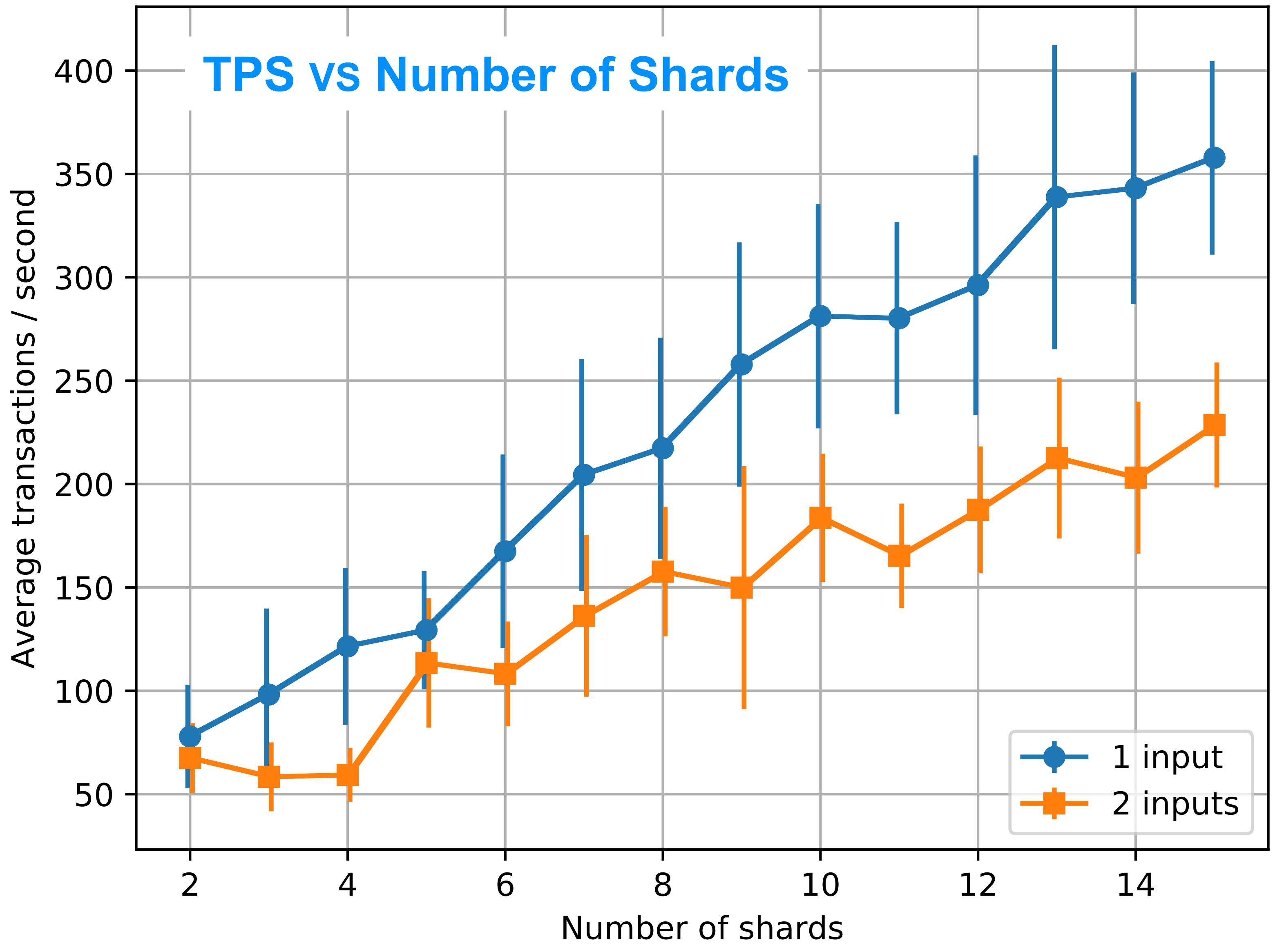
Helps developers
Simulation of the checker
No need for full deployment

Everything is released as open source software

`https://github.com/chainspace`

Performance

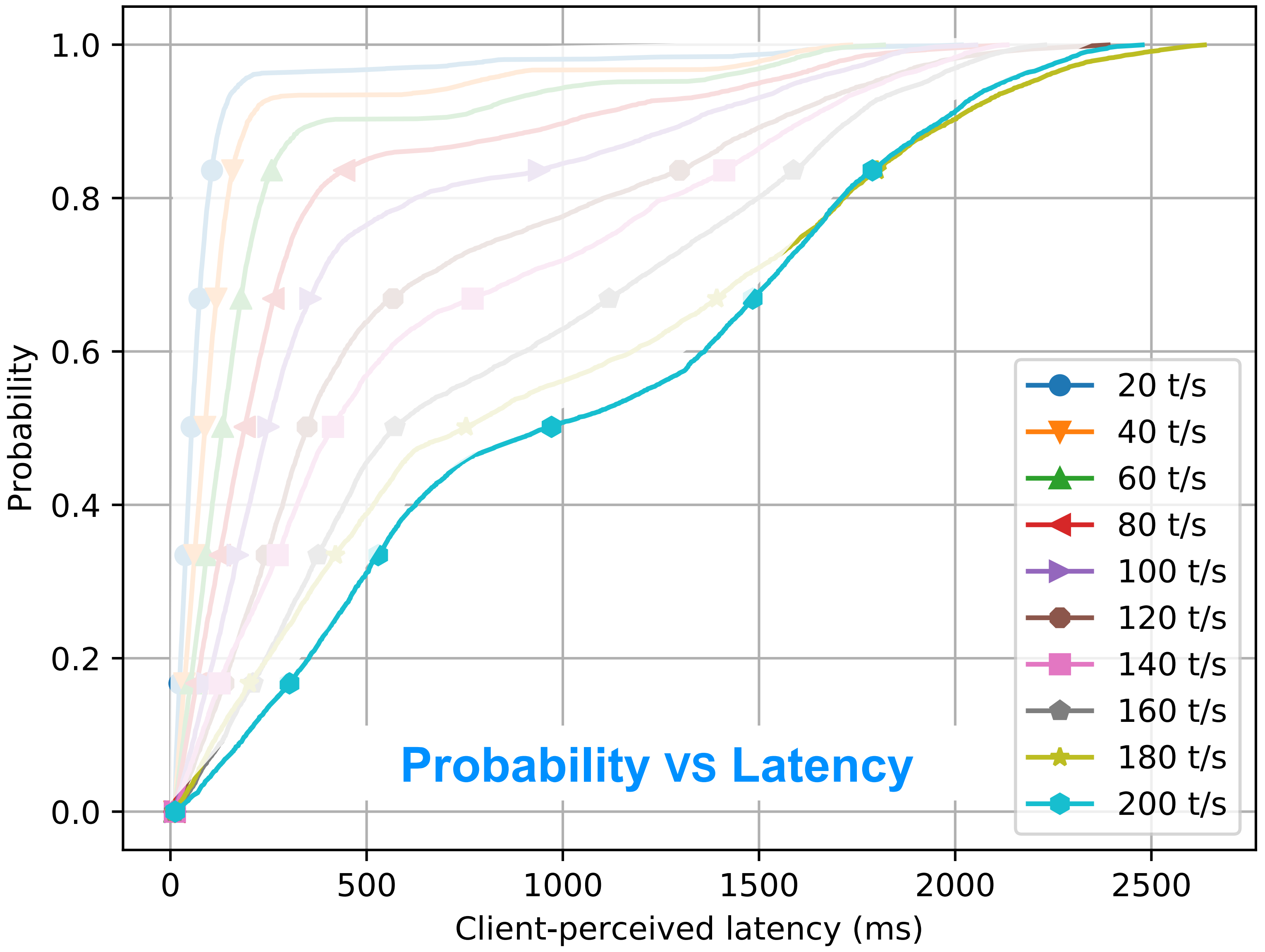
- How the number of shards influences the TPS?



TPS scales linearly with the number of shards

Performance

- How is the trade off between TPS and latency?



Low latency even when the system is heavy loaded

Example Applications

Smart metering contract

Platform for decision making

Benchmarking and evaluation

Chainspace: A Sharded Smart Contracts Platform

Mustafa Al-Bassam*, Alberto Sonnino*, Shehar Bano*, Dave Hrycyszyn† and George Danezis*
* University College London, United Kingdom
† constructiveproof.com

Abstract—**Chainspace is a decentralized infrastructure, known as a distributed ledger, that supports user defined smart contracts and executes user-supplied transactions on their objects. The correct execution of smart contract transactions is verifiable by all. The system is scalable, by sharding state and the execution of transactions, and using $\mathcal{S}$-BAC, a distributed commit protocol, to guarantee consistency. Chainspace is secure against subsets of nodes trying to compromise its integrity or availability properties through Byzantine Fault Tolerance (BFT), and extremely high-auditability, non-repudiation and 'blockchain' techniques. Even when BFT fails, auditing mechanisms are in place to trace malicious participants. We present the design, rationale, and details of Chainspace; we argue through evaluating an implementation of the system about its scaling and other features; we illustrate a number of privacy-friendly smart contracts for smart metering, polling and banking and measure their performance.**

I. Introduction

Chainspace is a distributed ledger platform for high-integrity and transparent processing of transactions within a decentralized system. Unlike application specific distributed ledgers, such as Bitcoin [Nak08] for a currency, or certificate transparency [LLK13] for certificate verification, Chainspace offers extensibility though smart contracts, like Ethereum [Woo14]. However, users expose to Chainspace enough information about contracts and transaction semantics, to provide higher scalability through sharding across infrastructure nodes: our modest testbed of 60 cores achieves 350 transactions per second, as compared with a peak rate of less than 7 transactions per second for Bitcoin over 6K full nodes. Etherium currently processes 4 transactions per second, out of theoretical maximum of 25. Furthermore, our platform is agnostic as to the smart contract language, or identity infrastructure, and supports privacy features through modern zero-knowledge techniques [BCCG16, DGFK14].

Unlike other scalable but 'permissioned' smart contract platforms, such as Hyperledger Fabric [Cac16] or BigchainDB [MMM+16], Chainspace aims to be an 'open' system: it allows anyone to author a smart contract, anyone to provide infrastructure on which smart contract code and state runs, and any user to access calls to smart contracts. Further, it provides ecosystem features, by allowing composition of smart contracts from different authors. We integrate a value system, named CSCoin, as a system smart contract to allow for accounting between those parties.

However, the security model of Chainspace, is different from traditional unpermissioned blockchains, that rely on proof-of-work and global replication of state, such as Ethereum. In Chainspace smart contract authors designate the parts of the infrastructure that are trusted to maintain the integrity of their contract—and only depend on their correctness, as well as the correctness of contract sub-calls. This provides fine grained control of which part of the infrastructure need to be trusted on a per-contract basis, and also allows for horizontal scalability.

This paper makes the following contributions:

- It presents Chainspace, a system that can scale arbitrarily as the number of nodes increase, tolerates byzantine failures, and can be fully and publicly audited.
- It presents a novel distributed atomic commit protocol, called $\mathcal{S}$-BAC, for sharding generic smart contract transactions across multiple byzantine nodes, and correctly coordinating those nodes to ensure safety, liveness and security properties.
- It introduces a distinction between parts of the smart contract that execute a computation, and those that check the computation and discusses how that distinction is key to supporting privacy-friendly smart-contracts.
- It provides a full implementation and evaluates the performance of the byzantine distributed commit protocol, $\mathcal{S}$-BAC, on a real distributed set of nodes and under varying transaction loads.
- It presents a number of key system and application smart contracts and evaluates their performance. The contracts for privacy-friendly smart-metering and privacy-friendly polls illustrate and validate support for high-integrity and high-privacy applications.

Outline: Section II presents an overview of Chainspace; Section III presents the client-facing application interface; Section IV presents the design of internal data structures guaranteeing integrity, the distributed architecture, the byzantine commit protocols, and smart contract definition and composition. Section V argues the correctness and security; specific smart contracts and their evaluations are presented in Section VI; Section VII presents an evaluation of the core protocols and smart contract performance; Section VIII presents limitation and Section IX a comparison with related work; and Section X concludes.

Future Work

1. **How to recover from malicious shards?**

2. **How can a smart contract creator avoid dishonest shards ?**

Future Work

3. **How to bootstrap shards?**

4. **How to incentivise nodes?**

Conclusions

contribution I

Scalable smart contract platform

sharding

contribution II

Supporting privacy

execution
/ checker

CHAINSPACE

Thanks
Q/A

@thatBano
s.bano@ucl.ac.uk
http://sheharbano.com

https://github.com/chainspace

EPSRC Engineering and Physical Sciences Research Council

decode

THE ALAN TURING INSTITUTE

This work is supported in part by EPSRC Grant EP/M013286/1, the EU H2020 DECODE project (grant agreement number 732546), and The Alan Turing Institute.