From Crypto-Paper to Crypto-Currency: the Cardano Consensus Layer

Real-World Crypto Symposium (RWC) 2021

Christian Badertscher
IOHK

Philipp Kant
IOHK

Duncan Coutts
IOHK
Well-Typed

Joint work with Peter Gaži, Aggelos Kiayias, Alexander Russell

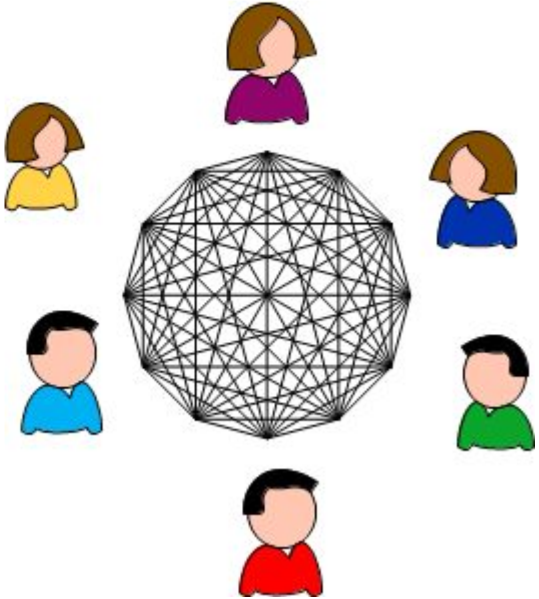
Overview

1. From Theory to Practice
2. Formal Methods and Implementation Correctness
3. What Could Have Gone Wrong
4. Path to Decentralisation

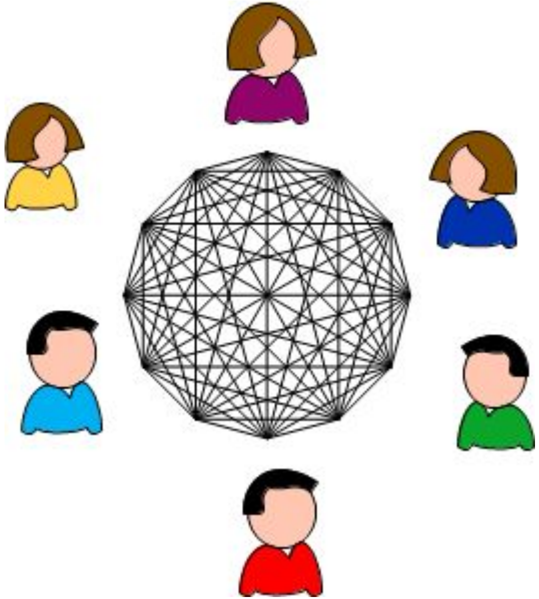
Overview

1. **From Theory to Practice**
2. Formal Methods and Implementation Correctness
3. What Could Have Gone Wrong
4. Path to Decentralisation

Distributed Computation over a Network



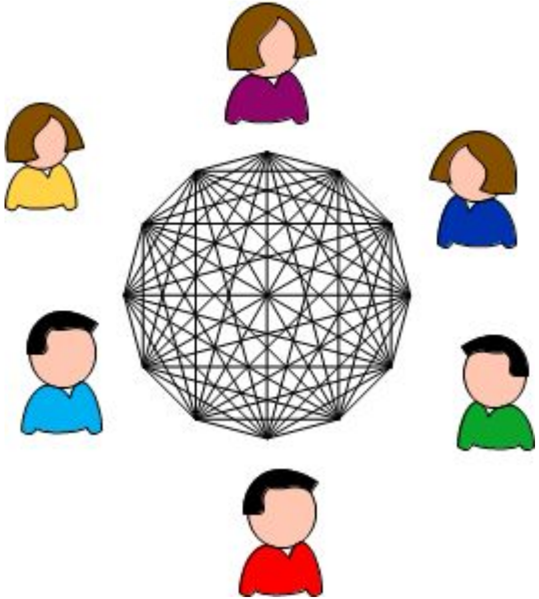
Distributed Computation over a Network



Protocol execution model:

- Interactive machines

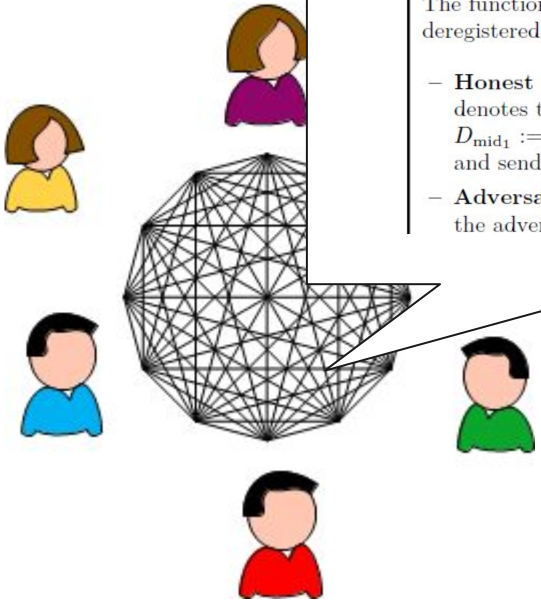
Distributed Computation over a Network



Protocol execution model:

- Interactive machines
- **Formalizations of assumed resources (network, clock, random oracle, …)**

Distributed Computation over a Network



Functionality $\mathcal{F}^{\Delta}_{\text{N-MC}}$

The functionality is parameterized with a set possible send
deregistered) party is added to (resp. deleted from) $\mathcal{P}$.

– **Honest sender multicast.** Upon receiving (MULTICAS
denotes the current party set, choose n new unique mes
$D_{\text{mid}_1} := D^{MAX}_{\text{mid}_1} \ldots := D_{\text{mid}_n} := D^{MAX}_{\text{mid}_n} := 1$, set $\vec{M} :=$
and send $(\text{MULTICAST}, \text{sid}, m, U_p, (U_1, \text{mid}_1), \ldots, (U_n, \text{mi}$

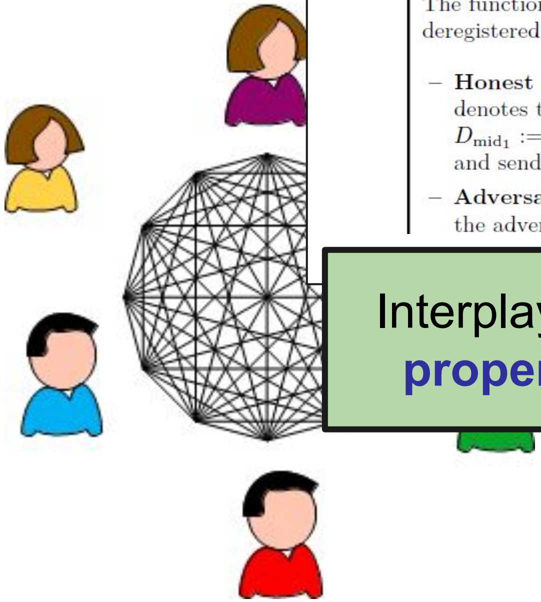
– **Adversarial sender (partial) multicast.** Upon recei
the adversary with $\{U_{i_1}, \ldots, U_{i_\ell}\} \subset \mathcal{P}$, choose ℓ new un

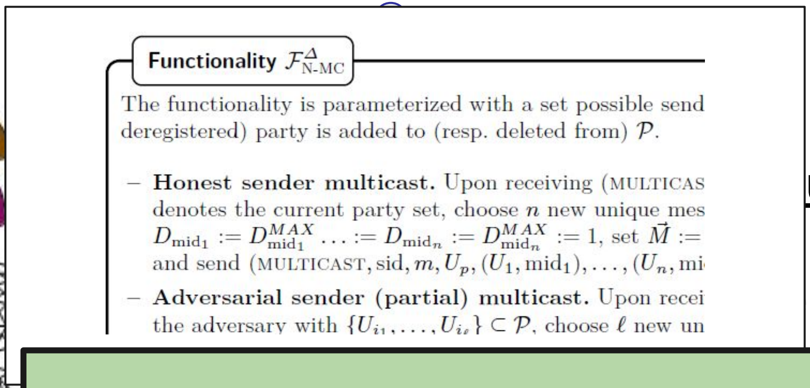
<u>ution model:</u>

machines

tions of assumed resources (network, clock, random oracle, …)

Distributed Computation over a Network





Functionality $\mathcal{F}^{\Delta}_{\text{N-MC}}$

The functionality is parameterized with a set possible send
deregistered) party is added to (resp. deleted from) $\mathcal{P}$.

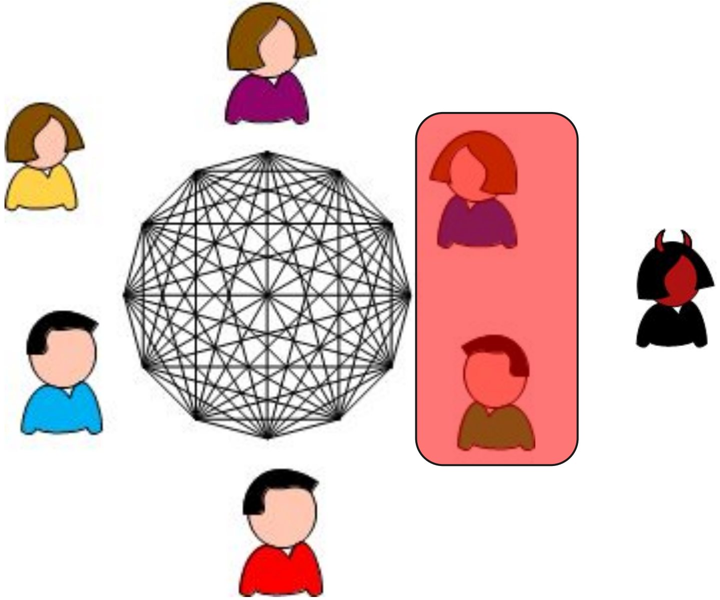
- **Honest sender multicast.** Upon receiving (MULTICAS
denotes the current party set, choose n new unique mes
$D_{\text{mid}_1} := D^{MAX}_{\text{mid}_1} \ldots := D_{\text{mid}_n} := D^{MAX}_{\text{mid}_n} := 1$, set $\vec{M} :=$
and send (MULTICAST, sid, $m, U_p, (U_1, \text{mid}_1), \ldots, (U_n, \text{mi}$
- **Adversarial sender (partial) multicast.** Upon recei
the adversary with $\{U_{i_1}, \ldots, U_{i_\ell}\} \subset \mathcal{P}$, choose ℓ new un

<u>ution model:</u>

machines

Interplay between theory and engineering: **relevant properties must be captured** by the abstraction

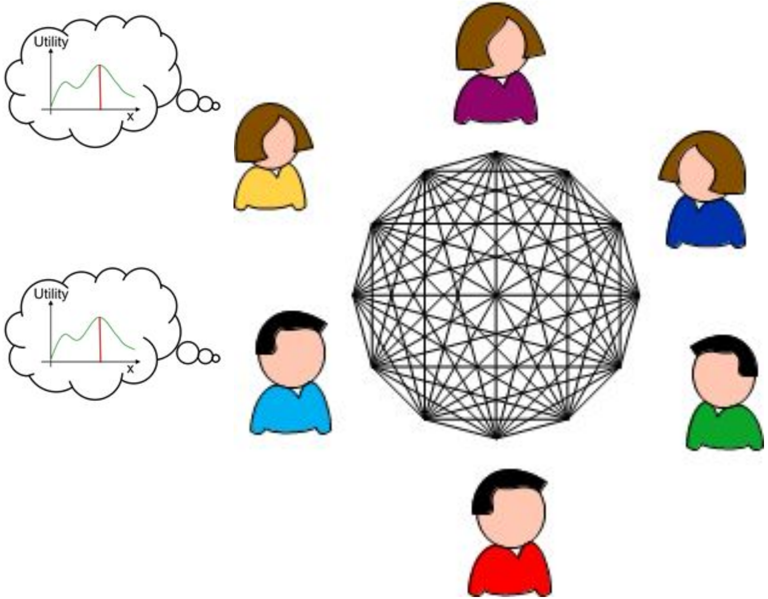
Distributed Computation over a Network



Protocol execution model:

- Interactive machines
- Formalizations of assumed resources (network, clock, random oracle, …)
- **Byzantine behavior**

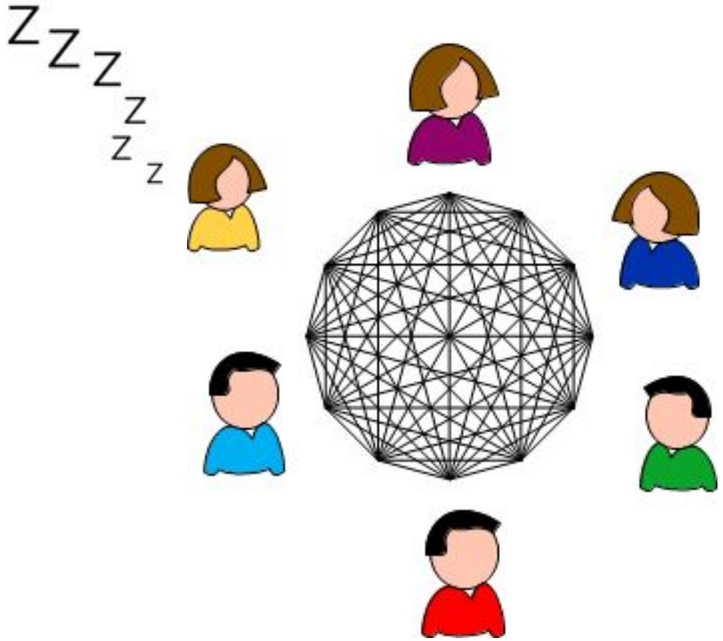
Distributed Computation over a Network



Protocol execution model:

- Interactive machines
- Formalizations of assumed resources (network, clock, random oracle, …)
- Byzantine behavior
- **Rational behavior**

Distributed Computation over a Network



Protocol execution model:

- Interactive machines
- Formalizations of assumed resources (network, clock, random oracle, …)
- Byzantine behavior
- Rational behavior
- **Machine failures**

Distributed Computation over a Network

Fine-Grained security model

+ **Security under composition:**
  Each external input might depend on the entire view of this and other protocols

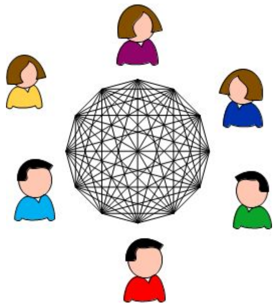
⇒ **A security proof is meaningful to practice.**

Protocol execution model:

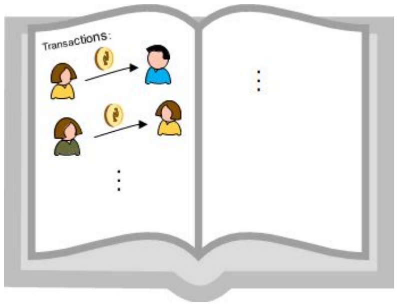
- Interactive machines
- Formalizations of assumed resources (network, clock, random oracle, …)
- Byzantine behavior
- Rational behavior
- Machine failures

Consensus Layer of Cardano

Goal:



Realize



<u>Ledger:</u>

- Persistence
- Liveness

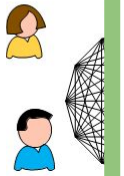
(Consensus on ledger state)

Result:

<u>Ouroboros:</u>

A Nakamoto-style **Proof-of-Stake Blockchain Protocol** realizing a ledger

Goal:



Nakamoto-Style Blockchain:

B_0

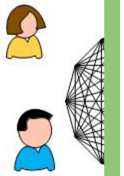
Genesis
Block

Result:

Ouroboros:

A Nakamoto-style **Proof-of-Stake Blockchain Protocol** realizing a ledger

Goal:



Nakamoto-Style Blockchain:

Genesis Block

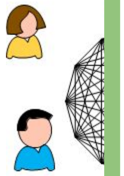
- Lottery on next block proposer(s).
- Proposers elected proportional to owned stake.

Result:

Ouroboros:

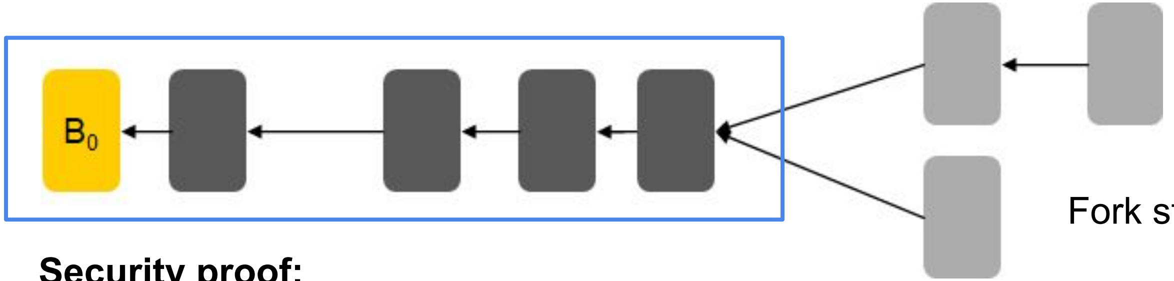
A Nakamoto-style **Proof-of-Stake Blockchain Protocol** realizing a ledger

Goal:



Result:

Nakamoto-Style Blockchain:

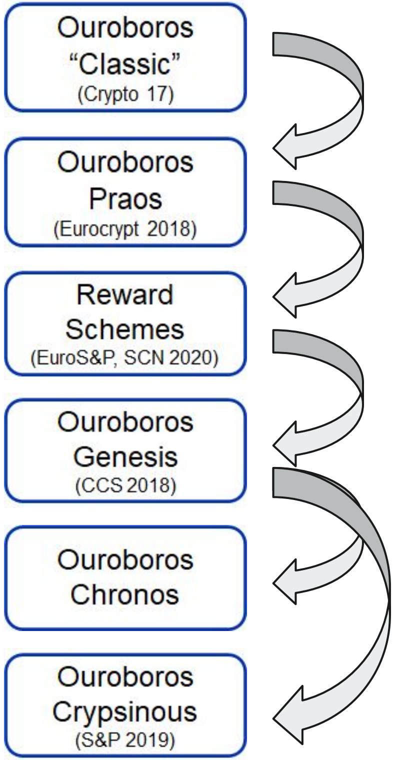


Security proof:
Establish **common-prefix, chain-growth, chain-quality** properties (under honest majority of stake assumption).

Ouroboros:

A Nakamoto-style **Proof-of-Stake Blockchain Protocol** realizing a ledger

Theory + Practice Interplay



- Series of results to overcome specific design challenges

- **Theoretical papers** live in an **abstract, proof-oriented world:**
  - Abstraction of components
  - Pseudocode
  - Asymptotic statements
  - Not usually implementation friendly

Theory + Practice Interplay

Ouroboros "Classic" (Crypto 17)

Ouroboros Praos (Eurocrypt 2018)

Reward Schemes (EuroS&P, SCN 2020)

Ouroboros Genesis (CCS 2018)

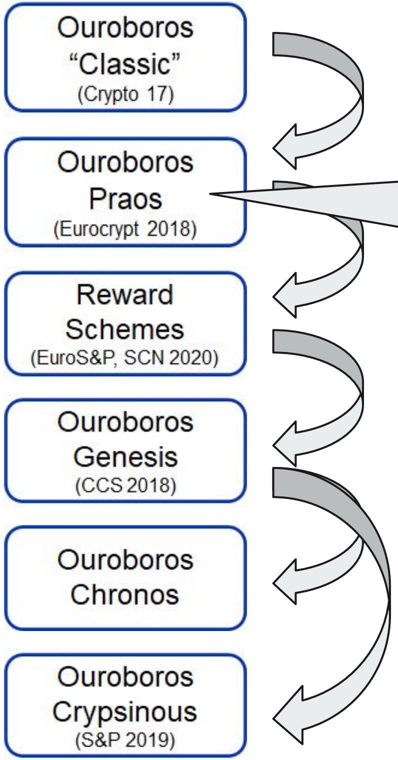
Ouroboros Chronos

Ouroboros Crypsinous (S&P 2019)

Synchronous model of computation and static corruption

- **Theoretical papers** live in an **abstract, proof-oriented world:**
  - Abstraction of components
  - Pseudocode
  - Asymptotic statements
  - Not usually implementation friendly

Theory + Practice Interplay



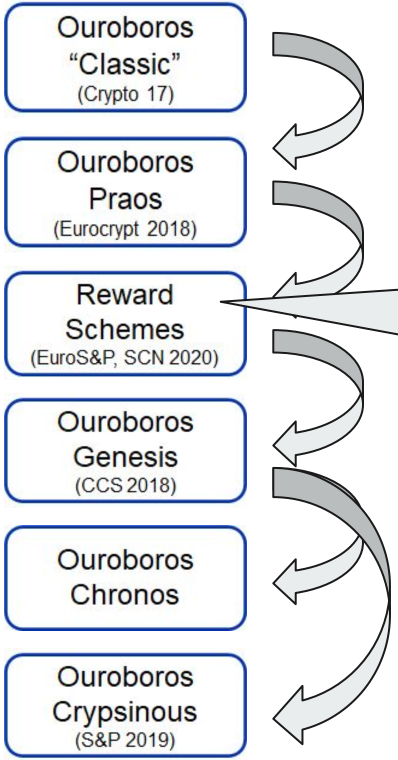
- Series of results to overcome specific design

Remove rigid message-passing round structure and tolerate adaptive corruptions

proof-oriented world:

- Abstraction of components
- Pseudocode
- Asymptotic statements
- Not usually implementation friendly

Theory + Practice Interplay

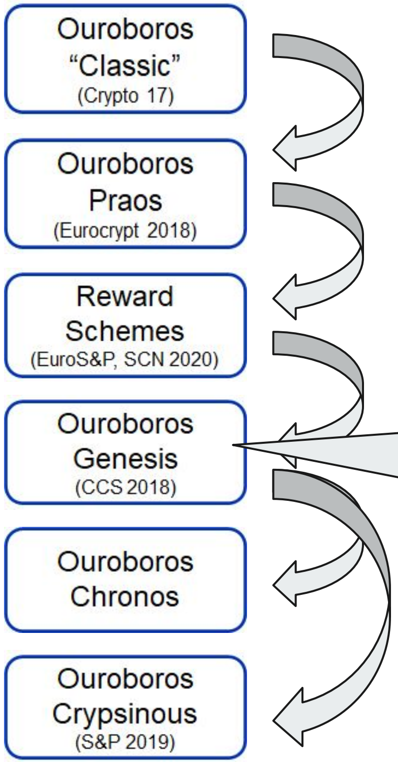


- Series of results to overcome specific design challenges

Reward sharing scheme for stake pools to establish an equilibrium with a large number of pools with high commitment to the system

- Pseudocode
- Asymptotic statements
- Not usually implementation friendly

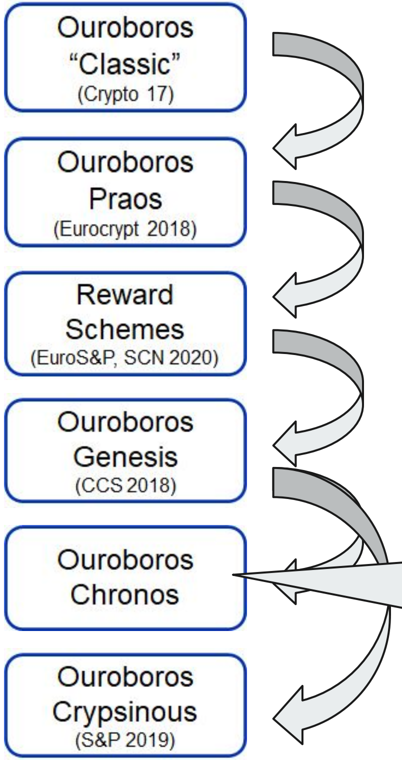
Theory + Practice Interplay



- Series of results to overcome specific design challenges
- **Theoretical papers** live in an **abstract, proof-oriented world:**
  - Not usually implementation friendly

Dealing with fluctuating participation, joining after disconnection in a decentralized manner

Theory + Practice Interplay

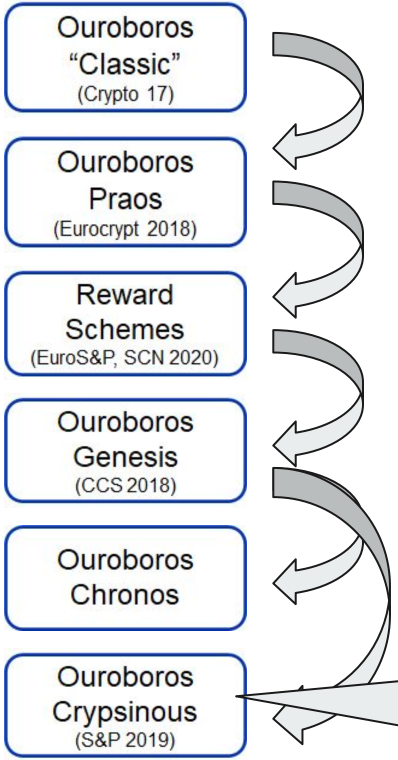


- Series of results to overcome specific design challenges

- **Theoretical papers** live in an **abstract, proof-oriented world:**
  - Abstraction of components
  - Pseudocode

More resilience by less dependency on external timing services

Theory + Practice Interplay

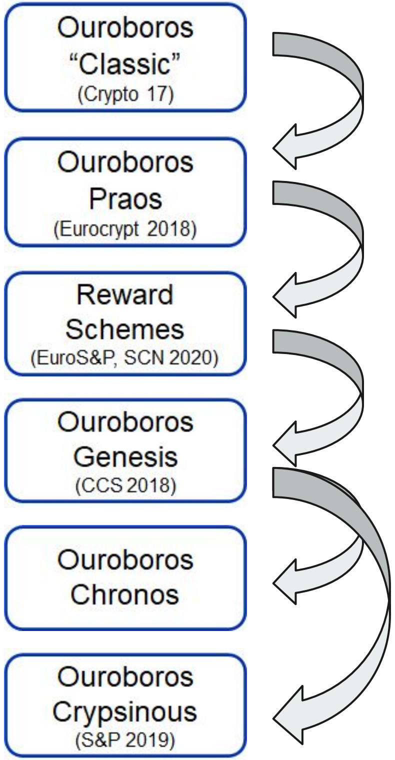


- Series of results to overcome specific design challenges

- **Theoretical papers** live in an **abstract, proof-oriented world:**
  - Abstraction of components
  - Pseudocode
  - Asymptotic statements

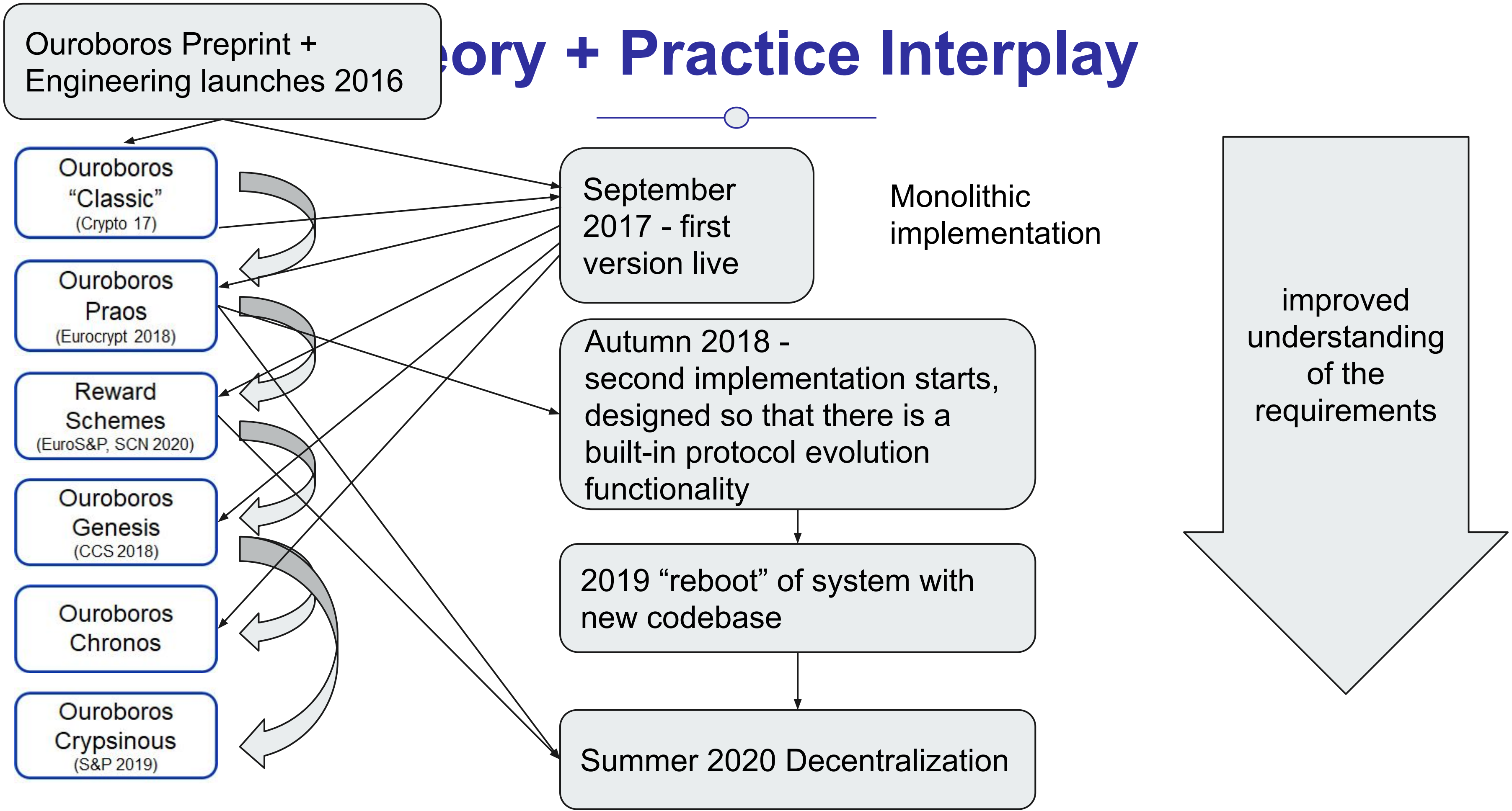
More functionality by privacy

Theory + Practice Interplay



Everything so ordered ...

eory + Practice Interplay



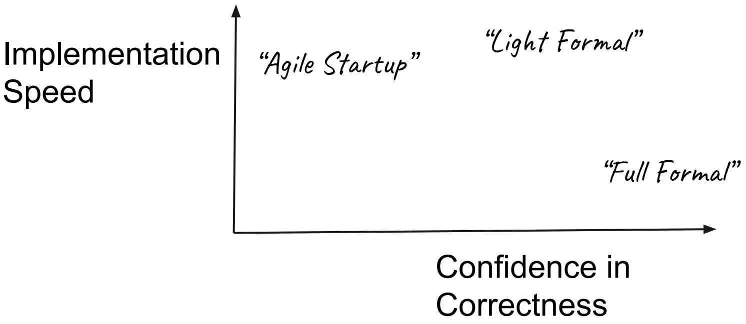
Overview

1. From Theory to Practice
2. **Formal Methods and Implementation Correctness**
3. What Could Have Gone Wrong
4. Path to Decentralisation

Implementing Correctly ~~or~~ and Quickly?

Questions before implementation

- Required level of confidence?
- Time available?
- Requirements fixed?
- Difficulty of Problem?



Correctness Toolbox

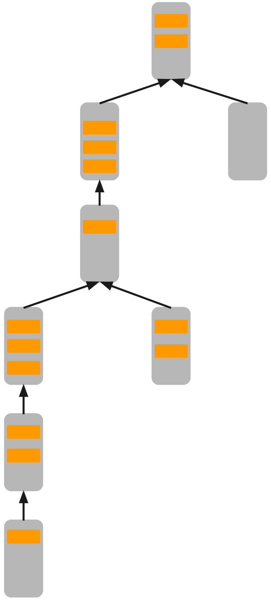
Variety of Tools

- Formal specifications
- Proofs (machine verifiable or manual)
- Model checking
- Property-based tests

System Properties

- Determinism?
- Concurrency?
- Finite state space?

Separating the Concerns

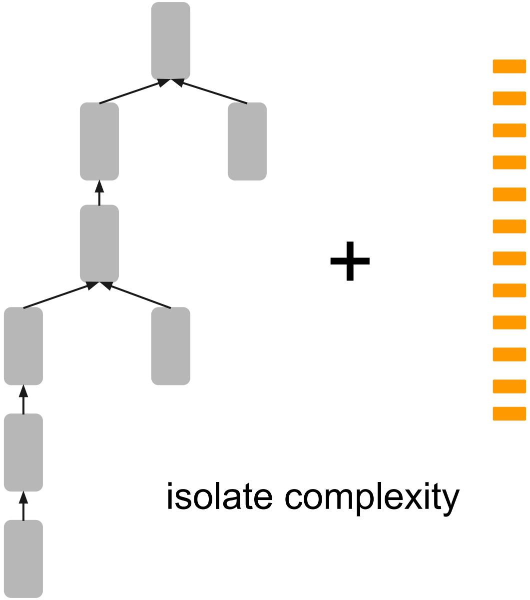


Blockchains are messy

- Forks,
  eventual consistency
- Distributed system,
  concurrency

Ledger can be simple

- Single, linear view
  of history



isolate complexity

Transaction Ledger

$$\frac{\text{txins}\ tx \subseteq \text{dom}\ utxo \qquad \text{minfee}\ pc\ tx \leqslant \text{txfee}\ tx \\ \text{balance (txouts}\ tx) + \text{txfee}\ tx = \text{balance (txins}\ tx \lhd utxo)}{pc \vdash utxo \xrightarrow[\text{UTXO}]{tx} (\text{txins}\ tx \not\lhd utxo) \cup \text{txouts}\ tx}$$

```
Rule
  [Predicate $ λ(_, utxo, tx) →
     txins tx ‘Set.isSubsetOf‘ dom utxo ?! BadInputs
  , Predicate $ λ(pc, _, tx) →
     pcMinFee pc tx ⩽ txfee tx            ?! FeeTooLow
  , Predicate $ λ(_, utxo, tx) →
        balance (txouts tx) <> txfee tx
      ≡ balance (txins tx ◁ utxo)          ?! IncreasedTotalBalance
  ]
  (Extension ∘ Transition $
     λ(pc, utxo, tx) → (txins tx ⋪ utxo) ∪ txouts tx)
```

Formal/executable specifications

- Describe microscopic behaviour
  “small step operational semantics”

Correctness:

- Formulate macroscopic properties
  “money is conserved”
- Property-based testing
  `QuickCheck`
- Prove them (manually, assisted)

Consensus and Networking

Harder: Concurrency, Distributed Computing

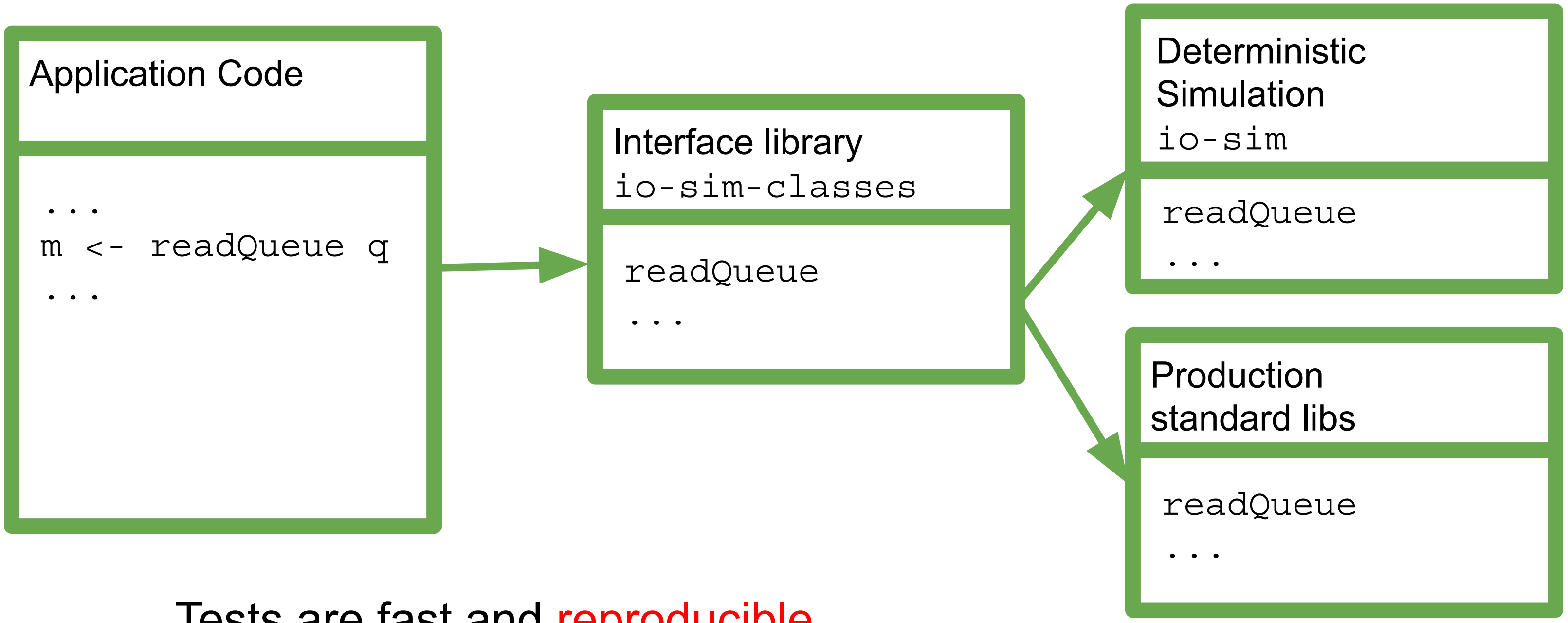
- Language of process calculi
- Not feasible to start from full formal specification within timeframe

Strategy

- Write testable code → deterministic simulation
  property-based testing

- Use type system for guaranteeing invariants
- “Catch Up” on the formal side

Testing Concurrent Code in Simulation



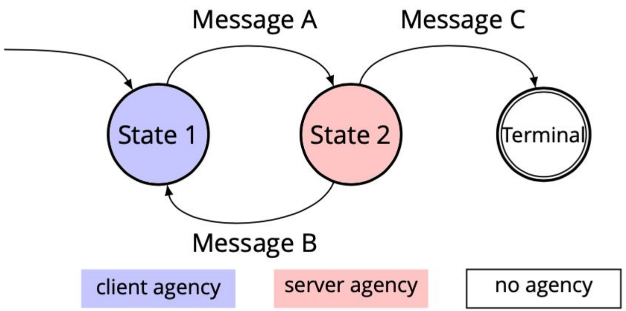
Tests are fast and reproducible

Shrink to minimal counterexample

Network Protocols: Session Types

Race Condition Too many are talking

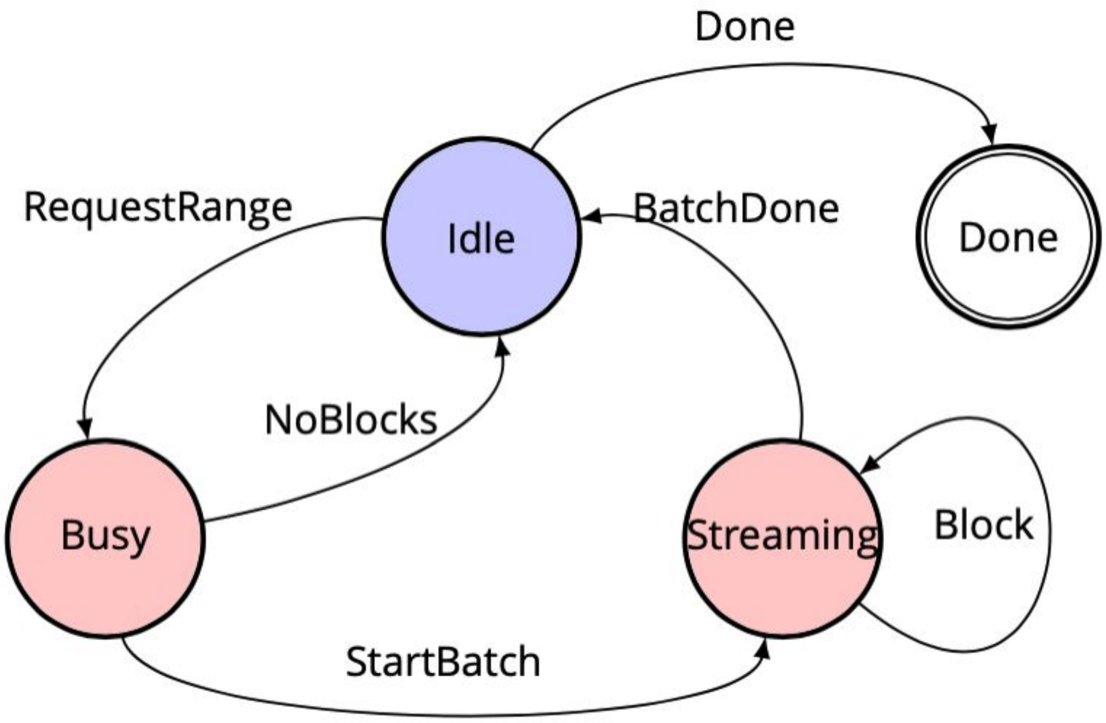
Deadlock Everyone is waiting



Typed Protocols

- Always one sender, one receiver
- Enforced by Haskell type system
  `Protocol` type class
- Violations prevented at compile time

Example: Fetching Blocks



Formal Treatment

Goal is to prove equivalence of high and low-level designs

- High level: described exactly as in the papers
- Low level:
  - a practical design;
  - matching how networks work; and
  - operate in bounded resources.
- Modular design: high and low-level related by a series of refinements
- Modular proof: prove equivalence of each refinement and compose
- Machine checked proofs

Formal Treatment

Results so far

- Formalised an asynchronous process calculus
  - Inspired by the π-calculus
  - Using Isabelle theorem prover
  - Proved all the usual properties of a process calculus
  - Framework for proving bisimilarity results
- Formalised high-level paper versions of Ouroboros BFT and Praos
- Proved equivalence of message broadcast vs relay over a network
- Proved equivalence of bulk vs incremental Ouroboros chain selection

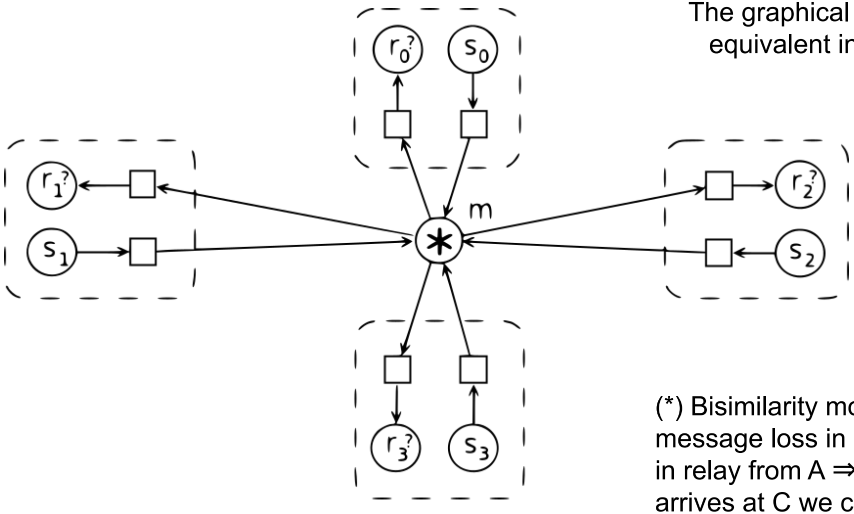
Formal Treatment

Message broadcast
(async, with loss and duplication)

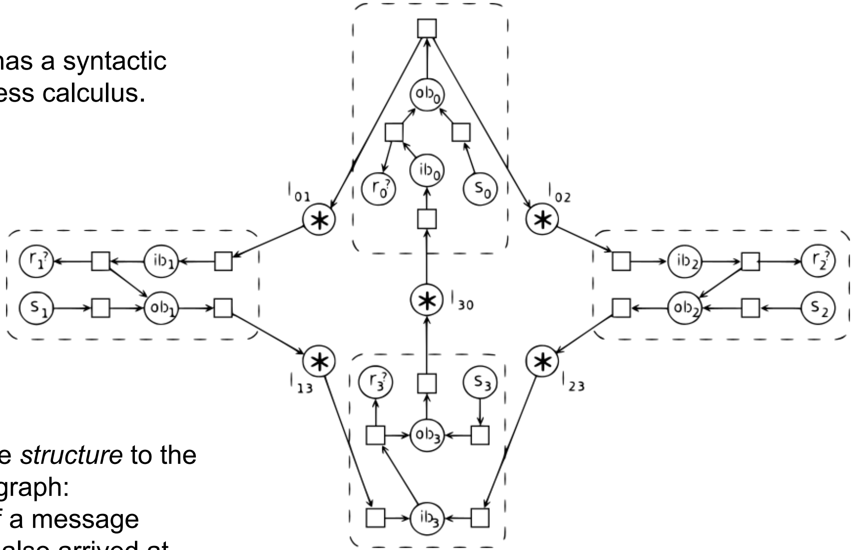
Bisimilarity(*) proved within the process calculus.

Relay over a directed graph
(async, with loss and duplication)

The graphical notation has a syntactic equivalent in the process calculus.

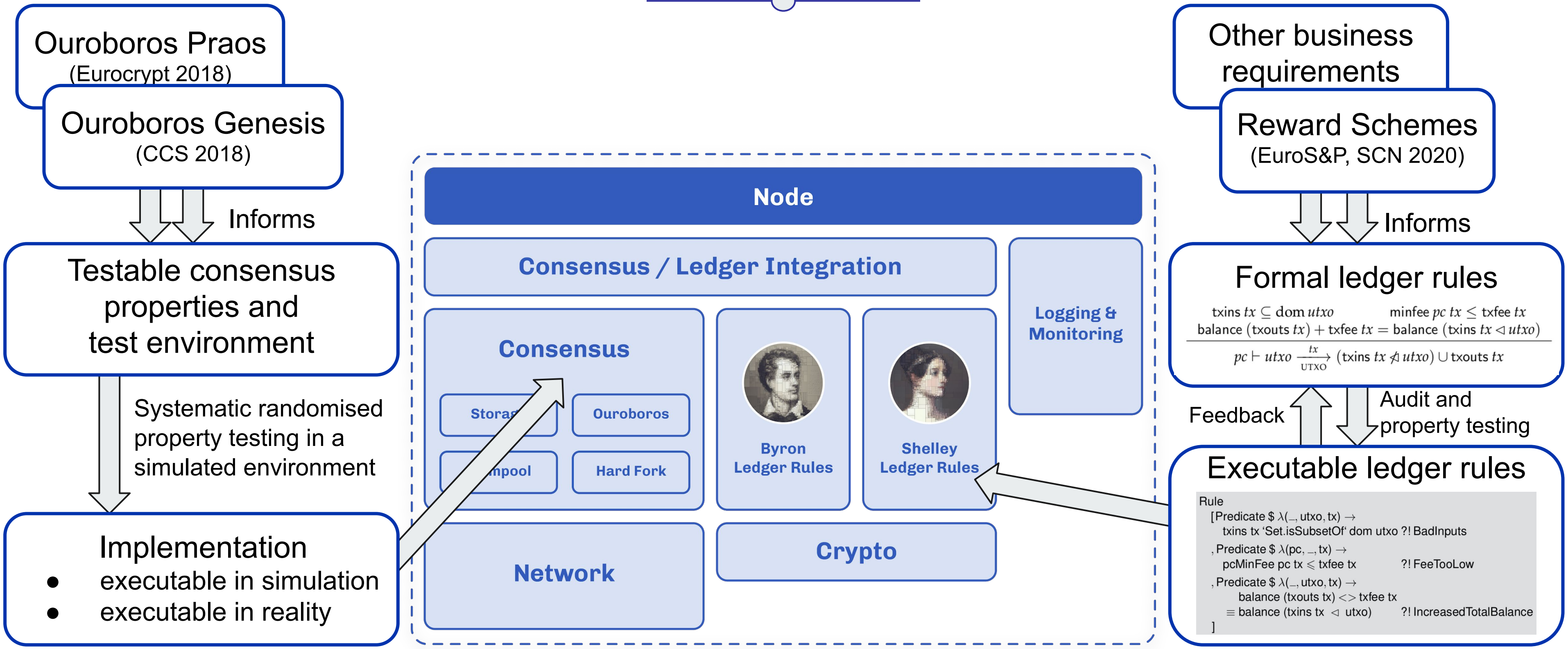


≈



(*) Bisimilarity modulo more *structure* to the message loss in the relay graph:
in relay from A ⇒ B ⇒ C, if a message arrives at C we can infer it also arrived at B, which is not true for broadcast.

Connecting theory with implementation



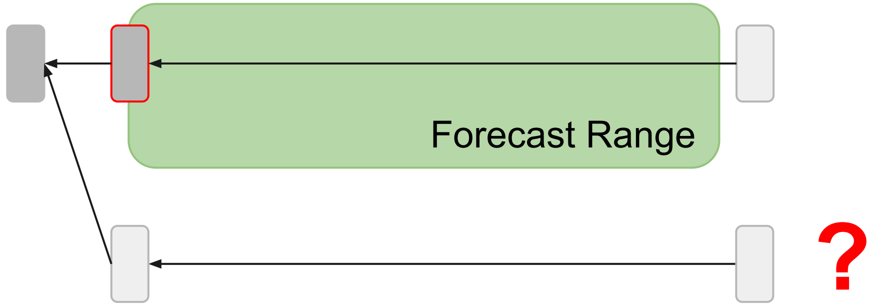
Overview

1. From Theory to Practice
2. Formal Methods and Implementation Correctness
3. **What Could Have Gone Wrong**
4. **Path to Decentralisation**

Block-Fetch, Forecasting, Denial of Service

Tension: Open Participation, Limited Resources

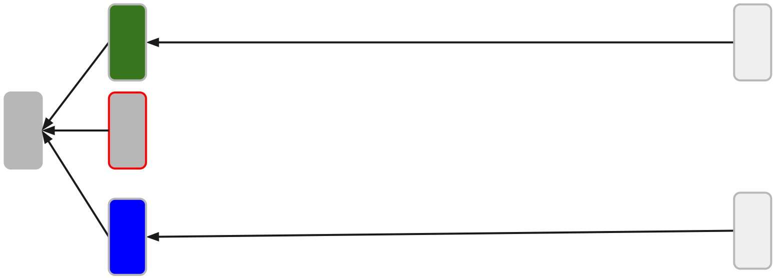
- Needs aggressive filtering
- Select chain based on headers before downloading blocks



Block-Fetch, Forecasting, Denial of Service

Edge Cases Require Block Download Before Choosing a Chain

- Proposal: only do that when there is no other way to follow any chain



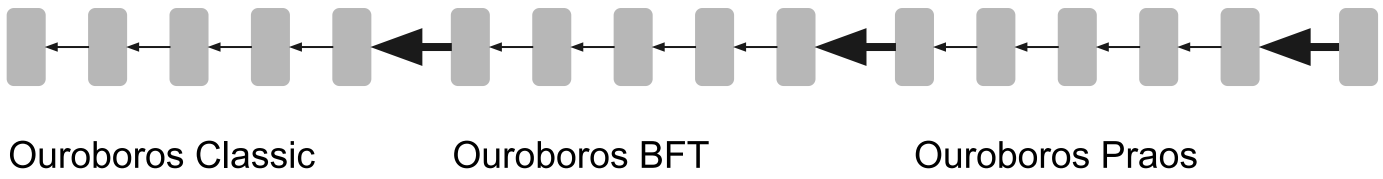
Implemented: tests show disagreement between nodes

Note: general, randomised test, not unit test

Hard-Fork Combinator

Research: One protocol, starting from scratch

Real-World System: Protocol evolves, but history is immutable



Challenges:

- One codebase needs to understand the whole chain
- Need agreement on when to transition

Hard-Fork Combinator:

- Define new protocol as sequential composition of protocols

De-Risking Decentralisation

Risks in Switching to Decentralised Protocol

- Operators need to gain experience → system not stable
- Little active stake → danger of 51% attack

Gradual Transition

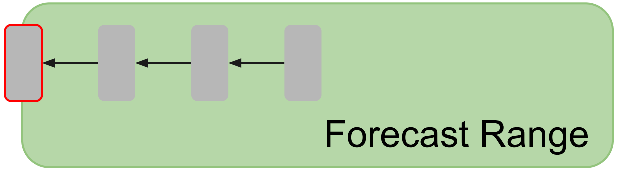
- Fraction `d` of blocks produced by fixed federation
- Gradually hand over control, from `d=1` to `d=0`

Real-World Concern: Finite Resources

Open Participation, Limited Resources: worry about DoS

- Proof of Work: producing a chain is much more difficult than validating it
  → obvious, significant advantage for honest nodes
- Proof of Stake: finer balance
  → problem: attacker can cheaply occupy honest resources
- Design needs to analyse worst-case behaviour
  excludes many off-the-shelf libraries

  e.g. aggressive filtering



Refine Carefully

Papers are Quite Abstract, Implementation Introduces Details

- Interactions of the adversary with the system also more detailed
- Avoid creating new interactions that are detrimental to the security, which are not describable at the high level
- Mind the resource balance between honest and adversarial nodes

Future Work

- Close remaining gaps between theory and implementation
- Formalised version of the security guarantees of Ouroboros
  Ph.D. thesis, using Isabelle theorem prover
- Implement newer versions of Ouroboros
- Adding additional ledger functionality
  smart contracts, decentralised software updates