INPUT | OUTPUT

# Mechanizing BFT consensus protocols in Agda

O. Melkonian, **Mauro Jaskelioff**,
J. Chapman, and J. Rossie

# Consensus Protocols

# Consensus Protocols

- Distributed fault-tolerant agreement on a certain truth

# Consensus Protocols

- Distributed fault-tolerant agreement on a certain truth
- A history that predates cryptocurrencies

# Consensus Protocols

- Distributed fault-tolerant agreement on a certain truth
- A history that predates cryptocurrencies
- Byzantine Fault Tolerance (BFT) protocols

# Consensus Protocols

- Distributed fault-tolerant agreement on a certain truth
- A history that predates cryptocurrencies
- Byzantine Fault Tolerance (BFT) protocols

  - Permissioned (fixed number of participants)
  - Finality (fast settlement).
  - Possibly corrupt players

# Main Properties

**Consistency**

**Liveness**

**Latency**

# Main Properties

| Consistency | Liveness | Latency |
| --- | --- | --- |
| If ch and ch’ are final, then ch ≼ ch' or ch' ≼ ch (there are no forks) | | |

# Main Properties

| Consistency | Liveness | Latency |
| --- | --- | --- |
| If ch and ch’ are final, then ch ≼ ch' or ch' ≼ ch (there are no forks) | Eventually, a new block will be finalized. | |

# Main Properties

| Consistency | Liveness | Latency |
| --- | --- | --- |
| If ch and ch’ are final, then ch ≼ ch' or ch' ≼ ch (there are no forks) | Eventually, a new block will be finalized. | Time in which a transaction is confirmed. |

# Formalizations at IO

- Source of Truth
- Source of Documentation
- Executable ⇒ Reference implementation

# Formalization of Consensus

Proving the Correctness of Disk Paxos in Isabelle/HOL

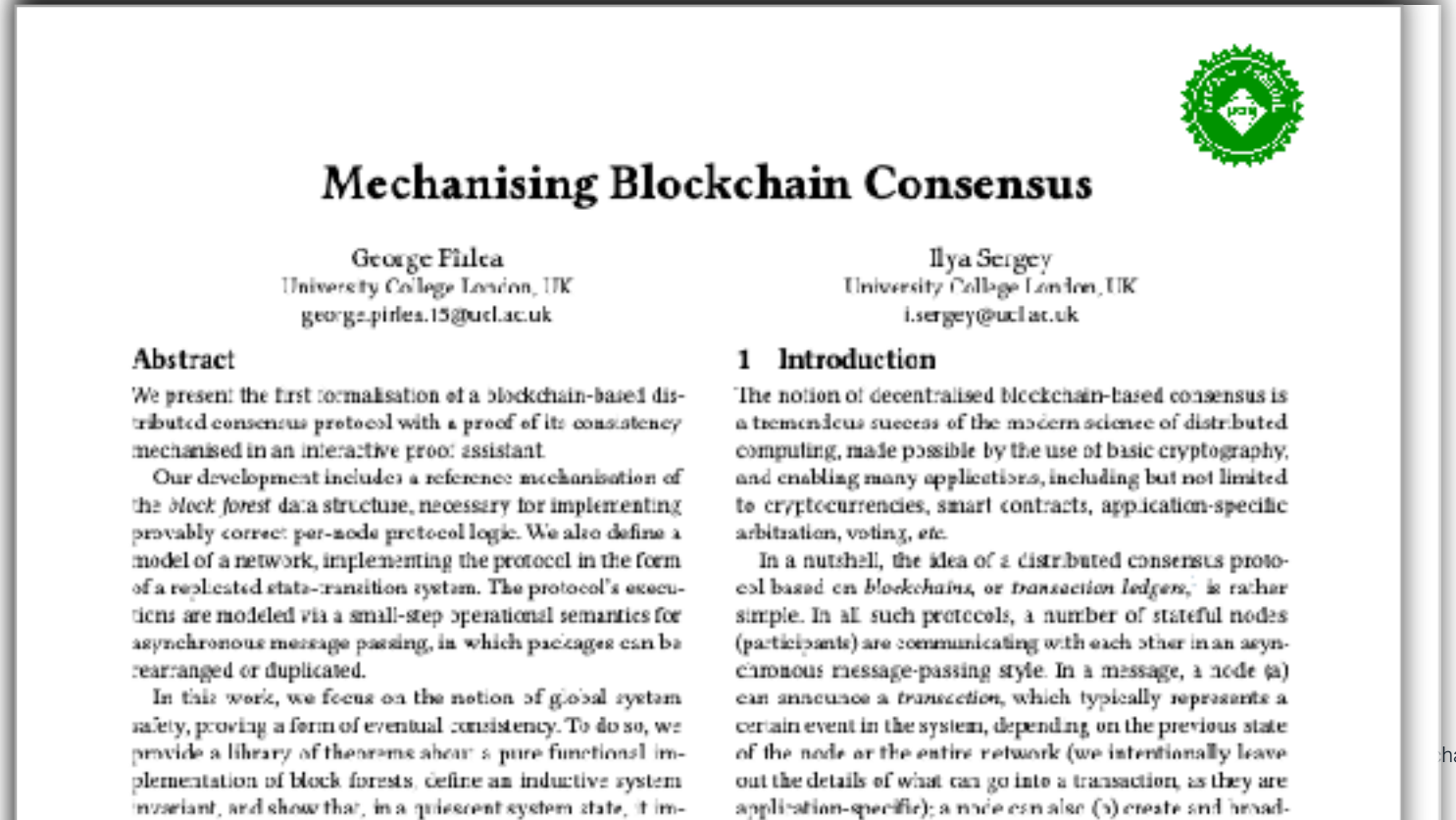
Mauro Jaskelioff Stephan Merz

October 14, 2005

**Abstract**

Disk Paxos [GL00] is an algorithm for building arbitrary fault-tolerant distributed systems. The specification of Disk Paxos has been

# Formalization of Consensus



## Mechanising Blockchain Consensus

George Pîrlea
University College London, UK
george.pirlea.15@ucl.ac.uk

Ilya Sergey
University College London, UK
i.sergey@ucl.ac.uk

### Abstract

We present the first formalisation of a blockchain-based distributed consensus protocol with a proof of its consistency mechanised in an interactive proof assistant.

Our development includes a reference mechanisation of the *block forest* data structure, necessary for implementing provably correct per-node protocol logic. We also define a model of a network, implementing the protocol in the form of a replicated state-transition system. The protocol's executions are modeled via a small-step operational semantics for asynchronous message passing, in which packages can be rearranged or duplicated.

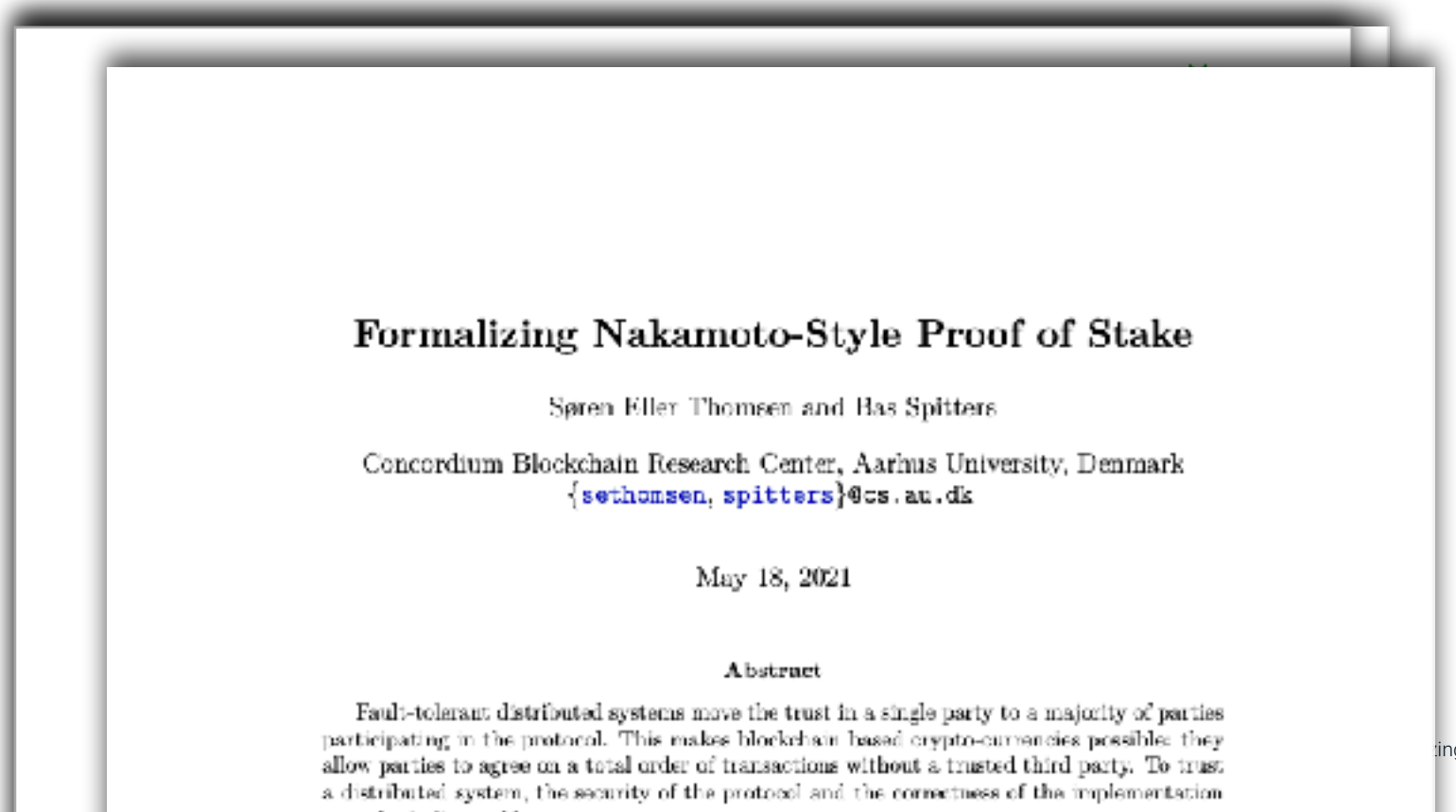
In this work, we focus on the notion of global system safety, proving a form of eventual consistency. To do so, we provide a library of theorems about a pure functional implementation of block forests, define an inductive system invariant, and show that, in a quiescent system state, it im-

### 1 Introduction

The notion of decentralised blockchain-based consensus is a tremendous success of the modern science of distributed computing, made possible by the use of basic cryptography, and enabling many applications, including but not limited to cryptocurrencies, smart contracts, application-specific arbitration, voting, *etc.*

In a nutshell, the idea of a distributed consensus protocol based on *blockchains*, or *transaction ledgers*, is rather simple. In all such protocols, a number of stateful nodes (participants) are communicating with each other in an asynchronous message-passing style. In a message, a node (a) can announce a *transaction*, which typically represents a certain event in the system, depending on the previous state of the node or the entire network (we intentionally leave out the details of what can go into a transaction, as they are application-specific); a node can also (s) create and broad-

# Formalization of Consensus



## Formalizing Nakamoto-Style Proof of Stake

Søren Eller Thomsen and Bas Spitters

Concordium Blockchain Research Center, Aarhus University, Denmark
{sethomsen, spitters}@cs.au.dk

May 18, 2021

**Abstract**

Fault-tolerant distributed systems move the trust in a single party to a majority of parties participating in the protocol. This makes blockchain based crypto-currencies possible: they allow parties to agree on a total order of transactions without a trusted third party. To trust a distributed system, the security of the protocol and the correctness of the implementation

# Formalization of Consensus

## Towards Formal Verification of HotStuff-based Byzantine Fault Tolerant Consensus in Agda*

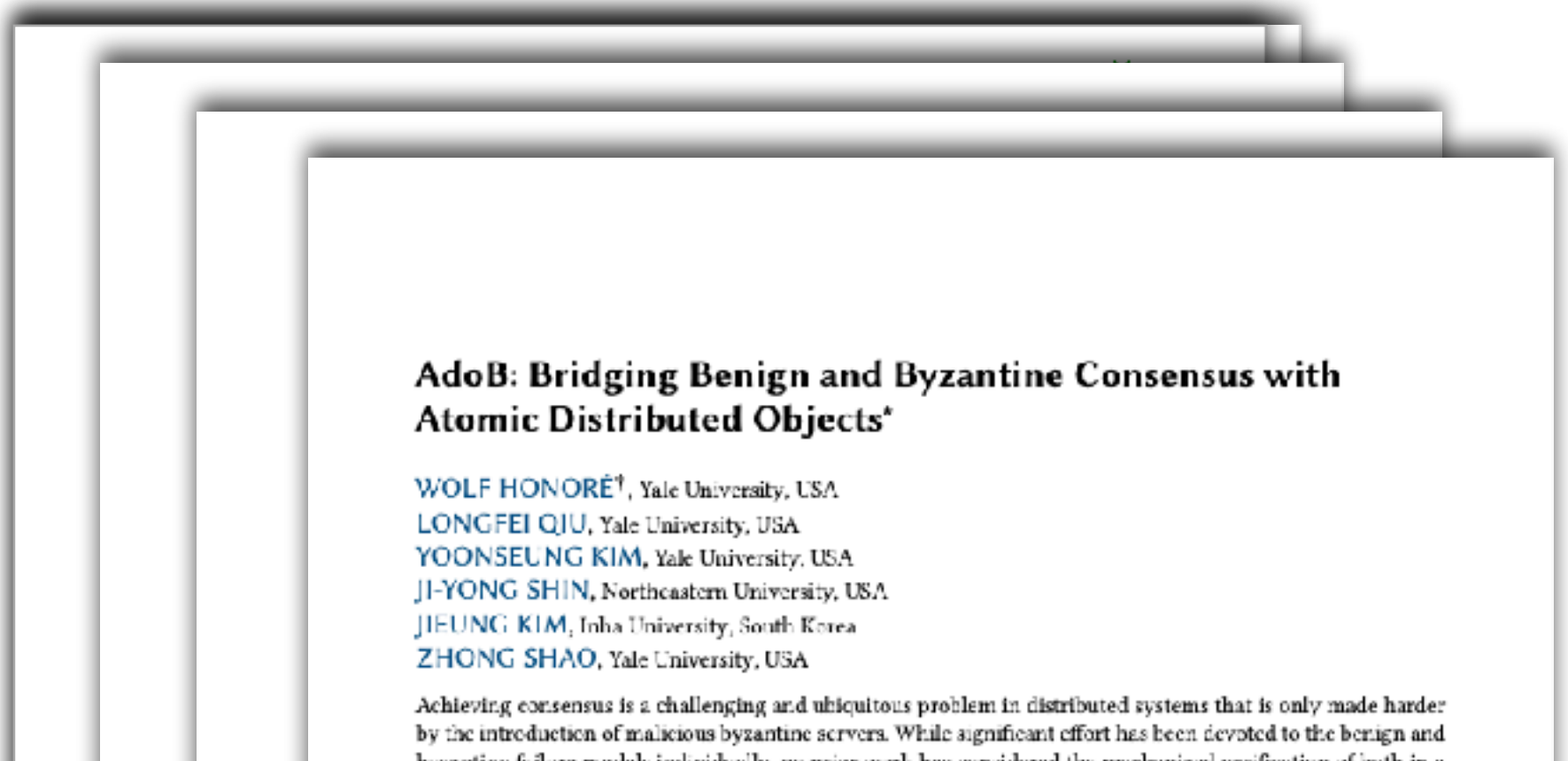
Harold Carr[1], Christopher Jenkins[2], Mark Muir[1], Victor Cacciari Miraldo[3], and Lisandra Silva[4]

[1] Oracle Labs, USA and New Zealand
[2] University of Iowa, USA
[3] Tweag, The Netherlands
[4] Runtime Verification, USA

**Abstract.** LibraBFT is a Byzantine Fault Tolerant (BFT) consensus protocol based on HotStuff. We present an abstract model of the protocol underlying HotStuff / LibraBFT and formal, machine-checked proofs of their core correctness (safety) property and an extended condition that enables non-participating parties to verify committed results. (Liveness properties would be proved for specific implementations, not for the abstract model presented in this paper.)

A key contribution is precisely defining assumptions about the behav-

# Formalization of Consensus

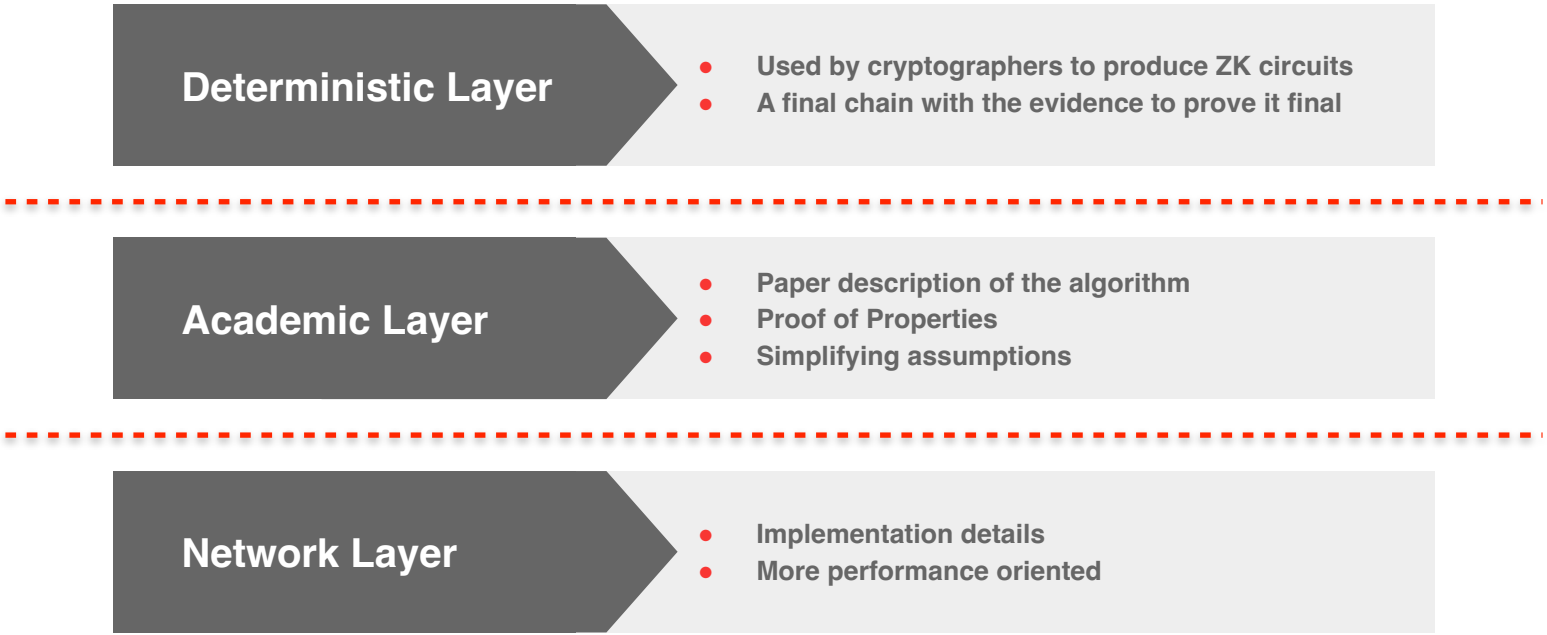


AdoB: Bridging Benign and Byzantine Consensus with Atomic Distributed Objects*

WOLF HONORÉ†, Yale University, USA
LONGFEI QIU, Yale University, USA
YOONSEUNG KIM, Yale University, USA
JI-YONG SHIN, Northeastern University, USA
JIEUNG KIM, Inha University, South Korea
ZHONG SHAO, Yale University, USA

Achieving consensus is a challenging and ubiquitous problem in distributed systems that is only made harder by the introduction of malicious byzantine servers. While significant effort has been devoted to the benign and

# Layers of Formalization in Agda



# Streamlet

- Simple BFT Protocol
- Good for initial formalization of academic layer

# Streamlet: Basic Structures

```
Epoch = ℕ
```

```
epochLeader : Epoch → Pid
```

# Streamlet: Basic Structures

```
Epoch = ℕ
```

```
epochLeader : Epoch → Pid
```

```
record Block : Type where
  constructor ⟨_,_,_⟩
  field parentHash : Hash
        epoch      : Epoch
        payload    : List Transaction
```

```
Chain = List Block
```

# Streamlet: Basic Structures

```
Epoch = ℕ
```

```
epochLeader : Epoch → Pid
```

```
record Block : Type where
  constructor ⟨_,_,_⟩
  field parentHash : Hash
        epoch      : Epoch
        payload    : List Transaction
```

```
Chain = List Block
```

```
data Message : Type where
  Propose : SignedBlock → Message
  Vote    : SignedBlock → Message
```

# Streamlet Notarization and Finalization

Assume a local database of messages `ms : List Message`

```
record NotarizedBlock (b : Block) : Type where
  votes : List Message
  votes = filter ((_≟ b) ∘ blockMessage) ms

  field enoughVotes : IsMajority votes

NotarizedChain : Chain → Type
NotarizedChain = All NotarizedBlock
```

# Streamlet Notarization and Finalization

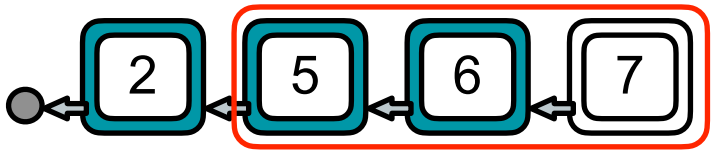
Assume a local database of messages `ms : List Message`

```
record NotarizedBlock (b : Block) : Type where
  votes : List Message
  votes = filter ((_≟ b) ∘ blockMessage) ms

  field enoughVotes : IsMajority votes

NotarizedChain : Chain → Type
NotarizedChain = All NotarizedBlock
```

```
data FinalizedChain : Block → Chain → Type
  where
    Finalize : ∀ {ch b₁ b₂ b₃} →
      • NotarizedChain (b₃ :: b₂ :: b₁ :: ch)
      • b₃ .epoch ≡ suc (b₂ .epoch)
      • b₂ .epoch ≡ suc (b₁ .epoch)
      ──────────────────────────────
      FinalizedChain b₃ (b₂ :: b₁ :: ch)
```



# Global Transition

```
data _—→_ (s : GlobalState) : GlobalState → Type
  where

  LocalStep : let ls = s @ p in
    p ▷ s .e-now ⊢ ls —[ mm ]→ ls′
    ──────────────────────────────────────
    s —→ broadcast p mm (updateLocal p ls′ s)

  Deliver :
    (envϵ : env ϵ s .networkBuffer) →
    ──────────────────────────────────
    s —→ deliverMsg s envϵ

  AdvanceEpoch :
    ──────────────────
    s —→ advanceEpoch s
```

```
record GlobalState : Type where
    field
      e-now         : Epoch

      stateMap      : Vec LocalState nodes

      networkBuffer : List Envelope
```

# Local Transition

```
data _▷_⊢_–[_]→_
  (p : Pid) (e : Epoch) (ls : LocalState) :
      Maybe Message → LocalState → Type where

  ProposeBlock :
    let b = ⟨ ch ♯ , e , txs ⟩
        m = Propose (signBlock p b)
        ls' = record ls { phase = Voted
                        ; db    = m :: ls .db }
    in • ls .phase ≡ Ready
       • p ≡ epochLeader e
       • ch –longest-notarized-chain-∈– ls .db
       • b –connects-to– ch
       ──────────────────────────────────────
         p ▷ e ⊢ ls –[ just m ]→ ls'
```

```
data Phase : Type where
  Ready Voted : Phase

record LocalState : Type where
  field phase : Phase
        db    : List Message
        inbox : List Message
        final : Chain
```

# Streamlet Local Transition

```
data _▷_⊢_–[_]→_
  (p : Pid) (e : Epoch) (ls : LocalState) :
    Maybe Message → LocalState → Type where

  ProposeBlock :
    let b = ⟨ ch ♯ , e , txs ⟩
        m = Propose $ signBlock p b
        ls' = record ls { phase = Voted
                        ; db = m :: ls .db }
    in
    • ls .phase ≡ Ready
    • p ≡  epochLeader e
    • ch longest-notarized-chain-∈ ls .db
    • b –connects-to– ch
    ──────────────────────────────────
    p ▷ e ⊢ ls –[ just m ]→ ls'

  FinalizeBlock : ∀ ch b →
    • ValidChain (b :: ch)
    • FinalizedChain (ls .db) ch b
    ──────────────────────────────
    p ▷ e ⊢ ls –[ nothing ]→ finalize ch ls
```

```
  VoteBlock :
    let L  = epochLeader e
        b  = ⟨ H , e , txs ⟩
      mᴾ = Propose $ signBlock L b
      m  = Vote    $ signBlock p b
    in
    ∀ (m∈ : AnyFirst (_≡ mᴾ) (ls .inbox)) →
    • ls .phase ≡ Ready
    • p ≢ L
    • ch longest-notarized-chain-∈ ls .db
    • b –connects-to– ch
    ──────────────────────────────────
    p ▷ e ⊢ ls –[ just m ]→ voteBlock ls m∈ m

  RegisterVote : let m = Vote sb in
      • m ∉ ls .db
    → (m∈ : m ∈ ls .inbox) →
    ──────────────────────────────
    p ▷ e ⊢ ls –[ nothing ]→ registerVote ls m∈
```

# Simplex Local Transition

```
data _⊢_—_s→_
  (p : Pid)(t : Time)(ls : LocalState) :
  List Message → LocalState → Type where
  ProposeBlock :
    let
      h = ls .height
      L = slotLeader h
      ph = ch ♯
      b = inj₁ ⟨ h , ph , trxs ⟩b
      m = Propose p b nc
    in
      (ncϵ : ch chain-notarized-ϵ
                (ls .db))
    → ls .phase ≡ Start
    • p ≡ L
    • nc ≡ get-notarized-chain ncϵ
    • ValidChain (b :: ch)
    ──────────────────────────────
      p ⊢ t ⊢ ls —[ m ]s→ startLS t ls

  InitNonLeader : (let h = ls .height) →
    • p ≢ slotLeader h
    • ls .phase ≡ Start
    ──────────────────────
      p ⊢ t ⊢ ls —s→ startLS t ls
```

```
VoteBlock :
    let
      h  = ls .height
      L  = slotLeader h
      ph = ch ♯
      rb =  ⟨ h , ph , trxs ⟩b
      b  = inj₁ rb
      mᴾ = Propose L (inj₁ rb) nc
      m  = Vote p b
    in
      (mϵ : AnyFirst (_≡ mᴾ)
(ls .inbox))
    → (ls .phase ≡ Vote-collecting)
    • ValidChain (b :: ch)
      ──────────────────────────────
      p ⊢ t ⊢ ls —[ m ]s→ voteProp ls mϵ

  RegisterMessage :
      (m ∈ (Vote p b ::
            Finalize p h ::
            View p nc :: []))
    → (mϵ : m ∈ ls .inbox)
    ──────────────────────
      p ⊢ t ⊢ ls —s→ regMessage ls mϵ
```

```
SeeNotarizedChain :
  let h  = ls .height
      vm = [ View p nc ]
      m = if ls. timer-fired?
           then vm
           else (Finalize p h) :: vm
  in
  • NotarizedChain ch
  • ValidChain ch
  • heightChain ch ≡ h
  • ch chain-notarized-ϵ (ls .db)
    ──────────────────────────────
    p ⊢ t ⊢ ls — m s→ next-iter ls

Finalize : let ms = ls .db in
    (h : Height)
  → h height-finalized-ϵ ms
  → (chϵ : ch chain-notarized-ϵ ms)
  → heightChain ch ≡ h
  → h > heightChain (ls .finalchain)
  ──────────────────────────────
    p ⊢ t ⊢ ls —s→ finalize ch ls
```

# Traces

- Decidability of rules premises yields execution traces

```
s11
  →⟨ Propose? 𝕃 [ b6 ; b5 ; b2 ] [] ⟩
s12
  →⟨ Deliver? [ 𝔸 | p7 ⟩ ⟩
s13
  →⟨ Vote? 𝔸 [ b6 ; b5 ; b2 ] [] ⟩
s14
  →⟨ Finalize? 𝔸 [ b6 ; b5 ; b2 ] b7 ⟩
s15
```

# Current State of Formalization

- Formalization of Deterministic Layer
  - Notarization
  - Change of committee
- Formalization of Academic Layer
  - Streamlet
    - Relational Spec
    - Execution Traces
    - Safety (WIP)
  - Simplex
    - Relational Spec