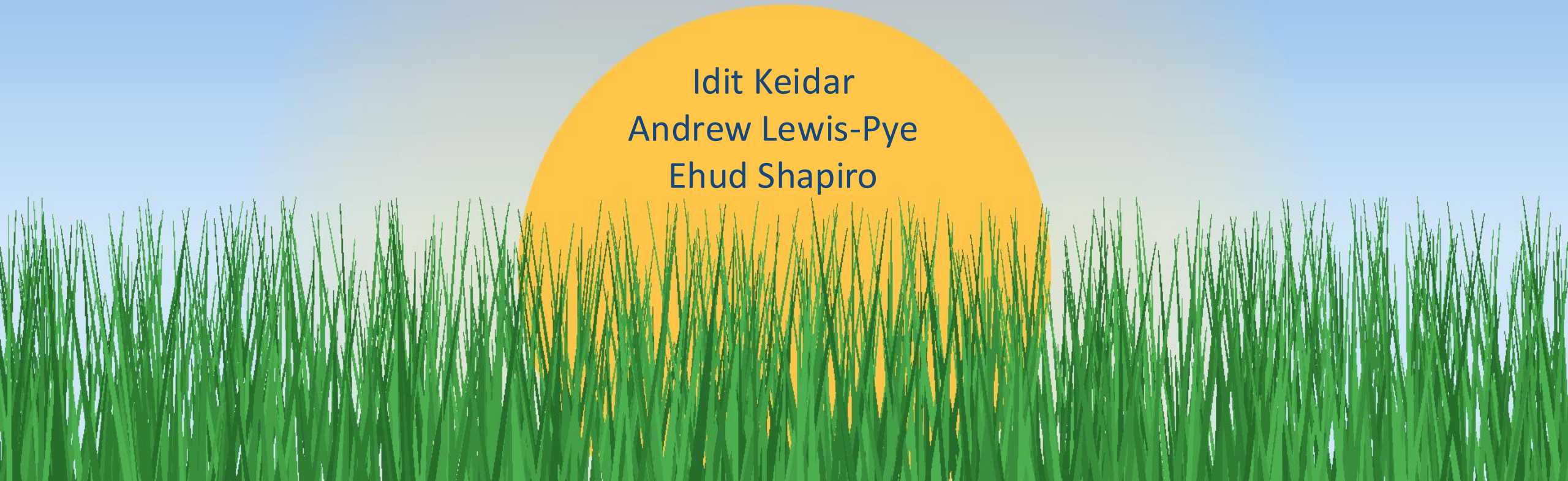
Grassroots Tutorial

Idit Keidar
Andrew Lewis-Pye
Ehud Shapiro



Tutorial Plan

1. Introduction: Grassroots & AI (Udi, 20 min)
2. Bellow consensus: Volitional agents, grassroots social networks and currencies (Andy, 40 min)
   --- *30 min coffee break* ---
3. Constitutional consensus for democratic governance (Idit, 40 min)
4. Wrap up: How can you help? (Udi, 20 min)

Each slot will have time for questions.

Why Grassroots?

- **1990's Dream:** the Internet will be the great equaliser
- **2020's Reality:** it is the dominant source of inequality
- **Cause:** "Principles of Distributed Computing" of the Internet:
  1. **Centralised:** Facebook/X/Amazon/Uber/... **Autocratic**
  2. **Decentralised:** Bitcoin/Ethereum/... **Plutocratic**
  3. **Egalitarian? Democratic?** Yok/Nada/None… [Scuttlebutt]
- **Grassroots:** An egalitarian and democratic alternative

So what is Grassroots?

Many answers…

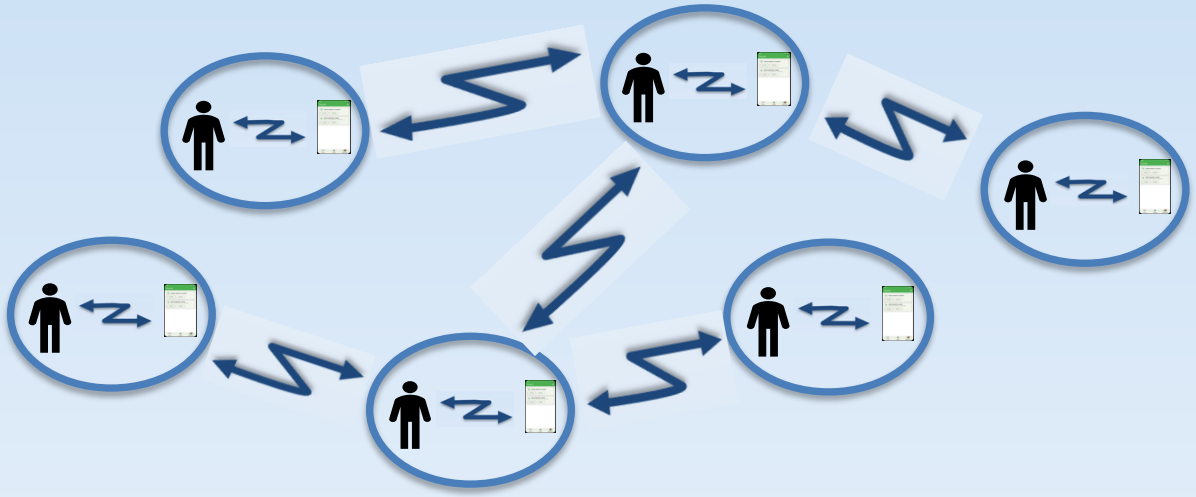
1. **Informal** property of distributed systems
2. **Comparison** with global platforms
3. **Formal** property of a formal protocol (Part 2)

1. Grassroots, informally

“Hardware” platform: people and their smartphones

- Excludes globally-named servers

An Agent



Multi-Agent Grassroots System

1. Grassroots, informally

“Hardware” platform: people and their smartphones

- Excludes globally-named servers

Software platform: (social network, currencies, consensus, etc.)

1. Can have **multiple independent instances**
   - Excludes all known centralised (one Facebook) and decentralised (one Bitcoin, DHT, IPFS) systems (but not Scuttlebutt)
2. Instances may **interact and coalesce**
   - Excludes permissioned consensus, independent DHT’s, etc.

2. Grassroots, by comparison

… of cardinality of essential agents:

How many agents need to be eliminated to disable communication?

Class	Essential Agents	Cardinality	Canonical Example
Centralised	The server	One	Facebook
Decentralised	Bootstrap nodes	Finite, >1	Bitcoin
Federated	All servers, or all clients	Infinite, but not universal	Mastodon
Grassroots	All (but one ☺)	Universal	Scuttlebutt

3. Grassroots, formally

[In the second part of the tutorial]

The Grassroots Vision

1. **Grassroots upgrades of** Facebook, X, Bitcoin, Ethereum, Uber, Airbnb, Deliveroo, Amazon Marketplace,…
2. **Novel grassroots platforms for** Child-Safe Social Networking, Digital Cooperatives, Civil Movements, Political Parties, Democratic Communities from local to global.

We already have grassroots specs for many of these… but

How to square the circle? Get funding and recruit the best programmers to compete with $100Ms budgets and 100Ks programmers?

Realising the Grassroots Vision with AI

Grassroots Bonds as a Foundation for Market Liquidity

Anonymous author

Anonymous affiliation

Abstract

Global cryptocurrencies are unbacked and have high transaction cost incurred by global consensus. Grassroots cryptocurrencies are backed by the goods and services of their issuers—any person, natural or legal—and have no transaction cost beyond operating a smartphone. Liquidity in grassroots cryptocurrencies arises from mutual credit via coin exchange among issuers. However, as grassroots coins are redeemable 1-for-1 against any other grassroots coin, the credit-forming exchange must also be 1-for-1, lest prompt redemption after exchange would leave the parties with undue profit or loss. Thus, grassroots coins are incongruent with liquidity through interest-bearing credit.

Here, we introduce grassroots bonds, which extend grassroots coins with a maturity date. Bond redemption generalises coin redemption, allowing the lending of liquid coins in exchange for interest-bearing future-maturity bonds. We show that the voluntary swap of grassroots bonds can realise the full gamut of financial instruments, including loans, sale of debt, forward contracts, options, collateral, guarantees, insurance, credit default swaps, letters of credit, and credit lines. Moreover, classical liquidity ratios are applicable just as well to grassroots bonds. Grassroots bonds may thus allow local digital economies to form and grow without initial capital or external credit, harnessing mutual trust within communities into liquidity.

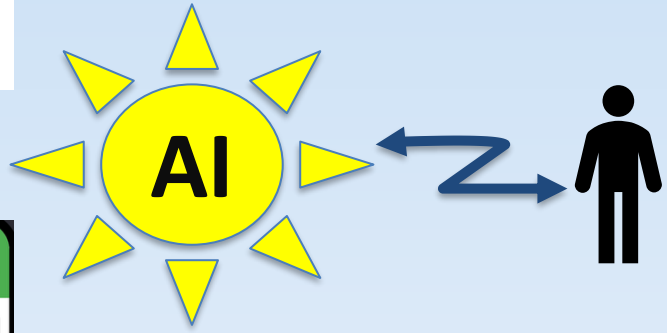
Grassroots bonds are formally specified via guarded multiagent atomic transactions and implemented in GLP, a concurrent logic programming language running on Dart for smartphone deployment. The implementation is illustrated by a running multiagent village market scenario in GLP. A secure implementation, recovering bond issuers and holders from loss of their signing key or device, is developed in a companion paper.

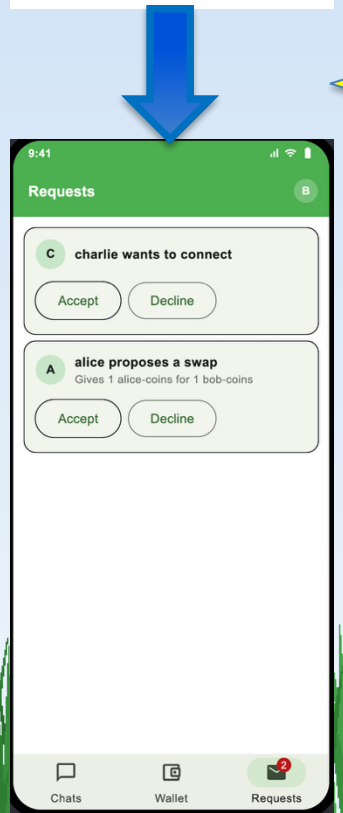
2012 ACM Subject Classification Applied computing → Digital cash; Theory of computation → Distributed computing models; Computer systems organization → Peer-to-peer architectures

Keywords and phrases Grassroots Bonds, Grassroots Cryptocurrencies, Financial Instruments, Liquidity, Guarded Multiagent Atomic Transactions, GLP

Digital Object Identifier 10.4230/LIPIcs...0

Scientific paper describing a grassroots platform





Smartphone App realizing the grassroots platform

Programming & AI: A Personal History

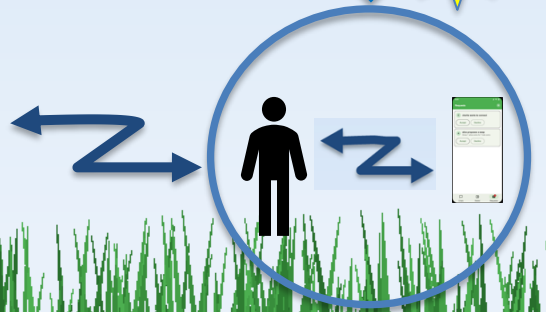
- **1971 FORTRAN in high school**: compile-run cycle — 1 week
- **1977 Cobol in a bank:** compile-run cycle — 24 hours
- **1980 Prolog at Yale:** interactive development of AI — machine learning/program synthesis/algorithmic debugging (IJCAI'81)
- **1982-1995 Concurrent Prolog:** Developed language and social networking app (Virtual Places)

Today: Implementing Grassroots with AI

Scientific paper describing a grassroots platform

GLP program realizing the grassroots platform

Dart/Flutter implementation of GLP



Grassroots App on iPhone/Android phones

End of Introduction

Questions?

Volitional Agents and Grassroots Protocols

Structure

1) Review: transition systems
2) Multi-agent transition systems (MATS)
3) Volitional states, volitional transactions, Volitional MATS.
4) Grassroots protocols.
5) Volitional Transactions-based protocols.
6) Example: Child-safe networks.

Transition systems

A **transition system** (S, T, L) is specified by:

- A set of **states** S (with a special designated *initial state*).
- A set of correct **transitions** $T \subseteq S^2$. Each $t \in T$ is a pair (s, s'), where $s \neq s'$.
- A set L of disjoint subsets of T. Each element of L can be thought of as a liveness class...

Transition systems

A **transition system** (S, T, L) is specified by:

- A set of **states** S (with a special designated *initial state*).
- A set of correct **transitions** $T \subseteq S^2$. Each $t \in T$ is a pair (s, s'), where $s \neq s'$.
- A set L of disjoint subsets of T. Each element of L can be thought of as a liveness class...

We'll see how L is used to specify liveness in a bit...

Transition systems

A **transition system** (S, T, L) is specified by:

- A set of **states** S (with a special designated *initial state*).
- A set of correct **transitions** $T \subseteq S^2$. Each $t \in T$ is a pair (s, s'), where $s \neq s'$.
- A set L of disjoint subsets of T. Each element of L can be thought of as a liveness class...

A **run** is a sequence of states in S, beginning with the initial state. A run is **safe** if every consecutive pair of states is a correct transition in T.

A **transition system** (S, T, L) is specified by:

- A set of **states** S (with a special designated *initial state*).
- A set of correct **transitions** $T \subseteq S^2$. Each $t \in T$ is a pair (s, s'), where $s \neq s'$.
- A set L of disjoint subsets of T. Each element of L can be thought of as a liveness class...

A set of transitions $M \in L$ is **enabled** at $s \in S$ if there exists some s' with $(s, s') \in M$, i.e., if it is now possible to make a transition in M.

A run r is **live** if no $M \in L$ is enabled in every state of a suffix of r without any transition in M occurring in the suffix.

A **transition system** (S, T, L) is specified by:

- A set of **states** S (with a special designated *initial state*).
- A set of correct **transitions** $T \subseteq S^2$. Each $t \in T$ is a pair (s, s'), where $s \neq s'$.
- A set L of disjoint subsets of T. Each element of L can be thought of as a liveness class...

A set of transitions $M \in L$ is **enabled** at $s \in S$ if there exists some s' with $(s, s') \in M$, i.e., if it is now possible to make a transition in M.

A run r is **live** if no $M \in L$ is enabled in every state of a suffix of r without any transition in M occurring in the suffix.

A run is **correct** if it is safe and live.

Multiagent Transition Systems

Given a set of **agents** P and a set of *underlying* states S, a **configuration** is an element of S^P, i.e., a configuration assigns a state in S to each agent in P.

The states of the MATS are configurations, and T is now contains pairs of configurations.

An Aside

Today we won't focus on faulty behaviour: we are focussed on high level specifications that determine correct behaviour.

Resilience to faults can be studied using the notion of **implementations**:

- For transition systems $TS = (S, T, L)$ and $TS' = (S', T', L')$, an implementation of TS by TS' is a function mapping states of S' to states of S.

An Aside

Today we won't focus on faulty behaviour: we are focussed on high level specifications that determine correct behaviour.

Resilience to faults can be studied using the notion of **implementations**:

- For transition systems $TS = (S, T, L)$ and $TS' = (S', T', L')$, an implementation of TS by TS' is a function mapping states of S' to states of S.
- An implementation is *correct* if it maps correct runs of TS' to correct runs of TS.

An Aside

Today we won't focus on faulty behaviour: we are focussed on high level specifications that determine correct behaviour.

Resilience to faults can be studied using the notion of **implementations**:

- For transition systems $TS = (S, T, L)$ and $TS' = (S', T', L')$, an implementation of TS by TS' is a function mapping states of S' to states of S.
- An implementation is *correct* if it maps correct runs of TS' to correct runs of TS.
- It is *complete* if every correct run of TS is mapped to by some run of TS'.

An Aside

Today we won't focus on faulty behaviour: we are focussed on high level specifications that determine correct behaviour.

Resilience to faults can be studied using the notion of **implementations**:

- For transition systems $TS = (S, T, L)$ and $TS' = (S', T', L')$, an implementation of TS by TS' is a function mapping states of S' to states of S.
- An implementation is *correct* if it maps correct runs of TS' to correct runs of TS.
- It is *complete* if every correct run of TS is mapped to by some run of TS'.

Given a high level specification TS, one can then consider lower level implementations, together with notions of fault tolerance for those implementations.

Volitions

Next, we want to introduce Volitional MATS.

First we have to introduce volitional states, and certain types of volitional transitions...

Volitions

Next, we want to introduce **volitions**: these are not just choices made at a point in time, but are first class objects in the model, reflected in persistent state.

The basic idea is that we separate an agent's state into two parts: their **machine state** and their **volitions**...

Agent state

Machine state	Volitional state
$s \in S$	A set of transactions they are willing to engage in

Volitions

To formalise this, we distinguish between transitions and **transactions**: a transaction just specifies the change in state for some subset $Q \subseteq P$.

Volitions

To formalise this, we distinguish between transitions and **transactions**: a transaction just specifies the change in state for some subset $Q \subseteq P$.

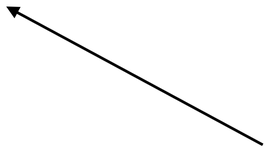
Guarded Machine Transactions. Given an arbitrary set S of *machine states* and agents $Q \subseteq P$, a **machine configuration** over Q is a member of S^Q, and a **machine transaction** over *participants* Q is a pair $(c, c') \in (S^Q)^2$ such that $c \neq c'$.

Volitions

To formalise this, we distinguish between transitions and **transactions**: a transaction just specifies the change in state for some subset $Q \subseteq P$.

Guarded Machine Transactions. Given an arbitrary set S of *machine states* and agents $Q \subseteq P$, a **machine configuration** over Q is a member of S^Q, and a **machine transaction** over *participants* Q is a pair $(c, c') \in (S^Q)^2$ such that $c \neq c'$.

Given such a machine transaction t, a **guarded transaction** over t is a pair (t, Q') where $Q' \subseteq Q$ are its **guards**.



The agents that must be willing for the transaction to take place.

Volitions

To formalise this, we distinguish between transitions and **transactions**: a transaction just specifies the change in state for some subset $Q \subseteq P$.

Guarded Machine Transactions. Given an arbitrary set S of *machine states* and agents $Q \subseteq P$, a **machine configuration** over Q is a member of S^Q, and a **machine transaction** over *participants* Q is a pair $(c, c') \in (S^Q)^2$ such that $c \neq c'$.

Given such a machine transaction t, a **guarded transaction** over t is a pair (t, Q') where $Q' \subseteq Q$ are its **guards**.

> When we say a transaction is "guarded by $\{p, q\}$," both must be willing; when we say it is "guarded by either p or q," we mean there are two guarded transactions over the same machine transaction, $(t, \{p\})$ and $(t, \{q\})$, so that either person's volition suffices.

Volitions

Next, we want to formally define volitional states and correct volitional transitions (basically those where the volitional state of the guards is as required)...

...but there is a slight complication that we really have to consider equivalence classes of transactions here...

Volitions

Distinct machine transactions can represent "the same action" in different configurations; we capture this with an equivalence relation on machine transactions.

Transaction Equivalence. Given a set of machine transactions T, a **transaction equivalence** is an equivalence relation $\sim$ on T such that $t \sim t'$ implies t and t' have the same participants. We write $[t]$ for the equivalence class of t under $\sim$.

Volitions

Distinct machine transactions can represent "the same action" in different configurations; we capture this with an equivalence relation on machine transactions.

Transaction Equivalence. Given a set of machine transactions T, a **transaction equivalence** is an equivalence relation $\sim$ on T such that $t \sim t'$ implies t and t' have the same participants. We write $[t]$ for the equivalence class of t under $\sim$.

> For example, a running example in this talk will be Child-Safe Social Networks. Here, we consider an evolving network of contacts, and the machine state of an agent is its present contacts. In this context, befriending another agent might be thought of as a single transaction, but is really an equivalence class of transactions (one for each state before the befriending).

Volitions

Distinct machine transactions can represent "the same action" in different configurations; we capture this with an equivalence relation on machine transactions.

Transaction Equivalence. Given a set of machine transactions T, a **transaction equivalence** is an equivalence relation $\sim$ on T such that $t \sim t'$ implies t and t' have the same participants. We write $[t]$ for the equivalence class of t under $\sim$.

Now we can formally define volitional states and correct volitional transitions...

Volitions

Given agents P, machine states S, a set of machine transactions T, and equivalence $\sim$ on T:

- An **agent state** is a pair $(V, m) \in \mathcal{A} = (2^{T/\sim} \times S)$ where V is its **volitional state** and $m \in S$ its **machine state**.
- An **agent configuration** c is a member $c \in \mathcal{A}^P$.

The volitional state is a set of (equivalence classes of) transactions the agent is willing to execute

Volitions

Given agents P, machine states S, a set of machine transactions T, and equivalence $\sim$ on T:

- An **agent state** is a pair $(V, m) \in \mathcal{A} = (2^{T/\sim} \times S)$ where V is its **volitional state** and $m \in S$ its **machine state**.
- An **agent configuration** c is a member $c \in \mathcal{A}^P$.

We let c_p^m denote the machine state of agent p in configuration c, while c_p^v denotes their volitional state in c.

Volitions

Volitional transactions are of two kinds:

- **Volition changes**: a single agent changes their volitional state.

Volitions

Volitional transactions are of two kinds:

- **Volition changes**: a single agent changes their volitional state.
- A **volitional machine transaction** induced by a guarded machine transaction (t, Q') for some $t = (d, d')$ with participants Q is a pair (c, c') where:
  - $c \neq c'$ are agent configurations such that $[t] \in c_q^v$ for every $q \in Q'$;

Every guard wills the transaction.

Volitions

Volitional transactions are of two kinds:

- **Volition changes**: a single agent changes their volitional state.
- A **volitional machine transaction** induced by a guarded machine transaction (t, Q') for some $t = (d, d')$ with participants Q is a pair (c, c') where:
  - $c \neq c'$ are agent configurations such that $[t] \in c^v_q$ for every $q \in Q'$;
  - $c^m_p = d_p$ and $c'^m_p = d'_p$ for every $p \in Q$, and;

The machine transaction is that specified by t.

Volitions

Volitional transactions are of two kinds:

- **Volition changes**: a single agent changes their volitional state.
- A **volitional machine transaction** induced by a guarded machine transaction (t, Q') for some $t = (d, d')$ with participants Q is a pair (c, c') where:
  - $c \neq c'$ are agent configurations such that $[t] \in c_q^v$ for every $q \in Q'$;
  - $c_p^m = d_p$ and $c'^m_p = d'_p$ for every $p \in Q$, and;
  - $c'^v_p = c_p^v \setminus \{[t]\}$ for every $p \in Q$.

Once the transaction is executed, we remove it from the volitional state.

Example

A **child-safe social network** is a grassroots social network in which a child's partaking in online activities is subject to parental consent.

Each agent p maintains a set of friends $c_p^m \subseteq P$ (their machine state). In what follows, r, s are children with respective parents p, q. We specify the protocol using two types of guarded machine transaction.

Example

A **child-safe social network** is a grassroots social network in which a child's partaking in online activities is subject to parental consent.

Each agent p maintains a set of friends $c_p^m \subseteq P$ (their machine state). In what follows, r, s are children with respective parents p, q. We specify the protocol using two types of guarded machine transaction.

(1) **Child befriend**: $(c')_r^m := c_r^m \cup \{s\}$, $(c')_s^m := c_s^m \cup \{r\}$, provided $q \in c_p^m$ and $s \notin c_r^m$. Guarded by $\{r, s, p, q\}$.
(2) **Child unfriend**: $(c')_r^m := c_r^m \setminus \{s\}$, $(c')_s^m := c_s^m \setminus \{r\}$, provided $s \in c_r^m$. Guarded by any one of $\{r, s, p, q\}$.

The precondition $q \in c_p^m$ requires the parents to be friends. Child befriending requires all four—both children and both parents—to be willing; child unfriending can be initiated by any one of the four.

Now that we know what states and transitions look like, we can define Volitional MATS...

Volitional Multiagent Transition Systems

Consider a set of agents P, machine states S, a set R of guarded machine transactions, and an equivalence $\sim$ on the set $T_R := \{t : (t, Q') \in R \text{ for some } Q'\}$ of underlying machine transactions.

The **volitional multiagent transition system induced by** $(S, R, \sim)$ **over** P is the multiagent transition system $(\mathcal{A}^P, T_V, \sim_V)$ where:

(1) $\mathcal{A} := 2^{T_R/\sim} \times S$ is the **agent state space**;

Volitional Multiagent Transition Systems

Consider a set of agents P, machine states S, a set R of guarded machine transactions, and an equivalence $\sim$ on the set $T_R := \{t : (t, Q') \in R \text{ for some } Q'\}$ of underlying machine transactions.

The **volitional multiagent transition system induced by** $(S, R, \sim)$ **over** P is the multiagent transition system $(\mathcal{A}^P, T_V, \sim_V)$ where:

(1) $\mathcal{A} := 2^{T_R/\sim} \times S$ is the **agent state space**;
(2) T_V consists of all transitions of one of two forms: (i) a **volition change**; or (ii) a **volitional machine transaction** induced by some guarded machine transaction $(t, Q') \in R$;

Volitional Multiagent Transition Systems

Consider a set of agents P, machine states S, a set R of guarded machine transactions, and an equivalence $\sim$ on the set $T_R := \{t : (t, Q') \in R \text{ for some } Q'\}$ of underlying machine transactions.

The **volitional multiagent transition system induced by** $(S, R, \sim)$ **over** P is the multiagent transition system $(\mathcal{A}^P, T_V, \sim_V)$ where:

(1) $\mathcal{A} := 2^{T_R/\sim} \times S$ is the **agent state space**;
(2) T_V consists of all transitions of one of two forms: (i) a **volition change**; or (ii) a **volitional machine transaction** induced by some guarded machine transaction $(t, Q') \in R$;
(3) $\sim_V$ is the set of equivalence classes of volitional machine transitions induced by R, relating two of them whenever their inducing machine transactions are $\sim$-equivalent.

The liveness requirement is that, for any volitional machine transaction enabled on all states in a suffix of a run, some equivalent transaction is eventually executed.

Structure

1) Review: transition systems
2) Multi-agent transition systems (MATS)
3) Volitional states, volitional transactions, Volitional MATS.
4) Grassroots protocols.
5) Volitional Transactions-based protocols.
6) Example: Child-safe networks.

Grassroots Protocols

We consider a potentially infinite set of agents Π: any protocol must run for any finite $P \subset \Pi$.

A **protocol** $\mathcal{F}$ is a family of multiagent transition systems that has exactly one multiagent transition system $\mathcal{F}(P) = (S(P), T(P), L(P))$ for every $P \subset \Pi$, where $P \subseteq P'$ implies $S(P) \subseteq S(P')$.

Grassroots Protocols

Informally, in a grassroots protocol two disjoint groups of agents can each operate independently—their interleaved correct runs are correct runs of the combined system—yet the combined system offers genuinely new behaviours that neither group could produce on its own.

To capture this notion formally, we first define the interleaving of runs of two disjoint groups...

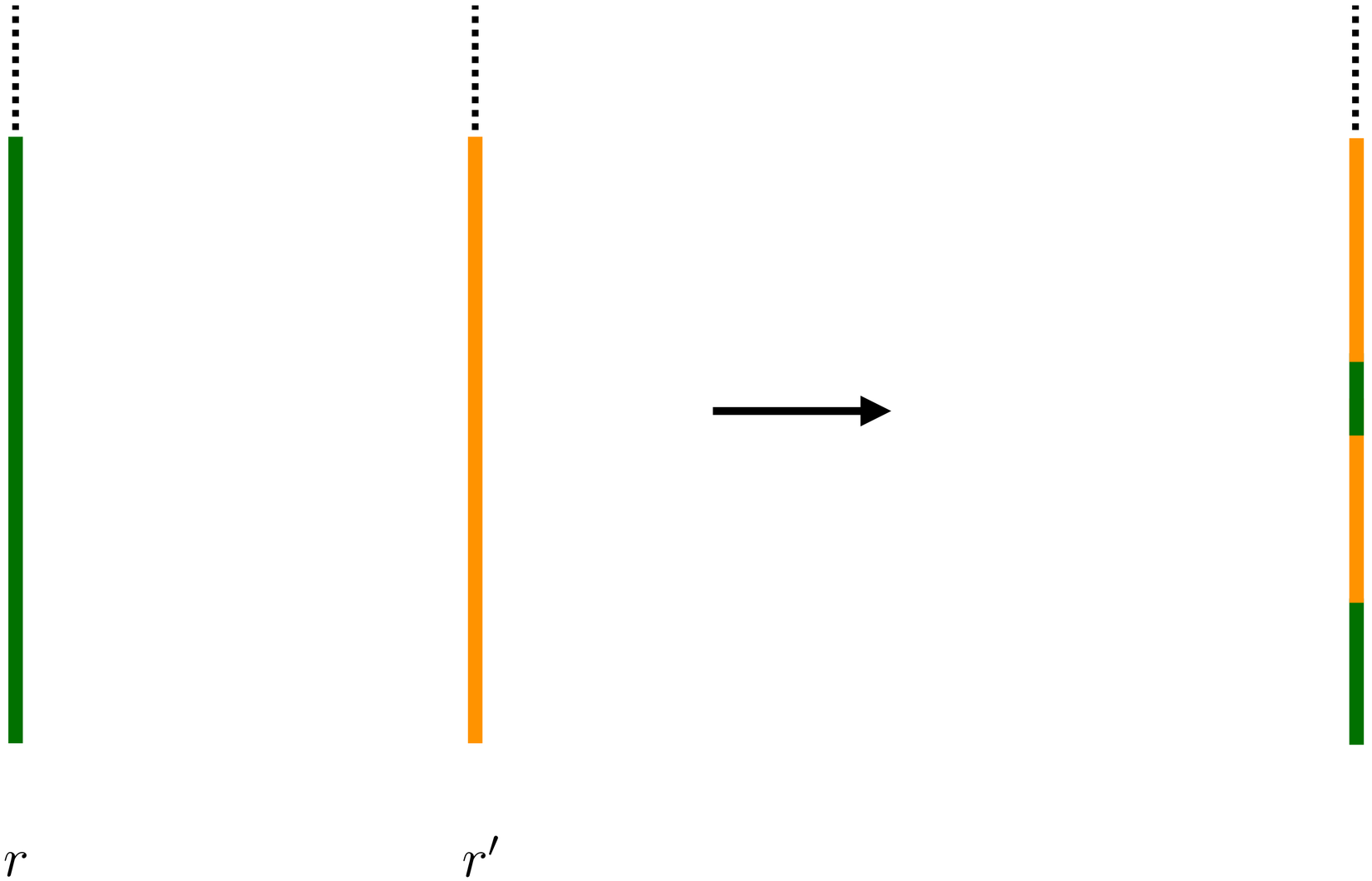
Grassroots Protocols

Let $P, P' \subset \Pi$ be disjoint nonempty sets of agents, r a run of $\mathcal{F}(P)$, and r' a run of $\mathcal{F}(P')$.

Roughly, an **interleaving** of r and r' is a sequence $c_0, c_1, \ldots$ of configurations in $C(P \cup P')$ such that each transition is either the *next* transition in r (leaving the states of members of P' unchanged) or the next transition in r' (leaving the states of members of P unchanged).

We also require that each transition in r is eventually executed, and similarly for r'.

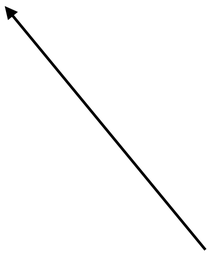
Grassroots Protocols



Interactive runs

Let $P, P' \subset \Pi$ be disjoint and nonempty. For a configuration c over $P \cup P'$ and $Q \subseteq P \cup P'$, write $c|_Q$ for the restriction of c to Q.

A transition (c, c') of $\mathcal{F}(P \cup P')$ is an **interaction between P and P'** if $c|_P \neq c'|_P$ and $c|_{P'} \neq c'|_{P'}$, and $(c|_P, c'|_P)$ is not a transition of $\mathcal{F}(P)$ or $(c|_{P'}, c'|_{P'})$ is not a transition of $\mathcal{F}(P')$.



Something genuinely new, that couldn't happen in $\mathcal{F}(P)$ or $\mathcal{F}(P')$.

Interactive runs

Let $P, P' \subset \Pi$ be disjoint and nonempty. For a configuration c over $P \cup P'$ and $Q \subseteq P \cup P'$, write $c|_Q$ for the restriction of c to Q.

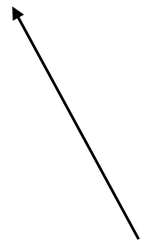
A transition (c, c') of $\mathcal{F}(P \cup P')$ is an **interaction between P and P'** if $c|_P \neq c'|_P$ and $c|_{P'} \neq c'|_{P'}$, and $(c|_P, c'|_P)$ is not a transition of $\mathcal{F}(P)$ or $(c|_{P'}, c'|_{P'})$ is not a transition of $\mathcal{F}(P')$.

A run r of $\mathcal{F}(P \cup P')$ is **interactive between P and P'** if some transition of r is an interaction between P and P'.

Grassroots Protocols

A protocol $\mathcal{F}$ is:

(1) **oblivious** if for every disjoint nonempty $P, P' \subset \Pi$, every interleaving of a correct run of $\mathcal{F}(P)$ and a correct run of $\mathcal{F}(P')$ is a correct run of $\mathcal{F}(P \cup P')$.

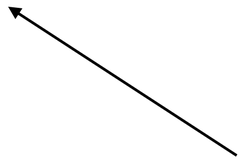


P and P' don't have to interact in an interesting way.

Grassroots Protocols

A protocol $\mathcal{F}$ is:

(1) **oblivious** if for every disjoint nonempty $P, P' \subset \Pi$, every interleaving of a correct run of $\mathcal{F}(P)$ and a correct run of $\mathcal{F}(P')$ is a correct run of $\mathcal{F}(P \cup P')$.
(2) **interactive** if for every disjoint nonempty $P, P' \subset \Pi$, some correct run of $\mathcal{F}(P \cup P')$ is interactive between P and P'.



P and P' can interact in an interesting way.

Grassroots Protocols

A protocol $\mathcal{F}$ is:

(1) **oblivious** if for every disjoint nonempty $P, P' \subset \Pi$, every interleaving of a correct run of $\mathcal{F}(P)$ and a correct run of $\mathcal{F}(P')$ is a correct run of $\mathcal{F}(P \cup P')$.
(2) **interactive** if for every disjoint nonempty $P, P' \subset \Pi$, some correct run of $\mathcal{F}(P \cup P')$ is interactive between P and P'.
(3) **grassroots** if it is oblivious and interactive.

Volitional Transactions-Based Protocols

Let S be a function mapping any P to a set of machine states. Let R be a set of guarded transactions over S with equivalence $\sim$.

Volitional Transactions-Based Protocols

Let S be a function mapping any P to a set of machine states. Let R be a set of guarded transactions over S with equivalence $\sim$.

The volitional transactions-based protocol $\mathcal{F}$ over R, S, and $\sim$ assigns to each set of agents $P \subset \Pi$ the volitional multiagent transition system $\mathcal{F}(P)$ induced by $(S(P), R(P), \sim)$ over P. In particular, $C(P) = \mathcal{A}(P)^P$ with agent state space $\mathcal{A}(P) := 2^{T_{R(P)}/\sim} \times S(P)$.

basically, the correct volitional machine transactions are those induced by guarded transactions in R

Volitional Transactions-Based Protocols

One of the reasons volitional transactions-based protocols are nice to work with is the following result...

Theorem. A volitional transactions-based protocol over a set of guarded transactions R is oblivious if every guarded transaction $(t, Q') \in R$ whose machine transaction t has two or more participants has a nonempty guard, $Q' \neq \emptyset$.

Back to child-safe networks...

Recall that each agent p maintains a set of friends $c_p^m \subseteq P$ (their machine state). In what follows, r, s are children with respective parents p, q. We specify the protocol using two types of guarded machine transaction.

(1) **Child befriend**: $(c')_r^m := c_r^m \cup \{s\}$, $(c')_s^m := c_s^m \cup \{r\}$, provided $q \in c_p^m$ and $s \notin c_r^m$. Guarded by $\{r, s, p, q\}$.
(2) **Child unfriend**: $(c')_r^m := c_r^m \setminus \{s\}$, $(c')_s^m := c_s^m \setminus \{r\}$, provided $s \in c_r^m$. Guarded by any one of $\{r, s, p, q\}$.

Back to child-safe networks...

Recall that each agent p maintains a set of friends $c_p^m \subseteq P$ (their machine state). In what follows, r, s are children with respective parents p, q. We specify the protocol using two types of guarded machine transaction.

(1) **Child befriend**: $(c')_r^m := c_r^m \cup \{s\}$, $(c')_s^m := c_s^m \cup \{r\}$, provided $q \in c_p^m$ and $s \notin c_r^m$. Guarded by $\{r, s, p, q\}$.
(2) **Child unfriend**: $(c')_r^m := c_r^m \setminus \{s\}$, $(c')_s^m := c_s^m \setminus \{r\}$, provided $s \in c_r^m$. Guarded by any one of $\{r, s, p, q\}$.

To prove that the protocol is grassroots, we need to prove it is oblivious and interactive. Proving that it is interactive is striaghtforward: any run of $\mathcal{F}(P \cup P')$ in which a member of P befriends a member of P' is interactive.

But given the previous theorem, obliviousness also follows directly.

Part 3: Constitutional Consensus for Democratic Governance

Part 3: Constitutional consensus for democratic governance

1. **Digital democracy, Paxos, and Consensus**
2. Constitutional consensus
3. Digital democratic governance

Democracy and Consensus go together

- No dictator
- Need distributed agreement on democratic decisions

Democratic governance vs Consensus

- Democratic governance is based on majority or super-majority votes
- Consensus algorithms use majority or super-majority

Q: Are they the same?

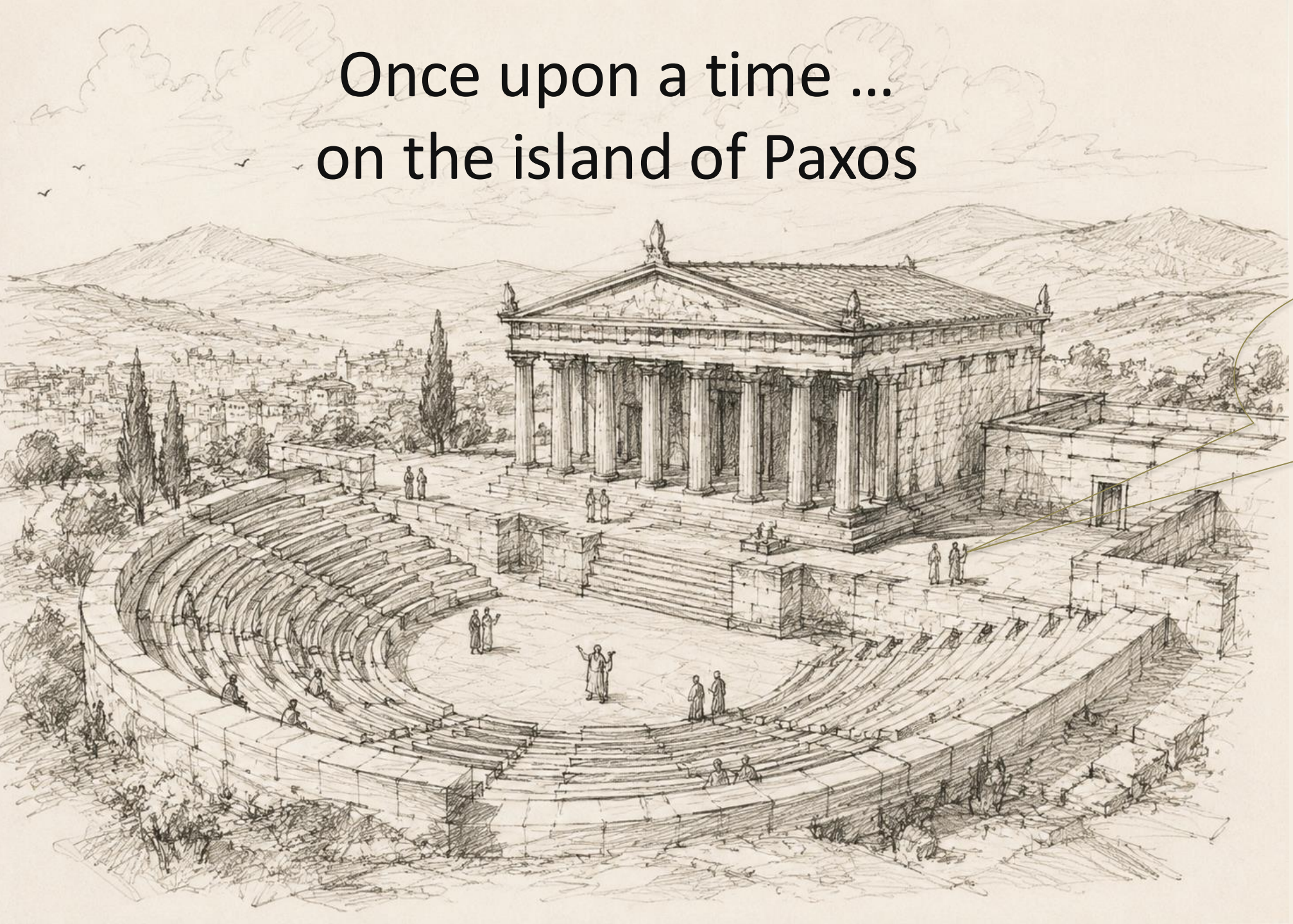
Democratic governance vs Consensus

Q: Are they the same?

A1: Consensus super-majorities do not reflect volitional agreement of participants, whereas democratic governance does.

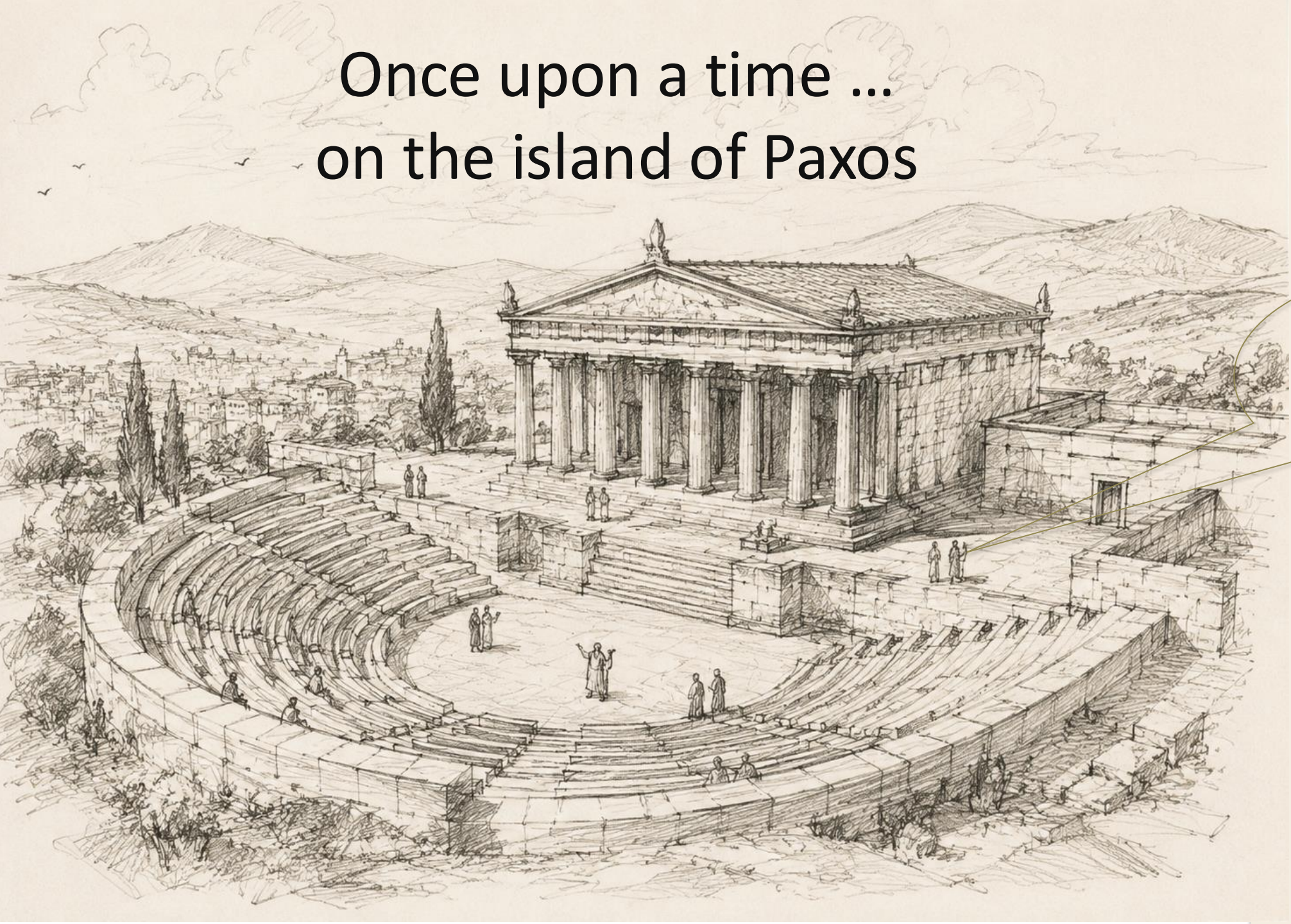
A2: For a majoritarian democratic decision to be accepted by everyone (including the losing minority), we need consensus that it is valid.

Once upon a time …
on the island of Paxos



155: The olive tax is 3 drachmas per ton

Once upon a time …
on the island of Paxos



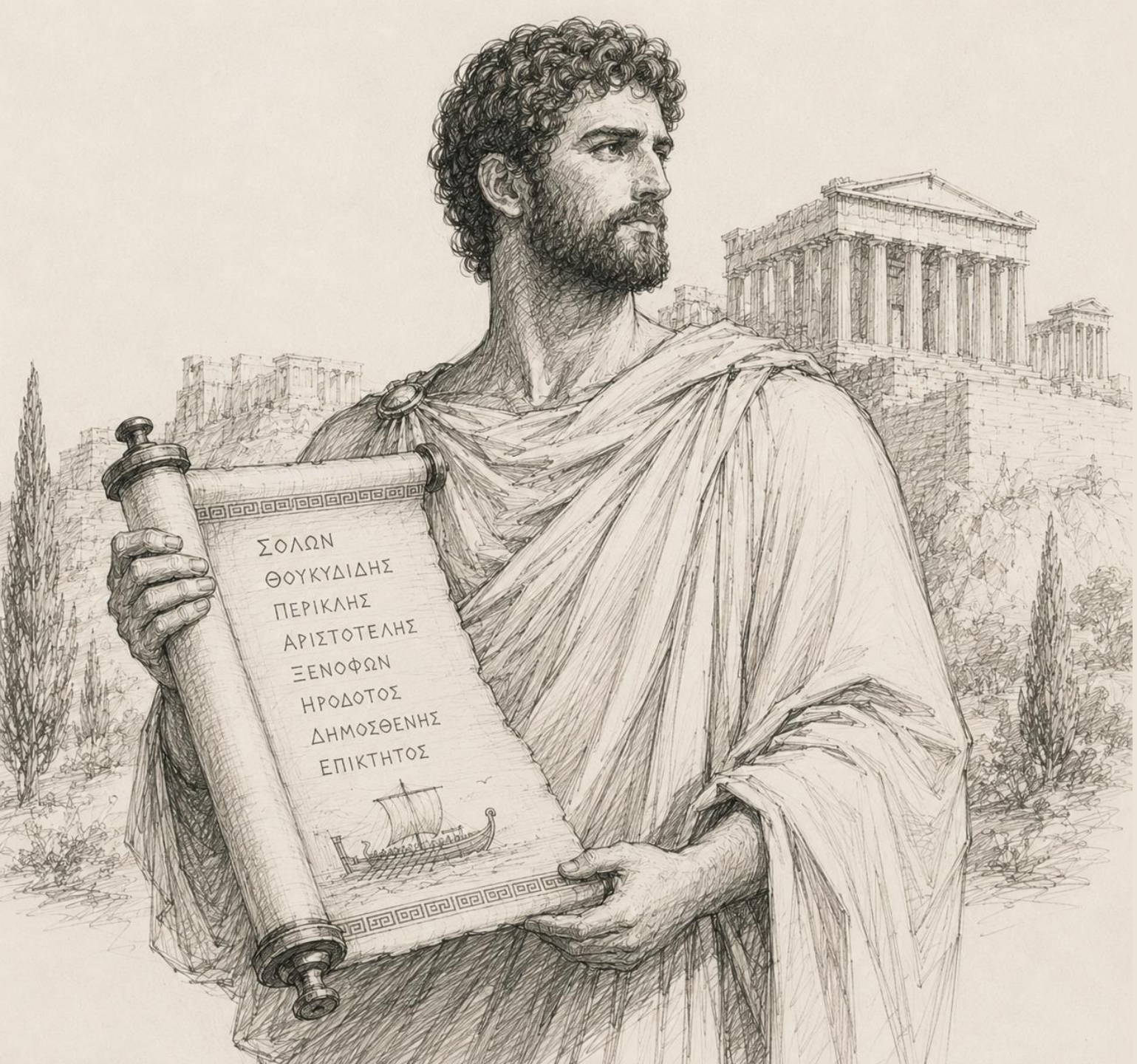
3252: Στρωνγ is now a legislator

Digital democracy using Consensus

1. Agree on a sequence of transactions
   - Consistent parliamentary record as in Paxos
2. Agree on the constitution and its amendments
   - Still a challenge in Paxos

From “The Part Time Parlament” Lamport 1998

In fact, the mechanism for choosing legislators led to the downfall of the parliamentary system in Paxos.



Because of a scribe’s error, a decree that was supposed to honor sailors who had drowned in a shipwreck instead declared them to be the only members of Parliament.

From “The Part Time Parlament” Lamport 1998

In fact, the mechanism for choosing legislators led to the downfall of the Parliamentary system in Paxos.

…

Its passage prevented any new decrees from being passed— including the decrees proposed to correct the mistake. Government in Paxos came to a halt.

Democracy needs a constitution

Q: Consensus that a democratic decision is valid — How?

A: With a constitution defining a valid democratic decision

Q: Democratically amending the constitution itself — How?

A: A difficult question…which we try to answer here

Constitutional Consensus principles

1. Vision: A consensus protocol to be used by grassroots democratic communities, grassroots cooperatives
2. Executed by *volitional* agents = people with smartphones → participants = users
3. Supports nascent communities → *quiescent*, efficient also at low transaction volume (Morpheus Consensus)
4. Supports constitutional *amendments* → Epochs, one per constitution

Part 3: Constitutional consensus for democratic governance

1. Digital democracy, Paxos, and Consensus
2. **Constitutional consensus**
3. Digital democratic governance

The model

- Set of agents Π
  - Each has a unique key-pair
  - Correct agents follow the protocol
  - Others are *faulty* or *Byzantine*
- Eventual synchrony
  - Bounded message delay δ after GST

The constitution

- Rules for consensus decisions
  - In our case: our protocol, 3 amendable parameters
- Rules for amending the constitution
  - Later

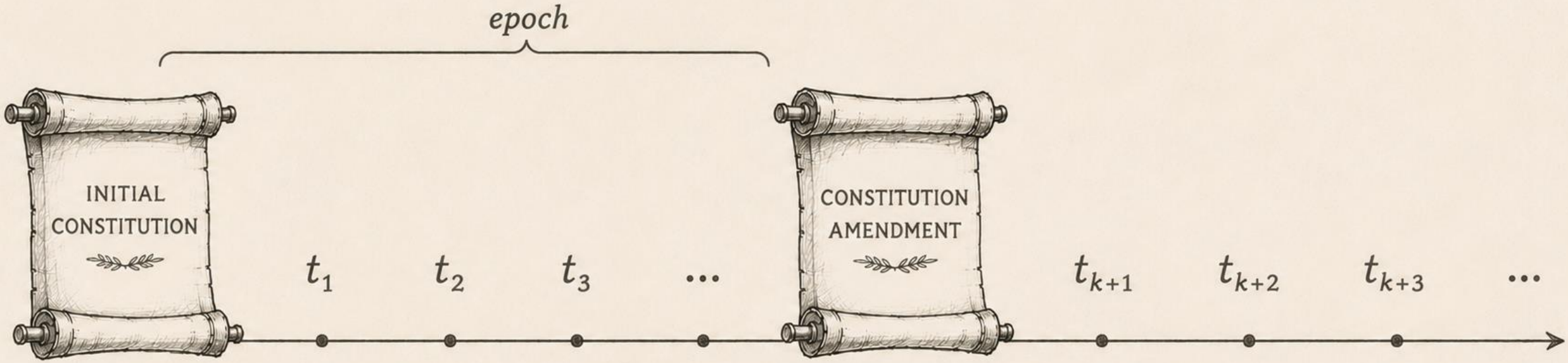
Amendable constitution parameters

- $P \subseteq \Pi$ – set of participants
- $\frac{1}{2} \leq \sigma < 1$ – super-majority threshold
  - Capturing Byzantines and sybils
- Δ – estimated latency bound after GST
  - Among correct participants

Constitution safety and liveness

- A constitution (P, σ, Δ) is *safe* if
  - P is the set of agents running the protocol;
  - there is a correct σ-supermajority in P , and every σ-supermajority in P includes a correct majority.
- The constitution is *live* if
  - #of faulty participants $f < \frac{1}{3}n$, and
  - $\delta \leq \Delta$

Constitutional consensus epochs

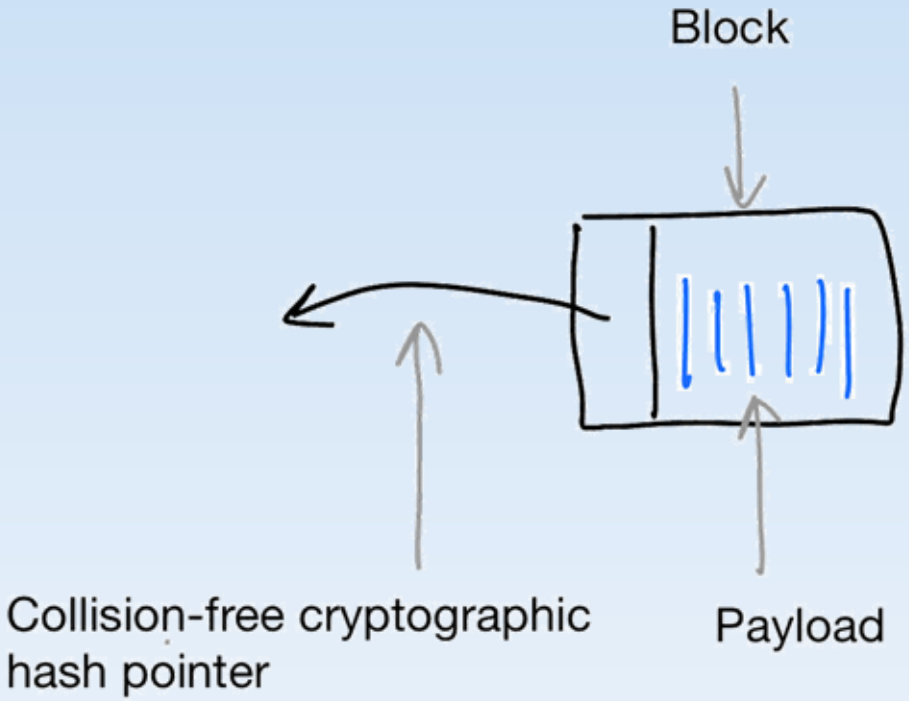


Every agent *p* sees a subsequence consisting of epochs whose amendments include *p.*

Single epoch protocol

- Invoked with a given safe and live constitution
- Orders transactions
- Ends upon ordering of a valid constitution amendment
- Based on a *blocklace* data structure
  - Generalization of blockchain

A Blockchain Block

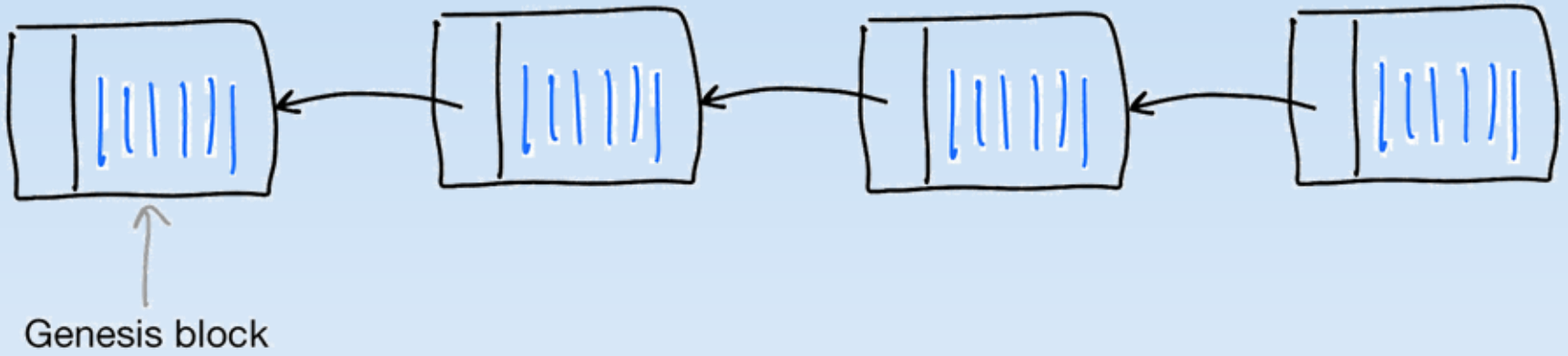


Hash pointer: the hash value $h(x)$ of another block x.

Collision-free: $x \neq y \rightarrow h(x) \neq h(y)$.

Cryptographic: Cannot compute x given $h(x)$; hence no cycles.

A Blockchain

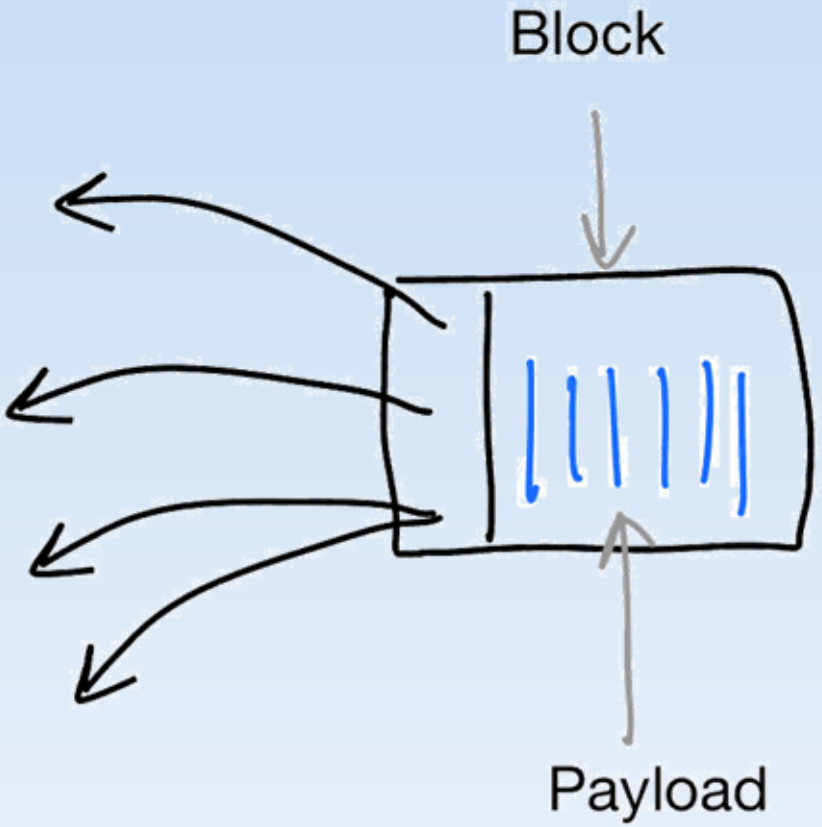


Tamper-proof: payload or pointer cannot be changed without being detected

Non-repudiable: Signed block or payload cannot be repudiated by signatory

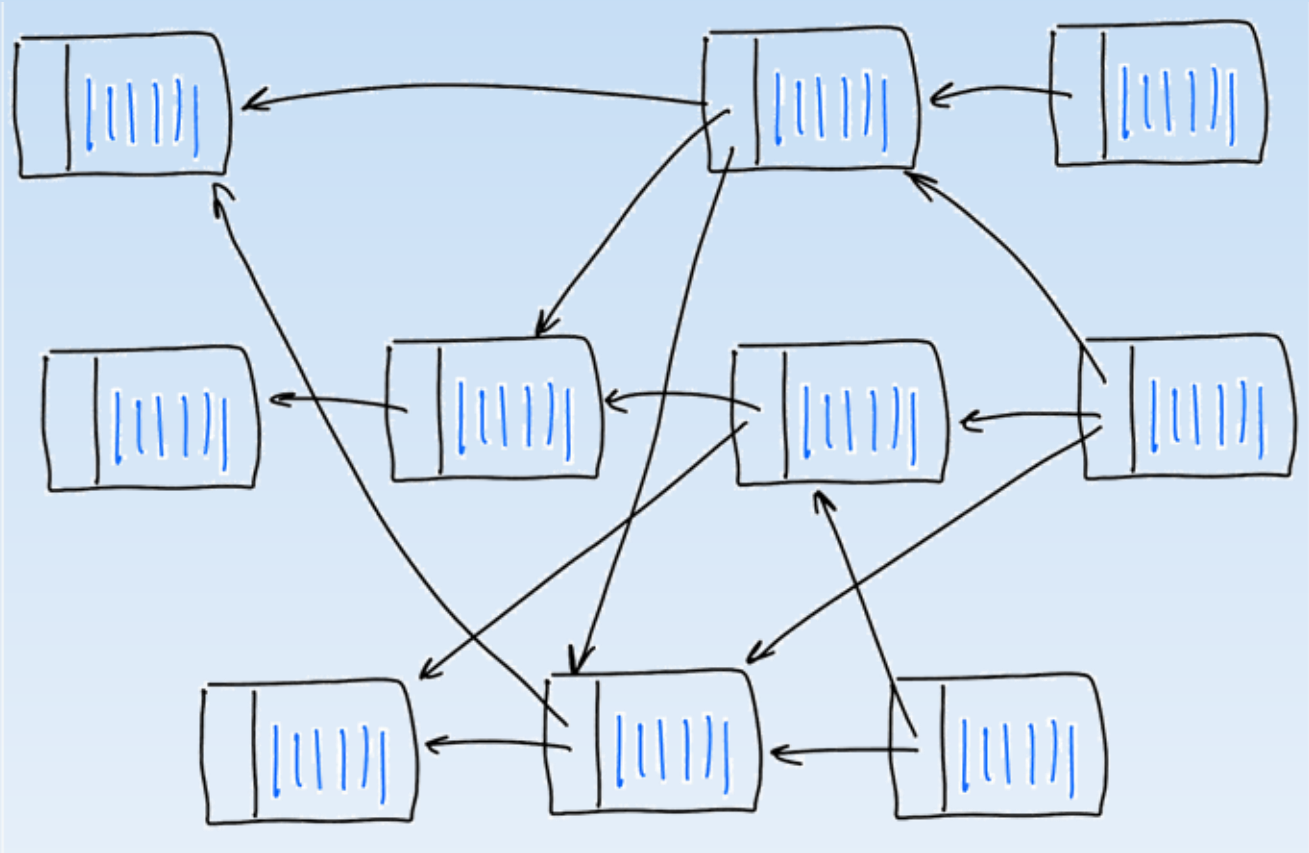
Totally-ordered

A Blocklace Block



hash pointers

A Blocklace

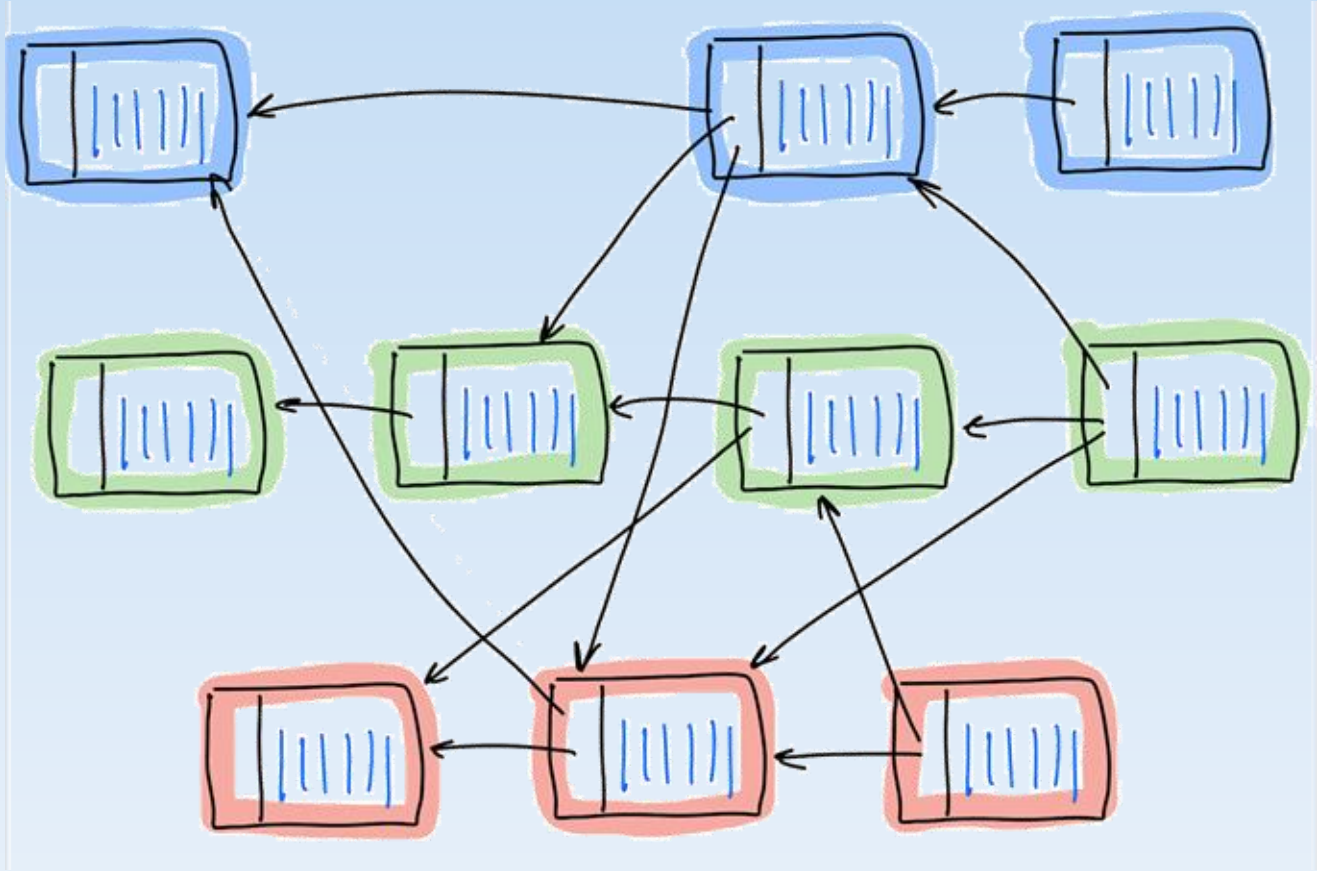


Tamper-proof

Non-repudiable - signed

~~**Totally**~~ **Partially-ordered** – induces a Directed Acyclic Graph (DAG)

A Blocklace

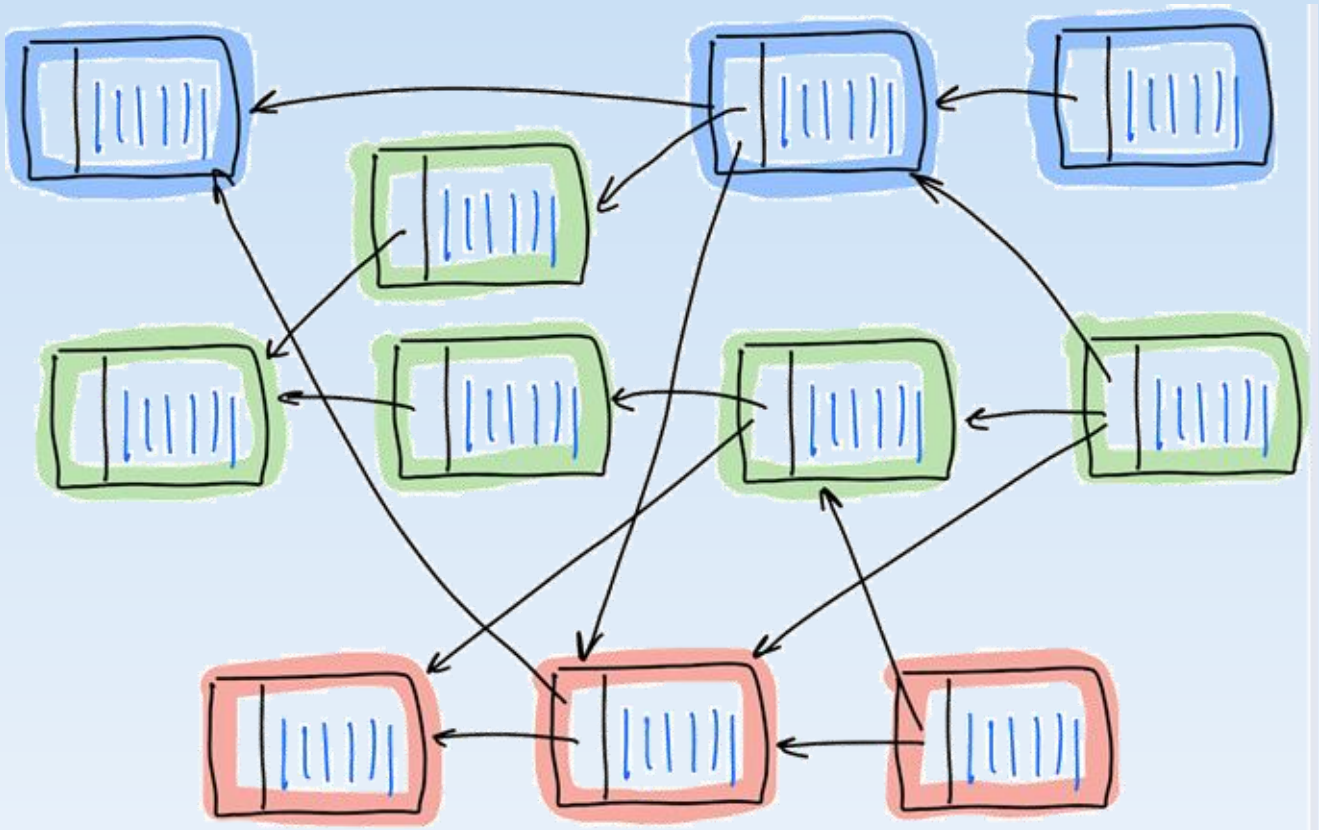


Tamper-proof

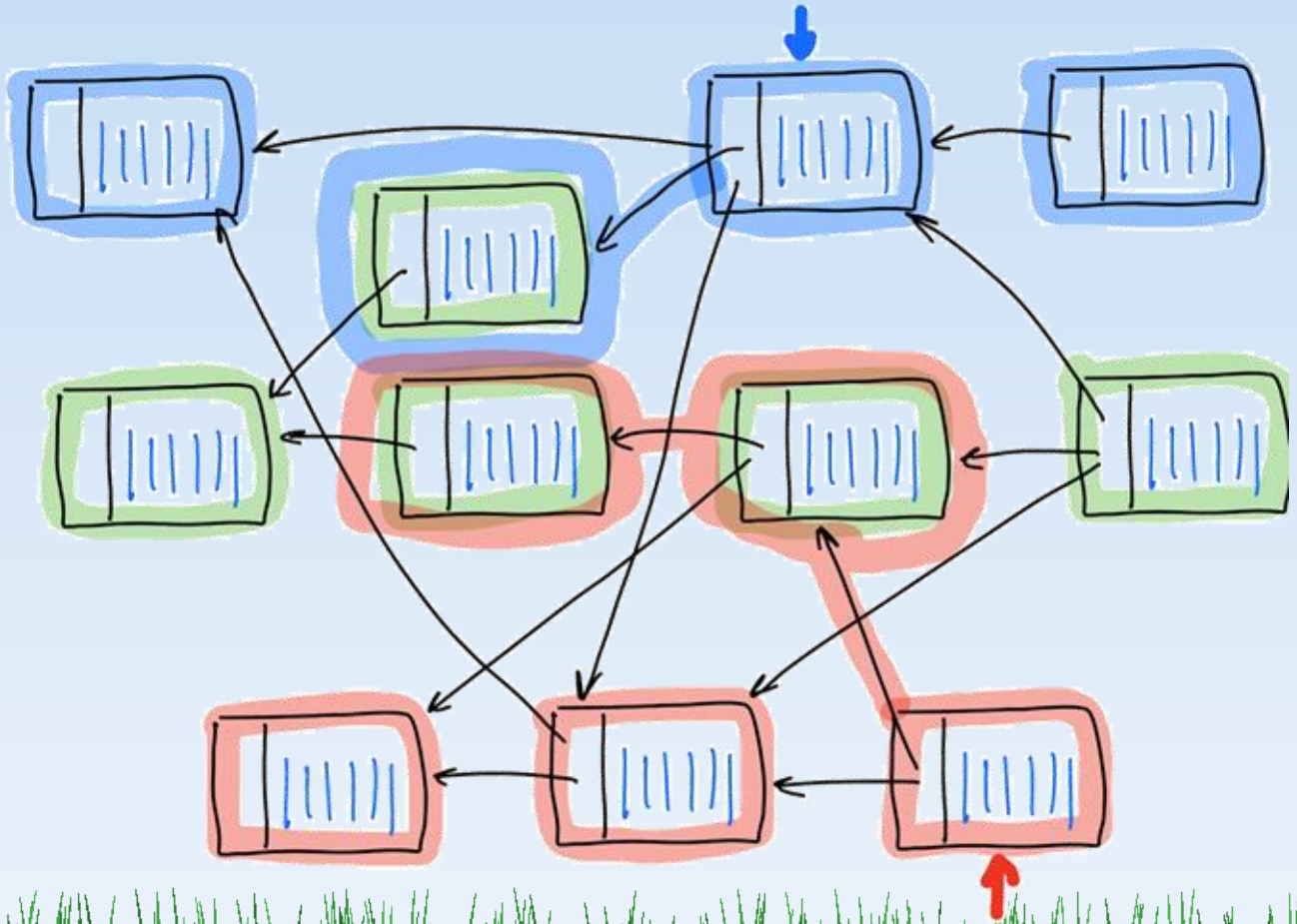
Non-repudiable - signed

~~**Totally**~~ **Partially-ordered** – induces a Directed Acyclic Graph (DAG)

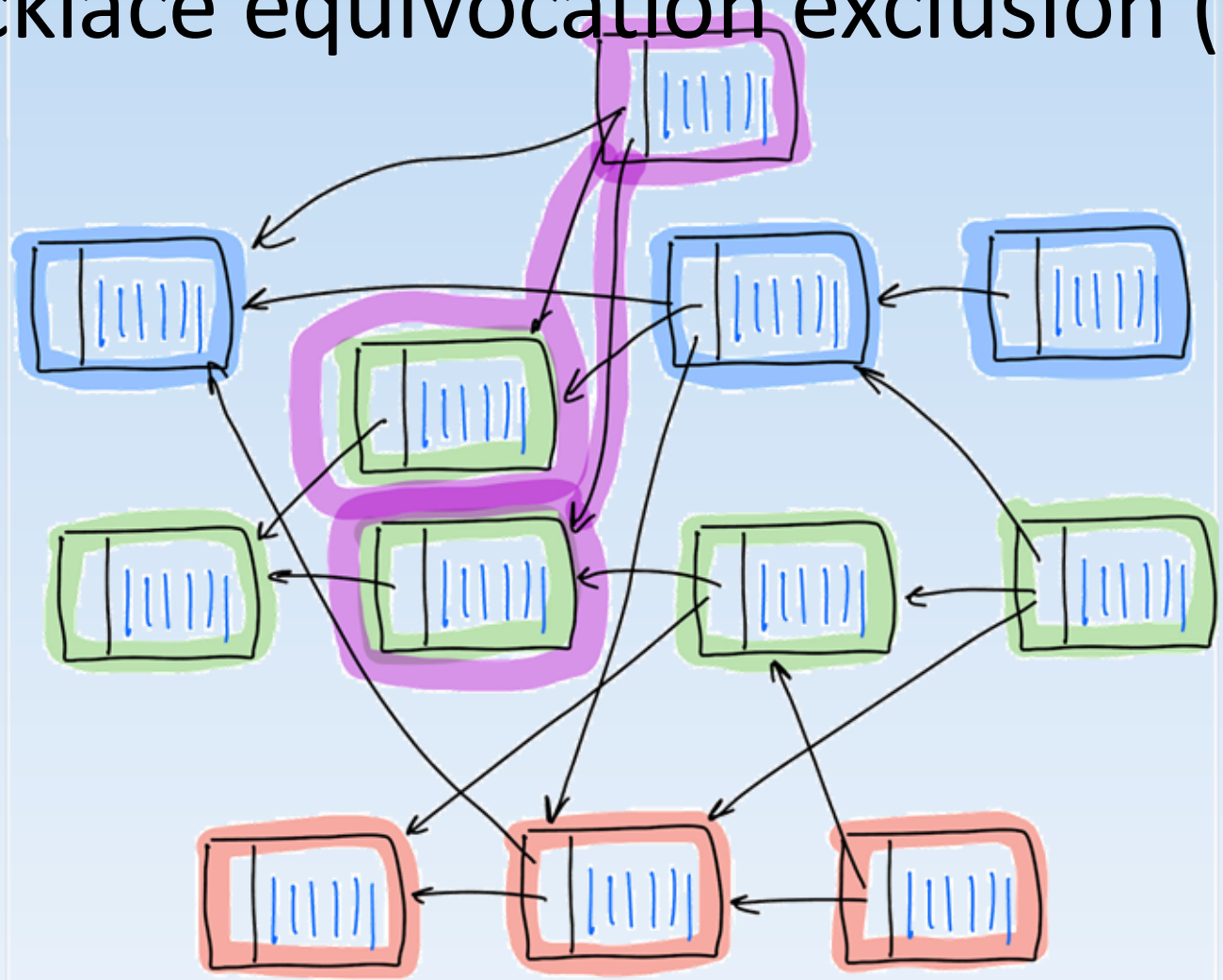
Blocklace equivocation exclusion (Cordial Miners)



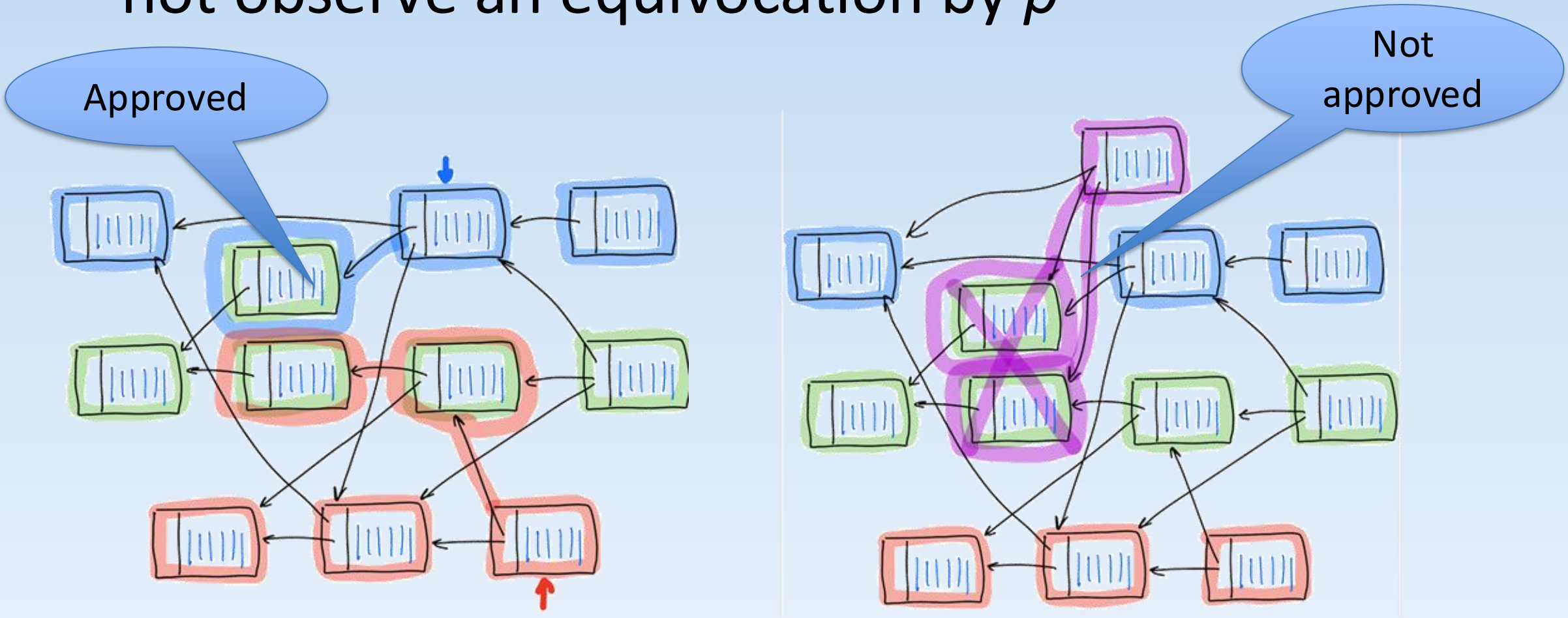
Blocklace equivocation exclusion (Cordial Miners)



Blocklace equivocation exclusion (Cordial Miners)



A block *approves* a *p*-block if it does not observe an equivocation by *p*

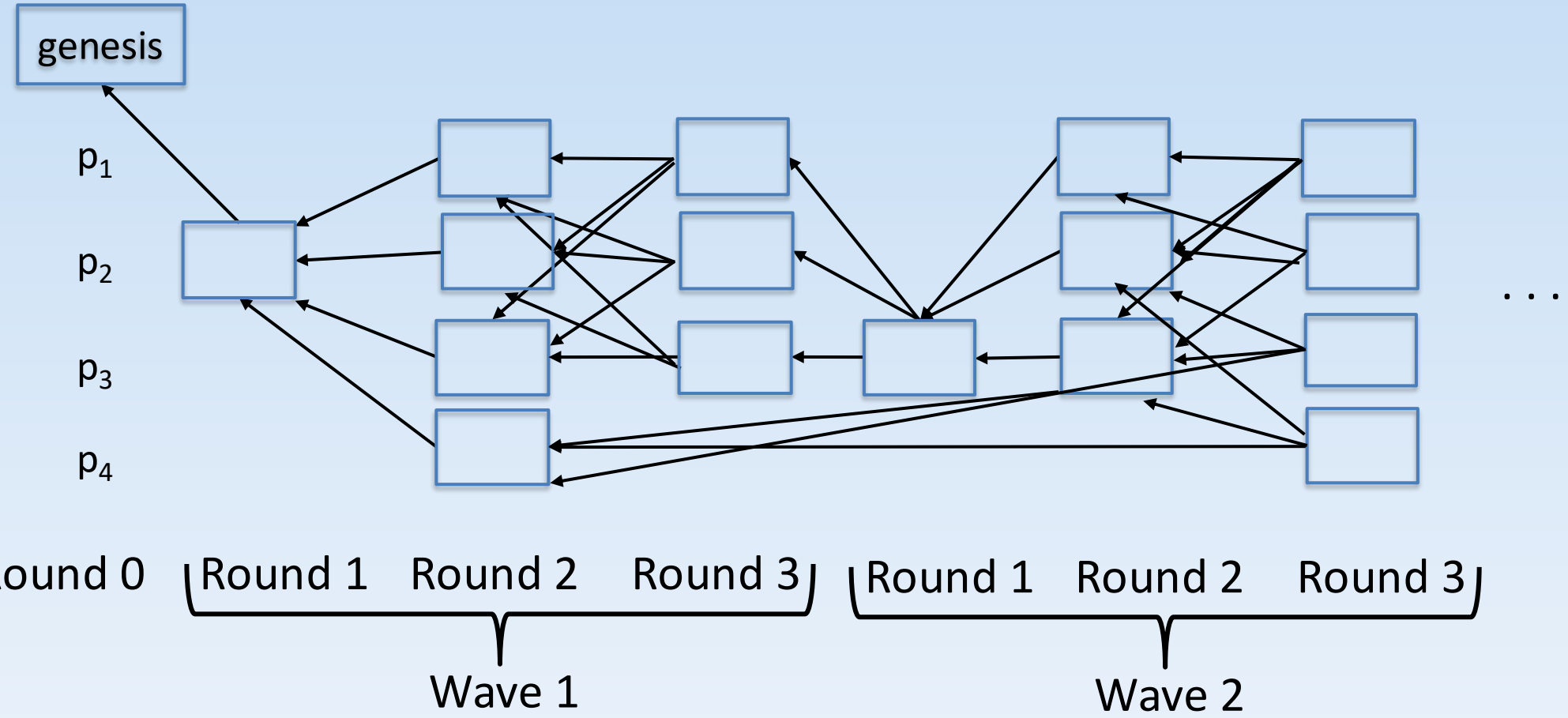


Blocklace eventual consistency

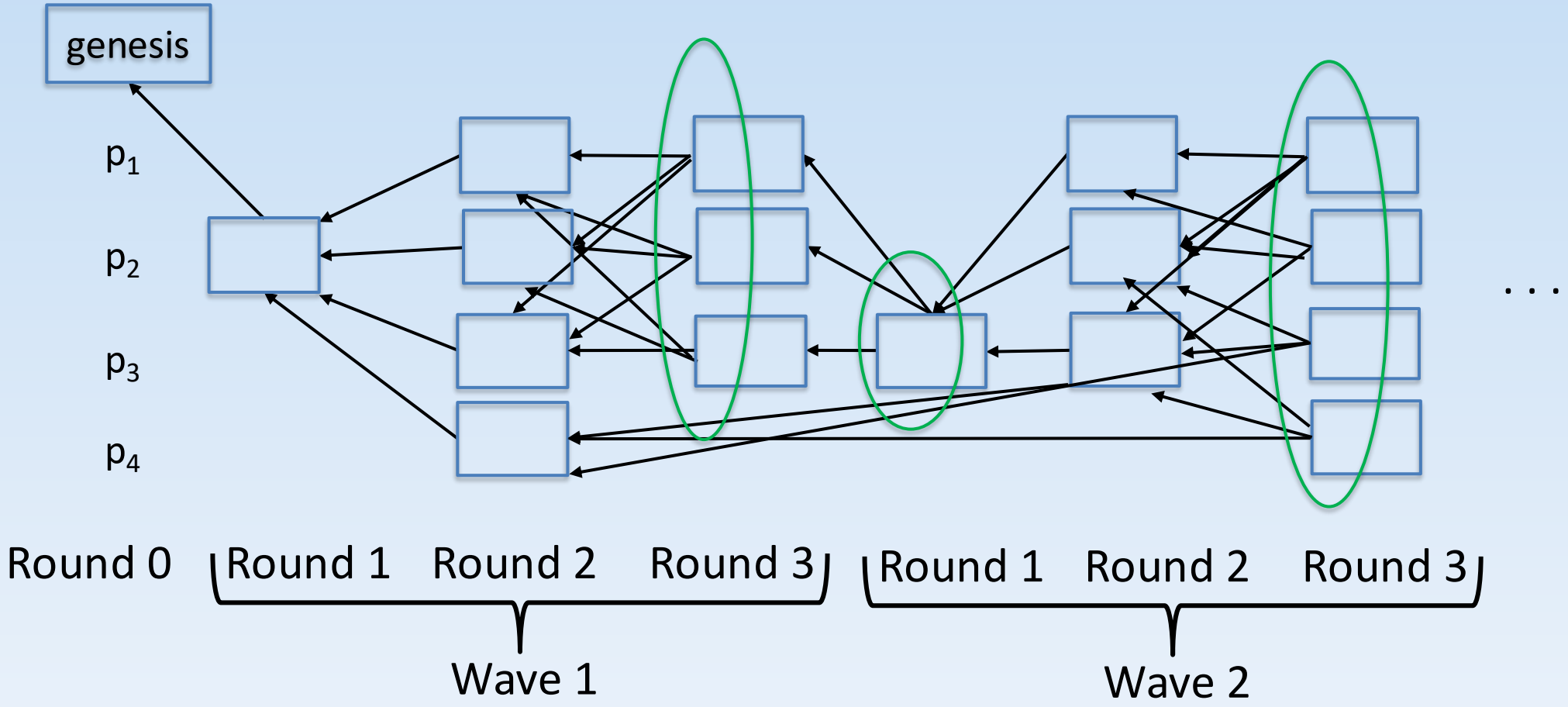
- The blocklace may be revealed to different agents in different orders
- An agent that receives a block pointing to a block it does not have, requests the missing block

∴ Eventually all agents observe a DAG including all the blocks generated by correct agents

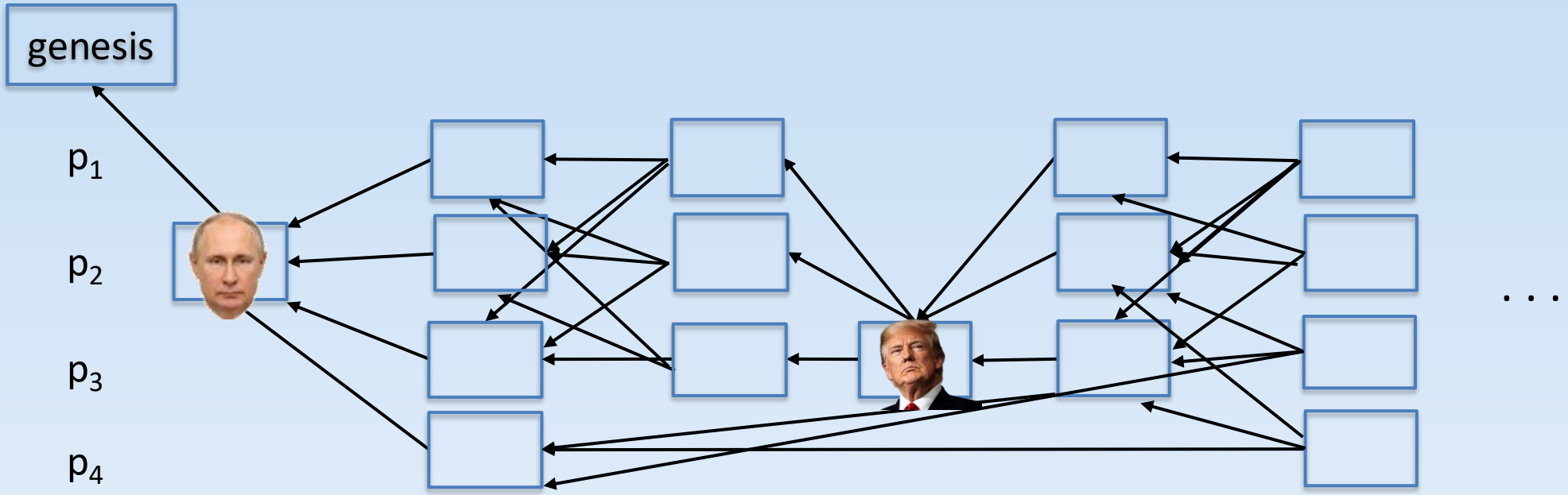
Constitutional Consensus
blocklace waves and rounds



Round 1 & 3 blocks point to a supermajority



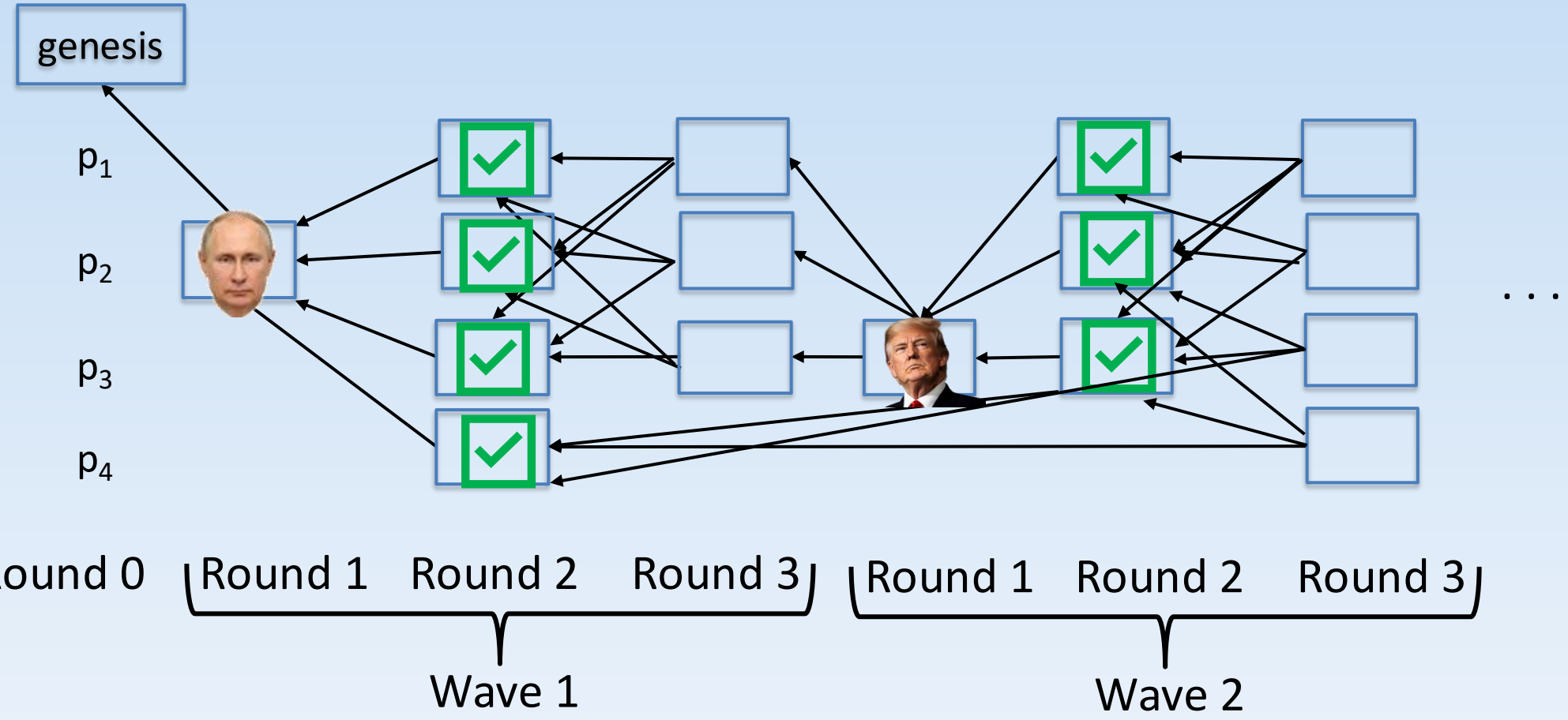
Round 1 is for leaders



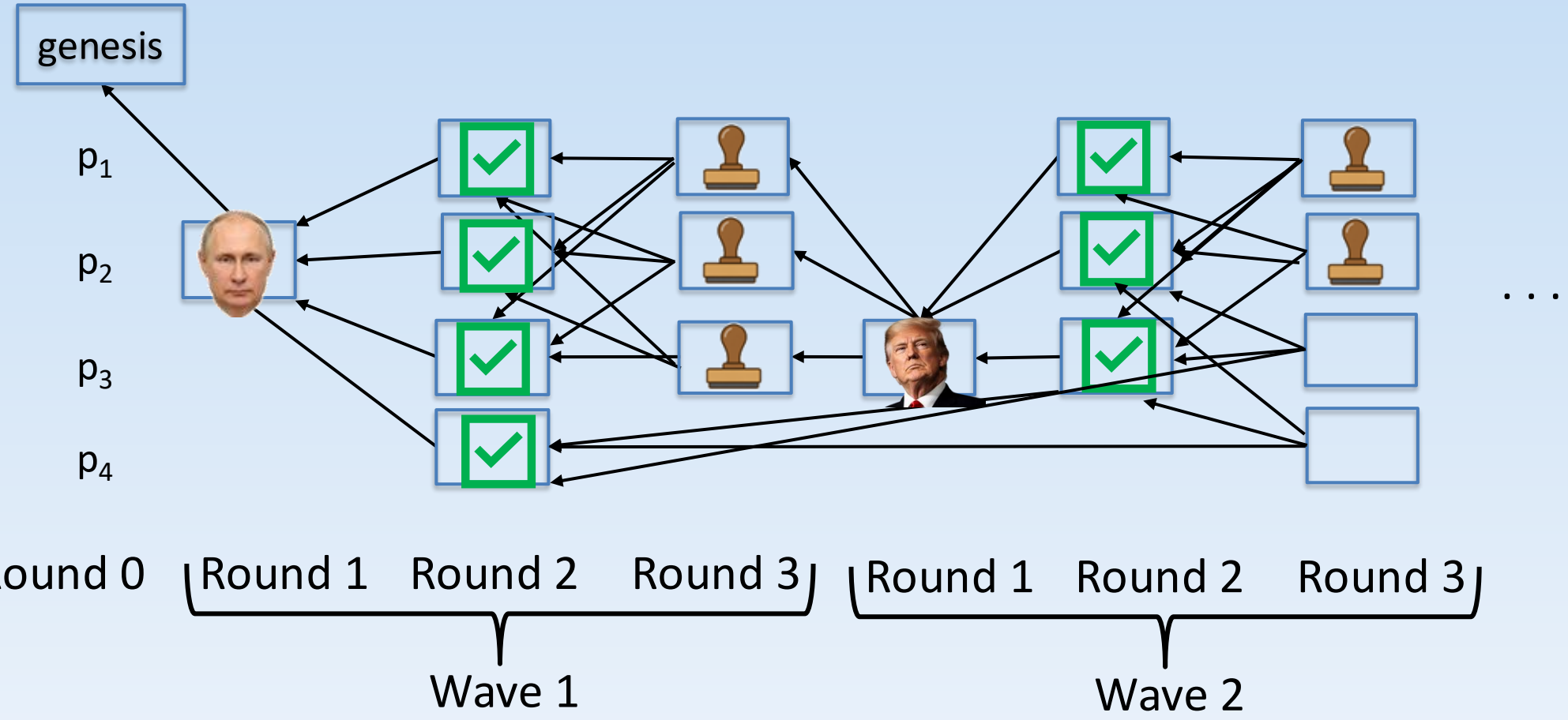
Round 0 | Round 1 | Round 2 | Round 3 | Round 1 | Round 2 | Round 3

Wave 1 | Wave 2

Round 2 is for *endorsing* one leader by approving only its block



Round 3 is for *ratification*
by approving a super-majority of endorsers

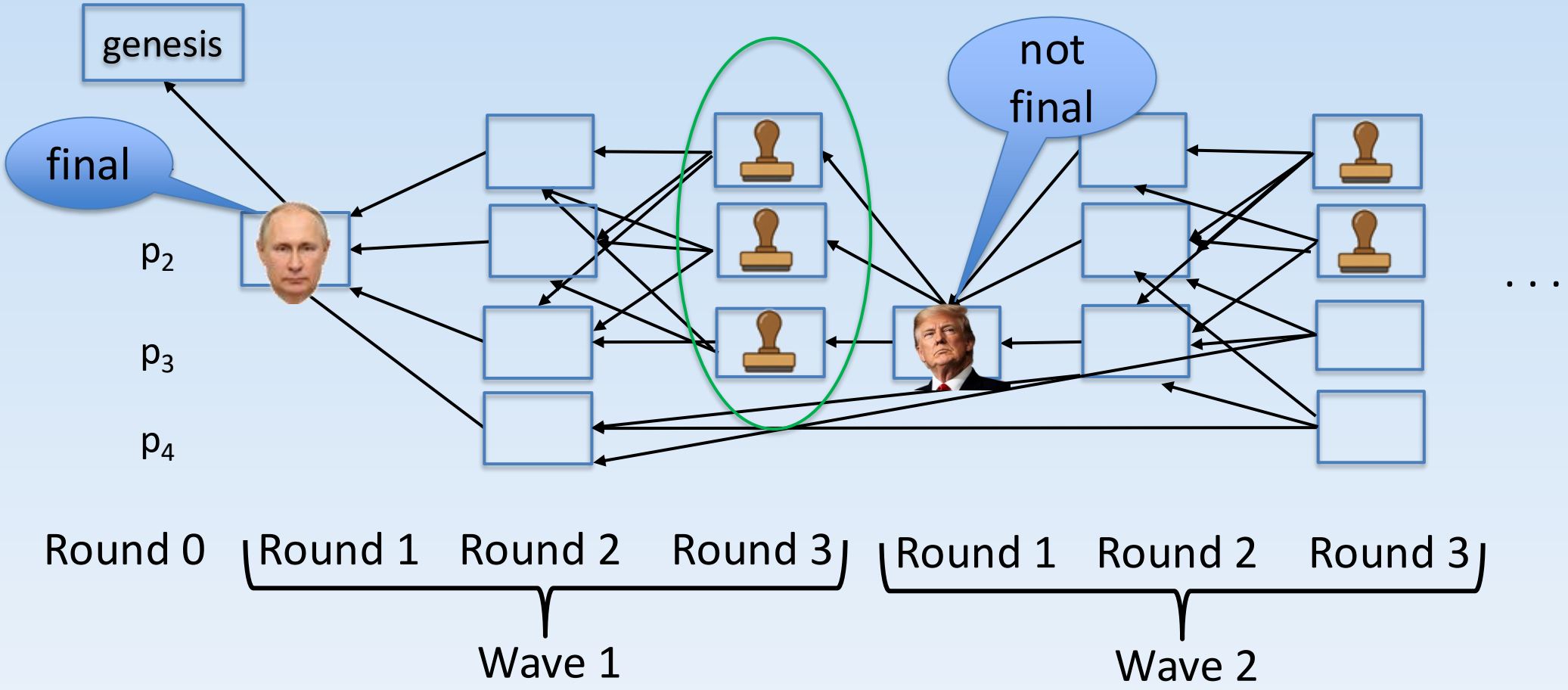


Ratification safety

- A round 2 block endorses at most one round 1 leader block
- Ratification requires a super-majority of endorsements
- Correct agents do not equivocate
- Two super-majorities intersect at at least one correct agent

∴ At most one block is ratified in a wave

A leader ratified by a super-majority is *final*

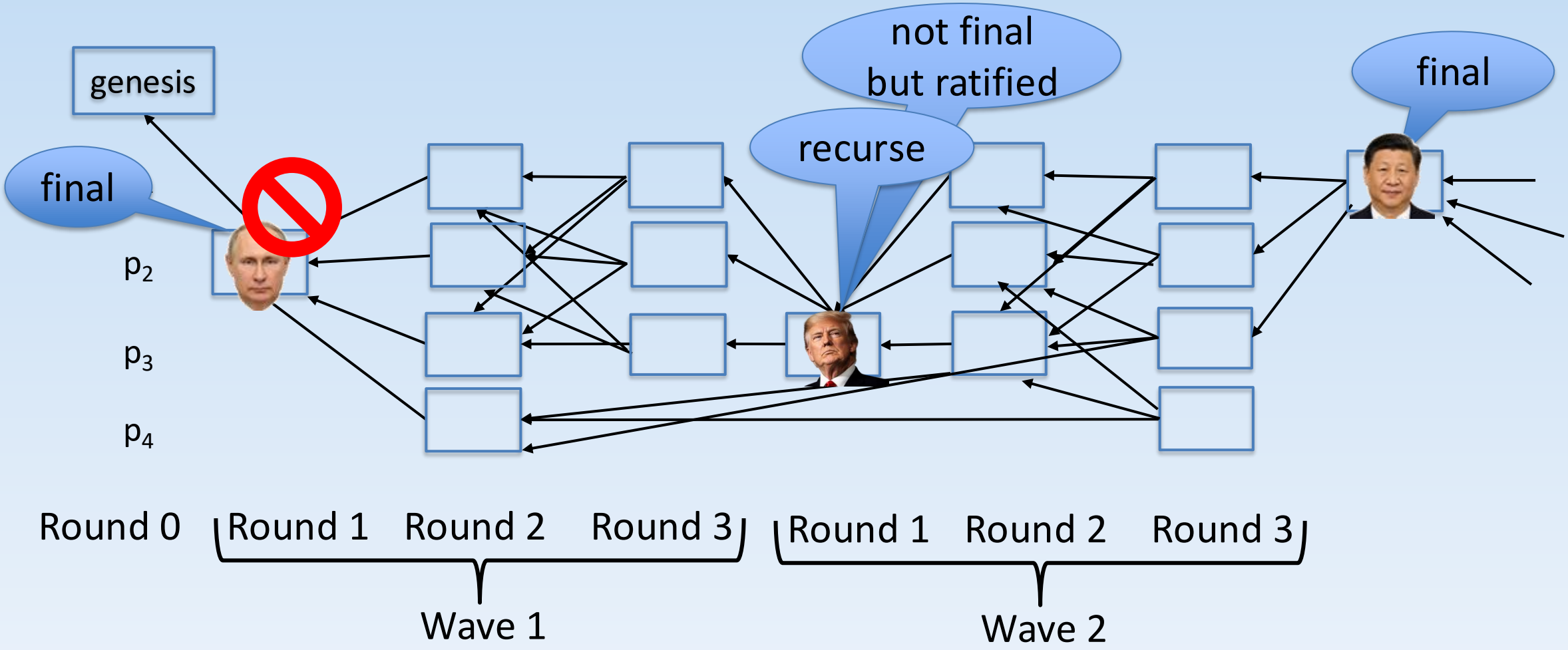


Finalization safety

- A final block b is ratified by a super-majority in round 3
- A round 1 block includes a super-majority of pointers, hence observes that b is ratified

∴ A final block is observed as ratified by every correct agent at every subsequent wave

Finalization triggers ordering on the spanned DAG

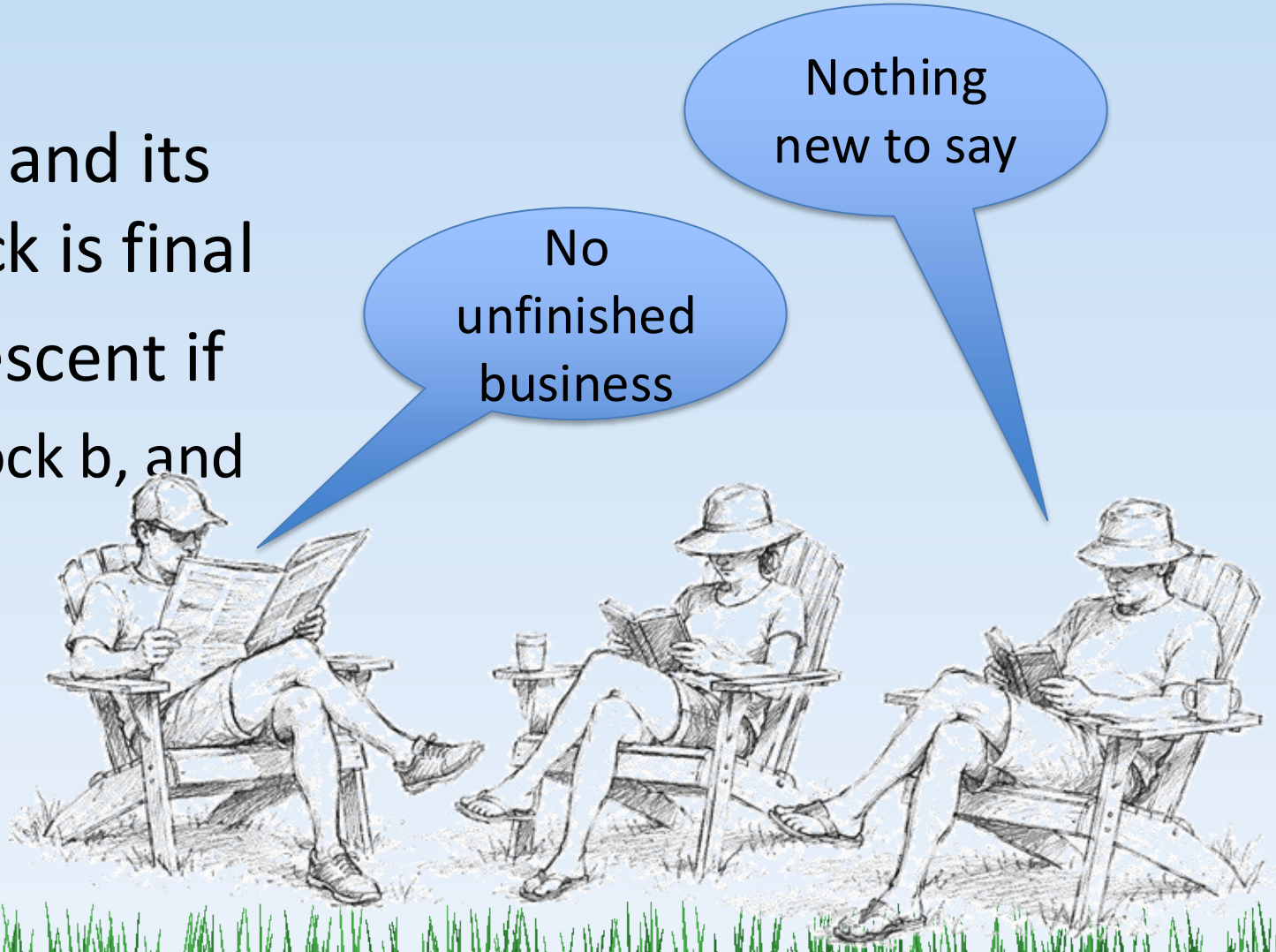


Nascent communities

- An instance of constitutional consensus may be invoked in a small community
  - which might later grow or coalesce with others
- There can be extensive periods with nothing to order
  - Minutes, hours, even days
- The system becomes *quiescent*

Quiescent waves

- The 0th-wave is quiescent and its constitutional genesis block is final
- A subsequent wave is quiescent if
  - it includes a final leader block b, and
  - all the blocks in the wave except b are empty



Low and high throughput (Morpheus)

- After a quiescent wave, the algorithm operates in *low throughput mode*.
  - Generates traffic only when agents have payloads.

- Otherwise, it operates in *high throughput mode*.
  - Generates blocks in all rounds.

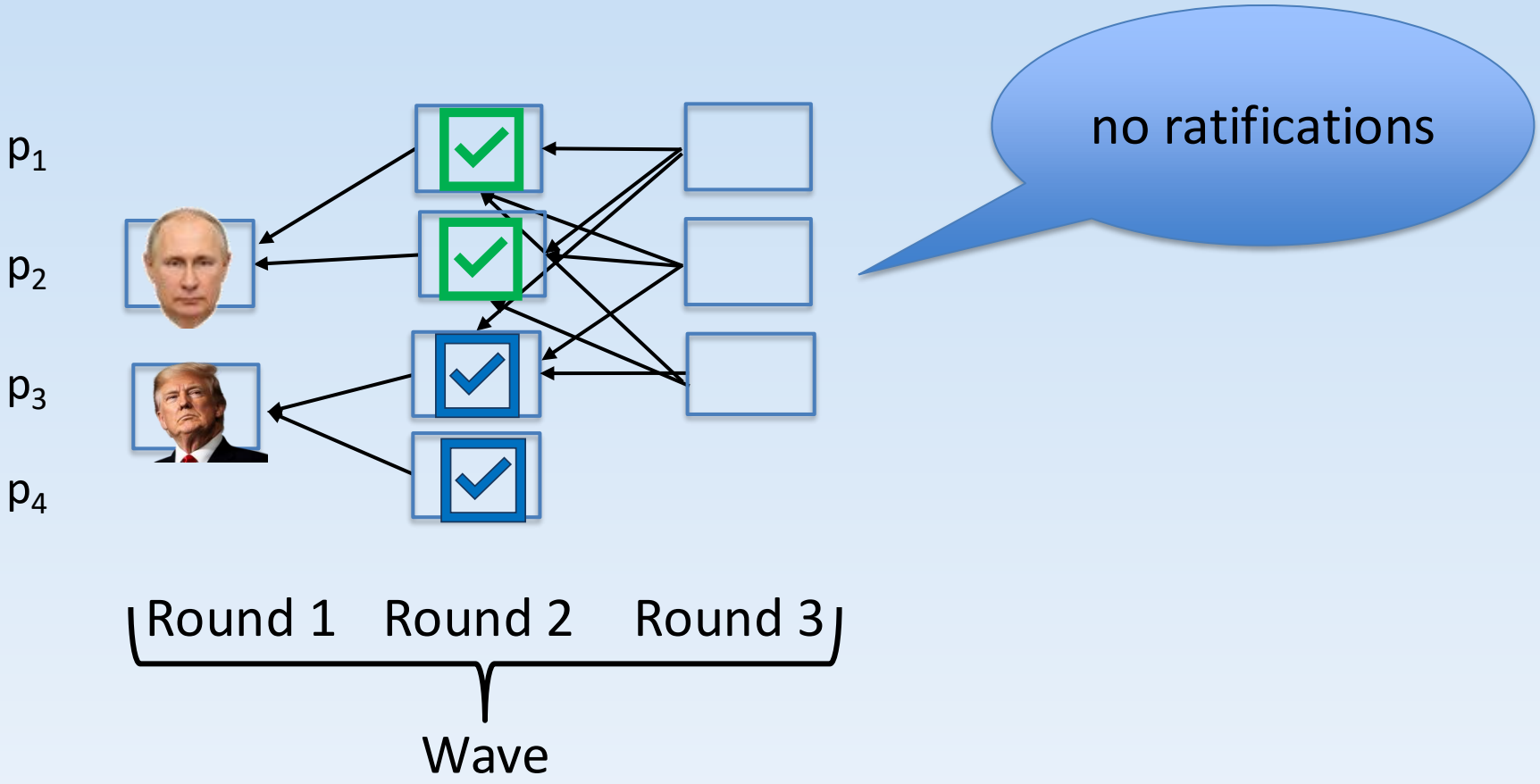
High throughput – formal leaders

- A deterministic and fair *leader* function selects a unique *formal leader* for each wave.
- In the good case, only the formal leader sends a round 1 block.
- In case of timeout, other agents send round 1 blocks.
  - May jeopardize finalization in this wave.
- Round 2 begins after there are blocks from either the formal leader or a supermajority.

Low throughput – spontaneous leaders

- All agents are quiet as long as they have no payloads.
- An agent with payload emerges as a *spontaneous leader.*
- If unique,
  it is endorsed by
  all correct agents.

Multiple leaders can prevent progress



Advanced rounds

- Round 0 is advanced.
- A round 1 is advanced if it includes a leader (either formal or spontaneous) or a super-majority.
- A round 2 or 3 is advanced if it includes a super-majority.

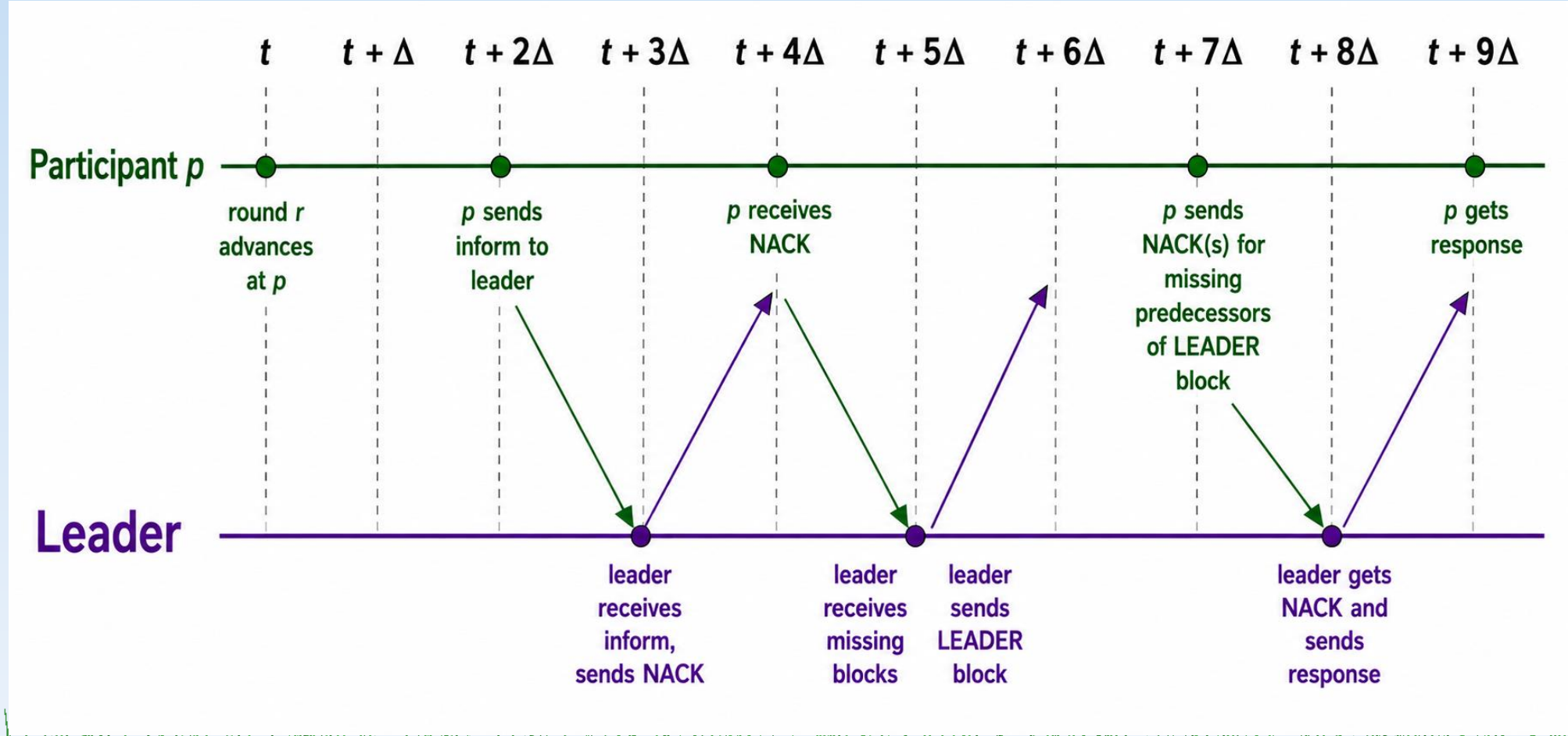
When to send blocks?

- Let r be the maximum advanced round
- Send a round r+1 block:
  - Rounds 2 and 3: immediately
  - Round 1 and quiescent: once I have payload
  - Round 1 and non-quiescent:
    - if I am the formal leader or
    - the round has been advanced for 9Δ
- Send backlog (in round r) if I have payload

NACKs, inform-leader, and timeouts

- Missing block (pointed by received block)
  - Wait Δ before sending a NACK
  - No NACKs in the good case (no failures, post-GST)
- Missing formal leader block in non-quiescent state
  - Maybe the leader is correct but doesn't know the 3rd round of the previous wave advanced
  - Wait 2Δ, then send inform-leader block to let it know the round advanced

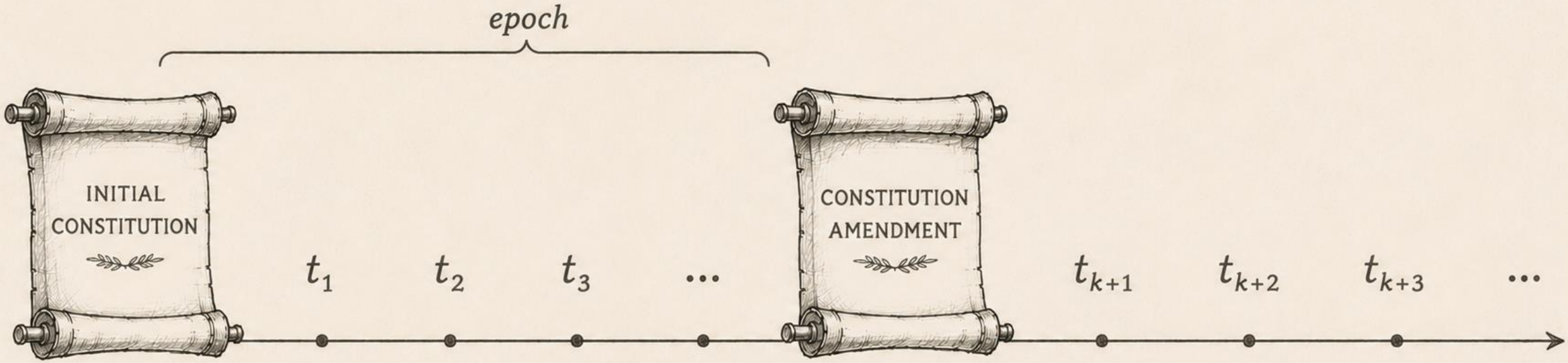
Worst-case delay after GST



Part 3: Constitutional consensus for democratic governance

1. Digital democracy, Paxos, and Consensus
2. Constitutional consensus
3. **Digital democratic governance**

Constitutional consensus democratic governance



1. Rules for producing a new *valid* constitution
2. Protocol for transitioning to the next epoch

Democratic governance is based on a constitution

- In general, the initial constitution specifies
  - an initial population,
  - an initial consensus protocol,
  - a rule for amending both, and
  - a rule for amending the constitution itself.
- In our case, amendments affect (P, σ, Δ)
  - P– set of participants
  - $\frac{1}{2} \leq \sigma < 1$ – super-majority threshold (tolerating Byzantines and sybils)
  - Δ – estimated latency bound after GST

Why these three parameters?

- P– members of the digital community should be sovereign to add/remove members
- σ – reflects the level of trust in the community
  - may be higher as new members are added, and lowered as trust in new members is gained.
- Δ – highly sensitive to the composition of P.

Amendments are decided by agents

- Volitional choice, not out-of-the-blue reconfiguration
- Crucial for democratic governance
- Constitution guards against problematic changes
  - Dead sailors
  - Voting to prevent future updates

Amending P to P’

- Requires signatures of
  - A super-majority in P – relinquishing control
  - All of P’ – adding only willing members (no dead sailors)
- The people who sign the amendment
  - Would only vote to add trusted individuals
  - Would only join if they trust a super-majority of the members of P’
  - See literature on sybil resistance admission and governance

Amending σ to σ’ - *h-rule*

- Studied in democratic context (Abramowitz et al.)
  - The only rule that satisfies certain intuitive and self-evident axioms.
- Each agent in P votes for their most-preferred new value of σ
- Then σ’ is
  - the maximal σ’ > σ for which there is a σ’-supermajority among P who voted for a value greater or equal to σ’
    (e.g., to increase to ¾, need a ¾-supermajority in P to vote for at least ¾)
  - or the minimal σ’ for which there is a σ-supermajority among P who voted for a value less than or equal to σ’
  - or σ remains unchanged.

Amending Δ to Δ' - *Suppress Outer-f*

- Shahaf et al. parameter update rule
  - for single-peaked domain with potential Sybil presence
- Each agent votes for its preferred $\Delta_i \in \mathbb{R}^+$
- If the median vote > Δ: discard the maximal f votes, take the median Δ' of the remaining votes.
- If the median vote < Δ: discard the minimal f votes, take the median Δ' of the remaining votes.
- Otherwise, Δ remains unchanged.

Properties of Suppress Outer-f

- If all Sybil agents wish to increase Δ (to slow down the protocol) but the correct agents do not, Δ will not increase;
  - symmetrically for decreasing.
- If all correct agents wish to update Δ to some Δ′, the update succeeds regardless of adversarial votes, provided $f < n/3$

Democratic amendment rules summary

A constitution (P, σ, Δ) is amended democratically to (P′, σ′, Δ′), using the prevailing (P, σ, Δ) as status quo:

1. P → P ′: per-member σ-supermajority among P, plus consent of all new members in P ′ \ P
2. σ → σ′: h-rule for constitutional amendment
3. Δ → Δ′: Suppress Outer-f parameter update rule

Valid constitution amendment decisions

- In a consensus epoch, agents run an *amendment protocol*

Part 3: Constitutional consensus for democratic governance

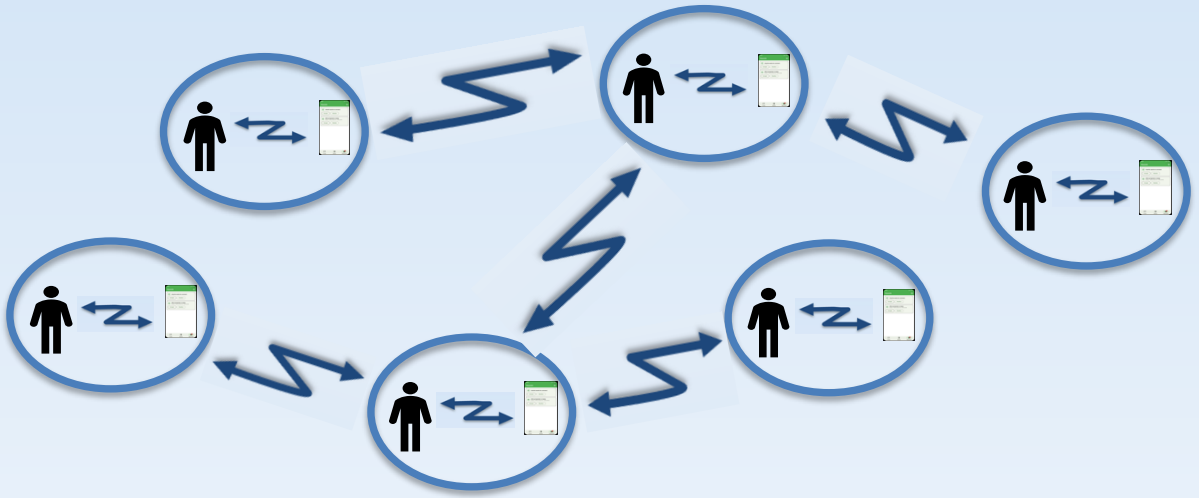
1. Digital democracy, Paxos, and Consensus
2. Constitutional consensus
3. Digital democratic governance

Questions?

Grassroots Wrap-up

Grassroots — Why and What?

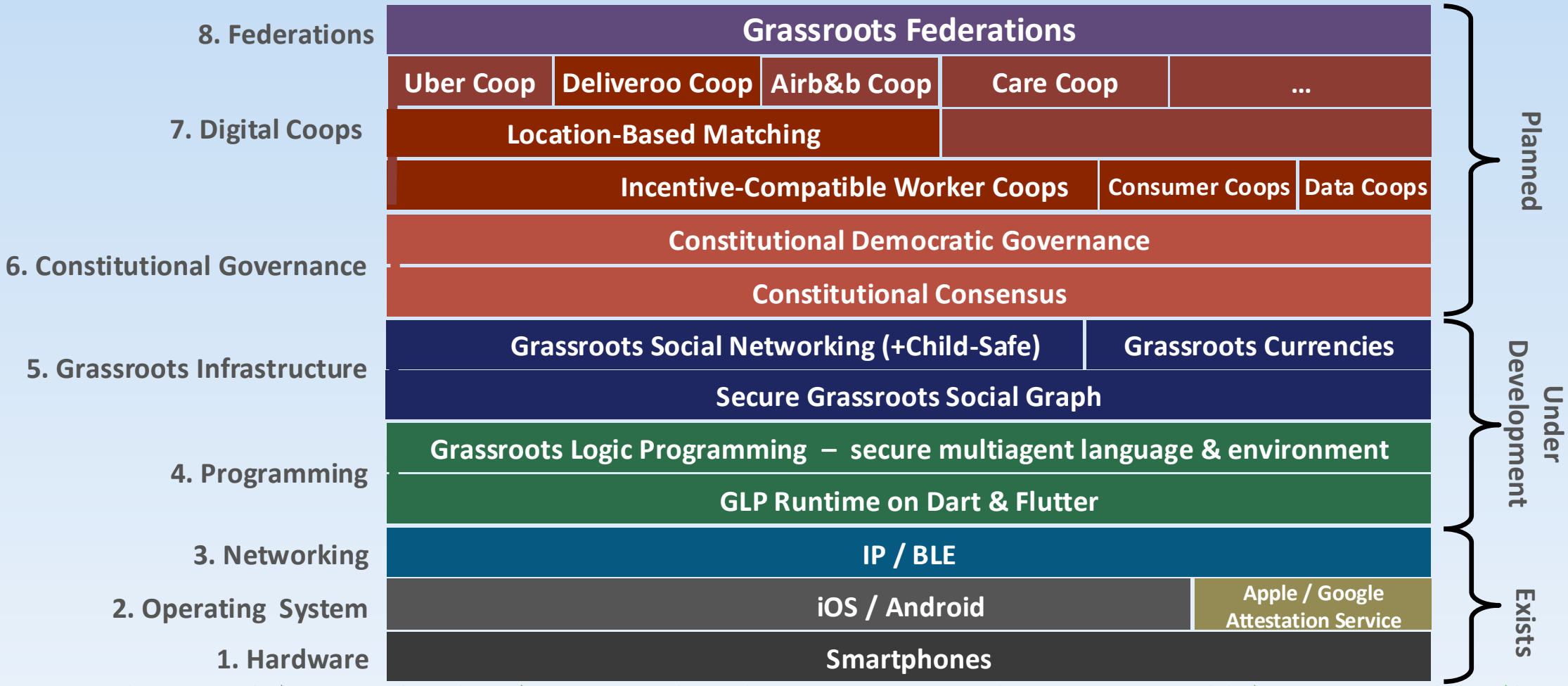
- An egalitarian & democratic alternative to global platforms (Facebook, X, Uber, Airbnb, Bitcoin, Ethereum, Amazon Marketplace,…)



Grassroots — Why and What?

- An egalitarian & democratic alternative to global platforms (Facebook, X, Uber, Airbnb, Bitcoin, Ethereum, Amazon Marketplace,…)
- How would a grassroots digital realm look like?

The Grassroots Stack



Grassroots — Why and What?

- An egalitarian & democratic alternative to global platforms (Facebook, X, Uber, Airbnb, Bitcoin, Ethereum, Amazon Marketplace,…)
- How would a grassroots digital realm look like?
- How will it be built?

Building Grassroots **Theory and Implementation** with AI

Grassroots Bonds as a Foundation for Market Liquidity

Anonymous author

Anonymous affiliation

Abstract

Global cryptocurrencies are unbacked and have high transaction cost incurred by global consensus. Grassroots cryptocurrencies are backed by the goods and services of their issuers—any person, natural or legal—and have no transaction cost beyond operating a smartphone. Liquidity in grassroots cryptocurrencies arises from mutual credit via coin exchange among issuers. However, as grassroots coins are redeemable 1-for-1 against any other grassroots coin, the credit-forming exchange must also be 1-for-1, lest prompt redemption after exchange would leave the parties with undue profit or loss. Thus, grassroots coins are incongruent with liquidity through interest-bearing credit.

Here, we introduce grassroots bonds, which extend grassroots coins with a maturity date. Bond redemption generalises coin redemption, allowing the lending of liquid coins in exchange for interest-bearing future-maturity bonds. We show that the voluntary swap of grassroots bonds can realise the full gamut of financial instruments, including loans, sale of debt, forward contracts, options, collateral, guarantees, insurance, credit default swaps, letters of credit, and credit lines. Moreover, classical liquidity ratios are applicable just as well to grassroots bonds. Grassroots bonds may thus allow local digital economies to form and grow without initial capital or external credit, harnessing mutual trust within communities into liquidity.

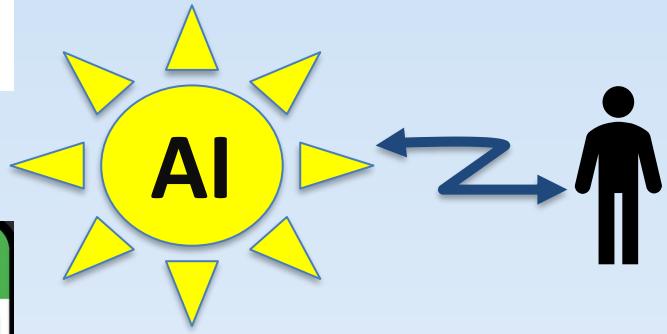
Grassroots bonds are formally specified via guarded multiagent atomic transactions and implemented in GLP, a concurrent logic programming language running on Dart for smartphone deployment. The implementation is illustrated by a running multiagent village market scenario in GLP. A secure implementation, recovering bond issuers and holders from loss of their signing key or device, is developed in a companion paper.

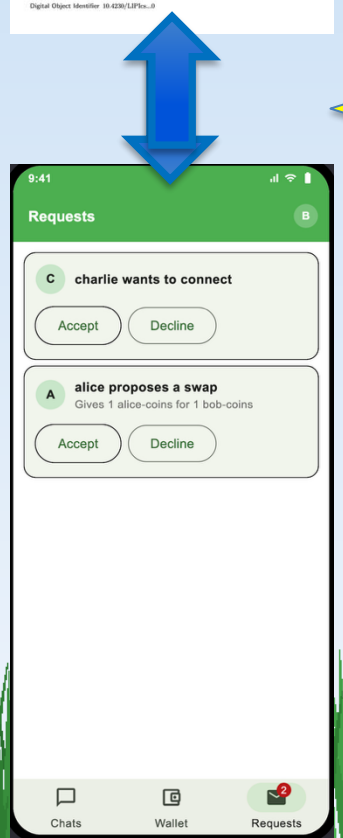
2012 ACM Subject Classification Applied computing → Digital cash; Theory of computation → Distributed computing models; Computer systems organization → Peer-to-peer architectures

Keywords and phrases Grassroots Bonds, Grassroots Cryptocurrencies, Financial Instruments, Liquidity, Guarded Multiagent Atomic Transactions, GLP

Digital Object Identifier 10.4230/LIPIcs...0

Scientific paper describing a grassroots platform





Smartphone App realizing the grassroots platform

Grassroots paper are on the arXiv

Showing 1–47 of 47 results for author: Shapiro, Ehud

Search v0.5.6 re

Shapiro, Ehud | Author(s)

◉ Show abstracts ○ Hide abstracts

50 results per page. Sort results by Relevance Go

1. arXiv:2607.02304 [pdf, ps, other] cs.DC cs.CR cs.MA

Securing People and their Machines Against Major Faults

Authors: Ohad Eitan, Idit Keidar, Ehud Shapiro

Abstract: We consider grassroots platforms -- distributed systems of agents consisting of people identified by self-chosen public keys and their machines (smartphones) -- and wish to make them secure against \emph{major faults}: the loss of their private keys and/or their smartphones. As grassroots platforms have no global resource to rely on for recovery, our peer-based solution is based on: (\ia) \emph{a... ▽ More

Submitted 2 July, 2026; **originally announced** July 2026.

2. arXiv:2606.19263 [pdf, ps, other] cs.SI cs.CY cs.MA econ.GN

Digital Speech Acts Retain Control of Copyright with People, Not Platforms

Authors: James Golike, Ehud Shapiro

Abstract: Legal precedents protect computer code as copyrightable expression. They have enabled centralized digital platforms -- operating from corporate servers that hold all user data -- to construct private governance regimes through the interaction of copyright, contract, and technical architecture: people who create virtually all platform value must surrender effective copyright control through Terms o... ▽ More

Submitted 27 June, 2026; **v1** submitted 17 June, 2026; **originally announced** June 2026.

3. arXiv:2605.13362 [pdf, ps, other] cs.MA cs.AI cs.DC cs.GT econ.TH

Constitutional Governance in Metric Spaces

Authors: Ehud Shapiro, Nimrod Talmon

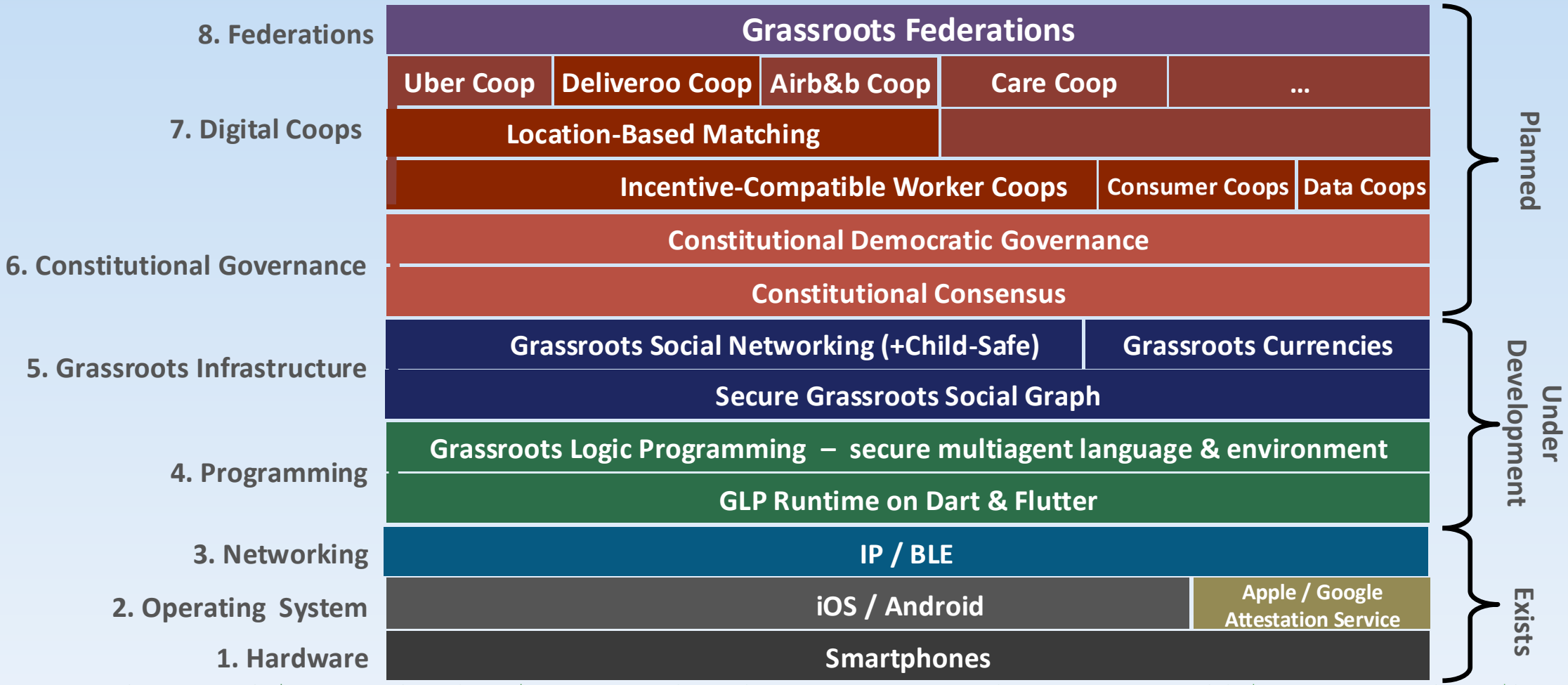
Abstract: Computational social choice and algorithmic decision theory offer rich aggregation theory but no comprehensive process for egalitarian self-governance: aggregation, deliberation, amendment, and consensus are each considered in isolation, with key metric-space aggregators being NP-hard. Here, we propose constitutional governance in metric spaces, integrating these stages into a coherent polynomial-... ▽ More

Submitted 14 May, 2026; **v1** submitted 13 May, 2026; **originally announced** May 2026.

4. arXiv:2604.25596 [pdf, ps, other] cs.DC cs.HC cs.MA cs.SI

Volitional Multiagent Atomic Transactions: Describing People and their Machines

Questions?


8. Federations
Grassroots Federations
Uber Coop
Deliveroo Coop
Airb&b Coop
Care Coop
...
7. Digital Coops
Location-Based Matching
Incentive-Compatible Worker Coops
Consumer Coops
Data Coops
6. Constitutional Governance
Constitutional Democratic Governance
Constitutional Consensus
5. Grassroots Infrastructure
Grassroots Social Networking (+Child-Safe)
Grassroots Currencies
Secure Grassroots Social Graph
4. Programming
Grassroots Logic Programming – secure multiagent language & environment
GLP Runtime on Dart & Flutter
3. Networking
IP / BLE
2. Operating System
iOS / Android
Apple / Google Attestation Service
1. Hardware
Smartphones
Planned
Under Development
Exists

2. Grassroots, by comparison

… of cardinality of essential agents:

How many agents need to be eliminated to disable communication?

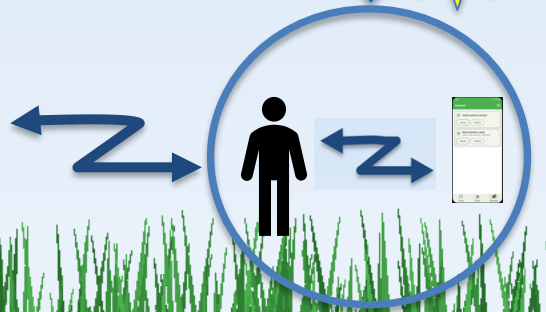
Class	Essential Agents	Cardinality	Canonical Example
Centralised	The server	One	Facebook
Decentralised	Bootstrap nodes	Finite, >1	Bitcoin
Federated	All servers, or all clients	Infinite, but not universal	Mastodon
Grassroots	All (but one ☺)	Universal	Scuttlebutt

Today: Implementing Grassroots with AI

Scientific paper describing a grassroots platform

GLP program realizing the grassroots platform

Dart/Flutter implementation of GLP



Grassroots App on iPhone/Android phones