SORA Nexus

powered by

HYPERLEDGER IROHA

Hyperledger Iroha 3: SORA Nexus

One World. One Economy. One Ledger.

Hyperledger Iroha Maintainers

2025-11-26 • Draft for Review

Contents

1 Abstract

SORA Nexus, built on Hyperledger Iroha 3, is a single logical ledger composed of cooperating validators and data spaces, designed to preserve privacy, determinism, and interoperability. It seamlessly combines private data spaces (for regulated applications) with public domains (for open innovation), and ensures that policies are enforced through governed configurations rather than ad hoc deployments. It also integrates modern cryptographic proofs and data-availability mechanisms so the system can scale without sacrificing correctness. This paper outlines SORA Nexus's goals, architecture, cryptography, execution environment, operations model, and roadmap for deploying sovereign or cross-border networks on a unified ledger.

2 Executive Summary

This whitepaper proceeds from first principles (explaining why a bespoke virtual machine and data-space ledger are needed). It then covers the architecture, cryptographic proofs, economics, and user experience. The goal is to present SORA Nexus as a cohesive system rather than a collection of disjoint features. Key themes include:

- **One World, One Economy, One Ledger**: a single global SORA Nexus built on Hyperledger Iroha 3, targeting 1 s finality, providing data sovereignty, and incorporating post-quantum security, no parallel chains needed.
- **Current problems**: public blockchains leak data and lack predictable performance; private silos lose composability; regulatory and audit requirements clash with openness; and atomicity across applications/layers is missing.
- **Solution pillars**: lanes plus a merge ledger yield one canonical order; data spaces serve as sovereign zones; IVM and Kotodama ensure deterministic execution; FASTPQ zk-STARK proofs enable compliance without revealing private books; a data availability layer provides horizontal scaling; ISO 20022 integration and Norito streaming facilitate interoperability.
- **Why now**: more than 130 central banks are exploring CBDCs, and fragmentation risk is rising; SORA Nexus builds on SORAMITSU's experience with Bakong (>$150B 2024 volume) and multiple CBDC pilots [1].
- **Performance**: FASTPQ proofs verify in <100 ms; lanes achieve 1 s finality; the data availability layer ensures recoverability without leaking private payloads.
- **Competitive posture**: sovereign control combined with global composability, no need to fork new chains, achieving privacy and openness together.
- **Single logical ledger**: organizes all activity into data spaces that share a unified address space while isolating each domain's privacy and performance. Each data space comes with explicit attestation and routing policies.

- **IVM-native runtime**: a custom Iroha Virtual Machine (IVM) runs Kotodama bytecode deterministically, using pointer-ABI types and Norito envelopes. This avoids the nondeterminism, reentrancy pitfalls, and legacy opcode baggage of Ethereum's EVM.
- **Deterministic BFT finality**: the SUMERAGI consensus pipeline achieves sub-second finality with no empty blocks, merging parallel lanes into one canonical chain. Commit windows are bounded (typically two slots).
- **Programmable and safe**: Kotodama smart contracts compile to deterministic IVM bytecode. Pointer-ABI types and Norito serialization ensure inputs are well-formed for both host functions and contracts.
- **Scalable data availability**: erasure-coded Kura/WSV storage, lane-level proofs, and DA certificates keep blocks verifiable while capping per-slot verification via deterministic budgets.
- **XOR-powered economics**: fees, staking rewards, and governance bonds use the SORA (XOR) token, giving builders, validators, and attesters a unified incentive model across both public and private data spaces.
- **Always-on payments**: supports exchanges of value even between offline devices via pre-signed vouchers, with deterministic reconciliation once connectivity returns.
- **Governance-first**: network parameters live in the `iroha_config` file; cryptography, admission rules, and scheduling are versioned and auditable. Post-quantum ready primitives (e.g., ML-DSA) are built in for certificates and quorum signatures.
- **Interoperability**: ISO 20022-aligned messaging, Norito serialization, and connectors let SORA Nexus bridge domestic rails, partner ledgers, and analytics systems without sacrificing policy control.

2.1 SORA Nexus Highlights

One World, One Economy, One Ledger: SORA Nexus uses Hyperledger Iroha 3 as a single global ledger (no parallel chains needed), targeting 1 s finality and providing data sovereignty and post-quantum security.

- **Current problems**: public blockchains leak data and lack predictable performance; private silos lose composability; regulatory and audit needs clash with open networks; and atomicity across applications/layers is missing.
- **Our solution**: one unified ledger with parallel lanes and a merge ledger for a single order; data spaces act as sovereign zones; FASTPQ zk-STARKs prove compliance without revealing books; effectively infinite scalability via the custom IVM, horizontal lanes, and a robust DA layer.

- **Why now**: roughly 90–94% of central banks are exploring CBDCs, raising fragmentation risks. SORA Nexus builds on lessons learned from Bakong (>$150B 2024 volume) and multiple CBDC pilots, positioning it for sovereign-grade deployment.
- **FASTPQ performance**: each data space has zero-knowledge proofs enforcing conservation and policy; 10k proofs/sec can be generated, and each verifies in <100 ms on consumer hardware; cryptography is post-quantum secure (ML-DSA-87, Poseidon2).
- **Competitive posture**: combines sovereign control with global composability, eliminating the trade-off between privacy and openness (unlike today's fragmented public vs. permissioned approaches).

3 SORA Nexus: The End of History

SORA Nexus is the single logical ledger that can process all of the transactions of the Universe. This claim is grounded in how it scales and evolves, not just in branding. The design ties throughput, latency, and growth directly to concrete mechanisms in the stack:

- **Lanes and merge ledger.** Parallel lanes execute disjoint data-space workloads. The merge ledger orders only lane tips and never rewrites lane history. Adding new lanes increases aggregate throughput while leaving each lane's validation cost, state layout, and client semantics unchanged.
- **Deterministic budgets and scheduling.** Start-Time Fair Queuing (SFQ) bounds the transfer-equivalent units (TEU) each lane and data space can use. Circuit breakers automatically cap verification and DA workload when needed. Lanes fuse or split according to load, keeping latency low under light traffic and scaling out horizontally when demand rises.
- **Efficient proofs and data availability.** FASTPQ zk-STARK proofs keep verification work quasi-logarithmic in trace size, and DA sampling over erasure-coded shards bounds the cost of data recovery. Both proof and DA parameters are governed, making scaling behavior explicit, observable, and tunable rather than emergent.
- **Deterministic runtime.** The Iroha Virtual Machine (IVM) removes execution-side nondeterminism: a fixed pointer ABI, Norito envelopes, explicit syscalls, and the absence of reentrancy races ensure that adding validators or changing hardware backends cannot alter state transition outcomes.
- **Governed evolution instead of new chains.** Governance controls configuration, data-space creation, and runtime upgrades via manifests and hash commitments. The ledger grows by adding data spaces, lanes, and parameter sets on the same network rather than spawning new chains.

With lanes, proofs, and DA providing horizontal scale under deterministic bounds, SORA Nexus can host both sovereign and open domains on a single network instead of requiring

separate ledgers per use case. In this sense, the "end-of-history" claim is an architectural statement: once these mechanisms are in place, adding capacity or new jurisdictions becomes a matter of configuration, not new infrastructure.

4 Context and Goals

We combine a purpose-built VM (IVM), a data-space ledger architecture, and hardware-agnostic proofs so that regulated and open domains can share a single deterministic computation model. The following sections build on this foundation, covering consensus and scheduling, data availability, privacy proofs, economics, and user experience.

4.1 Motivation

Hyperledger Iroha 3 was designed to meet requirements from central banks, financial market infrastructures, and open ecosystems, such as the need for predictable finality, verifiable execution, fine-grained policy control, and safe programmability. SORA Nexus addresses these requirements by unifying the ledger into distinct data spaces that can interoperate (when policy allows) or remain isolated (when necessary).

4.2 Design Goals

- **Determinism across hardware**: The system must produce identical results across different hardware environments. Whether validators run on commodity CPUs or use acceleration (SIMD, Metal, CUDA), the consensus and execution outcomes must match exactly, with a deterministic scalar fallback for consistency.
- **Privacy by construction**: Private data never leaves its designated data space, while data in public spaces is auditable through bounded data-availability sampling. Any cross-space calls or interactions require explicit permission, enforcing privacy by design.
- **Performance isolation**: The throughput of private domains should not be impacted by public traffic. Mechanisms like TEU budgets, start-time fair queuing (SFQ), and "must-serve" scheduling slices ensure each domain gets fair access to block capacity, guaranteeing transaction inclusion within bounded slots.
- **Operational resilience**: The system should remain available and efficient under varying conditions. Stateless gateway design, replicated consensus clusters, and the ability for lanes to fuse (merge) when load is low help reduce latency and operating cost while ensuring high availability.
- **Governable evolution**: All critical parameters (cryptographic primitives, pointer-ABI tables, syscall lists, admission policies, etc.) are versioned and anchored in configuration. Upgrades must occur through explicit governance processes rather than ad hoc code changes, ensuring controlled evolution.

5 IVM at the Core

The Iroha Virtual Machine (IVM) is purpose-built for SORA Nexus and replaces generic EVM-style runtimes. It executes Kotodama bytecode (‘.to‘) with strict determinism:

- **Wide opcodes and pointer ABI**: IVM uses its own 16-bit and 32-bit instruction encodings and stable pointer types for assets, accounts, and data spaces. All inputs are provided as Norito TLV envelopes; any malformed pointer causes a deterministic trap (error).
- **Registers and decoding**: The IVM provides 256 64-bit general-purpose registers (‘r0‘–‘r255‘), with ‘r0‘ always fixed to zero. Instruction streams can contain a mix of 16-bit and 32-bit opcodes. The IVM decoder recognizes only its own instruction formats. Compressed 16-bit opcodes typically map to common arithmetic or branching operations, whereas the 32-bit opcodes carry larger fields (5 or 9-bit register indices, syscall identifiers, or immediates).
- **State transition via ISIs**: All ledger changes are expressed through Iroha Special Instructions (ISIs). When the IVM executes a Kotodama smart contract, it emits a sequence of ISIs, and validators apply those ISIs to the World State View (WSV) and Kura storage in the same slot, once the transaction passes admission checks and proof verification. The ISI format is deterministic and Norito-encoded, ensuring that all nodes can exactly reproduce state transitions.
- **Deterministic safety**: The IVM ensures deterministic execution by design: it disallows floating-point arithmetic, nondeterministic system calls, and any hidden side effects (such as implicit gas consumption). Classic EVM pitfalls like reentrancy and other quirks are eliminated. All syscalls in IVM are explicit and versioned to maintain consistency.
- **Cross-domain composability**: By using pointer-ABI types, contracts can safely pass references across data spaces (when permitted by policy) without exposing raw memory. This means smart contracts can interoperate across domains without having to copy or interpret ambiguous data blobs, eliminating a class of cross-contract miscommunication errors.
- **Performance headroom**: The IVM allows optional use of vector/SIMD instructions or GPU acceleration to improve execution speed. However, any accelerated execution must produce bit-for-bit identical results to the standard scalar execution, ensuring that using faster hardware never changes consensus outcomes.
- **Auditability**: The IVM supports generating Merkle commitments over its memory and register state, and produces Norito-encoded execution receipts. This makes contract execution fully reproducible and auditable after the fact, in contrast to the opaque execution traces of EVM.

By making the IVM a first-class component, SORA Nexus is free from Ethereum's opcode legacy and avoids external compiler dependency risks. The platform can evolve its syscall interfaces and pointer types through governance without breaking determinism or interoperability.

6 Architecture Overview

We outline the architecture from the top-level network components (validators, gateways, data spaces) down to the execution core (IVM) and its supporting proof systems. Each layer provides a well-defined interface so that governance can evolve components independently while keeping the system deterministic intact.

6.1 Components and Roles

- **Validators**: Run the SUMERAGI BFT consensus, execute IVM smart contracts, verify lane proofs, and store the ledger (Kura blocks and the World State View).
- **Gateways**: Stateless entry points that authenticate clients, enforce policy (roles, fees, quotas), and route transactions to the appropriate data space and lane.
- **Provers and attesters**: Generate cryptographic proofs for each data space (FASTPQ proofs of ledger integrity) and collect data availability certificates for public data spaces.
- **Space Directory**: A registry mapping each data space ID to its routing information, governance owner, and admission policy.

6.2 Data Spaces

Data spaces are first-class partitions of the ledger, each with explicit privacy and routing policies. A private data space keeps its data and proofs confined to authorized participants. In a public data space, anyone can participate, but the same deterministic admission rules apply. Each data space produces cryptographic artifacts (such as state root hashes, data availability commitments, and attestation certificate hashes) that are collected into transaction envelopes and finalized by the global consensus.

6.3 Lanes and Merge Ledger

SORA Nexus uses multiple parallel lanes to scale throughput. Each lane produces blocks containing that lane's data space artifacts and lane-level proofs. A lightweight merge ledger then combines the tip of each lane into one single canonical block sequence. When load is low, lanes may deterministically fuse into one to minimize latency; as demand

increases, they split back into multiple lanes, without changing the observable behavior of the network.

7 Data Space Model and Governance

7.1 Private and Public Domains

- **Private DS**: A permissioned data space where data never leaves the domain. Attestation certificates (signed with ML-DSA-87) include the authorized validator set and any metadata needed for compliance and audits.
- **Public DS**: An open-participation data space with in-slot data availability sampling. Each block's certificate contains the Ed25519 signatures of the attester set (with reserved fields for future post-quantum audit data).

7.2 Governance Surfaces

- **Configuration**: All runtime behavior in production is defined by the `iroha_config` file; environment flags are only used for testing.
- **Parameter sets**: Cryptographic algorithms, scheduling budgets, and transaction admission caps are versioned and published on-chain. Validators will reject any parameter change that is not recognized or properly authorized.
- **Attestations**: Creating a new data space requires posting a bond and issuing a signed attestation certificate. If a transaction refers to an unknown data space ID (DSID) or an attestation is missing, it will be rejected according to policy.

8 Consensus, Scheduling, and Finality

8.1 SUMERAGI Pipeline

SUMERAGI is a Byzantine Fault Tolerant (BFT) consensus pipeline with deterministic leader rotation. Blocks are produced only when there are transactions ready (idle slots remain empty). Under typical conditions, each lane achieves finality in about one second, with commit windows bounded by $\Delta \leq 2$ slots. Quorum certificates (QCs) are post-quantum secure, using ML-DSA-87 signatures.

8.2 Admission and Fairness

Work is measured in transfer-equivalent units (TEU). A default lane capacity is around 20,000 TEU per second, and fairness is enforced by:

- **Start-time Fair Queuing**: Each data space (DS) is assigned virtual start and finish times to schedule its transaction envelopes, respecting that DS's guaranteed floor share and cap.
- **Must-serve slice**: Any data space with pending transactions is guaranteed to have one of its transactions included within a bounded number of slots (default $S = 120$), as long as those transactions fit within block size and data availability limits.
- **Circuit breakers**: If the system detects that certain metrics exceed safe thresholds (e.g., finality delays, oversized blocks, too many DA failures, or excessive CPU usage for verification), it automatically lowers per-DS throughput caps or block sizes to protect network health. These adjustments have built-in hysteresis to avoid oscillation once conditions normalize.

8.3 Lane Fusion and Split

Lane fusion keeps latency low during quiet periods; lane split restores parallelism under load. Policy parameters (fusion floor, exit threshold) are governed, and the transitions are deterministic so peers stay in sync.

- **Fusion trigger**: If the total transaction load stays below a threshold λ_{floor} for two consecutive slots, all lanes merge into a single Sumeragi pipeline (separately for public vs. private lanes as appropriate). Routing remains per data space; admission and proof processes are unchanged.
- **Split trigger**: If demand goes above λ_{exit}, the single lane splits back into K parallel lanes again, with a deterministic assignment of data spaces to lanes. This process does not reorder any transactions; any envelopes already finalized remain final, and new transactions are simply processed in parallel lanes going forward.
- **Deterministic mapping**: A fixed hashing scheme on (DSID $\|$ slot) maps each data space's work to a lane. The mapping remains the same before and after fusion, so all nodes agree on placement. Fusion/split only changes how many data space hashes map to each live lane.
- **State handling**: The underlying ledger state (both the world state view and the Kura block store) remains continuous through lane fusions and splits; there is no re-sharding or movement of historical data. Provers and data attesters simply adjust their per-slot work budgets (for example, $q_{\text{in\_slot\_per\_ds}}$) according to how many lanes are active, but the overall data availability and proof coverage guarantees remain the same.
- **Operator impact**: When lanes fuse into one, the consensus process involves fewer communication hops (reducing vote hops and network chatter) and thus reduces network overhead. When lanes split, throughput increases due to parallel processing. The system provides telemetry that shows the current number of lanes and the thresholds, allowing operators to monitor when the network changes mode.

8.4 Sumeragi Consensus: Safety and Liveness

Sumeragi is a BFT pipeline with deterministic leader rotation and two commit phases (propose, aggregate, commit). For each lane:

- **Committee**: size $N = 3f + 1$; quorum certificate (QC) requires $|\text{QC}| \geq 2f + 1$ distinct validators.
- **Assumptions**: up to f Byzantine faults, authenticated links, and partial synchrony after GST (Global Stabilization Time). Honest validators sign at most one proposal per height and slot, and follow the earliest valid proposal for that slot.
- **Rotation**: leaders are chosen by a deterministic PRF over the epoch seed, height, and view. The topology computes

  $$L_{h,v} = H_{\text{blake2b}}(\text{seed} \parallel h \parallel v) \bmod N,$$

  then rotates the peer list so index 0 is the leader. The seed comes from the prior epoch's beacon (VRF aggregate), so all nodes derive the same leader per height/view while making the order pseudorandom and grinding-resistant.
- **Pipeline**: *Proposal* (leader builds envelope with DA audit roots and proofs) $\rightarrow$ *Validate & Vote* (validators check signatures/proofs/size; sign QC) $\rightarrow$ *Commit* (upon QC, apply to WSV and persist Kura) $\rightarrow$ *Broadcast receipt.*
- **Leader failure**: if no valid proposal/QC by timeout T, the slot is skipped; next leader proceeds. Timeout is set $> 2\Delta$ to accommodate network delay.
- **Locking rule (as implemented)**: each node tracks *locked_qc* (height/view/hash). A new proposal must carry a *highest_qc* that extends the locked chain: height must be $\geq$ locked height, and if heights match, hashes must match. Ancestry is checked via Kura/WSV lookups. Locked QC updates monotonically when higher QCs form; divergent highest_qc are rejected before voting. This mirrors HotStuff-style locking to prevent regressions.
- **NEW_VIEW gating**: leaders only propose after collecting a NEW_VIEW quorum with a highest_qc that extends locked_qc; otherwise the slot is skipped.

Lemma 1 (Locked QC monotonicity)**.** *The node only updates locked_qc when a candidate (h, v) satisfies $(h, v) > (h_{cur}, v_{cur})$; otherwise the prior tuple remains. Thus locked height/view never decrease.*

Proof. The implementation uses a monotone compare before storing (height, view); lower tuples are ignored. Reset to $(0, 0)$ is the only downward move and represents genesis/init. ■

Lemma 2 (Ancestry enforcement)**.** *Before proposing, the leader enforces two checks: (i)* `ensure_locked_qc_allows` *rejects any* `highest_qc` *whose height is below the locked*

height or whose hash differs at the same height; (ii) ***highest_qc_extends_locked*** *walks parent hashes via pending blocks and Kura to ensure the chosen* ***highest_qc*** *is a descendant of the locked block.*

Proof. Step (i) is a direct comparison on heights/hashes; step (ii) uses `qc_extends_locked_with_lookup` to iteratively follow parent hashes until the locked height is reached, failing if a parent is missing or diverges. Only if both checks pass does the leader advance the slot. ■

Safety (permissioned Sumeragi) Let V be validators, $|V| = 3f+1$. A block commits only with a QC of size at least $2f + 1$.

Lemma 3 (QC intersection)**.** *For any two QCs Q_1, Q_2 over V, $|Q_1 \cap Q_2| \geq f + 1$.*

Proof. By set algebra, $|Q_1 \cap Q_2| \geq |Q_1| + |Q_2| - |V| \geq (2f + 1) + (2f + 1) - (3f + 1) = f + 1$. ■

Theorem 1 (Safety)**.** *Two conflicting blocks cannot both commit at the same height.*

Proof. Suppose conflicting blocks B, B' commit with QCs $Q_B, Q_{B'}$. By the lemma, $Q_B \cap Q_{B'}$ has at least $f + 1$ validators. At most f are Byzantine, so at least one honest validator would have to sign both, violating the single-signing rule. By the ancestry lemma, honest leaders only propose blocks whose highest_qc descends from the locked QC; the monotonic locked QC lemma prevents rollback to a lower view/height. Therefore honest validators cannot be led to conflicting commits, and no two divergent blocks can both commit. ■

Liveness (permissioned Sumeragi) After GST, message delay is bounded by Δ; slot timeout $T > 2\Delta$.

Lemma 4 (Honest leader window (PRF selection))**.** *With PRF/VRF-based leader selection over $N = 3f + 1$ validators (at most f Byzantine), the probability that a window of k consecutive slots contains no honest leader is at most $\left(\frac{f}{N}\right)^k$. For $k = f + 1$ this bound is $\left(\frac{f}{3f+1}\right)^{f+1}$, which vanishes quickly as N grows.*

Proof. Each slot samples a leader index as $L_{h,v} = H(\text{seed} \parallel h \parallel v) \bmod N$. Assuming the PRF output is uniformly distributed and unpredictable, the chance a given slot picks a Byzantine leader is $\frac{f}{N}$. Treating slots as independent draws (the standard VRF sampling assumption), the probability all k slots in a window are Byzantine is $(\frac{f}{N})^k$. The complement gives the stated bound that at least one slot in the window has an honest leader. ■

Lemma 5 (Honest slot commits)**.** *In a slot with an honest leader, if no conflicting QC exists at that height, the block commits within the slot.*

Proof. The honest leader proposes a valid block. All honest validators receive it within Δ and vote; at least $2f + 1$ votes arrive at the leader within another Δ. Since $T > 2\Delta$, the leader forms a QC before timeout and broadcasts it; honest validators commit on receipt. By safety and locking, no conflicting QC can form in the same height that extends a different chain. ■

Theorem 2 (Liveness (with overwhelming probability))**.** *With PRF/VRF leader selection and $T > 2\Delta$ post-GST, blocks from honest leaders commit infinitely often except with negligible probability.*

Proof. Work in the period after GST, when message delay is bounded by Δ and the timeout satisfies $T > 2\Delta$.

Consider any window of $f + 1$ consecutive slots. By the Honest leader window lemma, the probability that all slots are Byzantine is at most $\left(\frac{f}{3f+1}\right)^{f+1}$, so with overwhelming probability the window contains an honest leader. In such a slot, if no conflicting QC exists at that height, the Honest slot commits lemma implies the leader's block commits within the slot.

By safety and the locking rule, committed blocks extend the current chain, and locked QCs never roll back to a lower height/view. Thus once a block has committed in some slot, all subsequent honest leaders build on its descendants.

Because time is unbounded and slots advance monotonically, we can slide the $f + 1$-slot window forward indefinitely. Each window contains at least one slot with an honest leader except with negligible probability, and each such slot can commit a new descendant block. Hence infinitely many blocks from honest leaders eventually commit. ■

Sumeragi NPoS variant When committees are drawn from staked validators via Verifiable Random Functions (VRFs):

- **Selection**: committee C_s for slot s is VRF-sampled with expected size N; honest stake fraction $\alpha > 2/3$ yields Pr[honest majority] $\geq 1 - e^{-\kappa}$ for security parameter κ (Chernoff bound over honest stake).
- **Sortition details**: each validator draws $y = \mathrm{VRF}_{sk}(s)$; if $y < \tau \cdot \mathrm{stake}_v / \mathrm{stake}_{\mathrm{total}}$ it joins C_s. Threshold τ tunes expected committee size.
- **Stake-weighted ties**: if multiple proposals appear (e.g., due to faulty leader), validators prefer the highest-stake valid proposal for the slot before voting, preserving single-signing safety.
- **Slashing**: double-signing (conflicting votes/proposals at same height/slot) can be proven on-chain; NPoS governance may slash offending stake and evict keys from the VRF roster.
- **Quorum**: QC still requires $\geq 2f_s + 1$ where f_s is the faulty count within C_s; safety holds by the same intersection argument conditioned on an honest majority event.

- **Liveness**: With per-slot honest-majority probability $p \geq 1 - e^{-\kappa}$, the probability that no honest-majority slot occurs in k slots is $(1-p)^k$; choose k so $(1-p)^k$ is negligible to obtain high-probability eventual commit. Conditioned on an honest-majority slot, the permissioned safety/liveness lemmas apply.

Theorem 3 (Safety, NPoS)**.** *Conditioned on an honest-majority committee in slot s, no conflicting blocks at height h can both commit.*

Proof. Given honest majority in C_s, at most $f_s < |C_s|/3$ are Byzantine. Any two QCs at height h over C_s intersect in at least $f_s + 1$ validators (by the QC intersection lemma with f replaced by f_s). Honest validators single-sign and the locking rule still applies (highest_qc must extend locked_qc), preventing divergent ancestry. Unconditioning over slots, the failure probability is bounded by the probability of a faulty-majority slot, which is $\leq e^{-\kappa}$ per slot by the Chernoff bound. ■

Theorem 4 (Liveness, NPoS)**.** *With per-slot honest-majority probability $p \geq 1 - e^{-\kappa}$ and timeout $T > 2\Delta$, a block commits with probability $1 - (1-p)^k$ within any window of k slots; choosing k such that $(1-p)^k$ is negligible yields high-probability eventual commits.*

Proof. Assuming VRF draws are unpredictable/unbiasable and per-slot committees are independent, each slot has an honest-majority committee with probability at least p. In such a slot, the honest leader or the highest-stake valid proposal (if leadership is faulty) can gather $2f_s + 1$ votes within 2Δ and commit (by the Honest slot commits lemma adapted to C_s). The chance of no honest-majority slot in k tries is $(1-p)^k$; complement gives the stated bound. ■

Evidence horizon and view changes Equivocation proofs (e.g., double-signs) are only accepted within a bounded evidence horizon measured in blocks; stale evidence is ignored to prevent unbounded storage and grinding. View-change proofs must respect the locked QC ancestry checks before proposing; NEW_VIEW quorum selection is deterministic, and stale proofs are rejected. These guards ensure liveness under reconfiguring validators while maintaining safety.

Cross-lane safety Lanes are independent; the merge ledger only orders lane tips and never rewrites lane history. Since each lane's safety holds, the merged order is safe provided the merge QC also uses $2f + 1$ over the merge committee with the same intersection logic.

9 Data Availability and Storage

Execution on the IVM is paired with a dedicated data availability (DA) layer that ensures both scalability and privacy. Techniques like erasure-coded shards, DA sampling, and zero-knowledge binding proofs allow validators to prove that all data is available and correct without exposing sensitive private payloads, while keeping verification time budgets predictable.

9.1 Erasure-Coded Kura and WSV

Kura serves as the block storage (history) and WSV represents the current state. Both are designed to work with erasure-coded data shards, allowing the ledger to scale to very long histories without forcing every node to store everything. Erasure coding also preserves locality: data specific to a private data space can remain local to the authorized nodes, but those nodes can still reconstruct full blocks from the shares they hold, ensuring no private payloads leak to unauthorized parties.

9.2 Two-Dimensional Erasure Coding with ZK DA Proofs

SoraFS uses a two-dimensional erasure coding scheme: data shards are arranged in rows and columns such that any sufficiently large subset can reconstruct the original payload. This layout lets validators and auditors recover block envelopes even if some shards are missing, while keeping private data spaces' payloads encrypted or entirely absent from public shards. Each set of shards comes with zero-knowledge proofs that the erasure-coded pieces correspond exactly to the original data committed in the block (i.e., they match the Merkle root and DA audit roots included in the block header). Attester nodes (specialized validators for DA) randomly sample a number of shards, verify these ZK proofs, and sign DA certificates; the lane committees enforce that the sampled proofs validate before finality. The result is a DA layer that is both bandwidth-efficient and verifiable without revealing private contents.

Example configuration: splitting an 8 MB envelope into 32 data shards and 16 parity shards (using Reed–Solomon coding) yields 48 shards of about 1 MB each; any 32 shards can reconstruct the envelope. With $q_{\text{in\_slot\_total}} = 2048$ shard samples across lanes, data availability checks finish within the 300 ms window while keeping overhead around 1.5x.

9.3 DA Certificates and Sampling

- **Public DS**: Each block envelope carries a lane-level DA audit root plus a small sample of Ed25519 signatures (e.g., $q_{\text{in\_slot\_per\_ds}} = 8$) that are verified within the DA deadline (target 300 ms). Full threshold certificates are available off-path for additional assurance.
- **Private DS**: Data availability is handled internally within the domain via ML-DSA-87 attestations; artifacts are included directly (no sampling deferrals required in a permissioned setting).
- **Envelope sizing**: Block proposers attempt to keep each envelope within a typical 16 MB size (hard cap 32 MB). If adding more transactions would exceed these limits, they use micro-segmentation or defer some data spaces' work to later blocks, thereby respecting the budgets.

9.4 Proofs

FASTPQ-ISI proofs are generated per data space and aggregated into at most two lane-level proofs (public and private). Committees verify lane proofs within the per-slot budget (100–200 ms). Prover pipelines can use GPU acceleration but must remain bit-for-bit identical to scalar paths.

10 Cryptography and Serialization

10.1 Canonical Primitives

- **Hashing**: Poseidon2 (over the Goldilocks field) is used for internal traces and sparse Merkle trees; standard SHA-2/3 is used where required for compatibility (e.g., in transcript hashing).
- **Proof system**: A STARK proving system based on DEEP-FRI (with arity 8 or 16 and blow-up factor 8 or 16), targeting at least 128-bit security.
- **Signatures**: End-user keys use the Iroha crypto library defaults (Ed25519 for most, secp256k1 for legacy, and ML-DSA variants when enabled). Consensus artifacts and private data space attestations use the post-quantum ML-DSA-87. Public data availability samples are signed with Ed25519 (with reserved fields for future PQ attachments).
- **National suites**: Chinese SM2/SM3/SM4 and TC26 GOST R 34.10-2012 (256/512) suites are available. Hosts expose SM syscalls and GOST curve IDs only when governance/config allows them (e.g., `allowed_signing` / `allowed_curve_ids`), while the control plane keeps Ed25519 as the default so networks can admit these suites without changing consensus keys.

10.2 Serialization

Norito is the native serialization codec for all ledger data. Both binary and JSON representations of data are derived from Norito definitions (with dedicated helpers). Alternative serialization schemes like SCALE are not used in production. Each Norito payload begins with a versioned header to indicate its schema version. Blocks are persisted and distributed in a standardized format (SignedBlockWire), which includes a version byte followed by the Norito-encoded block data. This ensures that all nodes interpret the data consistently and that upgrades to data formats are managed in a controlled manner.

11 Smart Contracts and Execution

11.1 Kotodama to IVM

Kotodama smart contracts compile to IVM bytecode (files with extension '.to'). The IVM defines its own 16-bit and 32-bit opcode formats. Execution forbids floating-point operations and any nondeterministic syscalls, and hardware acceleration (SIMD/GPU) is allowed only if it produces bit-for-bit identical results (i.e., acts as a deterministic fallback). Kotodama bytecode includes:

- **Headers**: A magic constant (`IVM`), the `abi_version`, feature flags (`feature_bits`), vector table length (`vector_len`), a `max_cycles` limit (to bound execution time), and program metadata (e.g., required data spaces or permissions).
- **Pointer ABI**: Typed handles (e.g., AccountId, AssetDefinitionId, DataSpaceId) into the input TLV region. All host calls reference these pointer types rather than raw memory.
- **Gas**: Execution is bounded by the declared `max_cycles`. Each instruction and syscall has a fixed, deterministic gas cost. There are no gas refunds or other dynamic gas side effects.
- **Determinism**: No floating point operations, no nondeterministic syscalls. Any out-of-bounds memory or stack access traps reliably. The IVM uses a mix of 16-bit and 32-bit opcodes under a strict endianness, and a single-pass decoder to enforce instruction alignment.

11.2 Why a Custom IVM Instead of EVM

- **Predictable semantics**: Eliminates common Ethereum issues like reentrancy bugs, gas-refund oddities, and hidden opcode side effects. All IVM syscalls and opcodes are versioned, explicit, and thoroughly audited.
- **Structured inputs**: Using pointer-ABI types and Norito envelopes keeps contract inputs well-defined. A malicious caller cannot craft input data that one node might decode differently than another, eliminating cross-client ambiguity.
- **Cross-DS safety**: Smart contracts can safely reference resources across different data spaces using typed pointers. The host enforces access policies based on those pointer types, rather than trying to interpret raw bytes, which prevents cross-domain misuse.
- **Future-proof cryptography**: The IVM and the SORA Nexus platform support multiple cryptographic algorithms side by side and are ready for post-quantum signatures, without requiring new opcodes for each. The design is not tied to Ethereum's specific cryptographic defaults, allowing more flexibility as cryptography evolves.

- **Deterministic performance**: Any use of specialized hardware (SIMD or GPUs) is carefully controlled. The rule is that the outputs must match the standard CPU computation exactly bit-for-bit. This prevents any differences in hardware from ever causing consensus disagreements, ensuring performance improvements never come at the cost of determinism.

11.3 Pointer ABI and Memory Model

Programs specify an `abi_version` in their headers. All pointer types in the ABI (e.g., AccountId, AssetDefinitionId, DataSpaceId) have stable 16-bit type identifiers. The contract's input data is placed in a read-only Norito TLV envelope region; outputs are written to an append-only region and are validated upon first dereference. The IVM divides memory into distinct regions (code, heap, input, output, stack) and traps on any out-of-bounds or misaligned access. It produces Merkle commitments over the register file and memory, which allows zero-knowledge tracing and compact proofs of execution. The VM also maintains a deterministic call stack and enforces instruction alignment (16-bit opcodes must align to 2-byte boundaries, 32-bit opcodes to 4-byte boundaries); any misaligned instruction fetch triggers a `E_ILLEGAL_INSTR` error.

What the ABI is The Application Binary Interface (ABI) defines the contract between compiled smart contract bytecode and the host VM. It specifies how data types (accounts, assets, buffers, etc.) are laid out in memory, how pointers or reference IDs are encoded, which syscalls are available and how their arguments and return values are passed, and how errors are reported. By fixing these rules in versioned headers and stable pointer type IDs, the IVM ensures deterministic execution across all hardware and avoids any "it works on my machine" variability, every validator interprets the same bytecode in exactly the same way.

11.4 Syscalls and Gas

In the IVM, each system call (syscall) is identified by an 8-bit number and has a fixed, deterministic gas charge. The host environment provides certain verified helper functions (e.g., Merkle path retrieval, Norito decoding) as syscalls, and rejects any call to an unknown number with a `VMError::UnknownSyscall`. Gas cost tables are tied to the contract's ABI version specified in its header, and are audited (and updated through governance) alongside pointer-ABI changes.

12 Transaction Lifecycle and Networking

1. **Submission**: A client (user or application) creates a transaction by signing a set of instructions that are encoded in the Norito format. The transaction is submitted to a

gateway node that corresponds to the data space where the transaction should execute (ensuring it follows that data space's policy).

2. **Admission**: The gateway performs initial checks on the transaction, verifying that the sender has the required roles/permissions, the transaction includes appropriate fees, and any rate limits or quotas are respected. Once the transaction passes these checks, it is placed into the Start-Time Fair Queuing (SFQ) scheduler, which allocates execution slots according to each data space's guaranteed share of resources.

3. **Execution**: The validator nodes take the transaction and execute its instructions on the IVM deterministically. During execution, each data space involved collects the necessary cryptographic proofs (such as FASTPQ proofs for conservation/compliance) and data availability samples for that transaction or block. This is done within the commit window prior to finalizing the block.

4. **Aggregation**: After execution, each data space's outputs, state changes, proof objects, etc., are bundled together. These bundles from multiple data spaces are then aggregated into a block for a particular lane. Each lane's block includes at most two aggregated proofs (one covering all public data spaces and one for all private ones in that lane) along with the data availability audit roots for the transactions included.

5. **Consensus**: The SUMERAGI consensus protocol then takes over to finalize the blocks. Each lane runs its BFT consensus to commit its block (with a Quorum Certificate for proof), and then a top-level merge process finalizes an ordering of these lane blocks into one global sequence. Once a block is finalized, validators apply all the state changes to their World State View and store the block in Kura (using erasure-coded shards for distribution). Importantly, while multiple lanes run in parallel, the merge ledger's final ordering does not change the order of transactions that were in the same lane, it only interweaves whole blocks from each lane to create the single ledger.

6. **Exposure**: After transactions are finalized, clients or other systems can retrieve cryptographic proofs and receipts of those transactions (for example, Merkle roots of the state or inclusion proofs that a transaction was in a block). External systems, such as banking infrastructure or analytics tools, can subscribe to events or data from SORA Nexus via connectors. These connectors output information in standardized formats (for example, events can be formatted according to ISO 20022 messages or as Norito-encoded payloads) so that existing systems can integrate with the blockchain without any loss of information.

13 Account Model, Encoding, and Multisig

13.1 Nexus Account Structure

Accounts are first-class types in the IVM (pointer-ABI) and are serialized as Norito objects. They have a canonical binary layout on-chain, and a human-readable form similar to Bech32 (with a specific HRP prefix and a 4-byte checksum). The binary layout is:

```
| net (1B) | flags (1B) | dsid (16B UUID) | domain_len (1B) | domain [...] |
| name_len (1B) | name [...] | multisig_threshold (1B) | signers_len (1B) |
| signer_0 (type_id + key bytes) ... | checksum (4B blake2b-32 truncated) |
```

Bit-level breakdown (sizes in bits; var* = length-prefixed):

```
8  |8 |   128   | 8 | var* | 8 | var* | 8 | 8 | var* ... | 32
net flags dsid           domain        name      m  n  signers   cksum
```

- **net**: network identifier (e.g., 0x01 for mainnet, 0x02 for testnet, 0x03 for devnet). - **flags**: bit 0 indicates a multisig account, bit 1 indicates a hardware-bound key; others reserved. - **dsid**: the DataSpaceId (UUID, little-endian) specifying which data space the account resides in. - **domain/name**: UTF-8 strings (each up to 63 bytes) for the account's domain and name. - **multisig**: the multisig threshold m and total number of signers n; the ledger enforces that any transaction from this account has at least m valid signatures (per its policy in the WSV). - **signers**: each signer's public key is encoded as `(key_type_id | key_len | key_bytes)` using the same type identifiers as `iroha_crypto` (e.g., Ed25519, secp256k1, ML-DSA variants).

Example (mainnet, single-signature account):

```
net=0x01, flags=0x00, dsid=3fa85f64-5717-4562-b3fc-2c963f66afa6,
domain="payments", name="alice",
m=1, n=1,
signer=ed25519:0x...,
checksum=0x8b1a2f44
HRP string: snx1qqqp9payments3falice...
   ...8b1a2f44
```

Example (testnet, 2-of-3 multisig account):

```
net=0x02, flags=0x01 (multisig), dsid=2b1c...,
domain="treasury", name="vault", m=2, n=3,
signers: [ed25519:0xaaa..., ed25519:0xbbb..., ml-dsa-87:0xccc...],
checksum=0x5e72c901
HRP string: snt1qqqs4treasury5qvault...5e72c901
```

13.2 Account Lifecycle and State Application

1) **Creation**: An Iroha Special Instruction (ISI) transaction creates the account on-chain using the binary layout; Kura persists this in a new block. The data space/domain policy enforces allowed values for network IDs, flags, and key types at creation.

2) **Updates**: Changes to an account's configuration (e.g., updating signers or changing a multisig threshold) are performed via ISIs and are gated by the account's current policy

(for example, a 2-of-3 multisig account would require at least 2 of the existing 3 signers to approve an update to the signer list).

3) **Execution**: When accounts are used within smart contracts, the IVM uses pointer-ABI objects for ‘AccountId‘ and ‘DataSpaceId‘ to reference them. When a transaction references an account, the Norito decoder ensures the account ID is well-formed (correct lengths, flags, and checksum) before the IVM uses it, preventing any malformed account identifiers from entering execution.

4) **Multisig enforcement**: If an account is multisig (with threshold m out of n), the admission process will require at least m distinct valid signatures from that account's signer list for any transaction. If the same key is included multiple times, it only counts once. Mixed key schemes (e.g., Ed25519 and ML-DSA keys together) are supported; each signature is verified according to its type, and collectively they count toward the threshold (no special aggregated signature needed).

13.3 Triggers and Automation

Triggers are deterministic automations stored in WSV and executed by IVM contexts when conditions fire. A trigger bundles:

- **Condition**: a trigger can be set to fire based on a schedule (specific block height or time interval), an observed event (e.g., a particular account/domain/DS change), or a receipt predicate.
- **Action**: one or more ISIs or a Kotodama contract call (IVM bytecode) to execute when the condition is met. These run under a designated trigger authority account.
- **Budget**: a limit on TEU usage and fees to prevent runaway execution, enforced during the admission phase.

Example (time-based stipend):

```
condition: every 24h at slot boundary
action: Transfer { from: treasury#gov, to: stipend#user, amount: 50 XOR }
budget: 200 TEU, fee payer: treasury#gov
```

Example (event-driven compliance):

```
condition: OnAssetMint { asset: govbond#treasury, amount > 1_000_000 }
action: NotifyRegulator { payload: attestation_hash, dsid }
budget: 300 TEU, fee payer: regulator#ops
```

Triggers are stored on-chain (replicated across all validators); SUMERAGI schedules them within the same fairness and DA budgets as user transactions, ensuring they fire deterministically and in sync across nodes. Trigger handlers are Kotodama artifacts (IVM

bytecode); inputs use the pointer ABI (accounts/assets/DSIDs), and outputs are Norito receipts. The system will reject triggers whose actions would violate consensus safety (e.g., locked_qc ancestry) or data space policy. Governance can set TEU and fee ceilings per trigger class to prevent abuse.

14 Interoperability and Connectors

- **Messaging**: Nexus supports standard ISO 20022 message schemas for payments, treasury, and compliance reporting. The on-chain Norito structures align with these schemas, ensuring that on-chain data stays consistent with off-chain interfaces.
- **Bridges**: Nexus gateway nodes can translate data space artifacts into the formats required by external systems (such as RTGS systems, instant payment rails, or partner ledgers) while preserving complete audit trails.
- **APIs**: The platform offers deterministic JSON and binary APIs for external applications to query data space metadata, cryptographic proofs, and receipts. These interfaces provide analytics and risk engines the information they need without exposing any private payloads.

15 Ecosystem Components

15.1 SoraFS

SoraFS is the horizontally scalable, decentralized storage layer for Nexus. It pairs erasure-coded Kura blocks with WSV state snapshots so that both block history and state can be sharded across storage nodes, while still respecting the privacy boundaries of private data spaces.

Reconstructable shards provide data availability without exporting private payloads. Validators can answer audits and catch-up requests using Merkle and erasure-code commitments rather than raw ledger data, proving inclusion while keeping sensitive content opaque.

An orchestrator streams Norito/Kaigi chunks through relays, applies data-availability retry and backoff policies, and exposes health and pressure metrics (for example, RBC sessions, missing chunks, and store pressure). Operators can use these signals to tune capacity and spot emerging issues before they turn into outages.

At the protocol level, reliable broadcast sessions (READY/DELIVER with TTL and compaction) keep chunks flowing, DA flags and retry/backoff deadlines prevent silent loss, and metrics highlight missing data or overloaded stores. Streaming proxies mirror relay catalogs to disk, enforce room policies, and refresh manifests and health information from governance metadata, giving operations teams a clear, governed surface to intervene on.

By separating storage from execution in this way, SoraFS lets sovereign and private domains scale their data without forking chains or exposing raw ledger contents, while still integrating cleanly with the rest of the Nexus stack.

15.2 SoraNet

SoraNet is the SORA Nexus network overlay for a decentralized internet. The same onion-style transport that carries ledger traffic also carries websites and APIs, with Sora relays acting as a global CDN. Three-hop QUIC circuits, a hybrid post-quantum handshake, and ZK-backed access tickets give builders a place to host sites and data that is private by default, while exit nodes cache static assets so pages remain fast worldwide.

Sites and APIs are published as Norito artifacts stored in SoraFS. The SoraNet privacy overlay fetches these artifacts through exits that enforce policy, so teams get low-latency content delivery without giving up anonymity, governance, or determinism. Instead of launching a separate chain for each use case, regions and sectors introduce governed data spaces and compute routes on a single ledger, keeping composability and auditability intact while still isolating performance and privacy.

Core capabilities.

Data space directory.
An on-chain directory maps each data-space identifier (DSID) to its routing information, admission policy, data-availability rules, and governance owner. Gateways read this directory directly from consensus state, so policy is derived deterministically rather than from out-of-band configuration.

Privacy transport.
Client traffic runs over three-hop QUIC circuits that use a TLS-exporter-bound Noise XX hybrid handshake (Curve25519 + Kyber768). Every circuit advertises mandatory capability TLVs (post-quantum KEM/signature choices, role, version, GREASE) and is authenticated with either Argon2 puzzle tickets or signed `SNTK` tokens. Payloads are carried in fixed 1024,B cells with padding and optional dummy traffic. Connection identifiers are blinded using a rotating salt, `BLAKE3("soranet.blinding.canonical.v1" || salt || cid)`, and clients pin guards to resist correlation. Transcript hashes and capabilities are logged for downgrade detection, while exits cache static assets so SoraNet-hosted sites load quickly from nearby relays.

Web and content hosting.
DeFi and application builders publish websites and APIs as Norito/SoraFS artifacts. The privacy overlay retrieves this content through exits that behave like a CDN: they respect overlay policies and anonymity, but still serve cached assets close to users to keep latency low.

Privacy tickets.
Access to data spaces is controlled via Halo2-backed `TicketEnvelopeV1` structures.

Each envelope carries a blinded CID, scoped permissions (read/write/admin), a maximum number of uses, a validity window, a salt epoch, an issuer signature, and a nullifier. Relays and gateways can admit anonymous fetch or write requests based on these envelopes, while replay and abuse are limited by the nullifier and usage counters.

Governed composition.
New data spaces, lane parameters, fee floors, DA sampling strategies, and compute manifests/routes are all introduced via governance and recorded in WSV. This keeps the topology of SoraNet, and the rules under which it operates, on-ledger and auditable.

Deterministic admission.
Gateways schedule envelopes with an SFQ-style scheduler that supports must-serve slices and per-data-space caps. Before a request reaches consensus, it is checked against the data-space policy (roles, fee payer, TEU limits, and related constraints). DA and RBC parameters are configured per deployment so operators can tune durability and cost.

Compute lane.
Dynamic backends run on a paid compute lane governed by a `ComputeManifest`. Route entries (service/method to Kotodama entrypoint) declare codec allowlists, TTL/gas/response caps, determinism levels, and execution classes (CPU/GPU/TEE with deterministic fallback), as well as input limits for SoraFS streams, resource profiles, price families, fee splits, and sponsor policies. Calls (`ComputeCall`) are Norito-encoded with canonical request hashes, and pricing uses metering weights plus amplifiers for GPU, TEE, or best-effort classes.

State and recovery.
Each lane maintains its own Kura history and WSV snapshots. Data-availability certificates combined with erasure-coded shards allow nodes to catch up without seeing private payloads. The Merge ledger simply orders lane tips, so recovery can be done per lane while preserving global ordering.

Telemetry, rewards, and resilience.
Privacy-preserving metrics and relay incentive scaffolding (e.g., `soranet_privacy_total`, `soranet_reward_payouts`) expose how traffic and rewards flow through the network, while Torii surfaces consensus and DA health. Rolling upgrades gated on runtime hashes, plus circuit breakers and retries, localize faults and let operators roll out changes safely.

Taken together with SoraFS, SoraNet replaces a proliferation of bespoke networks with governed data spaces and scalable lanes on a single ledger, preserving composability, auditability, and deterministic behavior while still meeting practical requirements for privacy and performance.

15.3 Soracles

Soracles are deterministic oracle actors that inject external data (prices, FX rates, reference data) into designated data spaces. They are onboarded via governed attestation, publish Norito-encoded feeds, and are rate-limited and audited like any other participant. Mixed-scheme multisig is supported so oracle committees can blend Ed25519 and ML-DSA keys during the PQ transition. Soracle updates follow the same DA and proof budgeting rules as other DS artifacts, ensuring oracle data cannot bypass admission controls. Soracles solve the "trust the feed" problem: feeds are signed, rate-limited, replay-protected, and can be slashed/rotated via governance if they deviate. The code wires ISIs and metadata to register oracle signers, set rate limits, and bind feeds to DataSpaceIds; delivery rides the same DA/ISI path as other artifacts so quotas and DA checks prevent spam; governance can revoke/rotate keys, and Norito-typed payloads let consumers validate structure deterministically.

15.4 ISO 20022 Alignment

Nexus gateways expose ISO 20022 message schemas for payments, treasury, and compliance reporting. On-chain Norito payloads carry the same canonical fields so ISO 20022 messages can be emitted or consumed without lossy translation. Domain tags and business application headers (BAH) are bound to data space policies so institutions can reuse existing ISO processes (e.g., pacs/pain/camt messages) while settling on SoraNet. Connectors maintain deterministic mappings and validate schema versions to prevent drift. Examples:

- **pacs.008**: customer credit transfer maps to a Norito transfer ISI with debtor/creditor accounts bound to DataSpaceIds; remittance data is preserved in structured addenda.
- **pacs.009**: FI-to-FI transfer aligns with inter-DS settlement, keeping BIC/LEI identifiers in ISO fields and domain-qualified ledger accounts in Norito pointers.
- **pacs.004/camt.056**: returns and recalls map to reversible transfer flows authorized by DS policy; status reason codes round-trip without loss.
- **camt.052/053/054**: reporting messages map to ledger event streams with balances, entries, and booking timestamps; ISO sequencing (Stmt/IntrBkSttlm) is preserved for reconciliation.
- **pain.001/002**: customer initiation and status responses integrate with gateways; Norito ensures deterministic validation of mandate, charges, and scheme-specific options.

Example pacs.008 mapping:

ISO 20022 field	Norito / ledger field

GrpHdr/MsgId	DataSpaceId-qualified message id (UUID)
GrpHdr/CreDtTm	`created_at` (slot, lane)
PmtId/InstrId	Instruction hash (Norito envelope id)
PmtId/EndToEndId	User reference in metadata TLV
IntrBkSttlmAmt/Ccy	AssetDefinitionId (currency code)
IntrBkSttlmAmt/Value	Transfer amount (TEU)
Dbtr/DbtrAcct	Debtor AccountId (domain-qualified, DS-scoped)
Cdtr/CdtrAcct	Creditor AccountId
Purp	Purpose code, structured addenda
RgltryRptg	Regulatory tags bound to DS policy

ISO versioning: connectors pin schema versions, expose validation errors, and refuse ambiguous mappings. BAH/domain tags are bound to data space policy so institutions can enforce channel-specific rules (e.g., RTGS vs instant rail).

15.5 Norito Codec and Streaming

Norito is the canonical codec for all ledger payloads. Binary and JSON representations use Norito helpers; SCALE and other serializers are excluded from production paths. Norito payloads advertise layout via versioned headers, and blocks are persisted/distributed as canonical SignedBlockWire encodings that prefix version bytes with Norito headers.

Norito Streaming transports (Norito Stream and Kaigi stream) chunk Norito payloads over relay paths with deterministic framing. Each chunk carries:

- Stream tag (‘norito-stream‘ or ‘kaigi-stream‘), lane/dataspace routing, chunk index, total size, and checksum.
- Norito-encoded control ISIs for admission, replay, and health.

Relays spool routes on disk (‘exit-<relay-id>/norito-stream/*.norito‘ or ‘kaigi-stream/*.norito‘) and refresh them on governance updates. Streaming contracts enforce DA and retry policies; attestation/health events are emitted over Torii.

15.6 Kaigi and Taikai

Kaigi is the real-time conferencing subsystem; Taikai is the streaming/data pipeline. Both use the same Norito streaming transport as the lower-level “norito-stream”/“kaigi-stream” channels, so every packet (media or data) is chunked with deterministic framing (tag, routing, index/total, checksum). Relays spool these chunks on disk and refresh routing tables when governance updates manifests, which keeps delivery deterministic even when relays churn. Admission, routing, and retries are driven by ISIs, so control is on-ledger and auditable.

Kaigi flow: a caller issues ‘CreateKaigi‘, gateways enforce the room policy from domain metadata (‘stream.kaigi.public‘ or ‘stream.kaigi.authenticated‘), and relays are chosen from the governed manifest set (‘RegisterKaigiRelay‘/‘SetKaigiRelayManifest‘). Participants join/leave via ISIs, and usage is recorded (‘RecordKaigiUsage‘) with Halo2 `kaigi_zk` proofs that show roster membership and usage without revealing the roster. Validators verify those proofs on-chain; mocks are compile-gated for tests. Relays publish health/bandwidth classes over Torii REST/SSE, and governance can slash or evict relays that misbehave.

Taikai flow: bulk data/video uses the same Norito framing and relay manifests but tunes policies for throughput and data availability. Norito chunks are buffered to ‘exit-<relay-id>/norito-stream/*.norito‘ or ‘kaigi-stream/*.norito‘, and DA/retry settings dictate how many replicas and retries are required before a chunk is considered delivered. Governance can roll manifests or change DA policy to rebalance routes; ordering and checksums ensure receivers deterministically reassemble streams.

Key ISIs and behaviors:

- **Kaigi ISIs**: ‘CreateKaigi‘, ‘JoinKaigi‘, ‘LeaveKaigi‘, ‘EndKaigi‘, ‘RecordKaigiUsage‘, ‘RegisterKaigiRelay‘, ‘SetKaigiRelayManifest‘. Room policy (‘public‘/‘authenticated‘) binds to domain metadata prefixes (‘stream.kaigi.public‘ / ‘stream.kaigi.authenticated‘).
- **Privacy**: Halo2-IPA proofs for roster membership and usage (`kaigi_zk` circuits) are verified on-chain.
- **Telemetry**: relays expose health, manifests, and bandwidth classes via Torii REST/SSE; governance curates allowlists and can slash misbehavior.
- **Taikai**: Norito-streaming bridge for data/video; uses the same streaming transport with Norito chunks and DA policies; manifests and routes are governed identically.

16 Governance: SORA Parliament

The SORA Parliament governs parameter changes, data space creation, and policy updates. It operates on-chain with transparent Norito-encoded proposals and deterministic execution via IVM/ISIs.

16.1 Bodies and Selection

Parliament comprises two selectable bodies:

- **Council**: fixed seats elected by stake-weighted NPoS or appointed via governance (configurable in genesis). Seat count and term lengths are set in configuration.
- **Assembly**: broader participation body; can be stake-weighted or one-account-one-vote depending on parameterization. Used for broad-impact proposals (e.g., fee changes).

Selection algorithms mirror Sumeragi NPoS: VRF sortition with threshold τ over registered voters/validators, ensuring unpredictability and unbiasability. Membership snapshots and view hashes are published via Sumeragi status so nodes agree on the active roster. Slashing/eviction is applied for equivocation or double-voting on the same proposal.

- **Proposal submission**: Any eligible member (role-gated) submits a proposal object: `kind` (config change, DS creation, fee change), `payload` (Norito-encoded diff), `quorum`, `threshold`, `voting_window`, `enactment_slot`.
- **Staking/bonding**: XOR bond locks with the proposal; slashed if malformed or spam. Bonds return when the proposal is finalized (pass/fail).
- **Voting**: Members sign ballots (Ed25519/secp256k1/ML-DSA). Votes are tallied deterministically; duplicate or invalid keys are ignored. Mixed key types are allowed per voter registry.
- **Quorum & thresholds**: Proposals are valid when votes $\geq$ quorum and approvals $\geq$ threshold (e.g., 2/3 weighted by stake or seat). Abstain and veto counts are recorded for audit.
- **Finalization**: SUMERAGI includes the tally in a block once the voting window closes. A passed proposal is queued for enactment at `enactment_slot`; failed or expired proposals release bonds.
- **Enactment**: An ISI applies the change to `iroha_config` or DS registry in WSV and persists the update to Kura. Governance diffs are hashed and stored alongside the block for reproducibility.
- **Rollback guard**: If enactment fails validation (e.g., schema mismatch), the change is rejected and bonds are slashed; state remains unchanged.

Example (parameter change):

```
kind: UpdateConfig
payload:
  da.q_in_slot_total: 2048 -> 2304
  envelope_max_typ: 16MB -> 18MB
quorum: 60\%,
threshold: 2/3 yes,
voting_window: 7 days,
enactment_slot: +1440
```

Example (new data space):

```
kind: CreateDataSpace
payload:
  dsid: 7c8a...
```

```
  class: private
  policy: dual-sign DA
  roles: [...]
quorum: 50\%,
threshold: simple majority,
voting_window: 5 days,
enactment_slot: +720
```

All governance artifacts (proposals, votes, tallies, enactment receipts) are Norito-encoded, signed, and stored in Kura with WSV indexes so observers can replay decisions and verify that enacted rules match the voted payloads.

16.2 Runtime Upgrades

Runtime upgrades follow the same deterministic governance channel and keep state intact:

1. **Proposal**: Carry an upgrade manifest (hashes of IVM/Kotodama/host binaries, config diffs, ABI hash, activation slot) with an XOR bond.
2. **Vote and finalize**: Parliament quorum/threshold decides. The manifest hash is recorded in WSV; Kura stores the proposal and ballots.
3. **Staging**: Validators fetch signed artifacts, verify hashes, and stage binaries. A read-only health check runs against WSV/Kura snapshots to confirm schema compatibility before activation.
4. **Activation**: At `activation_slot`, nodes atomically switch to the staged runtime. SUMERAGI refuses proposals/blocks from nodes advertising a mismatched manifest until they update.
5. **Fallback**: If activation validation fails (e.g., ABI hash mismatch), the upgrade aborts, bonds may be slashed, and nodes remain on the prior runtime. A fresh proposal is required to retry.

Upgrades are lane-agnostic: all lanes switch at the same activation slot, and mixed-runtime participation is rejected by SUMERAGI. Operators can dry-run staged binaries against WSV/Kura snapshots; failed dry-runs should be signaled back into governance before activation to avoid bond loss. ABI hashes and pointer types stay versioned, so contracts/hosts can detect incompatibilities before execution. Telemetry exposes live manifest hashes for operators and observers.

16.3 Economic Model and XOR Utility

The native XOR token underpins the economic model across all SoraNet domains. All network fees, validator staking rewards, and bonded admissions for governance are

denominated in XOR, which means validators, data availability attesters, and oracle providers are all rewarded in the same asset. This gives users a single settlement currency across both public and private data spaces. Governance actions (such as parameter set changes or data space creation) require proposers to lock up XOR bonds, aligning their incentives with the network's health. If a data space introduces its own asset (like a CBDC token), it can be bridged or collateralized 1:1 with XOR, giving builders liquidity while preserving deterministic ledger-level attestability. Fee markets can be tuned per data space, but XOR remains the universal settlement asset for network services, ensuring predictable pricing and unified rewards for validators, provers, and gateways.

16.4 Offline-Offline Transactions

SORA Nexus supports value exchange even when both parties are offline by issuing pre-authorized, signed vouchers bound to specific state snapshots and revocation windows. Two devices can transact without network access by exchanging such a voucher, and later a gateway reconciles it once connectivity is restored. The gateway verifies the voucher's signatures, checks replay-protection counters to ensure it hasn't been used before, and ensures data availability inclusion rules are satisfied before final settlement. Circuit breakers enforce deterministic handling of replays (preventing double-spends), and policies can cap the amount transacted offline, require dual signatures for higher values, or mandate collateralized XOR locks to back the vouchers until reconciliation completes.

16.5 SDKs and Developer Experience

Developers can access SoraNet through language-specific SDKs for mobile, web, and server environments, offering Norito-first APIs, deterministic transaction builders, and easy queries for receipts and proofs. These SDKs provide helpers for ISO 20022 message formats, pointer-ABI encoding, and offline voucher flows, allowing wallets to support both online and offline transaction use cases. Builders can choose either a lightweight REST API or gRPC to communicate with Nexus gateways, while still retaining full auditability and policy enforcement for every data space they interact with. By using these SDKs, the user experience in wallets and services is naturally aligned with the core protocol primitives, IVM data types, Norito envelopes, XOR-based fee logic, and DA receipts, so applications inherit the same determinism and auditability as the SORA Nexus protocol itself.

17 Privacy, Security, and Compliance

- **Role and domain controls**: Fine-grained role-based permissions and per-domain policies restrict who may create assets, submit transactions, or view data within each data space.

- **Auditability**: Data availability certificates, attestation hashes, and Merkle proofs give regulators deterministic audit paths. Duplicate signatures are ignored deterministically; mixed-scheme multisig (multiple key types) is supported.
- **Deterministic rejections**: Invalid Norito envelopes, malformed opcodes, or policy violations trigger explicit errors rather than any undefined behavior.
- **PQ readiness**: Consensus and data space attestations already use ML-DSA-87; DA certificates reserve fields for post-quantum audit data so the network can enforce policies like dual-signatures without major protocol changes.

17.1 Privacy Transactions and Proof Systems

Shielded transfers: commitments $\mathsf{cm} = H(\text{tag} \parallel \text{payload})$ live in a shielded Merkle tree; spends reveal nullifiers $\mathsf{nf} = H(\text{sk} \parallel \mathsf{cm} \parallel \text{salt})$ to prevent double-spends. Inputs/outputs and paths are Norito-encoded; IVM enforces Merkle rules deterministically.

Halo2 circuits (PLONKish): Each circuit is a table of columns and rows. Advice columns hold witness values; fixed columns carry constants; selector columns indicate which gate applies on each row; instance columns are public inputs. Gates are polynomial equations, e.g., $q_{\text{add}}(X)(A_0 + A_1 - A_2) = 0$, that must evaluate to zero wherever the selector is 1. Copy constraints state that certain cells must be equal across rows/columns; the permutation argument enforces this by building a product $Z(X)$ that stays consistent only if the declared wiring is respected. Lookups assert that witness tuples appear in a predefined table: a multiset check compares products of $(f_i - \beta g_i)$ over witness vs. table values with a random challenge β. All column polynomials are committed (IPA/Bulletproofs), the prover derives challenges via Fiat–Shamir, and combines all gate, copy, and lookup constraints into one quotient polynomial $Q(X)$. The verifier opens columns at random points and checks that $Q(x) = 0$; if so, all constraints hold [2]. **Soundness & cost**: Range/consistency checks are encoded as dedicated gates/lookups; transcript binding is via Fiat–Shamir over all commitments, preventing adaptive rewiring. IPA commitments make verifier cost logarithmic in column length; typical circuits open a small set of points (e.g., 2–4 advice, 1 permutation, 1 lookup table), keeping verification within a few ms on modern CPUs while sustaining short proofs.

IPA commitments: commit $p(X) = \sum p_i X^i$ as $C = \sum p_i G_i$; prove $p(z) = v$ by recursive folding with challenges x_k, revealing L_k, R_k; verifier recomputes folded commitment; proof size is logarithmic [3].

Flow: wallet creates commitments/nullifiers and Halo2 proofs; gateways apply DA/policy; validators verify proofs, enforce nullifier uniqueness, update shielded roots, and include artifacts in lane envelopes with DA/FASTPQ proofs. Observers see nullifiers, roots, and receipts; values stay hidden.

18 Operations and Observability

- **Telemetry**: Lane finality, DA failure rates, envelope sizes, and scheduler metrics surface SLO adherence. Circuit-breaker state is observable for operators and governance.
- **Upgrades**: Rolling upgrades are supported through stateless gateways and replicated consensus clusters. Lane fusion under low load can reduce maintenance windows.
- **Resilience**: Geo-redundant deployments with quorum certificates keep state consistent even under data center failover. Provers can be scaled independently of validators.

19 Performance Targets and Defaults

Parameter	Default (illustrative)
Lane finality target	$\leq$ 1 s between non-empty blocks; no empty blocks
Commit window Δ	$\leq$ 2 slots for DA and proof completion
Lane count	$K = 4$ lanes, fusing under low load
Lane TEU budget	20,000 transfer-equivalent units per second
Envelope size	Typical $\leq$ 16 MB; hard cap 32 MB
DA sample	$q_{\text{in\_slot\_per\_ds}} = 8$ Ed25519 signatures; $q_{\text{in\_slot\_total}} \leq 2048$
Proof verify time	100–200 ms per lane committee for lane proofs
Validator quorum	Example 22 validators per lane (f = 7)
Attestation signatures	ML-DSA-87 for DS and QCs; Ed25519 for public DA samples

20 Mathematical Foundations

20.1 Start-Time Fair Queuing (SFQ)

For each data space i with weight w_i derived from fee density and packet size p_i (in TEU), SFQ assigns virtual start/finish times:

$$V(t) = \max_j \{s_j(t)\}, \tag{1}$$

$$s_i = \max(V, f_i^{\text{prev}}), \tag{2}$$

$$f_i = s_i + \frac{p_i}{w_i}. \tag{3}$$

Envelopes serve DSs in ascending f_i order while enforcing caps: $p_i \leq p_{\max}$, $\sum p_i \leq \text{TEU}_{\text{lane}}$, and must-serve age $a_i \leq S$. Fusion occurs when $\sum_i \lambda_i \leq \lambda_{\text{floor}}$ for two slots; split when $\sum_i \lambda_i \geq \lambda_{\text{exit}}$.

20.2 Lane and DA Budgets

DA sampling per public DS uses q signatures with $q \in [6, 16]$. The proposer enforces changed_public_ds $\leq \left\lfloor \frac{q_{\text{total}}}{q} \right\rfloor$ with $q_{\text{total}} \leq 2048$. DA verification must finish before $t_{\text{DA}} \leq 300\,\text{ms}$; lane proof verification must satisfy $t_{\text{proof}} \leq 200\,\text{ms}$. Envelope size constraint: $\text{bytes}_{\text{env}} \leq$ ENVELOPE_MAX_TYP (16 MB target, 32 MB hard).

20.3 Two-Dimensional Erasure Coding

Blocks are partitioned into an $m \times n$ matrix of shards. Row and column parities are computed with a systematic Reed–Solomon code over $GF(2^8)$:

$$p_r^{\text{row}} = \sum_{c=1}^{n} \alpha_c \cdot d_{r,c}, \tag{4}$$

$$p_c^{\text{col}} = \sum_{r=1}^{m} \beta_r \cdot d_{r,c}, \tag{5}$$

where α_c, β_r are generator coefficients and $d_{r,c}$ is the data shard at row r, column c. Recovery succeeds with any set of rows and columns satisfying $r_{\text{avail}} \geq k_r$ and $c_{\text{avail}} \geq k_c$. Reed–Solomon is a Maximum Distance Separable (MDS) code: any k original shards can be reconstructed from any k available coded shards. ZK proofs bind the parity roots to the Merkle root included in the block header, proving consistency without revealing private shards.

Reed–Solomon recap RS codes treat each shard as an element of a finite field (here $GF(2^8)$) and encode a polynomial $P(x)$ of degree $k-1$ through its evaluations at distinct points $\{x_i\}$. Given any k evaluations, Lagrange interpolation recovers $P(x)$ and all original shards. Systematic encoding stores the original data as the first k evaluations and parity as the remainder [4]. Shards map directly into Kura/WSV storage lanes; validators can reconstruct the full block state without ever exposing private DS payloads outside their permitted domains.

20.4 DA Sampling Confidence

Given a threshold attester set of size S with T required signatures, an in-slot sample of size q detects a fraction f of faulty attestations with probability:

$$P(\text{detect}) = 1 - \frac{\binom{S - fS}{q}}{\binom{S}{q}}. \tag{6}$$

Operators tune q to achieve target detection (e.g., $> 99.9\%$) within the verify-time budget while full T-of-S certificates are audited off-path.

20.5 STARK Security Budget

For DEEP-FRI with blow-up b, query count Q, and soundness error per query ϵ, the overall soundness satisfies $\epsilon_{\text{tot}} \leq Q \cdot b^{-\sigma}$ for rate parameter $\sigma \approx \log_b(\text{LD/HD})$. Parameter sets use $b \in \{8, 16\}$, $Q \in [34, 46]$, and grinding bits $g \in [21, 23]$ to target ≥ 128 bits of security. Fiat–Shamir challenge space is widened with grinding: effective security $\geq 2^g \cdot \epsilon_{\text{tot}}^{-1}$.

Privacy vs. audit modes FASTPQ can run in audit-only mode (no witness blinding) for regulated public data spaces, or in privacy mode with randomizers in the trace to hide witness values. DS policy selects the mode; the same AIR and DEEP-FRI stack are reused, so verification logic stays identical. Grinding bits and query counts are governed parameters so operators can dial target security without changing the proof format.

DEEP-FRI recap FRI (Fast Reed–Solomon IOPP) proves that an evaluation vector is close to a low-degree polynomial by repeatedly folding the codeword: at each round it combines paired points (a, b) into $a + rb$ using a random scalar r, halving the domain while preserving low-degree structure. DEEP-FRI augments FRI with domain extension and sampling tricks to reduce soundness error further. A STARK prover supplies commitment Merkle roots for each round; the verifier opens a small set of queried positions to check consistency with the claimed degree bound [5, 6]. The folding depth and query count govern soundness and runtime.

20.6 FASTPQ Constraint System

FASTPQ builds an AIR over Goldilocks for KV updates. Each row encodes key/value limbs, selectors, SMT neighbors, and metadata. Core constraints:

$$s_{\text{active}}(s_{\text{transfer}} + s_{\text{mint}} + s_{\text{burn}} + s_{\text{role\_grant}} + s_{\text{role\_revoke}} + s_{\text{meta\_set}} - 1) = 0, \tag{7}$$

$$\text{running\_asset\_delta}_i = (1 - r_{\text{start},i}) \cdot \text{running\_asset\_delta}_{i-1} + \delta_i, \tag{8}$$

$$\text{SMT}_\ell : \begin{cases} \text{node\_out} = \text{Poseidon2}(\text{node\_in}, \text{sibling}), & \text{if path\_bit} = 0, \\ \text{node\_out} = \text{Poseidon2}(\text{sibling}, \text{node\_in}), & \text{if path\_bit} = 1, \end{cases} \tag{9}$$

with padding enforced by monotone s_{active}. Constraints are lifted to the low-degree domain D and divided by the vanishing polynomial $Z_N(X) = X^N - 1$ to form composition polynomials $F_j(X)$.

20.7 FASTPQ Proof Composition

1. Commit to trace and lookup LDEs with Poseidon2 Merkle roots $(\mathsf{root}_{\text{trace}}, \mathsf{root}_{\text{lookup}})$.
2. Derive challenges $\gamma, \{\alpha_j\}, \{\beta_\ell\}$ via Fiat–Shamir; build lookup grand product Z and composition polynomial $C(X) = \sum_j \alpha_j F_j(X)$.

3. Run DEEP-FRI with arity $r \in \{8, 16\}$ over $C(X)$: fold layers, commit roots, and sample queries.

4. Open queried positions of trace, lookup, and FRI polynomials; include Merkle proofs for each opening.

5. Verifier recomputes challenges, checks lookup boundary $Z_{\text{final}}/T = 1$, verifies Merkle openings, and runs FRI checks to confirm $\deg(C) < N_{\text{eval}}$.

20.8 Fiat–Shamir Transform

The Fiat–Shamir transform turns interactive public-coin protocols into noninteractive ones in the random oracle model [7]. Instead of receiving verifier randomness, the prover derives challenges by hashing the transcript with domain-separated tags:

1. Initialize with protocol identifiers and public inputs.

2. After each commitment (e.g., Merkle root), hash the transcript to derive the next challenge scalar; append it to the transcript.

3. Use the challenge to sample queries, randomizers, or permutation challenges; continue until all rounds are defined.

4. Output commitments, openings, and any responses. The verifier recomputes the transcript and challenges identically and checks all relations.

Grinding expands the challenge space by requiring hashes with leading/trailing zeros, raising effective soundness for STARK/DEEP-FRI parameters.

20.9 FASTPQ End-to-End Example (Narrative)

- **Trace**: data space D processes transfers/mints/burns; for each state change it emits a row with packed key/value limbs, selectors, SMT siblings, and metadata (dsid, slot). Rows are sorted deterministically and padded to $N = 2^k$ with $s_{\text{active}} = 0$.

- **Commit**: prover Merkle-commits the low-degree extension of each column (trace and lookup tables).

- **Challenges**: transcript absorbs roots; derives γ for lookup products, α_j for composition, β_ℓ for DEEP-FRI.

- **Quotient**: build $C(X) = \sum_j \alpha_j F_j(X)$ by dividing each constraint by $Z_N(X)$; fold it via DEEP-FRI into layer roots.

- **Openings**: prover reveals a few queried positions of trace, lookup, and FRI layers with Merkle proofs. Verifier recomputes products (e.g., Z_{final}/T) and checks FRI consistency. If all pass, the state transition is accepted without seeing private books.

20.10 zk-STARK vs. STARK

FASTPQ is a zk-STARK: it is transparent (no trusted setup), hash-based, and can hide witness values via randomness in the trace. In audit-only modes, the same AIR/FRI structure can be used without zero-knowledge blinding; in privacy modes, blinding is enabled so witnesses remain hidden while constraints are enforced.

20.11 XOR Fee Equilibrium

Let F_i be the fee rate (in XOR) for data space i, d_i the demand, and c_i the TEU cost per unit work. Equilibrium fee for lane budget $\mathrm{TEU}_{\mathrm{lane}}$ satisfies:

$$\sum_i \min\left(d_i, \frac{F_i}{c_i}\right) \leq \mathrm{TEU}_{\mathrm{lane}}, \tag{10}$$

$$F_i^* = \arg\max_{F_i} \; F_i \cdot \min\left(d_i, \frac{F_i}{c_i}\right) \text{ subject to fairness caps.} \tag{11}$$

Governance can adjust F_i floors and ceilings to steer demand while keeping XOR-denominated rewards stable for validators and attesters.

21 Use Cases

- **Central bank digital currencies (CBDCs)**: Private data spaces for issuance and regulated wallets; public data spaces for innovation by licensed intermediaries, with audit-ready proofs and ISO 20022 message processing.
- **Interbank settlement**: Sub-second lane finality and deterministic admission make SORA Nexus suitable for RTGS modernization with programmable treasury workflows.
- **Tokenized assets and registries**: Pointer-ABI types and Norito schemas provide predictable encoding for digital securities, NFTs, and registries that span multiple data spaces while preserving privacy.
- **Cross-border payment corridors**: Nexus gateways translate artifacts into partner formats while preserving Merkle proofs, enabling regulatory assurance across ledgers.

22 Conclusion

SORA Nexus, built on Hyperledger Iroha 3, is designed as the single global, infinitely scalable ledger, an "end of history" for blockchains. One network, many data spaces, no parallel chains required: every transaction for mankind can live on SORA Nexus. IVM provides deterministic execution, data spaces enforce policy isolation, XOR aligns incentives, and proofs/DA preserve integrity and auditability. With these pieces, SORA

Nexus is positioned to carry CBDCs, capital markets, and open innovation on a single, endlessly scalable ledger; no other blockchain is needed.

References

[1] M. Takemiya, "Blockchain for empowering central bank digital currencies (cbdcs): Examples from industry," BCK24 (Blockchain Conference), University of Zurich, 2024.

[2] S. Bowe, J. Grigg, and D. Hopwood, "Halo: Recursive proof composition without a trusted setup," Cryptology ePrint Archive, Paper 2019/1021, 2019. [Online]. Available: https://eprint.iacr.org/2019/1021

[3] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell, "Bulletproofs: Short proofs for confidential transactions and more," in *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 315–334.

[4] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, 1960.

[5] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Fast reed-solomon interactive oracle proofs of proximity," in *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), vol. 107. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018, pp. 14:1–14:17.

[6] ——, "Scalable, transparent, and post-quantum secure computational integrity," Cryptology ePrint Archive, Paper 2018/046, 2018. [Online]. Available: https://eprint.iacr.org/2018/046

[7] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," *Advances in Cryptology – CRYPTO '86*, pp. 186–194, 1986.