Deterministic State Machines as a Trust Substrate: Relationship-Scoped Finality, Canonical Verification, and Why "World Computer" Must Become "World Channels"

Brandon "Cryptskii" Ramsay

December 22, 2025

Abstract

Blockchains proposed a universal trust layer: a decentralized "world computer" that replaces intermediaries with shared-state consensus. In practice, shared-state ordering forces unrelated interactions to contend for the same resources and exposes systemic adversarial surfaces (reorganizations, MEV, shared execution malware, congestion externalities). This report formalizes a Deterministic State Machine (DSM) architecture in which state transitions are relationship-scoped, forward-only, and self-verifying at endpoints, with non-authoritative storage nodes acting as availability infrastructure. We synthesize the full set of implications discussed in design review: why core verification must be canonical and singular; how online and offline transfers share the same guarantees; why recipient signatures are not required for sender-finality; what "pending in inbox" means precisely; why reorg and MEV are structurally absent; how conditional logic (pre-commit forking, DLVs) recreates contract-like expressiveness without a shared Turing-complete VM; how tokenless transitions generalize the primitive; and why the only credible "world computer" is a parallel composition of deterministic channels rather than a single globally ordered log. We conclude with DSM-specific downsides and boundary conditions, including explicit composability protocols, receiver-gated acceptance, identity bootstrapping, and the non-negotiable requirement that the bytes signed are the bytes executed.

1 Introduction: Narrative vs. Facts

The blockchain narrative frames a single decentralized machine executing global programs. The facts of distributed systems are uncompromising: a global shared ordering domain couples unrelated interactions into one contention surface and invites adversarial reordering and censorship. Even with scaling advances, a globally ordered shared state machine retains structural limits:

- **Global ordering** ⇒ contention, adversarial reordering, and extractable value.
- **Consensus** ⇒ systemic bottlenecks and delayed/probabilistic finality.
- **Shared execution** ⇒ ambient malware surfaces and "encode-anything" risk.
- **Congestion externalities** ⇒ one workload degrades unrelated workloads.

DSM adopts the opposite thesis: the world is not one computer. The world is many devices needing a standard trust primitive. The correct substrate is therefore *parallel relationship channels* with deterministic verification, not one global pipeline serializing all meaning.

2 System Model: Relationship-Scoped Deterministic Channels

DSM defines state as a set of *relationship-specific* straight hash chains, backed by Sparse Merkle Tree (SMT) commitments for auditability, with deterministic policies (e.g., CPTA token policy anchors) enforced at transition time.

2.1 Canonical Operation Bytes and Post-Quantum Signatures

A DSM transition is represented by an operation payload $\mathcal{O}$ with canonical serialization:

$$\mathsf{bytes}(\mathcal{O}) \triangleq \text{canonical protobuf encoding of } \mathcal{O}.$$

The sender produces a post-quantum signature (e.g., SPHINCS+) over these bytes:

$$\sigma \leftarrow \mathsf{Sign}_{sk}(\mathsf{bytes}(\mathcal{O})).$$

The receiver verifies:

$$\mathsf{Verify}_{pk}(\mathsf{bytes}(\mathcal{O}), \sigma) = \mathsf{true}.$$

A replay-resistant nonce n (for online delivery) is included *inside* $\mathcal{O}$ and therefore inside the signed bytes.

2.2 Forward-Only Acceptance and One-Way State

DSM is forward-only: the receiver either accepts a transition (advancing their local relationship state) or rejects it (no state change). There is no rollback mechanism because there is no globally shared history to reorganize.

2.3 Storage Nodes as Non-Authoritative Mailboxes (B0x)

Storage nodes are intentionally "dumb": they store and relay envelopes but do not adjudicate correctness. Endpoints verify and enforce policy. Finality does not require going online; when online, devices simply check their inbox (b0x) for waiting items.

3 Why Verification Must Live in the Core

3.1 Why It Sometimes Does *Not* Initially

Systems often begin with verification in outer layers (routers, SDK surfaces) because:

- integration velocity is higher when verification is localized near transport,
- legacy cores may predate the final canonical envelope model,
- early prototypes assume "trusted caller" boundaries that later must be removed.

This is a transitional state, not a safe endpoint.

3.2 The Non-Negotiable Production Invariant

The strongest DSM invariant is also the sharpest hazard:

The bytes signed must equal the bytes executed.

Formally, if $\mathcal{O}$ is accepted, then:

$$\mathsf{Execute}(\mathsf{bytes}(\mathcal{O})) \quad \text{must be the only execution path.}$$

Any drift between (a) the signed/verifiable payload and (b) a separately reconstructed internal operation creates *consensus-by-bug*: different layers agree on different realities. This is precisely why a state-machine-facing operation that omits nonce (or differs in any field ordering/serialization) is a latent systemic risk the moment core signature enforcement is introduced.

3.3 Canonicalization Strategy

A production DSM core must:

- accept the canonical operation bytes as input,
- verify signature *in core* against those bytes,
- parse the bytes to an internal representation,
- execute exactly that parsed operation (no rehydration from parallel fields).

This collapses the security model to one auditable object: the canonical bytestring.

4 Online vs. Offline: Same Guarantees, Different Delivery

DSM aims for identical correctness guarantees across transport modes. Only delivery differs.

4.1 Offline Transfers (Immediate Mutual Verification)

Offline bilateral transfer (e.g., Bluetooth) is a direct verification event. Both parties validate the transition and advance relationship state without involving storage nodes. Finality is achieved at acceptance.

4.2 Online Transfers (Mailbox Delivery)

Online transfer uses B0x as mailbox. The sender commits and submits an envelope; the receiver later retrieves and verifies. Tokens (or other state deltas) are best modeled as *pending* until receiver acceptance.

If invalid, what happens? An invalid envelope is rejected. Because DSM is forward-only, rejection is a pure no-op: nothing is "rolled back" because nothing was committed. The receiver state remains unchanged.

Where does the "sent value" sit before acceptance? Operationally, the envelope exists in the recipient's inbox (B0x) unaccepted. Cryptographically, it is not part of receiver state. Economically, the sender has already committed the spend on their side; the receiver has not credited it. This is not a contradiction: DSM cleanly separates *sender commitment* from *receiver acceptance*.

Conditionality and DLVs If an application requires atomic conditional release (timelocks, escrow-like behavior, multi-step negotiation), DSM uses Deterministic Limbo Vaults (DLVs) so that value is never ambiguously "lost in limbo." Instead, it is deterministically held until an agreed condition is satisfied.

5 Why No Recipient Signature Is Required During Online Send

The sender signature proves:

- authenticity of intent (who authorized the operation),
- integrity of the canonical operation bytes,
- replay protection via committed nonce and/or deterministic idempotency identifiers.

The recipient signature is not required for correctness because acceptance is enforced by the recipient's verification gate. In DSM terms: *verification is the receiver's action.* The recipient does not need to co-sign the sender's intent; they need to verify before accepting into their own forward-only state.

6 No Global Bottleneck: "No Relationship Waits on Any Other"

Each bilateral relationship is its own state progression. There is no bottleneck where unrelated relationships must serialize behind one ordering domain. This property is the architectural source of several downstream eliminations:

6.1 No Reorg

Reorganizations exist when multiple competing global histories can be adopted. DSM has no shared global history. Each relationship has one forward-only acceptance history. Therefore, "reorg" is not a meaningful operation.

6.2 No MEV

MEV (Miner/Maximal Extractable Value) exists when an intermediary controls ordering and inclusion (mempool/block production). DSM has:

- no global mempool,
- no privileged block producer,
- no shared ordering domain for unrelated transactions.

Therefore, the core precondition for MEV disappears.

7 Smart-Contract Malware and Hidden-Payload Attacks

Shared Turing-complete execution layers enable attacks where malicious behavior is embedded in code paths that become ambient infrastructure. DSM eliminates this entire class by construction:

- there is no shared VM that executes arbitrary code on behalf of third parties,
- transitions are explicit, canonical, and policy-validated,
- recipients verify before acceptance, and unknown/invalid senders are rejected.

The result is not "no security issues ever"; rather, it removes the shared-execution malware surface that makes one bad contract everyone's problem.

8 Pre-Commit Forking and Negotiated Conditionals (Non-Turing, Still Expressive)

DSM reproduces a large fraction of "smart contract" utility via deterministic commitment protocols.

8.1 Pre-Commit Forking

A party can pre-commit to multiple mutually exclusive branches:

- the space of possible outcomes is enumerated up front,
- the counterparty can accept exactly one branch,
- once a branch is chosen, the state progression is locked forward-only.

This provides "choose-one" contract semantics without global shared execution.

8.2 Bidirectional Negotiation (Counter-Conditions)

Commitments can go back and forth. A receiver can counter-propose conditions by issuing their own pre-commit fork set or by requiring a DLV-based release condition. DSM becomes a deterministic negotiation protocol: both sides can constrain the outcome space, but all constraints remain explicit and verifiable.

9 Tokenless Transitions: DSM as a General Primitive

Not all transitions involve tokens. Tokens are just one instance of state change. The same deterministic envelope model applies to:

- identity updates and key rotation,
- permissions and access grants,
- attestations and acknowledgments,
- message delivery receipts,
- application-specific state commitments.

This is why DSM is a primitive: a standard way to produce forward-only verifiable state transitions between devices, regardless of whether a token ledger is involved.

10 Performance and "Speeding Up the Internet" (What That Actually Means)

DSM does not make IP packets travel faster. It reduces *trust latency* and *coordination overhead*:

- No server round-trip is required for correctness between two parties.
- No global ordering delay is required for finality within a relationship.
- Unrelated interactions do not block each other.

Even relative to centralized systems, DSM can reduce operations that normally "bounce off" servers for authentication, authorization, and auditing, by enabling direct verifiable commitments between endpoints. The key is that the channel is deterministic and non-interfering: opening a thread for one relationship does not degrade others.

11 Messaging and Email: Redefining "Communication" as Verifiable Channels

11.1 Messaging vs. Server-Brokered E2EE

Signal/WhatsApp provide strong end-to-end encryption, but delivery and metadata still depend on central broker infrastructure. DSM enables a different security posture:

- direct sender→receiver verifiable envelopes,
- receiver-gated acceptance with explicit receipts,
- storage nodes as non-authoritative mailboxes rather than brokers of truth.

This can make messaging more sovereign: the trust substrate is endpoint verification, not service mediation.

11.2 Email as Deterministic Mailbox + Verifiable Receipts

Email's historic model assumes untrusted relays and retrofitted trust (SPF/DKIM/DMARC) layered onto a permissive protocol. DSM can redefine "email" as:

- canonical message objects,
- deterministic sender authorization signatures,
- recipient acceptance/receipt as first-class state transitions,
- mailbox storage as availability infrastructure, not authority.

The result is a strictly verifiable communication primitive where "delivered," "seen," and "accepted" are cryptographic facts rather than heuristics.

12 Storage Nodes as Critical Infrastructure (Without Becoming Trusted)

Storage nodes become crucial because they uphold availability and discoverability. However, crucial does not mean trusted:

- **Truth** lives at endpoints (verification and policy enforcement).
- **Reachability** is supported by storage replication and retrieval/ack semantics.

This separation prevents availability infrastructure from silently acquiring governance power over correctness.

13 CAP Theorem and the "World Computer" Claim

Brewer's CAP theorem states that in the presence of network partitions, a distributed system must choose between consistency and availability. The relevant insight is not that CAP "kills" decentralization; it is that *global shared-state systems pay CAP costs system-wide.*

13.1 Blockchain: System-Wide Coupling Under Partition

A single global ordering domain pushes consistency/availability trade-offs into one shared fate. When the global system is stressed, the whole system is stressed.

13.2 DSM: Localizing CAP to Each Relationship

DSM does not escape CAP; it *localizes* it. Under partition:

- offline relationships can remain consistent (acceptance happens locally when parties are present),
- online delivery can be delayed without corrupting unrelated relationships,
- acceptance remains receiver-gated: availability of delivery does not imply acceptance into state.

In short: DSM turns CAP from a global tax into a relationship-level design decision.

14 Upper Bounds: "World Computer" Must Mean "World Channels"

14.1 The Correct Upper Bound Metric

A credible world-scale trust substrate maximizes: *how many independent interactions can progress concurrently without forcing unrelated parties to serialize behind them.*

14.2 Blockchain Upper Bound: One Shared Ordering Domain

Blockchains must produce a globally accepted sequence of updates. Even with rollups or sharding, root settlement and data availability remain coupled. Scaling raises ceilings but does not remove the structural bottleneck of shared ordering and shared execution risk.

14.3 DSM Upper Bound: Parallel Relationship Channels

DSM moves from one global pipeline to R relationship pipelines. If each relationship can progress at rate t transitions per unit effort on commodity devices, the system-wide throughput scales with concurrent activity:

$$T_{\text{system}} \approx \sum_{i=1}^{R} t_i,$$

bounded by endpoint compute/bandwidth and storage-node replication capacity. There is no global serialization requirement.

14.4 Walkie-Talkie Analogy (Operational Interpretation)

A blockchain is one shared channel everyone speaks on; DSM is a standard device model where every pair has its own channel:

- same "brand" (shared rules, shared encoding, shared verification),
- different tuned channels (relationship-specific state progressions),
- interoperability without forced global contention.

This is why DSM can plausibly serve as a world-scale substrate: it composes by parallelism, not by global agreement.

15 Downsides and Boundary Conditions of DSM (Real Costs, Not Marketing)

DSM removes blockchain failure modes, but it does not remove reality.

15.1 Composability Becomes Explicit

Global shared execution gives effortless composability at the cost of global coupling. DSM requires explicit multi-step commitment protocols:

- pre-commit forks enumerate outcomes,
- counter-parties select or counter-propose,
- DLVs hold value under deterministic release conditions.

This is a feature (predictability and sovereignty) and a cost (protocol design burden).

15.2 Receiver-Gated Acceptance Is Fundamental

Online delivery can leave envelopes pending indefinitely if a receiver never accepts. This is correct and sovereign, but applications must model it cleanly:

- **Sent** = sender committed and delivered to mailbox.
- **Final** = receiver accepted and advanced state.

15.3 Identity Bootstrapping and Unknown Senders

Rejecting unknown senders by default blocks spam and impersonation, but requires robust identity establishment (contact exchange, attestation, or other bootstraps). This is not optional; it is the price of strict verification.

15.4 Canonical Serialization Must Be Singular

Any divergence between signed bytes and executed bytes is catastrophic. The system must enforce one canonical protobuf representation across:

- sender signing,
- receiver verification,
- core execution.

This is the "narrow waist" of the entire security model.

15.5 Availability Infrastructure Must Be Hardened

Storage nodes are not trusted for correctness, but they are important for availability. Replication, inbox semantics, and denial-of-service resistance must be engineered without turning storage into authority.

16 Conclusion

Blockchain promised decentralized trust by making everyone share a single machine. DSM fulfills the promise by refusing the premise. DSM replaces global consensus and shared execution with relationship-scoped, forward-only deterministic channels whose correctness is enforced at endpoints and whose availability is supported by deliberately non-authoritative storage nodes.

This architecture removes reorg risk, MEV, shared-execution malware surfaces, and congestion externalities by construction. The costs are explicit composability protocols, receiver-gated acceptance, strict identity bootstrapping, and an uncompromising requirement for canonical byte-level equivalence between signing, verification, and execution.

If the goal is a trust substrate shipped on all devices—a standard primitive for authentication, messaging, value transfer, and deterministic conditional agreements—the only scalable path is the one that makes unrelated relationships non-interfering. That is the DSM path.