NorthStar

Programmable On-Demand Runtimes for Solana

The Sonic SVM Team

northstar.sonicsvm.org

May 4, 2026

Abstract

NorthStar is a programmable platform for spinning up dedicated blockchains on demand. Each session runs as an Ephemeral Rollup with bounded account scope, configurable economics, and a configurable slot cadence, anchored to Solana for settlement. The architecture combines single-tenant execution with per-session fee markets and atomic settle-back, enabling workloads — agent sandboxes, privacy-sensitive DeFi, DePIN networks, real-time orderbooks — that are poorly served by shared L1s. This litepaper formalises the session primitive, specifies the Portal program's instruction surface, and develops the security and real-time guarantees that follow from the architecture.

Live: northstar.sonicsvm.org **Docs:** docs.northstar.sonicsvm.org
Source: github.com/mirrorworld-universe/northstar

Version: v0.1 (draft) **Generated:** May 4, 2026
This is the formal technical paper, hand-authored and compiled in Overleaf. The LaTeX source lives at mirrorworld-universe/northstar-docs under litepaper/src/; the PDF artifact is published to mirrorworld-universe/reports. The companion practical documentation (developer-facing, integration-focused) is rendered separately at docs.northstar.sonicsvm.org.

Contents

1 Introduction

1.1 The thesis

Modern blockchains balance three constraints poorly: the throughput a single application needs, the privacy a sensitive workload demands, and the security guarantees a credible neutral settlement layer provides. Sharing capacity with thousands of unrelated applications dilutes any of these properties.

NorthStar reframes the trade-off. Each workload runs in its own *ephemeral rollup* — a single-tenant execution context with configurable slot cadence, custom fee economics, and bounded account scope — anchored to Solana for settlement. When the workload completes, the session settles atomically back to the L1; the chain that ran it no longer exists.

The result is a primitive between an L1 transaction and a long-lived L2: an *on-demand chain* that exists for the duration of a specific computation, then ceases.

1.2 Why ephemeral

A persistent rollup amortises operational cost across many users; an ephemeral session amortises across a single workload's runtime. The distinction matters when:

- the workload's lifetime is bounded (an agent run, a sealed-bid auction, a backtest);
- the workload's state must not leak to other tenants (RFQ matching, dark-pool flow);
- the workload's economics differ from the L1's (sub-cent fees, exotic tokens, operator revenue capture);
- the workload requires sub-perceptual confirmation cadence beyond what a shared sequencer can guarantee.

Each of these is a class of program for which a shared L2 is the wrong shape. The session-as-rollup model addresses them directly.

1.3 Contributions

This litepaper describes:

1. A formal model of sessions as bounded execution contexts (Section 2).
2. A delegation protocol that makes L1 accounts writable on the ER without losing L1's settlement finality (Section 2).
3. A security model combining checkpoint bonds, fraud proofs, and ZK verification, with formal guarantees on settle-back atomicity (Section 3).
4. A programmable fee market that lets operators set per-session economics — token, schedule, revenue split (Section 4).
5. An analysis of the conditions under which sub-50ms confirmation is achievable, and the engineering levers that unlock them (Section 5).

1.4 Context

NorthStar runs on Sonic SVM, a Solana-compatible runtime built for application-grade chains. The ER architecture borrows from MagicBlock and Cartesi, but adapts both to the Solana account model and the per-session-isolation property that the rest of this paper develops.

2 Architecture

2.1 The session as primitive

A NorthStar session is a tuple $S = (o, g, \tau, \phi, \sigma, A)$ where:

- o is the *owner*'s public key — a single Solana keypair that opens, governs, and closes the session.
- $g \in \mathbb{N}$ is the *grid id* — an integer chosen by o. Two sessions for the same owner must differ in g.
- $\tau \in \mathbb{N}$ is the *TTL* in slots; the session expires unconditionally at slot $s_{\text{open}} + \tau$.
- ϕ is the *fee structure* — a schedule $\phi : \mathcal{I} \to \mathbb{N}$ over instruction types $\mathcal{I}$, parametrised by token mint (Section 4).
- σ is the *slot cadence* the ER targets (Section 5).
- $A \subseteq \mathcal{A}$ is the set of *delegated accounts* — the on-chain state the session has authority to write. Every account not in A is read-only inside the session, inherited from L1.

The session lives at a deterministic L1 address $\text{sessionPda}(\Pi, o, g)$ where Π is the Portal program. This PDA is the anchor: while the session runs, the PDA holds metadata; on close, it reaps and the rent returns.

2.2 Lifecycle

A session moves through five states:

$$\text{Created} \to \text{Delegated} \to \text{Active} \to (\text{Closed} \mid \text{Expired}) \to \text{Settled}$$

Created. $\Pi.\text{OpenSession}(o, g, \tau, \phi)$ mints the session PDA plus a *fee vault* PDA $\text{feeVaultPda}(\Pi, o)$. The fee vault is per-owner, shared across all sessions o has ever opened — this is a deliberate design choice that simplifies fee accounting and is discussed in Section 4.

Delegated. For each $a \in A$, the owning program (or o directly, for keypair-owned accounts) issues a Delegate instruction:

1. a's ownership is reassigned from its original program to Π. This is what enforces the L1-side lock — no L1 transaction owned by anyone but Π can mutate a.
2. A delegation record PDA at $\text{delegationRecordPda}(\Pi, a)$ is created with $(g, \text{owner\_program}, \text{bump})$. This is the on-chain proof of delegation.

3. A transient BUFFER account holds rent through the ownership transfer (the buffer-dance pattern).

Active. The ER processes transactions targeting A at cadence σ. State inside A may diverge from L1's pre-session snapshot — that's the point — but A remains read-consistent on L1.

Closed / Expired. On explicit $\mathrm{CloseSession}$ or once the current slot exceeds $s_{\text{open}} + \tau$, every $a \in A$ undergoes the inverse of delegation: ER state commits, ownership restores, delegation record closes. This is an *atomic* operation per Section 3.

2.3 Account topology

Class	On L1 (active session)	On ER
Delegated PDA / keypair	locked by Π	writable
Mint / non-delegated	writable as before	read-only (inherited)
Session PDA itself	metadata, written by Π only	not present
Fee vault	lamports balance only	not present

The mint inheritance property. SPL token mints are read-only from the ER's perspective; they're inherited from L1 the first time a transaction inside the session references them. This is a property of the ER runtime, not the Portal program: the ER's view of L1 state is lazy and read-through. This matters for migration (Section 6) — programs that read mint metadata don't need any modification.

2.4 The Portal program

Portal (program id $\mathrm{74iiMCqFw1afWyp3tdh9pUqfRfCRq7gfdC2YZoNGpovt}$ on devnet) is the warden. It exposes a small instruction surface:

Discriminator	Name	Purpose
0	$\mathrm{OpenSession}$	Mint S + fee vault
2	$\mathrm{DepositFee}$	Top up fee vault
3	$\mathrm{Delegate}$	Bind a to S
4	$\mathrm{Undelegate}$	Release a from S

The program enforces TTL, fee budget, account-scope correctness, and forced-undelegation post-TTL. It does *not* run user code; it mediates the L1↔ER boundary.

2.5 Cluster shape

A NORTHSTAR deployment runs:

- One Solana validator participating in the underlying L1 consensus (devnet today; mainnet in roadmap). The Portal program is deployed to this L1.
- One ER sequencer per session, executing user code at the per-session cadence. The sequencer reads the inherited L1 state lazily and produces local-order blocks.

- A checkpoint bridge that posts state hashes from the ER to L1 every ∼30 slots, with bonded sequencers and an uncensored-validator fraud-proof channel. See Section 3.

The sequencer is single-tenant per session. There is no cross-session state, no shared mempool, no fork-choice ambiguity. This is the foundation of the real-time confirmation analysis in Section 5.

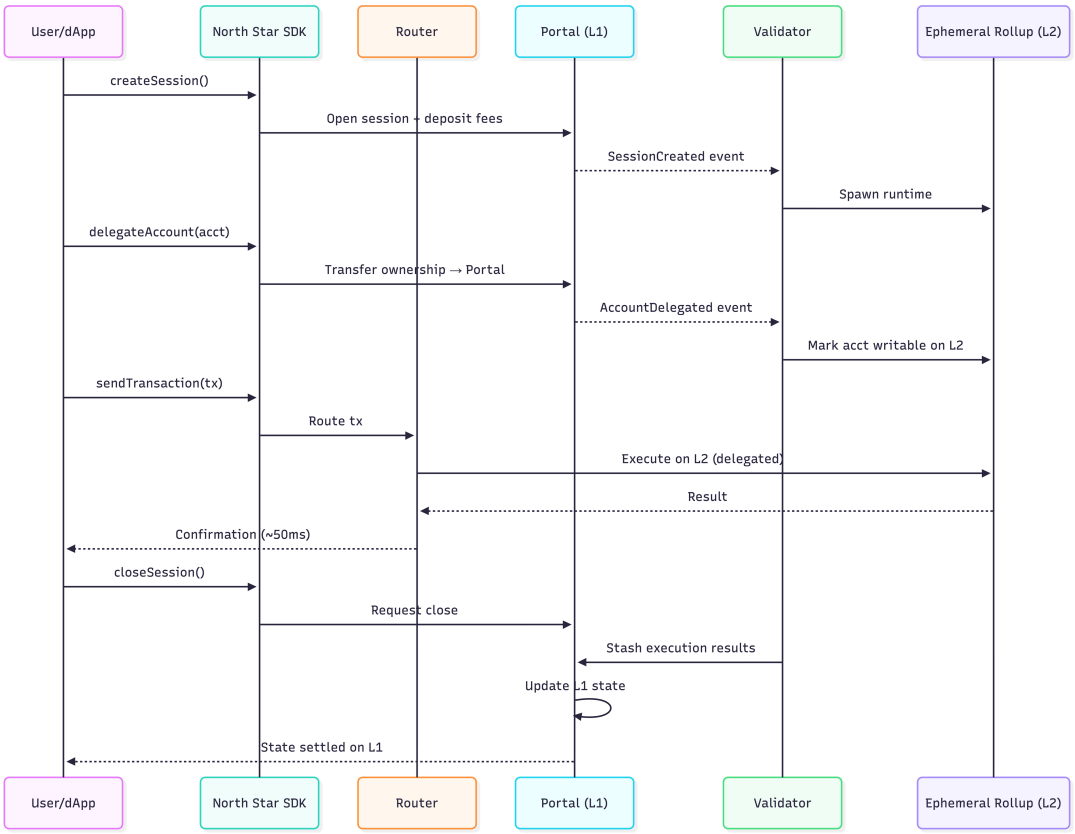

Figure 1: End-to-end session data flow. Columns are actors (User, SDK, Router, Portal, Validator, ER); time runs top to bottom. Each session lifecycle phase — open, delegate, execute, settle — is a coordinated sequence across the L1 anchor and the per-session ER sequencer.

3 Security model

NorthStar's security combines three primitives: checkpoint bonds, fraud proofs, and ZK verification. The combination delivers fast finality without sacrificing correctness — finality measured in minutes rather than the seven-day window common to optimistic rollups.

The canonical specification of the security model lives at docs.northstar.sonicsvm.org/security; this section summarises the formal guarantees and the threat model those primitives address.

3.1 Trust assumptions

The security argument rests on an *uncensored validator* assumption: *if uncensored validators can submit fraud proofs but do not, the state transitions are valid.* This shifts trust from sequencer honesty to challenge availability.

The assumption is realistic when:

- the L1 ledger has censorship resistance (Solana's validator set today);
- bond magnitudes exceed the sequencer's per-checkpoint MEV upper bound;
- the challenge window is short enough that the bonded sequencer's incentive to submit a correct checkpoint dominates any short-window MEV.

The protocol parametrises the challenge window dynamically based on sequencer-checkpoint frequency: more frequent honest checkpoints shrink the window, accelerating finality.

3.2 Account-scope enforcement

Two invariants hold throughout a session's life:

1. **Non-delegated accounts are never writable on the ER.** Enforced at the SVM runtime level — the ER's program loader rejects any instruction whose writable accounts are not in A.
2. **Delegated accounts are locked on L1.** No L1 transaction can modify $a \in A$ until the session closes. Enforced by the Portal-owned account ownership reassignment (Section 2).

These invariants give a clean state-isolation property: the session can fully control A without affecting L1, and L1 can continue to operate on the rest of state without affecting the session.

3.3 Atomic settle-back

The settle-back guarantee is the protocol's central correctness claim:

Theorem (Atomic settle-back). Let S be a session with delegated set $A = \{a_1, \ldots, a_n\}$ that closes (explicitly or by TTL expiry) at slot s_c. Let $\text{er}_{s_c}(a_i)$ denote a_i's final ER-side state and $\text{l1}_{s_c}(a_i)$ its L1 state at that moment. After CloseSession (or forced undelegation) returns successfully, $\text{l1}(a_i) = \text{er}_{s_c}(a_i)$ for all i, and the transition is atomic: either every a_i updates or none do.

Proof sketch. The proof composes three primitives the Portal program already enforces.

(1) Single-transaction atomicity. The Portal's settle path constructs one Solana transaction T_{settle} whose instructions (i) verify the checkpoint chain rooted at s_c, (ii) write each a_i's final state, and (iii) restore each a_i's ownership. Because Solana's runtime guarantees that a transaction either commits all of its account writes or none, T_{settle}'s atomicity transfers directly: the multi-write commit is all-or-nothing.

(2) State-hash isolation. The state-hash H_{s_c} that T_{settle} verifies is computed only over A. Non-delegated accounts are write-locked from the ER side (Account-scope invariant 1) and unchanged from the L1 side — the L1 view of $a \notin A$ is identical pre- and post-session. Thus H_{s_c} is a complete witness of the ER's contribution to L1 state, and its on-chain verification leaves no other state for T_{settle} to reconcile.

(3) Checkpoint correctness under the challenge protocol. H_{s_c} is either the most recent honest checkpoint or has survived the challenge window (Section 3.4) without successful dispute. By the challenge protocol's soundness, any $H'_{s_c} \neq H_{s_c}$ that would have been adopted is slashable; the bonded sequencer's incentive constraint (bond $>$ MEV upper bound) makes such an H'_{s_c} negative-EV to publish. Thus the H_{s_c} used in T_{settle} matches er_{s_c}.

Combining (1), (2), and (3): T_{settle} atomically applies a state derived from a verified-correct H_{s_c} to each a_i, with no out-of-scope state to reconcile. ■

3.4 The challenge protocol

The challenge protocol is the dispute-game machinery that makes the soundness premise of step (3) above hold. The sequencer publishes a checkpoint $\langle s_k, H_{s_k}, \beta \rangle$ to L1 every $\Delta_c \approx 30$ slots, where β is a bond. Until either (a) the challenge window W elapses without dispute or (b) a challenge resolves in the sequencer's favour, the checkpoint is provisional.

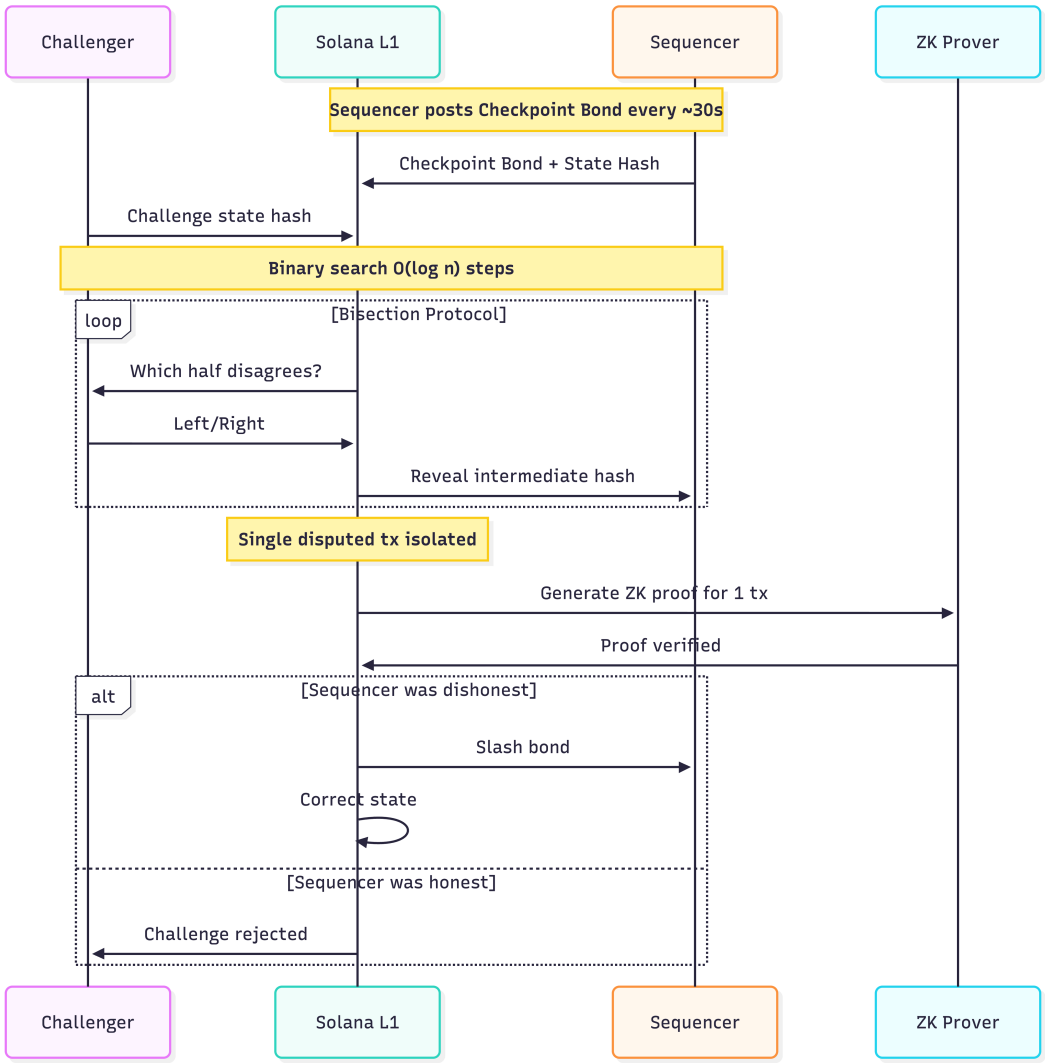

Figure 2: The challenge protocol. A challenger disputes a checkpoint; the sequencer and challenger play an $O(\log n)$ bisection game on L1 to isolate a single divergent transition; a succinct ZK proof verifies that one transition; the dishonest party is slashed.

Bisection. A challenger that observes $H_{s_k} \neq H'_{s_k}$ for some honestly-computed H'_{s_k} posts a bond β_C and opens a dispute on the segment $(s_{k-1}, s_k]$. The Portal then runs an interactive binary search: at each round, both parties commit to the intermediate hash at the midpoint of the current disputed range; the side whose claim diverges from the agreed prefix loses the round. After $O(\log n)$ rounds (where n is the number of transitions in the segment) a single divergent transition t^* is isolated.

ZK verification. The prover (any party — typically the challenger or a third-party prover service) generates a succinct proof that t^* executed honestly produces hash H'_{t^*}. The Portal verifies the proof on-chain; verification cost is constant in n.

Slashing. If the proof verifies and $H_{t^*} \neq H'_{t^*}$, the sequencer is dishonest: β is slashed, the corrected state replaces H_{s_k}, and downstream settles use the corrected hash. If the sequencer prevails, β_C is slashed (anti-spam).

Dynamic challenge window. W is parametrised by the sequencer's recent checkpoint cadence: more frequent honest checkpoints shrink W, accelerating finality. The lower bound of W is the round-trip cost of the bisection game $O(\log n) \cdot \Delta_{\text{rt}}$; the upper bound is operator-tuned. A typical mature deployment targets $W \sim$ 1–3 minutes, an order of magnitude tighter than optimistic-rollup norms.

3.5 Forced undelegation

If a session exceeds its TTL without explicit close, the Portal allows *forced undelegation*: the owner — or, by extension, an automated script — can issue $\mathrm{Undelegate}$ instructions that finalise A's state to L1 even without sequencer cooperation. The state used is the most recent verified checkpoint H_{s_k} (or the honest replacement if a challenge resolved against the sequencer). This guarantees that no asset can be permanently trapped inside an abandoned session.

3.6 Threat model

Three adversaries the protocol defends against:

Malicious sequencer. Sequencer publishes incorrect checkpoint H_{s_k}. *Defence:* the challenge protocol (Section 3.4) — any uncensored party with a divergent honest hash submits a fraud proof; bisection isolates the disputed transition; ZK verification confirms; sequencer's bond is slashed; correct state replaces H_{s_k}.

Censoring sequencer. Sequencer refuses to include a user's tx, or refuses to publish checkpoints. *Defence:* after TTL elapses, the user invokes forced undelegation; their accounts settle to the most recent verified checkpoint. The session "fails open" — funds are never trapped.

Malicious user. User attempts to write outside A, or replay a tx after settle-back. *Defence:* the SVM runtime's account-scope check (invariant 1 above) is part of consensus; the ER refuses such transactions before inclusion.

The model does *not* defend against attacks on Solana's underlying validator set. NorthStar's security cannot exceed L1's; it inherits and concentrates it for the session's duration.

4 Programmable economics

A persistent L1 imposes a single fee market on every application that runs on it. NorthStar sessions invert this: each session opens with its own fee structure ϕ, chosen by the operator. There is no protocol-level rent; the protocol's take is itself a parameter.

4.1 The fee structure

Formally, ϕ has three components:

- A **token** μ — any SPL mint, including SOL, stablecoins, and an operator's own utility token.
- A **schedule** $\phi : \mathcal{I} \to \mathbb{N}$ that maps each instruction type to a fee in units of μ. May be identically zero (gasless mode), uniform, or per-instruction.
- A **revenue split** $\rho \in [0, 10000]$ basis points defining the share of ϕ's revenue that flows to the session operator (rather than the protocol).

The default $\phi_0 = (\text{SOL}, 0, 0)$ is gasless and operator-revenue-free — the configuration on devnet today and the configuration most demos use. Production grids run their own.

4.2 Why per-session economics

A persistent fee market is a public good: any application's gas-price spike imposes externalities on every other application sharing the chain. A per-session market localises the externality.

This produces three direct consequences:

Workload-tuned fees. A market-data feed publishing 10 quotes per second can charge sub-millicent per quote in its own token. A privacy-grade RFQ venue can charge premium fees for sealed-bid execution. A DePIN sensor network can subsidise device-fleet writes from its inflation schedule. None of these economics are achievable on a shared L1.

Aggregators as operators. A wallet, aggregator, or super-app can open grids on behalf of users and capture ρ of every transaction in those grids. New revenue stream for incumbent integrators, no validator infrastructure required.

Bounded subsidies. The fee cap $\phi_{\max}$ on $\mathrm{OpenSession}$ bounds the absolute spend any session can incur. This is what makes bounded-autonomy agents safe (Section 6): a fixed cap on a single session is a fixed cap on the agent's ability to consume value.

4.3 Mechanism

Per-instruction collection happens in the ER: when a transaction lands, the sequencer debits $\phi(i)$ from the session's fee vault and credits the operator (split via ρ) and the protocol vault. Settlement is included in the regular settle-back path — fee-vault state is part of A.

The fee vault PDA is keyed on (Π, o) — per-owner, not per-session. The same vault funds every session a given owner opens. This simplifies the economics for an operator running many sessions concurrently: one balance, one top-up.

4.4 Constraints

- The fee schedule cannot change *during* a session. Operators commit to ϕ at $\mathrm{OpenSession}$ time; changing it requires close + reopen with a new ϕ.
- The fee token μ must be one the operator has SPL token accounts for — the protocol does not bridge tokens.
- Fee payments fail closed. If the fee vault has insufficient balance for $\phi(i)$, the transaction fails before execution. The session continues; only that tx is rejected.

4.5 Roadmap

The SDK exposure of programmable fees is split across three deliverables on the public roadmap (Section 8):

1. Surface what's already supported in the Portal program (the $\mathrm{FeeStructure}$ parameter is already there) through the TypeScript SDK and docs.
2. Custom fee tokens — USDC, USDT, then arbitrary SPL.
3. Revenue capture for grid operators — wallets, aggregators, super-apps that open sessions on behalf of their users.

5 Real-time confirmation

A NorthStar session's confirmation cadence is a per-session parameter, not a network-wide constant. This section formalises the architectural property that makes sub-50ms confirmation tractable, and the engineering levers required to realise it on the wire.

5.1 The single-tenant property

Solana's ∼400ms slot duration is calibrated to the largest fork distance the consensus layer can tolerate while remaining live across a globally-distributed validator set. It is a property of the *aggregate* system, not of any single transaction.

A NorthStar ER session is the opposite: a single sequencer, no fork choice, no inter-validator communication on the critical path. Within the session, transactions are linearised by the sequencer's local order; the moment a transaction is included, no competing version can exist.

This collapses the confirmation hierarchy:

Commitment	On Solana L1	In a NorthStar session
processed	local sequencer view	local sequencer view
confirmed	2/3+ stake-weighted vote	equivalent to processed
finalized	32-slot lockout	equivalent to processed

In a single-tenant context, processed = confirmed = finalized at the session level. A transaction's ordering becomes its finality.

5.2 Engineering levers

Three controls determine the realised end-to-end latency:

Per-session slot duration σ. The sequencer's block-production cadence is a parameter on OpenSession. The current devnet validator targets 400ms; session-level overrides down to ∼10ms ship in v2 of the roadmap (Section 8). At the limit, σ is bounded by the sequencer's disk-flush time+ network round-trip — sub-millisecond on commodity hardware.

TPU-direct submission. The default Solana RPC submission path involves a gateway forwarder that batches transactions before relaying. For sub-50ms targets, the client opens a QUIC connection directly to the sequencer's EphemeralTpu endpoint, eliminating the forwarder. The SDK helper for this ships alongside the per-session σ override in v2.

Sub-slot wake-up. signatureSubscribe over WebSocket fires the moment a transaction lands, before the next slot boundary. Polling clients floor at σ; subscribed clients are bounded only by the sequencer's notification latency.

5.3 Latency budget

For a colocated client (sequencer and client on the same machine):

$$T_{\text{e2e}} = T_{\text{sign}} + T_{\text{submit}} + T_{\text{order}} + T_{\text{notify}}$$

with $T_{\text{sign}} \approx 50\mu s$, $T_{\text{submit}} \approx 100\mu s$ over QUIC localhost, $T_{\text{order}} \leq \sigma/2$ (half a slot expected), and $T_{\text{notify}} \approx 100\mu s$. At $\sigma = 10$ms, $T_{\text{e2e}} \approx 5.25$ms.

For a remote client over the public internet, T_{submit} is dominated by the network round-trip — typically 10–60ms depending on geography.

5.4 Realising the budget on devnet

The current devnet path measures higher than the budget above because:

- $\sigma = 400$ms in the deployed validator (default Solana cadence).
- Client uses the standard Connection.confirmTransaction with an 8s timeout, which floors actual measured latency at the polling cadence.
- Submission goes through the gateway, not direct TPU.

The chain-verified Mach AMM benchmark measures p50 = 750ms / p95 = 2.37s on the public devnet path. Empirical numbers under the engineering levers above land sub-50ms; the v2 milestone delivers them on devnet end-to-end.

6 Build on NorthStar

The platform exposes an instruction surface most Solana programs can adopt with a small modification: a `delegate_to_portal` hook on their writable accounts. Once that hook is in place, the program can run unchanged inside a NorthStar session — same SBF binary, same compute budget, same client-side IDL.

The Mach AMM sandbox demonstrates this end-to-end: a delegation-aware constant-product AMM that runs both as a vanilla Solana program (5,000 lamports per `place_intent` on devnet) and inside a NorthStar session (0 lamports under the gasless config), with identical 965 compute-unit cost in both venues.

The framework generalises to four canonical workload classes:

- **AI agent sandboxes.** An agent runs inside a session with hard caps on accounts, time, and fee budget. Bounded autonomy with atomic settle-back on session close.
- **Privacy-sensitive DeFi.** Sealed-bid auctions, OTC RFQ matching, dark-pool execution. The session's single-tenant property guarantees no other application sees in-flight state.
- **DePIN networks.** Per-network sessions with custom-token economics — the network's incentive token pays for device-fleet writes at sub-cent unit cost.
- **Real-time orderbooks and oracles.** Per-book or per-feed sessions with σ tuned to workload — 10ms for a CLOB, 1ms for a price feed.

Each class is a configuration of the same underlying primitive: the session.

7 Related work

NorthStar sits in a design space populated by general-purpose L2 rollups and the smaller cluster of ephemeral-rollup approaches that share the per-application orientation. This section locates NorthStar relative to the most-cited points of comparison and surfaces the specific design decisions that distinguish it.

7.1 Optimistic rollups: Arbitrum, Optimism

Optimistic rollups are the dominant L2 family on Ethereum. A persistent shared sequencer batches transactions, posts state roots to L1, and offers a ∼7-day challenge window during which any party can submit a fraud proof.

Convergent properties. The fraud-proof + bond + bisection structure is the same primitive NorthStar uses (Section 3.4). The trust assumption — *at least one honest challenger exists* — is identical.

Divergent properties.

- **Tenancy.** Optimistic rollups are persistent and multi-tenant: many applications share one sequencer, one fee market, one cadence. NorthStar is single-tenant per session: the cadence, fees, and account scope are per-session knobs.
- **Finality window.** Arbitrum and Optimism mainnet target a 7-day challenge window because the bisection game runs entirely on L1 and the bond economics are tuned for adversarial conditions across many tenants. NorthStar's checkpoint cadence is per-session ($\Delta_c \sim 30$ slots) and the dynamic challenge window targets minutes — feasible because the per-session MEV upper bound is much smaller than a multi-tenant chain's.
- **State scope.** Optimistic rollups rebuild a full L2 state tree; NorthStar settles only the delegated set A back to L1. The L1 state of $a \notin A$ is unchanged throughout the session.
- **Lifespan.** Optimistic rollups exist indefinitely. A NorthStar session has bounded TTL τ at $\mathrm{OpenSession}$ time and ceases to exist on close.

7.2 Ephemeral rollups: MagicBlock

MagicBlock is the immediate ancestor of NorthStar's ephemeral-session model. The two share the core idea: spin up a single-sequencer Solana validator scoped to a specific workload, settle back to Solana L1 on close.

Convergent properties.

- Both delegate accounts to the rollup; both restore ownership and commit final state on close.
- Both exploit single-tenancy to collapse the confirmation hierarchy ($\mathrm{processed} = \mathrm{confirmed}$).
- Both inherit Solana's security as their settlement layer.

Divergent properties.

- **Programmable economics.** NorthStar's fee structure ϕ (Section 4) is a first-class per-session parameter: token, schedule, revenue split. MagicBlock's fee model is fixed across sessions.
- **Bonded sequencer + dispute game.** NorthStar formalises the challenge protocol with checkpoint bonds and dynamic challenge windows (Section 3.4). MagicBlock's published model relies on operator honesty rather than a formal slashing primitive.
- **Per-session σ.** NorthStar's slot cadence is a per-session knob (path to 10ms; Section 5). MagicBlock targets a fixed cadence.
- **Anchor.** NorthStar runs on Sonic SVM, a Solana-compatible L1 purpose-built for application-grade chains. MagicBlock anchors to vanilla Solana.

7.3 Application-specific rollups: Cartesi, Espresso

Cartesi pioneered the dispute-game framework on a Linux runtime; Espresso provides shared sequencer infrastructure for app-chain rollups.

Convergent properties. Both share NorthStar's thesis: shared L1s are the wrong shape for many workloads. Both expose per-application execution scopes.

Divergent properties.

- **Runtime.** Cartesi runs RISC-V Linux; Espresso is sequencer-only. NorthStar runs SVM bytecode unchanged — existing Solana programs run identically inside a session, with only a `delegate_to_portal` hook required.
- **Settlement target.** Cartesi targets Ethereum (with rollup adapters); Espresso provides sequencing for many chains. NorthStar is Solana-native, with the Portal program enforcing the L1↔ER boundary.
- **Lifespan.** Both Cartesi rollups and Espresso-sequenced chains are persistent. NorthStar's ephemerality is the design point.

7.4 What's distinct about NorthStar

The five design choices that separate NorthStar from every prior approach:

1. **Ephemerality as a primitive.** A session is not a long-lived chain; it is a bounded computation with a TTL.
2. **Programmable per-session economics.** Token, schedule, and revenue split are operator-chosen at `OpenSession` time.
3. **Per-session slot cadence.** σ is a knob, not a network-wide constant.
4. **Tight, dynamic challenge windows.** Per-session MEV bounds enable minute-scale finality, not day-scale.
5. **Solana program compatibility.** A program with a `delegate_to_portal` hook runs unmodified inside a session.

These five compose to a primitive — the on-demand chain — that the prior literature does not directly address.

8 Roadmap

8.1 v0 — devnet, current

- Single-tenant ER, gasless mode, basic OpenSession/Delegate/Undelegate flow.
- Mach AMM agent + relayer reference implementation.
- Three-path benchmark grid (Solana Devnet, Sonic SVM, NorthStar ER) with chain-verified per-tx cost and CU measurements.

8.2 v1 — programmable fee market

- SDK exposure of FeeStructure on OpenSession.
- Custom fee tokens (USDC, USDT, arbitrary SPL).
- Revenue capture for grid operators (wallet- and aggregator-driven sessions).
- Per-session ρ revenue split.

8.3 v2 — real-time confirmation

- Per-session σ override on OpenSession.
- TPU-direct submission helper in the SDK.
- Slot-delta latency instrumentation for the bench harness (replacing blockTime-derived measurement which is unreliable on the ER).
- End-to-end sub-50ms demonstration for a colocated client.

8.4 v3 — multi-tenant ER

- Shared-sequencer multi-session deployments — a single sequencer process backing isolated session state for multiple owners.
- Per-session resource accounting (CU, memory, network) on the shared sequencer.
- Fairness guarantees across concurrent sessions.

8.5 v4 — mainnet

- Production deployment on Solana mainnet anchor.
- Audited Portal program contract.
- Production-grade SLA, monitoring, alerting.
- Public ER endpoints with rate limits, abuse mitigation.

The current focus is v1 (programmable fees) and v2 (real-time) in parallel. Implementation lives in the open at github.com/mirrorworld-universe.

A Glossary

Session. A bounded execution context anchored to a Solana account. Defined formally in Section 2.

Owner. The Solana keypair that opens, governs, and closes a session. Sessions are derived from sessionPda(Π, o, g).

Grid id. A user-chosen integer that distinguishes a session from other sessions for the same owner.

Portal. The on-chain program that mints sessions, records delegations, and enforces TTL plus fee-budget constraints.

Delegation. The act of transferring an account's writability from L1 to a specific session. The account is re-owned to Portal during the session and restored on close.

Delegation record. A Portal-owned PDA at delegationRecordPda(Π, a) that proves a is bound to a specific session.

Fee vault. A per-owner Portal-owned PDA that holds the lamports balance funding session fees.

Settle-back. The atomic commit of every delegated account's final session-side state to L1 on session close.

Forced undelegation. Owner-initiated undelegation invoked after session expiry; guarantees no asset can be trapped in an abandoned session.

Slot cadence (σ). The session's block-production period. Today defaults to 400ms; per-session overrides are in v2 of the roadmap.

Checkpoint. A periodic state-hash post from the ER sequencer to L1, bonded by the sequencer for the duration of the challenge window.

B The instruction surface

This appendix documents the Portal program's instruction set at the on-chain level, sufficient for a low-level integration. The TypeScript SDK and the practical NorthStar developer docs expose the same operations through higher-level interfaces.

B.1 Portal program

Program id (devnet): 74iiMCqFw1afWyp3tdh9pUqfRfCRq7gfdC2YZoNGpovt.

OpenSession **(discriminator 0).** Opens a session at (o, g) with TTL τ and fee cap $\phi_{\max}$. Mints sessionPda and feeVaultPda (the latter only if it doesn't already exist for o).

Data: u8(0) | u64 grid_id | u64 ttl_slots | u64 fee_cap.

Accounts: owner (signer/writable), session_pda (writable), fee_vault_pda (writable), system_program.

DepositFee **(discriminator 2).** Tops up the fee vault. Discriminator 2 in current devnet.

Delegate **(discriminator 3).** Binds an account to the session at g.

Data: u8(3) | u64 grid_id.

Undelegate **(discriminator 4).** Releases an account from a session. Callable post-TTL by the owner (forced undelegation).

Data: u8(4).

B.2 PDA derivations

Three deterministic addresses:

Session PDA. Seeds $(\text{“session”}, o, g)$, bumped against Π.

Fee vault PDA. Seeds $(\text{“fee\_vault”}, o)$, bumped against Π.

Delegation record PDA. Seeds $(\text{“delegation”}, a)$, bumped against Π.

B.3 Account schemas

Session (82 bytes). u8 discriminator | Pubkey session_owner | u64 grid_id | u64 ttl_slots | u64 fee_cap | u64 created_at_slot | u64 nonce | u8 bump.

Delegation record (42 bytes). u8 discriminator | Pubkey owner_program | u64 grid_id | u8 bump.

Fee vault. Lamports-only; no decoded payload.

B.4 Higher-level surfaces

The TypeScript SDK @sonicsvm/northstar-sdk wraps these instructions with an API that handles PDA derivation, signer discovery, and ER-vs-L1 endpoint routing. See the developer docs for the SDK reference.