Project Aegis: Deterministic AI Agents for Solana using Raiku

Aegis Framework Design

October 21, 2025

Abstract

Today, on-chain automated strategies are fundamentally reactive. Bots and agents compete in a high-speed, probabilistic auction for blockspace, suffering from execution uncertainty, unpredictable timing, and MEV. This severely limits their strategic depth.

Project Aegis is a modular, open-source framework that enables developers to build and deploy proactive, deterministic AI agents on Solana. By leveraging Raiku's Ahead-Of-Time (AOT) and Just-In-Time (JIT) execution guarantees, Aegis transforms blockspace from a chaotic auction into a predictable, plannable resource. This allows agents to move beyond simple, reactive arbitrage and execute complex, time-based strategies with 100% certainty.

1. The Problem: The Wall of Uncertainty

Any automated agent on Solana today, whether for DeFi, gaming, or tooling, is crippled by a "wall of uncertainty." Key dimensions of this wall include:

- **Execution Uncertainty.** Transactions can and do fail during network congestion (e.g., a popular mint), rendering a well-timed strategy useless.
- **Timing Uncertainty.** An agent cannot guarantee when its transaction will land. It can only "hope" it lands in the next slot, but it could just as easily be multiple slots later.
- **Cost Uncertainty.** Priority fees are opaque and volatile, forcing agents into a high-stakes, expensive guessing game to ensure inclusion.
- **Atomicity Risk.** Complex strategies requiring multiple transactions (e.g., Update Oracle $\rightarrow$ Liquidate Position) are high-risk. The first transaction may succeed while the second fails, causing direct financial loss.

These uncertainties force on-chain automation into a reactive, high-speed paradigm, preventing the development of sophisticated, institutional-grade applications.

2. The Solution: The Aegis Framework

Aegis is a simple, modular framework that abstracts away execution complexity. It provides developers a clean interface to Raiku's primitives so they can focus on strategy logic rather than retry and inclusion logic.

The framework consists of three core components:

2.1 The Signal Processor (The "Brain")

This is the customizable, pluggable brain of an agent. Developers implement strategy logic here.

- **Function.** Ingests any combination of on-chain (e.g., pool liquidity, oracle prices) and off-chain (e.g., economic calendars, news APIs) data.
- **Output.** Emits a concise instruction such as "execute Strategy_A at timestamp_X" or "execute Strategy_B now." The output is intentionally simple and declarative.

2.2 The Action Template (The "Logic")

This component translates a strategy signal into a concrete, atomic set of on-chain instructions.

- **Function.** Action Templates are pre-defined, reusable bundles of transactions. A developer can create a template for a multi-leg arbitrage, a liquidity rebalancing, or a token mint.
- **Example.** A `liquid_stake_arbitrage` template might contain three ordered transactions:
  1. Swap SOL for mSOL on Orca.
  2. Swap mSOL for SOL on Raydium.
  3. Deposit profit (SOL) into a vault.
- **Raiku's Role.** The framework ensures the template is formatted as an atomic bundle. Raiku guarantees ordered, all-or-nothing execution.

2.3 The Aegis Conductor (The "Hands")

This is the execution layer. It translates agent intent into guaranteed actions by interacting with Raiku's slot market.

The Conductor exposes two primary methods (illustrative signatures):

`conductor.schedule(timestamp, action_template)` Used when the Signal Processor targets a specific, known-in-advance event (e.g., a token unlock at 08:00:00 UTC). The Conductor purchases an AOT slot reservation for that exact timestamp, turning the transaction from a hope into a booking.

`conductor.execute_now(action_template, priority)` Used for opportunistic, high-urgency strategies (e.g., sudden arbitrage). The Conductor submits the action template to Raiku's JIT slot auction to guarantee inclusion in the very next block, bypassing the public mempool.

Critically, the Aegis Conductor offloads retry logic to Raikus Ackermann Node. The framework receives a deterministic "success" or "fail" receipt from Raiku, freeing the agent from complex retry loops.

3. Use Cases: From Theory to Application

Aegis enables deterministic agents across many domains. Two representative use cases illustrate its value.

Use Case 1: The Deterministic Token Launch (AOT Scheduling)

The Problem. Token launches are chaotic. Teams struggle to add initial liquidity at a precise time while sniping bots spam the network, causing failed transactions and extreme volatility.

The Aegis Solution.

- **Signal.** The Signal Processor is configured for the public launch time (e.g., "Friday, 17:00:00 UTC").
- **Template.** A 3-step launch Action Template is defined:
  1. `create_pool`
  2. `add_initial_liquidity`
  3. `lock_lp_tokens`
- **Conductor.** Days in advance, the Aegis Conductor calls `conductor.schedule("Friday, 17:00:00 UTC", launch_template)`.
- **Raiku Guarantee (AOT).** Raiku's AOT reservation ensures the 3-transaction bundle executes atomically and in order in the exact slot corresponding to 17:00:00 UTC. The launch is predictable, fair, and protected from front-running.

Use Case 2: Risk-Free Liquidations (JIT Auction & Atomicity)

The Problem. A lending protocol's liquidation bot must update an on-chain price oracle before liquidating a position. It risks the oracle update succeeding while the liquidation fails, costing the protocol money.

The Aegis Solution.

- **Signal.** The Signal Processor monitors price feeds and identifies an urgent liquidation.
- **Template.** A 2-step `liquidate` Action Template:
  1. `update_oracle_price(BTC, $60,000)`
  2. `liquidate_position(user_account_XYZ)`
- **Conductor.** The Aegis Conductor immediately calls `conductor.execute_now(liquidate_template, high_priority)`.
- **Raiku Guarantee (JIT).** The bundle is submitted to Raiku's JIT auction, guaranteeing inclusion in the next block and atomicity. If the liquidation fails, the oracle update is reverted, eliminating the protocol's financial risk.

4. Why It Matters for Raiku

Project Aegis is an enabling layer for a broader ecosystem. It directly augments Raiku's mission in several ways:

- **Abstracting Primitives.** Aegis makes Raiku's powerful AOT/JIT features accessible to a wide developer audience, lowering integration friction and accelerating adoption.
- **Showcasing Determinism.** It demonstrates use cases that require deterministic execution (beyond HFT), providing a blueprint for complex automated systems in AI, DeFi, and DePIN.

- **Enabling New Markets.** By enabling deterministic agents to plan and purchase blockspace, Aegis turns agents into primary consumers of Raiku's slot marketplace, creating a strategy-driven economy for blockspace.

5. Appendix: Quick Comparison

Dimension	**Traditional Agents**	**Aegis + Raiku**
Execution certainty	Probabilistic	Deterministic (AOT/JIT)
Timing control	Slot-approximate	Exact (AOT) or next-slot (JIT)
Atomicity	Risk of partial failure	All-or-nothing bundles
Retry logic	Handled by agent	Offloaded to Raiku Ackermann Node
Cost predictability	Volatile	Market-driven, plannable