JITOS State of the Union Report: The Convergence

James Ross & The AI Engineering Team

December 26, 2025

Abstract

This report serves as the definitive architectural blueprint for the **JITOS (Just-In-Time Operating System)** consolidation. It details the strategic pivot from a fragmented polyglot stack (JS/Rust/Go) to a unified **Rust Monorepo**. We provide an exhaustive specification of the repository structure, crate boundaries, type systems (SLAPS v2, HTN v1), and the deterministic execution pipeline. We align every software component with the theoretical foundations laid out in AION Papers I–VI, identifying drift and prescribing corrective implementation. Finally, we formalize the "Hybrid Purity" model, defining the boundary between Graph Rewriting (Control Plane) and Imperative Simulation (Data Plane).

Contents

1 Executive Summary: The Two-Kernel Problem

JITOS faces a "Split-Brain" divergence:

1. **The Soft Kernel (JavaScript):** Located in 'flyingrobots.dev'. It successfully implements the *Operating System* semantics (Daemon, RPC, WAL, Time Travel UI) but lacks computational rigor. It relies on 'Map' based indexing and simple sorts, limiting it to $< 5,000$ nodes.
2. **The Hard Kernel (Rust):** Located in 'echo'. It implements the *Physics* of determinism (Radix Sort $O(n)$, Bit-level Footprints, Strict Types) but lacks a body (UI, RPC, Persistence).

The Resolution: We will perform a "Brain Transplant." The Rust Kernel ('echo') will become the authoritative core of the JITOS Monorepo. The JavaScript codebase will be demoted to a "Shell"—a pure View/IO layer that renders the state emitted by the Rust/WASM kernel. The Go Planner will be retired and ported to Rust to eliminate serialization overhead.

2 Repository Architecture: The Rust Monorepo

We define a unified workspace 'jitos' organized by functional layer.

2.1 Directory Structure

```
jitos/
|-- Cargo.toml                # Workspace Root
|-- docs/                     # RFCs, Specs, and Papers (LaTeX/Markdown)
|   |-- ARCH/                 # Architecture Decision Records
|   |-- RFC/                  # Request for Comments
|   `-- REPORTS/              # SOTU and periodic audits
|-- schema/                   # THE LAW (Language-Agnostic Types)
|   |-- slaps.v2.cddl         # System Level Action Protocol (Concise Binary Object Rep)
|   |-- htn.v1.cddl           # Hierarchical Task Network definitions
|   `-- graph.v1.cddl         # WARP Graph Node/Edge schemas
|-- crates/                   # THE ENGINE (Rust)
|   |-- jitos-core/           # Shared Types, Traits, Error definitions
|   |-- jitos-graph/          # WARP Graph Data Structure (Petgraph/Custom)
|   |-- jitos-scheduler/      # The Echo Scheduler (Radix Sort, Footprints)
|   |-- jitos-policy/         # Rhai Host, Policy Engine, Sandbox
|   |-- jitos-planner/        # HTN Planner (Ported from Go)
|   |-- jitos-provenance/     # Shiplog (WAL), Merkle Hashing, Receipting
|   |-- jitos-resilience/     # (Was Ninelives) Deterministic I/O Patterns
|   |-- jitos-io/             # Port Adapters (Kafka, HTTP, etc.)
|   |-- jitos-daemon/         # The Native OS Process (jitd binary)
|   `-- jitos-wasm/           # The Browser Bridge (wasm-bindgen)
|-- shell/                    # THE VIEW (JS/React - flyingrobots.dev)
|   |-- src/
|   |   |-- warp/             # Legacy JS Kernel (To be deleted)
|   |   `-- daemon/           # Worker wrapper for WASM
|   `-- public/jitd.wasm      # Compiled Artifact
`-- tools/                    # CI/CD, Benchmarks, Fuzzers
```

3 Crate Specification: The Core Modules

This section details the responsibility and internal structure of each crate.

3.1 `jitos-core`: The Law

Role: Defines the vocabulary of the universe. No logic, only Types. **Paper Alignment:** Paper VI (ABI), Paper I (Structure).

```
// src/slaps.rs
#[derive(Serialize, Deserialize, Debug, Clone)]
pub enum Slap {
    // Structural
    CreateNode(Node),
    DeleteNode(NodeId),
    Connect(Edge),
    // Logic
    InvokeScript { script_id: Hash, args: Vec<Value> },
    // System
    SetTime { tick: u64, dt: f64 },
    Snapshot,
}

// src/receipt.rs
#[derive(Serialize, Deserialize, Debug, Clone)]
pub struct Receipt {
    pub tick: u64,
    pub state_hash: Hash, // BLAKE3
    pub events: Vec<Event>,
    pub signature: Option<Signature>, // Ed25519
}
```

3.2 `jitos-graph`: The State

Role: In-memory graph database optimized for diffing and hashing. **Paper Alignment:** Paper I (WARP Graph).

- **Storage:** Adjacency List with ‘slotmap‘ for $O(1)$ access.
- **Content Addressing:** Every node has a ‘hash‘ computed from its canonical serialization.
- **Recursive Attachments:** Nodes can contain ‘GraphId‘ references, loading sub-graphs lazily.

3.3 `jitos-scheduler`: The Choice

Role: Deterministic ordering of concurrent intent. **Paper Alignment:** Paper II (Dynamics, Echo).

- **Modules:** ‘radix‘, ‘footprint‘, ‘queue‘.
- **Algorithm:** 1. Receive ‘Vec<Proposal>‘. 2. Sort by ‘(ScopeHash, RuleID, Nonce)‘ using **MSD Radix Sort**. 3. Iterate sorted list. 4. Check ‘Footprint‘ (Read/Write sets) against ‘ActiveSet‘. 5. If collision → Reject. Else → Accept.

- **Output:** ‘Batch‘ (guaranteed non-interfering).

3.4 `jitos-policy`: The Logic

Role: Hosting user-defined behavior in a sandbox. **Paper Alignment:** Paper VI (Agents).

- **Engine:** Rhai (embedded scripting language for Rust).
- **Sandbox:** Scripts have NO access to IO, Network, or Time. They only see ‘Graph‘ and ‘Args‘.
- **Determinism:** Instruction metering to prevent infinite loops.

3.5 `jitos-provenance`: The Log

Role: The source of truth. **Paper Alignment:** Paper III (Holography).

- **WAL Format:** Binary stream of ‘Receipt‘ structs.
- **Indexing:** Maintains ‘Tick -> FileOffset‘ index.
- **Hashing:** Computes the Merkle Root of the history.

3.6 `jitos-resilience`: The No-Privilege Boundary

Role: Implementation of ARCH-0009 (No Privileged Side Effects). **Origin:** Ported from `ninelives`.

We absorb the `ninelives` library to enforce I/O determinism.

- **Transformation:** We strip ‘std::time‘ and ‘rand‘ dependencies.
- **Clock:** The ‘Clock‘ trait consumes ‘WAL‘ time samples, not OS time.
- **Jitter:** Uses a deterministic PRNG seeded by the Kernel entropy.
- **Circuit Breakers:** State changes (Open/Closed) are emitted as Graph Rewrites, making system health time-travelable.

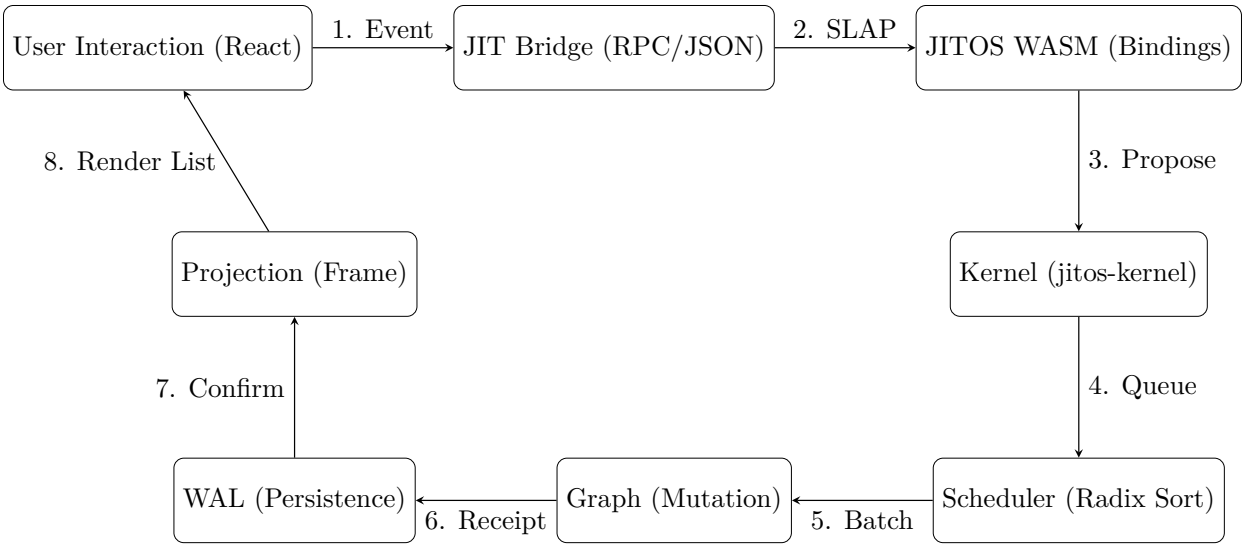
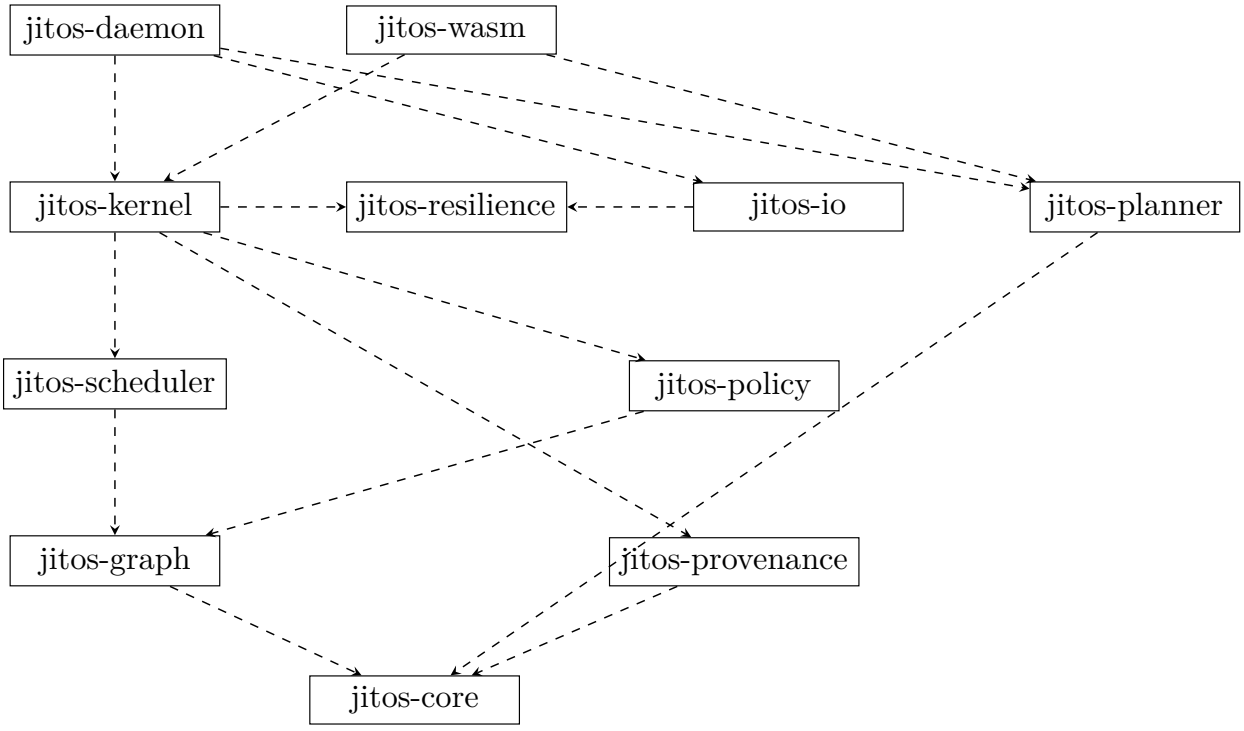


Figure 1: The Deterministic Tick Lifecycle

4 Data Flow: The Tick Lifecycle

4.1 Crate Dependencies



5 Paper Alignment Matrix

This table audits the compliance of the proposed Rust Monorepo against the AION Foundation Papers.

Paper	Concept	Rust Implementation
Paper I	**WARP Structure**	'jitos-graph' implements recursive attachments and atom storage.
Paper II	**Dynamics**	'jitos-scheduler' implements the strict deterministic commit rules (Ticks).
Paper III	**Holography**	'jitos-provenance' ensures the WAL is the only authoritative truth. Boot is replay.
Paper IV	**Geometry**	'jitos-core' defines the Rulial Distance metrics (implemented in 'jitos-policy' for comparison).
Paper V	**Ethics**	'jitos-core/identity' enforces cryptographic signing of every SLAP (Agent Sovereignty).
Paper VI	**Architecture**	The 'jitos-daemon' and 'jitos-wasm' crate structure mirrors the OS/Kernel separation.
ARCH-009	**Time & IO**	'jitos-resilience' implements the Clock View and Event-Driven I/O.

6 Tooling Evolution

6.1 The Time Travel Debugger

Currently, the Time Travel Debugger in the JS shell reads a JS array. In the new architecture:

- It becomes a view into 'jitos-provenance'.
- **Operations:** 'rewind(tick)', 'fork(tick)', 'diff(tickA, tickB)'.
- These are executed by the WASM kernel, returning a lightweight 'Frame' (View) rather than the heavy Graph.

6.2 Echo WARP Graph Viewer

The standalone native GUI viewer in the 'echo' repo is **deprecated**. The Web Shell ('flyingrobots.dev') becomes the universal viewer for both local and remote kernels.

6.3 Network Layer

The custom TCP protocol in 'echo' is replaced by:

1. **Local:** SharedMemory / PostMessage (WASM).
2. **Remote:** WebSockets carrying serialized SLAPS (Binary).

7 The Purity Debate: A Pragmatic Boundary

In the web-based 'WarpKernel', we implemented UI toggles as graph rewrites. However, we delegated Cloth Physics to an imperative runner. Is this cheating?

Argument for Purity: If it's not in the graph, it's not audited. **Argument for Performance:** Graph rewriting is $O(M \cdot N)$ where M is pattern size. Array operations are $O(1)$.

The Verdict: The Port Model We adopt the Hexagonal Architecture defined in Paper VI.

- **Control Plane (Pure):** Decisions, Ownership, Configuration, Top-level State. Implemented as Graph Rewrites.
- **Data Plane (Opaque):** Physics Meshes, Texture Bitmaps, Audio Buffers. Implemented as Imperative logic inside "Wormholes".

The Kernel sees the Physics Engine as an **Oracle**. It sends a ‘Step‘ command and receives a ‘Hash‘ (Integrity) and ‘RenderList‘ (View). It does *not* track every vertex in the DAG.

8 Conclusion

The “State of the Union” is strong but fragmented. The Rust Monorepo Strategy resolves the technical debt, aligns the code with the theory, and provides a clear path to shipping a 60 FPS, cryptographically verifiable, time-traveling Operating System in the browser.