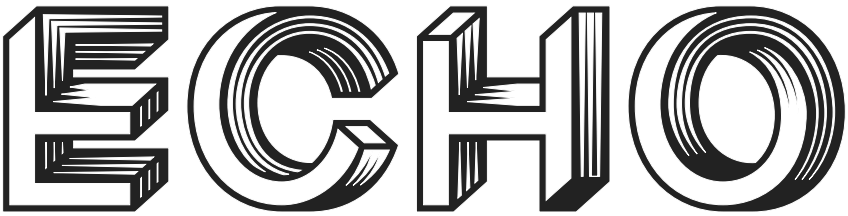

ECHO

Orientation

Version v0.1-draft
Commit: HEAD
December 4, 2025

James Ross & Echo Contributors

Source

This booklet is generated from the Echo repository documentation.

Foreword

Echo is a deterministic, multiverse-aware engine. This booklet walks you in with progressive layers: orient yourself, learn the core building blocks, then dive into math and operations. Each shelf can stand alone; together they form the full Echo field guide.

If you are new, start with the onboarding roadmap and glossary. If you build or extend Echo, keep the determinism contract and scheduler flow in view. Future work will deepen each part and add more diagrams as Echo evolves.

Contents

Part I

Orientation

Chapter 1

Onboarding Roadmap (First 30–60 Minutes)

1.1 Get the Toolchain

- Install Rust per `rust-toolchain.toml` (workspace MSRV).
- Install Node $\geq$ 18 and `pnpm` (see `packageManager` in `package.json`).
- Run `make hooks` once to enable repo checks.

1.2 Clone, Smoke Test, and Guard Rails

- **Clone**: `git clone` and `pnpm install` (docs) / `cargo fetch` (Rust).
- **Docs dev**: `pnpm docs:dev` to confirm VitePress works.
- **Rust sanity**: `cargo test -p rmg-core - -list` to ensure toolchain is healthy.
- **Hooks**: verify pre-commit runs (fmt/clippy/docs guard) by committing a no-op change locally.

1.3 Read This First

1. **Orientation**: `architecture-outline.md` (why Echo exists).
2. **Determinism contract**: skim `determinism-invariants.md` for guarantees.
3. **Branching model**: `branch-merge-playbook.md` (merge-only; no rebases/force pushes).
4. **Work map**: `execution-plan.md` to see current intent and milestones.

1.4 Do One Hands-On Task

- Run a focused test: `cargo test radix_drain -package rmg-core`.
- Explore a doc: open `docs/collision-dpo-tour.html` to see visuals.
- Record what you learned in `decision-log.md` if you touch code/docs.

1.5 Next Hour: Choose Your Track

- **Runtime**: read Scheduler + Storage specs, then run scheduler benchmarks.
- **Docs/Math**: read deterministic math specs; reproduce a figure in TikZ.
- **Ops**: rehearse the merge playbook and run Docs Guard locally.

Chapter 2

Glossary

Echo Deterministic, multiverse-aware ECS runtime.

Chronos / Kairos / Aion Time lenses: real-time execution (Chronos), branching timelines and merges (Kairos), and meta-time/agency (Aion).

Codex's Baby Echo's scheduler core that orders and drains work deterministically.

Branch Tree Structure tracking divergent timelines and merges; governed by the merge playbook.

Determinism Contract Guarantees covering hashing, scheduling order, canonical floats, and replayability.

Dirty Index Tracking of modified components for efficient archetype updates.

Scheduler Kind Configurable scheduler implementation (Radix default; Legacy/BTree fallback).

Temporal Bridge Mechanism for retro/forward delivery without paradox; works with paradox guard.

Snapshot / Commit Canonical state plus provenance (parents, digests) used for replay and verification.

Docs Guard Policy that code changes must update docs (execution plan + decision log).

2.1 What is Echo?

Echo is a **deterministic, multiverse-aware Entity Component System (ECS)** engine. Unlike traditional game engines that prioritize visual fidelity or editor convenience, Echo prioritizes **determinism**, **state preservation**, and **simulation integrity**.

It is built on the concept of a "Rulial Multi-Graph" (RMG), where the entire game state is a graph, and updates are deterministic graph rewrites. This allows for features that are practically impossible in standard engines:

- **Perfect Determinism:** Given the same initial state and inputs, the simulation will evolve identically on any machine (x86, ARM, WASM).

- **Time Travel:** The ability to rollback the simulation to any previous tick, branch time into alternative realities, and merge them back.
- **Renderer Agnosticism:** Echo does not care how it is drawn. It emits a ‘FramePacket‘
(render-agnostic draw list) that adapters feed to WebGL/WebGPU, a terminal UI, or a headless server.

2.1.1 Core Philosophy

Echo was born from the "Caverns" project and operates under a set of strict cultural principles:

1. **Just Ship, But Test:** Rigorous automated testing is not optional.
2. **Hexagonal Domain Boundary:** The core simulation never touches the DOM, WebGL, or system timers directly. Everything is mediated through Ports and Adapters.
3. **Predictable Loop:** A fixed time-step simulation is the default.

2.1.2 Who is Echo For?

Echo is designed for complex simulations where state consistency is paramount:

- **RTS Games:** Where thousands of units must remain in sync across a network.
- **Fighting Games:** Where rollback netcode (GGPO-style) is a requirement, not a feature.
- **Scientific Simulations:** Where reproducibility is critical.
- **Collaborative Tools:** Where multiple users edit a shared state (using the branching/merging capabilities).

2.2 Echo vs. The World

Echo differs significantly from general-purpose engines like Unity, Unreal, or Godot. While those engines optimize for "getting something on screen fast" and providing a rich editor, Echo optimizes for **correctness under complex temporal manipulation**.

2.2.1 Feature Comparison Matrix

2.2.2 Key Differentiators

1. Determinism as a Constraint

In Unity or Unreal, ‘float‘ math depends on the CPU architecture. A simulation running on an Intel chip might diverge from one on an ARM chip after a few thousand frames. Echo uses `F32Scalar` (canonicalized floats) or fixed-point math to ensure bit-perfect identity across all platforms. This enables:

Feature	Echo	Unity/Unreal	Godot
Architecture	Pure ECS (hexagonal ports/adapters)	GameObject / Actor	Node Tree
Determinism	**Core constraint**; radix or legacy scheduler, stable order	Best effort / plug-in	Best effort
State Model	Immutable + COW snapshots	Mutable in-place	Mutable in-place
Netcode	Native rollback/replay (input or state sync)	State sync (mostly)	RPC / state sync
Branching	Native multiverse timelines	Custom / atypical	Not supported
Math	Canonical zero float wrapper; fixed-point planned	Hardware float	Hardware float
Rendering	Decoupled port; adapters pick GPU/OS	Tightly coupled	Tightly coupled

Table 2.1: Comparison of core behaviors (emphasis on determinism and branching).

- **Replay Systems:** Store only inputs, replay the game exactly.
- **Anti-Cheat:** Verify game logic on the server with zero tolerance for deviation.
- **Desync Debugging:** If a bug happens, it happens exactly the same way every time.

2. The Multiverse (Branching)

Traditional engines have one "World". To do a "prediction" (e.g., for netcode), they must serialize the whole world, step forward, and then deserialize it back (a slow process). Echo uses **Copy-on-Write (COW)** archetype storage. It can fork the world instantly.

- **Main Timeline:** The "real" game.
- **Speculative Branch:** Used by AI to "think ahead" or by netcode to predict movement.

These branches share memory for unchanged data, making branching extremely cheap.

3. Rulial Multi-Graph vs. Object Update

In typical engines, objects have an 'Update()' function. Order of execution is often ill-defined or rigid. Echo uses a graph rewriting approach. Systems define rules. The Scheduler identifies independent subgraphs that can be updated in parallel (using MWMR - Multi-Writer Multi-Reader patterns) without race conditions, because the data dependency is explicit.

License and Legal Notice

This project is made available under an open source, dual-licensing model.

Code

All *code* in this repository—including Rust source files, scripts, build tooling, and any compiled binaries—is licensed under the **Apache License, Version 2.0**.

- Canonical text: `LICENSE-APACHE`
- SPDX identifier: `Apache-2.0`

Users may use, modify, and redistribute the code under the terms of the Apache License, Version 2.0.

Theory, Mathematics, and Documentation

The *theory*, *mathematics*, and *documentation* corpus associated with this project—for example LaTeX sources, notes, and expository materials—is dual-licensed under:

1. the Apache License, Version 2.0 (`Apache-2.0`), *or*
2. the MIND-UCAL License, Version 1.0 (`MIND-UCAL-1.0`),

at the user's option.

If you do not wish to use MIND-UCAL, you may freely use all theory, mathematics, and documentation under the Apache License, Version 2.0 alone. No part of this project requires adopting MIND-UCAL in order to be usable.

SPDX Headers

To make licensing machine-readable and unambiguous, the project uses SPDX license identifiers in file headers. Typical examples include:

- Code files (Rust, scripts, etc.):

```
// SPDX-License-Identifier: Apache-2.0
```

- Documentation and theory files (Markdown, LaTeX, etc.):

```
% SPDX-License-Identifier: Apache-2.0 OR MIND-UCAL-1.0
```

These identifiers correspond directly to the licenses described above.

Disclaimer

Unless required by applicable law or agreed to in writing, the material in this project is provided on an "AS IS" basis, without warranties or conditions of any kind, either express or implied. For the full terms, see `LICENSE-APACHE` and `LICENSE-MIND-UCAL`.