# **Doc 3A — Tech Stack & Determinism (targets, offline, numeric, ordering, RNG, parallelism)**

**Scope.** What the engine runs on; how it guarantees **offline**, **deterministic** results that satisfy Doc 6 reproducibility tests.

## **1) Targets (OS / arch)**

* **OS:** Windows 11, macOS 13+, Ubuntu 22.04+ (or equivalent LTS).
* **Arch:** x86-64 and arm64 on all three OSes.
* **UI:** Local desktop app via **Tauri** (Rust core + tiny WebView).
* **Maps:** **MapLibre** for on-device rendering only; tiles/styles are packaged (no network).

## **2) Offline policy**

* **No network access at runtime.** All inputs (DivisionRegistry, BallotTally, ParameterSet, Adjacency, AutonomyPackage) are local files.
* **No telemetry.** No analytics, crash uploaders, or update checks.
* **Fonts/styles/tiles** are bundled in the app; reports are fully self-contained (Doc 7).

## **3) Numeric rules (to avoid float drift)**

* **Counts** (votes, approvals, scores, seats): exact **integers**.
* **Ratios / comparisons** (threshold checks, divisors): compute using **integer arithmetic** where possible (e.g., lhs\*denR >= rhs\*denL), or **rational** (num/den) comparisons—never rely on float equality.
* **When real division is unavoidable** (e.g., for display): use IEEE-754 but **round only at presentation**; internal comparisons use exact integer/rational forms.
* **Rounding rule:** **round half to even** at defined comparison points (Docs 4A/4C).
* **Percent formatting:** Report layer shows **one decimal** (Doc 7).

## **4) Ordering rules (global)**

* **Stable total orders** everywhere:  
  + Units by **Unit ID** (lexicographic).
  + Options by **Option.order\_index**, then by Option ID.
  + Lists in outputs are **sorted** using these orders before hashing/serialization.
* Any parallel work must **reduce** results in this stable order (see §6).

## **5) RNG (for ties only)**

* RNG used **only** when tie\_policy = random.
* Algorithm: **ChaCha20** (stream RNG) with explicit **VM-VAR-052 rng\_seed**; seeding procedure and counter start are fixed and versioned.
* No OS RNG, time, or nondeterministic entropy sources.
* **Seed is recorded** in **RunRecord** and each TieLog entry (Docs 5C/7B).
* With the same seed and inputs, winners and TieLogs are **byte-identical** across OS/arch (Doc 6C-020).

## **6) Allowed parallelism**

* **Safe parallel stages:** per-Unit **Tabulate** and **Allocate** may run in parallel.
* **Deterministic reduction:** all merges/aggregations happen by the stable orders in §4.
* **No parallel RNG use.** Tie resolution is serialized in the order the ties appear by stable ordering of contexts.
* **I/O** (reads) may be parallel; **writes** (Result/RunRecord/FrontierMap) are single-writer, ordered.

## **7) File formats & normalization**

* **Serialization:** UTF-8, JSON with **sorted keys**; line endings **LF** on disk artifacts; canonical timestamp **UTC** ISO-8601.
* **Hashes:** Results/RunRecords’ IDs are derived from canonicalized bytes (inputs + engine + Formula ID), not from platform paths.

## **8) Third-party stack (pinned in 3B)**

* **Rust** (stable, pinned via rust-toolchain.toml).
* **Tauri** for packaging; **MapLibre** for local map rendering.
* No dynamic plugins; no runtime code download.

# **Doc 3B — Build & Release (repro builds, CI, perf/memory, deps, security, artifacts)**

**Scope.** How we build the same bits everywhere, keep them fast/safe, and ship verifiable artifacts.

## **1) Reproducible builds**

* **Pin toolchains:** rust-toolchain.toml (exact stable version); Cargo.lock committed.
* **Deterministic flags:** disable incremental, set a fixed codegen unit count; embed SOURCE\_DATE\_EPOCH in CI.
* **Assets lock:** versions/hashes of styles, fonts, tiles are recorded; embedded at build.
* **No build-time network for code.** Vendored crates via lock; if mirrors are used in CI, hashes must match Cargo.lock.

## **2) CI matrix (must pass on all)**

* **OS:** Windows, macOS, Ubuntu.
* **Arch:** x86-64 and arm64 (native or cross).
* **Jobs:**
  1. **Lint & unit tests.**
  2. **Determinism checks:** build twice; compare binary and artifact hashes; run VM-TST-001 end-to-end twice → identical Result/RunRecord.
  3. **Cross-OS determinism:** run VM-TST-001 on all OS; compare artifacts (Doc 6C-020).
  4. **Performance profile:** run the large synthetic (Doc 6C-019) and record time/memory to perf\_profile.json.
  5. **Security:** SBOM generation; license scan.

## **3) Performance & memory gates**

* **Reference profile** is stored as versioned perf\_profile.json (per OS/arch).
* A PR **fails** if runtime or memory **regresses beyond the configured tolerance** versus the last released profile for the same OS/arch.
* The **large deterministic pass** in Doc 6C-019 uses this profile to assert “within ceiling” (no hardcoded numbers here; the ceiling is the published profile).

## **4) Dependency policy**

* Only crates with **explicit versions** and compatible licenses.
* Any crate affecting math/serialization (e.g., RNG, JSON serializer) is **pinned** and listed in a **critical-deps** section; upgrades require a determinism re-cert run (6C-020).
* No optional features that alter output format unless guarded by a **feature gate** that is off for releases.

## **5) Security posture**

* **No telemetry** or analytics.
* **Code signing** on release binaries for each OS.
* **Sandboxing:** Tauri’s filesystem scope restricted to user-chosen folders; no shell command execution.
* **SBOM** (SPDX or CycloneDX) is built and shipped with each release.
* **No dynamic code loading**; plugins/themes are data-only.

## **6) Release artifacts (what we ship)**

* **Binaries:** signed installers/archives per OS/arch.
* **Checksums:** SHA-256 for every artifact (\*.sha256).
* **SBOM:** sbom.json.
* **Docs bundle:** the seven normative docs (1–7) that define the formula/rules used.
* **Formula ID:** a cryptographic **hash of the normative rule set** (Docs 4A/4B/4C with version markers). Printed in the app, **RunRecord**, and Report footer.
* **Engine Version:** semantic version of the implementation; printed with Formula ID.

## **7) Release process**

1. Tag repository with engine-vX.Y.Z and formula-vA.B.C.
2. CI builds all matrices, runs determinism/perf/security jobs.
3. On success, CI publishes artifacts + checksums + SBOM to the release page.
4. A **Repro Manifest** is published: toolchain hash, Cargo.lock, asset hashes, determinism proof (hashes of canonical test outputs).
5. A **ChangeLog** distinguishes **MAJOR/MINOR/PATCH** (Docs 7/5 conventions).

## **8) How this supports Doc 6 tests**

* **Doc 6C-019/020** reproducibility: pinned toolchains, canonical serialization, stable RNG, sorted keys, stable ordering rules.
* **Doc 6A/6B** seat math & gates: integer/rational comparisons and round-half-to-even ensure cross-OS equality.
* **Doc 7** report footer: Formula ID, Engine Version, Division Registry, Parameter Set, BallotTally label, Run timestamp, Results ID—**all pulled from RunRecord**.

## **9) Developer checklist (per PR)**

* No new network calls; no time-dependent logic.
* Keep Option/Unit ordering stable.
* If changing RNG/serializer/math crates or rules, bump **Formula ID** and re-run cross-OS determinism checks.
* Update perf\_profile.json only after investigating regressions.

**Status:** Tooling and release steps are unambiguous; determinism and offline guarantees satisfy the requirements referenced by Doc 6.