Pre-Coding Essentials (Component: crates/vm\_core/src/rng.rs, Version/FormulaID: VM-ENGINE v0) — 28/89

1. Goal & Success

Goal: Deterministic RNG utilities for tie resolution only, using a fixed, seeded stream cipher (ChaCha20).

Success: With the same **integer tie\_seed** and the same inputs, choices/shuffles are byte-identical across OS/arch; no reliance on OS entropy or time; API is minimal and safe.

1. Scope

In scope: Seed handling from **VM-VAR-033 tie\_seed** (integer), ChaCha20 wrapper, uniform choice without modulo bias, deterministic shuffle (Fisher–Yates), reproducible u64/u128 streams, small log hook.

Out of scope: Any non-tie randomness, parallel RNG (not permitted), OS RNG, time-based seeding.

1. Inputs → Outputs

Inputs: **tie\_seed (VM-VAR-033, integer ≥ 0)** when **tie\_policy (VM-VAR-032) = random**; candidate sets; optional domain bounds.

Outputs: Indices/permutes/integers; optional compact trace (context label + picks) to be forwarded to TieLog (owned by pipeline).

1. Entities/Tables (minimal)

None.

1. Variables

* **VM-VAR-032 tie\_policy** ∈ {status\_quo, deterministic, random} (default: status\_quo) — RNG used only if = random.
* **VM-VAR-033 tie\_seed** ∈ integers (≥ 0) (default: 0) — recorded in RunRecord/TieLog when used.

1. Functions (signatures only)

/// Opaque deterministic RNG for ties.

pub struct TieRng(ChaCha20Rng);

/// Build from integer tie\_seed; stable across platforms.

pub fn tie\_rng\_from\_seed(seed: u64) -> TieRng;

impl TieRng {

/// Next unbiased integer in [0, n) using rejection sampling.

pub fn gen\_range(&mut self, n: u64) -> u64;

/// Choose index of winner from non-empty slice; error on empty.

pub fn choose\_index<T>(&mut self, slice: &[T]) -> Result<usize, RngError>;

/// Deterministic in-place Fisher–Yates shuffle (stable given same seed).

pub fn shuffle<T>(&mut self, xs: &mut [T]);

/// Emit next u64 / u128 for audit or higher-level use.

pub fn next\_u64(&mut self) -> u64;

pub fn next\_u128(&mut self) -> u128;

/// Return how many 64-bit words have been consumed.

pub fn words\_consumed(&self) -> u128;

/// Optional: record a tiny crumb for TieLog (context/candidates/pick).

pub fn log\_pick(&self, ctx: &str, pick: usize) -> TieCrumb;

}

/// Small, serializable crumb (pipeline aggregates into TieLog).

pub struct TieCrumb { pub ctx: SmolStr, pub pick: u32, pub word\_index: u128 }

1. Algorithm Outline (implementation plan)

Seed handling

* Initialize RNG with **ChaCha20Rng::seed\_from\_u64(tie\_seed)**.
* Start counter at 0; bump by 1 per next\_u64 (two bumps for next\_u128).

Unbiased range generation

* Rejection sampling: draw 64-bit x; compute zone = u64::MAX - (u64::MAX % n); if x < zone, return x % n; else redraw.
* Handles any n ∈ [1, 2^63]; reject n = 0.

Choice & shuffle

* choose\_index: error on empty slice; otherwise gen\_range(len).
* shuffle: Fisher–Yates descending for i in (1..len).rev() with j = gen\_range(i as u64 + 1).

Audit/trace

* words\_consumed returns monotonic count; TieCrumb stores context string, chosen index (u32 ok for list sizes), and the word index when decision was made.

No parallelism

* Callers must serialize all tie resolutions in the deterministic order defined by the pipeline.

1. State Flow

Pipeline enters **RESOLVE\_TIES** only when needed; it constructs TieRng from **Params.tie\_seed** when **Params.tie\_policy = random**; each tie resolution calls choose\_index/gen\_range in stable context order; crumbs (optional) are collected and written into the final TieLog in **RunRecord/Result**.

1. Determinism & Numeric Rules

Identical **tie\_seed** ⇒ identical output sequence and crumbs.

No floats; no OS RNG/time; no global mutable state.

All consumers must keep a fixed call sequence (stable ordering from determinism module).

1. Edge Cases & Failure Policy

* gen\_range(0) or choose\_index([]) ⇒ RngError::EmptyDomain.
* Extremely skewed n values are fine (rejection loops terminate quickly on average).
* Do not expose internal state beyond words\_consumed (audit only).

1. Test Checklist (must pass)

* **Seed determinism:** same u64 seed → identical sequences for next\_u64, gen\_range, shuffle; different seeds differ.
* **Unbiasedness (sanity):** histogram for gen\_range(10) over large N is ~uniform (statistical smoke).
* **Choice:** empty slice errors; non-empty returns valid index.
* **Shuffle:** two runs with same seed produce identical permutation; changing seed changes permutation.
* **Crumbs:** log\_pick reports correct word index; sequence of crumbs matches call order.