# **Pre-Coding Essentials (Component: crates/vm\_core/src/determinism.rs, Version/FormulaID: VM-ENGINE v0) — 26/89**

## **1) Goal & Success**

Goal: Core utilities that enforce **stable total ordering** and **deterministic reduction** across the engine.

Success: All merges/sorts use canonical orders (Units by UnitId; Options by order\_index then OptionId); reductions are order-independent in parallel execution; byte output is unchanged across OS/arch.

## **2) Scope**

In scope: ordering traits/helpers, canonical sorting, deterministic reducers, hash-canonicalization glue traits.

Out of scope: RNG (rng.rs), numeric comparisons/rounding (rounding.rs), I/O or hashing implementations.

## **3) Inputs → Outputs**

Inputs: Collections of IDs/entities, partial results from parallel stages.

Outputs: Stably ordered slices/maps and order-independent reduction results suitable for canonical serialization and hashing.

## **4) Entities/Tables (minimal)**

## **5) Variables (only ones used here)**

## **6) Functions (signatures only)**

rust

CopyEdit

// 1) Ordering primitives

pub trait StableOrd { fn stable\_cmp(&self, other:&Self) -> core::cmp::Ordering; }

impl StableOrd for UnitId { /\* lexicographic \*/ }

impl StableOrd for OptionItem { /\* by order\_index, then id \*/ }

impl StableOrd for OptionId { /\* lexicographic \*/ }

// 2) Canonical sort helpers

pub fn sort\_units\_canonical<T: AsRef<UnitId>>(xs: &mut [T]);

pub fn sort\_options\_canonical(xs: &mut [OptionItem]); // (order\_index, id)

// 3) Deterministic reduction

pub fn reduce\_deterministic<T, F>(mut items: Vec<T>, mut combine: F) -> Option<T>

where

T: StableReduce + StableOrd, // StableReduce: identity() + combine()

F: Fn(T, T) -> T;

// Trait for values that can be reduced deterministically

pub trait StableReduce: Sized {

fn identity() -> Self;

}

// 4) Map canonicalization

pub fn btreemap\_from\_iter\_kv<K: Ord, V, I: IntoIterator<Item=(K,V)>>(it: I) -> alloc::collections::BTreeMap<K,V>;

// 5) Hash-canon glue (interface only; no I/O)

pub trait HashCanon { fn canonical\_bytes(&self) -> Vec<u8>; } // re-exported from lib

## **7) Algorithm Outline (implementation plan)**

**Stable orders**

UnitId: Ord on its canonical string is already total; StableOrd delegates to it.

OptionItem: compare order\_index first; on equality, compare OptionId.

**Canonical sort**

Provide thin wrappers that sort in-place using the above rules and are used by pipeline/report code before any hashing/serialization.

**Deterministic reduction**

Strategy: sort inputs using StableOrd, then fold with StableReduce::combine (provided by caller via closure or trait).

For parallel callers: reduce chunks locally, then call reduce\_deterministic on the chunk results to ensure final result does not depend on chunking order.

**Map canonicalization**

Always materialize key-sorted maps as BTreeMap (never HashMap) when order affects downstream bytes.

**No globals**

No thread-local state or OS calls; pure functions only.

## **8) State Flow**

Upstream stages generate partial results → call sort\_\*\_canonical and/or reduce\_deterministic → downstream serialization/hashing consumes already-canonical structures.

## **9) Determinism & Numeric Rules**

Determinism: stable total orders for Units/Options; reductions proceed in **sorted** order.

No numeric rounding here; numeric comparisons live in rounding.rs.

## **10) Edge Cases & Failure Policy**

reduce\_deterministic on an empty vector → None.

If caller’s combine is **not associative**, results may differ across chunkings; document this and keep associative in engine code paths.

Sorting helpers must not allocate unnecessarily for large slices; prefer in-place sort.

## **11) Test Checklist (must pass)**

Sorting:

Units sort lexicographically by UnitId and are stable across OS/arch.

Options sort by (order\_index, id); equal order\_index breaks ties by OptionId.

Reduction:

Partition input into random chunks, reduce in parallel (simulate), then merge with reduce\_deterministic ⇒ same result as single-thread fold.

Identity element is neutral: combine(x, identity()) == x.

Map canonicalization:

btreemap\_from\_iter\_kv iteration order is sorted by key; serializing keys to bytes yields identical order across runs.