# **Pre-Coding Essentials (Component: crates/vm\_algo/src/lib.rs, Version/FormulaID: VM-ENGINE v0) — 36/89**

## **1) Goal & Success**

Goal: Public surface for **algorithm primitives**: ballot tabulation, unit-level allocation, pairwise/ranked helpers, gates checks, frontier helpers, and small MMP utilities. Pure compute; no I/O.

Success: Deterministic, integer/rational math only; stable ordering; RNG only when injected (TieRng). API is minimal and maps 1:1 to pipeline steps.

## **2) Scope**

In scope: pub mod declarations and re-exports; function signatures for:

**Tabulation** (plurality, approval, score, IRV, Condorcet)

**Allocation** (WTA, D’Hondt, Sainte-Laguë, Largest Remainder)

**MMP** helpers

**Gates** (quorum/majorities, double-majority synthesis)

**Frontier** helpers (support computation only; not contiguity)

Out of scope: schema/JSON, path I/O, report formatting, pipeline orchestration.

## **3) Inputs → Outputs (with IDs/types)**

Inputs: vm\_core entities (UnitId, OptionId, OptionItem), counts from IO/loader, and Params for behavior switches.

Outputs: plain structs/maps (integers/ratios), plus audit logs for ranked methods; these are consumed by vm\_pipeline.

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

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

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

rust

CopyEdit

use vm\_core::{

ids::{OptionId, UnitId},

entities::OptionItem,

variables::Params,

rounding::{Ratio, cmp\_ratio\_half\_even as cmp\_ratio}, // or compare API

rng::TieRng,

};

// ---- Common structs returned by algorithms ----

pub struct UnitScores {

pub unit\_id: UnitId,

pub turnout: Turnout, // from vm\_core::entities

pub scores: BTreeMap<OptionId, u64>, // plural/approval/score sum; ranked fills winner-only or per-round tallies via logs

}

pub struct Allocation {

pub unit\_id: UnitId,

pub seats\_or\_power: BTreeMap<OptionId, u32>, // WTA => single 100% special handled by pipeline/report

pub last\_seat\_tie: bool, // true if tie policy had to be applied

}

// ---- Tabulation (deterministic) ----

pub fn tabulate\_plurality(unit\_id: UnitId,

votes: &BTreeMap<OptionId, u64>,

turnout: Turnout) -> UnitScores;

pub fn tabulate\_approval(unit\_id: UnitId,

approvals: &BTreeMap<OptionId, u64>,

turnout: Turnout) -> UnitScores;

pub fn tabulate\_score(unit\_id: UnitId,

score\_sums: &BTreeMap<OptionId, u64>,

turnout: Turnout,

params: &Params) -> UnitScores;

// Ranked IRV (audit log with eliminations/transfers; exhaustion fixed policy)

pub struct IrvRound { pub eliminated: OptionId, pub transfers: BTreeMap<OptionId, u64>, pub exhausted: u64 }

pub struct IrvLog { pub rounds: Vec<IrvRound>, pub winner: OptionId }

pub fn tabulate\_ranked\_irv(ballots: &[(Vec<OptionId>, u64)],

options: &[OptionItem],

params: &Params) -> (UnitScores, IrvLog);

// Condorcet (pairwise matrix + completion)

pub struct Pairwise { pub wins: BTreeMap<(OptionId, OptionId), u64> }

pub fn tabulate\_ranked\_condorcet(ballots: &[(Vec<OptionId>, u64)],

options: &[OptionItem],

params: &Params) -> (UnitScores, Pairwise);

// ---- Allocation inside a Unit ----

pub fn allocate\_wta(scores: &UnitScores, magnitude: u32,

options: &[OptionItem],

tie\_policy: TiePolicy, rng: Option<&mut TieRng>) -> Allocation;

pub fn allocate\_dhondt(seats: u32,

scores: &BTreeMap<OptionId, u64>,

options: &[OptionItem]) -> BTreeMap<OptionId, u32>;

pub fn allocate\_sainte\_lague(seats: u32,

scores: &BTreeMap<OptionId, u64>,

options: &[OptionItem]) -> BTreeMap<OptionId, u32>;

pub fn allocate\_largest\_remainder(seats: u32,

scores: &BTreeMap<OptionId, u64>,

threshold\_pct: u8,

options: &[OptionItem]) -> BTreeMap<OptionId, u32>;

// ---- MMP helpers (top-ups after local seats) ----

pub fn mmp\_target\_shares(total\_seats: u32,

vote\_totals: &BTreeMap<OptionId, u64>,

method: AllocationMethod) -> BTreeMap<OptionId, u32>;

pub fn mmp\_topups(local\_seats: &BTreeMap<OptionId, u32>,

targets: &BTreeMap<OptionId, u32>,

overhang\_policy: OverhangPolicy,

total\_seats\_model: TotalSeatsModel) -> BTreeMap<OptionId, u32>;

// ---- Gates (ratios; integers only) ----

pub struct GateInputs { pub valid\_ballots: u64, pub approvals\_for\_change: u64, pub eligible\_roll\_sum: u64 }

pub struct GateOutcome { pub pass: bool, pub observed: Ratio, pub threshold\_pct: u8 }

pub fn gate\_quorum(valid\_ballots: u64, eligible\_roll: u64, threshold\_pct: u8) -> GateOutcome;

pub fn gate\_majority(valid\_ballots: u64, approvals\_for\_change: u64, threshold\_pct: u8) -> GateOutcome;

// Double-majority (compose national & regional outcomes)

pub struct DoubleMajority { pub national: GateOutcome, pub regional: GateOutcome, pub pass: bool }

pub fn gate\_double\_majority(national: GateOutcome, regional: GateOutcome) -> DoubleMajority;

// ---- Frontier support helper (no topology) ----

pub fn frontier\_support\_ratio(approvals\_for\_change: u64, valid\_ballots: u64) -> Ratio;

// ---- Tie policy enum (re-export or local alias from vm\_core) ----

pub use vm\_core::variables::TiePolicy;

pub use vm\_core::variables::{AllocationMethod, OverhangPolicy, TotalSeatsModel};

## **7) Algorithm Outline (module layout)**

pub mod tabulation; (plurality/approval/score/IRV/Condorcet) → re-export main functions and logs.

pub mod allocation; (WTA/D’Hondt/Sainte-Laguë/LR) → pure integer math; stable option order.

pub mod mmp; (targets & top-ups) → apply policies; no floats.

pub mod gates\_frontier; → quorum/majorities/double-majority; frontier support ratio helper.

pub use selected structs (UnitScores, Allocation, IrvLog, Pairwise, GateOutcome, DoubleMajority).

## **8) State Flow**

vm\_pipeline calls tabulate\_\* → gets UnitScores; then allocate\_\* per unit; then aggregates and calls gate\_\* and frontier helpers; MMP functions only when selected by params. Tie resolution in WTA/last-seat handled via TiePolicy/TieRng.

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

Use vm\_core::determinism ordering: options sorted by (order\_index, OptionId); never depend on map iteration order.

Integer/rational math only; **round half to even** only at allowed decision points (via rounding helpers); no floats.

RNG only when TiePolicy::Random and TieRng is provided; otherwise deterministic order or status-quo policy.

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

Zero valid\_ballots ⇒ gates return pass=false, observed=0/1.

WTA with magnitude != 1 is **caller error**; allocation function asserts or returns error variant (pick one and keep consistent).

PR threshold excludes options strictly below threshold before seat calculation.

Ranked: IRV exhaustion uses fixed policy; Condorcet completion per Params.

Last-seat ties: if TiePolicy::DeterministicOrder, break by (order\_index, OptionId); if StatusQuo, prefer status-quo option; if Random, use TieRng.

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

**VM-TST-001** (Sainte-Laguë m=10, A/B/C/D=10/20/30/40) → seats 1/2/3/4.

**VM-TST-002** (WTA m=1 plurality A/B/C/D=10/20/30/40) → D wins 100%.

**VM-TST-003** (LR vs D’Hondt vs Sainte-Laguë convergence case) → same allocation vector.

IRV: known toy example with fixed exhaustion policy → winner & round log match expectation.

Condorcet: simple cycle resolved per selected completion rule (Schulze/Minimax).

Gates: quorum/majority computations match integer/rational comparisons; approval gate uses **approval rate** denominator (valid ballots).