# **Pre-Coding Essentials (Component: crates/vm\_algo/src/allocation/dhondt.rs, Version/FormulaID: VM-ENGINE v0) — 43/89**

## **1) Goal & Success**

Goal: Implement **D’Hondt (highest averages, favor big)**: sequentially award seats using divisors **1,2,3,…**, after applying the **PR entry threshold**. Deterministic ties per spec; integer math only.

Success: For any Unit with magnitude m, output seat vector summing to m; below-threshold options excluded; last-seat ties resolved per policy. Convergence test (A/B/C=34/33/33, m=7) returns **3/2/2**.

## **2) Scope**

In scope: Per-Unit D’Hondt allocation, threshold filter, quotient selection loop, deterministic/reproducible tie handling, stable ordering.

Out of scope: Tabulation, aggregation, gates/frontier, I/O/schema.

## **3) Inputs → Outputs**

Inputs:

seats: u32 (Unit.magnitude; validation ensures ≥1).

scores: &BTreeMap<OptionId,u64> (natural tallies from tabulation).

options: &[OptionItem] (provides (order\_index, id) and status-quo flag).

threshold\_pct: u8 (VM-VAR-012).

tie\_policy: TiePolicy, optional rng: &mut TieRng (if random).

Output: BTreeMap<OptionId, u32> seats per option, sum=seats.

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

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

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

rust

CopyEdit

use std::collections::BTreeMap;

use vm\_core::{

ids::OptionId,

entities::OptionItem,

rng::TieRng,

variables::TiePolicy,

};

/// D’Hondt allocation (highest averages with divisors 1,2,3,...).

pub fn allocate\_dhondt(

seats: u32,

scores: &BTreeMap<OptionId, u64>,

options: &[OptionItem],

threshold\_pct: u8,

tie\_policy: TiePolicy,

rng: Option<&mut TieRng>,

) -> Result<BTreeMap<OptionId, u32>, AllocError>;

// Helpers

fn filter\_by\_threshold(

scores: &BTreeMap<OptionId, u64>,

threshold\_pct: u8,

) -> BTreeMap<OptionId, u64>; // share uses ballot’s natural totals for allocation

fn next\_award(

seats\_so\_far: &BTreeMap<OptionId, u32>,

eligible\_scores: &BTreeMap<OptionId, u64>,

options: &[OptionItem],

tie\_policy: TiePolicy,

rng: Option<&mut TieRng>,

) -> OptionId; // chooses argmax of v/(s+1), ties per policy

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

**Threshold filter**

Compute each option’s **share** using the ballot’s **natural totals** (approval: approvals share; plurality: vote share; score: score-sum share). Drop options whose share is **strictly below** threshold\_pct.

**Initialize**

alloc[opt]=0 for all **eligible** options; pre-build an ordered vector of options by (order\_index, id) for stable scans.

**Seat loop** (repeat seats times)

For each eligible opt, compute the next quotient q = scores[opt] / (alloc[opt] + 1) **without floats** (compare via cross-multiplication).

Pick the **max** q; if multiple maxima:

Compare raw **scores** first if the tie is due to identical quotients at different alloc (spec’s “general tie” guidance). If still tied, apply **deterministic order**; if tie\_policy=random, draw with seeded RNG.

Increment that option’s seat.

**Finish**

Return alloc (sum must equal seats).

**Audit hooks** (optional struct for tests): emit the award trail (opt, divisor index) to reproduce steps in fixtures.

Note: The divisor sequence is **1,2,3,…** (classic D’Hondt) and must be applied exactly.

## **8) State Flow**

Called from **AllocateUnit** after Tabulate; applies before aggregation; respects PR threshold and tie rules from Doc 4B/4C.

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

Integer-only comparisons; implement quotient comparisons via (v1\*(s2+1)) vs (v2\*(s1+1)).

Stable ordering by (order\_index, OptionId) whenever a deterministic choice is needed.

If tie\_policy=random, use only the provided **seeded** RNG for reproducibility (no OS entropy). (General tie handling reference.)

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

seats == 0 ⇒ return empty alloc.

After threshold, **no eligible options** ⇒ AllocError::NoEligibleOptions (pipeline may label run accordingly).

All scores **0** with ≥1 seat ⇒ allocate seats entirely by tie policy (deterministic order unless random requested).

Overflow guards: use u128 for cross-multiplications.

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

**Convergence case**: A/B/C = 34/33/33, m=7 ⇒ **3/2/2**.

**Baseline sanity**: with A/B/C/D = 10/20/30/40, m=10, compare with Sainte-Laguë fixture (different allocation; here verify D’Hondt’s specific split—method difference is expected).

**Threshold filter**: set threshold\_pct>0 and ensure below-threshold options get **0** seats and never considered.

**Determinism**: permute input map insertion order; outcomes identical due to BTreeMap + canonical option order.

**Tie behavior**: craft equal quotient round; verify deterministic-order selection; with random + fixed seed, winner is reproducible.