# **Pre-Coding Essentials (Component: crates/vm\_algo/src/tabulation/approval.rs, Version/FormulaID: VM-ENGINE v0) — 38/89**

## **1) Goal & Success**

Goal: Deterministically compute **UnitScores** for **approval** ballots from per-option approval counts and turnout.

Success: Returns exact integer scores per OptionId, preserves canonical option order, and carries turnout (ballots\_cast, invalid\_or\_blank, valid\_ballots). No floats, no RNG. Support % for legitimacy gates is handled elsewhere (approval **rate** over **valid ballots**), not in this function.

## **2) Scope**

In scope: Per-unit approval tabulation; non-negative count checks; **per-option** cap ≤ valid\_ballots.

Out of scope: Allocation, gates/threshold math, aggregation, tie resolution, I/O/schema.

## **3) Inputs → Outputs**

Inputs:

unit\_id: UnitId

approvals: &BTreeMap<OptionId, u64> (per-option approval counts)

turnout: Turnout (ballots\_cast, invalid\_or\_blank, valid\_ballots)

options: &[OptionItem] (enforce canonical (order\_index, id) ordering)

Output:

UnitScores { unit\_id, turnout, scores: BTreeMap<OptionId, u64> } (scores keyed and iterated in canonical option order)

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

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

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

rust

CopyEdit

use std::collections::BTreeMap;

use vm\_core::{

ids::{UnitId, OptionId},

entities::{Turnout, OptionItem},

};

pub fn tabulate\_approval(

unit\_id: UnitId,

approvals: &BTreeMap<OptionId, u64>,

turnout: Turnout,

options: &[OptionItem],

) -> UnitScores;

fn canonicalize\_scores(

approvals: &BTreeMap<OptionId, u64>,

options: &[OptionItem],

) -> BTreeMap<OptionId, u64>;

fn check\_tally\_sanity(

scores: &BTreeMap<OptionId, u64>,

turnout: &Turnout,

) -> Result<(), TabError>;

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

**Canonical order**

Iterate options in (order\_index, OptionId) order; for each, read approvals.get(&opt.id).copied().unwrap\_or(0) and insert into a fresh BTreeMap<OptionId,u64> to ensure stable iteration downstream.

**Sanity checks**

All counts are non-negative (u64).

**Per-option cap:** for every option, approvals\_for\_option ≤ turnout.valid\_ballots (each ballot can approve an option at most once).

**Do not** enforce Σ approvals ≤ valid\_ballots (multiple approvals per ballot are legal).

Unknown option IDs in approvals ⇒ **error** (TabError::UnknownOption)—loader should prevent this, but defend here.

**Assemble result**

Return UnitScores{ unit\_id, turnout, scores }. No normalization or percentages here.

## **8) State Flow**

Pipeline: **TABULATE (approval)** → produce UnitScores → **ALLOCATE** (PR/WTA) → **AGGREGATE** → **GATES** (where approval **rate** is used for support %).

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

Determinism via canonical option iteration and BTreeMap storage.

Integer math only; no rounding; no RNG.

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

Missing option in input map ⇒ treated as 0.

Extra/unknown option in input map ⇒ TabError::UnknownOption.

valid\_ballots == 0 ⇒ all scores must be 0 (per-option cap enforces this).

Any per-option approvals exceeding valid\_ballots ⇒ TabError::OptionExceedsValid.

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

Happy path: A/B/C/D approvals (e.g., 10/20/30/40), turnout 100/0/100 → scores equal input; canonical order matches options.

Per-option cap: with valid\_ballots=50, any option >50 ⇒ **fail**.

Unknown option key in approvals ⇒ **fail**.

valid\_ballots=0 with non-zero approvals ⇒ **fail**.

Determinism: shuffle insertion order of approvals → identical UnitScores.scores order and canonical bytes.