pot@1.0: Exact Constant-Time Minting over Attested, Delegable External Deposits

Sam Williams
sam@arweave.org

Rani El Husseini
rani.elhusseini@arweave.org

Ayush Agrawal
ayush.agrawal@arweave.org

Markus Ihringer
markus@arweave.org

Noah Levenson
noah.levenson@arweave.org

July 2026

Abstract

We describe the mechanism implemented by the pot@1.0 AO-Core device: a token mint that distributes a time-scheduled emission across users in proportion to deposits of *other* assets, attested by trusted oracles. Instead of crediting every account on every tick—Θ(users) work per unit of time—the mechanism maintains a three-level tower of rolling accumulators (global, per-resource, per-user) so that any user's live balance is computable on demand in constant time per resource. All arithmetic is over the integers: the single lossy division point in the tower carries its remainder forward exactly, and the sole approximation—a floor at user settlement—is bounded by one indivisible token unit per settlement event. The mechanism further supports weighted multi-asset deposits, re-weighting in $O(1)$, in-ledger delegation of deposits with recursive unwinding, and composition into child mints that pay yield in a different token than the parent. Each deposit carries its oracle-attested (*originated*) stake, giving the ledger double-entry structure: exits are bounded by attested stake, recall the exiting party's delegations first, and cancel (net) delegation cycles—which provably carry no economic content—outright. We give the exact state-transition semantics of every operation and prove a conservation identity that permits line-item audit of every scaled emission unit; boundary behaviours are stated alongside the definitions that produce them. The core theorems are mechanized in Lean 4 and enforced as runtime properties against the production device.

1 Overview

A common requirement in token networks is to mint a new token to participants in proportion to some externally attested contribution—e.g. assets staked on other networks—accruing continuously over time. The naive construction runs a clock: at every tick, compute each participant's share and credit it. Its cost is $\Theta(|\mathcal{U}|)$ ledger writes per tick for $|\mathcal{U}|$ participants, and balances are stale between ticks.

pot@1.0 instead *rearranges the balance equation* so that the time-dependent part of every user's balance is a difference of two readings of a shared monotone accumulator, multiplied by the user's (piecewise-constant) stake. Global state advances in $O(1)$ per event, independent of the number of users; a user's balance is realized ("settled") only when it is observed or when their stake changes. The construction descends from the per-share accumulator family used by MakerDAO's DSR `pot` (the *chi* rate accumulator) [1], Synthetix's `rewardPerTokenStored` [2], and MasterChef's `accRewardPerShare` [3], but differs in several ways:

1. **Separate yield asset.** Deposits are units of foreign, oracle-attested assets ("resources"); yield is paid in the pot's own token. Deposits never enter the pot's ledger—only attestations do.
2. **Multiple weighted resources.** Emission is split across resources by governable weights. A two-level factorisation of the accumulator makes a weight change $O(1)$: it touches one resource node, never the users below it.
3. **Integer exactness.** There is exactly one division by a state-dependent quantity in the tower; its remainder is carried forward losslessly. Everything else is integer multiplication and one final floor per user settlement.

4. **Scheduled emission.** The mint follows a geometric approach-the-cap schedule evaluated in closed form over arbitrary elapsed time, with a certified interval-arithmetic fallback for astronomically large exponents: the evaluator either returns the exact value or aborts; it never approximates.
5. **Delegation.** Deposit units may be delegated between addresses. Yield follows possession; the delegation edge records the right to reclaim. The ledger keeps double-entry books: each deposit's possession always equals its oracle-attested stake plus delegations received minus delegations made. Reclaiming recursively unwinds the recipient's own outgoing delegations; delegation *cycles*, which carry no net content, are cancelled outright (netted), the way a clearing house cancels offsetting debts. Exits are bounded by attested stake and recall the exiting party's delegations first, so no unbacked reclaim rights survive an exit. Message-atomicity makes any failed unwind roll back completely.
6. **Composition.** Delegating to another pot process mirrors the stake into that "child" mint, yielding the child's token to the delegator—a yield swap—while the child process itself accrues the parent-token yield on the delegated stake.

Trust model: for the purposes of this paper the per-resource *authorities* (oracles attesting deposit and unstake events on the source networks) are assumed honest, as are *weight authorities* and, for child mints, the configured parent process. The mechanism's own arithmetic assumes nothing further.

Validation. Every theorem and invariant in this paper is either machine-checked or continuously tested, in three tiers. (i) *Proofs*: the Lean 4 development in `lean/` mechanizes the emission evaluator's equality with the closed form of eq. (2) (both branches, including soundness of the interval fallback), preservation of the conservation ledger (Theorem 1), the double-entry identity (Lemma 3) and stake conservation by every state-mutating primitive, the delegation-unwinding results of Section 6, and the graph-level recall liveness lemma for iterated recall of the exiting party's own outbound edges (`recall_liveness`, depending on the standard axioms alone). (ii) *Model*: a pure Erlang reference machine (`dev_pot_model`) mirrors the mechanized definitions and replays Lean-exported traces to field-exact equality. (iii) *Device*: the `hb_invariant` property suite (`rebar3 pot-props`) runs the production device in lockstep against the reference machine over randomized operation streams and adversarial scripted scenarios, and asserts every invariant of this paper—including the tested liveness property that attested exits never fail—after every step. A violation of any proven invariant at runtime is therefore conclusive evidence of an implementation or substrate fault, never a specification gap.

2 Notation and state

Constants. All quantities are non-negative integers (the host virtual machine has arbitrary-precision integers; no overflow analysis is required). Two fixed-point scales are used:

$$\Lambda = 10^{18} \quad \text{(deposit-unit scale)}, \qquad K = 10^{24} \quad \text{(accumulator scale)}.$$

K is composed as $10^{18} \cdot 10^{6}$ (*reward scale* × *price scale*) in the implementation; only the product matters to the semantics.

Clock. The pot reads an integer clock t from a configured field of each incoming message (`t-source`; wall-clock timestamp or scheduler slot). L denotes the clock value at the last global drip. Drips clamp $T \leftarrow \max(t, L)$, so the clock never rewinds and replayed or reordered messages cannot rewind emission; the schedule's real-time meaning is inherited from the chosen source's cadence. At initialization $L \leftarrow t_0$: no emission accrues before the first message.

State. Table 1 lists the state variables with their implementation keys. Users $u, v \in \mathcal{U}$ are addresses; resources $r \in \mathcal{R}$ are identifiers of foreign staked assets.

Deposit quantities are stored in normalized *pot units*: an attested raw amount x_{raw} of resource r becomes

$$n_r(x_{\text{raw}}) \;=\; \lfloor x_{\text{raw}}\, \Lambda / \sigma_r \rfloor\,, \tag{1}$$

so a source asset with 10^d base units per whole token and $\sigma_r = 10^d$ is represented at 18 decimals. Since $\sigma_r \le \Lambda$, normalisation never loses more than it would gain (it scales *up*), and it is exact whenever $\sigma_r \mid \Lambda$. Delegation amounts are already in pot units; σ_r is applied only at the deposit/withdraw boundary.

Symbol	Implementation key	Meaning
Global		
t, L	`t`, `last-drip`	clock; clock at last global drip
Ω	`mint-cap`	total emission cap
N, D	`mint-prop-numerator/-denominator`	per-step emission fraction N/D, $0 \le N \le D$
M	`minted`	cumulative *scheduled* emission, $M \le \Omega$
U	`undistributed-mint`	emission buffered while $W = 0$
A	`accumulator`	global accumulator (K-scaled tokens per weighted unit)
ρ	`accumulator-remainder`	exact carry of the global division, $0 \le \rho < W$
W	`total-weighted-units`	$W = \sum_r w_r Q_r$
b_u	`balances/`u	realized (spendable) balance of u
S	`total-supply`	realized supply $= S_0 + \sum$ settled yield
$\mathcal{D}$	`settlement-dust`	exact accumulated settlement dust (K-scaled)
n_s	`settlements`	count of settlement events
Per resource r		
w_r	`weight`	emission weight per deposit unit
σ_r	`quantity-scale`	raw-unit normalisation divisor, $1 \le \sigma_r \le \Lambda$
A_r	`accumulator`	resource accumulator (K-scaled tokens per deposit unit)
G_r	`last-global-accumulator`	value of A at last resource drip
Q_r	`total-deposits`	$Q_r = \sum_u q_{u,r}$
Per deposit (u, r)		
$q_{u,r}$	`deposits/`u`/quantity`	deposit units currently possessed by u
$o_{u,r}$	`deposits/`u`/originated`	net oracle-attested stake of u
$R_{u,r}$	`deposits/`u`/last-resource-accumulator`	value of A_r at u's last settlement
$\delta^r_{u\to v}$	`deposits/`u`/delegations/`v	delegation edge: units reclaimable by u from v

Table 1: State. An inverted index `users/`u`/deposits/`$r = q_{u,r}$ (kept while $q_{u,r} > 0$ or u has outgoing edges in r) lets reads iterate only over a user's own resources.

3 The emission schedule

Emission is a geometric approach to the cap: each clock step is scheduled to mint the fraction N/D of the *remaining* supply. Let $R = \Omega - M$ be the remaining emission and $\lambda = (D - N)/D$ the per-step retention ratio. The amount scheduled over an interval of $\Delta = T - L$ steps is

$$\mu(R, \Delta) \;=\; \left\lfloor R \cdot \frac{D^\Delta - (D-N)^\Delta}{D^\Delta} \right\rfloor \;=\; R - \left\lceil R\,\lambda^\Delta \right\rceil, \tag{2}$$

with $\mu = 0$ if $\Delta = 0$, $R = 0$ or $N = 0$, and $\mu = R$ if $N = D$. Equation (2) is evaluated with exact big-integer powers when $\Delta \cdot \lceil \log_{10} D \rceil \lesssim 10^3$; beyond that, the two powers are enclosed by interval arithmetic with directed rounding at fixed scales $10^{40}, 10^{60}, 10^{80}$ (tried in order). If the floor of eq. (2) is identical at both interval endpoints—checked both as $\lfloor R(1-\lambda^\Delta) \rfloor$ and as $R - \lceil R\lambda^\Delta \rceil$—that value is returned; otherwise the evaluation *aborts* the enclosing message. The fallback therefore certifies the exact value or fails closed; it never returns an approximation.

Lemma 1 (Schedule properties)**.** *For $0 \le N \le D$, $D \ge 1$, $R \ge 0$:*

(P1) *$0 \le \mu(R, \Delta) \le R$; hence M is non-decreasing and $M \le \Omega$ always.*

(P2) *The remaining supply after a drip is exactly $R' = \lceil R\,\lambda^\Delta \rceil$.*

(P3) *Refinement defers. For $\Delta = a + b$, $\lceil \lceil R\lambda^a \rceil\, \lambda^b \rceil \ge \lceil R\lambda^\Delta \rceil$, and each additional intermediate drip increases the final remaining supply by at most 1. Emission is thus path-dependent only up to one unit per drip, always in the conservative direction.*

(P4) *Perpetual residue. If $0 < N < D$ and $R \ge 1$ then $R' \ge 1$: the final unit of the cap is never emitted in finite time. ($N = D$ instead empties the cap at the next drip.)*

Proof. (P1), (P2): immediate from eq. (2) and $R - \lfloor R(1-\lambda^\Delta) \rfloor = \lceil R\lambda^\Delta \rceil$. (P3): $\lceil x \rceil\, \lambda^b \le (x+1)\lambda^b < x\lambda^b + 1$ and monotonicity of $\lceil \cdot \rceil$. (P4): $R\lambda^\Delta > 0$ so its ceiling is ≥ 1. □

4 The accumulator tower

Distribution proceeds through three levels, with exactly one state-dependent division (by W, remainder carried) and one floor per user settlement. The shape is familiar from fund accounting: the pot maintains

one running per-unit index, each resource scales it by its weight, and an individual account is brought up to date only when someone actually looks at it or changes it—nobody's statement is recomputed on anyone else's schedule:

$$\underset{\text{scheduled}}{\mu} \xrightarrow[\text{carry } \rho]{\div W} \underset{\text{per weighted unit}}{A} \xrightarrow{\times w_r} \underset{\text{per deposit unit}}{A_r} \xrightarrow[\text{floor}]{\times q_{u,r},\ \div K} \underset{\text{realized}}{b_u}\ .$$

Definition 1 (DripGlobal)**.** On any state-touching message carrying clock value t, let $T = \max(t, L)$ and $m = \mu(\Omega - M,\ T - L)$. Update $M \mathrel{+}= m$ and $L \leftarrow T$. Let $P = m + U$ be the distributable amount. Then:

$$\begin{cases} U \leftarrow P, & W = 0 \quad \text{(buffer; } A, \rho \text{ unchanged)} \\ \nu = PK + \rho, \quad A \mathrel{+}= \lfloor \nu / W \rfloor, \quad \rho \leftarrow \nu \bmod W, \quad U \leftarrow 0, & W > 0. \end{cases}$$

If $T = L$ the drip is a no-op (in particular, a buffered U is not released until the clock next advances). Two boundary behaviours are intended: emission scheduled while nothing is staked accrues to U and is captured entirely by whoever is staked at the first distributing drip; and when $KP < W$ the quotient is zero and the whole amount carries in ρ—deferral, never loss, with no minimum-emission threshold.

Definition 2 (DripResource(r))**.** $A_r \mathrel{+}= (A - G_r)\, w_r$; $\quad G_r \leftarrow A$.

Definition 3 (Settle(u, r))**.**

$$y = \lfloor (A_r - R_{u,r})\, q_{u,r} / K \rfloor, \qquad b_u \mathrel{+}= y, \qquad S \mathrel{+}= y, \qquad R_{u,r} \leftarrow A_r.$$

The truncated *dust* $d = (A_r - R_{u,r}) q_{u,r} - yK \in [0, K)$, worth strictly less than one indivisible token unit, is forfeited by u: it is never minted (it appears in neither any balance nor `total-supply`). Gratuitously frequent settlement is therefore marginally value-destructive for the settled user and never inflationary. The state accumulates $\mathcal{D} \mathrel{+}= d$ and $n_s \mathrel{+}= 1$ at every settlement, so the conservation identity of Theorem 1 is exactly checkable in any reachable state.

All three operators are idempotent at a fixed clock value, so they may be applied defensively before any read or mutation. Every operation below begins with DripGlobal; DripResource(r) and Settle(u, r) are applied to exactly the resources and users it touches.

Lemma 2 (Span alignment)**.** *Between any two consecutive resource drips of r, the weight w_r is constant; between any two consecutive settlements of (u, r), the stake $q_{u,r}$ is constant; and at every distributing global drip, $W = \sum_r w_r Q_r$ and $Q_r = \sum_u q_{u,r}$ hold. Consequently every increment ΔA of the global accumulator is later multiplied by exactly the w_r in force when it accrued, and every increment of A_r by exactly the $q_{u,r}$ in force when it accrued.*

Proof. By construction of the operations in Section 5: every mutation of w_r is immediately preceded by DripResource(r), which zeroes the pending span $A - G_r$; every mutation of $q_{u,r}$ is immediately preceded by Settle(u, r), which zeroes the pending span $A_r - R_{u,r}$; and W, Q_r are updated in the same atomic step as the w_r or $q_{u,r}$ mutation that changes them. □

Lemma 2 is the entire correctness argument for laziness: a user's pending yield $(A_r - R_{u,r})\, q_{u,r}/K$ equals the sum over all global drips since their last settlement of $\Delta A_i\, w_{r,i}\, q_{u,r}/K$—precisely the share $P_i \frac{w_r q_{u,r}}{W_i}$ of each distributed amount P_i, up to the two bounded rounding artifacts (ρ-carry, deferred but never lost; settlement floor, < 1 unit per settlement).

Theorem 1 (Conservation ledger)**.** *Define the outstanding (unsettled) scaled entitlement*

$$O \;=\; \sum_r \sum_u \big(A_r + (A - G_r)\, w_r - R_{u,r} \big)\, q_{u,r}.$$

Then in every reachable state,

$$K\,(M - U) \;=\; \rho \;+\; K\,(S - S_0) \;+\; O \;+\; \mathcal{D}, \tag{3}$$

where $\mathcal{D} = \sum_{\text{settlements } e} d_e$ is the exact accumulated settlement dust and $0 \le \mathcal{D} < K \cdot n_s$ for n_s settlement events (both tracked in state; Definition 3). In words: every K-scaled unit of distributed scheduled emission is, at all times, exactly one of (i) carried in ρ, (ii) realized into supply, (iii) outstanding as some user's pending yield, or (iv) forfeited dust.

Proof. Induction over operations. DripGlobal with $W > 0$ adds KP to the left side and $(\Delta A)W - \rho_{\text{old}} + \rho_{\text{new}}$ on the right; $(\Delta A)W$ enters O because $W = \sum_r w_r Q_r = \sum_r \sum_u w_r q_{u,r}$ term-by-term (Lemma 2). DripGlobal with $W = 0$ adds Km to both KM and KU. DripResource moves entitlement between the $(A - G_r)w_r$ and A_r terms of O without changing it. Settle removes $(A_r - R_{u,r})q_{u,r}$ from O and adds $Ky + d$ with $Ky + d = (A_r - R_{u,r})q_{u,r}$ by Definition 3. Stake mutations occur only at zero pending span (Lemma 2), where the affected terms of O vanish. □

Corollary 1. *$S - S_0 \le M - U \le M \le \Omega$: realized supply growth never exceeds scheduled emission, which never exceeds the cap. The shortfall decomposes exactly as ρ/K (deferred to the next distributing drip), O/K (claimable now by settlement), and $\mathcal{D}/K <$ #settlements indivisible units (burned by flooring).*

5 Operations

Each operation is one signed message processed atomically: *any* error (or authority failure) at any point rolls the state back to its value before the message. "Caller" identifies the authorisation required; `from` is always the verified signer of the message. All operations begin by adopting the message clock ($t \leftarrow$ `t-source` field) and running DripGlobal.

Operation 1 (Mint—normalize)**.** *Caller: anyone.* Runs DripGlobal. If a `subject` u is given, additionally DripResource(r); Settle(u, r) for each resource r, realizing u's pending yield into b_u. Settling a foreign subject only accelerates bookkeeping that the subject could perform themselves; it moves no value between parties.

Operation 2 (Register—create or reconfigure a resource)**.** *Caller: the parent process or the pot-wide `mint-authority`; the resource's `weight-authority` may update weight and scale only. Only the first two may set the per-resource authority policies themselves.* For a new or existing resource r with proposed weight w' and scale σ' ($0 \le w'$, $1 \le \sigma' \le \Lambda$), require $Q_r = 0$ if $\sigma' \ne \sigma_r$:

$$\text{DripGlobal}; \quad \text{DripResource}(r); \quad W \mathrel{+}= (w' - w_r)\, Q_r; \quad w_r \leftarrow w'; \quad \sigma_r \leftarrow \sigma'.$$

Past emission is settled to A_r at the old weight before the new weight takes effect (Lemma 2); no user state is touched. A `register` notice is broadcast to subscribers (Section 7). Changing σ_r is only allowed while the resource has no live deposits.

Operation 3 (Deposit—oracle attests new stake)**.** *Caller: the resource's `authority` (oracle policy; fails closed if unconfigured).* Given beneficiary u, resource r, raw amount $x_{\text{raw}} > 0$; let $x = n_r(x_{\text{raw}})$ per eq. (1):

$$\text{DripGlobal}; \quad \text{DripResource}(r); \quad \text{Settle}(u, r); \quad q_{u,r} \mathrel{+}= x; \quad o_{u,r} \mathrel{+}= x; \quad Q_r \mathrel{+}= x; \quad W \mathrel{+}= w_r\, x.$$

The resource must already be registered. New stake earns only from the next clock advance: the drip preceding the quantity change distributes strictly prior emission at the old stake.

Operation 4 (Withdraw—oracle attests unstake)**.** *Caller: the resource's `authority`.* Given u, r, raw amount $x_{\text{raw}} > 0$, $x = n_r(x_{\text{raw}})$. Require $x \le o_{u,r}$: nobody can exit more than they attested in. The exit may directly consume only u's *free* possession—what they staked and have not lent out:

$$\text{free}_{u,r} \;=\; o_{u,r} - \text{out}_{u,r}, \qquad \text{out}_{u,r} = \textstyle\sum_v \delta^r_{u \to v}. \tag{4}$$

The remainder, $x - \text{free}_{u,r}$ if positive, is first recalled from u's outbound delegations by Liquidate (Algorithm 1); if the recall detects a delegation cycle, the cycle is netted and the recall recomputed (netting shrinks $\text{out}_{u,r}$, so each pass strictly reduces total edge mass and the loop terminates). Then:

$$\text{DripGlobal}; \quad \text{DripResource}(r); \quad \text{Settle}(u, r); \quad q_{u,r} \mathrel{-}= x; \quad o_{u,r} \mathrel{-}= x; \quad Q_r \mathrel{-}= x; \quad W \mathrel{-}= w_r\, x.$$

Consequences of the free-possession rule: an exit can never consume possession that inbound delegators are entitled to reclaim, and a *full* exit ($x = o_{u,r}$) leaves u with no outstanding outbound edges at all—delegation state cannot outlive the stake that backed it. Fails (rolling back any partial recall) only if the recall itself cannot be covered; Section 6 discusses why honest exits succeed.

Algorithm 1 Delegation unwinding (mutually recursive with Undelegate; P is the path of edges whose unwind is in progress up the call stack)

procedure COVER(u, v, r, x, P) ▷ make v able to return x units to u
  loop
    if $\delta^r_{u\to v} < x$ **then fail** ▷ edge consumed by a recursive unwind
    if $q_{v,r} \geq x$ **then return**
    LIQUIDATE($v, r, x - q_{v,r}, P$)
  end loop
end procedure
procedure LIQUIDATE(v, r, a, P) ▷ recall a units from v's outgoing edges
  while $a > 0$ **do**
    $E \leftarrow \{(w, \delta^r_{v\to w}) : \delta^r_{v\to w} > 0,\ (v, w) \notin P\}$
    if $E = \emptyset$ **then**
      if some $(v, w) \in P$ has $\delta^r_{v\to w} > 0$ **then**
        $(w^*, \delta^*) \leftarrow$ such path edge of maximal (δ, w)
        NET(the cycle P closes through (v, w^*)); **restart** the outermost operation ▷ cancel min around the loop; edge mass strictly drops
      else fail
      end if
    end if
    $(w^*, \delta^*) \leftarrow$ edge of maximal (δ, w) in E ▷ quantity first, then address tie-break
    $c \leftarrow \min(a, \delta^*)$; UNDELEGATE($v, w^*, r, c$) with (v, w^*) pushed on P; $a \leftarrow a - c$
  end while
end procedure

Operation 5 (Delegate($u \to v,\ r,\ x$))**.** *Caller: u (the signer).* Requires $x > 0$; $u = v$ is a no-op.

$$\mathsf{DripGlobal};\ \mathsf{DripResource}(r);\ \mathsf{Settle}(u, \cdot);\ \mathsf{Settle}(v, \cdot); \qquad \text{require } q_{u,r} \geq x;$$

$$q_{u,r} \mathrel{-}= x; \qquad q_{v,r} \mathrel{+}= x; \qquad \delta^r_{u\to v} \mathrel{+}= x.$$

Q_r and W are unchanged: delegation moves possession, not stake. Yield accrues to the *possessor* v from this point; the edge $\delta^r_{u\to v}$ records u's right to reclaim. A delegation notice

$$\{\mathsf{action}{=}\mathsf{deposit},\ \mathsf{address}{=}u,\ \mathsf{resource}{=}r,\ \mathsf{quantity}{=}x\}$$

is sent to v (meaningful when v is a process; Section 7).

Operation 6 (Undelegate($u \to v,\ r,\ x$))**.** *Caller: u.* Requires $\delta^r_{u\to v} \geq x > 0$; $u = v$ is a no-op. First Cover (Algorithm 1) ensures v can return x units, recursively liquidating v's own outgoing delegations if $q_{v,r} < x$. If the walk closes a delegation cycle, the cycle is netted and the operation retried against the netted state; Cover re-checks the edge each round, so a request exceeding what remains after netting fails cleanly. Then:

$$\mathsf{DripGlobal};\ \mathsf{DripResource}(r);\ \mathsf{Settle}(u, \cdot);\ \mathsf{Settle}(v, \cdot); \qquad q_{v,r} \mathrel{-}= x; \quad q_{u,r} \mathrel{+}= x; \quad \delta^r_{u\to v} \mathrel{-}= x,$$

deleting the edge if it reaches zero, and a notice with action=withdraw is sent to v.

Operation 7 (Balance(u)—read-only)**.** Under *virtual* drips at the current clock (idempotent, so no state change is observable):

$$\mathsf{Balance}(u) \ = \ b_u \ + \sum_{r \in I(u)} \left\lfloor \left(A^\dagger_r - R_{u,r}\right) q_{u,r}/K \right\rfloor,$$

where $I(u)$ is u's inverted index and $A^\dagger_r$ is A_r after DripGlobal; DripResource(r). Balances are therefore *live*: they advance with the clock with no writes at all.

Operation 8 (Transfer interplay)**.** The hosting token device settles the *sender* (Mint with subject = sender) before checking sufficiency, so unclaimed yield is spendable in the same message that first observes it. Recipients are not settled. Transfers then move realized balance only.

6 Delegation-graph dynamics

Fix a resource r and consider the directed graph with edge weights $\delta^r_{u\to v}$ and node values $q_{u,r}$, $o_{u,r}$. In plain terms: o is what each participant actually staked (the oracle's word), q is what they currently hold (and earn on), and edges record who has lent holdings to whom. The system keeps double-entry books over these three:

Lemma 3 (Double entry)**.** *Every operation preserves, for every user and resource,*

$$q_{u,r} \;=\; o_{u,r} \;+\; \underbrace{\textstyle\sum_v \delta^r_{v\to u}}_{\mathrm{in}_{u,r}} \;-\; \underbrace{\textstyle\sum_v \delta^r_{u\to v}}_{\mathrm{out}_{u,r}}. \tag{5}$$

What you hold is what you staked, plus what others have lent you, minus what you have lent out. (Mechanized: `potIdWF` *preservation in* `lean/PotProofs/Ledger.lean`*.)*

Lemma 4 (Graph invariants)**.** *All operations preserve: (i) $q_{u,r} \geq 0$, $o_{u,r} \geq 0$, and $\delta^r_{u\to v} \geq 0$; (ii) $\sum_u q_{u,r} = \sum_u o_{u,r} = Q_r$; (iii) $W = \sum_r w_r Q_r$. Delegate, undelegate, and netting preserve Q_r, W, and every o; only Deposit/Withdraw move them, and only Register moves W independently.*

Delegation cycles arise naturally (delegated units are fungible: v may re-delegate units received from u), but by eq. (5) a cycle is *pure offsetting bookkeeping*: cancelling an equal amount from every edge around a loop changes no possession, no stake, and no net position—each member is entered and left exactly once, like a ring of IOUs a clearing house would strike. The mechanism therefore treats cycles as cancellable, and the two rules of Section 5 keep the graph solvent:

1. **Exits recall first (Operation 4).** A withdrawal may consume only free possession ($o - \mathrm{out}$); anything beyond that is recalled from the exiting party's own outbound edges. With eq. (5) this means an exit leaves $q_{u,r} = \mathrm{in}_{u,r}$: every claim still held *against* the exiting party remains fully backed, and a full exit ($x = o_{u,r}$) retires all of the party's outbound edges. Unbacked reclaim rights—claims that could otherwise bite a counterparty's *future, unrelated* deposits—cannot be created by exiting.
2. **Unwinding is a depth-first search that nets cycles.** Recalls commit as they go: each successful inner Undelegate permanently moves units and strictly reduces total edge mass, and outer failure aborts the whole message. The path of in-progress edges is threaded down the call stack; liquidation never recalls an edge already on the path (its capacity is spoken for), and when only path edges remain, the walk has found a cycle, which is netted by its minimum edge and the operation retried. Termination is two-layered: within one attempt, nesting depth is bounded by the finite edge set; across attempts, every net and every commit strictly reduces total edge mass. Both path-memory and netting are necessary, not defensive: the bare recursion diverges on zero-backed cycles, and a *greedy* recursion without path-memory diverges even on fully solvent graphs (largest-first re-picks a loop edge forever while a collectible escape edge waits). Both divergences and the netting variant's behaviour are mechanized in `lean/PotProofs/Unwind.lean`.

The remaining liveness question—can an honest, oracle-attested exit ever be rejected?—reduces by rule 1 to whether a recall of $\mathrm{out}_{u,r}$ can fail. A recall fails only if some recalled claim is transitively uncollectible, and rules 1–2 remove the known source of uncollectibility (unbacked cycles) at the moment it would arise. We state this as a tested property rather than a theorem: the randomized suite asserts that no withdrawal within the originated bound is ever rejected, and the netting unwind's concrete behaviour is machine-checked on the adversarial states; a general proof would require a max-flow argument we have not mechanized.

7 Composition: child mints

A pot may name a `parent` pot process. The composition uses only the operations above plus message forwarding:

1. *Config and delegation mirroring.* On initialization the child subscribes to the parent's Register broadcasts and delegation notices targeted at the child's process ID. Forwarded notifications are accepted only from the configured parent, and Register notices are applied with both the resource authority and weight authority set to the parent process: the child trusts its parent as the sole oracle for mirrored resources.

2. *Stake mirroring.* When u delegates x units of r to the *child process address* C in the parent, the parent (per Operation 5) moves possession to C—so C itself accrues the *parent*-token yield on x—and the delegation notice instructs the child to execute $\mathsf{Deposit}(u, r, x)$, so u accrues the *child*-token yield on x. This is a yield swap: u exchanges parent emission on x for child emission on x, and the child process treasury captures the parent emission stream that backs its own token's issuance. Child token process configuration should allow parent-delegated Notify, not direct parent-delegated Deposit or Withdraw.
3. *Unwinding.* Undelegate in the parent sends the corresponding action=withdraw notice, and the child executes $\mathsf{Withdraw}(u, r, x)$—which may itself trigger liquidation of u's onward delegations inside the child. Unstaking on the ultimate source network thus propagates: oracle → parent Withdraw (liquidating u's delegation to C if needed) → child Withdraw (liquidating onward), to arbitrary depth, each hop message-atomic in its own process.

Children run their own independent emission schedules (Ω, N, D) on their own token; the tower arithmetic is identical at every level.

Remark 1 (Mirrored quantity scales). Delegation notices carry normalized pot units from the parent's internal state. Child mints still mirror σ_r from the parent's Register broadcast for their own direct raw-token deposit and withdrawal requests, but parent-forwarded delegation notices enter through Notify and are applied as normalized units without re-applying eq. (1).

Corollary 2 (Mirrored deposits are always backed)**.** *A child pot process never re-delegates and holds no oracle stake of its own, so eq.* (5) *degenerates for a child account C to $q_{C,r} = \mathrm{in}_{C,r}$: the child's possession in the parent always exactly equals the edges into it, which equal the mirrored deposits it pays child-token yield on. Child emission is therefore never distributed against phantom stake, and a child can never sit on a delegation cycle (it receives possession but never passes it on).*

8 Complexity

No operation loops over the user set. Costs are per operation, with the drip row stated over an elapsed interval of Δ clock units so the two columns measure the same quantity: advancing time costs the pot logarithmically in Δ, and the naive model linearly. With $|\mathcal{R}|$ resources and $I(u)$ the user's active-resource index:

Operation	pot@1.0	Naive push model
Advance time by Δ (drip)	$O(\log \Delta)$ arith., $O(1)$ state	$\Theta(\lvert\mathcal{U}\rvert \cdot \lvert\mathcal{R}\rvert \cdot \Delta)$
Balance read	$O(\lvert I(u)\rvert)$	$O(1)$ (but stale)
Deposit / Withdraw	$O(1)$ per touched (u, r)	$O(1)$
Delegate / Undelegate	$O(1)$ + edges unwound	—
Re-weight a resource	$O(1)$	$\Theta(\lvert\mathcal{U}\rvert)$ or re-index

The $O(\log \Delta)$ term is the square-and-multiply evaluation of eq. (2); state growth is $O(1)$ per drip regardless of elapsed time. Balances are exact to the current clock at read time, rather than to the last tick.

9 Conclusion

The pot reduces continuous, proportional minting to constant work per interaction. Between the emission schedule and a user's balance sit exactly two rounding points: one division whose remainder is carried forward losslessly, and one floor at settlement whose dust is tracked to the unit. The resulting ledger equation (Theorem 1) accounts for every scaled unit of emission, at every reachable state, as carried, realized, outstanding, or forfeited—nothing else exists. Delegation adds a credit layer with the same discipline: possession equals attested stake plus loans received minus loans made (Lemma 3), exits are bounded by attested stake and settle their loans first, and delegation cycles—provably empty of content—are cancelled the way any clearing system strikes offsetting obligations.

Verification runs in three tiers: the arithmetic core and every invariant above are mechanized in Lean; a pure reference machine mirrors the mechanized definitions; and the production device is bound to that machine by replayed proof-derived traces and randomized lockstep. Two assumptions remain,

and only two: resource authorities attest stake honestly, and the liveness of attested exits—asserted on every step of the randomized suite—rests on tested evidence rather than a mechanized max-flow argument. Within those bounds the mechanism is exact, live, and $O(1)$ in its user set, and it composes: pots stack into trees of mints that pay yield across tokens while the identity of Corollary 2 keeps every mirrored deposit backed, all without ever taking custody of the stake being accounted.

A Worked example

The following trace is one of the package's test vectors (`simple_pot_process_test`), reproduced exactly by the semantics above. Configuration: $\Omega = 10\,000$, $N/D = 1/2$, one resource `oxygen`, initial balance $b_{\text{alice}} = 1000$ ($S_0 = 1000$), clock = scheduler slot, $\sigma = \Lambda$ (quantities pass through unscaled). Slots arrive at $t = 0, 1, 2, 3$.

t	Message	μ	U after	W after	A after (units of K)
0	Register(`oxygen`, w=100)	$\mu(10^4, 0) = 0$	0	0	0
1	Deposit(alice, 10)	$\mu(10^4, 1) = 5000$	5000	1000	0 (buffered: W=0 at drip)
2	Mint (global)	$\mu(5000, 1) = 2500$	0	1000	$7500K/1000 = 7.5K$
3	Transfer(alice→bob, 1)	$\mu(2500, 1) = 1250$	0	1000	$7.5K + 1250K/1000 = 8.75K$

At $t = 1$ the deposit's own drip precedes the stake change (Lemma 2), so the 5000 scheduled while the pot was empty buffers into U; it is released at $t = 2$, the first distributing drip, and captured entirely by the then-current staker (Definition 1). The transfer at $t = 3$ settles the sender: $A_r = 8.75K \times 100 = 875K$, so alice's yield is $\lfloor 875K \cdot 10/K \rfloor = 8750$ and

$$b_{\text{alice}} = 1000 + 8750 - 1 = 9749, \qquad b_{\text{bob}} = 1, \qquad S = 1000 + 8750 = 9750, \qquad M = 8750,$$

matching the vector's assertions. Theorem 1 at this state: $K(M - U) = 8750K$; realized $K(S - S_0) = 8750K$; $\rho = O = \mathcal{D} = 0$ (all divisions were exact here).

References

[1] MakerDAO. *The Dai Savings Rate (`pot.sol`): rate accumulation via the `chi` accumulator.* Multi-Collateral Dai documentation, 2019.

[2] Synthetix. *StakingRewards: `rewardPerTokenStored`.* Contract documentation, 2019.

[3] SushiSwap. *MasterChef: `accSushiPerShare` pool accounting.* Contract source, 2020.