www.veritglobal.com

Investor Deck

Deterministic Settlement Controller - "Pay Only When It's Proven Right"
Audience: CFO, CTO, VP Finance/Ops, Compliance, Payments Engineering
Use case focus: Creator / Ads / Affiliate networks (applies equally to royalties, gig, marketplaces)

Executive Summary

High-scale payout programs (creators, affiliates, royalties) suffer recurring reconciliation noise, dispute volume, and audit exposure. The root causes are (1) non-deterministic dataflows that produce slightly different results when re-run, (2) per-item rounding drift that accumulates across millions of micro-amounts, (3) disbursement decisions not explicitly bound to fresh, provable attestations, and (4) migration/change risk with no cryptographic evidence of correctness.

Our system fixes these by making payouts **deterministic, provable, and governed**: a fixed compute order, integer numerics with one-time rounding and a carry-ledger, cryptographic transcripts + output digests, and an acceptance matrix (Finance ACK, Tax/KYC, optional receipts) with freshness/quorum. Funds move **only** when replay equals the sealed digest **and** the acceptance bundle is satisfied.

Q&A

1) Situation: where the pain comes from (deep detail)

Profile: A global creator network paying 250–500k creators weekly from ad revenue, affiliate sales, bonuses, and clawbacks across 40–100M line-items per window.

Operational realities

- **Data heterogeneity:** events from ad servers, ecommerce, sponsorships; mixed schemas; late and duplicate arrivals.
- **Micro-amounts at scale:** tenths of a cent accrue; naive rounding per item leaks pennies.

- **Frequent corrections:** refunds/chargebacks, sponsor cancellations, policy tweaks.
- **Compliance pressure:** regulators and audit want evidence that every release met reserves, tax/withholding, and (optional) source-of-funds checks.
- **Change velocity:** new payout policies, risk controls, and shard rebalancing each quarter.

Observable symptoms

- **0.2–0.5% payout corrections** each month due to rounding drift and late data.
- **Spike of disputes** ("my numbers don't match"). Support reproduces via spreadsheets; Engineering re-runs jobs and gets slightly different numbers.
- **Slow, brittle reconciliations** across providers (bank, PSP, virtual cards). No single truth for auditors.
- **Change fear:** migrations revert after discovering off-by-cent errors weeks later.

Numeric example
With 5,000,000 micro-line-items/day, naive per-item rounding to cents can introduce up to **0.5¢** per item of drift: **≈ $25,000/day** to reconcile. Over a month, this becomes material and generates disputes.

---

2) Root causes (why traditional stacks fail)

1. **Non-determinism:** unordered map iterations, multi-writer races, wall-clock reads, and hidden I/O make "same inputs → same outputs" untrue.
2. **Rounding drift:** floating-point/early rounding per record accumulates penny errors; no documented bound or deterministic assignment of remainders.
3. **Proof gaps at disbursement:** money moves on pipeline completion, not on **fresh evidence** (reserves, tax, receipts) bound to the computed results.
4. **Change risk with no proofs:** re-sharding and policy changes lack a cryptographic criterion (digest equality) to promote/cutover; rollbacks are ad-hoc.
5. **Audit opacity:** no compact, signed record to replay the exact outputs; disputes devolve into spreadsheets.

---

3) What "good" looks like (target outcomes)

- **Bit-identical replay:** any independent verifier recomputes outputs and gets the *same digest*.
- **Penny-exact books:** one-time rounding with a recorded bound and deterministic assignment of sub-cents.
- **Evidence-gated disbursement:** funds release only on digest equality **and** a satisfied acceptance bundle (freshness + quorum).
- **Safe change:** canary + rollback with bounded loss; re-shard invariance proven by digest equality across versions.
- **Tiny, audit-ready artifacts:** transcripts small enough to store and exchange, yet sufficient for full verification.

---

4) Our solution at a glance

Deterministic Settlement System

- **Fixed, published compute order:** single-writer logs per (partition, window); deterministic fold order.
- **Integer numerics with late quantization:** 128-bit accumulators, one-time rounding at close; **carry-ledger** deterministically assigns sub-cents with ≤ ½ unit bound at the least significant place per allocation.
- **Content-addressed transcripts:** sealed, append-only records containing inputs, fold order, watermark, policy manifest hash, reason-coded decisions, and the **output digest** (hash over canonical allocation records + trailer).
- **Acceptance matrix:** payout header binds {window_id, policy_version, output_digest} to required attestations: **ACK(Finance)**, **CT** (tax/KYC/rights), optional **SPV** (receipts/headers). Enforces freshness *F* and quorum *Q*.
- **Governance & change safety:** deterministic canary cohorts; promotion only after *N* consecutive windows meet digest equality + acceptance; rollback produces a reason-coded, signed trail.

---

5) Key concepts (clear definitions)

- **Window:** logical settlement interval; closes under a **monotone watermark**. Late events beyond policy horizon are transcripted with drop/defer reasons.

- **Partition:** shard with a **single writer** per (partition, window); avoids write races.
- **Deterministic fold order:** fixed order (e.g., bucket → partition); forbids unordered iteration on payout paths.
- **Policy scale (S):** integer 10^k scaling for rounding; applied once at finalization.
- **Carry-ledger:** records per-principal remainders; assigns sub-cents deterministically with a documented bound.
- **Canonical serialization:** byte-wise spec (sorted map keys, fixed-width integers, length-prefixed fields) so heterogeneous systems produce the same digest.
- **Output digest:** cryptographic hash over canonical AllocationRecords plus trailer (watermark, fold-order descriptor, policy manifest hash).
- **Transcript (tiered):** append-only, content-addressed segments (inputs, states, outputs), signed root per window.
- **Payout header:** signed binding of {window_id, policy_version, output_digest} to the **acceptance bundle**.
- **Acceptance bundle/matrix:** required attestations (ACK/CT/SPV) with freshness *F* and quorum *Q*; results are reason-coded (e.g., STALE_PROOF, INSUFFICIENT_QUORUM).
- **Re-shard invariance:** versioned shard function + optional dual-write; promote only if output digests are equal across versions.
- **INVALID state:** on overflow/violations, mark window INVALID; block; recover deterministically from checkpoints.

---

6) Architecture (high-level components)

1. **Policy compiler:** declarative payout rules → constrained deterministic IR.
2. **Deterministic execution engine:** ingestion, ordering, integer accumulation, late quantization, fold in fixed order.
3. **Transcript service:** emits tiered, content-addressed records + window **transcript root**.
4. **Compatibility profile & verifier interfaces:** adapters for Finance ACK, KYC/Tax, rights, receipts (SPV, transparency, headers).
5. **Governance API:** activation, canary, rollback, promotion; signatures and versioning.

---

7) End-to-end flow (a week in the life)

Mon–Thu: ingest & compute

- Normalize events; compute idempotency keys; append to single-writer logs per partition. Late/duplicates handled deterministically.
- Integer accumulators update at native precision; no floating-point on payout paths.

Fri 17:00: window close

- Watermark condition met for all partitions; fold in published order.
- Apply **late quantization** once; record carry-ledger assignments (bound ≤ ½ unit at the last decimal per allocation).
- Seal the transcript; compute **output digest**; sign the transcript root.

Fri 17:05: acceptance gate

- **Payout header** demands: ACK(Finance) on reserves, CT(tax/withholding) for each principal, optional SPV (ad-spend receipt). Freshness *F* and quorum *Q* enforced per manifest.
- Any stale/missing attestation yields a **reason-coded hold**; unaffected principals proceed.

Fri 17:10: disbursement

- Independent replay reproduces the **same digest**; acceptance bundle satisfied → **funds move**. If not, **block** with reason codes in the transcript.

Sat: late refund arrives

- Posts as new events; next window replays deterministically. Support links a dispute to the prior transcript and the correction in the next.

---

8) Controls & guarantees (what we prove)

- **Replay identity:** same inputs → same digest (by construction).
- **Penny-exact numerics:** integer accumulators; one-time rounding; **carry-ledger** bound ≤ ½ unit per allocation.
- **Evidence-gated releases:** digest equality **and** acceptance success required; failures recorded with reason codes.

- **Auditability:** tiny transcripts (vs raw data) suffice to reproduce outputs; signatures separate transcript vs payout domains.
- **Performance predictability:** single-writer logs, fixed fold, and spill/merge in fixed order keep latency predictable under load.

---

9) Safe change (migrations without fear)

- **Deterministic canary cohorts:** stable under replay; no runtime state needed.
- **Promotion rules:** require **N** consecutive windows with (digest equality ∧ acceptance success).
- **Dual-write guard (optional):** emit transcripts under old/new shard functions; require **digest equality** across versions before cutover.
- **Rollback & freeze:** on DIGEST_MISMATCH/STALE_PROOF/INSUFFICIENT_QUORUM, freeze disbursements, re-execute prior manifest against the same transcript, re-run acceptance, then drain quarantine.

---

10) Integration touchpoints

- **Event Input API:** JSON/Avro with idempotency keys; schema and keys provided.
- **Verifier hooks:** pluggable adapters for Finance ACK, CT (tax/KYC/rights), SPV (receipts/headers), each with freshness *F* and quorum *Q*.
- **Payout trigger:** simple boolean gate honoring the **payout header** (digest-match + acceptance success) and returning reason codes on block.
- **Observability:** metrics—replay equality rate, time-to-release (p95), reason-coded blocks by type, net carry remainder.

---

11) KPIs & expected impact

- **Replay-equality rate:** ≥ 99.99% of windows match on first replay.
- **Rounding drift eliminated:** net carry remainders provably bounded and reconciled; penny-exact evidence per window.

- **Time-to-release (p95):** watermark close → authorized disbursement target minutes, not hours.
- **Dispute rate:** projected ↓ 30–60% after 2 cycles due to transcript-based proofs.
- **Change MTTR:** rollback + recovery bounded to a window when failures occur.

Back-of-envelope
5,000,000 line-items/day × 0.5¢ worst-case naive rounding = **$25,000/day** potential drift; deterministic carry removes reconciliation noise and converts disputes into proofs.

---

12) Case study narrative (creator network)

1. **Before:** periodic corrections, rising disputes, and manual reconciliations across PSPs; risky migrations.
2. **After 6 weeks pilot (10% cohort):**
    - Drift eliminated in the cohort; support resolves disputes by linking transcripts.
    - Finance ACK + Tax CT freshness enforced; stale proofs auto-block with reason codes.
    - Canary promotion requires 5 consecutive windows with digest equality + acceptance success; achieved in week 4.
3. **Full rollout:** time-to-release p95 from 6h → 25m; disputes ↓ 42% in 60 days; no regressions during re-shard.

---

13) CFO/CTO FAQs

Q: Why can’t our data warehouse jobs solve this?
A: Warehouses are great for analytics, not for **deterministic settlement**. They allow non-deterministic constructs and lack acceptance gating bound to cryptographic digests and signatures.

Q: What if a provider’s receipt is wrong?
A: The acceptance matrix treats receipts as attestations with freshness and quorum. If a receipt fails or is stale, that principal is held with a reason code; numbers remain deterministic and provable independent of external errors.

Q: Does this slow us down?
A: The system is designed for predictable latency: single-writer logs, fixed fold order, deterministic spill/merge, and compressed transcripts that do not affect digests.

Q: GDPR/PII?
A: Transcripts can avoid PII by using principal IDs; proofs/attestations bind to IDs rather than raw personal data.

---

14) Implementation outline (phased)

- **Phase 0 (2–4 weeks):** policy manifest capture, schema normalization, acceptance matrix design, KPI baseline.
- **Phase 1 (4–8 weeks):** ingest/ordering + deterministic engine + transcript service for a pilot cohort; shadow replay against existing pipeline.
- **Phase 2 (4–6 weeks):** acceptance hooks (Finance ACK, Tax/withholding, optional receipts), payout header wiring, go-live on bounded cohort.
- **Phase 3 (ongoing):** canary → promotion; optional dual-write for re-shard; expand to full population.

---

15) Risks & mitigations

- **Late/missing attestations:** auto-hold with reason codes; retriable without mutating outputs.
- **Overflow/invalid states:** mark **INVALID**, block; recover from checkpoints; bounded MTTR.
- **Hot partitions:** versioned shard function and re-keying protocol; digest equality across versions before promotion.

---

16) Integration with your payment system — what changes on your side

Below is the *minimal set* of changes to slot this system in front of your existing rails (bank/PSP). You **do not** replace your providers; you add a verifiable gate and a few attestations.

A. Insert a payout authorization gate (one call before you pay)

Replace any direct "compute → pay" step with:

1. **Close window** in the settlement engine (we expose an API/queue signal).
2. **Authorize payout**: call Authorize(window_id).
    - Engine verifies **digest equality** and **acceptance bundle** (ACK/CT/optional SPV) with freshness/quorum.
    - Returns **ALLOW** or **HOLD + reason codes** per principal.
3. **Disburse** only the ALLOW set through your existing PSP/bank.
4. **Post receipt** (optional SPV) back to the engine (provider batch id, totals, headers). Held principals will auto-release when proofs are fresh.

B. Add verifier hooks (attestations you already have, but now formal)

Stand up or map the following attestations as small, signed webhooks or messages:

- **Finance ACK** — "reserves ok for window W": {window_id, reserves_ok, signer, expires_at}.
- **Compliance/Tax CT** — per principal or cohort: {principal_id or cohort, status, constraints, expires_at}.
- **Provider receipt SPV (optional)** — proof that provider totals match the payable: {window_id, provider_batch_id, totals, headers/hash, observed_at}.

We provide schemas and signatures; you can front these with your existing risk/tax/finance systems.

C. Add a handful of fields to your payout metadata and finance DB

When you create a provider batch/payout, include the following metadata and store them in your finance/recon tables:

- window_id, policy_version, output_digest (hash), and transcript_root (id or URL).
- provider_batch_id (from your PSP/bank).
- **Reason codes** for any holds (e.g., STALE_PROOF, INSUFFICIENT_QUORUM).
  This links PSP settlements, GL entries, and transcripts into one audit trail.

D. Adopt the event data contract (ingest side)

- Emit usage events with **idempotency keys** (tenant_id, window_id, event_id) and **canonical timestamps**.
- Provide principal_id, currency, amount_native at **integer scale** (we supply the policy scale 10^k).

- Handle late/duplicate events by policy (we expose statuses so you can monitor).

E. Scheduling & cut-offs

- Pick a **window close** (e.g., Fri 17:00 in a declared timezone) and align PSP banking cut-offs.
- Configure **freshness (F)** for attestations (e.g., ≤24h for tax/finance, ≤60m for receipts).

F. Security & signing

- Use **service accounts + mTLS** for attestation hooks.
- Publish/rotate **JWKs**; sign ACK/CT/SPV payloads; verify signatures on our side.

G. Reconciliation changes (what your team stops doing)

- Stop spreadsheet-based re-calcs. For disputes and month-end, **replay the transcript**; numbers must match the stored output_digest.
- Reconcile provider reports against the **payout header** and transcript (provider metadata carries window_id and output_digest).

H. GL & reporting mapping (lightweight)

- Map each **payout window** to a GL batch; attach window_id, output_digest, provider_batch_id.
- (Optional) Export **carry_ledger** assignments for analytics; **no cash impact**—it documents deterministic sub-cent allocation.

I. Pilot path (how to roll in safely)

1. **Cohort-gate** 5–10% of creators/suppliers; run shadow replay for 1–2 weeks.
2. Turn on the **authorization gate** for the cohort; keep others as-is.
3. Promote when you've met **N consecutive windows** with (digest equality ∧ acceptance success) and clean reconciliations.

J. PSP-specific knobs (examples)

- Most PSPs/banks let you attach **metadata** to transfers/batches—use this for window_id and output_digest.
- Enable or schedule **settlement reports** you'll use as SPV receipts (pull within the configured freshness window).
- Keep your existing bank accounts, payout schedules, funding flows; the engine **governs the decision**, it doesn't replace the rail.

Net-net: you add one authorization call, two or three small attestations, a few metadata fields, and a replay-based audit path—without changing providers or rerouting funds.

17) What you must send us (inputs & configs) — crystal clear, no rebuild required

You can wire this up with **flat files (CSV/JSON)**, a **read-only DB view**, or **streaming** (Kafka/Pub/Sub/Webhooks). Pick one—no platform rewrite needed. Below are the **minimum** and **recommended** inputs.

A. Tier-0 (minimum viable) — a single daily file or feed of events

One record per earning/adjustment/refund.

Required fields

- event_id (string) — globally unique for ≥ 12 months.
- ts_occurred (ISO8601 UTC) — when the underlying business action happened.
- principal_id (string) — the payee/creator/supplier stable ID you already use.
- currency (ISO-4217) — e.g., USD, EUR.
- amount_minor (integer) — **net** amount in minor units (cents, pence, etc.). Use negative for refunds/chargebacks.
- source_type (enum) — earning | bonus | adjustment | refund | reversal (choose best fit).

Nice-to-have (if you have them already)

- order_id, campaign_id, product_id, region, external_ref (strings)
- gross_minor, fees_minor, tax_minor (integers) — if you want us to compute from gross.

Sample CSV (Tier-0)

event_id,ts_occurred,principal_id,currency,amount_minor,source_type,external_ref

EVT-9f3a,2025-09-05T16:22:10Z,CRE-18472,USD,117,earning,ORD-1029

EVT-9f3b,2025-09-05T18:03:51Z,CRE-18472,USD,-17,refund,ORD-1029

We derive window_id from ts_occurred and configured timezone; we assign bucket_id internally. No need to change your schemas to add those.

Delivery options: push to S3/GCS/SFTP; or we pull from a read-only view; or stream via Kafka/Webhook. We provide adapters for each.

B. Tier-1 (recommended) — principal registry snapshot (daily or on change)

One row per creator/supplier/payee; no PII required if you can reference tokens.

Required fields

- principal_id (string) — stable key matching event feed.
- payout_method_token (string) — your PSP/bank customer ID or token (no raw bank details).
- tax_status_code (string) — e.g., US_W9, US_W8BEN, EU_VAT_REG, or a code you already use.
- withholding_rate_bps (int) — if applicable (basis points, e.g., 1000 = 10%).
- residency_country (ISO-3166-1 alpha-2) — e.g., US, DE.
- (Optional) preferred_currency, hold_flags, contract_id.

Sample CSV (Tier-1)

principal_id,payout_method_token,tax_status_code,withholding_rate_bps,residency_country

CRE-18472,psp_cus_49ab,US_W9,0,US

C. Tier-2 (attestations) — tiny messages you already know how to produce

These are small JSON payloads (or rows in a view) that confirm facts **at payout time**.

1. **Finance ACK (window-level)** — reserves/funding OK

{ "window_id": "2025-09-05/weekly", "reserves_ok": true, "signer": "fin-ops@yourco", "expires_at": "2025-09-06T00:00:00Z" }

2. **Compliance/Tax CT (principal-level or cohort)** — cleared to pay

{ "principal_id": "CRE-18472", "status": "cleared", "constraints": [], "expires_at": "2025-09-06T00:00:00Z" }

3. **Provider receipt SPV (optional, window-level)** — provider totals match payable

{ "window_id": "2025-09-05/weekly", "provider_batch_id": "psp_batch_8831", "totals_minor": 41833741, "headers_hash": "0xabc..." }

You can publish these via webhook, message bus, or a materialized DB view we read. We validate freshness and signatures.

D. One-time configuration (we capture this with you)

- **Window schedule & timezone** — e.g., close Fridays 17:00 America/New_York.

- **Currencies supported & scale** — we default to minor units per ISO; can override with policy_scale if needed.
- **Acceptance matrix** — which attestations (ACK/CT/SPV), freshness F, quorum Q.
- **Rounding policy** — ties-to-even vs ties-up; carry assignment order (we recommend default).
- **Cohort definitions** — for canary/pilot (lists, predicates, or views).

E. Minimal reconciliation metadata (in your PSP/GL)

When you create a payout batch at your PSP/bank, include two metadata fields we return to you:

- window_id and output_digest (hash). Store alongside provider_batch_id in your finance tables.

F. Data quality expectations (lightweight)

- **Uniqueness:** event_id is globally unique for ≥ 12 months.
- **Clock:** ts_occurred in UTC (we accept timezone + offset too).
- **Sign:** use negative amount_minor for refunds/chargebacks.
- **Latency:** late events are fine—policy controls whether they land in current/next window.

Bottom line: If you can export a daily CSV of transactions and a simple registry of payees, plus two tiny attestations at payout time, our system will do the rest—no core rebuild required.

18) If you pay using NetSuite — exact mapping & steps

Goal: keep NetSuite as **system of record** for AP and cash, add our **authorization gate**, and avoid any ERP rebuild.

Choose one path

- **Path A — Pay via PSP/bank outside NetSuite, record in NetSuite (fastest).**
  Use your PSP to move funds; we create **Vendor Bills** (or Journals) and optional **Vendor Payments** for accurate books.
- **Path B — Originate ACH/SEPA from NetSuite (Electronic Bank Payments SuiteApp).**
  Use NetSuite to create the bank file from **Pay Bills**; we gate which bills are eligible.

Common one-time setup in NetSuite

1. **Custom Body Fields** on Vendor Bill & Vendor Payment:

    - custbody_payout_window_id (Text 64) — e.g., 2025-09-05/weekly
    - custbody_output_digest (Text 128) — truncated digest string
    - custbody_provider_batch_id (Text 64) — from PSP/bank file (if used)
    - custbody_transcript_url (URL) — link to sealed transcript viewer
2. **Accounts**
    - Expense: Creator Payout Expense
    - AP: your standard Accounts Payable
    - (Optional) Clearing: PSP Clearing if paying outside NetSuite and auto-matching bank feeds
3. **Vendors (creators/payees)**
    - Use **Vendor** records (1099 if applicable). Set **External ID = principal_id** from our feed.
    - If you plan **Path B**, maintain bank details & payment method (ACH/SEPA) per vendor (Electronic Bank Payments SuiteApp).

Data you give us (already covered, NetSuite-specific notes)

- **Events CSV/JSON:** include principal_id that equals Vendor **External ID** in NetSuite.
- **Principal registry:** map principal_id → vendor internal/external id, tax_status_code for 1099/VAT reporting (we don't need raw TINs).
- **Attestations (Finance ACK, CT):** can be produced from NetSuite (Saved Search + webhook) or your finance systems; we just need the small JSONs.

What we return to you (per window)

We can deliver a **ready-to-import CSV** for Vendor Bills (and, if desired, Vendor Payments). You can schedule a NetSuite **CSV Import** or use **REST Web Services**.

Path A — Pay via PSP/bank, record in NetSuite

Step A1: Create Vendor Bills (summary, one per vendor)

- Transaction type: **Vendor Bill**
- Header fields:
    - Vendor: by **External ID** (principal_id)

    - Date: window close date
    - Memo: Creator payout — window {window_id}
    - Custom: custbody_payout_window_id, custbody_output_digest, custbody_transcript_url
- Expense line(s):
    - Account: Creator Payout Expense
    - Amount: **amount in currency units** (we convert from minor units)
    - Department/Location/Class: optional

Sample CSV (Vendor Bills)

External ID,Vendor,Date,Currency,Memo,custbody_payout_window_id,custbody_output_digest,custbody_transcript_url,Expense Account,Expense Amount

BILL-2025-09-05-CRE-18472,CRE-18472,9/5/2025,USD,"Creator payout — window 2025-09-05/weekly",2025-09-05/weekly,0xabc123...,https://transcripts.example/w/2025-09-05,Creator Payout Expense,1.17

Tip: Use External ID on the Bill so re-imports are **idempotent** (updates, not duplicates).

Step A2: Record the external payment
Two options:

- Create a **Vendor Payment** per Vendor Bill (if you want AP aging accurate and payment history). Populate custbody_provider_batch_id with the PSP payout batch.
- Or, if your PSP consolidates many creators into one transfer, post a **Journal Entry** to clear AP and move cash via **PSP Clearing**; attach the provider report and transcript URL.

Vendor Payment CSV (optional)

External ID,Vendor,Date,Account (AP),Memo,custbody_payout_window_id,custbody_provider_batch_id,Apply Bill External ID,Payment Amount

PAY-2025-09-05-CRE-18472,CRE-18472,9/5/2025,Accounts Payable,"Payout — window 2025-09-05/weekly",2025-09-05/weekly,psp_batch_8831,BILL-2025-09-05-CRE-18472,1.17

If CSV-applying payments is cumbersome, use a small **SuiteScript Map/Reduce** to fetch unpaid Bills where custbody_payout_window_id = X and create matching Vendor Payments.

Bank reconciliation

- Use **Bank Feeds** to ingest the PSP settlement; match on provider_batch_id and total. Our window_id & digest live on the Bill/Payment for full traceability.

Path B — Pay from NetSuite via Electronic Bank Payments (EBP)

Prereqs: EBP SuiteApp installed; vendors have ACH/SEPA details and payment method.

Step B1: Create Vendor Bills — same CSV as Path A.

Step B2: Gate eligibility using our authorization

- Run a Saved Search Bills — Eligible to Pay with filter custbody_payout_window_id = {window_id} and **Status = Open**.
- (Optional) Add a custom checkbox Eligible to Pay that our integration sets only for **ALLOW** vendors.

Step B3: Pay Bills → EBP

- In **Pay Bills**, filter by the Saved Search; pay all **Eligible to Pay**.
- EBP generates the NACHA/SEPA file. Put window_id in the **Payment Memo** and (if format allows) include the truncated output_digest in addenda.
- We consume the EBP **Payment File Administration** ID as provider_batch_id.

Step B4: Post provider receipt (SPV)

- We (or you) post a tiny SPV JSON referencing the EBP batch and totals. Any holds remain as Bills until attestations are fresh.

Field mapping — our concepts ↔ NetSuite

Our Concept	NetSuite Object/Field
principal_id	Vendor **External ID** (or internal ID)
window_id	custbody_payout_window_id on Bill/Payment
output_digest	custbody_output_digest on Bill/Payment
provider_batch_id	custbody_provider_batch_id on Payment (and Bill memo if desired)
transcript URL	custbody_transcript_url (and file attachment in File Cabinet)
amount_minor	Vendor Bill **Expense Amount** (converted to currency units)

Operational notes

- **Scale:** Creating hundreds of thousands of Bills in one go is heavy. Use **scheduled CSV imports** in chunks (e.g., 25–50k) or REST with a Map/Reduce script.
- **Taxes/1099:** Keep 1099 classification on Vendor; our **CT** attestation drives whether a Bill is eligible; NetSuite 1099 reporting reads Vendor totals as usual.
- **Subsidiaries & multi-currency:** Include Subsidiary and Currency in the CSV if OneWorld/multi-currency are enabled. We can supply per-subsidiary files.
- **Audit trail:** Attach the transcript PDF/hash to the Bill (File Cabinet). Auditors can replay from the URL and match output_digest.

Bottom line for NetSuite:

- You import **one Bill per payee per window** (or per cohort) with window_id & output_digest.
- You pay either **in NetSuite** (EBP) or **via PSP** and record Vendor Payments/Journals.
- Every transaction is linked to a replayable transcript, so Finance can prove *why* every penny moved.

19) CFO case: why this matters beyond pennies

Even if individual misrounds look trivial, large-scale payout programs face **asymmetric, compounding, and tail risks** that dominate the cost of "a cent here or there." This system addresses those risks directly.

A. Quantified levers (illustrative — tune to your numbers)

- **Policy/migration drift (systemic, not random):**
  Example: 250k creators × $80 avg = **$20M/week**. A subtle 0.2% logic error during a policy change → **$40,000** leakage **per week** (≈ **$2.08M/year**) if undetected. Deterministic replay + canary + bounded-loss caps prevent broad release and surface exact variance before cash moves.
- **Dispute OPEX:**
  If **3%** of creators open a ticket monthly (7,500 tickets) at ~$10 all-in per ticket, that's **$75k/month** (≈ **$900k/year**). Transcript-based proofs reliably cut disputes by ~40% in similar programs → **$360k/year** saved, plus faster close.
- **FX slippage & multi-currency reconciliation:**
  If 30% of payouts are non-USD (~$6M/week), even a 10 bps pricing/rounding mismatch costs **$6,000/week** (≈ **$312k/year**). Canonical scaling + one-time quantization keeps provider/books/replay on the same penny.

- **Working capital & close predictability:**
  Bringing close forward and removing rework can reduce average revolver draw. Saving just **3 days/month** on a $10M float at 8% APR saves ≈ **$78.9k/year** — and reduces late-close risk.

None of the above includes the cost of re-issuing payments, clawbacks, customer concessions, or audit overruns. Those typically dwarf the penny math.

B. Tail-risk controls (where the real money is)

- **Release gating on evidence:** Money moves only when **digest equality** holds **and** the **acceptance bundle** (Finance ACK, Tax/CT, optional SPV) meets freshness/quorum. This blocks payouts when reserves aren't ok, tax attestations are stale, or provider receipts don't reconcile.
- **Bounded blast radius on change:** Canary cohorts + bounded-loss caps + dual-write (optional) mean a defect cannot propagate across the full population before detection. Promotion requires consecutive windows with equality.
- **Irreversible mistakes prevented:** Decimal-scale errors, unordered reduce bugs, or schema drift all surface as **digest mismatch** or **INVALID** states — stopping disbursement and documenting the reason with signed transcripts.
- **Third-party provability:** A tiny, signed transcript lets counterparties (auditors, partners, acquirers) independently replay to the **same result**. This reduces diligence friction and audit fees and creates credibility you can monetize (better partner terms, lower risk premiums).

C. Strategic upside

- **Creator trust → retention & mix:** Transparent, provable payouts improve creator NPS and decrease churn, protecting high-value cohorts (top 10% often drive the majority of GMV/engagement). Even a 0.5–1.0% churn improvement on top cohorts materially lifts contribution margin.
- **New products:** With provable cashflows, you can safely introduce features like **accelerated payouts** or **revenue advances**, often at lower funding spreads because the risk is objectively verifiable from transcripts.

D. CFO one-liners

- "**We don't pay on hope; we pay on proofs.** If proofs are stale or reconciliation fails, the system won't release cash — and it tells us exactly why."
- "**We've eliminated rework.** Disputes are resolved by replaying the sealed transcript, not by rebuilding spreadsheets."

- "**Change is controlled.** Every policy change is canaried with a capped exposure and must prove digest equality before promotion."
- "**Audit is a byproduct.** Our payout ledger has cryptographic receipts per window; auditors can self-verify."

Technical Appendixes

Appendix A — Reason codes (minimum)

DIGEST_MISMATCH, STALE_PROOF, INSUFFICIENT_QUORUM, INVALID_SIGNATURE, POLICY_VIOLATION, VERIFIER_UNAVAILABLE, OVERFLOW, MISSING_INPUT.

Appendix B — Acceptance matrix patterns (examples)

- **Creator/Ads:** ACK(Finance), CT(tax/withholding), SPV(ad-spend receipt); F ≤ 24h; Q = 2-of-N incl. Finance.
- **Royalties:** ACK, CT(rights/contract attestation); F ≤ 24h; Q ≥ 2.
- **Travel/OTA:** ACK, CT(risk/compliance), SPV(custody/issuer proof); F ≈ 5–60m; Q ≥ 2.

Appendix C — Data shapes (abridged)

AllocationRecord: window_id | policy_version | principal_id | bucket_id | amount_native | carry_delta
Trailer: watermark | fold_order_desc | policy_manifest_hash
PayoutHeader: binds {window_id, policy_version, output_digest} to acceptance bundle (*F, Q, expiry, kinds*).

Appendix D — Math & rounding bounds (intuition)

- Accumulate in integers at native scale; no floating-point on payout paths.
- Apply **ROUND(S, ties-to-even)** once at finalization.

- Track sub-unit remainders in the carry-ledger; assign deterministically (e.g., ascending principal_id with stable tiebreakers).
- Enforce bound: ≤ ½ of the last decimal place **per allocation**; record assignments in the transcript.

Appendix E — End-to-end 5-line example (VGOS + NetSuite)

Scenario
Window **2025-09-05/weekly**. Three creators. Your policy adds a **1% bonus** on net earnings (calculated at sub-cent precision, then quantized with VGOS's deterministic carry-ledger). Finance must ACK reserves; Tax/Compliance must clear each creator.

A) What you send VGOS — 5 event lines (Tier-0)

event_id,ts_occurred,principal_id,currency,amount_minor,source_type,external_ref

EVT-101,2025-09-05T16:22:10Z,CRE-18472,USD,117,earning,ORD-1029

EVT-102,2025-09-05T18:03:51Z,CRE-18472,USD,-17,refund,ORD-1029

EVT-103,2025-09-05T19:45:00Z,CRE-18472,USD,5,earning,ADJ-55

EVT-201,2025-09-05T12:01:09Z,CRE-29011,USD,33,earning,CAM-889

EVT-301,2025-09-05T09:12:34Z,CRE-99007,USD,49,earning,VID-223

B) Your registry snapshot (Tier-1)

principal_id,payout_method_token,tax_status_code,residency_country

CRE-18472,psp_cus_49ab,US_W9,US

CRE-29011,psp_cus_7kq2,US_W9,US

CRE-99007,psp_cus_m1d8,UNKNOWN,BR

C) VGOS computes deterministically (integer math + late quantization)

Net earnings before bonus

- CRE-18472: 117 − 17 + 5 = **105¢**
- CRE-29011: **33¢**
- CRE-99007: **49¢**

Policy bonus 1% (computed at sub-cent precision)

- 18472 → 1.05¢
- 29011 → 0.33¢
- 99007 → 0.49¢
  Total exact bonus: 1.87¢ → quantized to **2¢**.

Deterministic carry-ledger assignment (largest fractional remainder first)

- 18472 rounded = 1¢ (from 1.05¢)
- 99007 gets **+1¢ carry** (from 0.49¢)
- 29011 = 0¢ (from 0.33¢)

Final allocations (per principal)

Principal	Net (¢)	Bonus (¢)	Payout (¢)
CRE-18472	105	1	**106**
CRE-29011	33	0	**33**
CRE-99007	49	1	**50**

VGOS seals the transcript and computes **output_digest** (example): 0x9e8f3c…71a5.

D) Attestations (Tier-2) you/your systems provide

Finance ACK (window-level)

{ "window_id": "2025-09-05/weekly", "reserves_ok": true, "signer": "fin-ops@yourco", "expires_at": "2025-09-06T00:00:00Z" }

Compliance/Tax CT (principal-level)

{ "principal_id": "CRE-18472", "status": "cleared", "expires_at": "2025-09-06T00:00:00Z" }

{ "principal_id": "CRE-29011", "status": "cleared", "expires_at": "2025-09-06T00:00:00Z" }

{ "principal_id": "CRE-99007", "status": "hold_missing_tax", "expires_at": "2025-09-06T00:00:00Z" }

(Optional) **Provider receipt SPV** (will be posted *after* payment if Path A is used)

E) VGOS authorization result

Principal	Amount (¢)	Decision
CRE-18472	106	**ALLOW**
CRE-29011	33	**ALLOW**

CRE-99007	50	**HOLD — reason: CT(MISSING_OR_STALE)**

F) What happens in NetSuite (two paths)

Path A — Pay via PSP/bank, record in NetSuite

1. **Vendor Bills** (one per allowed/held principal; held bills simply won't be paid yet)

External ID,Vendor,Date,Currency,Memo,custbody_payout_window_id,custbody_output_digest,custbody_transcript_url,Expense Account,Expense Amount

BILL-2025-09-05-CRE-18472,CRE-18472,9/5/2025,USD,"Creator payout — window 2025-09-05/weekly",2025-09-05/weekly,0x9e8f3c…71a5,https://transcripts.example/w/2025-09-05,Creator Payout Expense,1.06

BILL-2025-09-05-CRE-29011,CRE-29011,9/5/2025,USD,"Creator payout — window 2025-09-05/weekly",2025-09-05/weekly,0x9e8f3c…71a5,https://transcripts.example/w/2025-09-05,Creator Payout Expense,0.33

BILL-2025-09-05-CRE-99007,CRE-99007,9/5/2025,USD,"Creator payout — window 2025-09-05/weekly (HOLD — CT)",2025-09-05/weekly,0x9e8f3c…71a5,https://transcripts.example/w/2025-09-05,Creator Payout Expense,0.50

2. **External payment via PSP** for ALLOWed principals only; PSP batch psp_batch_9001 totals **$1.39**.
3. **Vendor Payments** in NetSuite (optional) to reflect the PSP payment:

External ID,Vendor,Date,Account (AP),Memo,custbody_payout_window_id,custbody_provider_batch_id,Apply Bill External ID,Payment Amount

PAY-2025-09-05-CRE-18472,CRE-18472,9/5/2025,Accounts Payable,"Payout — window 2025-09-05/weekly",2025-09-05/weekly,psp_batch_9001,BILL-2025-09-05-CRE-18472,1.06

PAY-2025-09-05-CRE-29011,CRE-29011,9/5/2025,Accounts Payable,"Payout — window 2025-09-05/weekly",2025-09-05/weekly,psp_batch_9001,BILL-2025-09-05-CRE-29011,0.33

4. **SPV receipt** back to VGOS to close the loop:

{ "window_id": "2025-09-05/weekly", "provider_batch_id": "psp_batch_9001", "totals_minor": 139, "headers_hash": "0x7af…" }

(Held bill for CRE-99007 remains open; when CT clears, VGOS will return **ALLOW** next window and you can pay it.)

Path B — Pay from NetSuite (EBP SuiteApp)

- In **Pay Bills**, filter by custbody_payout_window_id = 2025-09-05/weekly and **Eligible to Pay = true** (VGOS sets this only for ALLOW).
- Generate the ACH/SEPA file; use the **Payment File Administration** ID as provider_batch_id.
- Post an SPV receipt referencing that batch ID and totals.

Audit & replay

- Any dispute (e.g., "I was short by a cent") → open the transcript URL, replay, show the carry-ledger assignment and the acceptance decisions. The **output_digest** on each Bill/Payment ties books to the exact computation.

Takeaway: In five lines of input and two tiny attestations, VGOS produces penny-exact, provable payouts, gates disbursement on facts, and writes clean, auditable entries into NetSuite—without changing your providers or rebuilding your ERP.

Appendix F — Tail-risk scenarios (and how the system contains them)

Scenario	Typical impact if undetected	How the system contains it
Policy bug introduces a 0.2% over-allocation	$40k/week on a $20M run-rate; ~$2.08M/year if persistent	Canary cohort + bounded-loss cap; transcript variance flags before release; promotion blocked until equality proved
Decimal scale misconfig (e.g., cents vs units)	Catastrophic (×100 payouts)	Digest mismatch/INVALID on first replay; disbursement blocked; reason-coded transcript for audit
Stale tax/KYC status pays blocked payee	Regulatory penalties, clawbacks, reputational damage	Acceptance matrix requires fresh CT per payee; stale → automatic HOLD with reason code
Provider file drift (format or totals)	Unreconcilable totals; manual rework; delayed close	SPV receipt check fails; window blocked; provider_batch_id + headers hashed into transcript

Unordered iteration in legacy job	Silent per-run variances; rolling recon effort	Deterministic IR forbids unordered reduce; fixed fold-order makes replays bit-identical
Re-shard migration drift	Split-brain numbers by cohort/shard	Versioned shard function + dual-write guard; cutover only on digest equality across versions

Takeaway: Even if "pennies net out," **systemic drift, compliance misses, and change risk don't**. This system eliminates those costs and caps tail risk before cash moves.

This Q&A is designed for investor due diligence and fundraising conversations. Financial projections are estimates based on market research and comparable company analysis.