TECHNICAL WHITE PAPER

Governance in the Movement Path

Quality, contracts, masking, schema-drift policy, column lineage, and SLAs — enforced inside the pipe, per page, before any write. One config block per guardrail, zero per-connector code, and a mask-first ordering guarantee that is fixed in one place and proven by a test.

faucet-stream · ~12 min read · Benchmark run pending

Online version: https://faucet-hq.github.io/papers/governance-in-the-movement-path

---

Abstract

The prevailing ELT pattern moves data first and governs it later: connectors land raw rows in a warehouse, and a separate stack of tools then tests, masks, and catalogs them. By then the sensitive field has already landed, the bad row is already queryable, and lineage has to be reverse-engineered. faucet-stream takes the opposite stance: governance is a first-class stage *in the movement path*, applied to every page of records before it reaches the sink. This paper describes the model, the ordering guarantee at its core — a masked field never reaches a sink, a dead-letter queue, or a lineage event in the clear — and the honest limits of the current implementation.

1. The problem: governance after the fact

A modern ELT stack is assembled from parts: a movement tool (Fivetran, Airbyte, Meltano) loads raw data; an orchestrator schedules it; and a governance layer — dbt tests, Great Expectations, a catalog, a data-observability monitor — runs *downstream*, after the load. Each part is good at its job, but the seam between them is where risk lives:

- **PII lands first.** Masking that happens after load means the raw value existed in the target — and often in staging, logs, and backups — before it was ever obscured.
- **Bad rows are already queryable.** Post-load assertions flag violations after they have been written, so a downstream consumer may read them in the window before the check runs.
- **Drift is discovered late.** A changed source shape surfaces as a broken model run, not as a decision made at ingest time.
- **Lineage is reconstructed.** A separate catalog infers relationships after the fact rather than recording them as the data moves.

2. The thesis: govern in the pipe

faucet-stream folds governance into the same streaming loop that moves the data. Every pipeline processes the source in bounded pages of records; each guardrail is a stage applied to a page before it is handed to the sink. The result is that the decision — mask this, reject that, quarantine the other — is made *before* the write, not after, and it is made in exactly one place regardless of which of the dozens of sinks the pipeline targets.

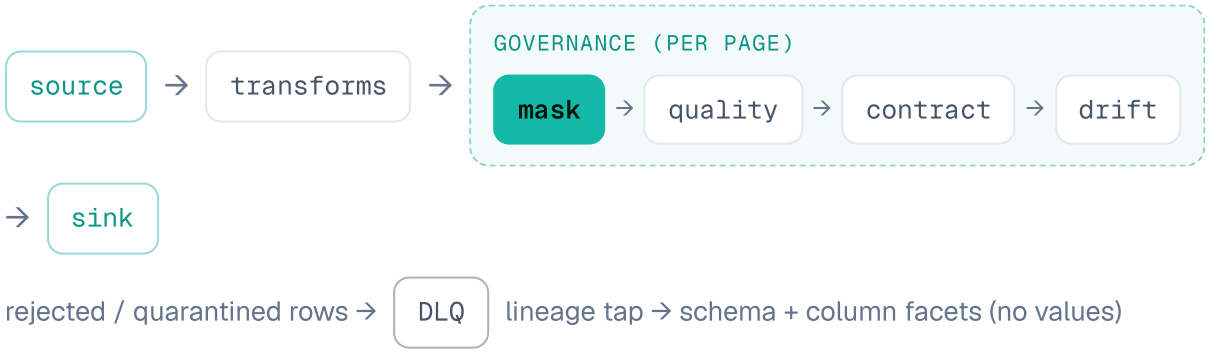


Figure 1 — governance stages run per page, before the sink. Masking is first, so every downstream surface sees masked data.

3. The six guardrails

Each is a config block; the engine applies them uniformly for every sink.

PII masking

Redact or hash sensitive fields. Runs first (see §4). Hashing is a keyed HMAC-SHA256, deterministic so hashed fields stay joinable across tables.

Data-quality checks

Row- and batch-level assertions. A failure either aborts the run or is quarantined to a dead-letter queue.

Data contracts

A versioned promise about the output shape — required fields, types, enums — enforced on the wire before the sink; breaches abort or quarantine.

Schema-drift policy

Detect when the source shape changes and decide what happens: `warn · evolve · ignore · quarantine · fail`.

Column-level lineage

OpenLineage events with dataset schemas and column-level edges derived from the transform chain — not hand-annotated.

Freshness & volume SLAs

Declare max staleness, a minimum row count, or a learned volume baseline; violations increment a Prometheus counter you can alert on.

4. The ordering guarantee (the core contribution)

The claim that makes in-flight governance trustworthy is an ordering property: **masking runs before quality, contract, and drift, and before the row reaches the sink, the dead-letter queue, or a lineage event.** This is not a convention that each connector must remember — it is fixed once, in the single streaming loop that every pipeline runs, in this order:

1. **Mask** — rewrites the page's records in place.
2. **Quality** — builds any quarantine envelopes from the (now masked) records.
3. **Contract** — enforces shape on the masked records.
4. **Drift** — applies the drift policy.

5. **Sink / DLQ** — writes surviving rows; routes rejected rows to the DLQ.

> Because masking rewrites the page before the quality and contract stages build their dead-letter envelopes from those same records, **a quarantined row carries the masked payload, not the raw one.** This is verified by a unit test that asserts the DLQ envelope's `email` is `"***"`, not the original address — the guarantee is enforced by the code and checked in CI, not merely documented.

Lineage is safe by a second mechanism: OpenLineage events emitted by faucet-stream never carry record *values* — only schema facets (field name and type) and column-lineage edges (field-name to field-name). No field value, masked or not, is ever placed in a lineage event.

5. Zero per-connector code

Every guardrail is invoked once, at the framework level, inside the shared run loop — no source or sink implements masking, quality, contracts, or drift. A connector cannot forget a guardrail, and it cannot bypass one: when masking is configured, faucet-stream disables its columnar (Arrow) fast path for that pipeline, so the data is always routed through the governed row loop rather than an ungoverned columnar shortcut. Governance is a property of the engine, not of any connector.

6. Configuration

All six guardrails on one pipeline. The order of blocks in the file doesn't matter: the engine fixes the runtime order described in §4. This config passes `faucet validate`, and the validator itself enforces two of the rules in §9: a `quarantine` action needs the `dlq:` sink, and the freshness SLA needs the `state:` block that holds run history.

```
version: 1
name: orders_governed

lineage:                                # schema + column-level lineage, no values
```

```
  namespace: prod.warehouse
  include_column_lineage: true
  transport: { type: http, config: { url: "${env:OPENLINEAGE_URL}" } }

sla:                                  # checked after every run
  max_staleness_secs: 86400
  min_rows_per_run: 1

pipeline:
  source:
    type: postgres
    config:
      connection_url: "${env:PG_URL}"
      query: "SELECT id, email, ssn, amount, status FROM orders"

  # Governance runs in the pipe, per page, before the sink, in this order.
  masking:
    key: "${env:MASKING_KEY}"           # keyed HMAC-SHA256: deterministic and joinable
    rules:
      - { name: hash-email, match: { fields: [email] }, action: { type: hash } }
      - { name: redact-ssn, match: { fields: [ssn] },   action: { type: redact } }

  quality:
    record:
      - { type: not_null, field: email,  on_failure: quarantine }
      - { type: compare,  field: amount, op: gte, value: 0, on_failure: abort }

  contract:
    version: "1"
    on_breach: quarantine
    allow_extra_fields: true
    fields:
      - { name: id,     type: integer }
      - { name: status, type: string, enum: [active, churned] }

  schema:
    on_drift: quarantine                # warn | evolve | ignore | quarantine | fail
```

```
  dlq:                                  # quarantined rows land here, already masked
    sink: { type: jsonl, config: { path: ./dlq/orders.jsonl } }


  sink:
    type: bigquery
    config:
      project_id: my-project
      dataset_id: raw
      table_id: orders
      write_mode: upsert
      key: [id]
      auth: { type: service_account_key, config: { json: "${env:GCP_KEY_JSON}" } }


  state:                                # bookmarks + run history (the freshness SLA reads
it)
    type: file
    config: { path: ./.faucet-state }
```

7. In-flight vs. downstream governance

CAPABILITY	FAUCET-STREAM (IN-FLIGHT)	ELT + GOVERNANCE STACK (POST-LOAD)	MOVEMENT-ONLY TOOLS
PII masked before it lands	Yes — before sink, DLQ, and lineage	No — masked (if at all) after load	No
Bad rows kept out of the target	Yes — quarantined to a DLQ pre-write	Partly — flagged after load	No
Contract enforced	On the wire, before the sink	Post-load assertions	No
Schema drift handled	Policy per run (5 actions)	Detected downstream	No
Column lineage	Derived from the transform chain	Separate catalog tool	Limited / add-on
Extra services to run	None — one config block	dbt + GE + catalog + monitor	Vendor add-ons

8. Performance

Governance is applied per page, so its memory cost is bounded by the batch size, not the dataset. The CPU overhead is the cost of the configured checks — a hash per masked field, a predicate per quality rule, a shape check per contract. A per-guardrail overhead table against the 1M-row reference move is pending a measurement run:

> Benchmark table — throughput with each guardrail off vs. on — **[measurement TBD]**.

One honest tradeoff: because a governance stage operates on materialized rows, enabling masking (or any value-shaped stage) turns off the Arrow columnar fast path for that pipeline. Governed pipelines run the row loop by design — that is the same property that makes masking impossible to bypass.

9. Limitations & honest scope

What the current implementation does *not* claim:

- **The masking guarantee covers sinks, DLQ envelopes, and lineage events.** The source-side lineage sampler keeps a small raw, pre-masking sample in memory (used by the movement catalog); that surface is not masked. The end-of-pipe surfaces are.
- **Hashing is pseudonymization, not a secret** unless a key is supplied. With no key, it falls back to plain SHA-256 — deterministic and joinable, but recomputable. Cross-run joinability requires the same key in both runs.
- **Column lineage is a v1 subset.** It is derived for field-preserving and explicit-mapping transforms. A structural transform (flatten, key-case rewrite, SQL) makes the pipeline emit schema lineage but *no* column-lineage facet — it never fabricates an edge it cannot prove.
- **SLA metrics are emitted by the CLI/server.** A library embedder driving the engine directly, bypassing the CLI, does not get the SLA Prometheus counter.
- **Quarantine requires a DLQ.** Any `quarantine` action needs a dead-letter sink configured, or the run is rejected at startup.

10. Conclusion

Governance and movement are usually two systems with a gap between them. faucet-stream collapses that gap: the same loop that moves a page of data masks it, checks it, enforces its contract, and records its lineage before the write — with an ordering guarantee that is fixed in one place and proven in CI. It is not a complete replacement for a mature catalog or a warehouse-native test suite, and this paper is deliberate about where the edges are. But for the class of guarantees that matter most at ingest — that a secret never lands, that a bad row never reaches the target — doing it in the pipe is the difference between a policy and a hope.

Was this useful?

React with 👍, ❤️ or 🚀 to vote, or leave a comment. You sign in with GitHub inside the box below; this site never sees your account.

Loading comments…