TECHNICAL WHITE PAPER

A Zero-Cost Columnar Fast Path

faucet-stream's record is a JSON value, and that is the right default. But when both ends of a pipeline already speak Apache Arrow, converting to rows and back at every boundary is pure tax. This paper describes the opt-in Arrow fast path: a true zero-cost abstraction — inert by default, byte-identical builds — that, when negotiated, hands columnar batches straight from source to sink, and is honest about the one cost it does not remove.

faucet-stream · ~11 min read · Micro-benchmarked; end-to-end run pending

Online version: https://faucet-hq.github.io/papers/columnar-fast-path

---

Abstract

A general-purpose data mover needs one record model that every connector understands. faucet-stream chooses `serde_json::Value` — schema-flexible, easy for third-party connectors, and the substrate every governance stage operates on. The cost of that choice shows up only in one place: a pipeline whose source and sink are *both* columnar (Parquet → Parquet, an object store → Delta, a warehouse Arrow stream → BigQuery) pays to decode Arrow into rows and re-encode rows into Arrow at each hop — work that produces nothing. This paper describes how faucet-stream removes that tax with an opt-in Arrow path that is negotiated per run, bypasses the row loop entirely when it is safe to, and — crucially — costs *nothing* when it is off. It is also deliberate about the meaning of "zero-cost": the abstraction is free when disabled, and the fast path eliminates the value-conversion tax, but it does not make columnar encode itself disappear.

1. The problem: the value tax

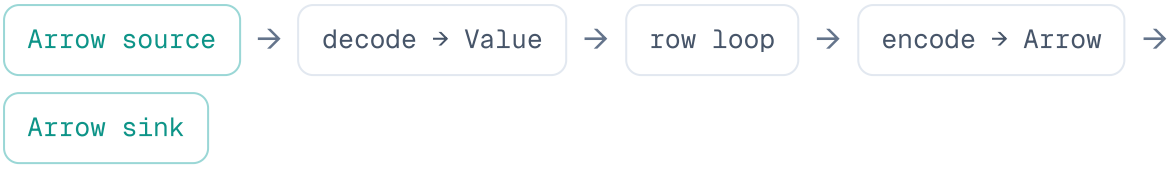
faucet-stream moves data in bounded pages. On the default path a page is a `StreamPage` — a `Vec<Value>` plus a resume bookmark — and every connector reads and writes rows. That uniformity is what lets one masking rule, one quality check, or one contract apply identically across dozens of sinks (see the companion paper, Governance in the Movement Path).

But some connectors are columnar to their core. A Parquet file is Arrow record batches on disk. BigQuery's Storage Read API streams Arrow. Delta and the object-store formats are columnar. When such a source feeds such a sink, the default path decodes each Arrow batch into a vector of JSON values and then immediately re-encodes those values back into an Arrow batch for the sink. The data ends where it began — columnar — having made a full round trip through a row representation for no semantic reason.

> We measured that round trip directly. On a six-column analytical page, a single **Arrow→Value→Arrow** conversion costs **~7–9× the entire Parquet decode+encode** it wraps — and it is paid at *every* boundary in the pipeline. The tax is larger than the useful work.

2. The thesis: an opt-in path that is free when off

The fix is not to replace the value model — that would break every connector and every guardrail. It is to add a second, parallel page type and let a pipeline take it only when both ends can. The Arrow support is a Cargo feature that is **inert by default**: with it off, the compiled binary is byte-identical to the value-only pipeline. Nothing about the default path changes; there is no runtime branch to pay for, no dependency pulled in. This is the first and most literal sense of "zero-cost" — a zero-cost abstraction in the C++ sense: you do not pay for what you do not use.



default path — correct everywhere, taxed at each boundary



fast path — negotiated when both ends speak Arrow and no stage needs rows

Figure 1 — the fast path carries an Arrow `RecordBatch` from source to sink with no value round trip. The same page bookmark drives crash-safe checkpointing on both paths.

3. Two page types, one contract

The row path carries a `StreamPage` (records as values). The fast path carries a `ColumnarPage` — an Arrow `RecordBatch` plus the *same* resume bookmark. The bookmark is the important part: it means the columnar path inherits the identical checkpoint semantics as the row path (write → flush → persist bookmark), so switching paths never weakens the crash-safety story.

A connector opts in through two trait methods, both compiled out when the feature is off:

`supports_columnar() → bool`

A capability flag, defaulting to `false`. A connector returns `true` only when it can produce or consume Arrow natively — and it may decide at runtime (e.g. an object-store connector says yes only when the file format is Parquet).

`stream_batches()` / `write_batch_columnar()`

The native Arrow implementations. The default `write_batch_columnar` is a safety net: it converts the batch to values and calls the row writer, so a half-migrated connector is still correct — just not fast.

4. Negotiation: a pure escape hatch

The fast path is never assumed; it is negotiated once, when the pipeline starts. It is taken only if every one of the following holds — otherwise the pipeline runs the ordinary row loop, unchanged:

GATE	WHY IT IS REQUIRED
Source speaks Arrow	the source overrides supports_columnar() → true
Sink speaks Arrow	the sink overrides supports_columnar() → true
No masking / quality / contract	a value-shaped guardrail must see rows
No schema-drift / adaptive / cleanup stage	these inspect record values
No dead-letter queue	the DLQ envelopes individual rows
At-least-once delivery	the exactly-once gate keys on row identity
Sink is not in overwrite mode	overwrite reconciles at the row level

The pattern is deliberate: the fast path is an optimization that must be *invisible*. If anything in the pipeline needs to see a row — a masking rule, a quality check, a dead-letter envelope, the exactly-once identity gate — the columnar shortcut is simply not taken, and correctness is preserved by construction rather than by a connector remembering to check. A negotiation test drives a columnar-only source and sink whose value methods deliberately error: the run succeeds only if the fast path is genuinely taken, and falls back cleanly when the sink cannot speak Arrow.

5. What is saved — and what still costs

On a true native chain the fast path eliminates the value round trip at every boundary. The measured tax it removes, per page:

ROWS / PAGE	ARROW→VALUE→ARROW (TAX REMOVED)	PARQUET DECODE+ENCODE (REAL WORK)	TAX ÷ WORK
1,000	698 µs	101 µs	~6.9×
10,000	7.01 ms	0.93 ms	~7.5×
50,000	41.7 ms	4.78 ms	~8.7×

The honest counterpoint: **"zero-cost" is about the abstraction, not the I/O.** The fast path removes the value tax; it does not remove the cost of columnar work itself. A Parquet or Snowflake sink still encodes each Arrow batch to Parquet and uploads it — that is irreducible. The conversion shim, when it *is* used, is not zero-copy either: it moves through an Arrow-JSON encoder/decoder. What the fast path guarantees is that on a negotiated native chain, none of that shim runs at all.

6. Which connectors speak Arrow

The fast path lights up when a native source meets a native sink:

ARROW-NATIVE SOURCES	ARROW-NATIVE SINKS
Parquet	Parquet
S3 / object store (Parquet)	S3 / object store (Parquet)
GCS (Parquet)	GCS (Parquet)
Delta Lake	Delta Lake
Databricks (Arrow stream)	BigQuery (Storage Write)
BigQuery (Storage Read)	Snowflake (bulk load)

The SQL transform is the one stage that participates columnar-side: a `parquet → sql → parquet` pipeline runs entirely in Arrow, because the SQL engine (DuckDB) exchanges record batches directly. Every other transform observes values, and so returns the pipeline to the row path.

7. Why governance turns it off — on purpose

The negotiation gates in §4 are not arbitrary. A value-shaped guardrail — masking, quality, contracts, drift — must operate on materialized rows, so enabling any of them disables the columnar shortcut for that pipeline. This is the same property, seen from the other side, that makes masking impossible to bypass: a governed pipeline runs the row loop by design. Speed and governance are not in tension here;

the engine simply picks the row path whenever a guardrail needs to see the data, and the fast path whenever nothing does.

8. Limitations & honest scope

- **"Zero-cost" means the abstraction, not the encode.** The default path is byte-identical with the feature off, and the fast path removes the value tax — but Arrow→Parquet encoding and upload remain, and are the real cost on a columnar sink.
- **The conversion shim is not zero-copy.** When a page does cross the row boundary, it goes through an Arrow-JSON encoder/decoder — allocating, not aliasing. The win is in *not* crossing, not in crossing cheaply.
- **The published numbers are micro-benchmarks.** They measure the conversion tax and the Parquet round trip in isolation. An end-to-end throughput figure for a live columnar pipeline — **[measurement TBD]** — is not yet published.
- **Only the SQL transform is columnar.** Any other transform, or any governance stage, returns the pipeline to the row path — correct, but taxed.
- **Not every columnar connector is on the path.** Some connectors that link Arrow (e.g. the Iceberg sink) do not yet override the capability flag, and message/DB/CDC connectors are row-native by nature.

9. Conclusion

A single record model keeps a connector ecosystem sane and governance uniform; a columnar fast path keeps the analytical, warehouse-to-lake pipelines fast. faucet-stream refuses to choose between them. The value model stays the default and the substrate for every guardrail; the Arrow path is an opt-in that costs nothing when off and, when both ends speak Arrow and no stage needs a row, carries batches straight through — no tax, no bypass of the checkpoint contract, and no way to accidentally skip governance. "Zero-cost" is a precise claim here, not a slogan: free when disabled, tax-free on the fast path, and honest about the encode it cannot make vanish.

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…