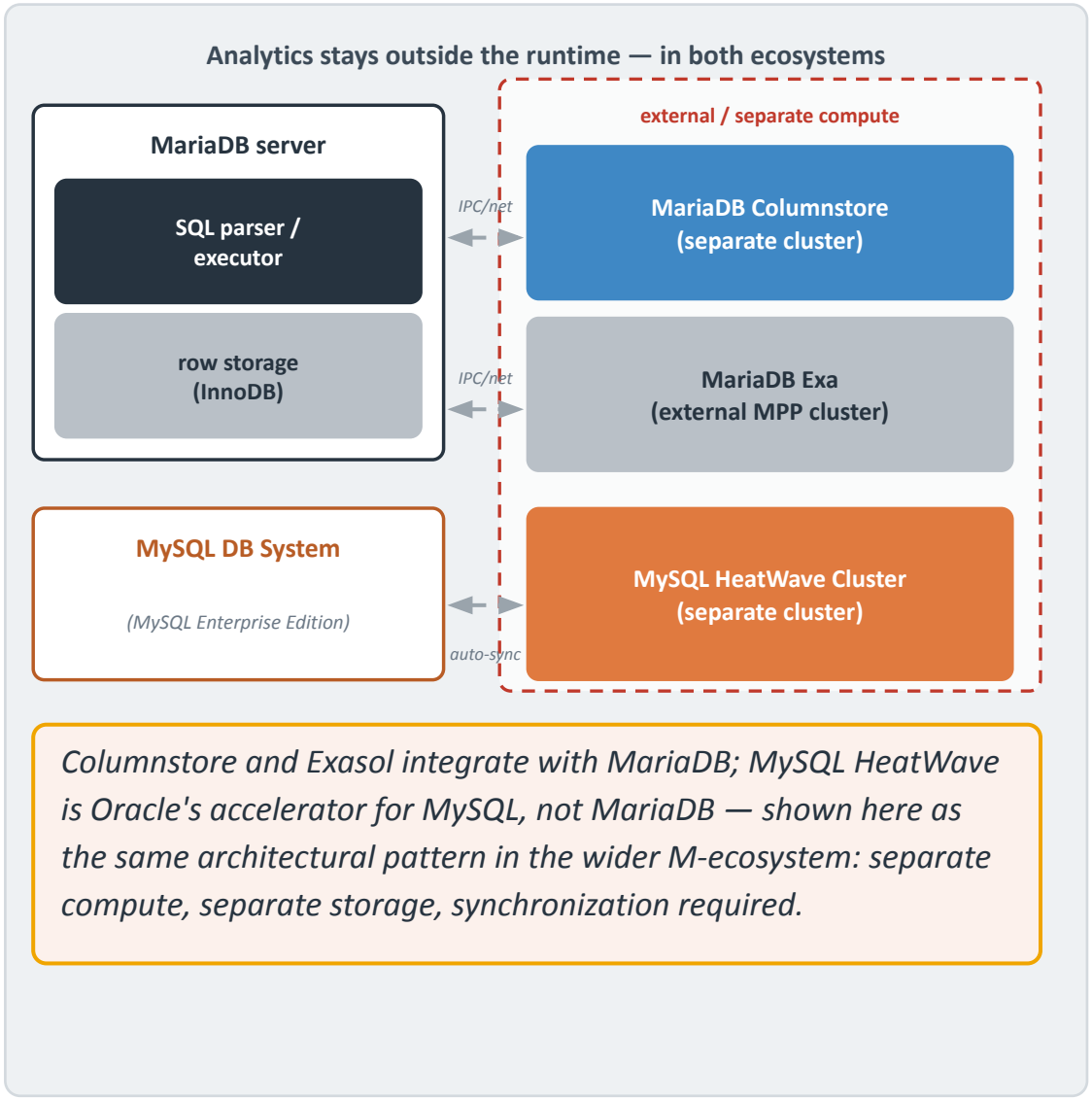


When Sea Lion Learns to Quack: DuckDB as a MariaDB storage engine

Roman Nozdrin
MariaDB

2026


MariaDB®
The Pond



The Pond
OLAP in the M-Ecosystem Today

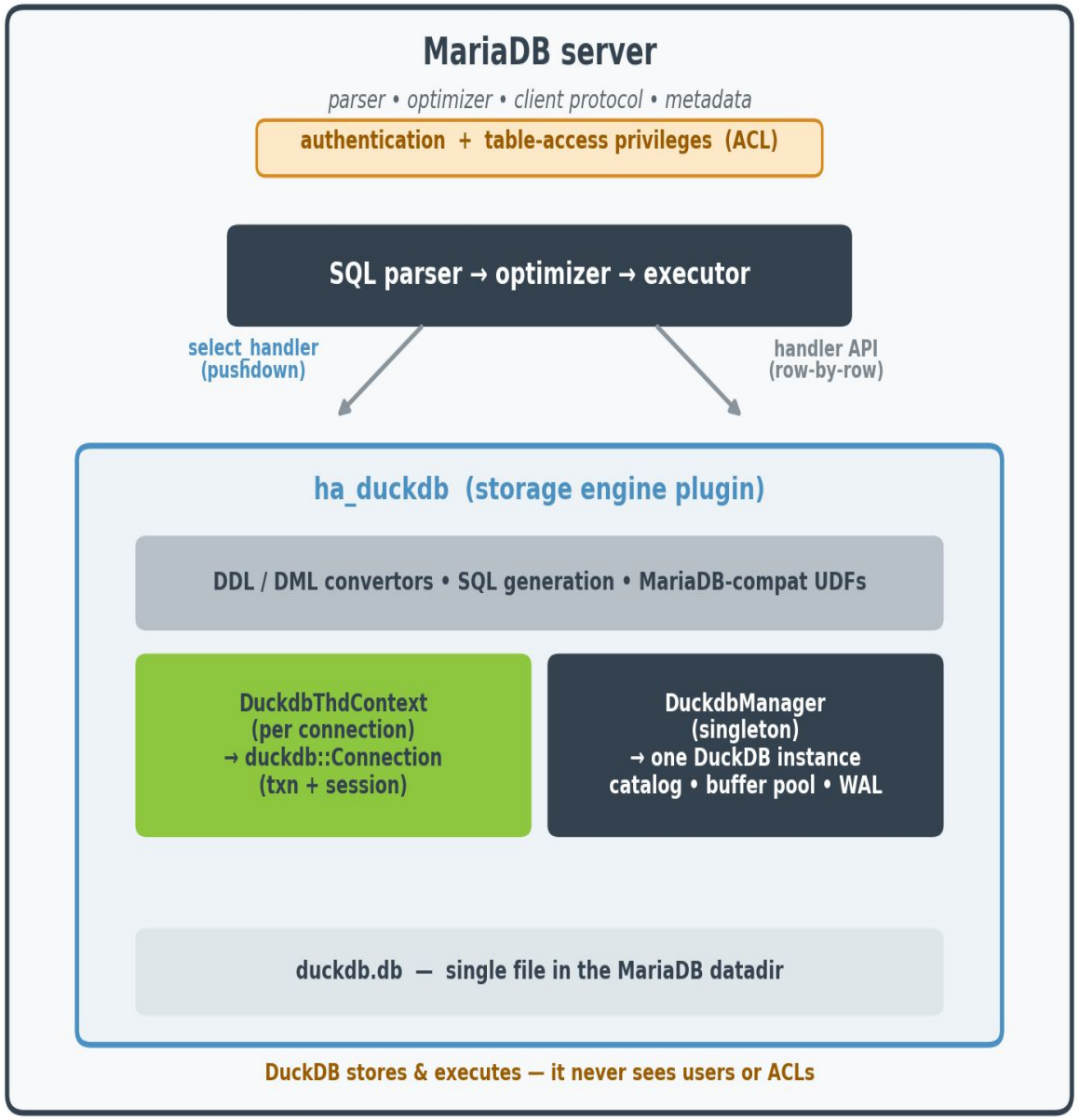
- MariaDB Columnstore: a columnar engine for analytics runs a separate MPP cluster.
- MariaDB Exa: a separate high-performance MPP database cluster.
- MySQL HeatWave: needs a separate cluster of nodes.
- All three run outside the core database runtime: a separate process boundary, network hops, data to keep in sync.

MySQL HeatWave facts: MySQL HeatWave User Guide, dev.mysql.com/doc/heatwave

MariaDB



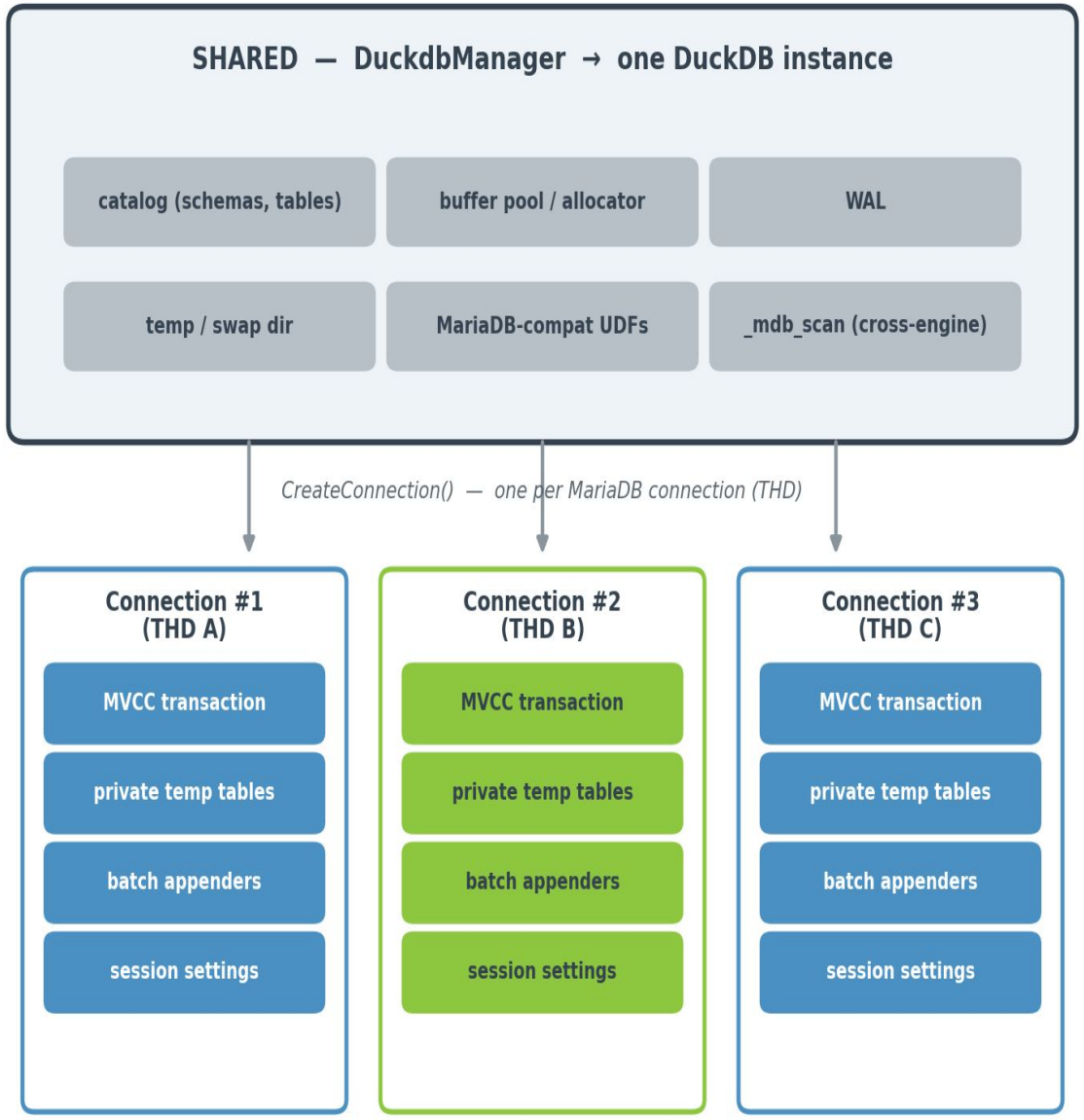
The Duck



DuckDB
Smart Storage Engine

- ha_duckdb is a storage-engine plugin: MariaDB keeps parsing, client protocol and metadata.
- DuckDB owns columnar storage, execution and MVCC - one shared instance (catalog, buffer pool, WAL).
- User authentication and table-access privileges stay in MariaDB - DuckDB never sees ACLs; per connection a DuckdbThdContext wraps a duckdb::Connection.

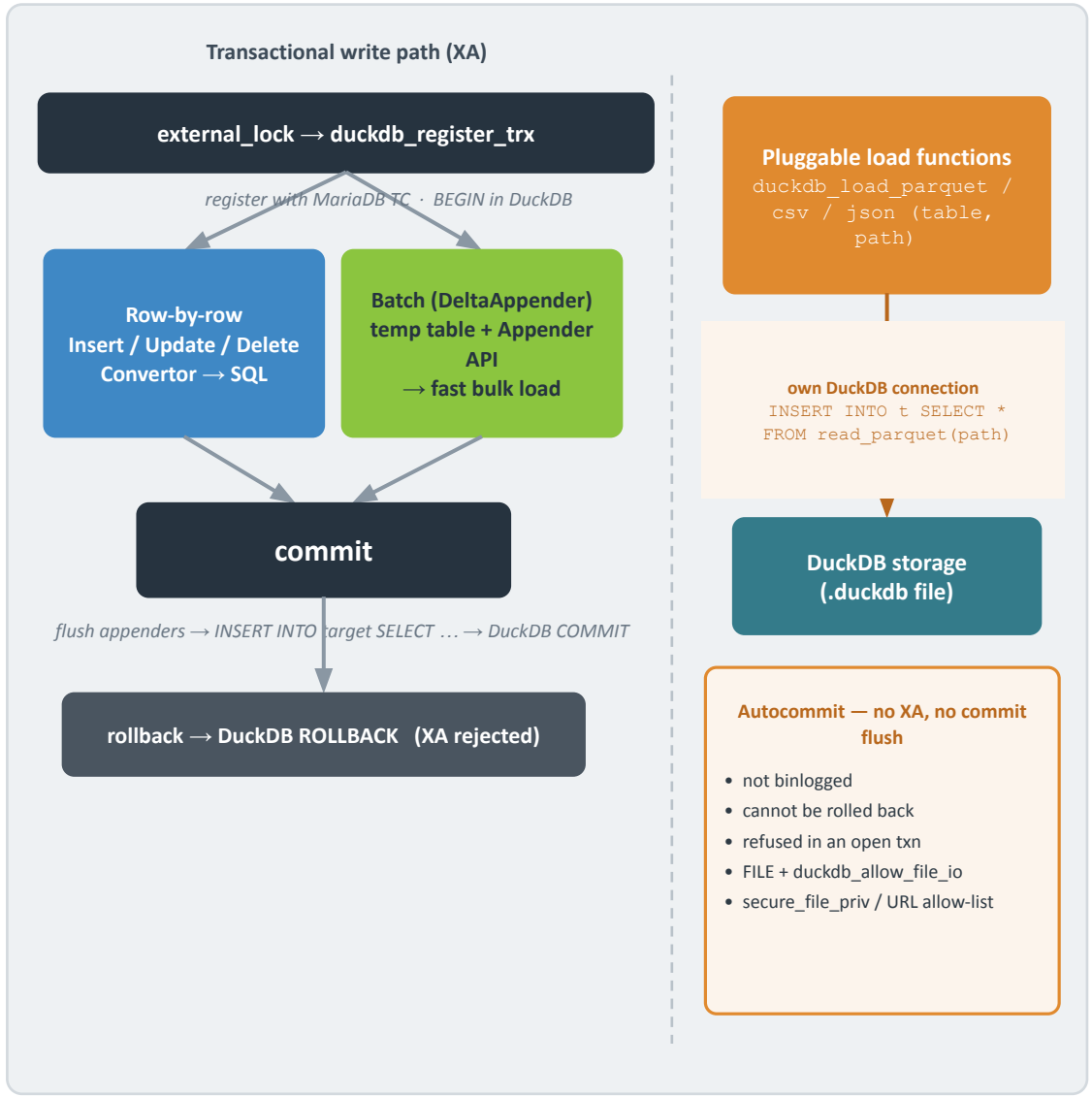
MariaDB



Shared Resources Across Connections

- Shared, one per server: the DuckDB instance - catalog, buffer pool, WAL, temp/swap, plus the compat UDFs and macros.
- Per connection: its own duckdb::Connection - an MVCC transaction, private temporary tables, batch appenders and session settings.
- Trade-off: reads are snapshot-isolated and writes serialized by DuckDB's single-writer lock - but the shared buffer pool means one big scan can evict another's pages.

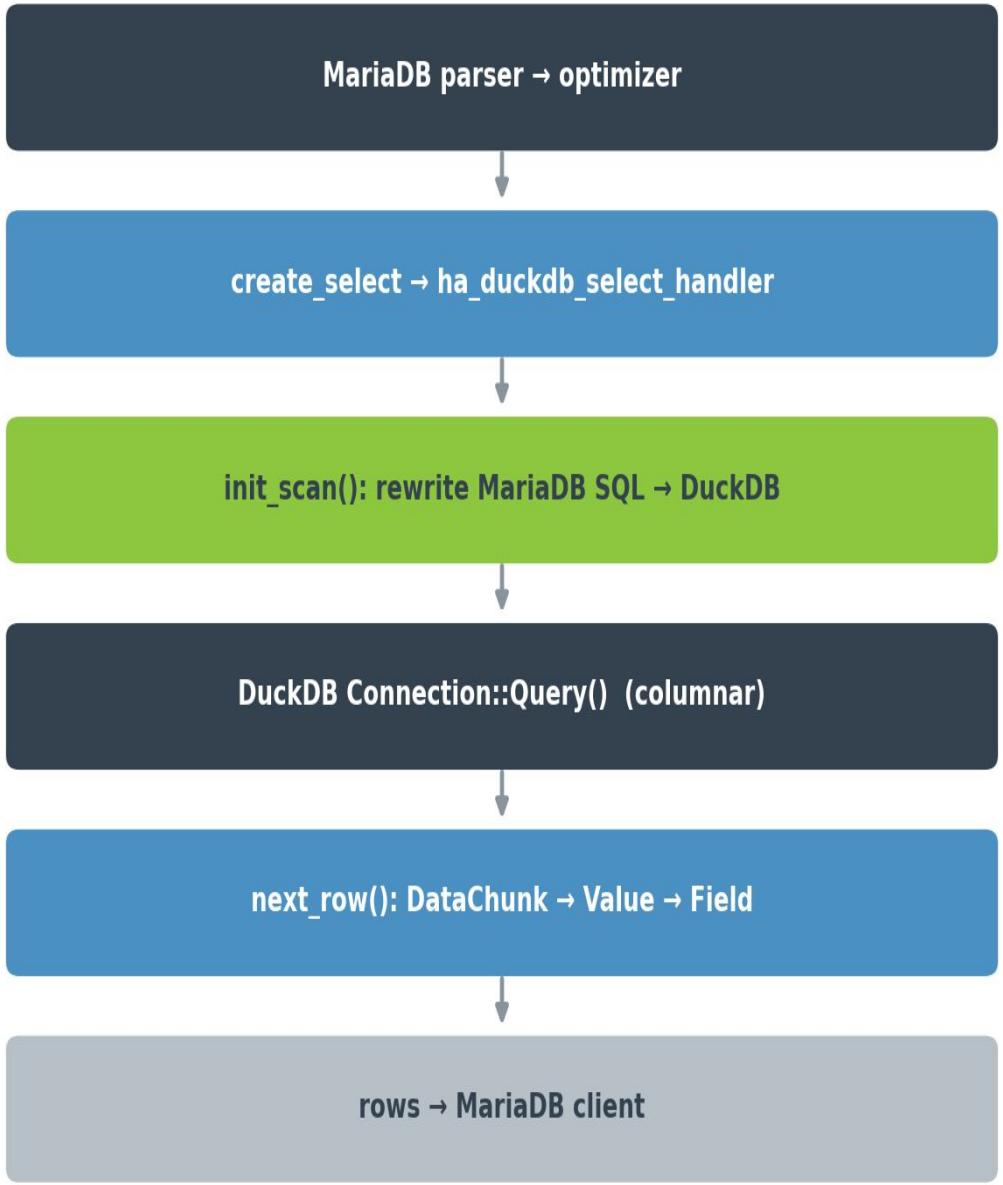
MariaDB



Write Path & Batch DML

- DDL maps through convertors (CREATE / ALTER / DROP); ALTER supports both MariaDB INSTANT and COPY protocols.
- Each DML statement registers with MariaDB's transaction coordinator; a matching BEGIN / COMMIT runs in DuckDB. Multi-engine transactions are supported.
- Small writes go row-by-row via DML convertors; bulk loads use the DeltaAppender — a temp table + Appender API, flushed as one INSERT … SELECT at commit.
- Third path: pluggable load functions duckdb_load_parquet / csv / json(table, path) run INSERT INTO t SELECT * FROM read_parquet(path) on their own DuckDB connection — straight into the .duckdb file, no DeltaAppender and no XA commit.

MariaDB



Happy Read Path: SELECT Handler Pushdown

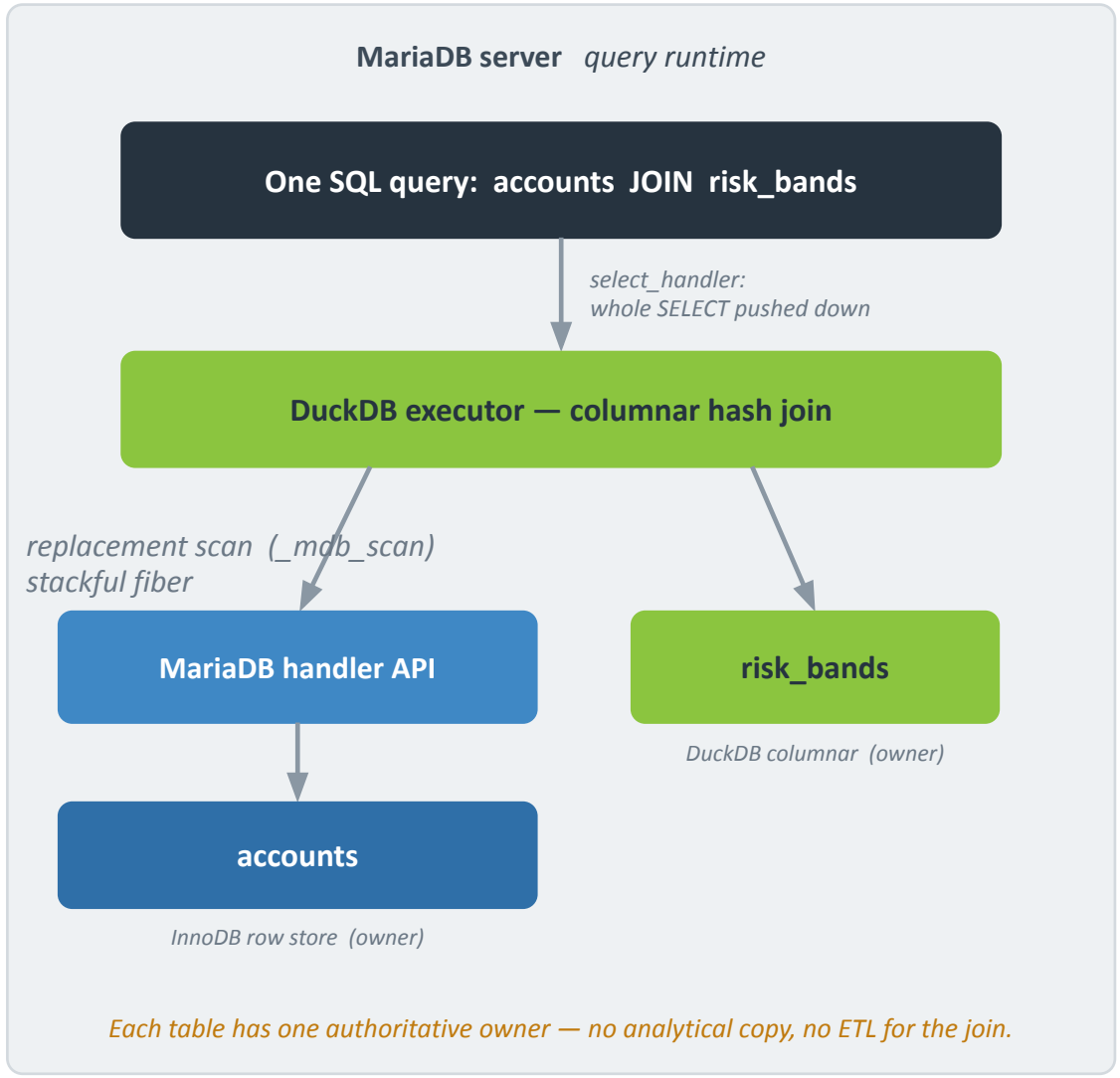
For analytics, MariaDB pushes the whole SELECT to DuckDB via select_handler.

init_scan() rewrites MariaDB syntax to DuckDB: WITH ROLLUP -> ROLLUP(), CONVERT -> CAST, REGEXP -> ~, LIMIT a,b -> LIMIT b OFFSET a; hints stripped.

Results stream back DataChunk-by-DataChunk (Value -> Field).

Cross-engine joins reach InnoDB/ARIA via a replacement scan + stackful fiber.

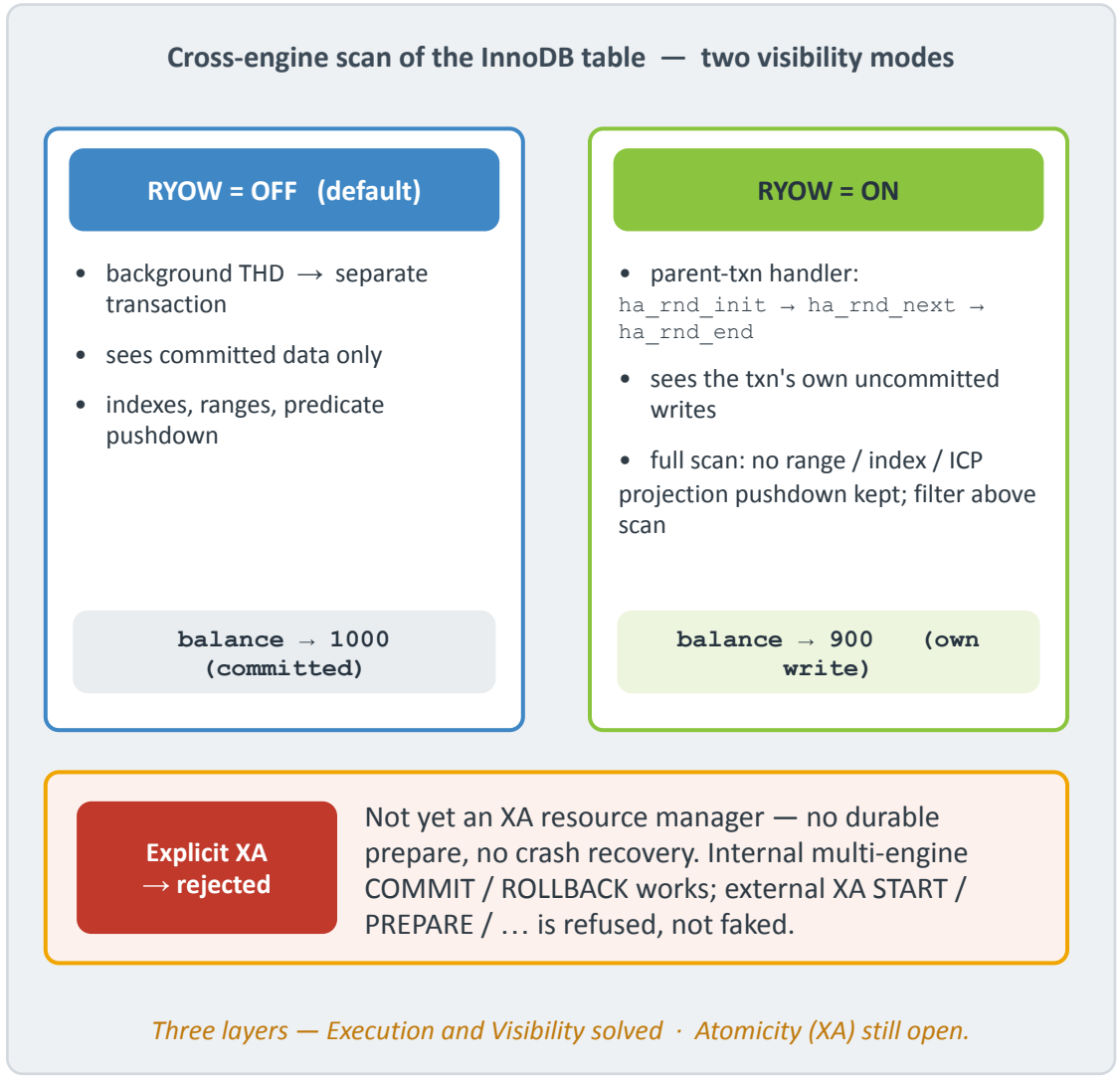
MariaDB



Cross-Engine Join Single-Copy Federation

- Single-copy federated execution: transactional tables stay in InnoDB, analytical tables stay in DuckDB one SQL query joins straight across the boundary.
- No second columnar copy and no ETL / CDC pipeline for the join. Each table has exactly one authoritative owner.
- DuckDB drives the query and reaches the InnoDB table through MariaDB's handler API via a replacement scan (_mdb_scan) on a stackful fiber; select_handler pushes the whole SELECT down.
- Federation, not dual storage: no table is ever held in row and column format at once.

MariaDB



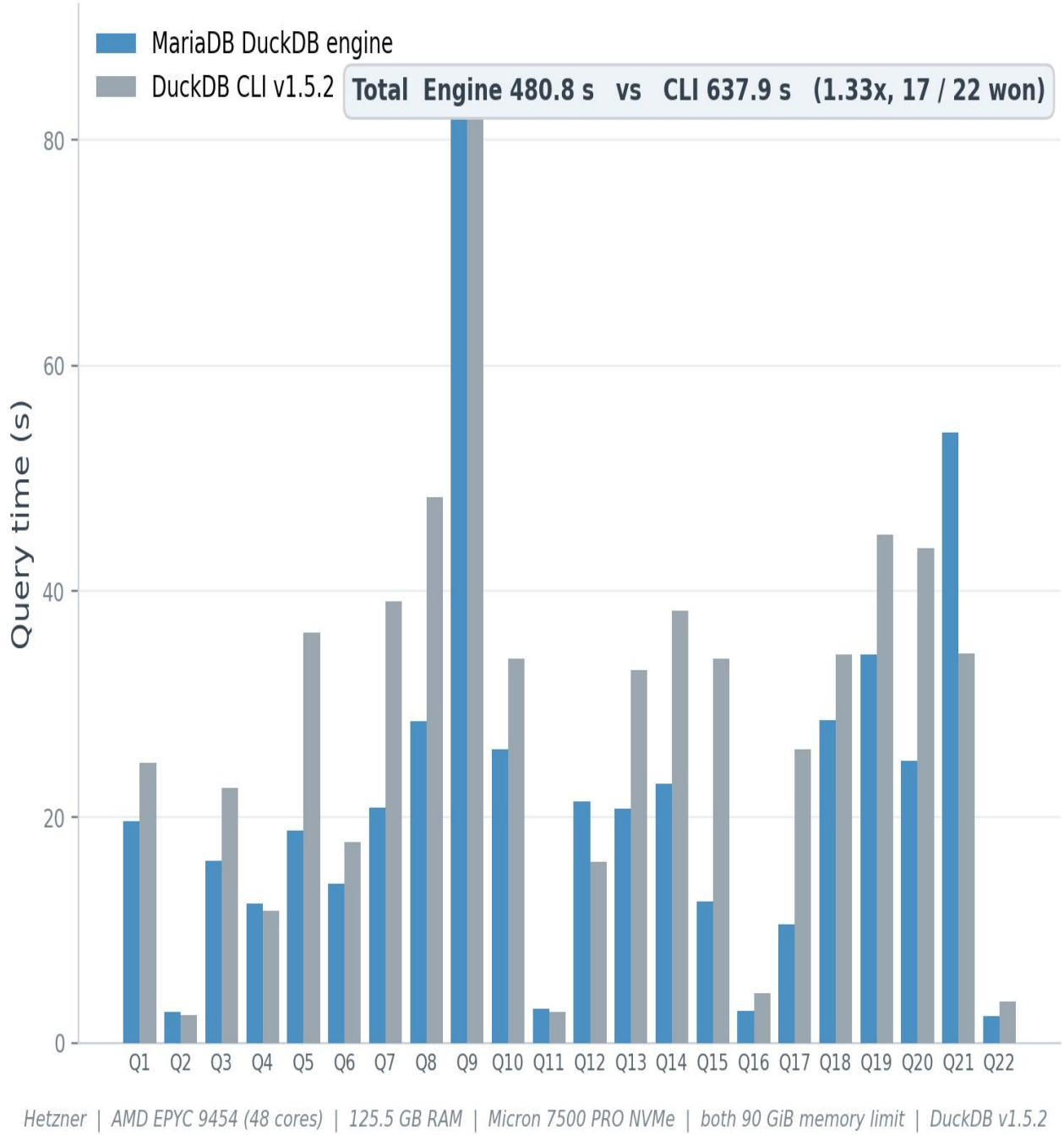
Read-Your-Own-Writes …and the XA Boundary

- Same query, two answers: with RYOW off the cross-engine scan runs in a background transaction and sees only committed rows (1000); with RYOW on, DuckDB scans through the parent transaction's handler and sees its own uncommitted write (900).
- Cost of correctness: the RYOW path is currently a full handler scan no range or index access, no index-condition pushdown. Projection pushdown stays and DuckDB keeps the filter above the scan.
- RYOW is not XA: visibility is not atomic commit.

MariaDB



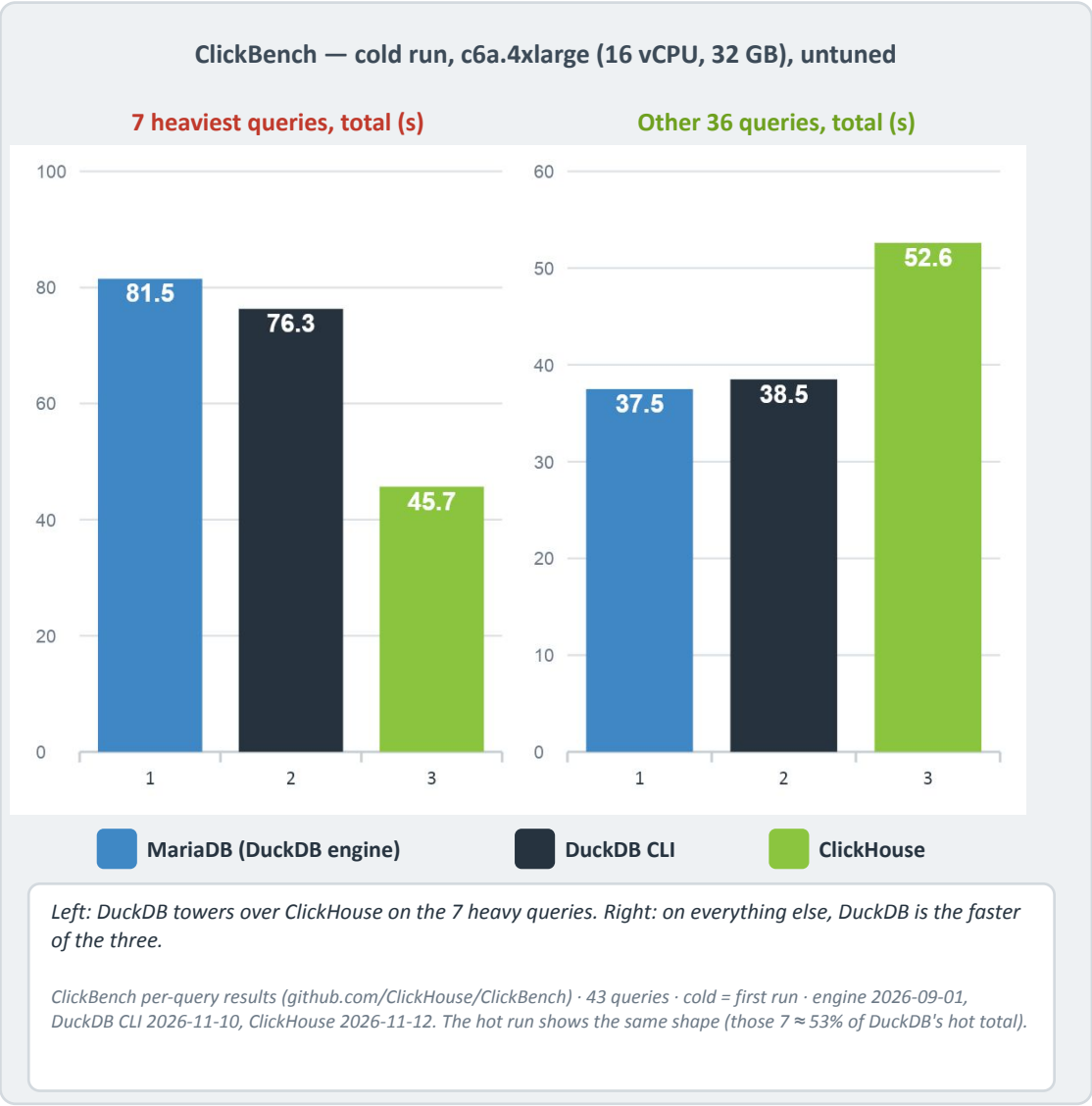
The First Sea Lion's Quack in Numbers



TPC-H SF1000: Engine vs DuckDB 1.5.2 CLI

- On TPC-H SF1000 (about 8.7 B rows, 400 GB Parquet), the engine totals 480.8 s vs 637.9 s for the DuckDB CLI v1.5.2: 1.33x faster, winning 17 of 22. Best case Q15 at 2.72x.
- Why: the engine is a persistent in-process DuckDB instance inside mariadbd with a warm buffer pool. The CLI spawns a fresh process per query, cold-starting the buffer manager every time. That cold start dominates, not raw execution.
- Honest read: single run, no repetition; a single-session CLI run would narrow the gap. Peak RSS is similar (112 vs 115 GB). One regression is Q21 (EXISTS patterns), where the CLI wins.

MariaDB



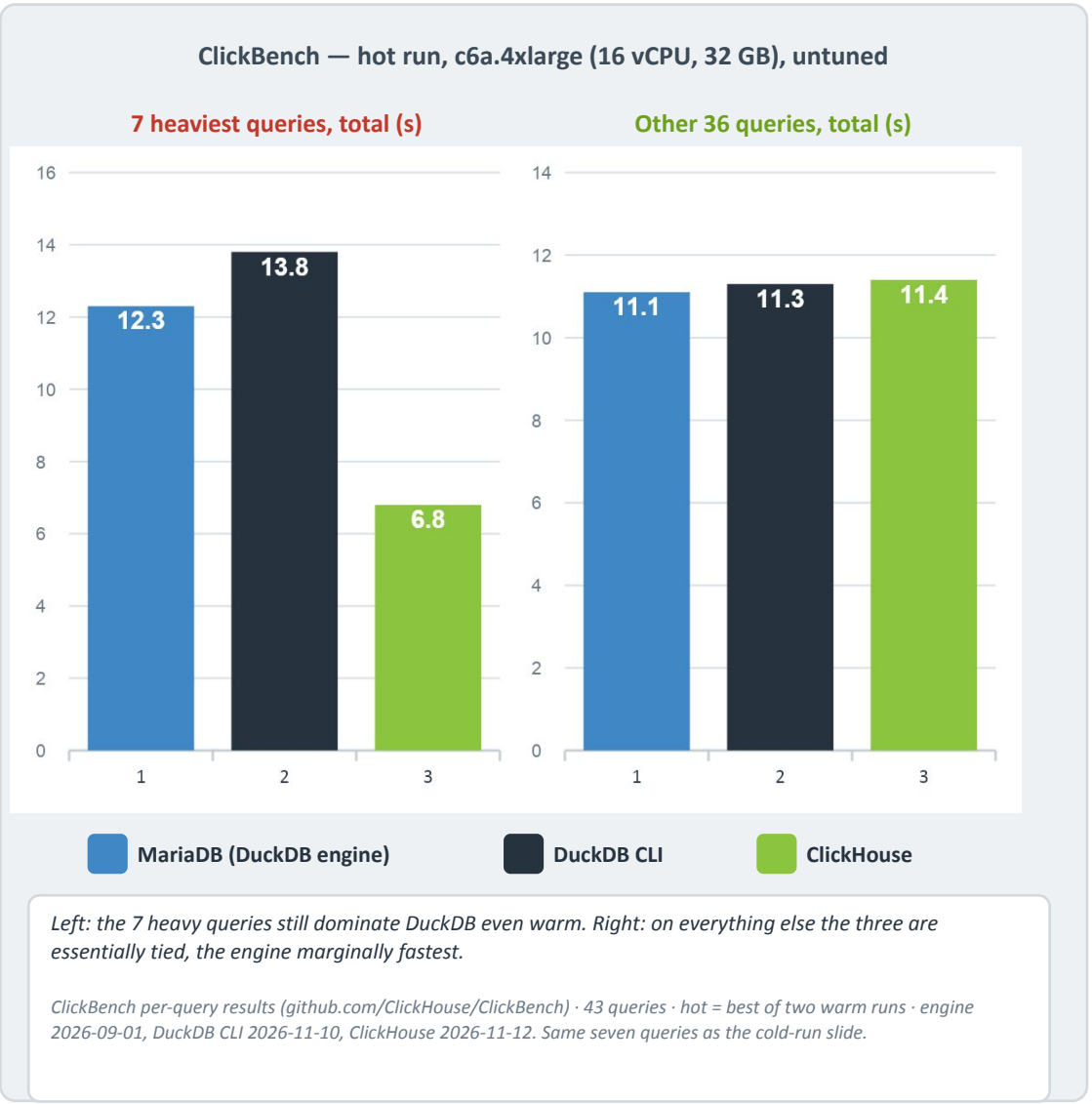
Left: DuckDB towers over ClickHouse on the 7 heavy queries. Right: on everything else, DuckDB is the faster of the three.

ClickBench per-query results (github.com/ClickHouse/ClickBench) · 43 queries · cold = first run · engine 2026-09-01, DuckDB CLI 2026-11-10, ClickHouse 2026-11-12. The hot run shows the same shape (those 7 ≈ 53% of DuckDB's hot total).

c6a.4xlarge: Seven Queries Set the Total

- Seven queries (Q20–22, Q27, Q28, Q33, Q34) are 68% of the DuckDB engine's cold total — 81.5 s of 119.0 s — but only 46% of ClickHouse's. They are wide string scans (LIKE / regex over URLs) and high-cardinality GROUP BYs.
- On exactly those seven, ClickHouse's columnar engine pulls far ahead — DuckDB spends about 1.8× its time (Q27 alone: 10.9 s vs 0.8 s).
- Remove them and the order flips: on the other 36 queries the engine runs 37.5 s vs ClickHouse 52.6 s, winning 24 of 36.

MariaDB



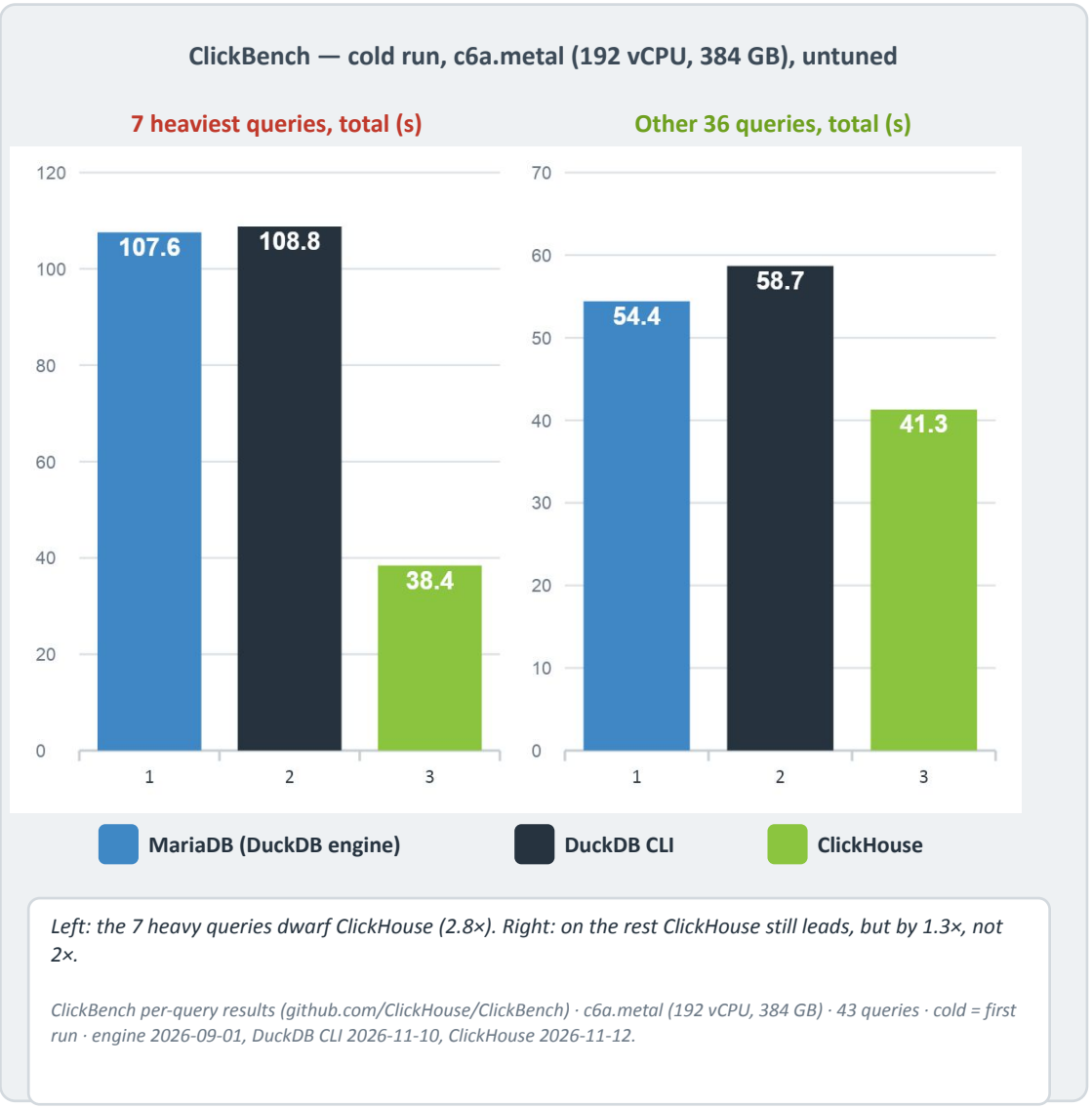
Left: the 7 heavy queries still dominate DuckDB even warm. Right: on everything else the three are essentially tied, the engine marginally fastest.

ClickBench per-query results (github.com/ClickHouse/ClickBench) · 43 queries · hot = best of two warm runs · engine 2026-09-01, DuckDB CLI 2026-11-10, ClickHouse 2026-11-12. Same seven queries as the cold-run slide.

c6a.4xlarge: Hot Run, the Same Seven

- Warm caches shrink every query, yet the same seven (Q20–22, Q27, Q28, Q33, Q34) are still 53% of the DuckDB engine's hot total — 12.3 s of 23.4 s — versus only 37% of ClickHouse's.
- Q28 alone — a regex REGEXP_REPLACE GROUP BY over Referer — is 6.6 s, 28% of the engine's hot total, against ClickHouse's 1.6 s.
- Strip the seven and the three converge: on the other 36 queries the engine runs 11.1 s, DuckDB CLI 11.3 s, ClickHouse 11.4 s — the engine a hair ahead, winning 21 of 36. Hot or cold, a few heavy queries set the total.

MariaDB



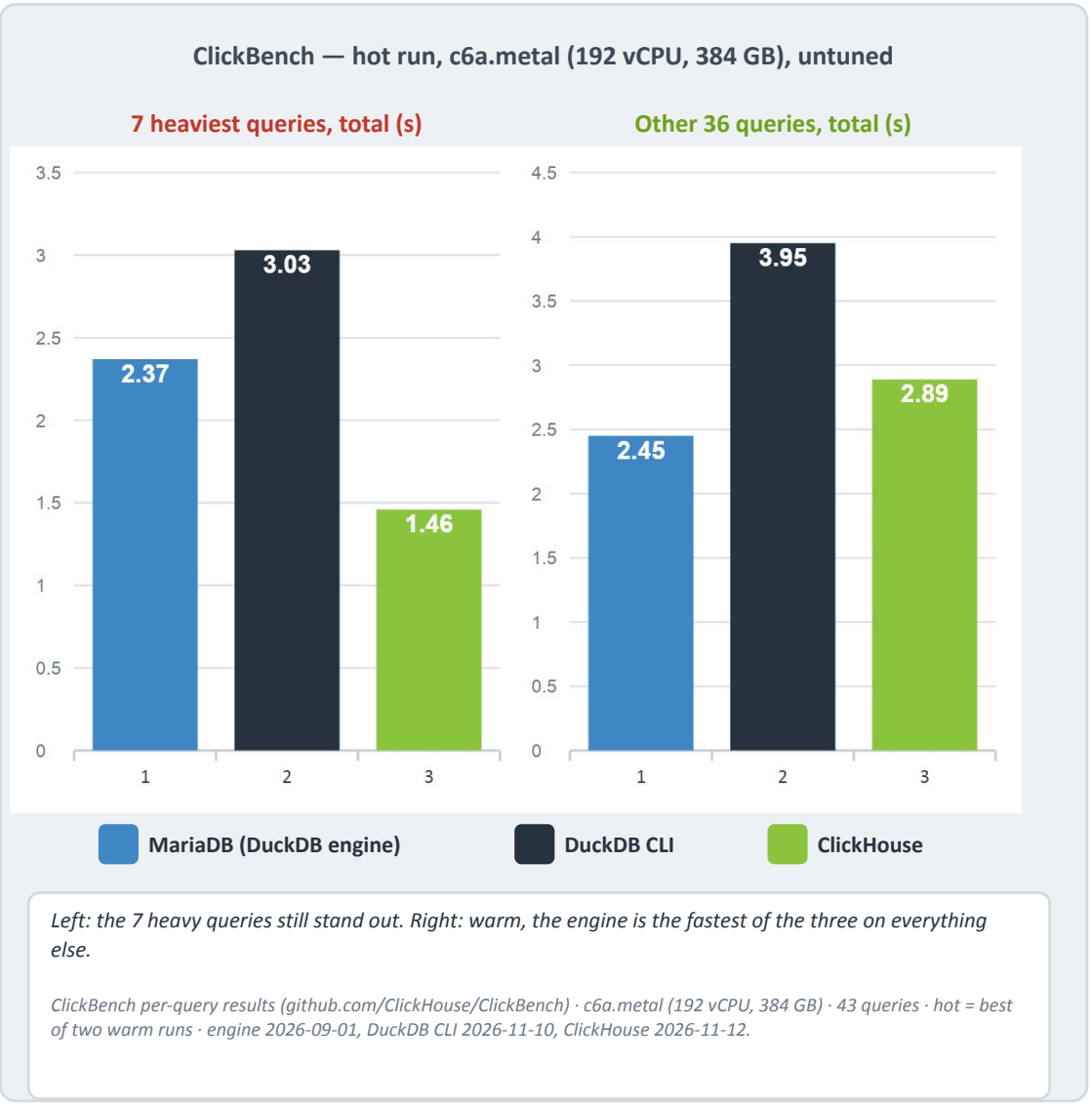
Left: the 7 heavy queries dwarf ClickHouse (2.8×). Right: on the rest ClickHouse still leads, but by 1.3×, not 2×.

ClickBench per-query results (github.com/ClickHouse/ClickBench) · c6a.metal (192 vCPU, 384 GB) · 43 queries · cold = first run · engine 2026-09-01, DuckDB CLI 2026-11-10, ClickHouse 2026-11-12.

c6a.metal: Seven Queries Set the Total

- On c6a.metal (192 vCPU, 384 GB) the extra parallelism favours ClickHouse for smaller data sets as ClickBench: its cold total is ~2× DuckDB's. The same seven queries (Q20–22, Q27, Q28, Q33, Q34) are 66% of the DuckDB engine's cold total — 107.6 s of 162.0 s — but only 48% of ClickHouse's.

MariaDB



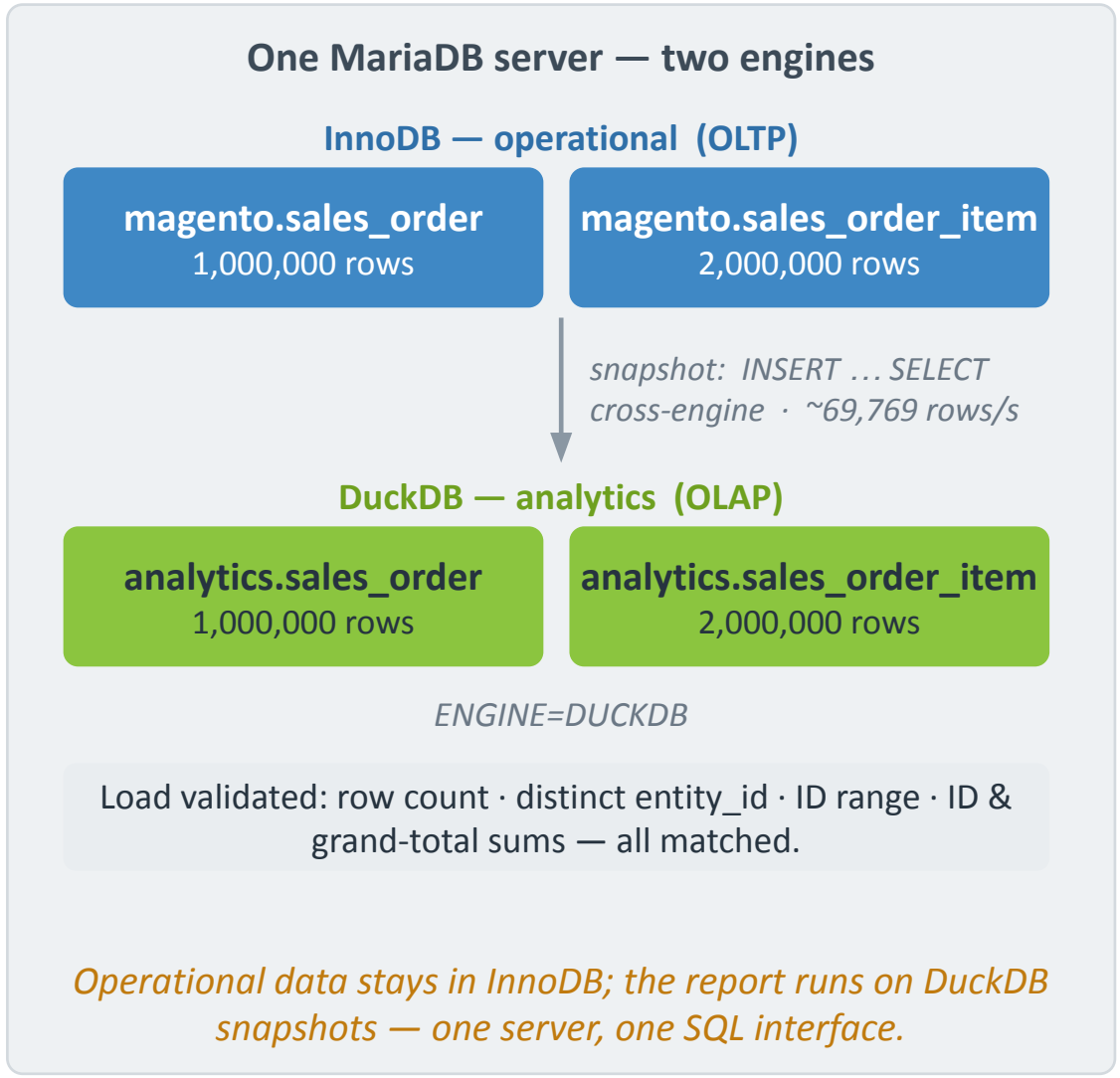
Left: the 7 heavy queries still stand out. Right: warm, the engine is the fastest of the three on everything else.

ClickBench per-query results (github.com/ClickHouse/ClickBench) · c6a.metal (192 vCPU, 384 GB) · 43 queries · hot = best of two warm runs · engine 2026-09-01, DuckDB CLI 2026-11-10, ClickHouse 2026-11-12.

c6a.metal: Hot Run, the Same Seven

- Warm on 192 cores the totals are tiny and tight: engine 4.82 s, ClickHouse 4.35 s, DuckDB CLI 6.98 s — the engine is within 11% of ClickHouse and clearly ahead of the standalone CLI.
- The same seven queries (Q20–22, Q27, Q28, Q33, Q34) are still the outliers — 49% of the engine's hot total (2.37 s of 4.82 s) vs 34% of ClickHouse's. Q28 alone (regex GROUP BY over Referer) is 22%: 1.08 s vs 0.30 s.
- Strip the seven and the engine is actually the fastest of the three: 2.45 s vs ClickHouse 2.89 s on the other 36, winning 27 of them. Warm, the headline is set almost entirely by a handful of heavy aggregations.

MariaDB



Magento on DuckDB Single-Copy Analytics

- Magento's operational tables stay in InnoDB; a structural clone in the analytics schema drops keys, indexes and FKs, then switches to ENGINE=DUCKDB.
- Snapshots load fast 1M orders in 14.3 s, 2M items in 20.2 s (~69,769 rows/s) via one cross-engine INSERT … SELECT, no external ETL.

Scan: the schema and full Orders Report query

MariaDB

Results
63.2× on the Report

InnoDB orders + InnoDB items	18.464 s	1.0×
DuckDB orders + InnoDB items	8.267 s	2.23×
DuckDB orders + DuckDB items	0.292 s	63.2×

COUNT(*) over 1M orders: ~121 ms InnoDB → 0.81 ms DuckDB (warmed) — metadata-answered, not representative of analytics.

Both queries · 1M orders × 2M items · identical results across all three configurations.

- Moving only sales_order to DuckDB cuts the report from 18.464 s to 8.267 s (2.23×) but the InnoDB scan and aggregation of sales_order_item stays the bottleneck.
- Moving both tables to DuckDB drops the combined runtime to 0.292 s a measured 63.2× for this dataset and query shape, with identical results.
- Honest read: explicit post-generation snapshots (not continuous sync); timings include client-invocation overhead.

MariaDB



So how does Sea Lion quack after the first lesson…?

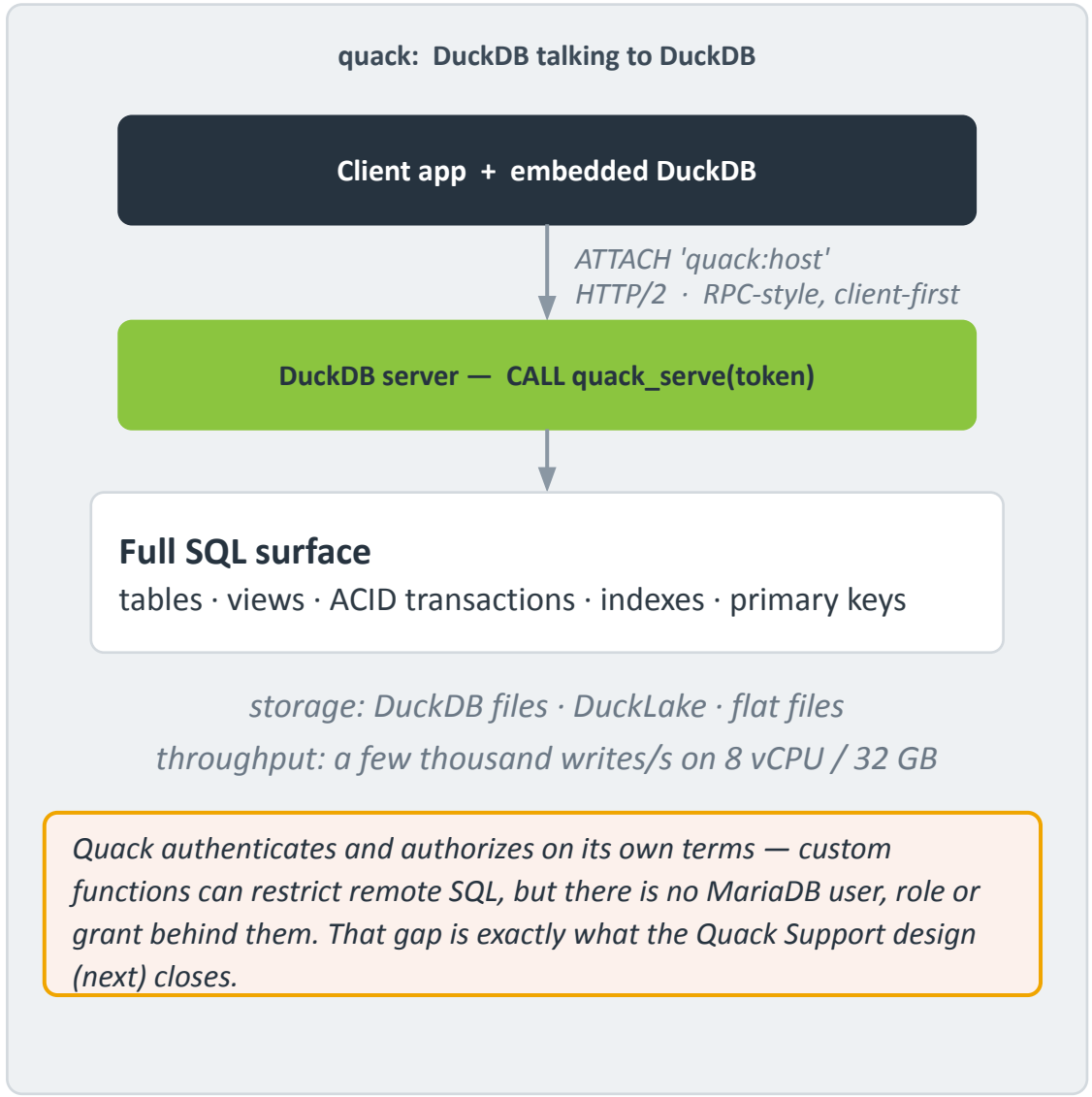
So how does Sea Lion quack after the first lesson…?



MariaDB



The Future of Sea Lion's Quacking

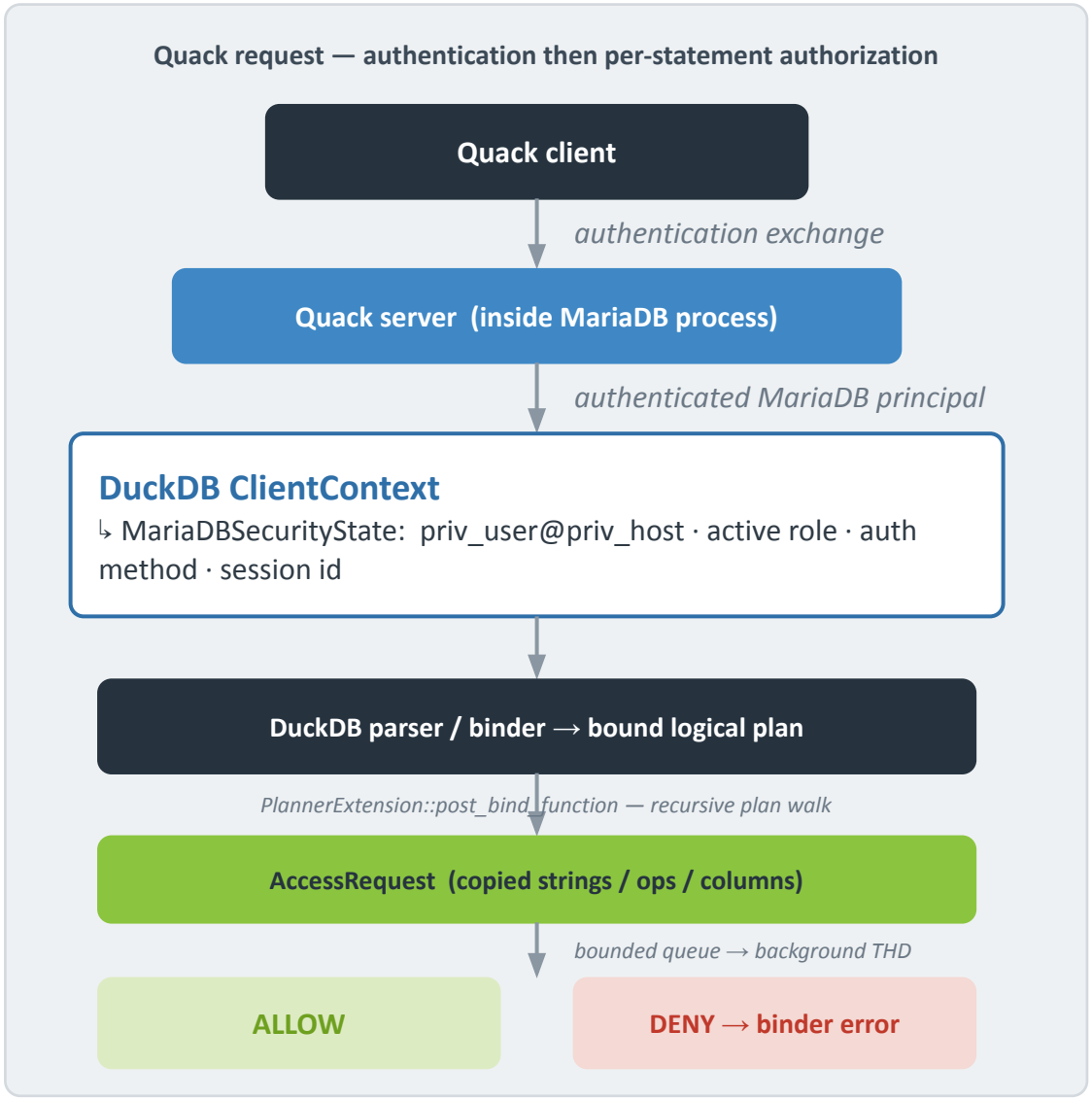


What Is Quack? DuckDB's Client/Server Protocol

- Quack turns DuckDB into a client-server database: a client runs ATTACH 'quack:host' and issues full SQL tables, views, ACID transactions, indexes and primary keys against a remote DuckDB server.
- Both ends are DuckDB effectively RPC between two DuckDB instances over HTTP/2; encryption is added via a reverse proxy, not by the protocol itself.
- Stability in DuckDB v2.0.

Quack docs
duckdb.org/docs
→ quack

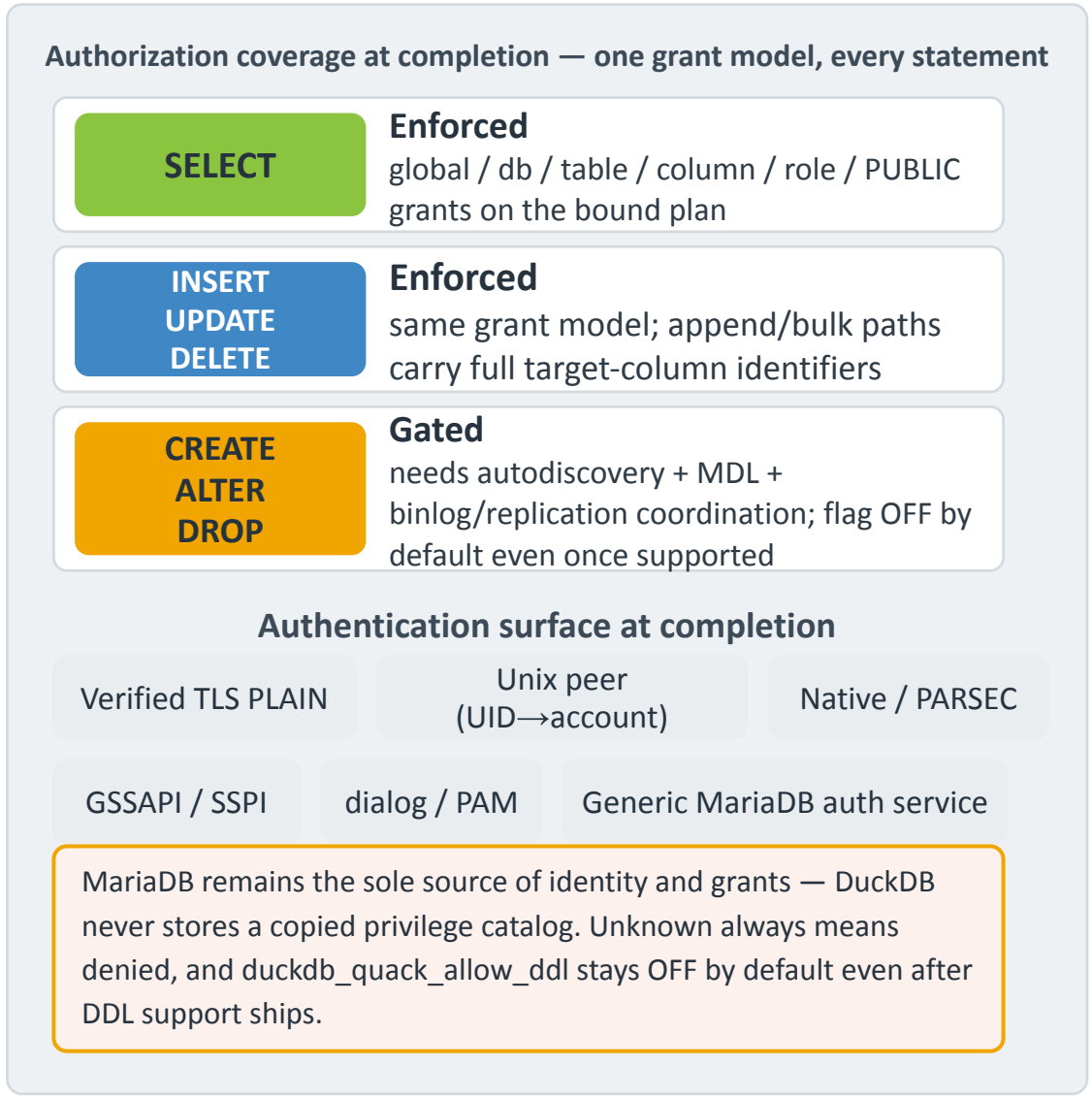
MariaDB



Quack Support MariaDB Stays the Security Boundary

- A Quack client talks to DuckDB directly, so no MariaDB THD, no grant check.
- SQL-text prefix matching can't safely cover CTEs, views, subqueries or table functions.
- Auth runs after DuckDB binding: the bound logical plan already has resolved catalogs, tables and columns and checks run on real objects, not SQL text.
- MariaDB ACL code runs only on a dedicated background THD.
- Unknown operator, table function or statement class are denied by default.

MariaDB



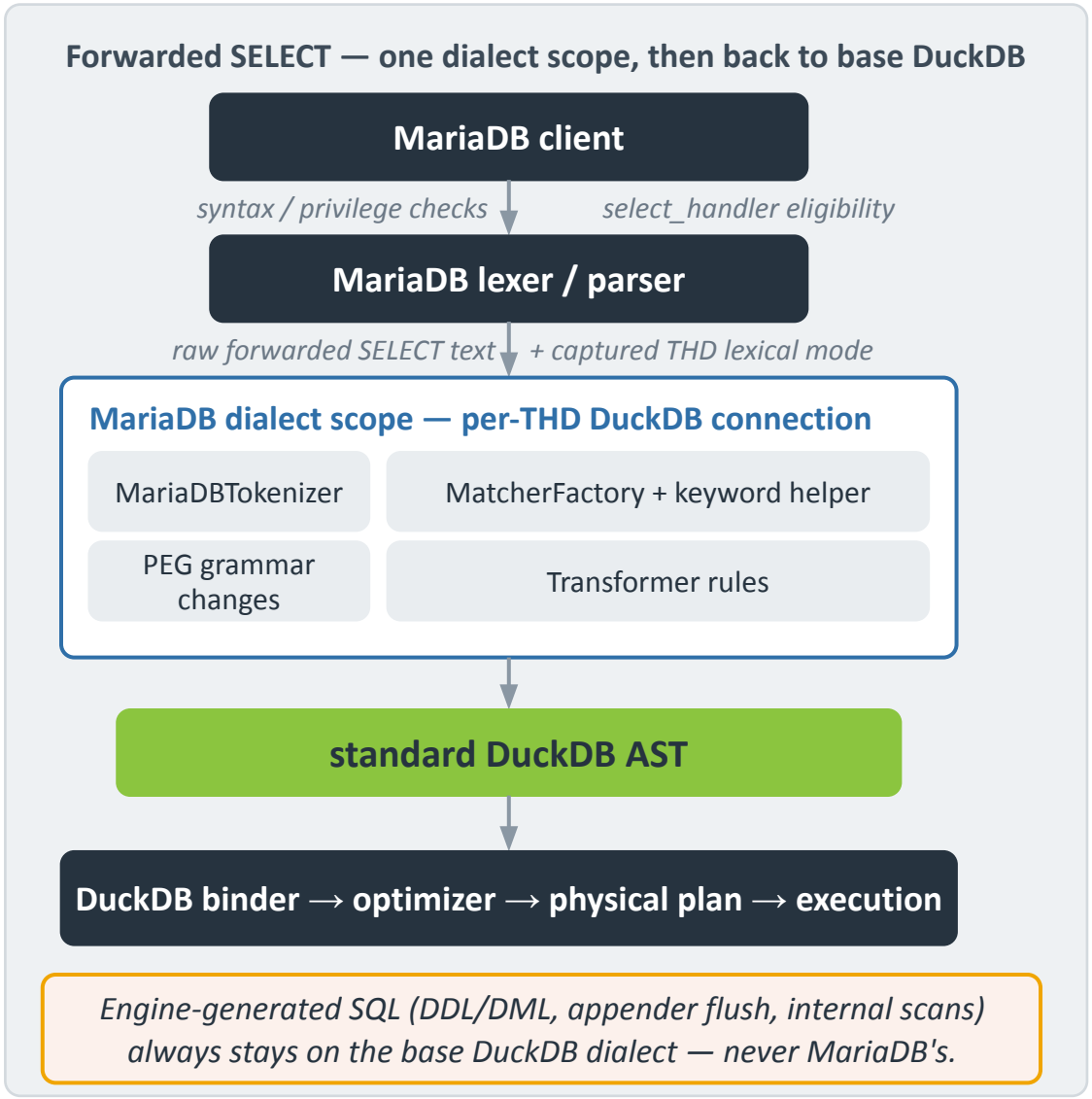
The Complete Design SELECT, DML, Coordinated DDL

- Every MariaDB privilege level global, database, table, column, role and PUBLIC is enforced live: GRANT, REVOKE, a role change or an account lock take effect on the very next statement.
- Authentication grows from TLS PLAIN / Unix peer into native/PARSEC, GSSAPI/SSPI and dialog/PAM through one reusable MariaDB authentication state machine.
- DDL is authorized the same way as SELECT and DML using MariaDB table autodiscovery feature.
- DuckDB stores no copied privilege catalog: unknown statement or object is always denied.

MariaDB



Duck's bark?



MariaDB Dialect Migration
One PEG Extension, Not a Rewriter

- init_scan() today does ad hoc text rewrites for WITH ROLLUP, CONVERT, the CURRENT_TIME family, STRAIGHT_JOIN etc.
- Target: MariaDB DialectExtension producing ordinary DuckDB AST wherever MariaDB syntax already has a DuckDB equivalent.
- DialectExtension, not GrammarExtension: it's mutually exclusive and owns the whole parser surface - backticks, ANSI_QUOTES-dependent quotes, NO_BACKSLASH_ESCAPES, executable comments, keyword classification. A composable grammar extension can't define a coherent dialect on its own.
- Requires a DuckDB 2.0 upgrade (pinned today: 1.5.5)

MariaDB

DuckDB engine description



How to install

- Available as MariaDB packag
  - 11.4.13
  - 11.8.9
  - 12.3.3



- Available as a Docker image:



- Soon available at the M-ecosystem Marina

MariaDB



Acknowledgement

- None of this would be possible without remarkable work of DuckDB Community and DuckLabs. Thank you for the Duck!
- MariaDB Foundation/Corporation for the Sea Lion!
- Alibaba and their engineering team for opensourcing AliSQL!
- Personally to Monty for suggesting Quacking narrative for this speech

MariaDB



Thank you and let's connect