AliSQL | MySQL + DuckDB

When MySQL Meets DuckDB

Best practices for columnar engines on MySQL



Zongzhi Chen
Head of Alibaba Relational Database Service (RDS)

Vadim Tkachenko
Percona CoFounder

PERCONA LIVE · SEPTEMBER 2026

AliSQL grows with open-source communities

01

MySQL community

200+
BUG REPORTS

80+
PATCHES

Selected AliSQL features shared upstream

- Hot-row updates
- Table recycle bin
- Row-level flashback
- Sequence engine
- Fast DROP for large tables

PRODUCTION FEATURES SHARED UPSTREAM

02

MariaDB community

Features merged upstream

- Large-transaction commit optimization
- Large-transaction recovery optimization
- Binlog format: FULL_NO_DUP

FEATURES VALIDATED AT SCALE

03

DuckDB community

30+
BUG REPORTS

28+
PATCHES

Features and fixes contributed upstream

- Transactions & concurrency
- Memory & storage
- Checkpointing
- Iceberg maintenance
- ART index
- WAL

FIXES & PATCHES SHARED UPSTREAM

Built on open source. Proven at scale. Contributed back upstream.

DuckDB: high-performance, open-source analytics

Run analytics where your data lives

In-process execution · No separate server to deploy

Python · Java · C++ · R · Wasm

Open data ecosystem Query files and cloud data directly

Parquet · JSON · S3 · Arrow · Iceberg · Delta · DuckLake

OPEN-SOURCE COMMUNITY

40K+ GitHub stars

3.6K Forks

MIT LICENSE

01

Column storage

Read only required columns.

Similar values compress well.

LESS I/O

02

Vector execution

Process values together.

Less work per row.

MORE WORK PER CYCLE

03

Parallel work

Scan and join across cores.

Aggregations · Disk spill

CORE UTILIZATION

04

Pruning

Zone maps skip chunks.

Statistics avoid wasted scans.

LESS WORK

TPC-H QUERIES

DuckDB vs. InnoDB speedup **>200×**

Evolution of HTAP systems

STAGE 01

OLTP only

MySQL

InnoDB rows

Fast transactions

Slow analytics

STAGE 02

Separate OLTP + OLAP

ROWS

ETL

COLUMNS

Faster analytics

More data movement and maintenance

STAGE 03 · INTEGRATED HTAP

One MySQL system

AliSQL

Protocol · ACL · SQL · metadata

InnoDB

OLTP · rows

DuckDB

OLAP · columns

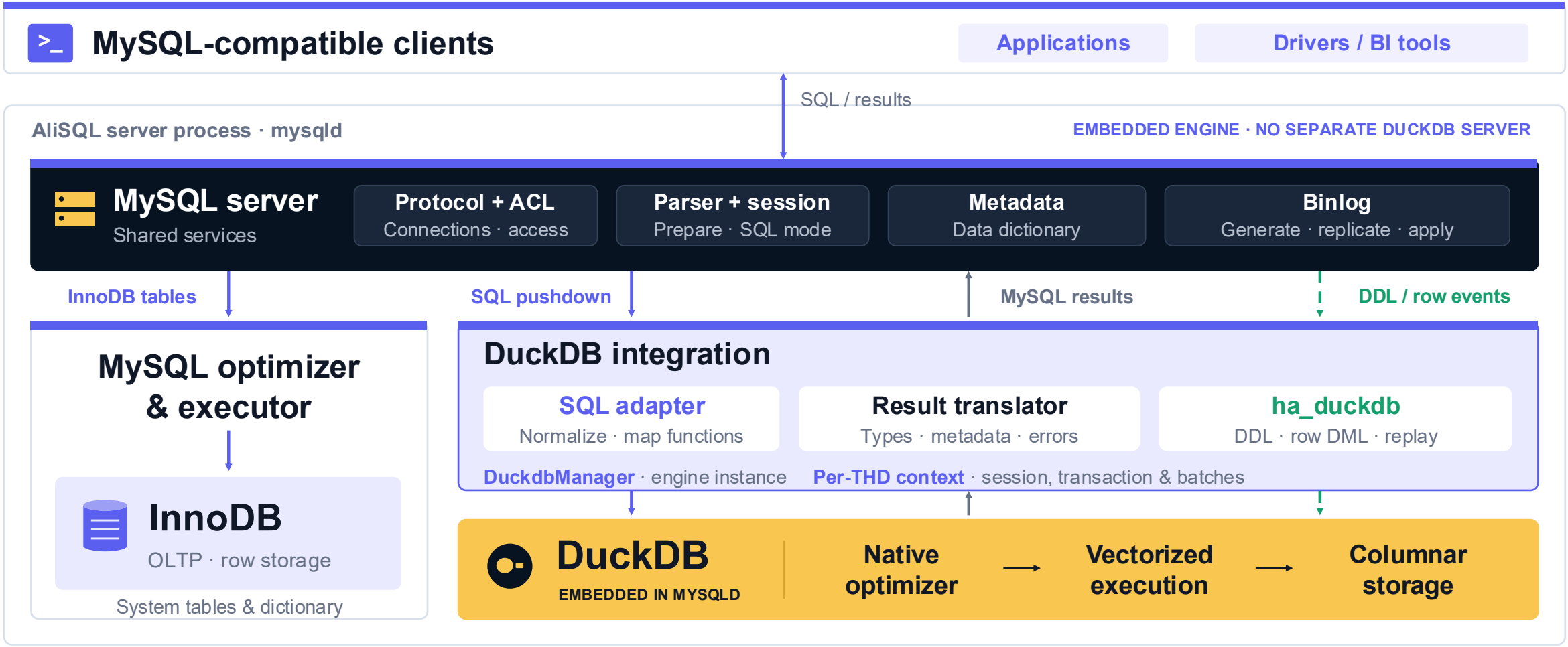
Choose InnoDB or DuckDB by workload.

THE OPERATING MODEL

One interface, fewer systems to maintain.

How DuckDB is embedded in the MySQL server

A shared MySQL server layer. A native DuckDB analytical execution path.



MySQL on the outside. **DuckDB execution on the inside.**

ENGINE INTEGRATION

Embedding DuckDB in AliSQL: three challenges

MySQL behavior. Columnar throughput. Safe recovery.

01

SQL compatibility

Preserve MySQL behavior

```
SELECT CONCAT('A', NULL);
```

MySQL **NULL** | Native DuckDB **'A'**

```
SELECT LENGTH('é');
```

MySQL **2** | Native DuckDB **1**

UTF-8: bytes vs. characters

99.99%

MySQL compatibility

02

Data synchronization

Turn rows into columnar batches

No batching — Low throughput



Batching — Amortized overhead



Same rows · fewer calls

Call overhead · Row data

Full import **~2M** rows/s

Binlog-based replay **~300K** rows/s

03

Crash recovery

No 2PC like InnoDB

Data ahead of replay progress



Replay → same final state

Validated in production

Illustrative examples · reported AliSQL results

Compatibility: end-to-end MySQL behavior

1

Normalize MySQL SQL

quotes · SQL mode · session

`` `order` → "order" ``

NORMALIZE FIRST

2

Reuse or map functions

reuse UDFs · fill gaps

`JSON_DEPTH('[1]') → 2`

MAKE LIMITS EXPLICIT

3

Run in DuckDB

keep the columnar plan

KEEP THE ENGINE NATIVE

4

Return MySQL results

types · metadata
warnings · errors

`TRUE / FALSE → 1 / 0`

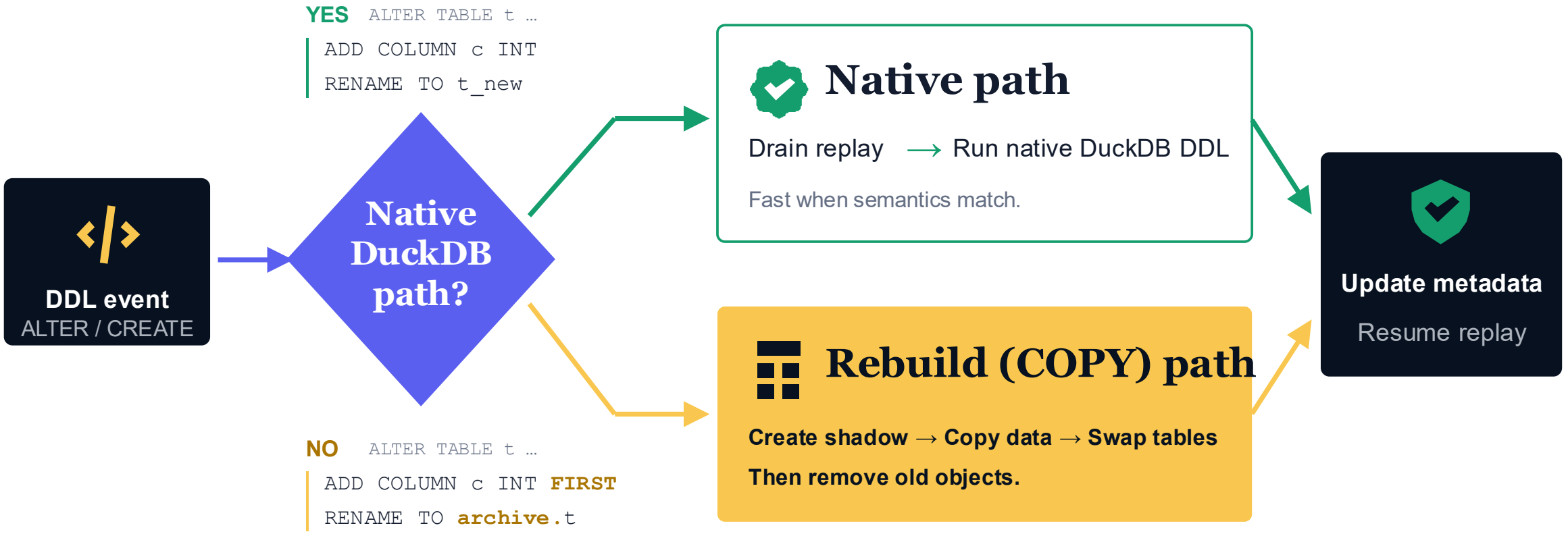
THE CLIENT SEES MYSQL

Supports 99.99% of MySQL statements

VALIDATE END TO END

SQL · types · collations
DDL · UDFs · session behavior

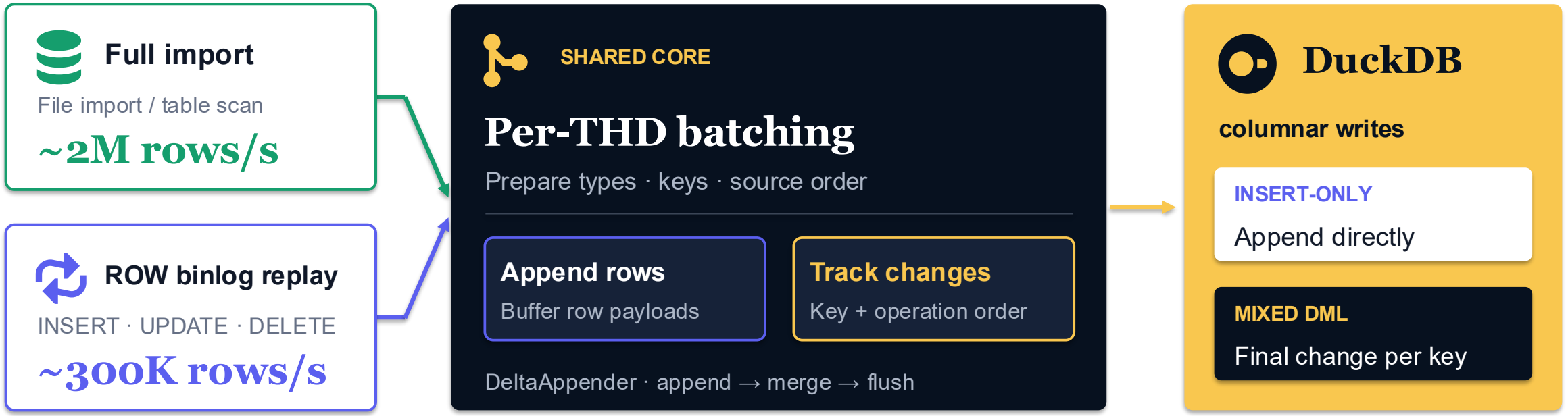
DDL compatibility: two execution paths



Rebuilds can be costly. Prefer native DuckDB DDL when semantics match.

Data Synchronization: batch row events/transactions

Buffer rows, track changes by primary key, and reconcile at flush.



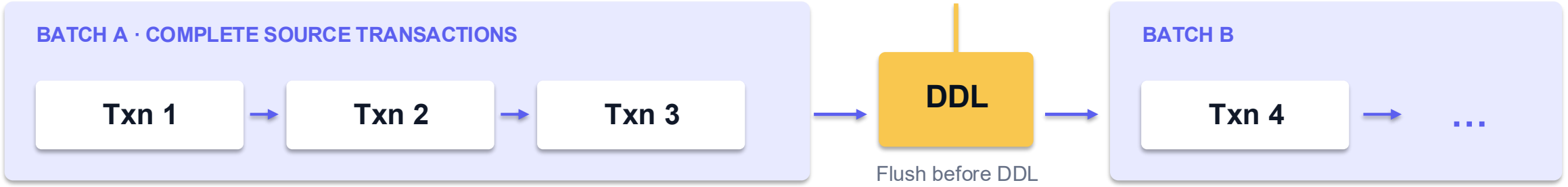
COMPUTE FINAL STATE vis PRIMARY KEY

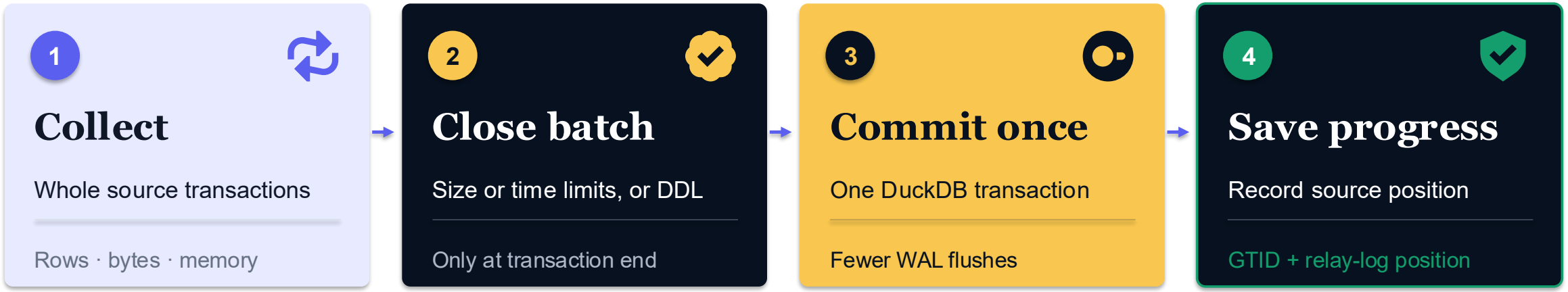
Example: primary key = 7

INSERT v = 10 → UPDATE v = 20 → UPDATE v = 30 → **Final row: (7, 30)**

Data Synchronization: when to stop batching

Group complete source transactions. Stop at DDL. Save replay progress.





THE BATCH-SIZE TRADE-OFF

Larger batches → fewer commits, but more recovery work.

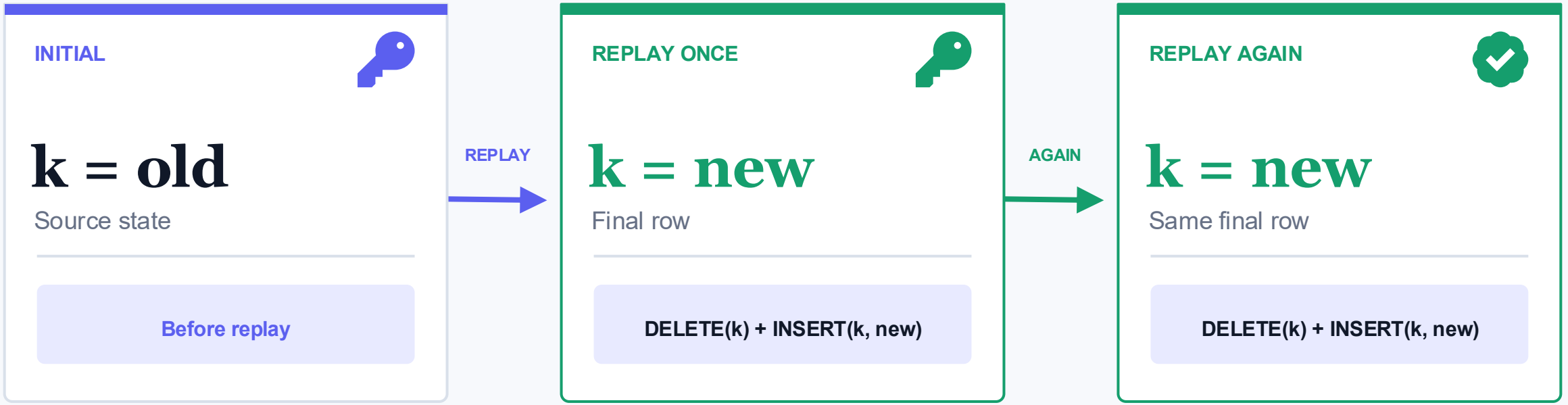
Recovery: handle an uncertain replay position

CRASH POINT	DUCKDB	REPLAY POSITION	ACTION
1 **Before data commit** Batch not durable.	Not advanced	Not advanced	Replay normally
2 **After data commit, before saving progress** Uncertain tail.	Advanced	Not advanced	**Replay safely** Deduplicate by primary key.
3 **After saving progress** Data and progress agree.	Advanced	Advanced	Resume after saved position

Keep the tail safe to replay until log progress is trusted.

Recovery: Consistent replay by primary key

Replay the same row events again. The final state stays the same.



IDEMPOTENT RESULT

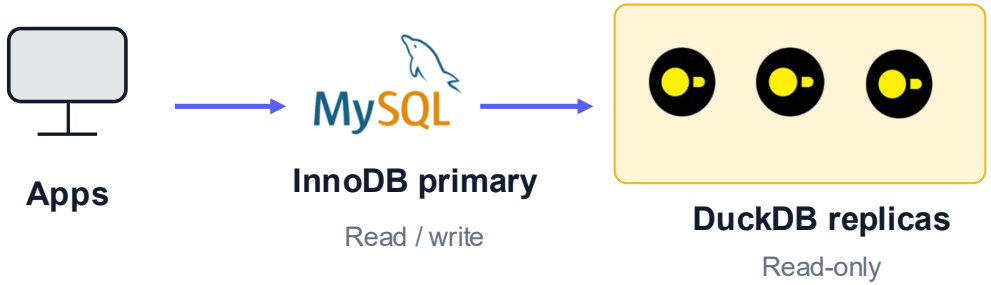
Same final row. No duplicates.

- **Stable primary key**
- **FULL row image**
- **PK change = delete old key + insert new key**

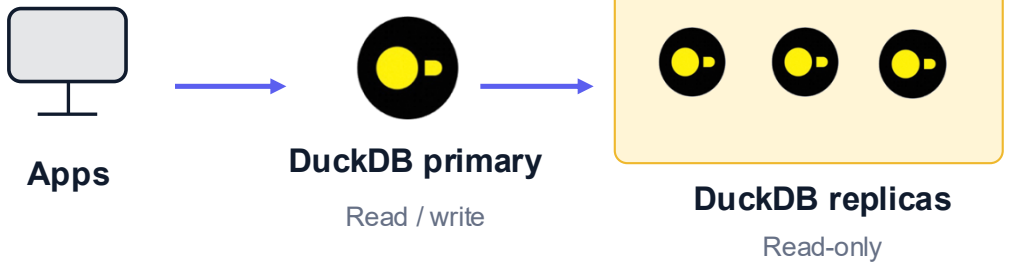
Deployment models

Choose a topology; keep MySQL-compatible access and native binlog replication.

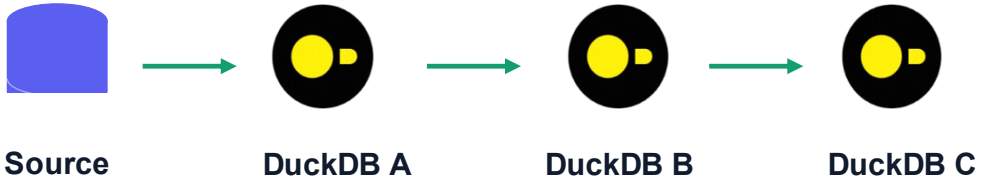
Analytical read-only



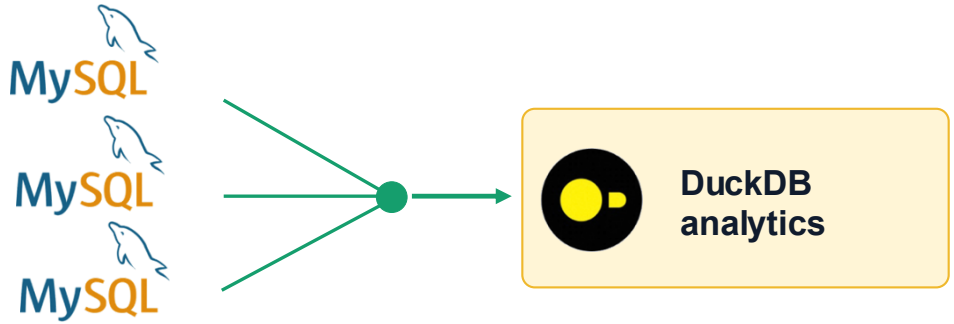
DuckDB analytical cluster



Cascading replication



Multi-source data aggregation



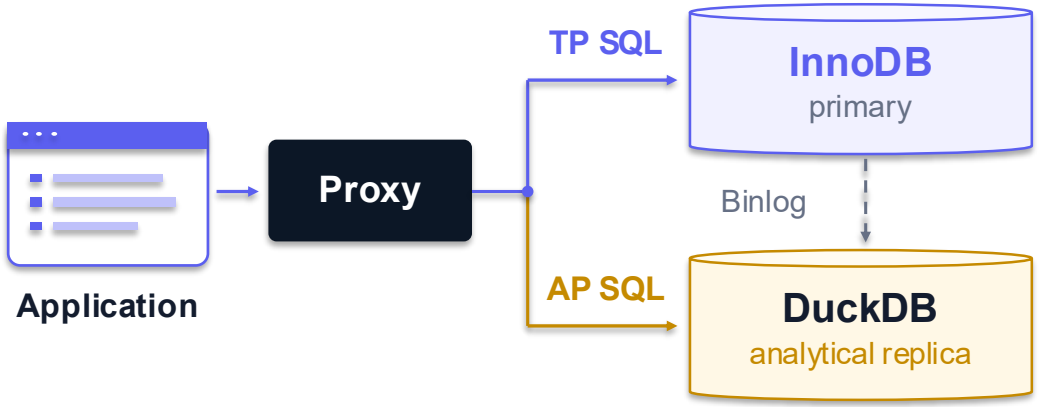
APPLICATION SCENARIOS

Applications of AliSQL DuckDB

Accelerate operational analytics and bring MySQL data together.

01 HTAP acceleration

Run reports on fresh operational data.



Workload separation

Keep transactions on InnoDB. Run reports on DuckDB.

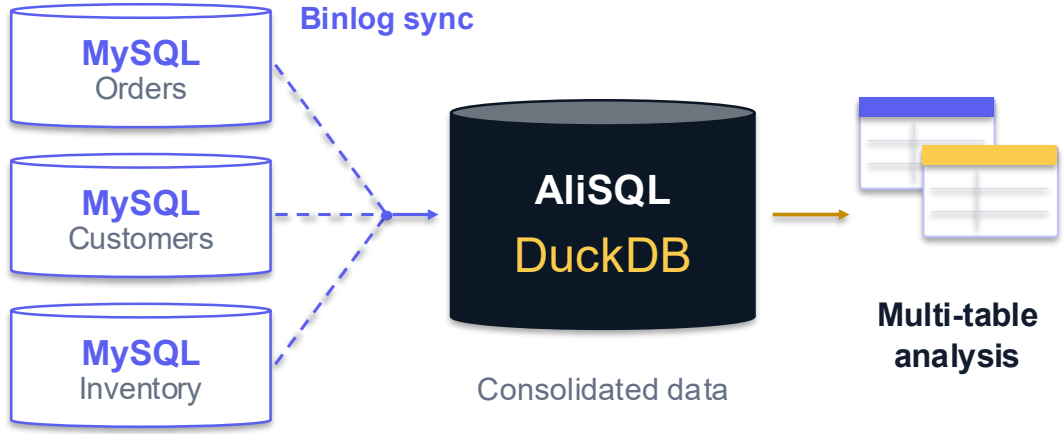
Fresh analytical data

Binlog replay refreshes the analytical replica.

EXAMPLE Order processing + live sales dashboards

02 Multi-source aggregation

Combine data from multiple MySQL instances.



One analytical dataset

Consolidate data from independent instances.

Cross-source analysis

Join tables and compute business-wide metrics.

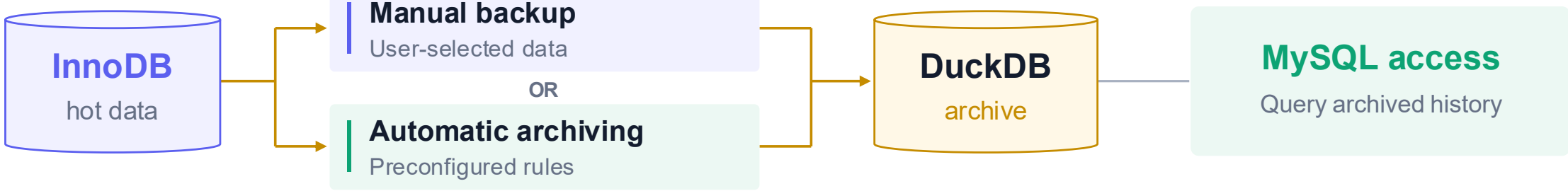
EXAMPLE Orders + customers → sales by region

Use existing MySQL clients and BI tools.

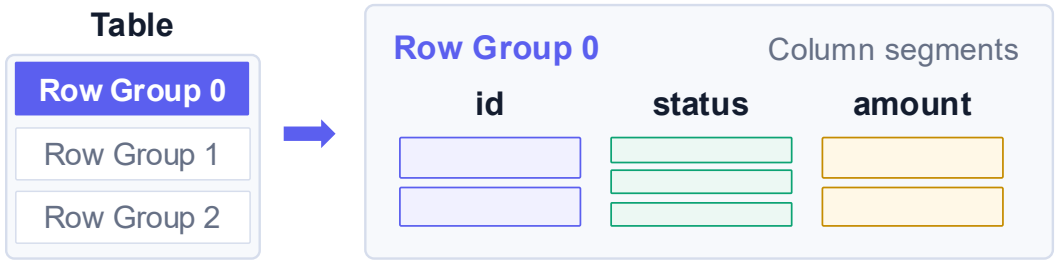
DATA ARCHIVING & STORAGE EFFICIENCY

Application: Archive data efficiently

Compressed history, still accessible through MySQL.



01 Columnar storage



Repeated values and local ranges compress well.

02 Adaptive compression

At checkpoint: choose encodings per column, within each row group.

- Run-length
- Bit packing
- Dictionary / FSST
- Floating-point
- Constant
- Validity bitmap

TPC-H SF100 Storage footprint

InnoDB ≈ 168 GB → DuckDB ≈ 26 GB

≈ 6.5× InnoDB / DuckDB

≈ 84% less storage

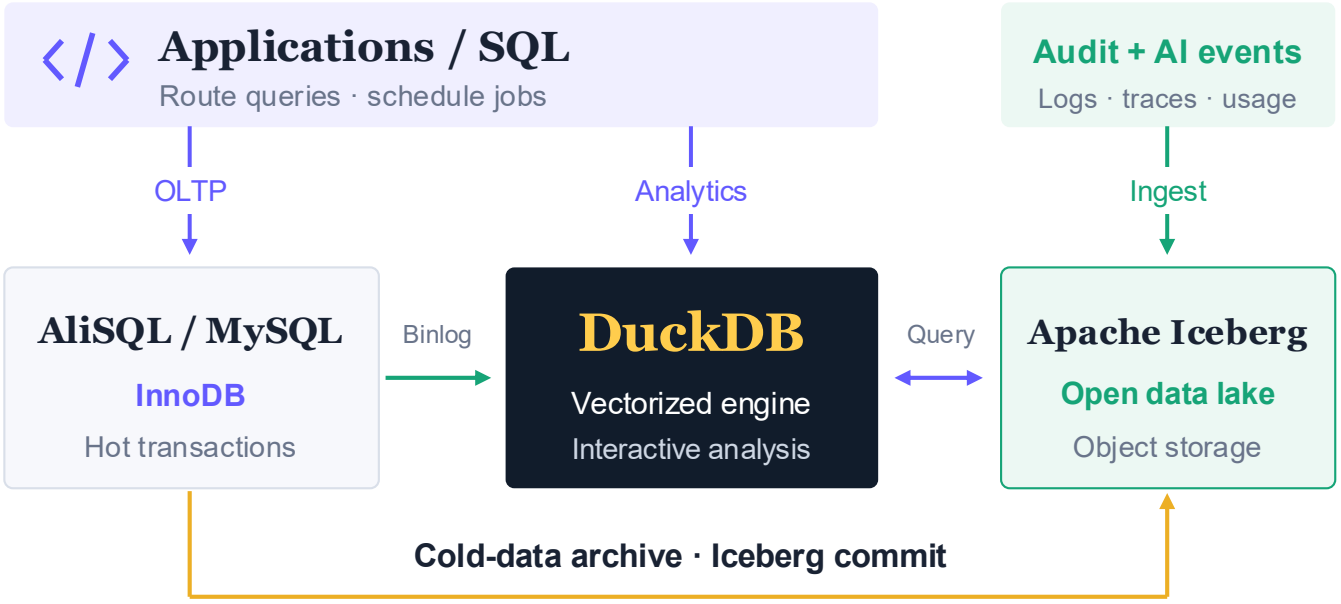
OPEN DATA LAKE

IN DEVELOPMENT

Internal service: DuckDB + Iceberg

Archive cold data. Analyze audit logs. Observe AI agents.

01 Open lake architecture



Keep hot data online. Retain history in the lake.

02 Service

Cold-data archiving

Retain history · read back on demand

Audit-log analytics

Filter · aggregate · investigate

AI-agent observability

- Traces
- Token usage
- Model usage
- Security events

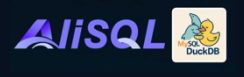
Dashboards · audit workflows

The integration idea is spreading

A shared direction, with different levels of maturity.

OPEN SOURCE

AliSQL



MySQL 8.0.44 branch

Guide specifies DuckDB 1.4.4

Includes replay, DDL, and recovery

ALPHA PREVIEW

MariaDB

Loadable engine plugin

Cross-engine joins

InnoDB + DuckDB in one query

EXPERIMENTAL

Percona

Selective query pushdown

Fast bulk loading

Analytical replica ROW + GTID

PRODUCTION GUARDRAILS

01 · Workload fit

02 · Compatibility tests

03 · Data synchronization

04 · Crash recovery

OPEN SOURCE · NEXT STEP

Read the code. Run the tests. Test the limits.

1

Adapt within the MySQL system

Keep HTAP behavior in one place.

2

Batch for columnar writes

Preserve row order. Write in larger batches.

3

Make recovery safe

Identify crash windows. Deduplicate replay.

https://github.com/alibaba/AliSQL

https://alisql.net

Thank you · Q&A