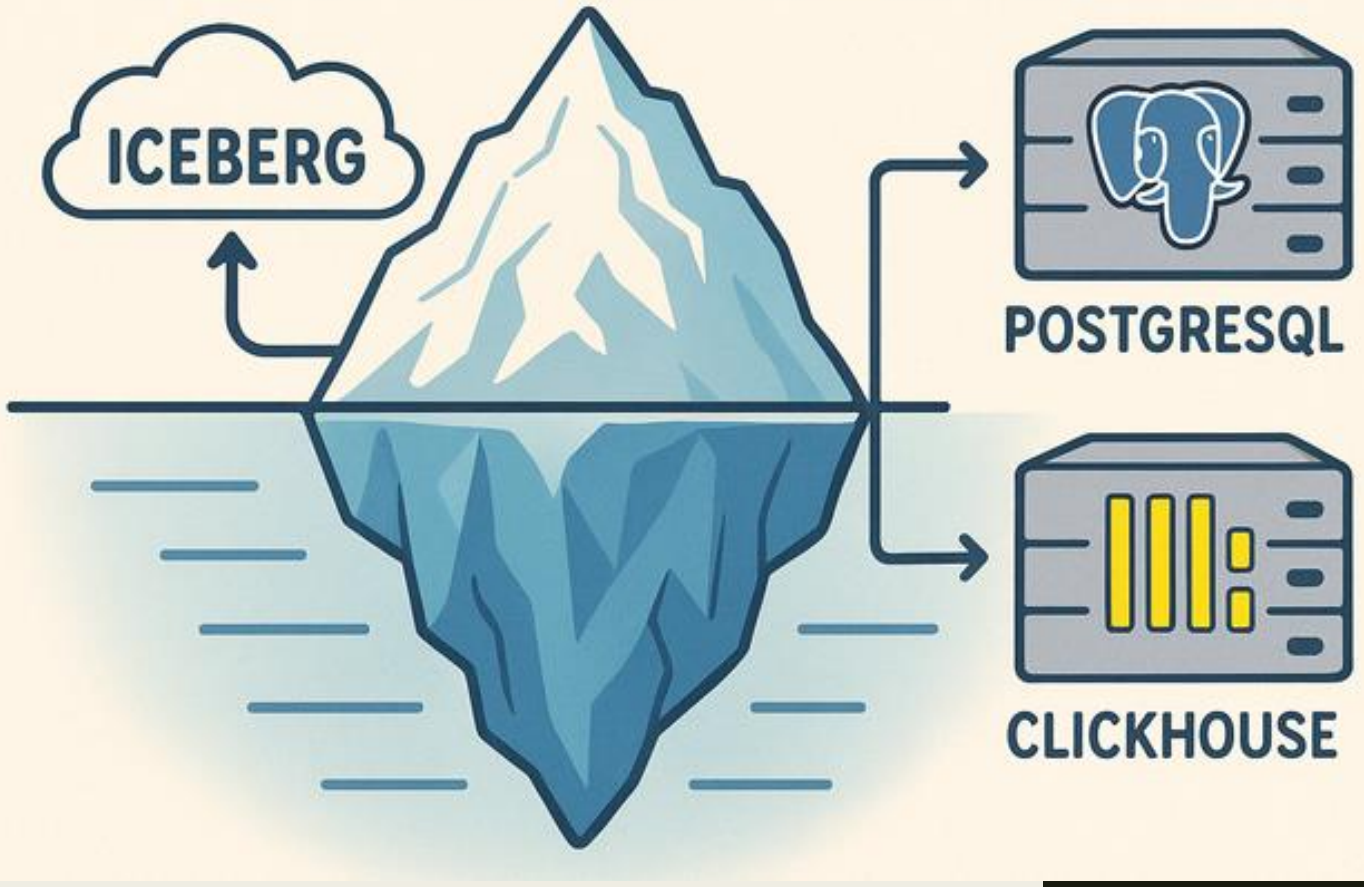
RUNNING ICEBERG ON EXISTING DBS



ABOUT US

Shivji Kumar Jha
Staff Engineer
Data Platforms, Nutanix

- Areas of Interest
  - Distributed Storage Infrastructure
  - Database & Streaming Internals
  - Application Architectures
- Passion for OSS DBs and Communities
  - Contributed to MySQL and Apache Pulsar
  - Exploring Postgres, Clickhouse & iceberg*
- 25+ talks at conferences & meetups
  - Slides: github.com/shiv4289/shiv-tech-talks/
  - Recordings: https://tinyurl.com/shiv-talks
- Text here: linkedin.com/in/shivjijha/

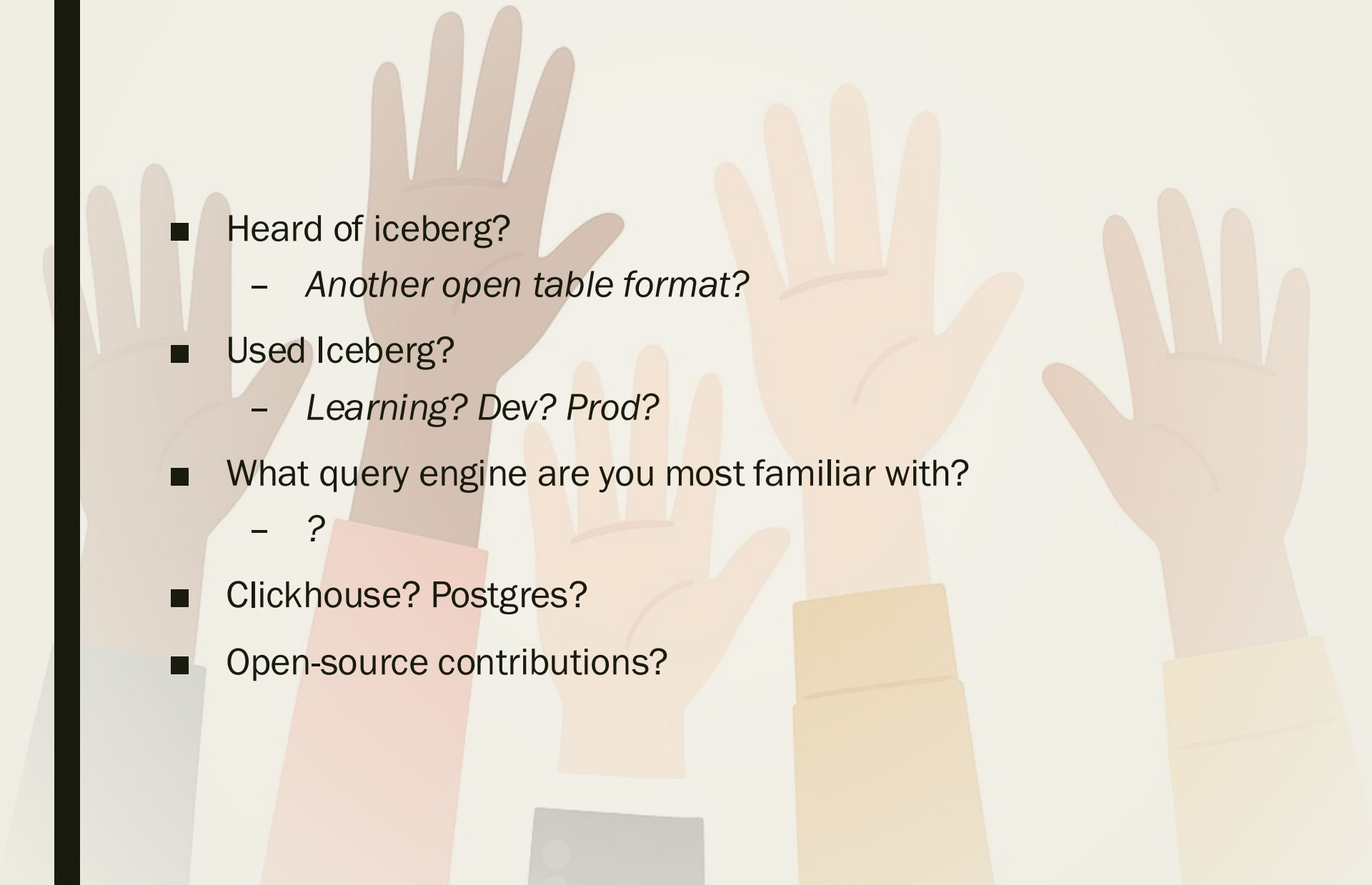
Saurabh Ojha
Software Engineer
Open-source hacker
Data Platforms, Nutanix

- Areas of Interest
  - Databases Internals
  - Streaming Internals
  - Linux kernel Features
- Open-Source Enthusiast
- Mostly active here:
  - linkedin.com/in/ojhasaurabh2099/

Safe Harbor Statement

The views, opinions, and conclusions presented in these slides are solely my own and do not reflect the official stance, policies, or perspectives of my employer or any affiliated organization. Any statements made are based on my personal experiences and research and should not be interpreted as official guidance or endorsement.

Show of hands



- Heard of iceberg?
  - *Another open table format?*
- Used Iceberg?
  - *Learning? Dev? Prod?*
- What query engine are you most familiar with?
  - *?*
- Clickhouse? Postgres?
- Open-source contributions?

Agenda

- Iceberg's Appeal & Adoption
- The ever-growing adoption
  - *Even your old boring open-source Databases*
- Iceberg and the OLAP Databases ft. Clickhouse
- Iceberg and the OLTP Databases ft. Postgres
- Demos (if time permits… hopefully ☺)
- Takeaway:
  - *Great time to be a builder*
  - *i.e., if you love hacking around open source*

ICEBERG'S APPEAL AND ADOPTION

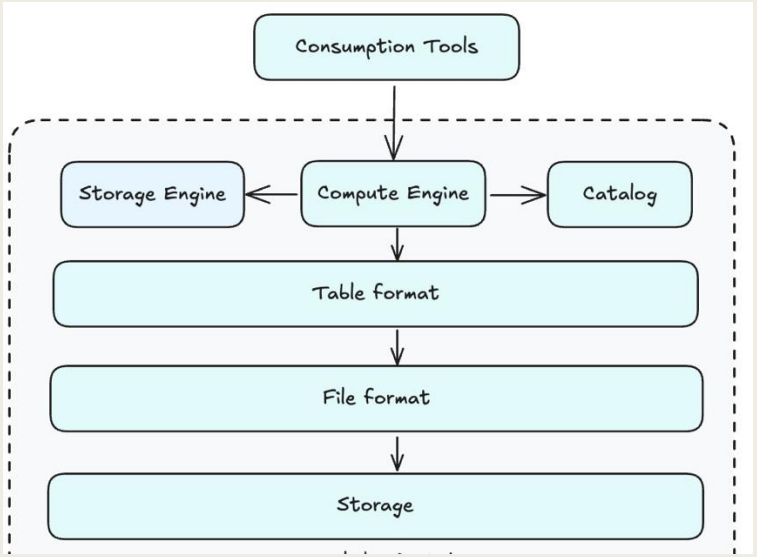
Setting the Context

- Open table formats like Iceberg are becoming foundational in modern data lakes
- Traditionally used with engines like Spark, Trino, Dremio etc
- Even stream-native platforms like Kafka (TableFlow), Pulsar (Ursa), and RisingWave are adopting Iceberg
- AWS S3 Tables saw a lot eyeballs
- Adoption signals a broader movement towards modular, decoupled lakehouse architectures
- Even your existing, old and boring databases are embracing it

Iceberg's Appeal

- Works well with object stores (e.g., S3, GCS)
- Open format, less lock-in
- Choice of query engine (More options, less locking of your data... the dreaded migrations!)
  - *Got a data migration story?*
- Decouples storage and compute
- Schema evolution, time travel, partition pruning, ACID
- Increasing support across the ecosystem
- A lot of mindshare at the moment!

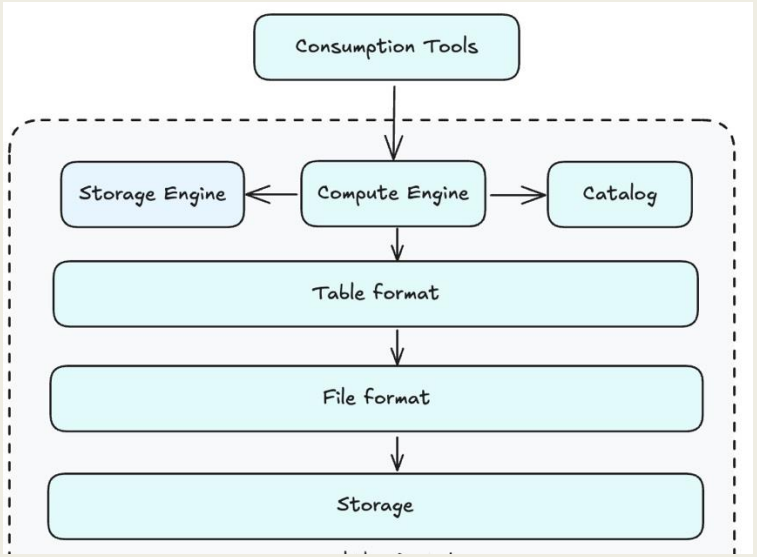
Deconstructing a database



- Storage
  - *Store large datasets*
  - *Local filesystem, attached storage, distributed file system, Object storage*
  - *Row vs column-oriented store*
- File format
  - *Impacts compression, data structure, performance.*
  - *Structured(CSV), semi-structured(JSON), unstructured(text file)*
  - *Row-oriented(CSV, Apache Avro), Column Oriented(Parquet, ORC)*
- Table format
  - *Metadata: specifies how datafiles should be laid out in storage*
  - *Abstract the complexity of physical data structure*
  - *Abstract with DMLs (insert, update, delete) schema evolution etc*
  - *Safe DML access with atomicity, consistency etc*

– Apache Iceberg the definitive guide (OREILLY)

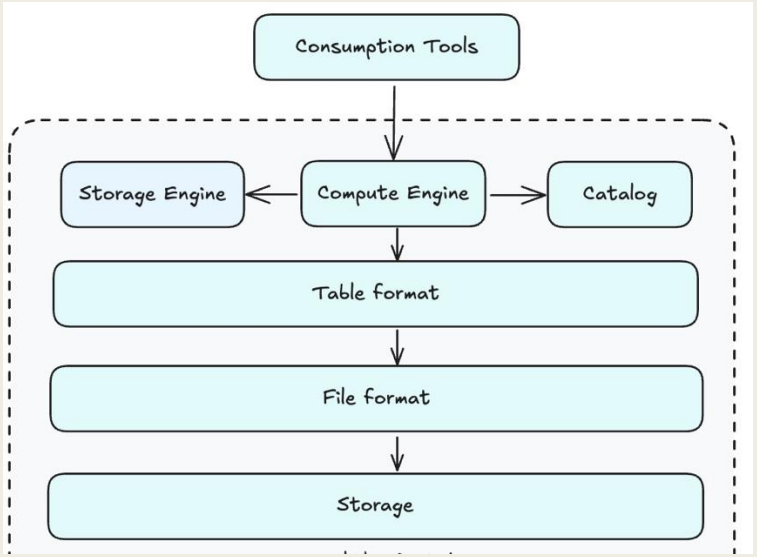
Deconstructing a database



- Table format
  - *Metadata: specifies how datafiles should be laid out in storage*
  - *Abstract the complexity of physical data structure*
  - *Abstract with DMLs (insert, update, delete) schema evolution etc*
  - *Safe DML access with atomicity, consistency etc*
- Storage Engine
  - *Layout data in the form specified by table format*
  - *Keep files and data structures up-to-date with new data*
  - *Physical optimization of data*
  - *Index maintenance*
  - *Expiry / delete old data*

– Apache Iceberg the definitive guide (OREILLY)

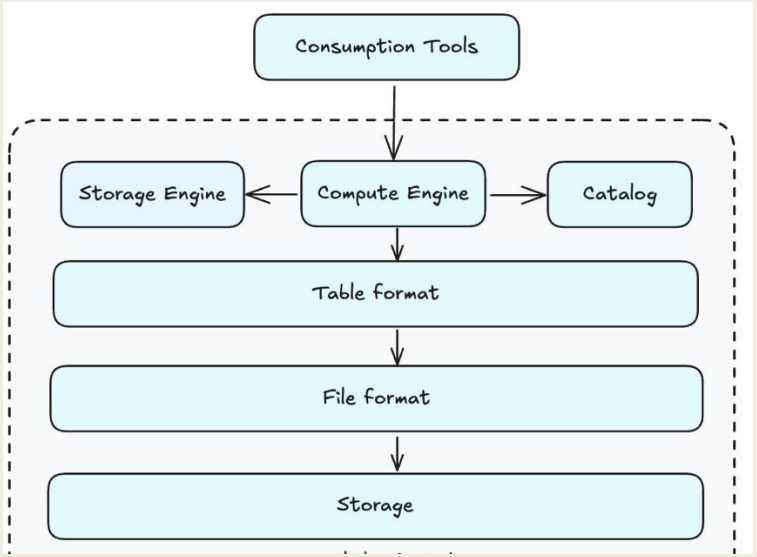
Deconstructing a database



- Storage Engine
  - *Layout data in the form specified by table format*
  - *Keep files and data structures up-to-date with new data*
  - *Physical optimization of data*
  - *Index maintenance*
  - *Expiry / delete old data*
- Catalog
  - *Identify dataset needed for analysis using metadata (from table format)*
  - *Users / compute engines go to catalog for existence of a table*
    - Get table name, table schema, location of data on storage
  - *Traditionally, internal to system and only engine can access*
  - *Hive: one of the first systems open for any system to use*
  - *Different from metadata catalog!*

– Apache Iceberg the definitive guide (OREILLY)

Deconstructing a database



- Catalog
  - *Identify dataset needed for analysis using metadata (from table format)*
  - *Users / compute engines go to catalog for existence of a table*
    - Get table name, table schema, location of data on storage
  - *Traditionally, internal to system and only engine can access*
  - *Hive: one of the first systems open for any system to use*
  - *Different from metadata catalog!*
- Compute Engine
  - *Run user workloads to process data*
  - *Choose as per data size, compute load, type of workload etc*
  - *Scale? Use distributed compute engines*
    - Spark, Dremio, e6Data

– Apache Iceberg the definitive guide (OREILLY)

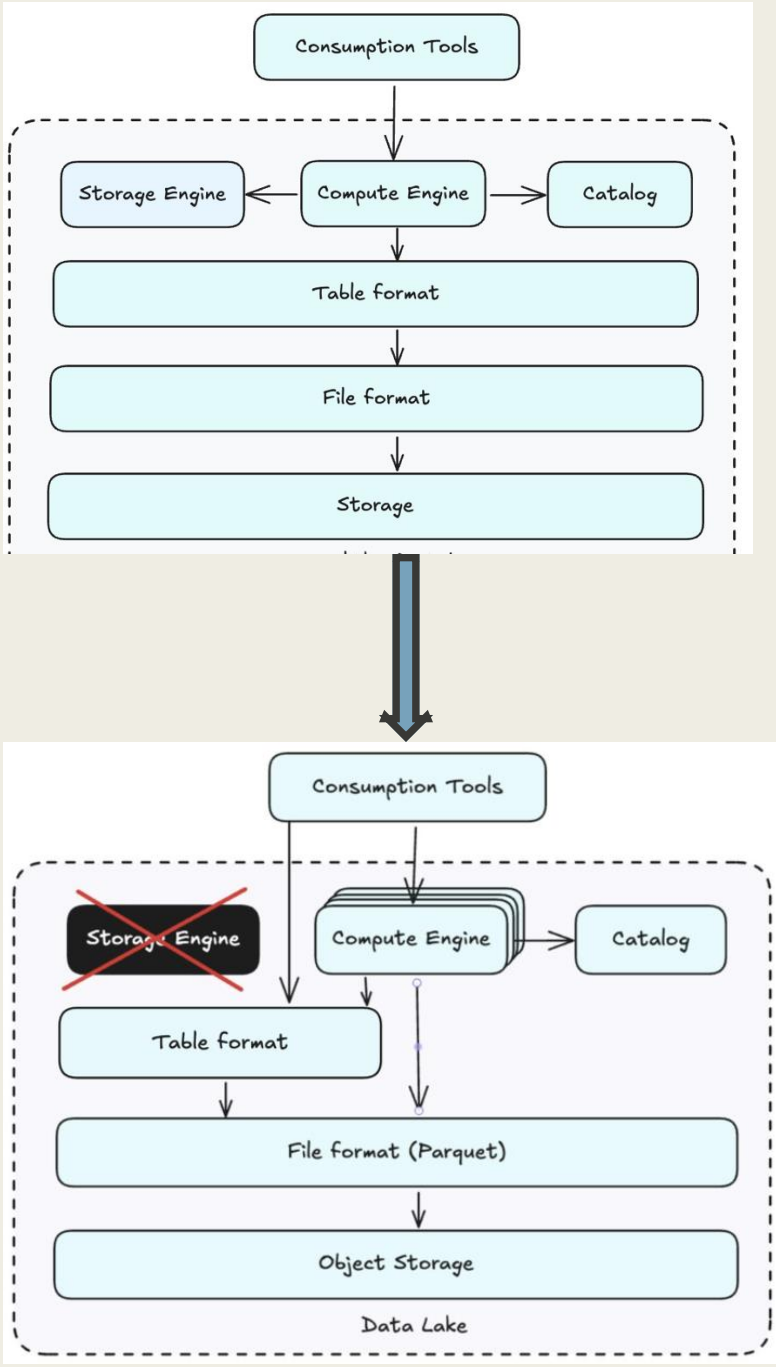
Warehouse & Lake

✅ Pros of a Data Lake

- Lower cost
- Stores data in open formats
- Handles unstructured data
- Supports ML use cases

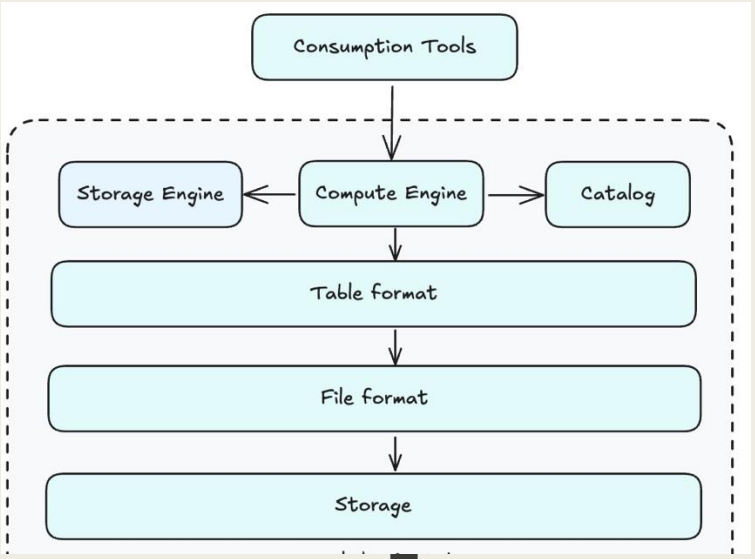
⚠️ Cons of a Data Lake

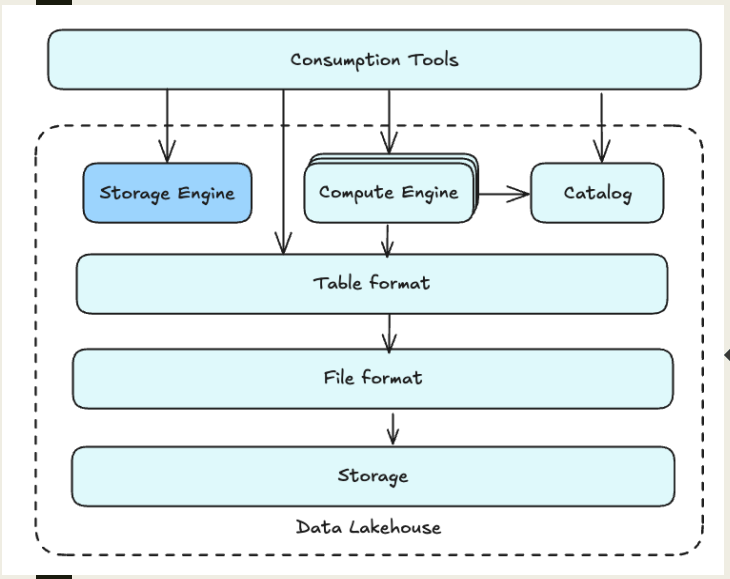
- Performance
- Lack of ACID guarantees
- Lots of configuration required

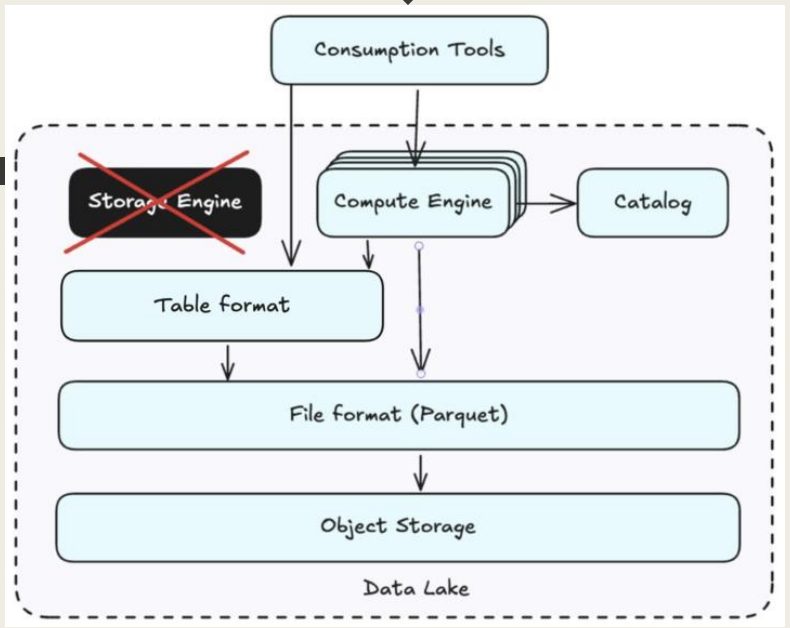


– Apache Iceberg the definitive guide (OREILLY)

Data Lake to Lakehouse

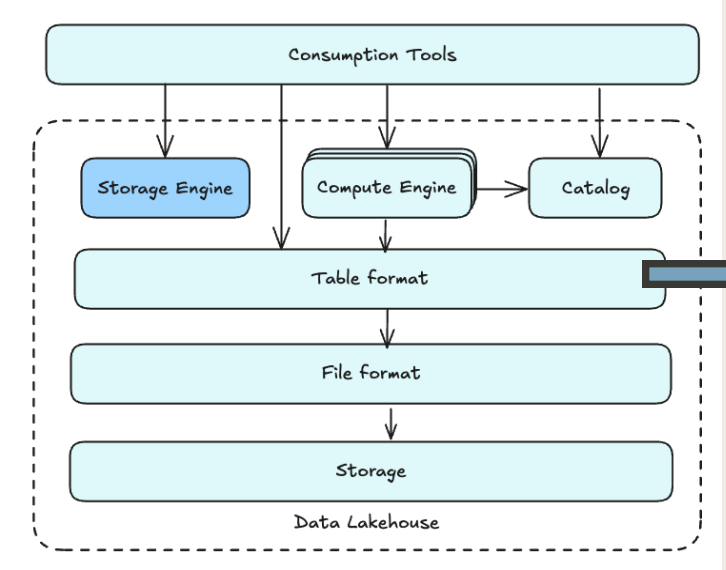




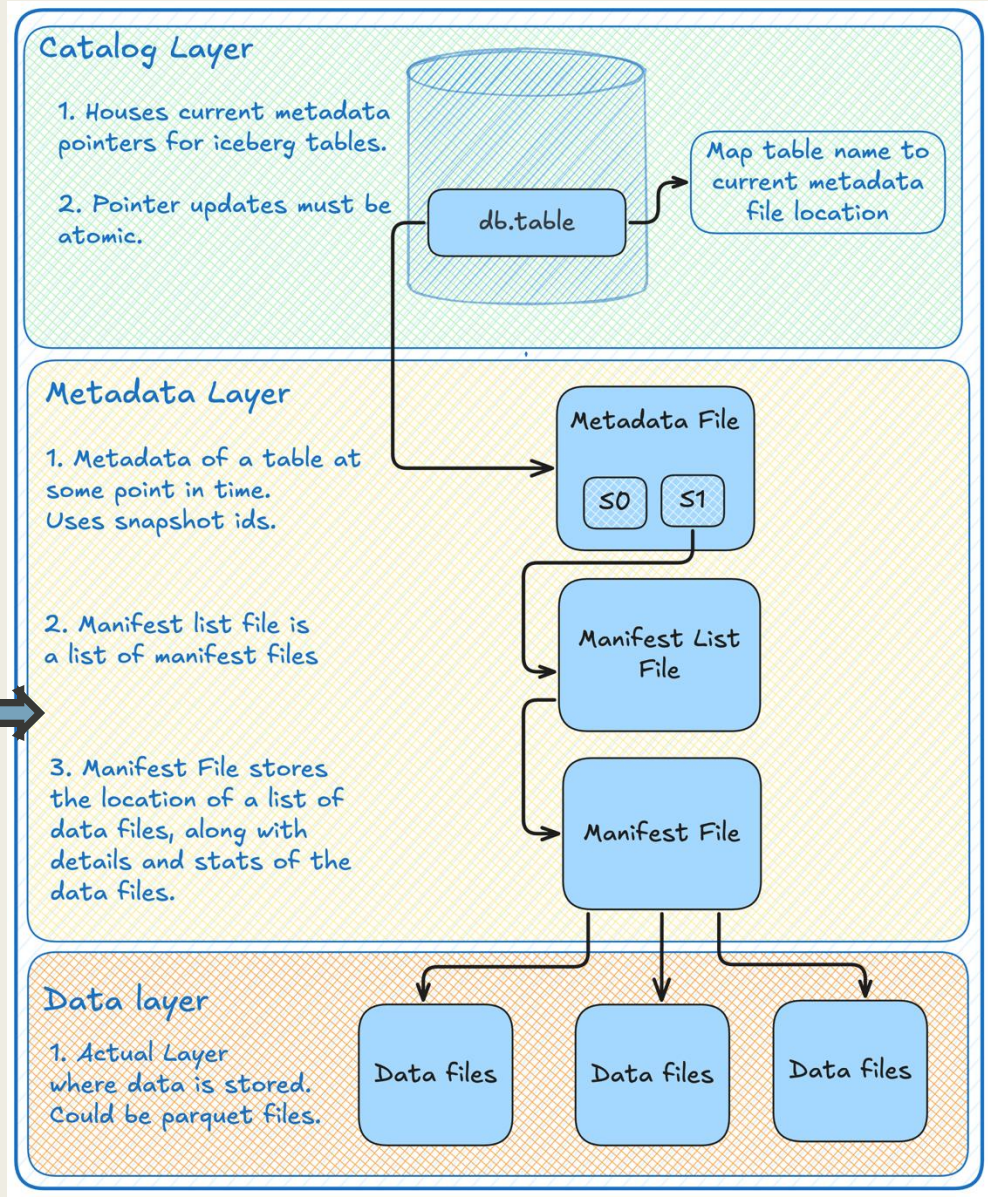


– Apache Iceberg the definitive guide (OREILLY)

Iceberg Table Format



– Apache Iceberg the definitive guide (OREILLY)

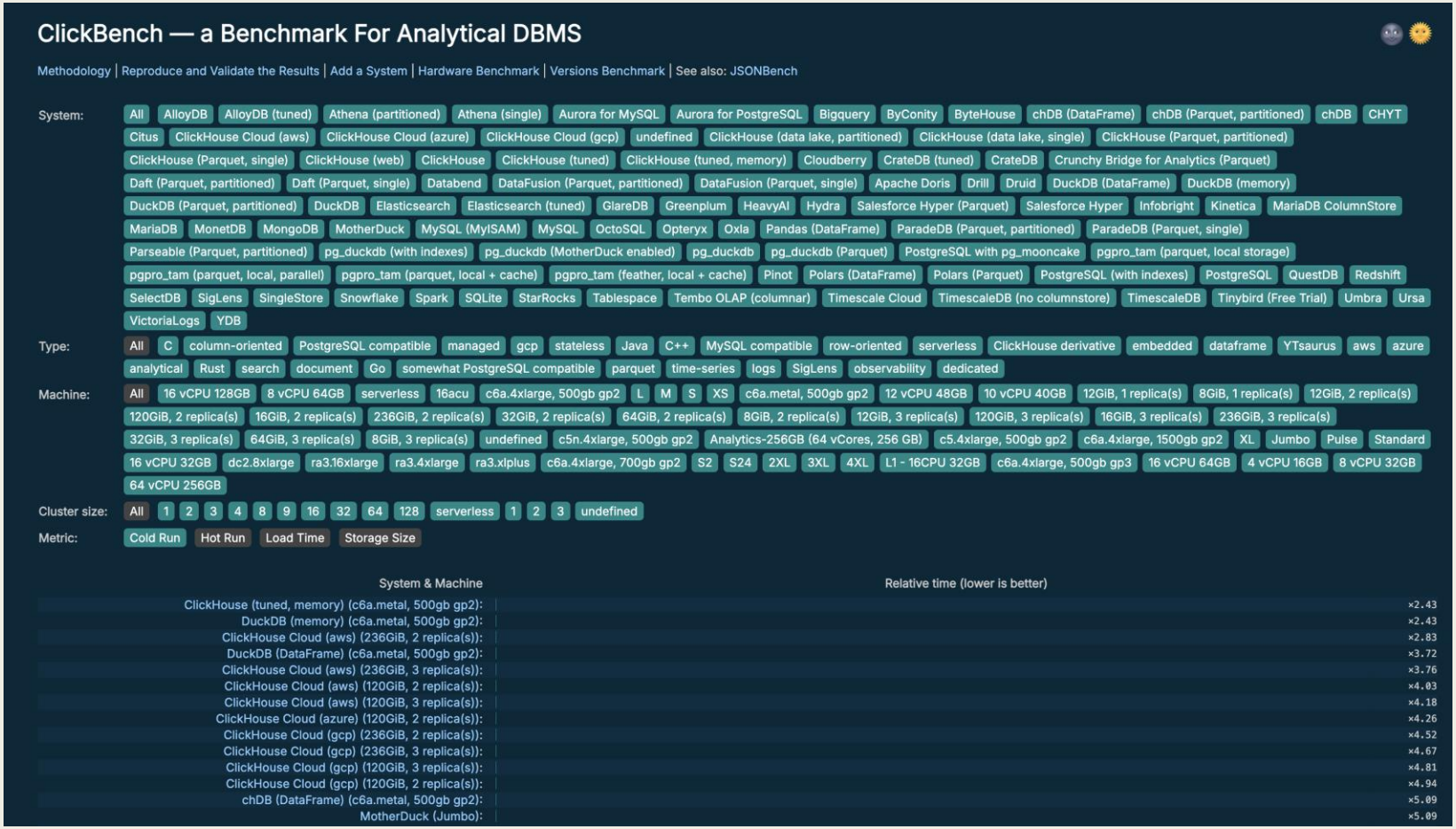


https://www.dremio.com/resources/guides/apache-iceberg-an-architectural-look-under-the-covers/

CLICKHOUSE & ICEBERG

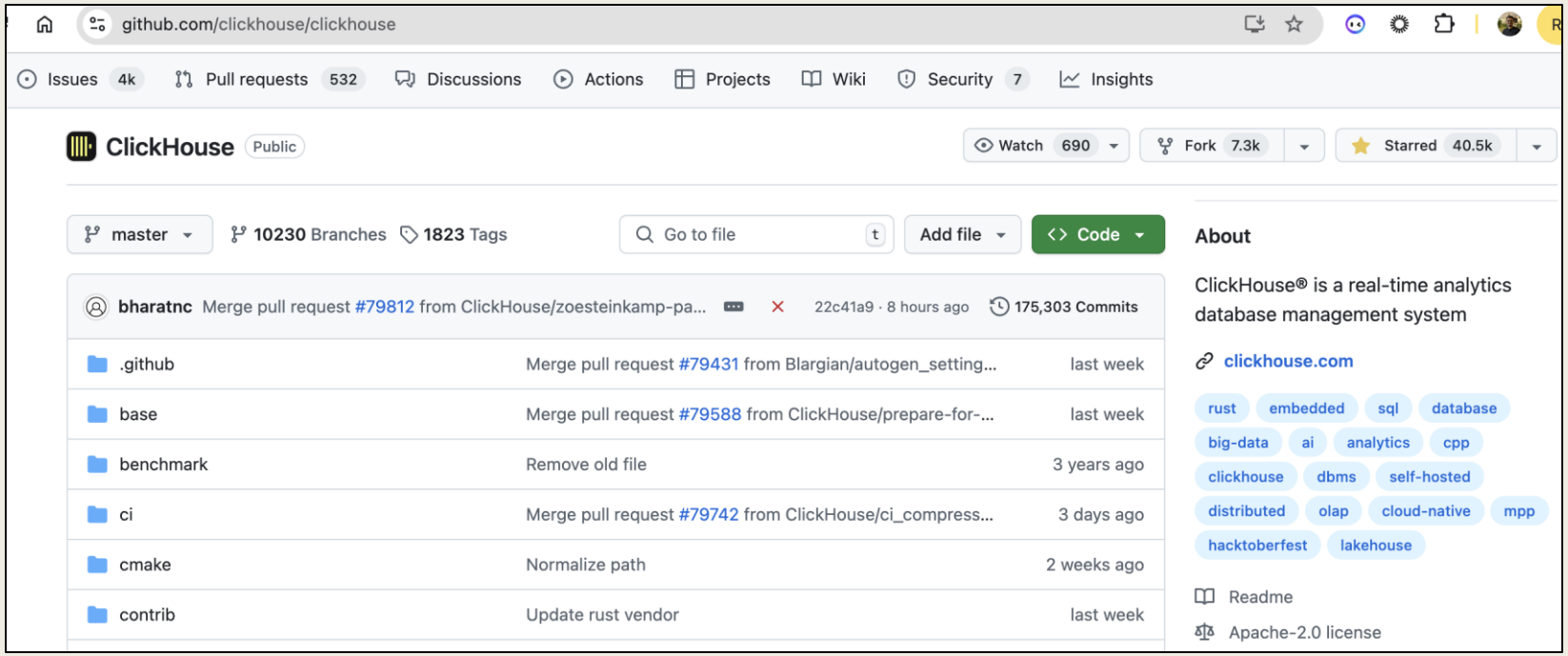
Introducing ClickHouse!

- Open source* OLAP database known for speed



Introducing ClickHouse!

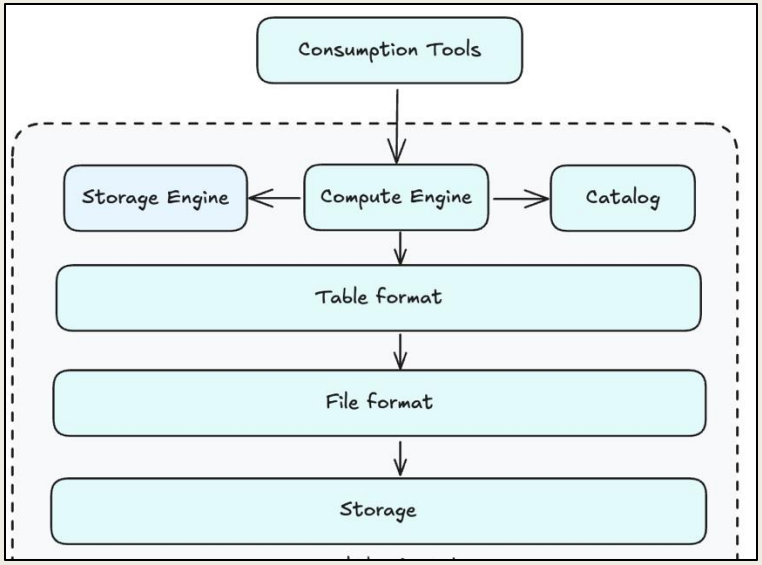
- Open source* OLAP database known for speed (see ClickBench!)
- Very easy to get started
  - *one binary, regular SQL/Tables*
  - *Very popular, open, well adopted!*

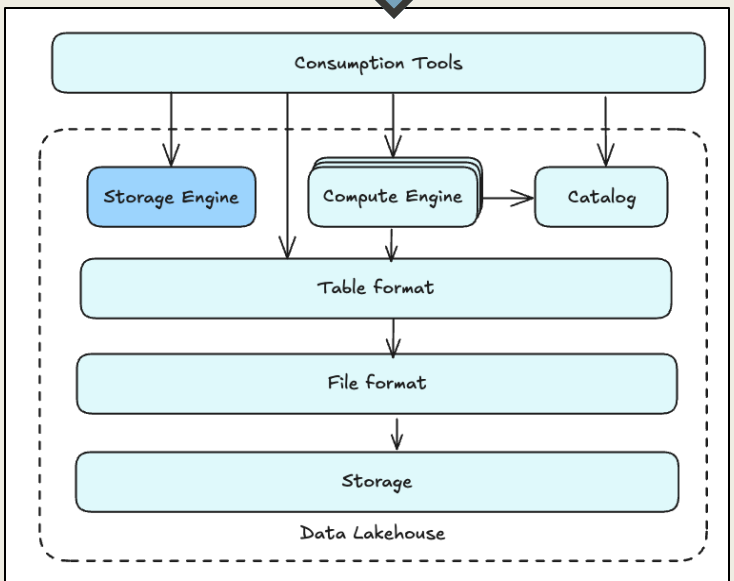


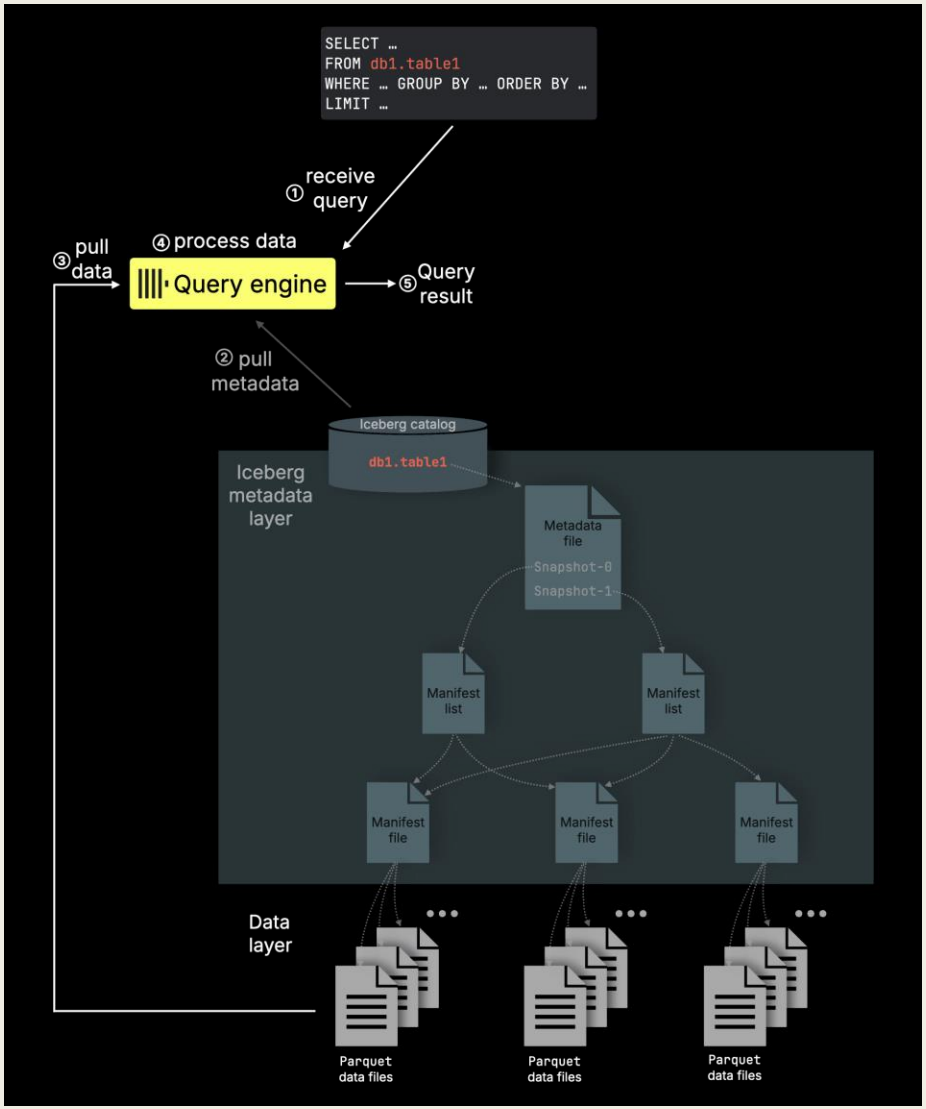
Introducing ClickHouse!

- Open source* OLAP database known for speed
- Very easy to get started
  - *one binary, regular SQL/Tables like RDBMS*
  - *Very popular, open, well adopted!*
- Note: Very different from an RDBMS (row vs column, oltp vs olap etc)
- Traditionally, loved big fat servers (vertical scaling…)
  - *Now shards & replicas too.*
- Storage Compute segregation(?)
  - *Possible with Open source, not trivial!*
- But a lot of integrations (engines, formatReaders etc)

Query Engine for lakehouse

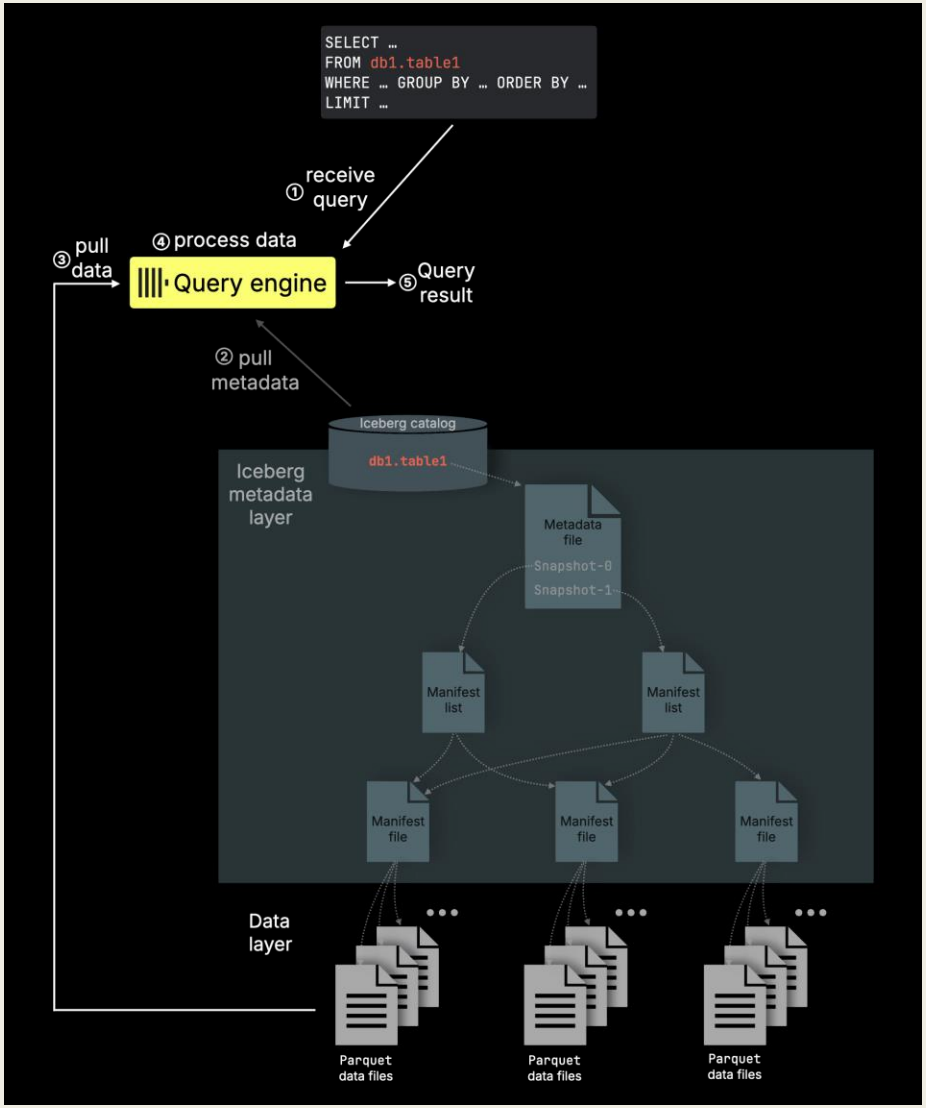


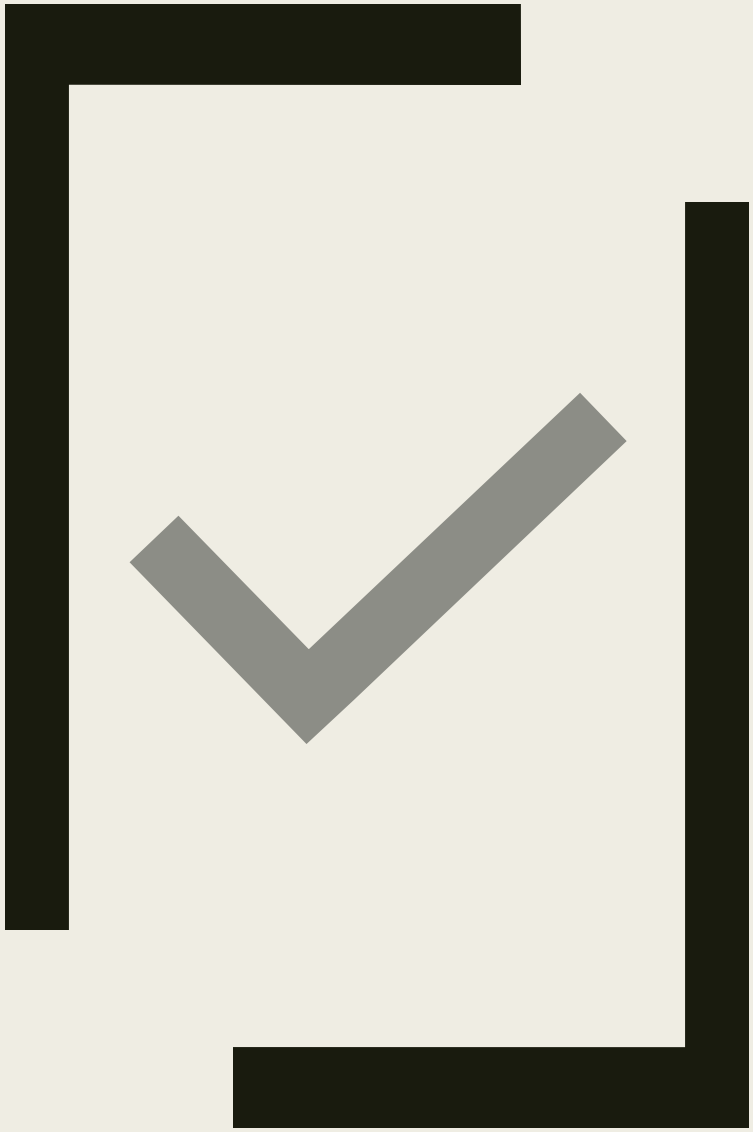




Query Engine Requirements

- Read data in object store
  - *Maybe cache too?*
  - *At least an option!*
- Query Parquet
  - *join with other data sources?*
- Data structures for meta files
  - *Caching too maybe?*
- Support Lakehouse features
  - *Catalog integrations*
  - *Time travel,*
  - *Partition Pruning*
  - *Schema Evolution etc*



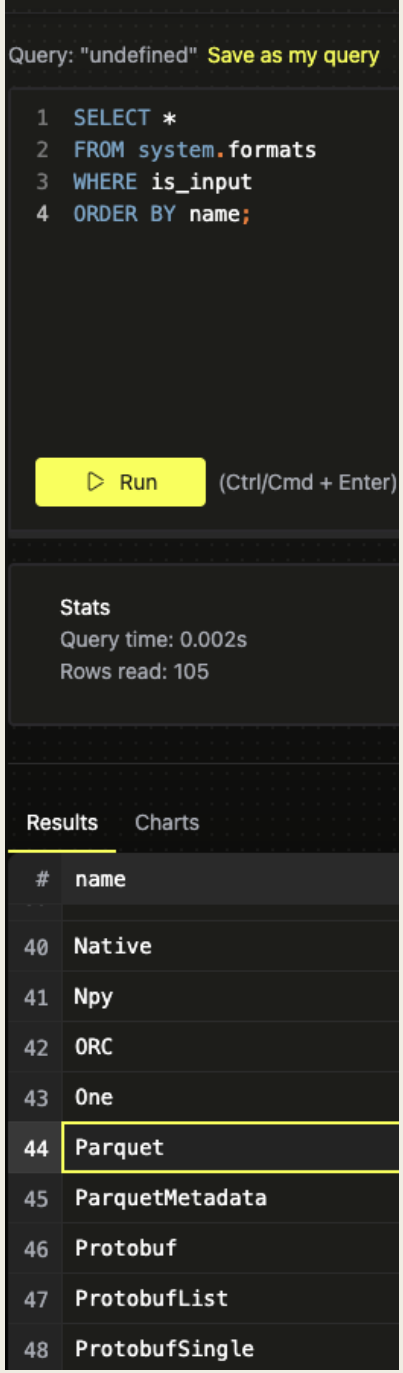


INTEGRATIONS IN CLICKHOUSE

Query file formats

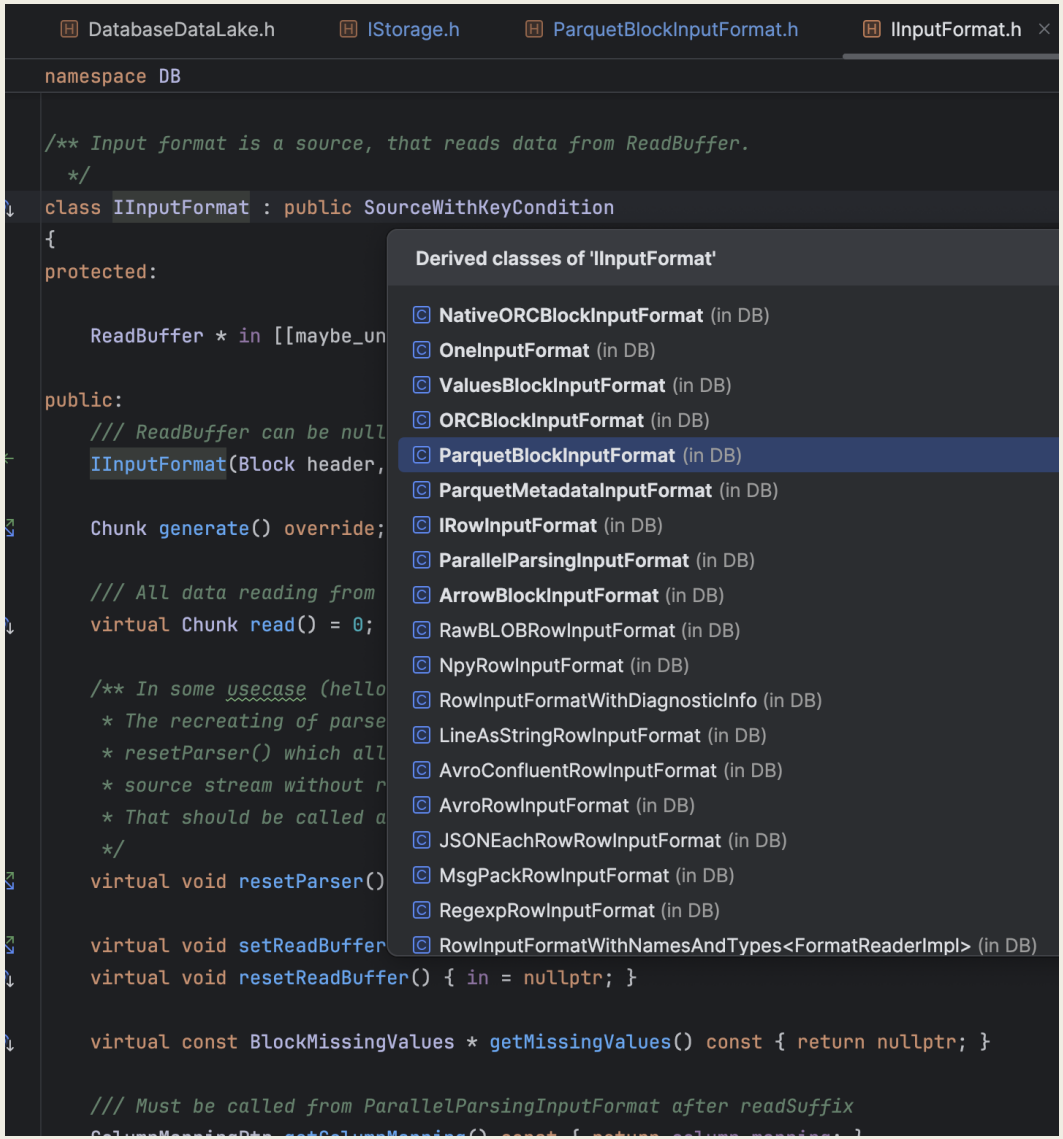
ClickHouse supports 70+ file formats out of the box"

- Parquet
- JSON
- CSV
- Apache Arrow
- ORC
- CSV
- XML
- … and more
- No ingestion required
  - *Direct SQL queries with dictionaries, joins, window functions, 160+ aggregations*



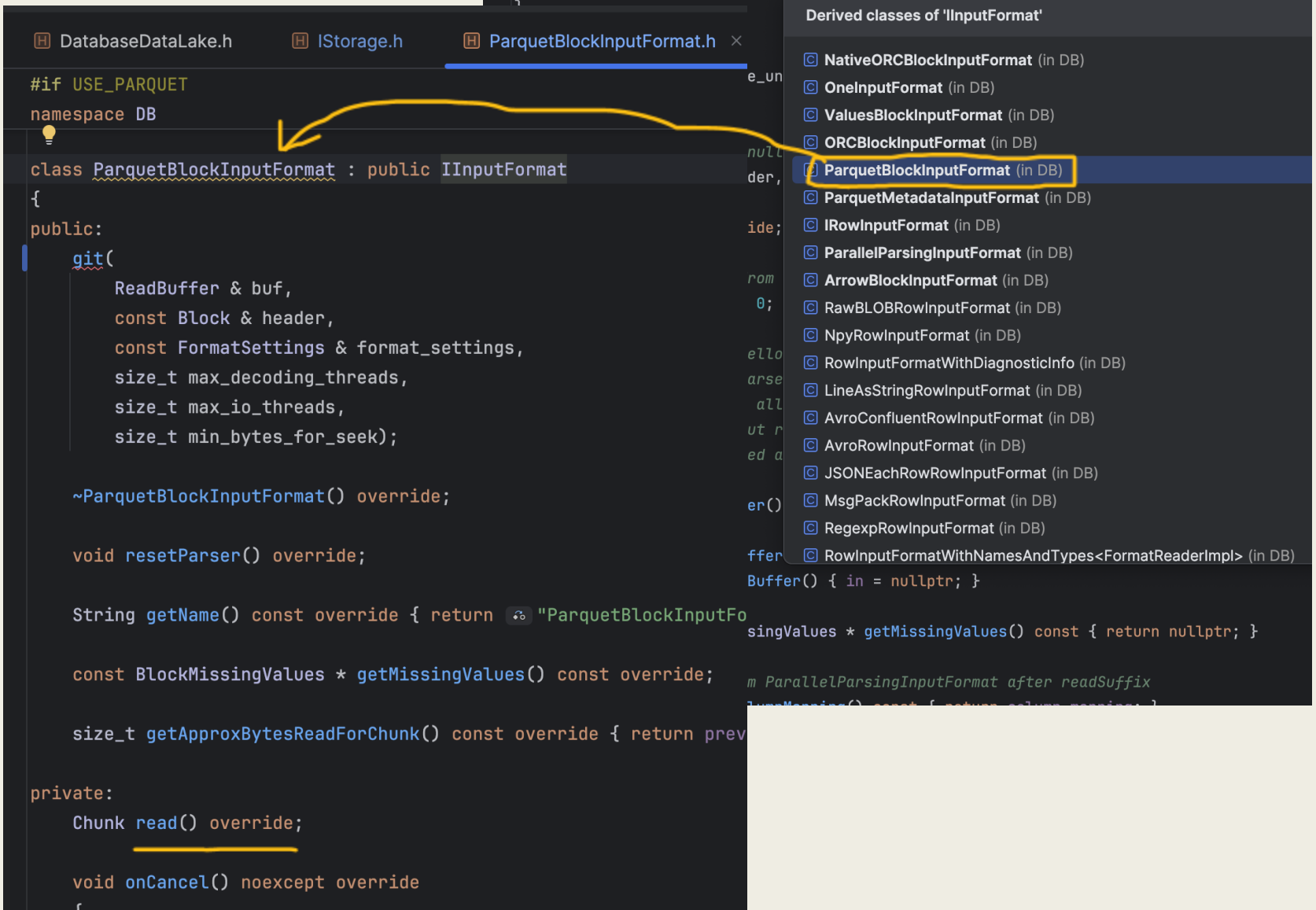
https://sql.clickhouse.com

Querying Foreign File Formats



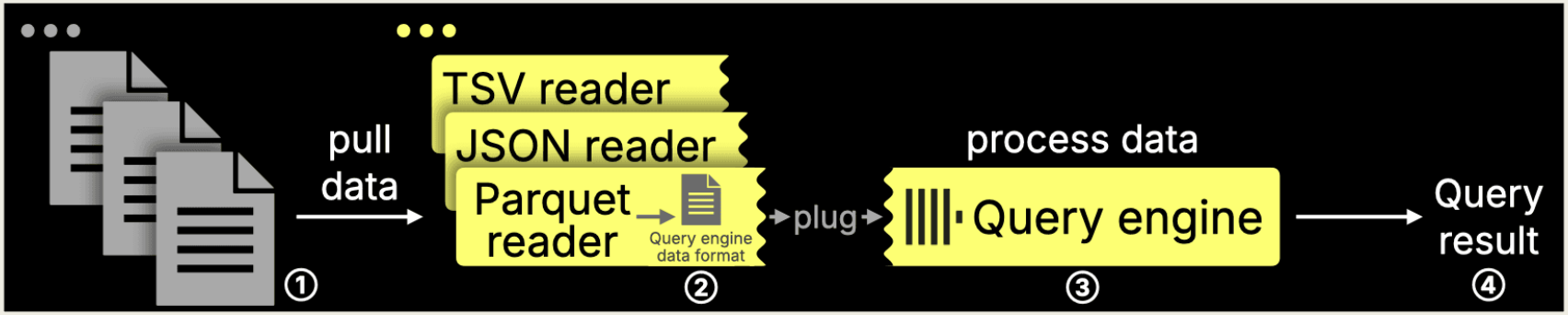
https://github.com/ClickHouse/ClickHouse/blob/master/src/Processors/Formats/IInputFormat.h

Parquet Reader



Reading non-native files

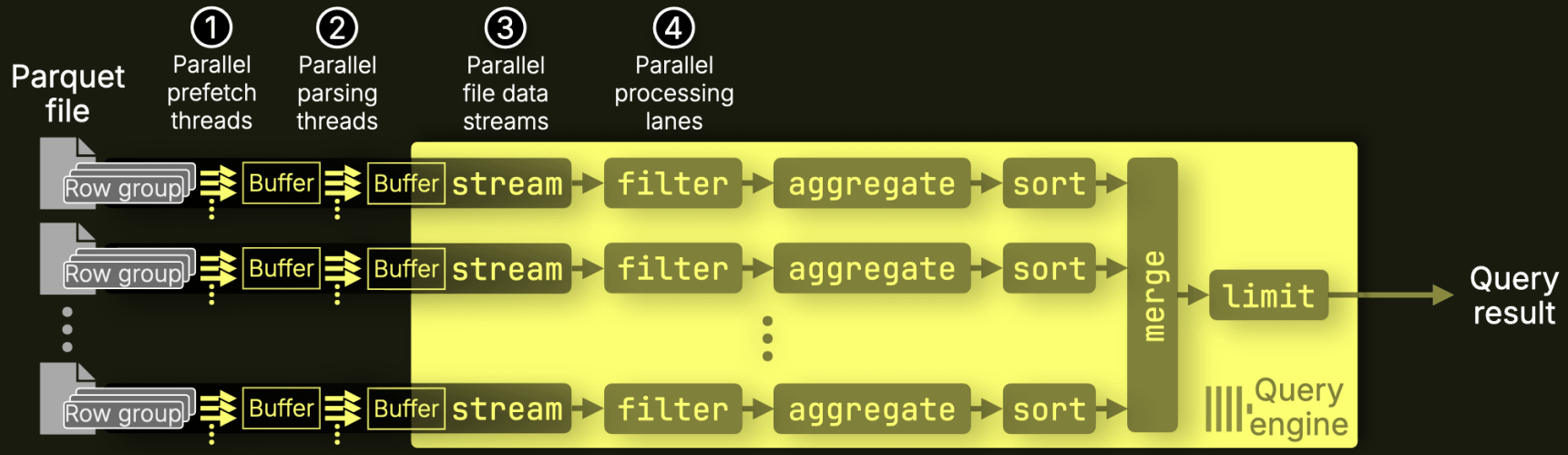
- 1: Format Reader reads and parses external files (eg., Parquet, CSV, JSON…)
- 2: Data converted into ClickHouse's internal columnar in-memory format
- 3: Query engine runs SQL operations (filter, join, aggregate) on in-memory data
- 4: Produces result set for client / downstream pipeline



https://clickhouse.com/blog/clickhouse-and-parquet-a-foundation-for-fast-

Performance with ParquetReader

- 1: Within a Parquet file, the Parquet file reader reads multiple row groups in parallel
- 2: Parsing threads read and parse data from multiple row groups within the same file in parallel
  - *If prefetching is active, they read from the prefetch buffer; otherwise, they read directly from the file*
- 3: Different Parquet files are processed concurrently, each with their own parallel prefetch and parsing threads, to maximize throughput across files
- 4: filtering, aggregation, and sorting happen across independent lanes for maximum concurrency



- Awesome that it is open & interoperable though! More room to improve performance
  - Brand new parquet reader loading at https://github.com/ClickHouse/ClickHouse/pull/78380

https://clickhouse.com/blog/clickhouse-and-parquet-a-foundation-for-fast-lakehouse-analytics

Query Data Anywhere

80+ built-in integrations with external systems

- S3
- GCS
- Azure Blob
- HDFS
- Kafka
- RabbitMQ
- ODBC/JDBC
- HIVE
- MySQL / PostgreSQL
- MongoDB
- Redis
- Elasticsearch
- ClickHouse-to-ClickHouse Distributed

Integrate the way you like

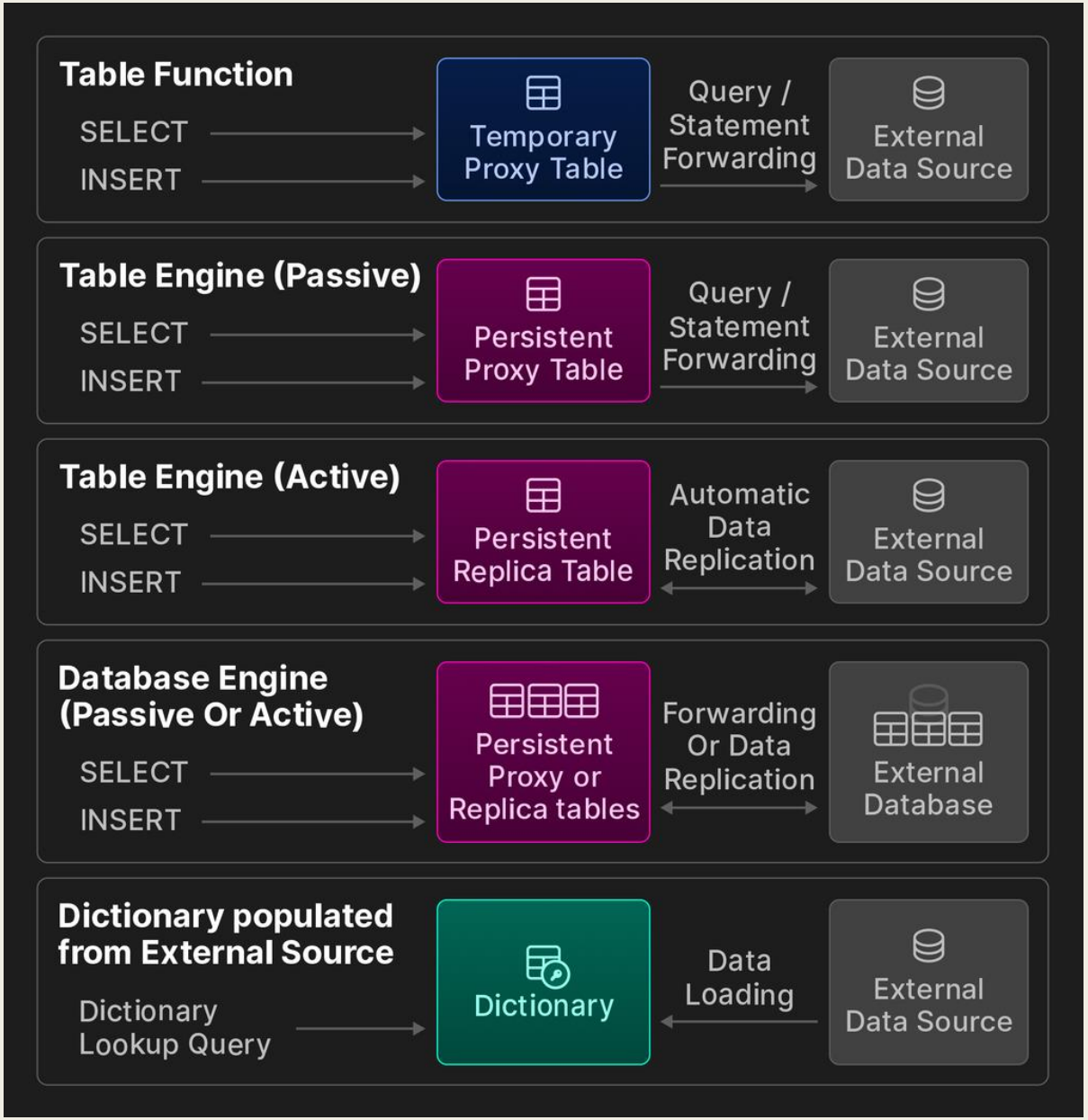
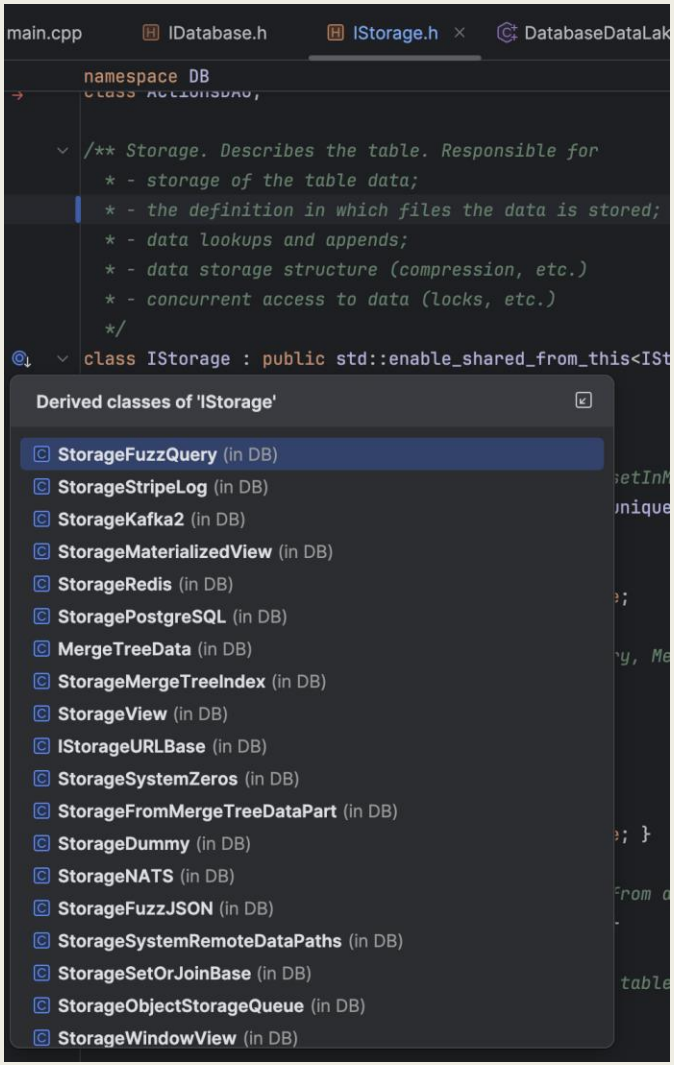


Table Engines

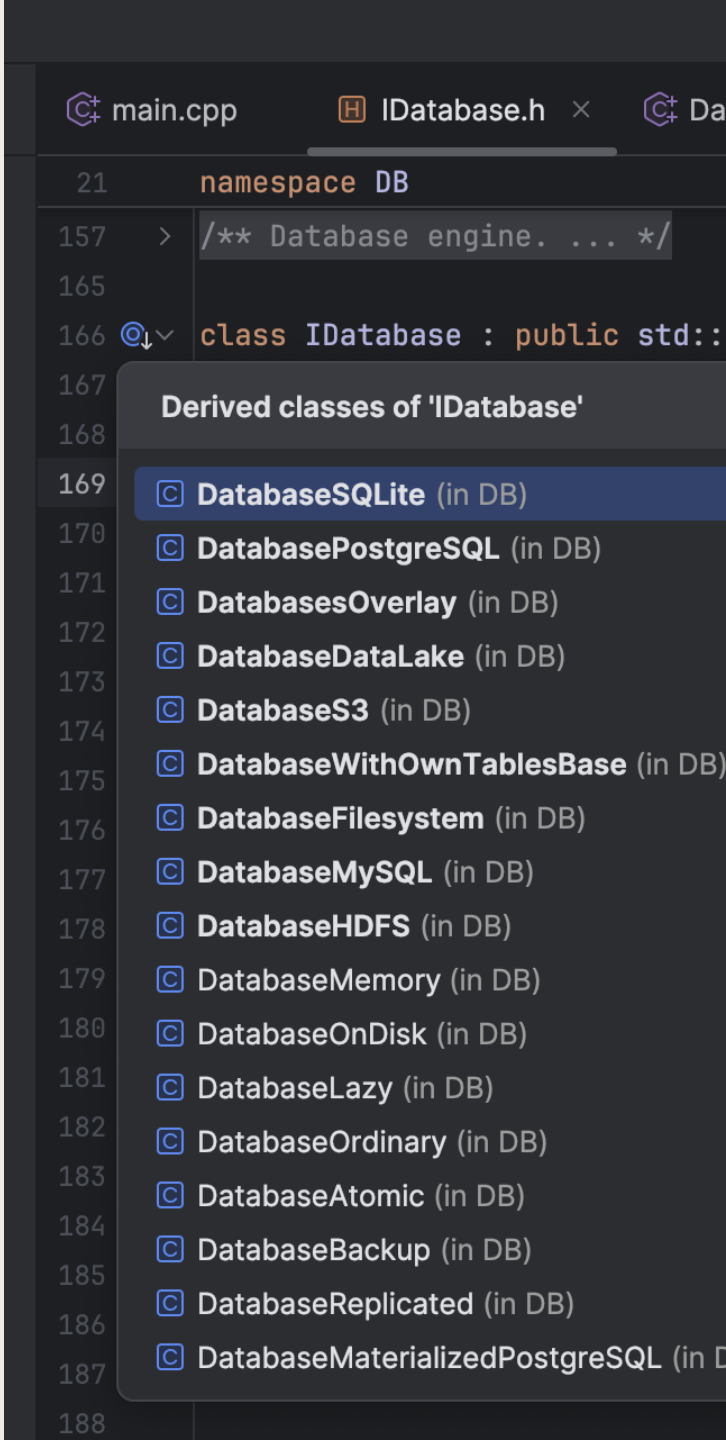
- **Istorage.h** : Interface for Table Engines
- Implementations => different table engine
  - *StorageMergeTree,*
  - *StoragePostgreSQL,*
  - *StorageKafka2*
  - *StorageObjectStorage (S3, AzureBlob, HDFS)*
  - *… 69 derived classes in total*
- Core responsibilities:
  - read(),
  - write(…) , alter(…), drop() etc



https://github.com/ClickHouse/ClickHouse/blob/master/src/Storages/IStorage.h

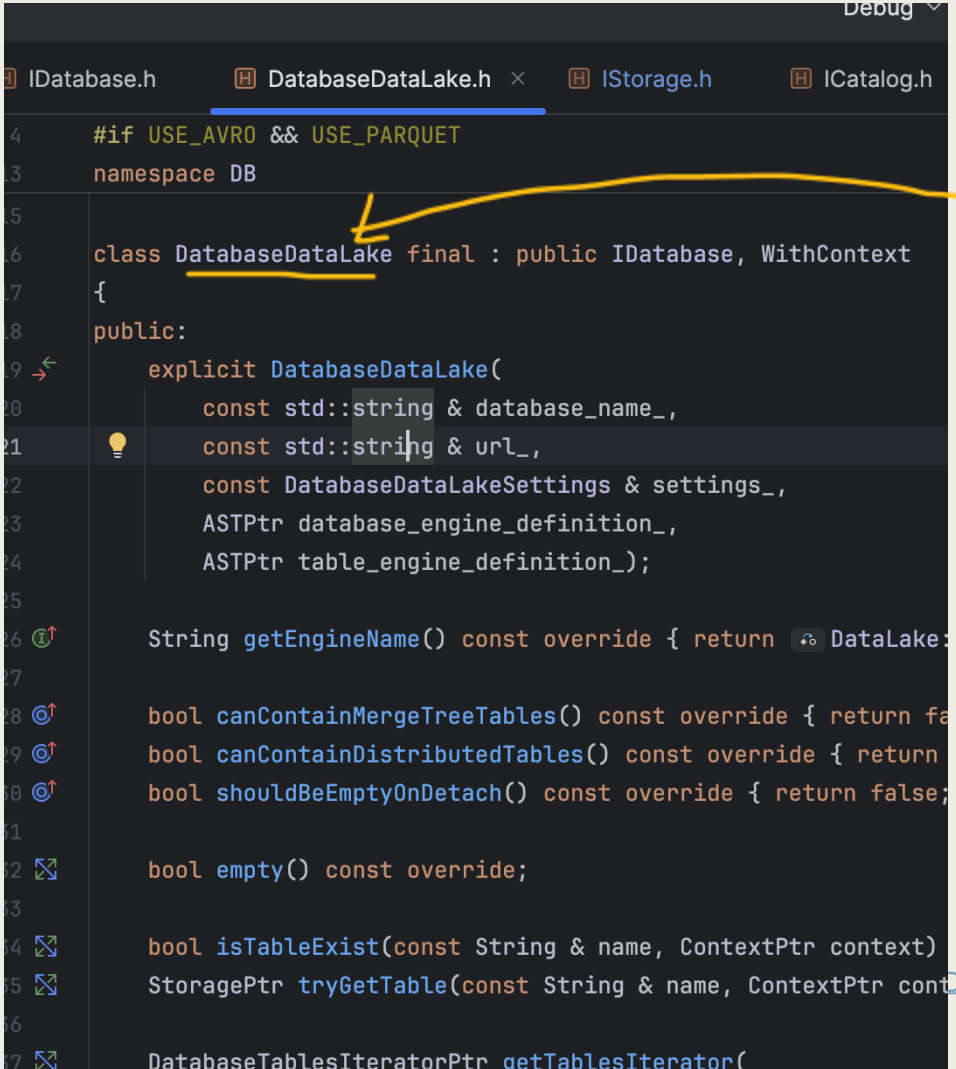
Database Engines

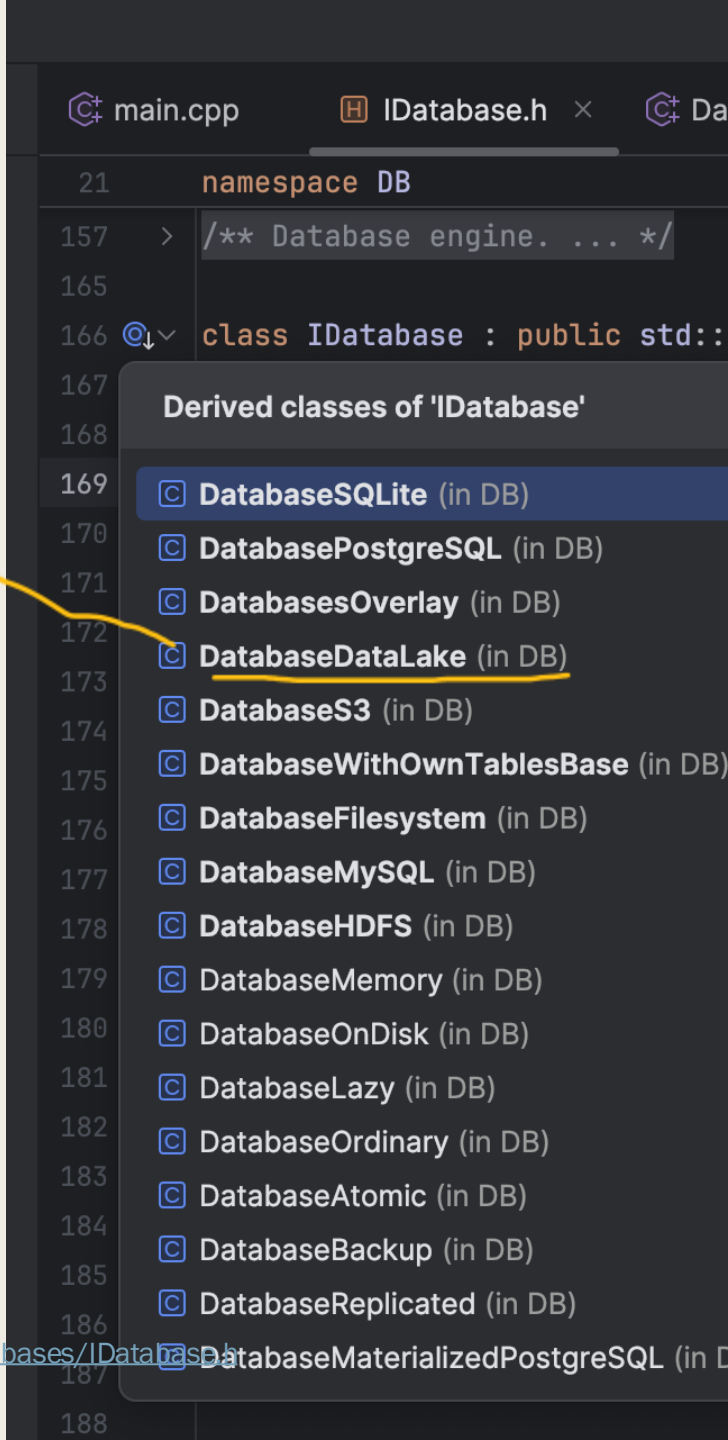
- IDatabase.h: interface for all **database engines.**
- Handles **collections of tables**
  - not individual rows or columns.
- Core Responsibilities (Table Management)
  - *createTable(), dropTable(), renameTable(),*
  - *attachTable(), detachTable(),*
  - *getTable(name),*
- *Query Support*
  - *getTablesIterator() — for queries like SHOW TABLES*
  - *getCreateTableQuery(),*
  - *getCreateDatabaseQuery()*



https://github.com/ClickHouse/ClickHouse/blob/master/src/Databases/IDatabase.h

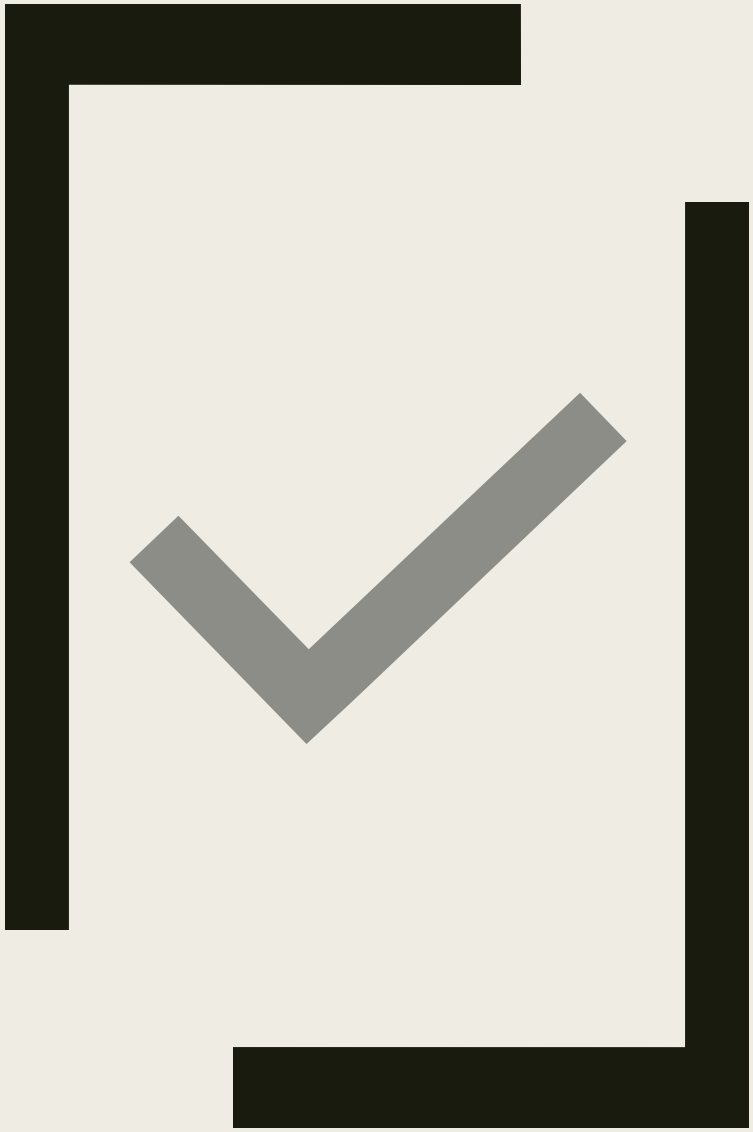
Datalake Engines





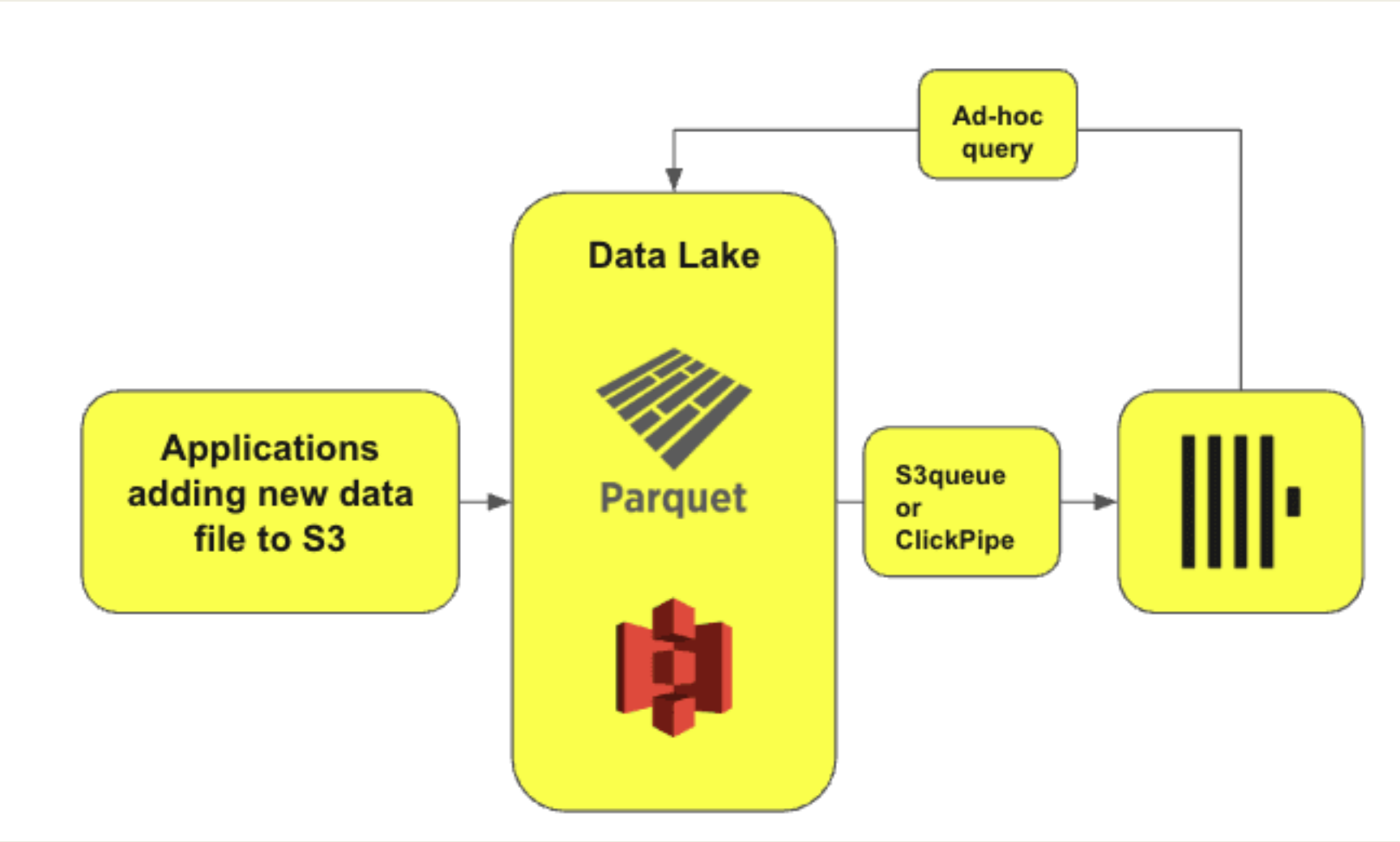
Why It Matters for Lakehouse

- If you have Parquet,
- if you have Iceberg,
- if you have data on object storage
  - *ClickHouse can query it*



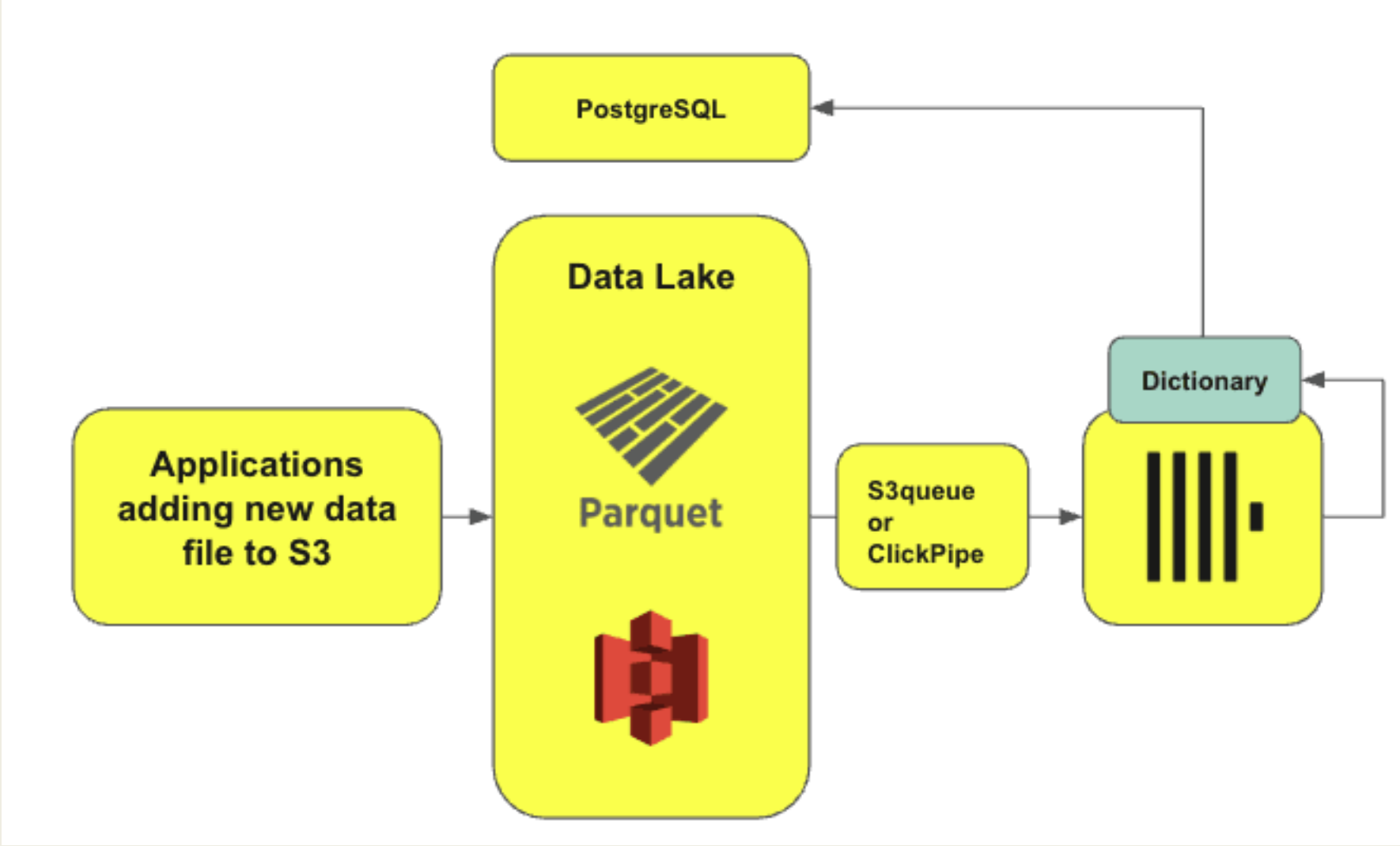
READ PARQUET & ICEBERG FROM CLICKHOUSE

Ad-hoc queries (Trino etal..)

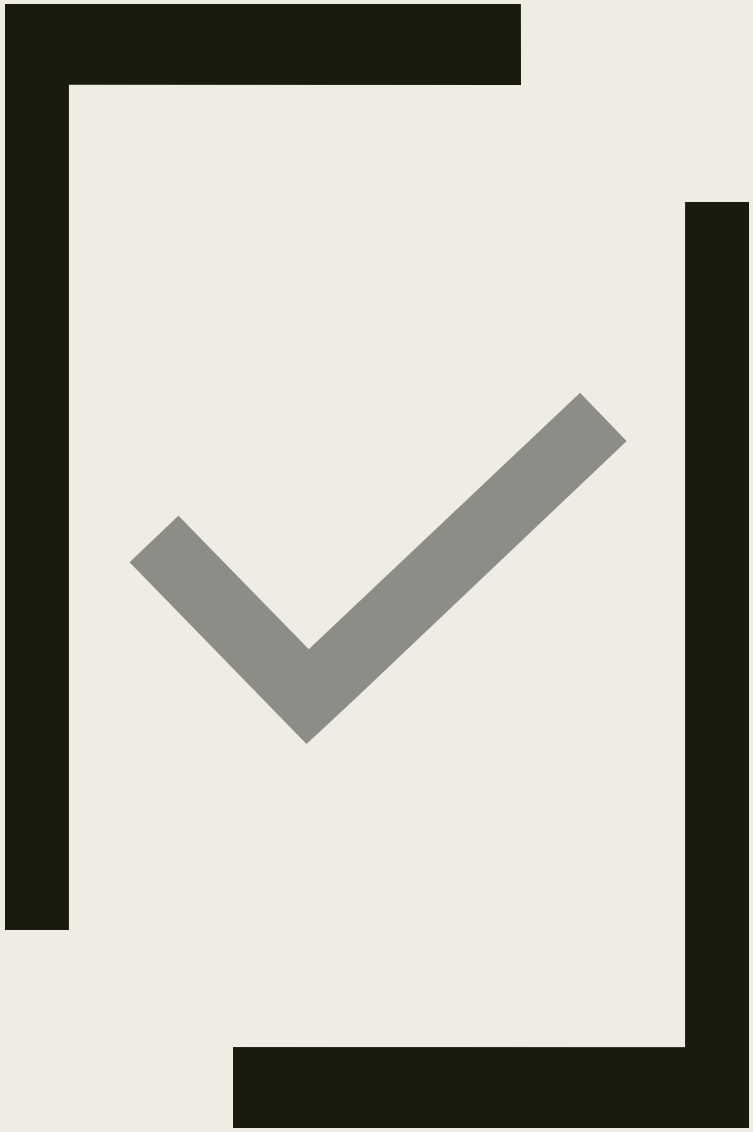


https://clickhouse.com/blog/climbing-the-iceberg-with-clickhouse

Querying multiple sources

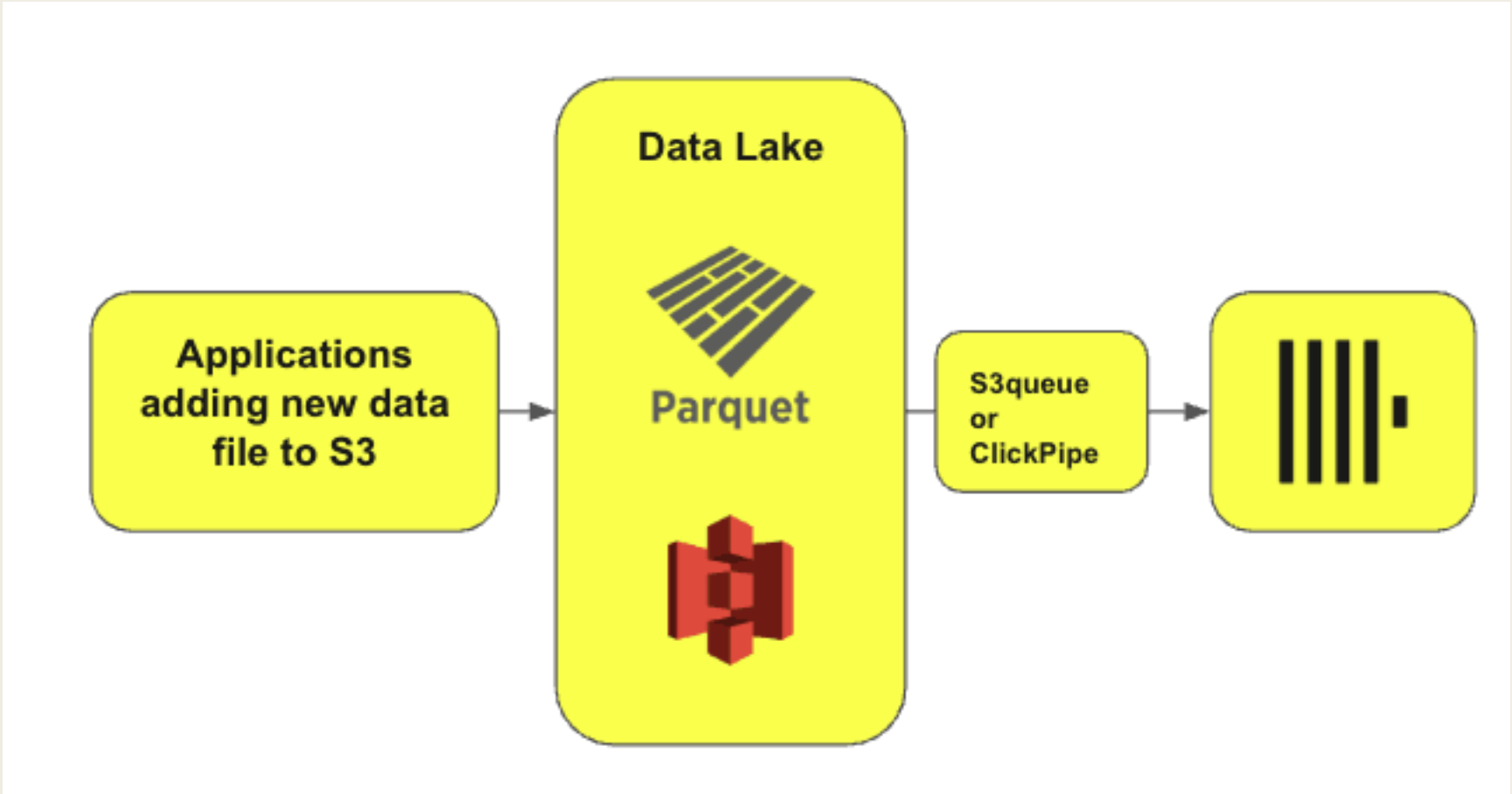


https://clickhouse.com/blog/climbing-the-iceberg-with-clickhouse



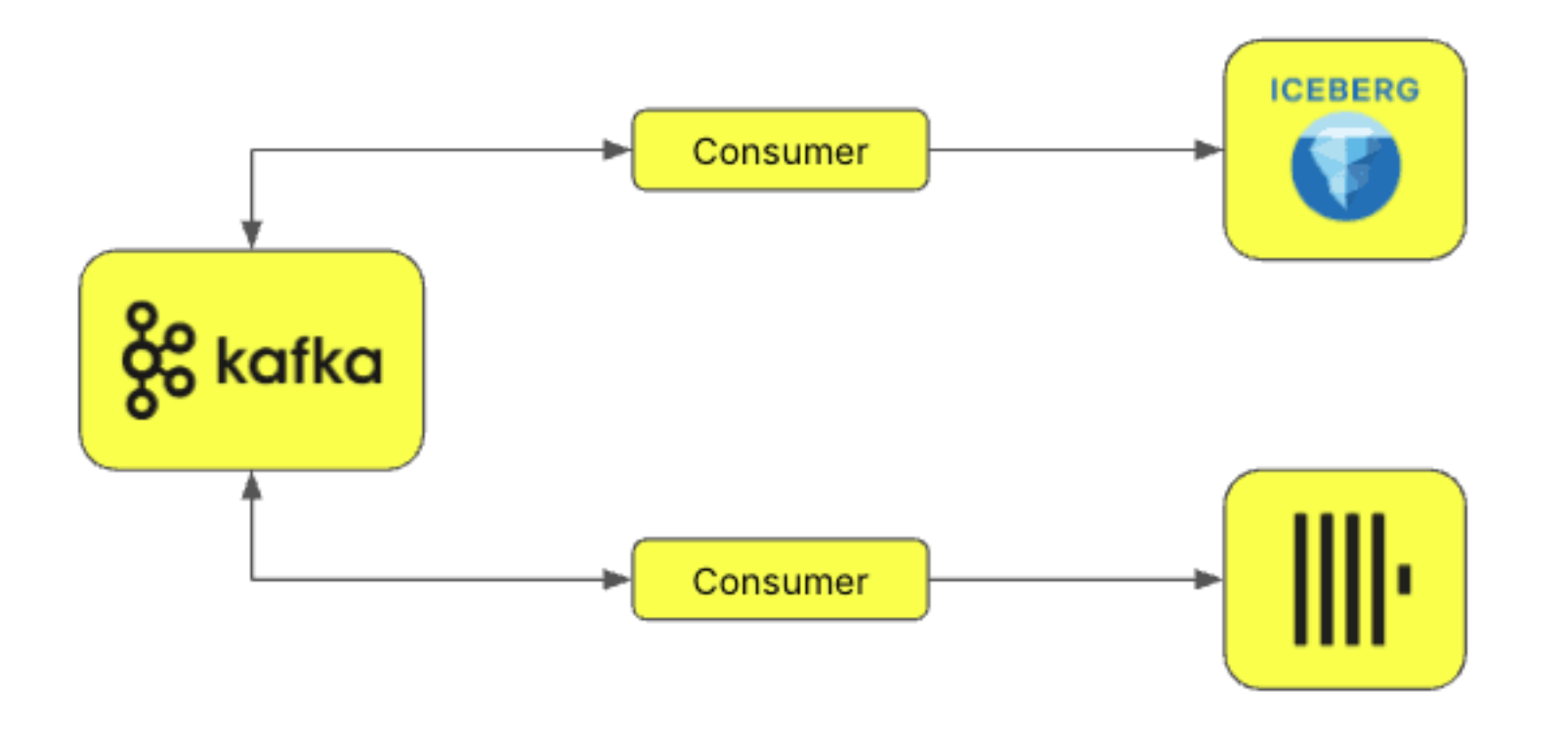
WRITE IN ICEBERG FORMAT

Writes(1): s3cluster/s3queue

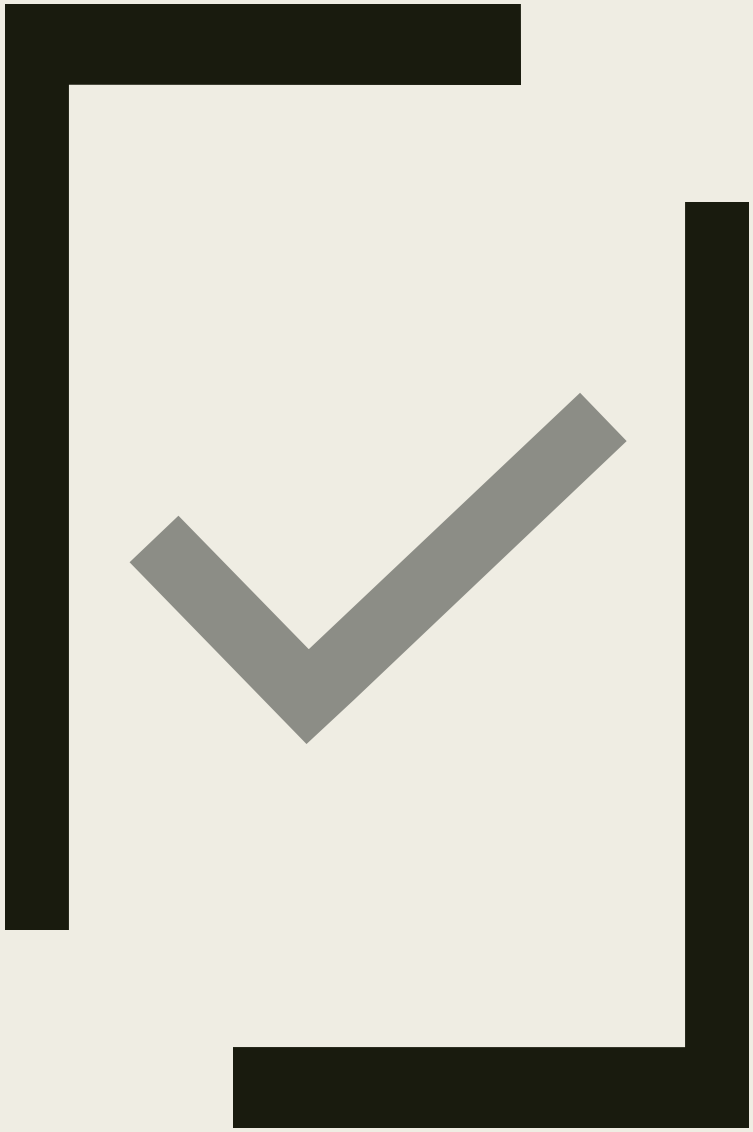


https://clickhouse.com/blog/climbing-the-iceberg-with-clickhouse

Writes(2): Double write/ttl/tiering

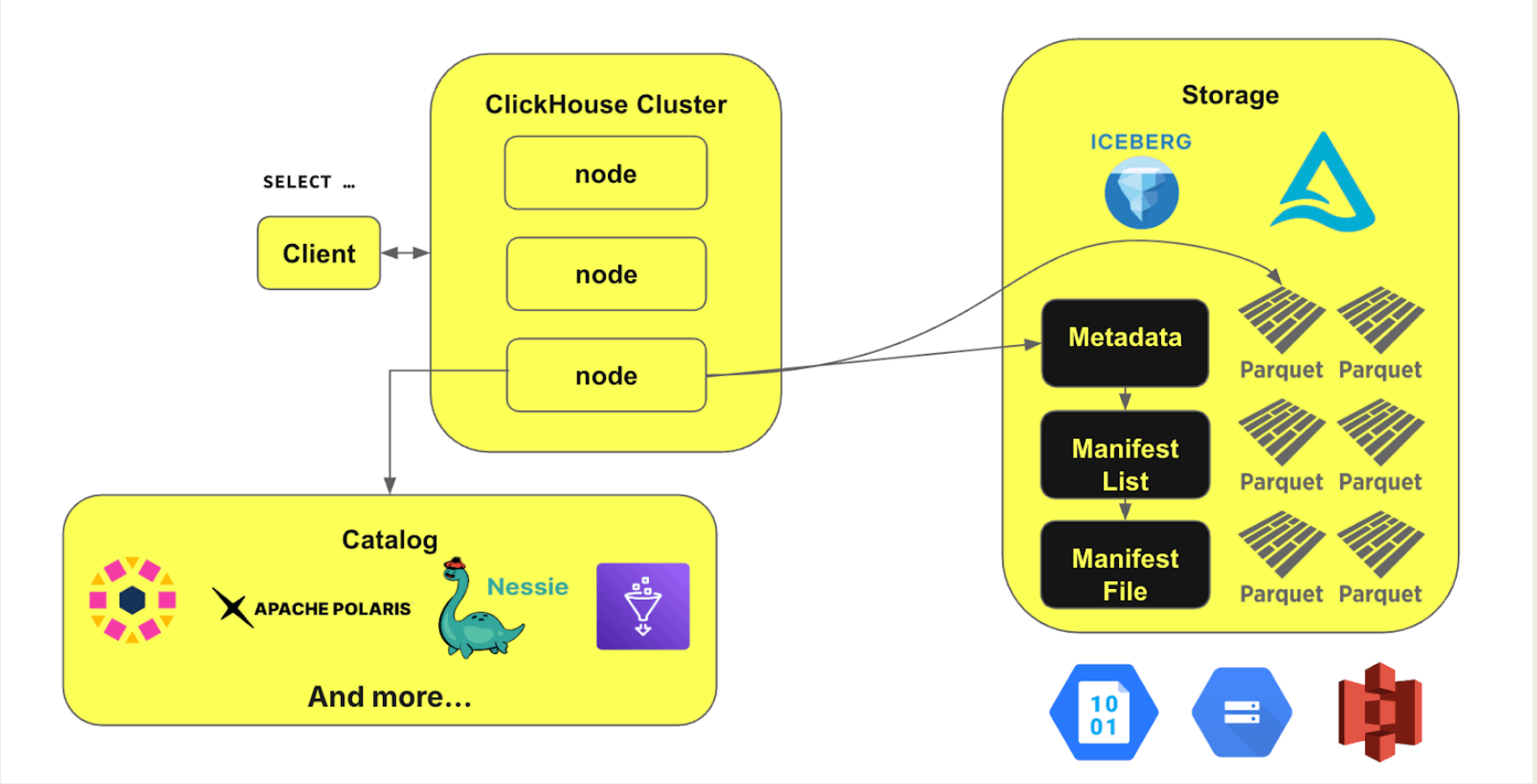


https://clickhouse.com/blog/climbing-the-iceberg-with-clickhouse



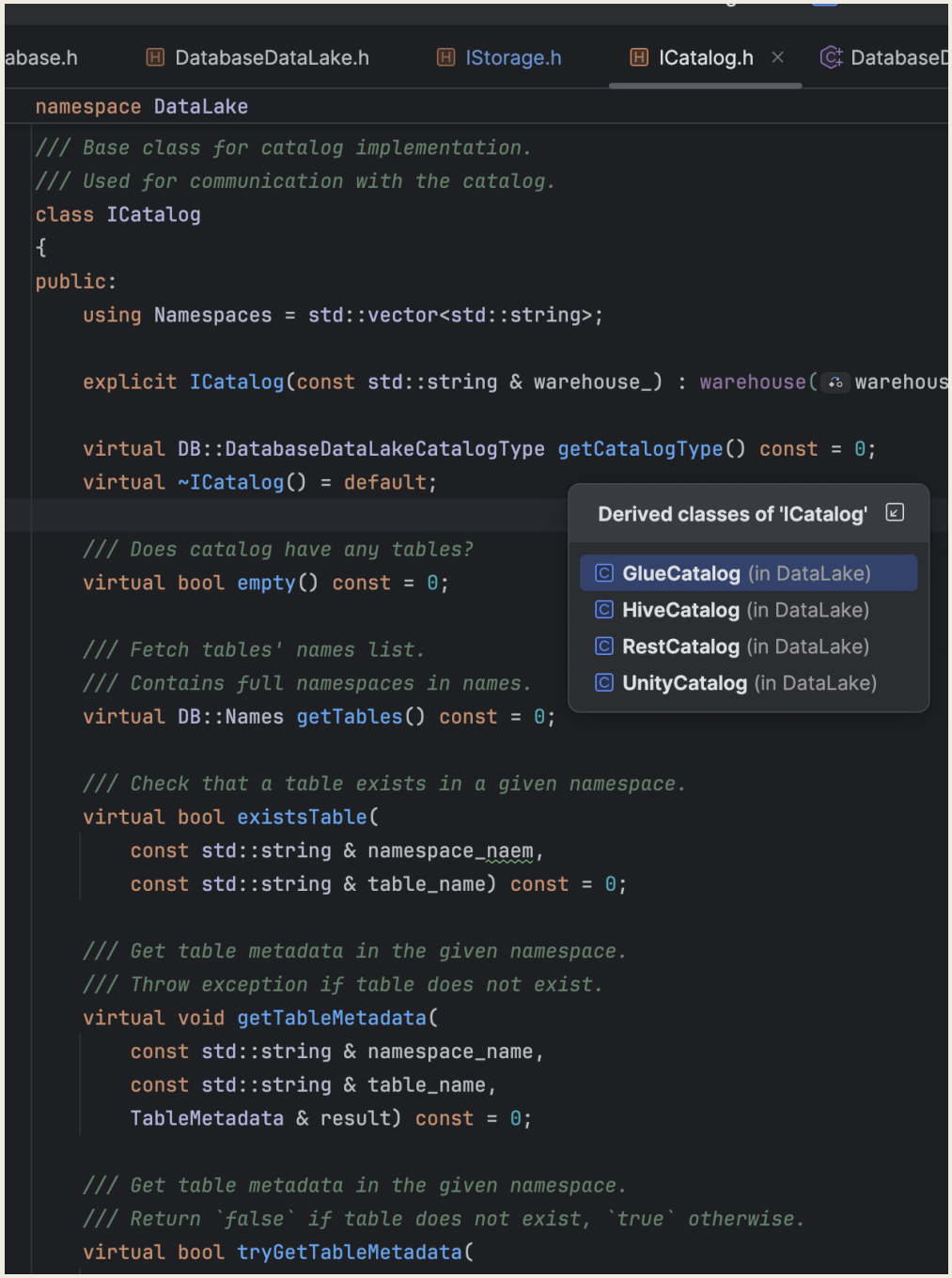
SUPPORT FOR OTHER ICEBERG GOODIES!

Catalog Integration

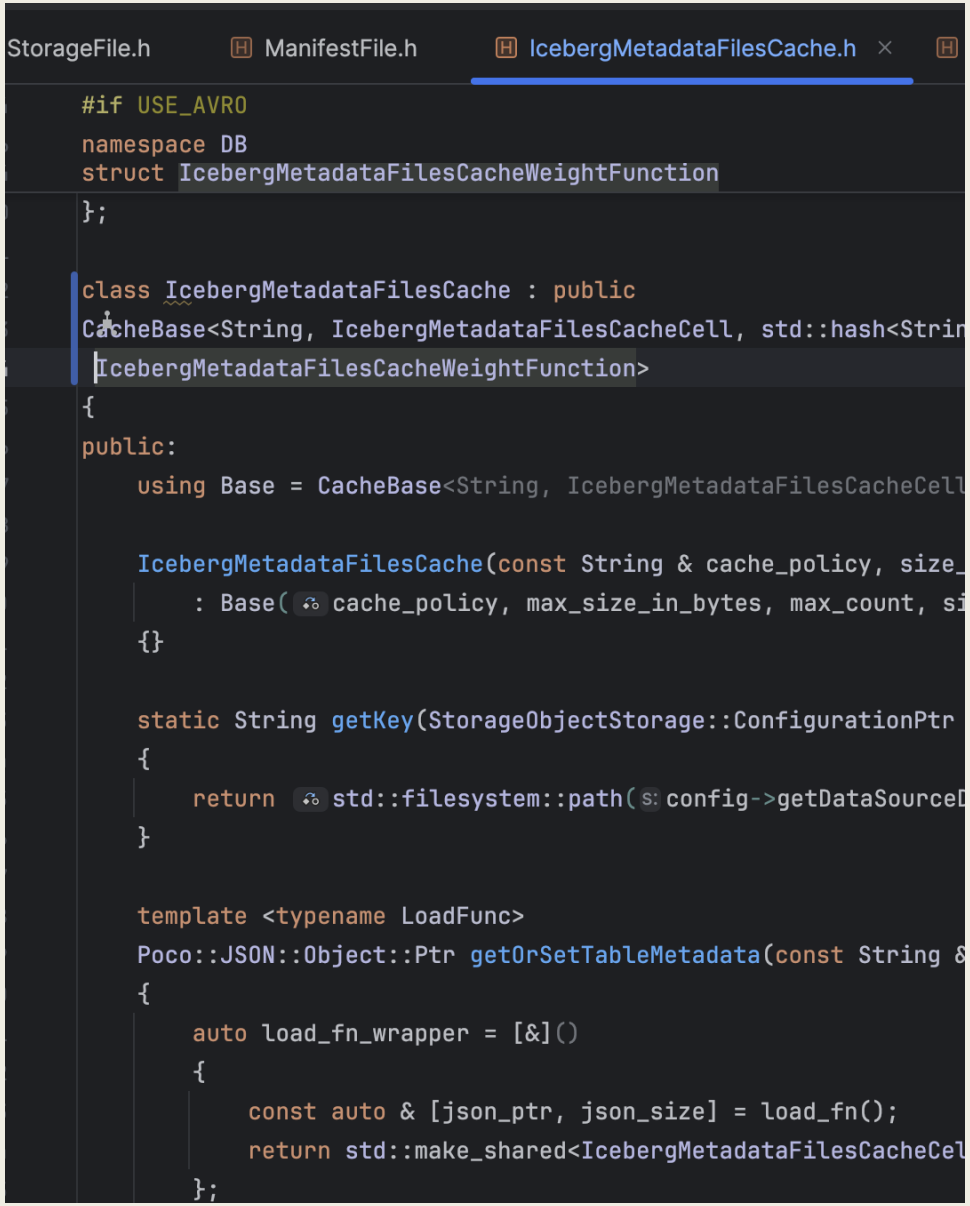


https://clickhouse.com/blog/climbing-the-iceberg-with-clickhouse

Catalog Integration



Metadata caching



What Works Today

- Experimental (still building…)
- Read-only queries, no writes
  - *In tests/Demo, we use spark for writes.*
- Catalog support (Glue, Hive, Unity, REST)
- Time Travel(PR), Partition Pruning(PR), Schema Evolution(PR)
- Metadata caching : in-memory (PR)

https://clickhouse.com/blog/climbing-the-iceberg-wi

THE ROAD AHEAD

thub.com/ClickHouse/ClickHouse/issues/74046

Roadmap 2025 #74046

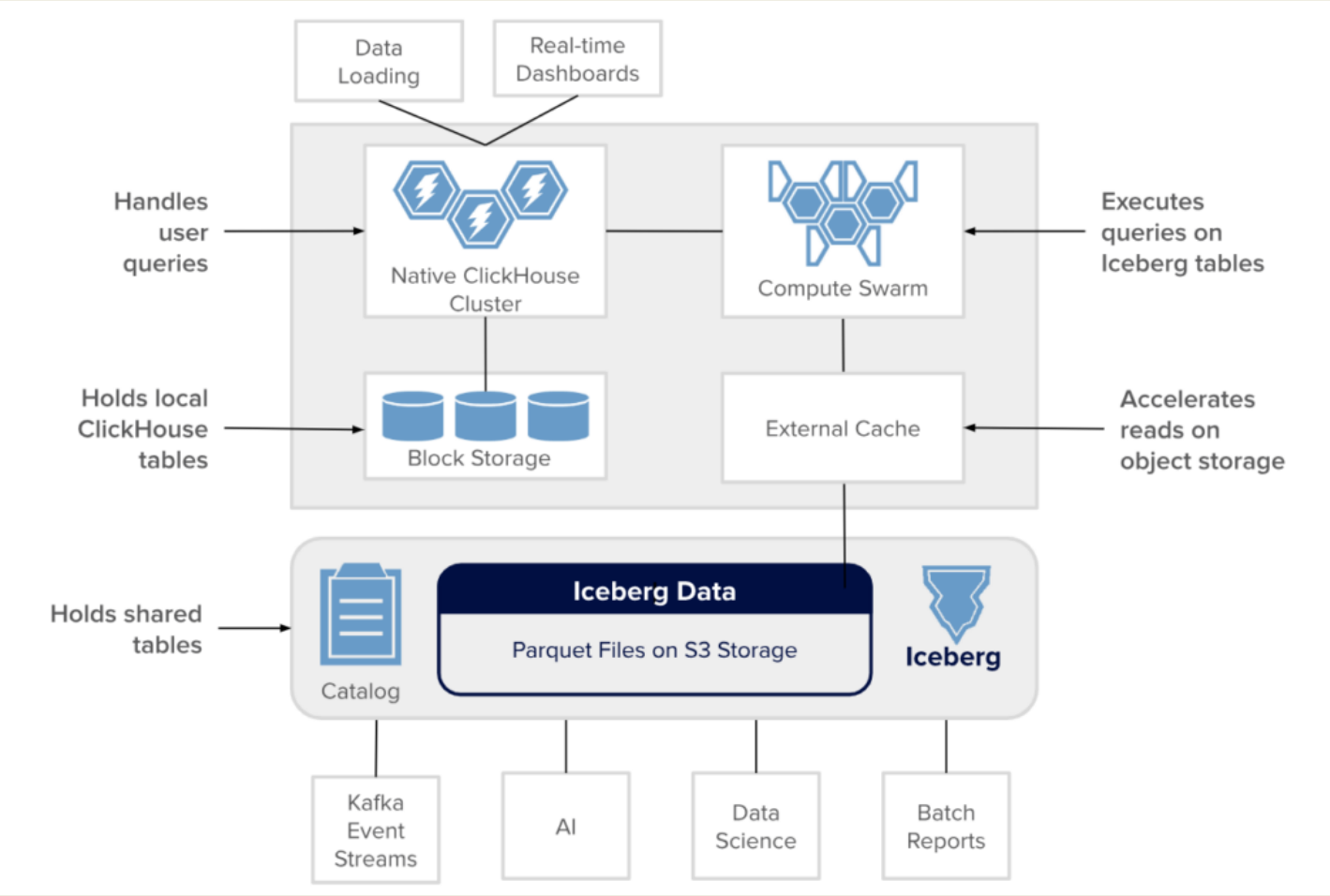
Data Lakes

- [x] Automatic use of cluster functions — Use cluster table functions automatically if parallel replicas are enabled #70659
- [ ] Parallel distributed INSERT SELECT by default
- [x] Enable Hive-style partitioning by default — Enable `use_hive_partitioning` by default #71636
- [ ] Uniform distribution of load across cluster — s3Cluster is suboptimal when the number of files is compa number of servers #70190
- [x] Partition pruning for Iceberg — Iceberg Partition Pruning for time-related partition transforms #72044
- [ ] Prewhere support for Parquet — [Feature] Add support for prewhere in parquet native reader #65527
- [ ] Unify different Parquet readers into one — A new parquet reader that supports filter push down, which i total time on clickbench by 50+% compared to arrow parquet reader #70611
- [x] Writing bloom filters for Parquet — Write Parquet bloom filters #71681
- [x] Time travel for Iceberg — Add setting to query Iceberg tables as of a specific timestamp #71072
- [ ] Support for deletions in Iceberg — Cannot read Iceberg table: positional and equality deletes are not supported #66588
- [x] In-memory metadata cache — A cache for data lake's metadata #71579
- [ ] Integration with Delta kernel — Integrate with delta_kernel #75255
- [x] Glue catalog support — Unity catalog integration #76988
- [x] Unity catalog support — Add glue catalog integration #77257
- [x] Support for Delta Lake on Azure — Support deltalake for AzureBlobStorage #74541
- [ ] Materialized views on top of data lakes
- [ ] Support arbitrary nesting in database and table names — Support arbitrary nesting in database and tabl names. #71171
- [ ] Writing into data lakes — Support for Writing to Apache Iceberg Tables in ClickHouse #49973
- [ ] Background merges for data lakes
- [ ] Subsets of a large file as a unit of work

PROJECT ANTALYA

1 more thing: Project Antalya

- Project Antalya Announcement Blog
- Example and documentation – github repo
- Antalya Open-source launch webinar (2025)
- Antalya Office Hours Q&A



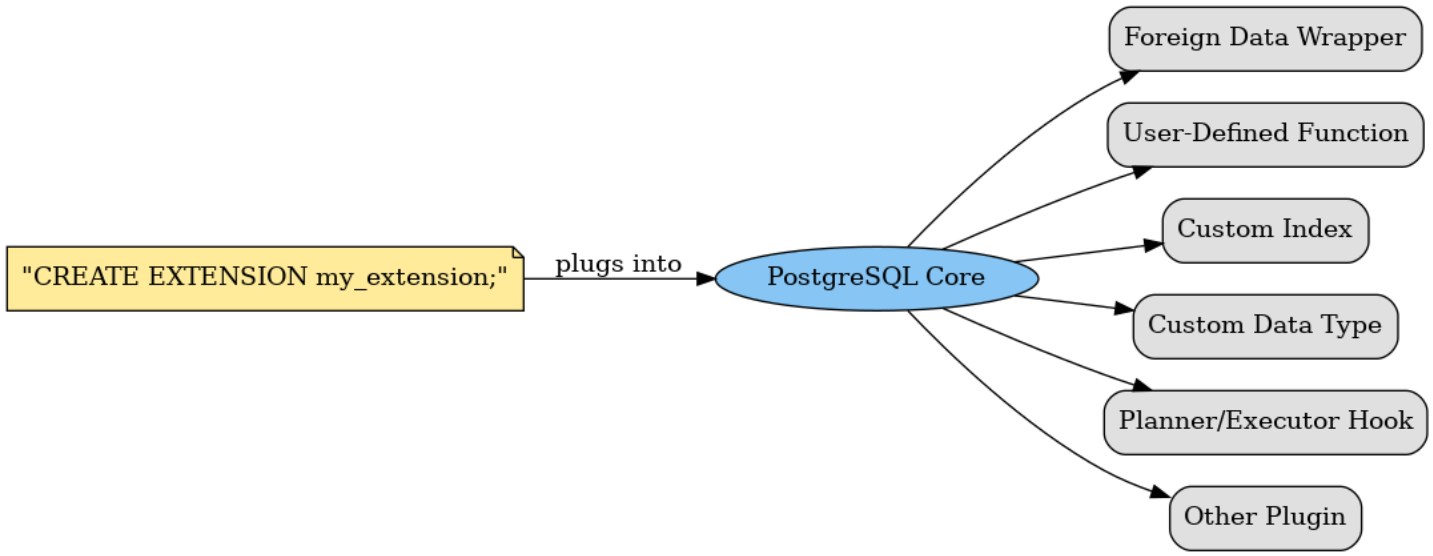
Live Demo

- Demo setup can be found here

HACKING POSTGRESQL FOR ICEBERG

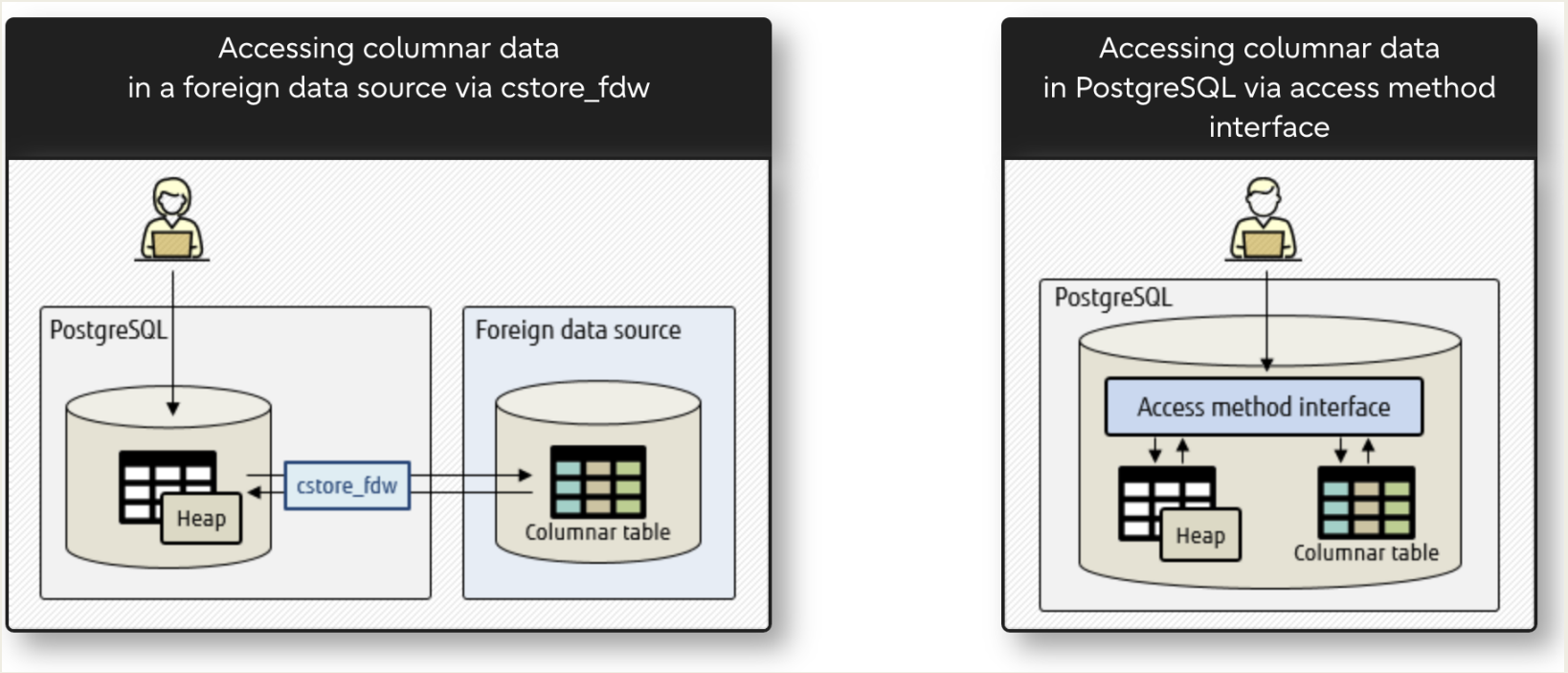
Built to extend

- Integration with ***Extensions***
  - *TimescaleDB (time-series),*
  - *Citus (distributed SQL),*
  - *PostGIS (geospatial data)*
  - ***pg_duckdb***



Extend even more!

- Foreign Data Wrappers (FDW)
  - *(parquet_fdw, clickhouse_fdw…)*
  - ***paradeDB's pg_analytics***
- Table Access Methods (TAM)
  - ***pg_mooncake (columnstore TAM)***
  - ***Hydra (columnar TAM)***



https://www.postgresql.fastware.com/blog/postgresql-table-access-methods

DuckDB inside Postgres, what?

- Embeddable, in-memory analytics engine
- Optimized for single-node execution
- Columnar format
- Vectorized processing
- Follows SQL Syntax Closely
- Flexible
  - *Can read (parquet / Iceberg) on S3*
  - *MotherDuck for DBaaS*

1. github.com/duckdb/pg_duckdb
2. github.com/hydradatabase/columnar
3. github.com/paradedb/pg_analytics
4. github.com/Mooncake-Labs/pg_mooncake

pg_duckdb == Postgres(PG) + DuckDB

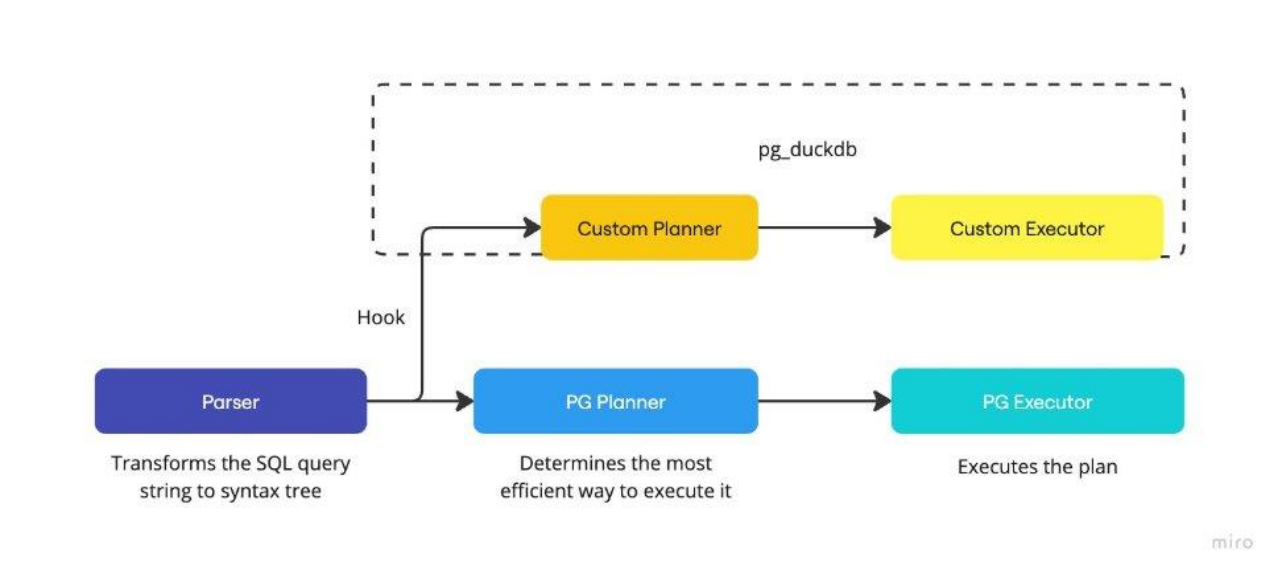
Data Lake Functions

Name	Description
`read_parquet`	Read a parquet file
`read_csv`	Read a CSV file
`read_json`	Read a JSON file
`iceberg_scan`	Read an Iceberg dataset
`iceberg_metadata`	Read Iceberg metadata
`iceberg_snapshots`	Read Iceberg snapshot information
`delta_scan`	Read a Delta dataset

https://github.com/duckdb/pg_duckdb/blob/main/docs/functions.md#data-lake-functions

LET’S GET OUR HANDS DIRTY, SHALL WE?

Query Stealing in pg_duckdb



- **planner_hook** to create a planned statement that DuckDb can understand and execute.
- **ExecutorStart_hook** and **ExecutorEnd_hook** used by pg_duckdb for snapshot internals in non SELECT queries.
- **ExplainOneQuery_hook** overrides the output of **EXPLAIN** query.

https://www.youtube.com/watch?v=i37nc2vQLdY

Hooks initialization and Query Stealing

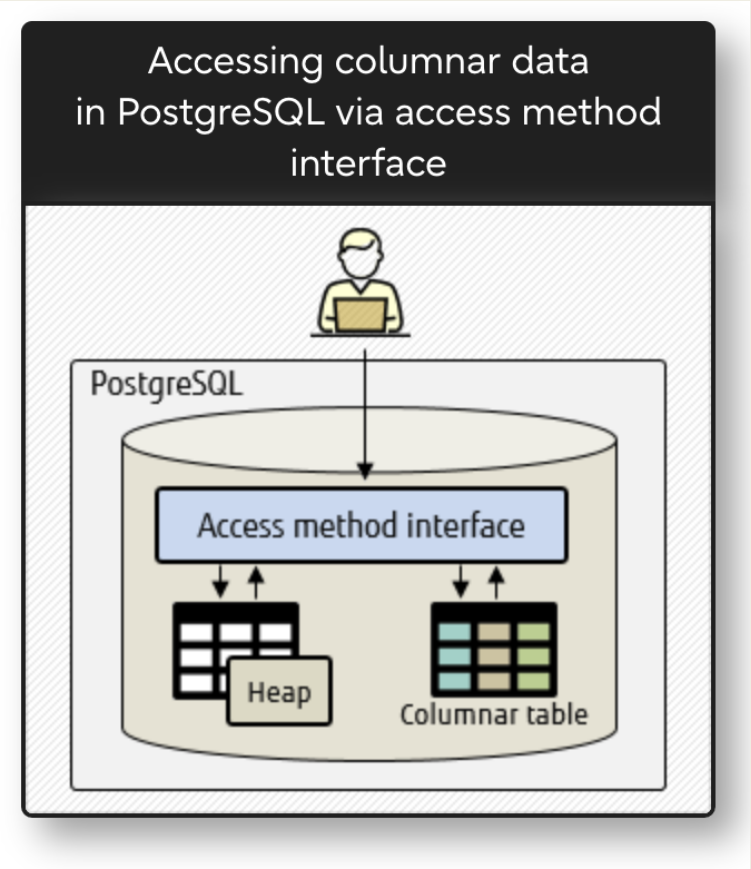
```
void DuckdbInitHooks(void) {
    prev_planner_hook = planner_hook;
    planner_hook = DuckdbPlannerHook;

    prev_executor_start_hook = ExecutorStart_hook ? Exe
    ExecutorStart_hook = DuckdbExecutorStartHook;

    prev_executor_finish_hook = ExecutorFinish_hook ? E
    ExecutorFinish_hook = DuckdbExecutorFinishHook;

    prev_explain_one_query_hook = ExplainOneQuery_hook
    ExplainOneQuery_hook = DuckdbExplainOneQueryHook;

    DuckdbInitUtilityHook();
}
```

DuckDbInitHooks

```
static PlannedStmt* DuckdbPlannerHook_Cpp(Query *parse
    const char *query_string, int cursor_options,
    ParamListInfo bound_params) {
    if (pgduckdb::IsExtensionRegistered()) {
        if (NeedsDuckdbExecution(parse)) {
            IsAllowedStatement(parse, true);
            return DuckdbPlanNode(parse, query_string,
        } else if (duckdb_force_execution && IsAllowed
            PlannedStmt *duckdbPlan =
            DuckdbPlanNode(parse, query_string, cursor
            if (duckdbPlan) {
                return duckdbPlan;
            }
            /* If we can't create a plan, we'll fall ba
        }
```

DuckDbPlannerHooks

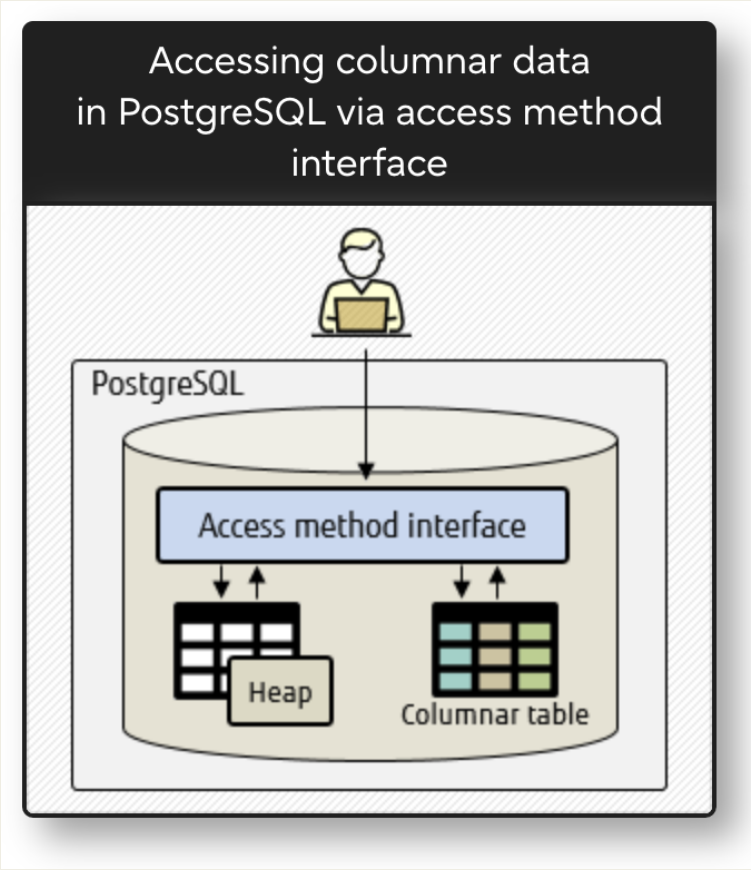
pg_mooncake (pg_duckdb++)



- New columnar storage behind TAM
- pg_mooncake supports loading data from
  - *Postgres heap tables,*
  - *Parquet, CSV, JSON files*
  - *Iceberg, Delta Lake tables*
- Write iceberg*/ delta tables with parquet
  - *On local disk*
  - *Or object store*

https://www.postgresql.fastware.com/blog/postgresql-table-access-methods

pg_mooncake (pg_duckdb++)



1. Enable the extension

```
CREATE EXTENSION pg_mooncake;
```

2. Create a columnstore table:

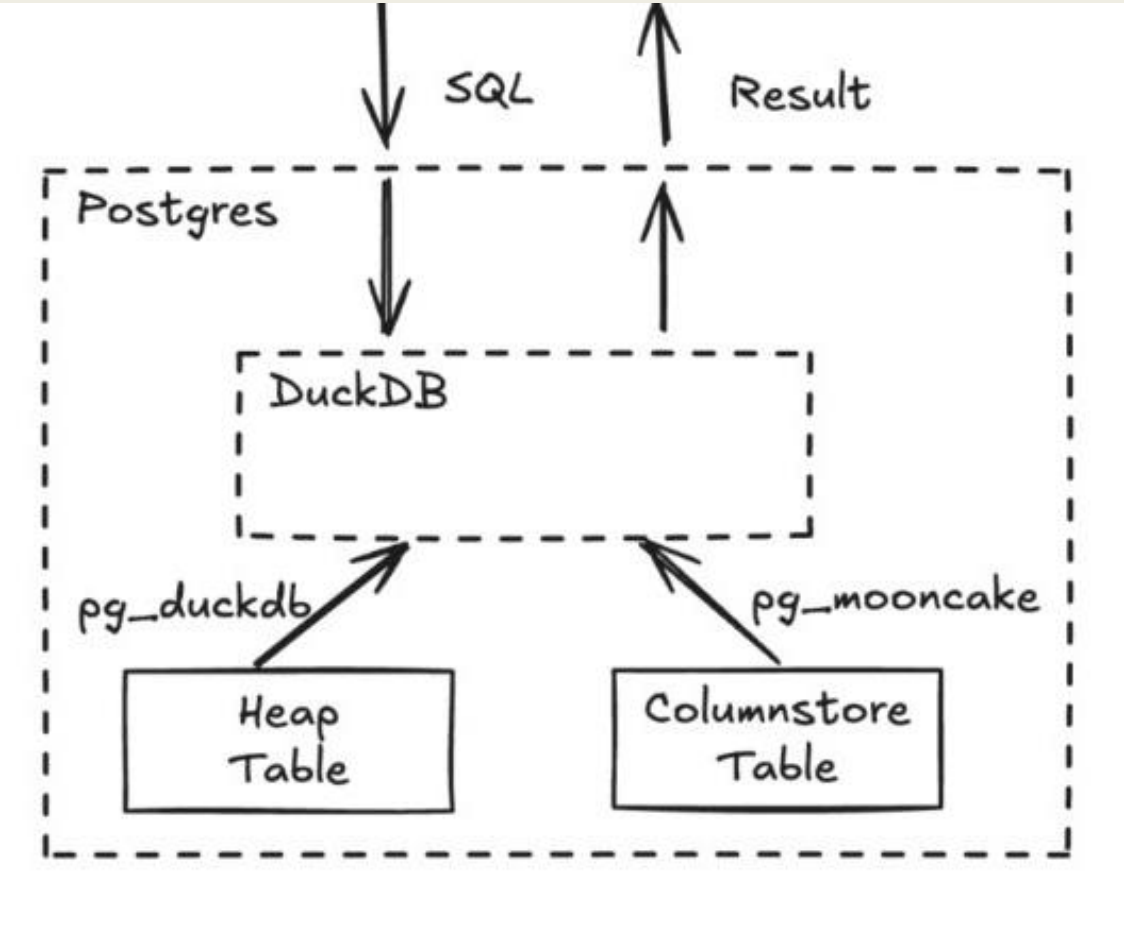
```
CREATE TABLE user_activity(
  user_id BIGINT,
  activity_type TEXT,
  activity_timestamp TIMESTAMP,
  duration INT
) USING columnstore;
```

3. Insert data:

```
INSERT INTO user_activity VALUES
  (1, 'login', '2024-01-01 08:00:00', 120),
  (2, 'page_view', '2024-01-01 08:05:00', 30),
  (3, 'logout', '2024-01-01 08:30:00', 60),
  (4, 'error', '2024-01-01 08:13:00', 60);

SELECT * from user_activity;
```

https://www.postgresql.fastware.com/blog/postgresql-table-access-methods

Query execution flow in pg_mooncake



https://www.mooncake.dev/blog/how-we-built-pgmooncake

Anatomy of a PostgreSQL Extension

Extensions are modular add-ons that can introduce new capabilities to vanilla PostgreSQL server.

- SQL file:
  - *Executes SQL commands to create objects (functions, types, operators, tables) inside PostgreSQL's system catalogs.*
- Control file:
  - *Declares extension metadata such as version, description, and installation behavior. Used by the extension loader to locate and manage the extension lifecycle.*
- Source code:
  - *Dynamically loaded at runtime to register native functions or hooks. User functions can be written in other languages like rust as well.*

PostgreSQL Hooks

- **Global Function pointers** exposed by the backend.
- Extensions can **intercept and modify internal behavior** at various stages of query processing.
- No core code patch needed!!
- Extensions assign their functions to the hook during `_PG_init() and remove them using _PG_fini()`
- `planner_hook` allows extensions to intercept and modify the query planning phase, enabling custom plan trees or query routing.
- **`Executor Hooks`** lets extensions inject logic at the various stages of execution (ExecutorStart_hook, ExecutorFinish_hook), often used for custom data writes or result handling.
- ExplainOneQuery_hook allows overriding the default **EXPLAIN** procedure for a single query.

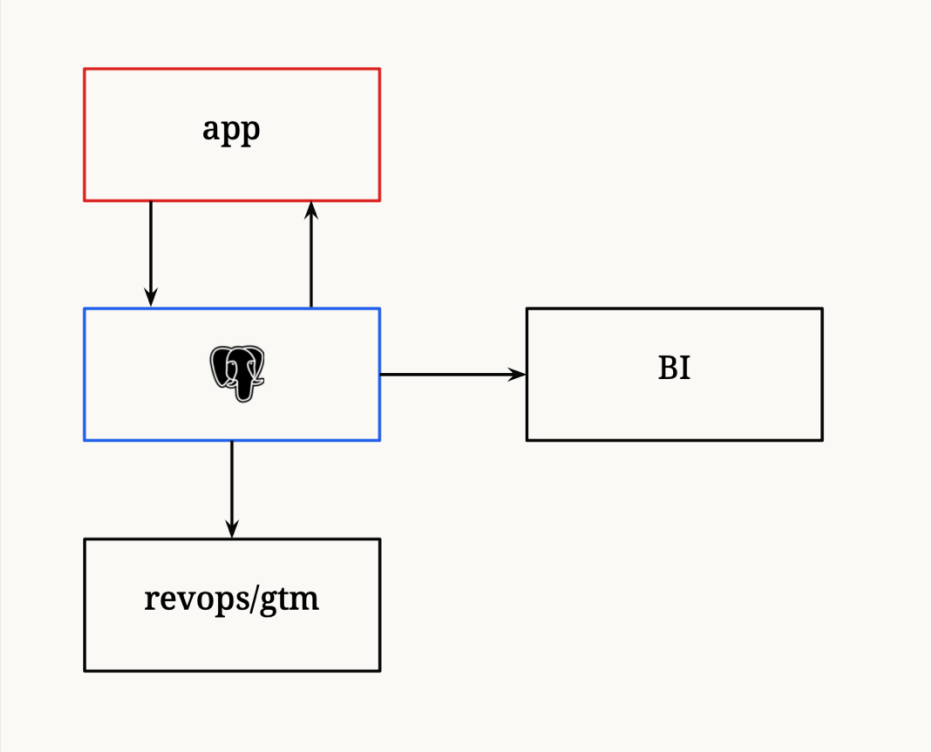
Query stealing

```
static PlannedStmt* DuckdbPlannerHook_Cpp(Query *parse,
    const char *query_string, int cursor_options,
    ParamListInfo bound_params) {
    if (pgduckdb::IsExtensionRegistered()) {
        if (NeedsDuckdbExecution(parse)) {
            IsAllowedStatement(parse, true);
            return DuckdbPlanNode(parse, query_string, cursor_options, bound_params, true);
        } else if (duckdb_force_execution && IsAllowedStatement(parse)) {
            PlannedStmt *duckdbPlan =
            DuckdbPlanNode(parse, query_string, cursor_options, bound_params, false);
            if (duckdbPlan) {
                return duckdbPlan;
            }
            /* If we can't create a plan, we'll fall back to Postgres */
        }
```

DuckdbPlannerHook

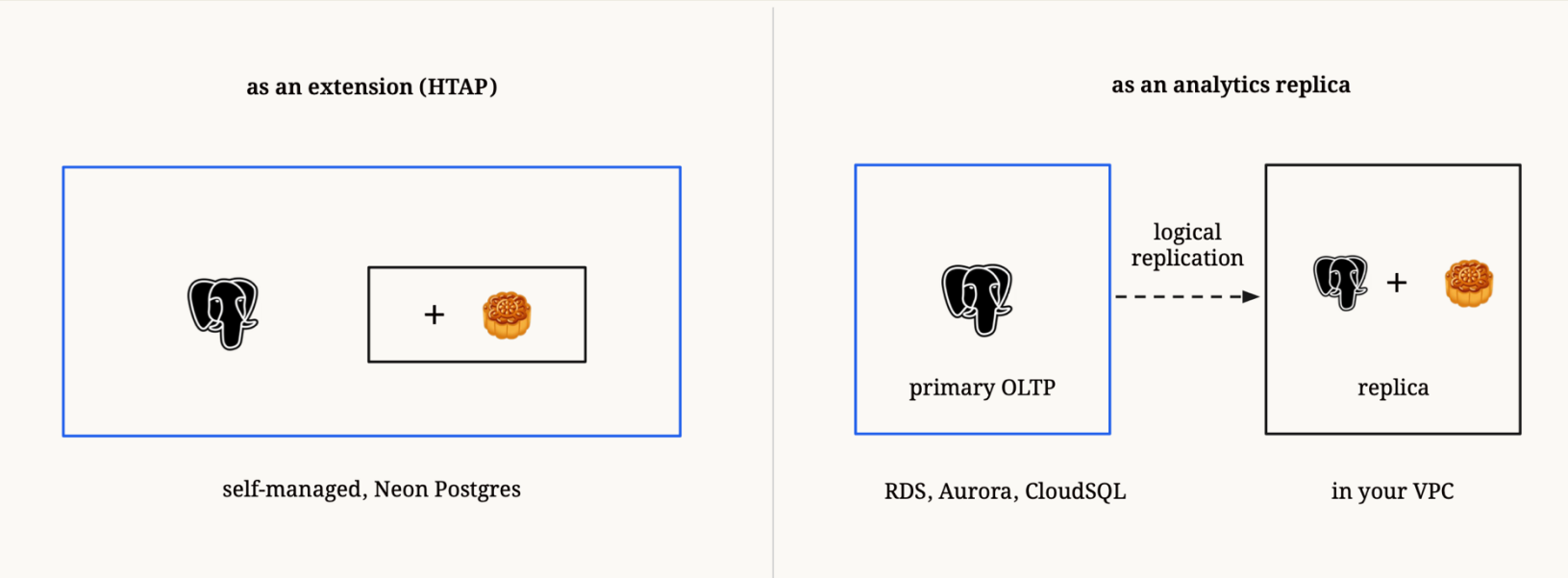
HOW DO I USE IT? LOADING.. ☺

Operations : day1



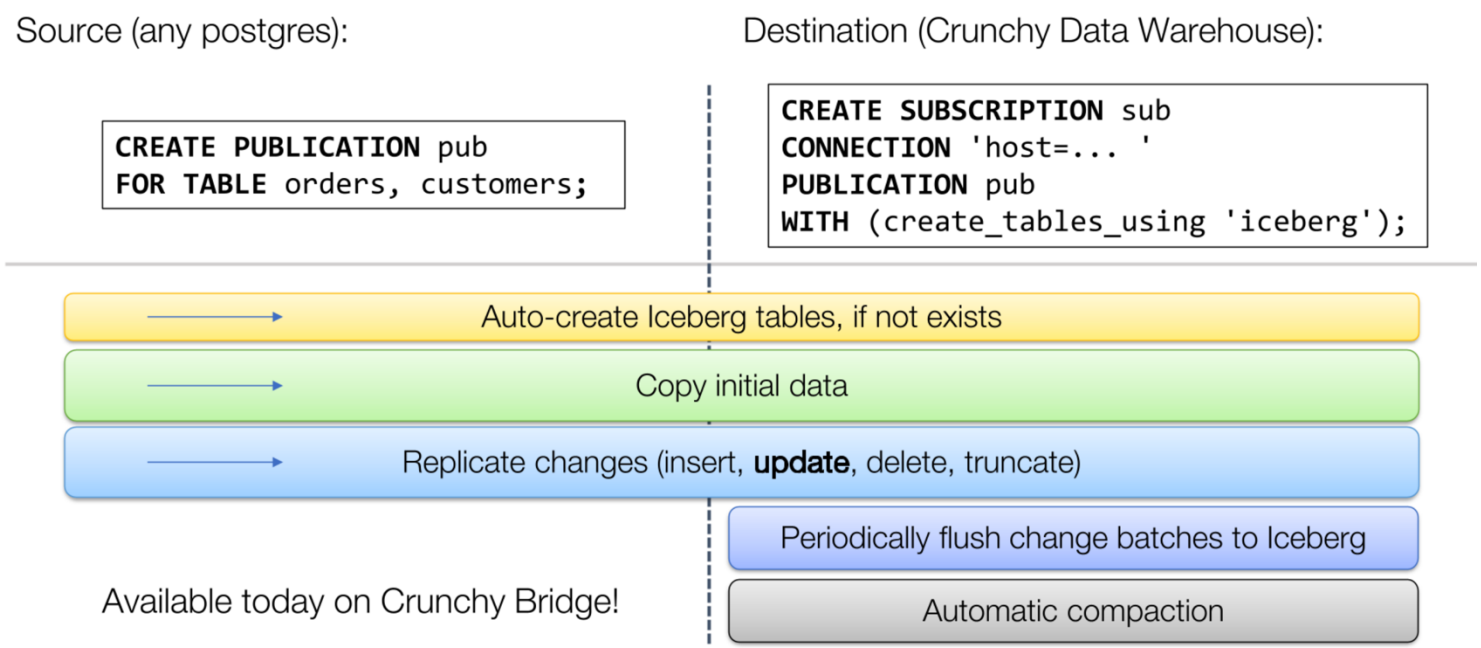
Source: +1 to your Postgres by Zhou Sun at DataCouncil 2025

Operations: Day 2 & 3



Source: +1 to your Postgres by Zhou Sun at DataCouncil 2025

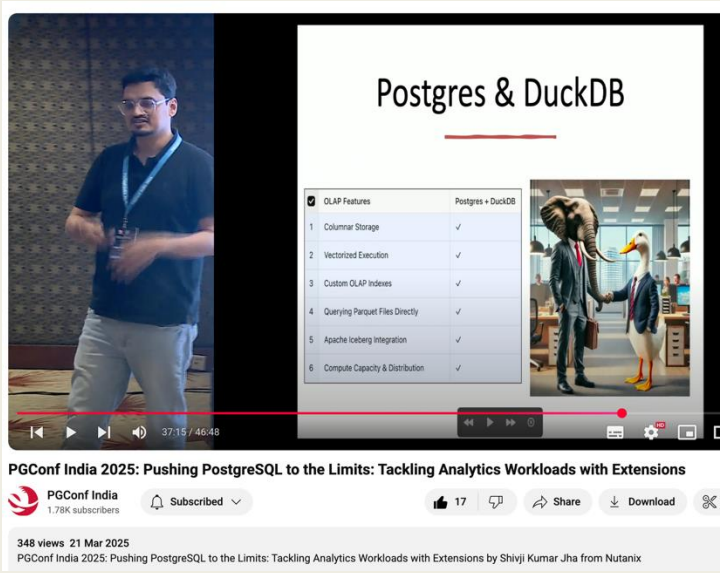
Logical replication from Postgres to Iceberg



Source: Converging Database Architectures by Marco Slot at DataCouncil 2025

Reading further!

- See my PGConf India talk for a deep dive (right)
- Signals that Iceberg is
  - *no longer just for distributed compute engines*
  - *travelling far and wide…*



tinyurl.com/shiv-pgconf-talk

tinyurl.com/shiv-pgcong-slide

Live Demo

- Demo setup can be found here

WRAPPING UP

Why This Matters

- Performance (native format) + openness (Iceberg)
  - *Can we stop dreaming of data migrations finally?*
- If your queries are spread across apps
  - *Committed to current query language*
  - *Maybe you just want to do a PoC first?*
- Great time for builders to engage early!
  - *If you just love hacking open-source code early ☺*
  - *Much easier to become a contributor **now**!*

THANK YOU!

Q&A

github.com/shiv4289/shiv-tech-talks/

linkedin.com/in/shivjijha

tinyurl.com/ch-lake-slides