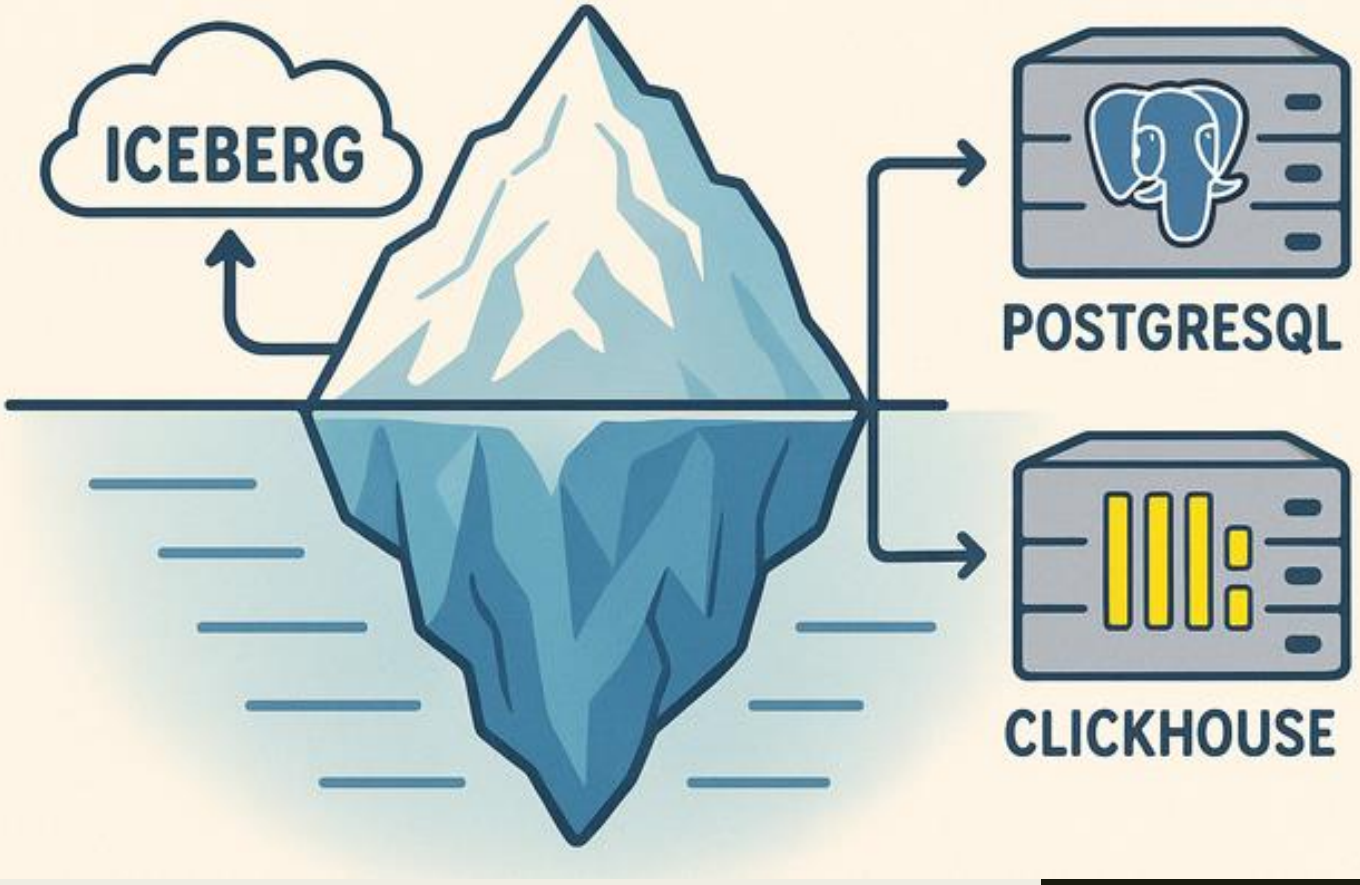
RUNNING ICEBERG ON EXISTING DBS



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
- Mostly active 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

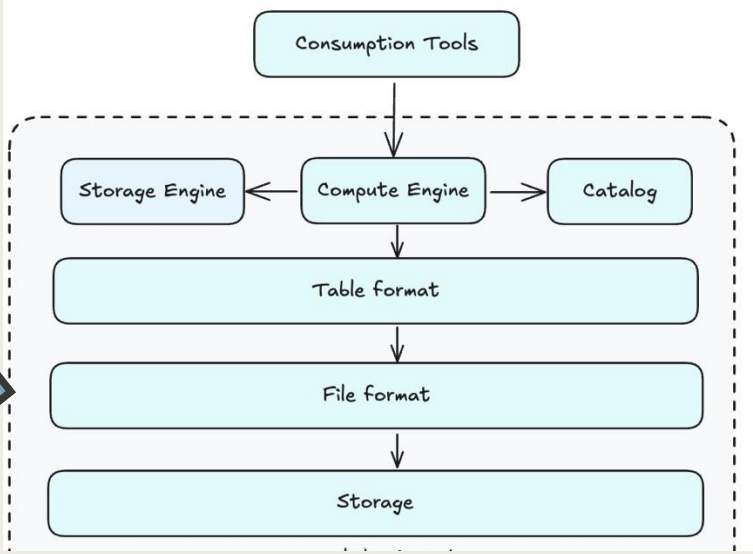
Setting the Context

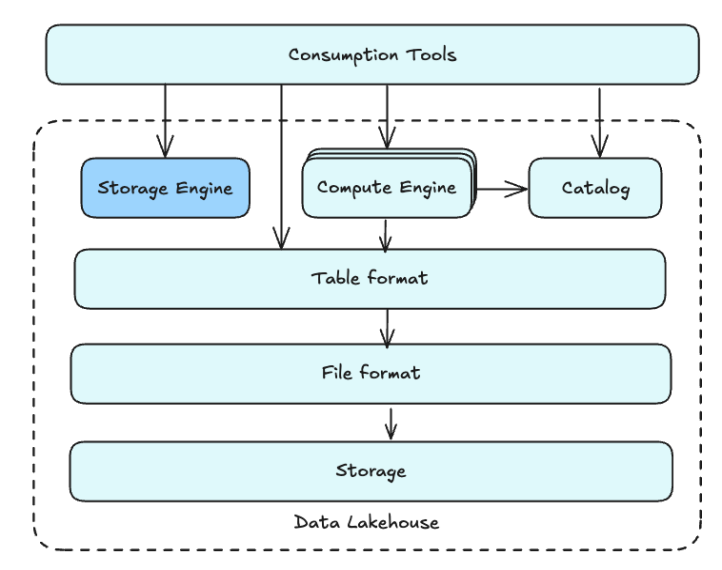
- Open table formats like Iceberg are becoming foundational in modern data lakes
- Traditionally used with engines like Spark, Trino, Dremio etc
- Even stream-native platforms like Kafka (TableFlow), Pulsar (Ursa), and RisingWave are adopting Iceberg
- AWS S3 Tables saw a lot eyeballs
- Adoption signals 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…)
- Decouples storage and compute
- Schema evolution, time travel, partition pruning, ACID
- Increasing support across the ecosystem
- A lot of mindshare at the moment!

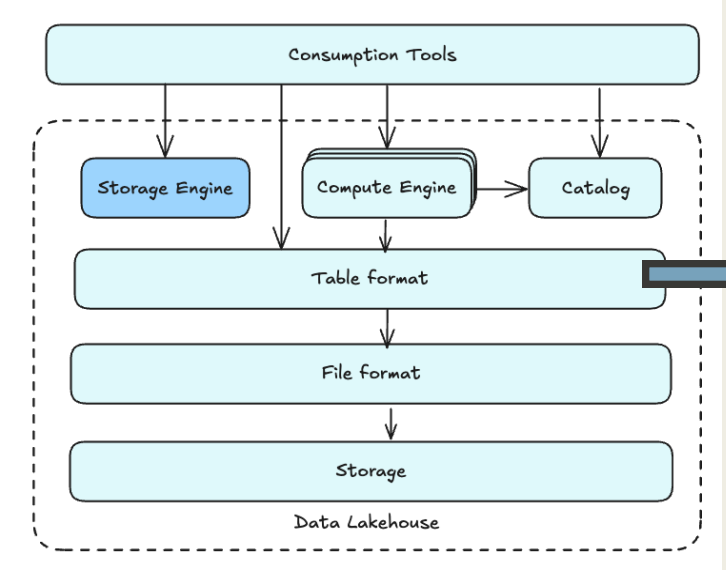
Warehouse to Lakehouse



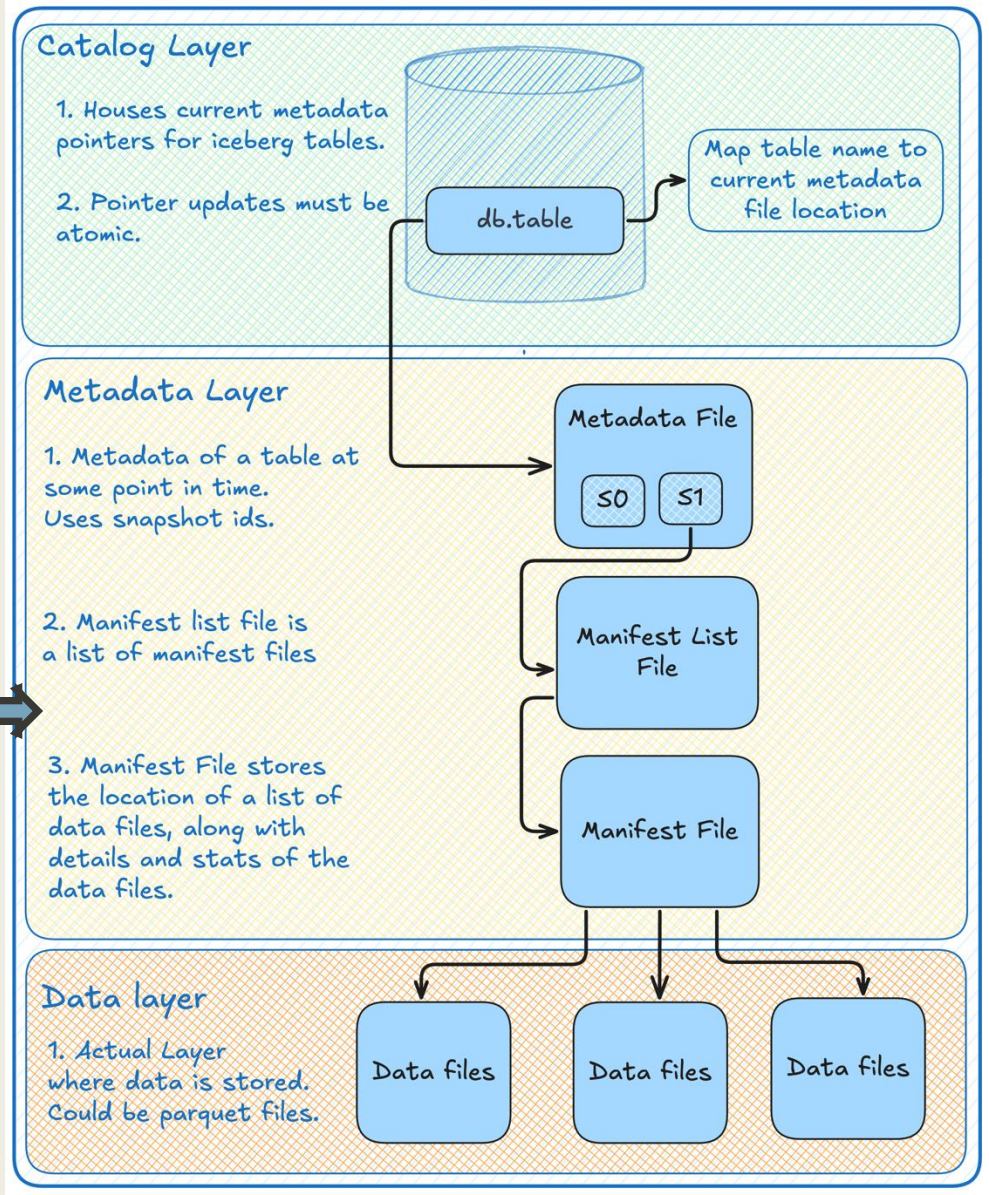


– Apache Iceberg the definitive guide (OREILLY)

Iceberg Table Format



– Apache Iceberg the definitive guide (OREILLY)

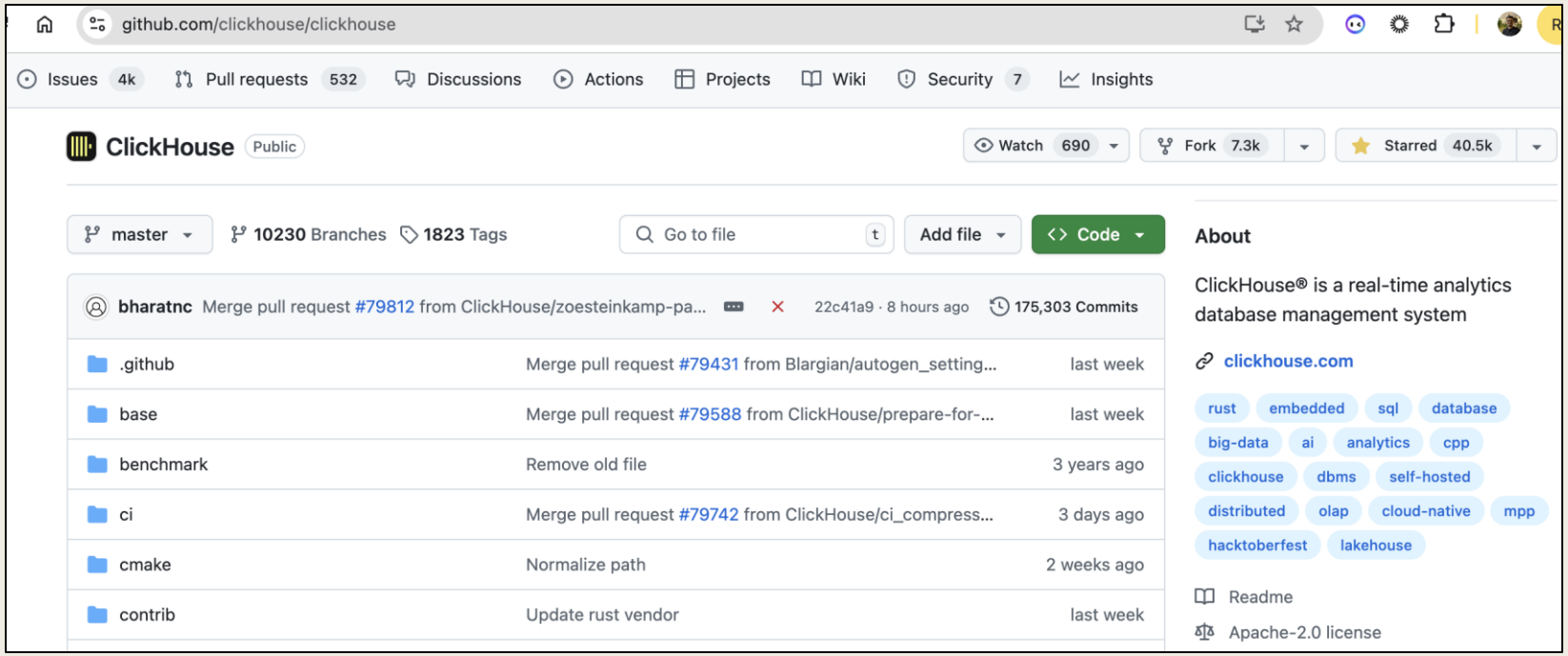


https://www.dremio.com/resources/guides/apache-iceberg-an-architectural-look-under-the-covers/

CLICKHOUSE & ICEBERG

Introducing ClickHouse!

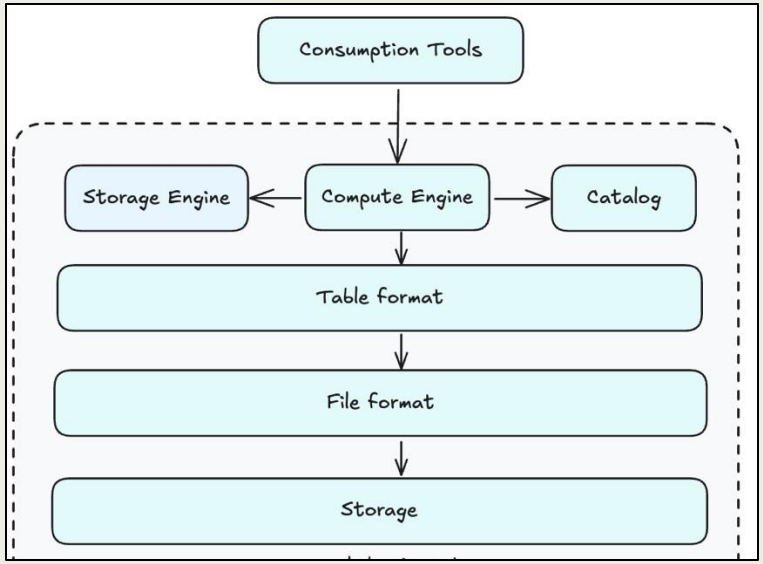
- Open source* OLAP database known for speed (see ClickBench!)
- Very easy to get started
  - *one binary, regular SQL/Tables*
  - *Very popular, open, well adopted!*

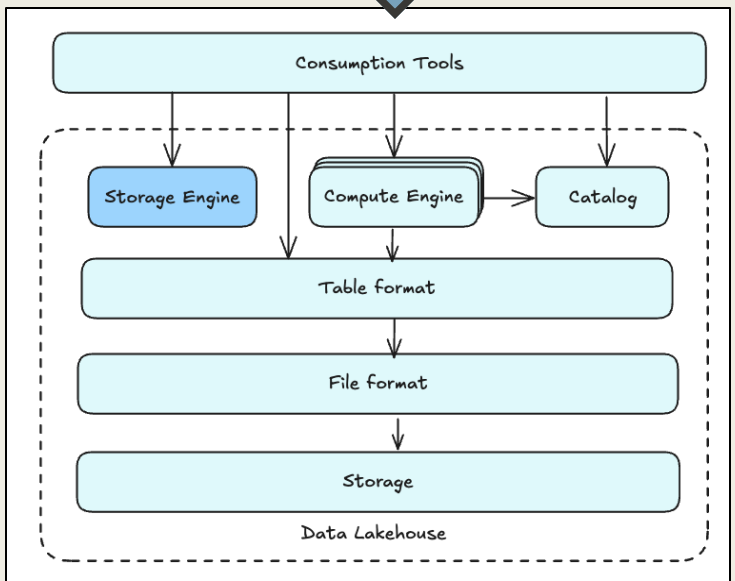


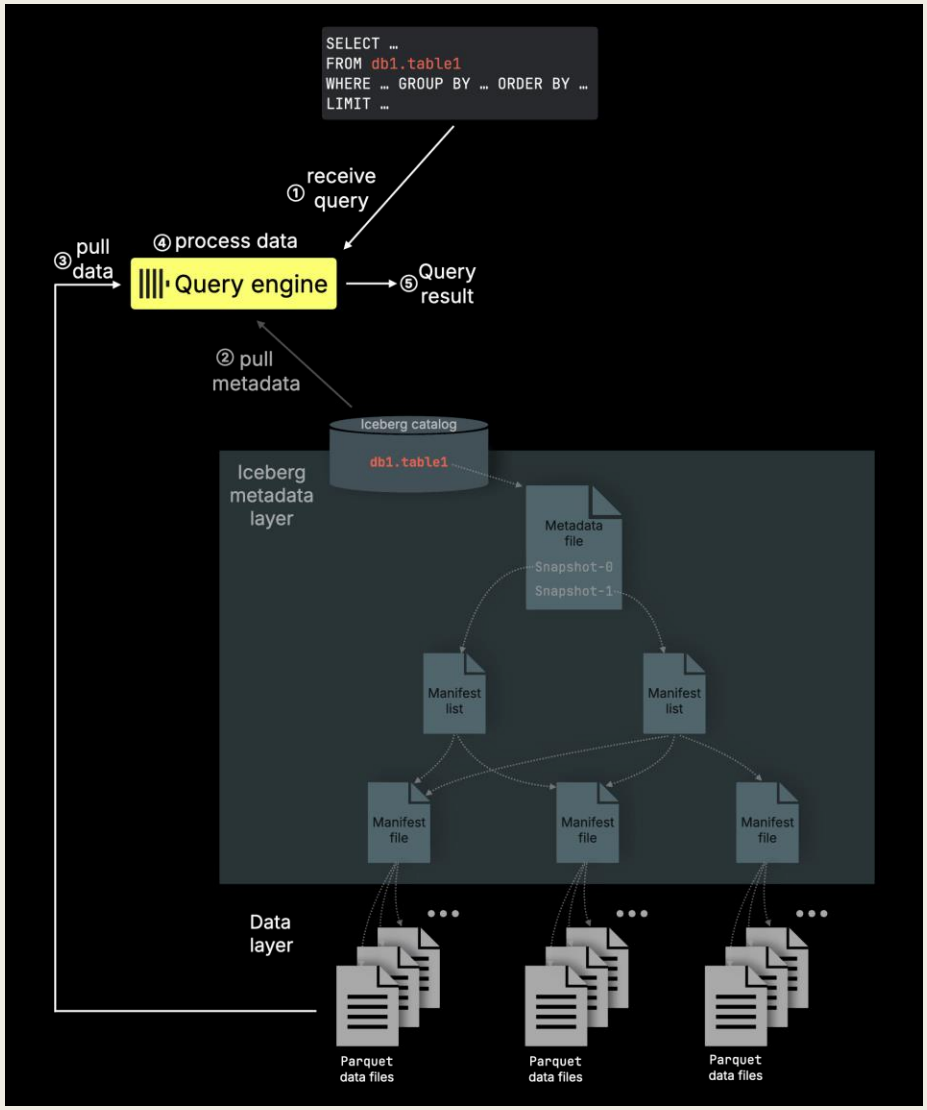
Introducing ClickHouse!

- 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

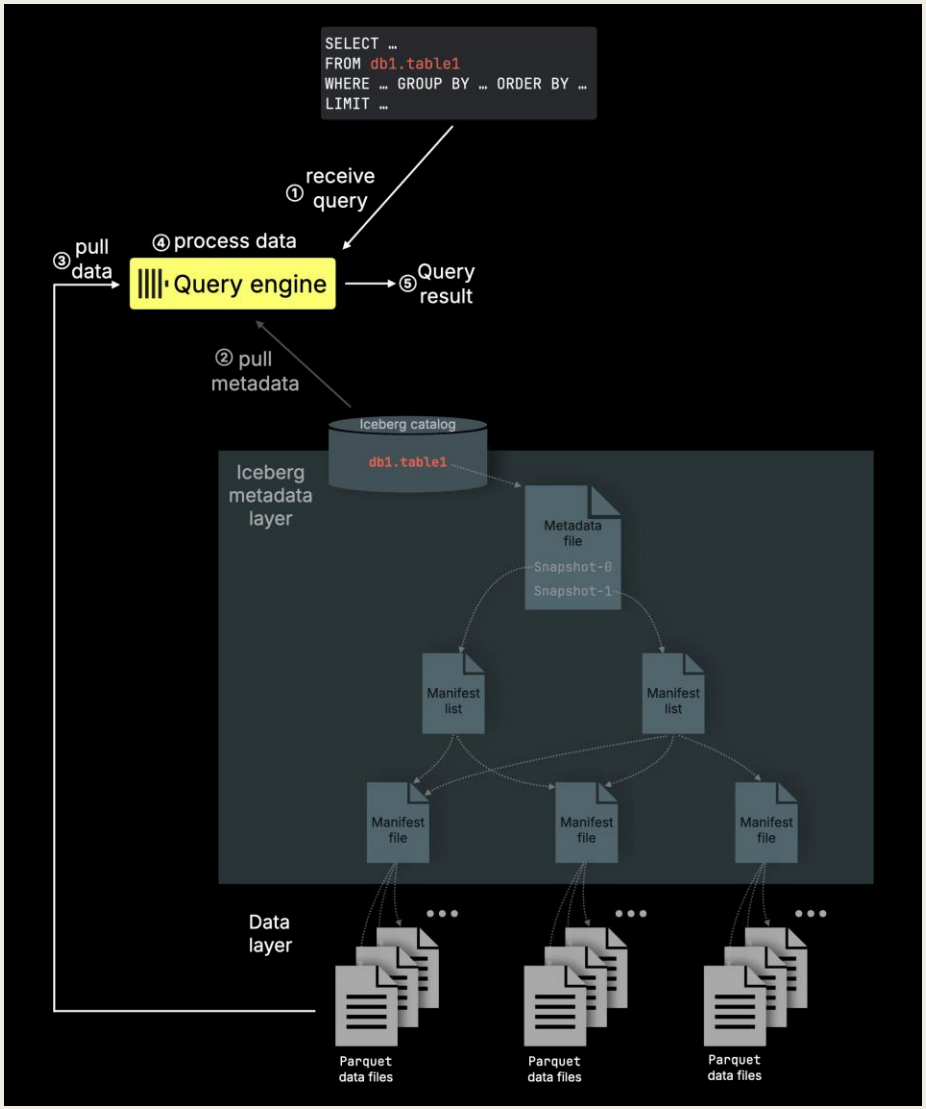


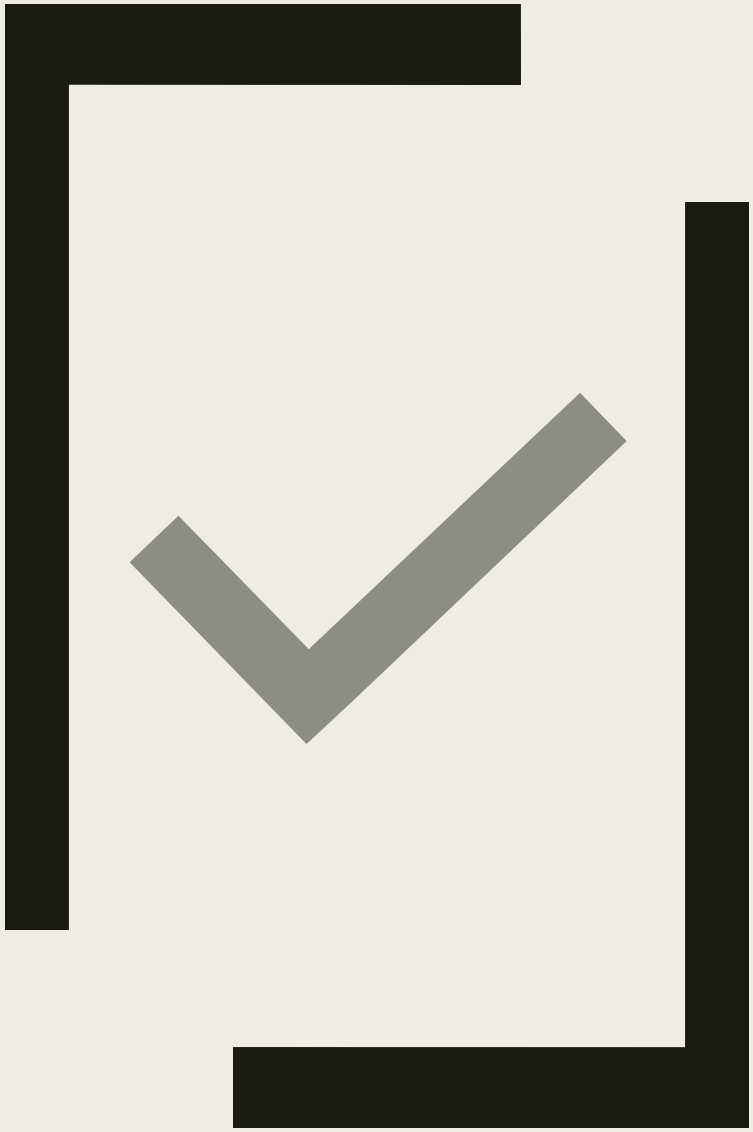




Query Engine Requirements

- 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*



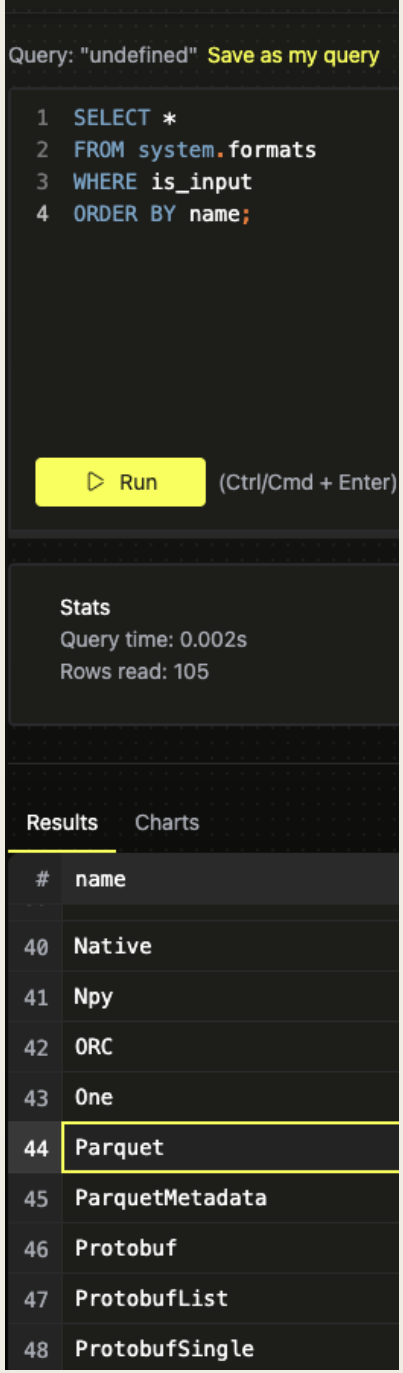


INTEGRATIONS IN CLICKHOUSE

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

DatabaseDataLake.h | IStorage.h | ParquetBlockInputFormat.h | IInputFormat.h

```
namespace DB

/** Input format is a source, that reads data from ReadBuffer.
  */
class IInputFormat : public SourceWithKeyCondition
{
protected:

    ReadBuffer * in [[maybe_un

public:
    /// ReadBuffer can be null
    IInputFormat(Block header,

    Chunk generate() override;

    /// All data reading from
    virtual Chunk read() = 0;

    /** In some usecase (hello
     * The recreating of parse
     * resetParser() which all
     * source stream without r
     * That should be called a
     */
    virtual void resetParser()

    virtual void setReadBuffer
    virtual void resetReadBuffer() { in = nullptr; }

    virtual const BlockMissingValues * getMissingValues() const { return nullptr; }

    /// Must be called from ParallelParsingInputFormat after readSuffix
```

Derived classes of 'IInputFormat'

- NativeORCBlockInputFormat (in DB)
- OneInputFormat (in DB)
- ValuesBlockInputFormat (in DB)
- ORCBlockInputFormat (in DB)
- ParquetBlockInputFormat (in DB)
- ParquetMetadataInputFormat (in DB)
- IRowInputFormat (in DB)
- ParallelParsingInputFormat (in DB)
- ArrowBlockInputFormat (in DB)
- RawBLOBRowInputFormat (in DB)
- NpyRowInputFormat (in DB)
- RowInputFormatWithDiagnosticInfo (in DB)
- LineAsStringRowInputFormat (in DB)
- AvroConfluentRowInputFormat (in DB)
- AvroRowInputFormat (in DB)
- JSONEachRowRowInputFormat (in DB)
- MsgPackRowInputFormat (in DB)
- RegexpRowInputFormat (in DB)
- RowInputFormatWithNamesAndTypes<FormatReaderImpl> (in DB)

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

Parquet Reader

DatabaseDataLake.h | IStorage.h | ParquetBlockInputFormat.h | IInputFormat.h

```
namespace DB

/** Input format is a source, that reads data from ReadBuffer.
  */
class IInputFormat : public SourceWithKeyCondition
{
```

Derived classes of 'IInputFormat'

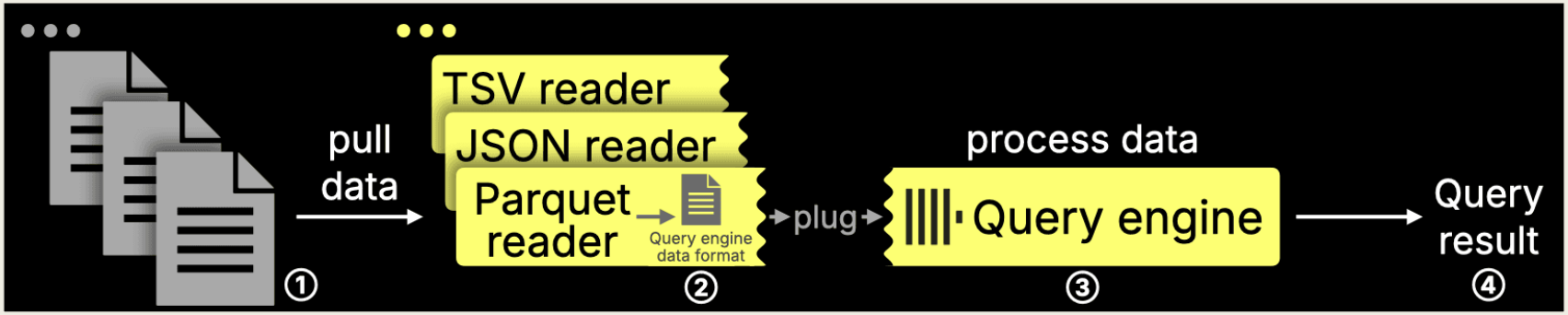
- NativeORCBlockInputFormat (in DB)
- OneInputFormat (in DB)
- ValuesBlockInputFormat (in DB)
- ORCBlockInputFormat (in DB)
- ParquetBlockInputFormat (in DB)
- ParquetMetadataInputFormat (in DB)
- IRowInputFormat (in DB)
- ParallelParsingInputFormat (in DB)
- ArrowBlockInputFormat (in DB)
- RawBLOBRowInputFormat (in DB)
- NpyRowInputFormat (in DB)
- RowInputFormatWithDiagnosticInfo (in DB)
- LineAsStringRowInputFormat (in DB)
- AvroConfluentRowInputFormat (in DB)
- AvroRowInputFormat (in DB)
- JSONEachRowRowInputFormat (in DB)
- MsgPackRowInputFormat (in DB)
- RegexpRowInputFormat (in DB)
- RowInputFormatWithNamesAndTypes<FormatReaderImpl> (in DB)

DatabaseDataLake.h | IStorage.h | ParquetBlockInputFormat.h

```
#if USE_PARQUET
namespace DB

class ParquetBlockInputFormat : public IInputFormat
{
public:
    ParquetBlockInputFormat(
        ReadBuffer & buf,
        const Block & header,
        const FormatSettings & format_settings,
        size_t max_decoding_threads,
        size_t max_io_threads,
        size_t min_bytes_for_seek);

    ~ParquetBlockInputFormat() override;

    void resetParser() override;

    String getName() const override { return "ParquetBlockInputFo

    const BlockMissingValues * getMissingValues() const override;

    size_t getApproxBytesReadForChunk() const override { return prev

private:
    Chunk read() override;

    void onCancel() noexcept override
```

Reading non-native files

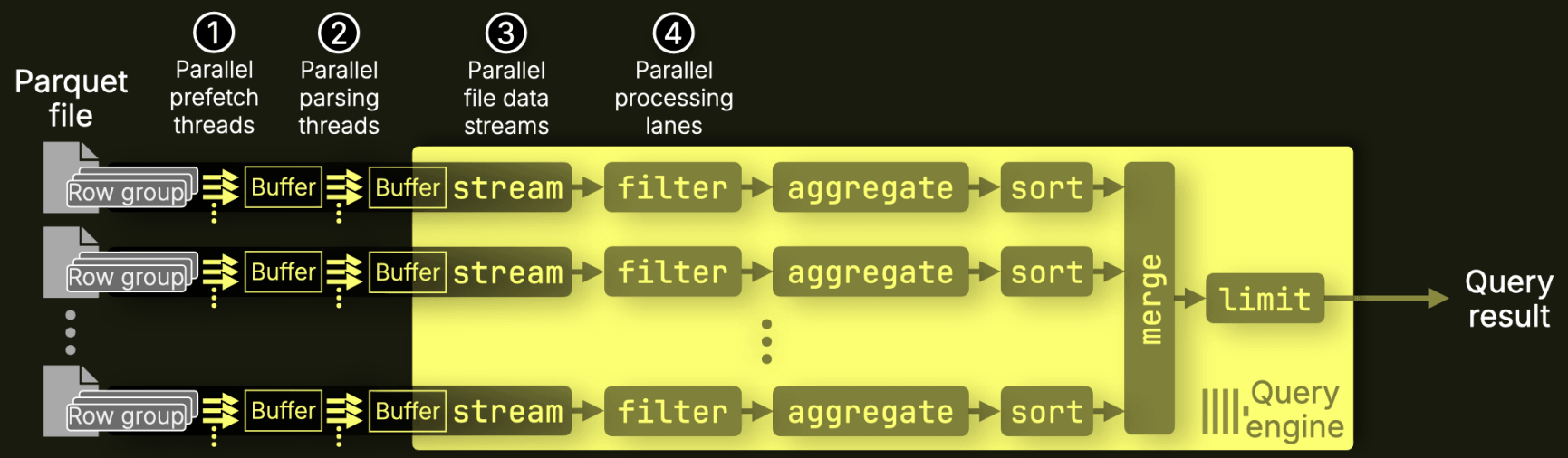
- 1: Format Reader reads and parses external files (eg., Parquet, CSV, JSON…)
- 2: Data converted into ClickHouse's internal columnar in-memory format
- 3: Query engine runs SQL operations (filter, join, aggregate) on in-memory data
- 4: Produces result set for client / downstream pipeline



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

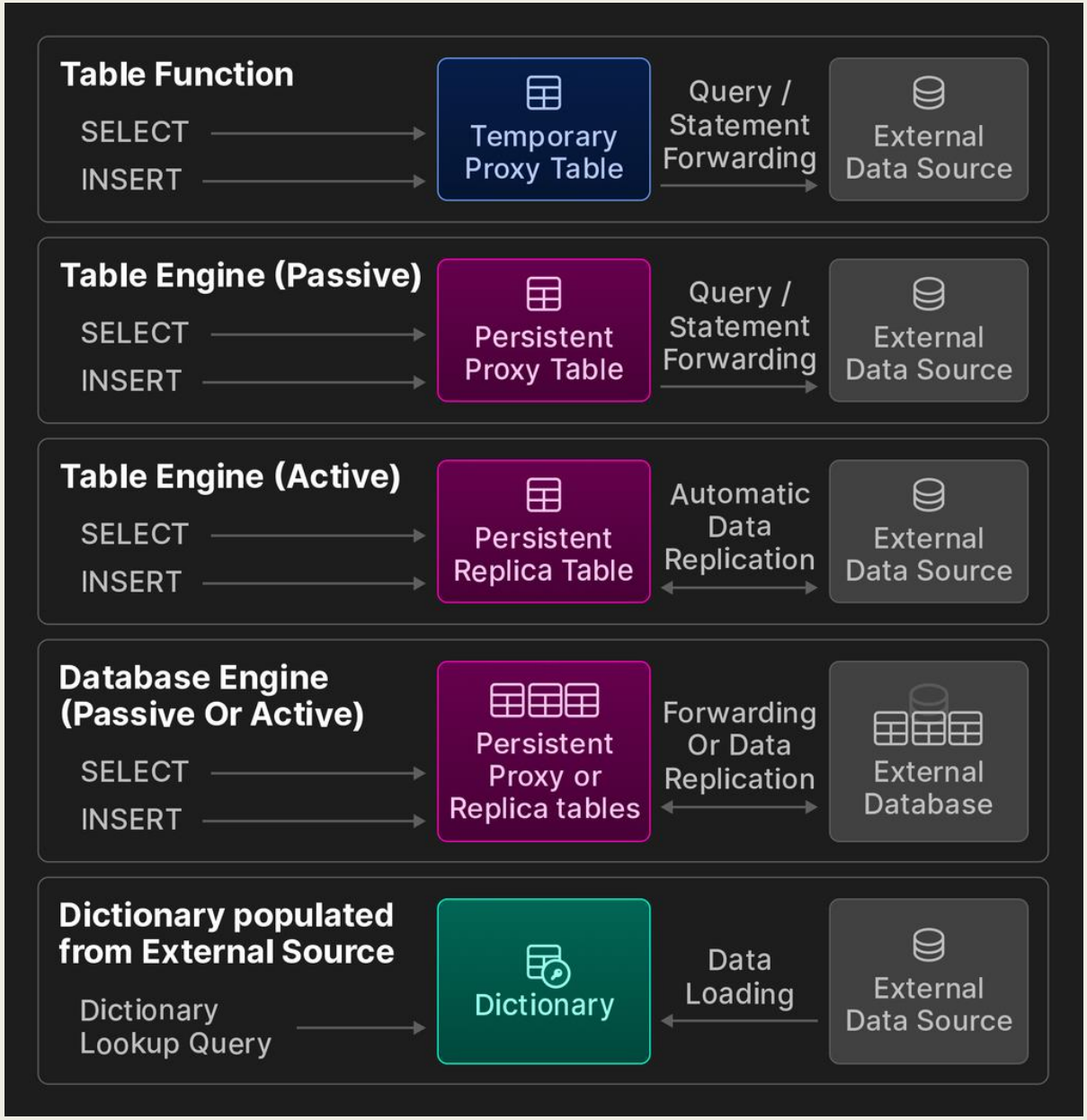
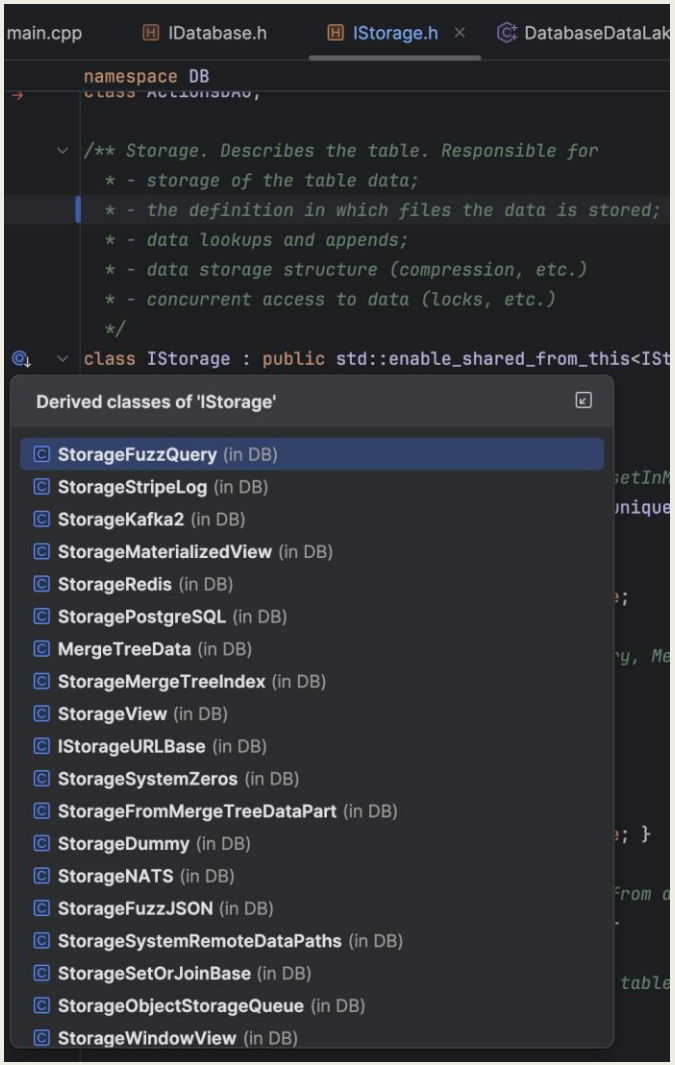


Table Engines

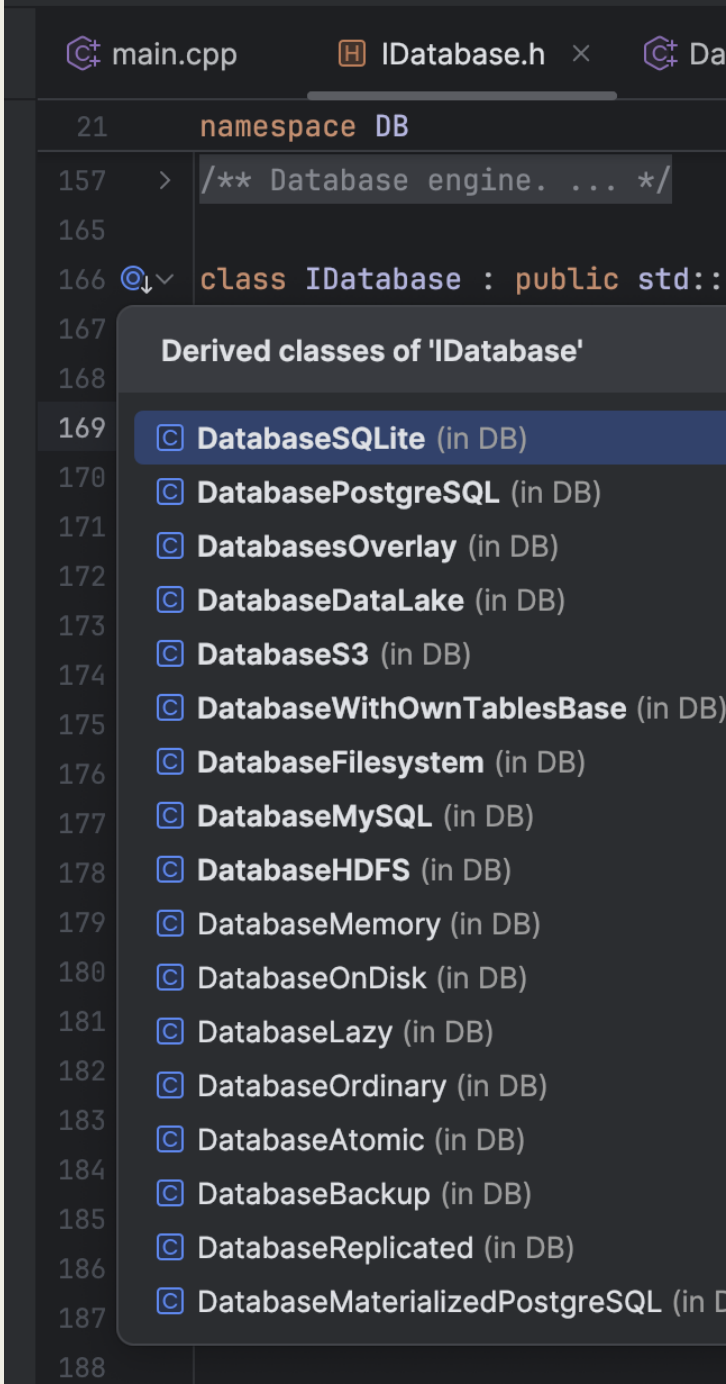
- Istorage.h : Interface for Table Engines
- Implementations => different table engine
  - *StorageMergeTree,*
  - *StoragePostgreSQL,*
  - *StorageKafka2*
  - *StorageObjectStorage (S3, AzureBlob, HDFS)*
  - *... 69 derived classes in total*
- Core responsibilities:
  - read(),
  - write(...) , alter(...),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

Debug

IDatabase.h | DatabaseDataLake.h | IStorage.h | ICatalog.h

```
#if USE_AVRO && USE_PARQUET
namespace DB

class DatabaseDataLake final : public IDatabase, WithContext
{
public:
    explicit DatabaseDataLake(
        const std::string & database_name_,
        const std::string & url_,
        const DatabaseDataLakeSettings & settings_,
        ASTPtr database_engine_definition_,
        ASTPtr table_engine_definition_);

    String getEngineName() const override { return DataLake:

    bool canContainMergeTreeTables() const override { return fa
    bool canContainDistributedTables() const override { return
    bool shouldBeEmptyOnDetach() const override { return false;

    bool empty() const override;

    bool isTableExist(const String & name, ContextPtr context)
    StoragePtr tryGetTable(const String & name, ContextPtr con

    DatabaseTablesIteratorPtr getTablesIterator(
```

main.cpp | IDatabase.h | Da

```
namespace DB
/** Database engine. ... */

class IDatabase : public std::
```

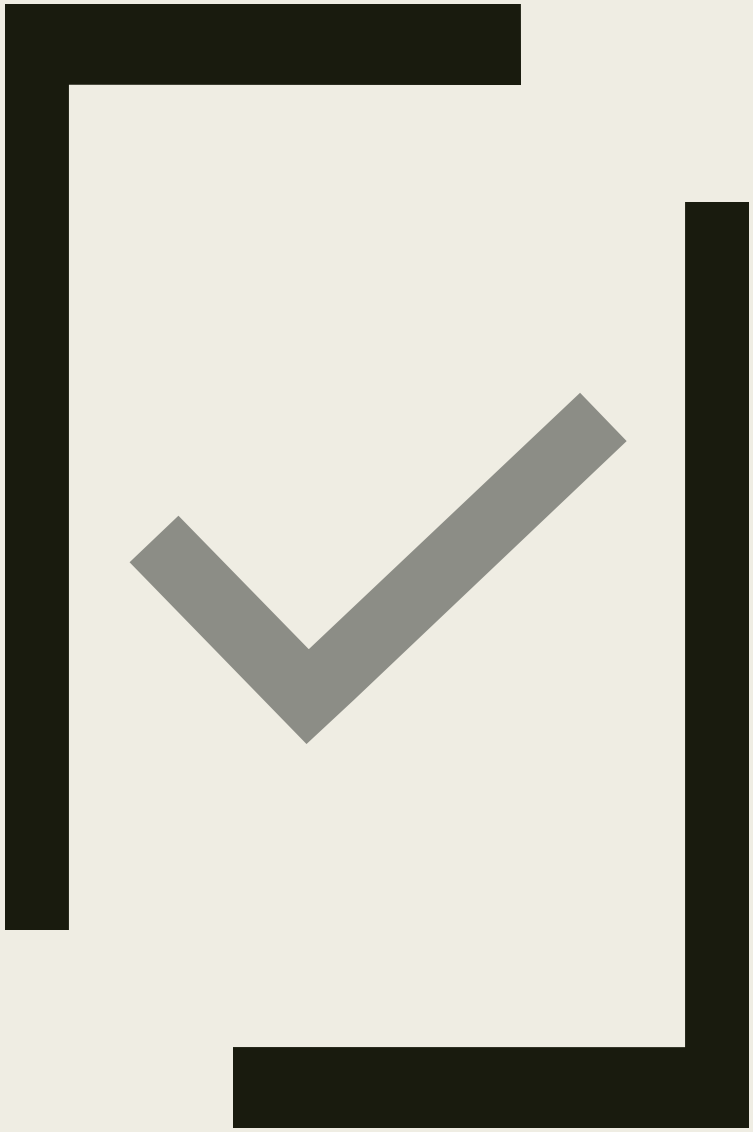
Derived classes of 'IDatabase'

- DatabaseSQLite (in DB)
- DatabasePostgreSQL (in DB)
- DatabasesOverlay (in DB)
- DatabaseDataLake (in DB)
- DatabaseS3 (in DB)
- DatabaseWithOwnTablesBase (in DB)
- DatabaseFilesystem (in DB)
- DatabaseMySQL (in DB)
- DatabaseHDFS (in DB)
- DatabaseMemory (in DB)
- DatabaseOnDisk (in DB)
- DatabaseLazy (in DB)
- DatabaseOrdinary (in DB)
- DatabaseAtomic (in DB)
- DatabaseBackup (in DB)
- DatabaseReplicated (in DB)
- DatabaseMaterializedPostgreSQL (in D

Databases/IDatabase.h

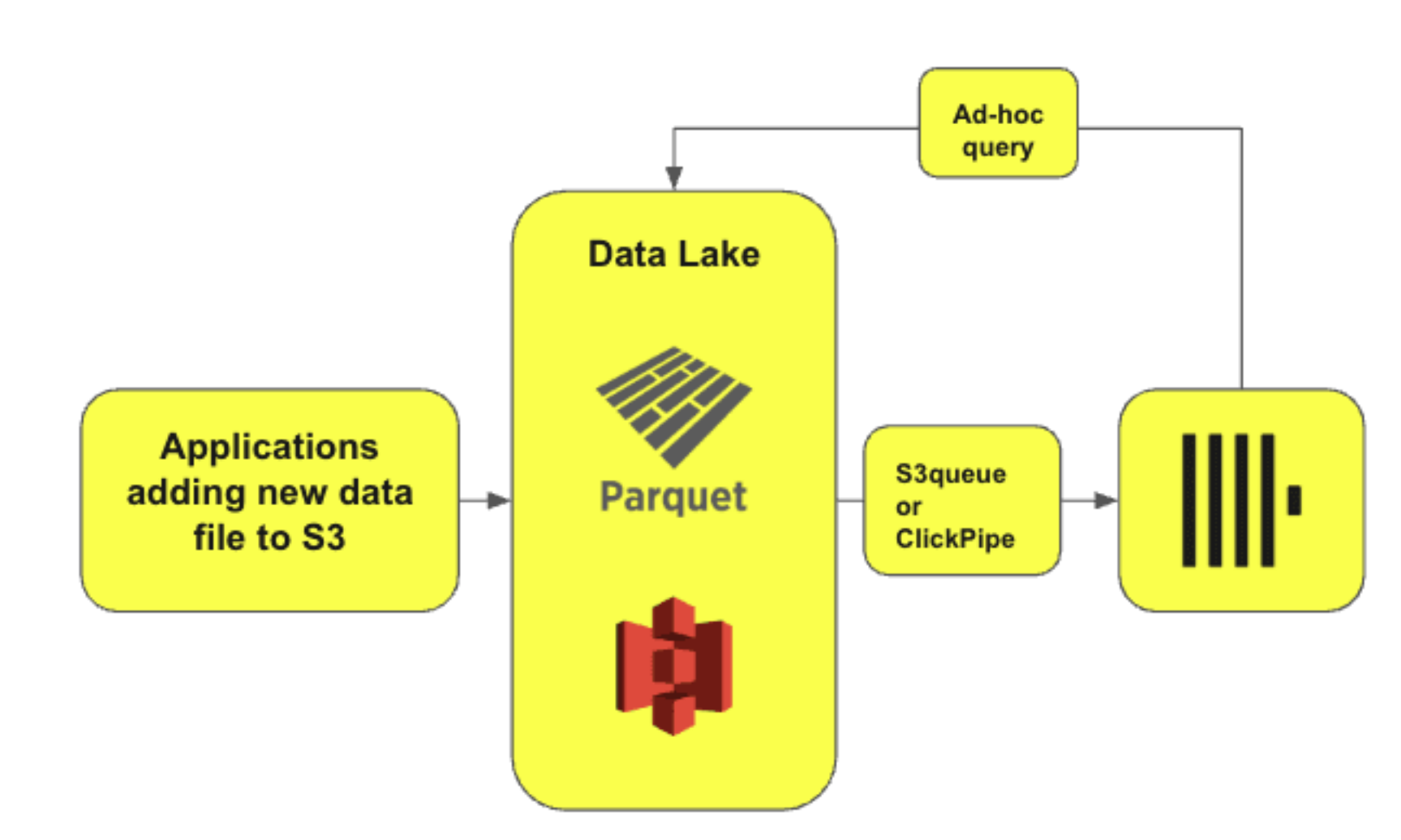
Why It Matters for Lakehouse

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



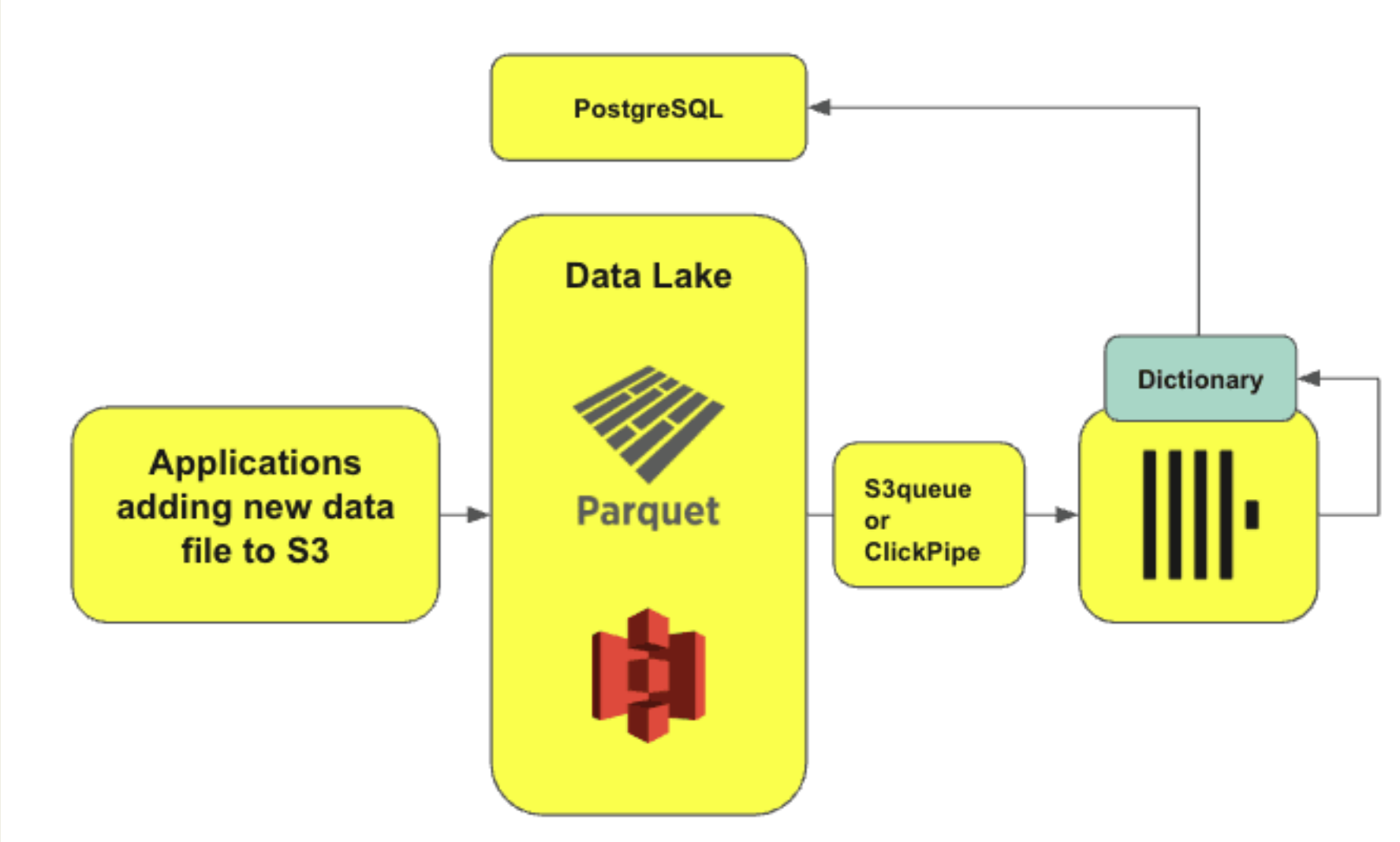
READ PARQUET & ICEBERG FROM CLICKHOUSE

Ad-hoc queries (Trino etal..)

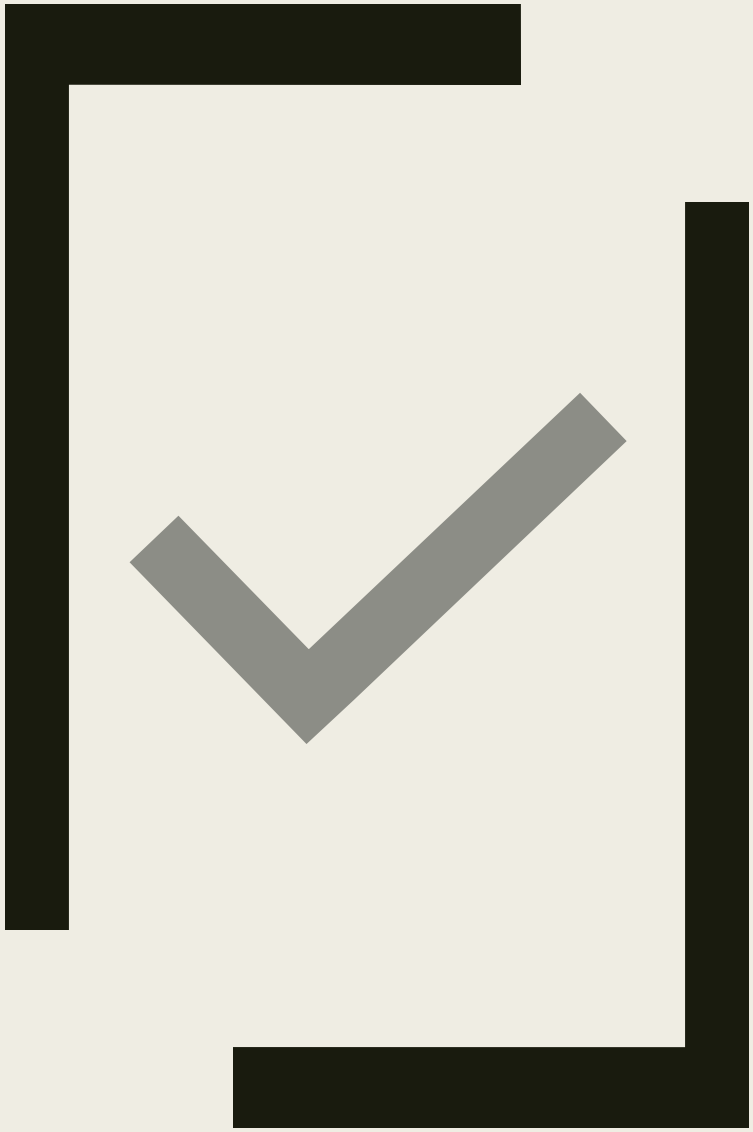


https://clickhouse.com/blog/climbing-the-iceberg-with-clickhouse

Querying multiple sources

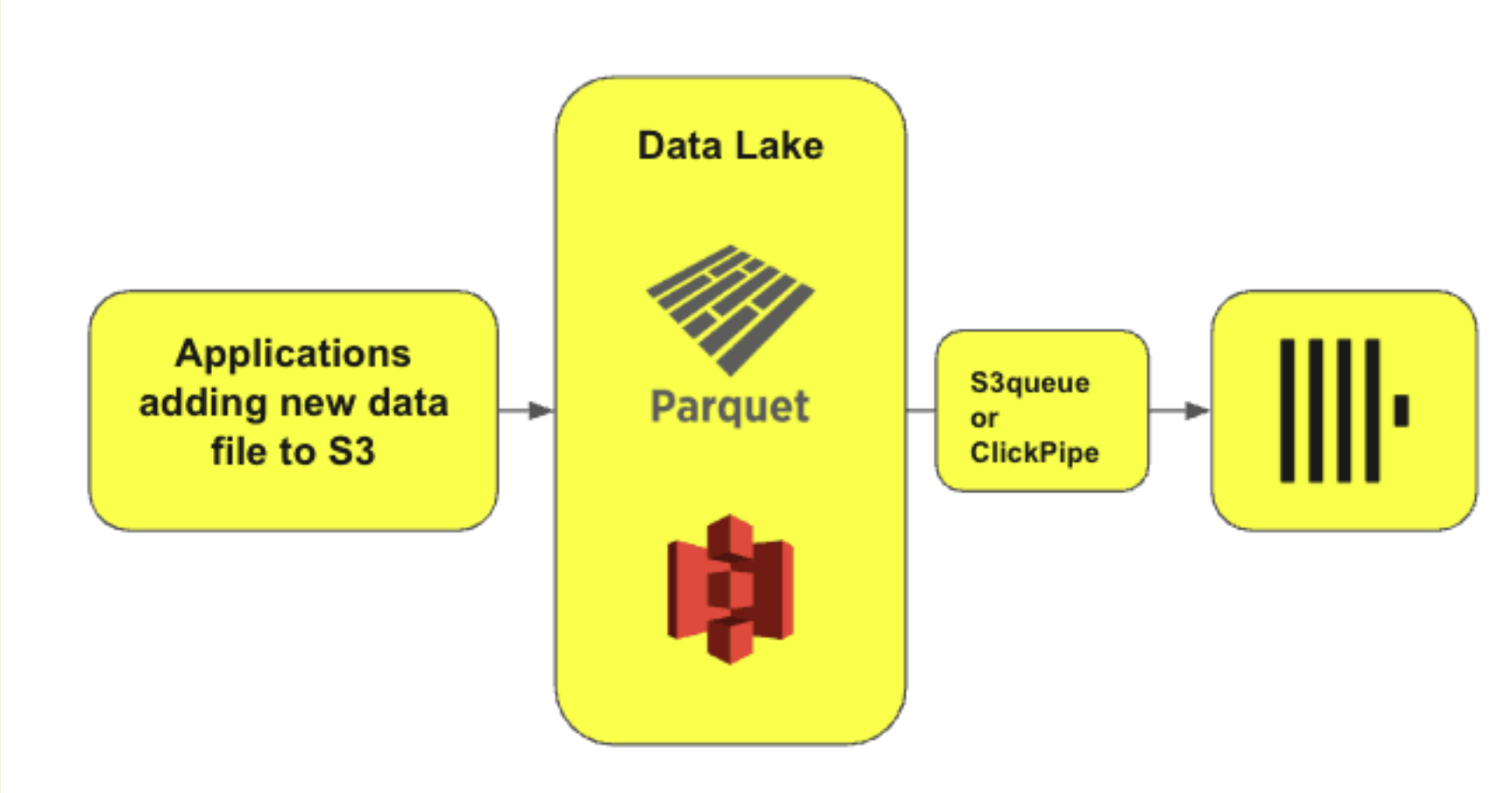


https://clickhouse.com/blog/climbing-the-iceberg-with-clickhouse



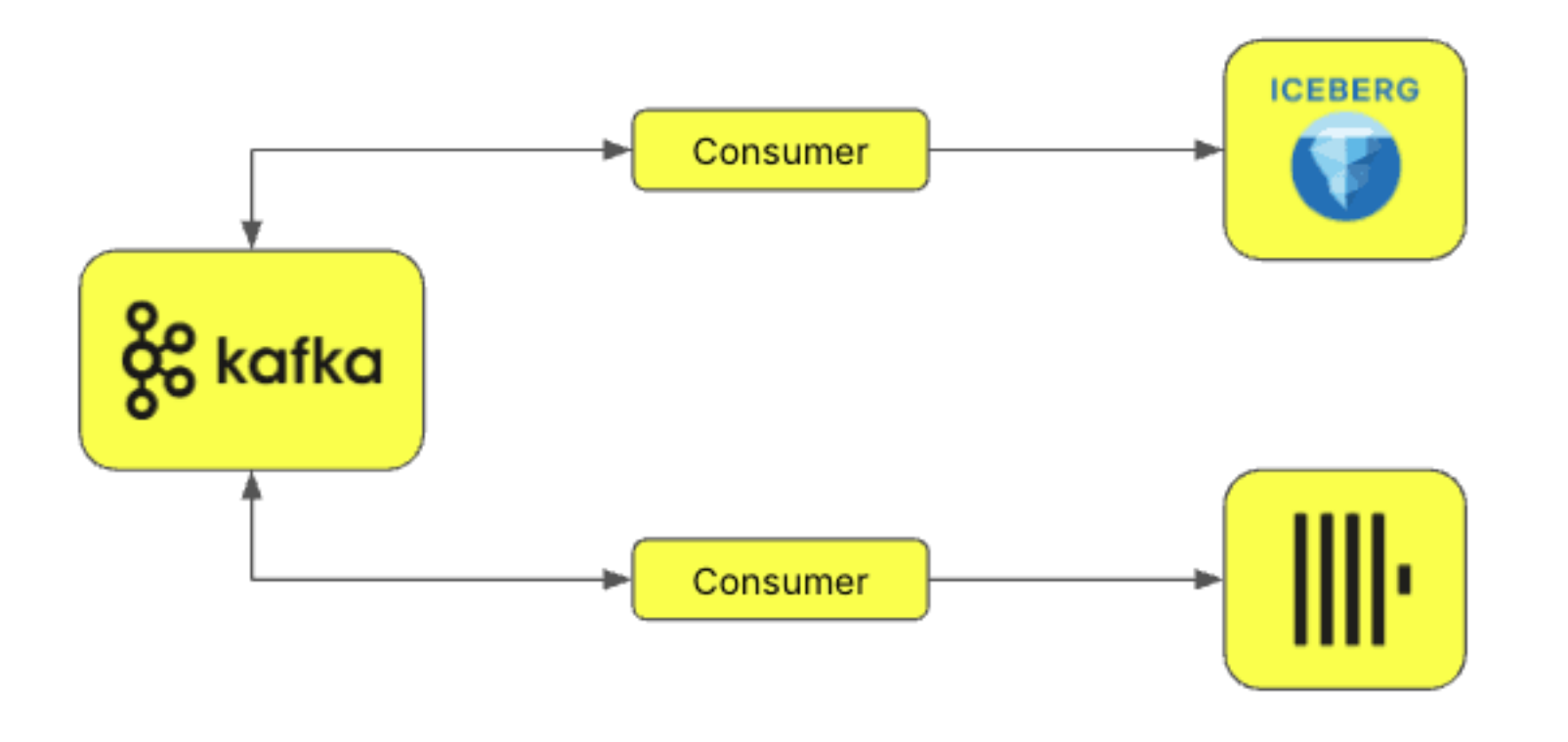
WRITE IN ICEBERG FORMAT

Writes(1): s3cluster/s3queue

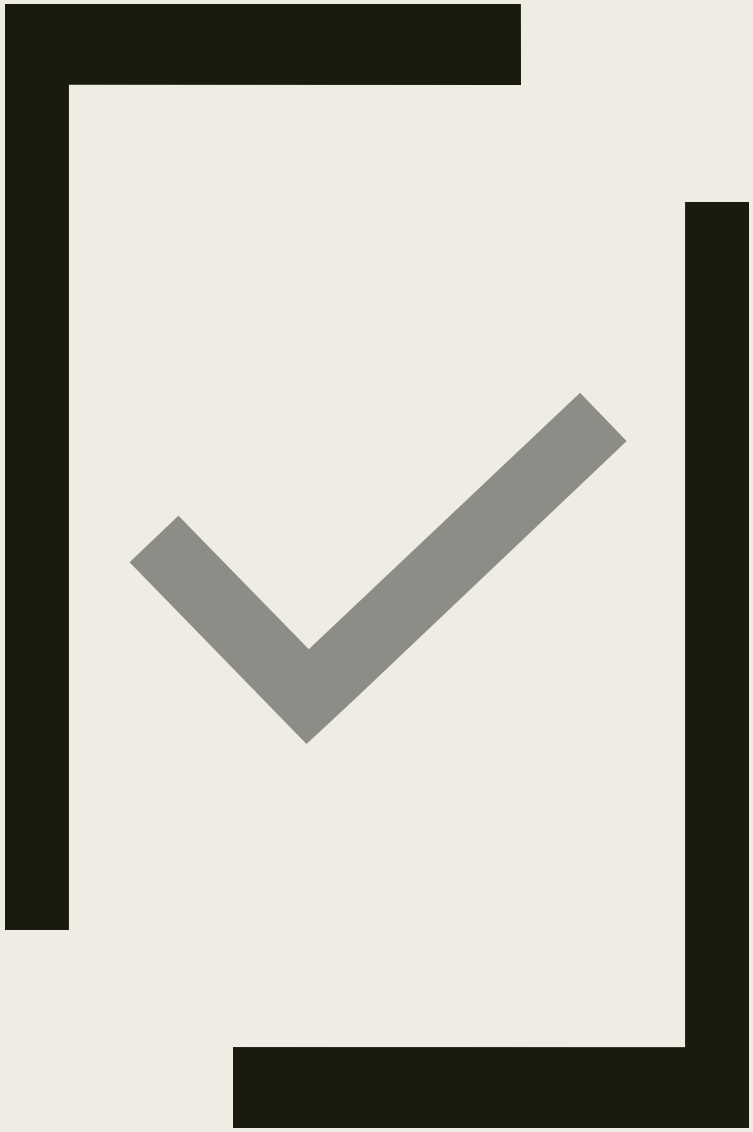


https://clickhouse.com/blog/climbing-the-iceberg-with-clickhouse

Writes(2): Double write/ttl/tiering

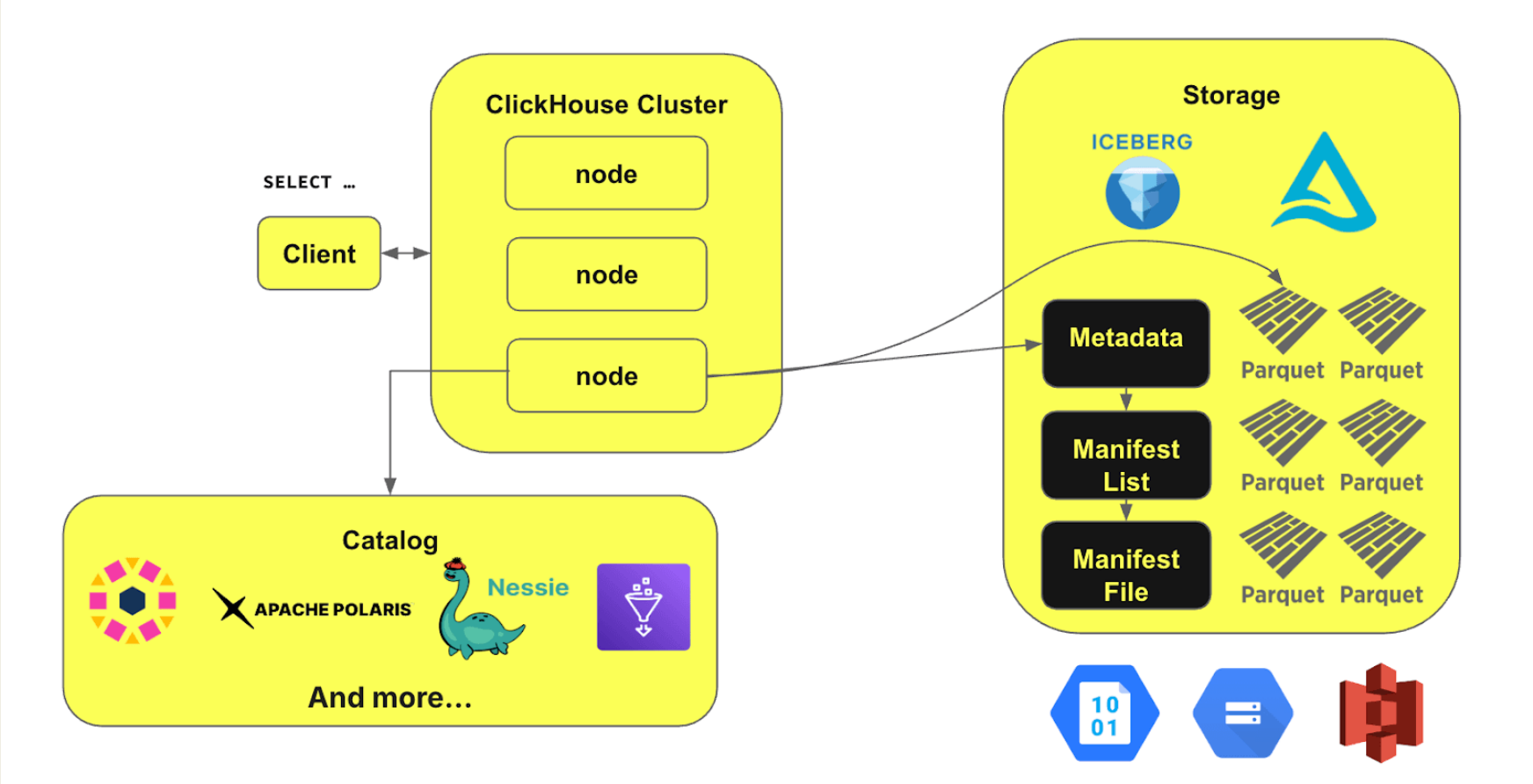


https://clickhouse.com/blog/climbing-the-iceberg-with-clickhouse



SUPPORT FOR OTHER ICEBERG GOODIES!

Catalog Integration



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

Catalog Integration

abase.h | DatabaseDataLake.h | IStorage.h | ICatalog.h | DatabaseD

```cpp
namespace DataLake
/// Base class for catalog implementation.
/// Used for communication with the catalog.
class ICatalog
{
public:
    using Namespaces = std::vector<std::string>;

    explicit ICatalog(const std::string & warehouse_) : warehouse( warehous

    virtual DB::DatabaseDataLakeCatalogType getCatalogType() const = 0;
    virtual ~ICatalog() = default;

    /// Does catalog have any tables?
    virtual bool empty() const = 0;

    /// Fetch tables' names list.
    /// Contains full namespaces in names.
    virtual DB::Names getTables() const = 0;

    /// Check that a table exists in a given namespace.
    virtual bool existsTable(
        const std::string & namespace_naem,
        const std::string & table_name) const = 0;

    /// Get table metadata in the given namespace.
    /// Throw exception if table does not exist.
    virtual void getTableMetadata(
        const std::string & namespace_name,
        const std::string & table_name,
        TableMetadata & result) const = 0;

    /// Get table metadata in the given namespace.
    /// Return `false` if table does not exist, `true` otherwise.
    virtual bool tryGetTableMetadata(
```

Derived classes of 'ICatalog'

- GlueCatalog (in DataLake)
- HiveCatalog (in DataLake)
- RestCatalog (in DataLake)
- UnityCatalog (in DataLake)

Metadata caching

StorageFile.h | ManifestFile.h | IcebergMetadataFilesCache.h

```
#if USE_AVRO
namespace DB
struct IcebergMetadataFilesCacheWeightFunction
};

class IcebergMetadataFilesCache : public
CacheBase<String, IcebergMetadataFilesCacheCell, std::hash<Strin
 IcebergMetadataFilesCacheWeightFunction>
{
public:
    using Base = CacheBase<String, IcebergMetadataFilesCacheCell

    IcebergMetadataFilesCache(const String & cache_policy, size_
        : Base( cache_policy, max_size_in_bytes, max_count, si
    {}

    static String getKey(StorageObjectStorage::ConfigurationPtr
    {
        return  std::filesystem::path( s: config->getDataSourceD
    }

    template <typename LoadFunc>
    Poco::JSON::Object::Ptr getOrSetTableMetadata(const String &
    {
        auto load_fn_wrapper = [&]()
        {
            const auto & [json_ptr, json_size] = load_fn();
            return std::make_shared<IcebergMetadataFilesCacheCel
        };
```

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

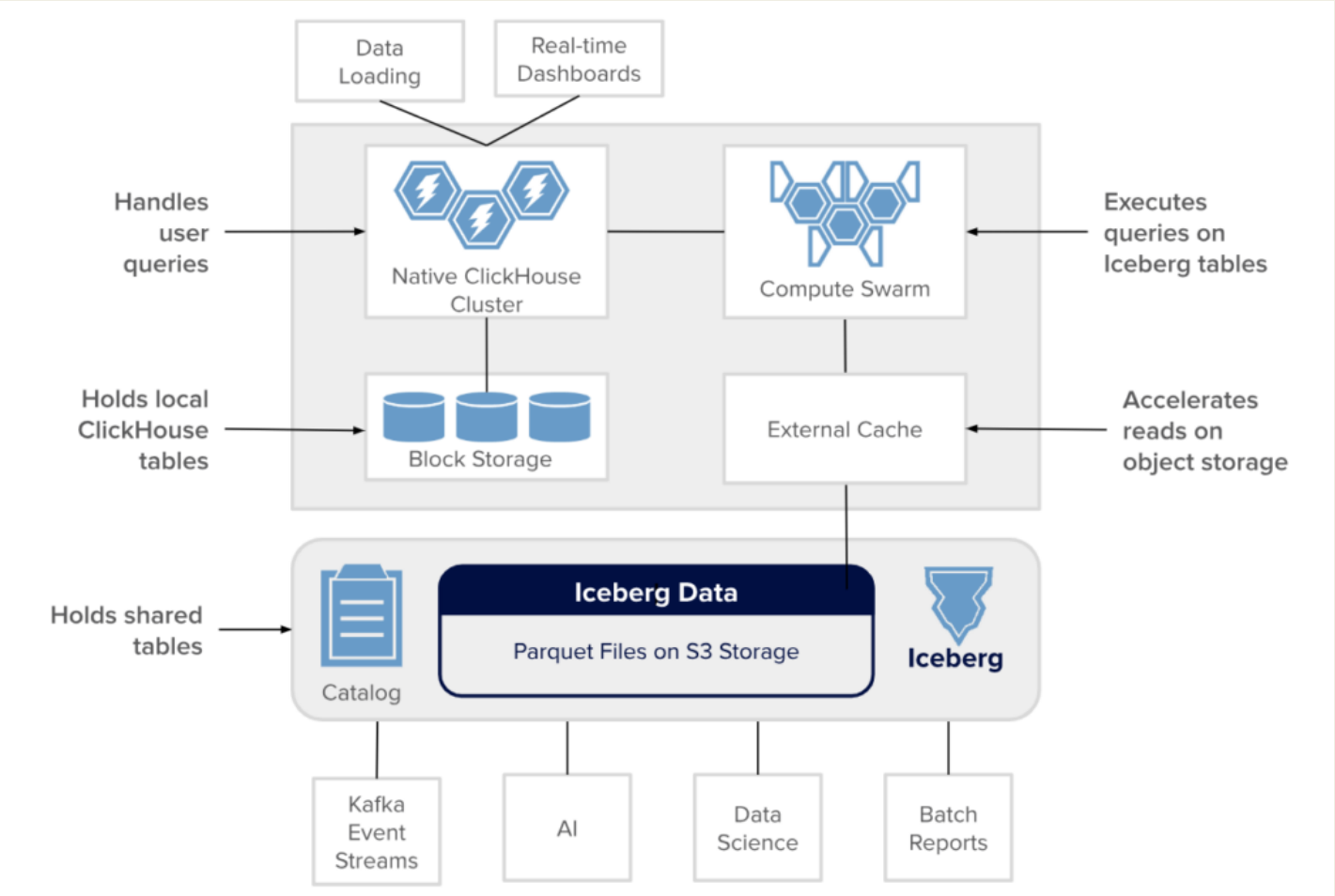
Data Lakes

- [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



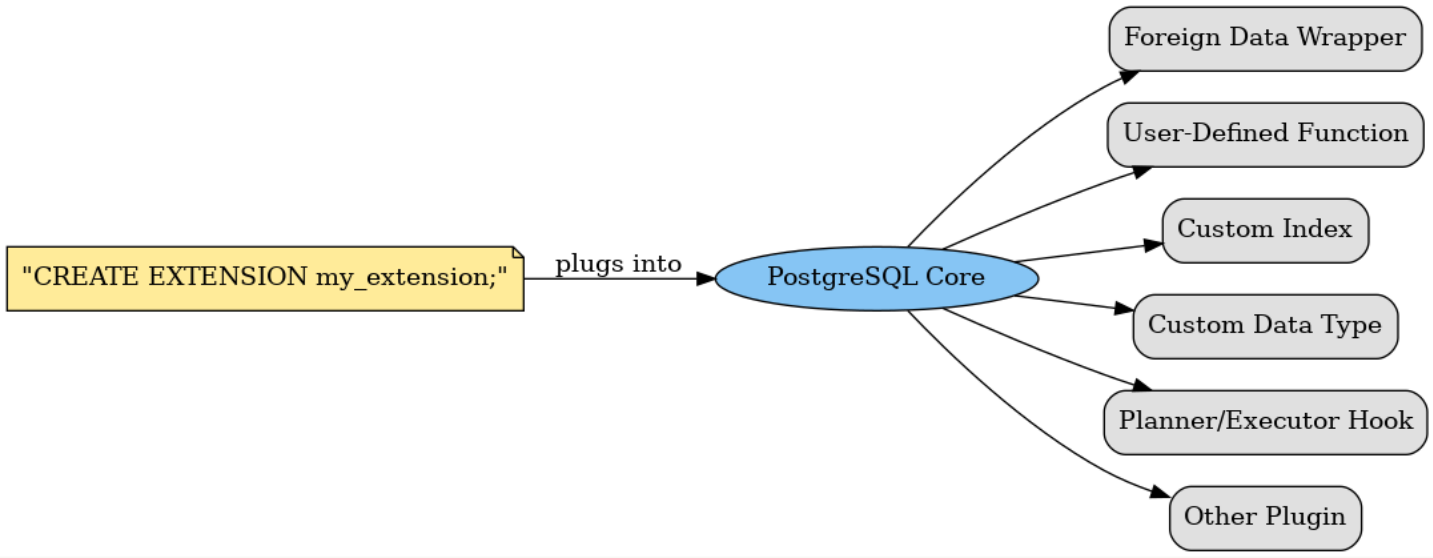
Live Demo

- Demo setup can be found here

HACKING POSTGRESQL FOR ICEBERG

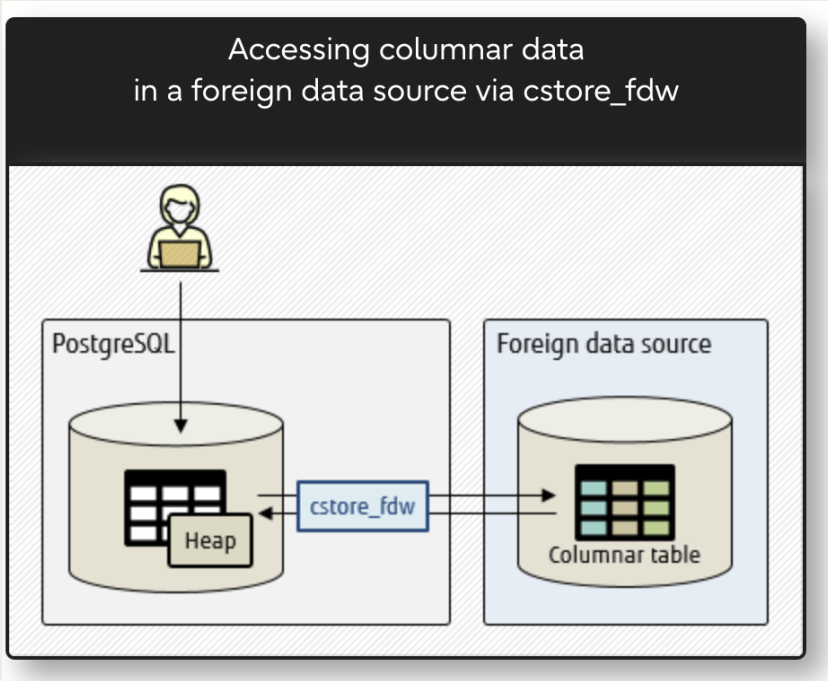
Built to extend

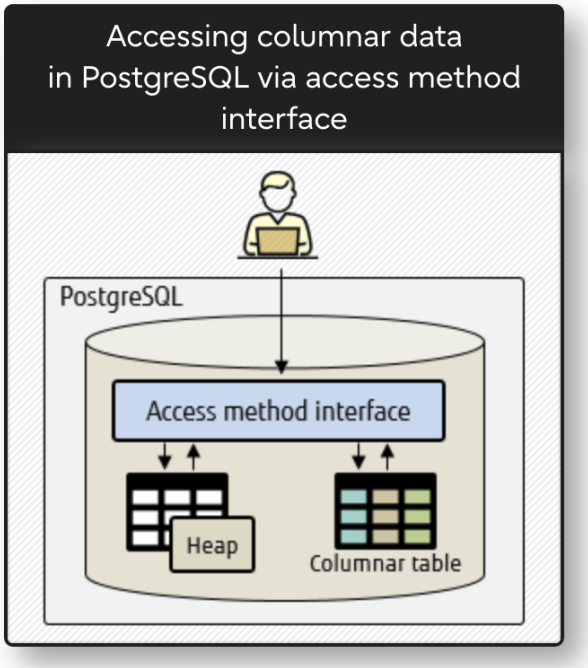
- Integration with ***Extensions***
  - *TimescaleDB (time-series),*
  - *Citus (distributed SQL),*
  - *PostGIS (geospatial data)*
  - ***pg_duckdb***



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*

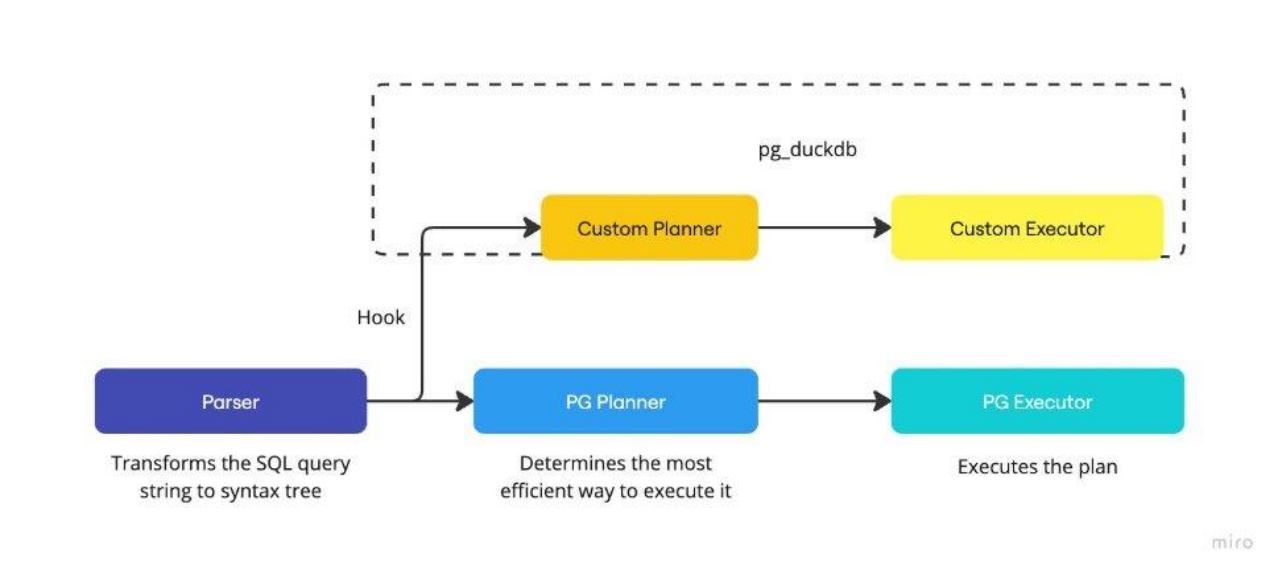
pg_duckdb == Postgres(PG) + DuckDB

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

Query Stealing in pg_duckdb



- planner_hook to create a planned statement that DuckDb execution engine can work on.
- ExecutorStart_hook to detect and override standard executor
- ExecutorEnd_hook to cleanup post duckdb execution
- ExplainOneQuery_hook to provide a better explanation of duckdb execution.

Hooks initialization and Query Stealing

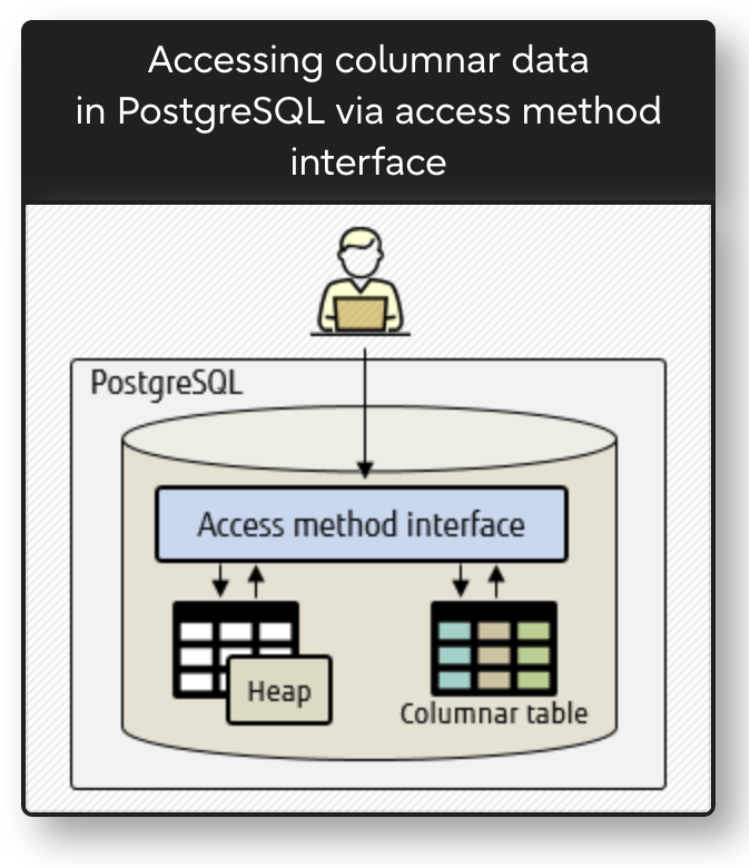
```
void DuckdbInitHooks(void) {
    prev_planner_hook = planner_hook;
    planner_hook = DuckdbPlannerHook;

    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

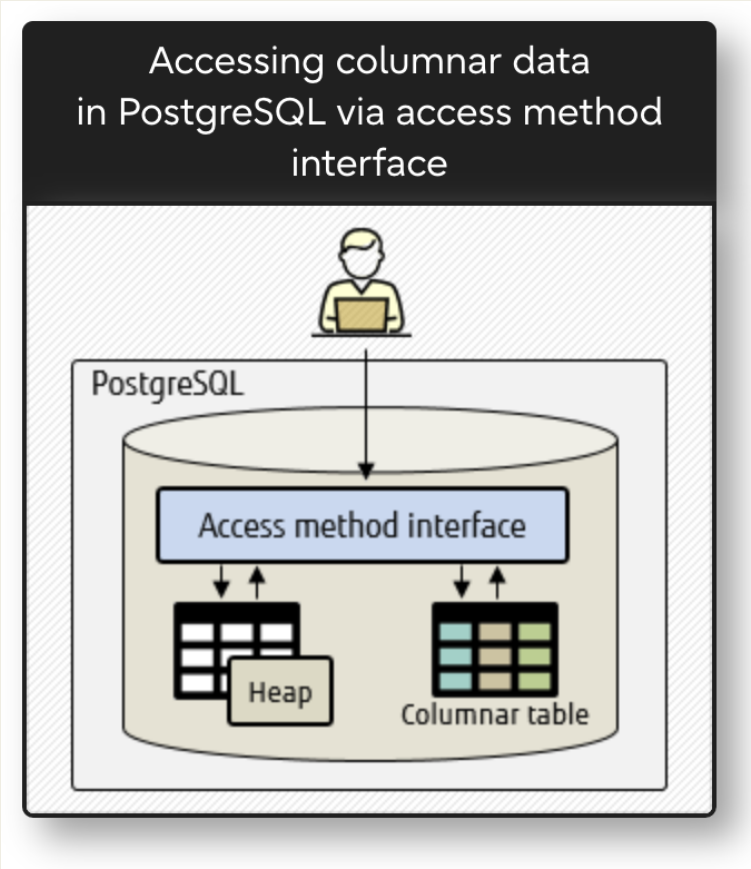
pg_mooncake (pg_duckdb++)



- New column 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:

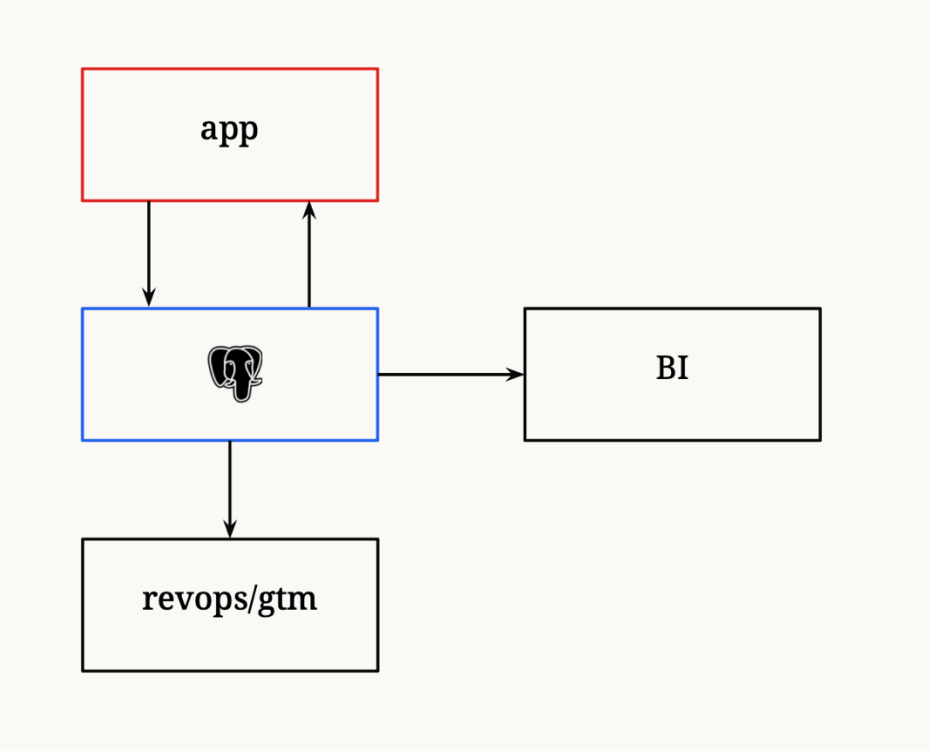
```
CREATE TABLE user_activity(
  user_id BIGINT,
  activity_type TEXT,
  activity_timestamp TIMESTAMP,
  duration INT
) USING columnstore;
```

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;
```

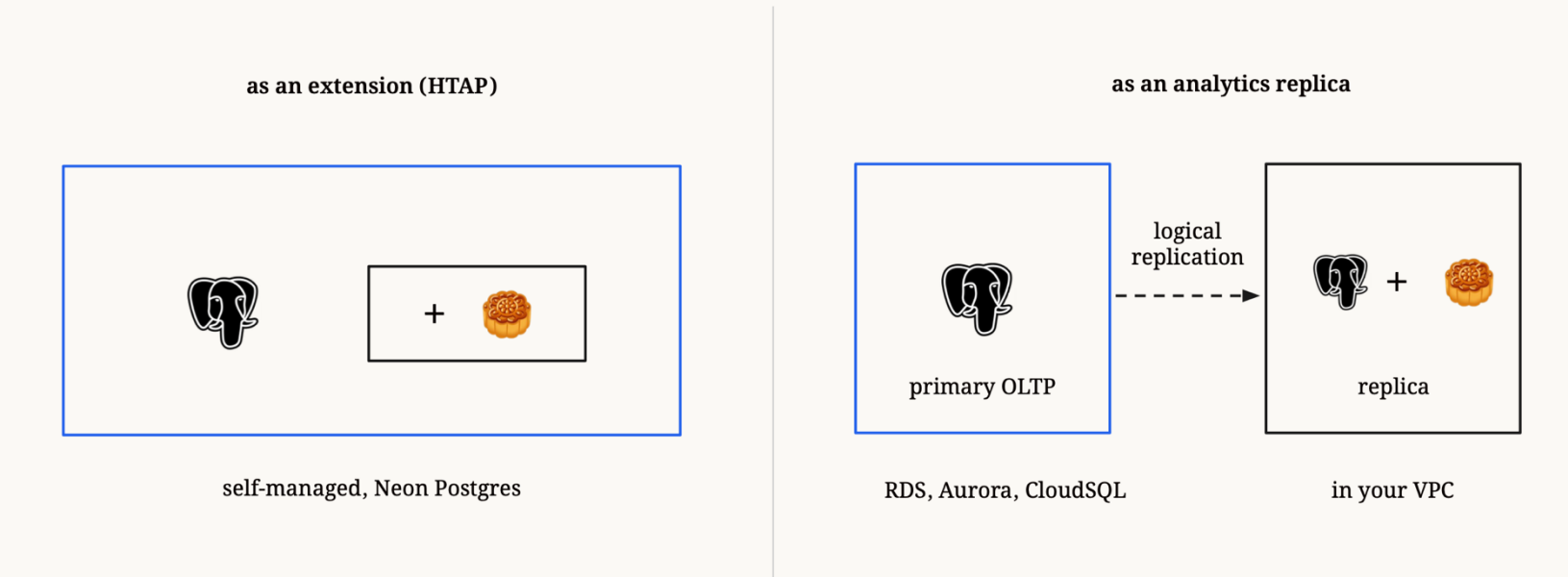
https://www.postgresql.fastware.com/blog/postgresql-table-access-methods

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 (any postgres):

```
CREATE PUBLICATION pub
FOR TABLE orders, customers;
```

Destination (Crunchy Data Warehouse):

```
CREATE SUBSCRIPTION sub
CONNECTION 'host=... '
PUBLICATION pub
WITH (create_tables_using 'iceberg');
```

Auto-create Iceberg tables, if not exists

Copy initial data

Replicate changes (insert, **update**, delete, truncate)

Periodically flush change batches to Iceberg

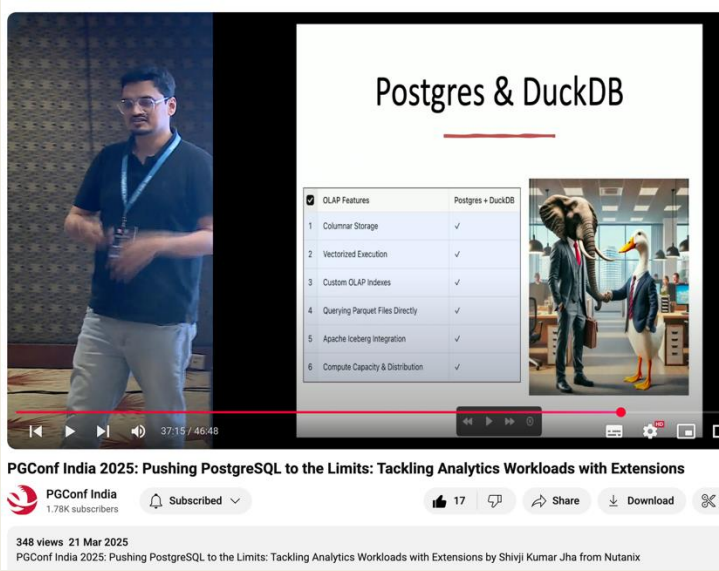
Automatic compaction

Available today on Crunchy Bridge!

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 <u>just love</u> hacking open-source code early* ☺
  - *Much easier to become a contributor* ***now****!*

THANK YOU!

Q&A

linkedin.com/in/shivjijha

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