CDMS 2022

9. September 2022

# DuckDB
# A Modern Modular & Extensible Database System

Mark Raasveldt

DuckDB Labs

# Who Am I?

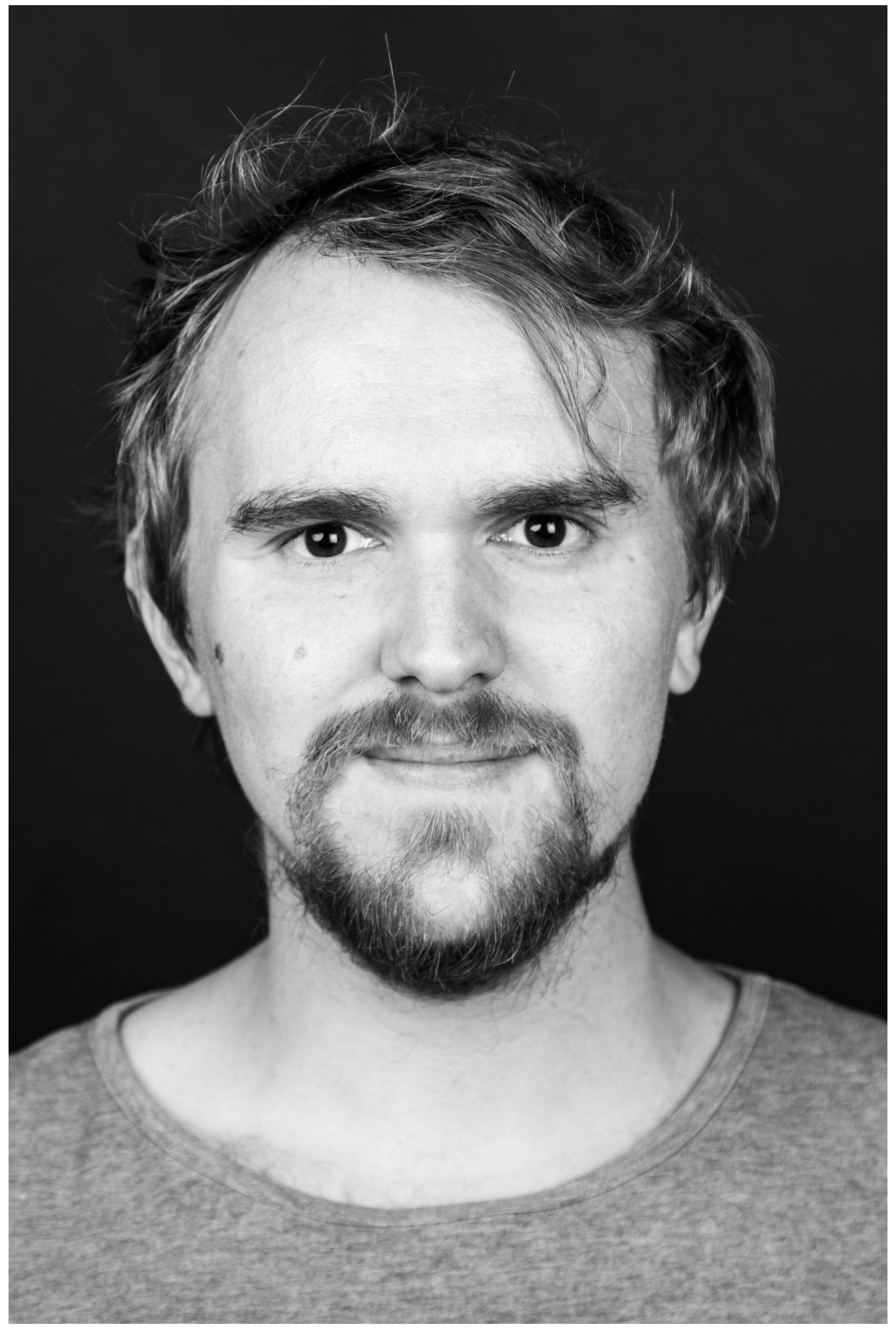
Mark Raasveldt

CTO of DuckDB Labs

Postdoc at CWI

Database Architectures Group

PhD at CWI



# What is DuckDB?

- DuckDB
- In-Process OLAP DBMS
    - “The SQLite for Analytics”

- Free and Open Source (MIT)
- duckdb.org

# What is DuckDB?

- SQLite inspired us in many ways:
  - Easy installation
  - Ease of use
  - Robustness

- DuckDB aims to be the “SQLite for Analytics”

# Data Analytics

Analytics has **unique challenges**

Transactional workloads = **simple queries**

SQLite can be feature complete with a very small footprint

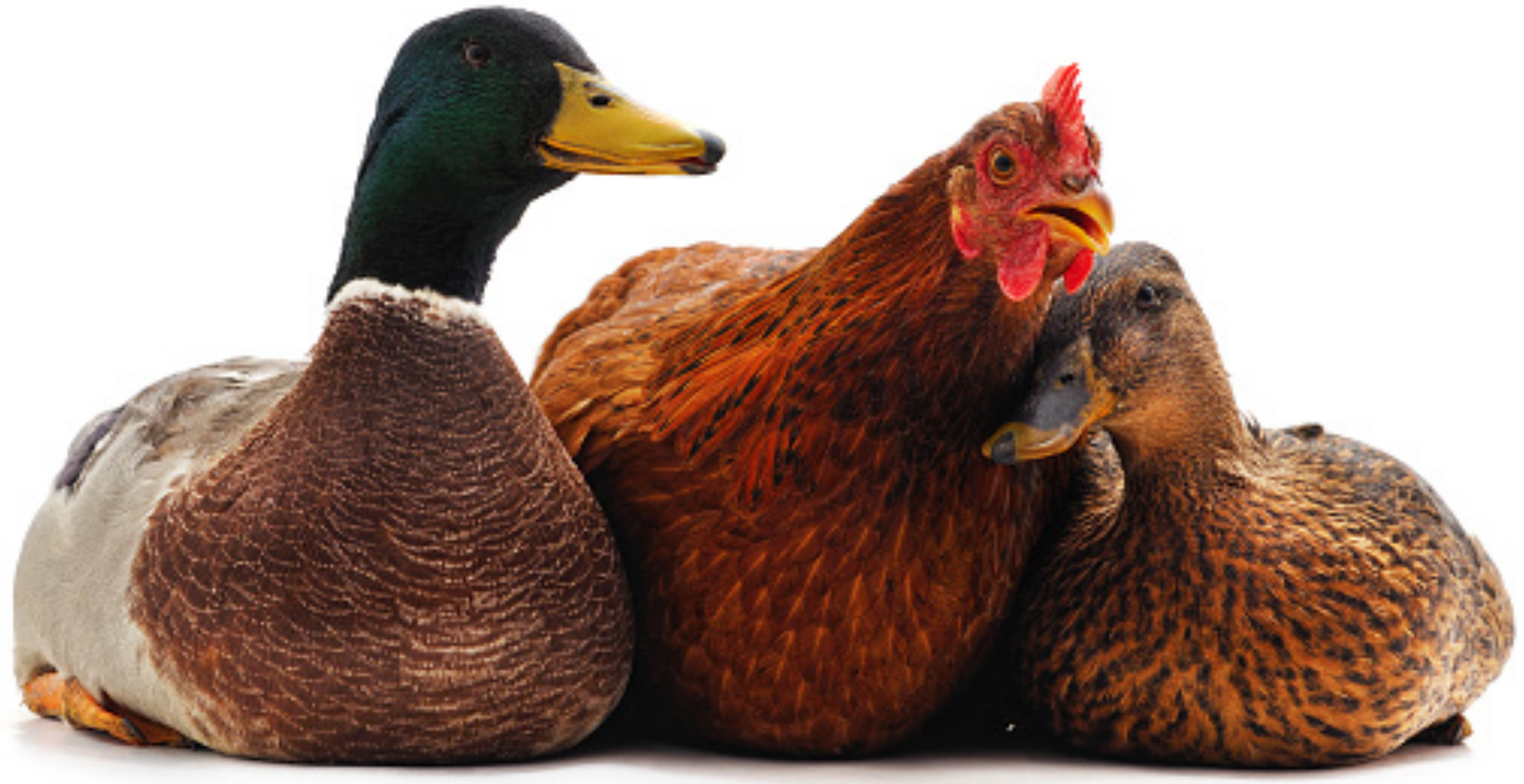
Analytical workloads = **complex queries**

DuckDB needs **many more** operations, functions, optimizers,….

# Collaboration in Analytics

Analytics requires a giant diversity of operations

**Collaboration is required!**



A single entity cannot hope to implement:

Data wrangling tools

Classification algorithms, ML toolkits

Data cleaning tools

etc…

# Collaboration in Analytics

**Research** other important collaboration area

DuckDB **originates from** research world

At CWI and elsewhere, people do their research in DuckDB

Important to allow **extending/modifying system**

e.g. add new join operator, new optimizer, …

# Collaboration in Analytics

**System builders** want to use **components** of DuckDB

For example:

- Use only the front-end
- Use only the back-end

# Collaboration in Analytics

How do we **enable this collaboration**?

Three aspects:

- Flexible **data import and export**
- **Extensibility** of the system
- Hooks in different locations in the system

# Flexible Data Import & Export

# Data Import & Export

## Data import and export

Crucial for communication **between libraries**

## Many use cases

Load data exported from other systems

Pre-process in DuckDB→ use plotting/statistics/ML libraries

Mix usage DuckDB + other data wrangling libraries

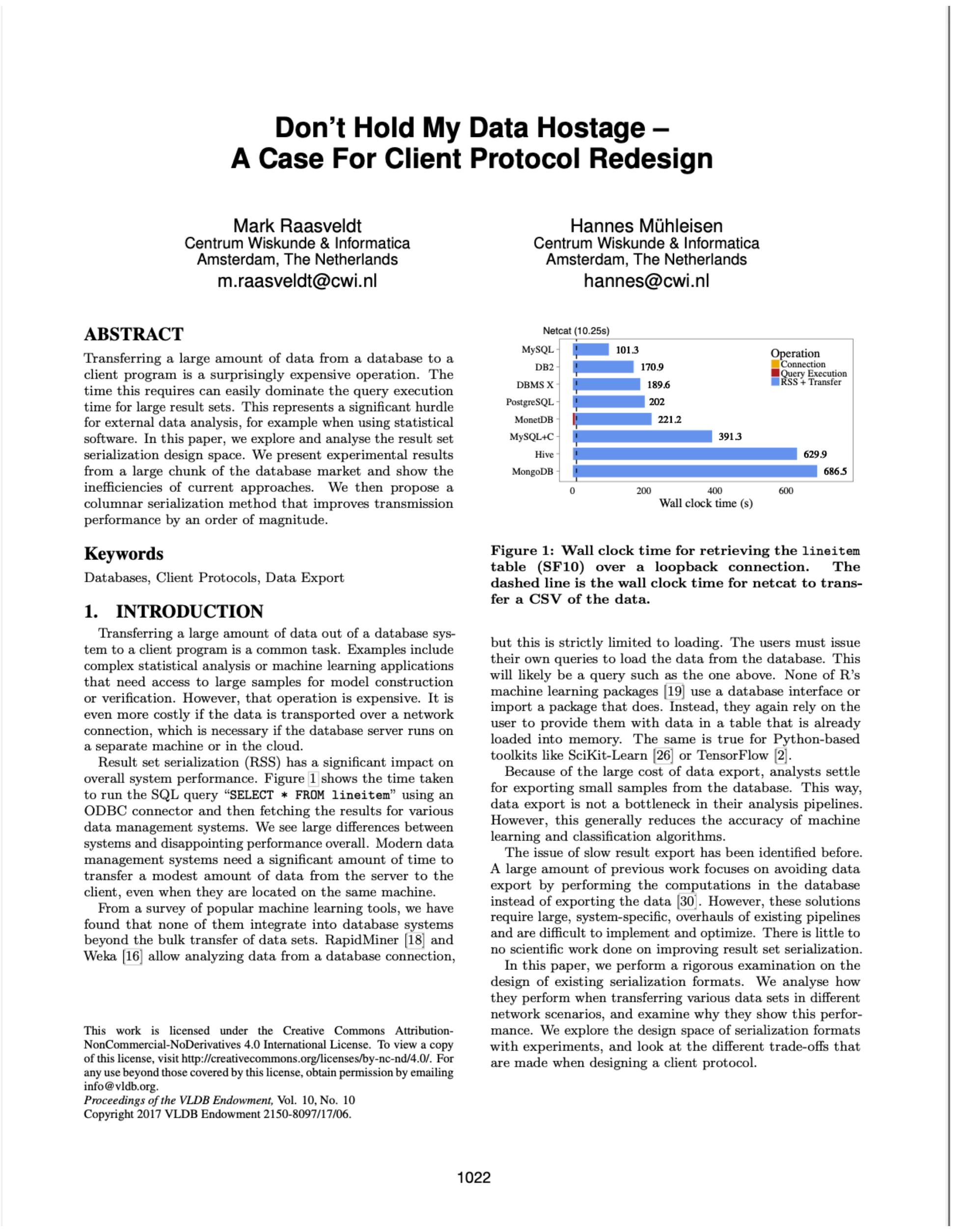
Export data to persistent storage (e.g. Parquet files on S3)

# Data Import & Export

What makes DuckDB different?

All database systems can import/export data

**… but very slowly!**



## Don’t Hold My Data Hostage – A Case For Client Protocol Redesign

Mark Raasveldt
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
m.raasveldt@cwi.nl

Hannes Mühleisen
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
hannes@cwi.nl

**ABSTRACT**

Transferring a large amount of data from a database to a client program is a surprisingly expensive operation. The time this requires can easily dominate the query execution time for large result sets. This represents a significant hurdle for external data analysis, for example when using statistical software. In this paper, we explore and analyse the result set serialization design space. We present experimental results from a large chunk of the database market and show the inefficiencies of current approaches. We then propose a columnar serialization method that improves transmission performance by an order of magnitude.

**Keywords**

Databases, Client Protocols, Data Export

### 1. INTRODUCTION

Transferring a large amount of data out of a database system to a client program is a common task. Examples include complex statistical analysis or machine learning applications that need access to large samples for model construction or verification. However, that operation is expensive. It is even more costly if the data is transported over a network connection, which is necessary if the database server runs on a separate machine or in the cloud.

Result set serialization (RSS) has a significant impact on overall system performance. Figure 1 shows the time taken to run the SQL query “`SELECT * FROM lineitem`” using an ODBC connector and then fetching the results for various data management systems. We see large differences between systems and disappointing performance overall. Modern data management systems need a significant amount of time to transfer a modest amount of data from the server to the client, even when they are located on the same machine.

From a survey of popular machine learning tools, we have found that none of them integrate into database systems beyond the bulk transfer of data sets. RapidMiner [18] and Weka [16] allow analyzing data from a database connection, but this is strictly limited to loading. The users must issue their own queries to load the data from the database. This will likely be a query such as the one above. None of R’s machine learning packages [19] use a database interface or import a package that does. Instead, they again rely on the user to provide them with data in a table that is already loaded into memory. The same is true for Python-based toolkits like SciKit-Learn [26] or TensorFlow [2].

Because of the large cost of data export, analysts settle for exporting small samples from the database. This way, data export is not a bottleneck in their analysis pipelines. However, this generally reduces the accuracy of machine learning and classification algorithms.

The issue of slow result export has been identified before. A large amount of previous work focuses on avoiding data export by performing the computations in the database instead of exporting the data [30]. However, these solutions require large, system-specific, overhauls of existing pipelines and are difficult to implement and optimize. There is little to no scientific work done on improving result set serialization.

In this paper, we perform a rigorous examination on the design of existing serialization formats. We analyse how they perform when transferring various data sets in different network scenarios, and examine why they show this performance. We explore the design space of serialization formats with experiments, and look at the different trade-offs that are made when designing a client protocol.

**Figure 1: Wall clock time for retrieving the `lineitem` table (SF10) over a loopback connection. The dashed line is the wall clock time for netcat to transfer a CSV of the data.**

*Proceedings of the VLDB Endowment,* Vol. 10, No. 10
Copyright 2017 VLDB Endowment 2150-8097/17/06.

Don’t Hold My Data Hostage -
A Case For Client Protocol Redesign

Ancient protocols, designed to transfer kilobytes of data

Unsuitable for modern workloads!

Import/export through text-based formats (CSV files)

# Data Import & Export

DuckDB is designed for **bulk data export/import**

- **In-process = zero-copy data sharing**

Versatile **input & output APIs**

**Same API** used for both internal tables & external sources

- Allows for **streaming data**, **parallel scans, projection & filter pushdown, index usage, …**

# Data Import & Export

DuckDB can **efficiently** consume and output **many formats**

pandas

R

julia

NumPy

Parquet

CSV

{JSON}

APACHE ARROW

SQLite

PostgreSQL

# Extensibility of the System

# Extensibility of the System

**Extensibility of the system**

DuckDB supports **extensions** that add new functionality

Allows users to **integrate new functionality in the system**

# Extensibility of the System

We create extensions ourselves liberally

**Eat your own duck food**

Allows us to add functionality…

…without **bloating the core system**

…that is **only included in certain distributions** (e.g. Python client)

…without introducing **external dependencies** to the core

# Extensibility of the System

## ICU Extension

- Adds support for **collations (language-based sorting, comparisons)**
- Adds support for **time-zone awareness**

Extension is **several megabytes**

- **Includes ICU localization data**
- Roughly same size as DuckDB core!

**As an extension we keep this optional**

# Extensibility of the System

## HTTP-FS Extension

Adds support for **reading/writing data over HTTP(S)**

Adds support for **reading/writing data with S3**

Adds **dependency on OpenSSL**

As an extension, we **keep the core dependency-free**

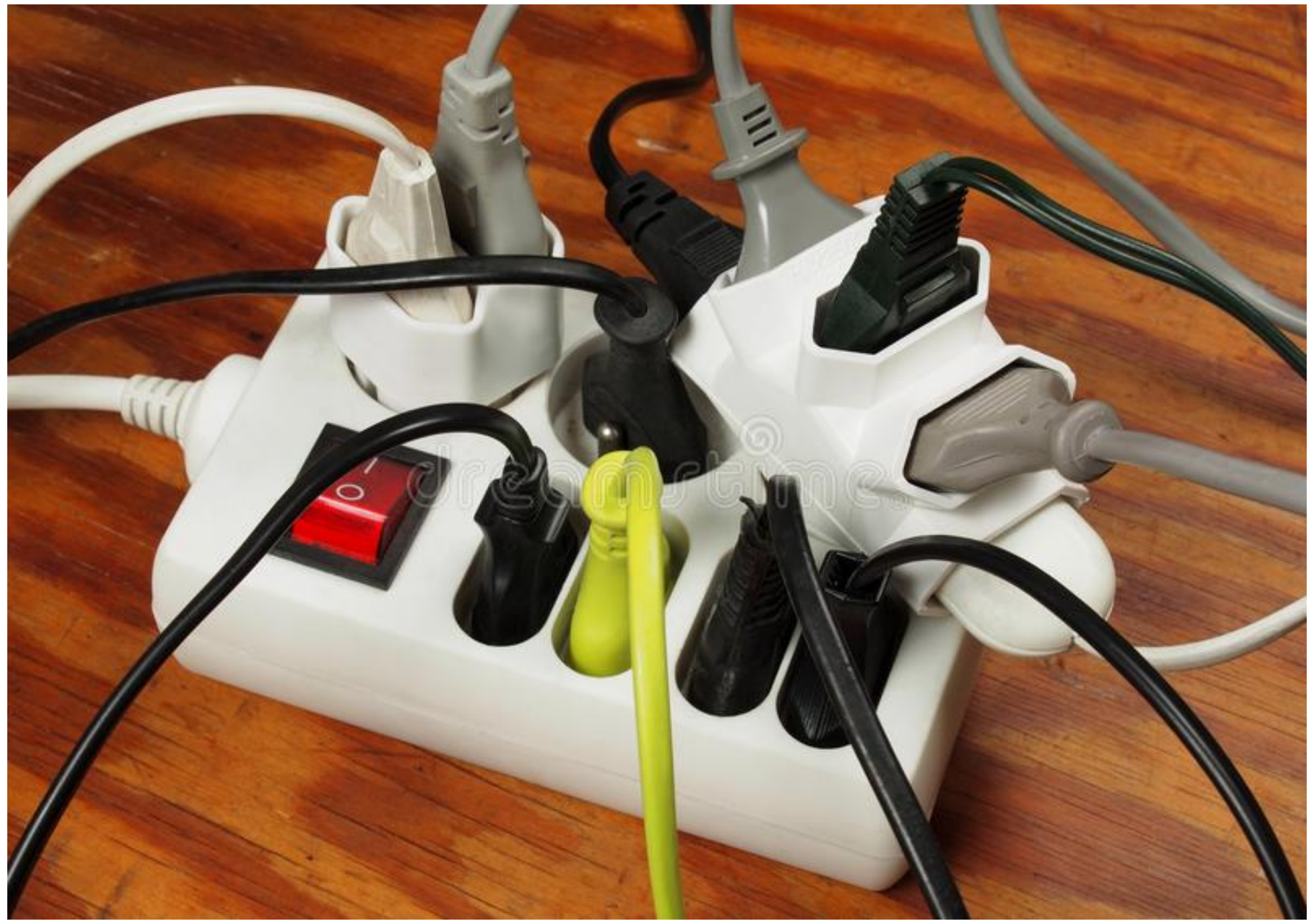
# Extensibility of the System

Extensions are **powerful**

- **Goal:** Allow **every component of the system** to be extended

Currently extensions can:

- Add functions (scalar, aggregate, window)
- Add new types
- Add data sources and sinks
- Add collations, time zones
- Add parser functionality, optimizers



# Extensibility of the System

**Goal:** allow users to **create and maintain their own extensions**

No need to **talk to us**

No need for us **to maintain their code**

**Extension Repository**

Extensions can be installed using SQL

```
INSTALL httpfs;
LOAD httpfs;
```

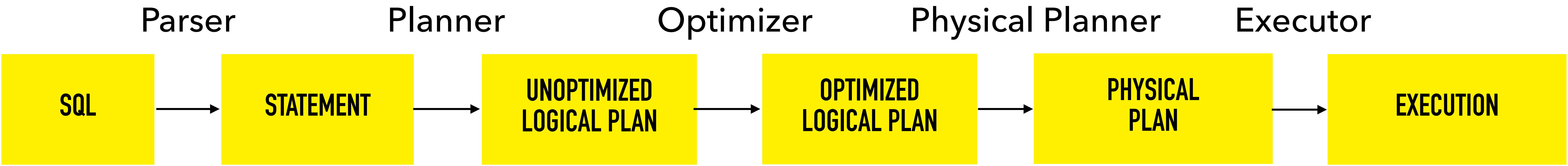
By default from our repository

Custom repository can be used

# Different Hooks Into The System

# Different Hooks Into The System

DuckDB uses a typical pipeline for query processing



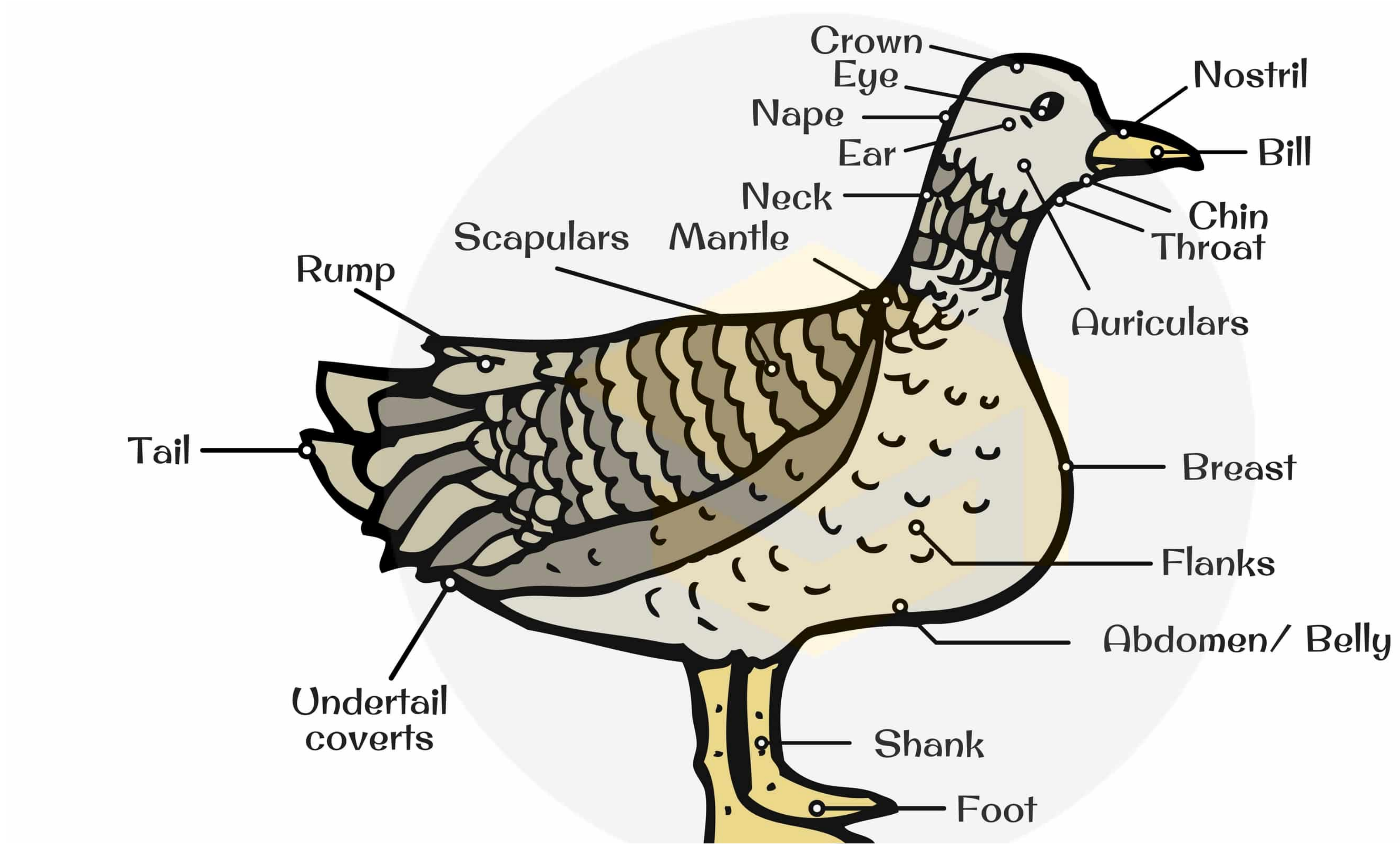
**Standard workflow: use all components**

# Different Hooks Into The System

**Standard pipeline: SQL input → Data output**

**System builders/researchers** often want to use **parts of DuckDB**

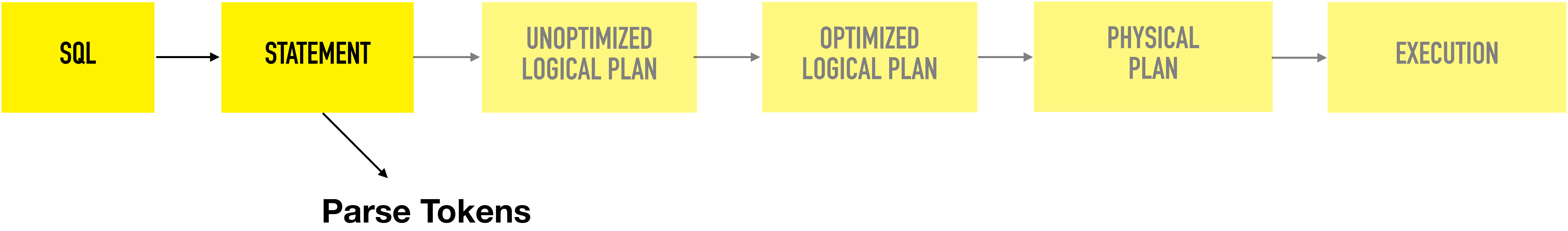
**Different hooks in/out of the system**



# Different Hooks Into The System

Velox: to use the parser

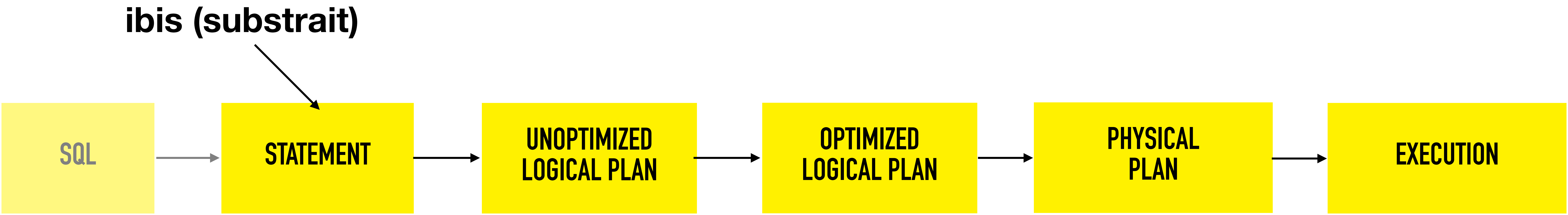
**Parser pipeline: SQL Input → Parse Tokens Output**



# Different Hooks Into The System

Ibis: provide a **new front-end to DuckDB**

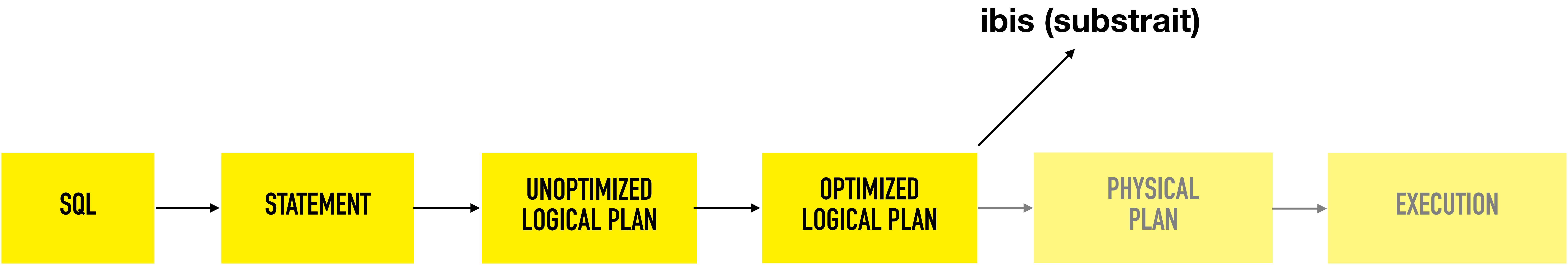
**Ibis front-end pipeline: Substrait plan input → Data Output**



# Different Hooks Into The System

Ibis: **consume SQL** and use a different back-end

**Ibis back-end pipeline: SQL input → Substrait plan output**

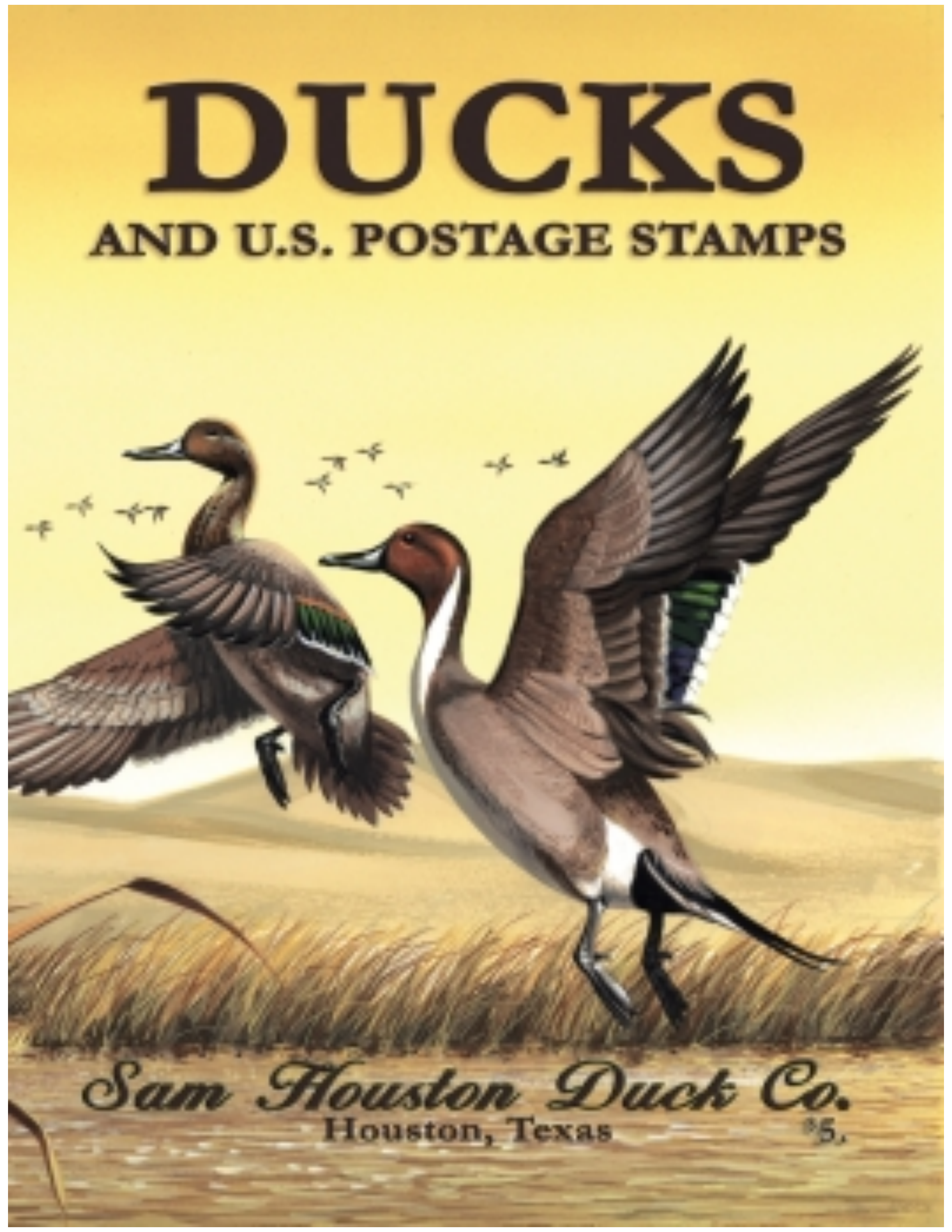


# Future Extensibility Plans

# Future Extensibility Plans

**Goal:** more **extensibility** options

- New index types
- Alternate versions of existing operators (e.g. new joins)
- Define new casts between types



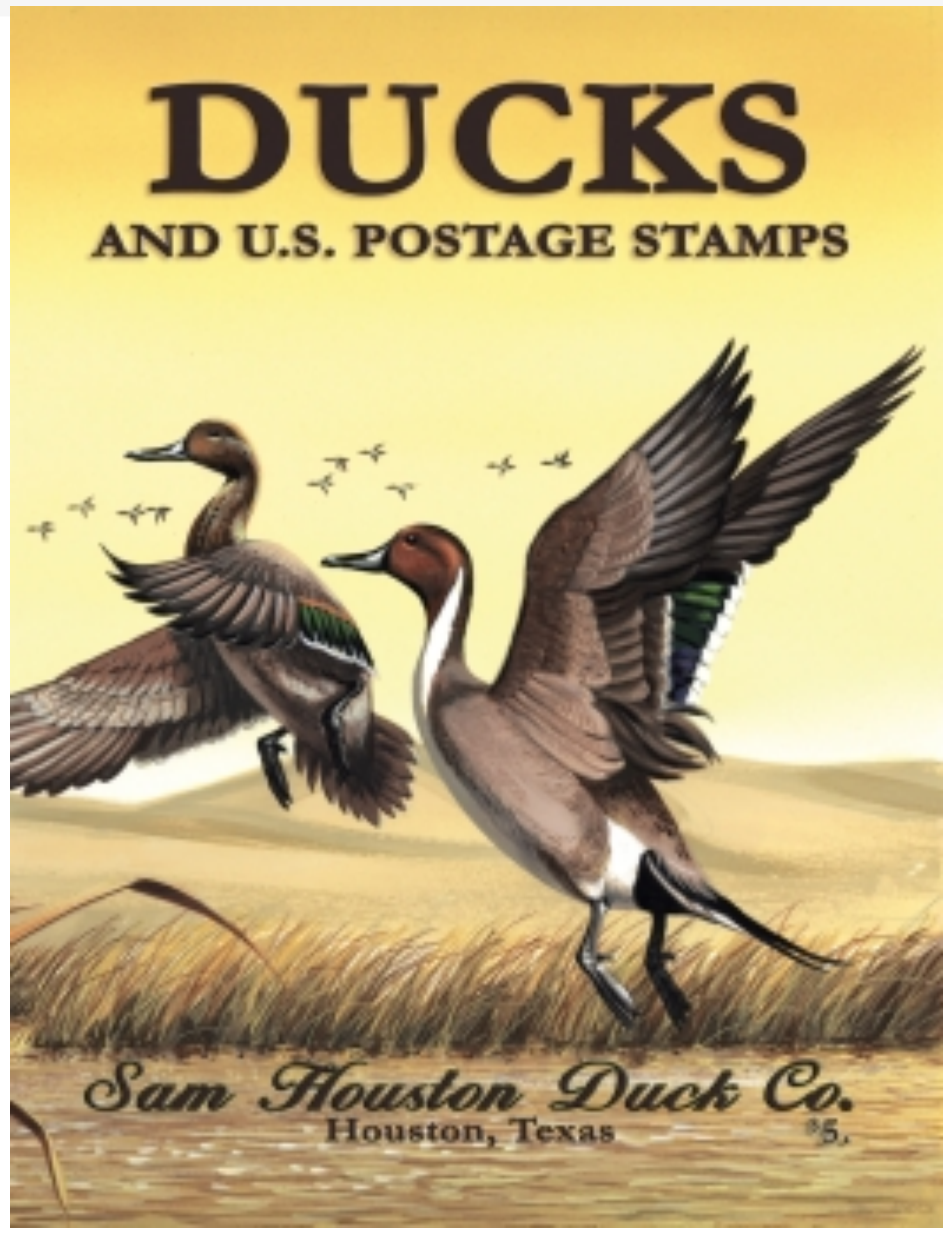
# Future Extensibility Plans

**Goal:** support fully custom storage back-end

Currently support **data-input** and **copy output**

**Missing support** for routing **insert, update, delete** to custom storage

# Future Extensibility Plans



**Goal:** extended custom **catalog** support

Currently support **replacement scans**

- This allows for **views** to be stored in a different catalog

Needs to be extended to fully support custom catalogs

- Including for all **different catalog types**

Thanks for having me!

Any questions?