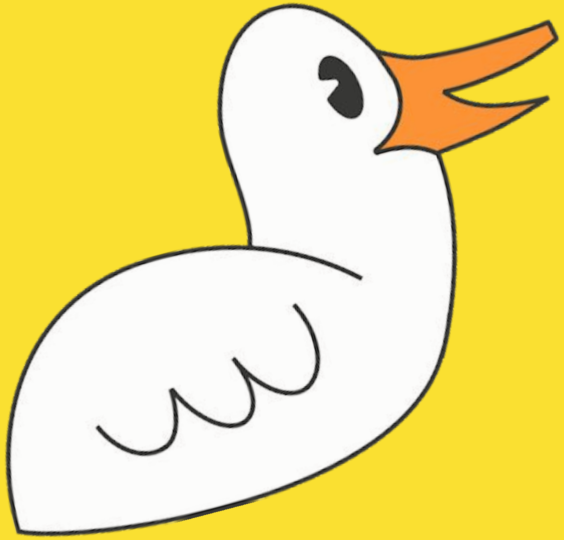
60fps cloud databases



Peter Boncz
(CWI & MotherDuck)

mosaic

Search ⌘K

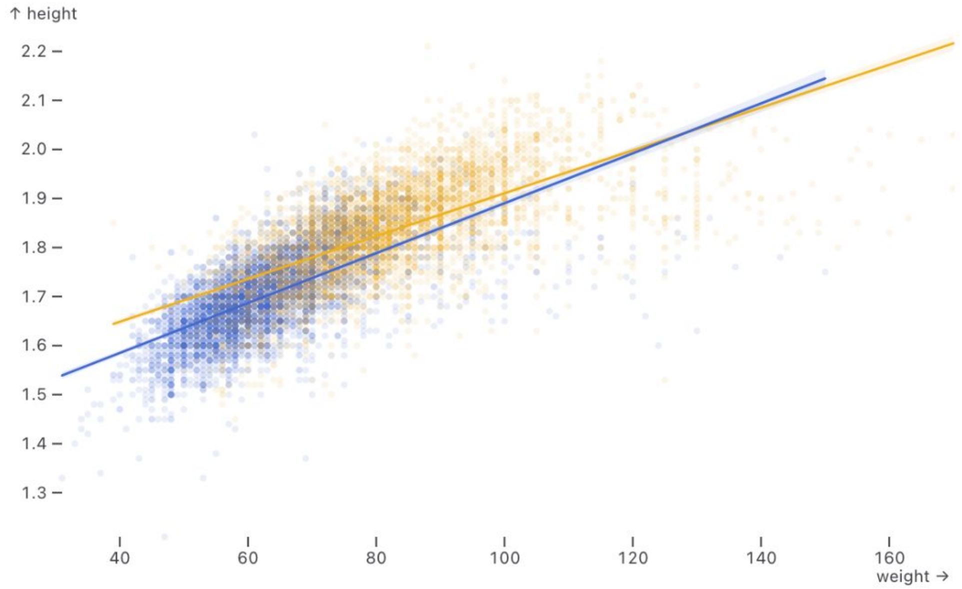
Overview Guide

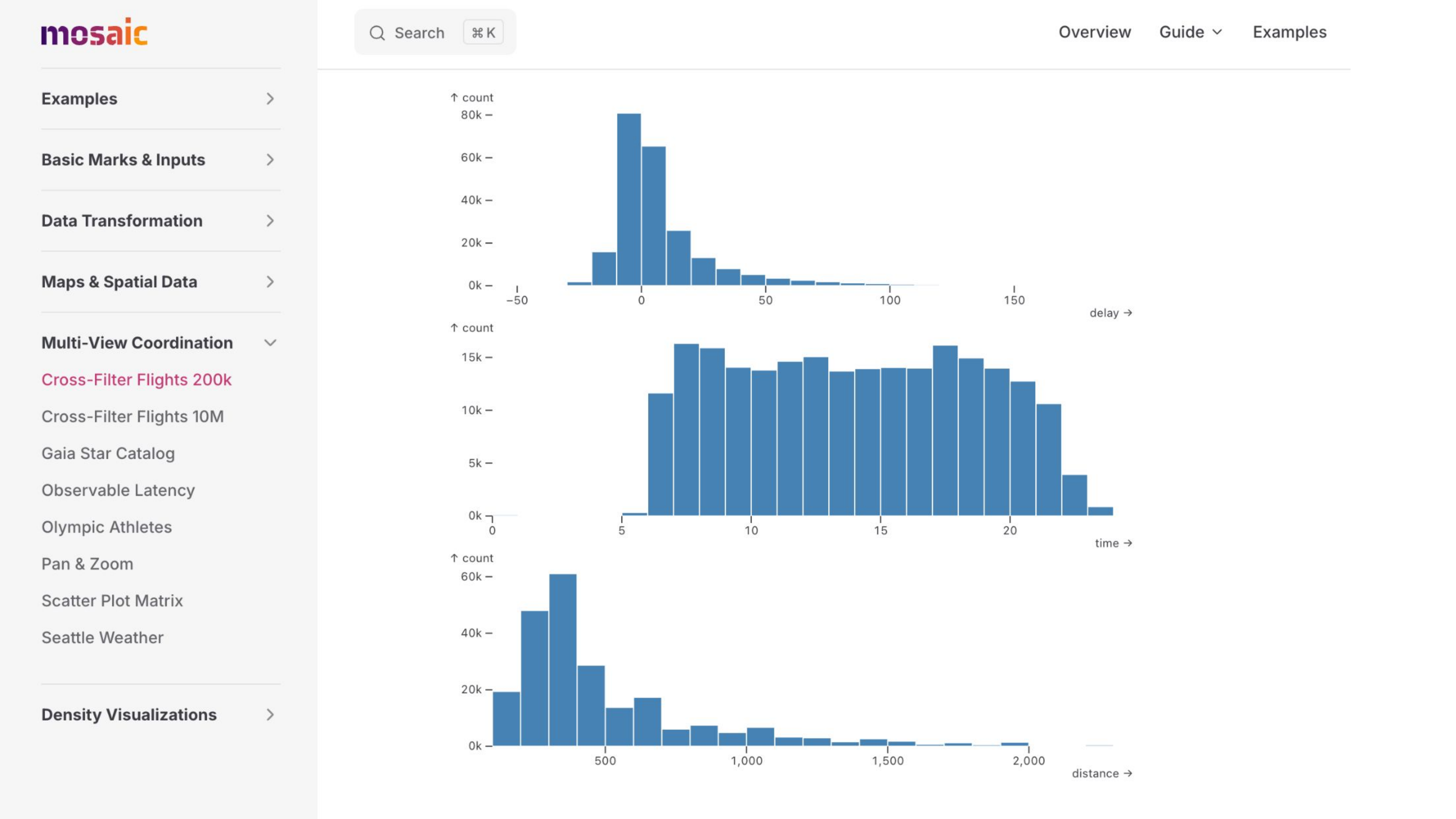
- Examples
- Basic Marks & Inputs
- Data Transformation
- Maps & Spatial Data
- Multi-View Coordination
  - Cross-Filter Flights 200k
  - Cross-Filter Flights 10M
  - Gaia Star Catalog
  - Observable Latency
  - Olympic Athletes
  - Pan & Zoom
  - Scatter Plot Matrix
  - Seattle Weather
- Density Visualizations

Olympic Athletes

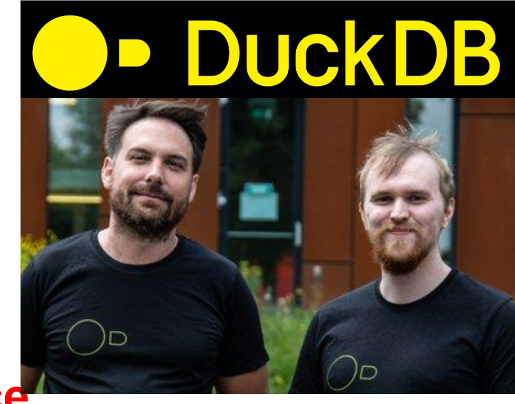
An interactive dashboard of athlete statistics. The menus and searchbox filter the display and are automatically populated by backing data columns.

Sport: All — Sex: All — Name: Query




mosaic
Search ⌘K
Overview
Guide
Examples
Examples
Basic Marks & Inputs
Data Transformation
Maps & Spatial Data
Multi-View Coordination
Cross-Filter Flights 200k
Cross-Filter Flights 10M
Gaia Star Catalog
Observable Latency
Olympic Athletes
Pan & Zoom
Scatter Plot Matrix
Seattle Weather
Density Visualizations
↑ count
80k
60k
40k
20k
0k
−50
0
50
100
150
delay →
↑ count
15k
10k
5k
0k
0
5
10
15
20
time →
↑ count
60k
40k
20k
0k
500
1,000
1,500
2,000
distance →

DuckDB: embedded analytics



- Created by Hannes Mühleisen and Mark Raasveldt
- Idea: **analytical SQL system as a linkable library**
- From research on **data systems support for data science.**
  - why don't data scientists use database systems?
    ⇒ make database technology better suited for data science
    Embedded databases, zero-copy dataframe access, ease-of-use
- Active discord, blog, starting events, traction:
  - >**22K** github stars, >**10M** downloads/month (**4x** increases YoYoYoY)
  - DuckDB Labs spin-off (+MotherDuck)

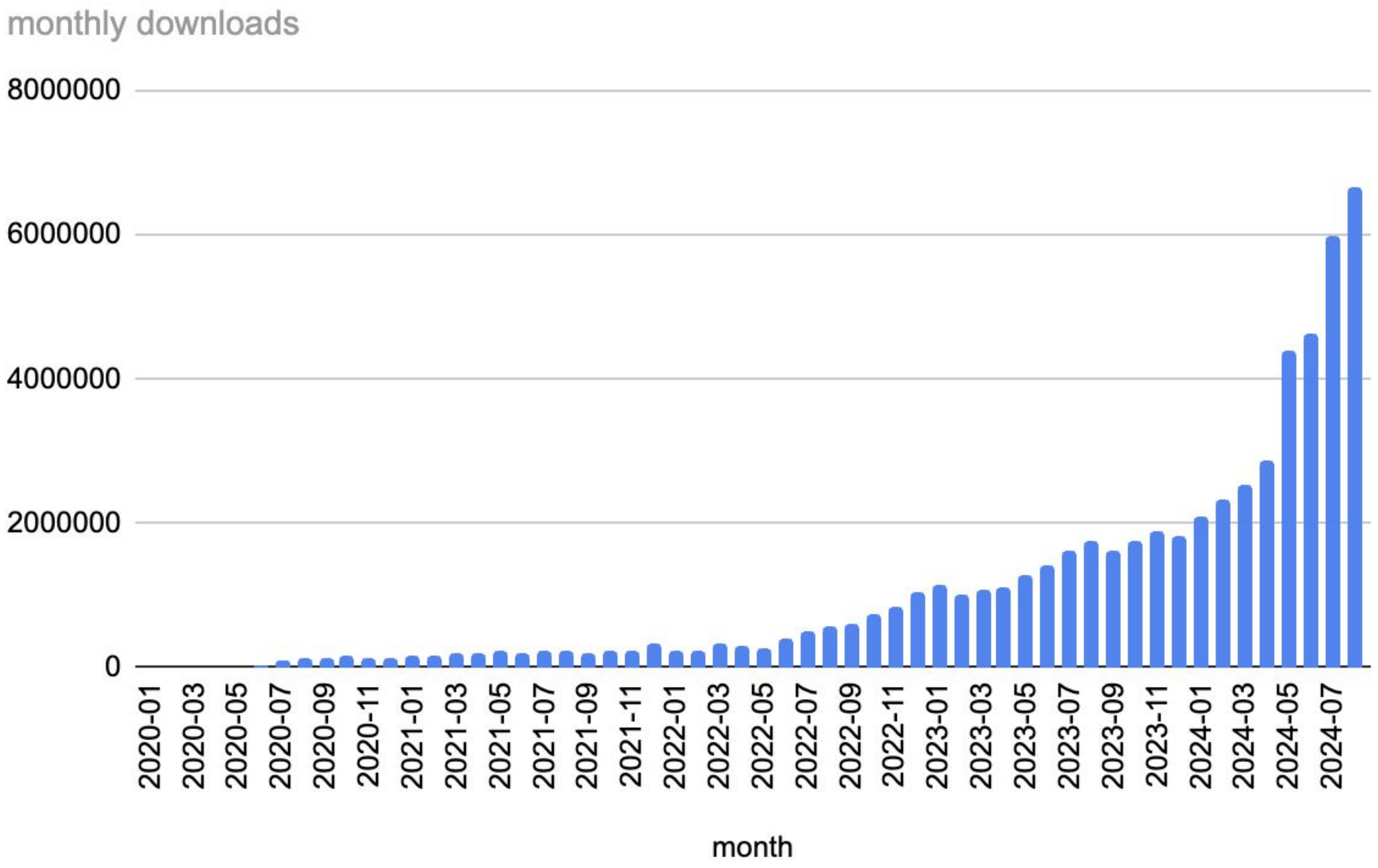
https://duckdb.org/ https://shell.duckdb.org

MAKING ANALYTICS EASY

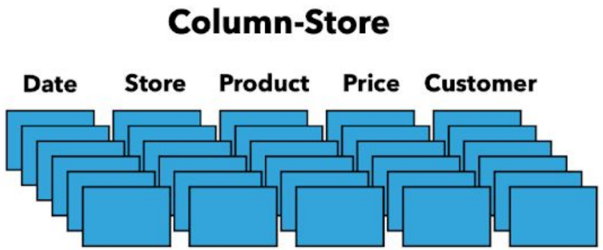
A great burger is more than just good beef!

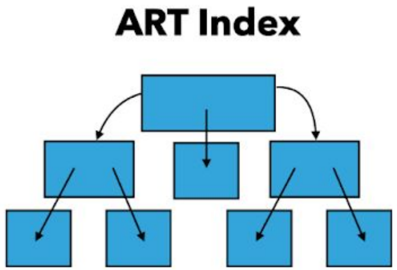
Easy to install / no dependencies
Run anywhere (including the browser)
Query dataframes directly
Friendliest SQL syntax in the world

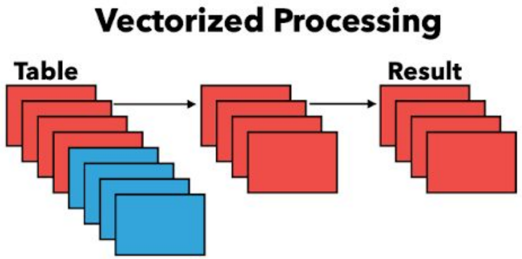
DuckDB package (PyPi)

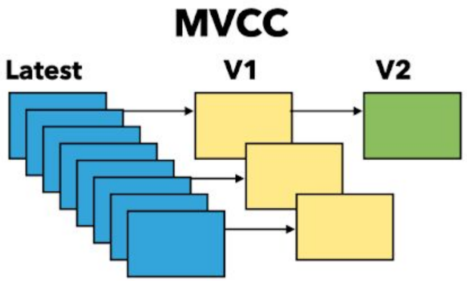


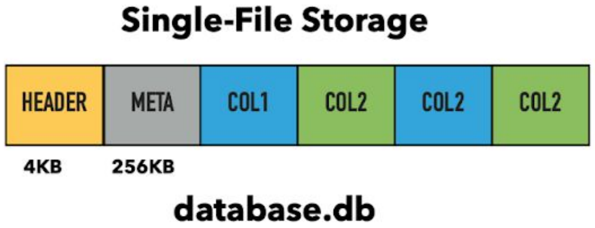
DuckDB - overview





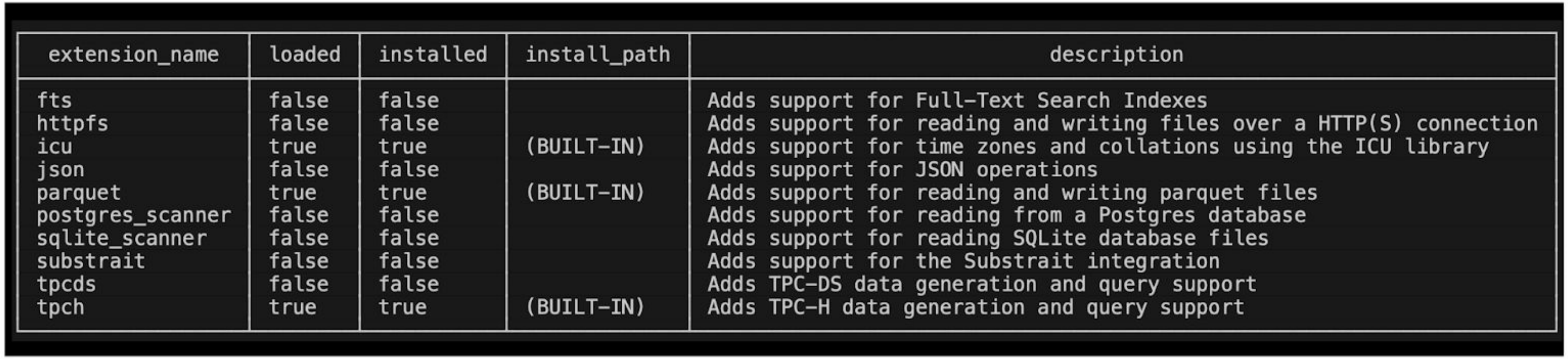






DuckDB - Extensions

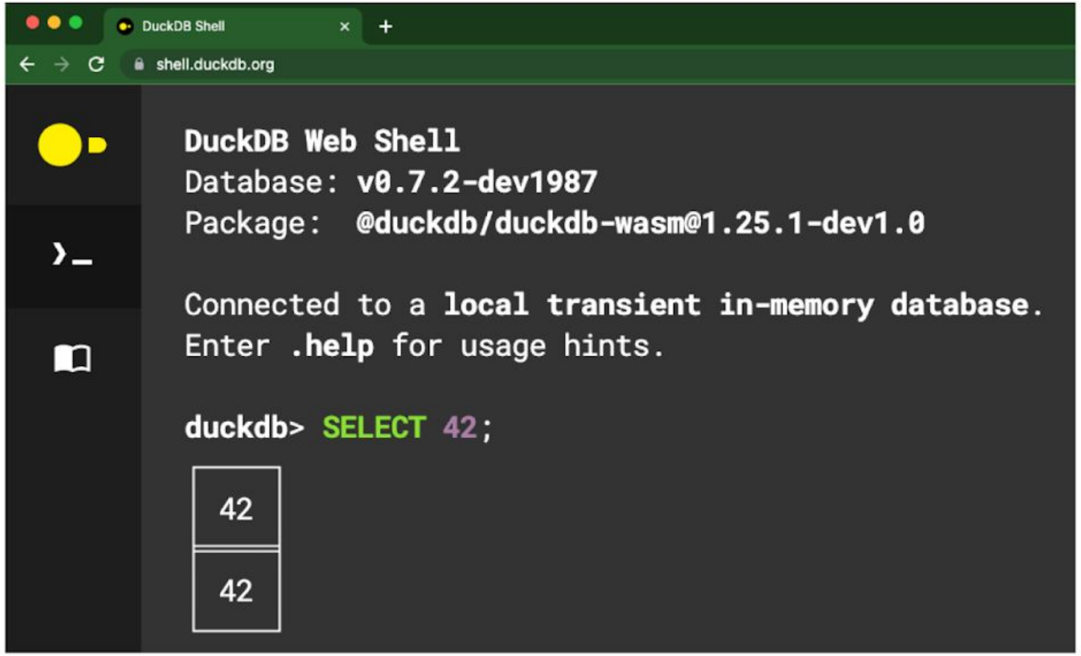
- DuckDB offers support for **extensions**
- Distributed through **INSTALL** and **LOAD** commands
  - Can be loaded as a **shared library**
- Many of our core features are implemented as extensions



```
┌─────────────────┬────────┬───────────┬──────────────┬──────────────────────────────────────────────────────────────┐
│ extension_name  │ loaded │ installed │ install_path │                         description                          │
├─────────────────┼────────┼───────────┼──────────────┼──────────────────────────────────────────────────────────────┤
│ fts             │ false  │ false     │              │ Adds support for Full-Text Search Indexes                    │
│ httpfs          │ false  │ false     │              │ Adds support for reading and writing files over a HTTP(S) connection │
│ icu             │ true   │ true      │ (BUILT-IN)   │ Adds support for time zones and collations using the ICU library │
│ json            │ false  │ false     │              │ Adds support for JSON operations                             │
│ parquet         │ true   │ true      │ (BUILT-IN)   │ Adds support for reading and writing parquet files           │
│ postgres_scanner│ false  │ false     │              │ Adds support for reading from a Postgres database            │
│ sqlite_scanner  │ false  │ false     │              │ Adds support for reading SQLite database files               │
│ substrait       │ false  │ false     │              │ Adds support for the Substrait integration                   │
│ tpcds           │ false  │ false     │              │ Adds TPC-DS data generation and query support                │
│ tpch            │ true   │ true      │ (BUILT-IN)   │ Adds TPC-H data generation and query support                 │
└─────────────────┴────────┴───────────┴──────────────┴──────────────────────────────────────────────────────────────┘
```

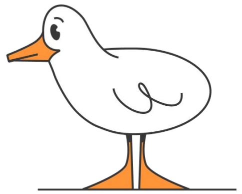
DuckDB - WASM

- DuckDB has a **WASM build**
- Runs inside the browser
    - And it is actually fast!



```
DuckDB Web Shell
Database: v0.7.2-dev1987
Package:  @duckdb/duckdb-wasm@1.25.1-dev1.0

Connected to a local transient in-memory database.
Enter .help for usage hints.

duckdb> SELECT 42;
┌────┐
│ 42 │
├────┤
│ 42 │
└────┘
```

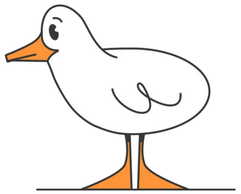
MotherDuck



What is MotherDuck?

A serverless DuckDB platform for low-cost, low-latency analytics that combines the power of your laptop and the modern cloud.

With **Dual Query Processing!**



What is **MotherDuck**?

A serverless DuckDB platform for low-cost, low-latency analytics that combines the power of your laptop and the modern cloud.

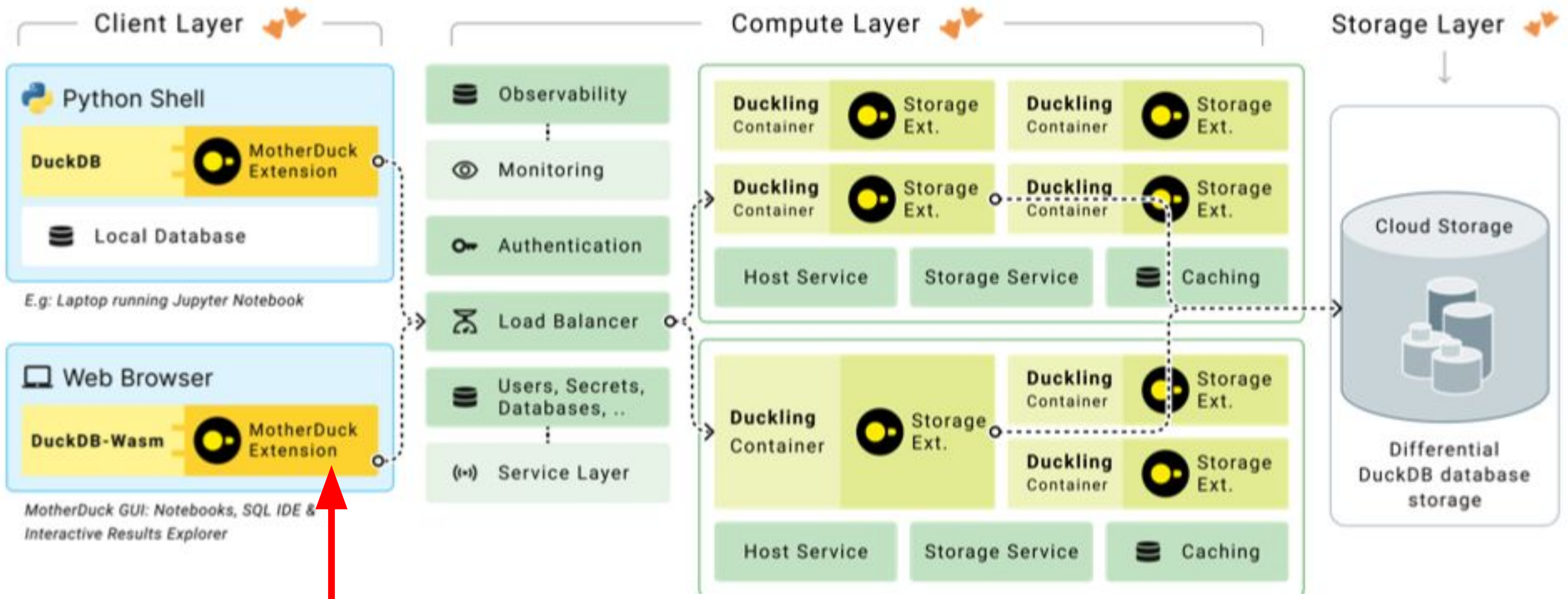
MotherDuck

What is Dual (Hybrid) Query Processing?

- Every client has a DuckDB
  - DuckDB is an embedded DBMS
  - So.. JDBC driver links DuckDB into your application
- Every DuckDB client can contact MotherDuck
  - `install motherduck;`
  - `load motherduck;`
- Local Databases and Remote Databases
  - Can be queried as one
  - Some execution local, some in the cloud

MotherDuck

MotherDuck Architecture



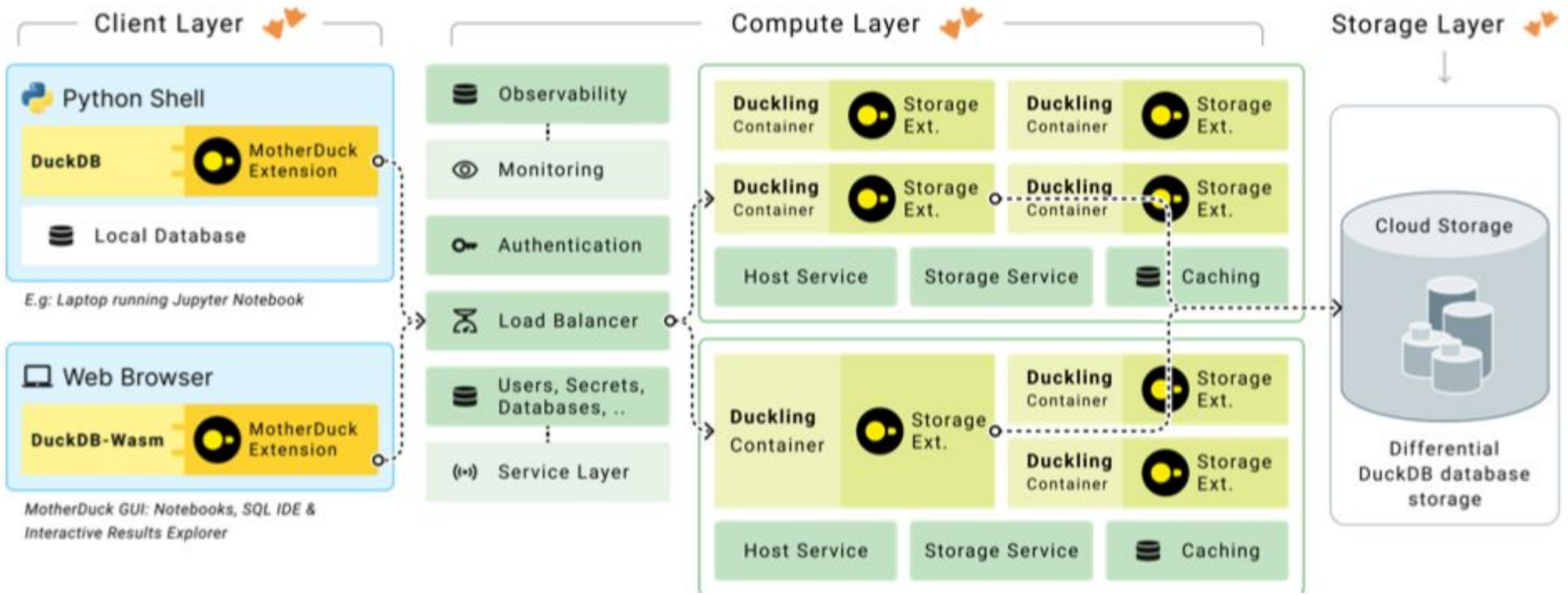
Client Extension

Why Dual Query Processing?

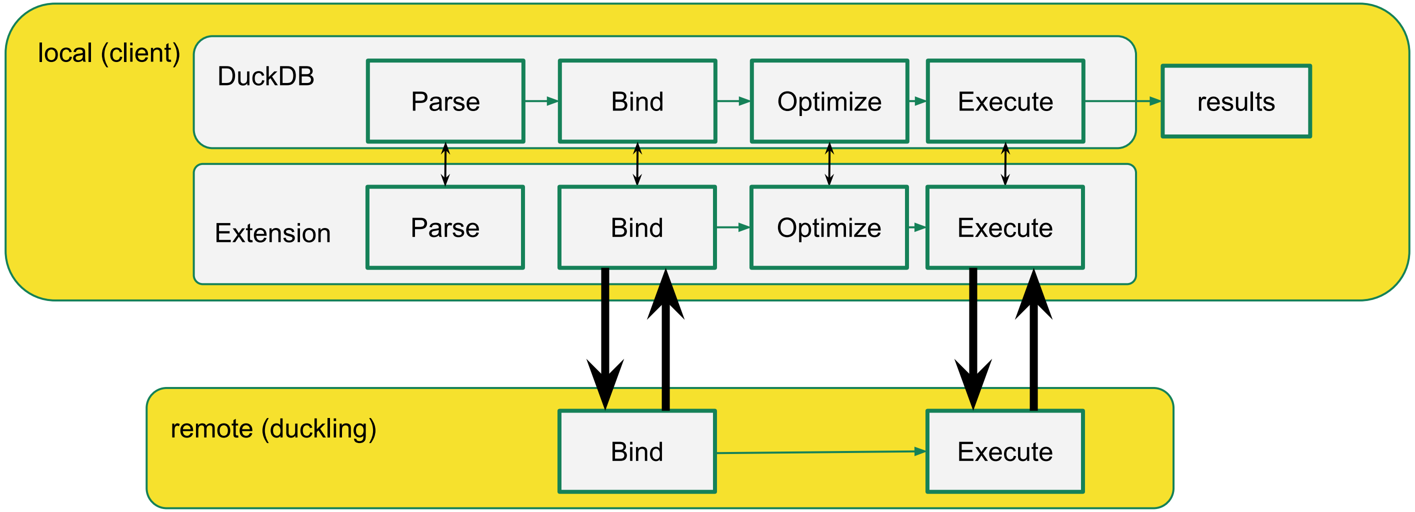
- Applications enabled by client-side queries
  - low latency: dashboards, interactive query formulation, spatial compute
  - not always connected applications (edge)
  - secure applications: decrypt on client (monomi)
- Reduce Cloud Compute
  - Leverage local compute resources
- Moving data science from laptop to cloud
  - Share data, bring pipelines in production
- Run PostgreSQL analytics on DuckDB (pg_duckdb)
  - ..backed by MotherDuck cloud storage

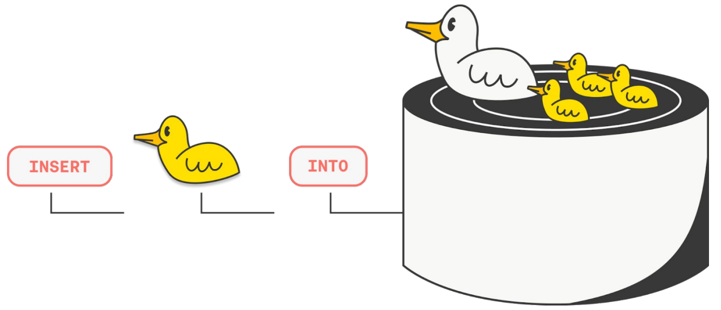
MotherDuck

MotherDuck Architecture

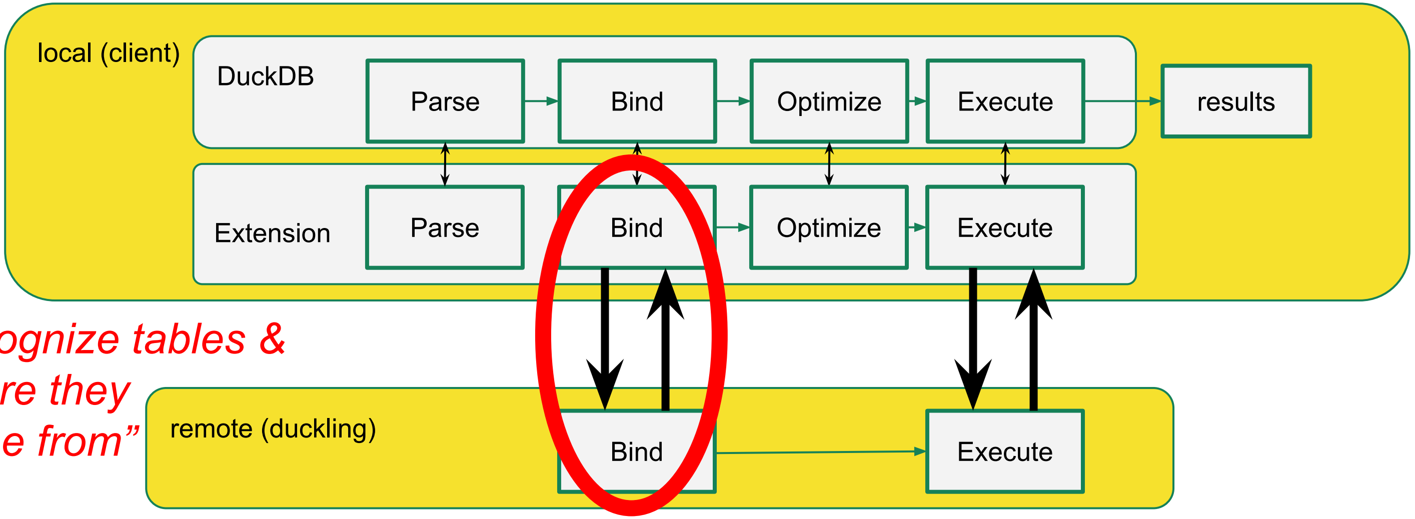


Dual Query Processing



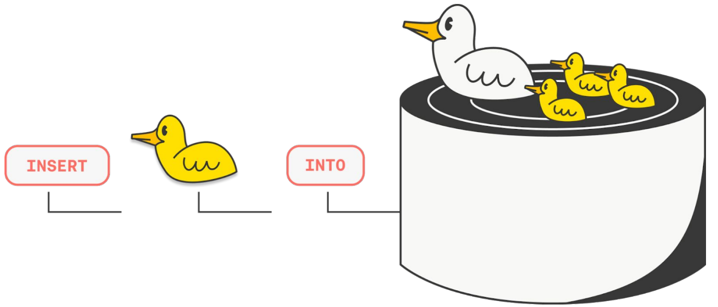


Dual Query Processing



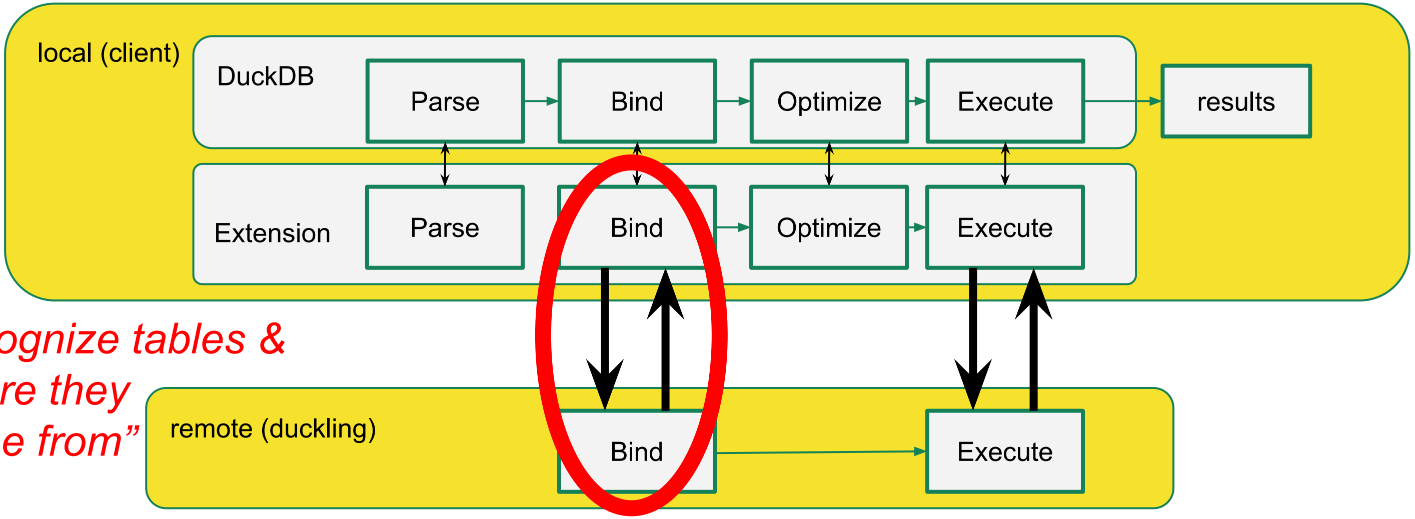
"recognize tables & where they come from"

```
FROM local_db.tab t SELECT sum(t.c)
```



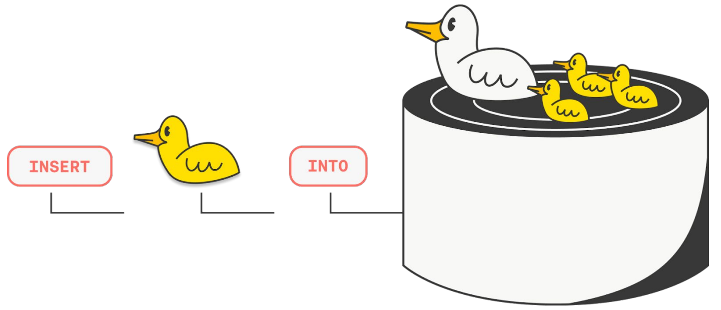
MotherDuck

Dual Query Processing

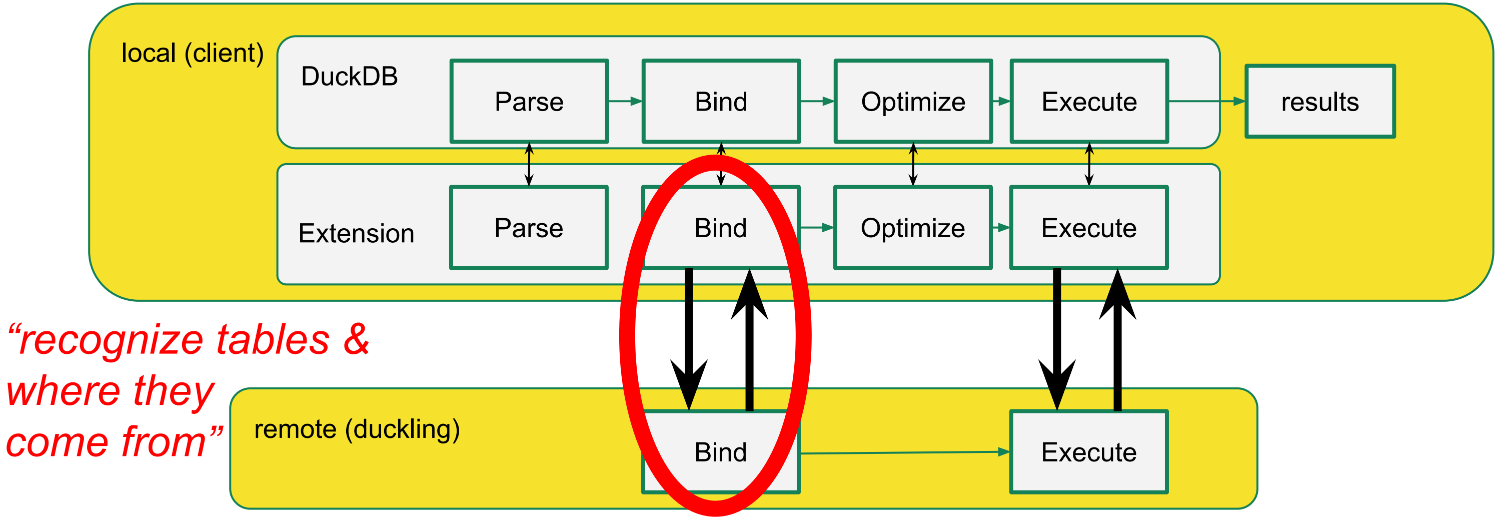


"recognize tables & where they come from"

```
FROM remote_db.tab t SELECT sum(t.c)
```

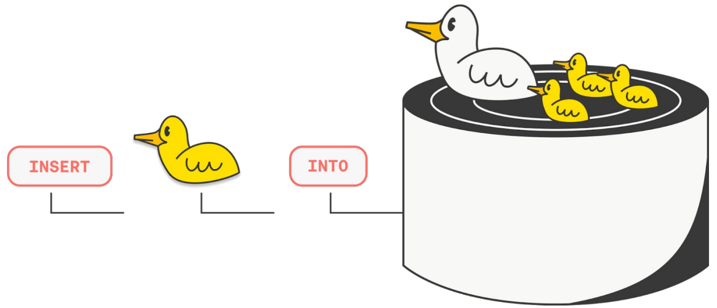


Dual Query Processing



```
FROM remote_db.tab t SELECT sum(t.c)
```

Virtual catalog

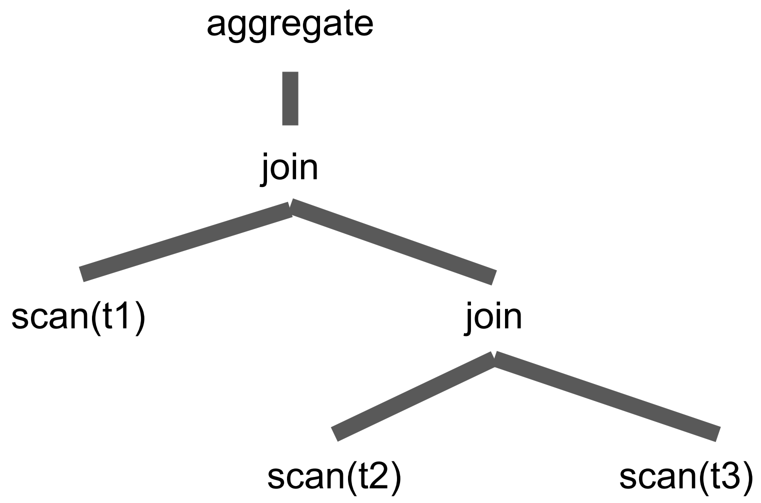


Query Pipelines

```
SELECT count(*) FROM t1 JOIN t2 ON t1.a = t2.a  JOIN t3 ON t3.b = t2.b
```

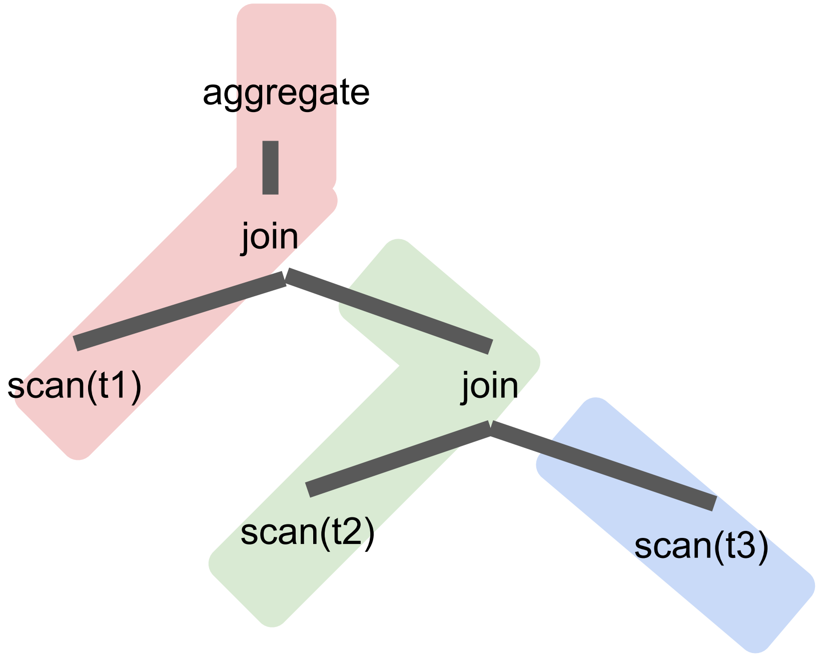
Query Pipelines

```
SELECT count(*) FROM t1 JOIN t2 ON t1.a = t2.a  JOIN t3 ON t3.b = t2.b
```

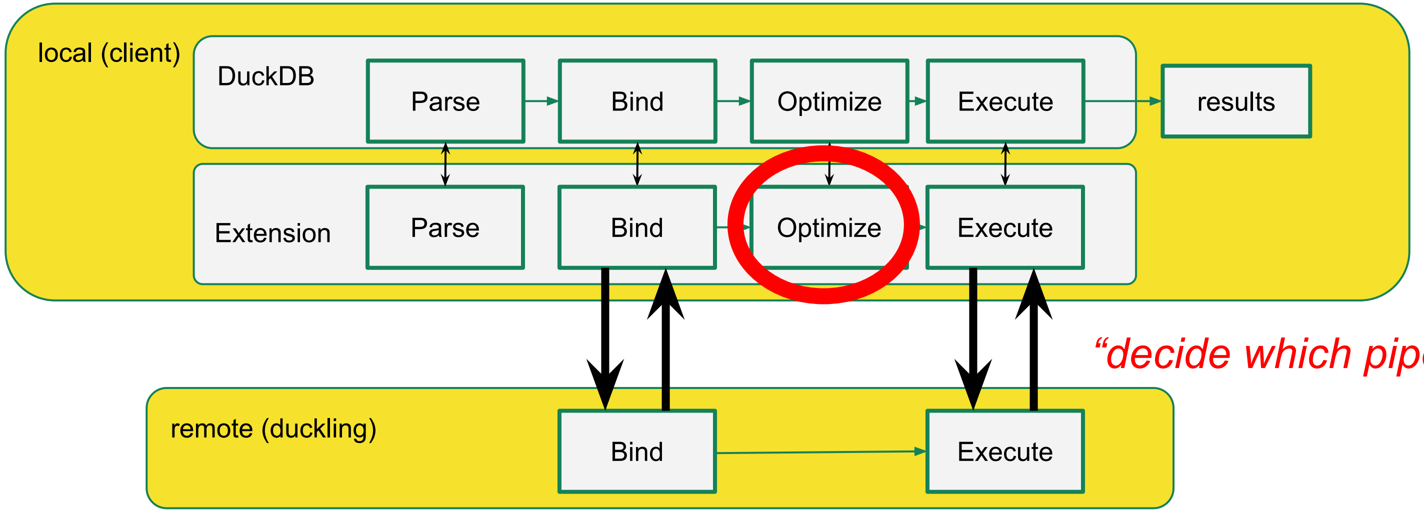


Query Pipelines

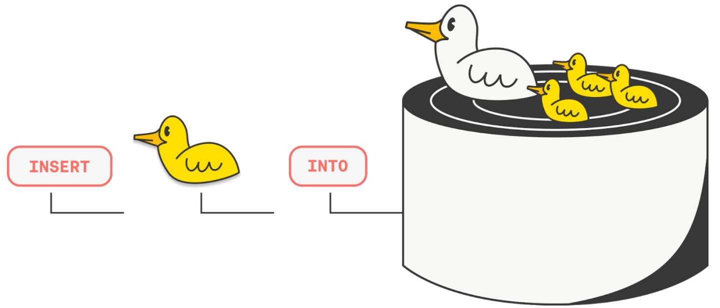
```
SELECT count(*) FROM t1 JOIN t2 ON t1.a = t2.a  JOIN t3 ON t3.b = t2.b
```



Dual Query Processing

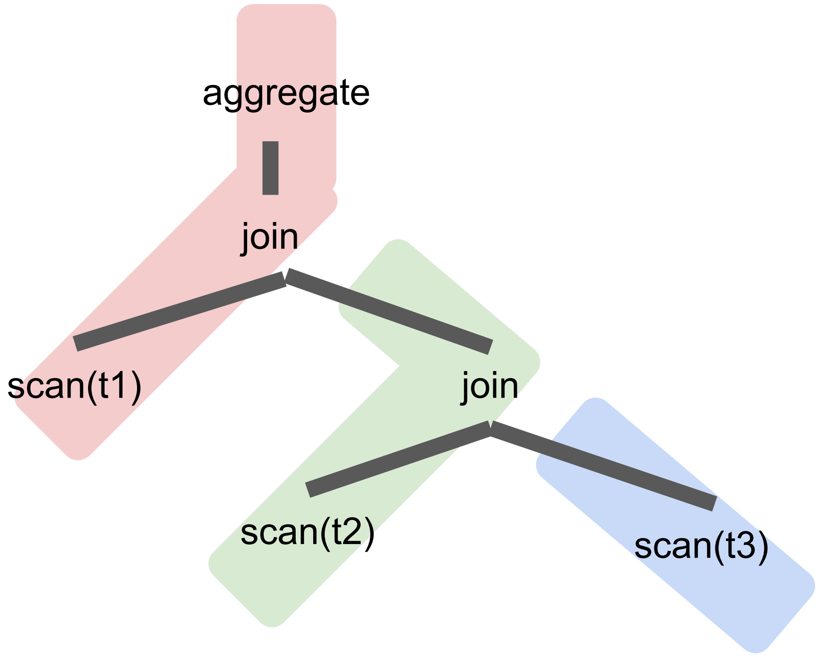


"decide which pipeline to run where"



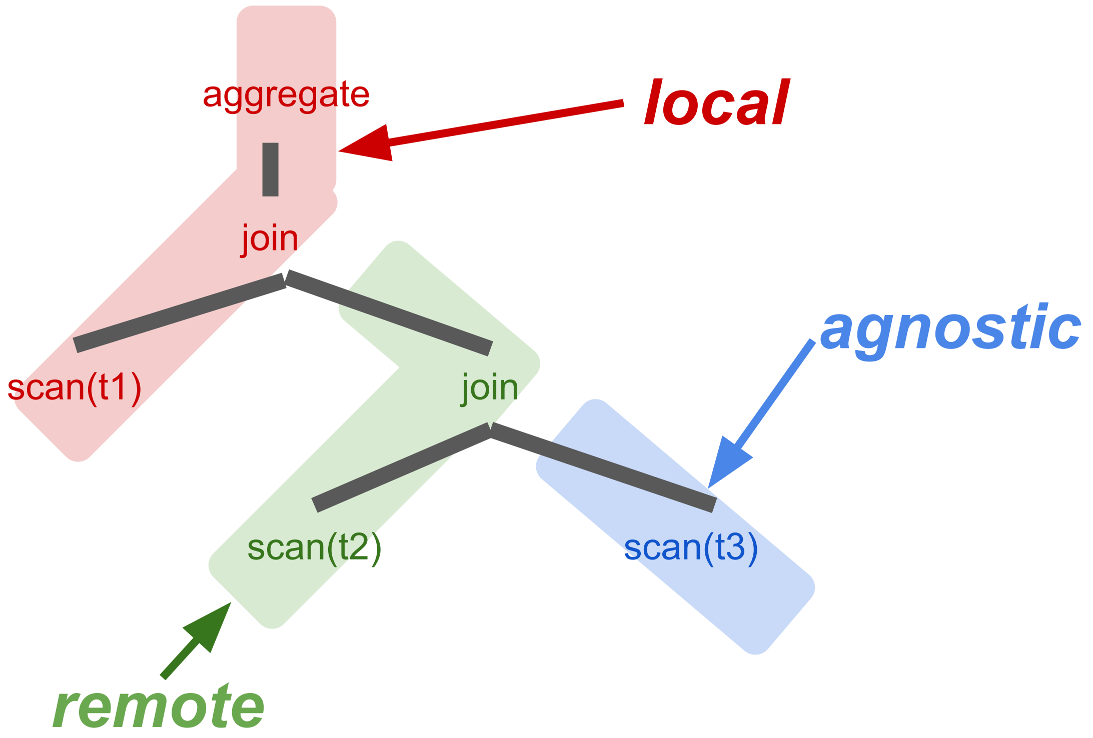
Local-Remote Planning

```
SELECT count(*) FROM t1 JOIN t2 ON t1.a = t2.a  JOIN t3 ON t3.b = t2.b
```



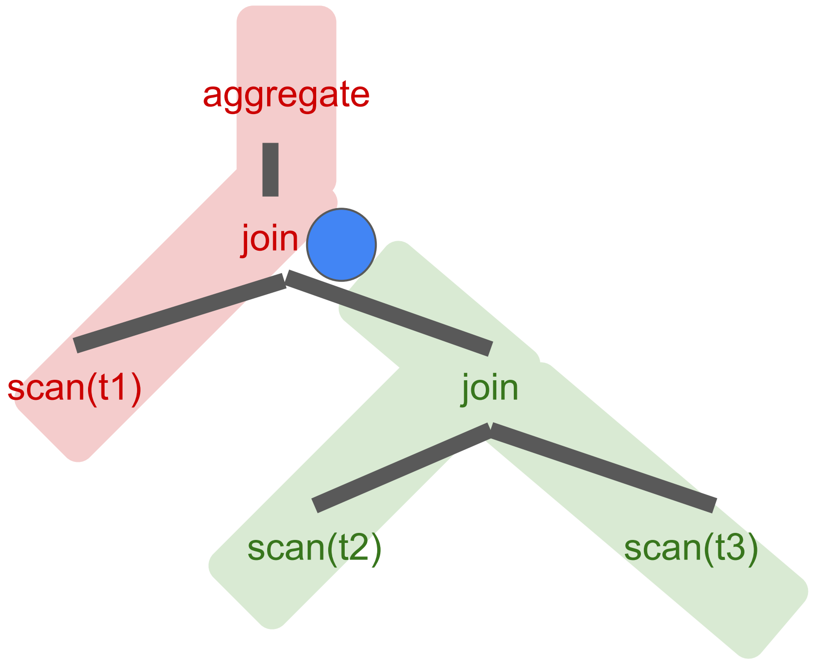
Local-Remote Planning

```
SELECT count(*) FROM t1 JOIN t2 ON t1.a = t2.a  JOIN t3 ON t3.b = t2.b
```



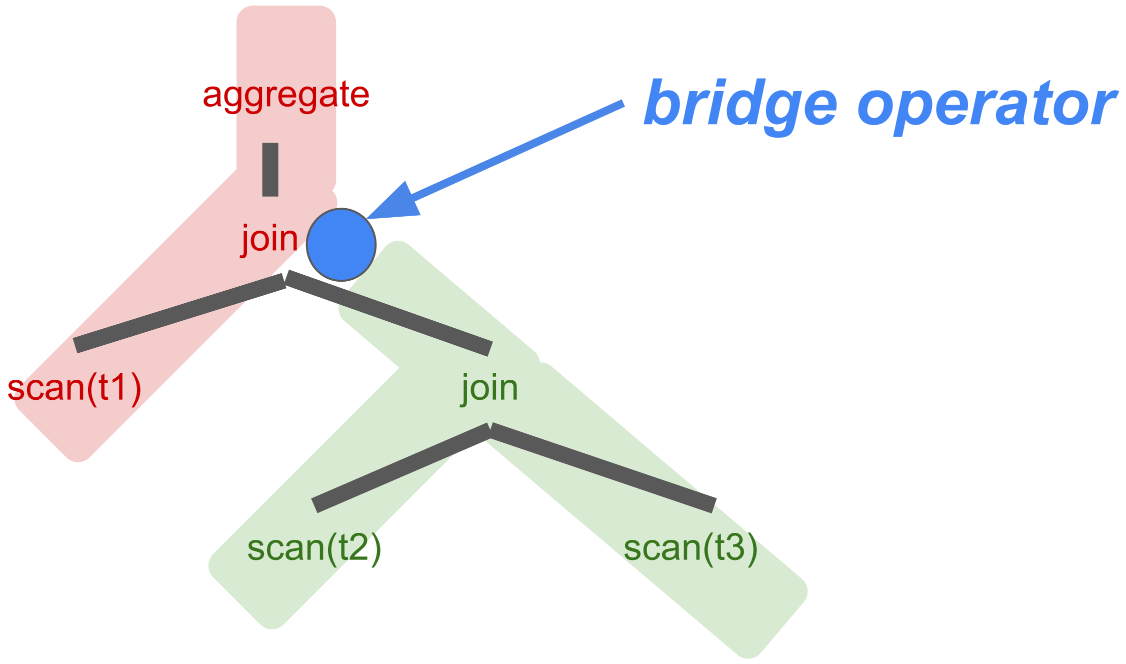
Local-Remote Planning

```
SELECT count(*) FROM t1 JOIN t2 ON t1.a = t2.a  JOIN t3 ON t3.b = t2.b
```



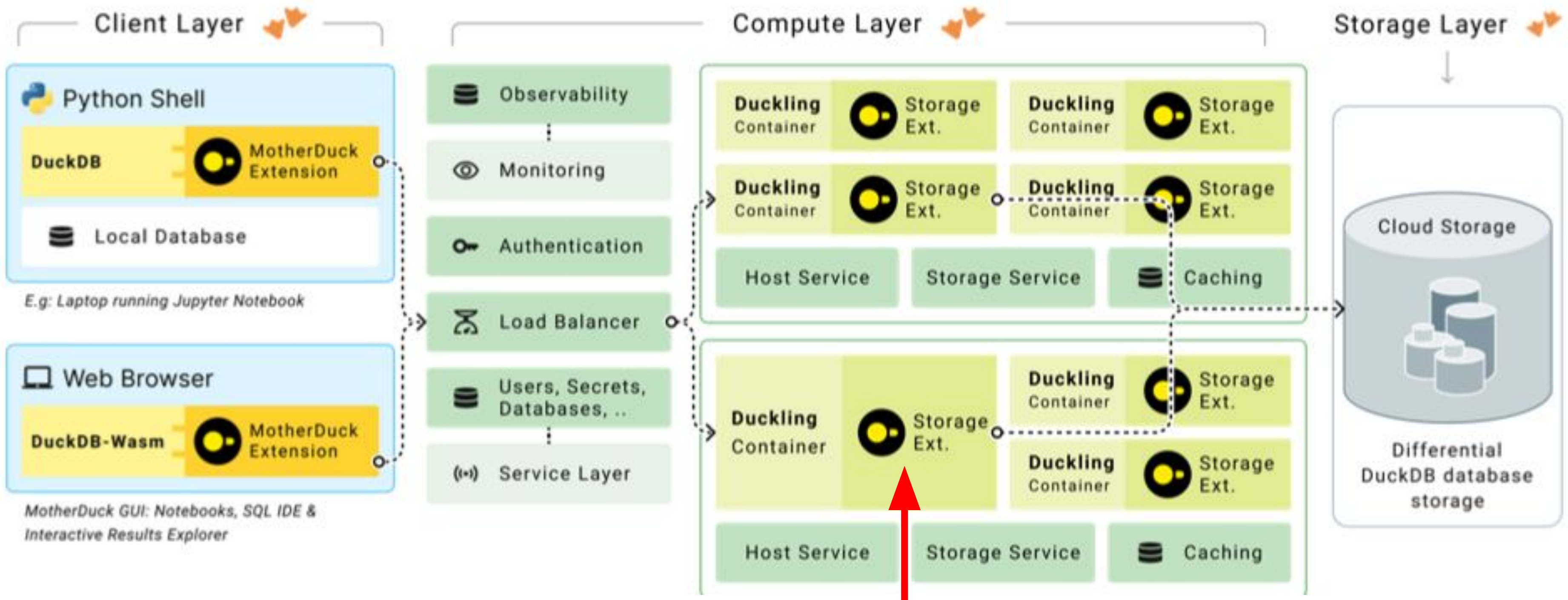
Bridge Operators

```
SELECT count(*) FROM t1 JOIN t2 ON t1.a = t2.a  JOIN t3 ON t3.b = t2.b
```



MotherDuck

MotherDuck Architecture

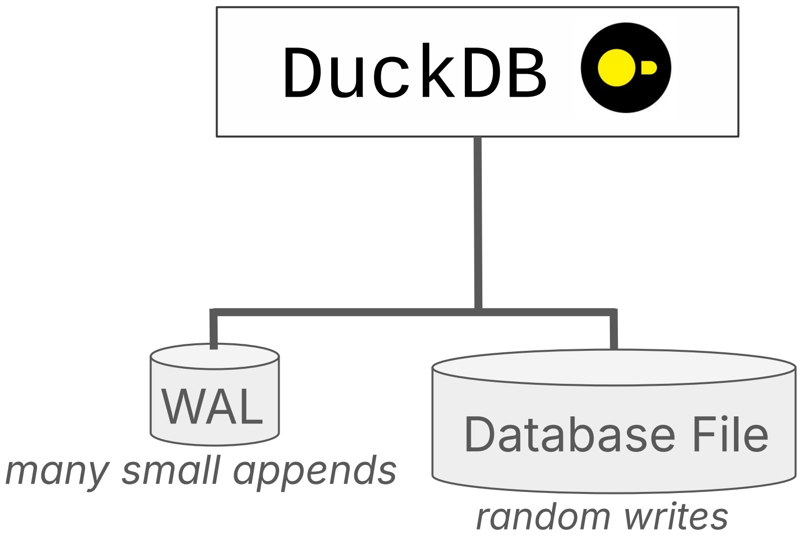


Storage Extension

MotherDuck

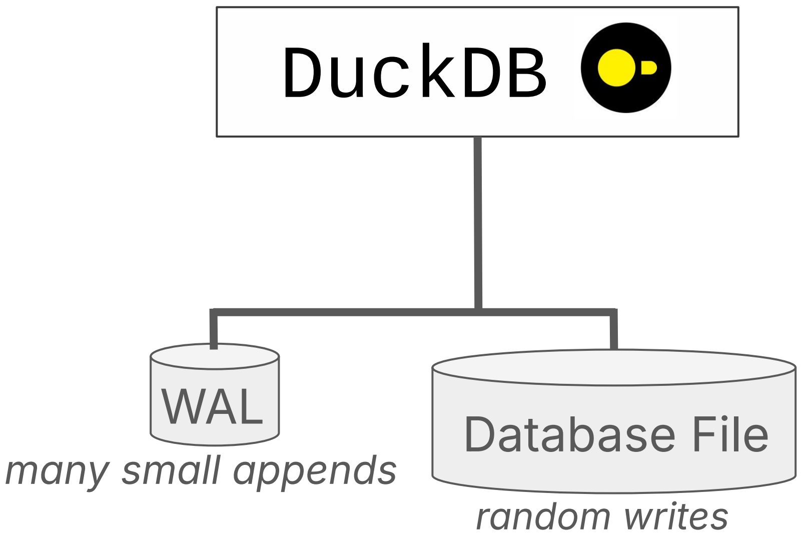
DuckDB Storage

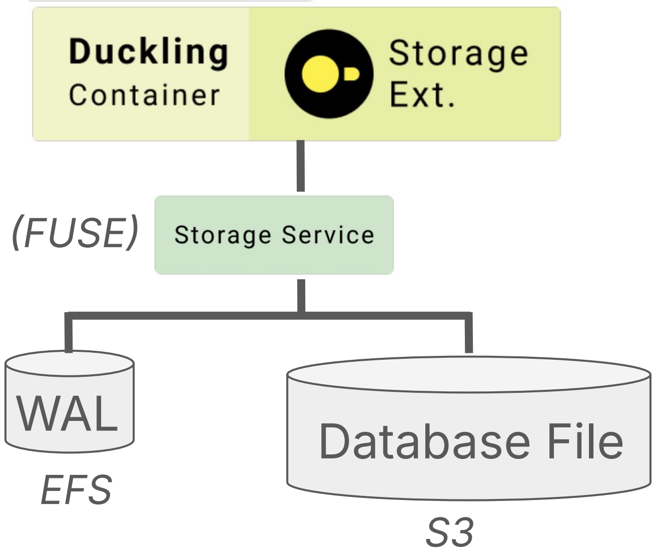
Commit() => Write Ahead Log => Checkpoint() => Database File



MotherDuck Storage

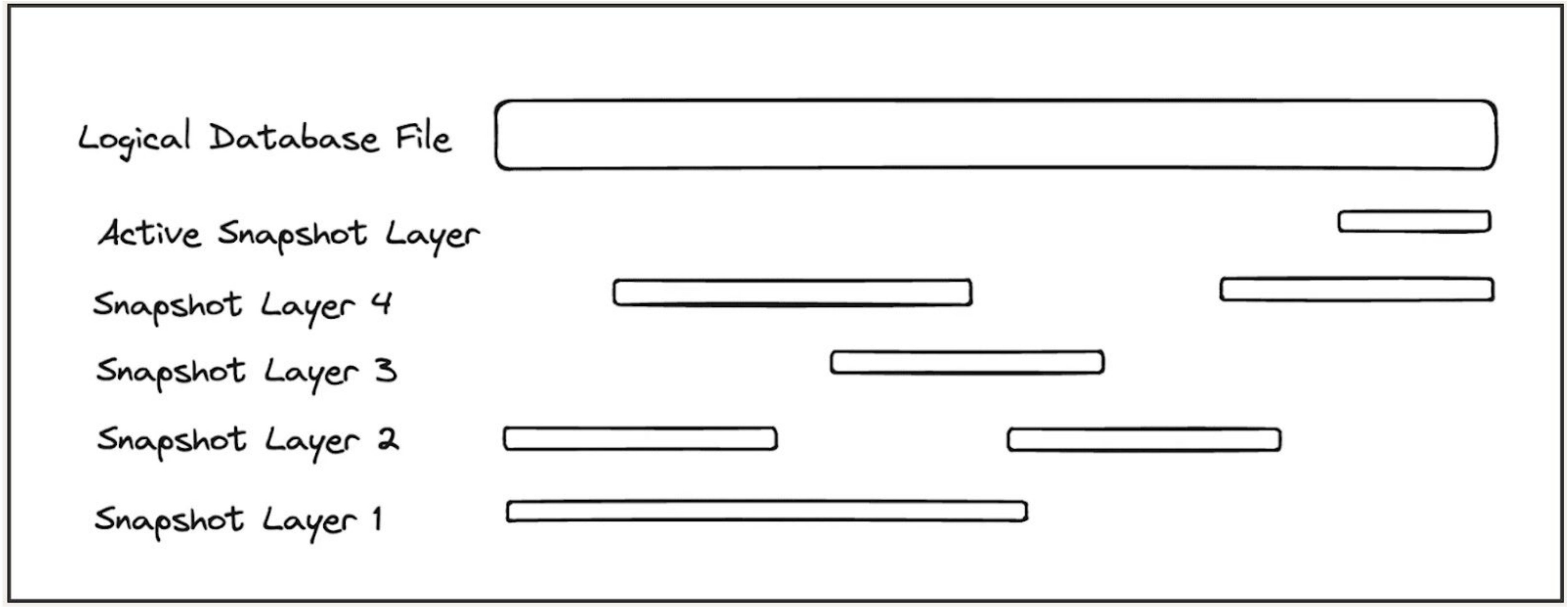
Mapping the same database format on cloud resources..



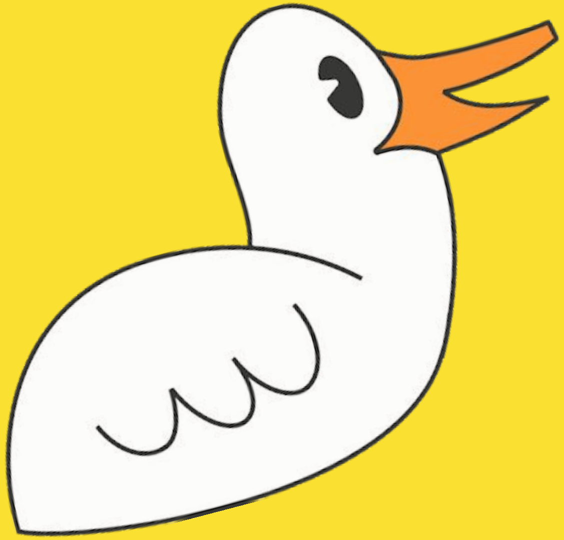


MotherDuck

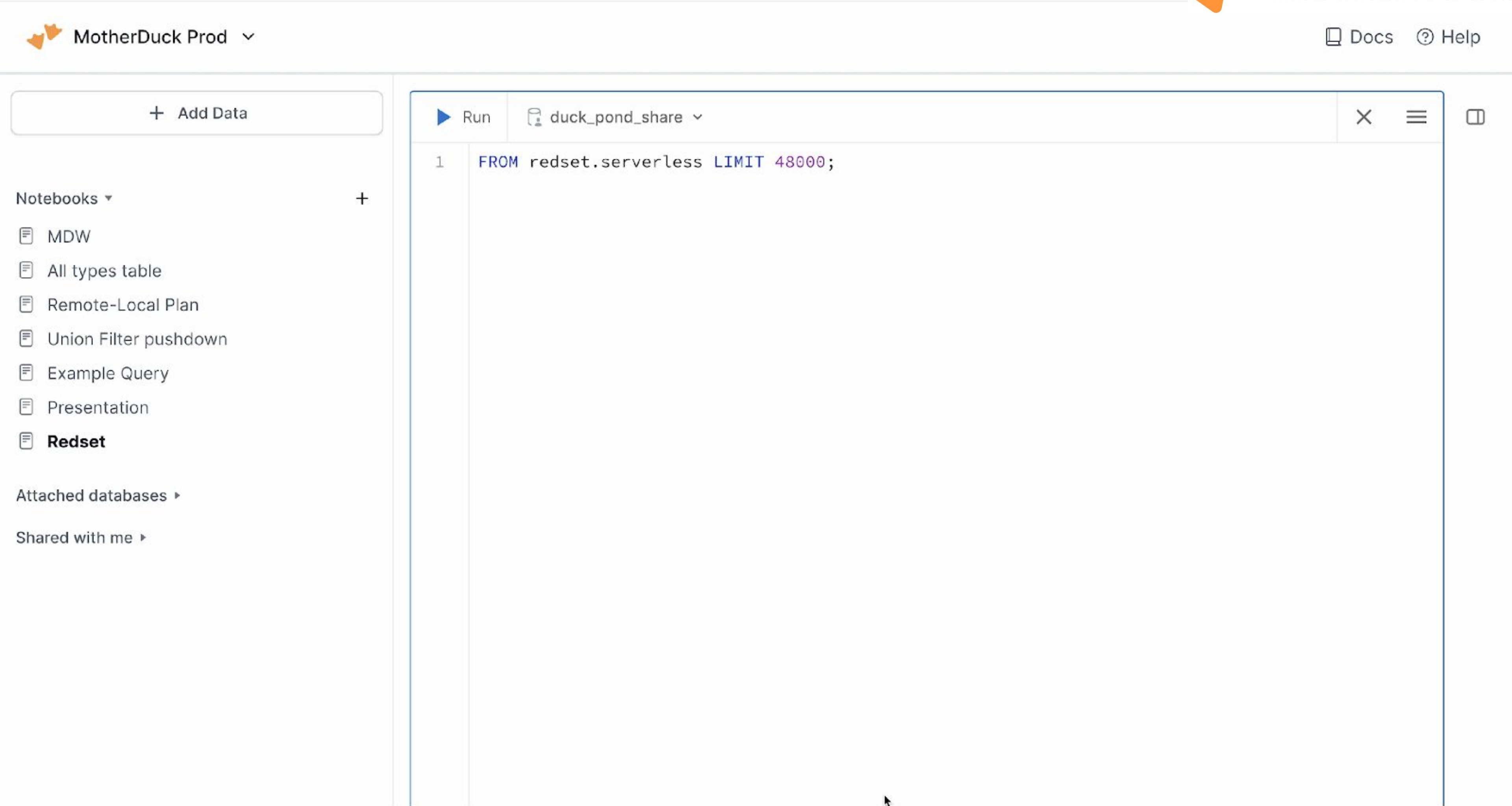
Differential Storage



Declarative Caching



Niclas Haderer
(MotherDuck)



```
FROM redset.serverless LIMIT 48000;
```

What Queries Happen Behind the Scene?

🟨 Local
🟦 Hybrid

Create Cache

```
CREATE TABLE cache AS FROM ... LIMIT 50'000;
```

Pivot Table Widget

```
SELECT * FROM cache LIMIT 1024;
```

```
SELECT count(*) FROM cache;
```

Column Explorer Widget

```
SELECT complex_stat_1() FROM cache;
```

```
SELECT complex_stat_2() FROM cache;
```

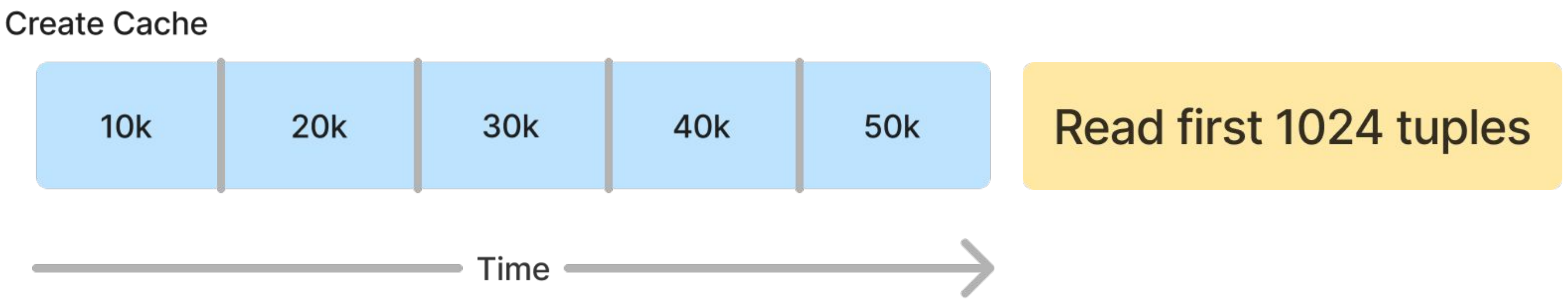
```
SELECT complex_stat_3() FROM cache;
```

```
...
```

MotherDuck

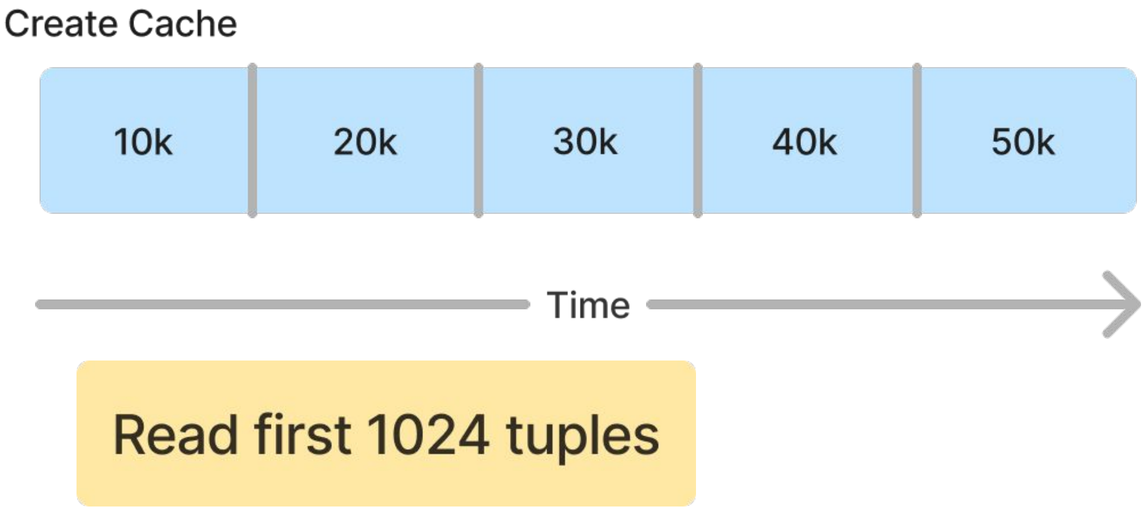
Opportunity #1: allow earlier cache reads

```
SELECT * FROM cache LIMIT 1024;
```

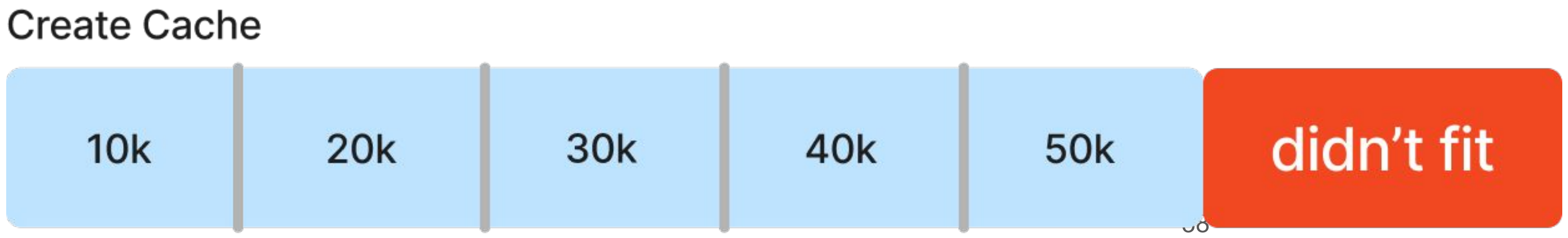


Opportunity #1: allow earlier cache reads

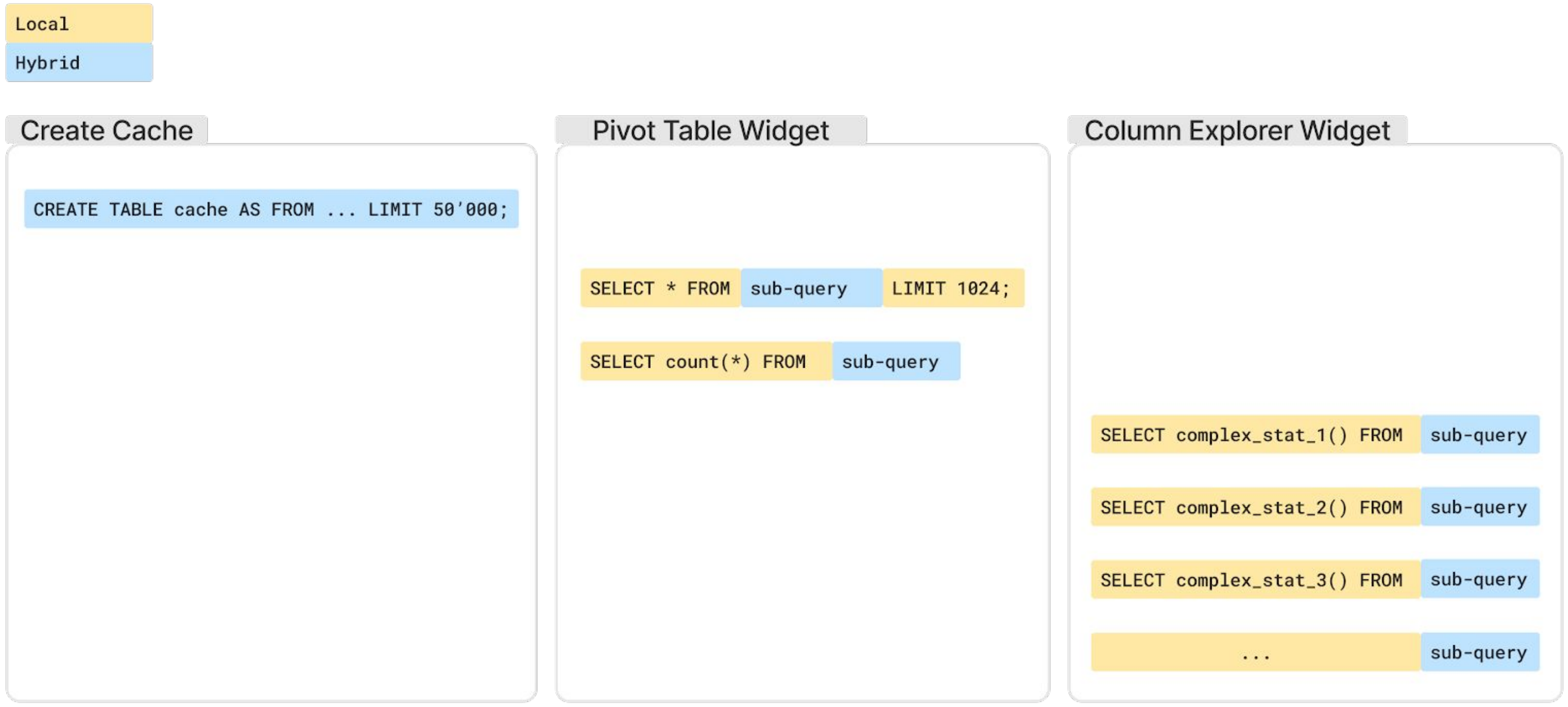
```
SELECT * FROM cache LIMIT 1024;
```



What if the query result is large?



What Queries Happen Behind the Scene?



🟨 Local

🟦 Hybrid

Create Cache

```
CREATE TABLE cache AS FROM ... LIMIT 50'000;
```

Pivot Table Widget

```
SELECT * FROM sub-query LIMIT 1024;
SELECT count(*) FROM sub-query
```

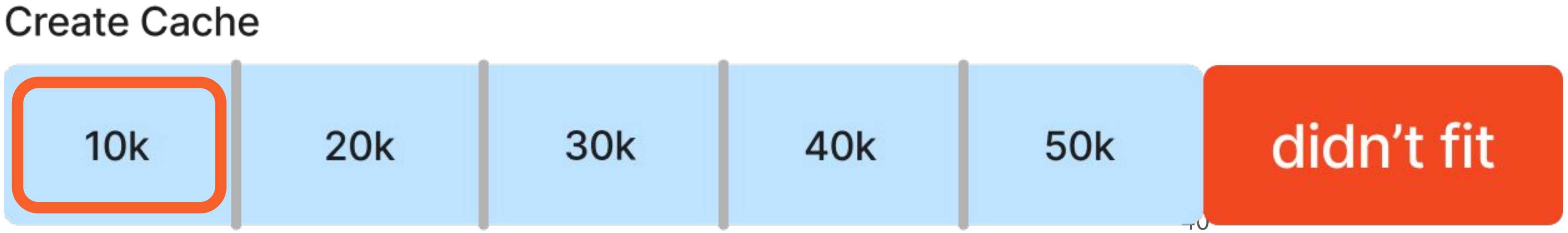
Column Explorer Widget

```
SELECT complex_stat_1() FROM sub-query
SELECT complex_stat_2() FROM sub-query
SELECT complex_stat_3() FROM sub-query
...                          sub-query
```

MotherDuck

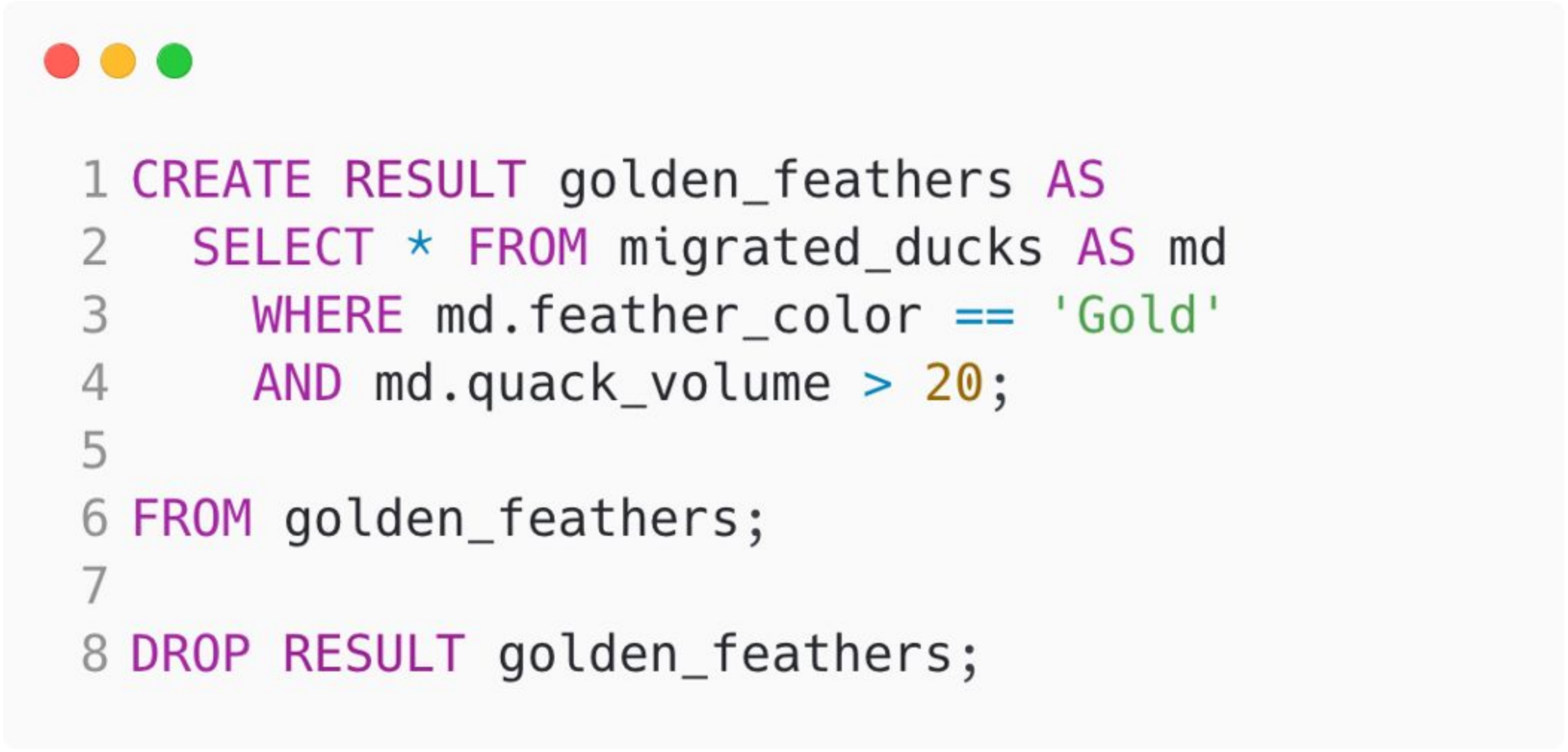
Opportunity #2: allow access to partial caches

```
SELECT * FROM cache LIMIT 1024;
```



MotherDuck

Introducing.. Cached Results

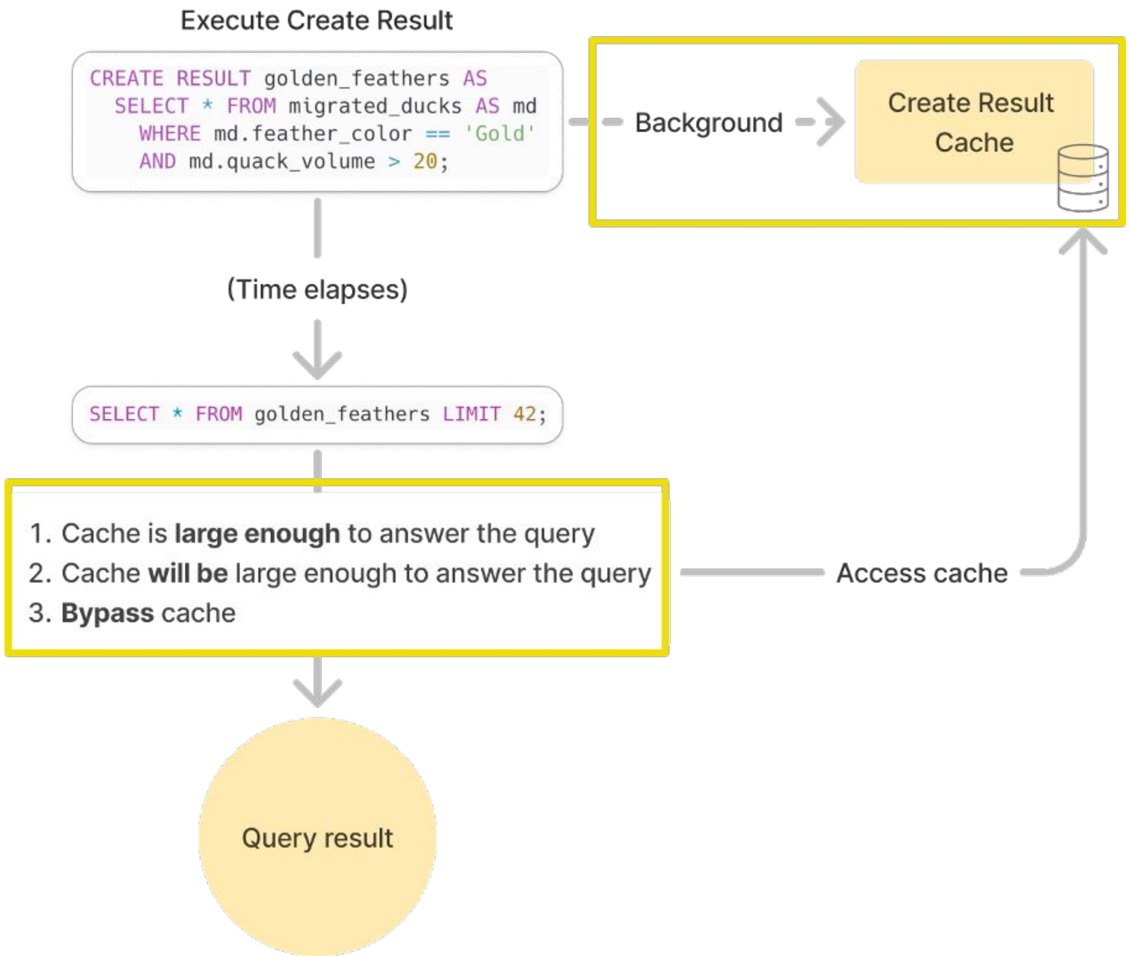


```sql
CREATE RESULT golden_feathers AS
  SELECT * FROM migrated_ducks AS md
    WHERE md.feather_color == 'Gold'
    AND md.quack_volume > 20;

FROM golden_feathers;

DROP RESULT golden_feathers;
```

Result Mechanics

Decision making by the query optimizer!

Is the query covered by the (still growing) cached **RESULT**?



```
CREATE RESULT golden_feathers AS
  SELECT * FROM migrated_ducks AS md
    WHERE md.feather_color == 'Gold'
    AND md.quack_volume > 20;
```

```
SELECT * FROM golden_feathers LIMIT 42;
```

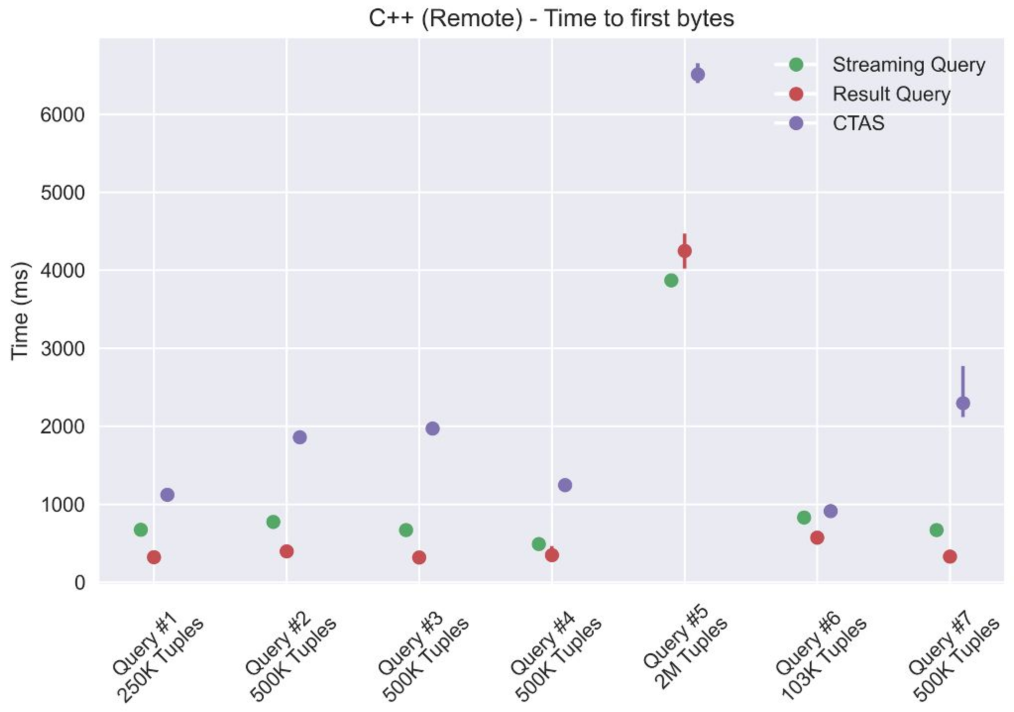
1. Cache is **large enough** to answer the query
2. Cache **will be** large enough to answer the query
3. **Bypass** cache

Some Performance Results

Streaming API: no caching

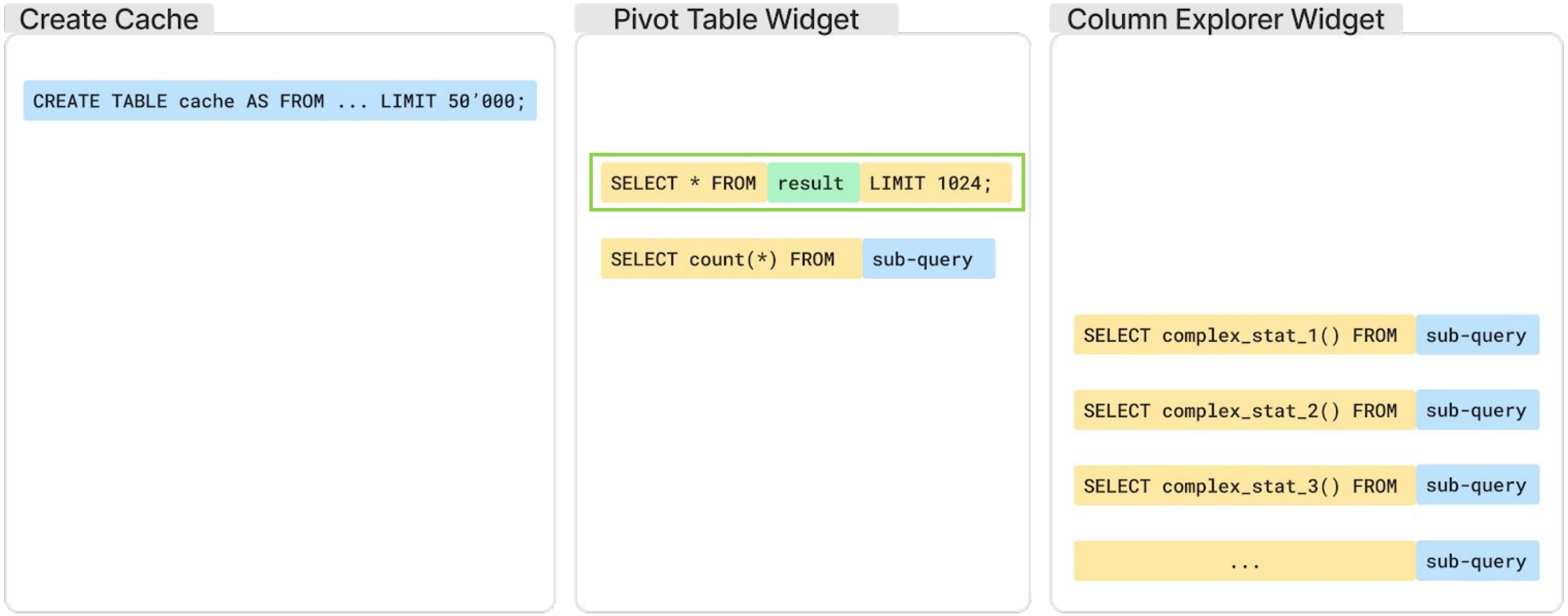
CTAS: current approach

RESULT: new approach



Future Work

🟨	Local
🟦	Hybrid

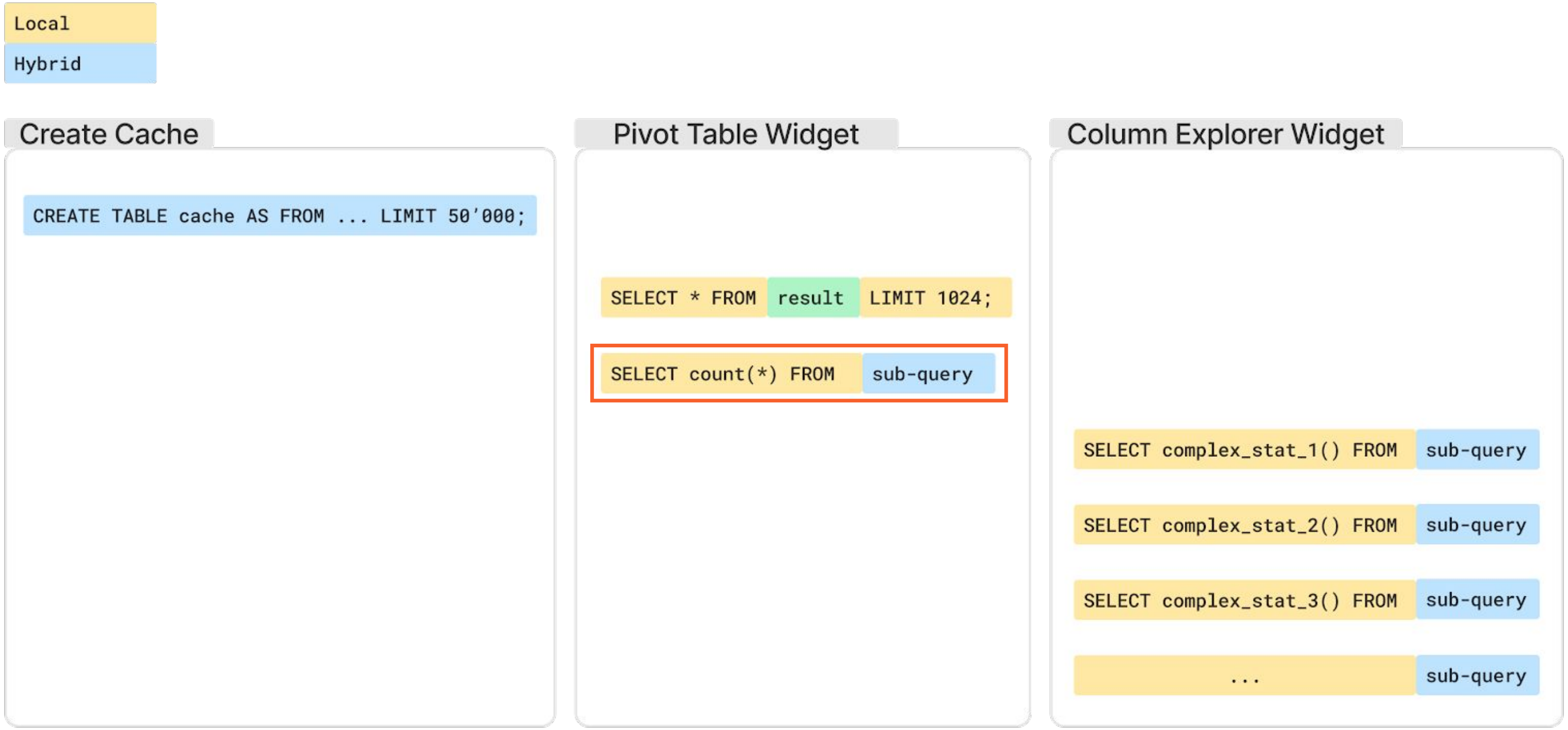


```
CREATE TABLE cache AS FROM ... LIMIT 50'000;
```

```
SELECT * FROM result LIMIT 1024;
SELECT count(*) FROM sub-query
```

```
SELECT complex_stat_1() FROM sub-query
SELECT complex_stat_2() FROM sub-query
SELECT complex_stat_3() FROM sub-query
...                          sub-query
```

Future Work



🟨 Local
🟦 Hybrid

Create Cache

```
CREATE TABLE cache AS FROM ... LIMIT 50'000;
```

Pivot Table Widget

```
SELECT * FROM result LIMIT 1024;
SELECT count(*) FROM sub-query
```

Column Explorer Widget

```
SELECT complex_stat_1() FROM sub-query
SELECT complex_stat_2() FROM sub-query
SELECT complex_stat_3() FROM sub-query
...                          sub-query
```

MotherDuck

Future Work

🟨 Local
🟦 Hybrid

Create Cache

```
CREATE TABLE cache AS FROM ... LIMIT 50'000;
```

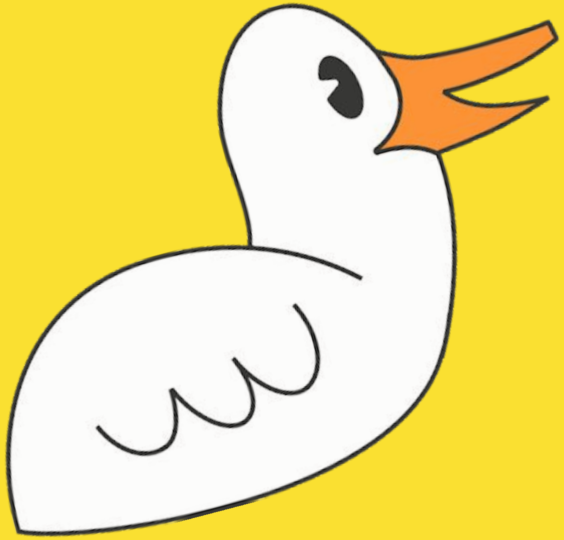
Pivot Table Widget

```
SELECT * FROM result LIMIT 1024;
SELECT count(*) FROM sub-query
```

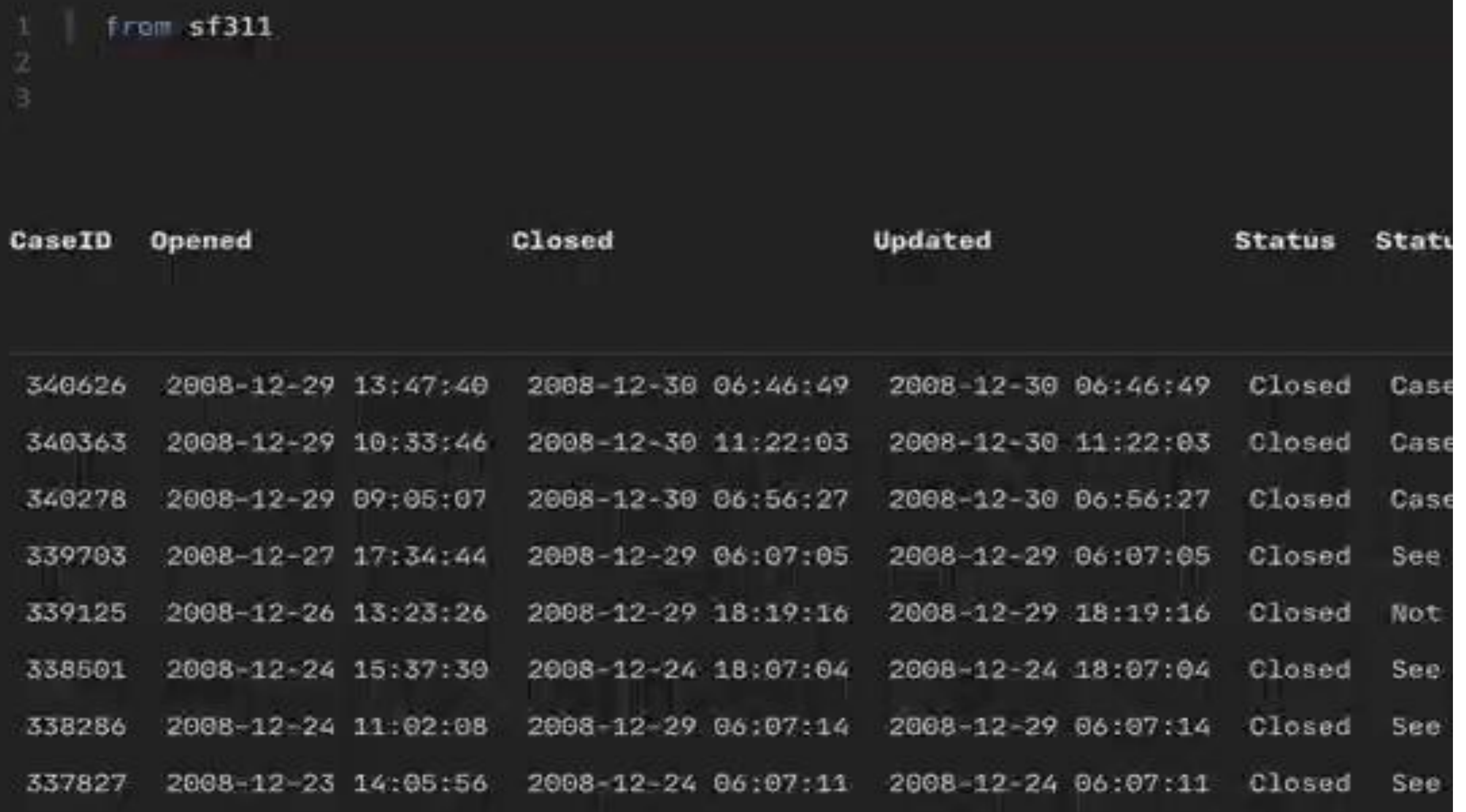
Column Explorer Widget

```
SELECT complex_stat_1() FROM sub-query
SELECT complex_stat_2() FROM sub-query
SELECT complex_stat_3() FROM sub-query
...                          sub-query
```

Instant Preview Mode



Hamilton Ulmer
(MotherDuck)



```
from sf311
```

```
CaseID  Opened               Closed               Updated              Status  Statu

340626  2008-12-29 13:47:40  2008-12-30 06:46:49  2008-12-30 06:46:49  Closed  Case
340363  2008-12-29 10:33:46  2008-12-30 11:22:03  2008-12-30 11:22:03  Closed  Case
340278  2008-12-29 09:05:07  2008-12-30 06:56:27  2008-12-30 06:56:27  Closed  Case
339703  2008-12-27 17:34:44  2008-12-29 06:07:05  2008-12-29 06:07:05  Closed  See
339125  2008-12-26 13:23:26  2008-12-29 18:19:16  2008-12-29 18:19:16  Closed  Not
338501  2008-12-24 15:37:30  2008-12-24 18:07:04  2008-12-24 18:07:04  Closed  See
338286  2008-12-24 11:02:08  2008-12-29 06:07:14  2008-12-29 06:07:14  Closed  See
337827  2008-12-23 14:05:56  2008-12-24 06:07:11  2008-12-24 06:07:11  Closed  See
```

Conclusion

Hybrid Execution in MotherDuck

- Move some processing to the client
  - Lower cloud bills
  - Lower latencies (“60 fps”)
  - Exploit client-side data
- Declarative Result Caching
  - Fast access to the first tuple
  - Can be done both on client- and server-side (small resp. big caches)
- Instant Preview Mode
  - Making query formulation easier
  - Direct Feedback + AI
  - Backed by “join synopses”

Thank You!