DuckDB

An in-process analytical database management system

Gábor Szárnyas
Developer Relations Advocate

DuckDB Labs

Design goals

- Easy-to-deploy SQL database
- Portable anywhere
- Performance of a data warehouse

In-process

Client–server setup

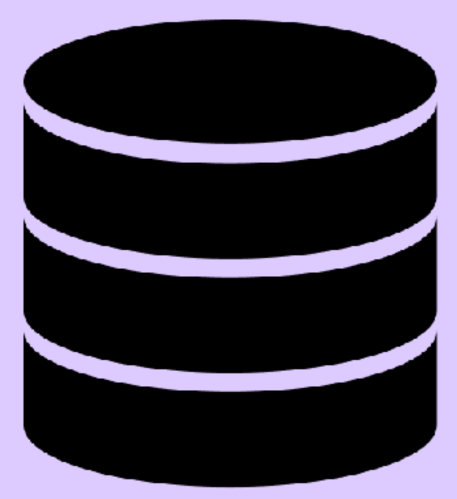
Client application

```
import psycopg2
con = psycopg2.connect(
  host="1.2.3.4",
  port=8000,
  user="my_user",
  password="my_password",
  db_name="my_database")
cur = con.cursor()
cur.execute("SELECT ...")
```

Connection setup and authentication

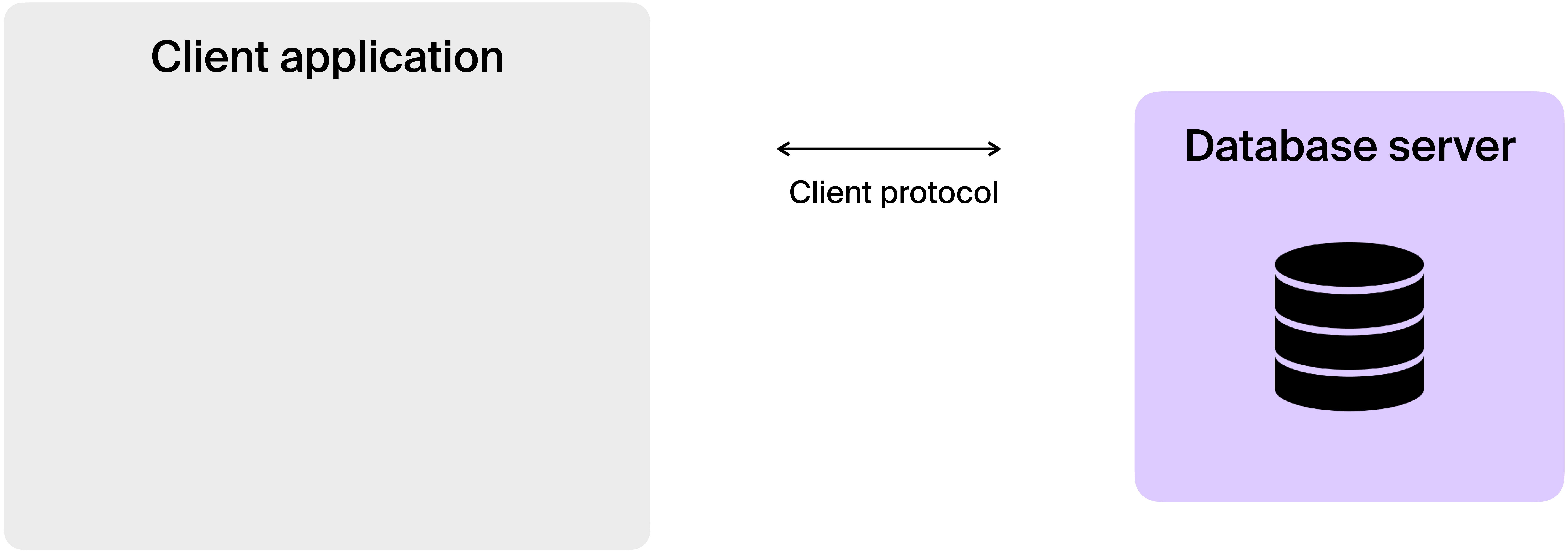
Transferring data back and forth: bottleneck

Database server

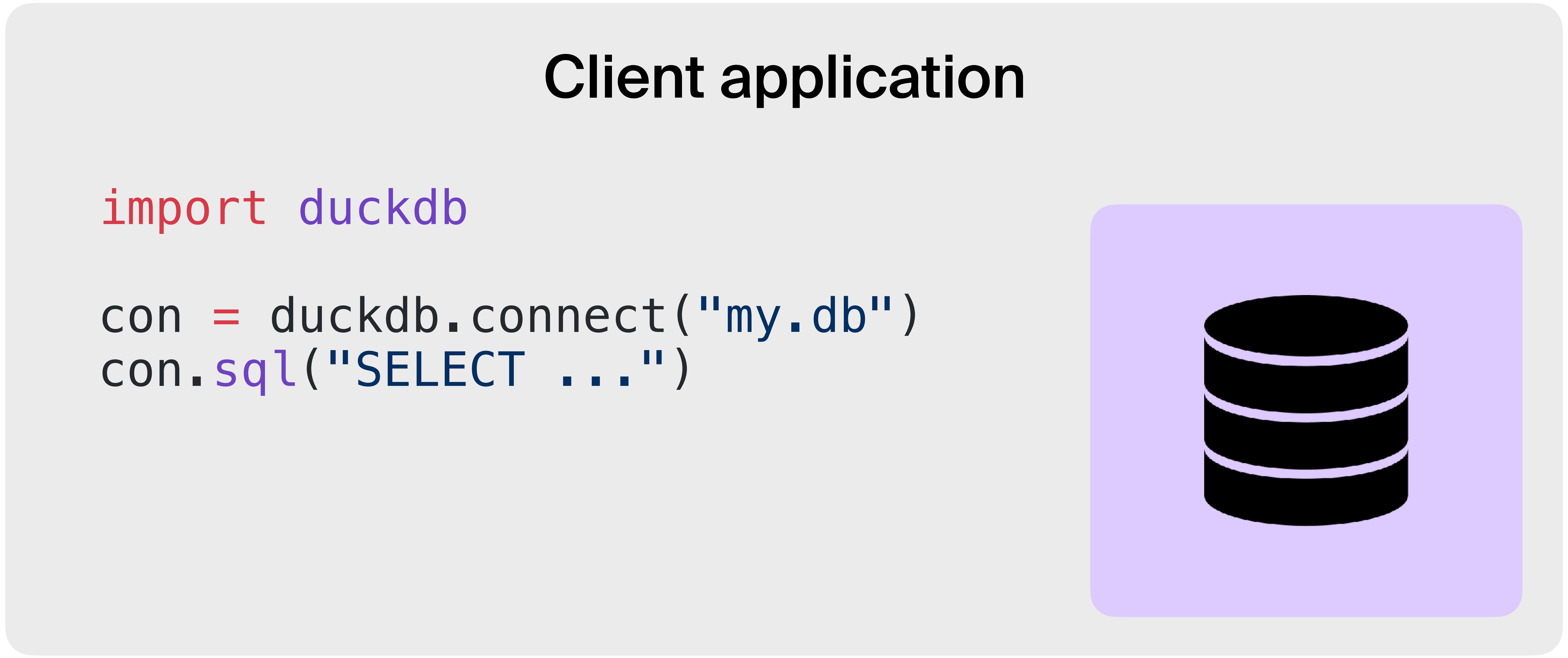


Configuration and operation

Client–server setup



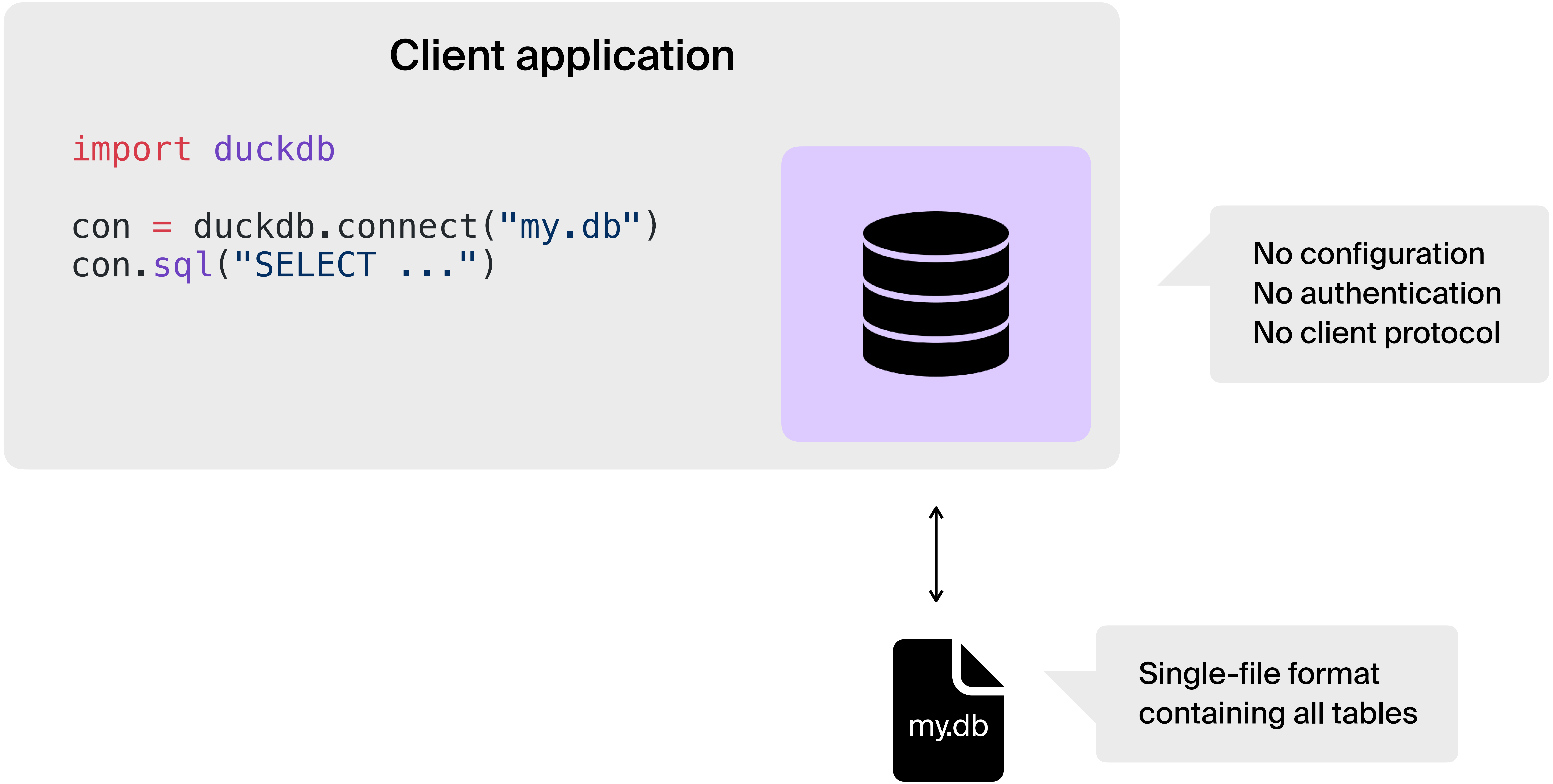
In-process setup



```python
import duckdb

con = duckdb.connect("my.db")
con.sql("SELECT ...")
```

In-process setup



```python
import duckdb

con = duckdb.connect("my.db")
con.sql("SELECT ...")
```

No configuration
No authentication
No client protocol

Single-file format
containing all tables

Categorization

	Transactional *row-oriented*	Analytical *column-oriented*
In-process	SQLite	DuckDB
Client–server	PostgreSQL, MySQL	VERTICA, snowflake

Portable

DuckDB runs anywhere

./duckdb

Pkg.add("DuckDB")

pip install duckdb

org.duckdb:duckdb_jdbc

install.packages('duckdb')

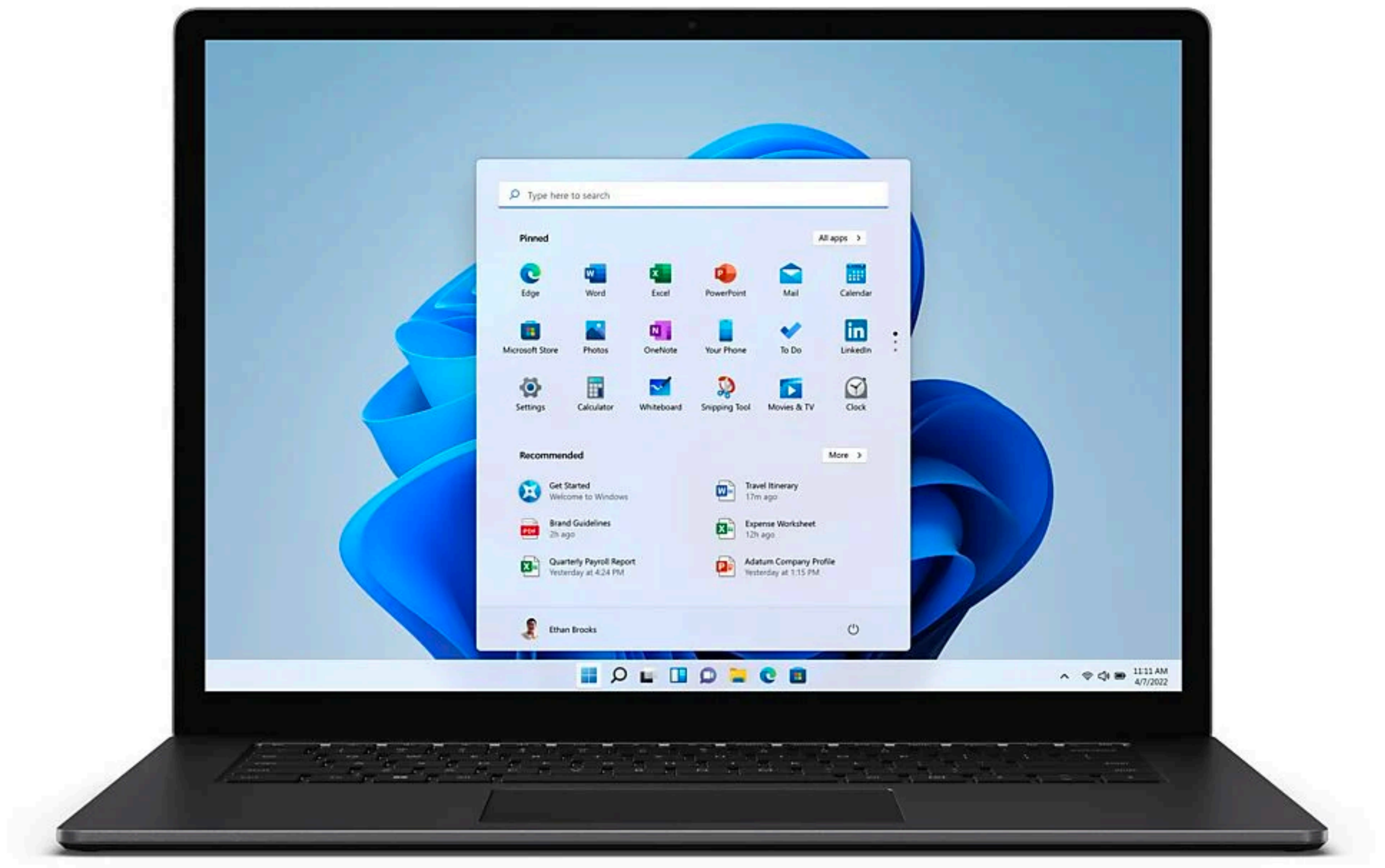
cargo add duckdb

Linux, macOS, Windows

web browsers (WebAssembly)

Performance

Your laptop is much faster than you think



Fast disk, 8+ CPU cores

CSV loader performance

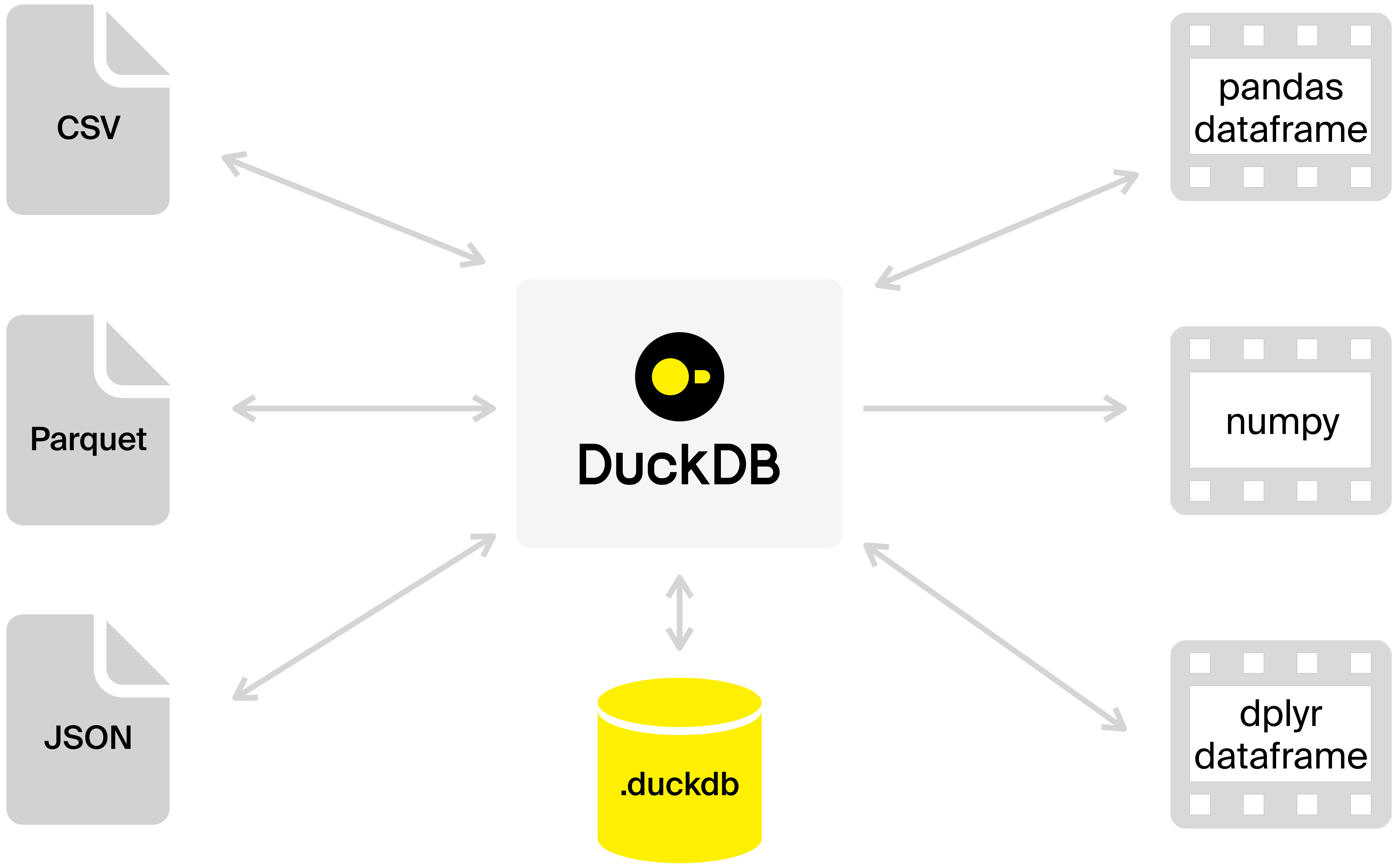
Loading CSV at more than 1 GB/s

CSV in GB	Load time
26 GB	20 s
253 GB	221 s

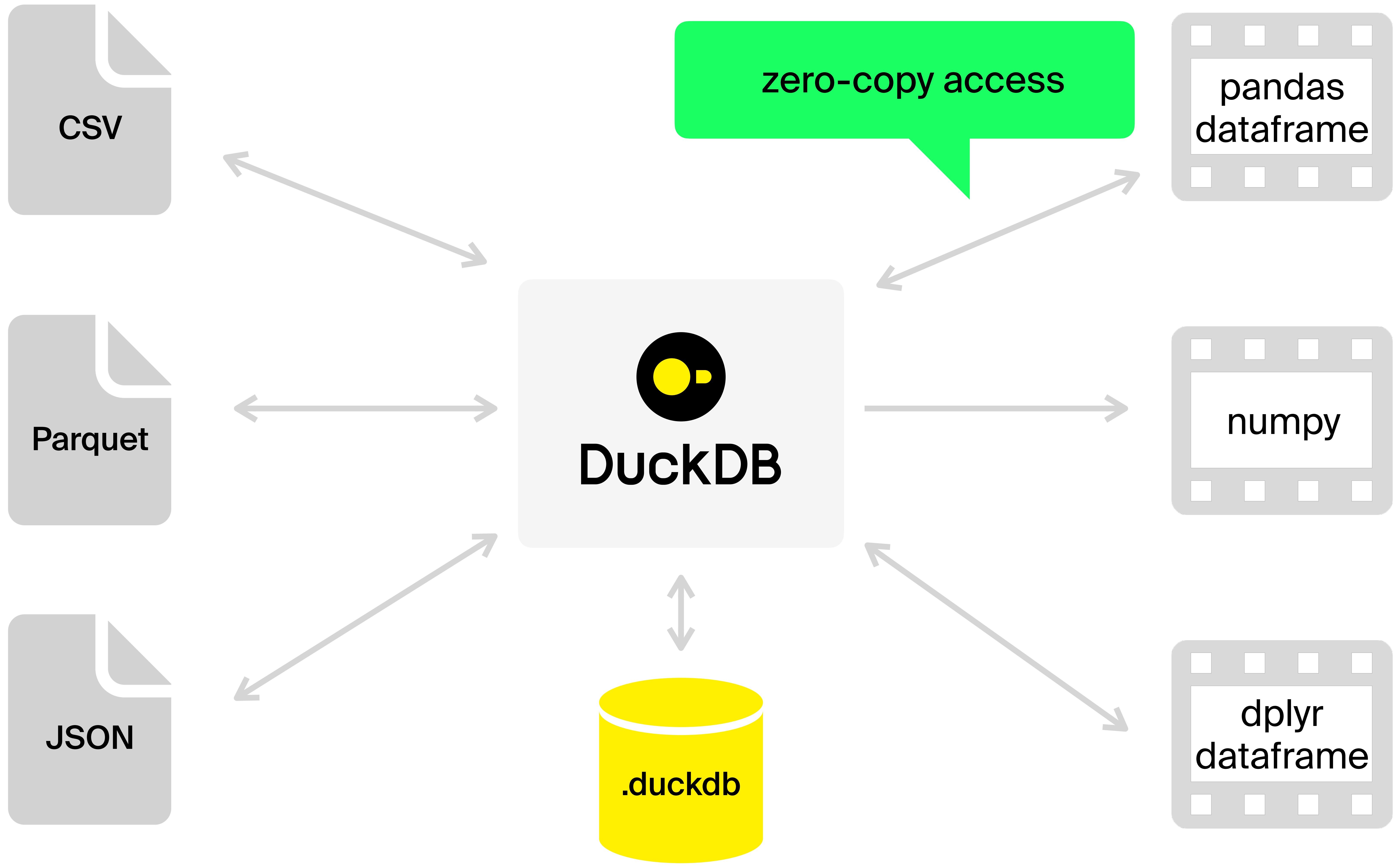
MacBook, M2Pro CPU, 32GB RAM

Feature-rich

Input and output formats



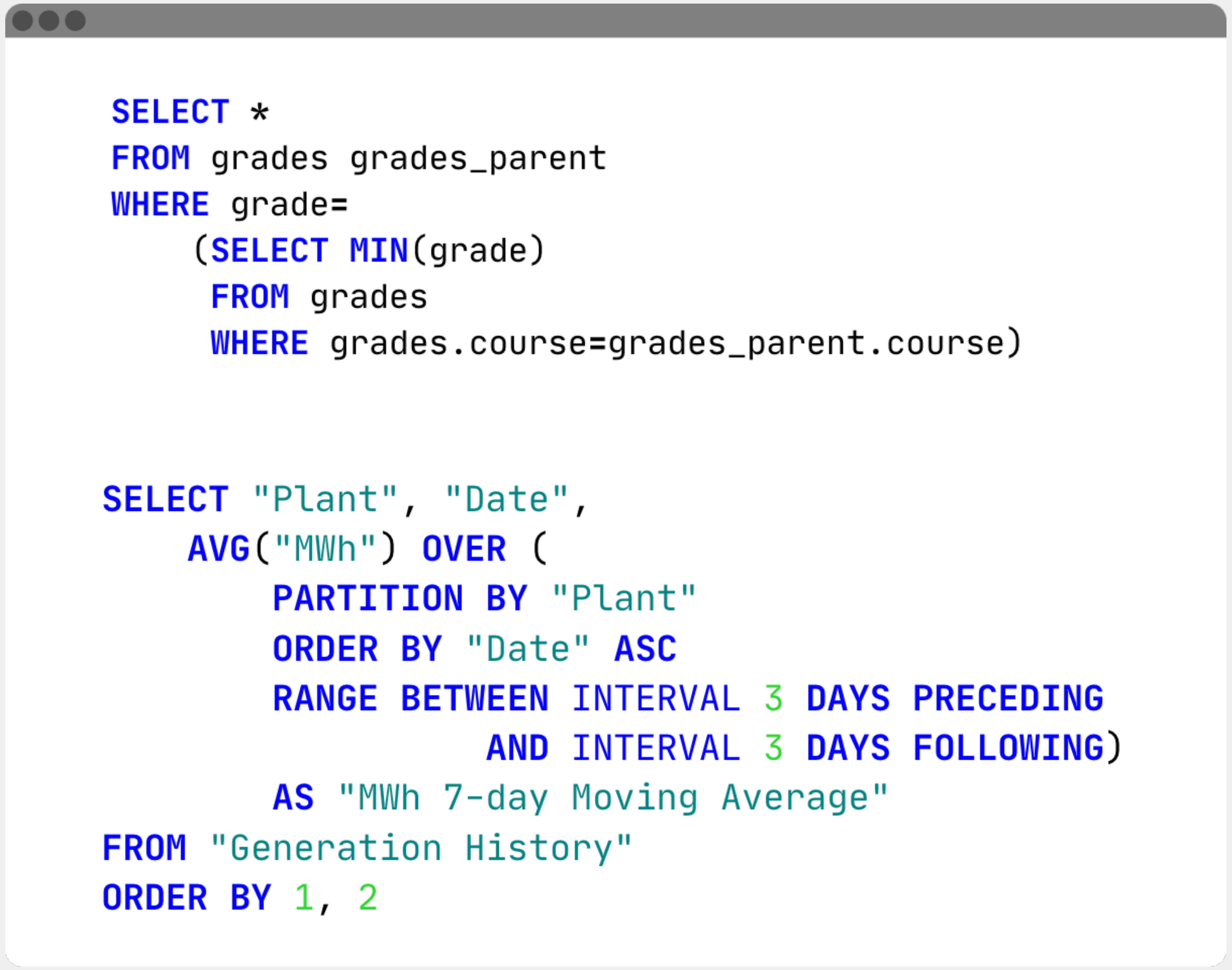
Input and output formats



Query language

PostgreSQL dialect:

- Filtering, joins, aggregates
- Subqueries
- Window functions
- Pivoting and unpivoting tables
- AsOf joins



```sql
SELECT *
FROM grades grades_parent
WHERE grade=
    (SELECT MIN(grade)
     FROM grades
     WHERE grades.course=grades_parent.course)


SELECT "Plant", "Date",
    AVG("MWh") OVER (
        PARTITION BY "Plant"
        ORDER BY "Date" ASC
        RANGE BETWEEN INTERVAL 3 DAYS PRECEDING
                  AND INTERVAL 3 DAYS FOLLOWING)
        AS "MWh 7-day Moving Average"
FROM "Generation History"
ORDER BY 1, 2
```

PIVOT and UNPIVOT

Country varchar	Name varchar	Year int32	Population int32
NL	Amsterdam	2000	1005
NL	Amsterdam	2010	1065
NL	Amsterdam	2020	1158
US	Seattle	2000	564
US	Seattle	2010	608
US	Seattle	2020	738
US	New York City	2000	8015
US	New York City	2010	8175
US	New York City	2020	8772

Country varchar	Name varchar	2000 int32	2010 int32	2020 int32
US	New York City	8015	8175	8772
US	Seattle	564	608	738
NL	Amsterdam	1005	1065	1158

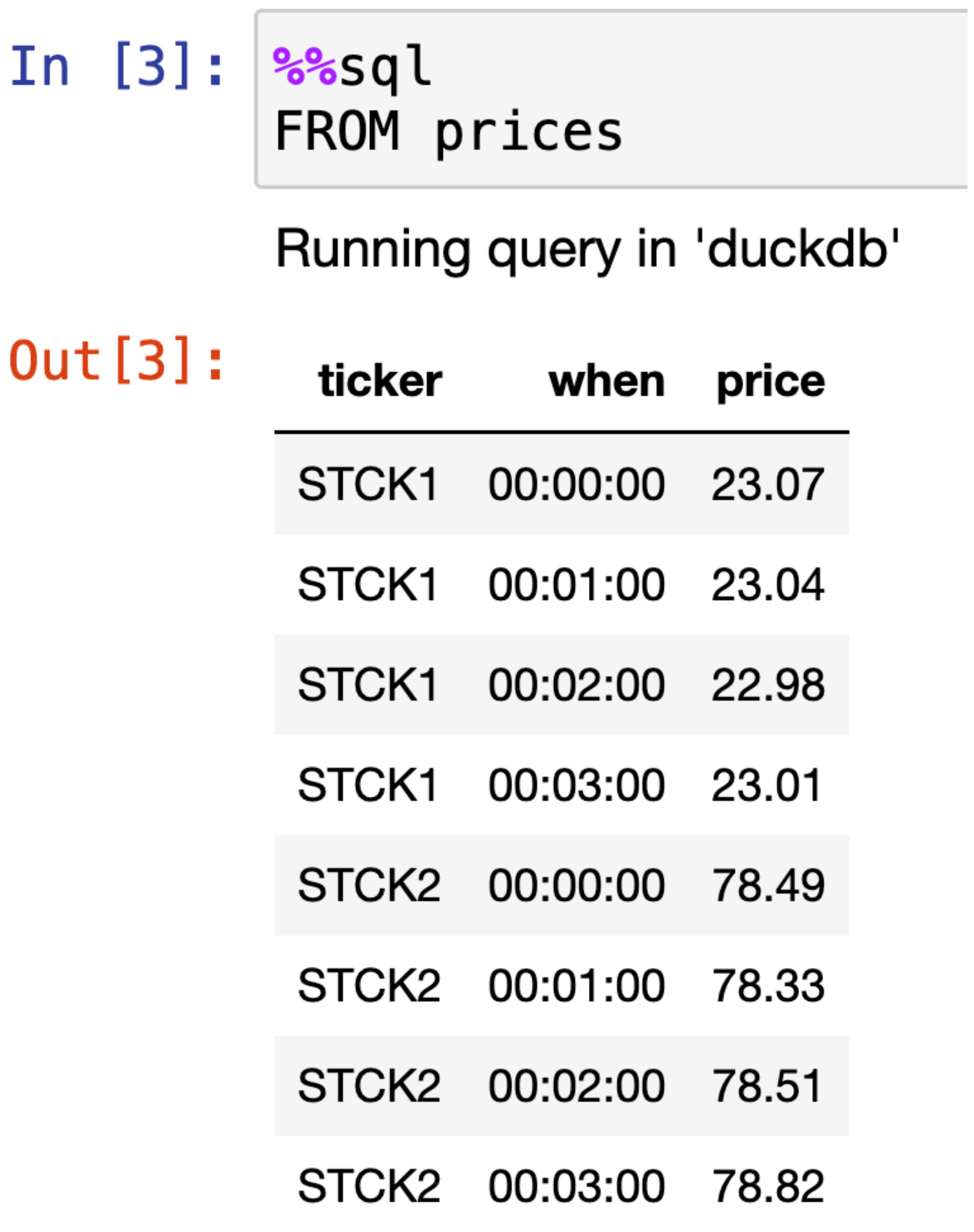
PIVOT

```
PIVOT Cities1
ON Year USING SUM(Population);
```

UNPIVOT

```
UNPIVOT Cities2 ON 2000, 2010, 2020
INTO
  NAME Year
  VALUE Population;
```

AsOf joins: Fuzzy temporal lookups

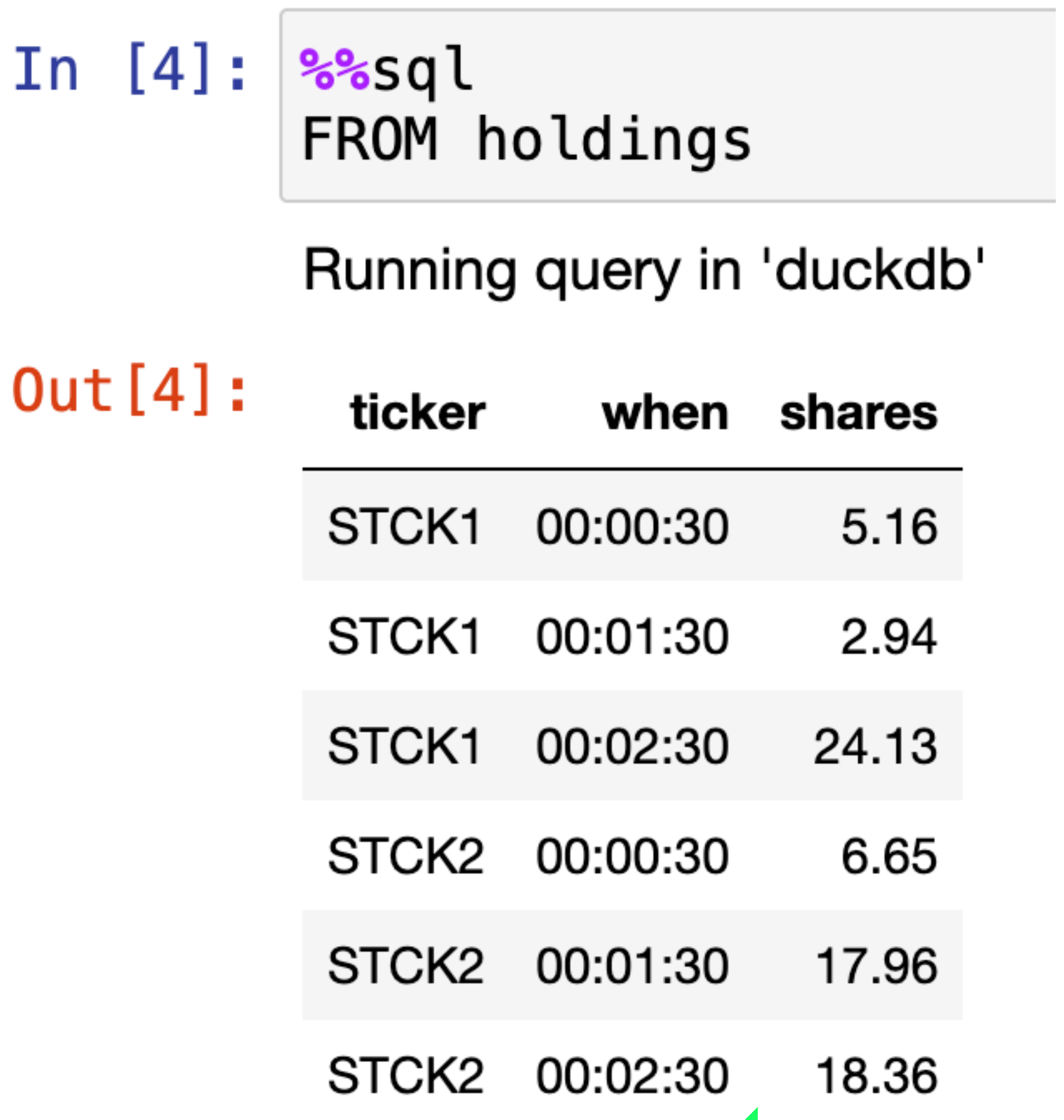


In [3]:
```
%%sql
FROM prices
```

Running query in 'duckdb'

Out[3]:

ticker	when	price
STCK1	00:00:00	23.07
STCK1	00:01:00	23.04
STCK1	00:02:00	22.98
STCK1	00:03:00	23.01
STCK2	00:00:00	78.49
STCK2	00:01:00	78.33
STCK2	00:02:00	78.51
STCK2	00:03:00	78.82



In [4]:
```
%%sql
FROM holdings
```

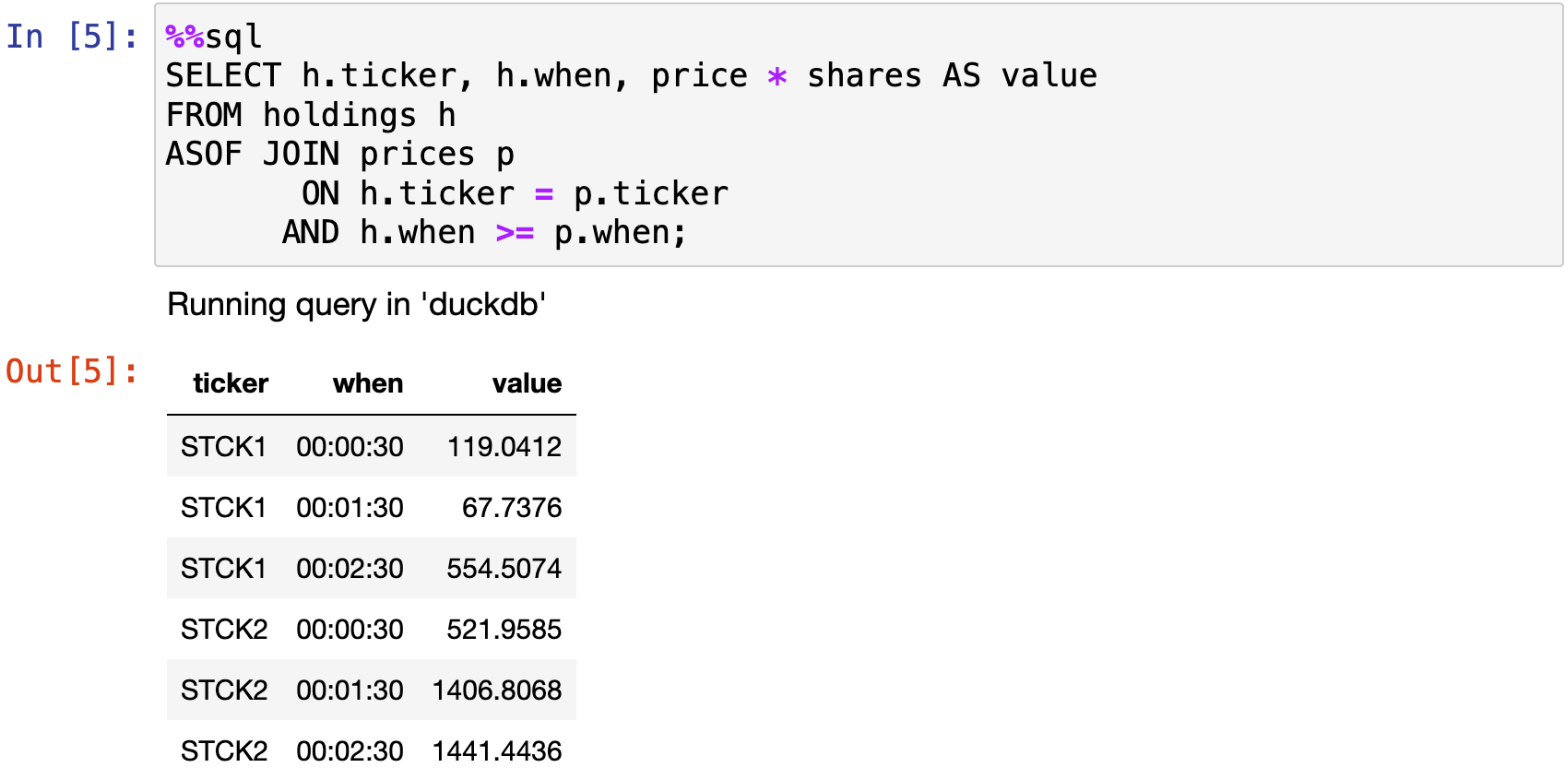
Running query in 'duckdb'

Out[4]:

ticker	when	shares
STCK1	00:00:30	5.16
STCK1	00:01:30	2.94
STCK1	00:02:30	24.13
STCK2	00:00:30	6.65
STCK2	00:01:30	17.96
STCK2	00:02:30	18.36

What is the price **as of** this time?

AsOf joins: Fuzzy temporal lookups



In [5]:

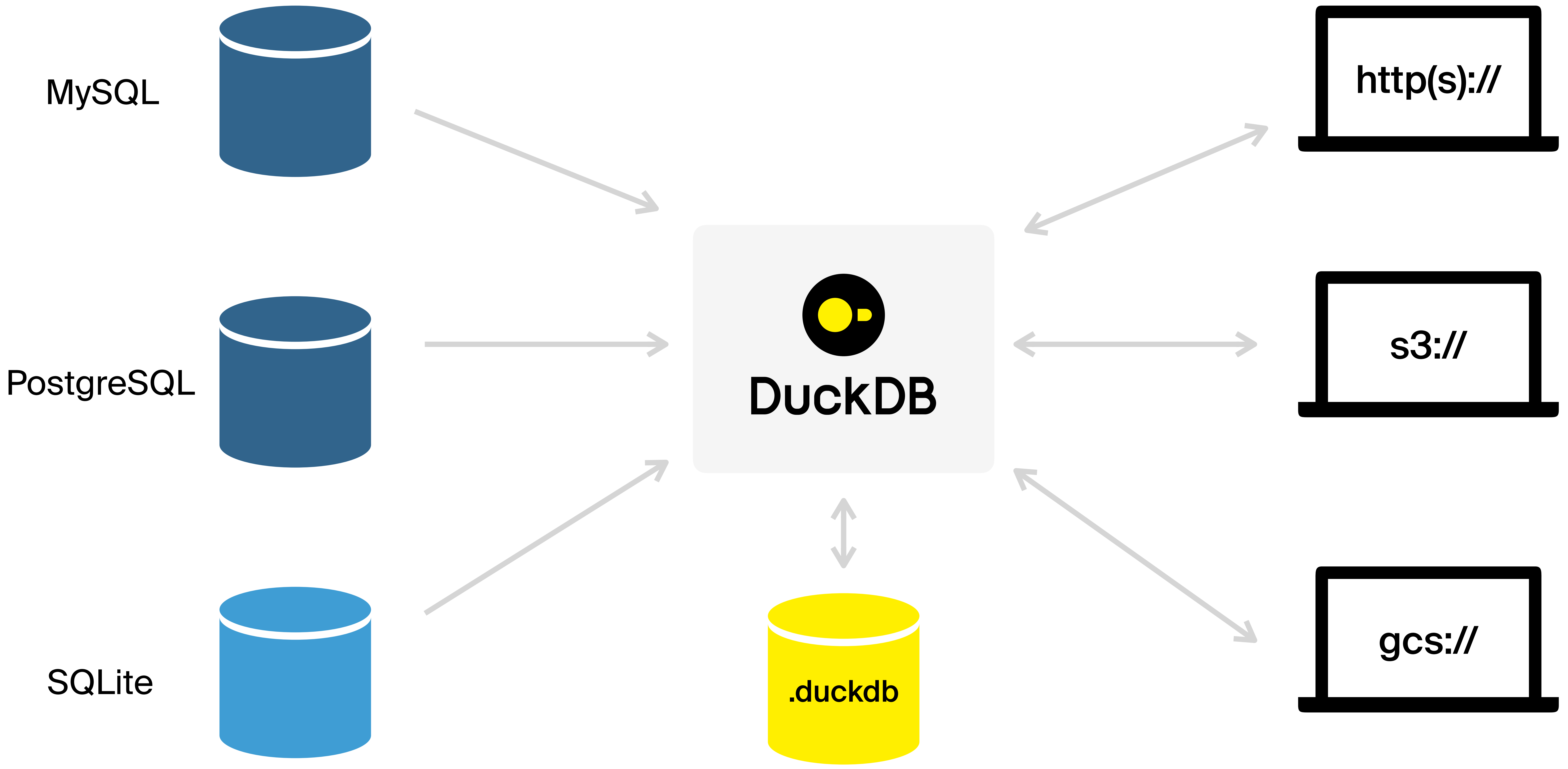
```
%%sql
SELECT h.ticker, h.when, price * shares AS value
FROM holdings h
ASOF JOIN prices p
       ON h.ticker = p.ticker
      AND h.when >= p.when;
```

Running query in 'duckdb'

Out[5]:

ticker	when	value
STCK1	00:00:30	119.0412
STCK1	00:01:30	67.7376
STCK1	00:02:30	554.5074
STCK2	00:00:30	521.9585
STCK2	00:01:30	1406.8068
STCK2	00:02:30	1441.4436

Data sources and destinations



Organizations around DuckDB

Open-source project (MIT license)

DuckDB

Commercial support

DuckDB Labs

Stay in touch

discord.duckdb.org

@duckdb

duckdb.org