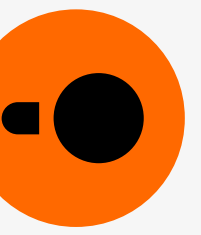


# An overview of DuckDB



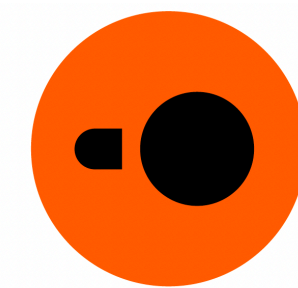
## Gábor Szárnyas

- 2014–2023: PhD + postdoc
- Research on databases and benchmarks



## Developer Relations @ DuckDB Labs

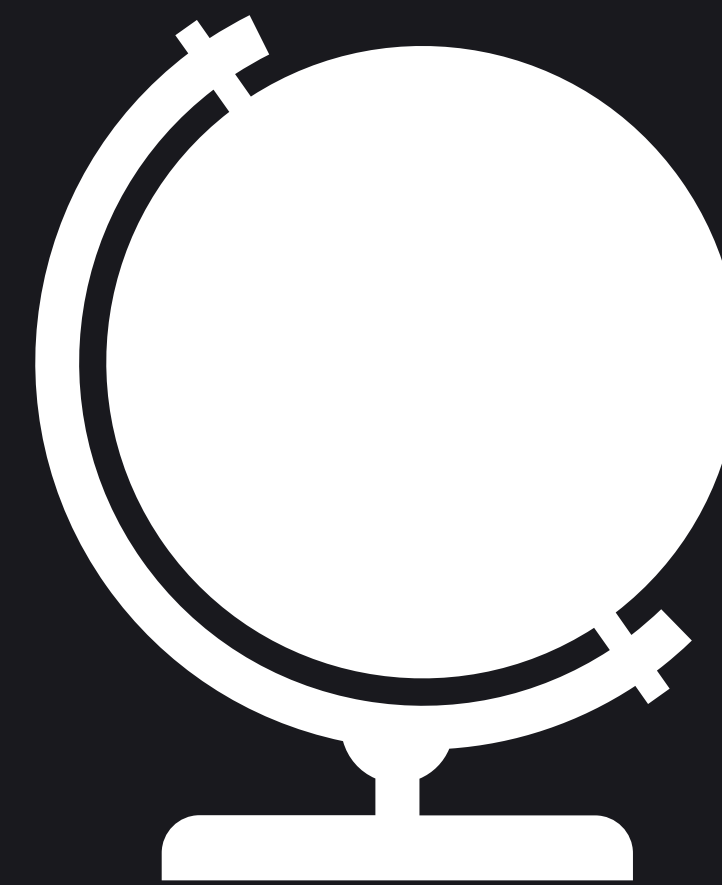
- Startup with 18 people
- Based in Amsterdam



**DuckDB Labs**



**Context**





**DHH** 

@dhh

The fact that mainstream developer laptops now ship with **16-core, 3nm CPUs** is one of those **THE PREMISE CHANGED** fundamentals [...].

**Time to reconsider some fundamentals of where things run, how, and when.**

6:15 PM · Oct 31, 2023



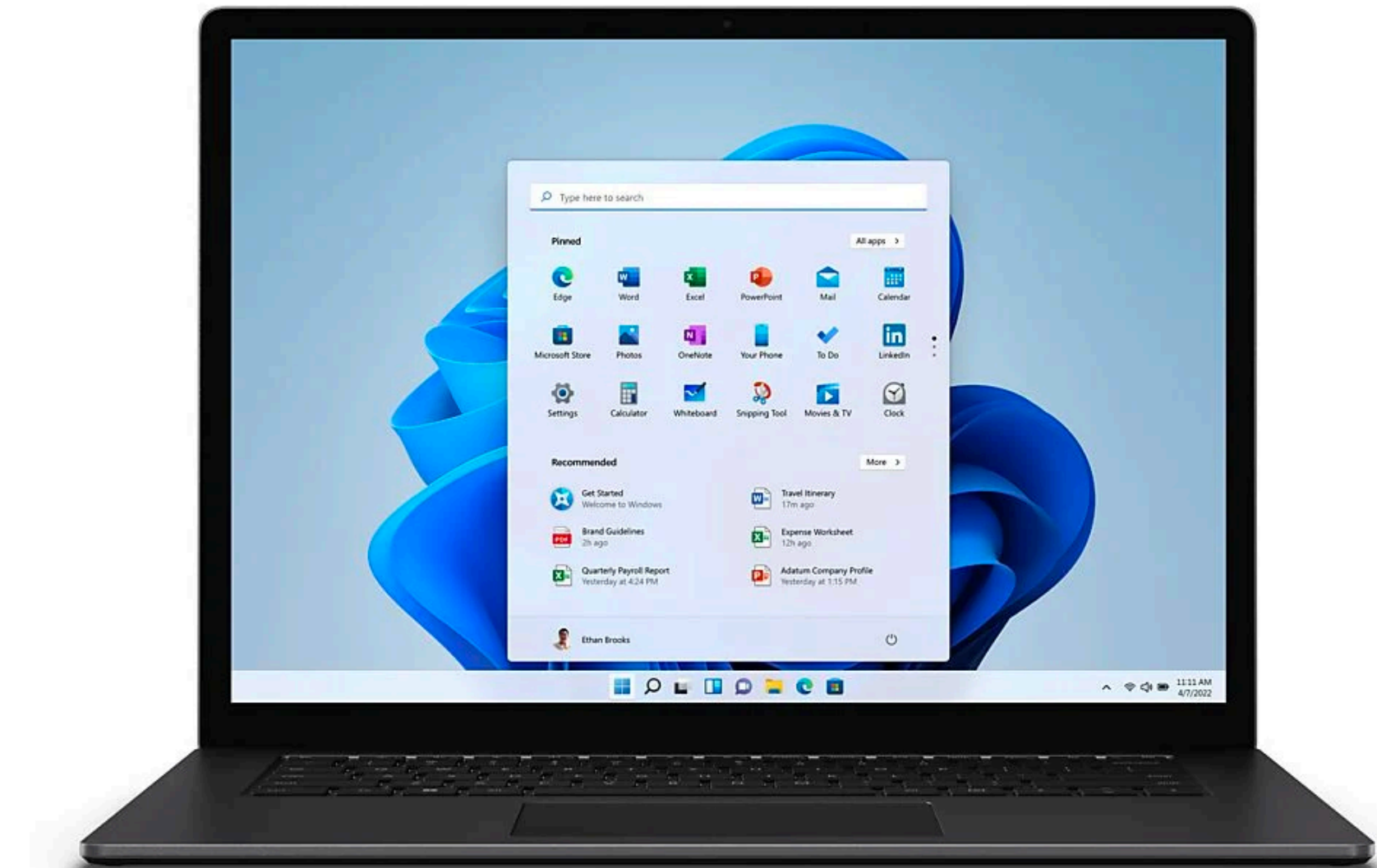
New



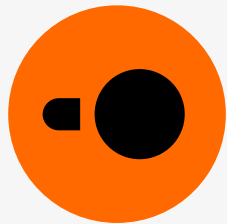
16-core CPU  
40-core GPU  
48GB Unified Memory  
1TB SSD Storage<sup>1</sup>



# DuckDB is an analytical database system built for powerful end-user devices



# DuckDB's key properties



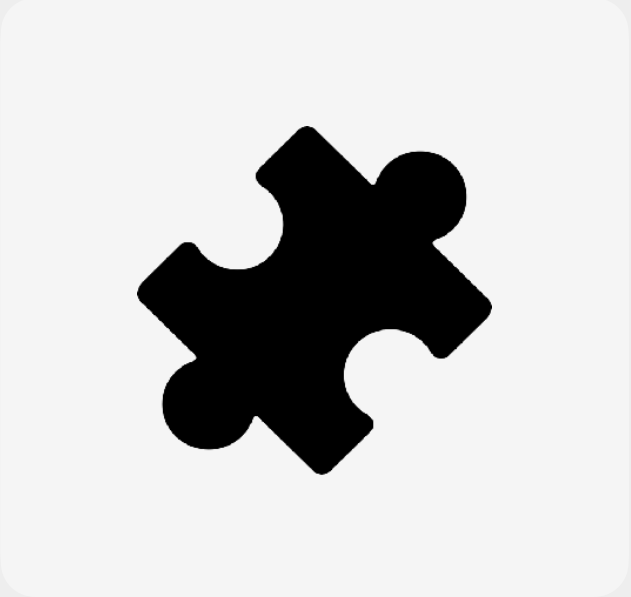
An analytical SQL database

Built to be portable and fast

Developed since 2018

Written in C++11

Open-source under the MIT license



In-process



Portable



Fast



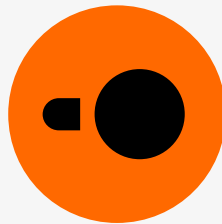
Open-source



# Deployment model



# Client-server setup



## Client application

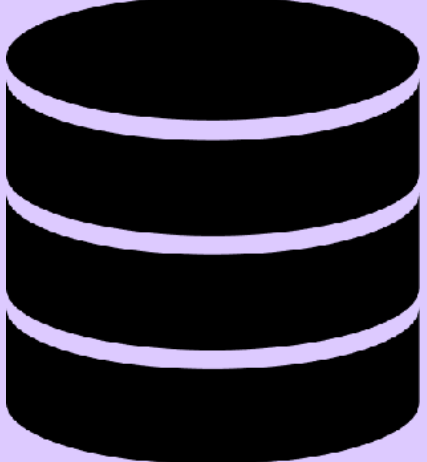
```
import psycopg

con = psycopg.connect(
    host="3.218.70.181",
    user="your_user",
    password="your_password",
    dbname="your_db"
)
con.execute("SELECT ...")
```



Bottleneck

## Database server

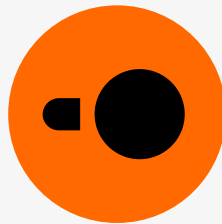


Connection setup and authentication

Pay for, configure, operate



# Client-server setup



## Client application

```
import psycopg

con = psycopg.connect(
    host="3.218.70.181",
    user="admin",
    password="admin",
    dbname="your_db"
)
con.execute("SELECT ...")
```

**Impractical!**



Still a bottleneck



Run in a container, need to configure, adjust ports, ...

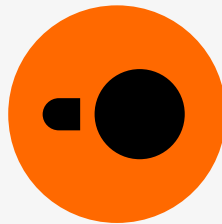


## Client application

```
import duckdb  
duckdb.sql("SELECT ...")
```



No configuration  
No authentication  
No client protocol

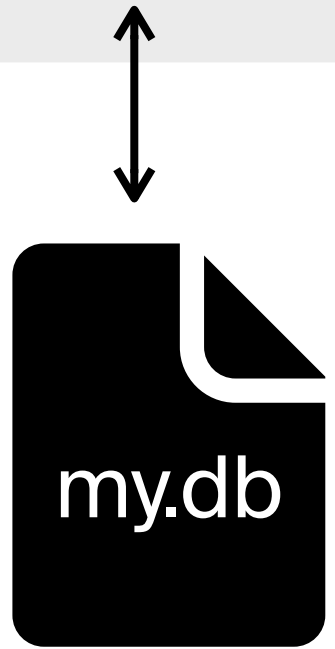


## Client application

```
import duckdb  
  
duckdb.sql("SELECT ...")  
  
# for persistence  
  
con = duckdb.connect("my.db")  
con.sql("SELECT ...")
```



No configuration  
No authentication  
No client protocol

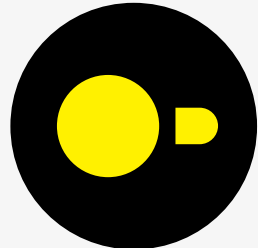


Single-file format  
containing all tables

# Database systems



In-process



DuckDB

Client-server



VERTICA

Transactional

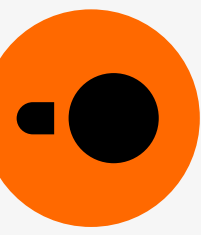
Analytical



**Portable**



# Installing DuckDB

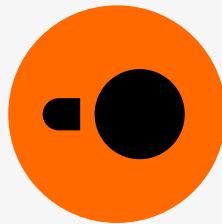


You can get started with DuckDB in **<15 seconds** on most popular platforms

This includes:

- Typing the commands
- Downloading the package
- Installing the package
- Launching DuckDB

# DuckDB packages



```
pip install duckdb
```



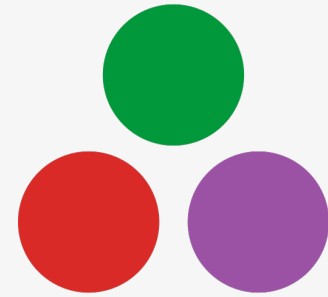
```
npm install duckdb
```



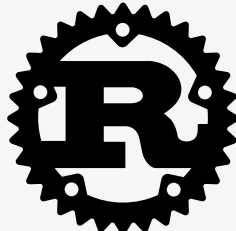
```
install.packages("duckdb")
```



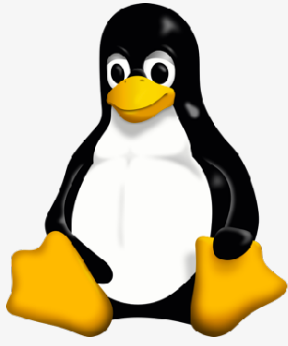
```
org.duckdb:duckdb_jdbc
```

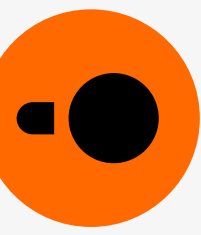


```
Pkg.add("DuckDB")
```



```
cargo add duckdb
```





# Why is installation so fast?

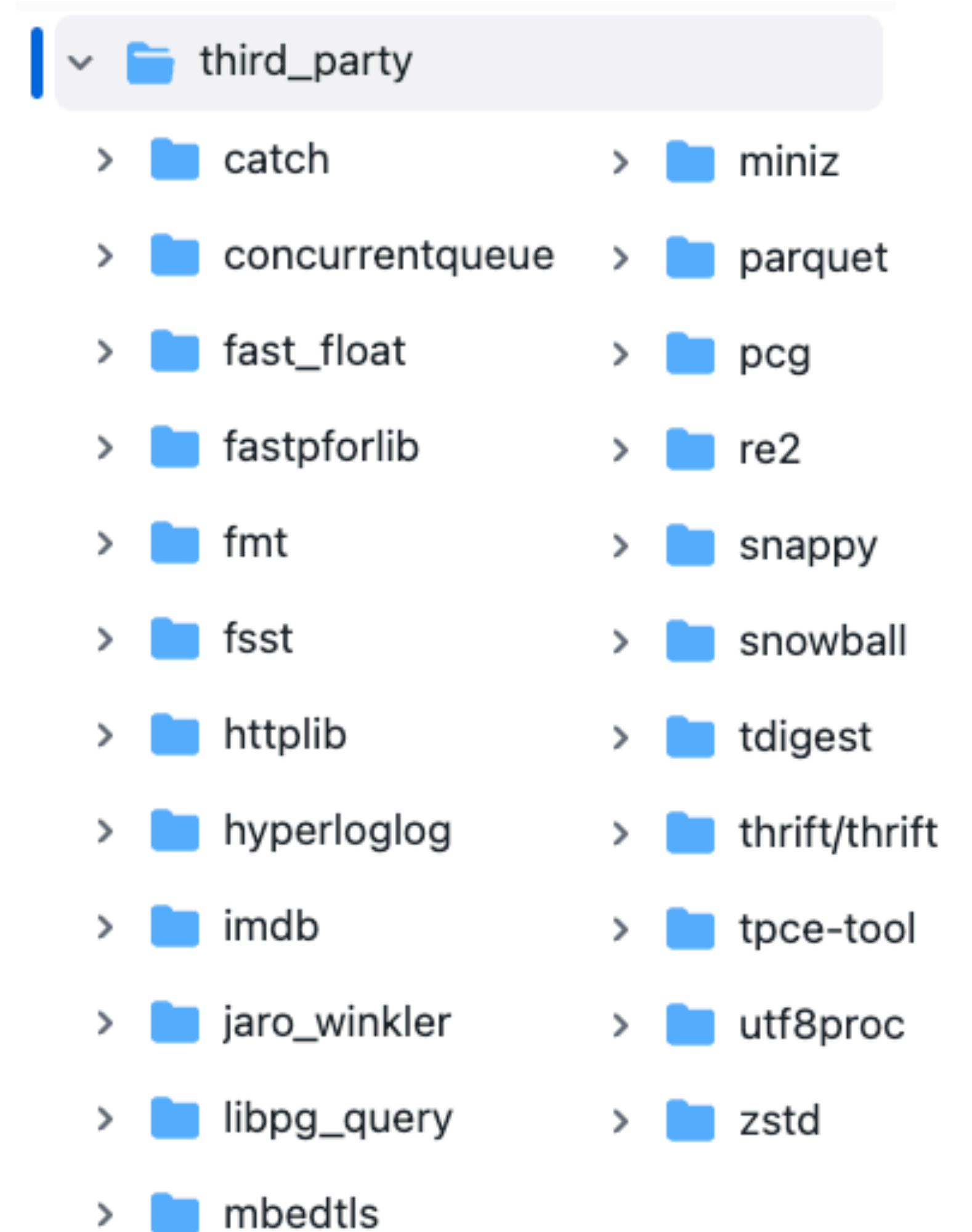
DuckDB has zero external dependencies

Dependencies are vendored in the codebase

Pure C/C++ codebase

Portable anywhere with a C++11 compiler

Small binary packages





# WebAssembly (Wasm)



```
DuckDB Web Shell
Database: v0.9.2
Package: @duckdb/duckdb-wasm@1.28.1-dev39.0

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

duckdb> INSTALL tpch;
...> LOAD tpch;
...> CALL dbgen(sf=1);
...> DESCRIBE;
...> COPY customer TO 'customer.parquet';
...>

█
Elapsed: 10.751 s

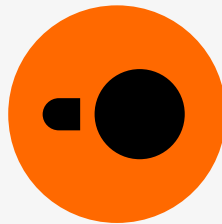
duckdb> .files download customer.parquet
Copied file: customer.parquet
```



Fast



# CSV reader performance



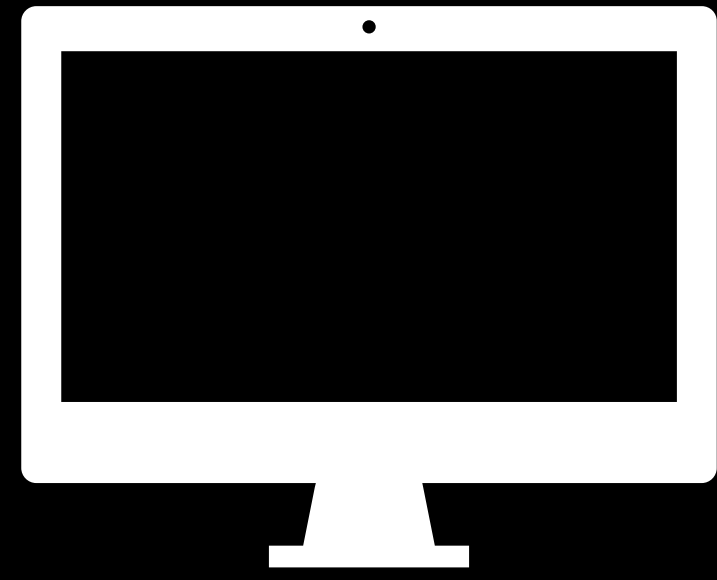
Test data: LDBC social network data set

Setup: M2Pro CPU, 32GB RAM, DuckDB v0.9.3-dev

Size	CSV disk usage	Load time	Database size
S	7 GB	6.5s	2.5 GB
M	24 GB	20.9s	8.5 GB
L	73 GB	67.1s	26.0 GB

≈3x compression

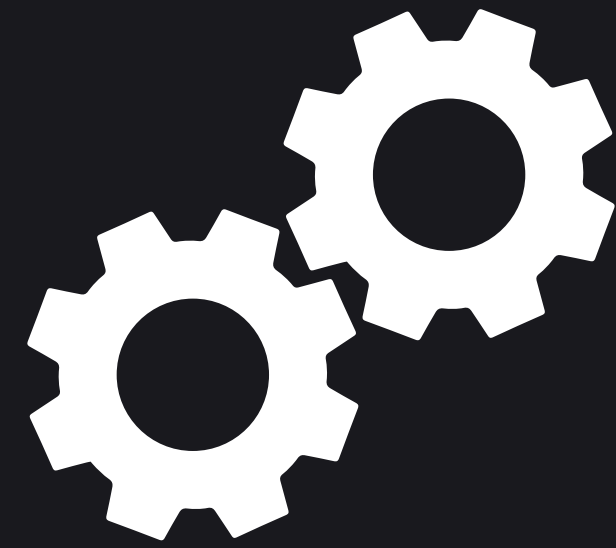
>1 GB/s for reading CSV, parsing, and writing to DuckDB



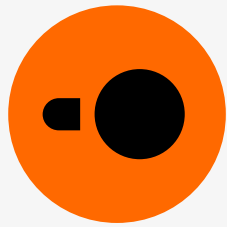
Demo



# Internals



# Storage



## row-based

time id content length


## column-based

time id content length


# Storage



## row-based

time id content length

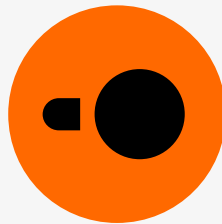
blue	blue	blue	blue
green	green	green	green
yellow	yellow	yellow	yellow
red	red	red	red

## column-based

time id content length

gray	gray	gray	gray
gray	gray	gray	gray
gray	gray	gray	gray
gray	gray	gray	gray

# Storage



## row-based

time id content length

blue	blue	blue	blue
green	green	green	green
yellow	yellow	yellow	yellow
red	red	red	red

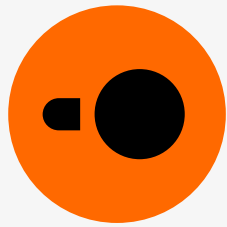
## column-based

time id content length

cyan	pink	blue	light red
cyan	pink	blue	light red
cyan	pink	blue	light red
cyan	pink	blue	light red



# Execution



### row-based

time	id	content	length
blue	blue	blue	blue
green	green	green	green
yellow	yellow	yellow	yellow
red	red	red	red

### column-based

time	id	content	length
cyan	pink	blue	red
cyan	pink	blue	red
cyan	pink	blue	red
cyan	pink	blue	red

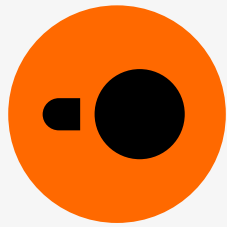
### tuple-at-a-time

time	id	content	length
blue	blue	blue	blue
green	green	green	green
yellow	yellow	yellow	yellow
red	red	red	red

### column-at-a-time

time	id	content	length
cyan	gray	gray	red
cyan	gray	gray	red
cyan	gray	gray	red
cyan	gray	gray	red

# Execution



### row-based

time	id	content	length

### column-based

time	id	content	length

### tuple-at-a-time

time	id	content	length

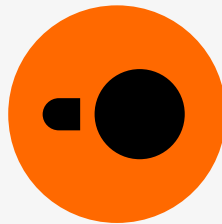
### column-at-a-time

time	id	content	length

### vectorized

time	id	content	length

# Vectorized execution



vectorized			
time	id	content	length
teal	grey	grey	light red
teal	grey	grey	light red
dark teal	grey	grey	red
dark teal	grey	grey	red
yellow	grey	grey	pink
yellow	grey	grey	pink
orange	grey	grey	magenta
orange	grey	grey	magenta

2 row groups

thread 1

L1 cache

thread 2

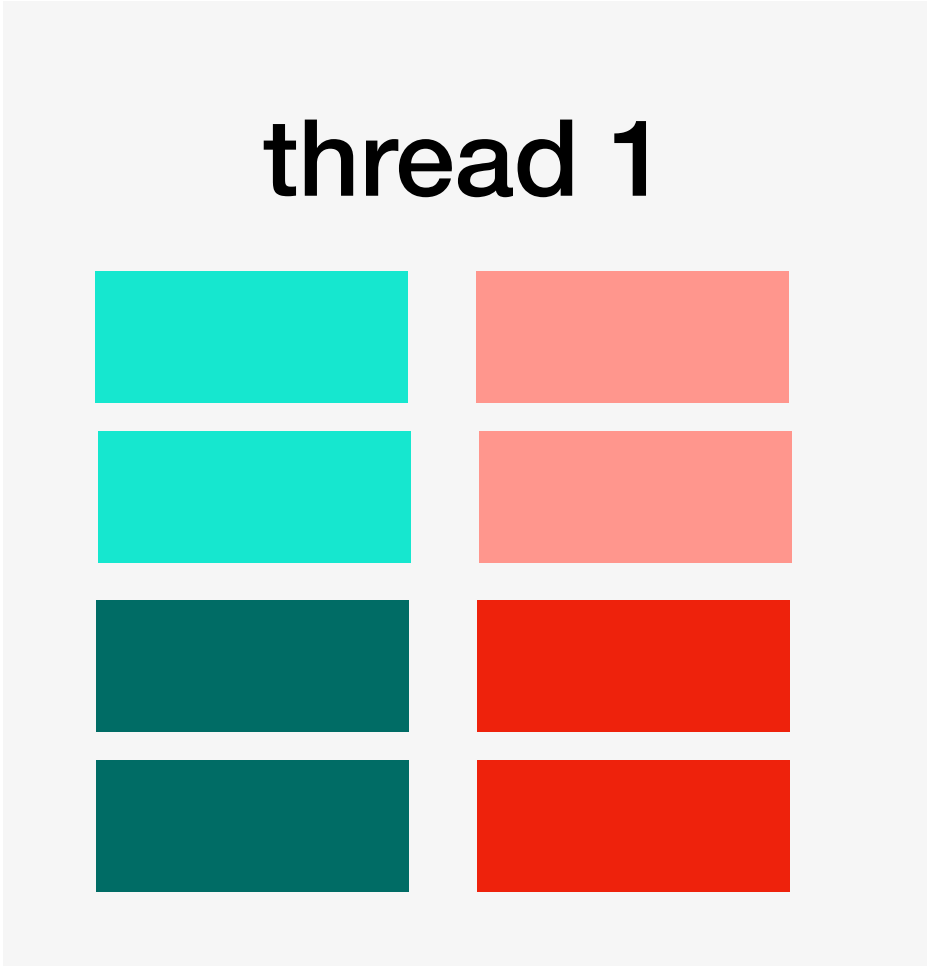
L1 cache

# Vectorized execution

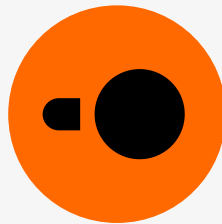


vectorized			
time	id	content	length

2 row groups

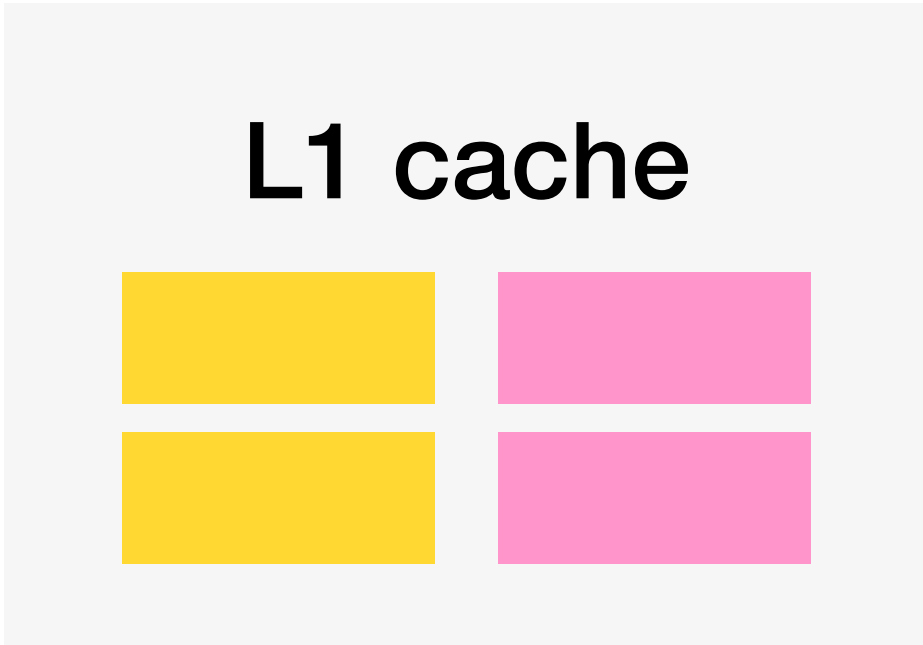
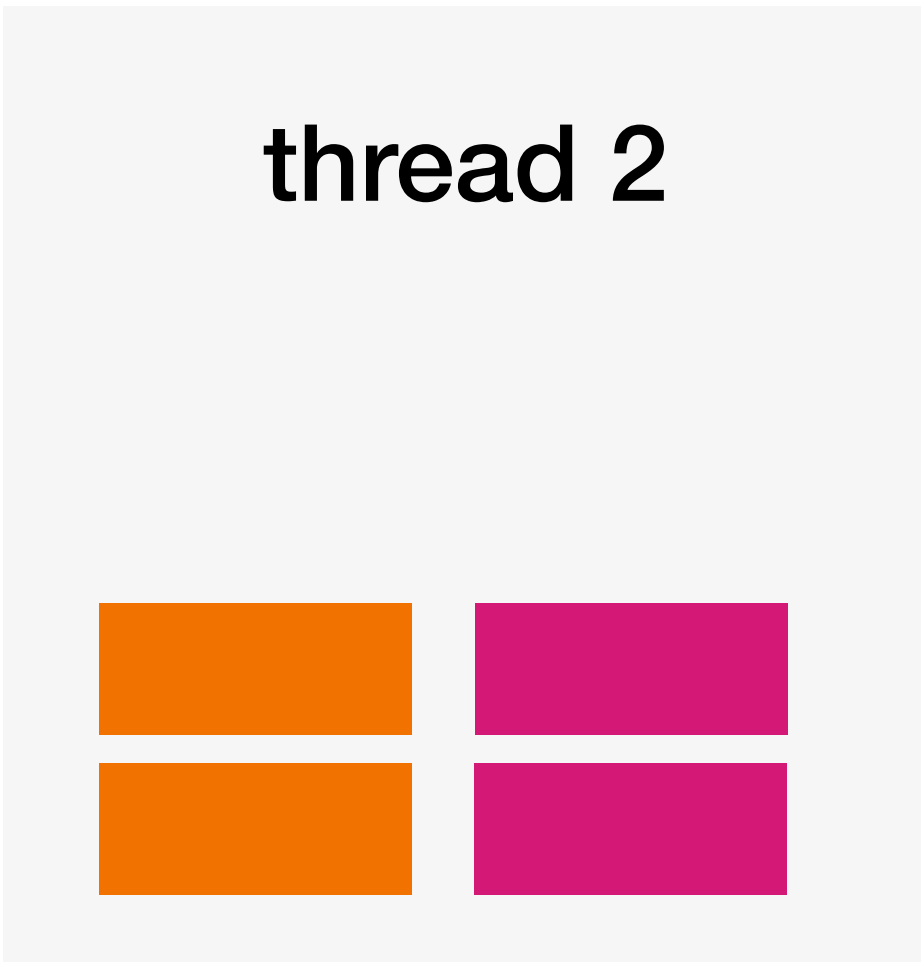
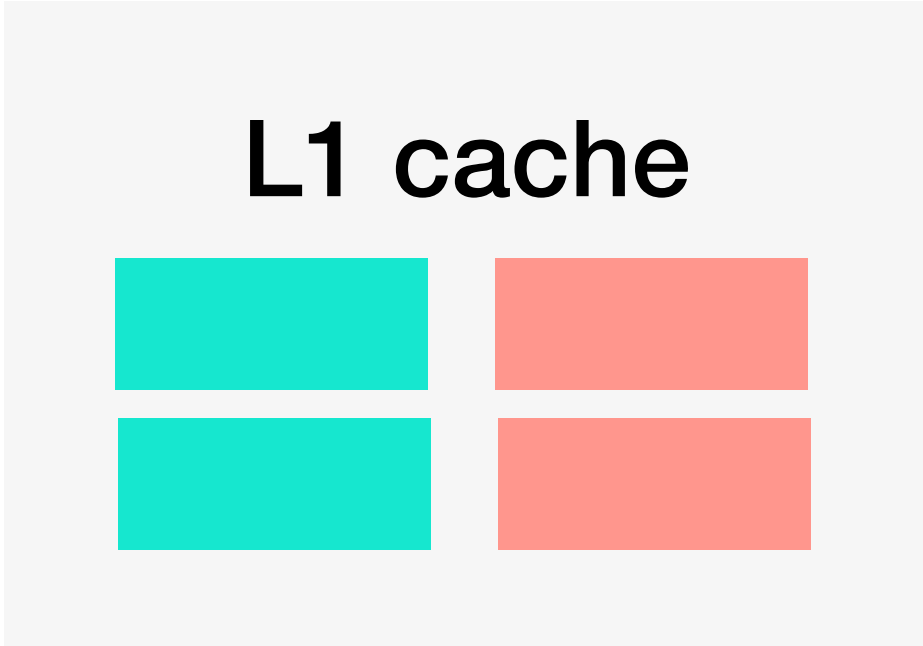
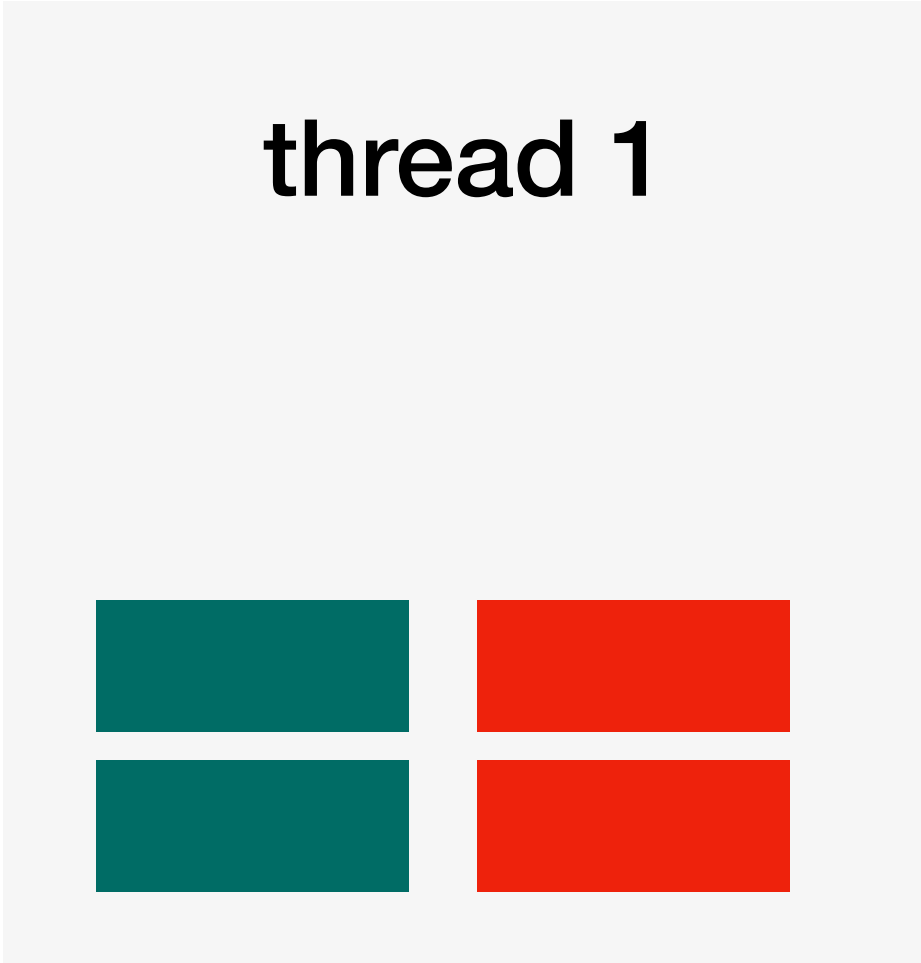


# Vectorized execution

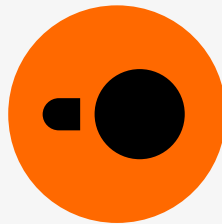


vectorized			
time	id	content	length

2 row groups



# Vectorized execution

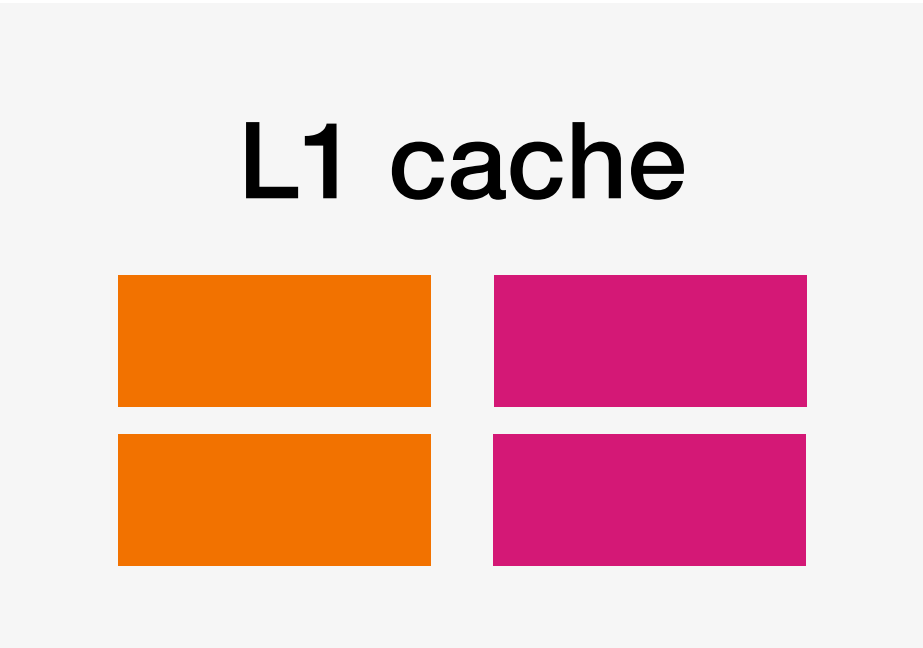
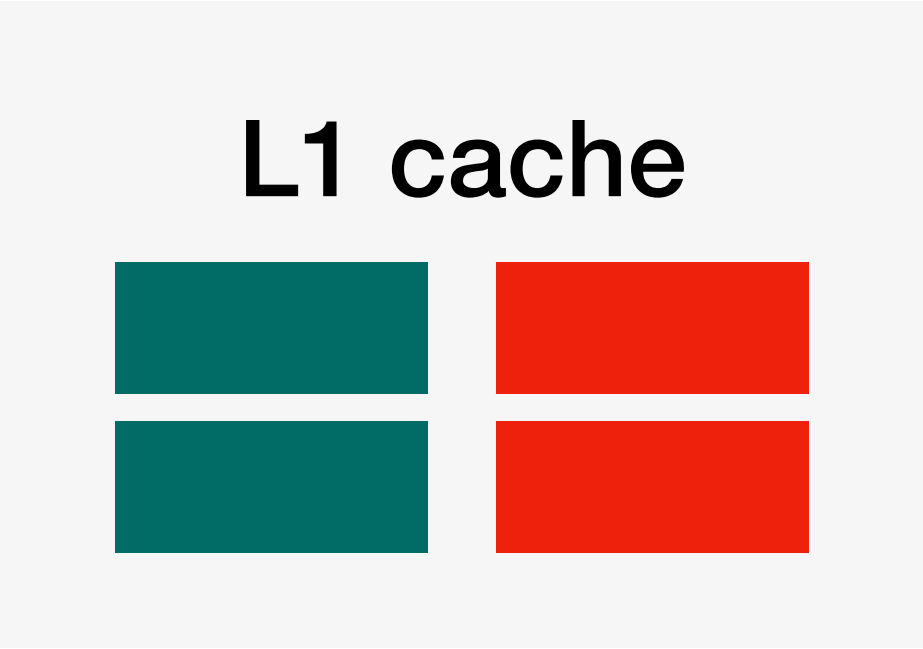


vectorized			
time	id	content	length

2 row groups

thread 1

thread 2



Vectors fit into L1 cache (32–128kB)

Modern compilers auto-vectorize code using SIMD instructions

# Indexing: Zone maps



For each column, DuckDB creates zone maps (min-max indexes)

min	max
Dec 7	Dec 10

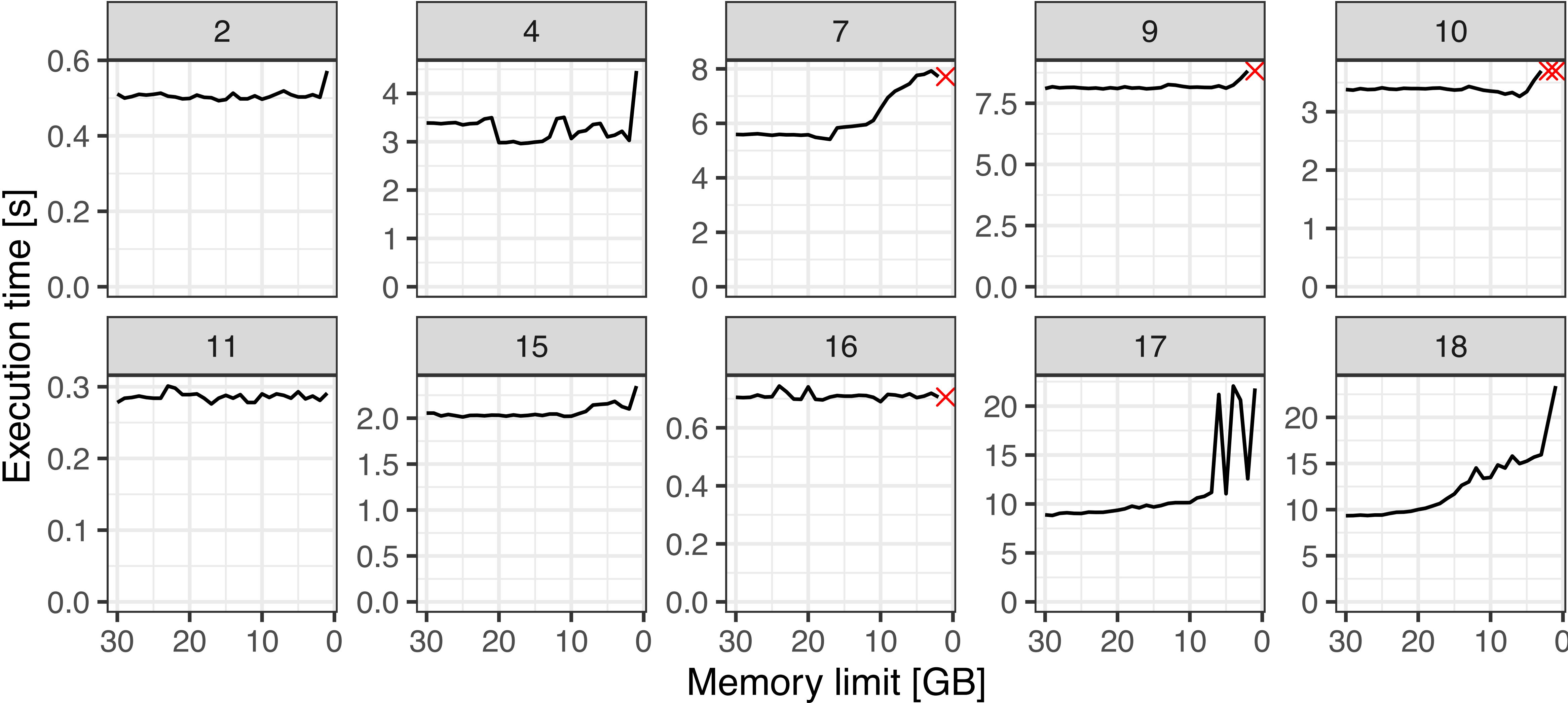
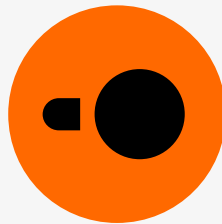
min	max
Dec 10	Dec 14

time	id	content	length
Dec 7			74
Dec 7			109
Dec 8			67
Dec 10			63
Dec 10			95
Dec 13			113
Dec 13			14
Dec 14			8

min	max
63	109

min	max
8	95

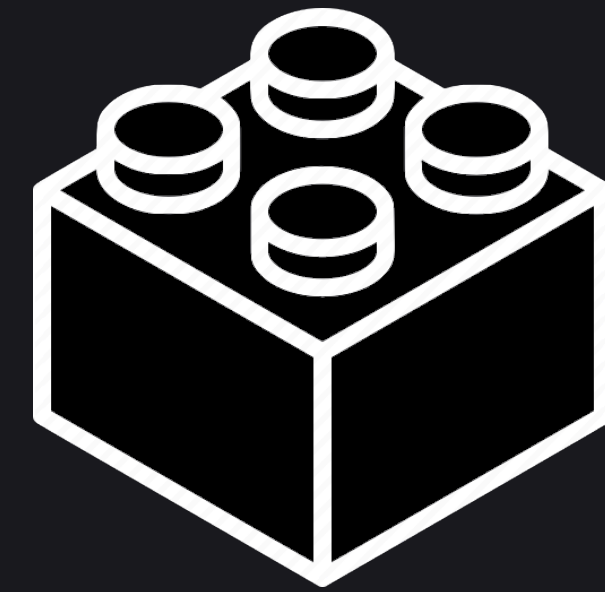
# Larger-than-memory execution: TPC-H queries on SF100



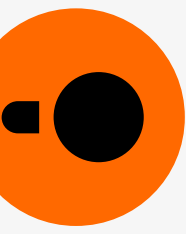




# Extensions



# Extensions



- Powerful extension mechanism:
  - new types and functions
  - data formats
  - operators
  - SQL syntax
  - memory allocator
- Many DuckDB features are implemented as extensions
  - httpfs
  - JSON
  - Parquet

☰ README.md

## DuckDB Extension Template [↗](#)

This repository contains a template for creating a DuckDB extension. The main goal of this template is to allow users to easily develop, test and distribute their own DuckDB extension. The main branch of the template is always based on the latest stable DuckDB allowing you to try out your extension right away.

### Getting started [↗](#)

First step to getting started is to create your own repo from this template by clicking `Use this template`. Then clone your new repository using

```
git clone --recurse-submodules https://github.com/
```



# Parquet + httpfs extensions to query stock data

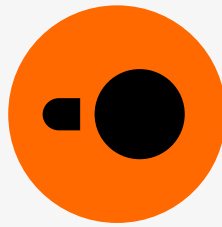
```
SELECT avg(price)
FROM 'https://duckdb.org/data/prices.parquet'
WHERE ticker = 'MSFT';
```

avg(price) double
2.0

It's not a full download:

- HTTP range requests so seek to the required data
- Only touch the ticker and price columns

# Wasm Shell: Spatial extension



The screenshot shows a web browser window titled "DuckDB Shell" with the URL "shell.duckdb.org". The main content is a terminal window where the following SQL commands were executed:

```
duckdb> INSTALL spatial;
...> LOAD spatial;
...> CREATE TABLE nyc AS SELECT
...>     borough,
...>     st_union_agg(geom) AS full_geom,
...>     st_area(full_geom) AS area,
...>     st_centroid(full_geom) AS centroid,
...>     count(*) AS count
...>     FROM st_read('https://raw.githubusercontent.com/duckdb/duckdb_spatial/main/test/da
..>> ta/nyc_taxi/taxi_zones/taxi_zones.shp')
...> GROUP BY borough;
...> SELECT borough, area, centroid::VARCHAR AS centroid, count FROM nyc;
```

The result of the query is displayed as a table with the following data:

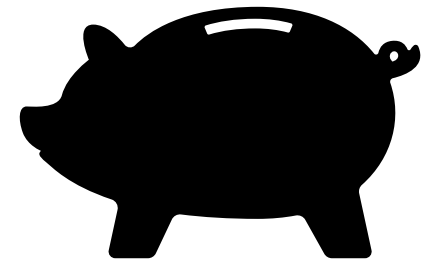
borough	area	centroid	count
EWR	79039526.6992143	POINT (935996.8210162065 191376.74953083202)	1
Queens	3114673359.8136516	POINT (1034369.2547920293 196362.59567639793)	69
Bronx	1186861426.418706	POINT (1021174.6612852946 249938.27047567436)	43
Manhattan	633962173.9489708	POINT (993373.4020104649 222568.94505742347)	69
Staten Island	1623229445.473062	POINT (941628.5554551884 150932.2710170759)	20
Brooklyn	1897084165.3549132	POINT (998191.0180332123 174468.03580197674)	61

At the bottom of the terminal window, it says "Elapsed: 02.533 s".



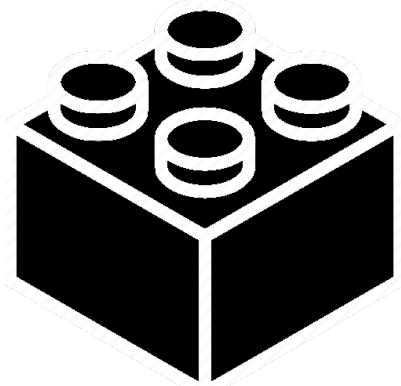
# Use cases





## Saving costs:

- Replacing (parts of) data warehouse jobs
- Running computation locally



## Building block:

- Just to perform a simple step
- E.g., converting from Parquet to CSV

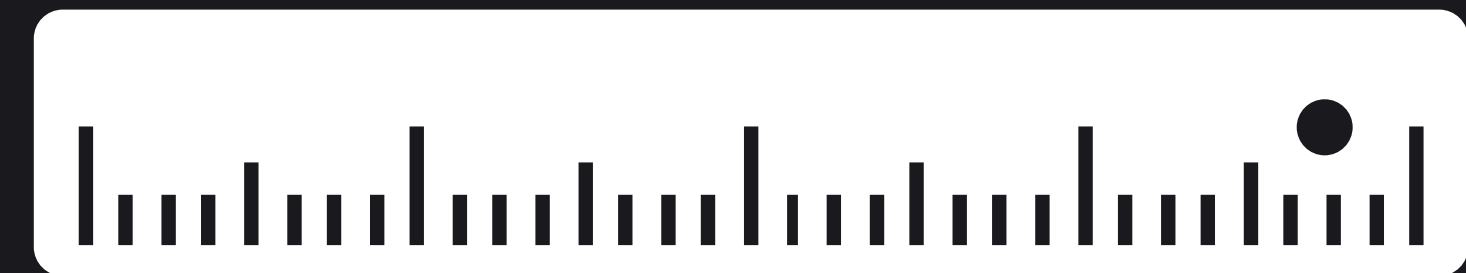


## Education:

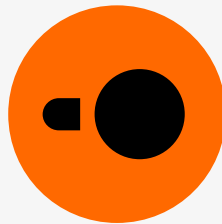
- Easy-to-install, open, standards-compliant system
- No configuration, no DBA



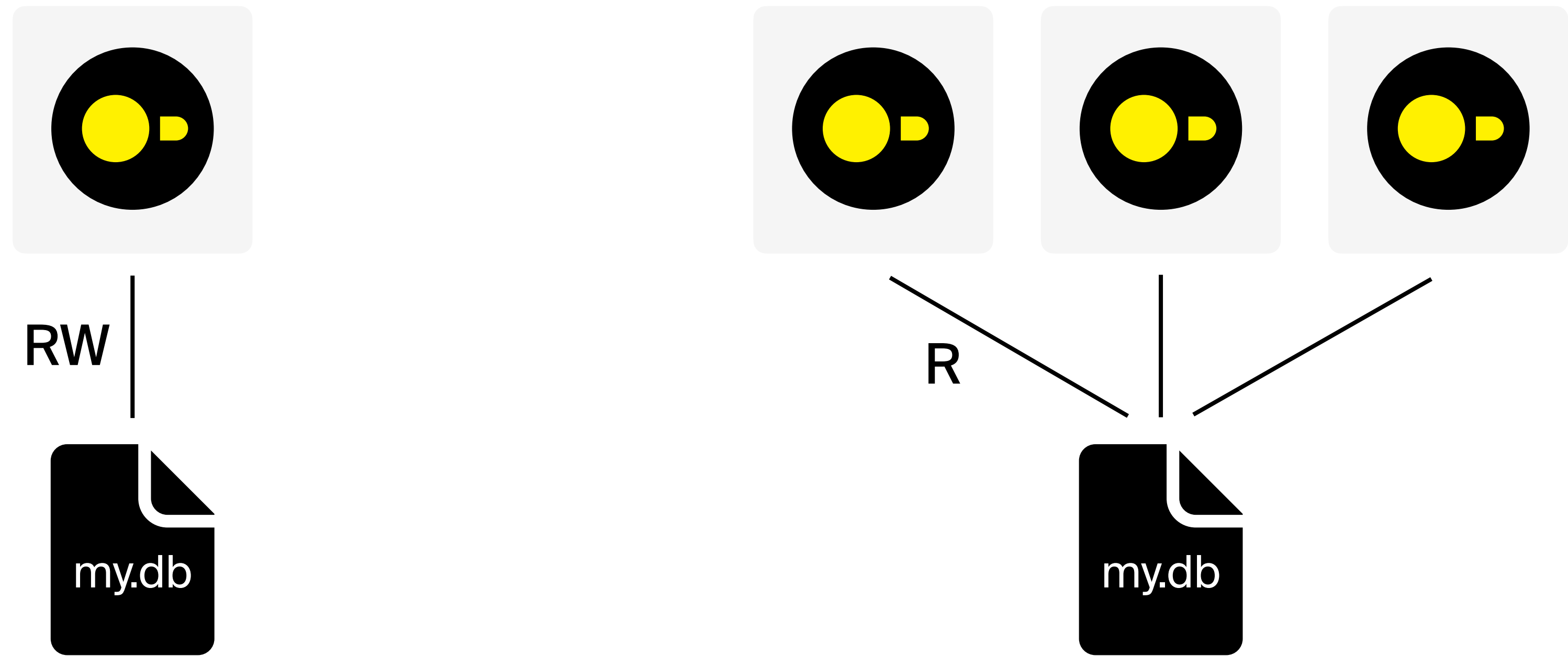
# Limitations



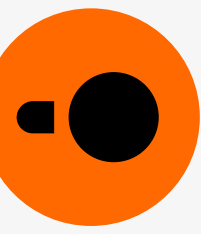
# Concurrency control



- ACID compliance via multi-version concurrency control (MVCC)
- Recovery using a write-ahead log (WAL)
- But: Not a good fit for write-heavy workloads







# Distributed execution

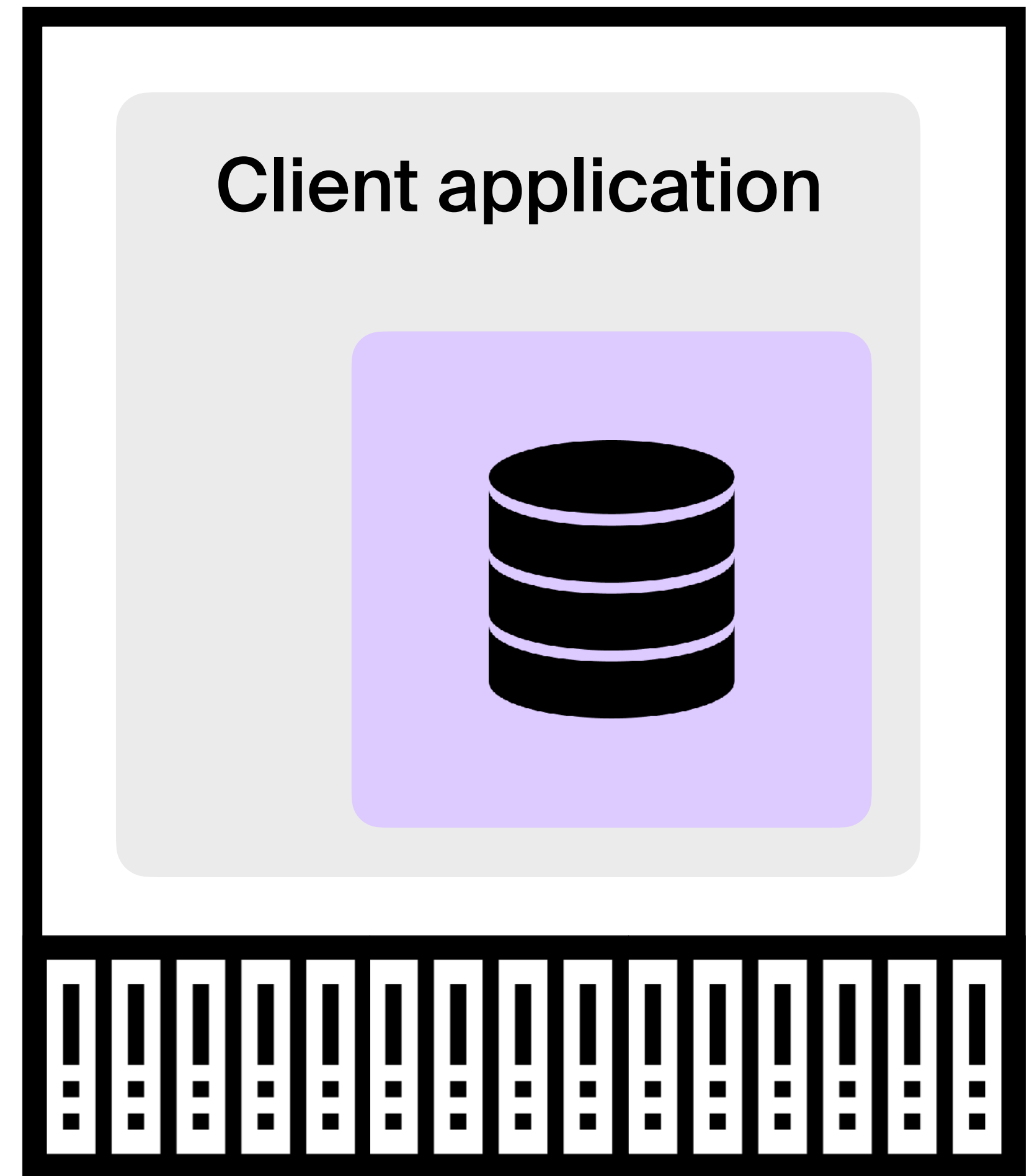
DuckDB only supports **single-node** execution

DuckDB can **scale up**:

- EC2 r6id.32xlarge: 1TB RAM, \$10/h
- EC2 x1e.32xlarge: 4TB RAM, \$28/h

Allows scaling for TBs of data

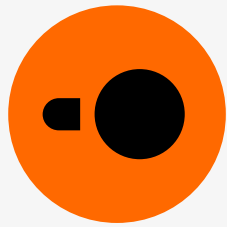
Store the data in S3, run short bursts of workloads





# The DuckDB landscape

# DuckDB versions



v0.9

Current version

v0.10

Early next year

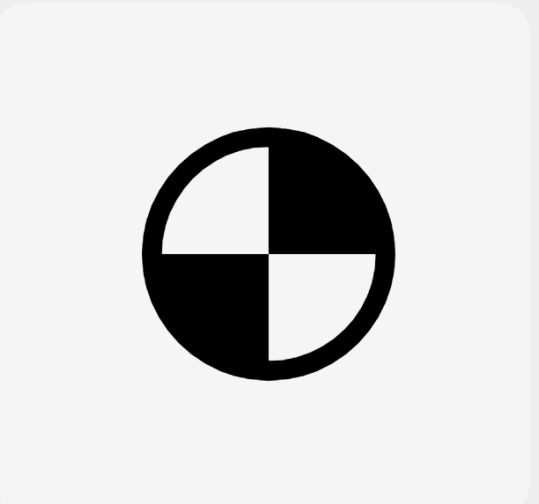
v1.0

Later next year

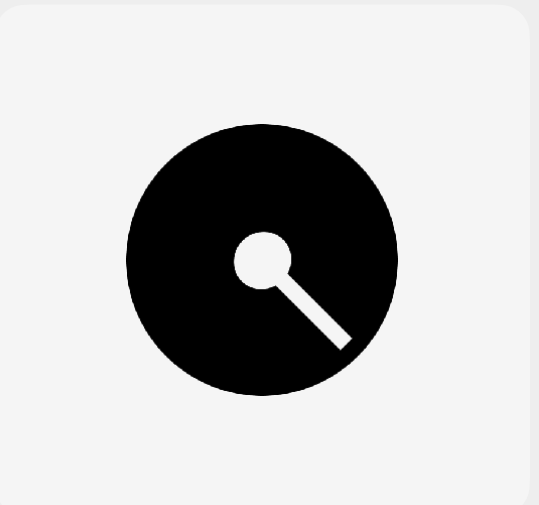
v1.0



Stable file format

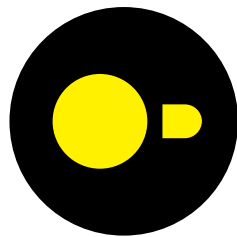


Stability and maturity improvements

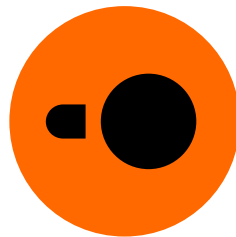


Performance optimizations

# Organizations around DuckDB



**DuckDB**



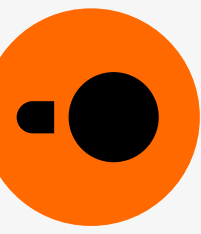
**DuckDB Labs**



**MotherDuck**



**Wrapping up...**



# DuckDB is old-school with state of the art internals

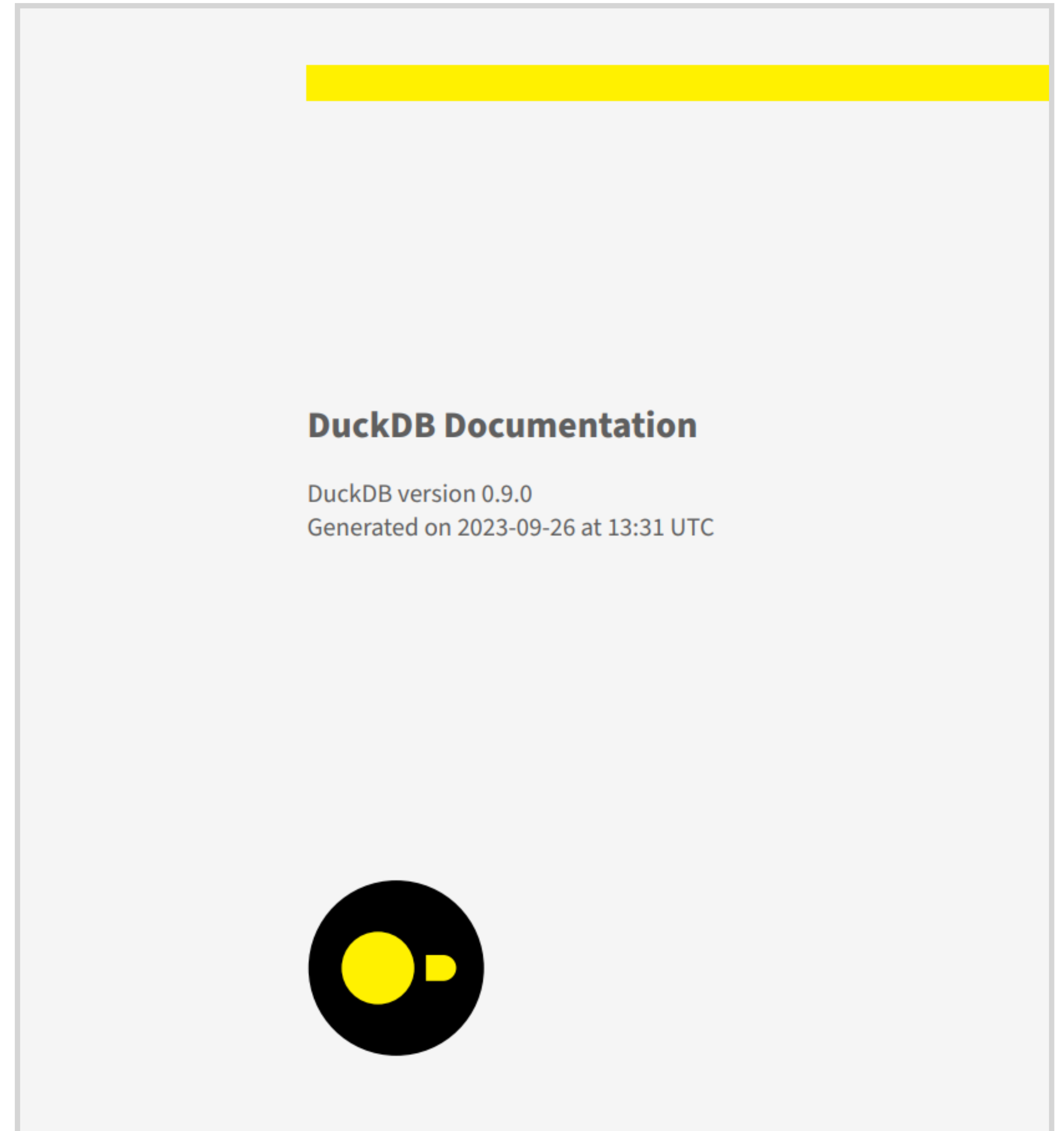
**Classic open-source project**

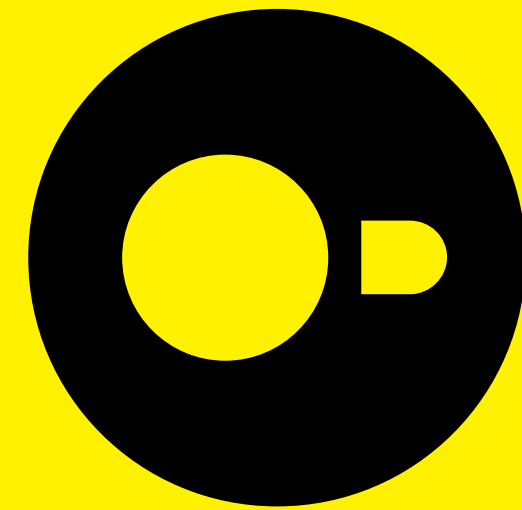
**Full-fledged CLI client**

**Works offline**

**No vendor lock-in**

```
EXPORT DATABASE 'my_db' (FORMAT CSV);  
EXPORT DATABASE 'my_db' (FORMAT PARQUET);
```





**Give DuckDB a spin!**

# Google Colab, shell.duckdb.org



DuckDB\_in\_Jupyter\_Notebooks.ipynb ☆

File Edit View Insert Runtime Tools Help Changes will not be saved

+ Code + Text Copy to Drive

### Connecting to DuckDB

Connect jupysql to DuckDB using a SQLAlchemy-style connection string.

```
[ ] %sql duckdb:///memory:  
# %sql duckdb:///path/to/file.db
```

### Querying DuckDB

Single line SQL queries can be run using `%sql` at the start of a line. Quer highlighting!

```
[ ] %sql SELECT 'Off and flying!' as a_duckdb_column
```

a_duckdb_column	
0	Off and flying!

← → ↻ shell.duckdb.org

### DuckDB Web Shell

Database: v0.9.1  
Package: @duckdb/duckdb-wasm@1.27.1-dev134.0

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

```
duckdb> FROM 'https://duckdb.org/data/prices.csv';
```

ticker	when	price
APPL	2001-01-01 00:00:00	1
APPL	2001-01-01 00:01:00	2
APPL	2001-01-01 00:02:00	3
MSFT	2001-01-01 00:00:00	1
MSFT	2001-01-01 00:01:00	2
MSFT	2001-01-01 00:02:00	3
GOOG	2001-01-01 00:00:00	1
GOOG	2001-01-01 00:01:00	2
GOOG	2001-01-01 00:02:00	3

Elapsed: 146 ms

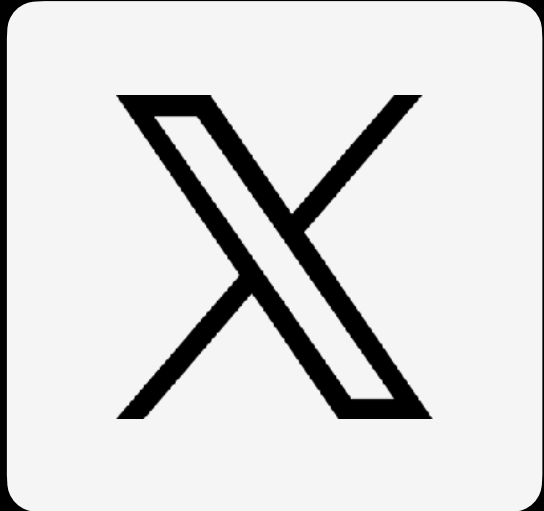
```
duckdb> █
```



# Stay in touch



[discord.duckdb.org](https://discord.duckdb.org)



[@duckdb](https://twitter.com/duckdb)



[duckdb.org](https://duckdb.org)

