An overview of DuckDB

Gábor Szárnyas
Developer Relations Advocate

DuckDB Labs

About me

Gábor Szárnyas

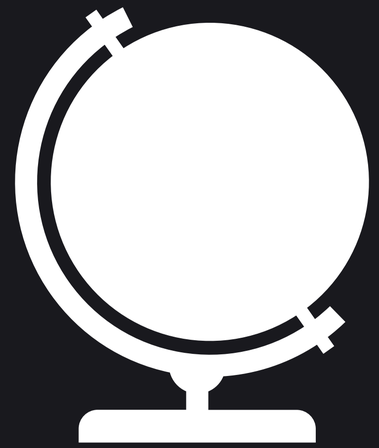
- 2014–2023: PhD + postdoc
- Research on databases and benchmarks

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.

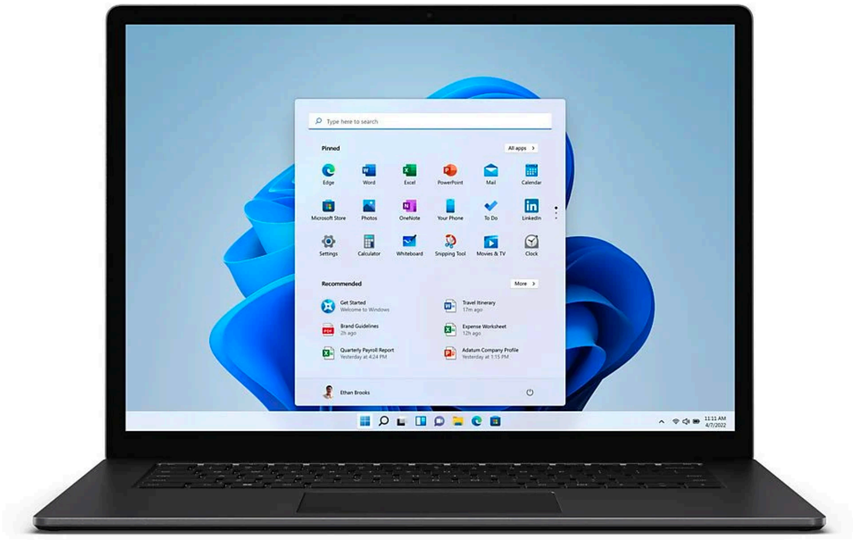
6:15 PM · Oct 31, 2023

New

M3 MAX

16-core CPU
40-core GPU
48GB Unified Memory
1TB SSD Storage[1]

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



DuckDB's key properties

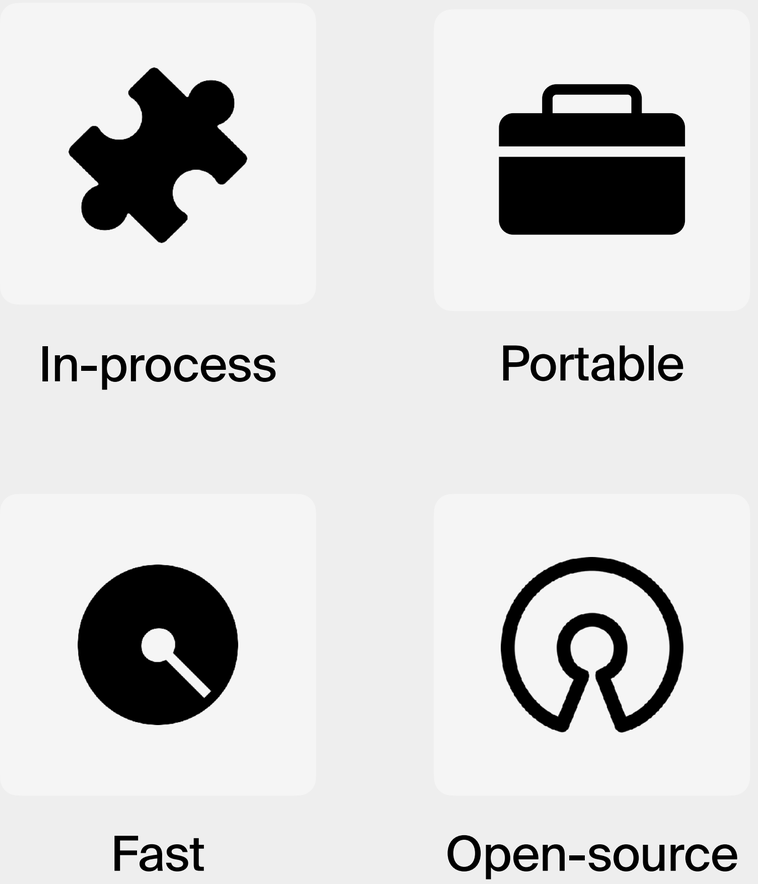
An analytical SQL database

Built to be portable and fast

Developed since 2018

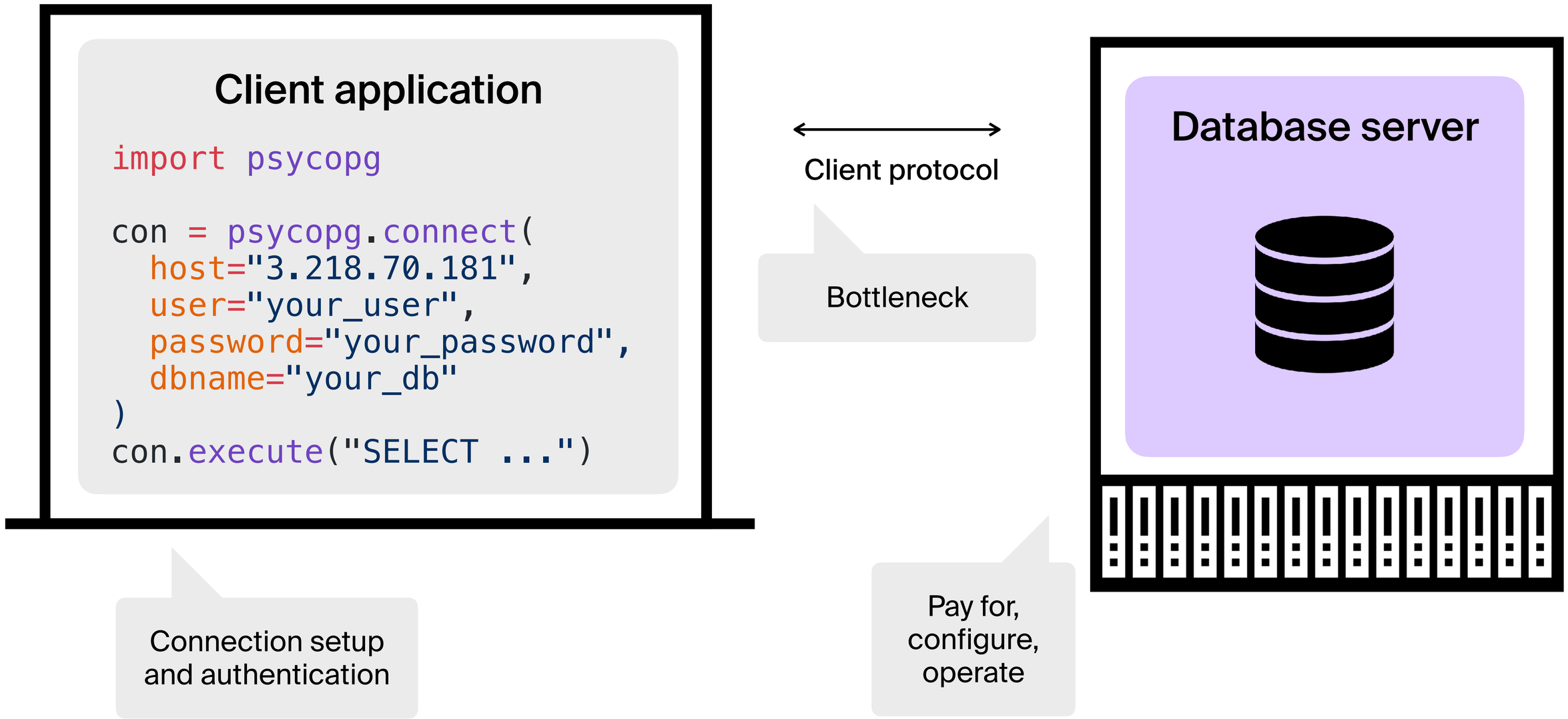
Written in C++11

Open-source under the MIT license

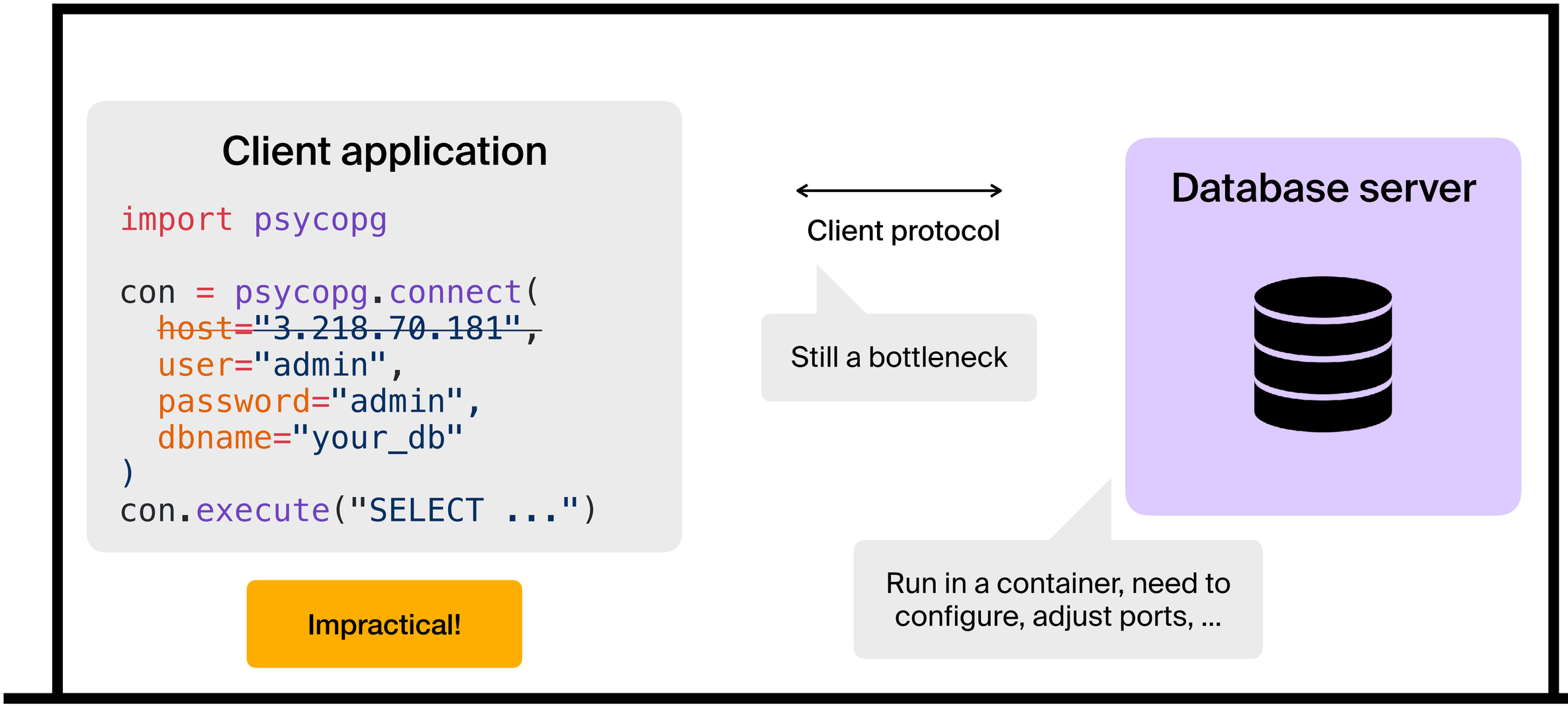


Deployment model

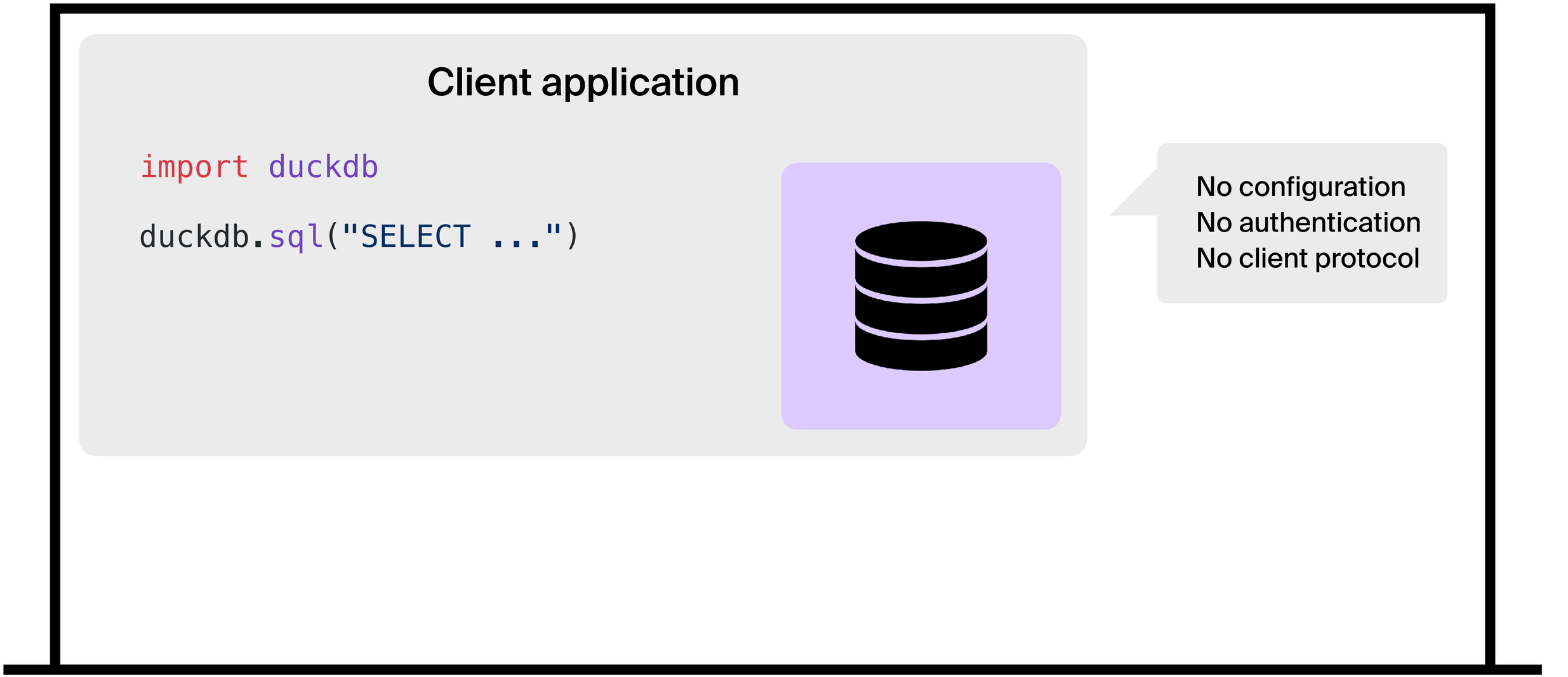
Client–server setup



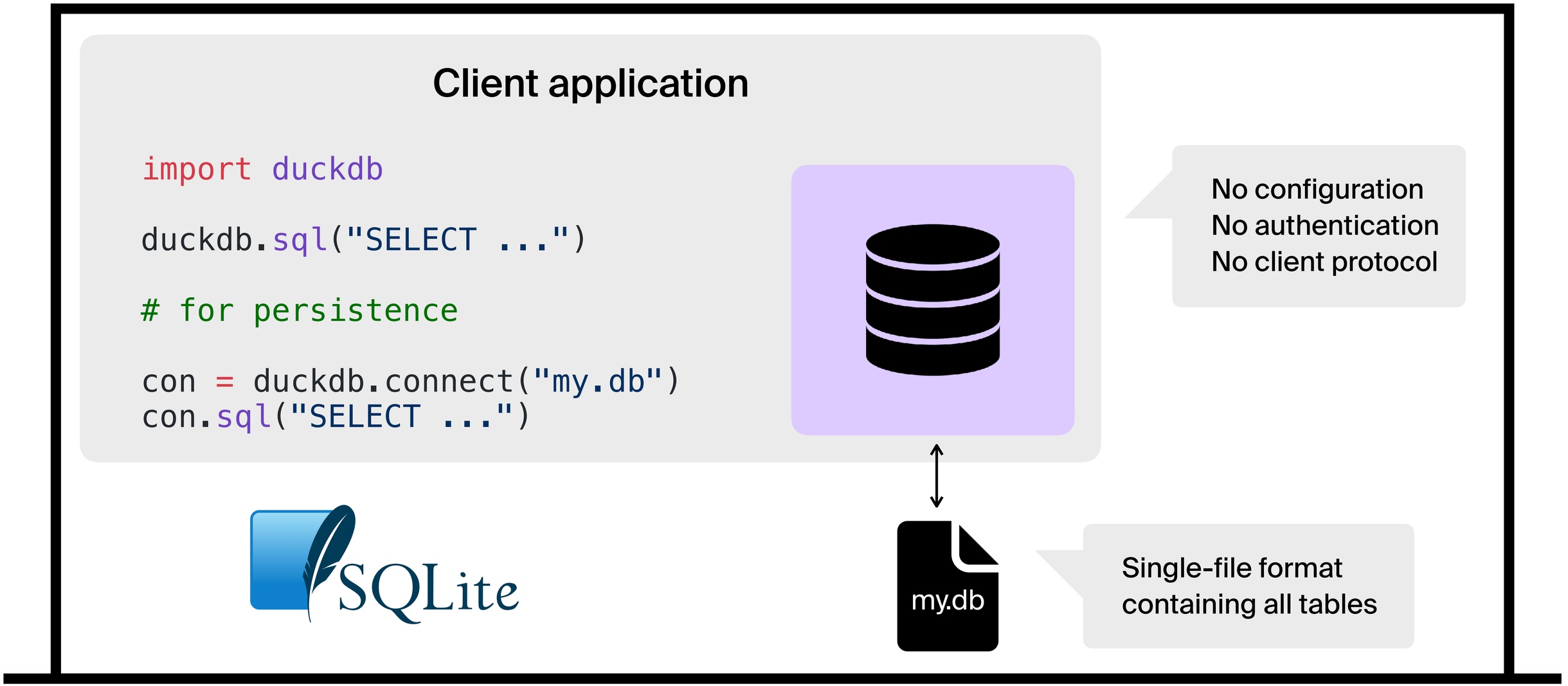
Client–server setup



In-process setup



In-process setup



Database systems

	Transactional	Analytical
In-process	SQLite	DuckDB
Client–server	MySQL, PostgreSQL	snowflake, VERTICA

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?

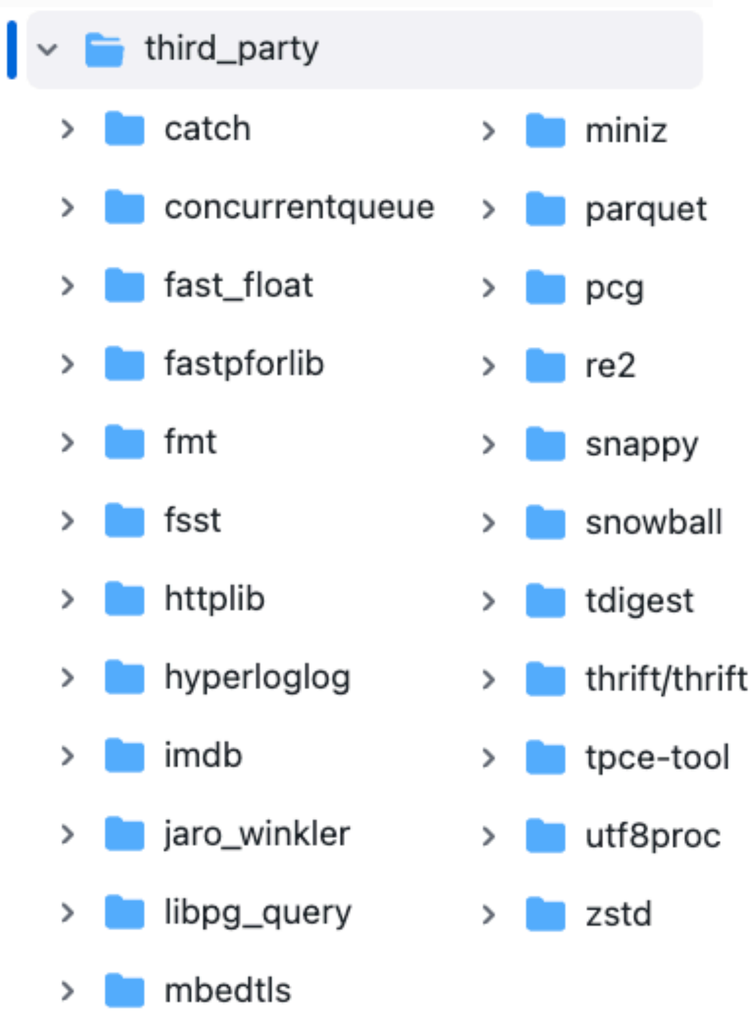
DuckDB has zero external dependencies

Dependencies are vendored in the codebase

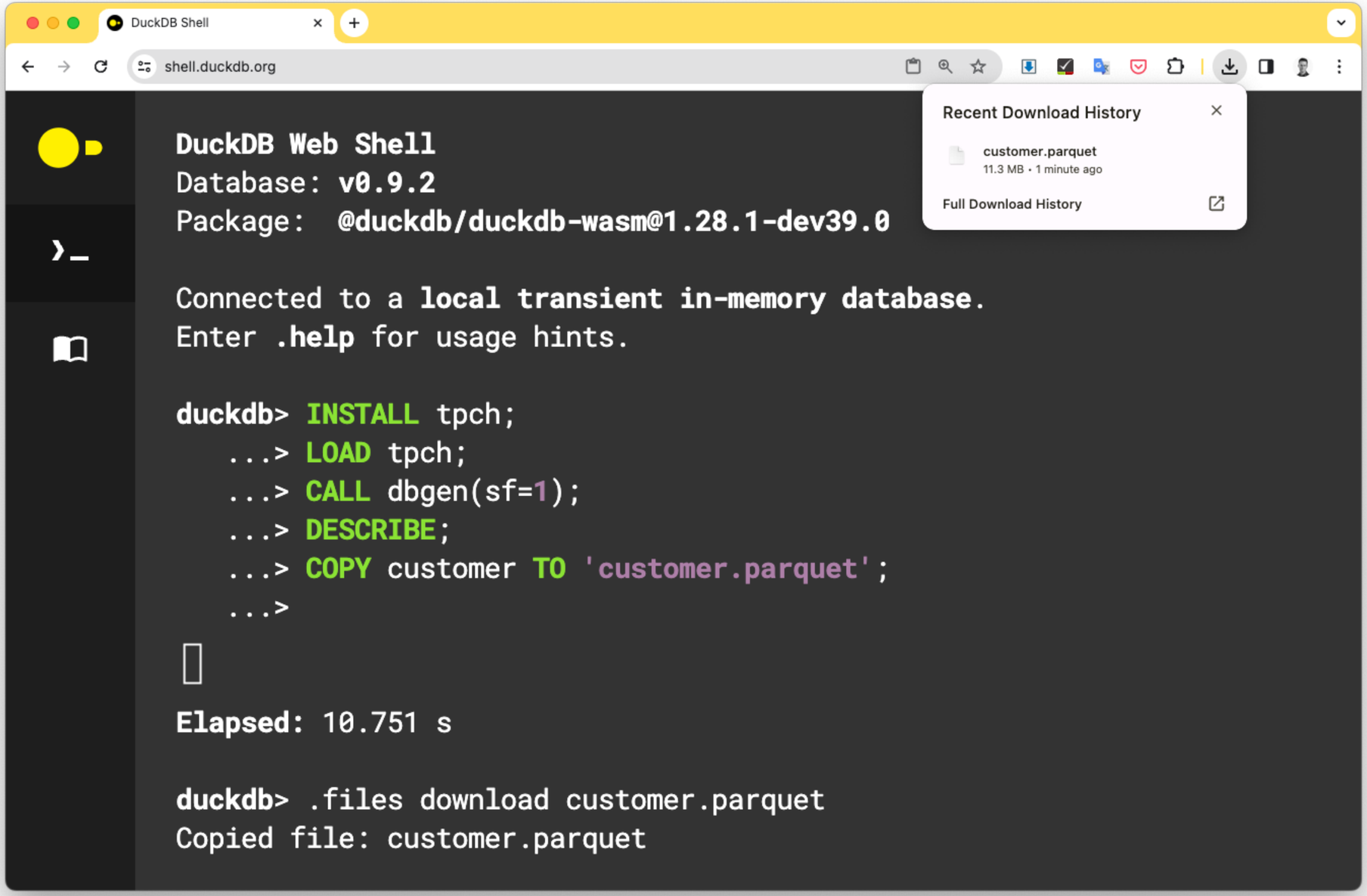
Pure C/C++ codebase

Portable anywhere with a C++11 compiler

Small binary packages



WebAssembly (Wasm)



Fast

CSV reader performance

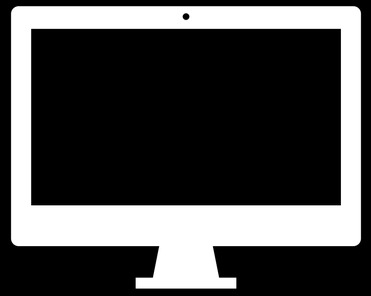
Test data: LDBC social network data set

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

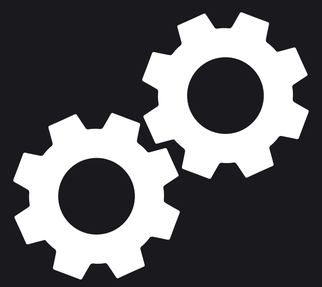
≈3x compression

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

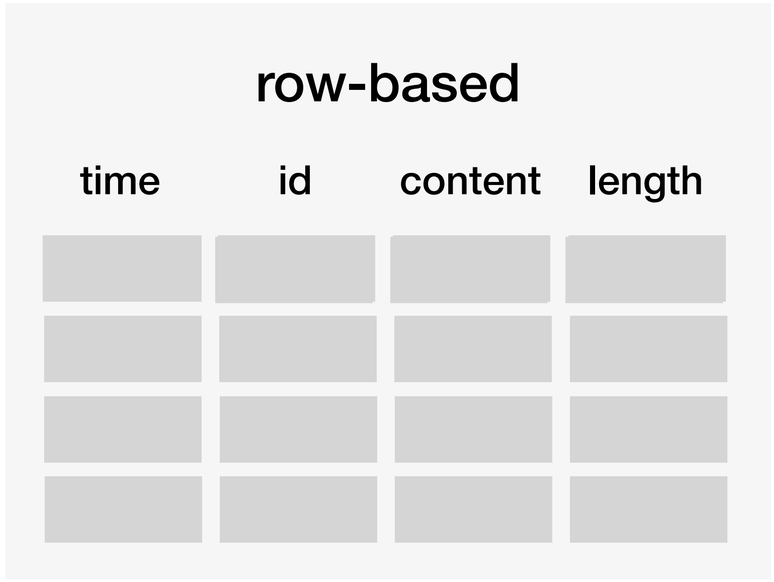


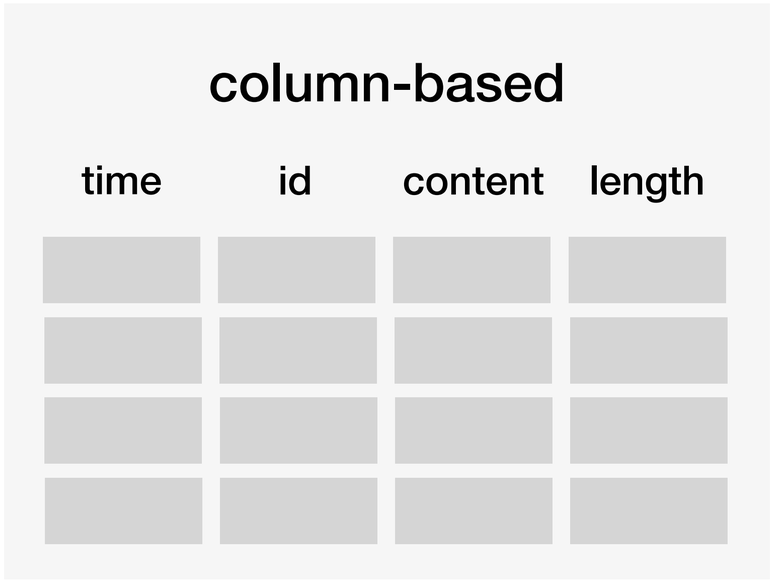
Demo

Internals

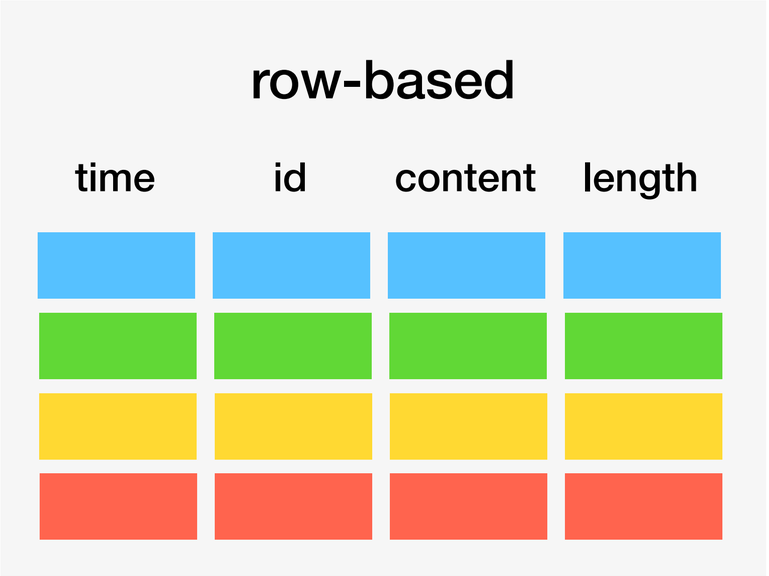


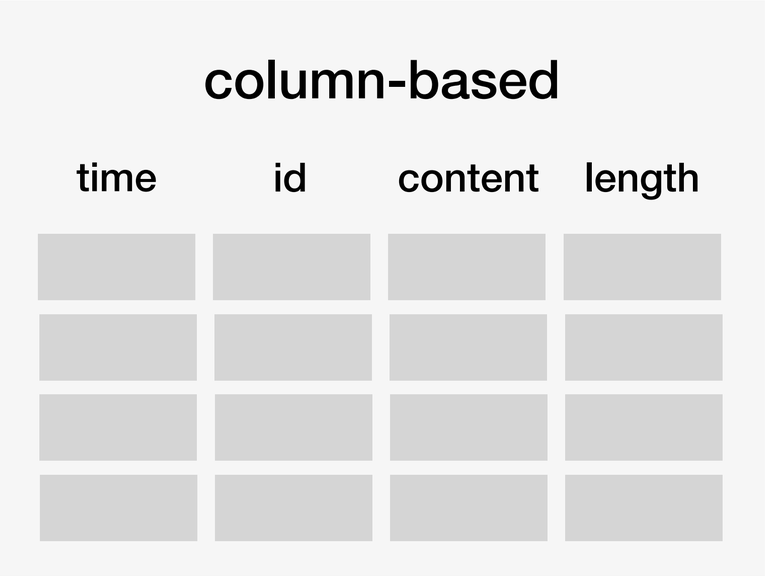
Storage


row-based
time
id
content
length

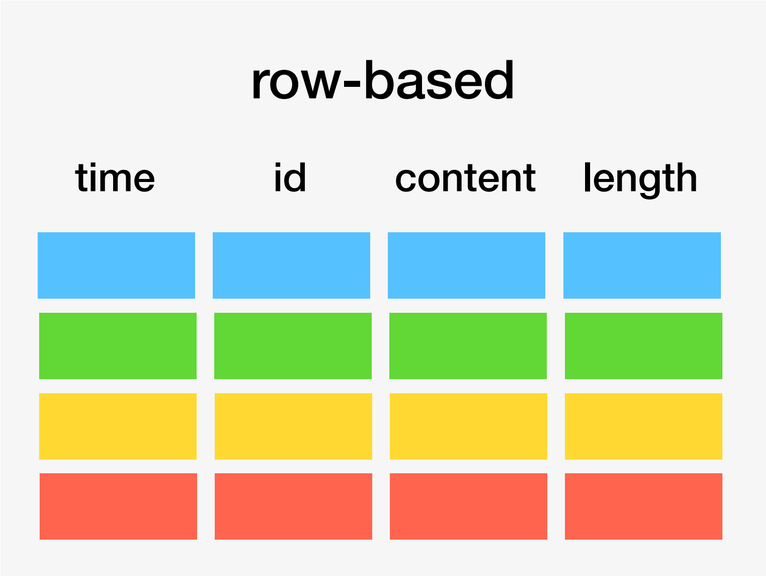

column-based
time
id
content
length

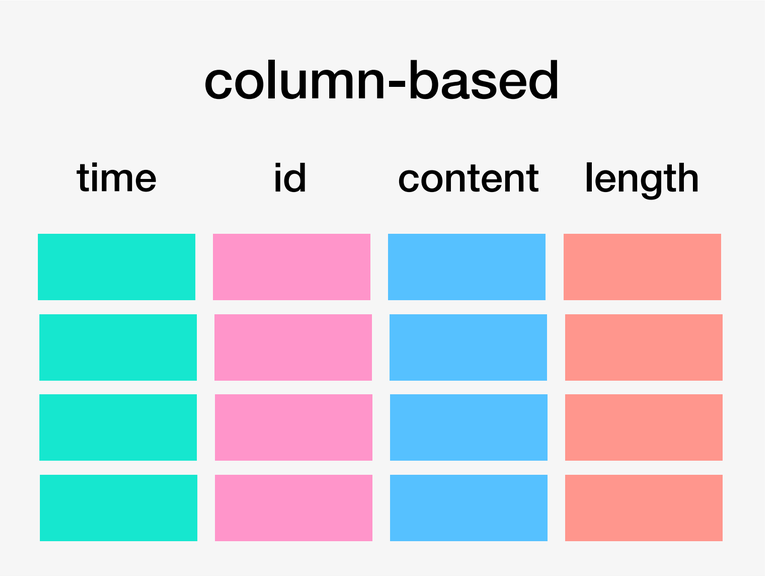
Storage


row-based
time
id
content
length

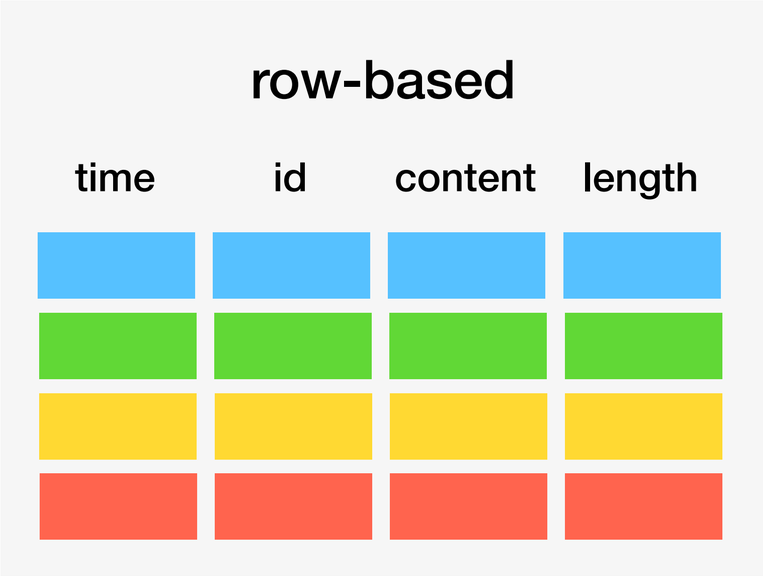

column-based
time
id
content
length

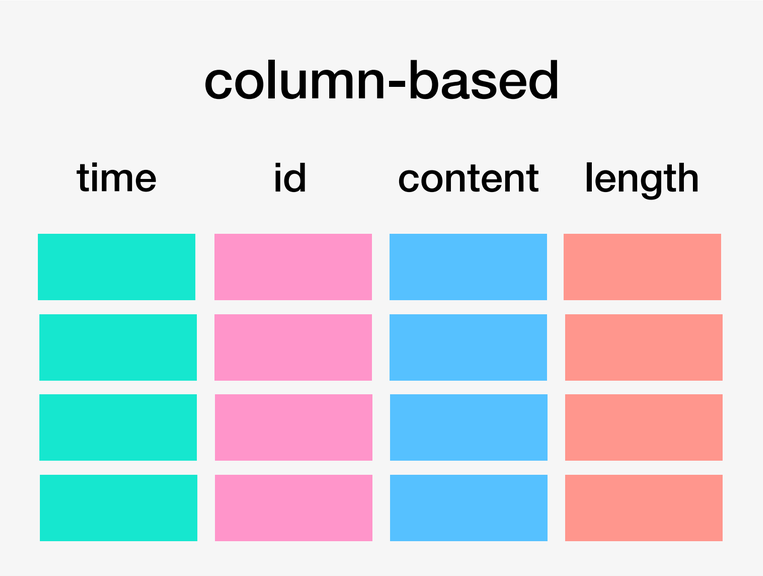
Storage

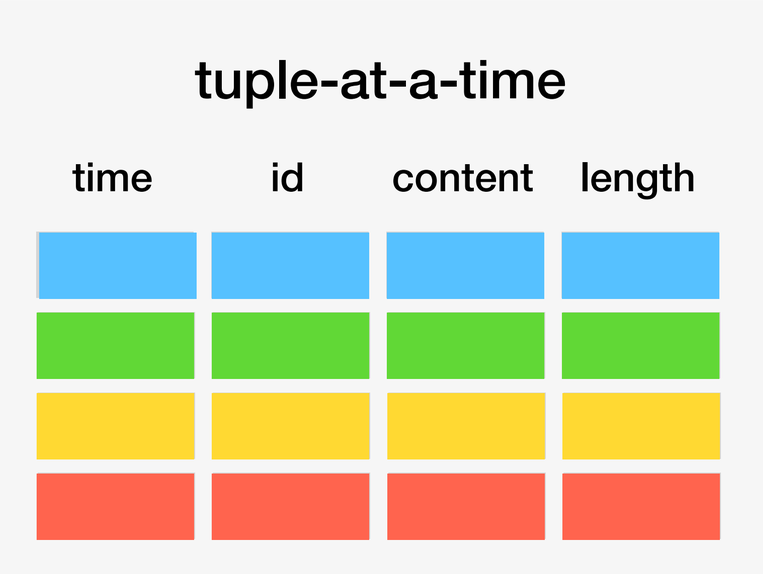

row-based
time
id
content
length

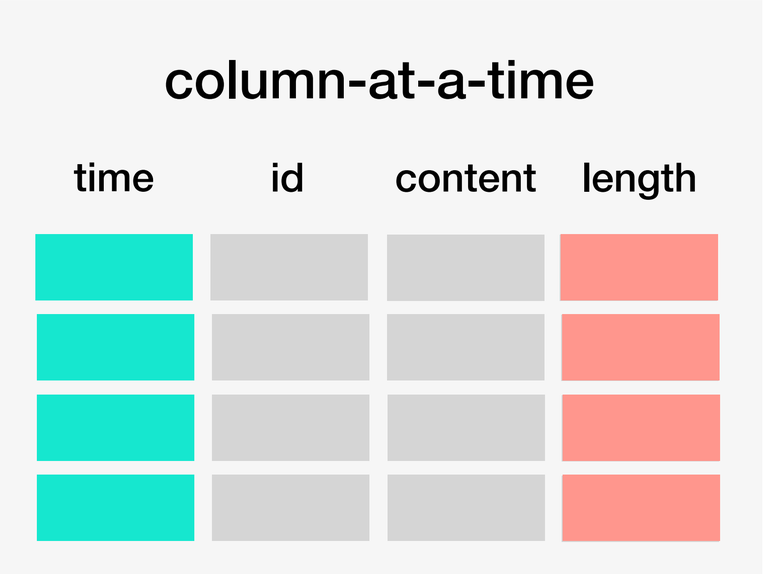

column-based
time
id
content
length

Execution

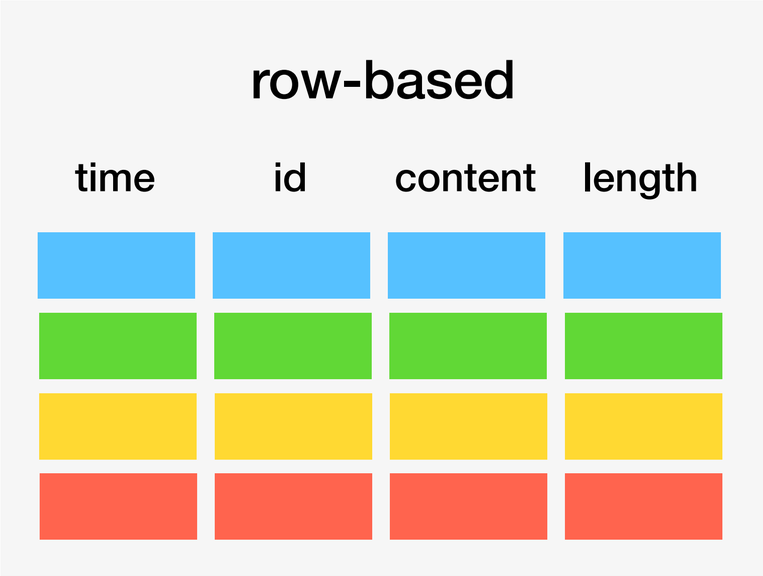

row-based
time
id
content
length

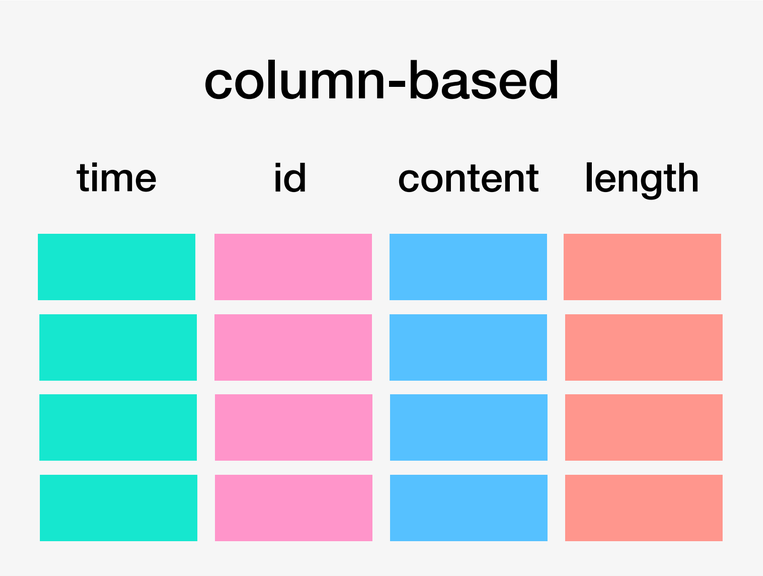

column-based
time
id
content
length

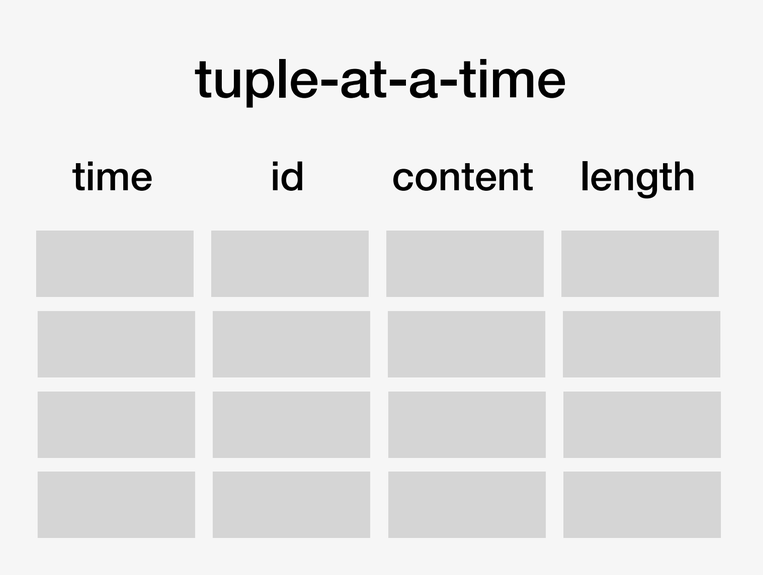

tuple-at-a-time
time
id
content
length

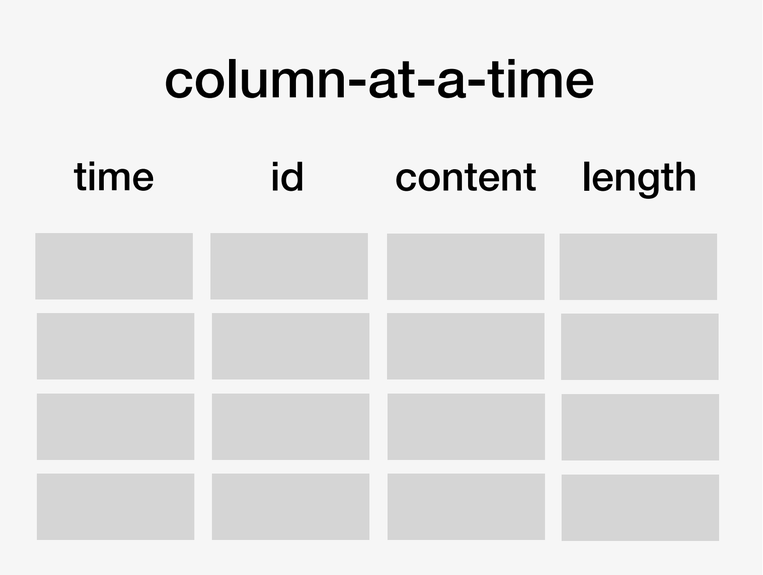

column-at-a-time
time
id
content
length

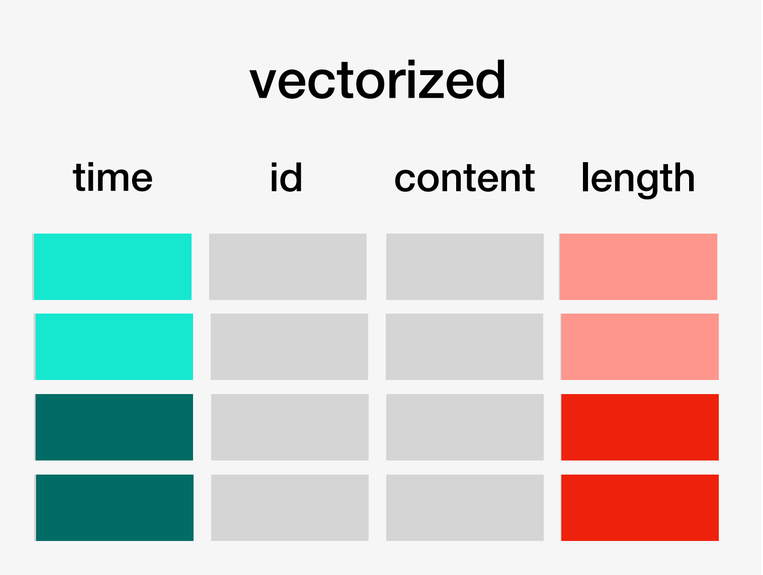
Execution


row-based
time
id
content
length

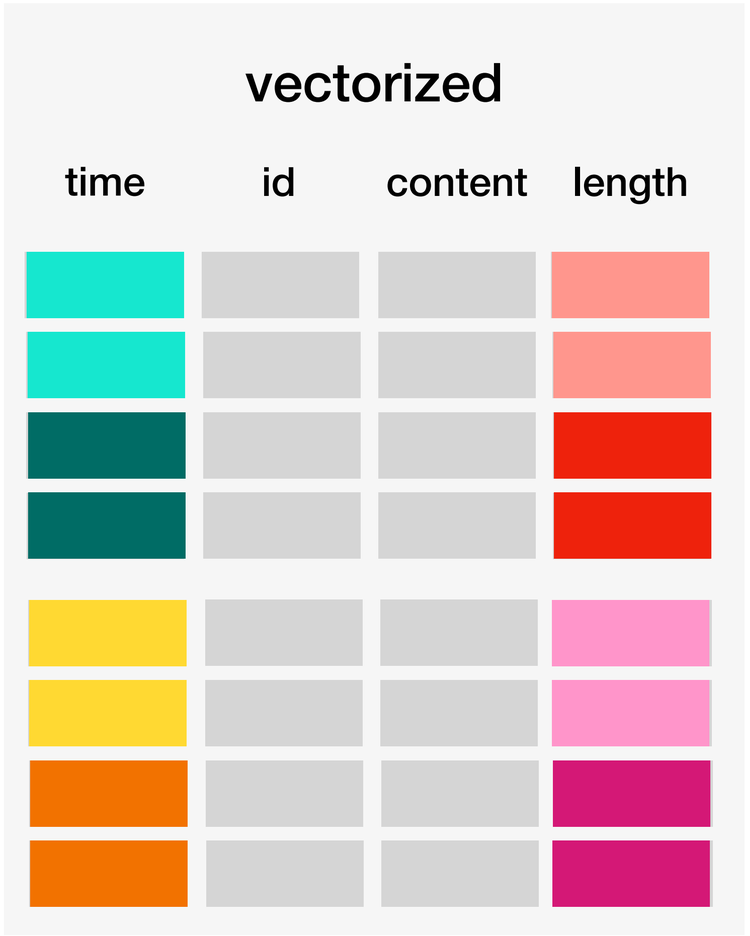

column-based
time
id
content
length


tuple-at-a-time
time
id
content
length


column-at-a-time
time
id
content
length


vectorized
time
id
content
length

Vectorized execution



2 row groups

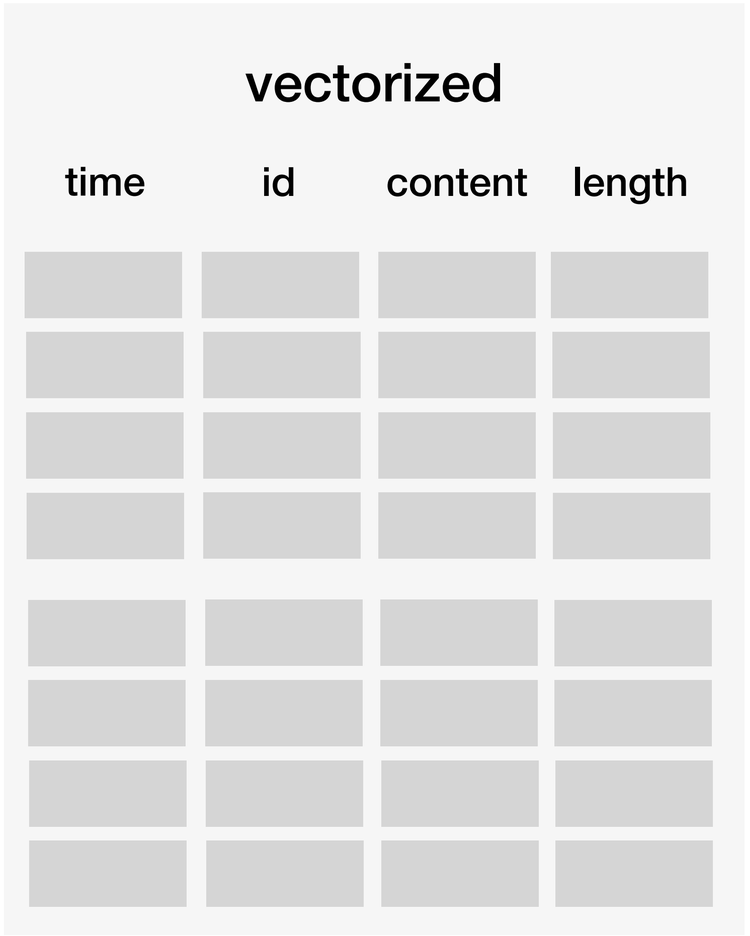
thread 1

thread 2

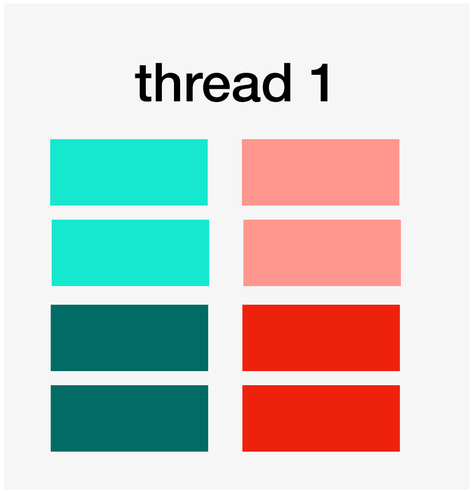
L1 cache

L1 cache

Vectorized execution



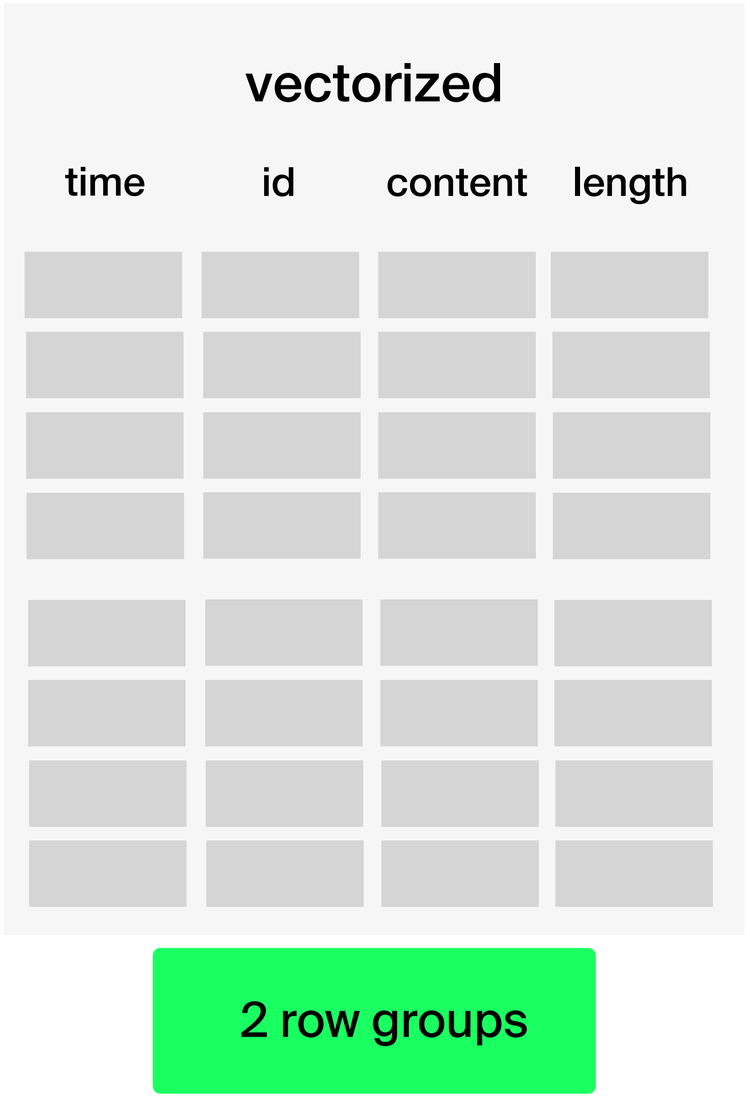
2 row groups

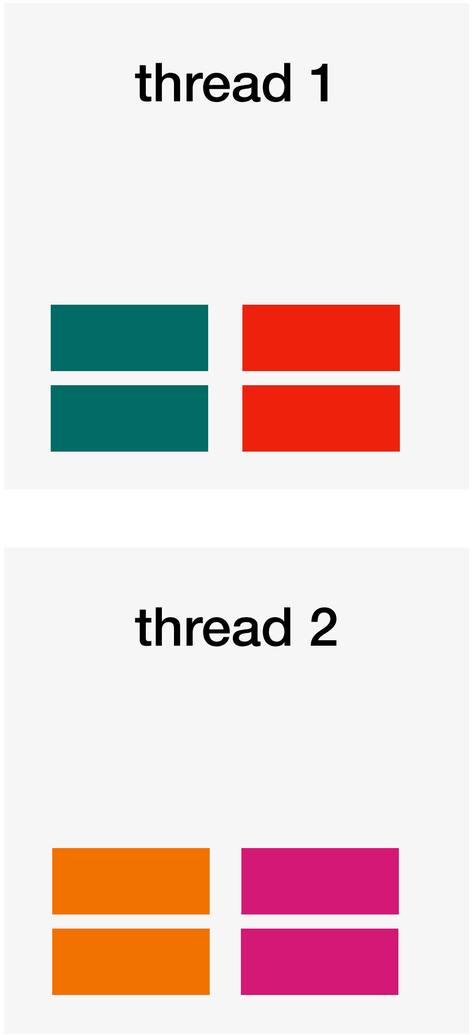


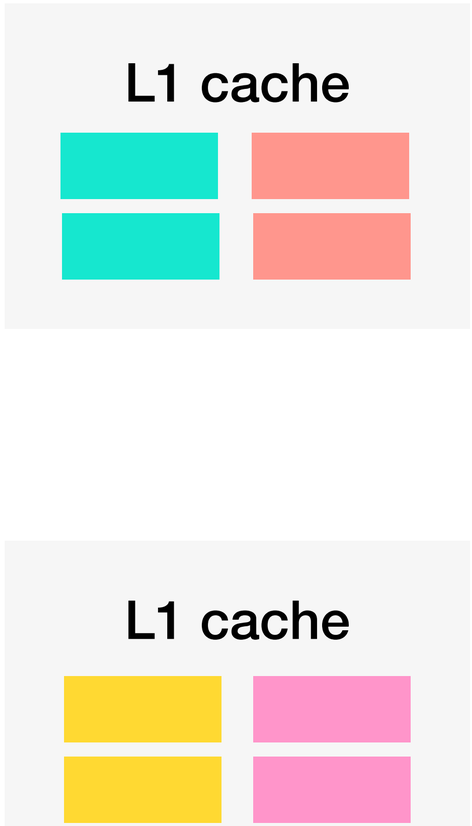
L1 cache

L1 cache

Vectorized execution







Vectorized execution

vectorized

time	id	content	length

2 row groups

thread 1

thread 2

L1 cache

L1 cache

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

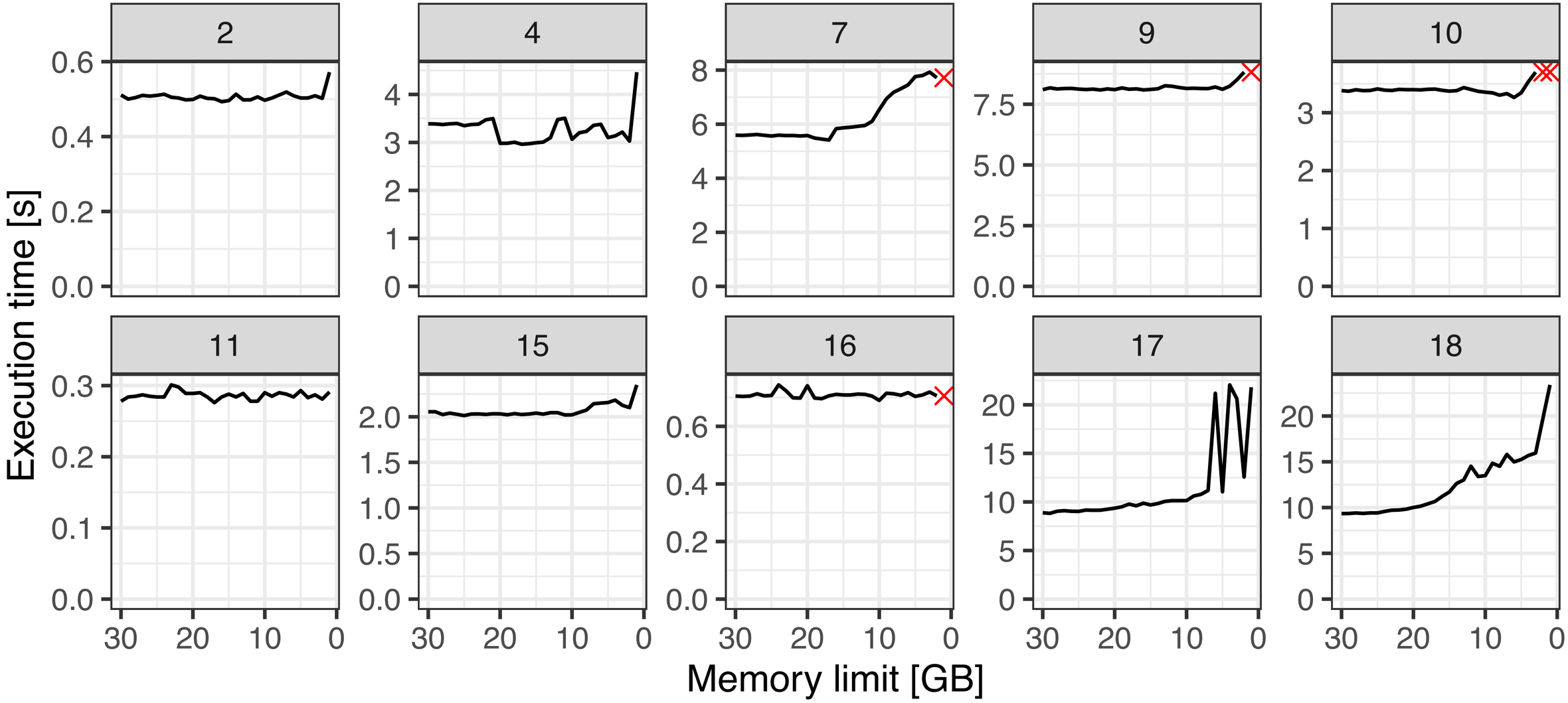
min	max
Dec 10	Dec 14

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

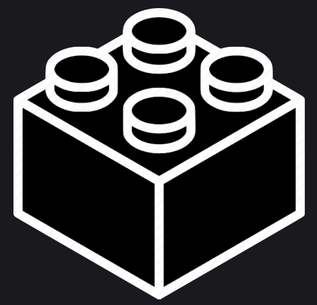
min	max
63	109

min	max
8	95

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



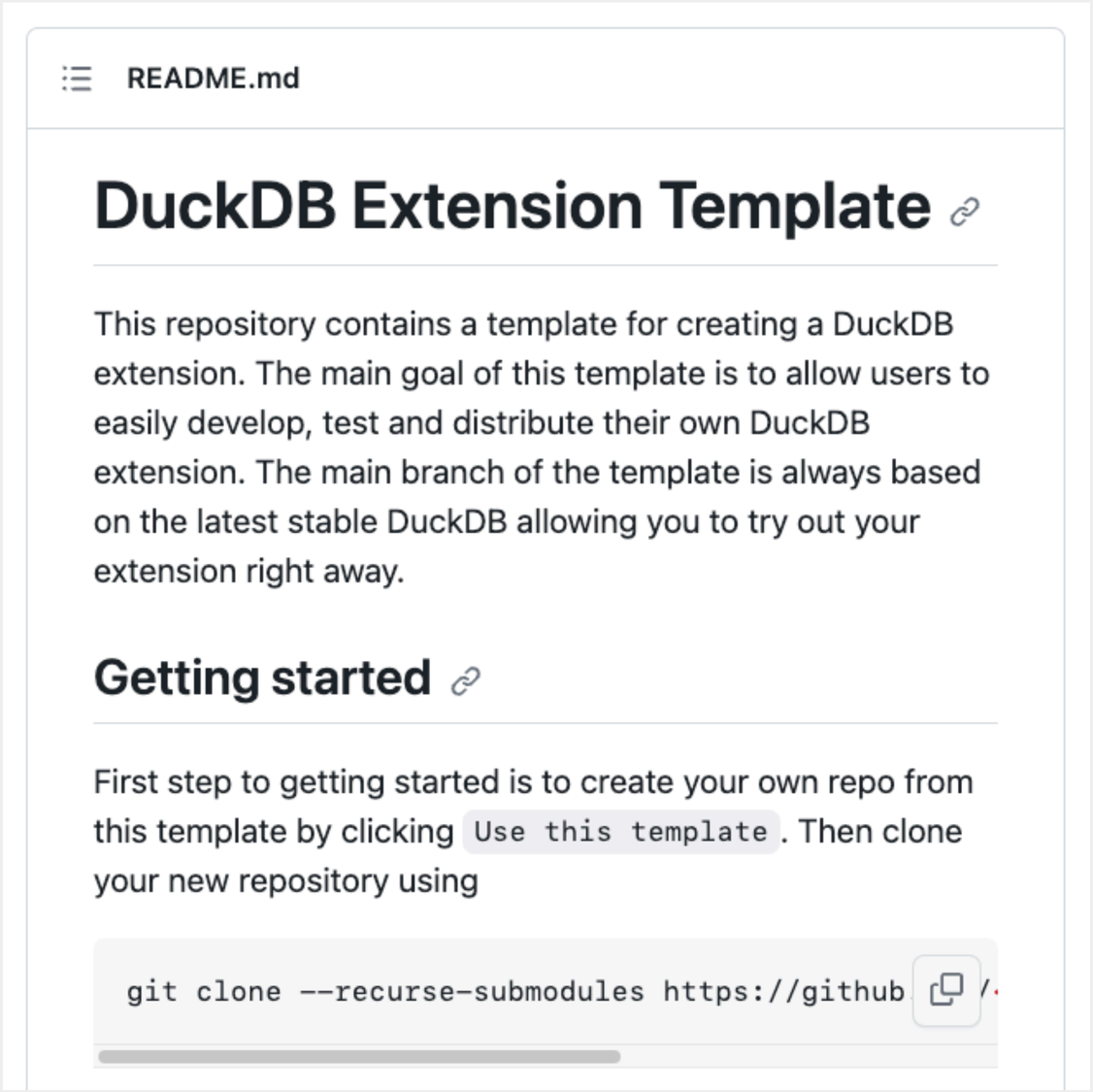
Extensions



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



Parquet + httpfs extensions to query stock data

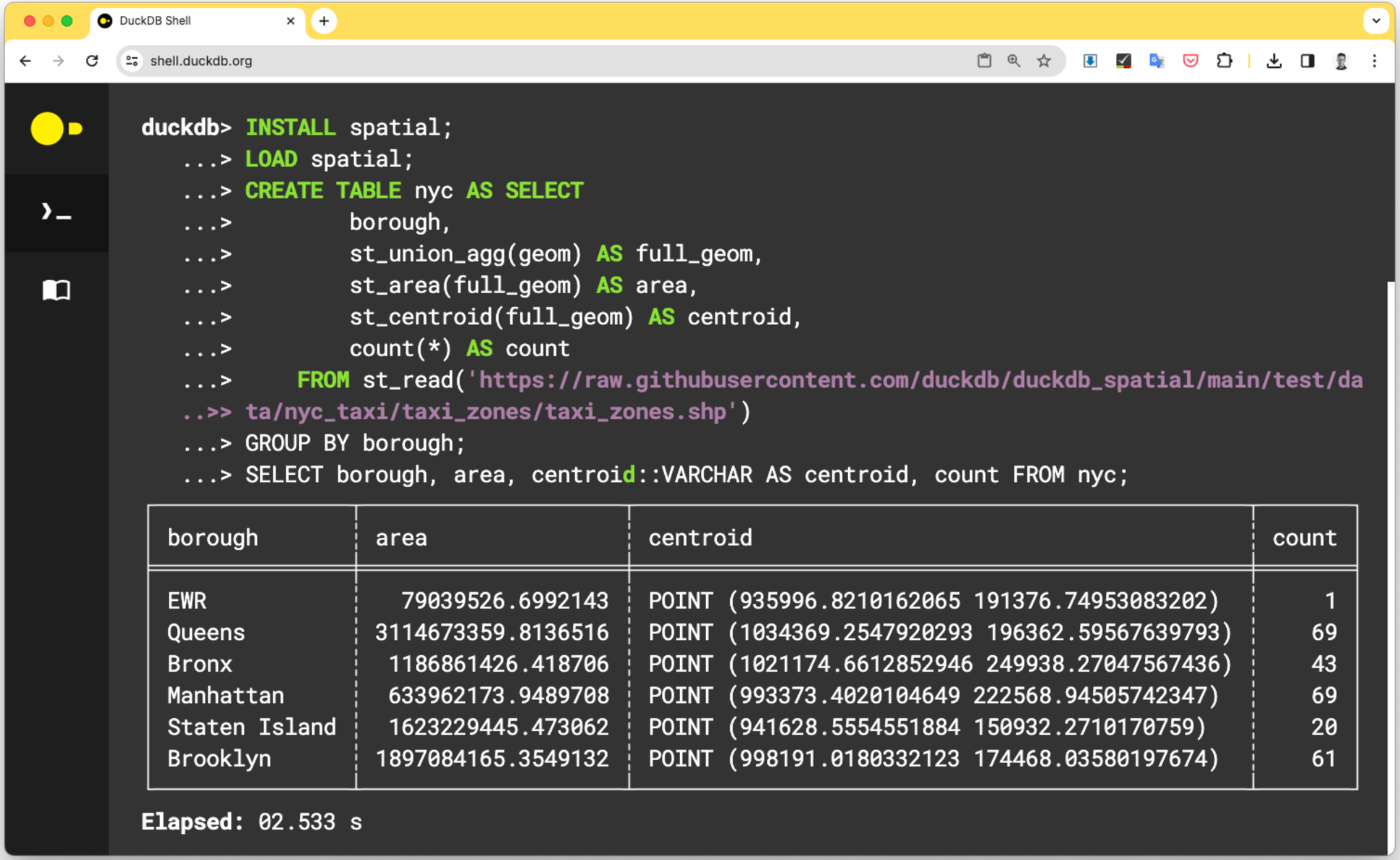
```
SELECT avg(price)
FROM 'https://duckdb.org/data/prices.parquet'
WHERE ticker = 'MSFT';
```

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



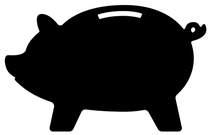

```
duckdb> INSTALL spatial;
   ...> LOAD spatial;
   ...> CREATE TABLE nyc AS SELECT
   ...>         borough,
   ...>         st_union_agg(geom) AS full_geom,
   ...>         st_area(full_geom) AS area,
   ...>         st_centroid(full_geom) AS centroid,
   ...>         count(*) AS count
   ...>     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;
```

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

```
Elapsed: 02.533 s
```

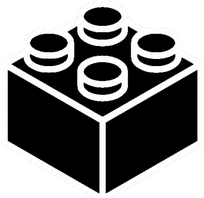
Use cases

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

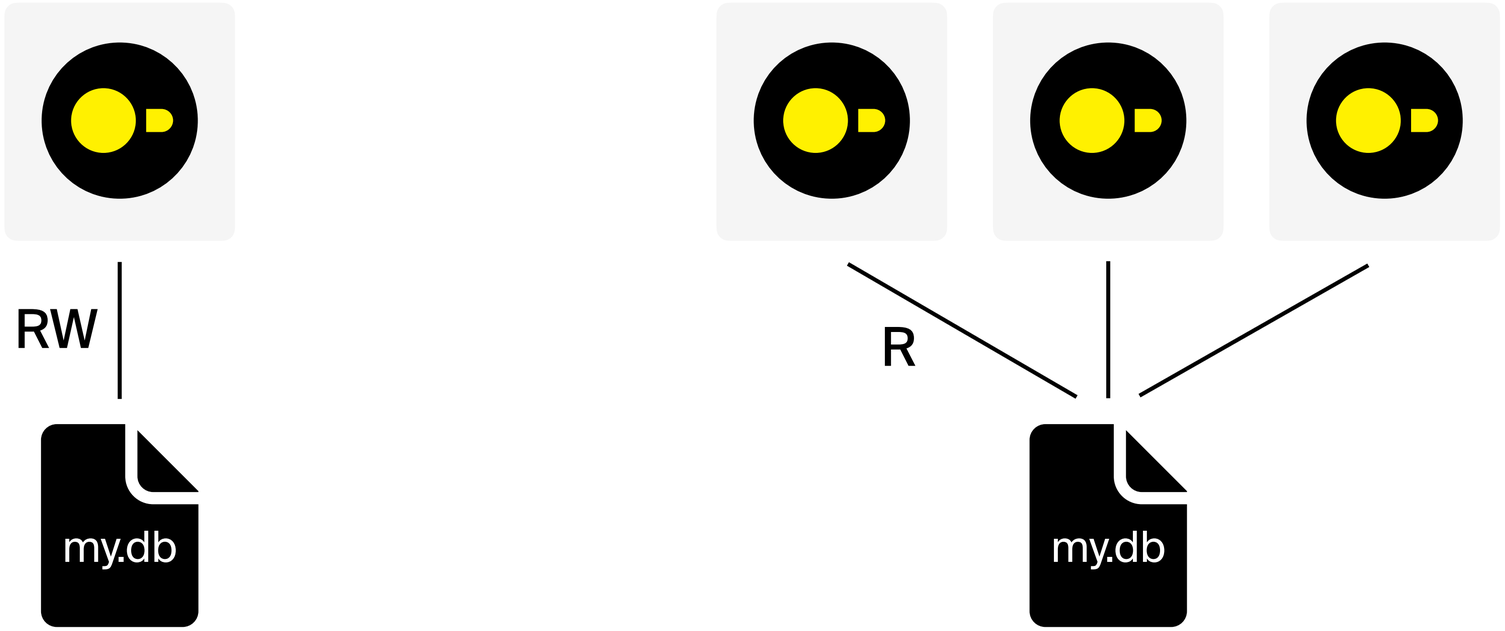
Education:

- 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

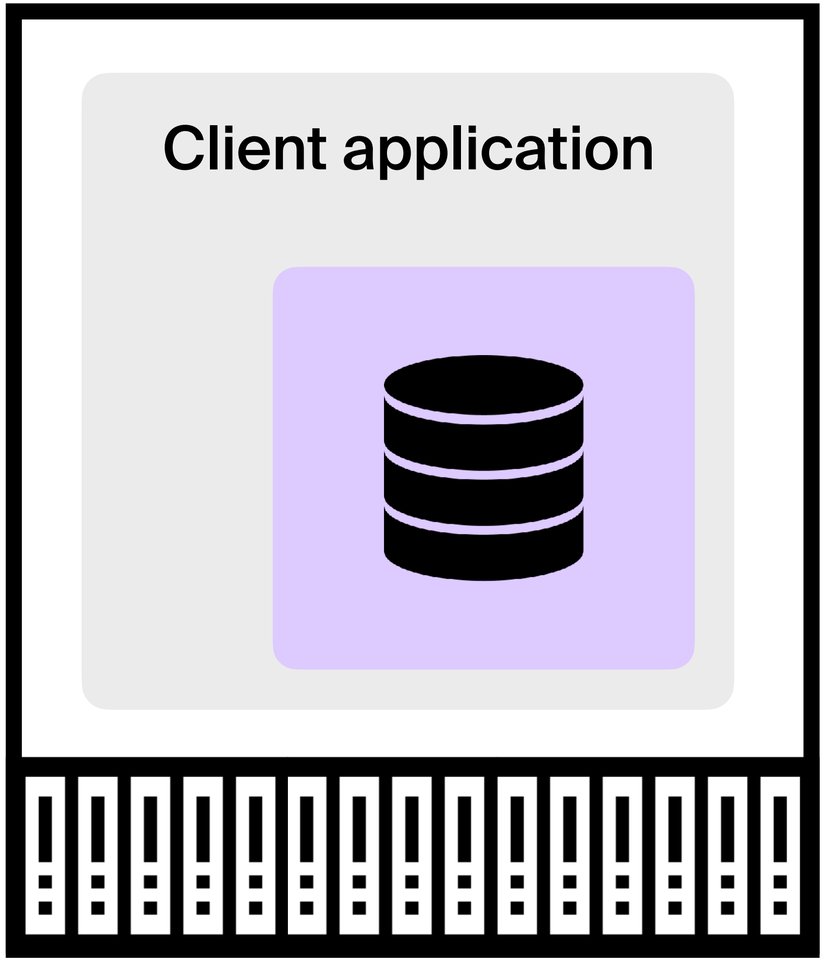
DuckDB only supports **single-node** execution

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

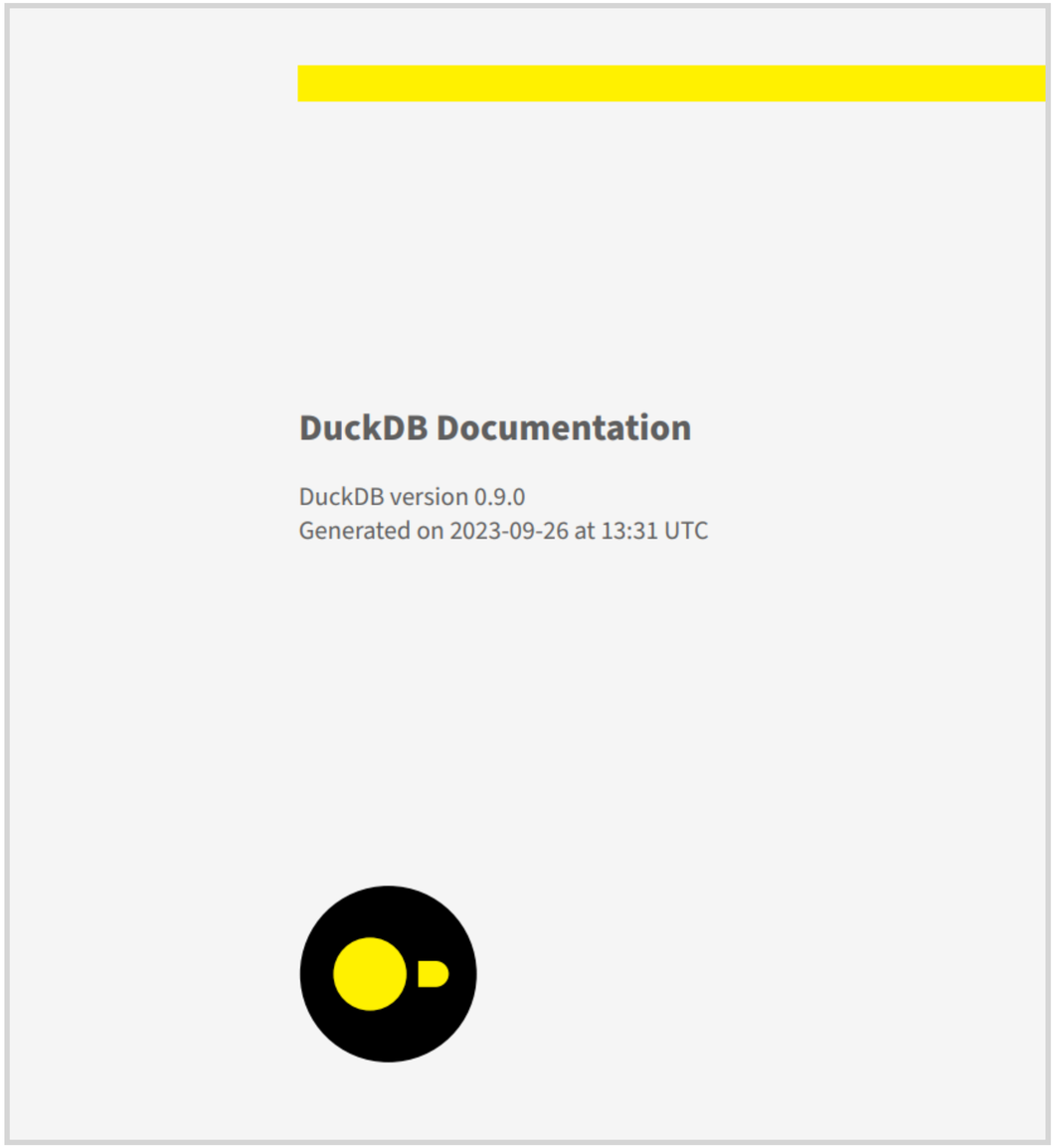
Classic open-source project

Full-fledged CLI client

Works offline

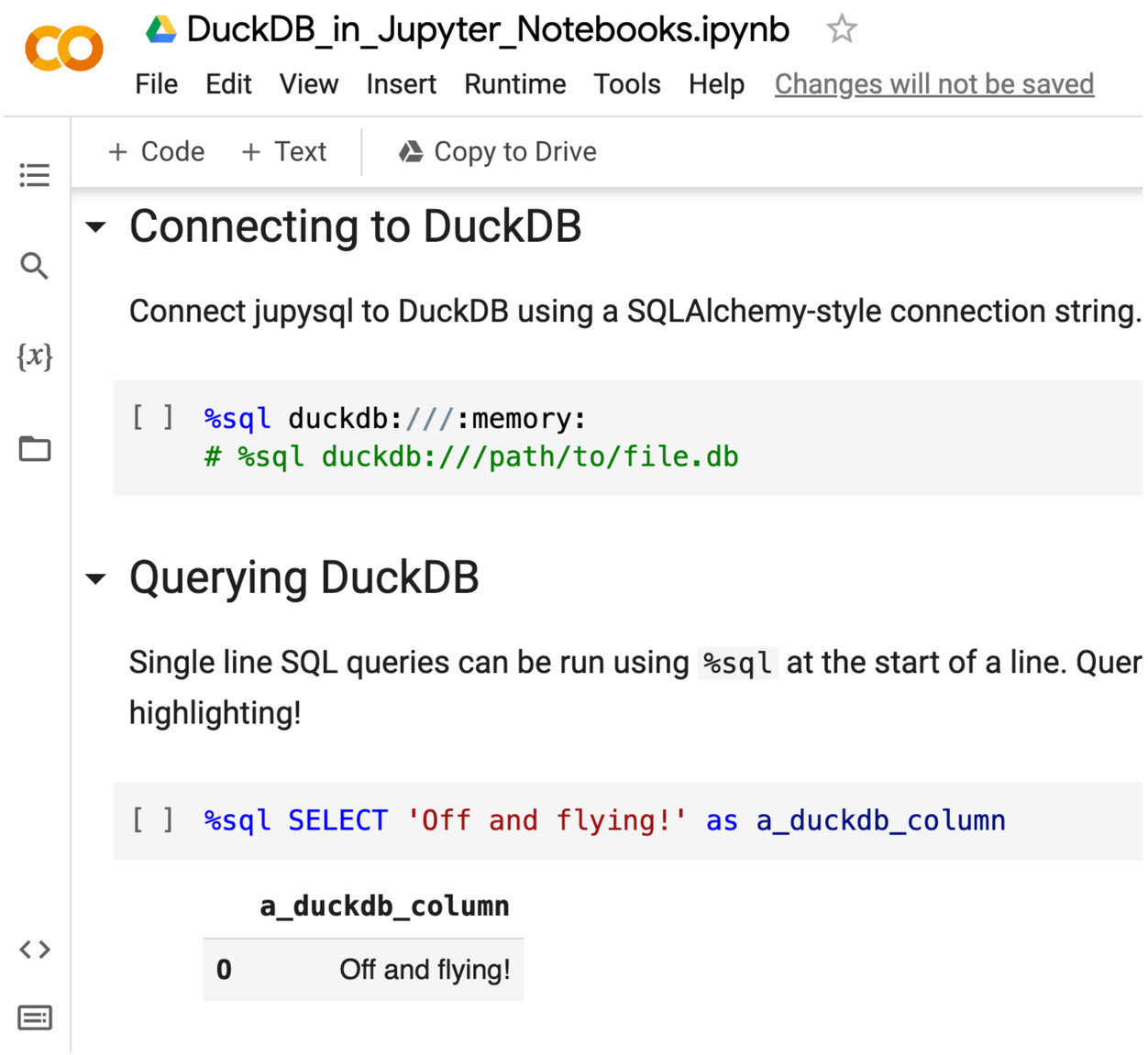
No vendor lock-in

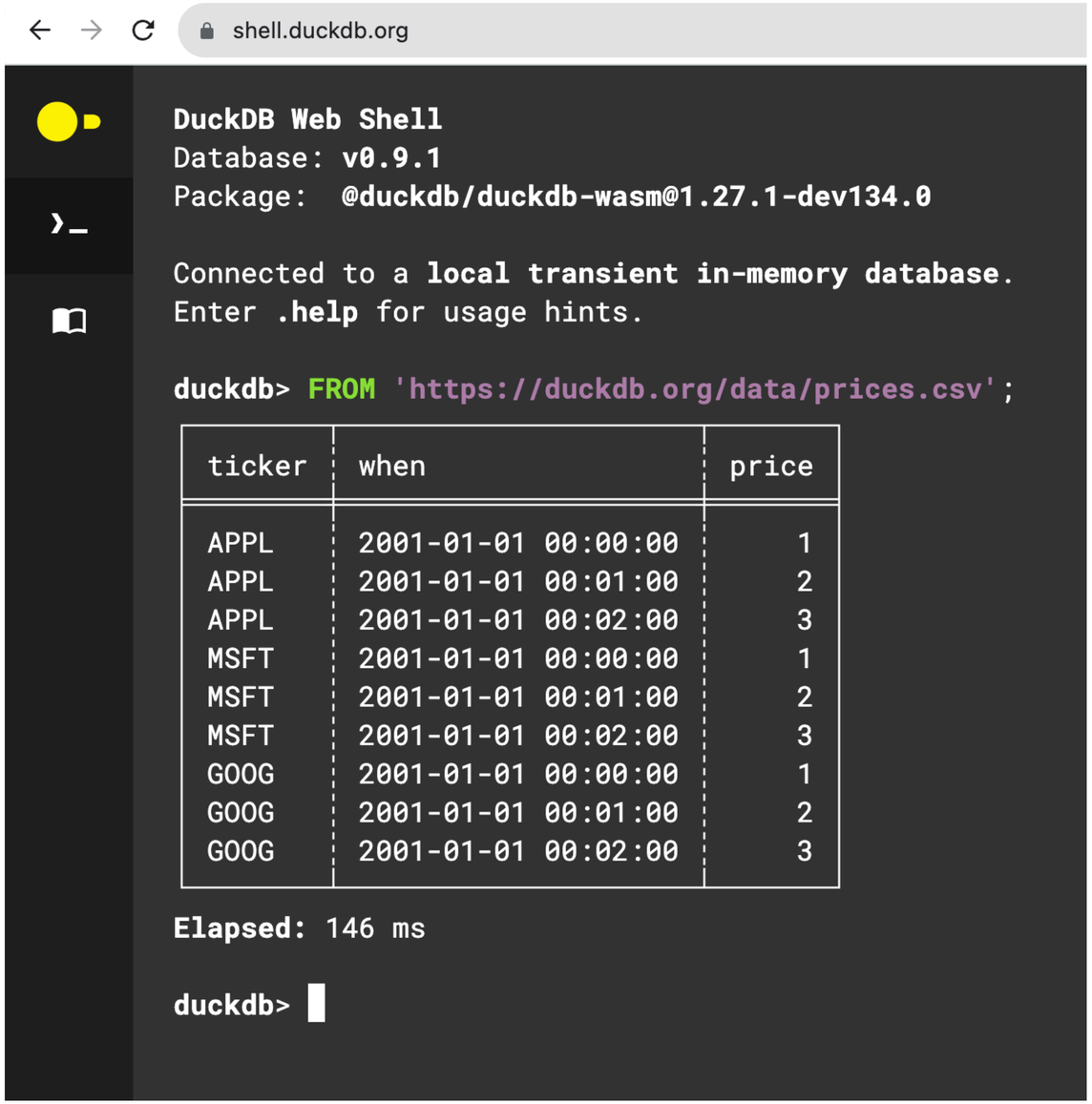
```
EXPORT DATABASE 'my_db' (FORMAT CSV);
EXPORT DATABASE 'my_db' (FORMAT PARQUET);
```



Give DuckDB a spin!

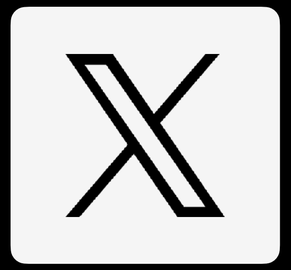
Google Colab, shell.duckdb.org





Stay in touch

discord.duckdb.org



@duckdb

duckdb.org