DuckDB

Harnessing in-process analytics for data science and beyond

Gábor Szárnyas
Developer Relations Advocate

DuckDB Labs

About me

Gábor Szárnyas

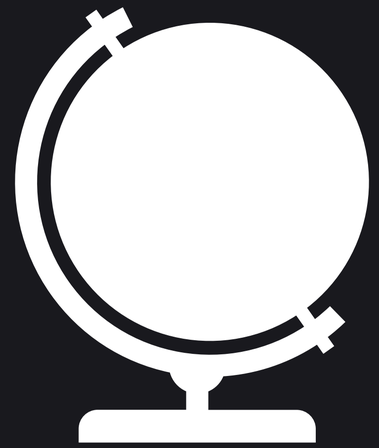
- 2014–2023: PhD + postdoc
- Research: benchmarks, graph processing

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.

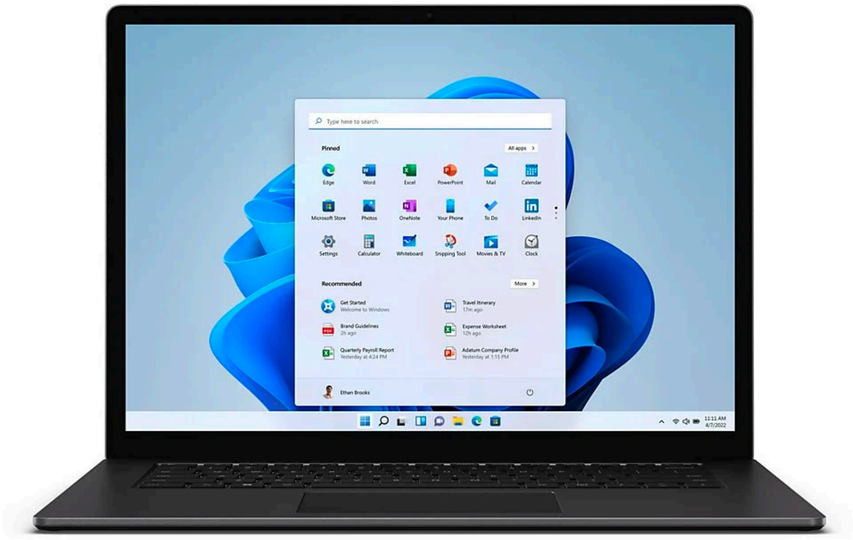
6:15 PM · Oct 31, 2023

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

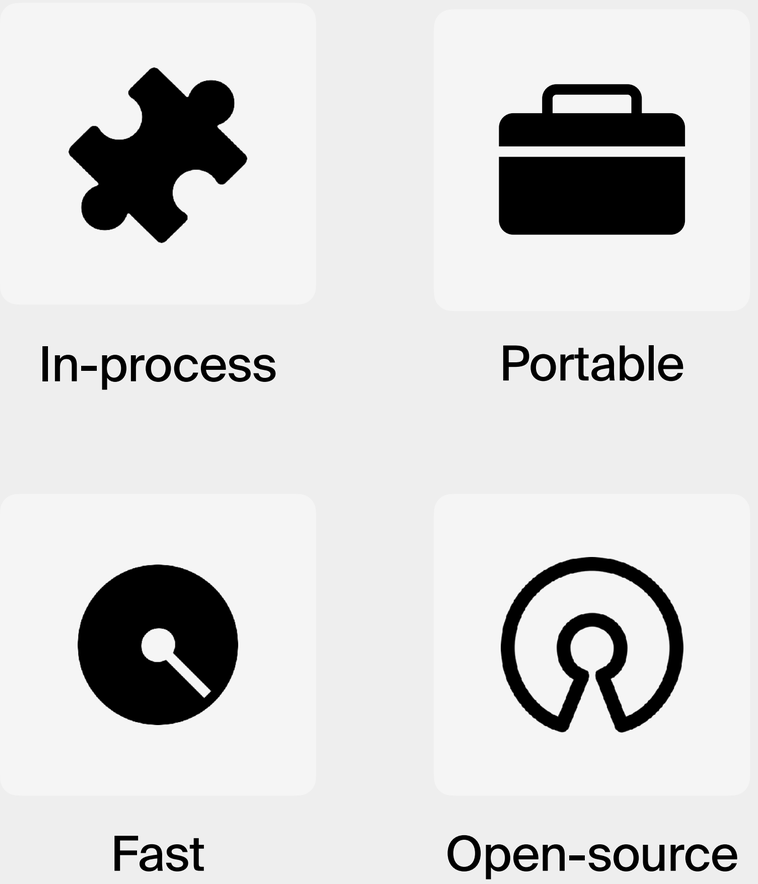
An analytical SQL database

Built to be portable and fast

Developed since 2018

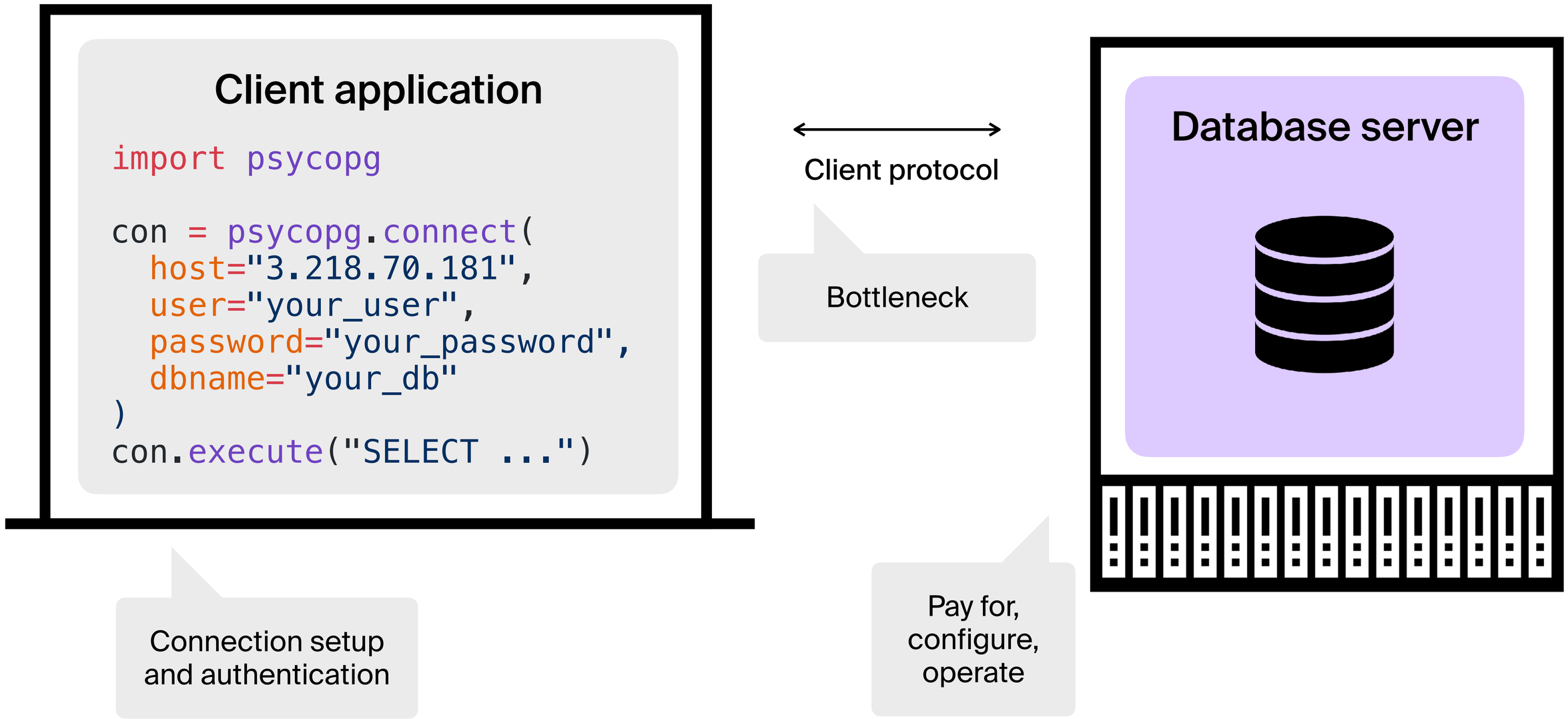
Written in C++11

Open-source under the MIT license

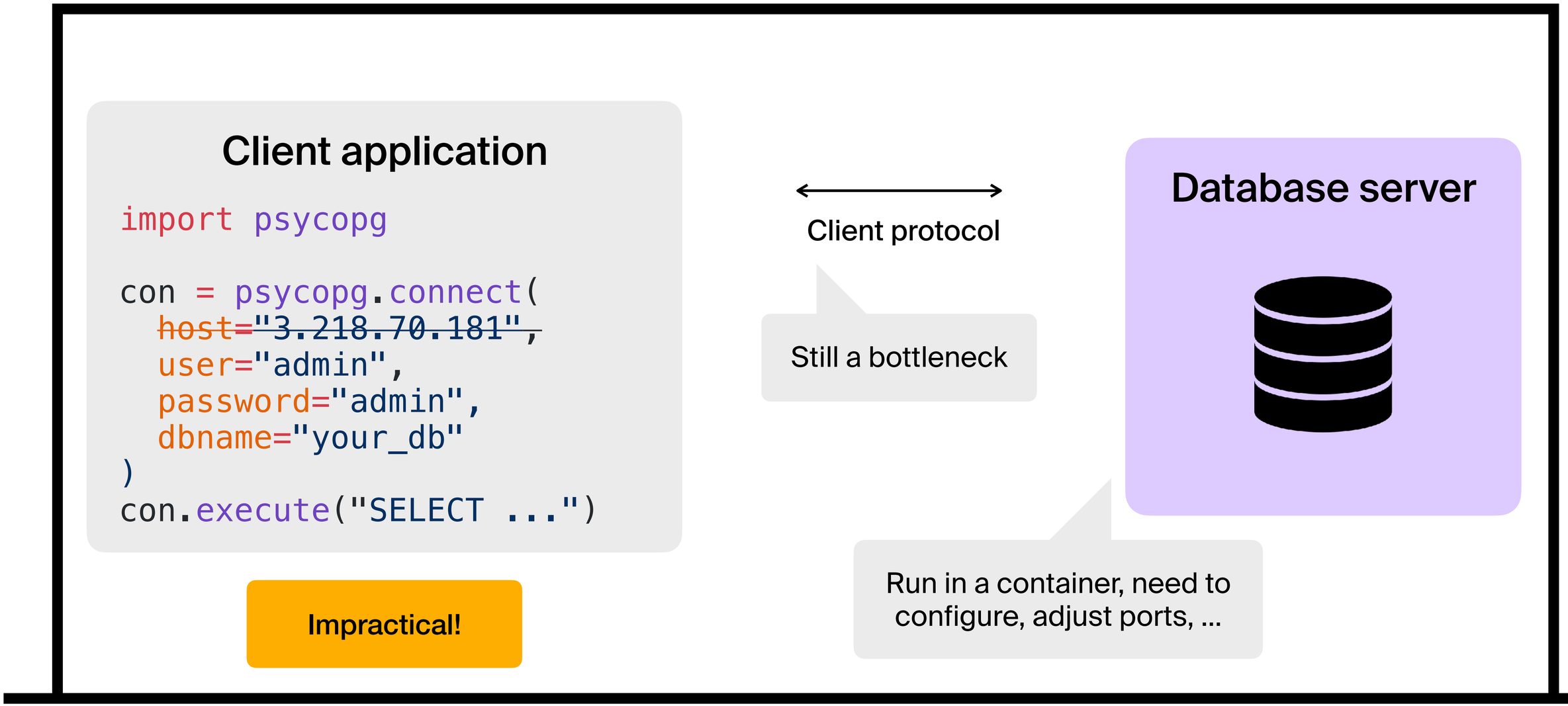


Deployment model

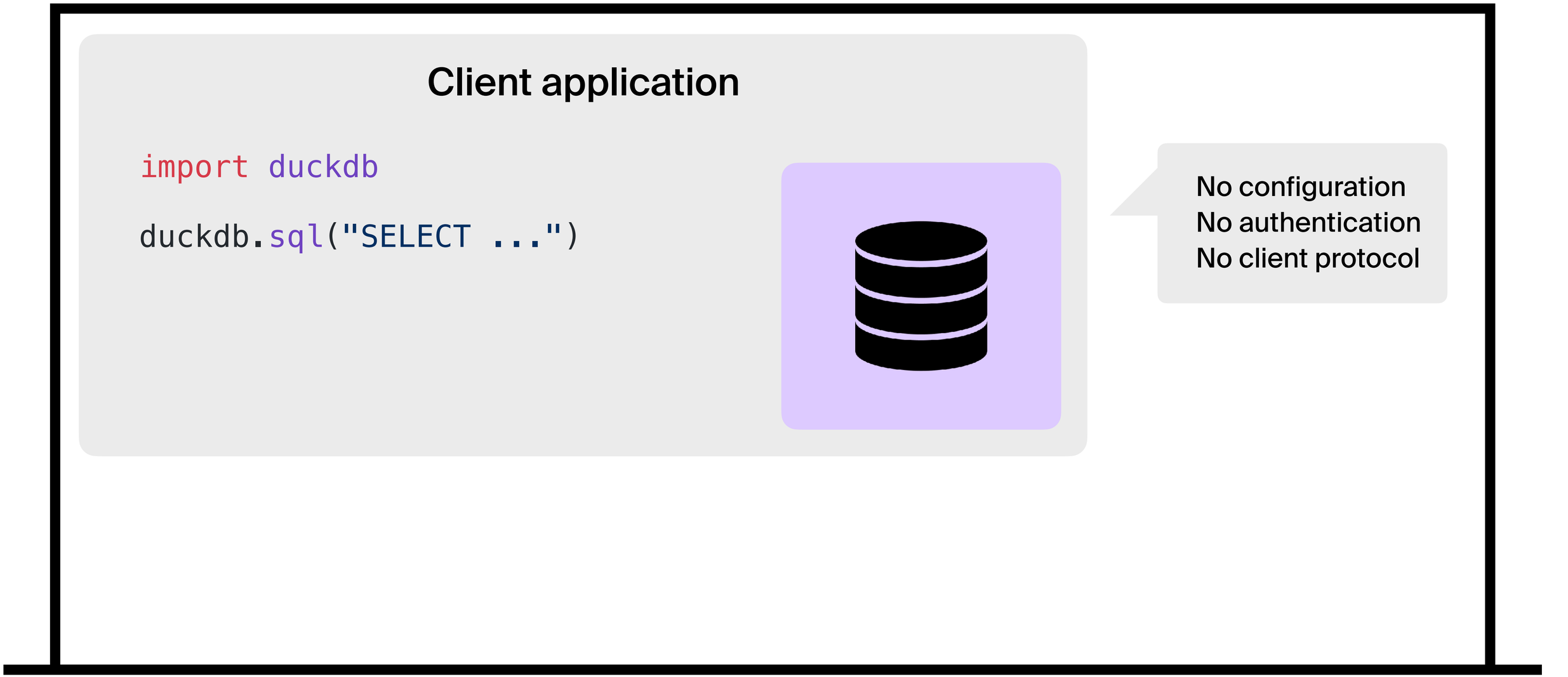
Client–server setup



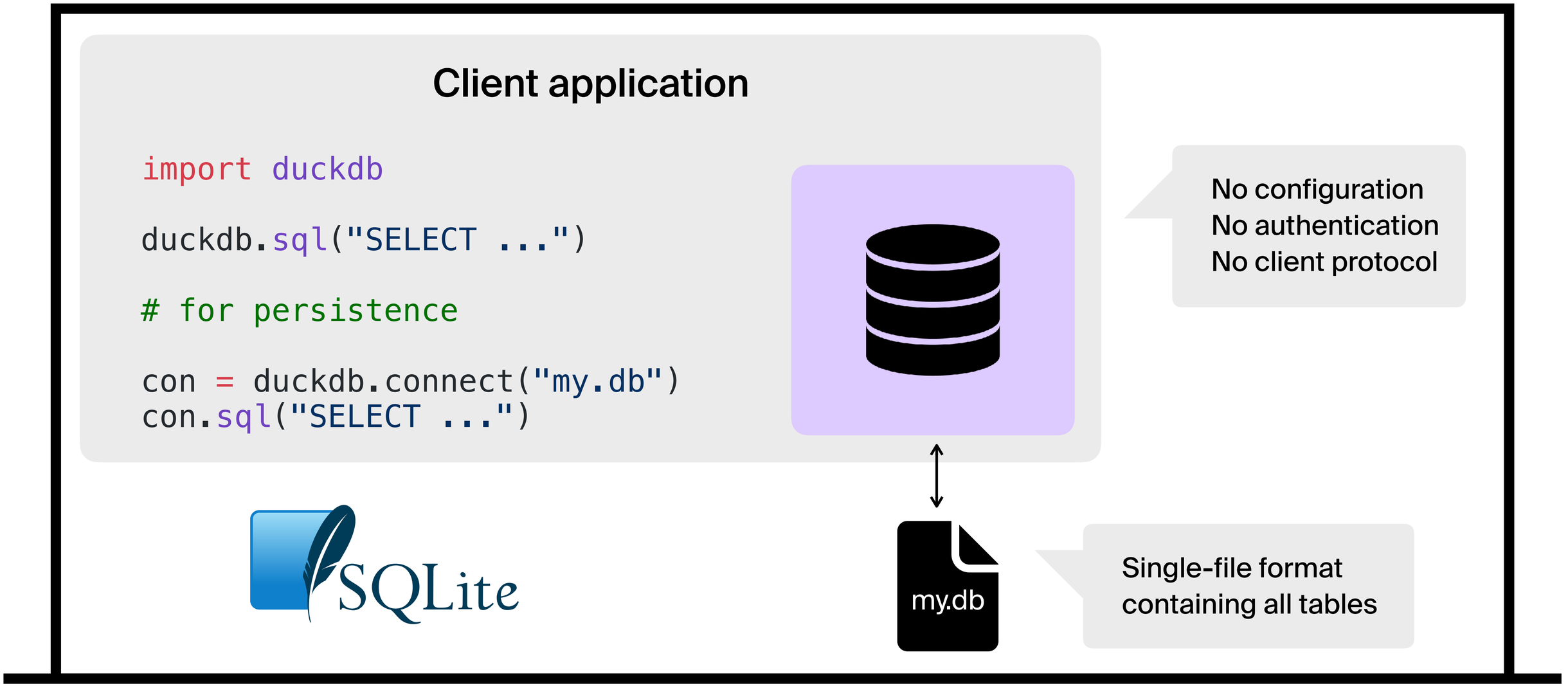
Client–server setup



In-process setup



In-process setup



Database systems

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

Portable

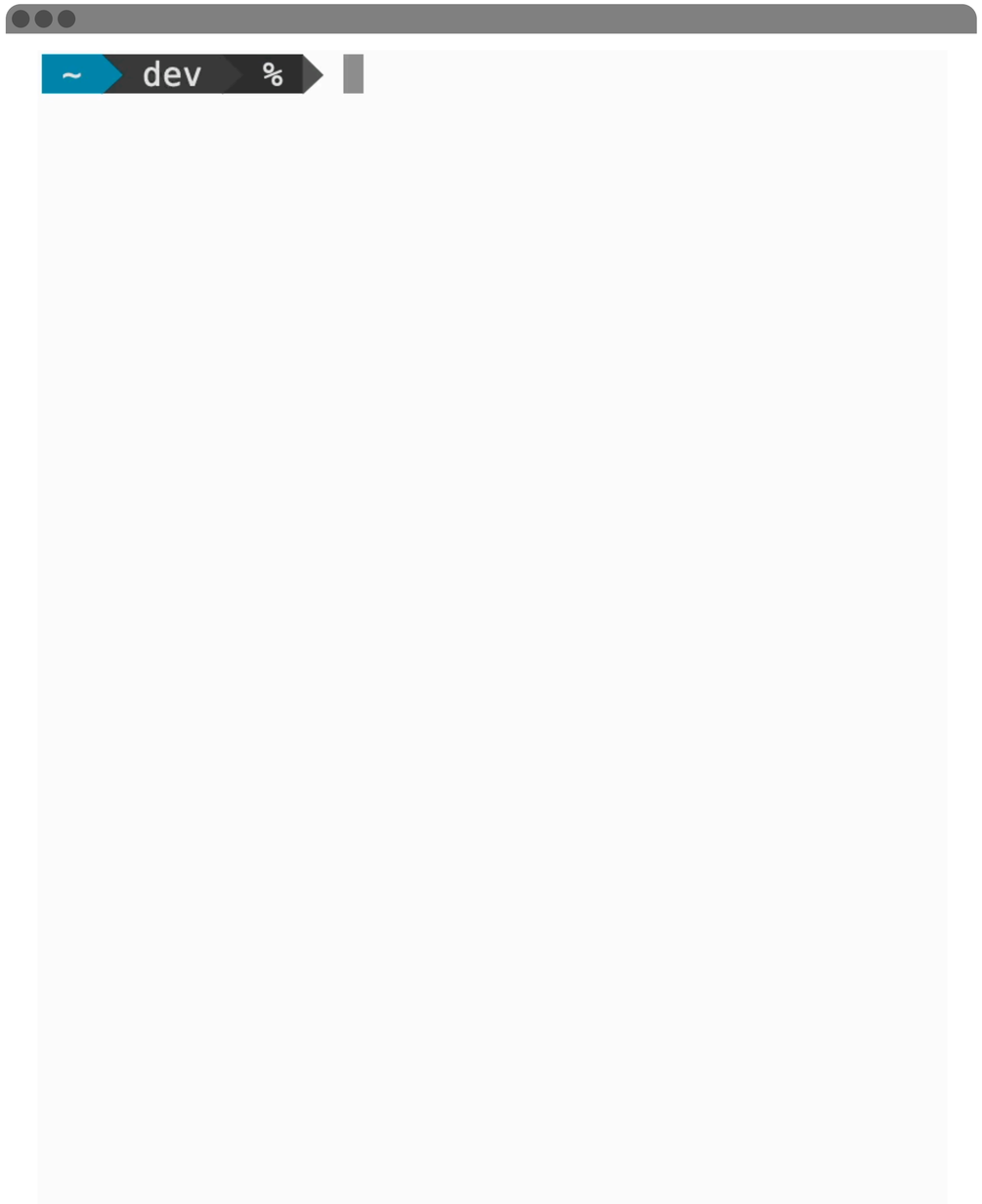
Installing DuckDB

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

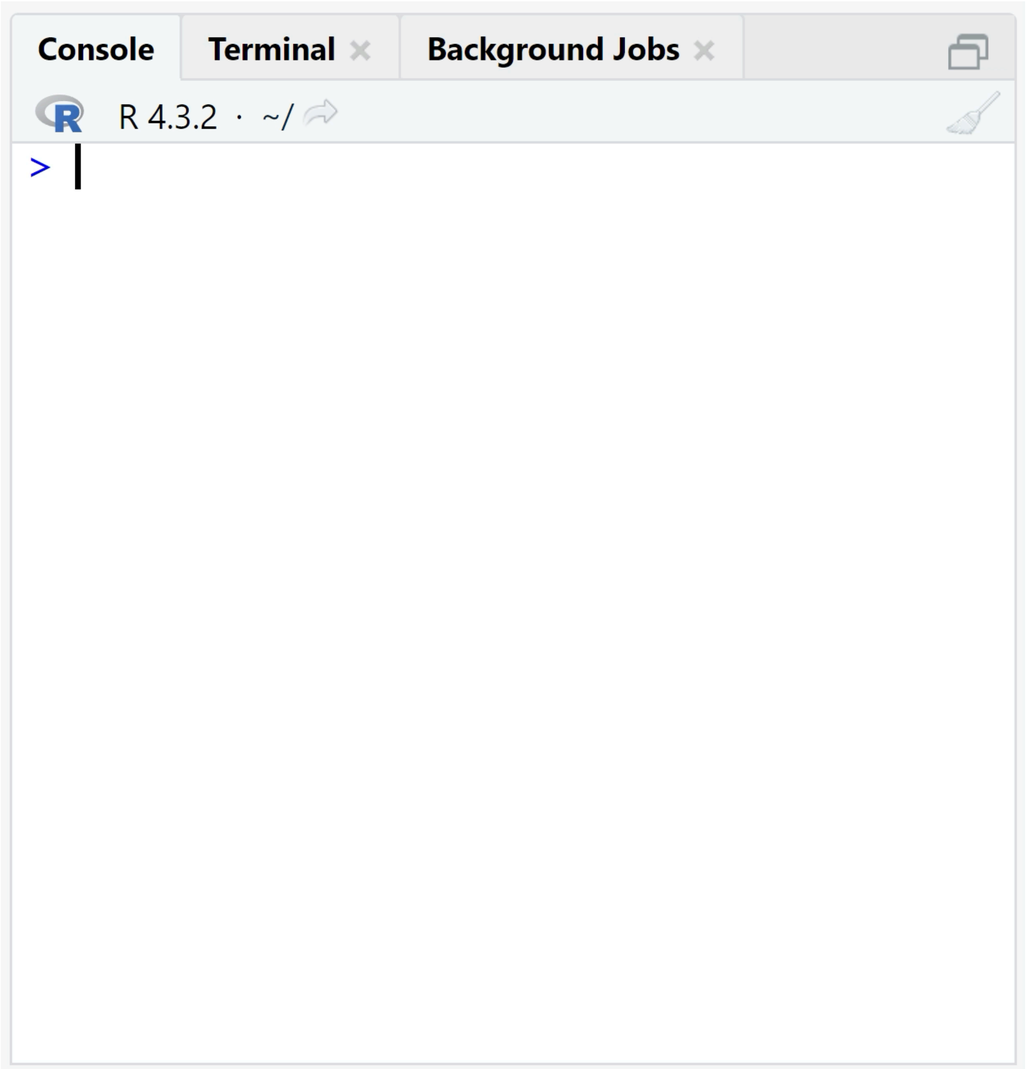
This includes:

- Typing the commands
- Downloading the packages from the internet
- Launching DuckDB

macOS: Python package



Windows: R package



...and more

```
pip install duckdb
```

```
npm install duckdb
```

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

```
org.duckdb:duckdb_jdbc
```

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

```
cargo add duckdb
```

Why is installation so fast?

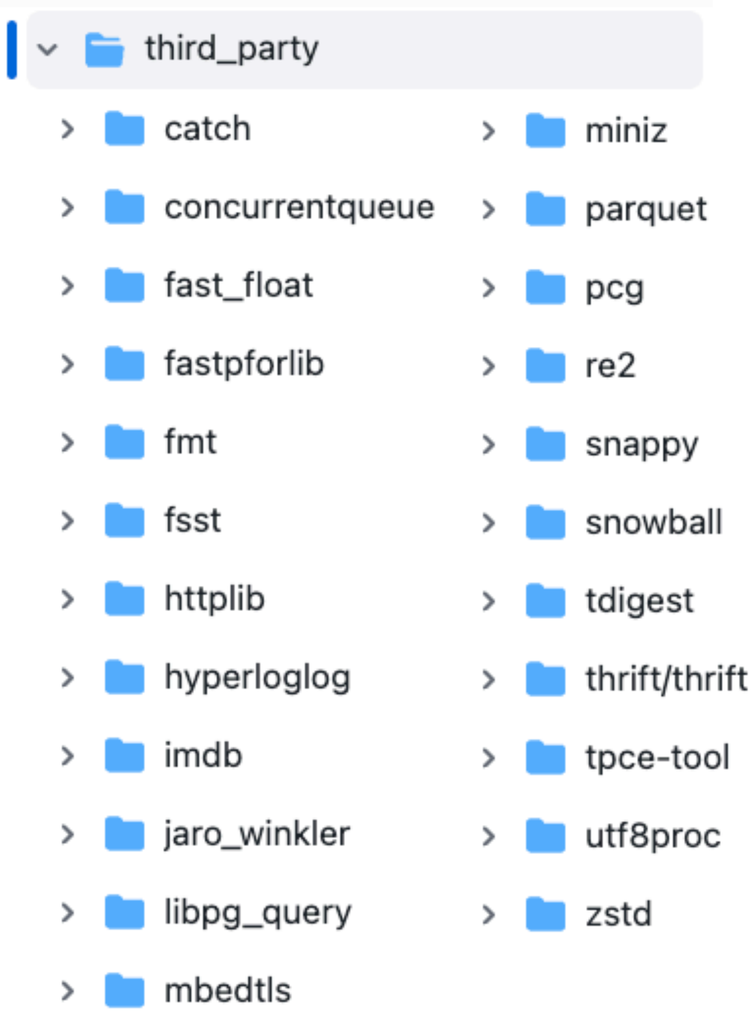
DuckDB has zero external dependencies

Dependencies are vendored in the codebase

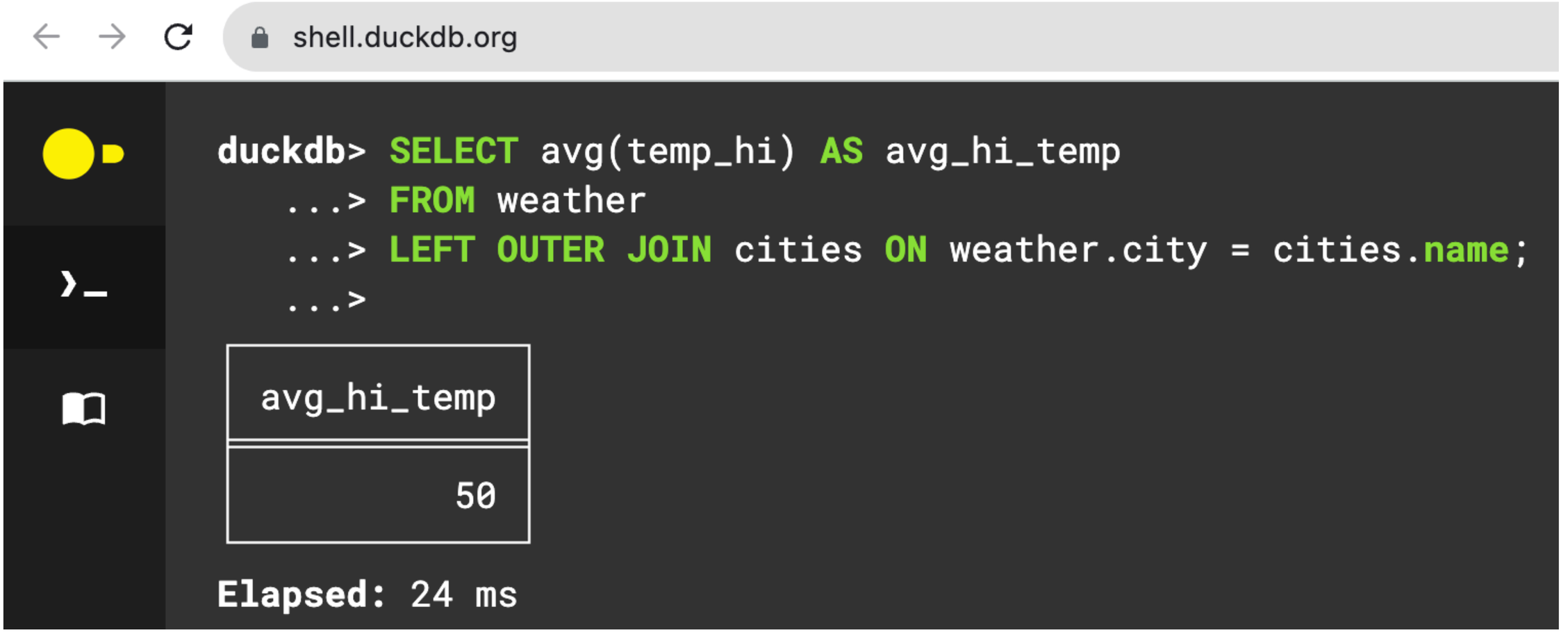
Pure C/C++ codebase

Portable anywhere with a C++11 compiler

Small binary packages



WebAssembly (Wasm)



Fast

CSV reader performance

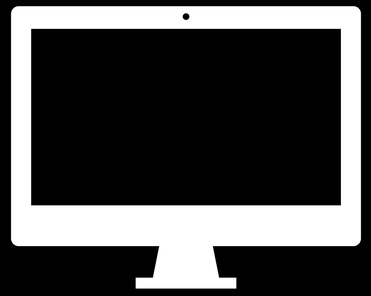
Test data: LDBC social network data set

Setup: M2Pro CPU, 32GB RAM, DuckDB v0.9.1

CSV size	Load time	Database size
3.4 GB	3.2 s	1 GB
35 GB	27 s	10 GB
360 GB	4 min 54 s	104 GB

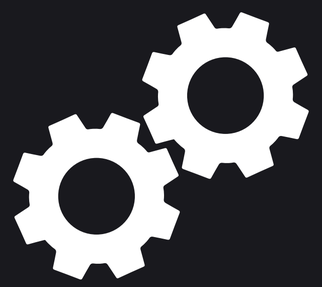
≈3.5x compression

>1.2 GB/s for reading CSV, parsing, and writing to DuckDB

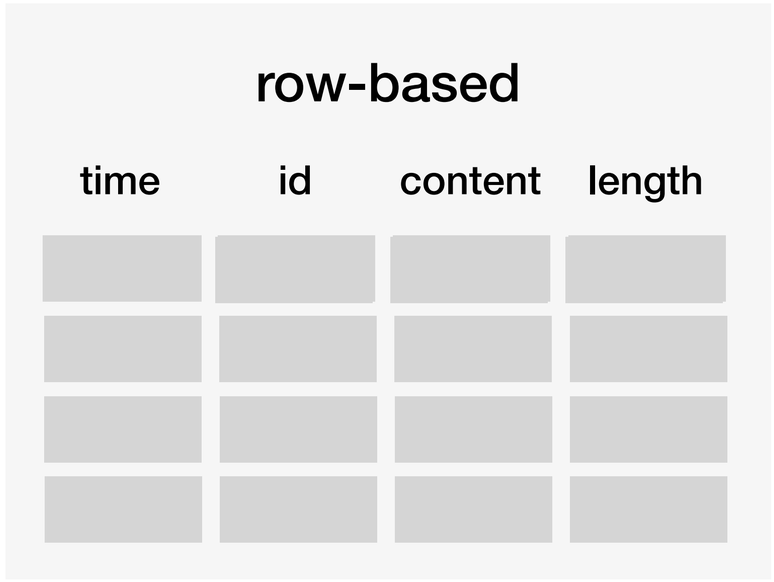


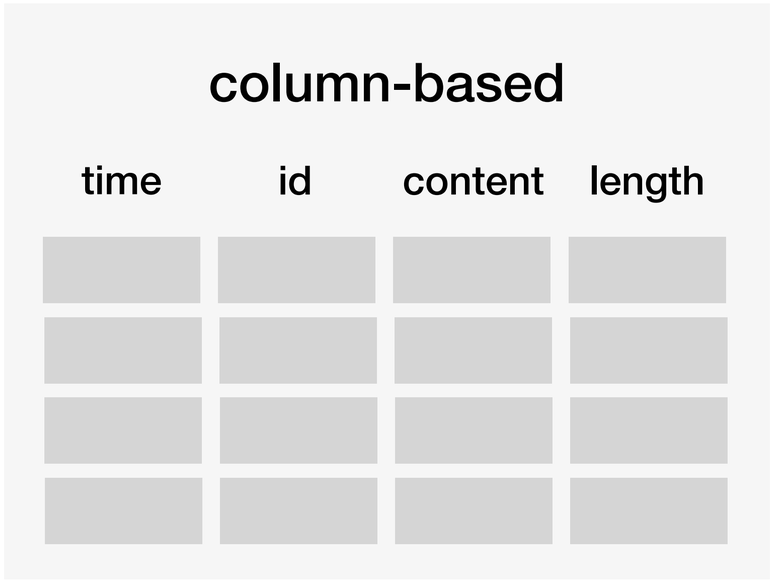
Demo

Internals

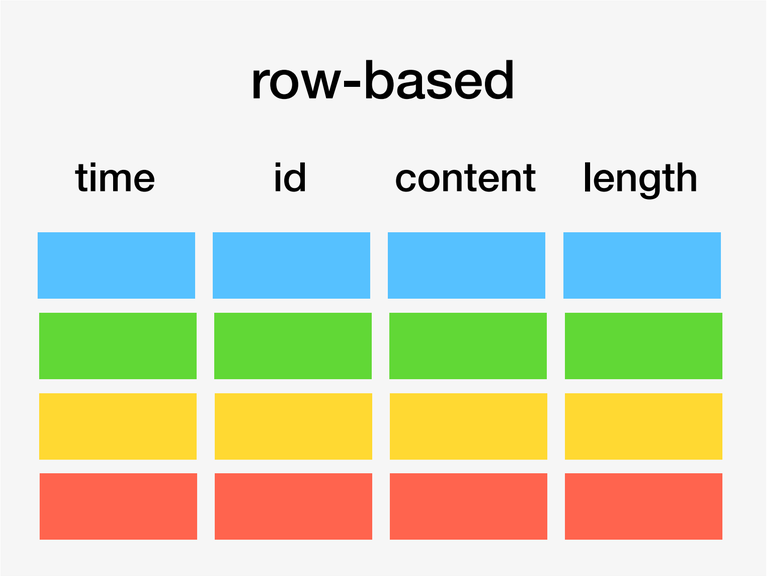


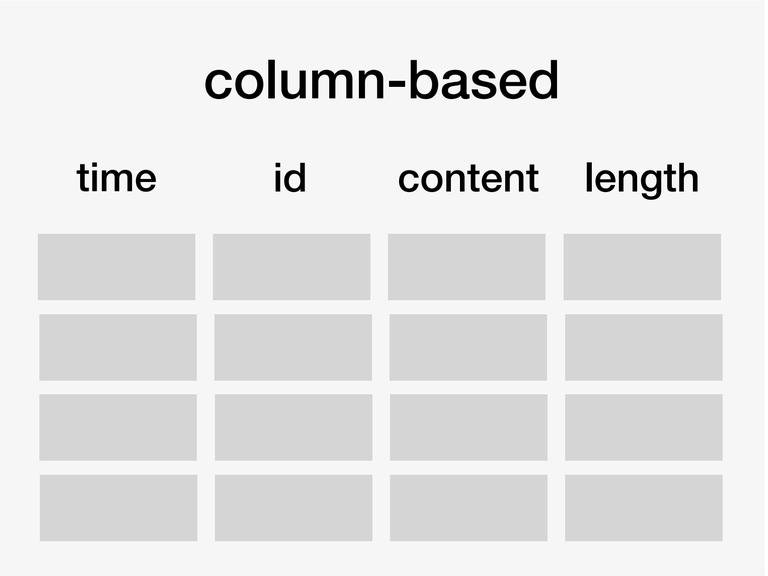
Storage


row-based
time
id
content
length

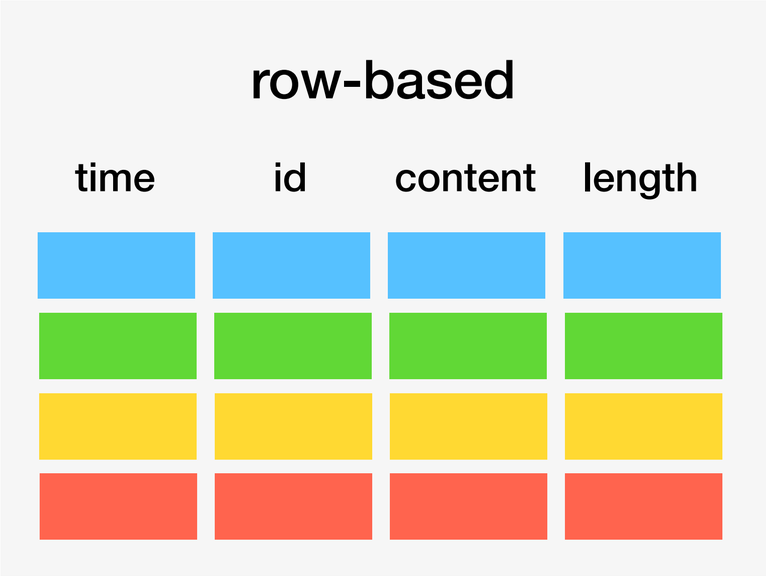

column-based
time
id
content
length

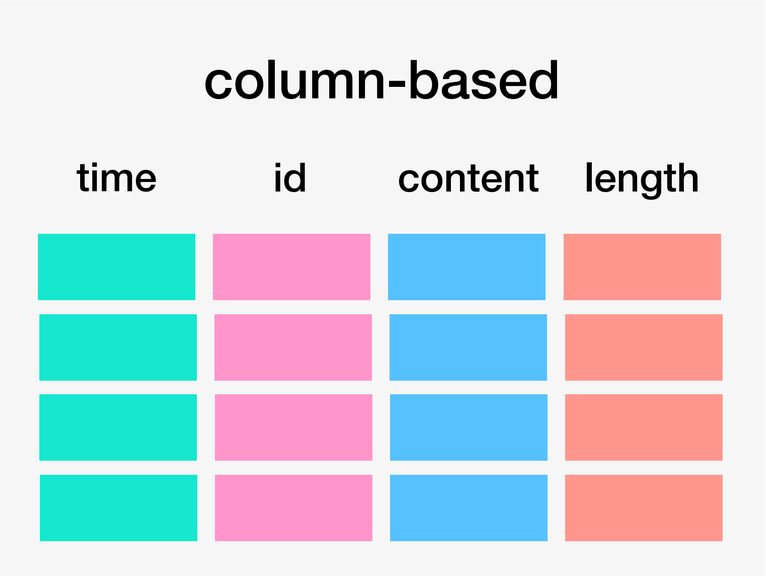
Storage


row-based
time
id
content
length

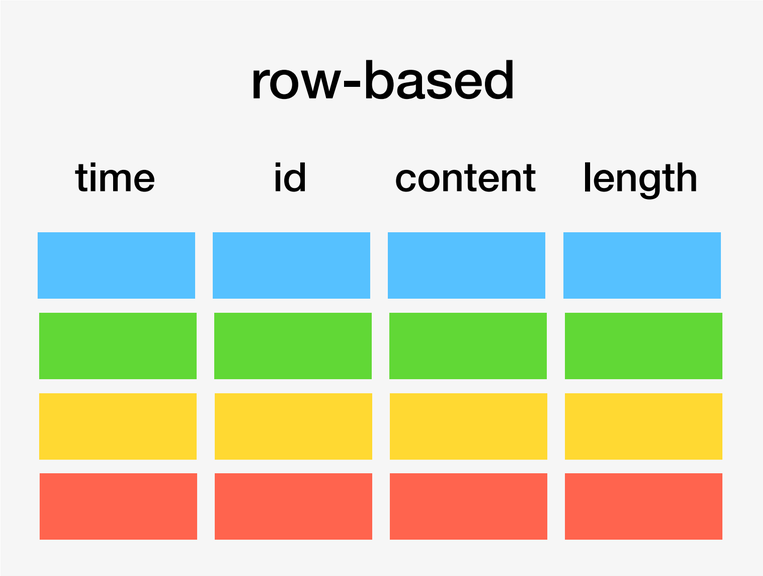

column-based
time
id
content
length

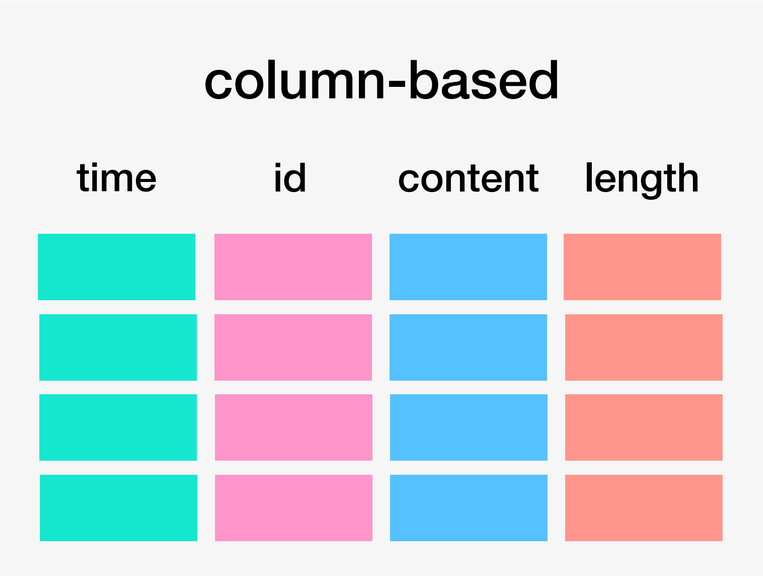
Storage

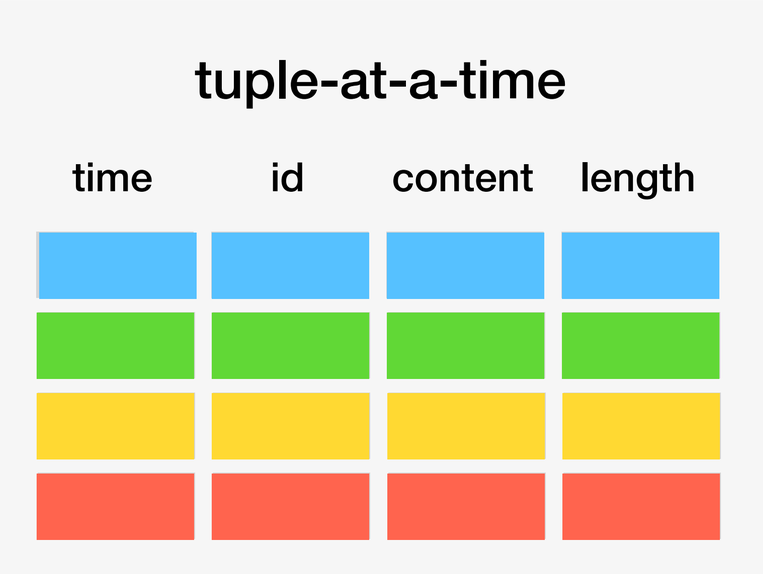

row-based
time
id
content
length

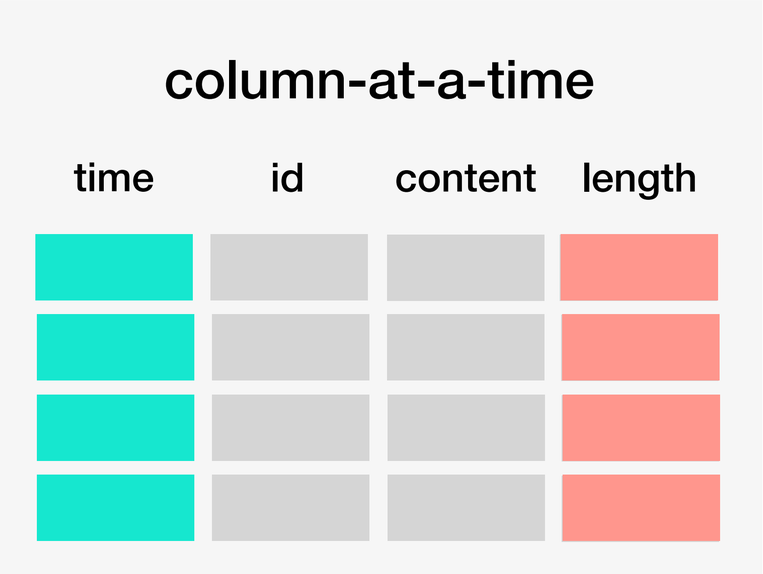

column-based
time
id
content
length

Execution

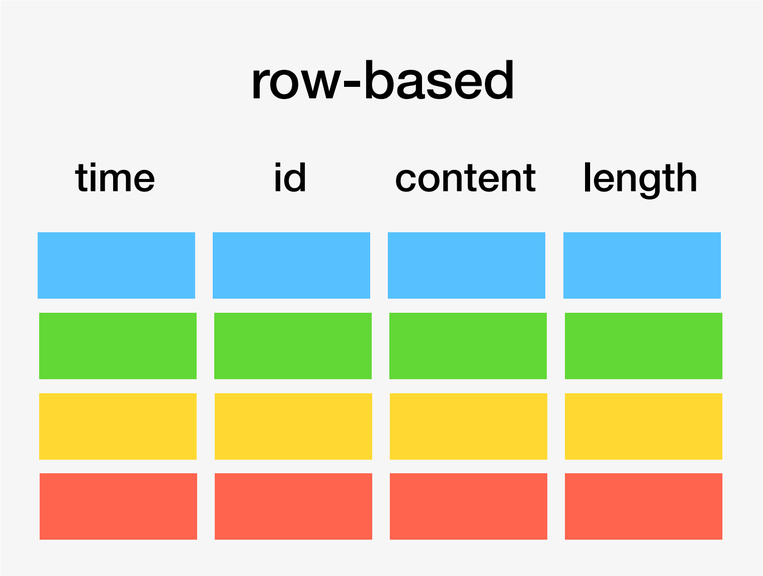

row-based
time
id
content
length

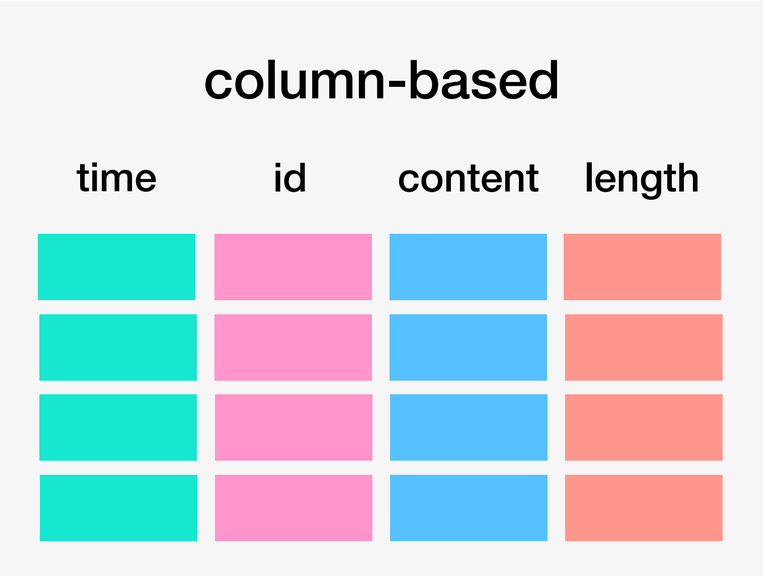

column-based
time
id
content
length

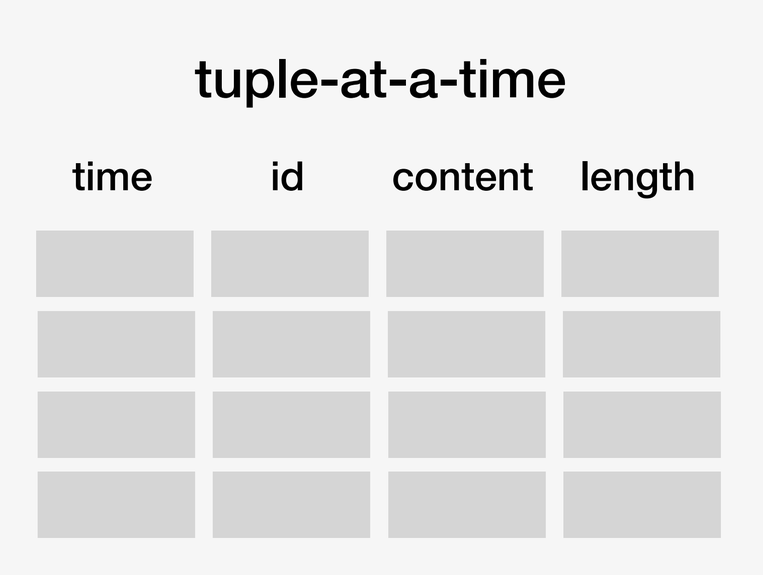

tuple-at-a-time
time
id
content
length

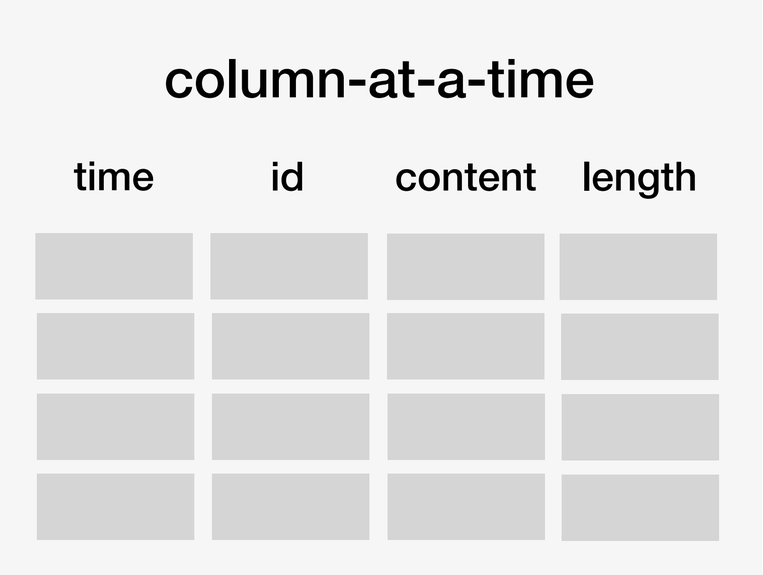

column-at-a-time
time
id
content
length

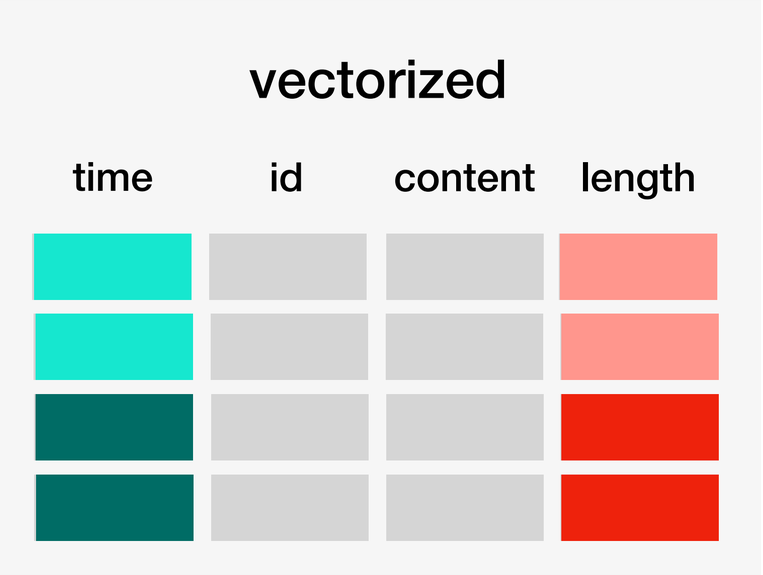
Execution


row-based
time
id
content
length

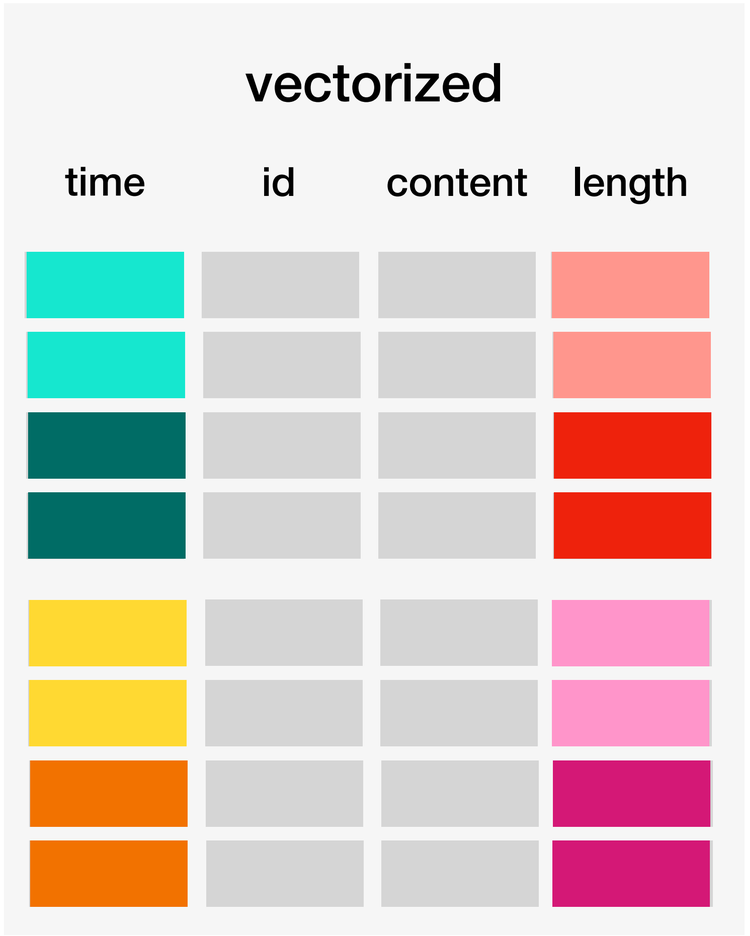

column-based
time
id
content
length


tuple-at-a-time
time
id
content
length


column-at-a-time
time
id
content
length


vectorized
time
id
content
length

Vectorized execution



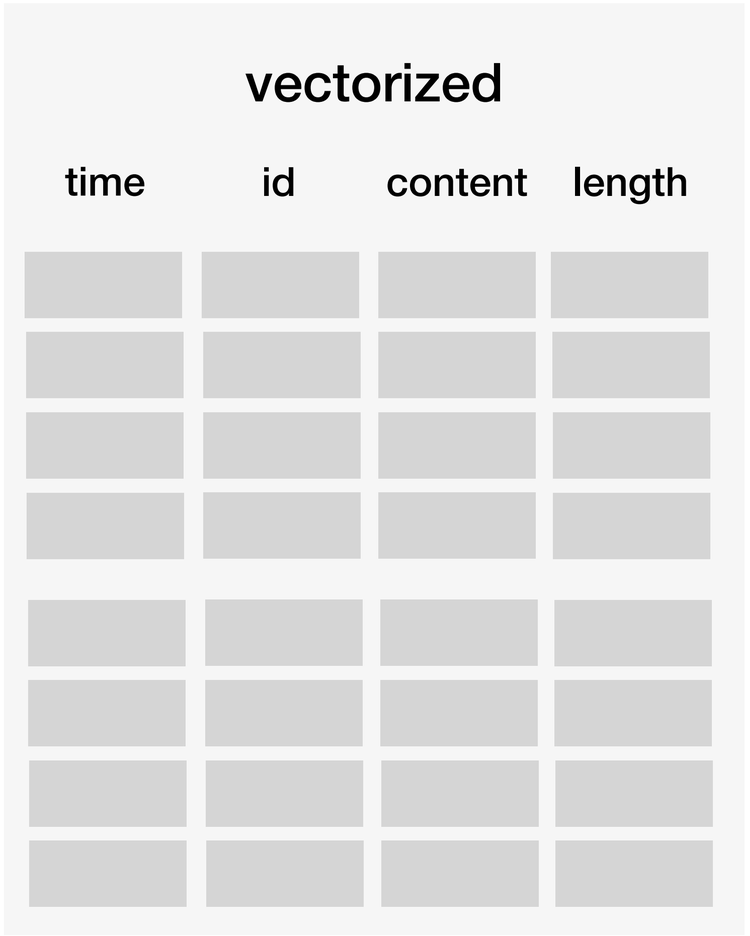
thread 1

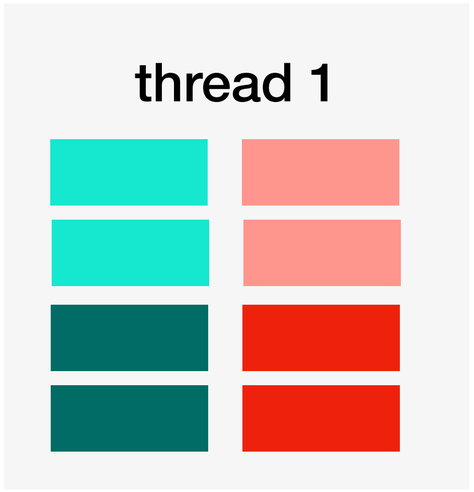
L1 cache

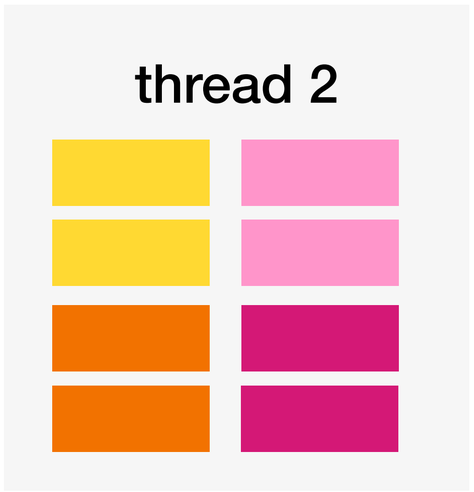
thread 2

L1 cache

Vectorized execution



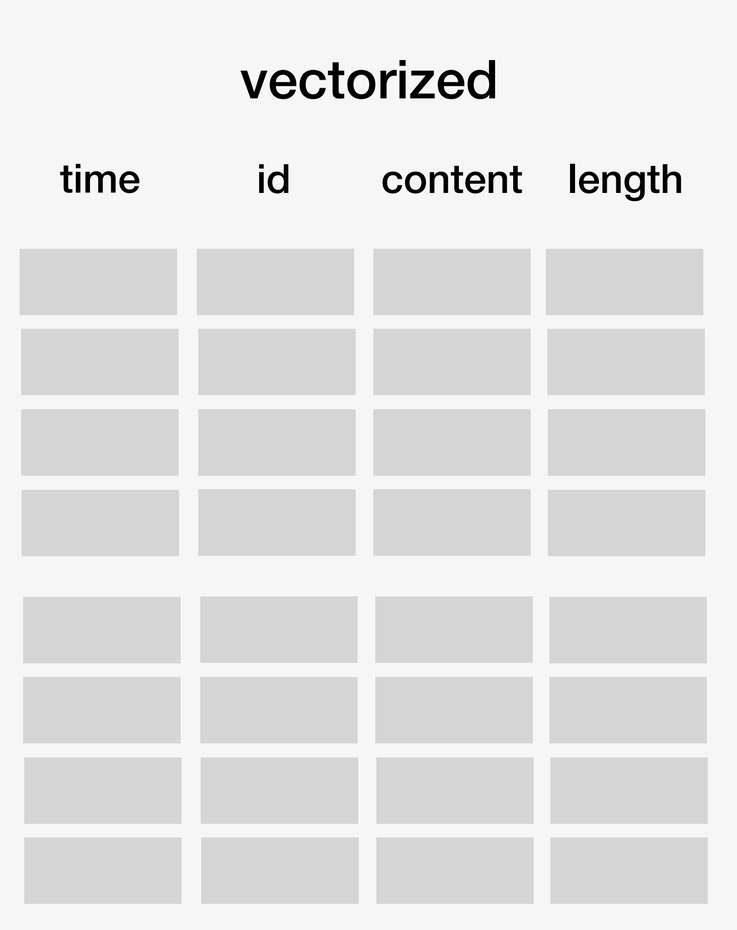


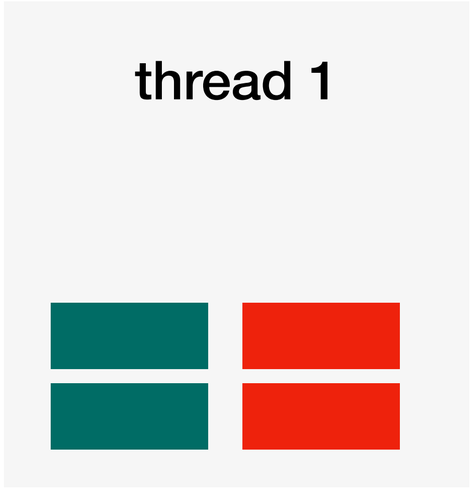


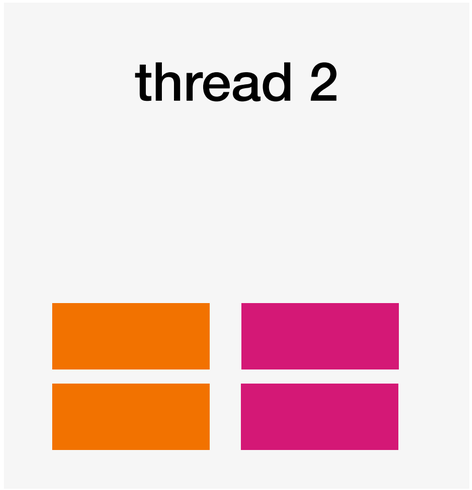
L1 cache

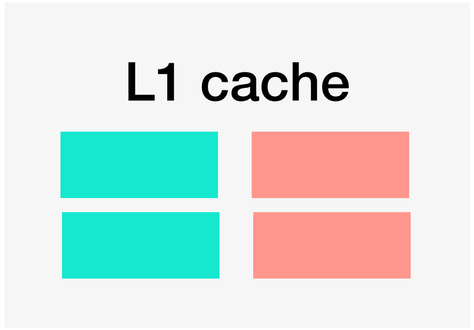
L1 cache

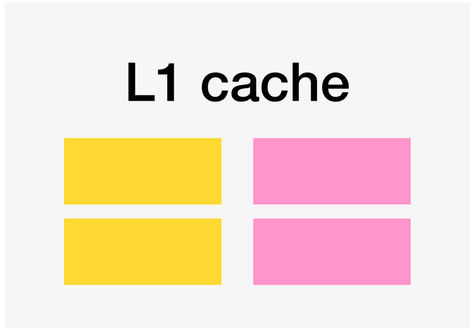
Vectorized execution



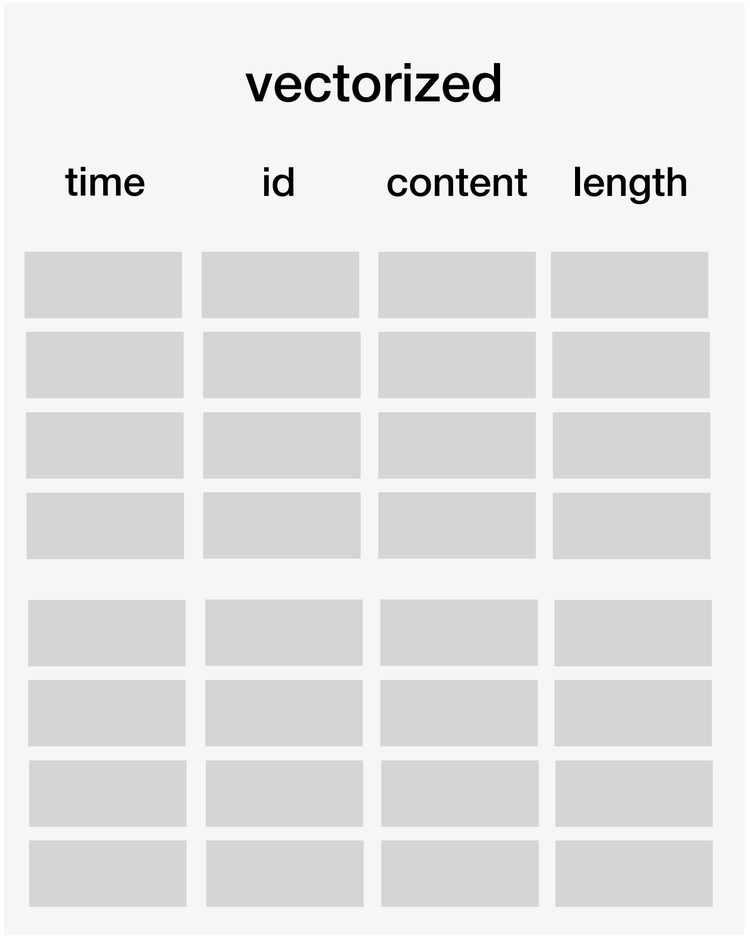






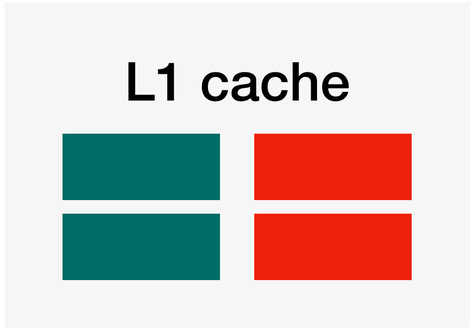


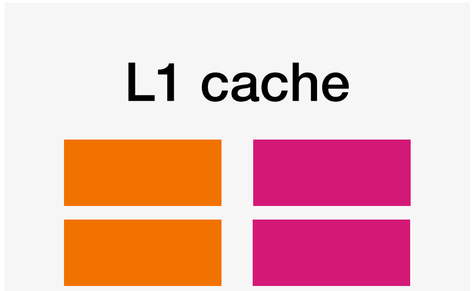
Vectorized execution



thread 1

thread 2





Vectors fit into
L1 cache (32–128kB)

SIMD vectorization
(AVX-512: 8×64-bit ints)

Modern compilers
auto-vectorize code

Parallelization along
row groups

Indexing: Zone maps

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

min	max
Nov 7	Nov 8

min	max
Nov 8	Nov 12

time	id	content	length
Nov 7			74
Nov 7			109
Nov 8			67
Nov 8			63
Nov 8			95
Nov 9			113
Nov 11			14
Nov 11			8

min	max
63	109

min	max
8	95

Indexing with the Adaptive Radix Tree (ART)

DuckDB supports secondary indexes:

- implicit indexes – primary key, foreign key, unique
- explicit indexes – `CREATE [UNIQUE] INDEX`

Tradeoffs:

- speed-up for high selectivity lookups
- negative performance impact for updates

Rule of thumb:

Most of the time indexes are not needed

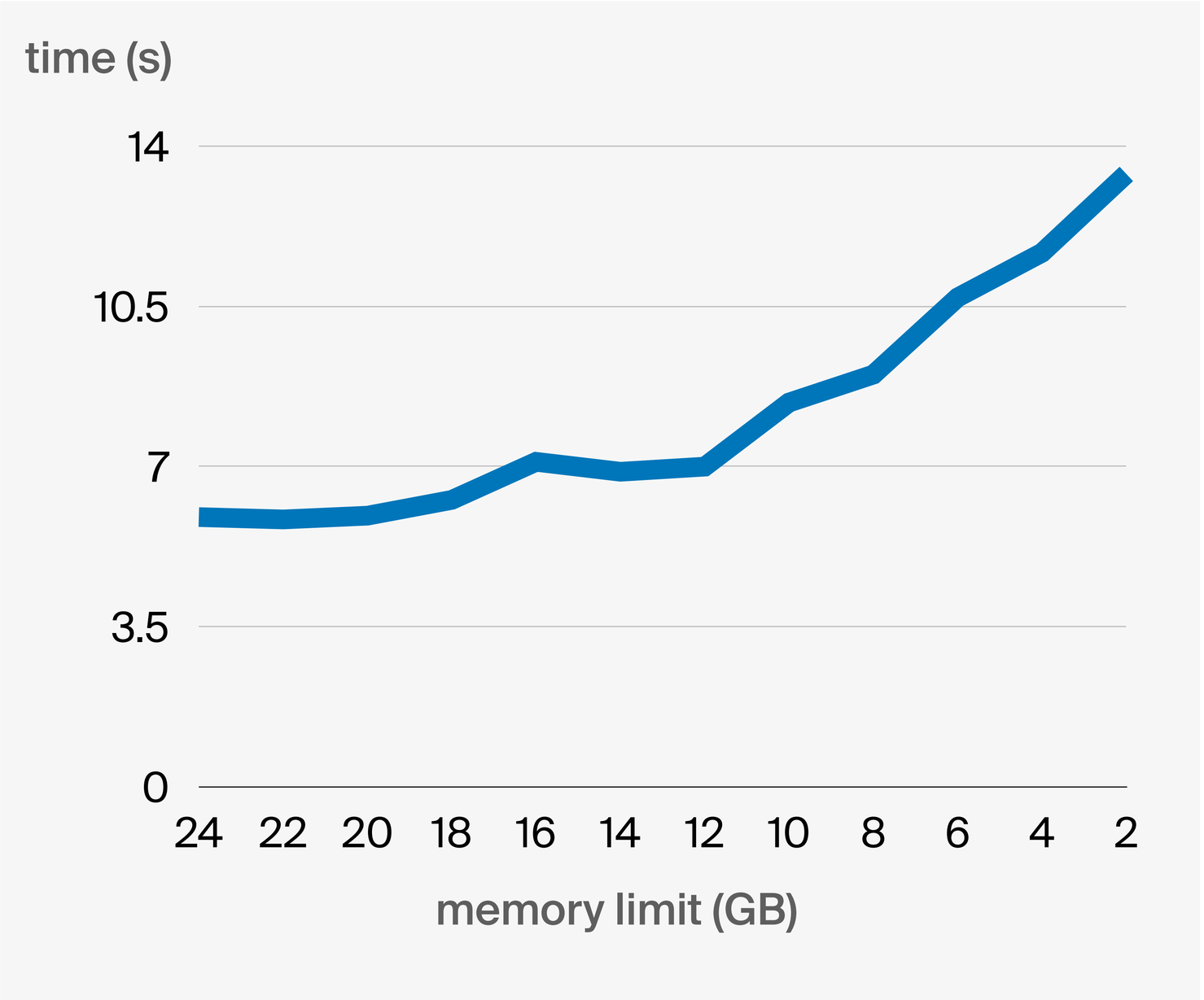
Larger-than-memory execution: Joins and aggregations

Larger-than-memory execution

- Graceful degradation
- Always try to finish

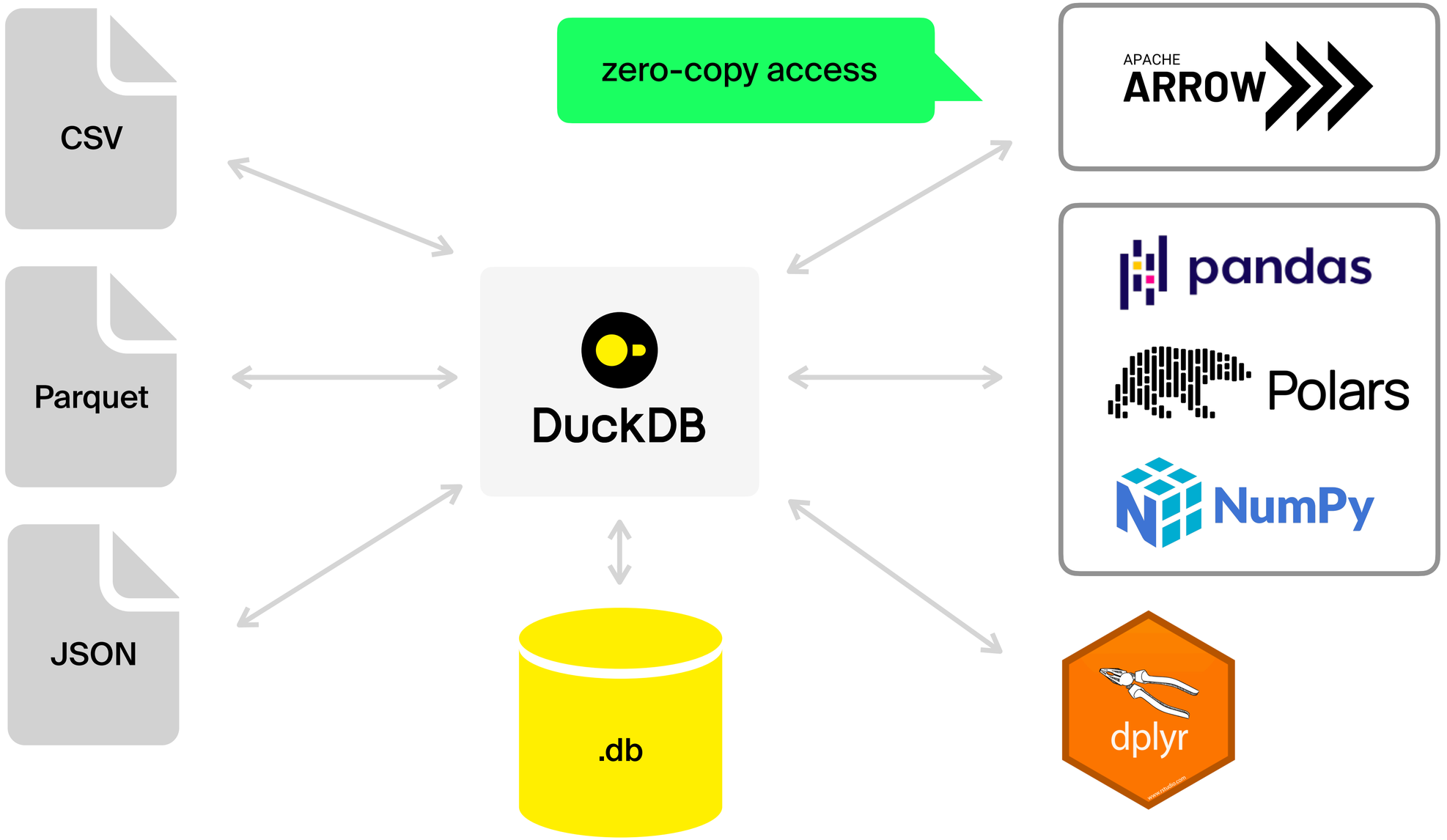
Example:

- TPC-H SF100
- Query 7



Feature-rich

Input and output formats



Query language

PostgreSQL dialect:

- Subqueries
- Window functions
- Common table extensions
- Lateral joins
- Range joins
- AsOf joins
- Pivoting and unpivoting tables

"Friendly SQL" extensions

```
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
```

DuckDB SQL: FROM-first syntax

Common pattern:

```
SELECT *
FROM Comment;
```

Friendly variant:

```
FROM Comment;
```

DuckDB SQL: EXCLUDE columns

Common pattern:

```
SELECT
  creationDate, id, locationIP, browserUsed, content,
  length, CreatorPersonId, LocationCountryId
FROM Comment;
```

Friendly variant:

```
SELECT * EXCLUDE (ParentCommentId, ParentPostId)
FROM Comment;
```

DuckDB SQL: GROUP BY ALL

Common pattern:

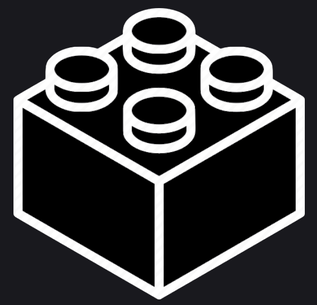
```
SELECT month(creationDay), count(*) AS numComments
FROM Comment;
```

--> syntax error

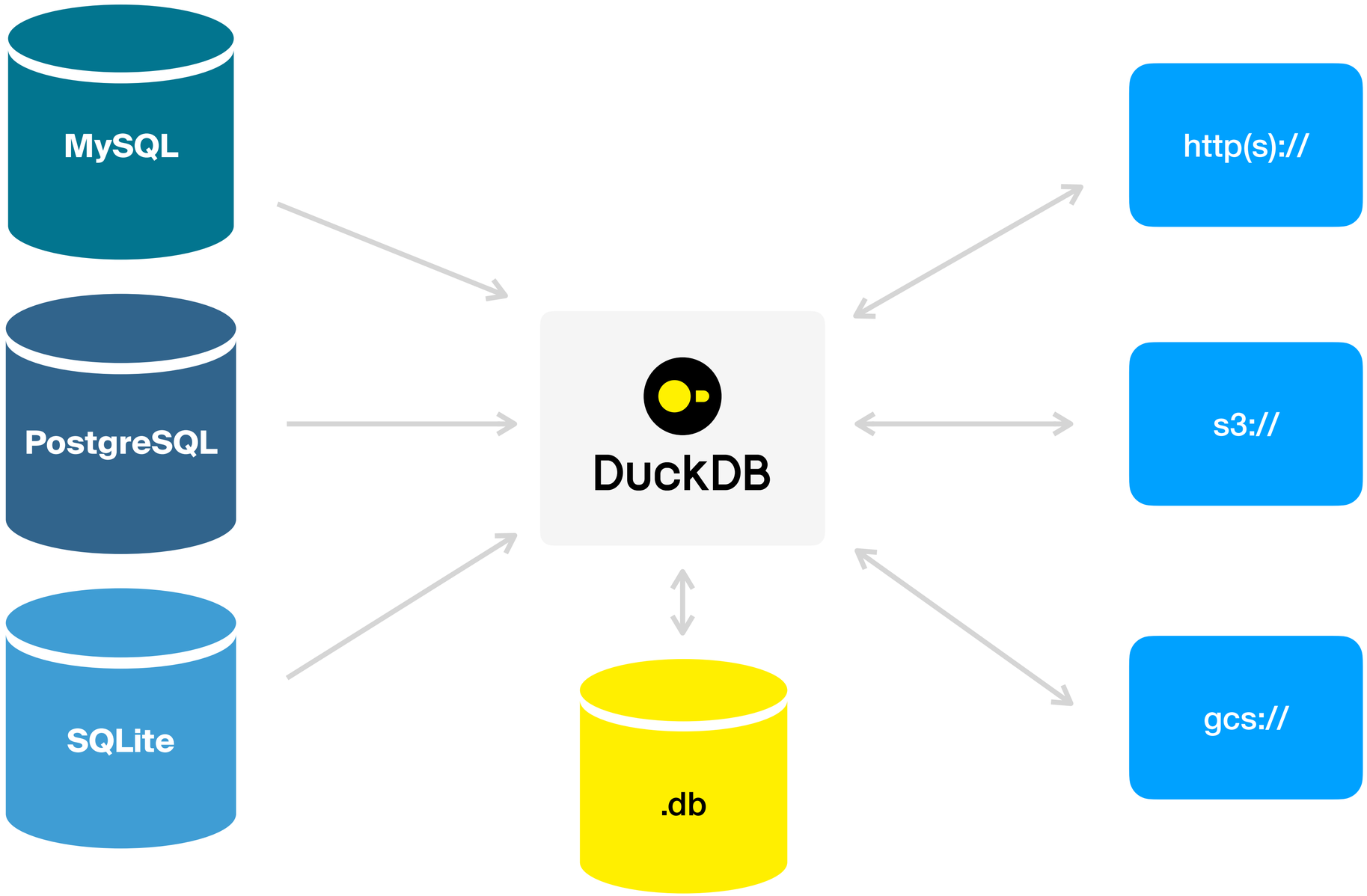
Friendly variant:

```
SELECT month(creationDay), count(*) AS numComments
FROM Comment
GROUP BY ALL;
```

Extensions



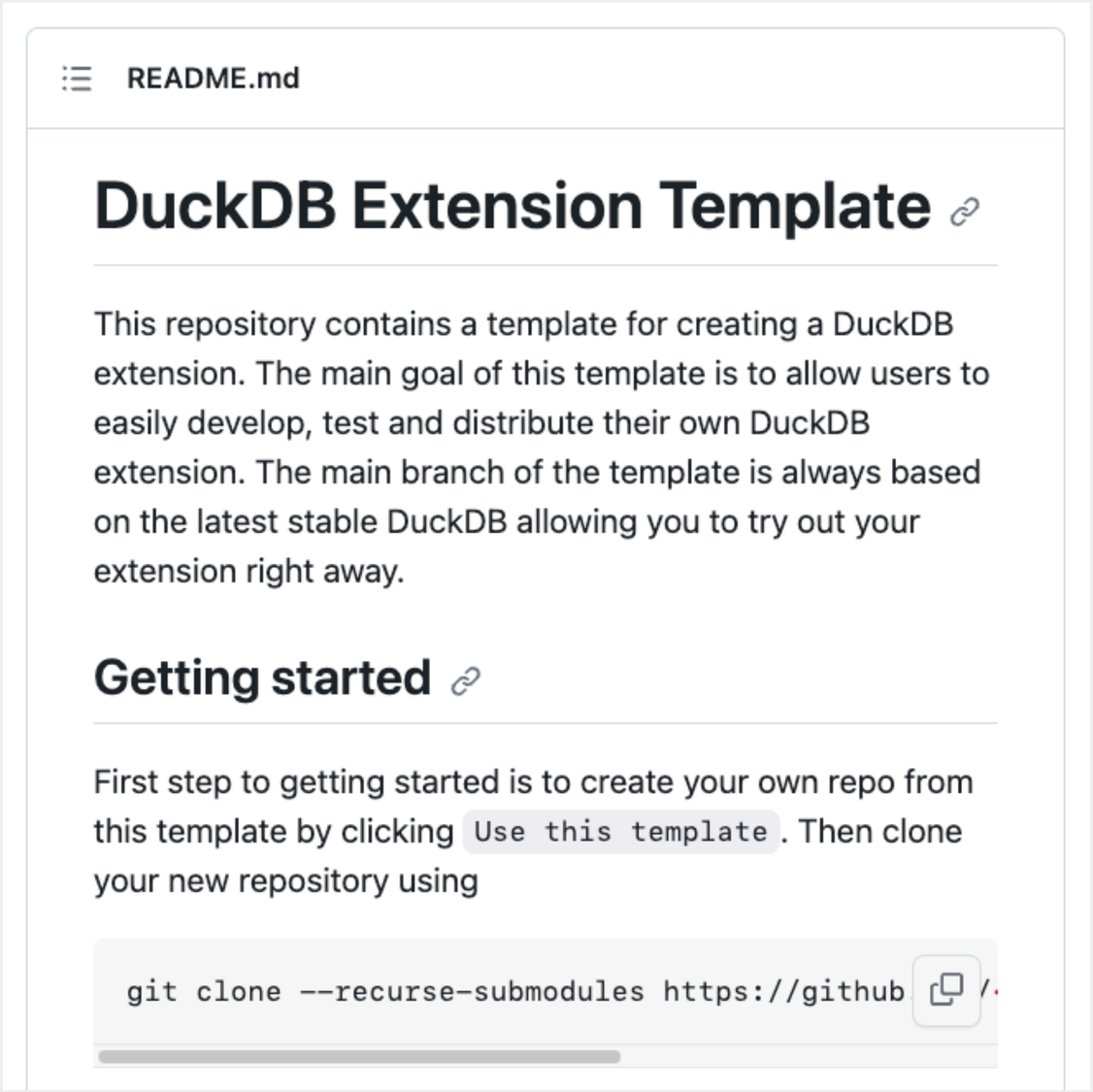
Data sources and destinations



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

Spatial extension

- Adds PostGIS-like functionality: geospatial types for points, polygons, etc.
- Adds functions for calculating distances

Example: aerial distance on the New York taxi data set

```
SELECT
  st_point(pickup_latitude, pickup_longitude) as pickup_point,
  st_point(dropoff_latitude, dropoff_longitude) as dropoff_point,
  dropoff_datetime::TIMESTAMP - pickup_datetime::TIMESTAMP AS time,
  trip_distance,
  st_distance(
    st_transform(pickup_point,  'EPSG:4326', 'ESRI:102718'),
    st_transform(dropoff_point, 'EPSG:4326', 'ESRI:102718')) / 5280 AS aerial_distance,
  trip_distance - aerial_distance AS diff
FROM rides
WHERE diff > 0
ORDER BY diff DESC;
```

DuckDB

Harnessing in-process analytics for data science and beyond

Gábor Szárnyas
Developer Relations Advocate

DuckDB Labs

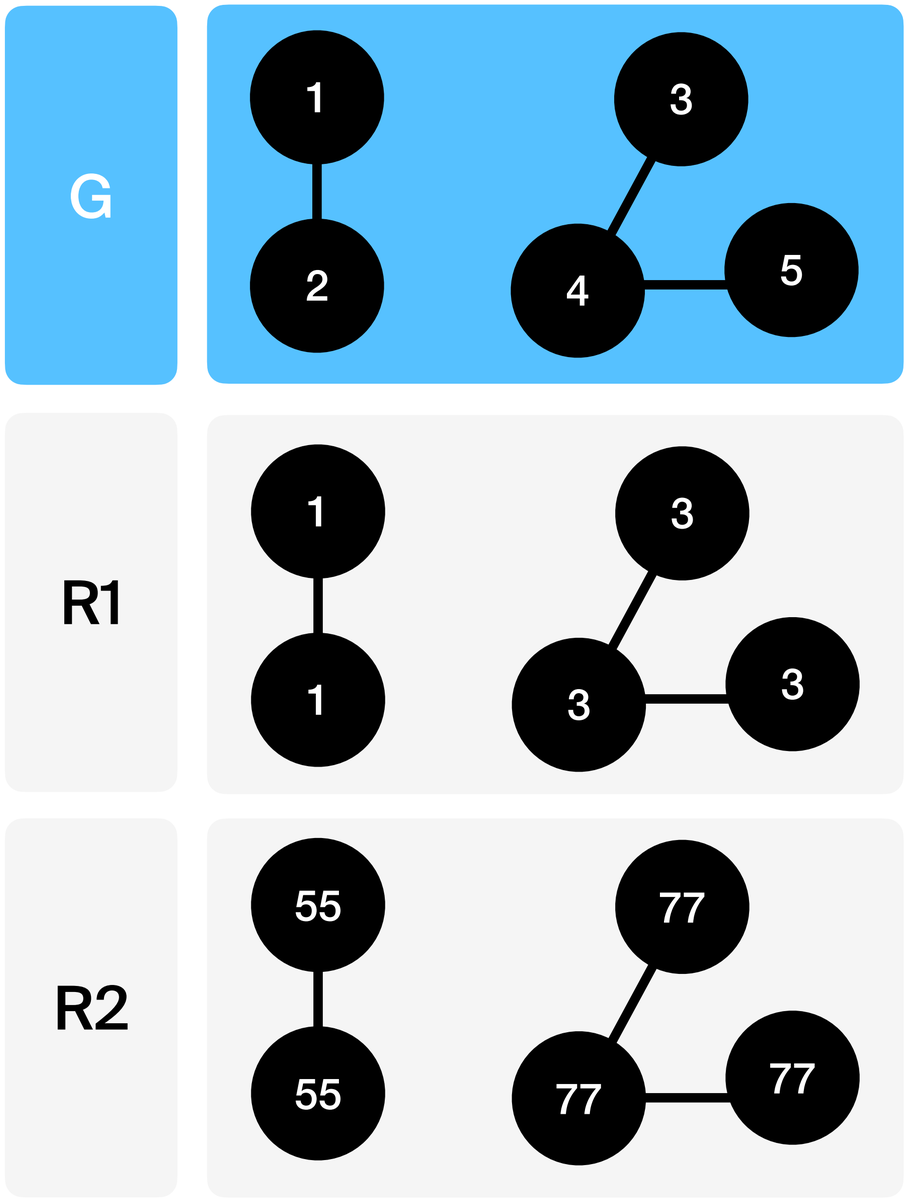
DuckDB

Harnessing in-process analytics for data science and beyond

Gábor Szárnyas
Developer Relations Advocate

DuckDB Labs

Modernizing a graph algorithm benchmark



Context:

Graph benchmark from 2015 (legacy code!)

Goal: find connected components quickly

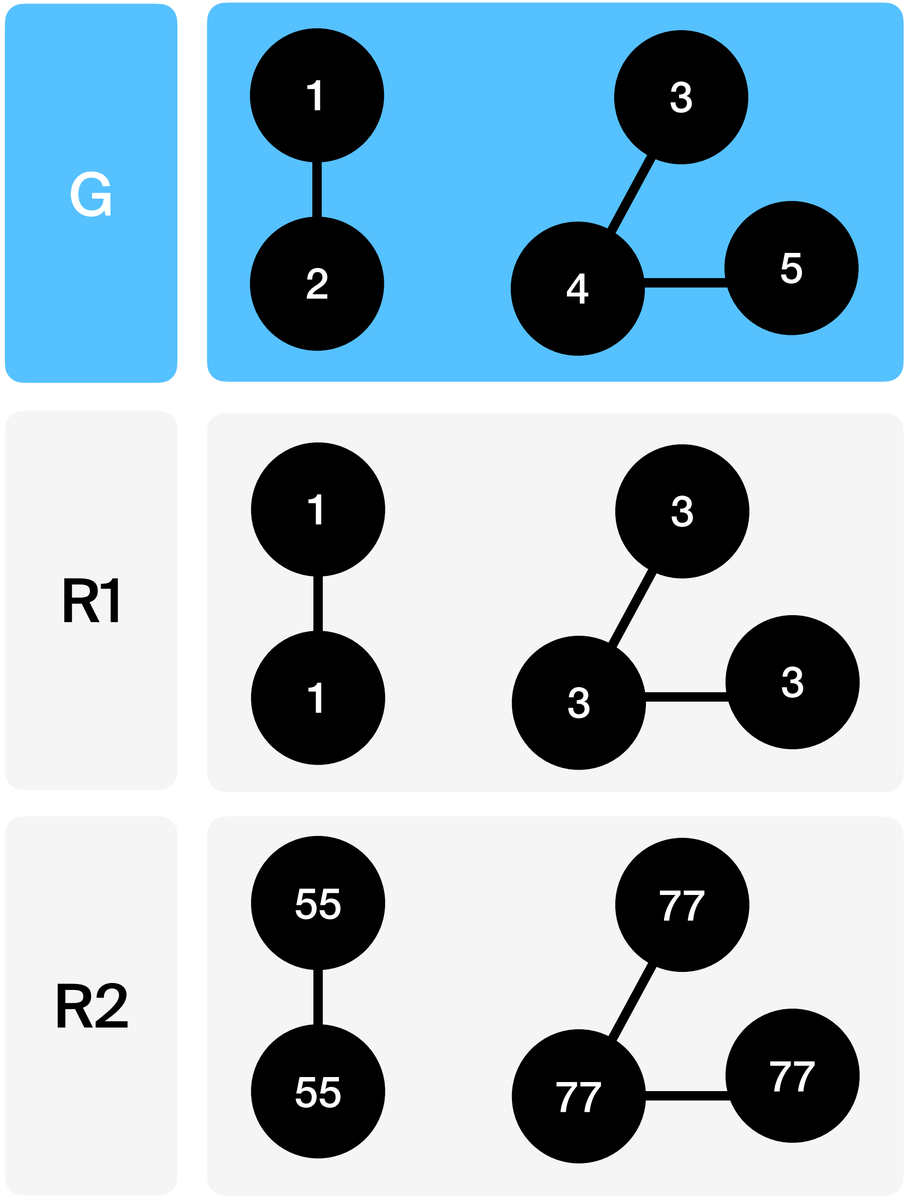
Validation rule:

The result encode equivalence classes (R1=R2)

Problem:

The validation became very slow for large graphs (single-threaded Java code building hashmaps)

Modernizing a graph algorithm benchmark



Output validation using matching in SQL #217

Merged **szarnyasg** merged 10 commits into `main` from `output-validation-using-matching`

Conversation 0 · Commits 10 · Checks 1 · Files changed 25

szarnyasg commented on Aug 24, 2022 • edited · Member

Will fix #205.

We can use the DuckDB appender to populate the tables.

Current validation scripts are in:

- https://github.com/ldbc/ldbc_graphalytics/tree/master/graphalytics-core/src/main/java/science/atlarge/graphalytics/validation
- https://github.com/ldbc/ldbc_graphalytics/tree/master/graphalytics-core/src/main/java/science/atlarge/graphalytics/validation/rule

A lot of time is spent parsing the results back from CSVs to Java data structures, this could also be improved by using DuckDB's

Reviewers
No reviews

Assignees
No one assigned

Labels
None yet

Projects
None yet

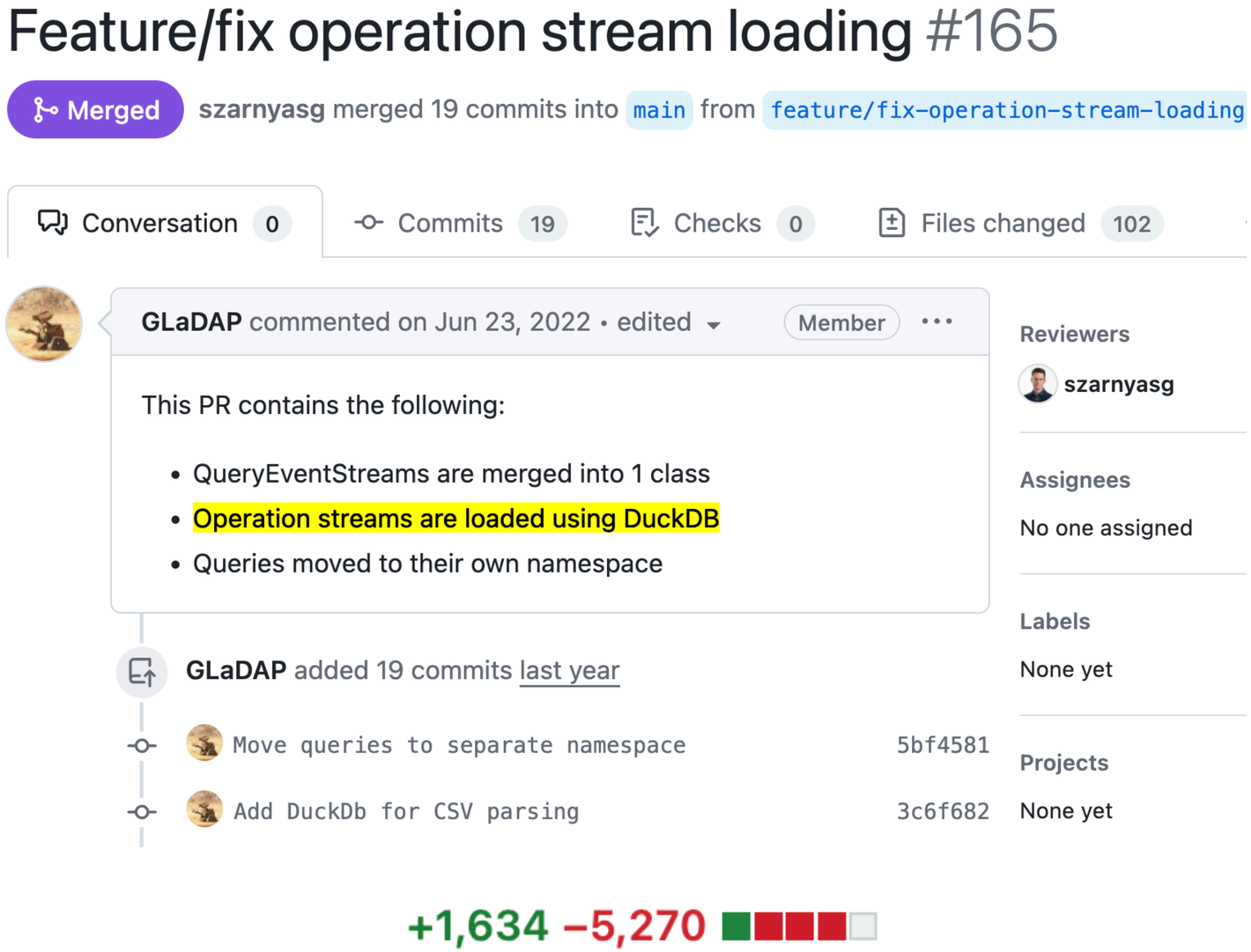
Milestone

+338 −457

More benchmark framework use cases

- Output validation
- Loading operation streams
- Query parameter generation
- Reading input parameters
- Preprocessing raw data
- Partitioning update streams
- Analyzing results

None of this is a DB problem...

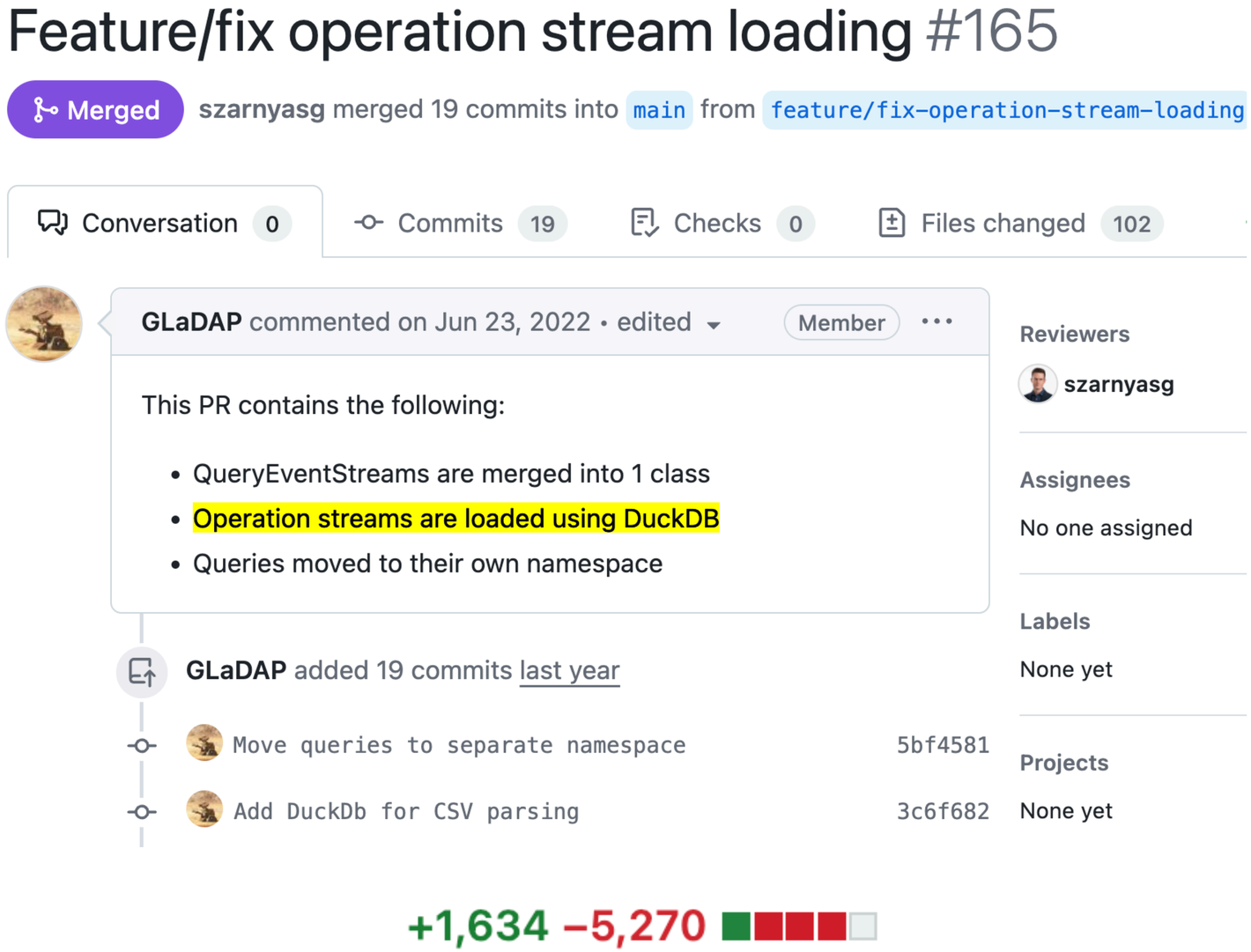


More benchmark framework use cases

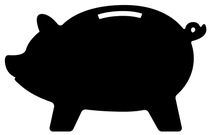
- Output validation
- Loading operation streams
- Query parameter generation
- Reading input parameters
- Preprocessing raw data
- Partitioning update streams
- Analyzing results

None of this is a DB problem...

But they are bulky operations on heavily structured data.

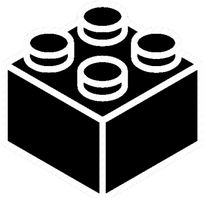


Use cases



Saving costs:

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



Building block in applications:

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

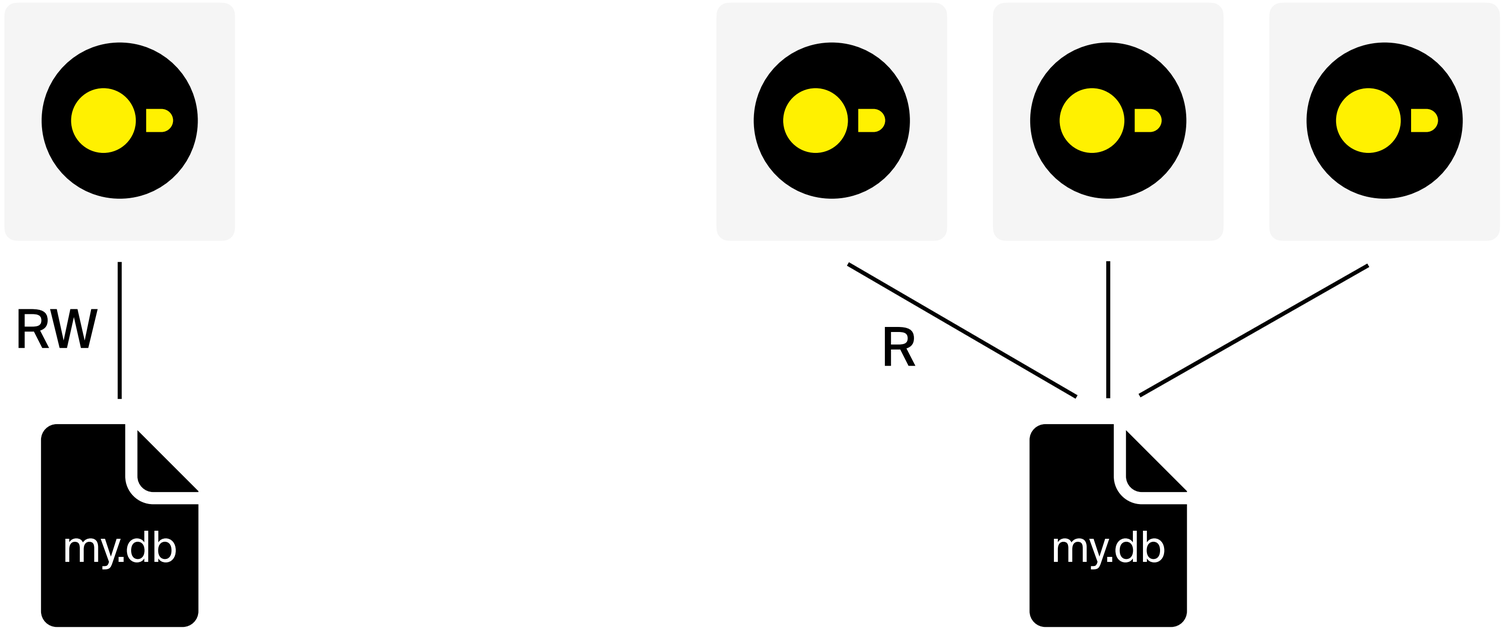
Education:

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

Limitations

Concurrency control

- ACID compliance via multi-version concurrency control (MVCC)
- WAL (write-ahead log) for recovery
- Not a good fit for write-heavy transactional workloads



Distributed execution

DuckDB only supports **single-node** execution

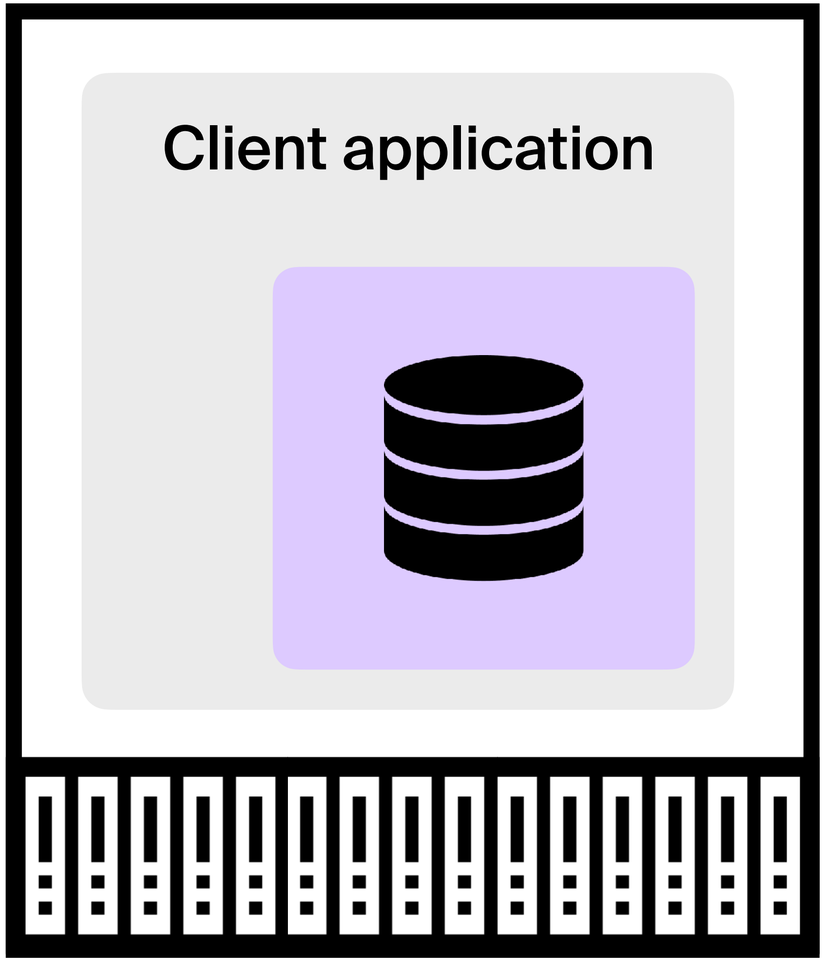
DuckDB can **scale up:**

- r6id.32xlarge instances have 1TB RAM for <$10/h
- x1e.32xlarge instances have 4TB RAM for ≈$28/h

Store the data in S3, run short bursts of workloads

Larger than memory execution allows scaling for TBs

For tens of TBs, a distributed setup is beneficial



The DuckDB landscape

DuckDB versions

`v0.9`	Current version

DuckDB versions

`v0.9`	Current version
`v0.10`	Early next year

DuckDB versions

`v0.9`	Current version
`v0.10`	Early next year
`v1.0`	Later next year

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

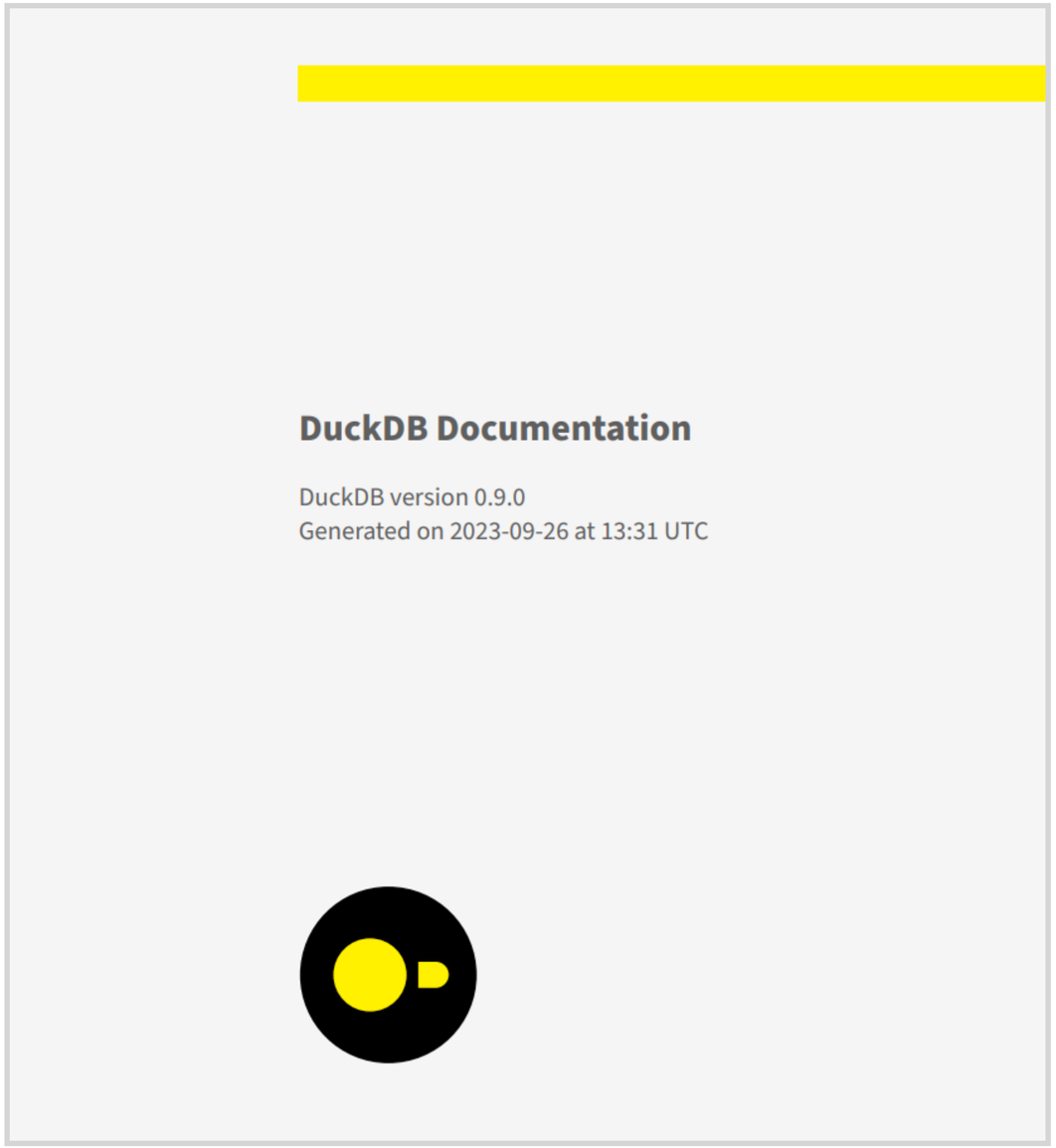
Classic open-source project

Full-fledged CLI client

Works when you're offline

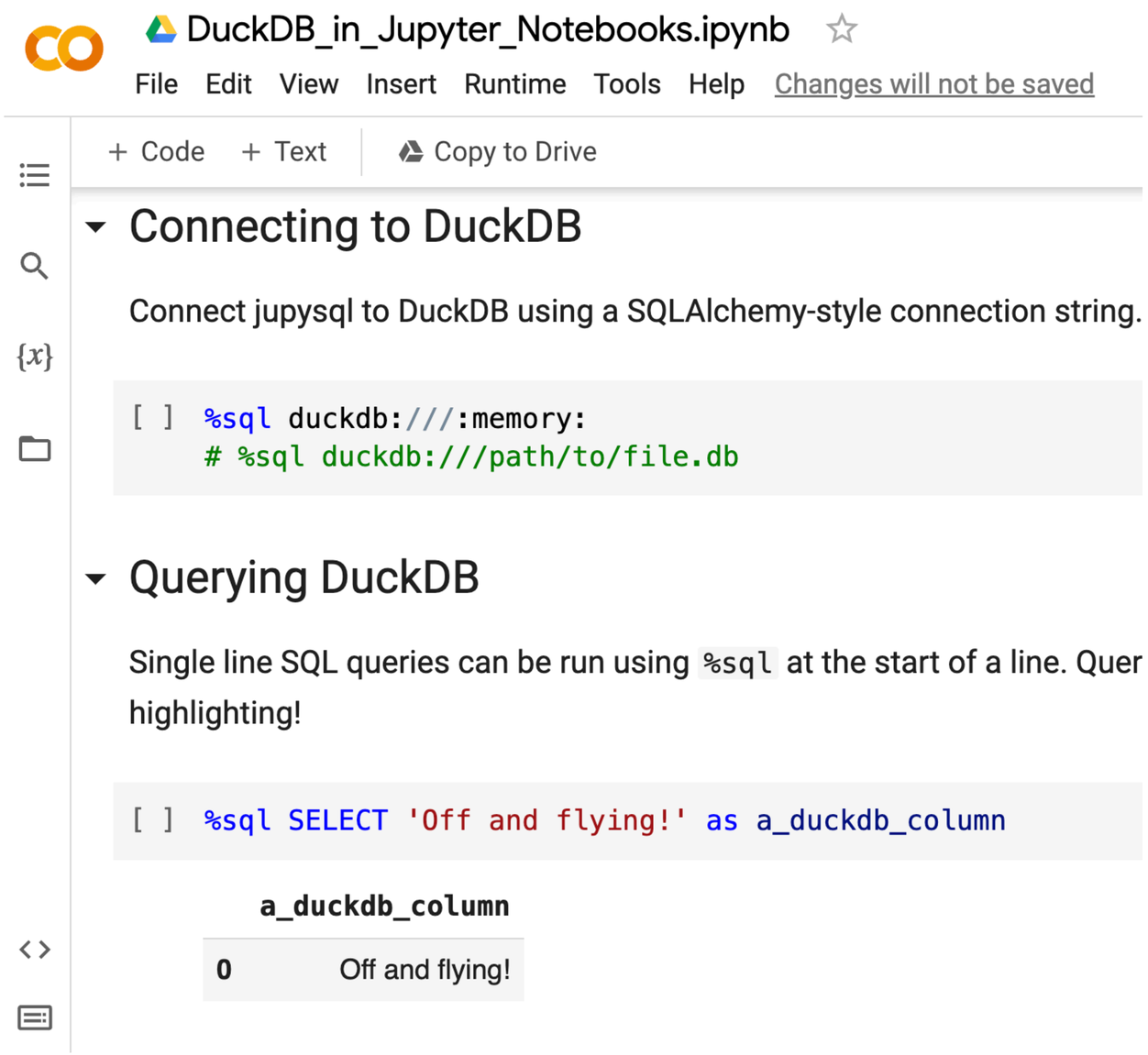
No vendor lock-in

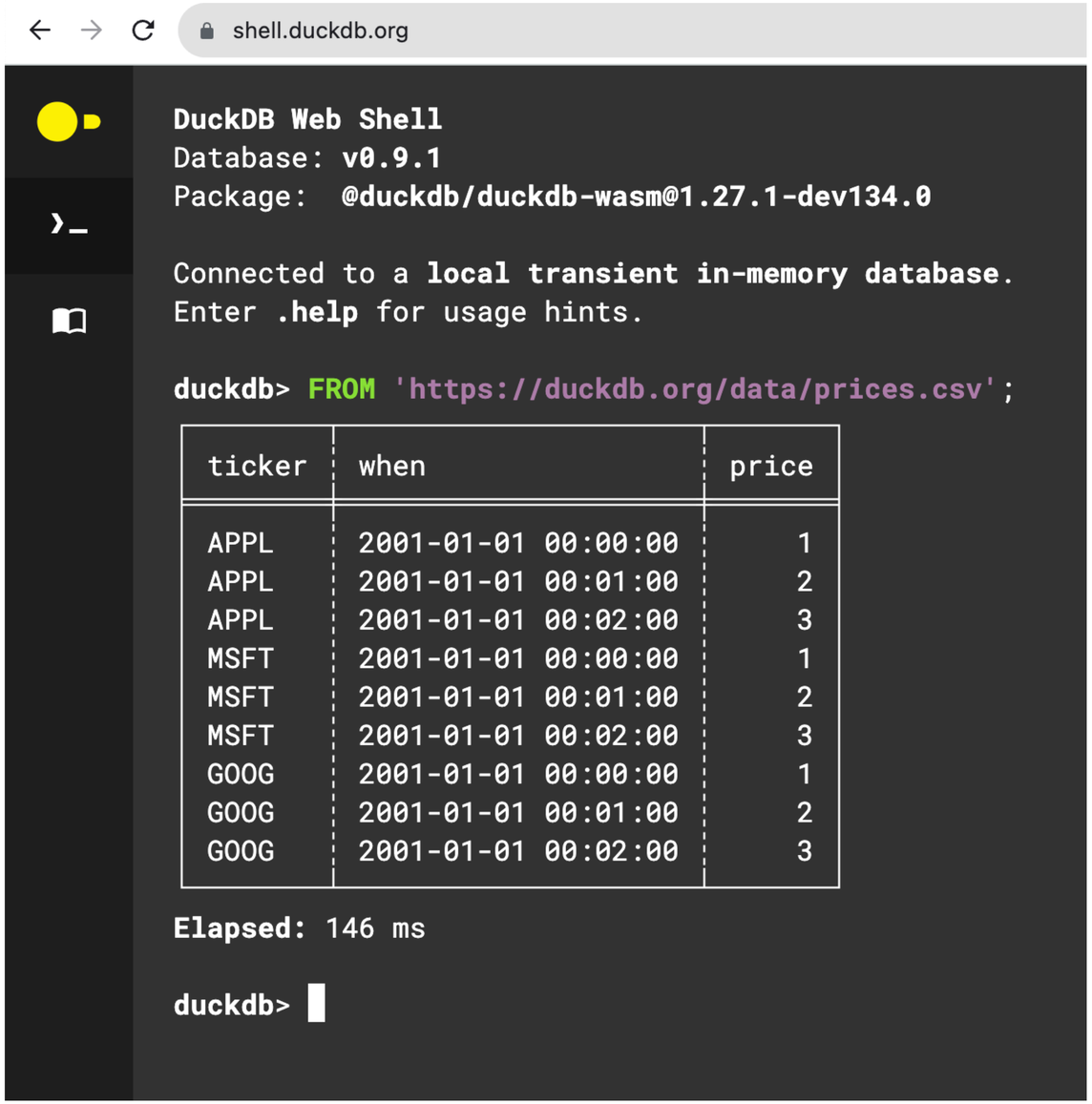
```
EXPORT DATABASE 'my_db' (FORMAT CSV);
EXPORT DATABASE 'my_db' (FORMAT PARQUET);
```



Give DuckDB a spin!

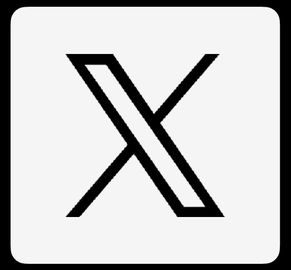
Google Colab, shell.duckdb.org





Stay in touch

discord.duckdb.org



@duckdb

duckdb.org

More DuckDB features

- Pandas-like relational API
- pySpark-compatible API
- Vectorized UDFs in Python
- Iceberg support
- JSON shredding
- Enum support
- Full text search
- dplyr integration
- Importing Hive-partitioned data
- dbt support
- Go client, Swift client, etc.