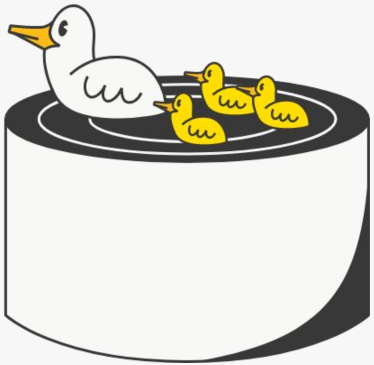
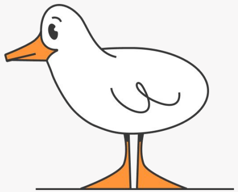
MotherDuck

PG_DUCKDB: DUCKING AWESOME ANALYTICS IN POSTGRES



Jelte Fennema-Nio (@JelteF)

2025-09-30



What is **pg_duckdb**?

pg_duckdb is a Postgres extension that embeds DuckDB inside Postgres

Ehhhmm what???

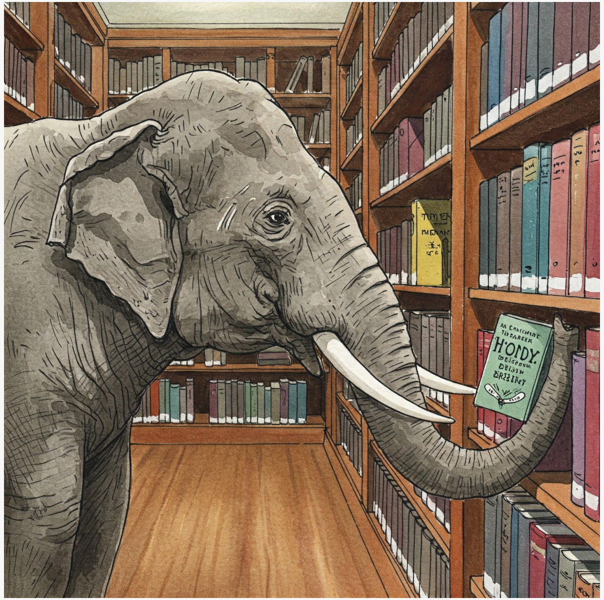
Postgres is an amazing database

MotherDuck

- Open source
- Many contributors
- Very stable
- Lots of built in functionality and extensible

MotherDuck

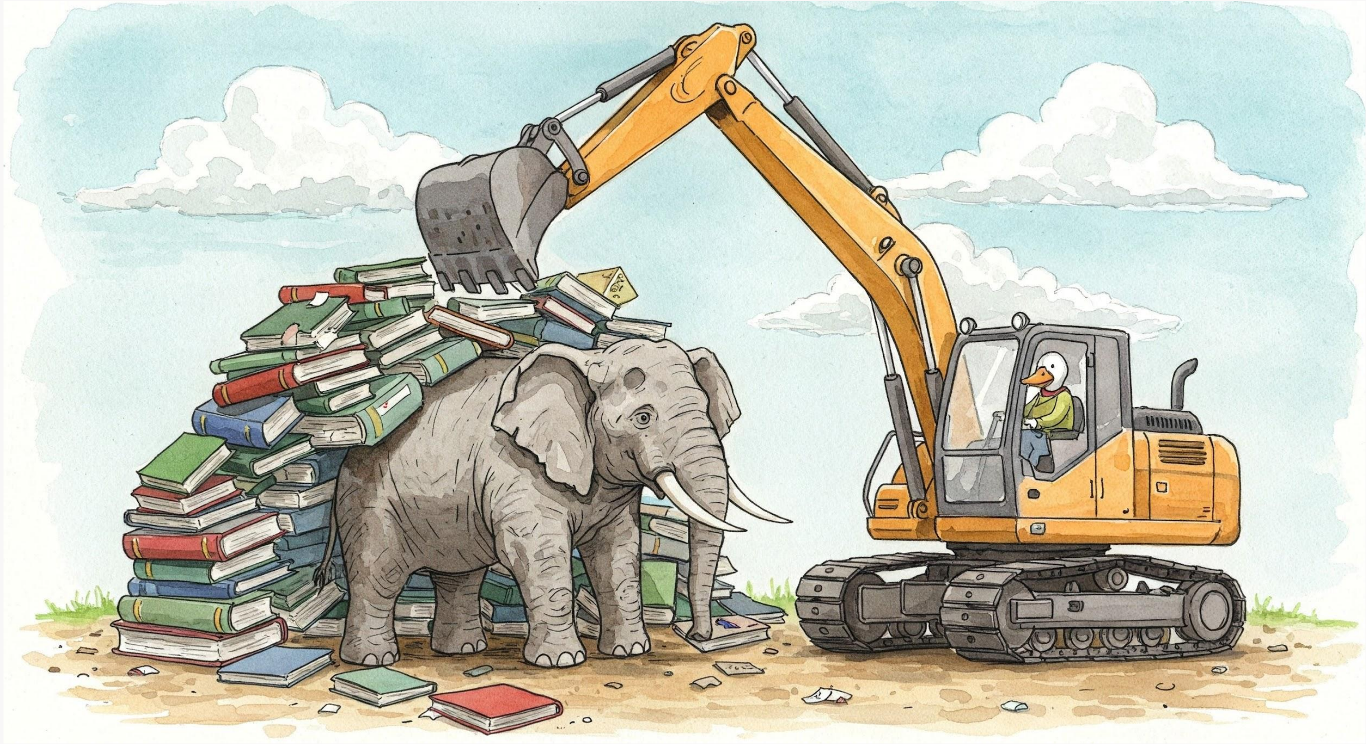
Great at transactional workloads (OLTP)

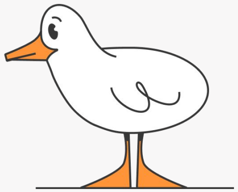


But not at analytics… (OLAP)

MotherDuck

DuckDB to the rescue

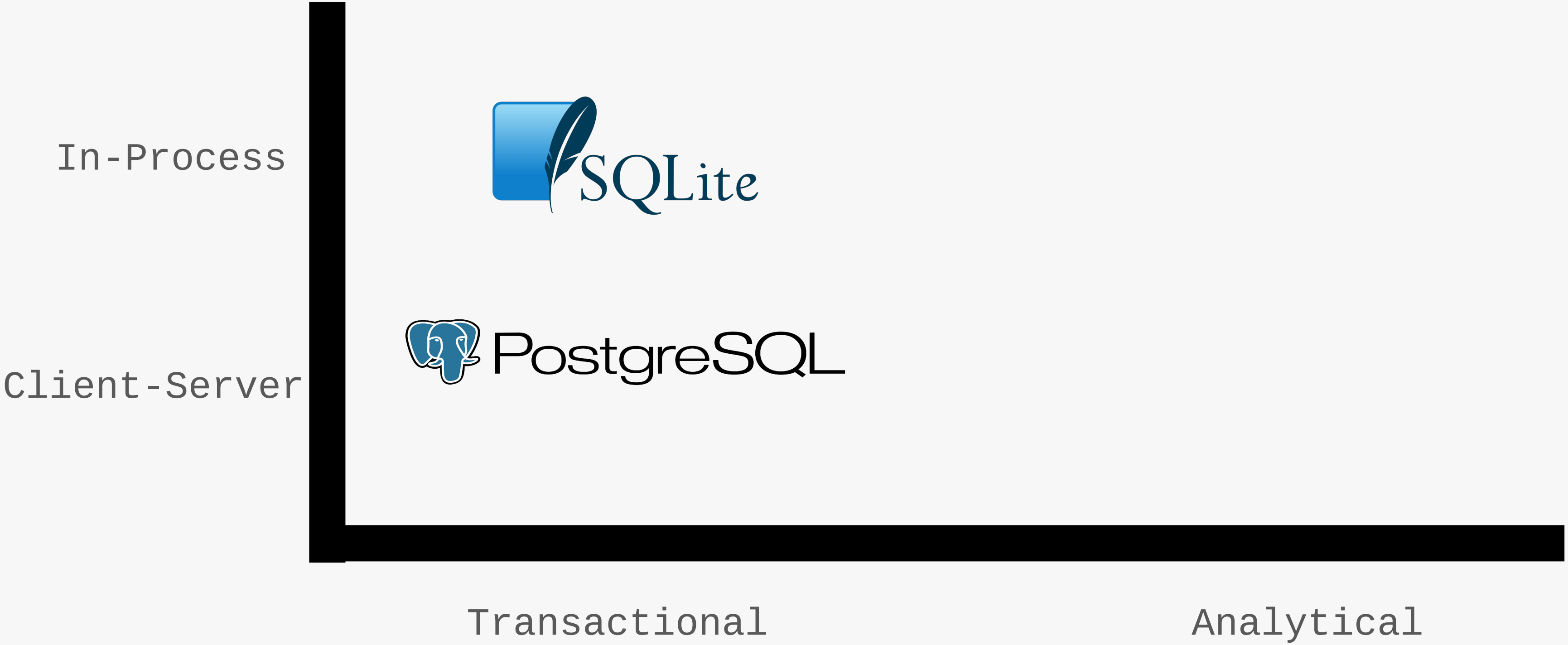




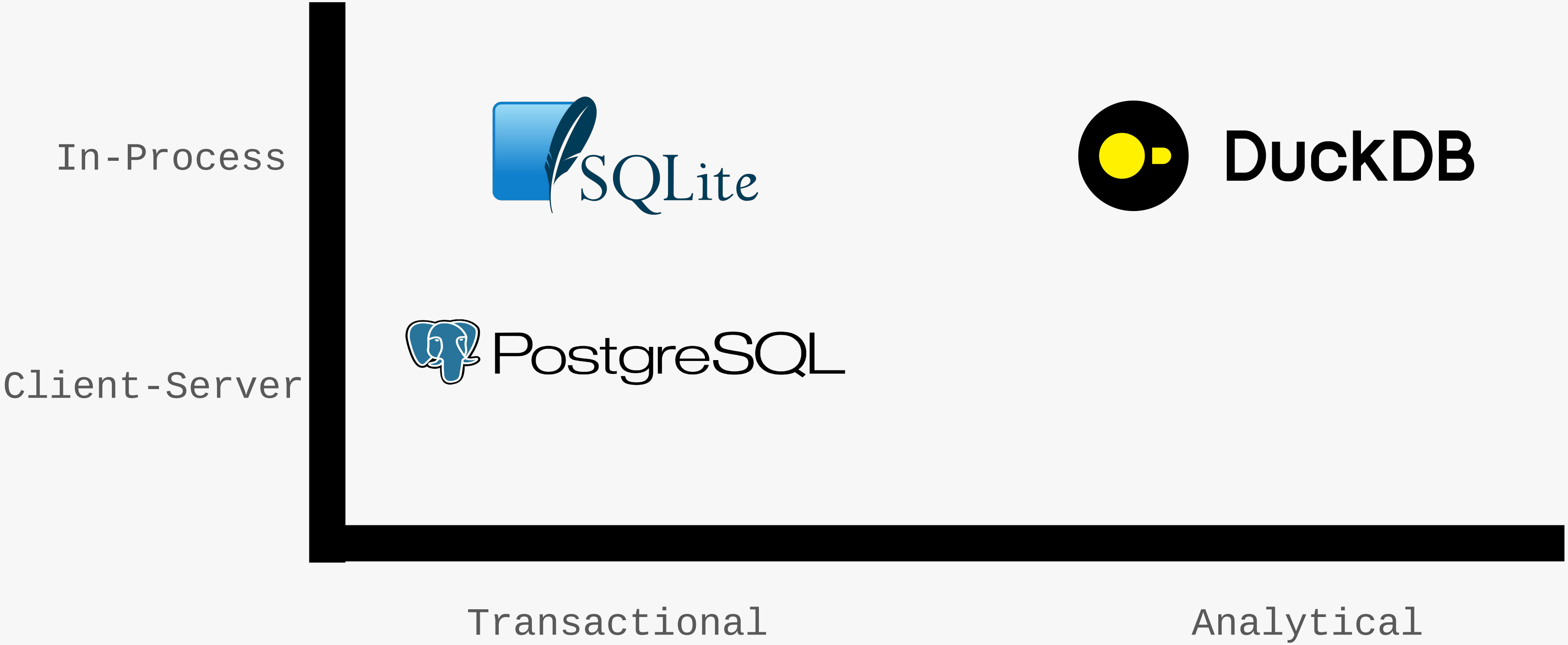
What is **DuckDB**?

Lightweight in-process SQL Analytics Engine

DuckDB is a new category of database



DuckDB is a new category of database



DuckDB

created at: CWI Research

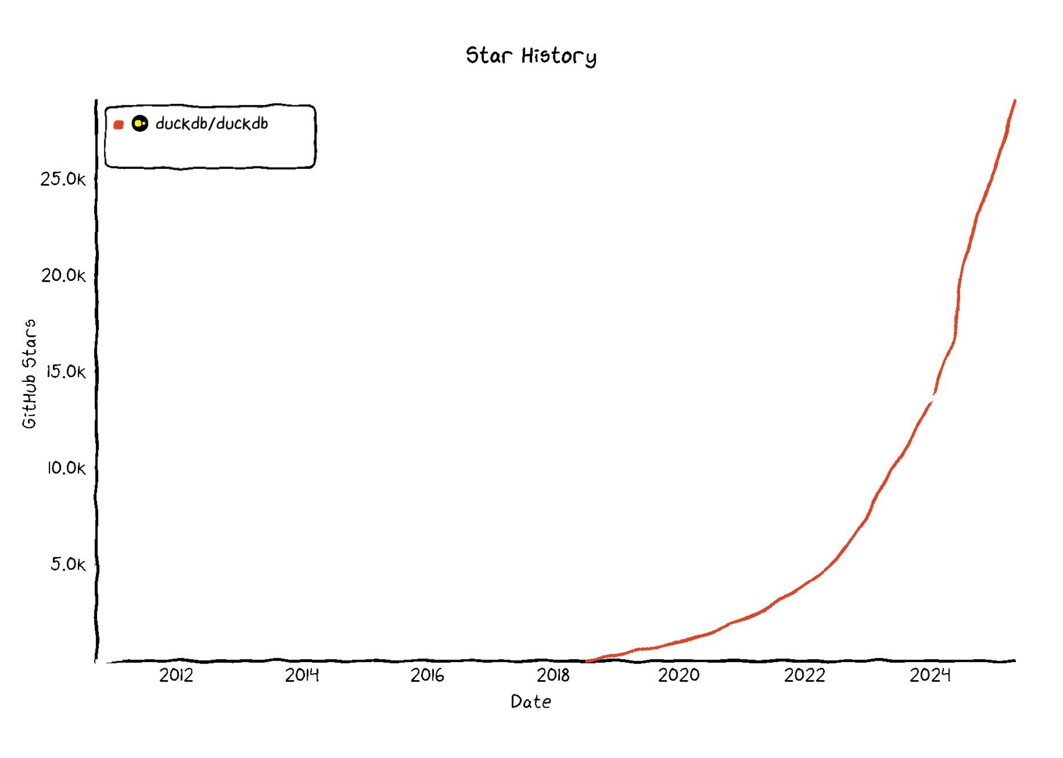
created by:

maintained by:

DuckDB Labs

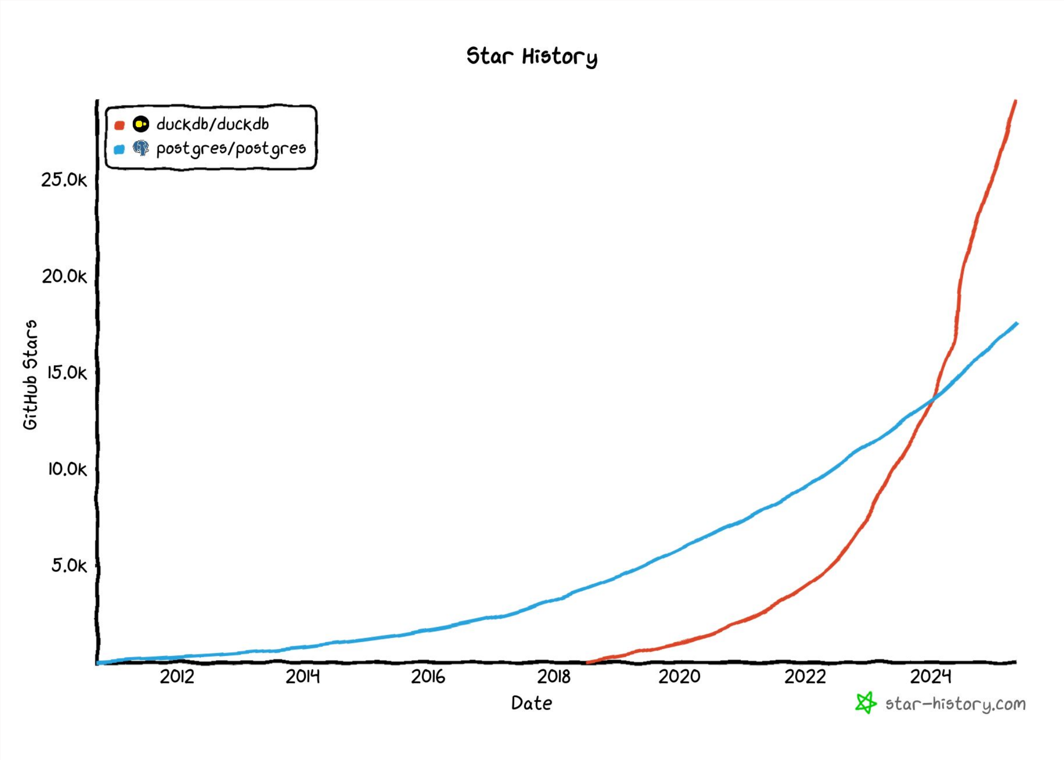
DuckDB Community & Foundation

And it’s very popular



MotherDuck

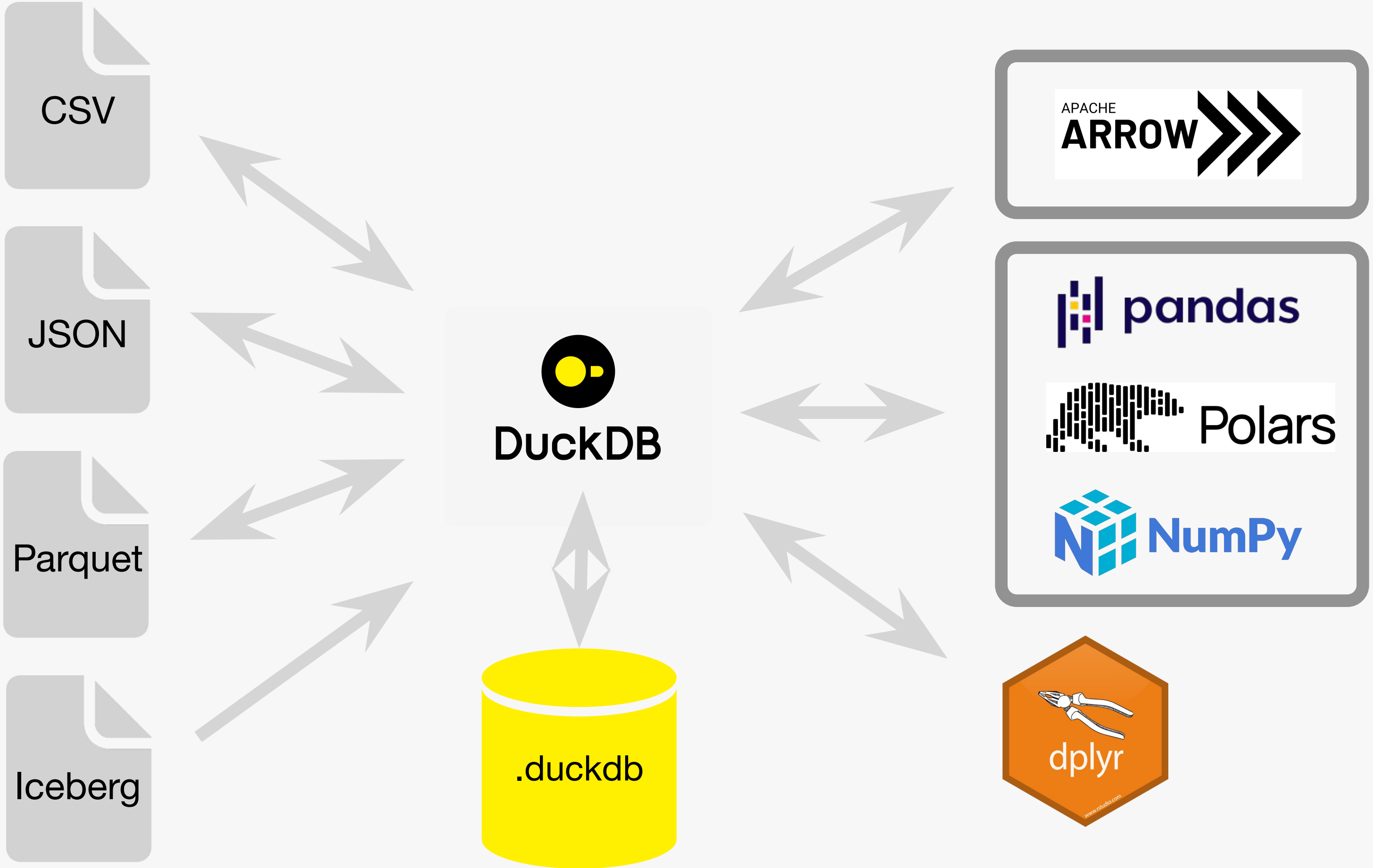
And it’s very popular



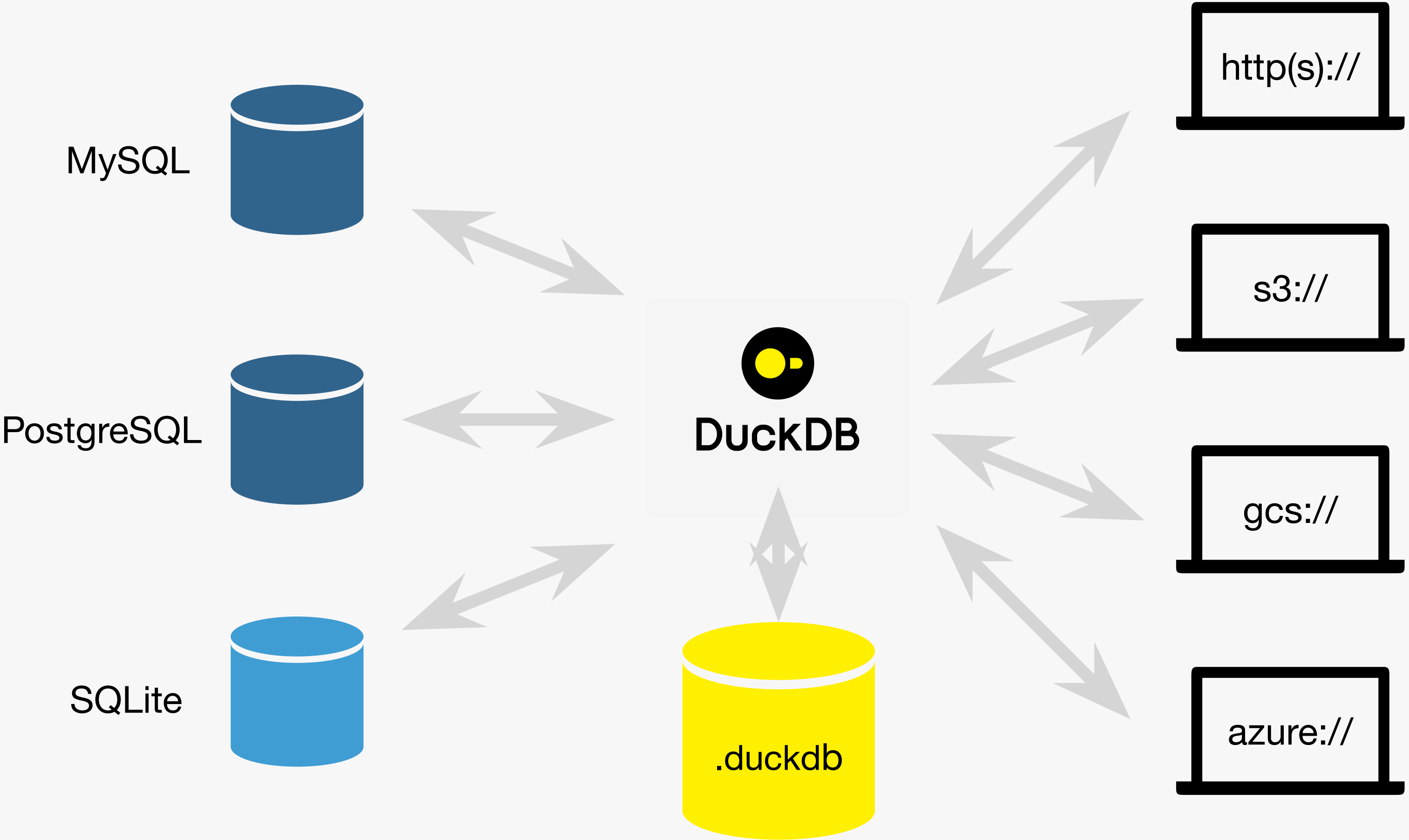
MotherDuck

Swiss army-knife for data

Input and output formats



Data sources and destinations



World’s best CSV parser

- An absurd amount of the world runs on CSV files
- An absurd amount of the world has broken / wonky CSV files
- An absurd amount of data engineering time is spent dealing with CSV file peculiarities
- Wouldn’t it be nice if they just … worked?

Multi-Hypothesis CSV Parsing

Till Döhmen
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
tilldoehmen@gmail.com

Hannes Mühleisen
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
hannes@cwi.nl

Peter Boncz
Centrum Wiskunde & Informatica
Amsterdam, The Netherlands
boncz@cwi.nl

ABSTRACT

Comma Separated Value (CSV) files are commonly used to represent data. CSV is a very simple format, yet we show that it gives rise to a surprisingly large amount of ambiguities in its parsing and interpretation. We summarize the state-of-the-art in CSV parsers, which typically make a linear series of parsing and interpretation decisions, such that any wrong decision at an earlier stage can negatively affect all downstream decisions. Since computation time is much less scarce than human time, we propose to turn CSV parsing into a ranking problem. Our quality-oriented *multi-hypothesis* CSV parsing approach generates several concurrent hypotheses about dialect, table structure, etc. and ranks these hypotheses based on quality features of the resulting table. This approach makes it possible to create an advanced CSV parser that makes many different decisions, yet keeps the overall parser code a simple plug-in infrastructure. The complex interactions between these decisions are taken care of by searching the hypothesis space rather than by having to program these many interactions in code. We show that our approach leads to better parsing results than the state of the art and facilitates the parsing of large corpora of heterogeneous CSV files.

CCS CONCEPTS

• **Information systems → Inconsistent data**;

ACM Reference format:
Till Döhmen, Hannes Mühleisen, and Peter Boncz. 2017. Multi-Hypothesis CSV Parsing. In *Proceedings of SSDBM ’17, Chicago, IL, USA, June 27-29, 2017,* 12 pages.
https://doi.org/http://dx.doi.org/10.1145/3085504.3085520

1 INTRODUCTION

Data scientists typically lose much time in importing and cleaning data, and large data repositories such as open government collections with tens of thousands of datasets remain under-exploited due to the high human cost of discovering, accessing and cleaning this data. CSV is the most commonly used data format in such repositories. The lack of explicit information on the CSV dialect,

SSDBM ’17, June 27-29, 2017, Chicago, IL, USA

ACM ISBN 978-1-4503-5282-6/17/06…$15.00
https://doi.org/http://dx.doi.org/10.1145/3085504.3085520

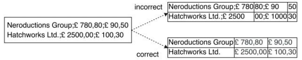


Figure 1: Ambiguous CSV file which is at risk to be parsed incorrectly, because the number of commas and the number of semi-colons per row are the same.

the table structure, and data types makes proper parsing tedious and error-prone.

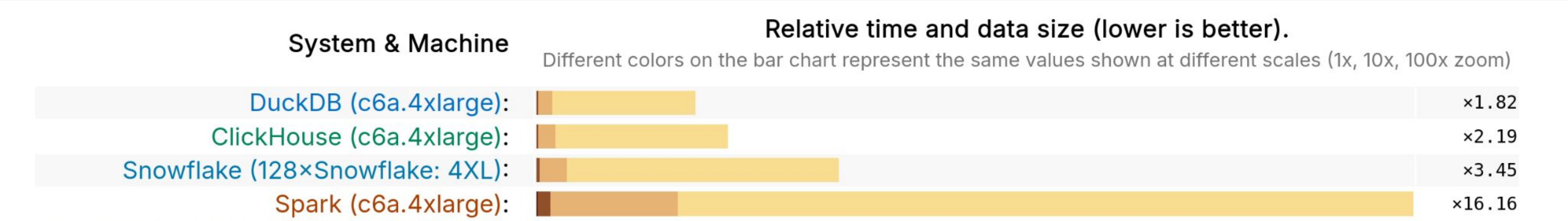
Tools currently popular among data scientists, such as `R` and `Python` offer robust CSV parsing libraries, which try to address parsing of messy CSV files with a number of practical heuristics. These libraries make a linear sequence of parsing and interpretation decisions, such that any wrong decision at an earlier stage (e.g. determining the separator character) will negatively affect all downstream decisions. Interlinking different parsing steps (backtracking on prior decisions) is not done, because if all parsing decisions affect each other, the parsing code becomes very complex (code size would need to grow quadratically in the amount of decisions or even worse).

Since CPU-cycles are currently plentiful but human time is not, this research pursues an approach where CSV parsing becomes an computerized search problem. Our quality-oriented CSV parsing approach generates several concurrent hypotheses about dialect, table structure, etc. and in the end ranks these hypotheses based on quality features of the resulting table, such that the top-1 would be the automatic parsing result, or a top-K of parsed tables could be presented to the user. A high absolute score from the quality function can also be used to automatically parse large amounts of files. Only ambigous cases would be presented to a user. This can strongly reduce human data interpretation effort.

This very practical problem touches on various areas of related work. In the extended version of this paper [6], we survey the state-of-the-art on this topic, which covers areas such as computer-assisted data-cleaning (*data-wrangling*), table-interpretation (e.g. on the web), automatic list extraction and even automated semantic (web) enrichment; covered more briefly in the related work Section 5.

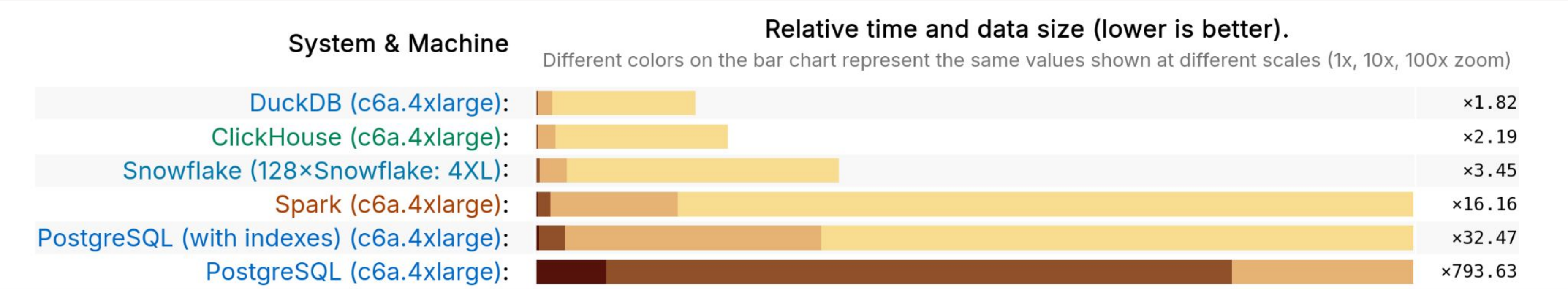
Outline. In Section 2 we explaine CSV parsing problem by example, and introduce our multi-hypothesis parsing framework in Section 3. We demonstrate the improved parsing quality of our approach with computed quality metrics on the full *data.gov.uk* dataset collection, as well as on a sample of this collection using human ground truth in Section 4. We summarize related work in Section 5 and describe next steps in Section 6 before concluding in Section 7.

And it's fast!



MotherDuck

And it's fast!



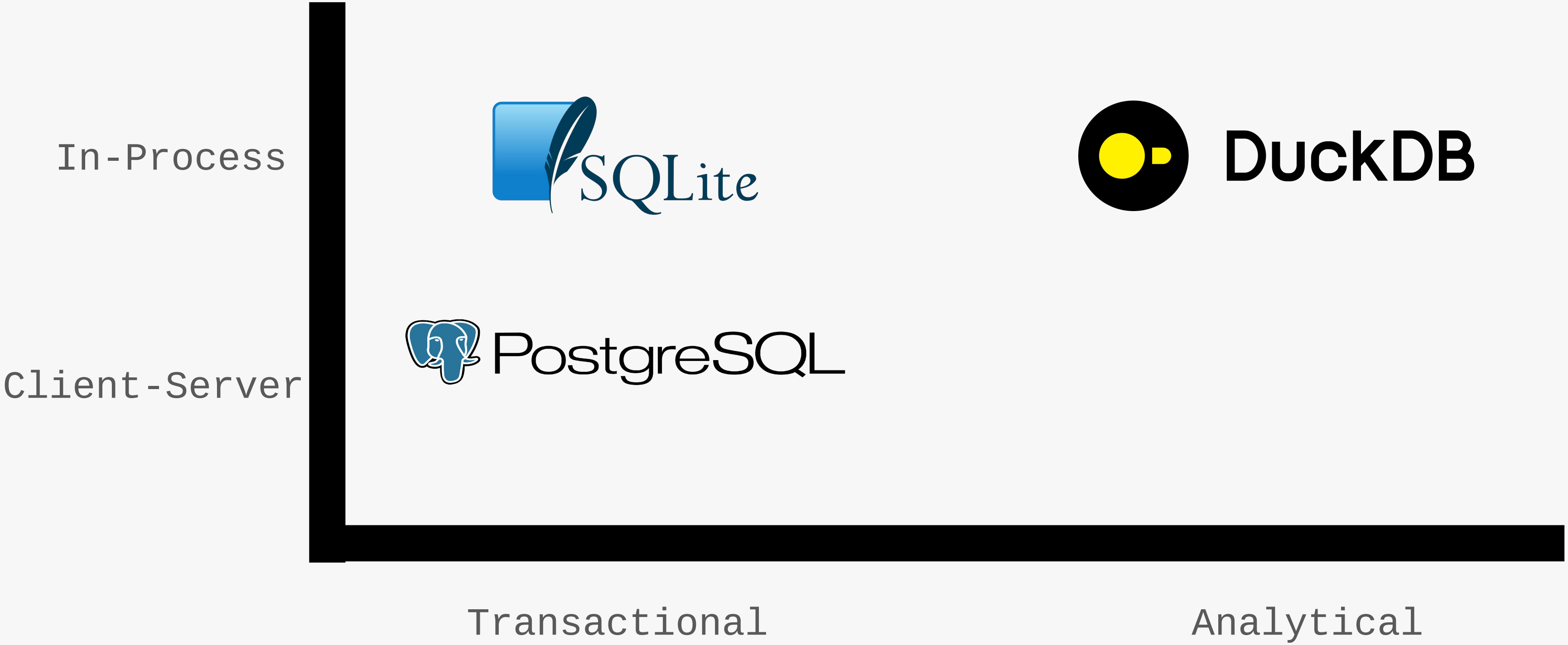
MotherDuck

A small recap

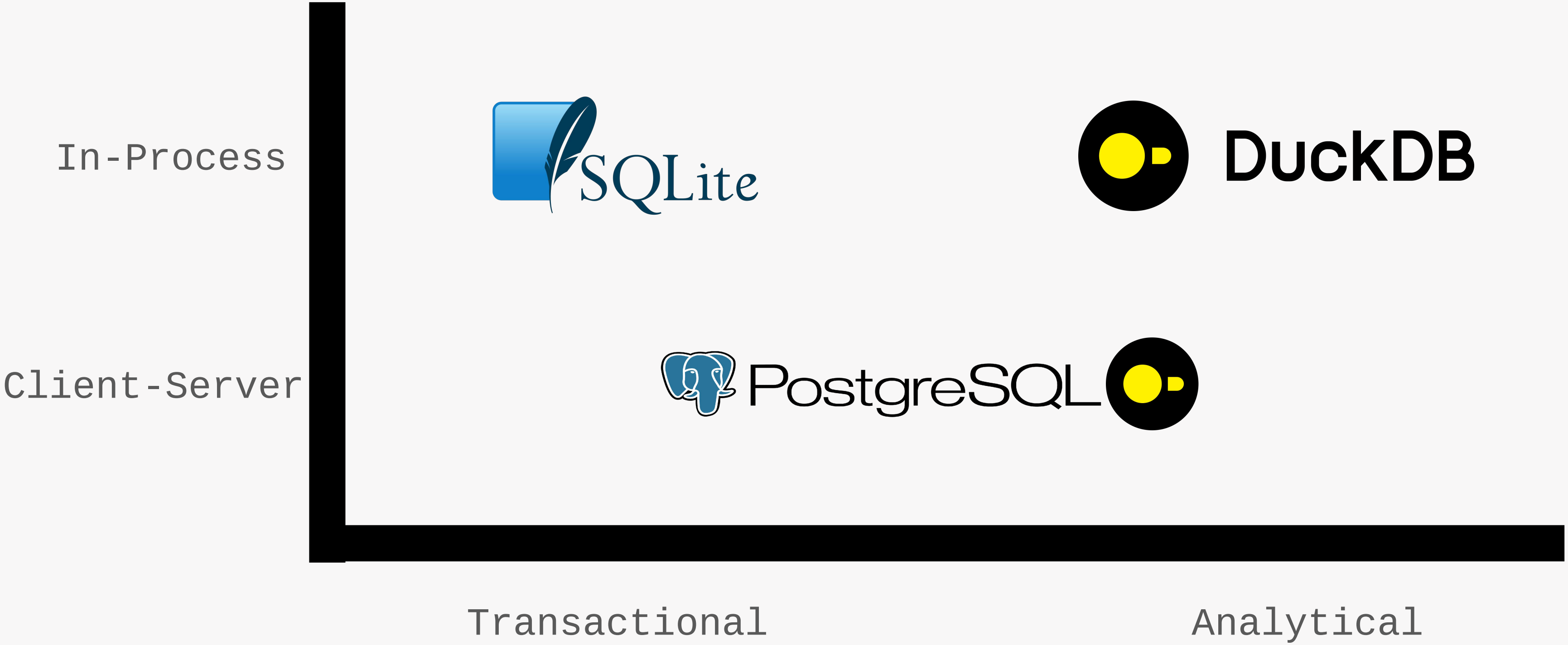
Now, we have two great databases

- Postgres for transactional workloads
- DuckDB for analytical workloads

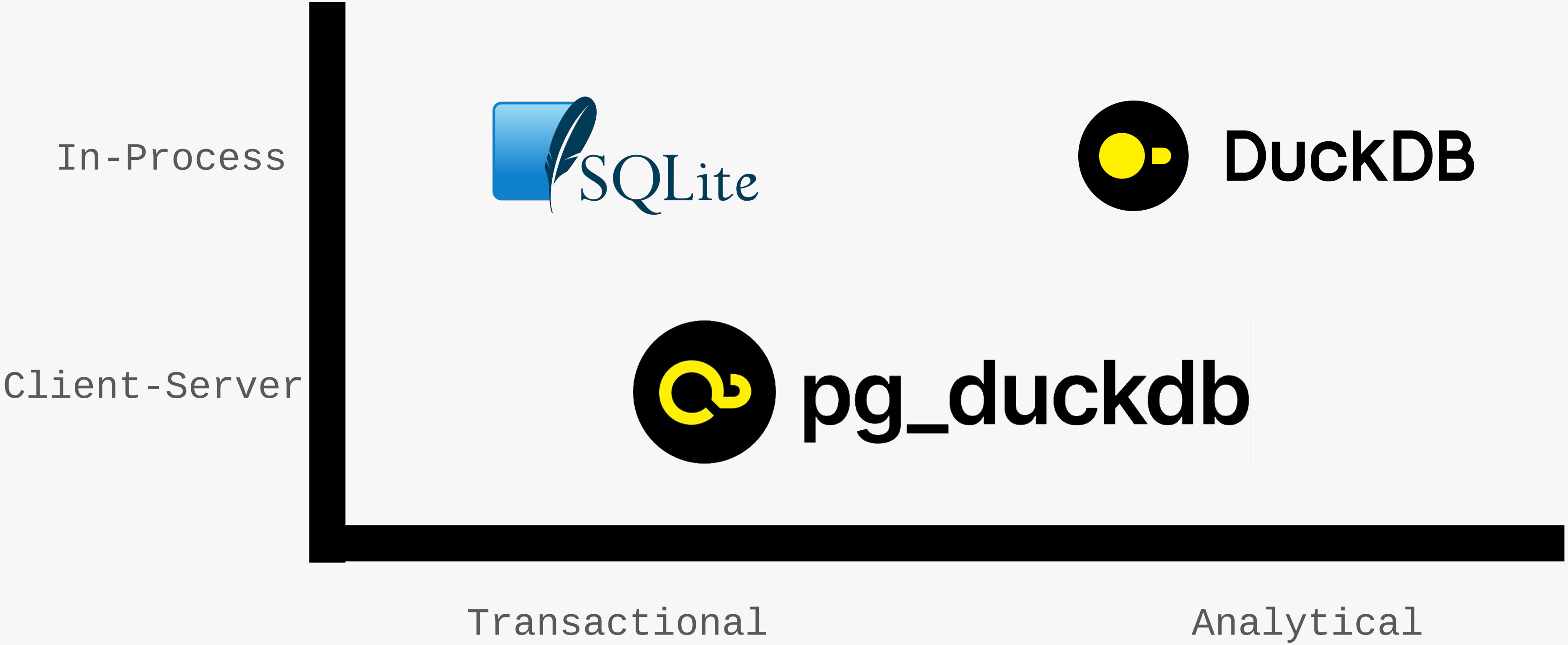
DuckDB is a new category of database



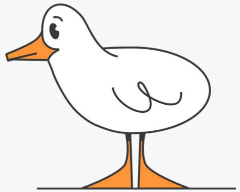
PG Analytics with DuckDB



PG Analytics with DuckDB



What does that look like?



What does that look like?

What does that look like?

Ducks & Elephants are different species

1989	vs	2019
C	vs	C++
elog(ERROR, …)	vs	exceptions
processes	vs	threads

So we did lots of work

Now this is where we're at

How does pg_duckdb work?

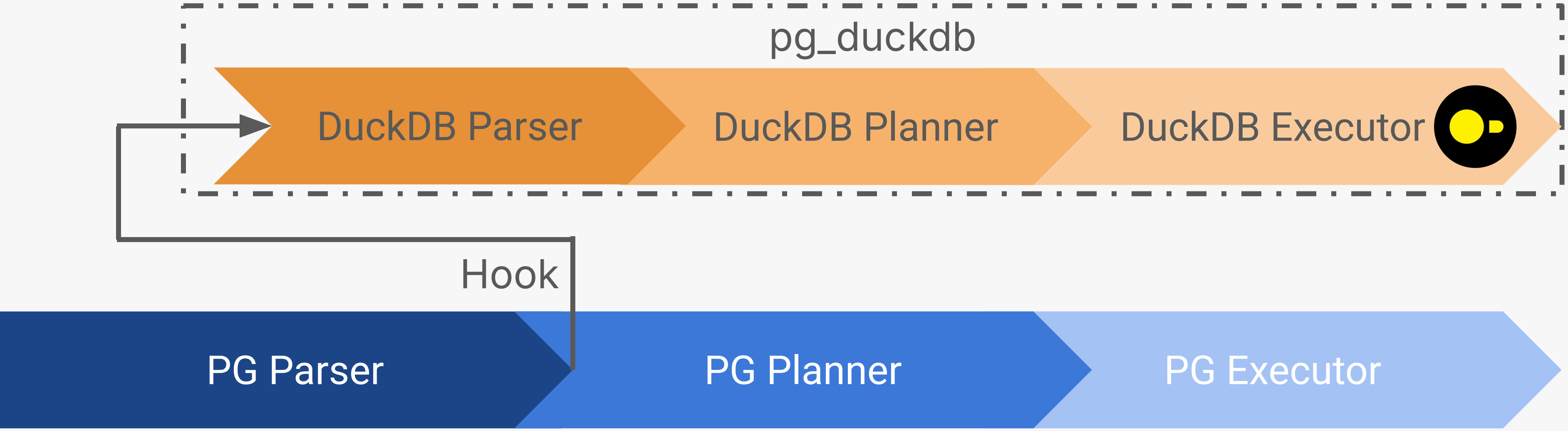
Transforms the SQL query string to syntax tree

Determines the most efficient way to execute it

Executes the plan

Simplified query processing in Postgres

pg_duckdb “steals” the query

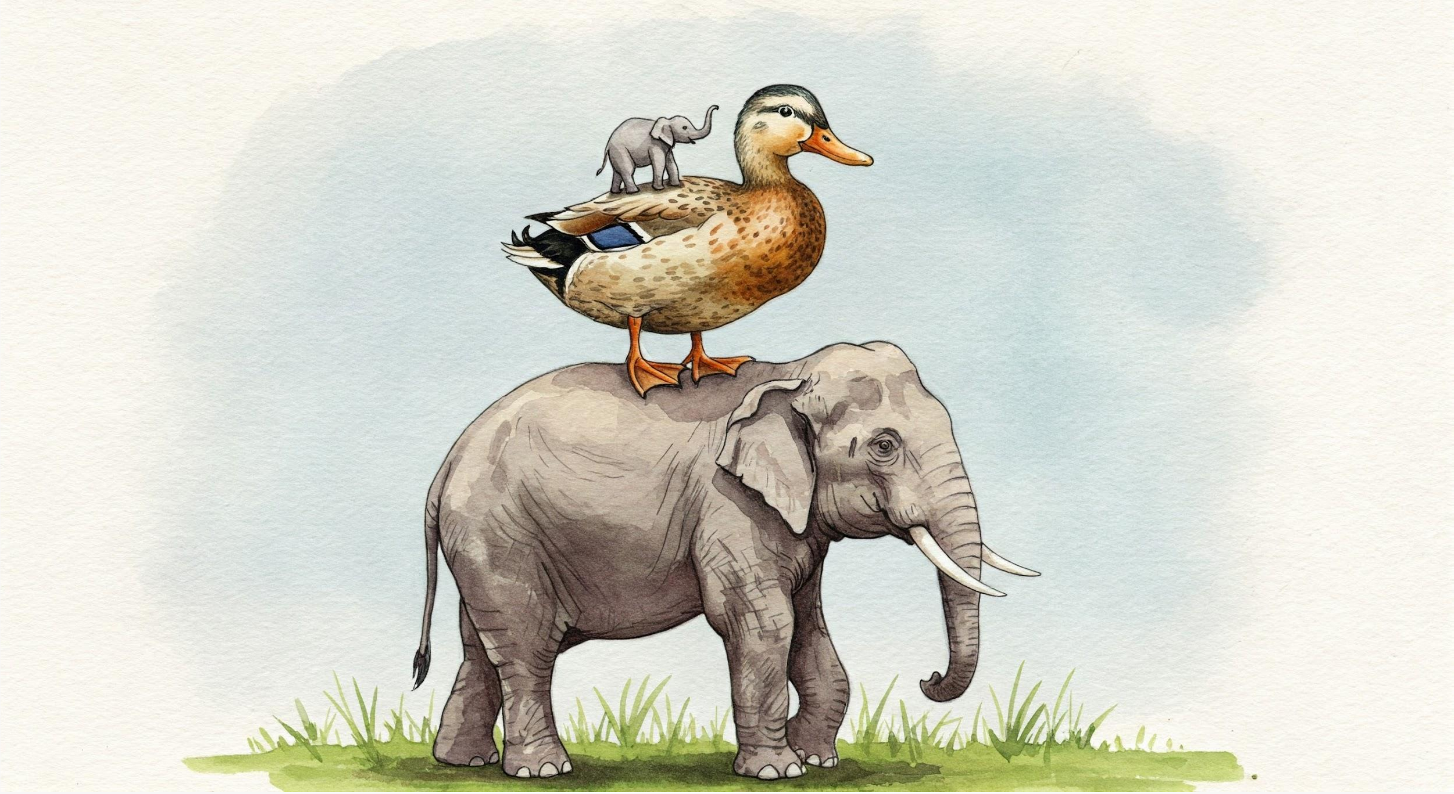


Transforms the SQL query string to syntax tree

Determines the most efficient way to execute it

Executes the plan

pg_duckdb can read PG data



What can it do?

What can it do?

1. Use DuckDB engine on Postgres tables

What can it do?

1. Use DuckDB engine on Postgres tables
2. Read/write data in blob storage

What can it do?

1. Use DuckDB engine on Postgres tables
2. Read/write data in blob storage
3. Offload analytics to MotherDuck

DuckDB engine on Postgres tables

DuckDB engine on Postgres tables

Very simple:

```
SET duckdb.force_execution = true;
```

But is it fast???

It depends…

But sometimes yes!

ClickBench results

		PostgreSQL (with indexes) (c6a.4xlarge)	pg_duckdb (with indexes) (c6a.4xlarge)
☑	Q11.	765.305s (×2.11)	**361.887s (×1.00)**
☑	Q12.	2.928s (×1.22)	**2.389s (×1.00)**
☑	Q13.	11.955s (×2.01)	**5.941s (×1.00)**
☑	Q14.	258.984s (×1.03)	**251.156s (×1.00)**
☑	Q15.	15.302s (×2.65)	**5.759s (×1.00)**
☑	Q16.	15.244s (×1.82)	**8.355s (×1.00)**

One extreme example

One extreme example

1. **Set up TPC-DS with 10GB and no indexes**

One extreme example

1. Set up TPC-DS with 10GB and no indexes
2. Run Q1 -> ⌛⌛⌛ wait 10 minutes and give up

One extreme example

1. Set up TPC-DS with 10GB and no indexes
2. Run Q1 -> ⌛⌛⌛ wait 10 minutes and give up
3. SET duckdb.force_execution = true;

One extreme example

1. Set up TPC-DS with 10GB and no indexes
2. Run Q1 -> ⌛⌛⌛ wait 10 minutes and give up
3. SET duckdb.force_execution = true;
4. Run Q1 -> done in 450ms!

One extreme example

1. Set up TPC-DS with 10GB and no indexes
2. Run Q1 -> ⌛⌛⌛ wait 10 minutes and give up
3. SET duckdb.force_execution = true;
4. Run Q1 -> done in 450ms!
5. Easiest query optimization ever 🎉

But how?

Morsel-Driven Parallelism

Morsel-Driven Parallelism: A NUMA-Aware Query Evaluation Framework for the Many-Core Age

Viktor Leis* Peter Boncz† Alfons Kemper* Thomas Neumann*

* Technische Universität München † CWI

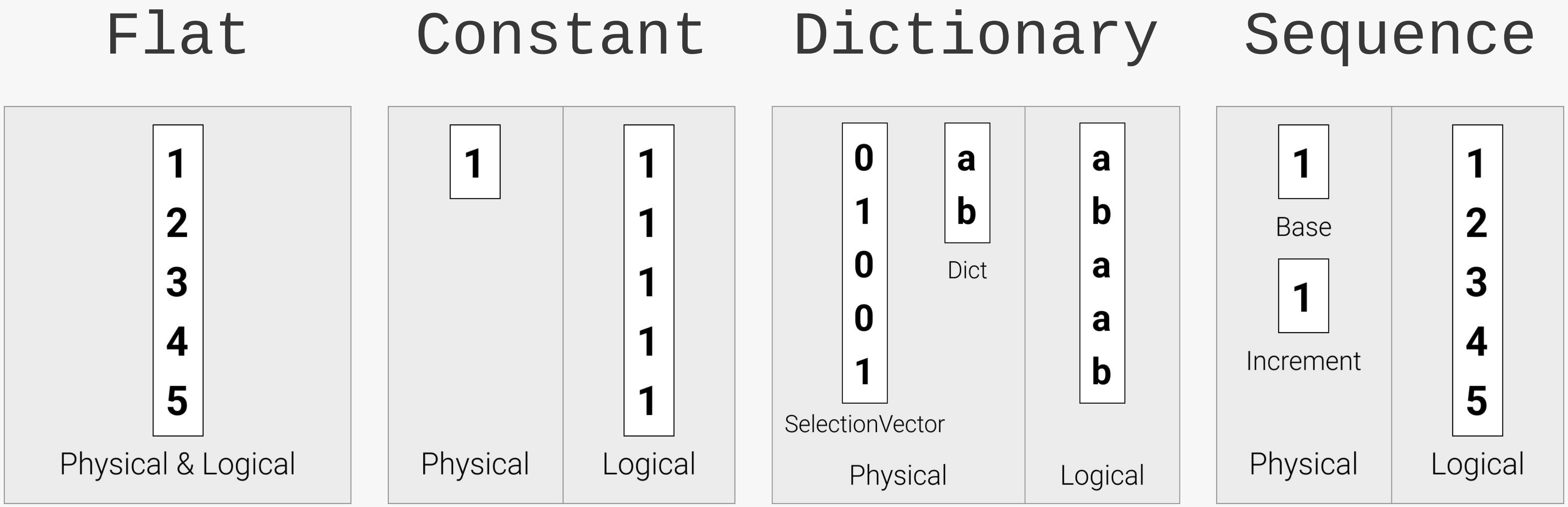
* {leis,kemper,neumann}@in.tum.de † p.boncz@cwi.nl

Source: https://db.in.tum.de/~leis/papers/morsels.pdf

Morsel-Driven Parallelism

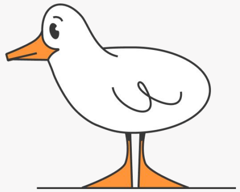
- **Morsel-driven query execution** is a new parallel query evaluation framework that fundamentally differs from the traditional Volcano model in that it distributes work between threads dynamically using work-stealing. This prevents unused CPU resources due to load imbalances, and allows for *elasticity*, i.e., CPU resources can be reassigned between different queries at any time.

Execution on compressed data



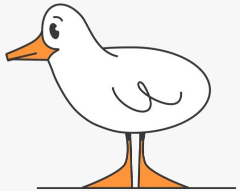
Source: https://duckdb.org/docs/stable/internals/vector

Read from blob storage



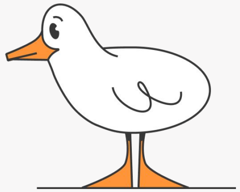
```sql
SELECT *
FROM read_parquet('s3://<my-bucket>/netflix_daily_top_10.parquet')
LIMIT 5;
```

Read from blob storage



```sql
SELECT r['Title'], max(r['Days In Top 10'])::int as MaxDaysInTop10
FROM read_parquet('s3://<my-bucket>/netflix_daily_top_10.parquet') r
WHERE r['Type'] = 'TV Show'
GROUP BY r['Title']
ORDER BY MaxDaysInTop10 DESC
LIMIT 5;
```

Read from blob storage



```
SELECT r['Title'], max(r['Days In Top 10'])::int as MaxDaysInTop10
FROM read_parquet('s3://<my-bucket>/netflix_daily_top_10.parquet') r
WHERE r['Type'] = 'TV Show'
GROUP BY r['Title']
ORDER BY MaxDaysInTop10 DESC
LIMIT 5;
```

MotherDuck

What it looks like in DuckDB

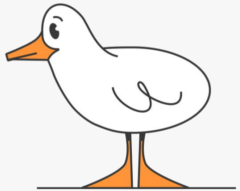
```
SELECT Title, max("Days In Top 10")::int as MaxDaysInTop10
FROM 's3://<my-bucket>/netflix_daily_top_10.parquet'
WHERE Type = 'TV Show'
GROUP BY Title
ORDER BY MaxDaysInTop10 DESC
LIMIT 5;
```

MotherDuck

What it looks like in DuckDB

```
SELECT Title, max("Days In Top 10")::int as MaxDaysInTop10
FROM 's3://<my-bucket>/netflix_daily_top_10.parquet'
WHERE Type = 'TV Show'
GROUP BY Title
ORDER BY MaxDaysInTop10 DESC
LIMIT 5;
```

Read from blob storage



```
SELECT r['Title'], max(r['Days In Top 10'])::int as MaxDaysInTop10
FROM read_parquet('s3://<my-bucket>/netflix_daily_top_10.parquet') r
WHERE r['Type'] = 'TV Show'
GROUP BY r['Title']
ORDER BY MaxDaysInTop10 DESC
LIMIT 5;
```

MotherDuck

Making Postgres behave like DuckDB

```
SELECT Title, max("Days In Top 10")::int as MaxDaysInTop10
FROM 's3://<my-bucket>/netflix_daily_top_10.parquet'
WHERE Type = 'TV Show'
GROUP BY Title
ORDER BY MaxDaysInTop10 DESC
LIMIT 5;
```

MotherDuck

Making Postgres behave like DuckDB

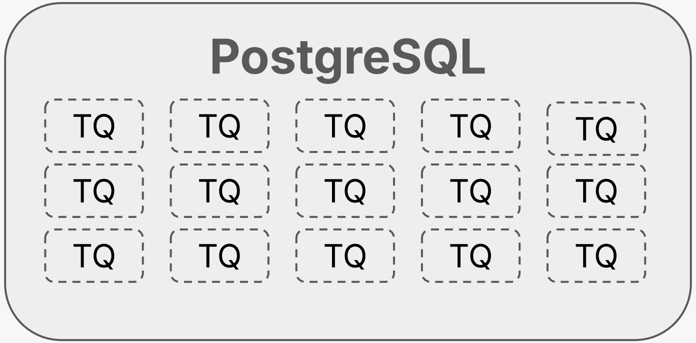
```
SELECT * FROM duckdb.query($$
SELECT Title, max("Days In Top 10")::int as MaxDaysInTop10
FROM 's3://<my-bucket>/netflix_daily_top_10.parquet'
WHERE Type = 'TV Show'
GROUP BY Title
ORDER BY MaxDaysInTop10 DESC
LIMIT 5
$$);
```

MotherDuck

Making Postgres behave like DuckDB

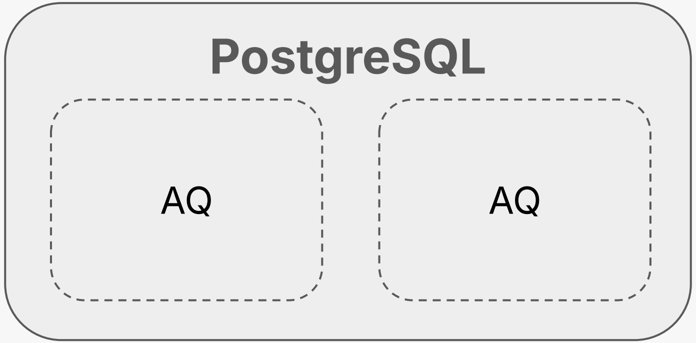
```
SELECT * FROM duckdb.query($$
FROM 's3://<my-bucket>/netflix_daily_top_10.parquet'
LIMIT 5
$$);
```

A few words about resources



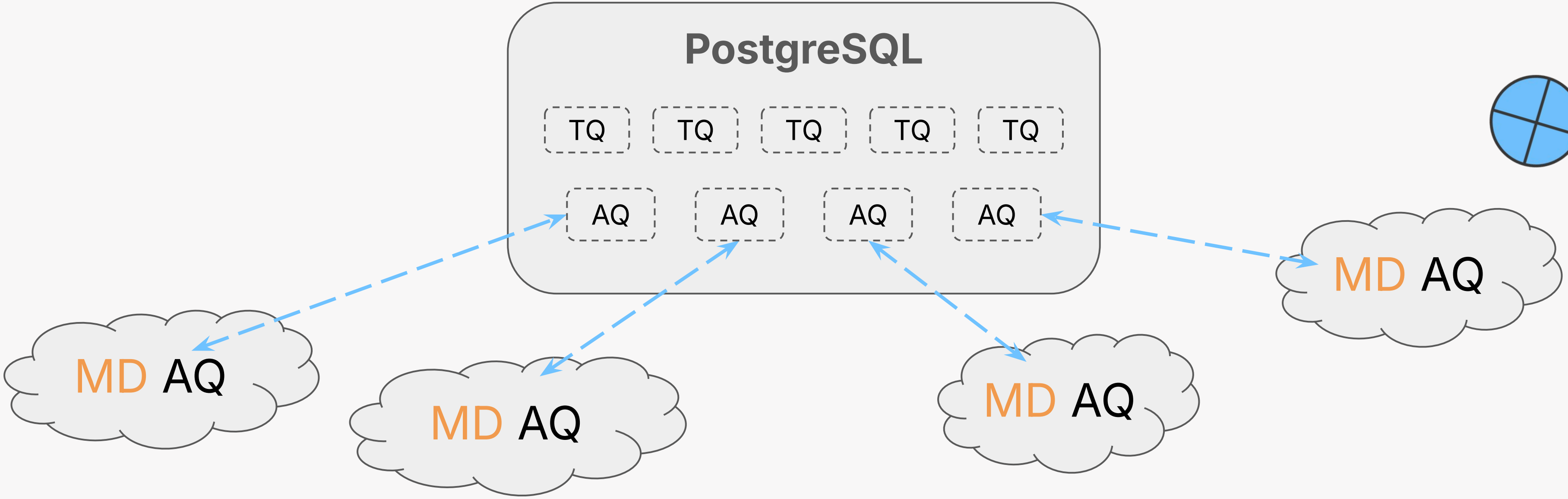
Lite transactional queries

A few words about resources



Analytics

A few words about resources



MotherDuck on-demand resources

Copy data to MotherDuck

```
CREATE TABLE hacker_news_motherduck_archive
USING duckdb AS
SELECT * FROM hacker_news;
```

Query it like normal

```sql
SELECT
    EXTRACT(YEAR FROM timestamp) AS year,
    EXTRACT(MONTH FROM timestamp) AS month,
    COUNT(*) AS keyword_mentions
FROM hacker_news_motherduck_archive
WHERE
    (title LIKE '%duckdb%' OR text LIKE '%duckdb%')
GROUP BY year, month
ORDER BY year ASC, month ASC;
```

Combine with PG data

```sql
SELECT
    EXTRACT(YEAR FROM timestamp) AS year,
    EXTRACT(MONTH FROM timestamp) AS month,
    COUNT(*) AS keyword_mentions
FROM (
  SELECT * FROM hacker_news_last_month UNION ALL
  SELECT * FROM hacker_news_motherduck_archive)
WHERE
    (title LIKE '%duckdb%' OR text LIKE '%duckdb%')
GROUP BY year, month
ORDER BY year ASC, month ASC;
```

MotherDuck

But is it fast???

For analytics: YES!

pg_duckdb (MotherDuck enabled) (Motherduck: Jumbo): ×1.19

PostgreSQL (with indexes) (c6a.4xlarge): ×23.69

Detailed Comparison

	pg_duckdb (MotherDuck enabled) (Motherduck: Jumbo)	PostgreSQL (with indexes) (c6a.4xlarge)
Load time:	**119s (×1.00)**	10357s (×87.32)
Data size:	**24.50 GiB (×1.00)**	115.84 GiB (×4.73)
Q0.	**0.075s (×1.00)**	1.834s (×21.76)
Q1.	**0.110s (×1.00)**	0.861s (×7.24)
Q2.	**0.108s (×1.00)**	240.433s (×2038.20)
Q3.	**0.112s (×1.00)**	3.041s (×25.05)
Q4.	**0.212s (×1.00)**	7.480s (×33.79)
Q5.	**0.257s (×1.00)**	7.622s (×28.60)
Q6.	0.084s (×2.56)	0.009s (×0.52)
Q7.	**0.072s (×1.00)**	0.872s (×10.71)
Q8.	**0.260s (×1.00)**	9.092s (×33.66)
Q9.	**0.323s (×1.00)**	270.081s (×810.94)
Q10.	**0.131s (×1.00)**	3.883s (×27.52)
Q11.	**0.120s (×1.00)**	765.305s (×5881.66)

Again… But how?

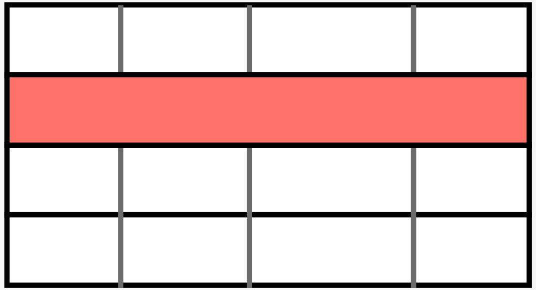
For analytics: YES!

Detailed Comparison

	pg_duckdb (MotherDuck enabled) (Motherduck: Jumbo)	PostgreSQL (with indexes) (c6a.4xlarge)
Load time:	[illegible]	[illegible]
Data size:	24.50 GiB (×1.00)	115.84 GiB (×4.73)
Q0.	[illegible]	[illegible]
Q1.	0.110s (×1.00)	0.861s (×7.24)
Q2.	0.108s (×1.00)	240.433s (×2038.20)
Q3.	0.112s (×1.00)	3.041s (×25.05)
Q4.	0.212s (×1.00)	7.480s (×33.79)
Q5.	0.257s (×1.00)	7.622s (×28.60)
Q6.	0.084s (×2.56)	0.009s (×0.52)
Q7.	0.072s (×1.00)	0.872s (×10.71)
Q8.	0.260s (×1.00)	9.092s (×33.66)
Q9.	0.323s (×1.00)	270.081s (×810.94)
Q10.	0.131s (×1.00)	3.883s (×27.52)
Q11.	0.120s (×1.00)	765.305s (×5881.66)

Postgres storage format

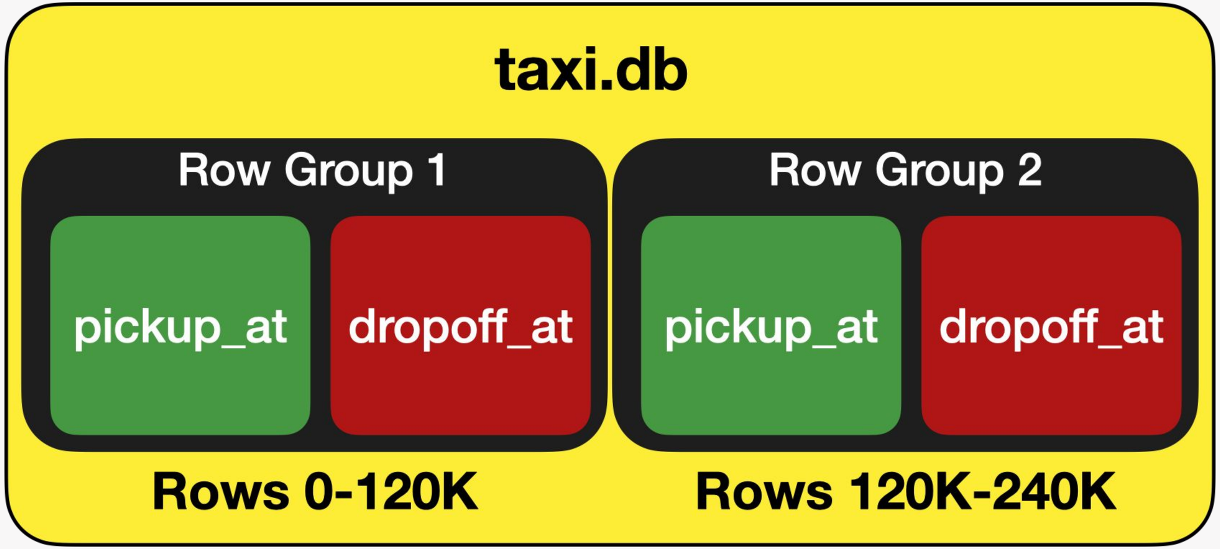
Row-based (tuples)



Optimized for:

* low memory footprint
* transactional workloads

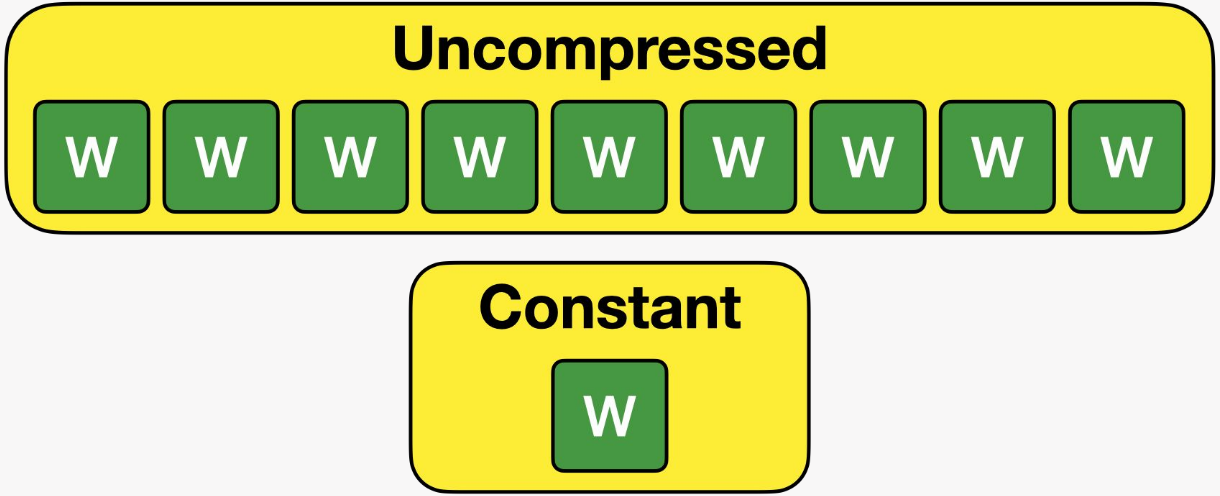
DuckDB storage format



Source: https://duckdb.org/2022/10/28/lightweight-compression.html

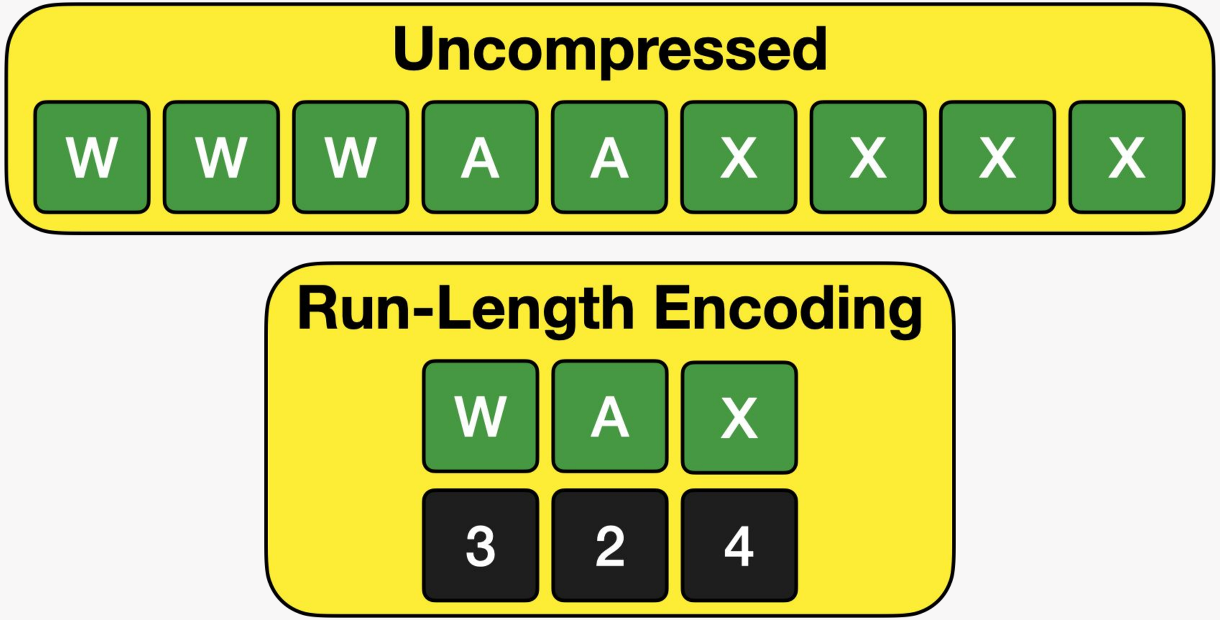
… with lightweight compression

Constant vectors



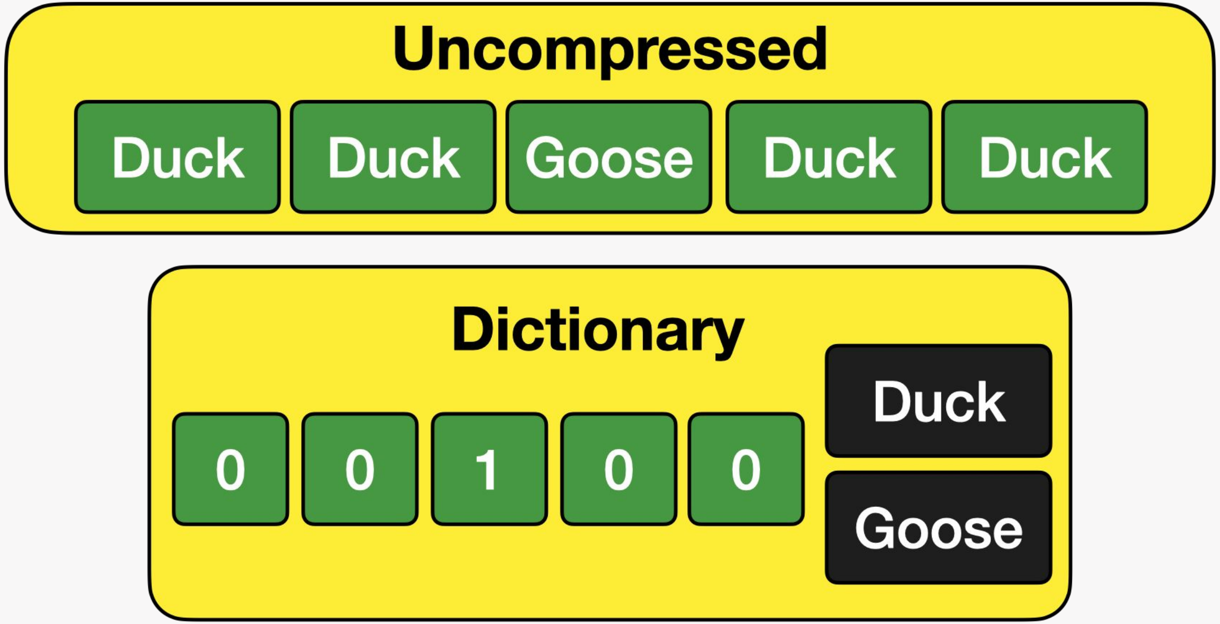
Source: https://duckdb.org/2022/10/28/lightweight-compression.html

Run-Length Encoding (RLE)



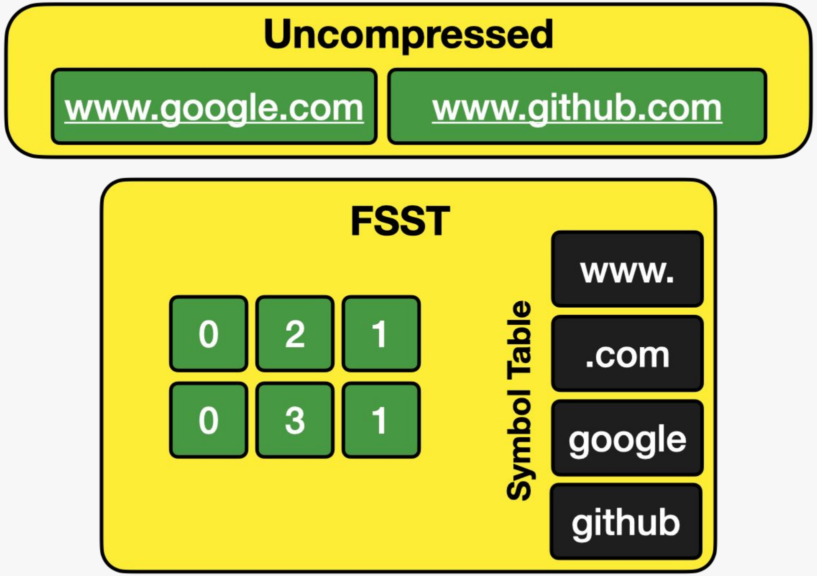
Source: https://duckdb.org/2022/10/28/lightweight-compression.html

Dictionary Encoding



Source: https://duckdb.org/2022/10/28/lightweight-compression.html

Fast Static Symbol Table



Source: https://duckdb.org/2022/10/28/lightweight-compression.html

To re-iterate

1. Use DuckDB engine to speed up existing queries
2. Blob storage integration
3. Offload analytics to MotherDuck for even more speed

And version 1.0 is out!!! 🎉🎉🎉

pg_duckdb v1.0.0 Latest

Compare ▾ Convert to draft

JelteF released this last month · 12 commits to main since this release · v1.0.0 · fad000f

± Commits

The 1.0 release is finally here! A ton of features were added, and performance improved immensely, and of course lots of fixes... And that means pg_duckdb is now ready for production!

Please try it

- MIT licensed
- github.com/duckdb/pg_duckdb
- motherduck.com/blog/pg-duckdb-release
- Feedback welcome