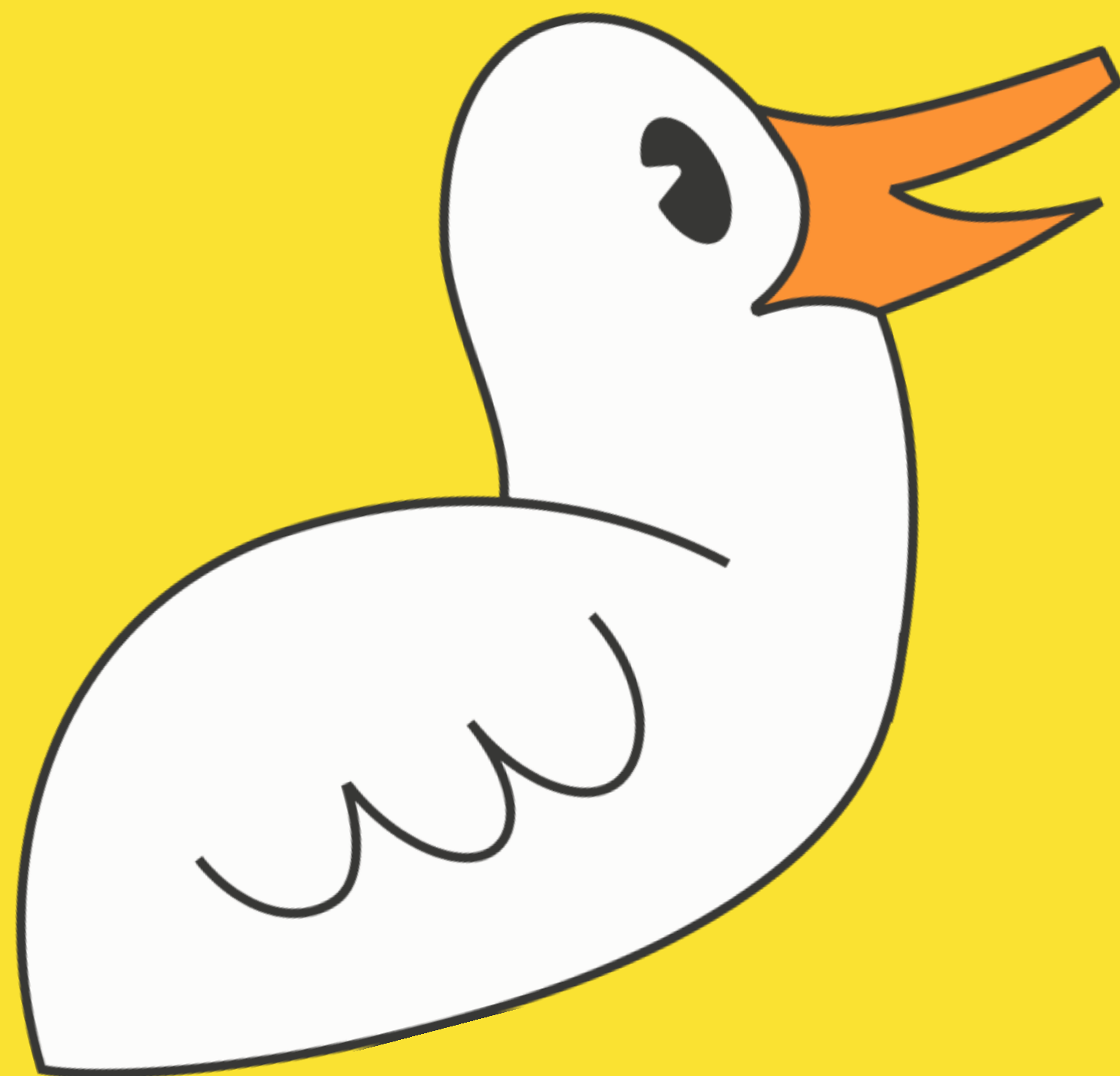
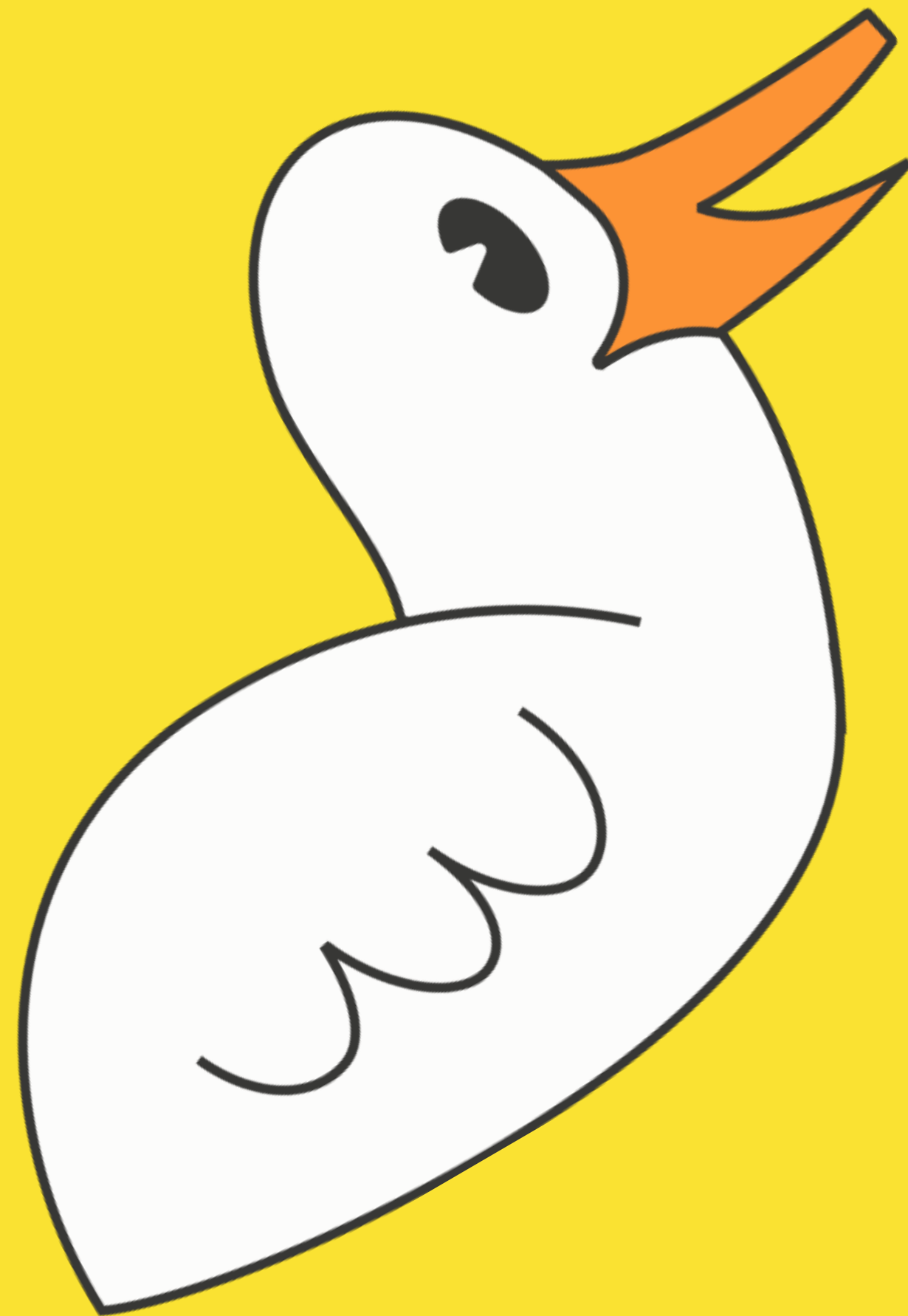


QUACK, QUACK,
QUACK, QUACK





If a Duck Quacks
In the Forest,
Should You Care?!?

#DuckTalk
#DuckPost
#DuckPoint

Ryan Boyd
Co-founder @ MotherDuck
@ryguyrg

THIS PRESENTATION IS NOT

A complete tutorial on DuckDB

THIS PRESENTATION DUCKING IS

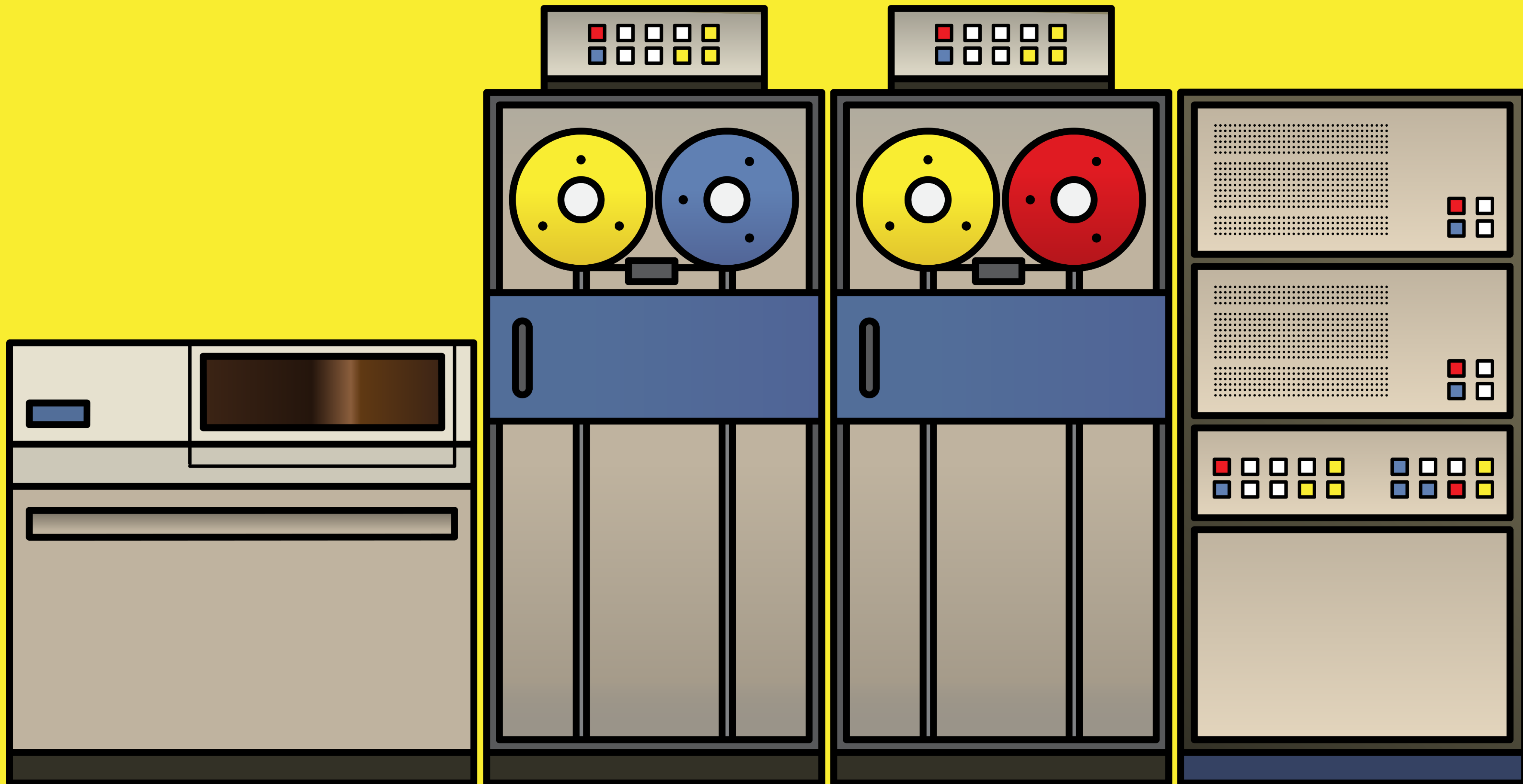
An explanation of the industry context in which DuckDB came to life

A discussion on OSS DuckDB and why it's awesome

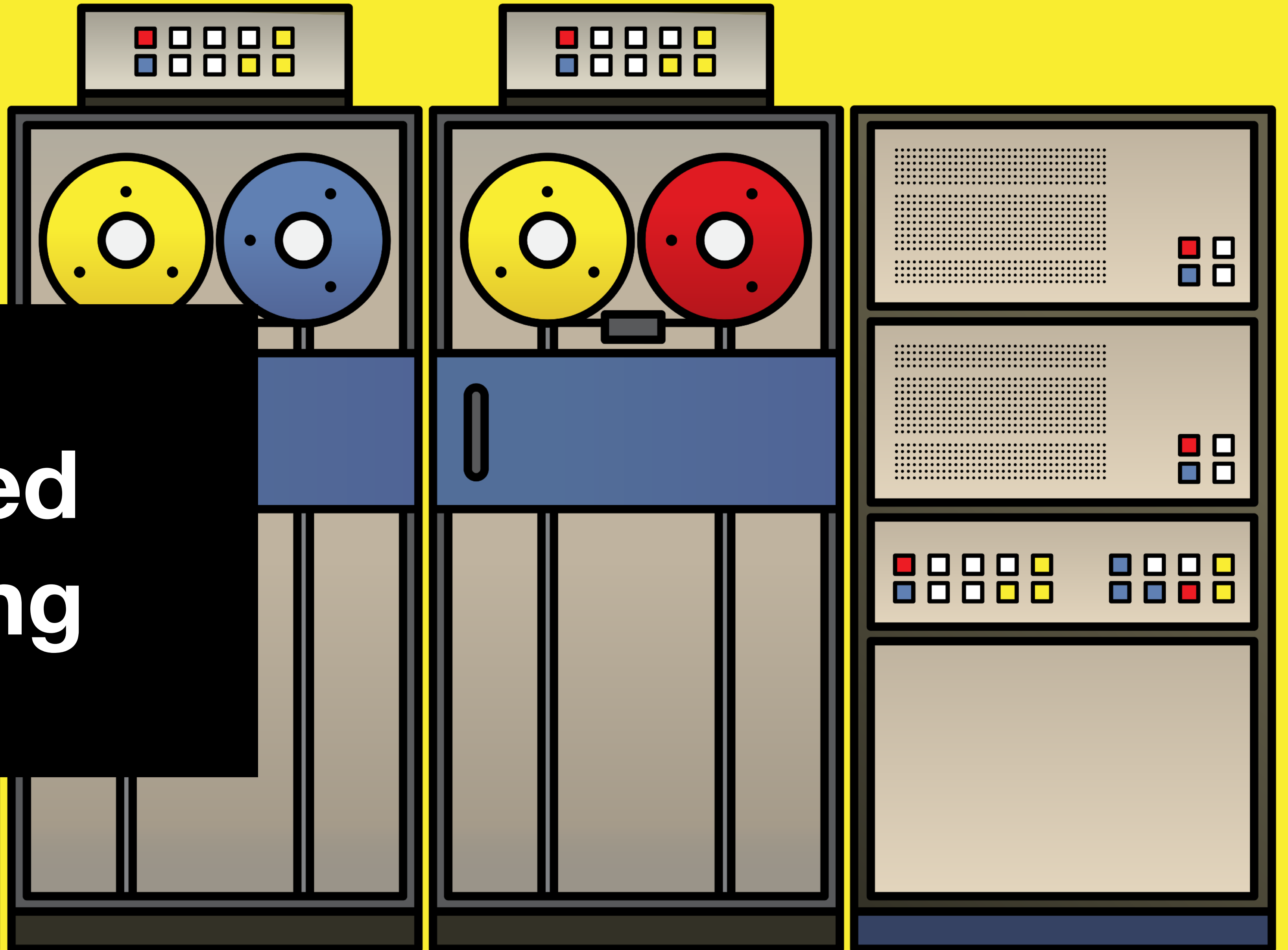
A spotlight on the internals of DuckDB and why

A demonstration of DuckDB's capabilities

A preview of MotherDuck and its architecture



Introduced Computing





The Good

Powerful
Centralized Compute
Shared Access

The Bad

Bulky
Expensive
Not Personalized



Popularized Computing

Why?

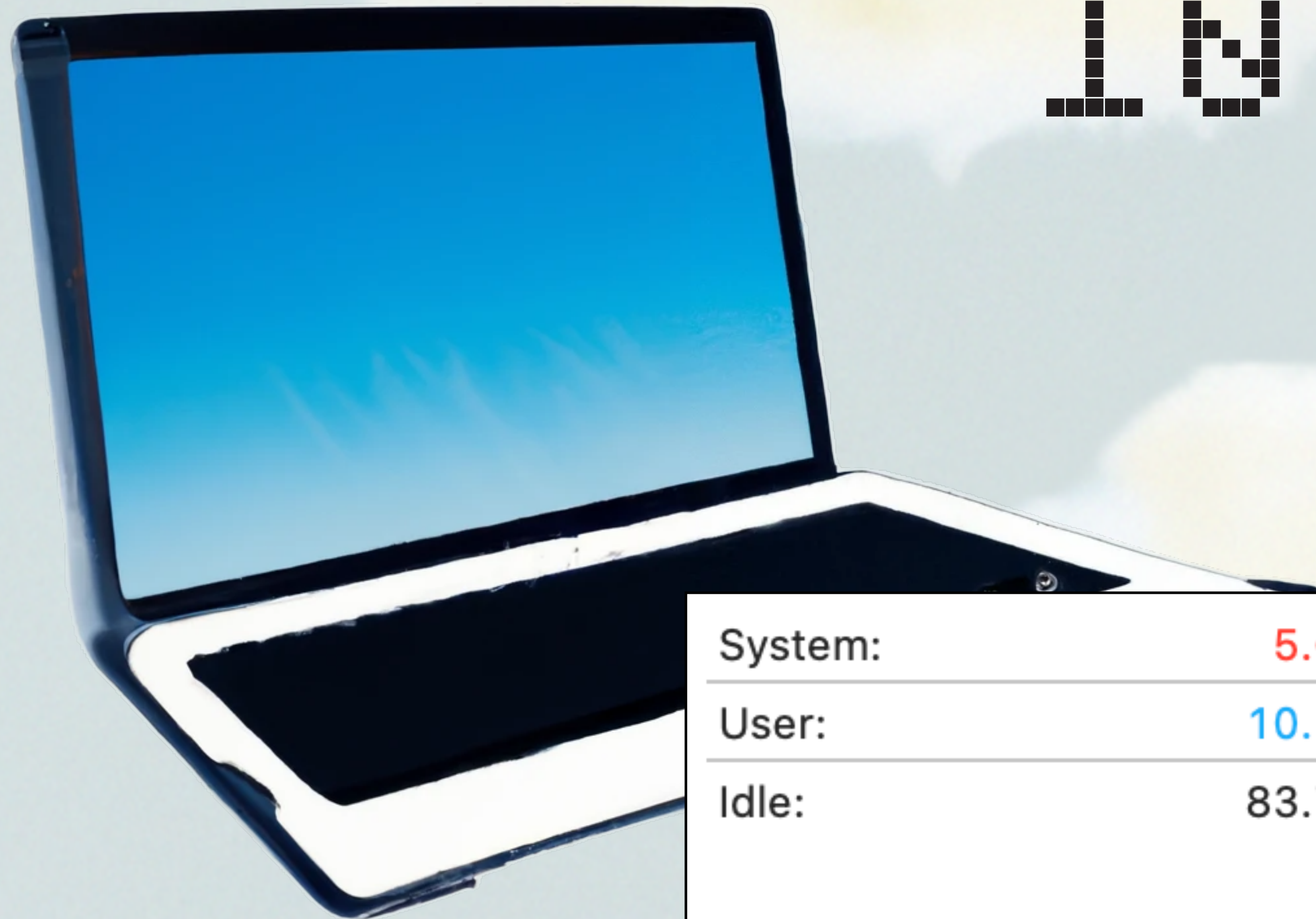
Dedicated
Personalized



The image features the letters 'CDW' in a large, bold, dark grey sans-serif font, centered horizontally. The background is a soft-focus photograph of a light blue sky with two large, white, fluffy clouds. One cloud is positioned above the text, and the other is below it, framing the letters.

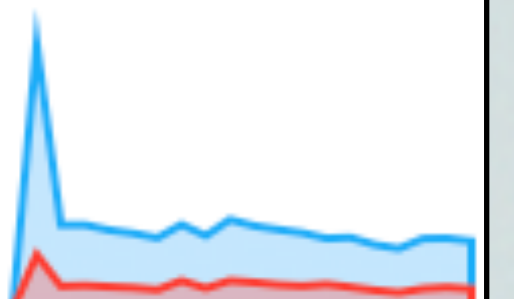
CDW

32 GB RAM
10 Cores

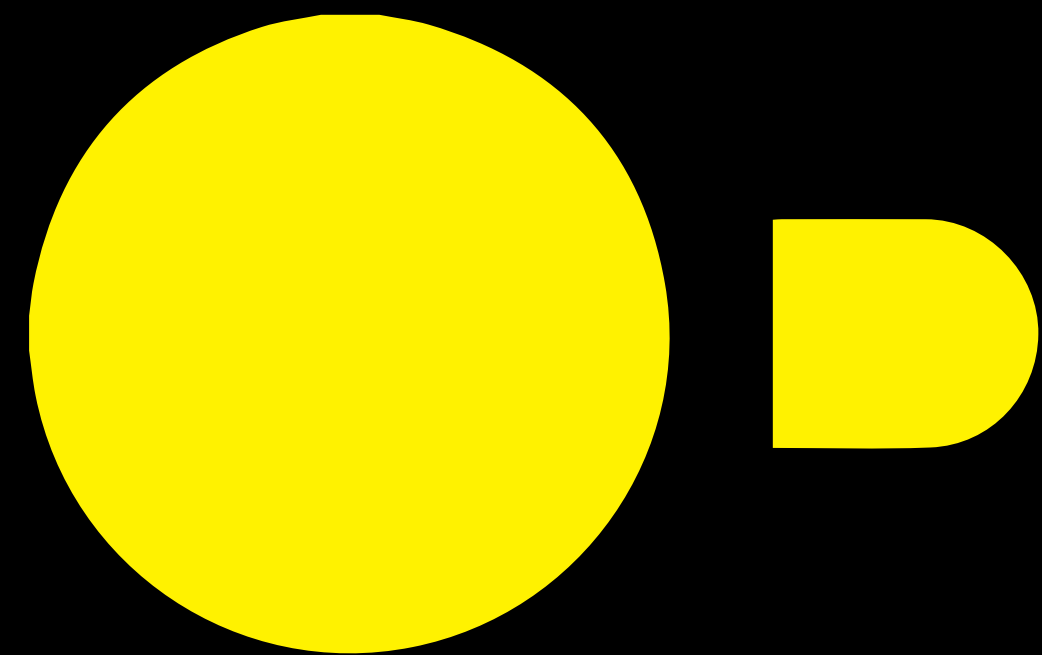


System:	5.66%
User:	10.56%
Idle:	83.77%

CPU LOAD







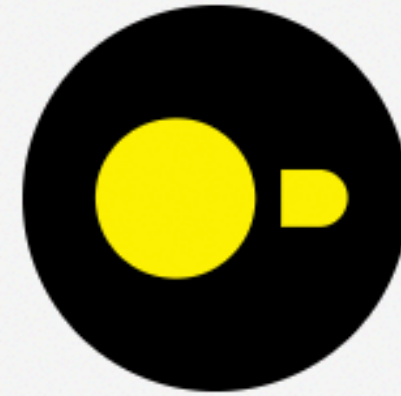
DuckDB



DuckDB



duckdb.org



DuckDB is an in-process SQL OLAP database management system



Simple

- In-process, serverless
- C++11, no dependencies, single file build
- APIs for Python/R/Java/...

[more](#) →



Feature-rich

- Transactions, persistence
- Extensive SQL support
- Direct Parquet & CSV querying

[more](#) →



Fast

- Vectorized engine
- Optimized for analytics
- Parallel query processing



Free

- Free & Open Source
- Permissive MIT License

[more](#) →



created at:



created by:



maintained by:

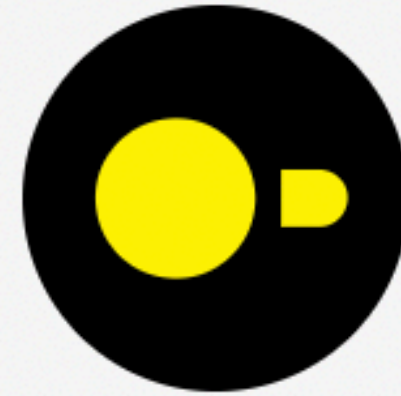




DuckDB



duckdb.org



DuckDB is an in-process SQL OLAP database management system



Simple

- In-process, serverless
- C++11, no dependencies, single file build
- APIs for Python/R/Java/...

[more](#) →



Feature-rich

- Transactions, persistence
- Extensive SQL support
- Direct Parquet & CSV querying

[more](#) →



Fast

- Vectorized engine
- Optimized for analytics
- Parallel query processing



Free

- Free & Open Source
- Permissive MIT License

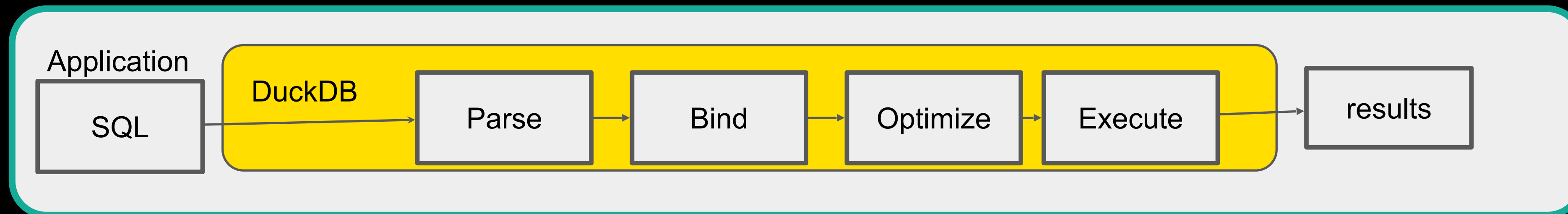
[more](#) →

Let's Break that Down

In-Process

Runs inside Python, R, C++.

No external dependencies.



Let's Break that Down

In-Process

Runs inside Python, R, C++.

No external dependencies.

OLAP

Full SQL support - things like window functions

High-performance aggregations



SQLite for Analytics

 DuckDB

created at:

created by:

maintained by:



 DuckDB Labs

 DuckDB Community
& Foundation

● DuckDB

- * High Performance Architecture
- * Simplified Data Access
- * OSS Community Flocks Together
- * Fast-paced innovation

High Performance Architecture

Comparing Database Engines

Row-based (tuples)

SQLite, PostgreSQL, etc

Columnar

Pandas, NumPy, etc.

Vector-based

DuckDB

Comparing Database Engines

Row-based (tuples)

SQLite, PostgreSQL, etc

Optimized for:

- * low memory footprint
- * transactional workloads

Comparing Database Engines

Columnar



Pandas, NumPy, etc.

Optimized for:

- * analytic workloads
- * aggregations
- * data compression on like data

Comparing Database Engines

Vector-based

DuckDB

Optimized for:

- * analytic workloads
- * aggregations
- * CPU - can do SIMD
- * CPU - cache locality

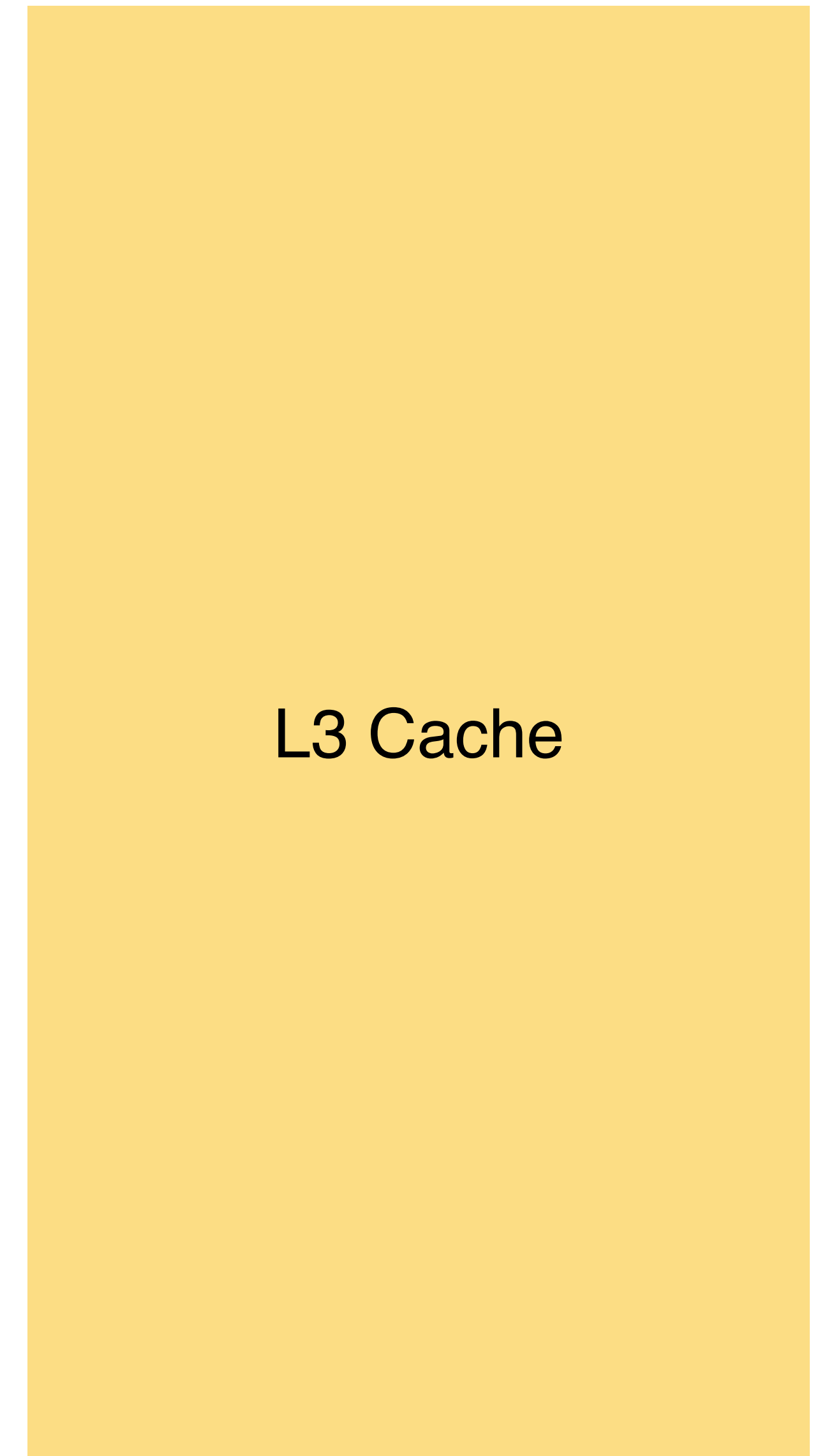
Why Vectorized?



2.9 MB



28 MB



48 MB

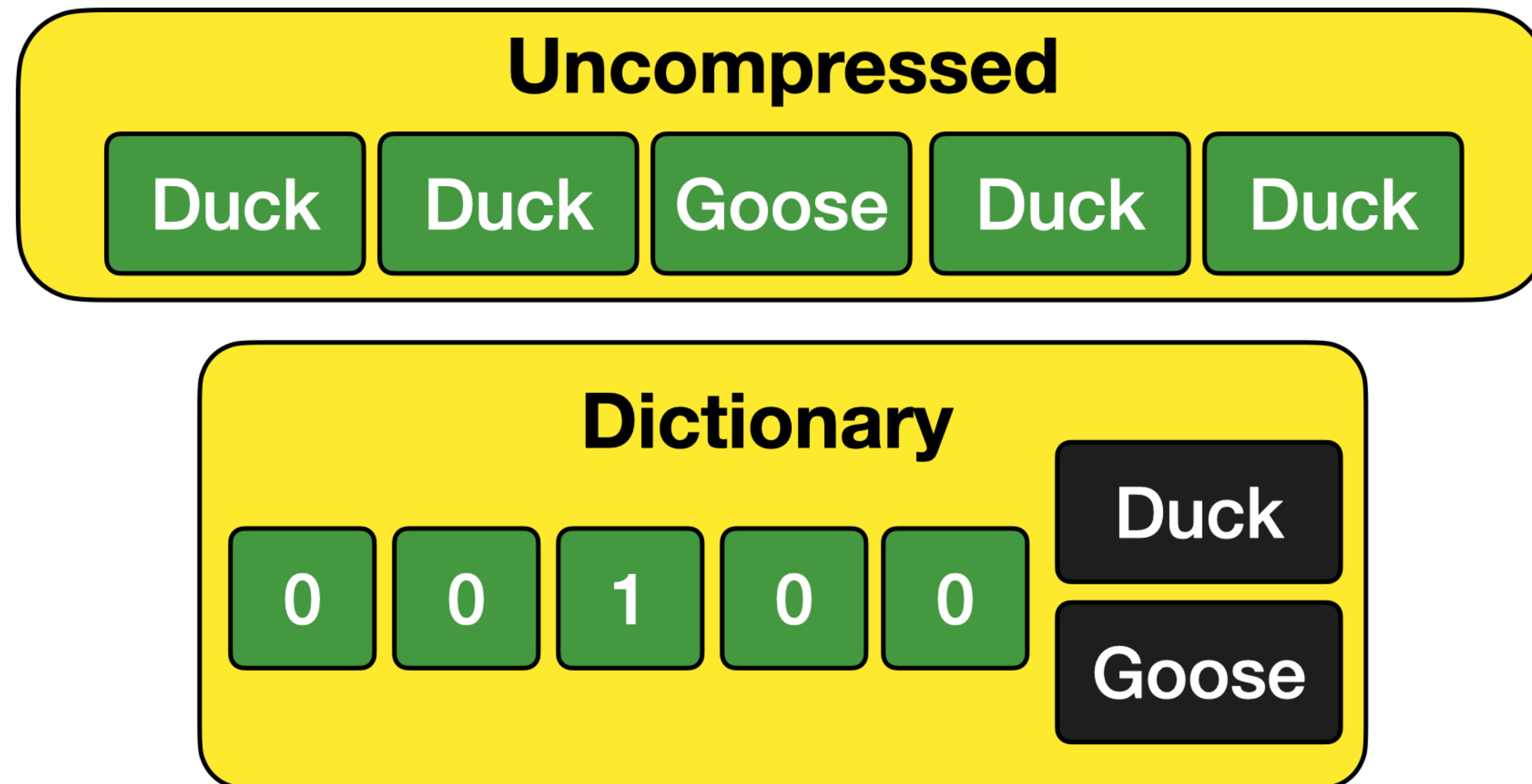
Not just in-memory

- * DuckDB has a native storage format
- * Supports larger-than-memory queries
(spillover to disk)

Native storage format

- * Columnar, partitioned
- * Efficient ACID-compliant updates
- * Stores an entire database, not just tables

Compression algos: dictionary



Compression algos: FSST

Uncompressed

www.google.com

www.github.com

FSST

0

2

1

0

3

1

Symbol Table

www.

.com

google

github

Compression algos: FSST

Version	Taxi	On Time	Lineitem	Notes	Date
DuckDB v0.2.8	15.3GB	1.73GB	0.85GB	Uncompressed	July 2021
DuckDB v0.2.9	11.2GB	1.25GB	0.79GB	RLE + Constant	September 2021
DuckDB v0.3.2	10.8GB	0.98GB	0.56GB	Bitpacking	February 2022
DuckDB v0.3.3	6.9GB	0.23GB	0.32GB	Dictionary	April 2022
DuckDB v0.5.0	6.6GB	0.21GB	0.29GB	FOR	September 2022
DuckDB dev	4.8GB	0.21GB	0.17GB	FSST + Chimp	NOW()
CSV	17.0GB	1.11GB	0.72GB		
Parquet (Uncompressed)	4.5GB	0.12GB	0.31GB		
Parquet (Snappy)	3.2GB	0.11GB	0.18GB		
Parquet (ZSTD)	2.6GB	0.08GB	0.15GB		

Compression algos: History

Version	Taxi	On Time	Lineitem	Notes	Date
DuckDB v0.2.8	15.3GB	1.73GB	0.85GB	Uncompressed	July 2021
DuckDB v0.2.9	11.1GB	1.25GB	0.79GB	RLE + Constant	September 2021
DuckDB v0.3.2	10.1GB	0.98GB	0.56GB	Bitpacking	February 2022
DuckDB v0.3.3	9.1GB	0.51GB	0.32GB	Dictionary	April 2022
DuckDB v0.5.0	6.1GB	0.21GB	0.29GB	FOR	September 2022
DuckDB dev	4.8GB	0.21GB	0.17GB	FSST + Chimp	NOW()
CSV	17.0GB	1.11GB	0.72GB		
Parquet (Uncompressed)	4.5GB	0.12GB	0.31GB		
Parquet (Snappy)	3.2GB	0.11GB	0.18GB		
Parquet (ZSTD)	2.6GB	0.08GB	0.15GB		

Compression algos

```
SELECT *  
FROM pragma_storage_info('persons')  
USING SAMPLE 100 rows
```

Compression algos

row_group_id int64	column_name varchar	column_id int64	segment_type varchar	count int64	compression varchar
10	email	3	VALIDITY	122880	Constant
79	phone	5	VALIDITY	122880	Constant
341	company	4	VARCHAR	31250	FSST
1007	email	3	VARCHAR	25355	FSST
1032	phone	5	VARCHAR	29526	FSST
1212	company	4	VARCHAR	30740	FSST
1252	company	4	VARCHAR	30570	FSST
1575	id	0	BIGINT	28672	BitPacking
1595	last_name	2	VARCHAR	122880	Dictionary
1733	id	0	VALIDITY	122880	Constant
2335	email	3	VARCHAR	25033	FSST

Simplified Data Access

PANDAS

create a sample pandas data frame

```
import pandas as pd
```

```
test_df = pd.DataFrame.from_dict({"i":[1, 2, 3, 4], "j":["one", "two", "three",  
"four"]})
```

make this data frame available as a view in duckdb

```
conn.register("test_df", test_df)
```

```
print(conn.execute("SELECT j FROM test_df WHERE i > 1").fetchdf())
```

HTTPS CSV

install and load httpfs

```
$ duckdb
```

```
D INSTALL httpfs;
```

```
D LOAD httpfs;
```

query as normal

```
SELECT * FROM 'https://rb-tmp-public.s3.amazonaws.com//  
persons_1.csv.gz' LIMIT 10
```

and do magic like

```
COPY(SELECT * FROM 'https://rb-tmp-public.s3.amazonaws.com//  
persons_1.csv.gz') TO 'persons_from_csv.parquet' (FORMAT  
PARQUET)
```

S3 PARQUET

install and load httpfs

```
$ duckdb
```

```
D INSTALL httpfs;
```

```
D LOAD httpfs;
```

set S3 creds

```
SET s3_region='us-east-1';
```

```
SET s3_access_key_id='AKIA42DX...BV';
```

```
SET s3_secret_access_key='w7oOQ60hyGAh...VGuQ6';
```

query as normal

```
SELECT * FROM read_parquet('s3://<bucket>/<file>');
```

or copy data over

```
CREATE TABLE trivia AS SELECT * FROM read_parquet('s3://  
<bucket>/<file>');
```

POSTGRESQL

install and load postgresql

```
$ duckdb
```

```
INSTALL postgres;
```

```
LOAD postgres;
```

attach postgresql tables as read-only views

```
CALL postgres_attach('dbname=postgres');
```

query as normal

```
SELECT AVG(id)
```

```
FROM persons
```

or query both DuckDB data and postgresql

or use the awesomeness of parquet creation

```
COPY(SELECT * FROM postgres_scan('dbname=postgres', 'public',  
'persons')) TO 'persons.parquet' (FORMAT PARQUET);
```

Simplified Data Access

**Parquet, CSV, SQLite, PostgreSQL, Arrow
Local, Remote on S3**

In-process, Command-line, In-browser, ODBC/JDBC

Simplified Data Access

Parquet, CSV, SQLite, PostgreSQL, Arrow
Local, Remote on S3

In-process, Command-line, **In-browser**, ODBC/JDBC

**For “normal
size” data**



David is at data-folks.masto.host

@DSJayatilake



The vast majority of orgs in the world do not have > billion row datasets, but many need the benefits of data.

This is partly why I'm so excited about [@duckdb](#); it affords the power of an expensive cloud dwh without the complexity, for smaller data for ALL.

8:10 AM · 17 Apr, 2022

2 replies 12 likes

**And larger than
normal size data**

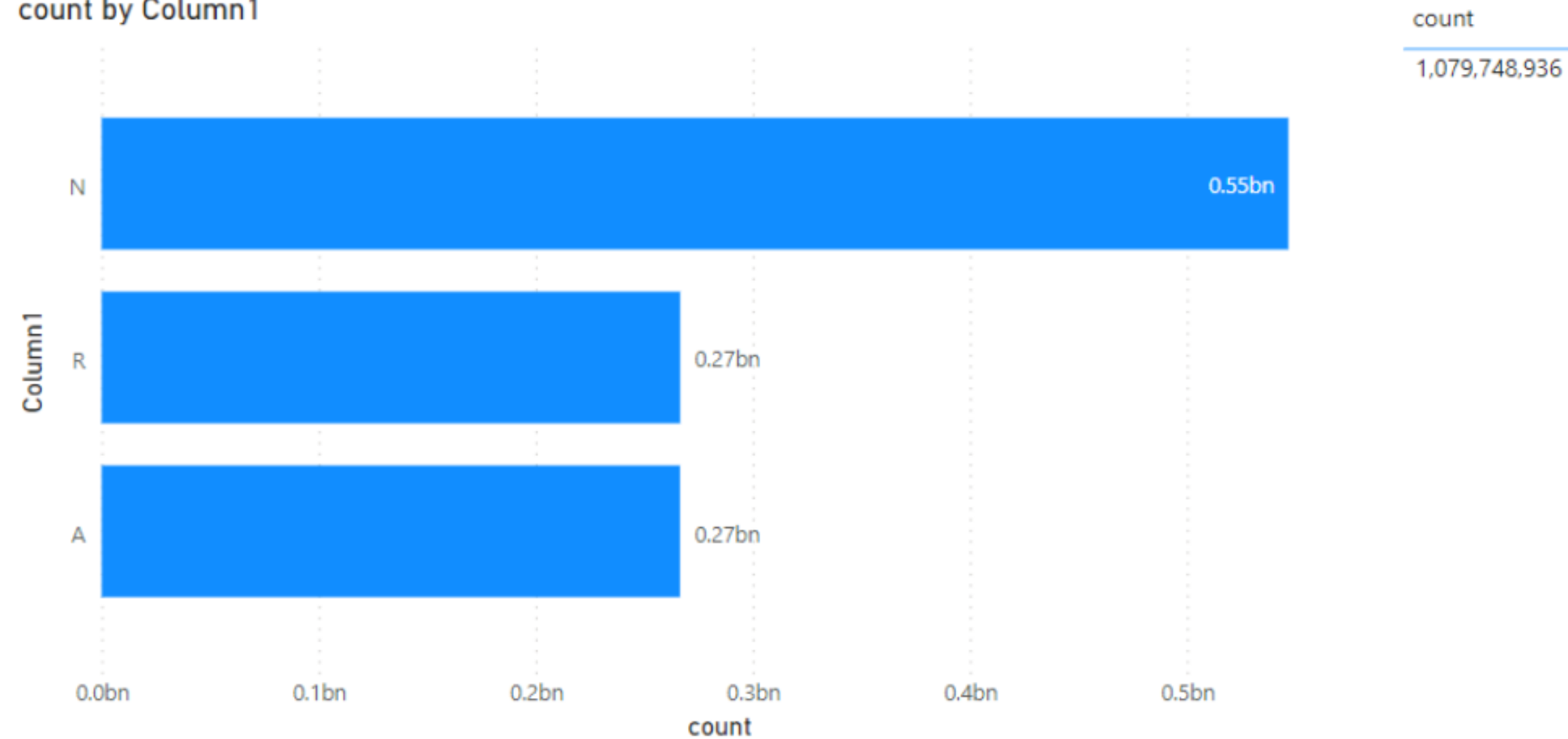


Mim
@mim_djo



the irony of life, I never done a Query on a Billion row Table, but when I did it , it was on my laptop using an in-process Database [#DuckDB](#)

count by Column1



5:57 AM · 17 Apr, 2022

2 replies 6 shares 43 likes



1,450,552,076

NYC Yellow Taxi Trips from 2010-present


```
$ duckdb
```

```
D load parquet;
```

```
D SELECT AVG(trip_distance) FROM  
'yellow_tripdata_20[12]*.parquet';
```

<code>avg(trip_distance)</code>
6.1801595481684

2.95s querying 1.45 billion rows

No indexes

```
$ duckdb taxis.ddb
```

```
D SELECT AVG(trip_distance) FROM yellow_tripdata;
```

avg(trip_distance)
6.1801595481684

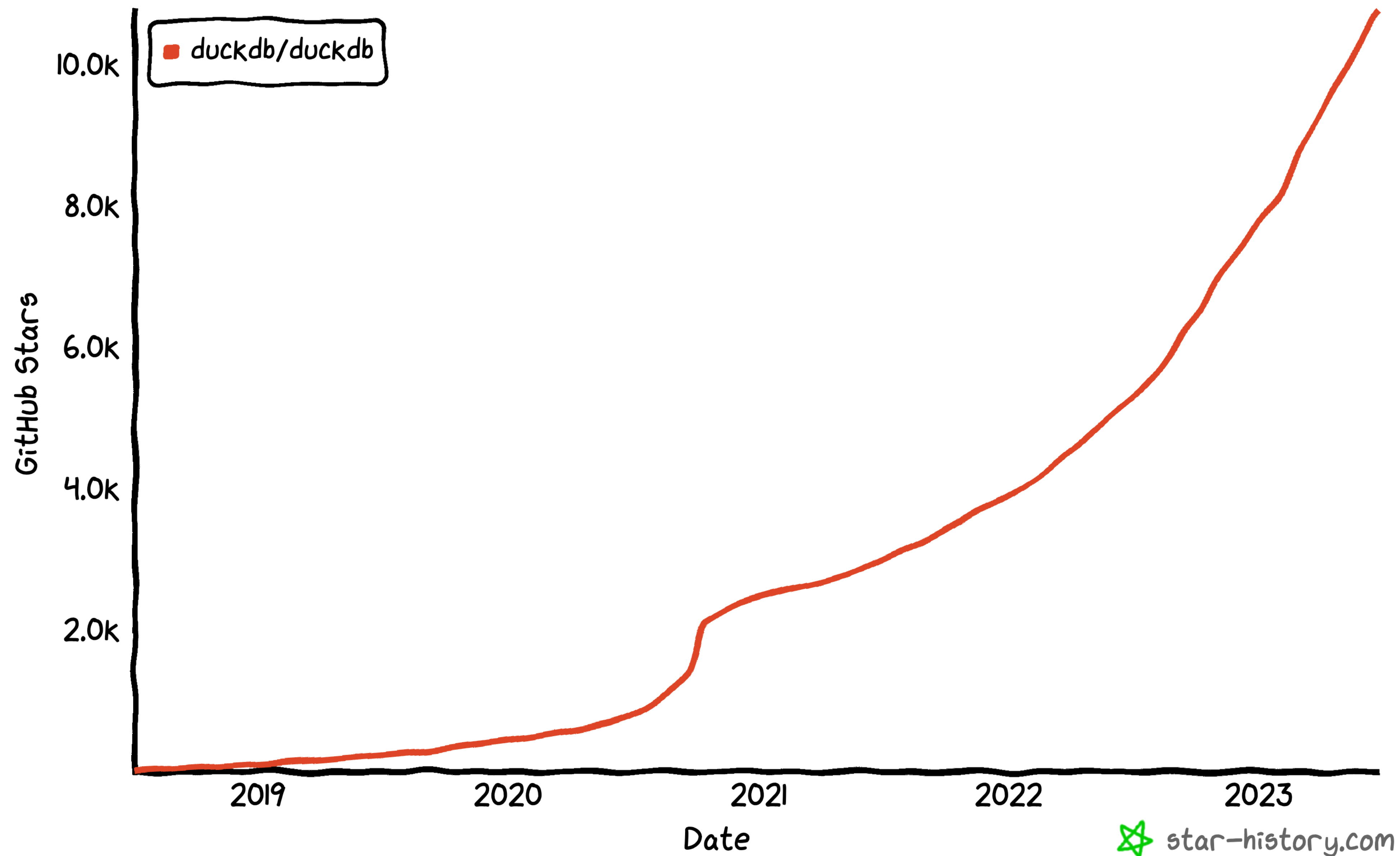
1.47s querying 1.45 billion rows

No indexes

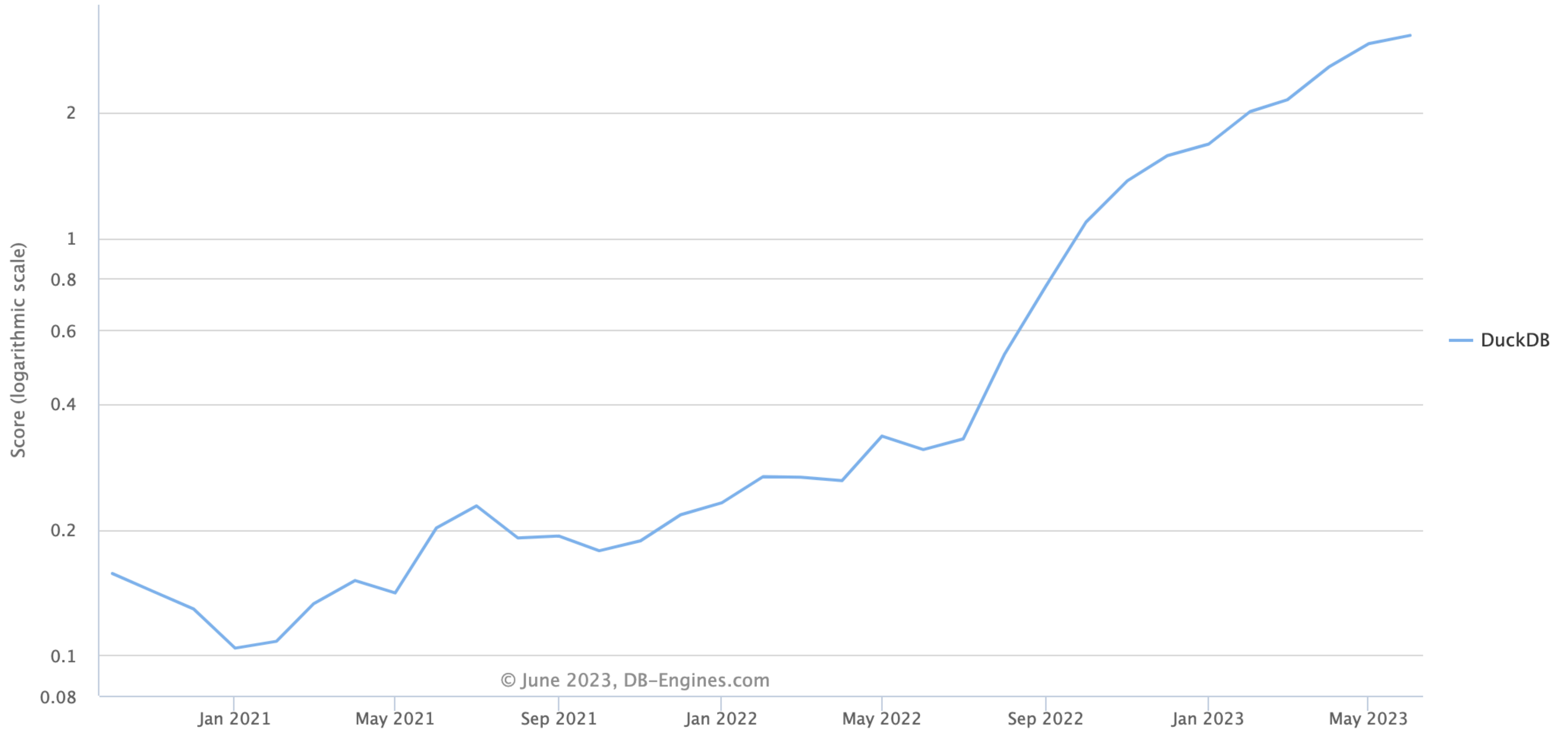
m6i.4xlarge	16 cores, 64GB RAM	1.020s
m6i.8xlarge	32 cores, 128GB RAM	0.509s
m6i.16xlarge	64 cores, 256GB RAM	0.268s
m6i.32xlarge	128 cores, 512GB RAM	0.145s

**OSS Community
Flocks Together**

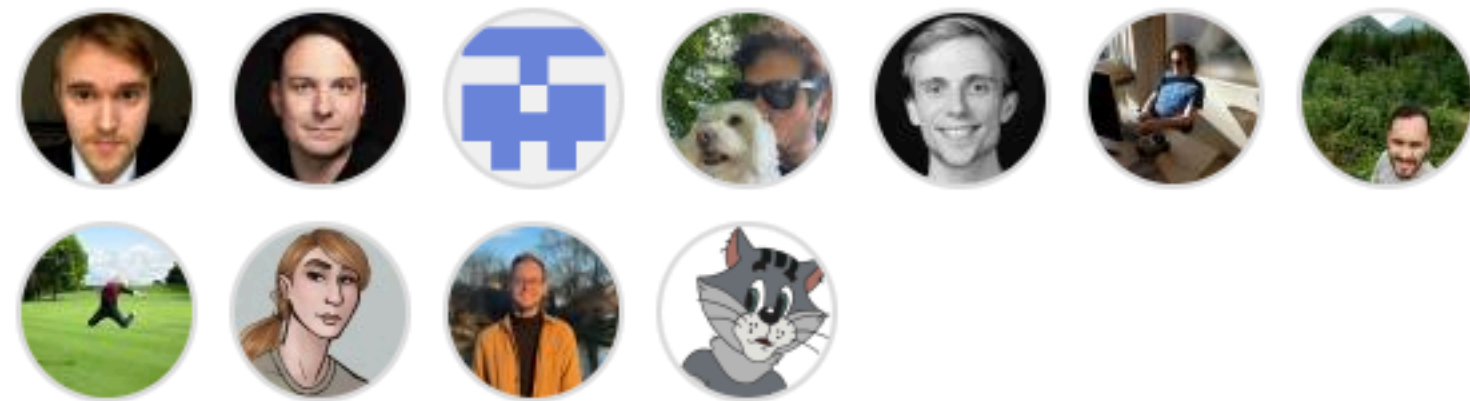
● Star History



DB-Engines Ranking of DuckDB



Contributors 216



+ 205 contributors



DuckDB



Community Server



302 Online



3,087 Members

DuckDB Ecosystem Monthly

Happy new year, friend 🙌

Hi, I'm [Marcos](#)! I'm a data engineer by day at X-Team, working for Riot Games. By night, I create newsletters for a few topics I'm passionate about: helping folks [find data digs](#) and AWS graviton. After getting involved in the DuckDB community, I saw a great opportunity to partner with the MotherDuck team to share all the amazing things happening in the DuckDB ecosystem.

In this first issue of the year 2023, we wanted to share some of the incredible stuff coming out of the global DuckDB community.

-Marcos

Feedback: duckdbnews@motherduck.com

Featured Community Members



Jacob Matson

Jacob is the writer of the [Modern Data Stack in a Box with DuckDB](#). A fast, free, and open-source [Modern Data Stack \(MDS\)](#) can now be fully deployed on your laptop or to a single machine using the combination of DuckDB, Meltano, dbt, and Apache Superset.

He is working today as the VP of Finance & Operations at Simetric, bringing IoT connectivity data into a single pane-of-glass. He also does SMB analytics consulting via his agency, Elliot Point LLC. You can find him on Twitter [@matsonj](#)

**Fast-paced
Innovation**

RELEASE 0.6

Optimistic writing to disk [assuming successful COMMIT]

Parallel data loading [91.4s -> 17.2s for 150 million rows!]

Three new compression algos: FSST, Chimp, Patas

Parallel

- * CSV reading
- * Index creation
- * COUNT(DISTINCT)

SELECT clause now optional

List comprehension support `[x + 1 for x in [1, 2, 3]]`

RELEASE 0.7

JSON ingestion via read_json

Partitioned export of CSV and Parquet

Parallel CSV and Parquet writing

Multi-database ATTACH support

SQLite backend support

Positional JOIN support

Upsert support

Better Python APIs

RELEASE 0.8

User-defined scalar functions for Python

Support for ADBC

Swift API

Parallel JSON writing

PIVOT and UNPIVOT

Lazy-loading table metadata



created at:



created by:



maintained by:





**But who the Duck
is MotherDuck?**



is: venture-backed startup

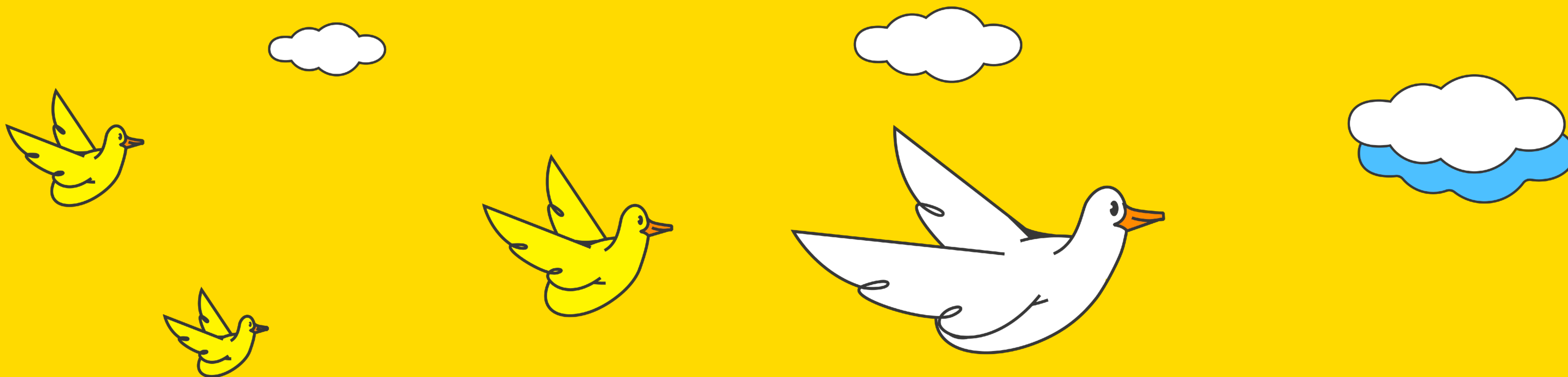
made up of:

data geeks from Google BigQuery,
Databricks, Snowflake, Meta, Elastic,
SingleStore, ++

doing:



LET'S MAKE
ANALYTICS
DUCKING AWESOME



TEACH YOUR DUCKDB TO FLY



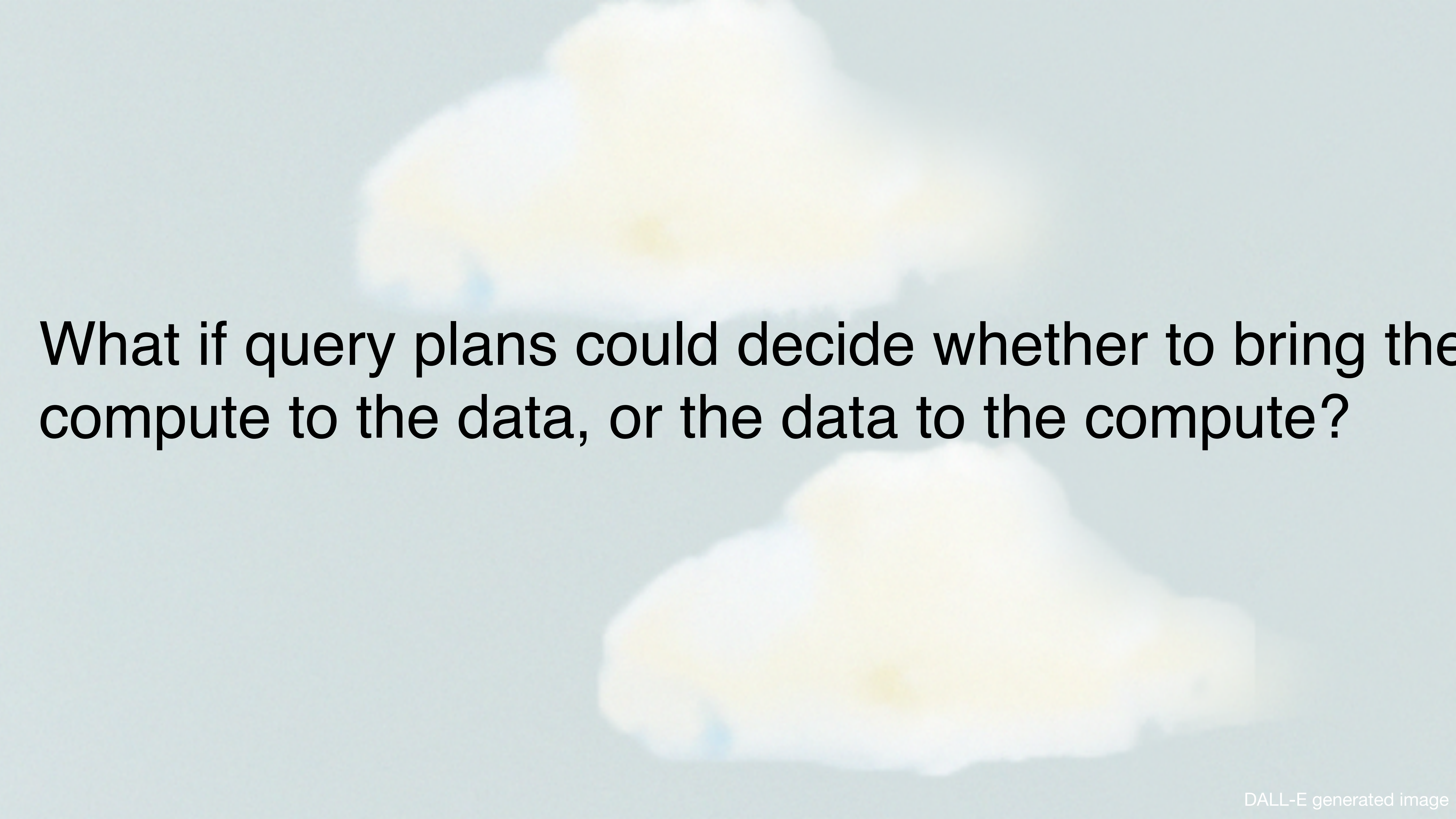
Marc Lamberti

@marclambertiml

Just an honest question, wasn't the purpose of DuckDB to run fast analytical queries in LOCAL? What's the need for a cloud version?

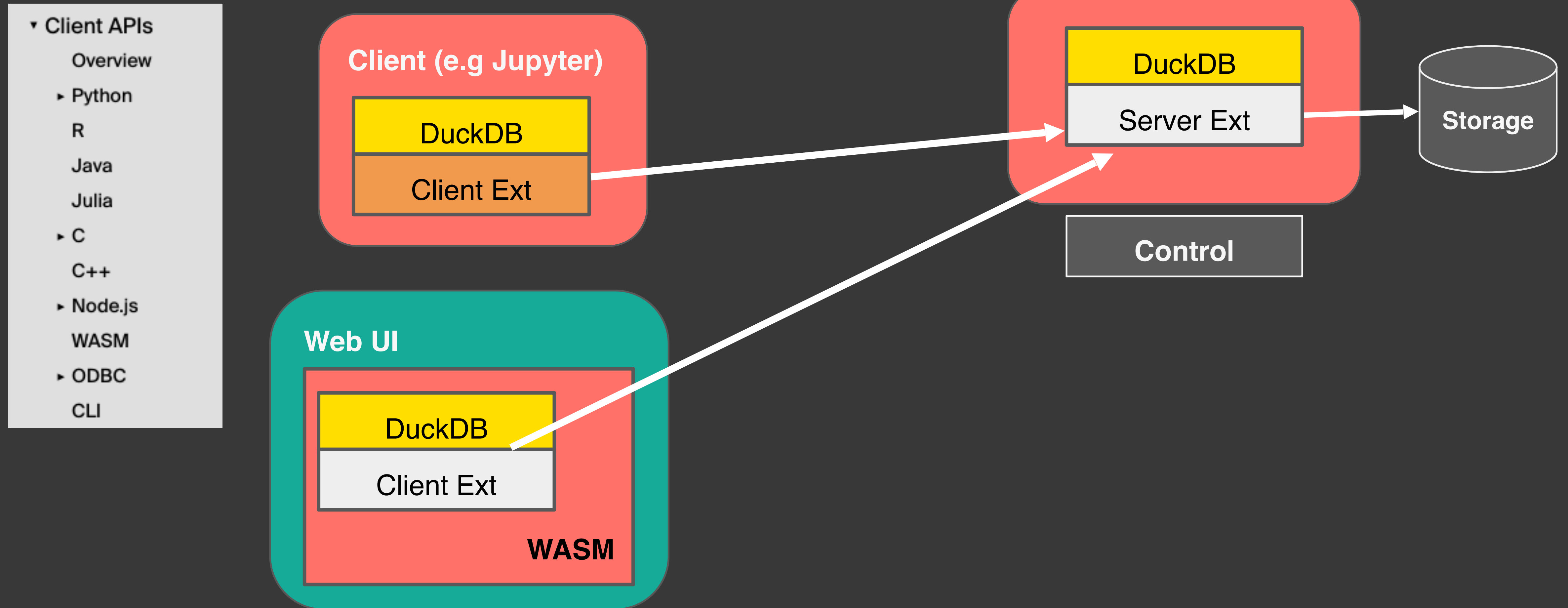
Our Beliefs @ MotherDuck





What if query plans could decide whether to bring the
compute to the data, or the data to the compute?

LET'S GET REAL



SELECT

cr.currency_code,
t.passenger_count,
AVG(t.total_amount * cr.exchange_rate) as average_converted_amount

FROM

sample_data.nyc.yellow_cab_nyc_2022_11 t

CROSS JOIN

(SELECT * FROM './popular_currency_rate_dollar_20230620.csv') cr

WHERE cr.currency_code = 'EUR'

GROUP BY

cr.currency_code, t.passenger_count

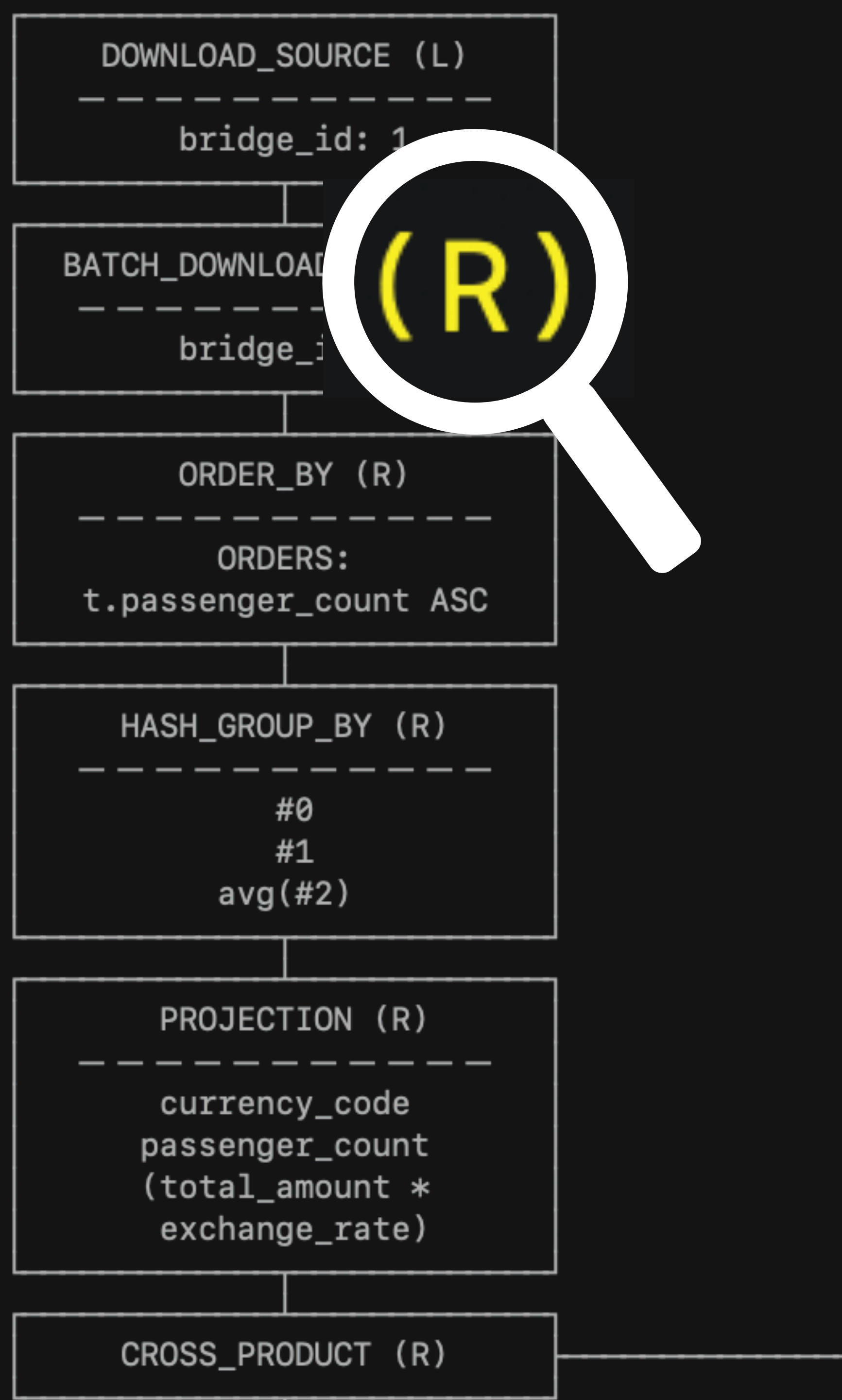
ORDER by t.passenger_count **ASC**;

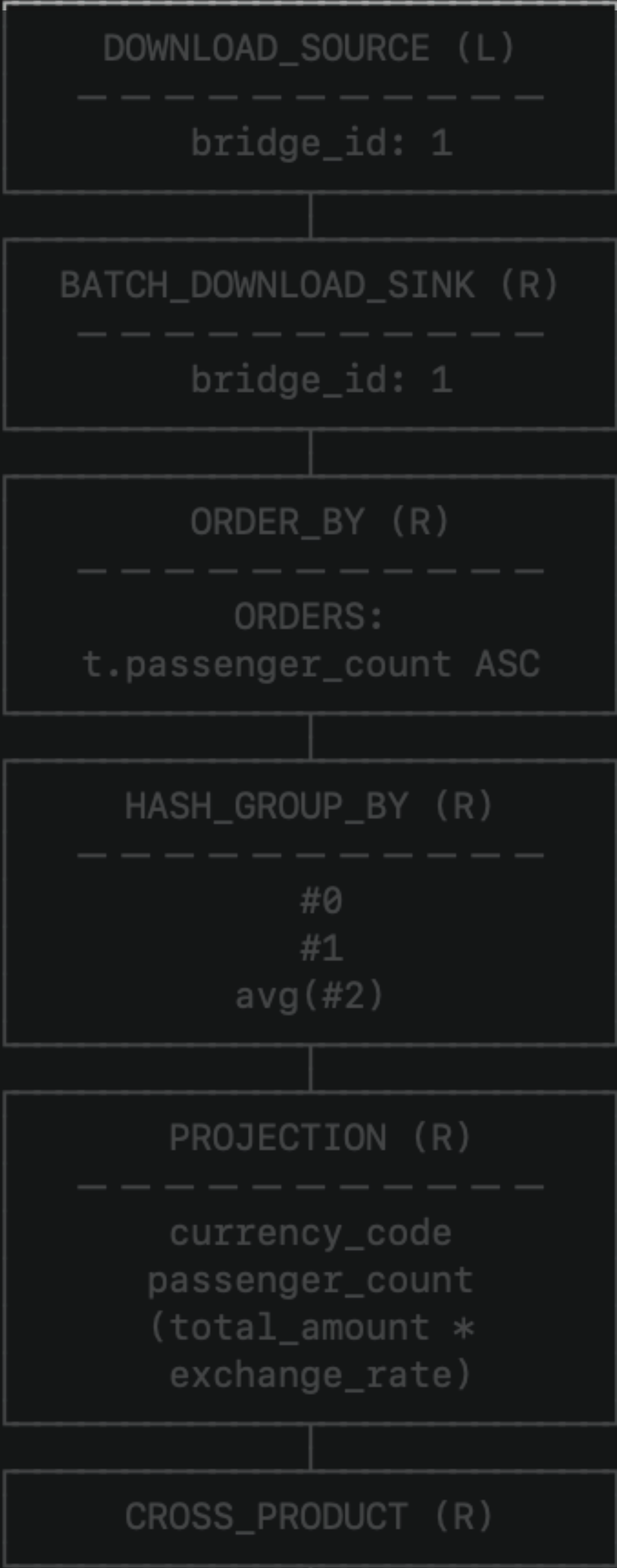

```
SELECT
    cr.currency_code,
    t.passenger_count,
    AVG(t.total_amount * cr.exchange_rate) as average_converted_amount
FROM
    sample_data.nyc.yellow_cab_nyc_2022_11 t
CROSS JOIN
    (SELECT * FROM './popular_currency_rate_dollar_20230620.csv') cr
WHERE cr.currency_code = 'EUR'
GROUP BY
    cr.currency_code, t.passenger_count
ORDER by t.passenger_count ASC;
```



```
SELECT
    cr.currency_code,
    t.passenger_count,
    AVG(t.total_amount * cr.exchange_rate) as average_converted_amount
FROM
    sample_data.nyc.yellow_cab_nyc_2022_11 t
CROSS JOIN
    (SELECT * FROM './popular_currency_rate_dollar_20230620.csv') cr
WHERE cr.currency_code = 'EUR'
GROUP BY
    cr.currency_code, t.passenger_count
ORDER by t.passenger_count ASC;
```







currency_code varchar	passenger_count double	average_converted_amount double
EUR	0.0	18.546608022179736
EUR	1.0	19.4198124783772
EUR	2.0	22.04667298565129
EUR	3.0	21.236435271048638
EUR	4.0	22.20329272449214
EUR	5.0	19.238401343401563
EUR	6.0	19.688502601081026
EUR	7.0	69.78117721000001
EUR	8.0	56.326965400000006
EUR	9.0	69.24747212666666
EUR		26.81066096388576
11 rows		3 columns

+ ADD FILES

My Databases

- > foo
- > localmemdb
- > sample_data

hn

hacker_news

main
- > nyc

fhvhv_tripdata_nyc_2022_11

service_requests_311_fro...

yellow_cab_nyc_2022_11
- > who

ambient_air_quality

```
7      MONTH(timestamp) AS month,
8      ROW_NUMBER(
9          OVER (PARTITION BY YEAR(timestamp), MONTH(timestamp) ORDER BY score DESC)
10         AS rn
11 FROM sample_data.hn.hacker_news
12 WHERE type = 'story'
13 )
14
15 SELECT
16     year,
17     month,
18     title,
19     hn_url,
20     score
21 FROM ranked_stories
22 WHERE rn = 1
23 ORDER BY year, month;
```

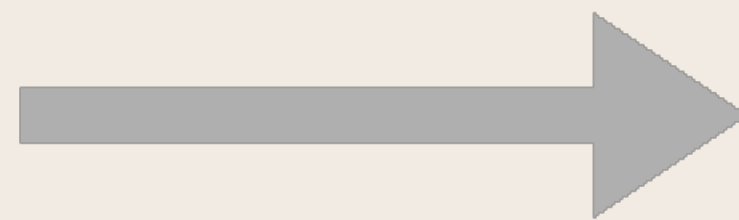
Query executed in 640 ms. Row count: 11

	year	month	title	hn_url	score
<div></div>		<div><div></div><div>2.01.0012</div></div>			<div><div></div><div>2.01,5004,000</div></div>
	2,022	1	My First Impressions of Web3	https://news.ycombinator.com/item?id=2984...	
	2,022	2	Google Search Is Dying	https://news.ycombinator.com/item?id=3034...	
	2,022	3	US Senate votes unanimously to make daylig...	https://news.ycombinator.com/item?id=3068...	
	2,022	4	Elon Musk makes \$43B unsolicited bid to tak...	https://news.ycombinator.com/item?id=3102...	
	2,022	5	Mechanical Watch	https://news.ycombinator.com/item?id=3126...	
	2,022	6	Supreme Court Overturns Roe vs. Wade	https://news.ycombinator.com/item?id=3186...	
	2,022	7	Ask HN: What are some cool but obscure dat...	https://news.ycombinator.com/item?id=3218...	
	2,022	8	Phishing kit for sale on GitHub	https://news.ycombinator.com/item?id=3240...	

MotherDuck Features

- Git-like collaboration: sharing snapshots
- If it can Duck, it can MotherDuck

```
$ duckdb my.ddb
```



```
$ duckdb md:mydb
```

HEX

dbt

Apache
Airflow

ASTRONOMER

Apache
Superset

omni

ASCEND.IO

Census

PONDER

LangChain

dagster

CloudQuery

Rill

expanso

colab

preset

LET'S GET REAL

```
$ duckdb my.db md:mydb
```

```
-- OAuth Loop for credentials
```

```
-- Create remote database and table
```

```
D CREATE DATABASE db1
```

```
D CREATE TABLE t1 as select 'abc' as x
```

```
-- Open local database and join to remote
```

```
D ATTACH local.db as L
```

```
D SELECT * from db1.t1
```

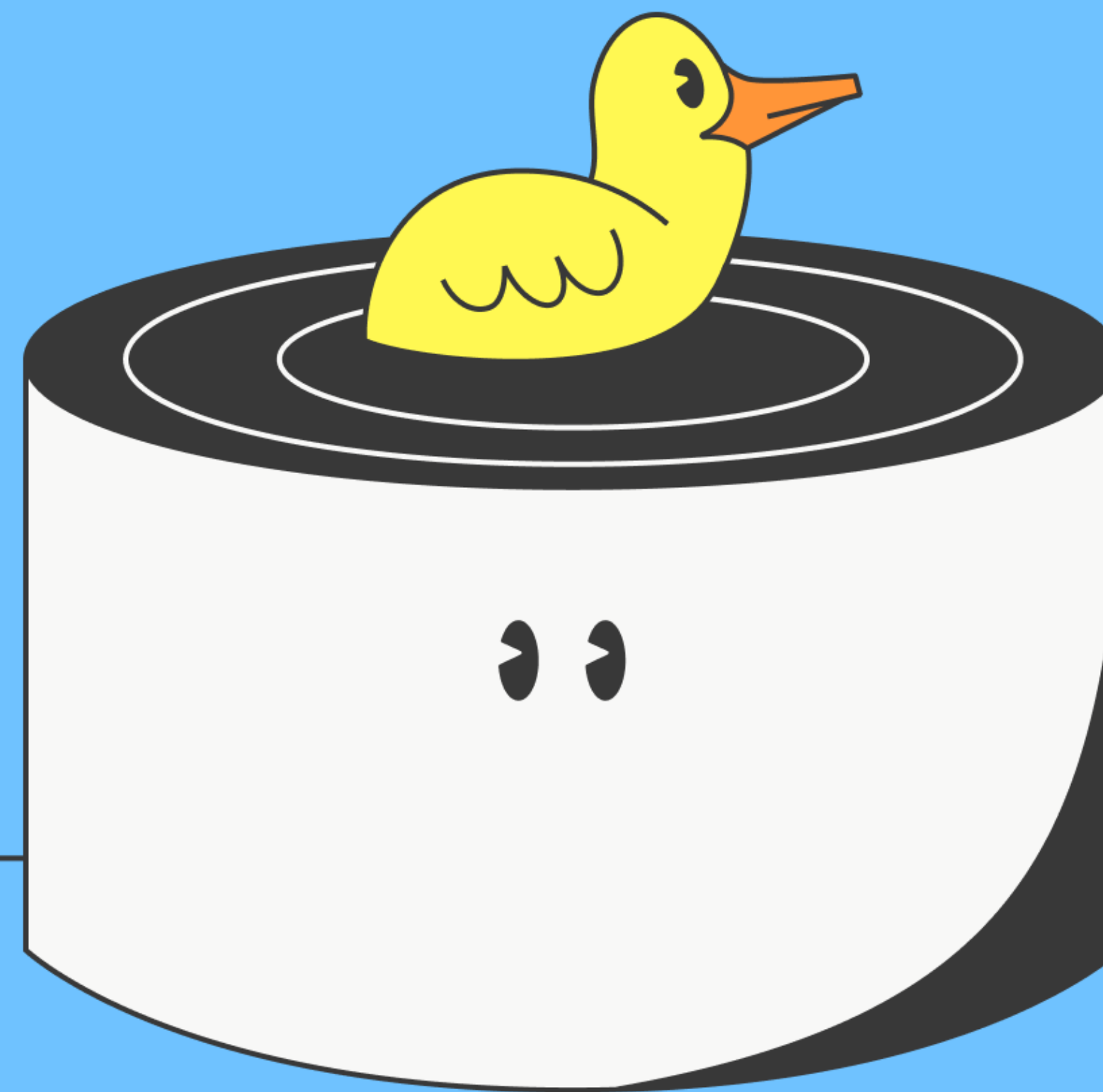
```
  JOIN L.t1 on (id)
```



DuckDB turns your laptop
into a personal analytics
engine



MotherDuck scales your
laptop into the cloud with
Hybrid Execution

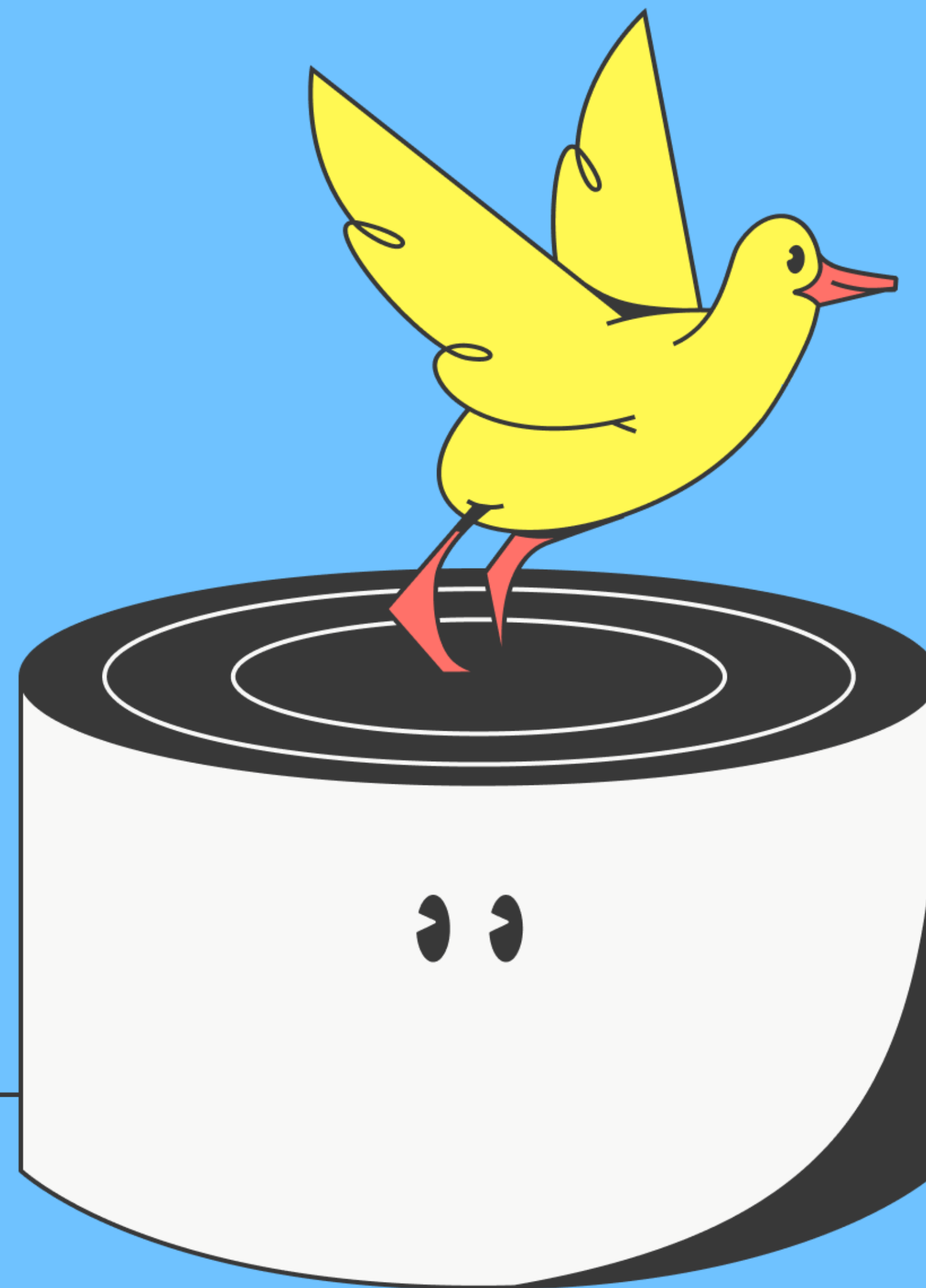




DuckDB is simple for dev
and prototyping with local
and remote data



MotherDuck lets you
move into production





DuckDB is an embedded database



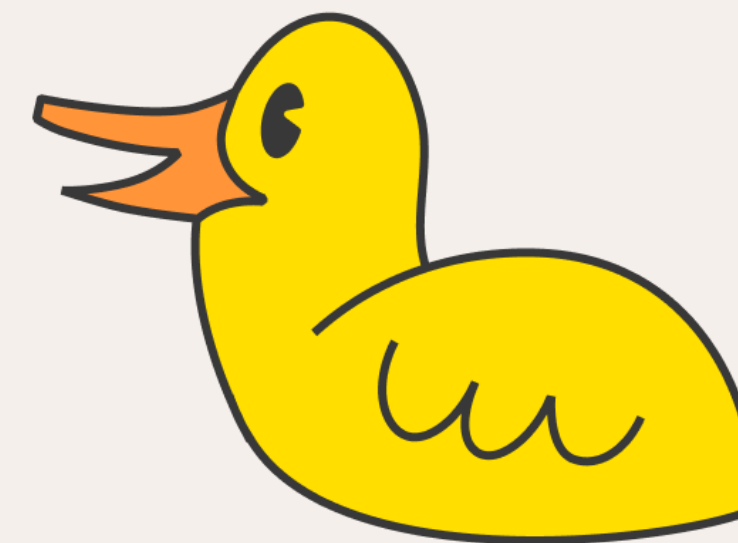
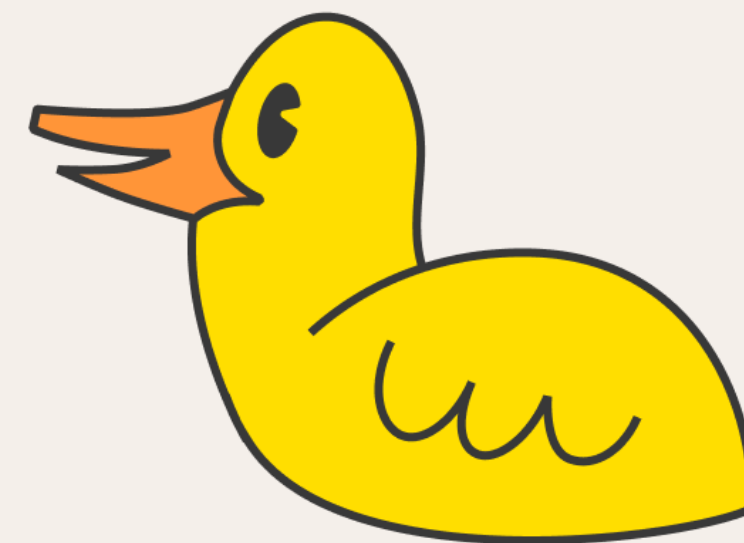
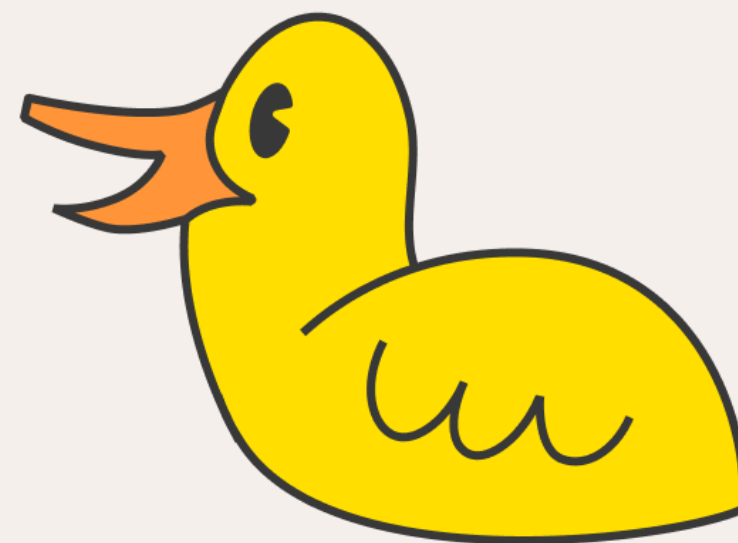
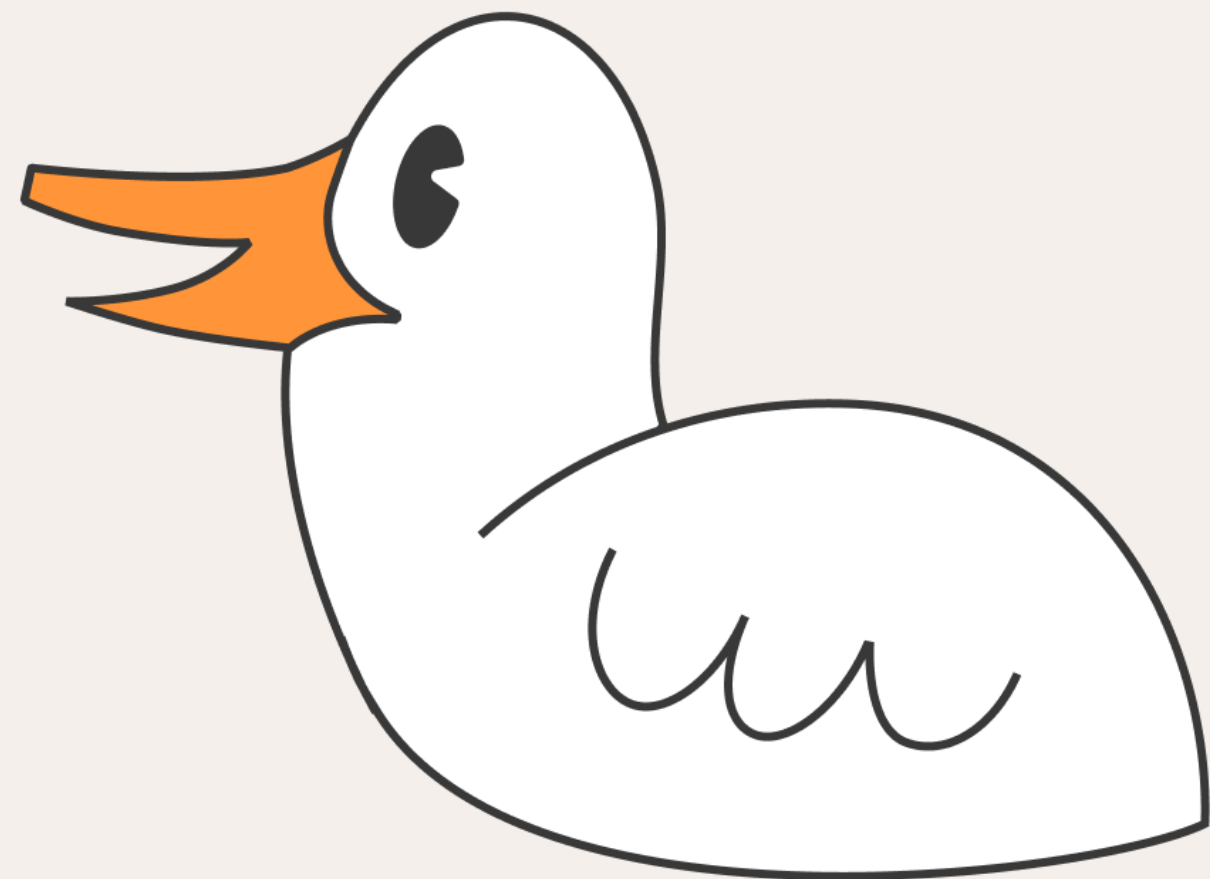
MotherDuck is a collaborative serverless analytics platform powered by DuckDB

motherduck.com

DUCKCON on THURSDAY

bit.ly/duckcon-sf

STOP QUACKING
& GET QUERYING!



Ryan Boyd
Co-founder @ MotherDuck
@ryguyrg

Our Beliefs @ MotherDuck





1. What is **Big Data** really?
2. How much data do people have?
3. How much data do they query?



DEFINITION OF BIG DATA

- More than can fit in Excel
- More than can fit on your laptop
- More than can fit in memory
- More than can fit on disk



DEFINITION OF BIG DATA

In 2012, when I worked on BigQuery:

Largest EC2 instance was 60.5GB of RAM

Today:

Largest EC2 instance is 25TB of RAM

400X



DEFINITION OF BIG DATA

In 2012, when I worked on BigQuery:
Largest MacBook Pro was 8GB of RAM

Today:
Largest MacBook Pro is 96GB of RAM

12X



DATA SIZES

- **BigQuery:** ~95% of customers with $< 1\text{TB}$
- **SingleStore:** ~80% of customers want S-00
- **Gartner:** Most EDWs $< 100\text{GB}$
- **A16Z:** B2B portfolio cos all had data $< 1\text{T}$
- **A16Z:** B2C portfolio cos had data $< 10\text{T}$

BIG DATA IS DEAD
LONG LIVE EASY DATA

DATA RECENCY

- Most data queried is from today
- Today's data is a small fraction of the whole
- Access history gets quiet very quickly
- Most data stored is very infrequently used





QUERY SIZES

- **BQ:** 90% of queries < 100 MB
- Just because you have a large data doesn't mean you query it
- Effective partitioning, compression, pushdowns, etc reduce query sizes