PYCON AUSTRIA | EISENSTADT | 2026-04-20

Why would you "import duckdb" in your Python project?

GÁBOR SZÁRNYAS
(DUCKDB LABS)

MSc

databases

PhD

query
processing

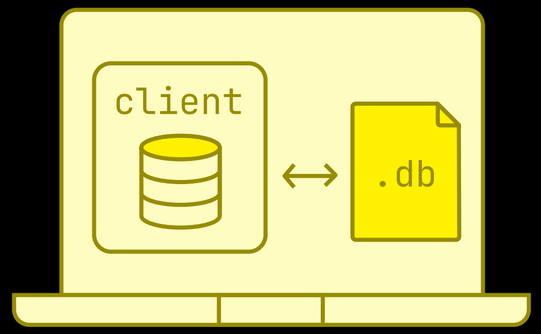
CWI

postdoc

graph databases,
benchmarks

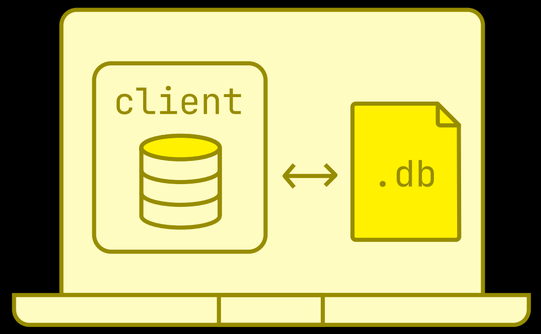
DuckDB Labs

DuckDB = a SQL database for analytics




```
$ curl https://install.duckdb.org | sh
$ duckdb my.db

D SELECT *
  FROM '/your/favorite/file.csv';
```

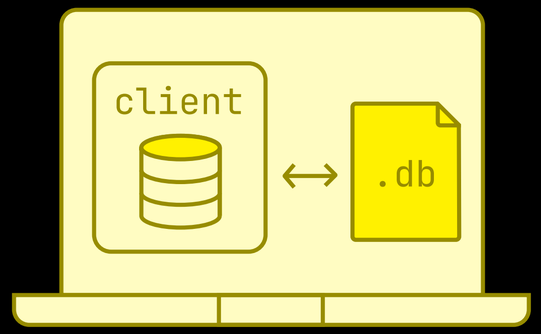
DuckDB = a SQL database for analytics




```
$ curl https://install.duckdb.org | sh
$ duckdb my.db

D SELECT *
  FROM '/your/favorite/file.csv';
```

```
$ pip install duckdb
$ python

>>> import duckdb
>>> duckdb.sql("FROM 'my_data.csv'")
```

DuckDB = a SQL database for analytics




```
$ curl https://install.duckdb.org | sh
$ duckdb my.db

D SELECT *
  FROM '/your/favorite/file.csv';
```

```
$ pip install duckdb
$ python

>>> import duckdb
>>> duckdb.sql("FROM 'my_data.csv'")
```

Zero-dependency C++ code: 15+ clients

PyPI Stats

Search

All packages
Top packages

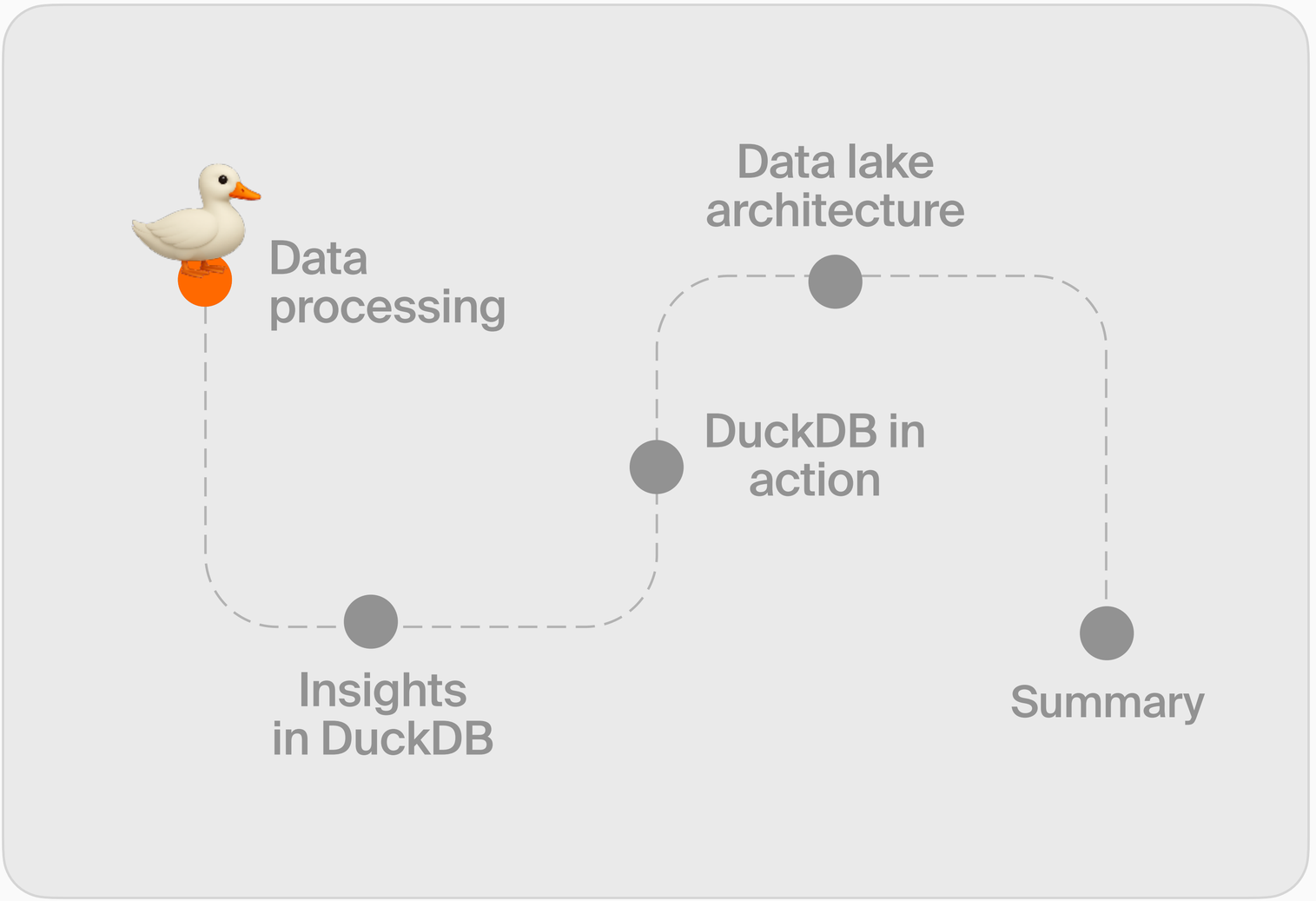
Track packages

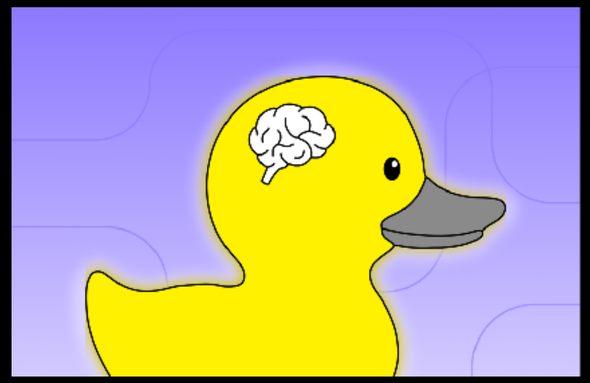
duckdb

PyPI page
Home page
Author: DuckDB Foundation
Summary: DuckDB in-process database
Latest version: 1.5.2

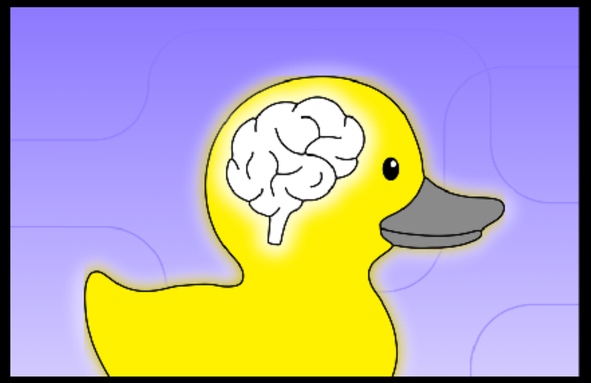
Optional dependencies: adbc-driver-manager | fsspec | ipython | numpy | pandas | pyarrow

Downloads last day: 762,722
Downloads last week: 10,178,060
Downloads last month: 39,030,034

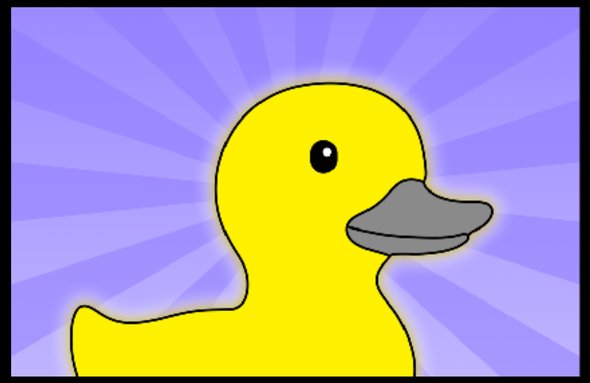

Data processing
Data lake architecture
DuckDB in action
Insights in DuckDB
Summary



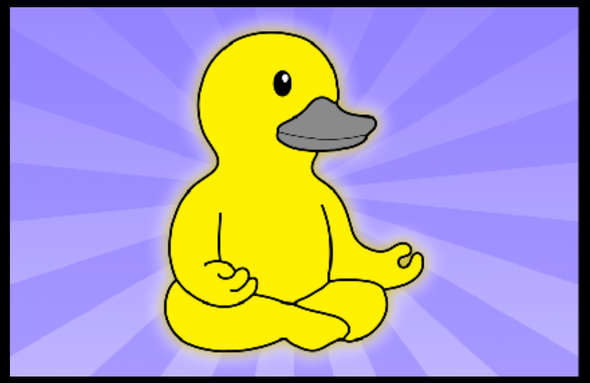
Text files



Binary files

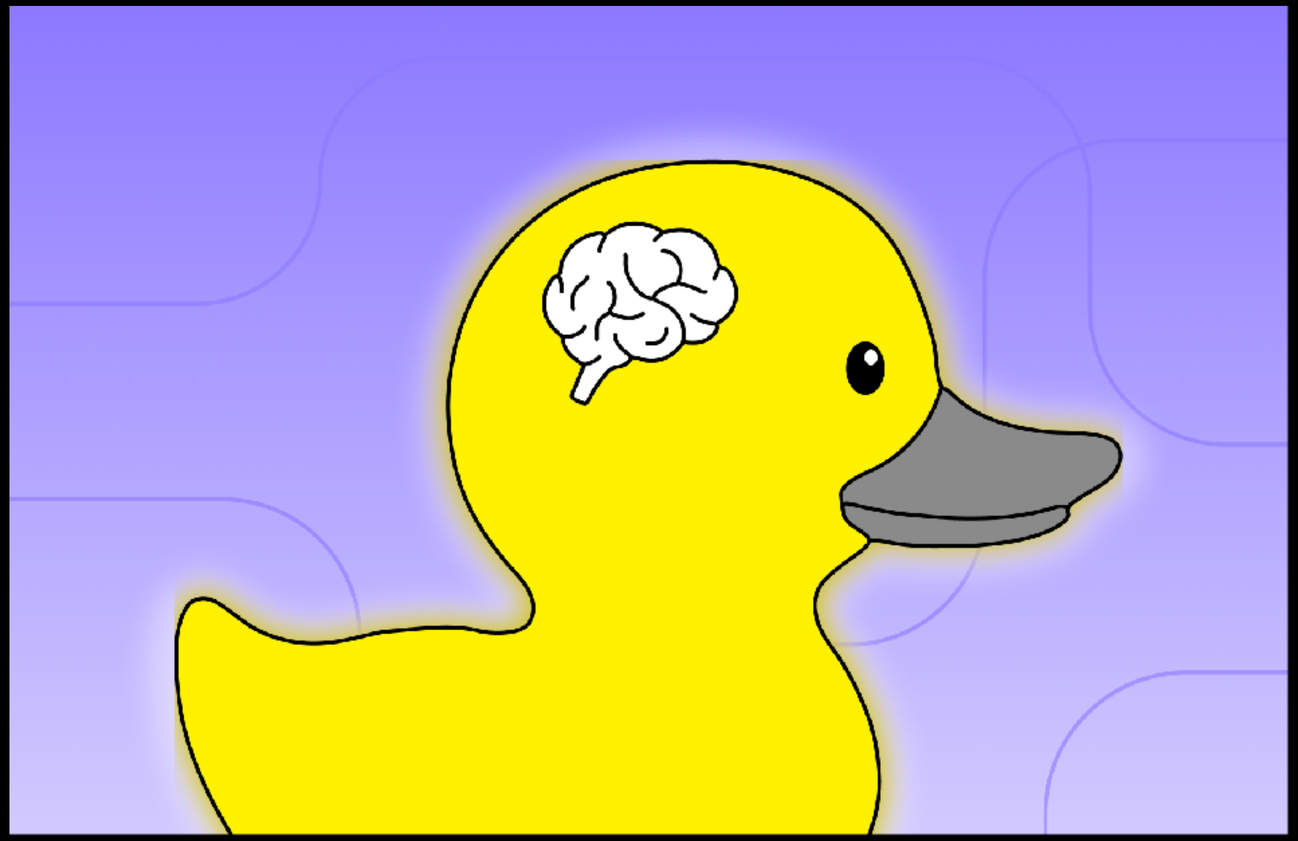


Database

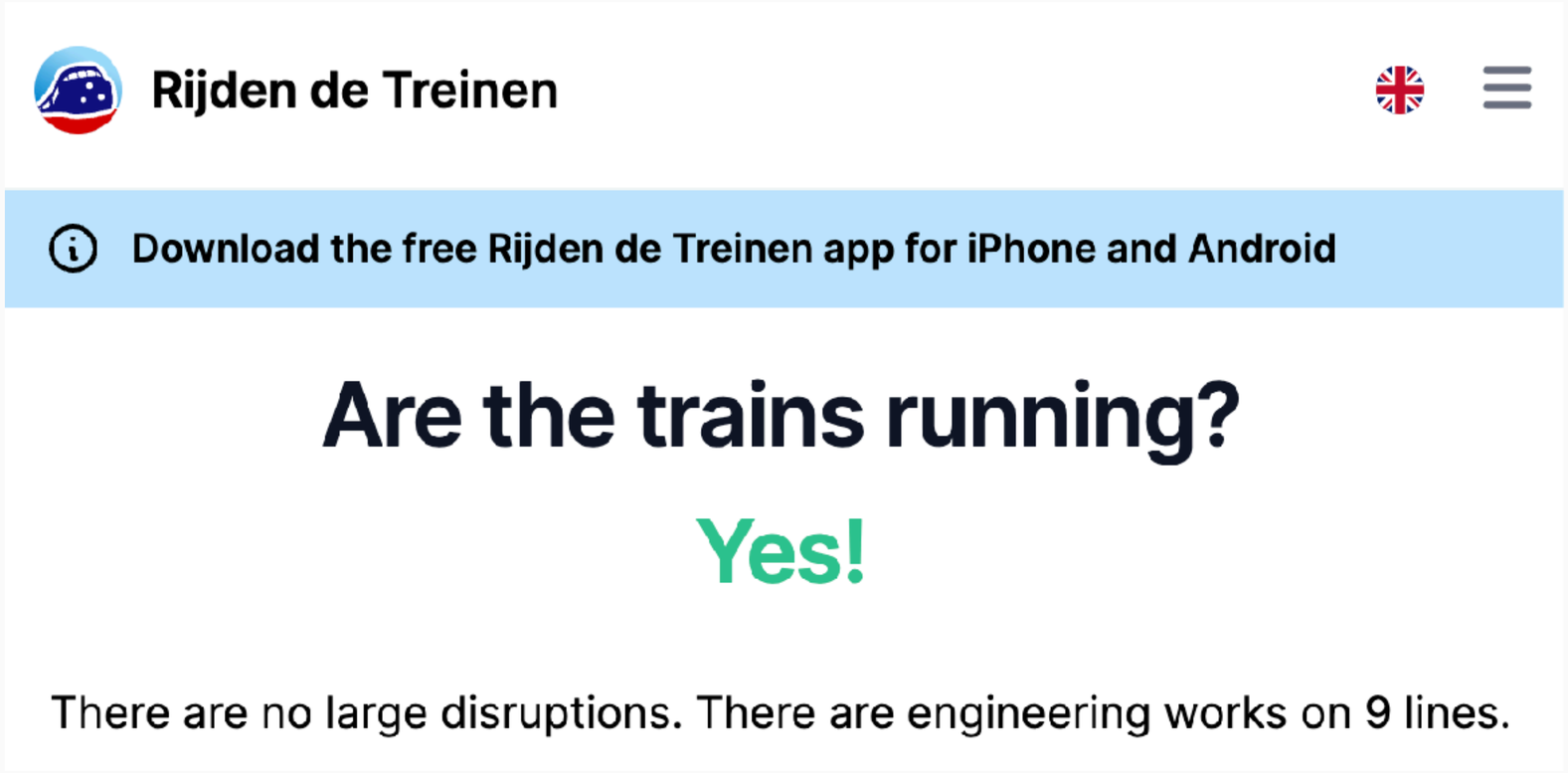


Data lake

Text files



Open dataset: Railway services



Services between
2019 and 2025

150 million record
as CSV files

2.3 GB compressed,
20.0 GB uncompressed

Open dataset: Railway services

id int64	date date	station varchar	delay int64
738804	2019-01-01	Rotterdam Centraal	NULL
738804	2019-01-01	Delft	0
738804	2019-01-01	Den Haag HS	1
738804	2019-01-01	Leiden Centraal	0
738804	2019-01-01	Schiphol Airport	0

Open dataset: Railway services

id int64	date date	station varchar	delay int64
738804	2019-01-01	Rotterdam Centraal	NULL
738804	2019-01-01	Delft	0
738804	2019-01-01	Den Haag HS	1
738804	2019-01-01	Leiden Centraal	0
738804	2019-01-01	Schiphol Airport	0

arrival delay

Open dataset: Railway services

id int64	date date	station varchar	delay int64
738804	2019-01-01	Rotterdam Centraal	NULL
738804	2019-01-01	Delft	0
738804	2019-01-01	Den Haag HS	1
738804	2019-01-01	Leiden Centraal	0
738804	2019-01-01	Schiphol Airport	0

arrival delay

Q

Which train stations had the worst delays on the weekends of Feb 2025?

Q

Which train stations had the worst delays on the weekends of Feb 2025?

```python
import pandas as pd

df = pd.read_csv("services.csv")
df["date"] = pd.to_datetime(df["date"])

result = (
    df[
        df["date"].between("2025-02-01", "2025-02-28") &
        df["date"].dt.isocalendar().day.isin([6, 7])
    ]
    .groupby("station", as_index=False)["delay"]
    .mean()
    .rename(columns={"delay": "avg_delay"})
    .sort_values("avg_delay", ascending=False)
    .head(3)
)
```

```python
import pandas as pd

df = pd.read_csv("services.csv")
df["date"] = pd.to_datetime(df["date"])

result = (
    df[
        df["date"].between("2025-02-01", "2025-02-28") &
        df["date"].dt.isocalendar().day.isin([6, 7])
    ]
    .groupby("station", as_index=False)["delay"]
    .mean()
    .rename(columns={"delay": "avg_delay"})
    .sort_values("avg_delay", ascending=False)
    .head(3)
)
```

Q

Which train stations had the worst delays on the weekends of Feb 2025?

```python
import pandas as pd

df = pd.read_csv("services.csv")
df["date"] = pd.to_datetime(df["date"])

result = (
    df[
        df["date"].between("2025-02-01", "2025-02-28") &
        df["date"].dt.isocalendar().day.isin([6, 7])
    ]
    .groupby("station", as_index=False)["delay"]
    .mean()
    .rename(columns={"delay": "avg_delay"})
    .sort_values("avg_delay", ascending=False)
    .head(3)
)
```

Q

Which train stations had the worst delays on the weekends of Feb 2025?

Q

Which train stations had the worst delays on the weekends of Feb 2025?

```python
import pandas as pd

df = pd.read_csv("services.csv")
df["date"] = pd.to_datetime(df["date"])

result = (
    df[
        df["date"].between("2025-02-01", "2025-02-28") &
        df["date"].dt.isocalendar().day.isin([6, 7])
    ]
    .groupby("station", as_index=False)["delay"]
    .mean()
    .rename(columns={"delay": "avg_delay"})
    .sort_values("avg_delay", ascending=False)
    .head(3)
)
```

Q

Which train stations had the worst delays on the weekends of Feb 2025?

```python
import pandas as pd

df = pd.read_csv("services.csv")
df["date"] = pd.to_datetime(df["date"])

result = (
    df[
        df["date"].between("2025-02-01", "2025-02-28") &
        df["date"].dt.isocalendar().day.isin([6, 7])
    ]
    .groupby("station", as_index=False)["delay"]
    .mean()
    .rename(columns={"delay": "avg_delay"})
    .sort_values("avg_delay", ascending=False)
    .head(3)
)
```

Q Which train stations had the worst delays on the weekends of Feb 2025?

```python
import pandas as pd

df = pd.read_csv("services.csv")
df["date"] = pd.to_datetime(df["date"])

result = (
    df[
        df["date"].between("2025-02-01", "2025-02-28") &
        df["date"].dt.isocalendar().day.isin([6, 7])
    ]
    .groupby("station", as_index=False)["delay"]
    .mean()
    .rename(columns={"delay": "avg_delay"})
    .sort_values("avg_delay", ascending=False)
    .head(3)
)
```

```
            station  avg_delay
415   Siegburg/Bonn   5.583333
177  Emmerich-Elten   3.965398
103    Brussel-Zuid   3.864253
```

Q Which train stations had the worst delays on the weekends of Feb 2025?

```python
import pandas as pd

df = pd.read_csv("services.csv")
df["date"] = pd.to_datetime(df["date"])

result = (
    df[
        df["date"].between("2025-02-01", "2025-02-28") &
        df["date"].dt.isocalendar().day.isin([6, 7])
    ]
    .groupby("station", as_index=False)["delay"]
    .mean()
    .rename(columns={"delay": "avg_delay"})
    .sort_values("avg_delay", ascending=False)
    .head(3)
)
```

```
            station  avg_delay
415   Siegburg/Bonn   5.583333
177  Emmerich-Elten   3.965398
103    Brussel-Zuid   3.864253
```

184 s

pandas

Q Which train stations had the worst delays on the weekends of Feb 2025?

```python
import pandas as pd

df = pd.read_csv("services.csv")
df["date"] = pd.to_datetime(df["date"])

result = (
    df[
        df["date"].between("2025-02-01", "2025-02-28") &
        df["date"].dt.isocalendar().day.isin([6, 7])
    ]
    .groupby("station", as_index=False)["delay"]
    .mean()
    .rename(columns={"delay": "avg_delay"})
    .sort_values("avg_delay", ascending=False)
    .head(3)
)
```

```
             station  avg_delay
415   Siegburg/Bonn   5.583333
177  Emmerich-Elten   3.965398
103    Brussel-Zuid   3.864253
```


184 s

pandas


65 s

Polars

```
$ duckdb
```

Q

Which train stations had the worst delays on the weekends of Feb 2025?

Q Which train stations had the worst delays on the weekends of Feb 2025?

```
$ duckdb

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM 'services.csv'
  WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

Q Which train stations had the worst delays on the weekends of Feb 2025?

```
$ duckdb
```

```
D SELECT
    station,
    avg(delay) AS avg_delay
  FROM 'services.csv'
  WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

Q Which train stations had the worst delays on the weekends of Feb 2025?

```
$ duckdb

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM 'services.csv'
  WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

Q Which train stations had the worst delays on the weekends of Feb 2025?

```
$ duckdb

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM 'services.csv'
  WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

Q Which train stations had the worst delays on the weekends of Feb 2025?

```
$ duckdb

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM 'services.csv'
  WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

Q Which train stations had the worst delays on the weekends of Feb 2025?

```
$ duckdb

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM 'services.csv'
  WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

station	avg_delay
Siegburg/Bonn	5.58
Emmerich-Elten	3.97
Brussel-Zuid	3.86

Q Which train stations had the worst delays on the weekends of Feb 2025?

```
$ duckdb

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM 'services.csv'
  WHERE date BETWEEN '2025-02-01' AND '2025-02-
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

station	avg_delay
Siegburg/Bonn	5.58
Emmerich-Elten	3.97
Brussel-Zuid	3.86


184 s


65 s

pandas

Polars

Q Which train stations had the worst delays on the weekends of Feb 2025?

```
$ duckdb

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM 'services.csv'
  WHERE date BETWEEN '2025-02-01' AND '2025-02-
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

station	avg_delay
Siegburg/Bonn	5.58
Emmerich-Elten	3.97
Brussel-Zuid	3.86


184 s

pandas


65 s

Polars


7 s

DuckDB

```
$ duckdb

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM 'services.csv'
  WHERE date BETWEEN '2025-02-01' AND '2025-02-
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

station	avg_delay

Q Which train stations had the worst delays on the weekends of Feb 2025?

184 s

pandas

65 s

Polars

7 s

DuckDB

How about using Python?

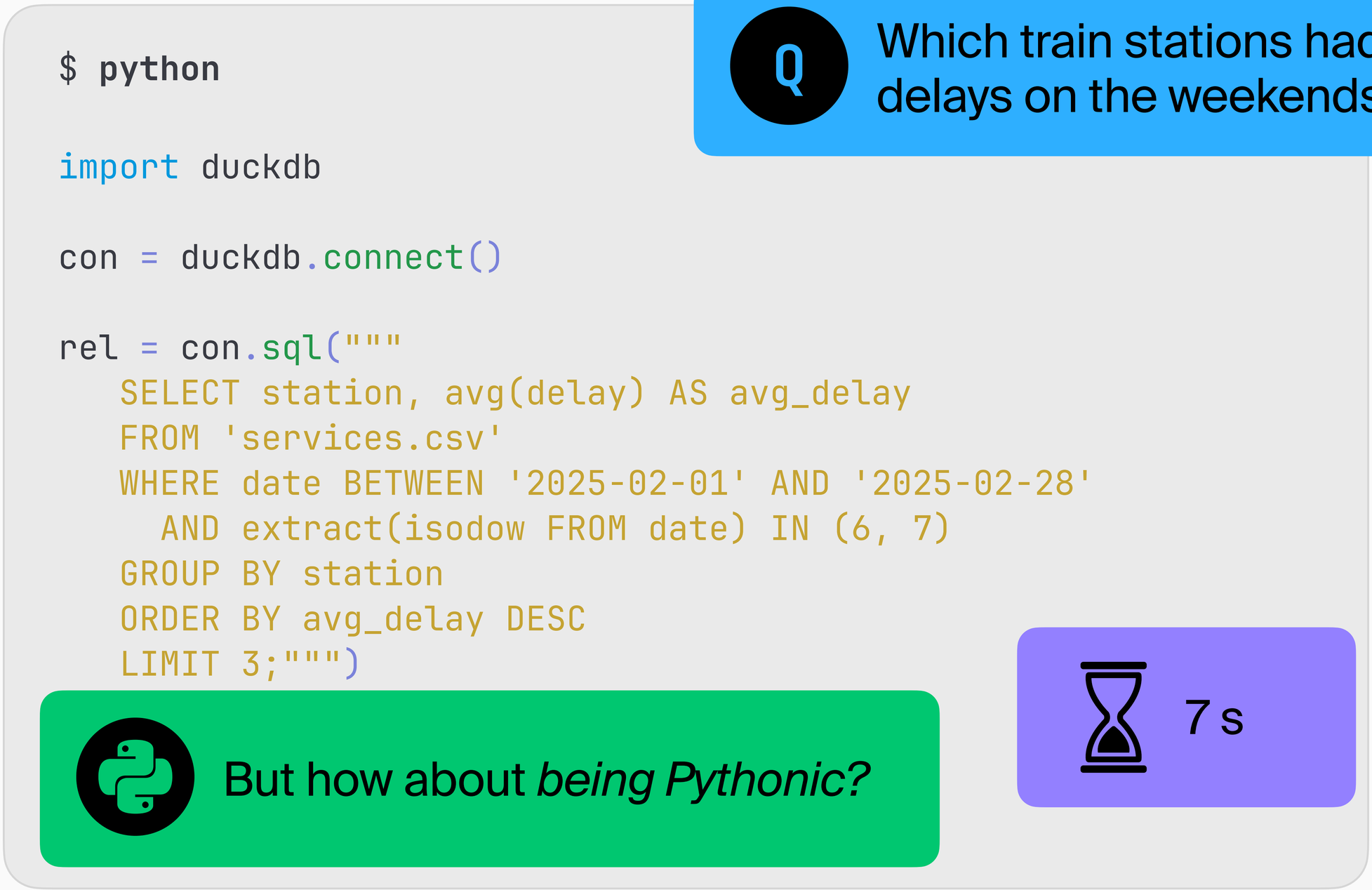
Q Which train stations had the worst delays on the weekends of Feb 2025?

```
$ python

import duckdb

con = duckdb.connect()

rel = con.sql("""
   SELECT station, avg(delay) AS avg_delay
   FROM 'services.csv'
   WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
     AND extract(isodow FROM date) IN (6, 7)
   GROUP BY station
   ORDER BY avg_delay DESC
   LIMIT 3;""")

rel.show()
```

```
$ python

import duckdb

con = duckdb.connect()

rel = con.sql("""
   SELECT station, avg(delay) AS avg_delay
   FROM 'services.csv'
   WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
     AND extract(isodow FROM date) IN (6, 7)
   GROUP BY station
   ORDER BY avg_delay DESC
   LIMIT 3;""")

rel.show()
```

Q Which train stations had the worst delays on the weekends of Feb 2025?

7 s


$ python
import duckdb
con = duckdb.connect()
rel = con.sql("""
SELECT station, avg(delay) AS avg_delay
FROM 'services.csv'
WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
AND extract(isodow FROM date) IN (6, 7)
GROUP BY station
ORDER BY avg_delay DESC
LIMIT 3;""")
Q
Which train stations had the worst delays on the weekends of Feb 2025?
But how about *being Pythonic?*
7 s

Q Which train stations had the worst delays on the weekends of Feb 2025?

```
$ python

import duckdb

con = duckdb.connect()

rel = (
    con.table("services.csv")
    .filter("""date BETWEEN '2025-02-01' AND '2025-02-28'
               AND extract(isodow FROM date) IN (6, 7)""")
    .aggregate("station, avg(delay) AS avg_delay", "station")
    .order("avg_delay DESC")
    .limit(3)
)

rel.show()
```

```
$ python

import duckdb

con = duckdb.connect()

rel = (
    con.table("services.csv")
    .filter("""date BETWEEN '2025-02-01' AND '2025-02-28'
               AND extract(isodow FROM date) IN (6, 7)""")
    .aggregate("station, avg(delay) AS avg_delay", "station")
    .order("avg_delay DESC")
    .limit(3)
)

rel.show()
```

Q Which train stations had the worst delays on the weekends of Feb 2025?

7 s

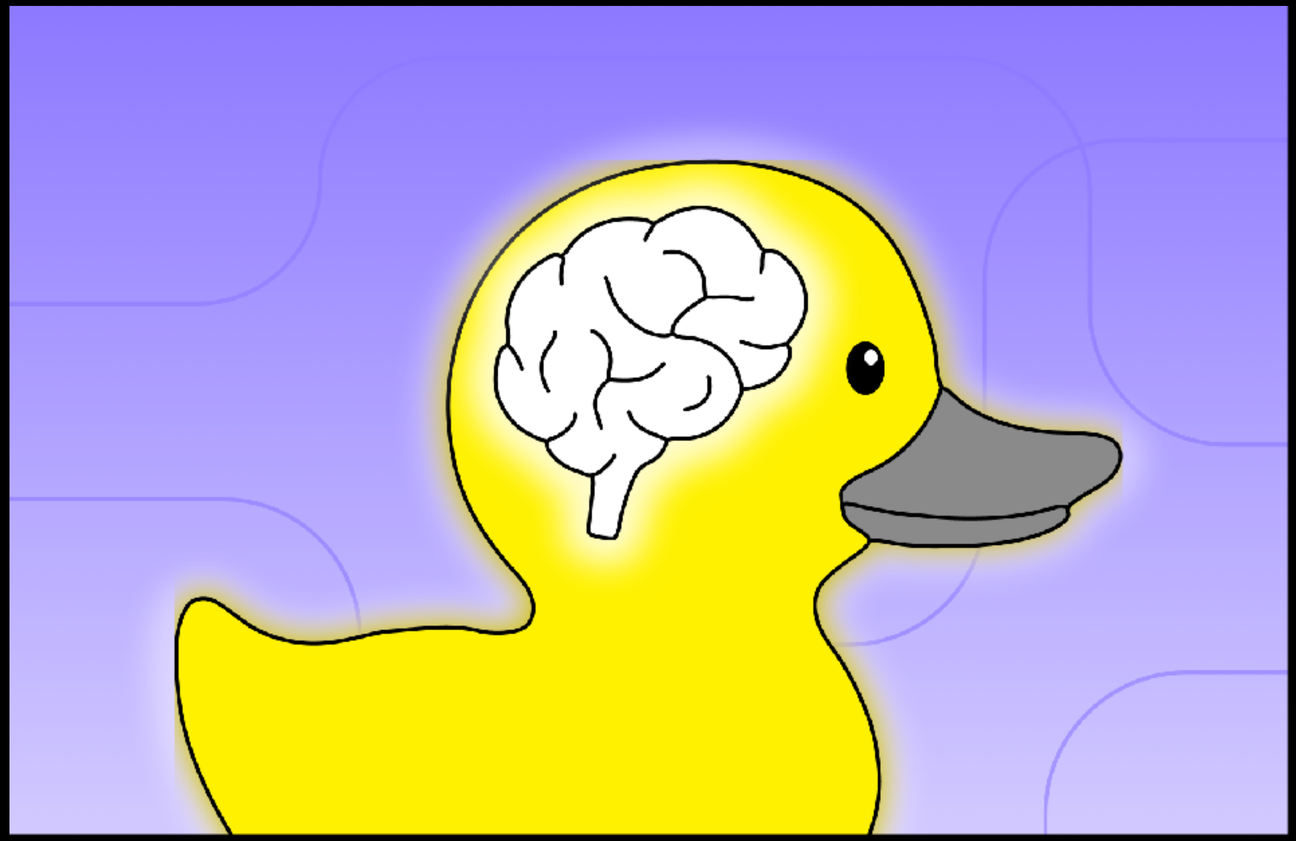
```
$ python

import duckdb

con = duckdb.connect()

rel = (
    con.table("services.csv")
    .filter("""date BETWEEN '2025-02-01' AND '2025-02-28'
               AND extract(isodow FROM date) IN (6, 7)""")
    .aggregate("station, avg(delay) AS avg_delay", "station")
    .order("avg_delay DESC")
    .limit(3)
)

rel.show()
```

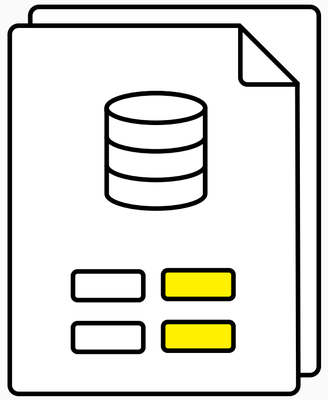
Q Which train stations had the worst delays on the weekends of Feb 2025?

 7 s

? Which one do you prefer?

Binary files





Tables on disk:

Row- or column-based storage?

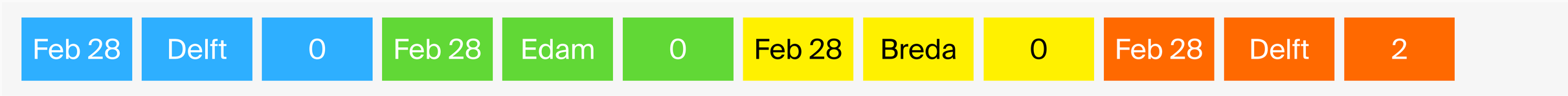
Row-based storage

date	station	delay
Feb 28	Delft	0
Feb 28	Edam	0
Feb 28	Breda	0
Feb 28	Delft	2

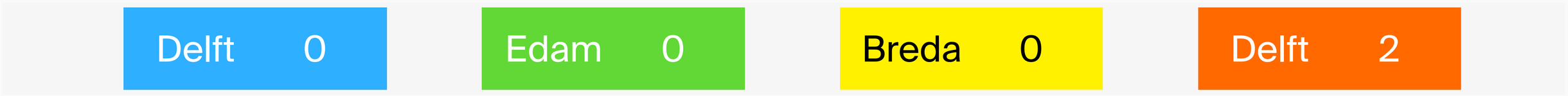
Row-based storage

date	station	delay
Feb 28	Delft	0
Feb 28	Edam	0
Feb 28	Breda	0
Feb 28	Delft	2

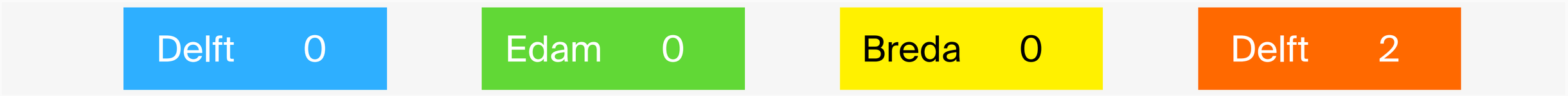
Row-based storage



Row-based storage



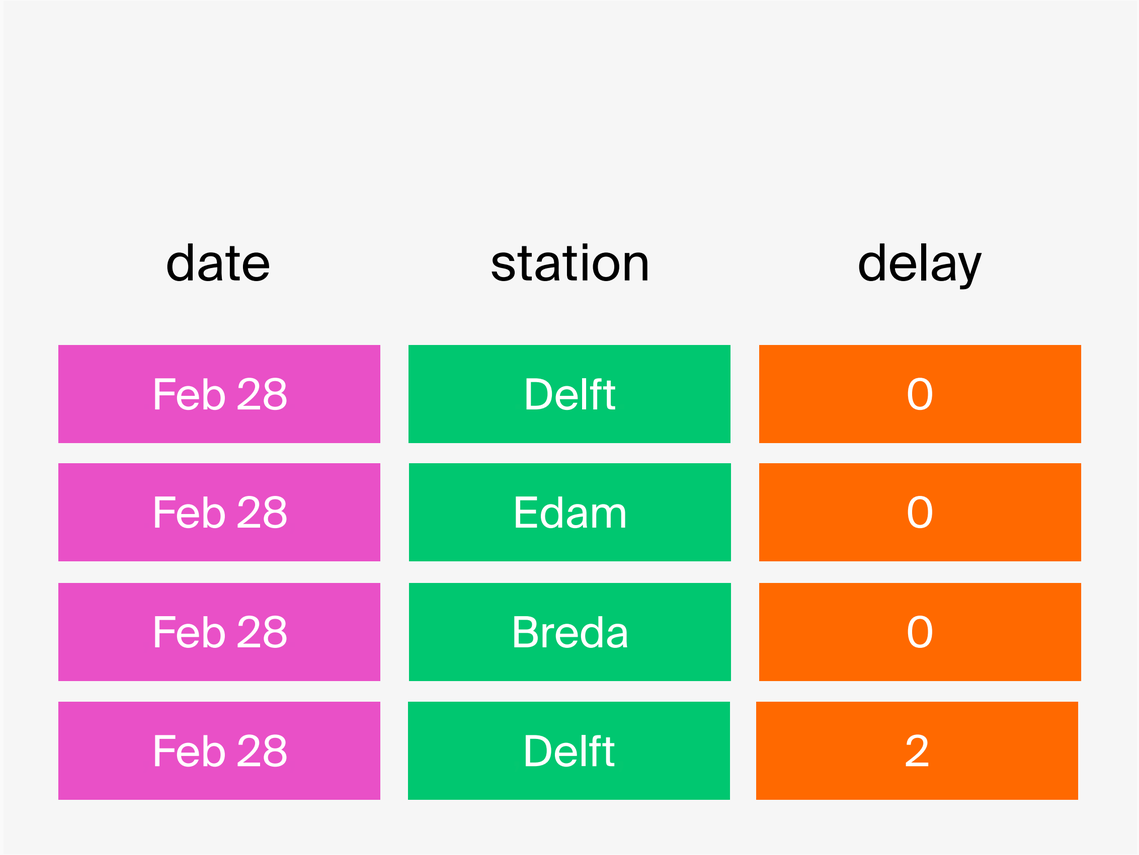
Row-based storage



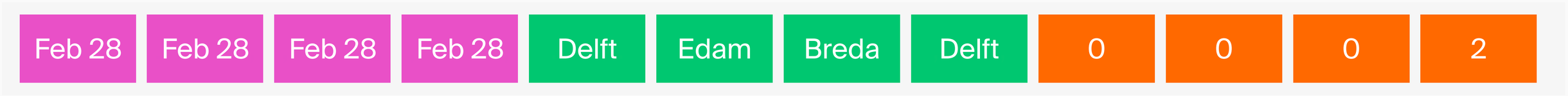
transactional databases, e.g., PostgreSQL

Avro format

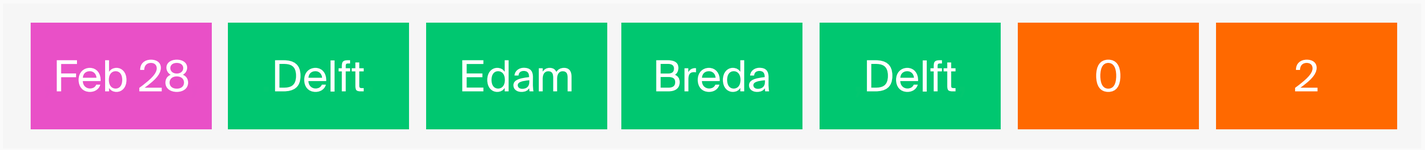
Column-based storage



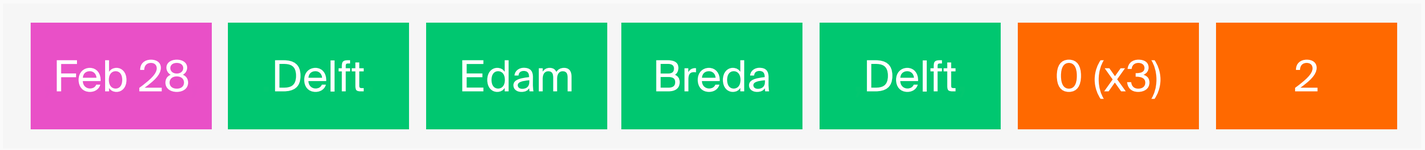
Column-based storage



Column-based storage



Column-based storage



Column-based storage

Column-based storage

analytical databases, e.g., DuckDB

Parquet format

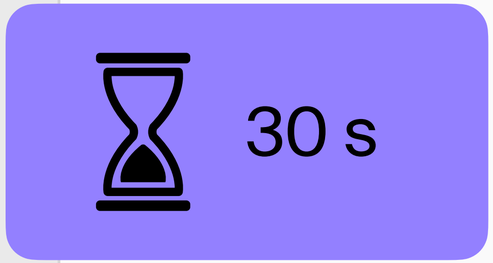
Column-based storage

analytical databases, e.g., DuckDB

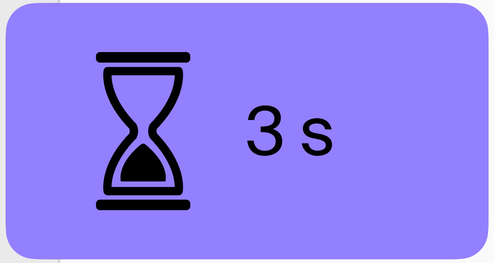
Parquet format

? Who has heard of Apache Parquet?

```
$ python pandas-analysis-parquet.py
```

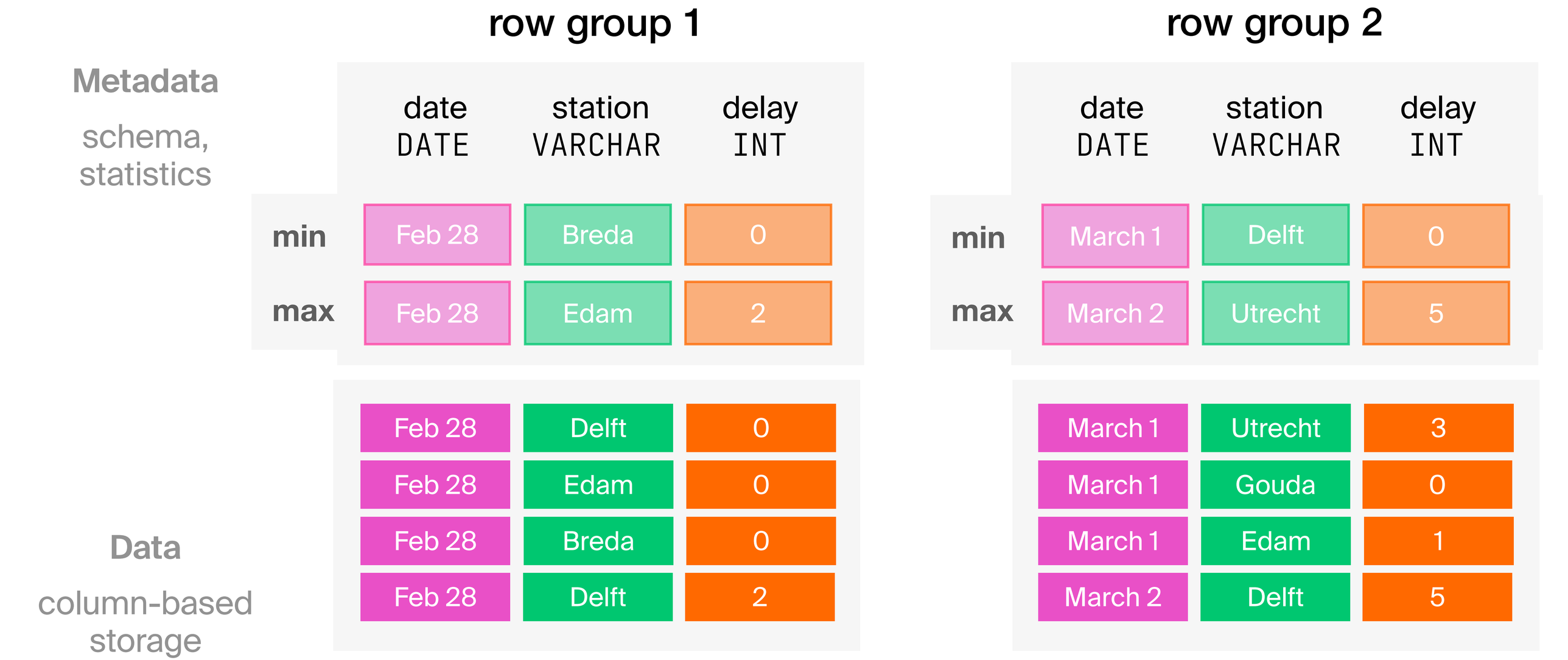



```
$ python polars-analysis-parquet.py
```




```
DuckDB: SELECT … FROM '….parquet';
```

Parquet format



Parquet format

Metadata

schema, statistics

Data

column-based storage

Built-in min-max indexes

row group 1

	date DATE	station VARCHAR	delay INT
min	Feb 28	Breda	0
max	Feb 28	Edam	2

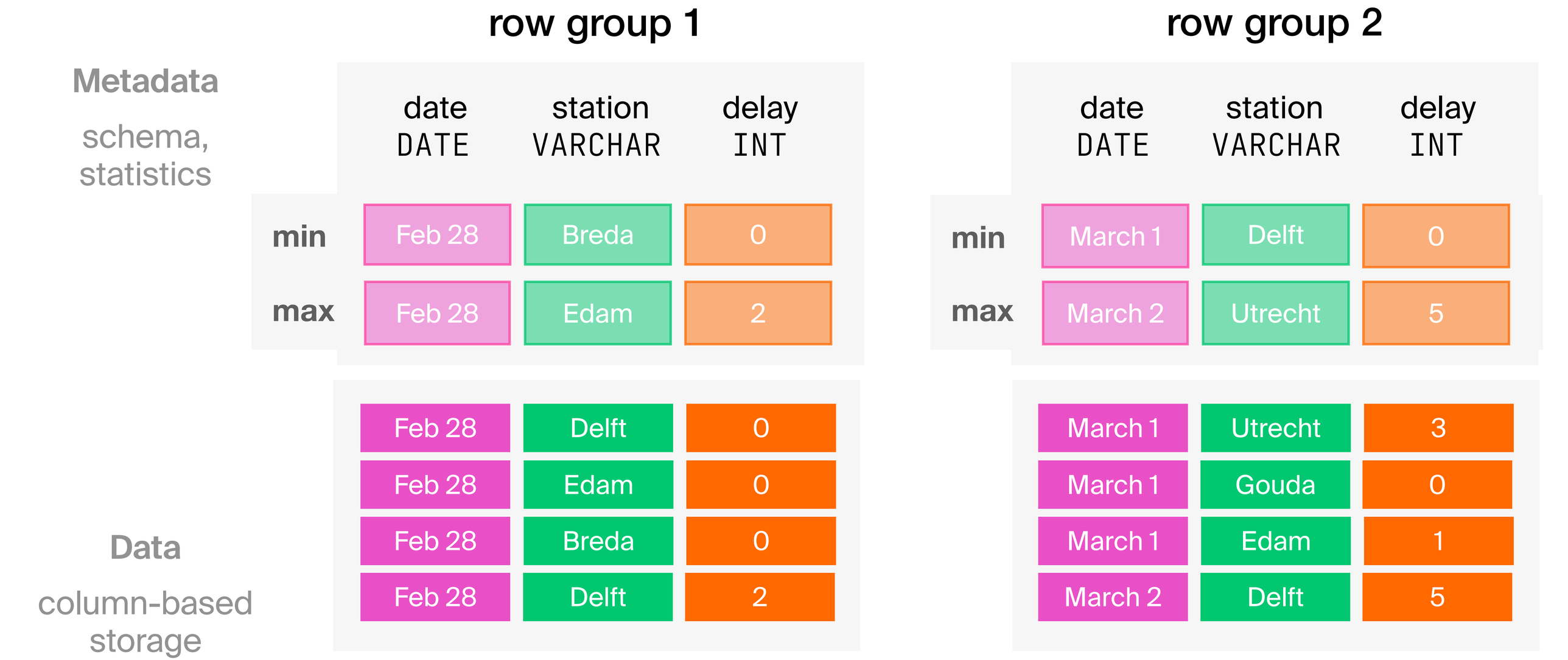
Feb 28	Delft	0
Feb 28	Edam	0
Feb 28	Breda	0
Feb 28	Delft	2

row group 2

	date DATE	station VARCHAR	delay INT
min	March 1	Delft	0
max	March 2	Utrecht	5

March 1	Utrecht	3
March 1	Gouda	0
March 1	Edam	1
March 2	Delft	5

Parquet format



Parquet format

Q Which train stations had the worst delays on the weekends of Feb (2025)?

row group 1

Metadata

schema, statistics

	date DATE	station VARCHAR	delay INT
min	Feb 28	Breda	0
max	Feb 28	Edam	2

Data

column-based storage

date DATE	station VARCHAR	delay INT
Feb 28	Delft	0
Feb 28	Edam	0
Feb 28	Breda	0
Feb 28	Delft	2

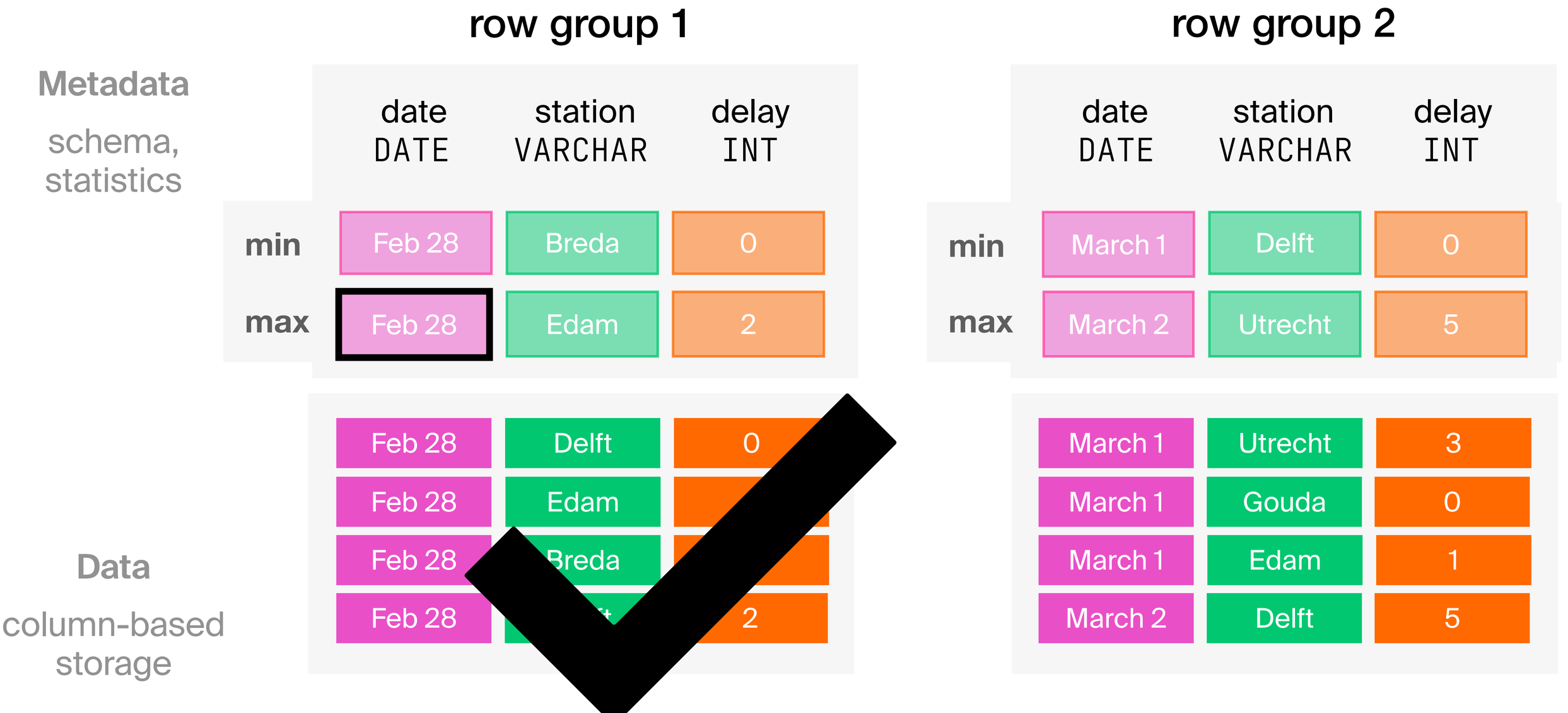
row group 2

	date DATE	station VARCHAR	delay INT
min	March 1	Delft	0
max	March 2	Utrecht	5

date DATE	station VARCHAR	delay INT
March 1	Utrecht	3
March 1	Gouda	0
March 1	Edam	1
March 2	Delft	5

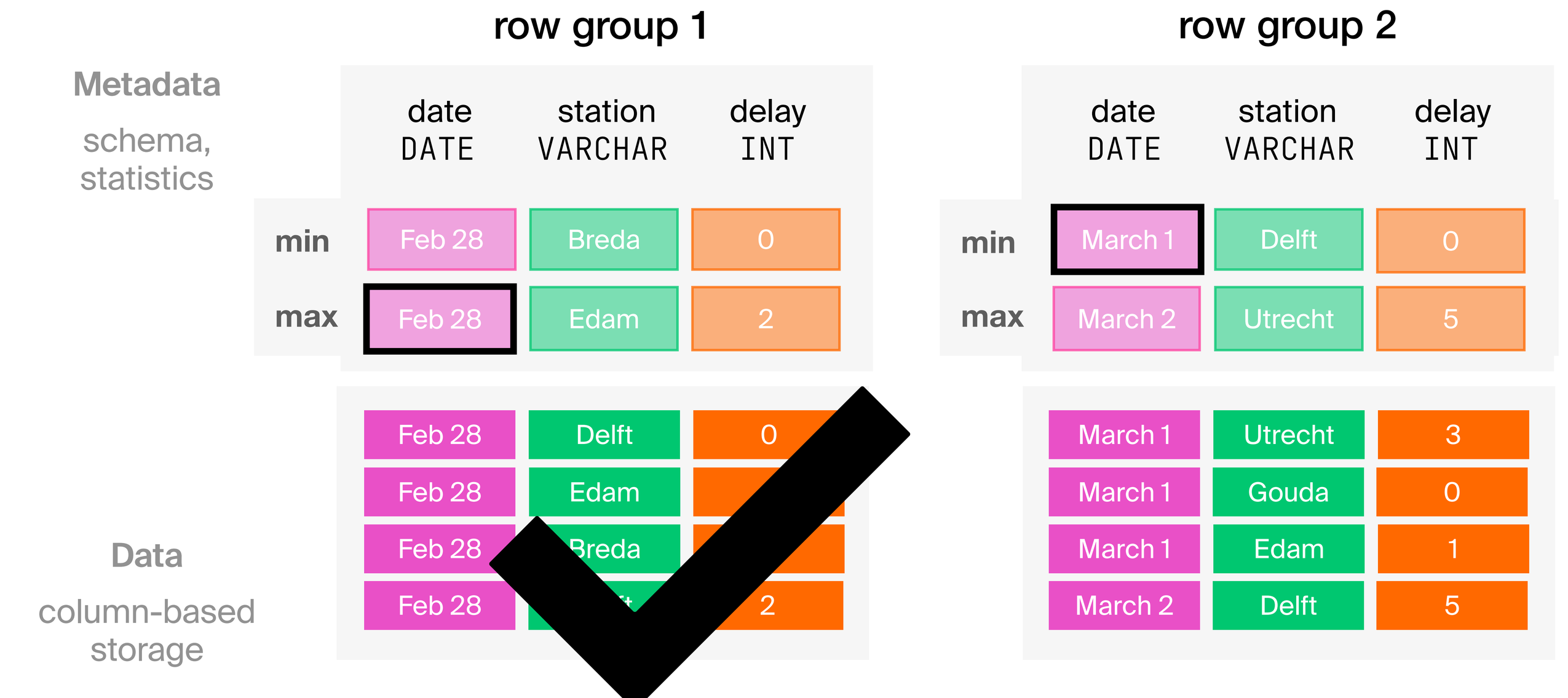
Parquet format

Q Which train stations had the worst delays on the weekends of Feb (2025)?



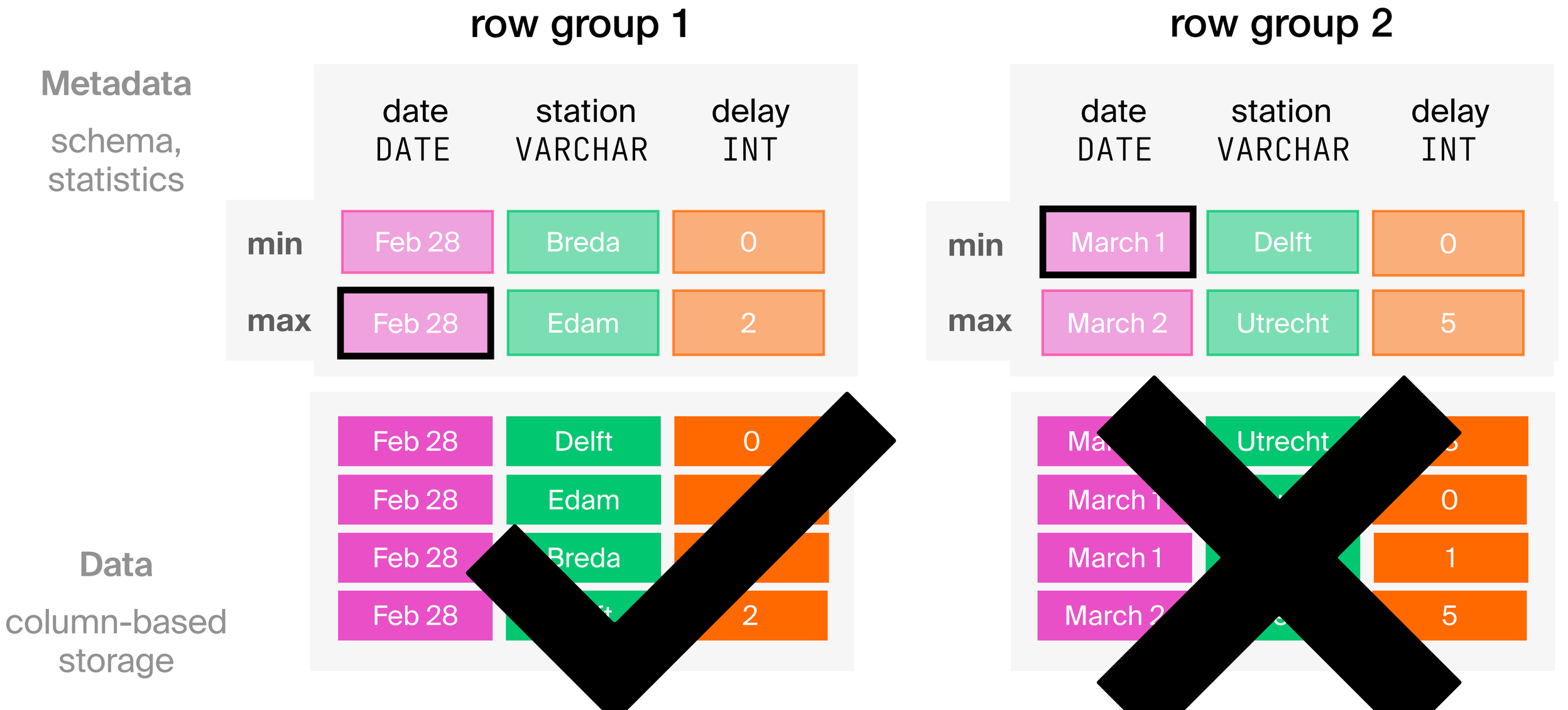
Parquet format

Q Which train stations had the worst delays on the weekends of Feb (2025)?

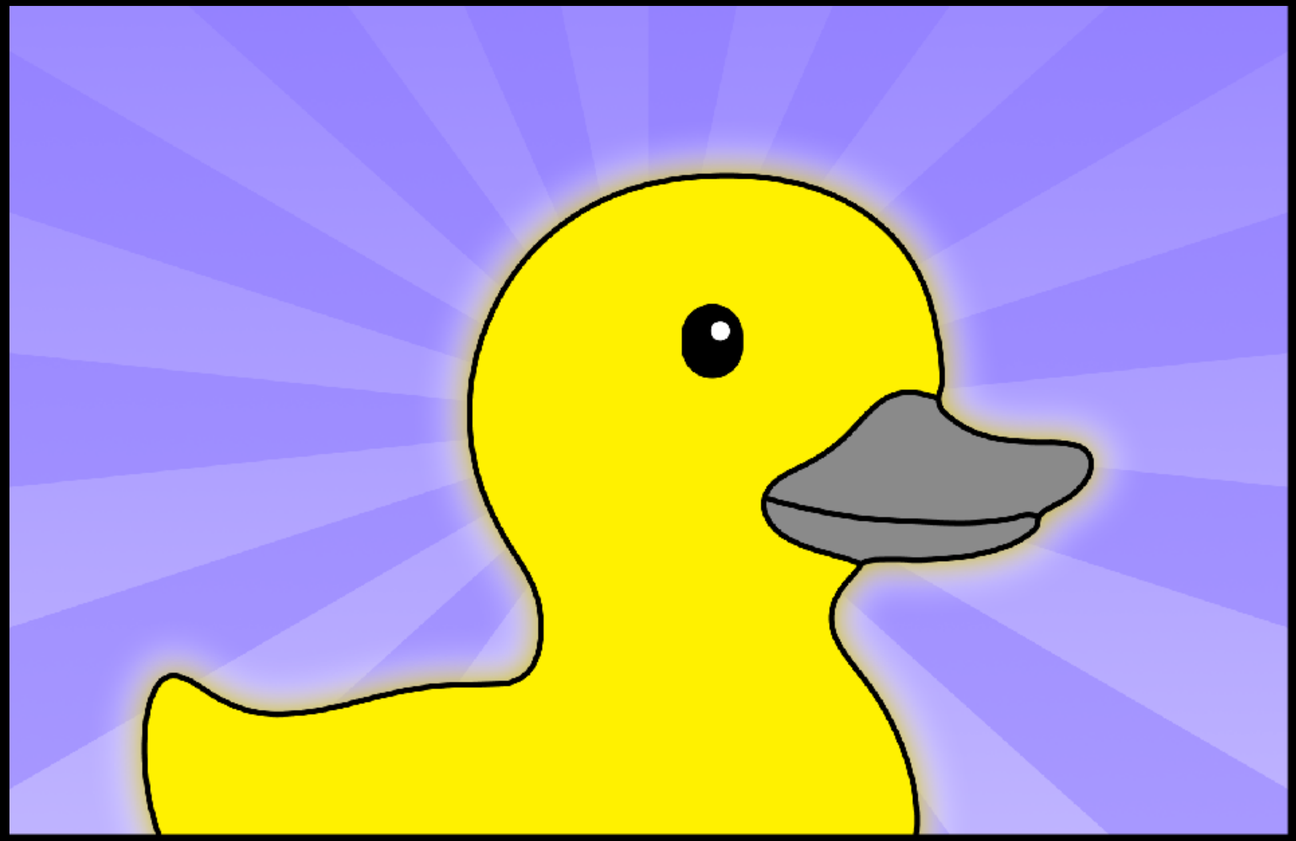


Parquet format

Q Which train stations had the worst delays on the weekends of Feb (2025)?



Database



```
$ duckdb services.duckdb

D CREATE TABLE services AS
  FROM 'services.csv';
```

```
$ duckdb services.duckdb

D CREATE TABLE services AS
  FROM 'services.csv';
```

17 s

Q Which train stations had the worst delays on the weekends of Feb 2025?

```
$ duckdb services.duckdb

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM services
  WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

station	avg_delay
Siegburg/Bonn	5.58
Emmerich-Elten	3.97
Brussel-Zuid	3.86

```
$ duckdb services.duckdb

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM services
  WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

station	avg_delay
Siegburg/Bonn	5.58
Emmerich-Elten	3.97
Brussel-Zuid	3.86

Which train stations had the worst delays on the weekends of Feb 2025?

0.03s

```
$ duckdb services.duckdb

D SELECT
    station,
    avg(delay) AS avg_delay
  FROM services
  WHERE date BETWEEN '2025-02-01' AND '2025-02-28'
    AND extract(isodow FROM date) IN (6, 7)
  GROUP BY station
  ORDER BY avg_delay DESC
  LIMIT 3;
```

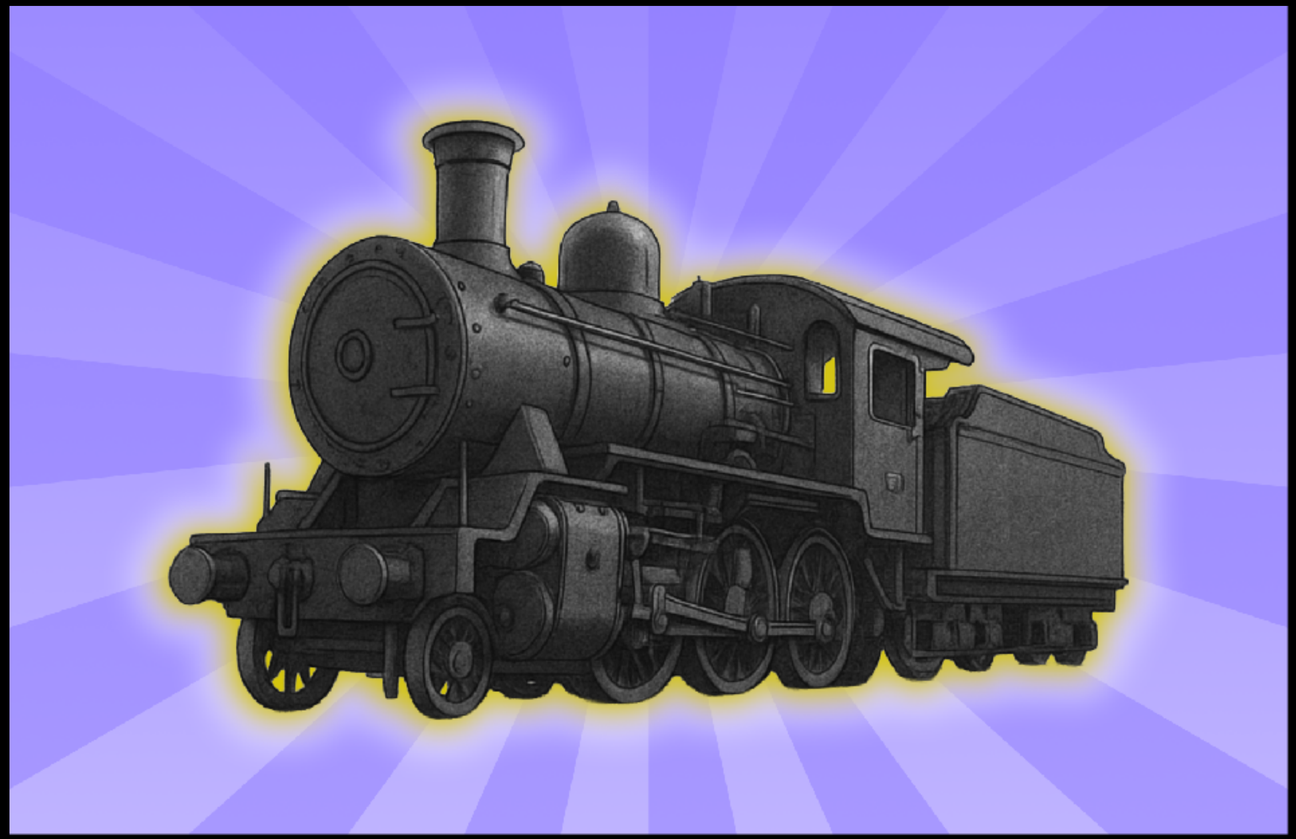
station	avg_delay
Siegburg/Bonn	5.58
Emmerich-Elten	3.97
Brussel-Zuid	3.86

Q Which train stations had the worst delays on the weekends of Feb 2025?

 0.03s

 The min-max indexes are created automatically for each row group

Railway operators



The biggest delay at Amsterdam Centraal is almost 3 hours!

```
SELECT date, train_number, delay
FROM services
WHERE date = '2025-02-27'
  AND station = 'Amsterdam Centraal'
  AND train_number = 420;
```

date	train_number	delay
2025-02-27	420	174

Let's renumber the services with large delays!

```
UPDATE services
SET train_number =
    train_number + (random() * 100 + 1)::INT
WHERE delay > 150;

(23 rows updated)
```

Let's renumber the services with large delays!

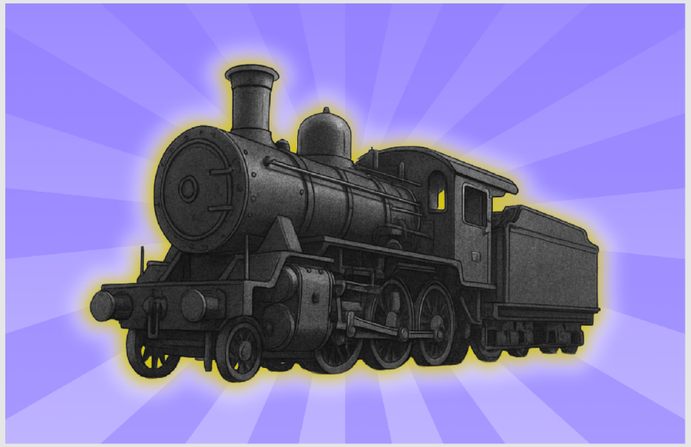
```
UPDATE services
SET train_number =
    train_number + (random() * 100 + 1)::INT
WHERE delay > 150;

(23 rows updated)

SELECT date, train_number, delay
FROM services
WHERE date = '2025-02-27'
  AND train_number = 420;
```

date	train_number	delay
0 rows		

Let's renumber the services with large delays!

```
UPDATE services
SET train_number =
    train_number + (random() * 100 + 1)::INT
WHERE delay > 150;

(23 rows updated)

SELECT date, train_number, delay
FROM services
WHERE date = '2025-02-27'
  AND train_number = 420;
```

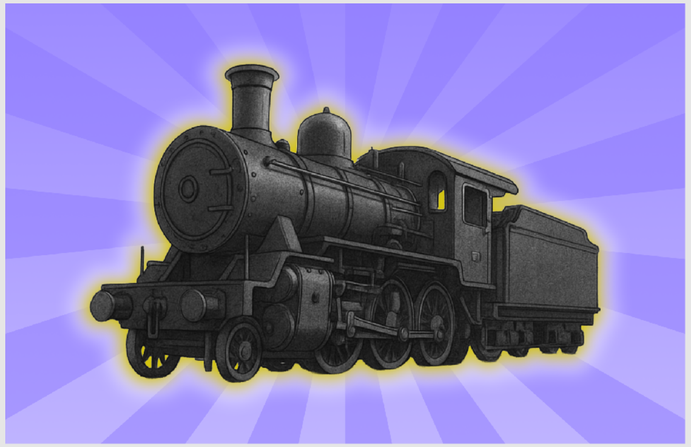
date	train_number	delay
0 rows		



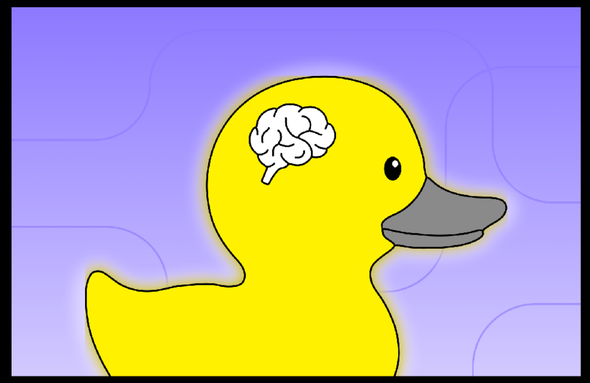
Let's renumber the services with large delays!

```
UPDATE services
SET train_number =
    train_number + (random() * 100 + 1)::INT
WHERE delay > 150;

(23 rows updated)

SELECT date, train_number, delay
FROM services
WHERE date = '2025-02-27'
  AND train_number = 420;
```

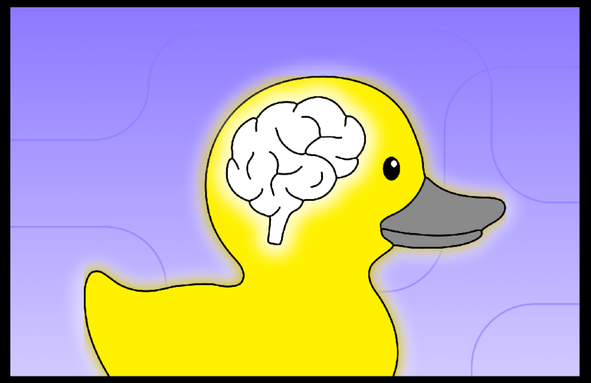
date	train_number	delay
0 rows		



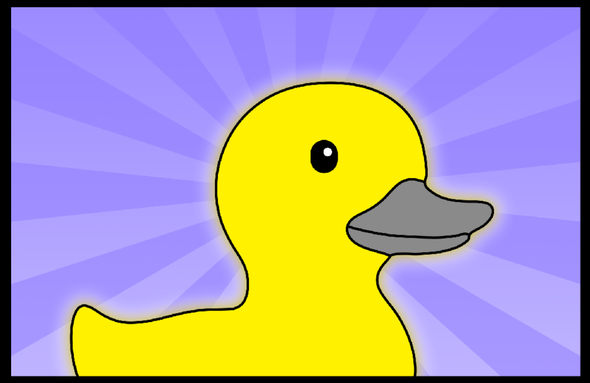
Databases are updatable!



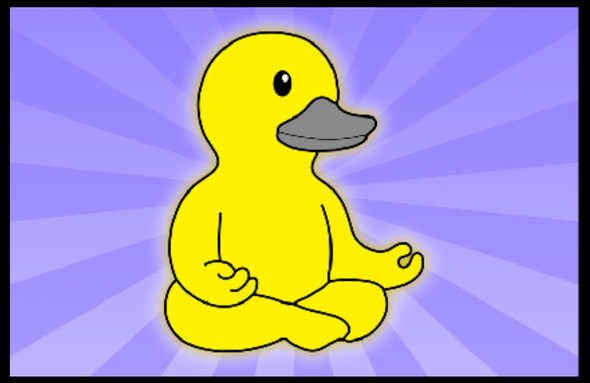
Text files



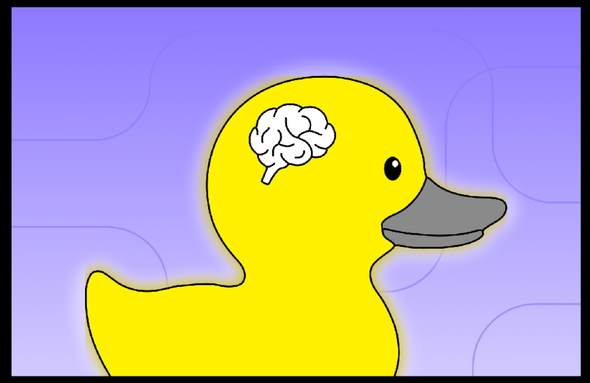
Binary files



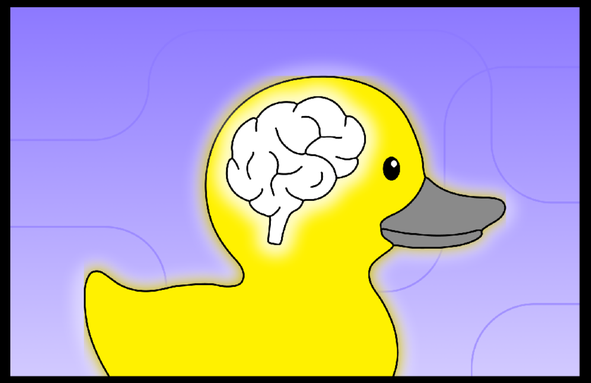
Database



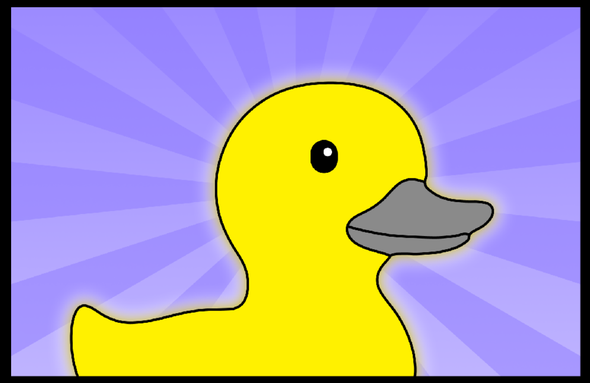
Data lake



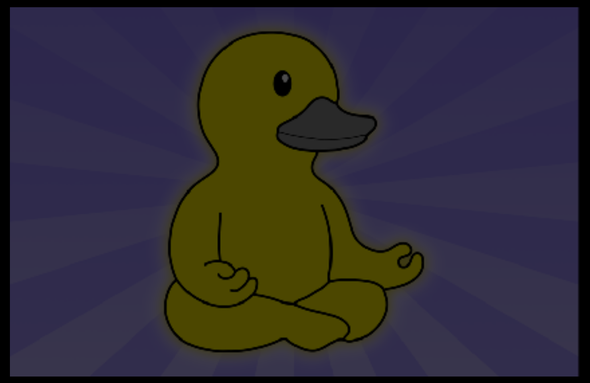
Text files

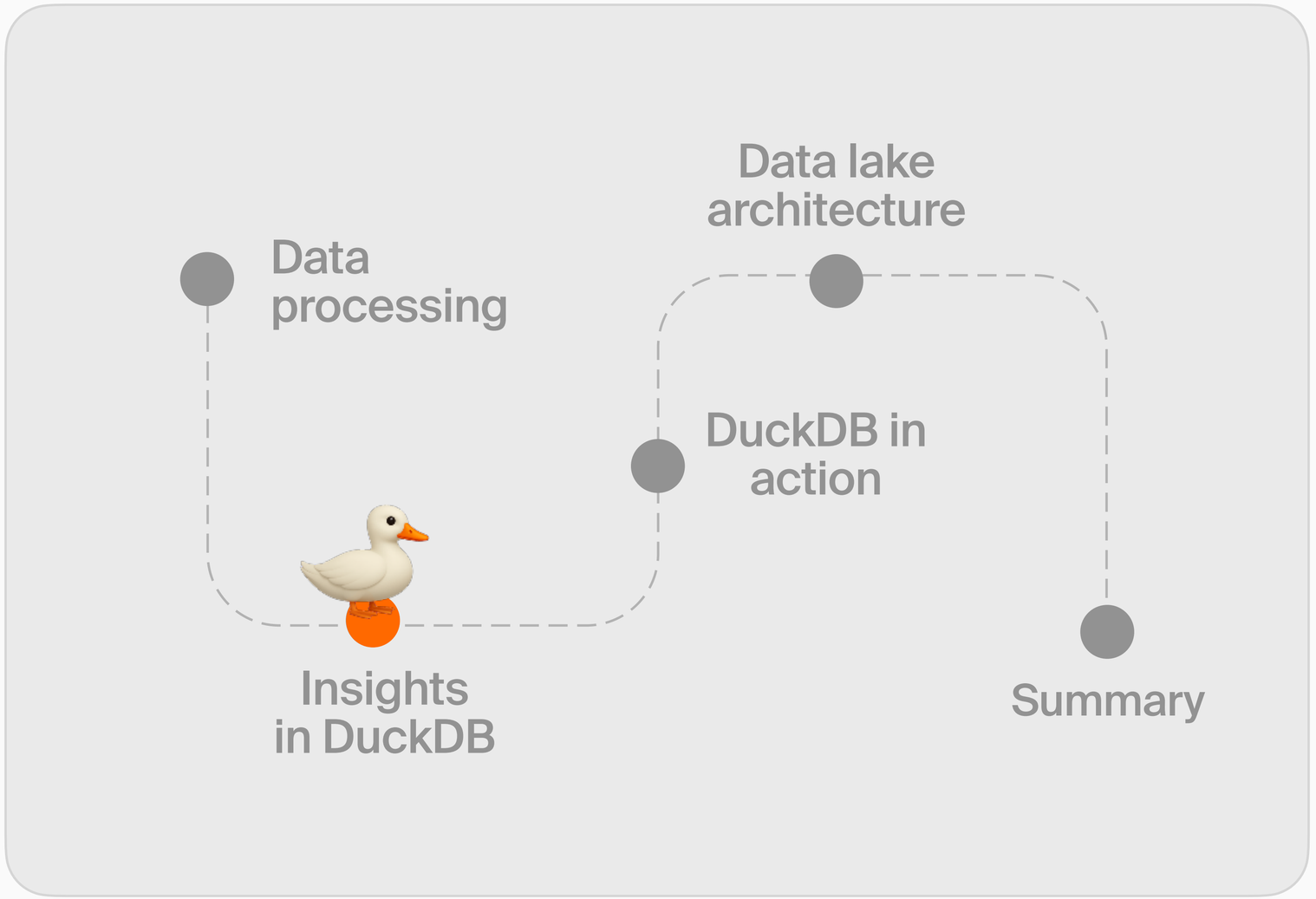


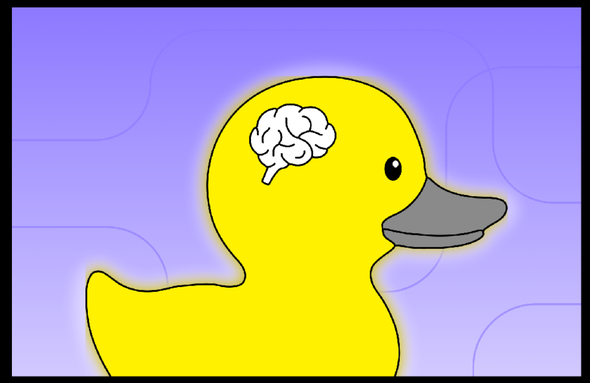
Binary files



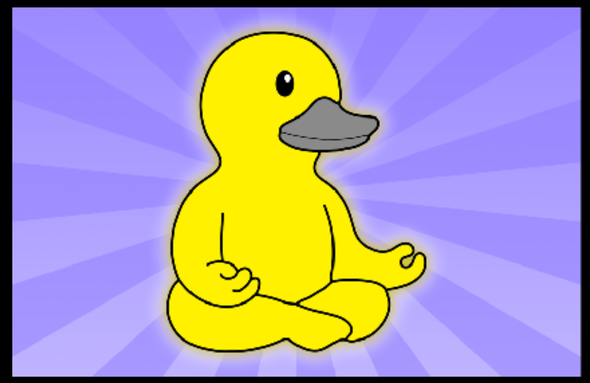
Database




Data lake architecture
Data processing
DuckDB in action
Insights in DuckDB
Summary



Text files



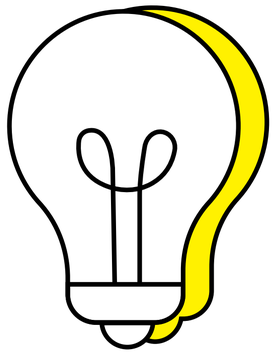
Database

Concise queries

Safe

Fast

Updatable



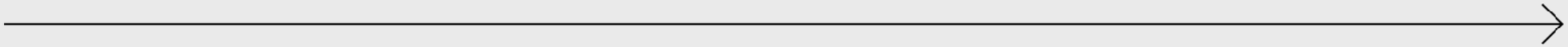
INSIGHT #1

Databases can target data science workloads

Data science landscape, 2016

?

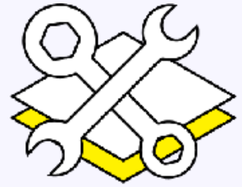
`data.tables`



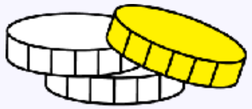
data size

Resistance: Why would I use a database?

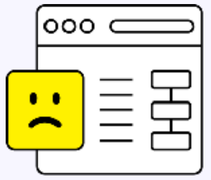
```
$ sudo apt install your_favorite_database
$ ... configure for 60 minutes ...
$ sudo service start your_favorite_database
$ wget https://blobs.duckdb.org/data/services.csv.gz
$ gunzip services.csv.gz
$ db_client -u admin -p admin

# CREATE TABLE ...; COPY ...; SELECT ...;
```



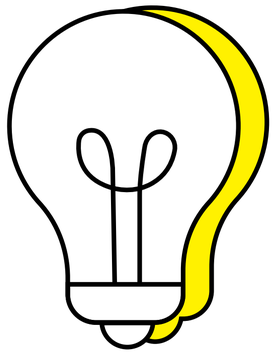
Difficult operation



Expensive



Bad UX



INSIGHT #2

Focus on usability

End-to-end performance

End-to-end performance

setup

define schema

write queries

load

run queries

End-to-end performance

setup

define schema

write queries

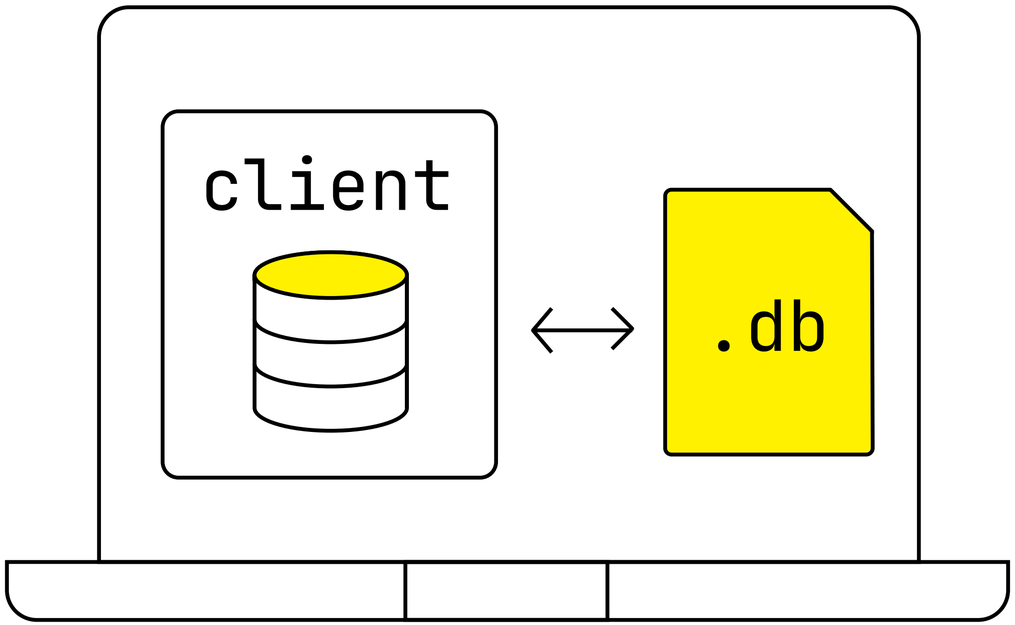
load

run queries

End-to-end performance

In-process architecture

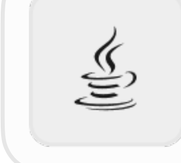
setup



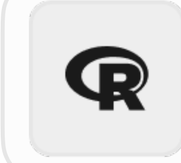
In-process architecture

```
pip install duckdb
```

```
curl https://install.duckdb.org | sh
```



```
go get github.com/duckdb/duckdb-go/v2
```



End-to-end performance

“Friendly” SQL

define schema

write queries

```
SELECT
    date, station, avg(delay),
--  min(delay)
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY ALL;
```

“Friendly” SQL

define schema

write queries

```
SELECT
    date, station, avg(delay),
--  min(delay)
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY ALL;
```

“Friendly” SQL

define schema

write queries

```
SELECT
    date, station, avg(delay),
--  min(delay)
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY ALL;
```

“Friendly” SQL

define schema

write queries

```
SELECT
    date, station, avg(delay),
--  min(delay)
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY ALL;
```

“Friendly” SQL

define schema

write queries

```sql
SELECT
    date, station, avg(delay),
--  min(delay)
FROM 'https://blobs.duckdb.org/data/services.csv.gz'
GROUP BY ALL;
```

DuckDB 2022

ClickHouse 2022

databricks 2023

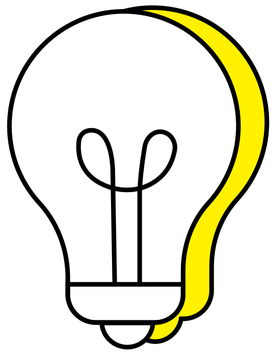
snowflake 2023

Google Big Query 2024

ORACLE 2025

PostgreSQL 2026

ISO SQL 2028 (?)

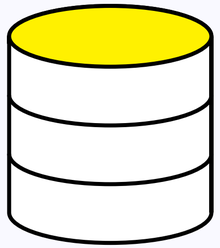


INSIGHT #3

“Big Data” is overrated

2011 Data processing landscape

Lots of data



Weak hardware

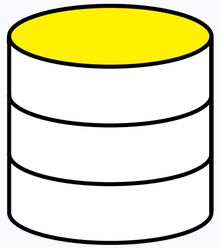
Inefficient software

```
  0[||||100.0%]   1[       0.0%]
  2[       0.0%]  3[       0.0%]
  4[       0.0%]  5[       0.0%]
  6[       0.0%]  7[       0.0%]
Mem[||||                362M/27.4G]
```

2026 Data processing landscape

Lots of data

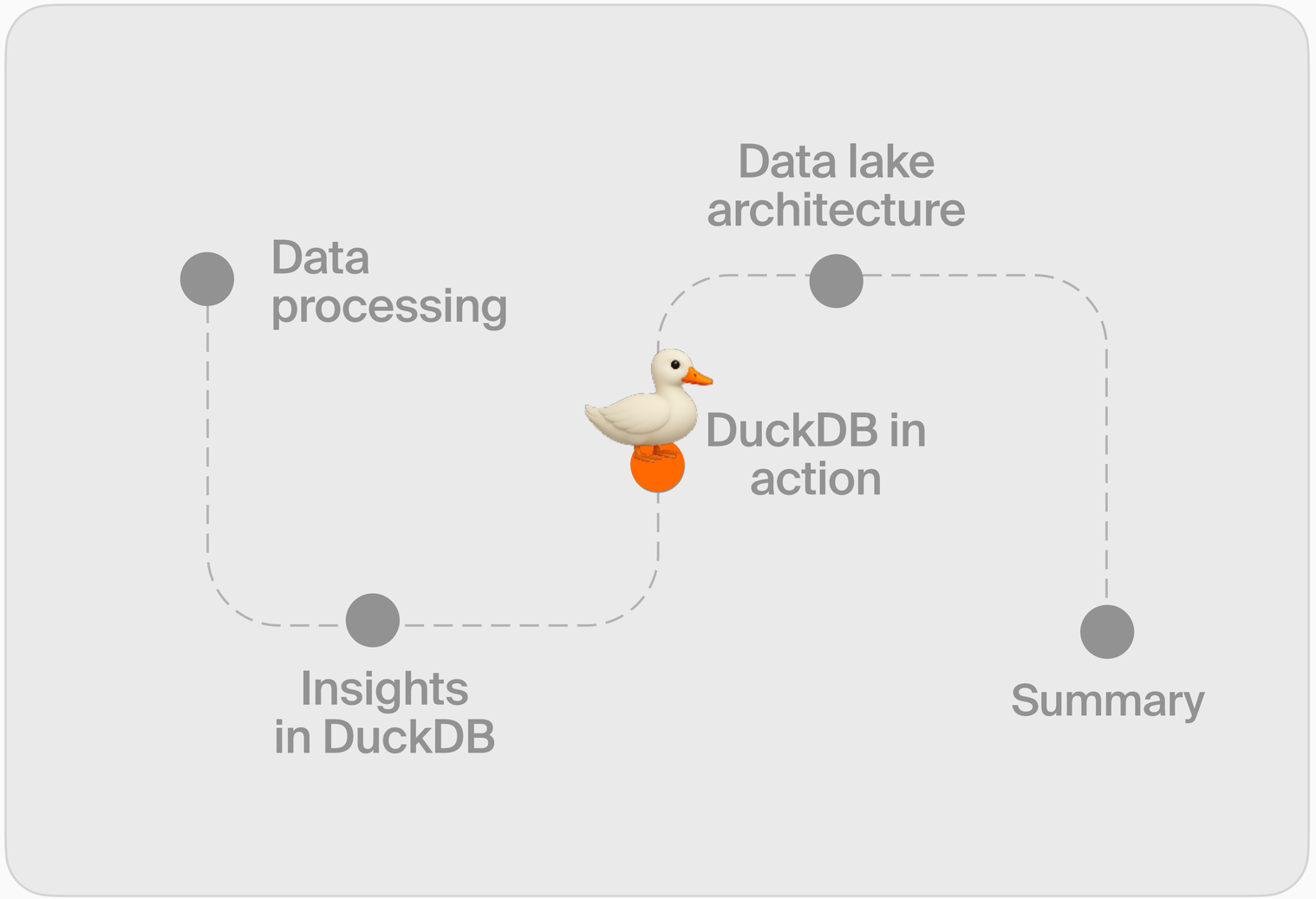
but most queries
only access a small amount

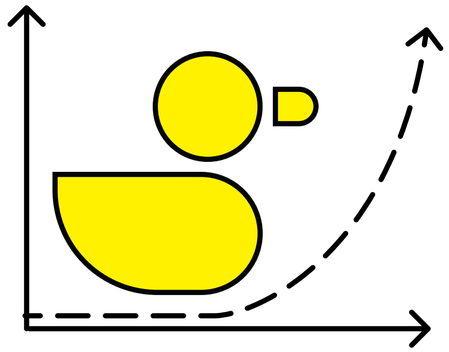


Strong hardware

New techniques

columnar storage,
lightweight compression,
vectorized query execution


Data lake architecture
Data processing
DuckDB in action
Insights in DuckDB
Summary



Performance & portability

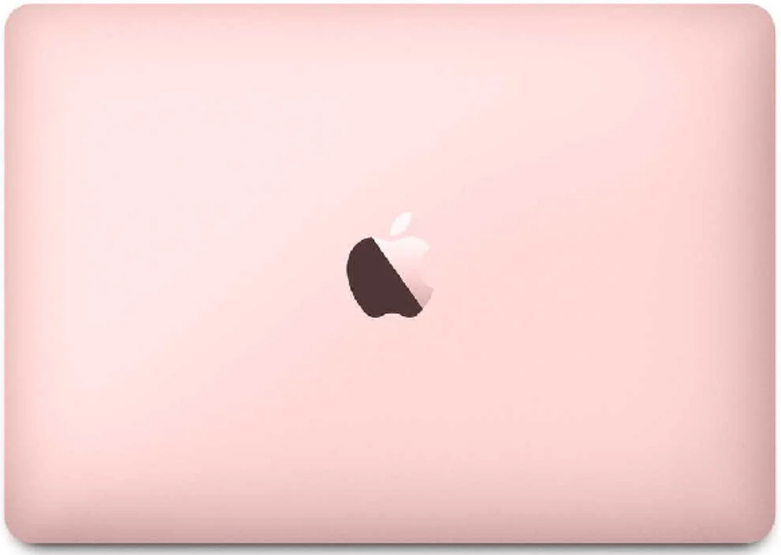
Analytical queries

Analytical queries

100 GB

Analytical queries

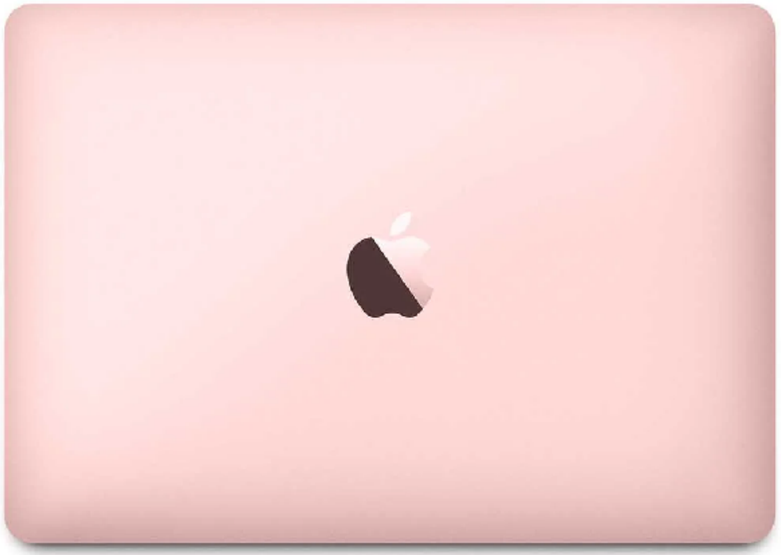
100 GB



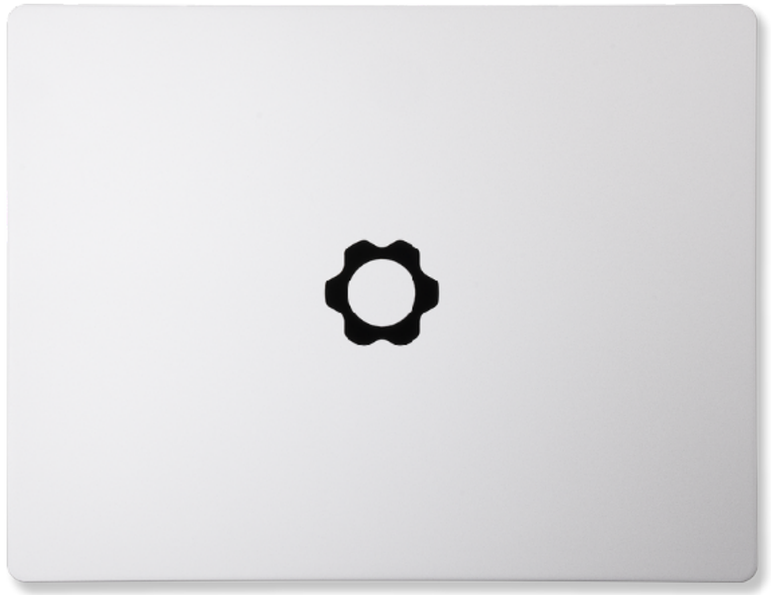
1000 GB

Analytical queries

100 GB

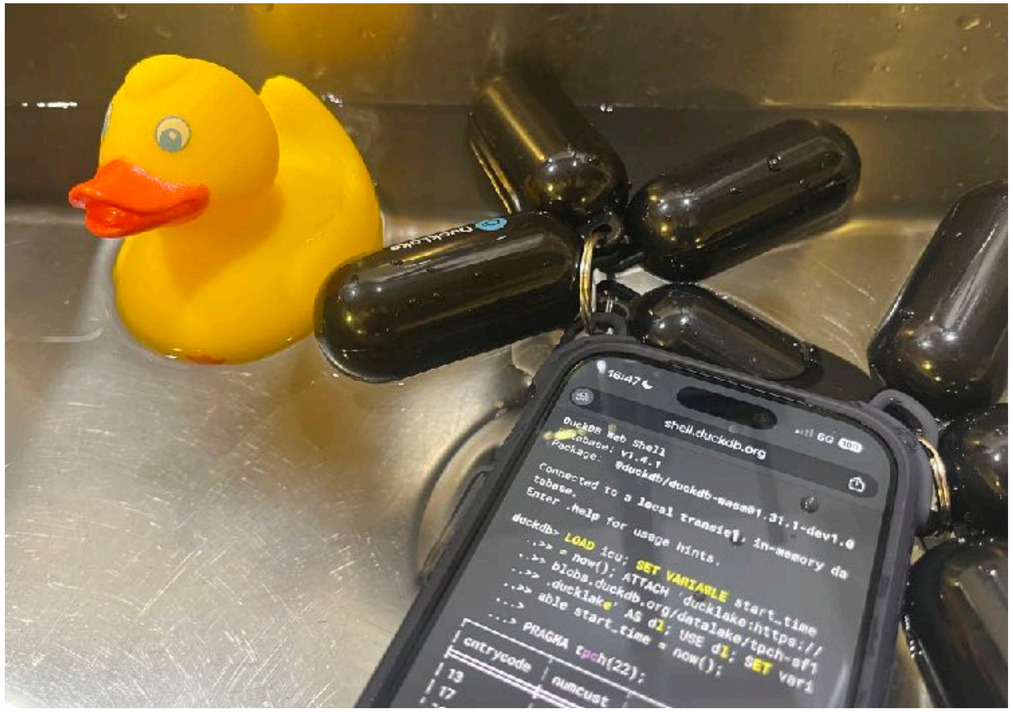


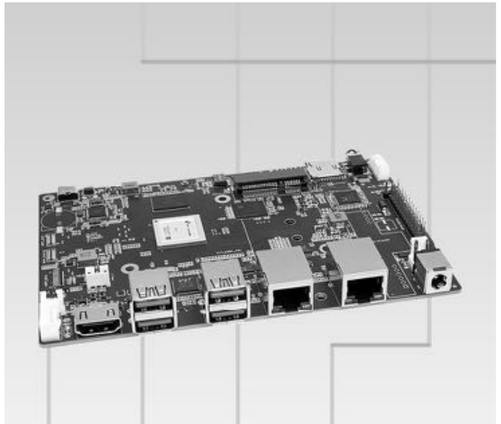
1 000 GB



10 000 GB


DuckDB Shell
shell.duckdb.org
150%
DuckDB Web Shell
Database: v1.4.3
Package: @duckdb/duckdb-wasm@1.33.1-dev16.0
Connected to a local transient in-memory database.
Enter .help for usage hints.
duckdb> LOAD icu; SET VARIABLE start_time = now(); ATTACH 'ducklake:https://blobs.duckdb.org/datalake/tpch-sf1.ducklake' AS dl; USE dl; SET variabl
..>> e start_time = now();
...>
...> PRAGMA tpch(22);
cntrycode | numcust | totacctbal
13 | 888 | 6737713.99
17 | 861 | 6460573.72
18 | 964 | 7236687.40
23 | 892 | 6701457.95
29 | 948 | 7158866.63
30 | 909 | 6808436.13
31 | 922 | 6886670.18
duckdb> SELECT epoch(now () - getvariable('start_time'))::DECIMAL(6, 2) AS runtime;
...>
runtime
12.24



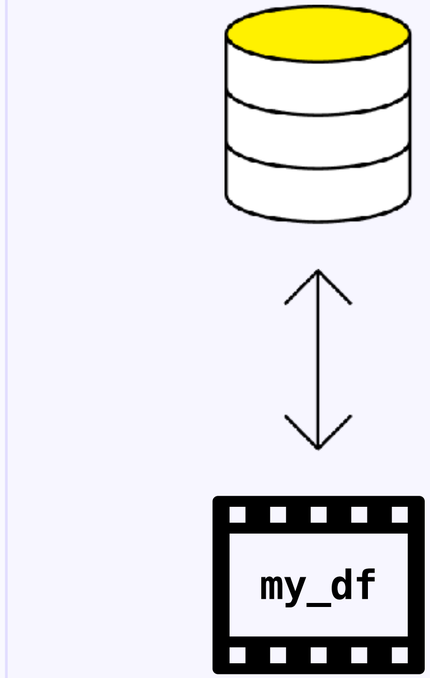



cocoon

Python integrations

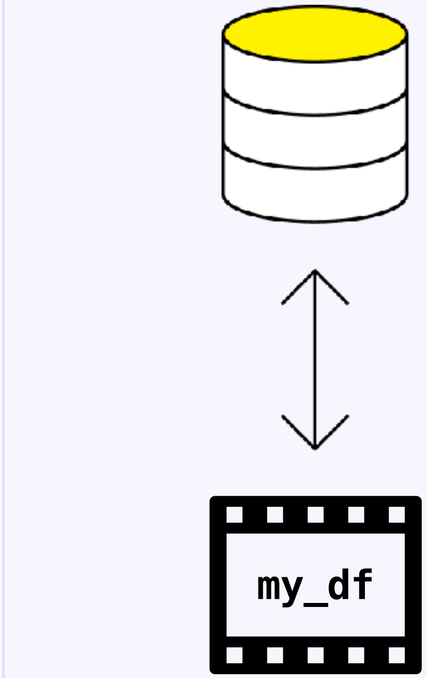
Pandas integration

```python
import duckdb
import pandas as pd

my_df = pd.DataFrame \
          .from_dict({'a': [42, 43]})

res = duckdb \
        .sql("SELECT avg(a) FROM my_df")

res.df()

#  avg(a)
# 0  42.5
```



Pandas integration

```
import duckdb
import pandas as pd

my_df = pd.DataFrame \
          .from_dict({'a': [42, 43]})

res = duckdb \
        .sql("SELECT avg(a) FROM my_df")

res.df()

#  avg(a)
# 0  42.5
```



pandas

Pandas integration

```python
import duckdb
import pandas as pd

my_df = pd.DataFrame \
          .from_dict({'a': [42, 43]})

res = duckdb \
        .sql("SELECT avg(a) FROM my_df")

res.df()

#   avg(a)
# 0   42.5
```

Replacement scan

pandas

Pandas integration

```python
import duckdb
import pandas as pd

my_df = pd.DataFrame \
          .from_dict({'a': [42, 43]})

res = duckdb \
        .sql("SELECT avg(a) FROM my_df")

res.df()

#  avg(a)
# 0  42.5
```

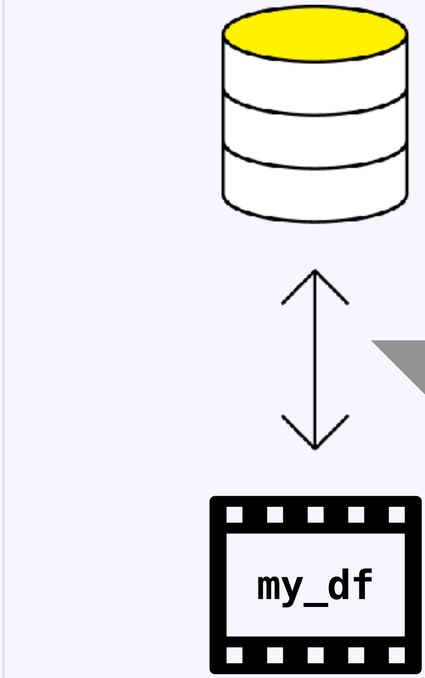
Replacement scan

pandas

Pandas integration

```python
import duckdb
import pandas as pd

my_df = pd.DataFrame \
          .from_dict({'a': [42, 43]})

res = duckdb \
        .sql("SELECT avg(a) FROM my_df")

res.df()

#  avg(a)
# 0  42.5
```

Replacement scan

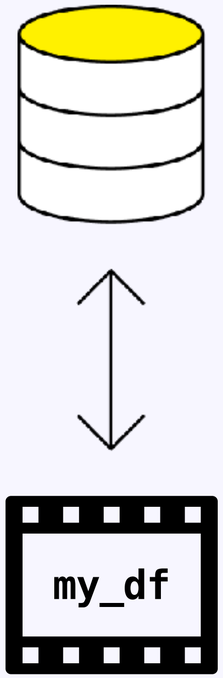


Zero-copy access

pandas

Polars integration

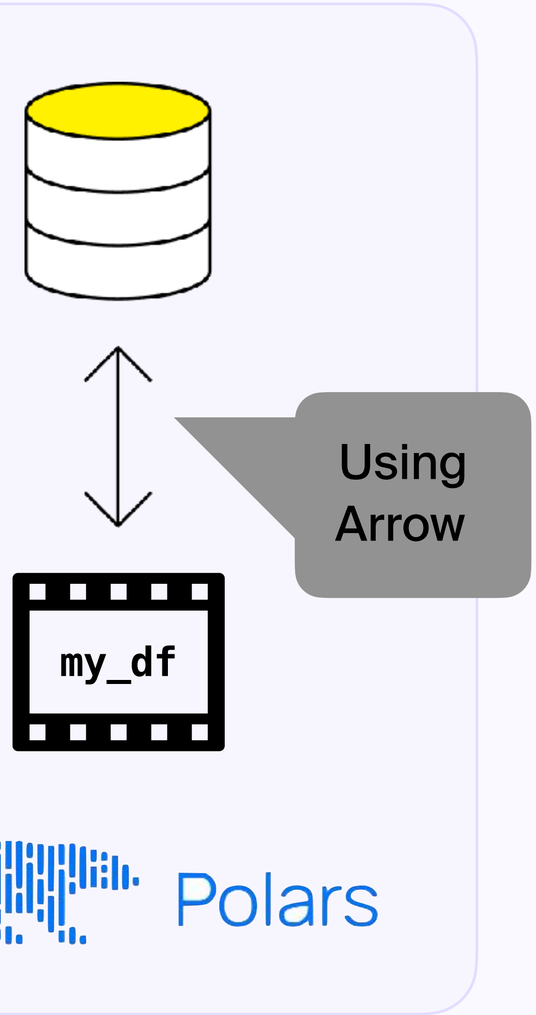
```python
import duckdb
import polars as pl

my_df = pl.DataFrame({'a': [42, 43]})
res = duckdb.sql("SELECT avg(a) FROM my_df")
res.pl()

# shape: (1, 1)
#
#  avg(a)
#  ---
#  f64
#
#  42.5
#
```



Polars

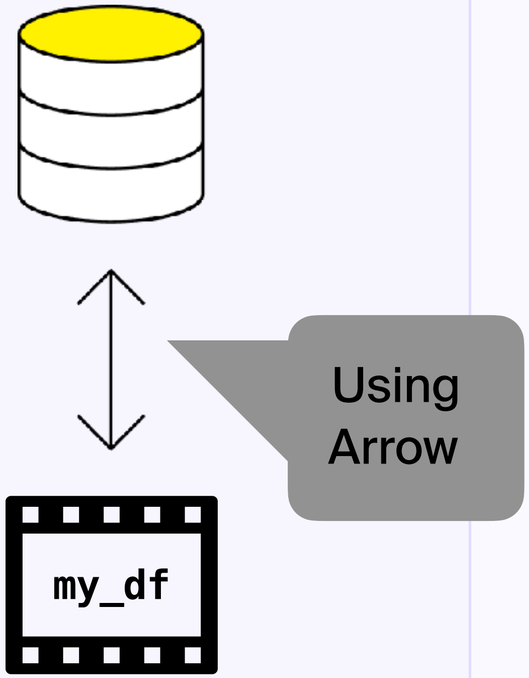
Polars integration

```
import duckdb
import polars as pl

my_df = pl.DataFrame({'a': [42, 43]})
res = duckdb.sql("SELECT avg(a) FROM my_df")
res.pl()

# shape: (1, 1)
#
#   avg(a)
#   ---
#   f64
#  ═══════
#   42.5
#
```



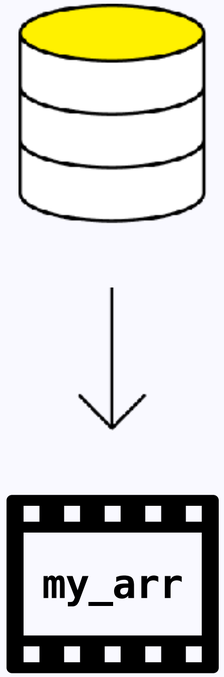
Polars integration

```
import duckdb
import polars as pl

my_df = pl.DataFrame({'a': [42, 43]})
res = duckdb.sql("SELECT avg(a) FROM my_df")
res.pl()

# shape: (1, 1)
#
#   avg(a)
#   ---
#   f64
#
#   42.5
#
```



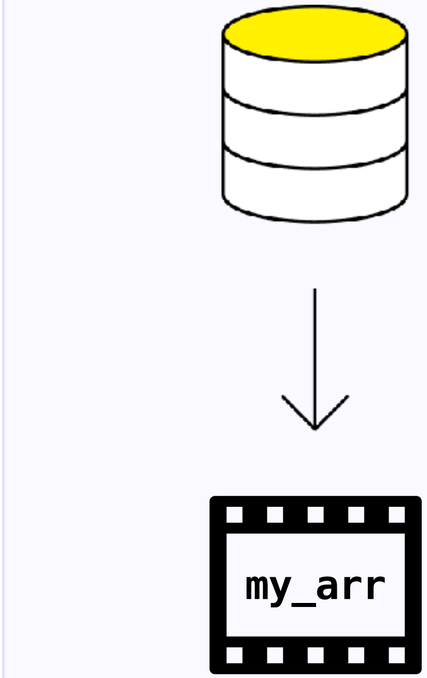
Numpy integration

```
import duckdb
import numpy as np

my_arr = duckdb \
    .sql("FROM (VALUES (23), (24)) t(a)") \
    .fetchnumpy()

np.sum(my_arr['a'])

# np.int64(47)
```



Numpy integration

```
import duckdb
import numpy as np

my_arr = duckdb \
    .sql("FROM (VALUES (23), (24)) t(a)") \
    .fetchnumpy()

np.sum(my_arr['a'])

# np.int64(47)
```



Support user-defined functions (UDFs)

```python
import duckdb
from faker import Faker

def get_random_name():
    return Faker().name()

duckdb.create_function(
    "random_name", get_random_name, [],
    duckdb.typing.VARCHAR
)

duckdb.sql("SELECT random_name()") \
    .fetchall()
```

```
SELECT random_name()
```

```
get_random_name()
```

Support user-defined functions (UDFs)

```python
import duckdb
from faker import Faker

def get_random_name():
    return Faker().name()

duckdb.create_function(
    "random_name", get_random_name, [],
    duckdb.typing.VARCHAR
)

duckdb.sql("SELECT random_name()") \
    .fetchall()
```

SELECT random_name()

get_random_name()

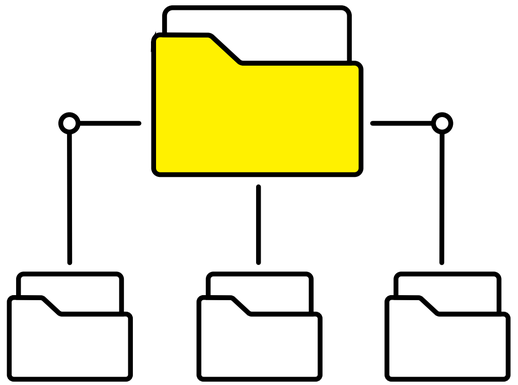
Support user-defined functions (UDFs)

```python
import duckdb
from faker import Faker

def get_random_name():
    return Faker().name()

duckdb.create_function(
    "random_name", get_random_name, [],
    duckdb.typing.VARCHAR
)

duckdb.sql("SELECT random_name()") \
    .fetchall()
```

Roadmap: harness no-GIL Python to parallelize

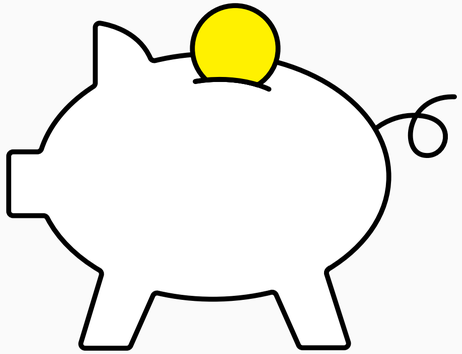
```
SELECT random_name()
```

```
get_random_name()
```



Use cases

Cost saving



Fast local development:
time & egress fees

Replace proprietary systems:
license cost

Replace distributed systems:
hardware cost

Last mile analytics

1 PB log →
pre-process with Spark

Process the 200 GB
aggregate DuckDB

Build fast dashboards

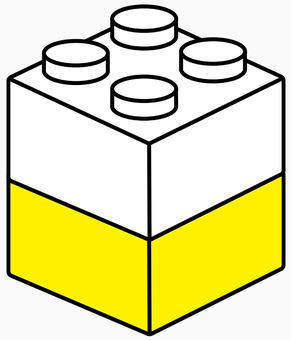
Education

Easy to install

Open-source

No DBA needed

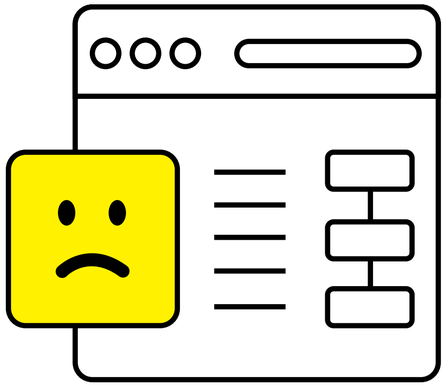
Modernizing codebases



Postgres-compatible SQL dialect

Builds on open formats (CSV, Parquet)

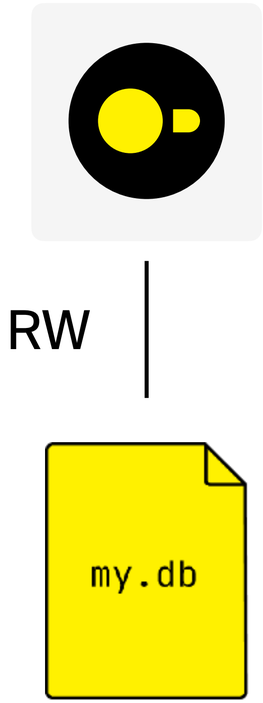
LLMs can refactor code to use DuckDB



Limitations

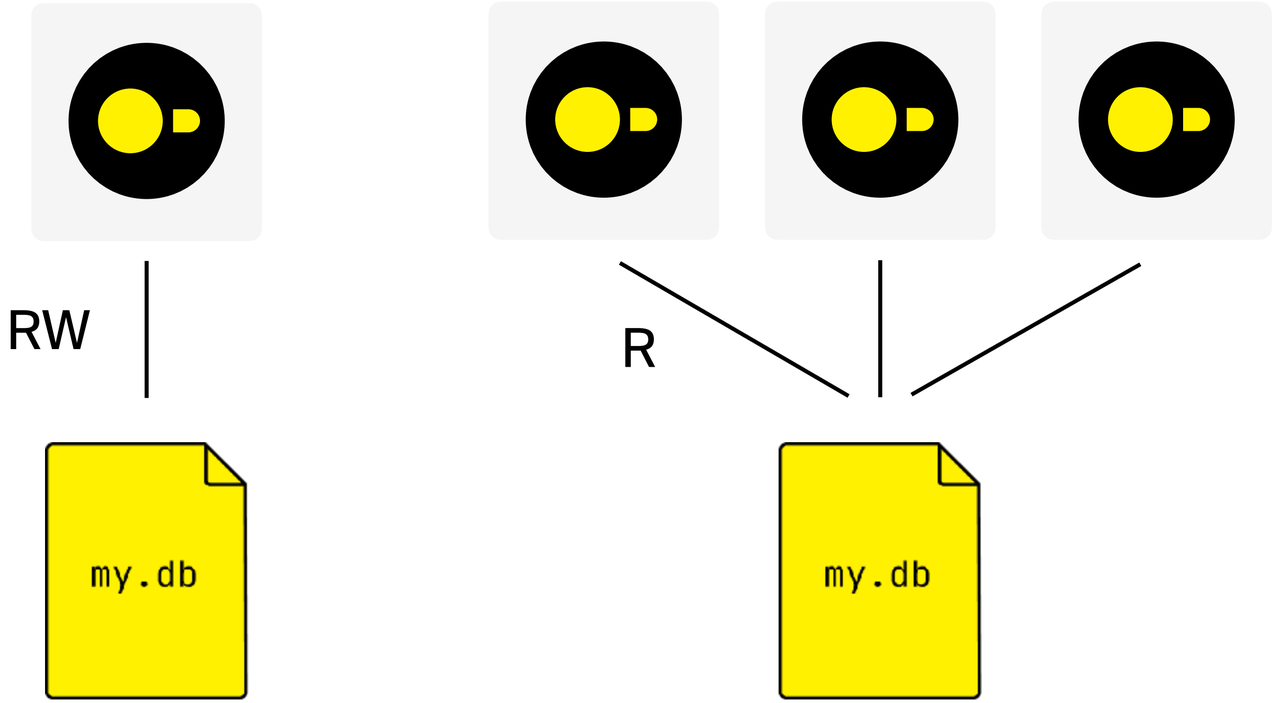
What are DuckDB's limitations?

Single-player mode: only a single process can attach in RW mode



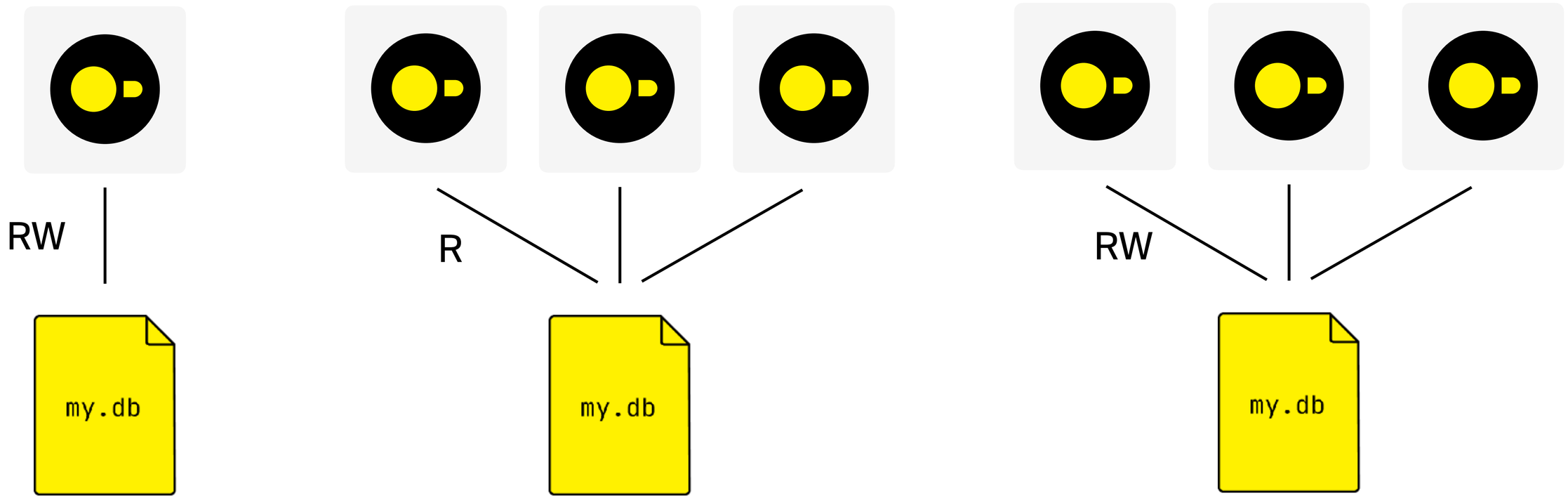
What are DuckDB's limitations?

Single-player mode: only a single process can attach in RW mode



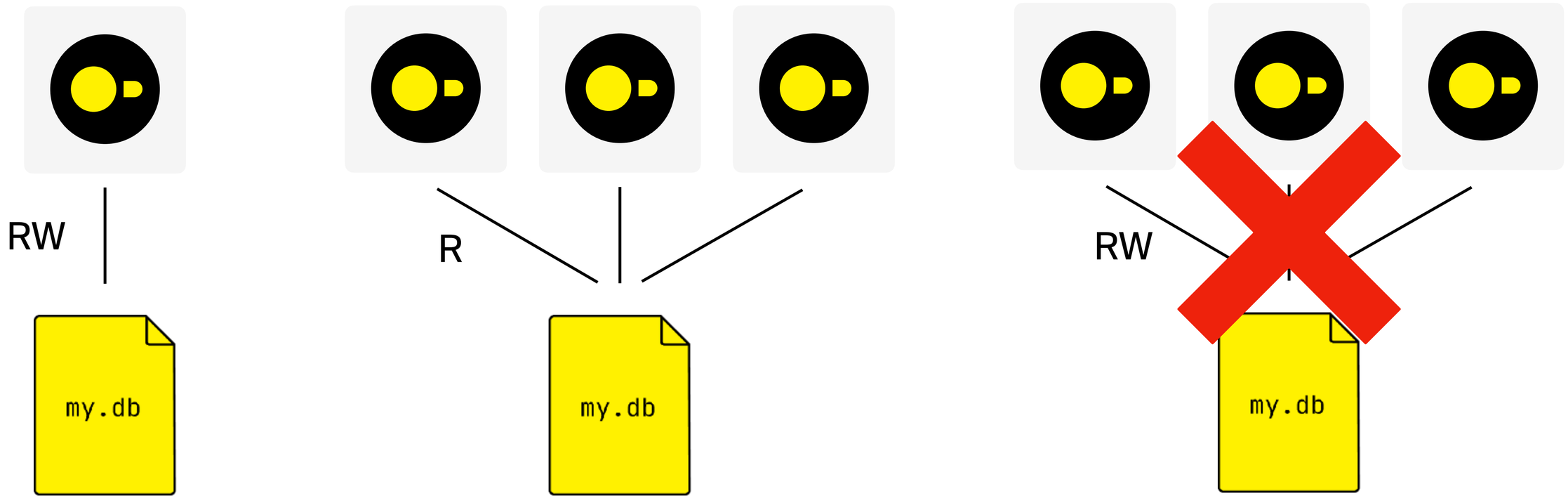
What are DuckDB's limitations?

Single-player mode: only a single process can attach in RW mode



What are DuckDB's limitations?

Single-player mode: only a single process can attach in RW mode



What are DuckDB's limitation?

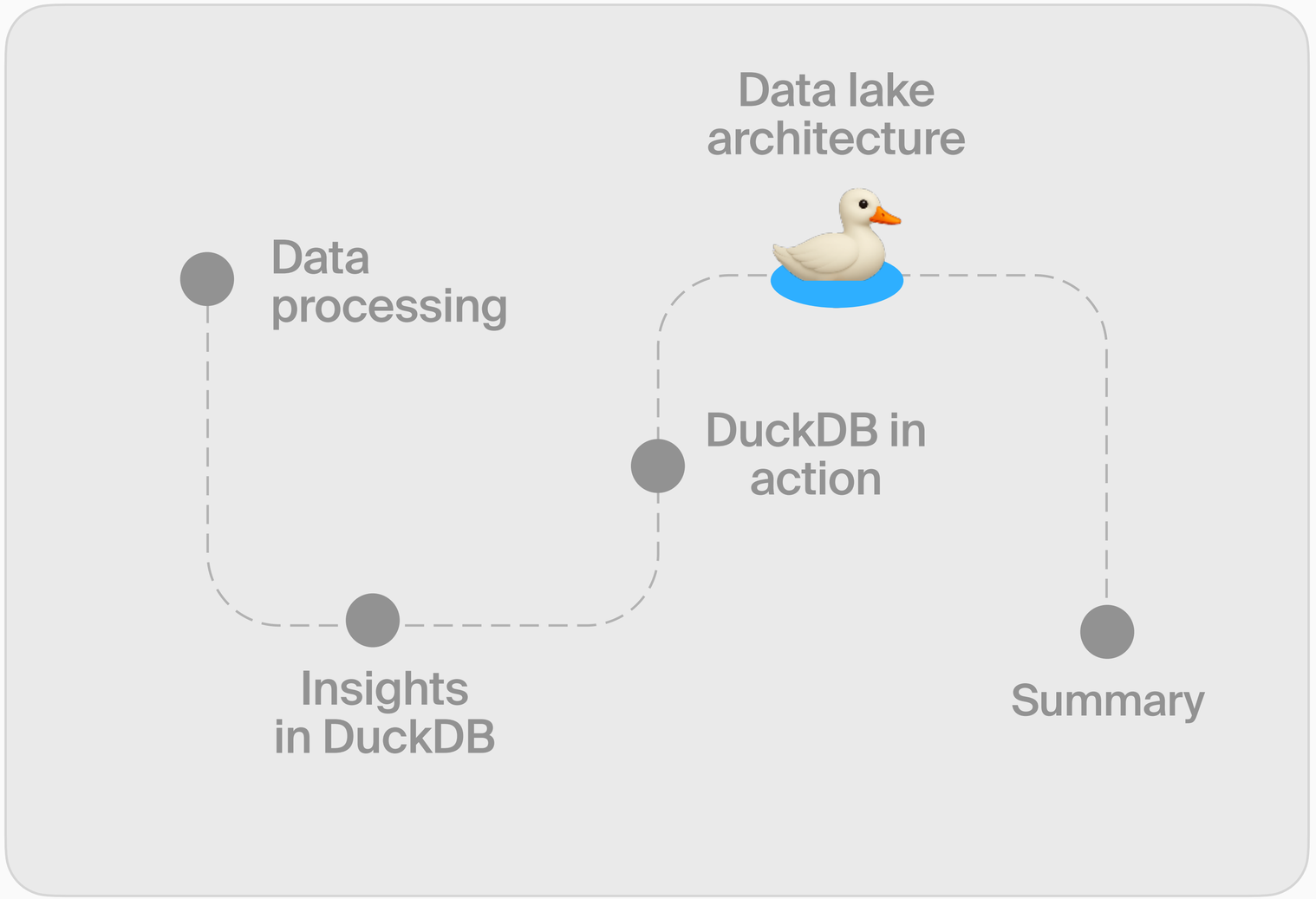
10 TB CSV = 2.7 TB database file

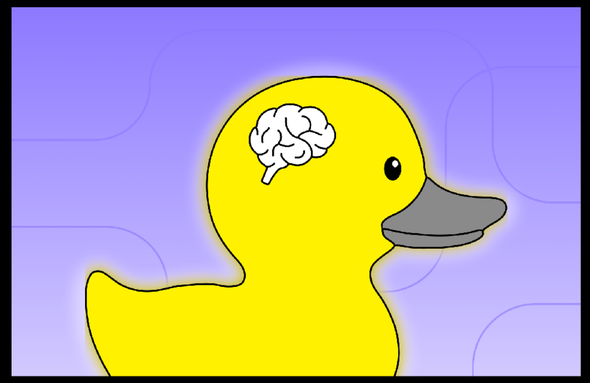
Optimized for batch loading

small writes are slow

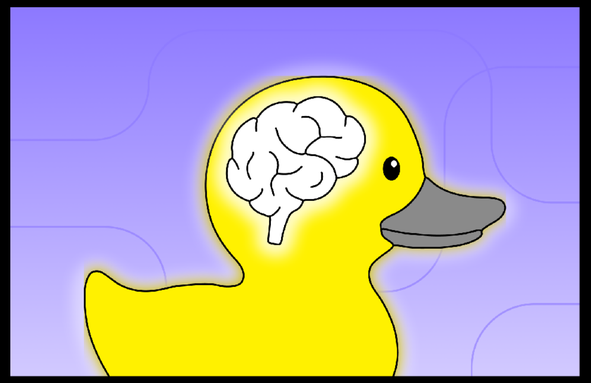
Huge database file

difficult to handle

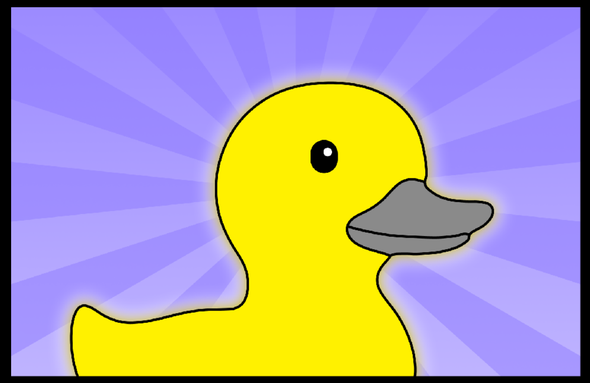

Data lake architecture
Data processing
DuckDB in action
Insights in DuckDB
Summary



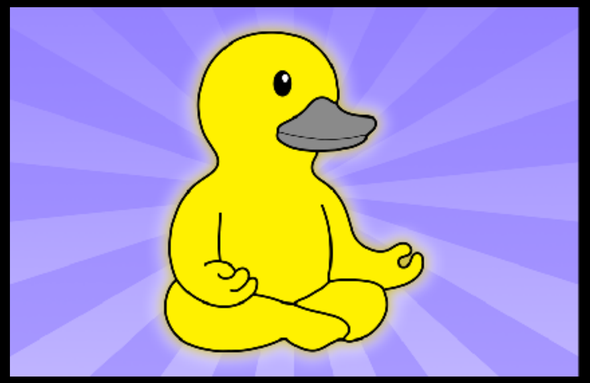
Text files



Binary files



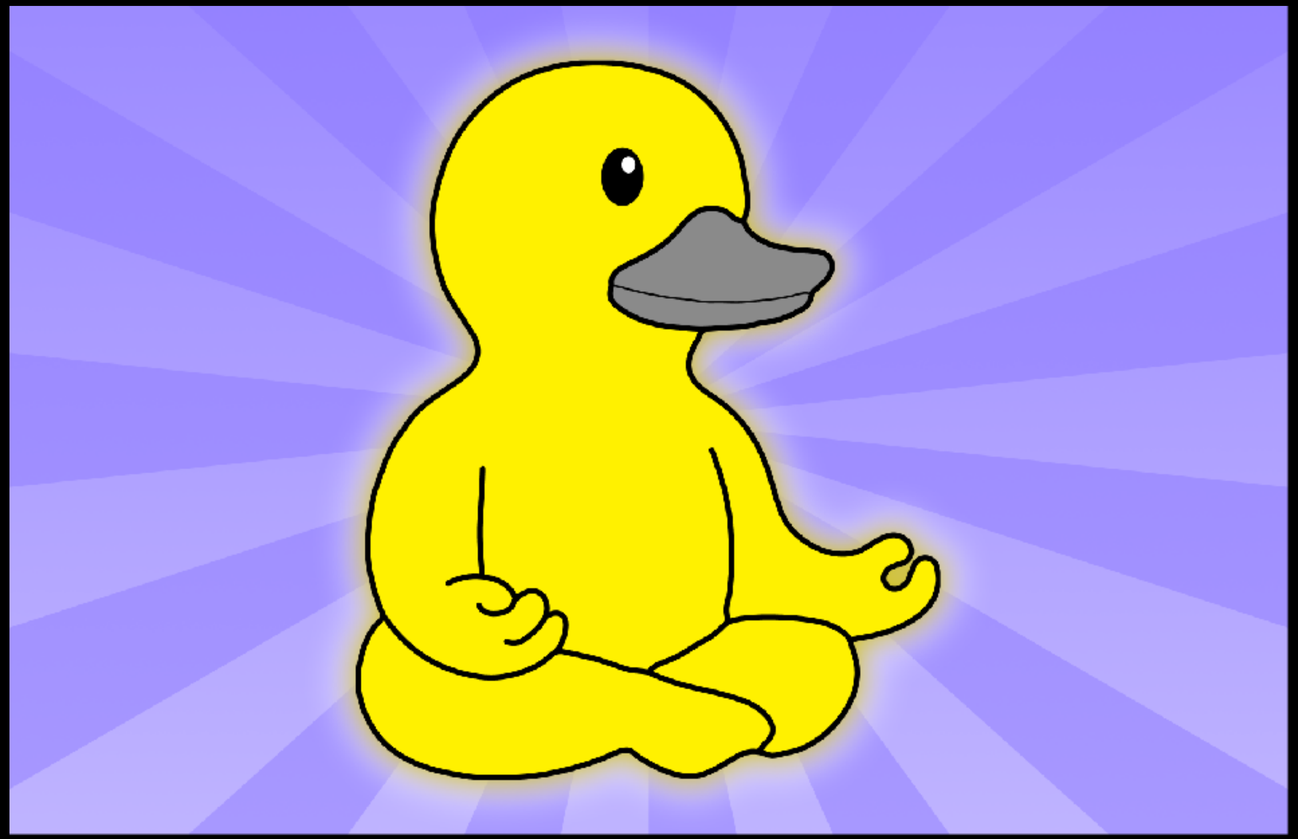
Database



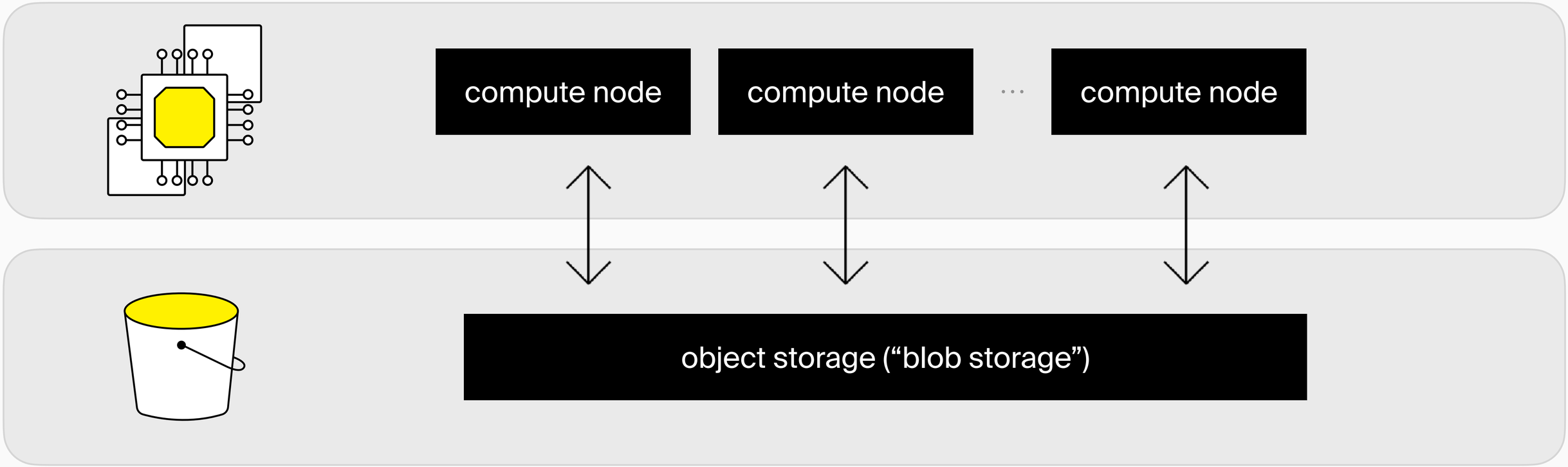
Data lake

Data lake

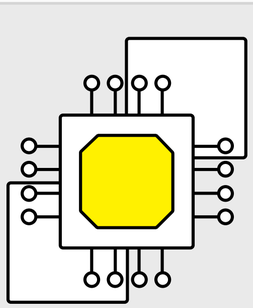
But first, some history on cloud architecture!



2005 Separation of compute and storage



2005 Separation of compute and storage

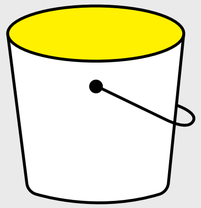


compute node

compute node

...

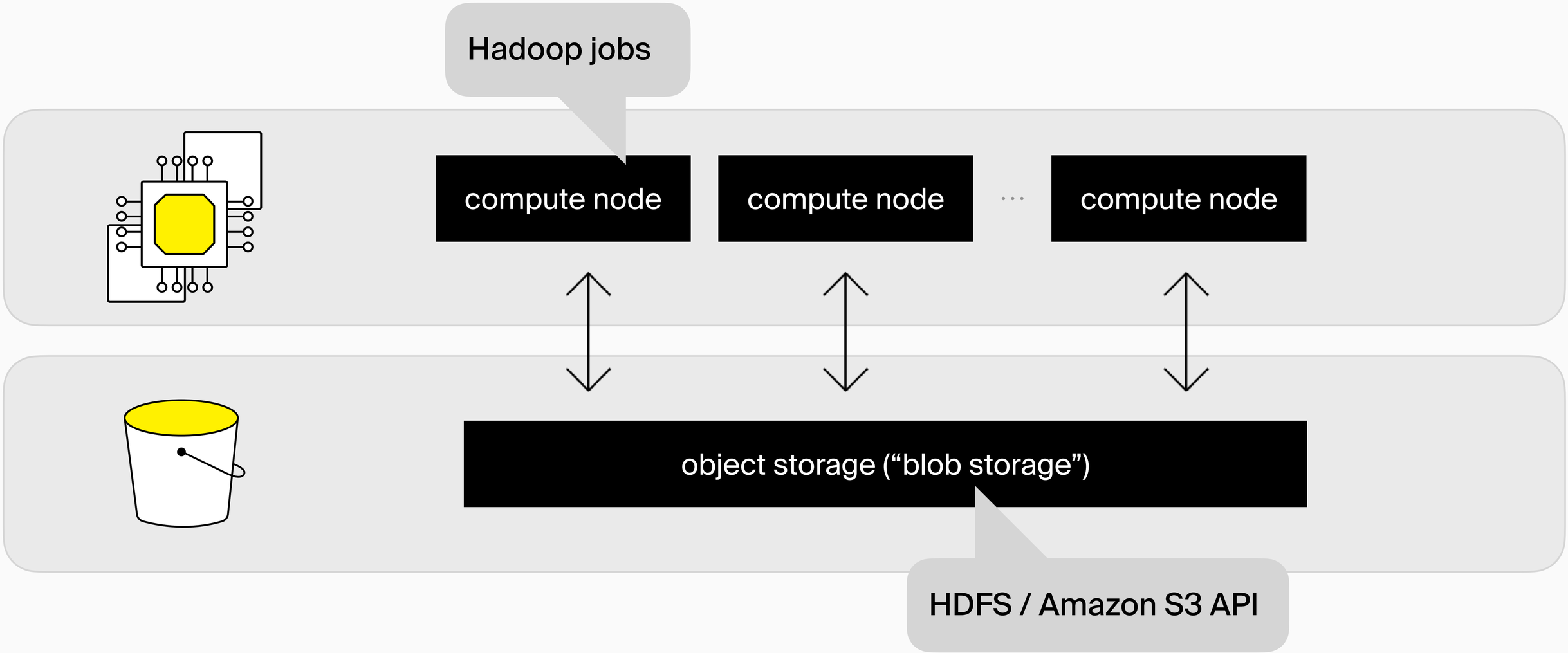
compute node



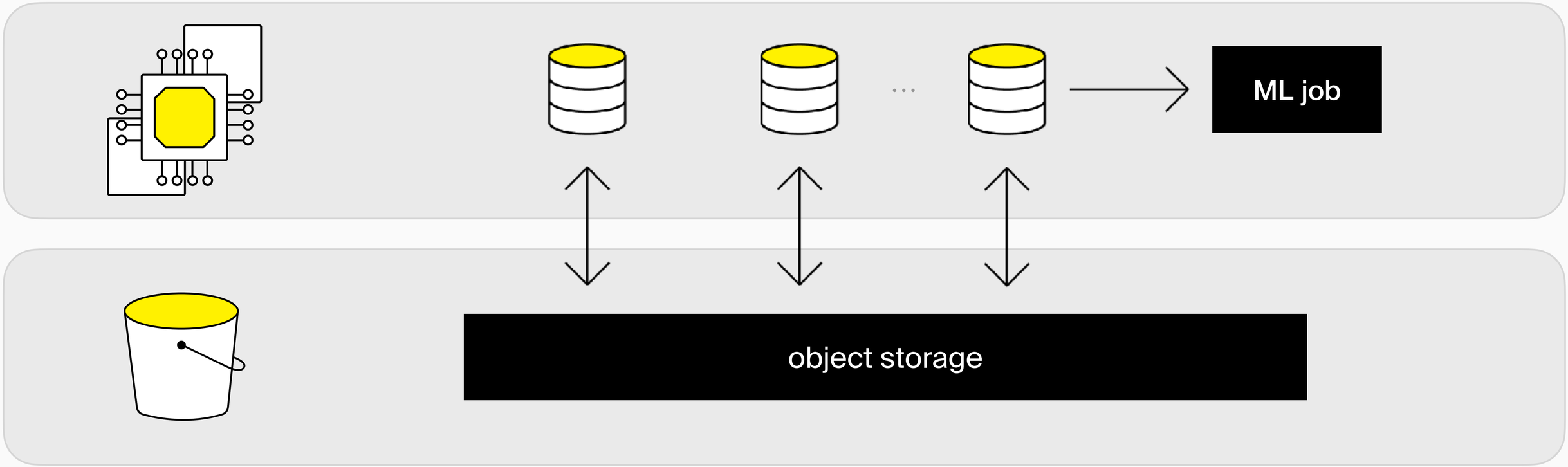
object storage (“blob storage”)

HDFS / Amazon S3 API

2005 Separation of compute and storage



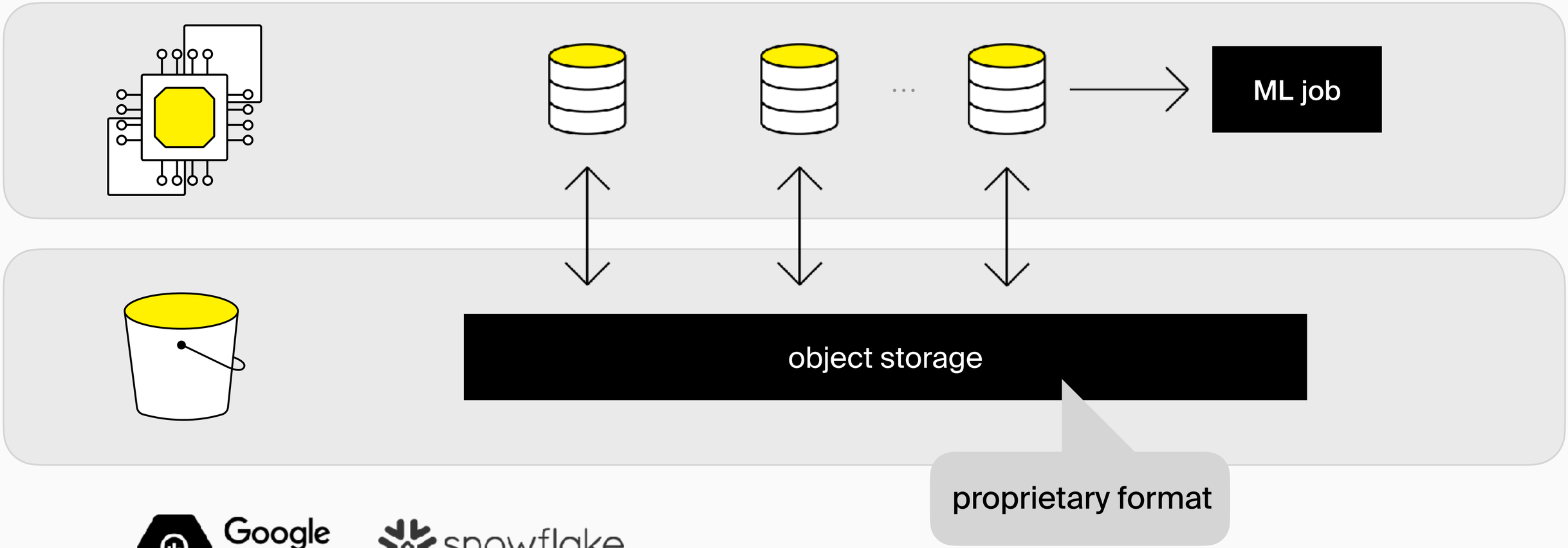
2010 Cloud data warehouses



Google Big Query

snowflake

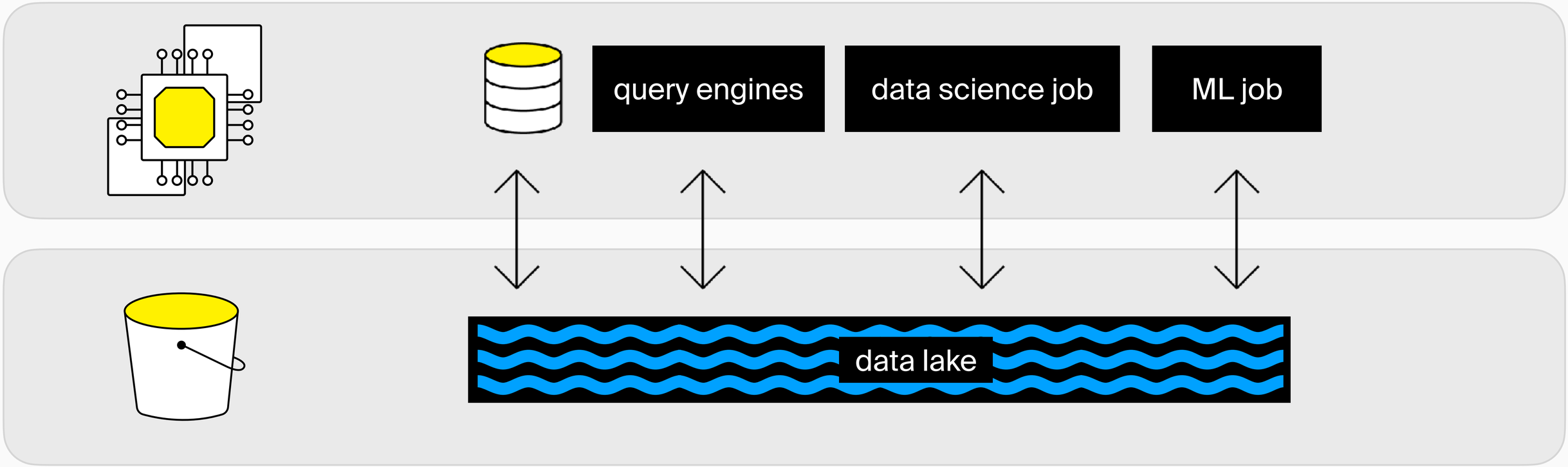
2010 Cloud data warehouses



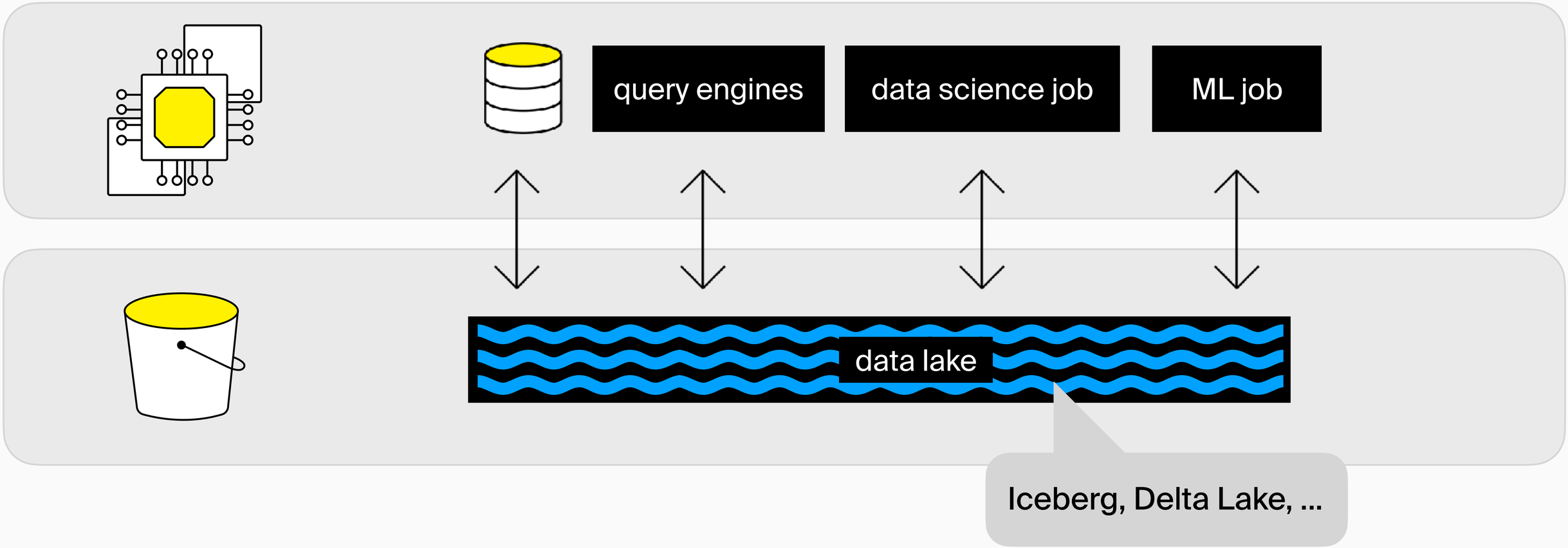
Google Big Query

snowflake

2015 Data lake architecture



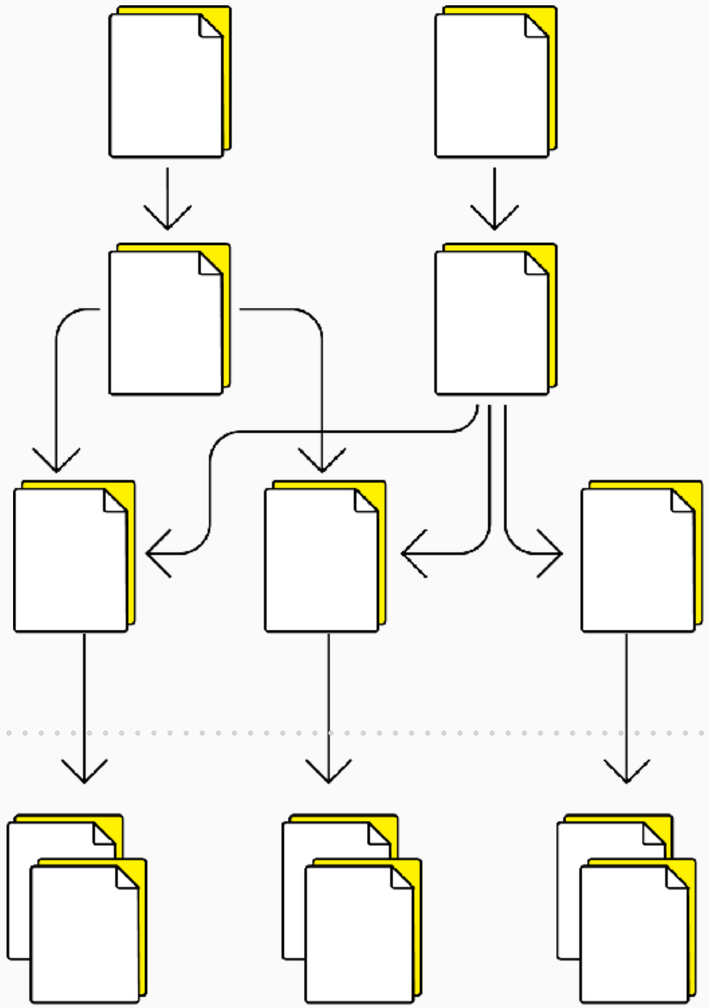
2015 Data lake architecture



ICEBERG

metadata
(json / avro)

statistics

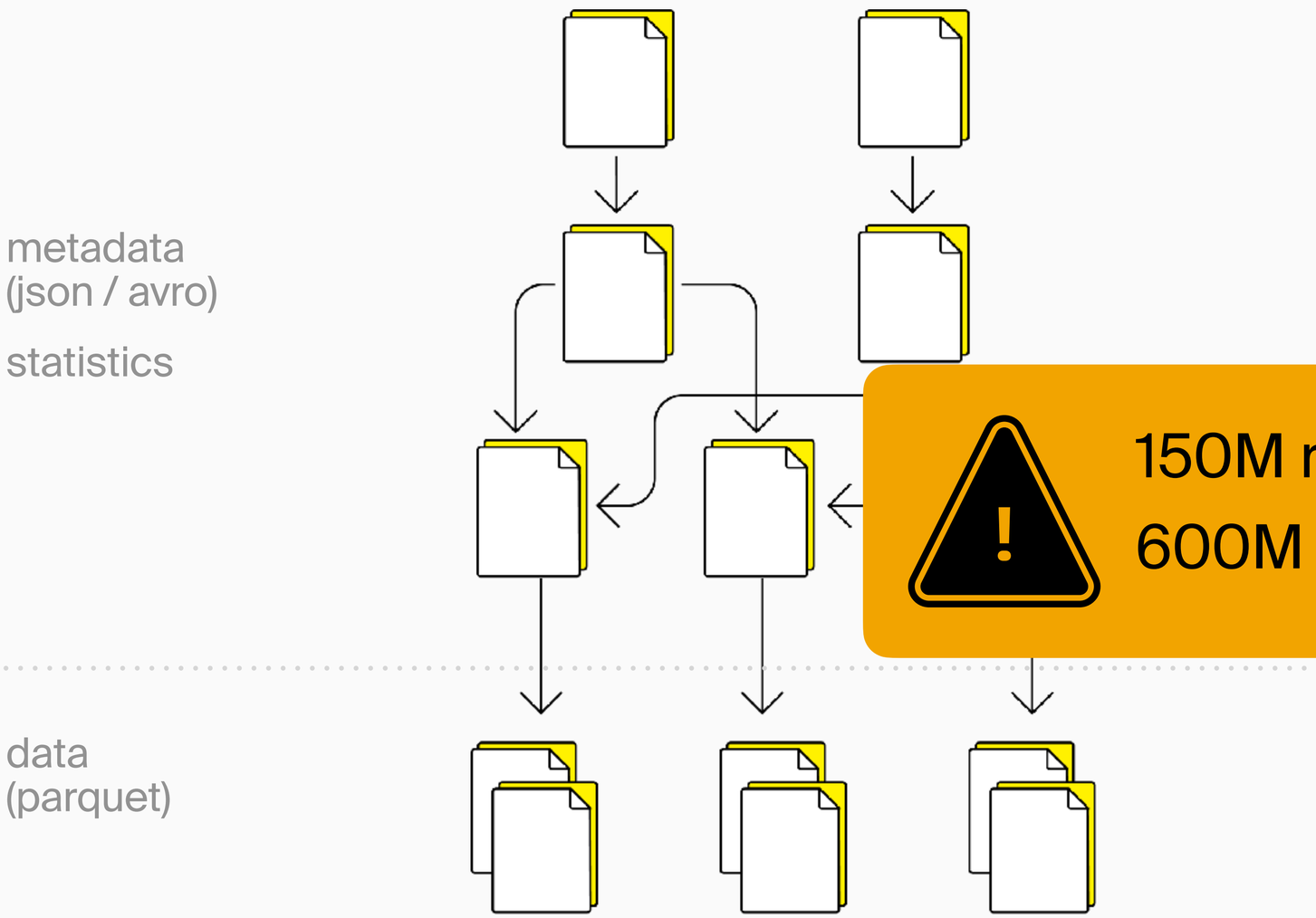


data
(parquet)

- File-only architecture
- Updatable table
- Schema evolution

- Performance problems
- “Small files” problem
- Only a single table

ICEBERG



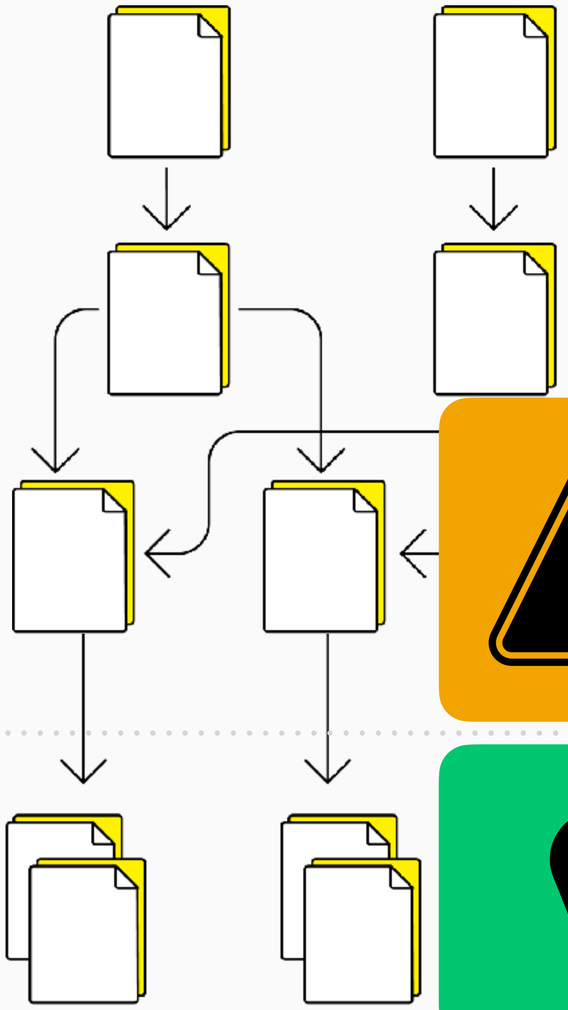
- File-only architecture
- Updatable table

150M record inserted one-by-one:

600M files (metadata + data)

- Performance problems
- “Small files” problem
- Only a single table

ICEBERG



- File-only architecture
- Updatable table

150M record inserted one-by-one:

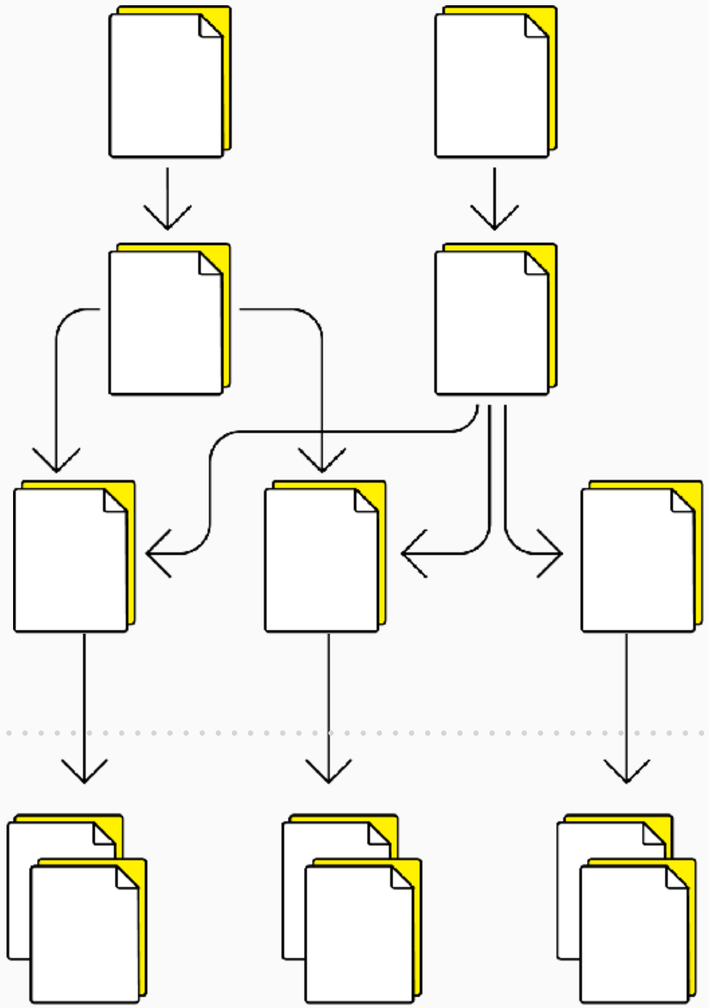
600M files (metadata + data)

Compaction algorithms

ICEBERG

metadata
(json / avro)

statistics



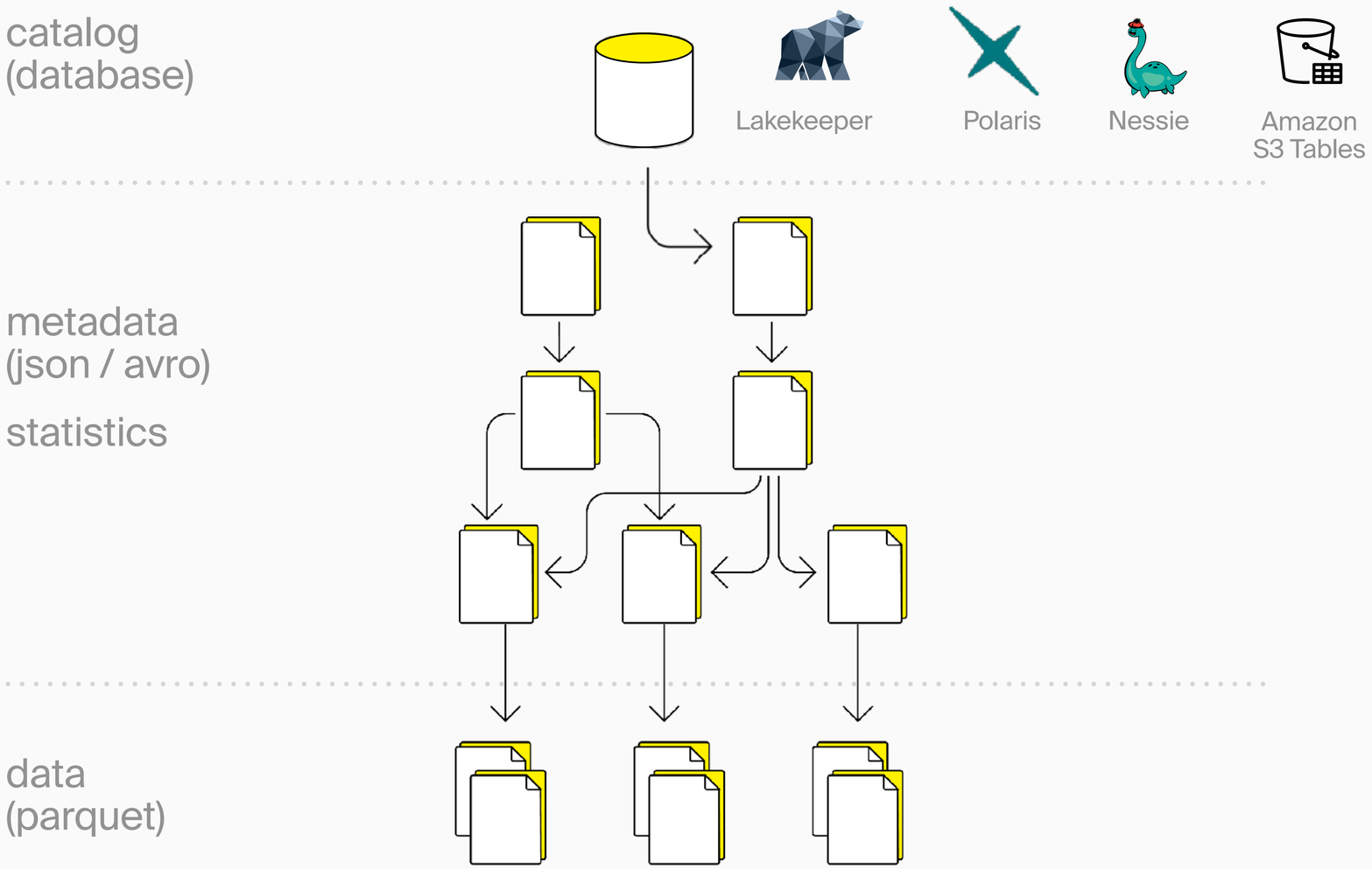
data
(parquet)

DuckDB has full Iceberg support!

```
INSTALL iceberg;
```

It's an alternative to the pyiceberg package, integrated into DuckDB.

2020 Lakehouse architecture

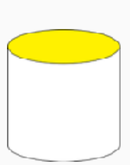


- Cost-effective
- Small catalog database
- Full database features

- Performance problems
- Complex operations

2020 Lakehouse architecture

catalog (database)



Lakekeeper

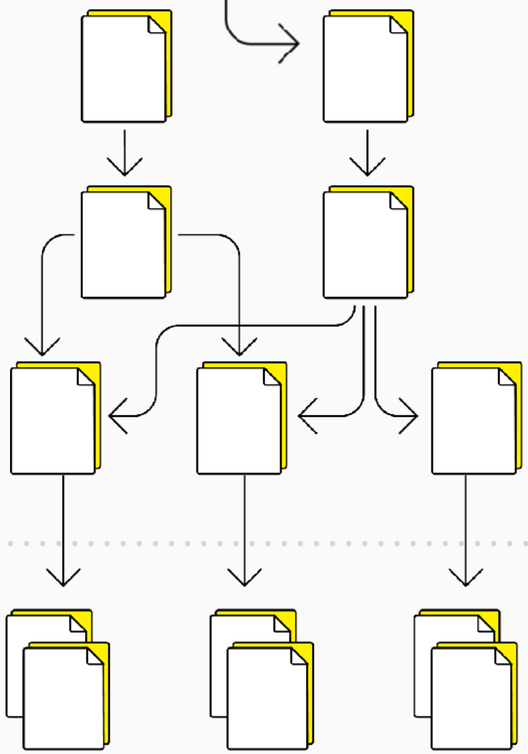
Polaris

Nessie

Amazon S3 Tables

metadata (json / avro)

statistics

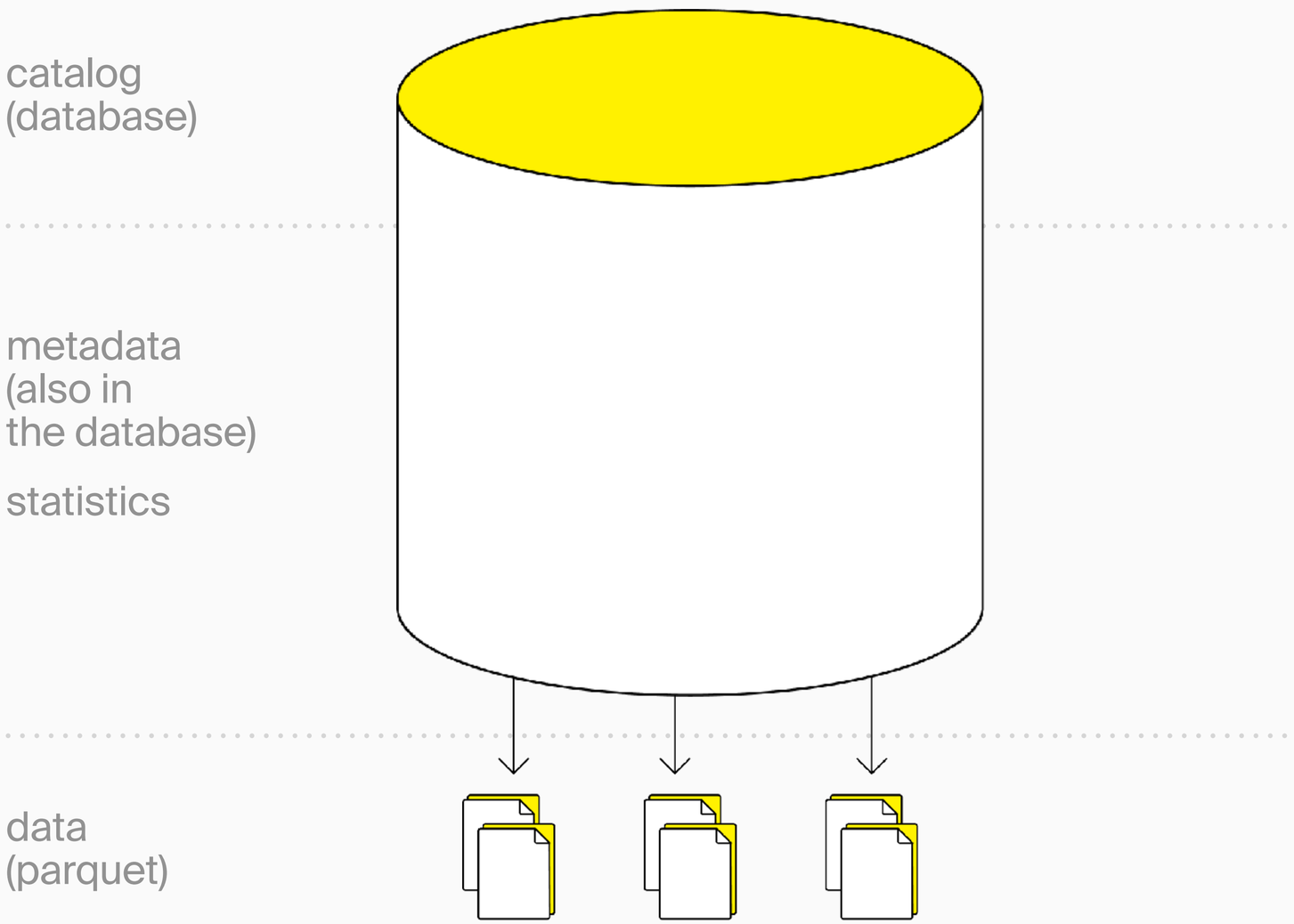


data (parquet)

- Cost-effective
- Small catalog database
- Full database features

- Performance problems
- Complex operations

2025 DuckLake



- Cost-effective
- Full database features
- Simple operations
- Scalable

- Snapshot mechanism
- Multi-table transactions
- High-performance

DuckLake

Quick setup

```
ATTACH 'ducklake:trains.ducklake' AS trains;
USE trains;

CREATE TABLE services AS
FROM 'services.csv';
```

DuckLake

Let's renumber the trains

```sql
UPDATE services
SET train_number =
    train_number + (random() * 100 + 1)::INT
WHERE delay > 150;
```

DuckLake

Let's renumber the trains

```sql
UPDATE services
SET train_number =
    train_number + (random() * 100 + 1)::INT
WHERE delay > 150;

SELECT snapshot_id, changes
FROM ducklake_snapshots('trains');
```

DuckLake

Let's renumber the trains

```
UPDATE services
SET train_number =
    train_number + (random() * 100 + 1)::INT
WHERE delay > 150;

SELECT snapshot_id, changes
FROM ducklake_snapshots('trains');
```

snapshot_id int64	changes map(varchar, varchar[])
0	{schemas_created=[main]}
1	{tables_created=[main.services], tables_inserted_into=[1]}
2	{tables_inserted_into=[1], tables_deleted_from=[1]}

DuckLake

Travel back in time

```sql
SELECT date, train_number, delay
FROM services AT (VERSION => 1)
WHERE date = '2025-02-27'
  AND station = 'Amsterdam Centraal'
  AND train_number = 420;
```

DuckLake

Travel back in time

```sql
SELECT date, train_number, delay
FROM services AT (VERSION => 1)
WHERE date = '2025-02-27'
  AND station = 'Amsterdam Centraal'
  AND train_number = 420;
```

DuckLake

Travel back in time

```
SELECT date, train_number, delay
FROM services AT (VERSION => 1)
WHERE date = '2025-02-27'
  AND station = 'Amsterdam Centraal'
  AND train_number = 420;
```

date	train_number	delay
2025-02-27	420	174

DuckLake

Travel back in time

```
SELECT date, train_number, delay
FROM services AT (VERSION => 1)
WHERE date = '2025-02-27'
  AND station = 'Amsterdam Centraal'
  AND train_number = 420;
```

date	train_number	delay
2025-02-27	420	174

DuckLake

Travel back in time

```
SELECT date, train_number, delay
FROM services AT (VERSION => 1)
WHERE date = '2025-02-27'
  AND station = 'Amsterdam Centraal'
  AND train_number = 420;
```

date	train_number	delay
2025-02-27	420	174

Passengers – Railway co.
1 – 0

DuckLake

Travel back in time

```
SELECT date, train_number, delay
FROM services AT (VERSION => 1)
WHERE date = '2025-02-27'
  AND station = 'Amsterdam Centraal'
  AND train_number = 420;
```

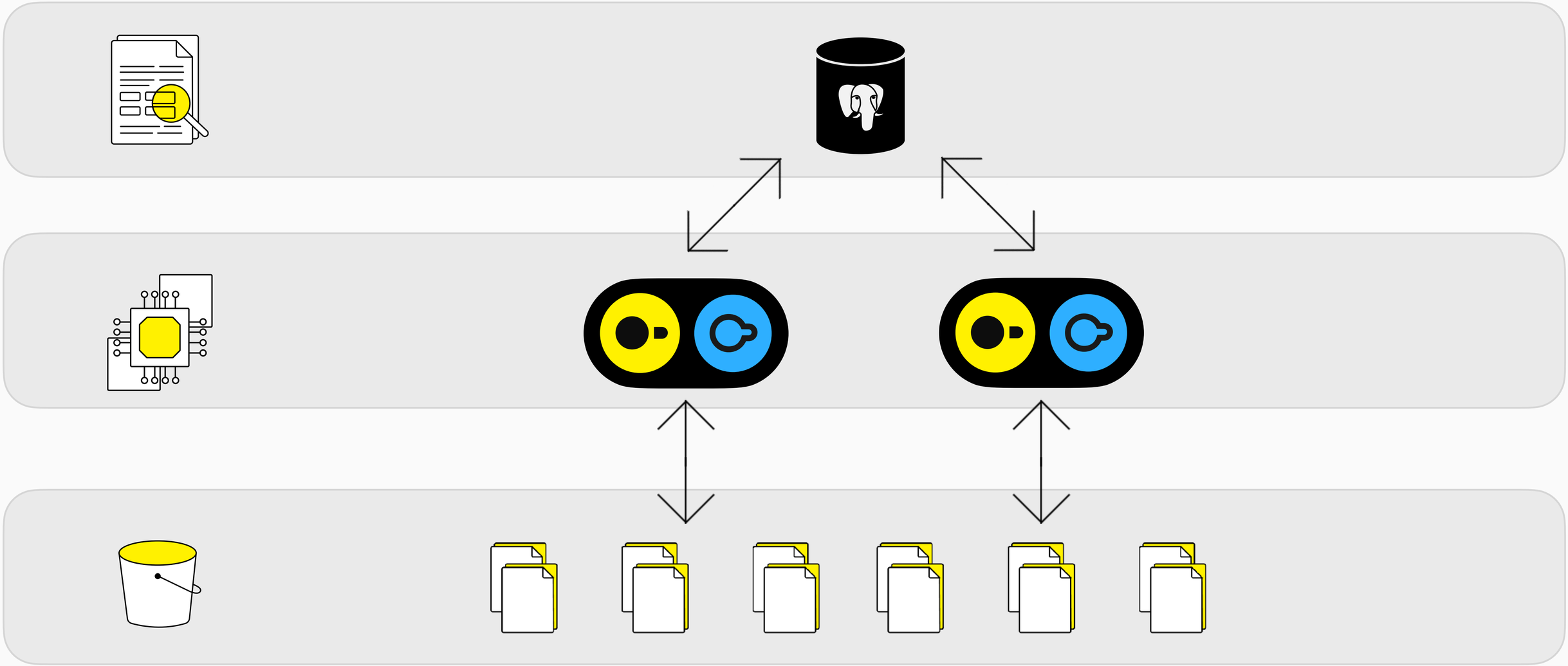
date	train_number	delay
2025-02-27	420	174

Passengers – Railway co.
1 – 0

Re-run improved scripts
on old datasets

DuckLake

Multiplayer DuckDB architecture

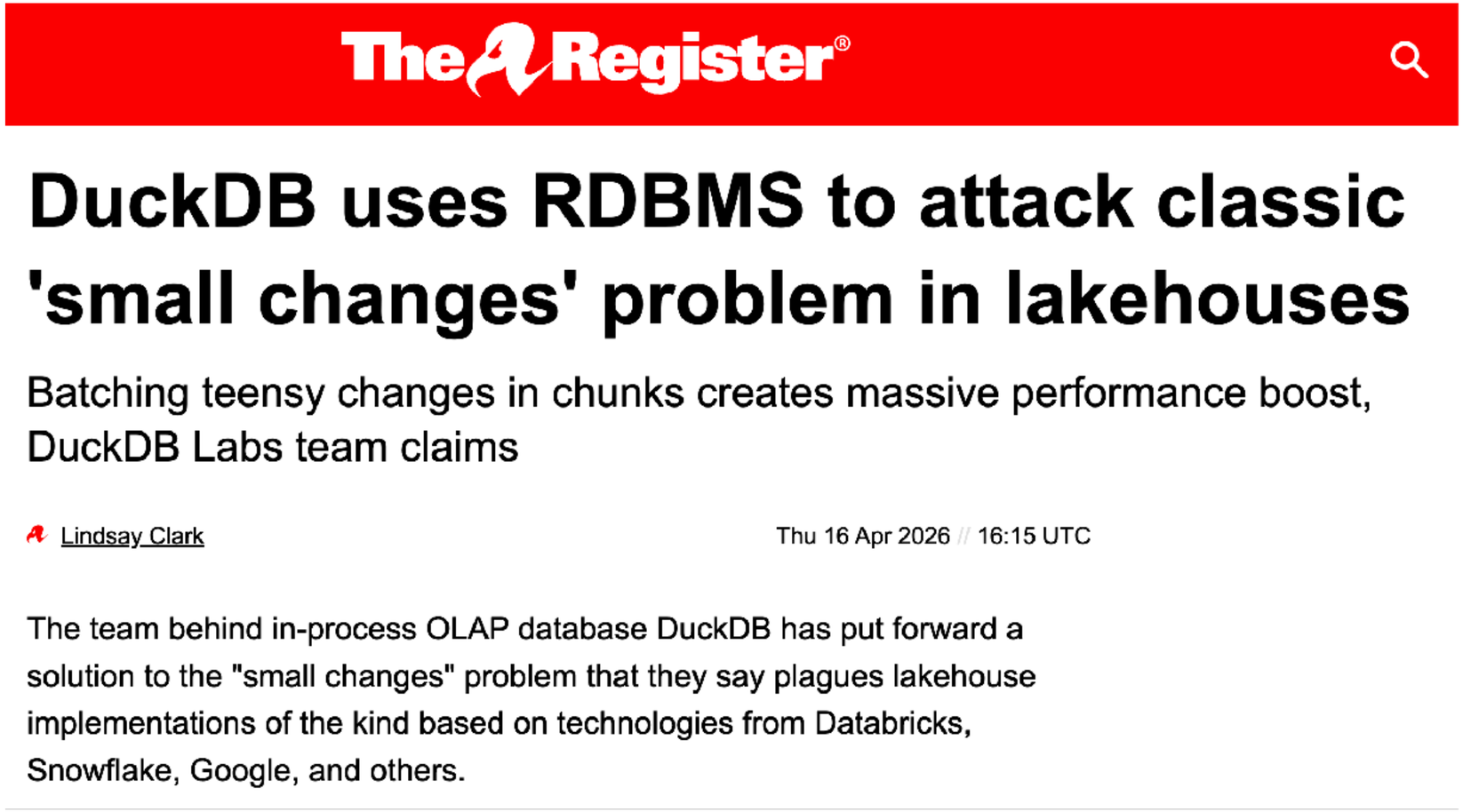


DuckLake

We release DuckLake v1.0
last Monday

Focus #1:
Stable specification

Focus #2:
Production-readiness



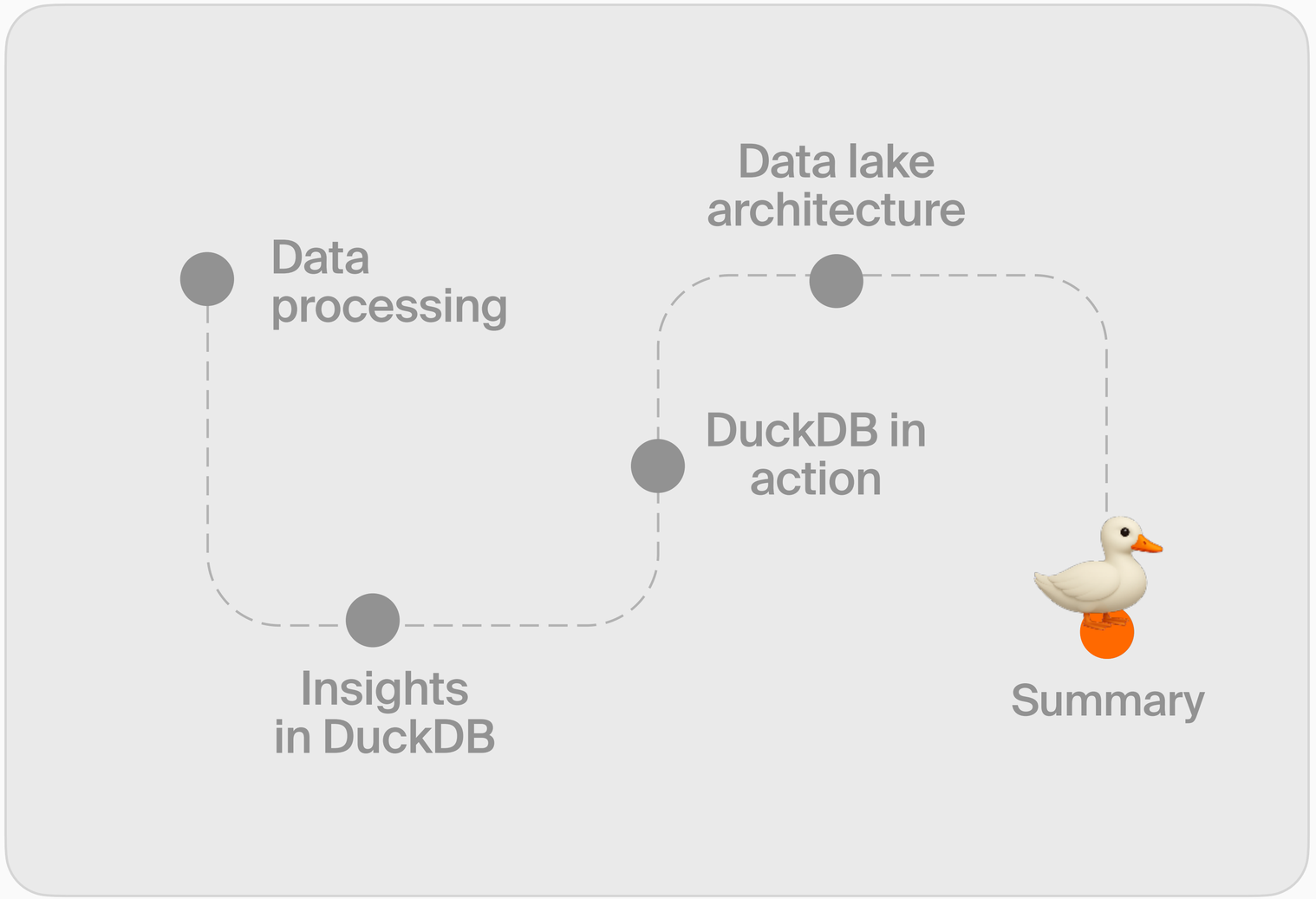
The Register®

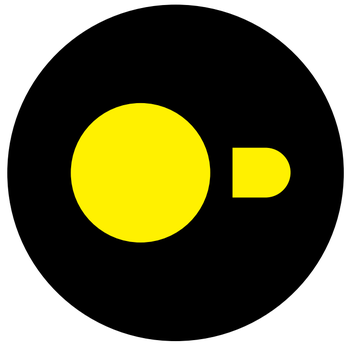
DuckDB uses RDBMS to attack classic 'small changes' problem in lakehouses

Batching teensy changes in chunks creates massive performance boost, DuckDB Labs team claims

Lindsay Clark — Thu 16 Apr 2026 // 16:15 UTC

The team behind in-process OLAP database DuckDB has put forward a solution to the "small changes" problem that they say plagues lakehouse implementations of the kind based on technologies from Databricks, Snowflake, Google, and others.


Data lake architecture
Data processing
DuckDB in action
Insights in DuckDB
Summary



DuckDB

An open-source in-process analytical database management system that you can import as a library

SQL and DuckDB internals from Universität Tübingen

Tabular Database Systems

Inner Join Follows Foreign Keys

vehicles

vid	vehicle	kind	seats	wheels?	pid
v_1	🚗	car	5	true	p_4
v_2	🚙	SUV	3	true	p_4
v_3	🚌	bus	42	true	▢
v_4	🚌	bus	7	true	▢
v_5	🚲	bike	1	true	p_2
v_6	🚜	tank	▢	false	p_3
v_7	🚗	cabrio	2	true	p_4

peeps

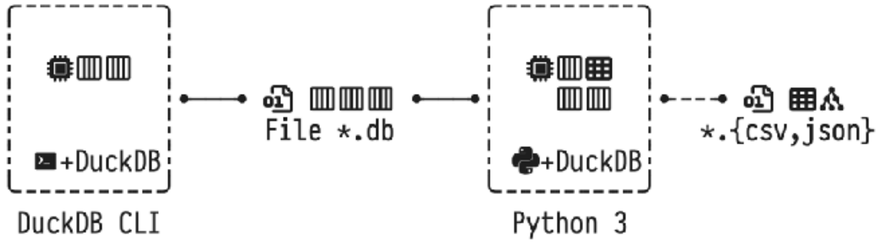
pid	pic	name	born
p_1	😐	Cleo	2013
p_2	🤓	Bert	1968
p_3	💀	Drew	▢
p_4	🧑	Alex	2002

```
SELECT v.vehicle, p.name AS driver, p.pic
FROM   vehicles AS v INNER JOIN peeps AS p ON (v.pid = p.pid)
                                          "equi-join" ↗
```

- Some rows in table peeps find multiple *join partners* in vehicles (like p_4 🧑), some find none (p_1 😐).
- General: A join between t_1 and t_2 may yield 0…$|t_1|\times|t_2|$ rows.

Design and Implementation of DuckDB Internals

DuckDB: A Tabular DBMS Inside Your Own Process



- DuckDB kernel and client share a **single process** ⬚.
  - Python: `import duckdb`, C/C++: link with `libduckdb`.
- Table data resides in a **single database file** *.db, native DuckDB data formats in file and in RAM are similar.
- DuckDB sees in-process client data and can **directly read/write client data structures** using SQL (**"zero copy"**, replacement scans).
- DuckDB allocates sizable RAM buffers (but can use disk for temporary storage if required).