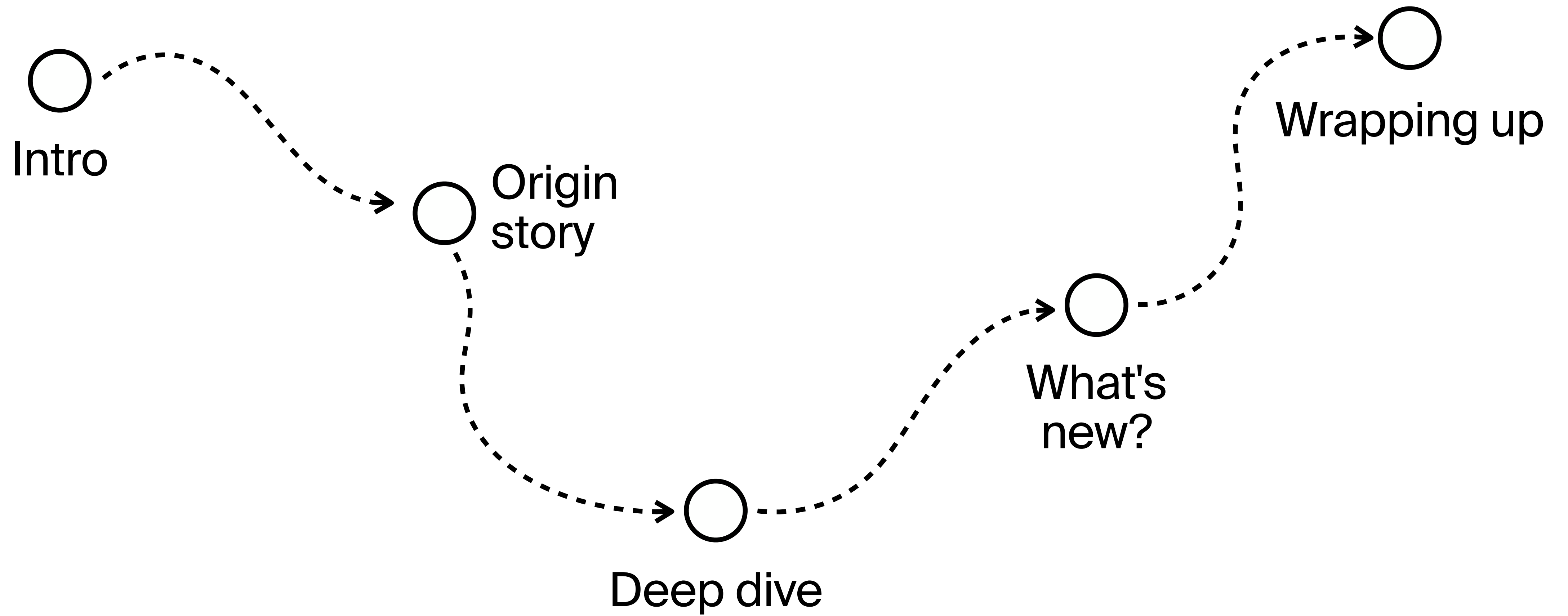
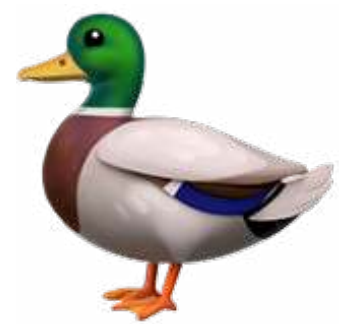


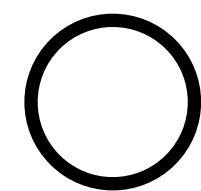
DuckDB: From Research Project to Enterprise Database System



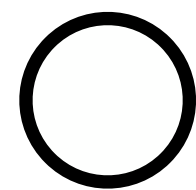




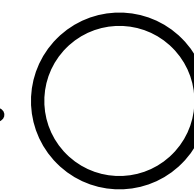
Intro



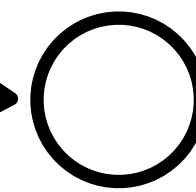
Origin
story



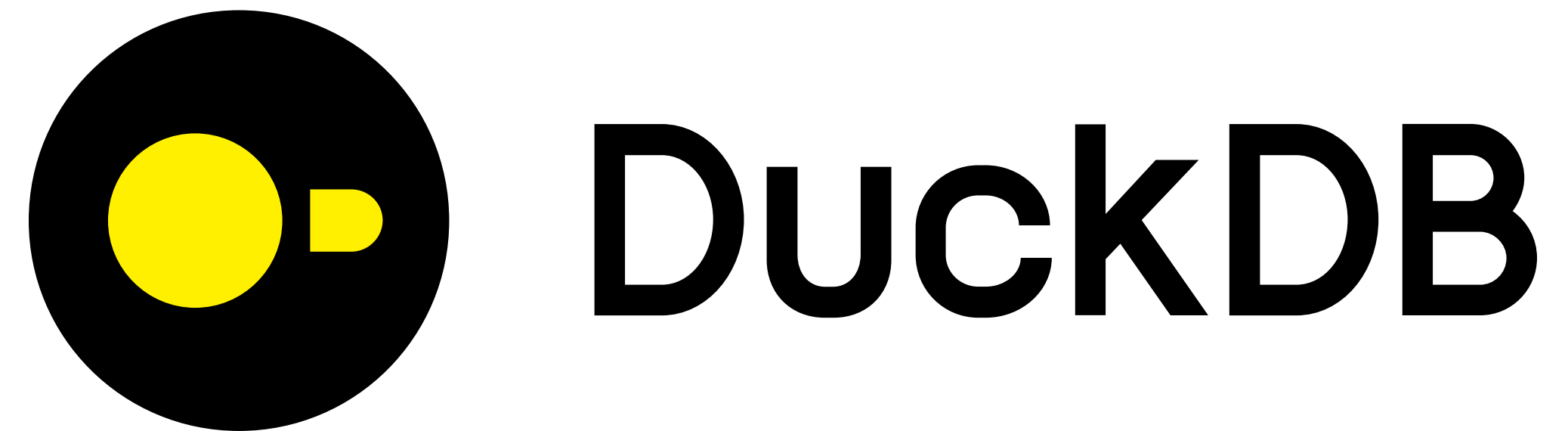
Deep dive



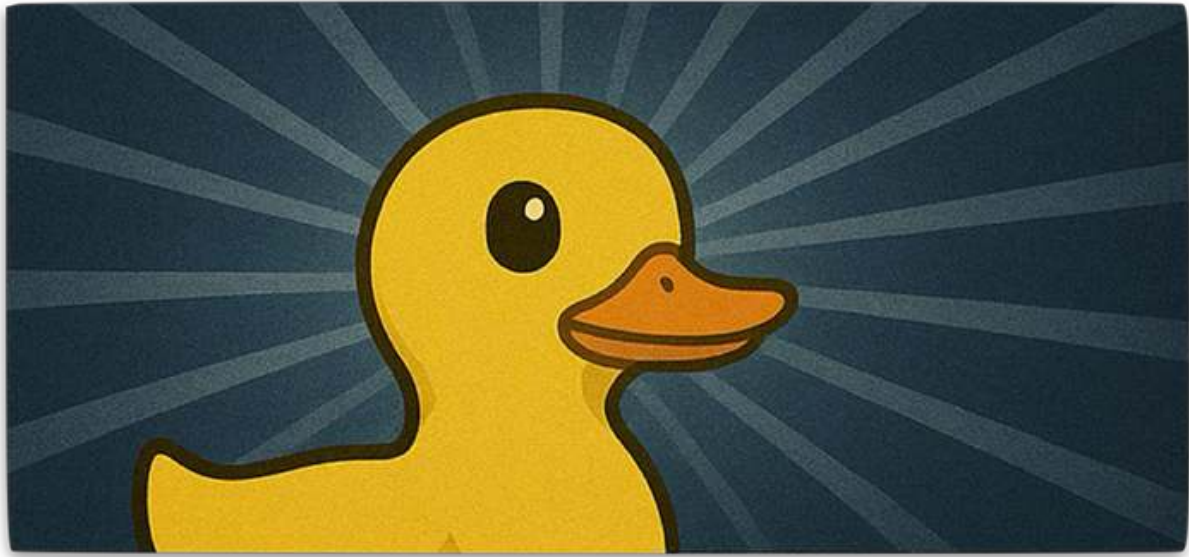
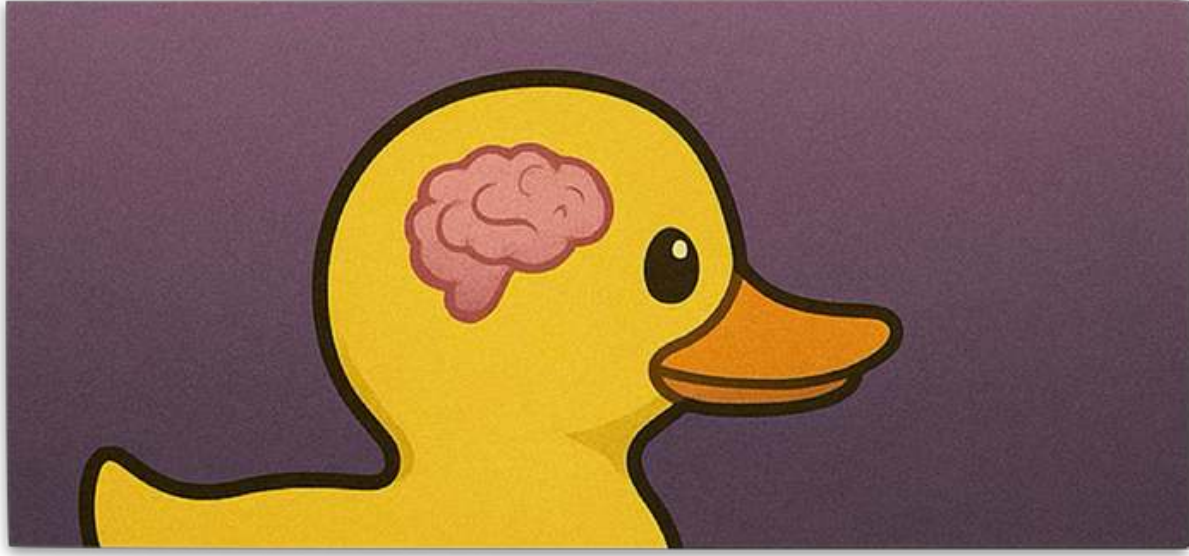
What's
new?

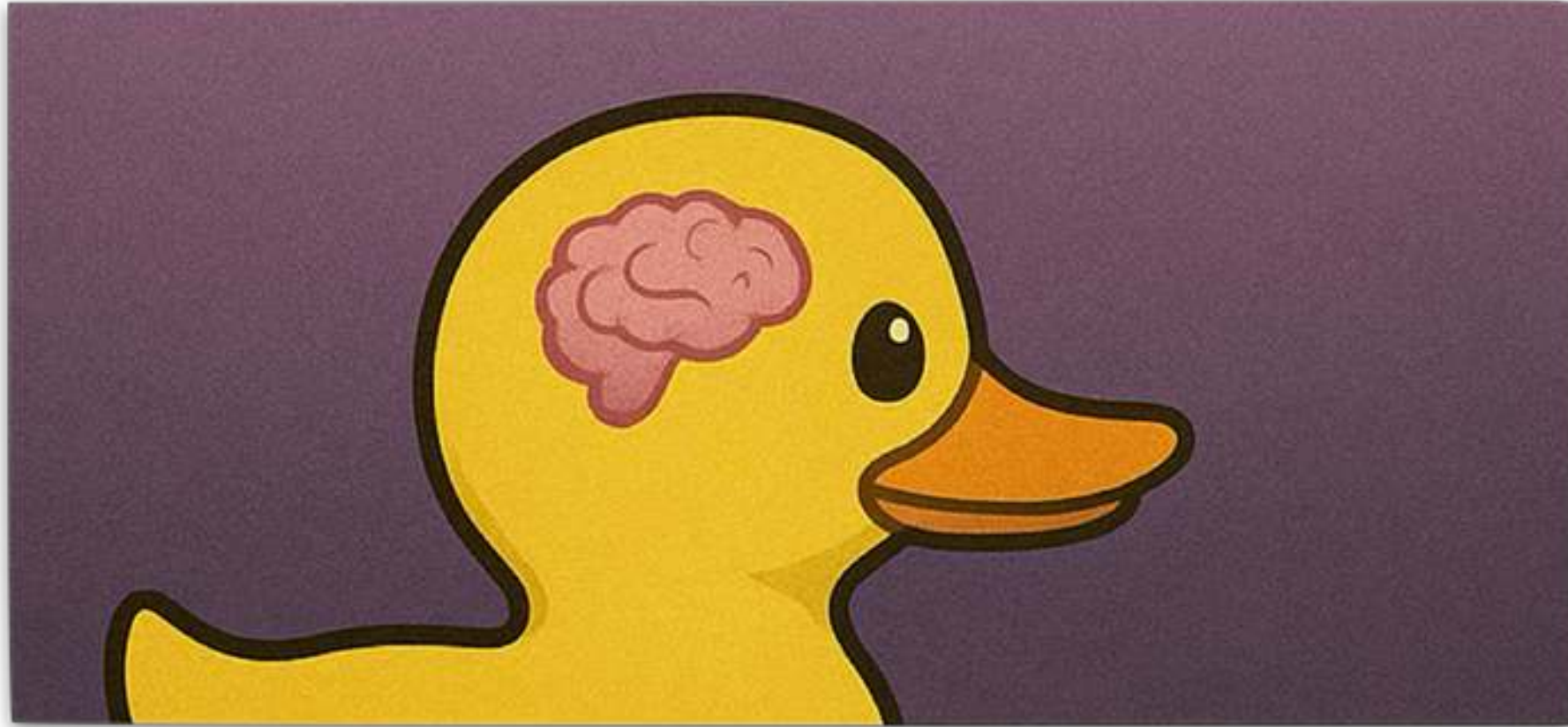


Wrapping up



Modern SQL database for analytics





Text files



- **Train archive**

Large dataset with all train services that are stored in the [train archive](#) (since 2019).



**Which stations had the worst delays
over the weekends of February 2025?**

```
$ wget https://blobs.duckdb.org/trains-2025-feb.csv.gz
$ gunzip trains-2025-feb.csv.gz
$ head -n 2 trains-2025-feb.csv
```

```
id,date,service_type,company,train_number,...
15277134,2025-02-01,Intercity,NS,1410,...
```

```
$ ./favorite_database
```

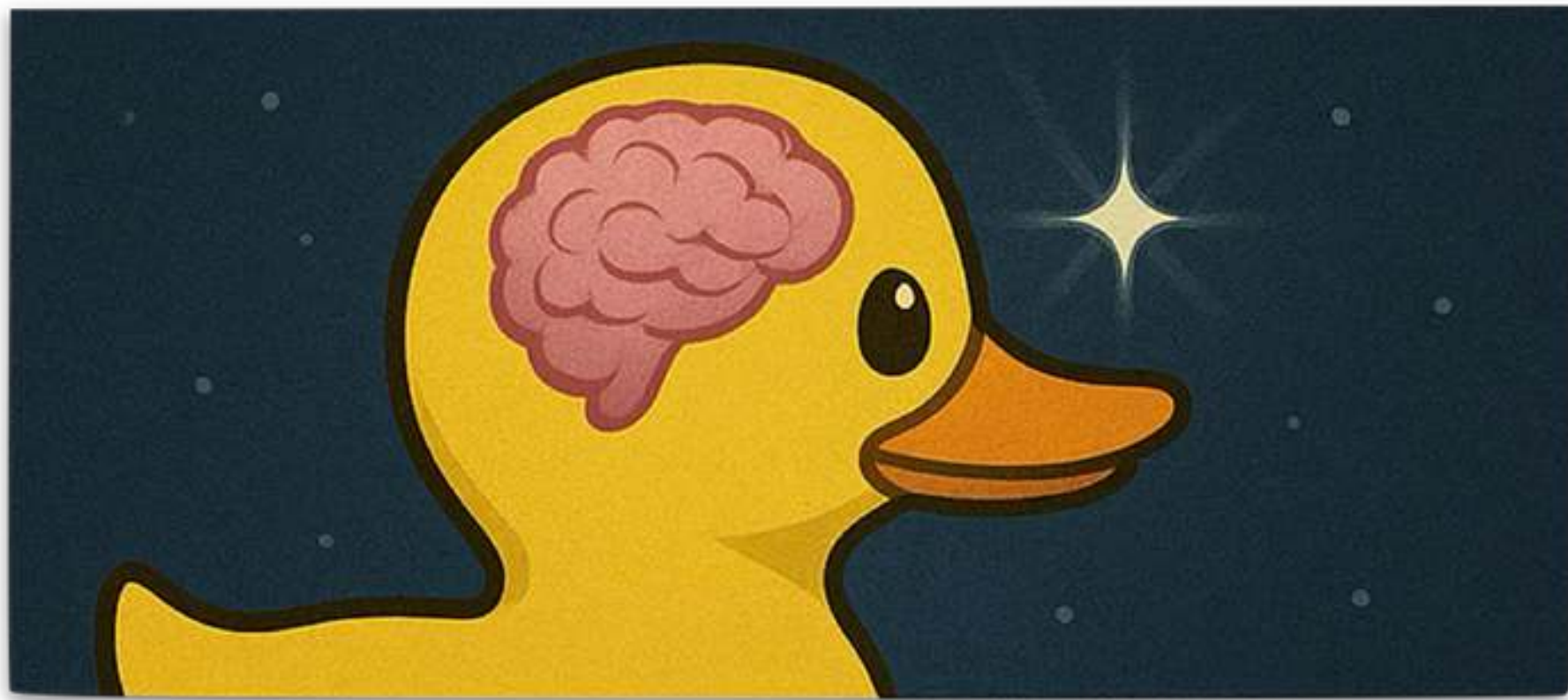
```
CREATE TABLE services (
    id BIGINT,
    date DATE,
    service_type VARCHAR,
    company VARCHAR,
    train_number BIGINT,
    ...
);
```



```
COPY services FROM 'trains-2025-feb.csv'
(HEADER true, DELIMITER ',');

SELECT
    station,
    avg(delay) AS avg_delay
FROM services
WHERE strftime('%a', date) IN ('Sat', 'Sun')
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



Text files (deluxe)

\$ duckdb

SELECT

station,
avg(delay) AS avg_delay

FROM <https://blobs.duckdb.org/trains-2025-feb.csv.gz>

WHERE strftime('%a', date) IN ('Sat', 'Sun')

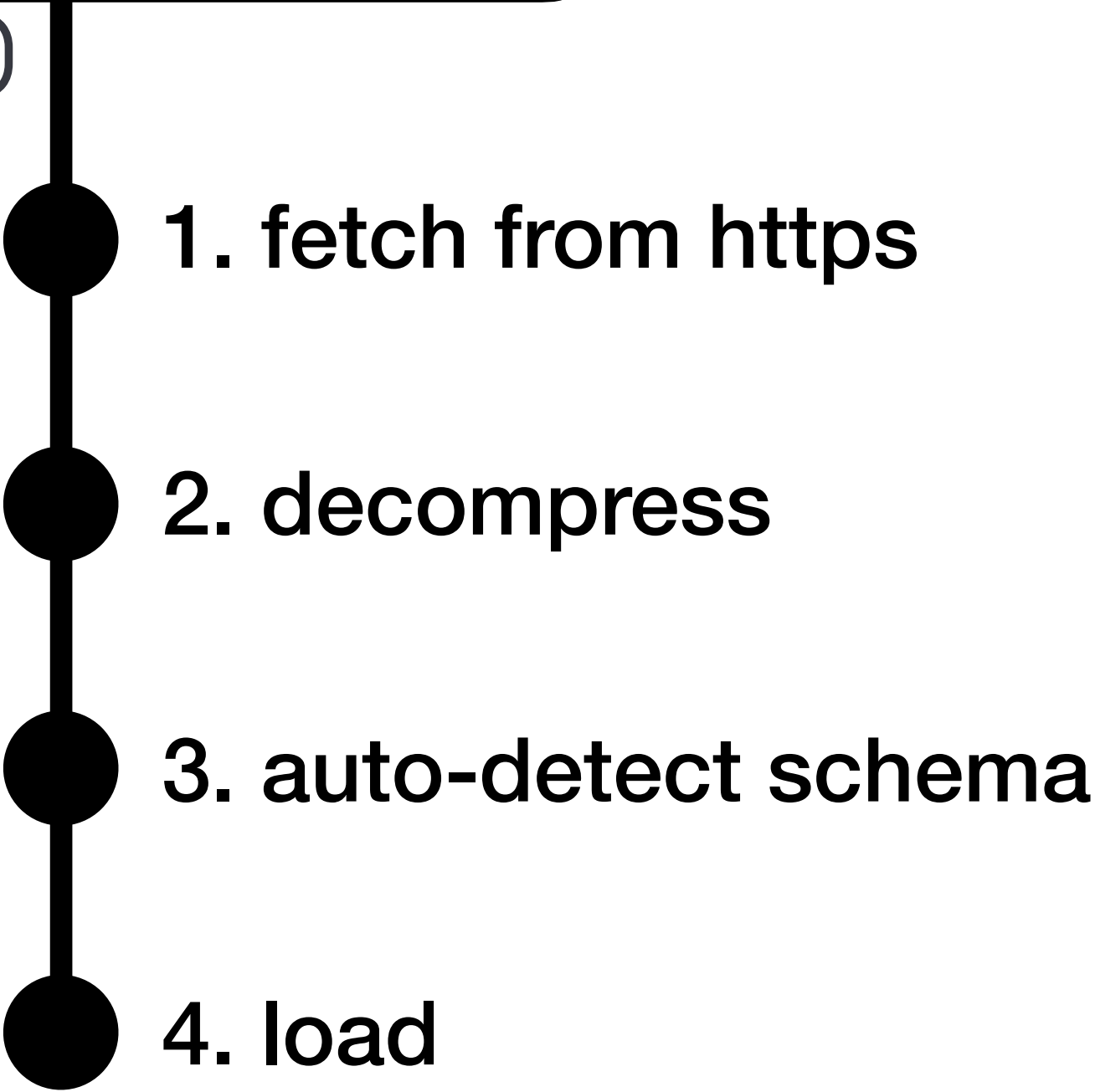
GROUP BY station

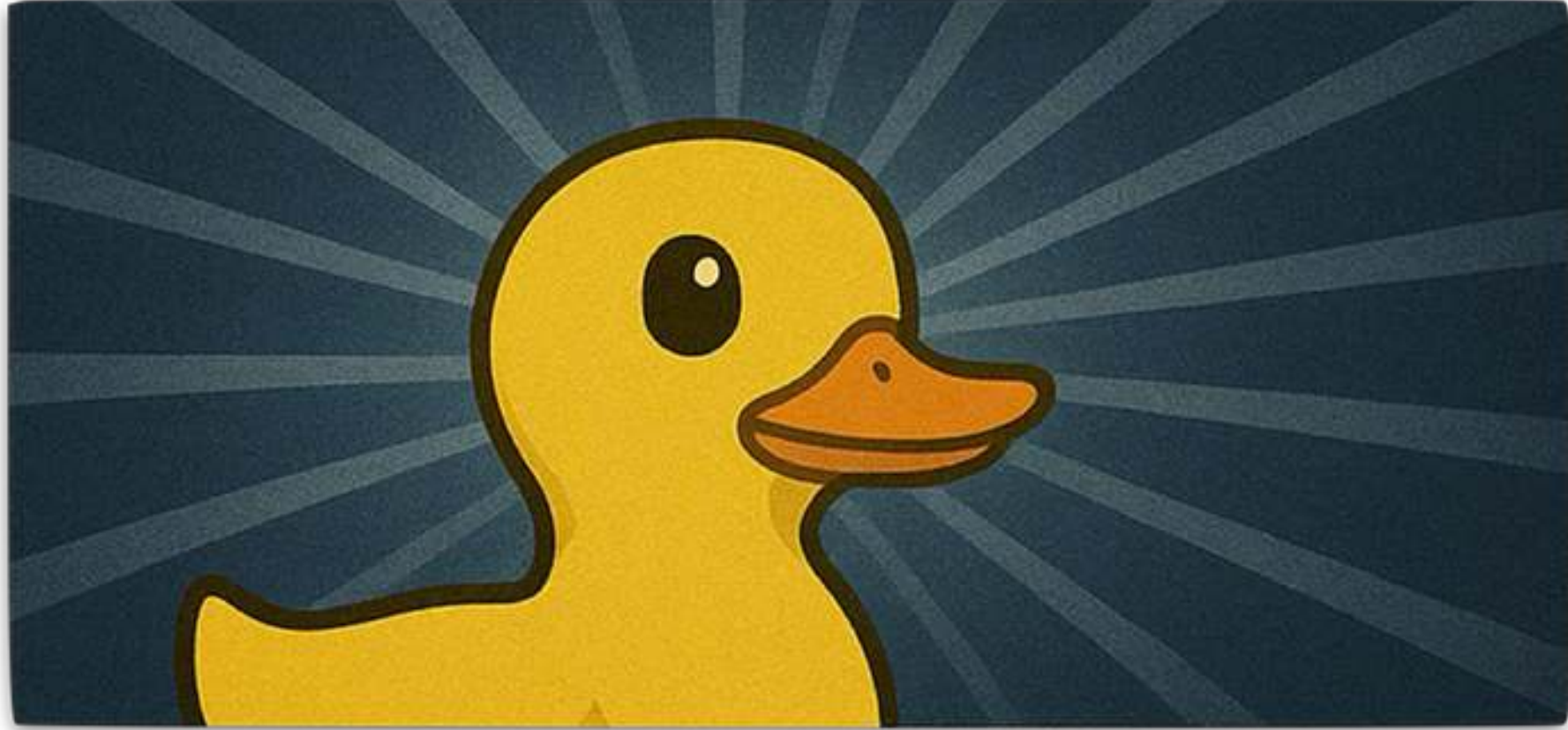
ORDER BY avg_delay DESC

LIMIT 3;

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

(4 seconds)





Binary files

\$ duckdb

SELECT

station,

avg(delay) AS avg_delay

FROM 'https://blobs.duckdb.org/trains-2025-feb.parquet'

WHERE strftime('%a', date) IN ('Sat', 'Sun')

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

(1 second)



Database



The worst delay in 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;
```

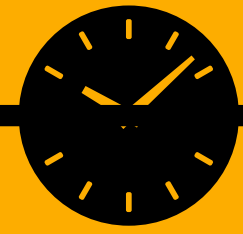
date	train_number	delay
2025-02-27	420	174



Change the train's number!

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

(23 rows updated)

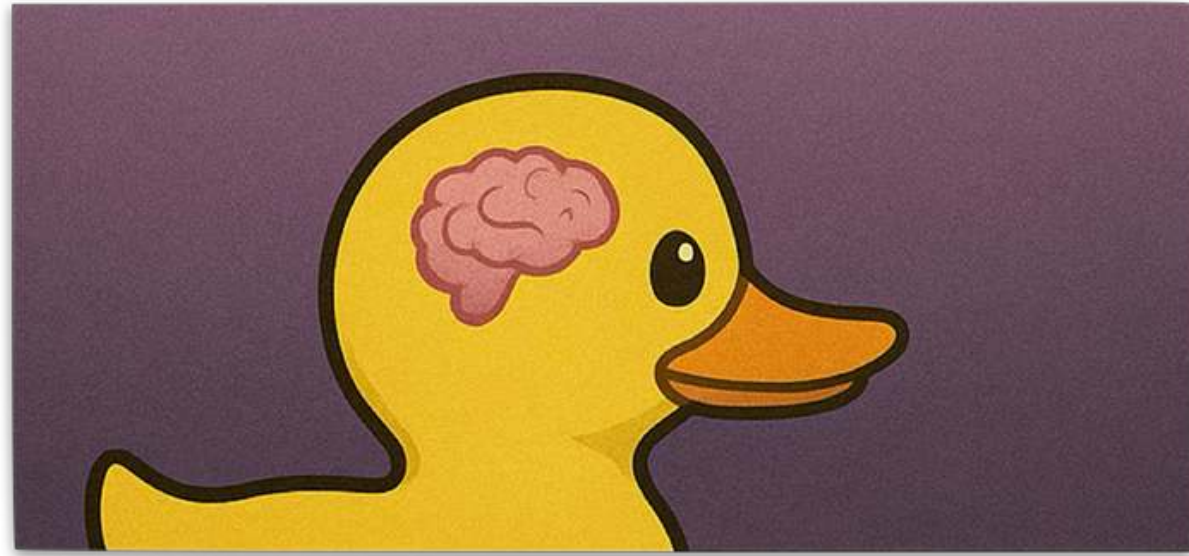


~~The worst delay in Amsterdam Centraal is almost 3 hours!~~

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

date	train_number	delay
0 rows		

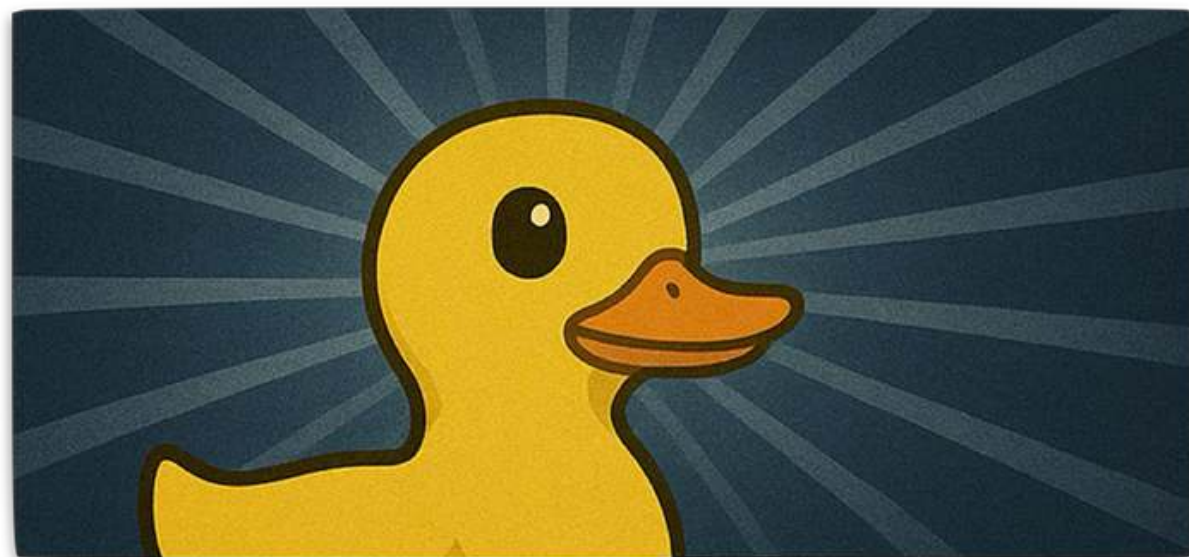




Text files



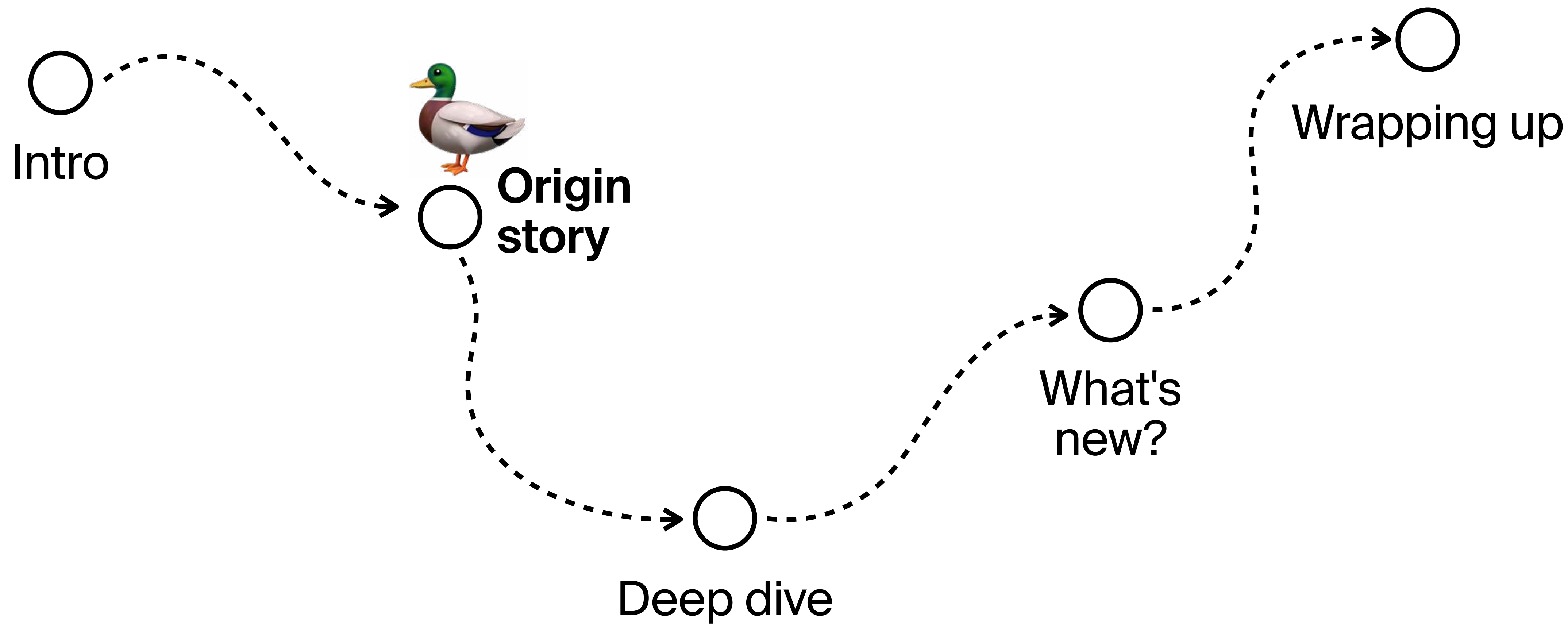
Text files (deluxe)

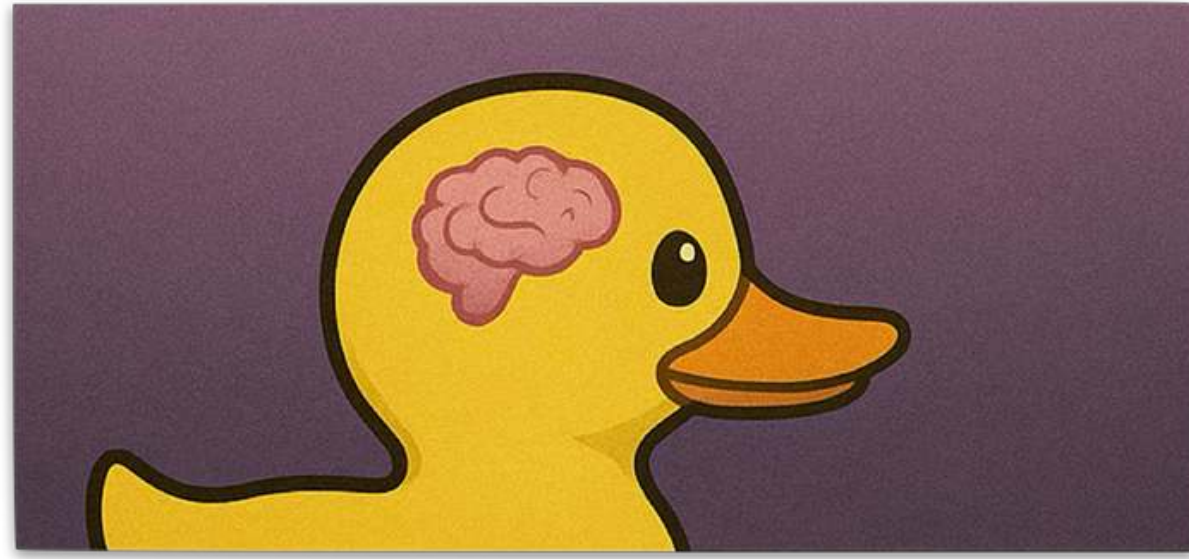


Binary files

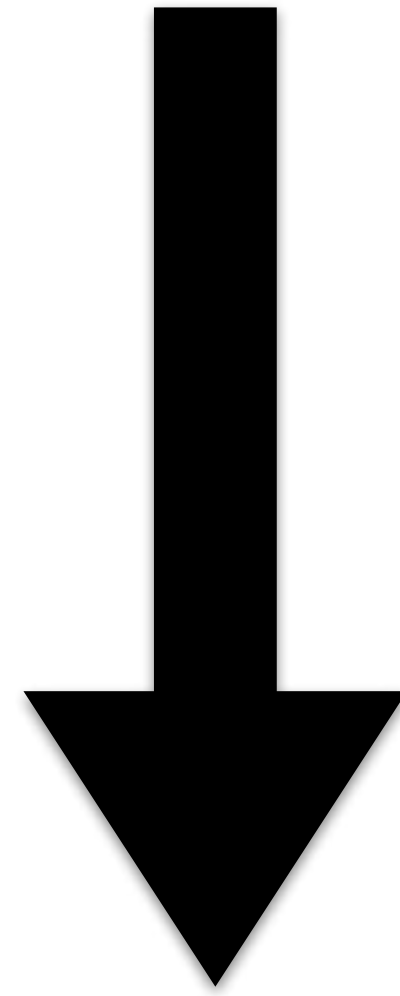


Database



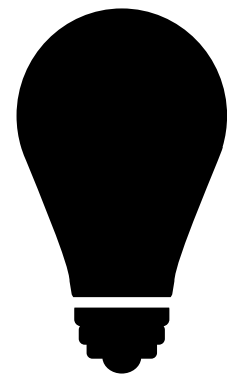


Text files



Database

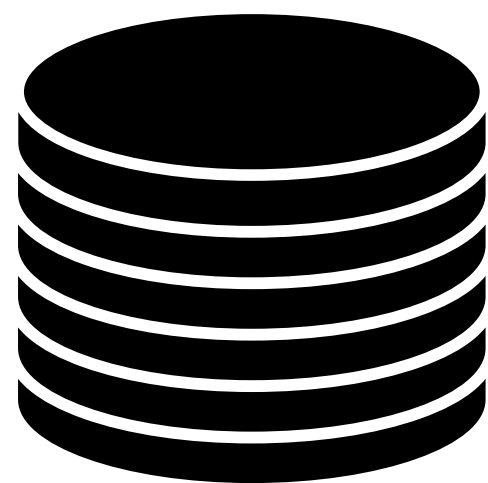




Insight #1:
“Big Data” is overrated

Data processing landscape, ~2008

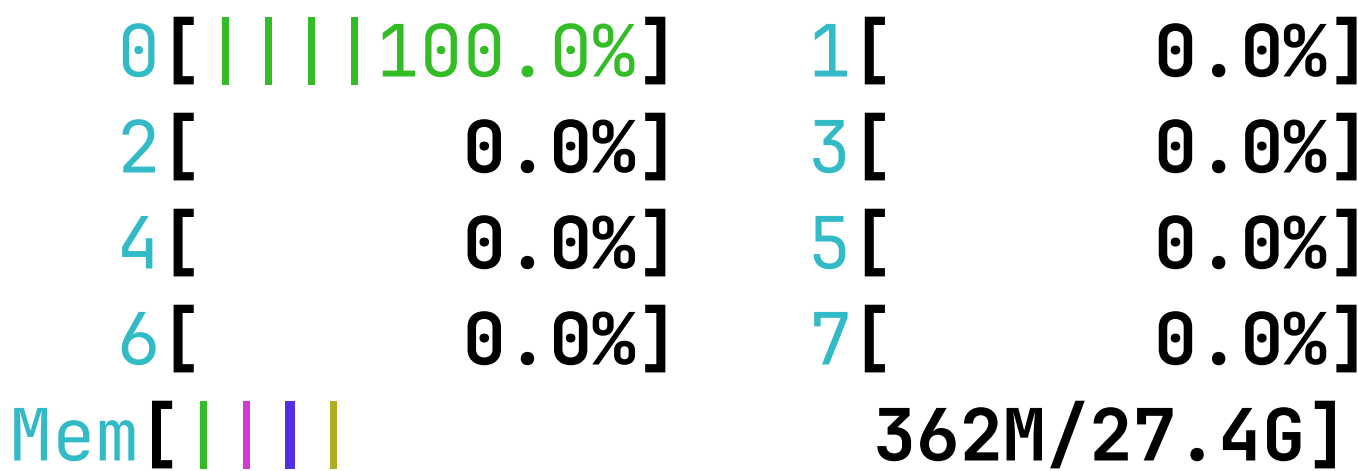
Lots of data



Weak hardware



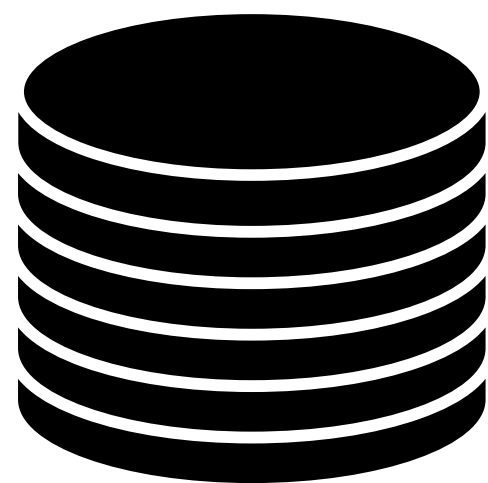
Inefficient software



Data processing landscape, ~2018

Lots of data

but most queries
only access a small amount



Strong hardware

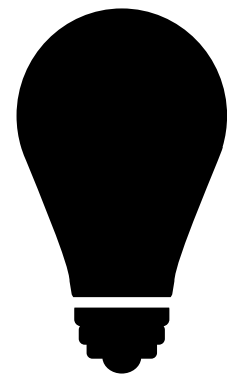


New techniques

columnar storage

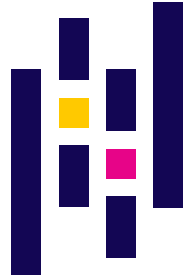
vectorization

compression



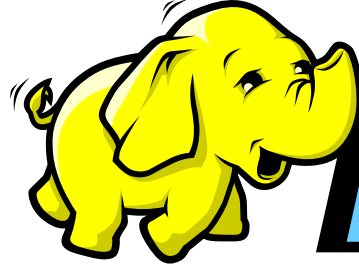
Insight #2:

Target data science workloads

 pandas

data.tables

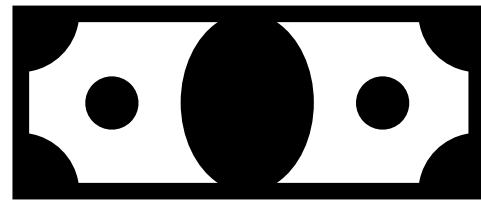
?

 **hadoop**

 **APACHE
Spark**TM

data size

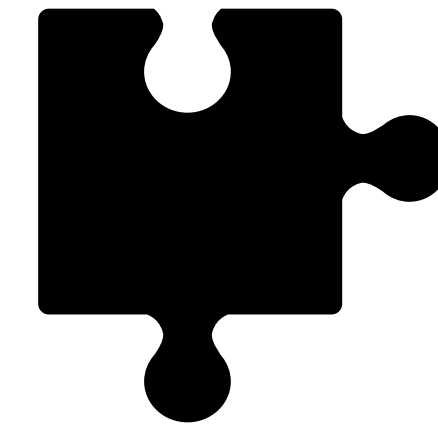
Resistance: Why use a database?



Expensive



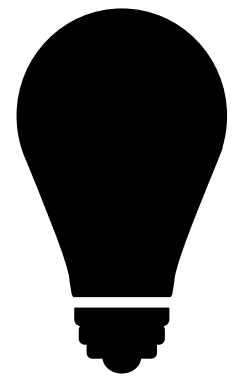
**Difficult
to operate**



**Unfriendly
interface**

```
$ wget https://blobs.duckdb.org/trains-2025-feb.csv.gz  
$ gunzip trains-2025-feb.csv.gz  
$ ./favorite_database
```

```
CREATE TABLE services (...);  
COPY services FROM 'trains-2025-feb.csv' (...);  
SELECT ...;
```

Insight #3:
Focus on usability

End-to-end performance

setup

define schema

load

write queries

run queries

End-to-end performance

setup

define schema

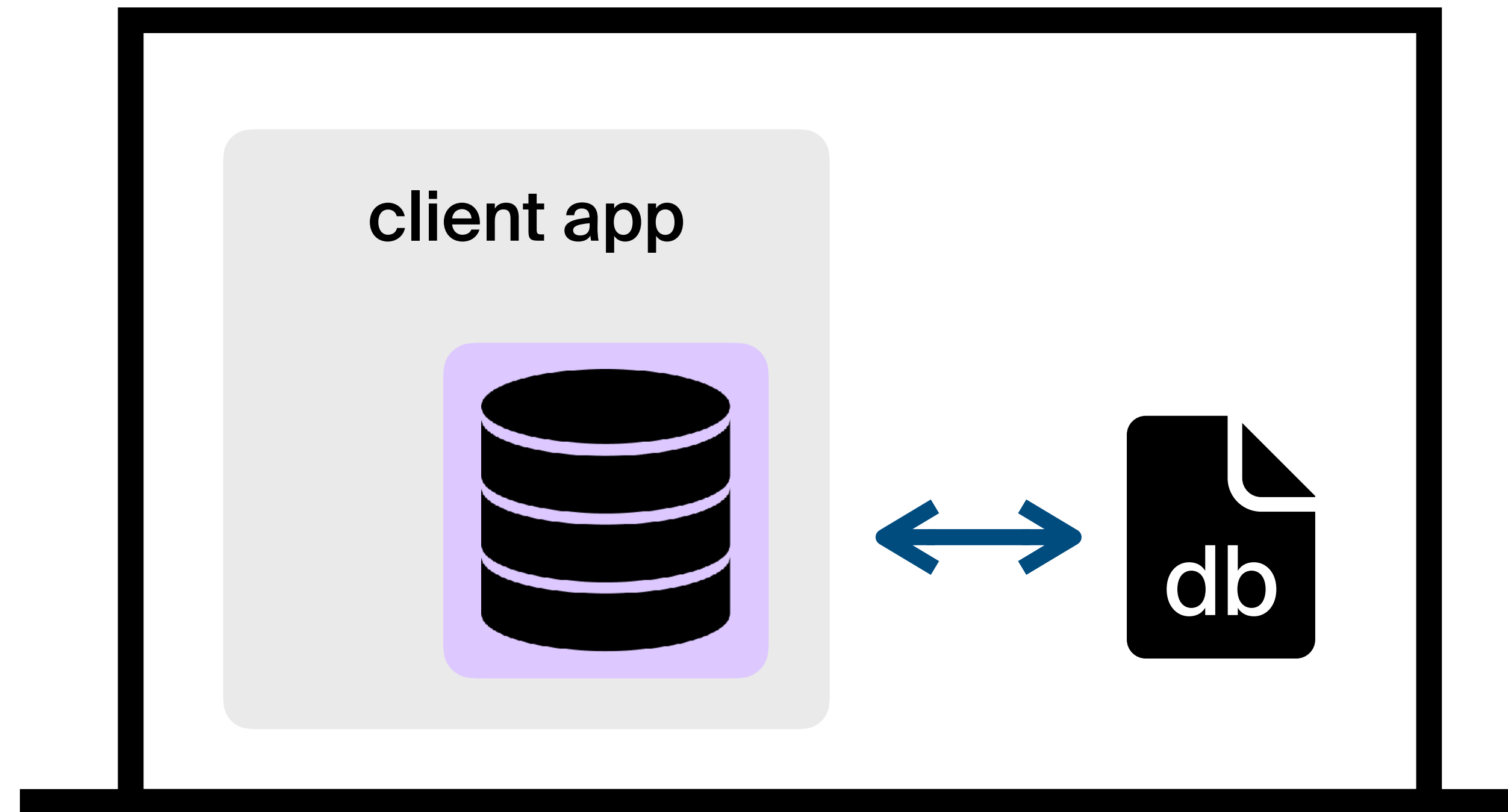
load

write queries

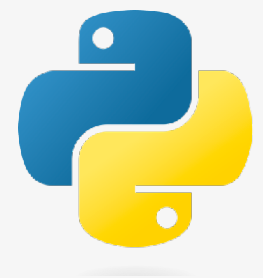
run queries

In-process architecture

setup



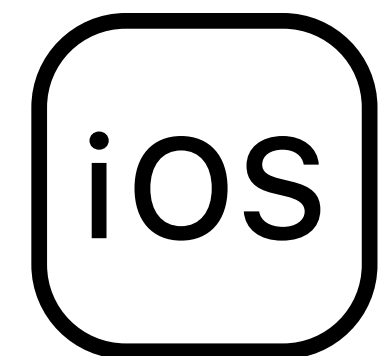
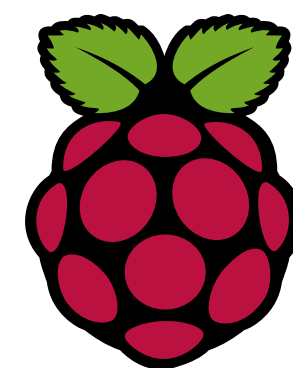
DuckDB clients



```
pip install duckdb
```



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



End-to-end performance

setup

define schema

load

write queries

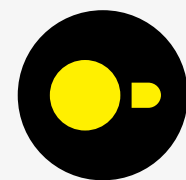
run queries

Friendly SQL

define schema

write queries

```
SELECT
  date,
  station,
  avg(delay),
  -- min(delay),
FROM 'https://blobs.duckdb.org/trains-2025-feb.csv.gz'
GROUP BY ALL;
```



DuckDB

2022



databricks

2023



snowflake

2023



Google
Big Query

2024

pandas integration

```
import duckdb
import pandas as pd
```

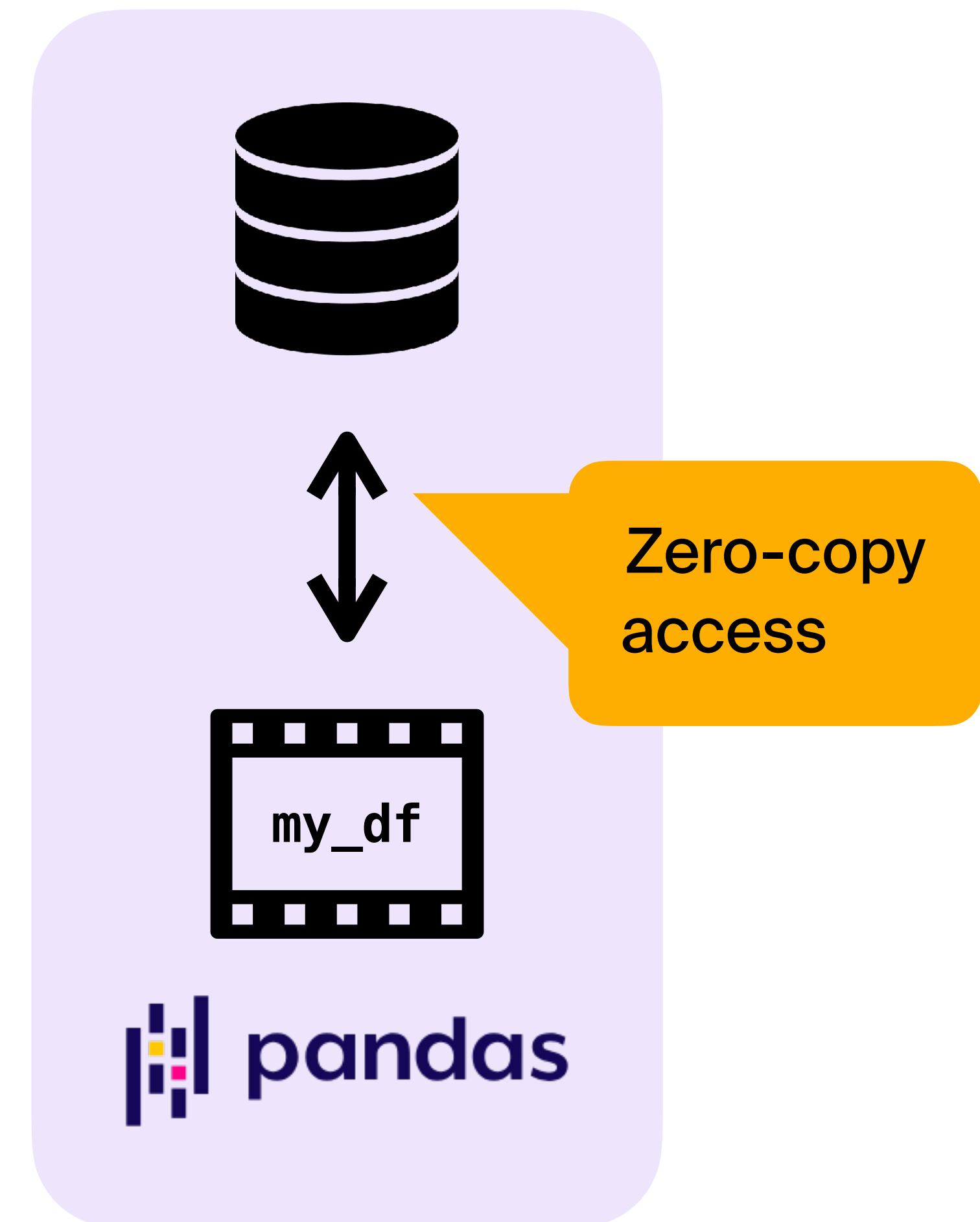
```
my_df = pd.DataFrame \
    .from_dict({'a': [42, 43]})
```

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

```
res.df()
```

```
#          s
# 0    42.5
```

Replacement scan



Python UDFs

```
import duckdb
from faker import Faker
```

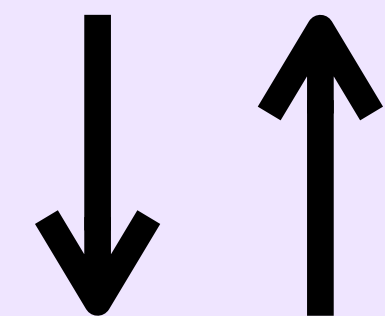
```
def get_random_name():
    return Faker().name()
```

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

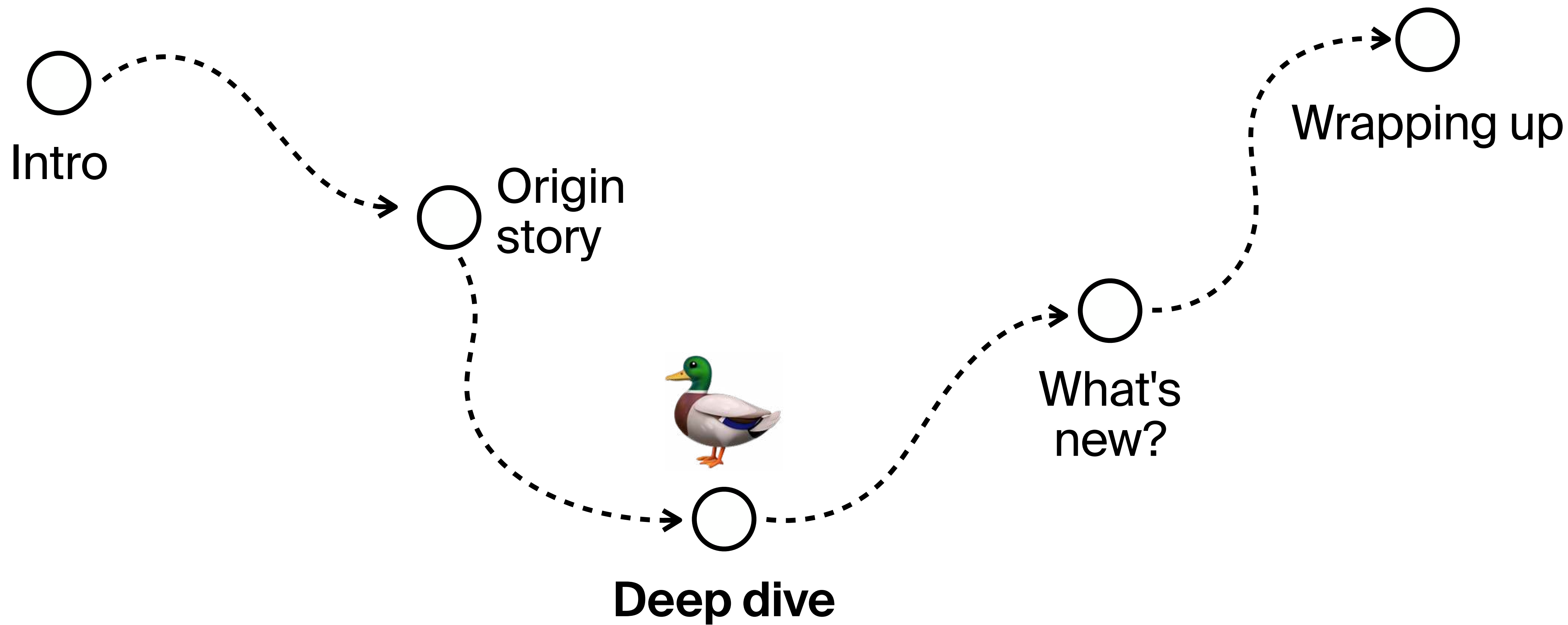
```
duckdb.sql("SELECT random_name()") \
    .fetchall()
```

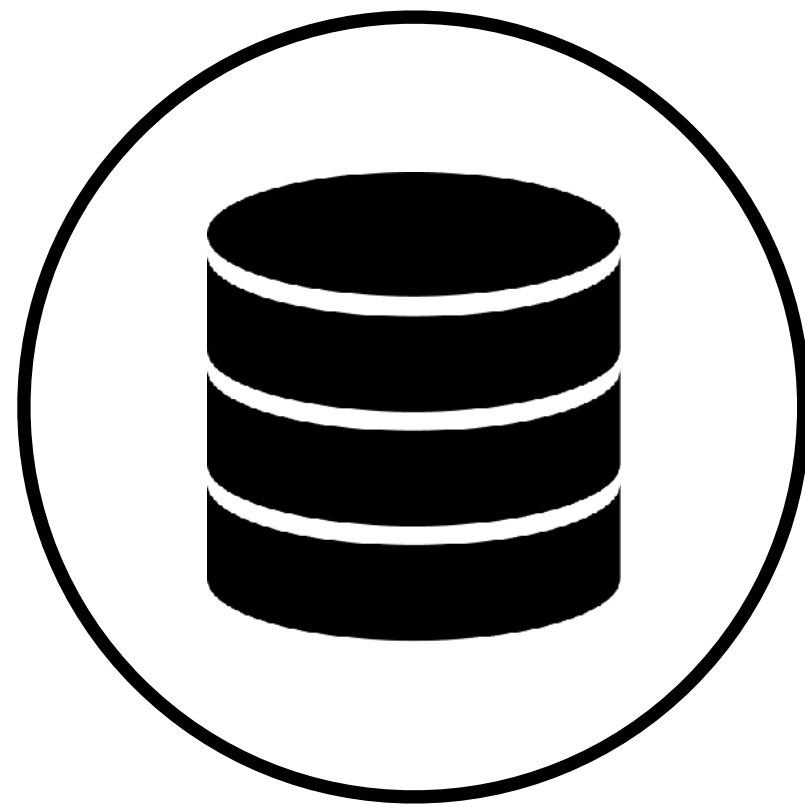


"SELECT random_name()"



def get_random_name()





Storage and execution

date station delay		
Feb 15	Delft	0.0
Feb 15	Edam	0.0
Feb 15	Breda	0.0
Feb 15	Delft	1.6

row-based storage

date	station	delay
Feb 15	Delft	0.0
Feb 15	Edam	0.0
Feb 15	Breda	0.0
Feb 15	Delft	1.6

row-based storage

date station delay

Feb 15	Delft	0.0	Feb 15	Edam	0.0	Feb 15	Breda	0.0	Feb 15	Delft	1.6
--------	-------	-----	--------	------	-----	--------	-------	-----	--------	-------	-----

column-based storage

date

station

delay

Feb 15

Delft

0.0

Feb 15

Edam

0.0

Feb 15

Breda

0.0

Feb 15

Delft

1.6

column-based storage

date

station

delay

date

station

delay

Feb 15

Feb 15

Feb 15

Feb 15

Delft

Edam

Breda

Delft

0.0

0.0

0.0

1.6

column-based storage

date

station

delay

date

station

delay

Feb 15

Delft

Edam

Breda

Delft

0.0 (x3)

1.6

column-based storage

date station delay

date

station

delay

Feb 15

Delft

Edam

Breda

Delft

0.0 (x3)

1.6

Row-based execution

date	station	delay
Feb 15	Delft	0.0
Feb 15	Edam	0.0
Feb 15	Breda	0.0
Feb 15	Delft	1.6
Feb 16	Utrecht	2.5
Feb 16	Gouda	0.0
Feb 16	Edam	0.8
Feb 23	Delft	3.7



What is the average delay per day?

result

date	avg delay
Feb 15	0.4
Feb 16	1.1
Feb 23	3.7

Simple but inefficient

Reads more data
than necessary

Not CPU-friendly

Column-based execution

date	station	delay
Feb 15	Delft	0.0
Feb 15	Edam	0.0
Feb 15	Breda	0.0
Feb 15	Delft	1.6
Feb 16	Utrecht	2.5
Feb 16	Gouda	0.0
Feb 16	Edam	0.8
Feb 23	Delft	3.7



What is the average delay per day?

thread 1

result

Column-based execution

date	station	delay
Feb 15	Delft	0.0
Feb 15	Edam	0.0
Feb 15	Breda	0.0
Feb 15	Delft	1.6
Feb 16	Utrecht	2.5
Feb 16	Gouda	0.0
Feb 16	Edam	0.8
Feb 23	Delft	3.7



What is the average delay per day?

thread 1

result

Column-based execution

date station delay

Delft
Edam
Breda
Delft
Utrecht
Gouda
Edam
Delft



What is the average delay per day?

thread 1

Feb 15	0.0
Feb 15	0.0
Feb 15	0.0
Feb 15	1.6
Feb 16	2.5
Feb 16	0.0
Feb 16	0.8
Feb 23	3.7

result

Column-based execution

date station delay

Delft
Edam
Breda
Delft
Utrecht
Gouda
Edam
Delft



What is the average delay per day?

thread 1

result

date avg delay

Feb 15	0.4
Feb 16	1.1
Feb 23	3.7

Simple but prone to OOM

Vectorized execution

date	station	delay
Feb 15	Delft	0.0
Feb 15	Edam	0.0
Feb 15	Breda	0.0
Feb 15	Delft	1.6
Feb 16	Utrecht	2.5
Feb 16	Gouda	0.0
Feb 16	Edam	0.8
Feb 23	Delft	3.7



What is the average delay per day?

thread 1

L1 cache

thread 2

L1 cache

Vectorized execution

date	station	delay
Feb 15	Delft	0.0
Feb 15	Edam	0.0
Feb 15	Breda	0.0
Feb 15	Delft	1.6
Feb 16	Utrecht	2.5
Feb 16	Gouda	0.0
Feb 16	Edam	0.8
Feb 23	Delft	3.7



What is the average delay per day?

thread 1

L1 cache

thread 2

L1 cache

Vectorized execution

date station delay

Delft
Edam
Breda
Delft
Utrecht
Gouda
Edam
Delft



What is the average delay per day?

thread 1

Feb 15	0.0
Feb 15	0.0
Feb 15	0.0
Feb 15	1.6

L1 cache

thread 2

Feb 16	2.5
Feb 16	0.0
Feb 16	0.8
Feb 23	3.7

L1 cache

Vectorized execution

date station delay

Delft
Edam
Breda
Delft
Utrecht
Gouda
Edam
Delft



What is the average delay per day?

thread 1

Feb 15	0.0
Feb 15	1.6

L1 cache

Feb 15	0.0
Feb 15	0.0

thread 2

Feb 16	0.8
Feb 23	3.7

L1 cache

Feb 16	2.5
Feb 16	0.0

Vectorized execution

date station delay

	Delft	
	Edam	
	Breda	
	Delft	
	Utrecht	
	Gouda	
	Edam	
	Delft	



What is the average delay per day?

thread 1

L1 cache

Feb 15	0.0
Feb 15	1.6

thread 2

L1 cache

Feb 16	0.8
Feb 23	3.7

Vectorized execution

date station delay

Delft
Edam
Breda
Delft
Utrecht
Gouda
Edam
Delft



What is the average delay per day?

thread 1

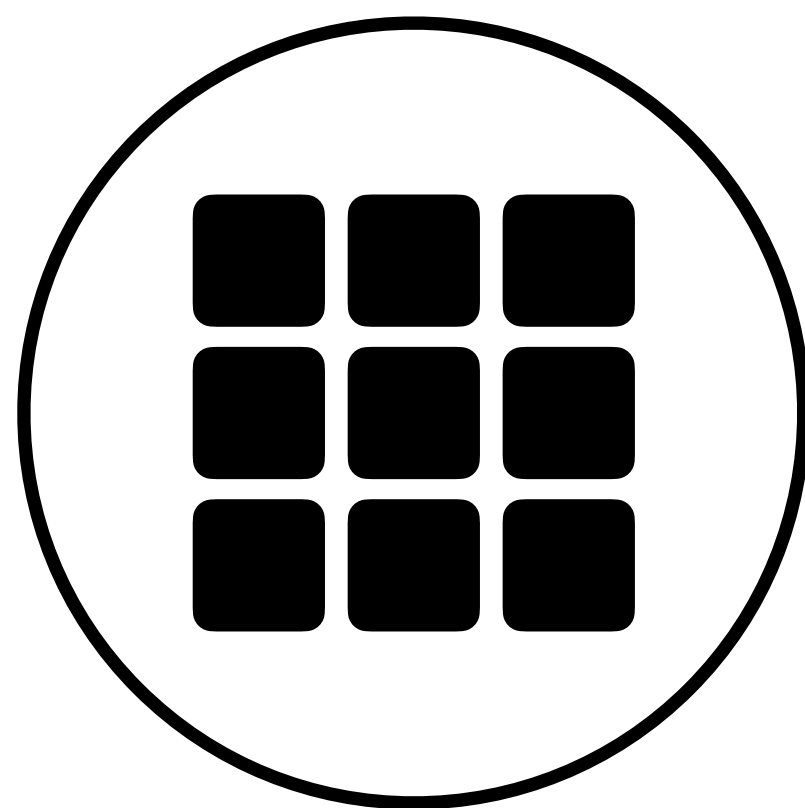
L1 cache

Prefetch-friendly

thread 2

Parallelism-friendly

Cache-friendly



Lightweight indexing



Which stations are affected by delays > 2 min?

row group 1

	date	station	delay
min	Feb 15	Breda	0.0
max	Feb 15	Delft	1.6

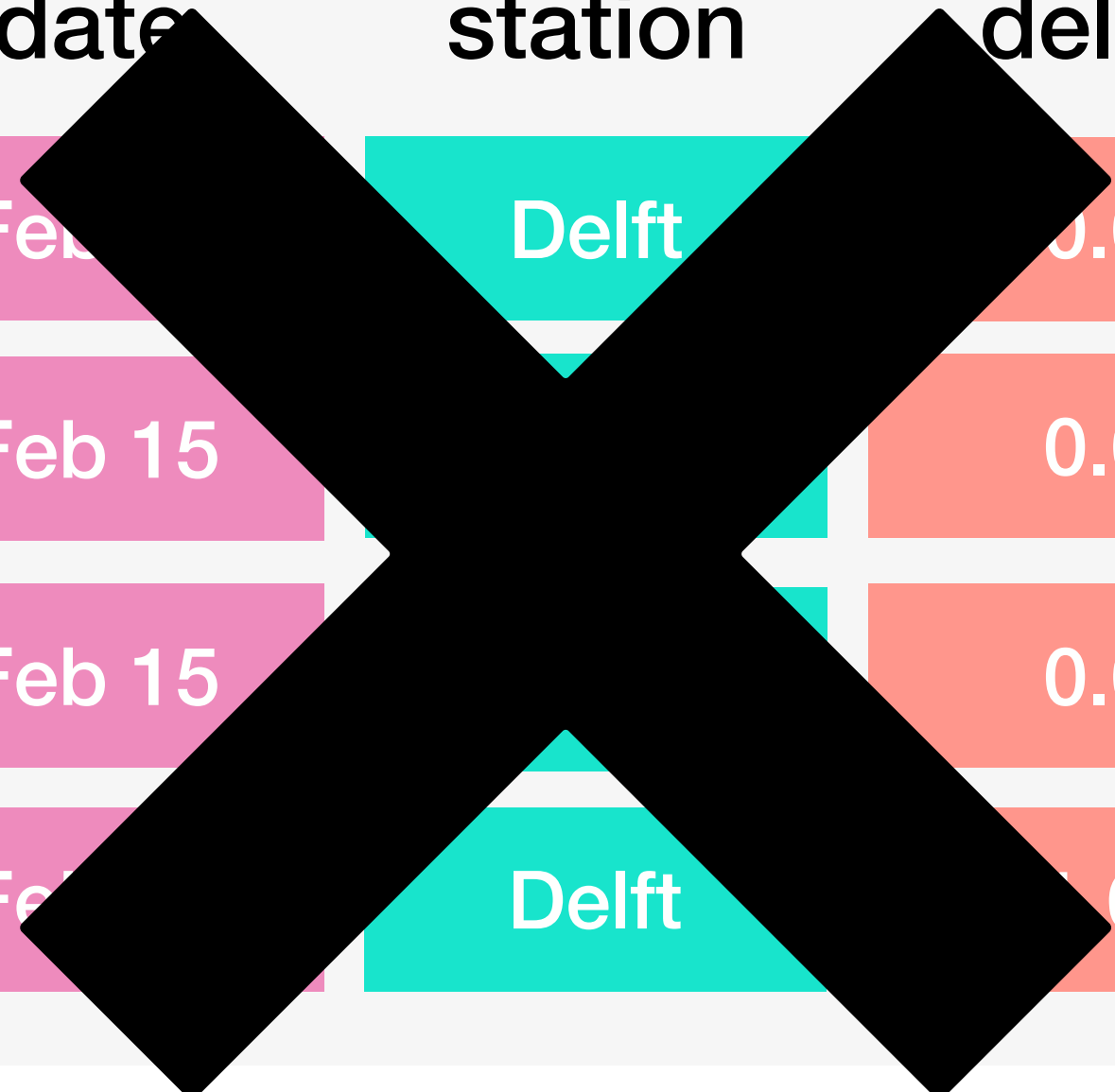


row group 2

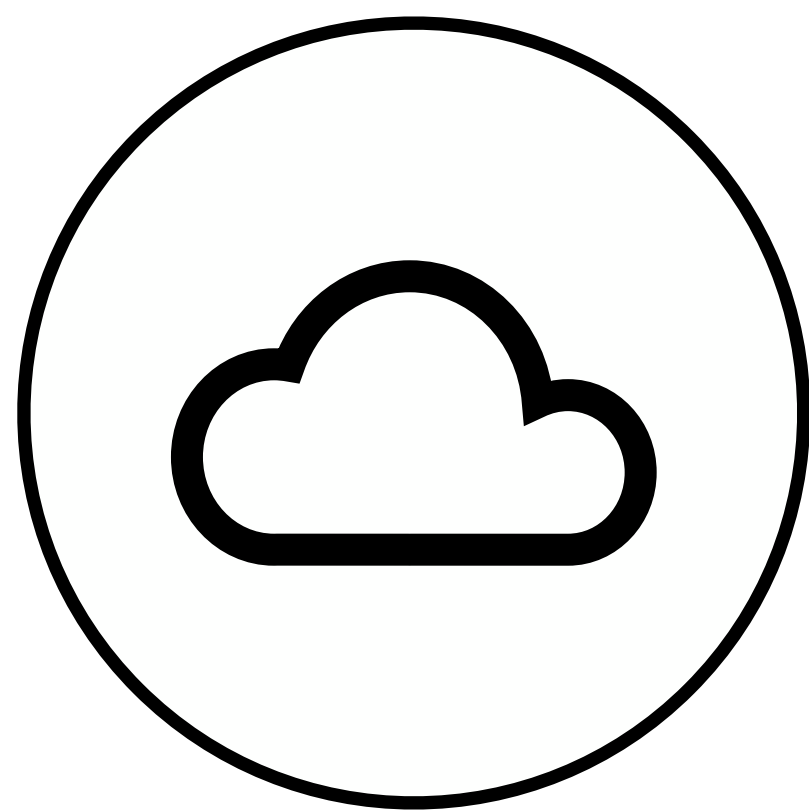
	date	station	delay
min	Feb 16	Edam	0.0
max	Feb 23	Delft	3.7



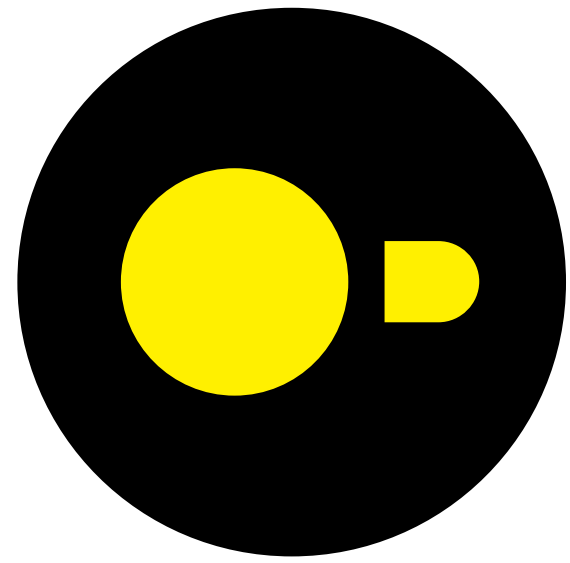
	date	station	delay
	Feb 15	Delft	0.0
	Feb 15		0.0
	Feb 15		0.0
	Feb 15	Delft	1.6



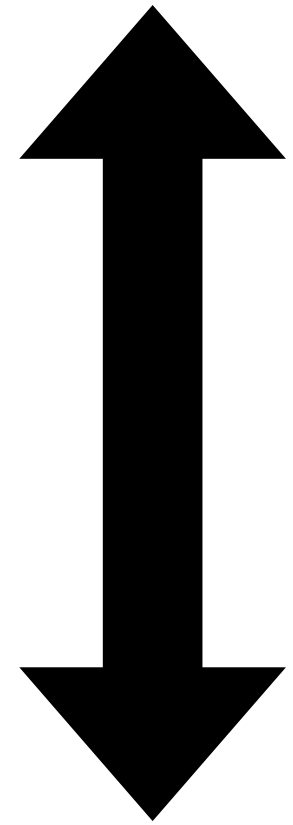
	date	station	delay
	Feb 16	Utrecht	2.5
	Feb 16	Gouda	0.0
	Feb 16	Edam	0.8
	Feb 23	Delft	3.7



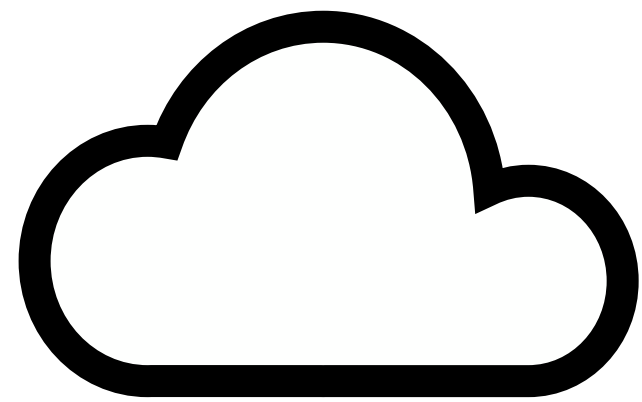
DuckDB in the cloud

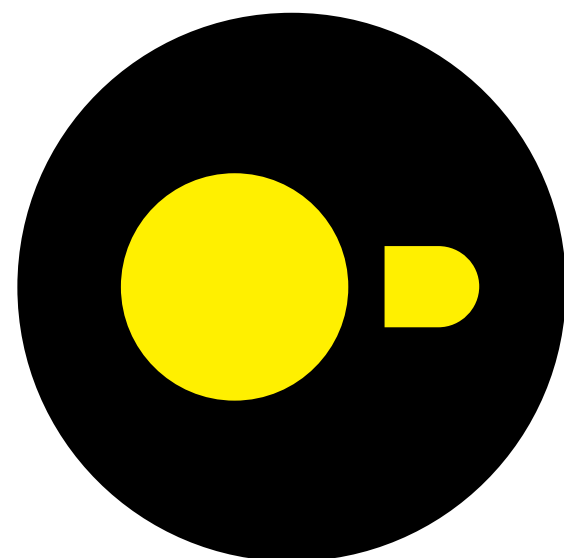


Which stations have
delays > 2 min?

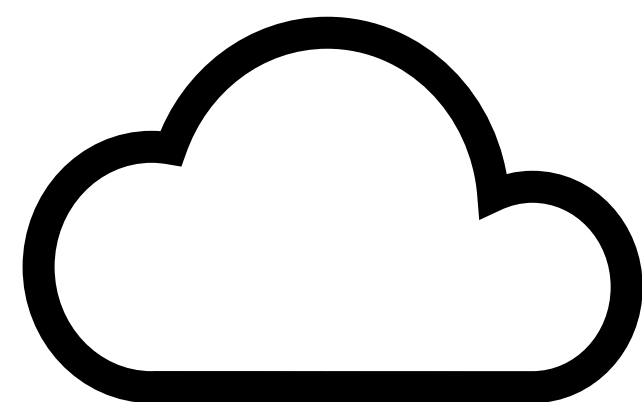
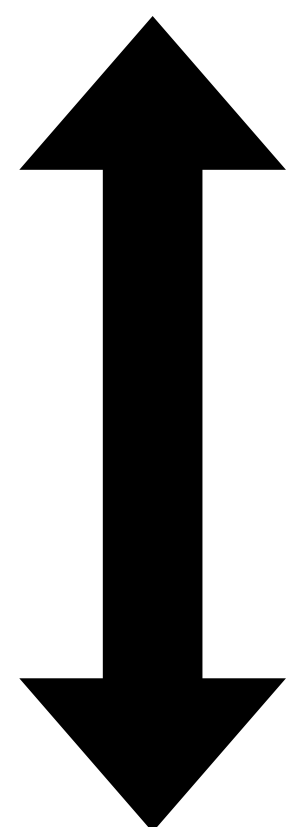


```
SELECT date, station, delay  
FROM 's3://my-bucket/trains-2025-feb.parquet'  
WHERE delay > 2;
```





Which stations have
delays > 2 min?



row group 1

	date	station	delay
min	Feb 15	Breda	0.0
max	Feb 15	Delft	1.6



date station delay

Feb 15	Delft	0.0
Feb 15		0.0
Feb 15		0.0

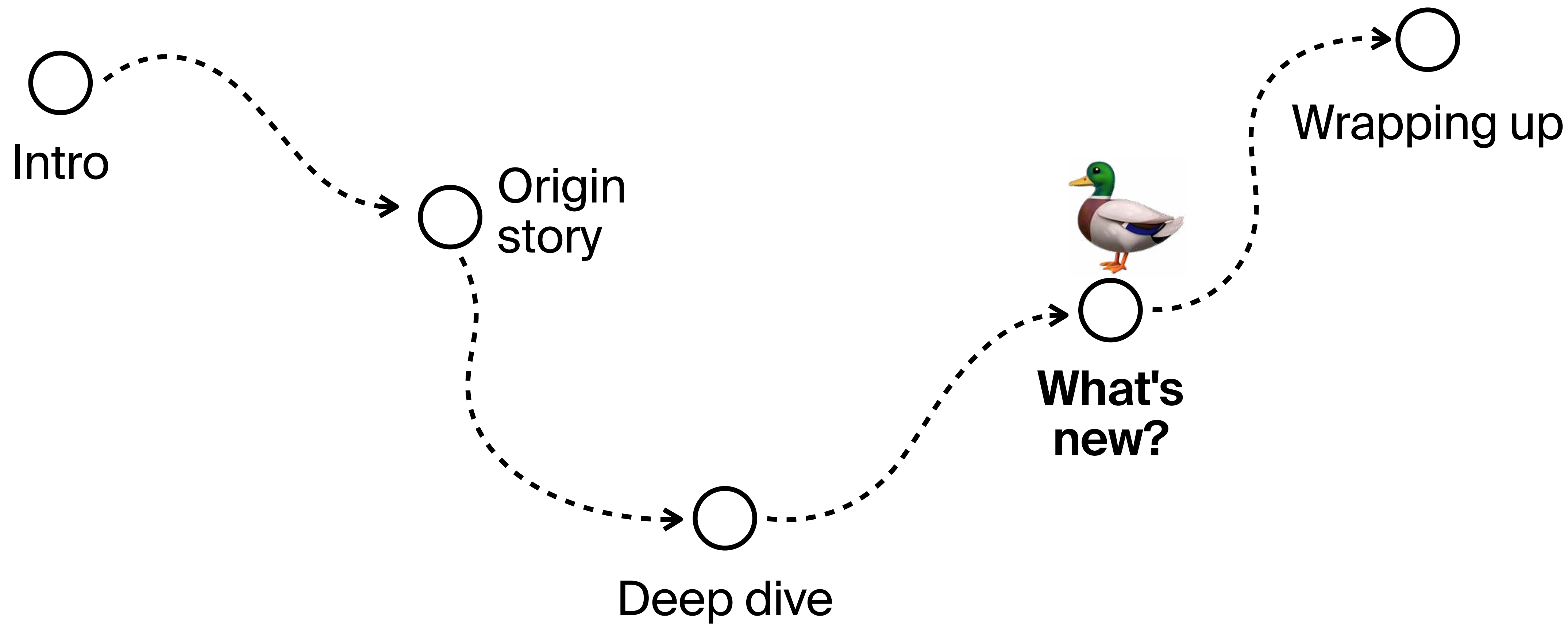
row group 2

	date	station	delay
min	Feb 16	Edam	0.0
max	Feb 23	Delft	3.7



date station delay

Feb 16	Utrecht	2.5
Feb 16	Gouda	0.0
Feb 16	Edam	0.8



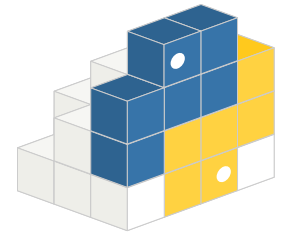
DuckDB: From Research Project to **Enterprise Database System**



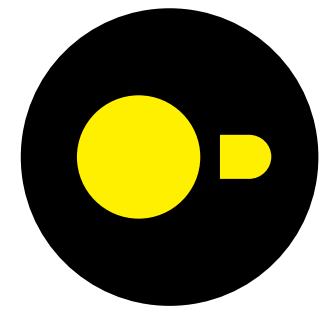
30k GitHub stars



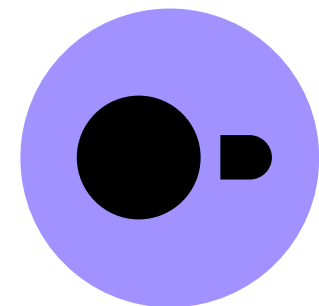
3M website visitors / month



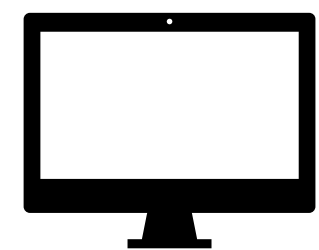
20M Python installs / month



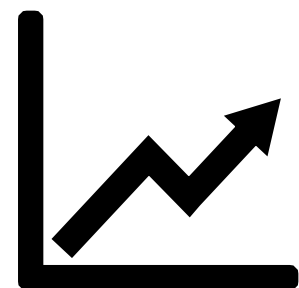
Released v1.0 last summer



Community extensions



Published the DuckDB UI



Lots of performance optimizations



✓ Instant SQL

```
1  SELECT
2      date,
3      train_number,
4      delay,
5  FROM services
6  WHERE station = 'Amsterdam Centraal'
7
```



Showing preview results

 date	123 train_number	123 delay
2025-02-01	1410	3
2025-02-01	701406	2
2025-02-01	1413	0
2025-02-01	1409	0

TPC-H scalability with DuckDB

1 TB



Raspberry Pi
16 GB RAM

10 TB

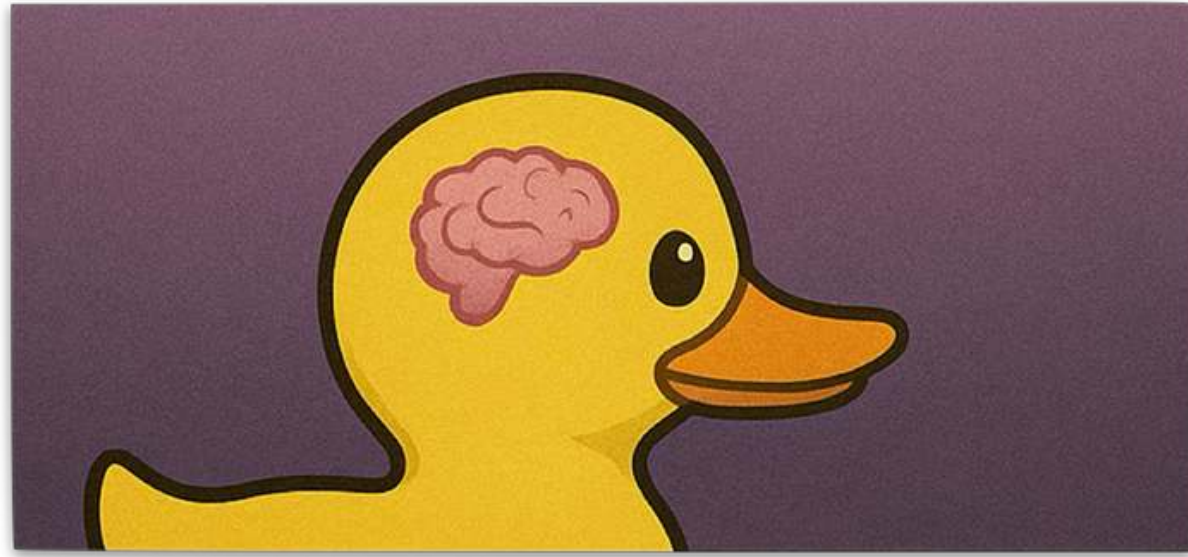


MacBook Pro
128 GB RAM

100 TB



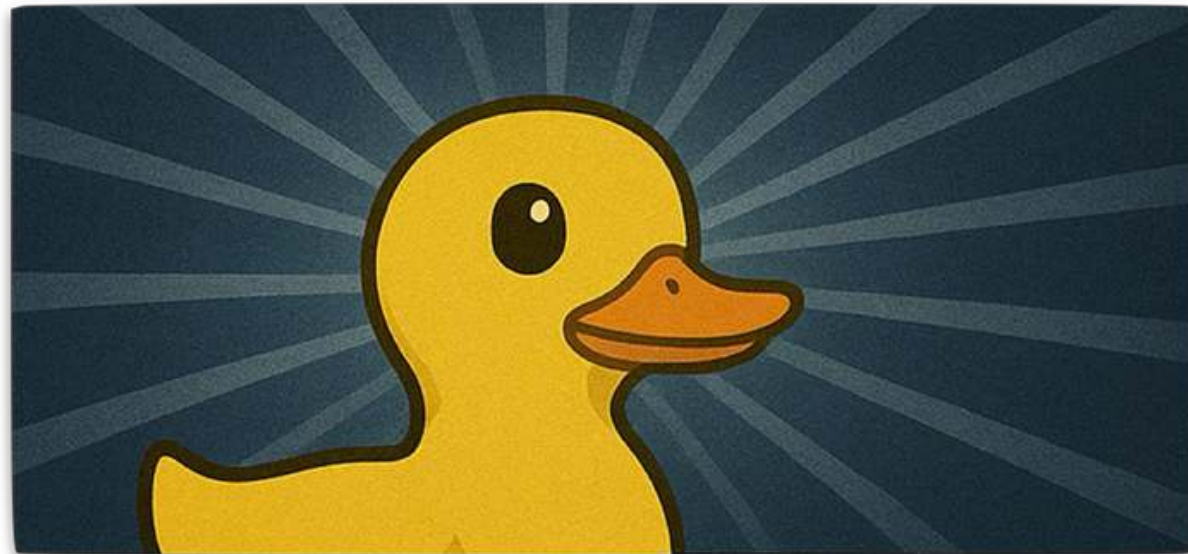
AWS EC2 instance
1.5 TB RAM



Text files



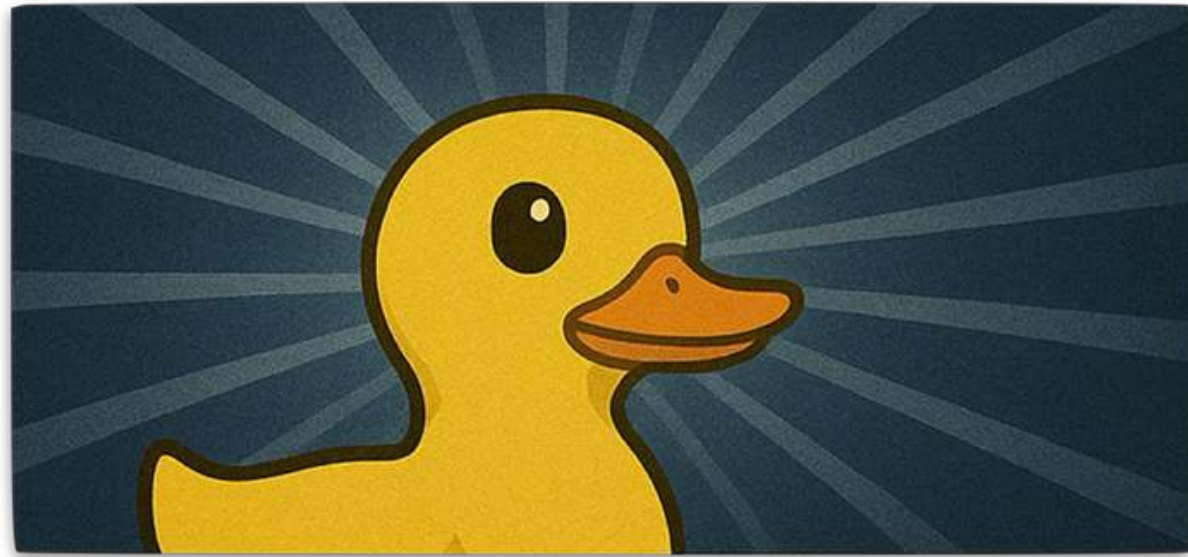
Text files (deluxe)



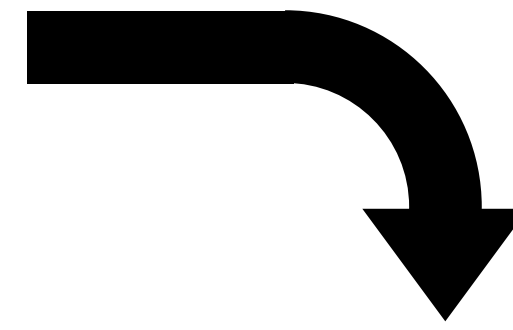
Binary files



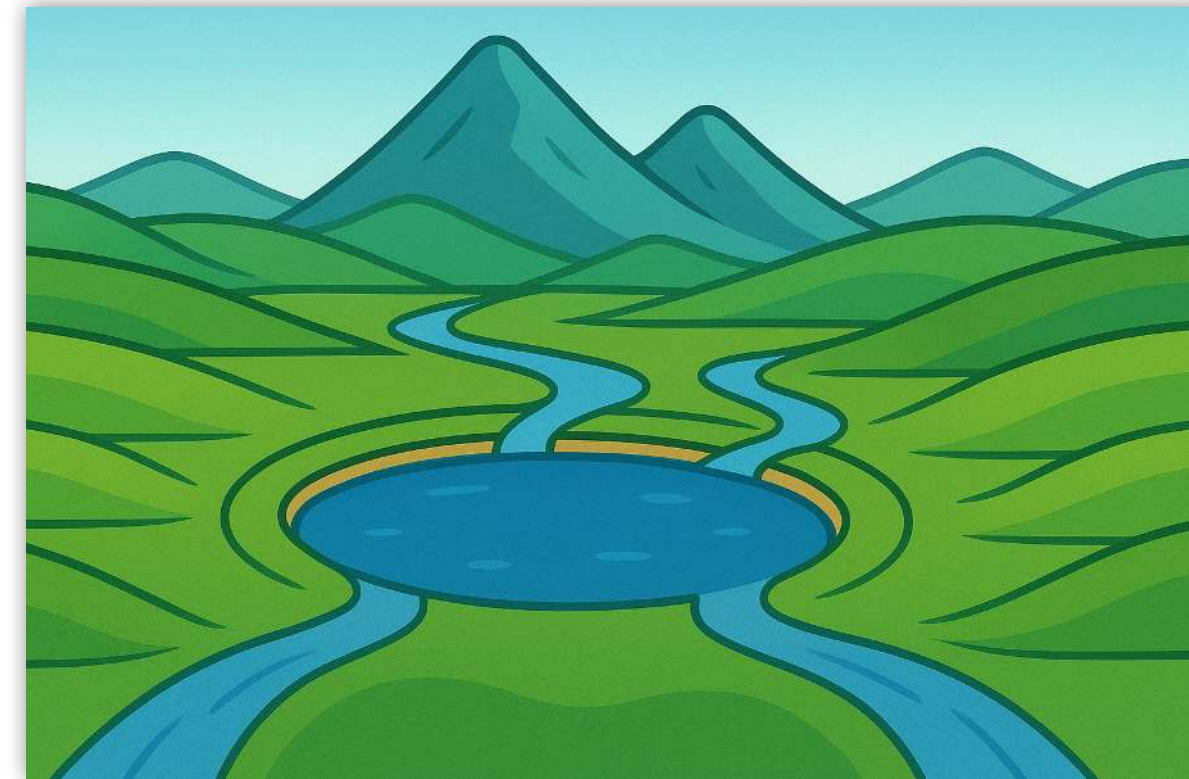
Database



Binary files

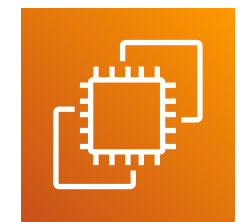


Database



Data lake

Disaggregated storage (2005–)



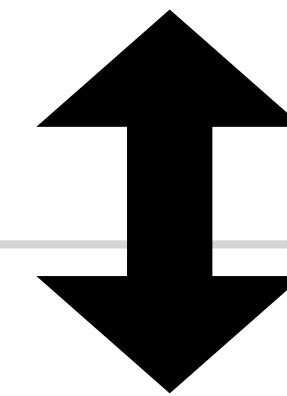
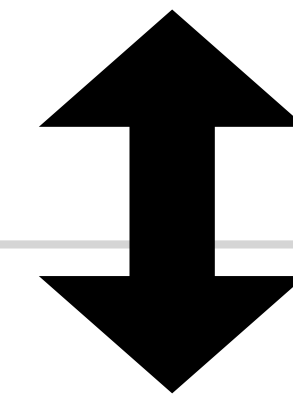
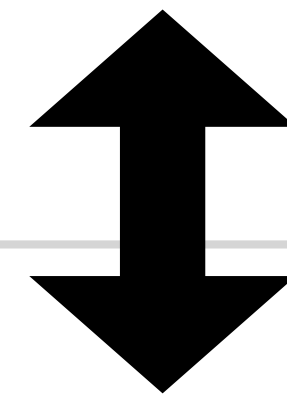
compute layer

compute node

compute node

...

compute node

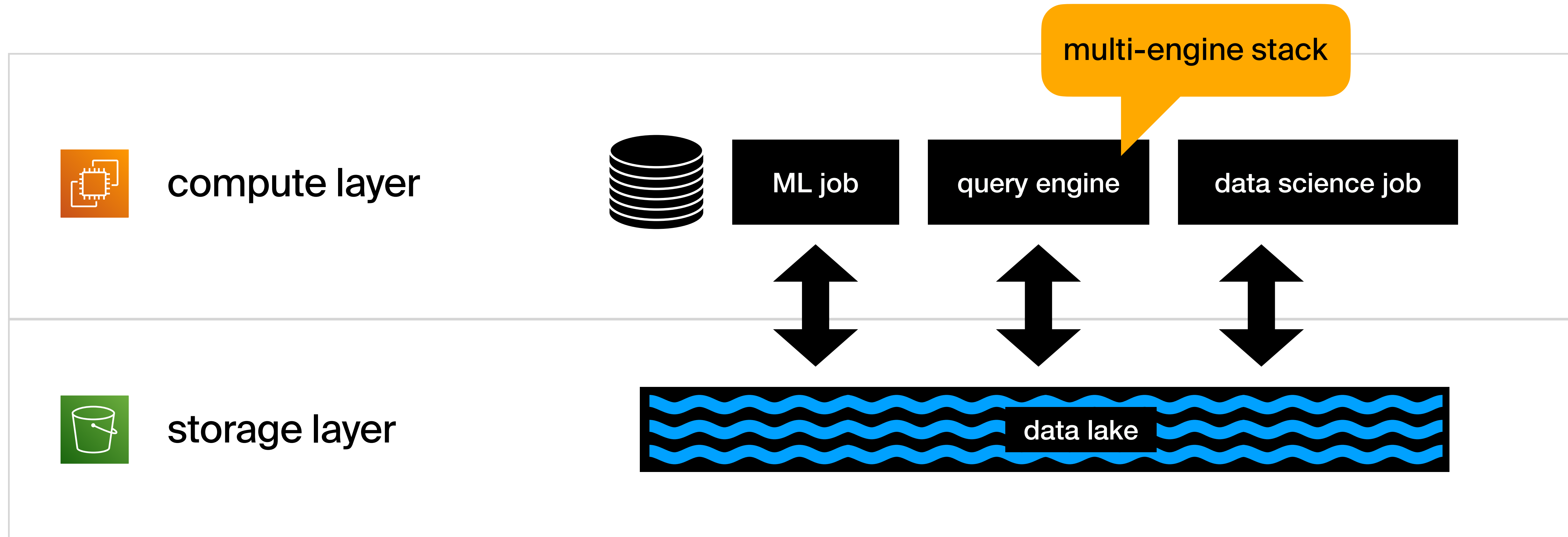


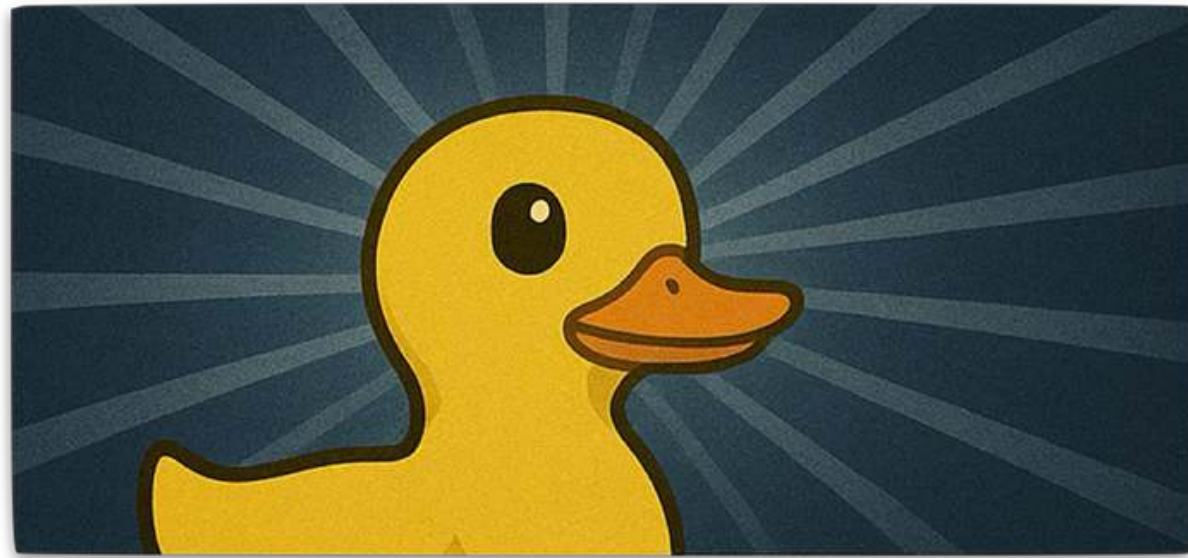
storage layer

object storage

immutable files

Data lake architecture (2015–)

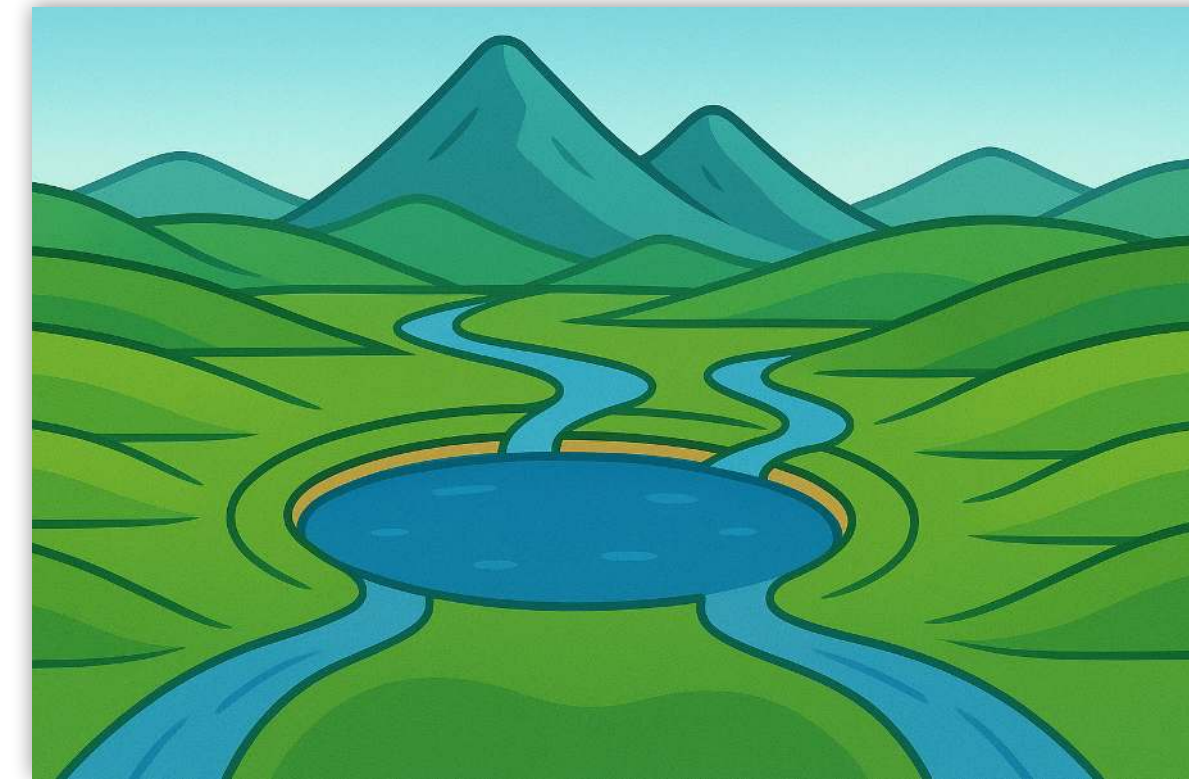
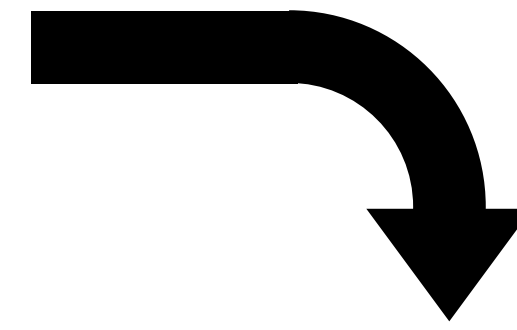




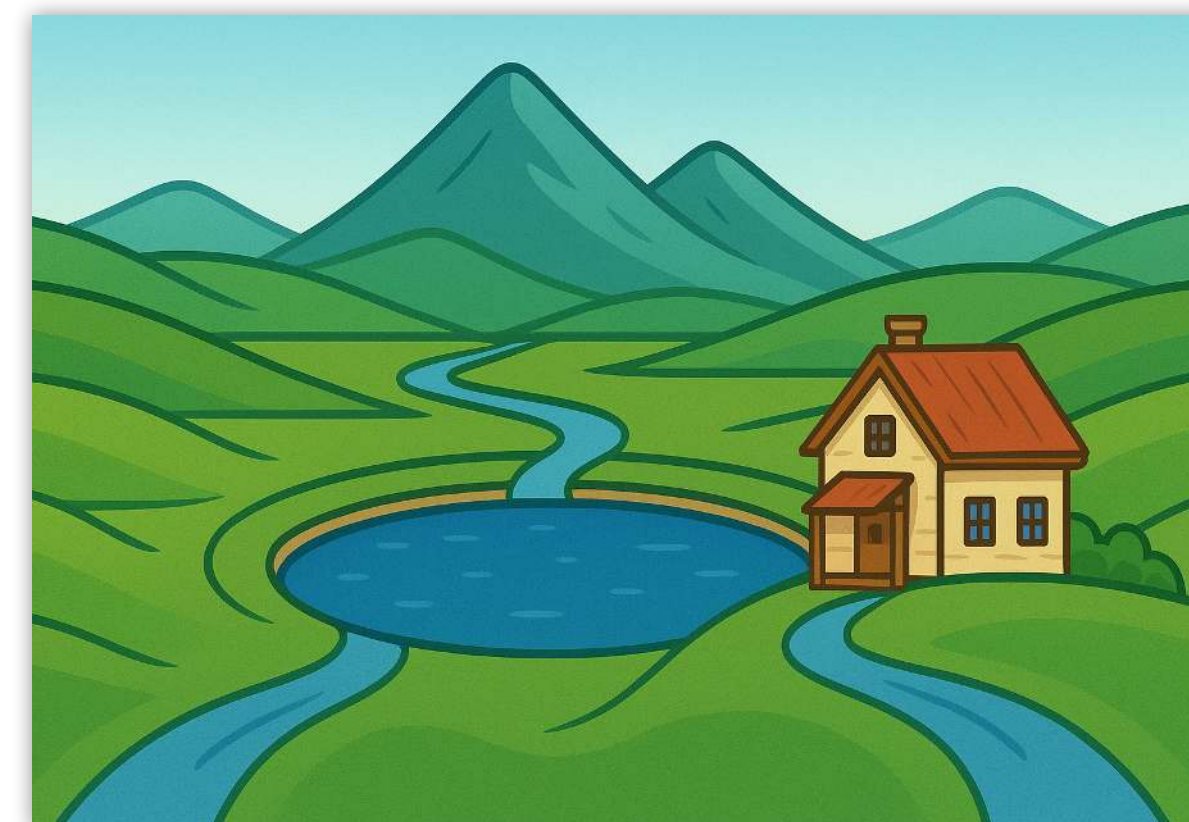
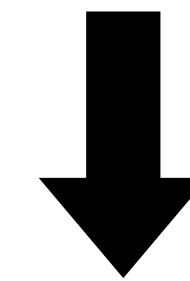
Binary files



Database

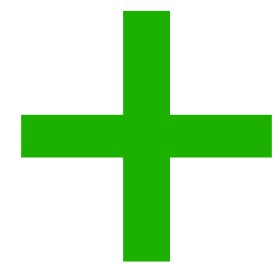


Data lake



Lakehouse

Data lake file formats



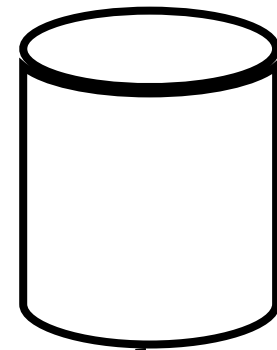
Transactional updates
Time travel queries
Schema evolution



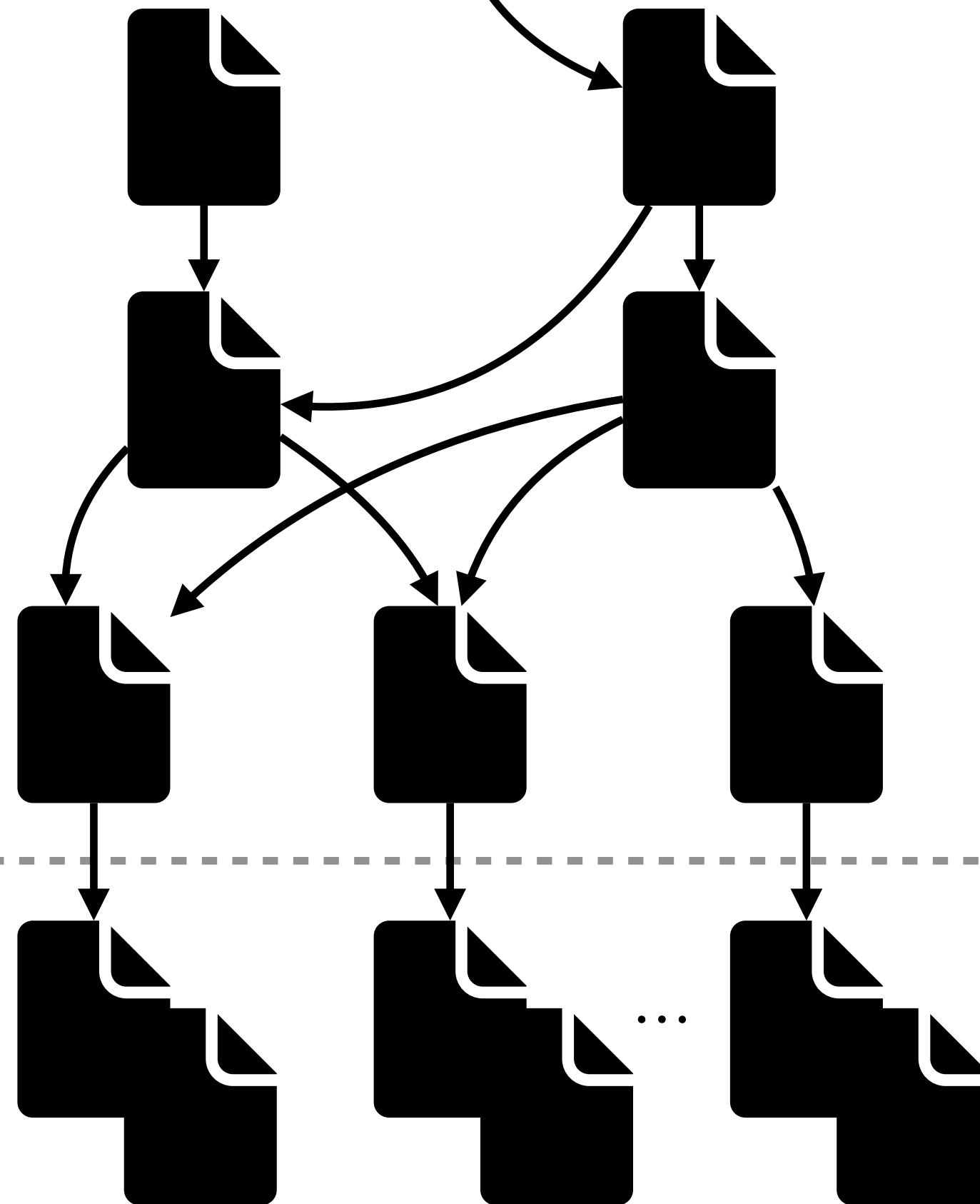
Performance issues
“Small files” problem
Limited to a single table



catalog layer
(database)



metadata layer
(json + avro)



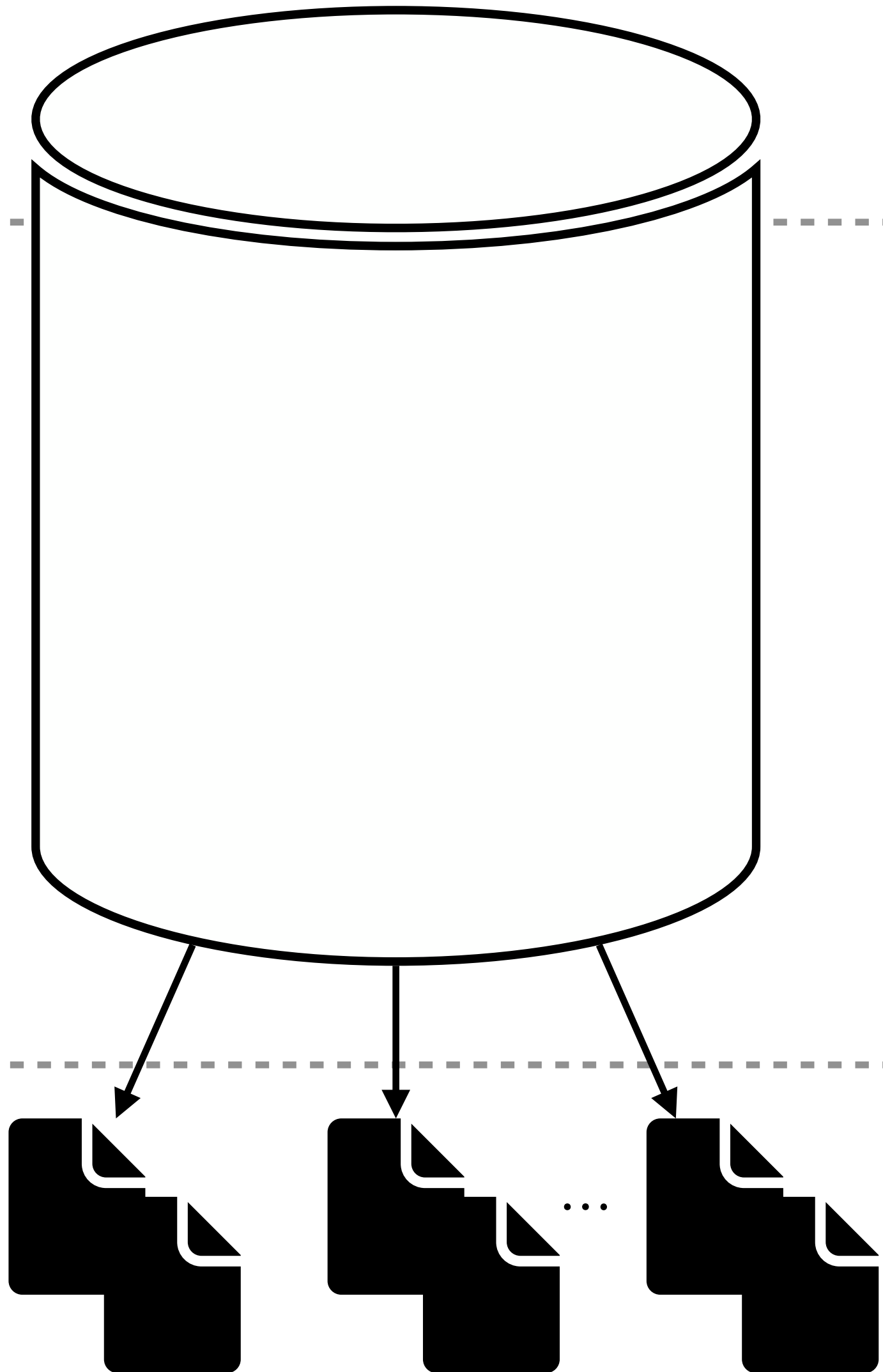
data layer
(parquet)



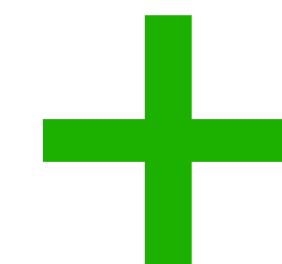
catalog layer
(database)

metadata layer
(database)

data layer
(parquet)



Transactional updates
Time travel queries
Schema evolution



Lightweight snapshots
Multi-table ACID
High performance
Data inlining

Data lake support in DuckDB





DuckLake



Quick setup

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

```
CREATE TABLE services AS  
  FROM 'trains-2025-02-feb.csv';
```




Adjust the train's number!

```
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



Use time travel

```
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



database

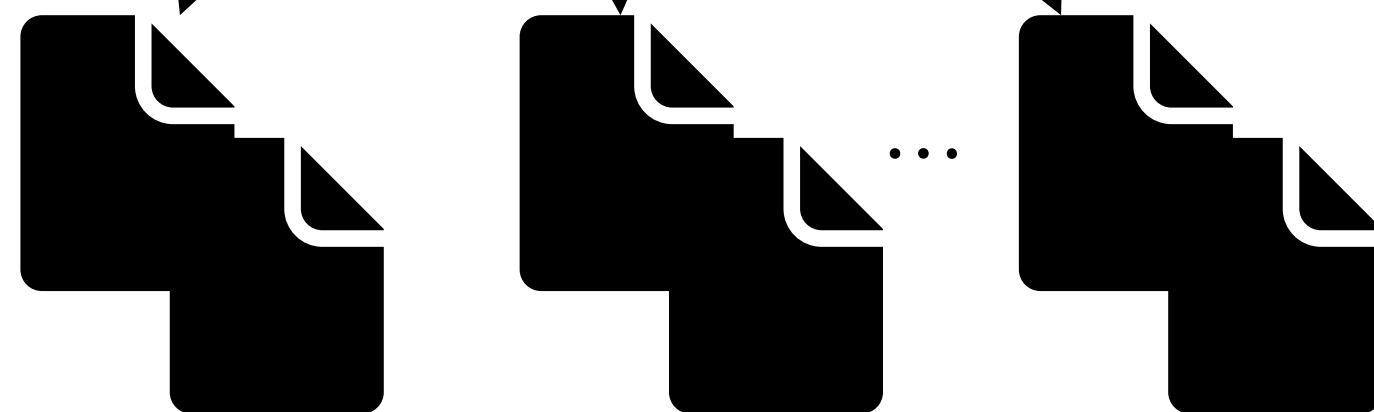


DuckLake client 1

DuckLake client 2

DuckLake client 3

object storage
(e.g., AWS S3)



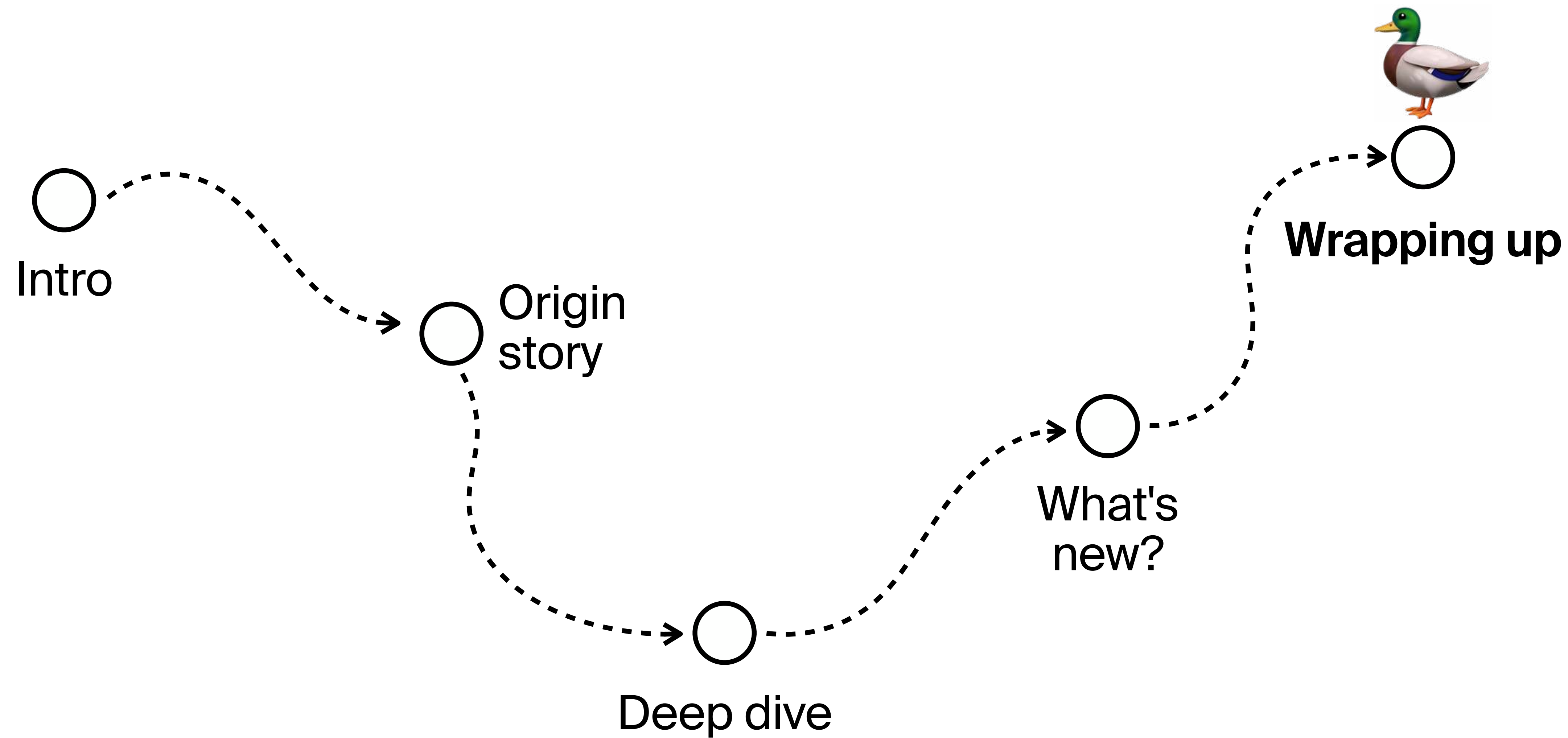


New kind of data architectures

No huge DuckDB file

Deletes can be faster

DuckLake 0.2 can directly import Parquet files

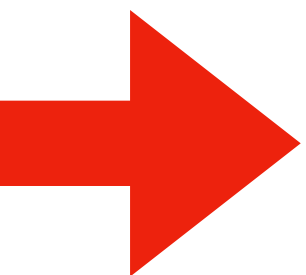




Open-source funding

Exotic platforms

LLMs are everywhere



Minor fixes #15715

Edit <> Code

Merged Mytherin merged 3 commits into `duckdb:main` from `Mytherin:secret` on Jan 14

Conversation 0 Commits 3 Checks 47 Files changed 3 +58 -23

Changes from all commits File filter Conversations 0 / 3 files viewed Review in codespace Review changes

Filter changed files

- src/include/duckdb/main
 - extension_entries.hpp
- tools/shell
 - include
 - shell_state.hpp
 - shell.cpp

src/include/duckdb/main/extension_entries.hpp

tools/shell/include/shell_state.hpp

```
@@ -119,6 +119,8 @@ struct ShellState {
    char thousand_separator = '\\';
    //! When to use formatting of large numbers (in DuckBox mode)
    LargeNumberRendering large_number_rendering =
        LargeNumberRendering::DEFAULT;
    +    //! The command to execute when '-ui' is passed in
    +    string ui_command = "CALL start_ui()";
    public:
```

