Beyond Pandas: Polars and DuckDB for Data Processing at Scale

Prof. Gregory Mermoud, PhD

Associate Professor – HES-SO Valais/Wallis

About me

Prof. Grégory Mermoud

Robust AI for Systems and Environments (RAISE) Lab

- Pioneered AI/ML at Cisco with 4 products shipped to 5000+ enterprise customers.
- Authored 200 granted patents (90+ pending).
- Hired 50+ engineers and built 3 engineering teams from the ground up.

Cisco AI Platform

My topics of interest: Robust and Explainable Machine Learning, Domain-informed AI, Neuromorphic Computing

Who’s the boss?

Suspect 1: Mr Excel

- Everyone knows him.
- Guilty of many things:
  - Maximum 2^{20} = 1’048’576 rows in a spreadsheet.
  - Translate functions depending on system language: `=PLANCHER(A4)` instead of `=FLOOR(A4)`.
  - Difficult to troubleshoot, impossible to review.
  - Shall I continue?
- Surely, he isn’t our guy…

Suspect 2: Mrs Pandas

- Got there first, took over most of the territory.
- Guilty of:
  - Being implemented in Python.
  - Allowing mutations.
  - Making a mess of indices (.loc vs .iloc).
- Not there yet…

Suspect 3: Mr Polars

- The new guy on the block.
- Guilty of:
  - Being written in Rust.
  - Being lazy at times (which is good!).
  - Using a columnar layout in memory.
  - Natively multi-threaded.
  - Supporting out-of-core computation (with caveats).
- We're getting there, but…

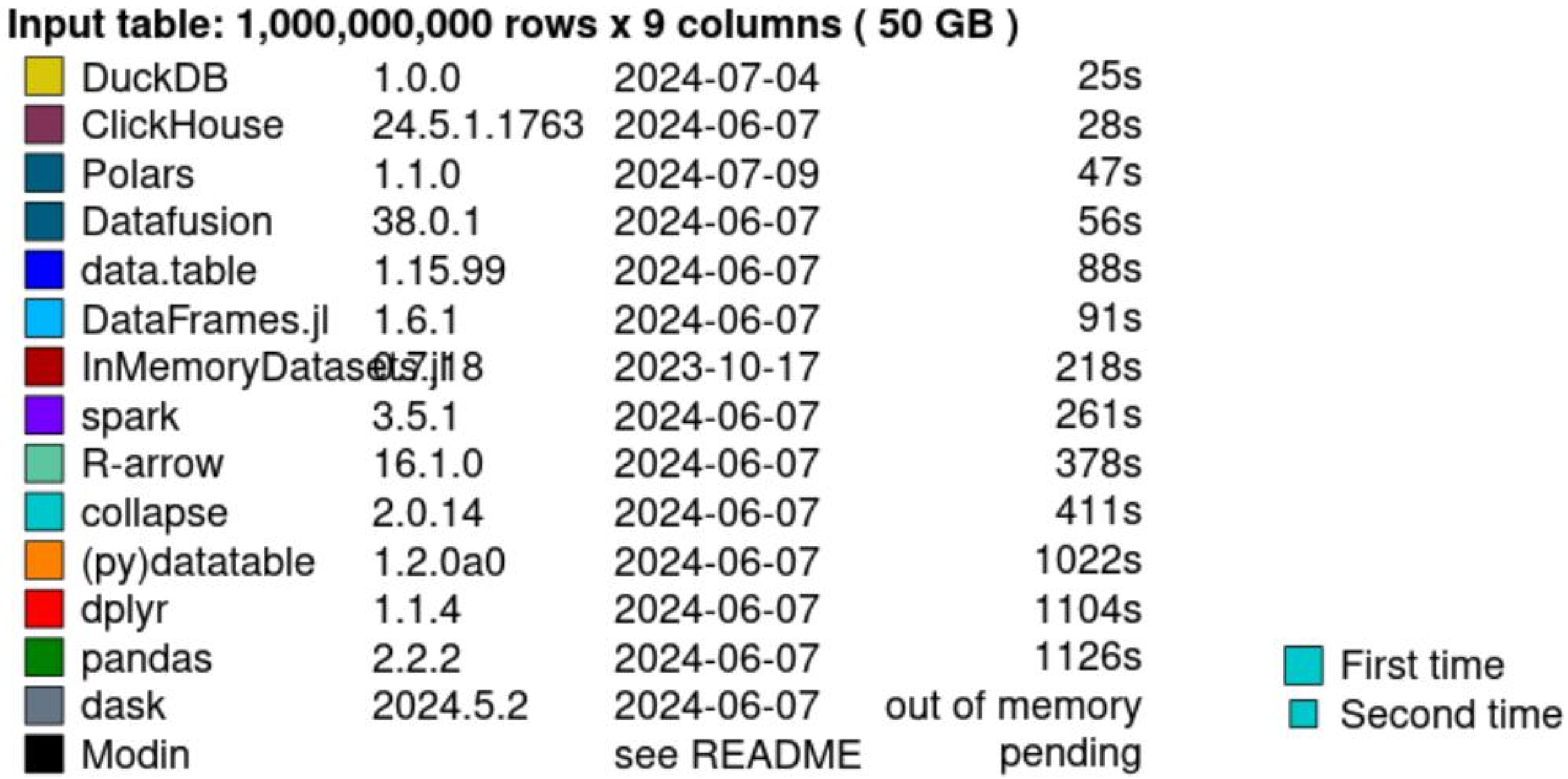
Suspect 4: Dr Duck

- It doesn’t look like much.
- But:
  - Written in C++.
  - Full-featured OLAP database that can act as a dataframe library.
  - Supports SQL and many languages (C, C++, Go, Julia).
  - Natively multi-threaded.
  - Very mature out-of-core computation.

Let’s see...

x45

x23



Input table: 1,000,000,000 rows x 9 columns (50 GB)

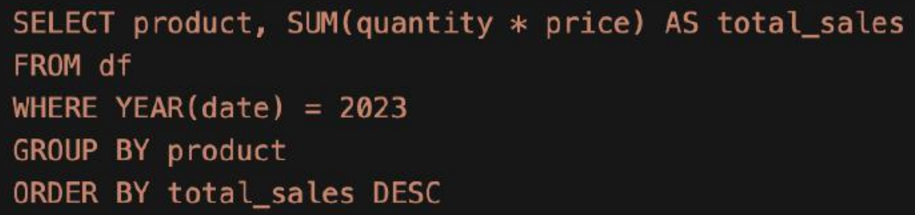
■ DuckDB	1.0.0	2024-07-04	25s
■ ClickHouse	24.5.1.1763	2024-06-07	28s
■ Polars	1.1.0	2024-07-09	47s
■ Datafusion	38.0.1	2024-06-07	56s
■ data.table	1.15.99	2024-06-07	88s
■ DataFrames.jl	1.6.1	2024-06-07	91s
■ InMemoryDatasets.jl	0.7.18	2023-10-17	218s
■ spark	3.5.1	2024-06-07	261s
■ R-arrow	16.1.0	2024-06-07	378s
■ collapse	2.0.14	2024-06-07	411s
■ (py)datatable	1.2.0a0	2024-06-07	1022s
■ dplyr	1.1.4	2024-06-07	1104s
■ pandas	2.2.2	2024-06-07	1126s
■ dask	2024.5.2	2024-06-07	out of memory
■ Modin		see README	pending

■ First time

■ Second time

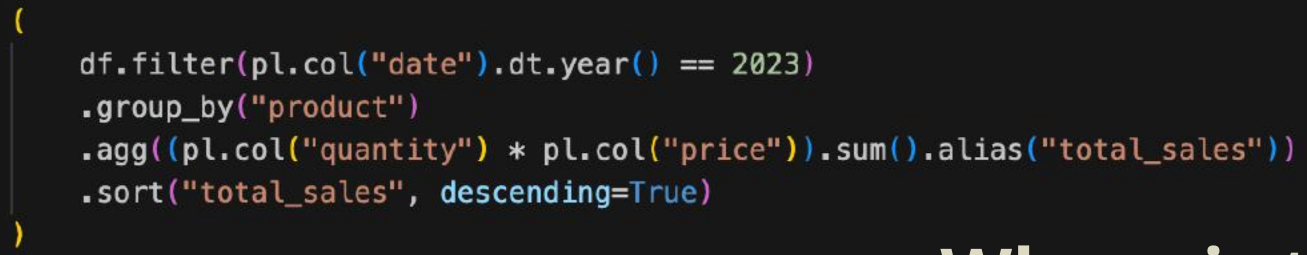
I came for the speed, but stayed for the syntax

DuckDB SQL



```sql
SELECT product, SUM(quantity * price) AS total_sales
FROM df
WHERE YEAR(date) = 2023
GROUP BY product
ORDER BY total_sales DESC
```

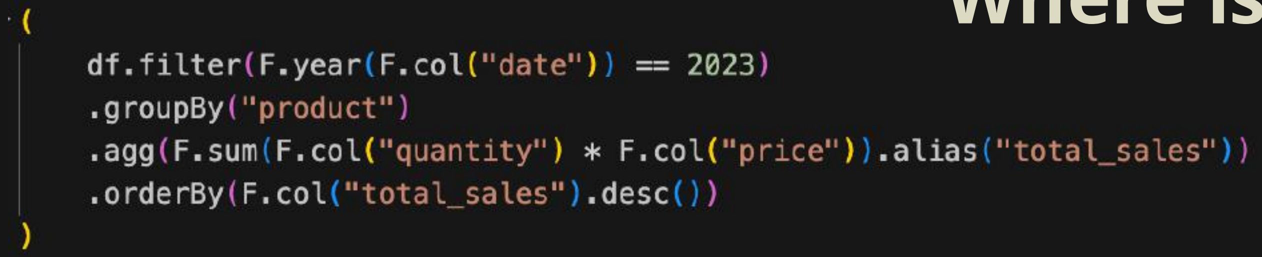
Polars Python API



```python
(
    df.filter(pl.col("date").dt.year() == 2023)
    .group_by("product")
    .agg((pl.col("quantity") * pl.col("price")).sum().alias("total_sales"))
    .sort("total_sales", descending=True)
)
```

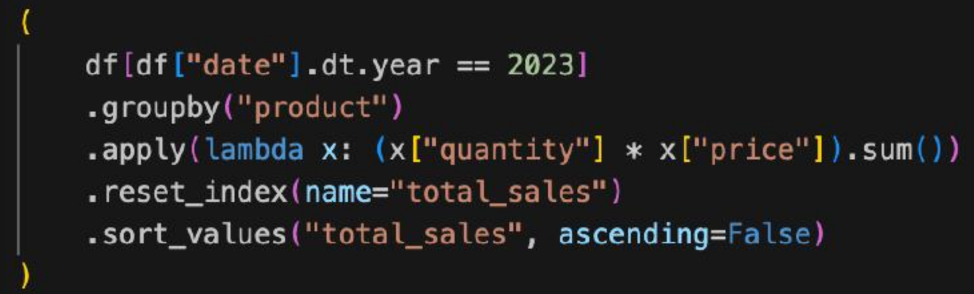
Where is the outlier?

Spark Dataframe API



```python
(
    df.filter(F.year(F.col("date")) == 2023)
    .groupBy("product")
    .agg(F.sum(F.col("quantity") * F.col("price")).alias("total_sales"))
    .orderBy(F.col("total_sales").desc())
)
```

Pandas API



```python
(
    df[df["date"].dt.year == 2023]
    .groupby("product")
    .apply(lambda x: (x["quantity"] * x["price"]).sum())
    .reset_index(name="total_sales")
    .sort_values("total_sales", ascending=False)
)
```

The importance of being lazy

- Polars, DuckDB and Spark have an amazing advantage over Pandas:

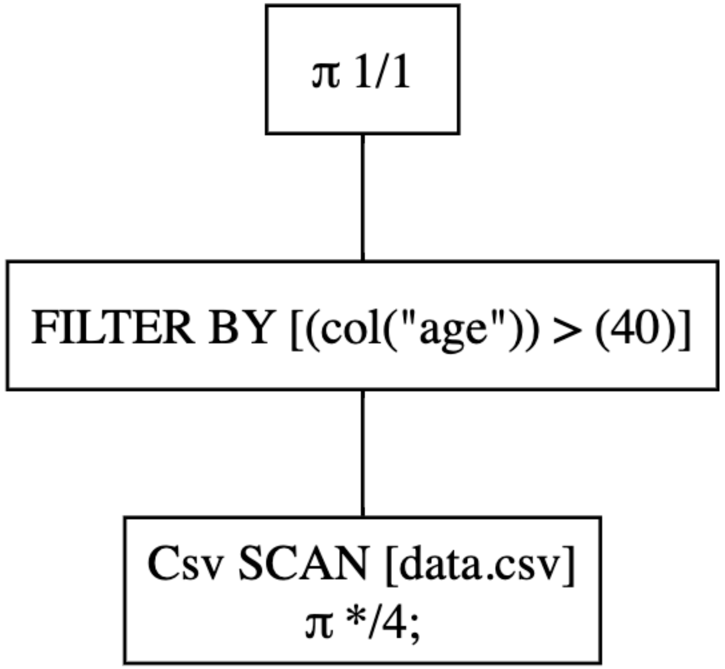
They can be lazy.

- Being lazy gives you the opportunity to apply optimizations, such as:
  - Predicate pushdown
  - Projection pushdown
  - Slice pushdown
  - Common subplan elimination
  - Expression simplification

An example: predicate pushdown

```
$ head -n5 data.csv && echo -n "Number of lines: " && wc -l < data.csv
age,net_worth,country,industry
21,939032.5995028166,FR,Finance
24,956118.193141526,UK,Oil & Gas
24,186605.8369502175,CA,Oil & Gas
60,37244.9790113912,UK,Finance
Number of lines: 1000001
```

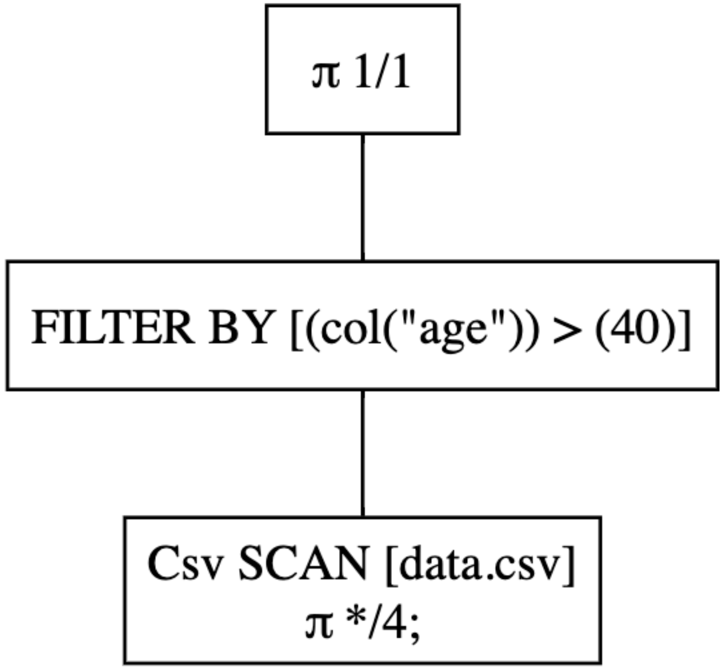
```
df = (
  pl.scan_csv("data.csv")
  .filter(pl.col("age") > 40)
  .select(pl.col("net_worth").mean())
)
```



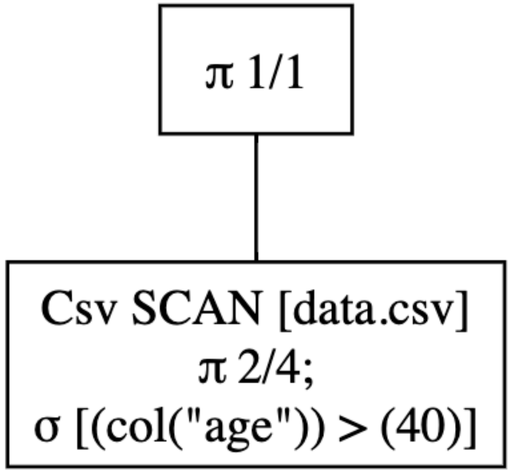
Unoptimized graph

An example: predicate pushdown

```
df = (
  pl.scan_csv("data.csv")
  .filter(pl.col("age") > 40)
  .select(pl.col("net_worth").mean())
)
```



Unoptimized graph



Optimized graph

Scaling things up...

We now consider a dataset of hourly measurements from 26 Meteosuisse stations in Valais for the last 15 years (3 million rows and 26 columns, 183 MB) and the following query:

Polars

```
df.group_by("stn").agg(
    pl.col("wind_dir_avg")
    .filter(
        pl.col("temp_avg") >
          pl.col("temp_avg").mean() +
          2.0 * pl.col("temp_avg").std()
    ).mean(),
    pl.col("wind_dir_avg").mean()
).sort(
    by="stn"
)
```

pandas

```
CPU times: user 10.9 s, sys: 1.62 s, total: 12.5 s
Wall time: 3.6 s
```

DuckDB

```
CPU times: user 3.31 s, sys: 541 ms, total: 3.85 s
Wall time: 2.04 s
```

Polars

```
CPU times: user 2.32 s, sys: 1.04 s, total: 3.36 s
Wall time: 1.13 s
```

Out of core computation

- NYC taxi dataset: 20 years of taxi pickups/drop-offs in New York City
  - 1.54 billion records, 48 GB of **compressed** Parquet on disk

DuckDB

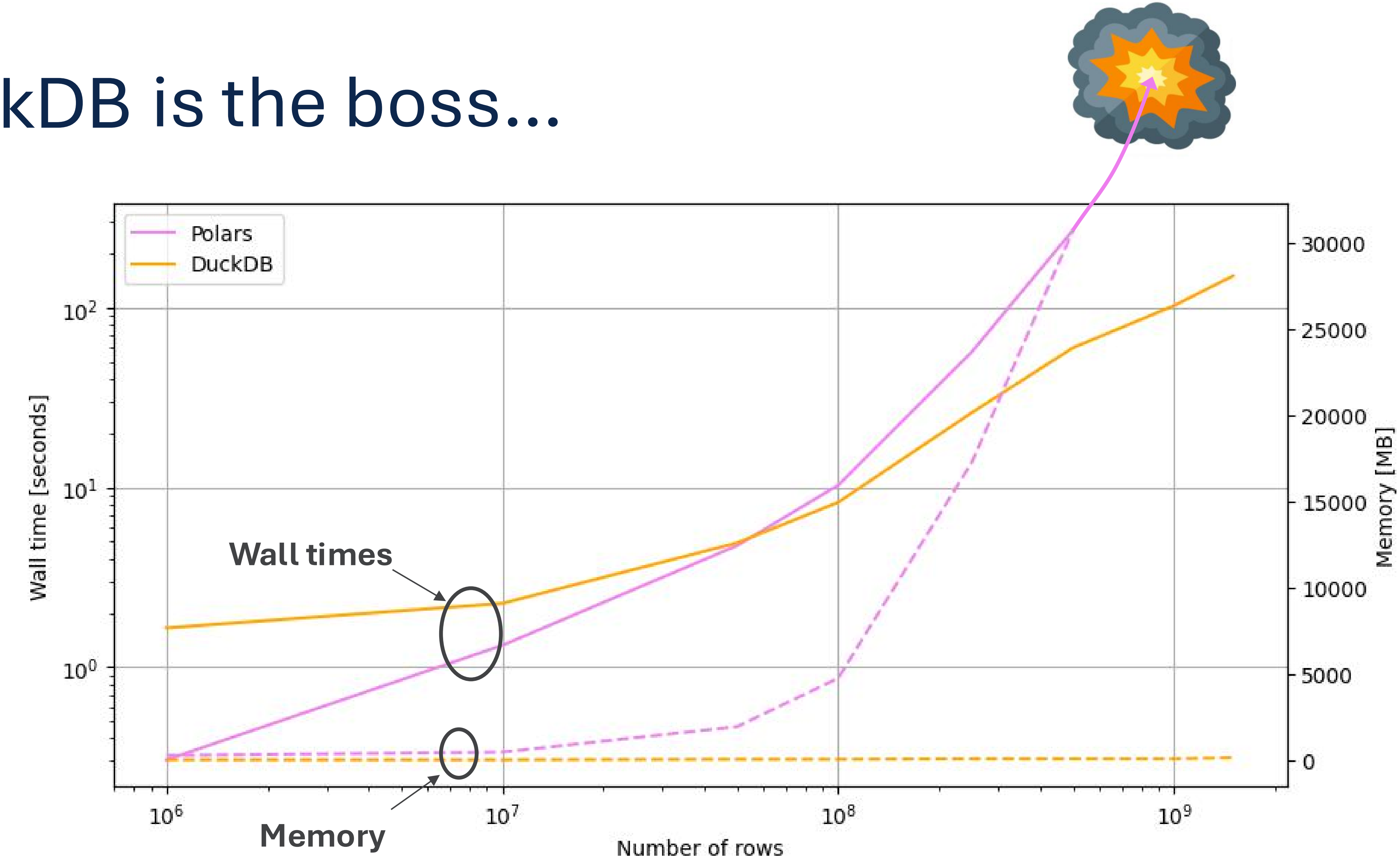
```
SELECT
 vendor_id,
 AVG(tip_amount / fare_amount) AS average_tip_rate,
 COUNT(
   CASE
     WHEN tip_amount > 0 THEN 1
   END
 ) * 1.0 / COUNT(*) AS fraction_tipped_trips,
FROM
 read_parquet('/home/shared/nyc-taxi.parquet')
WHERE
 fare_amount > 0
GROUP BY
 vendor_id
ORDER BY
 average_tip_rate DESC;
```

Polars

```
pl.scan_parquet("/home/shared/nyc-taxi.parquet")
.filter(pl.col("fare_amount") > 0)
.group_by("vendor_id")
.agg([
  (pl.when(pl.col("tip_amount") > 0).then(1).sum() /
     pl.len()).alias("fraction_tipped_trips"),
  (pl.col("tip_amount") / pl.col("fare_amount"))
  .mean()
  .alias("average_tip_rate"),
])
.sort("average_tip_rate", descending=True)
```

pandas is not even allowed to compete...

DuckDB is the boss…



Wall times

Memory

DuckDB’s streaming engine is **significantly more mature** than Polars’.

But there is very active development of Polars!

Conclusion

- Use the right tool for the problem at hand!
- Performance matters.
- Syntax and maintainability matter more.
- For out of core computation, DuckDB is still the boss… but Polars is improving fast!

Thanks for your attention!

Hes·so// VALAIS WALLIS

Let’s connect!