May 18, 2024

Intelligent Tomato group Data Analysis

Name any dataframe library you have used?


pandas

The largest number of rows

you have ever processed
using Pandas library in Python?

How about...
one billion rows?

Normal Pandas read_csv

```python
import pandas as pd

# Read the file into a DataFrame
df = pd.read_csv("measurements.txt", names=['station', 'measure'])

# Group by station and calculate min, max, and mean
result = df.groupby('station')['measure'].agg(['min', 'max', 'mean'])

print(result)

```

⊗

```
The Kernel crashed while executing code in the current cell or a previous cell.
Please review the code in the cell(s) to identify a possible cause of the failure.
Click here for more info.
View Jupyter log for further details.
```

The kernel crashes because
the file size exceeds the available RAM!

Pandas read_csv in chunks

```python
def process_chunk(chunk):
    # Aggregates the data within the chunk using Pandas
    aggregated = chunk.groupby('station')['measure'].agg(['min', 'max', 'mean']).reset_index()
    return aggregated

def create_df_with_pandas(filename, total_linhas, chunksize=chunksize):
    total_chunks = total_linhas // chunksize + (1 if total_linhas % chunksize else 0)
    results = []

    with pd.read_csv(filename, sep=';', header=None,
                     names=['station', 'measure'],
                     chunksize=chunksize) as reader:
        # Wrapping the iterator with tqdm to visualize progress
        with Pool(CONCURRENCY) as pool:
            for chunk in tqdm(reader, total=total_chunks, desc="Processing"):
                # Processes each chunk in parallel
                result = pool.apply_async(process_chunk, (chunk,))
                results.append(result)

            results = [result.get() for result in results]

    final_df = pd.concat(results, ignore_index=True)

    final_aggregated_df = final_df.groupby('station').agg({
        'min': 'min',
        'max': 'max',
        'mean': 'mean'
    }).reset_index().sort_values('station')

    return final_aggregated_df
```

Pandas allows us to read a large DataFrame in chunks and process them in parallel.

GUESS

Does Pandas chunking helps?

6 hours pass

and our Pandas still keep working so hard....

```
start_time = time.time()
df = create_df_with_pandas(filename, total_linhas, chunksize)
took = time.time() - start_time

print(df.head())
print(f"Pandas took: {took:.2f} sec")
```

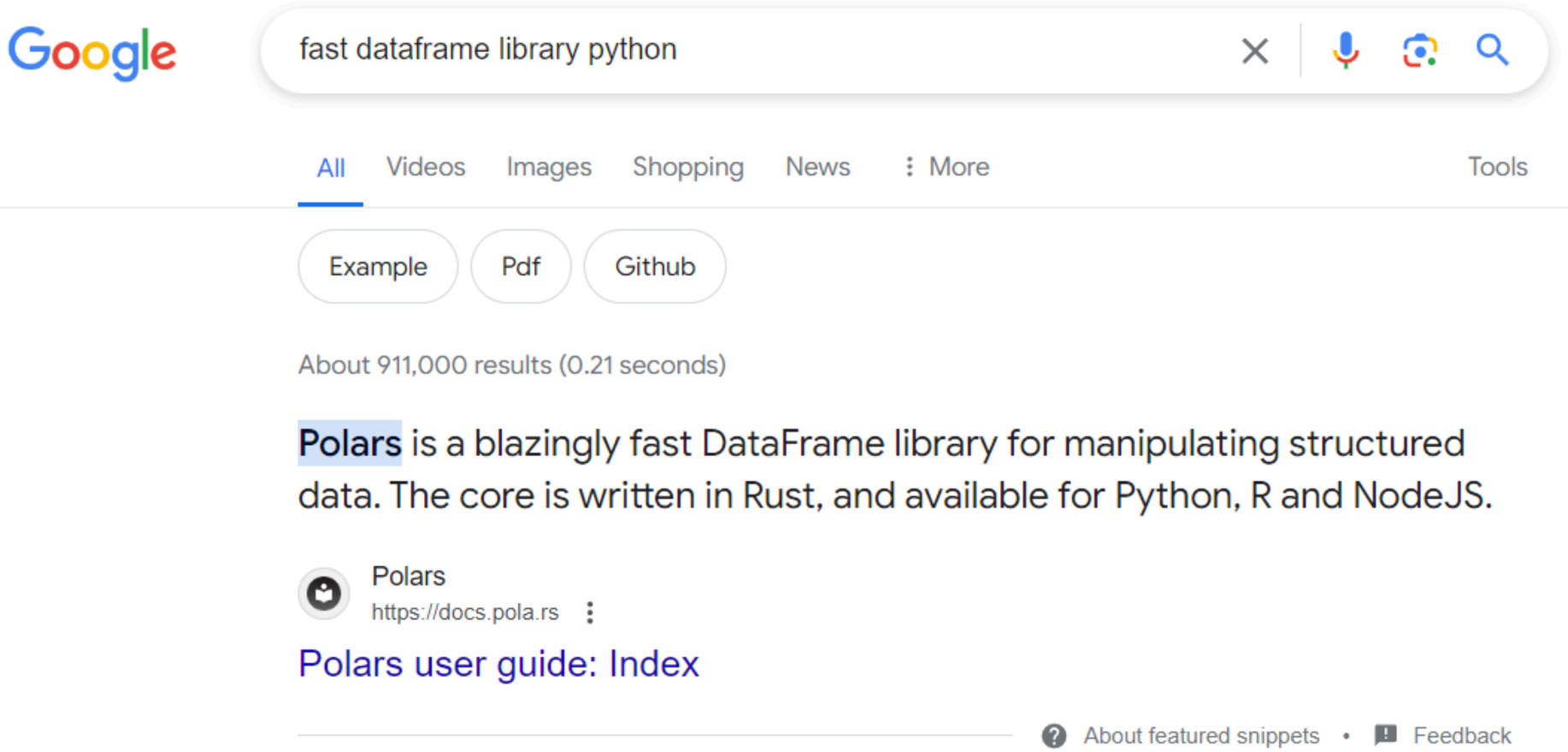
367m 58.6s

```
Processando: 100%|██████████| 10/10 [04:40<00:00, 28.02s/it]
```

Don’t be frustrated…

Let’s find a solution!

fast dataframe library python


Google
fast dataframe library python
All
Videos
Images
Shopping
News
More
Tools
Example
Pdf
Github
About 911,000 results (0.21 seconds)
Polars is a blazingly fast DataFrame library for manipulating structured data. The core is written in Rust, and available for Python, R and NodeJS.
Polars
https://docs.pola.rs
Polars user guide: Index
About featured snippets
Feedback


No way you can do it?!!!
Look at me!

Demonstrating...

~30 seconds

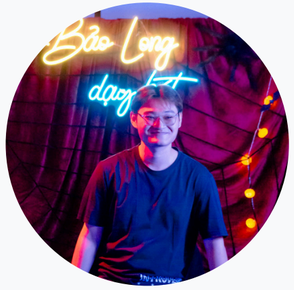
to find min, max, mean of

1 billion rows

A Lightning-Fast DataFrame Library

TOMATO TEAM MEMBERS

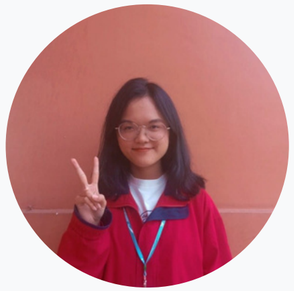
21120275
Huỳnh Cao Khôi



21120290
Hoàng Trung Nam

21120302
Huỳnh Trí Nhân

21120308
Phạm Lê Tú Nhi



21120533
Lê Thị Minh Phương

21120593
Võ Hoàng Hoa Viên

CONTENTS

POLARS: A LIGHTNING-FAST DATAFRAME LIBRARY

01 STORY

2020 - Coronavirus Pandamic

Vink's hobby project in **RUST** to learn:

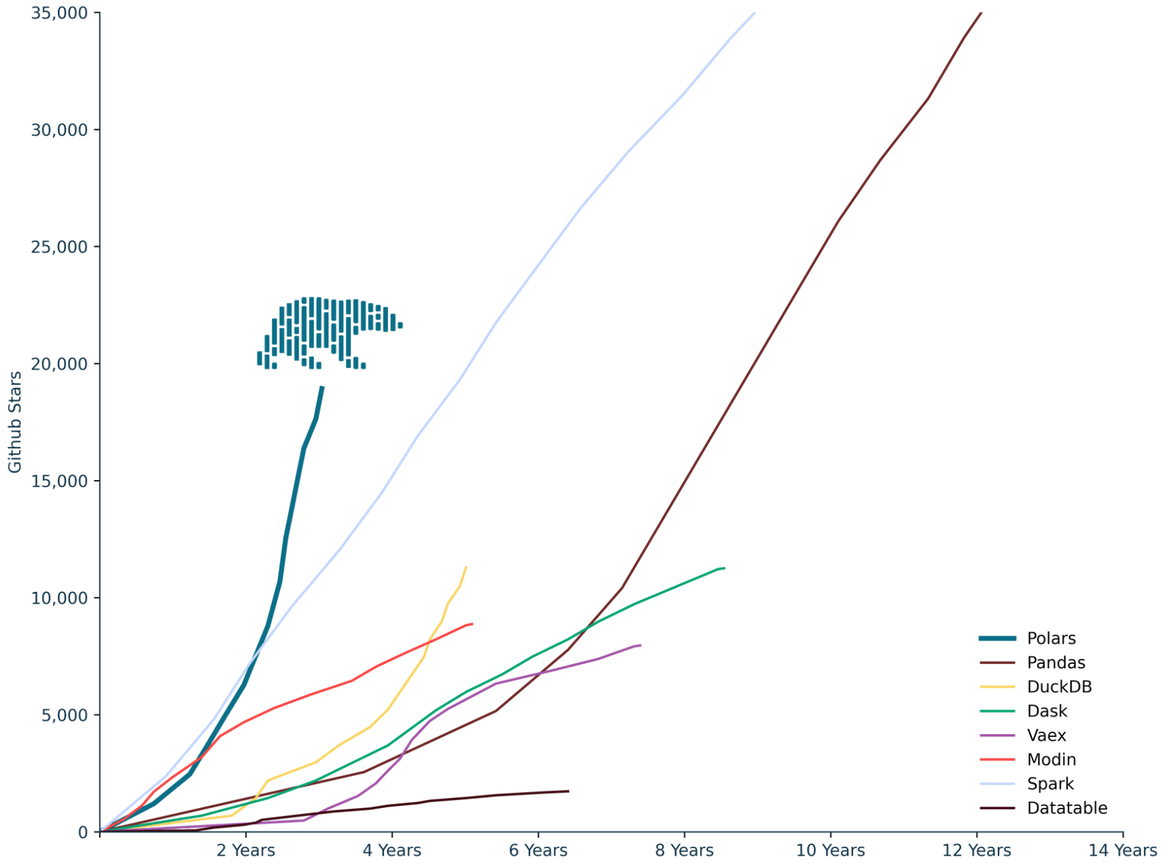
- Query engines
- Apache arrow
- Rust

First commit: 23 June 2020

Richie Vink

However...
Polars grew
beyond his expectation!!!

Benchmark
Graph



SUPPORTED LANGUAGES



officially


on the way

02 KEY DESIGNS

To understand Polars and its key design decisions,
we start with the familiar dataframe library

Pandas was build in April, 2008
by Wes McKinney

In 2017, Wes McKinney published a post named “Apache Arrow and the 10 Things I Hate About Pandas”.

> *I didn’t know much about software engineering or even how to use Python’s scientific computing stack well back then. My code was ugly and slow.*

Wes McKinney,
Apache Arrow and the 10 things i hate about pandas

The 10 things I hate about Pandas

by Wes McKinney

Internals too far from “the metal”	Complex groupby operations awkward and slow
No support for memory-mapped datasets	Appending data to a DataFrame tedious and very costly
Poor performance in database and file ingest/export	Limited, non-extensible type metadata
Warty missing data support	Eager evaluation model, no query planning
Weak support for categorical data	“Slow”, limited multicore algorithms

Source

> *I strongly feel that **Arrow** is a key technology for the next generation of data science tools.*
>
> *[...] building a faster, cleaner core pandas implementation, which we may call **Pandas 2**.*

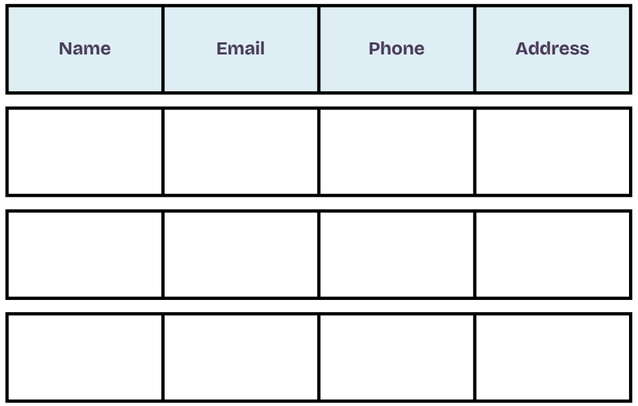
Wes McKinney,
Apache Arrow and the 10 things i hate about pandas

And **Apache Arrow**,
is actually the key technology
for both **Pandas 2** and **Polars**

How can APACHE ARROW optimize query engines?

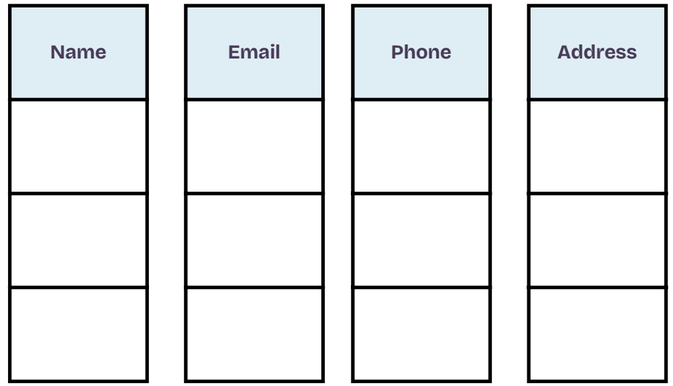
Apache Arrow
A language-independent columnar memory format

Row storage



Data is stored row by row

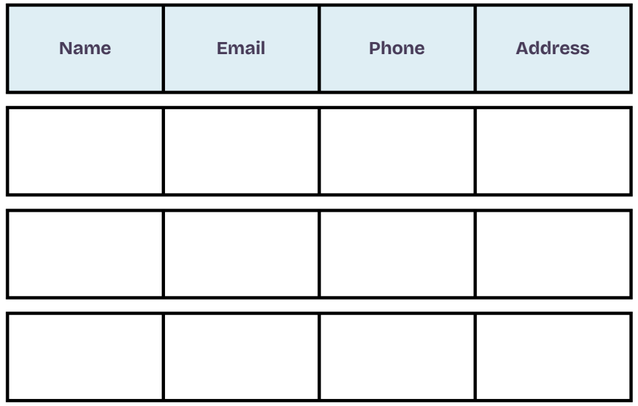
Columnar storage



Data is organized by columns

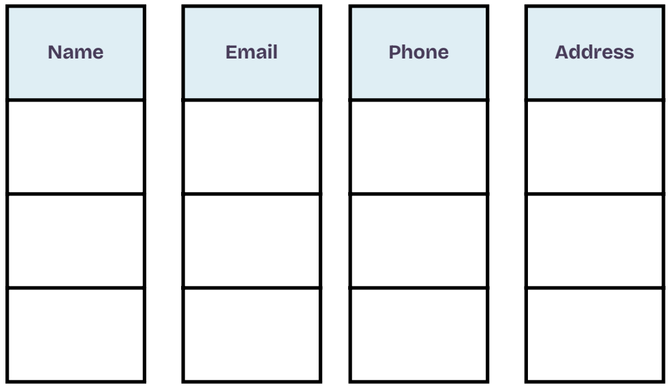
Apache Arrow
A language-independent columnar memory format

Row storage



Efficient for **transactional operations**
(selecting, inserting, updating, or deleting rows)

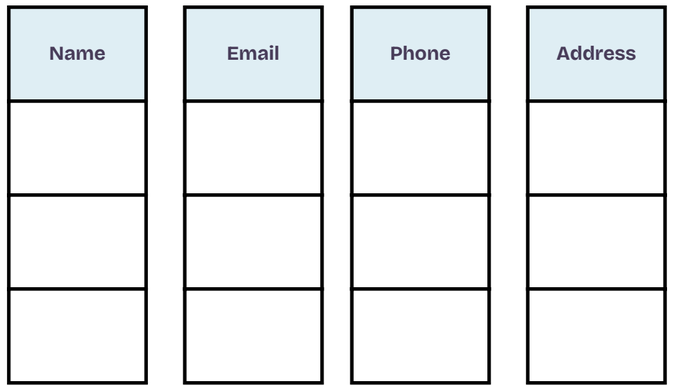
Columnar storage



Efficient for **analytical queries** that scan
large datasets but only access a few columns

Apache Arrow
A language-independent columnar memory format

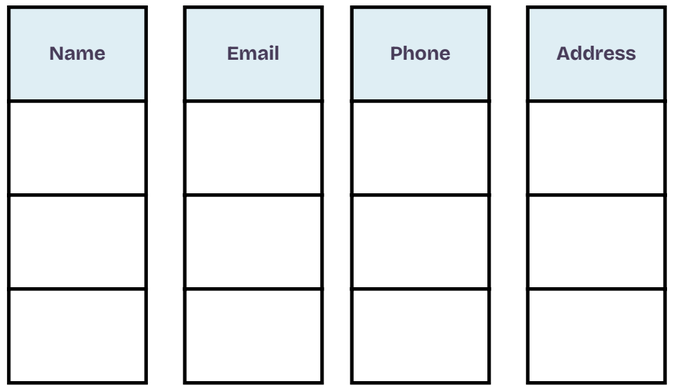
Columnar storage



Why is **columnar format** such powerful?

Apache Arrow
A language-independent columnar memory format

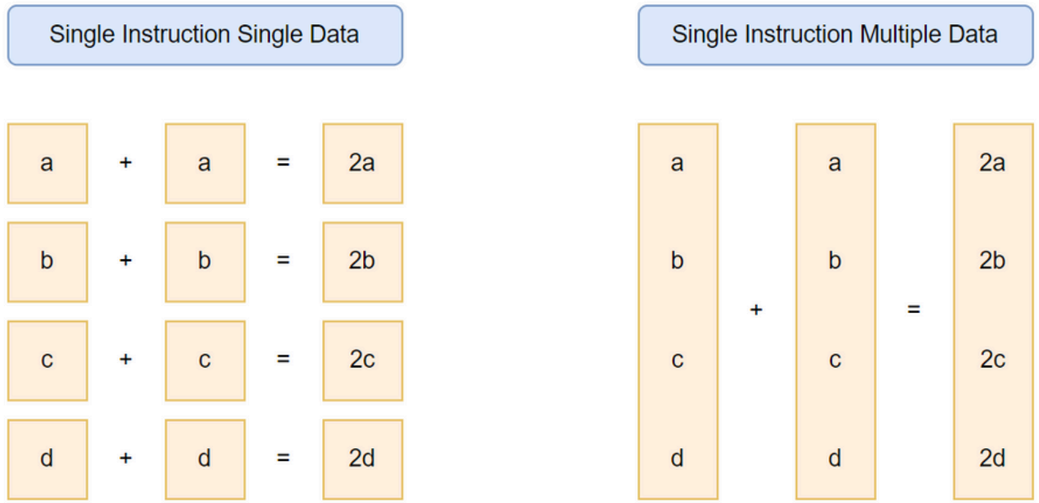
Columnar storage



Why is **columnar format** such powerful?

DATA PARALLELISM

Data parallelism



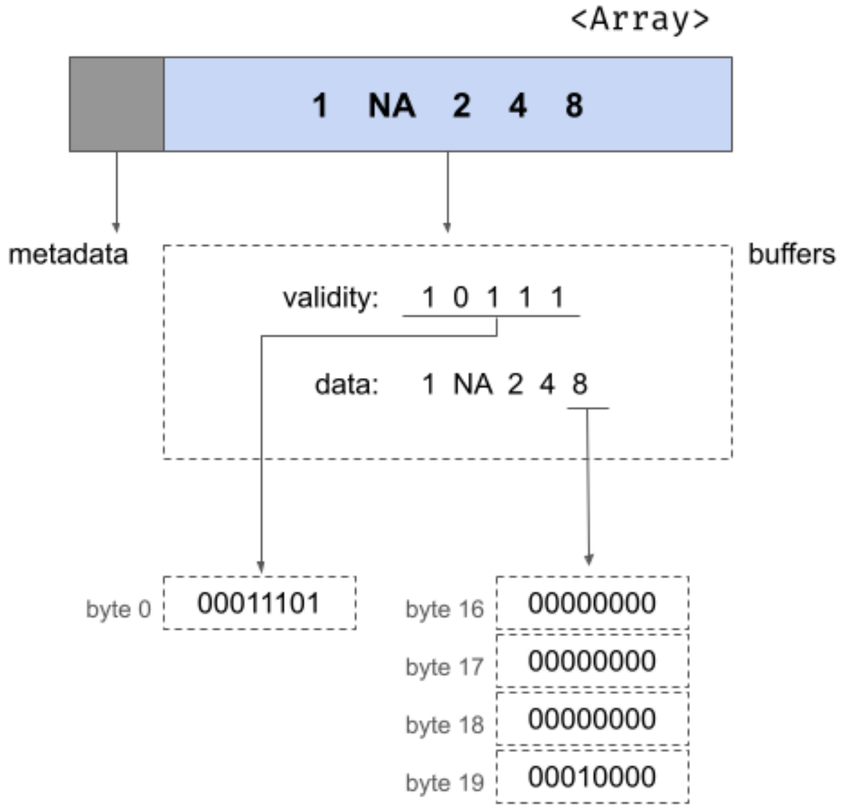
Perform a **Single Instruction** across **Multiple Data** at the same time (**SIMD**)

Pandas 2 and Polars levarage Arrow Columnar Format to solve some Pandas problems

Getting closer to the metal	**Better groupby operations**
No support for memory-mapped datasets	**DataFrame appending is more memory-efficient**
Poor performance in database and file ingest/export	Limited, non-extensible type metadata
Warty missing data support	Eager evaluation model, no query planning
Weak support for categorical data	“Slow”, limited multicore algorithms

Apache Arrow uses **Validity bitmap buffer**

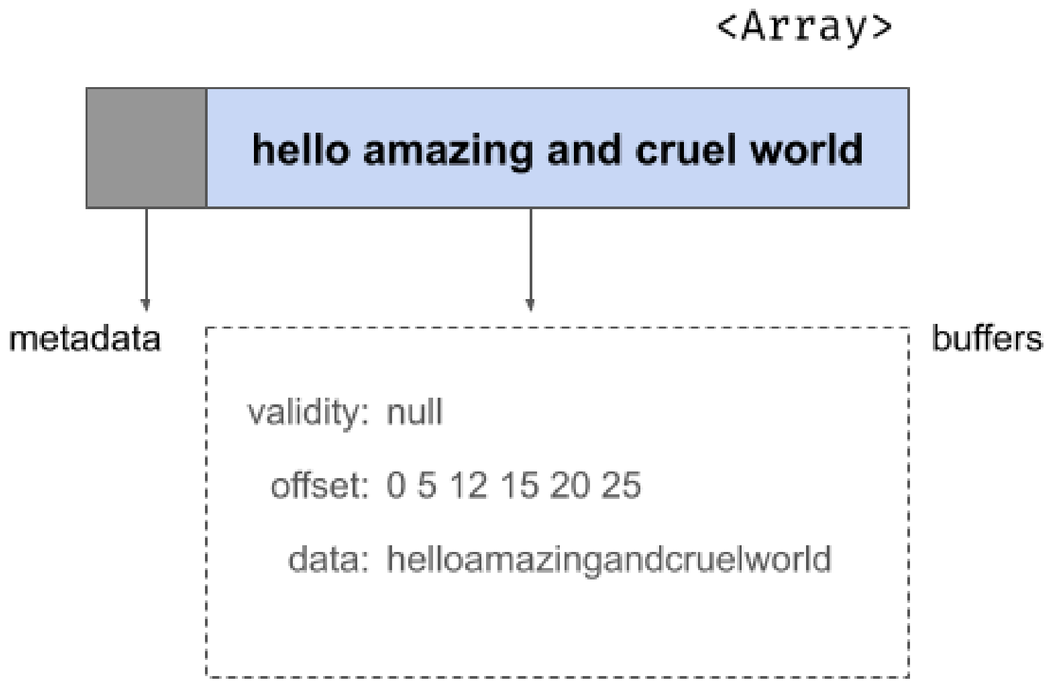
Apache Arrow uses Validity bitmap buffer



Pandas 2 and Polars levarage Arrow Validity bitmap buffer to solve some Pandas problems

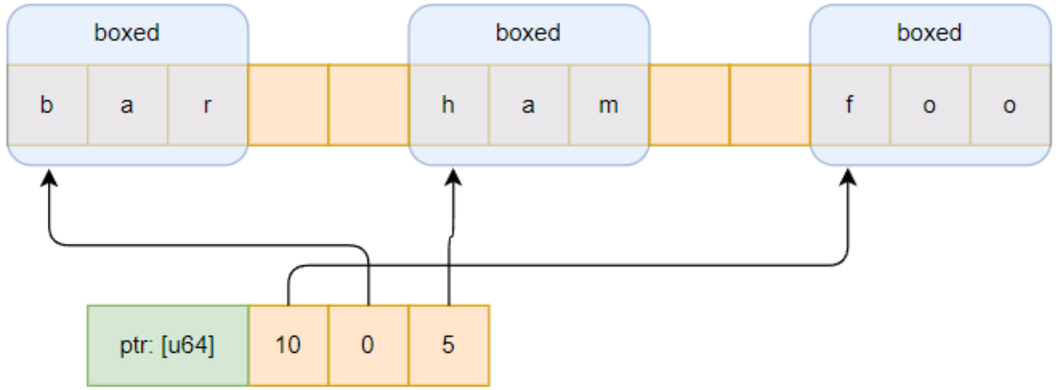
Getting closer to the metal	Better groupby operations
No support for memory-mapped datasets	DataFrame appending is more memory-efficient
Poor performance in database and file ingest/export	**Extensible type metadata**
Doing missing data right	**Eager evaluation model, no query planning**
Weak support for categorical data	**“Slow”, limited multicore algorithms**

Arrow string array is stored sequentially

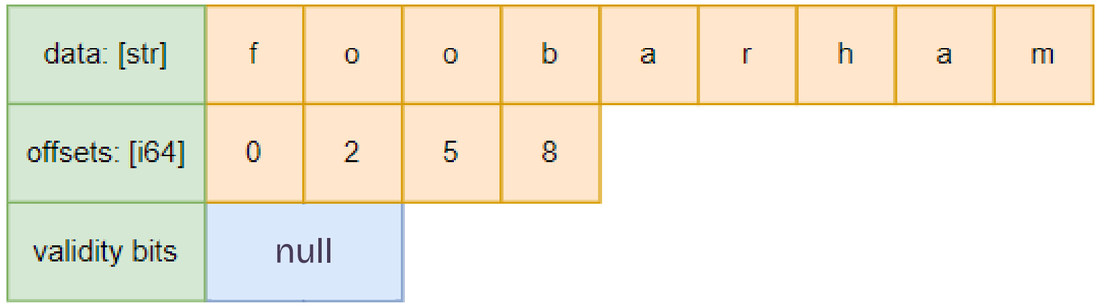


h	e	l	l	o	a	m	a	z	i	n	g	a	n	d	...
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...

Pandas string array
(Python object array)



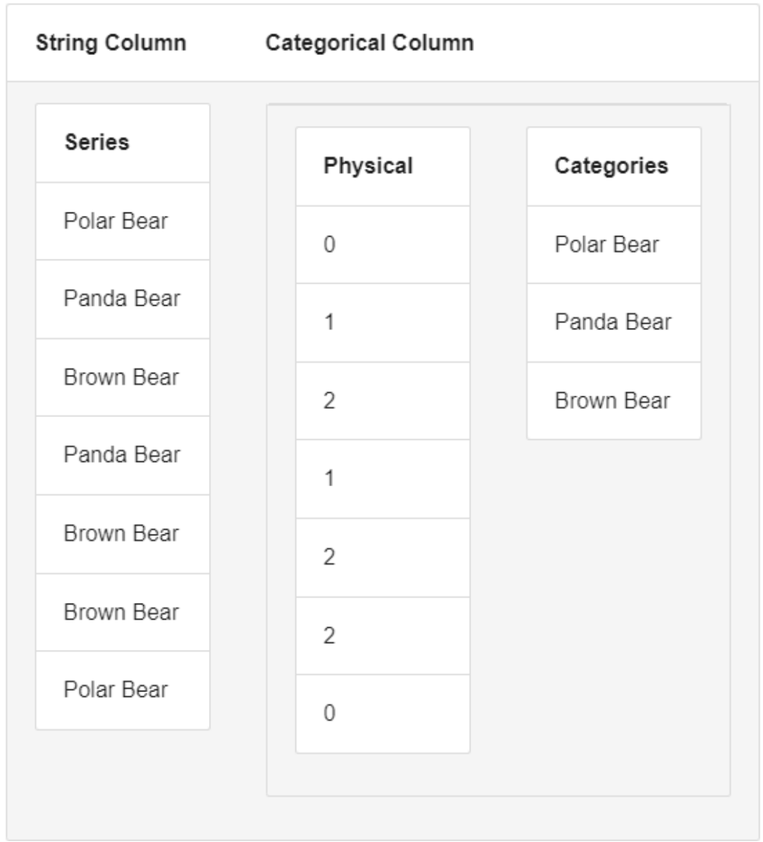
Polars string array
(Arrow LargeString array)



Pandas 2 and Polars levarage Arrow String to solve one more Pandas problem

Getting closer to the metal	Better groupby operations
No support for memory-mapped datasets	DataFrame appending is more memory-efficient
Poor performance in database and file ingest/export	Extensible type metadata
Doing missing data right	**Eager evaluation model, no query planning**
Weak support for categorical data	**“Slow”, limited multicore algorithms**

Apache Arrow
Supports categorical data well with dictionary



Memory-efficient
Performant

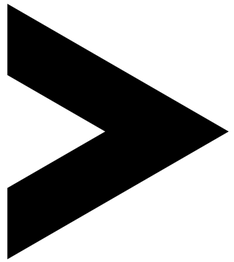
Pandas 2 and Polars levarage Arrow Dictionary Type to solve one more Pandas problem

Getting closer to the metal	Better groupby operations
No support for memory-mapped datasets	DataFrame appending is more memory-efficient
Poor performance in database and file ingest/export	Extensible type metadata
Doing missing data right	**Eager evaluation model, no query planning**
Better support for categorical data	**“Slow”, limited multicore algorithms**

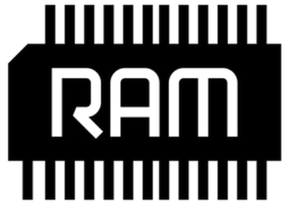
Arrow array supports memory-mapped datasets

Dataset

50GB

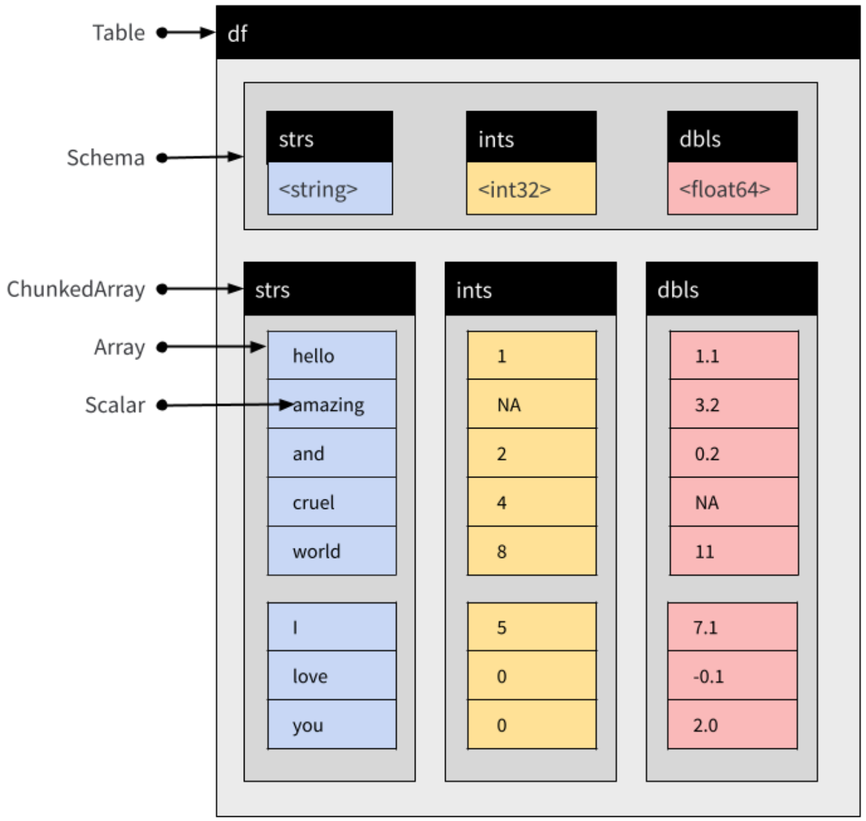


RAM

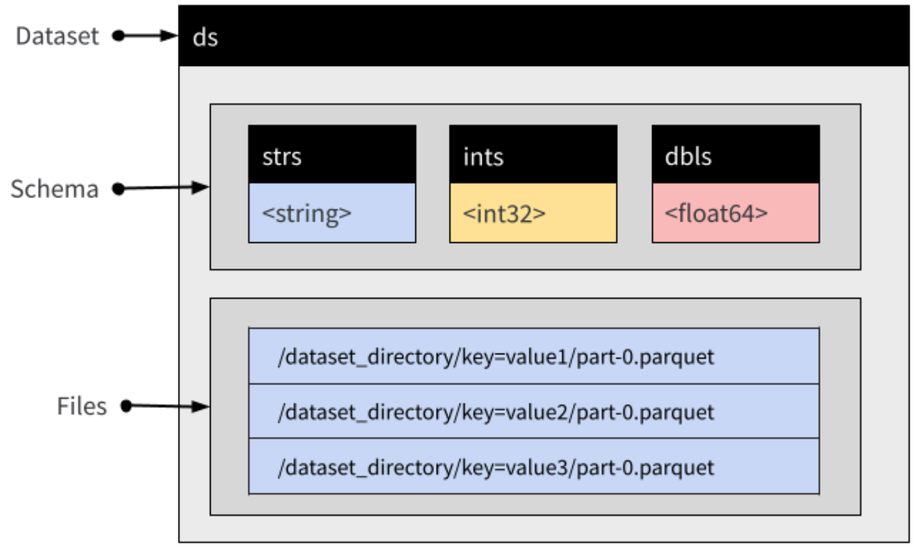


16GB

Arrow array supports memory-mapped datasets

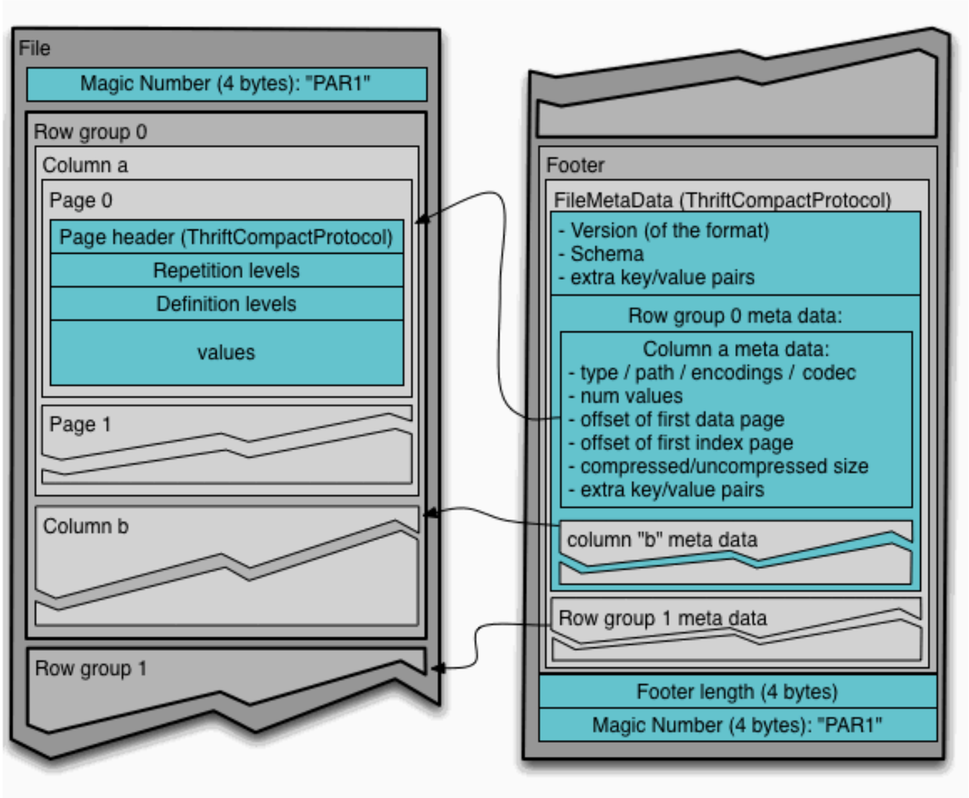


In-memory dataframe



On-disk dataset

Arrow array supports memory-mapped datasets



Parquet file structutre

Pandas 2 and Polars levarage memory-mapped dataset support of Arrow to solve one more Pandas problem

Getting closer to the metal	Better groupby operations
Support for memory-mapped datasets	DataFrame appending is more memory-efficient
High speed data ingest and export	Extensible type metadata
Doing missing data right	**Eager evaluation model, no query planning**
Better support for categorical data	**“Slow”, limited multicore algorithms**

Query planning

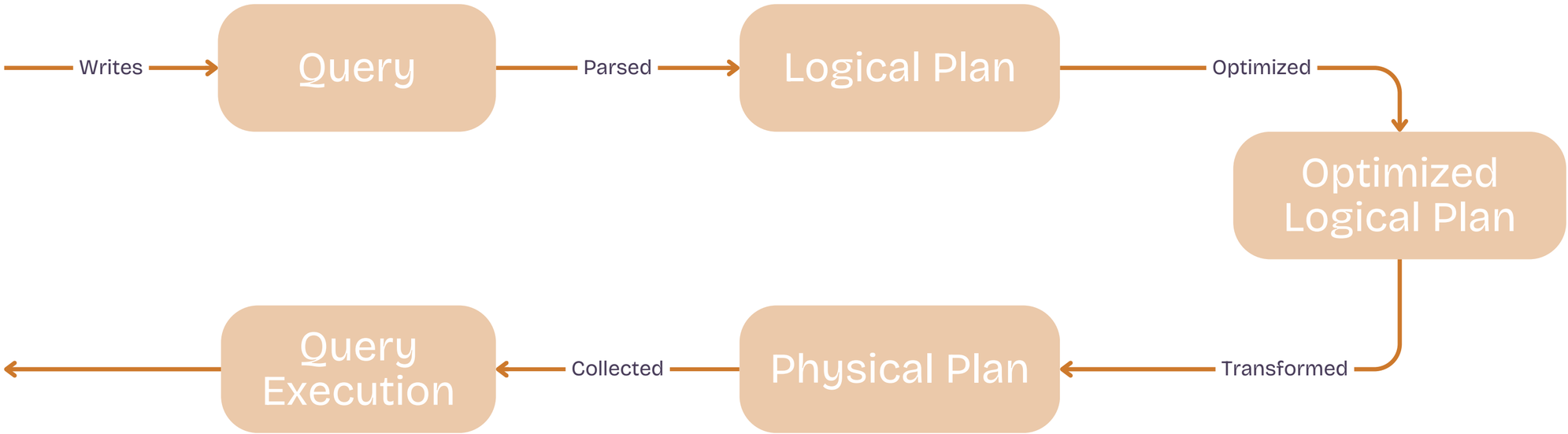
Eager evaluation
(imperative)

The expression
is evaluated
as soon as
it is called.

Lazy evaluation
(declarative)

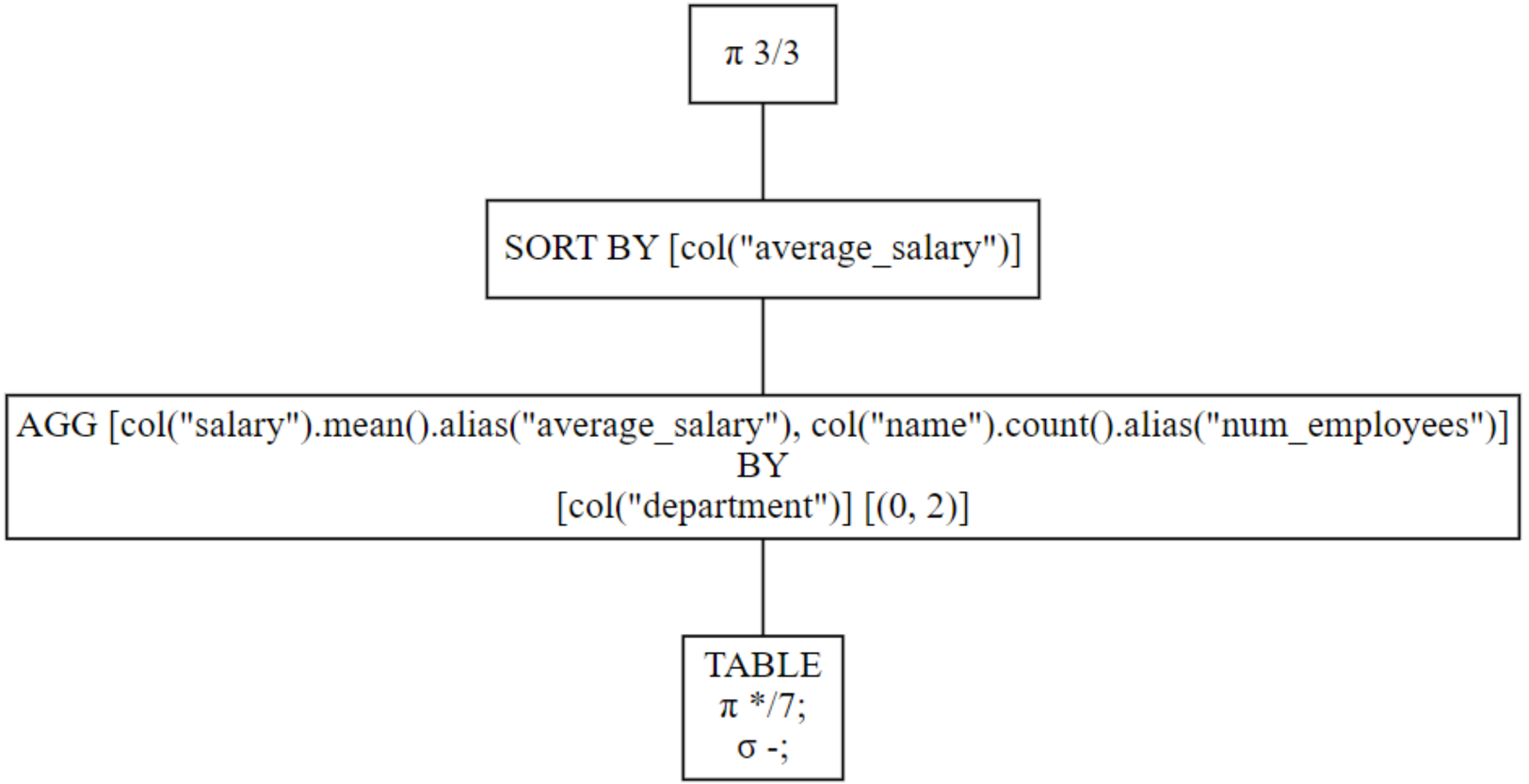
The expression
is stored as
computational
graph to be
evaluated
when needed.

Query planning

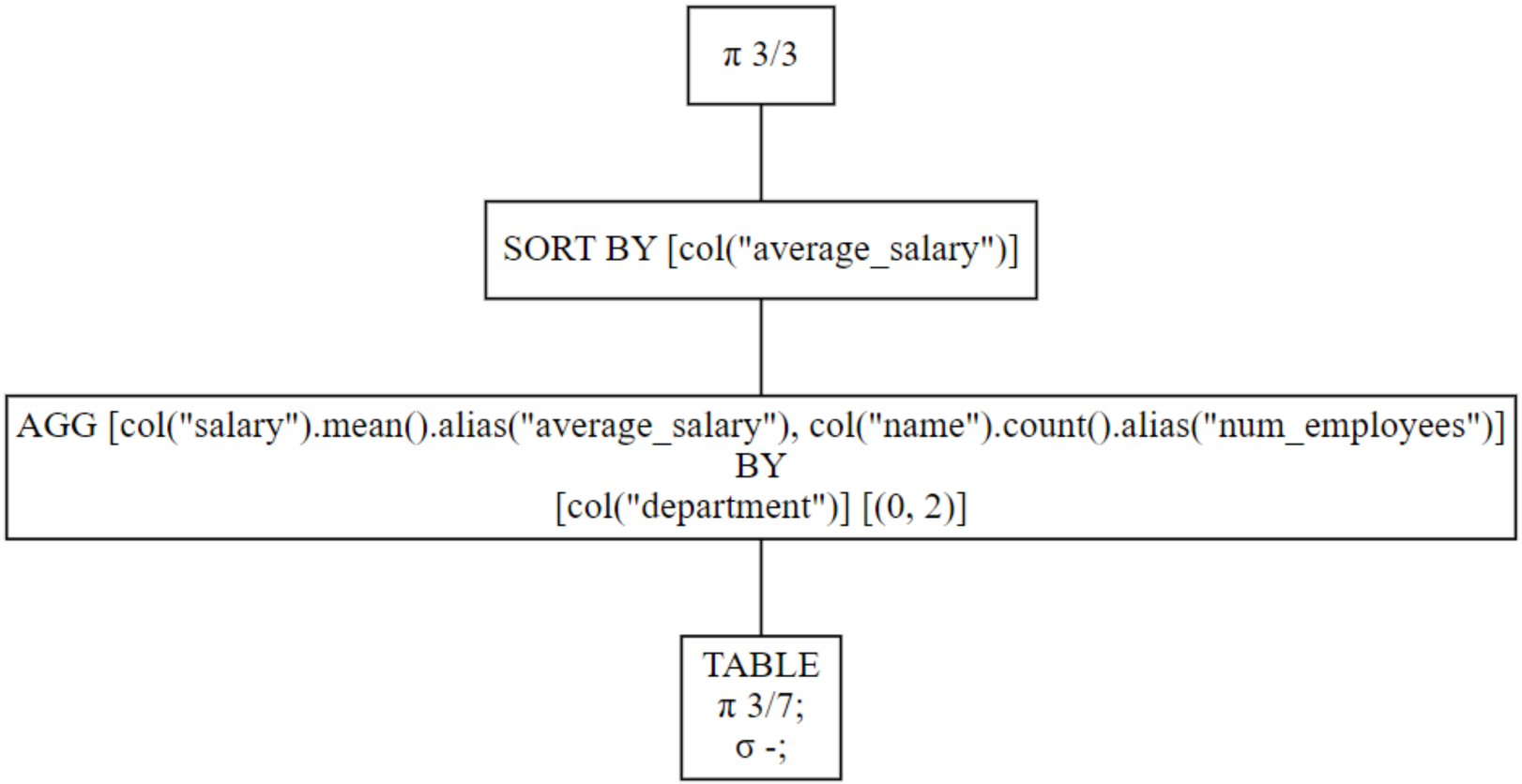



```
query.collect()
```

	pandas	
Eager evaluation	✓	✓
Lazy evaluation	✗	✓




```
query.show_graph(optimized=False)
```

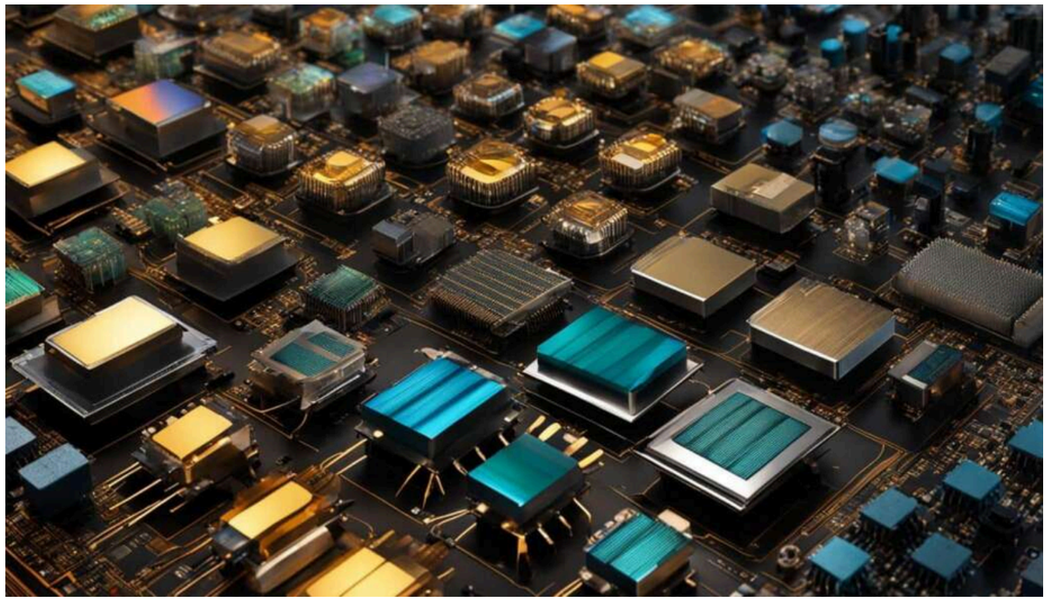



```
query.show_graph(optimized=True)
```

Polars supports Query Planning, which is a problem of Pandas (even Pandas 2)

Getting closer to the metal	Better groupby operations
Support for memory-mapped datasets	DataFrame appending is more memory-efficient
High speed data ingest and export	Extensible type metadata
Doing missing data right	**Support query planning with Lazy Evaluation**
Better support for categorical data	**“Slow”, limited multicore algorithms**

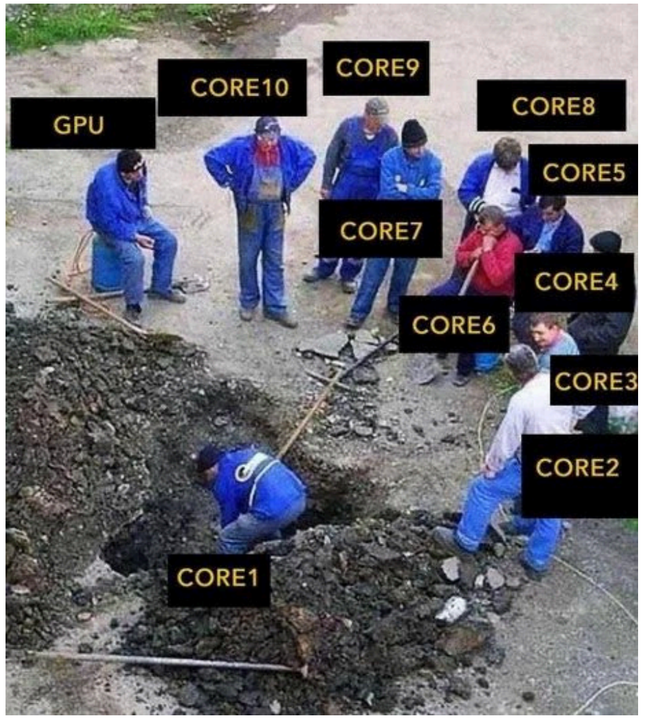
Parallelism

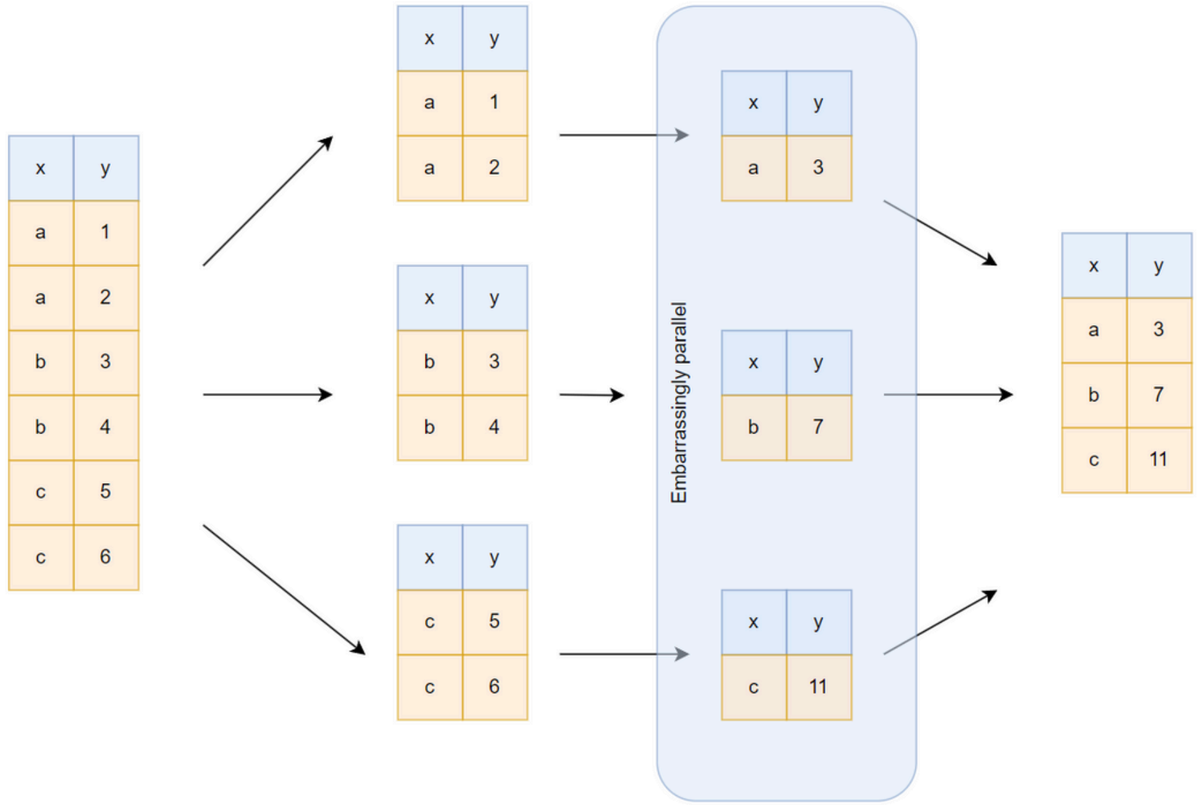


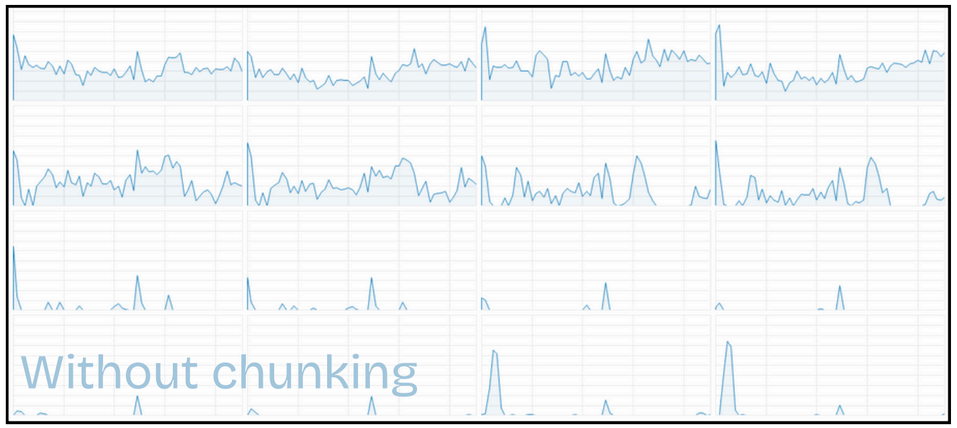
The advent of **multi-core processors** has revolutionized the world of computing with **parallel** processing.

Pandas does not take advantage of parallelism.

But Polars does!



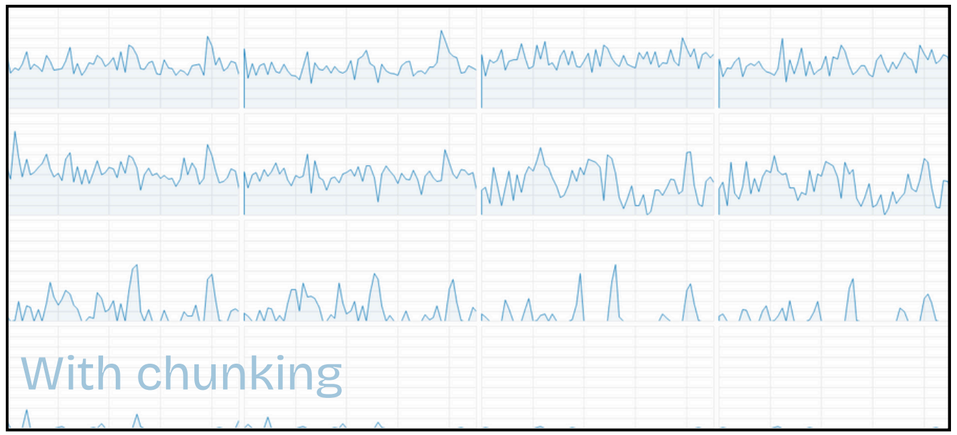

x y
a 1
a 2
b 3
b 4
c 5
c 6
x y
a 1
a 2
x y
b 3
b 4
x y
c 5
c 6
Embarrassingly parallel
x y
a 3
x y
b 7
x y
c 11
x y
a 3
b 7
c 11



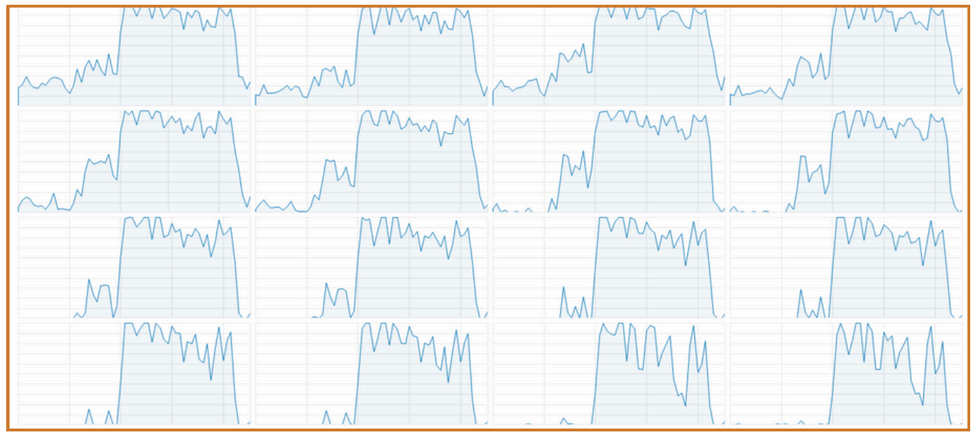
CPU utilization

of Pandas and Polars

in One Billion Row Challenge



Pandas



Polars

Polars supports Parallelism very well, which is a problem of Pandas (even Pandas 2)

Getting closer to the metal	Better groupby operations
Support for memory-mapped datasets	DataFrame appending is more memory-efficient
High speed data ingest and export	Extensible type metadata
Doing missing data right	Support query planning
Weak support for categorical data	**Levarage parallelism**

Polars

Apache Arrow

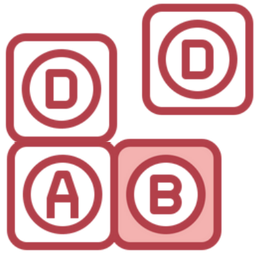
Query planning

Parallelism

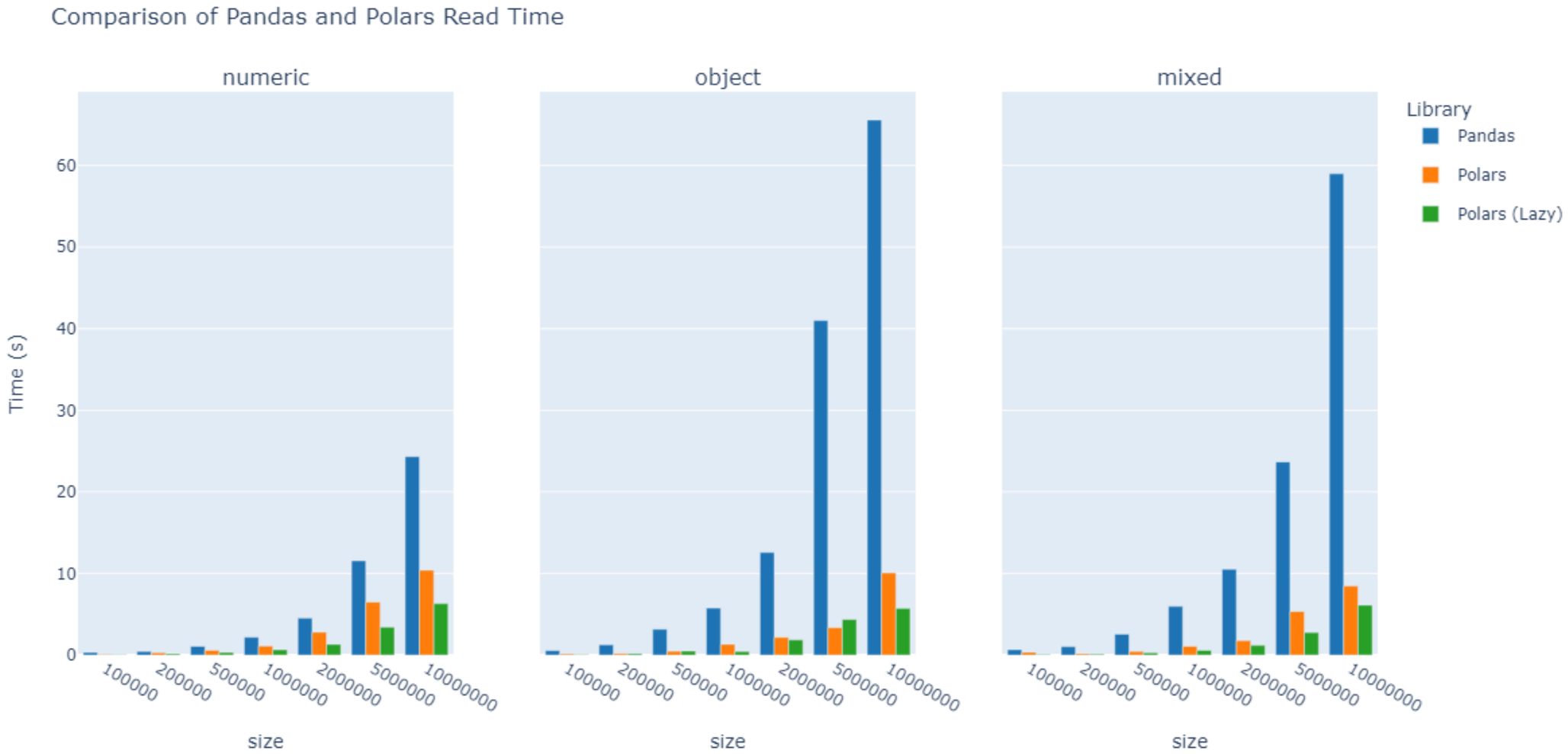
Columar format and SIMD, Validity buffer, Memory mapping, Categorical Type

03 COMPARISON

Compare run time of Pandas 1 and Polars on datasets

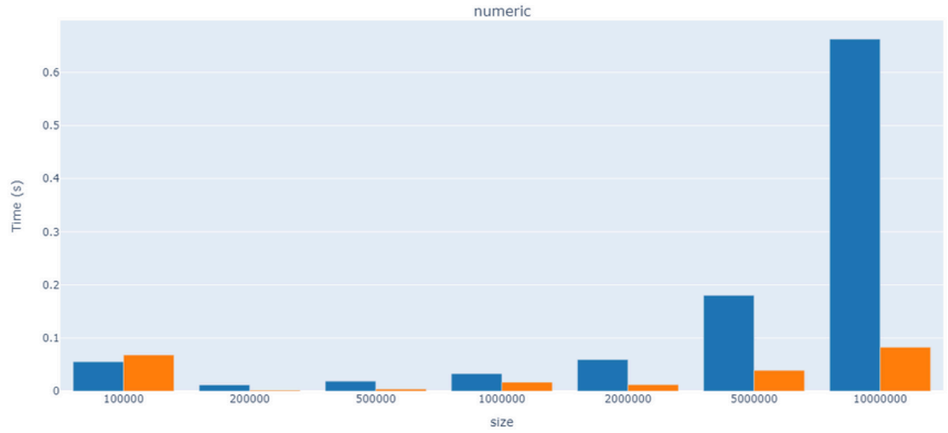
Dataset Type	 Numerical data only	 Objects only	 Mixed data
Dataset Size	100,000	200,000	500,000
	1,000,000	2,000,000	5,000,000
	10,000,000		

Read csv file

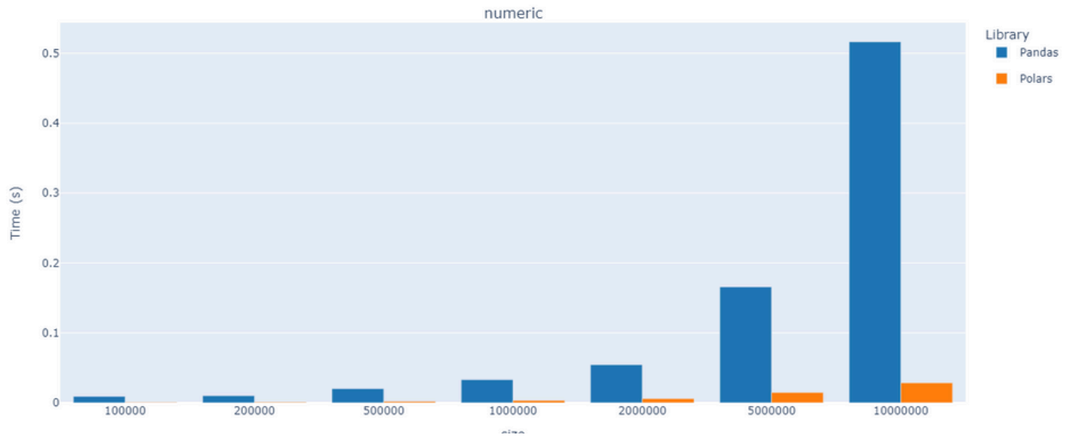


Aggregation

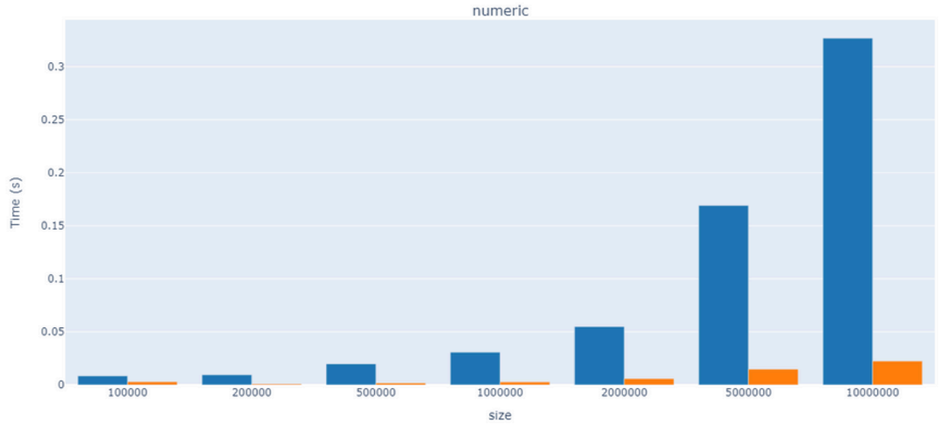
Min



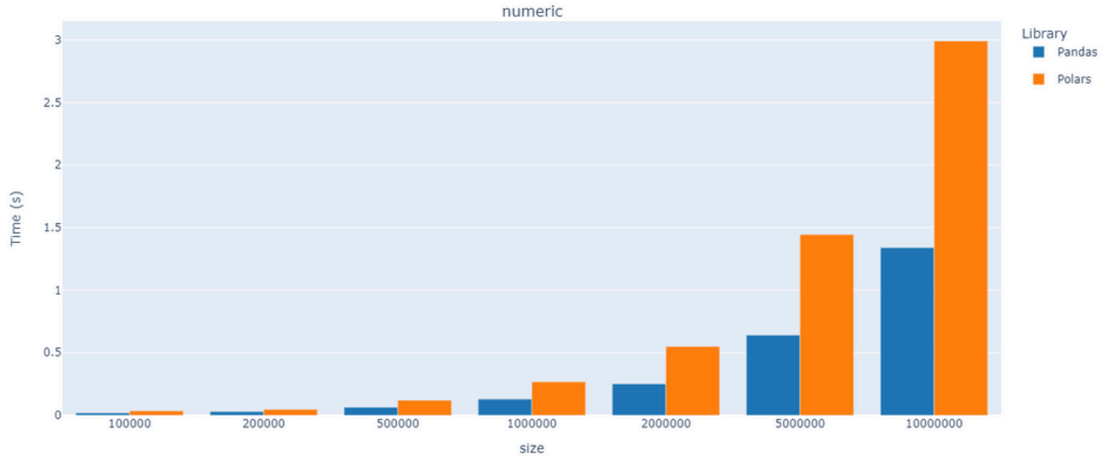
Max



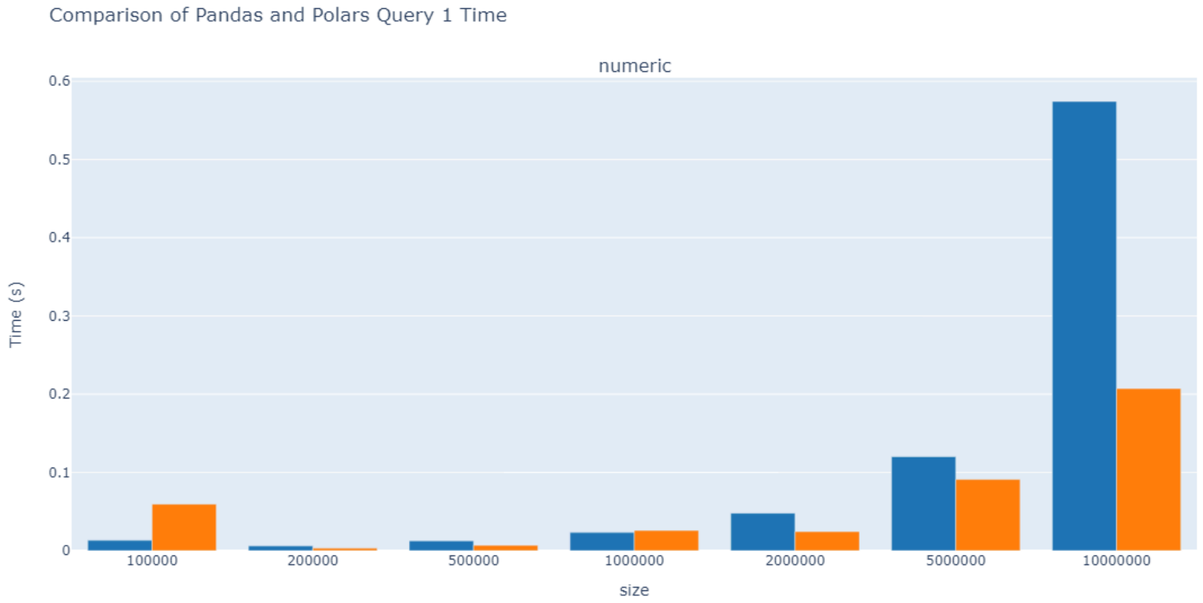
Mean



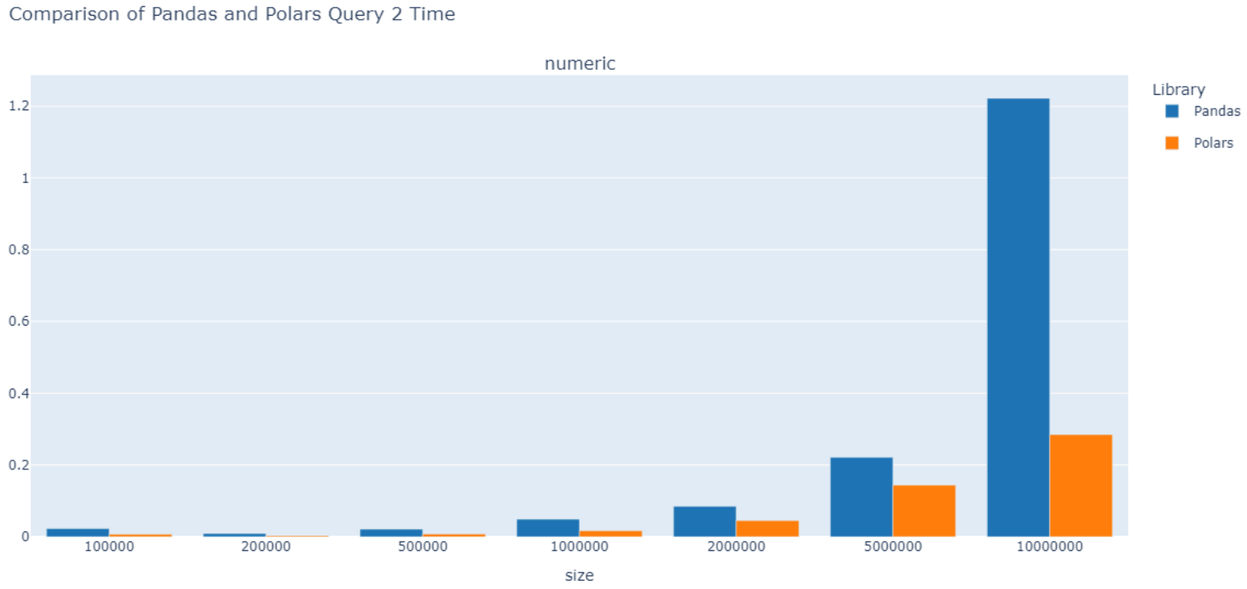
Median



Query

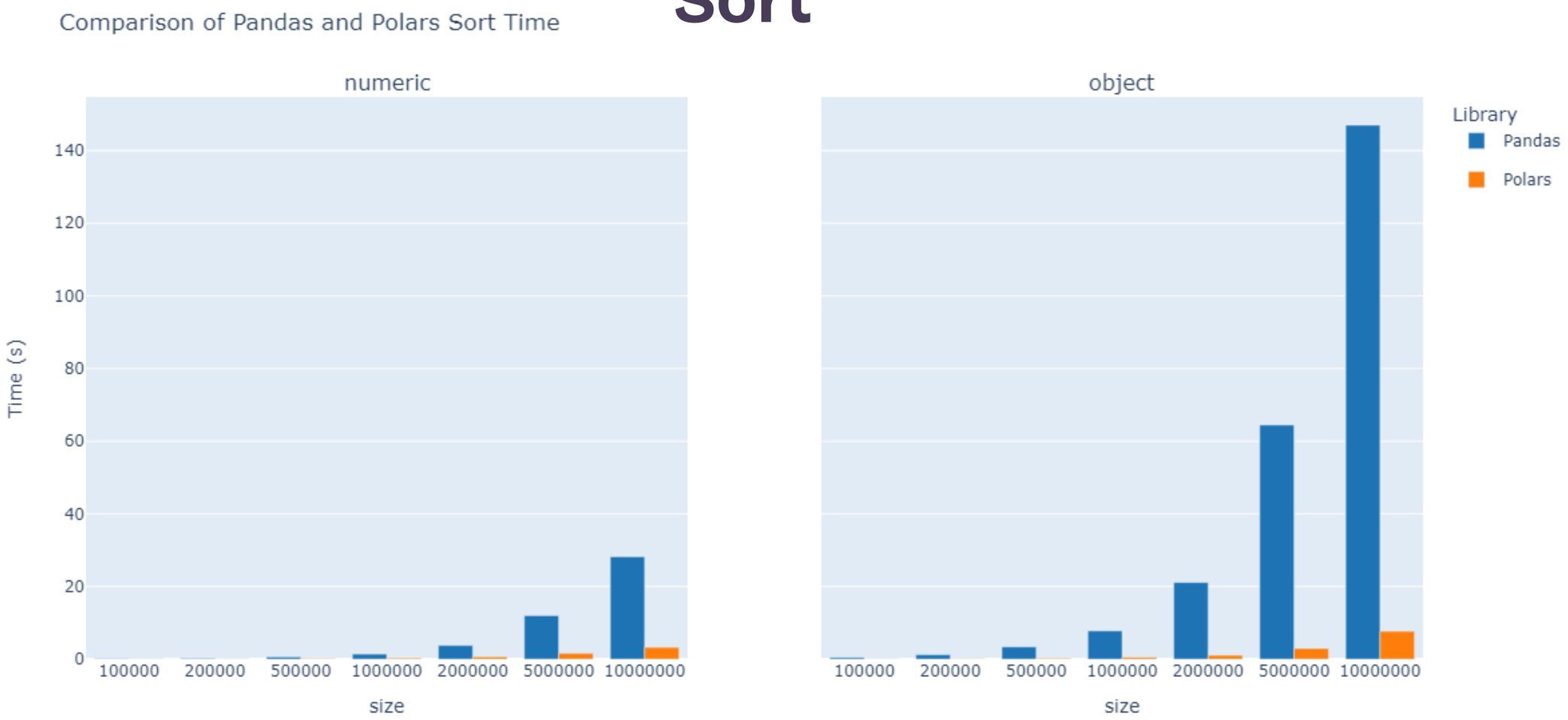


Query 1: Filter

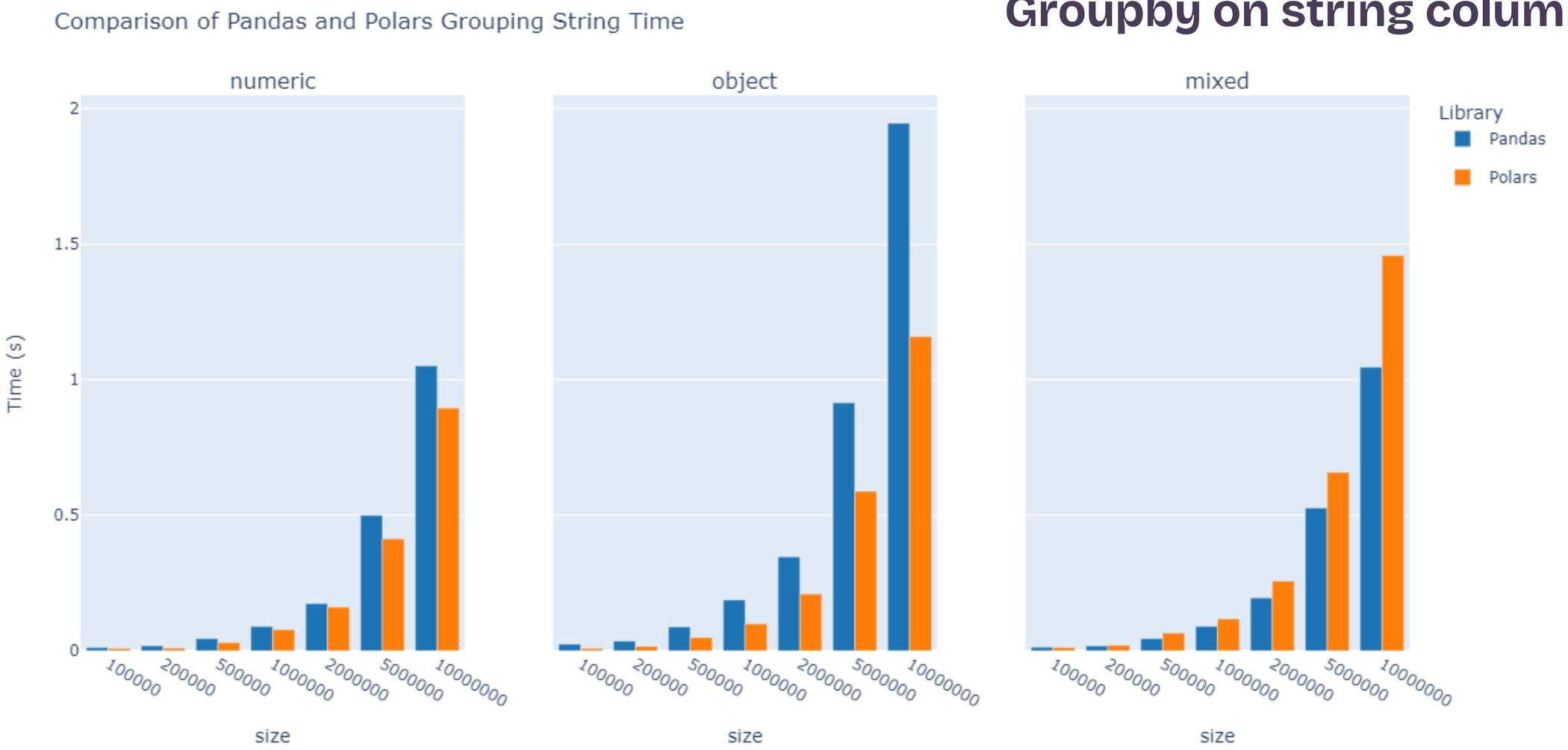


Query 2: Filter + Select

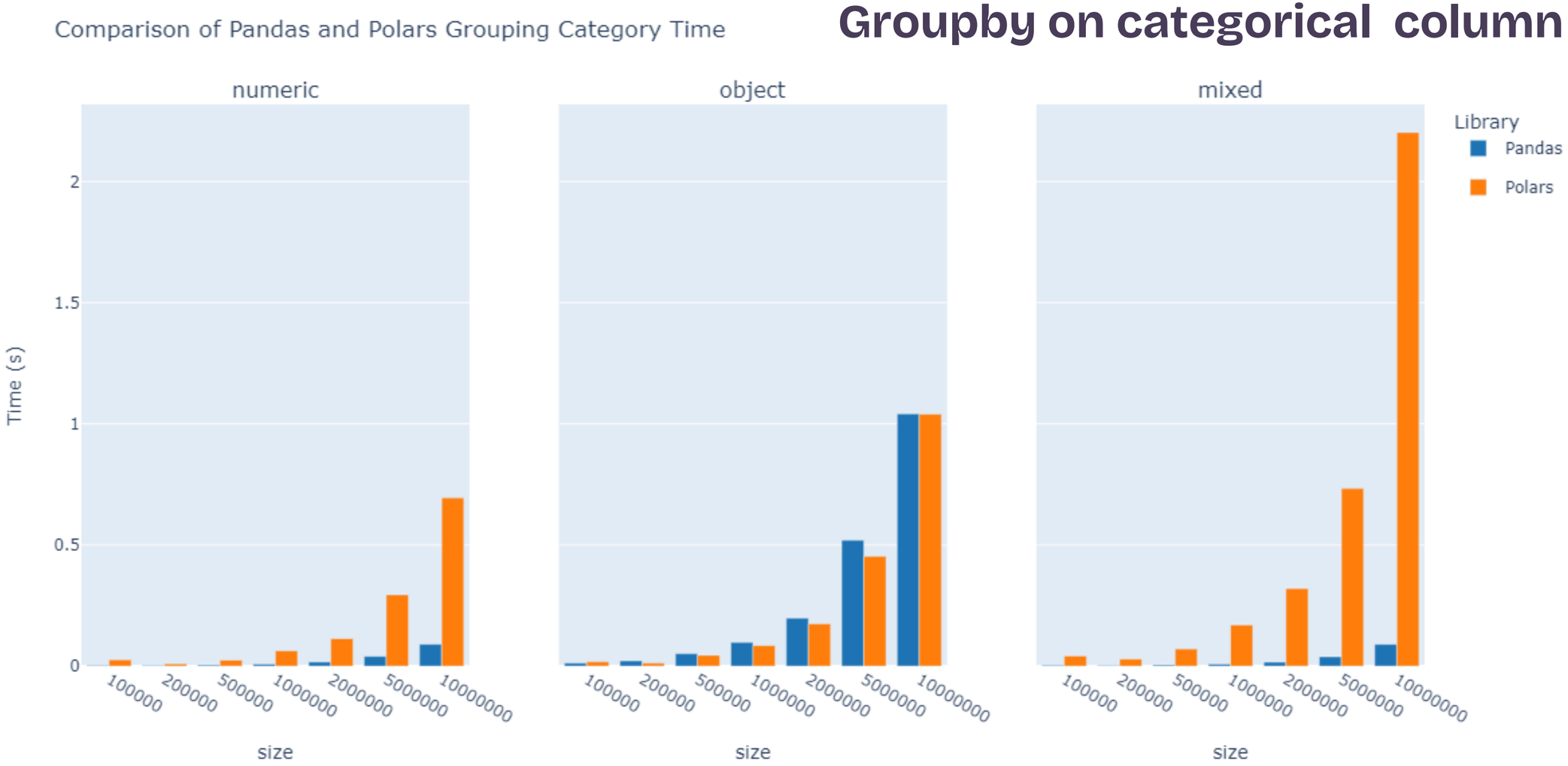
Sort


Comparison of Pandas and Polars Sort Time
numeric
object
Time (s)
0
20
40
60
80
100
120
140
100000
200000
500000
1000000
2000000
5000000
10000000
size
Library
Pandas
Polars

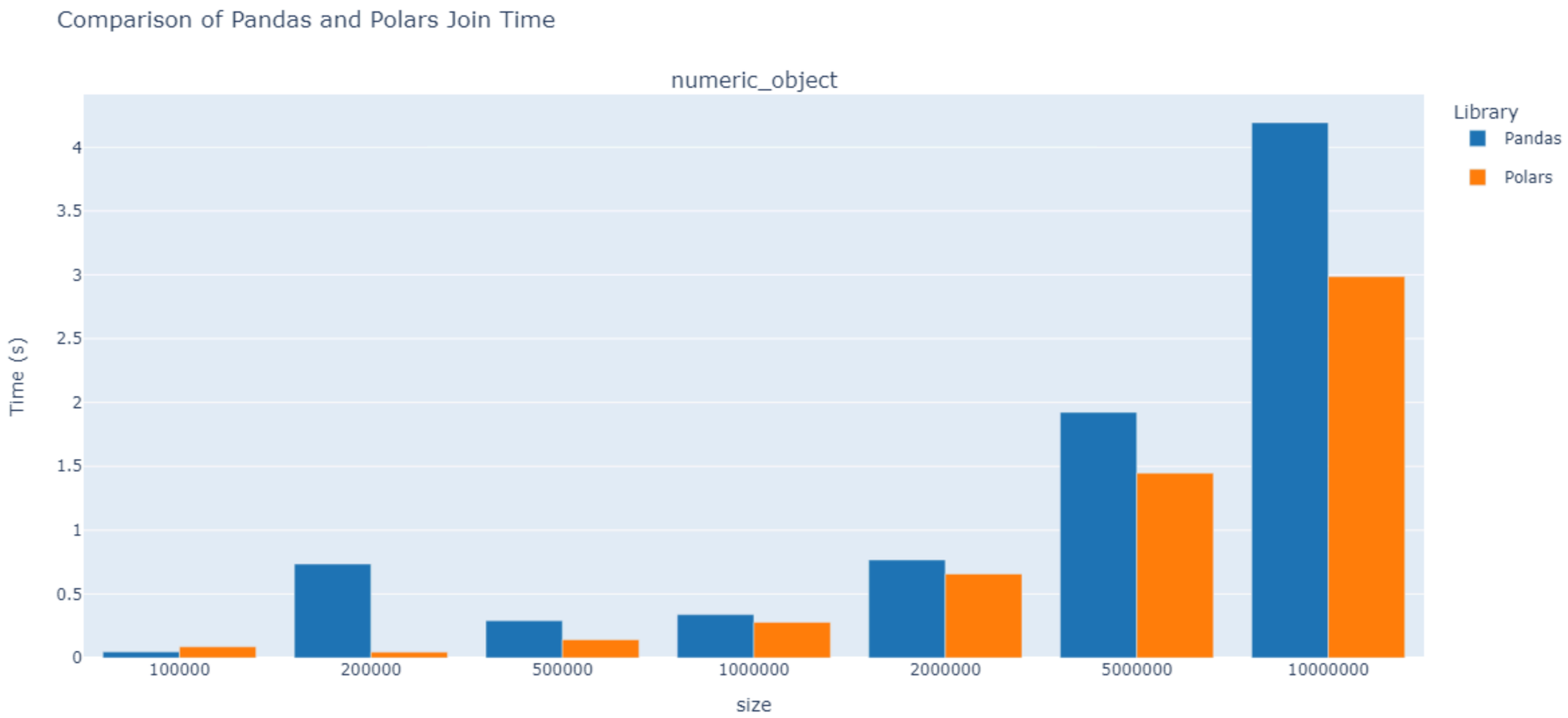
Groupby on string column



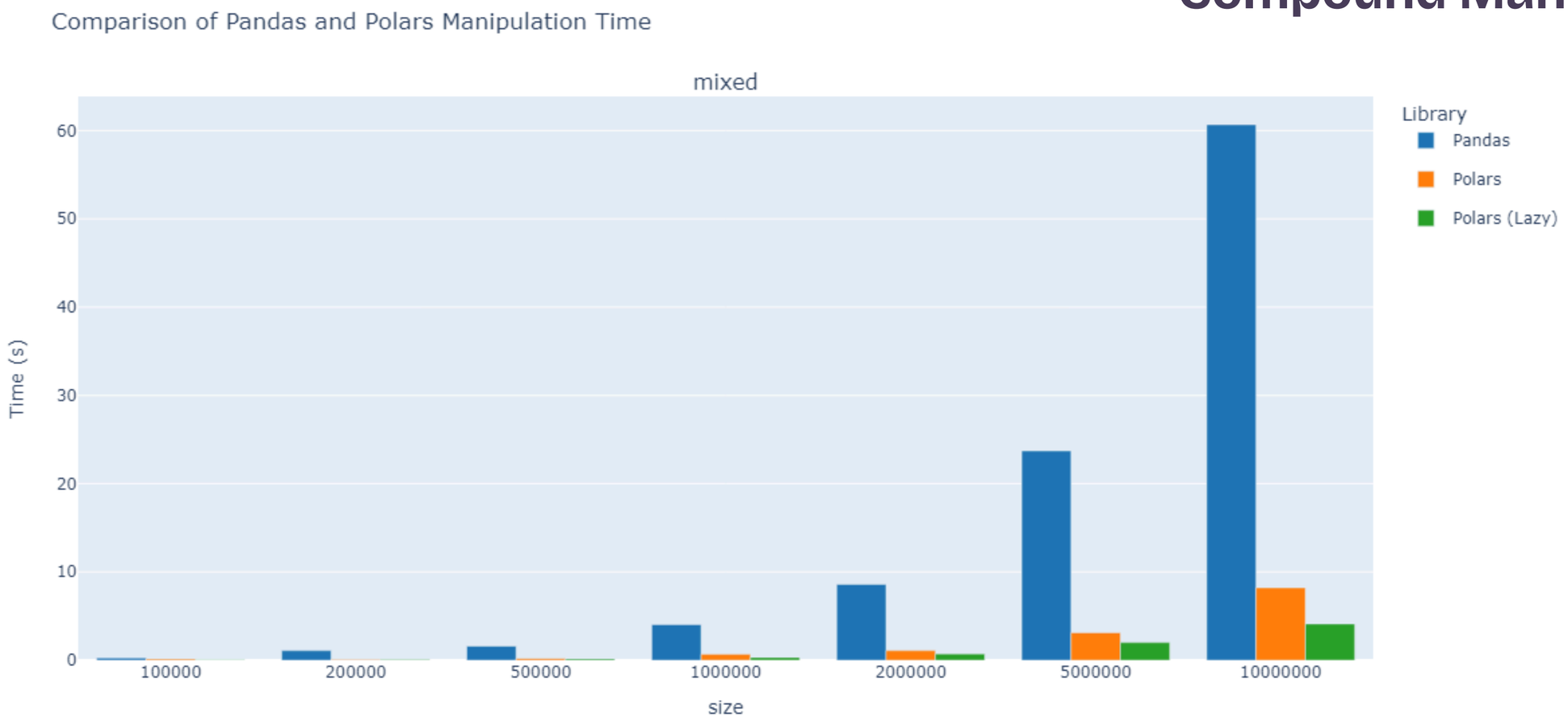
Groupby on categorical column



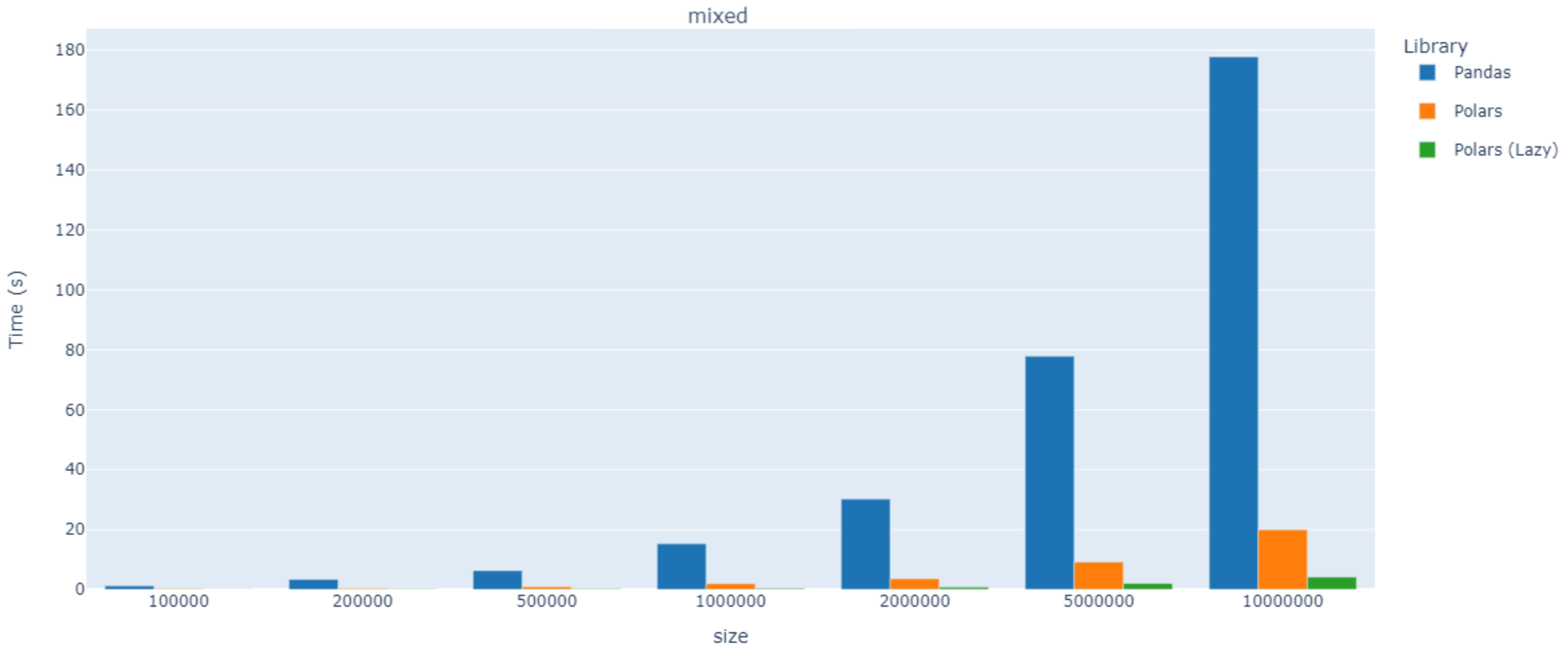
Join


Comparison of Pandas and Polars Join Time
numeric_object
Time (s)
0
0.5
1
1.5
2
2.5
3
3.5
4
100000
200000
500000
1000000
2000000
5000000
10000000
size
Library
Pandas
Polars

Compound Manipulation



Read + Manipulation



04 TUTORIAL

05 CONCLUSION

Pros | Polars over Pandas 1

Close to the metal	Better groupby operations
Support for memory-mapped datasets	DataFrame appending is more memory-efficient
High speed data ingest and export	Extensible type metadata
Doing missing data right	Support query planning
Better support for categorical data	Levarage parallelism

Pros | Polars over Pandas 2

~~Close to the metal~~	~~Better groupby operations~~
~~Support for memory-mapped datasets~~	~~DataFrame appending is more memory-efficient~~
~~High speed data ingest and export~~	~~Extensible type metadata~~
~~Doing missing data right~~	**Support query planning**
~~Better support for categorical data~~	**Levarage parallelism**

Cons | Polars has limited ecosystem

Only compatible with

Not compatible yet

`to_numpy()`

Thank you

for your attention!