scieneers
DRIVEN BY DATA

Raised by Pandas, striving for more: An opinionated introduction to Polars

Nico Kreiling

PyCon DE, 2023

scieneers
DRIVEN BY DATA

We gain knowledge from data and create value. For our customers, society and ourselves.

Nico Kreiling

Data Scientist @ scieneers

Host of techtiefen.de NicoKreiling

2013	2014	2015	2016	2017	2018	2019	2020	2021	2022	2023
R		pandas	Apache Spark			dask	MODIN	cuDF		

Why should you care about Polars?

There are a couple of “too good to be true”-like performance-benchmarks

Ritchie Vink (@ritchie46@fosstodon.... @RitchieVi... · 9. Juni 2021

Polars DataFrame library 0.8.4 is out. Latest weeks have seen a lot of **performance** improvements, leading to the fastest release to date. And it shows!

*Note that today's release is even faster to the result shown in the **benchmark** ;)

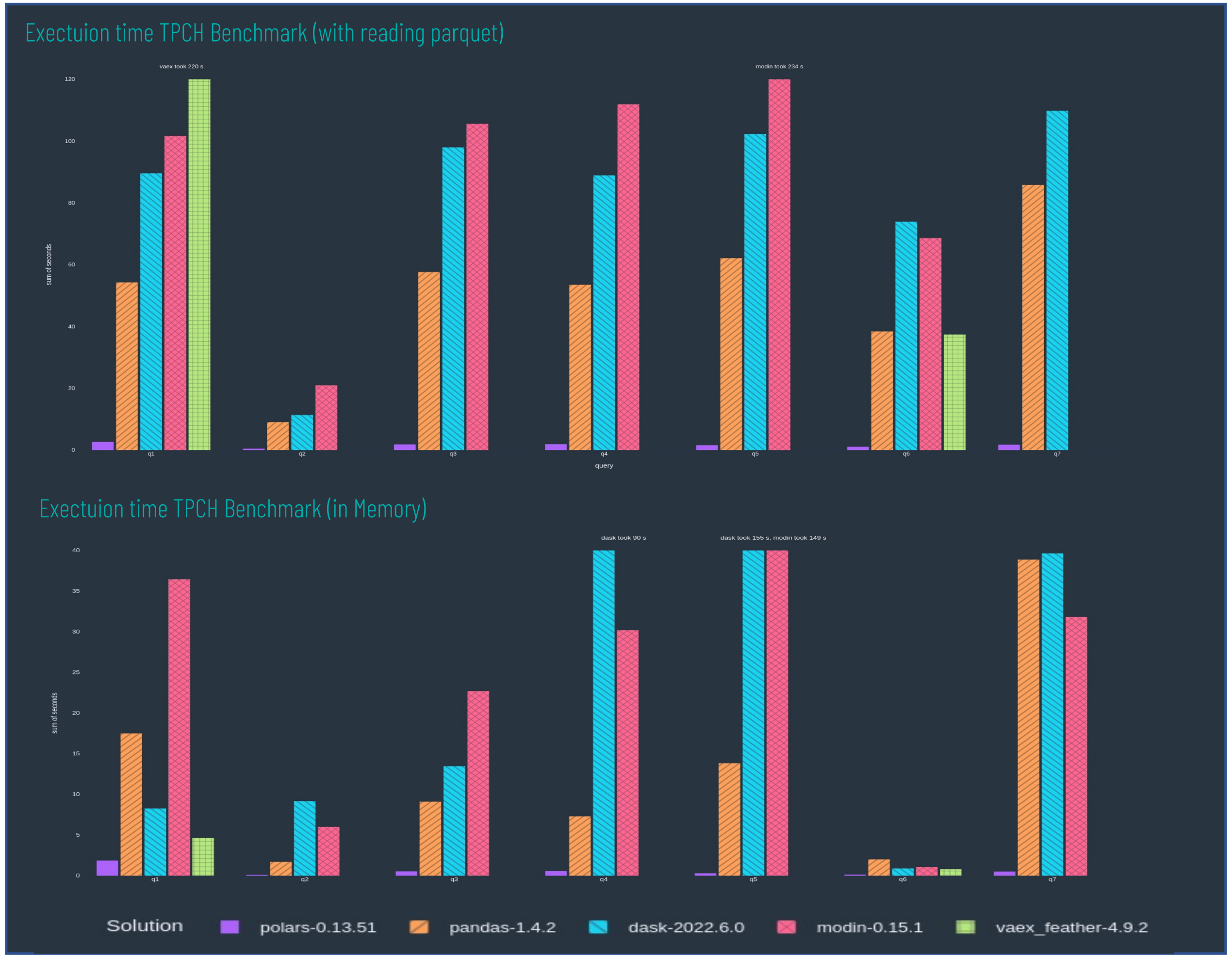
#rustlang #Python #data

advanced questions

Input table: 100,000,000 rows x 9 columns (5 GB)

Polars	0.8.3	2021-06-08	59s
ClickHouse	21.3.2.5	2021-05-12	69s
DataFrames.jl	1.1.1	2021-05-15	116s
data.table	1.14.1	2021-05-31	118s
(py)datatable	1.0.0a0	2021-06-08	312s
pandas	1.2.4	2021-04-29	1090s
dplyr	1.0.6	2021-05-08	4209s
Arrow	4.0.1	2021-05-31	4273s
spark	3.1.2	2021-05-31	not yet implemented
dask	2021.04.1	2021-05-09	internal error
cuDF*	0.19.2	2021-05-31	out of memory
DuckDB	0.2.6	2021-05-09	inaccurate

1 6



Pandas isn’t perfect

scieneers DRIVEN BY DATA

Critique on Pandas from the author Wes McKinney himself

Started Pandas in April 2008 as a side-project at night-time and on weekends

“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 (2017)

Taken from: https://wesmckinney.com/blog/apache-arrow-pandas-internals/ (2017)

Pandas problems are well known – at least since 2017?

scieneers DRIVEN BY DATA

In his article, Wes McKinney also highlights 11 points, where Pandas lacks

1. Internals too far from "the metal"
2. No support for memory-mapped datasets
3. Poor performance in database and file ingest / export
4. Warty missing data support
5. Lack of transparency into memory use, RAM management
6. Weak support for categorical data
7. Complex groupby operations awkward and slow
8. Appending data to a DataFrame tedious and very costly
9. Limited, non-extensible type metadata
10. Eager evaluation model, no query planning
11. "Slow", limited multicore algorithms

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 pandas2.

Source: https://wesmckinney.com/blog/apache-arrow-pandas-internals/ (2017)

In Case of performance issues, follow

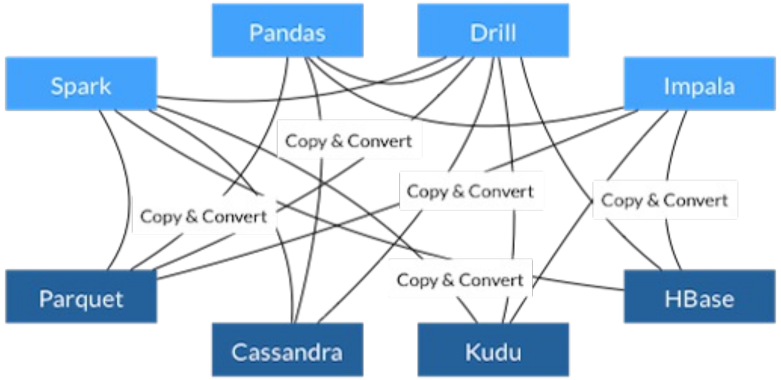
ARROW

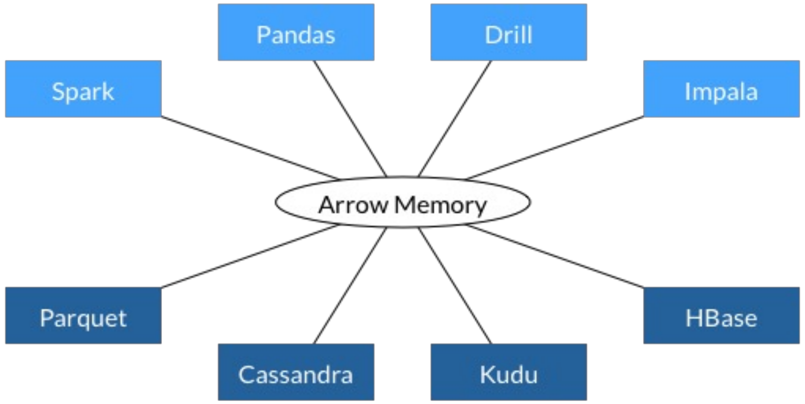
Gain 1: Apache Arrow

scieneers
DRIVEN BY DATA

Apache Arrow enables an efficient data access across libraries and languages

- Initial Release 2016
- Arrow standardizes a columnar data format across languages
- “has a very cache-coherent data structure” (Ritchie Vink)
- Natively supports missing data through additional validity bits
- Much better and faster string support





Source: https://arrow.apache.org/overview/

Arrow does solve many Pandas Problems

scieneers
DRIVEN BY DATA

Those are exactly the areas, where Pandas 2.0 made some big steps!

1. Internals too far from "the metal"
2. No support for memory-mapped datasets
3. Poor performance in database and file ingest / export — pandas v2
4. Warty missing data support — pandas v2
5. Lack of transparency into memory use, RAM management — pandas v2
6. Weak support for categorical data — pandas v2
7. Complex groupby operations awkward and slow
8. Appending data to a DataFrame tedious and very costly — pandas v2
9. Limited, non-extensible type metadata — pandas v2
10. Eager evaluation model, no query planning
11. "Slow", limited multicore algorithms

Gain 2: Single Instruction, Multiple Data (SIMD)

Handle full vectors instead of single numbers in a single CPU cycle

This is also why you should **avoid pandas apply**!

MIMD – Every Instruction is treated on its one

SIMD – Handle all data at once

Polars Expression API is very expressive

Which makes it easier to utilize SIMD instructions

`pl.DataFrame` works very similar to `pd.DataFrame`, just without indexes

`with_columns` is your go to function to add or change columns

`pl.lit` creates a suitable vector a single value (just like in spark)

`alias` defines the name of the columns (without this, it overwrites the modified column)

As Polars has no indexes, there is only one `sort` function

```
import polars as pl

gh_stars = pl.DataFrame({
    "year": [2019,2020,2021,2022,2023],
    "Dask": [3763,5853,7487,9204,10586],
    "Pandas": [16608, 21835, 26996, 31544, 36208],
    "Polars": [None, None,544,3914, 11128]
}).with_columns(pl.lit("stars").alias("metric"))

gh_pullrequests = pl.DataFrame({
    "year": [2023,2022,2021,2020,2019],
    "Dask": [26886, 23365, 19282, 13973, 10464],
    "Pandas": [5007, 4331, 3585, 2996, 2343],
    "Polars": [3542, 1365, 96, None, None],
}).with_columns(pl.lit("pullrequests").alias("metric"))

df = pl.concat([gh_stars, gh_pullrequests]).sort(["metric","year"])
```

Resulting DataFrame:

year	Dask	Pandas	Polars	metric
i64	i64	i64	i64	str
2019	10464	2343	null	"pullrequests"
2020	13973	2996	null	"pullrequests"
2021	19282	3585	96	"pullrequests"
2022	23365	4331	1365	"pullrequests"
2023	26886	5007	3542	"pullrequests"
2019	3763	16608	null	"stars"
2020	5853	21835	null	"stars"
2021	7487	26996	544	"stars"
2022	9204	31544	3914	"stars"
2023	10586	36208	11128	"stars"

Polars Expression API is very expressive

Which makes it easier to utilize SIMD instructions

`pl.col` is probably the most typed function: Reference one or more columns

Polars provides many expressions out of the box, such as `diff` or `pct_change`

`prefix` is similar to `alias`, just for multiple columns

```
stats = df.with_columns([
    pl.col(["Dask","Pandas","Polars"]).diff().over("metric").prefix("delta_"),
    pl.col(["Dask","Pandas","Polars"]).pct_change().over("metric").prefix("perc_")
])
```

`over` is a powerfull keyword to limit the option range by given columns

year	Dask	Pandas	Polars	metric	delta_Dask	delta_Pandas	delta_Polars	perc_Dask	perc_Pandas	perc_Polars
i64	i64	i64	i64	str	i64	i64	i64	f64	f64	f64
2019	10464	2343	null	"pullrequests"	null	null	null	null	null	null
2020	13973	2996	null	"pullrequests"	3509	653	null	0.33534	0.278703	null
2021	19282	3585	96	"pullrequests"	5309	589	null	0.379947	0.196595	null
2022	23365	4331	1365	"pullrequests"	4083	746	1269	0.211752	0.208089	13.21875
2023	26886	5007	3542	"pullrequests"	3521	676	2177	0.150695	0.156084	1.594872
2019	3763	16608	null	"stars"	null	null	null	null	null	null
2020	5853	21835	null	"stars"	2090	5227	null	0.555408	0.314728	null
2021	7487	26996	544	"stars"	1634	5161	null	0.279173	0.236364	null
2022	9204	31544	3914	"stars"	1717	4548	3370	0.229331	0.168469	6.194853
2023	10586	36208	11128	"stars"	1382	4664	7214	0.150152	0.147857	1.843127

Resulting DataFrame

Polars API – Key Take-Aways

- Polars has **no indexes**
- Powerfull Expression API to better use SIMD performance boost
- **`over`** keyword is more concise then `groupby` and `join` combination
- Polars supports both: eager and lazy execution

```
%%time
df = pl.read_parquet("berlin_stations.parquet")
df.head()
```

```
CPU times: user 935 ms, sys: 654 ms, total: 1.59 s
Wall time: 933 ms
```

```
%%time
lazy_df = pl.scan_parquet("berlin_stations.parquet")
lazy_df.head().collect()
```

```
CPU times: user 23.2 ms, sys: 10.7 ms, total: 33.8 ms
Wall time: 14.9 ms
```

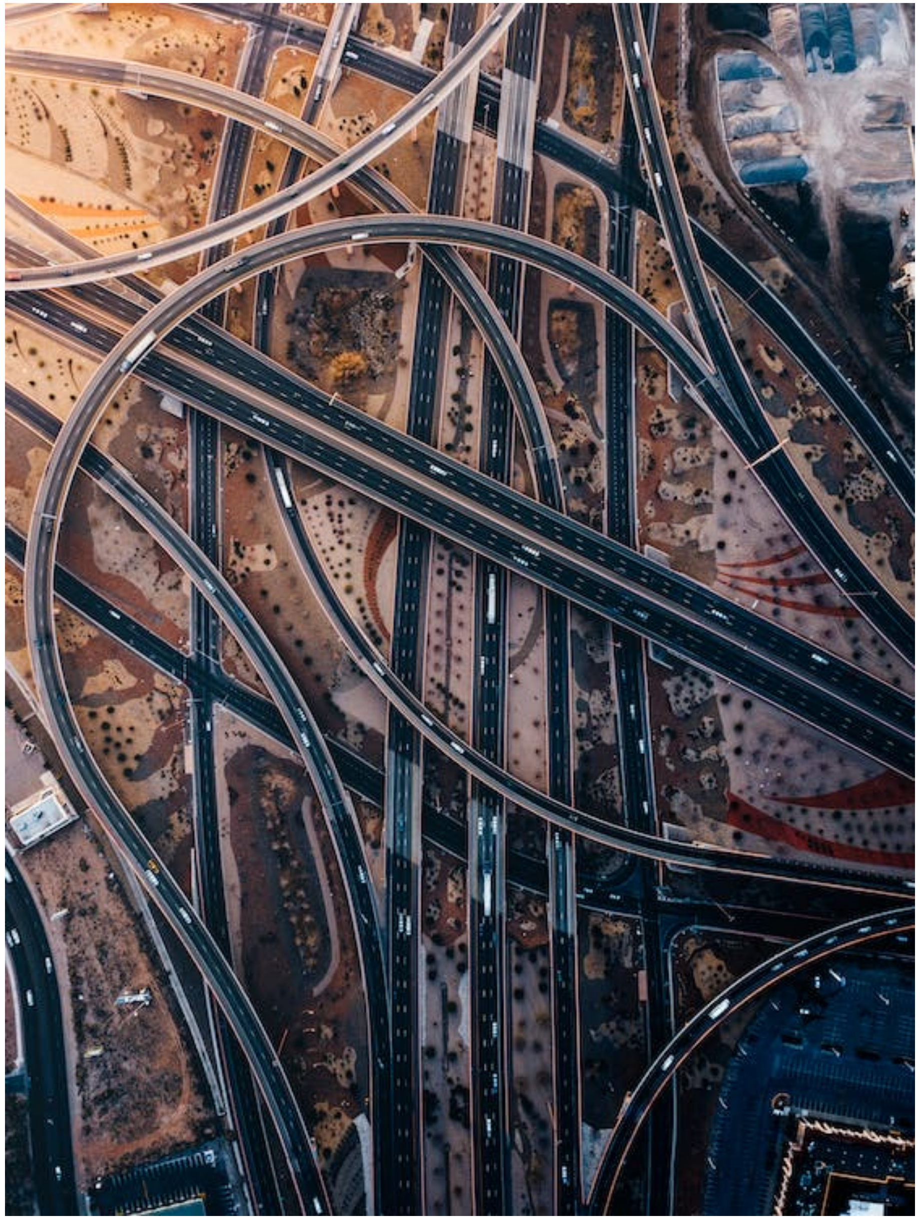
`read` trigger eager execution (imperative)

`scan` trigger lazy execution (declarative)

Query Optimization

The declarative DSL of Polars allows query optimization

- Reduce Cache misses
- Optimizing branch predictions
- Drop unnecessary computations
- Rewrite execution order and operations



The Polars API is very expressive and flexible

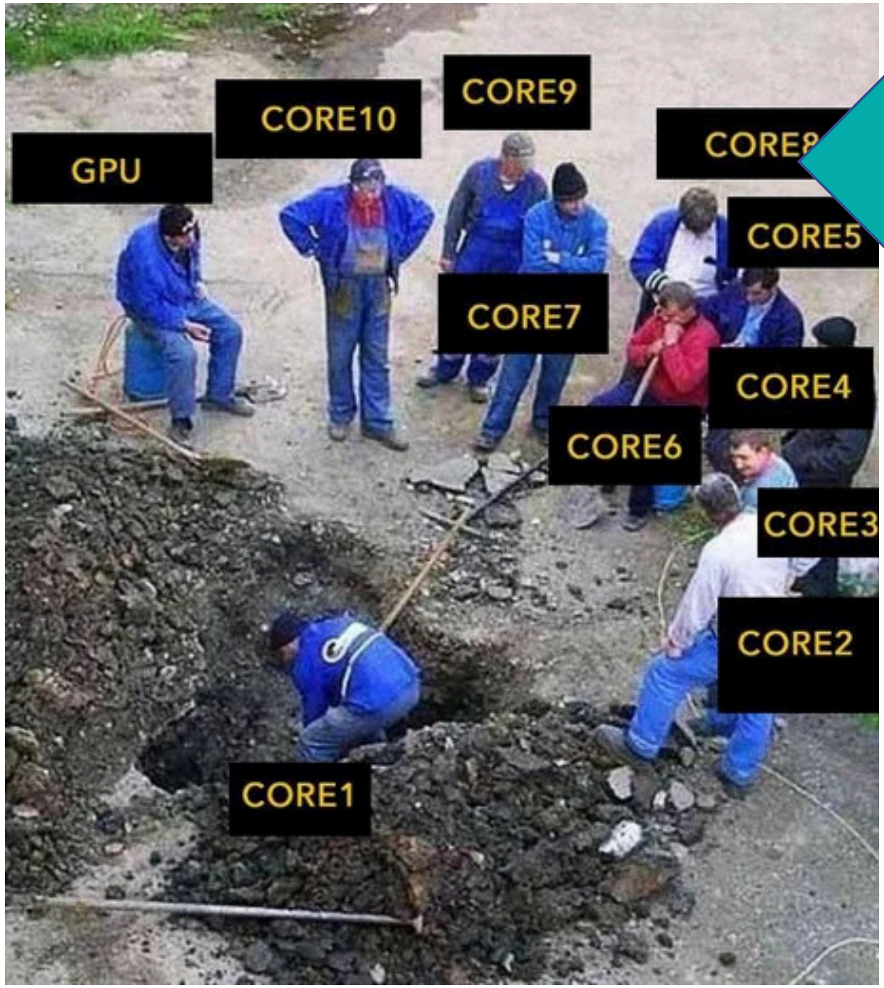
scieneers
DRIVEN BY DATA

This makes it fast

1. Internals too far from "the metal"
2. No support for memory-mapped datasets
3. Poor performance in database and file ingest / export — pandas V2
4. Warty missing data support — pandas V2
5. Lack of transparency into memory use, RAM management — pandas V2
6. Weak support for categorical data — pandas V2
7. Complex groupby operations awkward and slow
8. Appending data to a DataFrame tedious and very costly — pandas V2
9. Limited, non-extensible type metadata — Apache Spark, pandas V2
10. Eager evaluation model, no query planning — Apache Spark
11. "Slow", limited multicore algorithms

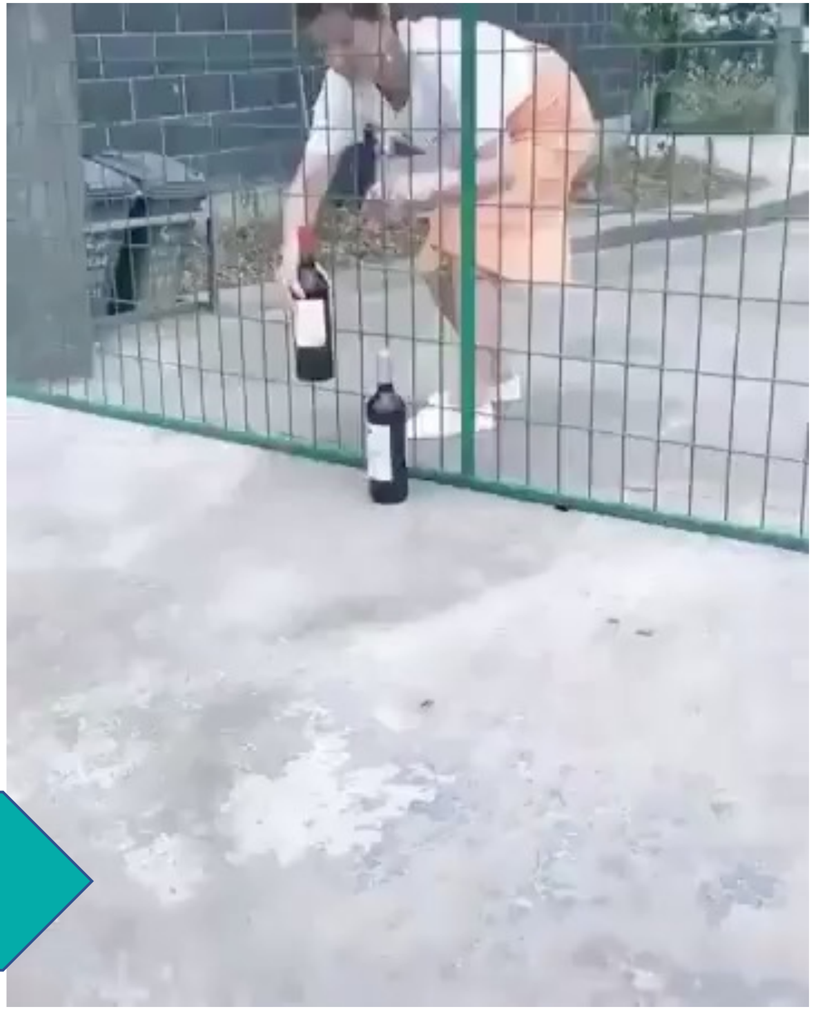
Gain 3: “Embarrassingly parallel”

Having a well defined DSL enables better parallelization



Without Parallelization (most of pandas operations)

Multi-Core Parallelization done wrong

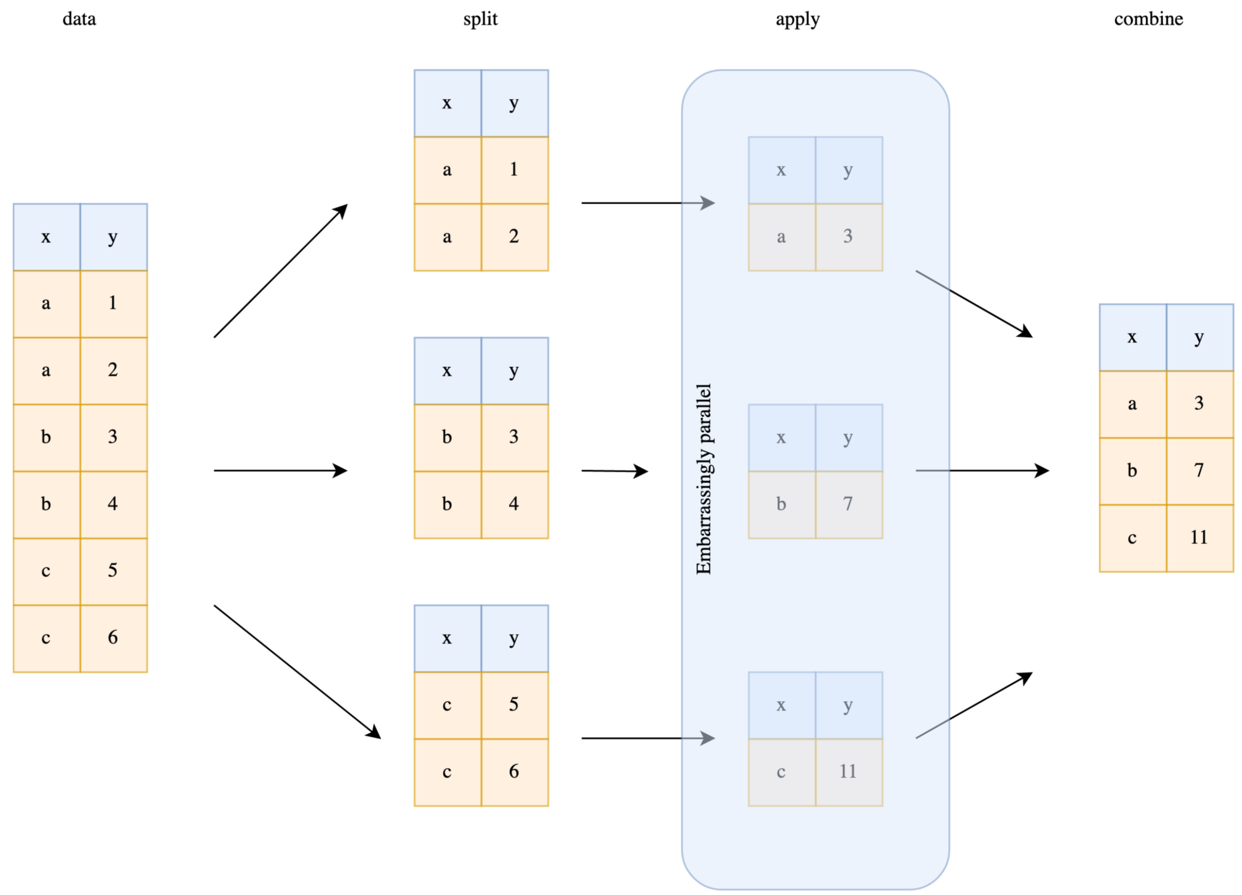


Embarrassingly parallel task execution

scieneers
DRIVEN BY DATA

Polars parallelizes everything that does not require communication

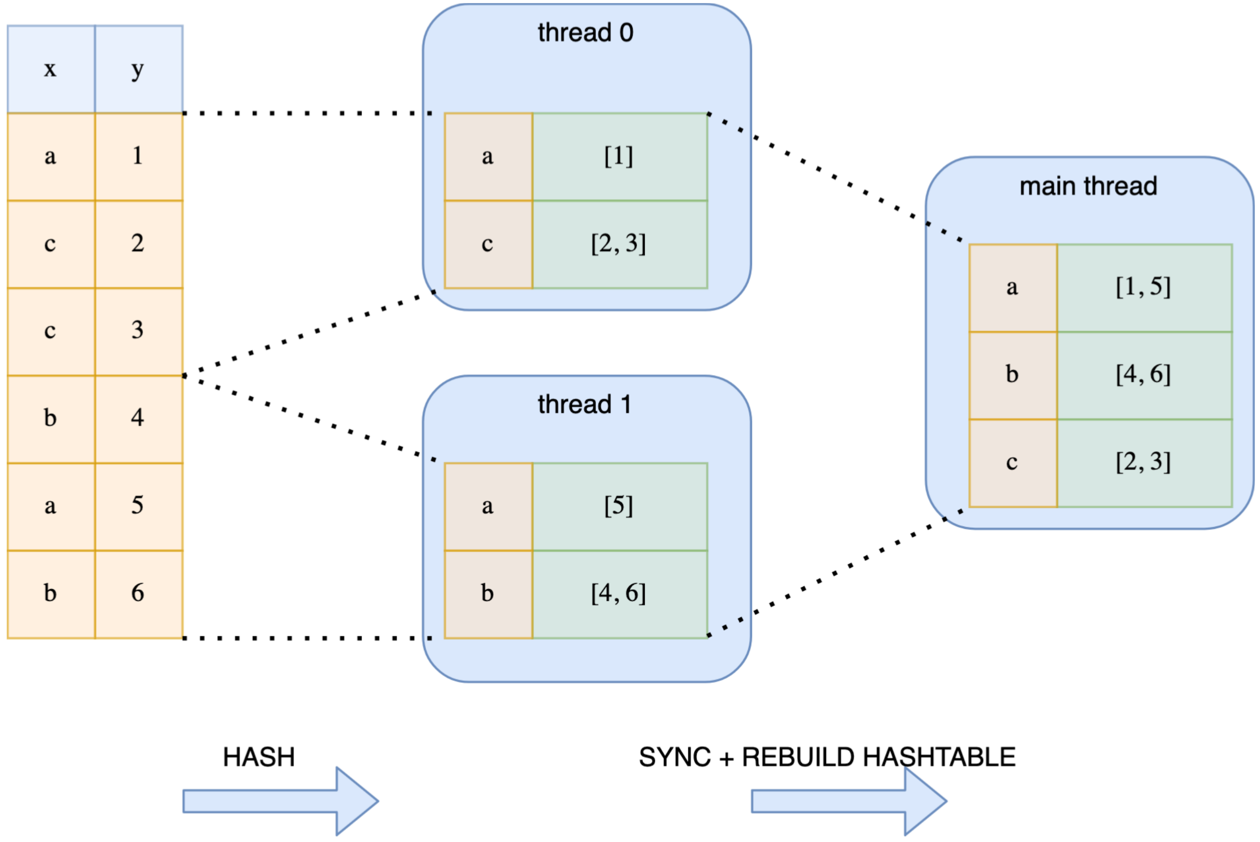
- aggregations across different columns can be parallelized easily
- Groupby-apply operations can be also be parallized



https://www.ritchievink.com/blog/2021/02/28/i-wrote-one-of-the-fastest-dataframe-libraries/

Prepare data for tasks that require communication

Idea 1: To simply split data by thrads does not work well



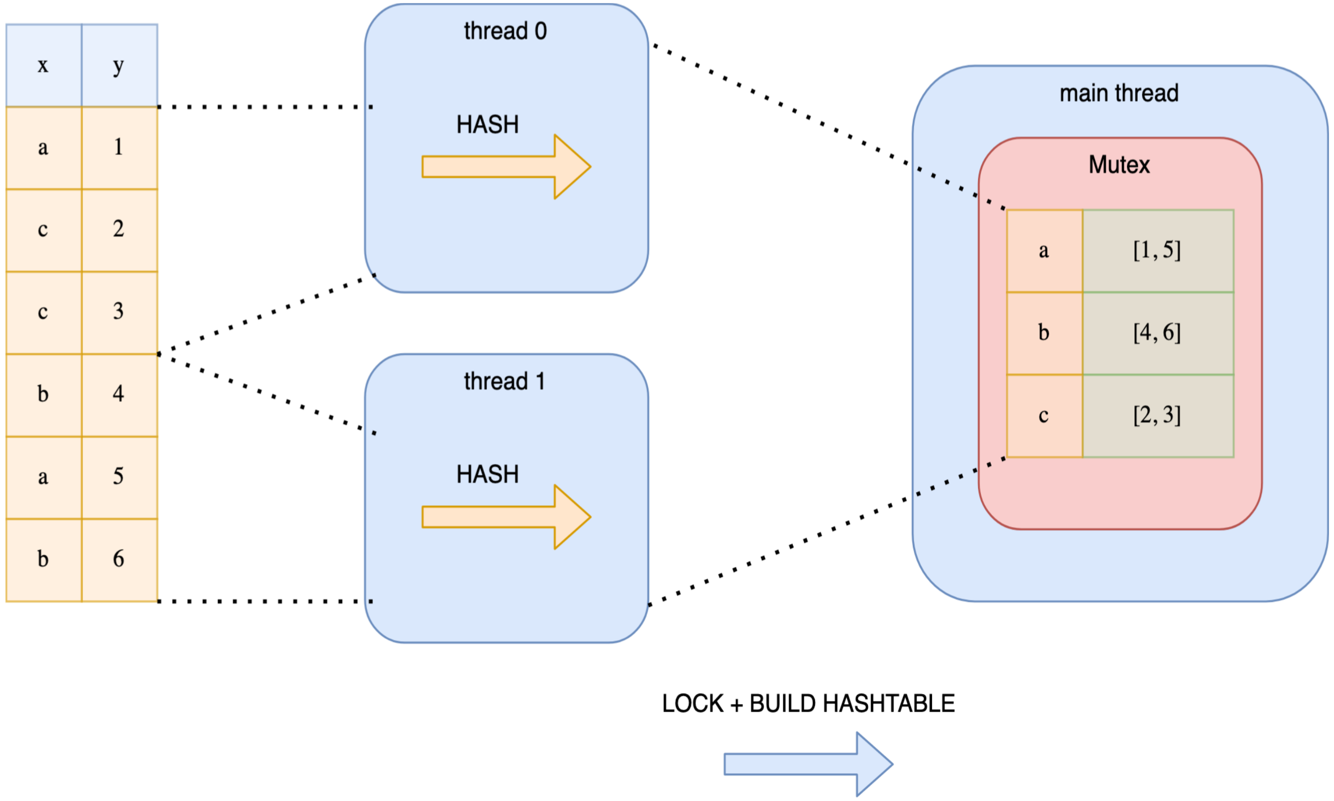
- Data is split into threads
- Each thread applies operation indepdently
- There is no guarantee, that a key doesn't fall into multiple threads
- Makes an **extra synchronoization step necessary**

https://www.ritchievink.com/blog/2021/02/28/i-wrote-one-of-the-fastest-dataframe-libraries/

Prepare data for tasks that require communication

scieneers DRIVEN BY DATA

Idea 2: Split data but allows threads to communicate via mutex is to slow



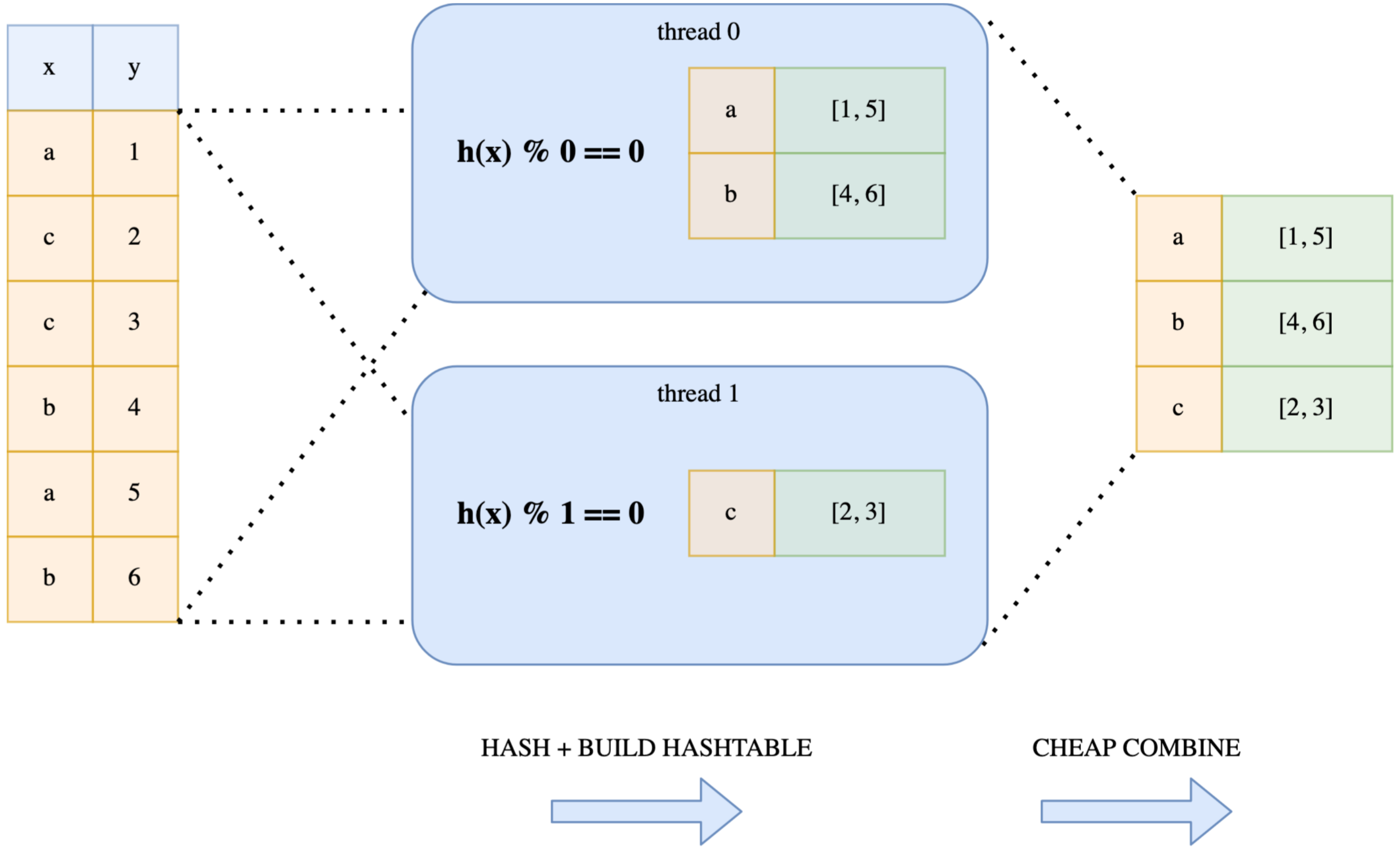
- Data is split into threads
- Threads have a shared storage (**mutex**) to prevent duplicates
- However different threads **block each other** (especially with higher prallelization)

https://www.ritchievink.com/blog/2021/02/28/i-wrote-one-of-the-fastest-dataframe-libraries/

Prepare data for tasks that require communication

scieneers DRIVEN BY DATA

Idea 3: Give threads access to all data, so they can work independently without duplicaters



- All threads load the full data
- Threads **independently decide which values to operate** on by using a modulo function
- Results can be cheaply combined by trivial concatination

https://www.ritchievink.com/blog/2021/02/28/i-wrote-one-of-the-fastest-dataframe-libraries/

The Polars API is very expressive and flexible

scieneers
DRIVEN BY DATA

This makes it fast

1. Internals too far from "the metal"
2. No support for memory-mapped datasets
3. Poor performance in database and file ingest / export
4. Warty missing data support
5. Lack of transparency into memory use, RAM management
6. Weak support for categorical data
7. Complex groupby operations awkward and slow
8. Appending data to a DataFrame tedious and very costly
9. Limited, non-extensible type metadata
10. Eager evaluation model, no query planning
11. "Slow", limited multicore algorithms

But wait, there is more...

scieneers
DRIVEN BY DATA

- Written in Rust
  - Super fast
  - No hard python dependencies

```
➜  ~ pip install polars
Collecting polars
  Downloading polars-0.17.2-cp37-abi3-macosx_10_7_x86_64.whl (16.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 16.3/16.3 MB 18.4 MB/s eta
Requirement already satisfied: typing_extensions>=4.0.1 in ./.pyenv/versi
Installing collected packages: polars
Successfully installed polars-0.17.2
```

- Out-of-Memory Sorting and Deduplicate operations

```python
import polars as pl

from ..paths import DATA_DIR

q11 = (
    pl.scan_csv(f"{DATA_DIR}/reddit.csv")
    .with_columns(pl.col("name").str.to_uppercase())
    .filter(pl.col("comment_karma") > 0)
    .sink_parquet(f"{DATA_DIR}/reddit.parquet")
)
```

- Cheap switching between polars and pandas dataframes thanks to Apache Arrow

```python
%%timeit
df = pdf.to_pandas()
pdf2 = pl.from_pandas(df)
```

```
1.26 ms ± 32.3 µs per loop
```

Polars ticks all of Wes McKinney pandas pain points

Written in Rust Polars can get along with minimal computation and memory footprint

1. Internals too far from "the metal"
2. No support for memory-mapped datasets
3. Poor performance in database and file ingest / export
4. Warty missing data support
5. Lack of transparency into memory use, RAM management
6. Weak support for categorical data
7. Complex groupby operations awkward and slow
8. Appending data to a DataFrame tedious and very costly
9. Limited, non-extensible type metadata
10. Eager evaluation model, no query planning
11. "Slow", limited multicore algorithms

Polars is great for its speed, but can’t replace pandas (yet)

scieneers
DRIVEN BY DATA

My personal list of things I love and miss in polars

Things I love about Polars

- The **speed**!!!
- The support of **eager and lazy mode**
- **Expression API** and **over-keyword**
- That API-code is **nicely structured**

Good first Use-Cases:

- datawrangling pipelines
- Non-trivial feature-engineering

Things I miss in Polars

- **Dot-Notation** to autocomplete column names (especially within notebooks)
- No **Plotting API**
- **Compatibility** with other libraries (scikit-learn, seaborn, pytorch...)
- The **typing efficiency** within the API

Not recommended Use-Cases :

- Data exploration
- Python Glue-Code projects

Link to the slides:

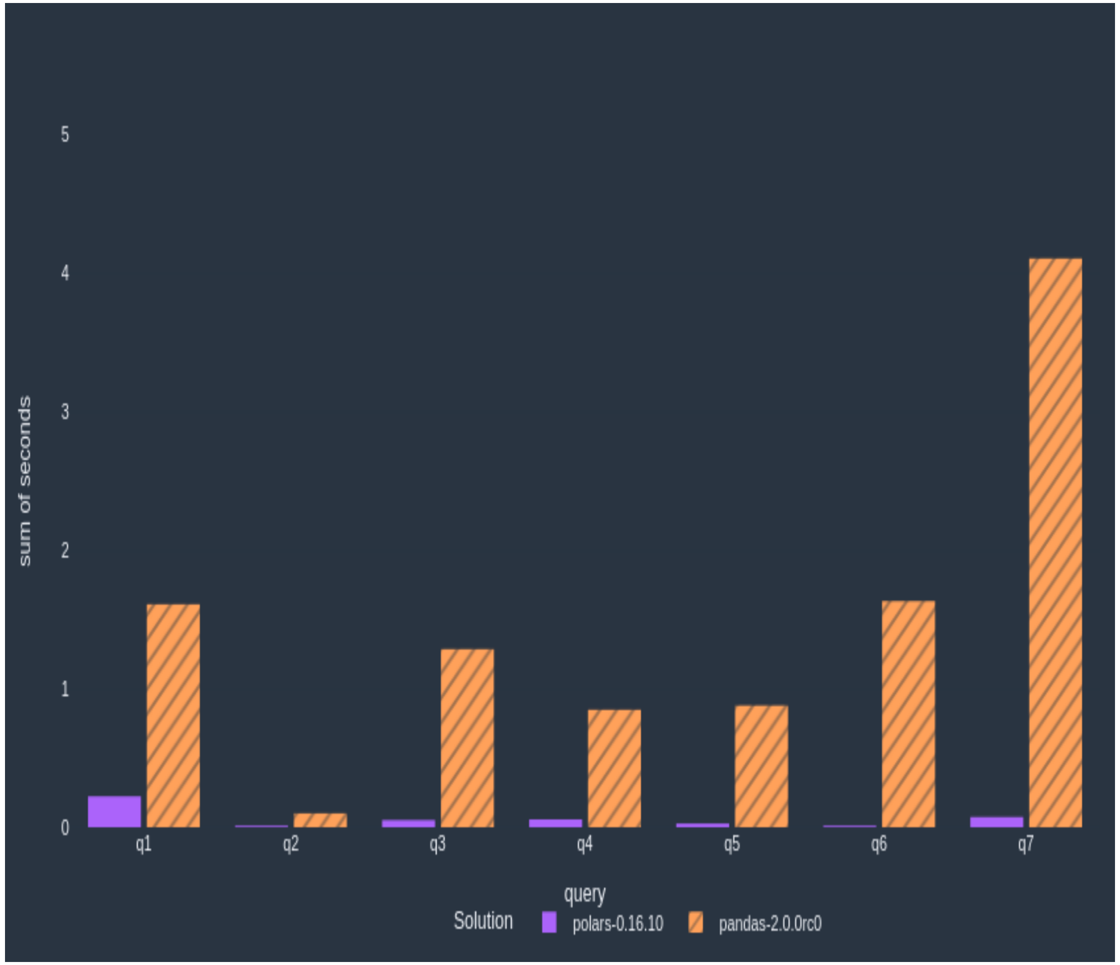
Tip: Next Talk here will also be on Polars and DuckDB

Q&A

Further reading:

- Ritchie Vink giving some speed insights into Polars
- Apache Arrow and the "10 Things I Hate About pandas"
- “Polars in a nutshell” talk by Ritchie Vink
- Comparison of Pandas 2.0 with other frameworks
- Comparison of many popular dataframe libraries

Backup: Speed Comparison Pandas 2.0



https://twitter.com/RitchieVink/status/1632334005264580608

groupby join groupby2014

0.5 GB 5 GB 50 GB

basic questions

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

duckdb-latest*	0.8.0	2023-04-13	76s
Polars	0.16.18	2023-04-05	127s
DuckDB*	0.7.1	2023-04-05	143s
ClickHouse	22.12.1.1752	2023-03-24	189s
data.table	1.14.9	2023-03-24	191s
spark	3.3.2	2023-03-24	389s
Arrow	11.0.0.3	2023-04-12	624s
(py)datatable	1.1.0a0	2023-03-24	870s
pandas	2.0.0	2023-04-07	2015s
dask	2023.3.2	2023-04-07	3990s
Modin		see README	pending

First time
Second time

https://duckdblabs.github.io/db-benchmark/