Polars

An API for fast analytics

Ritchie Vink
Machine Learning Engineer, Xomnia

ORGANIZED BY databricks

About

Ritchie Vink

- Author of Polars
- Background in Machine Learning and Software development
- Part-time open source developer @ Xomnia

Agenda

- Why polars?
- Foundations
- Performance
- Expression API
- Small example

Current DataFrame implementations

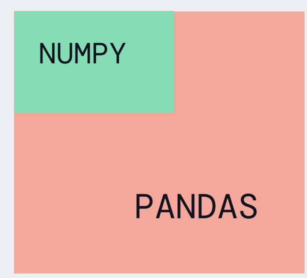
*Python oriented

50+ years of RDMS design are not really applied in the data science community

- Almost all implementation are eager -> no query optimization
- Huge wasteful materializations
- Responsibility on fast/memory efficient compute on user (most users are no OLAP experts)
- Terrible memory representation of string data == terrible performance

Current DataFrame implementations

*Python oriented

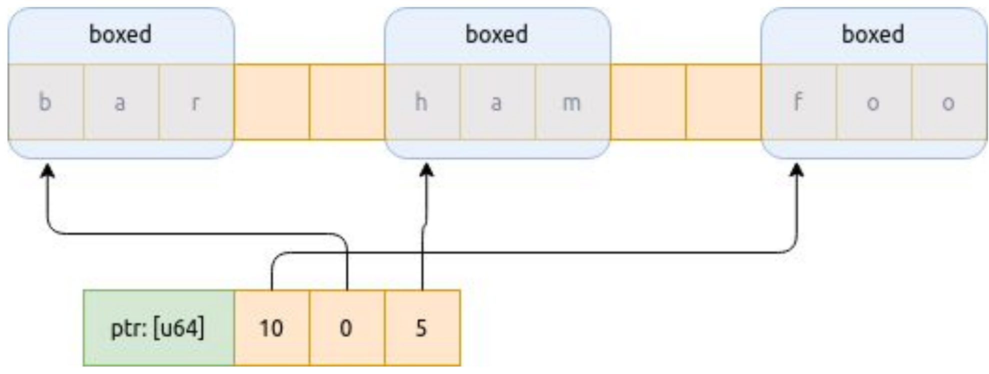


Problem: Numerical compute != Relational compute

- no missing data -> ambiguous use of NaN
- no string data -> boxed python strings -> ptr chasing
- eager evaluation -> no holistic query optimizations
- huge intermediate materializations

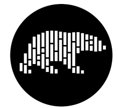
Example: string data

python strings



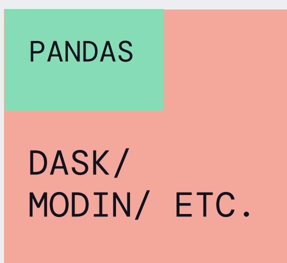
arrow strings

data: [str]	f	o	o	b	a	r	h	a	m
offsets: [i64]	0	2	5	8					
validity bits	01101110101								



Current DataFrame implementations

*Python oriented



Problem: Single threaded

- "simply" throw more CPU power to the problem
  - instead of rethinking the root problem
- All problems inherited from pandas
  - bad memory representation
  - high memory usage
  - can you really control memory?

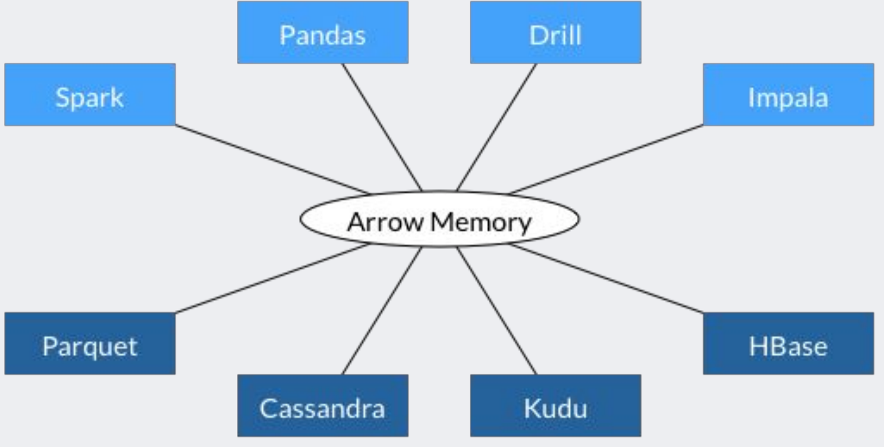
Introducing Polars

DataFrame backend/ query engine written in Rust

- Front end: {Python, Rust, NodeJS}
- Abstraction over arrow memory*
- Vectorized parallel query engine
- Query planning/optimizations
- Powerful expression API*
- COW++ semantics*
- Fast native IO reader/writers

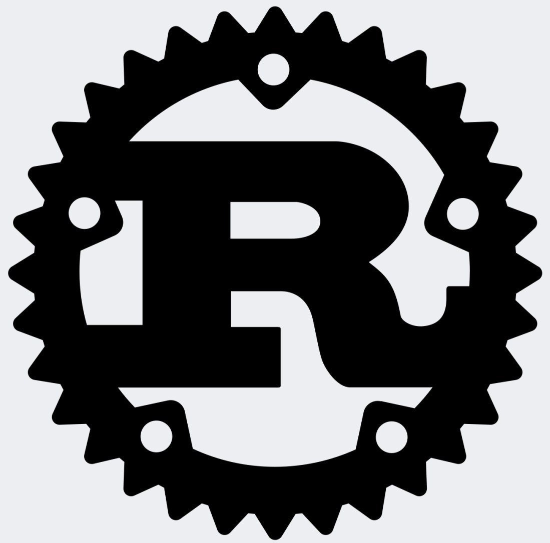
Foundations: Arrow

- The future of (large) data communication
- << serialization/deserialization cost
- Within same process, free ptr sharing
  - (partial) zero copy interop with:
    - pyarrow
    - ray datasets
    - duckdb
    - dremio
    - …
- Arrow2: native Rust implementation



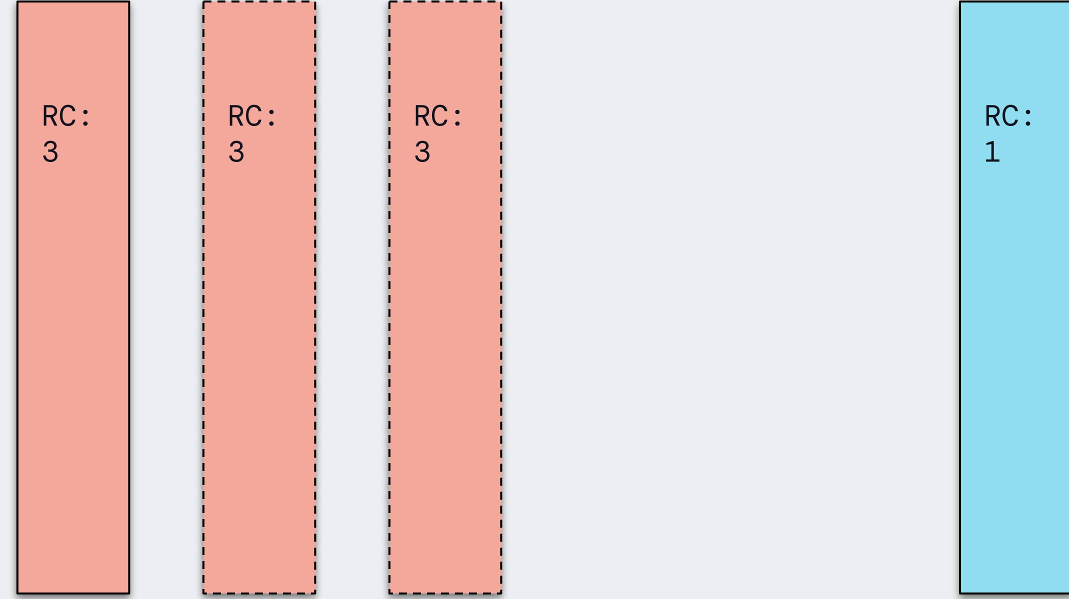
Foundations: Rust

- Fast C/C++
- Borrow checker/ Ownership
  - memory safety guarantees (without losing control)
  - fearless concurrency



Example COW++

- Atomically reference counted
- Rust: No mutable aliases



Polars' performance

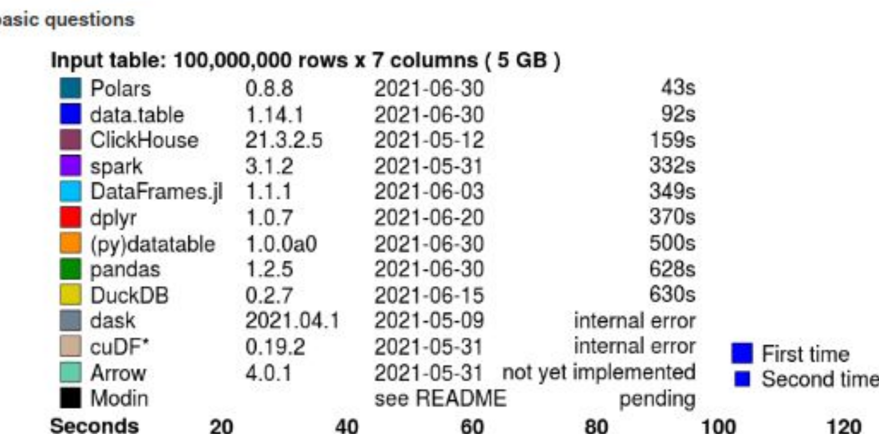
Polars' performance

- Fast cache friendly data-structures and algorithms
- Work stealing parallelism
- In-node parallelism
  - radix groupby
  - partitioned groupby
  - radix join
- SIMD operations
- Query optimizatons
- native IO

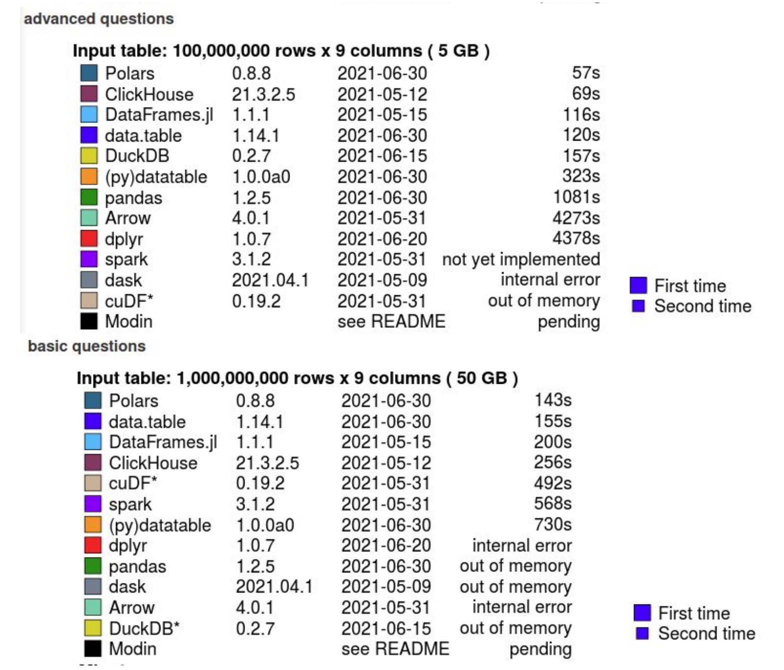
H2OAI's db-benchmark

groupby + join benchmarks

join summary



groupby summary



H2OAI's db-benchmark

Xomnia
Impact with AI

only groupby

Dataframe GroupBy Performance

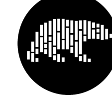
0 200 400 600 800 1000 1200 1400 1600

Engine - Cores - Data Size

Pandas - 1 - 500MB
Modin - 16 - 500MB
Dask - 16 - 500MB
Spark - 16 - 500MB
Bodo - 8 - 500MB
Bodo - 16 - 500MB
Bodo - 126 - 500MB
Bodo - 252 - 500MB
Polars - 16 - 500MB
Datafusion - 32 - 500MB
Pandas - 1 - 5000MB
Modin - 16 - 5000MB
Dask - 16 - 5000MB
Spark - 16 - 5000MB
Bodo - 8 - 5000MB
Bodo - 16 - 5000MB
Bodo - 126 - 5000MB
Bodo - 252 - 5000MB
Polars - 16 - 5000MB
Datafusion - 32 - 5000MB
Bodo - 126 - 50000MB
Bodo - 252 - 50000MB
Polars - 16 - 50000MB
Polars - 32 - 50000MB

Elapse Time (sec)

IO_time_sec grouping time_sec



Polars' Expressions

Expressions

- Strive for a small API surface
  - composable blocks
- Vocabulary of a programming language small AND powerful
- control AND performance

Expressions

```
type Expr = Fn(Series) -> Series

where Series: <column> | <group> | <list el>

// f(g(Series)) = (f º g)(Series)
// lazy
// optimizable
```

Expressions

```
// very composable/flexible
my_expr = pl.col("foo").sort_by("bar") /
pl.col("ham").where(pl.col("spam") > 19).sum()
```

Expressions

```
// very composable/flexible
my_expr = pl.col("foo").sort_by("bar") /
pl.col("ham").where(pl.col("spam") > 19).sum()
```

Expressions

Contexts deal with the expression exectution

- embarrassingly parallel
- cache and optimize expressions
- in-expression parallelism

```
df.select([
    pl.col("foo").rank(),
    pl.when(pl.col("bar") > 10).then(pl.col("foo".first())).otherwise("ham"),
    pl.col("foo").diff().alias("diff_foo")
])
```

Example

DATA+AI SUMMIT 2022

Thank you

Ritchie Vink

- github: https://github.com/pola-rs/polars
- discord: https://discord.com/invite/4UfP5cfBE7