Research Data Engineering with Apache Parquet

Introduction

Hi, I'm Dave Bunten!

With the **Software Engineering Team** at CU Anschutz DBMI

Department of Biomedical Informatics
SCHOOL OF MEDICINE
UNIVERSITY OF COLORADO **ANSCHUTZ MEDICAL CAMPUS**

Visit us for more info: https://cu-dbmi.github.io/set-website/

Gratitude

Big ***thank you*** for attending!

Presentation Outline

1. 😰 Data Challenges
2. 📂 Parquet Format
3. ⏩ The Future

Why?

> “You will never ***find*** time for anything.
> If you want time, you must ***make*** it.”

— Charles Buxton, Notes of Thought (1883)

Why?

🕓😰 Data takes time to store and retrieve from files!

- What is the time cost of the file formats you use today?
- Are we prepared for the future of data needs?
- Can we *"make"* time by using alternatives?

Why?

A theory: we need to act a bit like time travelers in order to steward good research data engineering.

- Time traveling as reducing time durations.
- Time traveling as avoiding the need to reduce time in the future.
- Time traveling as informing those in the future about what's aleady happened.

Why?

This presentation will cover how Apache Parquet can help us be good stewards of research by using less time and enhancing the approaches with data.

First, we'll cover some brief examples of data challenges.

Data Challenges

Col_A	Col_B	Col_C
1	a	0.01
2	b	0.02

A visual example of tabular data.

Data Challenges - An Example

```
1 Col_A,Col_B,Col_C
2 1,a,0.01
3 2,b,0.02
```

A CSV (comma delimited spreadsheet) version of the same table.

Data Challenges - An Example

Strengths of CSV's

- Simple
- Interoperable
- Human-readable

Data Challenges - An Example

```
1 Col_A,Col B,Col_C,COL_D
2 ,a,"0.01"
3 2,null,0.02,{'color':'blue'}
```

But what about more challenging scenarios?

Data Challenges - An Example

Challenges with CSV's

- No data types (implied translation every read)
- Expensive to slice (cols or rows)
- Missing data handling
- 2D dimensionality
- Row-wise orientation (more on this later)

Data Challenges

CSV's are sometimes used because they were the best option available at the time.

They also are used because they're easy to see (and as a result, trust).

Data Challenges

Large data handling often involve working with data values we cannot see (especially from big data V's: volume, variety, and velocity).

(Image: "Kit's Deluge" by Henning Falk, ©2022 NumFOCUS, CC BY 4.0.)

Data Challenges

> “Some things have to be believed to be seen.”

- Madeleine L’Engle, A Wrinkle in Time (1962)

Believe with me for a moment in the promise of Parquet and the data that we cannot directly see.

Data Challenges

Besides believing, there are many measured performance benefits to using formats like Parquet (typically orders of magnitude).

See:

- CSV vs Parquet - Speed up data analytics and wrangling with Parquet files (Posit)
- CSV vs Parquet - Apache Parquet vs. CSV Files (DZone)

Parquet - Definition

“Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem, regardless of the choice of data processing framework, data model or programming language.”

- Apache Parquet Website

Parquet - History

Some rough history:

- Inspired by earlier work from Google Research in 2010: Dremel: Interactive Analysis of Web-Scale Datasets.
- Originally introduced by Twitter in collaboration with Cloudera in July 2013 (link).
- Parquet joined the Apache Software Foundation in 2015 as a Top-Level Project (TLP) (link)

Parquet - As a file

```
1 file.parquet (unable to preview with a text editor)
```

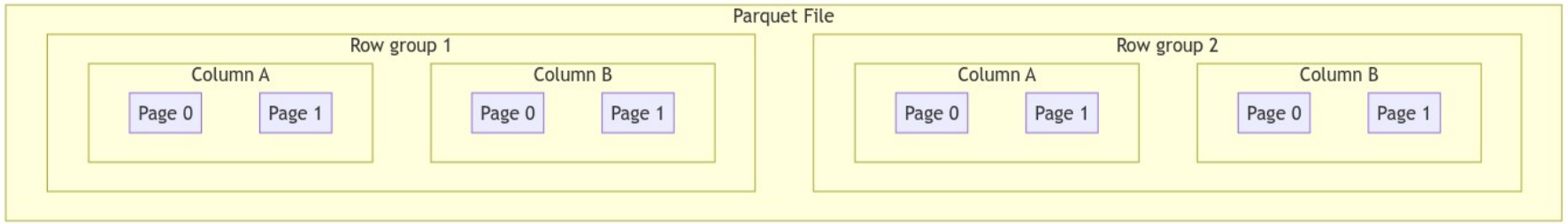
Parquet file of the table we saw earlier.

Parquet - "Columnar data"

Data Type	Data	How data is organized on file
CSV	1 Grapes 2 Oranges	1 Grapes 2 Oranges
Parquet	1 Grapes 2 Oranges	1 2 Grapes Oranges (simplification)

What does *"**columnar data**"* mean?

Parquet - Internals



Parquet organizes column values into **pages** inside of **row groups** (link).

Parquet - Notable Features

“Okay already, show me why any of this matters!”

(***How can I use this today?!***)

Parquet - "Strongly-typed" Data

```
1 Col_A,Col B,Col_C,COL_D
2 ,a,"0.01"
3 2,null,0.02,{'color':'blue'}
```

Have you ever experienced mixed data types in your data?

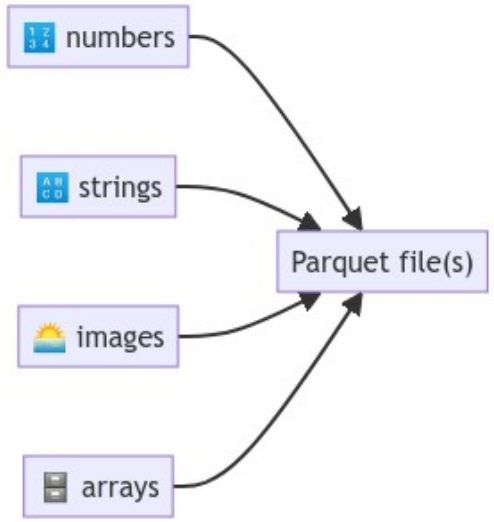
These are challenging (and expensive) to resolve!

Parquet - "Strongly-typed" Data

```python
import pyarrow as pa
from pyarrow import parquet

# create a pyarrow table
table = pa.Table.from_pydict(
    {
        "A": [1, 2, 3],
        "B": ["foo", "bar", 1],
        "C": [0.1, 0.2, 0.3],
    }
)

# write the pyarrow table to a parquet file
parquet.write_table(table=table, where="example.parquet")

# raises exception:
# ArrowTypeError: Expected bytes, got a 'int' object (for column B)
# Note: while this is an Arrow in-memory data exception, it also
# prevents us from attempting to perform incompatible operations
```

Data within Parquet is considered "strongly-typed", entailing specific data types (such as integer, string, etc.) associated with each column and value.

Parquet - Complex data handling



Parquet files may store many data types that are complicated or impossible to store in other formats.

Parquet - Complex data handling

```python
import pyarrow as pa
from pyarrow import parquet

# create a pyarrow table with complex data types
table = pa.Table.from_pydict(
    {
        "A": [{"key1": "val1"}, {"key2": "val2"}],
        "B": [[1, 2], [3, 4]],
        "C": [
            bytearray("😊".encode("utf-8")),
            bytearray("🌻".encode("utf-8")),
        ],
    }
)

# write the pyarrow table to a parquet file
parquet.write_table(table=table, where="example.parquet")

# read the schema of the parquet file
```

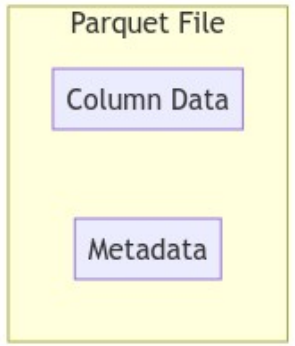
Parquet can handle these complex types using various means.

Parquet - Compression

```python
import os
import pyarrow as pa
from pyarrow import parquet

# create a pyarrow table
table = pa.Table.from_pydict(
    {
        "A": [1, 2, 3, 4, 5],
        "B": ["foo", "bar", "baz", "qux", "quux"],
        "C": [0.1, 0.2, 0.3, 0.4, 0.5],
    }
)

# Write Parquet file with Snappy compression
parquet.write_table(table=table,
  where="example.snappy.parquet",
  compression="SNAPPY"
)
```

Parquet files may leverage compression to help reduce file size and increase data performance.

Parquet - Metadata as “first-class” object



The Parquet format treats data about the data (metadata) distinctly from that of column value data (link).

Parquet - Metadata as "first-class" object

```python
import pyarrow as pa
from pyarrow import parquet

# create a pyarrow table
table = pa.Table.from_pydict(
    {
        "A": [1, 2, 3],
        "B": ["foo", "bar", "baz"],
        "C": [0.1, 0.2, 0.3],
    }
)

# add custom metadata to table
table = table.replace_schema_metadata(metadata={"data-producer": "CU DBMI S

# write the pyarrow table to a parquet file
parquet.write_table(table=table, where="example.snappy.parquet", compressio

# read the schema
```

We can customize metadata inside of a Parquet file so that the files always include certain information.

Parquet - Multi-file “datasets”

```
parquet_datasets/
├── dataset0/
│   ├── part-0.parquet
│   ├── part-1.parquet
│   └── ...
├── dataset1/
│   ├── part-0.parquet
│   ├── part-1.parquet
│   └── ...
└── ...
```

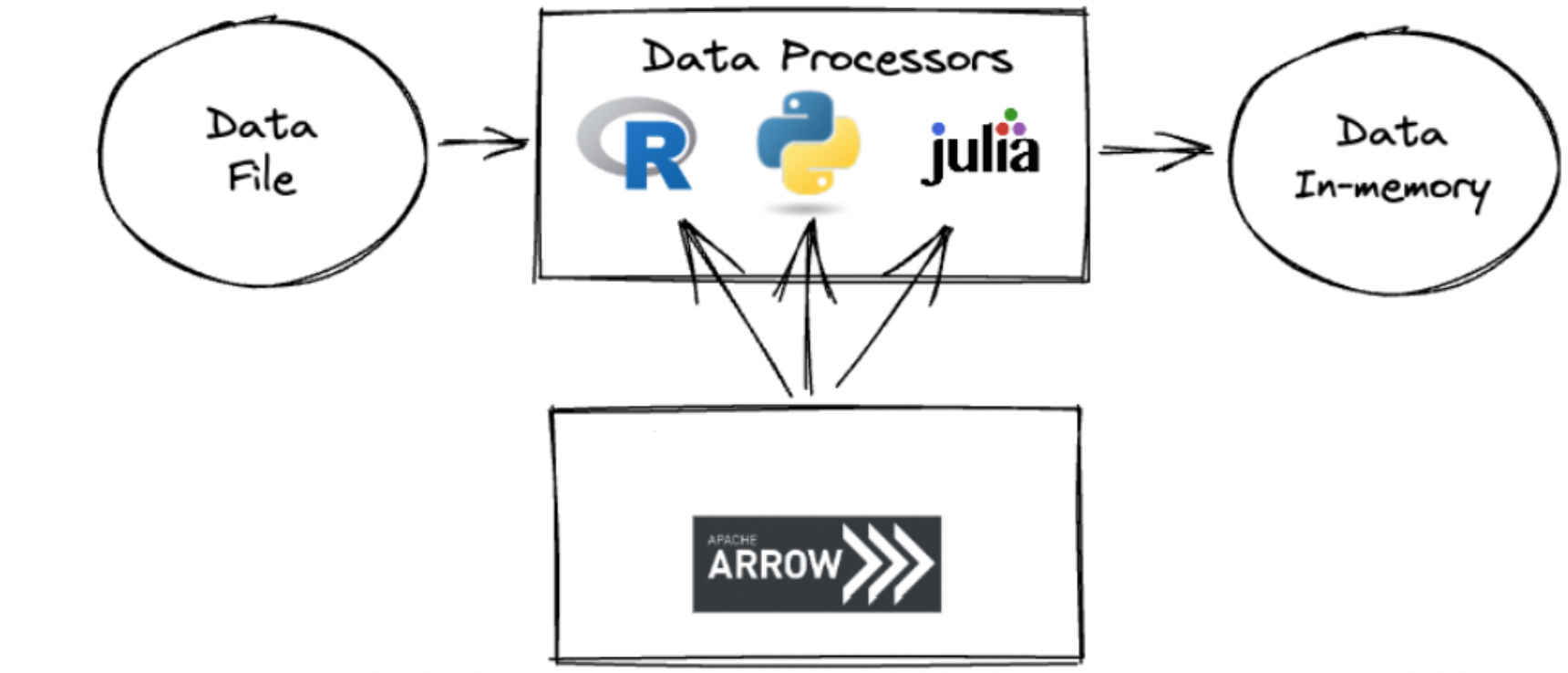
Parquet files may be used individually or treated as a “dataset” through file groups which include the same schema (column names and types).

Parquet - Multi-file "datasets"

```
import pathlib
import pyarrow as pa
from pyarrow import parquet

pathlib.Path("./dataset").mkdir(exist_ok=True)

# create pyarrow tables
table_1 = pa.Table.from_pydict({"A": [1]})
table_2 = pa.Table.from_pydict({"A": [2, 3]})

# write the pyarrow table to parquet files
parquet.write_table(table=table_1, where="./dataset/example_1.parquet")
parquet.write_table(table=table_2, where="./dataset/example_2.parquet")

# read the parquet dataset
print(parquet.ParquetDataset("./dataset").read())

# prints (note that, for ex., [1] is a row group of column A)
# pyarrow.Table
```

Parquet datasets may be read using directory paths (or lists of individual files).

Parquet - Arrow cross-lingual interchange



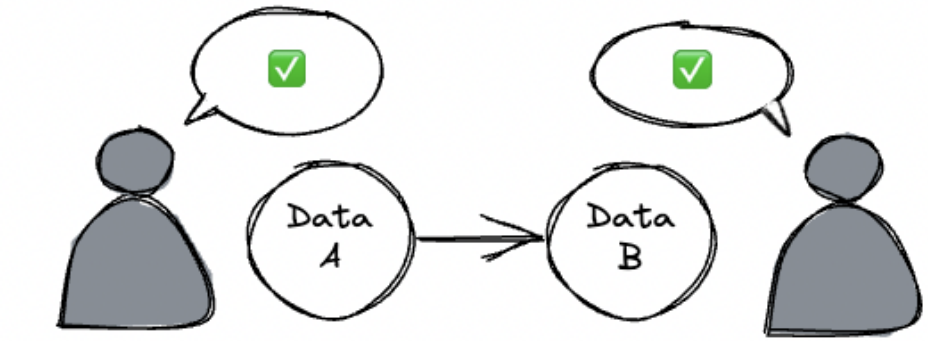
The Parquet format has robust support and integration with the Apache Arrow memory format, enabling consistent multi-lingual interchange.

Parquet - Arrow cross-lingual interchange

```python
import pathlib
import pyarrow as pa
from pyarrow import parquet

# create a pyarrow table
table = pa.Table.from_pydict(
    {
        "A": [1, 2, 3],
        "B": ["foo", "bar", "baz"],
        "C": [0.1, 0.2, 0.3],
    }
)

# write the pyarrow table to a parquet file
parquet.write_table(table=table, where="example.parquet")

# show schema of table and parquet file
print(table.schema.types)
print(parquet.read_schema("example.parquet").types)
```

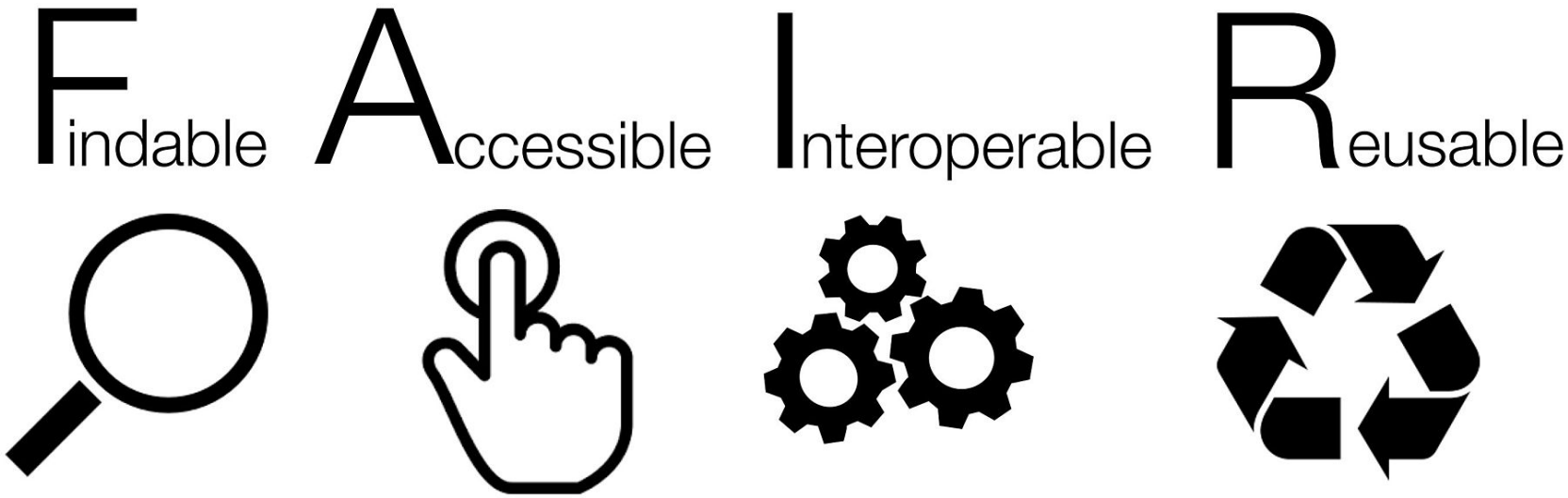
We can consistently read from Parquet files using Arrow to unify how in-memory data are treated.

Parquet - Arrow cross-lingual interchange



- Cross-lingual implementation means we have more people participating in the same “data conversations”.
- It also decouples us from the limitations of “one language”.

Parquet - FAIR data



- These same principles can help enable FAIR data practices.

(Image: "FAIR guiding principles for data resources" by SangyaPundir, 2016, Wikimedia Commons, CC BY-SA 4.0 DEED)

How can you use Parquet?

Below are a list of just a few popular places where you can use Parquet.

- **Python**
  - Pandas (`pd.DataFrame.to_parquet()`, `pd.read_parquet()`)
  - Apache Spark (Spark SQL Guide: Parquet Files)
  - PyTorch (`ParquetDataFrameLoader`)
  - PyArrow (PyArrow: Reading and Writing the Apache Parquet Format)

How can you use Parquet?

Below are a list of just a few popular places where you can use Parquet.

- **R**
  - dplyr (Arrow: Working with Arrow Datasets and dplyr)
  - Arrow (`write_parquet()`, `read_parquet()`)
  - DuckDB (DuckDB R API, DuckDB: Reading and Writing Parquet Files)

You might also be interested in...

- **Apache Arrow** - an in-memory format which enables high-performance and unified memory handling across multiple languages. (Apache Arrow documentation)
- **DuckDB** - which enables SQL queries over local or remote Parquet files. (DuckDB documentation, CU-DBMI blog post)
- **Ibis** - Pythonic API for Parquet (and many other format) data handling (Ibis documentation)

Thank you!

Thank you for attending! Questions / comments?

Please don’t hesitate to reach out!

- </> CU Anschutz DBMI SET Team
- @d33bs