

#ExploreSAS

# SAS EXPLORE

Level Up Your Skills in AI and Analytics



# Arrow/Parquet in SAS Compute

Jerry Adair

Principal Developer, Compute Services, SAS



## Apache Software Foundation (ASF)

- Where it begins



## Arrow/Parquet

- Overview
- Libraries



## Open Source File Formats (OSFF)

- Background @ SAS



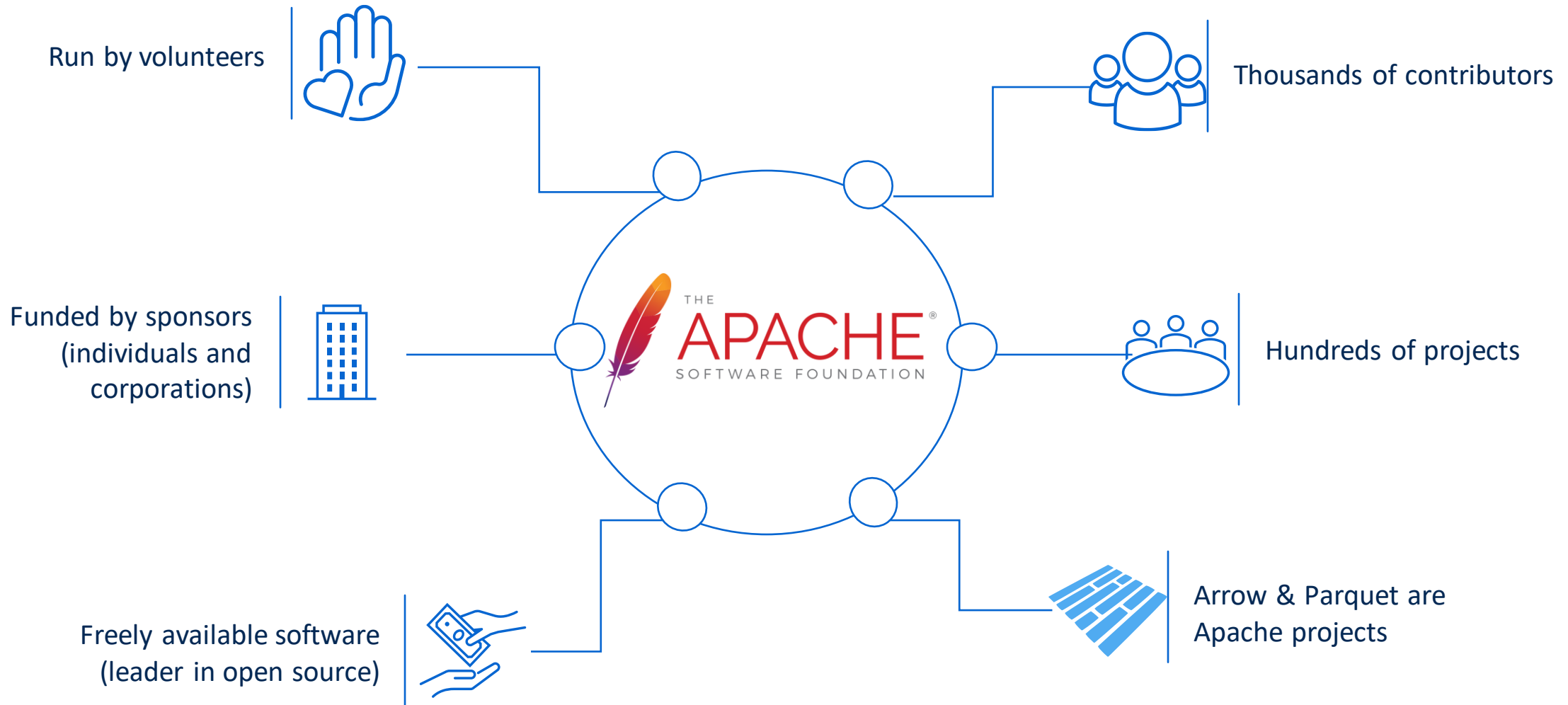
## Compute Parquet

- Current feature set
- Future development



The Apache Software Foundation (ASF) exists to provide software for the public good. We believe in the power of community over code, known as The Apache Way.”

Apache Software Foundation (ASF)  
ASF website



# Arrow vs Parquet

Arrow is a library that provides columnar data structures for in-memory computing. The Arrow format was designed for high-speed operation by vectorized computational kernels.

Parquet is a columnar on-disk file format designed for space and I/O efficiency, though at the expense of CPU utilization.

They are intended to be compatible and can be used together, depending on the use case scenario. A Parquet file can be read into Arrow data structures. That data can then be utilized during in-memory processing.

- 1 Arrow and Parquet are complementary technologies
- 2 Designed with differing purposes, to perform different tasks
- 3 A Parquet read requires decompression and decoding of data
- 4 Arrow has an on-disk format, but is less efficient than Parquet
- 5 Parquet does not offer data structures for in-memory computing
- 6 Arrow random access is  $O(1)$ , Parquet random access is costly



---

An open source, column-oriented data file format

---

Designed for efficient data storage and retrieval

---

Efficient data compression and encoding schemes

---

Enhanced performance for complex data in bulk

---

Language libraries (C++, Java, Python)

---

Originally designed by the creator of Hadoop

---

First release July 2013 (Apache Parquet 1.0)

---



# Open Source File Formats (OSFF)

Background @ SAS

*Team OSFF  
Circa 2019*



Apache  
orc™



# Open Source File Formats (OSFF)

Background @ SAS



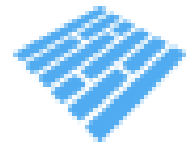
Google  
BigQuery



**READ ONLY**

# Open Source File Formats (OSFF)

Background @ SAS



**Parquet**

*Compute Services  
Circa 2022*



## Compute Parquet

---

Has the most support @ SAS  
Key features delivered  
Developing new features

## Compute ORC

---

Well supported  
Not currently under development

## Compute Avro

---

Support for read only  
Also not currently being developed

# Compute Parquet Features

## Present Capabilities



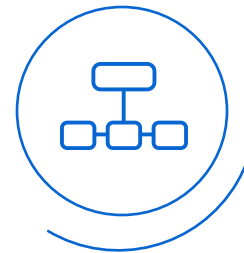
### Read/Write

- Local disk
- Cloud (GCS/S3)
- File/column compression (write)



### Logical Types

- Temporal, String
- Numeric, Embedded
- Nested types unsupported

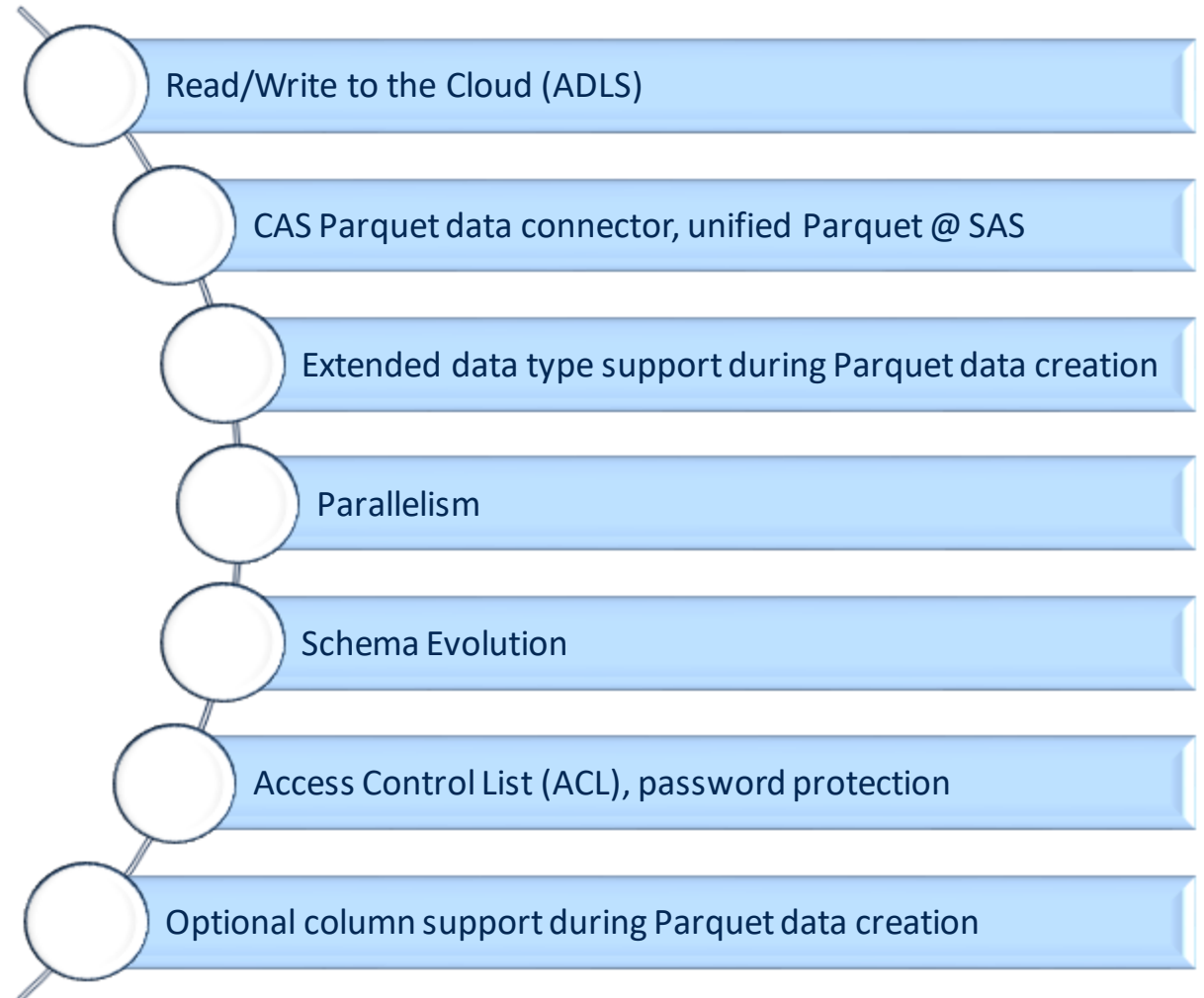


### Partitioned Datasets

- Directories as data
- Parquet partitioning

# Compute Parquet Features

## Future Development



# Compute Parquet

## Further Information



Apache Software Foundation (ASF)  
<https://projects.apache.org/>  
<https://projects.apache.org/project.html?parquet>

SAS Documentation  
<https://documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=default&docsetId=enghdff&docsetTarget=titlepage.htm>



Jerry Adair  
[jerry.adair@sas.com](mailto:jerry.adair@sas.com)

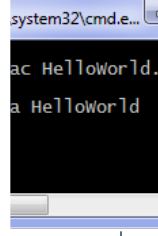


# Parquet Logical Types

## And Primitives Too



Boolean  
Int32  
Int64  
Int96 (deprecated)  
Float  
Double  
Byte Array  
Fixed-Length Byte Array



STRING TYPES  
String  
Enum  
UUID



NUMERIC TYPES  
Signed Integers  
Unsigned Integers  
Decimal



TEMPORAL TYPES  
Date  
Time  
Timestamp  
Interval



EMBEDDED TYPES  
JSON  
BSON



NESTED TYPES  
Lists  
Maps



UNKNOWN  
(always NULL)





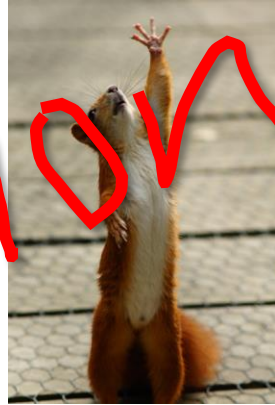
# Parquet Logical Types

## What Are They?



- Primitive
- Logical

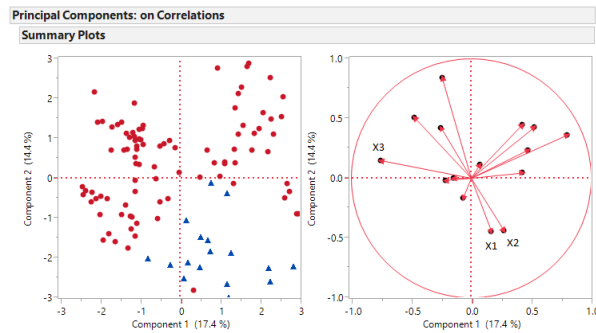
Annotation



Logical extends primitive



Logical stored as primitive  
(DATE = Int32)



Interpretation drives result



- ✓ Minimized set of types
- ✓ Reuses efficient primitive encodings

# Parquet Logical Types

Ain't No School Like The Old School



Primitive

Logical



Converted Type

- Older representation of logical type annotations
- Flattened enum structure like ORC
- Cumbersome to add type parameters
- Replaced by the Logical type
- Now deprecated – best practice is to only write Logical types
- Backward compatibility with old Parquet readers/writers

PM: We support old files containing converted types



# Parquet Logical Types

## Temporal Logicals



### DATE

- Days since UNIX Epoch
- No representation of time
- Stored as Int32
- SAS numeric, DATE9. format



### TIME

- Since midnight, without a date
- Millis, Micros, Nanos precision
- Can be adjusted to UTC
- Millis stored as Int32
- Micros, Nanos stored as Int64
- SAS numeric, TIME8. format
- No UTC adjustment in SAS

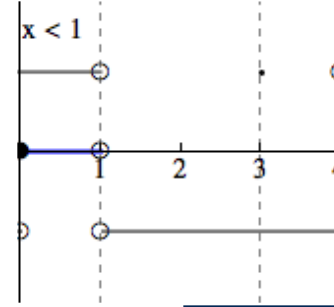
# Parquet Logical Types

## Temporal Logicals

	C
2016	12/2/2
0 AM	10:45:00

### TIMESTAMP

- Time since UNIX Epoch
- Millis, Micros, Nanos precision
- Can be adjusted to UTC
- Millis, Micros, Nanos stored as Int64
- SAS numeric, DATETIME21.2 format
- No UTC adjustment in SAS



### INTERVAL

- Time span between two points
- Stored as 12-byte fixed length byte array
- 4-byte months, 4-byte days, 4-byte milliseconds
- Each 4-byte component is independent
- SAS character, \$char12. format





# The Temporal Problem

## A Funny Thing About Time

### SAS

- Only 2 types: character and numeric
- Time-related columns are numeric, annotated by format/informat
- 3 time-related formats/informats: DATE, TIME or DATETIME

### ORC

- 19 types in flattened type structure (long-ish enumeration)
- Only 2 time-related types: DATE and TIMESTAMP

### Parquet

- 23 types in different classifications/representations
- Primitive versus logical types (union of structs)
- 4 time-related types: DATE, TIME, TIMESTAMP, INTERVAL

# Parquet Logical Types

## String Logicals



### STRING

- Annotates the binary primitive type
- The byte array is interpreted as a UTF-8 encoded character string
- The sort order used for STRING strings is unsigned byte-wise comparison

blue  
green  
yellow  
block

### ENUM

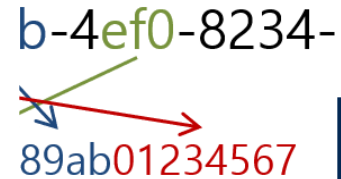
- Annotates the binary primitive type
- The sort order used for ENUM values is unsigned byte-wise comparison

# Parquet Logical Types

## String Logicals

MAC Addr -

b-4ef0-8234-  
89ab01234567



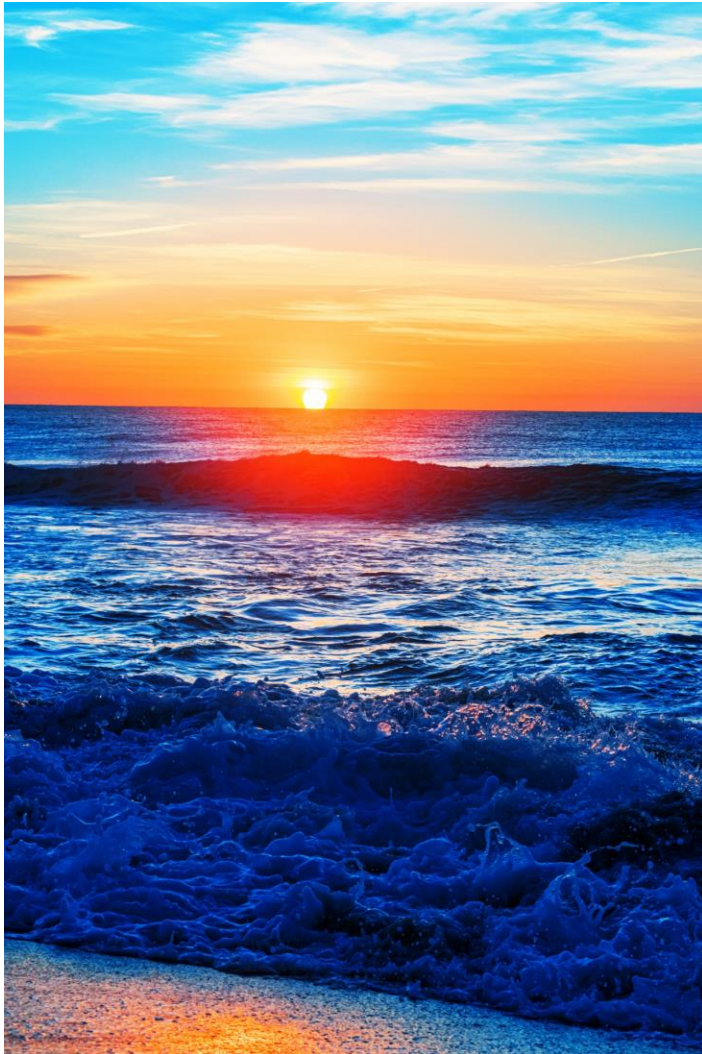
UUID

- Universally Unique Identifier
- 128-bit label to uniquely identify an entity
- Annotates a 16-byte fixed-length binary in Parquet
- The sort order used for UUID values is unsigned byte-wise comparison



# The Problem Of Creation

Just How Are You Gonna Do That?



## SAS

- Only 2 types: character and numeric

## Parquet

- 23 types in different classifications/representations
- Primitive versus logical types (union of structs)
- 3 string-related types: STRING, ENUM, UUID

## The Issue

- Can create only a subset of Parquet logical column types from within a traditional SAS program
- Develop support for Parquet types during data creation

# Parquet Logical Types

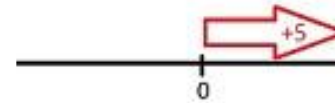
## Numeric Logicals

7) - (-3)

### SIGNED INTEGERS

- INT8, INT16, INT32 annotate INT32 primitive type
- INT64 annotates INT64 primitive type
- Values larger than maximum allowed by the annotation = undefined behavior

i) = (+10)

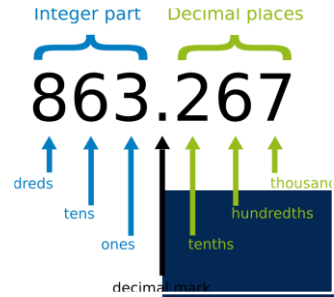


### UNSIGNED INTEGERS

- UINT8, UINT16, UINT32 annotate INT32 primitive type
- UINT64 annotates INT64 primitive type
- Values larger than maximum allowed by the annotation = undefined behavior

# Parquet Logical Types

## Numeric Logicals



## DECIMAL

- Annotated as INT32 or INT64
- INT32  $\leq$  9 digits of precision
- INT64  $\leq$  18 digits of precision
- Primitive type stores an unscaled integer value
- $0 \leq \text{scale} < \text{precision}$  (default = 0)

# The Problem Of Creation

Just How Are You Gonna Do That?



## SAS

- Only 2 types: character and numeric
- 4. -> INT8      5. -> UINT8      6. -> INT16      7. -> UINT16
- 11. -> INT32    12. -> UINT32    20. -> INT64    21. -> UINT64
- ~~DECIMAL~~

## Parquet

- 23 types in different classifications/representations
- Primitive versus logical types (union of structs)
- 9 numeric-related types: INT8-64, UINT8-64, DECIMAL

## The Issue

- SAS program creates only a subset of Parquet logical column types

# Parquet Logical Types

## EMBEDDED LOGICALS



JSON

- Lightweight data-interchange format
- Easy for humans to read and write
- Embedded JSON document
- Annotates byte array primitive type

```
4 05 06 07 08 09 0A 0B 0C 0D
0 77 69 6C 6C 20 62 65 20 61
2 79 74 65 20 73 74 72 65 61
3 48 65 61 64 65 72 00 4E 00
8 65 61 64 65 72 31 00 21 00
5 00 05 00 00 00 42 6F 6E 64
E 73 65 00 07 00 00 00 00 03
4 65 72 32 00 10 00 00 00 08
6 65 00 01 00 00 0A 50 61 79
```

BSON

- Binary format
- Zero or more ordered key/value pairs = a single entity
- Each entity = *document*
- Embedded BSON document
- Annotates byte array primitive type

# Parquet Logical Types

## NULL LOGICAL

**null**

could be assigned

typeof object

undefined === null //false

NULL

- The entire data in a given column is NULL
- Can annotate any primitive type
- Use case: writing data from loosely typed systems to Parquet
- Type cannot be inferred if all data in a column is NULL
- The NULL logical type can be utilized
- If schema evolution is supported, type can be developed later



# The Problem Of Creation

JUST HOW ARE YOU GONNA DO THAT?



## SAS

- Only 2 types: character and numeric
- Can only read embedded logical types (character)
- Same applies to NULL logical type (primitive type dependent)

## Parquet

- 23 types in different classifications/representations
- Primitive versus logical types (union of structs)
- No means of determining which embedded or NULL type is needed

## The Issue

- SAS program creates only a subset of Parquet logical column types



*Team HDFS  
Circa 2019*

# Partitioned Data Background

What are we talking about here?



ORC Format

**Directories AS Data**





Parquet

# SAS HDFS Partitioned Data

All In The Folder



All files must possess  
identical schema

>	CS47732					
	DAD		read_test_1.parquet	10/27/2022 1:44 PM	PARQUET File	5 KB
	DIRECTORY		read_test_2.parquet	10/27/2022 1:44 PM	PARQUET File	5 KB

All data files located  
in one folder  
(no other content)

Separate files,  
Independent,  
Disconnected

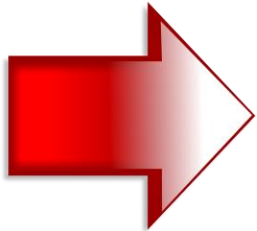
# A Tale Of Two Partitions

Well kinda... sorta...



*ORC/Parquet support one variant of partitioned data*

PM wanted something different







# Parquet Partitioned Data

## What is it? Why does anyone care?



- ▼ yellow\_tripdata\_v1\_0\_brotli
  - ▼ pickup\_hour=0
    - passenger\_count=0.0
    - passenger\_count=1.0
    - passenger\_count=2.0
    - passenger\_count=3.0
    - passenger\_count=4.0
    - passenger\_count=5.0
    - passenger\_count=6.0
  - > pickup\_hour=1
  - > pickup\_hour=2

Data distributed amongst partitions in a folder tree based on key-value pairs (Hive Partitioning)

Unique values in partitioned column	2	5	1000
Query against partitioned column	74.5%	46.3%	16.66%
Query against other columns	89.51%	191.01%	556.99%
Total count query	136.79%	163.68%	1194.88%

Relative elapsed query times of a partitioned parquet filter as a ratio to elapsed query times against a non-partitioned parquet file.

- Queries against a partitioned column improve performance
- Other operations are (much) slower
- The number of unique values impacts performance
- Time required to configure the metadata is longer
- Different partitioning types are available



# Parquet Partitioned Data

## For Example

## Pseudo-code:

```
// Create a table with the following schema:  
// field( "first_name"      )  
// field( "last_name"      )  
// field( "residence_area" )  
  
// AppendValues( { "Tim",    "Fred",    "Wilma", "Joan", "Burt",  
                  "Lance", "Robert", "Lane",  "Karl", "Ted"  } );  
  
// AppendValues( { "Smith",   "Jones", "Johnson", "Myers", "Lenay",  
                  "Richards", "Tuni",  "Peters",  "Zimmer", "Rhodes" } );  
  
// AppendValues( { "City", "City",   "Suburb", "Country", "City",  
                  "City", "Suburb", "Suburb",  "Country", "City"  } );  
  
// Create a partitioned dataset based on the "residence_area" column.
```

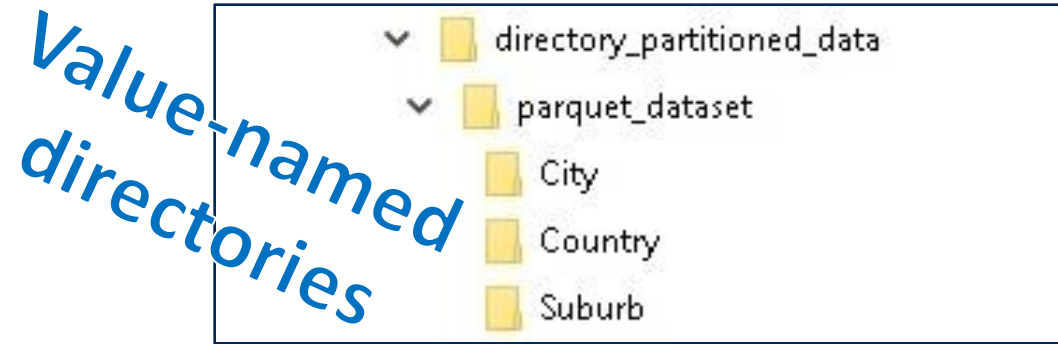
# Parquet Partitioned Data

## The Three Tenors

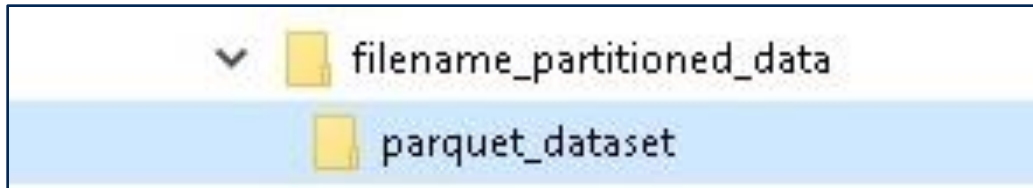
### Hive Partitioning



### Directory Partitioning



### Filename Partitioning



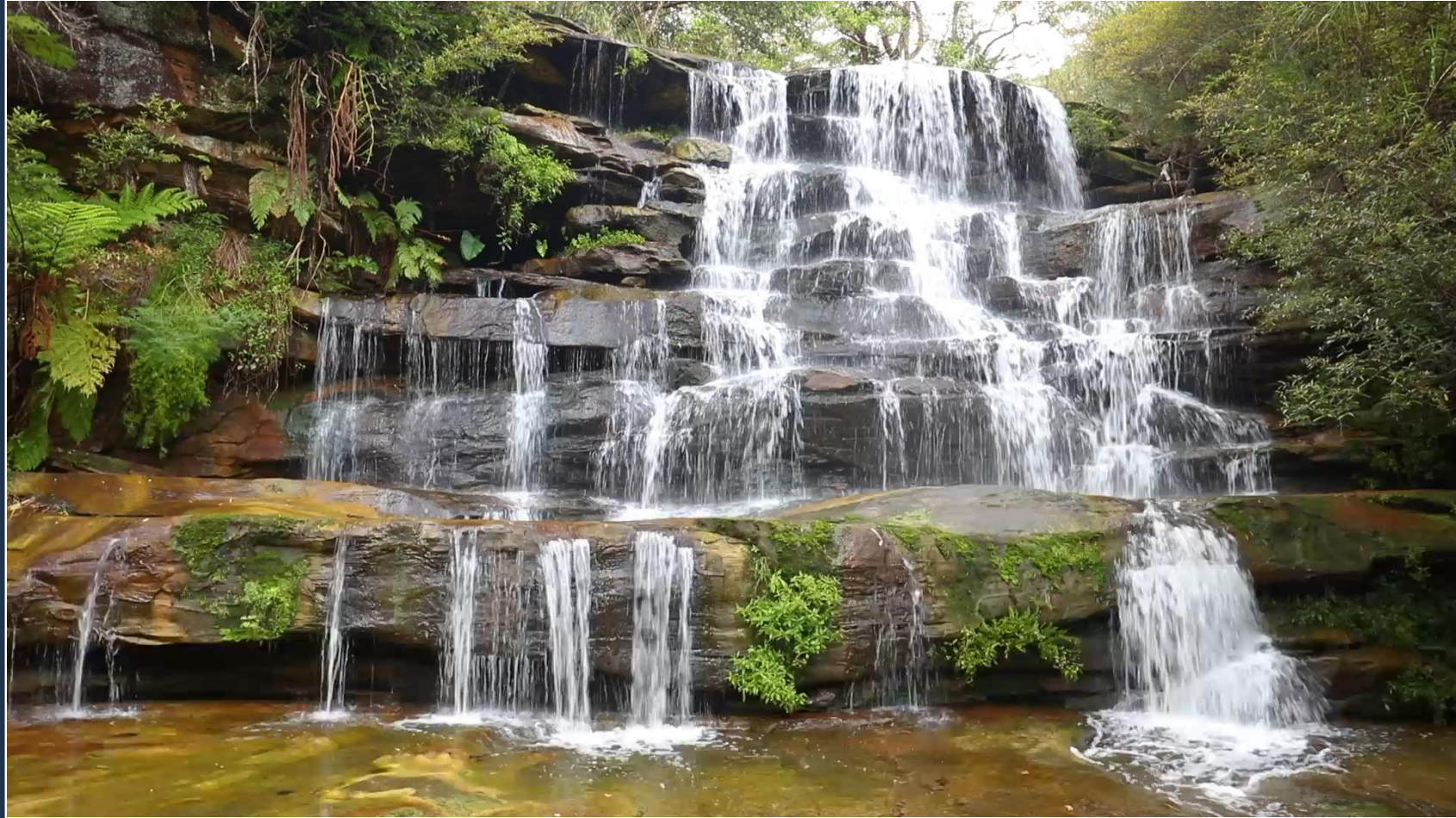
City_part0.parquet	5/15/2023 1:37 PM	PARQUET File	1 KB
Country_part0.parquet	5/15/2023 1:37 PM	PARQUET File	1 KB
Suburb_part0.parquet	5/15/2023 1:37 PM	PARQUET File	1 KB

Value\_part# filenames



# Parquet Partitioned Write

While we would prefer to stream the data to disk in chunks





# Parquet Partitioned Write



There is no means to do that via the Arrow datasets library. Therefore we must fill the buffer and write once.

However I found a path forward. It will require research, design and implementation.

With input from PM we can schedule that work for a future sprint.

# Parquet Partitioned Write

- Arrow was designed to perform data analysis on in-memory tables.
- It is therefore efficient in storing data via its in-memory datasets.
- However, be mindful of your memory footprint when writing Parquet partitioned data.



# ORC vs Parquet

## The Two Columnar Data Stores



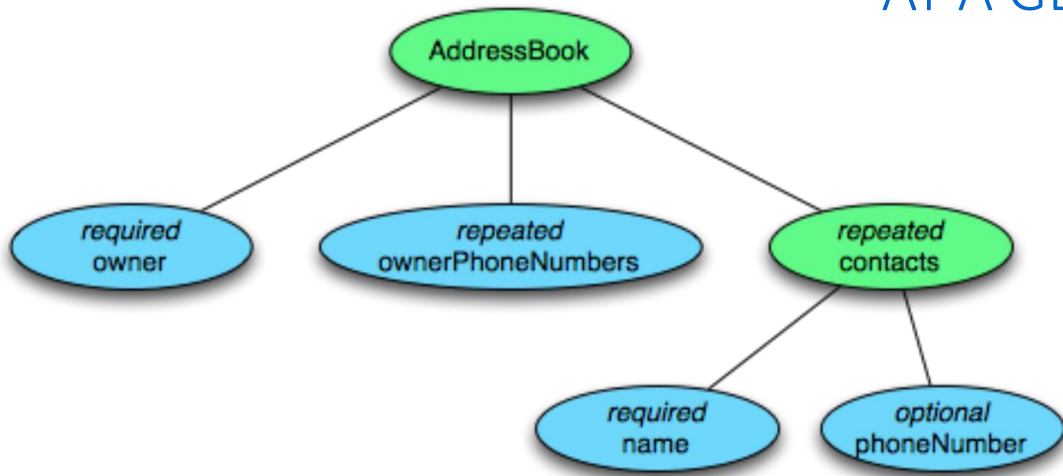
- Inspired from RCFile developed by Facebook
- Development supported by Hortonworks/Facebook
- Supports only Hive and Pig
- Optimized for Hive
- Provides ACID support (Hive)
- Files organized into data stripes
- Better suited for write-heavy operations
- More efficient in writing data

- Inspired by nested data format (Dremel paper)
- Development supported by Cloudera/Twitter
- Broad range of support for Hadoop projects
- Works well with Apache Spark
- Stores data in pages with definition and repetition levels
- Excels in write-once, read-many scenarios
- Highly efficient data compression and decompression
- Supports multiple programming languages
- Columnar compression



# Parquet Model (Dremel Paper)

AT A GLANCE – A QUICK 411



- Minimalistic, columnar storage model in which data structures are used to represent schema
- Nested data = groups of fields
- Repeated data = repeated fields
- Combination of the above eliminates need for complex types such as maps
- Schema root is a group of fields called a message
- Fields have a name, type, and a repetition
- Definition level specifies where the value is NULL, repetition level defines the level at which a new list begins

Column	Max Definition level	Max Repetition level
owner	0 (owner is <i>required</i> )	0 (no repetition)
ownerPhoneNumbers	1	1 ( <i>repeated</i> )
contacts.name	1 (name is <i>required</i> )	1 (contacts is <i>repeated</i> )
contacts.phoneNumber	2 (phoneNumber is <i>optional</i> )	1 (contacts is <i>repeated</i> )

- **required:** exactly one occurrence
- **optional:** 0 or 1 occurrence
- **repeated:** 0 or more occurrences