# Apache Parquet

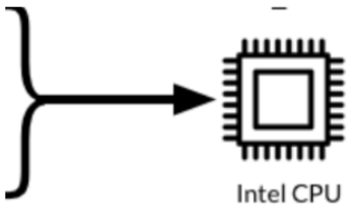
# Apache Parquet

- *Apache Parquet is an binary*
- *Efficient columnar data format*
- uses various techniques to store data in a CPU and I/O efficient way
- row groups, compression for pages in column chunks or dictionary encoding for columns
- Index hints and statistics to quickly skip over chunks of irrelevant data enable efficient queries on large amount of data
- *Supported in Hadoop, Spark, Pandas, AWS, Azure Big Data Platforms*

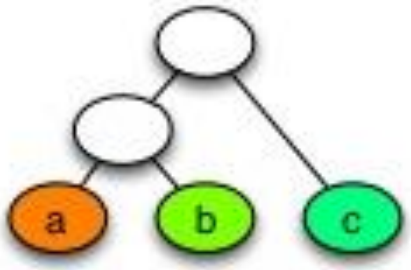
|  | day | location | product | sale |
|---|---|---|---|---|
| row 1 | 2017-01-01 | l1 | p1 | 300 |
| row 2 | 2017-01-01 | l1 | p2 | 40 |
| row 3 | 2017-01-01 | l2 | p1 | 44 |
| row 4 | 2017-02-01 | l1 | p1 | 200 |

| Traditional Memory Buffer |  |
|---|---|
| row 1 | 2017-01-01 |
|  | l1 |
|  | p1 |
|  | 300 |
| row 2 | 2017-01-01 |
|  | l1 |
|  | p2 |
|  | 40 |
| row 3 | 2017-01-01 |
|  | l2 |
|  | p1 |
|  | 44 |

| Columnar Storage |  |
|---|---|
| day | 2017-01-01 |
|  | 2017-01-01 |
|  | 2017-01-01 |
|  | 2017-01-02 |
| location | l1 |
|  | l1 |
|  | l2 |
|  | l1 |
| product | p1 |
|  | p2 |
|  | p1 |
|  | p1 |



# Columnar storage

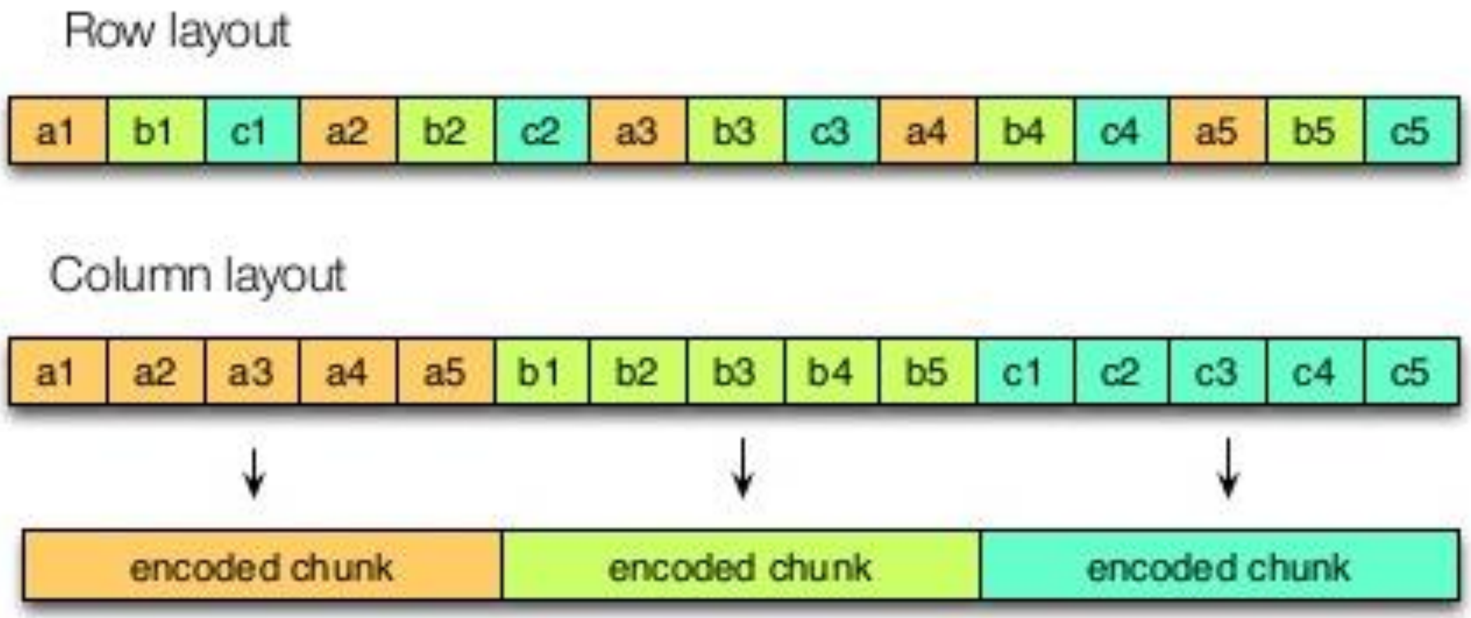


Nested schema

Logical table representation

| a | b | c |
|---|---|---|
| a1 | b1 | c1 |
| a2 | b2 | c2 |
| a3 | b3 | c3 |
| a4 | b4 | c4 |
| a5 | b5 | c5 |

**On Disk:**



Row layout

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a1 | b1 | c1 | a2 | b2 | c2 | a3 | b3 | c3 | a4 | b4 | c4 | a5 | b5 | c5 |

Column layout

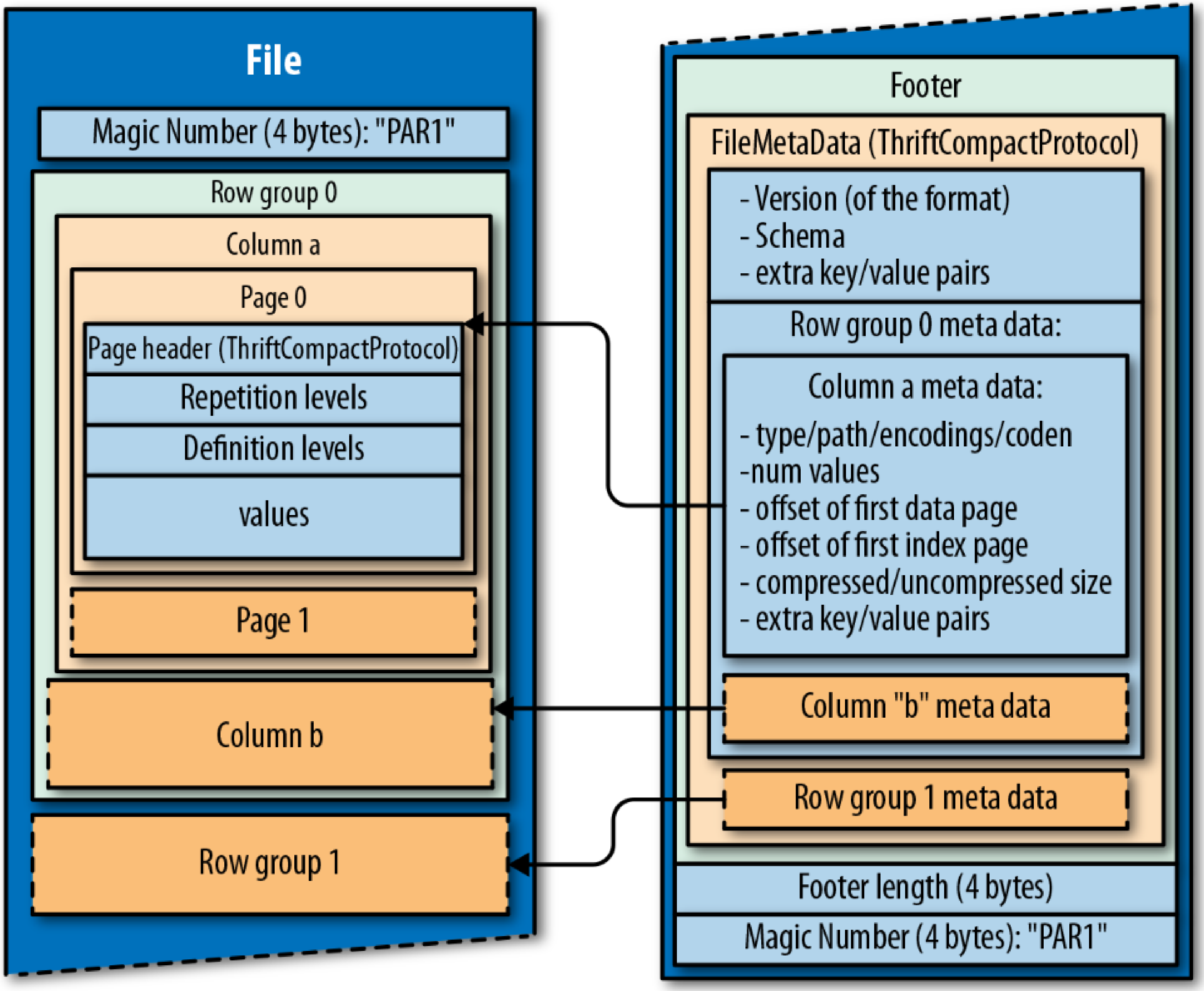
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a1 | a2 | a3 | a4 | a5 | b1 | b2 | b3 | b4 | b5 | c1 | c2 | c3 | c4 | c5 |

| | | |
|---|---|---|
| encoded chunk | encoded chunk | encoded chunk |

Encodings: Dictionary, RLE, Delta, Prefix

Key Concepts

Block Size
Row Group
Page



# Unit of parallelization

- MapReduce - File/Row Group
- IO - Column chunk
- Encoding/Compression - Page

# Compression

- Supports (page) compression and splitting
- Supports nested columns (Dremel encoding)
- Supports minimal number of types

# Glossary

- Block (hdfs block): This means a block in hdfs and the meaning is unchanged for describing this file format. The file format is designed to work well on top of hdfs.

- File: A hdfs file that must include the metadata for the file. It does not need to actually contain the data.

- Row group: A logical horizontal partitioning of the data into rows. There is no physical structure that is guaranteed for a row group. A row group consists of a column chunk for each column in the dataset.

- Column chunk: A chunk of the data for a particular column. These live in a particular row group and is guaranteed to be contiguous in the file.

- Page: Column chunks are divided up into pages. A page is conceptually an indivisible unit (in terms of compression and encoding). There can be multiple page types which is interleaved in a column chunk.

- Hierarchically, a file consists of one or more row groups. A row group contains exactly one column chunk per column. Column chunks contain one or more pages

# File Format

- 4-byte magic number "PAR1"
- <Column 1 Chunk 1 + Column Metadata>
- <Column 2 Chunk 1 + Column Metadata>
- ...
- <Column N Chunk 1 + Column Metadata>
- <Column 1 Chunk 2 + Column Metadata>
- <Column 2 Chunk 2 + Column Metadata>
- ...
- <Column N Chunk 2 + Column Metadata>
- ...
- <Column 1 Chunk M + Column Metadata>
- <Column 2 Chunk M + Column Metadata>
- ...
- <Column N Chunk M + Column Metadata>
- File Metadata
- 4-byte length in bytes of file metadata
- 4-byte magic number "PAR1"

There are N columns in this table, split into M row groups. The file metadata contains the locations of all the column metadata start locations. More details on what is contained in the metadata can be found in the thrift files.

Metadata is written after the data to allow for single pass writing.

Readers are expected to first read the file metadata to find all the column chunks they are interested in

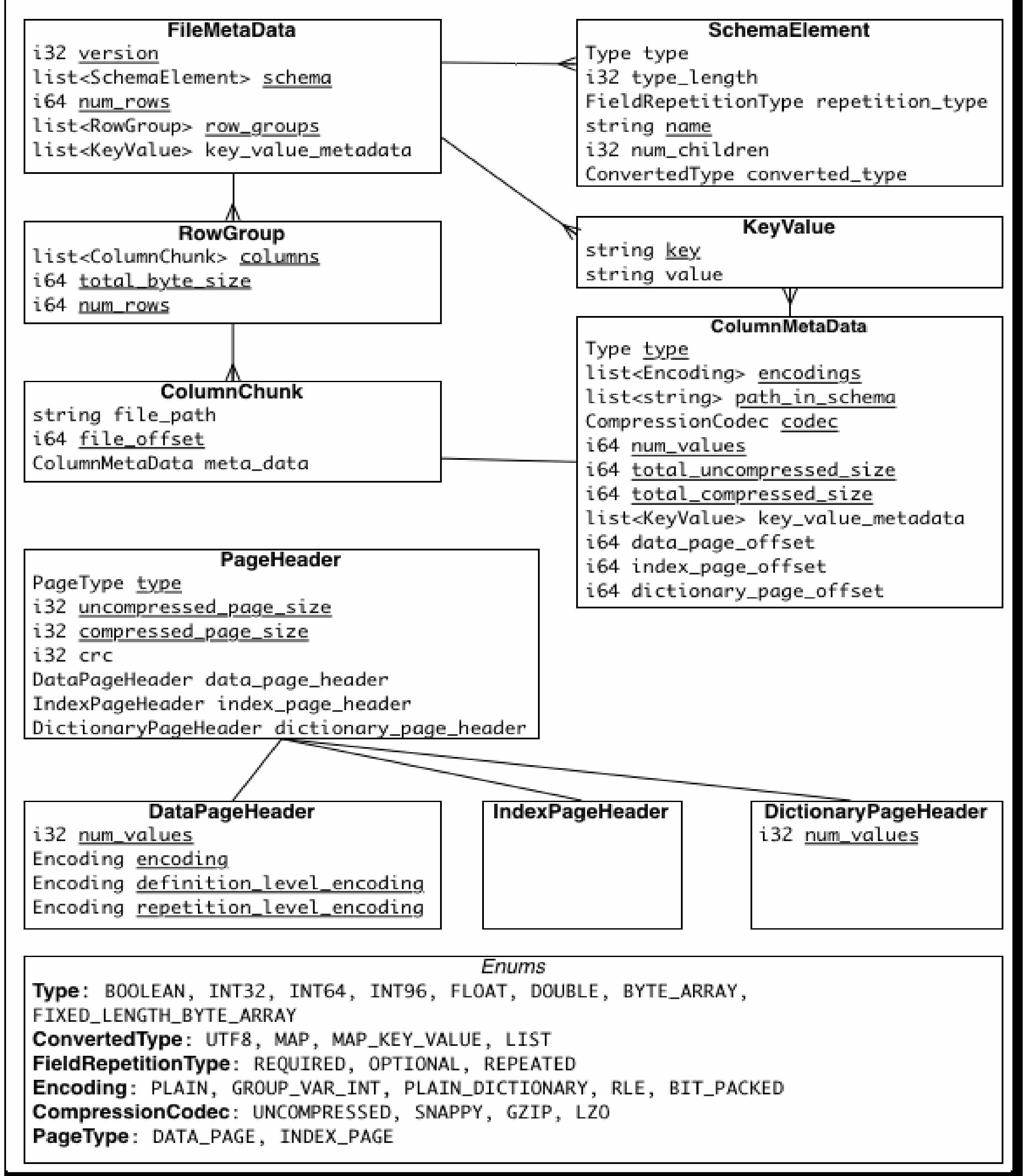
The columns chunks should then be read sequentially

# Meta Data

three types of metadata

- file metadata
- column (chunk) metadata
- page header metadata

All thrift structures are serialized using the TCompactProtocol



**FileMetaData**
i32 version
list<SchemaElement> schema
i64 num_rows
list<RowGroup> row_groups
list<KeyValue> key_value_metadata

**SchemaElement**
Type type
i32 type_length
FieldRepetitionType repetition_type
string name
i32 num_children
ConvertedType converted_type

**RowGroup**
list<ColumnChunk> columns
i64 total_byte_size
i64 num_rows

**KeyValue**
string key
string value

**ColumnChunk**
string file_path
i64 file_offset
ColumnMetaData meta_data

**ColumnMetaData**
Type type
list<Encoding> encodings
list<string> path_in_schema
CompressionCodec codec
i64 num_values
i64 total_uncompressed_size
i64 total_compressed_size
list<KeyValue> key_value_metadata
i64 data_page_offset
i64 index_page_offset
i64 dictionary_page_offset

**PageHeader**
PageType type
i32 uncompressed_page_size
i32 compressed_page_size
i32 crc
DataPageHeader data_page_header
IndexPageHeader index_page_header
DictionaryPageHeader dictionary_page_header

**DataPageHeader**
i32 num_values
Encoding encoding
Encoding definition_level_encoding
Encoding repetition_level_encoding

**IndexPageHeader**

**DictionaryPageHeader**
i32 num_values

*Enums*
**Type**: BOOLEAN, INT32, INT64, INT96, FLOAT, DOUBLE, BYTE_ARRAY, FIXED_LENGTH_BYTE_ARRAY
**ConvertedType**: UTF8, MAP, MAP_KEY_VALUE, LIST
**FieldRepetitionType**: REQUIRED, OPTIONAL, REPEATED
**Encoding**: PLAIN, GROUP_VAR_INT, PLAIN_DICTIONARY, RLE, BIT_PACKED
**CompressionCodec**: UNCOMPRESSED, SNAPPY, GZIP, LZO
**PageType**: DATA_PAGE, INDEX_PAGE

# Types

- BOOLEAN: 1 bit boolean
- INT32: 32 bit signed ints
- INT64: 64 bit signed ints
- INT96: 96 bit signed ints
- FLOAT: IEEE 32-bit floating point values
- DOUBLE: IEEE 64-bit floating point values
- BYTE_ARRAY: arbitrarily long byte arrays.

16-bit ints are not explicitly supported in the storage format
since they are covered by 32-bit ints with an efficient encoding

# Logical Types

- Logical types are used to extend the types that parquet can be used to store
- strings are stored as byte arrays (binary) with a UTF8 annotation
- Annotations are stored as a ConvertedType in the file metadata
- STRING, ENUM, UUID, SIGNED INT, UNSIGNED INT, DECIMAL, DATE, TIME, TIMESTAMP
- For more https://github.com/apache/parquet-format/blob/master/LogicalTypes.md

# Reference

- http://parquet.apache.org/documentation/latest/