Parquet

Columnar storage for the people

Julien Le Dem @J_ Processing tools lead, analytics infrastructure at Twitter

Nong Li nong@cloudera.com Software engineer, Cloudera Impala

http://parquet.io

Outline

- **Context from various companies**
- **Early results**
- **Format deep-dive**

Twitter Context

- **Twitter's data**
  - 200M+ monthly active users generating and consuming 400M+ tweets a day.
  - 100TB+ a day of compressed data
  - Scale is huge: Instrumentation, User graph, Derived data, ...

- **Analytics infrastructure**:
  - Several 1K+ node Hadoop clusters
  - Log collection pipeline
  - Processing tools

The Parquet Planers
Gustave Caillebotte

Twitter's use case

- Logs available on HDFS
- Thrift to store logs
- example: one schema has 87 columns, up to 7 levels of nesting.

```
struct LogEvent {
  1: optional logbase.LogBase log_base
  2: optional i64 event_value
  3: optional string context
  4: optional string referring_event
...
  18: optional EventNamespace event_namespace
  19: optional list<Item> items
  20: optional map<AssociationType,Association> associations
  21: optional MobileDetails mobile_details
  22: optional WidgetDetails widget_details
  23: optional map<ExternalService,string> external_ids
}
```

```
struct LogBase {
  1: string transaction_id,
  2: string ip_address,
  ...
  15: optional string country,
  16: optional string pid,
}
```

Goal

To have a state of the art columnar storage available across the Hadoop platform

- Hadoop is very reliable for big long running queries but also IO heavy.
- Incrementally take advantage of column based storage in existing framework.
- Not tied to any framework in particular

Columnar Storage

- **Limits the IO to only the data that is needed.**
- **Saves space:** columnar layout compresses better
- **Enables better scans:** load only the columns that need to be accessed
- **Enables vectorized execution engines.**

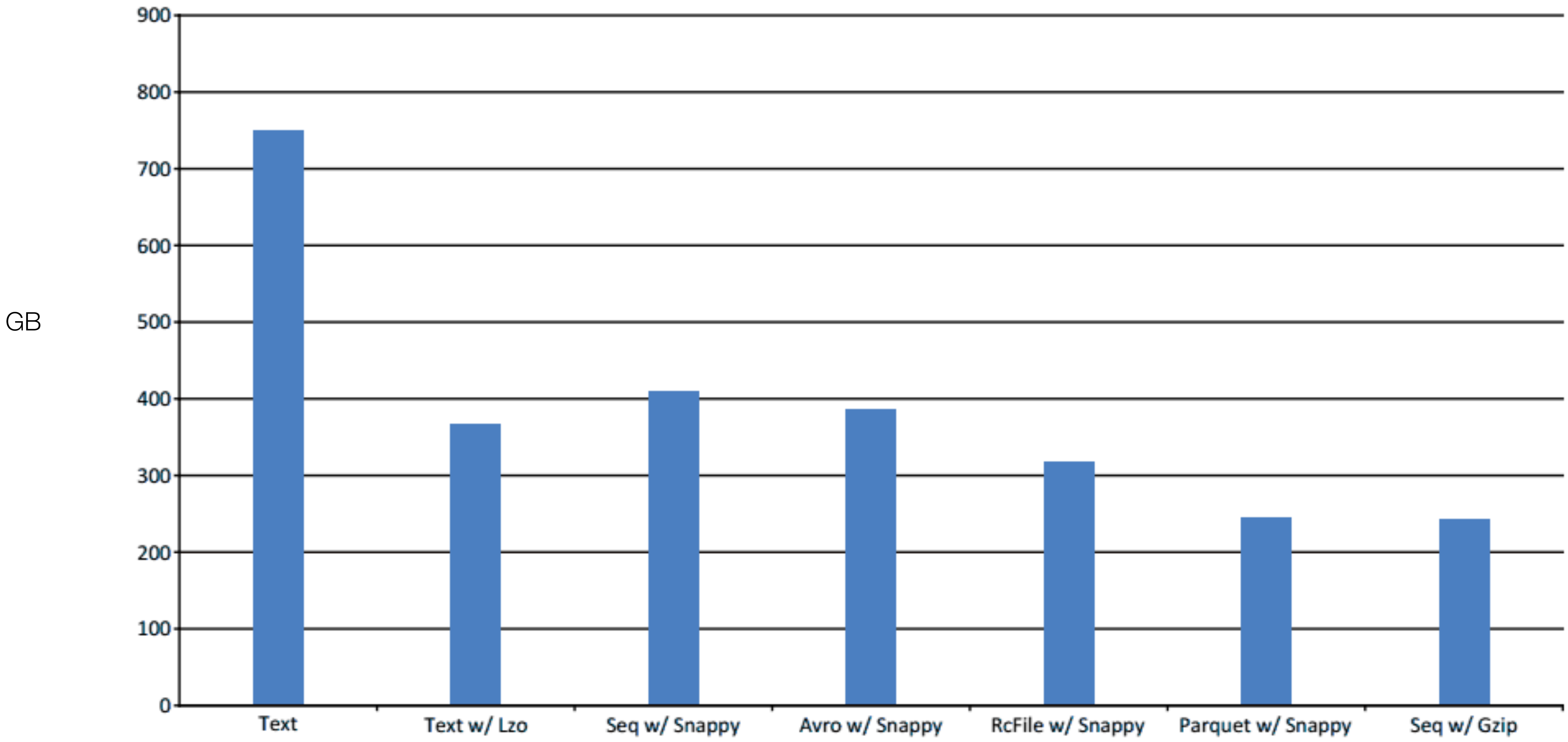
Collaboration between Twitter and Cloudera:

- **Common file format definition:**
  - Language independent
  - Formally specified.

- **Implementation in Java for Map/Reduce:**
  - https://github.com/Parquet/parquet-mr

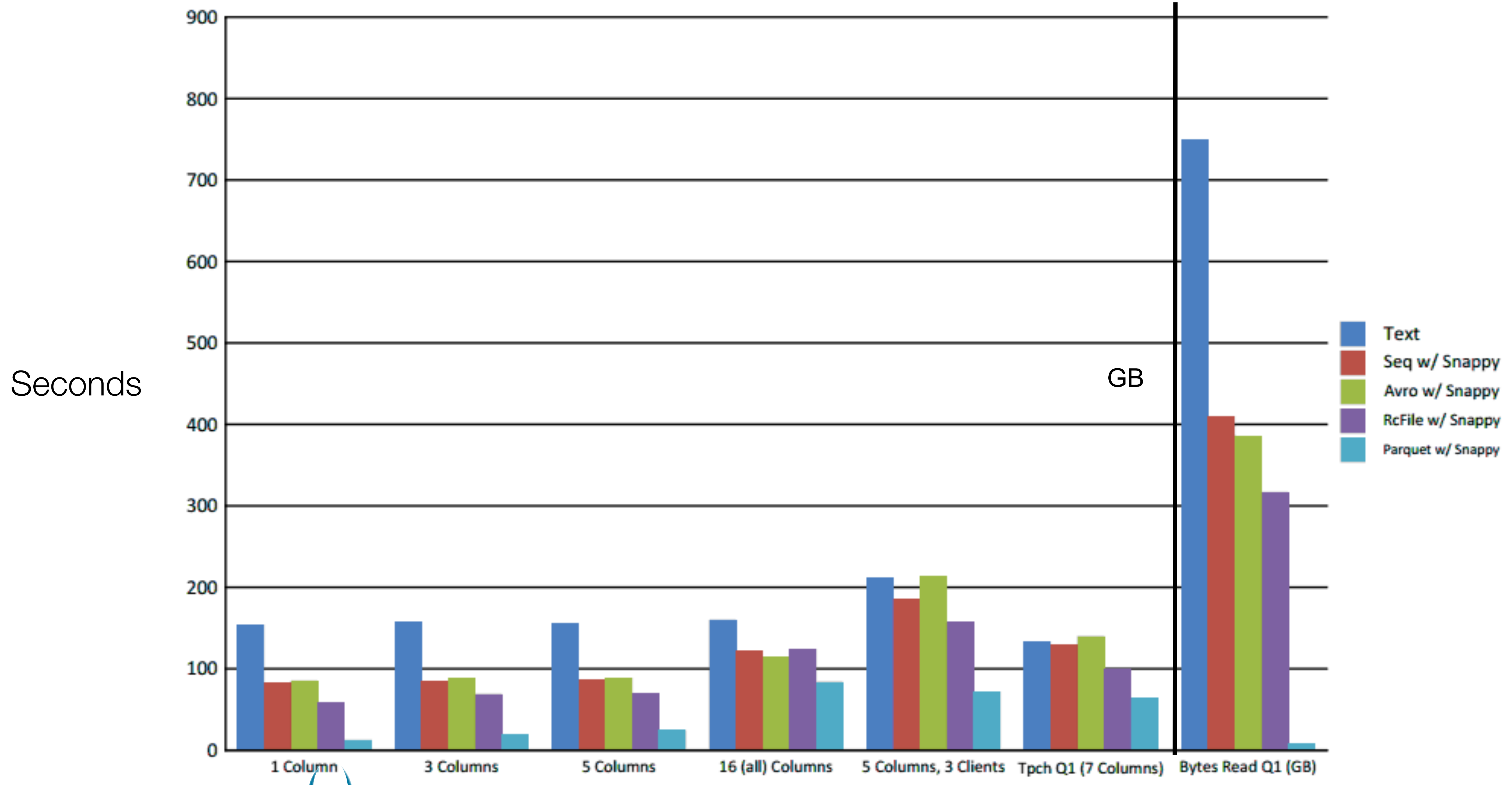
- **C++ and code generation in Cloudera Impala:**
  - https://github.com/cloudera/impala

Results in Impala **TPC-H lineitem table @ 1TB scale factor**



http://parquet.io

Impala query times on TPC-H lineitem table



http://parquet.io

Criteo: The Context

- ~20B Events per day
- ~60 Columns per Log
- Heavy analytic workload
- BI Analysts using Hive and RCFile
- Frequent Schema Modifications

==

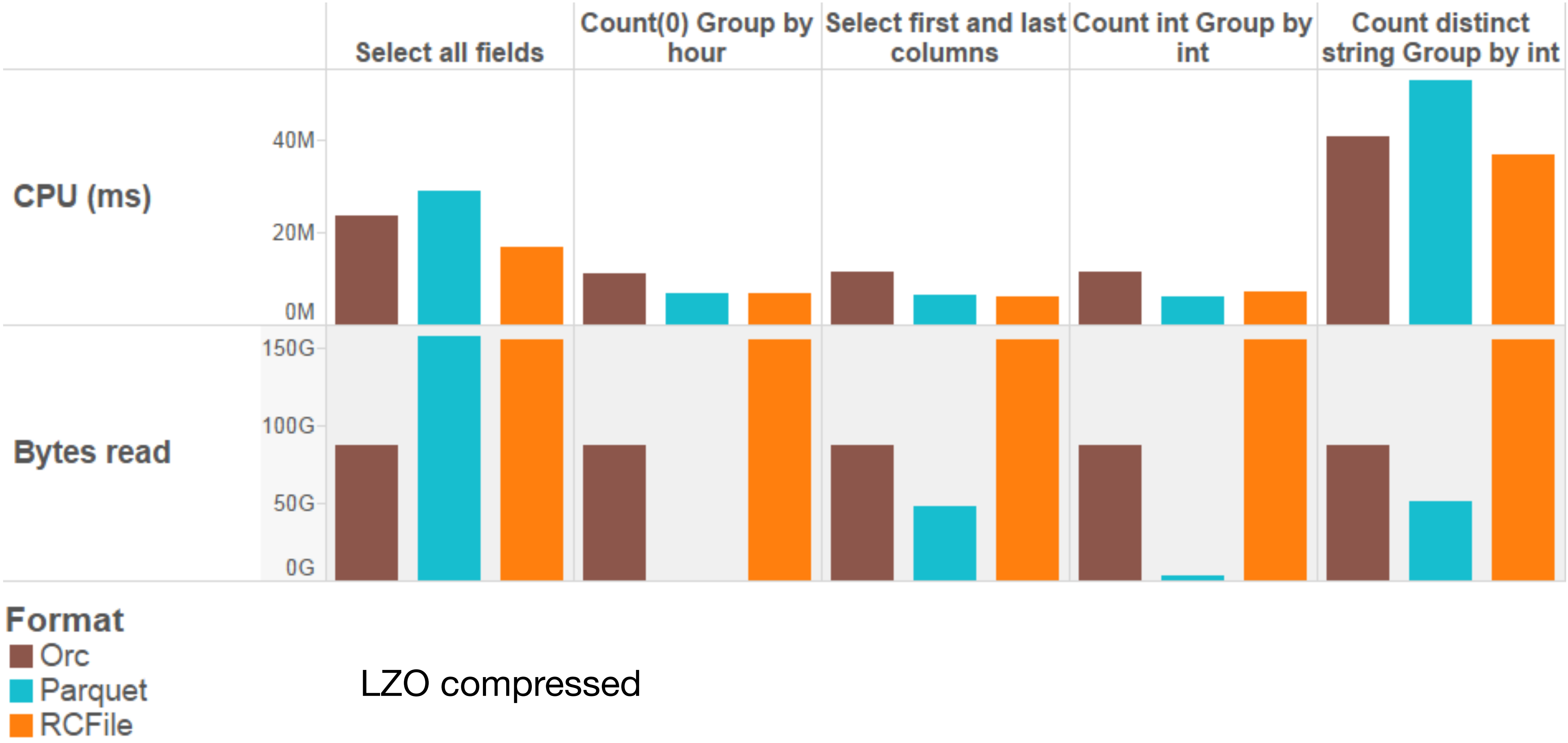
- Perfect use case for Parquet + Hive !

Parquet + Hive: Basic Reqs

- **MapRed Compatibility due to Hive**
- **Correctly Handle Different Schemas in Parquet Files**
- **Read Only The Columns Used by Query**
- **Interoperability with Other Execution Engines (eg Pig, Impala, etc.)**
- **Optimize Amount of Data Read by each Mapper**

Parquet + Hive: Early User Experience

Relative Performance of Hive+Parquet vs Orc and RCFile:

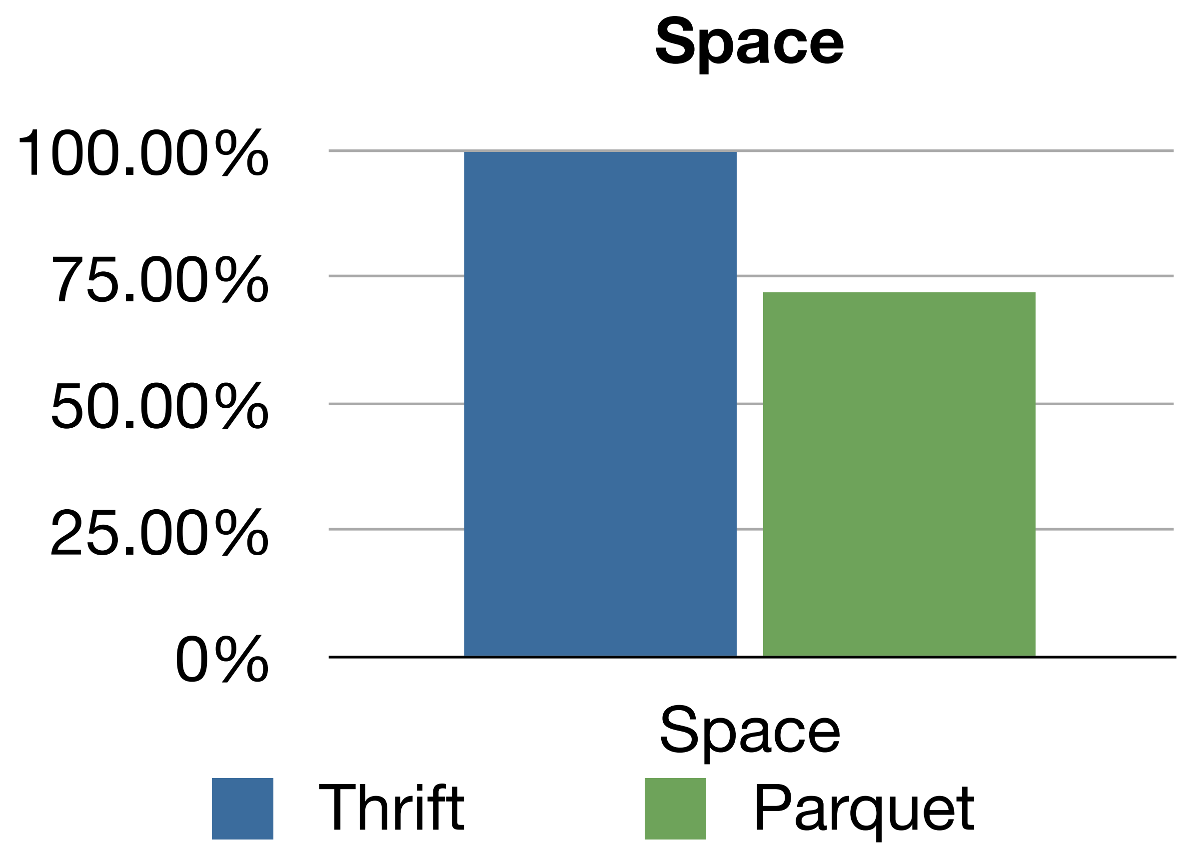


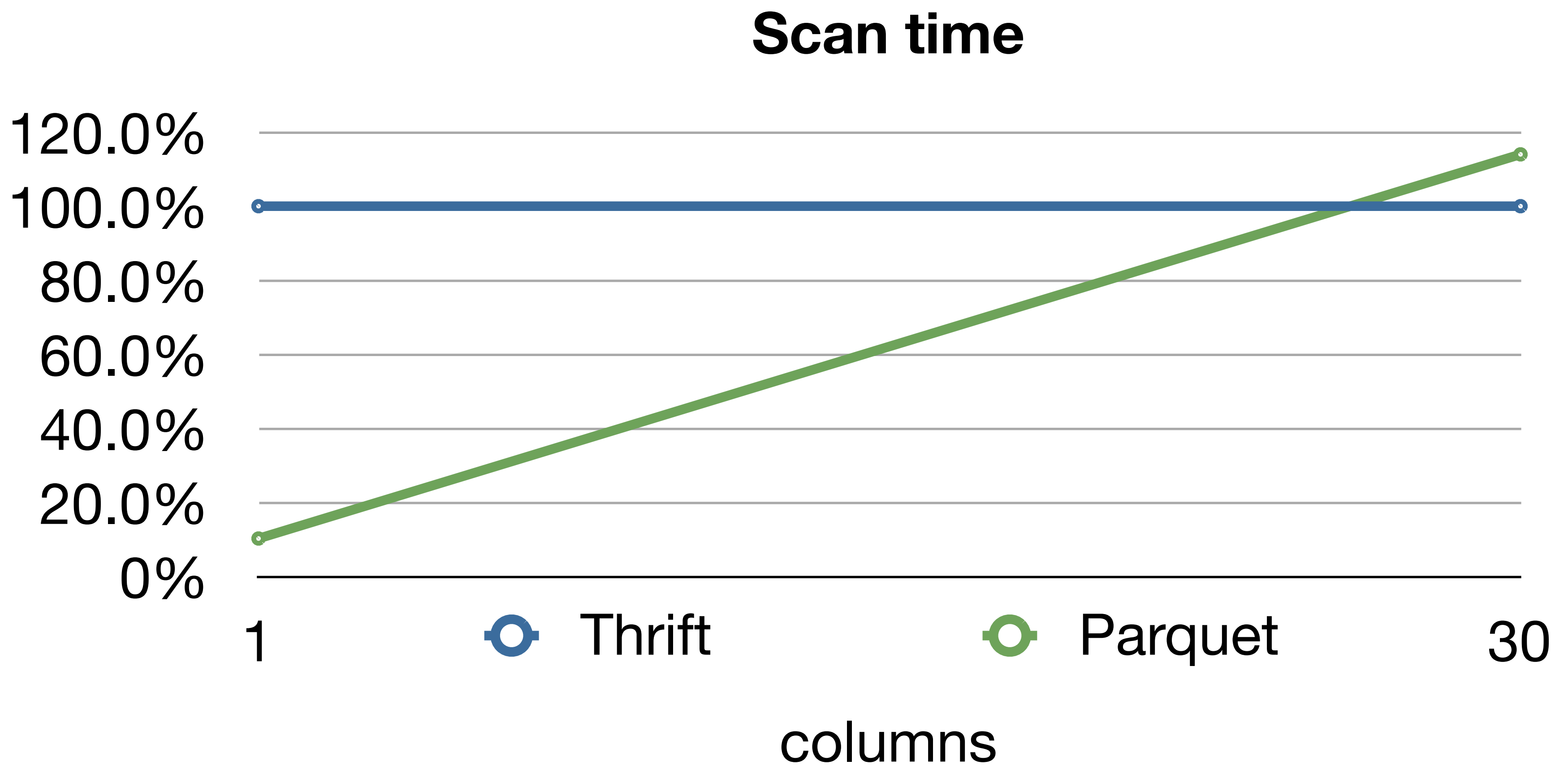
LZO compressed

Twitter: Initial results

Data converted: similar to access logs. 30 columns.

Original format: Thrift binary in block compressed files





Space saving: 28% using the same compression algorithm

Scan + assembly time compared to original:

One column: 10%

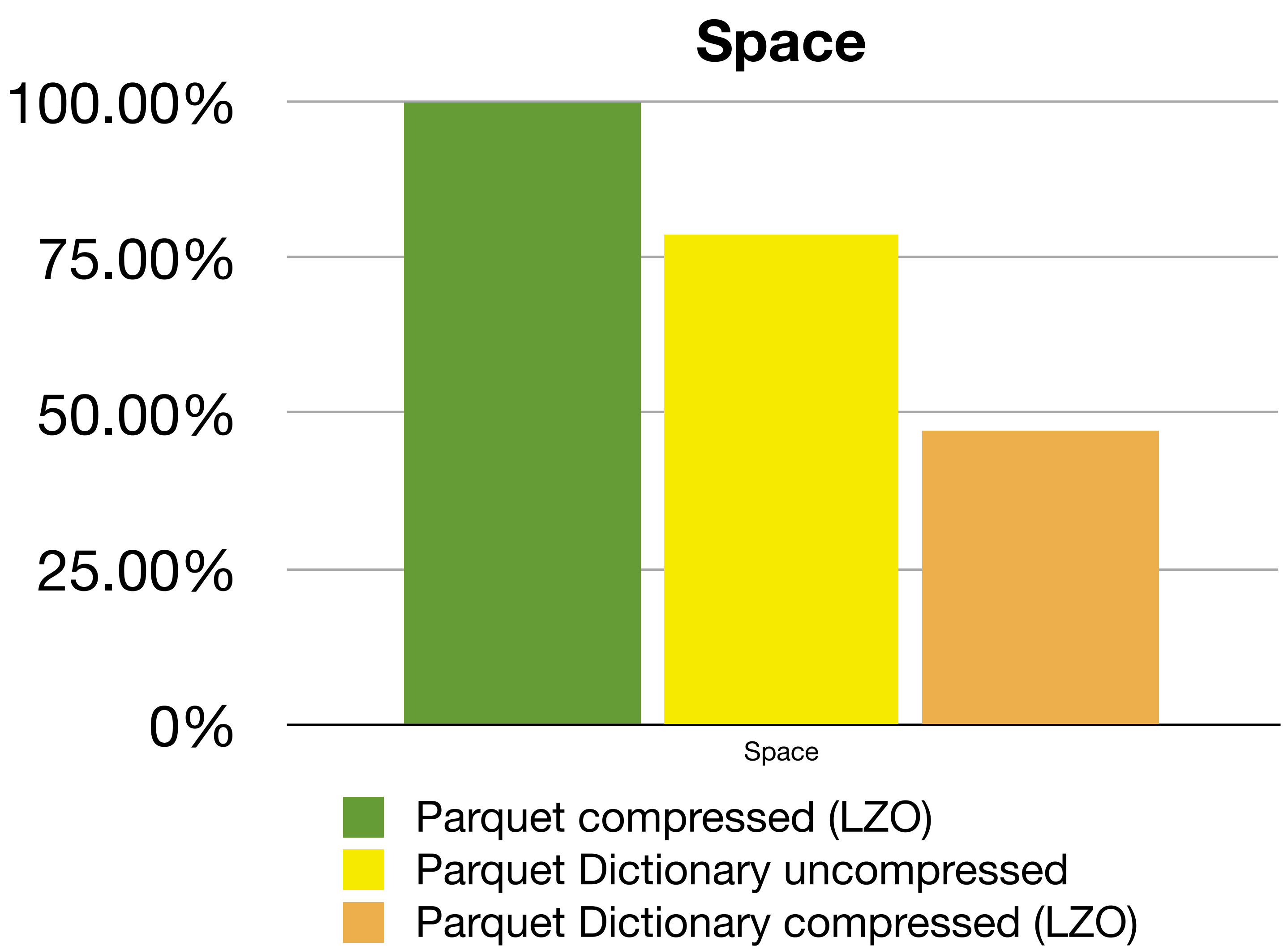
All columns: 114%

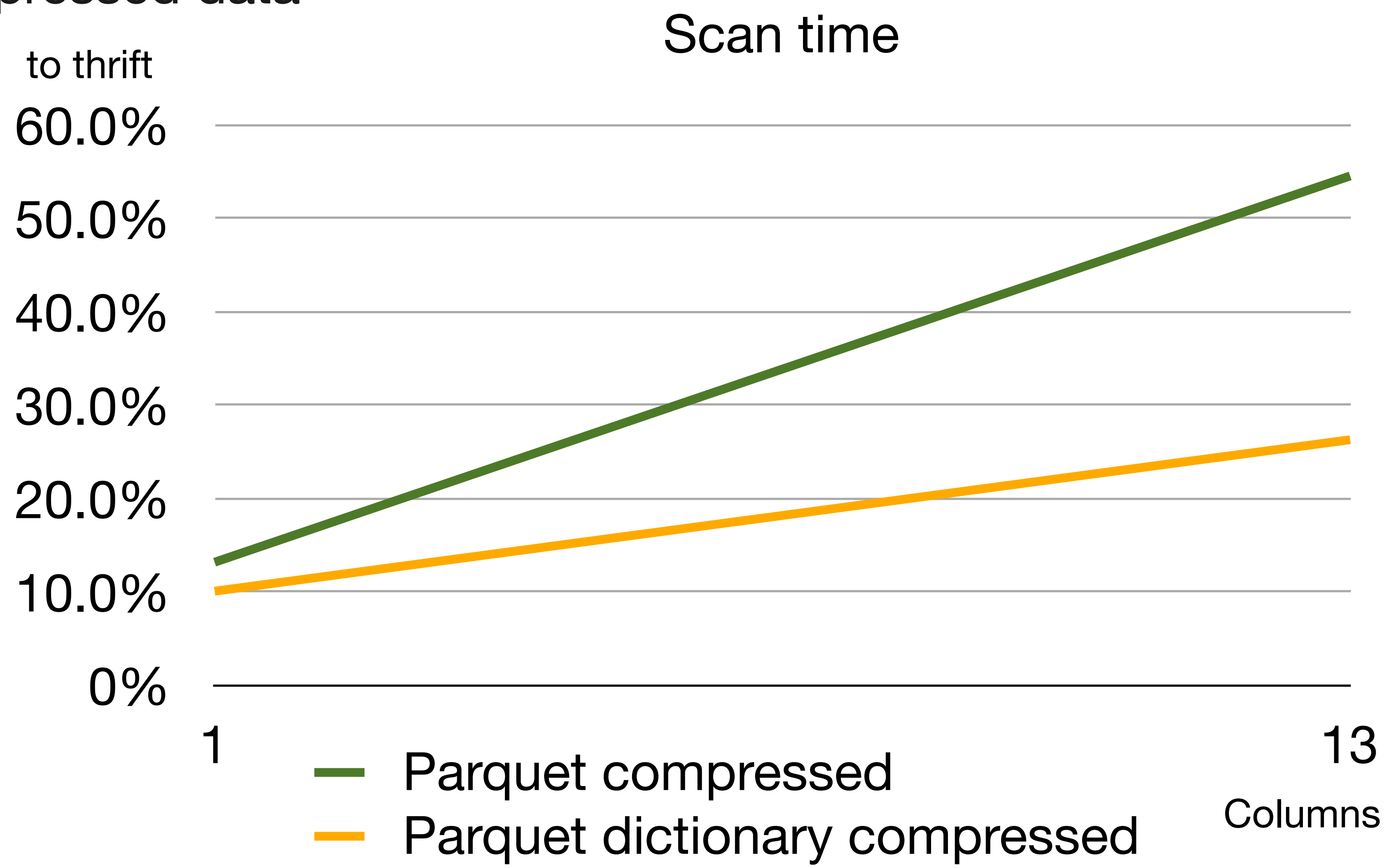
http://parquet.io

Additional gains with dictionary encoding

13 out of the 30 columns are suitable for dictionary encoding:

they represent 27% of raw data but only 3% of compressed data





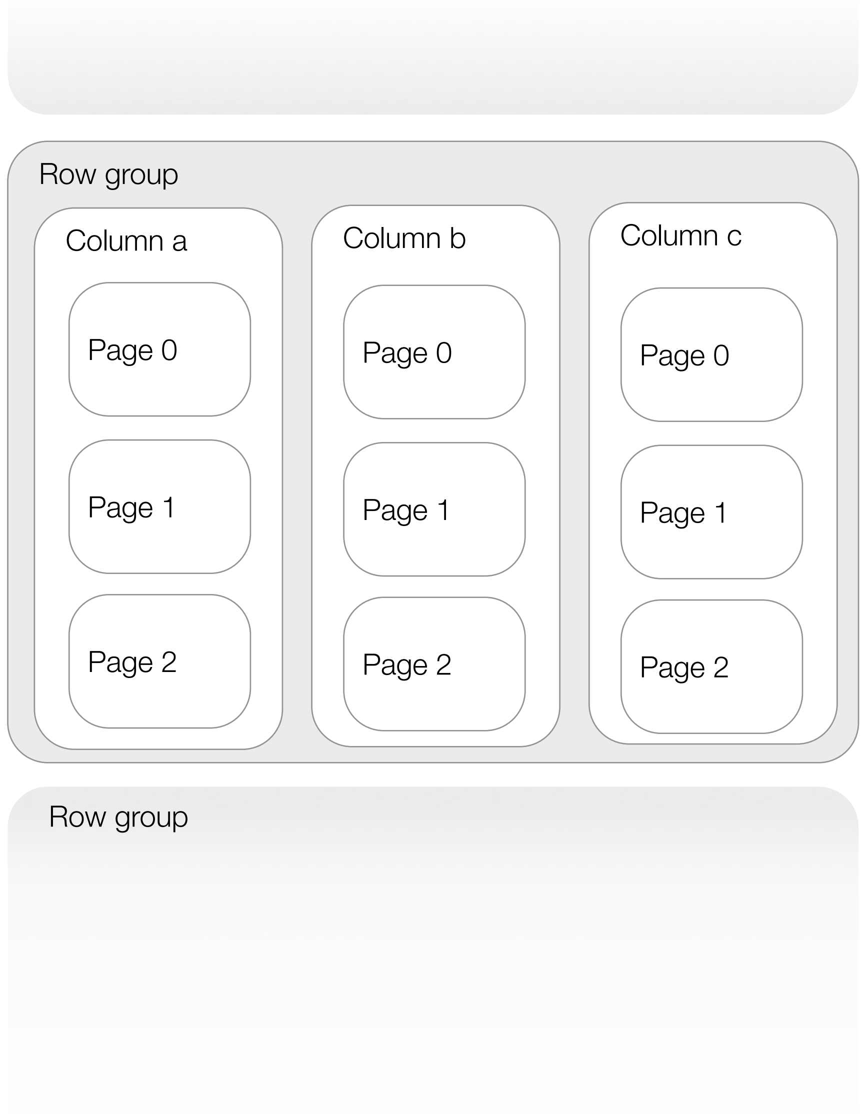
Space saving: another 52% using the same compression algorithm (on top of the original columnar storage gains)

Scan + assembly time compared to plain Parquet:

All 13 columns: 48% (of the already faster columnar scan)

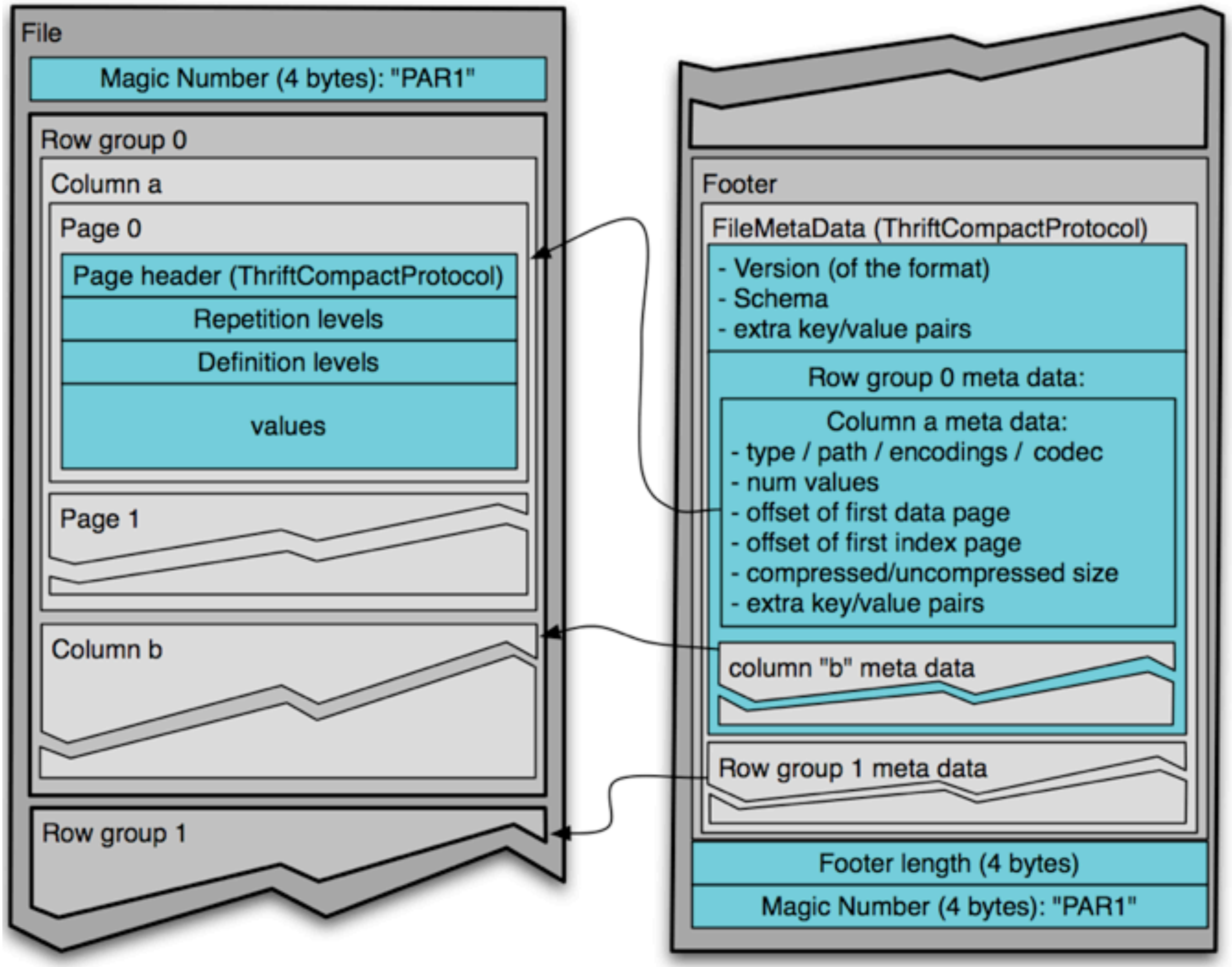
http://parquet.io

Format



- **Row group:** A group of rows in columnar format.
  - Max size buffered in memory while writing.
  - One (or more) per split while reading.
  - roughly: 50MB < row group < 1 GB
- **Column chunk:** The data for one column in a row group.
  - Column chunks can be read independently for efficient scans.
- **Page:** Unit of access in a column chunk.
  - Should be big enough for compression to be efficient.
  - Minimum size to read to access a single record (when index pages are available).
  - roughly: 8KB < page < 1MB

Format



Layout:

Row groups in columnar format. A footer contains column chunks offset and schema.

Language independent:

Well defined format. Hadoop and Cloudera Impala support.

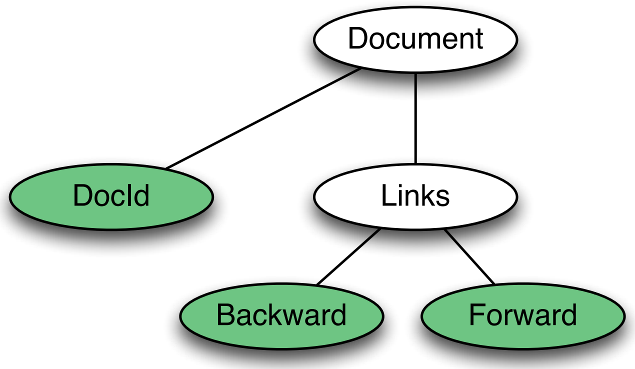
http://parquet.io

Nested record shredding/assembly

- Algorithm borrowed from Google Dremel's column IO
- Each cell is encoded as a triplet: **repetition level, definition level, value**.
- Level values are bound by the depth of the schema: **stored in a compact form.**

Schema:

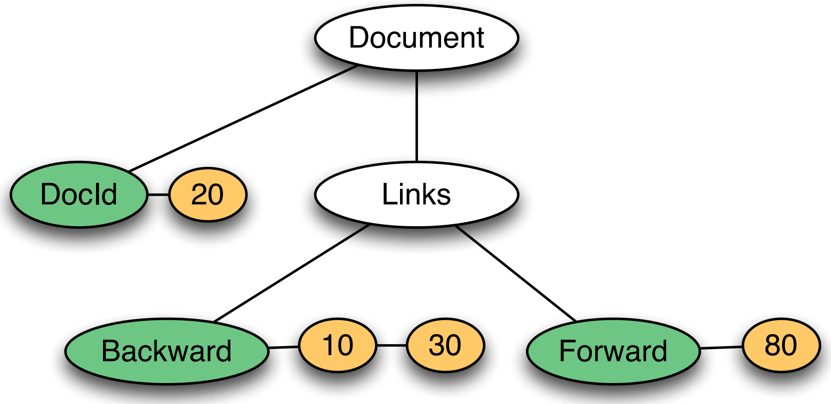
```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward;
  }
}
```



Columns	Max rep. level	Max def. level
DocId	0	0
Links.Backward	1	2
Links.Forward	1	2

Record:

```
DocId: 20
Links
  Backward: 10
  Backward: 30
  Forward:  80
```



Column	value	R	D
DocId	20	0	0
Links.Backward	10	0	2
Links.Backward	30	1	2
Links.Forward	80	0	2

http://parquet.io

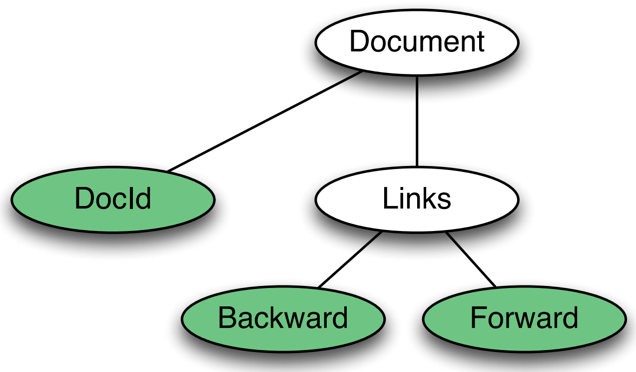
Differences of Parquet and ORC Nesting support

Parquet:

- Repetition/Definition levels capture the structure.

    => one column per *Leaf* in the schema.

- Array<int> is one column.
- Nullity/repetition of an inner node is stored in each of its children
- => One column independently of nesting with some redundancy.



ORC:

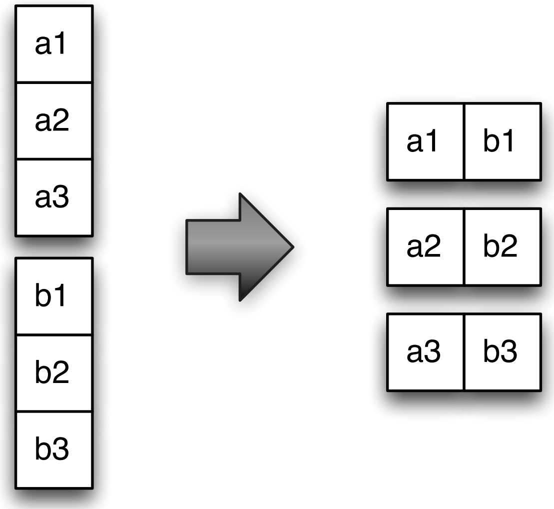
- An extra column for each Map or List to record their size.

    => one column per *Node* in the schema.

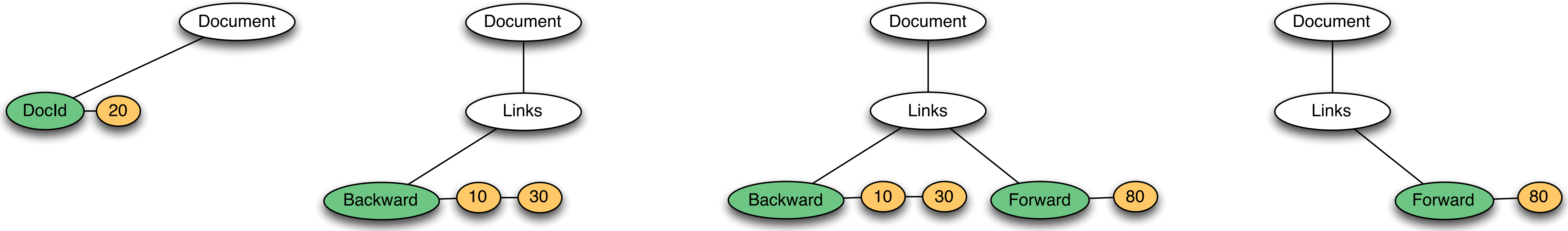
- Array<int> is two columns: array size and content.
- => An extra column per nesting level.

Iteration on fully assembled records

- To integrate with existing row based engines (Hive, Pig, M/R).
- Aware of dictionary encoding: enable optimizations.



- Assembles projection for any subset of the columns: only those are loaded from disc.



http://parquet.io

Iteration on columns

- To implement column based execution engine
- Iteration on triplets: repetition level, definition level, value.
- Repetition level = 0 indicates a new record.
- Encoded or decoded values: computing aggregations on integers is faster than on strings.

Row:	R	D	V
0	0	1	A
1	0	1	B
	1	1	C
2	0	0	
3	0	1	D

R=1 => same row

D<1 => Null

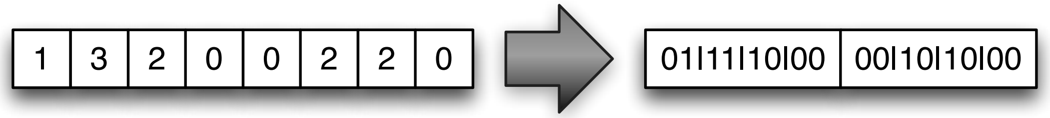
APIs

- **Schema definition and record materialization:**
  - Hadoop does not have a notion of schema, however Impala, Pig, Hive, Thrift, Avro, ProtocolBuffers do.
  - Event-based SAX-style record materialization layer. No double conversion.

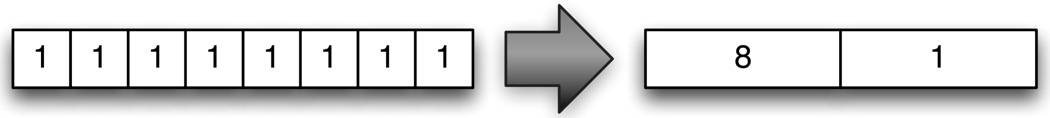
- **Integration with existing type systems and processing frameworks:**
  - Impala
  - Pig
  - Thrift and Scrooge for M/R, Cascading and Scalding
  - Cascading tuples
  - Avro
  - Hive

Encodings

- **Bit packing:**



  - Small integers encoded in the minimum bits required
  - Useful for repetition level, definition levels and dictionary keys
- **Run Length Encoding:**



  - Used in combination with bit packing,
  - Cheap compression
  - Works well for definition level of sparse columns.
- **Dictionary encoding:**
  - Useful for columns with few (< 50,000) distinct values
- **Extensible:**
  - Defining new encodings is supported by the format

Contributors

Main contributors:

- Julien Le Dem (Twitter): Format, Core, Pig, Thrift integration, Encodings
- Nong Li, Marcel Kornacker, Todd Lipcon (Cloudera): Format, Impala
- Jonathan Coveney, Alex Levenson, Aniket Mokashi (Twitter): Encodings
- Mickaël Lacour, Rémy Pecqueur (Criteo): Hive integration
- Dmitriy Ryaboy (Twitter): Format, Thrift and Scrooge Cascading integration
- Tom White (Cloudera): Avro integration
- Avi Bryant (Stripe): Cascading tuples integration

Future

- **Indices for random access (lookup by ID).**
- **More encodings.**
- **Extensibility**
- **Statistics pages (max, min, ...)**

How to contribute

Questions? Ideas?

Contribute at: github.com/Parquet

Come talk to us:

Twitter booth #26

Cloudera booth #45