An Empirical Evaluation of Columnar Storage Formats

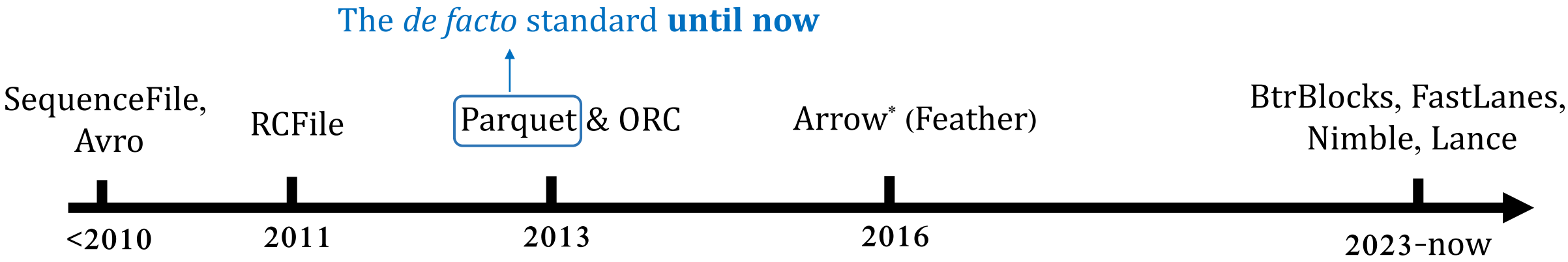
Xinyu Zeng, Yulong Hui, Jiahong Shen,
Andrew Pavlo[1], Wes McKinney[2], Huanchen Zhang

Tsinghua University

Carnegie Mellon University [1]

VOLTRON DATA [2]

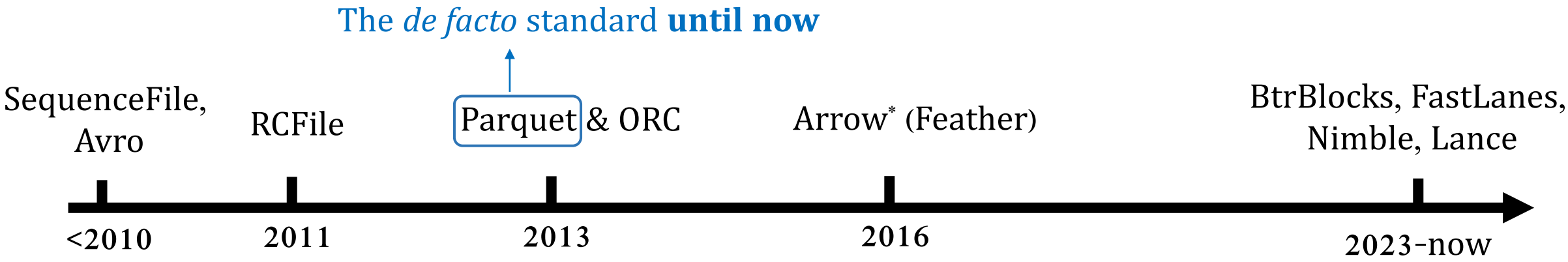
File Formats Evolution



Are Parquet and ORC still good for modern workload and hardware?

*Arrow is not a file format but I include it here because of its importance

File Formats Evolution

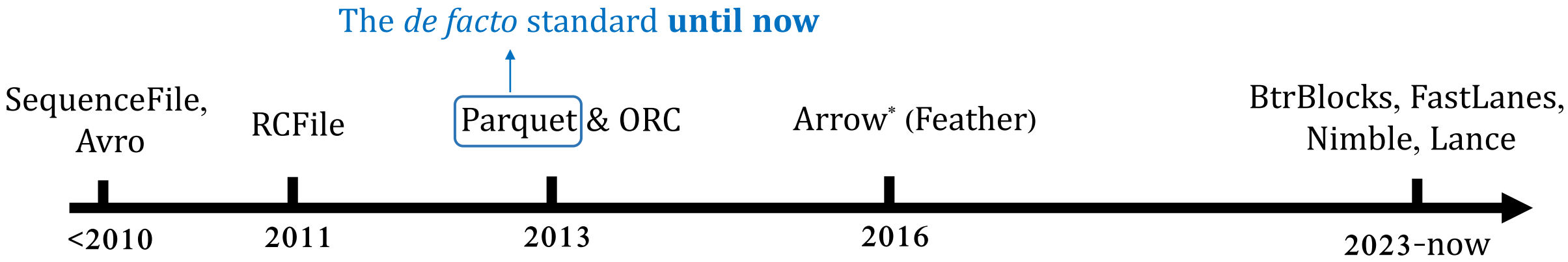


Are Parquet and ORC still good for modern workload and hardware?

No prior work provides an in-depth analysis of the format internals

*Arrow is not a file format but I include it here because of its importance

File Formats Evolution



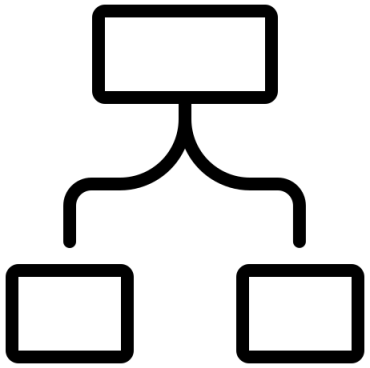
Are Parquet and ORC still good for modern workload and hardware?

No prior work provides an in-depth analysis of the format internals

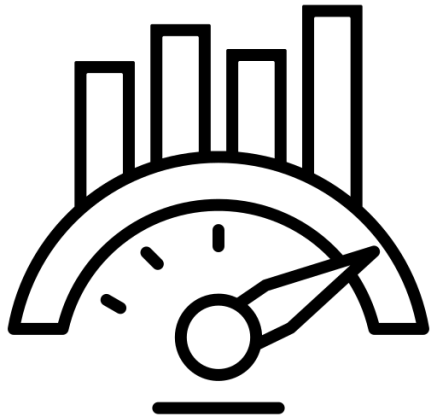
Goal: analyze Parquet and ORC to provide insights

*Arrow is not a file format but I include it here because of its importance

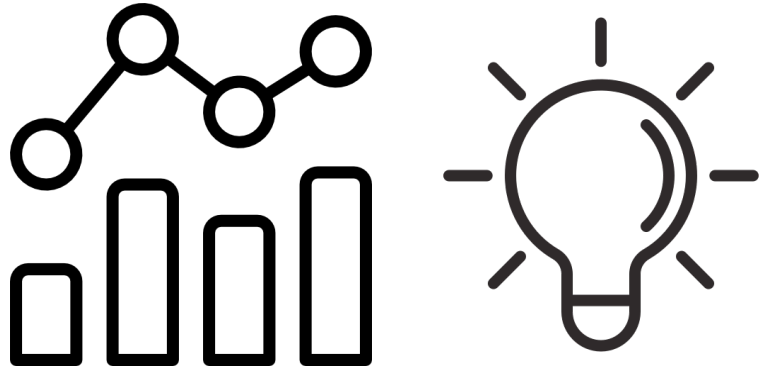
Main Contributions



Feature Taxonomy



Columnar Storage Benchmark



Comprehensive Experiments and Lessons Learned

Feature Taxonomy

File Metadata

Format Layout

Type System

Encoding Schemes

Block Compression

Filters

Nested Data

Feature Taxonomy

File Metadata

Format Layout

Type System

Encoding Schemes

Block Compression

Filters

Nested Data

Encodings

Parquet

Dict + RLE/BP → String, Int, Float

- ☑ Better when high repetition
- ☑ Less branches

Apache ORC

Dict + RLE/BP/Delta/FOR → String

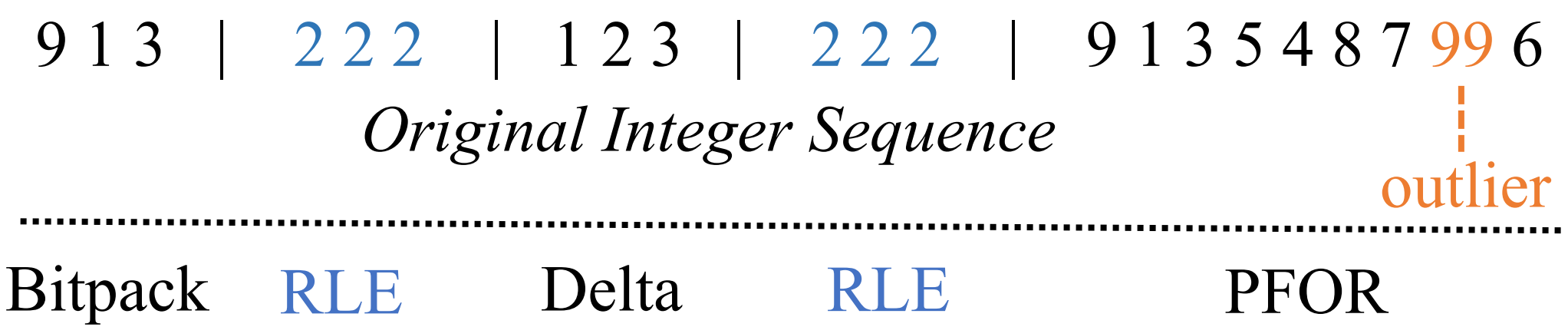
RLE/BP/Delta/FOR → Int

Plain → Float

- ☑ Better compression ratio when data fits Delta/FOR

ORC Integer Encoding

Interleaved (small) encoded sequences introduces branches



Encodings

Parquet

Dict + RLE/BP → String, Int, Float

- ☑ Better when high repetition
- ☑ Less branches

Apache ORC™

Dict + RLE/BP/Delta/FOR → String

RLE/BP/Delta/FOR → Int

Plain → Float

- ☑ Better compression ratio when data fits Delta/FOR

Why just discuss the Default? - Most commonly used

Columnar Storage Benchmark

Why not TPCH?

- Synthetic data with uniform distribution
- Not real-world data

Why not directly use real data?

- Unable to sweep data distribution
- Many different data sets

⇨ Idea: analyze real data's properties, and generate synthetic data that are close to real data but can control certain property

Workloads and Corresponding Data Sets

bi: Public BI

classic: IMDb, Yelp, ClickHouse examples

geo: Geonames, ClickHouse examples

log: EDGAR log

ml: UCI-ML

core: mix of the above

Column Properties

NDV Ratio: # of distinct values / # of total values

Null Ratio: # of nulls / # of total values

Value Range: average value and variance

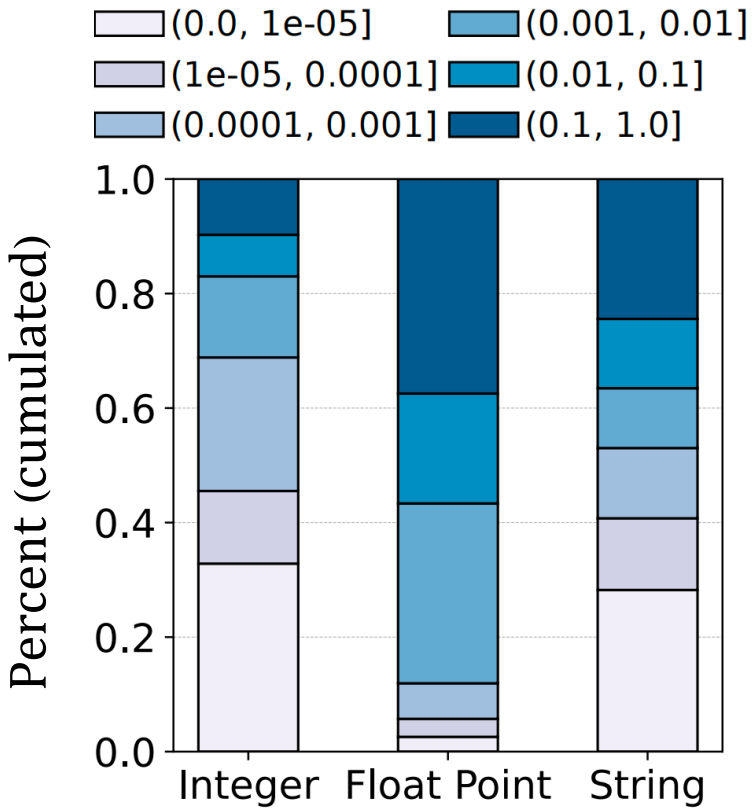
Sortedness: degree of sortedness

Skew Pattern: uniform, gentle zipf, hotspot, single/binary

Distribution in the real world

NDV Ratio

(# of distinct values / # of total values)



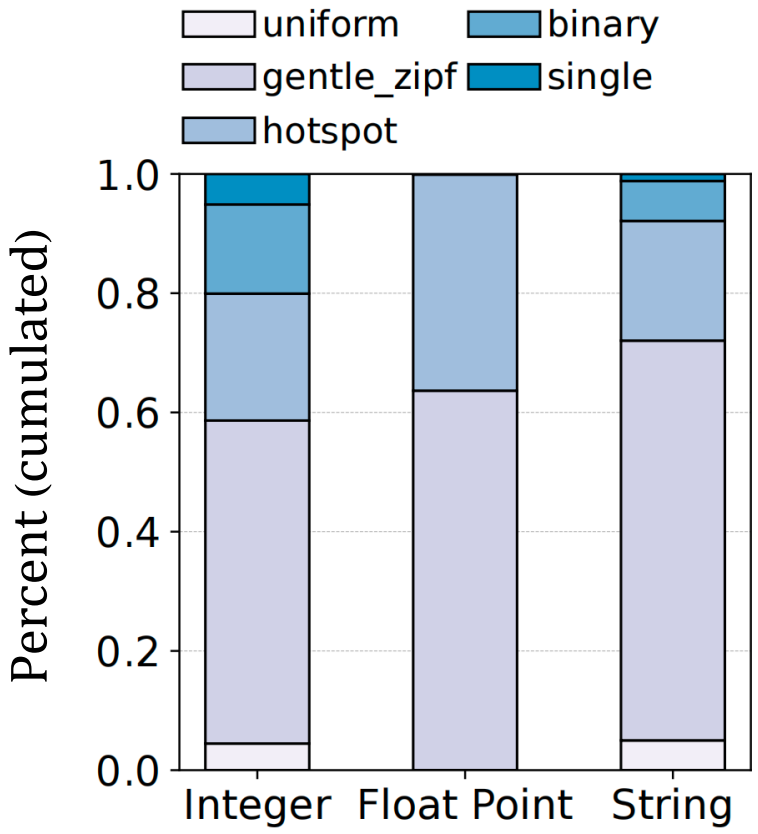
→ Low NDV

Highly skewed ←

↓

Dictionary wins

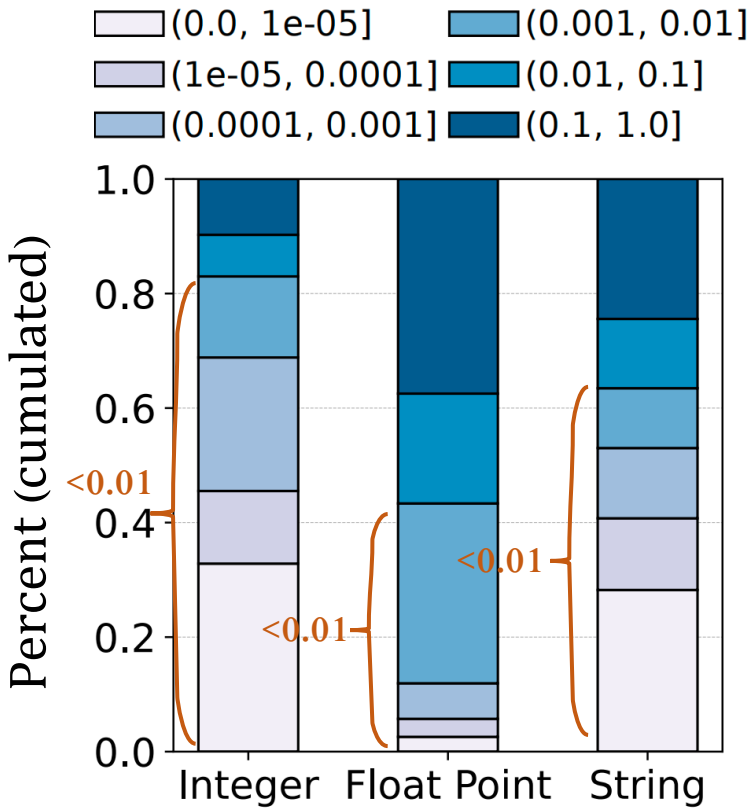
Skew Pattern



Distribution in the real world

NDV Ratio

(# of distinct values / # of total values)



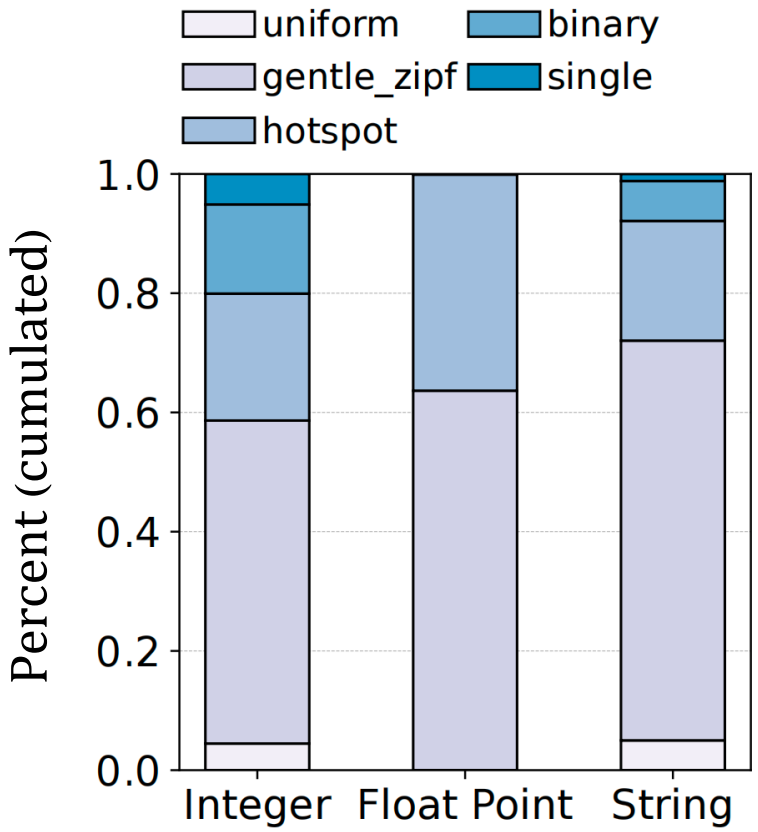
→ Low NDV

Highly skewed ←

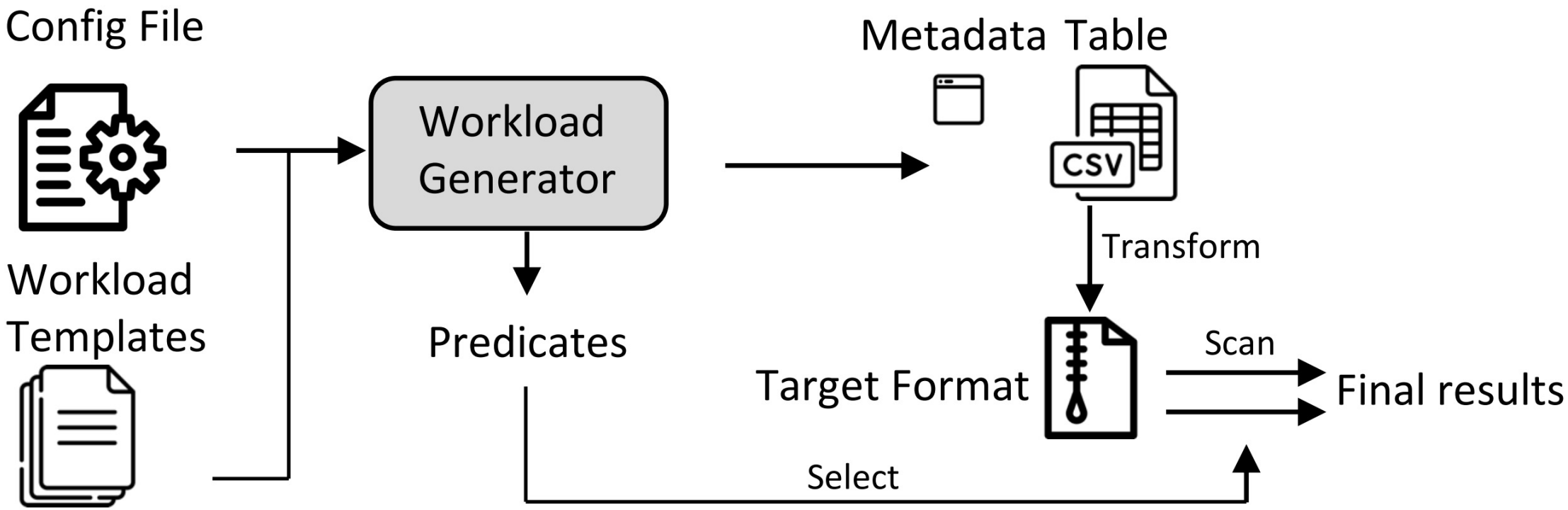
↓

Dictionary wins

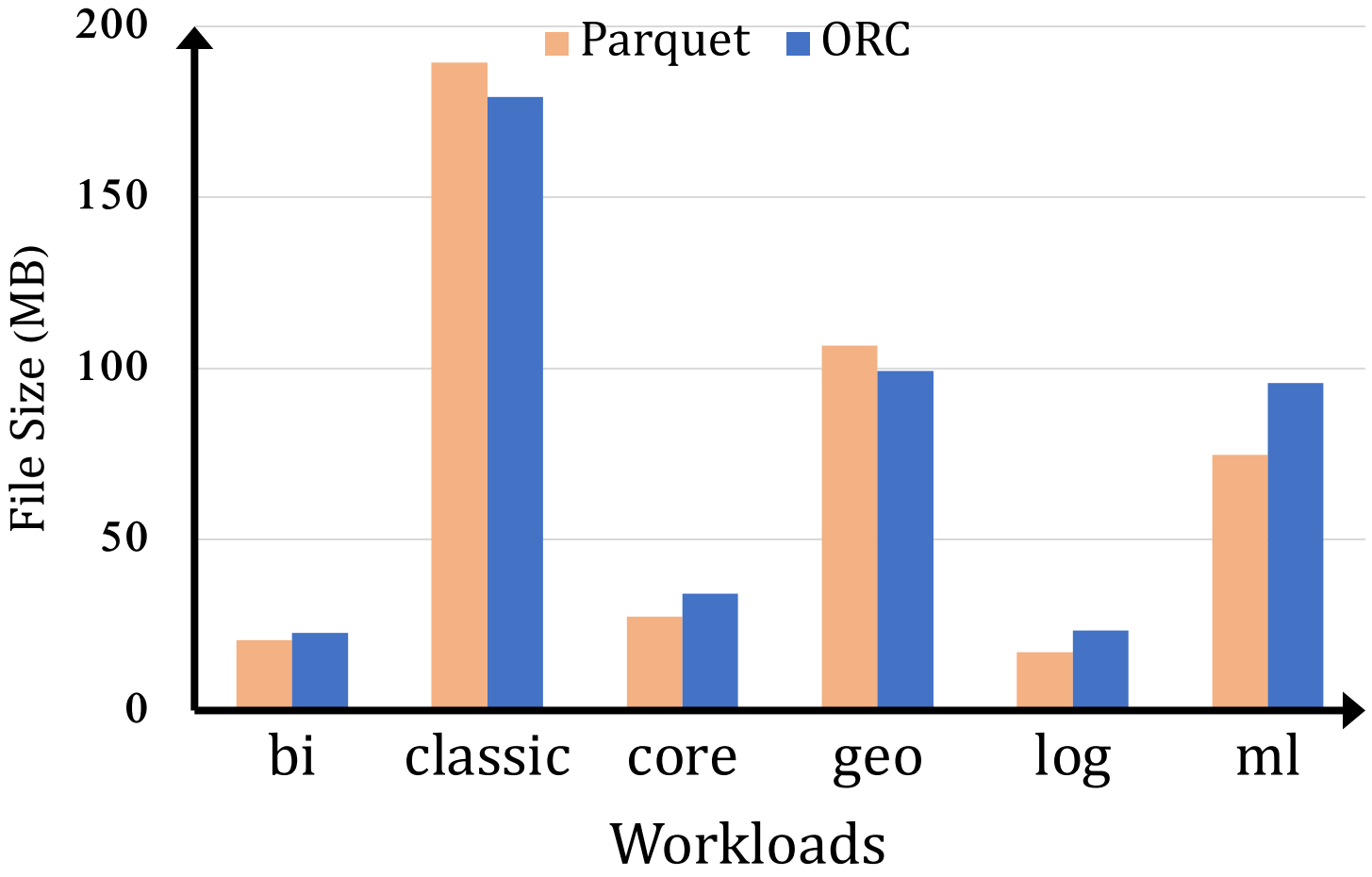
Skew Pattern



Benchmark Methodology

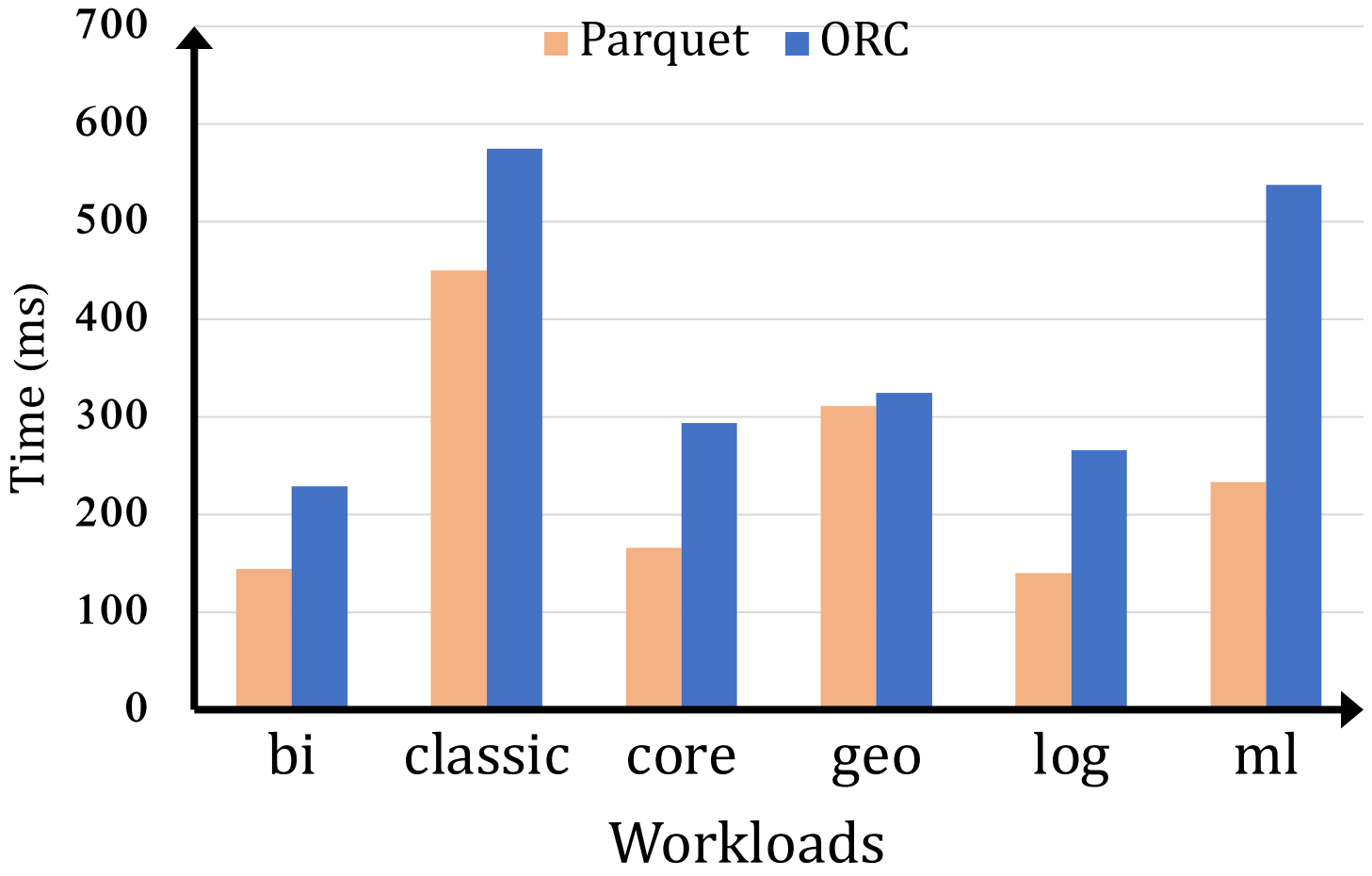


File Size



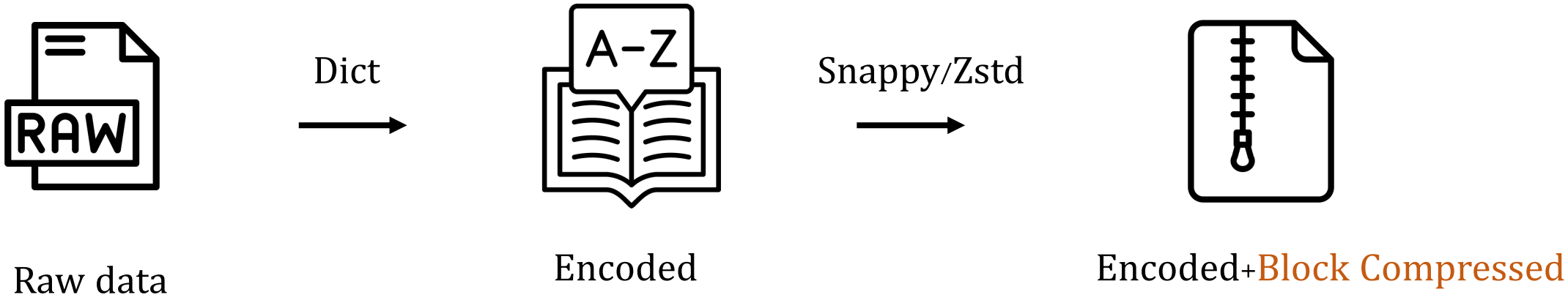
- No clear winner
- Parquet wins on log and ml because it applies dictionary aggressively

Decoding Speed



Parquet is faster
-> Simpler integer encoding

Block Compression



Raw data

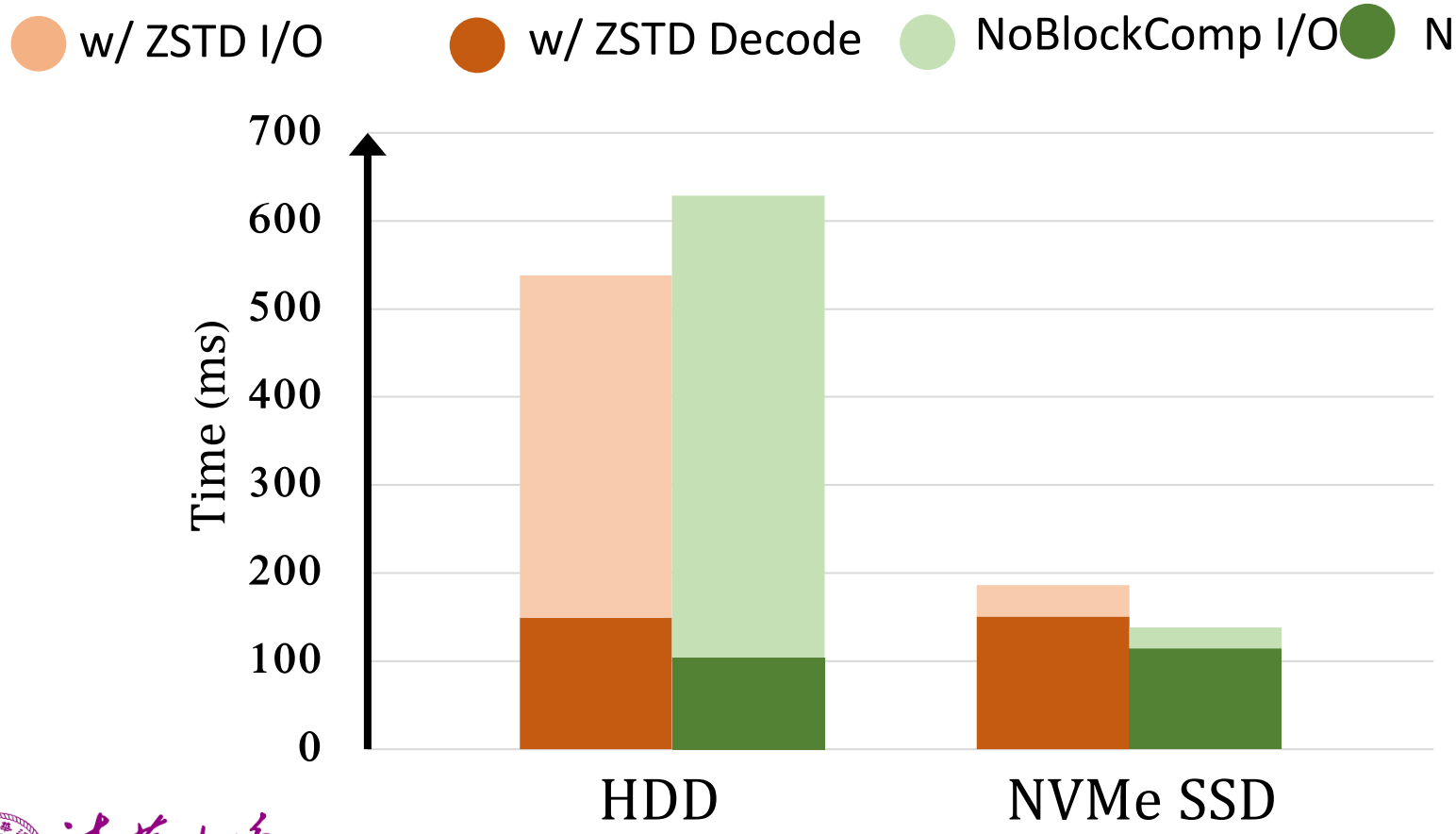
Encoded

Encoded+Block Compressed

Will this another layer benefit overall scan performance?

Block Compression

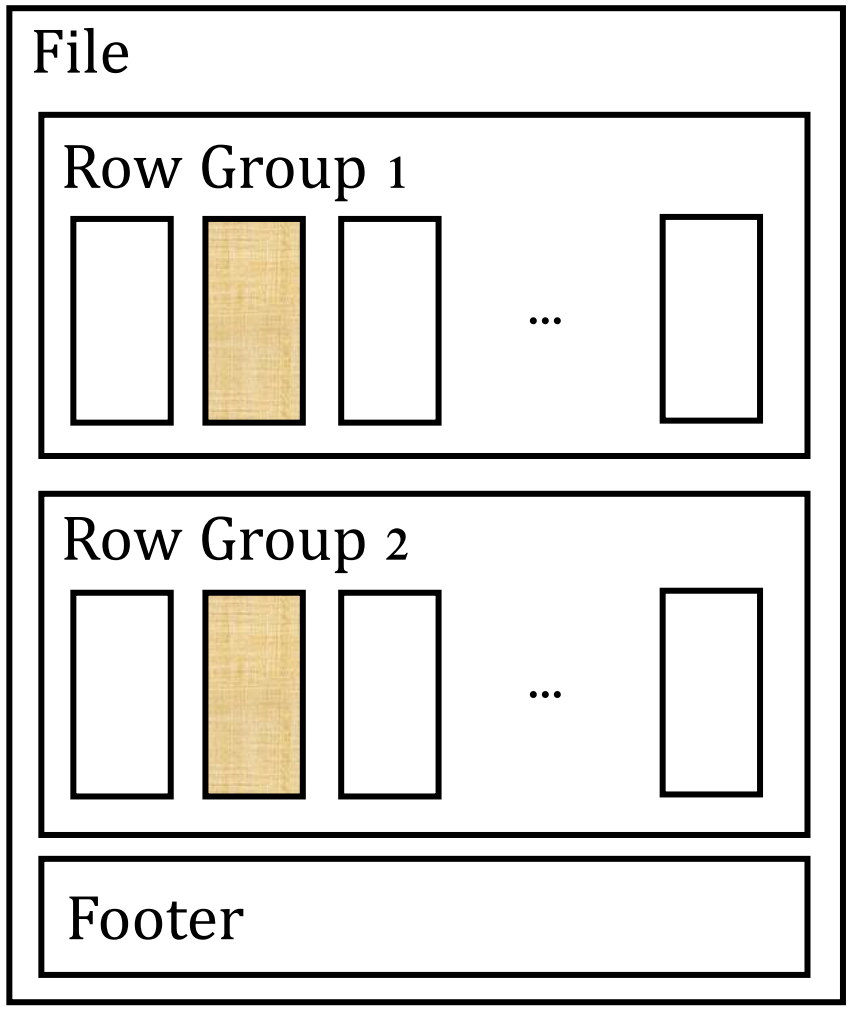
Reading a Parquet file w/ varying block compression on *core* workload



Block compression hurts performance on fast storage

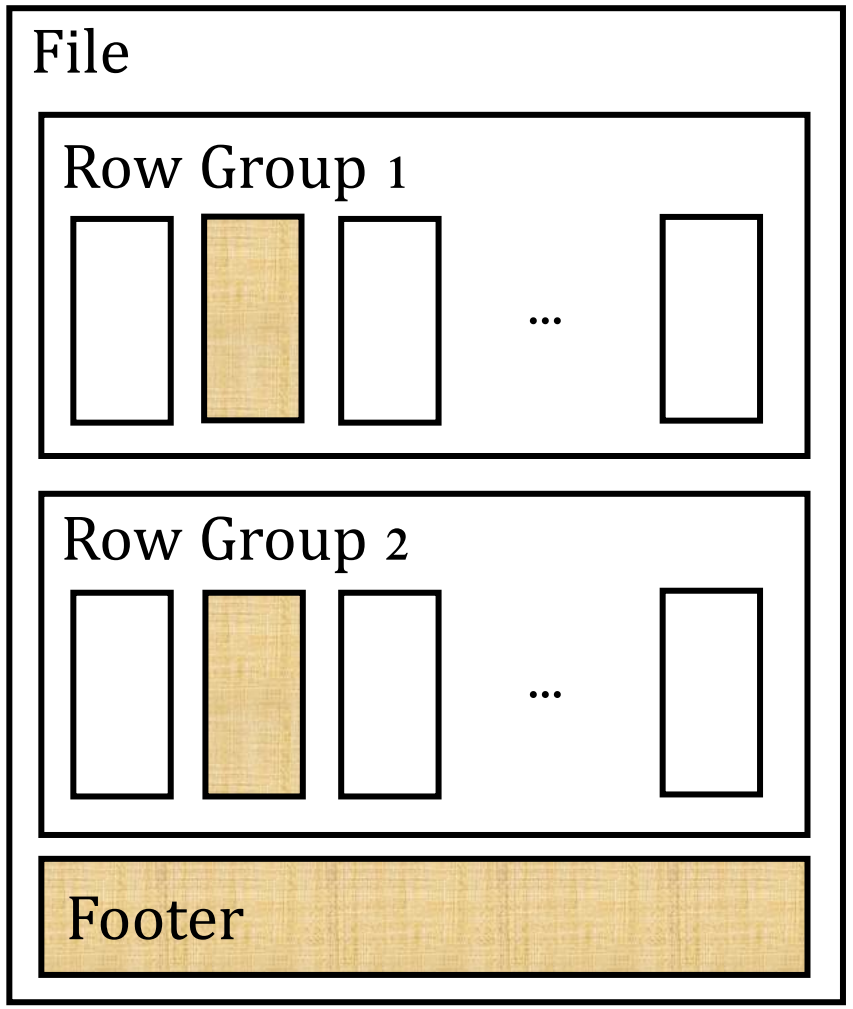
Wide-Table Projection

"Read one column from a table with 10000 columns"



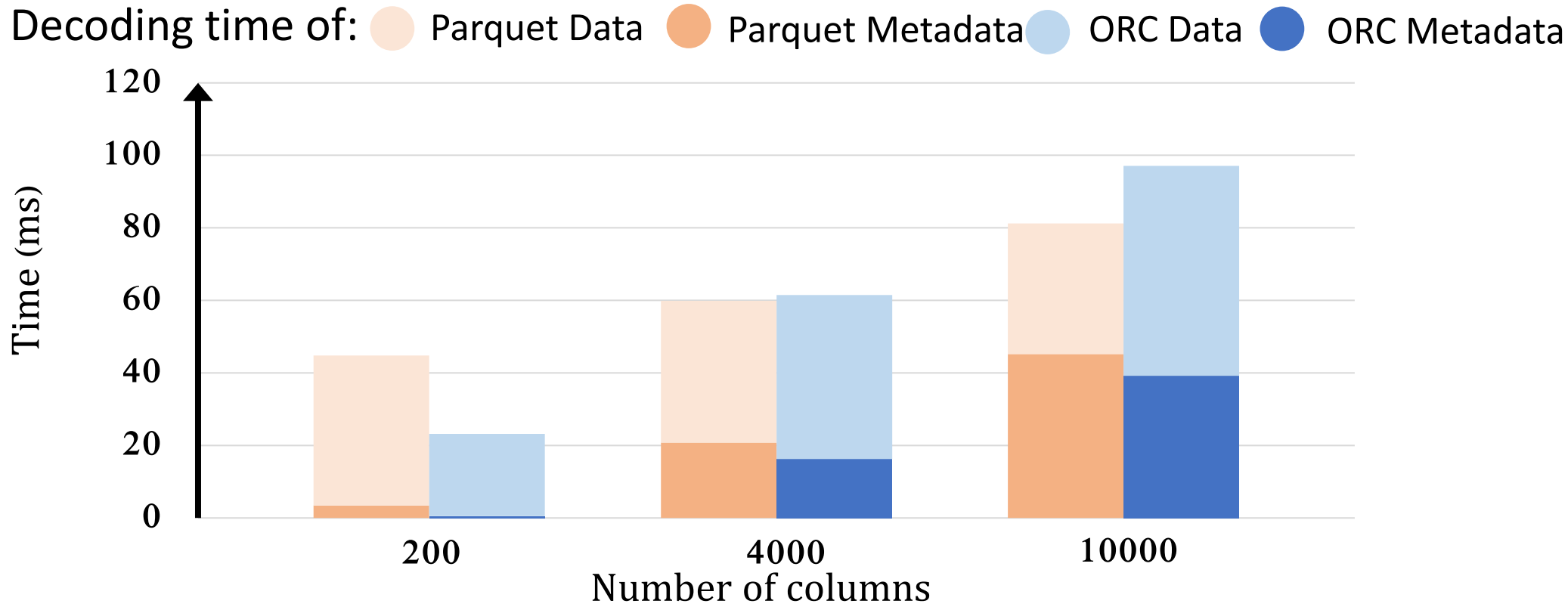
Wide-Table Projection

"Read one column from a table with 10000 columns"



Need to read the whole footer <-

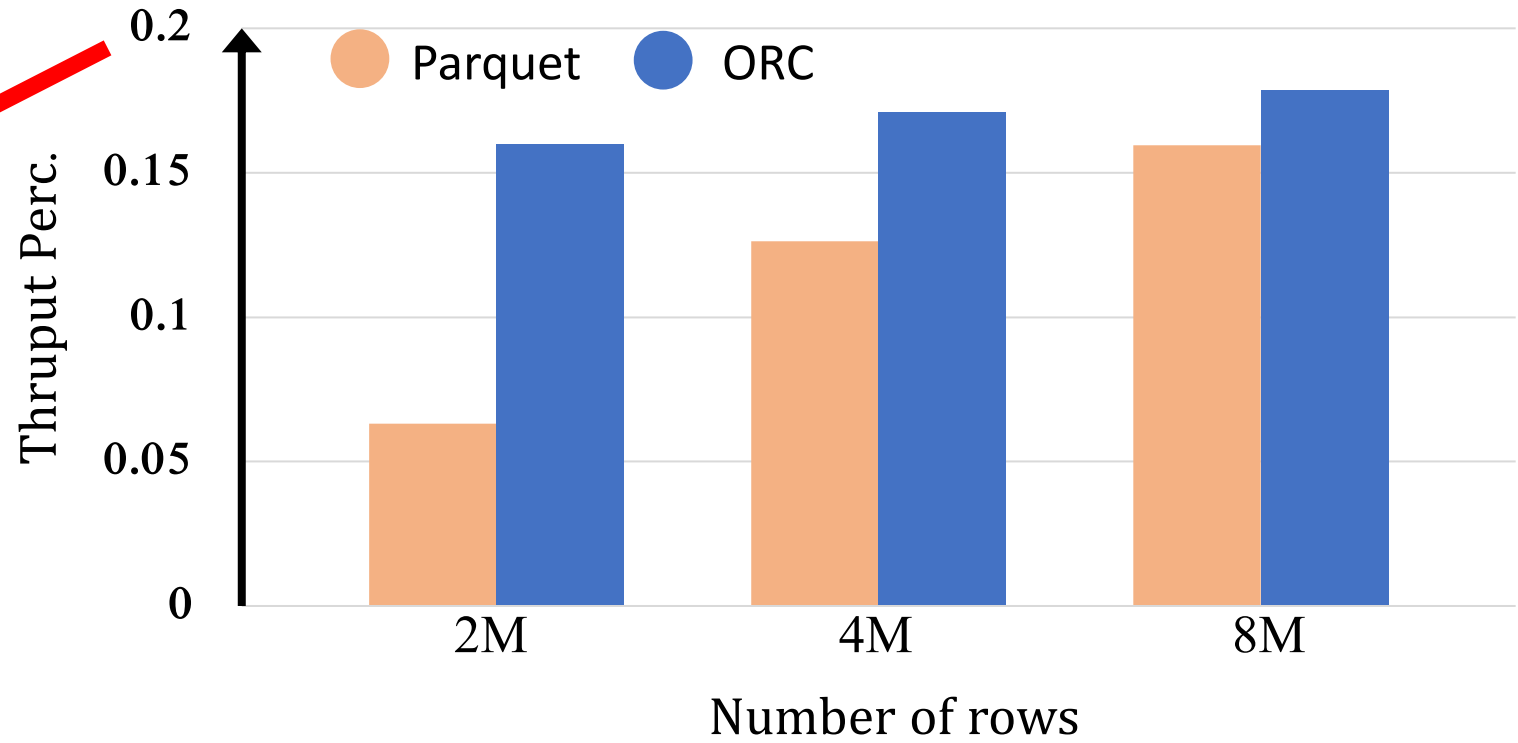
Wide-Table Projection



Metadata serialization (Thrift or Protobuf) are not projection friendly

GPU Decoding

Low compute utilization for both formats!



Too large chunk -> Not enough parallelizable blocks

Data dependency -> Underutilized threads

Takeaways

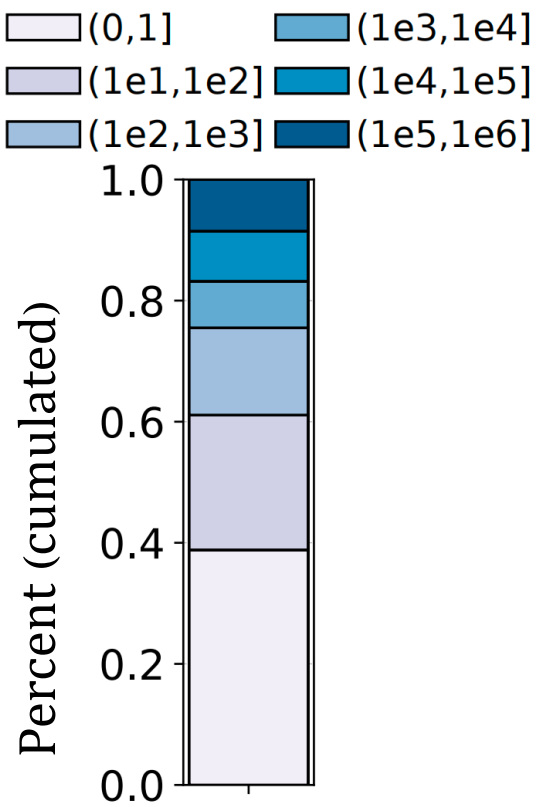
Contact: xzeng56@gmail.com

- ⇨ Dictionary is effective across data types
- ⇨ Encoding should avoid branches to benefit vectorized decoding
- ⇨ Limit the use of block compression
- ⇨ Metadata should be projection-friendly
- ⇨ ML workloads require fast selection and handling blob data
- ⇨ Redesign to fit parallel hardware like GPU

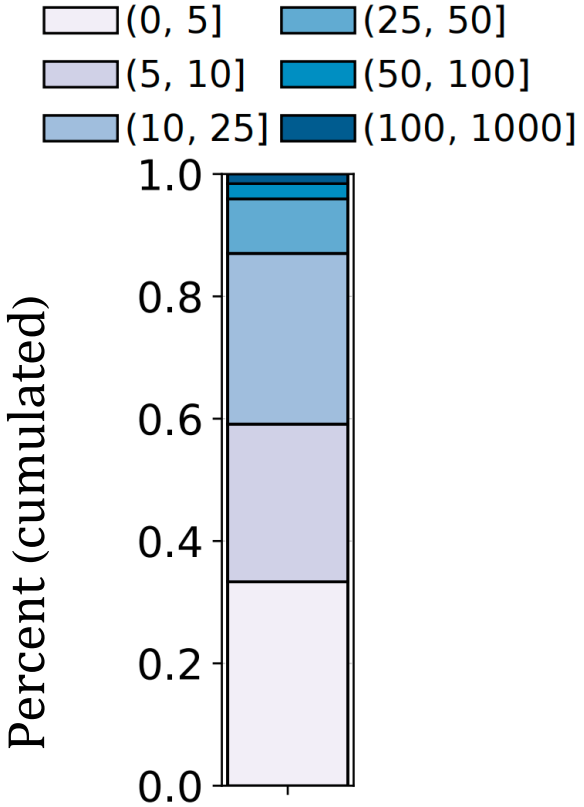
More lessons learned in the paper!

Backup Slides

Distribution in the real world



Integer Value Range



String Length

GPU Decoding

Unknown uncompressed size requires a first sequential pass

GPU

Unkno

PARQUET-2261: add statistics for better estimating unencoded/uncompressed sizes and finer grained filtering #197

Code

Merged

emkornfield merged 34 commits into `apache:master` from `emkornfield:plain_stats` on Nov 15, 2023

Conversation 246 | Commits 34 | Checks 2 | Files changed 1 | +91 −11

Changes from **all commits** | **File filter** | **Conversations** | **Jump to** | 0 / 1 files viewed | **Review changes**

102 src/main/thrift/parquet.thrift — Viewed

```
@@ -191,6 +191,52 @@ enum FieldRepetitionType {
191 191    REPEATED = 2;
192 192  }
193 193
194     + /**
195     +  * A structure for capturing metadata for estimating the unencoded,
196     +  * uncompressed size of data written. This is useful for readers to estimate
197     +  * how much memory is needed to reconstruct data in their memory model and for
198     +  * fine grained filter pushdown on nested structures (the histograms contained
199     +  * in this structure can help determine the number of nulls at a particular
200     +  * nesting level and maximum length of lists).
201     +  */
202     + struct SizeStatistics {
```