Lecture 2

Data Formats & Encoding

Iztok Savnik, FAMNIT

October, 2025.

Sources

- Course:
  - Carnegie Mellon University (CMU)
  - Advanced Database Systems
  - Prof. Andy Pavlo
  - Transparencies
- Papers:
  - Conferences VLDB, SIGMOD, CIDR, etc.
  - Journals ACM TODS, IS, VLDBJ, etc.

Outline

- Storage Models
- Persistent Data Formats

OBSERVATION

- OLAP workloads perform sequential scans on large segments of read-only data.
  - The DBMS only needs to find individual tuples to "stitch" them back together.

- OLTP workloads use indexes to find individual tuples without performing sequential scans.
  - Tree-based indexes (B+Trees) are meant for queries with low selectivity predicates.
  - Also need to accommodate incremental updates.

SEQUENTIAL SCAN OPTIMIZATIONS

- Data Encoding / Compression
- Prefetching
- Parallelization
- Clustering / Sorting
- Late Materialization
- Materialized Views / Result Caching
- Data Skipping
- Data Parallelization / Vectorization
- Code Specialization / Compilation

STORAGE MODELS

A DBMS's storage model specifies how it physically organizes tuples on disk and in memory.

Choice #1: **N-ary Storage Model** (NSM)
Choice #2: **Decomposition Storage Model** (DSM)
Choice #3: **Hybrid Storage Model** (PAX)

N-ARY STORAGE MODEL (NSM)

The DBMS stores (almost) all the attributes for a single tuple contiguously in a single page.

Ideal for OLTP workloads where txns tend to access individual entities and insert-heavy workloads.

- Use the tuple-at-a-time iterator processing model.

NSM database page sizes are typically some constant multiple of 4 KB hardware pages.

- Example: Oracle (4 KB), Postgres (8 KB), MySQL (16 KB)

DECOMPOSITION STORAGE MODEL (DSM)

The DBMS stores a single attribute for all tuples contiguously in a block of data.

Ideal for OLAP workloads where read-only queries perform large scans over a subset of the table's attributes.

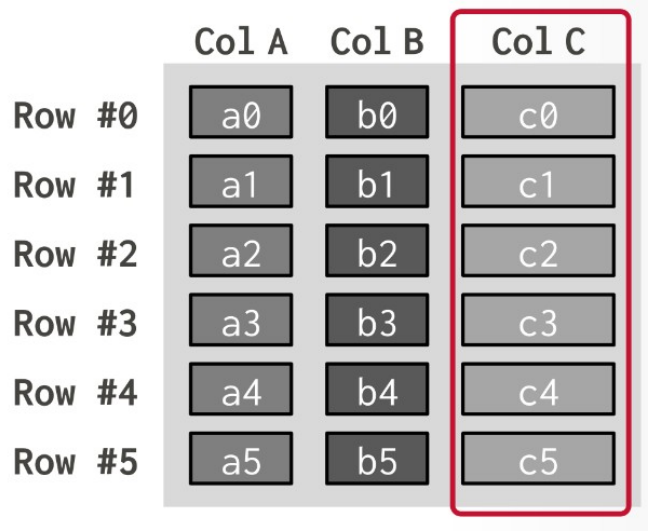
- Use a batched vectorized processing model.

File sizes are larger (100s of MBs), but it may still organize tuples within the file into smaller groups.

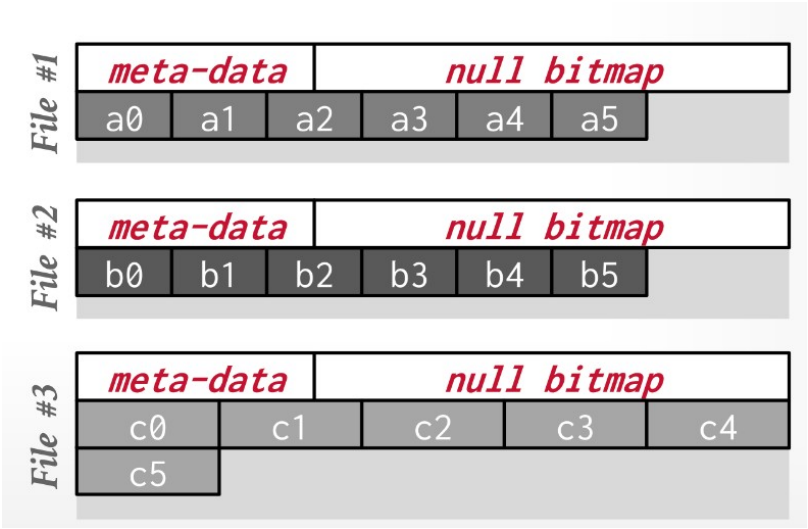
DSM: PHYSICAL ORGANIZATION

Store attributes and meta-data (e.g., nulls) in separate arrays of fixed-length values.

- Most systems identify unique physical tuples using offsets into these arrays.



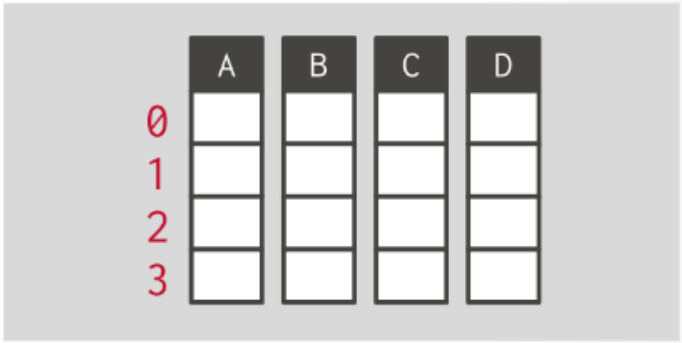
Maintain a separate file per attribute with a dedicated header area for meta-data about entire column.



DSM: TUPLE IDENTIFICATION

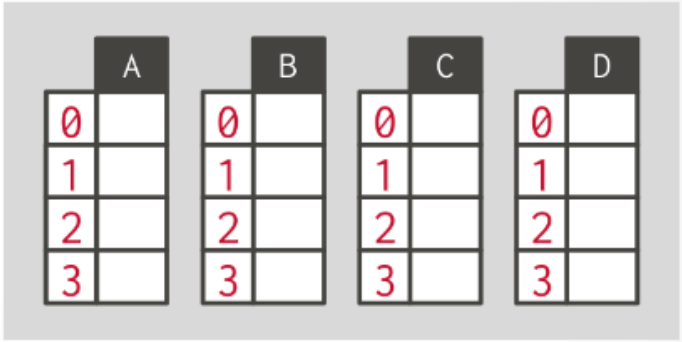
Choice #1: Fixed-length Offsets

- Each value is the same length for an attribute. Use simple arithmetic to jump to an offset to find a tuple.
- Need to convert variable-length data into fixed-length values.



Choice #2: Embedded Tuple Ids

- Each value is stored with its tuple id in a column.
- Need auxiliary data structures to find offset within a column for a given tuple id.



DSM: VARIABLE-LENGTH DATA

Padding variable-length fields to ensure they are fixed-length is wasteful, especially for large attributes.

A better approach is to use dictionary compression to convert repetitive variable-length data into fixed-length values (typically 32-bit integers).

Still need to handle semi-structured data…

OBSERVATION

OLAP queries almost never access a single column in a table by itself.

- At some point during query execution, the DBMS must get other columns and stitch the original tuple back together.

But the DBMS needs to store data in a **columnar format** for storage + execution benefits.

We need columnar scheme that still stores attributes separately but *keeps the data for each tuple physically close to each other*…

PAX STORAGE MODEL

Partition Attributes Across (PAX) is a hybrid storage model that vertically partitions attributes within a database page.

- This is what Paraquet and Orc use.

The goal is to get the benefit of faster processing on columnar storage while retaining the spatial locality benefits of row storage.

Copeland, Khoshafian, A DECOMPOSITION STORAGE MODEL, ACM, 1985.

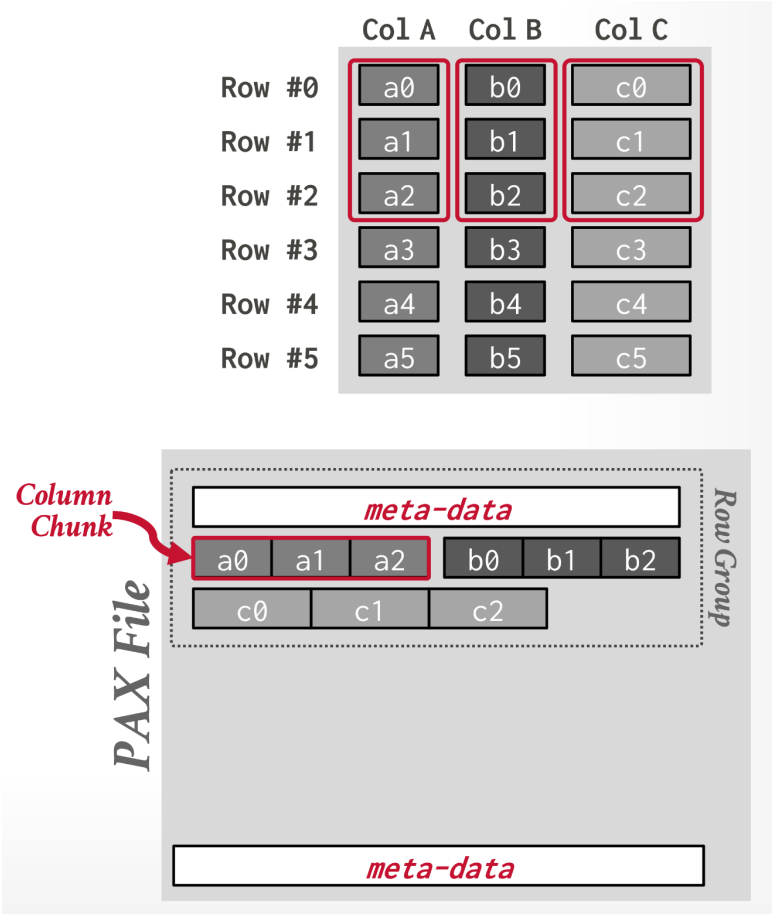
PAX: PHYSICAL ORGANIZATION

Horizontally partition data into row groups. Then vertically partition their attributes into column chunks.

Global meta-data directory contains offsets to the file's row groups.

- This is stored in the footer if the file is immutable (Parquet, Orc).

Each row group contains its own meta-data header about its contents.



OBSERVATION

Most DBMSs use a proprietary on-disk binary file format for persistent data. The only way to share data between systems is to convert data into a common text-based format

- Examples: CSV, JSON, XML

There are open-source binary file formats that make it easier to access data across systems and libraries for extracting data from files.

- Libraries provide an iterator interface to retrieve (batched) columns from files.

OPEN-SOURCE PERSISTENT DATA FORMATS

HDF5 (1998)

- Multi-dimensional arrays for scientific workloads.

Apache Avro (2009)

- Row-oriented format for Hadoop that replace SequenceFiles.

Apache Parquet (2013)

- Compressed columnar storage from Cloudera/Twitter for Impala.

Apache ORC (2013)

- Compressed columnar storage from Meta for Apache Hive.

Apache CarbonData (2016)

- Compressed columnar storage with indexes from Huawei.

Apache Arrow (2016)

- In-memory compressed columnar storage from Pandas/Dremio.

FORMAT DESIGN DECISIONS

File Meta-Data

Format Layout

Type System

Encoding Schemes

Block Compression

Filters

Nested Data

FILE META-DATA

Files are **self-contained** to increase portability.

They contain all the necessary information to interpret their contents without external data dependencies.

Each file maintains global meta-data (usually in its footer) about its contents:

- Table Schema (e.g., Thrift, Protobuf)
- Row Group Offsets / Length
- Tuple Counts / Zone Maps

FORMAT LAYOUT

The most common formats use the PAX storage model that splits data in row groups that contain one or more column chunks.

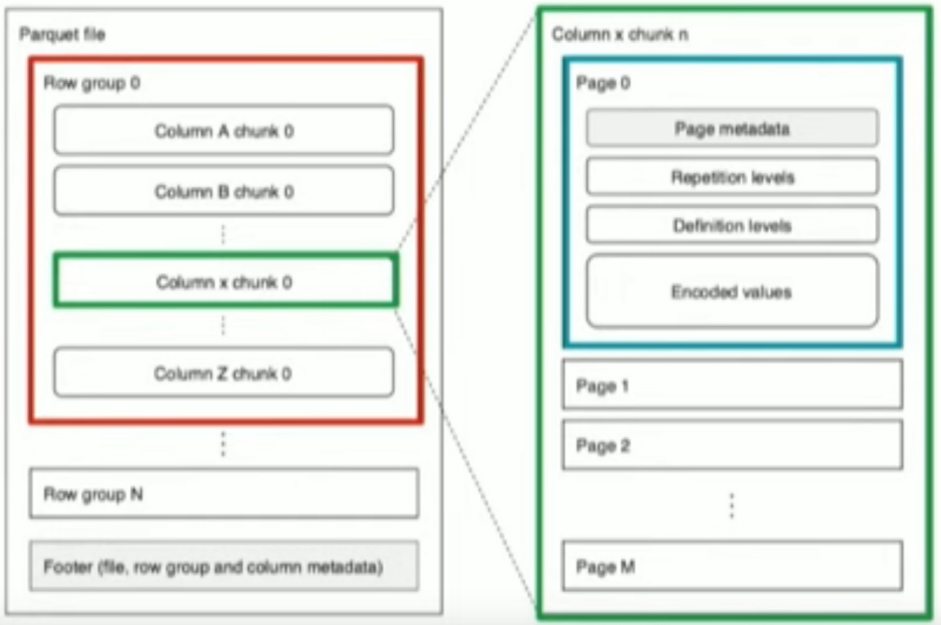
The size of row groups varies per implementation and makes compute/memory trade-offs:

- Parquet: Number of tuples (e.g., 1 million).
- ORC: Physical Storage Size (e.g., 250 MB).
- Arrow: Number of tuples (e.g., 1024*1024).

Parquet: data organization

Data organization

- Row-groups (*default 128MB*)
- Column chunks
- Pages (*default 1MB*)
  - Metadata
    - Min
    - Max
    - Count
  - Rep/def levels
  - Encoded values



From Databrics lecture: www.youtube.com/watch?v=1j8SdS7s_NY&t=705s

TYPE SYSTEM

Defines the data types that the format supports.

- Physical: Low-level byte representation (e.g., IEEE-754).
- Logical: Auxiliary types that map to physical types.

Formats vary in the complexity of their type systems that determine how much upstream producer / consumers need to implement:

- **Parquet**: Minimal # of physical types. Logical types provide annotations that describe interpretation of primitive type data.
- **ORC**: More complete set of physical types.

TYPE SYSTEM - Parquet

The types supported are intended to be as minimal as possible.

- BOOLEAN: 1 bit boolean
- INT32: 32 bit signed ints
- INT64: 64 bit signed ints
- INT96: 96 bit signed ints (deprecated)
- FLOAT: IEEE 32-bit floating point values
- DOUBLE: IEEE 64-bit floating point values
- BYTE_ARRAY: arbitrarily long byte arrays
- FIXED_LEN_BYTE_ARRAY: fixed length byte arrays

TYPE SYSTEM - ORC

Integer
- boolean, tinyint, smallint, int, bigint

Floating point
- float, double

String types
- string, char, varcha

Binary blobs
- binary

Decimal type
- decimal

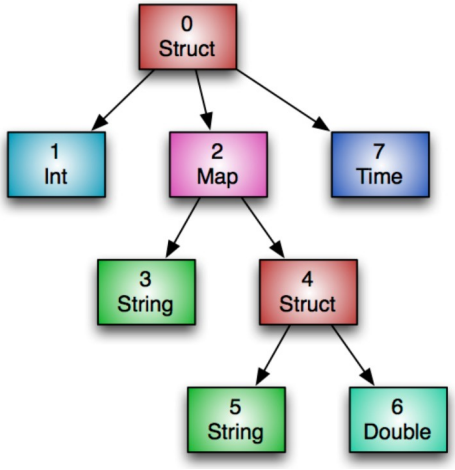
Date/time
- timestamp, timestamp (LD), date

Compound types
- struct, list, map, union

ORC provides a rich set of scalar and compound types:

Example:

```
create table Foobar (
 myInt int,
 myMap map<string,
 struct<myString : string,
 myDouble: double>>,
 myTime timestamp
);
```



ENCODING SCHEMES

An encoding scheme specifies how the format stores the bytes for contiguous/related data.

- Can apply multiple encoding schemes on top of each other to further improve compression.

Dictionary Encoding

Run-Length Encoding (RLE)

Bitpacking

Delta Encoding

Frame-of-Reference (FOR)

DICTIONARY COMPRESSION

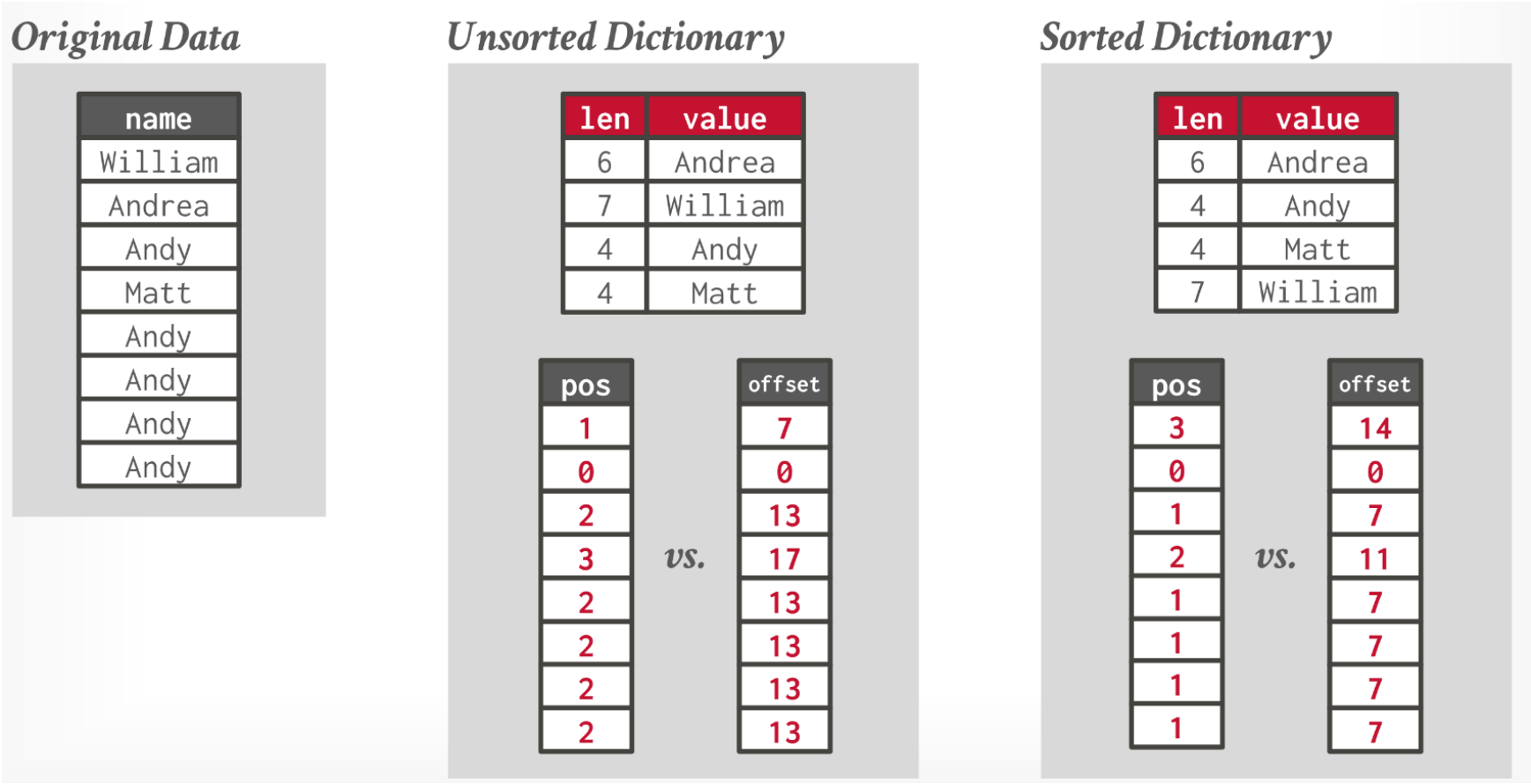
Replace frequent values with smaller fixed-length codes and then maintain a mapping (dictionary) from the codes to the original values.

- Codes could either be positions (using hash table) or byte offsets into dictionary.
- Optionally sort values in dictionary.
- Further compress dictionary and encoded columns.

Format must handle when the number of distinct values (NDV) in a column chunk is too large.

- **Parquet**: Max dictionary size (1 MB).
- **ORC**: Pre-compute NDV and disable if too large.

DICTIONARY COMPRESSION



DICTIONARY COMPRESSION

Design Decision #1: Eligible Data Types

- Parquet: All data types
- ORC: Only strings

Design Decision #2: Compress Encoded Data

- Parquet: RLE + Bitpacking
- ORC: RLE, Delta Encoding, Bitpacking, FOR

Design Decision #3: Expose Dictionary

- Parquet: Move toward supporting
- ORC: Move toward supporting

BLOCK COMPRESSION

Compress data using a general-purpose algorithm.

Scope of compression is only based on the data provided as input.

- LZO (1996), LZ4 (2011), Snappy (2011), Zstd (2015)
  - Snappy (Google) -- speed and simplicity, based on LZ77 family, no entropy coding (Huffman)
  - Zstd (Meta) -- balance speed and compress. ratio, repeating sequences in data, based on LZ77 family, entropy coding, pre-trained dictionary.
  - LZ77 -- Sliding window dictionary (A.Lempel, J.Ziv, 1977)

Considerations:

- Computational overhead
- Compress vs. decompress speed
- Data opaqueness

FILTERS

Zone Maps:

- Maintain min/max values per column at the file-level and row group-level.
- By default, both Parquet and ORC store zone maps in the header of each row group.

Bloom Filters:

- Track the existence of values for each column in a row group. More effective if values are clustered.
- Parquet uses Split Block Bloom Filters from Impala.

NESTED DATA

Real-world data sets often contain semi-structured objects (e.g., JSON, Protobufs).

A file format will want to encode the contents of these objects as if they were regular columns.

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
   Code: 'en-gb'
   Country: 'gb'
```

```
DocId: 20
Name:
  Url: 'http://C'
```

Approach #1: Record Shredding

Approach #2: Length+Presence Encoding

Melnik, et.al., Dremel: A Decade of Interactive SQL Analysis at Web Scale, VLDB, 2020.

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
   Code: 'en-gb'
   Country: 'gb'
```

Shredded Columns

DocID

value	r	d
10	0	0

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
   Code: 'en-gb'
   Country: 'gb'
```

Shredded Columns

DocID

value	r	d
10	0	0

Name.Language.Code

value	r	d
en-us	0	2

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
   Code: 'en-gb'
   Country: 'gb'
```

Shredded Columns

DocID

value	r	d
10	0	0

Name.Language.Code

value	r	d
en-us	0	2

Name.Language.Country

value	r	d
us	0	3

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
   Code: 'en-gb'
   Country: 'gb'
```

Shredded Columns

DocID

value	r	d
10	0	0

Name.Language.Code

value	r	d
en-us	0	2
en	1	2

Name.Language.Country

value	r	d
us	0	3

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
   Code: 'en-gb'
   Country: 'gb'
```

Shredded Columns

DocID

value	r	d
10	0	0

Name.Language.Code

value	r	d
en-us	0	2
en	1	2

Name.Language.Country

value	r	d
us	0	3
NULL	1	2

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
   Code: 'en-gb'
   Country: 'gb'
```

Shredded Columns

DocID

value	r	d
10	0	0

Name.Url

value	r	d
http://A	0	2

Name.Language.Code

value	r	d
en-us	0	2
en	1	2

Name.Language.Country

value	r	d
us	0	3
NULL	1	2

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
   Code: 'en-gb'
   Country: 'gb'
```

Shredded Columns

DocID

value	r	d
10	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2

Name.Language.Code

value	r	d
en-us	0	2
en	1	2

Name.Language.Country

value	r	d
us	0	3
NULL	1	2

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
   Code: 'en-gb'
   Country: 'gb'
```

Shredded Columns

DocID

value	r	d
10	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2

Name.Language.Code

value	r	d
en-us	0	2
en	1	2
NULL	1	1

Name.Language.Country

value	r	d
us	0	3
NULL	1	2
NULL	1	1

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
    Code: 'en-gb'
    Country: 'gb'
```

Shredded Columns

DocID

value	r	d
10	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2

Name.Language.Code

value	r	d
en-us	0	2
en	1	2
NULL	1	1
en-gb	1	2

Name.Language.Country

value	r	d
us	0	3
NULL	1	2
NULL	1	1

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
    Code: 'en-gb'
    Country: 'gb'
```

Shredded Columns

DocID

value	r	d
10	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2

Name.Language.Code

value	r	d
en-us	0	2
en	1	2
NULL	1	1
en-gb	1	2

Name.Language.Country

value	r	d
us	0	3
NULL	1	2
NULL	1	1
gb	1	3

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
   Code: 'en-gb'
   Country: 'gb'
```

Shredded Columns

DocID

value	r	d
10	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2
NULL	1	1

Name.Language.Code

value	r	d
en-us	0	2
en	1	2
NULL	1	1
en-gb	1	2

Name.Language.Country

value	r	d
us	0	3
NULL	1	2
NULL	1	1
gb	1	3

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 20
Name:
  Url: 'http://C'
```

Shredded Columns

DocID

value	r	d
10	0	0
20	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2
NULL	1	1

Name.Language.Code

value	r	d
en-us	0	2
en	1	2
NULL	1	1
en-gb	1	2

Name.Language.Country

value	r	d
us	0	3
NULL	1	2
NULL	1	1
gb	1	3

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 20
Name:
  Url: 'http://C'
```

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

Shredded Columns

DocID

value	r	d
10	0	0
20	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2
NULL	1	1
http://C	0	2

Name.Language.Code

value	r	d
en-us	0	2
en	1	2
NULL	1	1
en-gb	1	2

Name.Language.Country

value	r	d
us	0	3
NULL	1	2
NULL	1	1
gb	1	3

NESTED DATA: SHREDDING

Store paths in nested structure as separate columns with additional meta-data about paths.

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 20
Name:
  Url: 'http://C'
```

Definition Level: How many optional elements are defined in the path to an attribute.

Repetition Level: How many times a structure has been repeated.

Shredded Columns

DocID

value	r	d
10	0	0
20	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2
NULL	1	1
http://C	0	2

Name.Language.Code

value	r	d
en-us	0	2
en	1	2
NULL	1	1
en-gb	1	2
NULL	0	1

Name.Language.Country

value	r	d
us	0	3
NULL	1	2
NULL	1	1
gb	1	3
NULL	0	1

NESTED DATA: LENGTH+PRESENCE

Store paths in nested structure as separate columns but maintain additional columns to

1) track the number of entries at each path level (**length**) and
2) whether a key exists at that level for a record (**presence**).

```
message Document {
  required int64 DocId;
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country;
    }
    optional string Url;
  }
}
```

```
DocId: 10
Name:
  Language:
    Code: 'en-us'
    Country: 'us'
  Language:
    Code: 'en'
  Url: 'http://A'
Name:
  Url: 'http://B'
Name:
  Language:
   Code: 'en-gb'
   Country: 'gb'
```

```
DocId: 20
Name:
  Url: 'http://C'
```

DocId	
value	p
10	true
20	true

Name	
len	
3	
1	

Name.Url	
value	p
http://A	true
http://B	true
	false
http://C	true

Name.Language	
len	
2	
0	
1	
0	

Name.Language.Code	
value	p
en-us	true
en	true
en-gb	true

Name.Language.Country	
value	p
us	true
	false
gb	true

EXPERIMENTAL EVALUATION

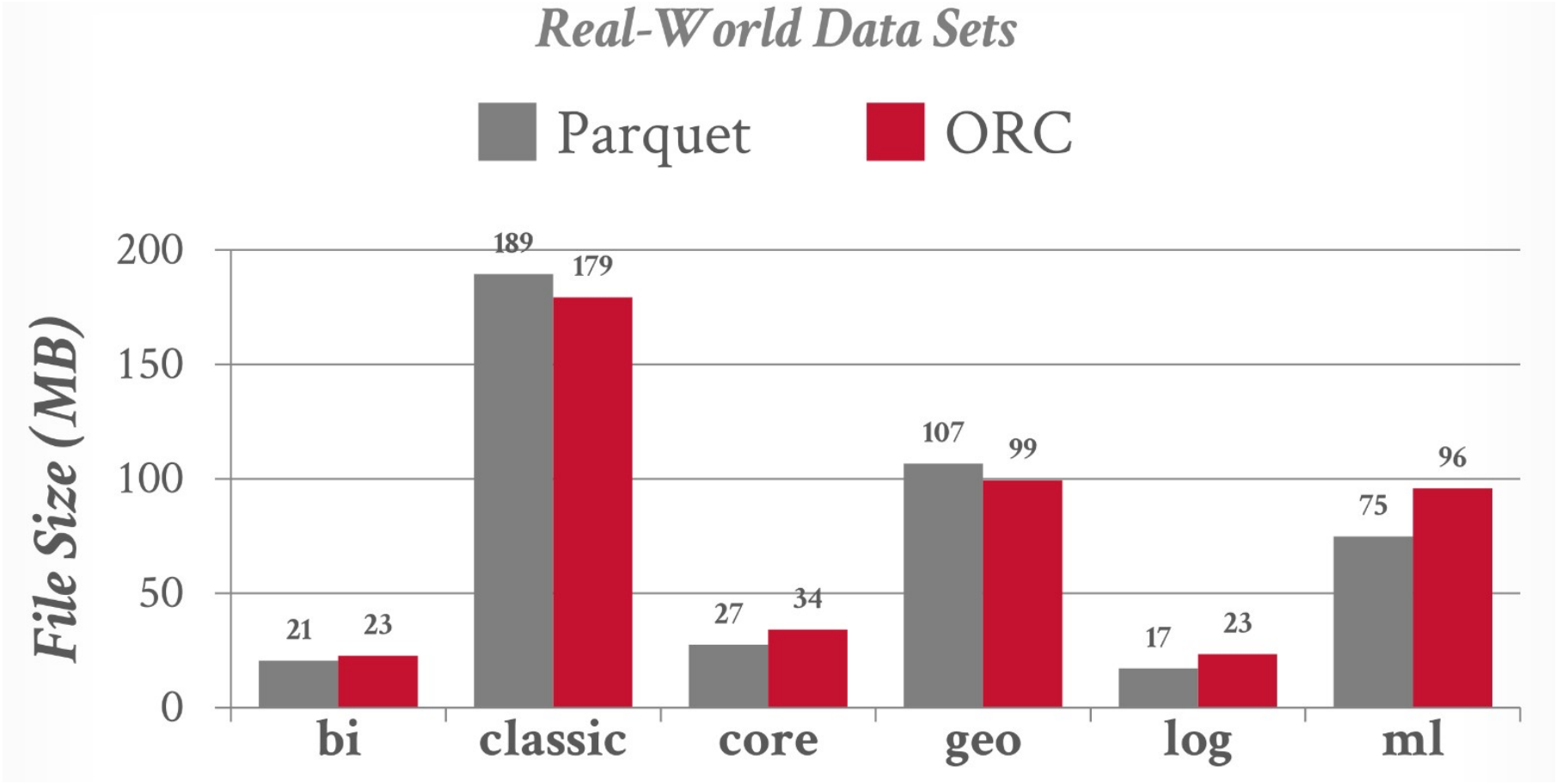
Analyze real-world data sets to extract key properties. Then create a microbenchmark to create synthetic data sets and workloads that vary these properties.

Use Arrow's C++ Parquet/ORC access libraries for most benchmarks.

- Wildly different completeness / optimizations across implementations.

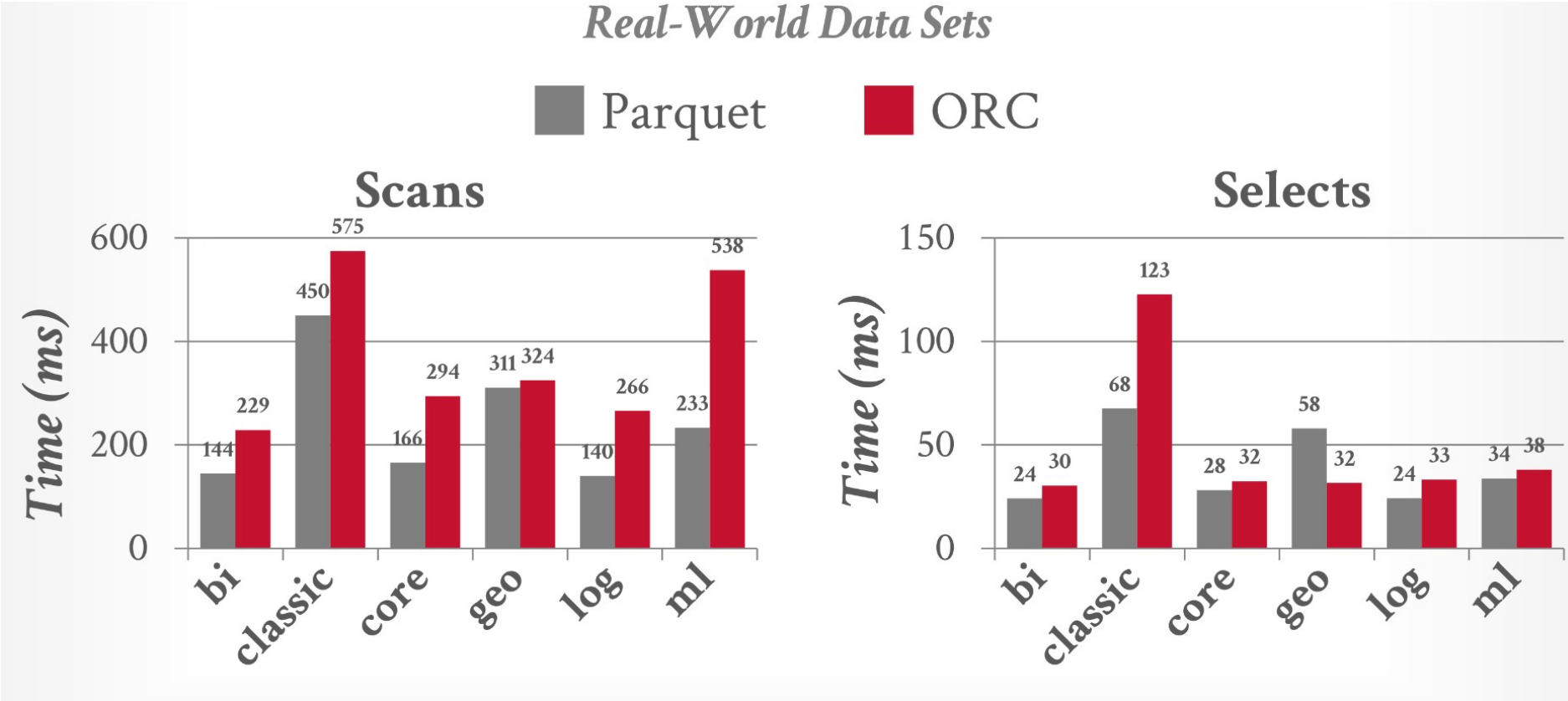
Zeng, et.al., An Empirical Evaluation of Columnar Storage Formats, VLDB23.

COMPRESSION RATIO



Xinyu Zheng: https://xinyuzeng.github.io/

DECODING PERFORMANCE



Xinyu Zheng: https://xinyuzeng.github.io/

LESSONS

Dictionary encoding is effective for all data types and not just strings.

- Real-world data is repetitive and converting arbitrary datato integers in a small domain enables better compression.

Simplistic encoding schemes are better on

modern hardware.

- Determining which encoding scheme a chunk is using at runtime causes branch mispredictions.

Avoid general-purpose block compression.

- Network/disk are no longer the bottleneck relative to CPU performance.

PARTING THOUGHTS

Hardware has changed in the last 10 years that we need to reassess how a DBMS should store data.

Although widely successful and deployed, there are several deficiencies with Parquet/ORC.

- No statistics (e.g., histograms, sketches).
- No incremental schema deserialization.
- Numerous implementations of varying completeness.

NEXT CLASS

- MonetDB/X100 Analysis
- Processing Models
- Plan Processing Direction
- Filter Representation