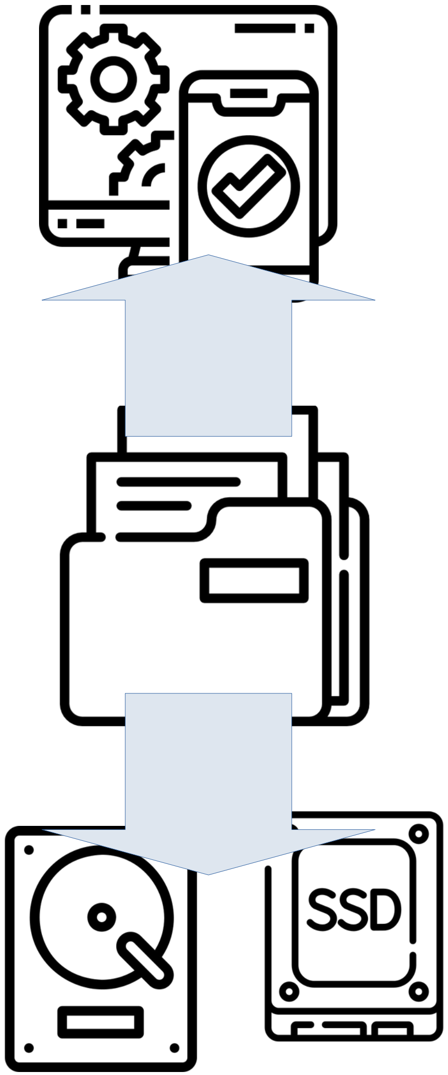
Distributed Data Processing Environments

José Orlando Pereira

Departamento de Informática
Universidade do Minho

Storage stack

- What is stored in files?
- How are files stored?



Icons by Flaticon.com.

Motivation

- Tabular data
- Multiple data types
- Optional (null) values
- No nested or repeated values
- Large number of columns

Id	***Name***	***Location***
1	aa	Braga
2	bbb	Porto
3	cc	Porto
4	dddddd	
5	eee	Lisboa
...	...	...

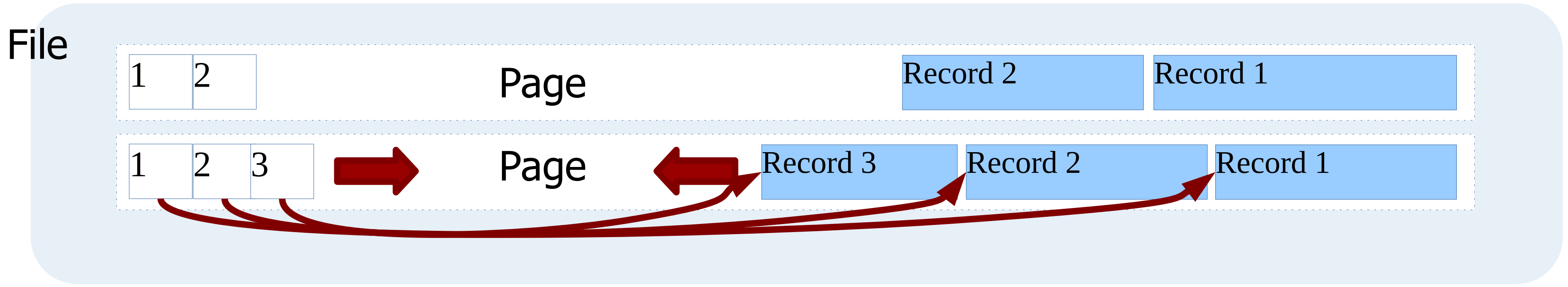
Issues

- Representation of types
  - Compactness and ambiguity
- Data that needs to be moved for:
  - Selection (range scan)
  - Projection
- Compression
- Updates

Database systems

- File format is tightly coupled to system internals
  - Optimized for direct access and caching
  - Not portable, often, even between different versions and processor architectures
- Metadata stored in schema

Row layout



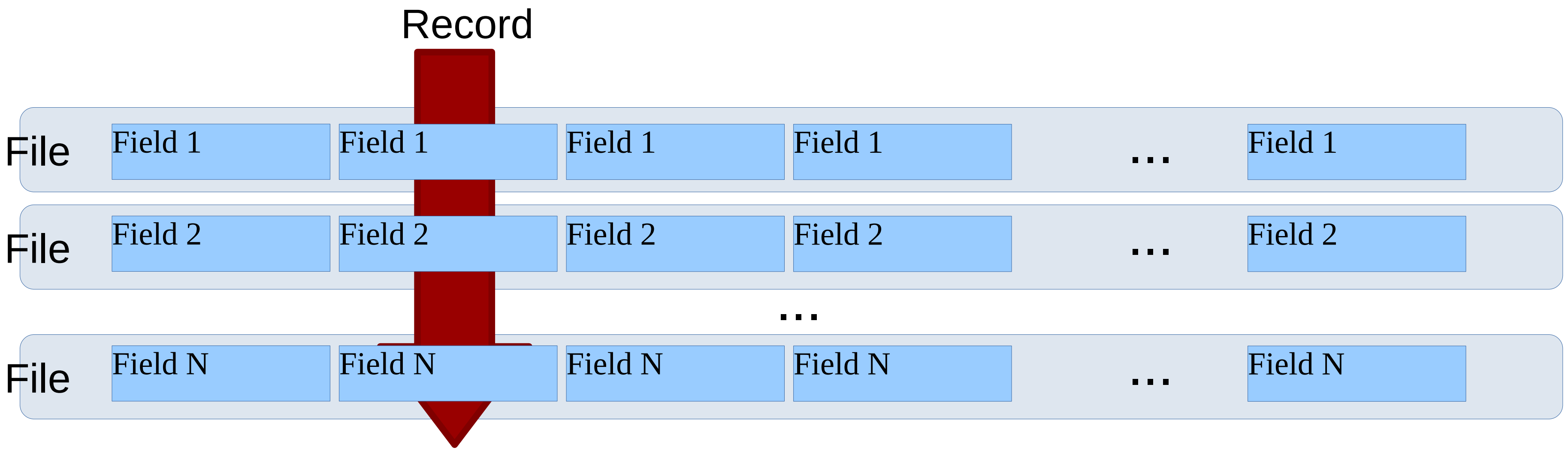
- Page based
  - Two stacks: Offset table and records
  - Fields packed in each record
- Reference for indexing:
  - (Block offset , Record index)

Consequences

- Efficient direct access to individual records
- I/U/D impact only one page
- Cannot avoid reading the full width of the table when scanning a column
  - e.g. select sum(a) from verylargetable;

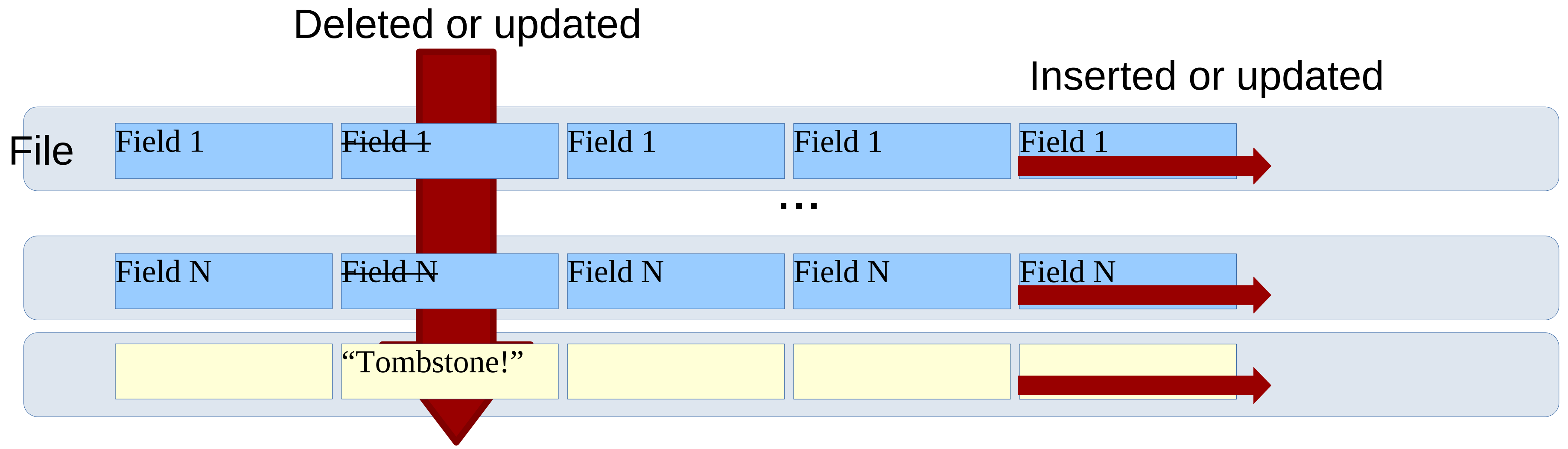
- Good for <u>transactional</u> processing

Columnar layout



- Each file holds a column (i.e., the same field for all rows)

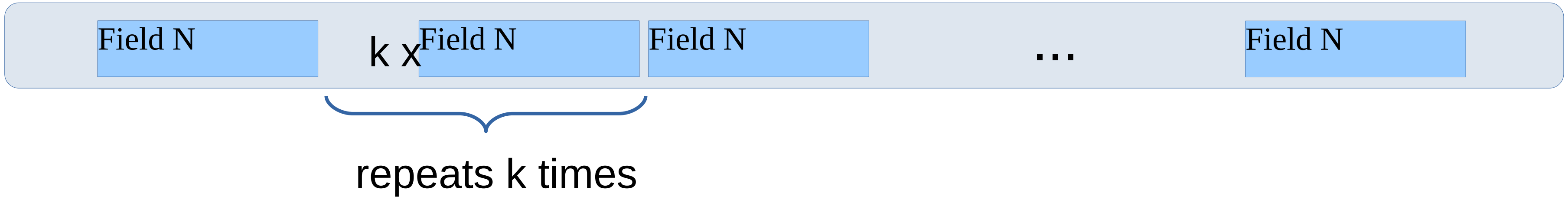
Columnar layout



- Efficient insert (append data)
- Updates and deletes require:
  - "Tombstone" column
  - Re-writing the file

Columnar layout

- Sequences of fields of the same data type can be compressed easily and efficiently:
  - Run length encoding
  - Bitmaps and dictionaries
- Operations on compressed data improve efficiency on all levels of the memory hierarchy:



Consequences

- Column stores: Update/delete are costly operations
- Inserts are cheap by appending to files
- Efficient scan of a subset of columns from a large table
- Data already in a compact vector format

- Better for analytical processing

Autonomous files

- File format is decoupled from processing systems
  - Optimized for exchange and storage in a diversity of media
  - Standardized and portable
- Absent or explicitly included metadata
  - Need for external metadata management

Text (CSV)

- Simple to produce and consume
- Schema can be inferred
- Redundancy and verbose representation (numbers)
- Ambiguity in separators and missing fields
- Difficult to page, especially when compressed

Id	***Name***	***Location***
1	aa	Braga
2	bbb	Porto
3	cc	Porto
4	dddddd	
5	eee	Lisboa
...	...	...

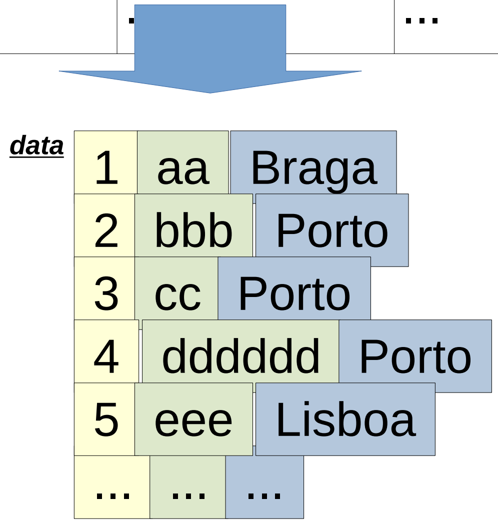
data.csv

```
“1”,”aa”,”Braga”
“2”,”bbb”,”Porto”
“3”,”cc”,”Porto”
“4”,”dddddd”,
“5”,”eee”,”Lisboa”
…,…,…
```

Binary rows

- Compact and unambiguous
- Not ideal for compression, as different data types are interleaved
- All data is read for projections and selections

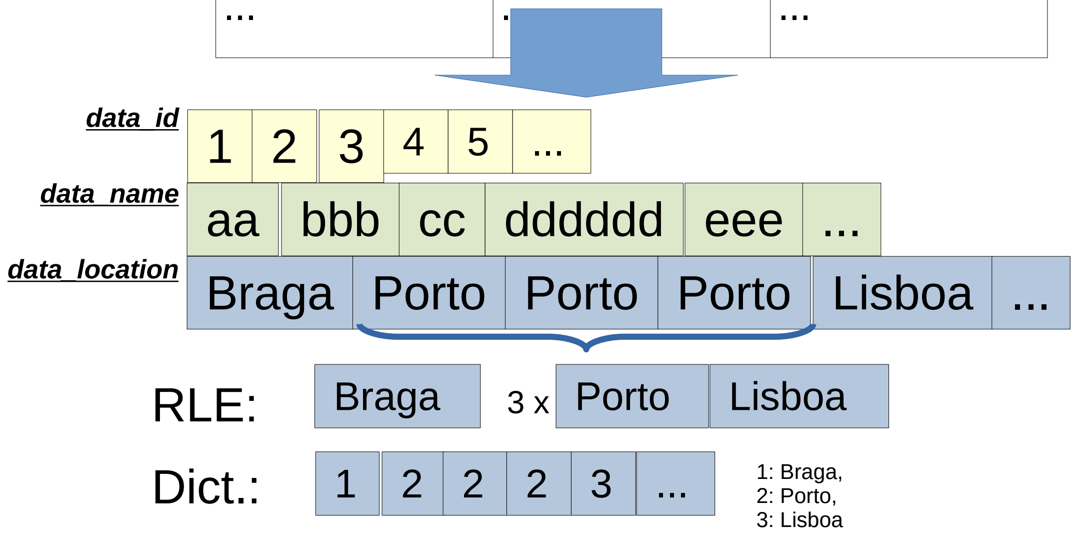
Id	***Name***	***Location***
1	aa	Braga
2	bbb	Porto
3	cc	Porto
4	dddddd	
5	eee	Lisboa
...	...	...



Columnar

- Efficient projections
- Compressed very efficiently
  - Dictionary and/or
  - Run Length Encoding (RLE)
- Inefficient range scan

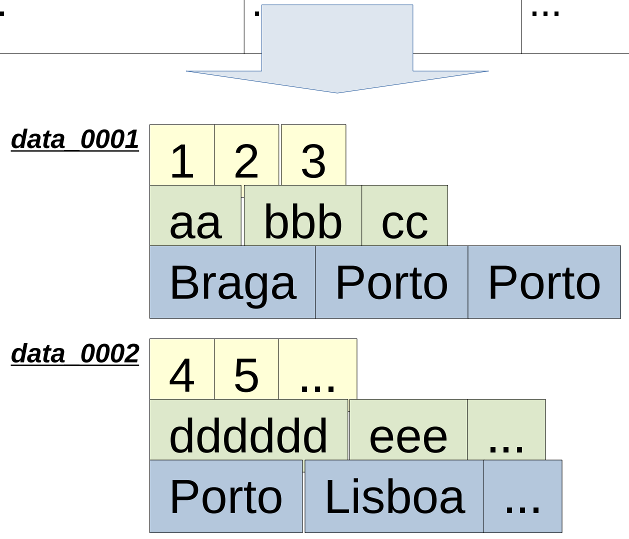
Id	***Name***	***Location***
1	aa	Braga
2	bbb	Porto
3	cc	Porto
4	dddddd	
5	eee	Lisboa
...	...	...



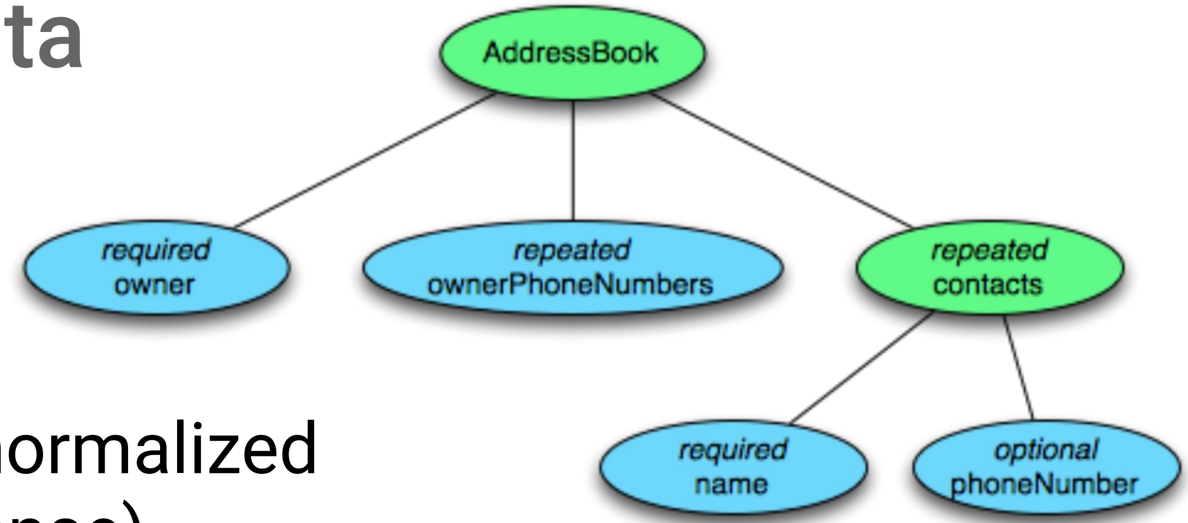
Hybrid

- Columnar segments, that can be accessed and compressed separately
- Good trade-off:
  - I/U/D updates only one segment
  - Range scans can read only some segments
  - Projections can easily skip columns

Id	*Name*	*Location*
1	aa	Braga
2	bbb	Porto
3	cc	Porto
4	dddddd	
5	eee	Lisboa
...	...	...



Hierarchical data



- Data that is not normalized (in a relational sense)
  - Nested structures
  - Repeated fields
- Useful as it avoids multiple files and foreign keys

Image from:https://blog.twitter.com/engineering/en_us/a/2013/dremel-made-simple-with-parquet.html

JSON

- Well-known and widely supported
- Row-based
- Not splitable

```
{
  "AddressBook": [
    {
      "owner": "Jason F.",
      "ownerPhoneNumbers": [
        "123456789",
        "987654321"
      ],
      "contacts": [
        { "name":  "John" },
        { "name":  "Joe", "number":  "214365879" }
      ]
    },
    {
      "owner": "Joe G.",
      "ownerPhoneNumbers": [
        "214365879"
      ]
    }
  ]
}
```

Dremel splitting

- One columnar file for each leaf attribute
- How to match records in different columns?
- Avoid additional information: record numbers, keys, …

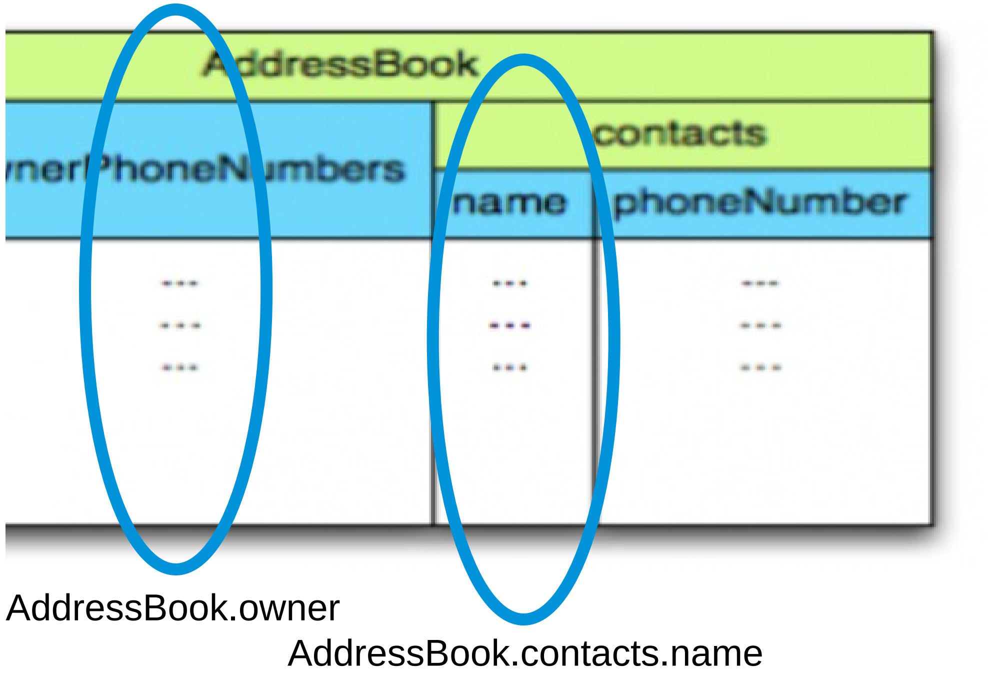
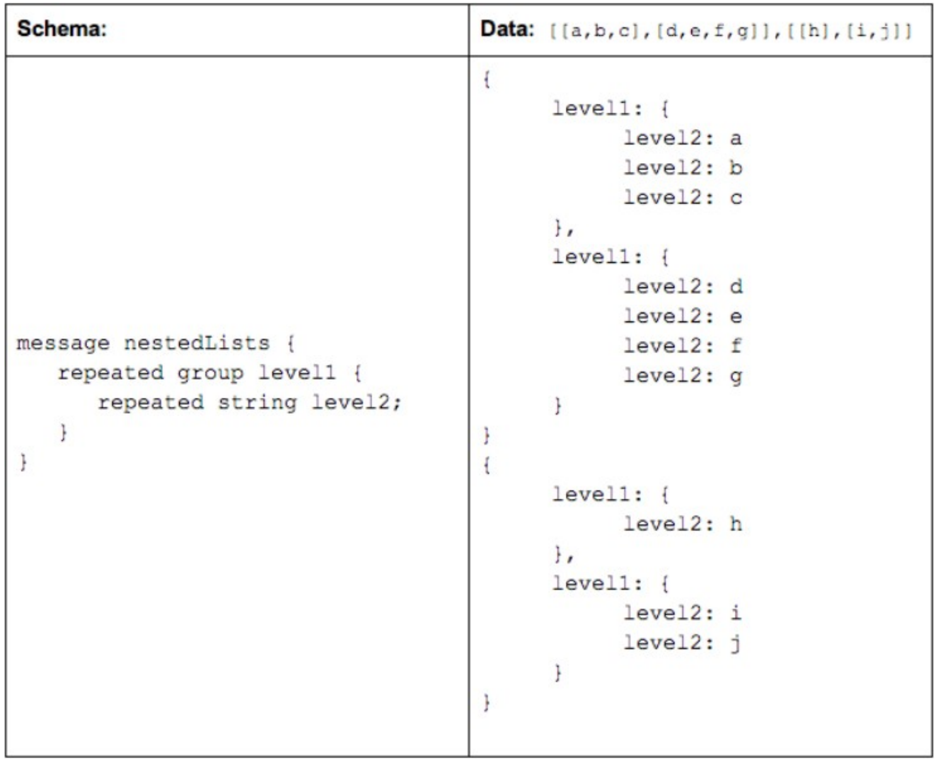


Image from:https://blog.twitter.com/engineering/en_us/a/2013/dremel-made-simple-with-parquet.html

Dremel splitting



Schema:

```
message nestedLists {
   repeated group level1 {
      repeated string level2;
   }
}
```

Data: `[[a,b,c],[d,e,f,g]],[[h],[i,j]]`

```
{
      level1: {
            level2: a
            level2: b
            level2: c
      },
      level1: {
            level2: d
            level2: e
            level2: f
            level2: g
      }
}
{
      level1: {
            level2: h
      },
      level1: {
            level2: i
            level2: j
      }
}
```

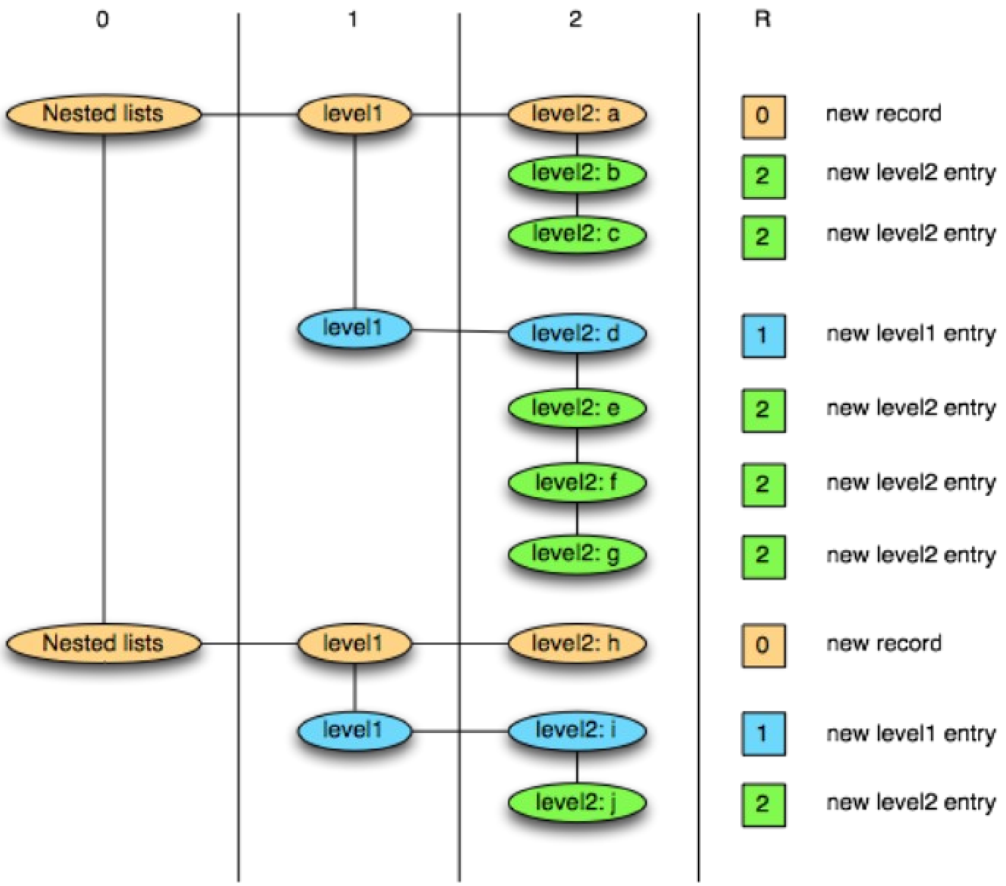


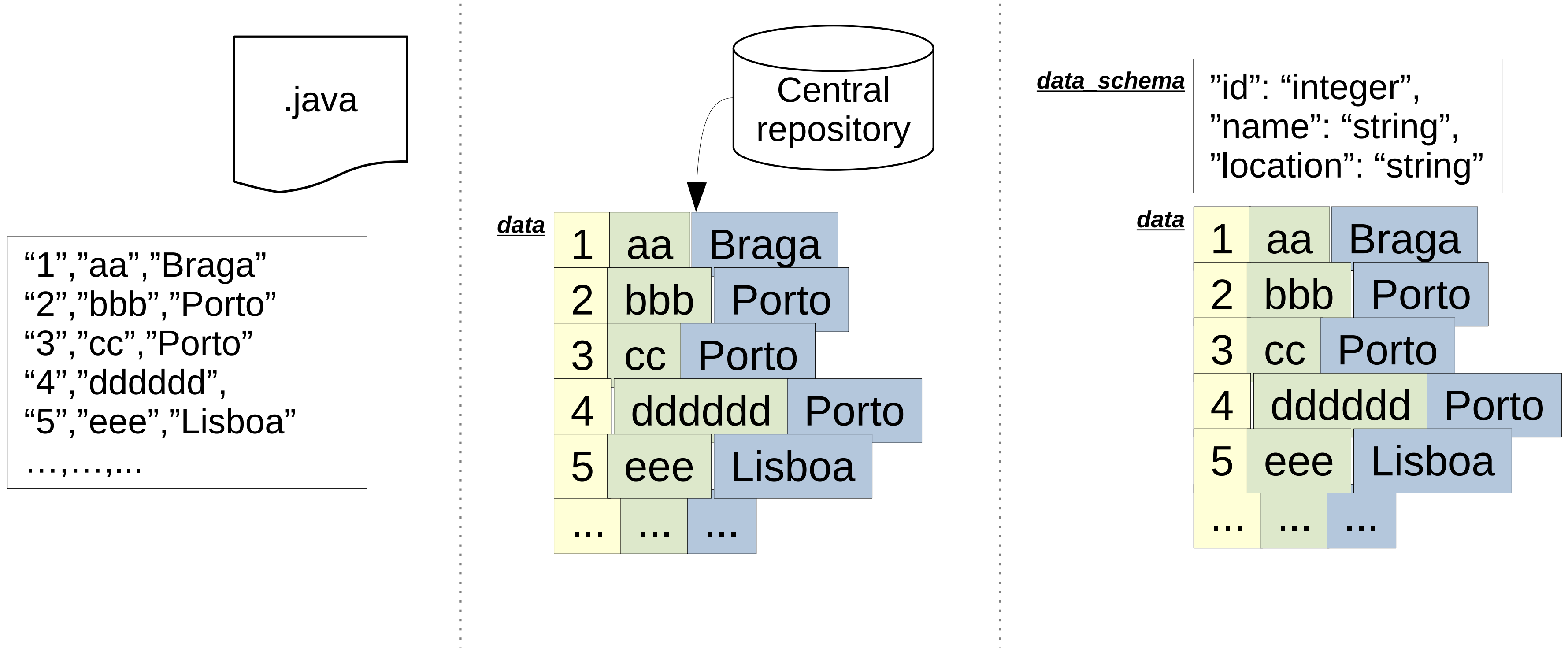
Image from:https://blog.twitter.com/engineering/en_us/a/2013/dremel-made-simple-with-parquet.html

Types of metadata

- <u>Technical</u>
  - Types, representation, …
- <u>Operational</u>
  - Versioning, location (indexing), cardinality, …
- Business
  - What it means, quality, ...

Schema

- Information about data items and types
- Implicit, central or embedded:



```
“1”,”aa”,”Braga”
“2”,”bbb”,”Porto”
“3”,”cc”,”Porto”
“4”,”dddddd”,
“5”,”eee”,”Lisboa”
…,…,...
```

1	aa	Braga
2	bbb	Porto
3	cc	Porto
4	dddddd	Porto
5	eee	Lisboa
...	...	...

```
”id”: “integer”,
”name”: “string”,
”location”: “string”
```

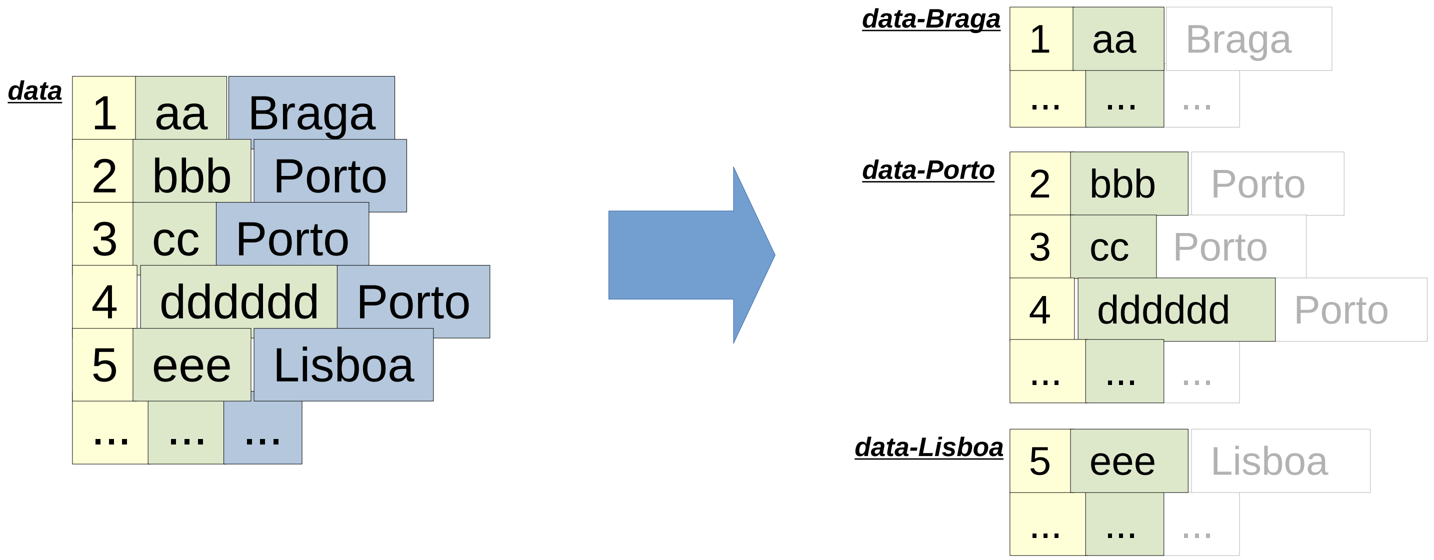
1	aa	Braga
2	bbb	Porto
3	cc	Porto
4	dddddd	Porto
5	eee	Lisboa
...	...	...

Example

```
create external table title_basics (
        tconst string,
        titleType string,
        primaryTitle string,
        originalTitle string,
        isAdult boolean,
        startYear integer,
        endYear integer,
        runtimeMinutes integer,
        genres array<string>)
    row format delimited
    fields terminated by '\t'
    collection items terminated by ','
    lines terminated by '\n'
    stored as textfile
    location 'hdfs://namenode/title_basics'
    tblproperties ("skip.header.line.count"="1");
```

Partitions

- Partition files by a low cardinality column
- Encode partition key in the file name
- Used often with locations and dates
- Useful to avoid reading data

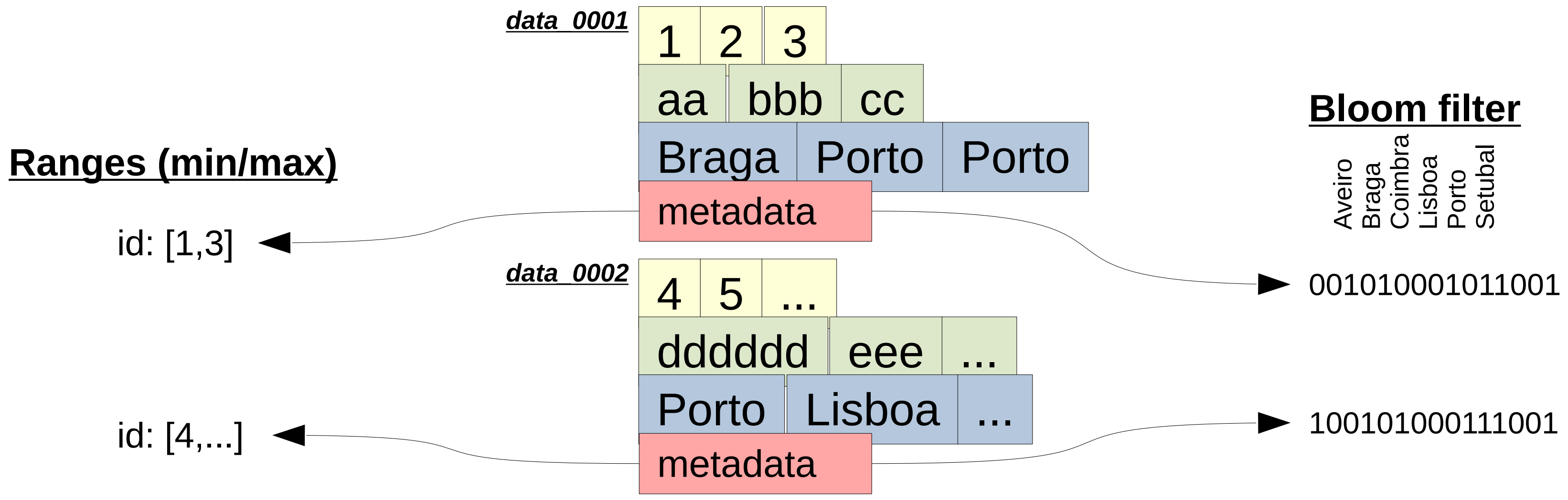


Example

```
create table title_basics_pq (
         tconst string,
         primaryTitle string,
         originalTitle string,
         isAdult boolean,
         startYear integer,
         endYear integer,
         runtimeMinutes integer,
         genres array<string>)
    partitioned by (titleType string)
    stored as parquet;
```

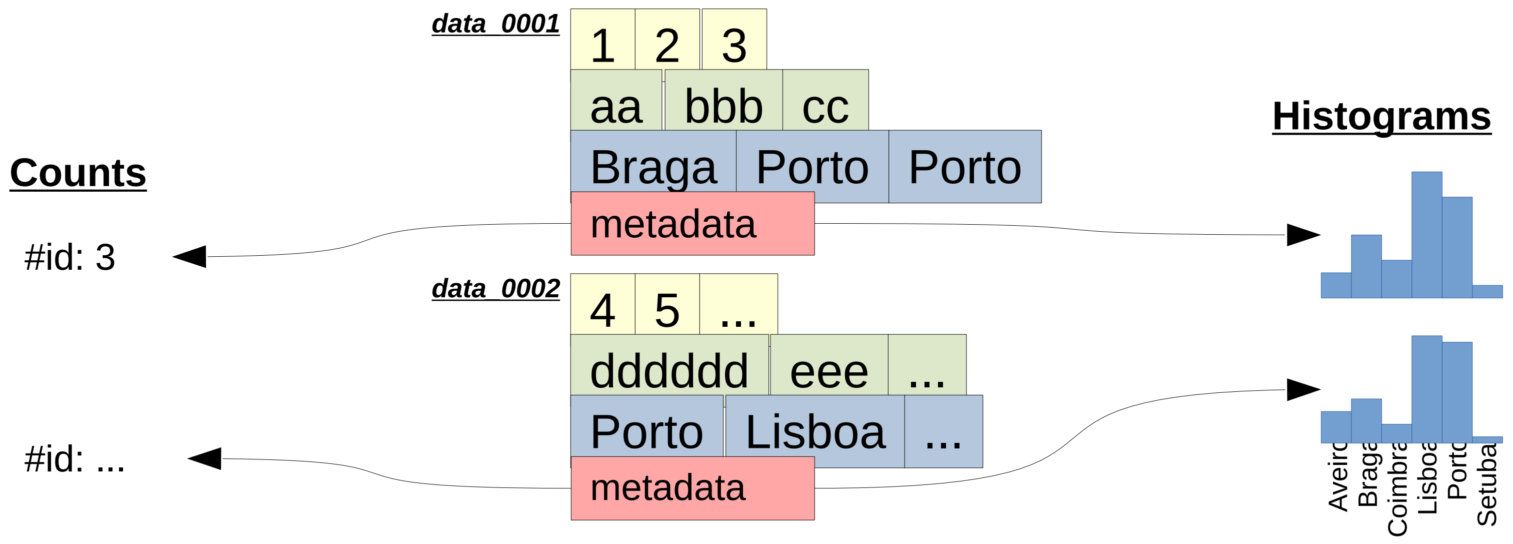
Value summaries / indexes

- Range [min,max] of values in each column
- Compact representation (e.g., Bloom filter) of values in each column
- Useful to avoid reading data



Cardinality summaries

- Number of distinct values in each column
- Compact representation (e.g., histogram) of repetitions of values in intervals, for each column
- Useful to predict how much data will be processed and stored



Hive Metastore

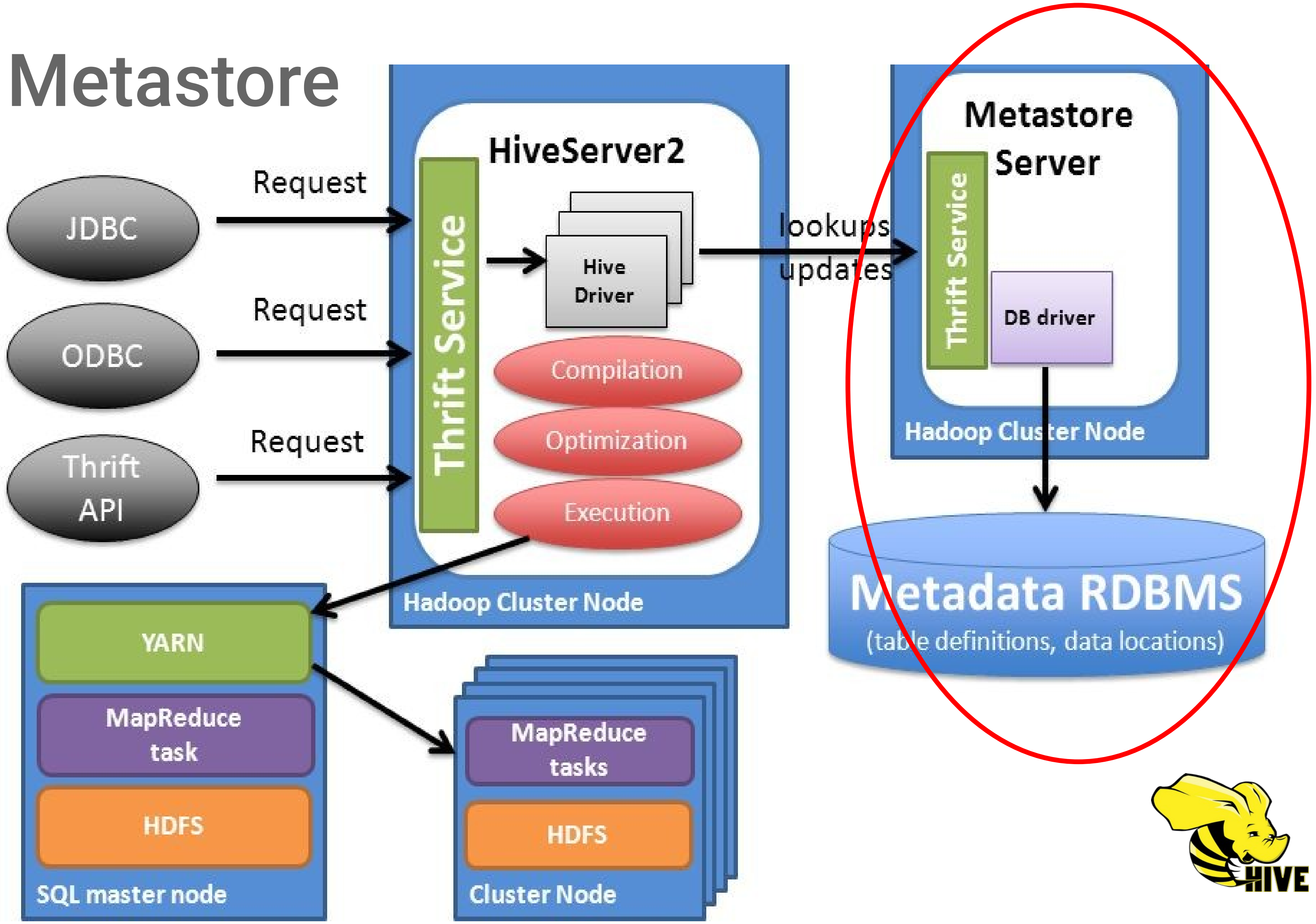


Image from: https://andr83.io/1123/

Hive Metastore

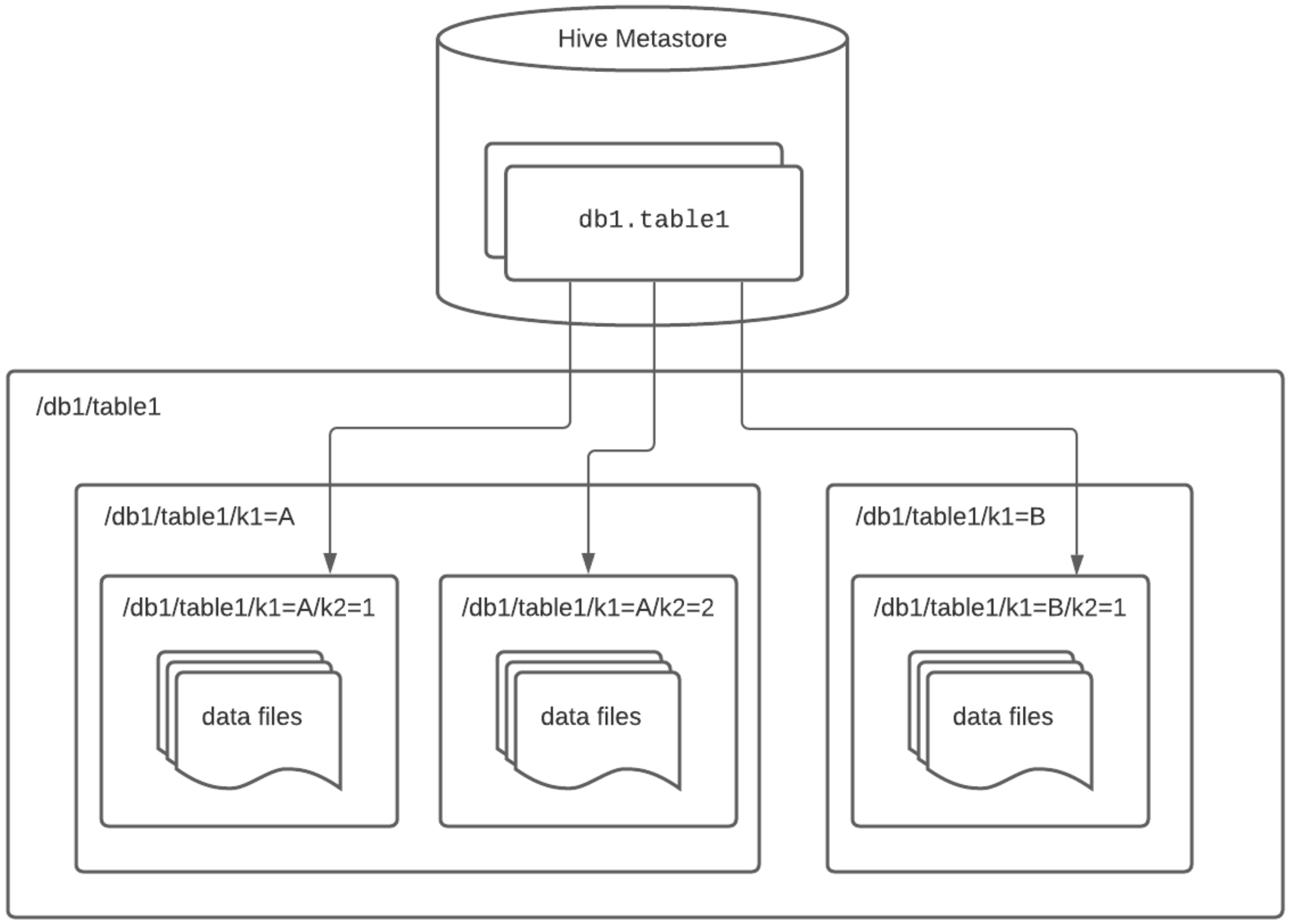
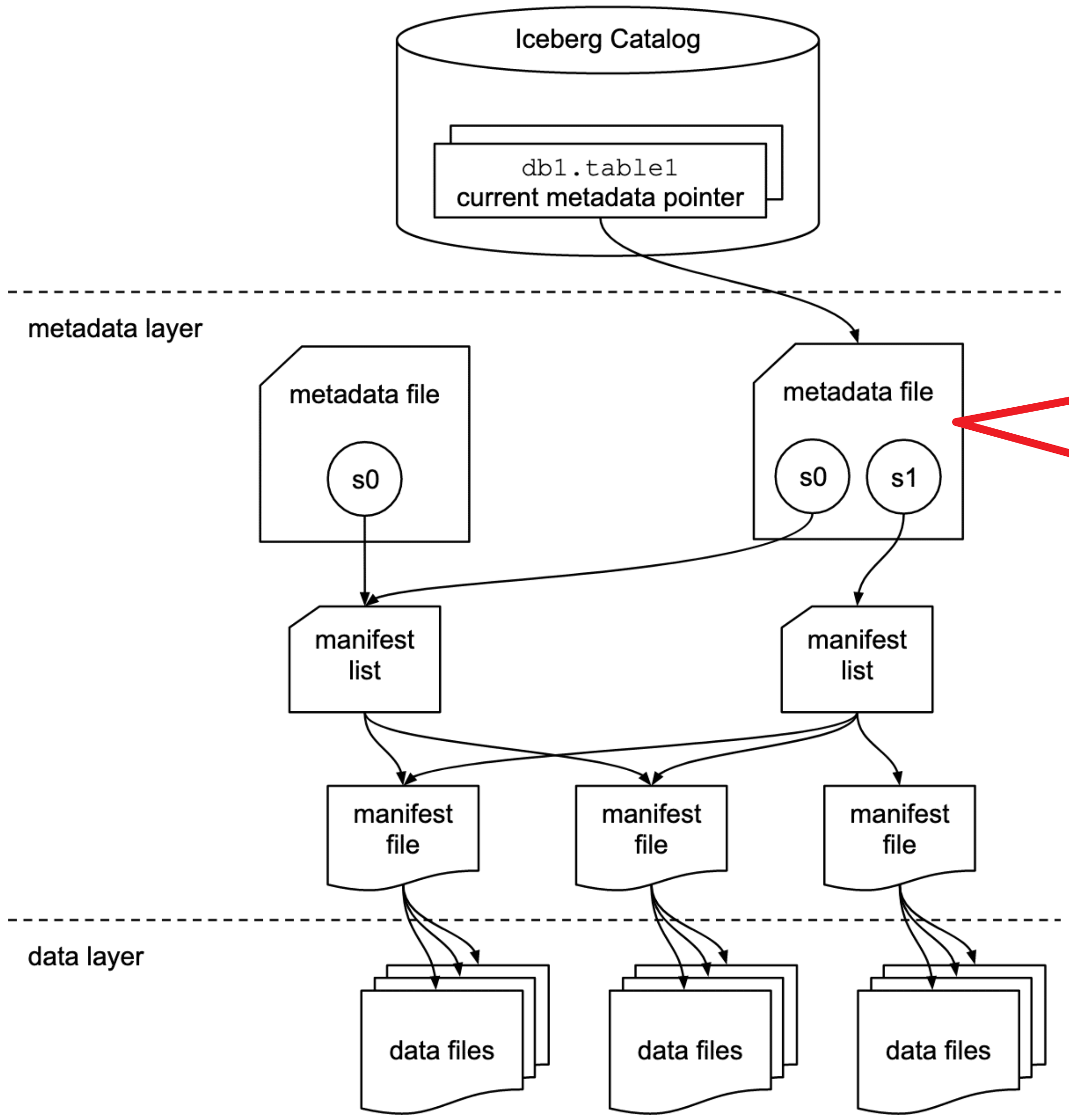


Image from: https://www.dremio.com/resources/guides/apache-iceberg-an-architectural-look-under-the-covers/

Hive Metastore

- Central repository for technical and operational metadata
  - Individual file locations are implicitly defined by partitions
- Enables SQL query processing over data files
  - “Datalake”
- Limited support for updating files
  - Can only add partitions
  - No transactional isolation
  - Statistics can become stale

Iceberg



file location is explicitly stored in files

transactionally isolated updates possible by creating new metadata files

Image from: https://www.dremio.com/resources/guides/apache-iceberg-an-architectural-look-under-the-covers/

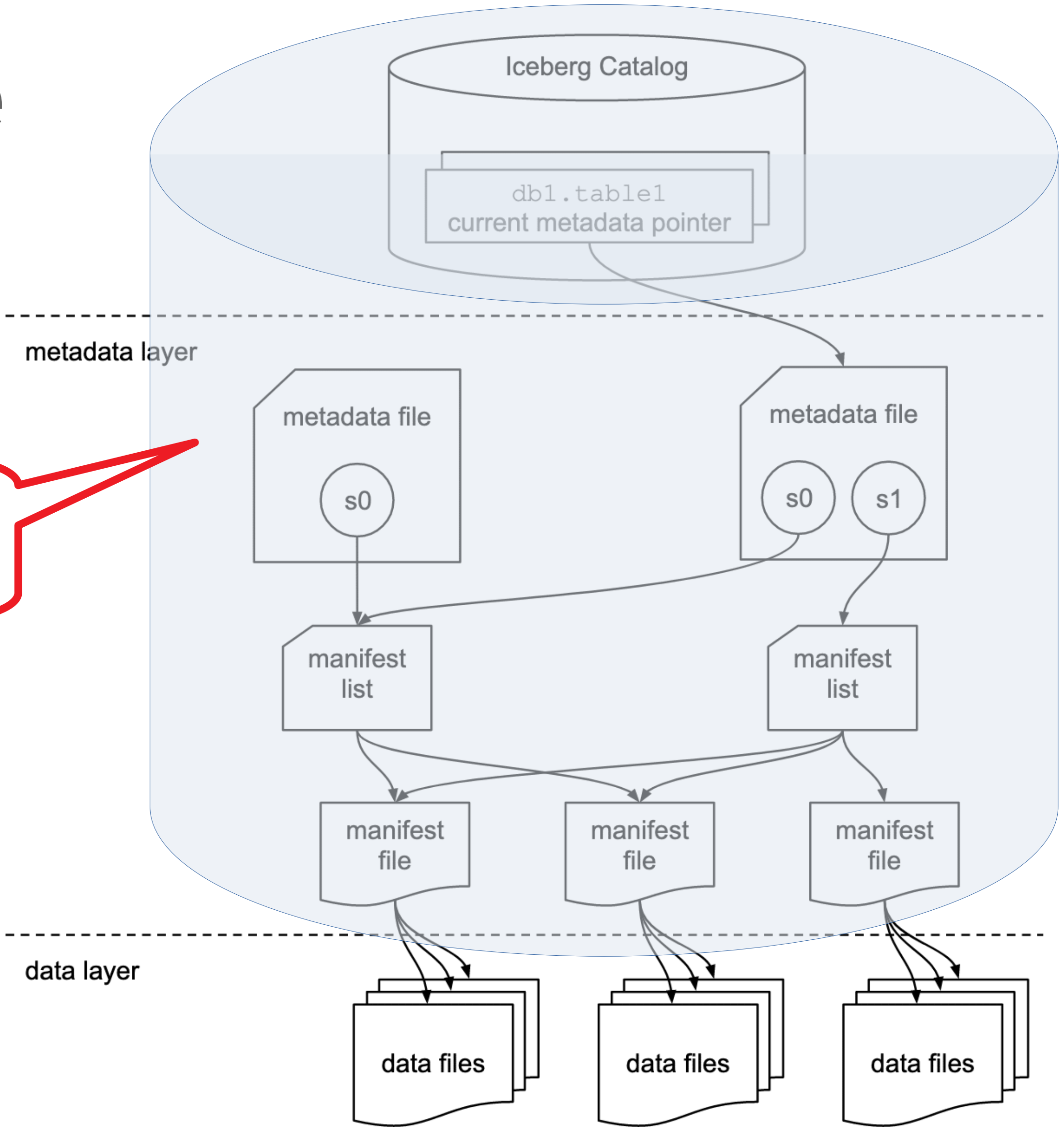
Iceberg

- Metadata pointer identifies the file that contains the most current information
  - Might itself be stored in a file
- A “metadata file” contains a list of snapshots, that described the file at different times
- A “manifest list” contains a list of fragments that exist at the same time
- A “manifest file” decribes an actual physical file
  - Contains statistics
- All information, except the pointer is immutable
  - No inconsistency
  - Can be stored in cloud object stores (e.g. S3)

Transactions on Iceberg

- New file fragments of new versions of existing fragments can be created by I/U/D
  - Corresponding Manifest, Manifest list, and Metadata files are created describing the new version
  - The new version is registered as current, but keeps history of previous snapshots
- Enables data modification
  - “Lakehouse”

DuckLake



all metadata moved back into a database

DuckLake

Image from: https://www.dremio.com/resources/guides/apache-iceberg-an-architectural-look-under-the-covers/

Summary

- Minimizing data movement in analytical operations
  - Physical representation
  - Metadata
- Metadata stores key to interpreting file collections as tables
  - Datalakes
  - Lakehouses