BigData Introdution

Ricardo Vilaça

HASLab/MACC

University of Minho

rmvilaca@di.uminho.pt

Presentation

- https://rmpvilaca.github.io/
- MACC & HASLab Researcher @ UM & INESC TEC
- Research Interests:
  - Energy-efficient scheduling
  - HPC, Big Data, and AI Convergence
  - Edge-Cloud-HPC Computing Continuum
  - Scalable data processing

HASLab INESCTEC

Minho Advanced Computing Center

Roadmap

- BigData and Data Lakes
- Distributed Storage
  - Object Storage
  - File Systems
- File Formats

Big Data

Image from shutterstock

Big Data

- Data Independence
- Full scan vs point queries
  - Projection and selection
- Data denormalization
  - Good for read-intensive scenarios
    - Not having to perform join improves performance

Scale challenges

- Very large amount of data:
  - Disk size
  - Caching performance
- Very large number of queries (reading):
  - Available CPU
- Very large number of queries (writing):
  - Available CPU and disk bandwidth
- Heterogeneous data
  - Nested data
  - Variety of data structures
- I/O bottleneck
  - High disk access time with respect to main memory access time
  - Ever growing processor speeds
- (And high availability...)

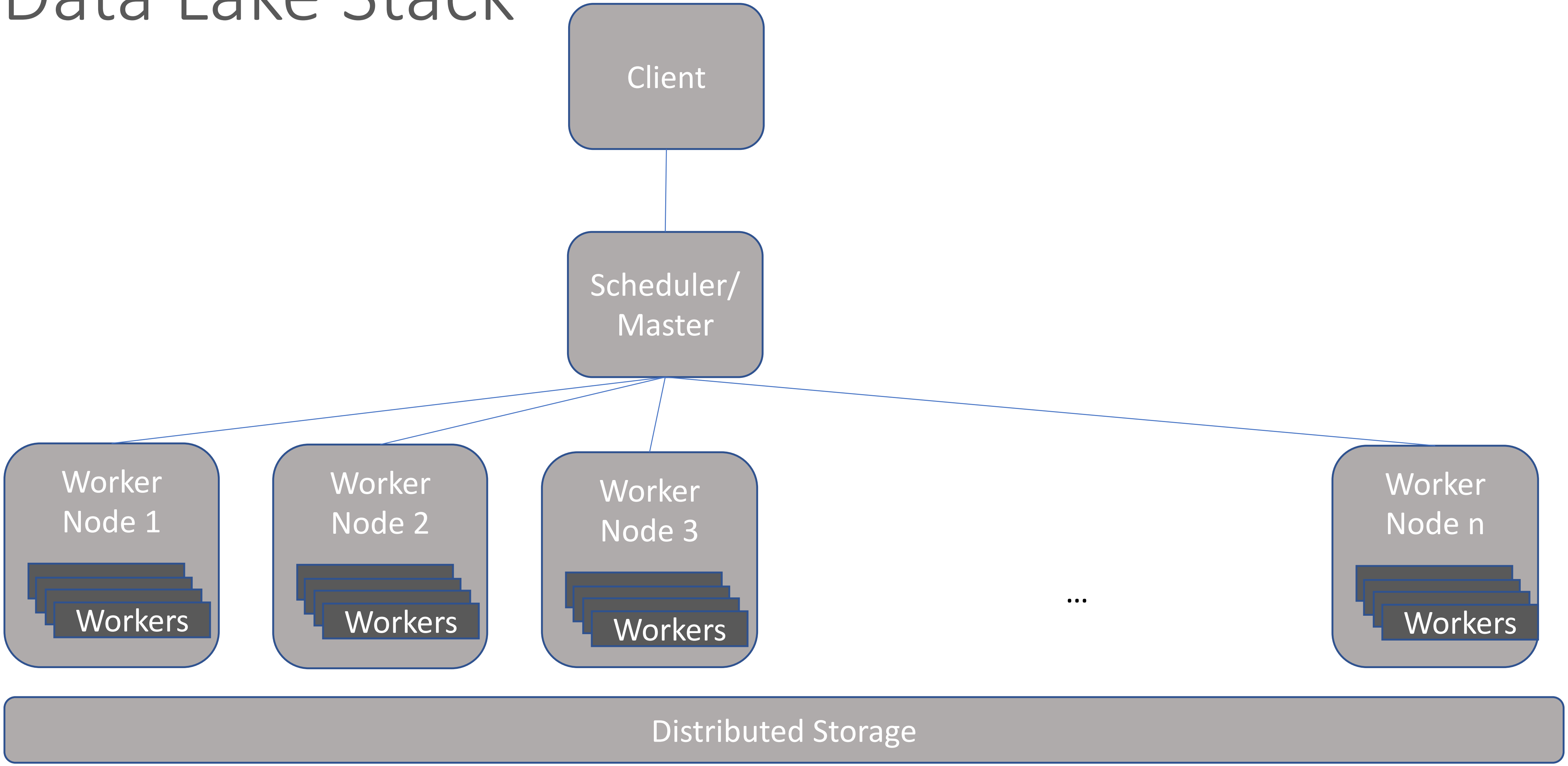
ETL vs ELT

- ETL (Extract, Transform, Load)
  - Imported into the database
  - Transform before load to a data warehouse
  - Schema on write
  - Complex transformations of smaller data sets
- ELT (Extract, Load, Transform)
  - Read from a file system (data lake)
  - Schema on read
  - More flexibility
  - Both structured and unstructured data

Push vs Pull

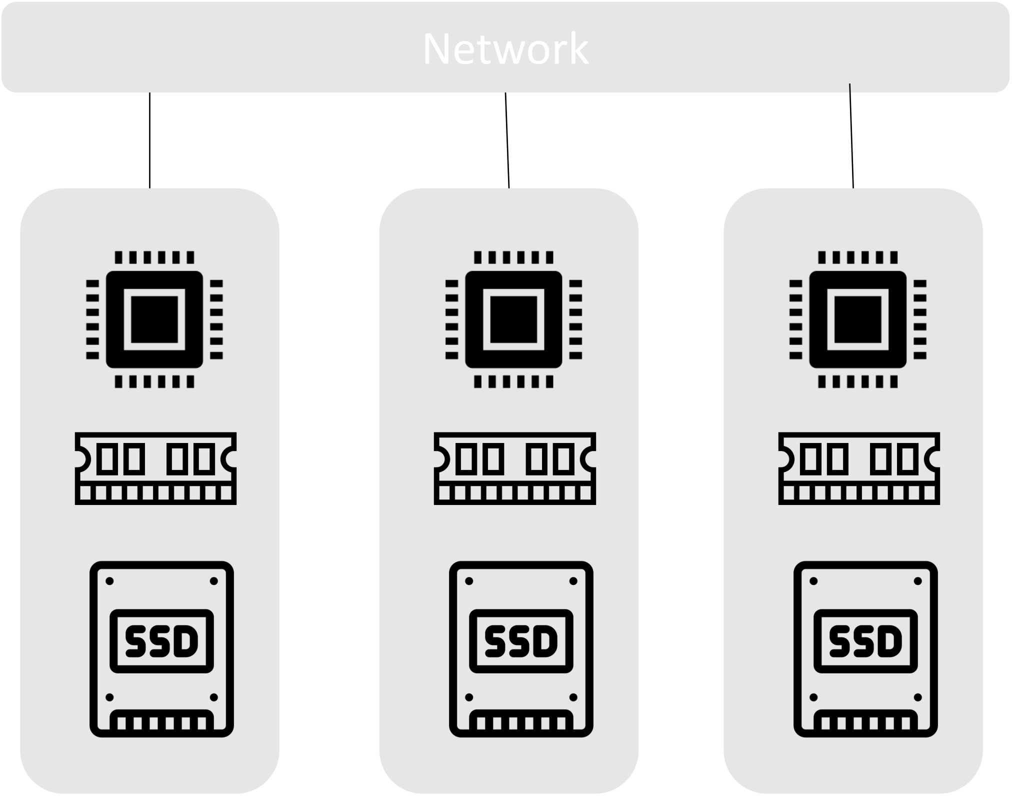
- Push Query to Data
  - Send the query (or a portion of it) to the node that contains the data
  - Perform as much filtering and processing as possible where data resides before transmitting over network
- Pull Data to Query
  - Bring the data to the node that is executing a query that needs it for processing
  - This is necessary when there is no compute resources available where persistent data files are located

Data Lake Stack



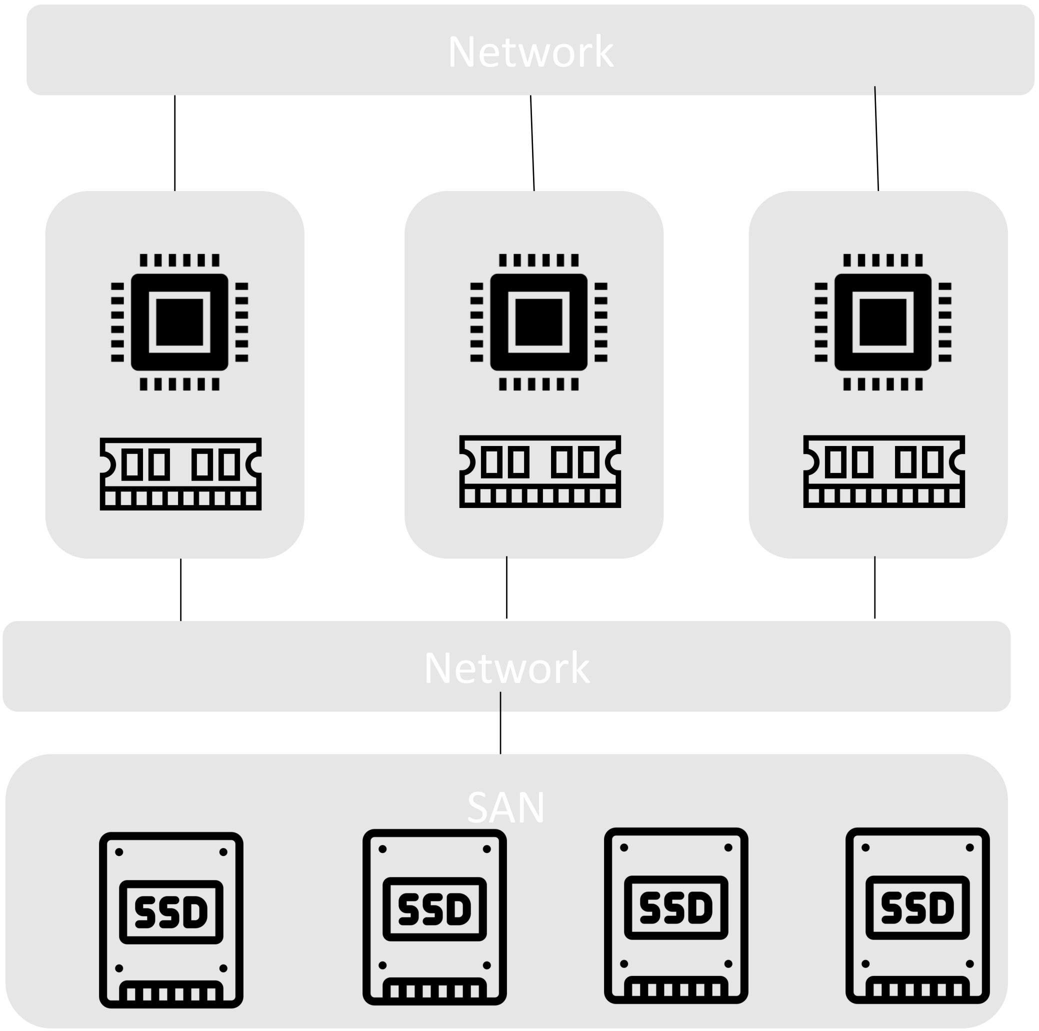
Shared Nothing

- Each instance has its own CPU, memory, locally-attached disk.
- Nodes only communicate with each other via network.
- Data partitioned into disjoint subsets across nodes.
- Cost-effective
  - off-the-shelf components
- Coupling with distributed file systems



Shared Disk

- Each node accesses a single logical disk via an interconnect, but also have their own private memory and ephemeral storage.
  - Must send messages between nodes to learn about their current state
- Cloud Object stores



Shared Nothing vs Shared Disk

- Shared Disk
  - Scale compute layer independently from the storage layer
  - Easy to shutdown idle compute layer resources
  - May need to pull uncached persistent data from storage layer to compute layer before applying filters
- Shared Nothing
  - Dominant parallel architecture for big data systems
  - Harder to scale capacity (data movement)
  - Potentially better performance & efficiency
  - Apply filters where the data resides before transferring

Distributed Challenges

- Nodes fail
  - 1 in 1000 nodes fail a day
    - ***Duplicate Data***
- Network is a bottleneck
  - Typically 1-10 Gb/s throughput
    - **Bring computation to nodes, rather than data to nodes**
- Traditional distributed programming is
  - often ad-hoc and complicated
    - **Stipulate a programming system that can easily be distributed (MapReduce/Spark)**

Distributed Storage

Distributed Storage System

Storing and managing data across the nodes of a cluster

- Object-based
  - Object = ⟨oid, data, metadata⟩
  - Metadata can be different for different object
  - Easy to move
  - Flat object space → billions/trillions of objects
  - Easily accessed through REST-based API (get/put)
- File-based
  - Data in files of fixed- or variable-length records
  - Metadata-per-file stored separately from file
  - For large data, a file needs to be partitioned and distributed

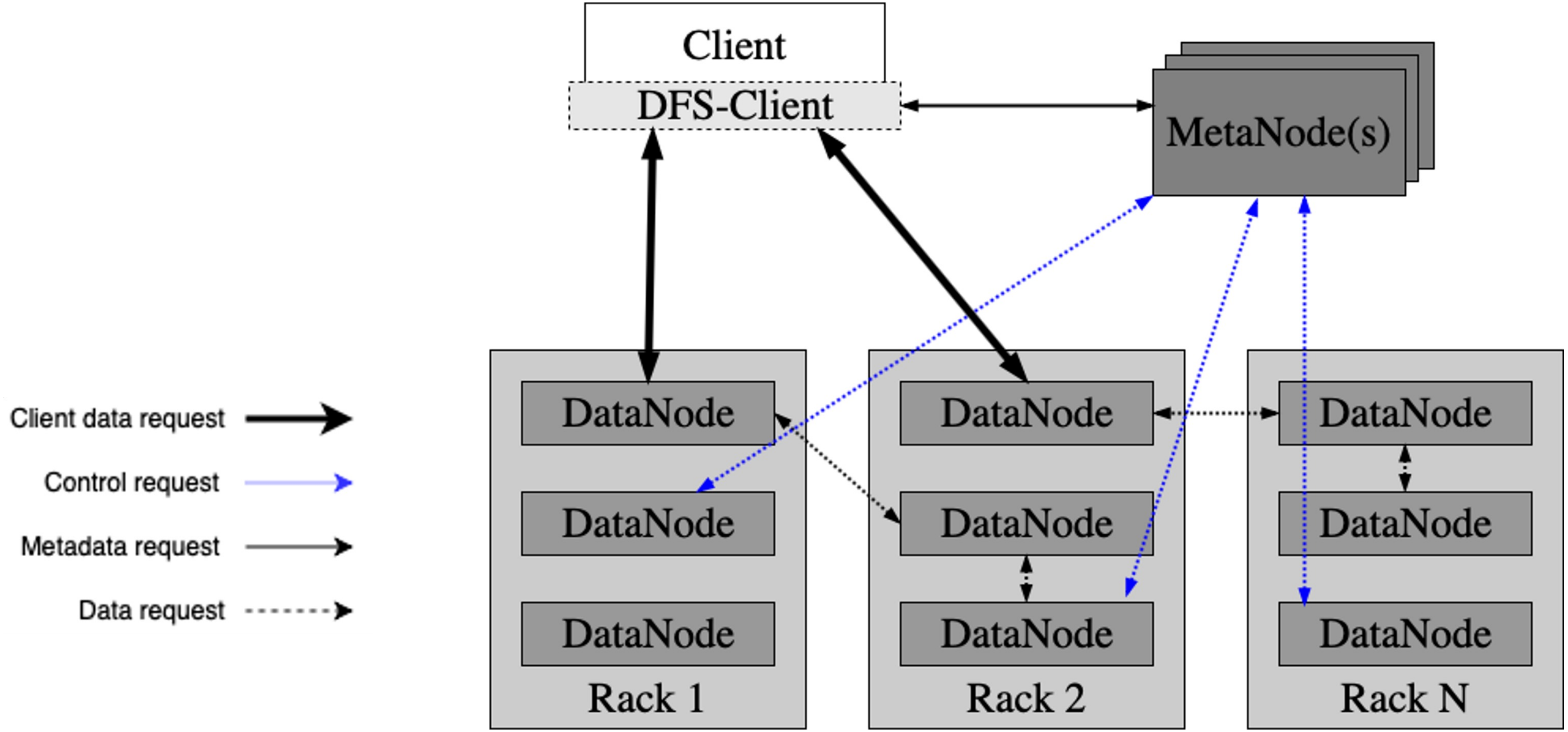
Object Store

- Partition the persistent data into large, immutable files stored in an object store
  - All attributes for a tuple are stored in the same file in a columnar layout (PAX)
  - Header (or footer) contains meta-data about columnar offsets, compression schemes, indexes, and zone maps
- No hierarchy
- Each cloud vendor provides their own proprietary API to access data (PUT, GET, DELETE).
  - Some vendors support predicate pushdown (S3)

Hadoop Distributed File System (HDFS)

- Distributed architecture
- Master/slave topology
- Independent data and metadata handling
- POSIX-like Interface
- Designed for large data sets
- Throughput over latency

HDFS Architecture



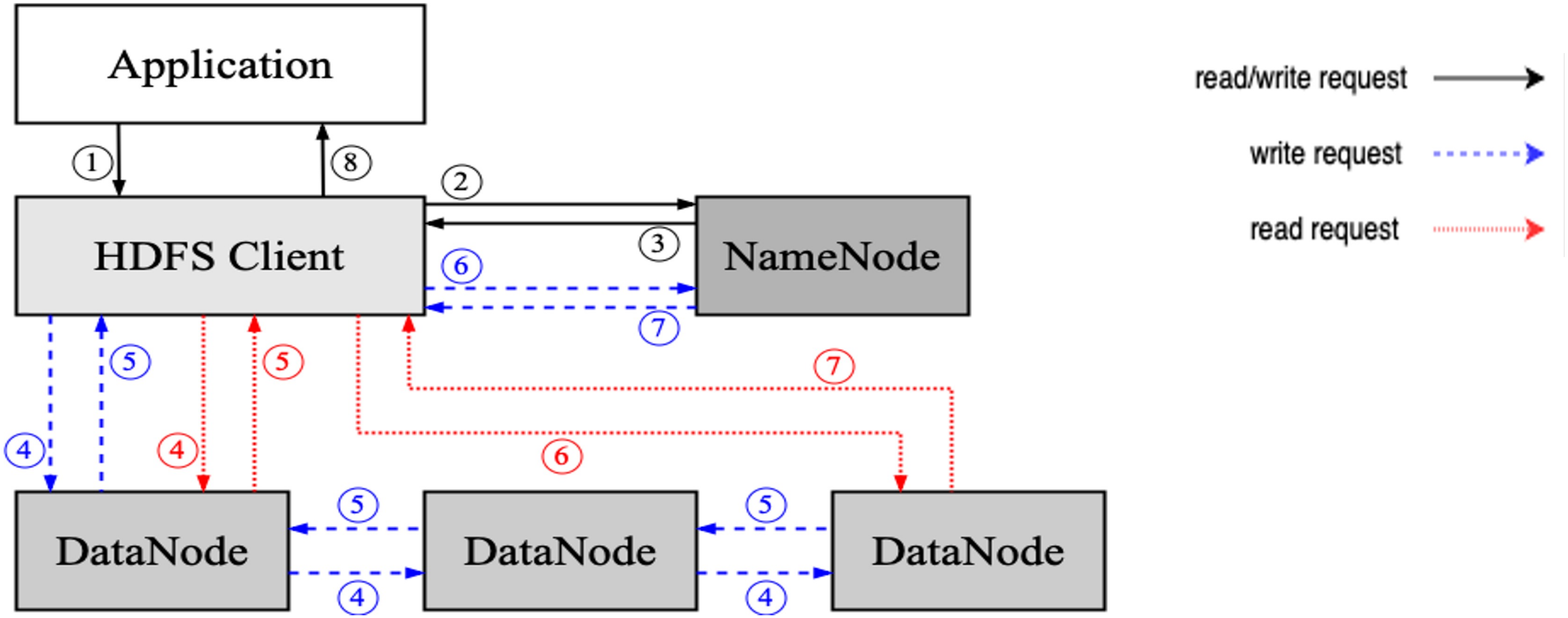
NameNode

- Maintains the file system metadata
  - *e.g.,* namespace, access control, file-to-block mapping, block locations
- Manages and controls system-wide activities (*e.g.,* load balancing, locking)
- HDFS-client <-> NameNode
  - Exchange metadata requests
- NameNode <-> DataNode
  - Periodically exchange block reports and heartbeats

DataNode

- Storage units of the file system
- Store and retrieve files/blocks to HDFS-clients
- Files are split into blocks
  - *dfs.blocksize* is typically 64MiB or 128MiB
  - Blocks are stored across different DataNodes for increased availability and parallelism (defined by *dfs.replication*)
  - Default replication factor is 3

Requests lifecycle



File vs Objects

- File systems
  - tied to a particular hierarchical organisation and storage device
  - files can be modified or accessed arbitrarily
  - size limit determined by underlying storage facility
  - accessed by programs running in a host that mounts the file system
- Objects
  - immutable
  - data storage not bound logically to any storage facility
  - whole-object operations
  - accessible over the Internet

File Formats

Relational Data Format

- 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
...	...	...

Text (CSV)

- Row-oriented
- Simple to produce and consume
- Schema can be inferred
- Redundancy and verbose representation (numbers)
- Ambiguity in separators and missing fields
- Only primitive types
- 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”
…,…,...

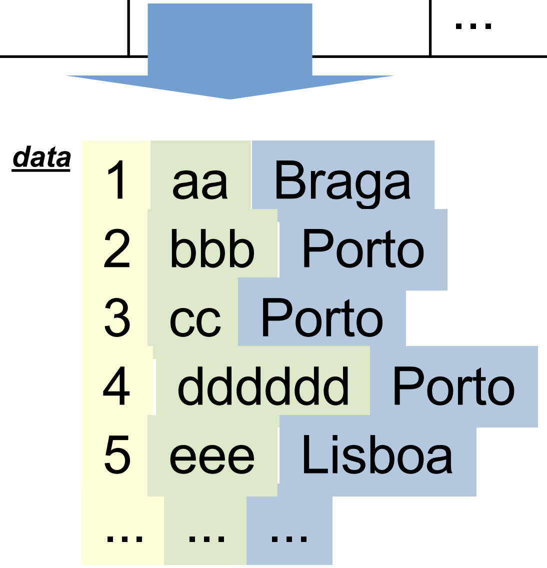
Data Formats Challenges

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

Binary rows

- Compact and unambiguous
- Efficient I/U/D
- Can be paged and compressed
  - Not efficient as different data types are interleaved
- All data is read for projections

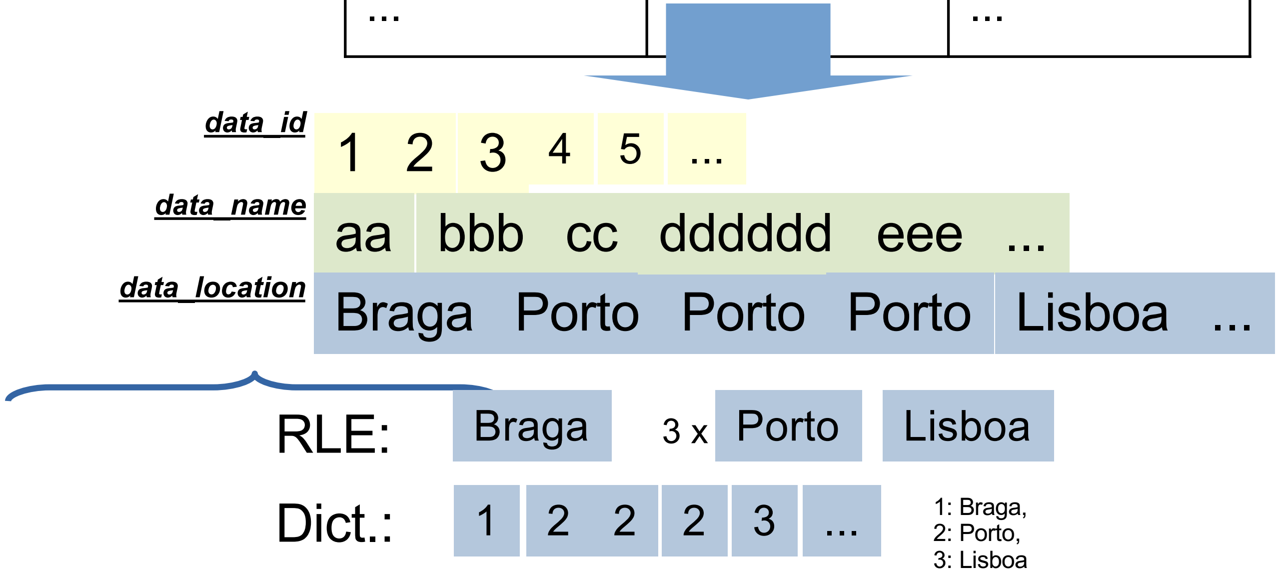
Id	*Name*	*Location*
1	aa	Braga
2	bbb	Porto
3	cc	Porto
4	dddddd	
5	eee	Lisboa
...		...



Columnar

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

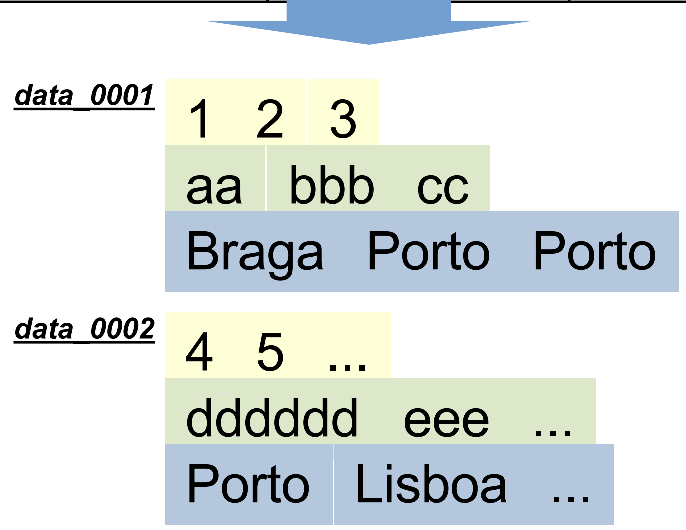
Id	*Name*	*Location*
1	aa	Braga
2	bbb	Porto
3	cc	Porto
4	dddddd	
5	eee	Lisboa
...		...



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

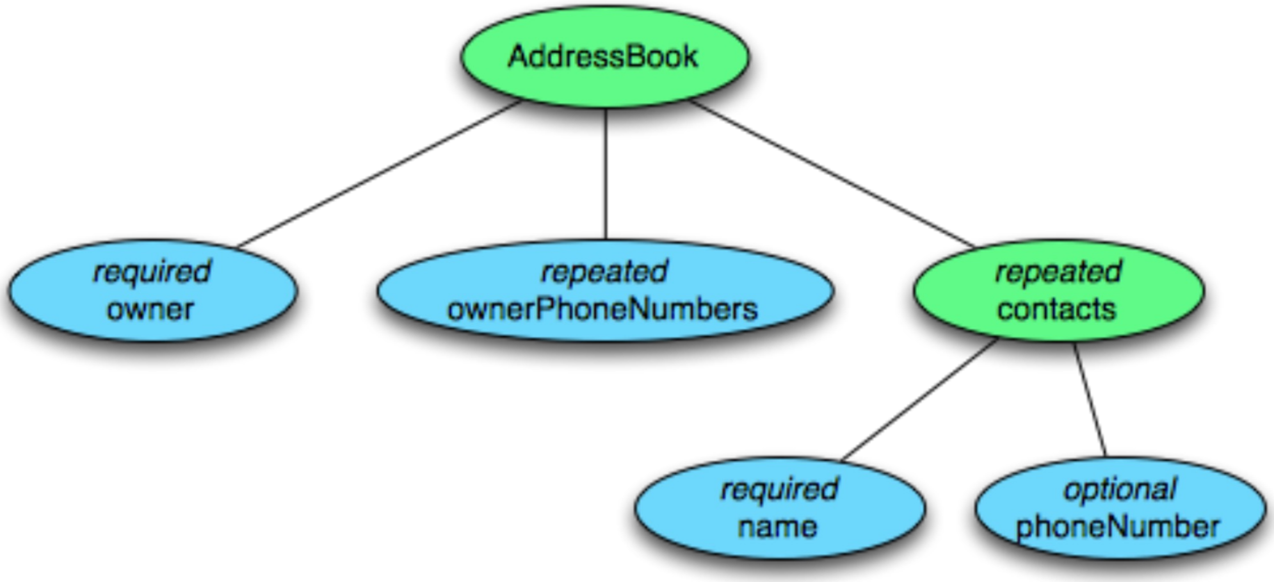


Image from:https://blog.twitter.com/engineering/en_us/a/2013/dremel-made-simple-with-parquet.html

JSON

- Well-known and widely supported
- No explicit schema
- Row-based
- Not splitable
- Complex data like structs and arrays

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

Types of metadata

- Technical
  - Types, representation, …
- Operational
  - Location (indexing), cardinality, …
- Business
  - What it means, quality, ...

Schema

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

.java

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

Central repository

data

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

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

data

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

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

data

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

data-Braga

1	aa	Braga
...	...	...

data-Porto

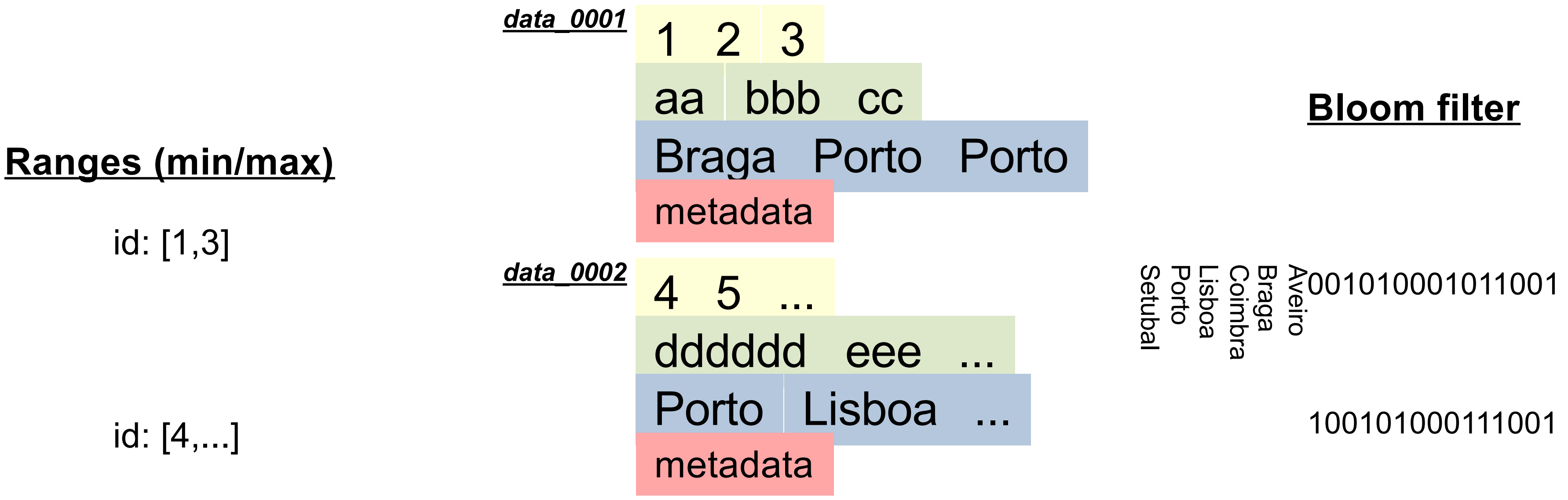
2	bbb	Porto
3	cc	Porto
4	dddddd	Porto
...	...	...

data-Lisboa

5	eee	Lisboa
...	...	...

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

Counts

#id: 3

#id: ...

data_0001

1 2 3

aa bbb cc

Braga Porto Porto

metadata

data_0002

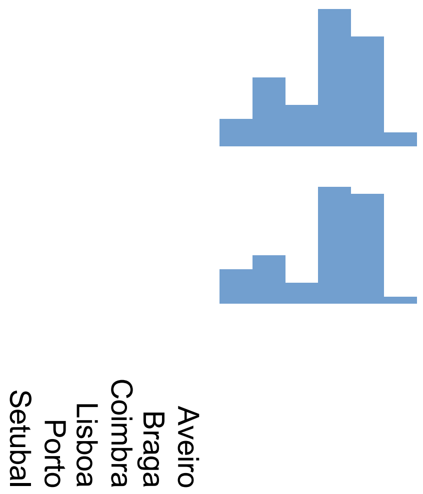
4 5 ...

dddddd eee ...

Porto Lisboa ...

metadata

Histograms



Compression tradeoffs

- Compression ratio vs splittable
- Cold data vs hot data
- Codecs
  - SNAPPY
  - GZIP
  - LZO

Big Data File Formats

- 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
- Highly efficient data compression techniques
- Support for schema evolution
- Faster analytics workloads
  - Less I/O usage
- Splittable file formats
  - Spread between more than one worker node

Avro file

- JSON for data types and protocols
- Row-based
- Compacts binary format but not efficient data compression
- Language-neutral data serialization system
- Schemas: Primitive, Records, Enums, Arrays, Maps …
  - Supports evolution of schemas
- Efficient for use with write-intensive, big data operations.

ORC File

- Compressed columnar storage from Apache Hive
- Block-mode compression
- Data type support
- Ordered data store (within one stripe)
- Indices with column-level aggregated values (min, max, sum, …)

Apache Parquet

- Compressed columnar storage from Cloudera/Twitter
- Highly integrated with Apache Spark
- Supports (page) compression and splitting
- Supports nested columns (Dremel encoding)
- Supports complex nested data structures in a flat columnar format
- Supports minimal number of types
- Uses data skipping to locate specific column values

Apache Parquet File Format

- Row group: A logical horizontal partitioning of the data into rows
- Column chunk: A chunk of the data for a particular column.
- Page: Column chunks are divided up into pages written back to back.

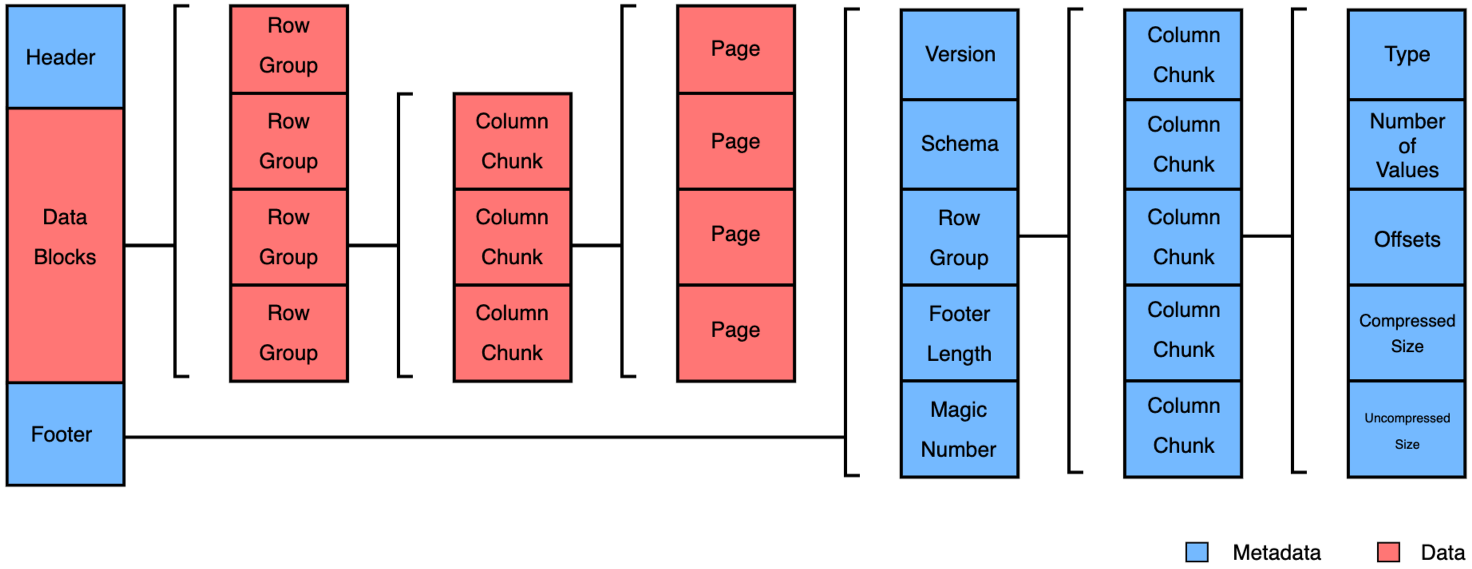


Image from https://dkharazi.github.io/blog/parquet

Apache Parquet File

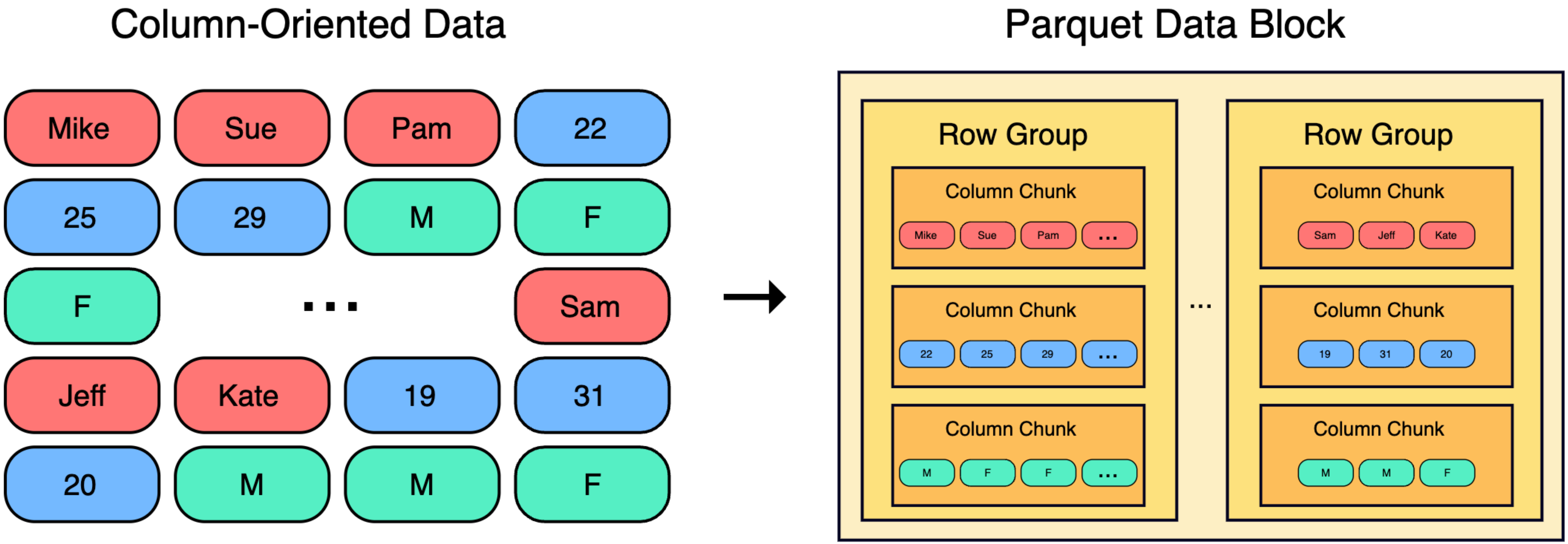


Image from https://dkharazi.github.io/blog/parquet

Apache Arrow

- Language-independent columnar memory format made for flat and hierarchical data
- Efficient analytic operations on modern hardware, CPUs and GPUs
- Primarily in-memory compressed columnar storage for vectorized processing
- Complementary to Parquet
  - Allows easier and more efficient movement of data from RAM to disk

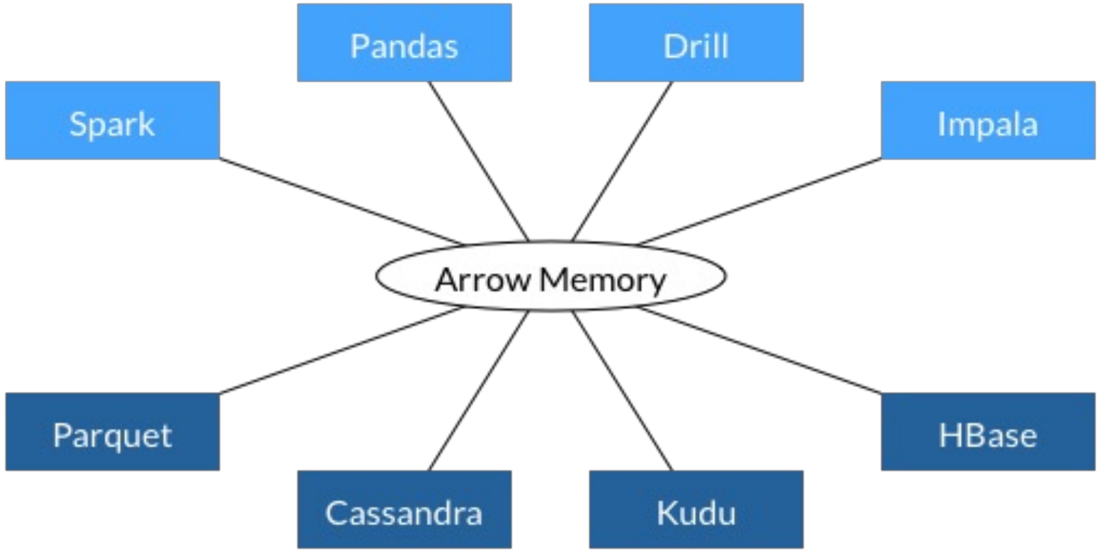


Image from https://arrow.apache.org/overview/

Others formats

- Apache CarbonData
  - Compressed columnar storage with indexes from Huawei
- Apache IceBerg
  - Flexible data format that supports schema evolution from Netflix
- Delta Lake
  - Enables building lakehouses, with ACID transactions, time travel, …
- Not everything tabular
  - Array
    - HDF5
      - Multi-dimensional arrays for scientific workloads
    - Zarr
      - Zarr is a format for the storage of chunked, compressed, N-dimensional arrays
  - Graph
    - RDF
    - JSON-LD

More information

- https://www.researchgate.net/publication/361334530_The_Big_Data_Textbook_-_teaching_large-scale_databases_in_universities
- https://github.com/apache/parquet-format/blob/master/BloomFilter.md
- https://github.com/apache/parquet-format/blob/master/Compression.md
- Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. 2010. Dremel: interactive analysis of web-scale datasets. Proc. VLDB Endow.