CM2606 Data Engineering

Big Data Storage 01

Week 03 | Piumi Nanayakkara

ROBERT GORDON UNIVERSITY ABERDEEN
TEF Gold
INFORMATICS INSTITUTE OF TECHNOLOGY

Learning Outcomes

- Covers LO1 and L02 for Module
- On completion of this lecture, students are expected to be able to:
  - Understand data lake and distributed file systems concepts
  - Understand different row big data storage options and select the appropriate one for a given scenario

Content

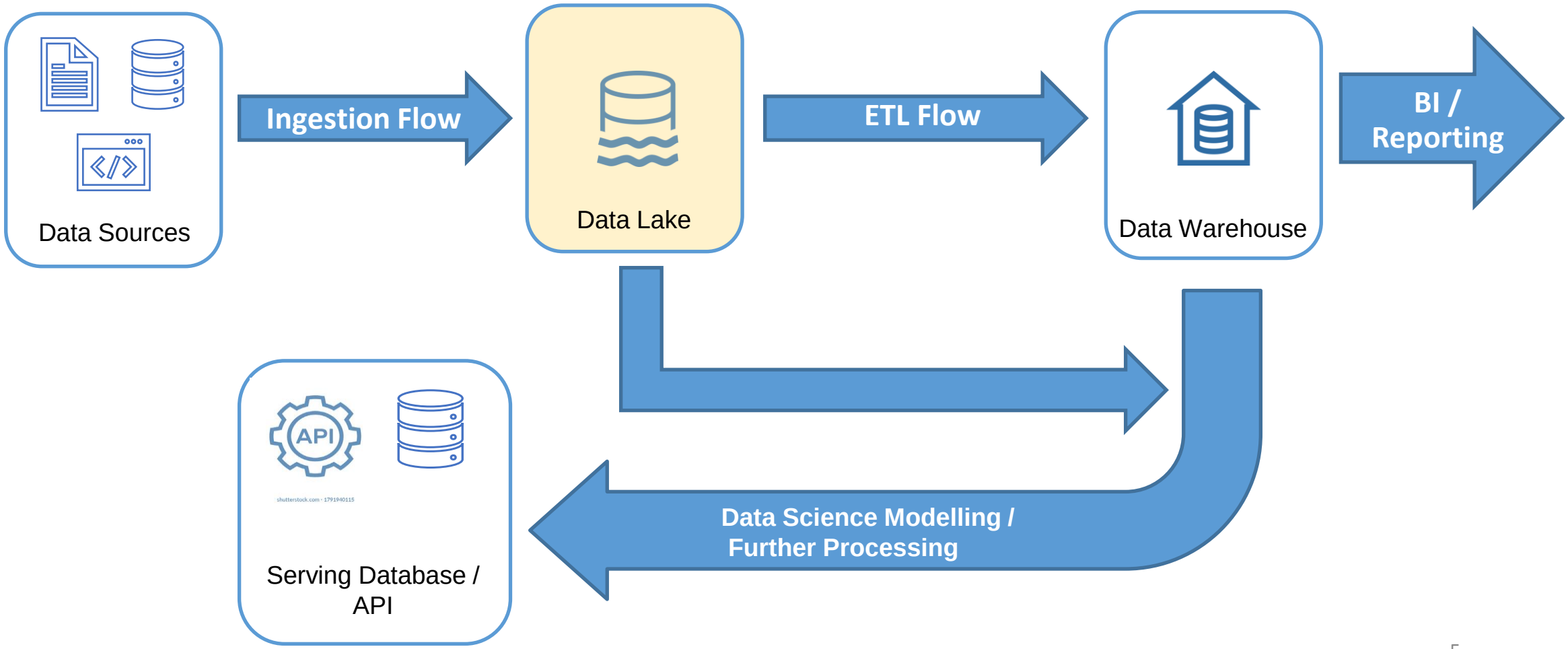
- Data Lake
  - Distributed File Systems
  - HDFS
- Big Data File Formats
  - Choosing a File Format

Data Lake

- Pentaho CTO James Dixon coined the term Data Lake.

" The data lake resembles the lake where the water ***comes in from various sources and stay in the native form****, whereas package bottle of water resembles a data mart which undergoes several filtrations and purification process similarly the data is processed for a data mart"*

Data Pipeline: Common Usage



Data Lake

- Philosophy: “Load first think later”
  - Design should be driven by what is available instead of what is required.
- Support
  - All data formats – structured, semi/unstructured
  - Batch, Streaming and One-Off loads
- Need to manage meta data properly to avoid this is becoming a dumping ground.
- Generally, a distributed file system is used either on-prem or cloud

Need for a Data Lake

- To reduce contention on source systems
- Deal with the ingestion of source systems on different schedules
- Join data together from different source systems
- Rerun failed data warehouse loads from a staging area

Distributed File Systems

- A file system that is
  - distributed on multiple file servers or multiple locations
  - allowing programmers to access files from any network or computer.

- There are two components:
  - Location Transparency – achieved through the namespace component.
  - Redundancy – achieved through a file replication component.

Distributed File Systems: Features

- Transparency :
  - Structure transparency – There is no need for the client to know about the number or locations of file servers and the storage devices.
  - Access transparency – Both local and remote files should be accessible in the same manner.
  - Naming transparency – There should not be any hint in the name of the file to the location of the file.
  - Replication transparency – If a file is copied on multiple nodes, both the copies of the file and their locations should be hidden from one node to another.

Distributed File Systems: Features

- **User mobility :** Automatically bring the user's home directory to the logged in node.
- **Performance :** Computed based on the average amount of time needed to convince the client requests. Preferably be similar to that of a centralized file system.
- **High availability :** Should be able to continue in case of any partial failures

Distributed File Systems: Features

- **Scalability :** Must be designed to scale rapidly and service should not be substantially disrupted as the number of nodes and users grows.
- **High reliability :** A file system should create backup copies of key files that can be used if the originals are lost.
- **Data integrity :** Concurrent access requests from many users who are competing for access to the same file must be correctly synchronized using a concurrency control method.

Distributed File Systems: HDFS

- A highly fault-tolerant distributed file system that is designed to be deployed on low-cost hardware
- Suitable for applications that have large data sets.
- HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories.
- Major component of Apache Hadoop eco system on top of which HBase database and Hive data warehouse built.

HDFS Architecture

- There's a Name Node and multiple data nodes.
- The Name Node contains metadata on "which data nodes contain which blocks".
- Data nodes are organized into Racks (collection of machines)
- Each data block is replicated in 3 data nodes (default replication factor), across 2 racks.

HDFS Read/Write Requests

- All Read/Write requests are received by Name node, which then informs client from/to which data node data could be read/write.
- When writing multiple blocks, the 1st replica of each block is written parallelly, other replicas are written sequentially.
- After a write request is completed, the client informs the Name Node so that the metadata is updated.

HDFS Architecture

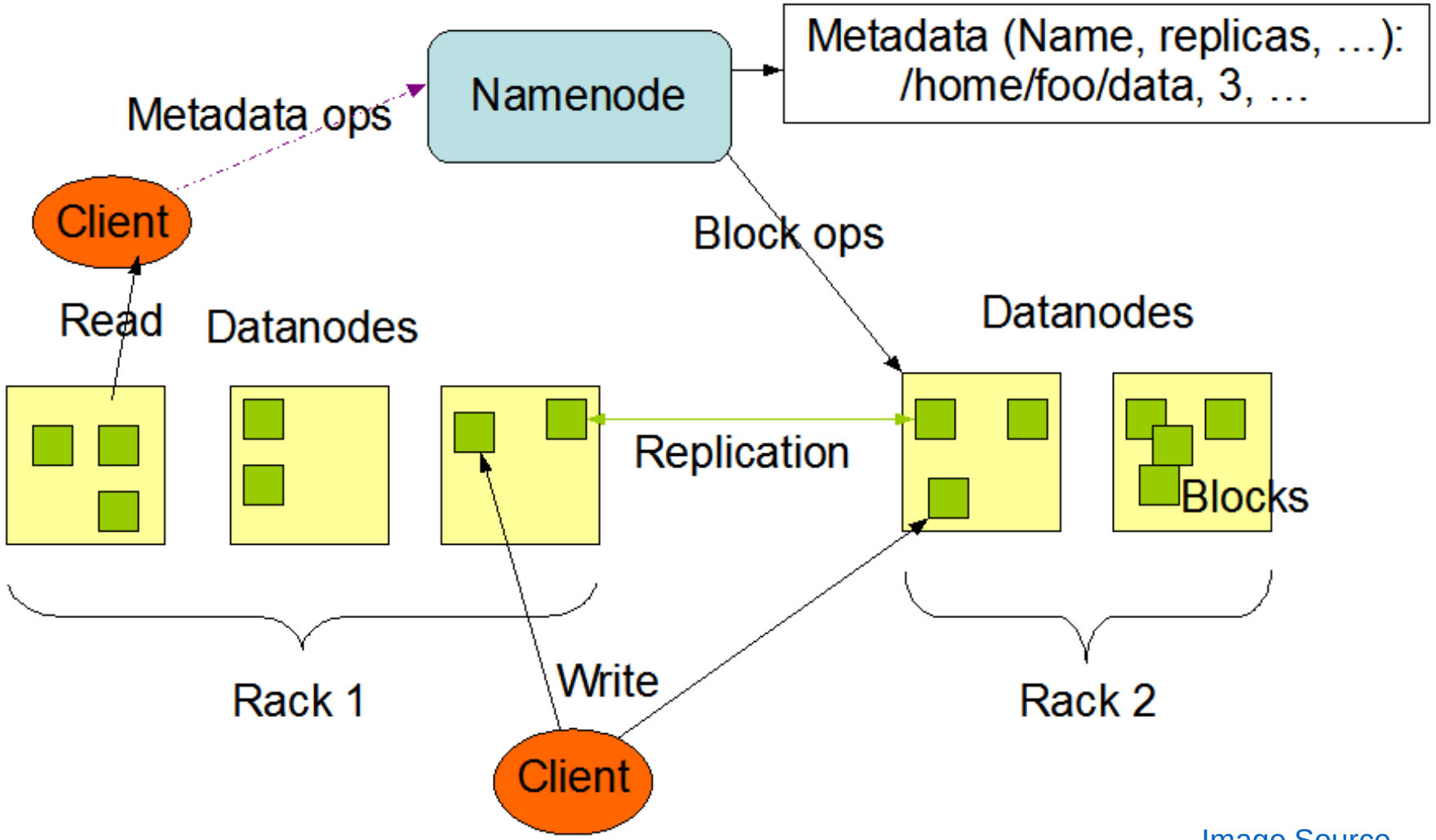


Image Source

HDFS: Small File Problem

- Occurs when the average file is very low than block size resulting in millions of small files.
- Reading through small files involve lots of seeks and lots of hopping between data node to data node increasing processing time.
- More time will be wasted starting and killing JVMs,
- Also increase the size of metadata in the name node as well.
- Solution: When writing data have appropriate no. of reducers/tasks

Big Data File Formats

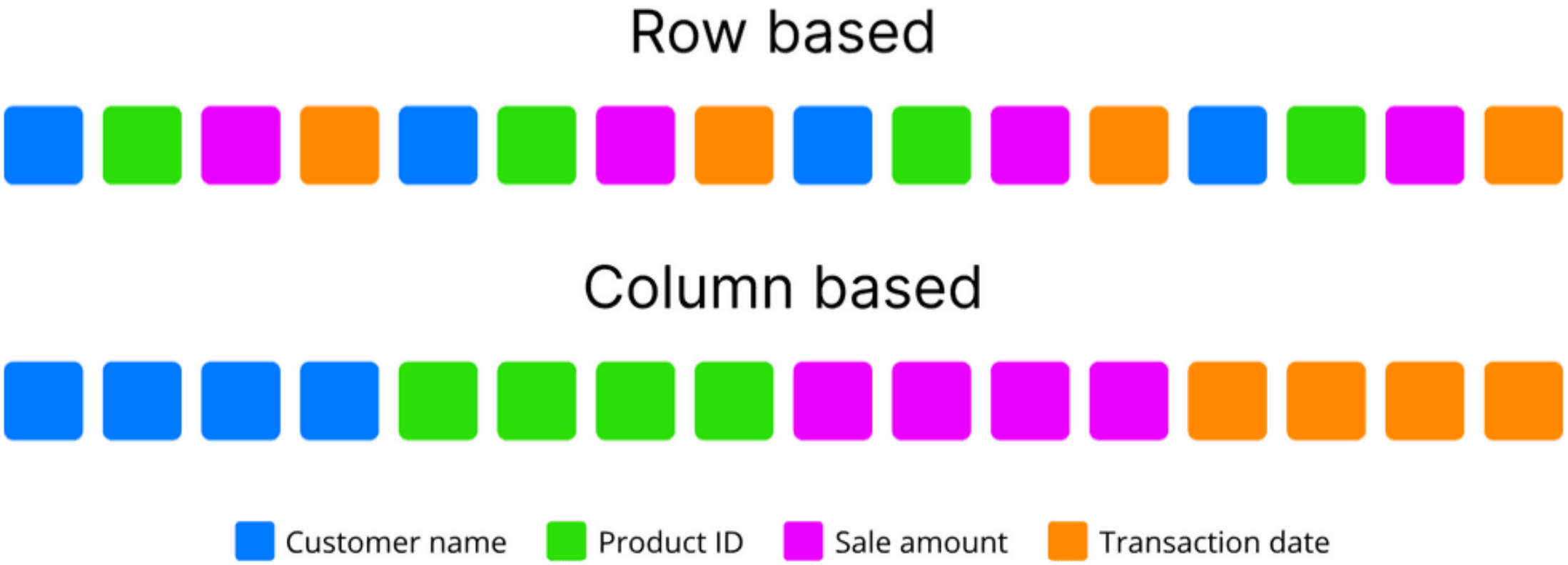


Image Source

Big Data File Formats: CSV

- When initiating the data pipelines, many data sources would produce data in csv format either manually maintained or database exports.
- Way to represent tabular data in plain text. Files may use separators other than commas, such as tabs or spaces.
- Row oriented. May or may not contain a header row - containing column names for the data
- Initially doesn't contain hierarchical or relational data, these needs to be established by using multiple CSV files

Big Data File Formats: CSV

- Pros:
  - Human readable and can be edited manually.
  - Easy to parse / read for processing
  - Compact when compared to XML

- Cons:
  - No Column types. Text and Numbers are not differentiated
  - Work mainly with flat data. Nested types are should be handled separately. E.g., a product list
  - No universal standard → leads to issues when importing
    - E.g., no difference between NULL and quotes

Big Data File Formats: JSON

- Semi structured format where data is presented in key value pairs.
- User readable and much smaller than xml.
- Widely encountered when extracting data using in web services, web scraping etc.

Big Data File Formats: JSON

- Pros:
  - Support hierarchical structures and complex relationships.
  - Can be directly stored in document databases.
  - Language support exists to serialize/de-serialize

- Cons:
  - Poor memory consumption due to field names being repeated
  - Lacks indexing
  - Poor support for special characters

Big Data File Formats: Parquet

- Column Oriented file format
- Binary files containing metadata about content such as column names, data types and even some basic statistical characteristics such as min max values
- Meta data is stored at the end of the file which are used by processing engines such as Spark for faster reads.
- Optimized for Write Once Read Many (WORM) paradigm.
  - Write slowly reads quickly
- Ideal when reading subset of columns or queries with filters, due to projection pushdown & predicate pushdown.

Predicate Pushdown/ Filter Pushdown

- Any filtering criterions in a query is evaluated when reading the records from the file, instead of loading all data into memory and then filtering.
  - This reduce the memory consumption and query time considerably.
- This is made easier by the meta data stored such as min max values
- This concept is followed by most DBMS, as well as big data storage formats such as Parquet and ORC.

Big Data File Formats: Parquet

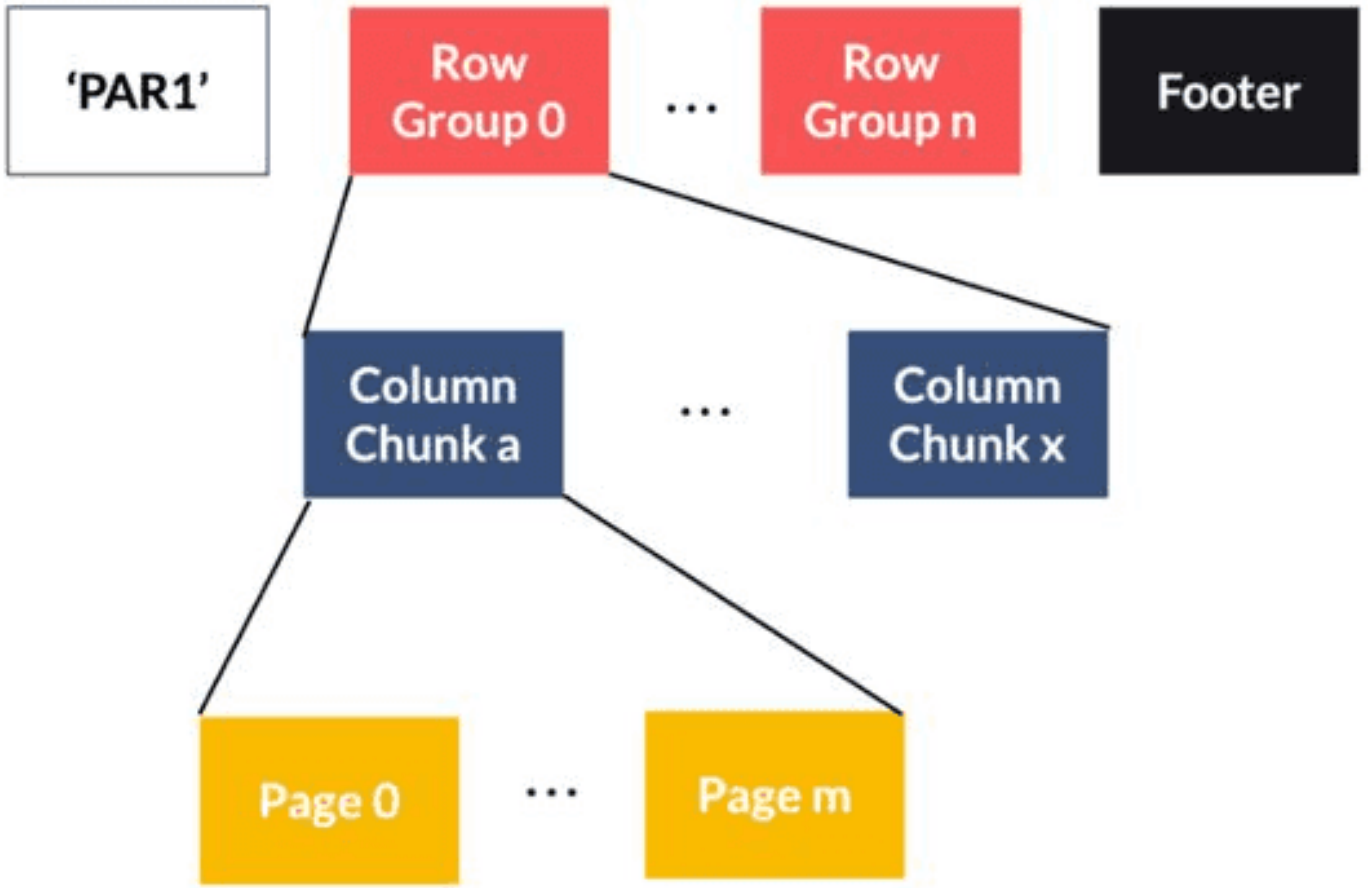


Image Source

Big Data File Formats: Parquet

- Pros:
  - Can be highly compressed
  - Built-in support in Spark
  - Meta data being stored – data is self-describing

- Cons:
  - Less support in tools other than spark
  - Immutable. Need to overwrite

Big Data File Formats: ORC

- ORC (Optimized Row Columnar) is a column-oriented format.
- Contain groups of row data called stripes, along with auxiliary information in a file footer.
  - Default stripe size is 250 MB. Large stripe sizes enable large efficient reads from HDFS
- At the end of the file, a postscript holds compression parameters and the size of the compressed footer.

Big Data File Formats: ORC

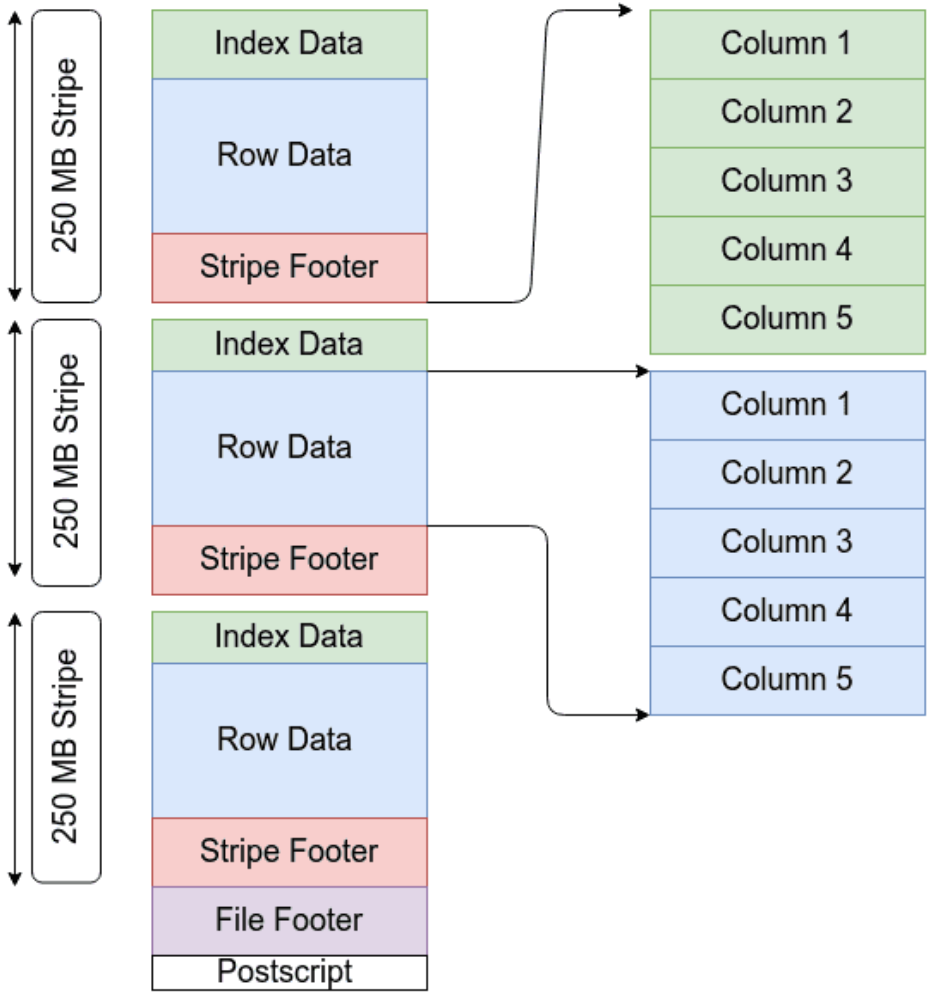


Image Source

Big Data File Formats: ORC

- Pros:
  - Highly compressible, specially with flattened data
  - Can serialize data within data in 1 minute. Thus, suitable for streaming data.

- Cons:
  - Less support for nested data

Big Data File Formats: Avro

- Row-based format where the schema is stored in JSON format, while the data is stored in binary format, which minimizes file size
- Also described as a data serialization system like Java Serialization.
- Avro has reliable support for schema evolution by managing added, missing, and changed fields.
  - The schema used to read Avro files does not necessarily have to be the same as the one used to write the files.

Big Data File Formats: Avro

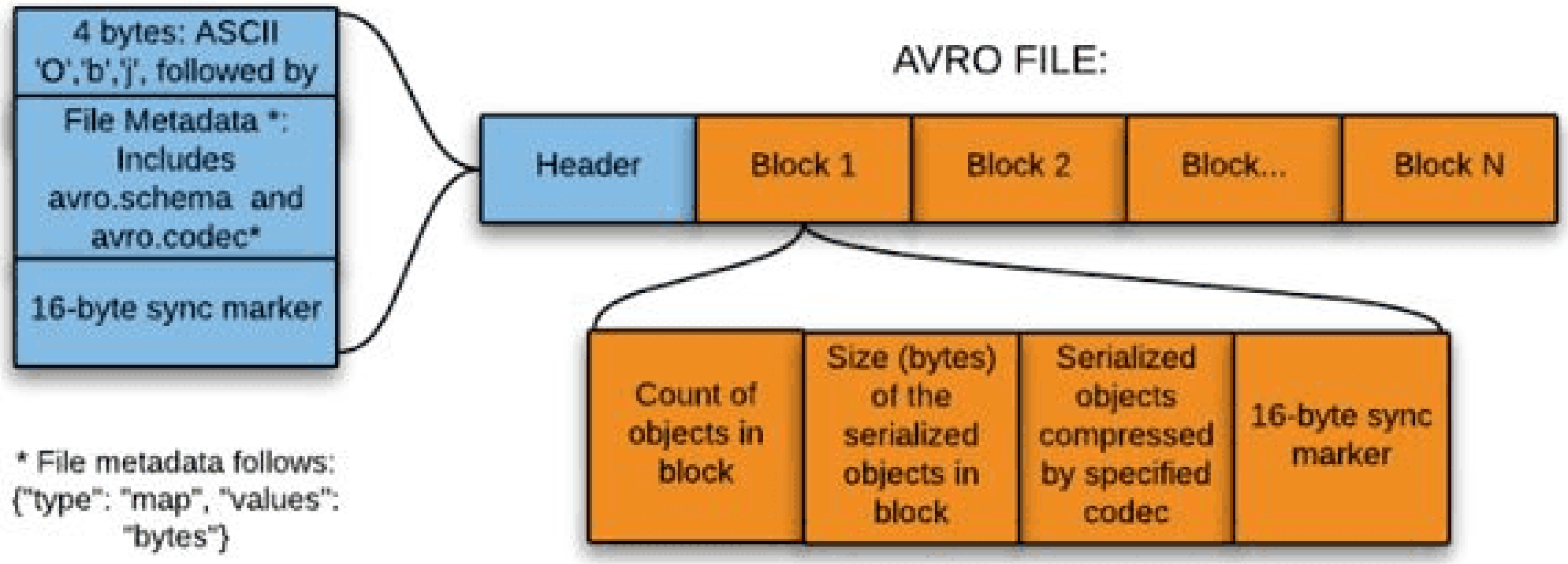


Image Source

Big Data File Formats: Avro

- Pros:
  - Data is self-describing
  - Easy and fast data serialization and deserialization, providing very good ingestion performance
  - Support for schema evolution

- Cons:
  - Data is not human-readable;
  - Not integrated into every programming language.

Choosing File Format

- **Text vs Binary:**
  - Text based file formats:
    - human readable/editable
    - Usually compressed to reduce their storage footprints.
  - Binary file formats:
    - Can not be read / edited by human
    - Provides better performance by optimizing the data serialization.

- **Integration with third-party applications**
  - E.g., when choosing between Parquet or ORC with Hive, it is recommended to use the former on Cloudera platforms and the latter on Hortonworks platforms.

- **Evolution of the schema**
  - Allows to update the schema used to write new data while maintaining backwards compatibility with the schema of your old data: AVRO

Choosing File Format

- **Read Performance:**
  - If reads are significantly higher than writes:
    - When reading all columns (e.g., ingestion) – Avro
    - When reading a subset (e.g., after processing) – Parquet or ORC

- **Write Performance:**
  - If the operation involves lot of writes and less reads (e.g., Data Lake landing):
    - AVRO format works out the best due to serialized row-based storage.

Choosing File Format

- **Support for specific data types**
  - Can complex types be stored: e.g., arrays
  - Impact storage footprint once converted to binary, text take more space than numeric or binary

- **Storage Cost:**
  - Data stored in columnar formats works out the best, when it comes to compression. ORC format offers the best compression characteristics

- **Human Readability:** JSON

Further Reading Material

- K. Shvachko, H. Kuang, S. Radia and R. Chansler, "The Hadoop Distributed File System," 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), 2010

- Borthakur, Dhruba. "HDFS architecture guide." Hadoop apache project 53.1-13 (2008): 2.

- Nexla Whitepaper: An Introduction to Big Data Formats.