DFS and File Formats

Agenda

→ Changes in the exercise hours

→ Discord!

→ Some terms

→ Hadoop’s HDFS

→ Parquet

→ Avro

SDU

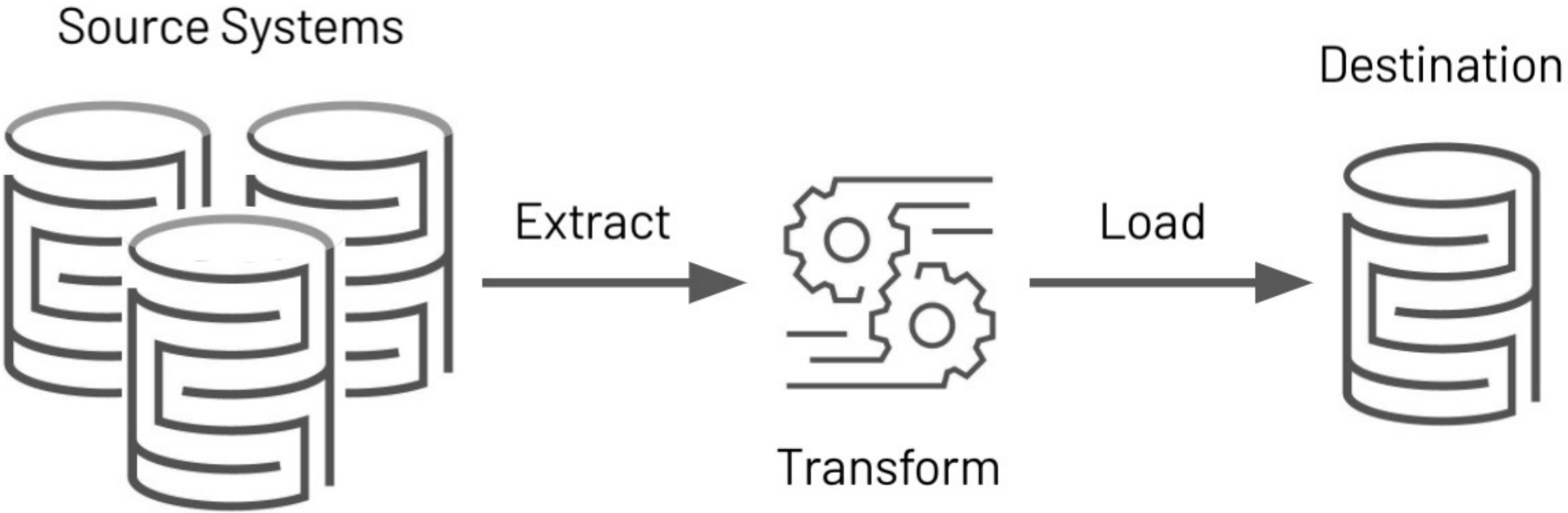
Discord!

- → Our common chatroom!
- → Announcements!
- → Ask the teacher, instructors, alumni, or other students!
- → Join here: https://discord.gg/mHvjRdn28
- → Change your name to your real name to get access!

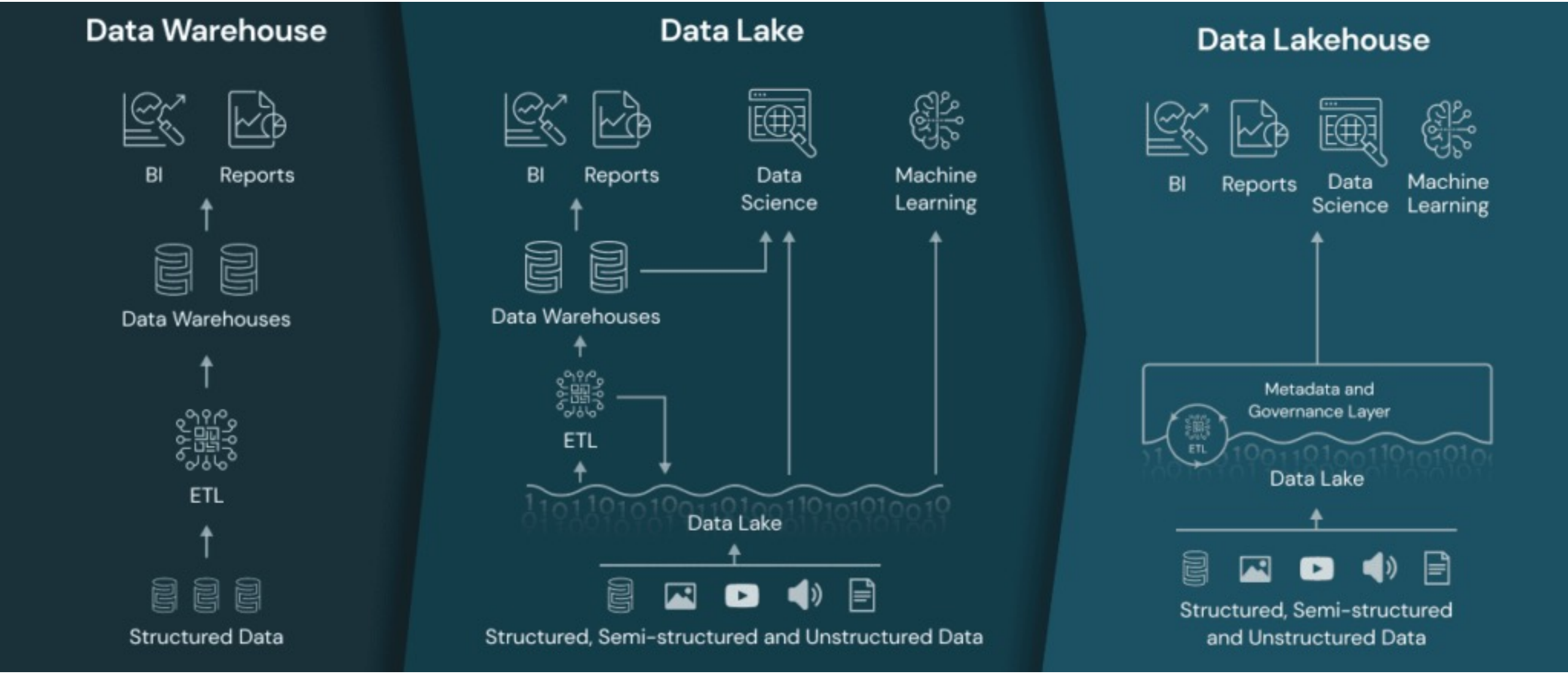
SDU

First some terms

ETL vs ELT vs EtLT



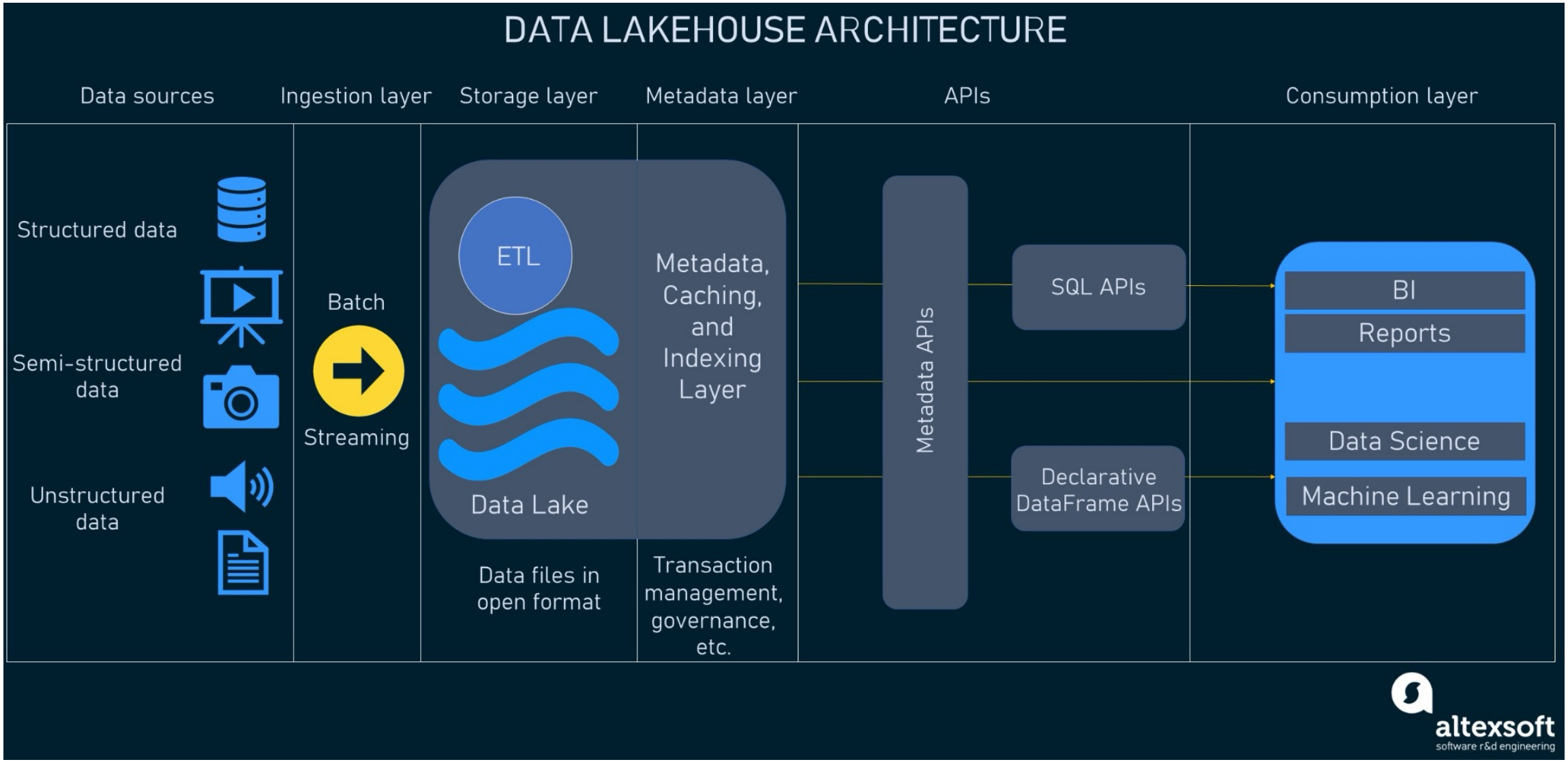
Data Warehouse/Lake/Lakehouse (and Data Mesh)



SDU

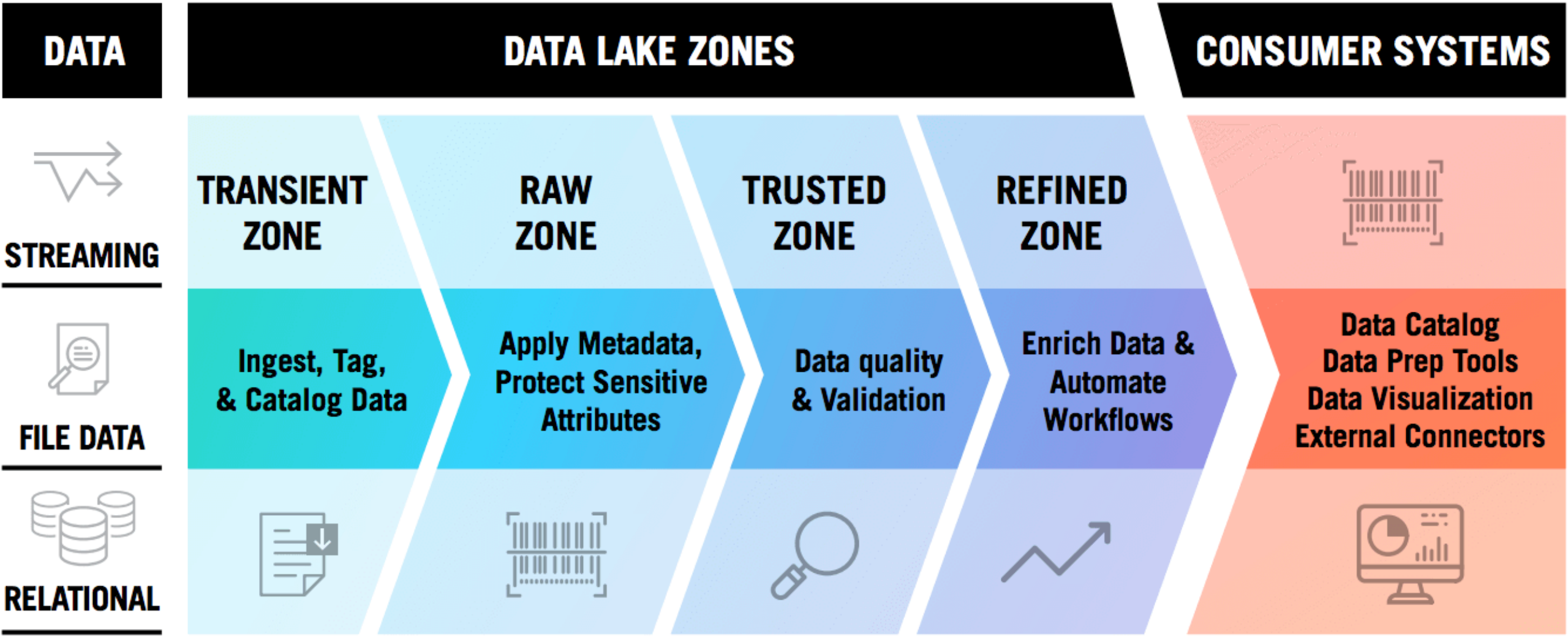
Graphic on loan from: https://www.databricks.com/blog/2020/01/30/what-is-a-data-lakehouse.html

Typical structure and missing use cases



Graphic on loan from: https://www.altexsoft.com/blog/data-lakehouse/

Data Lake Zones



Data Lake Storage Zones

Landing

- Data Source
  - Bad/Quarantine

- Transient folder for data to land
- Data move to Bad/Quarantine if data are corrupted for manual intervention

Raw

- Data Source
  - 2019
  - 2020
    - 202001
      - Online_Sales_20200101.csv
      - Online_Sales_20200102.csv
      - Online_Sales_20200103.csv
    - 202002

- Valid data are moved to Raw folder in native format and ready for Ingestion
- Data are categorised into Data Source/Year/Month
- Depending on requirement, folders can further categorised into Day/Hour

Standardized

- Data Source
  - 2019
  - 2020
    - 202001
      - Online_Sales_20200101.csv
      - Online_Sales_20200102.csv
      - Online_Sales_20200103.csv
    - 202002

- Optional Layer
- Standard Format best suitable for Curated Layer
- Have option to perform some standardise data cleansing (e.g. removing special characters)
- Characters encoding

Curated

- Sales
  - Sales.txt

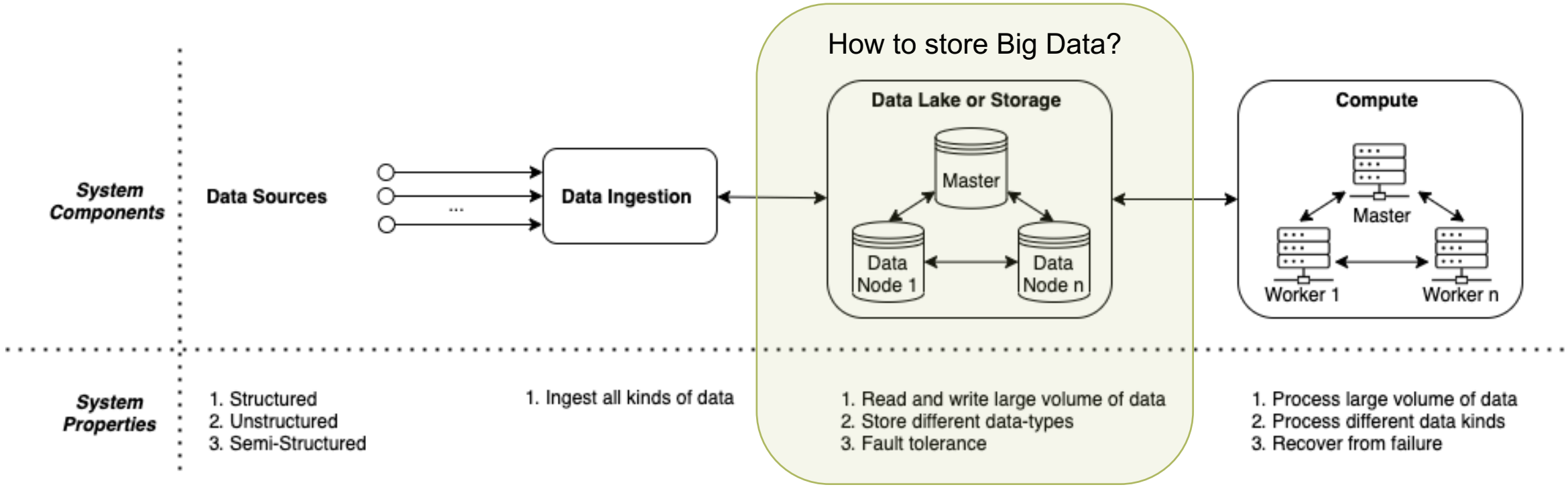
- Data is cleansed and Ready for Consumption
- Categorized into Subject Area/Files
- Option to partition data if requirement exist Subject Area/File/Partition Structure (e.g. Year/Month , Region)

Sandbox

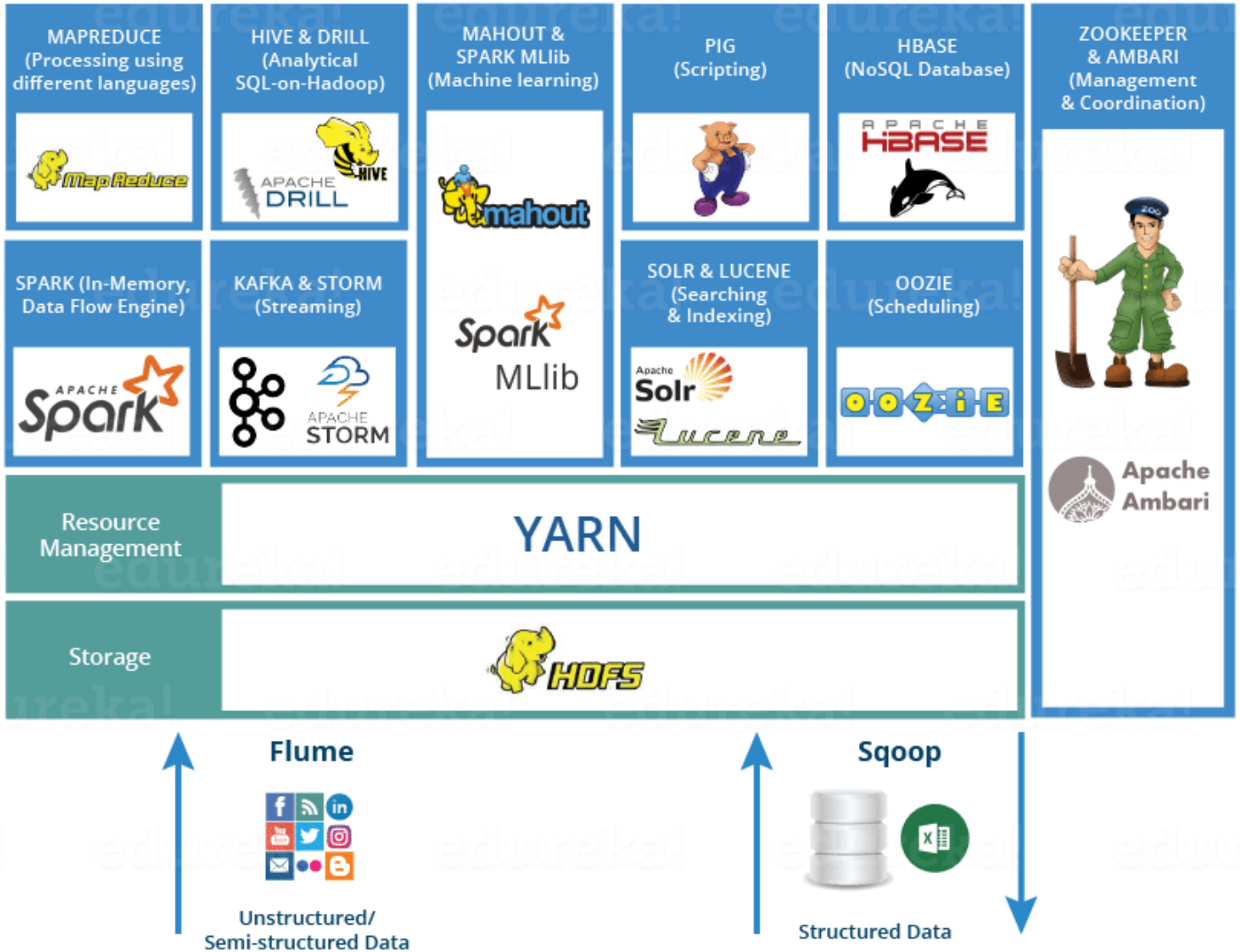
- Customer Classification
- Product Recommendation
- Personalized Marketing

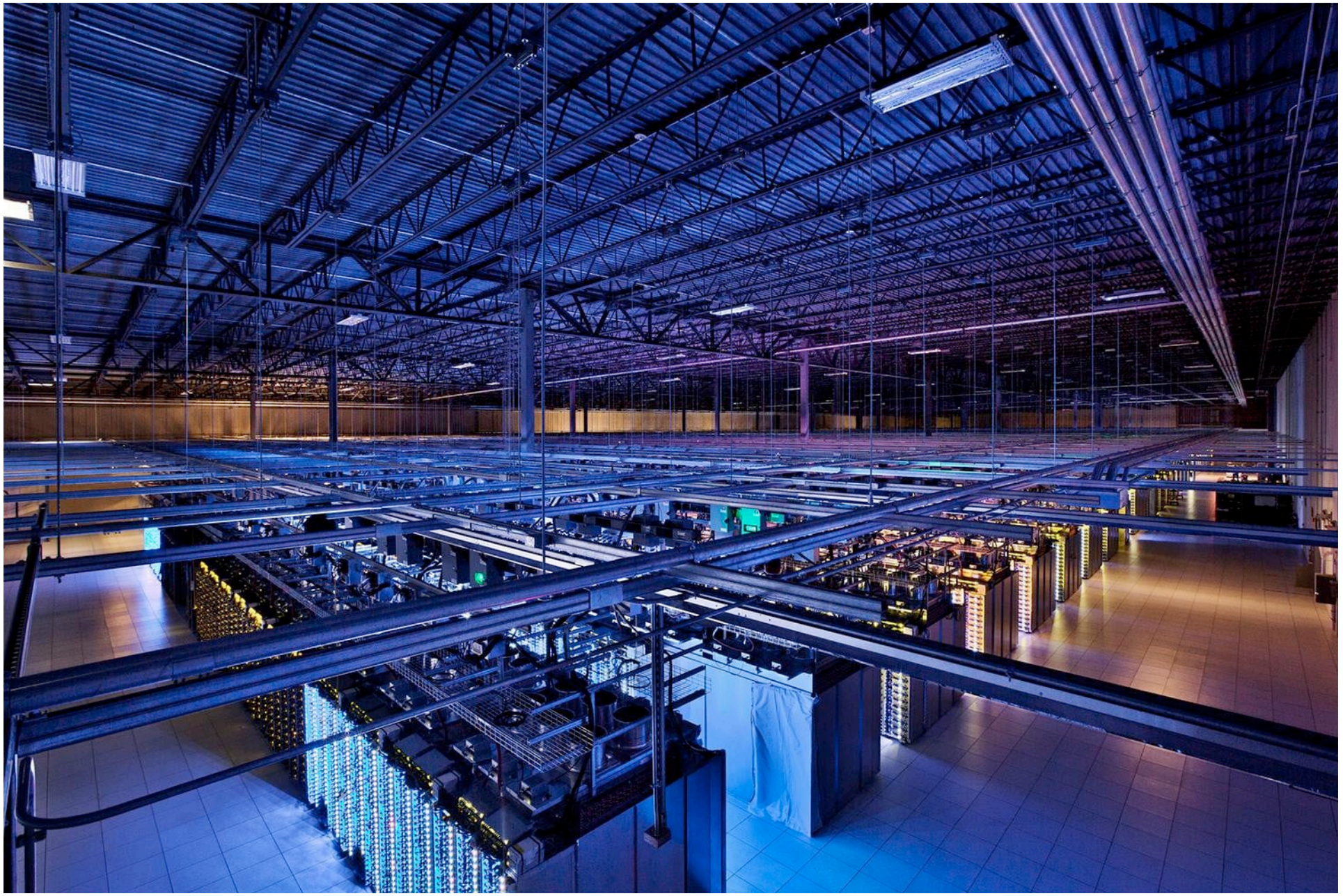
- Mostly targeted for Data Scientist
- Organised per Project
- Data comes from Curated and may also source from Standardized layer

Hadoop


How to store Big Data?
System Components
Data Sources
...
Data Ingestion
Data Lake or Storage
Master
Data Node 1
Data Node n
Compute
Master
Worker 1
Worker n
System Properties
1. Structured
2. Unstructured
3. Semi-Structured
1. Ingest all kinds of data
1. Read and write large volume of data
2. Store different data-types
3. Fault tolerance
1. Process large volume of data
2. Process different data kinds
3. Recover from failure

Center of the Apache Ecosystem





SDU

(Google data center at Council Bluffs, Iowa [W2])

The cluster fails

→ Google cluster: 103 commodity Linux servers

- 1-5% of disks will crash per year,
- frequent network failures

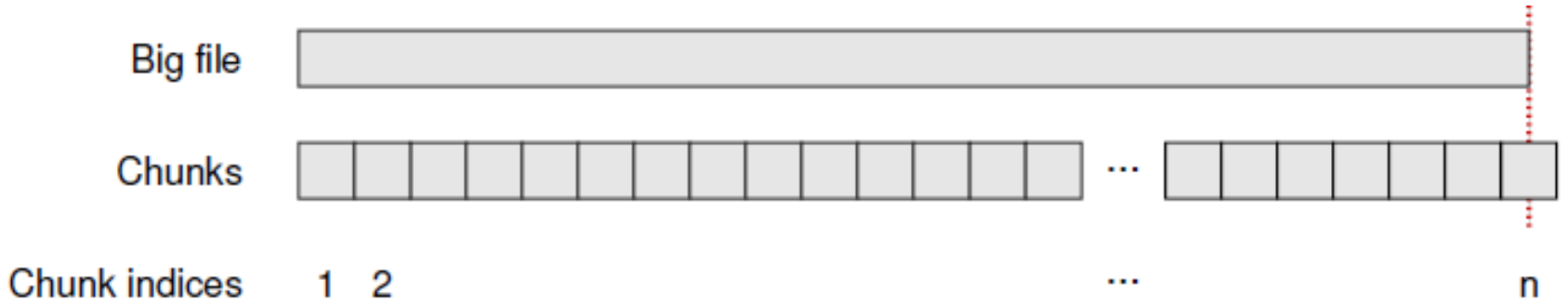
→ Some component of the DFS is always non-functional.

→ A DFS must be highly fault-tolerant!

DFS design

→ Files (> 1 TB) won't fit on a disk, so:

- divide files into chunks (64 or 128 MB)



- a large chunk size is advantageous for storing big files

DFS design

→ Since cluster components fail,

- store all data redundantly
- Typically 3 chunk replicas, on different machines, on different racks
- if one copy is unreachable, read another copy.

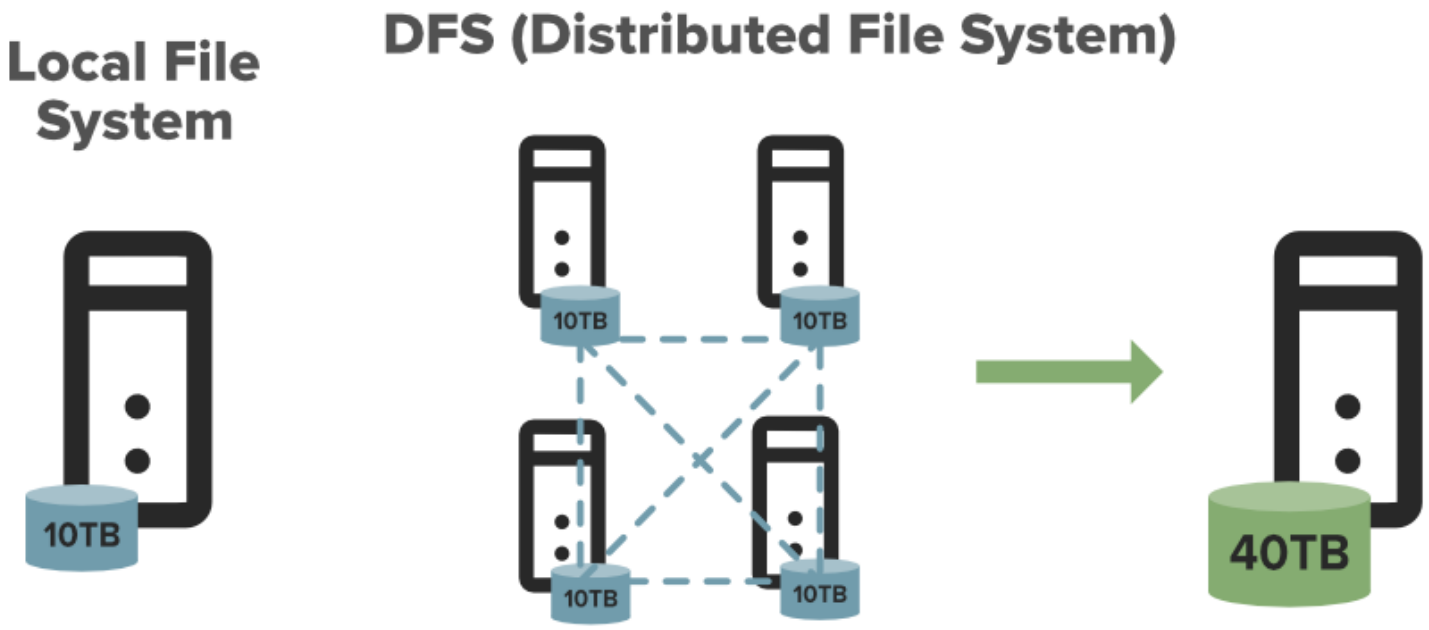
→ _____

→ To implement redundancy, you need:

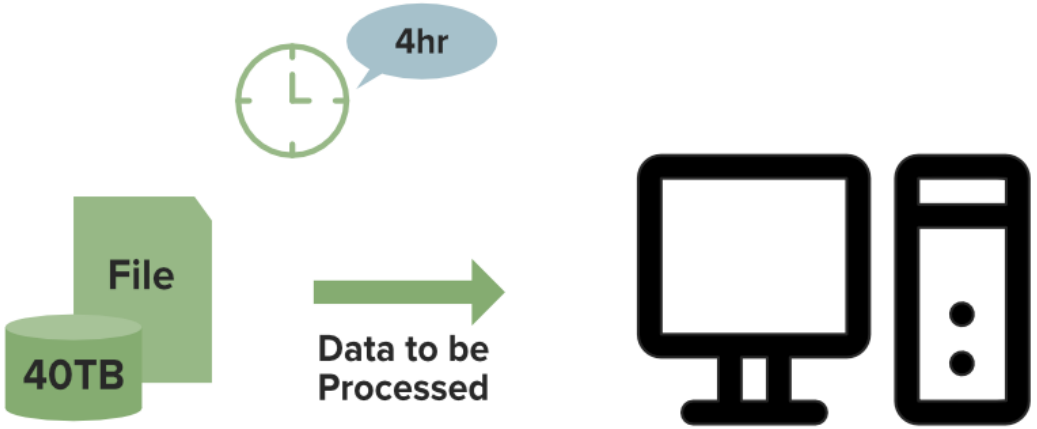
→ metadata for each file (that maps the file into chunks)

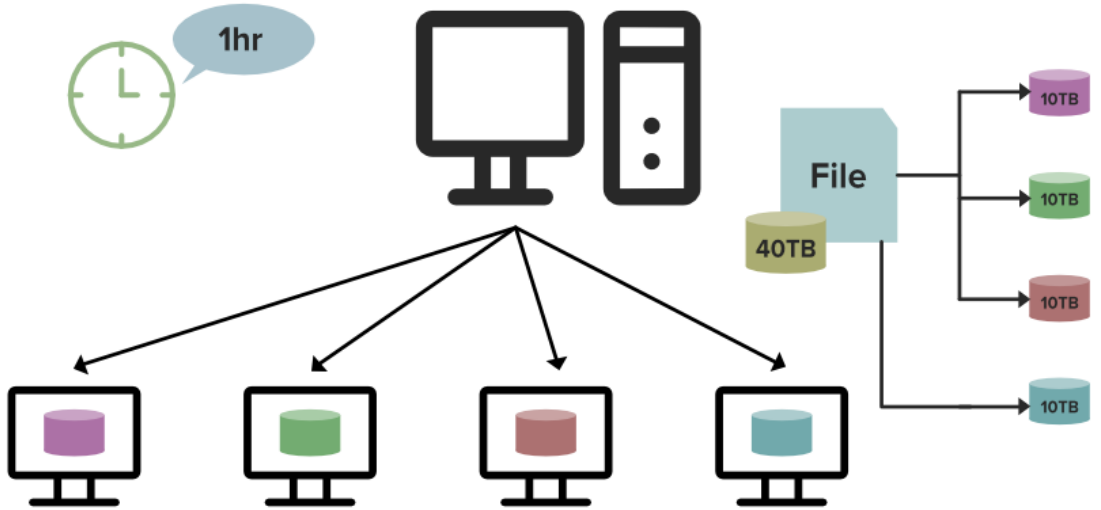
→ a master for the DFS (to maintain metadata).

LFS vs DFS

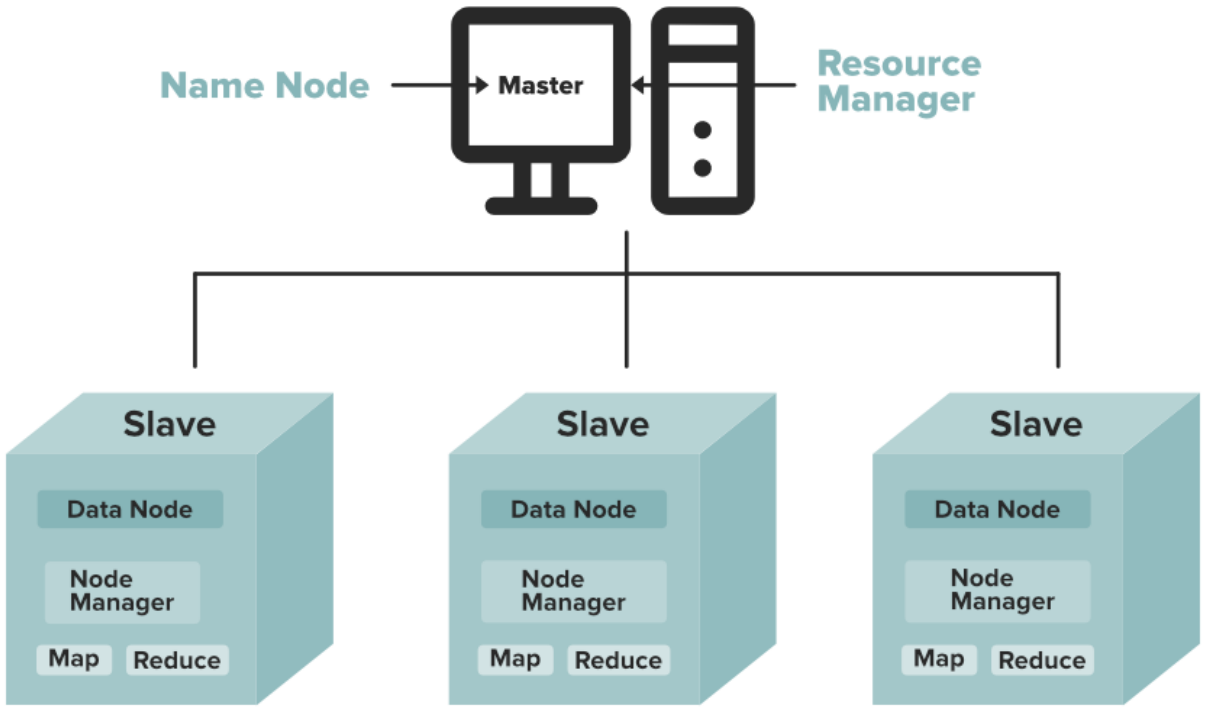

Local File System
10TB
DFS (Distributed File System)
10TB
10TB
10TB
10TB
40TB

DFS – Distribution of data (remove bottleneck)

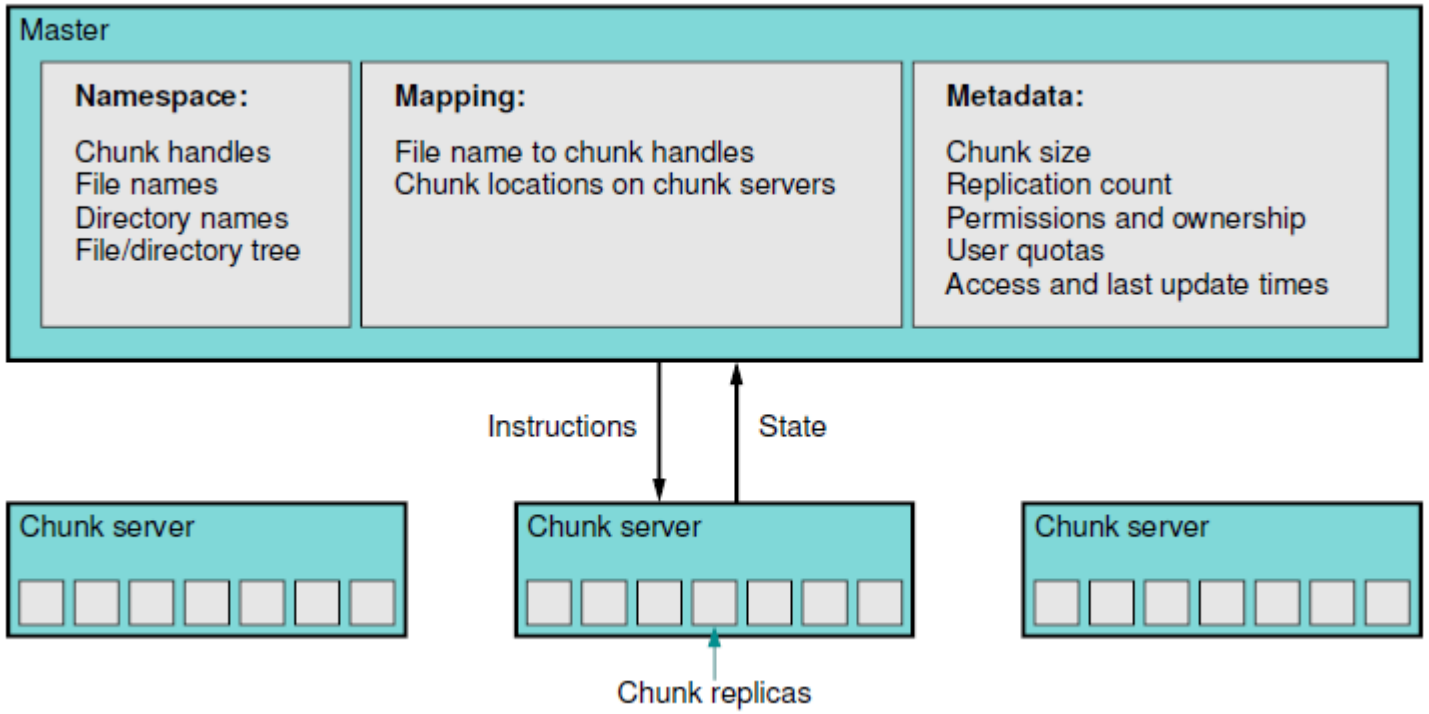




The Hadoop Setup



The Namenode (Master server)

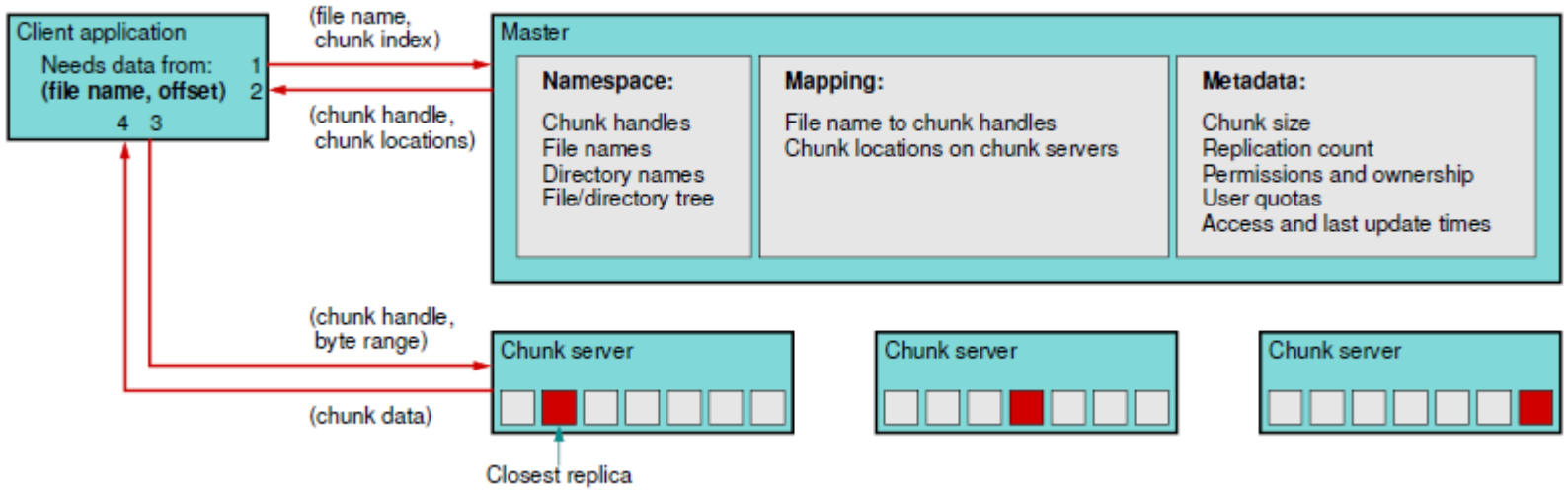


→ Global, single master (bottleneck). ”Shadow” masters.

→ All metadata is kept in memory (fast!). Most is also logged on disk for reliability.

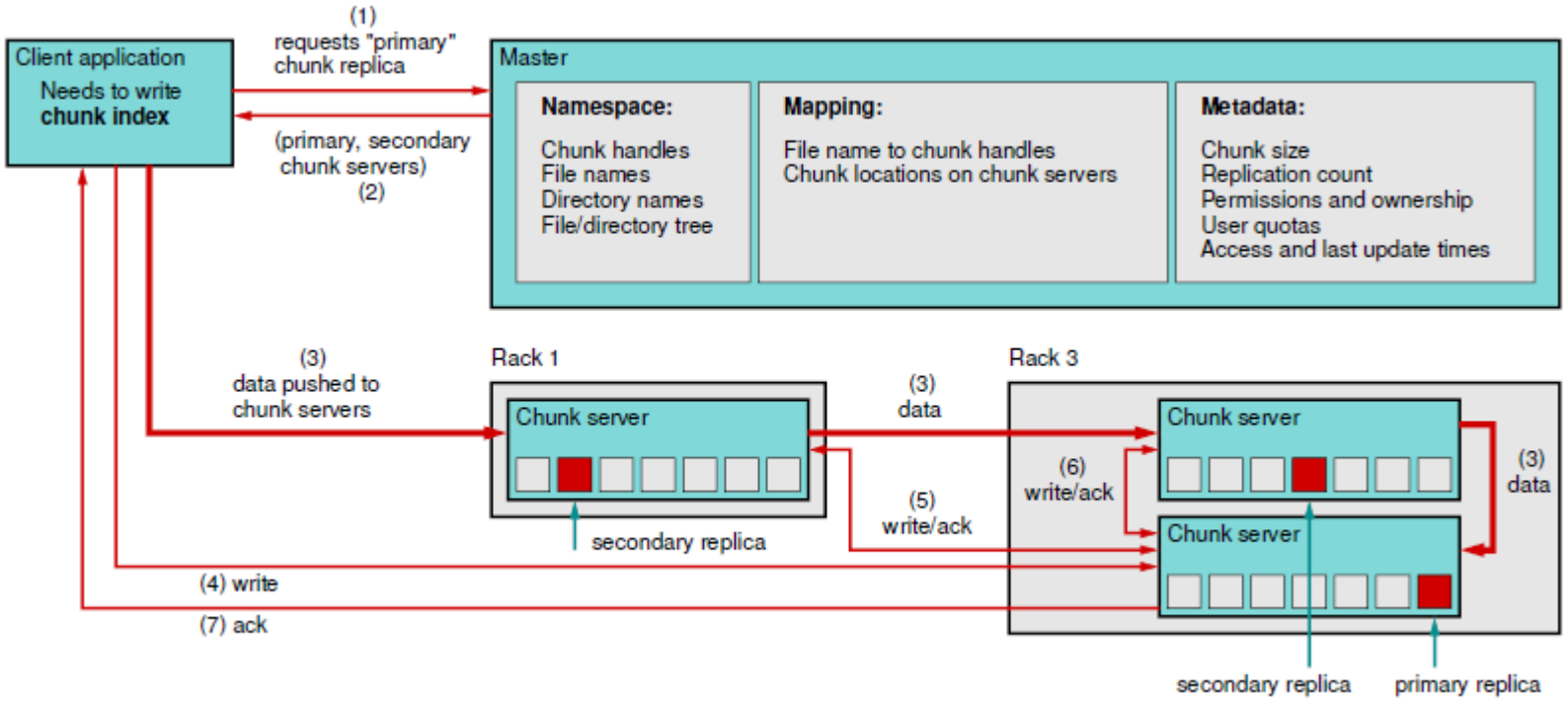
→ Exception: the chunk locations. The master simply polls the servers at startup.

A read operation



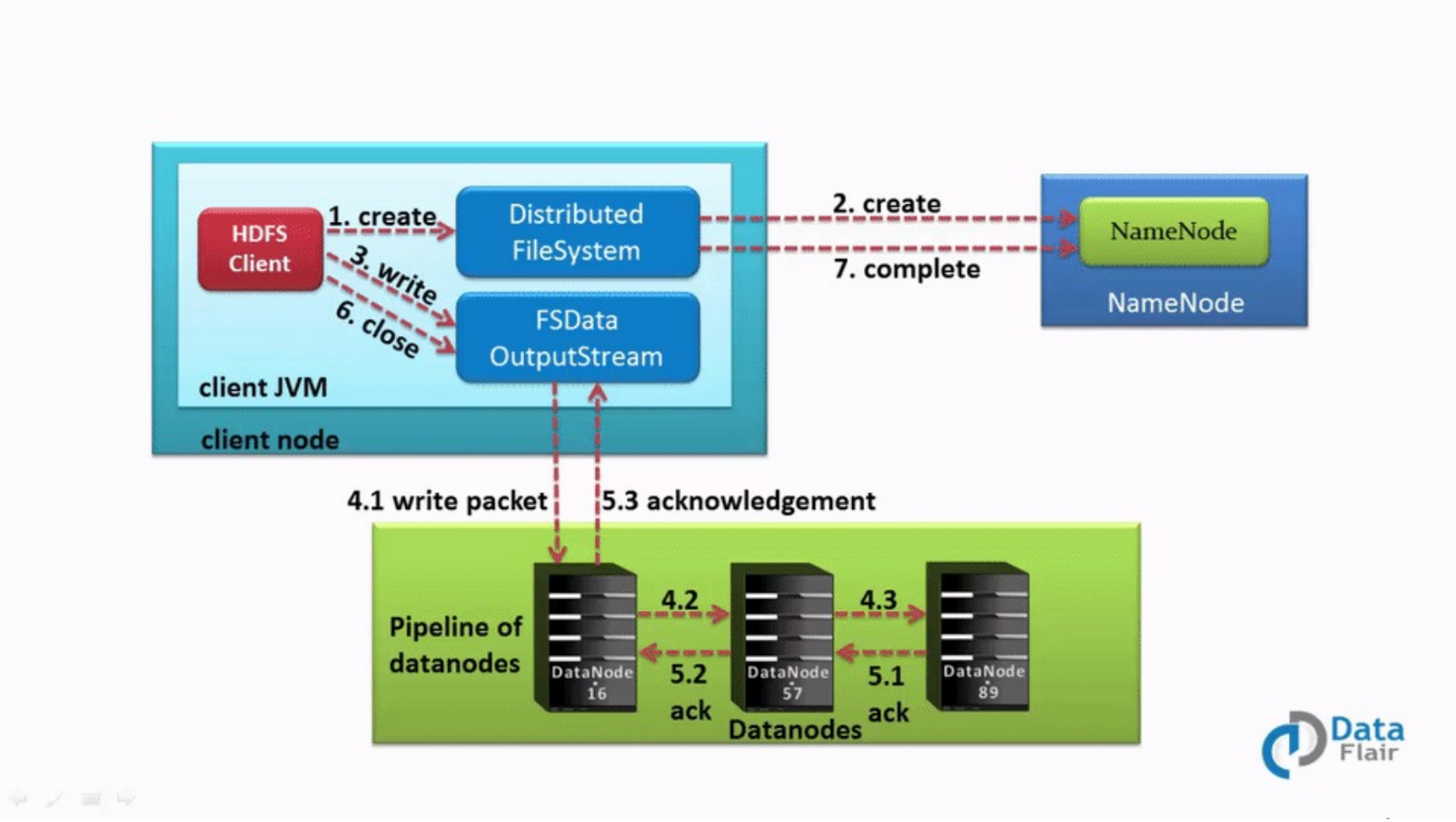
→ Data doesn't pass through the Master.

A write operation



→ Data doesn't pass through the Master.

A write operation



→ Src: https://data-flair.training/blogs/hdfs-data-write-operation/

Reading data on a DFS

→ In practice, reads come in two types:

- large contiguous reads (1 MB+)
- small random reads (1 KB).

→ **Random reads** should be avoided:

→ performance-conscious applications batch and sort their small reads to advance linearly through the file.

Writing data on a DFS

- Once written, the files are mostly read, and mostly sequentially.
- Most files are mutated by appending new data rather than overwriting data.

→ **Random writes** are practically non-existent:

→ they are supported, but are inefficient.

DFS implementations

GFS : ***Google File System, the original DFS***

HDFS : ***Hadoop Distributed File System***
a GFS clone

open-source, distributed by Apache Software Foundation

many others : see [W4]

→ Facebook HDFS (2012): 100+ petabytes of disk space in one of its largest HDFS clusters (hundreds of thousands of servers).

Some Sources for further reading:

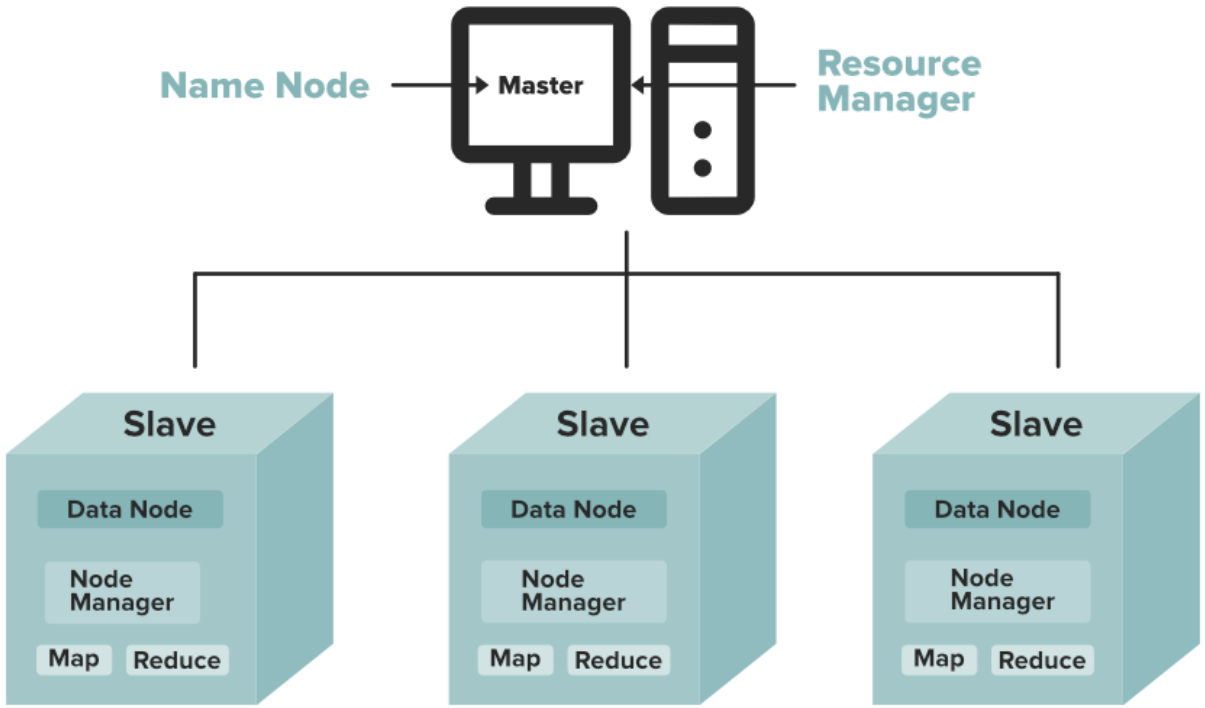
W1. Facebook is collecting your data - 500 terabytes a day (2012)
https://goo.gl/x2DZGz (gigaom.com)

→W2. Where the Internet lives. Take a look inside Google's high-tech data centers
https://www.google.com/about/datacenters/gallery/

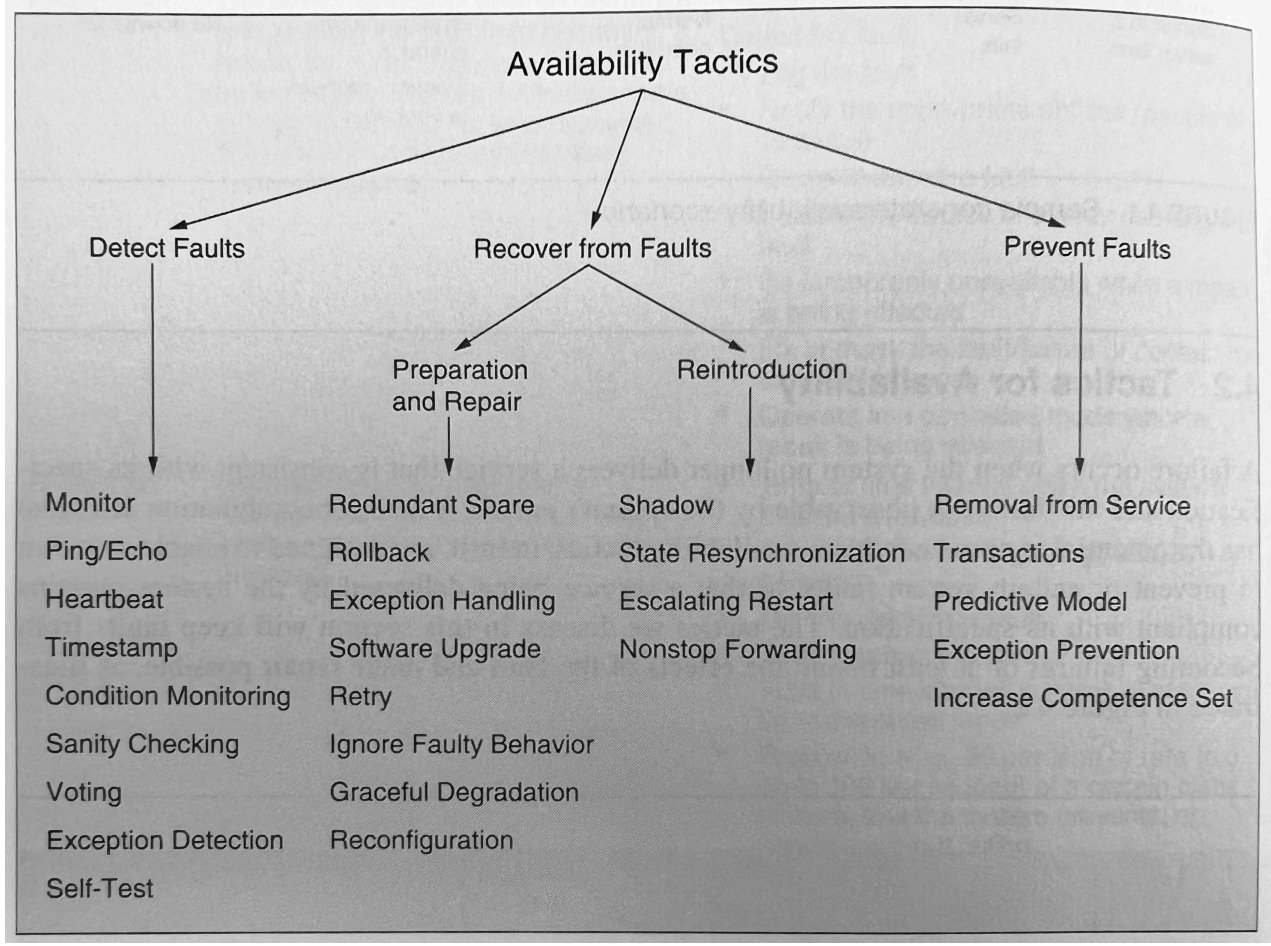
→W3. The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines (2013)
https://goo.gl/zrWA3G

→W4. Distributed parallel fault-tolerant file systems
https://goo.gl/h8q37d (Wikipedia)

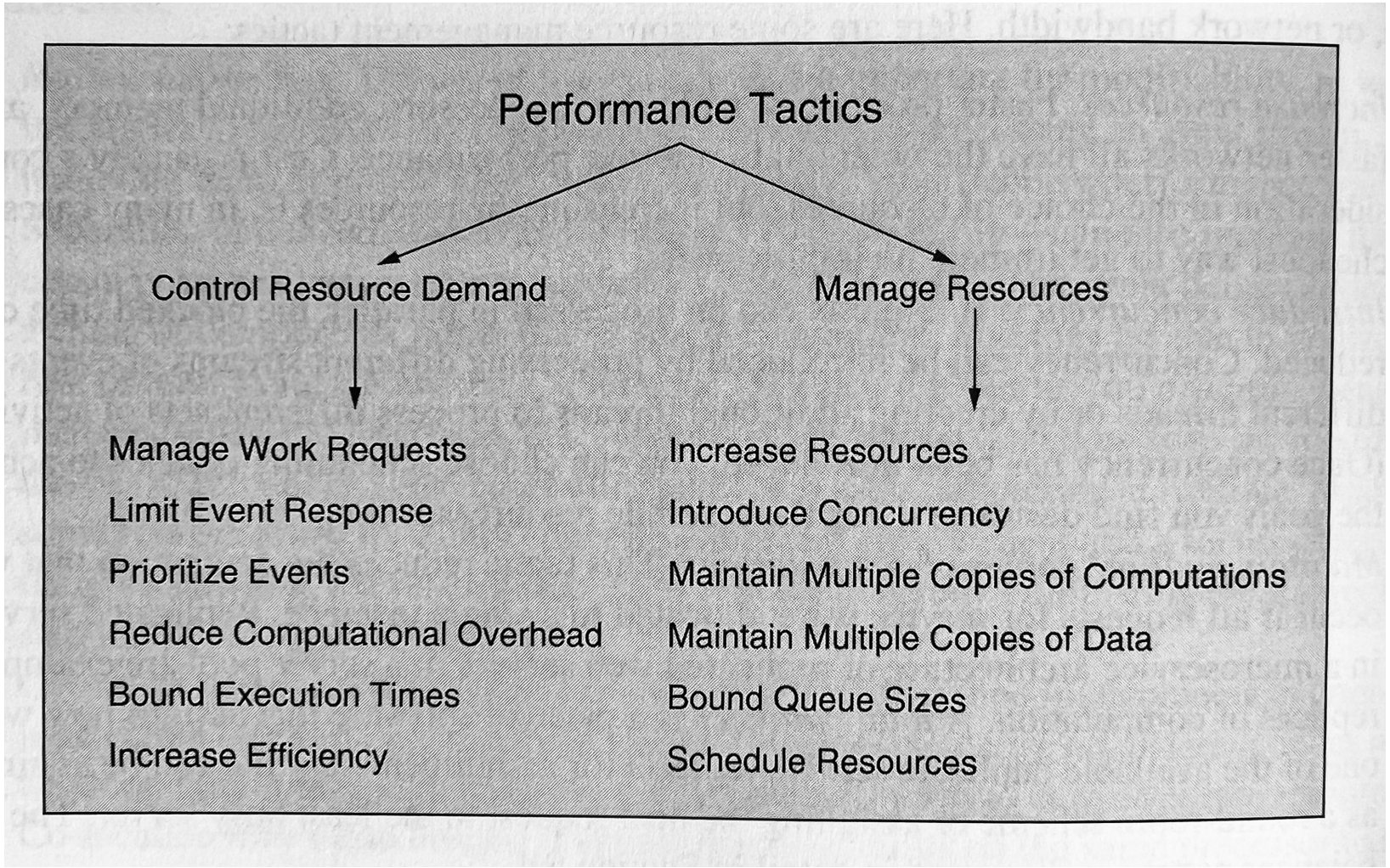
Architecture: SPOF and Tactics



Availability Tactics



Performance



Fileformats: Avro & Parquet

SDU

		
Format Type	Column-based	Row-based
Built For/with	Optimized column-wise compression and querying in a splittable file format designed for efficient Map-Reduce processing	Compact binary storage and exchange of records, with schema evolution and support for many different programming languages
Schema Storage	Column metadata stored at the end of the file (allows for fast, one-pass writing)	Stored in human-readable JSON format at the beginning of each message or file
Use Case	Quickly query all the values from a particular column across a very large dataset. For example, compute the average price of purchases over millions of purchase records	Share entire records between applications. For example, event data of in-app purchases for use by many downstream applications such as logging, auditing, and business analytics

How do they compare?

- Data format for Data Serialization
- Touple based
  - Great when you read entire records
- Schema Based
- Schema Evolution
  - Can change current props and add

- Data format for Data Serialization
- Column based
  - Great for wide tables, where only few are used for each query
- Schema Based
- Schema Evolution
  - Can add props, but not change them
- Good for nested datasets

Avro Primitives

Type	Description	Schema
`null`	The absence of a value	`"null"`
`boolean`	A binary value	`"boolean"`
`int`	32-bit signed integer	`"int"`
`long`	64-bit signed integer	`"long"`
`float`	Single-precision (32-bit) IEEE 754 floating-point number	`"float"`
`double`	Double-precision (64-bit) IEEE 754 floating-point number	`"double"`
`bytes`	Sequence of 8-bit unsigned bytes	`"bytes"`
`string`	Sequence of Unicode characters	`"string"`

Avro Complex Data Types

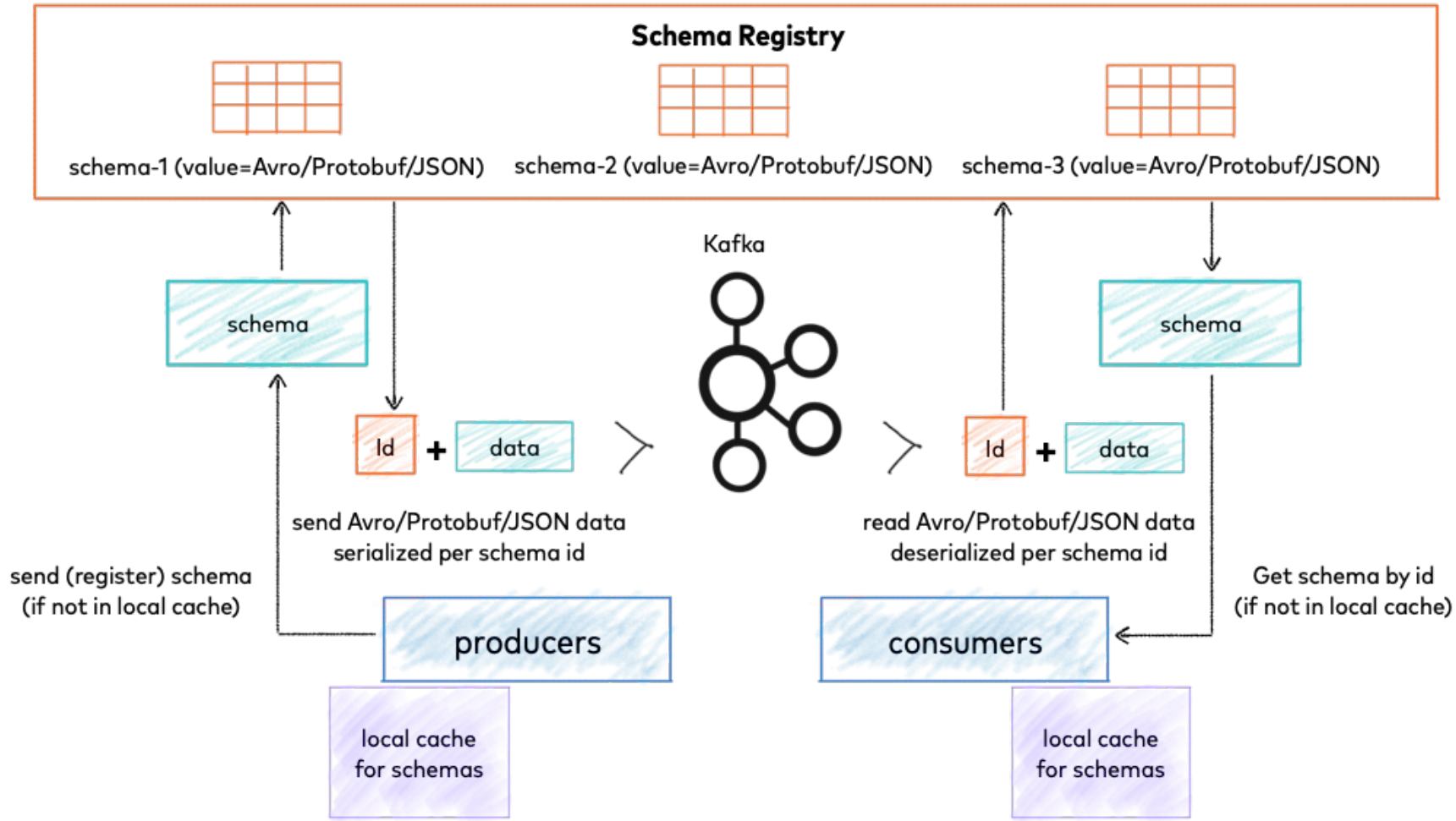
Type	Description	Schema example
`array`	An ordered collection of objects. All objects in a particular array must have the same schema.	`{ "type": "array", "items": "long" }`
`map`	An unordered collection of key-value pairs. Keys must be strings and values may be any type, although within a particular map, all values must have the same schema.	`{ "type": "map", "values": "string" }`
`record`	A collection of named fields of any type.	`{ "type": "record", "name": "WeatherRecord", "doc": "A weather reading.", "fields": [ {"name": "year", "type": "int"}, {"name": "temperature", "type": "int"}, {"name": "stationId", "type": "string"} ] }`
`enum`	A set of named values.	`{ "type": "enum", "name": "Cutlery", "doc": "An eating utensil.", "symbols": ["KNIFE", "FORK", "SPOON"] }`
`fixed`	A fixed number of 8-bit unsigned bytes.	`{ "type": "fixed", "name": "Md5Hash", "size": 16 }`

SDU

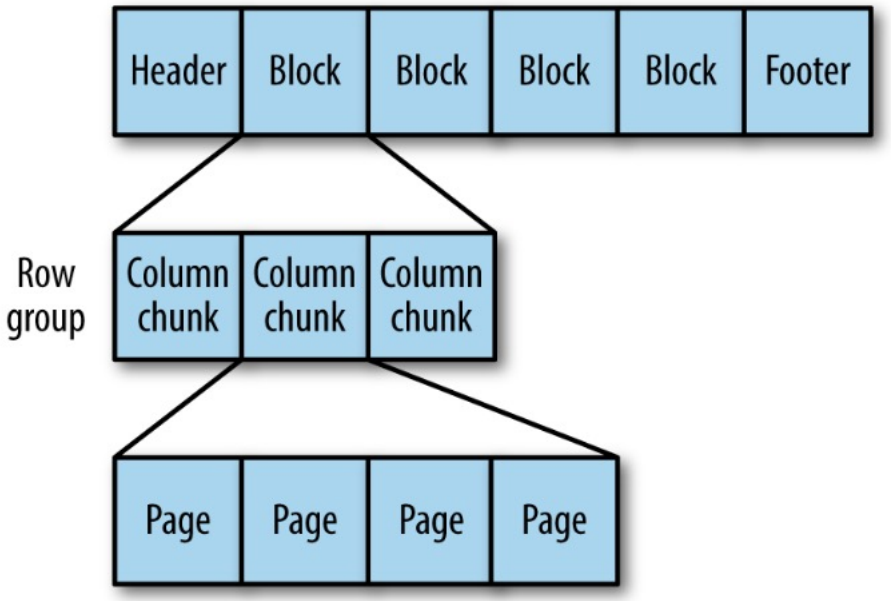
Avro Schema

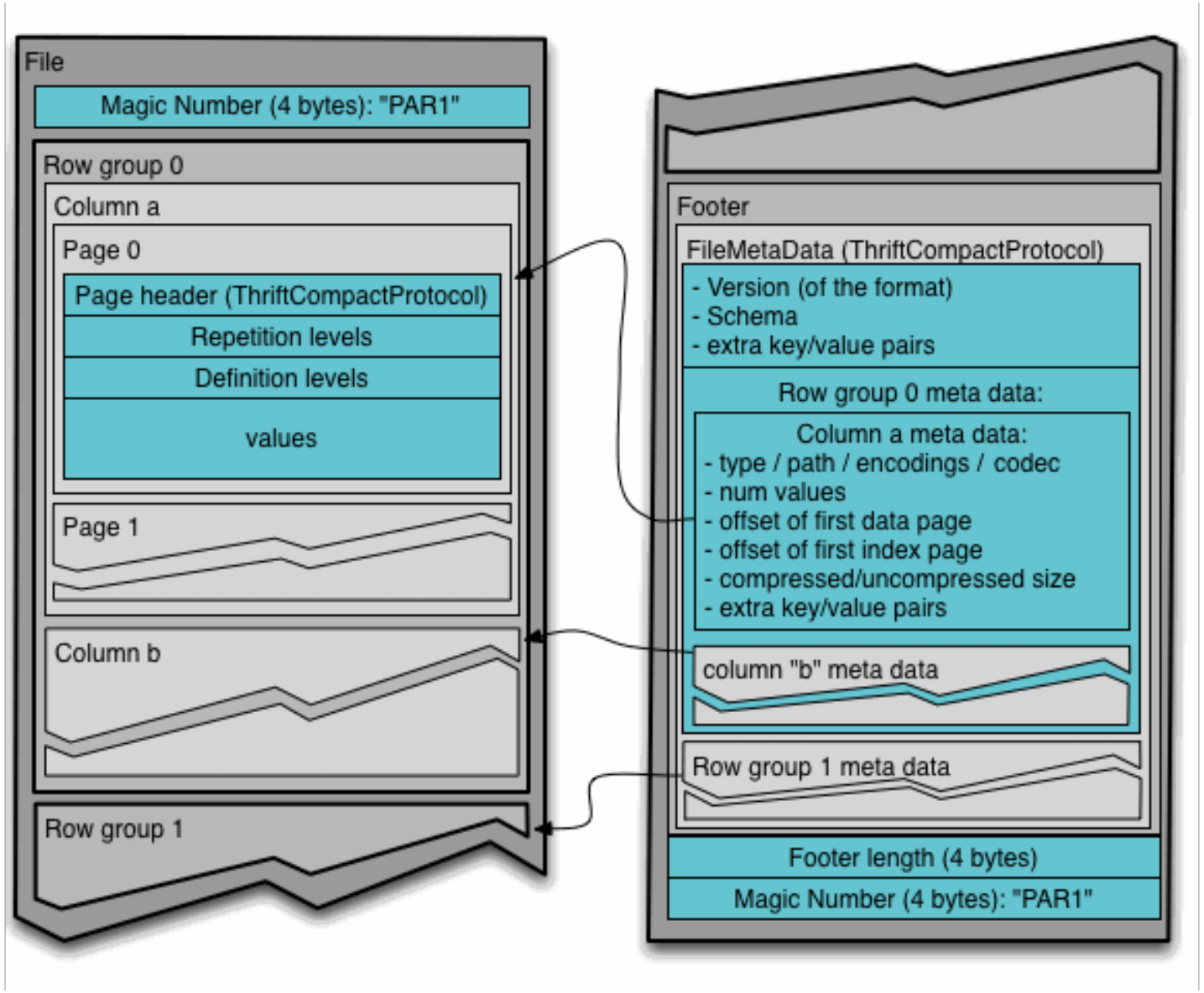
```
{"namespace": "example.avro",
 "type": "record",
 "name": "User",
 "fields": [
     {"name": "name", "type": "string"},
     {"name": "favorite_number",  "type": ["int", "null"]},
     {"name": "favorite_color", "type": ["string", "null"]}
 ]
}
```

Usage example



Parquet Schema and Structure





Parquet Primitives

Type	Description
`boolean`	Binary value
`int32`	32-bit signed integer
`int64`	64-bit signed integer
`int96`	96-bit signed integer
`float`	Single-precision (32-bit) IEEE 754 floating-point number
`double`	Double-precision (64-bit) IEEE 754 floating-point number
`binary`	Sequence of 8-bit unsigned bytes
`fixed_len_byte_array`	Fixed number of 8-bit unsigned bytes

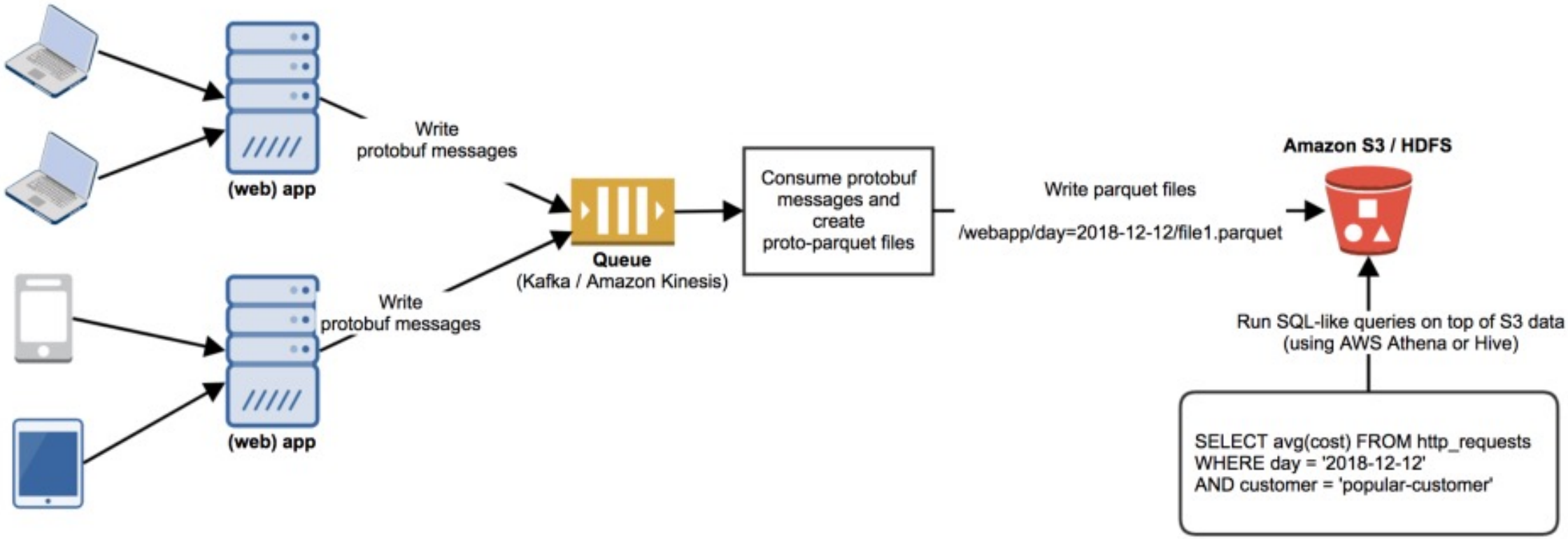
Parquet Logical Types

Logical type annotation	Description	Schema example
UTF8	A UTF-8 character string. Annotates `binary`.	`message m { required binary a (UTF8); }`
ENUM	A set of named values. Annotates `binary`.	`message m { required binary a (ENUM); }`
DECIMAL(*precision*,*scale*)	An arbitrary-precision signed decimal number. Annotates `int32`, `int64`, `binary`, or `fixed_len_byte_array`.	`message m { required int32 a (DECIMAL(5,2)); }`
DATE	A date with no time value. Annotates `int32`. Represented by the number of days since the Unix epoch (January 1, 1970).	`message m { required int32 a (DATE); }`
LIST	An ordered collection of values. Annotates `group`.	`message m { required group a (LIST) { repeated group list { required int32 element; } } }`
MAP	An unordered collection of key-value pairs. Annotates `group`.	`message m { required group a (MAP) { repeated group key_value { required binary key (UTF8); optional int32 value; } } }`

Parquet Schema Example

```
message WeatherRecord {
  required int32 year;
  required int32 temperature;
  required binary stationId (UTF8);
}
```

Parquet Usage Example



Like to know more about parquet?

→Read the Dremel Paper:

→https://storage.googleapis.com/pub-tools-public-publication-data/pdf/36632.pdf

Exercises!