Big Data Processing and Applications – Lecture 5

Ioana Ciuciu
ioana.ciuciu@ubbcluj.ro

Course structure

Date	Course/Week	Title
03.10.2025	1	Introduction to Data Science and Big Data (Part 1)
10.10.2025	2	Introduction to Data Science and Big Data (Part 2)
17.10.2025	3	Industrial standards for data mining projects. Big data case studies from industry – invited lecture from Bosch
24.10.2025	4	Data systems and the lambda architecture for big data
31.10.2025	5	Lambda architecture: batch layer
	6	Lambda architecture: serving layer
	7	Lambda architecture: speed layer
	8	NoSQL Solutions for Big Data – invited lecturer from UBB
	9	Data Ingestion
	10	Introduction to SPARK – invited lecture from Bosch
	11	Data visualization
	12	Presentation research essays
	13	Presentation research essays + Project Evaluation during Seminar
	14	Presentation research essays + Project Evaluation during Seminar

Slight modifications in the structure are possible

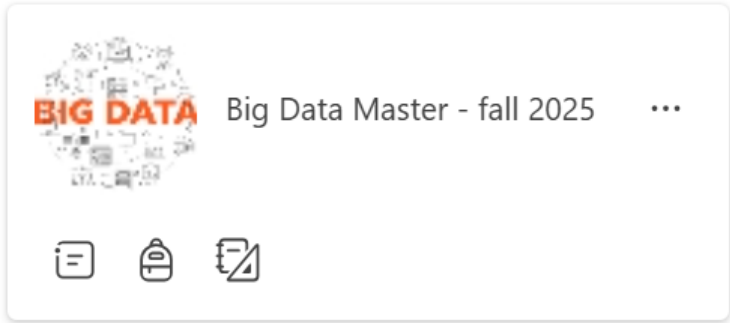
Semester Project

- Team-based (2-5 students with precise roles)
- Multidisciplinary: 3 CS Master Programes (HPC, SI, DS) + Master in Bioinformatics
- Real use cases: collaboration with local IT industry - TBA
- High degree of autonomy in selecting the topic and proposing the solution
- Implementation prototype
- Prototype demo & evaluation – student workshop (last seminar – weeks 13 & 14)
- *Best projects – disseminated in various events (TBD), scientifically disseminated (workshops/conferences/journals) AND/ OR possibility for a dissertation thesis*

Evaluation

- The final grade will be computed as follows:
  - 50% semester project (must be >= 5)
  - 50% research presentation or written exam (must be >= 5)

- Semester project
  - Details available on the course team
    - MS Team: **Big Data Master - fall 2025**
    - Access code: **j1exenq**



Agenda

Lambda architecture: Batch Layer

- Big data storage: the master dataset
  - Data model for Big Data (Chapter 2)
  - Storage requirements for Big Data (Chapter 4)
- Computing arbitrary functions on the master dataset (Chapter 6)
- Hadoop Ecosystem

Additional support material: **Nathan Matz, James Warren**, Big Data: *Principles and Best Practices of Scalable Real Time Data Systems*

Recall: Lambda architecture

Big Data Systems: Lambda Architecture

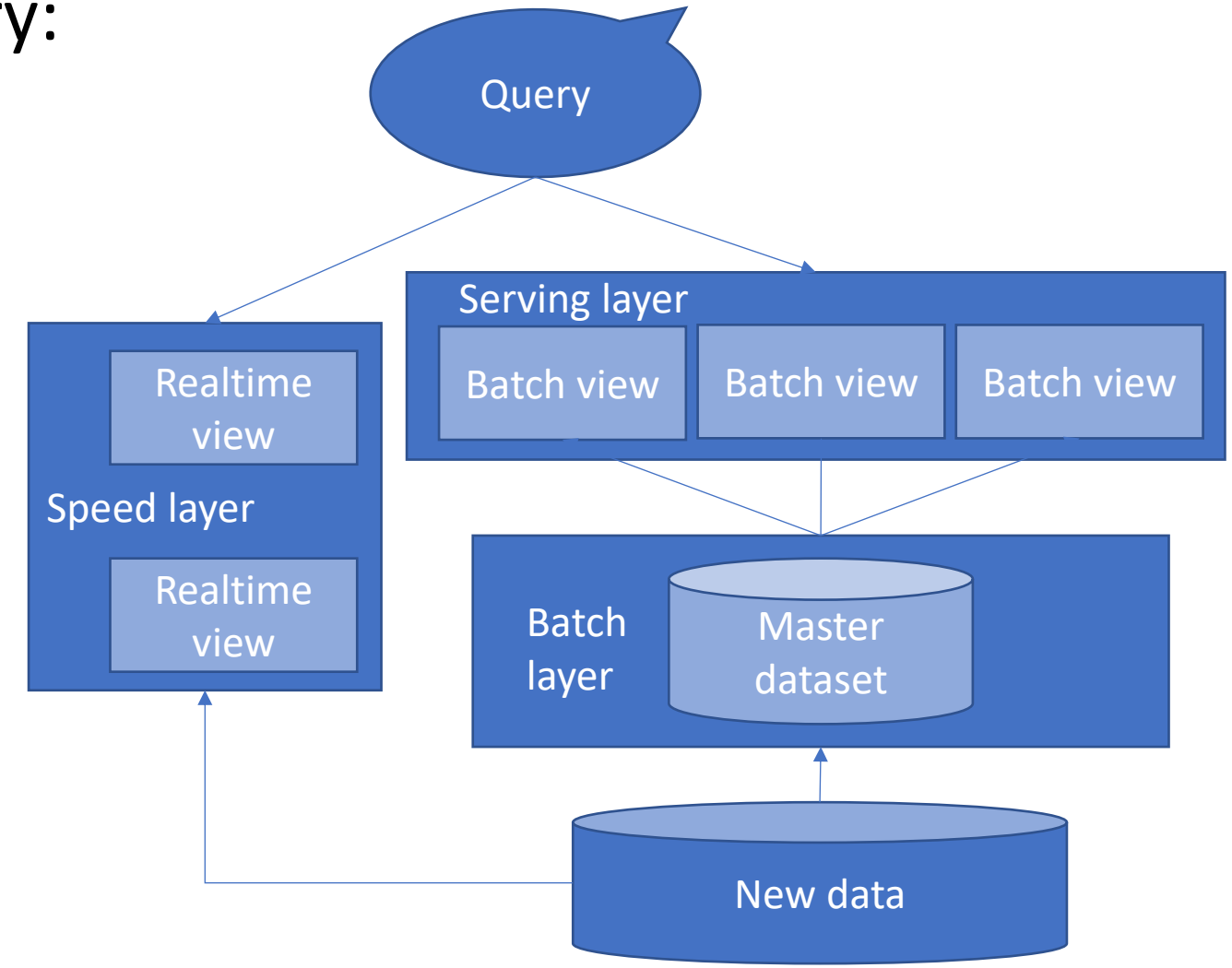
Lambda Architecture summary:

Three equations to summarize the Lambda Architecture.

batch view = function (all data)

realtime view = function (realtime view, new data)

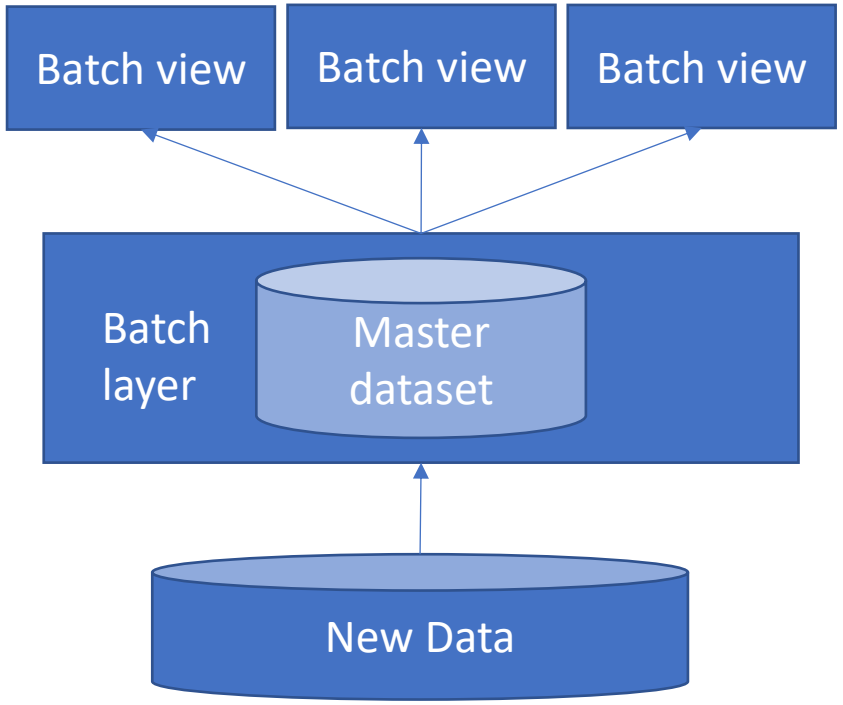
query = function (batch view, realtime view)



Lambda Architecture: the batch layer

Batch layer: implements the equation below

batch view = function (all data)



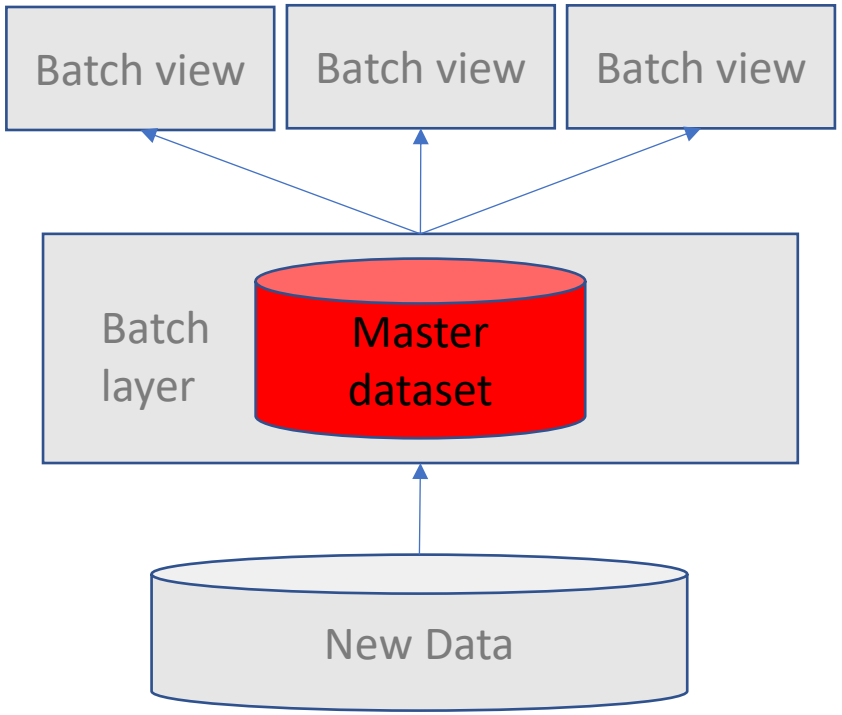
Batch layer functionalities

1. Stores a master copy of the dataset
2. Precomputes batch views

Main challenges

- The master dataset is constantly growing
- Compute arbitrary functions on that dataset

Lambda Architecture: the batch layer



Key properties of the master dataset

1. It is the source of truth in the Lambda architecture
2. Even though the serving and/or speed layer are lost, the whole application can be reconstructed from the master dataset
3. The master dataset is the only part that absolutely must be protected from corruption

Two main components of the master dataset:

- The data model
- The storage model ***or*** how to physically store the master dataset

Batch layer storage: the data model

Batch layer storage: the data model

Quiz: What is a data model?

- An abstract representation of the data describing how elements of data relate to one another and to the properties of real-world entities.
- A function used to make inference based on the elements of data
- A description the data and the corresponding real-world entities

Batch layer storage: the data model

Quiz: What is a data model?

- An abstract representation of the data describing how elements of data relate to one another and to the properties of real-world entities.
- A function used to make inference based on the elements of data
- A description the data and the corresponding real-world entities

Batch layer storage: the data model

Data model: definition

A data model is an abstract model that organizes elements of data and standardizes how they relate to one another and to the properties of real-world entities.

Batch layer storage: the data model

Data model: how to define it?

Example: assume you're designing a social network

When John Doe joins your social network, he will invite his friends, family etc. What information should you store about John Doe's connections?

Option 1: The sequence of friend and unfriend events

4/10 Add Marc
4/12 Add Frank
4/16 Add Nick
4/20 Remove Frank
4/23 Add Anna
5/02 Remove Nick
5/10 Add Megan

Option 2: Current list of friends

Marc
Anna
Megan

Option 3: Current number of friends

3

Batch layer storage: the data model

Data model: how to define it?

Example: assume you're designing a social network

When John Doe joins your social network, he will invite his friends, family etc. What information should you store about John Doe's connections?

Option 1: The sequence of friend and unfriend events

4/10 Add Marc
4/12 Add Frank
4/16 Add Nick
4/20 Remove Frank
4/23 Add Anna
5/02 Remove Nick
5/10 Add Megan

Option 2: Current list of friends

Marc
Anna
Megan

Option 3: Current number of friends

3

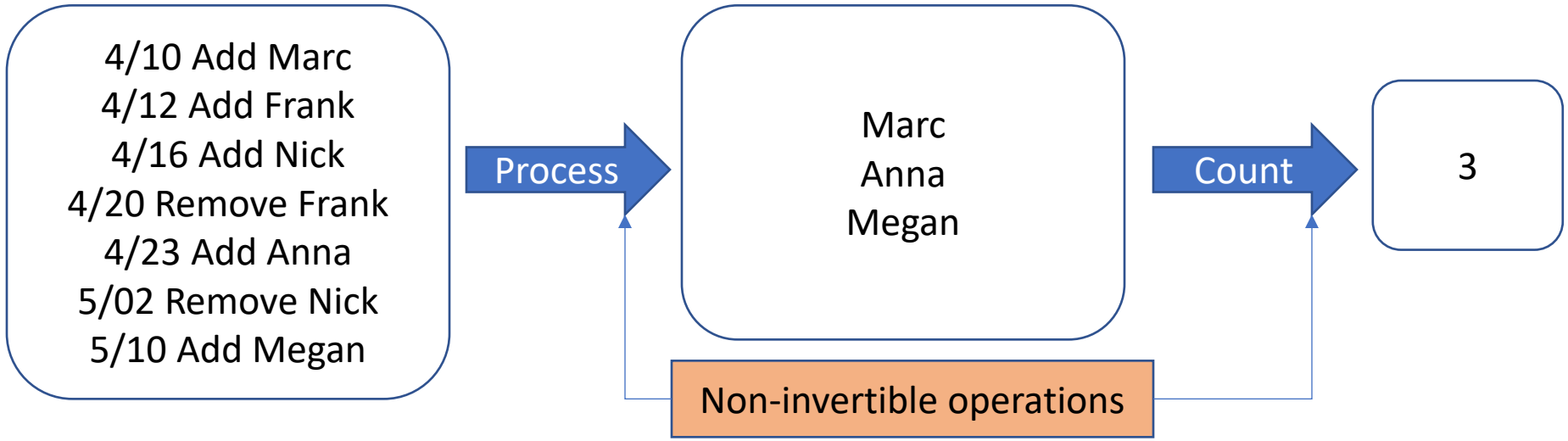
Batch layer storage: the data model

Data model: how to define it?

Option 1: The sequence of friend and unfriend events

Option 2: Current list of friends

Option 3: Current number of friends



Remarques:

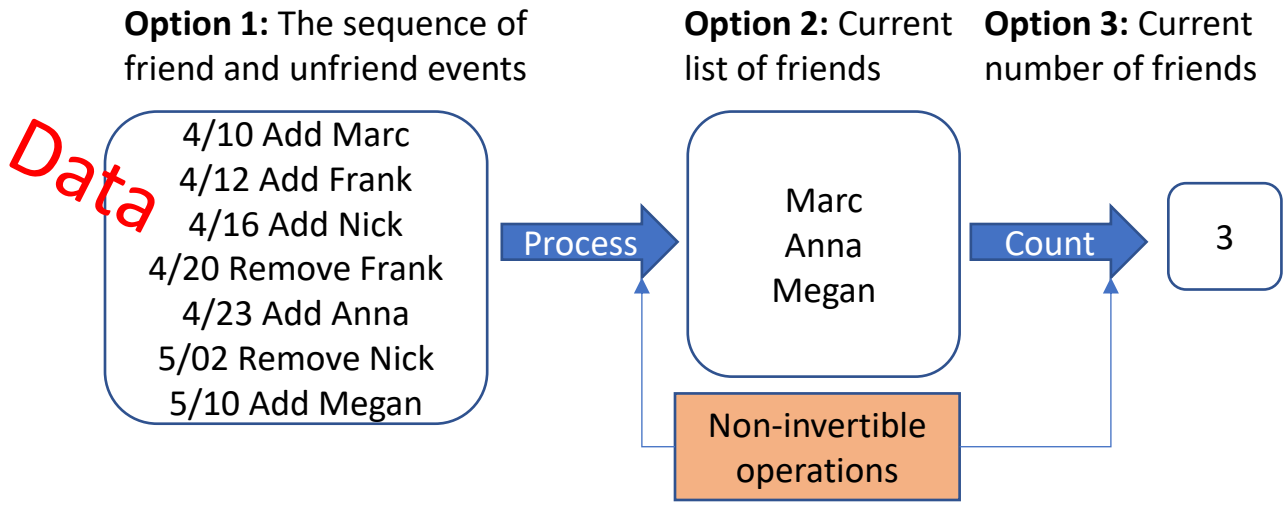
1. Each layer of information can be derived from the previous one
2. This is a one way process (the information from previous layers cannot be derived from the upstream layers)

Batch layer storage: the data model

Data model: how to define it?

What information should you store about John Doe's connections?

Illustration of the information dependency



Some definitions

1. **Information:** is the general collection of knowledge relevant to the Big Data system
2. **Data:** information that cannot be derived from anything else. Data serves as axioms from which anything else derives
3. **Queries:** are questions you ask about your data
4. **Views:** information that has been derived from your base data

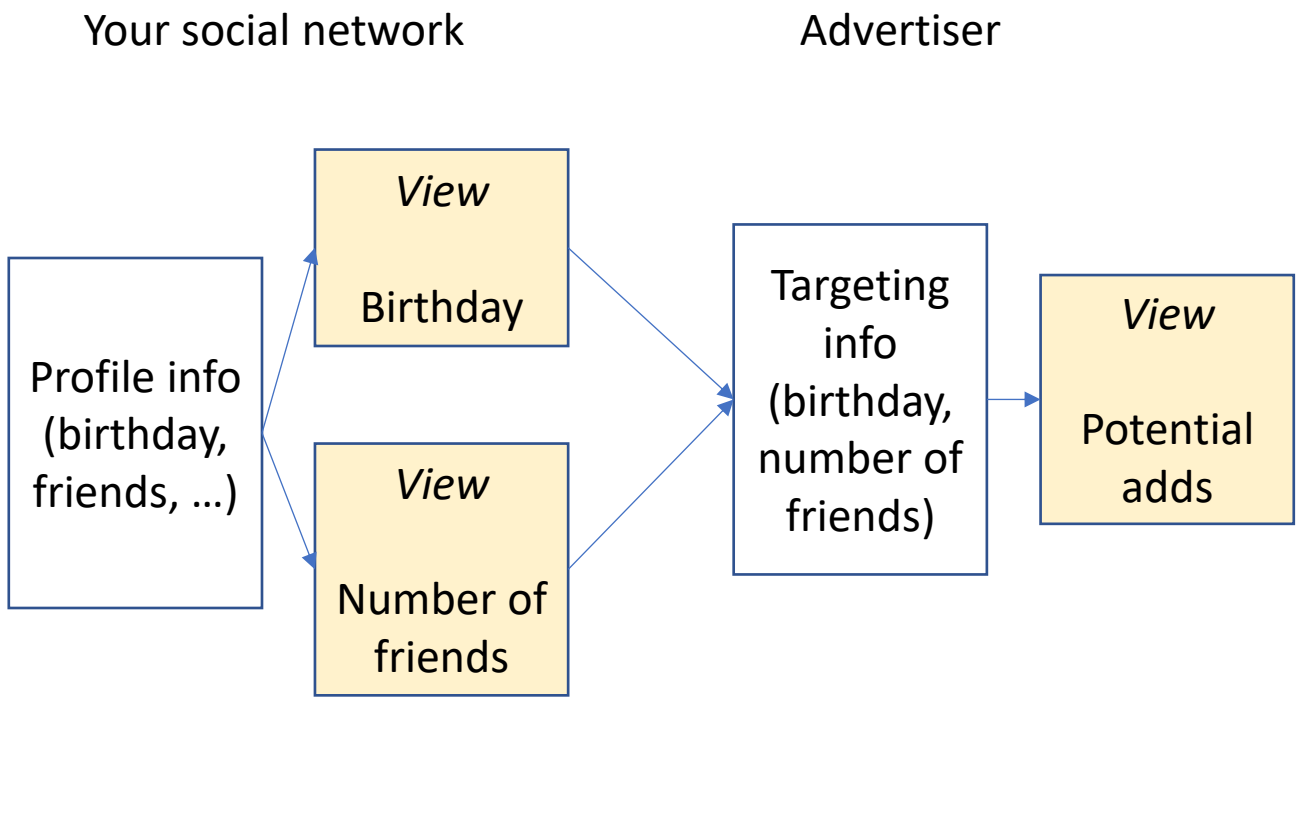
Batch layer storage: the data model

Data model: how to define it?

Some definitions

1. **Information:** is the general collection of knowledge relevant to the Big Data system
2. **Data:** information that cannot be derived from anything else. Data serves as axioms from which anything else derives
3. **Queries:** are questions you ask about your data
4. **Views:** information that has been derived from your base data

One person's view can be another's data



Batch layer storage: the data model

Key properties of data

1. Data is raw (rawness)
2. Data is immutable (immutability)
3. Data is eternally true (perpetuity)

Batch layer storage: the data model

Key properties of data: data is raw (rawness)

Big Data systems are designed to answer questions about information you acquired in the past.

The user should be able to answer as many questions as possible.

Definitions

Data rawness -> the inner capacity of the data stored to answer questions.

Raw data or primary data

- Collected directly related to their object of study

VS

Secondary data

- Data derived from raw data
- It no longer contains all of the information of the original investigation

Q: Why storing raw data is so valuable?

Batch layer storage: the data model

Quiz: Why storing raw data is so valuable?

Q: Which statement is relevant in the context of big data applications

A: Storing raw data is valuable because you do not know all the question you want to answer in advance

B: Storing raw data is not recommended as it dramatically increases the resources needed for storage

C: Raw data cannot be directly used; therefore, raw data should not be stored.

Batch layer storage: the data model

Quiz: Why storing raw data is so valuable?

Q: Which statement is relevant in the context of big data applications

A: Storing raw data is valuable because you do not know all the question you want to answer in advance

B: Storing raw data is not recommended as it dramatically increases the resources needed for storage

C: Raw data cannot be directly used; therefore, raw data should not be stored.

Batch layer storage: the data model

Key properties of data: data is raw (rawness)

What information you should store as your raw data?

Unstructured data vs semantically normalized data

Semantic normalization refers to the process of reshaping free-form information into a structured form

Unstructured data is rawer than normalized data

Better store unstructured data because you can normalize your data later if your normalization algorithm gets better

Rule of thumb: if your algorithm for extracting data is simple and accurate you should store the results of your algorithm. If the algorithm is subject to change, store the unstructured form of the data

Batch layer storage: the data model

Key properties of data: data is immutable (immutability)

Data immutability means that data cannot be altered (modified or deleted) once it is stored.

Basic storage functions	Traditional database systems	Big Data systems
Create	√	√
Read	√	√
Update	√	
Delete	√	

Advantages of immutable schemas:

1. Human fault tolerance – *no data can be lost*
2. Simplicity

Disadvantages of immutable schemas:

Uses more storage than mutable schemas

Batch layer storage: the data model

Key properties of data: data is eternally true (perpetuity)

Data perpetuity is a consequence of immutability and it means that each piece of data is true in perpetuity.

Data is eternally true by tagging each piece of data with a timestamp.

Special cases when data is deleted:

1. Garbage collection: delete all data which has low value
2. Regulations: government regulations might impose to delete data from your storage under certain conditions

Batch layer storage: the data model

Data models: the fact-based model

Different ways to store data: relational tables, JSON documents, XML files etc.

The fact-based model deconstructs the data into fundamental units called facts.

Three main properties of facts:

1. Atomicity – facts cannot be subdivided further into meaningful components
2. Timestamped – ensures data immutability and perpetuity
3. Identifiability – you can write the same fact to the master dataset multiple times without changing the semantics of the master dataset. Duplicates are filtered out using queries.

Batch layer storage: the data model

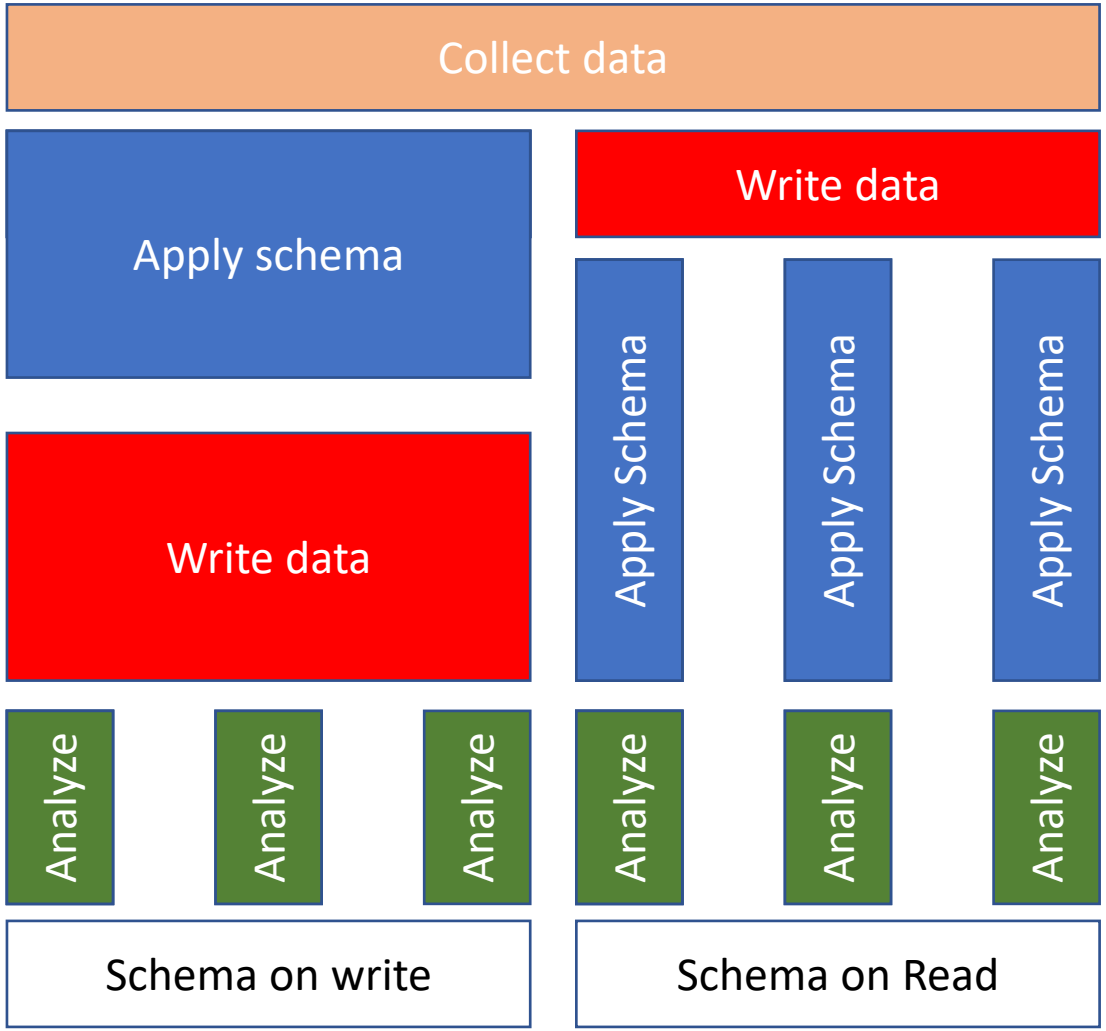
Data models: the fact-based model

Benefits of the fact-based model

1. **Data is *queryable* at any time in the history** – all history is stored in the master dataset
2. **Tolerates human error** – erroneous facts are simply deleted
3. **Handles partial information** – storing facts makes it easy to handle partial information about the entity. Facts do capture partial information about the entity
4. **Has the advantages of both normalized and denormalized forms** – data is stored on two layers. On the batch layer, data is stored in a normalized form to reduce redundancy and ensure consistency. On the serving layer, the data is stored in a denormalized form to ensure fast data retrieval.

Batch layer storage: the data model

Data models: schema on write vs schema on read data systems



Batch layer storage: the data model

Data models: quiz

Q: Which statement is correct?

RDMS are:

A: Schema on read systems

B: Schema on write systems

C: Neither schema on read nor schema on write

Batch layer storage: the data model

Data models: quiz

Q: Which statement is correct?

RDMS are:

A: Schema on read systems

B: Schema on write systems

C: Neither schema on read nor schema on write

Batch layer storage: the data model

Data models: the need of an enforceable schema

Here we define ***schema*** as a predefined structure of the data model.
A schema can be seen as a function that takes as input a piece of data and returns whether it is valid or not

Why an enforceable schemas? Example:

Representation of a user's age using JSON

```
{"id": 3, "field": "age", "value": 28, "timestamp": 1333589484}
```

Representation of other users' age using JSON

```
{"name": "Anna", "field": "age", "value": 28, "timestamp": "2012/03/29 08:12:24"}
```

```
{"id": 2, "field": "age", "value": 28}
```

Inconsistent formats or missing data

Batch layer storage: the data model

Data models: the need of an enforceable schema

Representation of a user's age using JSON

```
{"id": 3, "field": "age", "value": 28,
"timestamp": 1333584484}
```

Representation of other users' age using JSON

```
{"name": "Anna", "field": "age", "value": 28,
"timestamp": "2012/03/29 08:12:24"}

{"id": 2, "field": "age", "value": 28}
```

Inconsistent formats or missing data

Enforceable schemas are needed to be defined in order to use effectively the data

They require all fields are present and ensure all values are of the expected type

Main advantage: when a mistake is made an enforceable schema will give an error at that time you read the data rather than when someone is trying to use the data later

Batch layer storage: the data model

Data models: serialization

Serialization is the process of translating a data structure / schema or an object state into a format that can be stored

Serialization frameworks require a schema to validate the data.

Example of serialization frameworks: **JSON, AVRO, PARQUET, Thrift** etc.

Example: schema for storing JSON files

```
{
"$id": "https://example.com/person.schema.json",
"$schema": "http://json-schema.org/draft-07/schema#",
"title": "Person",
"type": "object",
"properties":
{
  "firstName": { "type": "string","description": "The person's first name." },
  "lastName": { "type": "string", "description": "The person's last name." },
  "age": { "description": "Age.", "type": "integer", "minimum": 0
}
}
}
```

Relational databases also have a defined schema. So what is the difference?

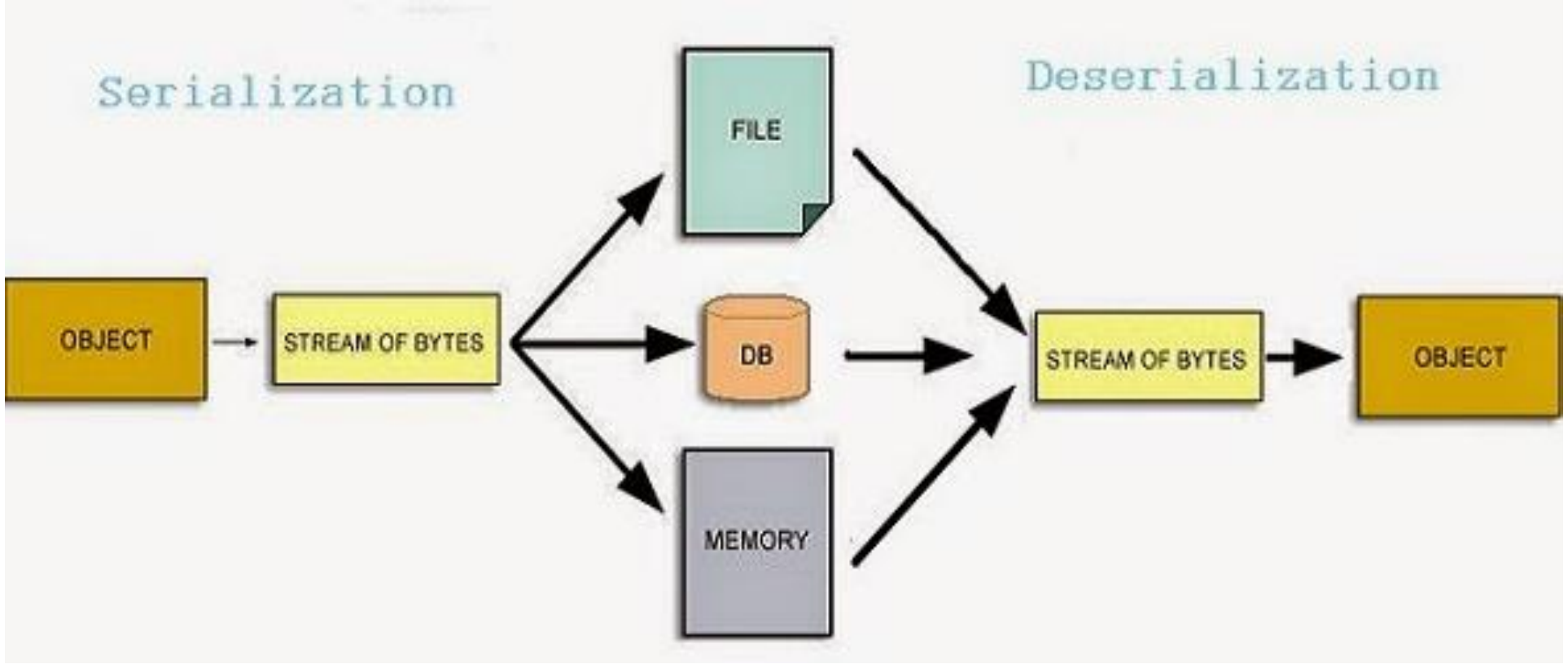
Nested objects are difficult to be represented

A schema is defined by the tables structure and the relations between the tables.
Schema migration in relational databases is difficult

Batch layer storage: the data model

Data models: serialization

Deserialization is the reverse process of creating an object from a file



Batch layer storage requirements

Storage Models

- A **storage model** is the core of any big data related systems

- It affects
  - Scalability,
  - Data structures,
  - Programming & computational models for the systems that are built on top of any big data system

- Understanding the storage model is key for understanding the entire spectrum of big data frameworks

Storage requirements for the master dataset

Storage requirements

Operation	Requisite	Discussion
Write	Efficient appends of new data	The only operation required is to add new fact, therefore it must be easy and efficient to append a new set of data objects to the master dataset.
	Scalable storage	It must be easy to scale the storage as the dataset grows
Read	Support for parallel processing	Computing the batch views requires computing functions on the entire master dataset. The batch storage must support parallel processing to handle large amounts of data
Both	Tunable storage and processing costs	The batch layer should offer the flexibility to decide how to store and compress your data to suit your needs
	Enforceable immutability	It's critical to be able to enforce immutability on your master dataset. This can be done by putting checks in place to disallow mutable operations.

Storage Models for Big Data

- Three main storage models:
  - File-based storage or *distributed filesystems*
  - Object-based storage
  - Key-value storage (will be discussed during the speed layer lecture)

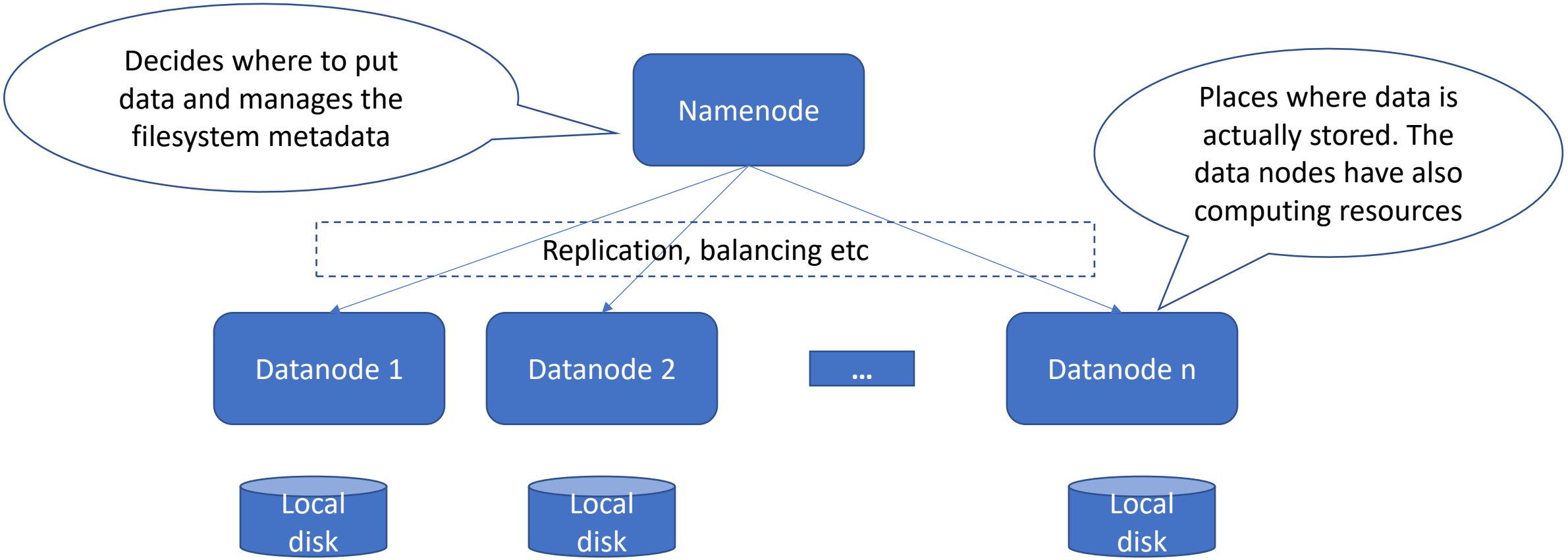
Storage Models

Distributed file systems: short introduction

- The most common storage model
  - Relatively easy to implement and use
- Inherits from the traditional file system architecture
- Considers data as files that are maintained in a hierarchical structure
- Files are added sequentially on disk
- Offer full control on the file size and the way you want to compress
- They scale by adding more machines to the cluster
- They are designed to be fault tolerant , meaning that when you lose a machine all your data will still be accessible

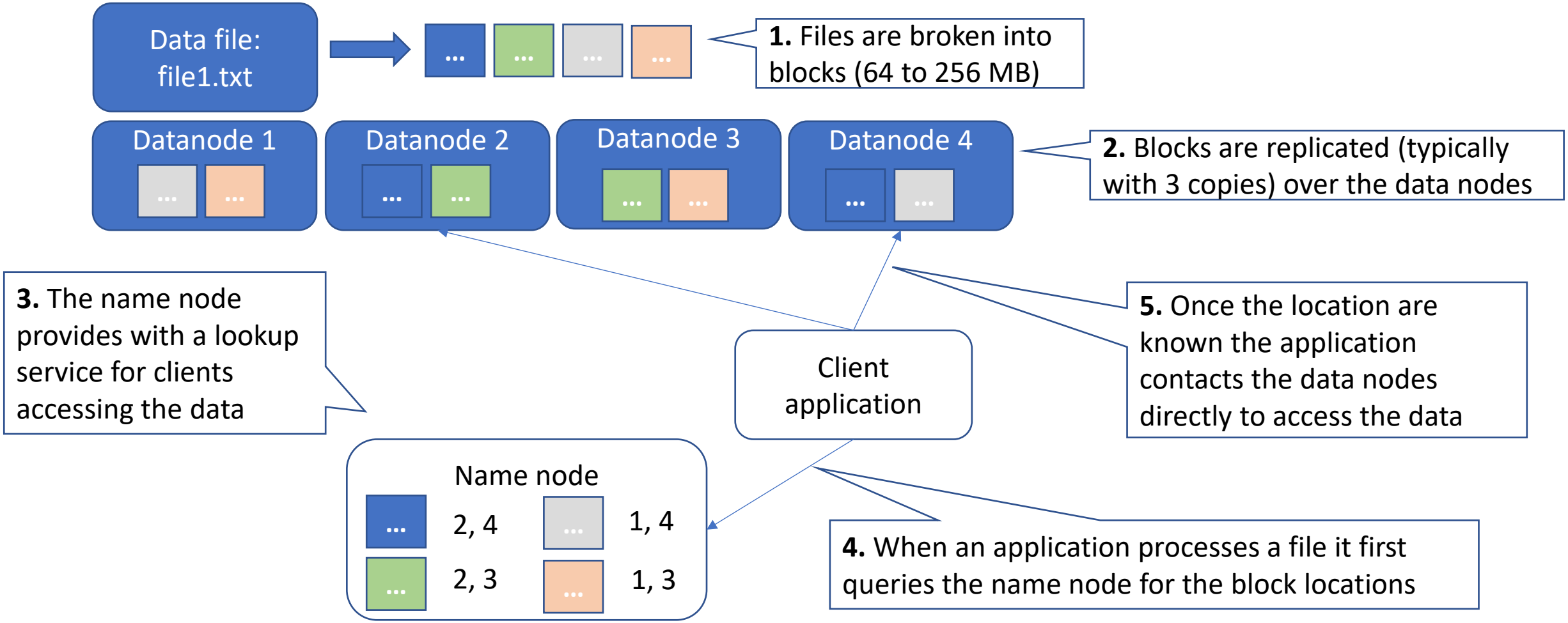
Storage Models

Distributed file systems: architecture



Storage Models

Distributed file systems: how do they work?



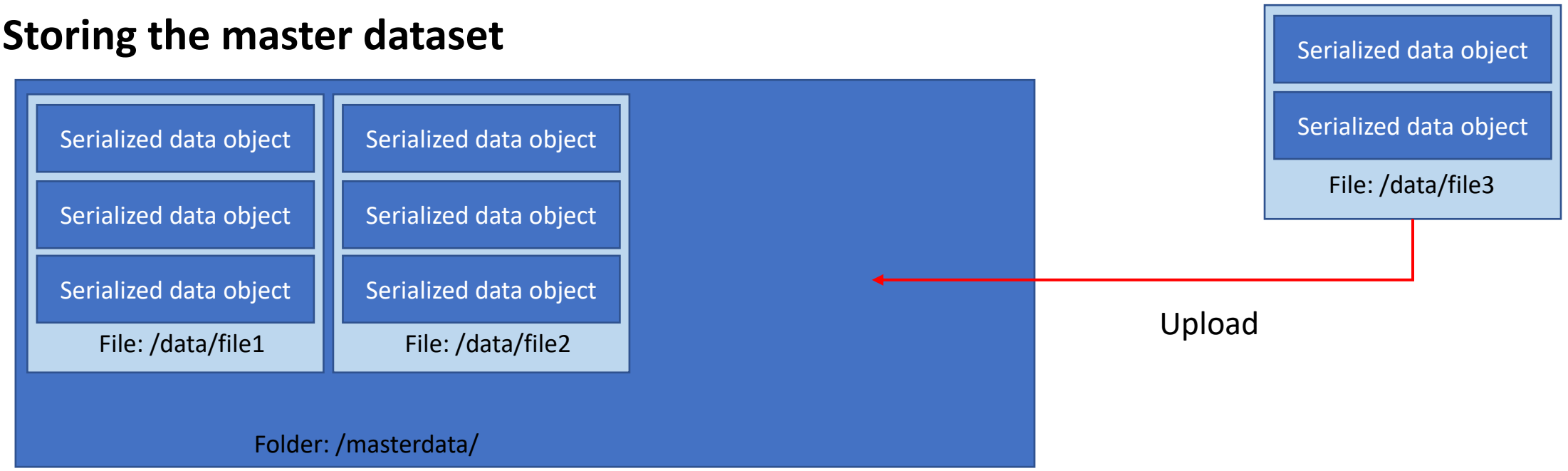
Storage Models

Distributed file systems: how do they work?

Summary:

- Files are spread across multiple machines for scalability and also enable parallel processing
- File blocks are replicated across multiple nodes for fault tolerance

Storing the master dataset



Storage Models

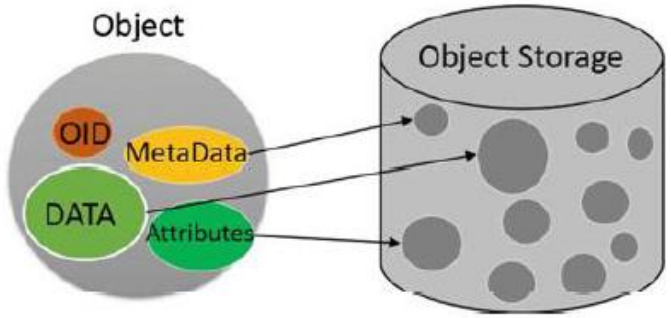
Object-based storage

Object-based storage also known as document storage.

Key features:

- Data is managed and manipulated as distinct units, called objects or documents.
- Objects are kept in a single storehouse and are not ingrained in files inside other folders.

- Object are composed of:
  - the pieces of data that make up a file
  - relevant metadata
  - attributes
  - a custom identifier.

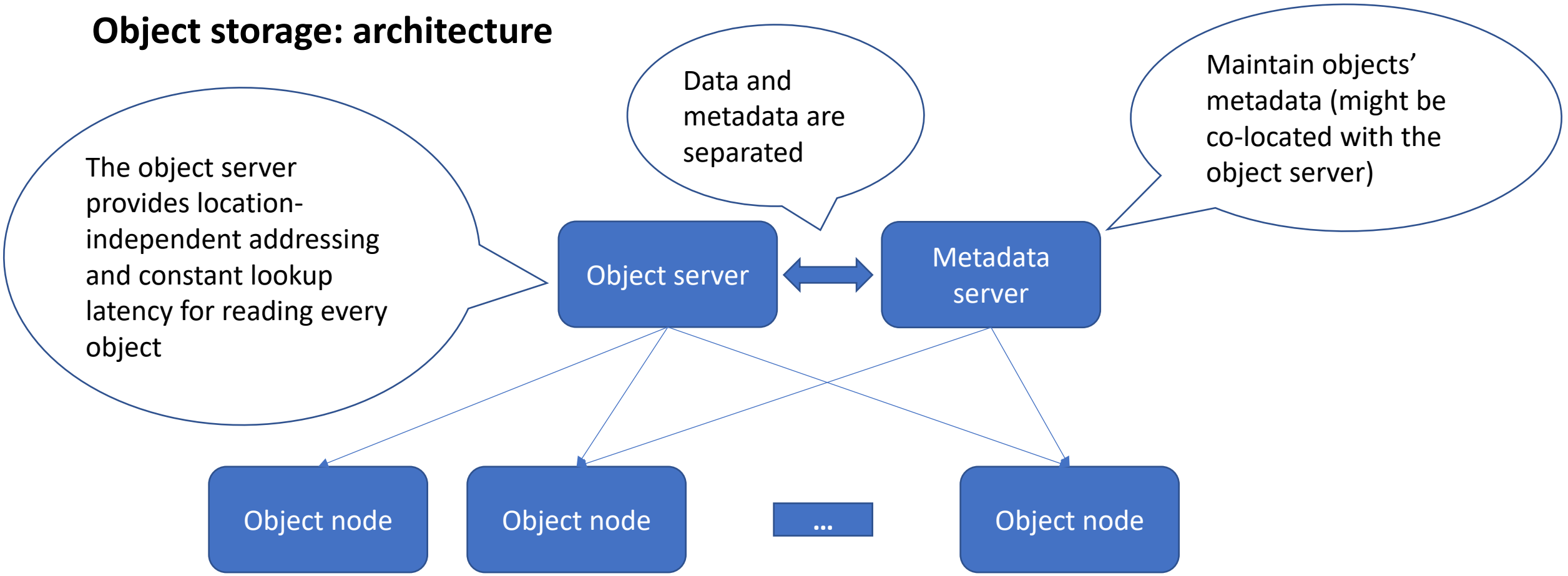


- The encapsulation of metadata to the file, allows to eliminating the tiered file structure used in file storage, and places everything into a flat address space, called a storage pool.

- Metadata is key to the success of object storage in that it provides deep analysis of the use and function of data in the storage pool.

Storage Models

Object storage: architecture



Storage Models

Object-based storage

Advantages

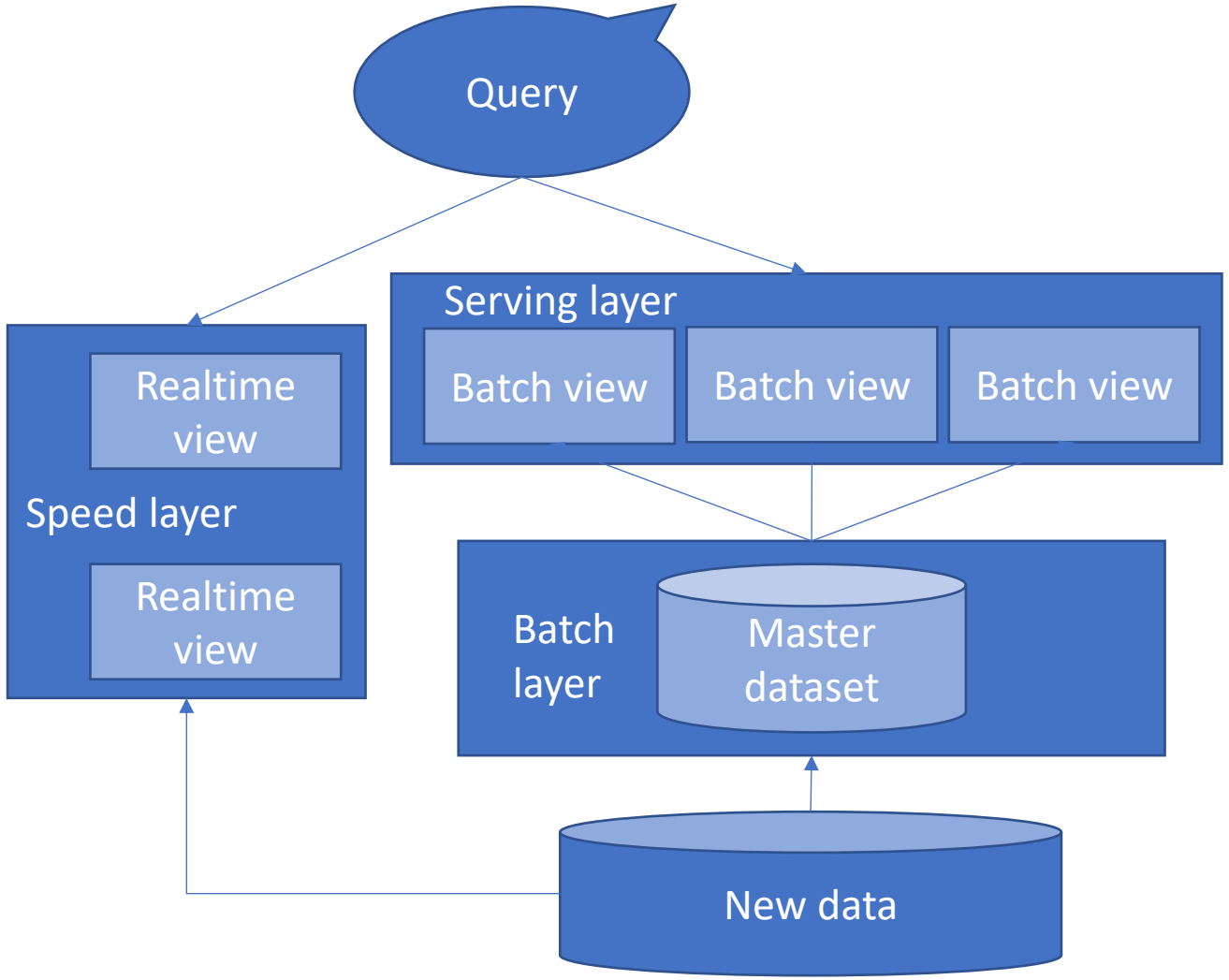
- **Greater data analytics.** Object storage is driven by metadata, and with this level of classification for every piece of data, the opportunity for analysis is far greater.
- **Infinite scalability.** Keep adding data, forever. There's no limit.
- **Faster data retrieval.** Due to the categorization structure of object storage, and the lack of folder hierarchy, you can retrieve your data much faster.
- **Reduction in cost.** Due to the scale-out nature of object storage, it's less costly to store all your data.
- **Optimization of resources.** Because object storage does not have a filing hierarchy, and the metadata is completely customizable, there are far fewer limitations than with file or block storage.

Computing arbitrary functions on the master dataset

Computing on the batch layer

Computing on the batch layer

batch view = function (all data)



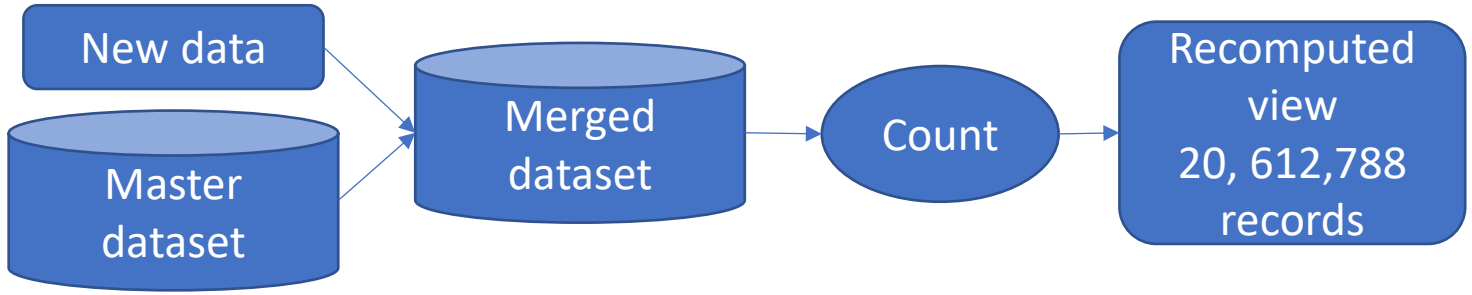
Computing on the batch layer

Computing on the batch layer

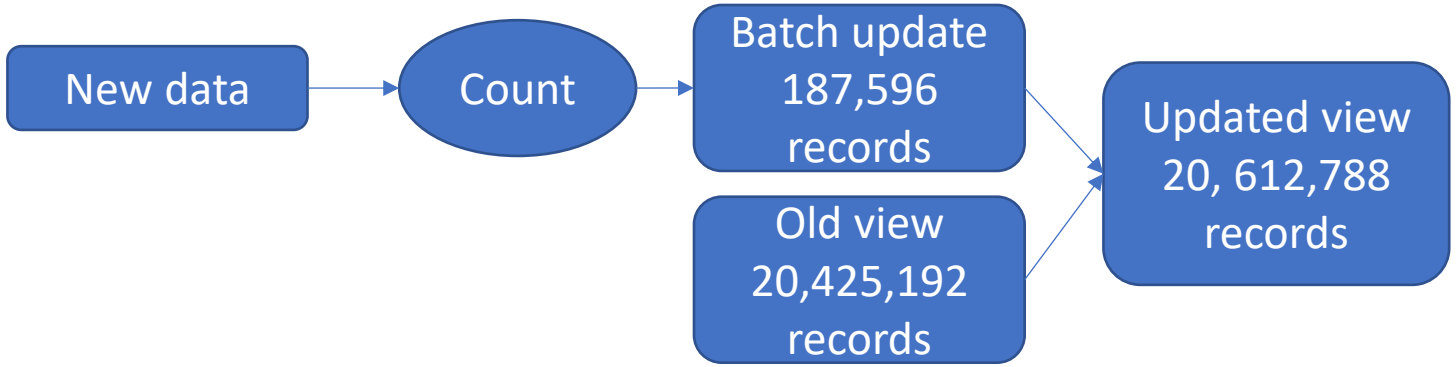
Strategies for creating and updating the batch views

batch view = function (all data)

Recomputation algorithms: updates the batch view by first adding new data to the master dataset and then computing the function



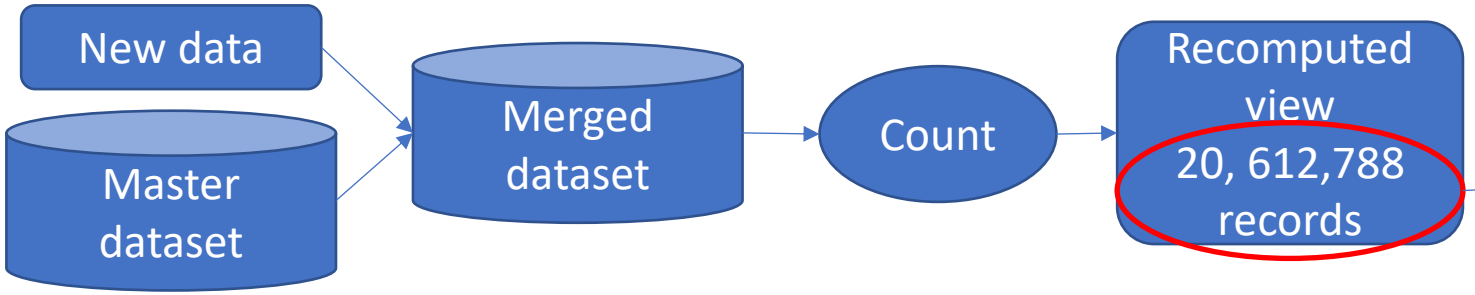
Incremental algorithms: will compute the function on the new data and then updates the batch view with the new results



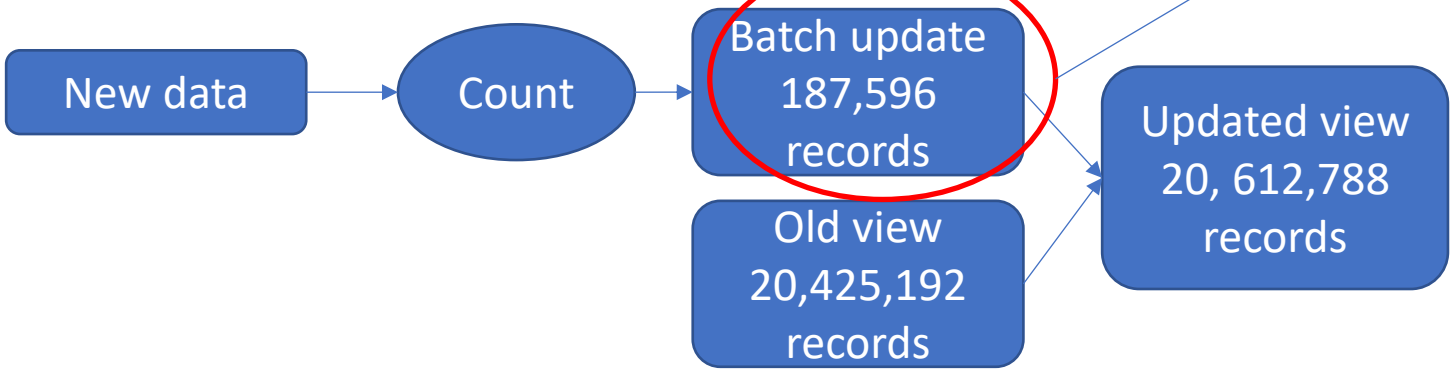
Computing on the batch layer

Computing on the batch layer

Recomputation algorithms: updates the batch view by first adding new data to the master dataset and then computing the function



Incremental algorithms: will compute the function on the new data and then updates the batch view with the new results



Is the incremental algorithm always more efficient than a recomputation algorithm?

NO! The efficiency is a trade-off between performance, human-fault tolerance and the generality of the algorithm.

Computing on the batch layer

Computing on the batch layer

Recomputation algorithms vs incremental algorithms

	Recomputation algorithms	Incremental algorithms
Performance	Requires computational effort to process the entire dataset	Requires less computational resources but may generate much larger batch views
Human-fault tolerance	Extremely tolerant of human errors because the batch views are constantly rebuilt	Doesn't facilitate repairing errors in the batch view; repairs are ad-hoc and may require estimates
Generality	Complexity of the algorithm is addressed during precomputation, resulting in simple batch views of low-latency, on-the-fly processing	Requires special tailoring; may shift complexity to the on-the-fly query processing
Conclusion	Essential for supporting a robust data-processing system ☺	Can increase efficiency on your system but only as a supplement to recomputation algos ☹

Computing on the batch layer

MapReduce: a paradigm for Big Data computing

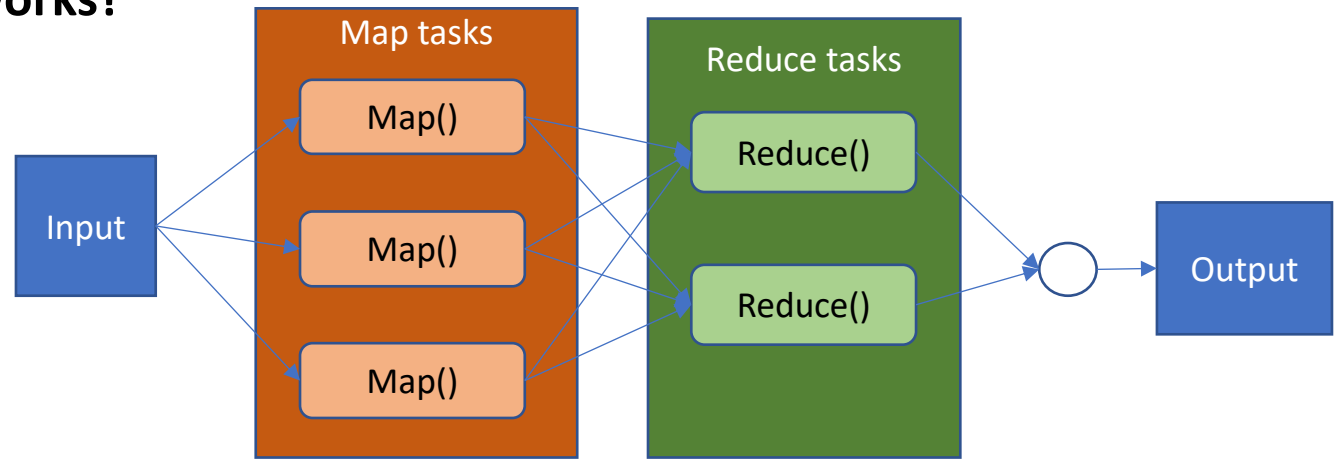
MapReduce properties

1. It is a distributed computing paradigm originally pioneered by Google.
2. Provides with primitive for scalable and fault-tolerant batch computation.
3. The computations are written in terms of *map* and *reduce* functions that manipulate key/value pairs.
4. The primitives are generic enough to implement nearly any function.
5. The functions are executed over the master dataset in a distributed and robust manner.

Computing on the batch layer

MapReduce: a paradigm for Big Data computing

MapReduce how it works?



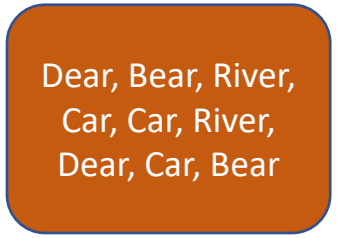
1. The first is the *Map* job, where a block of data is read and processed to produce key-value pairs as intermediate outputs.
2. The output of a Mapper or map job (key-value pairs) is input to the Reducer.
3. The reducer receives the key-value pair from multiple map jobs.
4. Then, the reducer aggregates those intermediate data tuples (intermediate key-value pair) into a smaller set of tuples or key-value pairs which is the final output.

Computing on the batch layer

MapReduce: a paradigm for Big Data computing

MapReduce how it works? Example: word count

Input: a file
containing words



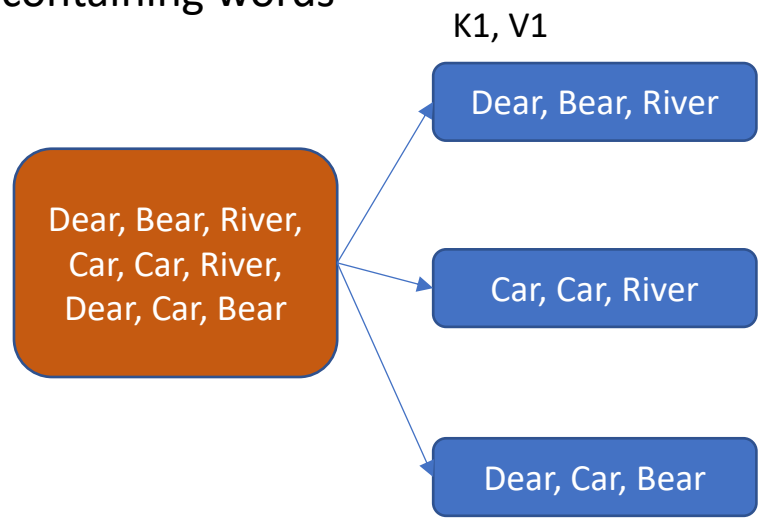
Computing on the batch layer

MapReduce: a paradigm for Big Data computing

MapReduce how it works? Example: word count

Input: a file containing words

Split

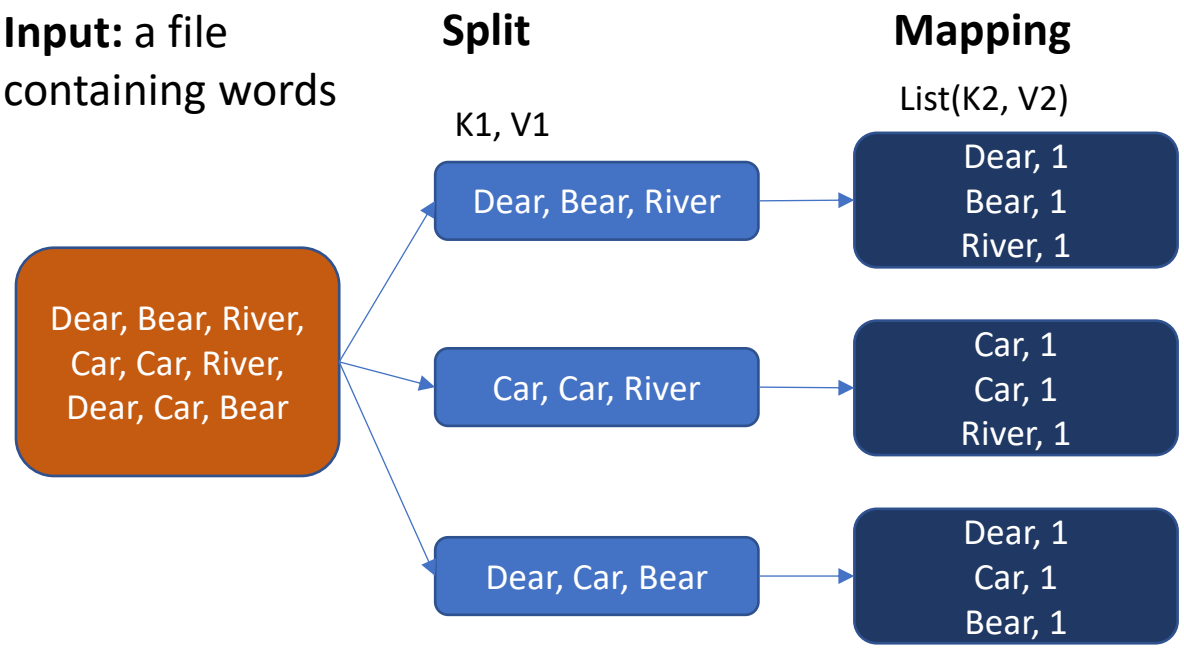


First, we divide the input into three splits as shown in the figure. This will distribute the work among all the map nodes.

Computing on the batch layer

MapReduce: a paradigm for Big Data computing

MapReduce how it works? Example: word count

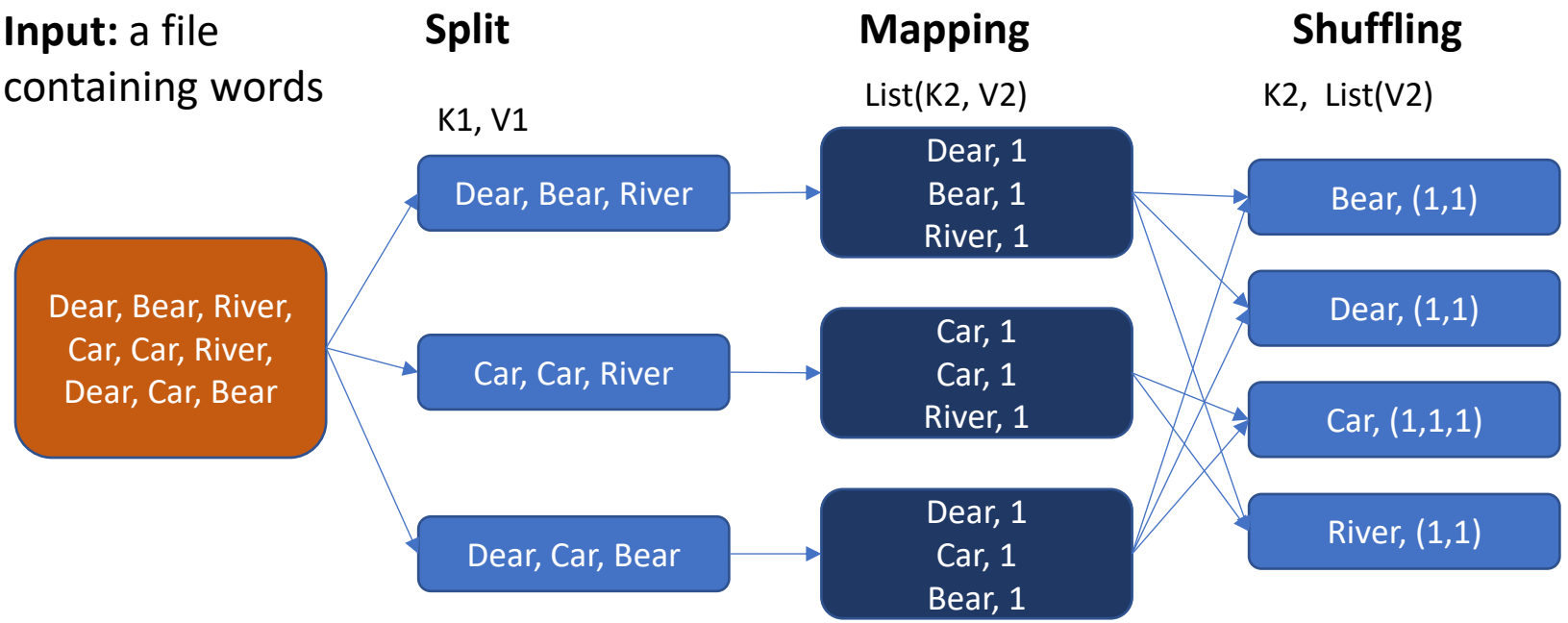


Then, we tokenize the words in each of the mappers and give a hardcoded value (1) to each of the tokens or words. The rationale behind giving a hardcoded value equal to 1 is that every word, in itself, will occur once.

Computing on the batch layer

MapReduce: a paradigm for Big Data computing

MapReduce how it works? Example: word count

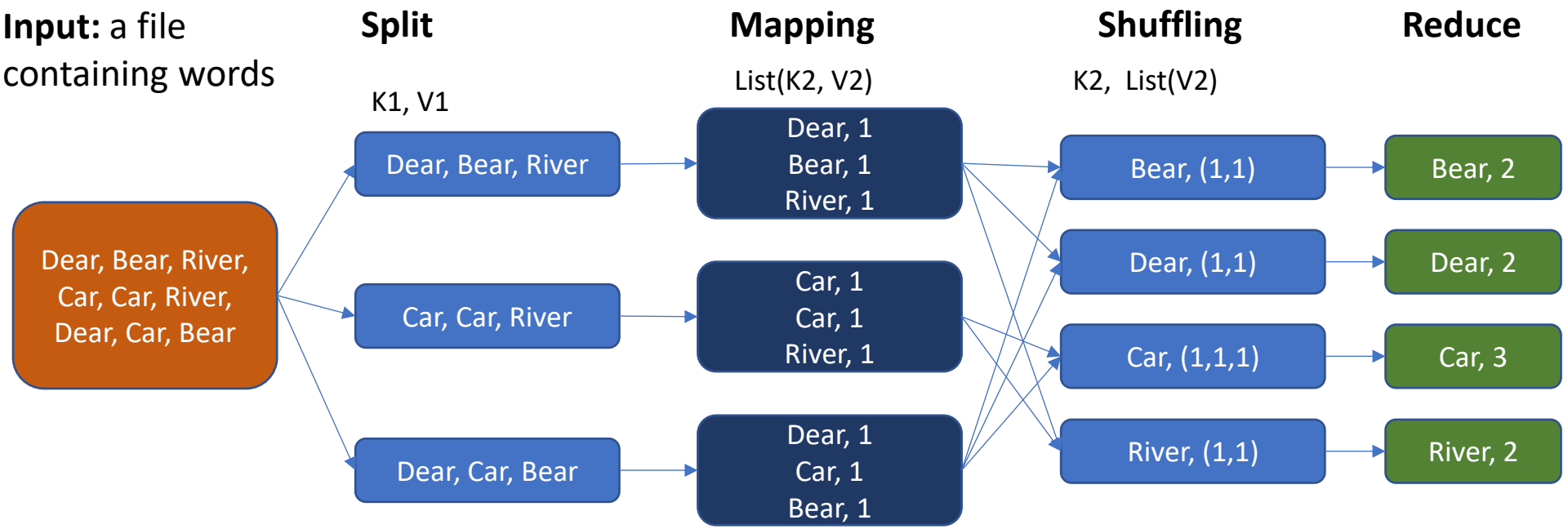


After the mapper phase, a partition process takes place where sorting and shuffling happen so that all the tuples with the same key are sent to the corresponding reducer.

Computing on the batch layer

MapReduce: a paradigm for Big Data computing

MapReduce how it works? Example: word count

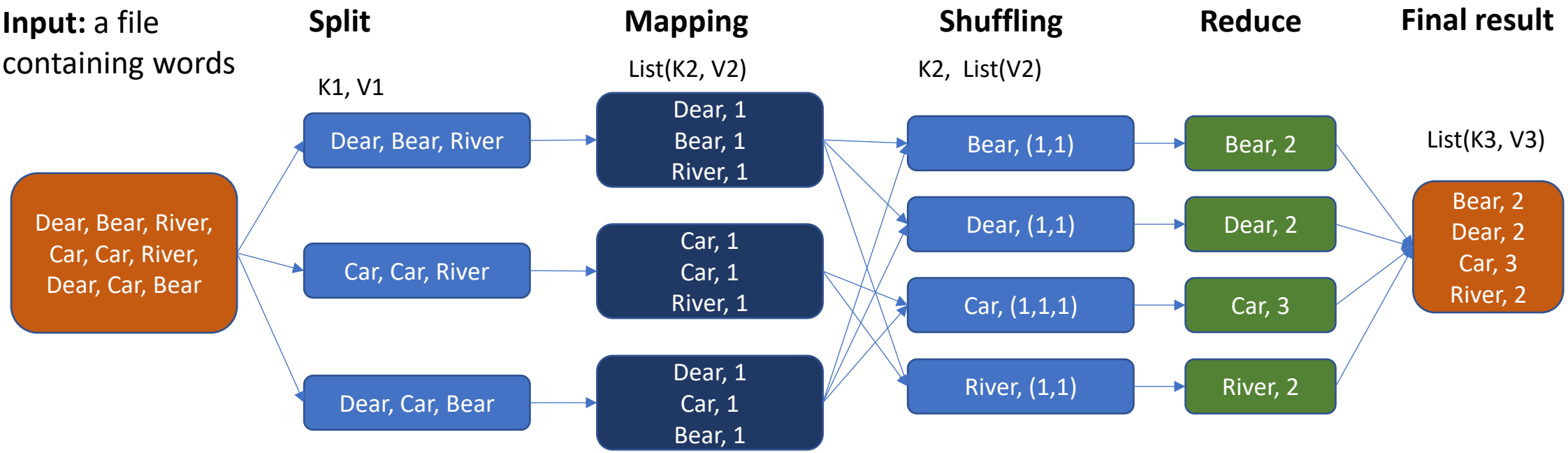


Each reducer counts the values which are present in that list of values.

Computing on the batch layer

MapReduce: a paradigm for Big Data computing

MapReduce how it works? Example: word count



Computing on the batch layer

MapReduce: a paradigm for Big Data computing

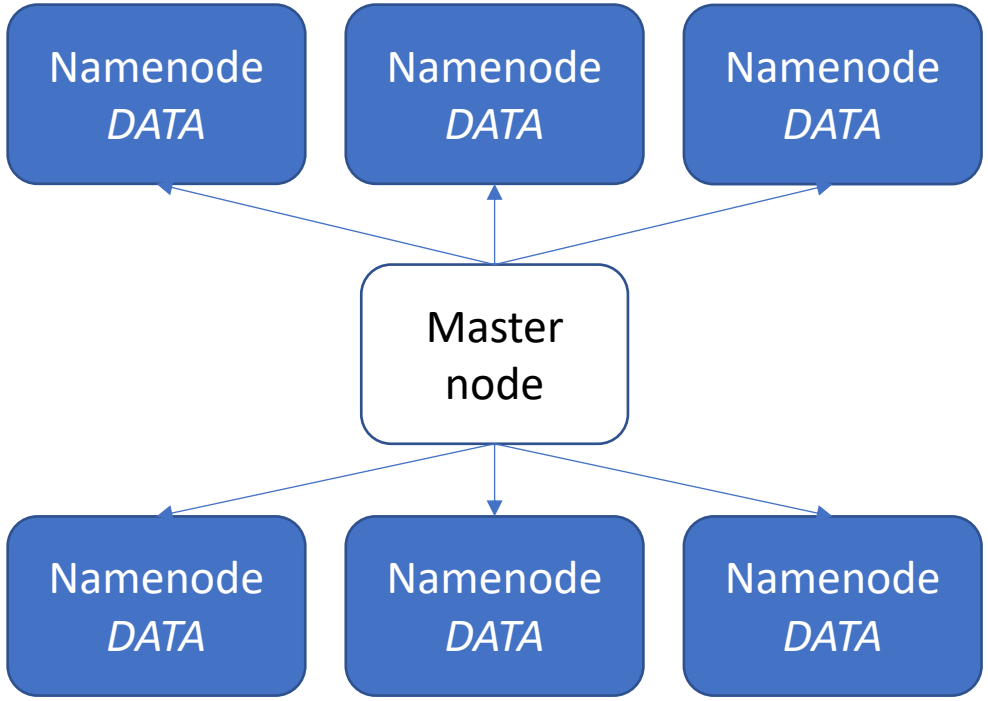
MapReduce: advantages

1. Parallel processing

- MapReduce is based on Divide and Conquer paradigm which allows data processing using different machines: a job is divided among multiple nodes and each node works with a part of the job simultaneously.
- As the data is processed by multiple machines instead of a single machine in parallel, the time taken to process the data gets reduced by a tremendous amount

2. Data locality

- Instead of moving data to the processing unit, we are moving the processing unit to the data in the MapReduce Framework.
- The data is distributed among multiple nodes where each node processes the part of the data residing on it.



Batch layer – take home

- Properties of the data
- The fact-based model for Big Data
- Storage models for Big Data
- Computing on the batch layer: *MapReduce* paradigm

Hadoop Ecosystem

Hadoop Ecosystem: Introduction

- Apache Hadoop is an implementation of Google's MapReduce framework
- A tool that can be used for the Batch Layer of the Lambda Architecture
- It is an open source project hosted by Apache Software Foundation
- It is implemented in Java with bindings for many other languages
- Commercial support and vendor specific additions are available
- Hadoop simplifies the development of data processing tasks and runs on commodity hardware (standardized servers that can be bought from any vendor)

In a nutshell, Hadoop provides a reliable, scalable platform for storage and analysis. Moreover, because it runs on commodity hardware and is open source, Hadoop is affordable.

Hadoop Ecosystem: a brief history

- Hadoop was created by Doug Cutting, the creator of Apache Lucene (the widely used text search library)
- Hadoop has its origins in Apache Nutch, an open web search engine (part of the Lucene project)

The Origin of the Name "Hadoop"

The name Hadoop is not an acronym; it's a made-up name. The project's creator, Doug Cutting, explains how the name came about:

> The name my kid gave a stuffed yellow elephant. Short, relatively easy to spell and pronounce, meaningless, and not used elsewhere: those are my naming criteria. Kids are good at generating such. Googol is a kid's term.

Projects in the Hadoop ecosystem also tend to have names that are unrelated to their function, often with an elephant or other animal theme ("Pig," for example). Smaller components are given more descriptive (and therefore more mundane) names. This is a good principle, as it means you can generally work out what something does from its name. For example, the namenode[8] manages the filesystem namespace.

Source: Hadoop: the Definitive Guide, by Tom White

Hadoop Ecosystem: a brief history

2002: the start of Nutch; a working crawler and search system quickly emerged

2003: a paper describing the architecture of Google's distributed filesystem (GFS) was published

2004: Nuthc's developers started to write an open source implementation, the Nutch Distributed Filesystem (NDFS)

2004: Google published the paper that introduced MapReduce[1]

2005: all major Nutch algorithms had been ported to run using MapReduce and NDFS

2006: an independent subproject of Lucene was started, called Hadoop

At around the same time, Doug Cutting joined Yahoo!, providing a dedicated team and the resources to turn Hadoop into a system that ran at web scale

2008: Yahoo! announced that its production search index was being generated by a 10,000-core Hadoop cluster

1 Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," December 2004.

Hadoop Ecosystem: a brief history

2008, January: Hadoop was made its own top-level project at Apache

Many other companies were using Hadoop during this time, such as Last.fm, Facebook, and the *New York Times*

2008, April: Hadoop broke a world record to become the fastest system to sort an entire terabyte of data on a 910-node cluster

Hadoop sorted 1 terabyte in 209 seconds (just under 3.5 minutes), beating the previous year's winner of 297 seconds

2008, November: Google reported that its MapReduce implementation sorted 1 terabyte in 68 seconds

2009, April: a team at Yahoo! has used Hadoop to sort 1 terabyte in 62 seconds

Since then, the trend has been to sort even larger volumes of data at ever faster rates

Hadoop Ecosystem: a brief history

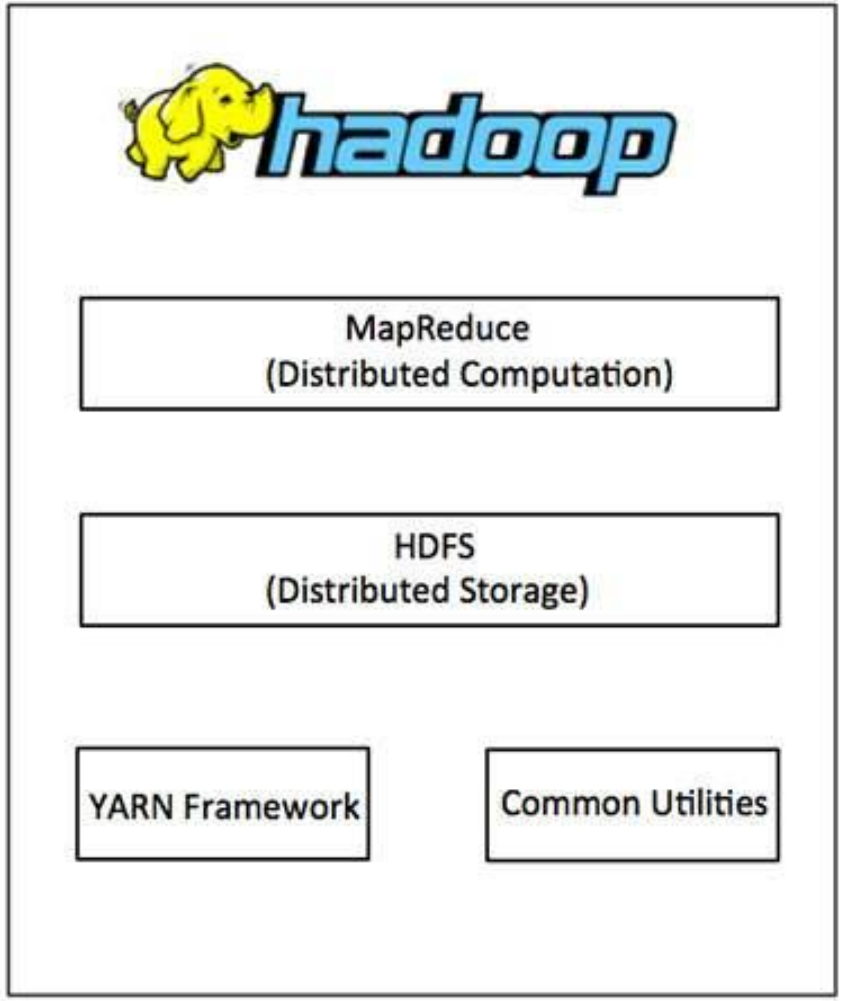
Today: Hadoop is widely used in mainstream enterprises
The industry recognized Hadoop's role as a general-purpose storage and analysis platform for Big Data

Commercial Hadoop support is available from large vendors, including EMC, IBM, Microsoft and Oracle

Commercially available Hadoop platforms:

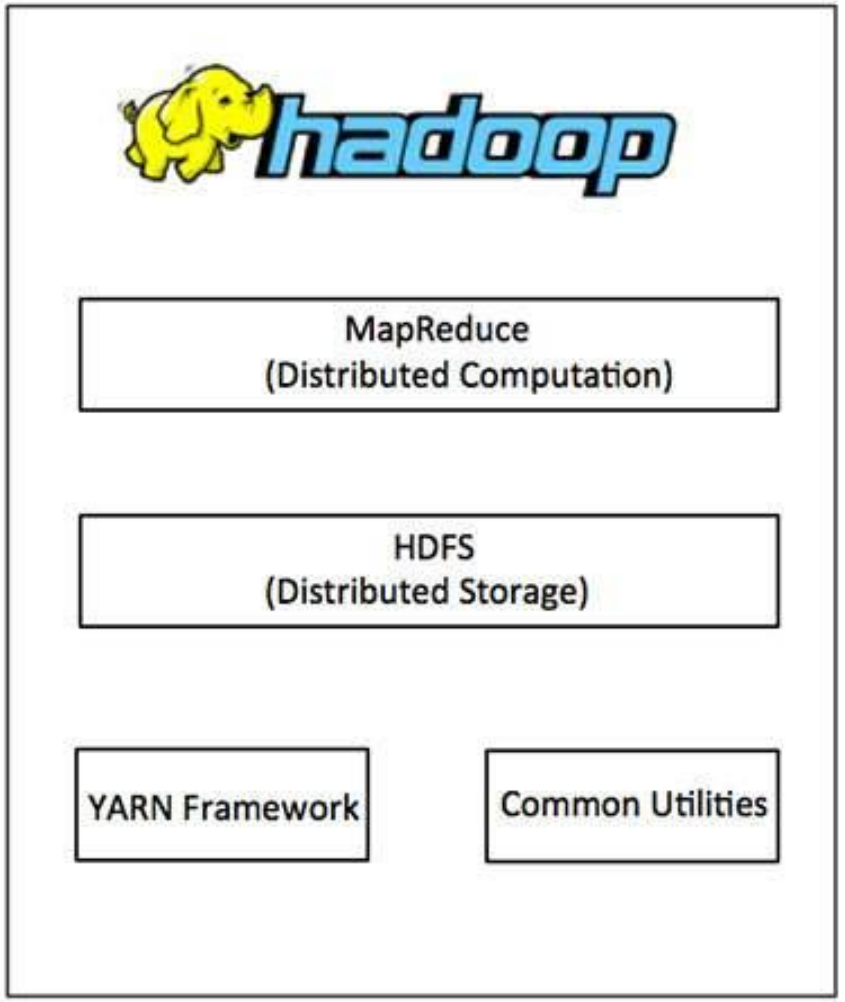
- Cloudera (previously also Hortonworks (merged with Cloudera)
- MapR (merged with HP))
- Amazon Web Services
- IBM
- Microsoft HDInsight
- Intel Distribution for Apache Hadoop
- …

Hadoop Ecosystem: Architecture



Hadoop Ecosystem: Architecture

MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

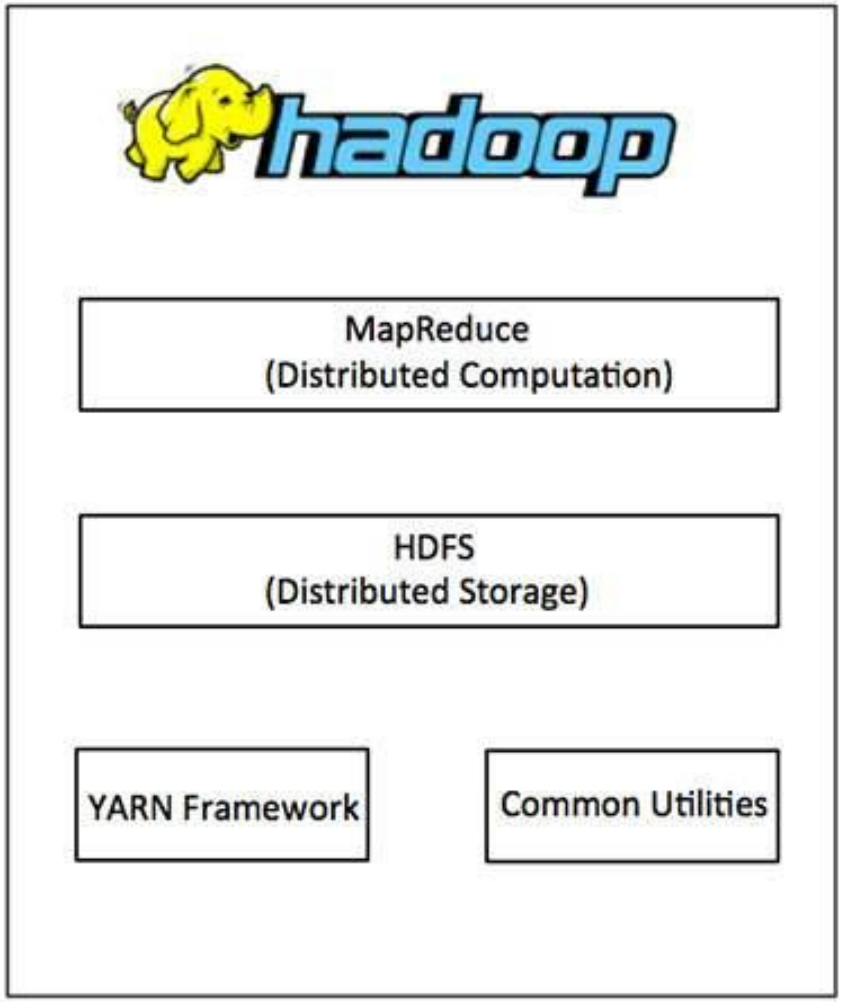


Hadoop Ecosystem: Architecture

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems.

Key features for HDFS:

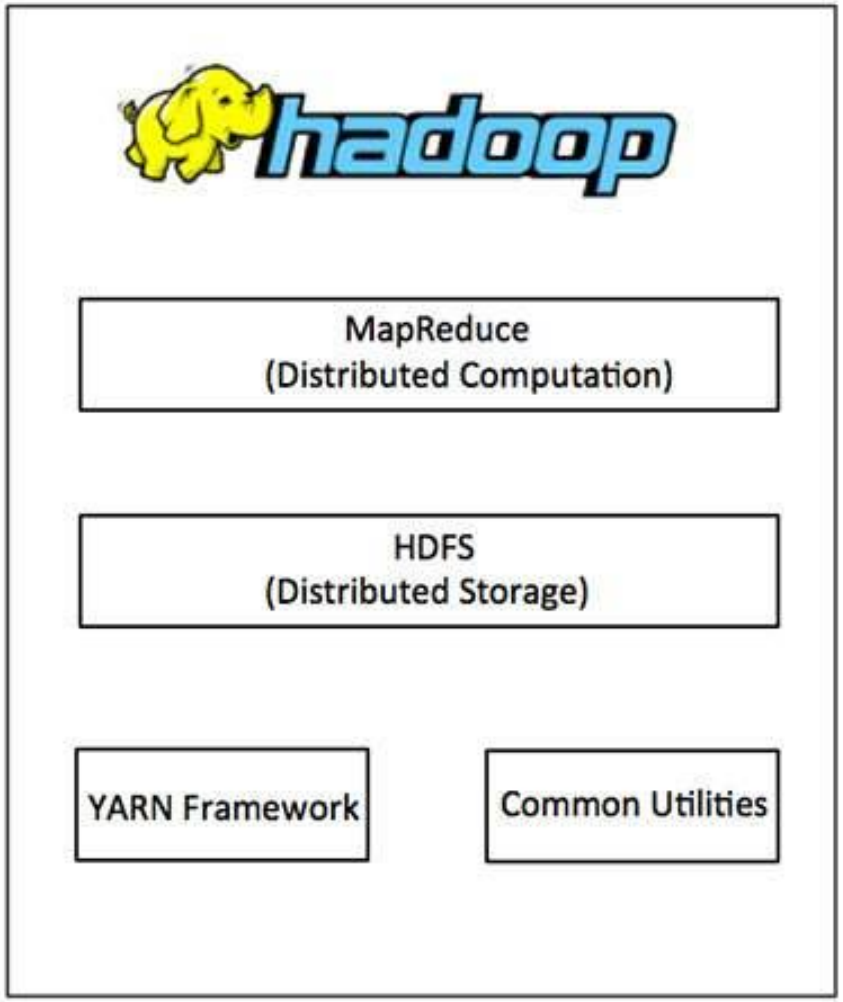
- It is highly fault-tolerant and is designed to be deployed on low-cost hardware.
- It provides high throughput access to application data and is suitable for applications having large datasets.



Hadoop Ecosystem: Architecture

Hadoop Common − Contains Java libraries and utilities required by other Hadoop modules.

Hadoop YARN − Is a framework for job scheduling and cluster resource management.



Next lecture

Hadoop Part 2

Lambda architecture: serving layer