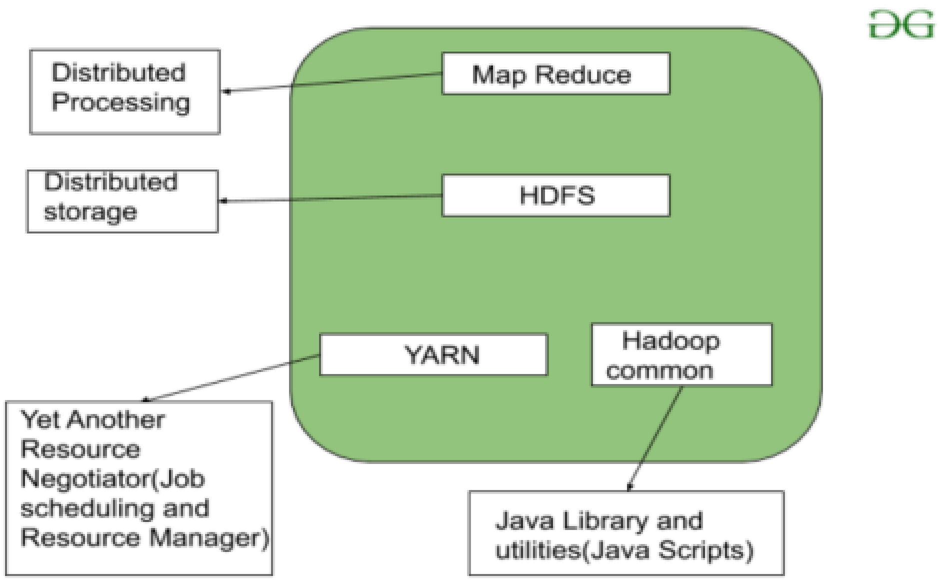
Introduction to Hadoop:

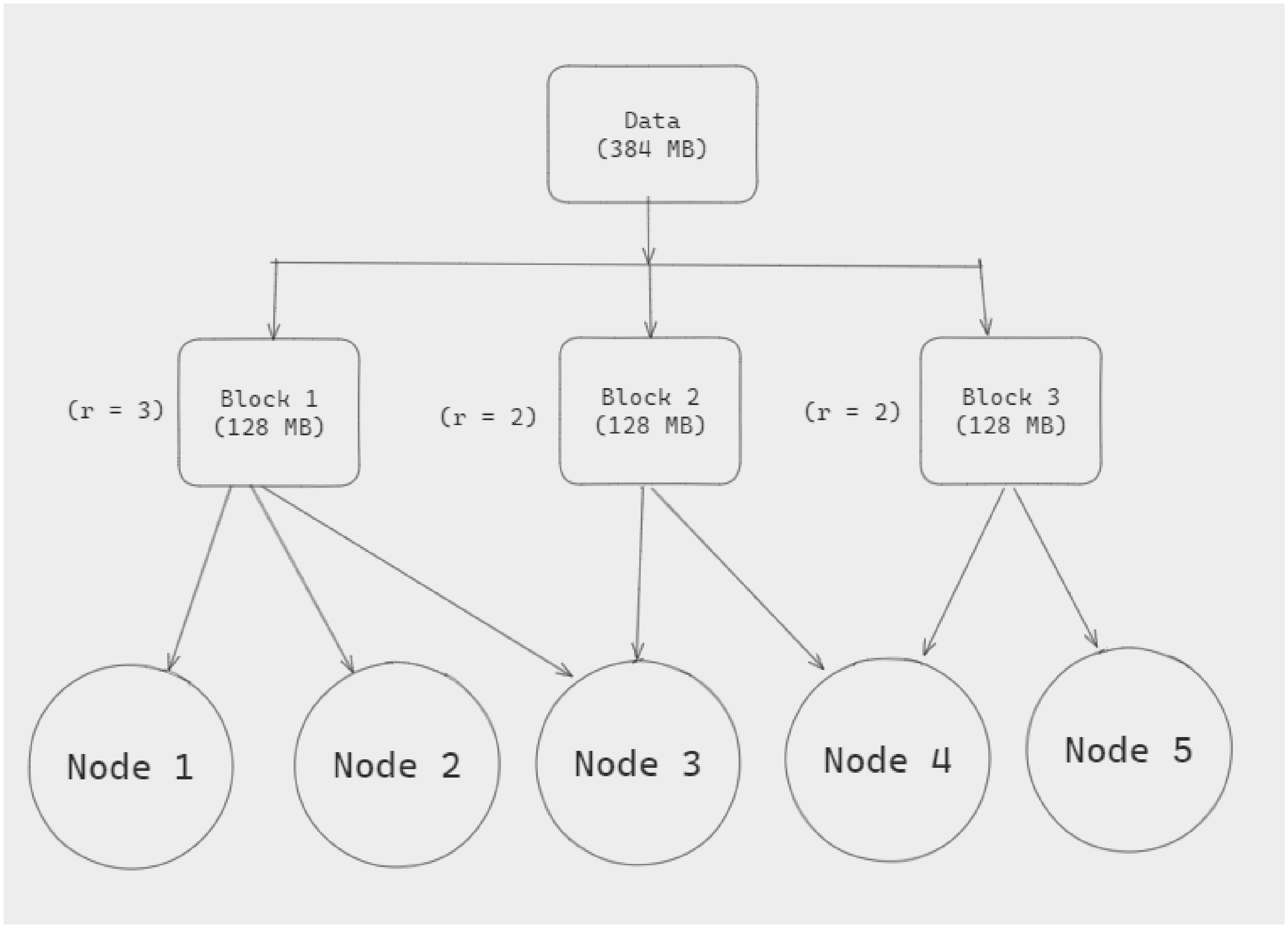
- Ever since camera devices came, the Data Generating Rate has been high and aggressively increasing over time.
- Big MNCs like Facebook, Youtube, Google etc, get TBs of data every minute generated on their Databases.
- Hence, they require an efficient, scalable and robust data-storing and managing architecture.
- Hence, in this case of Big Data, vertical scaling is not possible. (i.e, single machine/node used as a database management system)
- Vertical Scaling is highly NOT suitable for Big Data management because of few reasons:
    - If the only machine/node fails, the whole data will not be available and will be at data stake.
      (i,e., faults cannot be fixed and lost data cannot be recovered)
    - The physical space size for the machine (Database system) increases as data keeps on generating.
- Hence the scaling method that is suitable for Big Data management is Horizontal Scaling (distributed environment).
- In Horizontal scaling, multiple machines/nodes are used for storing the data.
- That is, if a machine/node's space/capacity is full, then another machine/node will be used to store the next upcoming data.
- And the Framework that supports this kind of distributed environment is Hadoop.
- Hadoop (or Apache Hadoop) is an open-source software framework that is used for storing and processing large amounts of data in a distributed computing environment.
- It is designed to handle big data and is based on the MapReduce programming model, which allows for the parallel processing of large datasets.
- It is commonly used in big data scenarios such as data warehousing, business intelligence, and machine learning.
- It's also used for data processing, data analysis, and data mining.
- It enables the distributed processing of large data sets across clusters of computers using a simple programming model.
- It has Three main components:


Distributed Processing
Map Reduce
Distributed storage
HDFS
YARN
Hadoop common
Yet Another Resource Negotiator(Job scheduling and Resource Manager)
Java Library and utilities(Java Scripts)

1. HDFS (Hadoop Distributed File System):

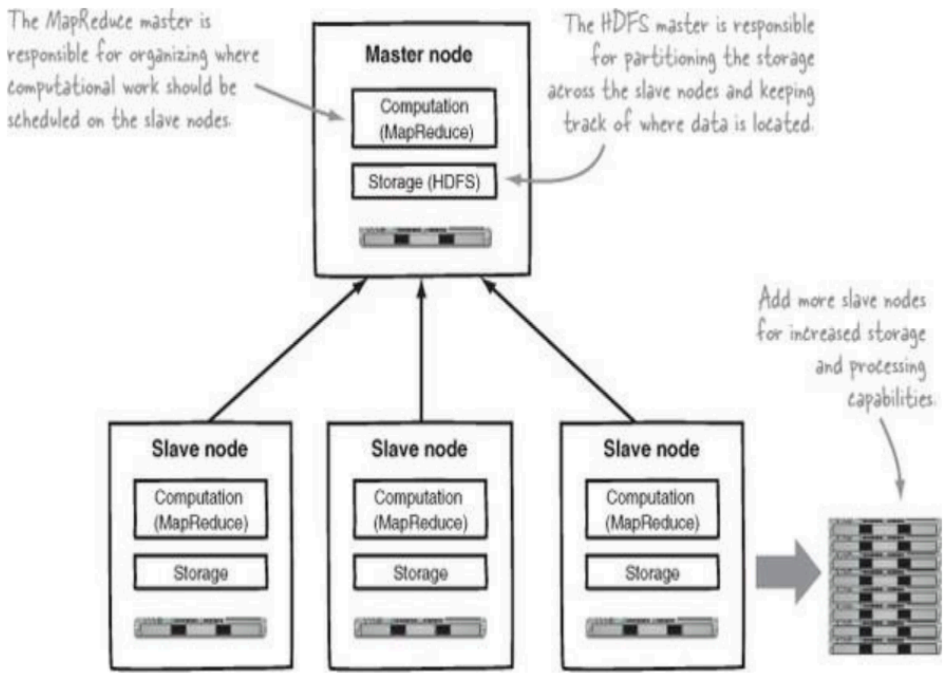
- It is the primary storage system used by Hadoop.
- It's designed to store large datasets reliably and efficiently across a cluster of commodity hardware(machines which are always running are available for renting to get compute services).

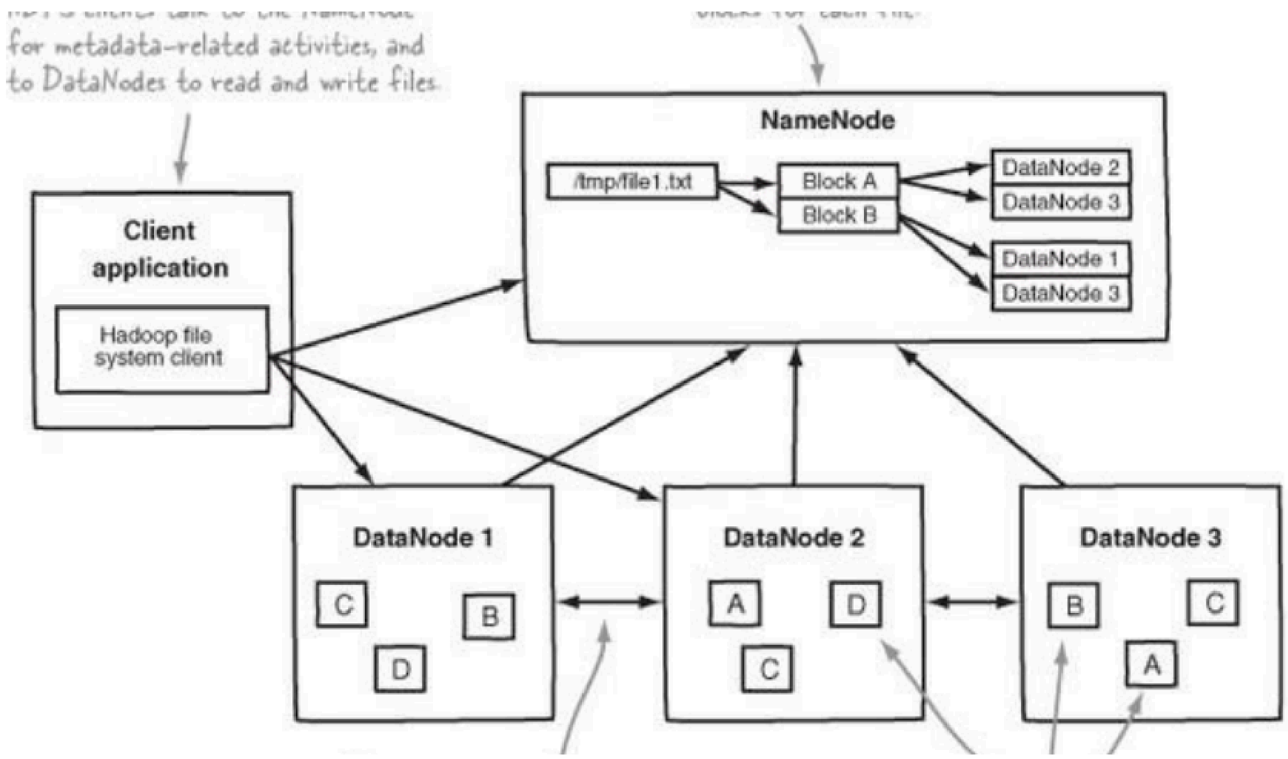
- HDFS distributes data across multiple nodes in a cluster, breaking large files into smaller blocks (typically 128 MB or 256 MB in size) and replicating each block across multiple nodes for fault tolerance.
- The replication factor is mostly 2 (or) 3. That is, when a data is divided into multiple fixed sized blocks, then each block is not only stored at one node, but stored in 2 (or) 3 other nodes to use as backup, when the data block at the main node is corrupted.
- By distributing data across multiple machines, HDFS provides high throughput and scalability for storing large datasets.



- The HDFS ecosystem contains two types of nodes:

Figure High-level Hadoop architecture


The MapReduce master is responsible for organizing where computational work should be scheduled on the slave nodes.
Master node
Computation (MapReduce)
Storage (HDFS)
The HDFS master is responsible for partitioning the storage across the slave nodes and keeping track of where data is located.
Add more slave nodes for increased storage and processing capabilities.
Slave node
Computation (MapReduce)
Storage



a. Master Node (Name Node):

- **Cluster Management:** It is the single and main node for a cluster, which manages all Hadoop related activities.
- **Jobs Scheduling, Resource Management & Data Distribution:** It manages/controls the hadoop cluster which includes activities like, Job Scheduling & Resource Management (YARN), Dividing the data into multiple data nodes and distributing across 2 or 3 nodes, and processing the data fetched from the nodes (MapReduce).
- **Metadata Management:** And it has a core part which manages the information about the metadata and namespaces for the file system, such as directory structure(location of

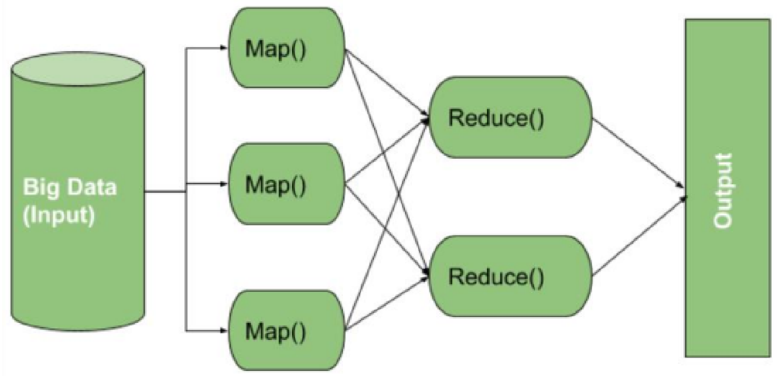
data blocks) in a node, file permissions and mapping of data-blocks with data-nodes, known as Name-Node.

b. Slave Node (Data Node/Worker Node):
   - **Task Execution:** These are multiple machines/nodes of a cluster (all other than master node) connected with master node and with each other and perform the actual data processing tasks assigned/delegated by the master node.
   - **Data Storage**: Store the actual data blocks that make up the files in HDFS.
   - **Contains YARN & MapReduce:** Just like in Master Node, these nodes will too contain the YARN and MapReduce related tasks.
   - **Heartbeat and Block Reports**: Regularly send heartbeat signals to the Master Node to confirm their operational status and block reports to provide the status of the data blocks stored on them.

2. YARN (Yet Another Resource Negotiator):

- It is the resource management and Job(task) Scheduling component of Hadoop
- **Function**: YARN manages the allocation of resources such as CPU, memory, and disk space across the Hadoop cluster.
- **ResourceManager**: Acts as the master and manages resources. It decides how to assign resources to applications.
- **NodeManager**: Runs on each node and is responsible for monitoring resource usage and reporting it to the ResourceManager. It manages containers on the node, which are allocated resources for executing tasks.
- **Linked with Operating System:** YARN is often likened to an operating system for the Hadoop ecosystem. It abstracts the resource management and job scheduling functions, enabling efficient data processing and scalability.

3. Map-Reduce:



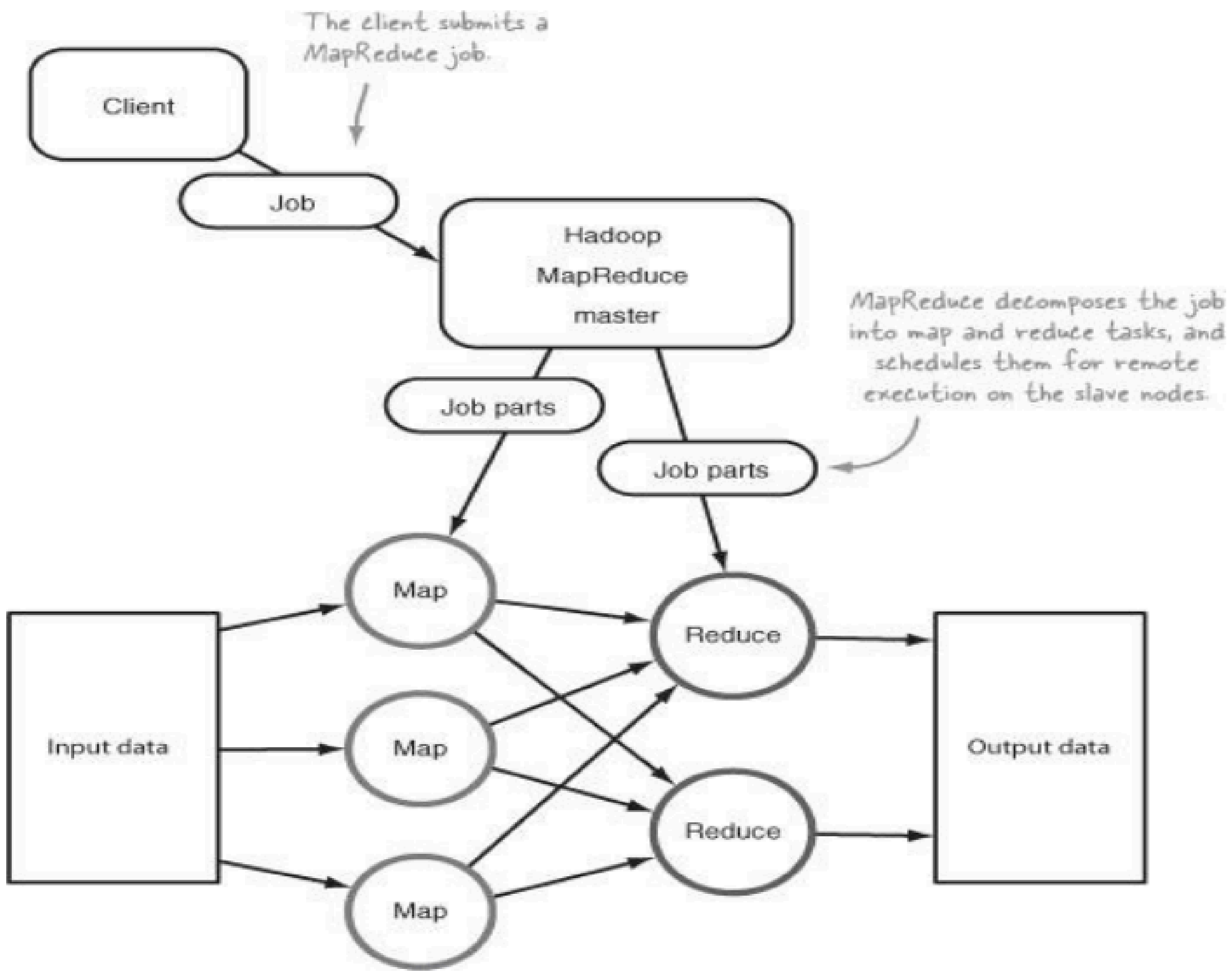
- Since when the new data arrives at the master node, it divides it into multiple parts and stores each part in individual slave nodes.
- This stored data is unprocessed data. To make this data suitable for performing data analytics tasks, it must be processed first. Hence, MapReduce is like a preprocessing task for data analytics.
- For processing the same data, the same parts must be fetched from the same slave nodes and then processed. This can be done by MapReduce.
- Hence, MapReduce retrieves different parts of the same data from the slave nodes and reduces them to have a reduced or aggregated version of the original data.
- The reduced output data represents summaries, counts, averages, or other aggregated information derived from the original dataset, which are used in data analytics.
- MapReduce is a component with two functions, viz:

- Map Function: It processes the original data and produces intermediate key-value pairs.
- Reduce Function: It aggregates or summarizes the intermediate key-value pairs to produce the final output.

- After the Map phase, another phase is performed to ensure the correct order of the generated key-value pairs, known as Shuffle and Sort Phase.

- **Shuffle and Sort Function (performed after map function and before reduce function):** The intermediate key-value pairs are shuffled and sorted based on the key.

<u>**Series of Events:**</u>


The client submits a MapReduce job.
Client
Job
Hadoop MapReduce master
MapReduce decomposes the job into map and reduce tasks, and schedules them for remote execution on the slave nodes.
Job parts
Job parts
Map
Map
Map
Reduce
Reduce
Input data
Output data

Data Storage in HDFS:

- **Initial Data Storage**: Data is initially loaded into HDFS.
- **Data Replication**: HDFS splits the data into blocks (default size 128 MB) and replicates these blocks across multiple DataNodes for fault tolerance.

Job Submission:

- A user submits a MapReduce job to the Hadoop cluster, specifying the input data (already stored in HDFS), the Map function, the Reduce function, and the output path (where the results will be stored in HDFS).

Map Phase:

- **Data Retrieval**: Map tasks are executed on DataNodes where the data blocks are stored (data locality principle). The Map function processes the input data, generating intermediate key-value pairs.
- **Intermediate Data**: The intermediate key-value pairs are stored temporarily on local disk space of the DataNodes.

Shuffle and Sort Phase:

- **Data Transfer**: Intermediate data generated by Map tasks is shuffled (transferred across the network) to the nodes where Reduce tasks will be executed. This data is sorted based on the key.
- **Sorting**: This phase ensures that all values associated with a particular key are grouped together for the Reduce phase.

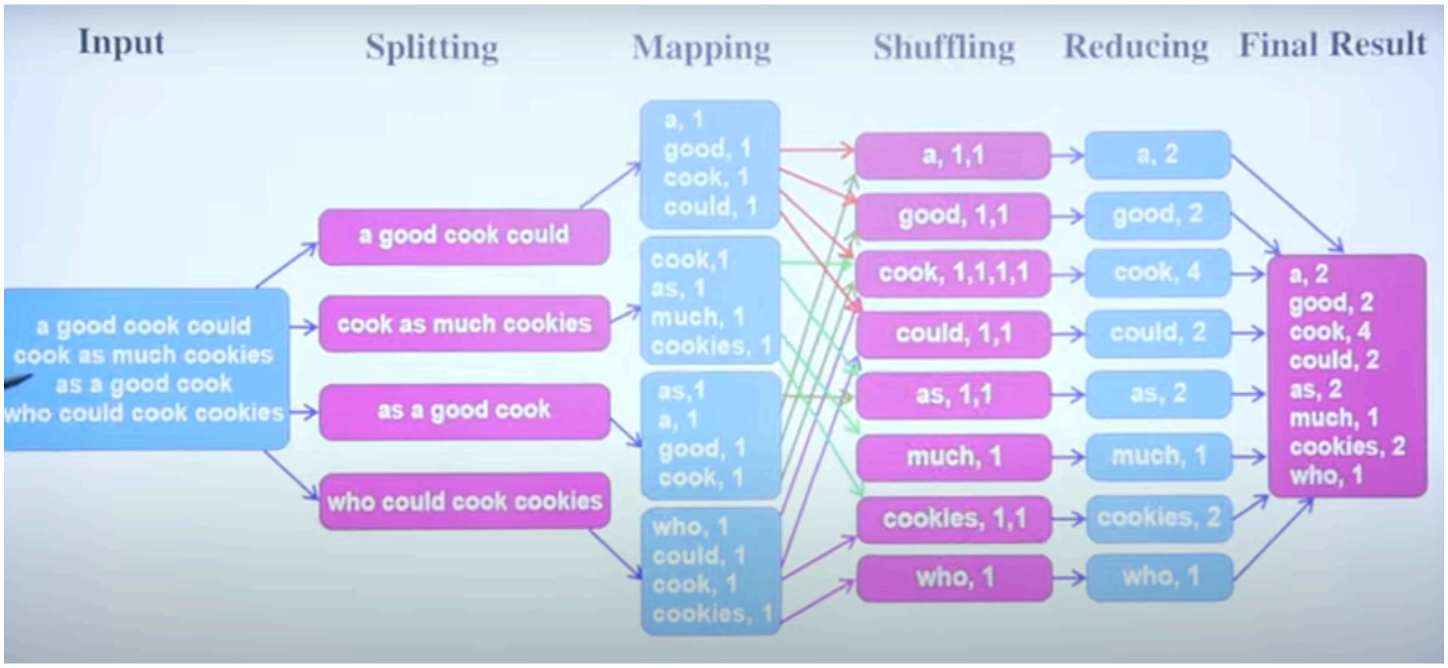
Reduce Phase:

- **Data Aggregation**: Reduce tasks process the intermediate key-value pairs, aggregating or summarizing the data based on the key.
- **Final Output**: The Reduce function writes the final output to HDFS at the specified output path.

Job Completion:

- The job is marked as complete once all Map and Reduce tasks are finished. The final output is now stored in HDFS and can be accessed or further processed as needed.

Ex:


Input
Splitting
Mapping
Shuffling
Reducing
Final Result
a good cook could cook as much cookies as a good cook who could cook cookies
a good cook could
cook as much cookies
as a good cook
who could cook cookies
a, 1 good, 1 cook, 1 could, 1
cook,1 as, 1 much, 1 cookies, 1
as,1 a, 1 good, 1 cook, 1
who, 1 could, 1 cook, 1 cookies, 1
a, 1,1
good, 1,1
cook, 1,1,1,1
could, 1,1
as, 1,1
much, 1
cookies, 1,1
who, 1
a, 2
good, 2
cook, 4
could, 2
as, 2
much, 1
cookies, 2
who, 1
a, 2 good, 2 cook, 4 could, 2 as, 2 much, 1 cookies, 2 who, 1

__Data serialization__

- Data serialization in big data refers to the process of converting data structures or objects into a format that can be easily stored, transmitted, and reconstructed later (such as XML, JSON, etc).

- This is crucial in big data environments where data needs to be efficiently processed, transferred between systems, and stored.

- Serialization ensures that data can be saved in a compact and consistent manner, making it easier to handle large volumes of data.

- Here are some key points about data serialization in big data:

1. **Compact Storage**: Serialized data is typically more compact than its original form, which helps save storage space and improves performance in storage systems.
2. **Efficient Transmission**: Serialized data can be easily transmitted over networks, enabling efficient data exchange between different systems or components in a distributed environment.
3. **Interoperability**: Serialization allows different systems to understand and process data consistently. It enables interoperability between different programming languages, platforms, and frameworks.
4. **Data Formats**: Common serialization formats used in big data include JSON, XML, Protocol Buffers (protobuf), Apache Avro, Apache Thrift, and MessagePack. Each format has its own advantages in terms of speed, efficiency, and compatibility.
5. **Deserialization**: This is the reverse process of serialization. It involves converting the serialized data back into its original structure or object. Deserialization ensures that data can be accurately reconstructed and used by the application.
6. **Use Cases**:
    - **Data Storage**: Serialized data can be stored in databases, file systems, or distributed storage systems like Hadoop HDFS.
    - **Data Transfer**: Serialized data can be transferred between different components of a big data pipeline, such as from data ingestion systems to processing engines.
    - **Data Processing**: Big data processing frameworks like Apache Spark and Apache Flink often use serialization to manage data efficiently during processing.

Example Scenarios

- **Apache Kafka**: Kafka uses serialization to convert messages into byte streams for efficient storage and transmission.
- **Apache Hadoop**: Serialization is used to store data in HDFS and for transferring data between MapReduce tasks.
- **REST APIs**: Data exchanged through REST APIs is often serialized in JSON or XML format.

Serialization is a fundamental concept in big data because it ensures data can be managed efficiently and reliably throughout the data lifecycle.