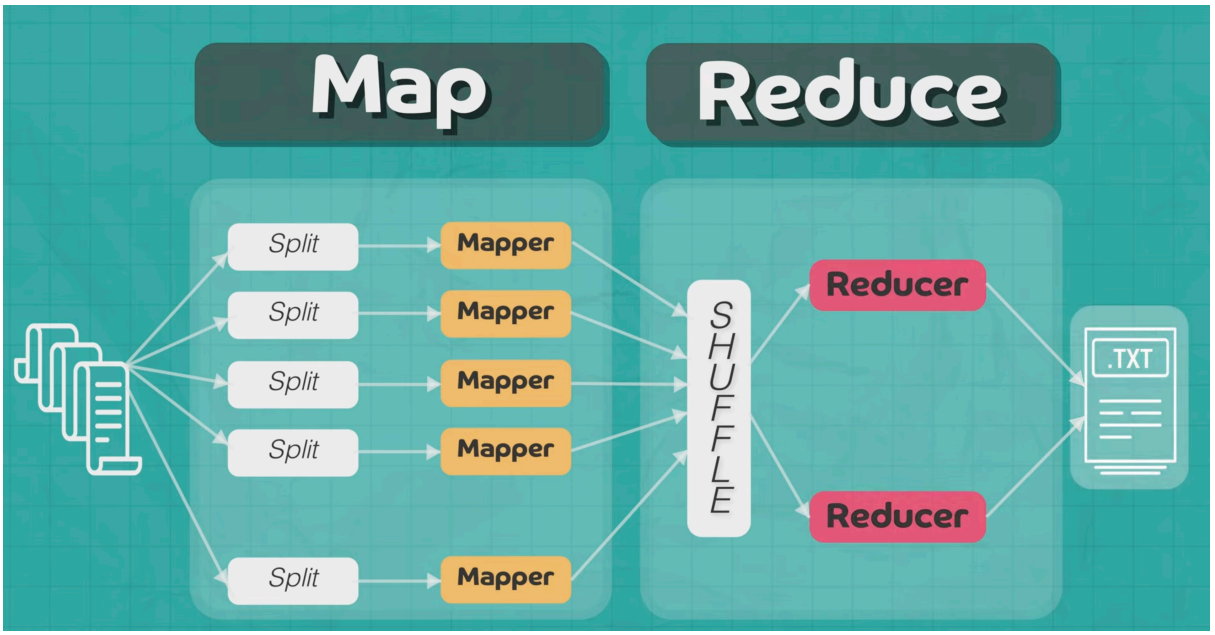
02 Introduction to Map-Reduce

🔷 What is MapReduce?

MapReduce is Hadoop's original processing framework for handling large datasets in a distributed manner. It breaks jobs into smaller tasks and executes them in parallel across multiple nodes. The goal is to process data where it lives on the Data-Nodes minimizing the need to transfer large datasets across the network.

🔷 How MapReduce Works

MapReduce works in **two main phases**: **Map** and **Reduce**, with a **Shuffle/Sort** step in between.



1) Map Phase

- The input dataset is split into chunks.
- Each chunk is processed independently by a **Mapper task**.
- Mapper converts input into **key-value pairs**.

2) Shuffle and Sort Phase

- The system groups values with the same key.
- All similar key values go to one reducer.
- This step involves heavy **network and disk I/O**.

3) Reduce Phase

- Reducer aggregates grouped data (like sum, count).
- Produces final output.

⚙️ Step-by-Step Breakdown of the MapReduce Data Flow

Goal: Let say we have orders data and we want total sales for each category.

order_id	category	product_name	quantity	price	order_date	country
1001	Electronics	Mobile	1	20000	2025-10-10	India
1002	Fashion	T-Shirt	2	800	2025-10-10	India
1003	Electronics	Laptop	1	60000	2025-10-11	USA
1004	Grocery	Rice	5	100	2025-10-11	India
1005	Fashion	Jeans	1	1200	2025-10-12	UK
1006	Electronics	Headphones	3	1500	2025-10-12	USA
1007	Grocery	Sugar	2	50	2025-10-12	India
1008	Electronics	Laptop	1	65000	2025-10-13	India
1009	Fashion	Shoes	1	2500	2025-10-13	UK
1010	Grocery	Milk	5	60	2025-10-13	India

1️⃣ Map Phase (Local Processing)

Each mapper:

- Reads its **local data block** from HDFS
- Emits intermediate key–value pairs like

  `("Electronics", 200)`, `("Fashion", 500)`, `("Electronics", 100)`

These intermediate results are stored **locally on the same machine** in temporary files.

No data is sent across the network yet.

Everything here is **local**.

map java

2️⃣ Shuffle Phase (Network Movement)

Once all mappers finish:

- The Hadoop framework figures out which **reducer** is responsible for which key.
- Example:
  - Reducer 1 → Keys starting with A–M
  - Reducer 2 → Keys starting with N–Z

Then:

- Each mapper **sends** (over the network) the relevant key-value pairs to the appropriate reducer.

So:

Mapper	Emits	Sent to Reducer
Mapper 1	(Electronics, 200)	Reducer 1
Mapper 2	(Electronics, 400)	Reducer 1
Mapper 3	(Fashion, 100)	Reducer 2

✅ This cross-node movement is the **shuffle** — and it's usually the **most expensive phase** (because it involves heavy network I/O).

3️⃣ Sort Phase (At the Reducer Node)

When data arrives at each reducer node:

- Hadoop automatically **sorts all the keys** so that the reducer function receives keys in sorted order.
- It also **groups all values for the same key** together.

Example (Reducer 1):

```
Input before sort:
("Electronics", 400)
("Fashion", 100)
("Electronics", 200)
```

After Hadoop's internal sort:

```
Electronics → [200, 400]
Fashion → [100]
```

✅ Sorting happens **on the reducer node**, but during the shuffle transfer Hadoop also starts sorting the intermediate segments as they are fetched.

So you can say:

> Sort starts during shuffle (partial merge on mapper output)
>
> and completes before reducer input (final merge sort on reducer side)

4️⃣ Reduce Phase (Final Aggregation)

Now, the reducer function runs once per unique key.

Example reducer logic:

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class CategorySalesReducer
      extends Reducer<Text, IntWritable, Text, IntWritable> {

   private IntWritable result = new IntWritable();

   @Override
   protected void reduce(Text key, Iterable<IntWritable> values, Context context)
         throws IOException, InterruptedException {

      int sum = 0;
      for (IntWritable v : values) {
         sum += v.get();
      }
```

```
        result.set(sum);
        context.write(key, result);
    }
}
```

For our data:

```
Electronics → [200, 400] → sum = 600
Fashion → [100] → sum = 100
```

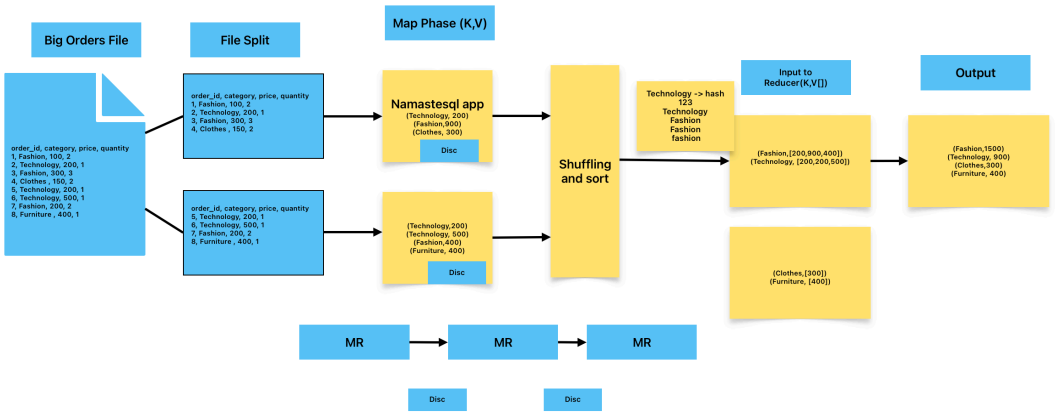
✅ Aggregation happens **inside the reducer** after sorting and grouping.

5️⃣ Output Phase

Reducers write their final aggregated results **back to HDFS**.

Each reducer writes one output file (e.g. `part-r-00001`, `part-r-00002`, etc.), all of which together form the final output dataset.

Board:



→ Challenges with MapReduce:

1- Performance issues : Lot of disk IOs

2- Hard to write the code

3- MapReduce supports only batch processing

4- lots of things to learn : pig,hive,hbase, sqoop etc

5- no interactive mode ..have to bundle full code and run it on cluster and output files will be generated

package/bundle → Hadoop cl

map program → reduce

ETL → extract , tranform load

ingesting data to HDFS → MP→ write o/p

hdfs / s3/ mysql/azure storage → spark → write back

hdfs / datalake