How MapReduce works on data stored in HDFS?

Data is Stored in HDFS

- When you upload a file to HDFS (e.g., input.txt), it's automatically split into blocks (usually 128MB or 64MB).
- These blocks are distributed across nodes in the Hadoop cluster.

Job Submission

- A MapReduce job (e.g., a Word Count program) is submitted to the Hadoop system.
- The **JobTracker** (or ResourceManager in **YARN**) coordinates the job execution.

Input Splits & Task Assignment

- The JobTracker divides input data (HDFS blocks) into InputSplits.
- For each split, a Map Task is assigned—ideally on the node where the data block resides (called data locality).

Map Phase

- Each Map task reads its assigned block from HDFS, processes it line by line.
- Mapper emits key-value pairs. For example:

Input: "cat dog cat"

Output: ("cat", 1), ("dog", 1), ("cat", 1)

Shuffle and Sort

- All mapper outputs are shuffled and sorted by key.

For example:

("cat", 1), ("cat", 1), ("dog", 1)

becomes:

("cat", [1, 1]), ("dog", [1])

Reduce Phase

- Each key and its list of values is sent to a Reduce Task.
- Reducer processes them and emits final results:

Input: ("cat", [1, 1])

Output: ("cat", 2)

Output Written to HDFS

- The final reducer output is written back to HDFS, typically in a directory like /output.
- Result file(s): part-00000, part-00001, etc.

Example Flow

```
hadoop fs -put input.txt /input
hadoop jar WordCount.jar WordCount /input /output
```

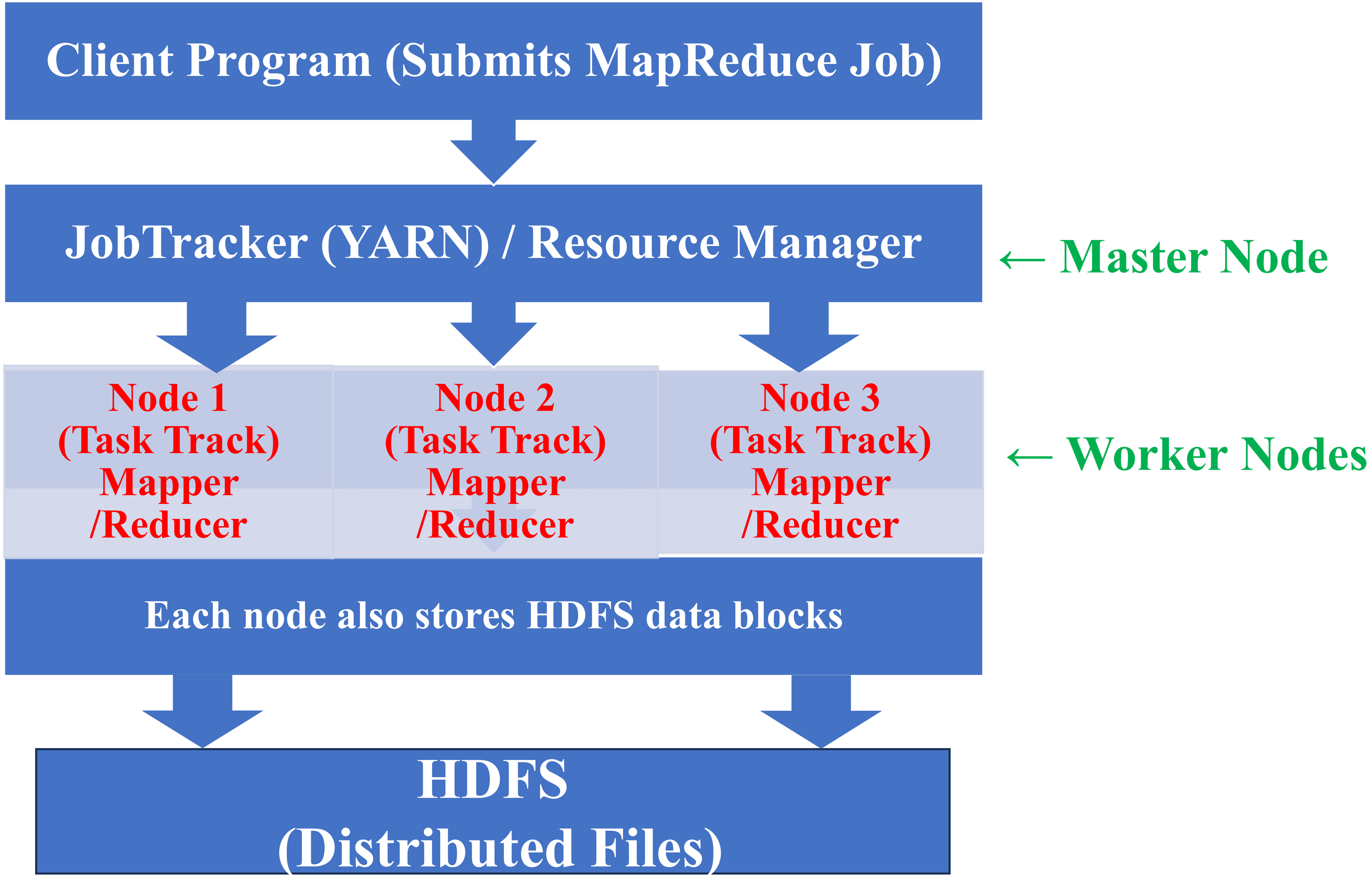
Behind the scenes:

- Input file split → Mapper runs on each block
- Map emits words and 1s → Shuffle groups by word
- Reducer adds counts → Output written to /output/part-00000

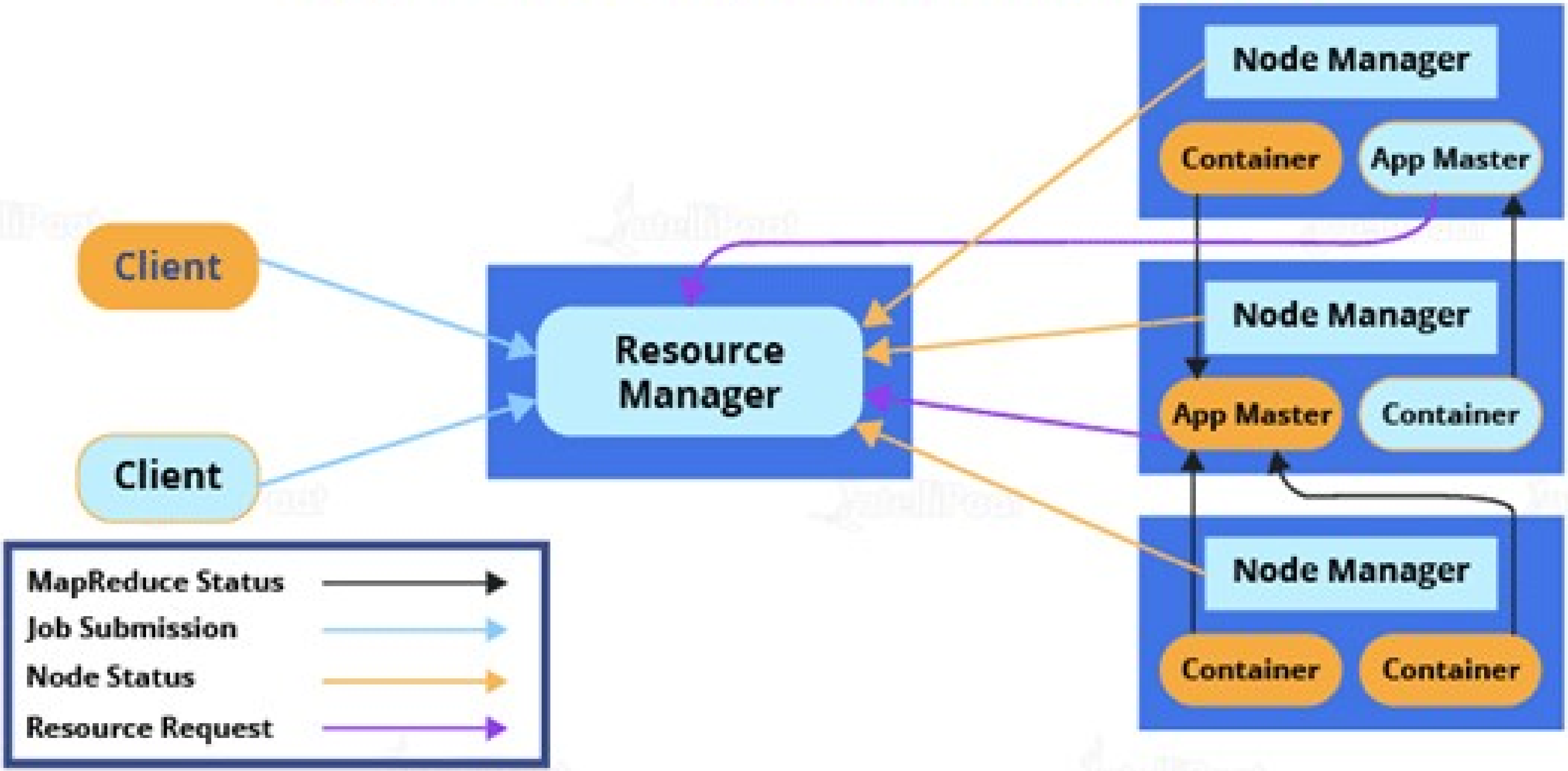
Overall Process

Phase	**Purpose**	**Works On**
Input	Split HDFS blocks	HDFS
Mapper	Process input → emit (k,v)	Node-local data
Shuffle/Sort	Group by key	Hadoop network
Reducer	Aggregate values	Sorted data
Output	Save results	HDFS

MapReduce Architecture (with HDFS)



Apache Hadooop YARN: Architecture



Explanation of Architecture Components

Client

- Submits the job to Hadoop (e.g., a Java .jar or a streaming job with Python scripts).

JobTracker / ResourceManager (Master Node)

- Divides the job into smaller tasks.
- Assigns Map and Reduce tasks to available **TaskTrackers (or NodeManagers in YARN**).
- Tracks progress, handles failure, and aggregates final results.

TaskTracker / NodeManager (Worker Nodes)

- Each worker node runs:
  - **Mapper tasks**: Process input data from HDFS, emit intermediate key-value pairs.
  - **Reducer tasks**: Aggregate intermediate data and produce final output.

HDFS (Hadoop Distributed File System)

- **Input** files are stored and split into blocks across nodes.
- **Output** of the MapReduce job is also saved back into HDFS.

Full Workflow: How MapReduce Works on HDFS

1. Upload to HDFS

```
hadoop fs -put input.txt /input
```

- File is split and distributed across cluster nodes.

2. Job Submission

```
hadoop jar WordCount.jar WordCount /input /output
```

- The client sends job details to the JobTracker.

3. Input Splits

- JobTracker reads metadata from HDFS.
- Divides input into splits (usually matching block size: 128MB).
- Assigns each split to a node containing the block (data locality optimization).

4. Map Phase

- Each mapper reads its split from HDFS.
- Emits key-value pairs like: ("hadoop", 1), ("mapreduce", 1)

Full Workflow: How MapReduce Works on HDFS

5. Shuffle and Sort

- Hadoop groups all values by key across the cluster.
- Example: All "hadoop" keys from all mappers go to the same reducer.
- Sorted intermediate data is sent to reducers.

6. Reduce Phase

- Reducer aggregates values:

("hadoop", [1, 1, 1]) → ("hadoop", 3)

- Writes final output to HDFS.

7. Output Storage

hadoop fs -cat /output/part-00000

(reducer number (in 5-digit format), e.g., part-00000 → output from reducer 0, part-00001 → output from reducer 1)

- Results stored in /output directory in HDFS.

Advantage of this Architecture

Feature	Why It's Useful
Data Locality	Move compute to data, not data to compute
Scalability	Add more nodes → handle more data
Fault Tolerance	Auto-retry failed tasks
Parallelism	Multiple mappers/reducers run in parallel
Batch Processing	Handles large-scale data jobs efficiently

HDFS – MapReduce - YARN

- **HDFS** stores the data while **YARN** coordinates the resources needed to process that data through containers.

- **MapReduce** jobs are executed on the resources allocated by YARN and make use of the **HDFS** for storing input and output data.

- **YARN** handles **resource management** (scheduling, allocation, and monitoring of tasks across the cluster).