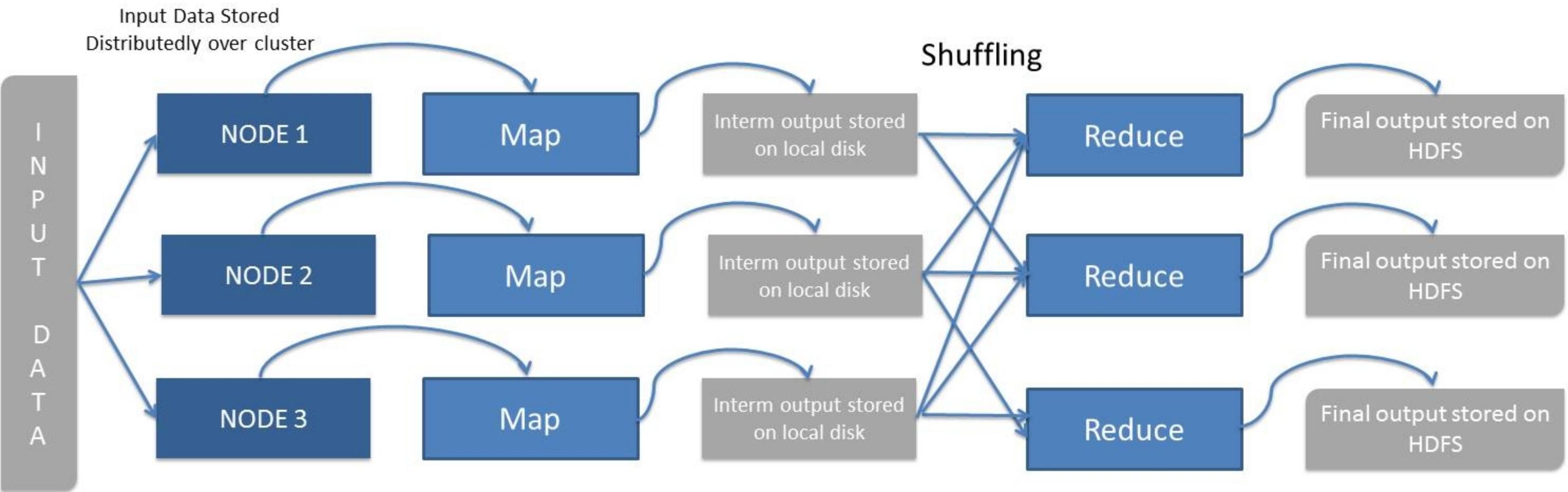


Advanced MapReduce Concepts

- ✓ Anatomy of MapReduce
- ✓ Hadoop data types
- ✓ Map Reduce – A closer look
- ✓ Map Reduce Components
- ✓ Map Reduce in a Nut-Shell

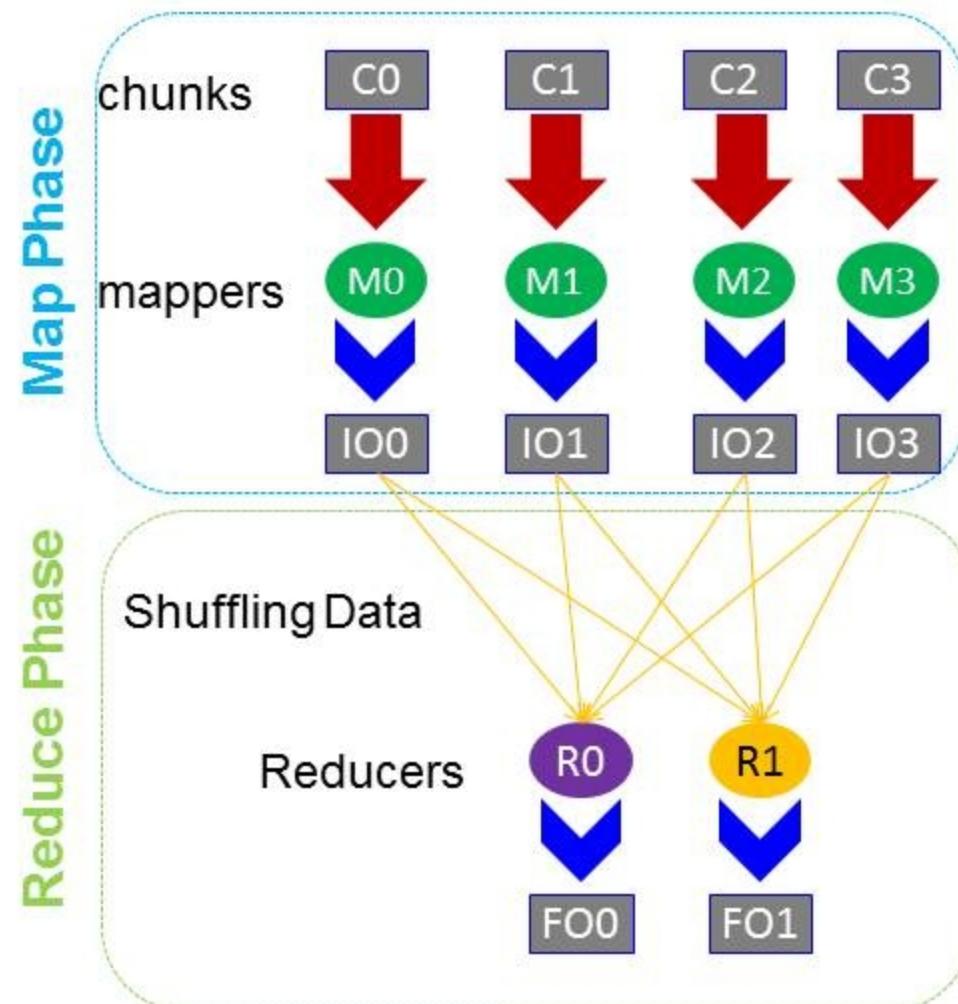


Anatomy of MapReduce



Anatomy of MapReduce

- In MapReduce, chunks are processed in isolation by tasks called *Mappers*
- The outputs from the mappers are denoted as *intermediate outputs* (IOs)
- The process of bringing together IOs into a set of Reducers is known as *Shuffling and Sort process*
- Intermediate outputs will be given as input into second set of tasks called *Reducers*
- The Reducers produce the *final outputs* (FOs)
- Overall, *MapReduce* breaks the data flow into two phases, map phase and reduce phase



- MR has a defined way of keys and values types for it to move across cluster:
 - Values → Writable
 - Keys → WritableComparable<T>
 - WritableComparable = Writable+Comparable<T>

| Java type | Hadoop type |
|-----------|-----------------|
| Boolean | BooleanWritable |
| Byte | ByteWritable |
| Double | DoubleWritable |
| Integer | IntWritable |
| Long | LongWritable |
| String | Text |
| null | NullWritable |

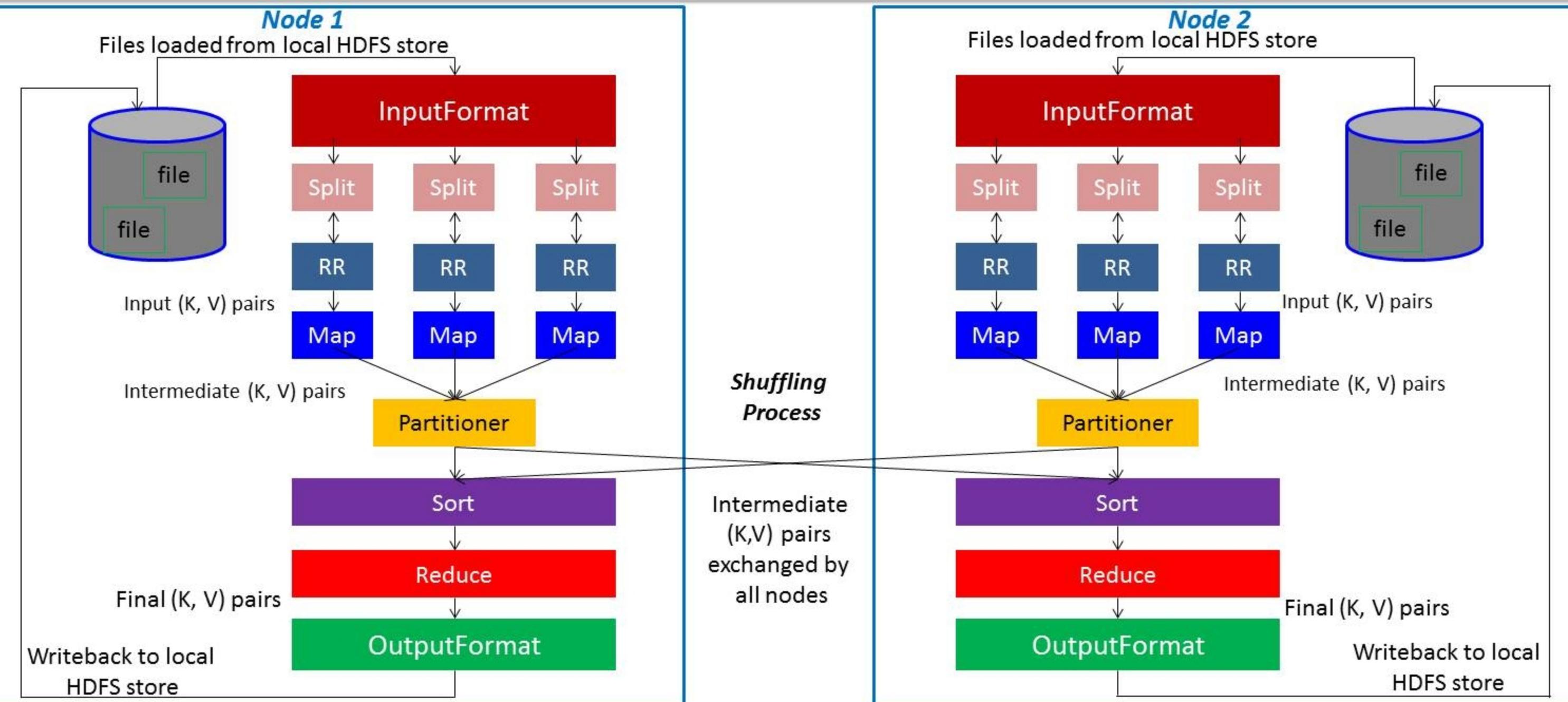
For any class to be value, it has to implement

- org.apache.hadoop.io.Writable
 - write(DataOutput out)
 - readFields(DataInput in)

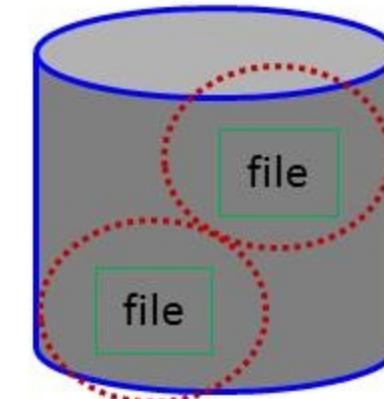
For any class to be key, it has to implement

- org.apache.hadoop.io.WritableComparable<T>
 - write(DataOutput out)
 - readFields(DataInput in)
 - compareTo(T o)

MapReduce – A Closer Look



- Input files are where the data for a MapReduce task is initially stored
- The input files typically reside in a distributed file system (e.g. HDFS)
- The format of input files is arbitrary
 - Line-based log files
 - Binary files
 - Multi-line input records
 - Or something else entirely

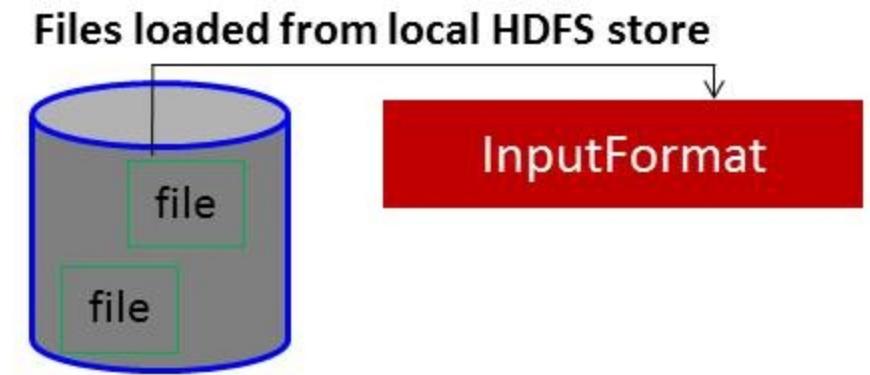


Large File



- Big File is divided into multiple blocks and stored in hdfs.
- This is a physical division of data
- `dfs.block.size` (128MB default size)

- How the input files are split up and read is defined by the *InputFormat*
- InputFormat is a class that does the following:
 - Select the files that should be used for input
 - Defines the *InputSplits* that break a file
 - Provides a factory for *RecordReader* objects that read the file



InputFormat (Contd..)

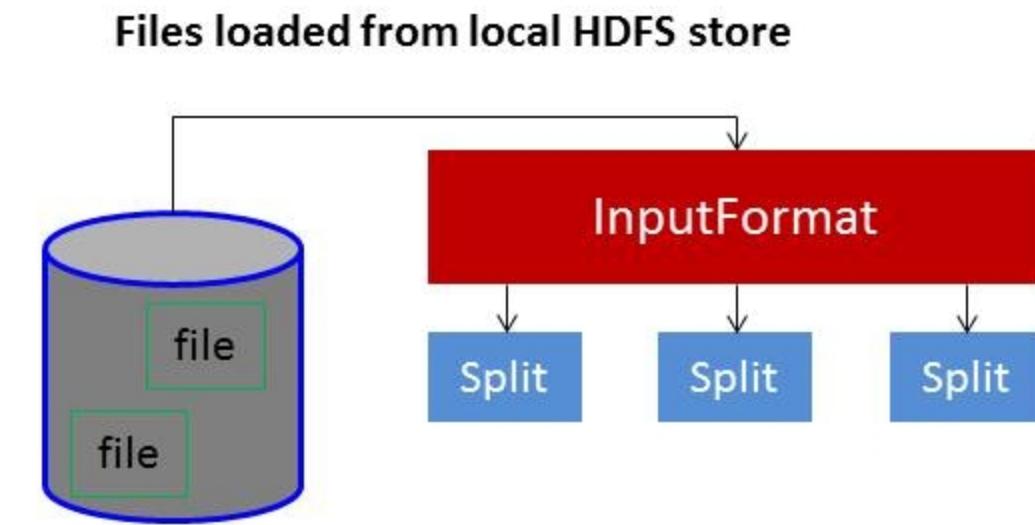
- An InputFormat is responsible for creating the input splits and dividing them into records.
- How we get the data to mapper

```
public abstract class InputFormat<K, V> {  
    public abstract List<InputSplit> getSplits(JobContext context)  
        throws IOException, InterruptedException;  
  
    public abstract RecordReader<K, V>  
        createRecordReader(InputSplit split,  
        TaskAttemptContext context) throws IOException,  
        InterruptedException;  
}
```

The Hadoop framework provides a large variety of input formats:

- **TextInputFormat**: The key is the byte offset, and the value is the line.
- **KeyValueInputFormat**: Key upto first tab char, rest is the value
- **NLineInputFormat**: Similar to TextInputFormat, but it provides n lines of input.
- **MultiFileInputFormat**: An abstract class that lets the user implement an input format that aggregates multiple files into one split.
- **SequenceFileInputFormat**: The input file is a Hadoop sequence file, containing serialized key/value pairs.

- An *input split* describes a unit of work that comprises a single map task in a MapReduce program
- By default, the InputFormat breaks a file up into 64MB splits
- By dividing the file into splits, it allows several map tasks to operate on a single file in parallel
- If the file is very large, this can improve performance significantly through parallelism
- Each map task corresponds to a *single* input split



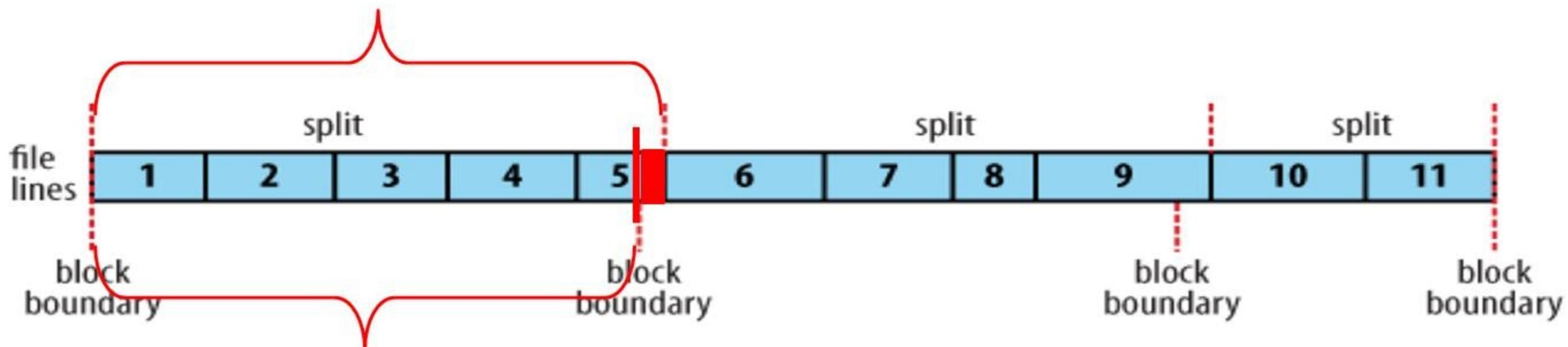
Input split

- A chunk of data processed by a mapper
- Further divided into records
- Map process these records
 - Record = key + value
- How to correlate to a DB table
 - Group of rows → split
 - Row → record

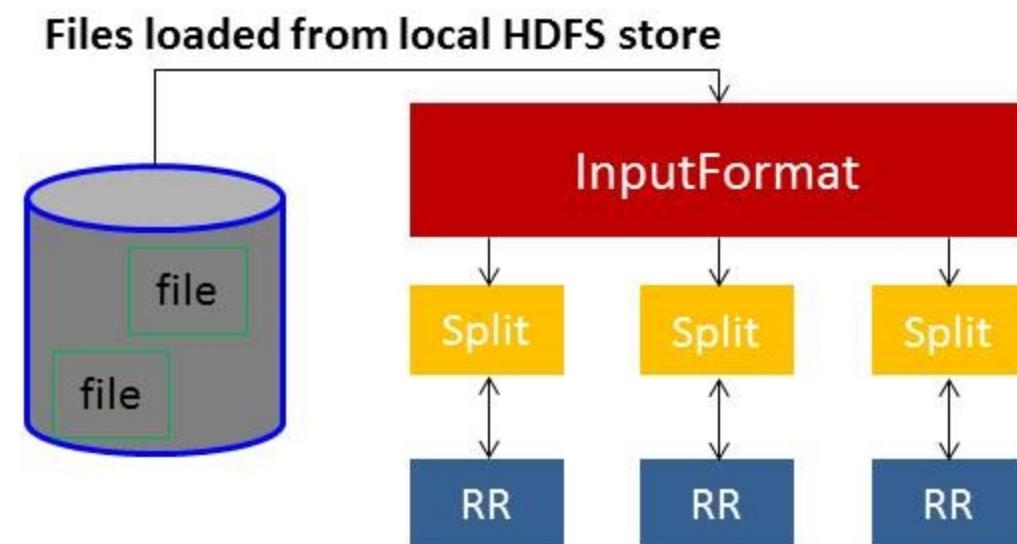


| Key | Value |
|-----|-------|
| | |
| | |
| R1 | R1 |
| R2 | R2 |
| R3 | R3 |
| | |
| | |

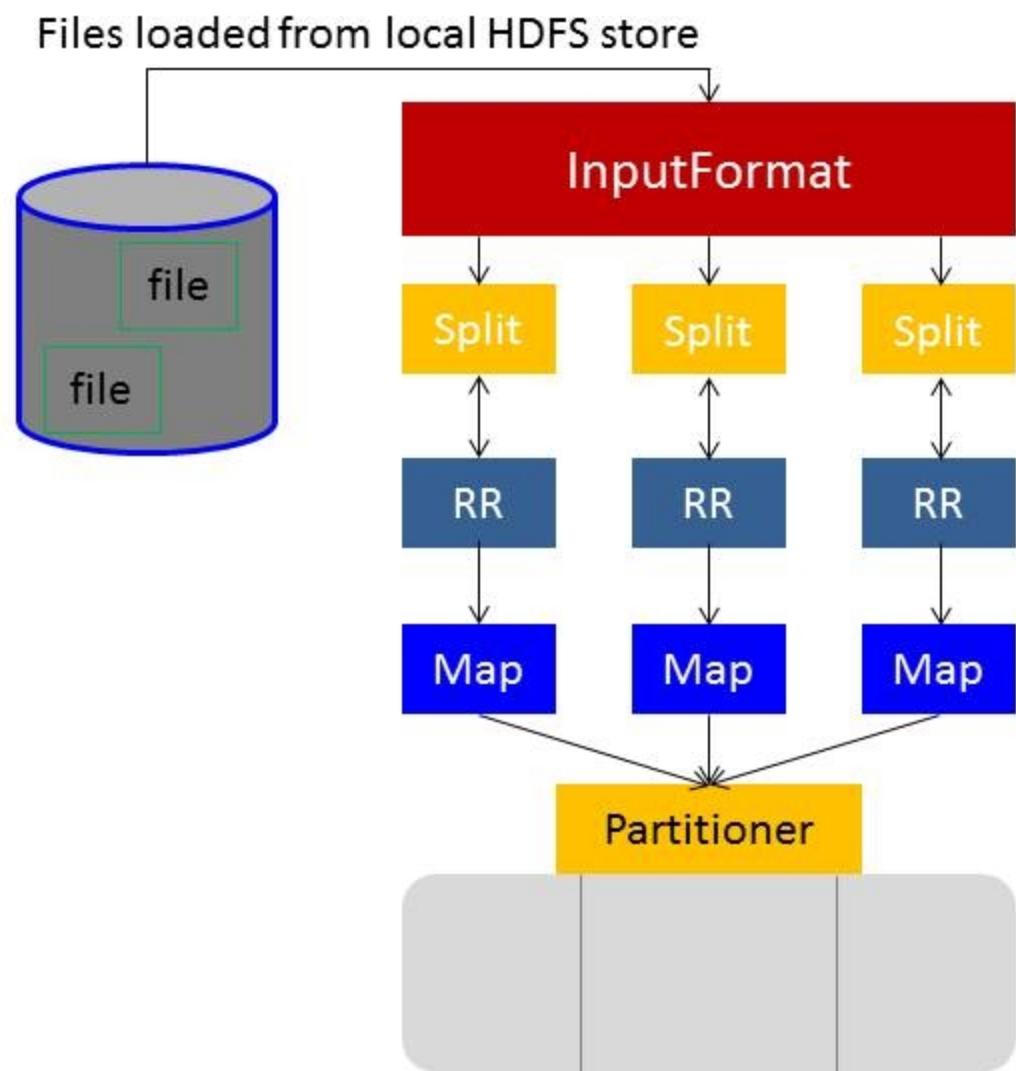
- Split gives logical record boundaries
- Blocks – physical boundaries (size)
- Even in data locality some remote reading is done, a slight overhead.



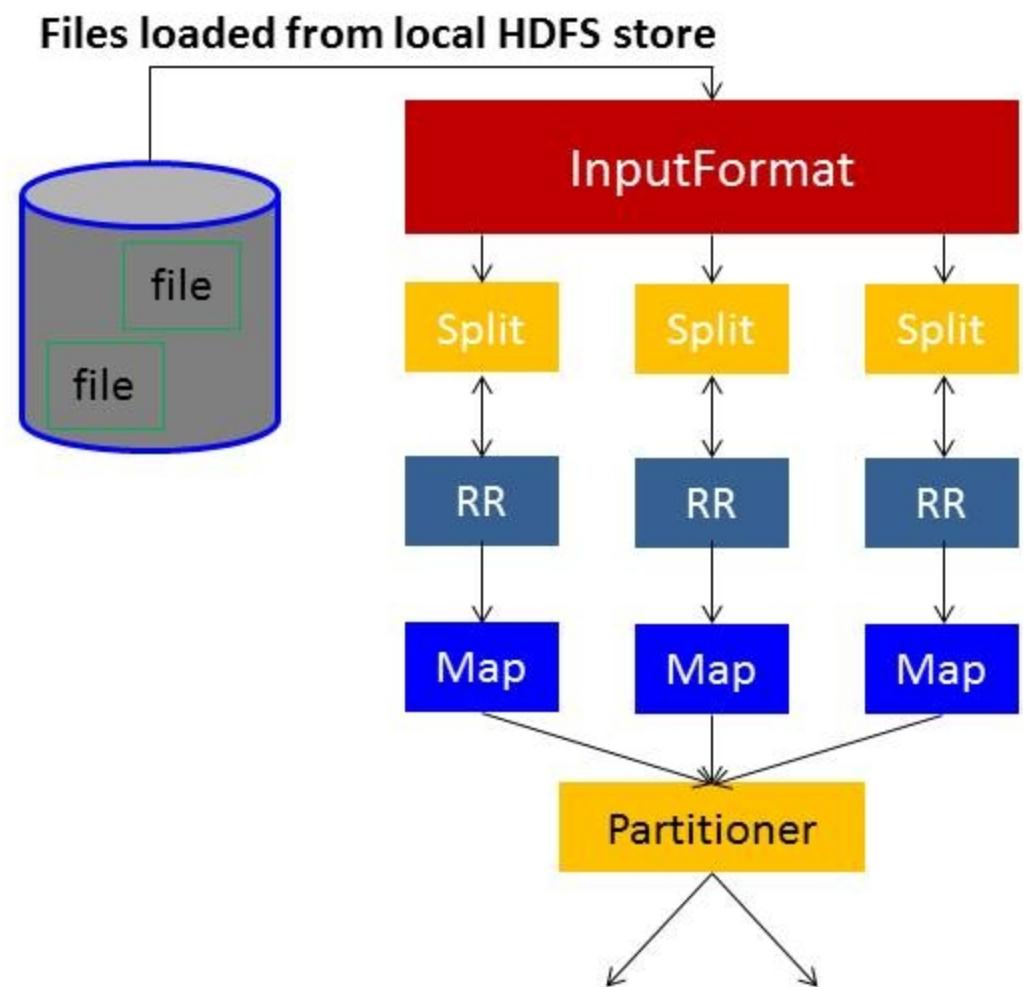
- The input split defines a slice of work but does not describe how to access it
- The *RecordReader* class actually loads data from its source and converts it into (K, V) pairs suitable for reading by Mappers
- The RecordReader is invoked repeatedly on the input until the entire split is consumed
- Each invocation of the RecordReader leads to another call of the map function defined by the programmer



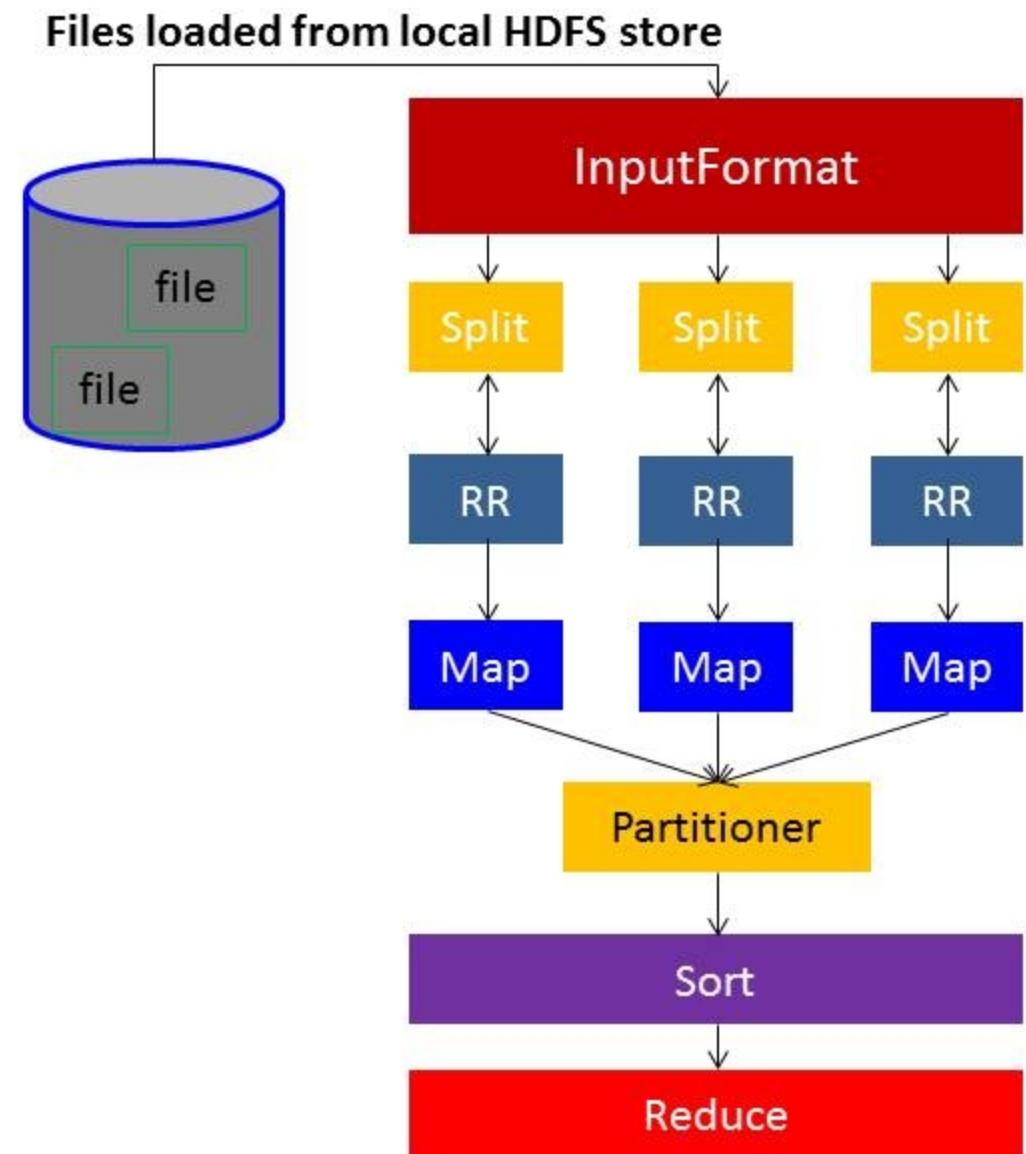
- Each mapper may emit (K, V) pairs to *any* partition
- Therefore, the map nodes must all agree on where to send different pieces of intermediate data
- The *partitioner* class determines which partition a given (K,V) pair will go to
- The default partitioner computes a *hash value* for a given key and assigns it to a partition based on this result



- The Process of moving map outputs to the reducers is known as *Shuffling*
- After the first map task has completed, the nodes start exchanging intermediate outputs from map tasks to the reducers
- Shuffling of the data is done via the network, hence MapReduce job's speed is limited to network bandwidth



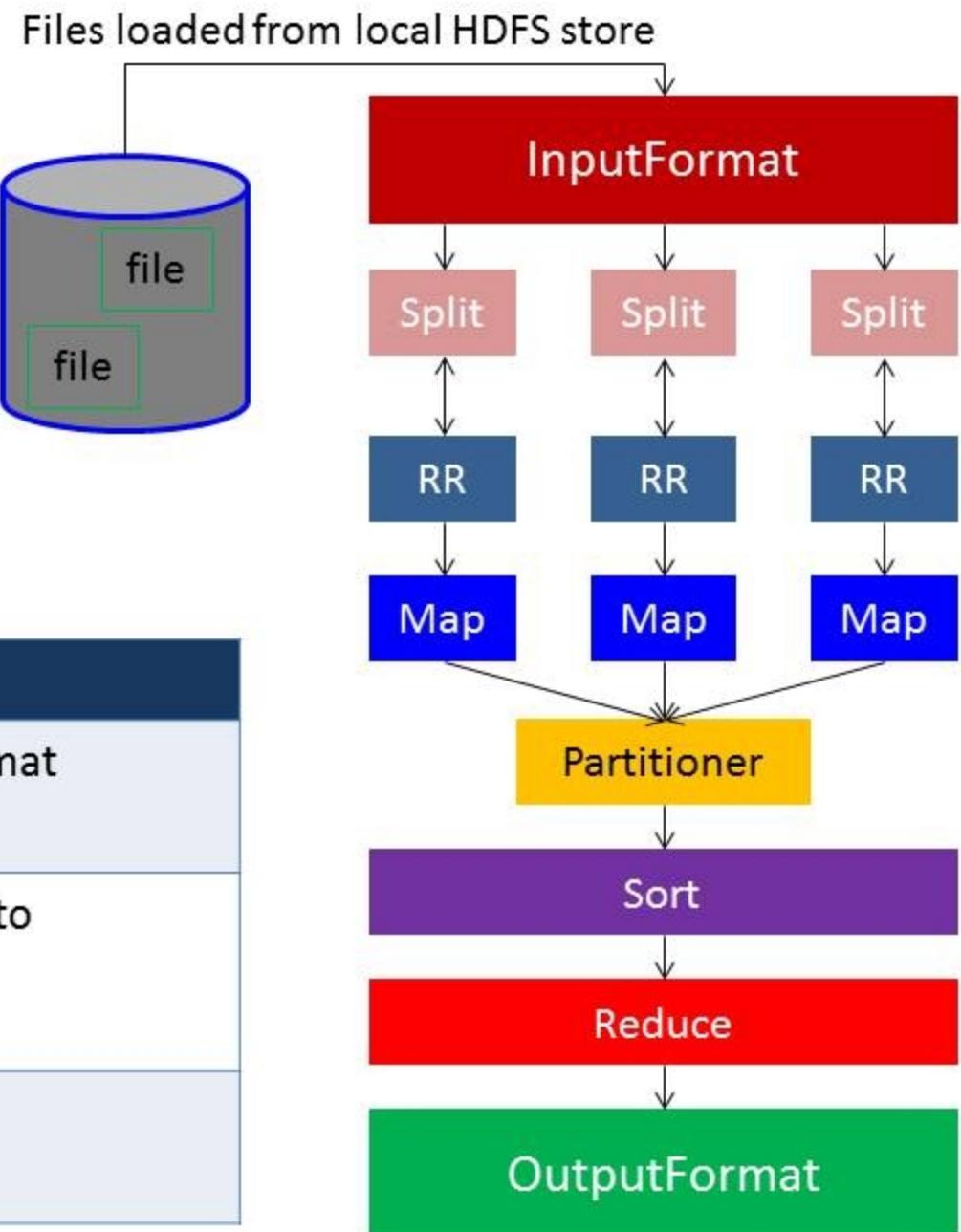
- Each Reducer is responsible for reducing the values associated with (several) intermediate keys
- The set of intermediate keys on a single node is *automatically sorted* by MapReduce before they are presented to the Reducer



OutputFormat

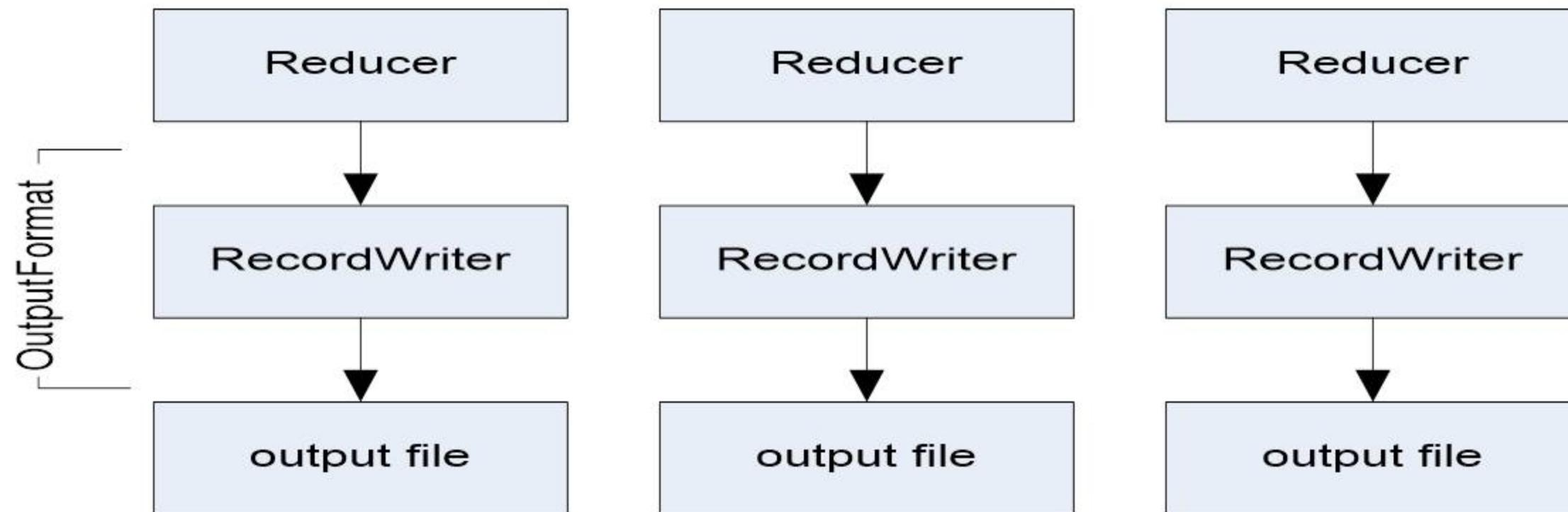
- The *OutputFormat* class defines the way (K,V) pairs produced by Reducers are written to output files
- The instances of *OutputFormat* provided by Hadoop write to files on the local disk or in HDFS
- Several *OutputFormats* are provided by Hadoop:

| OutputFormat | Description |
|--------------------------|---|
| TextOutputFormat | Default; writes lines in "key \t value" format |
| SequenceFileOutputFormat | Writes binary files suitable for reading into subsequent MapReduce jobs |
| NullOutputFormat | Generates no output files |

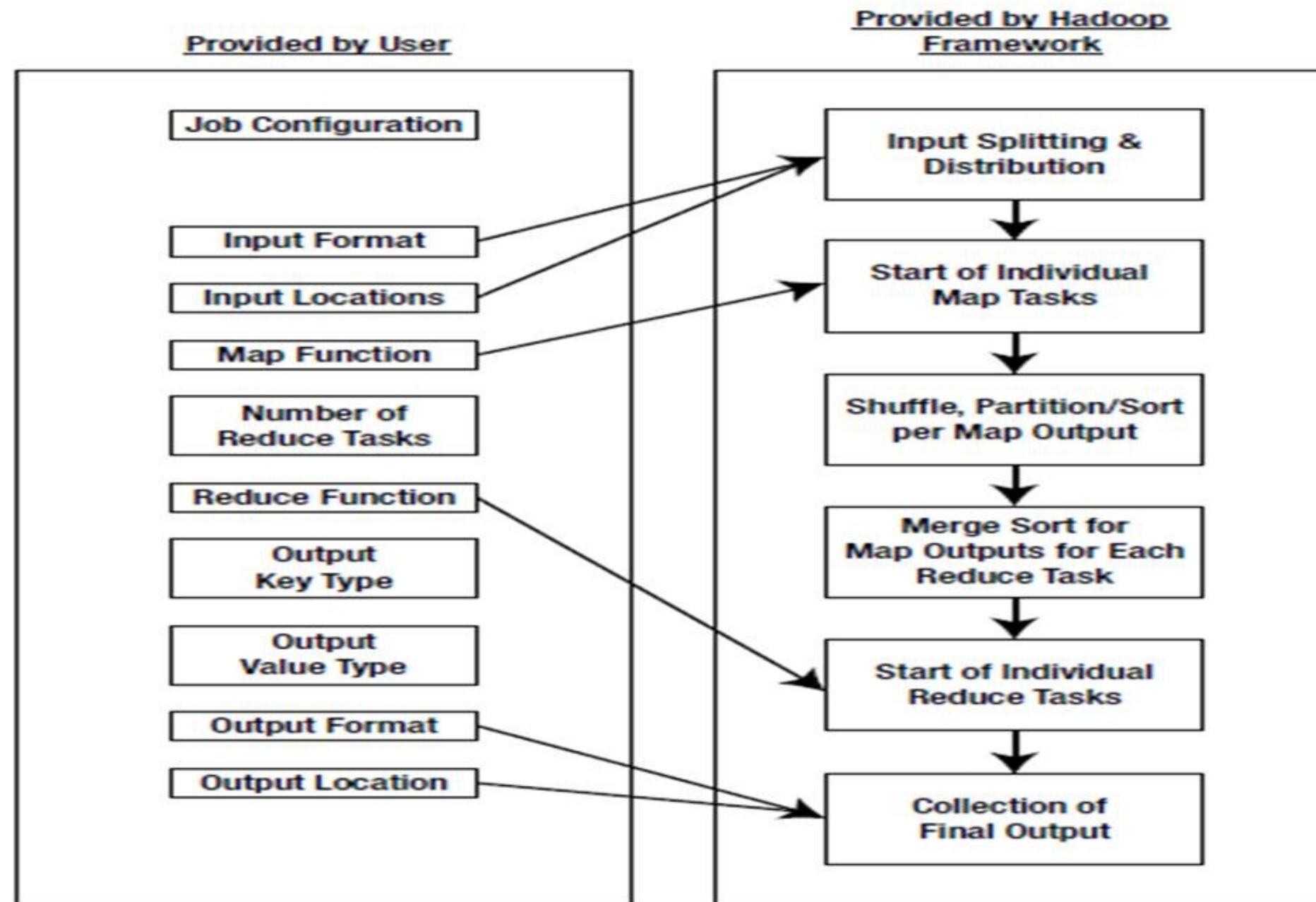


OutputFormat Continued...

- Given the key, value pairs and a filename, writes the reduce task output to persistent store.
- RecordWriter classes handle the job of taking an individual key-value pair and writing it to the location prepared by the OutputFormat.



MapReduce in a NutShell



Thank You

DataFlair Web Services Pvt Ltd

+91-8451097879 / +91-7718877477

info@data-flair.com

<http://data-flair.com>

<https://www.facebook.com/DataFlairWS>