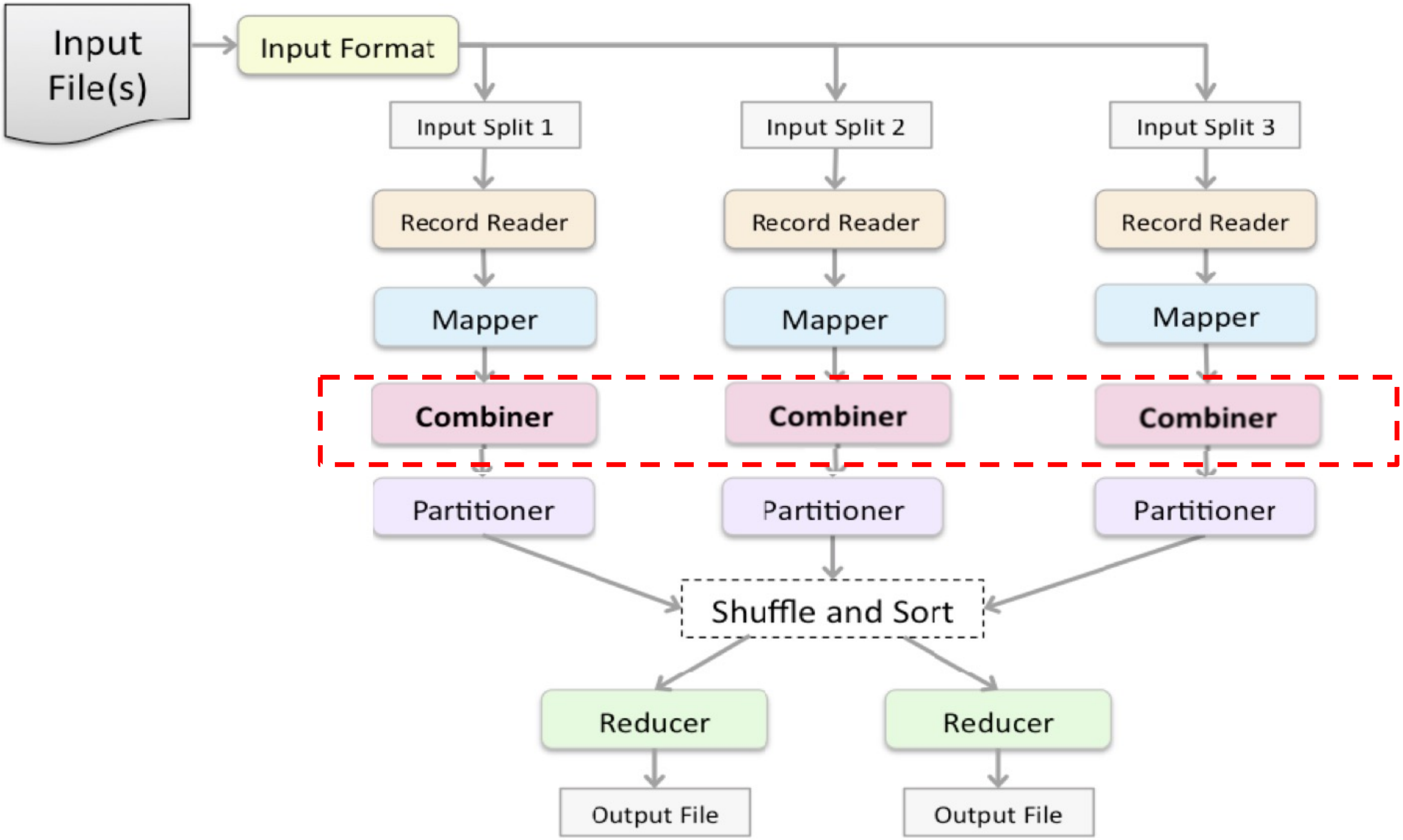
The MapReduce Flow (Recap)



The Combiner (Revisit)

Combiner (Mini-reducer): performs local aggregation on a single mapper's output.

Helps to minimize the data transfer between mapper and reducer.

Example:

Input:

Block 1: Hello World Bye World

Block 2: Hello Hadoop Goodbye Hadoop

8 keys

Output of Mapper 1: <Hello, 1> <World, 1> <Bye, 1> <World, 1>

Output of Mapper 2: <Hello, 1> <Hadoop, 1> <Goodbye, 1> <Hadoop, 1>

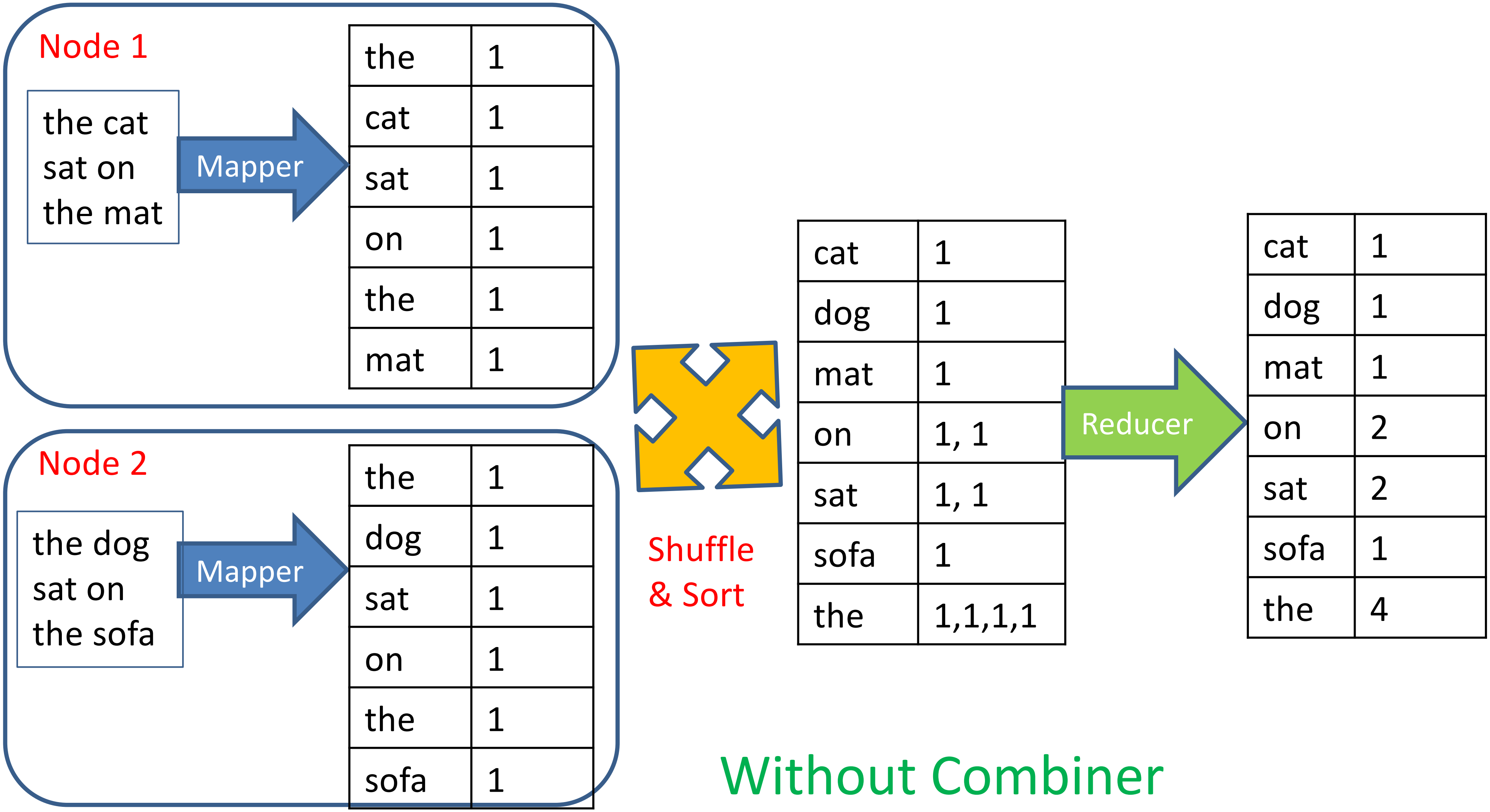
6 keys

With Combiner:

The output of Combiner 1: <Bye, 1> <Hello, 1> <World, 2>

The output of Combiner 2: <Goodbye, 1> <Hadoop, 2> <Hello, 1>

Another Example



Node 1

the cat
sat on
the mat

Mapper

the	1
cat	1
sat	1
on	1
the	1
mat	1

Node 2

the dog
sat on
the sofa

Mapper

the	1
dog	1
sat	1
on	1
the	1
sofa	1

Shuffle
& Sort

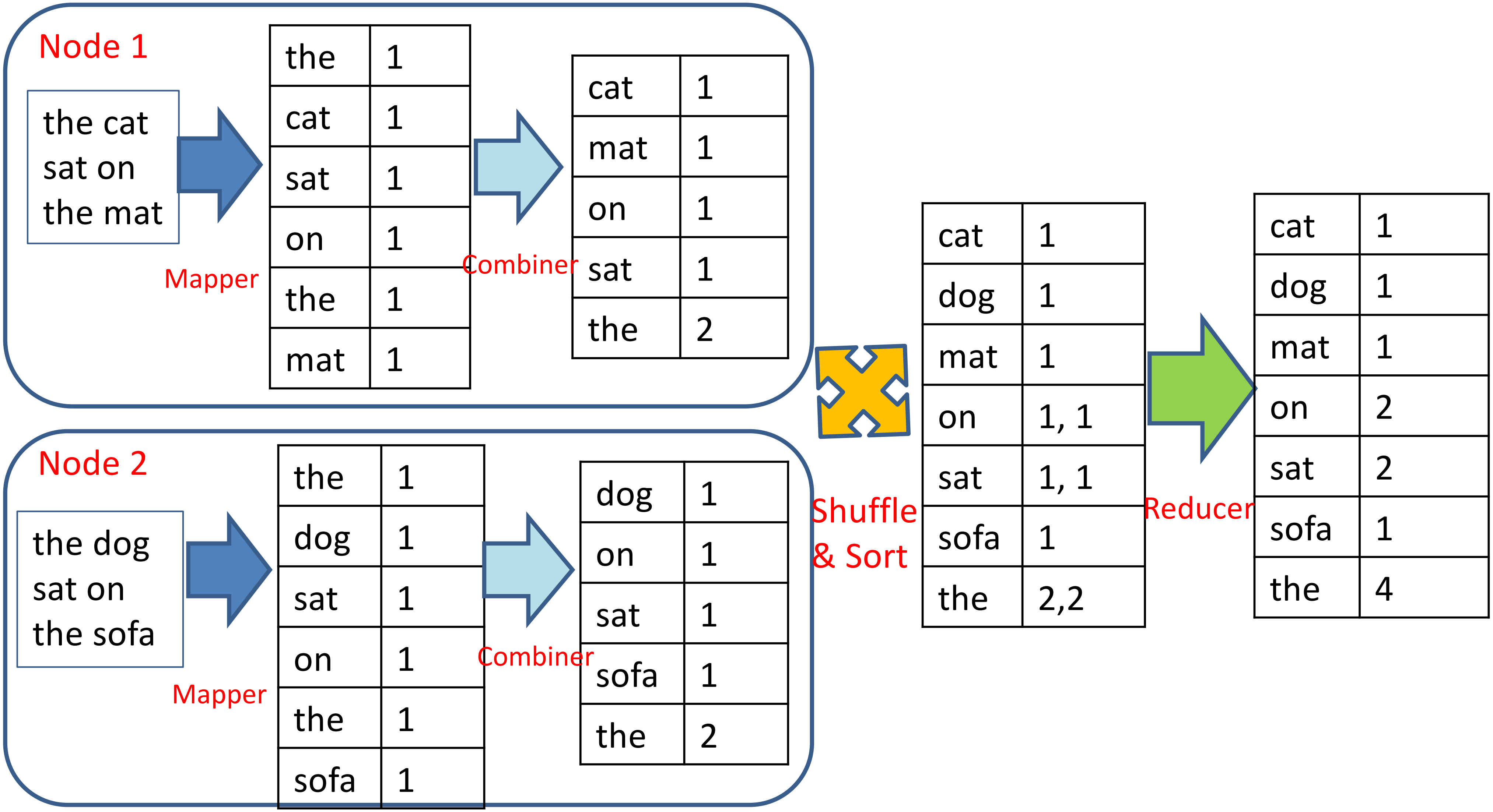
cat	1
dog	1
mat	1
on	1, 1
sat	1, 1
sofa	1
the	1,1,1,1

Reducer

cat	1
dog	1
mat	1
on	2
sat	2
sofa	1
the	4

Without Combiner

Another Example (Cont.)



Node 1

the cat
sat on
the mat

Mapper

the	1
cat	1
sat	1
on	1
the	1
mat	1

Combiner

cat	1
mat	1
on	1
sat	1
the	2

Node 2

the dog
sat on
the sofa

Mapper

the	1
dog	1
sat	1
on	1
the	1
sofa	1

Combiner

dog	1
on	1
sat	1
sofa	1
the	2

Shuffle
& Sort

cat	1
dog	1
mat	1
on	1, 1
sat	1, 1
sofa	1
the	2,2

Reducer

cat	1
dog	1
mat	1
on	2
sat	2
sofa	1
the	4

With Combiner

Writing a Combiner

- Combiner and Reducer code are often identical (i.e. Reducer can be reused as a combiner)

  - if the operation performed is commutative and associative

    ✓ Commutative: a + b = b + a

    ✓ Associative : a + (b + c) = (a + b) + c

  - Input and output data types for the Combiner/Reducer must be identical

    ✓ Takes in a key and a list of values

    ✓ Outputs (key, value) pairs

Understanding Commutative and Associative

- Example: Suppose we're analyzing the traffic of a website. Given an input file with the number of visits per day, we want to find which is the day with the highest number of visits.

Input file:

20180201	200
20180131	2000
20180130	1600
20180129	4800
20180128	2600

date # of visits

max function is both commutative and associative, so the reducer can be reused as combiner

Assume we have two mappers: the first one receives the first three lines and the second receives the last two.

Without Combiner:
Then the reducer will do something like:
max(200, 2000, 1600, 4800, 2600)
The result is 4800.

With Combiner:
Then each combiner will evaluate locally the max function.
And then the reducer will do something like:
max(max(200, 2000, 1600), max(4800, 2600))
The result is 4800.

Understanding Commutative and Associative

- What about *sum*?

  - commutative and associative. e.g. You can re-use your Reducer as a Combiner for WordCount problem

- What about *average*? In this case, the Reducer can NOT be re-used as Combiner

Input file:

date	# of visits
20180201	200
20180131	2000
20180130	1600
20180129	4800
20180128	2600

Example: Given an input file with the number of visits per day, we want to compute the average number of visits per day.
Assume we have two mappers: the first one receives the first three lines and the second receives the last two.

Without Combiner:
Then the reducer will do something like:
avg(200, 2000, 1600, 4800, 2600)
The result is 2240.

With Combiner:
Then each combiner will evaluate locally the average. And then the reducer will do something like:
avg(avg(200, 2000, 1600), avg(4800, 2600)) = avg(1266, 3700) ❌
The result is 2483.

Specifying a Combiner

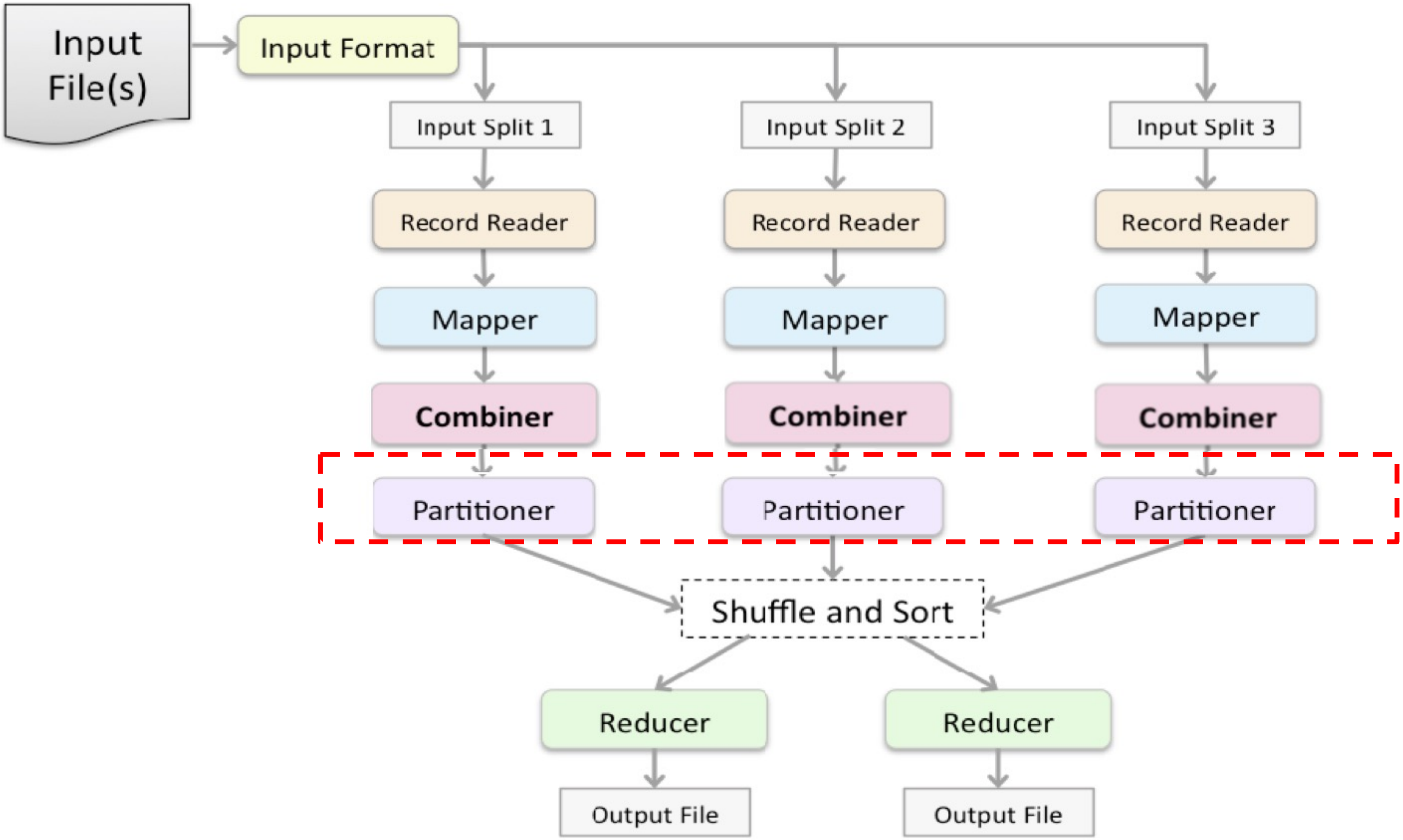
- If we’re computing a commutative and associative function and want to improve the performance of our job, we can re-use the reducer as the combiner.

  - set the combiner in the driver code

```
job.setMapperClass(WordMapper.class);
Job.setReducerClass(SumReducer.class);
job.setCombinerClass(SumReducer.class);
```

- If we want to improve performance but we’re computing a function that is NOT commutative and associative, we have to rewrite the mapper or to write a new combiner.

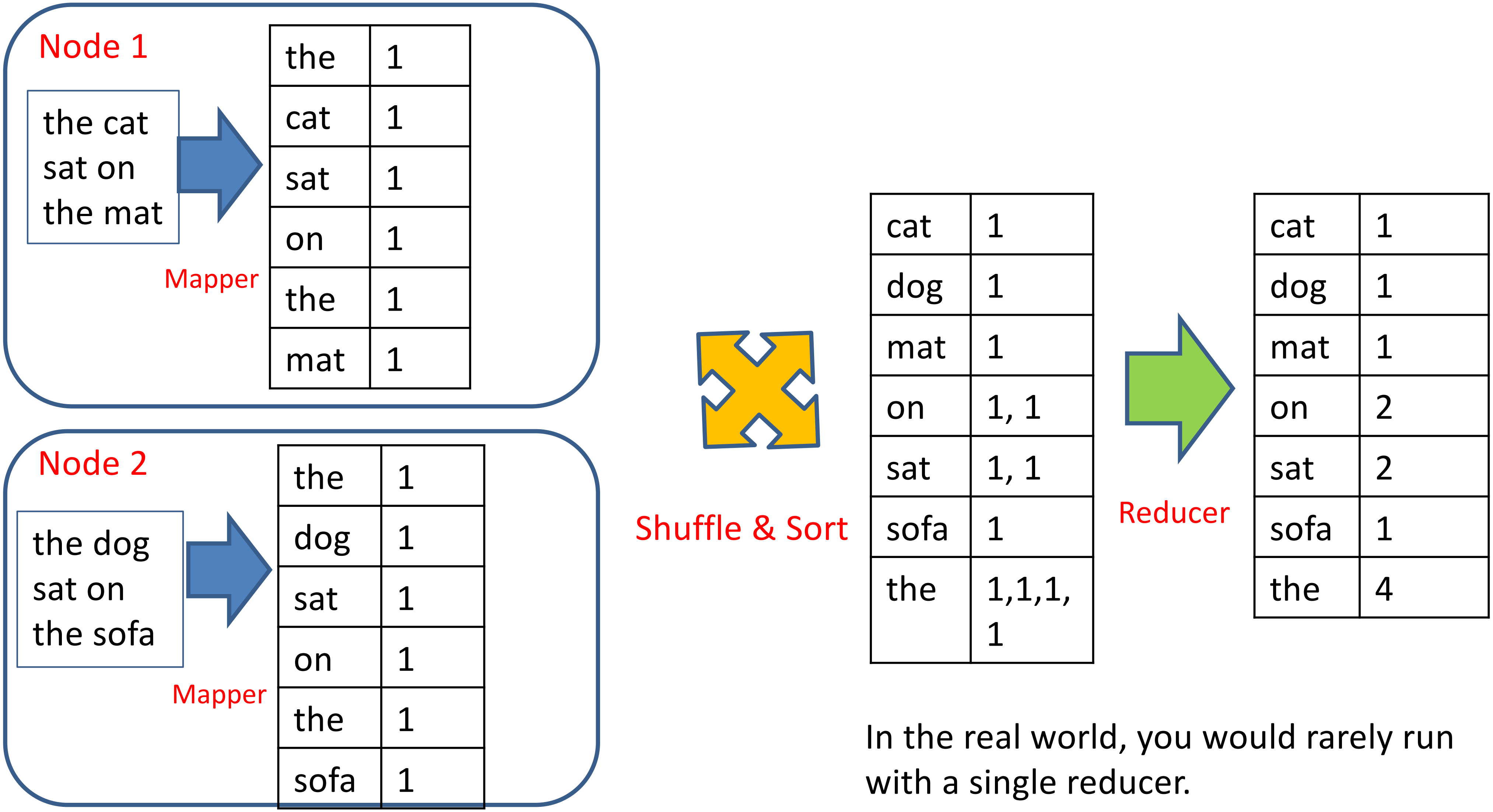
The MapReduce Flow (Recap)



What Does the Partitioner Do?

- The Partitioner determines which Reducer each intermediate key and its associated values goes to
- The default Partitioner (*HashPartitioner*) tries to evenly distribute the keys across all the Reducers
- The default number of Reducer on the VM in this class is 1 (in this case, no Partitioner is used)

Example: WordCount with One Reducer



Node 1

the cat
sat on
the mat

Mapper

the	1
cat	1
sat	1
on	1
the	1
mat	1

Node 2

the dog
sat on
the sofa

Mapper

the	1
dog	1
sat	1
on	1
the	1
sofa	1

Shuffle & Sort

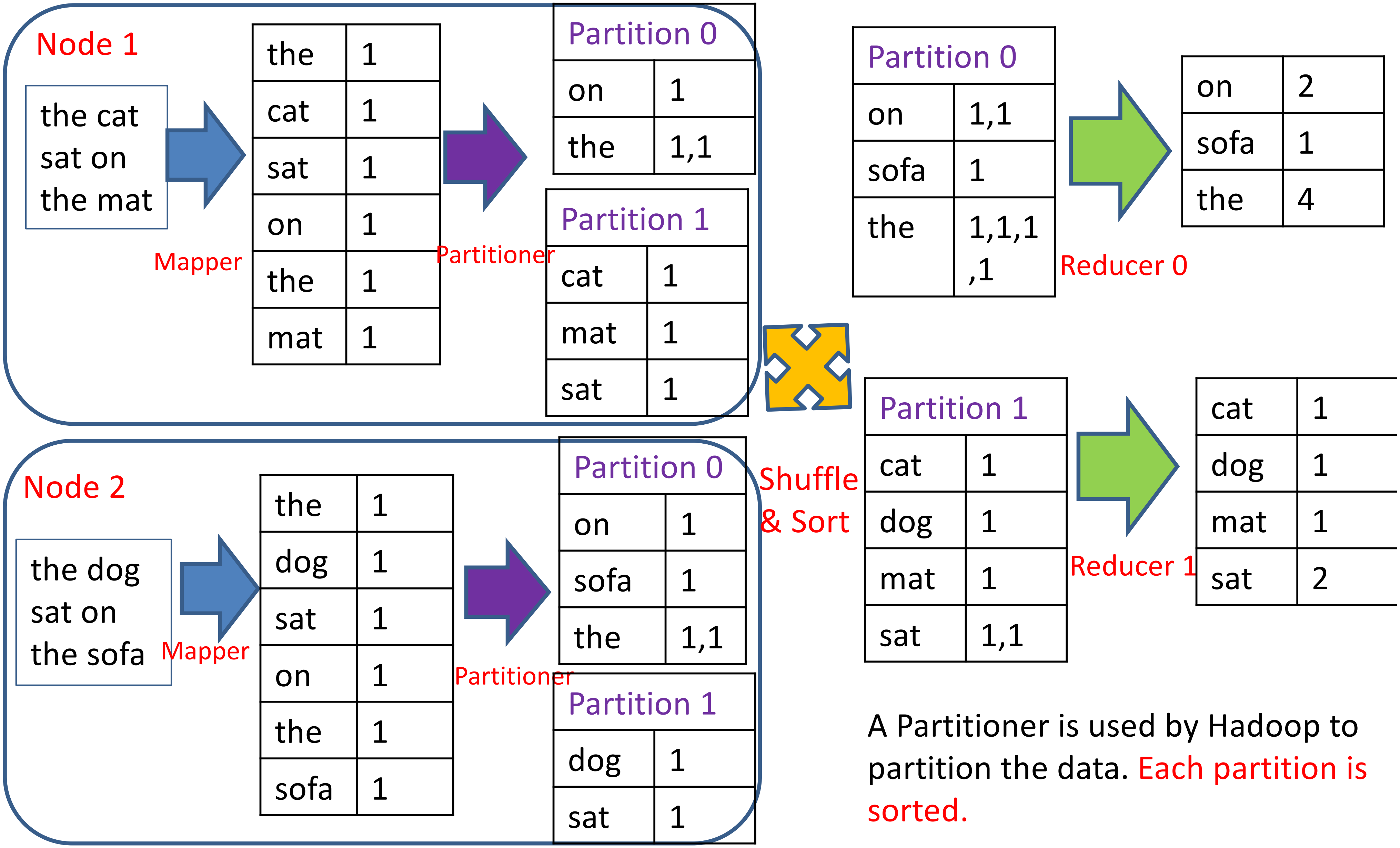
cat	1
dog	1
mat	1
on	1, 1
sat	1, 1
sofa	1
the	1,1,1, 1

Reducer

cat	1
dog	1
mat	1
on	2
sat	2
sofa	1
the	4

In the real world, you would rarely run with a single reducer.

Example: WordCount with Two Reducers



Node 1

the cat
sat on
the mat

Mapper

the	1
cat	1
sat	1
on	1
the	1
mat	1

Partitioner

Partition 0	
on	1
the	1,1

Partition 1	
cat	1
mat	1
sat	1

Node 2

the dog
sat on
the sofa

Mapper

the	1
dog	1
sat	1
on	1
the	1
sofa	1

Partitioner

Partition 0	
on	1
sofa	1
the	1,1

Partition 1	
dog	1
sat	1

Shuffle
& Sort

Partition 0	
on	1,1
sofa	1
the	1,1,1 ,1

Reducer 0

on	2
sofa	1
the	4

Partition 1	
cat	1
dog	1
mat	1
sat	1,1

Reducer 1

cat	1
dog	1
mat	1
sat	2

A Partitioner is used by Hadoop to partition the data. Each partition is sorted.

The Default Partitioner

- The default Partitioner is the *HashPartitioner*

  - use the Java *hashCode* method

  - guarantee all pairs with the same key go to the same partition

Type of key and value (the same as type of output key and value of Mapper)

```
public class HashPartitioner<K, V> extends Partitioner<K, V> {

     // key: the key to be partitioned
    // value: the entry value
    // numReduceTasks: total number of reducer (= total number of partitions)
    // returns the partition number for the key
    public int getParition(K key, V value, int numReduceTasks) {
          // in a more readable way would be
         // "return Math.abs(key.hashCode() % numReduceTasks);"
          return (key.hashcode() & Integer.MAX_VALUE) %  numReduceTasks;
    }
```

How Many Reducers?

- An important consideration when creating your job is to determine the number of Reducers specified.
- Pros and Cons of a single Reducer
  - o Advantage:

    - The output can be in completely sorted order
  - o Disadvantage:

    - can cause significant problems if there is a large amount of intermediate data

      - ✓ The Reducer will take a long time to run
      - ✓ Node on which the Reducer is running may not have enough disk space to hold all intermediate data

Jobs Which Require a Single Reducer

- If a job needs to output a file where all keys are listed in sorted order, a single Reducer can be used
- Alternatively, the TotalOrderPartitioner can be used
  - ✓ Uses an externally generated file which stores the sorted partition keyset
  - ✓ Partitions data such that all keys which go to the first Reducer are smaller than any which go to the second, etc.
  - ✓ In this way, multiple Reducers can be used
  - ✓ Concatenating the Reducer’s output files results in a totally ordered list

Jobs Which Require Multiple Reducers

- Example 1: We want to generate retail sales report based on month

  - 12 Reducers

- Example 2: We want to know how much traffic do top websites get per day of a week

  - 7 Reducers

A custom Partitioner should be written to send records to Reducer (e.g. send all records for Monday to Reducer 0, send all records for Tuesday to Reducer 1, etc.)

Creating and Using a Custom Partitioner

- Create a class that extends Partitioner
- Override the *getPartition* method

the same as the output key and value type of Mapper

```
import org.apache.hadoop.mapreduce.Partitioner;
public class MyPartitioner extends Partitioner<K, V> {
     @Override
    public int getParition(K key, V value, int numReduceTasks) {
       //determine reducer number between 0 and numReduceTasks and return it
       //e.g. if there are 10 Reducers, return an int between 0 and 9
      ……
    }
}
```

- Specify the custom Partitioner and the number of Reducers in your driver code

```
job.setNumReduceTasks(12); //e.g. specify 12 reducers
job.setPartitionerClass(MyPartitioner.class);
```

Setting up Variables for your Partitioner

- If you need to set up variables in your partitioner, it should implement *Configurable*
- If a Hadoop object implements *Configurable*, its *setConf()* method will be called once, when it is instantiated
- You can therefore set up variables in the *setConf()* method which your *getPartition()* method will then be able to access

The In-class exercise later will help you understand this.

Setting up Variables for your Partitioner (Cont.)

```
public class MyPartitioner<K, V> extends Partitioner<K, V> implements Configurable {
    private Configuration configuration;
   / / Define your own variables here
  ….
   @Override
   //set the configuration to be used by the partitioner
    public void setConf(Configuration configuration) {
        this.configuration = configuration;
        // Set up your variables here
    }
   @Override
   //return the configuration used by the partitioner
   //must implement the getConf() method for the Configurable interface
    public Configuration getConf() {
        return configuration;
    }
    @Override
     public int getParition(K key, V value, int numReduceTasks) {
         //determine reducer number between 0 and numReduceTasks and return it
         //e.g. if there are 10 Reducers, return an int between 0 and 9
        // use your variables here
        ……
     }
}
```

In-Class Exercise: Customize Partitioner

- Analyze a log file from a web server to count the number of hits made from each unique IP address.

- The final output should consist of 12 files, one for each month of the year: January, February, and so on.

- Each file should contain a list of IP addresses, and the number of hits from that address in that month.

Discussion

- What are the key and value of input data to Mapper?

  key: byte offset within the file; value: text of each line

- What are the key and value of output data of Mapper?

  key: IP address; value: the month

- What are the key and value of input data to Reducer?

  key: IP address;
  value: a list of text (the month) (e.g. Jan, Jan)

- What are the key and value of output data of Reducer?

  key: IP address;
  value: the total of hits from the IP address in that month

Discussion (Cont.)

- How many Reducers?

  12

- How to partition?

```
public int getParition(Text key, Text value, int numReduceTasks) {
    //this method receives the month (e.g. Jan, Feb) as its value
    //and returns an integer representation of the month
    // e.g. For Jan, return 0; For Feb, return 1, and so on.
   ……
}
```

Now Let’s write programs!