BIG DATA

Map Reduce Programming model and Architecture

K V Subramaniam
Computer Science and Engineering

What we have learnt so far..

- Large amounts of data to be processed.
- We have HDFS as a distributed store.
- We need to distribute the processing also.

What is Map Reduce ?

BIG DATA

Map Reduce Programming model and Architecture

Why Map-Reduce ?

- A new fundamental way to process extremely large data (?)
- We are going to
  - Study Map-Reduce paradigm
  - Study Hadoop architecture
    - Open Source implementation of Map-Reduce

What is MapReduce ?

- Origin from Google, [OSDI'04]
- A simple programming model
- Functional model
- For large-scale data processing
  - Exploits large set of commodity computers
  - Executes process in distributed manner
  - Offers high availability

Big Data: Map Reduce Motivating Example

Map Reduce - Motivation

- Lots of demands for very large scale data processing

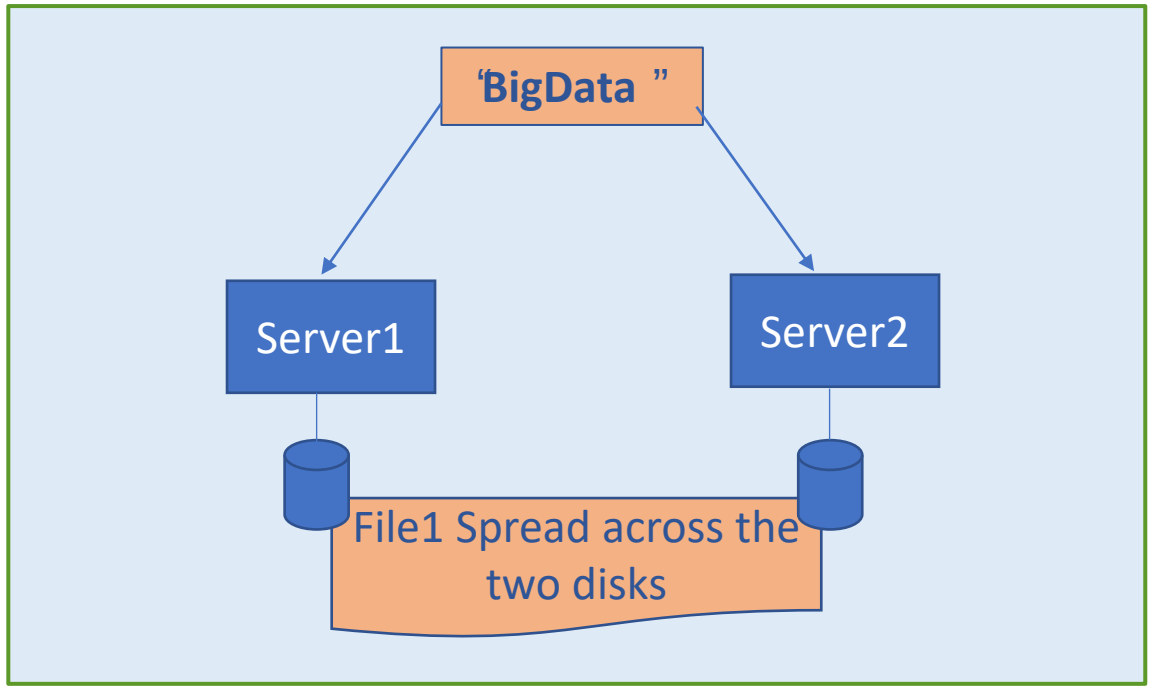
- Lots of machines needed (scaling)
- Two basic operations on the input
  - Map
  - Reduce

- To understand what Map/Reduce really are..

BIG DATA

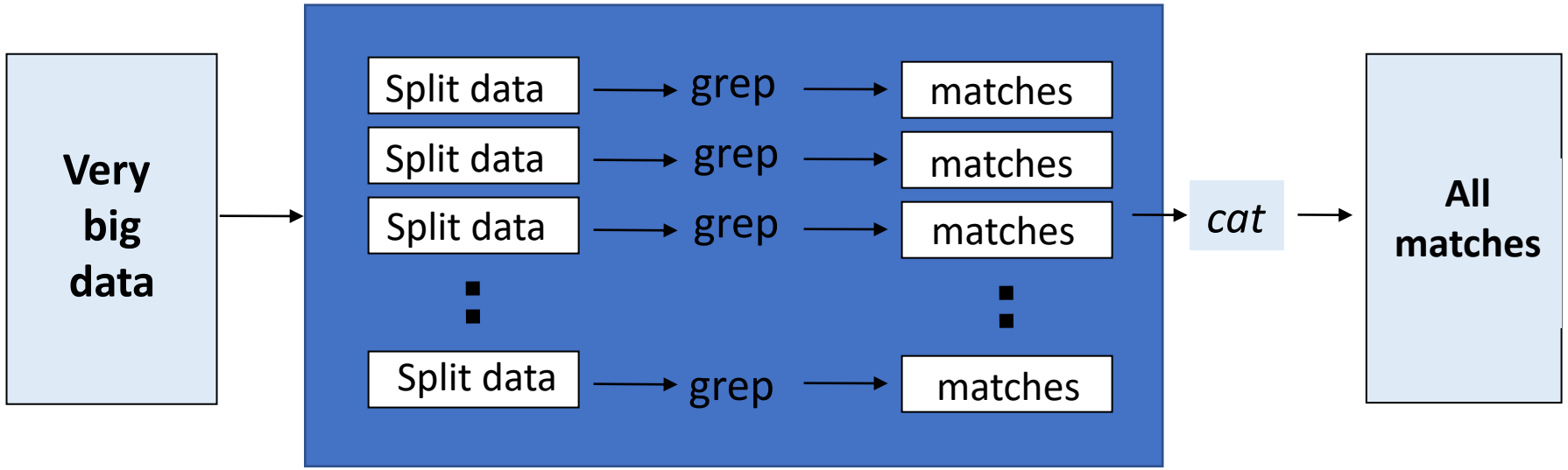
A MapReduce Example

- Consider a very large text file and you want to determine if a word exists in this file?
  - The file is stored in HDFS across two machines.
  - How will you search if BigData is present in the file?



BIG DATA

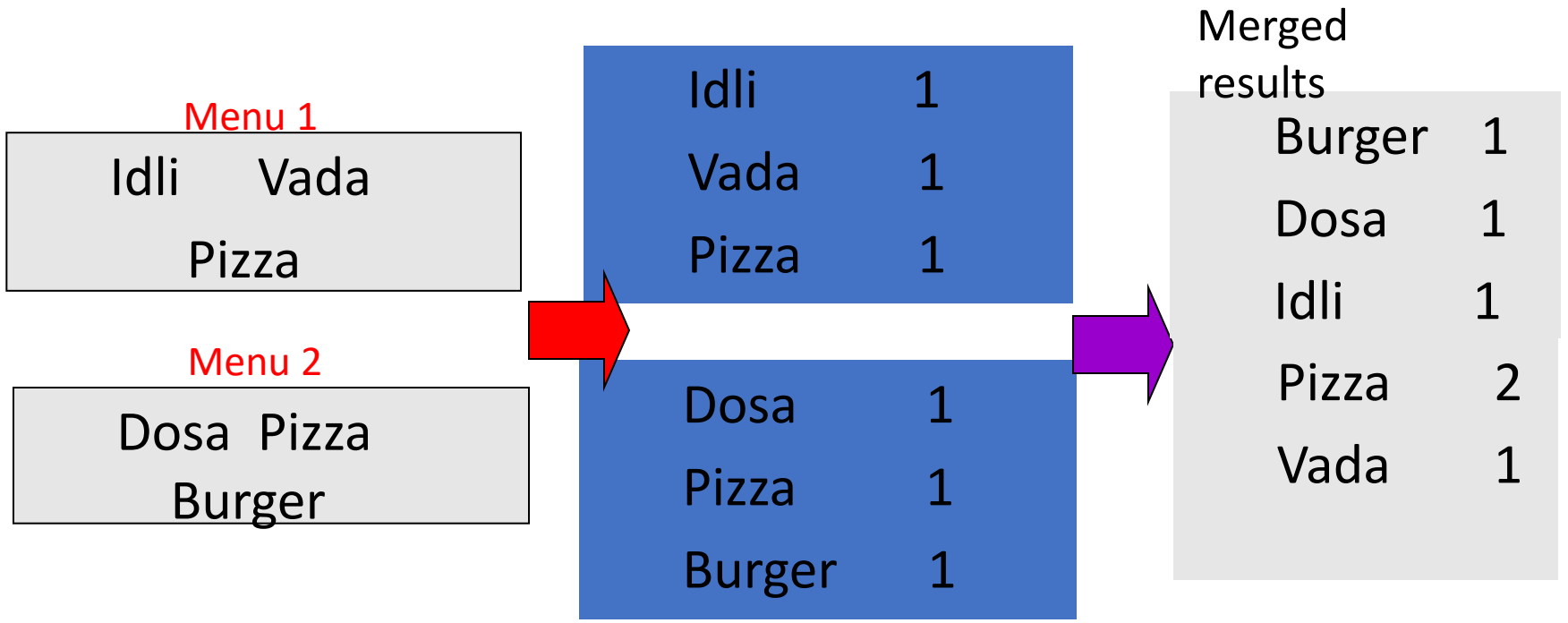
Map Reduce: Distributed Grep-Solution



Distributed Word Count

BIG DATA

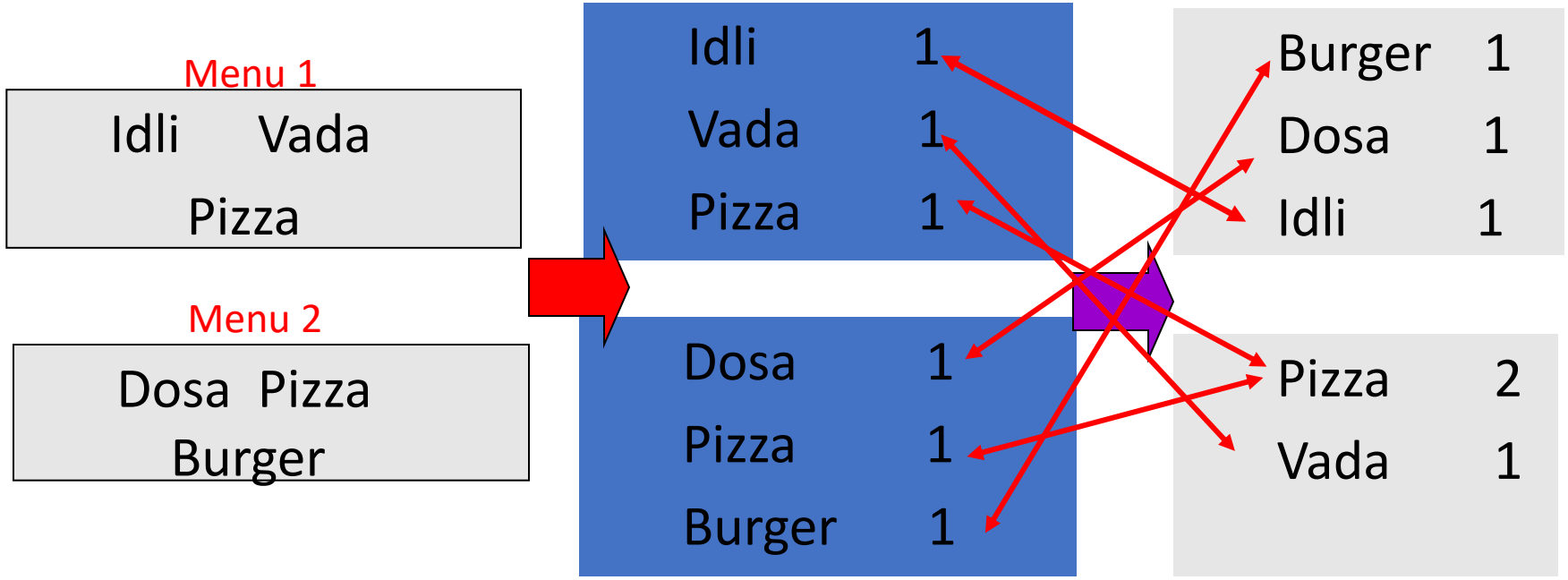
Example: find the number of restaurants offering each item?



- Suppose we need parallelism of the merge program.
- Would the earlier approach work?
- What do we need to do?

BIG DATA

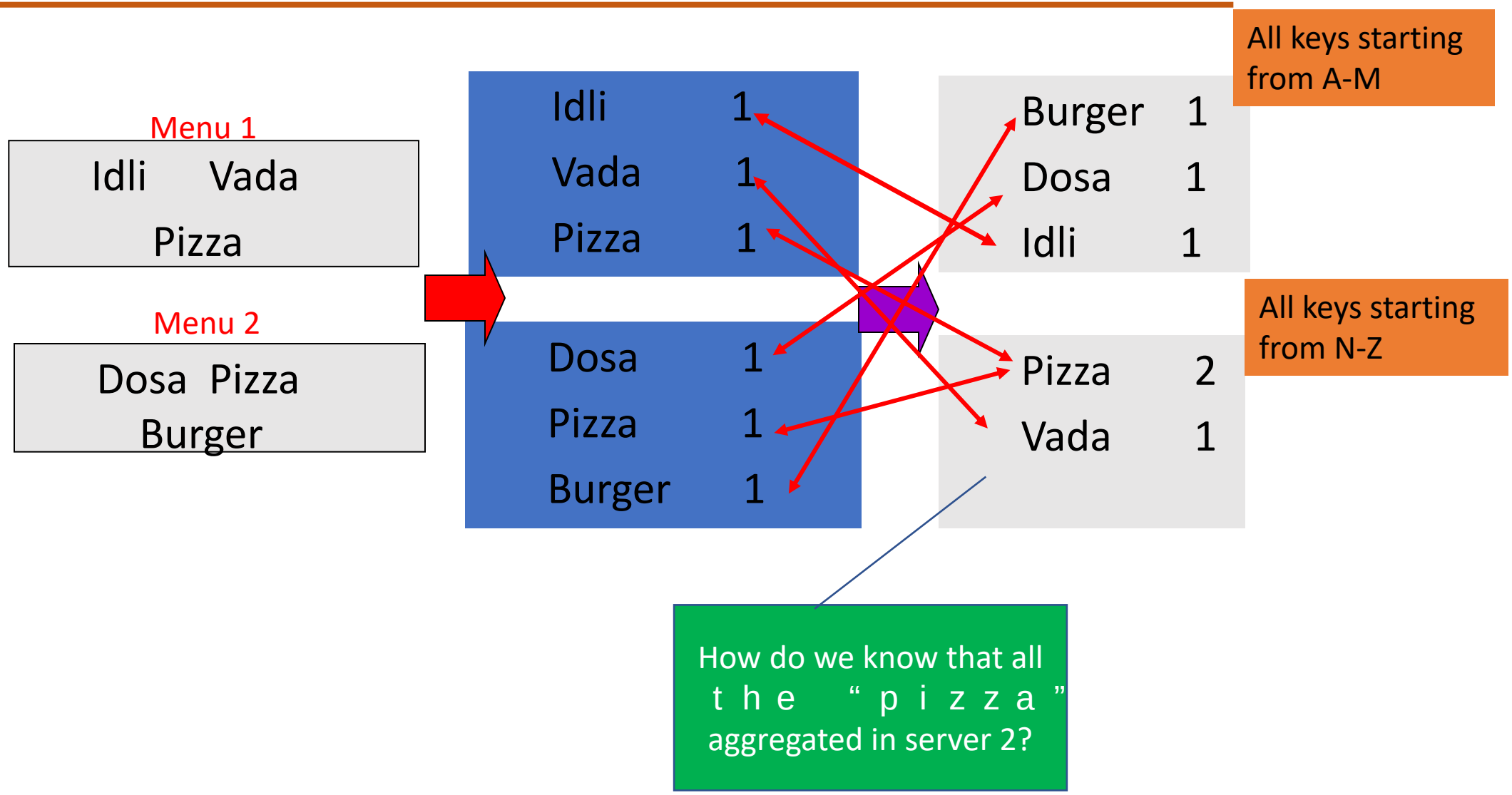
Example: find the number of restaurants offering each item?



- Treat output of first program as a key value pair
- Partition the keys between the second program

BIG DATA

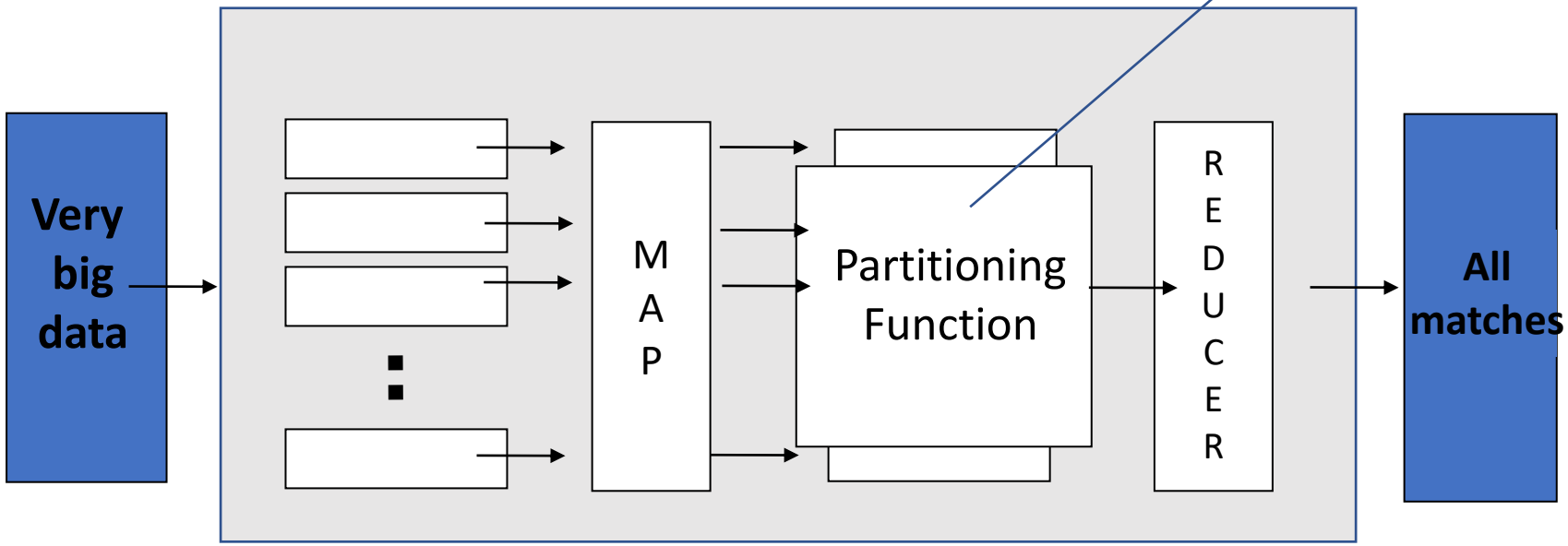
Distributed Word Count



BIG DATA

Map + Reduce

Ensures that all similar keys are aggregated at the same reducer. Each mapper has the same partition function



- Map:
  - Accepts *input* key/value pair
  - Emits *intermediate* key/value pair

- Reduce :
  - Accepts *intermediate* key/value* pair
  - Emits *output* key/value pair

Map Reduce: A look at the code

Map Reduce Programming model

- **Data type:** key-value *records*
- **Map function:**

$$(K_{in}, V_{in}) \rightarrow list(K_{inter}, V_{inter})$$

- **Reduce function:**

$$(K_{inter}, list(V_{inter})) \rightarrow list(K_{out}, V_{out})$$

BIG DATA

Map Reduce- Mapper for Word Count

```
public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{

   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(Object key, Text value, Context context
                   ) throws IOException, InterruptedException {
     StringTokenizer itr = new StringTokenizer(value.toString());
     while (itr.hasMoreTokens()) {
       word.set(itr.nextToken());
       context.write(word, one);
     }
   }
 }
```

(Key , value)

List (Key ,value)

$(K_{in}, V_{in}) \rightarrow list(K_{inter}, V_{inter})$

BIG DATA

Map Reduce -Reducer for Word Count

```
public static class IntSumReducer
      extends Reducer<Text,IntWritable,Text,IntWritable> {
   private IntWritable result = new IntWritable();

   public void reduce(Text key, Iterable<IntWritable> values,
                      Context context
                      ) throws IOException, InterruptedException {
     int sum = 0;
     for (IntWritable val : values) {
       sum += val.get();
     }
     result.set(sum);
     context.write(key, result);
   }
 }
```

Key ,List (value)

List (Key ,value)

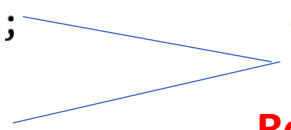
$(K_{inter}, list(V_{inter})) \rightarrow list(K_{out}, V_{out})$

BIG DATA

Map Reduce - Main Program for Word Count

```
public static void main(String[] args) throws Exception {
 Configuration conf = new Configuration();
 String[] otherArgs = new GenericOptionsParser(conf, args).
                              getRemainingArgs();
 if (otherArgs.length < 2) {
   System.err.println("Usage: wordcount <in> [<in>...] <out>");
   System.exit(2);
 }
 Job job = new Job(conf, "word count");
 job.setJarByClass(WordCount.class);
 job.setMapperClass(TokenizerMapper.class);

 job.setReducerClass(IntSumReducer.class);
 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(IntWritable.class);
 for (int i = 0; i < otherArgs.length - 1; ++i) {
   FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
 }
 FileOutputFormat.setOutputPath(job,
   new Path(otherArgs[otherArgs.length - 1]));
 System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```



Set Mapper and Reducer class

Map Reduce: Sample Exercise

Input: file(lineNumber, line) records and pattern

Output: lines matching a given pattern

What will be the mapper and reducer? What will be the keys?

Map:

Reduce:

Map Reduce , Search - Solution

Input: file (lineNumber, line) records and pattern

Output: lines matching a given pattern

Map:

```
if(line matches pattern):
        output(line)
```

Reduce: identity function

- Alternative: no reducer (map-only job)

Map Reduce , Functions in the Model

- **Map**
  - Process a key/value pair to generate intermediate key/value pairs
  - Sorts all key/value pairs before sending to reducer
- **Reduce**
  - Merge all intermediate values associated with the same key
  - Runs after all Map tasks are finished (why?)
- **Partition**
  - By default : **`hash(key) mod R`** (Well balanced)
  - There are cases where this can be more complex

Map Reduce: Revision exercise

Map Reduce, Sort

Input: (key, value) records

Output: same records, sorted by key

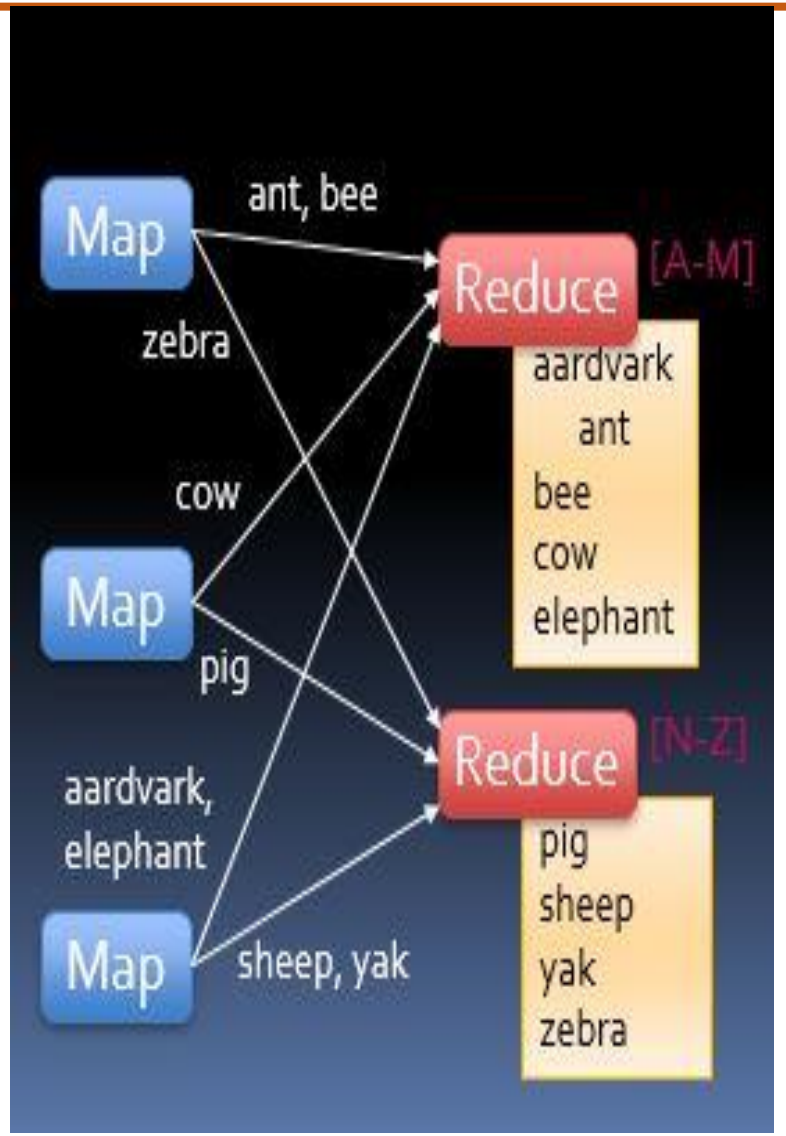
What will be the mapper and reducer?
What will be the partition function?

Map:

Reduce:

Map Reduce, Sort - Solution

- **Input**: (key, value) records
- **Output**: same records, sorted by key
- **Map**: identity function
- **Reduce**: identity function
- **Trick**: Pick partitioning function *p* so that $k_1 < k_2 \Rightarrow p(k_1) < p(k_2)$
- Works because map sorts output keys.

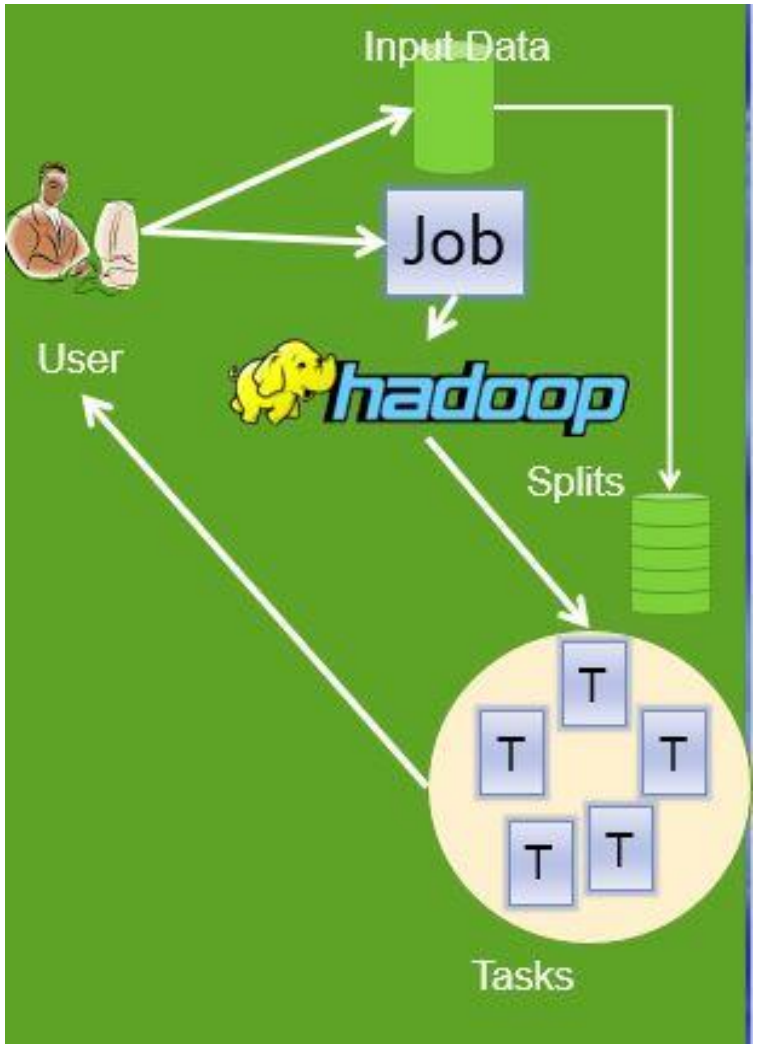


Map Reduce: Job Submission Flow

BIG DATA

Map Reduce , Hadoop Flow.

- User submits job
  - input data, MapReduce program, and configuration information
- How is parallelization achieved?
  - Divide input into smaller chunks –called **input splits**
- Hadoop divides jobs into tasks
  - Map tasks, reduce tasks
  - One map task per split
  - Tasks run in parallel



BIG DATA

MapReduce Flow for A SINGLE REDUCE Task

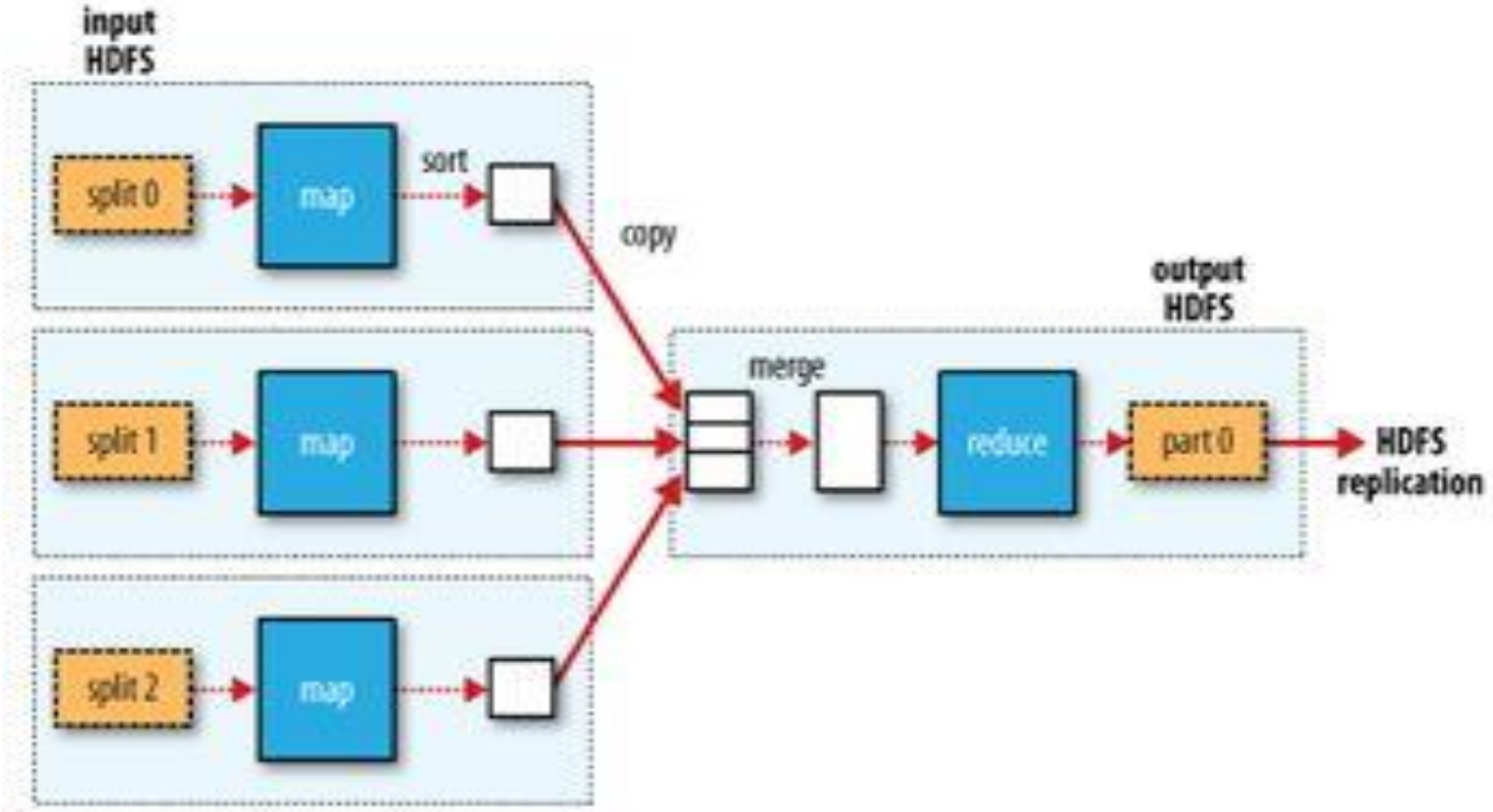


Figure: Map reduce data flow for single reducer task.

Map Reduce:
Exercise : Job Submission Flow with two Reducers

MapReduce Flow

- How will it work when there are two reducers?
- Where will the outputs be?

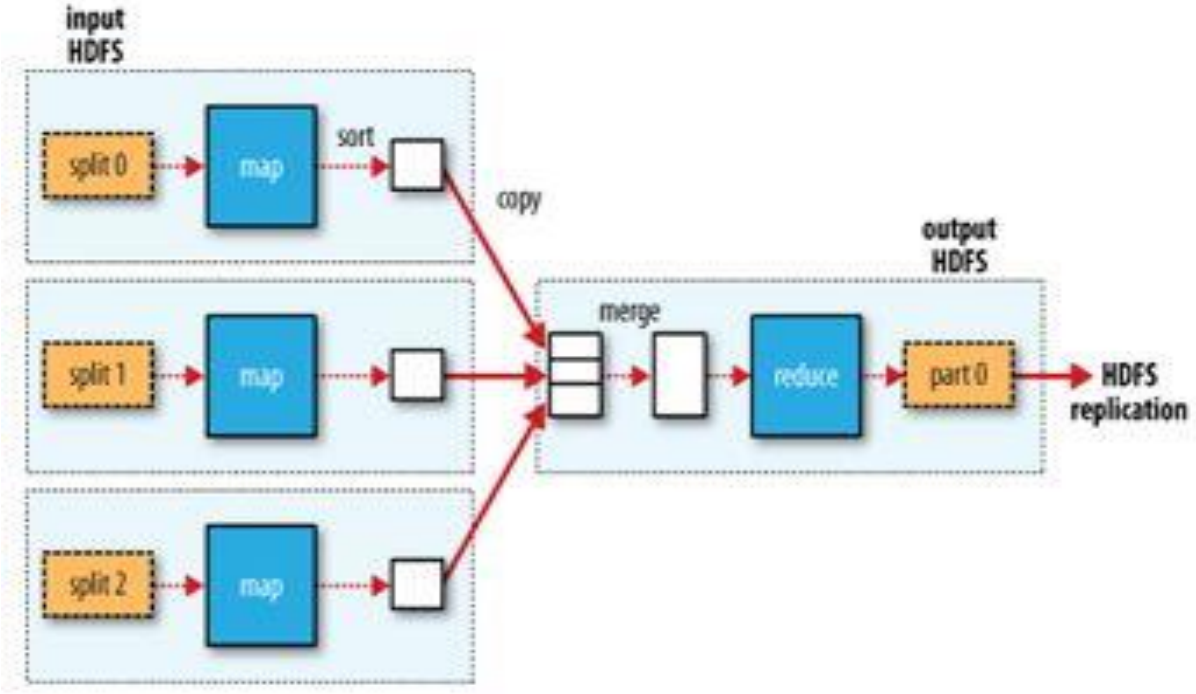
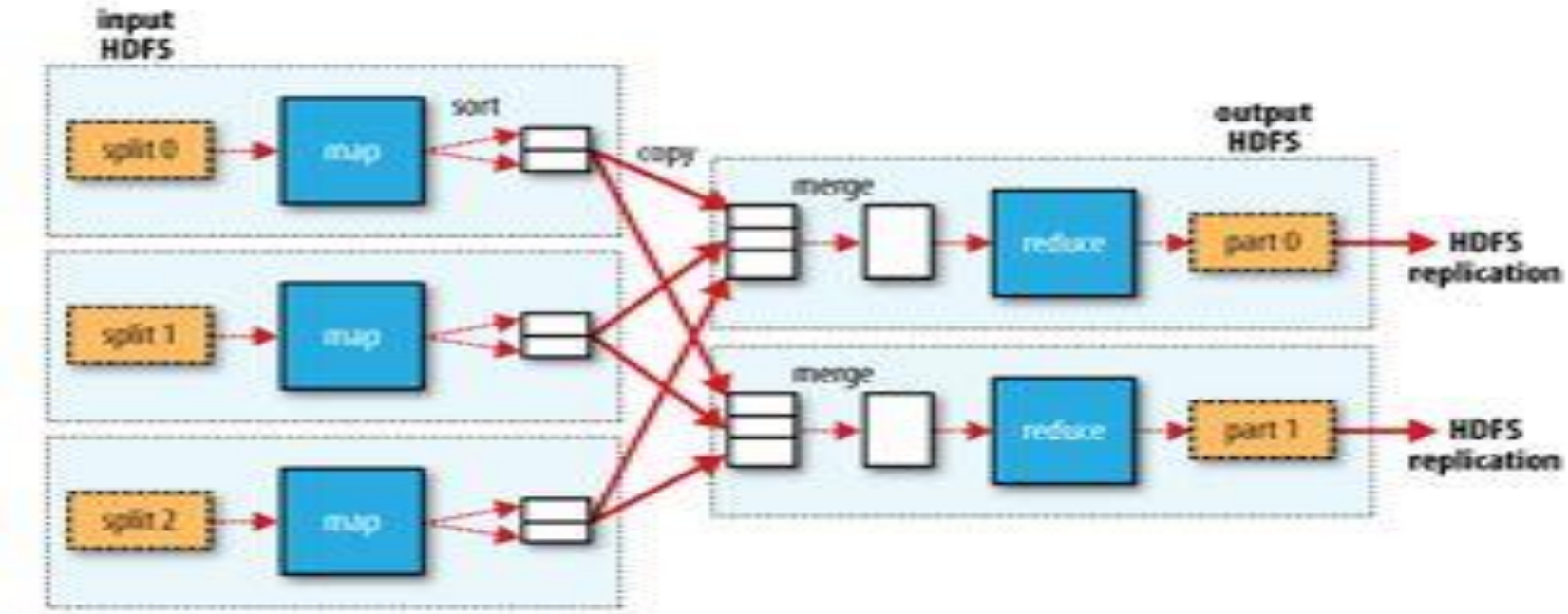


Figure: Map reduce data flow for single reducer task.

BIG DATA

MapReduce Flow for MULTIPLE REDUCE Tasks



- Outputs are created on two different nodes and have to be merged.
- But available through HDFS on any node

Map Reduce: Splits

Map Reduce Split size considerations

- Split size proportional to parallelism
- Small split size
  - Advantages
    - Large #splits
    - Increased parallelism
    - Increased load balancing
  - Disadvantages
    - the overhead of managing the splits and of map task creation
    - begins to dominate the total job execution time.

Optimal split size == size of HDFS block (128MB) (default) on Hadoop v2

Map Output

Split == Block size

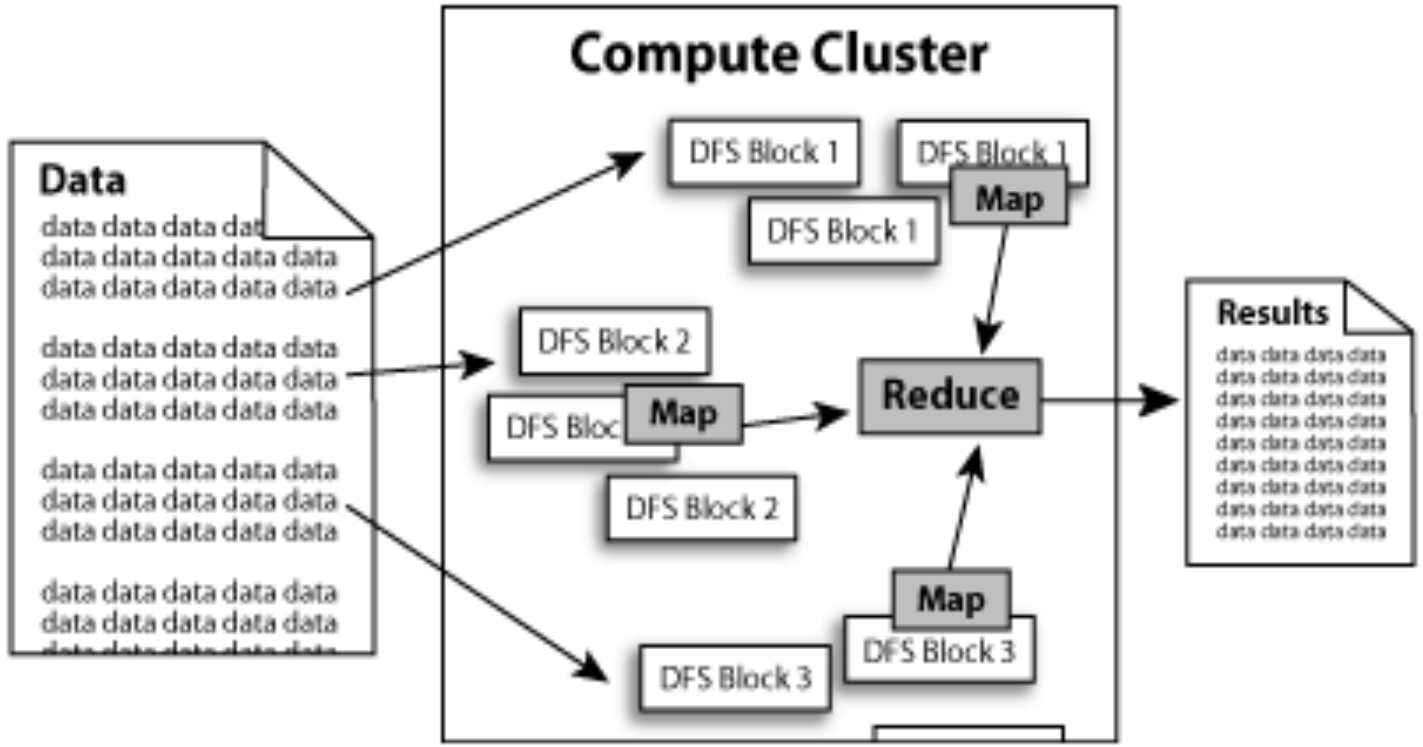
- All data required for Map
  - In the same node
  - No inter-node data transfer is required

Split != block size

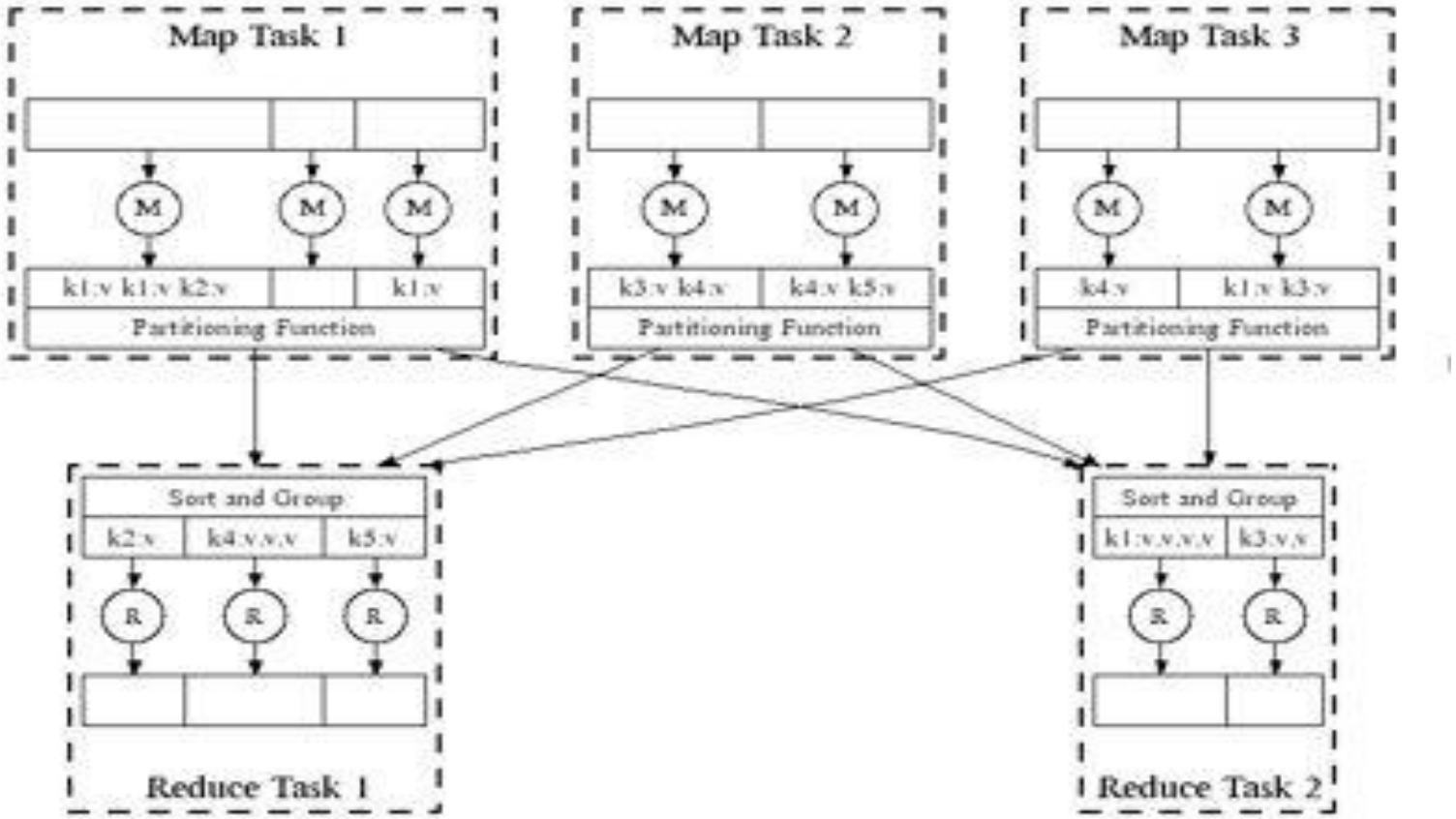
- Data transfer across multiple nodes
- Impacts performance

BIG DATA

Traditional: Move Data to Compute



Big Data: Move Compute to Data



Parallel Execution

Map Output

- **Where is Map output written to?**
  - Local disk and not HDFS
  - Why? Temporary output to be discarded after reduce.

- **Failure**
  - If the node running the map task fails
    - before the output has been consumed by reducer
  - Automatically rerun map task on another node

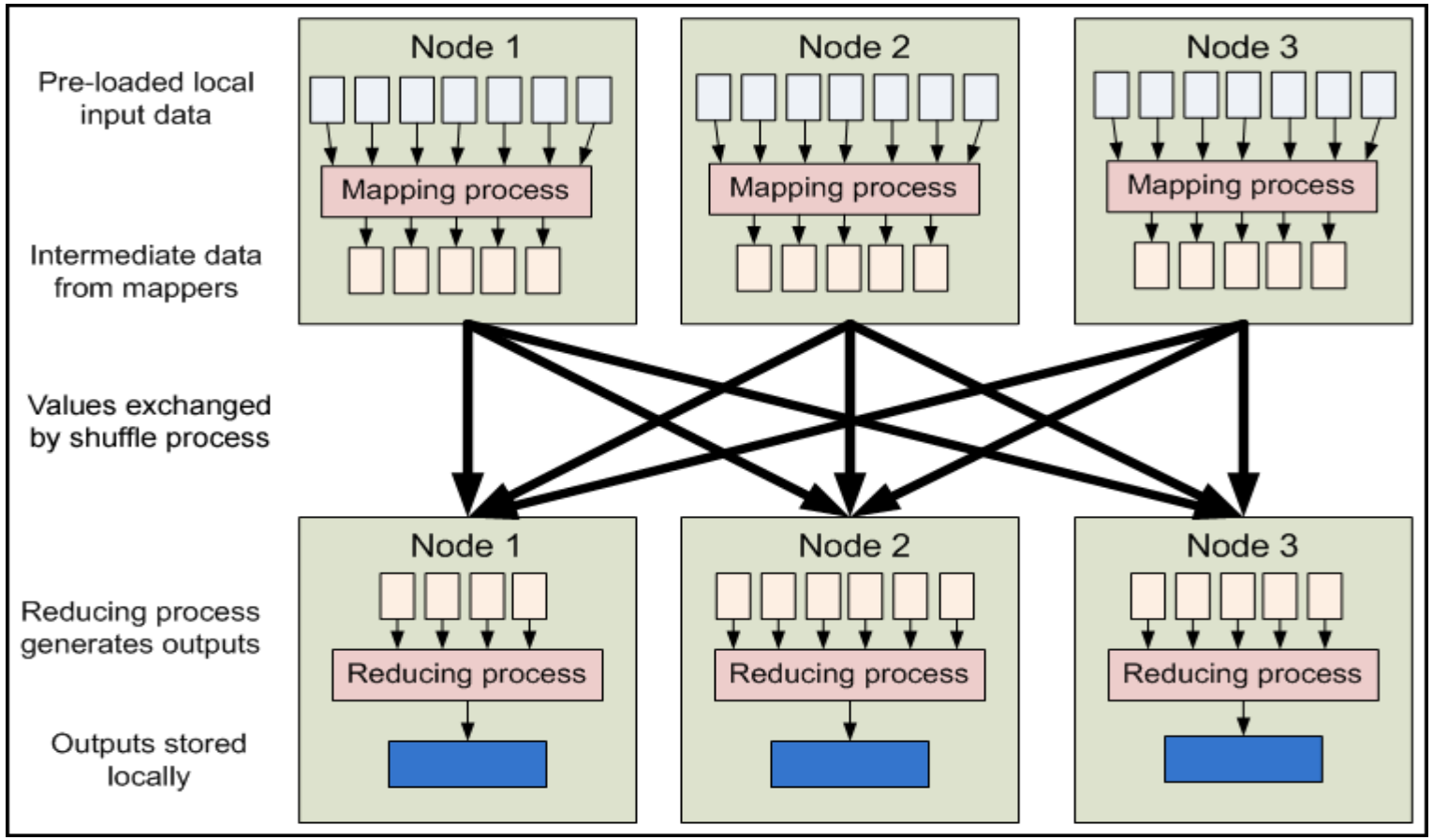
Reduce Tasks

- Reduce tasks don't have the advantage
  - the input to a single reduce task is normally the output from all mappers.
- Sorted map outputs
  - have to be transferred across the network
  - Where to?
    - To the node where the reduce task is running
    - Merge data from different mappers
    - Then passed to the user-defined reduce function.
- The output of the reduce is normally stored in HDFS for reliability.
- Where is the reduce output stored
  - 1 on the local node where the reduce happens
  - Other replicas on off-rack nodes.
  - Consumes network bandwidth

Map Reduce: Working

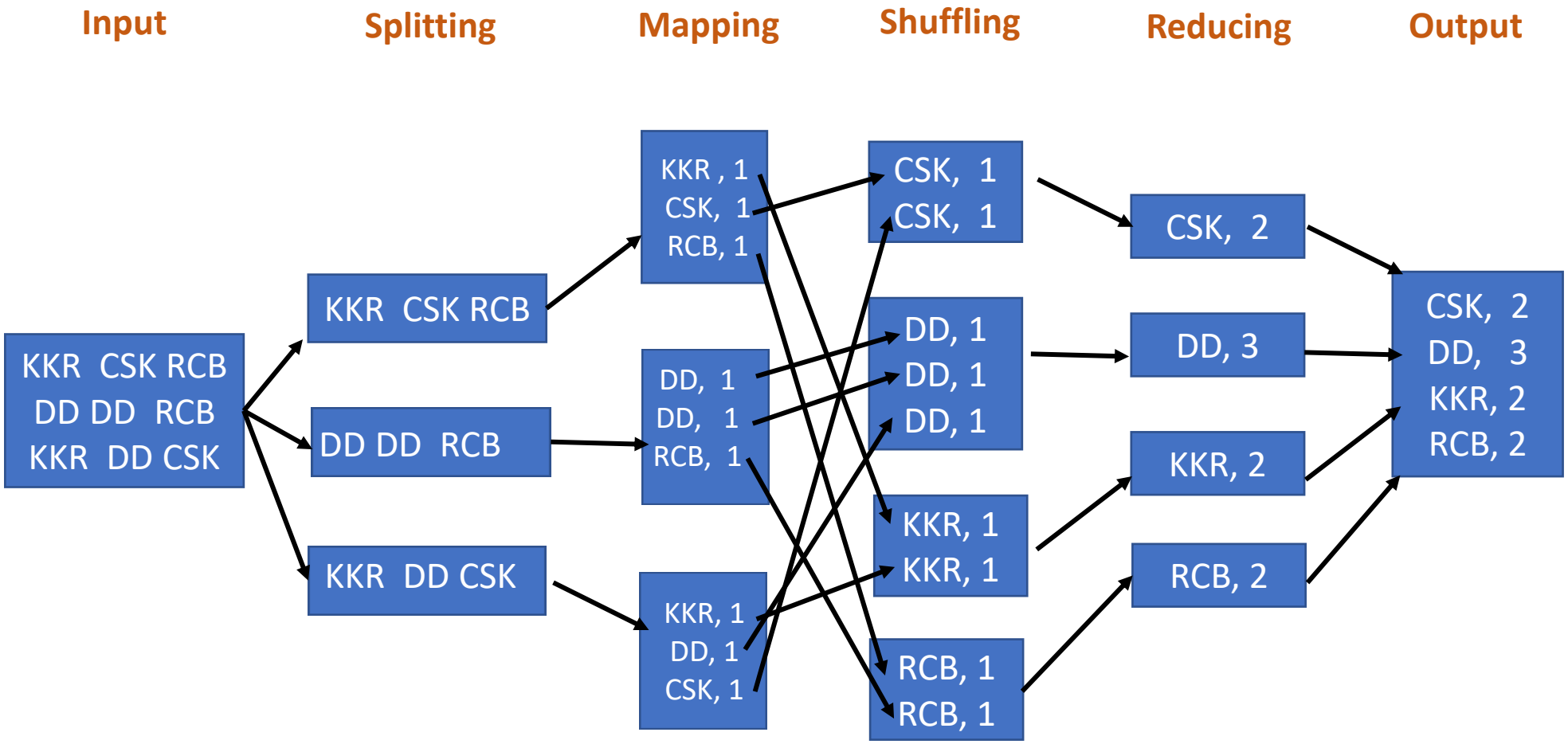
BIG DATA

High level view



BIG DATA

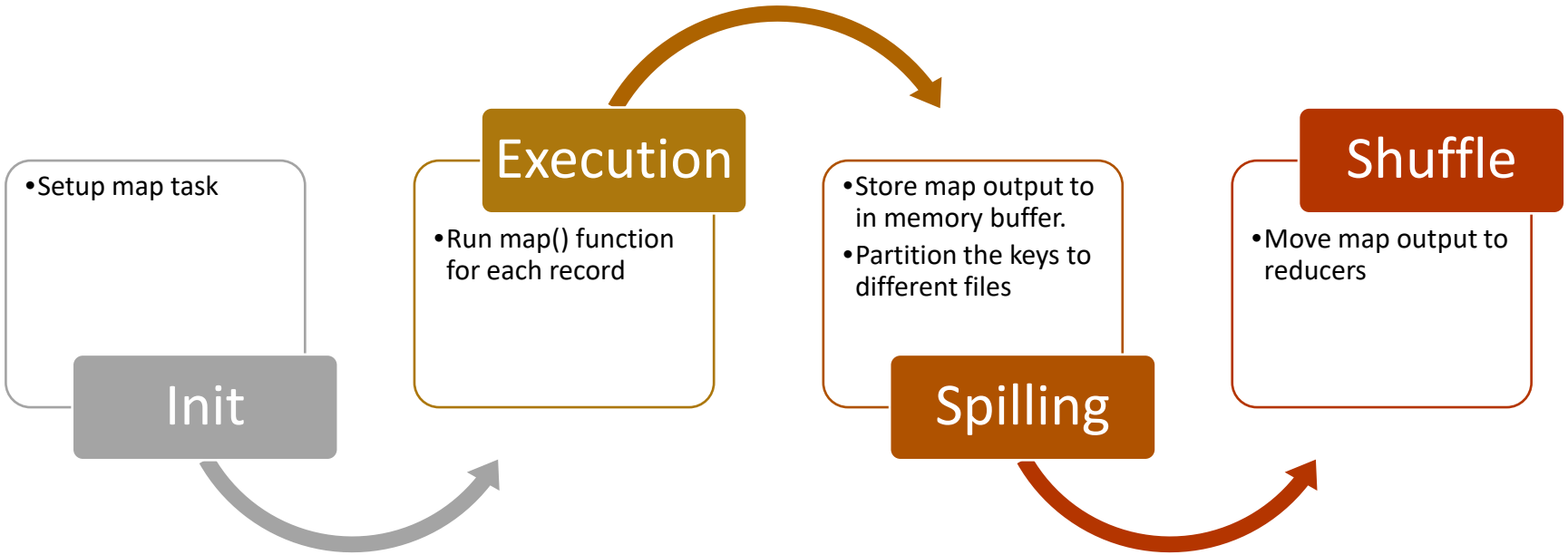
Shuffling - example

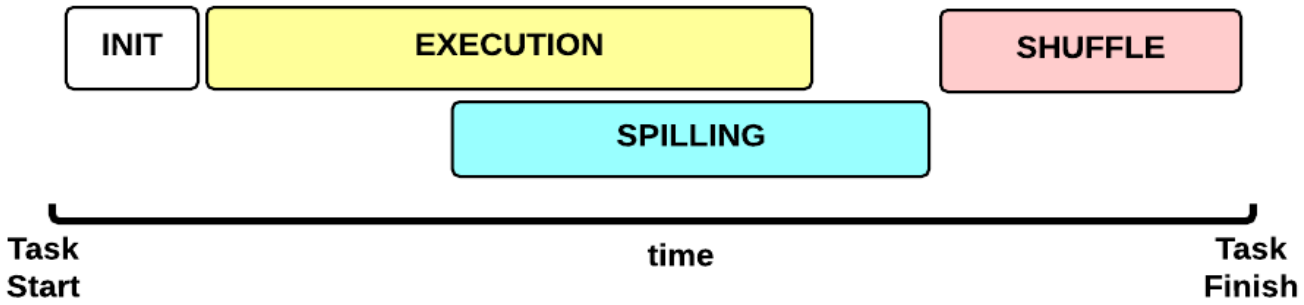


The overall Map-Reduce word count Process

BIG DATA

Closer look - Map

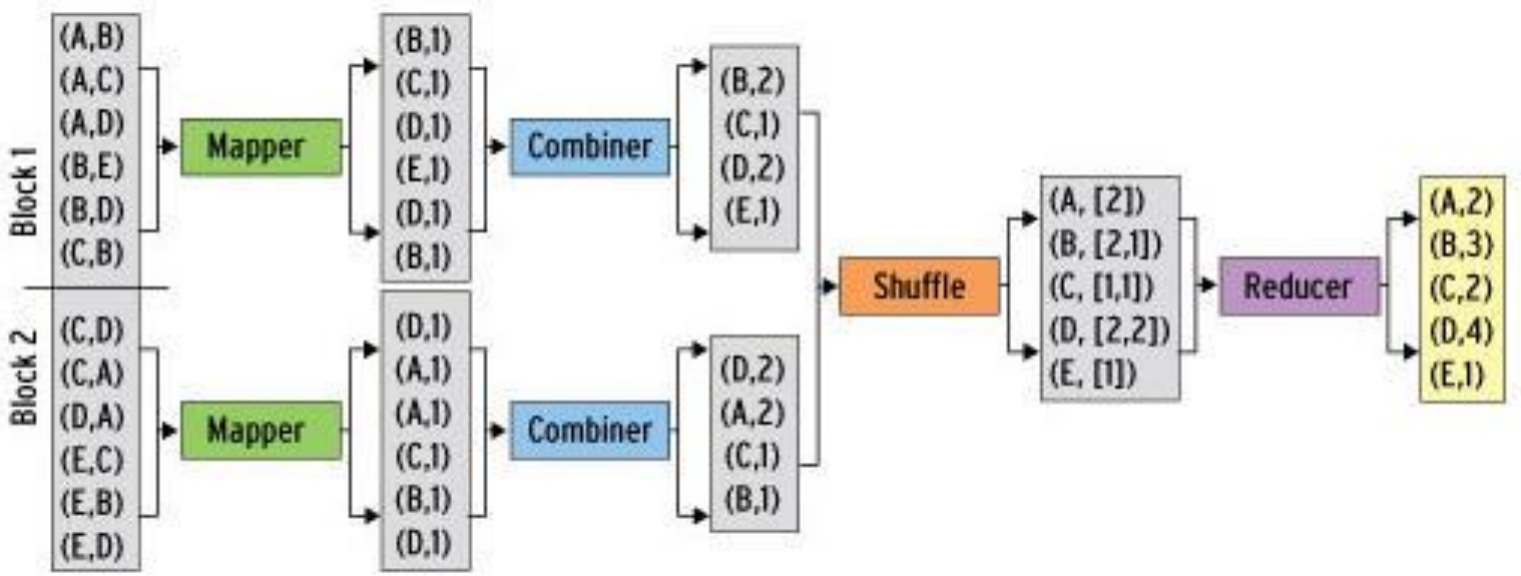




Map Reduce: Combiners

BIG DATA

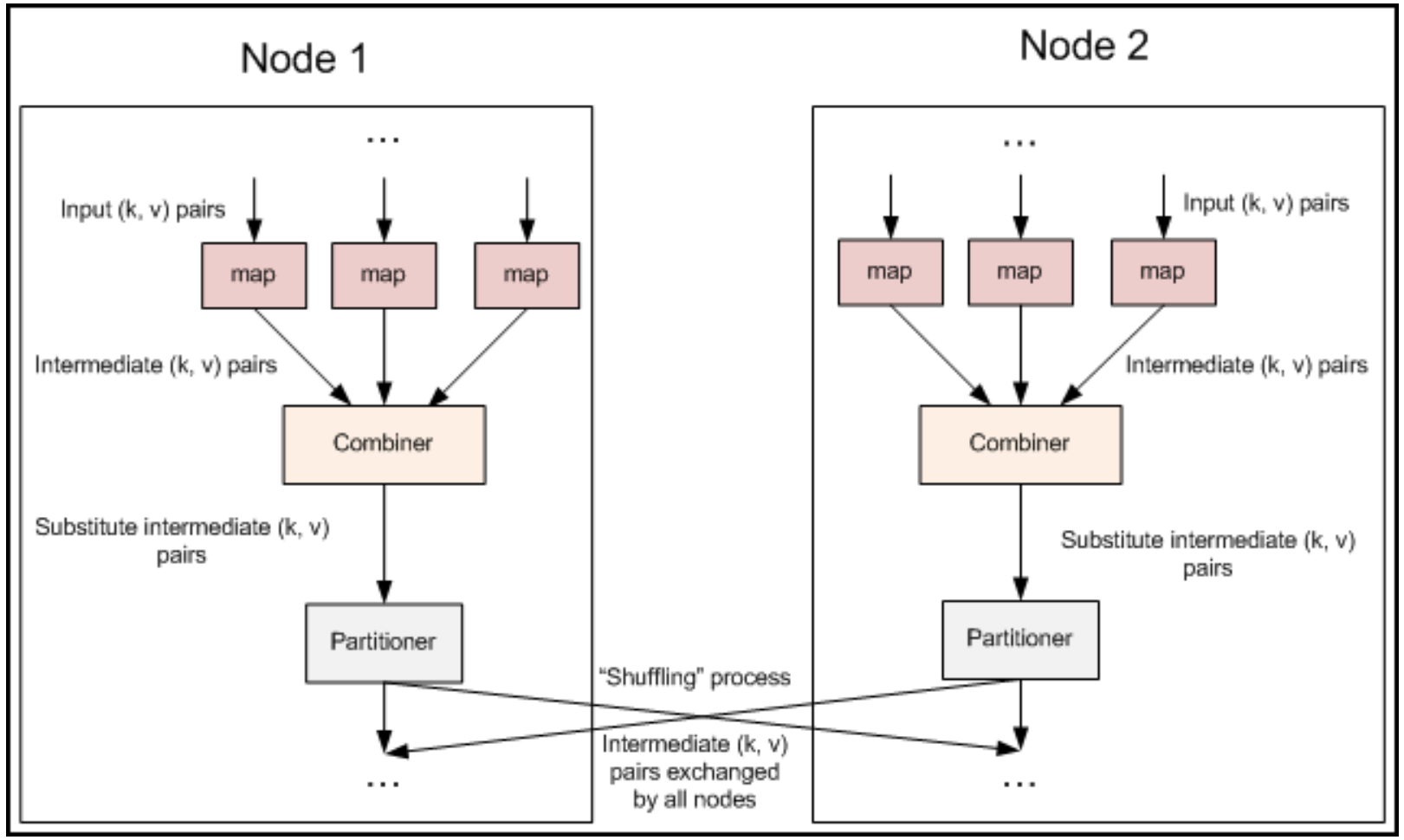
MapReduce –Combiners - motivation



- Combine multiple map outputs before doing a reduce
- Can write a combiner function in program
  - Combiner will be run before reduce
- Mini-reducer

BIG DATA

Combiner –when does it run?



BIG DATA

Map Reduce - Main Program for Word Count

```
public static void main(String[] args) throws Exception {
 Configuration conf = new Configuration();
 String[] otherArgs = new GenericOptionsParser(conf, args).
                           getRemainingArgs();
 if (otherArgs.length < 2) {
   System.err.println("Usage: wordcount <in> [<in>...] <out>");
   System.exit(2);
 }
 Job job = new Job(conf, "word count");
 job.setJarByClass(WordCount.class);
 job.setMapperClass(TokenizerMapper.class);
 job.setCombinerClass(IntSumReducer.class);
 job.setReducerClass(IntSumReducer.class);
 job.setOutputKeyClass(Text.class);
 job.setOutputValueClass(IntWritable.class);
 for (int i = 0; i < otherArgs.length - 1; ++i) {
   FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
 }
 FileOutputFormat.setOutputPath(job,
   new Path(otherArgs[otherArgs.length - 1]));
 System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Combiner is set here

Review Questions

Review Questions

- Questions from T1, LOR 2.4

Sample questions

- How many mappers and reducers will get started when trying to process a 230 MB file with Hadoop v2?
  - Ans: Block size = 128MB, so there will be two blocks. Assuming one block per split there will be 2 mappers
    - #reducers is configurable.
- Where is a combiner executed?
  - On the mapper.
- Write mappers and reducer pseudo code showing keys for counting #unique words in a file?
  - Similar to word count. Just that reducer does not have to write the count.

Additional Notes, References and Videos

Notes

- Chapter 2.4 from T1 –Rajkamal
- Tom whites book is an excellent reference for the programming component.

THANK YOU

K V Subramaniam
Dept. of Computer Science and Engineering

subramaniamkv@pes.edu