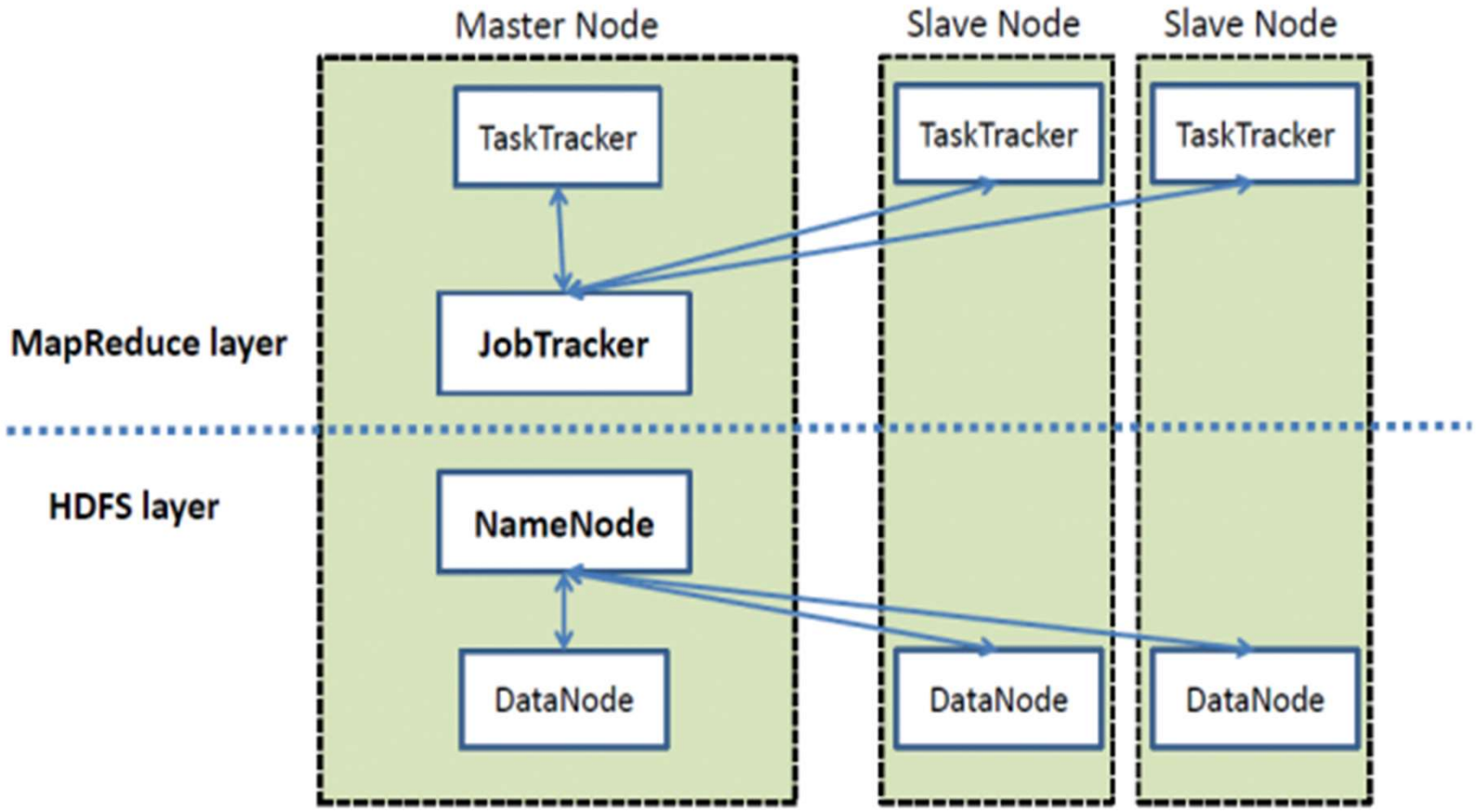
MapReduce

1. What is the need of MapReduce
2. Map, Reduce, Combiner Tasks
3. MapReduce Execution Flow
4. MapReduce algorithm for Word Count

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org
Big Data Analytics : Wiley Publications, Radha Shankarmani, M VijayaLaxmi
Guru99.com
https://tutorials.freshersnow.com/map-reduce-tutorial/key-value-pair-in-mapreduce/

High Level Architecture of Hadoop



Hadoop – Points to Remember

It is a **framework** for large scale distributed batch processing.

It efficiently distributes large amount of data for processing across a set of machines.

Scale-out approach and **easy to scale down** also.

Co-location of data and computation at data.

Small number of components with **well defined interfaces.**

Manages **inter-process communications**.

Handles machine **failures**..

Frees the developer from worries of race condition, data starvation, processing pipelines, data partitioning, code distribution so the focus on **application development and business logic.**

Hadoop Core Components

1. Hadoop Common Libraries
2. HDFS: It Stores data in terms of chunks (may be 64 or 128MB) across different machines.
3. YARN: Job Scheduling and Resource allocation

4. MapReduce: Google's Programming Model based on sort-merge based distributed computing. (search engine- indexing purpose)
   Now extensively used by other organizations like Yahoo, Amazon, IBM etc.

   New software stack, a high level programming system.

   It uses in functional programming language LISP that is naturally parallelizable across large cluster of workstations.

   It helps in parallel processing so that results are obtained in an efficient manner.

What is MapReduce?

- **MAPREDUCE** is a software framework helps in Parallel Processing.
  It is a programming model used for processing huge amounts of data.
- **MapReduce** program work in two phases, namely, Map and Reduce.
  Map tasks deal with **splitting** and **mapping** of data while
  Reduce tasks **shuffle** and **reduce** the data.
- Hadoop is capable of running MapReduce programs written in various languages: **Java, Ruby, Python, and C++.**
- MapReduce programs are parallel in nature, thus are very useful for performing large-scale data analysis using multiple machines in the cluster.
- The input to each phase is **key-value** pairs. In addition,
- Every programmer needs to specify two functions: **map function** and **reduce function**.
- The system manages parallel execution and coordination between tasks.

Programming Model: MapReduce

Warm-up task:

- We have a huge text document

- Count the number of times each distinct word appears in the file

- **Sample application:**
  - Analyze web server logs to find popular URLs

Task: Word Count

Case 1:

- File too large for memory, but all <word, count> pairs fit in memory.
  HashTable(index by word)

Case 2:

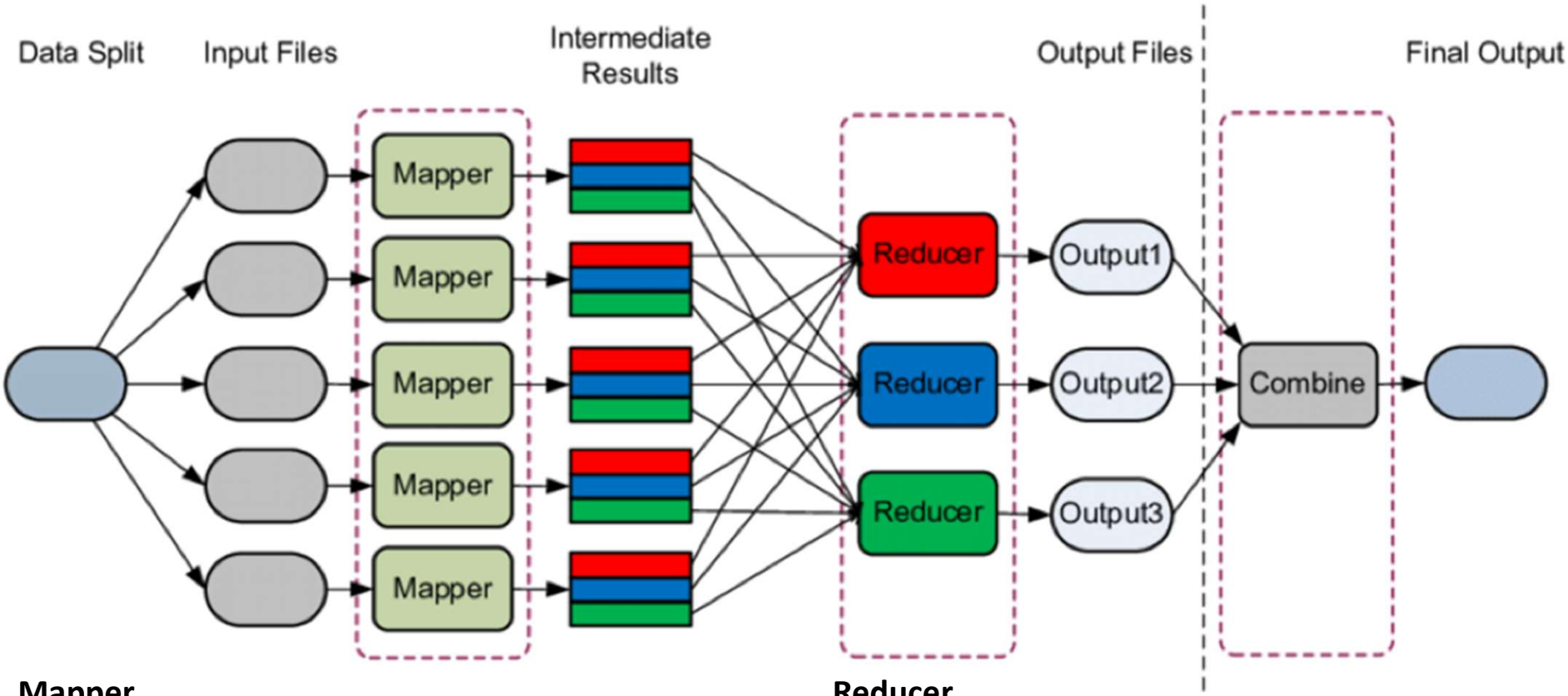
- Count occurrences of words:
  - **`words(doc.txt) | sort | uniq -c`**
    - where **`words`** takes a file and outputs the words in it, one per a line

- Case 2 captures the essence of **MapReduce**
  - Great thing is that it is naturally parallelizable

MapReduce: Overview

- Sequentially read a lot of data
- **Map:**
  - Scans input file record at a time
  - Extract something you care about (i.e. words- keys)
- **Group by key:** Sort and Shuffle on the basis of keys.
- **Reduce: (uniq –c)**
  - Aggregate, summarize, filter or transform
- Write the result

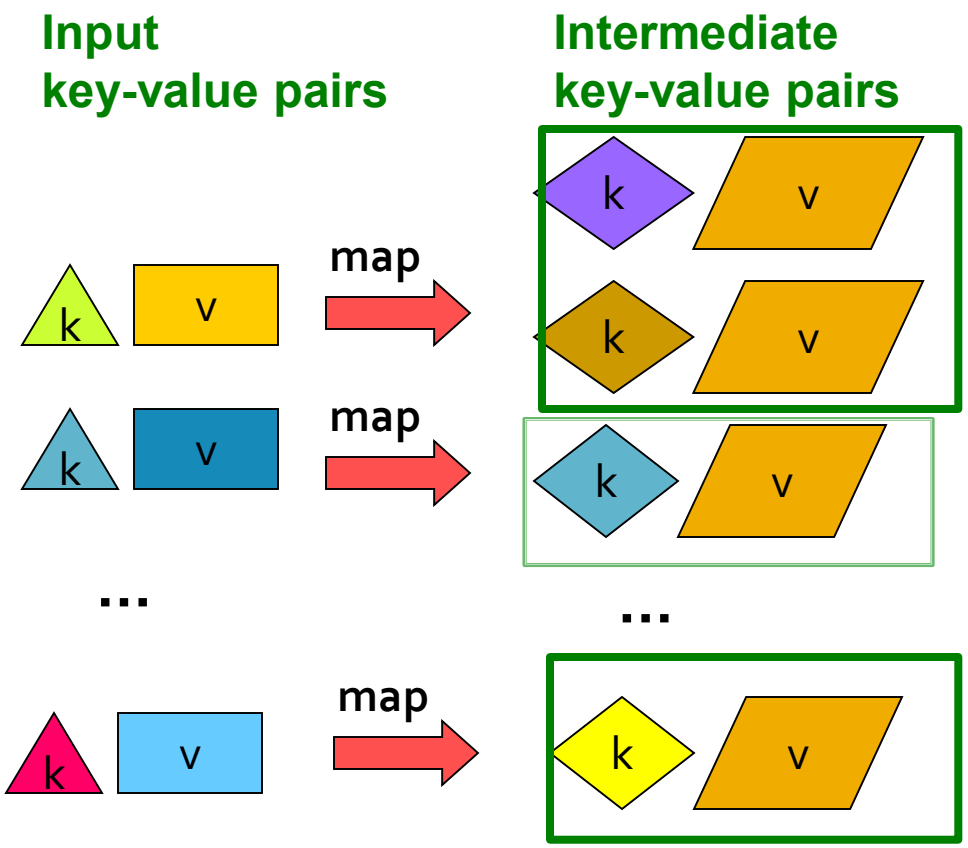
Outline stays the same, **Map** and **Reduce** change to fit the problem



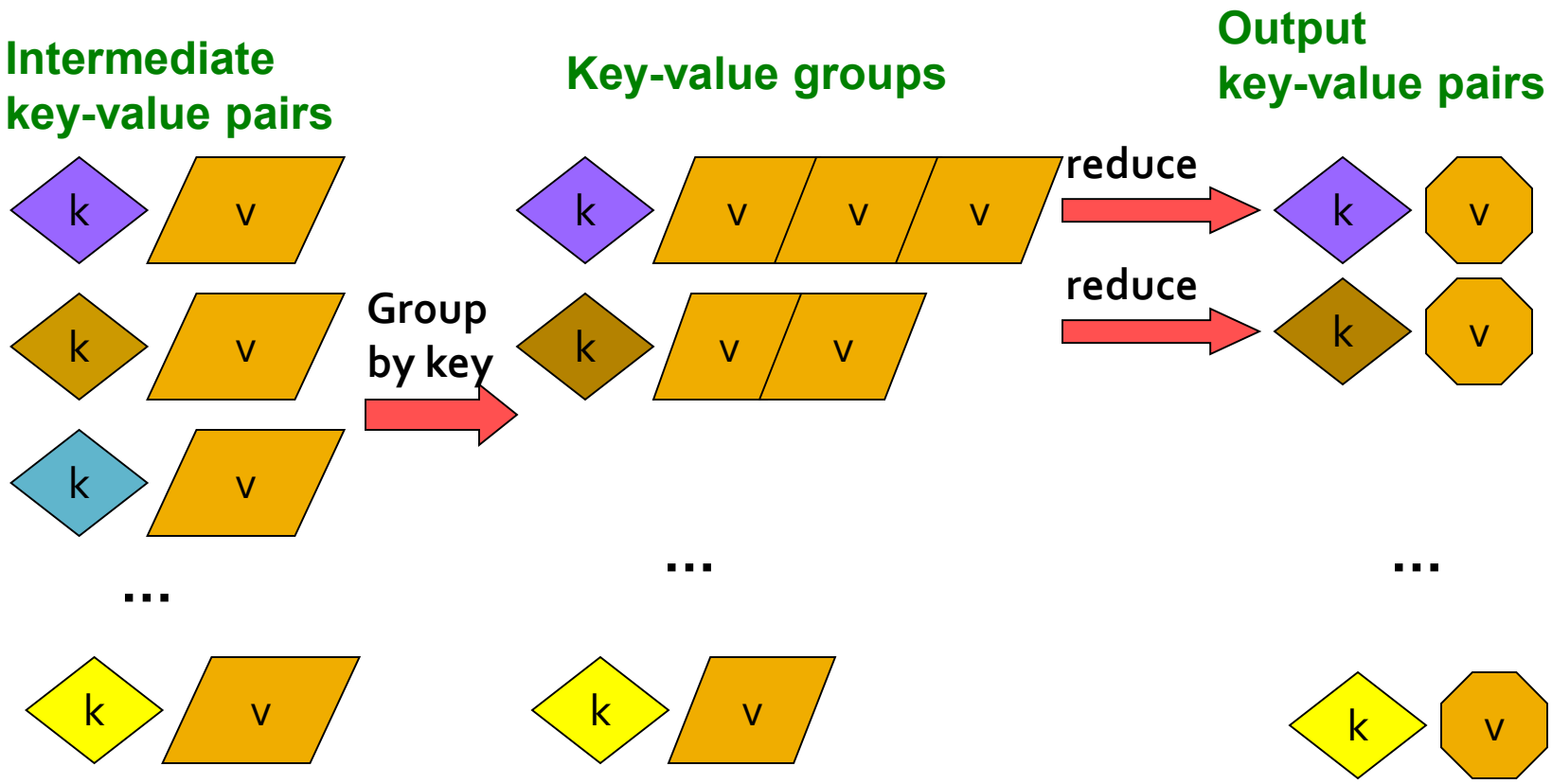
Mapper
complex logic/business rules/costly code.

Reducer
light-weight processing like aggregation/summation.

MapReduce: The Map Step



MapReduce: The Reduce Step



WordCount Exercise

COMBINER (Optional)

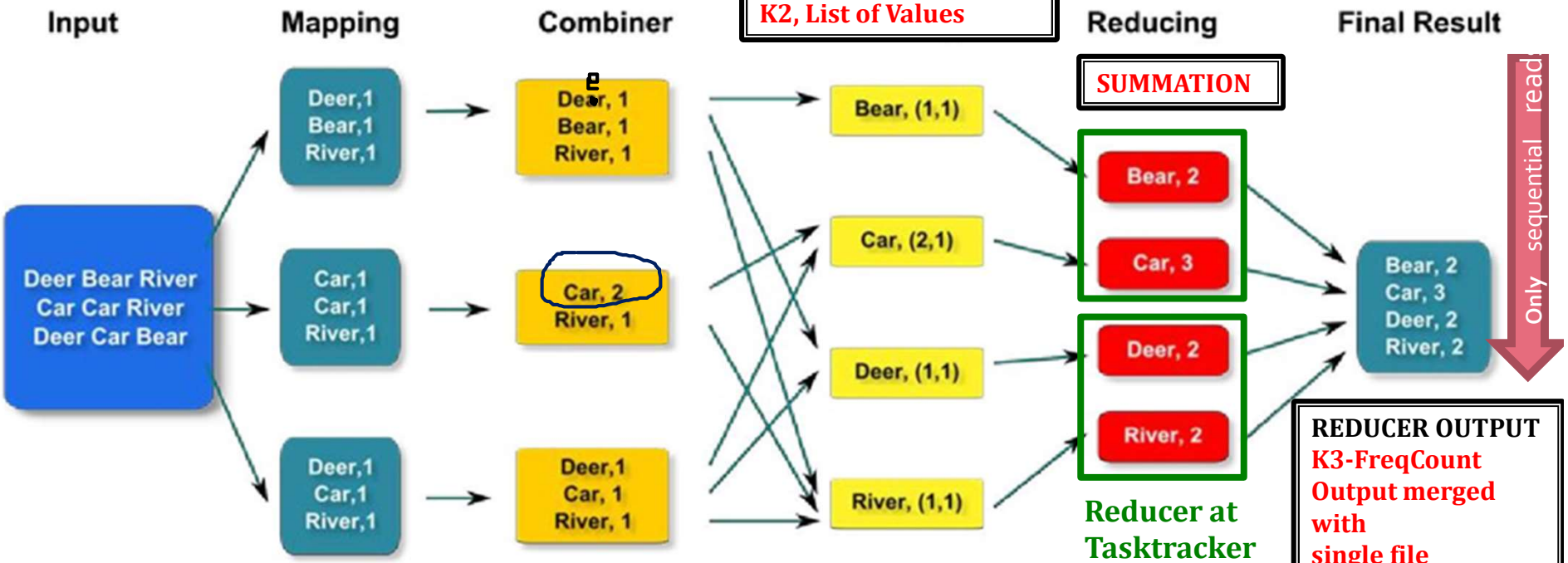
1. Some of the tasks of reducer can be assigned to combiner
2. Instead of sending all values to reducer, some value can be calculated at mapper side for same keys
3. This reduces INPUT OUTPUT operations between Mapper and Reducer.

Line or tuple or Document.
K1= name of doc
V1 = block-line

MAPPER+COMBINER OUTPUT
K2=Deer V2=1
KEY NEED NOT BE UNIQUE

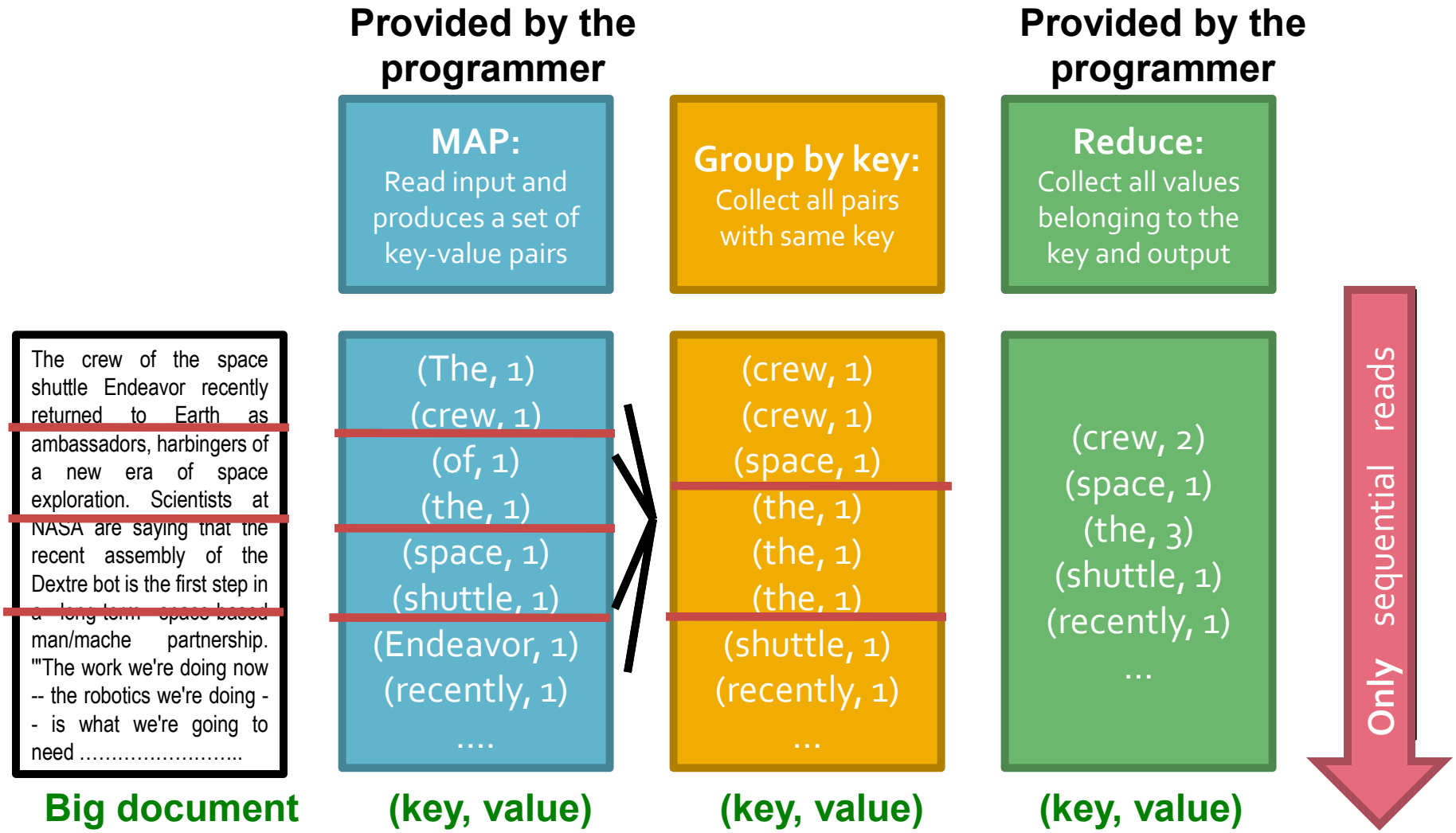
Master Controller

SHUFFLER OUTPUT = REDUCER INPUT
Group By KEY K2
K2, List of Values



Refernece :https://tutorials.freshersnow.com/map-reduce-tutorial/combiner-in-hadoop-mapreduce/

MapReduce: Word Counting



Word Count Using MapReduce

```
map(key, value):
// key: document name; value: text of the document
   for each word w in value:
      emit(w, 1)
```

```
reduce(key, values):
// key: a word; count: an iterator over values
      result = 0
      for each count v in values:
            result += v
      emit(key, result)
```

Map-Reduce: Environment

Map-Reduce environment takes care of:

- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the **group by key** step
- Handling machine failures
- Managing required inter-machine communication

Data Flow

- **Input and final output are stored on a distributed file system (FS):**
  - Scheduler tries to schedule map tasks "close" to physical storage location of input data
- **Intermediate results are stored on local FS of Map and Reduce workers**
- **Output is often input to another MapReduce task**

Coordination: Master

- **Master node takes care of coordination:**

  Task status: (idle, in-progress, completed)

  - **Idle tasks** get scheduled as workers become available

  - When a map task **completes**, it sends the master the location and sizes of its *R* intermediate files, one for each reducer

  - Master **pushes** this info to reducers

- Master pings workers periodically to detect failures

Dealing with Failures

1. Client submits the Job to JobTracker node (Master)
2. JobTracker keeps track of Map and Reduce tasks at TaskTracker nodes. (Worker)

- **Master failure –** Only one JobTracker per cluster that runs in its own JVM. All the slave nodes are configured with JobTracker. So if Master fails then all the tasks at slave are halted. The whole Map-Reduce job is to be restarted.
  - MapReduce task is aborted and client is notified

- **Map worker failure**
  - Map tasks completed or in-progress at worker are reset to idle
  - Reduce workers are notified when task is rescheduled on another worker

- **Reduce worker failure**
  - Only in-progress tasks are reset to idle
  - Reduce task is restarted

How many Map and Reduce jobs?

- *M* map tasks, *R* reduce tasks
- **Rule of a thumb:**
  - Make *M* much larger than the number of nodes in the cluster
  - One DFS chunk per map is common
  - Improves dynamic load balancing and speeds up recovery from worker failures
- **Usually *R* is smaller than *M***
  - Because output is spread across *R* files

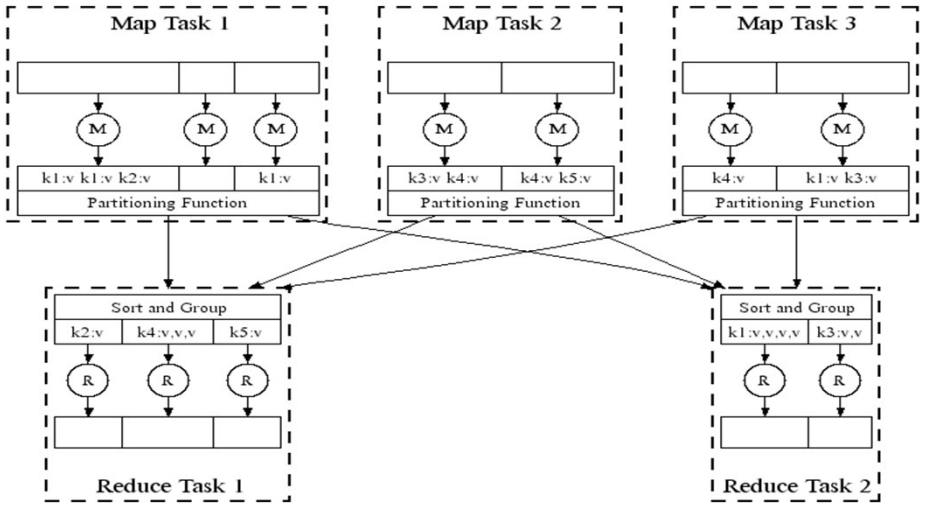
Refinements: Backup Tasks

- **Problem**
  - Slow workers significantly lengthen the job completion time:
    - Other jobs on the machine
    - Bad disks
    - Weird things
- **Solution**
  - Near end of phase, spawn backup copies of tasks
    - Whichever one finishes first "wins"
- **Effect**
  - Dramatically shortens job completion time

Refinement: Combiners

- Often a Map task will produce many pairs of the form $(k,v_1), (k,v_2), \ldots$ for the same key k
  - E.g., popular words in the word count example

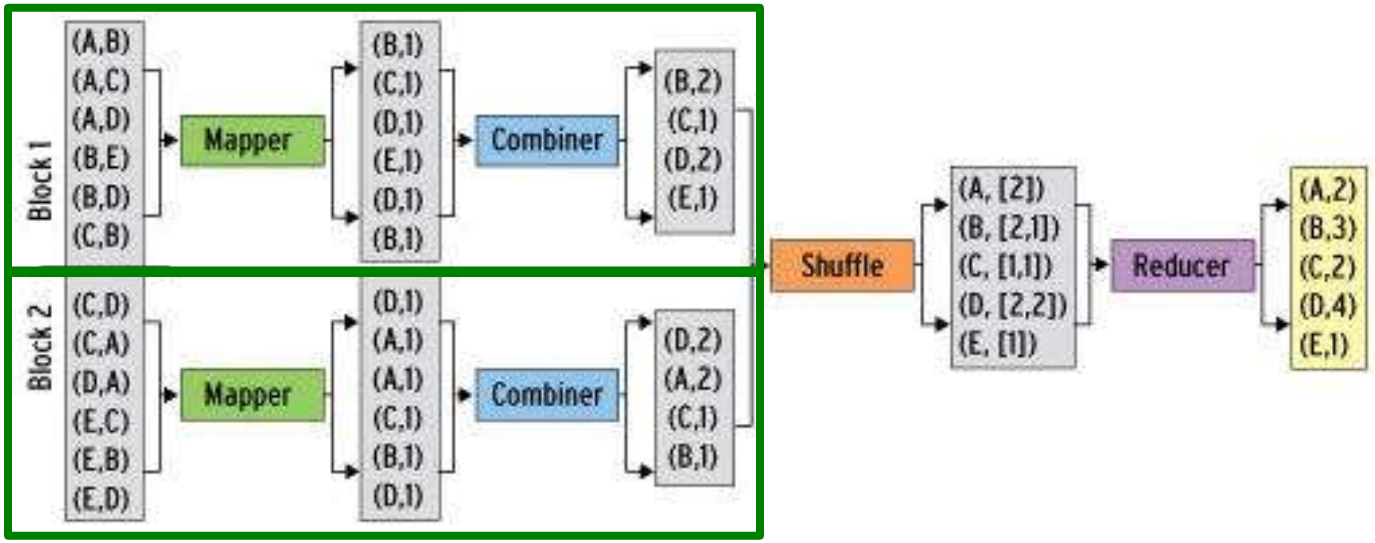
- **Can save network time by pre-aggregating values in the mapper:**
  - combine(k, list(v_1)) → v_2
  - Combiner is usually same as the reduce function



- Works only if reduce function is commutative and associative

Refinement: Combiners

- **Back to our word counting example:**
  - Combiner combines the values of all keys of a single mapper (single machine):



  - Much less data needs to be copied and shuffled!

Refinement: Partition Function

- **Want to control how keys get partitioned**
  - Inputs to map tasks are created by contiguous splits of input file
  - Reduce needs to ensure that records with the same intermediate key end up at the same worker
- **System uses a default partition function:**
  - **hash(key) mod *R***
- **Sometimes useful to override the hash function:**
  - E.g., **hash(hostname(URL)) mod *R*** ensures URLs from a host end up in the same output file

Problems Suited for Map-Reduce

Example: Host size

- **Suppose we have a large web corpus**
- Look at the metadata file
  - Lines of the form: (URL, size, date, …)
- **For each host, find the total number of bytes**
  - That is, the sum of the page sizes for all URLs from that particular host
- **Other examples:**
  - Link analysis and graph processing
  - Machine Learning algorithms

Example: Language Model

- **Statistical machine translation:**
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents

- **Very easy with MapReduce:**
  - **Map:**
    - Extract (5-word sequence, count) from document
  - **Reduce:**
    - Combine the counts