CSE 424
Big Data

MapReduce

Slides 6

Instructor: Asst. Prof. Dr. Hüseyin ABACI

Outline

- MapReduce patterns
- MapReduce dataflow
- MapReduce job run, submission, job initialization, task assignment, task execution
- MapReduce examples
- Numerical summarization, count and topN calculation

Bahga, Arshdeep, and Vijay Madisetti. Big data science & analytics: A hands-on approach. VPT, 2016.
Tom White. Hadoop: The definitive guide. O'Reilly Media, May 2009.

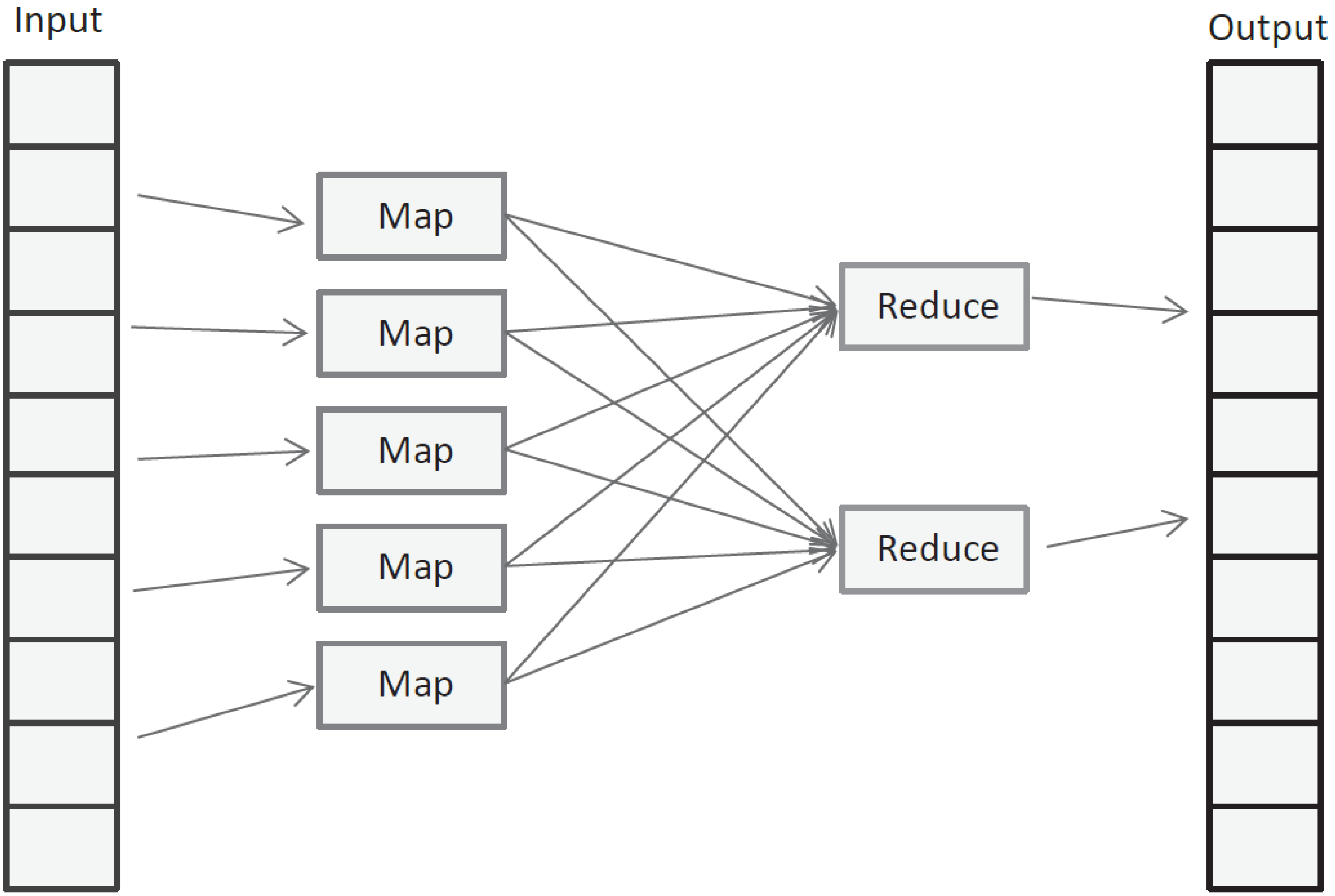
MapReduce Patterns

- MapReduce programming model for processing data on large clusters was originally proposed by Dean and Ghemawat .
- MapReduce allows the developers to focus on developing data-intensive applications without having to worry about issues such as input splitting, scheduling, load balancing and failover.
- The MapReduce run-time systems take care of tasks such partitioning the data, scheduling of jobs and communication between nodes in the cluster.
- MapReduce model has two phases: Map and Reduce.
- In the Map phase, data is read from a distributed file system, partitioned among a set of computing nodes in the cluster, and sent to the nodes as a set of key-value pairs.
- The Map tasks process the input records independently of each other and produce intermediate results as key-value pairs.
- The intermediate results are stored on the local disk of the node running the Map task.

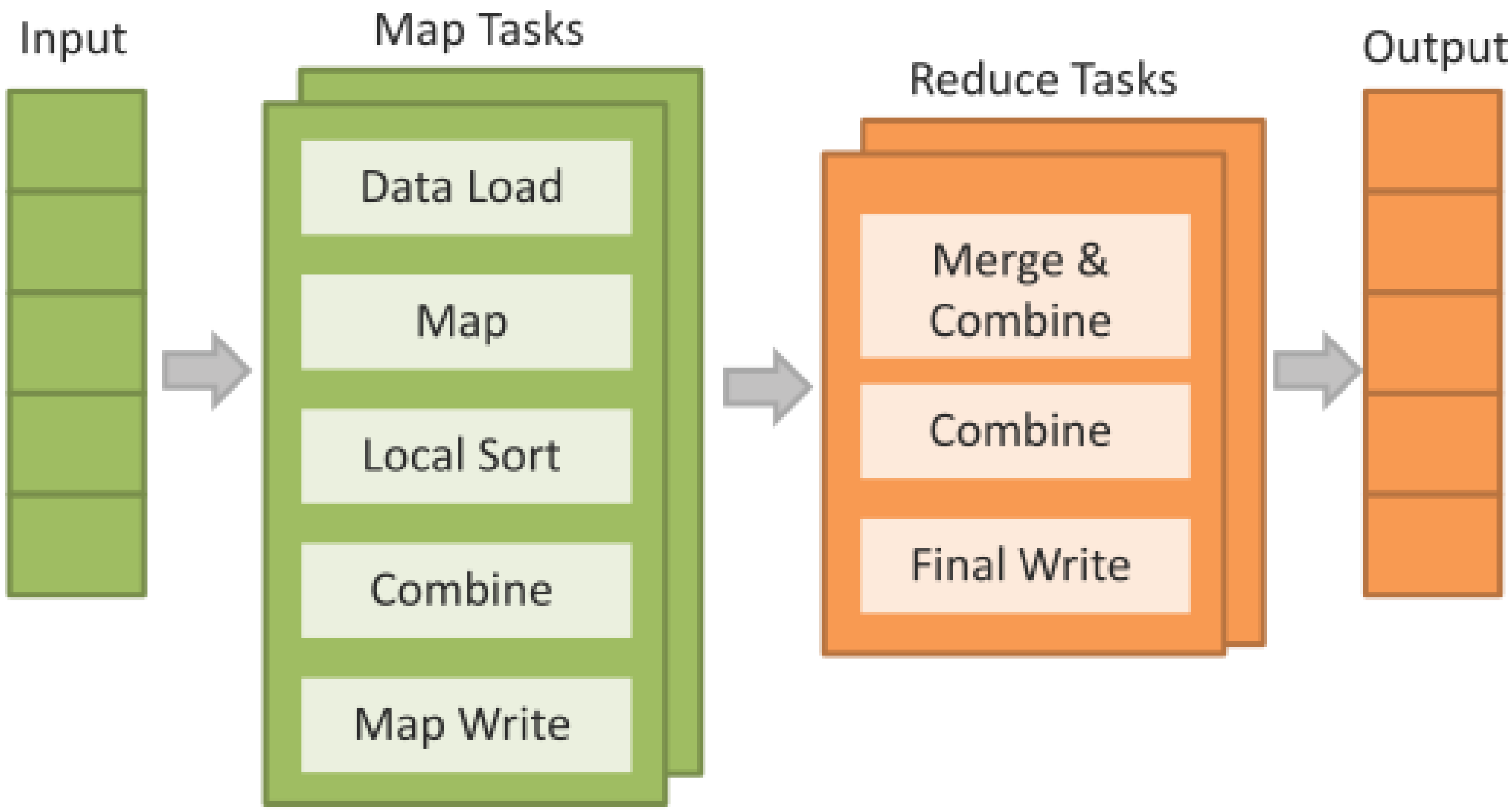
MapReduce Patterns (cont’d)

- When all the Map tasks are completed, the Reduce phase begins with the shuffle and sort step, in which the intermediate data is sorted by the key and the key-value pairs are grouped and shuffled to the reduce tasks.
- The reduce tasks then take the key-value pairs grouped by the key and run the reduce function for each group of key-value pairs.
- The data processing logic in reduce function depends on the analysis task (you need to write) to be accomplished.
- An optional Combine task can be used to perform data aggregation on the intermediate data of the same key for the output of the mapper before transferring the output to the Reduce task.
- The Python implementations use the MRJob Python library which lets you write MapReduce jobs in Python and run them on several platforms including local machine, Hadoop cluster and Amazon Elastic MapReduce (EMR).

MapReduce Patterns (cont'd)

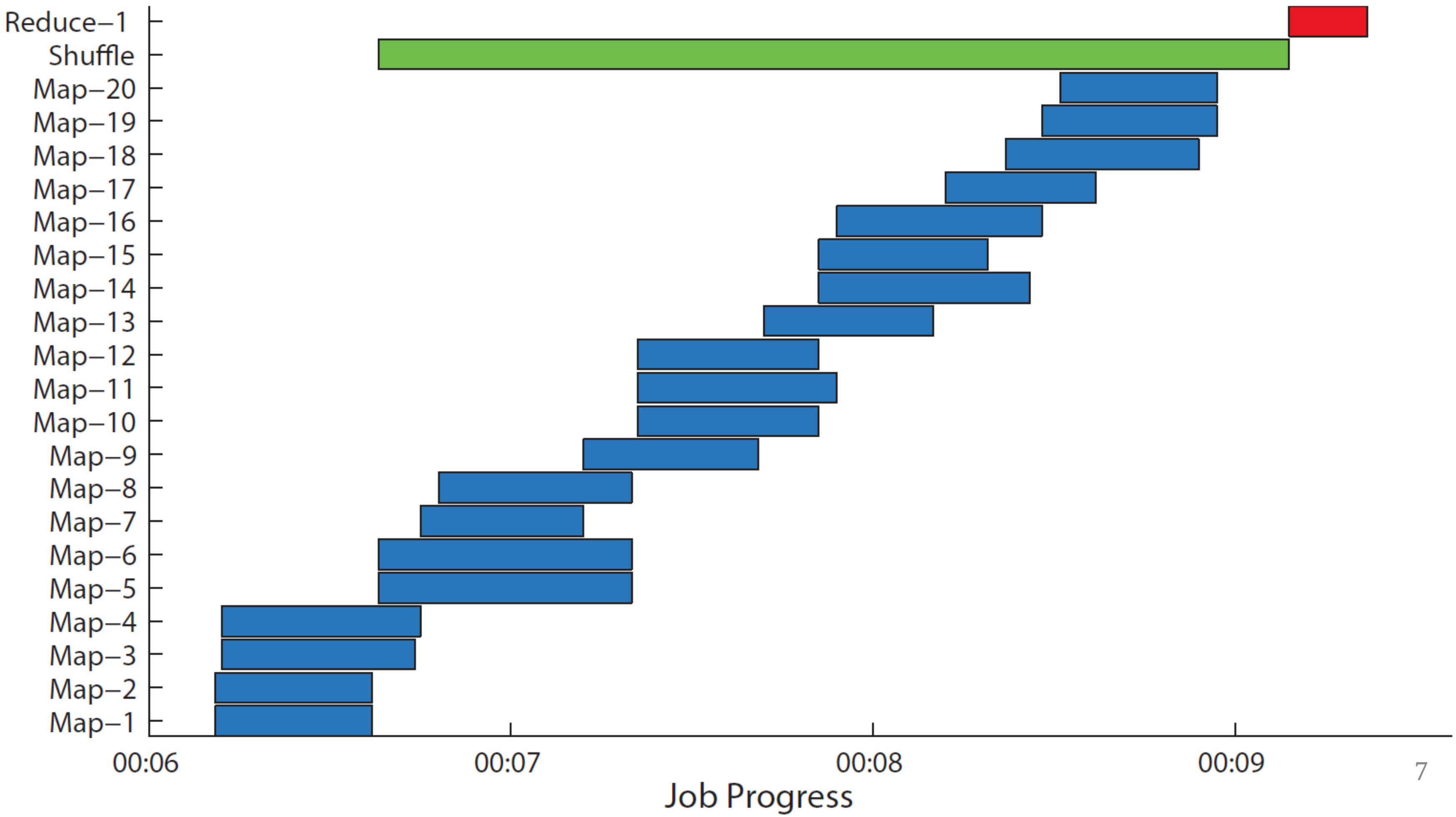


MapReduce Patterns (cont'd)

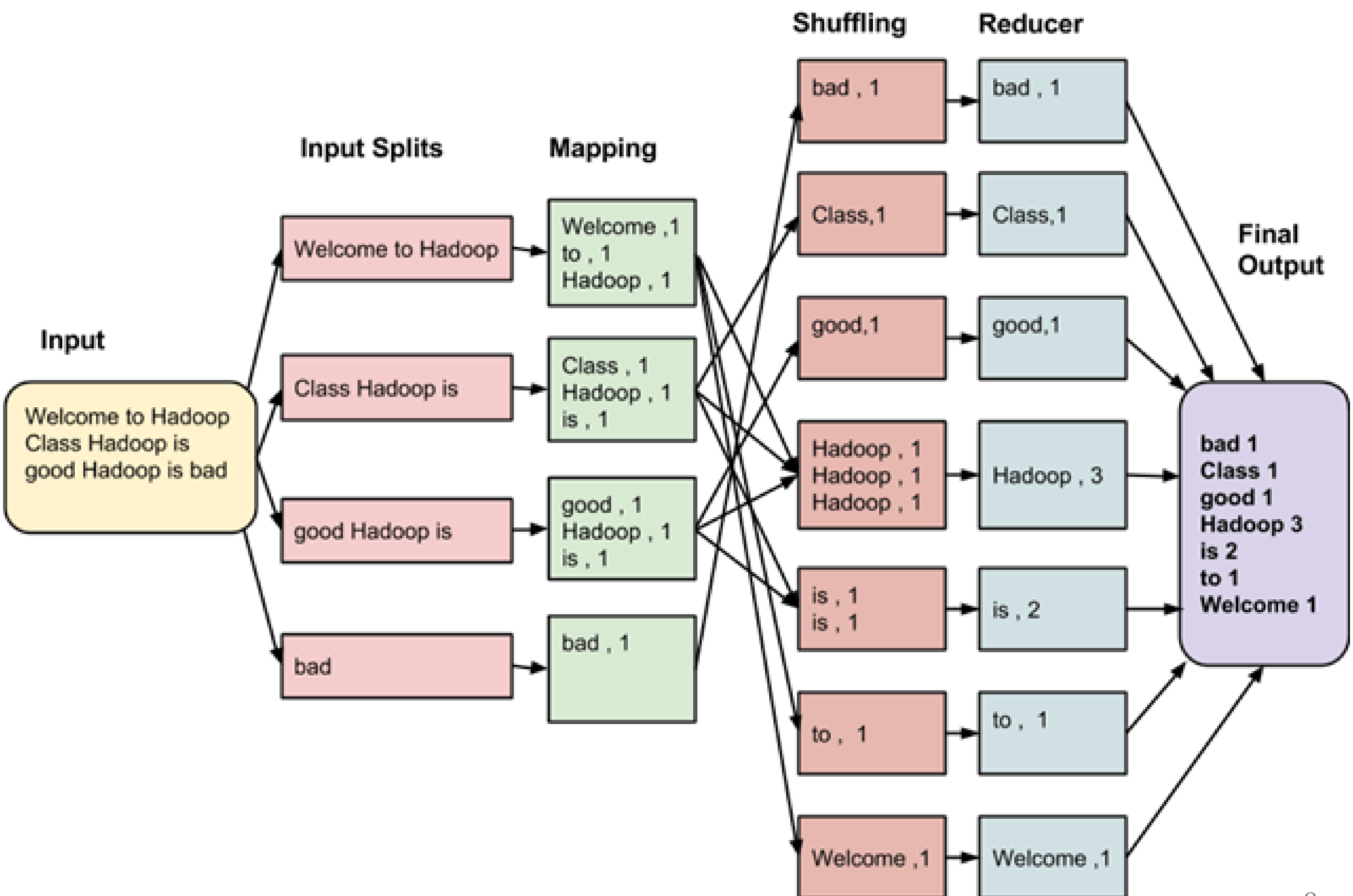


MapReduce - Execution Flow

- Figure show the execution flow of word count MapReduce jobs.
- As seen from these figures, the sort and shuffle phase begins as soon as a part of map task completes and the reduce phase begins after the intermediate key-value pairs from all the map tasks are sorted and shuffled to the reducer.

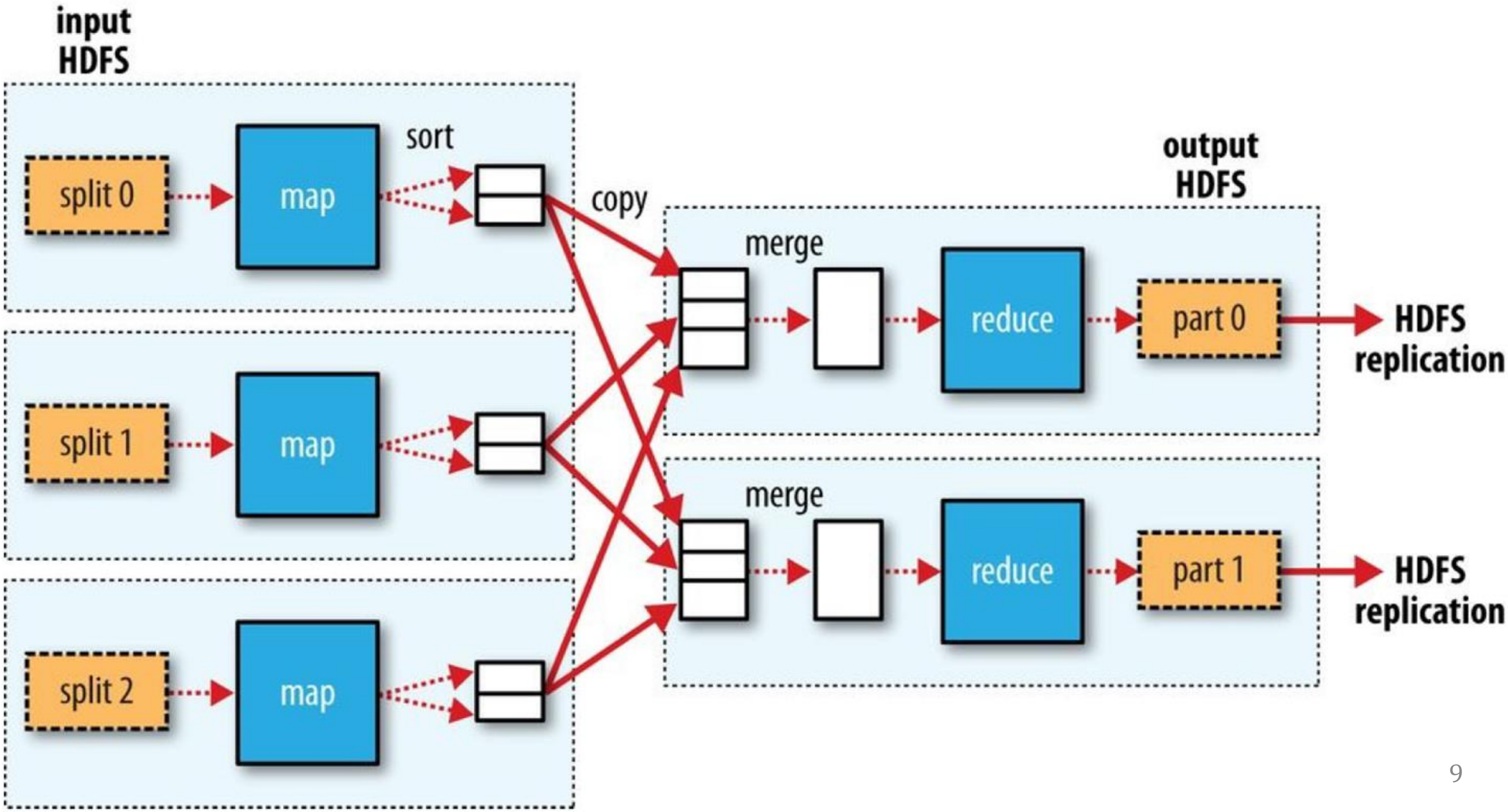


A Simple MapReduce Example

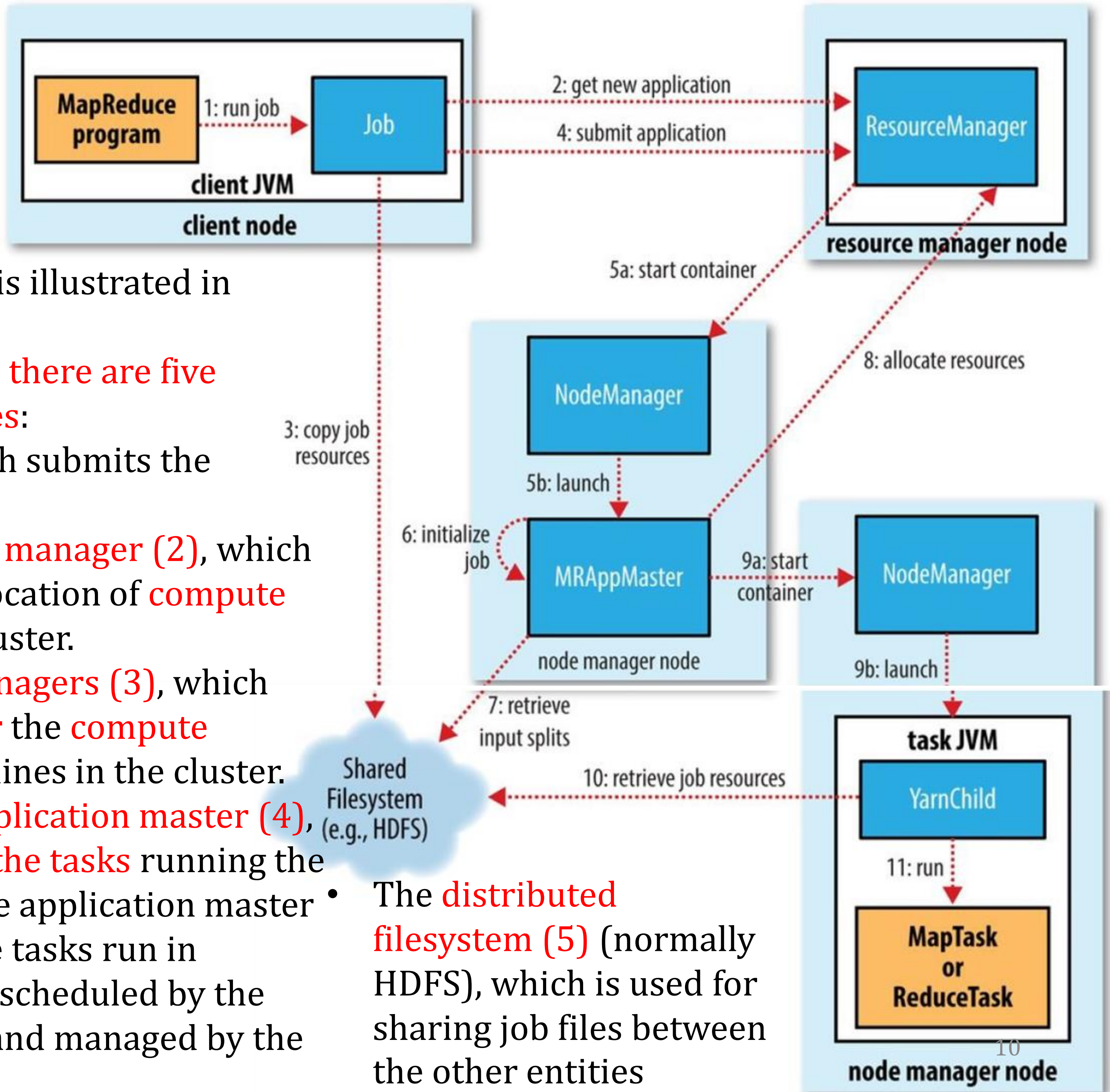


MapReduce Data Flow

- The dotted boxes indicate nodes, the dotted arrows show data transfers on a node, and the solid arrows show data transfers between nodes.

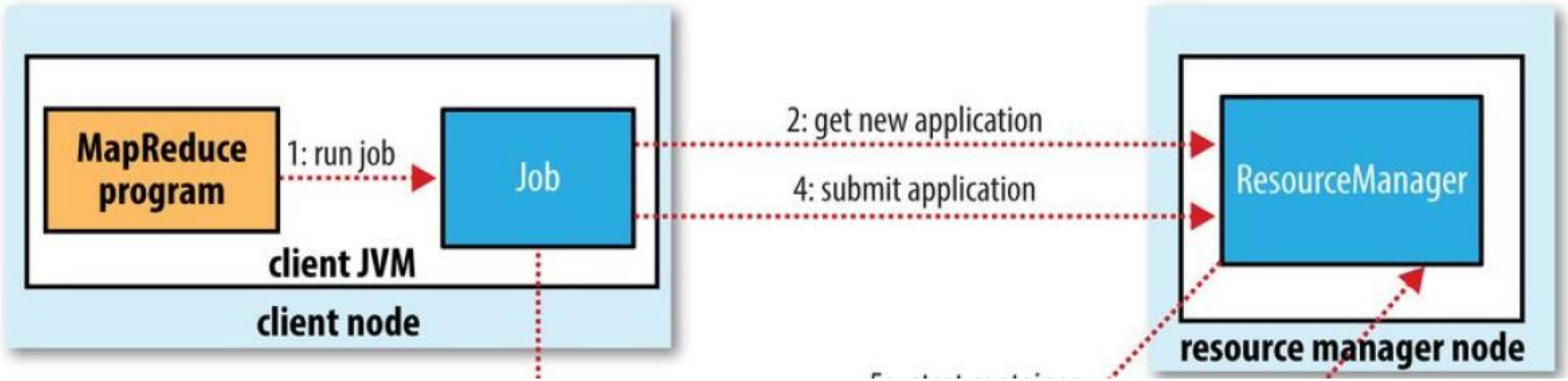


MapReduce Job Run



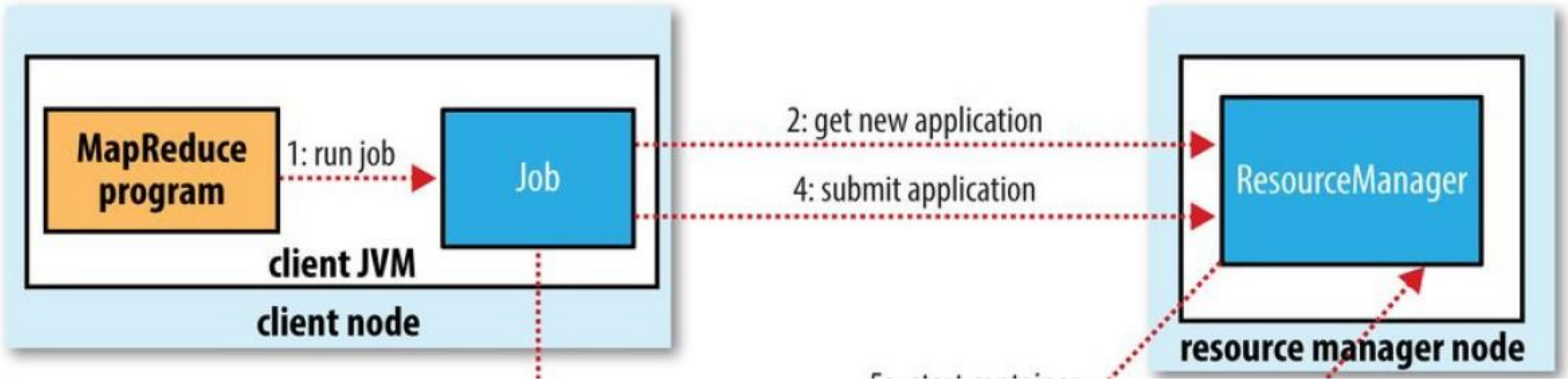
- The whole process is illustrated in Figure.
- At the highest level, there are five independent entities:
- The client (1), which submits the MapReduce job.
- The YARN resource manager (2), which coordinates the allocation of compute resources on the cluster.
- The YARN node managers (3), which launch and monitor the compute containers on machines in the cluster.
- The MapReduce application master (4), which coordinates the tasks running the MapReduce job. The application master and the MapReduce tasks run in containers that are scheduled by the resource manager and managed by the node managers.
- The distributed filesystem (5) (normally HDFS), which is used for sharing job files between the other entities

MapReduce – Job Submission



- The *submit()* method on Job creates an internal *JobSubmitter* instance and calls *submitJobInternal()* on it (**step 1**).
- *waitForCompletion()* polls the job's progress once per second and reports the progress to the console if it has changed since the last report.
- When the job completes successfully, the job counters are displayed. Otherwise, the error that caused the job to fail is logged to the console.
- The job submission process implemented by ***JobSubmitter*** does the following:
- Asks the resource manager for a new application ID, used for the MapReduce job ID (**step 2**).
- Checks the output specification of the job. For example, if the output directory has not been specified or it already exists, the job is not submitted and an error is thrown to the MapReduce program.

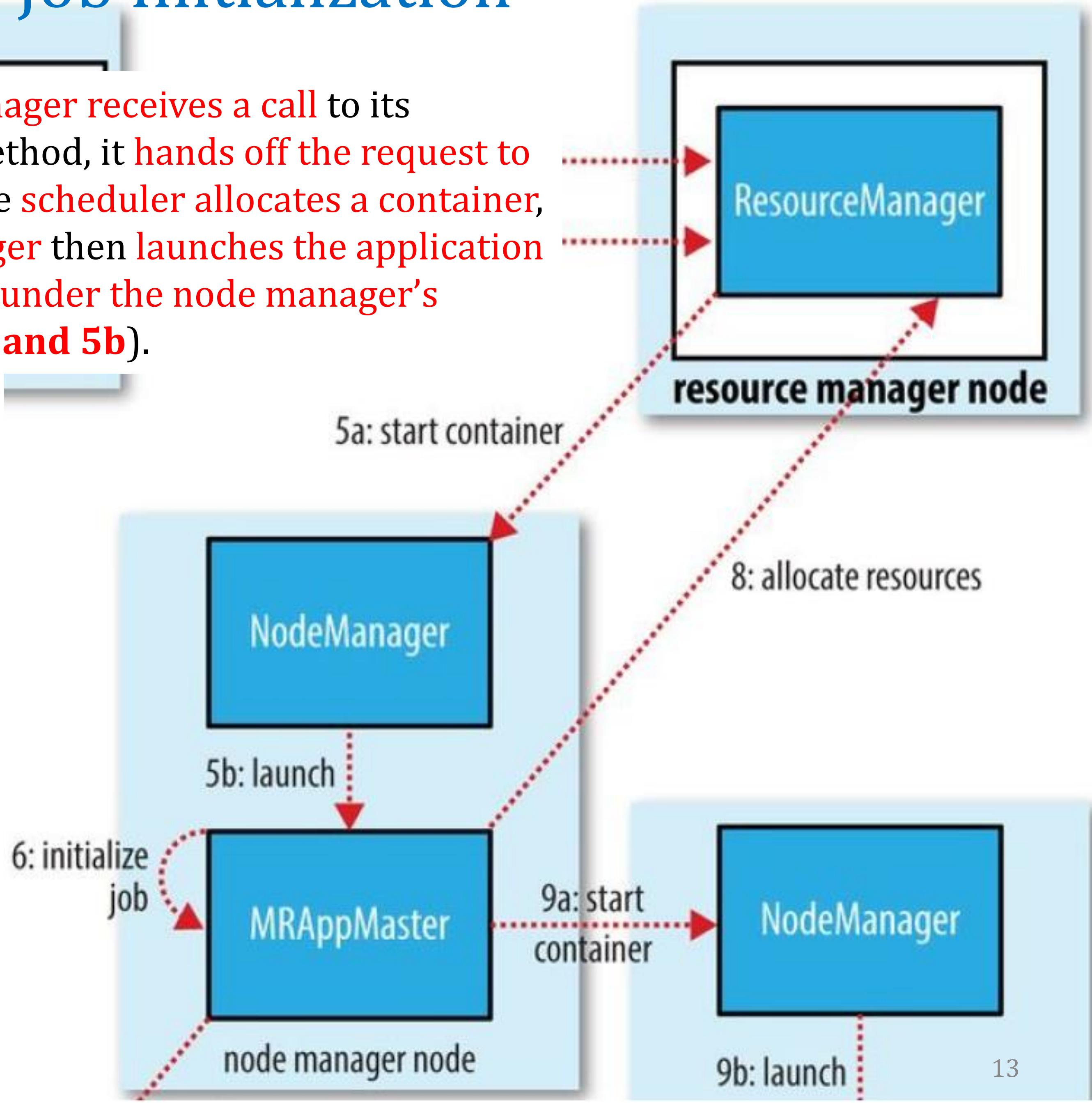
MapReduce – Job Submission



- Computes the input splits for the job. If the splits cannot be computed (because the input paths don't exist, for example), the job is not submitted and an error is thrown to the MapReduce program.
- Copies the resources needed to run the job into the HDFS, including the job JAR file, the configuration file, and the computed input splits, to the shared filesystem in a directory named after the job ID (**step 3**). The job JAR is copied with a high replication factor (controlled by the *mapreduce.client.submit.file.replication* property, which defaults to 10).
- Submits the job by calling *submitApplication()* on the resource manager (**step 4**).

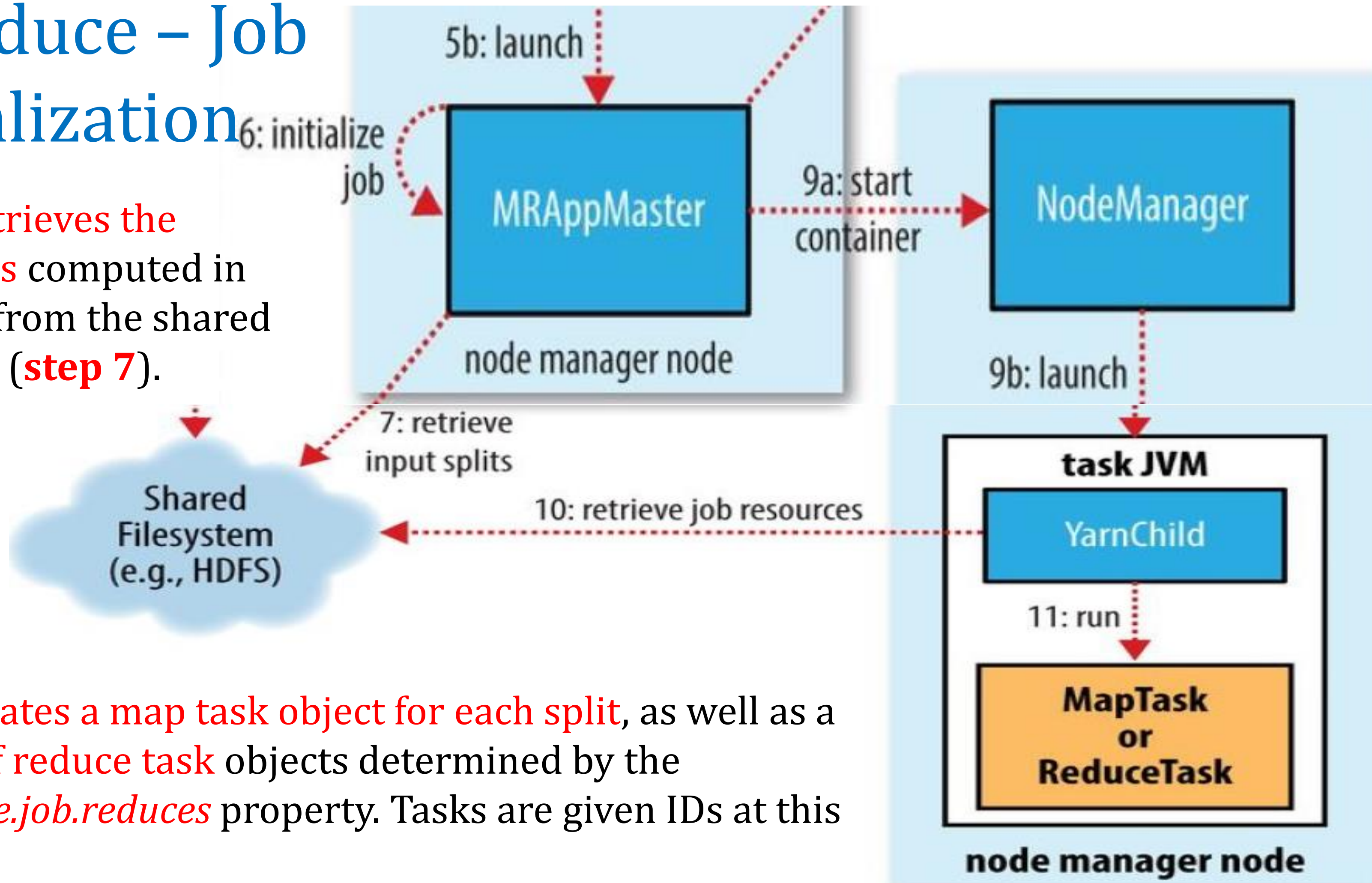
MapReduce – Job Initialization

- When the resource manager receives a call to its submitApplication() method, it hands off the request to the YARN scheduler. The scheduler allocates a container, and the resource manager then launches the application master's process there, under the node manager's management (**steps 5a and 5b**).
- The application master for MapReduce jobs is a Java application whose main class is *MRAppMaster*. It initializes the job by creating a number of bookkeeping objects to keeptrack of the job's progress, as it will receive progress and completion reports from the tasks (**step 6**).



MapReduce – Job Initialization

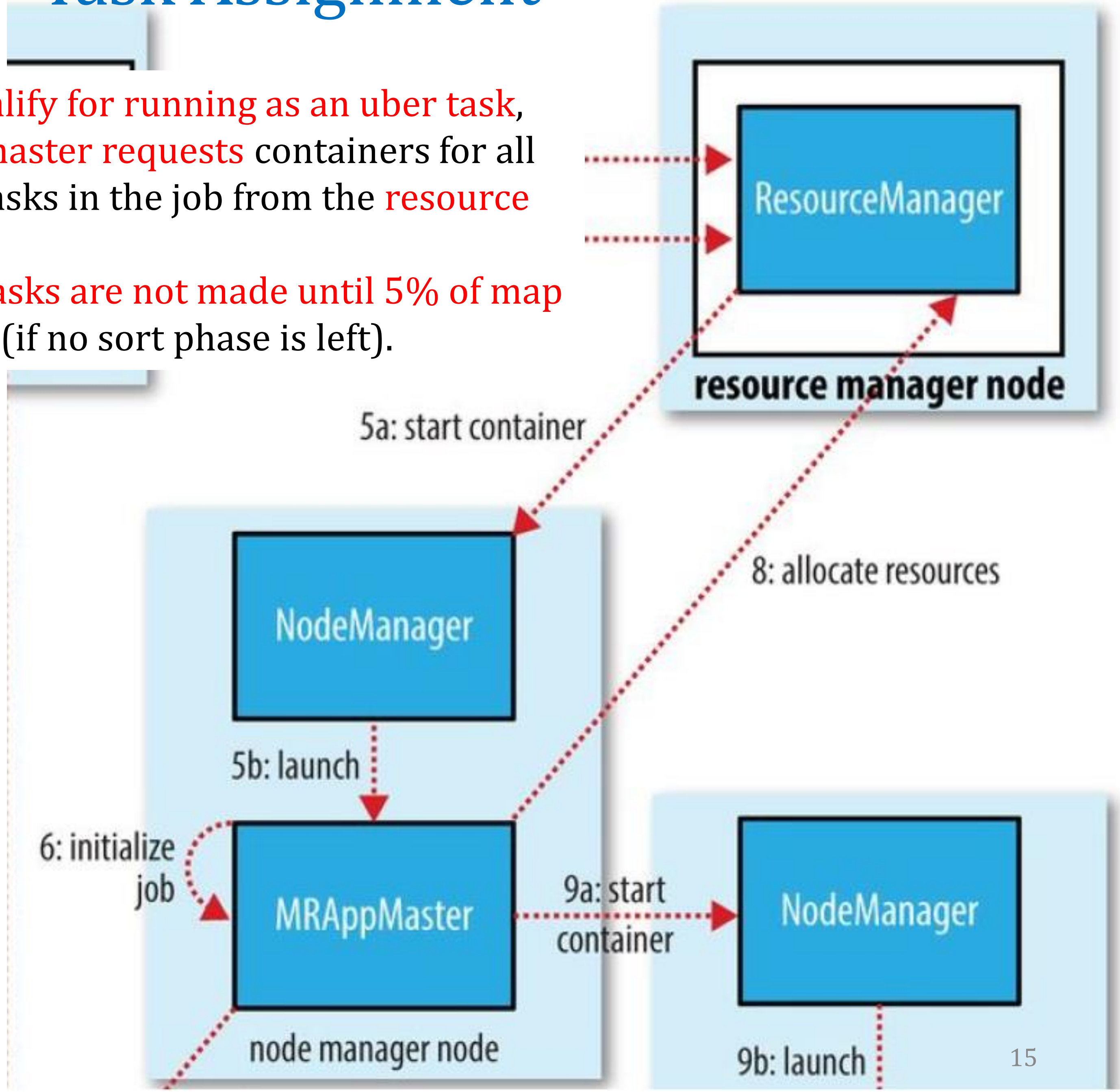
- Next, it retrieves the input splits computed in the client from the shared filesystem (**step 7**).



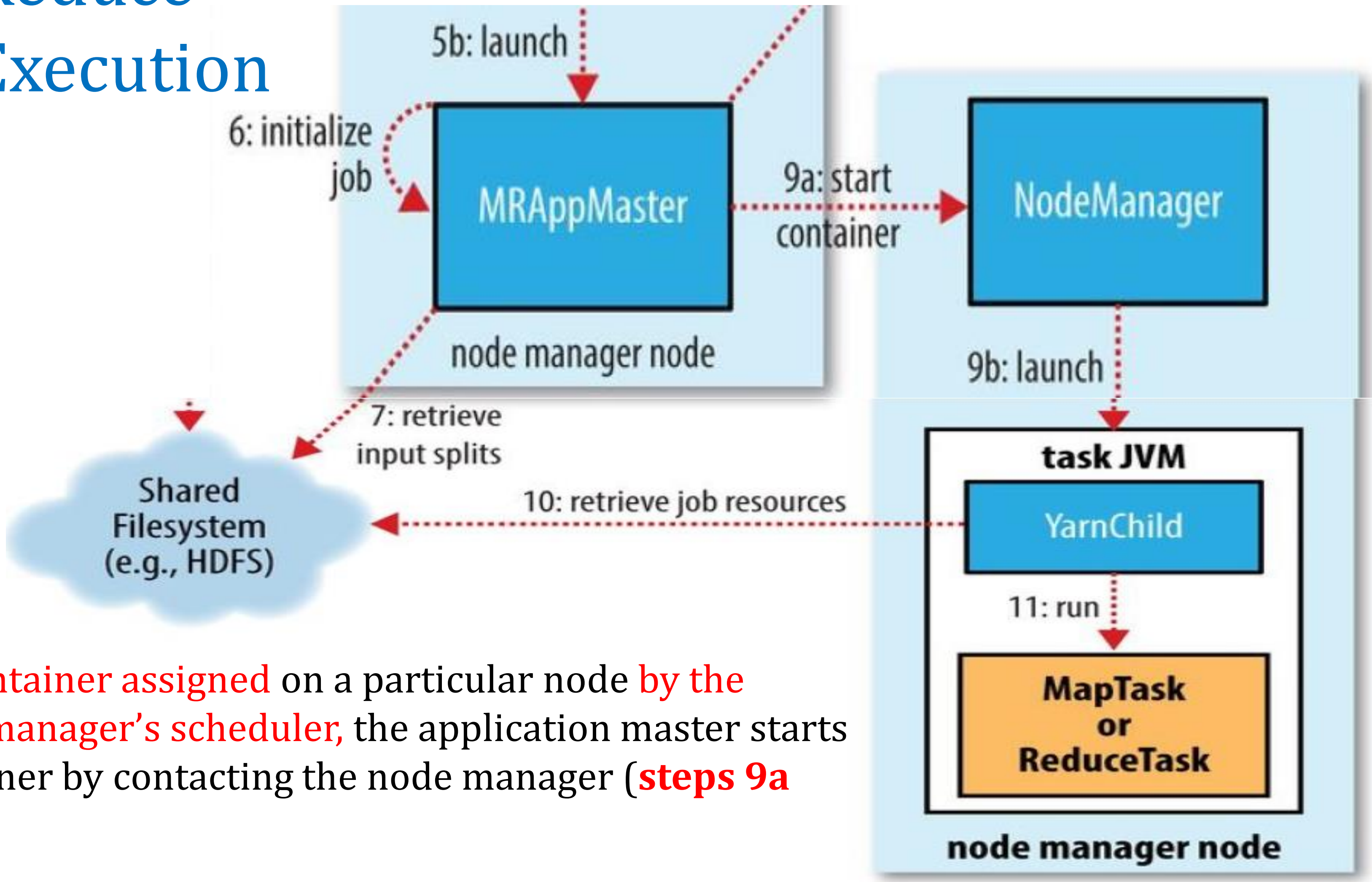
- It then creates a map task object for each split, as well as a number of reduce task objects determined by the *mapreduce.job.reduces* property. Tasks are given IDs at this point.
- The application master must decide how to run the tasks that make up the MapReduce job. If the job is small (job has <10 mappers, one reducers or input size < HDFS block), the application master may choose to run the tasks in the same JVM as itself. This happens when it judges that the overhead of allocating and running tasks in new containers outweighs the gain to be had in running them in parallel, compared to running them sequentially on one node. Such a job is said to be *uberized*, or run as an *uber task*.

MapReduce – Task Assignment

- If the job does not qualify for running as an uber task, then the application master requests containers for all the map and reduce tasks in the job from the resource manager (**step 8**).
- Requests for reduce tasks are not made until 5% of map tasks have completed (if no sort phase is left).



MapReduce – Task Execution



- Once a container assigned on a particular node by the resource manager's scheduler, the application master starts the container by contacting the node manager (**steps 9a and 9b**).
- The task is executed by a Java application whose main class is YarnChild. Before it can run the task, it localizes the resources that the task needs, including the job configuration and JAR file, and any files from the distributed cache (**step 10**). Finally, it runs the map or reduce task (**step 11**).

MapReduce - Numerical Summarization

- Numerical summarization patterns are used to compute various statistics such as counts, maximum, minimum, mean, etc.
- For example, computing the total number of likes for a particular post, computing the average monthly rainfall or finding the average number of visitors per month on a website.
- We will use synthetic data similar to the data collected by a web analytics service that shows various statistics for page visits for a website.
- Each page has some tracking code which sends the visitor's IP address along with a timestamp to the web analytics service. The web analytics service keeps a record of all page visits and the visitor IP addresses and uses MapReduce programs for computing various statistics.

MapReduce - Numerical Summarization

- Each visit to a page is logged as one row in the log. The log file contains the following columns:
- Date (YYYY-MM-DD), Time (HH:MM:SS), URL, IP, Visit-Length.

Input

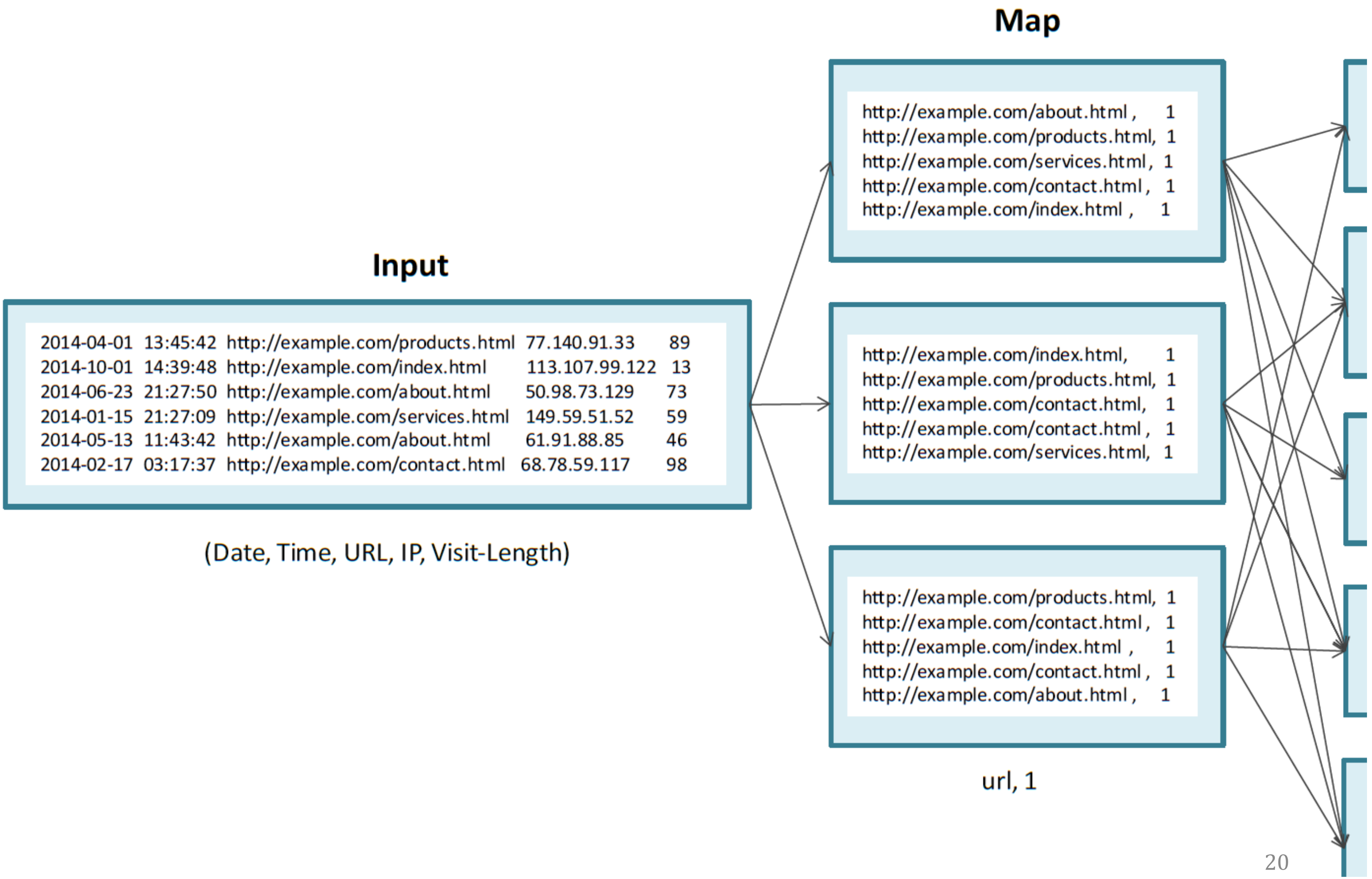
2014-04-01	13:45:42	http://example.com/products.html	77.140.91.33	89
2014-10-01	14:39:48	http://example.com/index.html	113.107.99.122	13
2014-06-23	21:27:50	http://example.com/about.html	50.98.73.129	73
2014-01-15	21:27:09	http://example.com/services.html	149.59.51.52	59
2014-05-13	11:43:42	http://example.com/about.html	61.91.88.85	46
2014-02-17	03:17:37	http://example.com/contact.html	68.78.59.117	98

(Date, Time, URL, IP, Visit-Length)

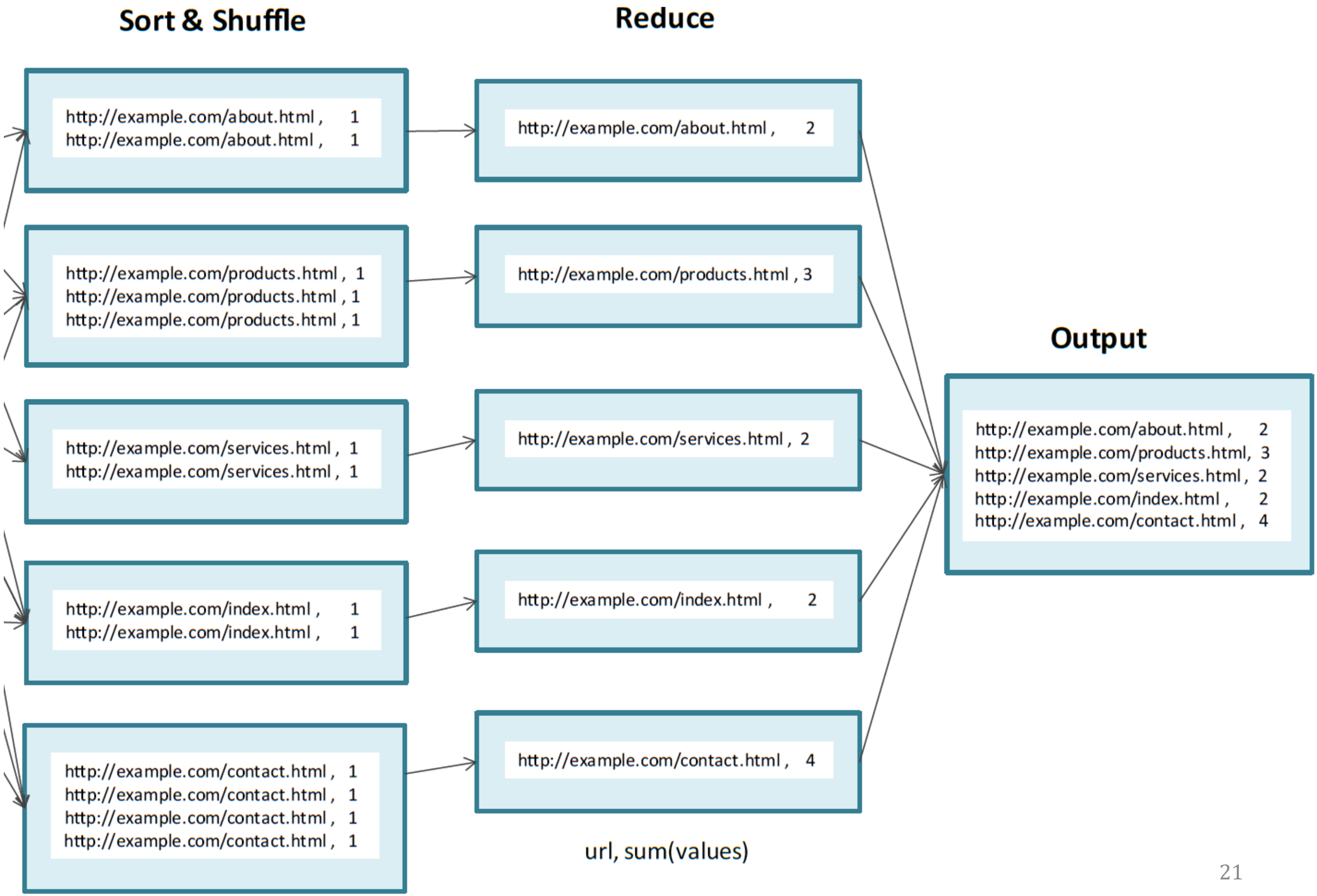
Numerical Summarization - Count

- To compute count, the **mapper function** emits key-value pairs where the key is the field to group-by and value is either ‘1’ or any related items required to compute count.
- The **reducer function** receives the key-value pairs grouped by the same key and adds up the values for each group to compute count.
- Let us look at an example of computing the total number of times each page is visited in the year 2014, from the web analytics service logs.
- The mapper function in this example parses each line of the input and emits key-value pairs where the key is the URL and value is ‘1’.
- The reducer receives the list of values grouped by the key and sums up the values to compute count.

Numerical Summarization – Count (cont'd)



Numerical Summarization – Count (cont'd)



Count – Python Program

```
#Total number of times each page is visited in the year 2014
#in order to run type
#>python your_mr_job_sub_class.py < log_file_or_whatever > output
#Ex: >python MRmyjob.py web_click.txt

from mrjob.job import MRJob

class MRmyjob(MRJob):
  def mapper(self, _, line):
    #Split the line with tab separated fields
    data=line.split('\t')

    #Parse line
    date = data[0].strip()
    time = data[1].strip()
    url = data[2].strip()
    ip = data[3].strip()
```

MapReduce Patterns

```
    #Extract year from date
    year=date[0:4]

    #Emit URL and 1 if year is 2014
    if year=='2014':
      yield url, 1

  def reducer(self, key, list_of_values):
    yield key,sum(list_of_values)

if __name__ == '__main__':
  MRmyjob.run()
```

MapReduce - TopN

- To find the top-N records, the mapper function emits key-value pairs where the key is the field to group by and value contains related items required to compute top-N.
- The reducer function receives the list of values grouped by the same key, sorts the values and emits the top-N values for each key.
- In an alternative approach, each mapper emits its local top-N records and the reducer then finds the global top-N.
- Let us look at an example of computing the top 3 visited page in the year 2014.
- In this example, a two-step job was required because we need to compute the page visit counts first before finding the top 3 visited pages.
- The mapper function in this example parses each line of the input and emits key-value pairs where the key is the URL and value is '1'.
- The reducer receives the list of values grouped by the key and sums up the values to count the visits for each page. The reducer emits None as the key and a tuple comprising of page visit count and page URL and the value. The second reducer receives a list of (visit count, URL) pairs all grouped together (as the key is None). The reducer sorts the visit counts and emits top 3 visit counts along with the page URLs.

MapReduce – TopN (cont'd)

Map

about.html, 1
products.html, 1
index.html, 1
contact.html, 1
index.html, 1

Input

2014-04-01	13:45:42	http://example.com/products.html	77.140.91.33	89
2014-10-01	14:39:48	http://example.com/index.html	113.107.99.122	13
2014-06-23	21:27:50	http://example.com/about.html	50.98.73.129	73
2014-01-15	21:27:09	http://example.com/services.html	149.59.51.52	59
2014-05-13	11:43:42	http://example.com/about.html	61.91.88.85	46
2014-02-17	03:17:37	http://example.com/contact.html	68.78.59.117	98

(Date, Time, URL, IP, Visit-Length)

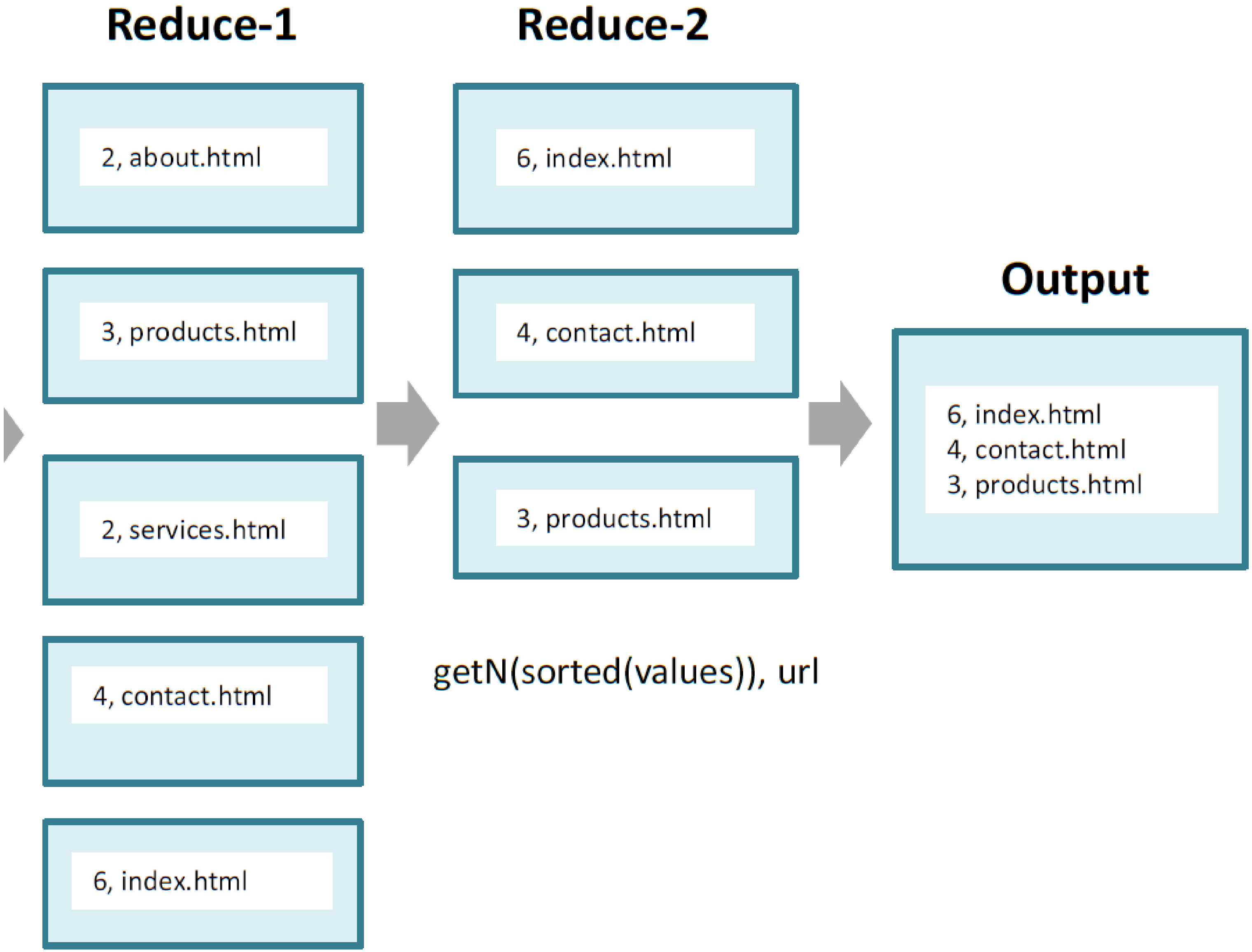
index.html, 1
products.html, 1
contact.html, 1
contact.html, 1
index.html, 1
index.html, 1

products.html, 1
contact.html, 1
index.html, 1
contact.html, 1
about.html, 1
services.html, 1

Non

url, 1

MapReduce – TopN (cont'd)



None, (sum(values), url)

MapReduce – TopN (cont'd)

```
# Top 3 visited page in year 2014
from mrjob.job import MRJob, MRStep

class MRmyjob(MRJob):
  def mapper(self, _, line):
    #Split the line with tab separated fields
    data=line.split('\t')

    #Parse line
    date = data[0].strip()
    time = data[1].strip()
    url = data[2].strip()
    ip = data[3].strip()
    visit_len=int(data[4].strip())
    #Extract year from date
    year=date[0:4]

    #Emit url and 1 if year is 2014
    if year=='2014':
    yield url, 1
```

MapReduce – TopN (cont'd)

```
def reducer1(self, key, list_of_values):
  total_count = sum(list_of_values)
  yield None, (total_count, key)

def reducer2(self, _, list_of_values):
  N = 3
  list_of_values = sorted(list(list_of_values), reverse=True)
  return list_of_values[:N]

def steps(self):
  return [MRStep(mapper=self.mapper, reducer=self.reducer1),
          MRStep(reducer=self.reducer2)]

if __name__ == '__main__':
  MRmyjob.run()
```