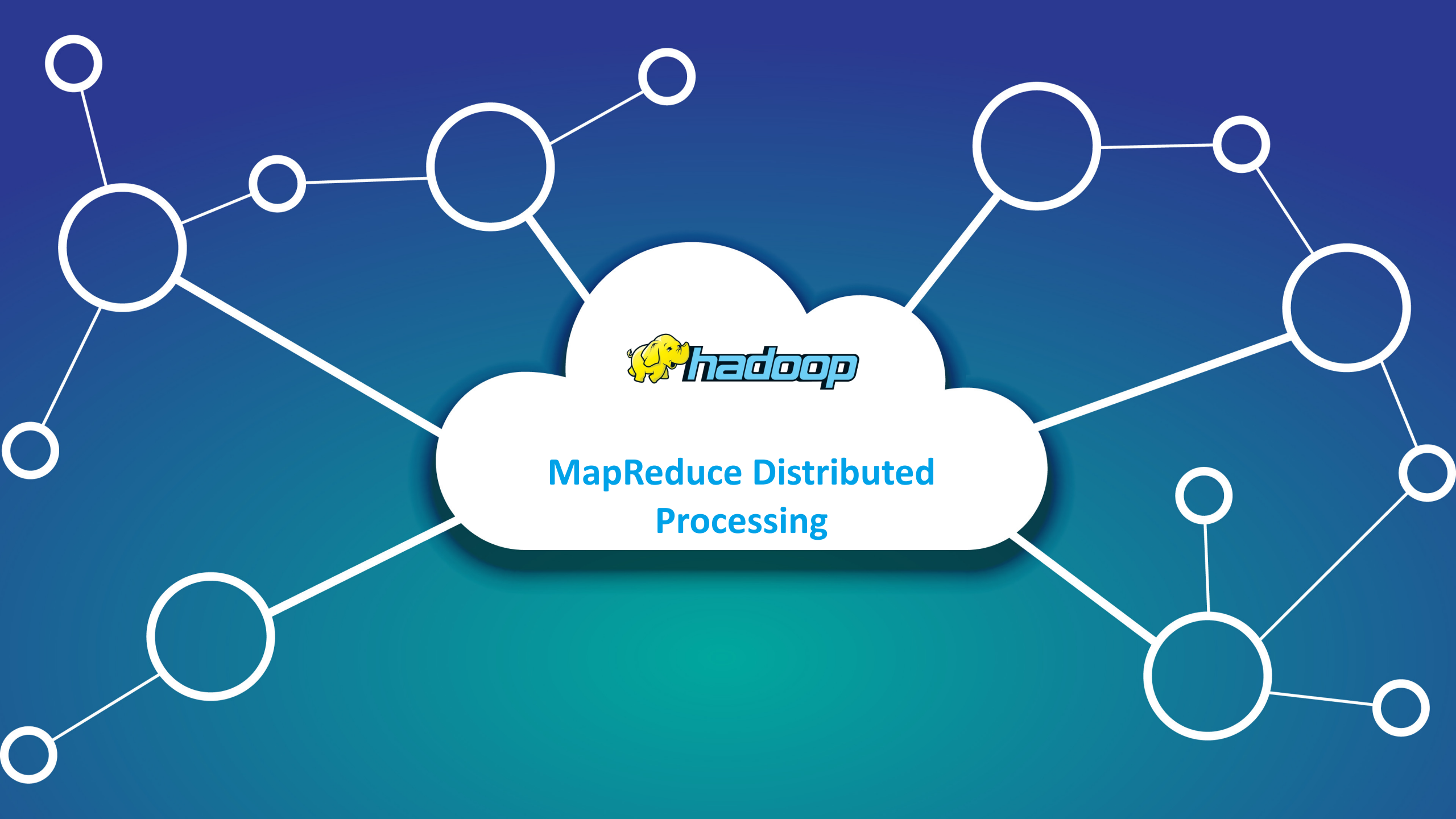


MapReduce Distributed Processing

Contents

Before MapReduce

- Large scale data processing was difficult!
  - Managing hundreds or thousands of processors
  - Managing parallelization and distribution
  - I/O Scheduling
  - Status and monitoring
  - Fault/crash tolerance
- MapReduce provides all of these, easily!

Overview of MapReduce

- Hadoop Map-Reduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.
  - Programming model used by Google
  - A combination of the Map and Reduce models with an associated implementation
  - MapReduce is highly scalable and can be used across many computers and answers all problems mentioned earlier.
  - Many small machines can be used to process jobs that normally could not be processed by a large machine.
  - Programming Languages Supported: Java
    - Hadoop Streaming: Python, Ruby, Perl etc
    - Hadoop Pipes: C++

MapReduce Used for

- **At Google:**
  - Index building for Google Search
  - Article clustering for Google News
  - Statistical machine translation
- **At Yahoo!:**
  - Index building for Yahoo! Search
  - Spam detection for Yahoo! Mail
- **At Facebook:**
  - Ad optimization
  - Spam detection

Mapper

- Mapper reads the data in key/value pairs
- Outputs zero or more key/value pairs

  map(in_key, in_value) ->
  (inter_key, inter_value) list
- Mapper may use or ignore key
  - The key is the byte offset into the file at which the line starts
  - The value is the contents of the line itself
  - Typically the key is considered irrelevant
- If the Mapper writes anything out, the output must be in the form of key/value pairs

Example: Turn into upper case

let map(k, v) = emit(k.toUpper(), v.toUpper())

Input - (‘hadoop', ‘big data') -> output - (‘HADOOP', ‘BIG DATA')

Reducer

- After Map phase, all intermediate values are combined together for given intermediate key into a list.
- All values associated with a particular intermediate key are guaranteed to go to the same Reducer
- The intermediate keys, and their value lists, are passed to the
- Reducer in sorted key order
- This step is known as the ‘shuffle and sort’
- Reducer outputs zero or more key/value pairs

Example:

```
let reduce(k, vals)
    sum = o
    foreach  int i in vals:
            sum += i
    emit(k, sum)
Input - (’bee', [111, 100, 110]) ->
Output -(’bee', 321)
```

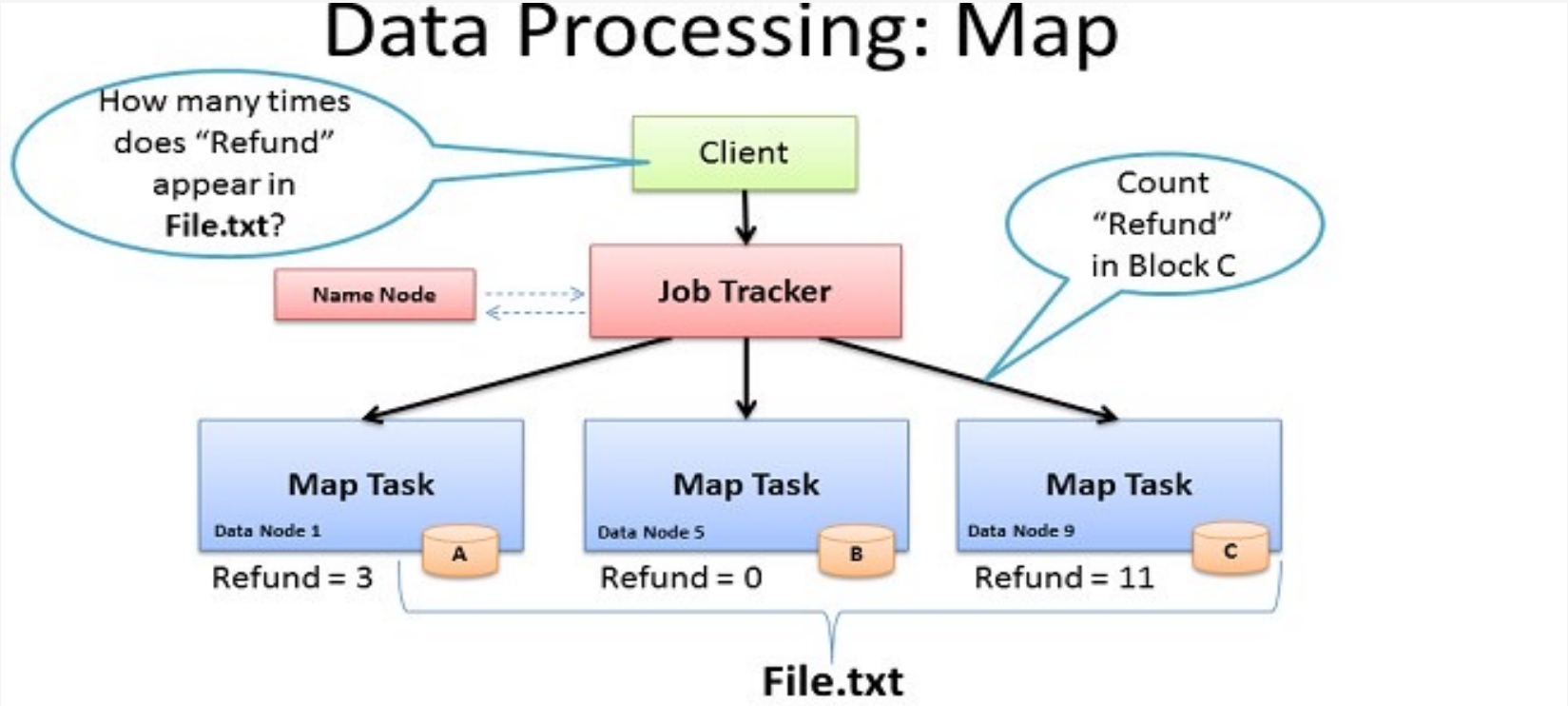
```
Identity Reducer:
let reduce(k, vals) =
   foreach v in vals:
       emit(k, v)

Input - (‘abc', [111, 222, 333]) -> Output
(‘abc', 111), (‘abc', 222), (‘abc', 333),
```

How many Maps/Reducers?

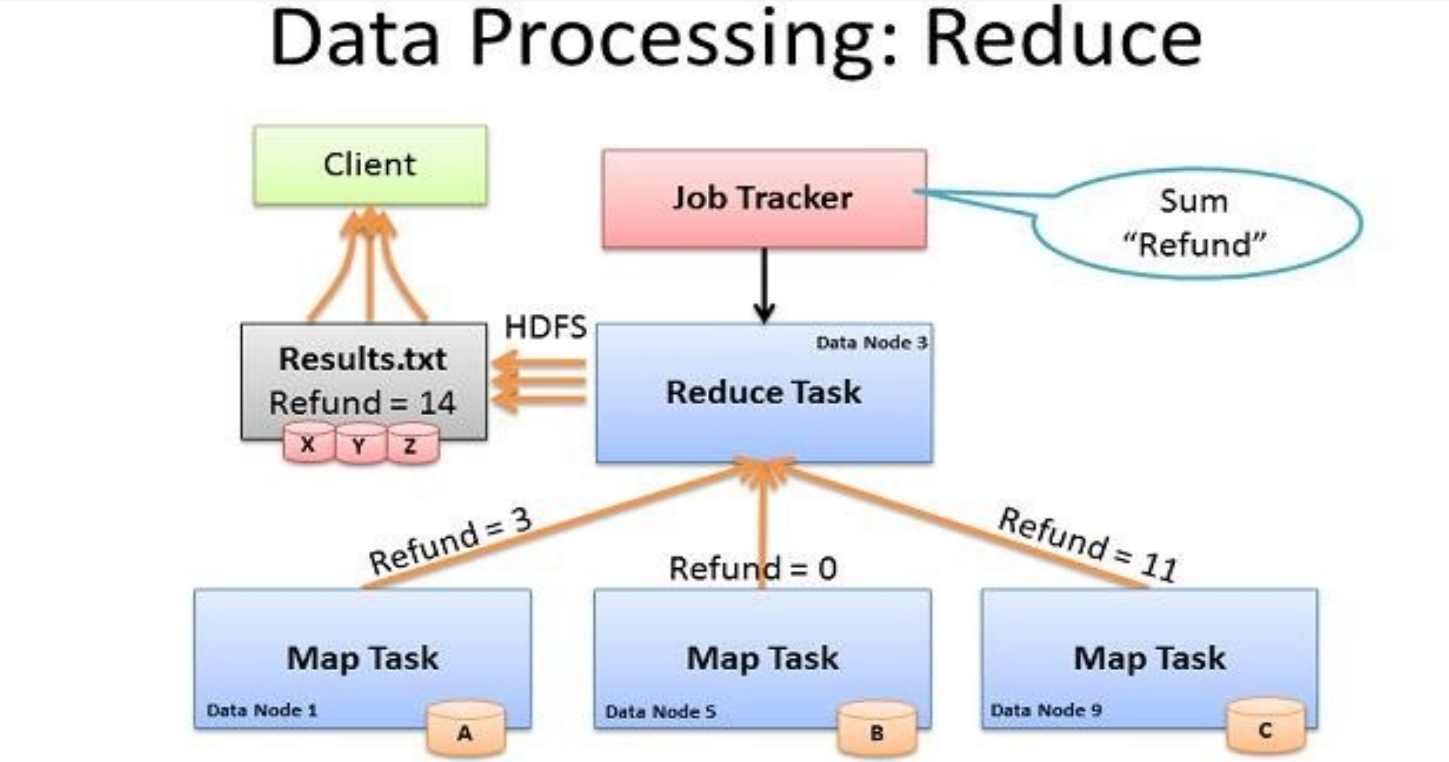
- **Maps**
  - Usually as many as the number of HDFS blocks being processed, this is the default.
  - Else the number of maps can be specified as a hint.
  - The number of maps can also be controlled by specifying the *minimum split size*
  - The actual sizes of the map inputs are computed by:
    max(min(block_size, data/#maps), min_split_size)
- **Reduces**
  - Unless the amount of data being processed is small
    *0.95*num_nodes*mapred.tasktracker.tasks.maximum*

Understanding Map Reduce Cluster



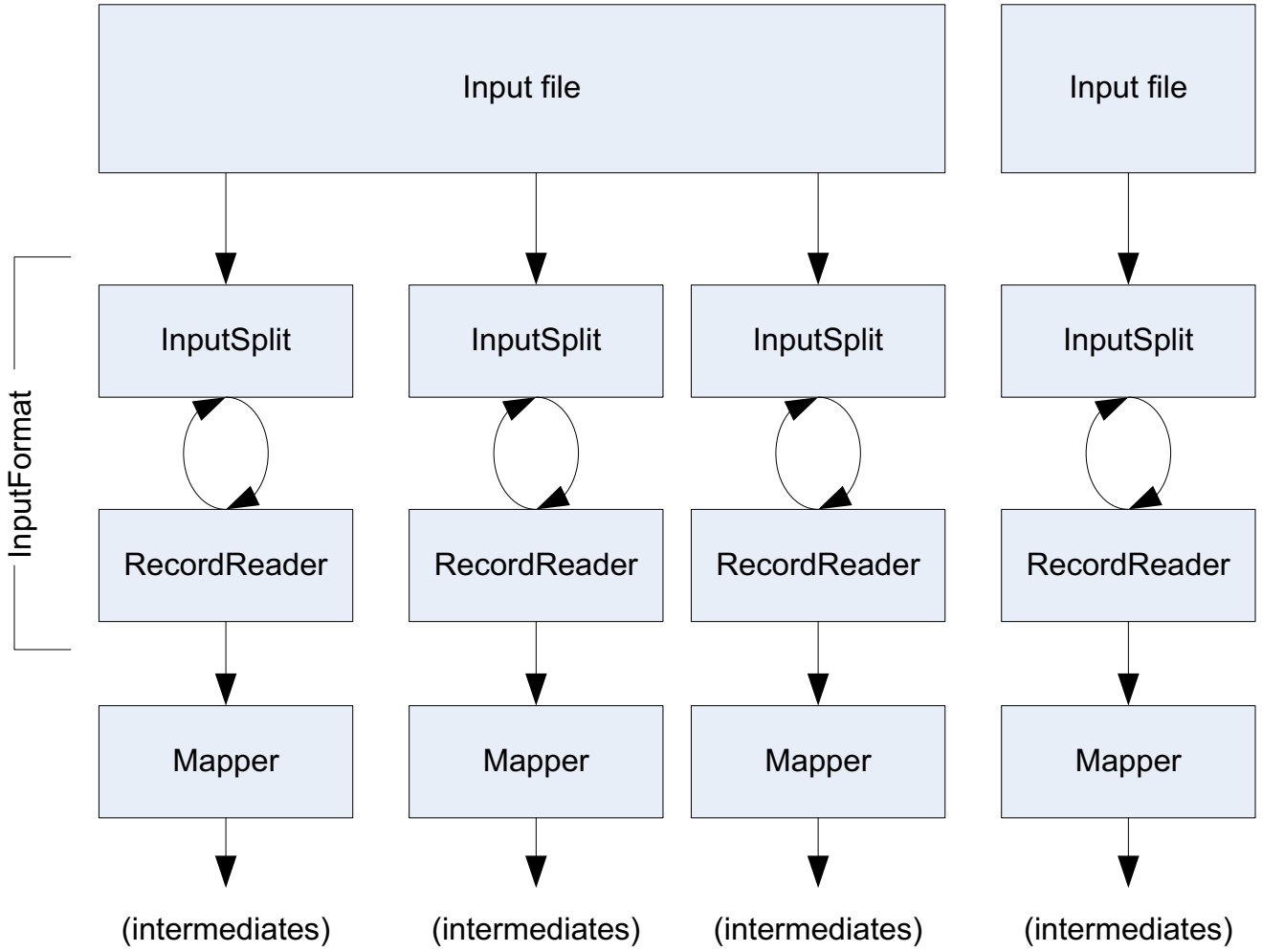
- **Map:** "Run this computation on your local data"
- Job Tracker delivers Java code to Nodes with local data

Understanding Map Reduce Cluster

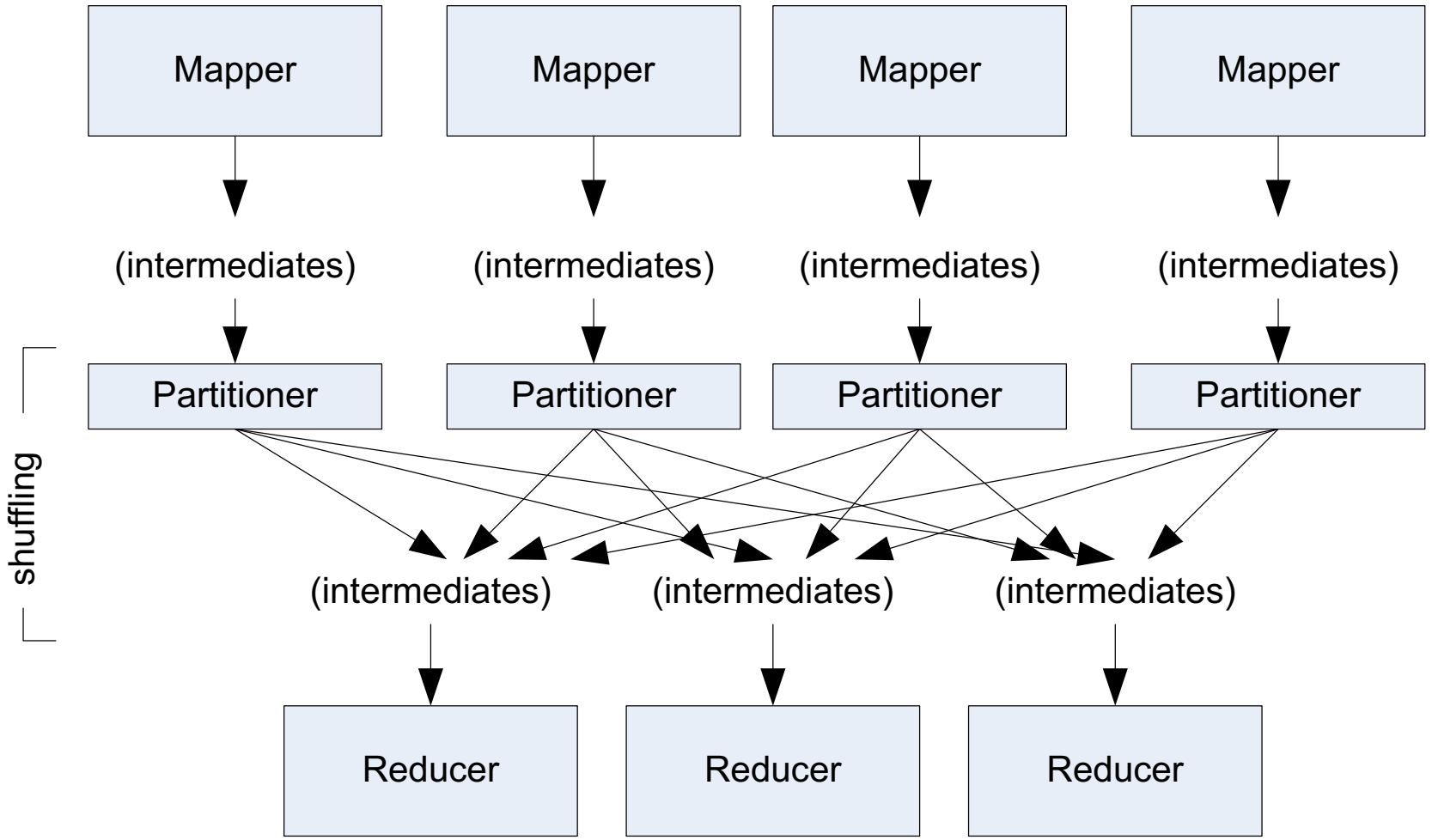


- **Reduce:** "Run this computation across Map results"
- Map Tasks send output data to Reducer over the network
- Reduce Task data output written to and read from HDFS

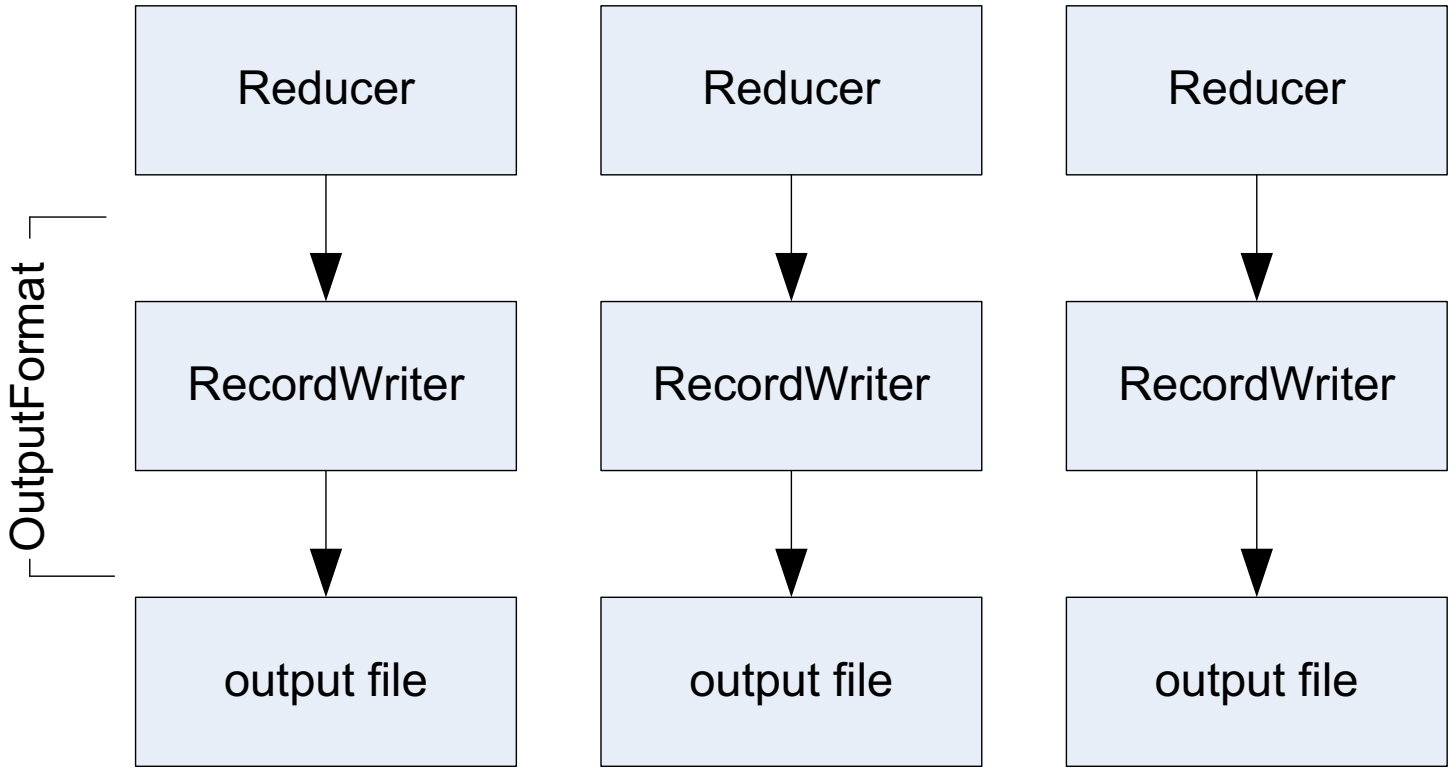
Map Reduce Flow – step1



Map Reduce Flow – Step2



Map Reduce Flow – Step3



Word Count – Distributed Solution

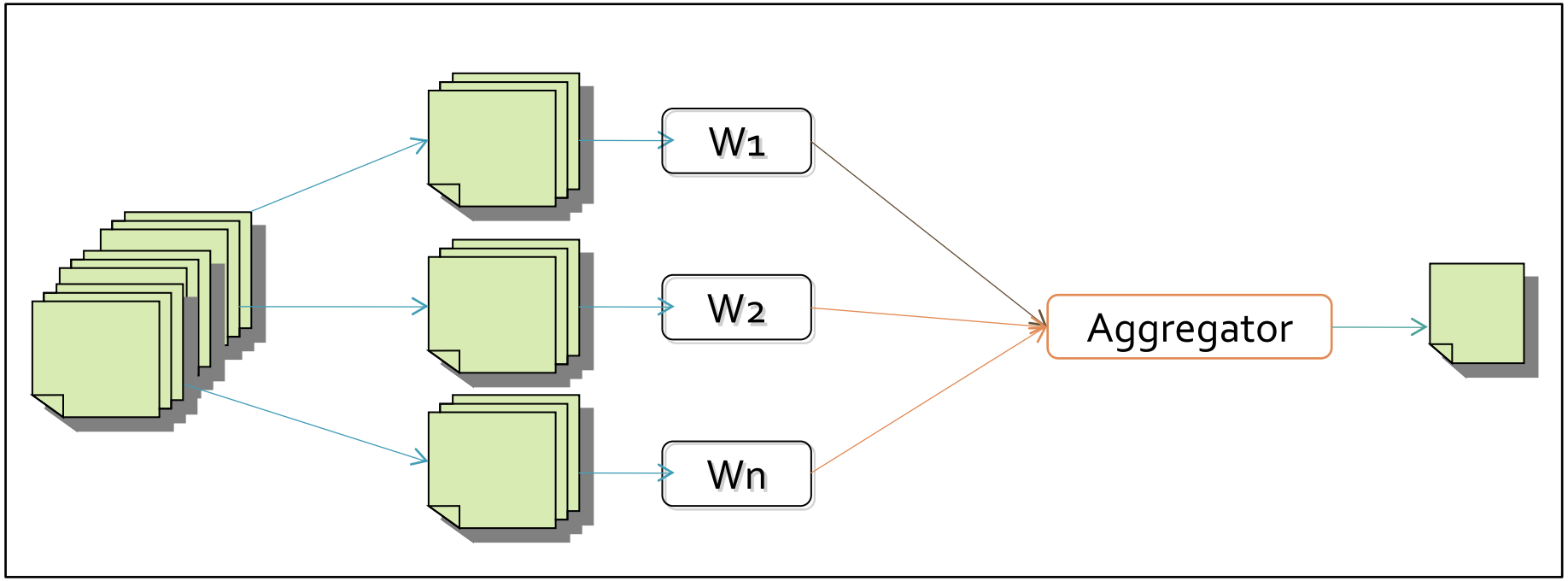
Problem:

Large quantity of documents

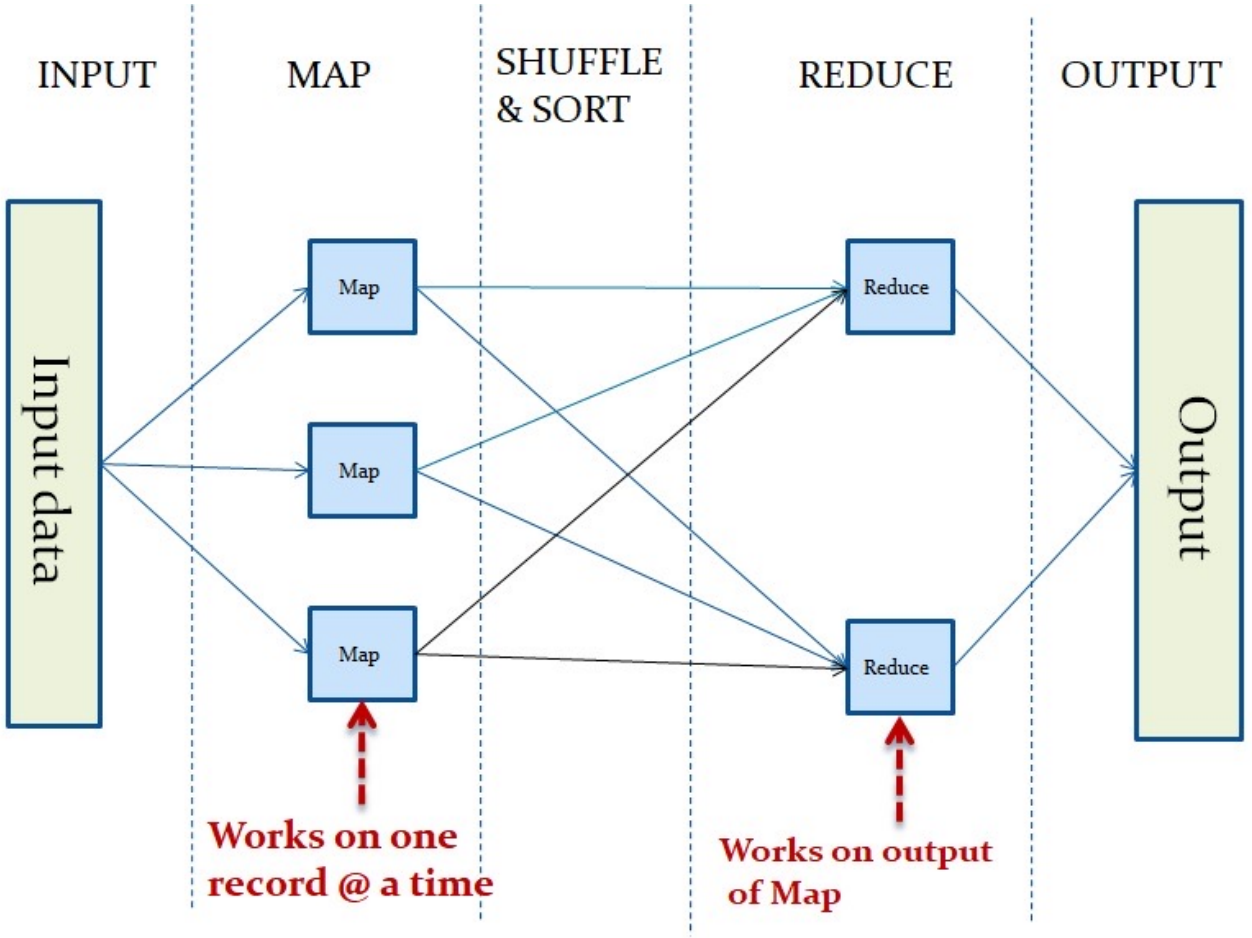
Count the number of times each distinct word occurs in the documents

Solution:

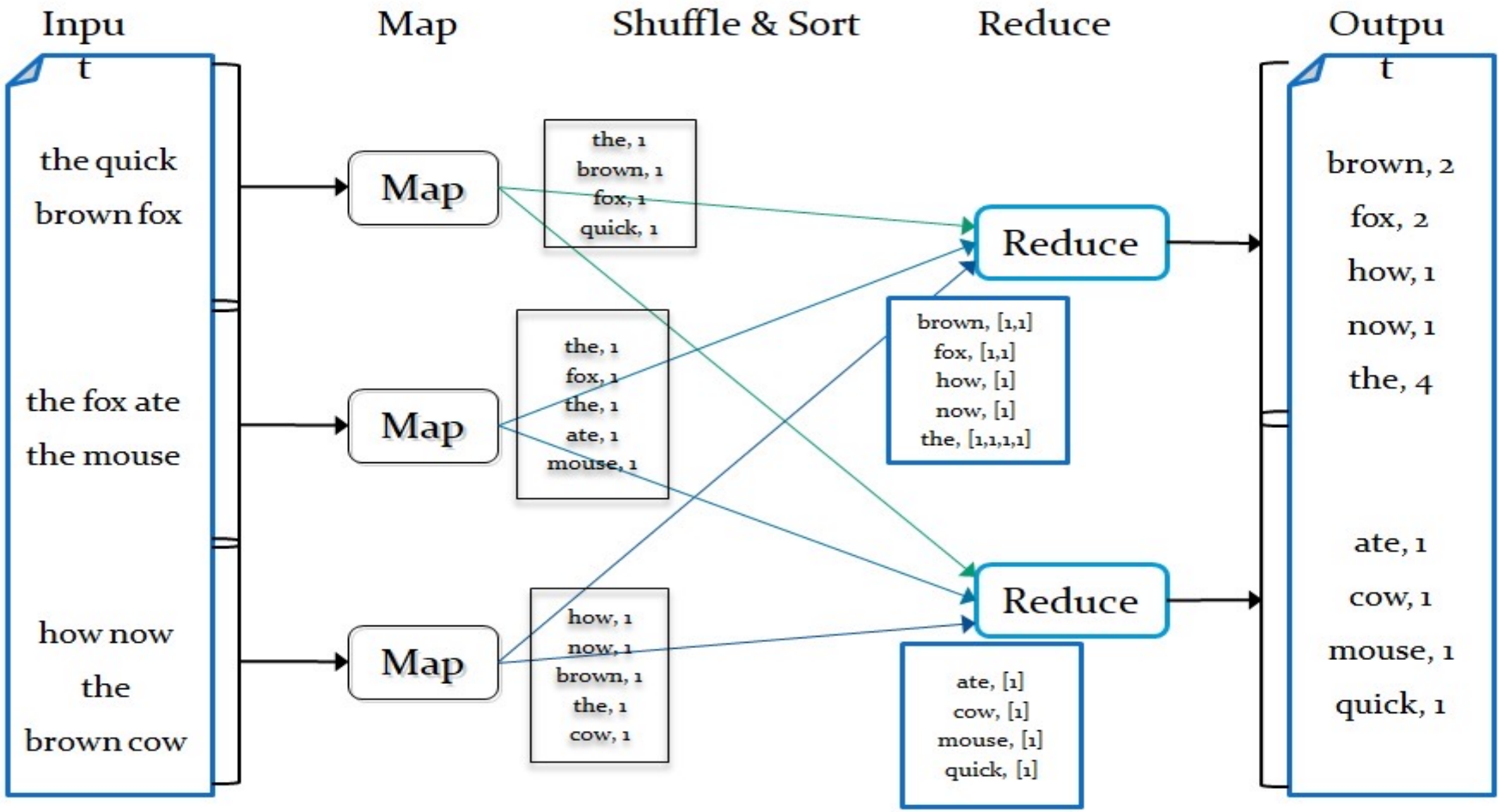
Divide and Conquer



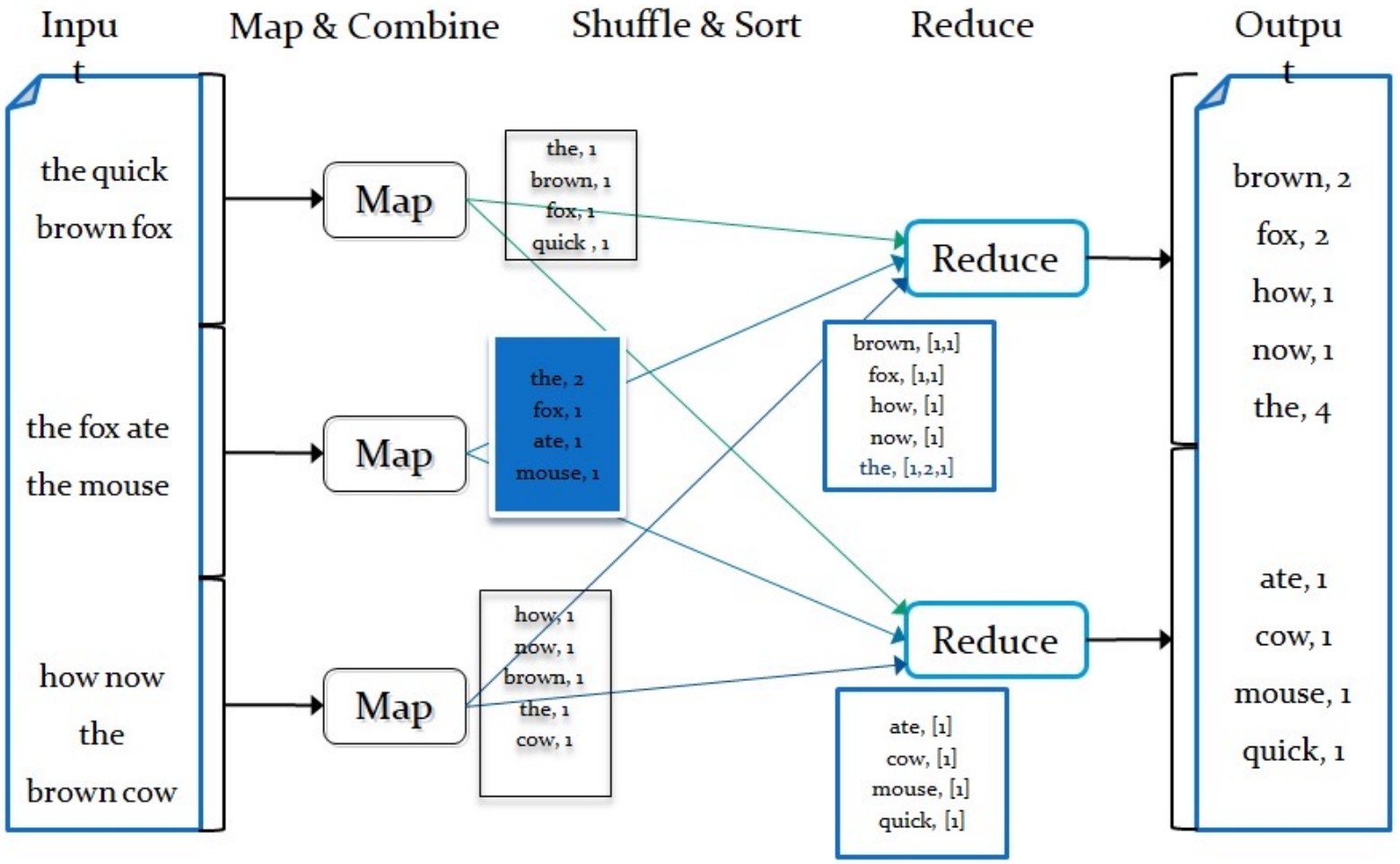
Map Reduce Flow



Word Count – Distributed Solution



Word Count – Distributed Solution



Word Count – Distributed Solution

```
def mapper(line):
    foreach word in line.split():
        output(word, 1)

def reducer(key, values[]):
    output(key, sum(values))
```

Map: (k1, v1) → list (k2, v2)

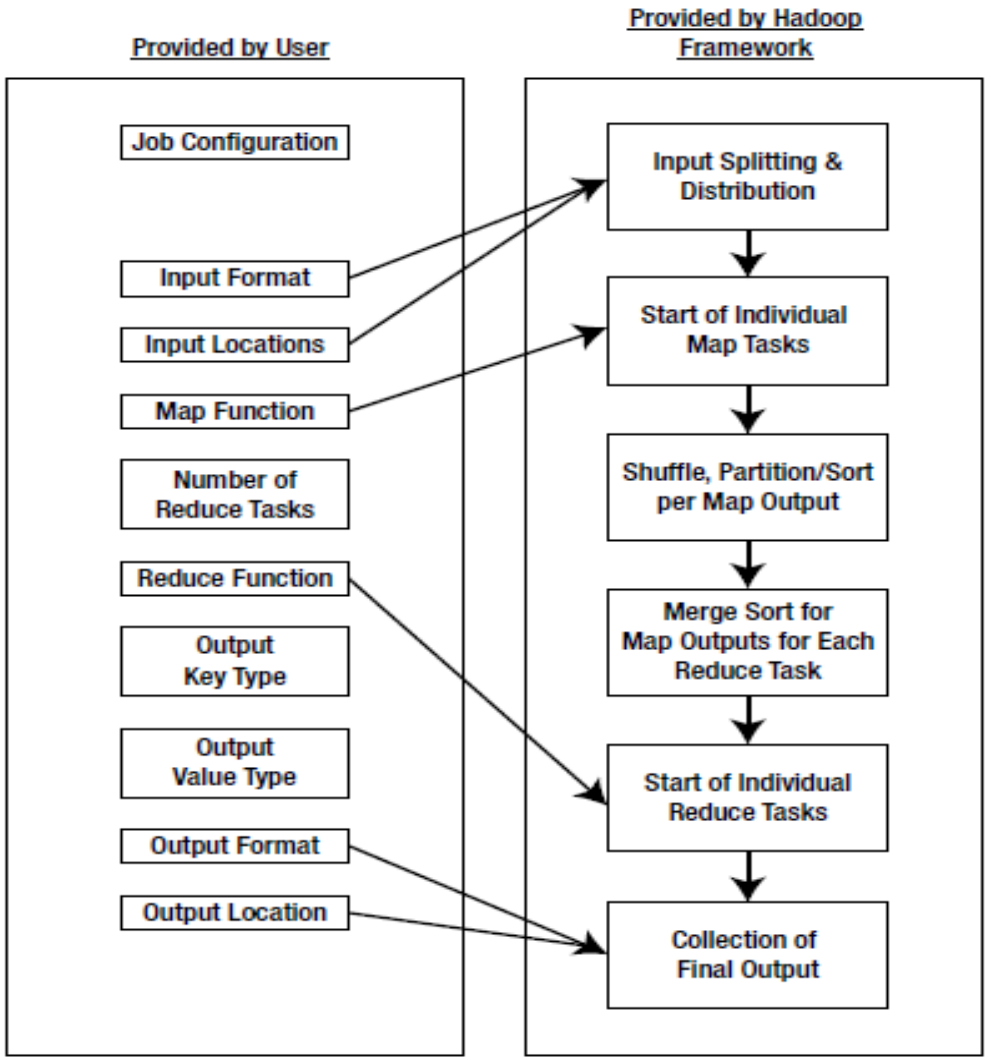
Reduce: (k2, list v2) → list (k3, v3)

Word Count – Thoughts

- Resource Requirements:
  - Use of commodity machines than super computers
  - Lot of storage space
  - Compute near data
  - Need a reliable coordinator
    - Distribute the Data
    - Coordinate the tasks
- Challenges:
  - A task on machine can crash, A task on machine can run very slow, A machine can crash
  - There may be data loss while data being moved
  - Coordinator has to be very reliable
  - Corpus division amongst nodes needs to be fairly equal

WORD COUNT WAS A SIMPLE PROBLEM! IF SOMEBODY TAKES CARE OF THESE STANDARD THINGS, THEN IT CAN BE STILL KEPT SIMPLE

Parts Of MapReduce Job



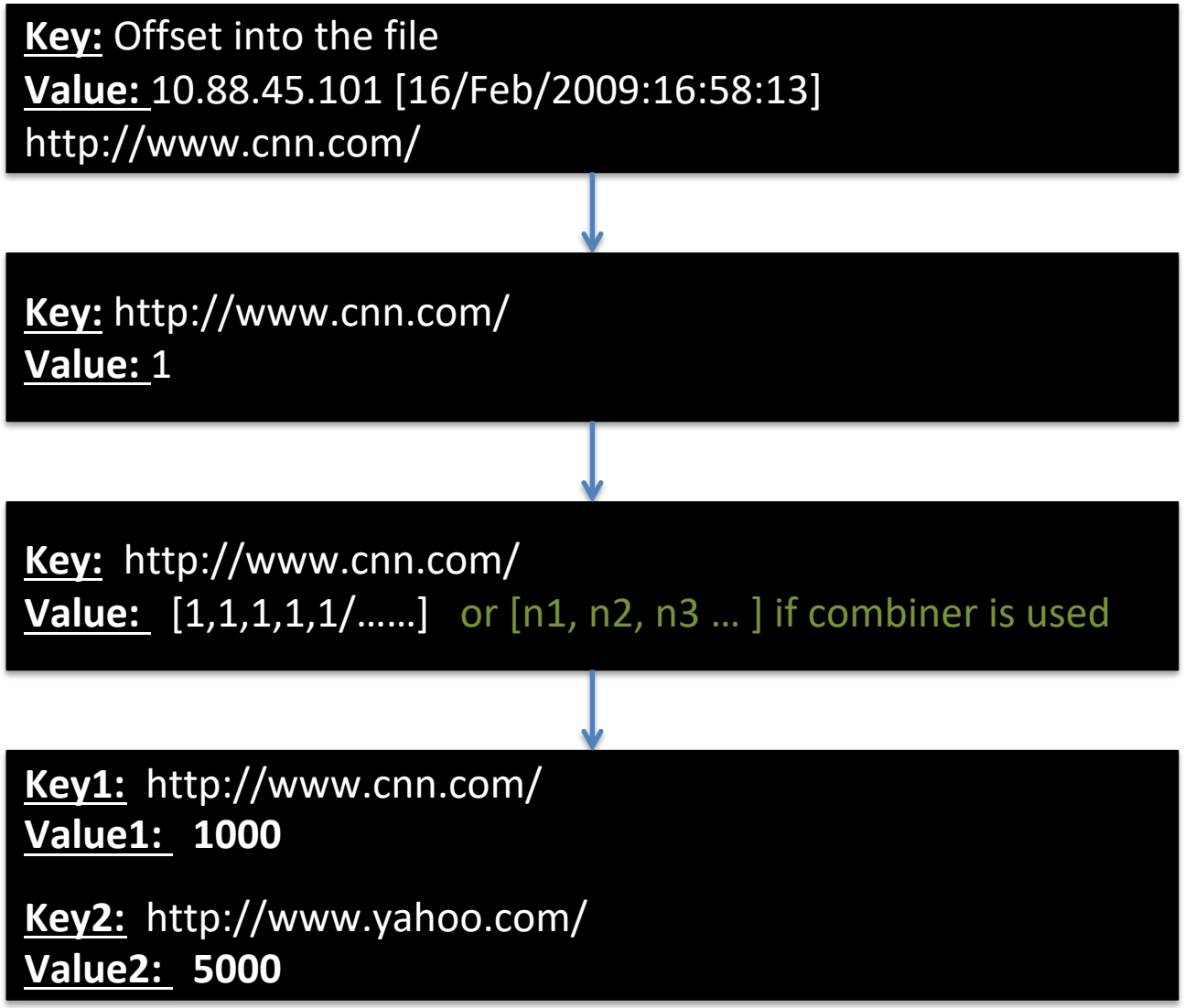
Log Processing

```
.....
10.77.25.72 [16/Feb/2009:14:29:27] http://www.google.com/
10.77.25.72 [16/Feb/2009:14:29:27] http://www.rediff.com/
...
10.88.45.101 [16/Feb/2009:16:58:13] http://www.cnn.com/
10.88.45.101 [16/Feb/2009:16:58:13] http://www.yahoo.com/
10.77.45.82 [17/Feb/2009:09:35:42] http://news.cnet.com/
10.88.45.101 [18/Feb/2009:16:03:07] http://www.cnn.com/
....
10.88.45.101 [18/Feb/2009:16:03:07] http://www.yahoo.com/
10.88.88.69 [09/Jun/2009:15:43:50] http://www.cnn.com/
10.88.88.69 [11/Jun/2009:16:11:25] http://news.cnet.com/
10.88.88.69 [11/Jun/2009:16:11:25] http://www.cnn.com/
....
10.77.224.66 [12/Jun/2009:14:41:54] http://www.rediff.com/
10.77.6.54 [12/Jun/2009:15:05:31] http://www.yahoo.com/
10.88.88.69 [19/Jun/2009:16:31:11] http://www.cnn.com/
10.77.222.22 [26/Jun/2009:15:07:46] http://www.yahoo.com/
10.77.222.22 [26/Jun/2009:15:07:50] http://www.cnn.com/
10.77.222.22 [26/Jun/2009:15:11:56] http://www.yahoo.com/
10.88.88.69 [26/Jun/2009:15:48:15] http://www.cnn.com/
10.88.88.69 [02/Jul/2009:16:11:51] http://www.yahoo.com/
.....
....
```

Log file may be in Terabytes

Find Url Frequencies

Log Processing



Input to Mapper

Output of Mapper

Input to Reducer

Output from Reducer

Next Session:
YARN Architecture