Let's talk MapReduce

Cloudwick Technologies

Agenda

- Before Map Reduce…
- Map Reduce Overview
- Word Count Problem
- Word Count Flow and Solution
- Map Reduce Flow
- Algorithms for few Simple Problems
- Algorithms for few Complex Problems

Cloudwick Technologies

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!

Cloudwick Technologies

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++

Cloudwick Technologies

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:
  - Data mining
  - Ad optimization
  - Spam detection

Cloudwick Technologies

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')

Cloudwick Technologies

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 = 0
    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),
```

Cloudwick Technologies

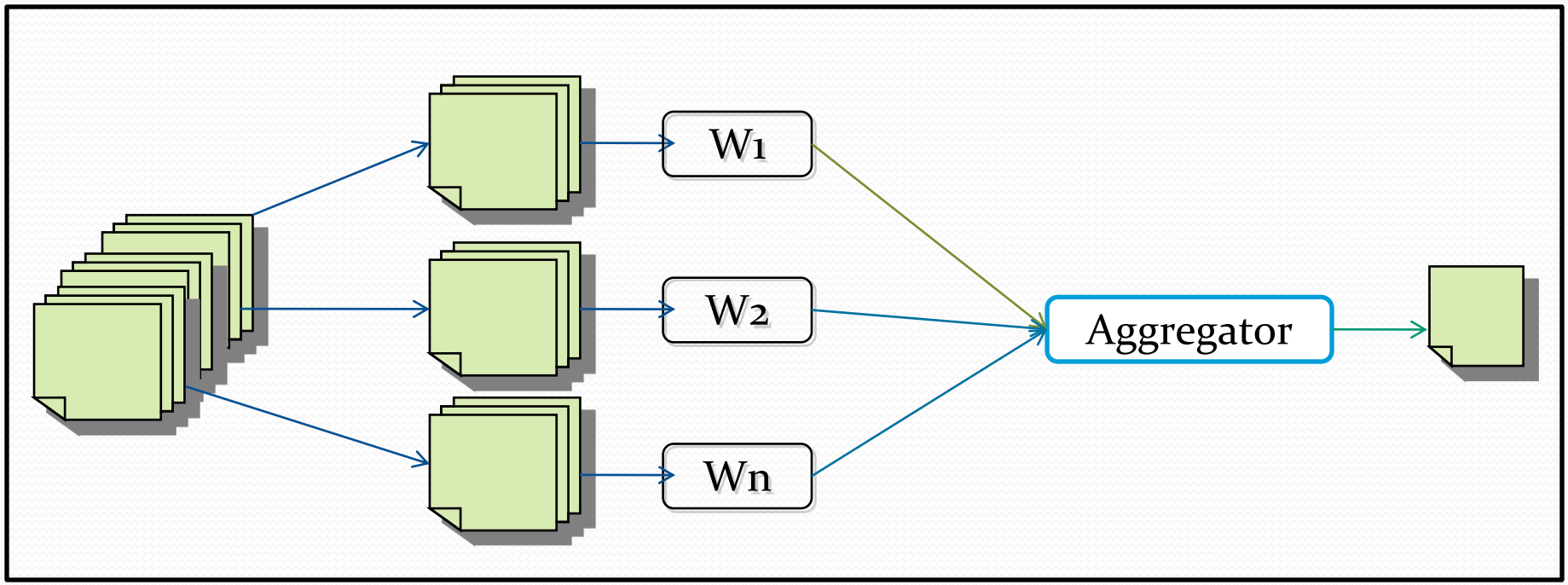
How many Maps/Reducers?

- 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*

Cloudwick Technologies

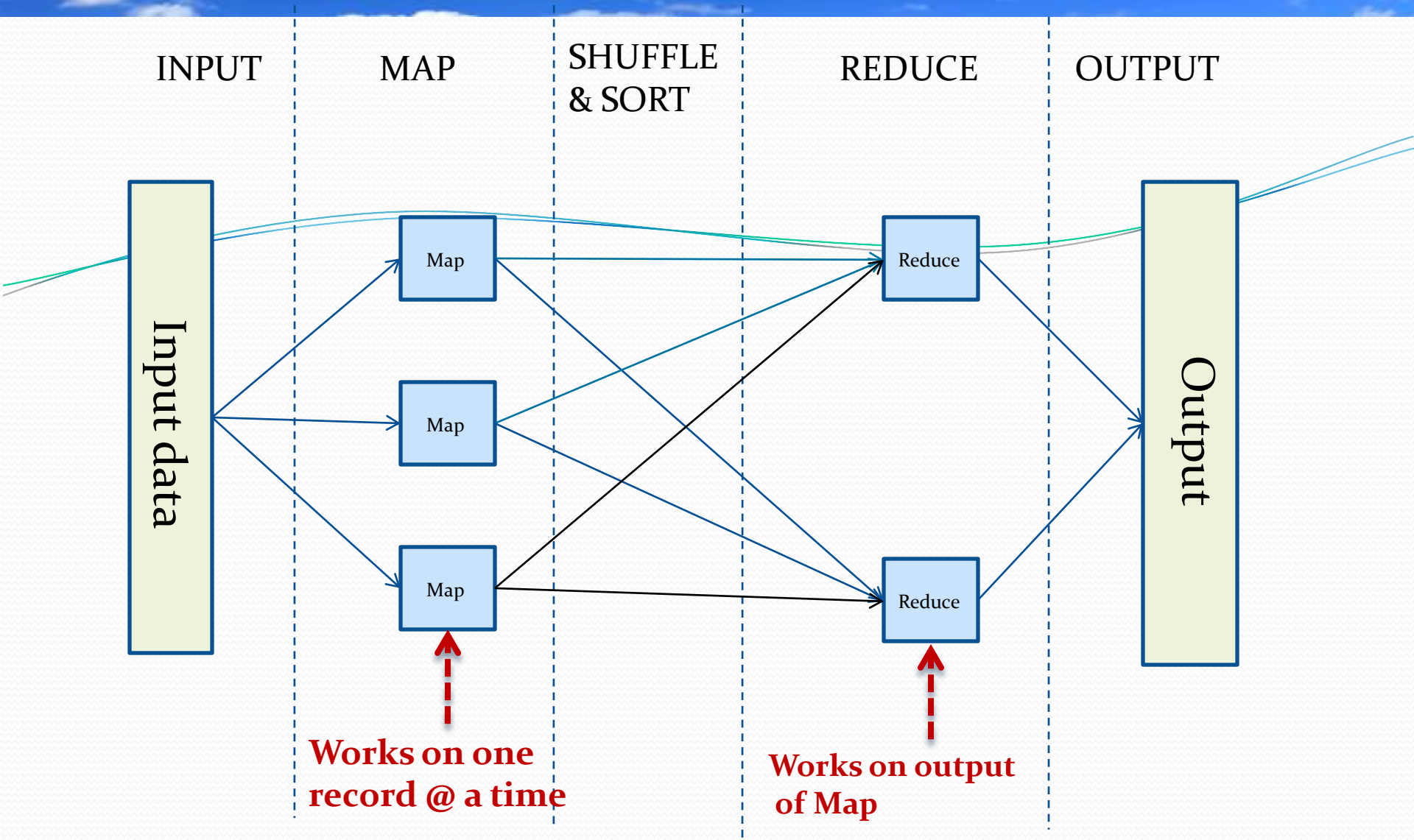
Word Count – Distributed Solution

- Problem:
  - Large quantity of documents
  - Count the number of times each distinct word occurs in the documents
- Solution:
  - Divide and Conquer



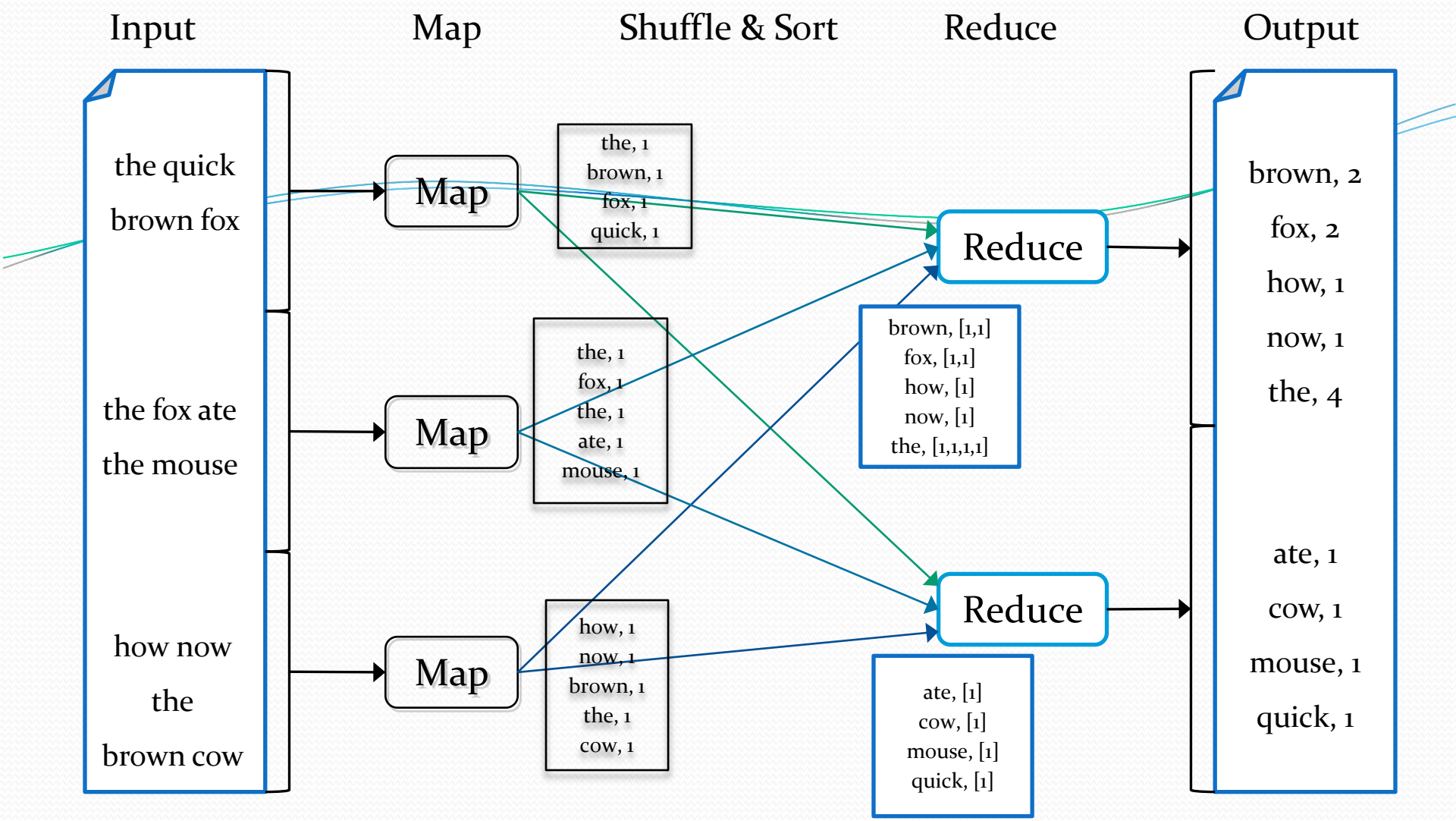
Cloudwick Technologies

Map Reduce Flow



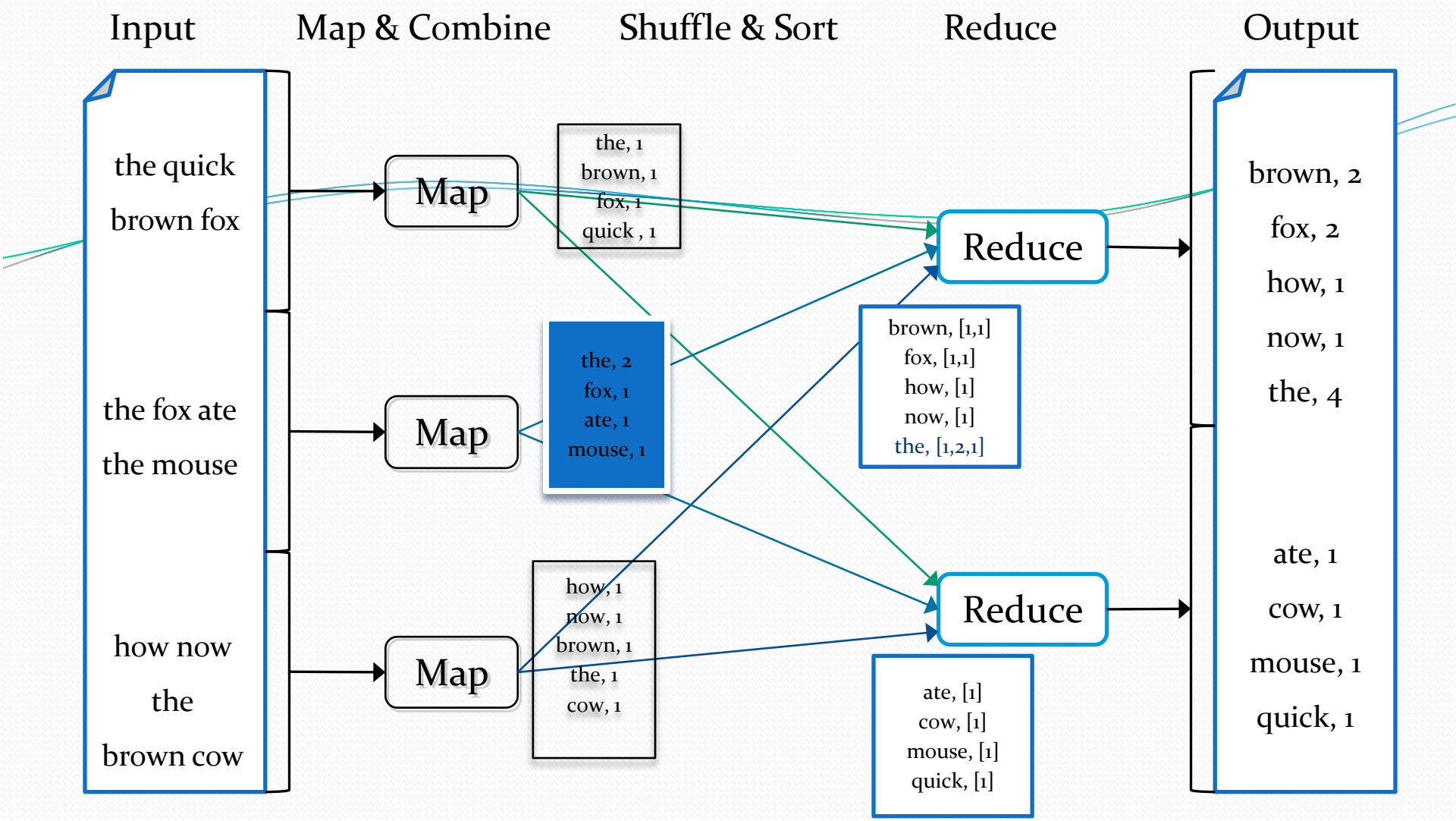
Cloudwick Technologies

Word Count – Distributed Solution



Cloudwick Technologies

Word Count – Distributed Solution



Cloudwick Technologies

Word Count – Distributed Solution

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

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

Map: $(k_1, v_1) \rightarrow \text{list}\ (k_2, v_2)$

Reduce: $(k_2, \text{list}\ v_2) \rightarrow \text{list}\ (k_3, v_3)$

Cloudwick Technologies

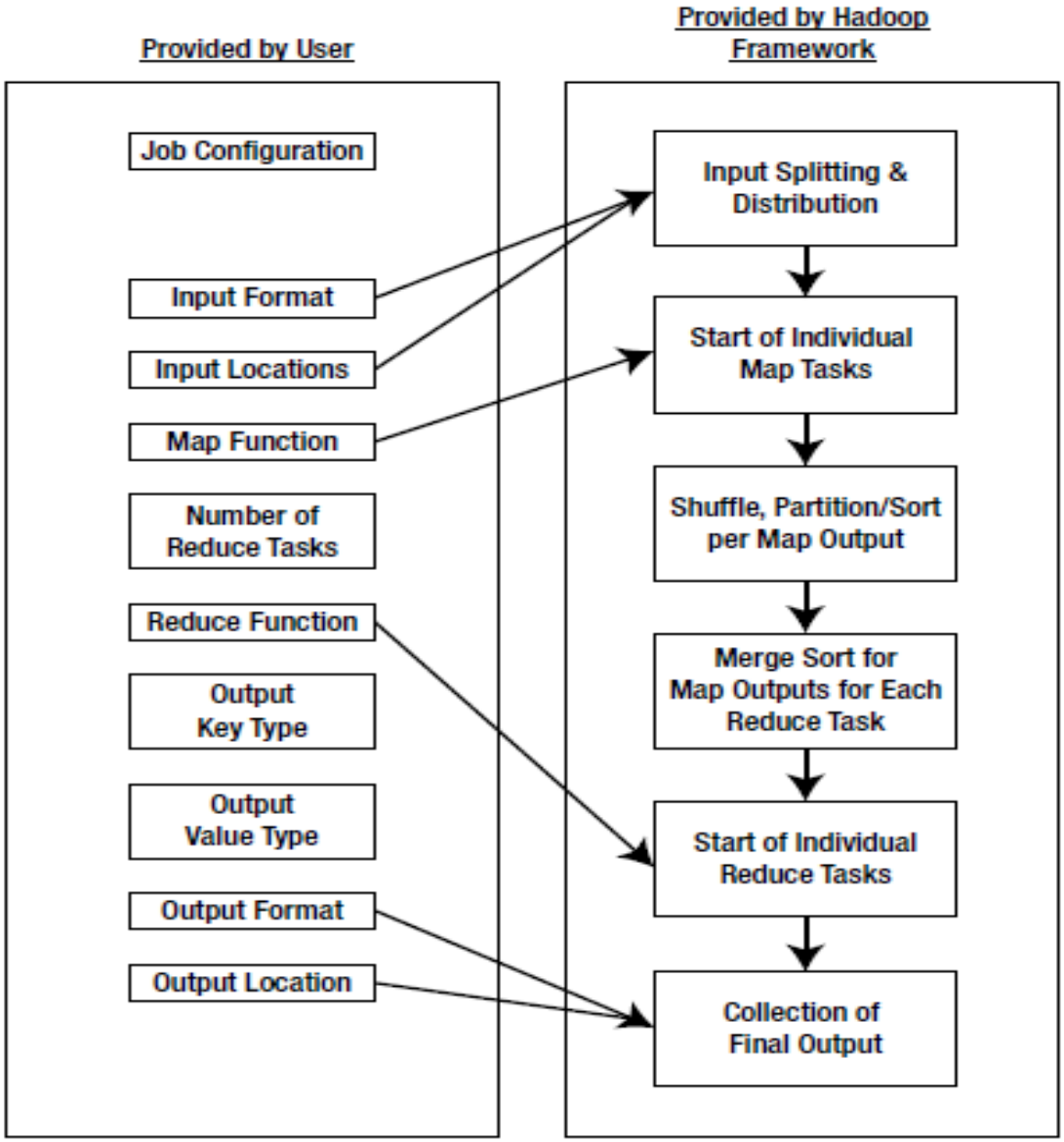
Word Count - Thoughts

- 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

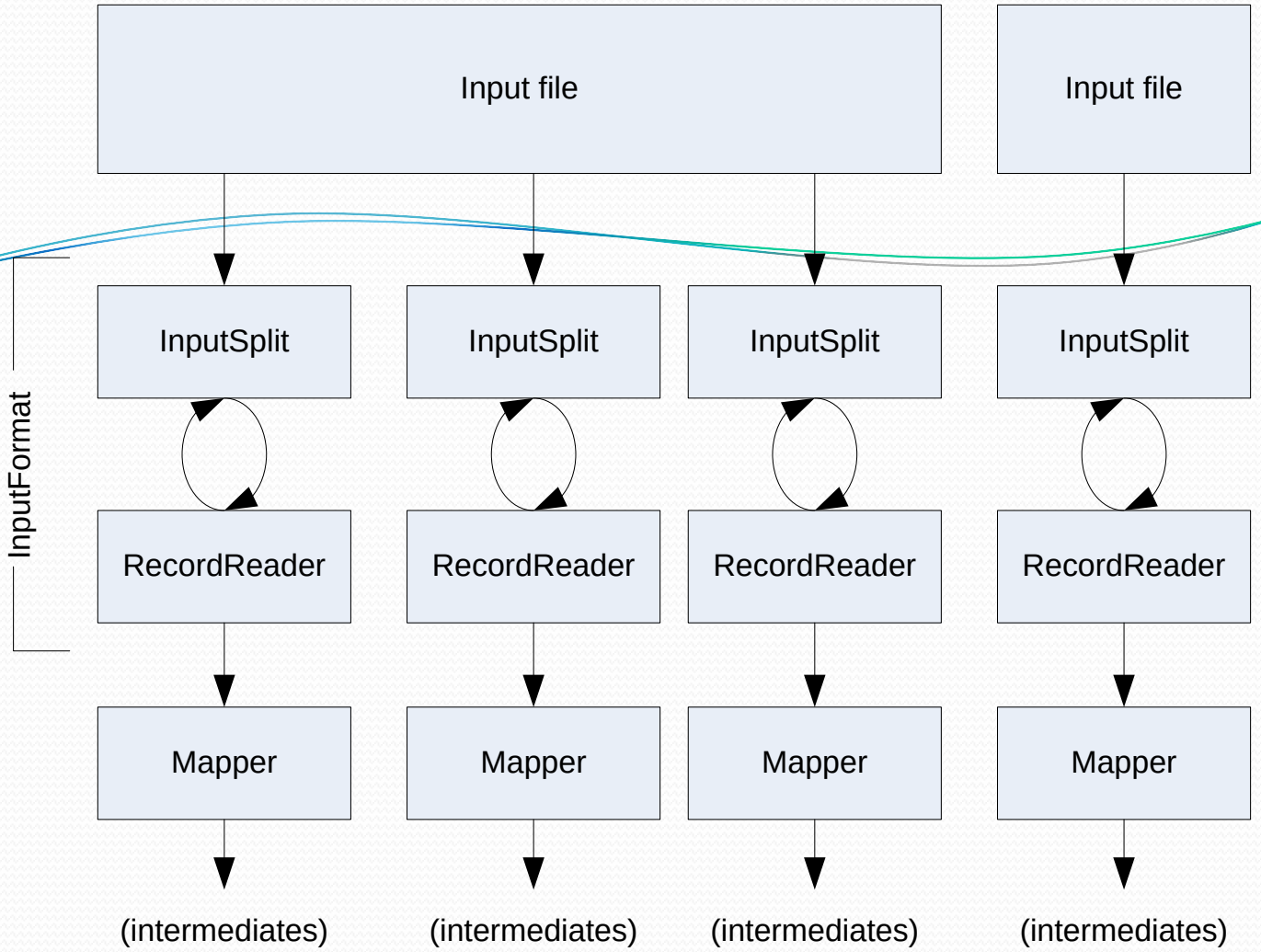
Cloudwick Technologies

Parts of MapReduce Job



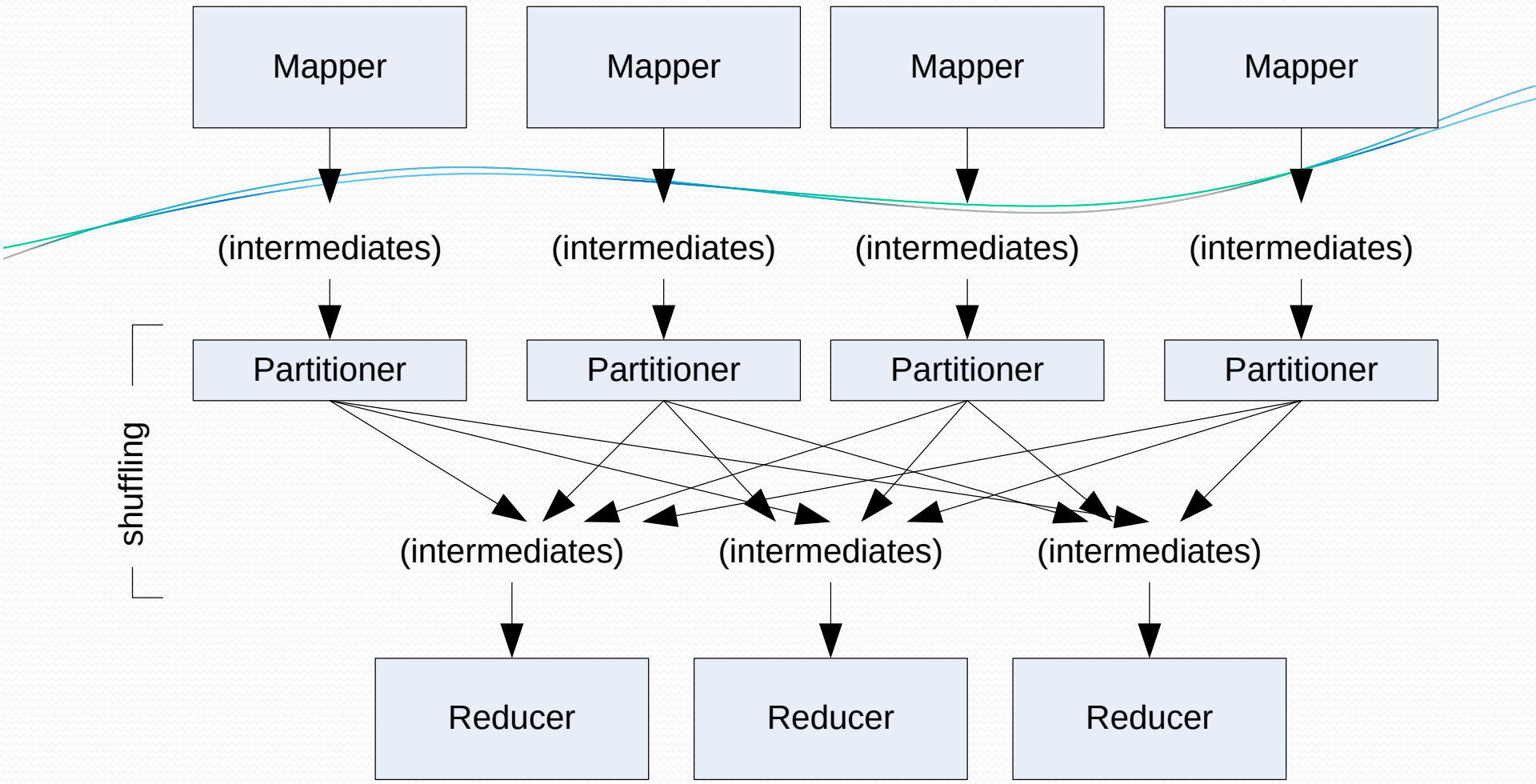
Cloudwick Technologies

MapReduce Flow – Step1



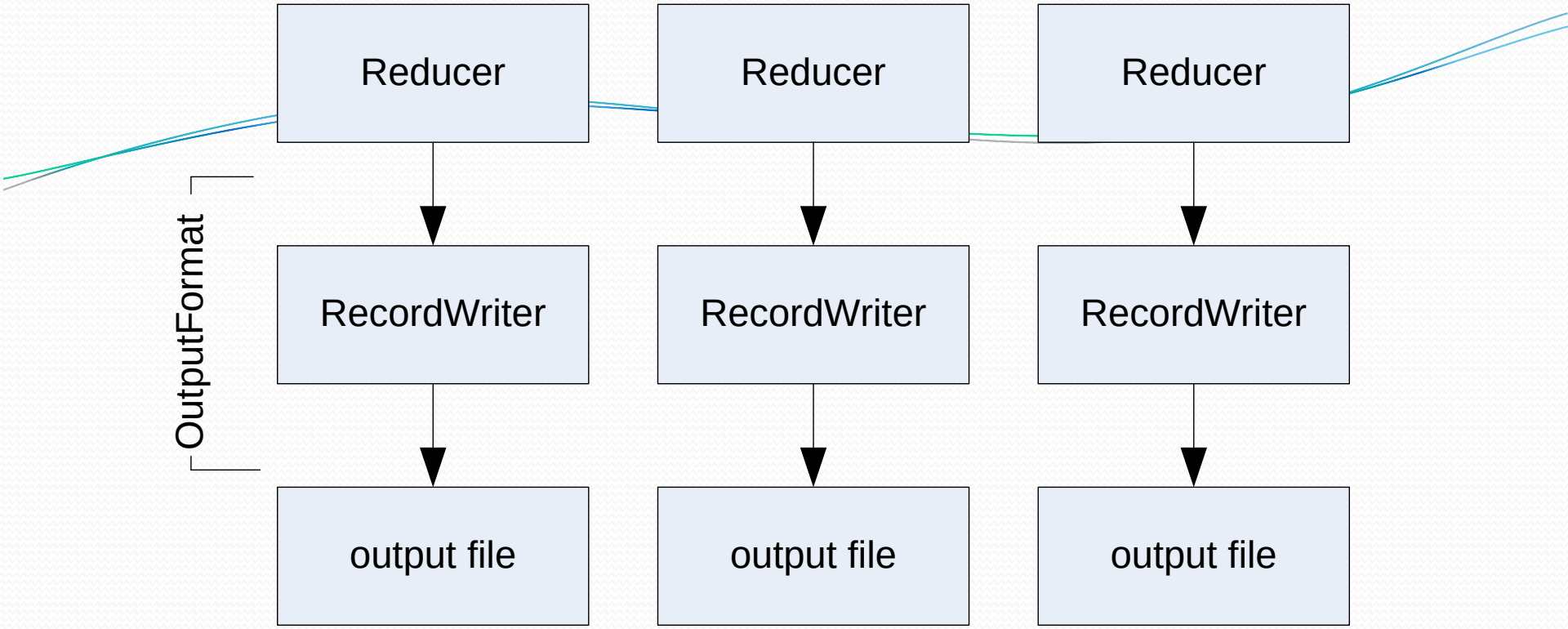
Cloudwick Technologies

MapReduce Flow – Step2



Cloudwick Technologies

MapReduce flow – Step 3



Cloudwick Technologies

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

Cloudwick Technologies

Log processing

Key: Offset into the file
Value: 10.88.45.101 [16/Feb/2009:16:58:13] http://www.cnn.com/

Input to Mapper

Key: http://www.cnn.com/
Value: 1

Output of Mapper

Key: http://www.cnn.com/
Value: [1,1,1,1,1/……] or [n1, n2, n3 …] if combiner is used

Input to Reducer

Key1: http://www.cnn.com/
Value1: **1000**

Key2: http://www.yahoo.com/
Value2: **5000**

Output from Reducer

Cloudwick Technologies

Search

- **Input:** (lineNumber, line) records
- **Output:** lines matching a given pattern

- **Map:**

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

- **Reduce:** identify function
  - Alternative: no reducer (map-only job)

Cloudwick Technologies

Sort

- **Input:** (key, value) records
- **Output:** same records, sorted by key

- **Map:** identity function
- **Reduce:** identify function

- **Trick:** Pick partitioning function h such that

  $k_1 < k_2 \Rightarrow h(k_1) < h(k_2)$

Cloudwick Technologies

Inverted Index

- **Input:** (filename, text) records
- **Output:** list of files containing each word

- **Map:**

```
foreach word in text.split():
    output(word, filename)
```

- **Combine:** uniquify filenames for each word

- **Reduce:**

```
def reduce(word, filenames):
    output(word, sort(filenames))
```

Cloudwick Technologies

Most Popular words

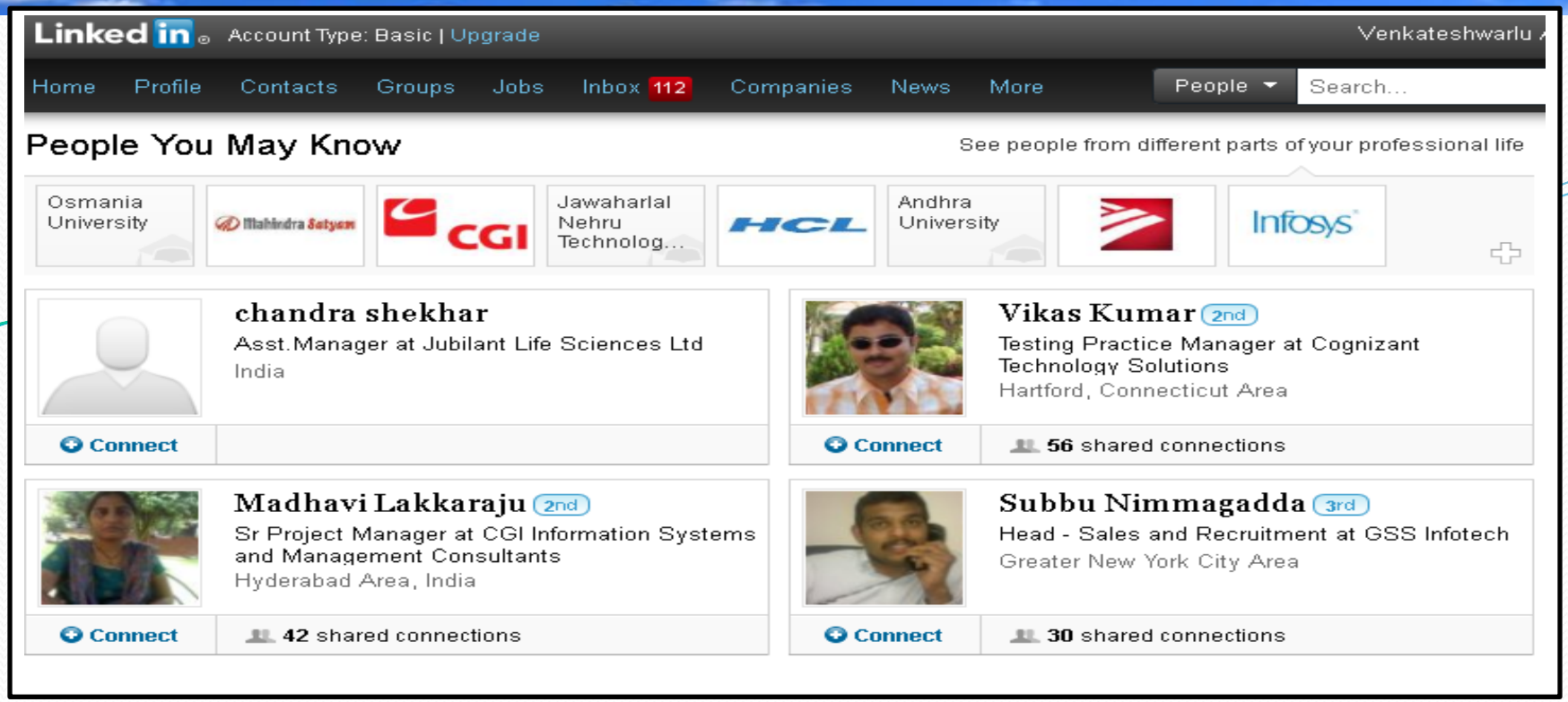
- **Input:** (filename, text) records
- **Output:** the 100 words occurring in most files

- Two-stage solution:
  - **Job 1:**
    - Create inverted index, giving (word, list(file)) records
  - **Job 2:**
    - Map each (word, list(file)) to (count, word)
    - Sort these records by count as in sort job

- Optimizations:
  - Map to (word, 1) instead of (word, file) in Job 1
  - Estimate count distribution in advance by sampling

Cloudwick Technologies

Recommendation System



Input: 50M users and their connections

TB's of input data

Output: List(User, List(Top N probable connections))

Insight: You might know Vikas if a lot of people you know, know Vikas

Cloudwick Technologies

Recommendation System

```
// Input: UserList:      List (username, List(Connections))

For each user "U" in UserList:   //50 Million users
    For each connection "X" in connections(U):  // 100 Connections
            For each connection "Y" in connections(X):  // 100 Connections
                    if ("Y" not in Connections(U)):  //500 Billion iterations
                            Count(U, Y)++;

    Sort (U, Y) order by Count (U, Y)

     Store top N  as (U, {Yo, Y1, …., Yn})
```

Cloudwick Technologies

Recommendation System – MapReduce

"Ankam" => ["Ravi", "Kiran", "Sameer", "Sandeep"]
"Ravi" => ["Ankam", "Tushar", "Mukund", "Krishna"]
"Sandeep" => ["Anil", "Ankam", "Mukund", "Sameer"]

Sample Input Record: ***"Ankam" => ["Ravi, "Kiran," "Sameer", ...]***

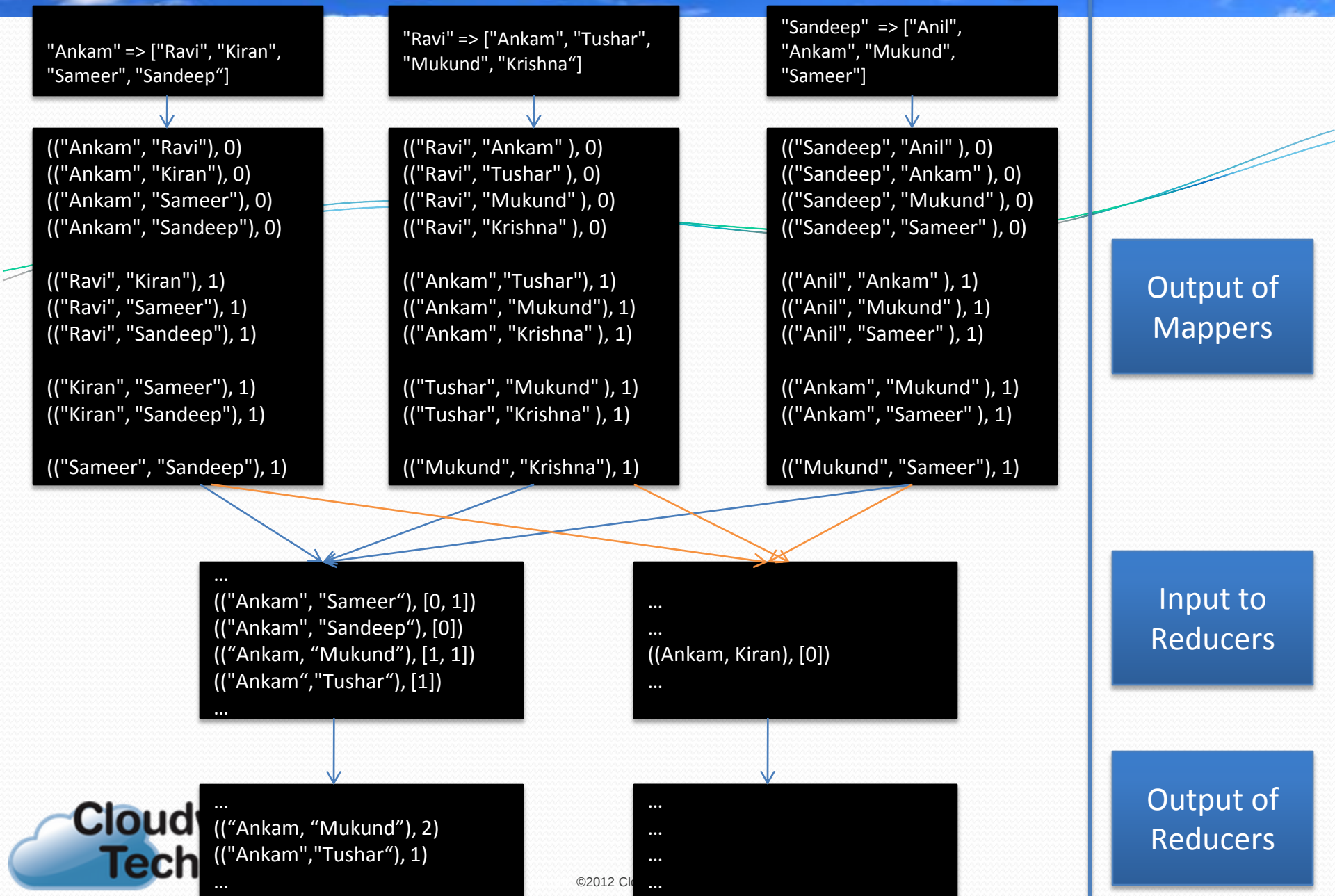
Job1:

- Map:
  - Input: (Person, Connection_list)
  - Output: PersonPair: ((Person, Friend1), 0) and ((Friend1, Friend2), 1)
- Reduce:
  - Input: (Person, Friend1), List (degree)
  - Sum all frequencies for Person_Pair – Ignore if "0"
  - Output: ((Person1, Person2), Path_count)

Sample Output Record: ***("Ankam", "Mukund") => 10***

Cloudwick Technologies

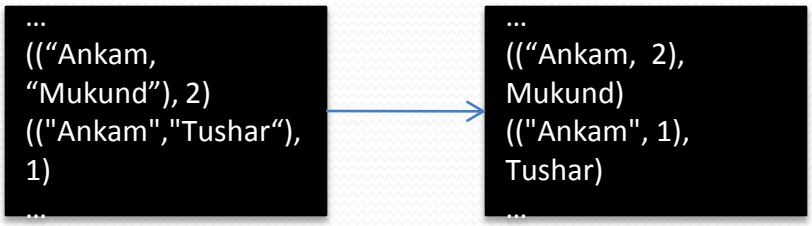
Recommendation System



Recommendation System

Job2:

- Map: Just rearrange the records
  - Input: ((Person_Pair), path_count)
  - Output: ((Person1, path_count), Person2)
- Partition on first element of the key
- Reduce:
  - Input: (Connection_count_pair, candidate_list)
  - All the connections for a person are sorted by path count. Just take the top N and output
  - Output: Person, List (Friend, Count)



Since Partitioning is on first element of Key – All records for a Person will be given to one Reducer

Cloudwick Technologies

TF-IDF

- Term frequency–Inverse document frequency
- TF-IDF measure used to evaluate how important a word is to a document in a collection
- Used in **information retrieval** and **text mining**

TF : Number of times a given terms appears in a document

(Term frequency) / (Size of the doc)

IDF: Measures the general importance of the term:

Log (Total number of docs / Number of docs in which term appears)

Cloudwick Technologies

Why IDF?

- Query: *Details on Hadoop*: 194,000 results (194K)
  - **Details**: 1,300,000,000 (1.3B results)
  - **On**: 11,340,000,000 (**11.34B** results)
  - **Hadoop**: 1,810,000 (1.8M results)

Rare words (for a given doc collection) in query should be given more weightage

Cloudwick Technologies

TF-IDF Algo

// **Input: List(Docs)**

Foreach Doc in Input :

Tokenize and Get the words

For each Word:

Count number of times word occurs

// Result: TF = ((Word, DocID), TF)

Group by words and count

// Result: List (word, (List(DocID), count))

Compute IDF and then Compute TF*IDF

Cloudwick Technologies

TF-IDF MapReduce

- **Job1: Word Frequency in Doc and Total word count for docs**
  - Mapper:
    - Input: (docname, content)
    - Output: [(word, docname), (n, N)]
  - Reducer:
    - Identity Reducer

n : Number of times a term appears in given Doc
N : Total number of words in given doc

Doc1
… Hadoop …
Details…. On

Doc2
…. Details on
Cluster …

Docn
Hadoop…
On….

Input

[(Hadoop, Doc1), (1, 1000)]
[(Details, Doc1), (10,1000)]
[(On, Doc1), (100, 1000)]
[(Details, Doc2), (200, 2000)]
….

[(Hadoop, Doc10), (5, 5000)]
[(Details, Doc10), (50,5000)]
[(On, Doc25), (500, 5000)]
[(Details, Doc25), (400, 4000)]
….

Output

Cloudwick Technologies

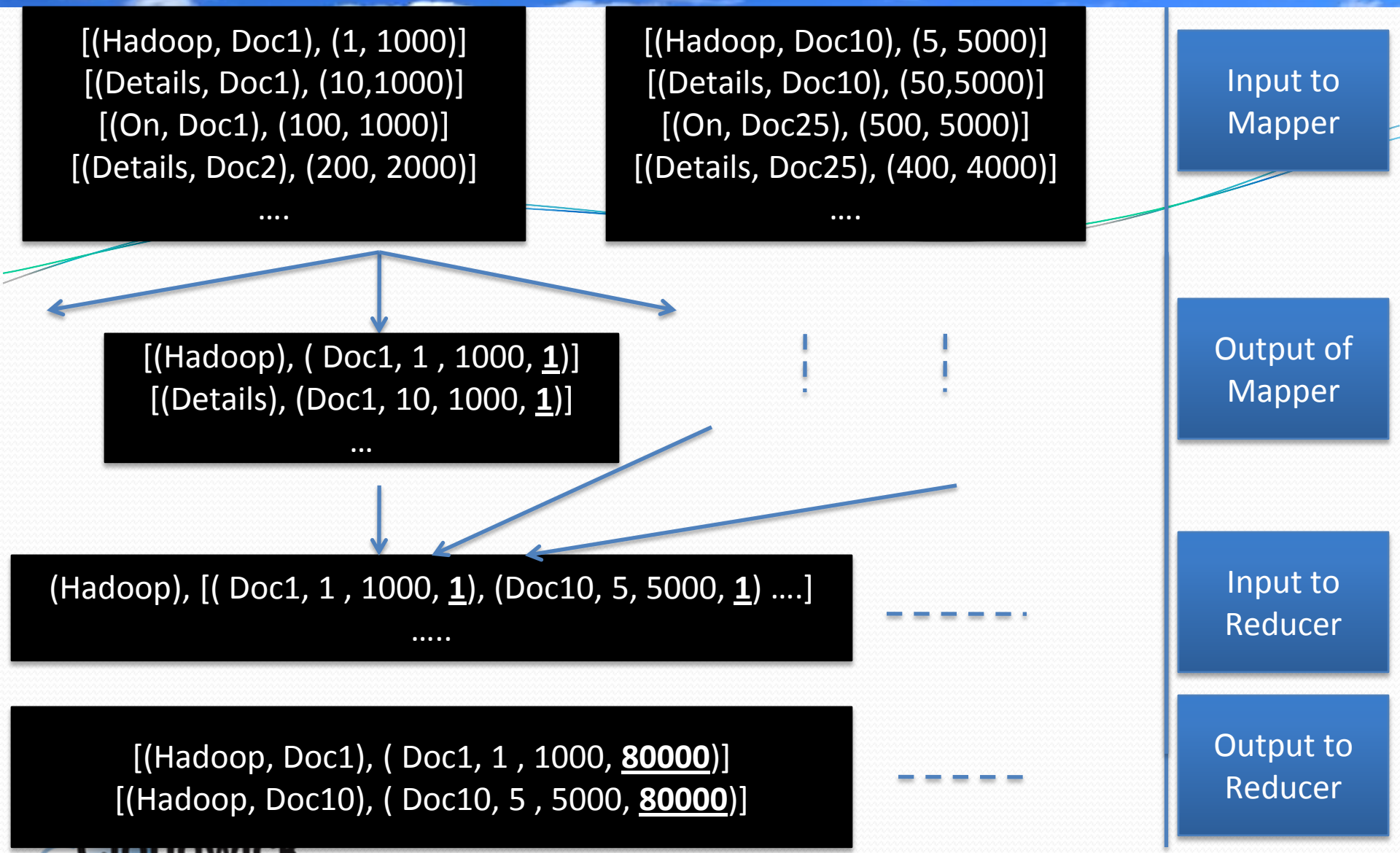
TF-IDF MapReduce

- **Job2: Word Frequency in corpus**
  - Mapper:
    - Input: ((word, docname), (n,N))
    - Output: (word, (docname, n, N, 1))
  - Reducer:
    - Input: (word, [(docname1, n1, N1, 1), (docname2, n2, N2, 1) .....])
    - Add-up count for word in corpus
    - Output: (word, document), (n,N,m))

n : Number of times a term appears in given Doc
N : Total number of words in given doc
m : Number of documents in which word occurs

Cloudwick Technologies

TF-IDF MapReduce



Cloudwick Technologies

TF-IDF MapReduce

- **Job3: Calculate TF-IDF**
  - Mapper:
    - Input: (word, document), (n,N,m))
    - Output: ((word, docname), TF*IDF)
  - Reducer:
    - Do Nothing: Identity Function

TF = (Term frequency) / (Size of the doc)

IDF = Log (Total number of docs / Number of docs in which term appears)

n : Number of times a term appears in given Doc
N : Total number of words in given doc
m : Number of documents in which word occurs

Cloudwick Technologies

Questions?