MapReduce

MapReduce

- MapReduce Architecture
- MapReduce Internals
- MapReduce Examples
- JobTracker Interface

MapReduce: A Real World Analogy

Coins Deposit

MapReduce: A Real World Analogy

Coins Deposit

DEPOSIT

Coins Counting Machine

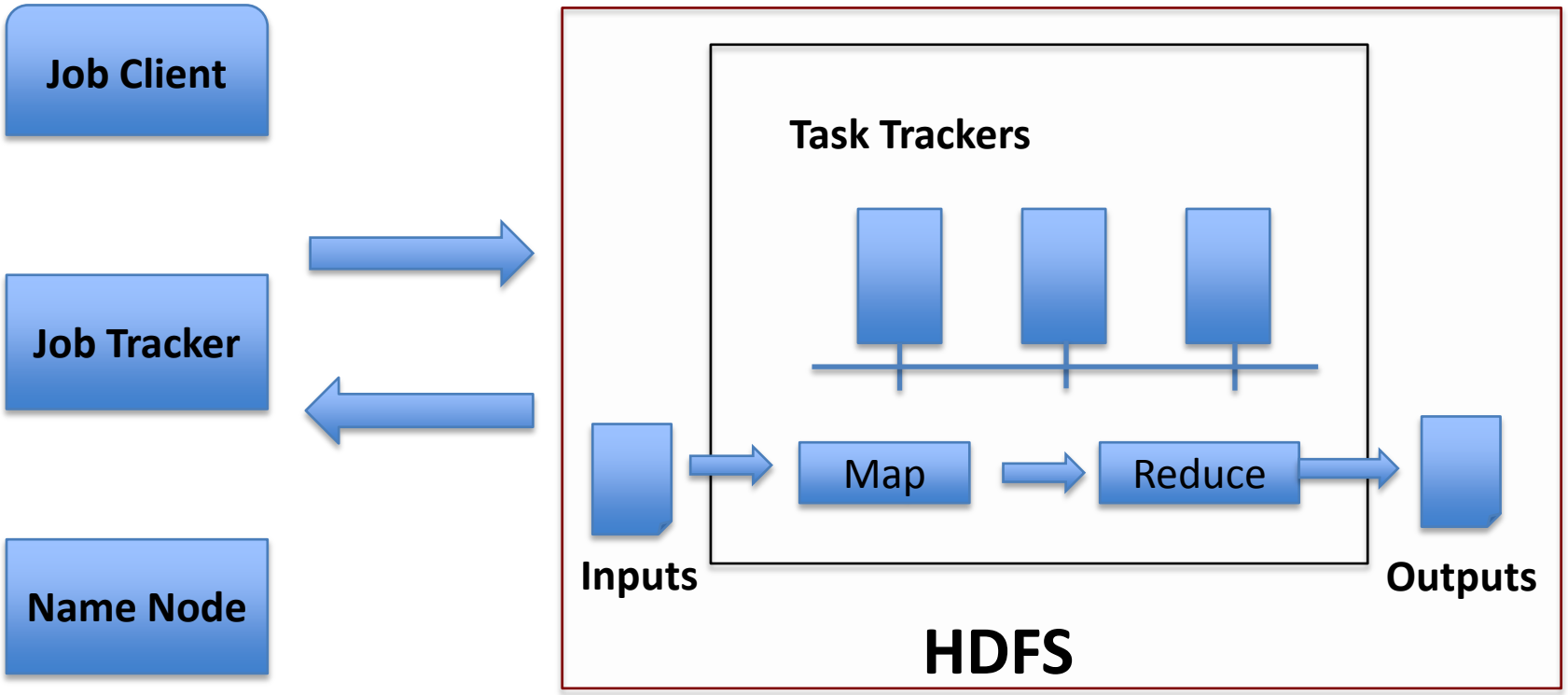
MapReduce: A Real World Analogy

Coins Deposit

Mapper: Categorize coins by their face values

Reducer: Count the coins in each face value ***in parallel***

MapReduce Architecture: Master-Slaves



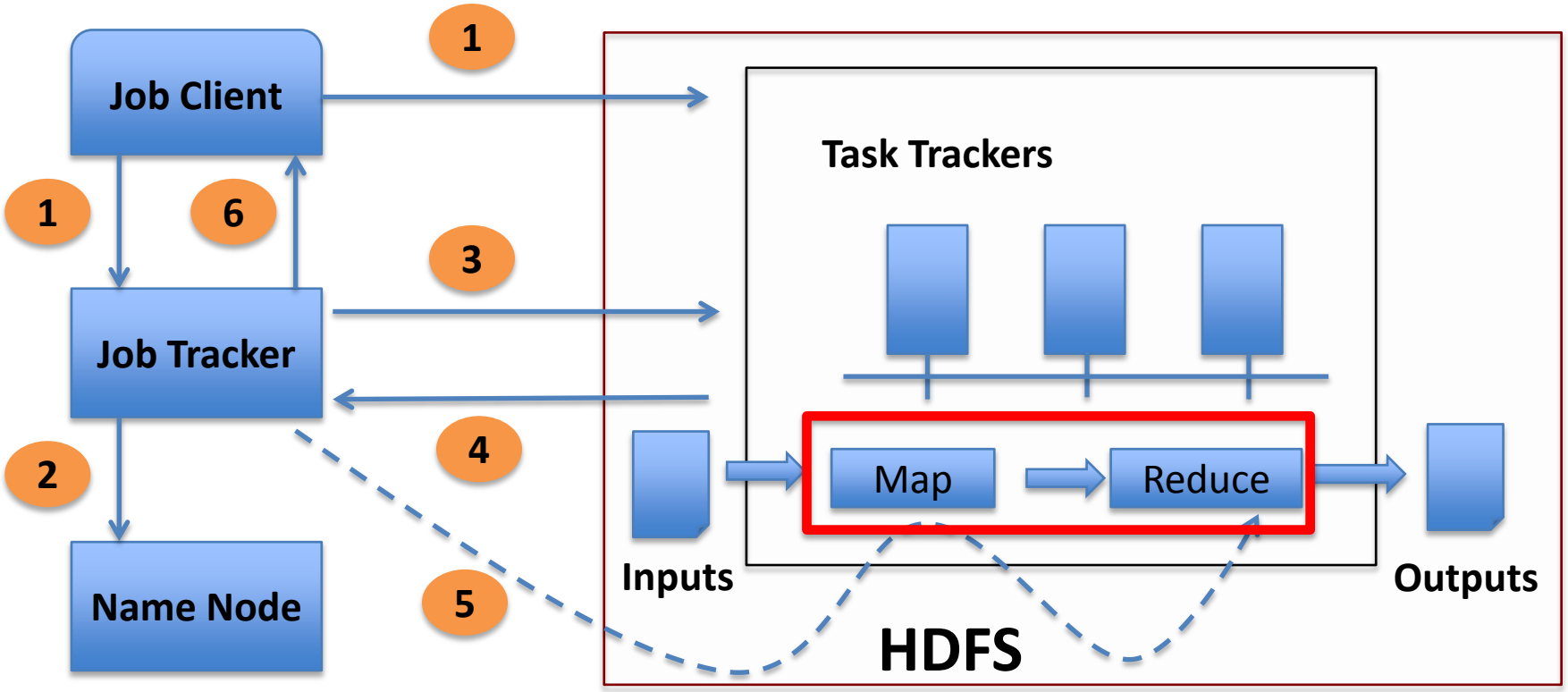
Job Client: Submit Jobs

Task Tracker: Execute Jobs

Job Tracker: Coordinate Jobs (Scheduling, Phase Coordination, etc.)

Job: MapReduce Function+ Config

MapReduce Architecture: Workflow



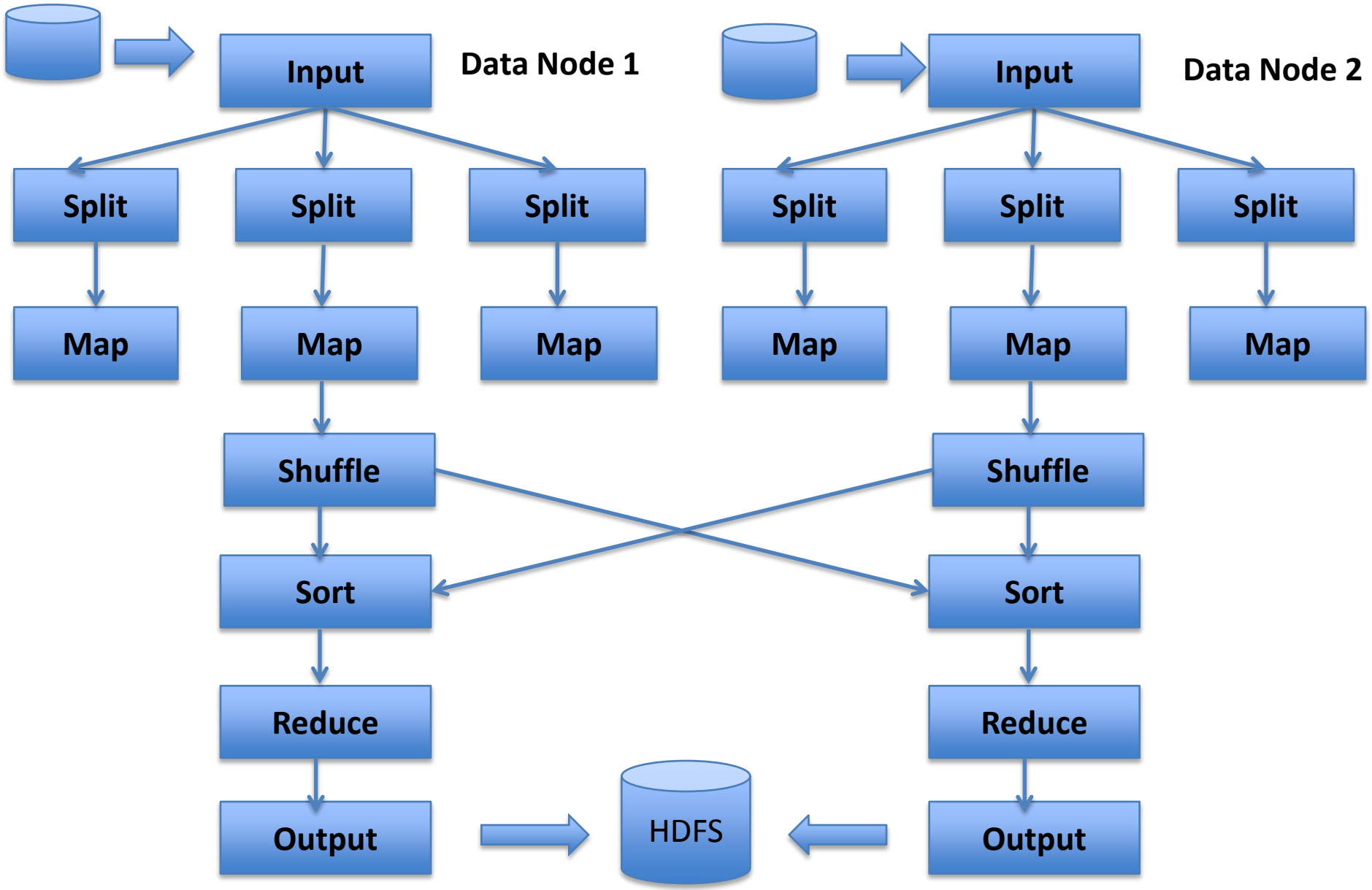
1. Client submits job to Job Tracker and copy code to HDFS
2. Job Tracker talks to NN to find data it needs
3. Job Tracker creates execution plan and submits work to Task Trackers

4. Task trackers do the job and report progress/status to Job Tracker
5. Job Tracker manages task phases
6. Job Tracker finishes the job and updates status

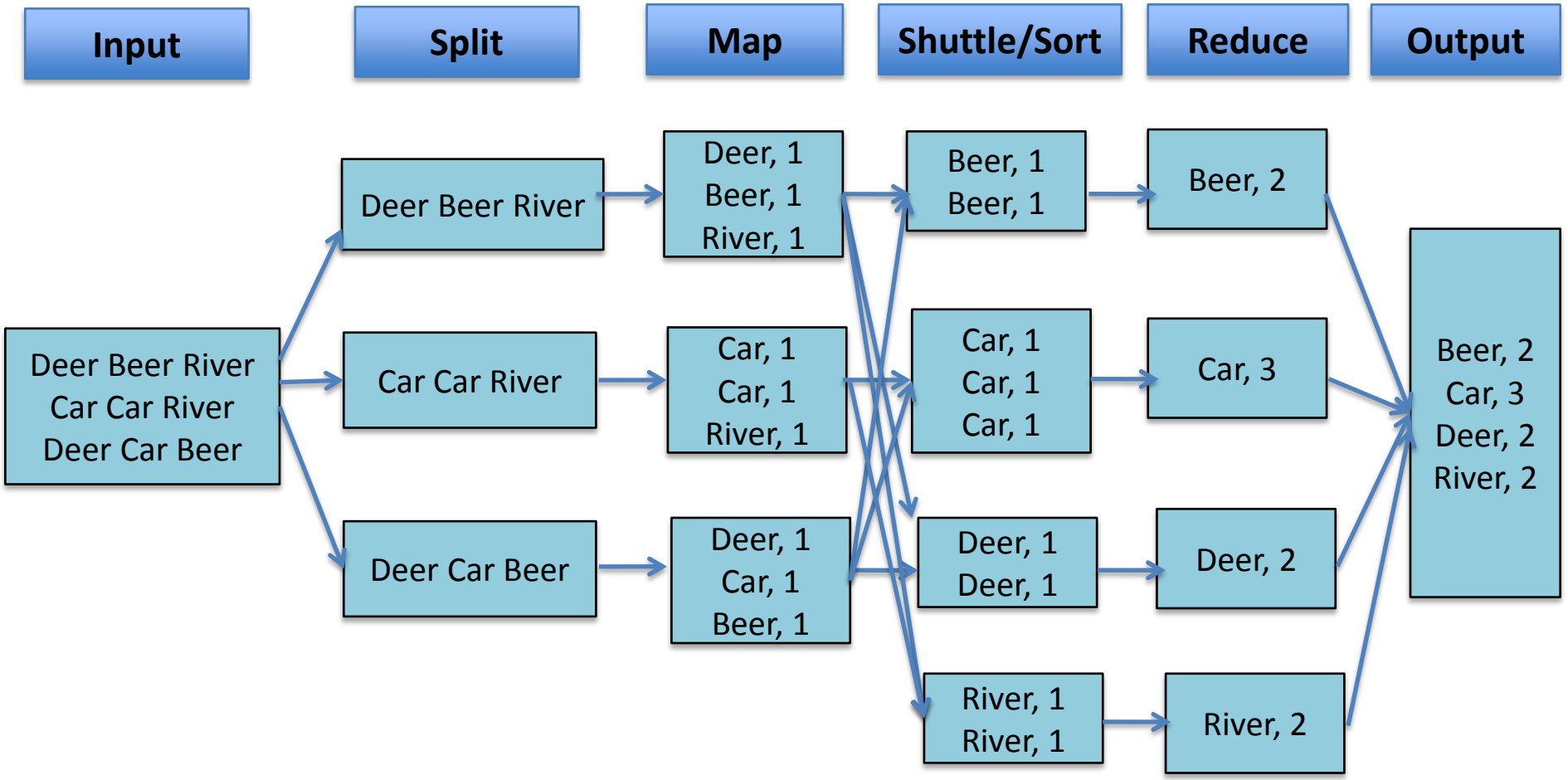
MapReduce Paradigm

- Implement two functions:
  - **Map** (k1,v1) -> list (k2, v2)
  - **Reduce**(k2, list(v2)) -> list (v3)
- Framework handles everything else
- Value with the **same key** go to the same reducer

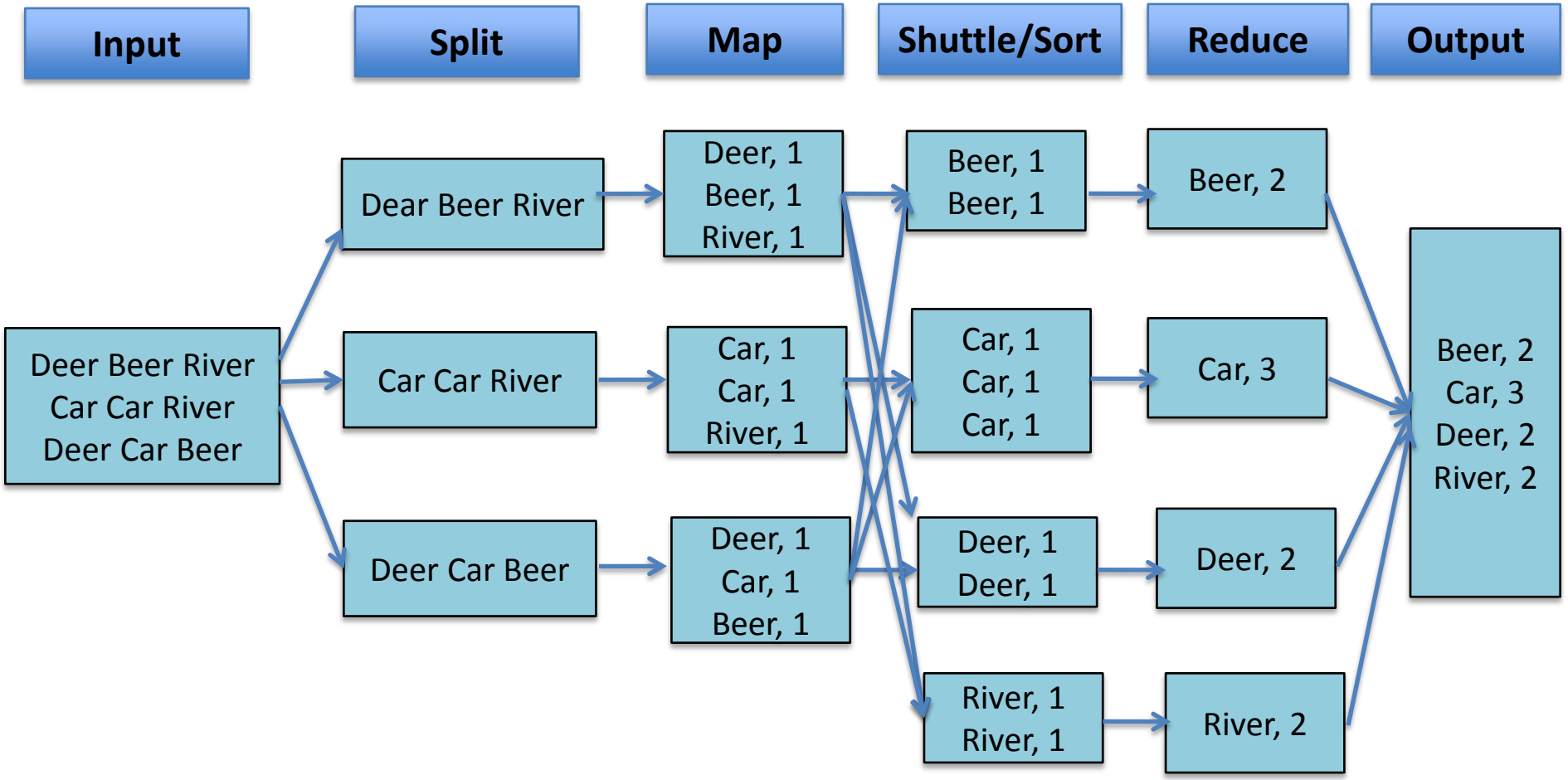
MapReduce Internal



MapReduce Example: Word Count



MapReduce Example: Word Count



Q: What are the Key and Value Pairs of Map and Reduce?

Map: Key=word, Value=1

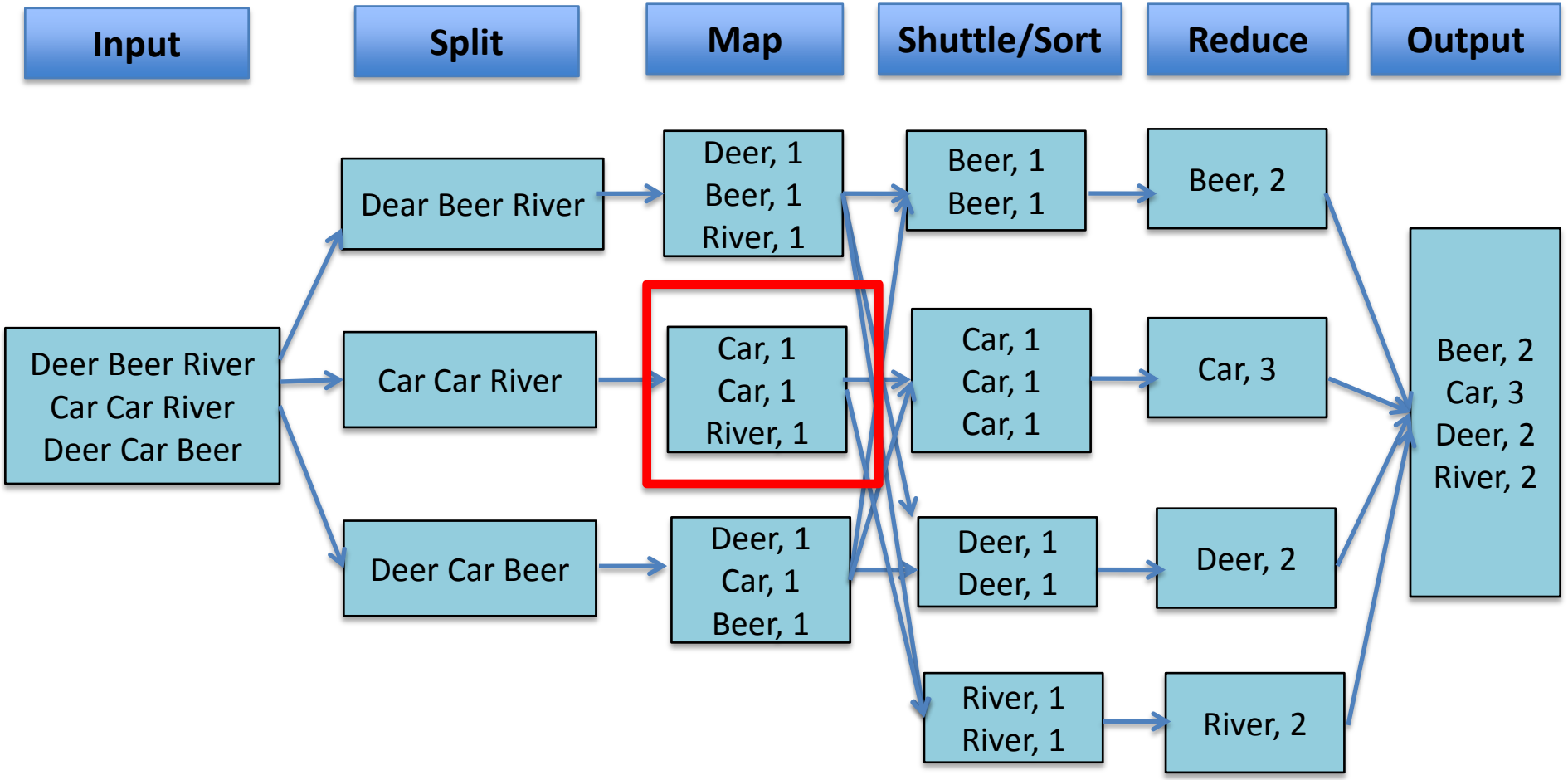
Reduce: Key=word, Value=aggregated count

Mapper and Reducer of Word Count

- Map(key, value){
  // key: line number
  // value: words in a line
  for each word w in value:
    Emit(w, "1");}
- Reduce(key, list of values){
  // key: a word
  // list of values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(key, result);}

Combiner is the same as Reducer

MapReduce Example: Word Count

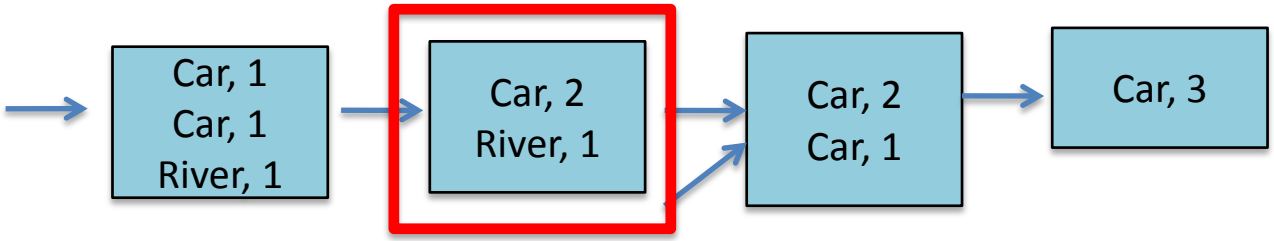


Any place we can improve the efficiency?

Local aggregation at mapper will be able to improve MapReduce efficiency.

MapReduce: Combiner

- **Combiner:** do local aggregation/combine task at mapper



- **Q: What are the benefits of using combiner:**
  - Reduce memory/disk requirement of Map tasks
  - Reduce network traffic
- **Q: Can we remove the reduce function?**
  - No, reducer still needs to process records with same key but from *different mappers*
- **Q: How would you implement combiner?**
  - It is the same as Reducer

MapReduce WordCount 2

- **New Goal:** output all words sorted by their frequencies (total counts) in a document.
- **Question:** How would you adopt the basic word count program to solve it?
- **Solution:**
  - **Sort words by their counts in the reducer**
  - **Problem: what happens if we have more than one reducer?**

MapReduce WordCount 2

- **New Goal:** output all words sorted by their frequencies (total counts) in a document.
- **Question:** How would you adopt the basic word count program to solve it?
- **Solution:**
  - **Do two rounds of MapReduce**
  - **In the 2nd round, take the output of WordCount as input but *switch key and value pair***
  - **Control the sorting capability of *shuffle/sort* to do the global sorting**

MapReduce WordCount 3

- **New Goal:** output the top K words sorted by their frequencies (total counts) in a document.
- **Question:** How would you adopt the basic word count program to solve it?
- **Solution:**
  - **Use the solution of previous problem and only grab the top K in the final output**

MapReduce WordCount 3

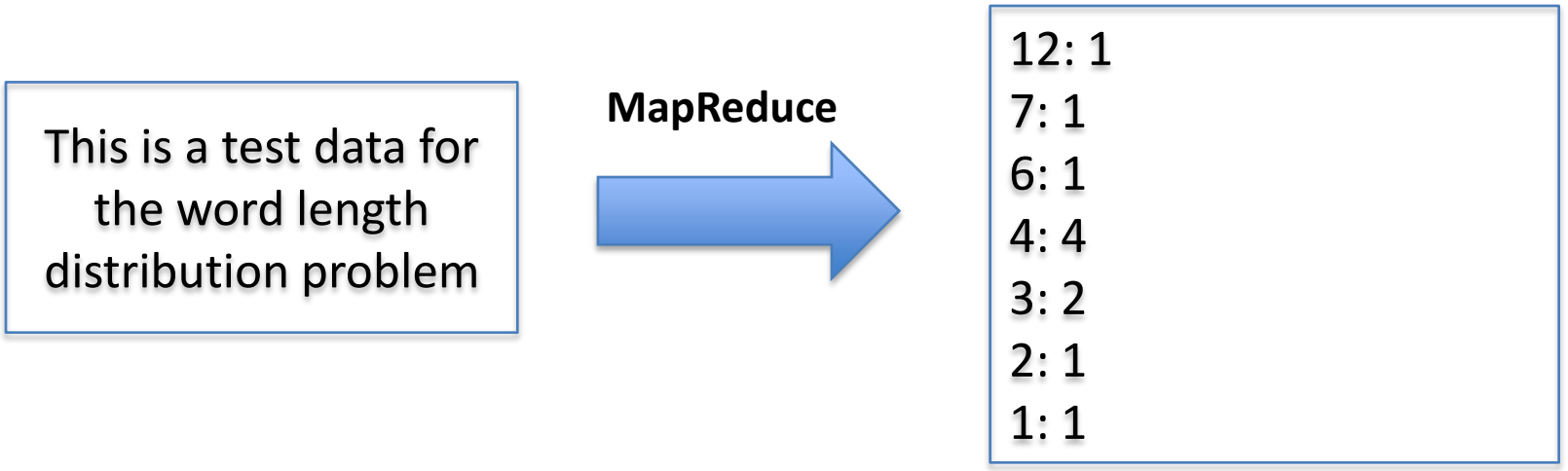
- **New Goal:** output the top K words sorted by their frequencies (total counts) in a document.
- **Question:** How would you adopt the basic word count program to solve it?
- **Solution:**
  - **Add a sort function to the *reducer* in the first round and only output the top K words**
  - **Intuition: the *global top K must be a local top K* in any reducer!**

Distributive Functions

- **Functions that can use combiner are called *distributive*:**
  - Distributive: Min/Max(), Sum(), Count(), TopK()
  - Non-distributive: Mean(), Median(), Rank()

Map Reduce Problems Discussion

- **Problem 1:** Find Word Length Distribution
- **Statement:** Given a set of documents, use Map-Reduce to find the length distribution of all words contained in the documents
- **Question:**
  - What are the Mapper and Reducer Functions?



Map Reduce Problems Discussion

- **Problem 1:** Find Word Length Distribution
- **Mapper and Reducer:**
  - Mapper(document)

    { Emit (**Length(word), word**) }
  - Reducer(output of map)

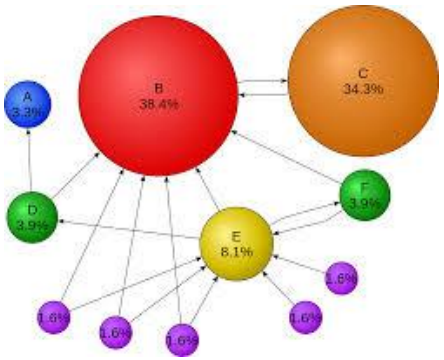
    { Emit (Length(word), **Size of (List of words at a particular length))**}

Mapper and Reducer of Word Length Distribution

- Map(key, value){
  // key: document name
  // value: words in a document
  for each word w in value:
    Emit(length(w), w);}
- Reduce(key, list of values){
  // key: length of a word
  // list of values: a list of words with the same length
  Emit(key, size_of(values));}

Map Reduce Problems Discussion

- **Problem 2:** Indexing & Page Rank
- **Statement:** Given a set of web pages, each page has a **page rank** associated with it, use Map-Reduce to find, for each word, *a list of pages (sorted by rank)* that contains that word
- **Question:**
  - What are the Mapper and Reducer Functions?



MapReduce

Word 1: [page x1, page x2, ..]

Word 2: [page y1, page y2, …]

…

Map Reduce Problems Discussion

- **Problem 2:** Indexing and Page Rank
- **Mapper and Reducer:**
  - Mapper(page_id, <page_text>)

    { Emit (**word, <page_id, page_rank>**) }
  - Reducer(output of map)

    { Emit (word, **List of pages contains the word sorted by their page_ranks)**}

Mapper and Reducer of Indexing and PageRank

- Map(key, value){
  // key: a page
  // value: words in a page
  for each word w in value:
    Emit(w, <page_id, page_rank>);}
- Reduce(key, list of values){
  // key: a word
  // list of values: a list of pages containing that word
  sorted_pages=sort(values, page_rank)
  Emit(key, sorted_pages);}