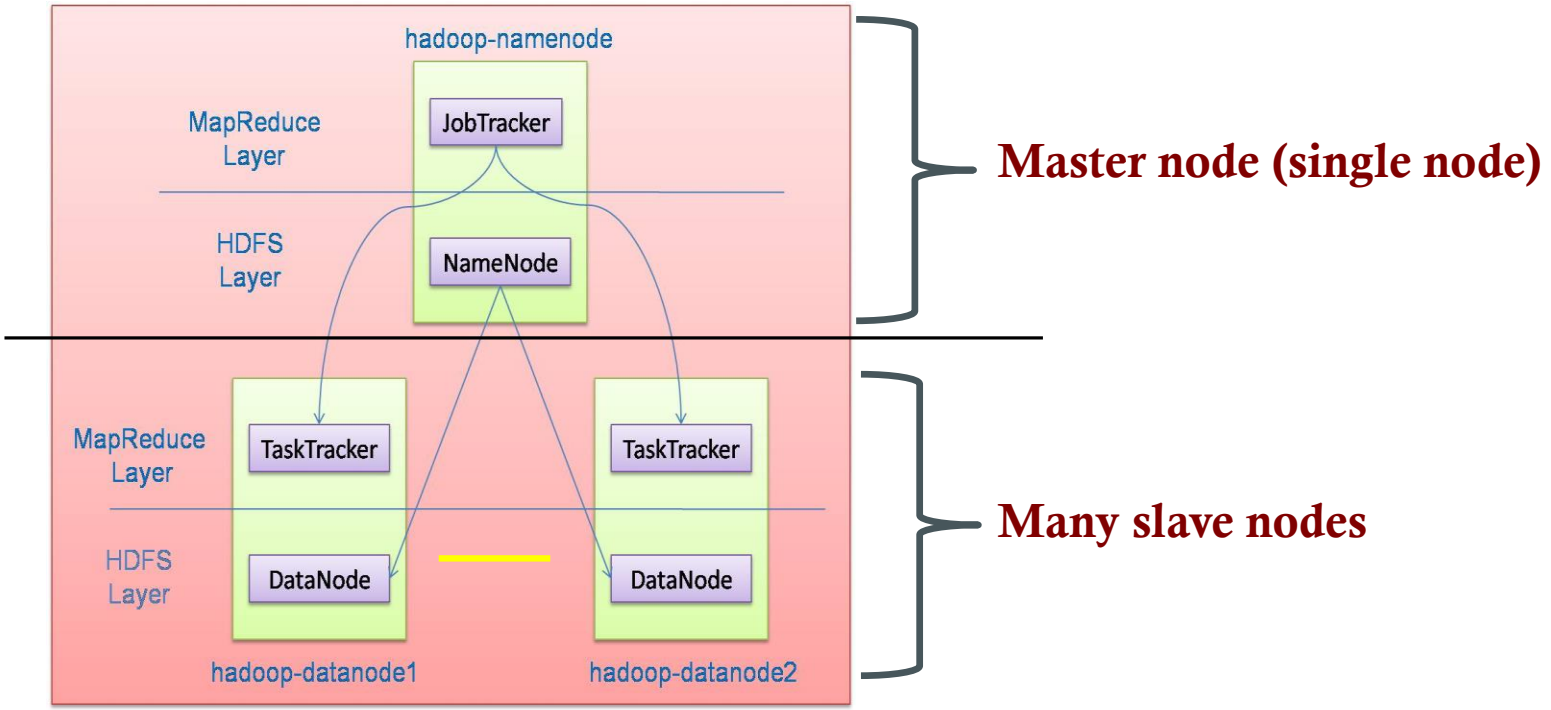
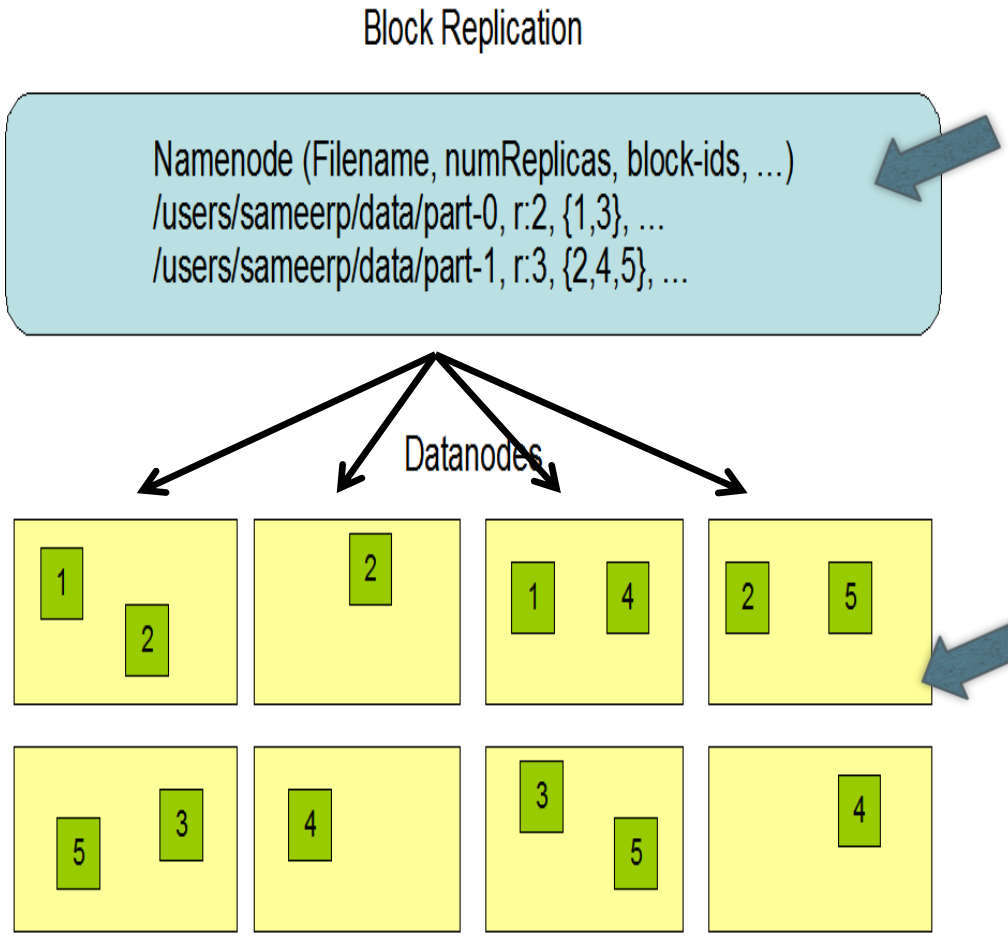
Hadoop/MapReduce: Mappers & Reducers

Hadoop Architecture

- Distributed file system (HDFS)
- Execution engine (MapReduce)



Hadoop Distributed File System (HDFS)



Centralized namenode

- Maintains metadata info about files

File *F* 1 2 3 4 5

Blocks (64 MB)

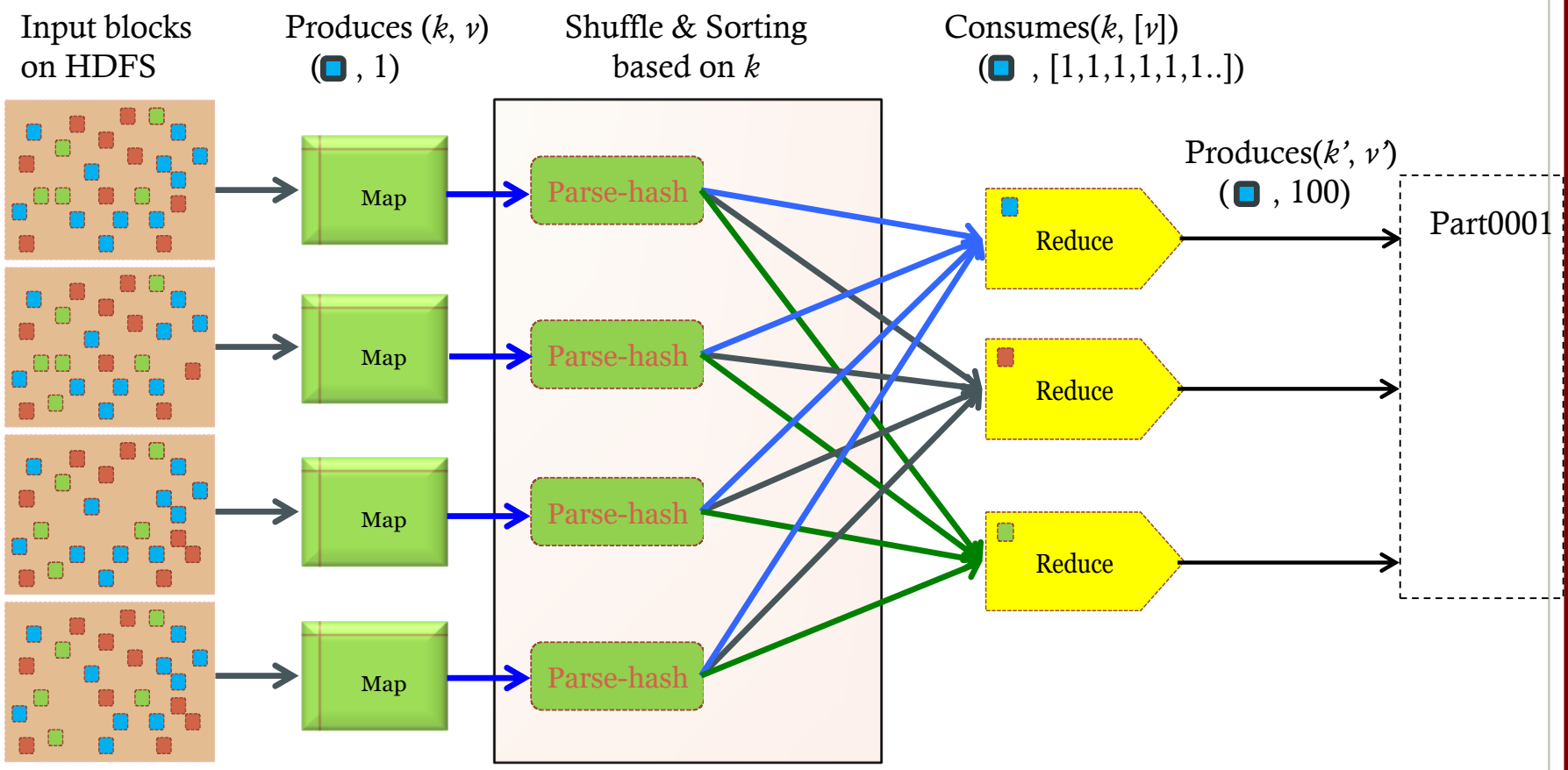
Many datanode (1000s)

- Store the actual data
- Files are divided into blocks
- Each block is replicated *N* times (Default = 3)

Main Properties of HDFS

- ***Large:*** An HDFS instance may consist of thousands of server machines, each storing part of the file system's data
- ***Replication:*** Each data block is replicated many times (default is 3)
- ***Failure:*** Failure is the norm rather than exception
- ***Fault Tolerance:*** Detection of faults and quick automatic recovery from them is a core architectural goal of HDFS:
  - **Name-node is consistently checking Data-nodes**

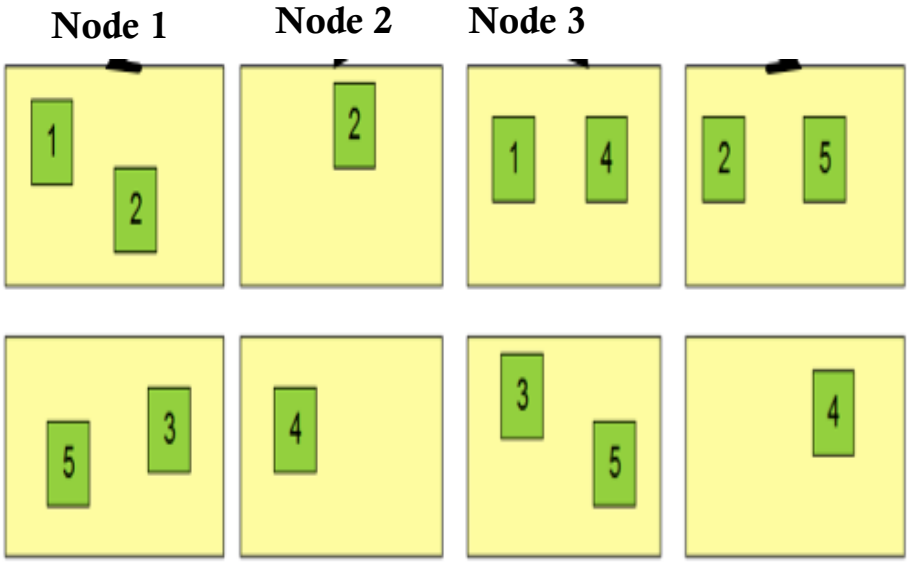
Map-Reduce Execution Engine (Example 1: Color Count)



Users only provide the "Map" and "Reduce" functions

Job Tracker : MapReduce Engine

- **JOB Tracker is the master node (runs with namenode)**
  - Receives the user's "job"
  - Decides on how many tasks will run (eg. number of mappers)
  - Decides on where to run each mapper (concept of locality)



File *F*

- This file has 5 Blocks;
- So how many map tasks ?

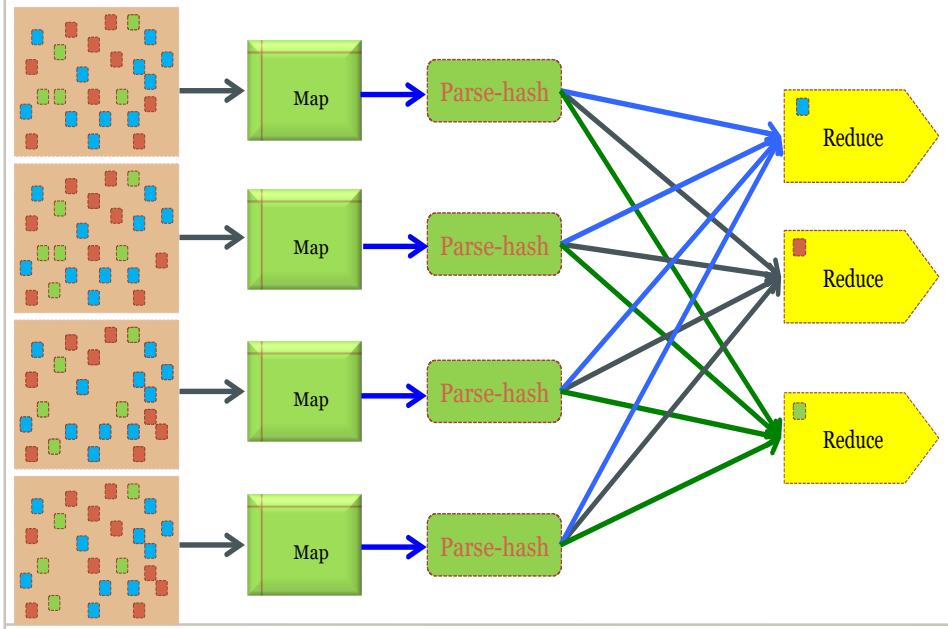
→ *Run 5 map tasks*

- Where to run the task reading "block 1"?

Run it on Node 1 or Node 3

Task Tracker : MapReduce Engine

- **TASK Tracker is the slave node (runs on each datanode)**
  - Receives the task from Job Tracker
  - Runs the task until completion (either map or reduce task)
  - Always in communication with Job Tracker reporting progress



In this example:
1 map-reduce job
consists of 4 map tasks
and 3 reduce tasks

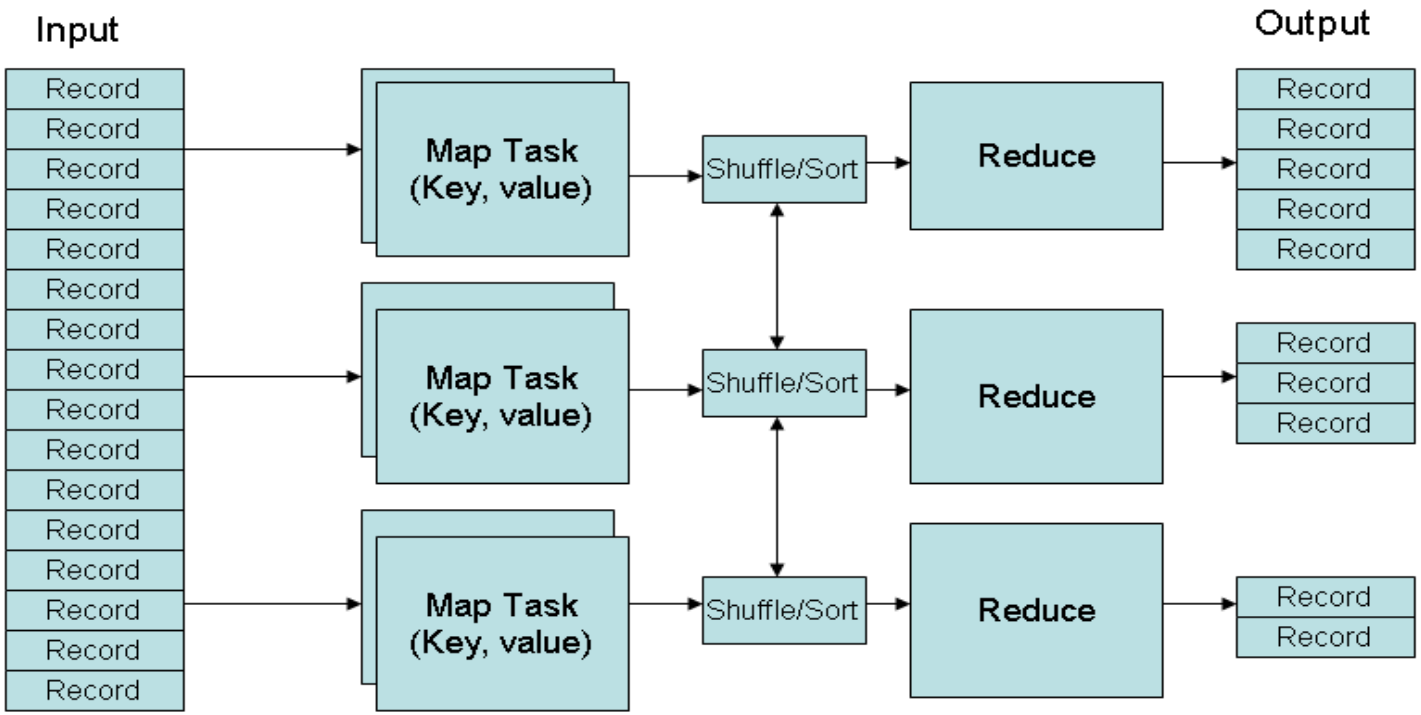
Key-Value Pairs

- Developer provides Mapper and Reducer functions
- Developer decides what is key and what is value
- Developer must follow the "*key-value pair*" interface

Key-Value Pairs

- **Mappers:**
  - Consume <key, value> pairs
  - Produce <key, value> pairs
- **Reducers:**
  - Consume <key, <list of values>>
  - Produce <key, value>
- **Shuffling and Sorting:**
  - Hidden phase between mappers and reducers
  - Groups all similar keys from all mappers, sorts and passes them to a particular reducer in the form of <key, <list of values>>

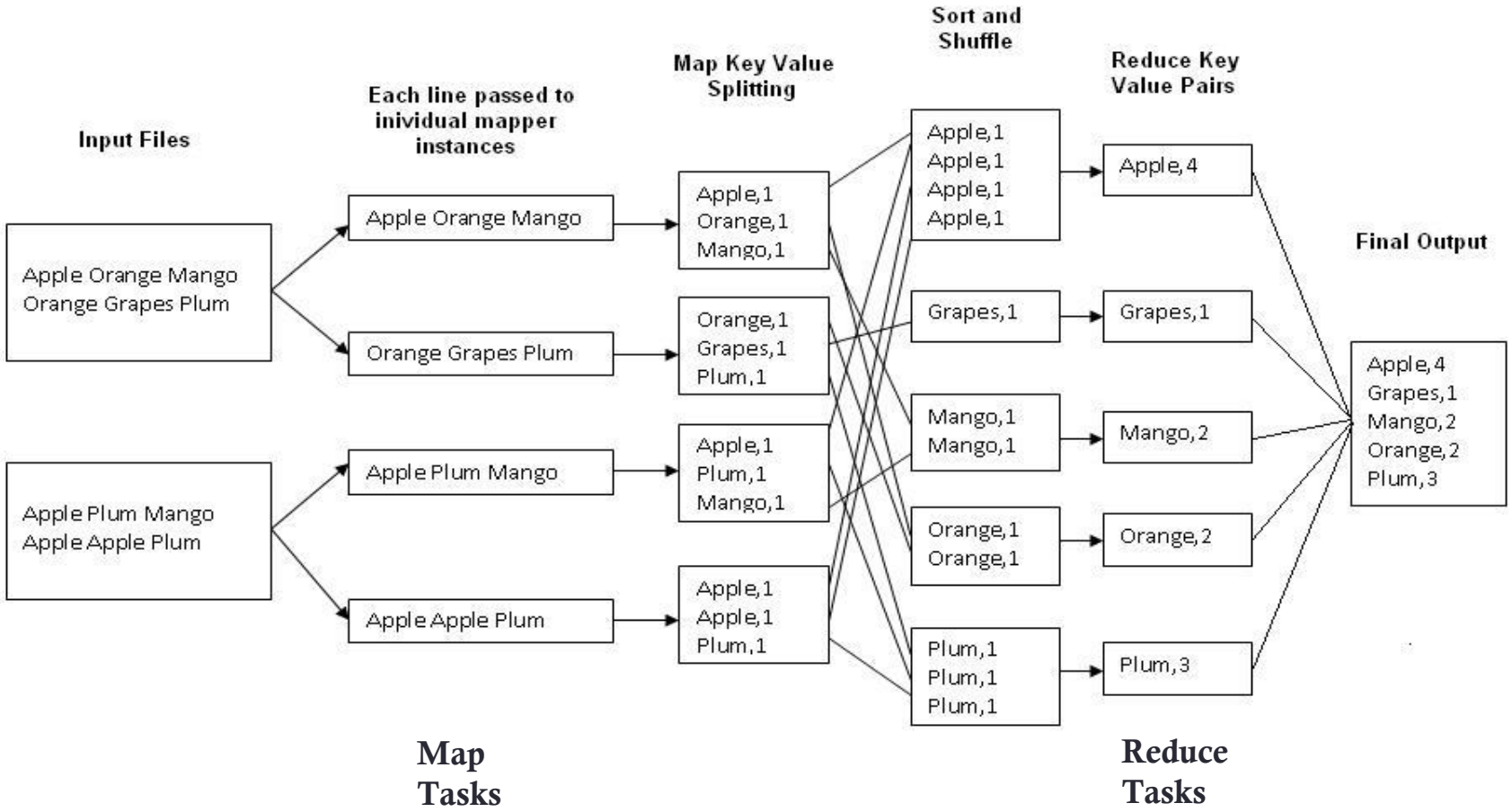
MapReduce Phases



Deciding on what will be key and value ➔ developer's responsibility

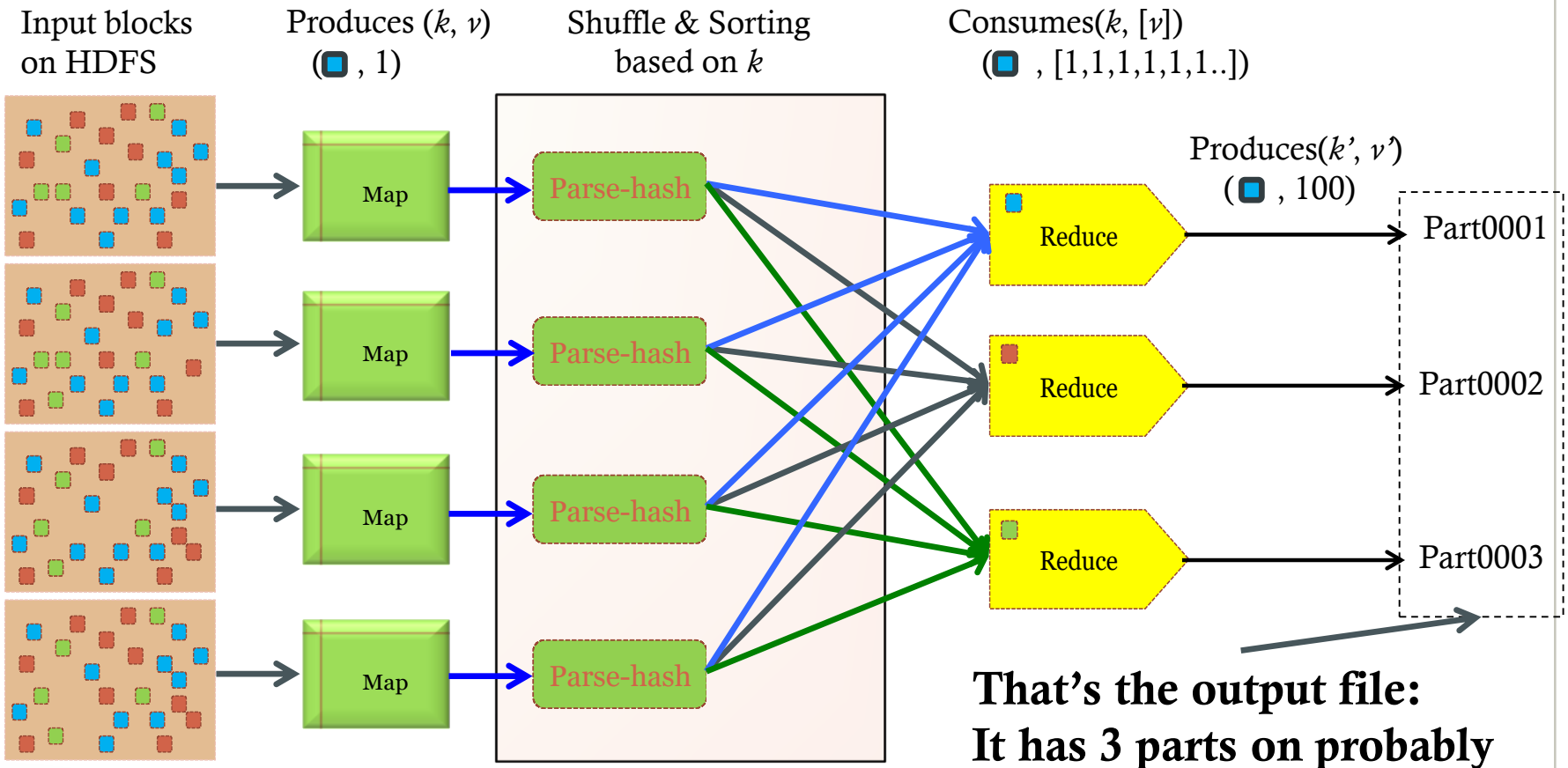
Example 2: Word Count

- **Job: Count occurrences of each word in a data set**



Example 1: Color Count: Output?

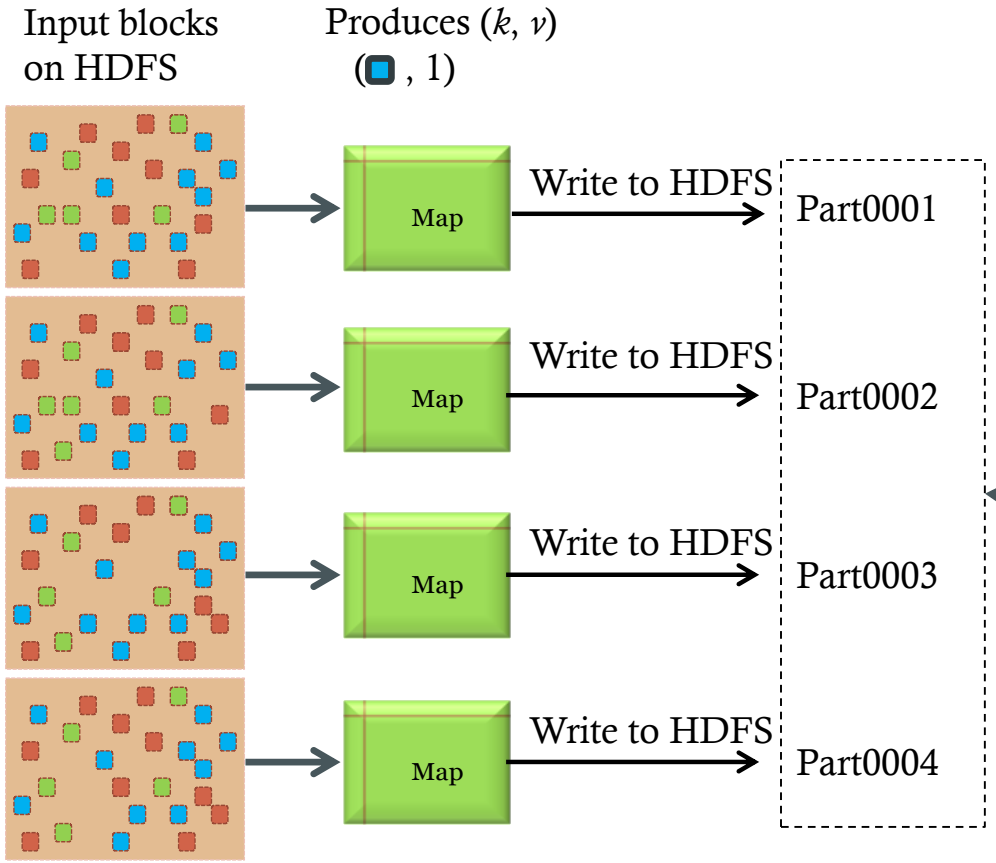
Job: Count the number of each color in a data set



That's the output file:
It has 3 parts on probably 3 different machines

Example 3: Color Filter

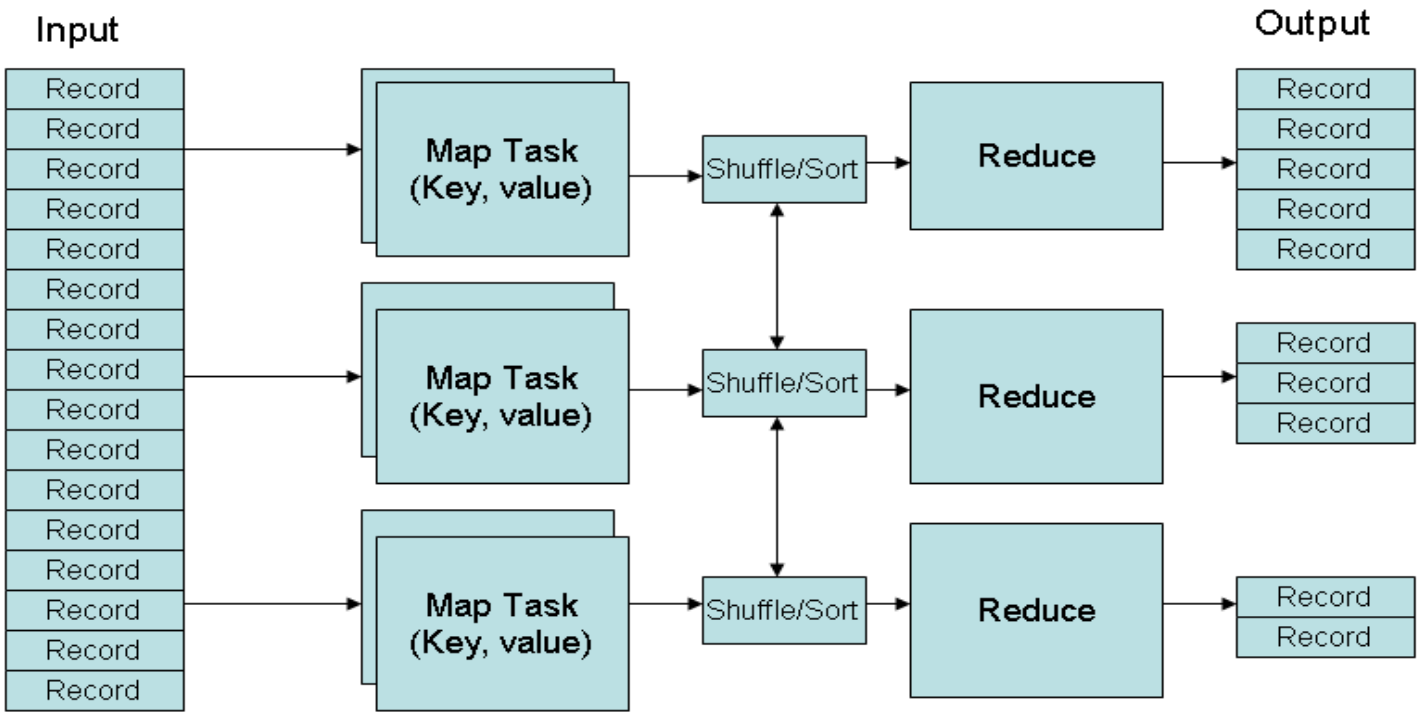
Job: Select only the blue and the green colo



- Each map task will select only the blue or green colors
- No need for reduce phase

That's the output file, it has 4 parts on probably 4 different machines

MapReduce Phases



Deciding on what will be the key and what will be the value ➔ developer's responsibility

Summary of MapReduce Model

- **Just for now ➔ Assume a "split" is a HDFS Block**

Book: Pro Hadoop, Jason Venner

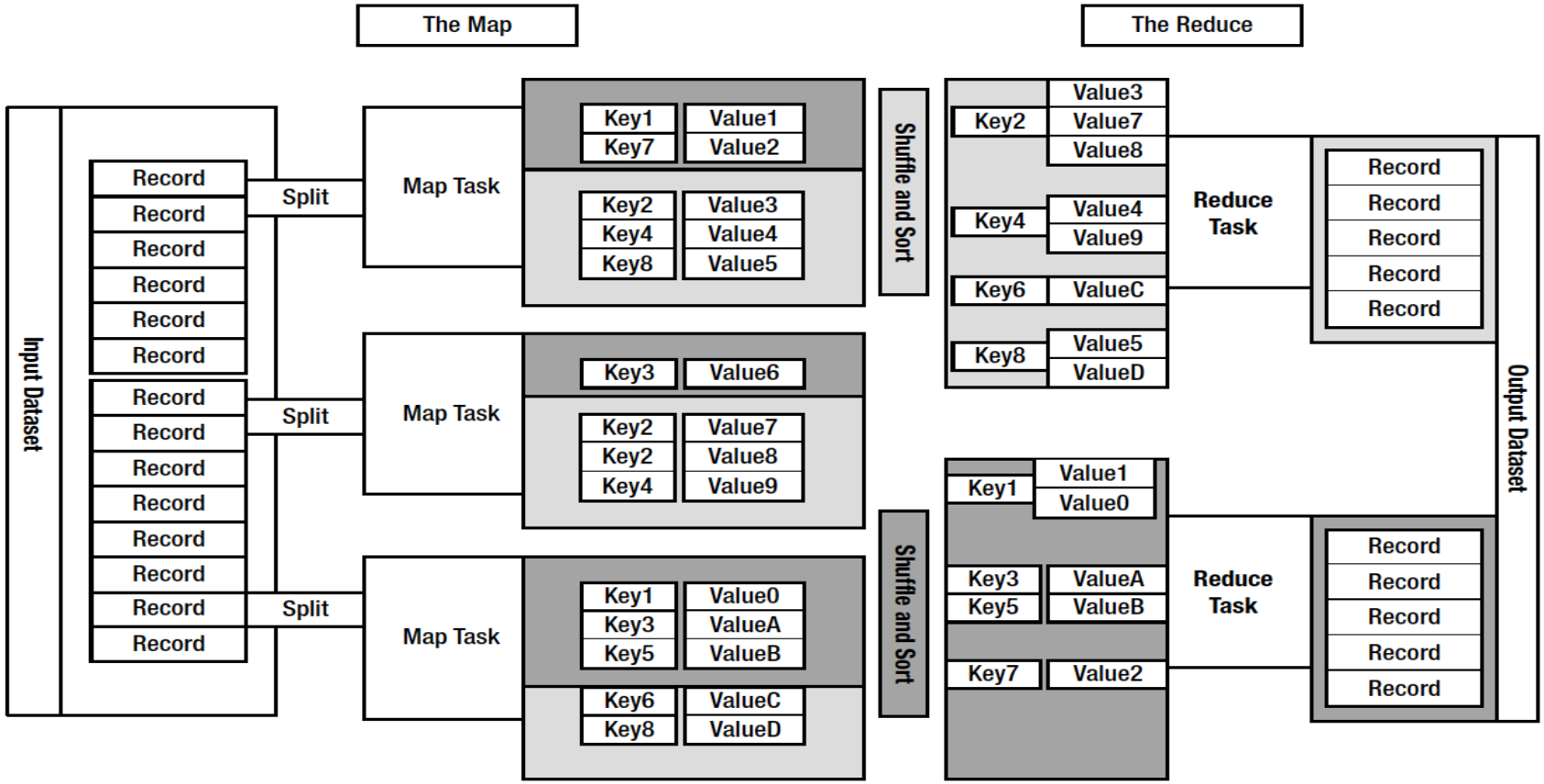


Figure 1-1. *The MapReduce model*

Map Task: Behind the scene

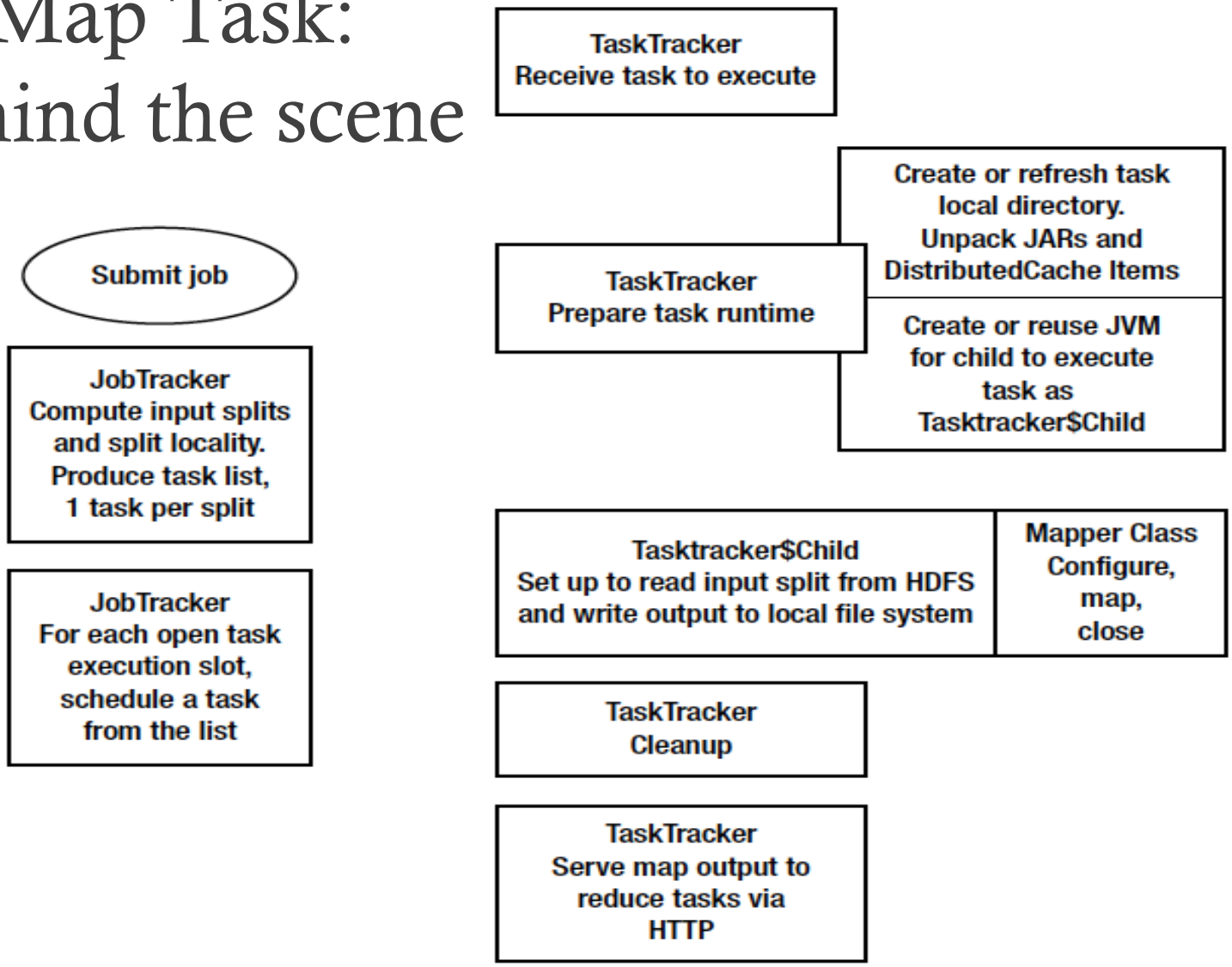


Figure 6-3. *Behind the scenes in a map task*

Book: Pro Hadoop, Jason Venner

Reduce Task: Behind the scene

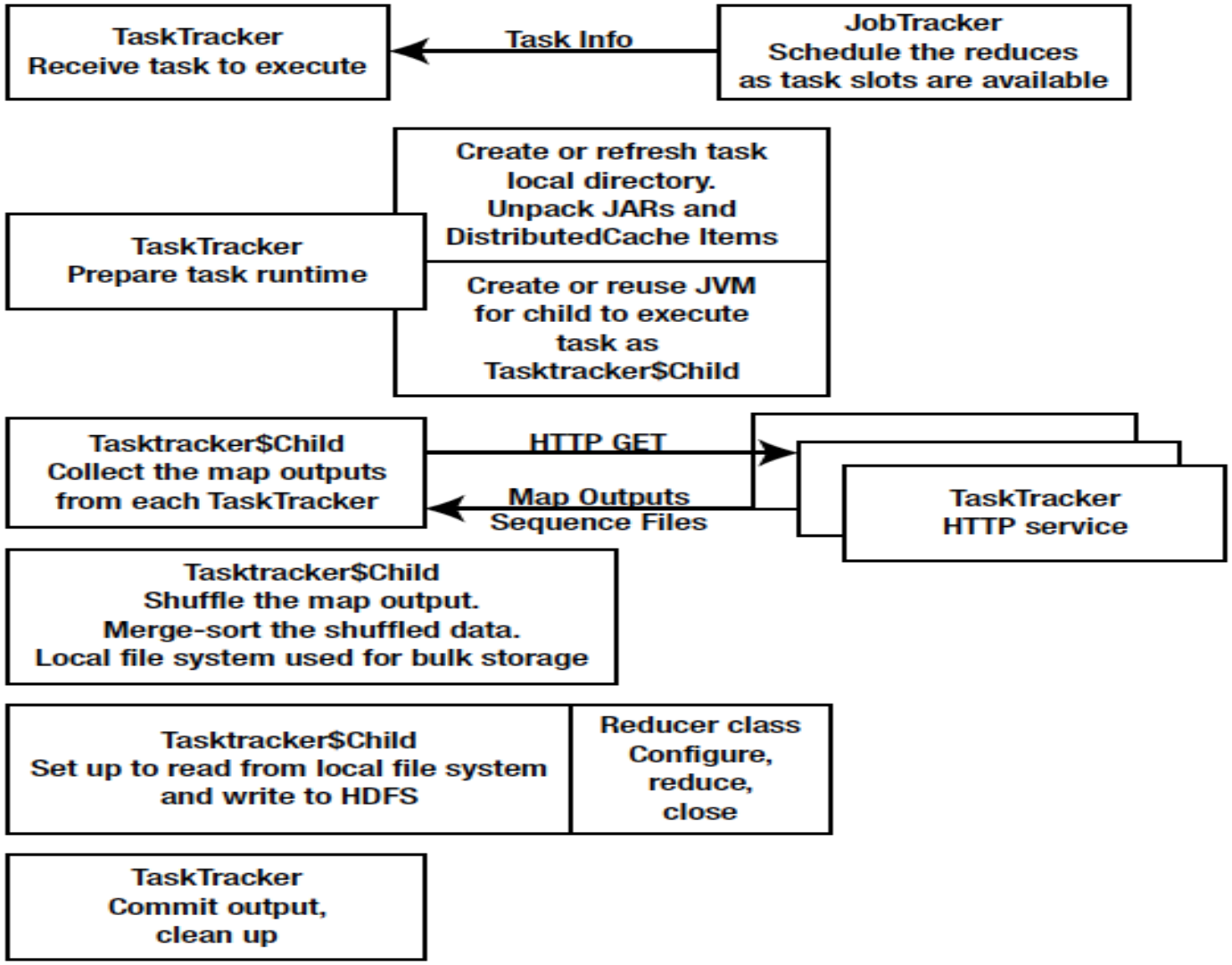


Figure 6-5. *Behind the scenes in the reduce task*

Book: Pro Hadoop, Jason Venner

Processing Granularity

- **Mappers**
  - Run on a record-by-record basis
  - Your code processes that record and may produce:
    - *Zero, one, or many outputs*
- **Reducers**
  - Run on a group-of-records bases (having same key)
  - Your code processes that group and may produce:
    - *Zero, one, or many outputs*