Deep Dive into MapReduce

Agenda

- ✓ Why MapReduce
- ✓ Introduction to MapReduce
- ✓ MapReduce Concepts
- ✓ MapReduce Terminologies
- ✓ Daemons Insights
- ✓ MapReduce Flow
- ✓ Word Count Example

Before MapReduce

- Large scale data processing was very difficult:
  - Manage hundreds or thousands of processors
  - Manage parallelization and distribution
  - Movement of petabytes of data
  - I/O Scheduling
  - Status and monitoring
  - Fault tolerance

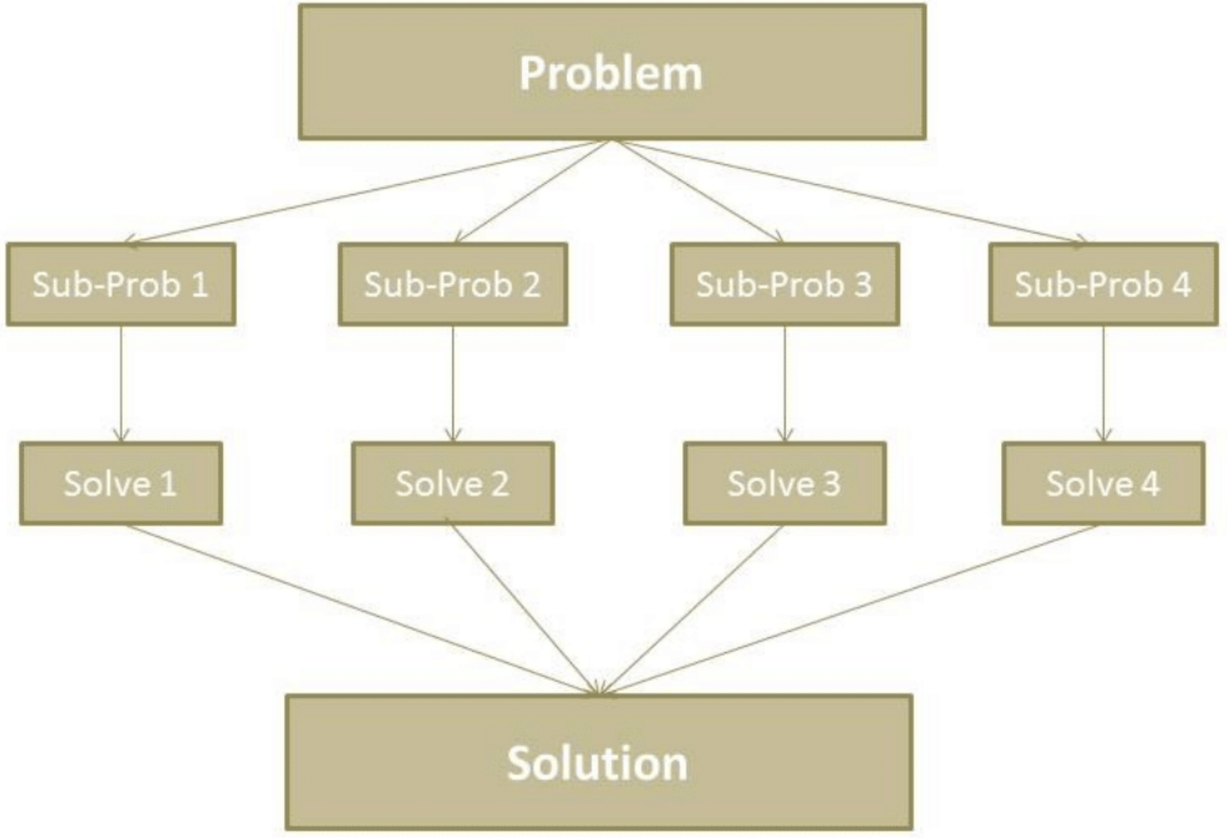
- MapReduce provides all these Features

Introduction to MapReduce

- Model for processing large amounts of data in parallel
  - On commodity hardware
  - Divides the work into a set of independent tasks
  - Lots of nodes work in parallel
- Programming model invented by Google
- Derived from functional programming
  - Map and reduce functions
  - Processes lists of data
- Can be implemented in multiple languages
  - Java, C, C++, Ruby, Groovy, Perl, Python etc.

MapReduce – High level understanding

- Map-Reduce divides the work into small parts, each of which can be done in parallel on cluster of servers.
- Map-Reduce is highly scalable and can be used across many computers.
- Many small machines can be used to process jobs that normally could not be processed by a large machine.



MapReduce Concepts

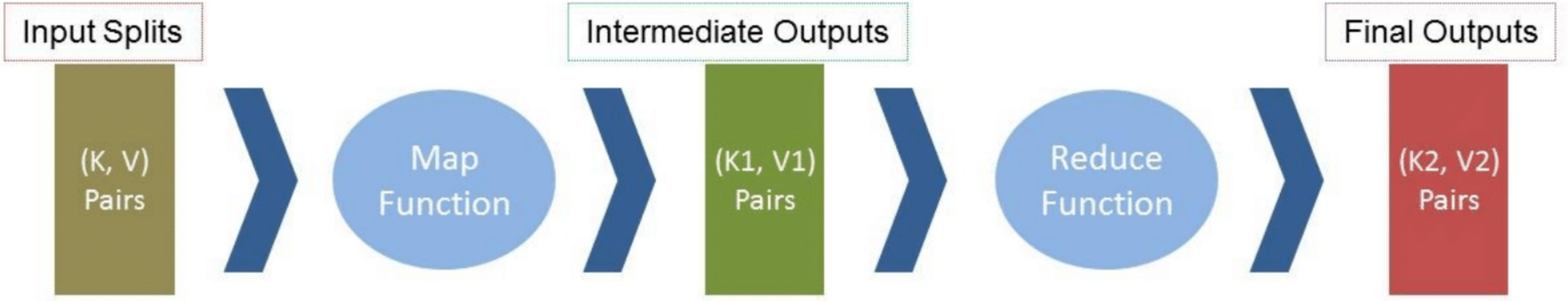
- Conceptually, Map-Reduce programs transform lists of input data elements into lists of output data elements. A Map-Reduce program will do this twice, using two different list processing idioms
  - Map
  - Reduce
- The programmer in MapReduce has to specify two functions, the map function and the reduce function that implement the Mapper and the Reducer in a MapReduce program

MapReduce Terminologies

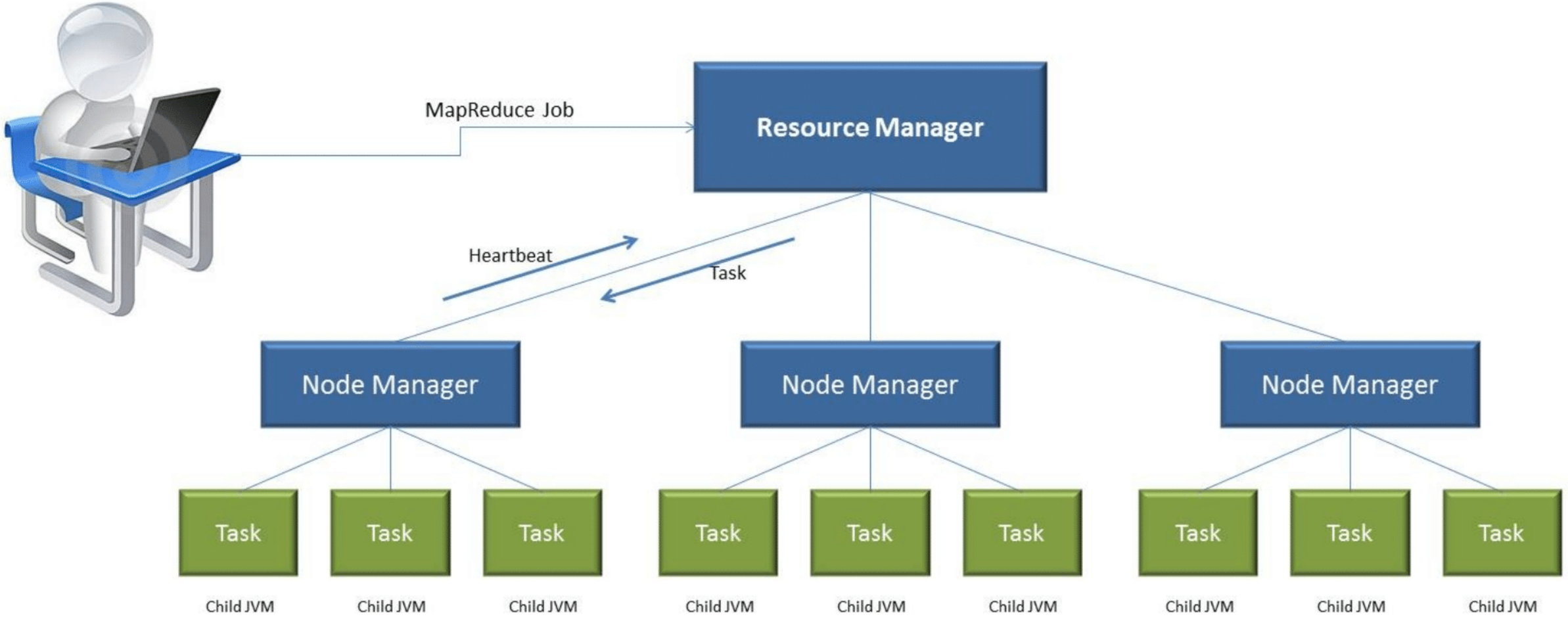
- **Job**
  - A Complete program an execution of a Mapper and Reducer across a data set
  - A Map-Reduce job is the work that the client wants to be performed it consist of the input data, Map-Reduce program and configuration info
  - Client creates a job, configures it and submits it to Master
- **Task**
  - An execution of a Mapper or a Reducer on a slice of data
  - Master divides the work into task and schedules it on the slaves

Keys and Values

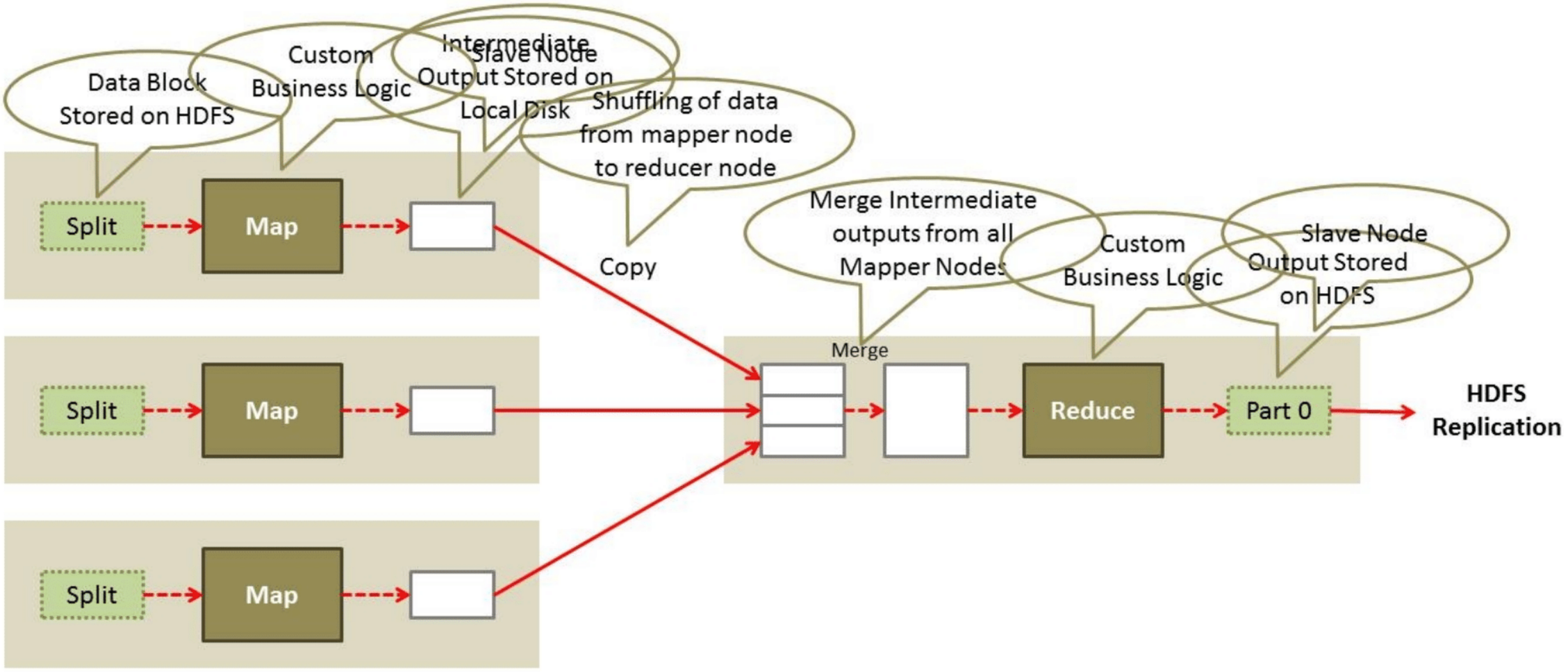
- In MapReduce data elements are always structured as key-value (i.e., (K, V)) pairs
- The map and reduce functions receive and *emit* (K, V) pairs



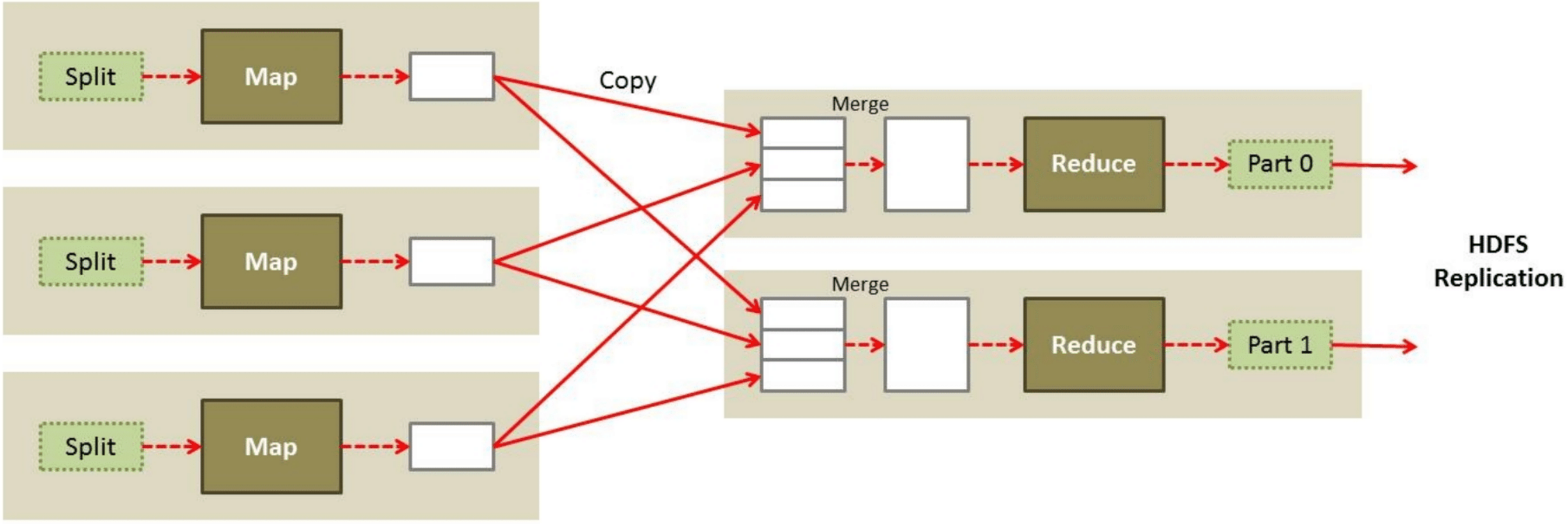
Daemons Insights



MapReduce Flow

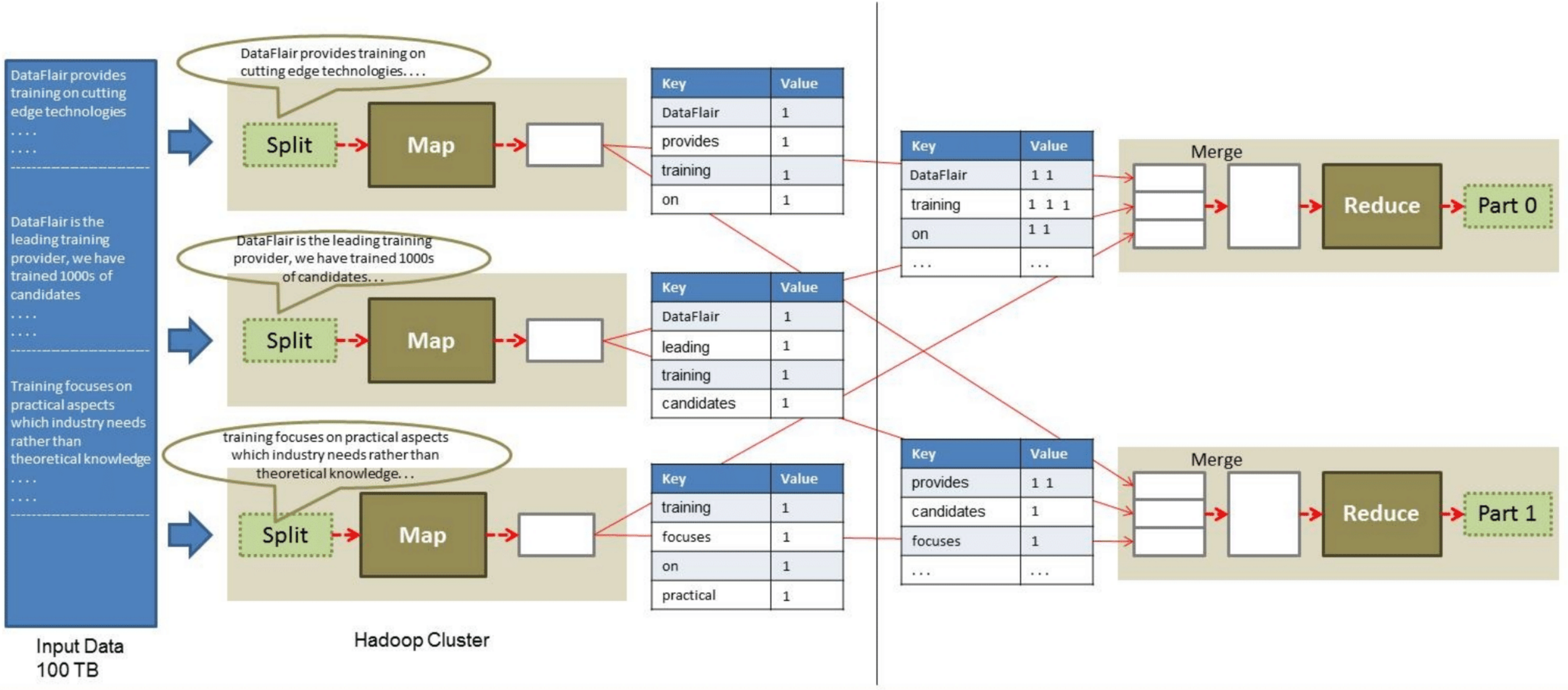


MapReduce Flow [Cntd..]

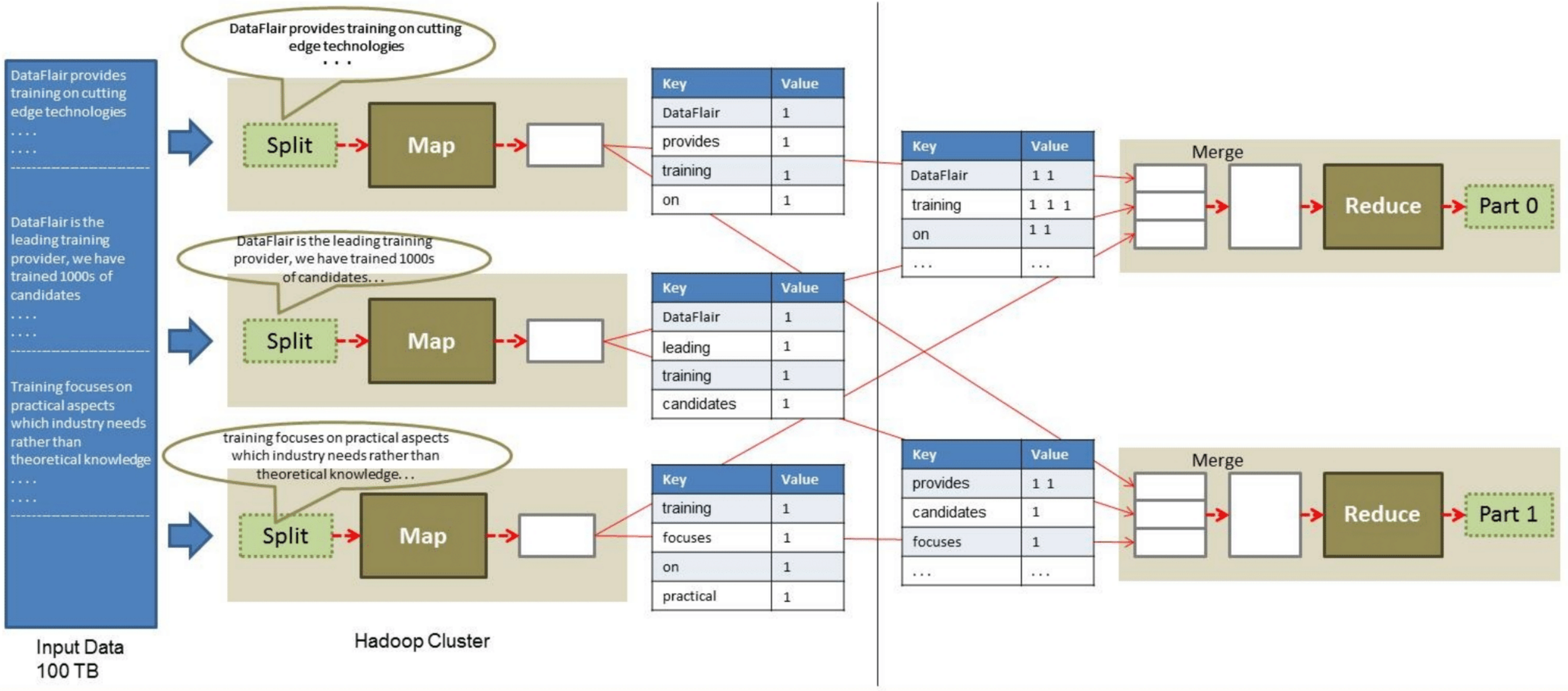


Word Count Example

MapReduce Flow



MapReduce Flow



Map Abstraction

- Takes key/value pair as input
  - Key is a reference to the input value
  - Value is the data set on which to operate
- Processing
  - Function defined by user
  - Applies to every value in value input
- Produces a new list of key/value pairs
  - Output of Map is called intermediate output
  - Can be different type from input pair
  - Output is stored in local disk

Mapper

- Mapper is a program that performs Map phase of MapReduce Job.
- Takes data in the form of Key value pairs.
- Outputs zero or more key-value pairs
- Map() function is the workhorse that does the job. User writes his **custom code** here
- Map() function receives one key-value pair at a time
- Assuming an inputsplit has 4 key-value pairs, the Mapper will have map() function called 4 times.
- In the most frequent cases, the mapper operates on the data that is present locally.

How many Mappers?

- The number of maps is driven by the total size of the inputs
- Hadoop has found the right level of parallelism for maps is between 10-100 maps/node
- If you expect 10TB of input data and have a block size of 128MB, you will have 82,000 maps
- Number of tasks are controlled by number of splits returned and can be user overridden

Reduce Abstraction

- Takes intermediate Key / Value pairs as input
  - Generated by Map
  - Key / Value pairs are sorted by key
- Processing
  - Function defined by user
  - Iterator supplies the values for a given key to the Reduce function.
- Produces final list of key/value pairs
  - Output of Reduce is called Final output
  - Can be different type from input pair
  - Output is stored in HDFS

Reducer

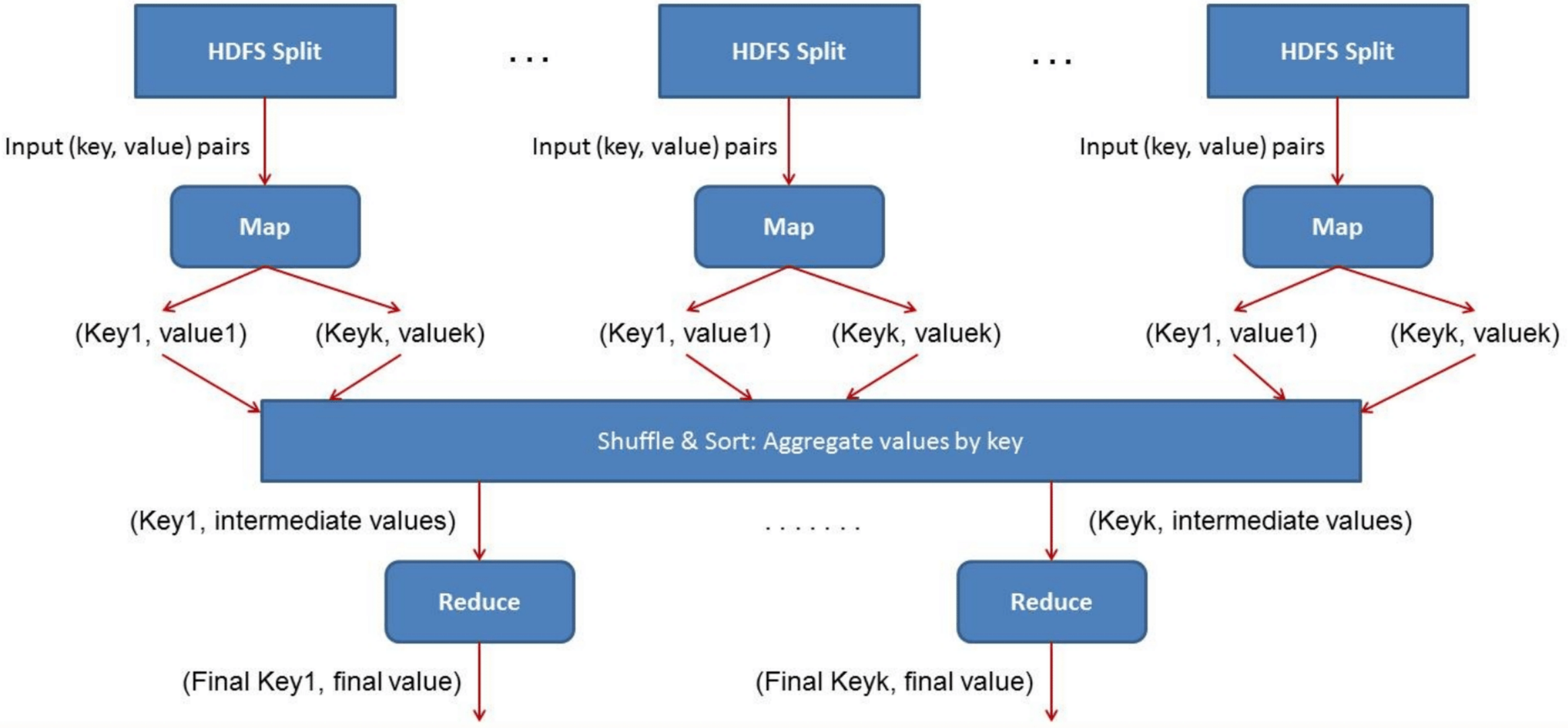
- Reducer is a program that performs reduce phase of MapReduce Job.
- Takes data in the form of key-value pairs.
- Output zero or more key-value pairs.
- Reduce() function is the workhorse that does the job. User writes his **custom code** here
- Reduce() function receives one key and all values pertaining to that key.
- Reducer normally has nothing to do with data locality i.e., data to place where reducer is going to be executed.

How many Reducers ?

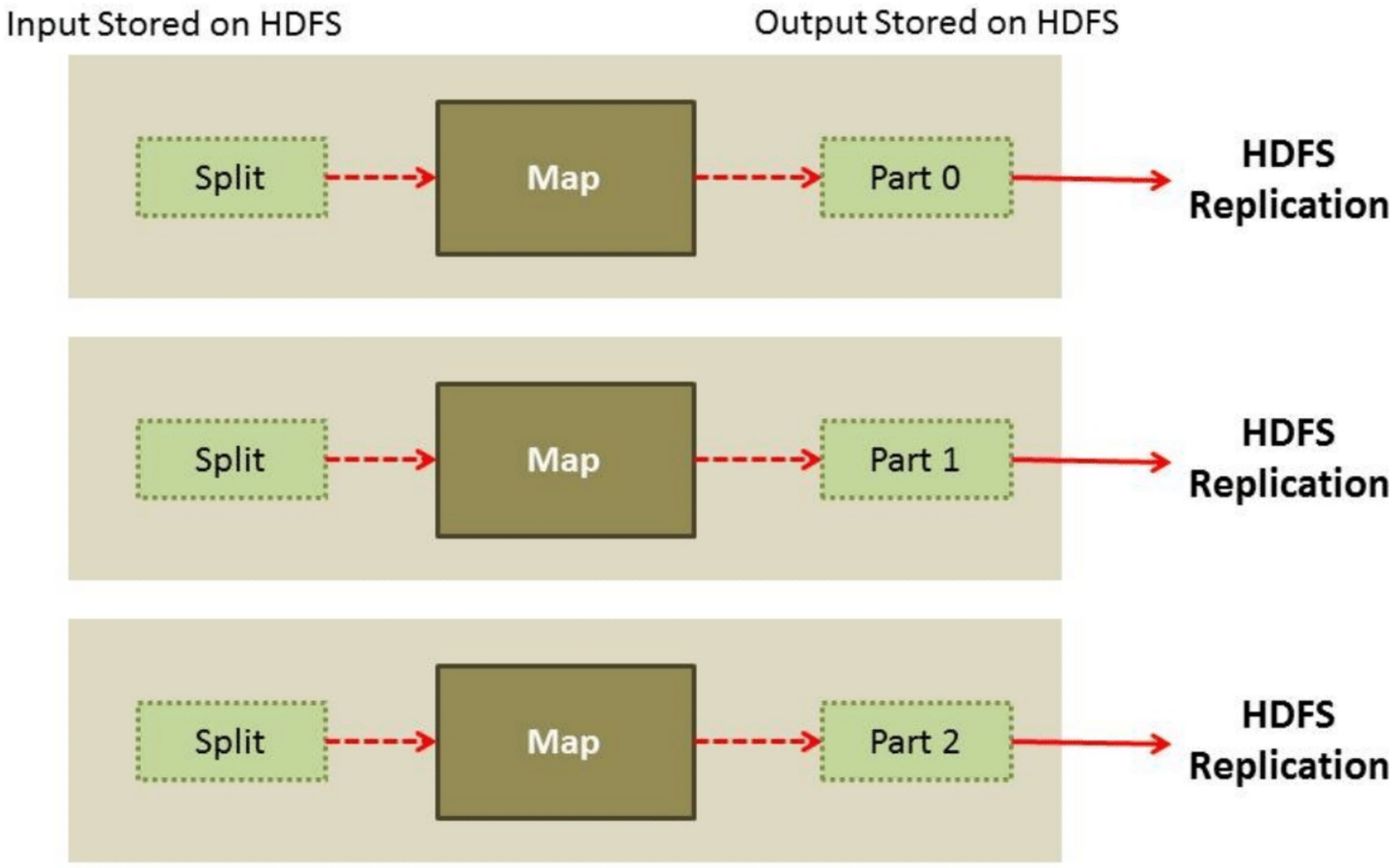
- The right number of reduces is 0.95 or 1.75 multiplied by (mapred.tasktracker.reduce.tasks.maximum).
- With 0.95 all of the reduces can launch immediately and start transferring map outputs as the maps finish.
- With 1.75 the faster nodes will finish their first round of reduces and launch a second wave of reduces doing a much better job of load balancing.

How Map and Reduce Work Together

Map-Reduce Detailed View



Map Only Job



Map Only Job

- There is no reducer in Map only Job
- No Need of Shuffling (no need of **data movement**)
- Output of Mapper is written directly on HDFS
- Map only jobs are **more efficient** than MapReduce Job
- Certain types of problems can be solved with Map Only Job like data parsing (conversion of data from one format to another)

Combiner

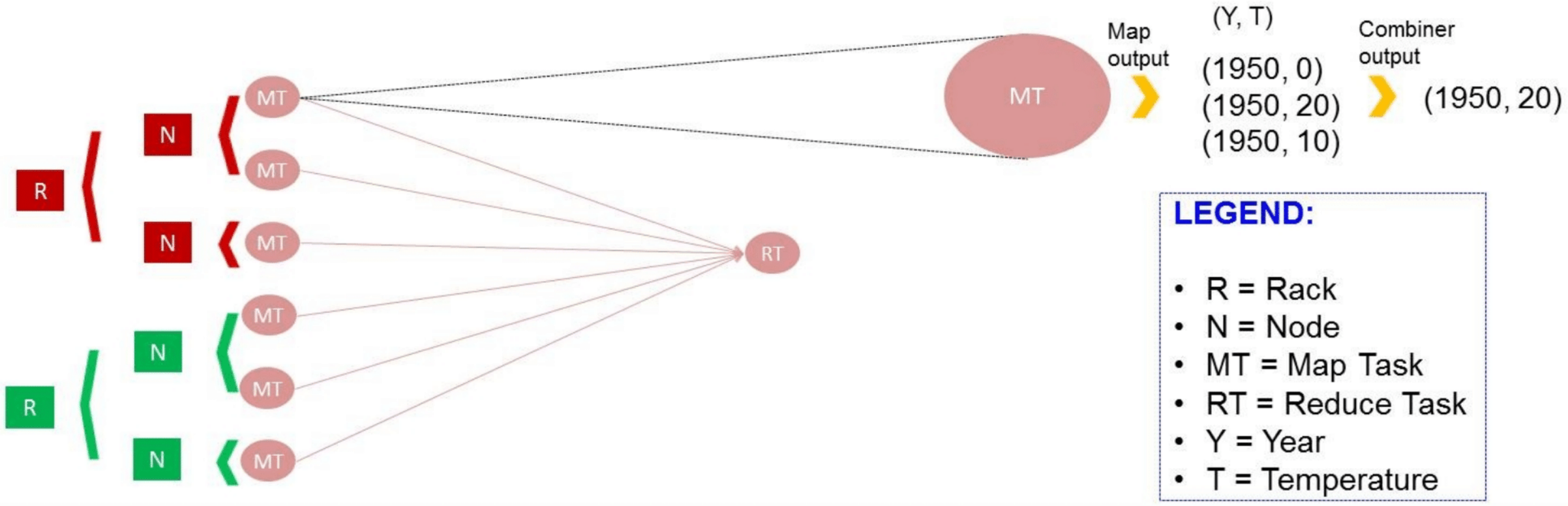
- MapReduce applications are limited by the bandwidth available on the cluster
- It pays to minimize the data shuffled between map and reduce tasks
- Hadoop allows the user to specify a combiner function (just like the reduce function) to be run on a map output

Combiner

- Mini-Reducer process which operates only on data generated by one machine.
- The combiner function runs on the output of the map phase and is used as a filtering or an aggregating step to lessen the number of intermediate keys that are being passed to the reducer.
- In most of the cases the reducer class is set to be the combiner class.
- The combiner is used as an optimization for the MapReduce job.

Combiner

- Mapper generates (Year, Temperature) as (key, value). The objective of MapReduce job is to get the highest temperature each year

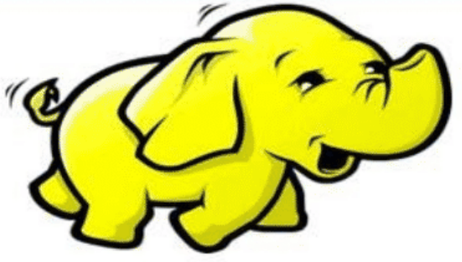


Data Locality

- **Move computation** close to the data, rather then data to computation
- This **minimizes network congestion** and increases the overall throughput of the system.
- The assumption is that it is often better to migrate the computation closer to where the data is located rather than moving the data to where the application is running. HDFS provides interfaces for applications to move themselves closer to where the data is located.

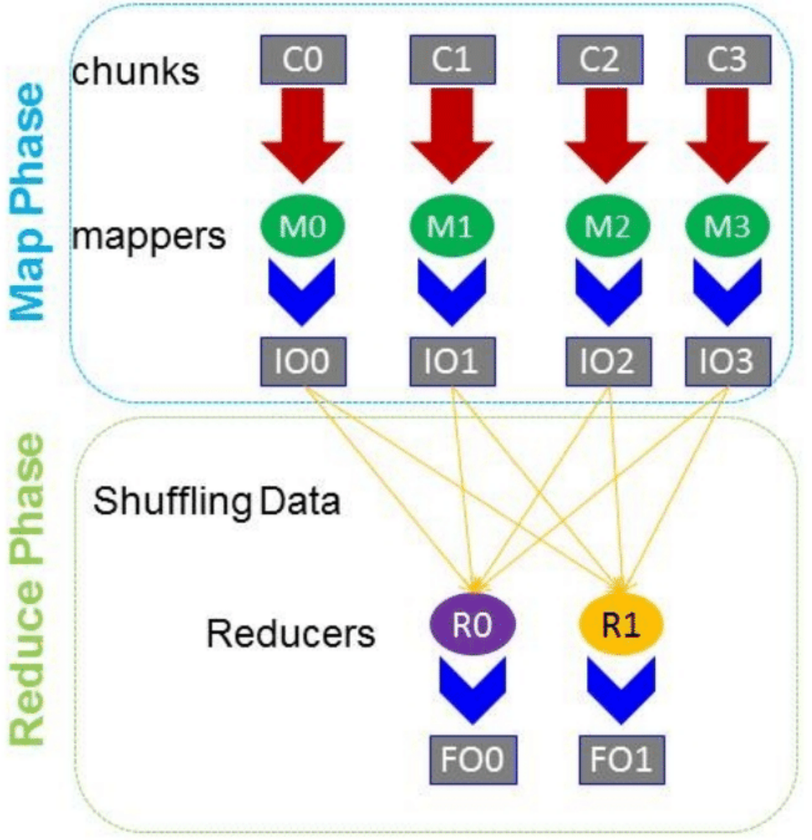
Comparison: RDBMS vs Hadoop

	Traditional RDBMS	Hadoop / MapReduce
Data Size	Gigabytes	Petabytes (Hexabytes)
Access	Interactive and Batch	Batch – NOT Interactive
Updates	Read / Write many times	Write once, Read many times
Structure	Static Schema	Dynamic Schema
Integrity	High (ACID)	Low
Scaling	Nonlinear	Linear
Query Response Time	Can be near immediate	Has latency (due to batch processing)



Anatomy of MapReduce

- In MapReduce, chunks are processed in isolation by tasks called *Mappers*
- The outputs from the mappers are denoted as *intermediate outputs* (IOs)
- The process of bringing together IOs into a set of Reducers is known as *Shuffling and Sort process*
- Intermediate outputs will be given as input into second set of tasks called *Reducers*
- The Reducers produce the *final outputs* (FOs)
- Overall, *MapReduce* breaks the data flow into two phases, map phase and reduce phase



Thank You

DataFlair Web Services Pvt Ltd

+91-8451097879 / +91-7718877477

info@data-flair.com

http://data-flair.com

https://www.facebook.com/DataFlairWS