Parallel Data Processing with MapReduce

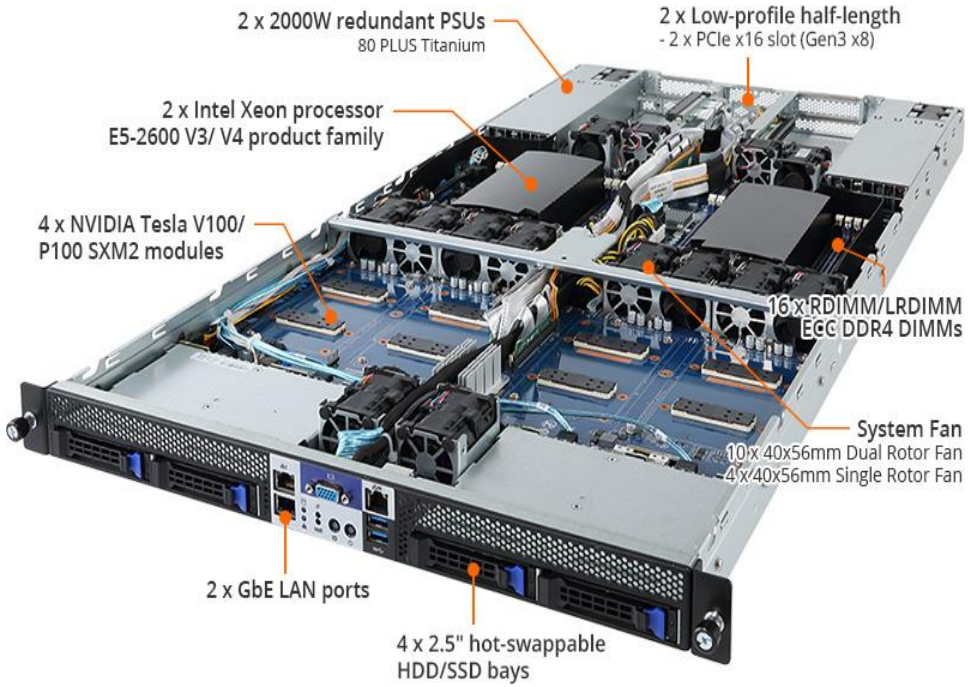
CS140 Tao Yang

Overview

- **What is MapReduce?**
- **Parallel data processing with MapReduce**
  - Example with word frequency count of documents
  - Hadoop distributed file system
  - Execution of MapReduce programs

Motivations

- **Target:** Parallel processing on clusters with or without disk I/O
- **Functionalities**
  - Automatic parallelization & distribution with fault-tolerance
  - A clean abstraction for programmers
  - Suitable for large batch data processing jobs
    - » Not targeted for interactive client-server applications



GIGABYTE's 1U 4 GPU Server

Compute-intensive vs Data-intensive Applications

	Compute-intensive	Data-intensive
Where does program spend time?	CPU-intensive	I/O-intensive
Memory vs disk usage	Data fits in memory	Data cannot fit in memory
Application examples	• Google PageRank • Equation solving • Model training • Web request processing	Analyze user behavior from web access log files of hundreds of terabytes

Parallel Iterative Algorithm for Google PageRank

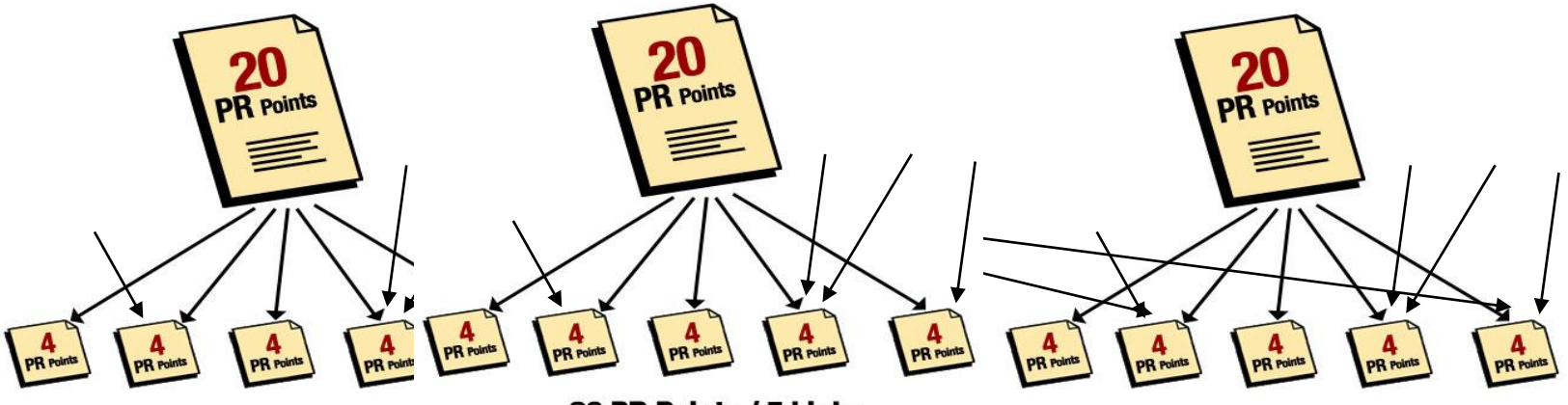
Parents

$$PR(x) = (1-d) + d\sum_{i=1}^{n} \frac{PR(t_i)}{C(t_i)}$$

- PR(x) is the page rank of each page.
- C(t) is out-degree of parent node t.
- d is a damping factor. $0 \le d \le 1$

Let each process (or thread) be responsible for a subset of graph vertices (web pages). Repeat the following map-reduce phases:

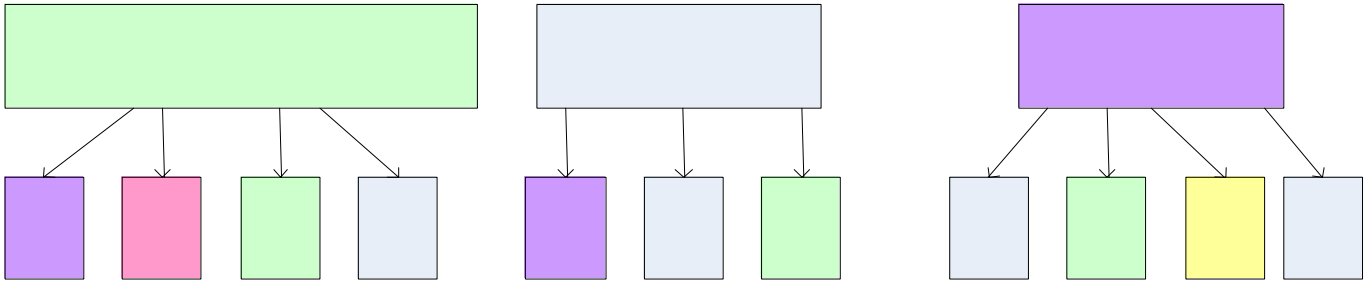
1. **Map:** Every process sends credits of web pages to their outgoing neighbors (children)
2. **Reduce:** Every process receives credits from the parents of its assigned web pages, and updates their page rank value



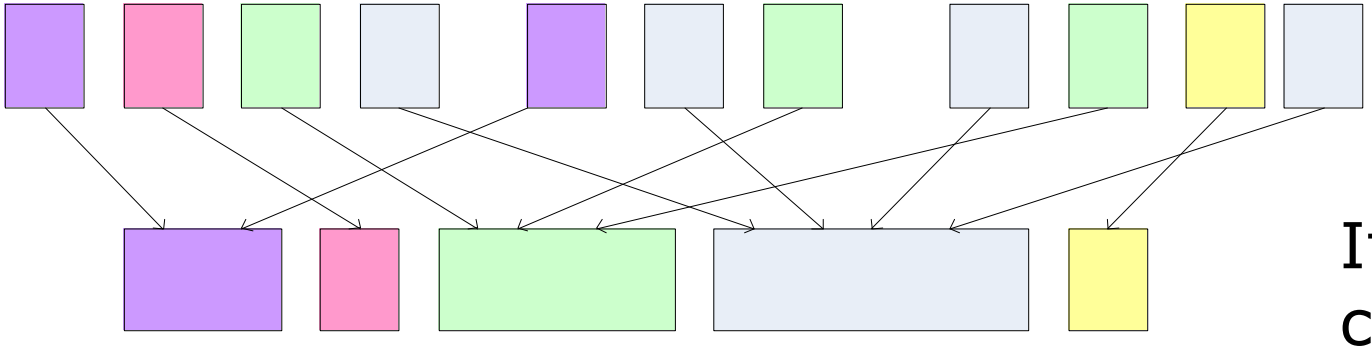
Parallel Algorithm for PageRank

Let each color represent an assigned thread (or process)

Map Phase: distribute PageRank "credit" to outgoing neighbors



Reduce Phase: gather up PageRank "credit" from multiple sources to compute new PageRank value



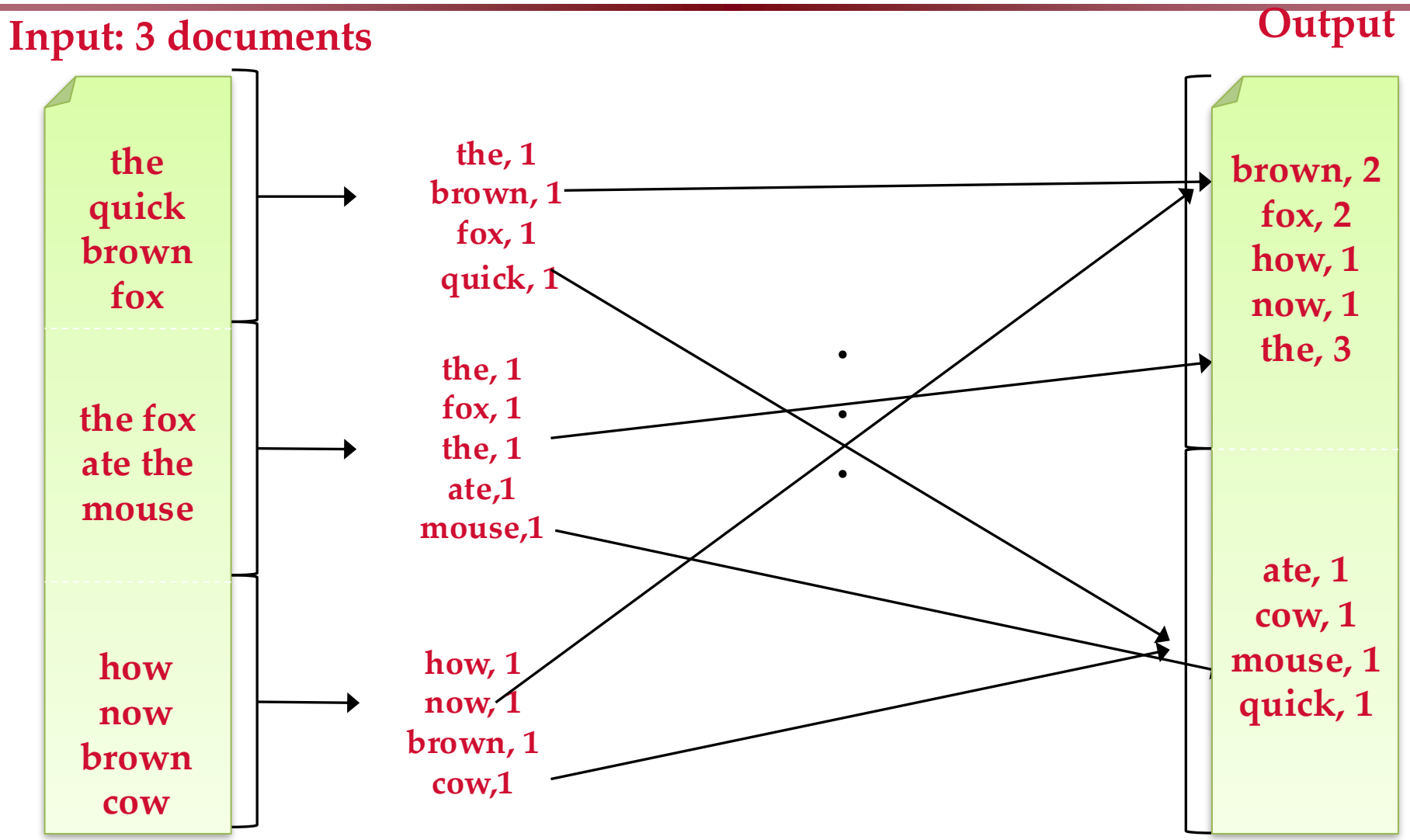
Iterate until convergence

Source of Image: Lin 2008

MapReduce Programming Model

- **Data: a set of key-value pairs**
  - Initially input data is stored in files
- **Parallel computation to manipulate data:**
  - A set of Map tasks and reduce tasks to access and produce key-value pairs
  - Map Function: (key1, val1) → (key2, val2)
    - » Apply it to many key-value pair records
  - Reduce: (key2, [val2 list]) → [val3]
    - » Apply it to many key-value pairs with the same key
- **Input/output files are stored in distributed file system built on a cluster of machines→ Looks like one machine**

Example of Word Count Problem: Count word frequency in billions of documents



Key-Value Pairs Manipulated by Map/Reduce Tasks

- ***Map* function**
  - Apply a function to a set of values
  - Example: Apply function length() to a list:

    (() (a) (ab) (abc)))

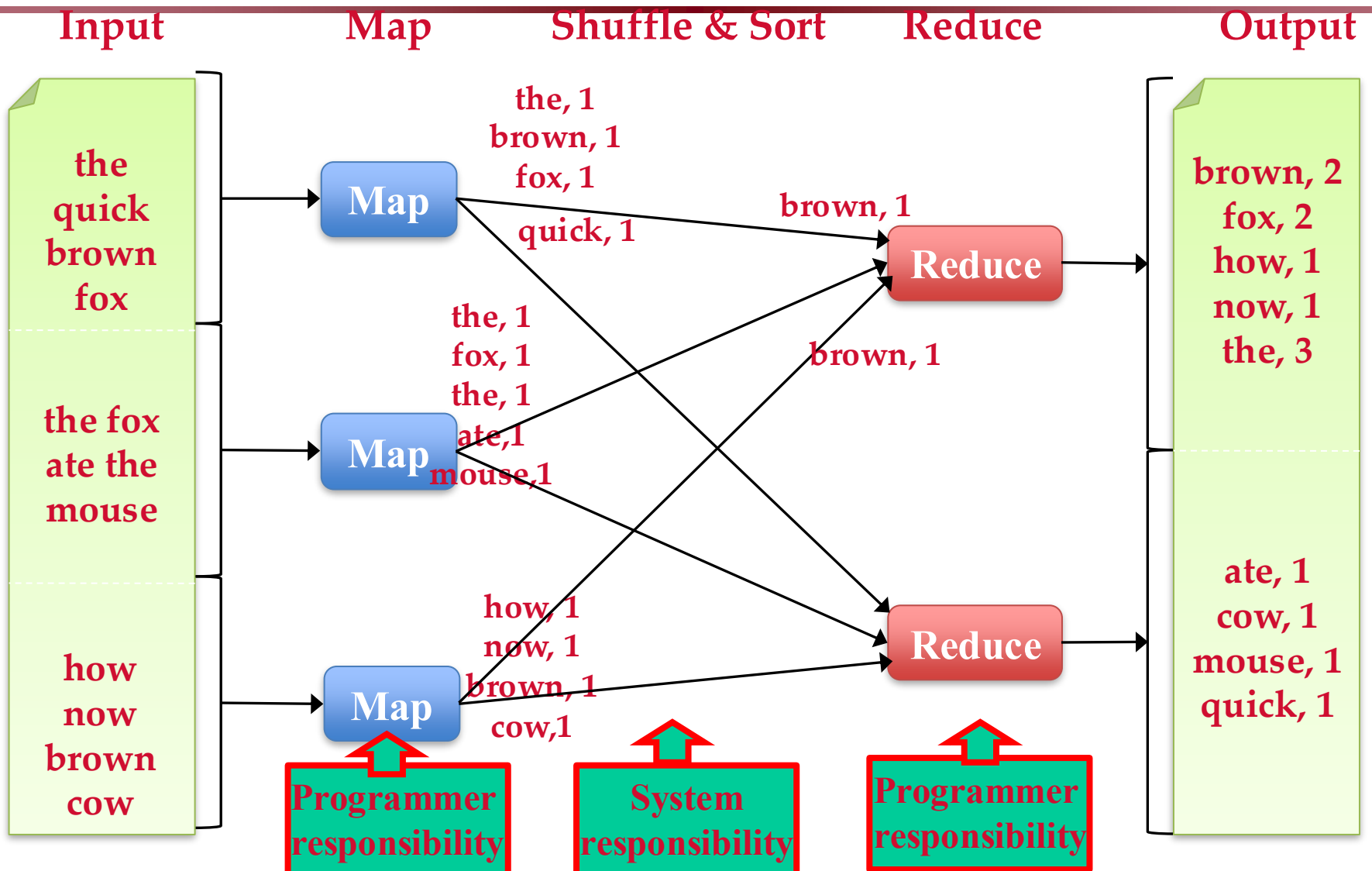
    →(length(()) length(a) length(ab) length(abc))

    → (0 1 2 3)

- *Reduce* function
  - It combines all the values together using a function.
  - Example: apply function add() to a list (0 1 2 3)

    → 6

Example of Word Count with Map/Reduce Processing



Pseudo-code for Word Count

```
map(String input_key, String input_value):
// input_key: document name
// input_value: document contents
    for each word w in input_value:
        EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
// output_key: a word
// output_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result  = result + ParseInt(v);
    Emit(output_key, AsString(result));
```

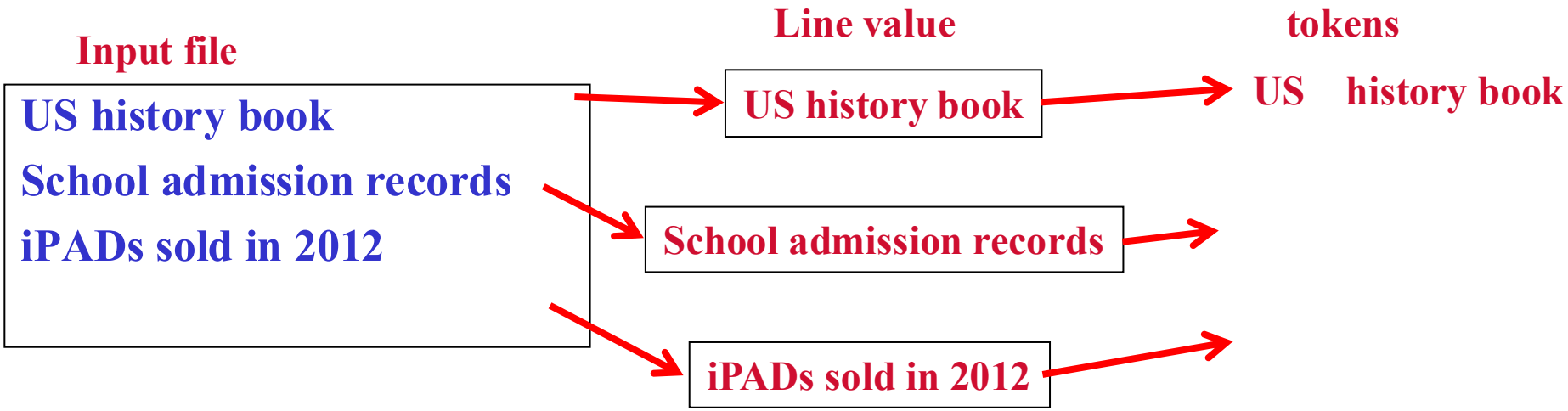
- MapReduce code in Java is popular
- Do not need to worry too much about the location of data on a large cluster storage.
- The system will distribute data, calling your methods to process in parallel

MapReduce WordCount Example

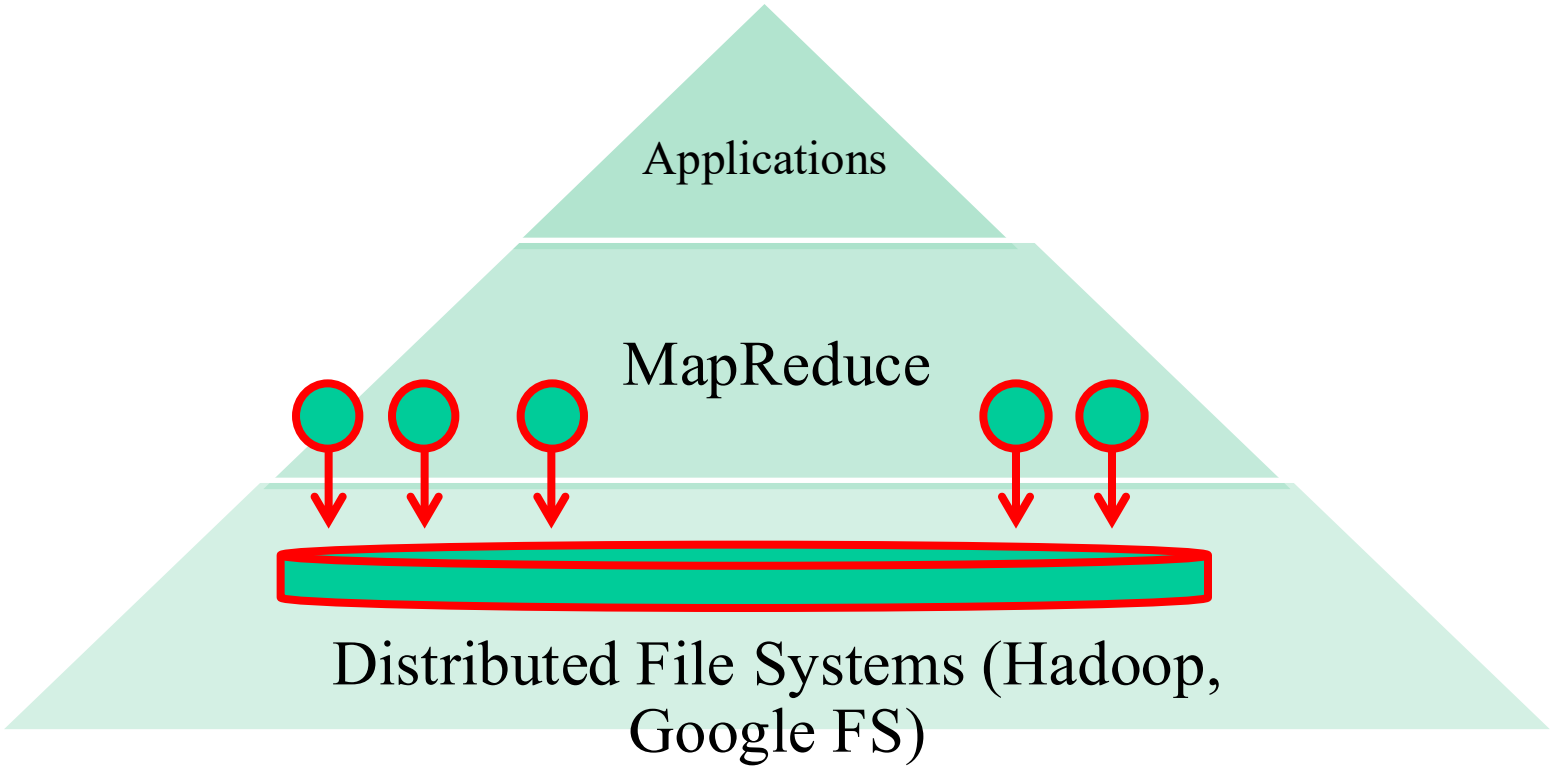
map() gets a key, value, and context

- key - "bytes from the beginning of the line?“
- value - the current line;

in the while loop, each token is a "word" from the current line



Systems Support for MapReduce



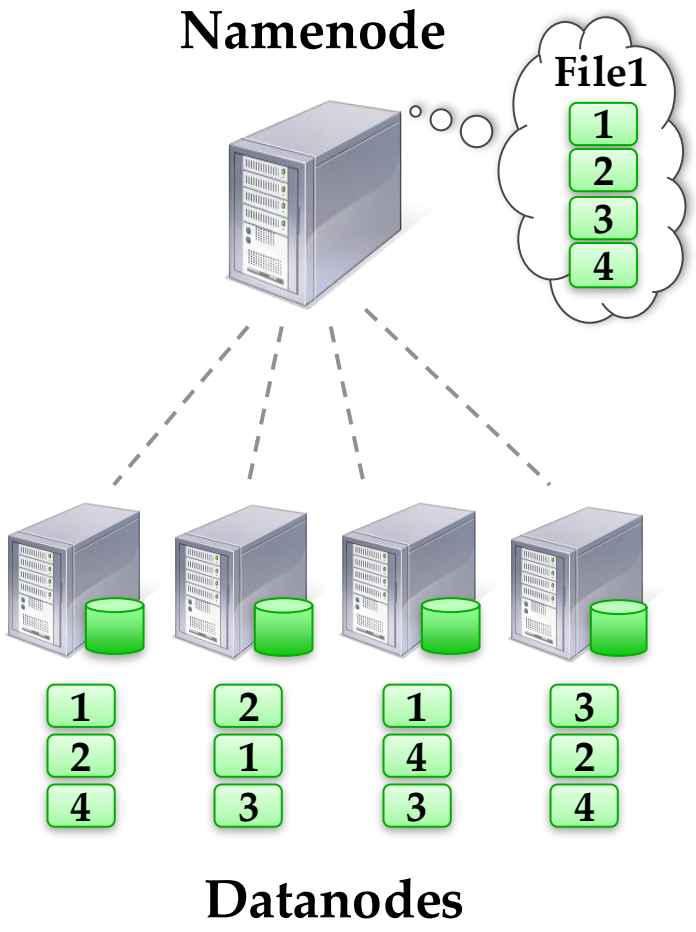
Hadoop and its MapReduce are implemented in Java

Distributed Filesystems

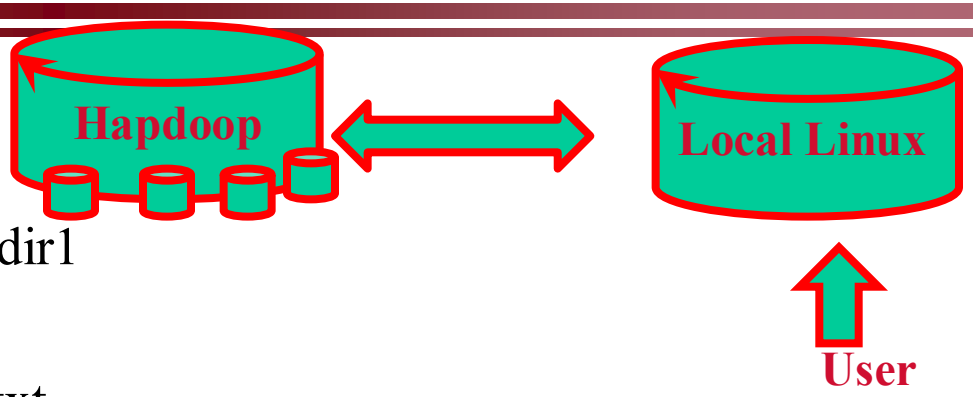
- **Google file system and Hadoop (HDFS)**
- **The interface is the same as a single-machine file system**
  - create(), open(), read(), write(), close()
- **Distribute file data to a number of machines (storage units).**
  - Support replication
- **Support concurrent data access**
  - Fetch content from remote servers. Local caching
- **"Modest" number of HUGE files**
  - Each is 100MB or larger; multi-GB files typical
- **High sustained throughput favoured over low latency**

Hadoop Distributed File System

- **Files split into 64 MB blocks**
- **Blocks replicated across several datanodes (3)**
- **Namenode stores metadata (file names, locations, etc)**
- **Files are append-only. Optimized for large files, sequential reads**
  - Read: use any copy
  - Write: append to 3 replicas



Shell Commands for Hadoop File System

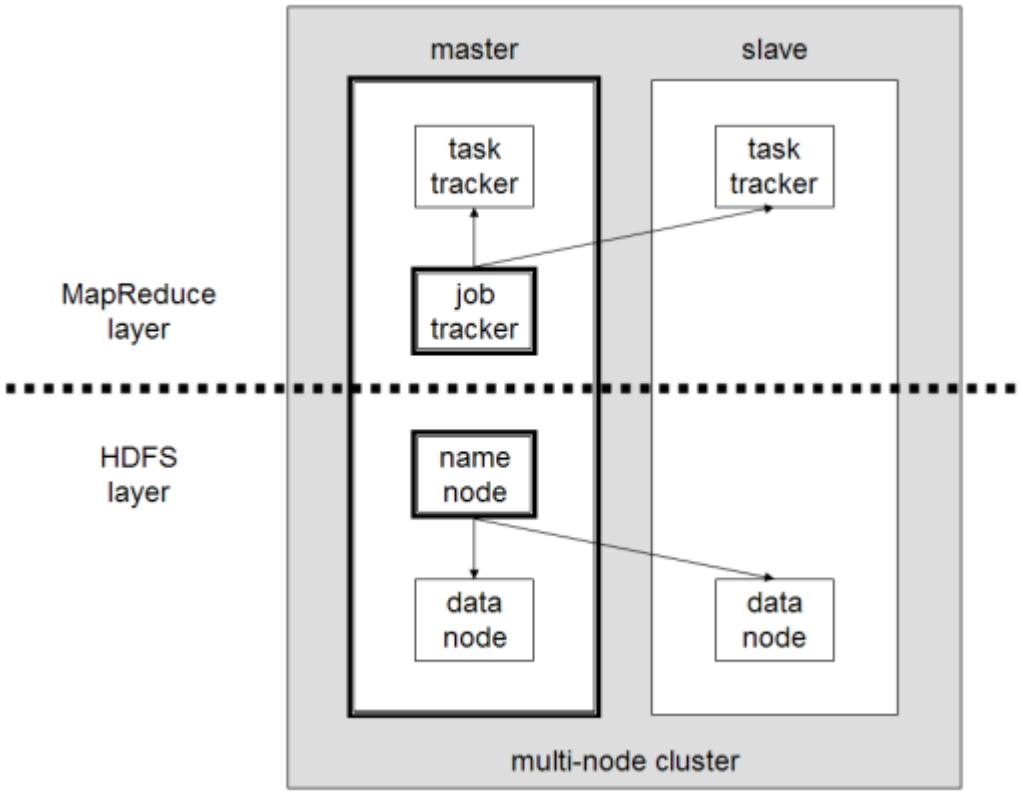


- **Mkdir, ls, cat, cp**
  - hadoop fs -mkdir /user/deepak/dir1
  - hadoop fs -ls /user/deepak
  - hadoop fs -cat /usr/deepak/file.txt
  - hadoop fs -cp /user/deepak/dir1/abc.txt /user/deepak/dir2
- **Copy data from the local file system to HDF**
  - hadoop fs -copyFromLocal <src:localFileSystem> <dest:Hdfs>
  - Ex: hadoop fs –copyFromLocal /home/hduser/def.txt /user/deepak/dir1
- **Copy data from HDF to local**
  - hadoop fs -copyToLocal <src:Hdfs> <dest:localFileSystem>

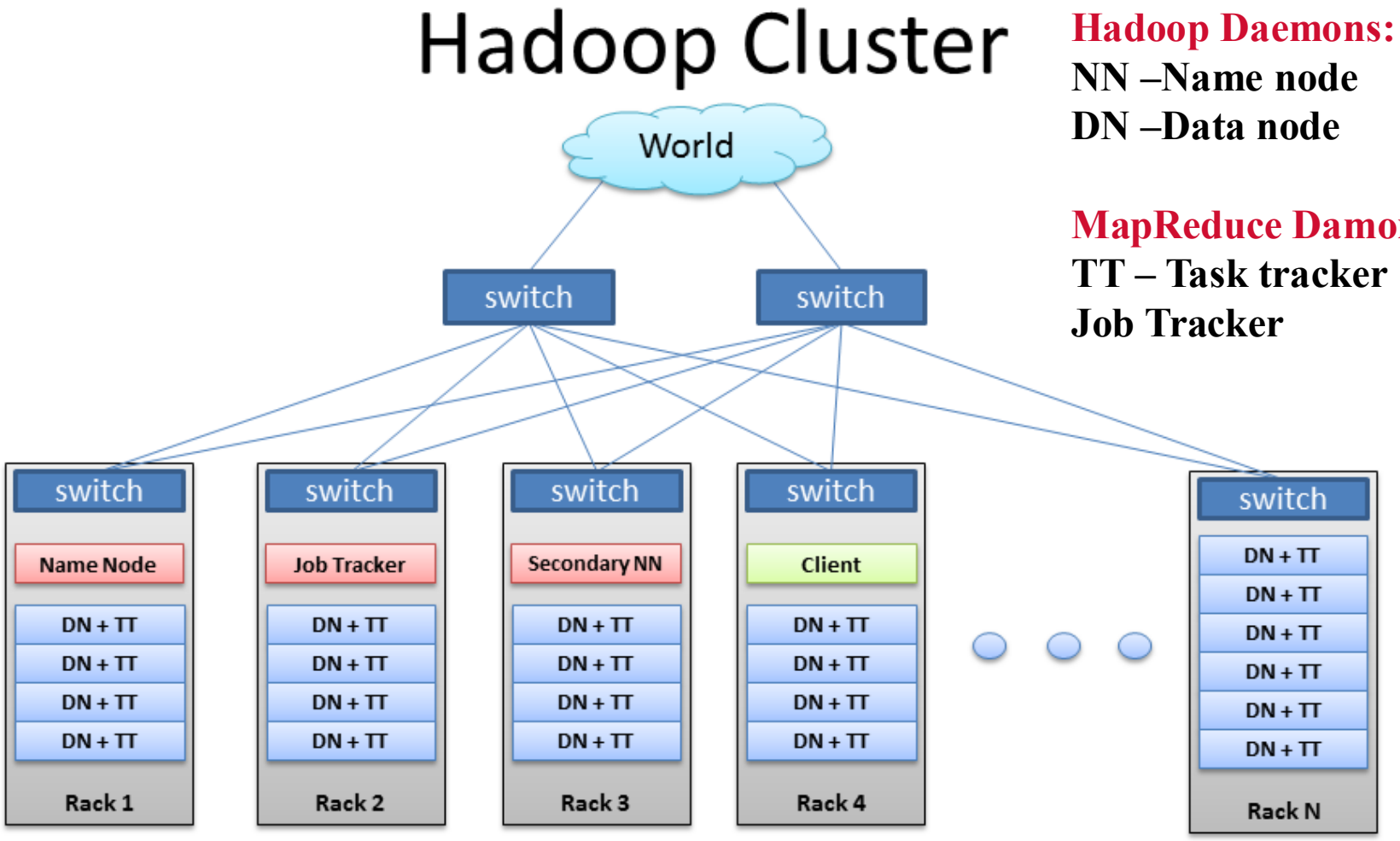
http://www.bigdataplanet.info/2013/10/All-Hadoop-Shell-Commands-you-need-Hadoop-Tutorial-Part-5.html

Daemons for Hadoop/Mapreduce

- **Hadoop daemons**
  - Name node (master) to identify where file blocks are distributed.
  - Secondary name node
  - data nodes to manage the severing of file blocks
- **Mapreduce daemons**
  - Task tracker to execute map/reduce tasks and monitor their status
  - Job tracker to schedule where to execute tasks



Hadoop Cluster with MapReduce

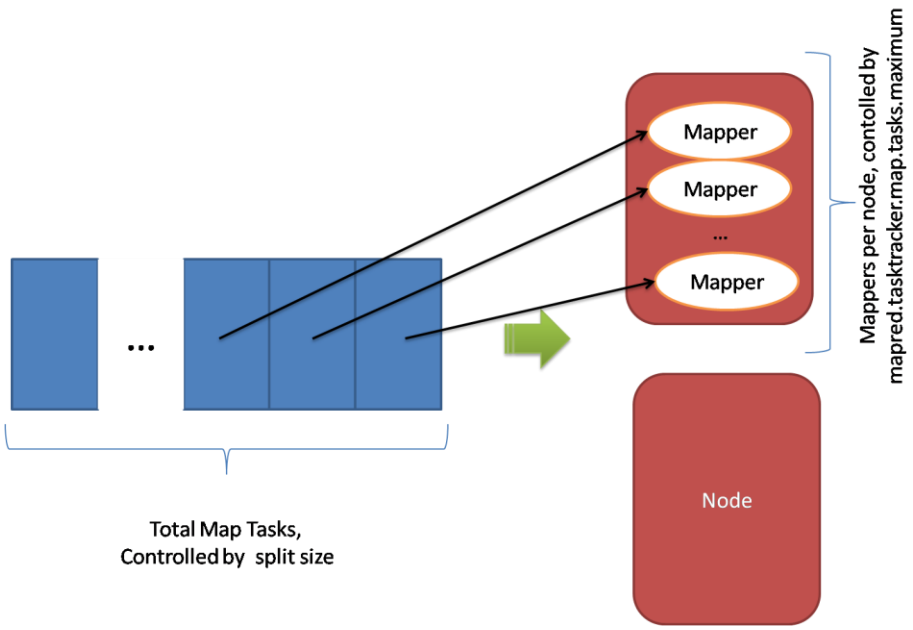


Hadoop Daemons:
NN –Name node
DN –Data node

MapReduce Damons:
TT – Task tracker
Job Tracker

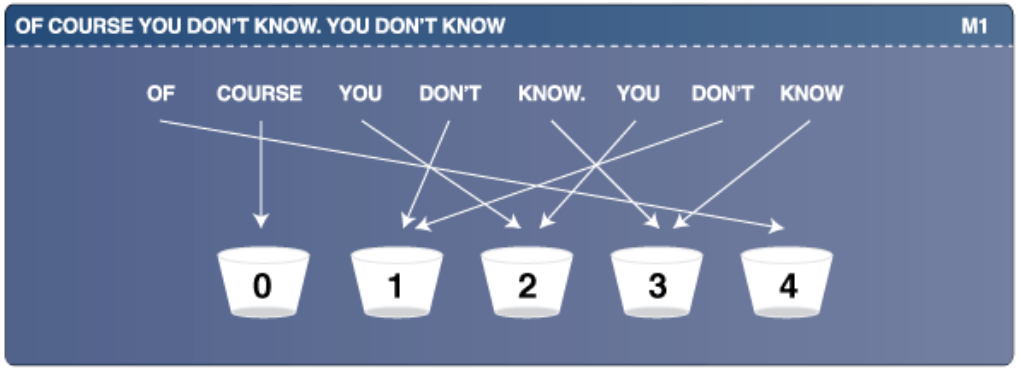
How to create and execute map tasks?

- **The system spawns a number of mapper processes and reducer processes**
  - A typical/default setting 2 mappers and 1 reducer per core.
  - User can specify/change setting
- **Input reader**
  - Input is typically a directory of files.
  - Divide each input file into splits,
  - Assign each split to a Map task
- **Map task**
  - Executed by a mapper process
  - Apply the user-defined map function to each record in the split
  - Each Map function returns a list of (key, value) pairs



How to create and execute reduce tasks?

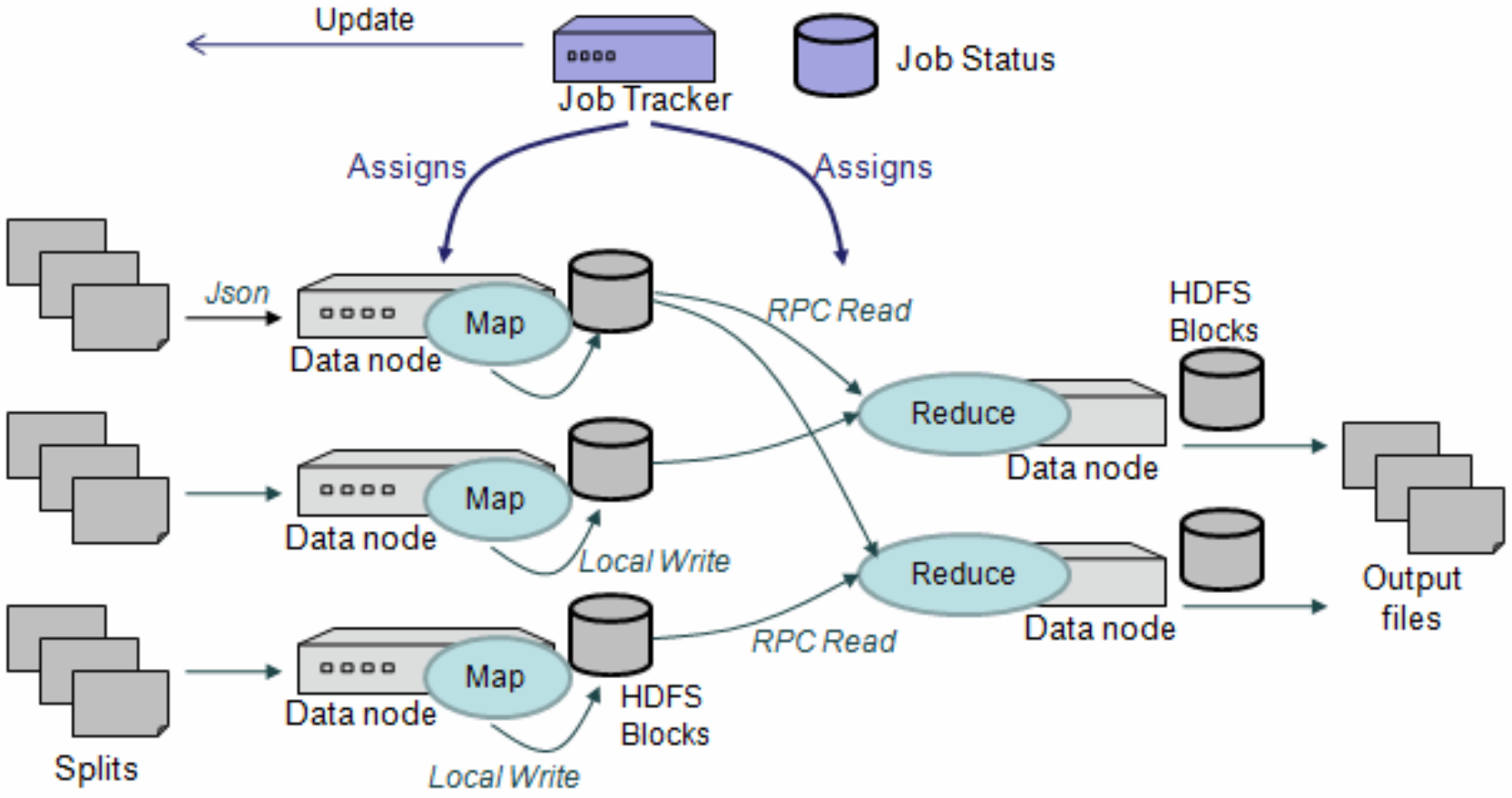
- **Partition (key, value) output pairs of map tasks**



Partitioning is based on hashing, and can be modified.

Key	Hash	Hash % 4
of	-1463488791	4
course	2334184425	0
you	1116843962	2
don't	-482782459	1
know	326123353	3

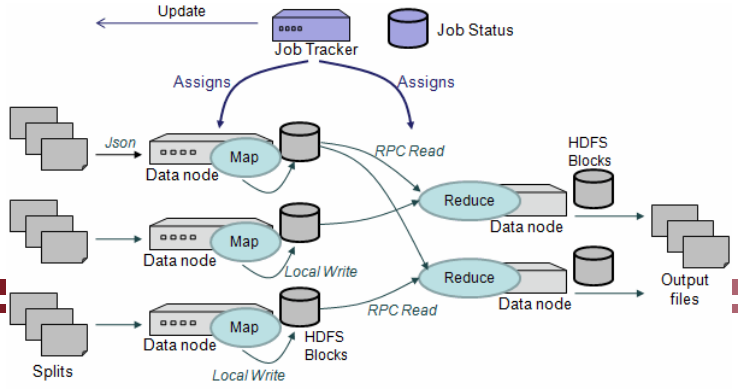
Execute MapReduce on a cluster of machines with Hadoop DFS



MapReduce System: Execution Flow

- **Input reader**
  - Divide input into splits, assign each split to a Map task
- **Map task for data parallelism**
  - Apply the Map function to each record in the split
  - Each Map function returns a list of (key, value) pairs
- **Shuffle/Partition and Sort**
  - Shuffle distributes sorting & aggregation to many reducers
  - All records for key *k* are directed to the same machine that runs the reduce task
  - Sort groups the same keys together, and prepares for aggregation
- **Reduce task for data parallelism**
  - Apply the Reduce function to each key
  - The result of the Reduce function is a list of (key, value) pairs
- **Performance consideration in mappers/reducers: Too many key-value pairs?**

Fault Tolerance Handling



- **What to do when some tasks fail?**

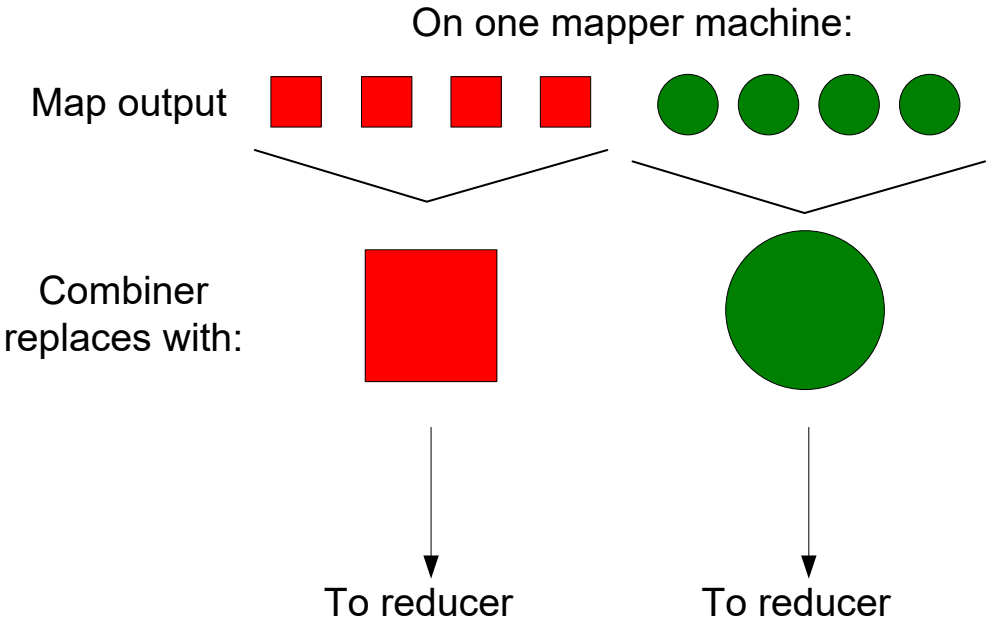
- **Mappers save outputs to local disk before serving to reducers**
  - Allows recovery if a reducer crashes
  - Allows running more reducers than # of nodes
- **If a task crashes:**
  - Retry on another node
  - » OK for a map because it had no dependencies
  - » OK for reduce because map outputs are on disk
  - If the same task repeatedly fails, fail the job or ignore that input block
  - : For the fault tolerance to work, *user tasks must be deterministic and side-effect-free*

2. If a machine node crashes:

- Relaunch its current tasks on other nodes
- Relaunch any maps the node previously ran
- » Necessary because their output files were lost along with the crashed node

User Code Optimization: Combining Phase

- Run on map machines after map phase
  - "Mini-reduce," only on local map output
  - E.g. job.setCombinerClass(Reduce.class);
- save bandwidth before sending data to full reduce tasks
- Requirement: commutative & associative

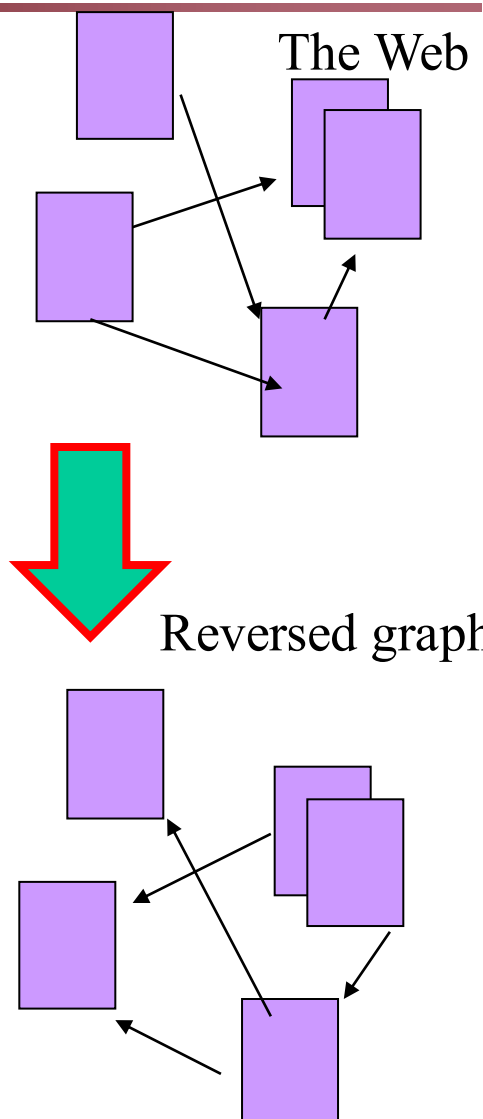


Types of MapReduce Applications

- **Map only parallel processing**
  - Count word usage for each document
  - Distributed grep (search for words)
    - Example: grep “error” *.log
    - *Map*: emit a line if it matches a given pattern within a file
- **Map-reduce two-stage processing**
  - Count word usage for the entire document collection
  - URL access frequency from many web access logs
    - *Map*: process one log file of web page access; output frequency for each URL
    - *Reduce*: add all values for the same URL

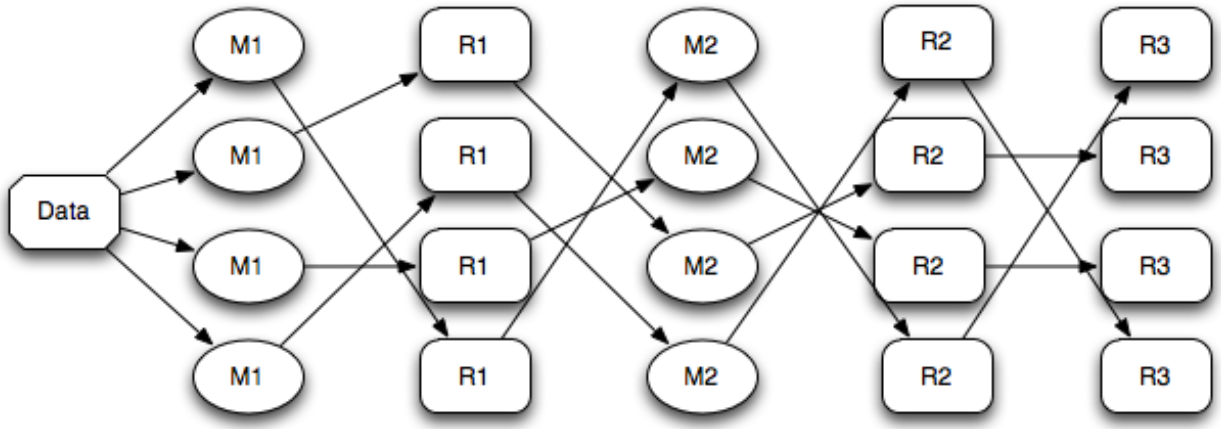
MapReduce Applications: Build a large graph for computing PageRank

- **Input:** a set of web pages and their outgoing links
- **Output:** Reversed web-link graph
  - A set of web page IDs and their incoming links.
- **Parallel code**
  - *Map*: Input is a web page containing outgoing links. Output each link with the target URL as a key.
  - *Reduce*: Concatenate the list of all source pages associated with a target URL



MapReduce Job Chaining

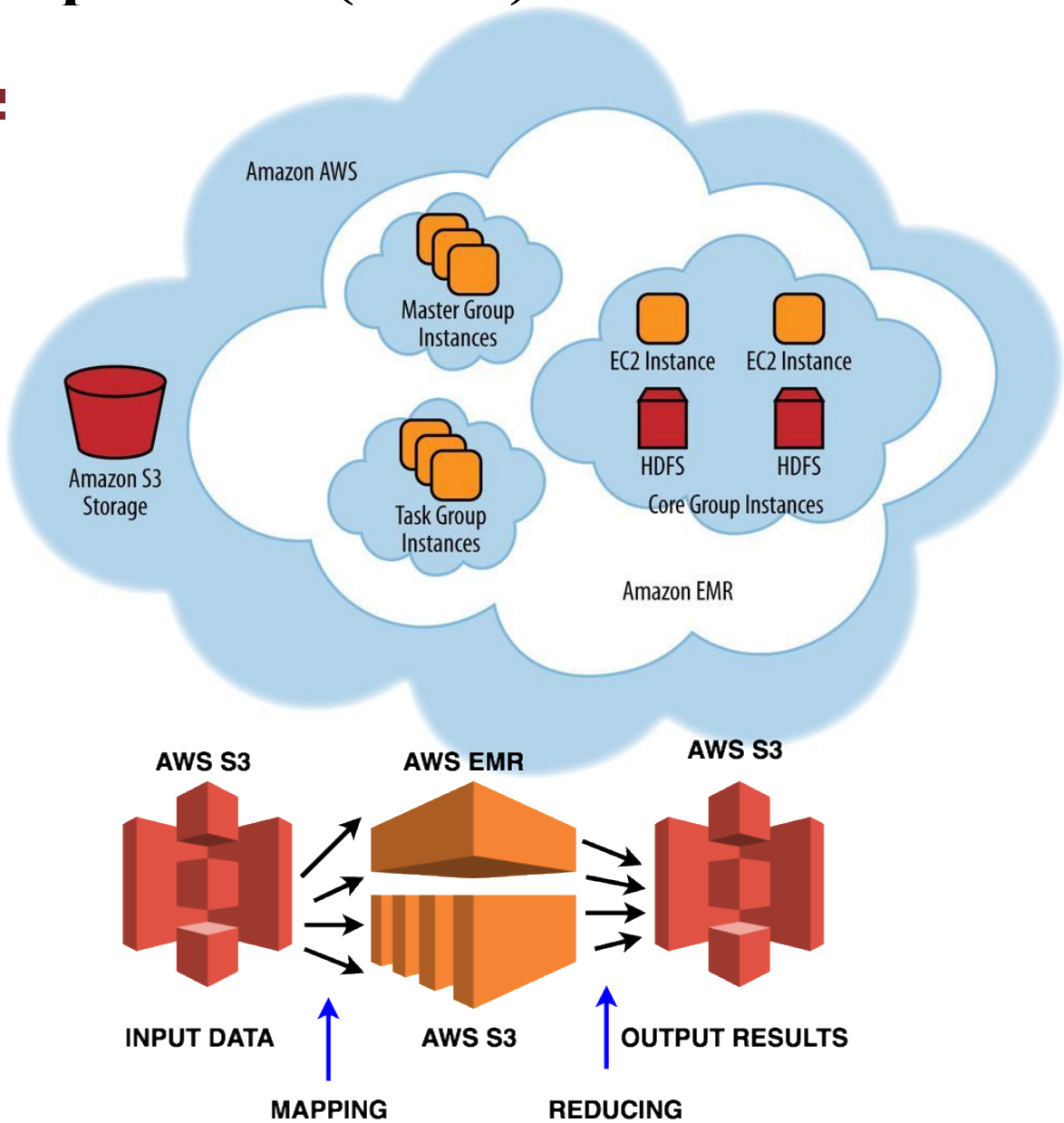
Run a sequence of map-reduce jobs



- **Example**
  1. Count word usage in a document set
  2. Identify most frequent words in each document, but exclude those most popular words in the entire document set

Amazon Elastic MapReduce (EMR)

- Web interface and command-line tools for running Hadoop jobs on EC2
- Data stored in Amazon S3
- Monitors job and shuts machines after use
- Can also create Hadoop clusters manually using scripts included with Hadoop



Summary: MapReduce

- **Architecture**
  - MapReduce with Distributed File System (e.g. Hadoop)
- **Powerful programming abstraction for many data-intensive applications**
  - Word frequency count for documents
  - Product popularity analysis from web user access log
- **Functionalities**
  - Automate handling of large storage and parallel I/O
    - » Parallel & distributed processing with fault-tolerance
  - Suitable for a large batch data processing job which takes hours/days/weeks
    - » Not targeted for processing interactive processing requests with a real time response