Lecture 33

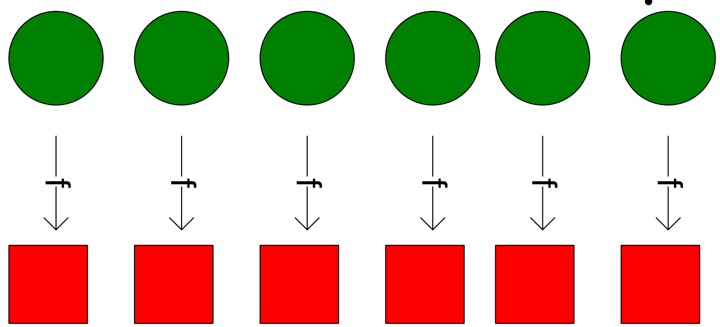
Map-Reduce

Map-Reduce

- Google processes 20 Petabytes/day
- Ebay 150 billion records/day
- Facebook 15 Terabytes/data
- 10,000 server cluster - 10 failures/day
- Era of “Big Data”,

map (Functional Programming)

Creates a new list by applying f to each element of the input list; returns output in order.




```
fun map f []       = []
  | map f (x::xs) = (f x) :: (map f xs)
```

MapReduce Programming Model

- Programmers think in a *data-centric* fashion (transform the data)

- Have the framework handle the Hard Stuff:
  - Fault tolerance
  - Distributed execution, scheduling, concurrency
  - Coordination
  - Network communication

Bacis API for MapReduce

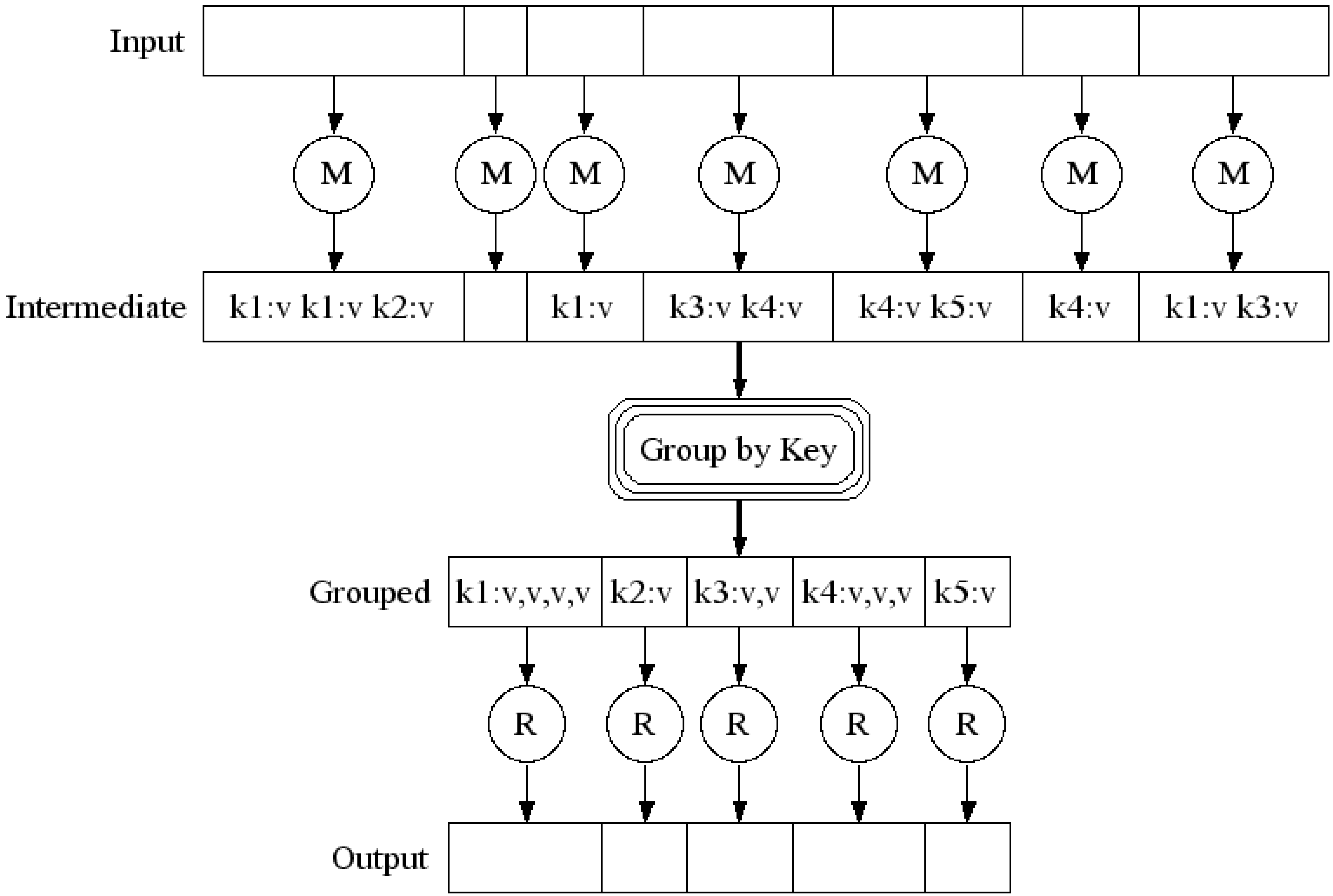
Map: (key1, value1)→[(key2, value2)]

Mapper is applied to every key-value pair that is input and computes one or more pairs of a new key with a new value. (for each key-value emits a list of pairs of key-values)

Reduce: (key2, [value2])→[(key3, value3)]

For each key2 a reducer receives the key2 in sorted order and produces one or more pairs of a new key with a new value. (for each key2 emits a list of pairs of key-values)

Map-Reduce

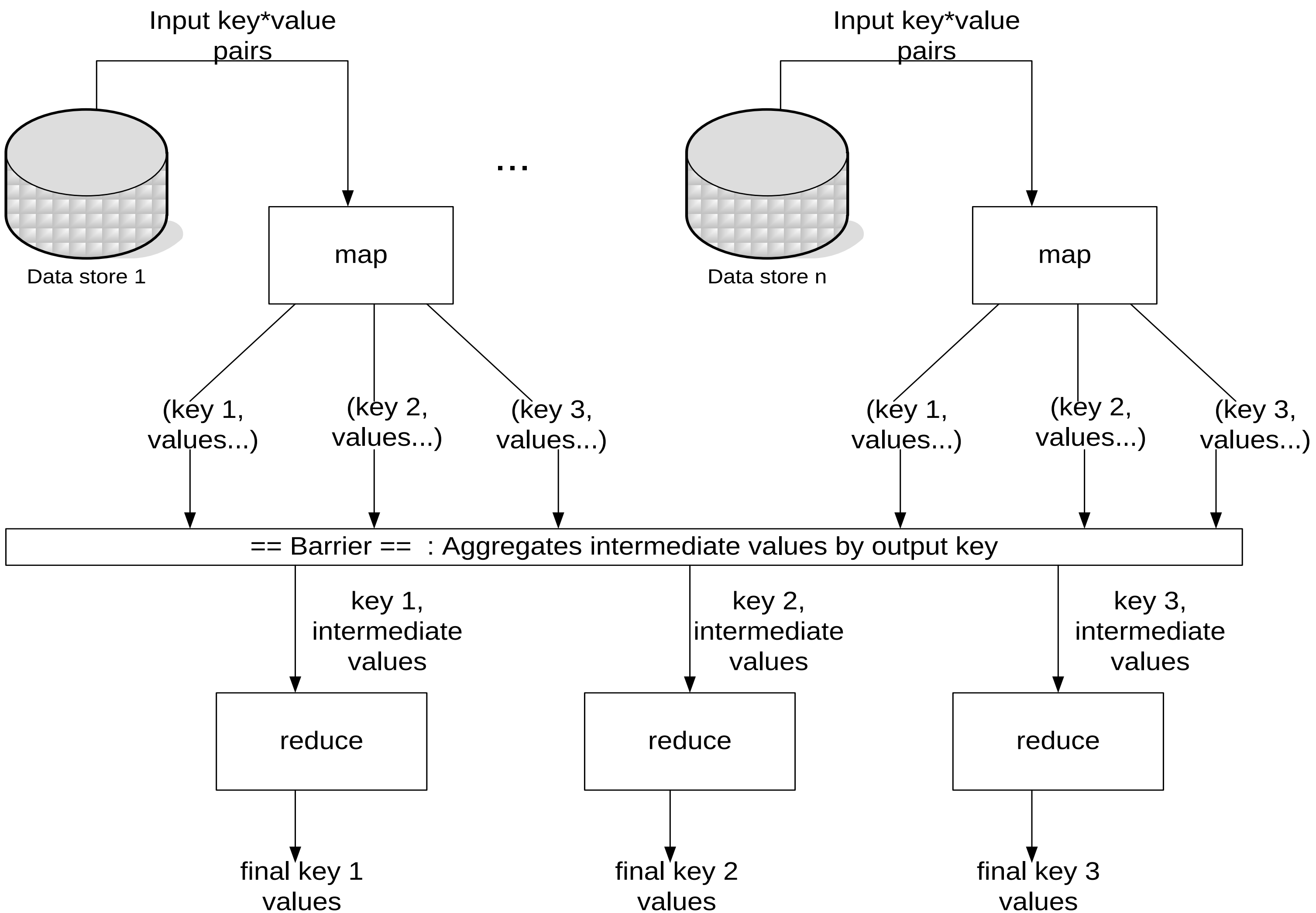

Input
M
M
M
M
M
M
M
Intermediate
k1:v k1:v k2:v
k1:v
k3:v k4:v
k4:v k5:v
k4:v
k1:v k3:v
Group by Key
Grouped
k1:v,v,v,v
k2:v
k3:v,v
k4:v,v,v
k5:v
R
R
R
R
R
Output

MapReduce System Model

- Designed for batch-oriented computations over large data sets (external memory algorithms)
  - Each map-reduce runs to completion before producing any output
  - Map-reduce output is written to stable storage
    - Map output to local disk, reduce output to HDFS
  - Designed to optimize "disk utilization"
    - Keep as many spindles active at once

- Simple, fault tolerance model
  - Can abort and restart a mapper or reducer.
  - check-point restart simple in this model

Map-Reduce System



Word count using MapReduce

Word count for a collection of documents:

Docid 1

This is a cat

Cat sits on a roof

Docid 2

The roof is a tin roof

There is a tin can on the roof

Docid 3

Cat kicks the can

It rolls on the roof and falls on the next roof

Docid 4

The cat rolls too

It sits on the can

Word Count using MapReduce

```
map(key, value):
     // key: document name; value: text of document
  for each word w in value:
      emit(w, 1)


reduce(key, values):
    // key: a word; values: iterator over counts
    result = 0
    for each count v in values:
            result += v
    emit(key,result)
```

Word Count using MapReduce: Mapper

This is a cat
Cat sits on a roof

<this 1> <is 1> <a <1,1,>> <cat <1,1>> <sits 1> <on 1> <roof 1>

The roof is a tin roof
There is a tin can on the roof

<the <1,1>> <roof <1,1,1>> <is <1,1>> <a <1,1>> <tin <1,1>> <then 1> <can 1> <on 1>

Cat kicks the can
It rolls on the roof and falls on the next roof

<cat 1> <kicks 1> <the <1,1>> <can 1> <it 1> <roll 1> <on <1,1>> <roof <1,1>> <and 1> <falls 1> <next 1>

The cat rolls too
It sits on the can

<the <1,1>> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <cat 1>

Combiners

- Often a map task will produce many pairs of the form (k,v1), (k,v2), … for the same key k
  - E.g., popular words in Word Count
- Can save network time by pre-aggregating at mapper
  - combine(k1, list(v1)) → v2
  - Usually same as reduce function
- Works only if reduce function is commutative and associative

Word Count using MapReduce: Combiner, Reducer

<this 1> <is 1> <a <1,1,>> <cat <1,1>> <sits 1> <on 1> <roof 1>

<the <1,1>> <roof <1,1,1>> <is <1,1>> <a <1,1>> <tin <1,1>> <then 1> <can 1> <on 1>

<cat 1> <kicks 1> <the <1,1>> <can 1> <it 1> <roll 1> <on <1,1>> <roof <1,1>> <and 1> <falls 1> <next 1>

<the <1,1>> <cat 1> <rolls 1> <too 1> <it 1> <sits 1> <on 1> <cat 1>

Combine the counts of all the same words:

<cat <1,1,1,1>>

<roof <1,1,1,1,1,1>>

<can <1, 1,1>>

…

Reduce (sum in this case) the counts:

<cat 4>

<can 3>

<roof 6>

Map-Reduce

- Mapper
- Reducers
- Combiners

Data-intensive text processing with MapReduce
Jimmy Lin and Chris Dyer.
Book, Online in English, 2010
(http://lintool.github.com/MapReduceAlgorithms/)

MapReduce/Hadoop

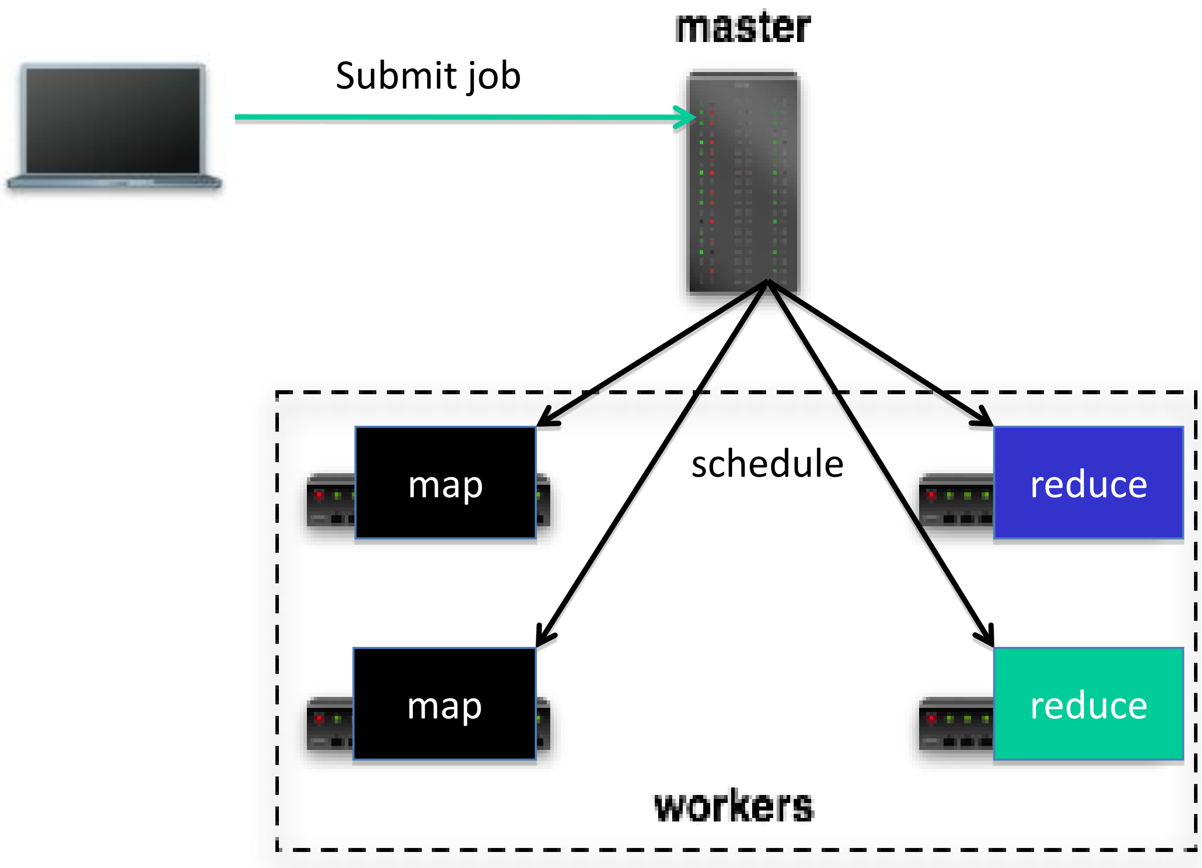
- Around 2008, Yahoo! developed the open-source variant of MapReduce named Hadoop
- After 2008, MapReduce/Hadoop become a key technology component in cloud computing
- In 2010, the U.S. conferred the MapReduce patent to Google

MapReduce

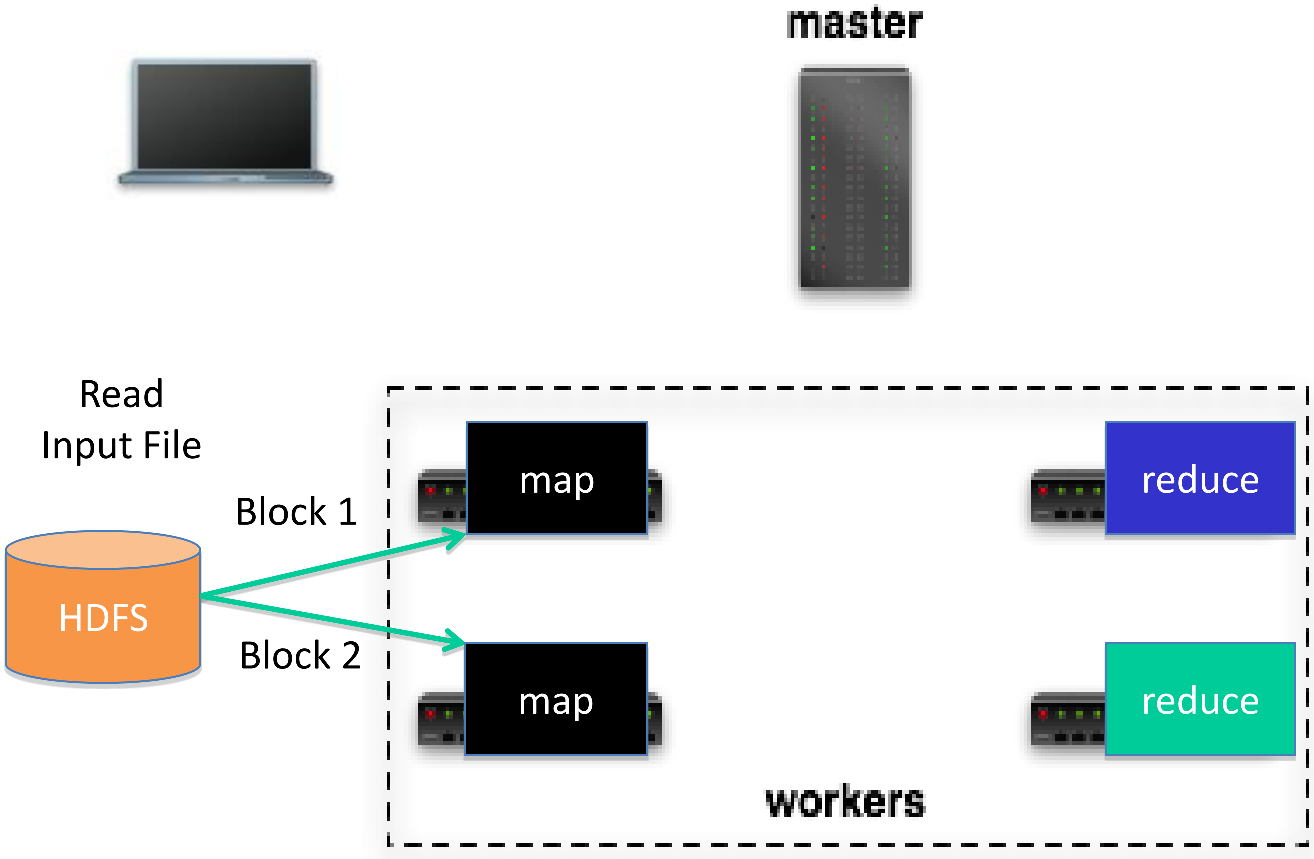
Hadoop

… Hadoop or variants …

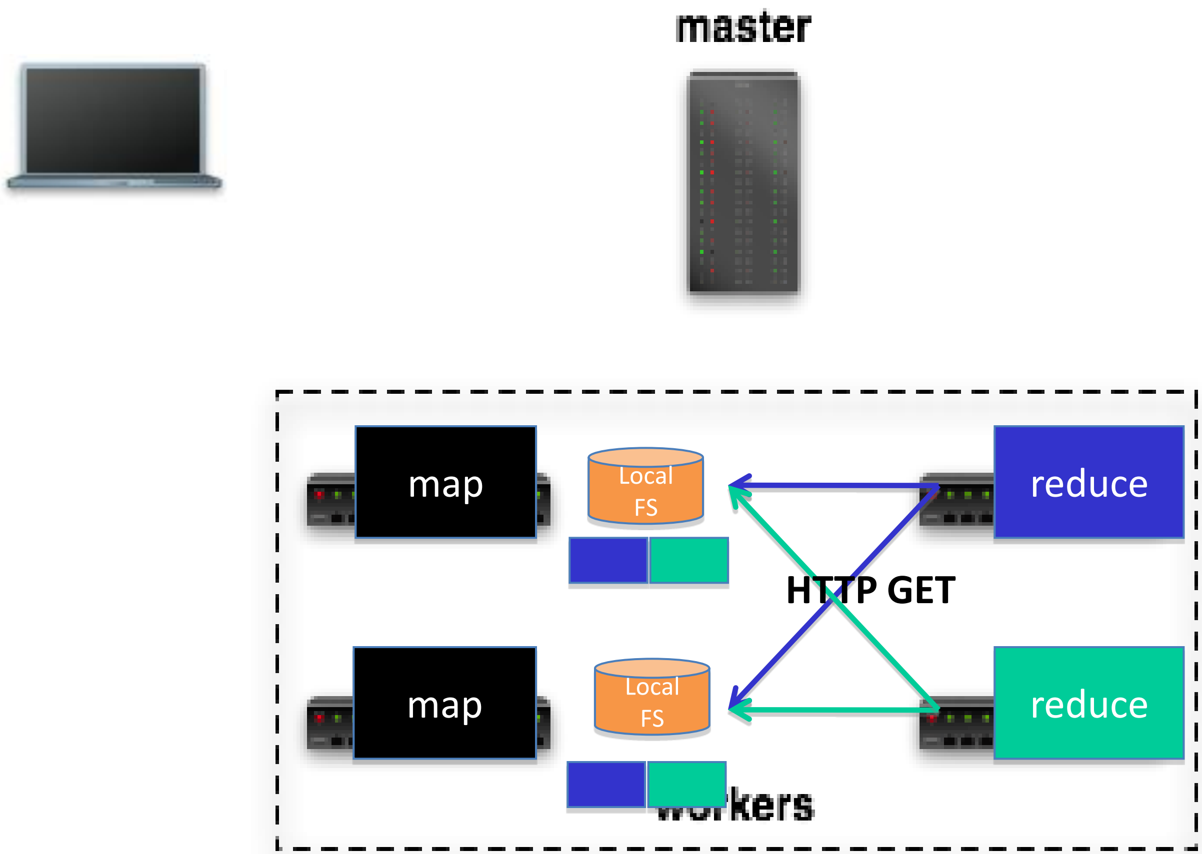
Dataflow in Hadoop



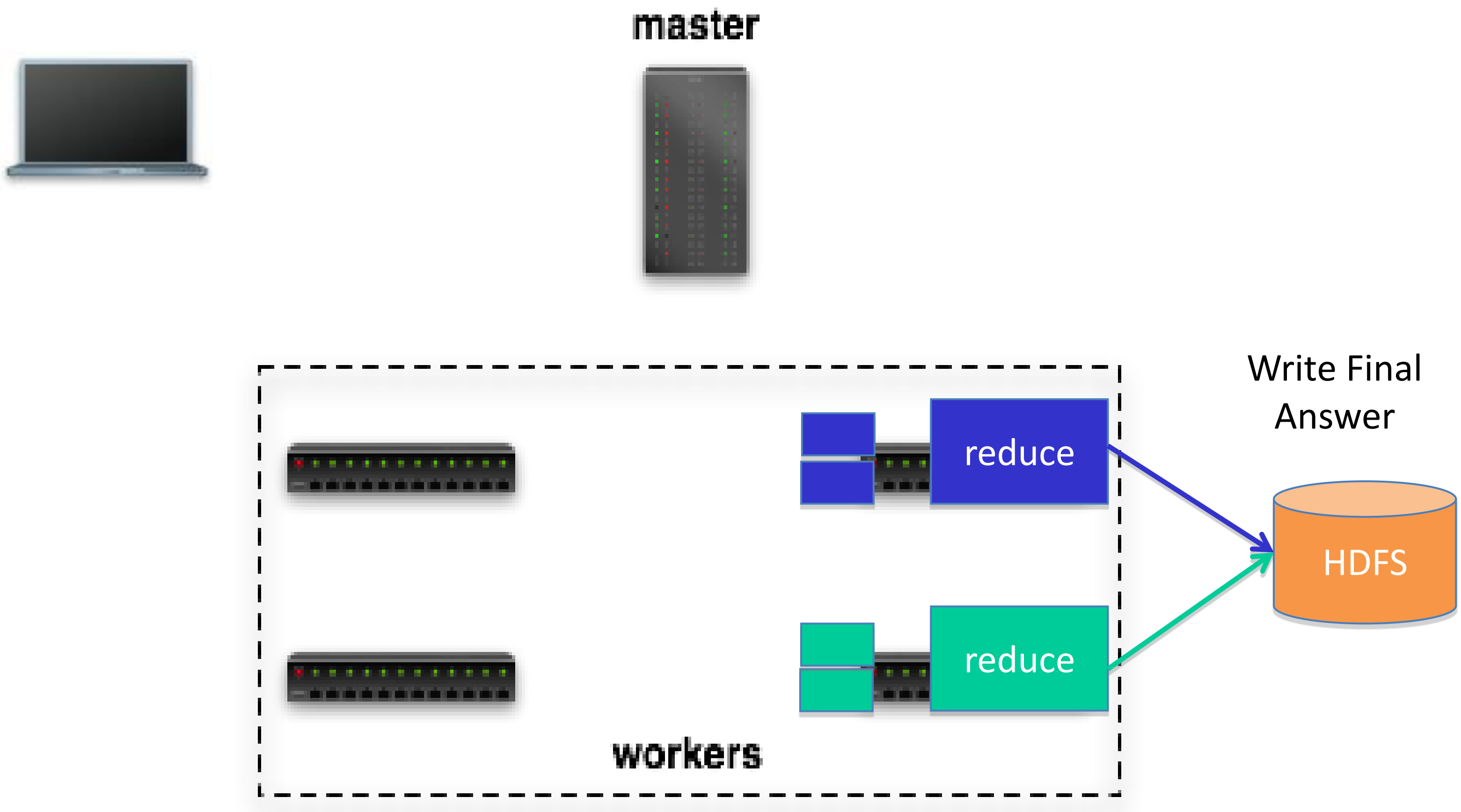
Dataflow in Hadoop



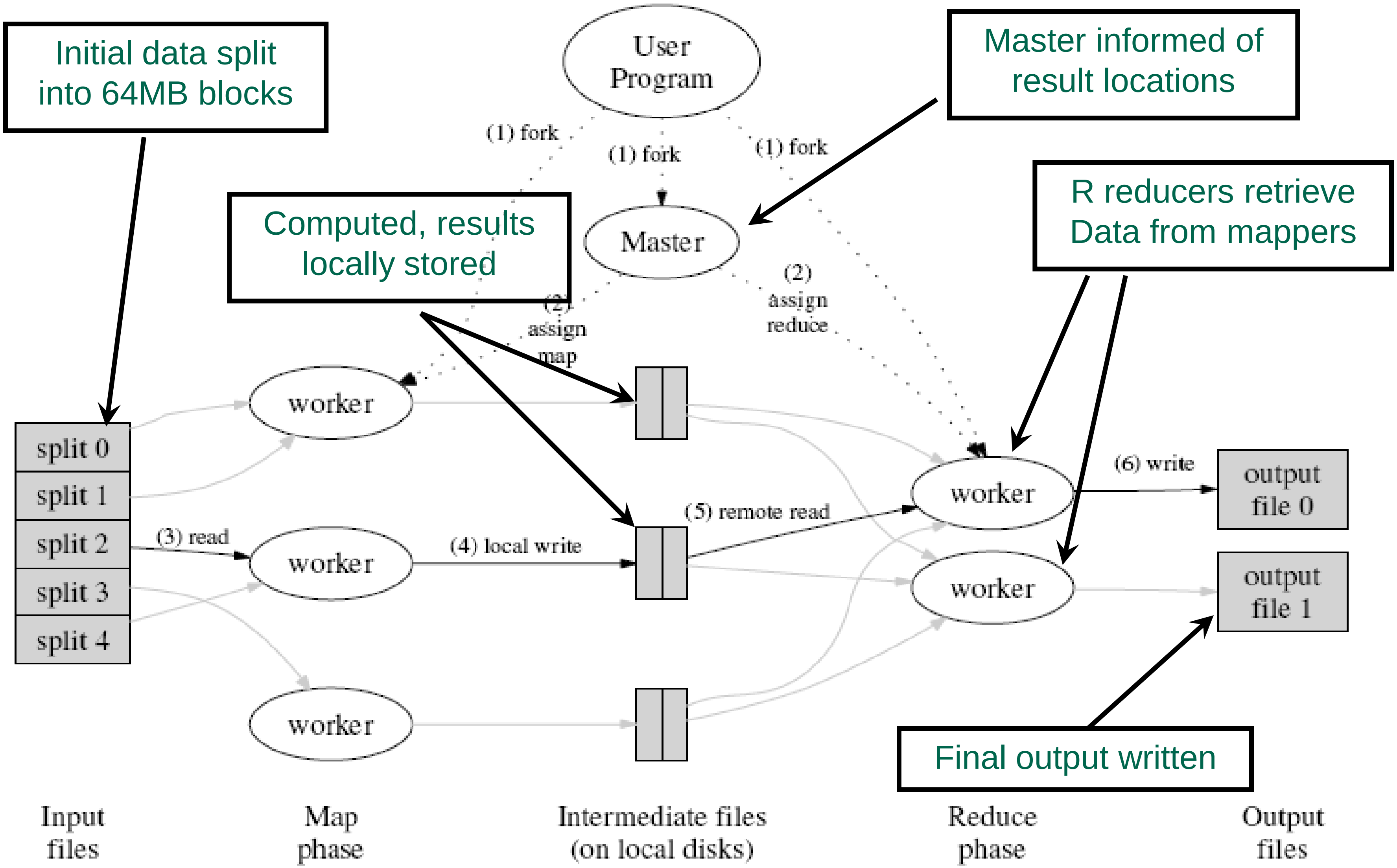
Dataflow in Hadoop



Dataflow in Hadoop



Workflow



Initial data split into 64MB blocks
Computed, results locally stored
Master informed of result locations
R reducers retrieve Data from mappers
Final output written
User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

Critique

MapReduce: A major step backwards
-- David J. DeWitt and Michael Stonebraker

(MapReduce) is

- A giant step backward in the programming paradigm for large-scale data intensive applications
- A sub-optimal implementation, in that it uses brute force instead of indexing
- Not novel at all
- Missing features
- Incompatible with all of the tools DBMS users have come to depend on

Questions

- MapReduce provides an easy-to-use framework for parallel programming, but is it the most efficient and best solution to program execution in datacenters?

- Comparison to database systems:
  - MapReduce is not a database system, nor a query language
  - It is possible to use MapReduce to implement some of the parallel query processing functions
  - What are the real limitations?
  - Scalability

Limitations

- Inefficient for general programming (and not designed for that)
  - Hard to handle data with complex dependence, frequent updates, etc.
  - High overhead, bursty I/O, streaming data
- Difficult to optimize (limited opportunities)
- Hadoop
  - Typically orders of magnitude slower than a parallel computing solution
  - BUT, environment is different because it depends on file-system (designed for scalability!)

Limitations

- Proprietary solution developed in an environment with one prevailing application (web search)
  - The assumptions introduce several important constraints in data and logic
  - Not a general-purpose parallel execution technology
- Design choices in MapReduce
  - Optimizes for throughput rather than latency
  - Optimizes for large data set rather than small data structures (external data algorithms)
  - Optimizes for coarse-grained parallelism rather than fine-grained

Presentations

Josh Carter, http://multipart-mixed.com/software/mapreduce_presentation.pdf

Ralf Lammel, Google's MapReduce Programming Model – Revisited
http://code.google.com/edu/parallel/mapreduce-tutorial.htm

References

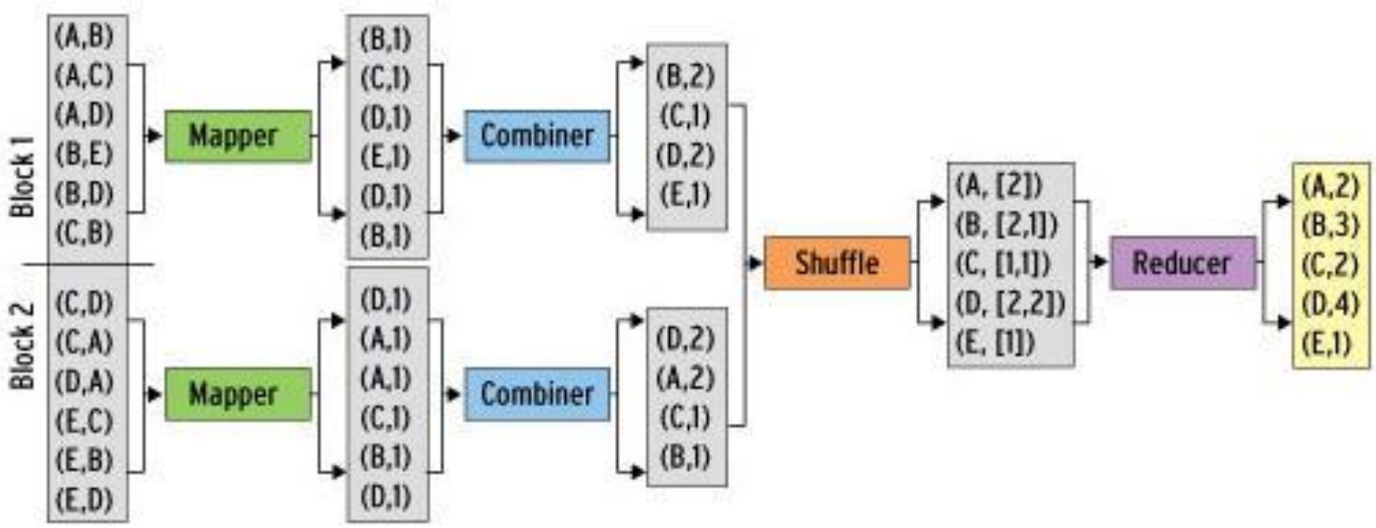
- [Al-Fares] Al-Fares,M., Loukissas, A., and Vahdat, A. A scalable, commodity data center network architecture. In Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (Seattle, WA, USA, August 17 - 22, 2008). SIGCOMM '08. 63-74.
- [Andersen] David G. Andersen, Jason Franklin, Michael Kaminsky, Amar Phanishayee, Lawrence Tan, Vijay Vasudevan. FAWN: A Fast Array of Wimpy Nodes. SOSP'09.
- [Barraso] Luiz Barroso, Jeffrey Dean, Urs Hoelzle, "Web Search for a Planet: The Google Cluster Architecture," IEEE Micro, vol. 23, no. 2, pp. 22-28, Mar./Apr. 2003
- [Ghemawat] Ghemawat, S., Gobioff, H., and Leung, S. 2003. The Google file system. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (Bolton Landing, NY, USA, October 19 - 22, 2003). SOSP '03. ACM, New York, NY, 29-43.
- [Guo 2009] Chuanxiong Guo, Guohan Lu, Dan Li, Xuan Zhang, Haitao Wu, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu, BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers, in ACM SIGCOMM 09.

- [Chang] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E. Bigtable: a distributed storage system for structured data. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation (Seattle, Washington, November 06 - 08, 2006). 205-218.
- [Cooper] Cooper, B. F., Ramakrishnan, R., Srivastava, U., Silberstein, A., Bohannon, P., Jacobsen, H., Puz, N., Weaver, D., and Yerneni, R. PNUTS: Yahoo!'s hosted data serving platform. Proc. VLDB Endow. 1, 2 (Aug. 2008), 1277-1288.
- [Dean] Dean, J. and Ghemawat, S. 2004. MapReduce: simplified data processing on large clusters. In Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6 (San Francisco, CA, December 06 - 08, 2004).
- [Ghemawat] Ghemawat, S., Gobioff, H., and Leung, S. 2003. The Google file system. In Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles (Bolton Landing, NY, USA, October 19 - 22, 2003). SOSP '03. ACM, New York, NY, 29-43.
- [Guo 2009] Chuanxiong Guo, Guohan Lu, Dan Li, Xuan Zhang, Haitao Wu, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu, BCube: A High Performance, Server-centric Network Architecture for Modular Data Centers, in ACM SIGCOMM 09.

Combiners

- Often a map task will produce many pairs of the form (k,v1), (k,v2), … for the same key k
  - E.g., popular words in Word Count
- Can save network time by pre-aggregating at mapper
  - combine(k1, list(v1)) → v2
  - Usually same as reduce function
- Works only if reduce function is commutative and associative

Word Count Example

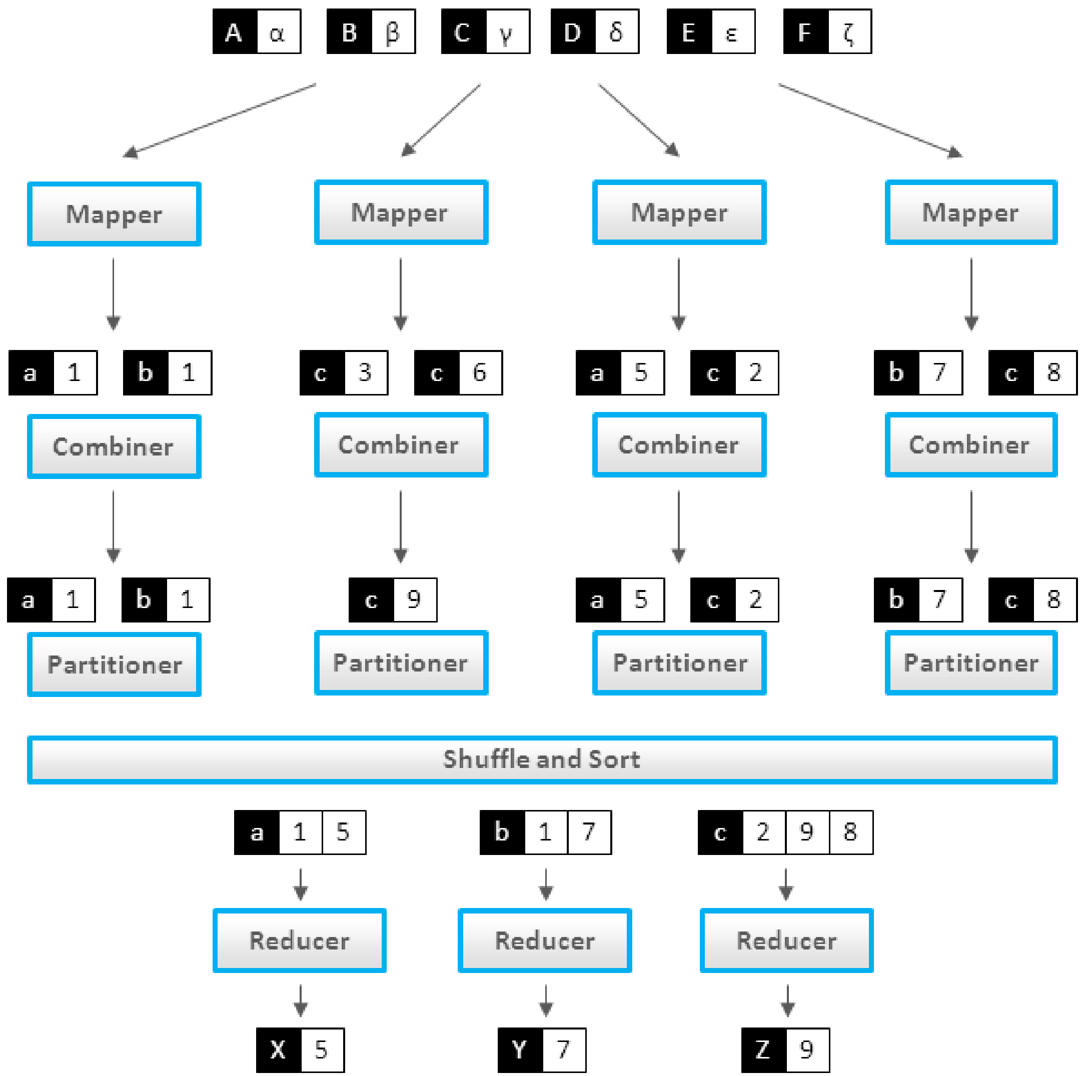


Partition Function

- Inputs to map tasks are created by contiguous splits of input file
- For reduce, we need to ensure that records with the same intermediate key end up at the same worker
- System uses a default partition function e.g., hash(key) mod R
- Sometimes useful to override
  - E.g., hash(hostname(URL)) mod R ensures URLs from a host end up in the same output file

Map-Reduce

Word Count using MapReduce



Text Processing --Co-occurrence

Co-occurrence of two words in a document, in general co-occurrence with respect to some **context**

Docid 1

This is a cat

Cat sits on a roof

Docid 2

The roof is a tin roof

There is a tin can on the roof

	This	Is	A	Cat	Sits	On	The	Roof	Tin	There	can
This	-	1	1	1	1	1	0	1	1	1	1
Is		-	2	1	1	2				1	
A			-							1	
Cat				-						0	
Sit					-						
On						-					
The							-				
Roof								-			
Tin									-		
There										-	1
can											-

Relative Frequency

Frequency of one co-occurrence with all the co-occurrence:

$$f(w_j | w_i) = \frac{N(w_i, w_j)}{\sum_{w'} N(w_i, w')}$$

Relative Frequency using MapReduce

Collection of documents:

Docid 1

This is a cat

Cat sits on a roof

Docid 2

The roof is a tin roof

There is a tin can on the roof

Docid 3

Cat kicks the can

It rolls on the roof and falls on the next roof

Docid 4

The cat rolls too

It sits on the can

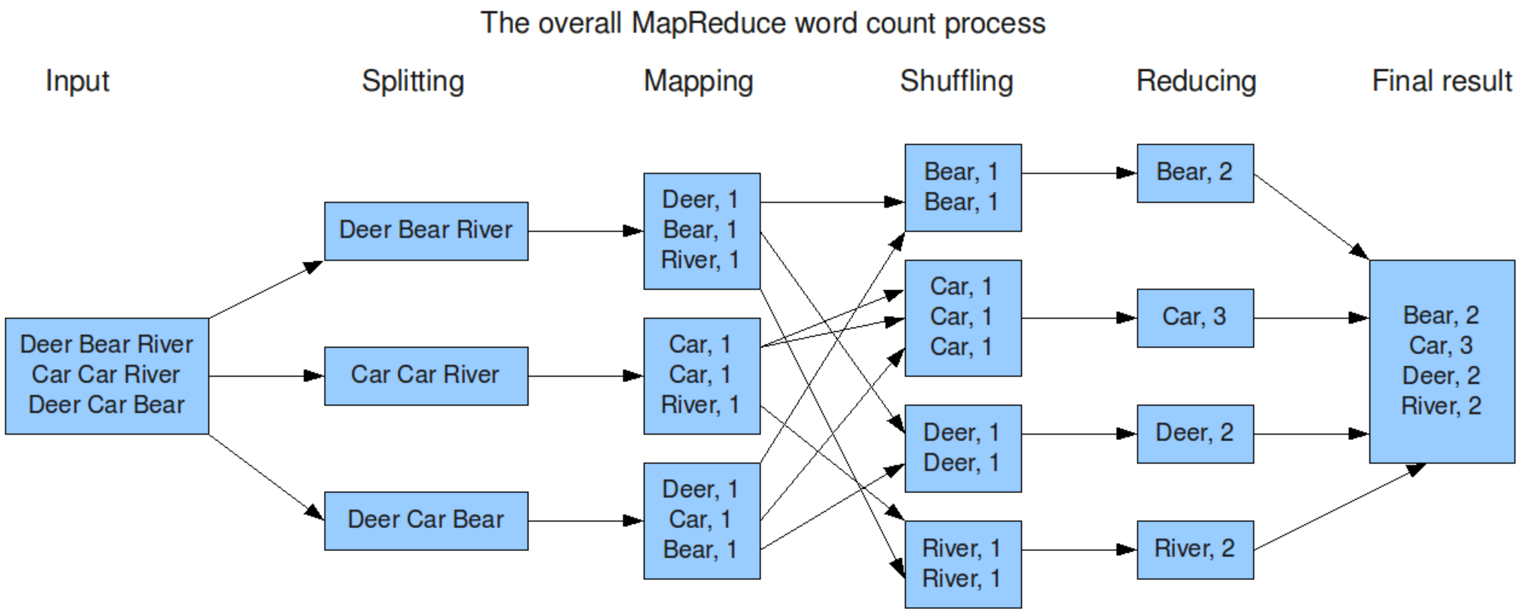
Order Inversion

- Emitting a special key-value for each pair to capture the contribution to the sum
- Controlling the sort order of the intermediate key so that the sum communication appears before the pairs
- Defining a custom partitioner to ensure the pairs are sorted in lexigraphical order and appear at the same reducer
- Preserve state across multiple keys in the reducer to first compute total sum and then dividing this with the co-occurrence value.

Data-Warehousing (OLAP)

- One-to-one Joins
  - Reduce-side joining
- One-to-many Joins
- Many-to-many Joins
- Simple merging with the mapper

Word Count Example



Word Count using MapReduce

```
map(key, value):
     // key: document name; value: text of document
   for each word w in value:
      emit(w, 1)


reduce(key, values):
    // key: a word; values: iterator over counts
    result = 0
    for each count v in values:
            result += v
    emit(key,result)
```