

Introduction to Map Reduce

Map Reduce: Motivation

*“We realized that most of our computations involved applying a **map** operation to each logical record in our input in order to compute a set of intermediate key/value pairs, and then applying a **reduce** operation to all the values that shared the same key in order to combine the derived data appropriately.”*

*“The issues of how to **parallelize** the computation, **distribute** the data, and **handle failures** conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.”*

Dean, Ghemawat. MapReduce: simplified data processing on large clusters. CACM. ACM 51, 1 January 2008

Problem Scope

- Need to scale to 100s or 1000s of computers, each with several processor cores
- How large is the amount of work?
 - Web-Scale data on the order of 100s of GBs to TBs or PBs
 - It is likely that the input data set will not fit on a single computer's hard drive
 - Hence, a distributed file system (e.g., Google File System- GFS) is typically required

Problem Scope

- Scalability to large data volumes:
 - Scan 1000 TB on 1 node @ 100 MB/s = 24 days
 - Scan on 1000-node cluster = 35 minutes
- Required functions
 - Automatic parallelization & distribution
 - Fault-tolerance
 - Status and monitoring tools
 - A clean abstraction for programmers
 - Functional programming meets distributed computing
 - A batch data processing system



Commodity Clusters

- Need to efficiently process large volumes of data by connecting many commodity computers together to work in parallel
- A theoretical 1000-CPU machine would cost a very large amount of money, far more than 1000 single-CPU or 250 quad-core machines

Mapreduce & Hadoop - History

- 2003: Google publishes about its cluster architecture & distributed file system (GFS)
- 2004: Google publishes about its MapReduce model used on top of GFS
 - Both GFS and MapReduce are written in C++ and are closed-source, with Python and Java APIs available to Google programmers only
- 2006: Apache & Yahoo! -> Hadoop & HDFS
 - open-source, Java implementations of Google MapReduce and GFS with a diverse set of API available to the public
 - Evolved from Apache Lucene/Nutch open-source web search engine
- 2008: Hadoop becomes an independent Apache project
 - Yahoo! Uses Hadoop in production
- Today: Hadoop is used as a general-purpose storage and analysis platform for big data
 - Other Hadoop distributions from several vendors including EMC, IBM, Microsoft, Oracle, Cloudera, etc.
 - Many users (<http://wiki.apache.org/hadoop/PoweredBy>)
 - Research and development actively continues...

Google Cluster Architecture: Key Ideas

- **Single-thread performance doesn't matter**
 - For large problems, **total throughput/\$** is more important than peak
- **Stuff breaks**
 - If you have 1 server, it may stay up three years (1,000 days).
 - If you have 10,000 servers, expect to lose 10 per day.
- **“Ultra-reliable” hardware doesn't really help**
 - At large scales, the most reliable hardware still fails, albeit less often
 - Software still needs to be fault-tolerant
 - Commodity machines without fancy hardware give better **performance/\$**
- Have a reliable computing infrastructure from clusters of unreliable commodity PCs.
- Replicate services across many machines to increase request throughput and availability.
- Favor price/performance over peak performance.

What Makes MapReduce Unique?

- Its **simplified programming model** which allows the user to quickly write and test distributed systems
- Its efficient and **automatic distribution** of data and workload across machines
- Its **flat scalability** curve. Specifically, after a Mapreduce program is written and functioning on 10 nodes, very little-if any- work is required for making that same program run on 1000 nodes.
- MapReduce ties smaller and more reasonably priced machines together into a single cost-effective **commodity cluster**

Isolated Tasks

MapReduce divides the workload into multiple independent tasks and schedules them across cluster nodes

A work performed by each task is done in isolation from one another

The amount of communication which can be performed by tasks is mainly limited for scalability reasons

The communication overhead required to keep the data on the nodes synchronized at all times would prevent the model from performing reliably and efficiently at large scale

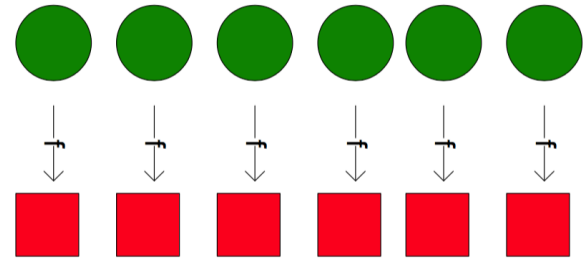
MapReduce in a Nutshell

- Given:
 - a very large dataset
 - a well-defined computation task to be performed on elements of this dataset (preferably, in a parallel fashion on a large cluster)
- Map Reduce framework:
 - Just express what you want to compute (map() & reduce()).
 - Don't worry about parallelization, fault tolerance, data distribution, load balancing (MapReduce takes care of these).
 - What changes from one application to another is the actual computation; the programming structure stays similar.
- In simple terms
 - Read lots of data.
 - Map: extract something that you care about from each record.
 - Shuffle and sort.
 - Reduce: aggregate, summarize, filter, or transform.
 - Write the results.
- One can use as many Maps and Reduces as needed to model a given problem.

Functional programming “foundations”

Note: There is no precise 1-1 correspondence. Please take this just as an analogy.

- map in MapReduce \leftrightarrow map in FP
 - $\text{map}::(a \rightarrow b) \rightarrow [a] \rightarrow [b]$
 - Example: Double all numbers in a list.
 - ```
> map ((* 2) [1, 2, 3])
> [2, 4, 6]
```

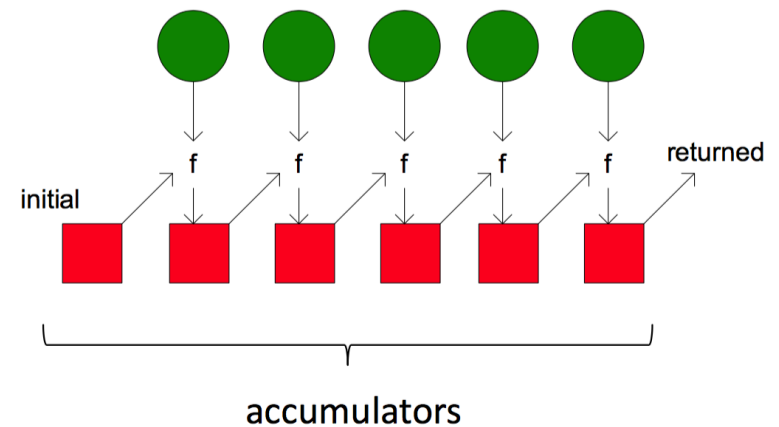


- In a purely functional setting, an element of a list being computed by map **cannot see the effects** of the computations on other elements.
- If the order of application of a function  $f$  to elements in a list is commutative, then we can **reorder or parallelize** execution.

# Functional programming “foundations”

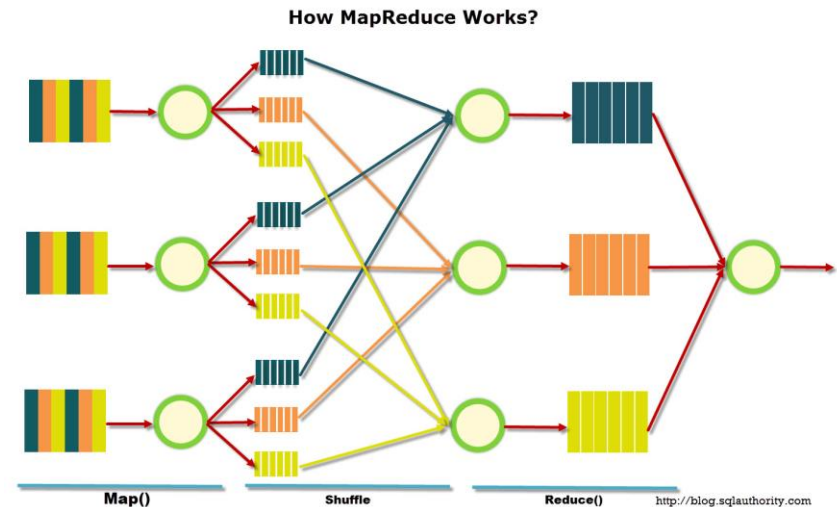
Note: There is no precise 1-1 correspondence. Please take this just as an analogy.

- Move over the list, apply **f** to each element and an **accumulator**. **f** returns the next accumulator value, which is combined with the next element.
- reduce in MapReduce  $\leftrightarrow$  fold in FP
  - $\text{foldl} :: (b \rightarrow a \rightarrow b) \rightarrow b \rightarrow [a] \rightarrow b$
  - Example: Sum of all numbers in a list.
  - $> \text{foldl } (+) 0 [1, 2, 3]$   $\text{foldl } (+) 0 [1, 2, 3]$   
 $> 6$



# MapReduce Basic Programming Model

- Transform a set of input key-value pairs to a set of output values:
  - Map:  $(k1, v1) \rightarrow \text{list}(k2, v2)$
  - MapReduce library groups all intermediate pairs with same key together.
  - Reduce:  $(k2, \text{list}(v2)) \rightarrow \text{list}(v2)$



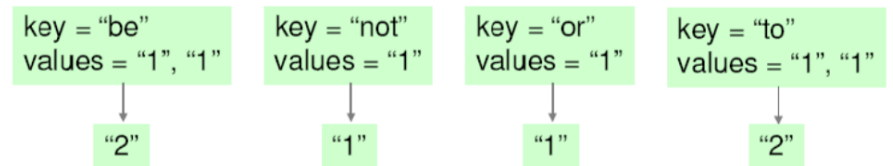
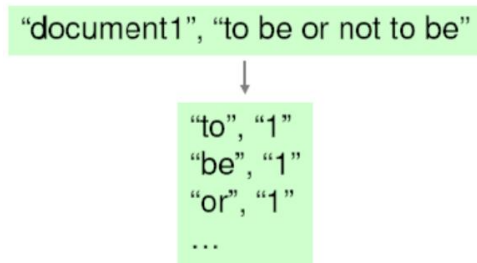
# Word Count

**map(k1, v1) → list(k2, v2)**

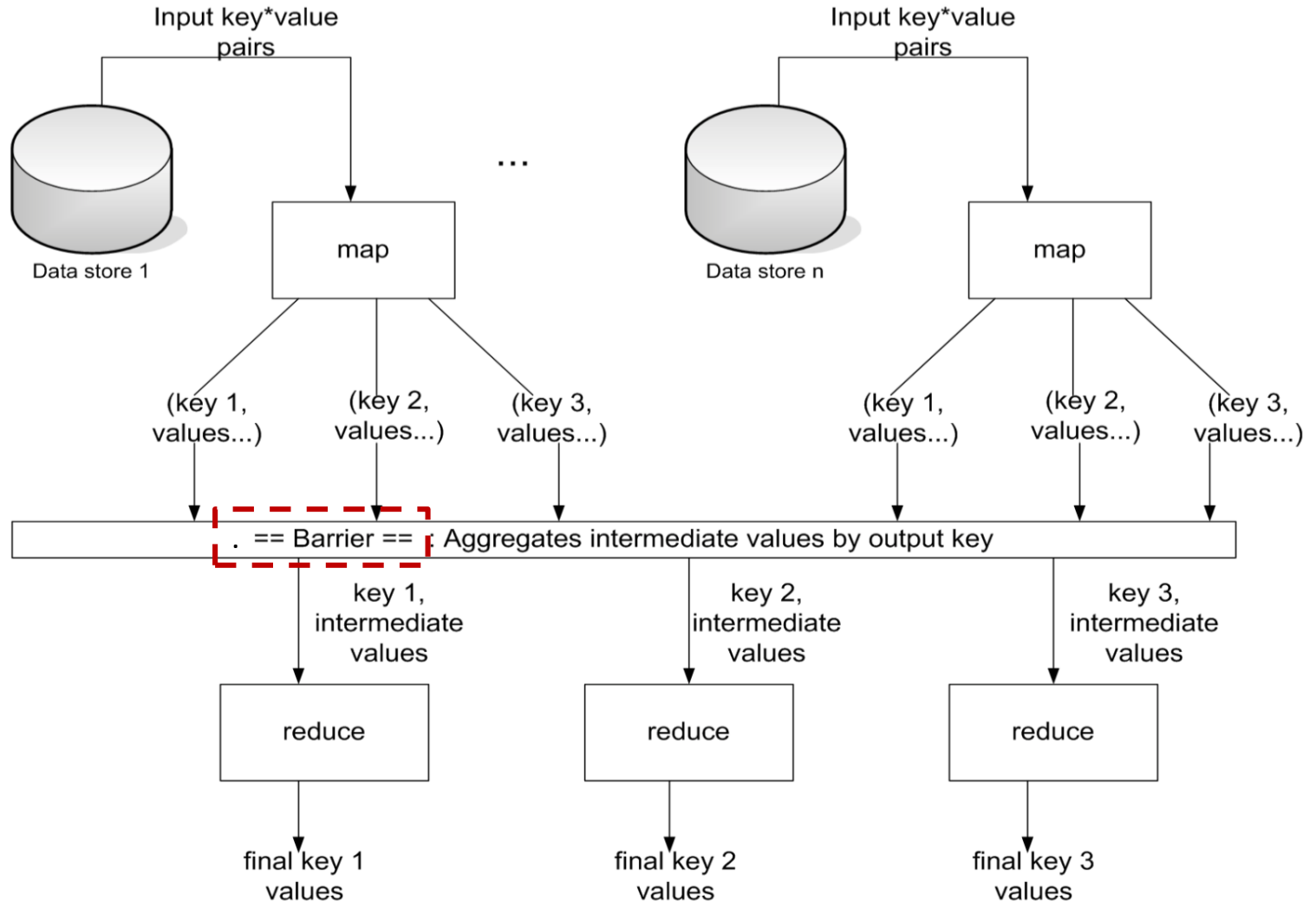
```
map(String key, String value):
// key: document name
// value: document contents
 for each word w in value:
 EmitIntermediate(w, "1");
```

**reduce(k2, list(v2)) → list(v2)**

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

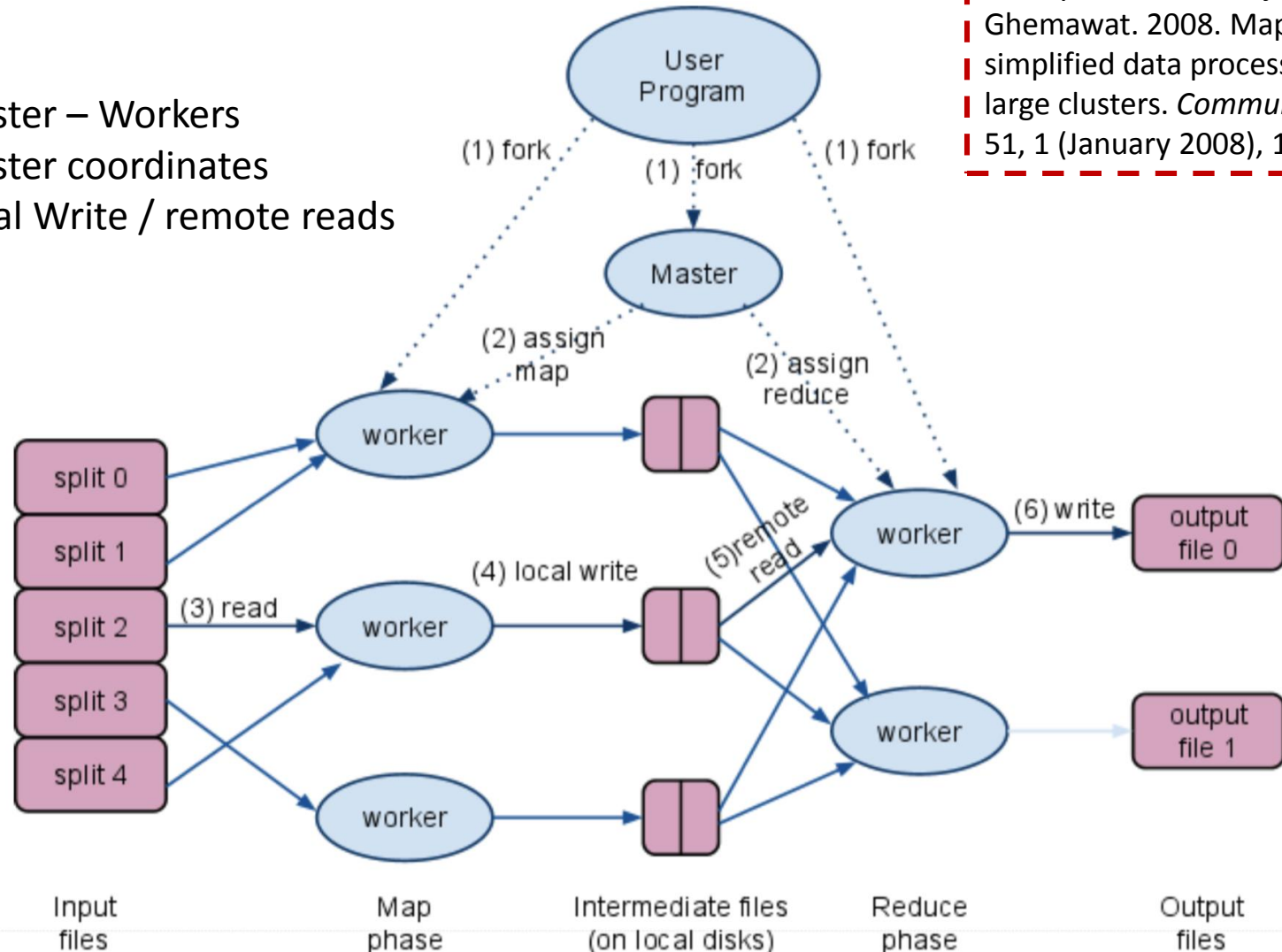


# Parallel processing model



# Execution overview

- Master – Workers
- Master coordinates
- Local Write / remote reads



**Read as part of this lecture!**  
Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (January 2008), 107-113.

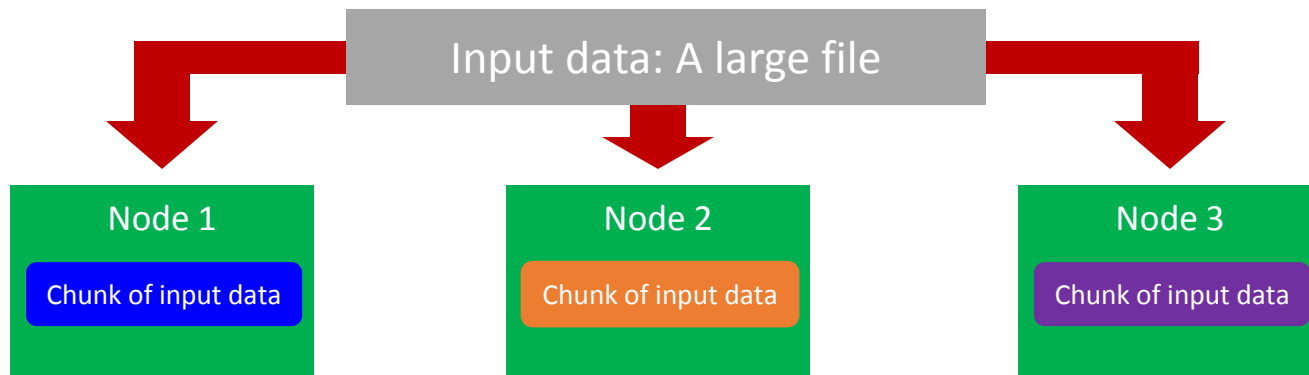


# MapReduce Scheduling

- One master, many workers
  - Input data split into M map tasks (typically 64 MB (~ chunk size in GFS))
  - Reduce phase partitioned into R reduce tasks ( $\text{hash}(k) \bmod R$ )
  - Tasks are assigned to workers dynamically
- Master assigns each map task to a free worker
  - Considers locality of data to worker when assigning a task
  - Worker reads task input (often from local disk)
  - Worker produces R local files containing intermediate k/v pairs
- Master assigns each reduce task to a free worker
  - Worker reads intermediate k/v pairs from map workers
  - Worker sorts & applies user's reduce operation to produce the output

# Data Distribution

- In a MapReduce cluster, data is distributed to all the nodes of the cluster as it is being loaded in
- An underlying distributed file systems (e.g., GFS) splits large data files into chunks which are managed by different nodes in the cluster

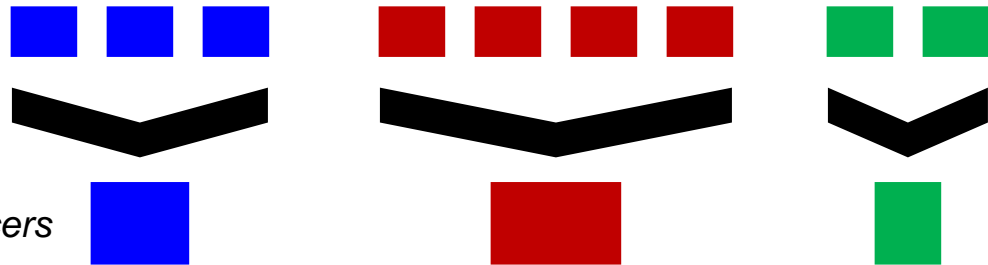


- Even though the file chunks are distributed across several machines, they form a single namespace

# Partitions

- In MapReduce, intermediate output values are not usually reduced together
- All values with the same key are presented to a single Reducer together
- More specifically, a different subset of intermediate key space is assigned to each Reducer
- These subsets are known as partitions

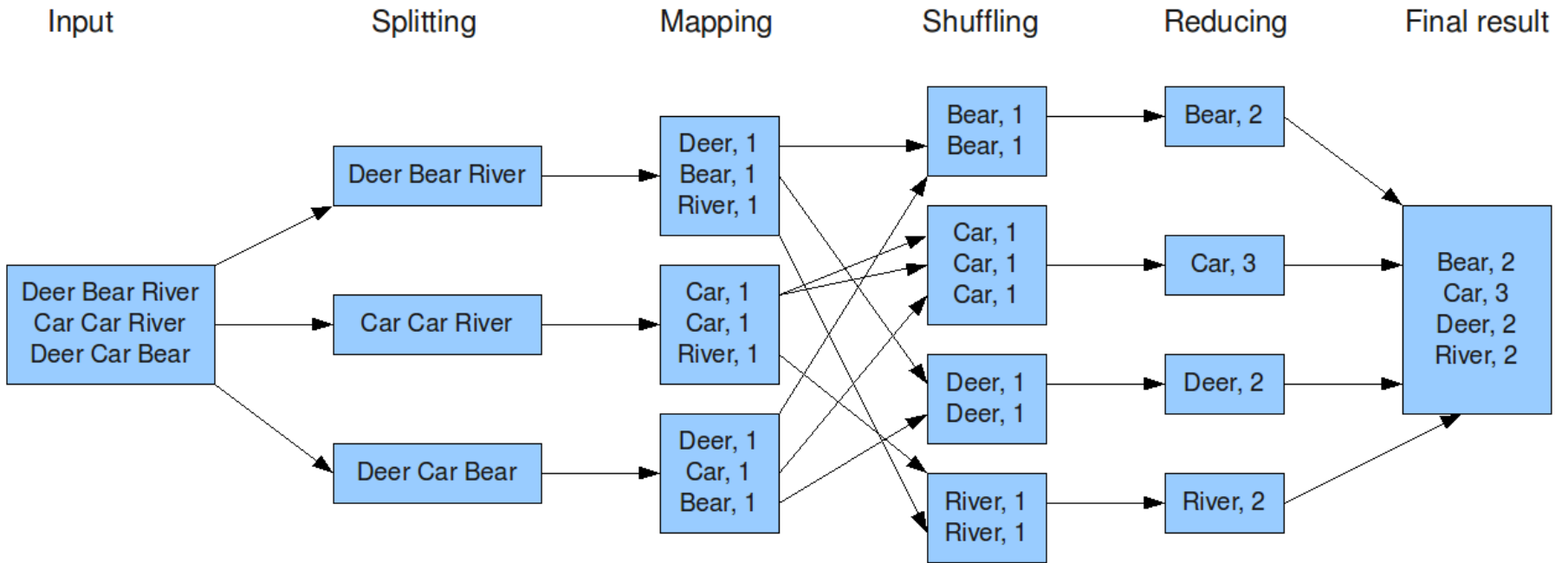
*Different colors represent different keys (potentially) from different Mappers*



*Partitions are the input to Reducers*

# Word count again

The overall MapReduce word count process



# Choosing M and R

- M = number of map tasks, R = number of reduce tasks
- Larger M, R: creates smaller tasks, enabling easier load balancing and faster recovery (many small tasks from failed machine)
- Limitation:  $O(M+R)$  scheduling decisions and  $O(M*R)$  in-memory state at master
  - Very small tasks not worth the startup cost
- Recommendation:
  - Choose M so that split size is approximately 64 MB
  - Choose R a small multiple of the number of workers; alternatively choose R a little smaller than #workers to finish reduce phase in one “wave”

# MapReduce Fault Tolerance

- On worker failure:
  - Master detects failure via periodic **heartbeats**.
  - Both completed and in-progress map tasks on that worker should be re-executed (→ output stored on local disk).
  - Only in-progress reduce tasks on that worker should be re-executed (→ output stored in global file system).
  - All reduce workers will be notified about any map re-executions.
- On master failure:
  - State is check-pointed to GFS: new master recovers & continues.
- Robustness:
  - Example: Lost 1600 of 1800 machines once, but finished fine.

# MapReduce Data Locality

- Goal: To conserve network bandwidth.
- In GFS, data files are divided into 64MB blocks and 3 copies of each are stored on different machines.
- Master program schedules map() tasks based on the location of these replicas:
  - Put map() tasks physically on the same machine as one of the input replicas (or, at least on the same rack / network switch).
- This way, thousands of machines can read input at local disk speed. Otherwise, rack switches would limit read rate.

# Stragglers & Backup Tasks

- Problem: “Stragglers” (i.e., slow workers) significantly lengthen the completion time.
- Solution: Close to completion, spawn backup copies of the remaining in-progress tasks.
  - Whichever one finishes first, “wins”.
- Additional cost: a few percent more resource usage.
- Example: A sort program without backup = 44% longer.



# Other Practical Extensions

- User-specified **combiner functions** for partial combination within a map task can save network bandwidth (~ mini-reduce)
  - Example: WordCount
- User-specified **partitioning functions** for mapping intermediate key values to reduce workers (by default:  $\text{hash}(\text{key}) \bmod R$ )
  - Example:  $\text{hash}(\text{Hostname}(\text{urlkey})) \bmod R$
- **Ordering guarantees**: Processing intermediate k/v pairs in increasing order
  - Example: reduce of WordCount outputs ordered results.
- Custom input and output format handlers
- Single-machine execution option for testing & debugging

# Basic MapReduce Program Design

- Tasks that can be performed independently on a data object, large number of them: Map
- Tasks that require combining of multiple data objects: Reduce
- Sometimes it is easier to start program design with Map, sometimes with Reduce
- Select keys and values such that the right objects end up together in the same Reduce invocation
- Might have to partition a complex task into multiple MapReduce sub-tasks

# MapReduce vs. Traditional RDBMS

|                  | MapReduce                   | Traditional RDBMS         |
|------------------|-----------------------------|---------------------------|
| <b>Data size</b> | Petabytes                   | Gigabytes                 |
| <b>Access</b>    | Batch                       | Interactive and batch     |
| <b>Updates</b>   | Write once, read many times | Read and write many times |
| <b>Structure</b> | Dynamic schema              | Static schema             |
| <b>Integrity</b> | Low                         | High (normalized data)    |
| <b>Scaling</b>   | Linear                      | Non-linear (general SQL)  |

# More Hadoop details

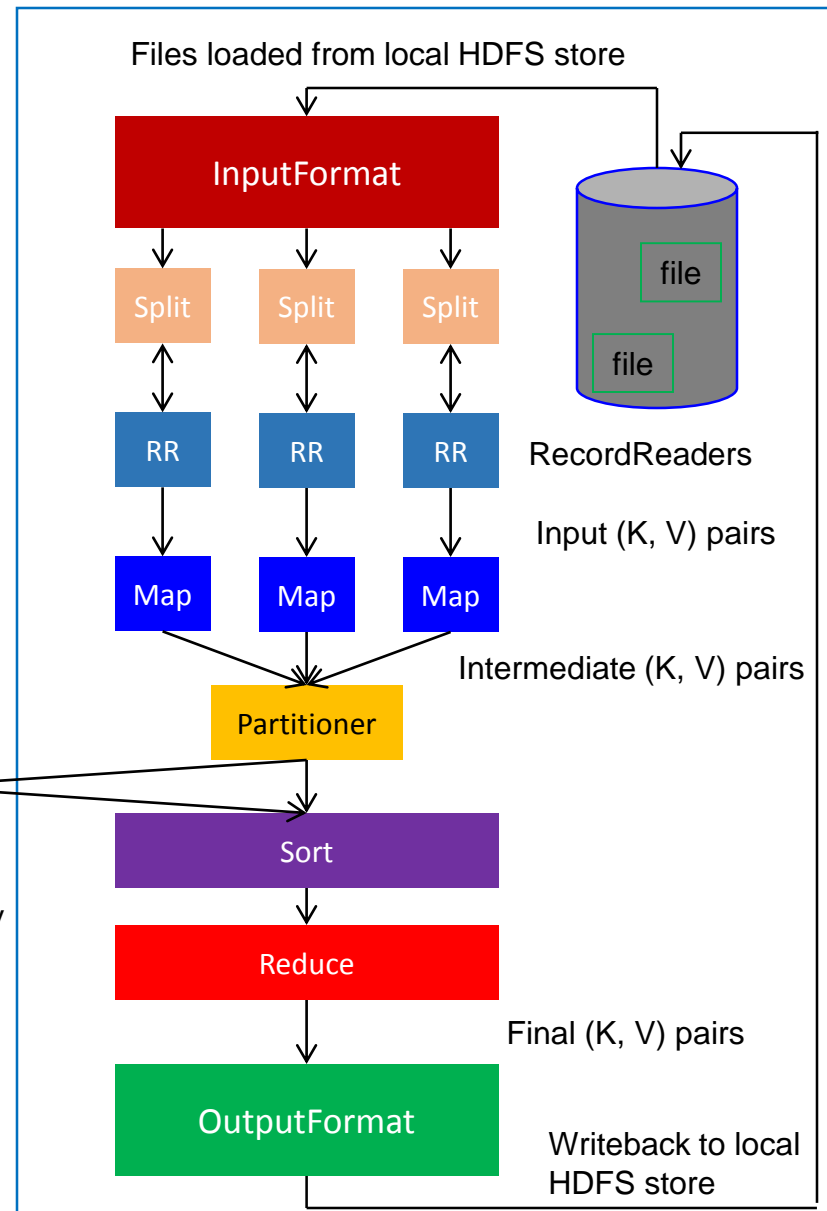
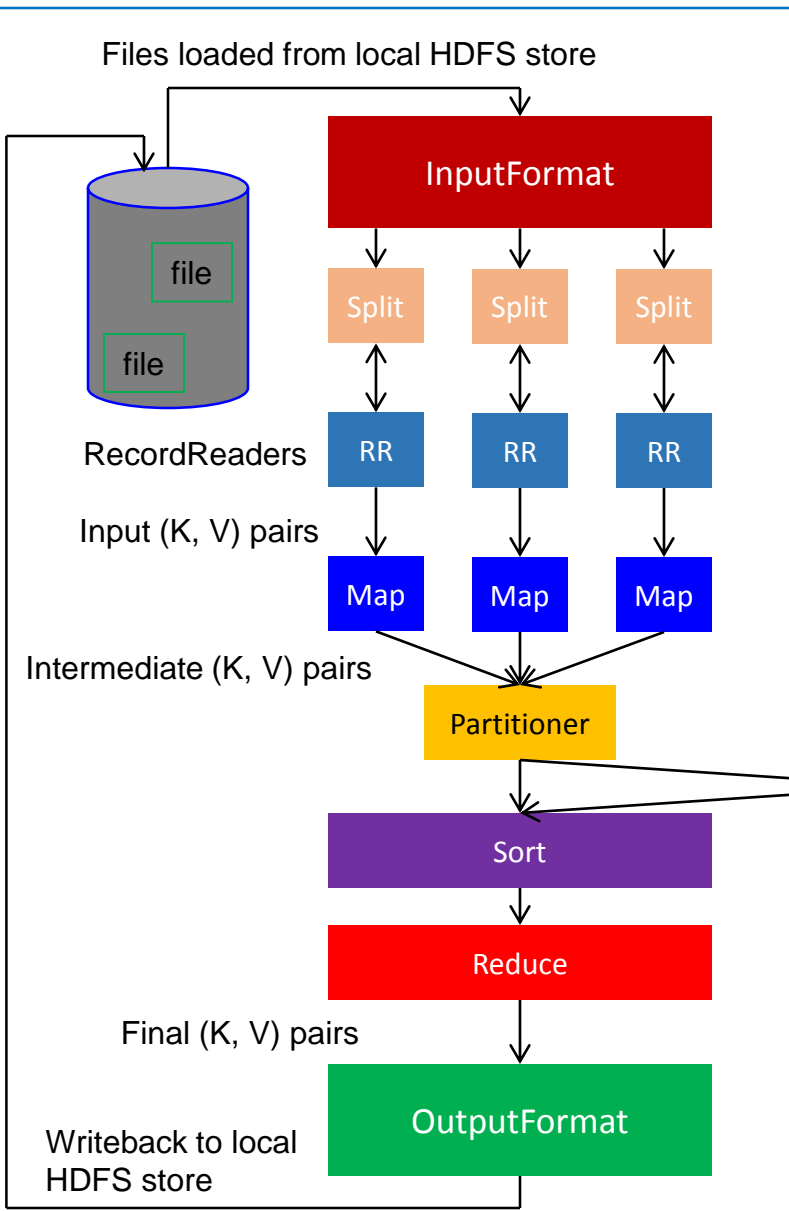
# Hadoop

- Since its debut on the computing stage, MapReduce has frequently been associated with *Hadoop*
- Hadoop is an open source implementation of MapReduce and is currently enjoying wide popularity
- Hadoop presents MapReduce as an analytics engine and under the hood uses a distributed storage layer referred to as Hadoop Distributed File System (*HDFS*)
- HDFS mimics Google File System (*GFS*)

# Hadoop MapReduce: A Closer Look

Node 1

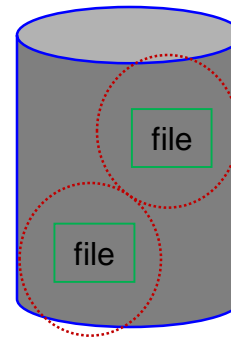
Node 2



**Shuffling Process**  
Intermediate (K,V) pairs exchanged by all nodes

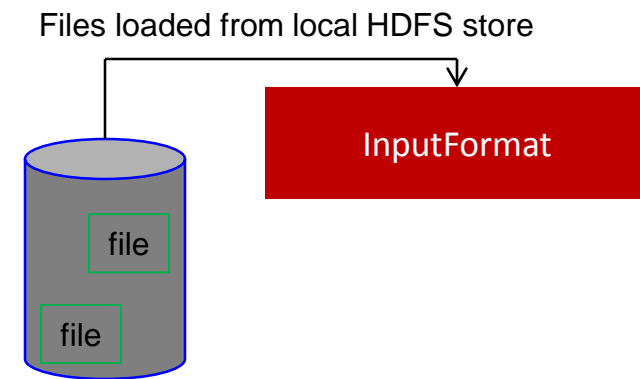
# Input Files

- Input files are where the data for a MapReduce task is initially stored
- The input files typically reside in a distributed file system (e.g. HDFS)
- The format of input files is arbitrary
  - Line-based log files
  - Binary files
  - Multi-line input records
  - Or something else entirely



# InputFormat

- How the input files are split up and read is defined by the InputFormat
- InputFormat is a class that does the following:
  - Selects the files that should be used for input
  - Defines the InputSplits that break a file
  - Provides a factory for RecordReader objects that read the file





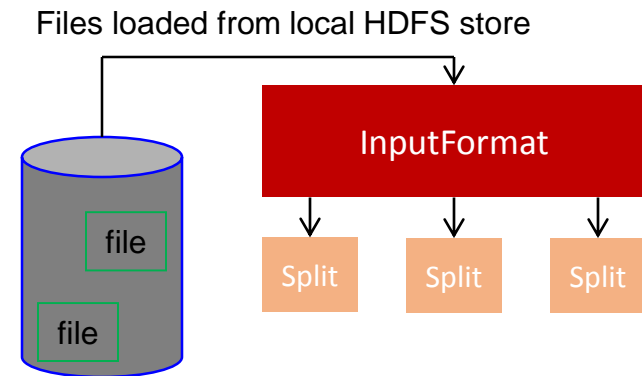
# InputFormat Types

Several InputFormats are provided with Hadoop:

| InputFormat             | Description                                      | Key                                      | Value                     |
|-------------------------|--------------------------------------------------|------------------------------------------|---------------------------|
| TextInputFormat         | Default format; reads lines of text files        | The byte offset of the line              | The line contents         |
| KeyValueInputFormat     | Parses lines into (K, V) pairs                   | Everything up to the first tab character | The remainder of the line |
| SequenceFileInputFormat | A Hadoop-specific high-performance binary format | user-defined                             | user-defined              |

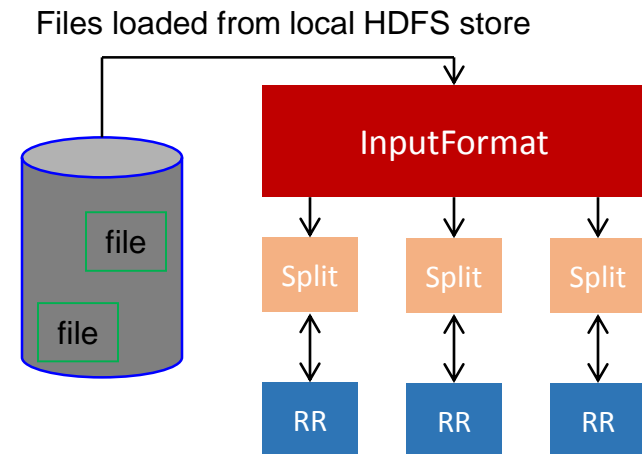
# Input Splits

- An input split describes a unit of work that comprises a single map task in a MapReduce program
- By default, the InputFormat breaks a file up into 64MB splits
- By dividing the file into splits, we allow several map tasks to operate on a single file in parallel
- If the file is very large, this can improve performance significantly through parallelism
- Each map task corresponds to a single input split



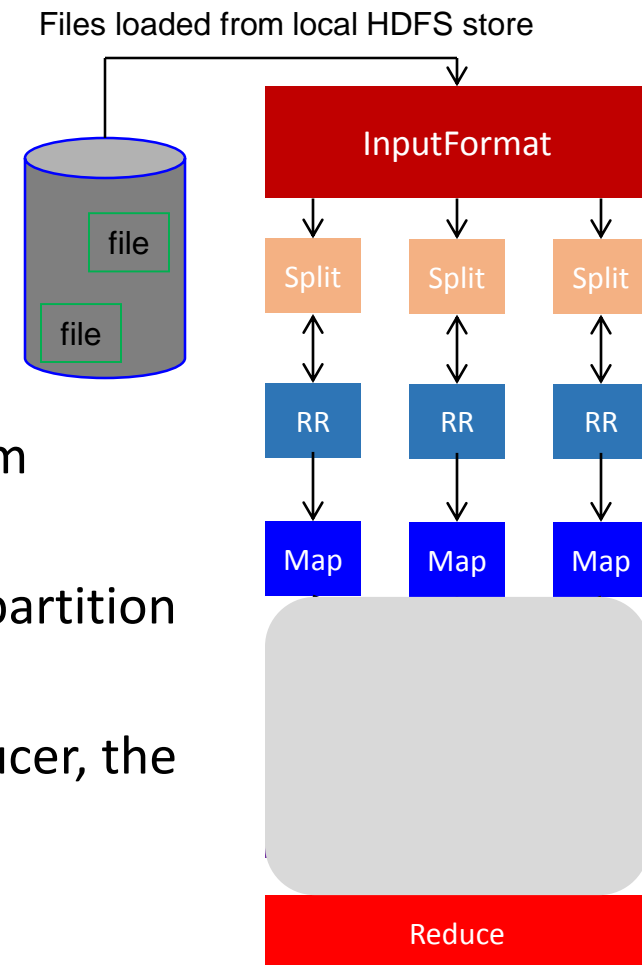
# RecordReader

- The input split defines a slice of work but does not describe how to access it
- The RecordReader class actually loads data from its source and converts it into (K, V) pairs suitable for reading by Mappers
- The RecordReader is invoked repeatedly on the input until the entire split is consumed
- Each invocation of the RecordReader leads to another call of the map function defined by the programmer



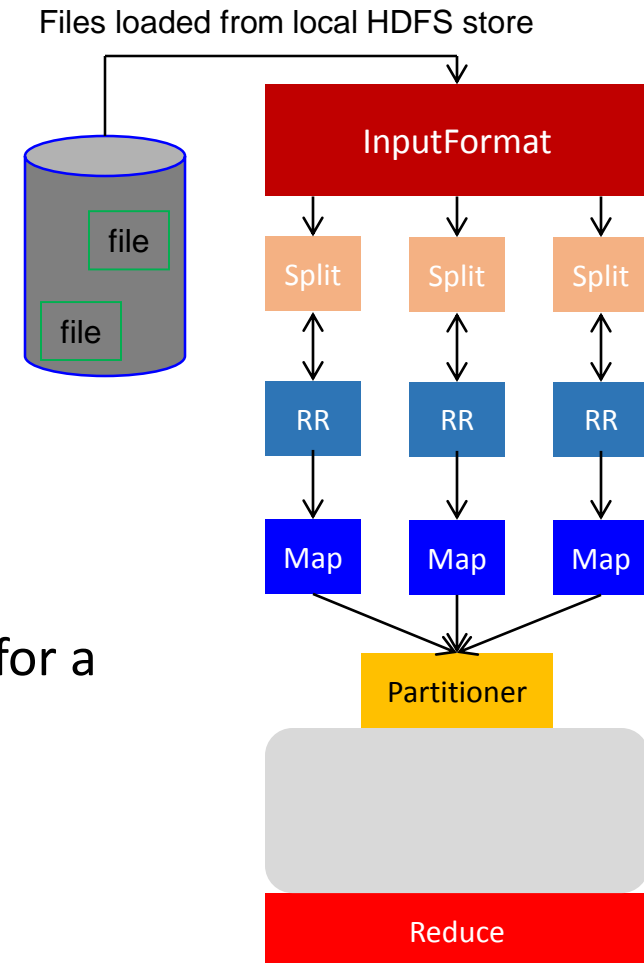
# Mapper and Reducer

- The Mapper performs the user-defined work of the first phase of the MapReduce program
- A new instance of Mapper is created for each split
- The Reducer performs the user-defined work of the second phase of the MapReduce program
- A new instance of Reducer is created for each partition
- For each key in the partition assigned to a Reducer, the Reducer is called once



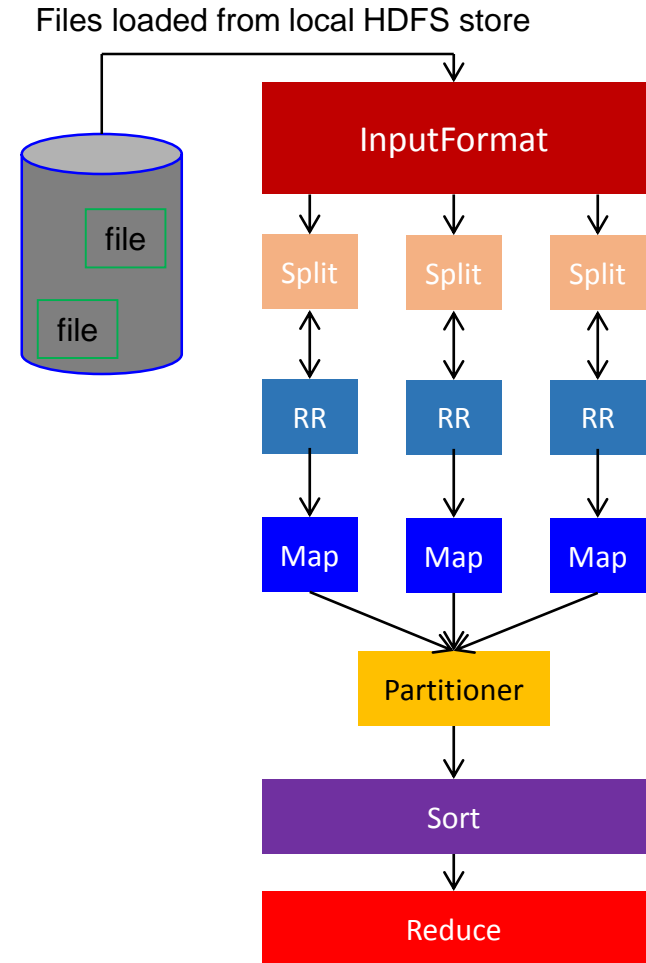
# Partitioner

- Each mapper may emit (K, V) pairs to any partition
- Therefore, the map nodes must all agree on where to send different pieces of intermediate data
- The partitioner class determines which partition a given (K,V) pair will go to
- The default partitioner computes a hash value for a given key and assigns it to a partition based on this result



# Sort

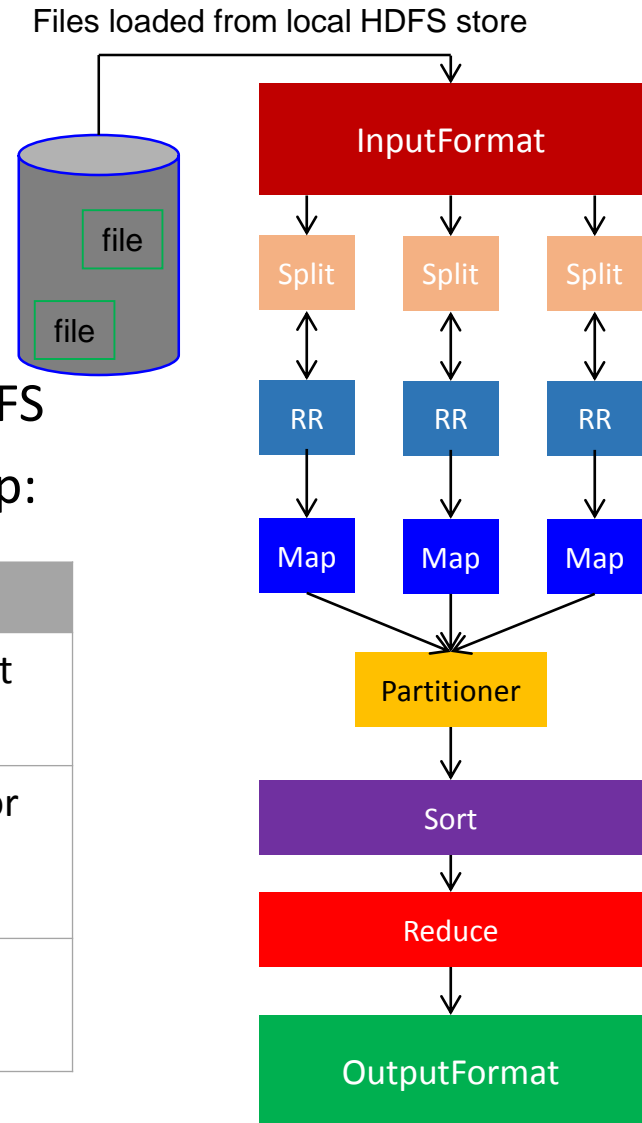
- Each Reducer is responsible for reducing the values associated with (several) intermediate keys
- The set of intermediate keys on a single node is **automatically sorted** by MapReduce before they are presented to the Reducer



# OutputFormat

- The OutputFormat class defines the way (K,V) pairs produced by Reducers are written to output files
- The instances of OutputFormat provided by Hadoop write to files on the local disk or in HDFS
- Several OutputFormats are provided by Hadoop:

| OutputFormat             | Description                                                             |
|--------------------------|-------------------------------------------------------------------------|
| TextOutputFormat         | Default; writes lines in "key \t value" format                          |
| SequenceFileOutputFormat | Writes binary files suitable for reading into subsequent MapReduce jobs |
| NullOutputFormat         | Generates no output files                                               |



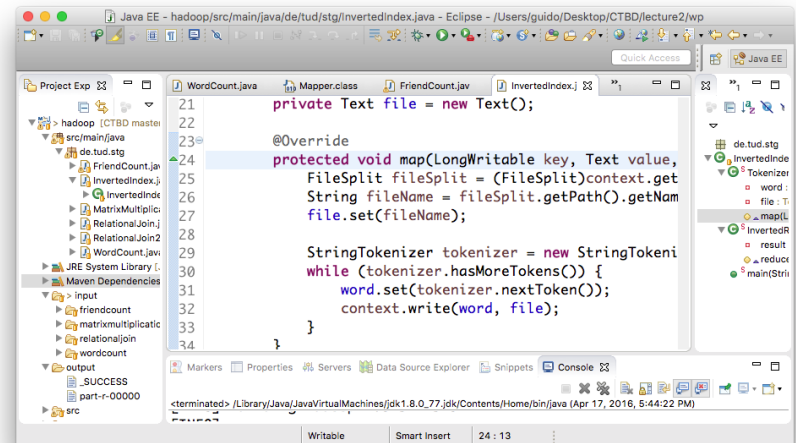
Questions?



# Exercise

# Exercise

- Read the original Map Reduce paper
  - Answer some questions
- Implement “friends count”
- Fill “word length” (why fill, anyway?)
- Understand and run “inverted indexes”
- Code available as a Maven or Eclipse project: Just run locally



The screenshot shows the Eclipse IDE with a Java project named 'hadoop' and a package 'de.tud.stg'. The main class is 'InvertedIndex.java'. The code is as follows:

```
21 private Text file = new Text();
22
23 @Override
24 protected void map(LongWritable key, Text value,
25 FileSplit fileSplit = (FileSplit)context.get
26 String fileName = fileSplit.getPath().getNam
27 file.set(fileName);
28
29 StringTokenizer tokenizer = new StringTokeni
30 while (tokenizer.hasMoreTokens()) {
31 word.set(tokenizer.nextToken());
32 context.write(word, file);
33 }
34 }
```

# MapReduce Use Case: Word Length

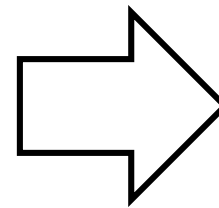
## Abridged Declaration of Independence

A Declaration By the Representatives of the United States of America, in General Congress Assembled.

When in the course of human events it becomes necessary for a people to advance from that subordination in which they have hitherto remained, and to assume among powers of the earth the equal and independent station to which the laws of nature and of nature's god entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the change.

We hold these truths to be self-evident; that all men are created equal and independent; that from that equal creation they derive rights inherent and inalienable, among which are the preservation of life, and liberty, and the pursuit of happiness; that to secure these ends, governments are instituted among men, deriving their just power from the consent of the governed; that whenever any form of government shall become destructive of these ends, it is the right of the people to alter or to abolish it, and to institute new government, laying it's foundation on such principles and organizing it's power in such form, as to them shall seem most likely to effect their safety and happiness. Prudence indeed will dictate that governments long established should not be changed for light and transient causes: and accordingly all experience hath shewn that mankind are more disposed to suffer while evils are sufferable, than to right themselves by abolishing the forms to which they are accustomed. But when a long train of abuses and usurpations, begun at a distinguished period, and pursuing invariably the same object, evinces a design to reduce them to arbitrary power, it is their right, it is their duty, to throw off such government and to provide new guards for future security. Such has been the patient sufferings of the colonies; and such is now the necessity which constrains them to expunge their former systems of government. the history of his present majesty is a history of unremitting injuries and usurpations, among which no one fact stands single or solitary to contradict the uniform tenor of the rest, all of which have in direct object the establishment of an absolute tyranny over these states. To prove this, let facts be submitted to a candid world, for the truth of which we pledge a faith yet unsullied by falsehood.

Big = Yellow = 10+ letters  
Medium = Red = 5..9 letters  
Small = Blue = 2..4 letters  
Tiny = Pink = 1 letter



Big 37  
Medium 148  
Small 200  
Tiny 9

# MapReduce Use Case: Word Length

Split the document into  
chunks and process  
each chunk  
on a different computer

## Abridged Declaration of Independence

A Declaration By the Representatives of the United States of America, in General Congress Assembled.

When in the course of human events it becomes necessary for a people to advance from that subordination in which they have hitherto remained, and to assume among powers of the earth the equal and independent station to which the laws of nature and of nature's god entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the change.

We hold these truths to be self-evident; that all men are created equal and independent; that from that equal creation they derive rights inherent and inalienable, among which are the preservation of life, and liberty, and the pursuit of happiness; that to secure these ends, governments are instituted among men, deriving their just power from the consent of the governed; that whenever any form of government shall become destructive of these ends, it is the right of the people to alter or to abolish it, and to institute new government, laying it's foundation on such principles and organizing it's power in such form, as to them shall seem most likely to effect their safety and happiness. Prudence indeed will

dictate that governments long established should not be changed for light and transient causes: and accordingly all experience hath shewn that mankind are more disposed to suffer while evils are sufferable, than to right themselves by abolishing the forms to which they are accustomed. But when a long train of abuses and usurpations, begun at a distinguished period, and pursuing invariably the same object, evinces a design to reduce them to arbitrary power, it is their right, it is their duty, to throw off such government and to provide new guards for future security. Such has been the patient sufferings of the colonies; and such is now the necessity which constrains them to expunge their former systems of government. the history of his present majesty is a history of unremitting injuries and usurpations, among which no one fact stands single or solitary to contradict the uniform tenor of the rest, all of which have in direct object the establishment of an absolute tyranny over these states. To prove this, let facts be submitted to a candid world, for the truth of which we pledge a faith yet unsullied by falsehood.

# MapReduce Use Case: Word Length

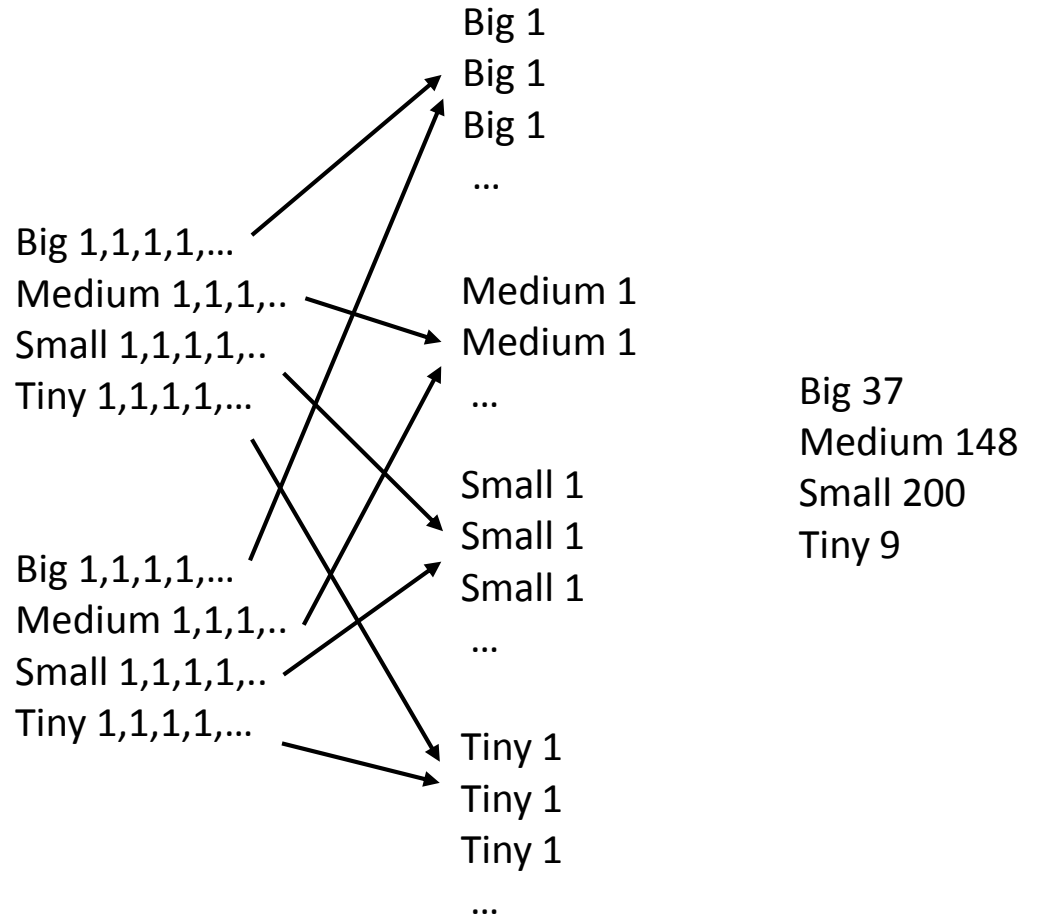
## Abridged Declaration of Independence

A Declaration By the Representatives of the United States of America, in General Congress Assembled.

When in the course of human events it becomes necessary for a people to advance from that subordination in which they have hitherto remained, and to assume among powers of the earth the equal and independent station to which the laws of nature and of nature's god entitle them, a decent respect to the opinions of mankind requires that they should declare the causes which impel them to the change.

We hold these truths to be self-evident; that all men are created equal and independent; that from that equal creation they derive rights inherent and inalienable, among which are the preservation of life, and liberty, and the pursuit of happiness; that to secure these ends, governments are instituted among men, deriving their just power from the consent of the governed; that whenever any form of government shall become destructive of these ends, it is the right of the people to alter or to abolish it, and to institute new government, laying it's foundation on such principles and organizing it's power in such form, as to them shall seem most likely to effect their safety and happiness. Prudence indeed will

dictate that governments long established should not be changed for light and transient causes: and accordingly all experience hath shewn that mankind are more disposed to suffer while evils are sufferable, than to right themselves by abolishing the forms to which they are accustomed. But when a long train of abuses and usurpations, begun at a distinguished period, and pursuing invariably the same object, evinces a design to reduce them to arbitrary power, it is their right, it is their duty, to throw off such government and to provide new guards for future security. Such has been the patient sufferings of the colonies; and such is now the necessity which constrains them to expunge their former systems of government. the history of his present majesty is a history of unremitting injuries and usurpations, among which no one fact stands single or solitary to contradict the uniform tenor of the rest, all of which have in direct object the establishment of an absolute tyranny over these states. To prove this, let facts be submitted to a candid world, for the truth of which we pledge a faith yet unsullied by falsehood.

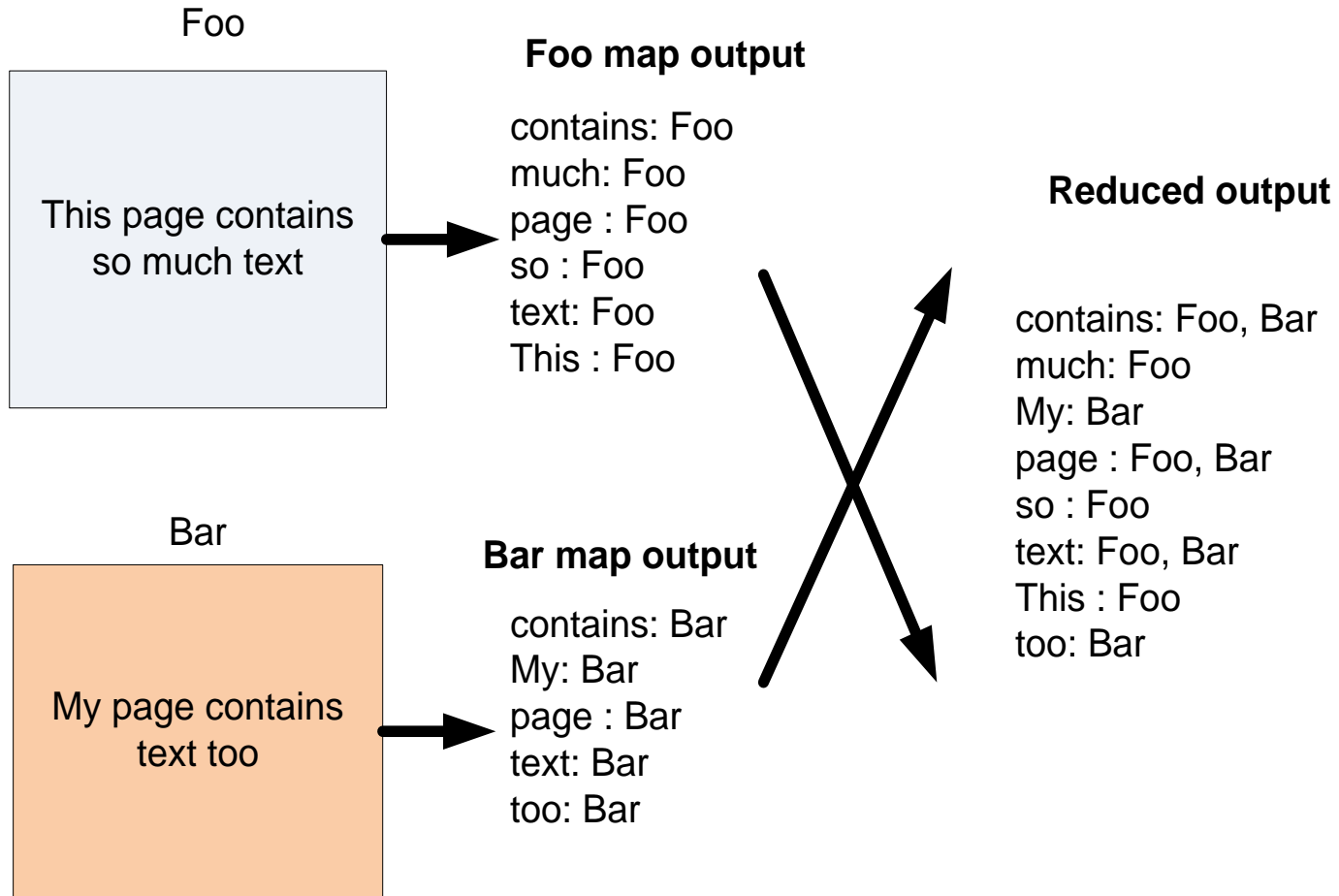


# MapReduce Use Case: Inverted Indexing

Construction of inverted lists for document search

- Input: documents: (docid, [term, term..]), (docid, [term, ..]), ..
- Output: (term, [docid, docid, ...])
  - E.g., (apple, [Foo.txt, Bar.txt, Boo.txt, ...])

# Inverted Index: Data flow





# MapReduce Use Case: Inverted Indexing

A simple approach to creating inverted lists

- Each Map task is a document parser
  - Input: A stream of documents
  - Output: A stream of (term, docid) tuples
    - (long, Foo.txt) (ago, Foo.txt) (and, Foo.txt) ... (once, Bar.txt) (upon, Bar.txt) ...
    - We may create internal IDs for words.
- Shuffle sorts tuples by key and routes tuples to Reducers
- Reducers convert streams of keys into streams of inverted lists
  - Input: (long, Foo.txt) (long, Bar.txt) (long, Boo.txt) (long, ...) ...
  - The reducer sorts the values for a key and builds an inverted list
  - Output: (long, [Foo.txt, Bar.txt, ...])



Questions?

# Sources & References

Excellent intro to MapReduce:

- [https://websci.informatik.uni-freiburg.de/teaching/ws201213/infosys/slides/m3\\_l1\\_mapreduce.pdf](https://websci.informatik.uni-freiburg.de/teaching/ws201213/infosys/slides/m3_l1_mapreduce.pdf)
- [http://www.systems.ethz.ch/sites/default/files/file/BigData\\_Fall2012/BigData-2012-M3.pdf](http://www.systems.ethz.ch/sites/default/files/file/BigData_Fall2012/BigData-2012-M3.pdf)

MapReduce & Functional Programming:

- <https://courses.cs.washington.edu/courses/cse490h/08au/lectures/mapred.ppt>

For the introductory part:

- <http://www.cs.ucsb.edu/~tyang/class/140s14/slides/CS140TopicMapReduce.pdf>

A lot of details about the Hadoop case:

- [www.qatar.cmu.edu/~msakr/15440-f11/.../Lecture18\\_15440\\_MHH\\_9Nov\\_2011.ppt](http://www.qatar.cmu.edu/~msakr/15440-f11/.../Lecture18_15440_MHH_9Nov_2011.ppt)