

# STATS 701

# Data Analysis using Python

## Lecture 19: MapReduce

Some slides adapted from C. Budak and R. Burns

# Parallel processing and “big data”

The next few lectures will focus on “big data” and the MapReduce framework

**Today:** overview of the MapReduce framework

**Next lectures:**

Python package `mrjob`, which implements MapReduce

Apache Spark and the Hadoop file system

# The big data “revolution”

Sloan Digital Sky Survey <https://www.sdss.org/>

Generating so many images that most will never be looked at...

Genomics data: [https://en.wikipedia.org/wiki/Genome\\_project](https://en.wikipedia.org/wiki/Genome_project)

Web crawls

>20e9 webpages; ~400TB just to store pages (*without* images, etc)

Social media data

Twitter: ~500e6 tweets per day

YouTube: >300 hours of content uploaded per minute

(and that number is several years old, now)

# Three aspects to big data

**Volume:** data at the TB or PB scale

Requires new processing paradigms

e.g., Distributed computing, streaming model

**Velocity:** data is generated at unprecedented rate

e.g., web traffic data, twitter, climate/weather data

**Variety:** data comes in many different formats

Databases, but also unstructured text, audio, video...

Messy data requires different tools

This requires a very different approach to computing from what we were accustomed to prior to about 2005.

# How to count all the books in the library?



Peabody Library, Baltimore, MD USA

# How to count all the books in the library?

I'll count this side...



...you count this side...

...and then we add our counts together.

Peabody Library, Baltimore, MD USA



# Congratulations!



You now understand the MapReduce framework!

## **Basic idea:**

- Split up a task into independent subtasks

- Specify how to combine results of subtasks into your answer

**Independent** subtasks is a crucial point, here:

- If we constantly have to share information, then it's inefficient to split the task

- Because we'll spend more time communicating than actually counting

# MapReduce: the workhorse of “big data”

Hadoop, Google MapReduce, Spark, etc are all based on this framework

- 1) Specify a “map” operation to be applied to every element in a data set
- 2) Specify a “reduce” operation for combining the list into an output

Then we split the data among a bunch of machines, and combine their results



# MapReduce isn't really new to you

You already know the **Map** pattern:

```
Python: [f(x) for x in mylist]
```

...and the **Reduce** pattern:

```
Python: sum( [f(x) for x in mylist] ) (map and reduce)
```

```
SQL: aggregation functions are like “reduce” operations
```

The only thing that's new is the computing model

# MapReduce, schematically, cartoonishly

Map:  $f(x) = 2x$



Reduce: sum



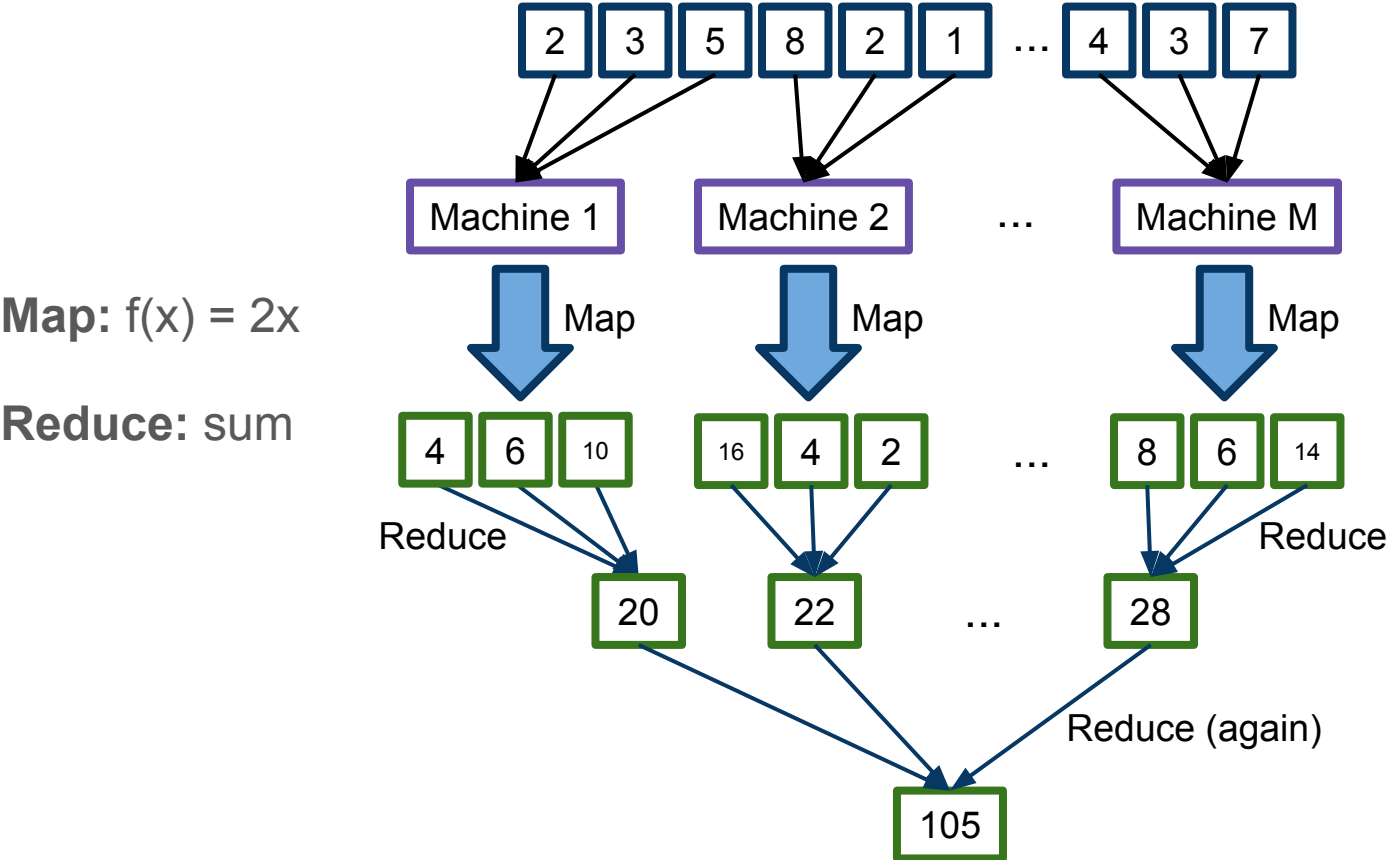
...but this hides the distributed computation.

# Assumptions of MapReduce

- Task can be split into pieces
- Pieces can be processed **in parallel**...
- ...with **minimal communication** between processes.
- Results of each piece can be combined to obtain answer.

Problems that have these properties are often described as being **embarrassingly parallel**: [https://en.wikipedia.org/wiki/Embarrassingly\\_parallel](https://en.wikipedia.org/wiki/Embarrassingly_parallel)

# MapReduce, schematically (slightly more accurately)



# Less boring example: word counts



Suppose we have a giant collection of books...

e.g., Google ngrams: <https://books.google.com/ngrams/info>

...and we want to count how many times each word appears in the collection.

## Divide and Conquer!

1. Everyone takes a book, and makes a list of (word,count) pairs.
2. Combine the lists, adding the counts with the same **word** keys.

This still fits our framework, but it's a little more complicated...

...and it's just the kind of problem that MapReduce is designed to solve.

# Fundamental unit of MapReduce: (key,value) pairs

## Examples:

Linguistic data: <word, count>

Enrollment data: <student, major>

Climate data: <location, wind speed>

Values can be more complicated objects in some environments

e.g., lists, dictionaries, other data structures

Social media data: <person, list\_of\_friends>

Apache Hadoop doesn't support this directly

but can be made to work via some hacking

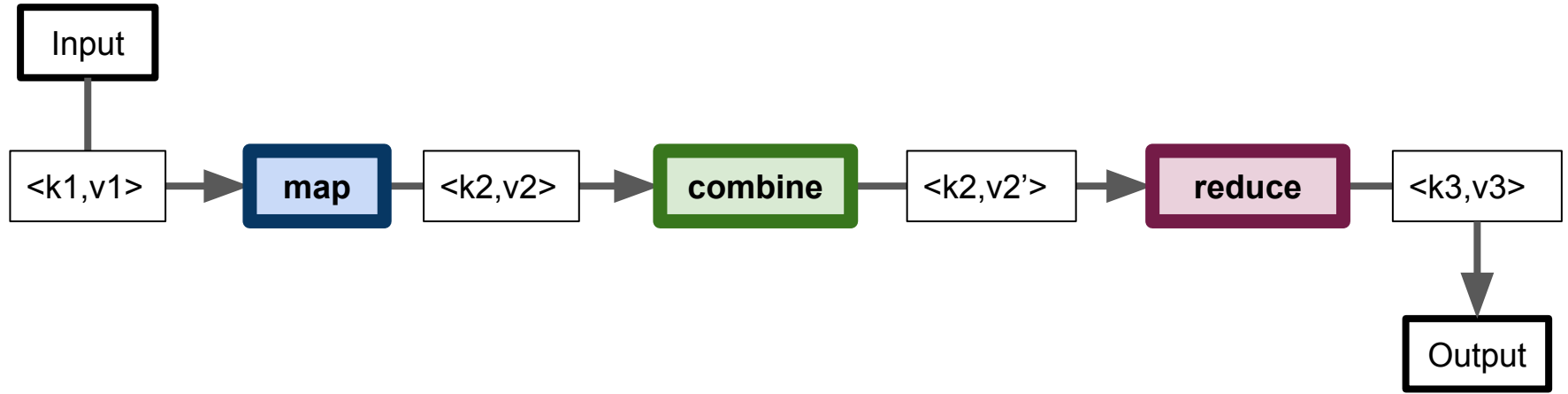
mrjob and Spark are a little more flexible

# A prototypical MapReduce program

1. Read records (i.e., pieces of data) from file(s)
2. **Map:**  
For each record, extract information you care about  
Output this information in <key,value> pairs
3. **Combine:**  
Sort and group the extracted <key,value> pairs based on their keys
4. **Reduce:**  
For each group, summarize, filter, group, aggregate, etc. to obtain some new value, v2  
Output the <key, v2> pair as a row in the results file



# A prototypical MapReduce program



**Note:** this output could be made the input to another MR program. We call one of these input->map->combine->reduce->output chains a **step**. Hadoop/mrjob differs from Spark in how these steps are executed, a topic we'll discuss in our next two lectures.

# Clarifying terminology

**MapReduce:** a large-scale computing framework initially developed at Google

Later open-sourced via the Apache Foundation as **Hadoop MapReduce**

**Apache Hadoop:** a set of open source tools from the Apache Foundation

Includes Hadoop MapReduce, Hadoop HDFS, Hadoop YARN

**Hadoop MapReduce:** implements MapReduce framework

**Hadoop YARN:** resource manager that schedules Hadoop MapReduce jobs

**Hadoop Distributed File System (HDFS):** distributed file system

Designed for use with Hadoop MapReduce

Runs on same commodity hardware that MapReduce runs on

Note that there are a host of other loosely related programs, such as Apache Hive, Pig, Mahout and HBase, most of which are designed to work atop HDFS.

# MapReduce: vocabulary

**Cluster:** a collection of devices (i.e., computers)

Networked to enable fast communication, typically for purpose of distributed computing

Jobs scheduled by a program like Sun/Oracle grid engine, Slurm, TORQUE or YARN

[https://en.wikipedia.org/wiki/Job\\_scheduler](https://en.wikipedia.org/wiki/Job_scheduler)

**Node:** a single computing “unit” on a cluster

Roughly, computer==node, but can have multiple nodes per machine

Usually a piece of commodity (i.e., not specialized, inexpensive) hardware

**Step:** a single map->combine->reduce “chain”

A step need not contain all three of map, combine and reduce

**Note:** some documentation refers to each of map, combine and reduce as steps

**Job:** a sequence of one or more MapReduce steps

# More terminology (useful for reading documentation)

**NUMA:** non-uniform memory access

Local memory is much faster to access than memory elsewhere on network

[https://en.wikipedia.org/wiki/Non-uniform\\_memory\\_access](https://en.wikipedia.org/wiki/Non-uniform_memory_access)

**Commodity hardware:** inexpensive, mass-produced computing hardware

As opposed to expensive specialized machines

E.g., servers in a data center

**Hash function:** a function that maps (arbitrary) objects to integers

Used in MapReduce to assign keys to nodes in the reduce step

# So MapReduce makes things much easier

Instead of having to worry about splitting the data, organizing communication between machines, etc., we only need to specify:

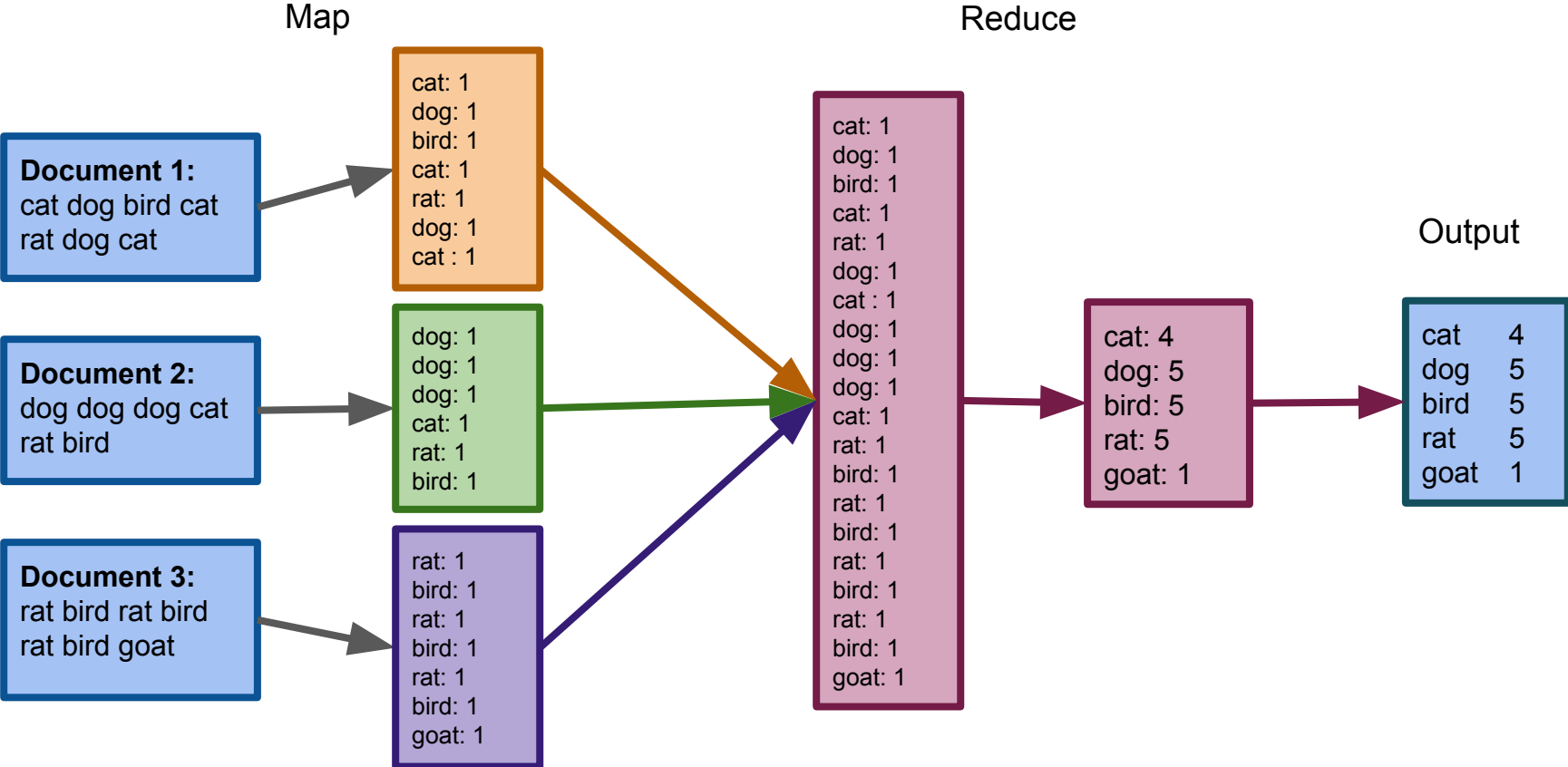
**Map**

**Combine** (optional)

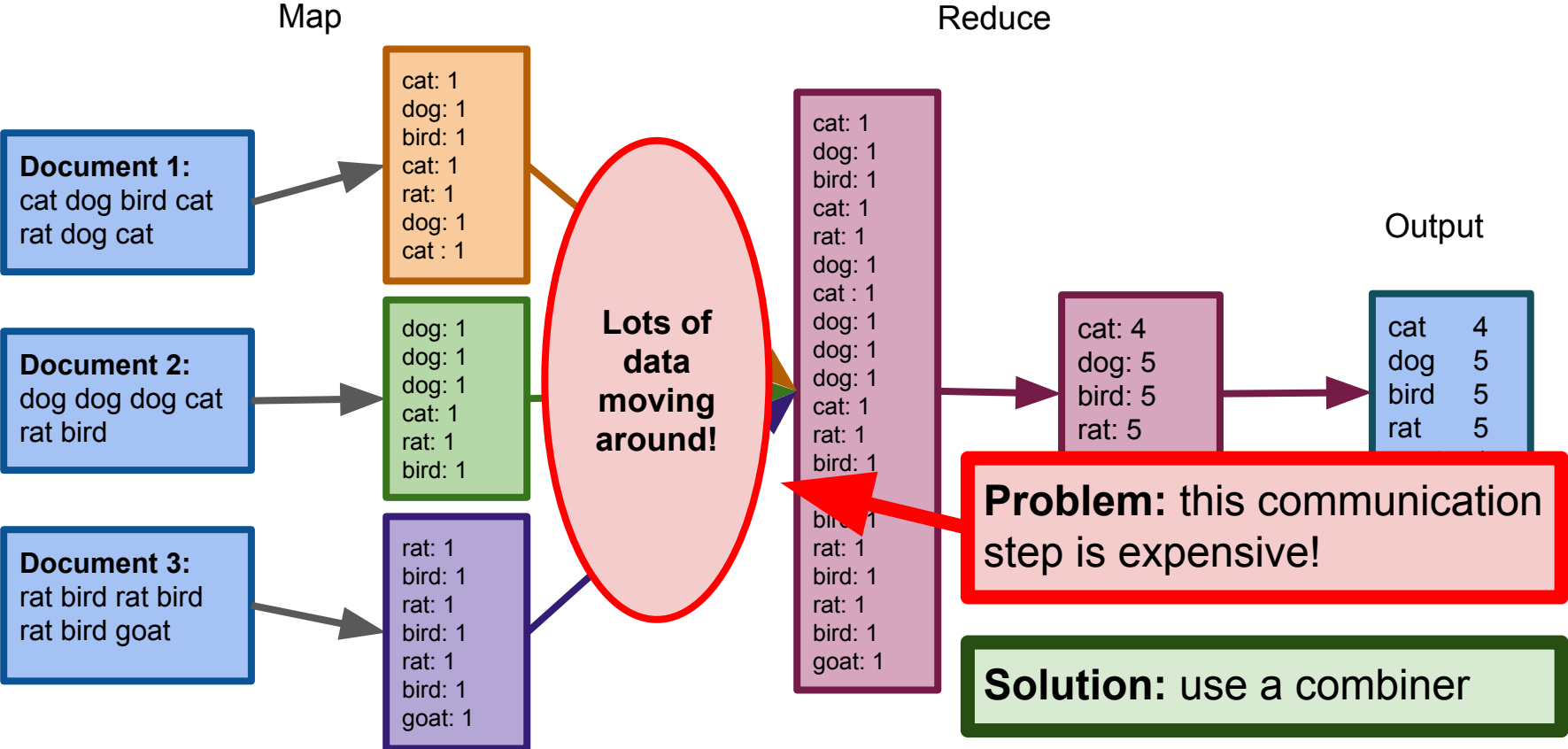
**Reduce**

and the Hadoop backend will handle everything else.

# Counting words in MapReduce: version 1

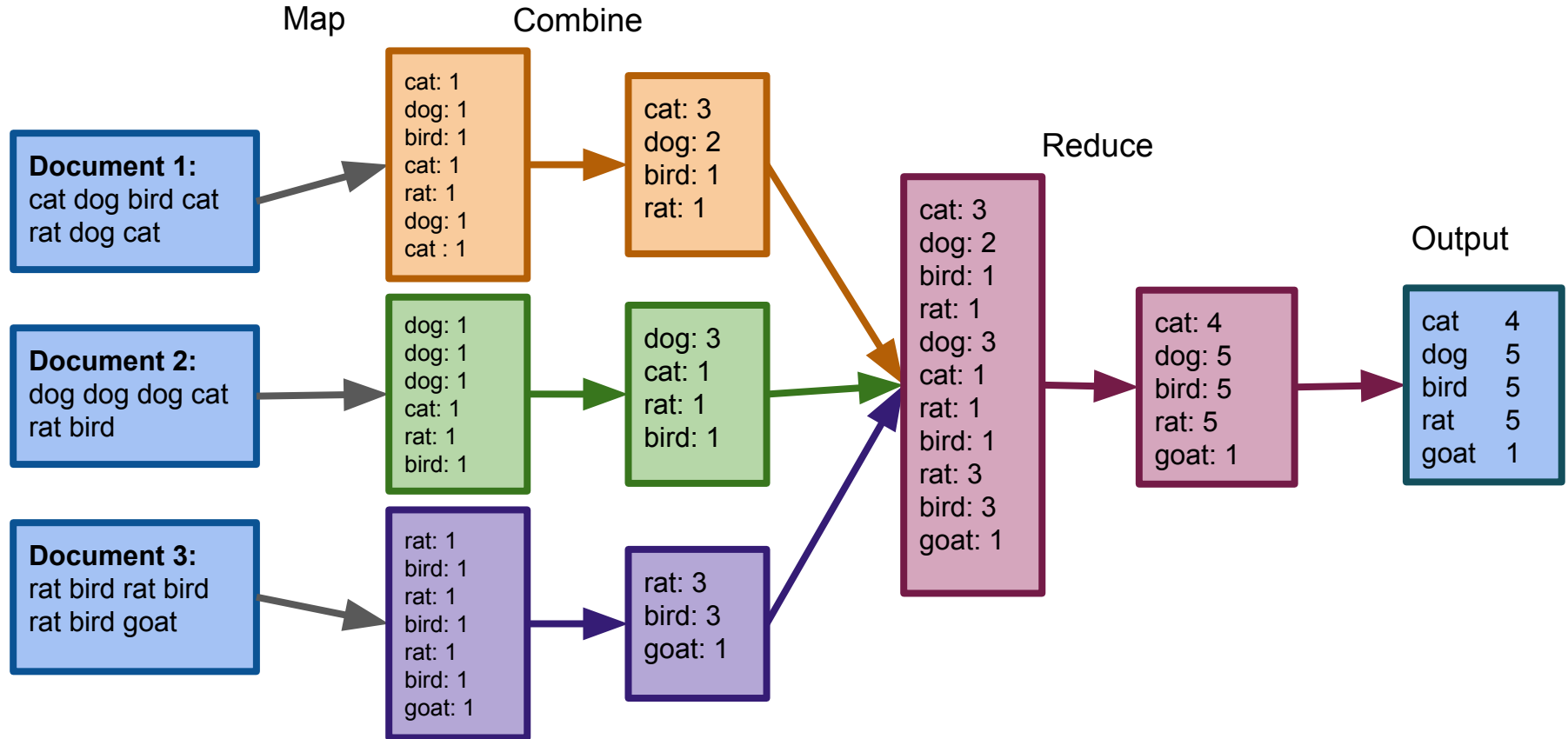


# Counting words in MapReduce: version 1

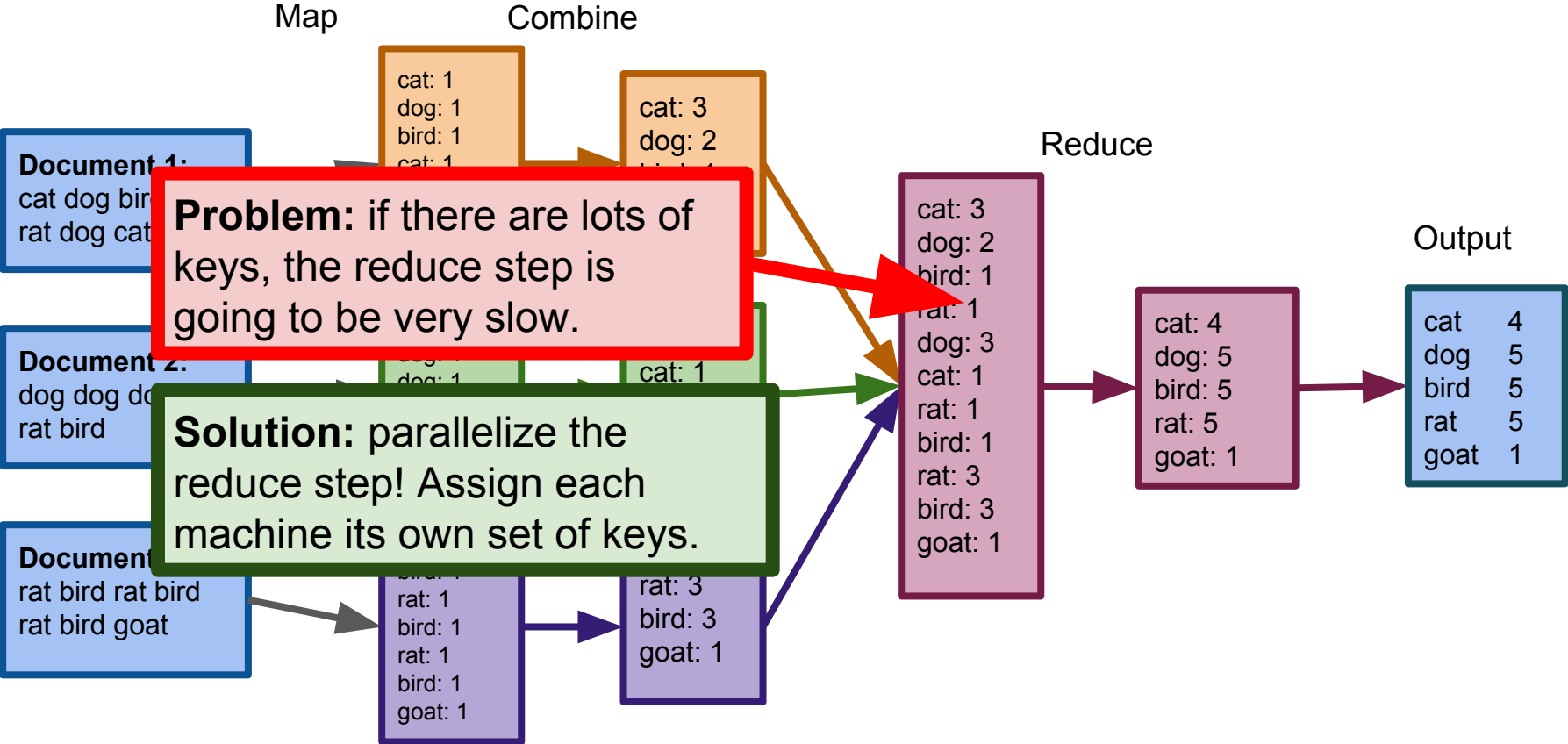




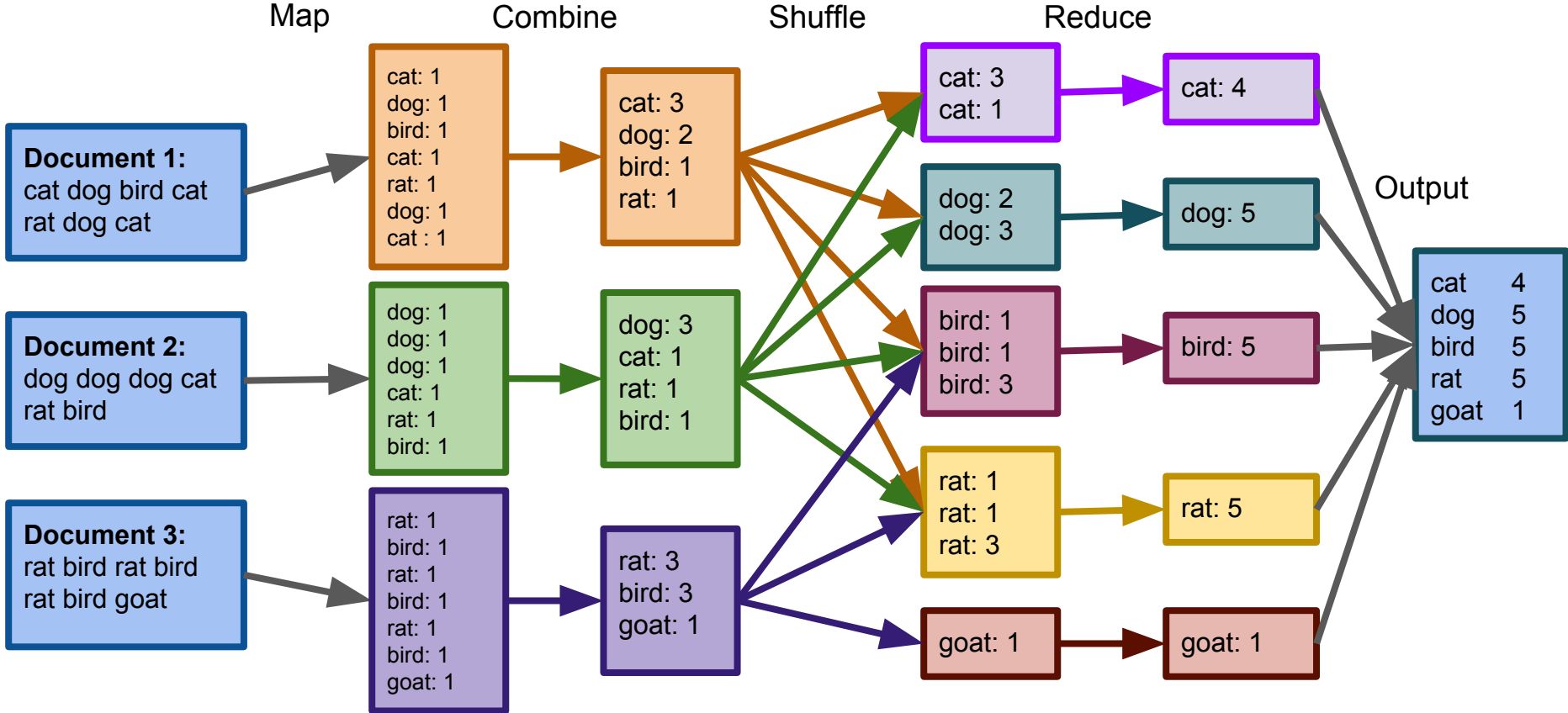
# Counting words in MapReduce: version 2



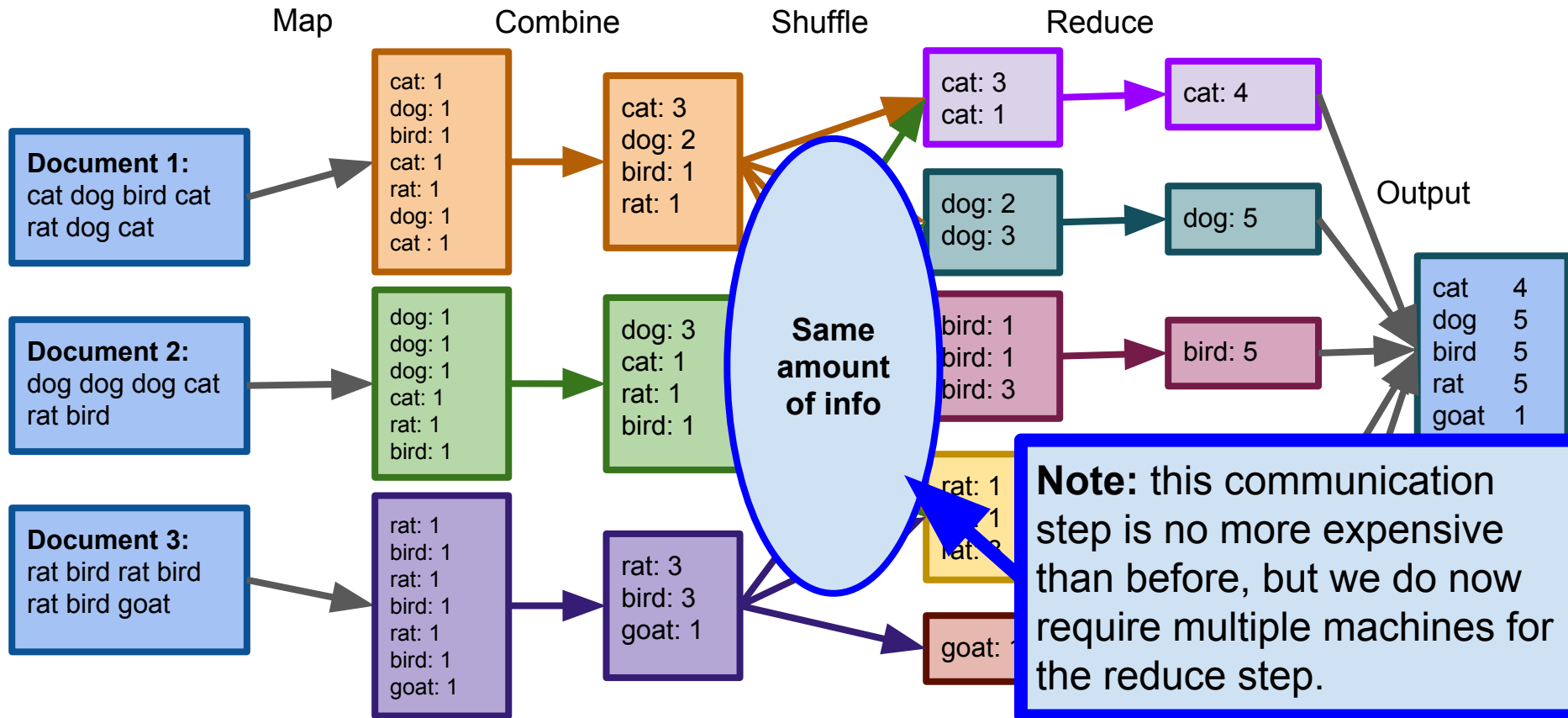
# Counting words in MapReduce: version 2



# Counting words in MapReduce version 3



# Counting words in MapReduce version 3



# MapReduce: under the hood

MR job consists of:

- A master **job tracker** or **resource manager** node

- A number of **worker** nodes

## **Resource manager:**

- schedules and assigns tasks to workers

- monitors workers, reschedules tasks if a worker node fails

- [https://en.wikipedia.org/wiki/Fault-tolerant\\_computer\\_system](https://en.wikipedia.org/wiki/Fault-tolerant_computer_system)

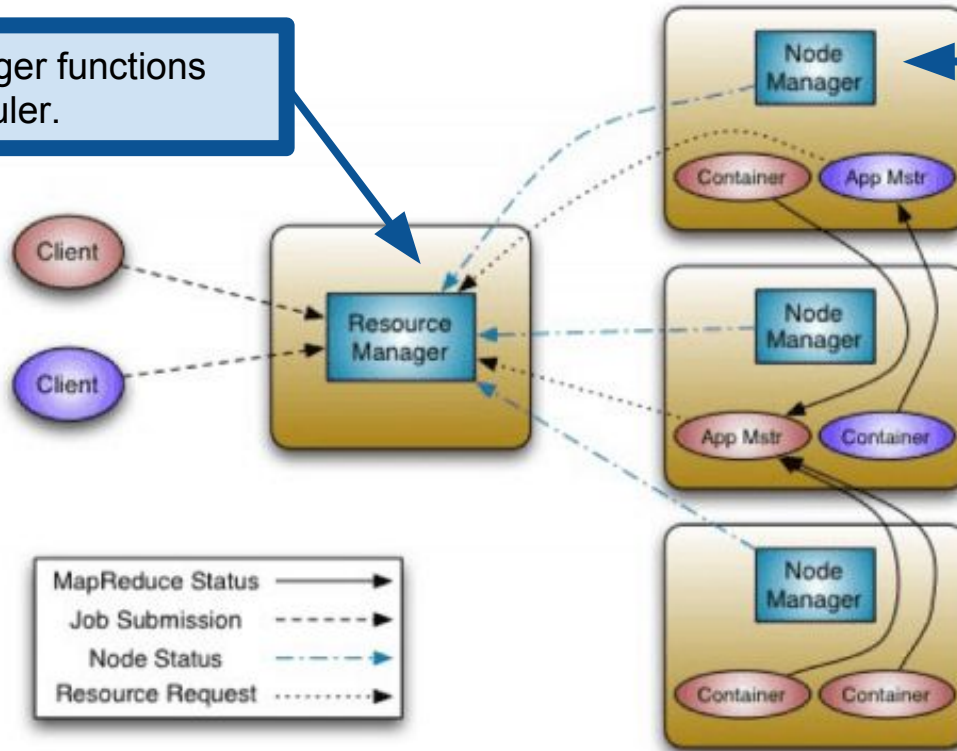
## **Worker nodes:**

- Perform computations as directed by resource manager

- Communicate results to downstream nodes (e.g., Mapper -> Reducer)

# Hadoop v2 YARN schematic

Resource manager functions only as a scheduler.



**Note:** manager is a process (i.e., program) that runs on a node and controls processing of data on that node.

So everything except allocation of tasks is performed at the **worker nodes**. Even much of the resource allocation is done by worker nodes via the **ApplicationMaster**.

H

Res  
only

You do not have to commit any of this to memory, or even understand it all. The important point here is that **Hadoop/YARN hides a whole bunch of complexity** from you so that you don't have to worry about it.

d

er  
the  
via  
r.



# Hadoop Distributed File System (HDFS)

Storage system for Hadoop

File system is **distributed** across multiple nodes on the network  
In contrast to, say, all of your files being on one computer

## **Fault tolerant**

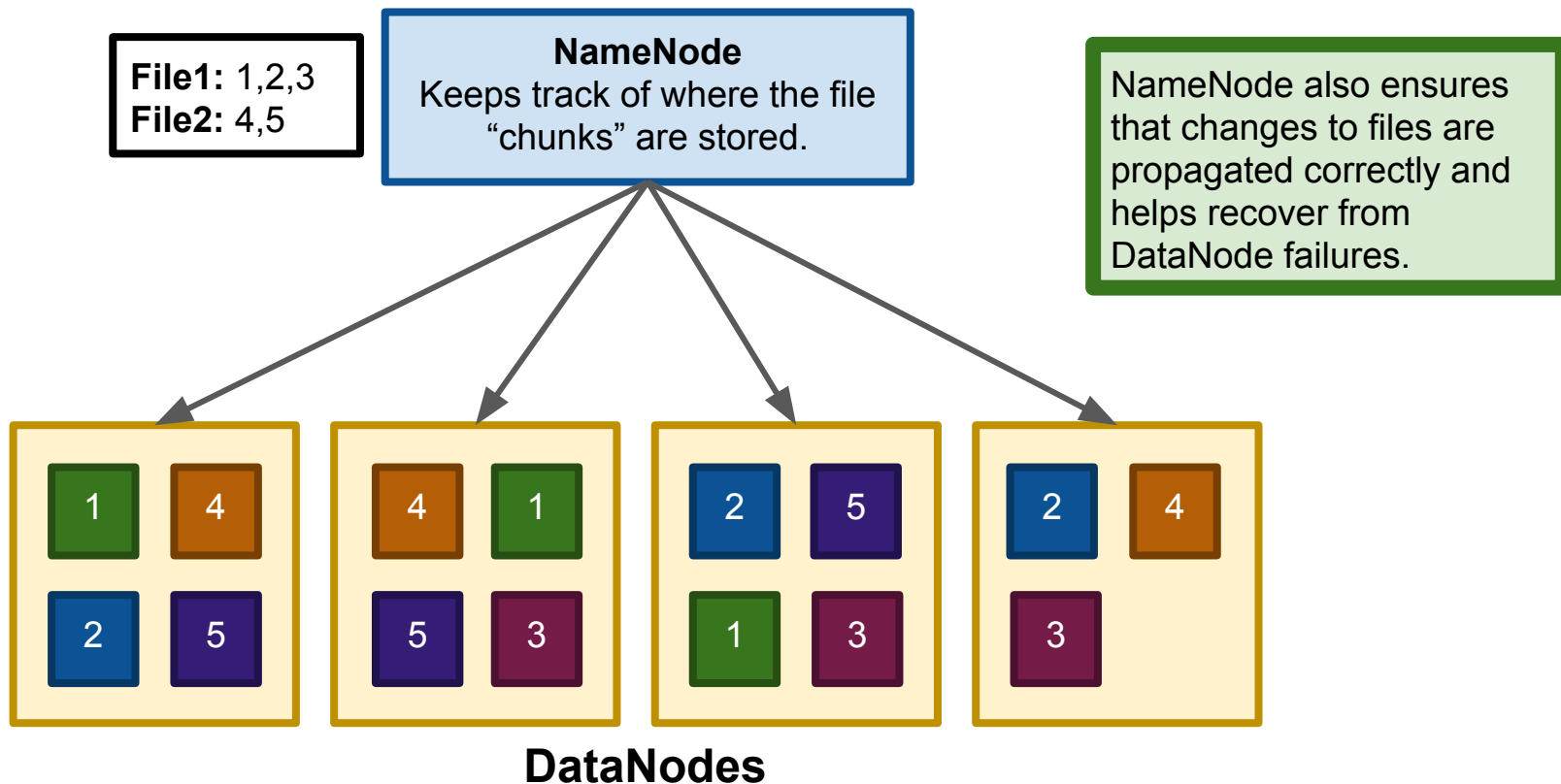
Multiple copies of files are stored on different nodes  
If nodes fail, recovery is still possible

## **High-throughput**

Many large files, accessible by multiple readers and writers, simultaneously

Details: <https://www.ibm.com/developerworks/library/wa-introhdfs/index.html>

# HDFS Schematic



# HDFS Schematic

File1: 1.2.3

NameNode

Keeps track of where the file

NameNode also ensures

Again, the important point is that HDFS does all the hard work so you don't have to!

DataNodes

# Readings

## Required:

J. Dean and S. Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters” in Proceedings of the Sixth Symposium on Operating System Design and Implementation, 2004.

<https://research.google.com/archive/mapreduce.html>

This is the paper that originally introduced the MapReduce framework, and it's still, in my opinion, an excellent place to start. Don't worry too much about understanding every bit of the paper-- it's written for computer systems engineers.

## Recommended:

“Introduction to HDFS” by J. Hanson

<https://www.ibm.com/developerworks/library/wa-introhdfs/index.html>