CSE 6242 / CX 4242

Scaling Up 1

Hadoop, Pig

Duen Horng (Polo) Chau
Georgia Tech

Some lectures are partly based on materials by
Professors Guy Lebanon, Jeffrey Heer, John Stasko, Christos Faloutsos, Le Song

How to handle data that is **really** large?

Really big, as in...

- Petabytes (PB, about 1000 times of terabytes)
- Or beyond: exabyte, zettabyte, etc.

Do we *really* need to deal with such scale?

- Yes!

Big Data is Quite Common...

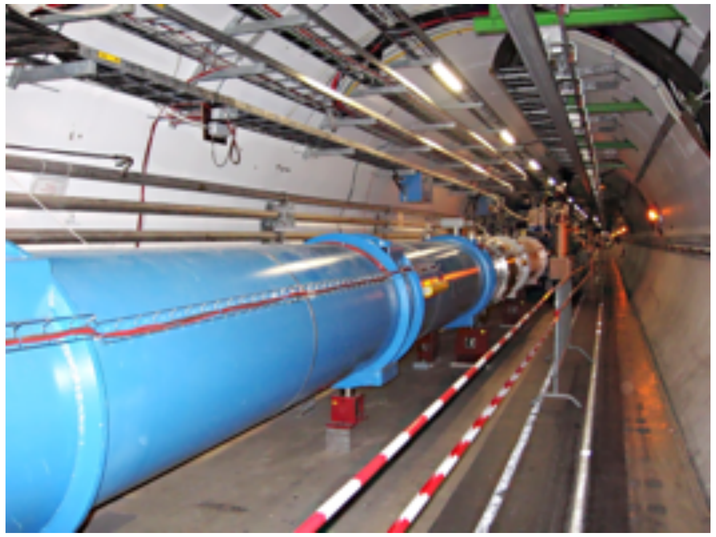
Google processed **24 PB / day** (2009)

Facebook's add **0.5 PB / day** to its data warehouses

CERN generated **200 PB** of data from "Higgs boson" experiments

Avatar's 3D effects took **1 PB** to store

So, think **BIG**!



http://www.theregister.co.uk/2012/11/09/facebook_open_sources_corona/
http://thenextweb.com/2010/01/01/avatar-takes-1-petabyte-storage-space-equivalent-32-year-long-mp3/
http://dl.acm.org/citation.cfm?doid=1327452.1327492

How to analyze such large datasets?

First thing, how to **store** them?

Single machine? 6TB Seagate drive is out.

Cluster of machines?

- How many machines?
- Need to worry about machine and drive failure. **Really?**
- Need data backup, redundancy, recovery, etc.

3% of 100,000 hard drives fail within **first 3 months**

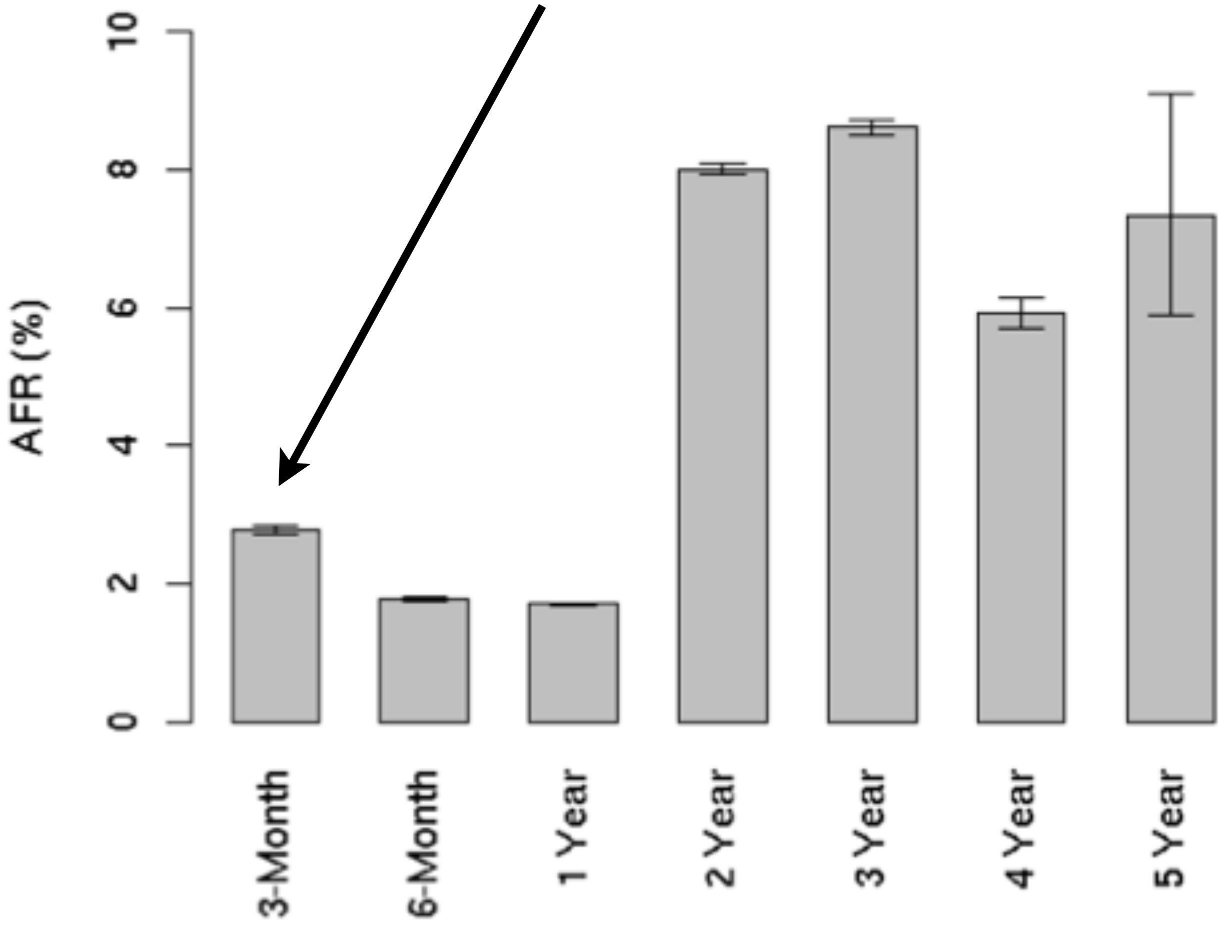


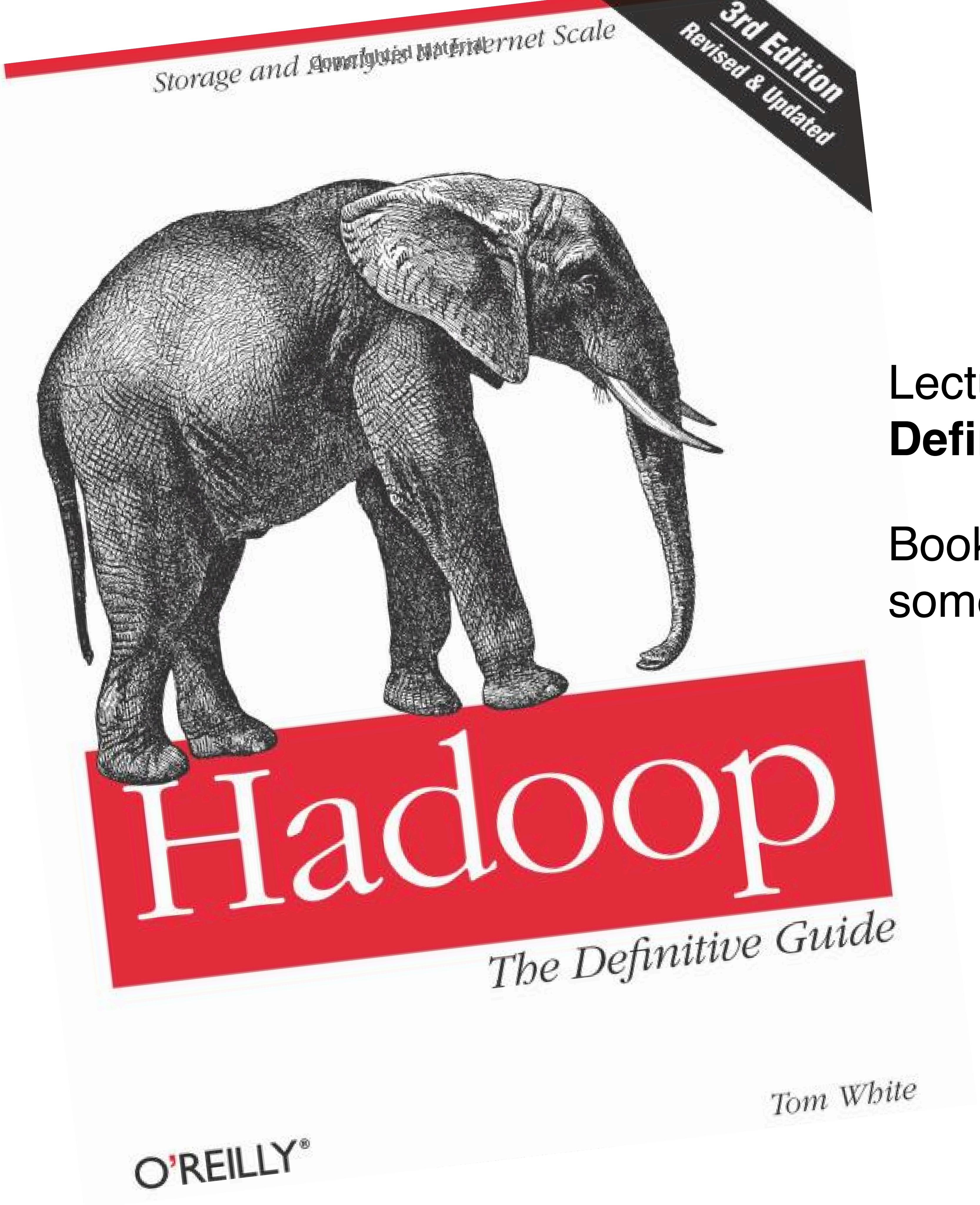
Figure 2: Annualized failure rates broken down by age groups

Failure Trends in a Large Disk Drive Population
http://static.googleusercontent.com/external_content/untrusted_dlcp/research.google.com/en/us/archive/disk_failures.pdf

How to analyze such large datasets?

How to analyze them?

- What software libraries to use?
- What programming languages to learn?
- Or more generally, what framework to use?



Lecture based on **Hadoop: The Definitive Guide**

Book covers Hadoop, some Pig, some HBase, and other things.

http://goo.gl/YNCWN

hadoop

Open-source software for reliable, scalable, distributed computing

Written in Java

Scale to thousands of machines

- Linear scalability (with good algorithm design): if you have 2 machines, your job runs twice as fast

Uses simple programming model (MapReduce)

Fault tolerant (HDFS)

- Can recover from machine/disk failure (no need to restart computation)

http://hadoop.apache.org

Why learn Hadoop?

Fortune 500 companies use it

Many research groups/projects use it

Strong community support, and favored/backed my major companies, e.g., IBM, Google, Yahoo, eBay, Microsoft, etc.

It’s **free**, open-source

Low cost to set up (works on commodity machines)

Will be an “essential skill”, like SQL

http://strataconf.com/strata2012/public/schedule/detail/22497

Elephant in the room

Hadoop created by Doug Cutting and Michael Cafarella while at Yahoo

Hadoop named after Doug’s son’s toy elephant

How does Hadoop scales up computation?

Uses master-slave architecture, and a simple computation model called **MapReduce** (popularized by Google's paper)

Simple explanation

1. Divide data and computation into smaller pieces; each machine works on one piece
2. Combine results to produce final results

MapReduce: Simplified Data Processing on Large Clusters
http://static.usenix.org/event/osdi04/tech/full_papers/dean/dean.pdf

How does Hadoop scales up computation?

More technically...

1. **Map phase**
   Master node **divides** data and computation into smaller pieces; each machine (“mapper”) works on one piece independently in parallel
2. **Shuffle phase** (automatically done for you)
   Master **sorts and moves** results to “reducers”
3. **Reduce phase**
   Machines (“reducers”) **combines** results independently in parallel

An example
Find words’ frequencies among text documents

Input

- “Apple Orange Mango Orange Grapes Plum”
- “Apple Plum Mango Apple Apple Plum”

Output

- Apple, 4
  Grapes, 1
  Mango, 2
  Orange, 2
  Plum, 3

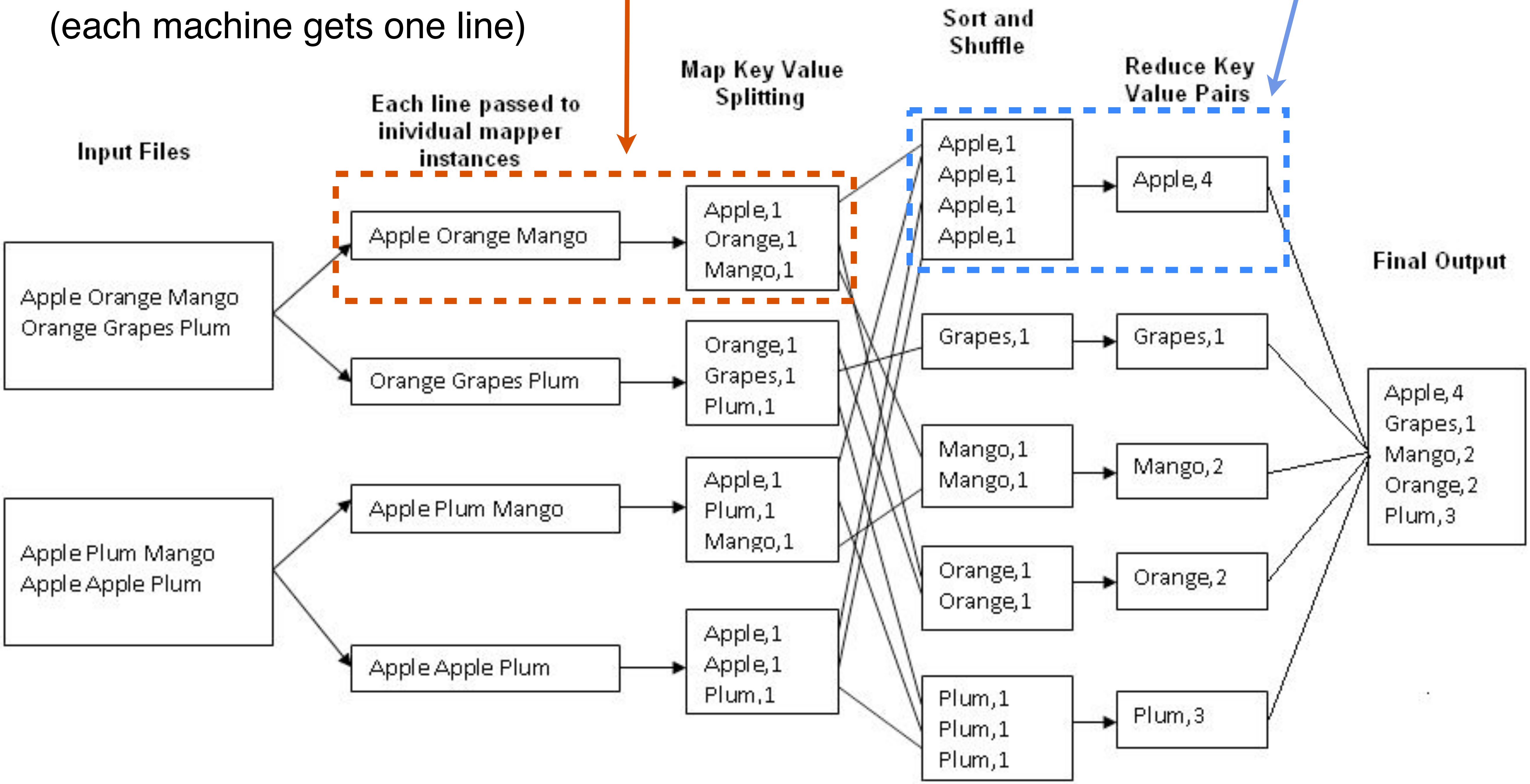
http://kickstarthadoop.blogspot.com/2011/04/word-count-hadoop-map-reduce-example.html

Each machine (mapper) outputs a **key-value pair**

Pairs sorted by key
(automatically done)

Each machine (reducer) combines pairs into one

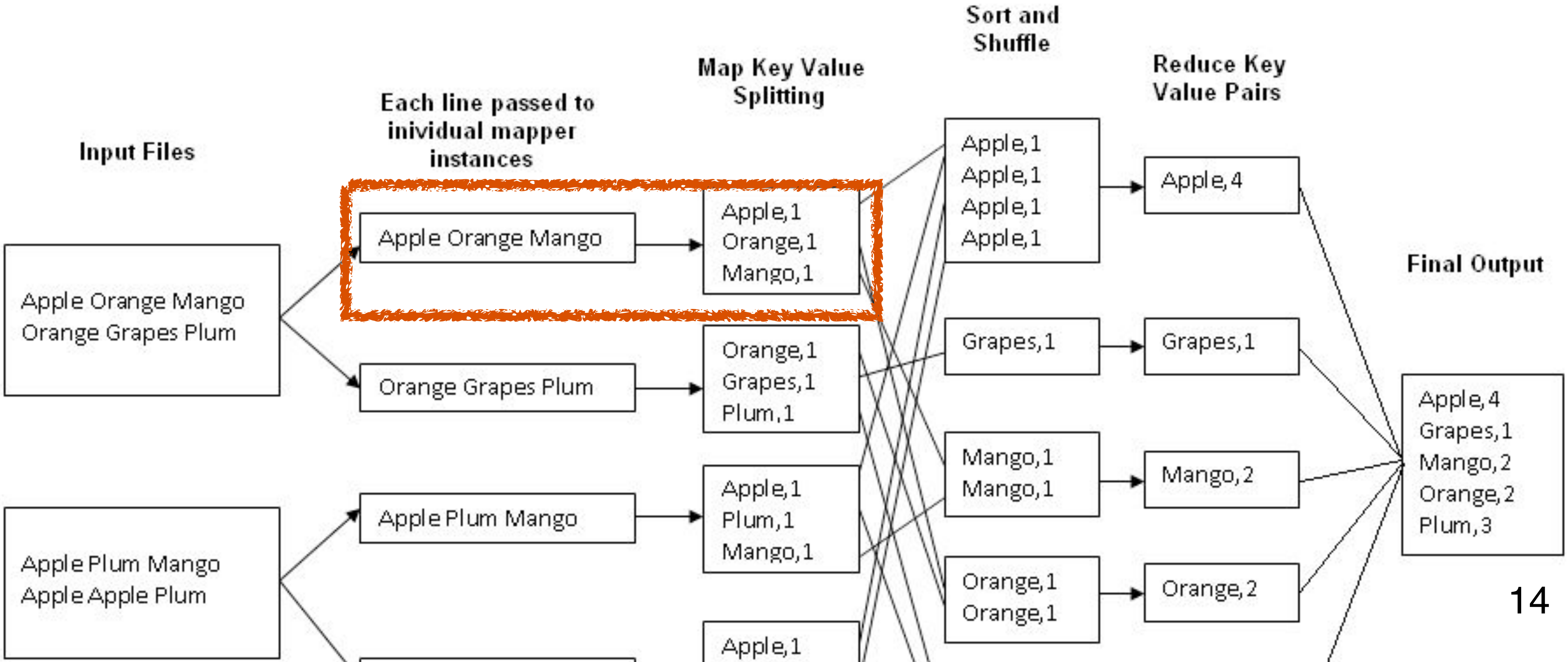
Master divides the data
(each machine gets one line)



A machine can be **both** a mapper and a reducer

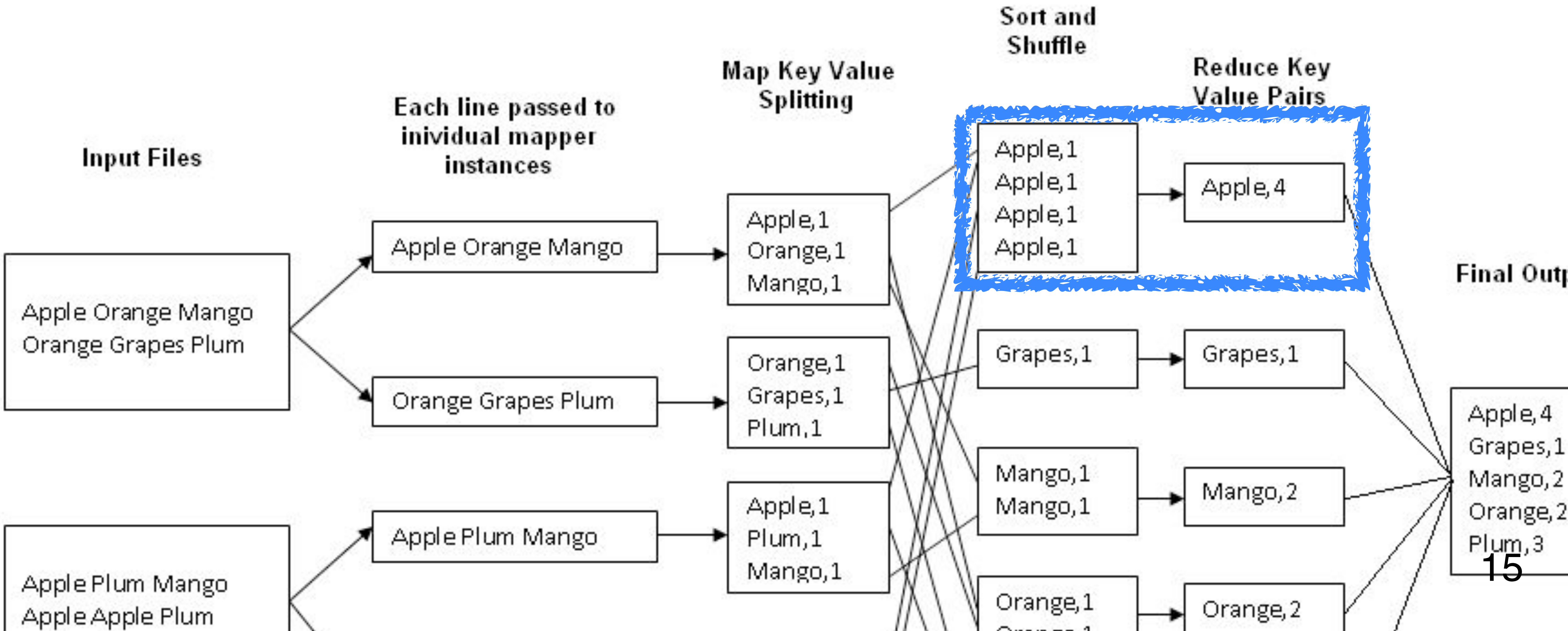
How to implement this?

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



How to implement this?

```
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));
```



What can you use Hadoop for?

As a “swiss knife”.

Works for many types of analyses/tasks (but not all of them).

What if you want to write less code?

- There are tools to make it easier to write MapReduce program (**Pig**), or to query results (**Hive**)

What if a machine dies?

Replace it!

- “map” and “reduce” jobs can be redistributed to other machines

Hadoop’s HDFS (Hadoop File System) enables this

HDFS: Hadoop File System

A distribute file system

Built on top of OS's existing file system to provide redundancy and distribution

HSDF hides complexity of distributed storage and redundancy from the programmer

In short, you don't need to worry much about this!

How to try Hadoop?

Hadoop can run on a single machine (e.g., your laptop)

- Takes < 30min from setup to running

Or a “home-brew” cluster

- Research groups often connect retired computers as a small cluster

Amazon EC2 (Amazon Elastic Compute Cloud)

- You only pay for what you use, e.g, compute time, storage
- You will use it in our next assignment (tentative)

http://aws.amazon.com/ec2/

Pig

http://pig.apache.org

High-level language

- instead of writing low-level map and reduce functions

Easy to program, understand and maintain

Created at Yahoo!

Produces sequences of Map-Reduce programs

(Lets you do “joins” much more easily)

Pig

http://pig.apache.org

Your data analysis task -> a **data flow** sequence

- **Data flow** sequence
  = sequence of **data transformations**

Input -> **data flow**-> output

You specify the **data flow** in **Pig Latin** (Pig's language)

- Pig turns the data flow into a sequence of MapReduce jobs automatically!

Pig: 1st Benefit

Write only a few lines of Pig Latin

Typically, MapReduce development cycle is long

- Write mappers and reducers
- Compile code
- Submit jobs
- ...

Pig: 2nd Benefit

Pig can perform a **sample run** on representative subset of your input data automatically!

Helps debug your code (in smaller scale), before applying on full data

What Pig is good for?

Batch processing, since it’s built on top of MapReduce

- Not for random query/read/write

May be slower than MapReduce programs coded from scratch

- You trade ease of use + coding time for some execution speed

How to run Pig

Pig is a client-side application
(run on your computer)

Nothing to install on Hadoop cluster

How to run Pig: 2 modes

Local Mode

- Run on your computer
- Great for trying out Pig on small datasets

MapReduce Mode

- Pig translates your commands into MapReduce jobs and turns them on Hadoop cluster
  - Remember you can have a single-machine cluster set up on your computer

Pig program: 3 ways to write

Script

Grunt (interactive shell)

- Great for debugging

Embedded (into Java program)

- Use PigServer class (like JDBC for SQL)
- Use PigRunner to access Grunt

Grunt (interactive shell)

Provides “code completion”; press “Tab” key to complete Pig Latin keywords and functions

Let’s see an example Pig program run with Grunt

- Find highest temperature by year

Example Pig program

Find highest temperature by year

```
records = LOAD 'input/ ncdc/ micro-tab/ sample.txt'
  AS (year:chararray, temperature:int, quality:int);

filtered_records =
  FILTER records BY temperature != 9999
  AND (quality = = 0 OR quality = = 1 OR
       quality = = 4 OR quality = = 5 OR
       quality = = 9);

grouped_records = GROUP filtered_records BY year;

max_temp = FOREACH grouped_records GENERATE
  group, MAX(filtered_records.temperature);

DUMP max_temp;
```

Example Pig program

Find highest temperature by year

```
grunt>
records = LOAD 'input/ ncdc/ micro-tab/ sample.txt'
  AS (year:chararray, temperature:int, quality:int);
```

```
grunt> DUMP records;
```

```
(1950,0,1)
(1950,22,1)
(1950,-11,1)
(1949,111,1)
(1949,78,1)
```

called a “tuple”

```
grunt> DESCRIBE records;
```

```
records: {year: chararray, temperature: int, quality: int}
```

Example Pig program

Find highest temperature by year

```
grunt>
filtered_records =
  FILTER records BY temperature != 9999
  AND (quality = = 0 OR quality = = 1 OR
       quality = = 4 OR quality = = 5 OR
       quality = = 9);

grunt> DUMP filtered_records;
```

```
(1950,0,1)
(1950,22,1)
(1950,-11,1)
(1949,111,1)
(1949,78,1)
```

In this example, no tuple is filtered out

Example Pig program

Find highest temperature by year

```
grunt> grouped_records = GROUP filtered_records BY year;

grunt> DUMP grouped_records;
```

```
(1949,{(1949,111,1), (1949,78,1)})
(1950,{(1950,0,1),(1950,22,1),(1950,-11,1)})
```

called a “bag”

= unordered collection of tuples

```
grunt> DESCRIBE grouped_records;
```

alias that Pig created

```
grouped_records: {group: chararray,
filtered_records: {year: chararray, temperature:
int, quality: int}}
```

Example Pig program

Find highest temperature by year

```
(1949,{(1949,111,1), (1949,78,1)})
(1950,{(1950,0,1),(1950,22,1),(1950,-11,1)})
```

```
grouped_records: {group: chararray, filtered_records: {year:
chararray, temperature: int, quality: int}}
```

```
grunt> max_temp = FOREACH grouped_records GENERATE
  group, MAX(filtered_records.temperature);

grunt> DUMP max_temp;
```

```
(1949,111)
(1950,22)
```

Run Pig program on a subset of your data

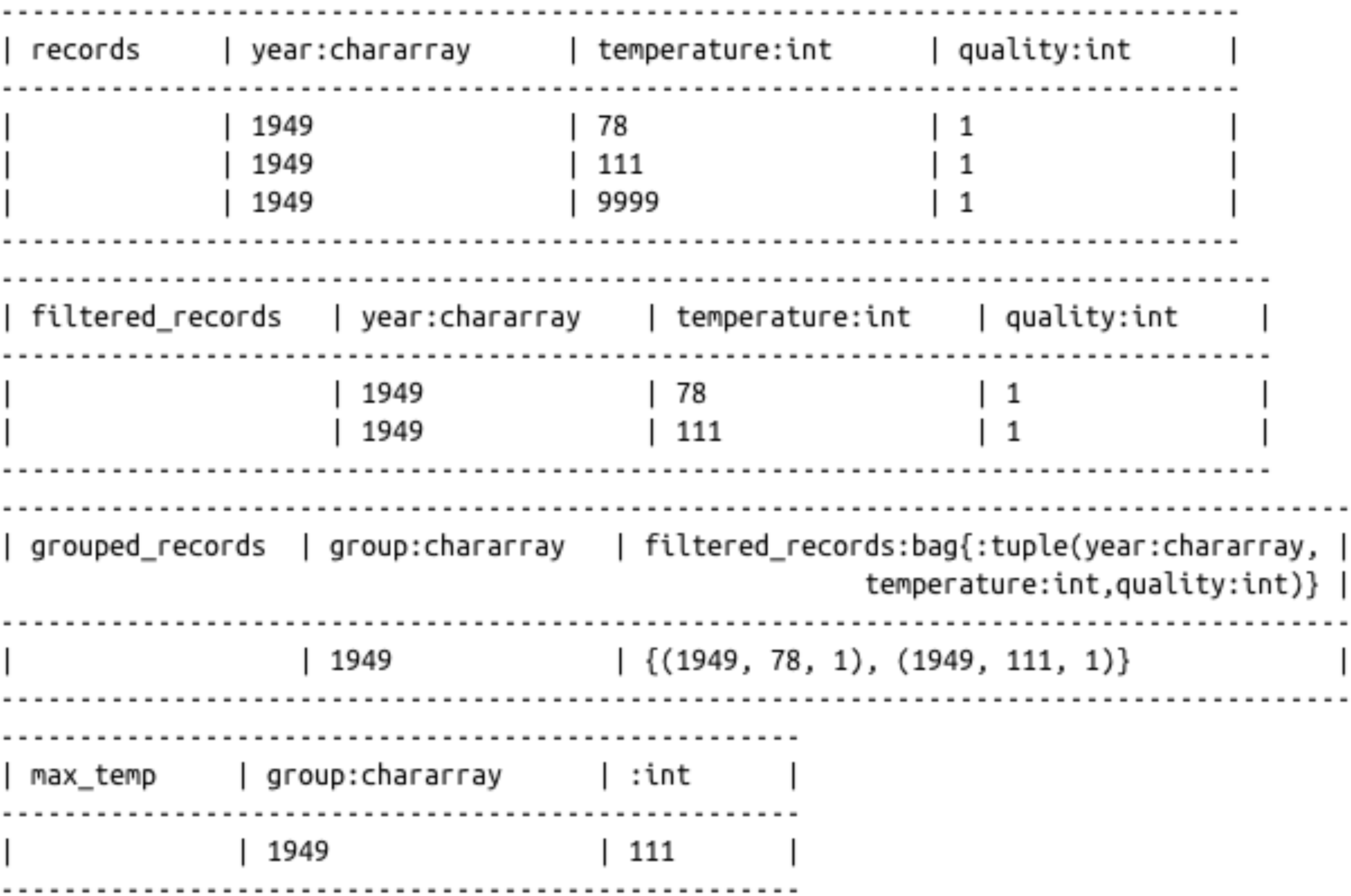
You saw an example run on a tiny dataset

How to do that for a larger dataset?

- Use the **ILLUSTRATE** command to generate sample dataset

Run Pig program on a subset of your data

```
grunt> ILLUSTRATE max_temp;
```



```
-------------------------------------------------------------------------------
| records       | year:chararray        | temperature:int       | quality:int  |
-------------------------------------------------------------------------------
|               | 1949                  | 78                    | 1            |
|               | 1949                  | 111                   | 1            |
|               | 1949                  | 9999                  | 1            |
-------------------------------------------------------------------------------
----------------------------------------------------------------------------------
| filtered_records   | year:chararray    | temperature:int      | quality:int     |
----------------------------------------------------------------------------------
|                    | 1949              | 78                   | 1               |
|                    | 1949              | 111                  | 1               |
----------------------------------------------------------------------------------
---------------------------------------------------------------------------------------
| grouped_records  | group:chararray    | filtered_records:bag{:tuple(year:chararray, |
                                                      temperature:int,quality:int)} |
---------------------------------------------------------------------------------------
|                  | 1949               | {(1949, 78, 1), (1949, 111, 1)}            |
---------------------------------------------------------------------------------------
--------------------------------------------------
| max_temp      | group:chararray      | :int     |
--------------------------------------------------
|               | 1949                 | 111      |
--------------------------------------------------
```

How does Pig compare to SQL?

SQL: “fixed” schema

PIG: loosely defined schema, as in

```
records = LOAD 'input/ ncdc/ micro-tab/ sample.txt'
  AS (year:chararray, temperature:int, quality:int);
```

How does Pig compare to SQL?

SQL: supports fast, random access
(e.g., <10ms)

PIG: batch processing

Much more to learn about Pig

Relational Operators, Diagnostic Operators (e.g., describe, explain, illustrate), utility commands (cat, cd, kill, exec), etc.

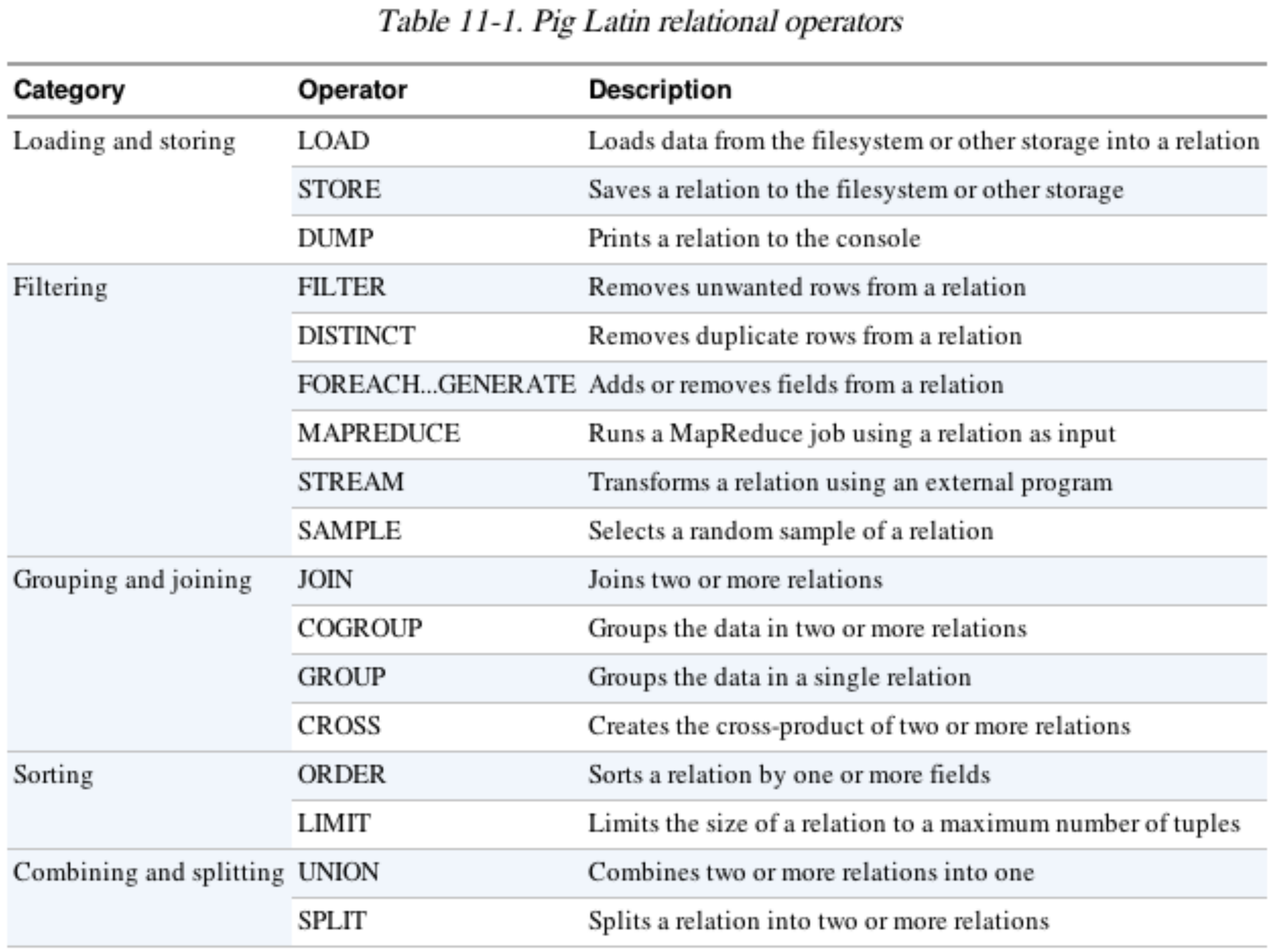


Table 11-1. Pig Latin relational operators

Category	Operator	Description
Loading and storing	LOAD	Loads data from the filesystem or other storage into a relation
	STORE	Saves a relation to the filesystem or other storage
	DUMP	Prints a relation to the console
Filtering	FILTER	Removes unwanted rows from a relation
	DISTINCT	Removes duplicate rows from a relation
	FOREACH...GENERATE	Adds or removes fields from a relation
	MAPREDUCE	Runs a MapReduce job using a relation as input
	STREAM	Transforms a relation using an external program
	SAMPLE	Selects a random sample of a relation
Grouping and joining	JOIN	Joins two or more relations
	COGROUP	Groups the data in two or more relations
	GROUP	Groups the data in a single relation
	CROSS	Creates the cross-product of two or more relations
Sorting	ORDER	Sorts a relation by one or more fields
	LIMIT	Limits the size of a relation to a maximum number of tuples
Combining and splitting	UNION	Combines two or more relations into one
	SPLIT	Splits a relation into two or more relations

What if you need **real-time** read/write for large datasets?