# An Introduction to Cluster Computing

**Apostolos N. Papadopoulos**

Assistant Professor
Data Engineering Lab,
Department of Informatics
Aristotle University of Thessaloniki
GREECE

# Outline

- Why one machine is not enough ?

- Parallel architectures

- Scaling out vs scaling up

- Important issues in cluster computing

- Hadoop MR recap

- Theoretical Issues

# Motivation

We need **more CPUs** because:

we can run programs faster

We need **more disks** because:

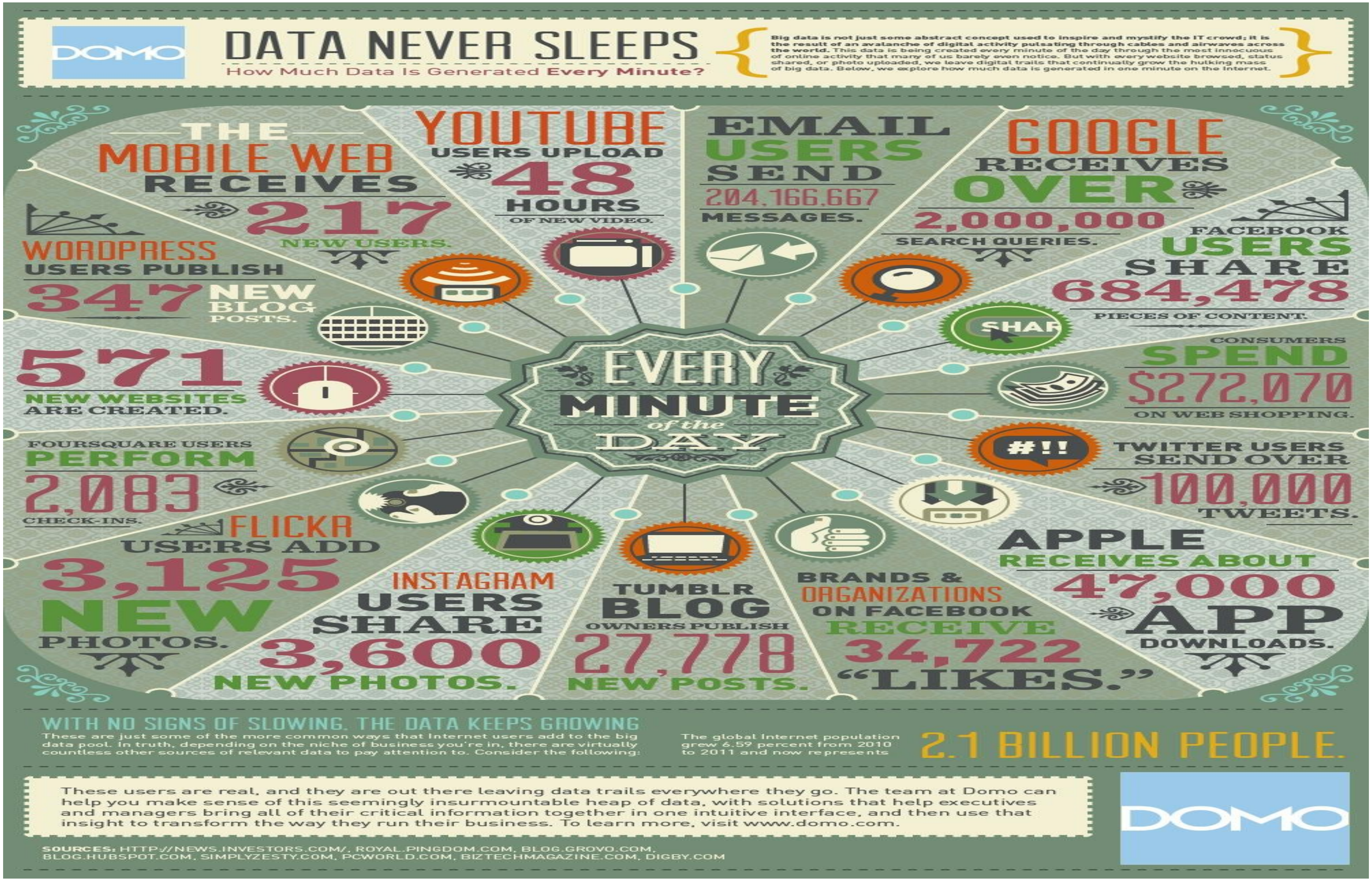
modern applications require huge amounts of data

with many disks we can perform I/O in parallel

Assume that we are able to build a single disk with **500 TB** capacity. This is enough to store more than **20 billion webpages** (assuming an average size per page of **20KB**).

However, **just to scan** these 500 TB we need **more than 4 months** if the disk can bring **40 MB/sec**. Imagine the time required to process the data !

# What is Happening Today



DOMO

DATA NEVER SLEEPS

How Much Data Is Generated **Every Minute?**

**Big data is not just some abstract concept used to inspire and mystify the IT crowd; it is the result of an avalanche of digital activity pulsating through cables and airwaves across the world.** This data is being created every minute of the day through the most innocuous of online activity that many of us barely even notice. But with every website browsed, status shared, or photo uploaded, we leave digital trails that continually grow the hulking mass of big data. Below, we explore how much data is generated in one minute on the Internet.

EVERY MINUTE *of the* DAY

THE MOBILE WEB RECEIVES 217 NEW USERS.

YOUTUBE USERS UPLOAD 48 HOURS OF NEW VIDEO.

EMAIL USERS SEND 204,166,667 MESSAGES.

GOOGLE RECEIVES OVER 2,000,000 SEARCH QUERIES.

FACEBOOK USERS SHARE 684,478 PIECES OF CONTENT.

CONSUMERS SPEND $272,070 ON WEB SHOPPING.

TWITTER USERS SEND OVER 100,000 TWEETS.

APPLE RECEIVES ABOUT 47,000 APP DOWNLOADS.

BRANDS & ORGANIZATIONS ON FACEBOOK RECEIVE 34,722 "LIKES."

TUMBLR BLOG OWNERS PUBLISH 27,778 NEW POSTS.

INSTAGRAM USERS SHARE 3,600 NEW PHOTOS.

FLICKR USERS ADD 3,125 NEW PHOTOS.

FOURSQUARE USERS PERFORM 2,083 CHECK-INS.

571 NEW WEBSITES ARE CREATED.

WORDPRESS USERS PUBLISH 347 NEW BLOG POSTS.

WITH NO SIGNS OF SLOWING, THE DATA KEEPS GROWING

These are just some of the more common ways that Internet users add to the big data pool. In truth, depending on the niche of business you're in, there are virtually countless other sources of relevant data to pay attention to. Consider the following:

The global Internet population grew 6.59 percent from 2010 to 2011 and now represents 2.1 BILLION PEOPLE.

These users are real, and they are out there leaving data trails everywhere they go. The team at Domo can help you make sense of this seemingly insurmountable heap of data, with solutions that help executives and managers bring all of their critical information together in one intuitive interface, and then use that insight to transform the way they run their business. To learn more, visit www.domo.com.

DOMO

**SOURCES:** HTTP://NEWS.INVESTORS.COM/, ROYAL.PINGDOM.COM, BLOG.GROVO.COM, BLOG.HUBSPOT.COM, SIMPLYZESTY.COM, PCWORLD.COM, BIZTECHMAGAZINE.COM, DIGBY.COM

# In the Near Future

“***IBM Research*** *and Dutch astronomy agency* ***Astron*** *work on new technology to handle*

***one exabyte of raw data per day***

*that will be gathered by the world largest radio telescope, the* ***Square Kilometer Array****, when activated in* ***2024****.*”

## Some Challenges

- Scalability
- Load balancing
- Fault Tolerance
- Efficiency
- Data Stream processing
- Support for complex objects
- Accuracy/Speed tradeoffs (with performance guarantees)

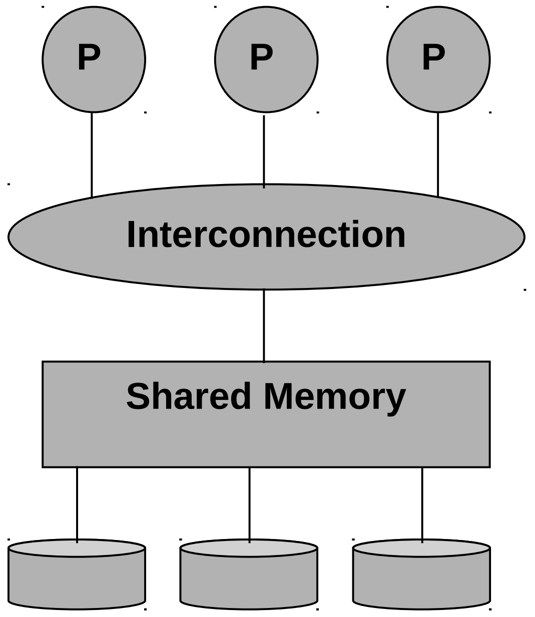
# Parallel Architectures

**Shared Memory**: processors share a common main memory and also share secondary storage (e.g., disks)

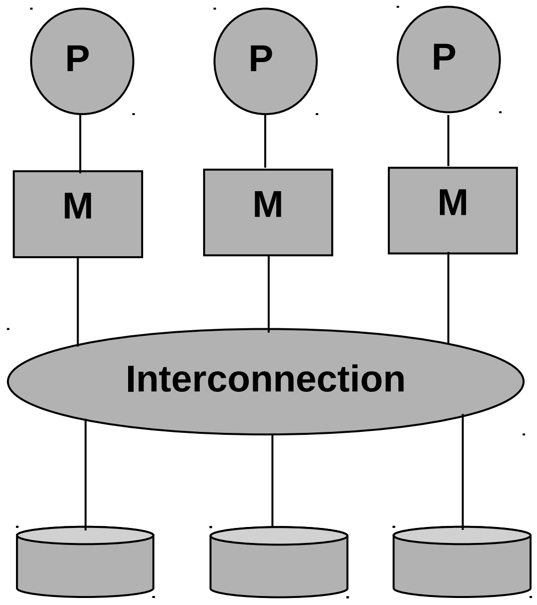
**Shared Disk**: processors share only secondary storage, whereas each processor has its own private memory

**Shared Nothing**: processors do not share anything, each one has private secondary storage and memory

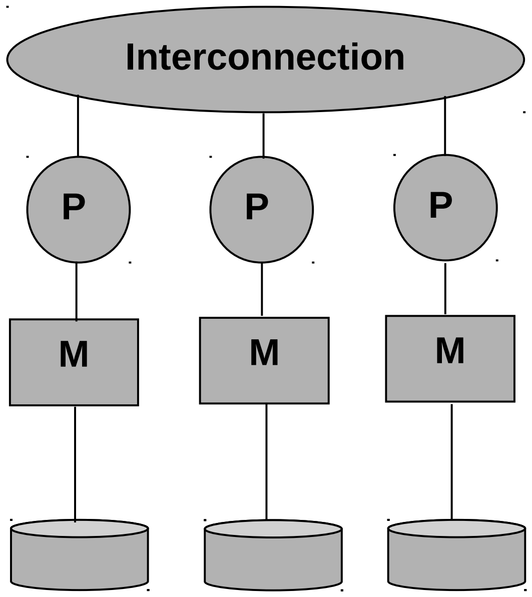
# Parallel Architectures



shared memory



shared disk



shared nothing

# Scalability

**Scale-Up**: put more resources into the system to make it bigger and more powerful

**Scale-Out**: connect a large number of “ordinary” machines and create a cluster

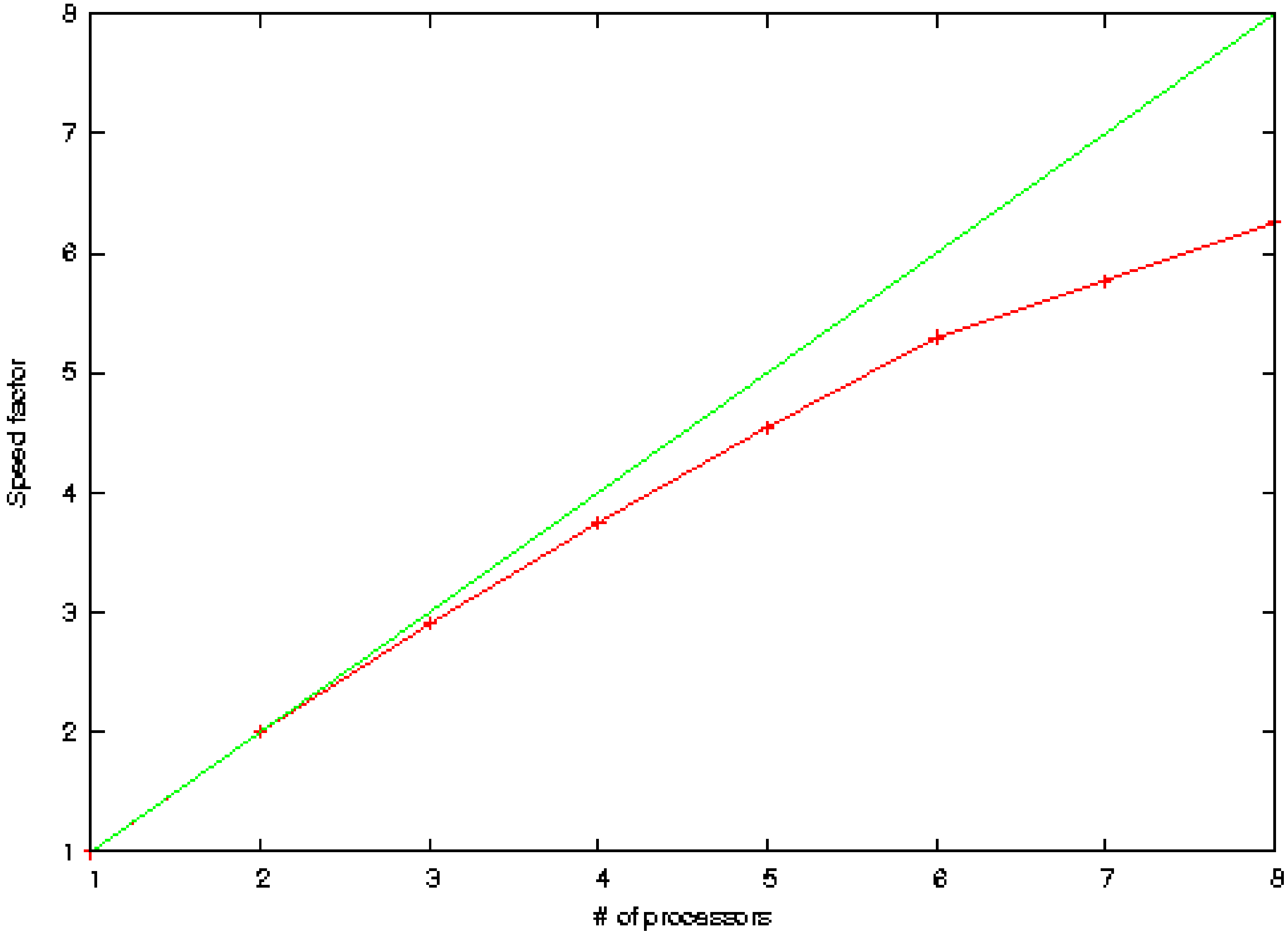
Scale-Out is **more powerful** than Scale-Up, and also **less expensive**

# Scalability: measures

Among the three parallel architectures, shared-nothing is the one that **scales best**. This is the main reason for being adopted for building massively parallel systems (thousands of processors)

- **Speedup**: monitor performance by increasing the number of processors
- **Sizeup**: monitor performance by increasing only the dataset size
- **Scaleup**: monitor performance by increase both the number of processors and the dataset size

# The Speedup Curve



## Real Curves are Non-Linear

Why ?

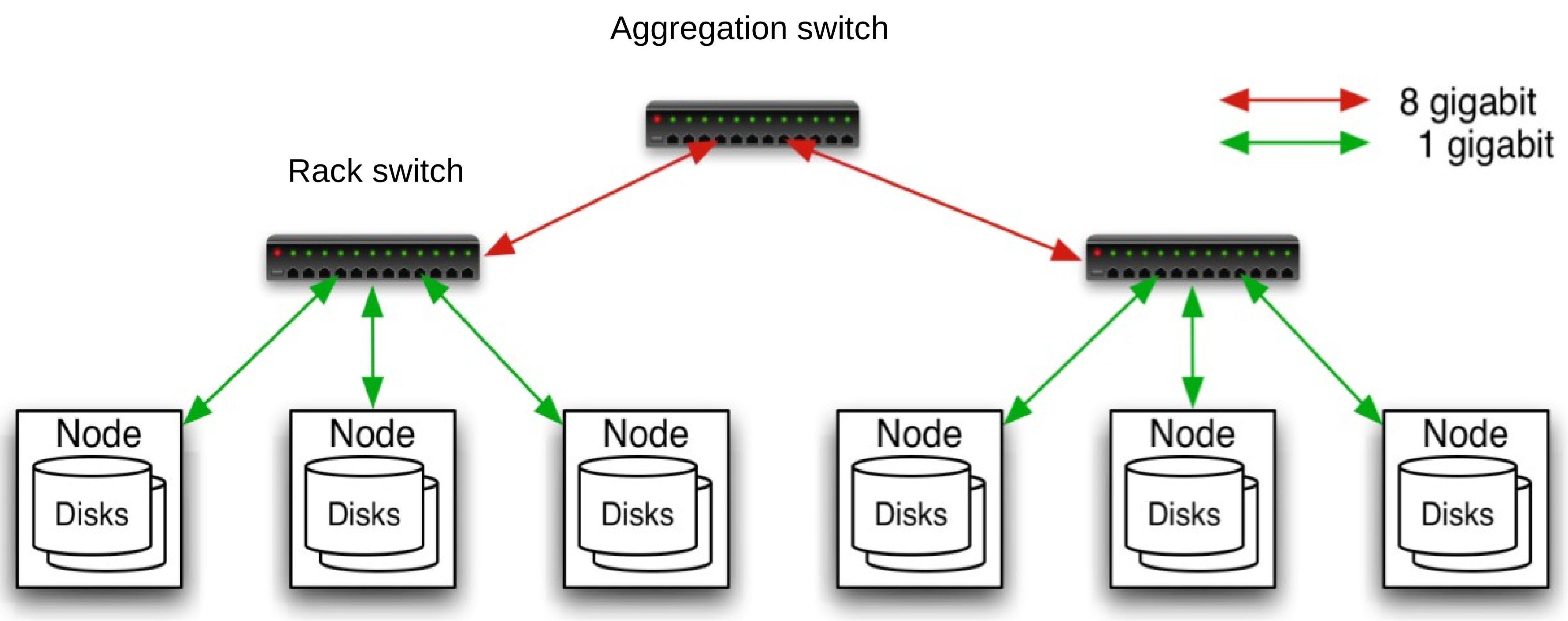
**Start-up costs**: cost for starting an operation in a processor

**Interference**: cost for communication among processors and resource congestion

**Skew**: either in data or tasks → the slowest processor becomes the bottleneck

**Result formation**: partial results from each processor must be combined to provide the final result.

# Cluster Configuration Example



- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth within rack, 8 Gbps out of rack
- 8 x 2GHz cores, 8 GB RAM, 4 disks (= 4 TB?)

*Source: Matei Zaharia*

# Fault Tolerance

**Failures are very common in massively parallel systems**

Let *P* the probability that a disk will fail in the next month. If we have *D* disks in total, the probability that at least one disk will fail is given by:

Prob {*at least one disk failure*} = 1 - (1- *P*)^*D*

e.g., *D* = 10000, *P* = 0.0001

Prob {*at least one disk failure*} = 0.63

## Fault Tolerance

Failures may happen because of:

**Hardware** not working properly

- Disk failure
- Memory failure (8% of DIMMs have problems)
- Inadequate cooling (CPU overheating)

**Resource** unavailability

- Due to overload

We must provide fault tolerance in the cluster!

## Fault Tolerance

Simplest protocol: if there is a failure, restart the job.

Assume a job that requires 1 week of processing. If there is a failure once per week, the job will never finish!

## Fault Tolerance

A better protocol:

Replicate the data and also split the job in parts and replicate them as well.

# Hadoop

A very successful model and platform to run jobs in massively parallel systems (thousands of processors and disks)

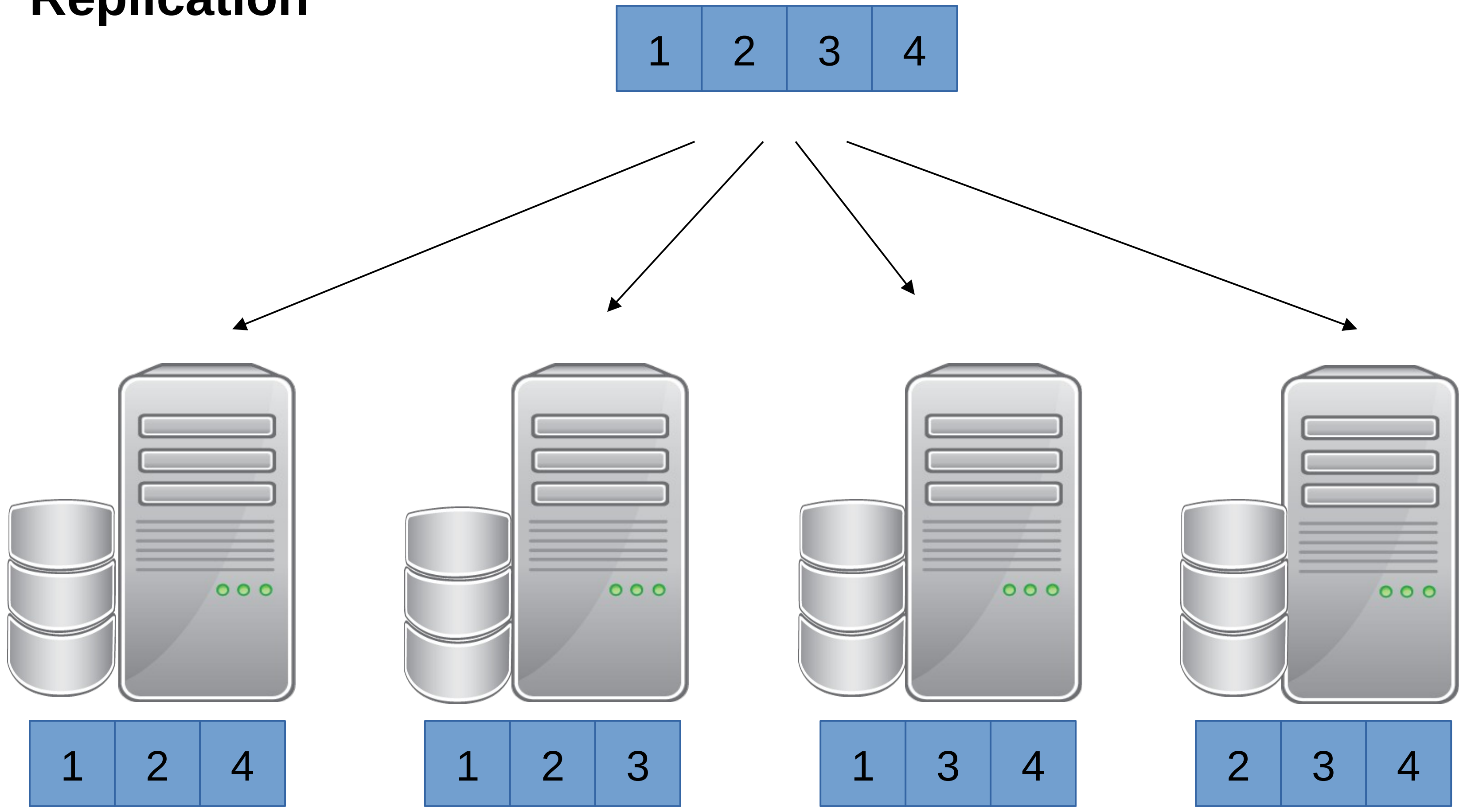
It contains two parts:

- the Hadoop MapReduce layer
- the Hadoop Distributed File System (HDFS)

Hadoop is the open-source alternative of MapReduce and Google File System (GFS) invented by Google. It has been used in Google's data centers mainly for: 1) constructing and maintaining the **Inverted Index** and 2) executing the **PageRank** algorithm.

# Hadoop

**Replication**



The the file is split in chunks. Each is replicated three times in this example.

# Processing in Hadoop

Based on key-value pairs

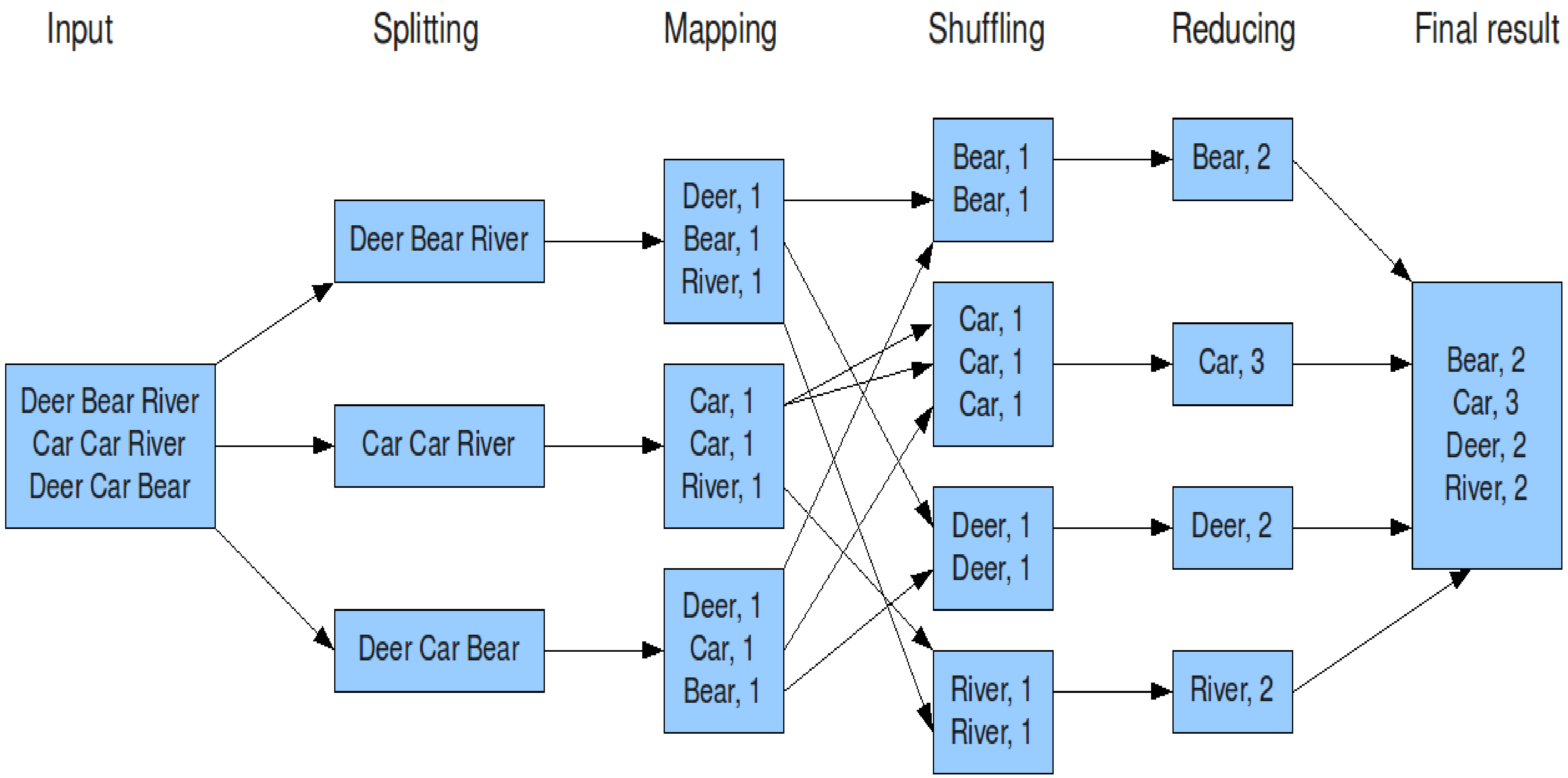
Each job is composed of one or more MR stages

Each MR stage comprises:

- the **map** phase
- the **shuffle-and-sort** phase
- the **reduce** phase

The programmer **focuses on the problem**. Replication, fault tolerance, scheduling, re-scheduling and other low level processes are handled by Hadoop.

# WordCount: the "hello world" of Hadoop



# Hadoop MR API

The programmer must implement the following functions:

**map()**: accepts a set of key-value pairs and generates another list of key-value pairs.

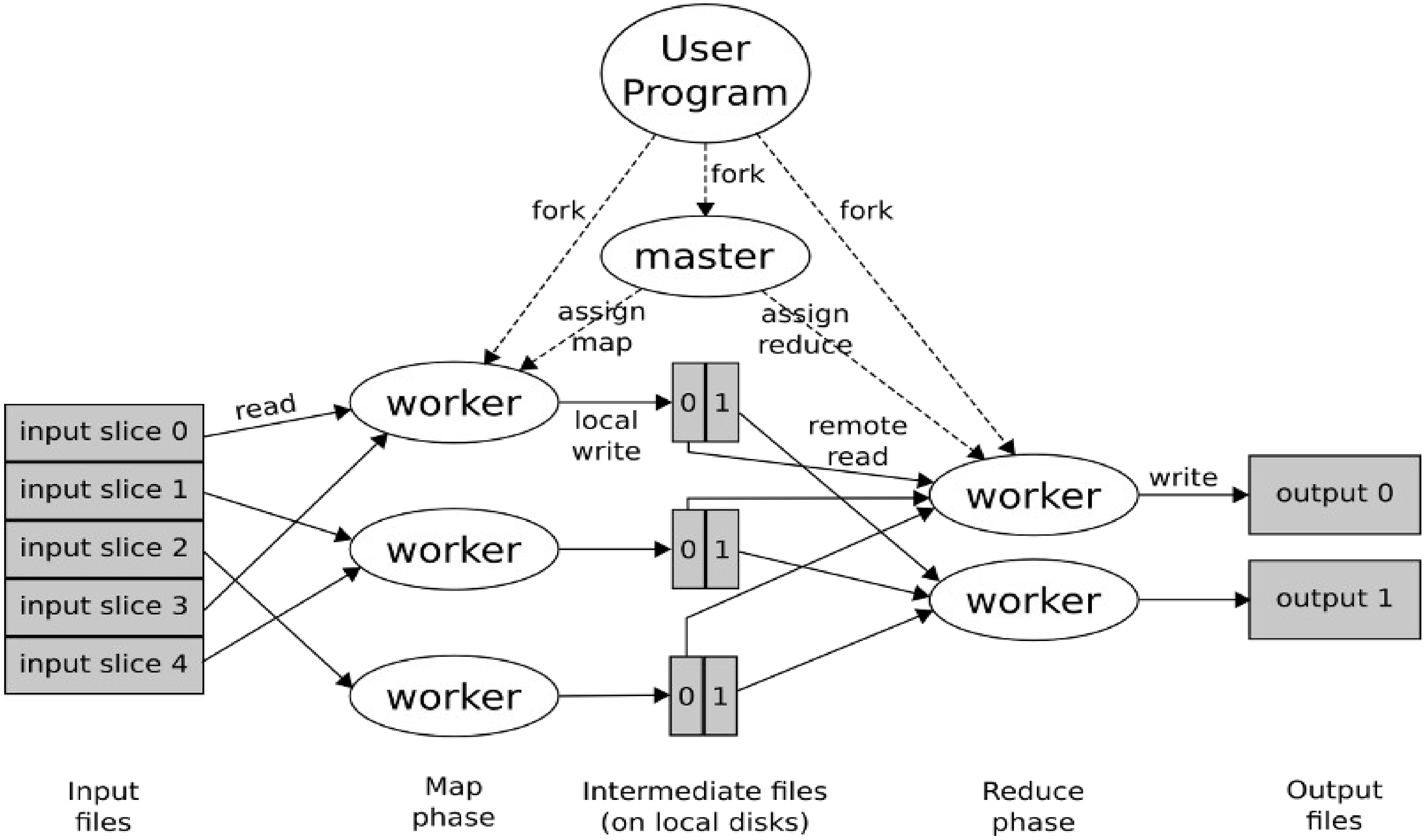
**combine()**: performs an aggregation before sending the data to reducers (reduces network traffic).

**partition()**: uses a hash function to distribute data to reducers (load balancing, avoids hotspots).

**reduce()**: accepts a key and a list of values for this specific key and performs an aggregation.

*Note: combine() and partition() are optional*

# Workflow in Hadoop



# Hadoop Ecosystem

HIVE

APACHE HBASE

hadoop Map Reduce

hadoop HDFS

## Theoretical Issues in MR

Paper titles:

- “On the Computational Complexity of MapReduce”
- “A new Computation Model for Cluster Computing”
- “Fast Greedy Algorithms in MapReduce and Streaming”
- “Minimal MapReduce Algorithms”
- “Filtering: A Method for Solving Graph Problems in MapReduce”
- “A Model of Computation for MapReduce” (**SODA 2010**)

# MR Limitations

Difficult to design efficient/optimal algorithms (everything must be expressed in key-value pairs)

A lot of disk I/Os (mappers reading HDFS and writing local data)

A lot of network traffic (shuffling is expensive)

Difficult to handle data skew (the curse of the last reducer!)

Not very good for iterative processing (requires many MR stages)

Not very good for **streaming applications**

## Is There an Alternative ?

Spark