How much do you think can happen in a minute?

$ 400 M sales on Alibaba

439, 000 page views on Wikipedia

194, 000 apps downloaded

31, 700 hours of music played on Pandora

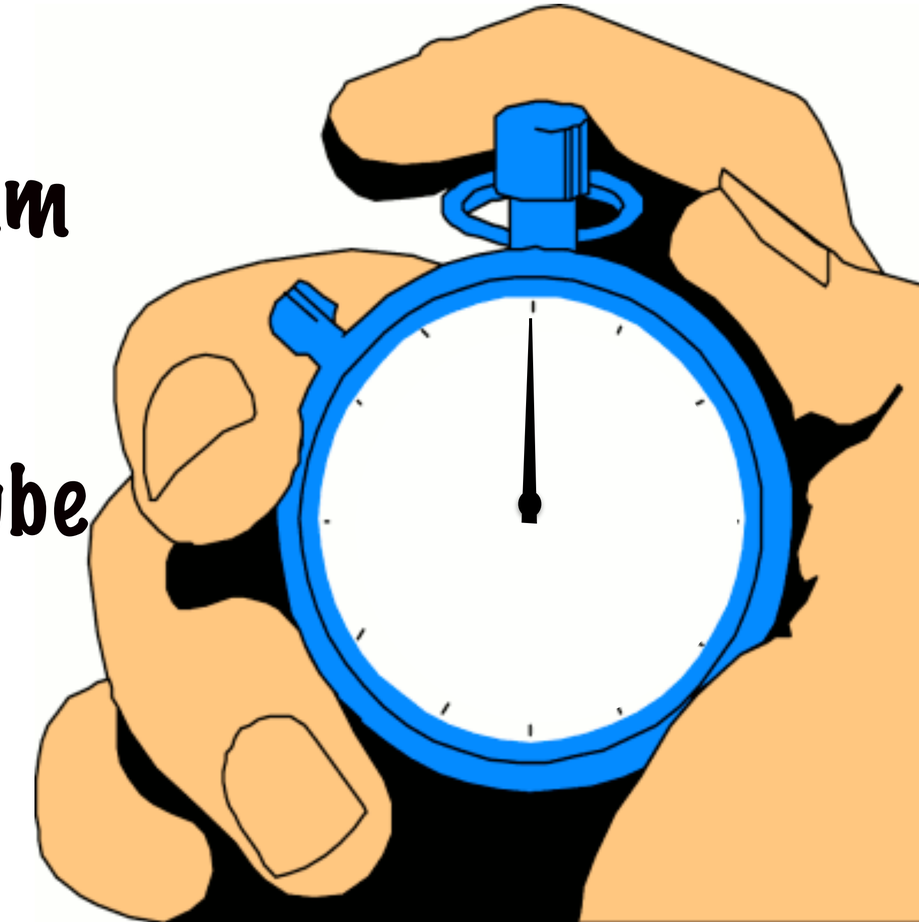
38, 000 photographs uploaded to Instagram

4.1 Million searches on Google

139, 000 hours of video watched on Youtube

10 million ads displayed

3.3 million shares on Facebook



1 internet minute

- $ 400 M sales on Alibaba
- 439, 000 page views on Wikipedia
- 194, 000 apps downloaded
- 31, 700 hours of music played on Pandora
- 38, 000 photographs uploaded to Instagram
- 4.1 Million searches on Google
- 139, 000 hours of video watched on Youtube
- 10 million ads displayed
- 3.3 million shares on Facebook

Each of these activities generates

DATA

DATA

1 internet minute

- $ 400 M sales on Alibaba
- 439, 000 page views on Wikipedia
- 194, 000 apps downloaded
- 31, 700 hours of music played on Pandora
- 38, 000 photographs uploaded to Instagram
- 4.1 Million searches on Google
- 139, 000 hours of video watched on Youtube
- 10 million ads displayed
- 3.3 million shares on Facebook

1 internet minute

$ 400 M sales on Alibaba

439, 000 page views on Wikipedia

194, 000 apps downloaded

31, 700 hours of music played on Pandora

38, 000 photographs uploaded to Instagram

4.1 Million searches on Google

139, 000 hours of video watched on Youtube

10 million ads displayed

3.3 million shares on Facebook

DATA

$ 400 M sales on Alibaba

Product views

Orders

Ratings

Reviews

DATA

1 internet minute

$ 400 M sales on Alibaba

439, 000 page views on Wikipedia

194, 000 apps downloaded

31, 700 hours of music played on Pandora

38, 000 photographs uploaded to Instagram

4.1 Million searches on Google

139, 000 hours of video watched on Youtube

10 million ads displayed

3.3 million shares on Facebook

DATA

1 internet minute

$ 400 M sales on Alibaba

439, 000 page views on Wikipedia

194, 000 apps downloaded

31, 700 hours of music played on Pandora

38, 000 photographs uploaded to Instagram

4.1 Million searches on Google

139, 000 hours of video watched on Youtube

10 million ads displayed

3.3 million shares on Facebook

4.1 Million searches on Google

Results returned

Results viewed

Results clicked

DATA

1 internet minute

$ 400 M sales on Alibaba

439, 000 page views on Wikipedia

194, 000 apps downloaded

31, 700 hours of music played on Pandora

38, 000 photographs uploaded to Instagram

4.1 Million searches on Google

139, 000 hours of video watched on Youtube

10 million ads displayed

3.3 million shares on Facebook

DATA

10 million ads displayed

Ad impressions
Ads clicked

1 internet minute

- $ 400 M sales on Alibaba
- 439, 000 page views on Wikipedia
- 194, 000 apps downloaded
- 31, 700 hours of music played on Pandora
- 38, 000 photographs uploaded to Instagram
- 4.1 Million searches on Google
- 139, 000 hours of video watched on Youtube
- 10 million ads displayed
- 3.3 million shares on Facebook

DATA

1 internet minute

$ 400 M sales on **Alibaba**

439, 000 page views on **Wikipedia**

194, 000 apps downloaded

31, 700 hours of music played on **Pandora**

38, 000 photographs uploaded to **Instagram**

4.1 Million searches on **Google**

139, 000 hours of video watched on **Youtube**

10 million ads displayed

3.3 million shares on **Facebook**

Alibaba
Wikipedia
Pandora
Instagram
Google
Youtube
Facebook

These companies and others are collecting

PetaBytes of data every minute

PetaBytes of data every minute

What does this mean?

1 PetaByte ~ 1000 TeraBytes

1 PetaByte ~ 1000 TeraBytes

This is a 1 TB hard disk drive

1 PetaByte ~ 1000 TeraBytes

1000s of such
1 TB drives are
filled up
every minute by
data collected
on the web!!

Why are web companies collecting truckloads (literally) of data?

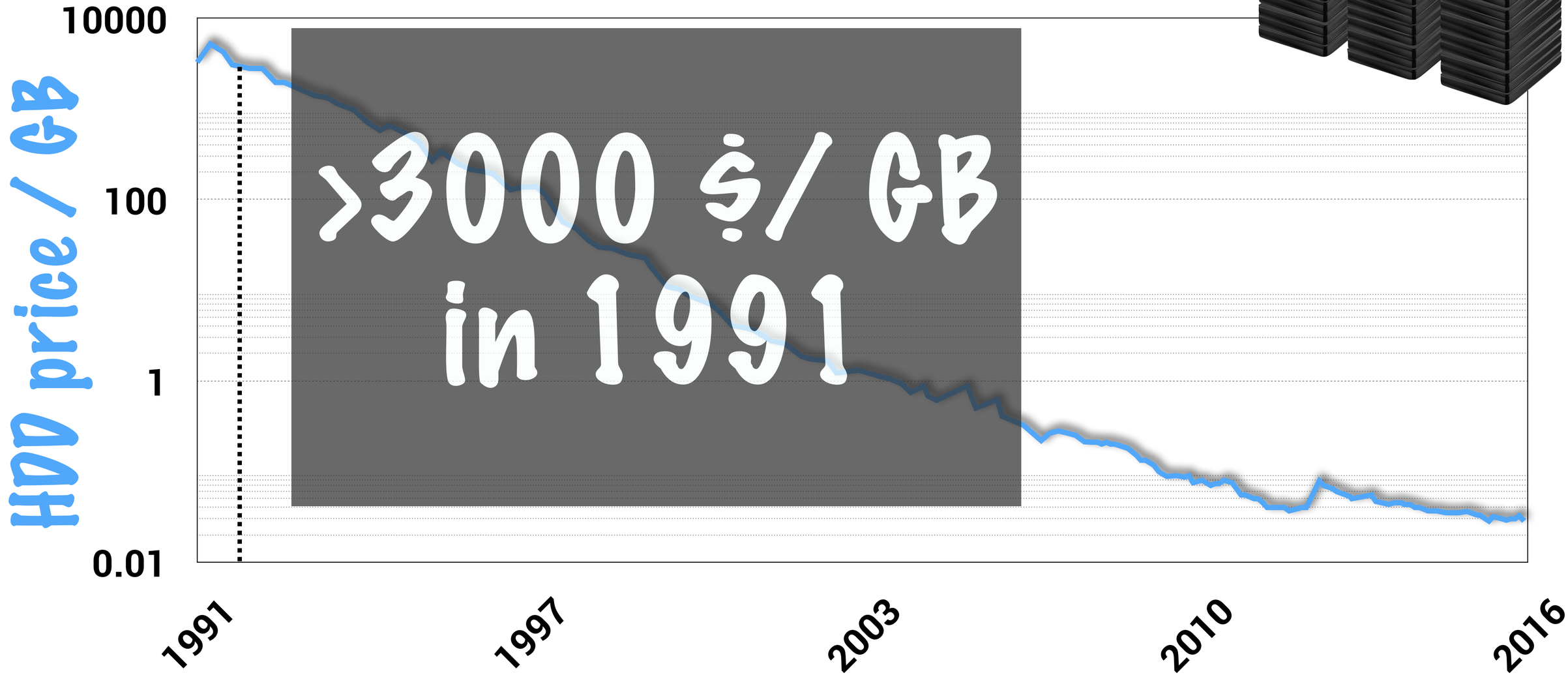
Reason # 1

Because they can afford it

Storage prices have dropped like crazy over the last 2 decades

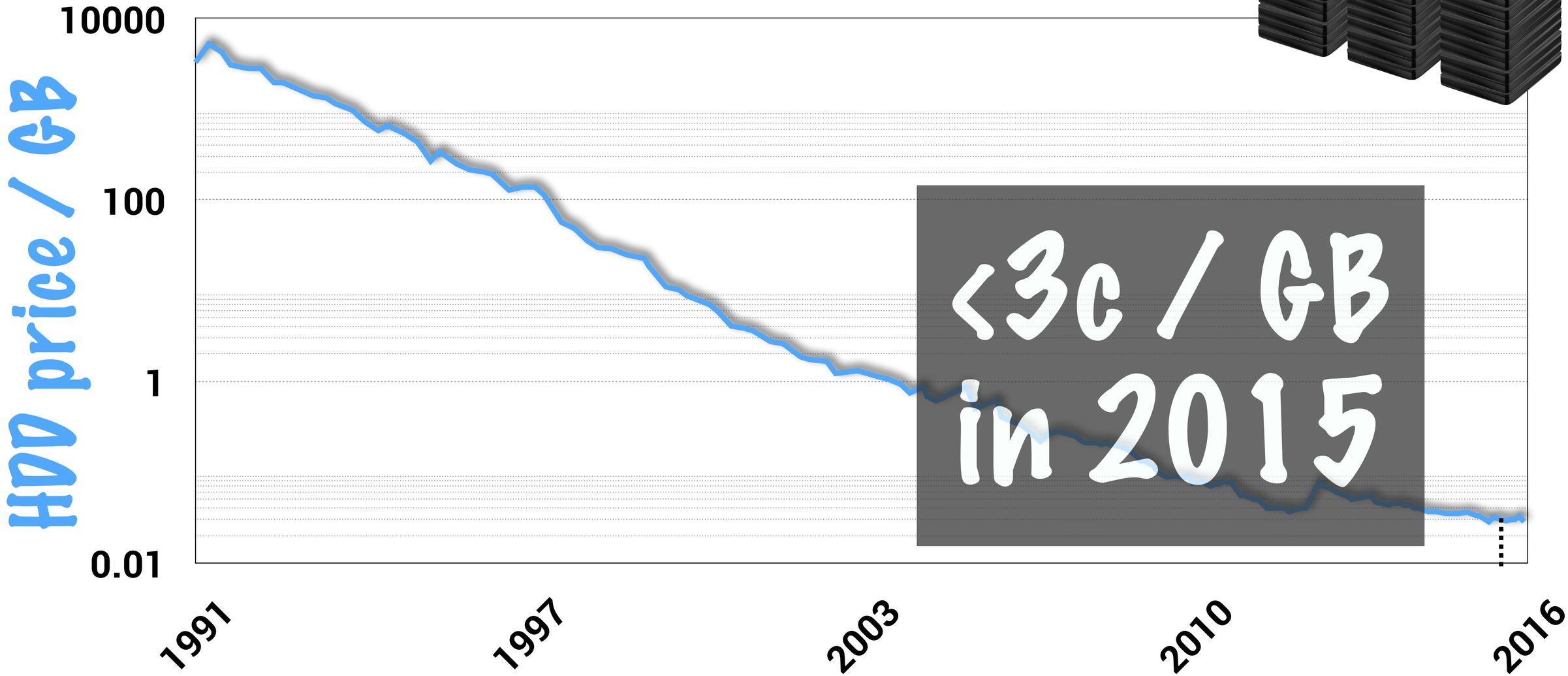
Reason # 1

Storage prices have dropped like crazy over the last 2 decades



Reason # 1

Storage prices have dropped like crazy over the last 2 decades



Reason # 2

Because they can monetize it

Large scale data can be processed to derive huge amounts of value

Reason # 2

Large scale data can be processed to derive huge amounts of value

Everything is personalized

Product Recommendations on Amazon,
Newsfeed on Facebook,
Homepage on Netflix

Ads, Offers, Promotions just for you!

Reason # 2

Large scale data can be processed to derive huge amounts of value

Really cool products can be built

Google Maps,
Apple Siri

Reason # 2

Large scale data can be processed to derive huge amounts of value

Other companies pay to mine the data

Twitter Firehose

Facebook Topic Data

How do we go from

Truckloads of data

Monetizable products

Recommendations
Newsfeed
Maps

Companies like

Google,
Apple,.
Amazon,
Facebook etc

own Huge Data Centers

Huge Data Centers

covering 100s of acres

with millions of servers

Huge Data Centers

with millions
of servers

running
sophisticated
proprietary
software

Huge Data Centers

with millions
of servers

running
sophisticated
proprietary
software

to process
TBs/PBs of
data

The Big Data Paradigm

Huge Data Centers

with lots of servers

running sophisticated proprietary software

to process TBs/PBs of data

The Big Data Paradigm

TBs/PBs of data

lots of servers

sophisticated proprietary software

There are only a handful of companies in the world that have all of the above

The Big Data Paradigm

TBs/PBs of data

lots of servers

sophisticated proprietary software

So, should the rest of us even care ?

TBs/PBs of data

lots of servers
sophisticated proprietary software

Even small companies with few 1000s of orders/visits per day collect

Terabytes of data over their lifetime

TBs/PBs of data

lots of servers

sophisticated proprietary software

Cool new technologies like Self-Driving cars and Internet of Things

will use sensors that generate huge amounts of data

TBs/PBs of data

lots of servers

sophisticated proprietary software

Because of cloud companies like
AWS, Microsoft Azure, GCP

Anyone can requisition 100s of servers
at a moment's notice

TBs/PBs of data

lots of servers

sophisticated proprietary software

Netflix, Pinterest, AirBnB

run their entire business just using cloud services like AWS

TBs/PBs of data

lots of servers

sophisticated proprietary software

Open Source Technologies

Hadoop, Spark, HBASE, Hive and many others

TBs/PBs of data

lots of servers

sophisticated proprietary software

Open Source Technologies

Hadoop, Spark, HBASE, Hive and many others

Highly sophisticated software that anyone can use to process Big Data

We just saw 3 major developments

TBs/PBs of data is generated everywhere

Physical/cloud infrastructure is now highly affordable

Sophisticated proprietary software

Open source technology that's available to all

The Big Data Paradigm

is here to stay!!

What kind of computing architecture does Big Data require?

You work for an e-commerce startup

You collect 1 TB worth of weblogs everyday

At the end of the day you want to publish a report on traffic for the day

Option # 1

Use a single powerful server

1 TB hard disk drive (minimum)

Option # 1

1 TB hard disk drive (minimum)

This approach has issues because of current limitations of hard disk technology

Transfer speed

Disk Size

Option # 1

1 TB hard disk drive (minimum)

Transfer speed		Disk Size
Max transfer speed of data for a 1 TB drive	~	100 MB/S

Option # 1

1 TB hard disk drive (minimum)

Transfer speed

Disk Size

Max transfer speed of data for a 1 TB drive ~ 100 MB/S

Time required to read the data ~ 10,000 s ~ 2.5 HRs

Option # 1

1 TB hard disk drive (minimum)

Transfer speed

Disk Size

Hard disk drive sizes
have an upper bound

~10 TB (in 2016)

Option # 2

Distribute the data on multiple servers

Option # 2

Distribute the data on multiple servers

This approach overcomes both the limitations of the single drive option

Transfer speed

Disk Size

Option # 2

Distribute the data on multiple servers

Transfer speed

Disk Size

We'll divide the data into
10 blocks of 100 GB each

Option # 2

Distribute the data on multiple servers

Transfer speed

Disk Size

10 blocks of 100 GB

Time required to read the data ~ 2.5 HRs /10

Option # 2

Distribute the data on multiple servers

Transfer speed

Disk Size

10 blocks of 100 GB

2.5 HRs/10

We have essentially parallelized the read task

Option # 2

Distribute the data on multiple servers

~~Transfer speed~~

Disk Size

Since the data is stored on multiple disks this is no longer a problem

Option # 2

Distribute the data on multiple servers

Transfer speed

Disk Size

Distributing the data has one more advantage

Option # 2

Distribute the data on multiple servers

Since each disk only has a small amount of the data, you can use smaller, cheaper processors to read them

Option # 2

Distribute the data on multiple servers

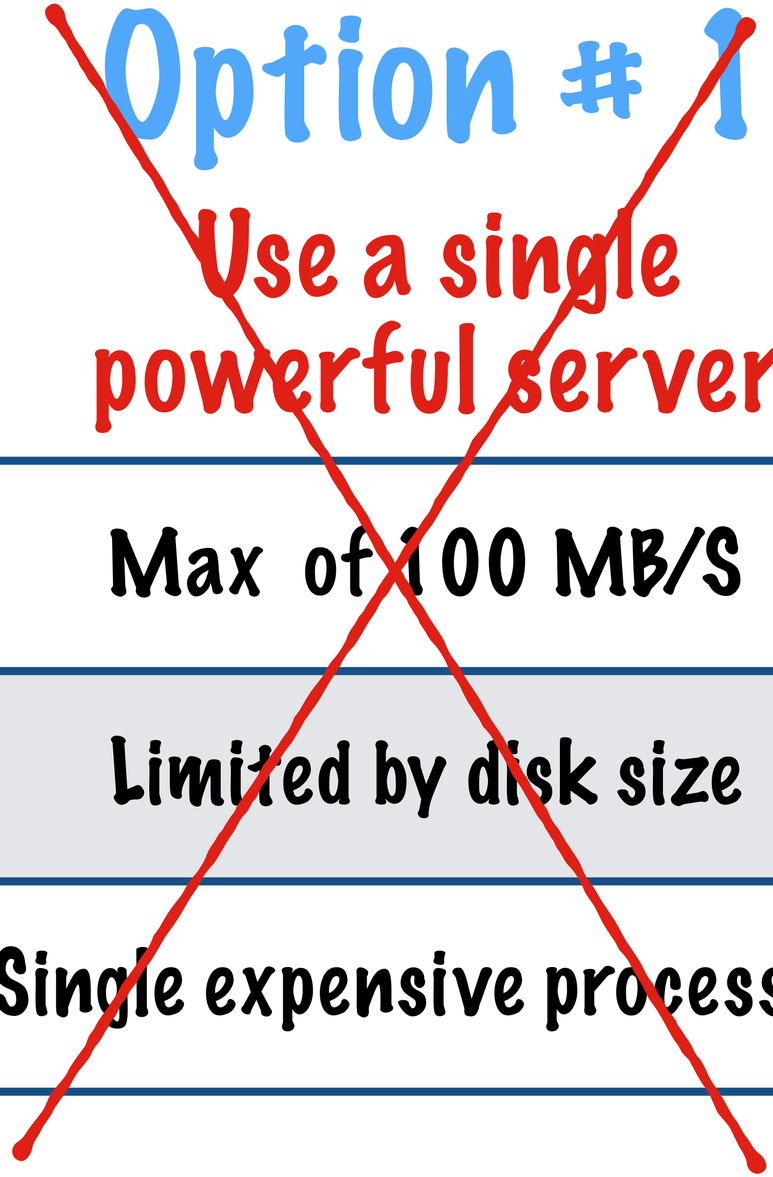
Since each disk only has a small amount of the data, you can use smaller, cheaper processors to read them

Option # 2

Distribute the data on multiple servers

Since each disk only has a small amount of the data, **you can use smaller, cheaper processors to read them**

	Option # 1 Use a single powerful server	Option # 2 Distribute the data on multiple servers
Transfer Speed	Max of 100 MB/S	> 100MB/S (due to parallelization)
Data Size	Limited by disk size	Unlimited
Processor Cost	Single expensive processor	Multiple Cheap processors



How do we distribute and process data on multiple servers?

HADOOP

is a distributed computing framework
developed and maintained by

THE APACHE SOFTWARE FOUNDATION

written in Java

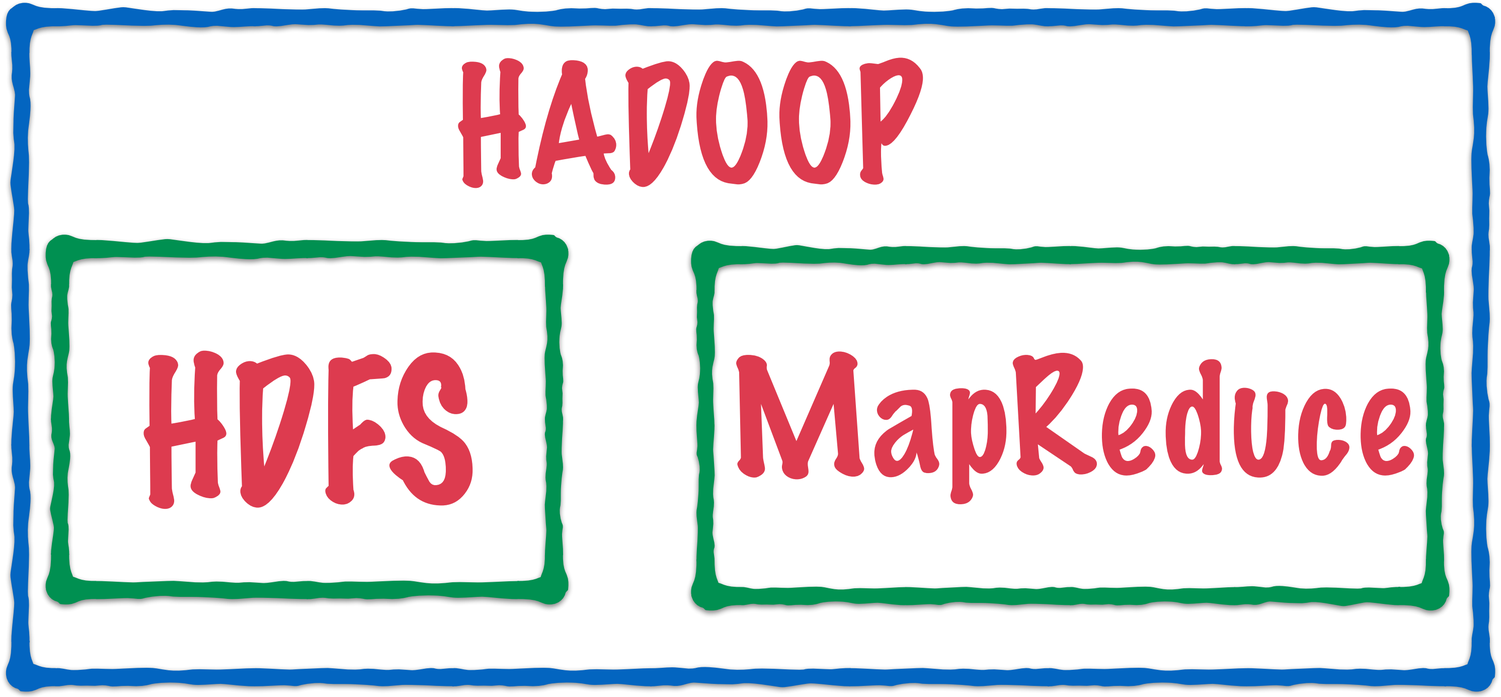
HADOOP

HDFS

A file system to manage the storage of data

MapReduce

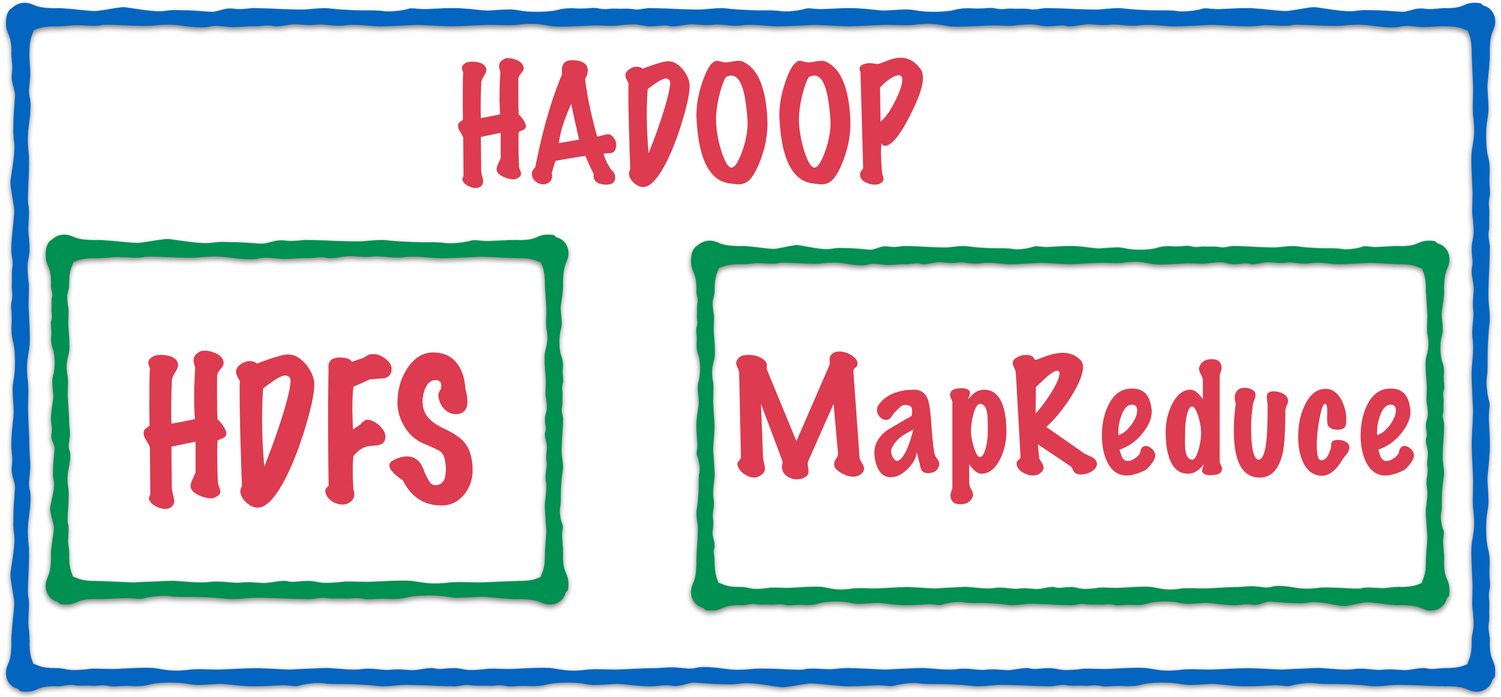
A framework to process data across multiple servers



Hadoop is an open source implementation of 2 proprietary technologies by Google

GFS

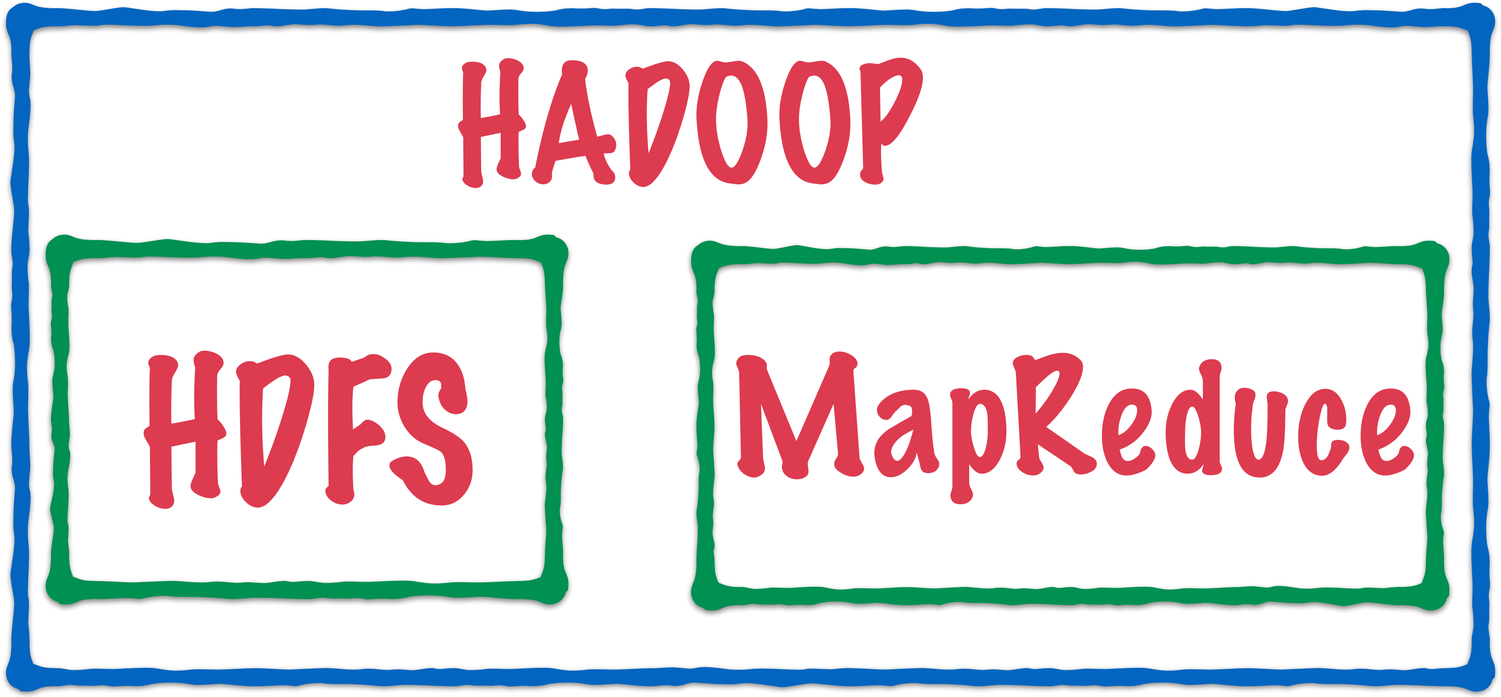
MapReduce



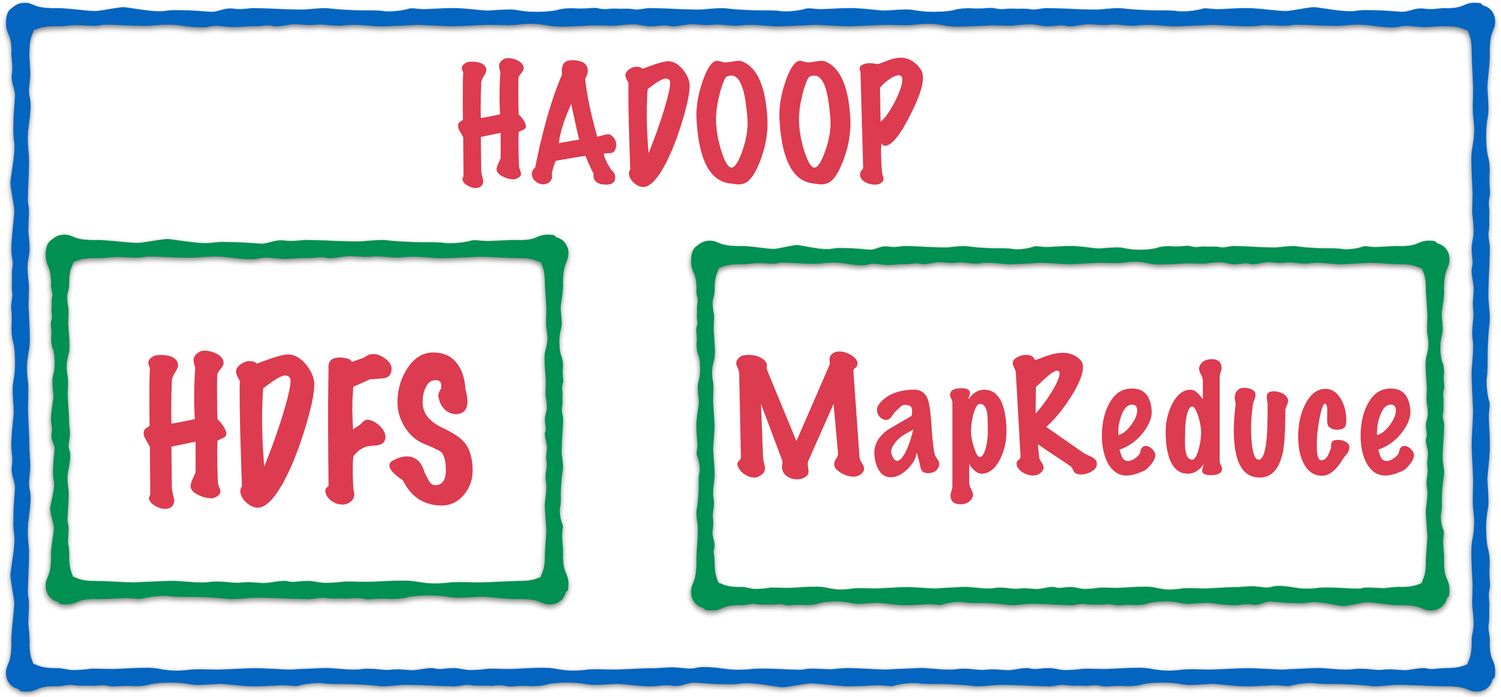
GFS

MapReduce

These are technologies originally built to power Google Search



In 2014, Apache released a new version of Hadoop



In Hadoop 2.0 the MapReduce block was broken into 2 parts

HADOOP

HDFS

MapReduce

A framework to process data across multiple servers

HADOOP

HDFS

YARN

A framework to run the data processing task

MapReduce

A framework to define a data processing task

HADOOP

HDFS

YARN

MapReduce

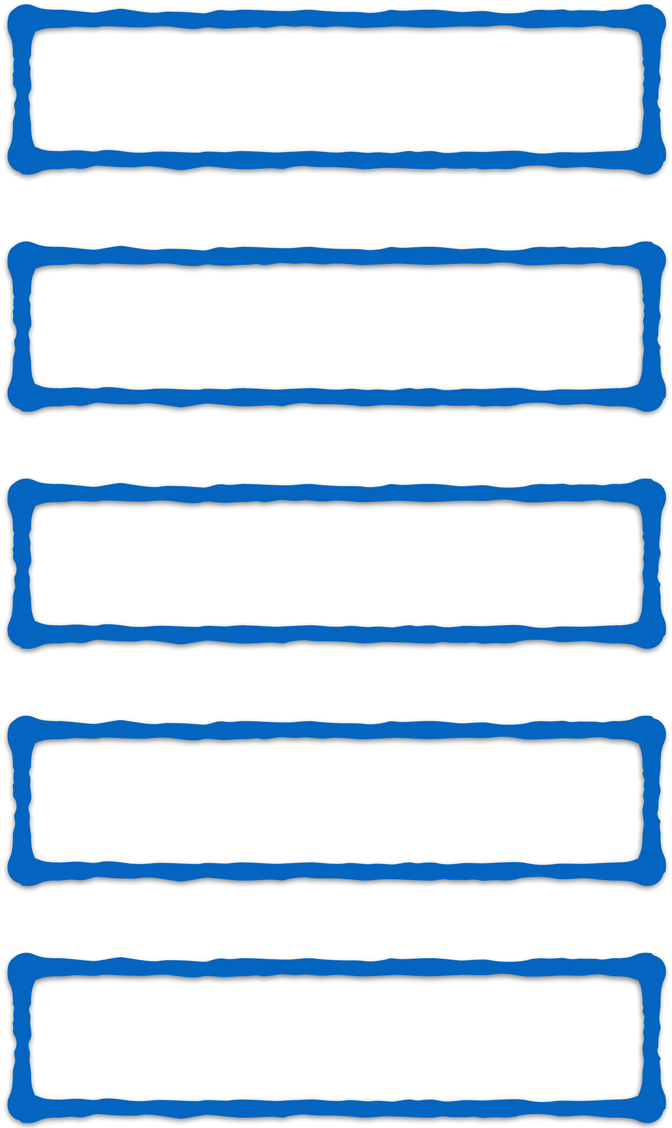
Let's go through each of these blocks 1 by 1

HDFS

The Hadoop Distributed File System

Hadoop uses this to store data across multiple disks

HDFS

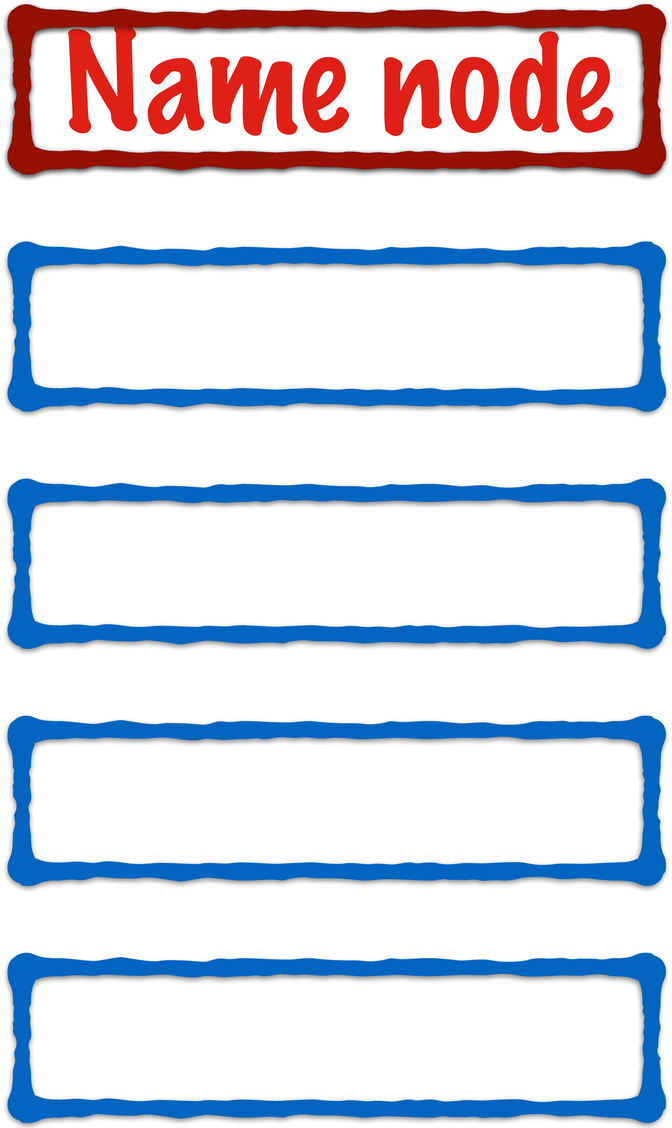


Hadoop is normally deployed on a group of machines

Cluster

Each machine in the cluster is a node

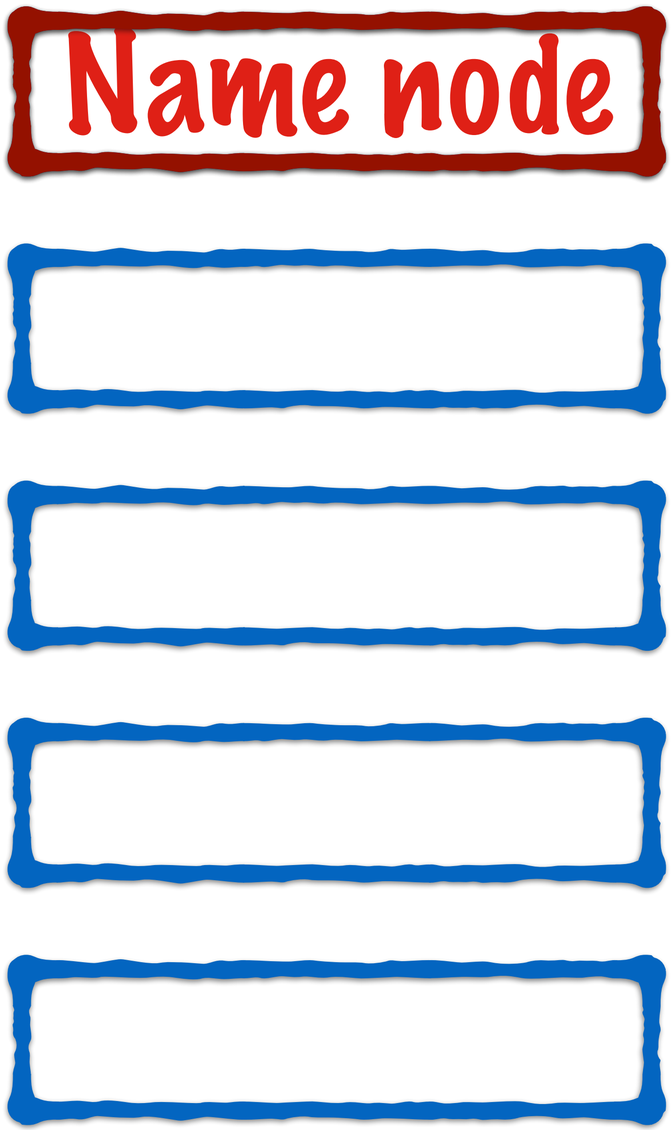
HDFS



One of the nodes acts as the **master node**

This node **manages** the overall file system

HDFS

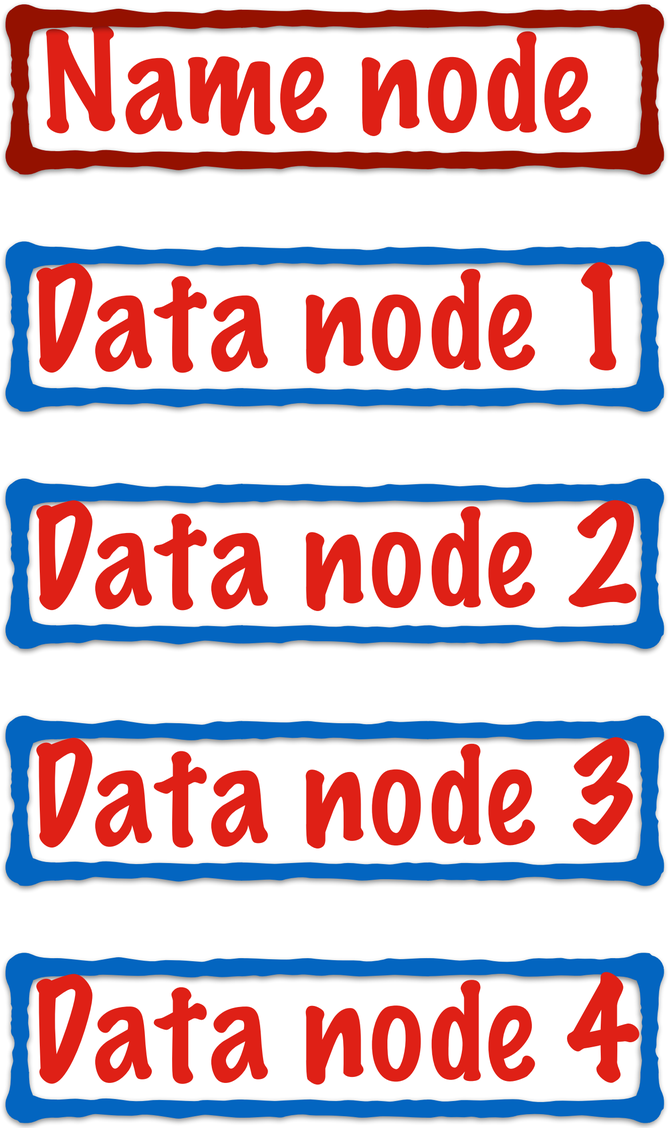


The name node stores

1. The directory structure

2. Metadata for all the files

HDFS



Other nodes are called data nodes

The data is physically stored on these nodes

HDFS

Here is a large text file

next up previous contents index
Next: Dynamic indexing Up: Index construction Previous: Single-pass in-memory indexing Contents Index

Distributed indexing

Collections are often so large that we cannot perform index construction efficiently on a single machine. This is particularly true of the World Wide Web for which we need large computer clusters [*]to construct any reasonably sized web index. Web search engines, therefore, use distributed indexing algorithms for index construction. The result of the construction process is a distributed index that is partitioned across several machines - either according to term or according to document. In this section, we describe distributed indexing for a
term-partitioned index . Most large search engines prefer a document-partitioned index (which can be easily generated from a term-partitioned index). We discuss this topic further in Section 20.3 (page [*]).

The distributed index construction method we describe in this section is an application of MapReduce , a general architecture for distributed computing. MapReduce is designed for large computer clusters. The point of a cluster is to solve large computing problems on cheap commodity machines or nodes that are built from standard parts (processor, memory, disk) as opposed to on a supercomputer with specialized hardware. Although hundreds or thousands of machines are available in such clusters, individual machines can fail at any time. One requirement for robust distributed indexing is, therefore, that we divide the work up into chunks that we can easily assign and - in case of failure - reassign. A master node directs the process of assigning and reassigning tasks to individual worker nodes.

The map and reduce phases of MapReduce split up the computing job into chunks that standard machines can process in a short time. The various steps of MapReduce are shown in Figure 4.5 and an example on a collection consisting of two documents is shown in Figure 4.6 . First, the input data, in our case a collection of web pages, are split into n splits where the size of the split is chosen to ensure that the work can be distributed evenly (chunks should not be too large) and efficiently (the total number of chunks we need to manage should not be too large); 16 or 64 MB are good sizes in distributed indexing. Splits are not preassigned to machines, but are instead assigned by the master node on an ongoing basis: As a machine finishes processing one split, it is assigned the next one. If a machine dies or becomes a laggard due to hardware problems, the split it is working on is simply reassigned to another machine.

Figure 4.5: An example of distributed indexing with MapReduce. Adapted from Dean and Ghemawat (2004).
\includegraphics[width=11.5cm]{art/mapreduce2.eps}
In general, MapReduce breaks a large computing problem into smaller parts by recasting it in terms of manipulation of key-value pairs . For indexing, a key-value pair has the form (termID,docID). In distributed indexing, the mapping from terms to termIDs is also distributed and therefore more complex than in single-machine indexing. A simple solution is to maintain a (perhaps precomputed) mapping for frequent terms that is copied to all nodes and to use terms directly (instead of termIDs) for infrequent terms. We do not address this problem here and assume that all nodes share a consistent term $\rightarrow$ termID mapping.

The map phase of MapReduce consists of mapping splits of the input data to key-value pairs. This is the same parsing task we also encountered in BSBI and SPIMI, and we therefore call the machines that execute the map phase parsers . Each parser writes its output to local intermediate files, the segment files (shown as \fbox{a-f\medstrut} \fbox{g-p\medstrut} \fbox{q-z\medstrut} in Figure 4.5).

For the reduce phase , we want all values for a given key to be stored close together, so that they can be read and processed quickly. This is achieved by partitioning the keys into j term partitions and having the parsers write key-value pairs for each term partition into a separate segment file. In Figure 4.5 , the term partitions are according to first letter: a-f, g-p, q-z, and $j=3$. (We chose these key ranges for ease of exposition. In general, key ranges need not correspond to contiguous terms or termIDs.) The term partitions are defined by the person who operates the indexing system (Exercise 4.6). The parsers then write corresponding segment files, one for each term partition. Each term partition thus corresponds to r segments files, where r is the number of parsers. For instance, Figure 4.5 shows three a-f segment files of the a-f partition, corresponding to the three parsers shown in the figure.

Collecting all values (here: docIDs) for a given key (here: termID) into one list is the task of the inverters in the reduce phase. The

Let's see how this file is stored in HDFS

HDFS

Block 1

Block 2

Block 3

Block 4

Block 5

Block 6

Block 7

Block 8

First the file is broken into blocks of size 128 MB

HDFS

Block 1

Block 2

Block 3

Block 4

Block 5

Block 6

Block 7

Block 8

First the file is broken into

blocks of size 128 MB

This size is chosen to minimize the time to seek to the block on the disk

HDFS

Block 1

Block 2

Block 3

Block 4

Block 5

Block 6

Block 7

Block 8

These blocks are then stored across the data nodes

HDFS

Data node 1

Block 1 | Block 2

Data node 3

Block 5 | Block 6

Data node 2

Block 3 | Block 4

Data node 4

Block 7 | Block 8

Name node

The name node stores metadata

HDFS

Block locations for each file are stored in the name node

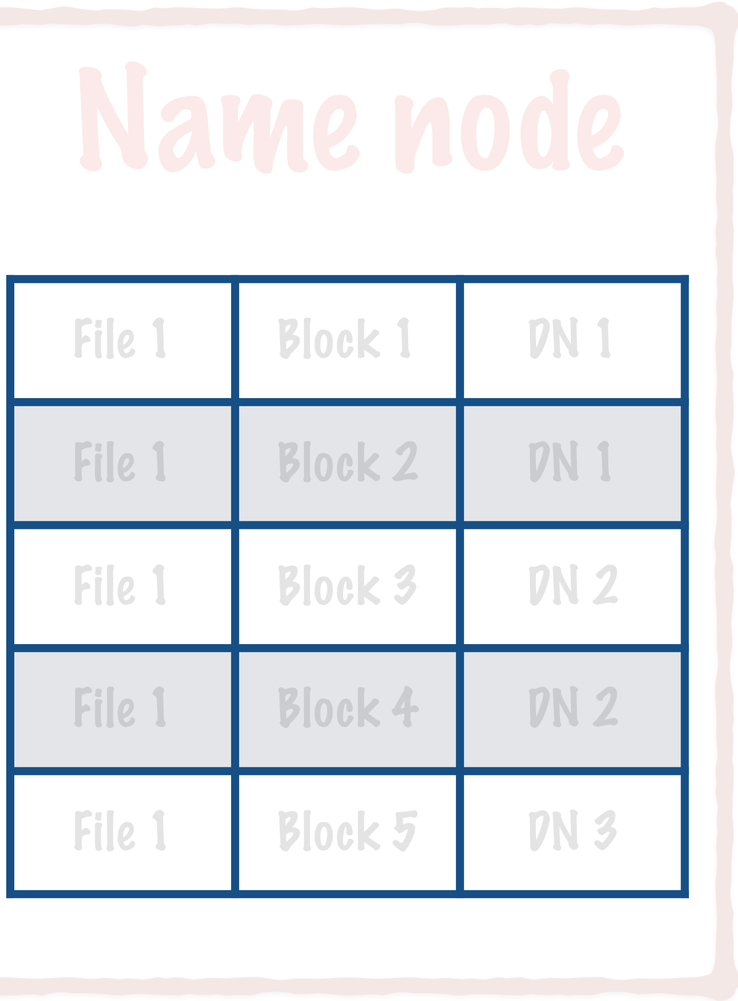
Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3

HDFS

A file is read using

1. The **metadata** in name node
2. The **blocks** in the data nodes



HDFS

Data node 1

Block 1 | Block 2

Data node 3

Block 5 | Block 6

Data node 2

Data node 4

Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3

What if one of the blocks gets corrupted?

HDFS

Data node 1

Block 1 | Block 2

Data node 3

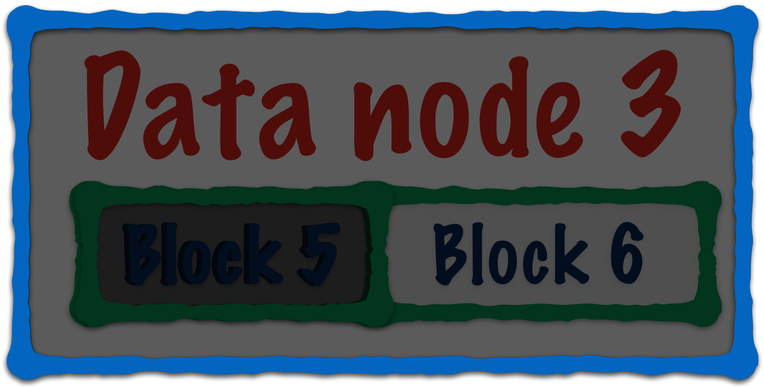
Block 5 | Block 6

Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3

Or one of the data nodes crashes?

HDFS



This is one of the key challenges in distributed storage

Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3

HDFS

Data node 1

Block 1

Data node 3

Block 6

Data node 2

Block 3

Block 4

Data node 4

Block 7

Block 8

Name node

File 1	Block 1	DN 1
File 1	Block 2	DN 1
File 1	Block 3	DN 2
File 1	Block 4	DN 2
File 1	Block 5	DN 3

You can define a replication factor in HDFS

HDFS

Data node 1

Block 1 | Block 2

Data node 2

Block 3 | Block 4

Block 1 | Block 2

Data node 3

Block 5 | Block 6

Name node

Each block is replicated, and the replicas are stored in different data nodes

HDFS

Data node 1

Block 1 | Block 2

Data node 3

Block 5 | Block 6

Data node 2

Data node 4

Name node

File 1	Block 1	Master	DN 1
File 1	Block 1	Replica	DN 2
..	..	..	..
..	..	..	..
..	..	..	..

The replica locations are also stored in the name node

HADOOP

HDFS

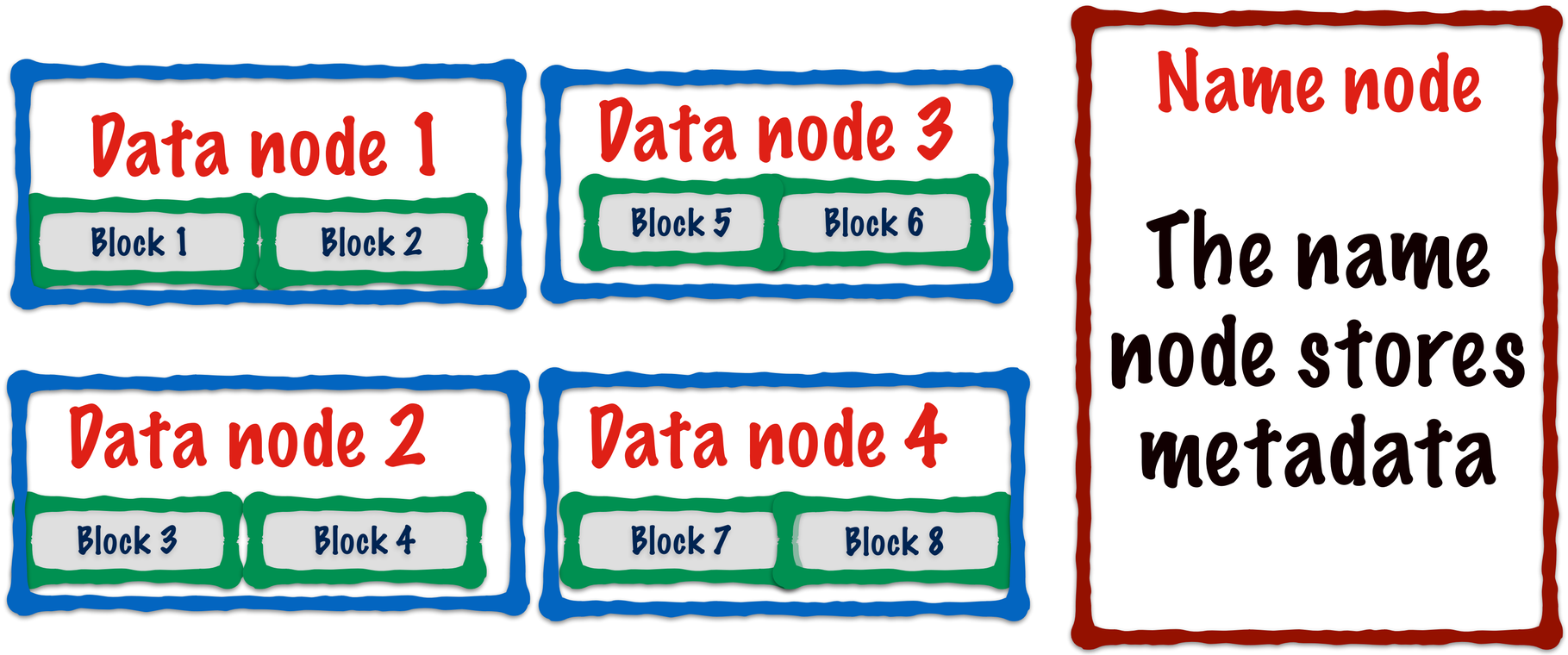
YARN

MapReduce

Let's go through each of these blocks superficially

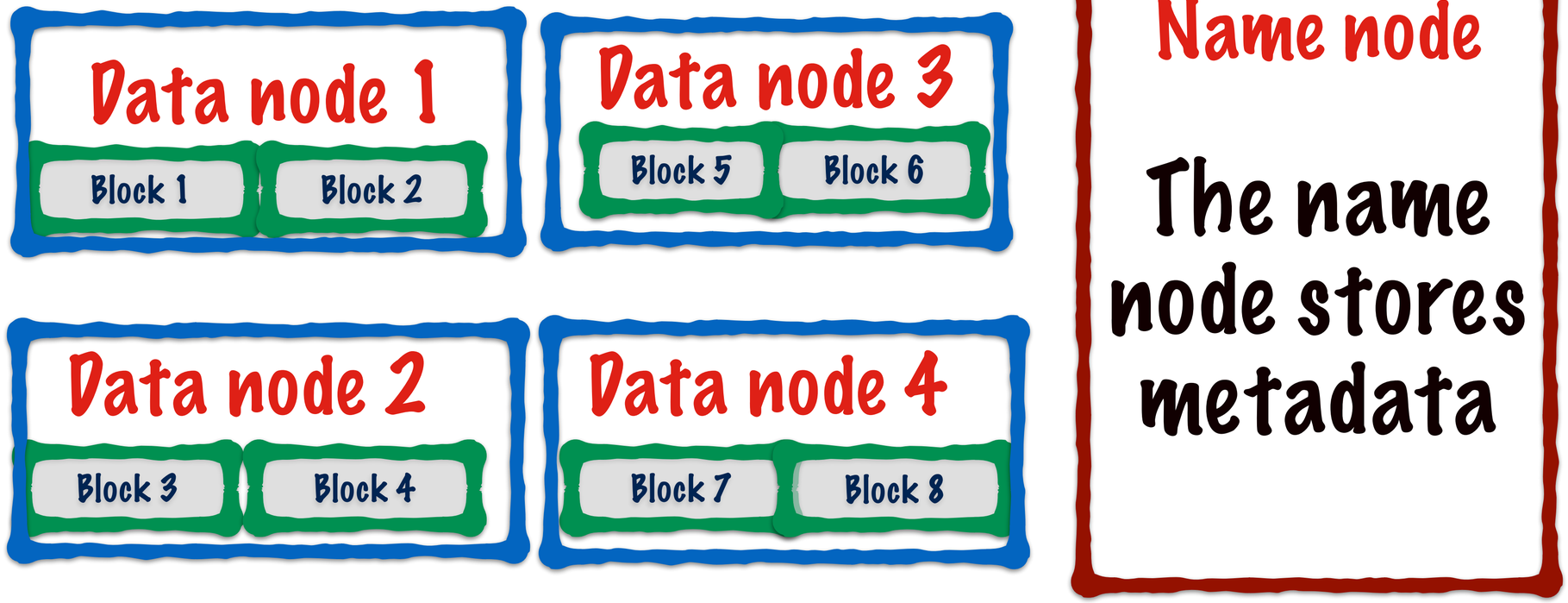
MapReduce

We've stored a text file in HDFS



MapReduce

The file consists of blocks stored on different nodes



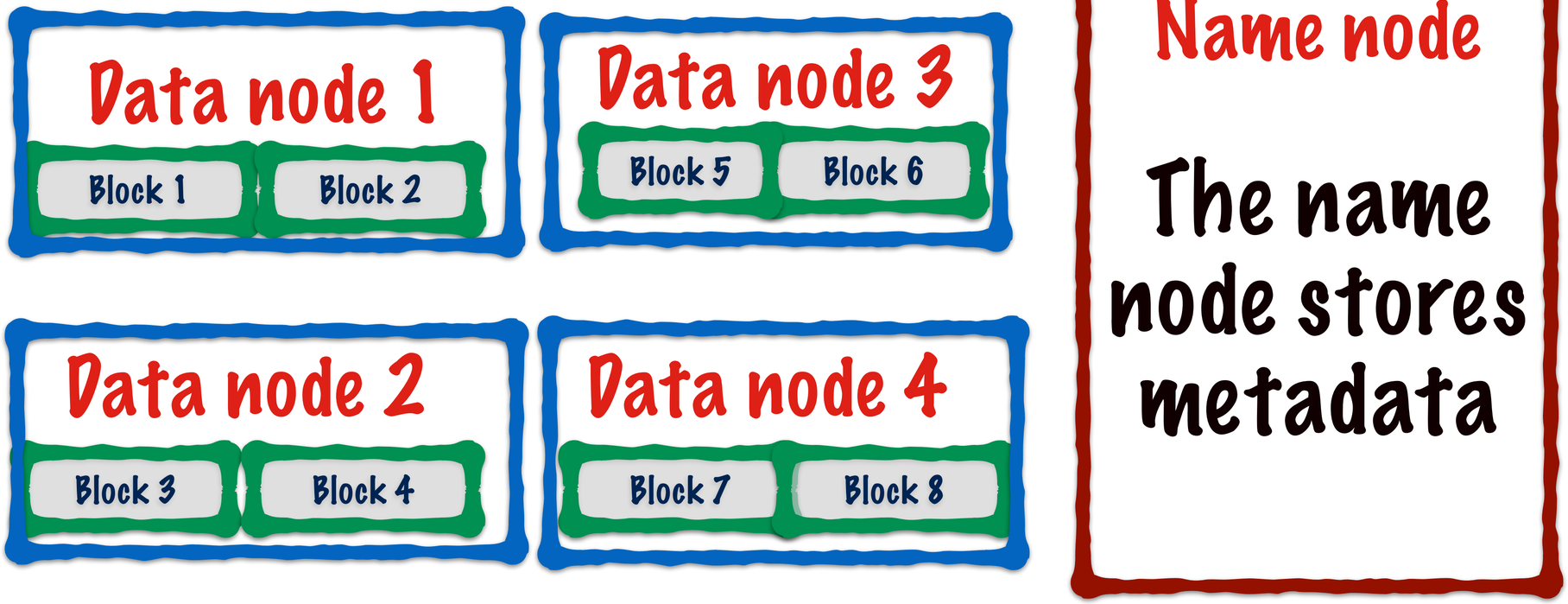
MapReduce

Objective: To find the number of times the word "hello" occurs in the file

MapReduce

Option 1:

1. Reconstruct the complete file in 1 node

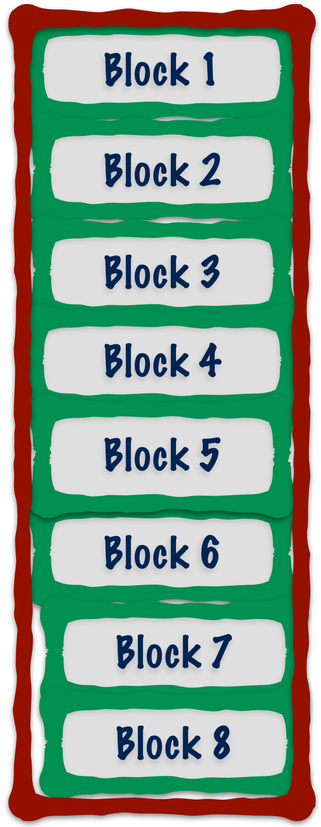


MapReduce

Option 1:

1. Reconstruct the complete file in 1 node

2. Process the file to count # of "hello"s



MapReduce

Option 1:

1. Reconstruct the complete file in 1 node
2. Process the file to count # of "hello"s

Pro: This program is relatively simple to write

MapReduce

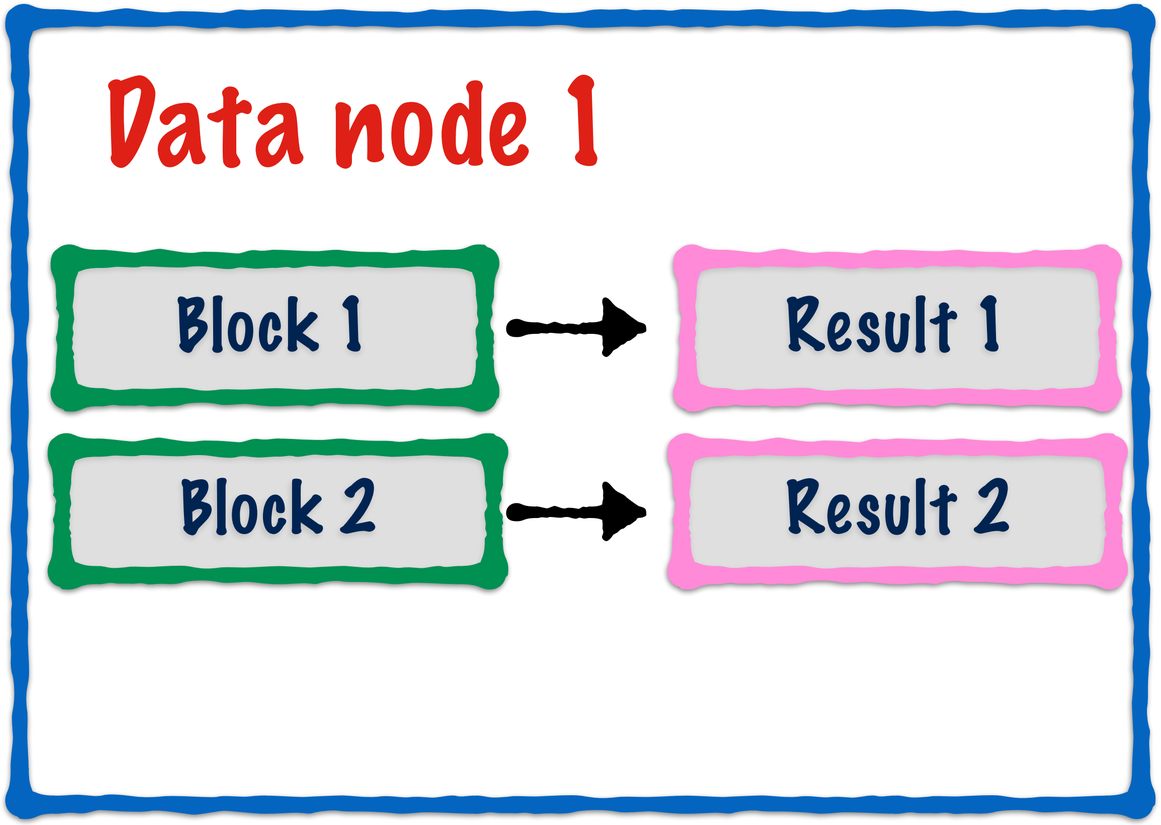
Option 1:

1. Reconstruct the complete file in 1 node
2. Process the file to count # of "hello"s

Con: We are not taking advantage of the parallelism inherent in Hadoop

MapReduce Option 2:

1. Process each block in the node it is stored in

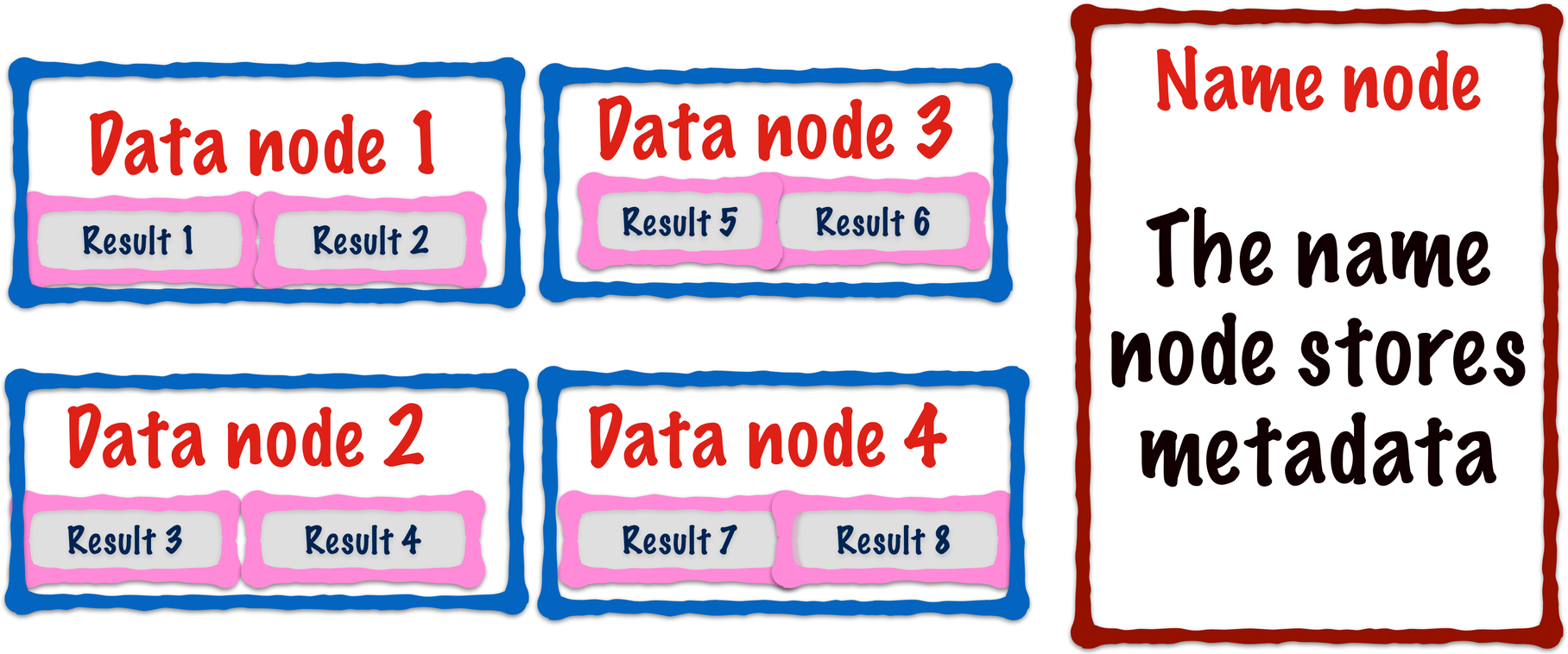


Each result is the # "hello"s in the block

MapReduce

Option 2:

2. Take all the results to one node and combine them



MapReduce

Option 2:

2. Take all the results to one node and combine them

The answer is simply the sum of all the results

MapReduce

1. Process each block in the node it is stored in
2. Take all the results to one node and combine them

This is exactly the idea behind MapReduce

MapReduce

1. Process each block in the node it is stored in
2. Take all the results to one node and combine them

MapReduce is a way to parallelize a data processing task

MapReduce

1. Process each block in the node it is stored in
2. Take all the results to one node and combine them

MapReduce tasks have 2 phases

MapReduce

1. Process each block in the node it is stored in

2. Take all the results to one node and combine them

Map phase

MapReduce

1. Process each block in the node it is stored in

2. Take all the results to one node and combine them

Reduce phase

MapReduce

Distributed computing
can get very complicated

Managing resources and
memory across multiple nodes

MapReduce

Distributed computing can get very complicated

What to do if a node goes down?

MapReduce

MapReduce abstracts the programmer from all these complications

MapReduce

You just define 2 functions

map() reduce()

MapReduce

map() reduce()

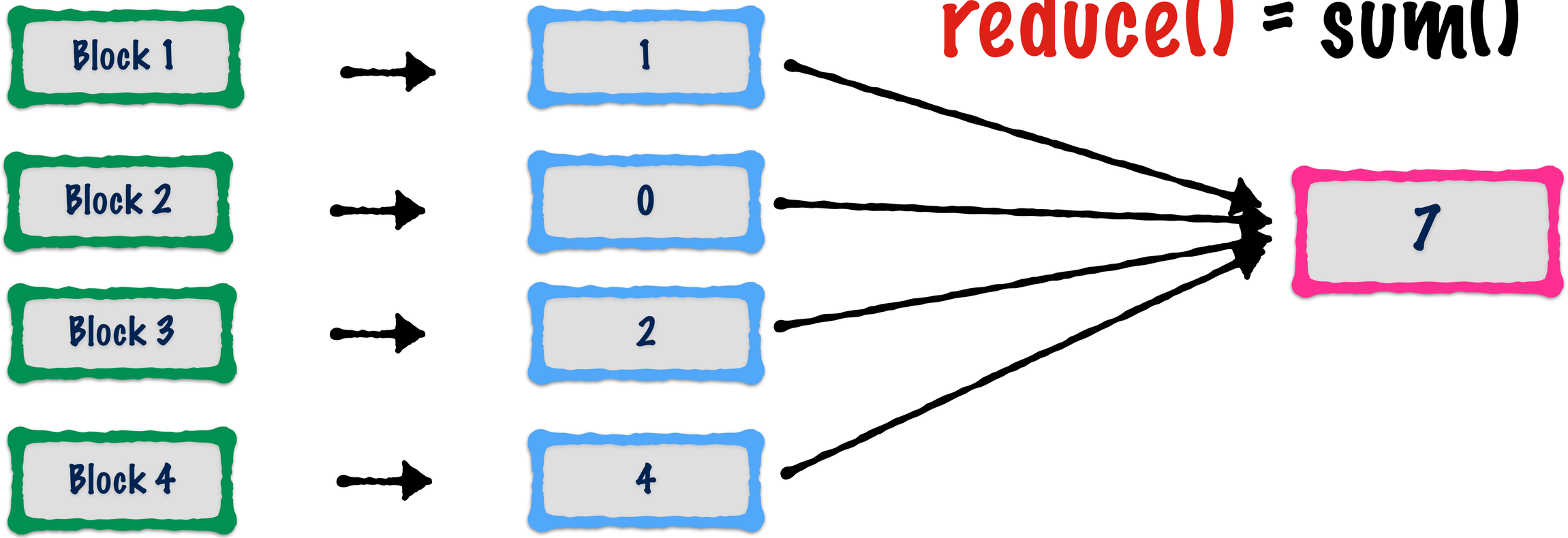
The rest is taken care of by Hadoop!

MapReduce

In our example : counting # of "hello"s

map() = count "hello"s

reduce() = sum()



MapReduce

There are more nuances
we'll get into later

But for now the takeaway is,

you can take advantage of the
parallelization in Hadoop by expressing
any task as map + reduce tasks

HADOOP

HDFS

YARN

MapReduce

Let's go through each of these blocks superficially

YARN

Yet Another Resource Negotiator

YARN was introduced in Hadoop 2.0 to separately handle

the management of resources on the Hadoop cluster

YARN

Yet Another Resource Negotiator

YARN co-ordinates all the different MapReduce tasks running on the cluster

YARN

Yet Another Resource Negotiator

YARN also monitors for failures and assigns new nodes when others fail

YARN

Yet Another Resource Negotiator

We'll dig deeper into YARN once we have run some MapReduce jobs

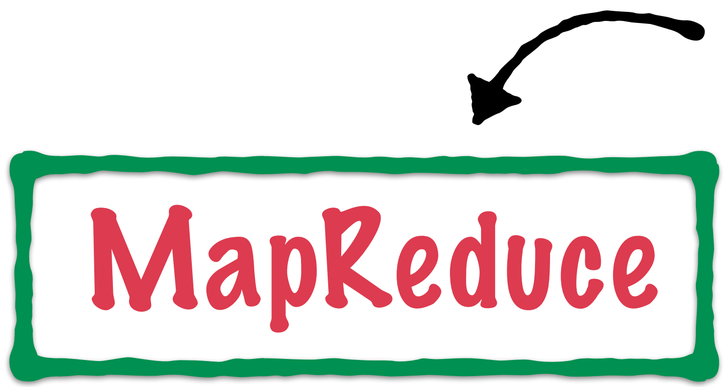
HADOOP

HDFS

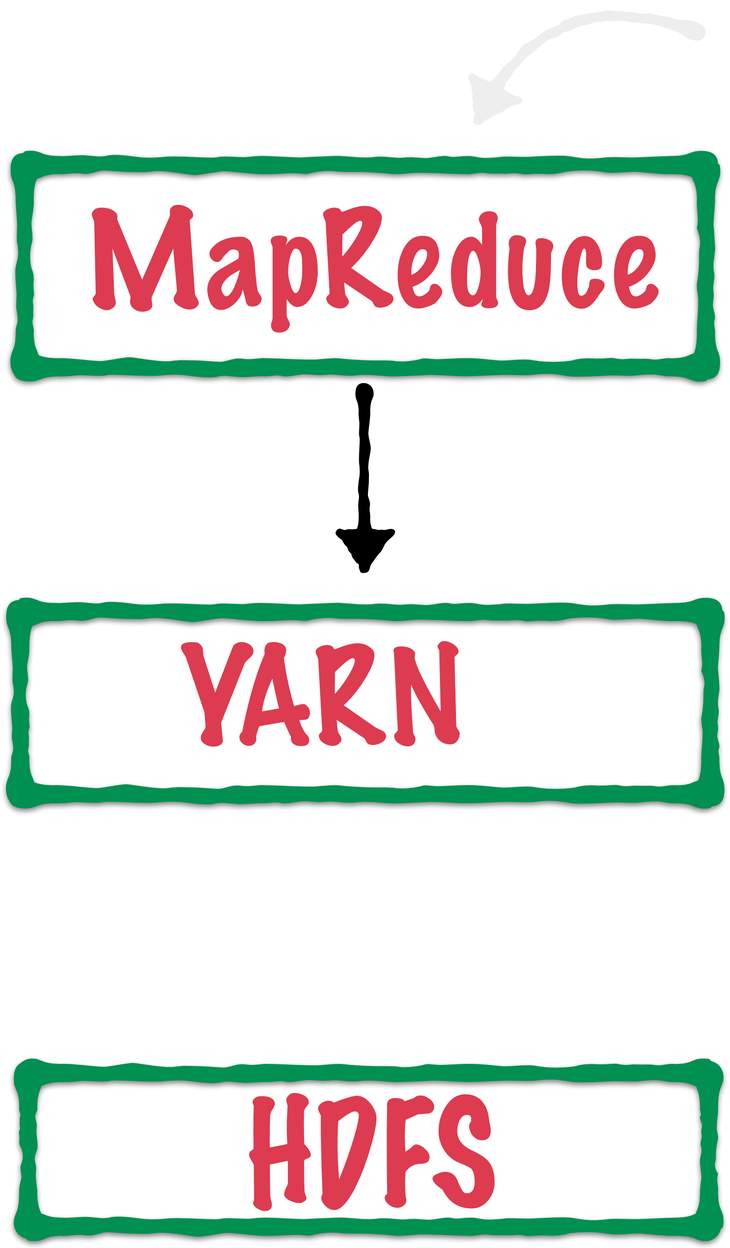
YARN

MapReduce

Here's how these different blocks communicate

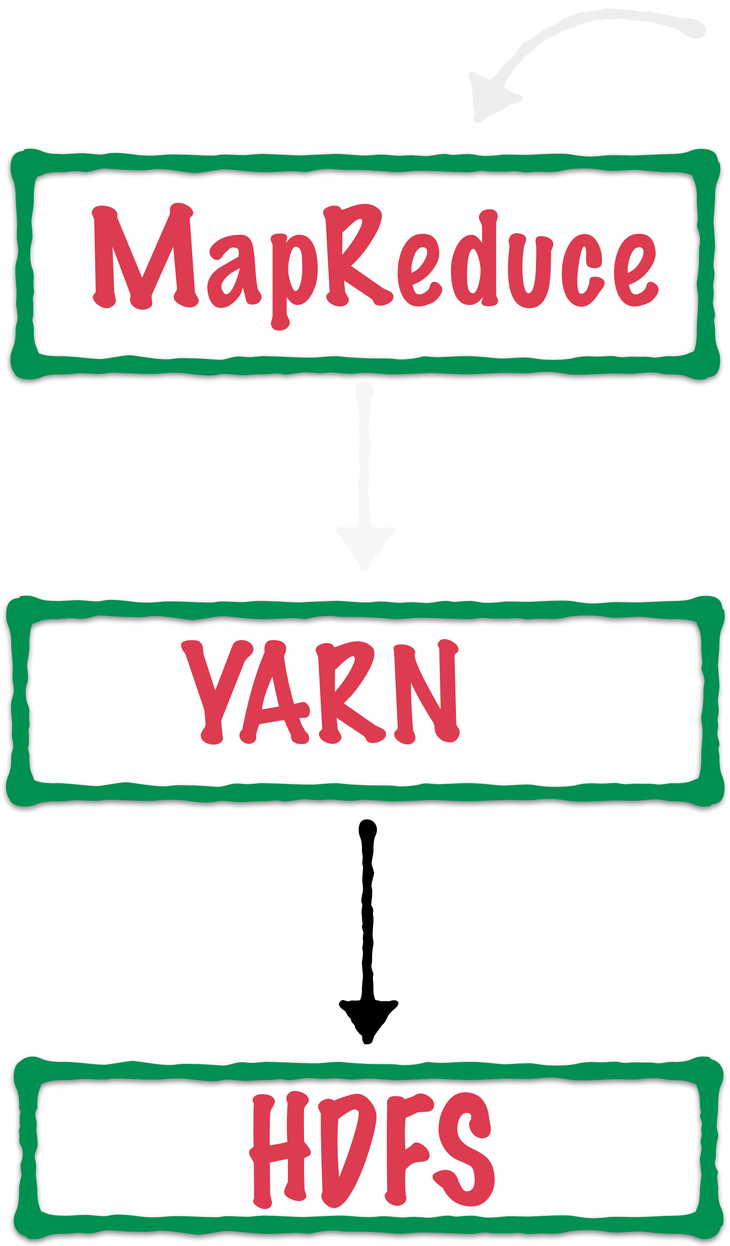


User defines map and reduce tasks using the MapReduce API



User defines map and reduce tasks using the MapReduce API

A job is triggered on the cluster



User defines map and reduce tasks using the MapReduce API

A job is triggered on the cluster

YARN figures out where and how to run the job, and stores the result in HDFS

Installing Hadoop

Installing Hadoop

Hadoop runs on
Linux/Unix based
Systems

Installing Hadoop

There are 3 different modes in which Hadoop can be installed and run

Installing Hadoop

3 different modes

Standalone
Pseudo-distributed
Fully-distributed

Installing Hadoop

Standalone

In Standalone mode, Hadoop runs with a single node

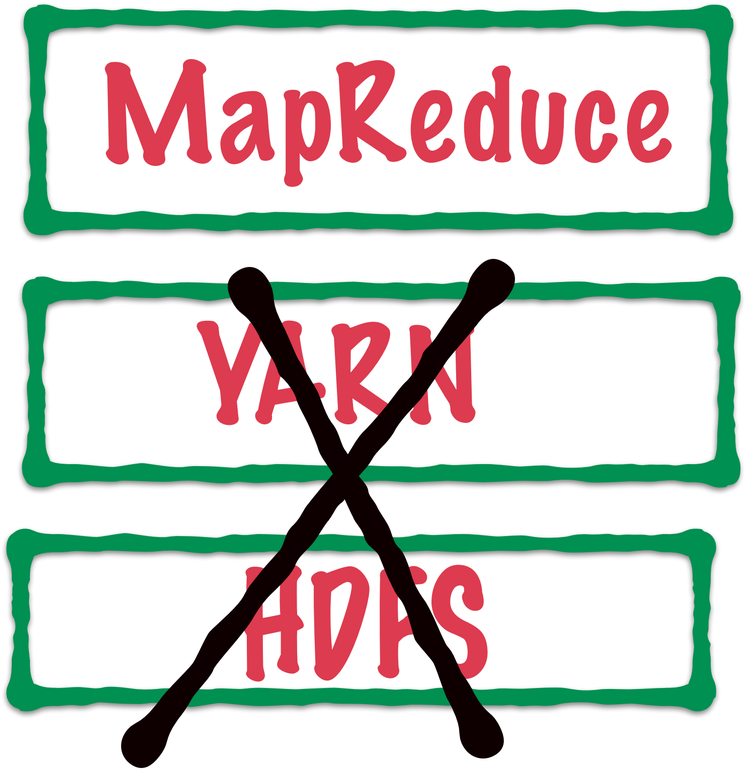
Installing Hadoop

Standalone

In this mode, both HDFS and YARN do not run

Installing Hadoop

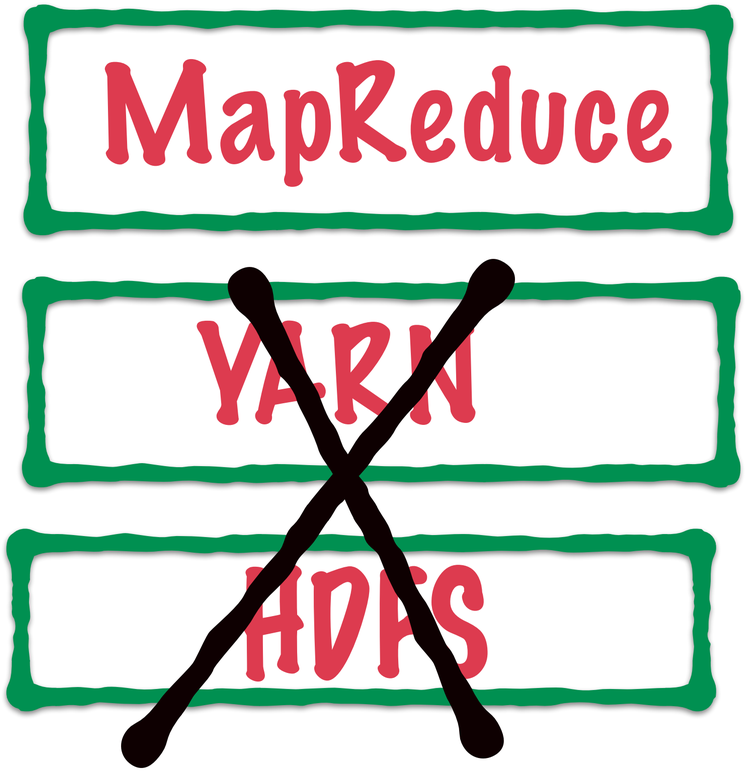
Standalone



Of the 3 components of Hadoop, only MapReduce runs in this mode

Installing Hadoop

Standalone



This mode is used to test MapReduce programs before running them on a cluster

Installing Hadoop

3 different modes

✓Standalone

Pseudo-distributed

Fully-distributed

Installing Hadoop

Pseudo-distributed

In this mode Hadoop is run on a single machine with 2 JVMs

Installing Hadoop

Pseudo-distributed

2 JVMs

One acts as a name node and other as a data node

Installing Hadoop

Pseudo-distributed

All 3 components of Hadoop run in this mode

- MapReduce
- YARN
- HDFS

Installing Hadoop

Pseudo-distributed

The pseudo-distributed mode can be used as a fully-fledged test setup

Installing Hadoop

3 different modes

✓ Standalone

✓ Pseudo-distributed

Fully-distributed

Installing Hadoop

Fully-distributed

This is a full-fledged setup with Hadoop running on a cluster of machines

Installing Hadoop

Fully-distributed

The cluster can be made up of

1. Linux Servers in a data center
2. Machines requisitioned on a Cloud service like AWS/Azure

Installing Hadoop

Fully-distributed

Manually configuring and managing nodes in a Hadoop cluster is complicated and error-prone

Installing Hadoop

Fully-distributed

Enterprise editions of Hadoop

Cloudera, Hortonworks, MapR

make it easier to manage large Hadoop clusters

Installing Hadoop

Insert Hadoop
install Standalone
and Pseudo