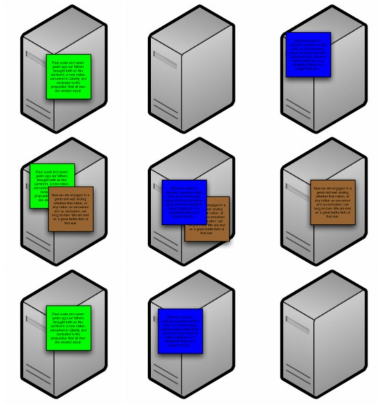
Distributed Storage

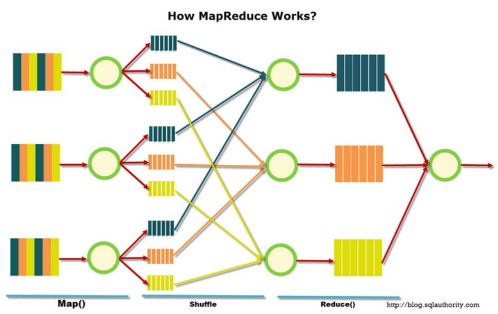
[Hadoop & Friends]

INTERNET & DATA LAB

Agenda for today



Distributed storage
- Moving beyond classical storage
- HDFS as a use case



Distributed processing
- MapReduce paradigm
- Improvements via Spark

Algorithmic complexity

Bubble sort, quick sort, Radix sort...

Which operations should we count?

- multiply / divide
- add / subtract

Nope!

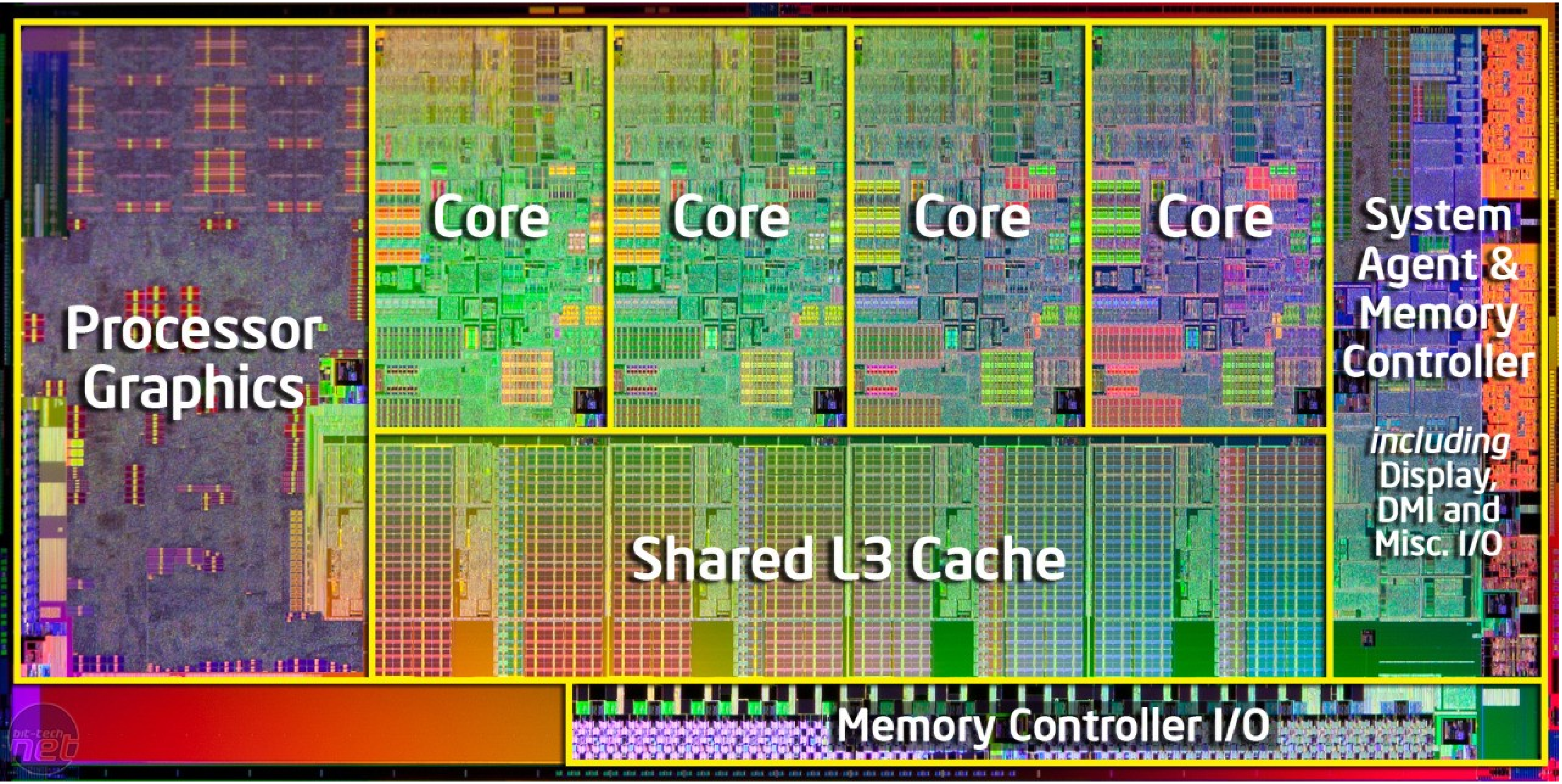
- page fault
- cache miss
- memory access
- disk access (swap space)
- network fetch
- communication between workers

$O(n^2)$

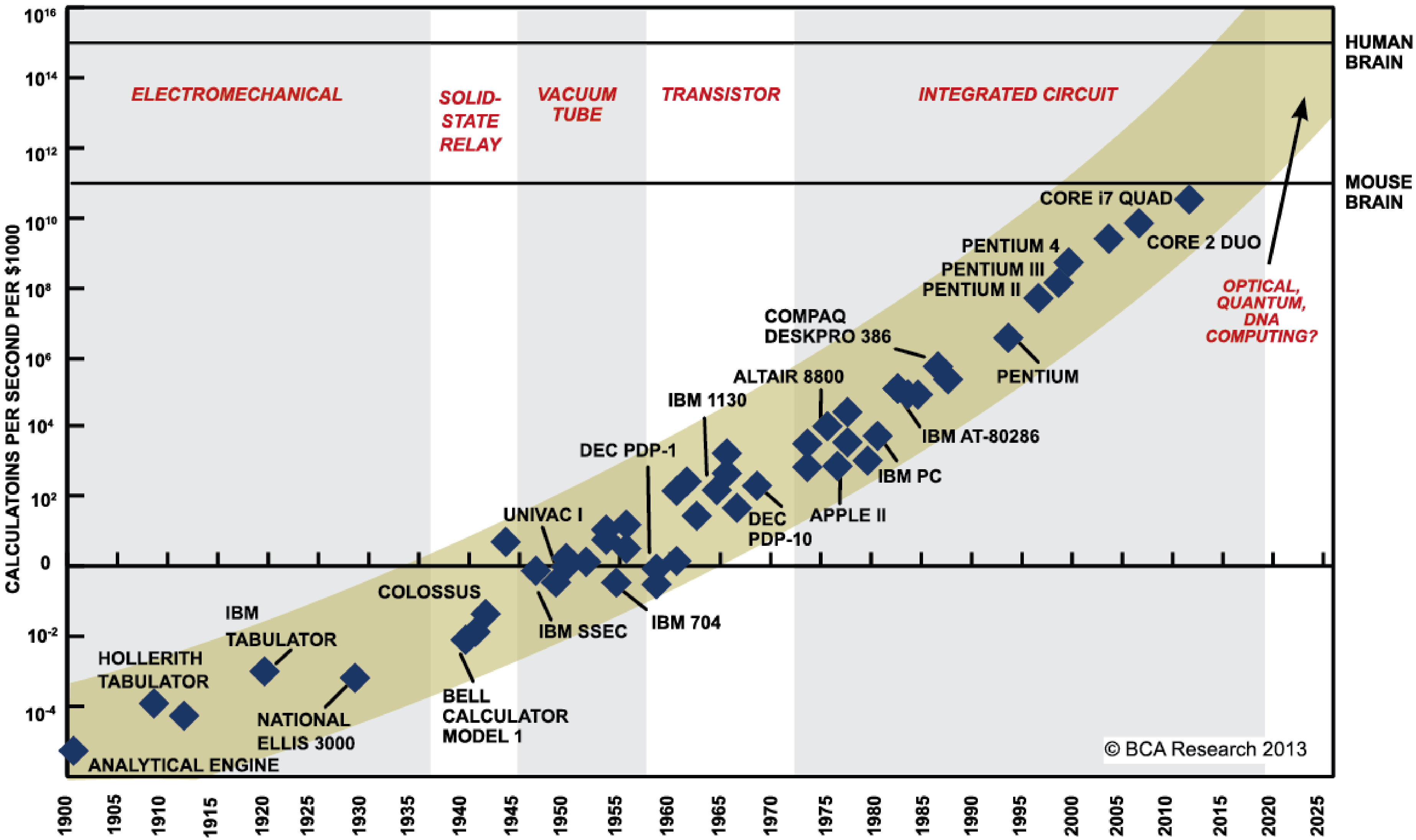
$O(N^{2.8074})$

$\mathcal{O}(n \log n)$

Paradigm shift



Moore’s law



SOURCE: RAY KURZWEIL, "THE SINGULARITY IS NEAR: WHEN HUMANS TRANSCEND BIOLOGY", P.67, *THE VIKING PRESS*, 2006. DATAPOINTS BETWEEN 2000 AND 2012 REPRESENT BCA ESTIMATES.

Interpretation of Moore’s law

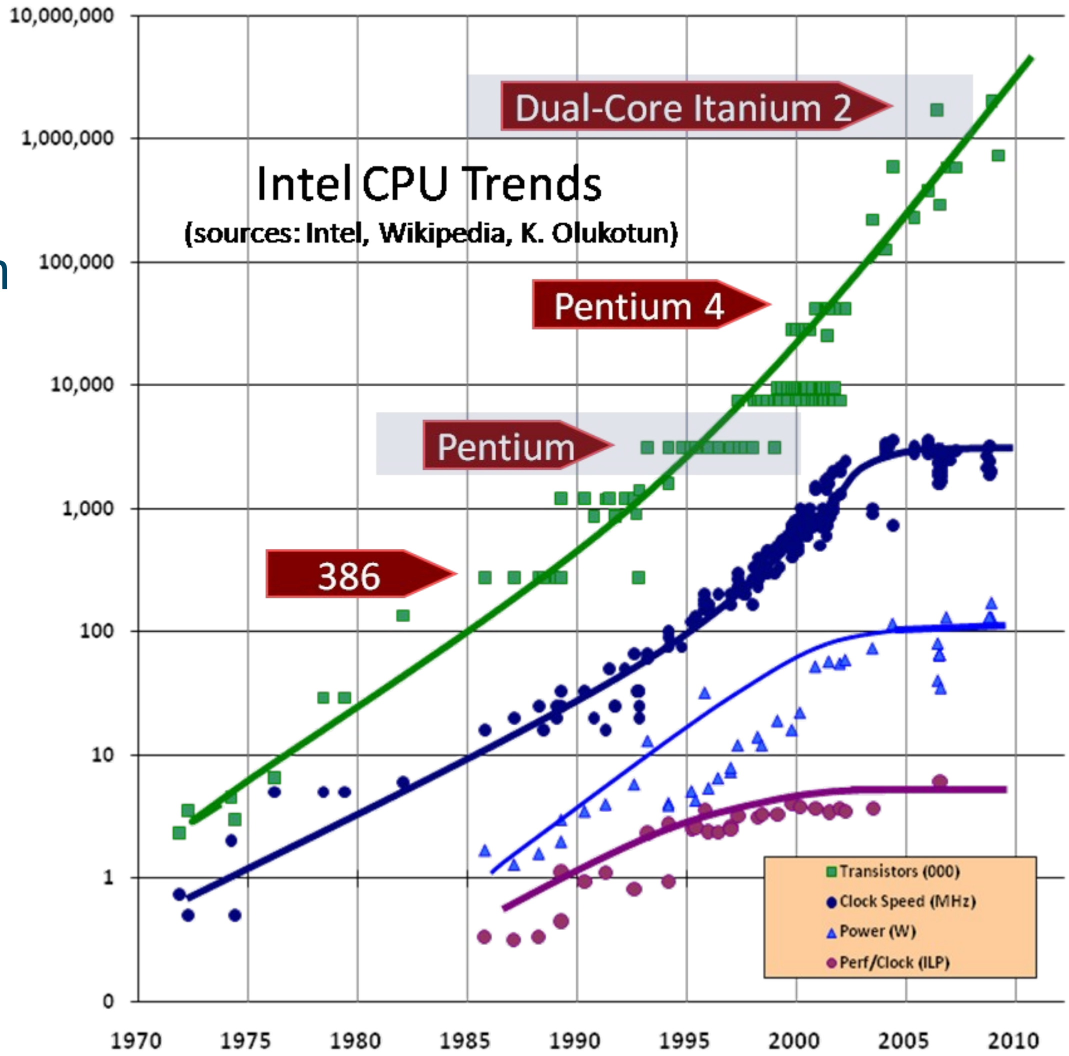
Until 2005 faster execution

Since 2005 parallel execution

Why?

- speed of light
- atomic boundaries
- limited 3D layering

⇒ Paradigm Shift



Basics of parallel processing

[Introduction to Hadoop]

Basics of parallel processing

[Introduction to Hadoop]

IDLab
INTERNET & DATA LAB

imec

What is Hadoop?

Apache top level project, open-source implementation of **frameworks for reliable, scalable, distributed computing and data storage.**

It is a flexible and highly-available architecture for large scale computation and data processing **on a network of commodity hardware**.

What happens in a Google Cluster?

- 1000 individual machine failures
- 1000’s of disk failures
- 1 PDU failure (~500-1000 machines disappear for ~6 h)
- 20 rack failures (40-80 machines disappear for 1-6 hours)
- 5 racks go wonky (40-80 machines see 50% packet loss)
- 3 router failures (have to immediately pull traffic for 1h)
- ...

Most (large) jobs see failures!

No, we’re not smart enough to program around it

Google Origins

2003

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

hadoop HDFS

2004

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

hadoop Map Reduce

2006

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

APACHE HBASE

Google MapReduce

A framework for processing LOADS of data

Framework's job: fault tolerance, scaling & coordination

Programmer's job: write program in MapReduce form

Hadoop is…

HDFS

Hadoop Distributed File System

Big Data Storage

&

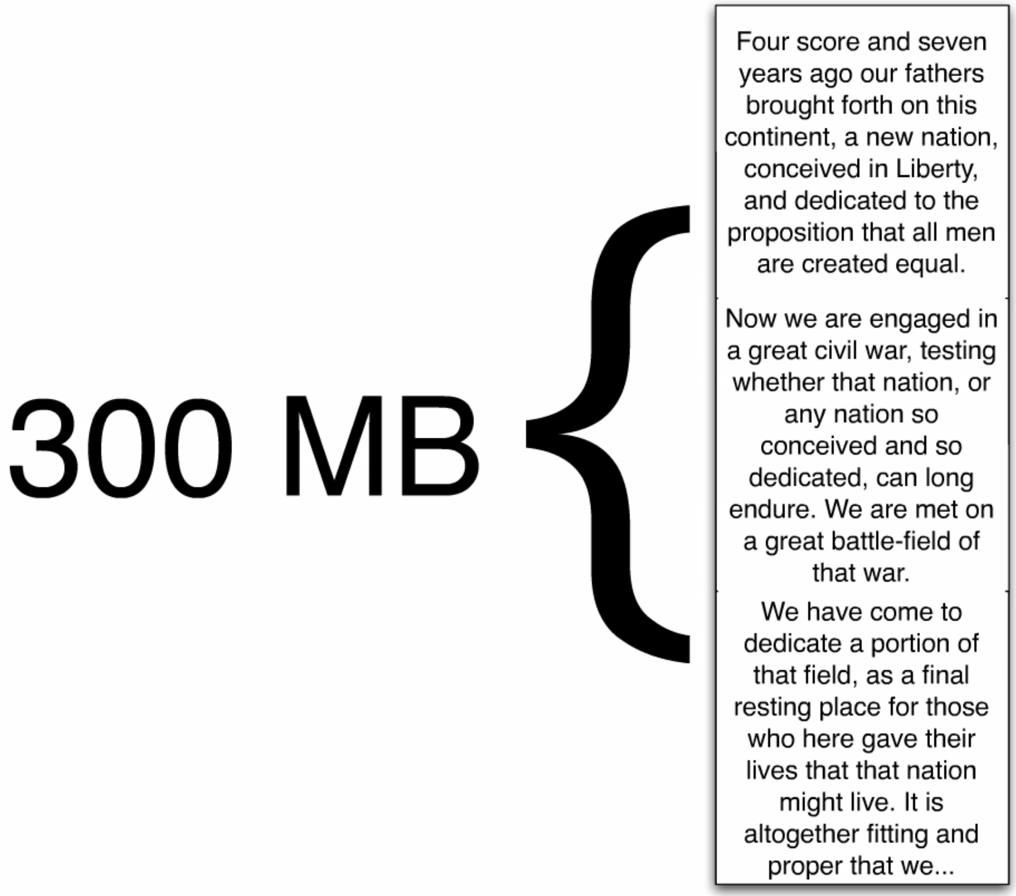
MapReduce

Big Data Processing

HDFS

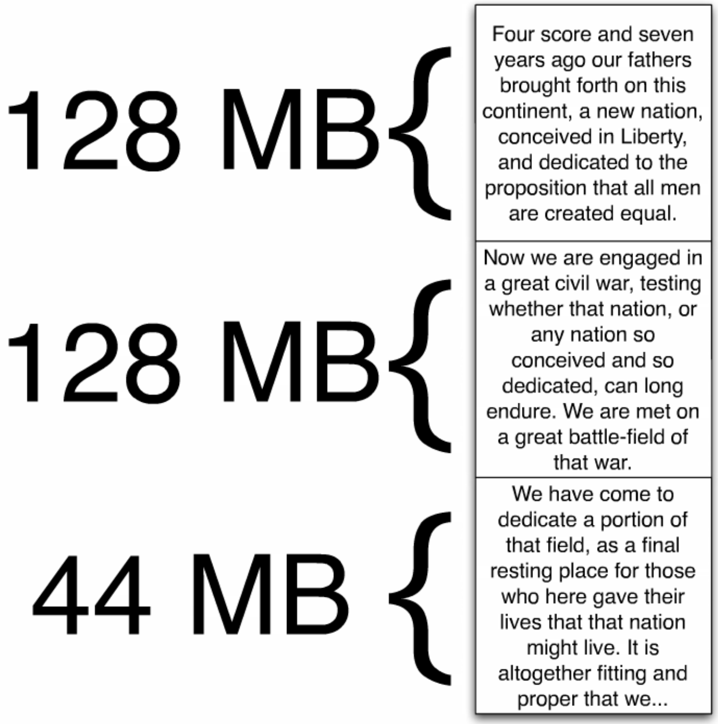
Hadoop Distributed File System

HDFS: storing large files



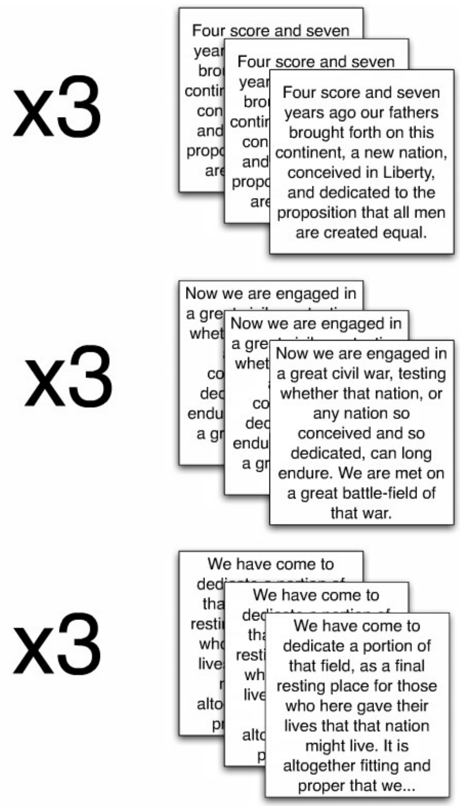
We have a file

HDFS: storing large files



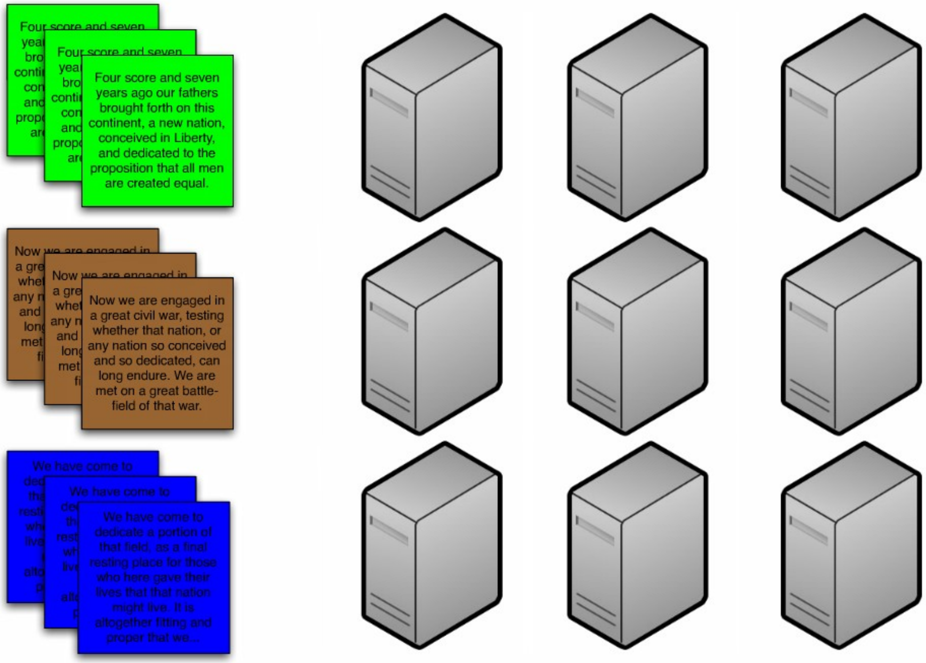
HDFS splits it into blocks

HDFS: storing large files



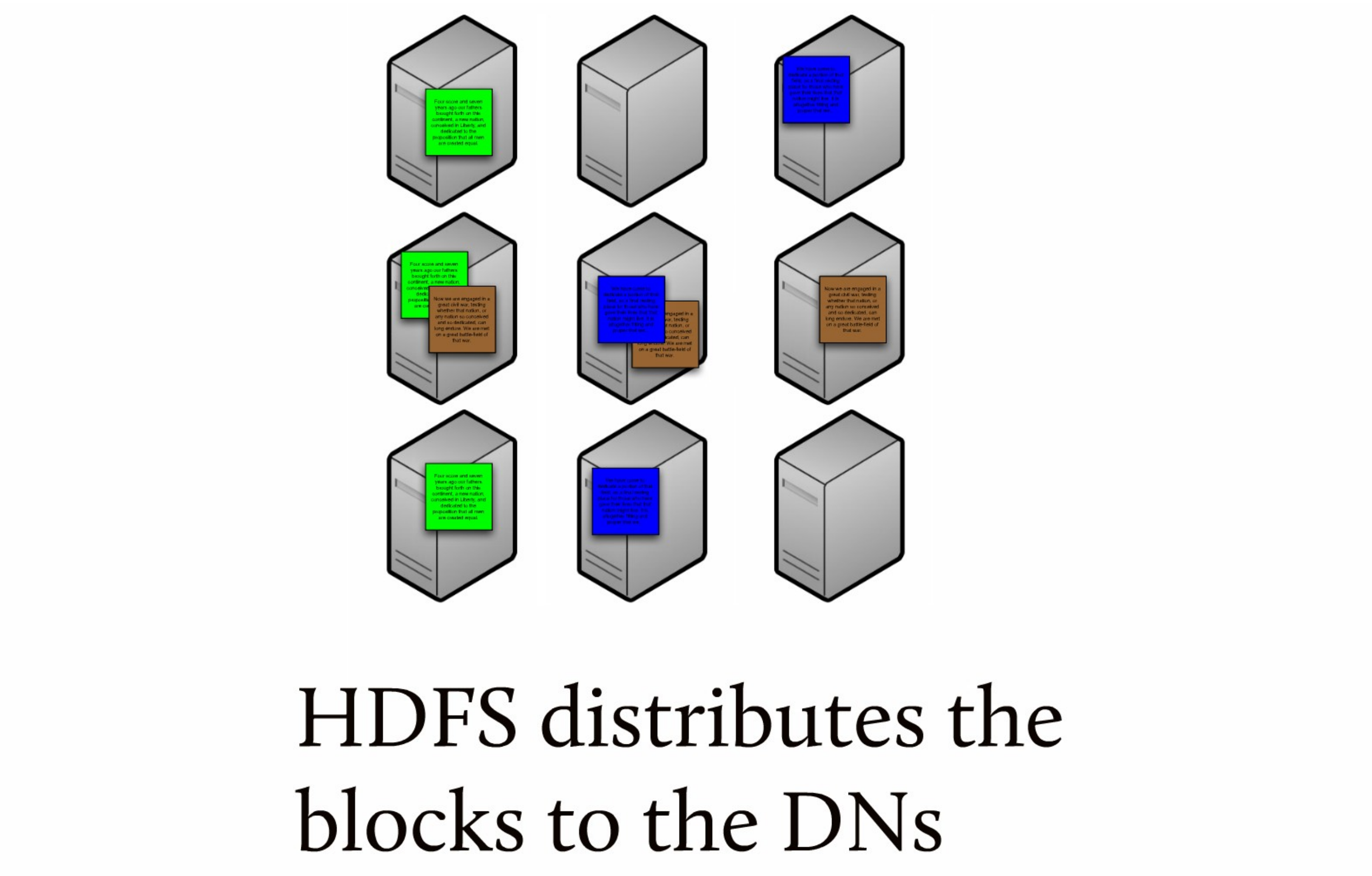
HDFS will keep 3 copies of each block

HDFS: storing large files

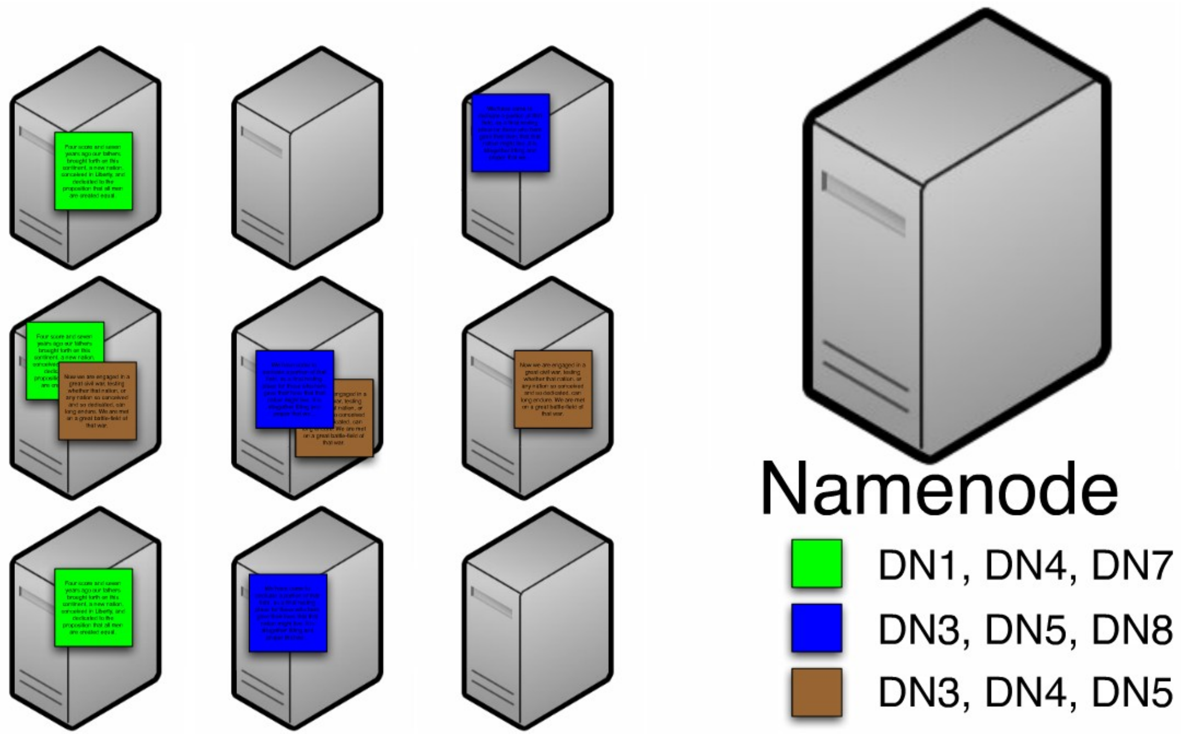


HDFS stores these blocks on datanodes

HDFS: storing large files

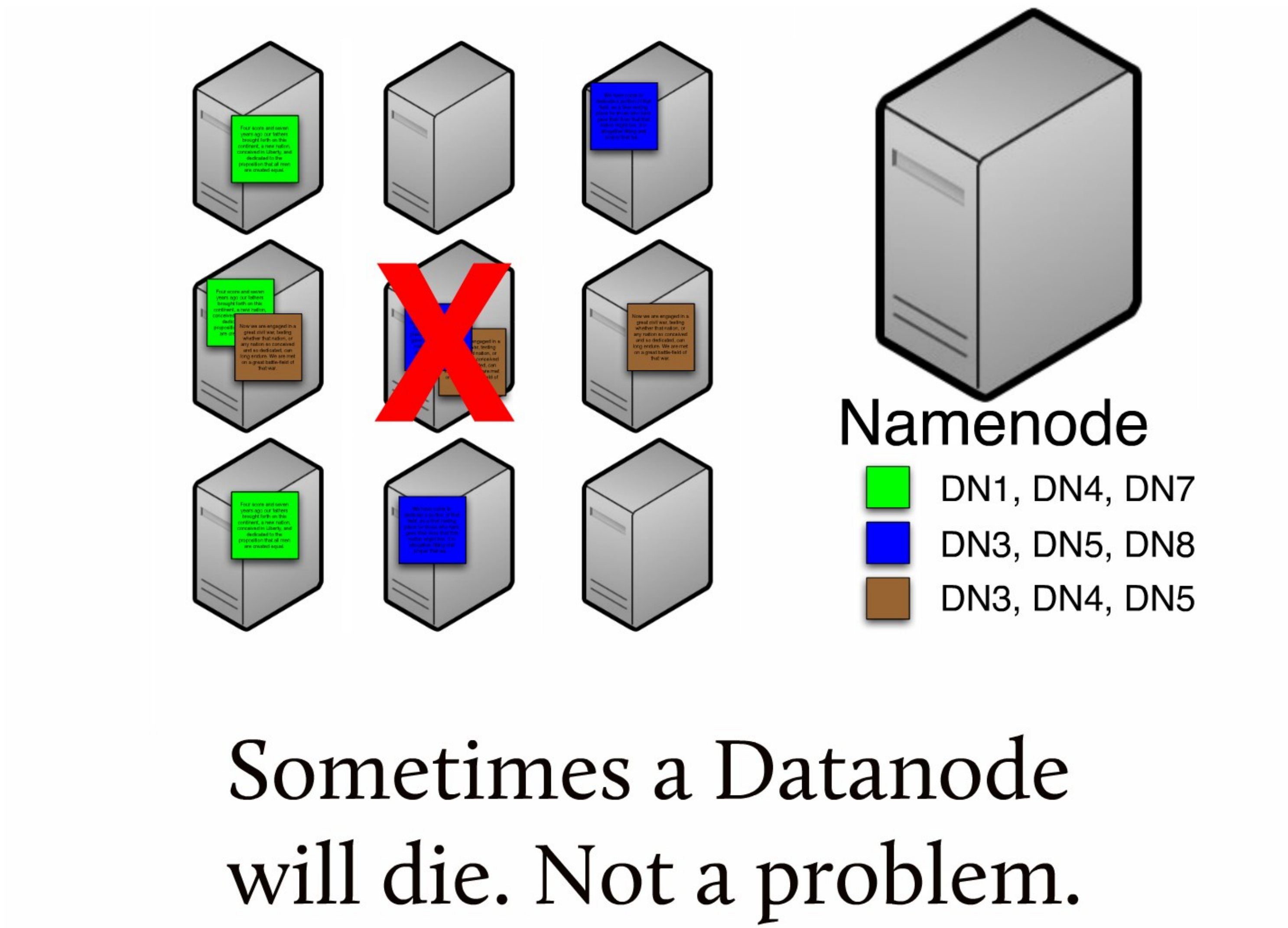


HDFS: storing large files

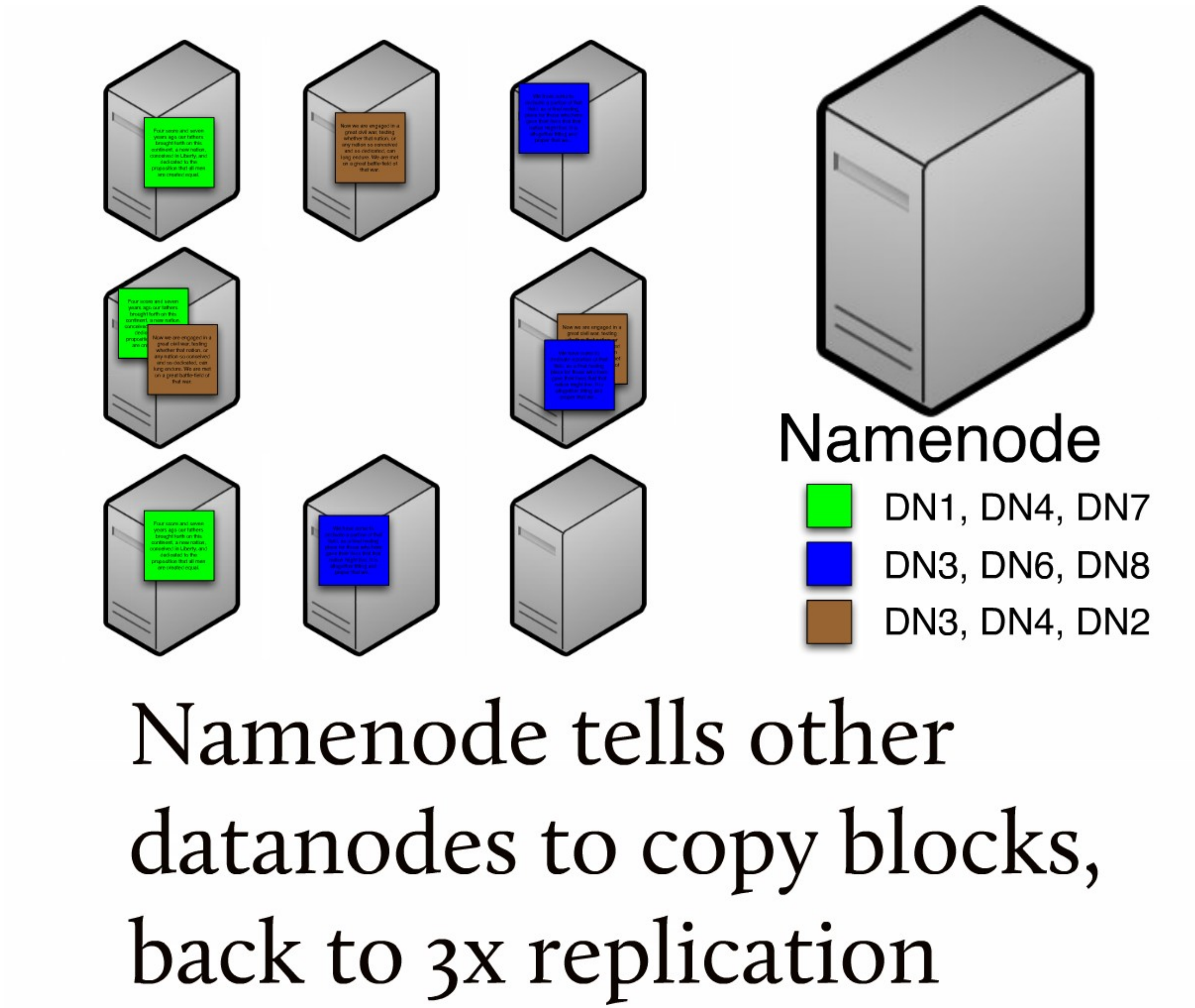


The NameNode tracks blocks and Datanodes

HDFS: storing large files



HDFS: storing large files



Map Reduce

Hadoop MapReduce

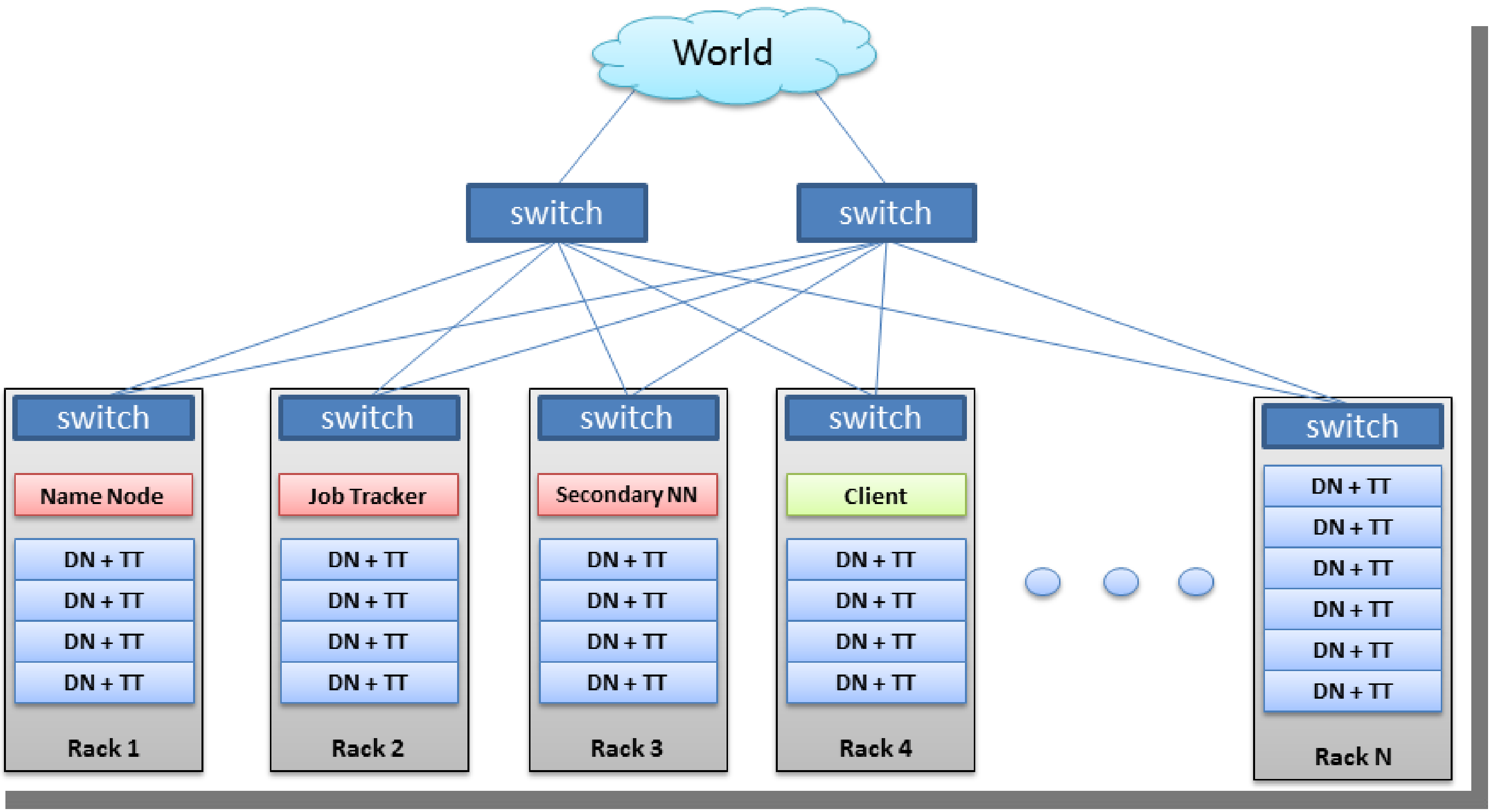
Model for distributed processing

Mantra:

Move the computation to the data

In practice this means running the processing on a machine whose HDFS Datanode holds the data

Hadoop architecture



TT : task tracker

Hadoop MapReduce

Data in the form of a <key, value> pair

- <byte, text>
- <user id, user profile>
- <timestamp, log entry>
- <user id, list of user id's of friends>
- …

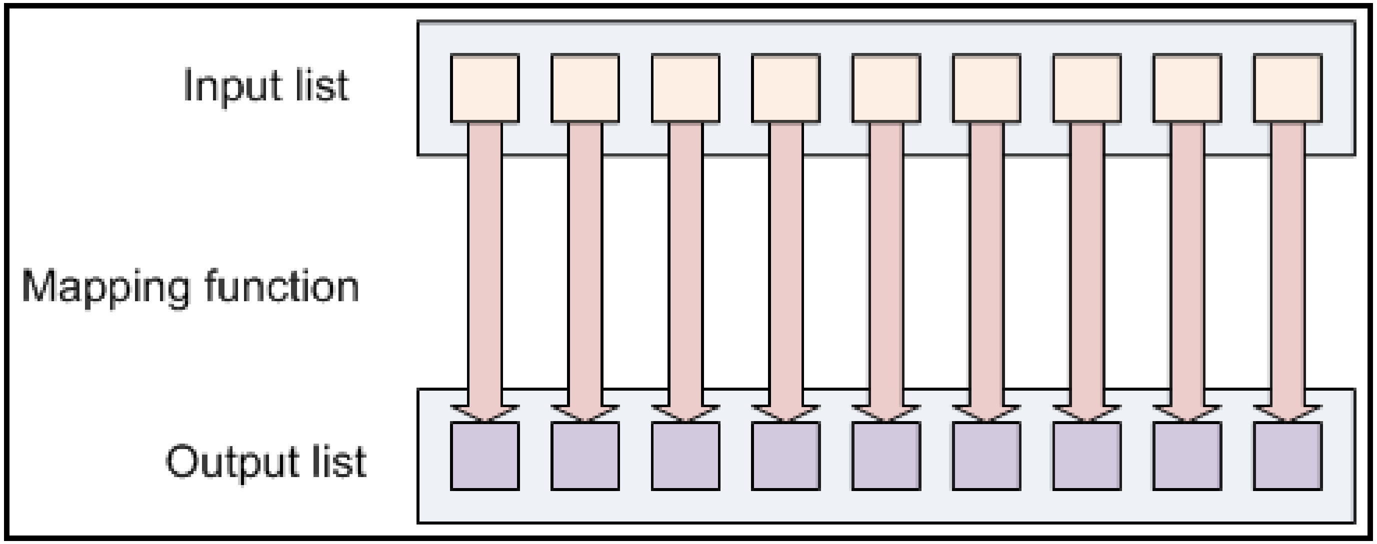
Inspired by list processing (Lisp) , functional programming:

- immutable data
- pure functions (no side effects): map, reduce

Simple model = easy to reason about

Google MapReduce

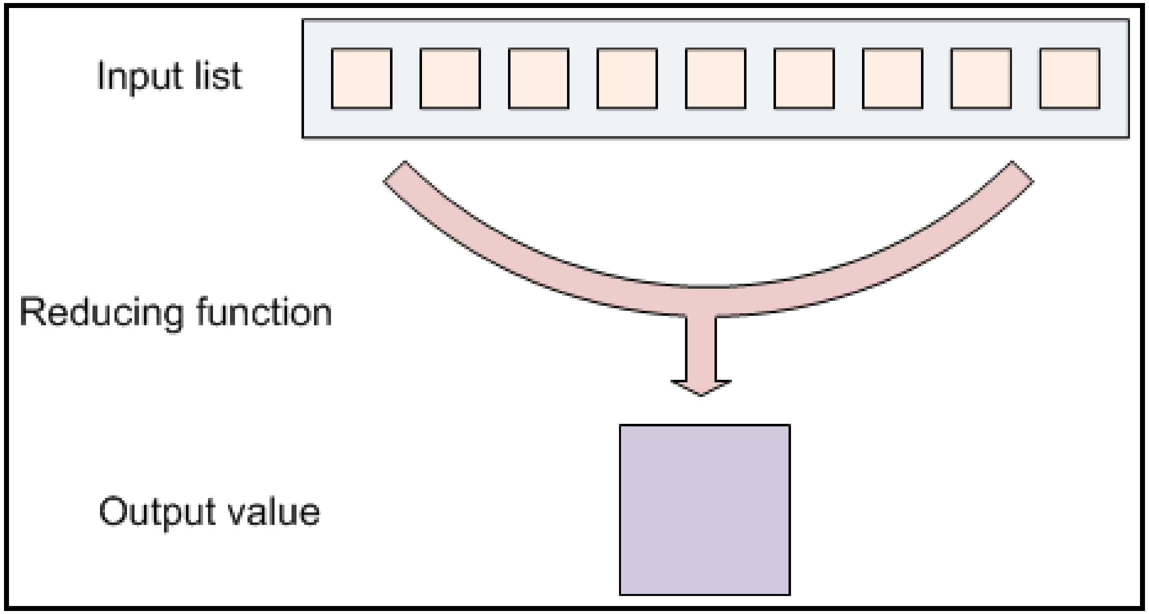
Map: map each <key, value> of input list onto 0, 1, or more pairs of type <key2, value2> of output list



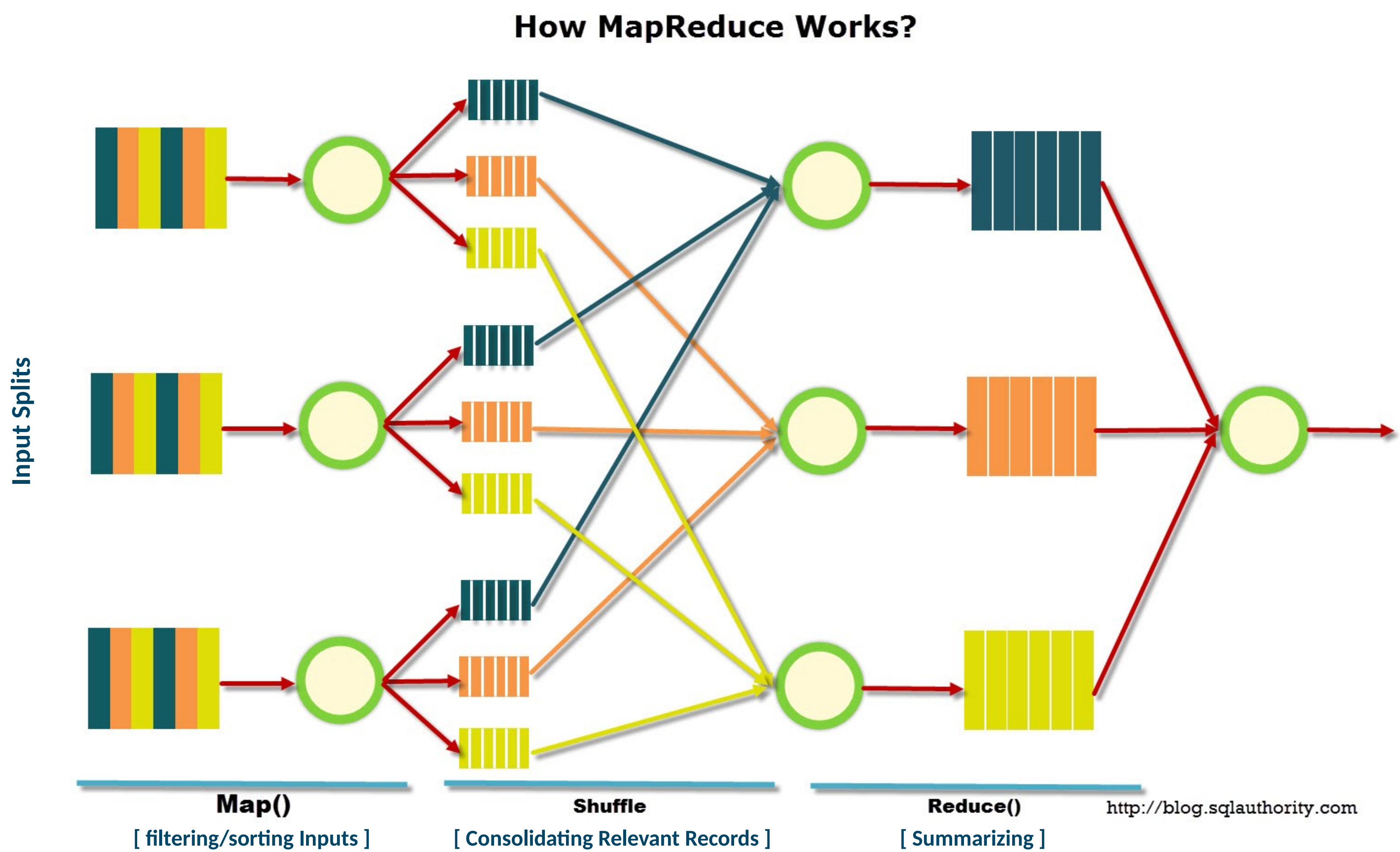
Behavior:
- Map to 0 elements in the output → filtering
- Map to +1 elements in the output → distribution

Google MapReduce

Reduce: combine the <key, value> pairs of the input list to an aggregate output value



Bringing it together



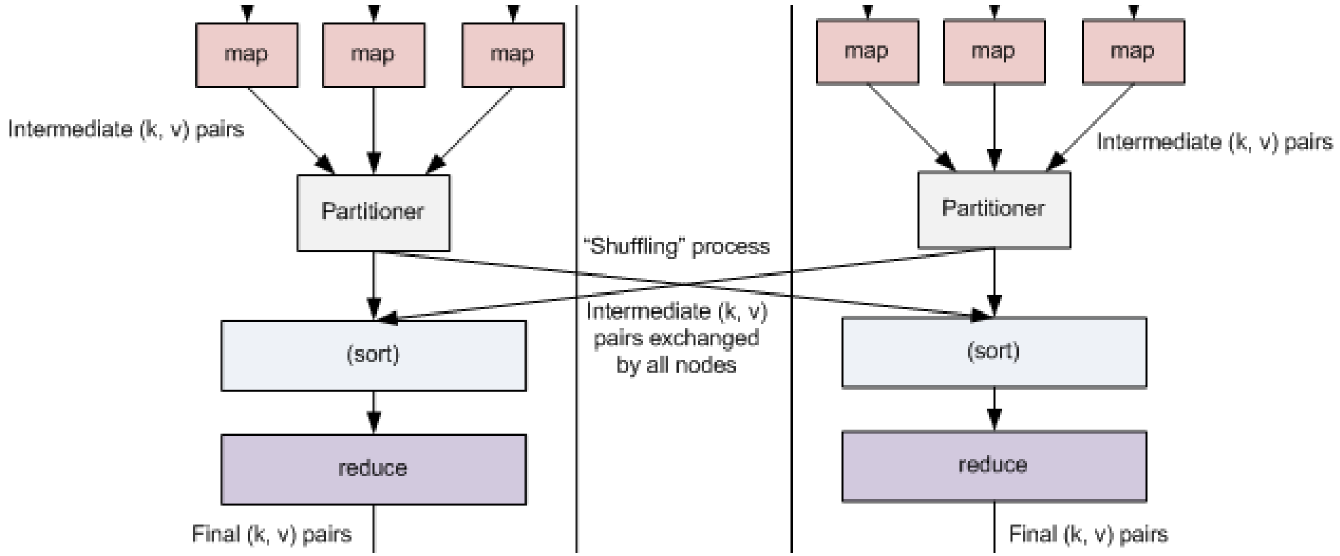
What you need to write

Mapper: application code

Partitioner: send data to correct Reducer machine

Sort: group input from different mappers by key

Reducer: application code



TokenizerMapper

```
public static class TokenizerMapper
      extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

TokenizerMapper

```
public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{

    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(Object key, Text value, Context context
                    ) throws IOException, InterruptedException {
      StringTokenizer itr = new StringTokenizer(value.toString());
      while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
      }
    }
  }
```

Breaking value into tokens

Pushing every token into context

IntSumReducer

```
public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
```

IntSumReducer

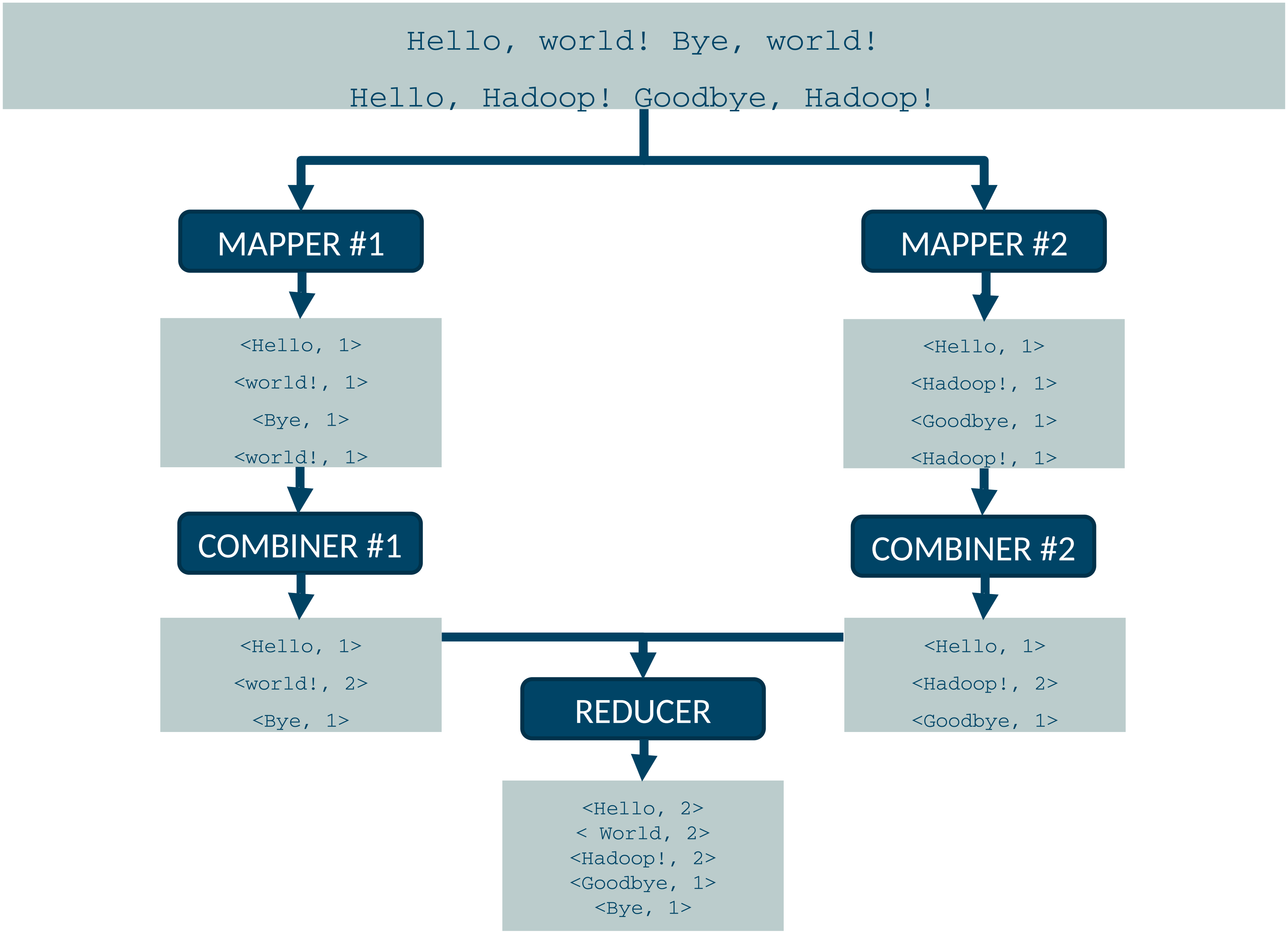
```
public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> {
    private IntWritable result = new IntWritable();

    public void reduce(Text key, Iterable<IntWritable> values,
                       Context context
                       ) throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
        sum += val.get();
      }
      result.set(sum);
      context.write(key, result);
    }
  }
```

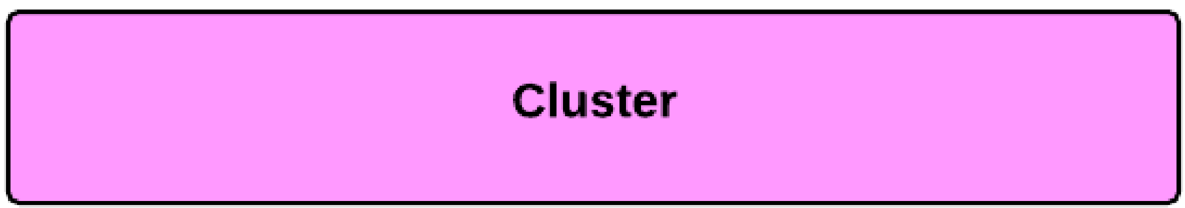
Accumulating all the values (int)

Pushing the sum (result) into context

Small example



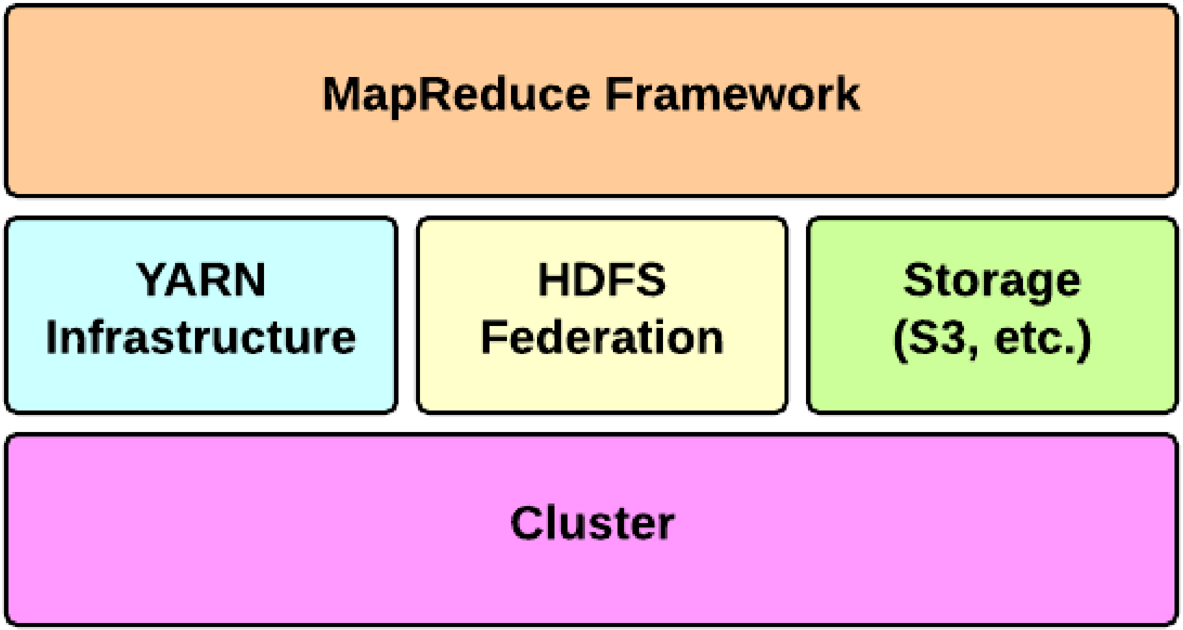
Hadoop Architecture Overview



Components

- **MapReduce Framework:** implement MapReduce paradigm
- **Cluster:** host machines (nodes).
- **HDFS federation:** provides logical distributed storage.
- **YARN Infrastructure:** assign resources (CPU, memory, etc.=

Hadoop Architecture Overview



Components

- **MapReduce Framework:** implement MapReduce paradigm
- **Cluster:** host machines (nodes).
- **HDFS federation:** provides logical distributed storage.
- **YARN Infrastructure:** assign resources (CPU, memory, etc.=

YARN Infrastructure

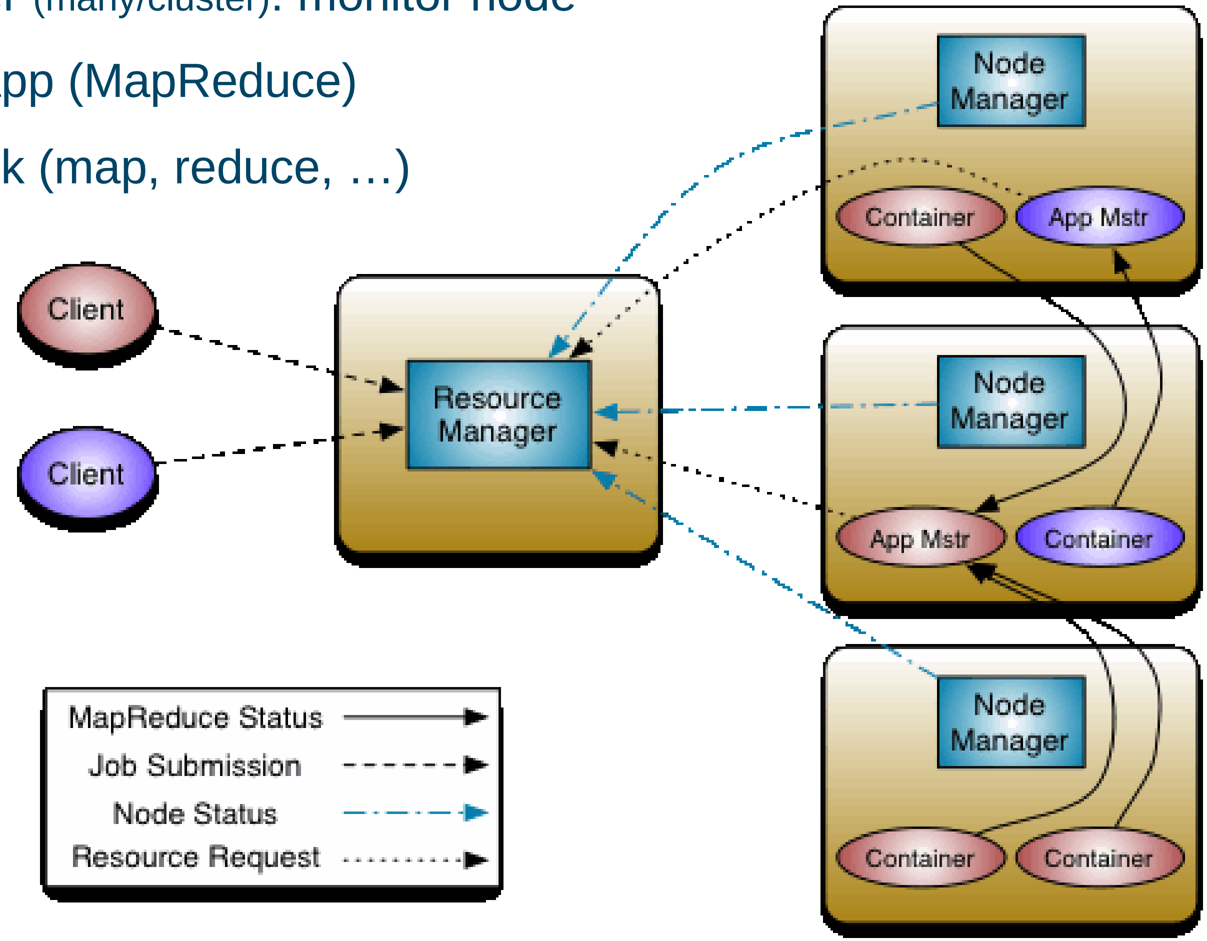
Yet Another Resource Negotiator

Resource Manager (1/cluster): assign cluster resources to applications

Node Manager (many/cluster): monitor node

App Master: app (MapReduce)

Container: task (map, reduce, …)

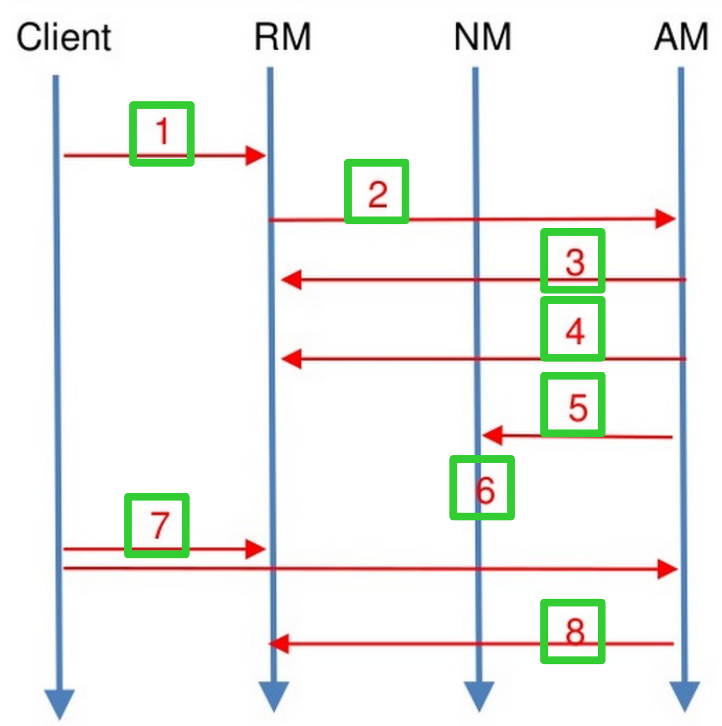


https://ercoppa.github.io/HadoopInternals/HadoopArchitectureOverview.html

YARN application lifecycle

RM: resource manager
NM: node Manager
AM: application master

1. Client submits app
2. RM allocates AM container
3. AM registers with NM
4. AM requests containers from RM
5. AM tells NM to launch containers
6. Application code is executed
7. Monitor app status in RM/AM
8. AM unregisters with RM



Application developers only need to write code for 6

Break

[See you in 15 mins.]

Hadoop in the wild

Hadoop in action: real-life examples

Largest Hadoop cluster in Europe?: 1650 machines (~20K jobs/day)

Use:

- Reporting to record labels so everyone gets paid
- Creating top lists of what is the most popular music right now
- Getting feedback on different aspects of the product

  → improve user experience
- Powering intelligent radio and discovery features

Data Infrastructure:

- 1650 Hadoop Nodes
- 65 PB Storage, 70 TB RAM
- *20 TB data ingested via Kafka/day*
- *200 TB generated by Hadoop/day*

https://cwiki.apache.org/confluence/display/HADOOP2/PoweredBy

Hadoop in action: real-life examples

What is trending? What causes a tweet to be trending?
General statistics, etc.

Ad reporting. Fake news detection. Recommendation, etc.

Crawling the web

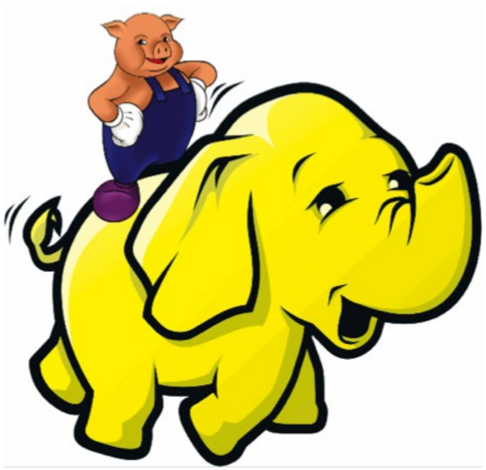
Will you switch banks?
Linked with social media data

Web clicks, where are users coming from?
Following up on user visits

the Hadoop eco-system

Pig: scripting language

```
a = load '/user/porky/word_count_text.txt';
b = foreach a generate  flatten(TOKENIZE((chararray)
$0)) as word;
c = group b by word;
d = foreach c generate COUNT(b), group;
    store d into '/user/porky/pig_wordcount';
```



Hive: SQL language

```
CREATE TABLE docs (line STRING);
LOAD DATA INPATH 'text' OVERWRITE INTO TABLE docs;
CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, '\s')) AS word FROM docs) word
GROUP BY word
ORDER BY word;
```

Kafka: the Log

Distributed commit log / message service

Store all events: richer than a DB (= only last value)

Will be HUGE once the Internet of Things matures

Multithreaded programming

Multithreaded programming

Shortcoming of MapReduce

Forces your data processing into **MAP** and **REDUCE**

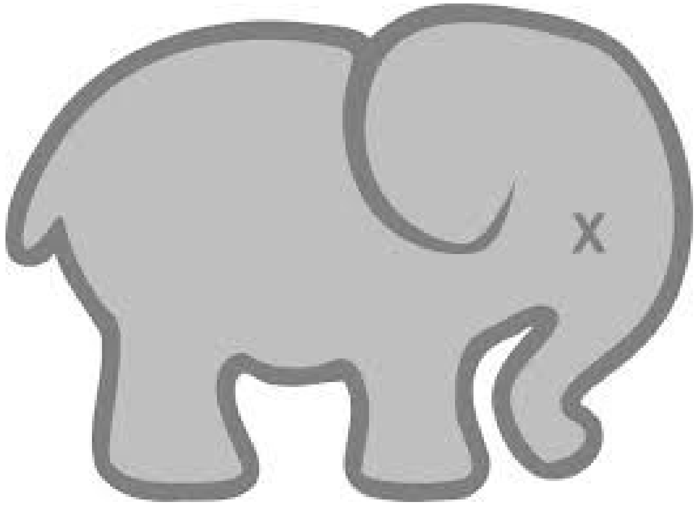
- Other workflows missing include join, filter, flatMap, groupByKey, union, intersection, …

Based on "Acyclic Data Flow" from Disk to Disk (HDFS)

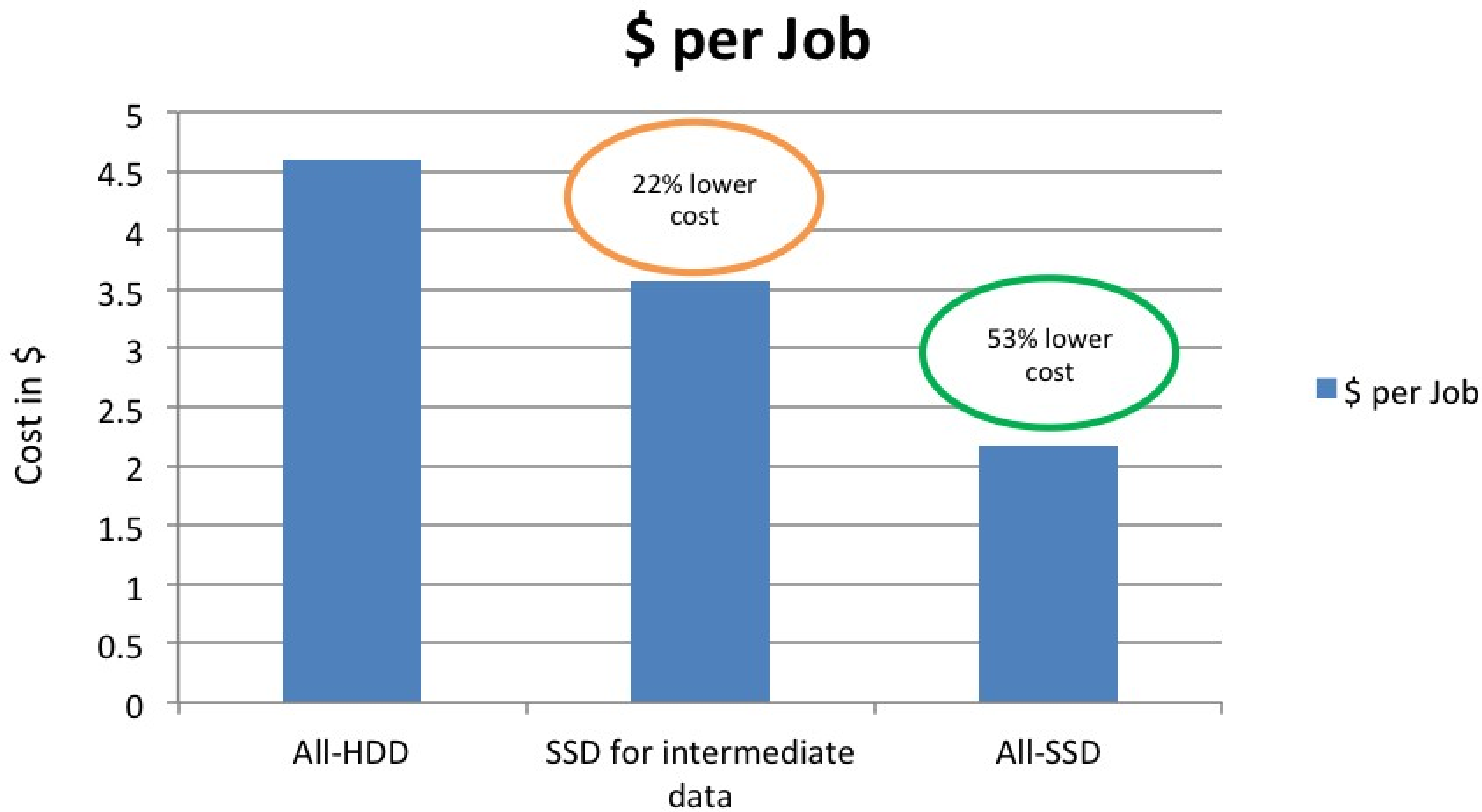
- Not efficient for iterative tasks, i.e. Machine Learning

Only for Batch processing

- Interactivity, streaming data



Hadoop and disks



Hard drive access is killing performance and blocking functionality

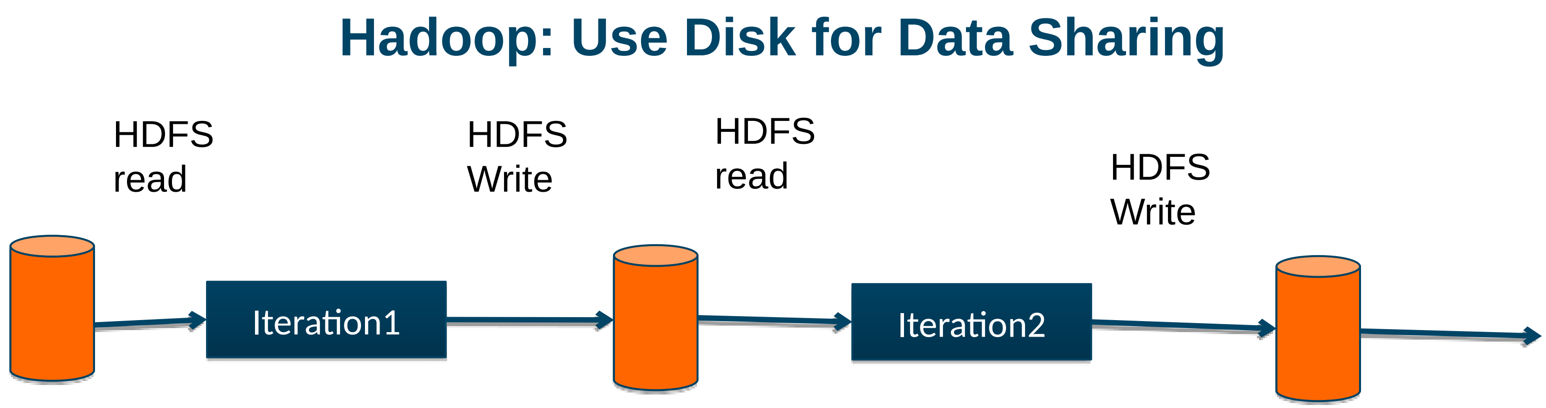
One Solution is Apache Spark

Works on top of Hadoop, HDFS, ……

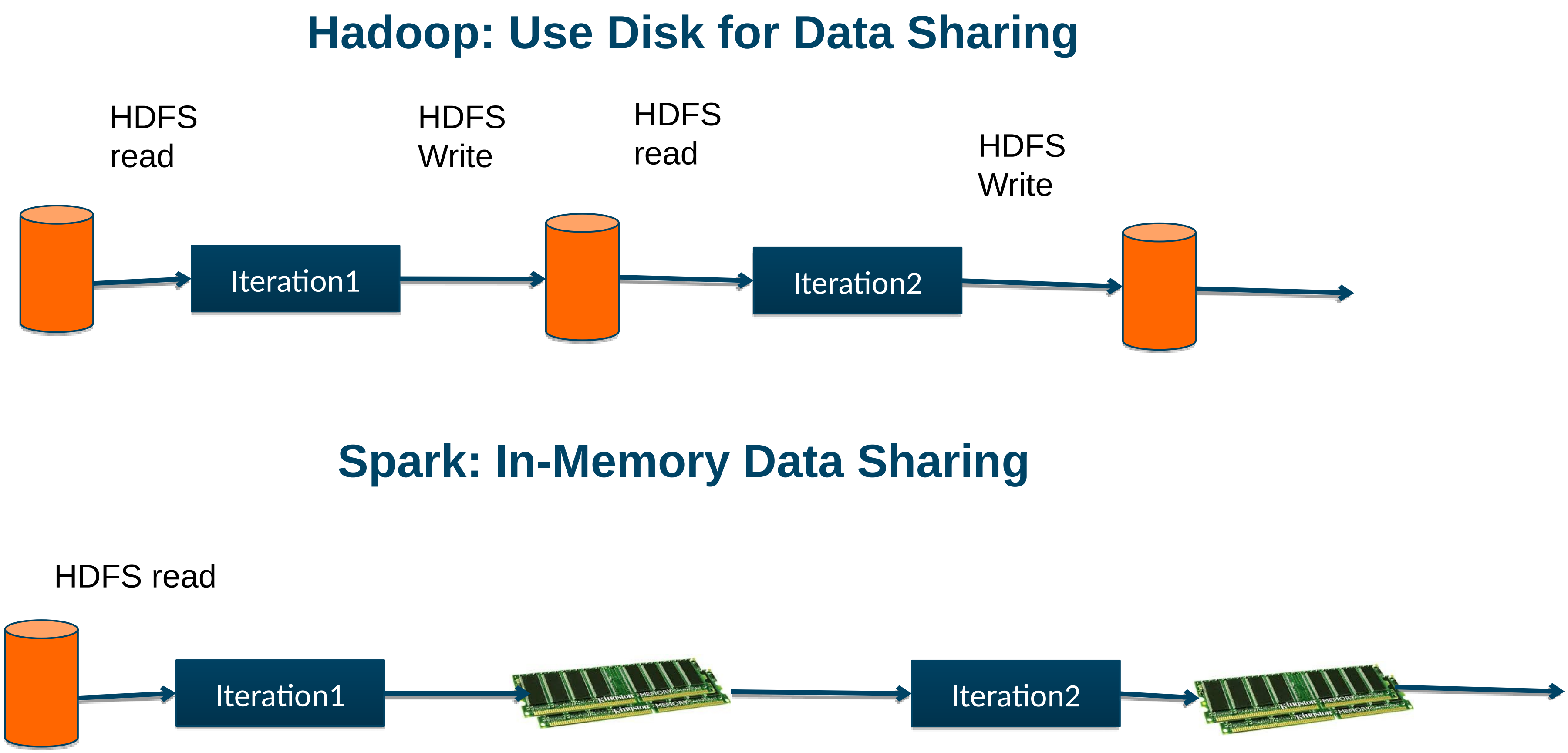
Has many other workflows, i.e. join, filter, flatMapdistinct, groupByKey, reduceByKey, sortByKey, collect, count, first… (around 30 efficient **distributed operations**)

In-memory caching of data (for iterative, graph, and machine learning algorithms, etc.)

Spark Uses Memory instead of Disk



Spark Uses Memory instead of Disk



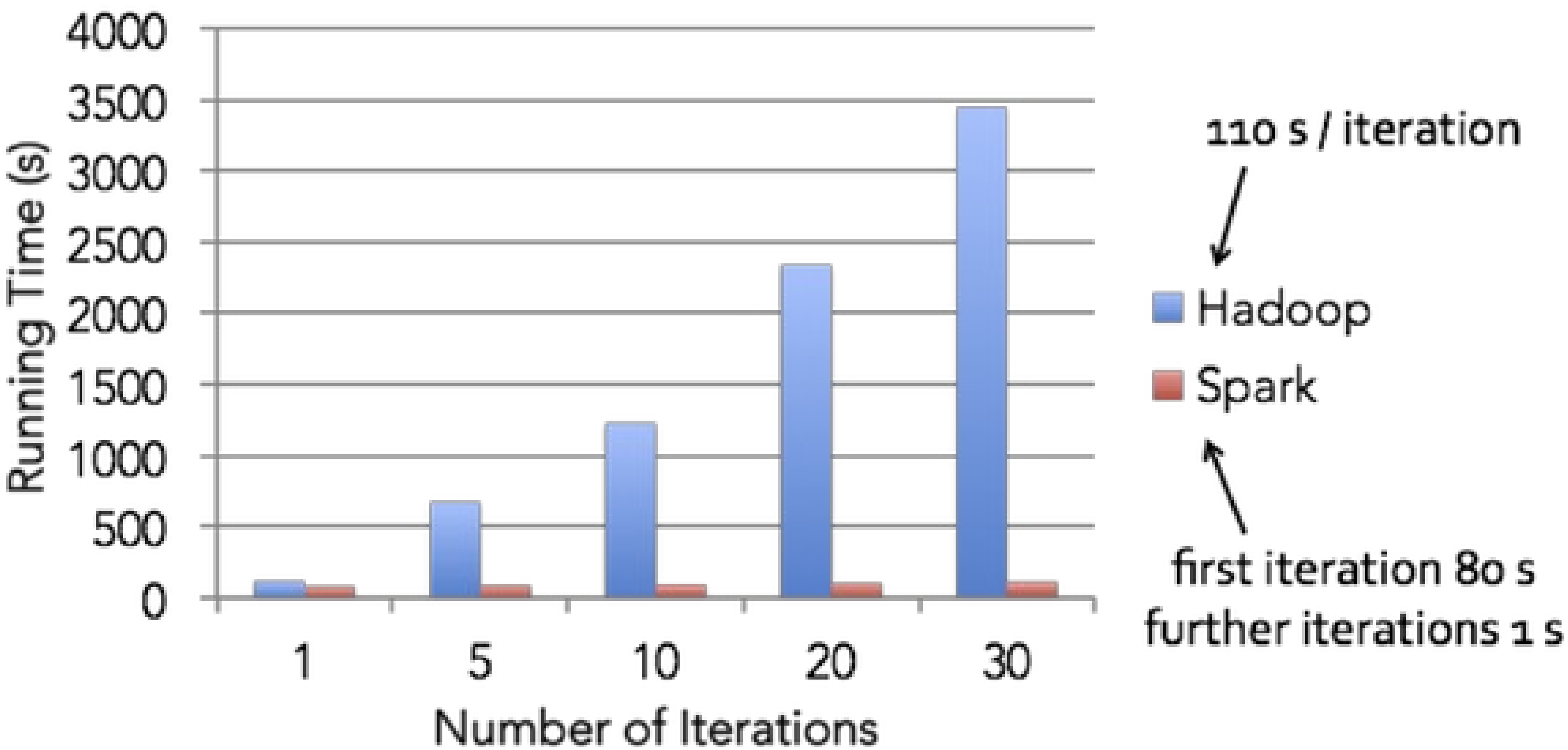
Sort competition

	Hadoop MR Record (2013)	Spark Record (2014)
Data Size	102.5 TB	100 TB
Elapsed Time	72 mins	23 mins
# Nodes	2100	206
# Cores	50400 physical	6592 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**

Spark, 3x faster with 1/10 the nodes

Sort benchmark, Daytona Gray: sort of 100 TB of data (1 trillion records)
http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html

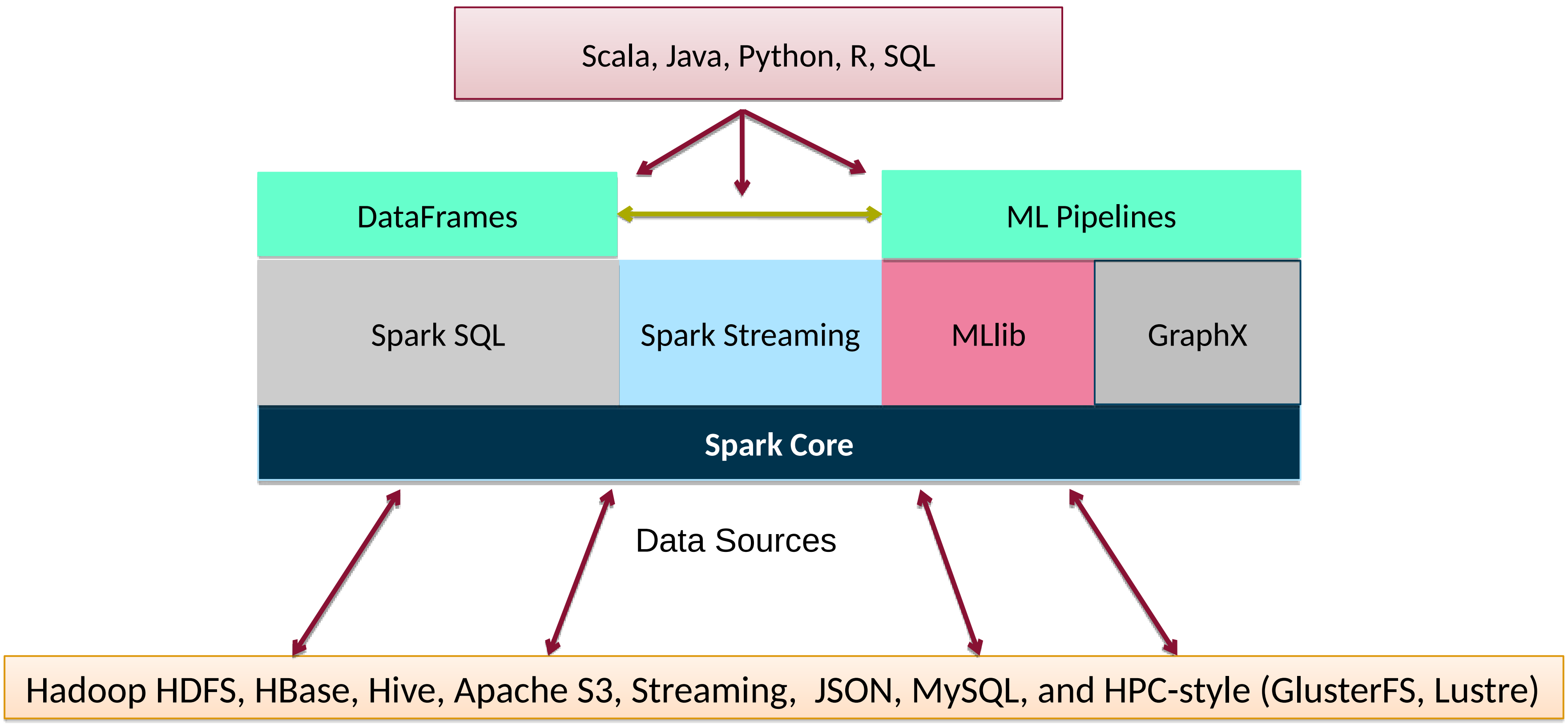
Logistic regression performance



Apache Spark

Apache Spark supports data analysis, machine learning, graphs, streaming data, etc.

It can read/write from a **range of data types** and allows **development in multiple languages**.



Resilient Distributed Datasets (RDDs)

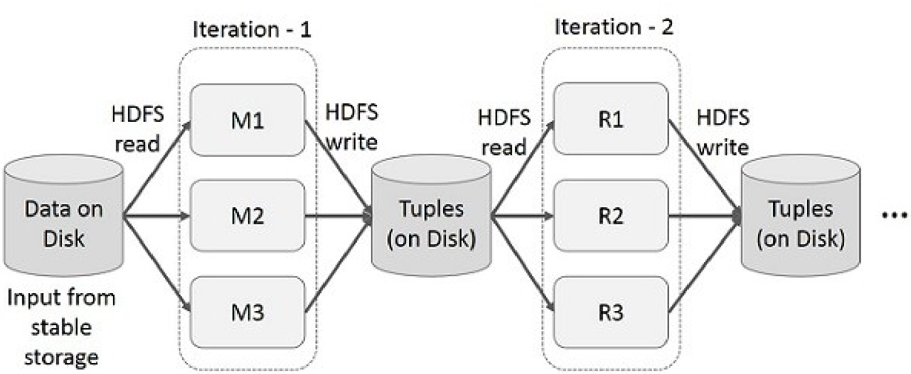
Immutable distributed collection of objects

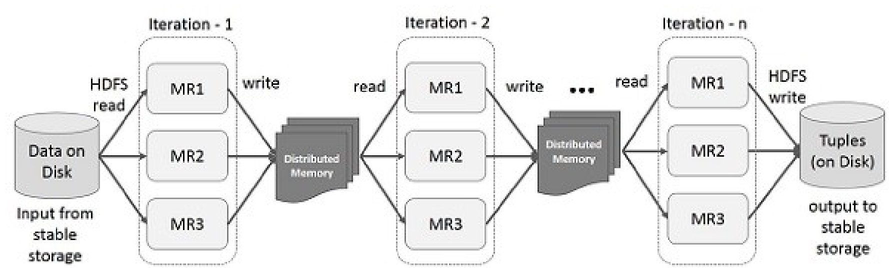
All Spark components use RDDs

Use transformations to create new RDDs

- From storage
- From other RDDs

Fault tolerant



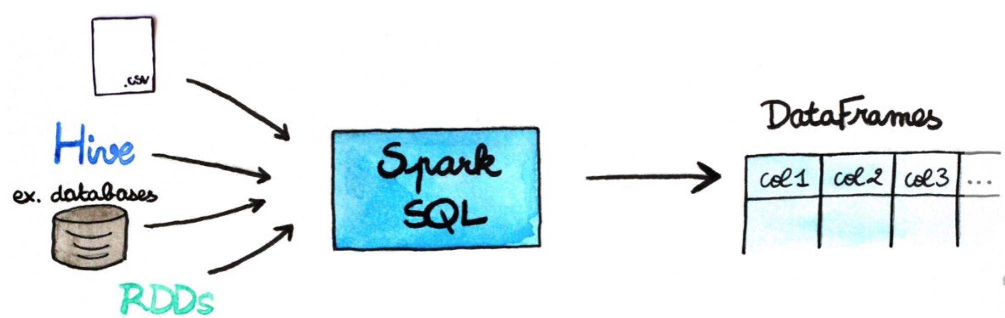


DataFrames & SparkSQL

Organize the data in named columns

Similar to a relational database…

- Immutable once constructed
- Enable distributed computations



DataFrames & SparkSQL

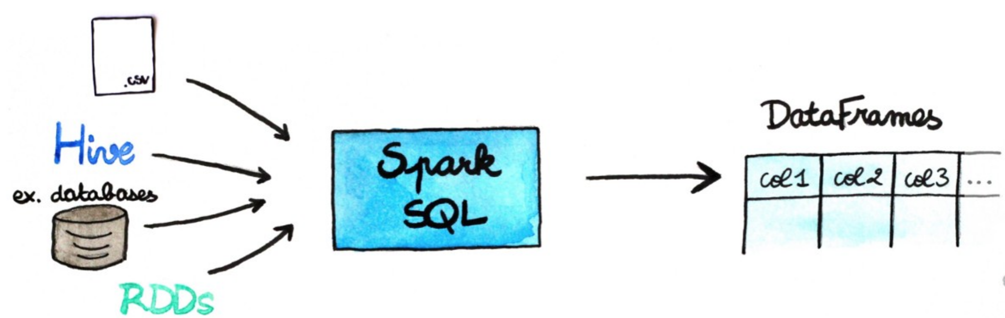
Organize the data in named columns

Similar to a relational database…

- Immutable once constructed
- Enable distributed computations

How to construct Dataframes

- Read from file(s)
- Transforming an existing DFs
- Parallelizing a python collection list
- Apply transformations and actions



DataFrame example

```
// Create a new DataFrame that contains “students”
students = users.filter(users.age < 21)

//Count the number of students users by gender
students.groupBy("gender").count()

// Join young students with another DataFrame
called logs
students.join(logs, logs.userId == users.userId,
“left_outer")
```

RDDs vs. DataFrames

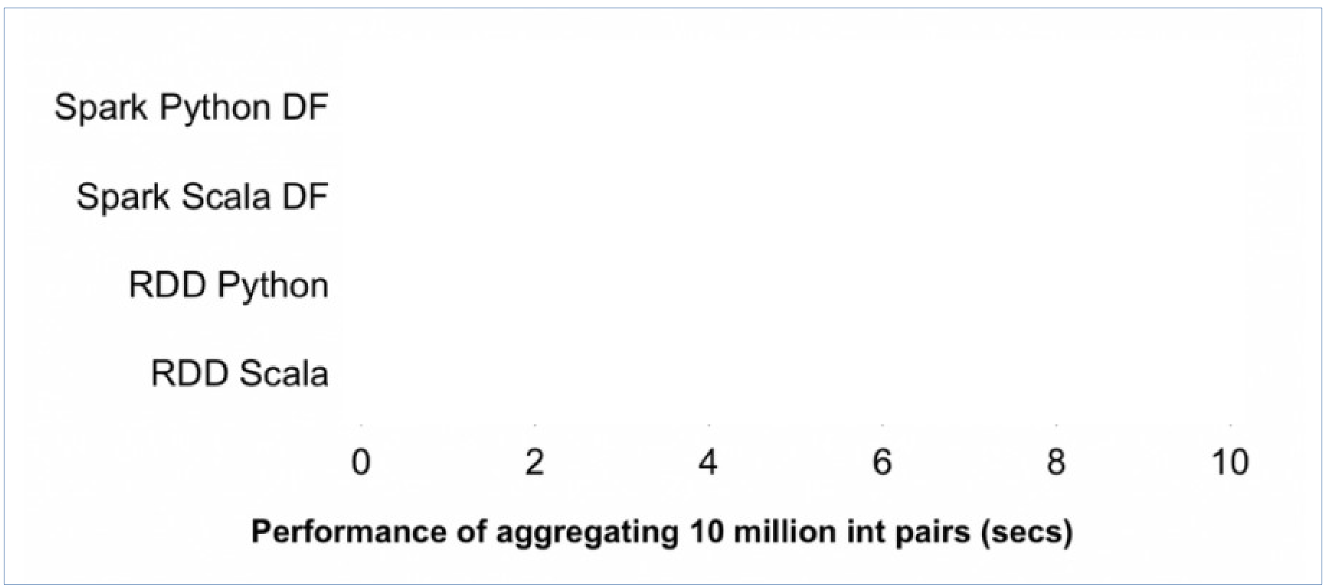
RDDs **provide a low level interface** into Spark

DataFrames **have a schema**

DataFrames are **cached and optimized by Spark**

DataFrames are **built on top of the RDDs and the core Spark API**

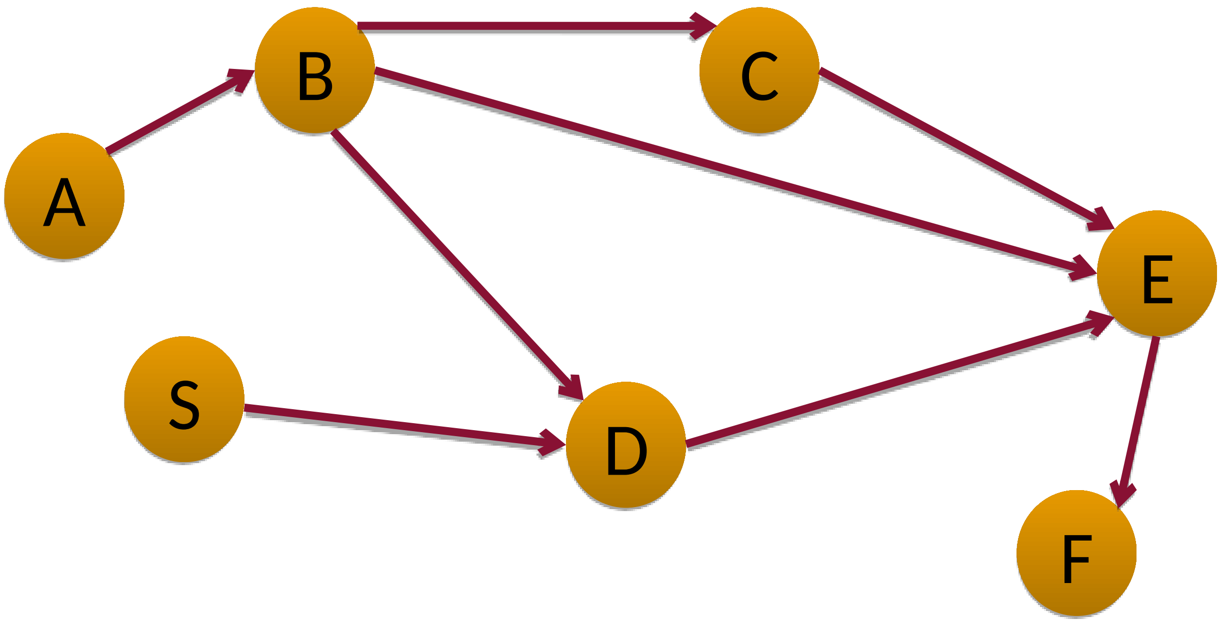
Example: performance



Spark Operations

Transformations (create a new RDD)	map filter sample groupByKey reduceByKey sortByKey intersection	flatMap union join cogroup cross mapValues reduceByKey
Actions (return results to driver program)	collect Reduce Count takeSample take lookupKey	first take takeOrdered countByKey save foreach

Directed Acyclic Graphs (DAG)



DAGs track dependencies
(also known as Lineage)

- nodes are RDDs
- arrows are Transformations

Why?

- *Program resonates with humans and computers*
- Improvement via:
  - Sequential access to data
  - *- Predictive processing*

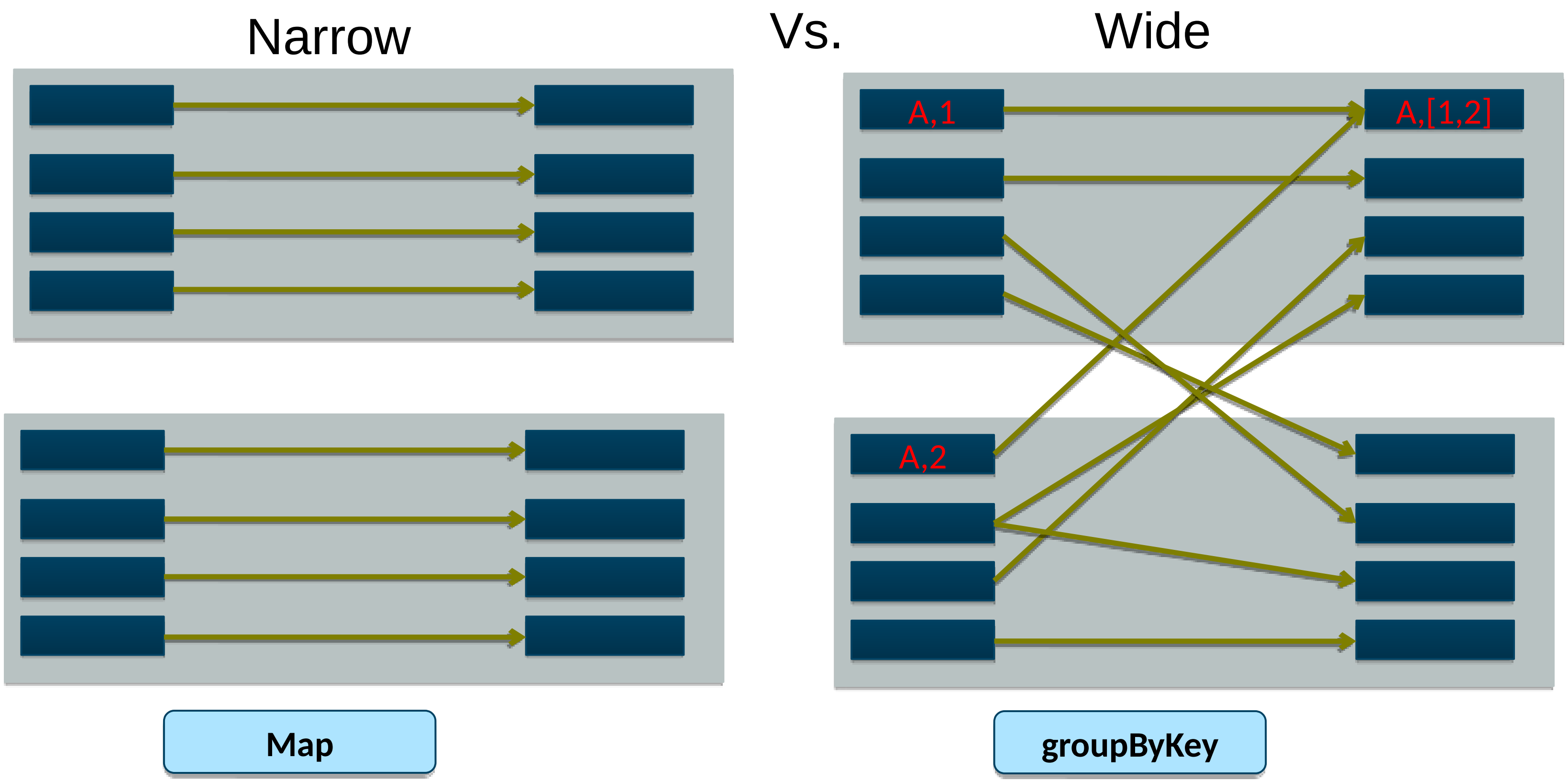
Narrow Vs. Wide transformation

Narrow

Map

Required elements for computation in a single partition **live in the single partition** of parent RDD

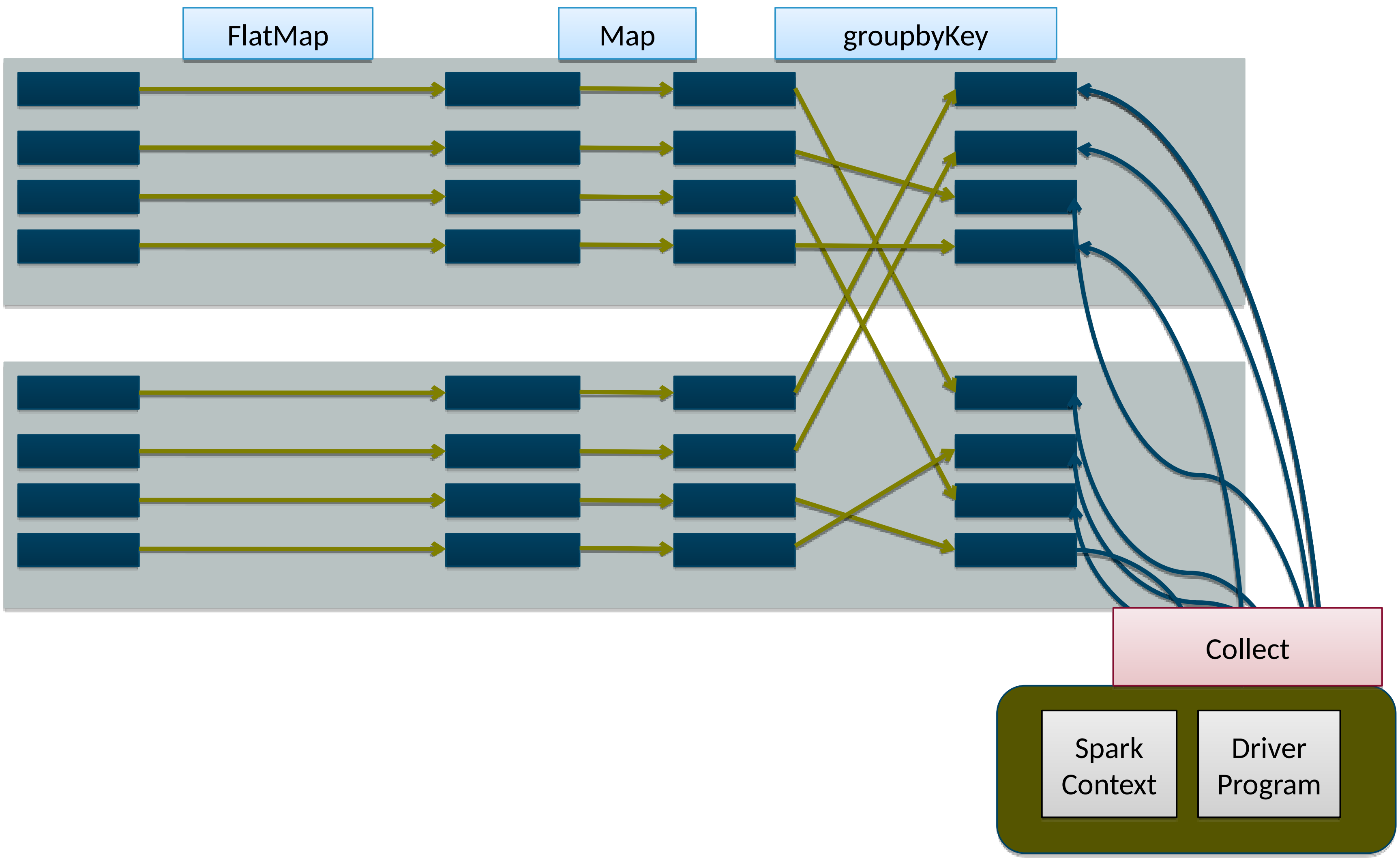
Narrow Vs. Wide transformation



Required elements for computation in a single partition **live in the single partition** of parent RDD

Required elements for computation in a single partition **may live in many partitions** of parent RDD

Spark Workflow



Python RDD API Examples

Word count

```
text_file = sc.textFile("hdfs://usr/godil/text/book.txt")
counts = text_file.flatMap(lambda line: line.split(" ")) \
             .map(lambda word: (word, 1)) \
           .reduceByKey(lambda a, b: a + b)
counts.saveAsTextFile("hdfs://usr/godil/output/wordCount.txt")
```

Logistic Regression

```
# Every record of this DataFrame contains the label and
# features represented by a vector.
df = sqlContext.createDataFrame(data, ["label", "features"])
# Set parameters for the algorithm.
# Here, we limit the number of iterations to 10.
lr = LogisticRegression(maxIter=10)
# Fit the model to the data.
model = lr.fit(df)
# Given a dataset, predict each point's label, and show the results.
model.transform(df).show()
```

Examples from http://spark.apache.org/

Spark’s Main Use Cases

Streaming Data

Machine Learning

Interactive Analysis

Data Warehousing

Batch Processing

Exploratory Data Analysis

Graph Data Analysis

Spatial (GIS) Data Analysis

And many more

Spark in the Real World (I)

Uber – the online taxi company gathers terabytes of event data from its mobile users every day.

- By using Kafka, Spark Streaming, and HDFS, to build a continuous ETL (*extract*, *transform*, *load*) pipeline
- Convert raw unstructured event data into structured data as it is collected
- Uses it further for more complex analytics and optimization of operations

Pinterest – Uses a Spark ETL pipeline

- Leverages Spark Streaming to gain immediate insight into how users all over the world are engaging with Pins—in real time.
- Can make more relevant recommendations as people navigate the site
- Recommends related Pins
- Determine which products to buy, or destinations to visit

Spark: when not to use

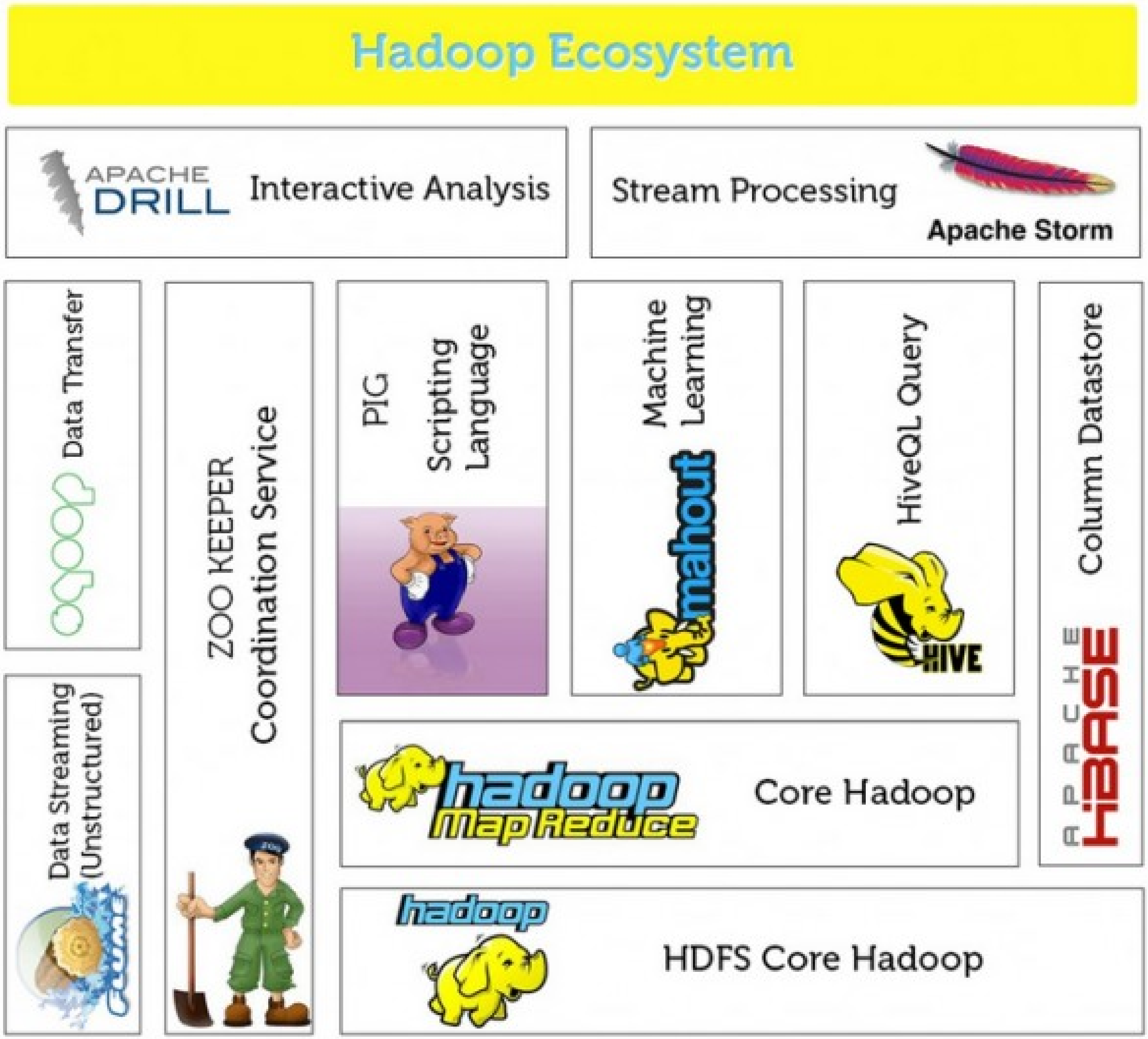
Even though Spark is versatile, that doesn't mean Spark's in-memory capabilities are the best fit for all use cases:

- For many **simple use cases** Apache MapReduce and Hive might be a more appropriate choice

- Spark was not designed as a **multi-user environment**

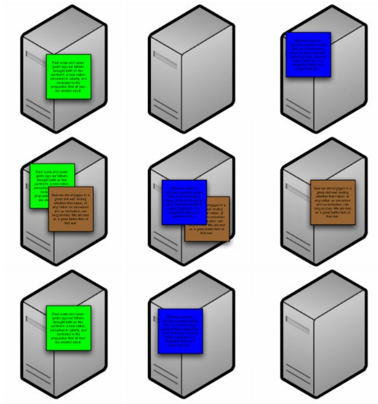
- Spark users are **required to know that memory they have is sufficient** for a dataset

- **Adding more users adds complications**, since the users will have to coordinate memory usage to run code



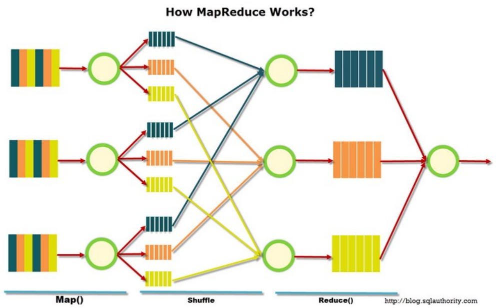
Hadoop Ecosystem
APACHE DRILL
Interactive Analysis
Stream Processing
Apache Storm
Data Transfer
ZOO KEEPER
Coordination Service
PIG
Scripting
Language
Machine
Learning
mahout
HiveQL Query
HIVE
Column Datastore
APACHE HBASE
Data Streaming
(Unstructured)
hadoop
Map Reduce
Core Hadoop
hadoop
HDFS Core Hadoop

Summarizing



Distributed storage

- HDFS and related technologies enable resilient storage



Distributed processing

- MapReduce paradigm
- Spark enables interactive processing and a richer set of operations.

Questions?

Further Reading

- **Hadoop**
  http://hadoop.apache.org/
  https://cwiki.apache.org/confluence/display/HADOOP2
  https://ercoppa.github.io/HadoopInternals/HadoopArchitectureOverview.html

- **MapReduce**
  https://blog.sqlauthority.com/2013/10/09/big-data-buzz-words-what-is-mapreduce-day-7-of-21/
  https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/Mapper.html
  https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/Reducer.html
  https://hadoop.apache.org/docs/current/api/org/apache/hadoop/mapreduce/Partitioner.html

- **Scaling Computing @Spotify**
  https://www.youtube.com/watch?v=cdsfRXr9pJU
  https://www.slideshare.net/RafaWojdya/the-evolution-of-hadoop-at-spotify-through-failures-and-pain
  https://www.slideshare.net/JoshBaer/how-apache-drives-music-recommendations-at-spotify

- **Evolution and Limits of Computation**
  Markov, I. Limits on fundamental limits to computation. Nature 512, 147–154 (2014).
  Published Article: https://doi.org/10.1038/nature13570 arXiv:1408.3821.
  Preprint: https://arxiv.org/abs/1408.3821 *(free access)*

Distributed Storage
[Hadoop & Friends]

INTERNET & DATA LAB