cloudera®
Ask Bigger Questions

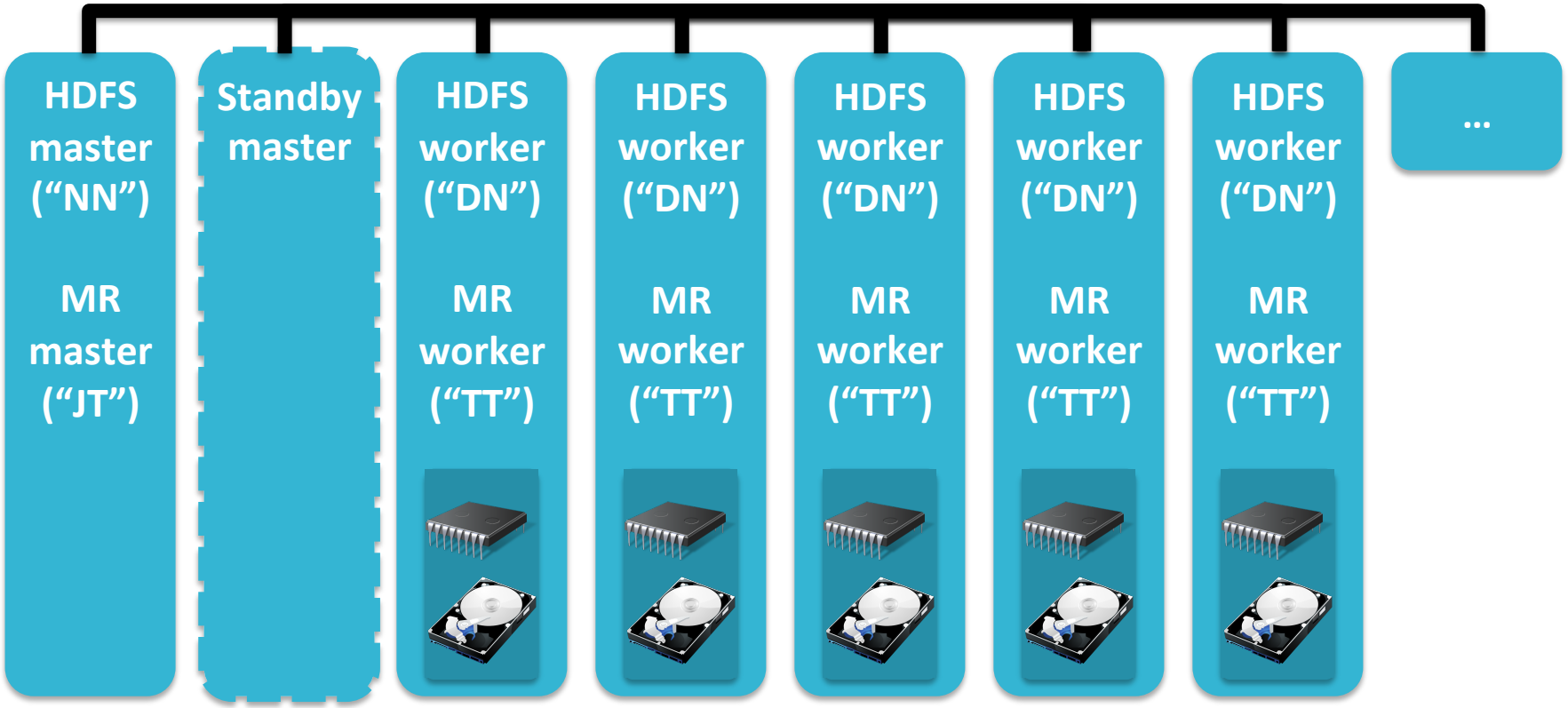
# Introduction to Spark

Ben White – Systems Engineer, Cloudera

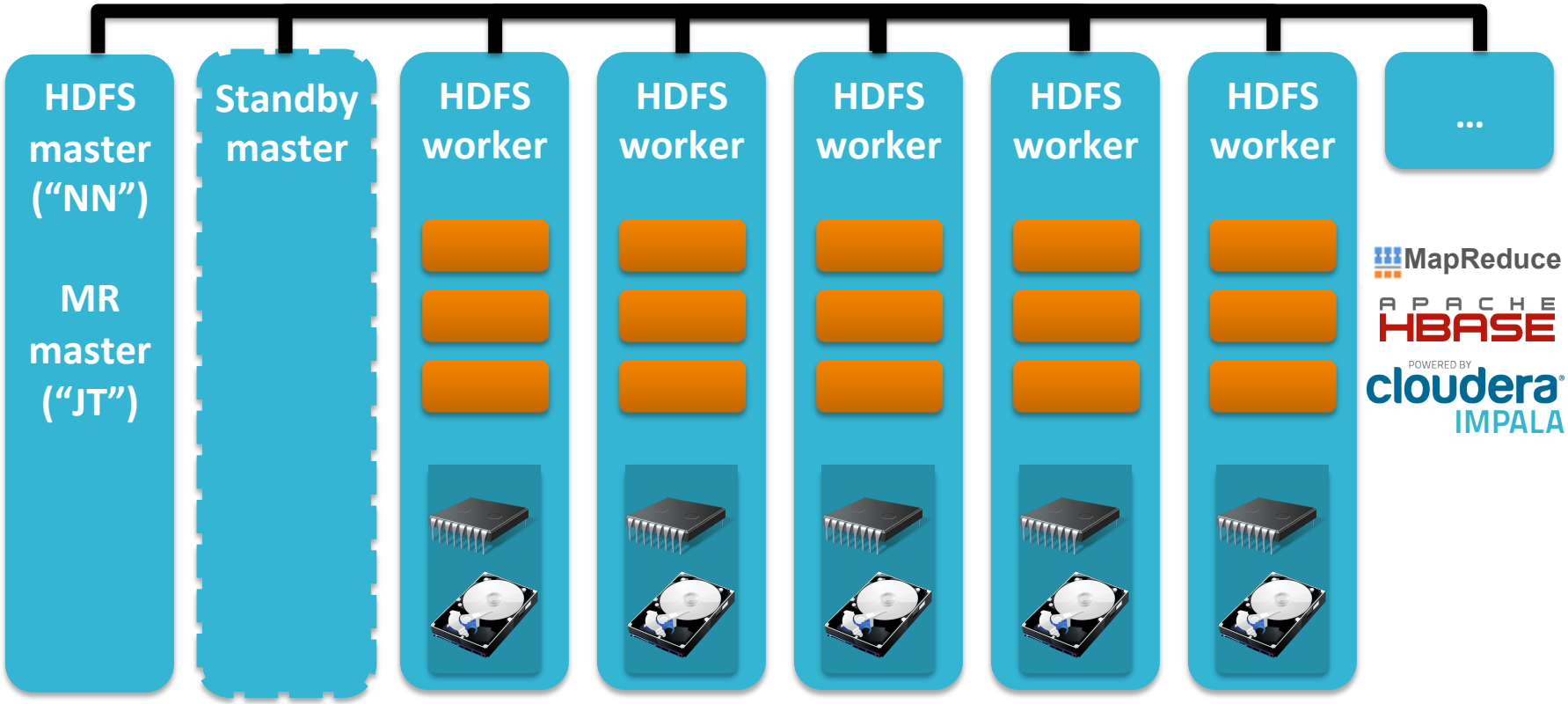
cloudera®
Ask Bigger Questions

# But first… how did we get here?

# What does Hadoop look like?

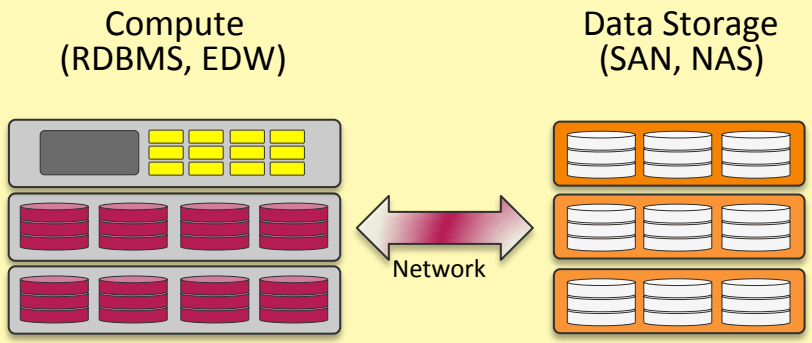


# But I want MORE!



# Hadoop as an Architecture

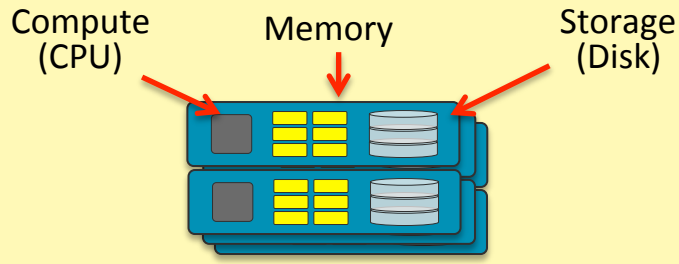
## The Old Way




**Expensive, Special purpose, "Reliable" Servers**
**Expensive Licensed Software**

- Hard to scale
- Network is a bottleneck
- Only handles relational data
- Difficult to add new fields & data types

## The Hadoop Way




**Commodity "Unreliable" Servers**
**Hybrid Open Source Software**

- Scales out forever
- No bottlenecks
- Easy to ingest any data
- Agile data access

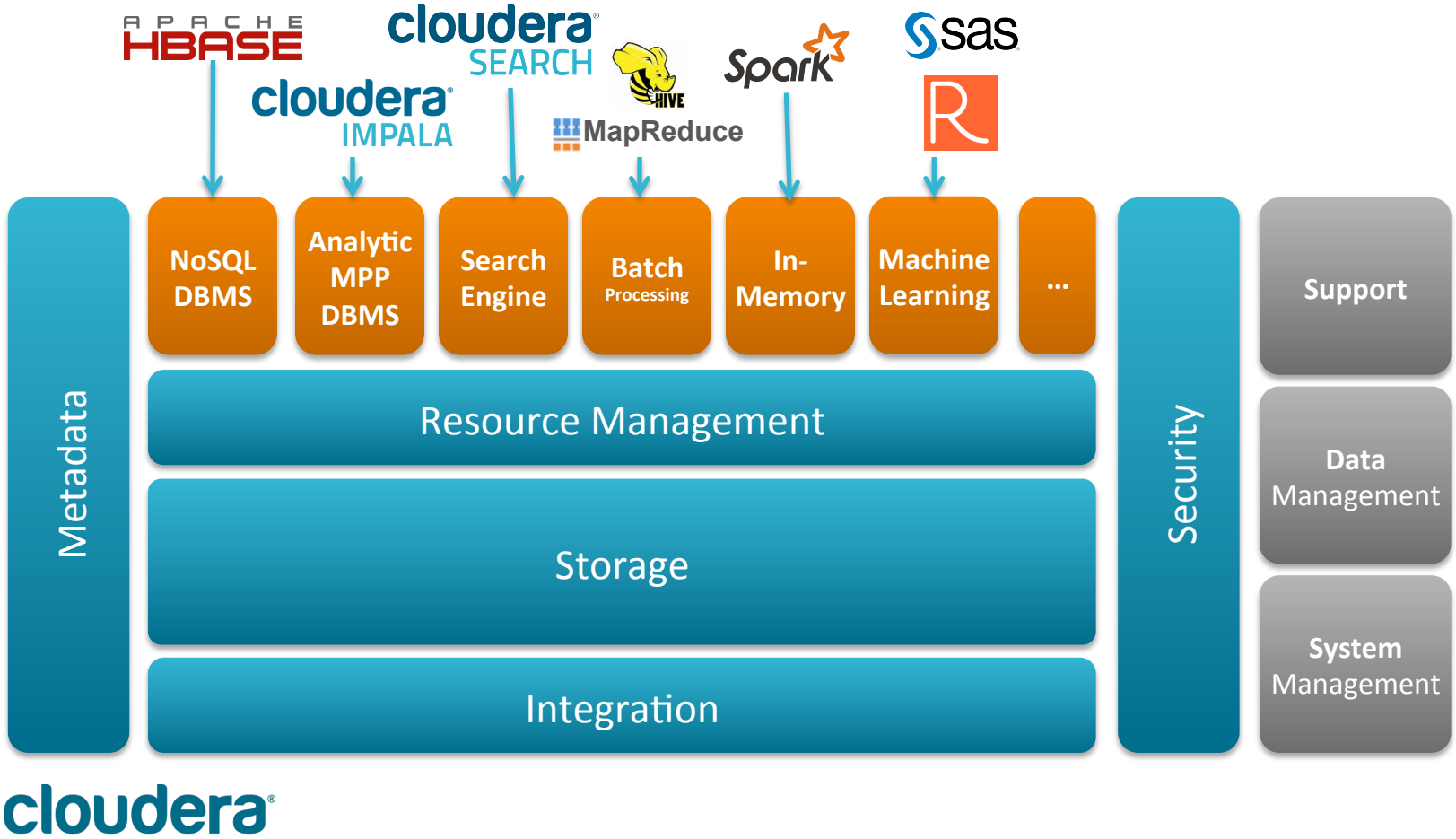
## Expensive & Unattainable

**$30,000+ per TB**

## Affordable & Attainable

**$300-$1,000 per TB**

# CDH: the App Store for Hadoop



cloudera
Ask Bigger Questions

# Introduction to Apache Spark

Credits:

- Todd Lipcon
- Ted Malaska
- Jairam Ranganathan
- Jayant Shekhar
- Sandy Ryza

# Can we improve on MR?

- Problems with MR:
  - **Very low-level**: requires a lot of code to do simple things
  - **Very constrained**: everything must be described as "map" and "reduce". Powerful but sometimes difficult to think in these terms.

# Can we improve on MR?

- Two approaches to improve on MapReduce:

1. Special purpose systems to solve one problem domain well.
   - Giraph / Graphlab (graph processing)
   - Storm (stream processing)

2. Generalize the capabilities of MapReduce to provide a richer foundation to solve problems.
   - Tez, MPI, Hama/Pregel (BSP), Dryad (arbitrary DAGs)

*Both are viable strategies depending on the problem!*

# What is Apache Spark?

Spark is a general purpose computational framework

**Retains the advantages of MapReduce:**

- Linear scalability
- Fault-tolerance
- Data Locality based computations


**…but offers so much more:**

- Leverages distributed memory for better performance
- Supports iterative algorithms that are not feasible in MR
- Improved developer experience
- Full Directed Graph expressions for data parallel computations
- Comes with libraries for machine learning, graph analysis, etc

# Getting started with Spark

- Java API
- Interactive shells:
  - Scala (spark-shell)
  - Python (pyspark)

**Python**

```
lines = sc.textFile(...)
lines.filter(lambda s: "ERROR" in s).count()
```

**Scala**

```
val lines = sc.textFile(...)
lines.filter(s => s.contains("ERROR")).count()
```

**Java**

```
JavaRDD<String> lines = sc.textFile(...);
lines.filter(new Function<String, Boolean>() {
  Boolean call(String s) {
    return s.contains("error");
  }
}).count();
```

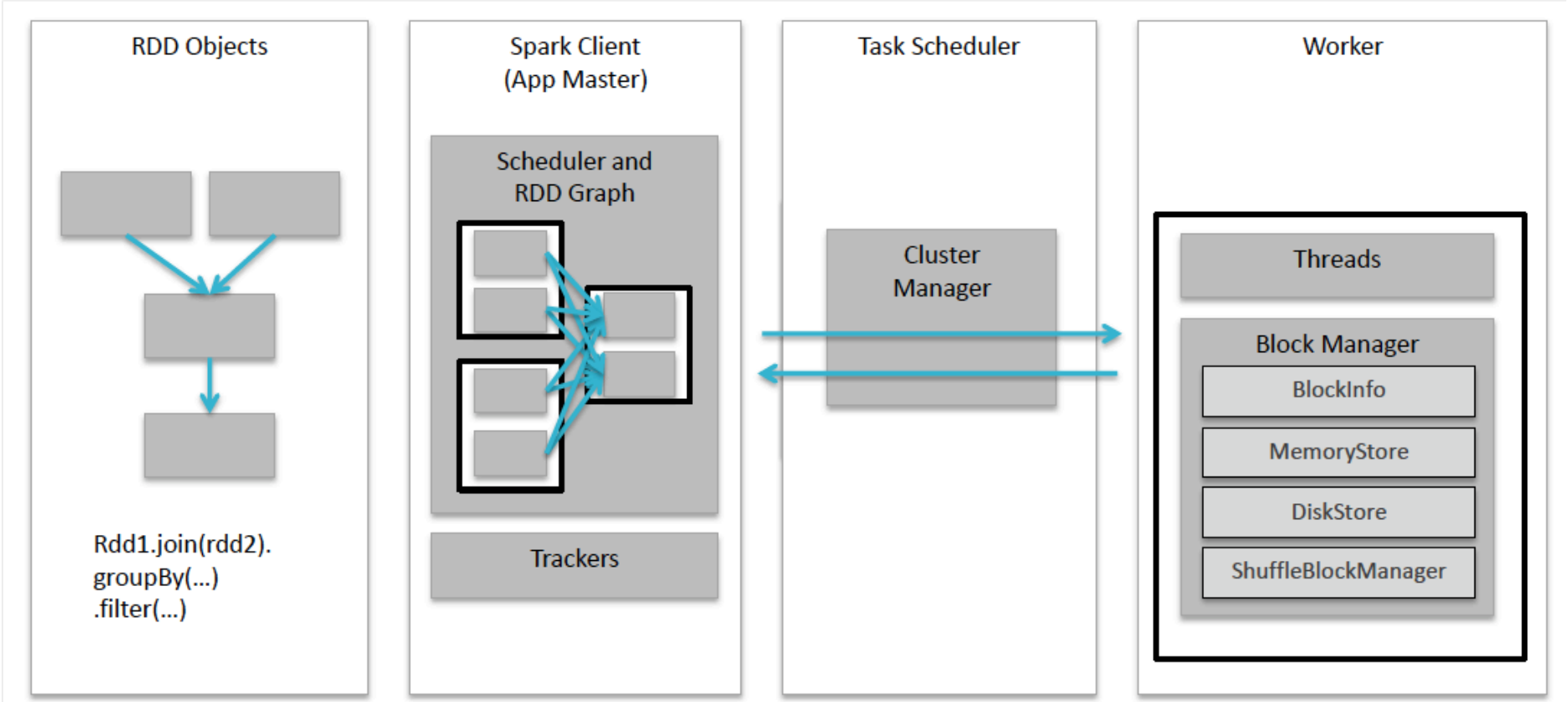
# Execution modes

- Standalone Mode
  - Dedicated master and worker daemons
- YARN Client Mode
  - Launches a YARN application with the driver program running locally
- YARN Cluster Mode
  - Launches a YARN application with the driver program running in the YARN ApplicationMaster

Dedicated Spark runtime with static resource limits

Dynamic resource management between Spark, MR, Impala…

# Spark Concepts



# Parallelized Collections

```
scala> val data = 1 to 5
data: Range.Inclusive = Range(1, 2, 3, 4, 5)

scala> val distData = sc.parallelize(data)
distData: org.apache.spark.rdd.RDD[Int] =
                                        ParallelCollectionRDD[0]
```

Now I can apply parallel operations to this array:

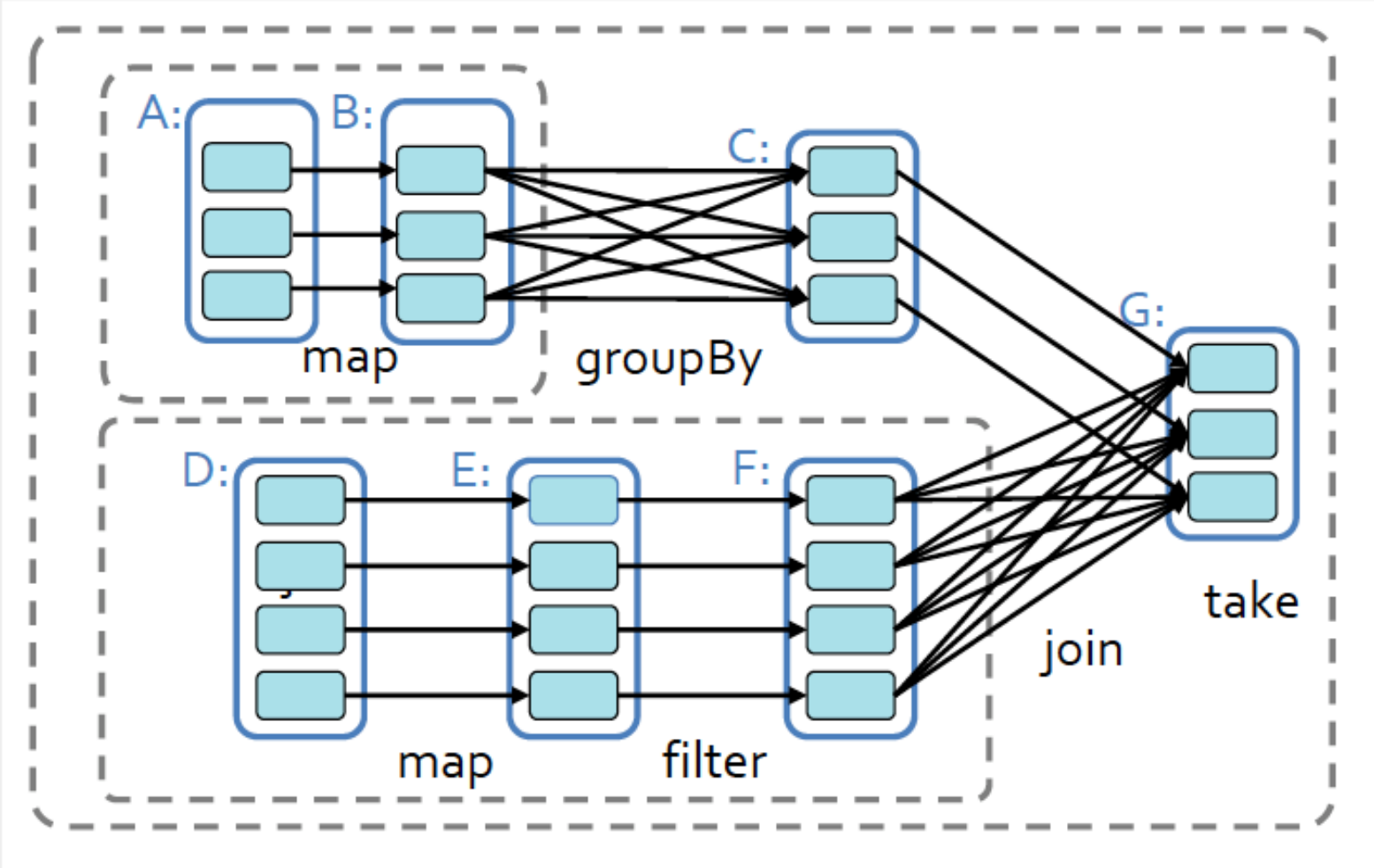
```
scala> distData.reduce(_ + _)
[… Adding task set 0.0 with 56 tasks …]
res0: Int = 15
```

What just happened?!

# RDD – Resilient Distributed Dataset

- Collections of objects partitioned across a cluster
  - Stored in RAM or on Disk
  - You can control persistence and partitioning
- Created by:
  - Distributing local collection objects
  - Transformation of data in storage
  - Transformation of RDDs
- Automatically rebuilt on failure (resilient)
  - Contains lineage to compute from storage
  - Lazy materialization

# RDD transformations



# Operations on RDDs

**Transformations** lazily transform a RDD to a new RDD

- map
- flatMap
- filter
- sample
- join
- sort
- reduceByKey
- …

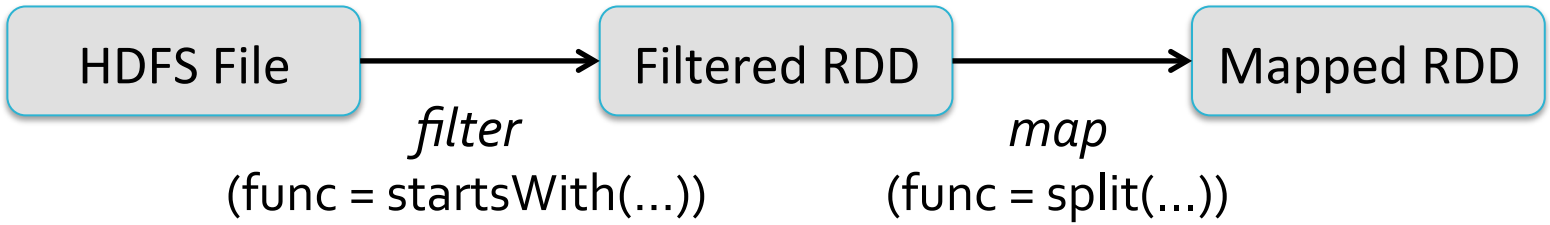
**Actions** run computation to return a value

- collect
- reduce(func)
- foreach(func)
- count
- first, take(n)
- saveAs
- …

# Fault Tolerance

- RDDs contain lineage.
- Lineage – source location and list of transformations
- Lost partitions can be re-computed from source data

```
msgs = textFile.filter(lambda s: s.startsWith("ERROR"))
               .map(lambda s: s.split("\t")[2])
```



cloudera®
Ask Bigger Questions

# Examples

# Word Count in MapReduce

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class WordCount {

 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

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

```
 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

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

 public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();

      Job job = new Job(conf, "wordcount");

   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);

   job.setMapperClass(Map.class);
   job.setReducerClass(Reduce.class);

   job.setInputFormatClass(TextInputFormat.class);
   job.setOutputFormatClass(TextOutputFormat.class);

   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));

   job.waitForCompletion(true);
 }

}
```

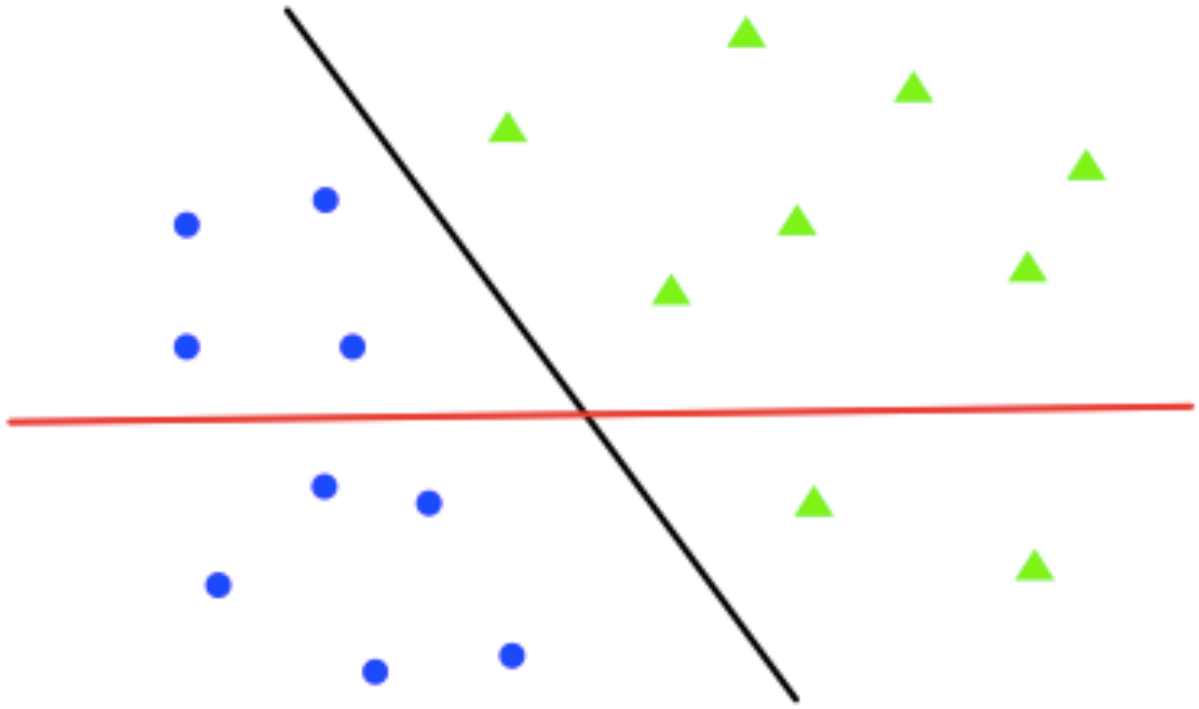
# Word Count in Spark

```
sc.textFile(“words”)
    .flatMap(line => line.split(" "))
    .map(word=>(word,1))
    .reduceByKey(_+_).collect()
```

# Logistic Regression

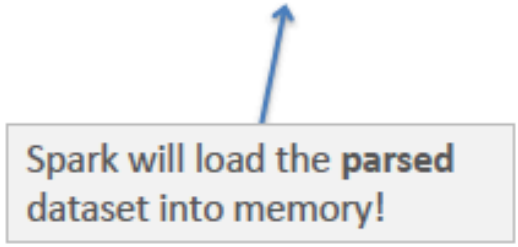
- Read two sets of points
- Looks for a plane W that separates them
- Perform gradient descent:
  - Start with random W
  - On each iteration, sum a function of W over the data
  - Move W in a direction that improves it

# Intuition

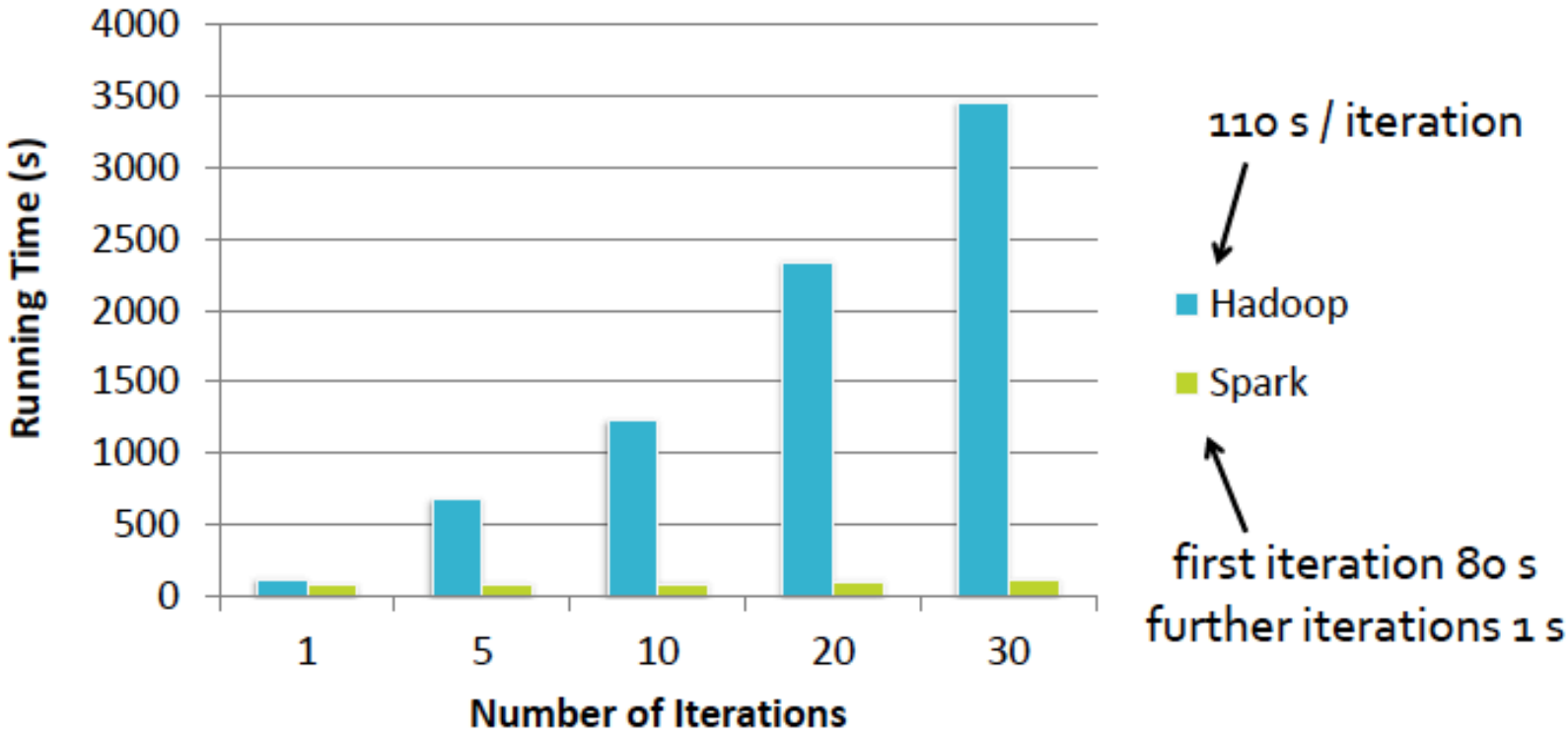


# Logistic Regression

```
data = spark.textFile(...).map(parseDataPoint).cache()

w = numpy.random.rand(D)

for i in range(iterations):
    gradient = data
     .map(lambda p: (1 / (1 + exp(-p.y * w.dot(p.x))))
          * p.y * p.x)
     .reduce(lambda x, y: x + y)
    w -= gradient

print "Final w: %s" % w
```

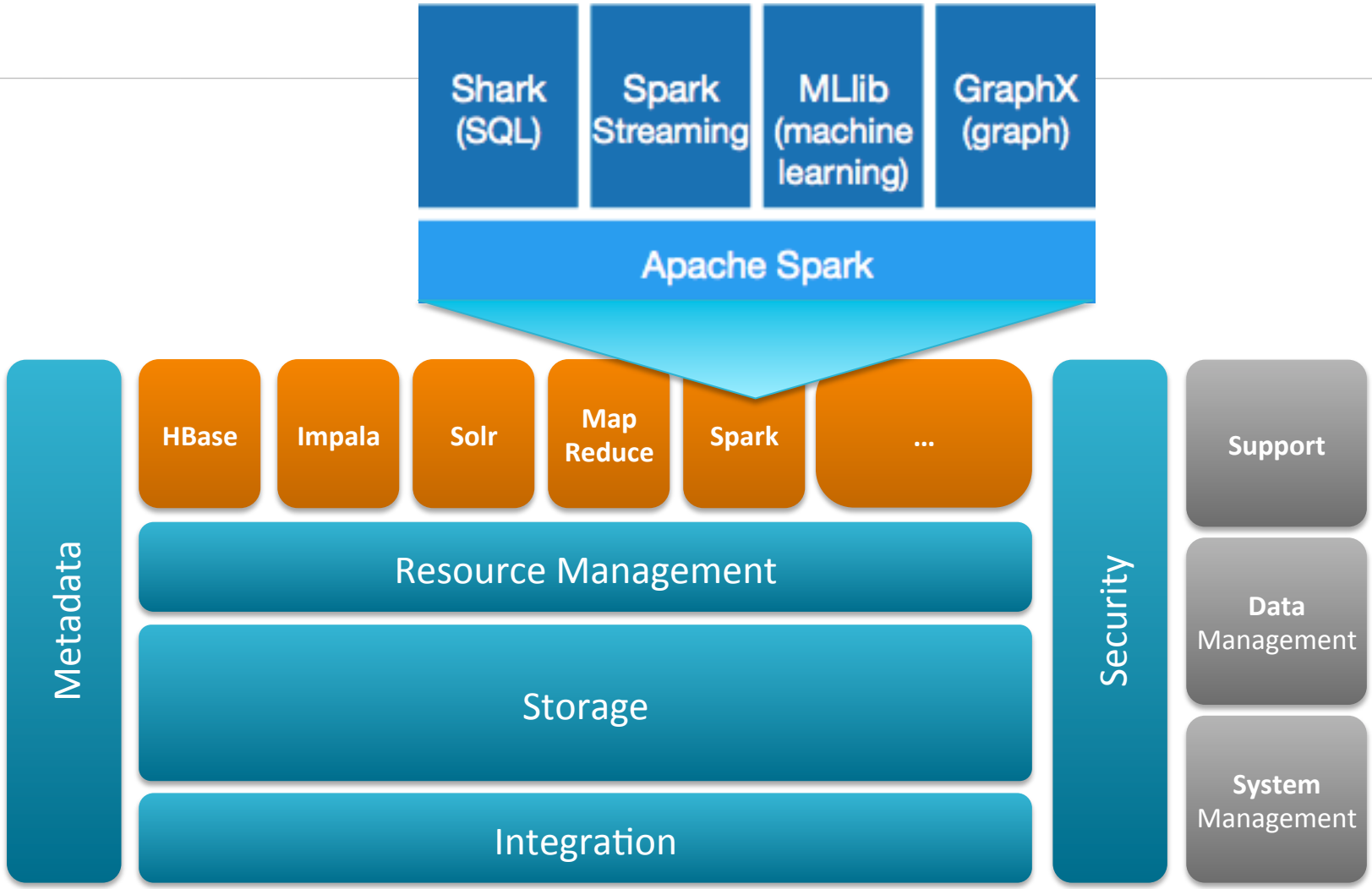


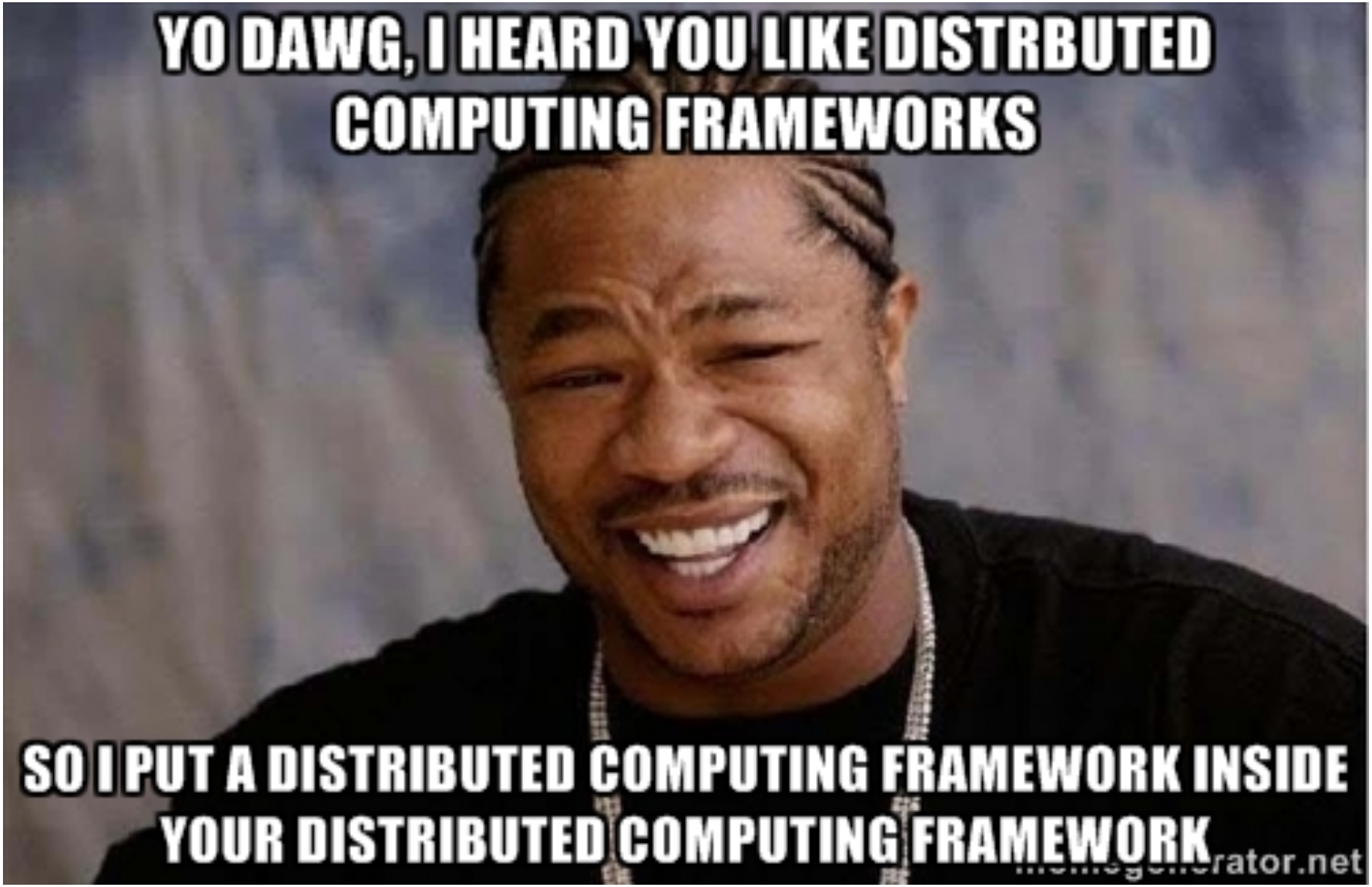
# Logistic Regression Performance

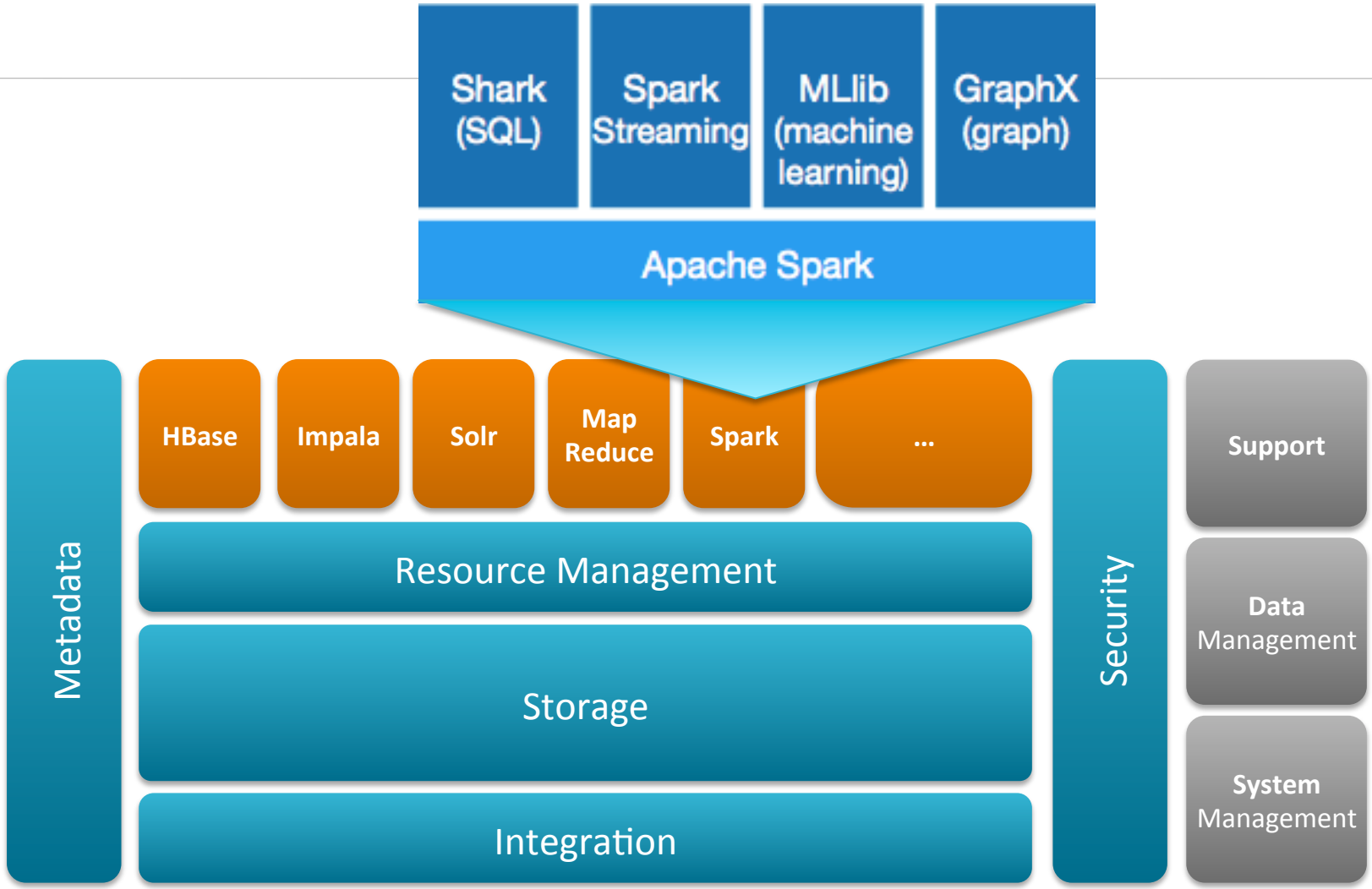


cloudera
Ask Bigger Questions

# Spark and Hadoop: a Framework within a Framework





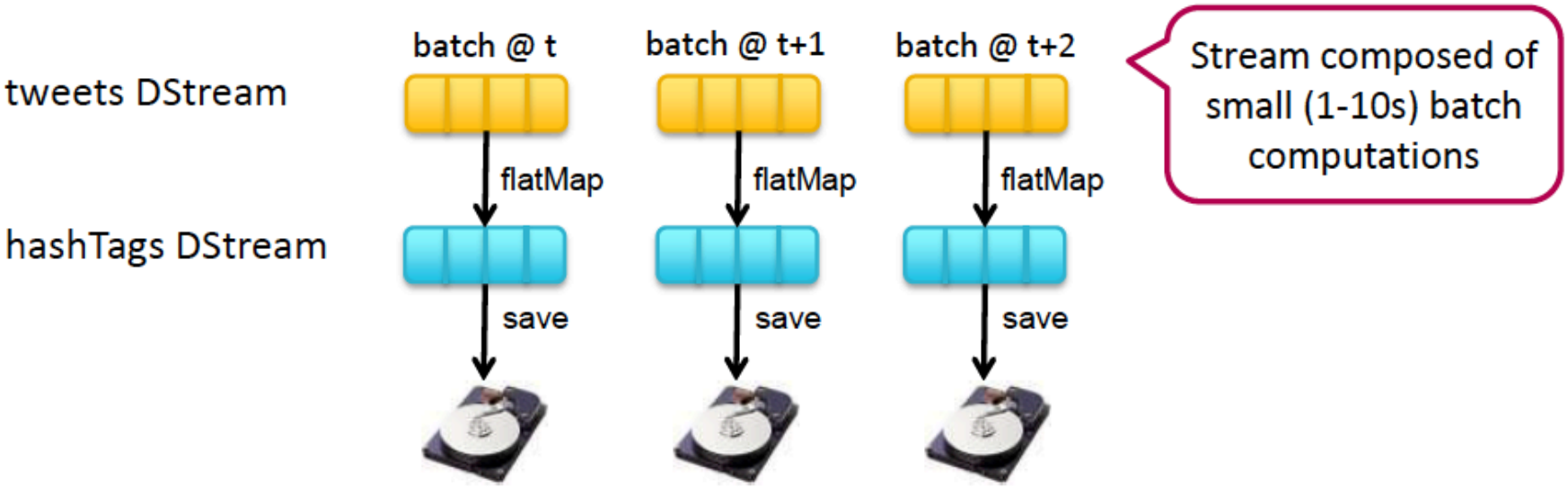


# Spark Streaming

- Takes the concept of RDDs and extends it to *DStreams*
  - Fault-tolerant like RDDs
  - Transformable like RDDs
- Adds new "rolling window" operations
  - Rolling averages, etc
- But keeps everything else!
  - Regular Spark code works in Spark Streaming
  - Can still access HDFS data, etc

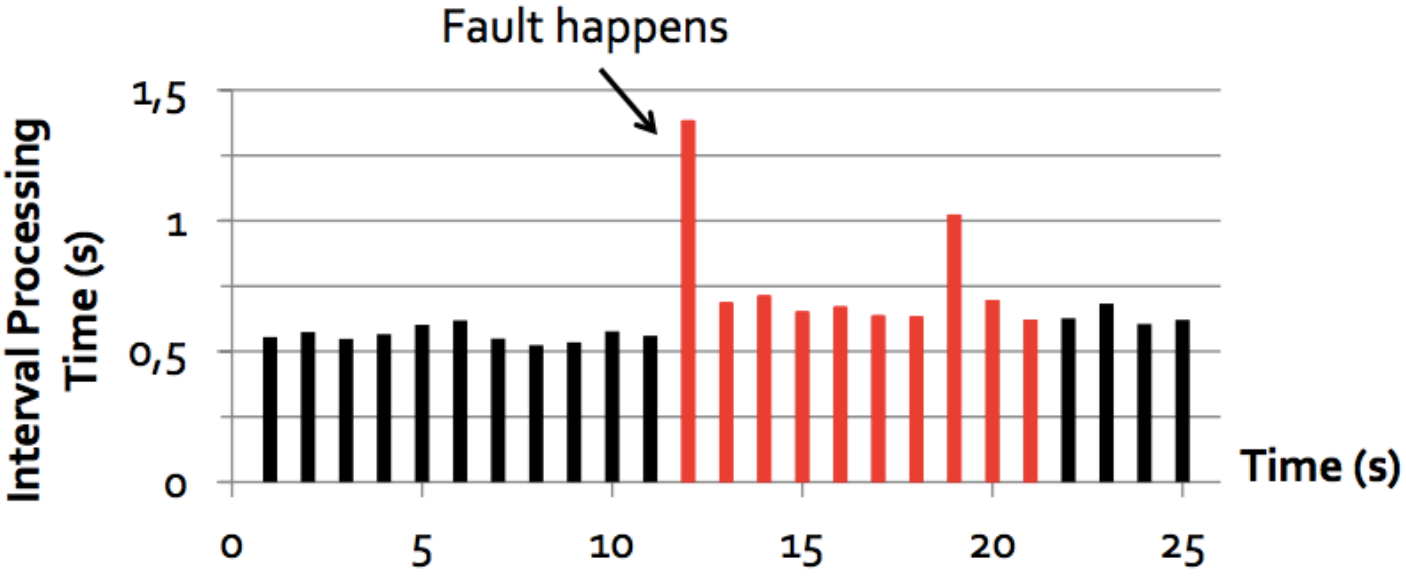
# Micro-batching for on the fly ETL

```
val tweets = ssc.twitterStream()
val hashTags = tweets.flatMap (status => getTags(status))
hashTags.saveAsHadoopFiles("hdfs://...")
```



# Fault recovery

## How fast can the system recover?




Sliding WordCount on 20 nodes with 10s checkpoint interval

# Fault Recovery

- RDDs store dependency graph
- Because RDDs are deterministic:
  Missing RDDs are rebuilt **in parallel** on other nodes
- Stateful RDDs can have infinite lineage
- Periodic checkpoints to disk clears lineage
- Faster recovery times
- Better handling of stragglers vs row-by-row streaming

cloudera®
Ask Bigger Questions

# Summary

# Why Spark?

- Flexible like MapReduce
- High performance
- Machine learning, iterative algorithms
- Interactive data explorations
- Concise, easy API for developer productivity

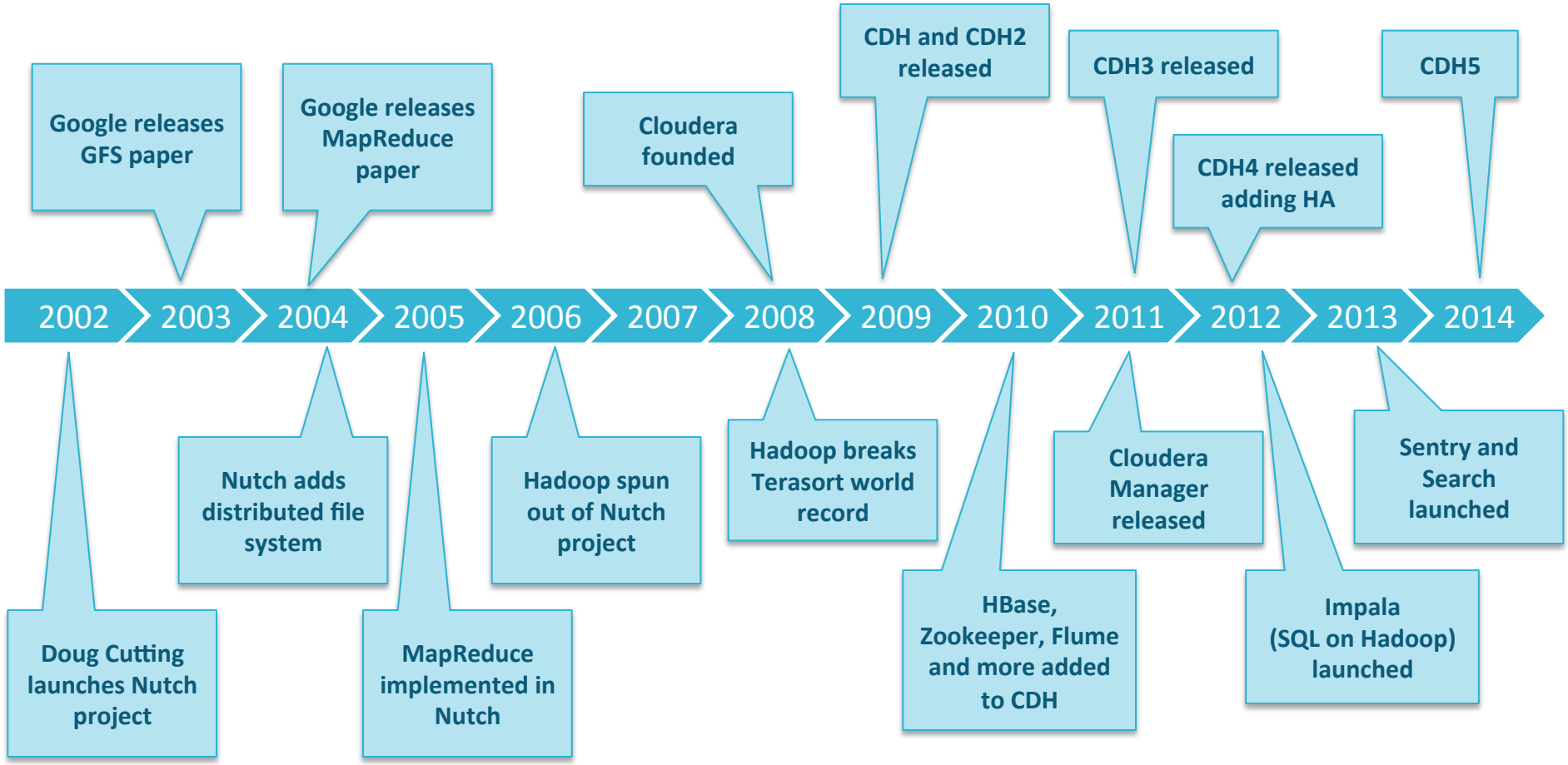

cloudera®
Ask Bigger Questions

# Spark

http://www.cloudera.com/content/cloudera/en/products-and-services/cdh/spark.html

http://www.cloudera.com/content/cloudera-content/cloudera-docs/CM5/latest/Cloudera-Manager-Installation-Guide/cm5ig_install_spark.html

# A Brief History



# What is Apache Hadoop?

- An open-source implementation of Google's GFS and MapReduce papers
- An Apache Software Foundation top-level project
- Good at storing and processing all kinds of data
- Reliable storage at terabyte/petabyte-scale on unreliable (cheap) hardware
- A distributed system for counting words ☺

# What is Apache Hadoop?

**Apache Hadoop** is an open source platform for data storage and processing that is…

- ✓ Scalable
- ✓ Fault tolerant
- ✓ Distributed

CORE HADOOP SYSTEM COMPONENTS


## Has the Flexibility to Store and Mine Any Type of Data

- Ask questions across structured and unstructured data that were previously impossible to ask or solve
- Not bound by a single schema

## Excels at Processing Complex Data

- Scale-out architecture divides workloads across multiple nodes
- Flexible file system eliminates ETL bottlenecks

## Scales Economically

- Can be deployed on industry standard hardware
- Open source platform guards against vendor lock