Big Data Engineering With MapReduce and Hive

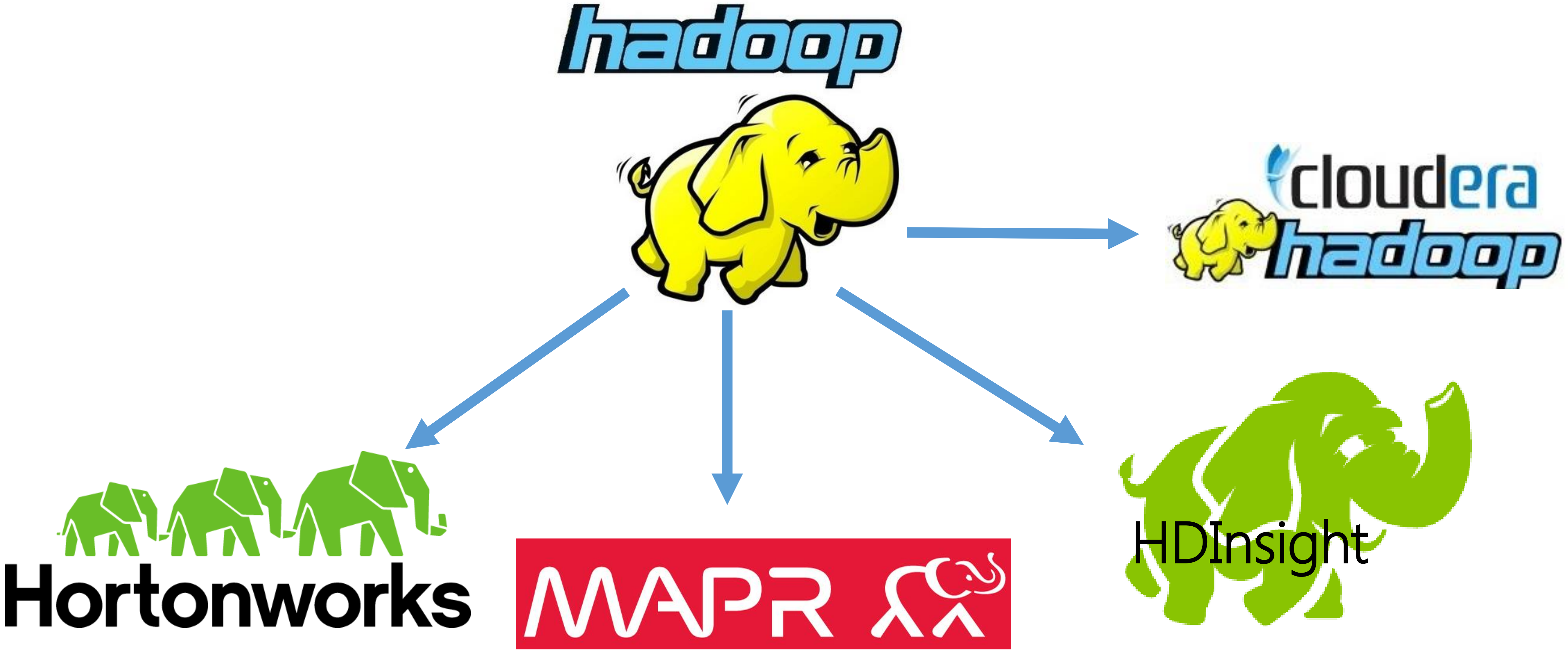
datasciencedojo
unleash the data scientist in you

Agenda

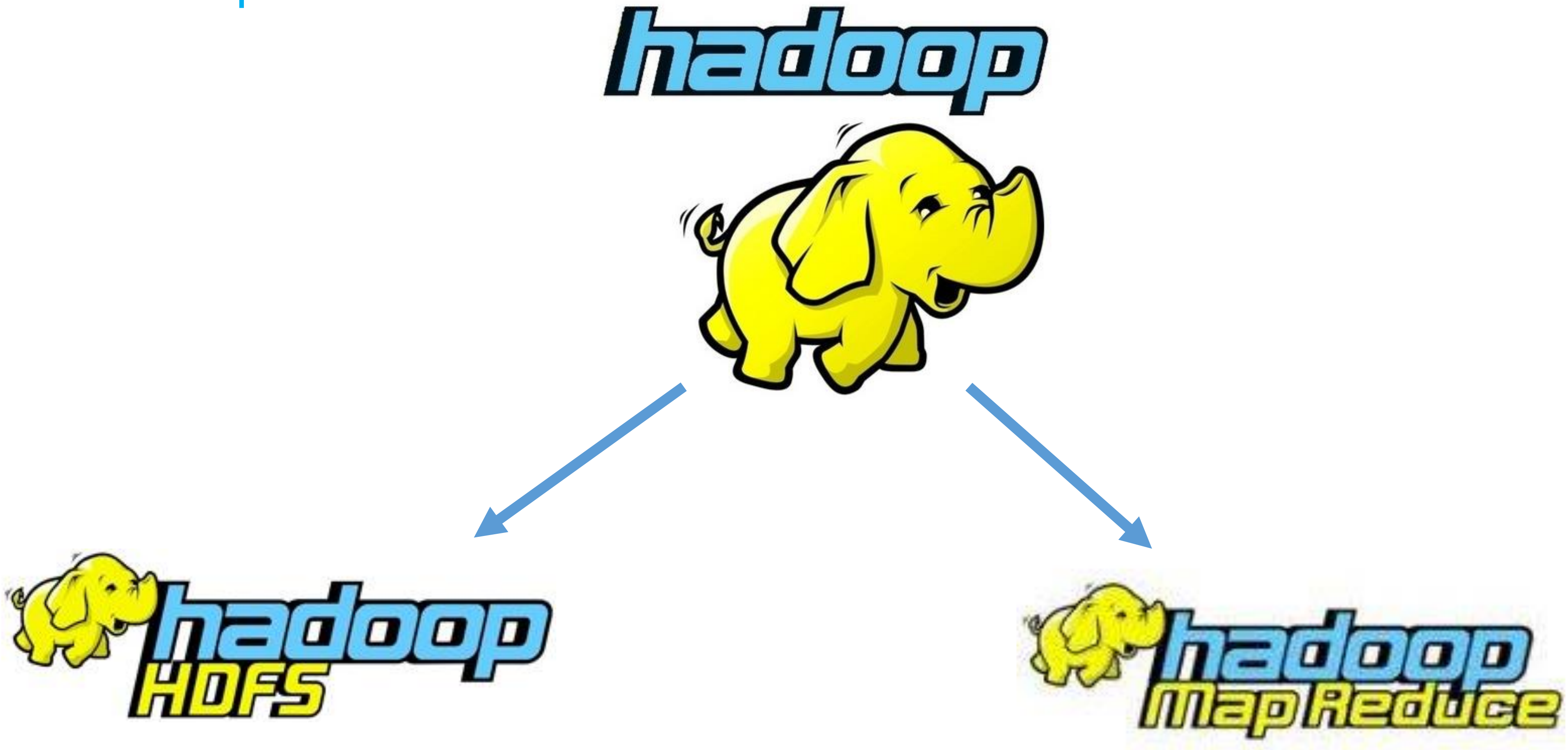
hadoop

HIVE

Hadoop Implementations

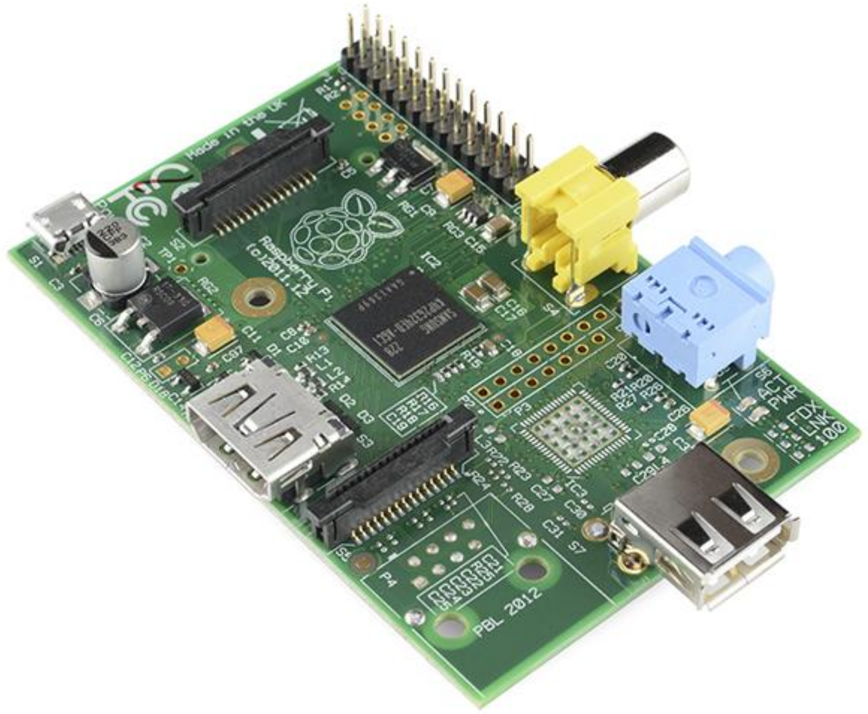


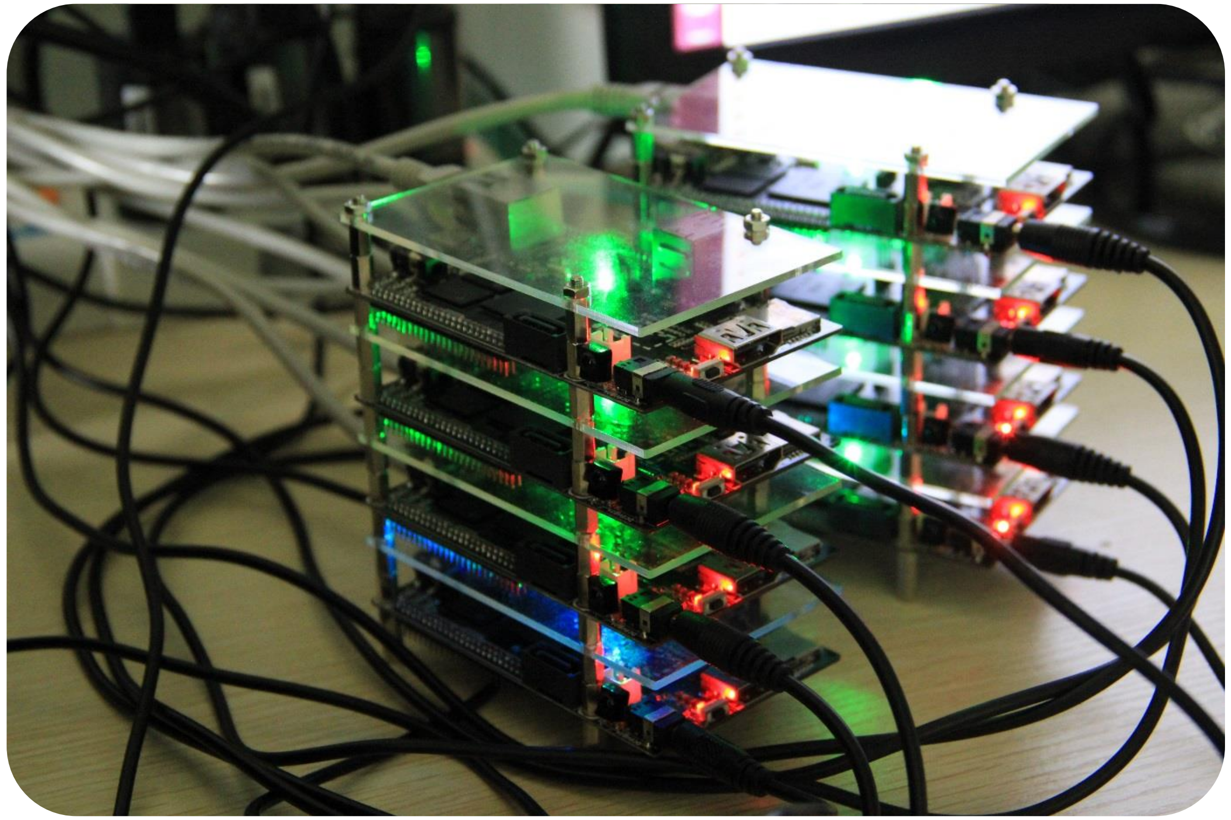
Hadoop



Turn back the clock, the mainframe

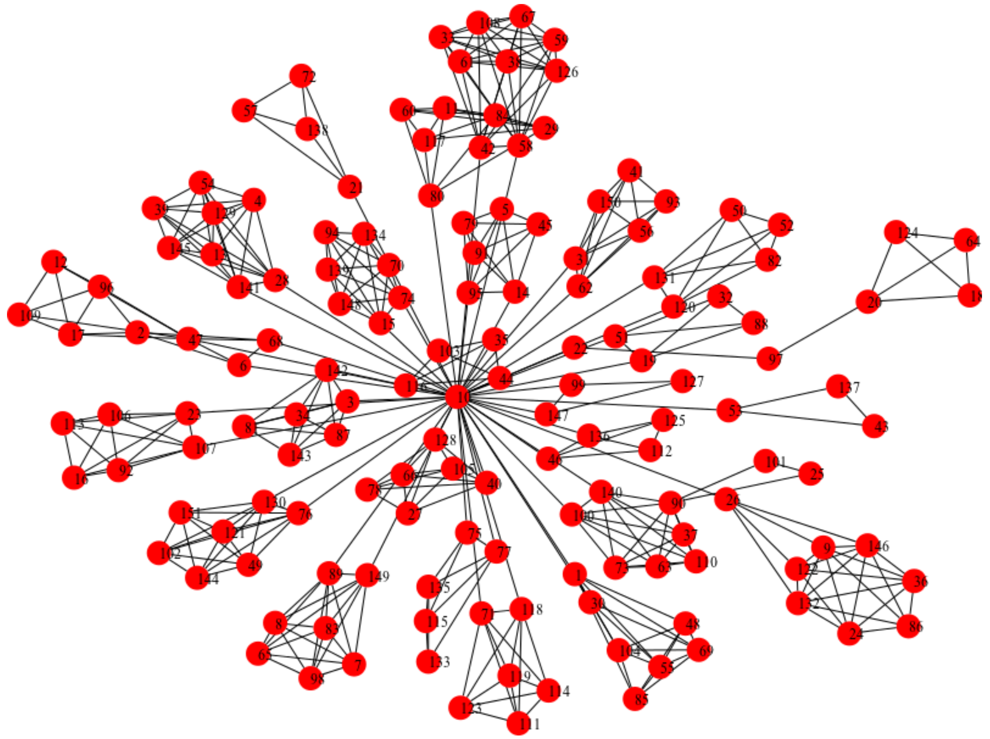
Distributed Computing





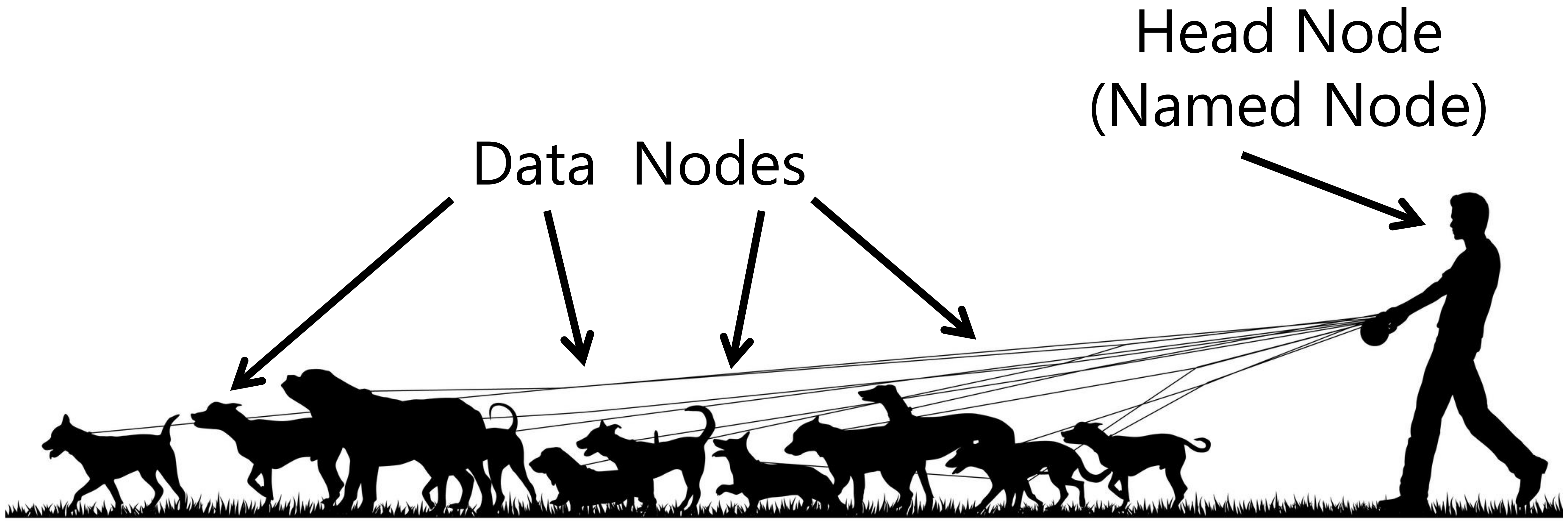
datasciencedojo
unleash the data scientist in you

Cloud Computing



datasciencedojo
unleash the data scientist in you

If dogs were servers…



datasciencedojo
unleash the data scientist in you

HDFS & MapReduce

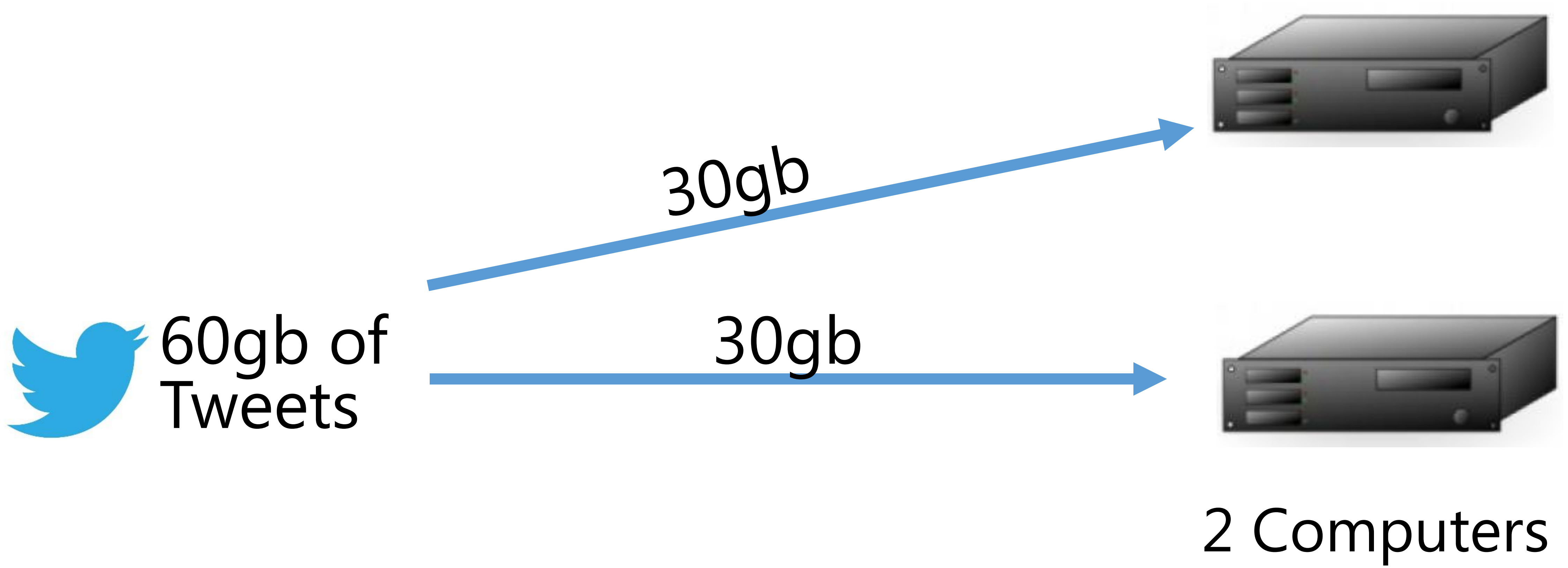
60gb of Tweets

$\xrightarrow{\text{60gb}}$

1 Computer

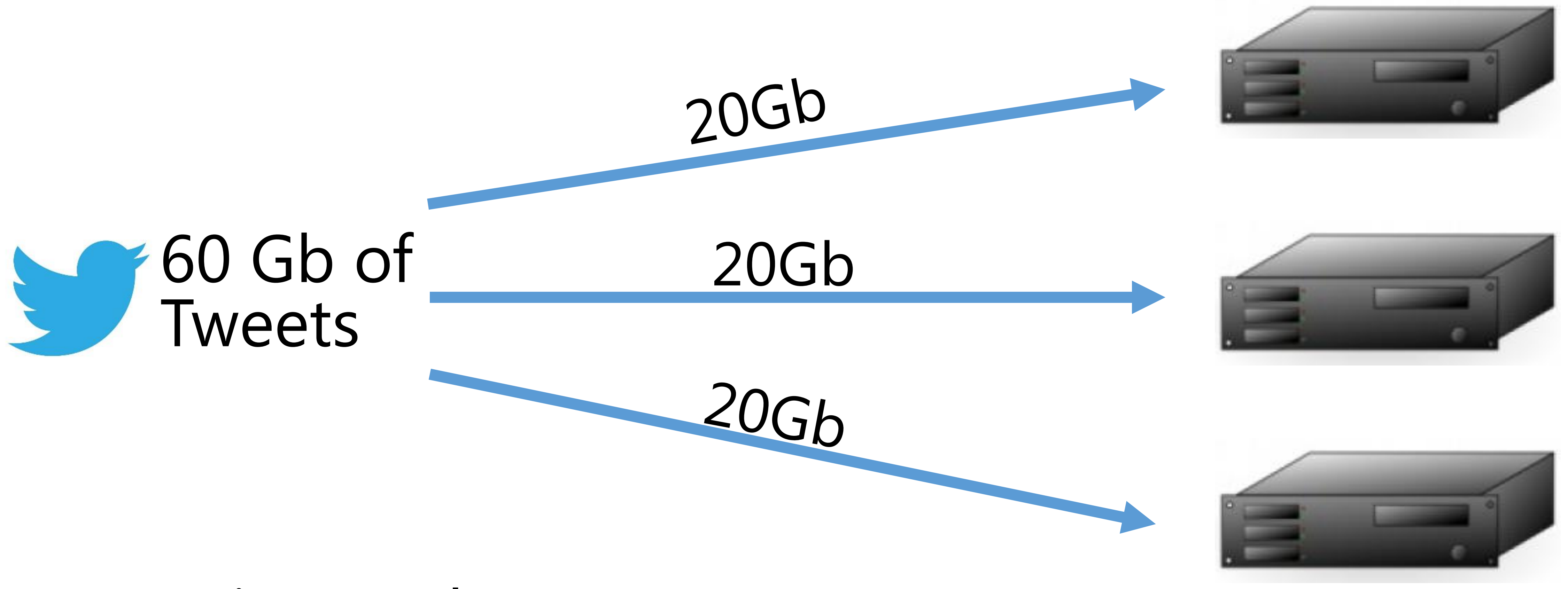
Processing: 30 hours

HDFS & MapReduce



Processing: 15 hours

HDFS & MapReduce



Processing: 10 hours

3 Computers

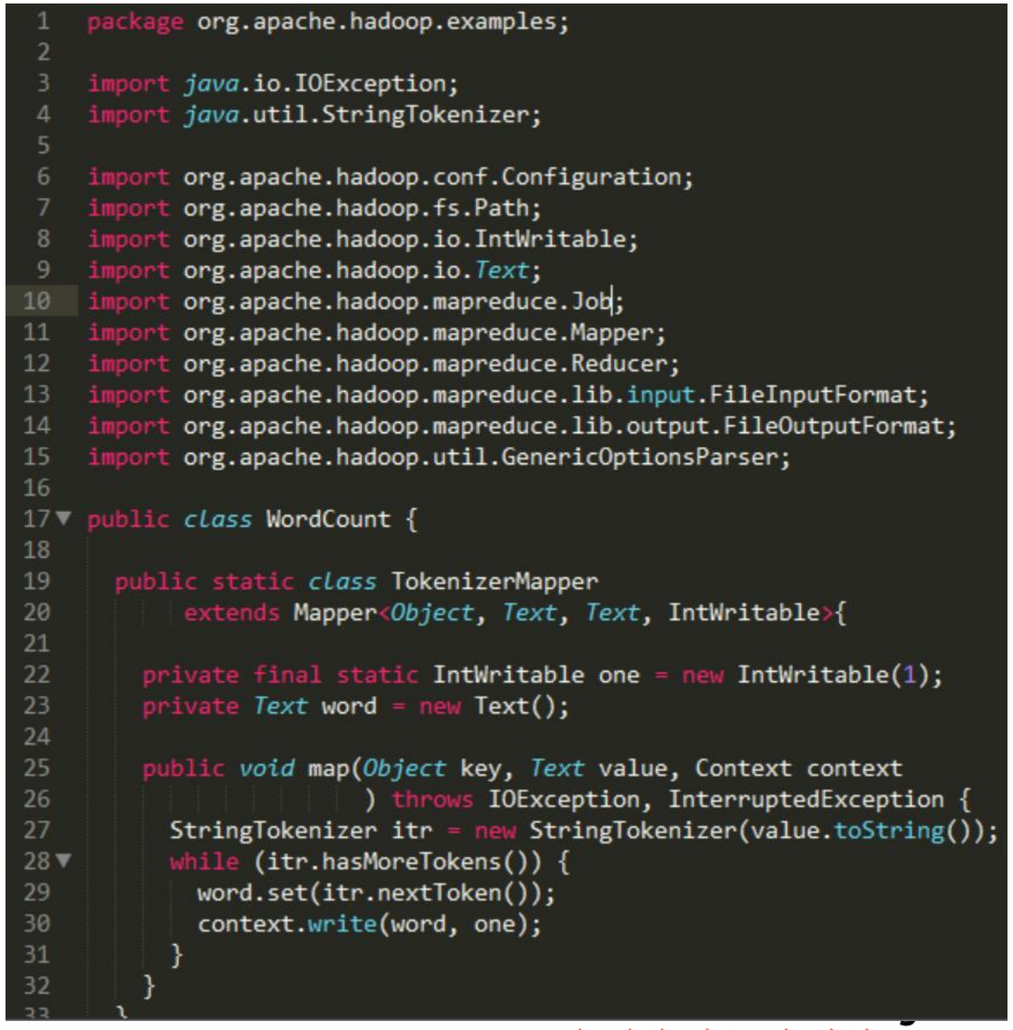
datasciencedojo
unleash the data scientist in you

Most cases, linear scaling of processing power

Number of Computers	Processing Time (hours)
1	30
2	15
3	10
4	7.5
5	6
6	5
7	4.26
8	3.75
9	3.33

Limitations with MapReduce

- ~200 lines of code to do anything
- Slow
- Troubleshooting multiple computers
- Good devs are scarce
- Expensive certifications



```java
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

  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);
      }
    }
```

Ambari: Cluster provisioning, management, and monitoring

Avro (Microsoft .NET Library for Avro): Data serialization for the Microsoft .NET environment

HBase: Non-relational database for very large tables

HDFS: Hadoop Distributed File System

Hive: SQL-like querying

Mahout: Machine learning

MapReduce and YARN: Distributed processing and resource management

Oozie: Workflow management

Pig: Simpler scripting for MapReduce transformations

Sqoop: Data import and export

Storm: Real-time processing of fast, large data streams

Zookeeper: Coordinates processes in distributed systems

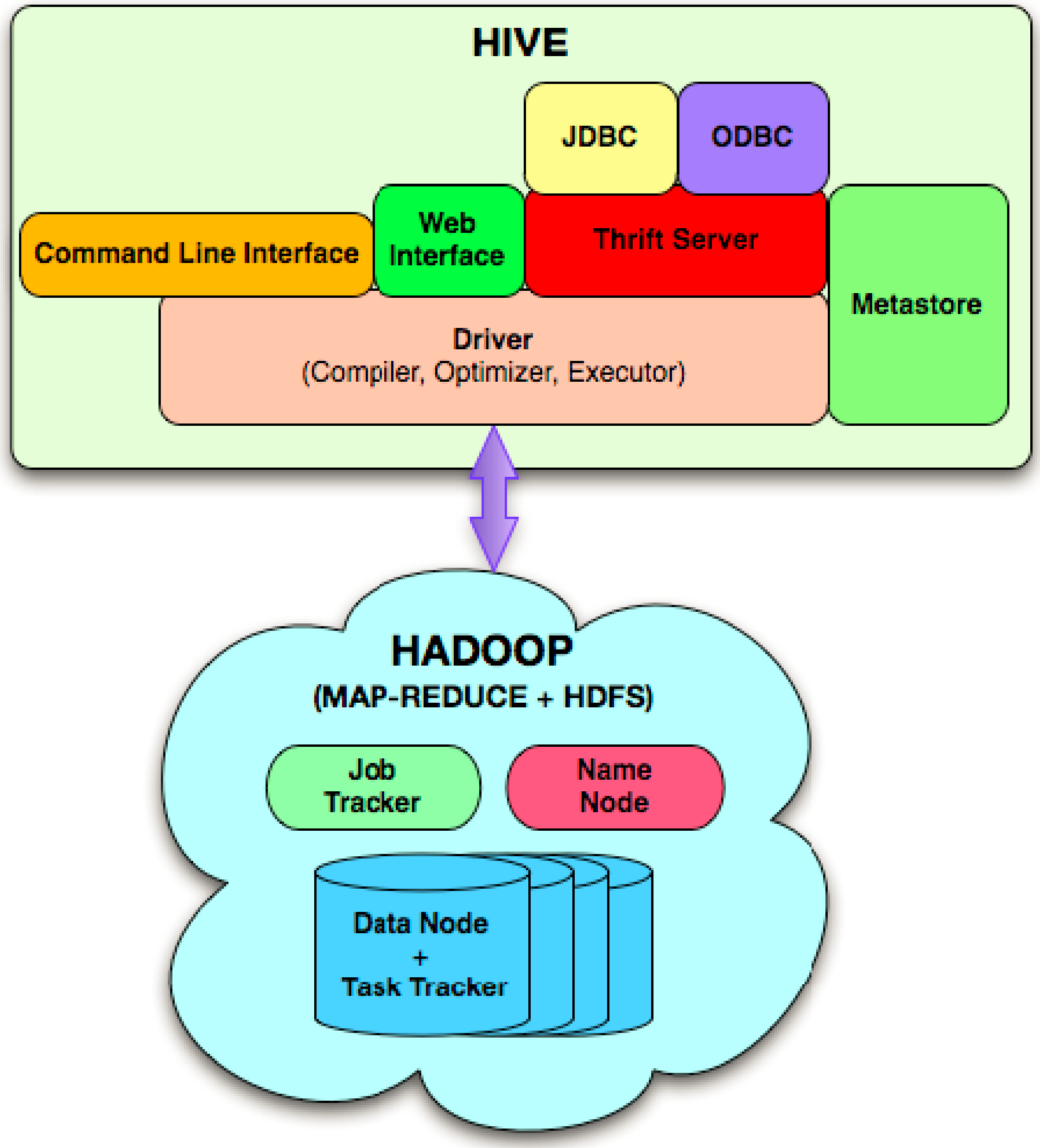
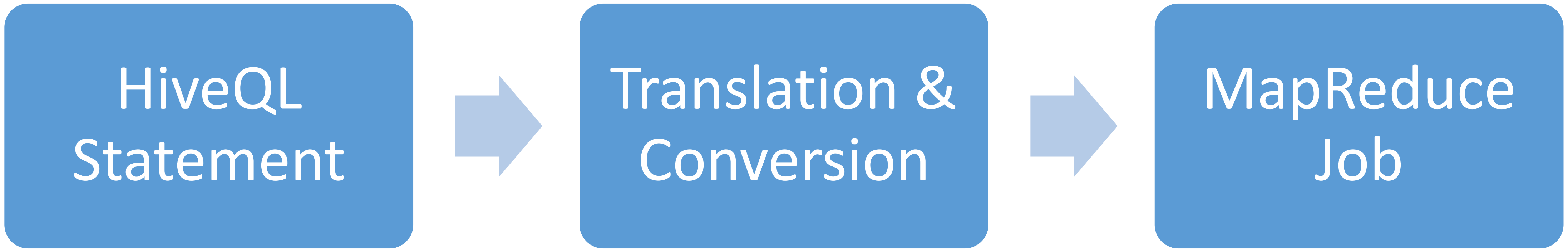
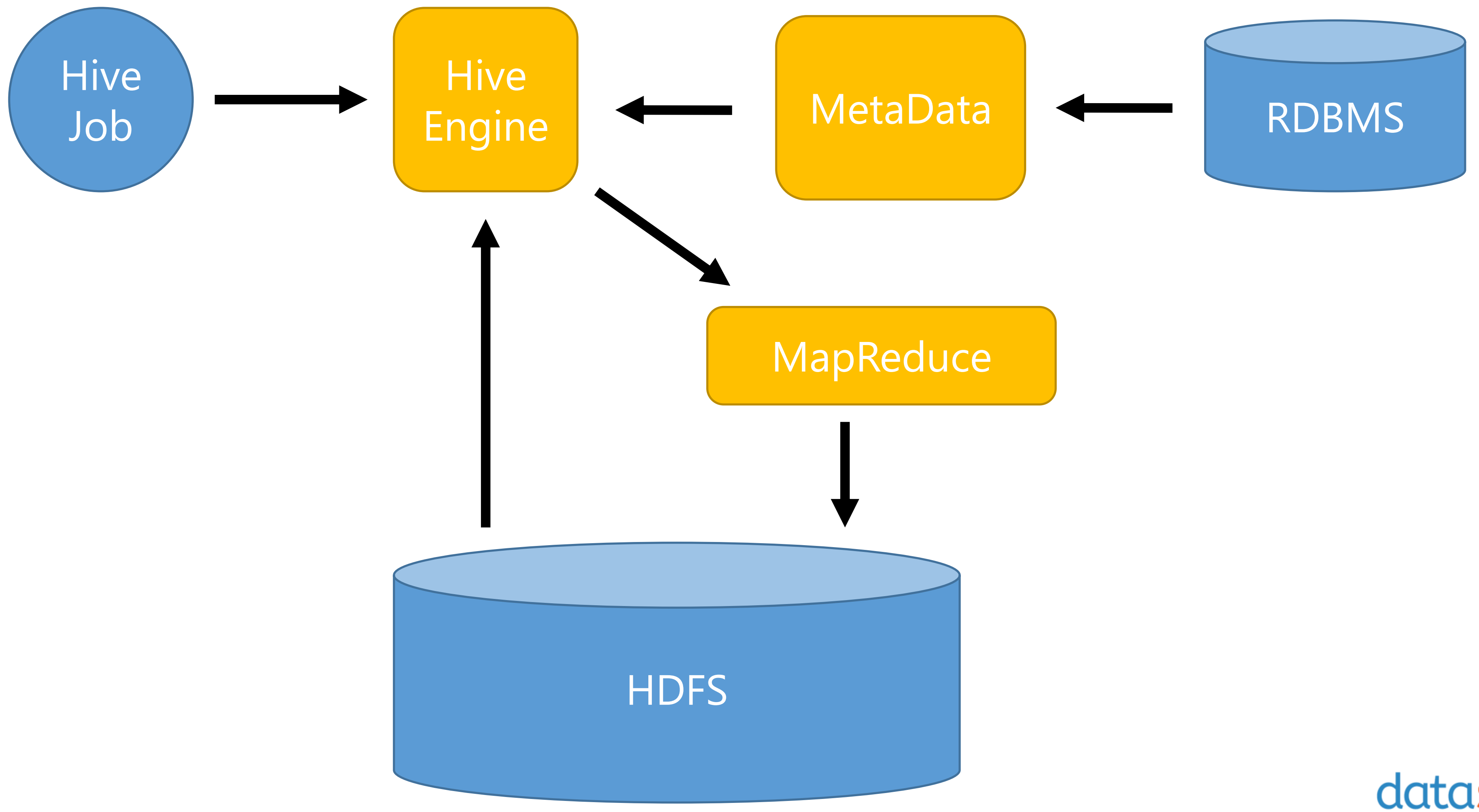


Fig. 1: Hive System Architecture

Hive Jobs



Hive Architecture



datasciencedojo
unleash the data scientist in you

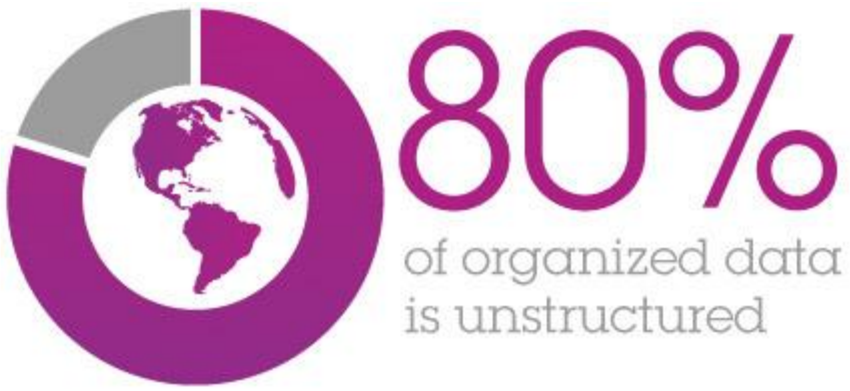
Data File = Unstructured Data

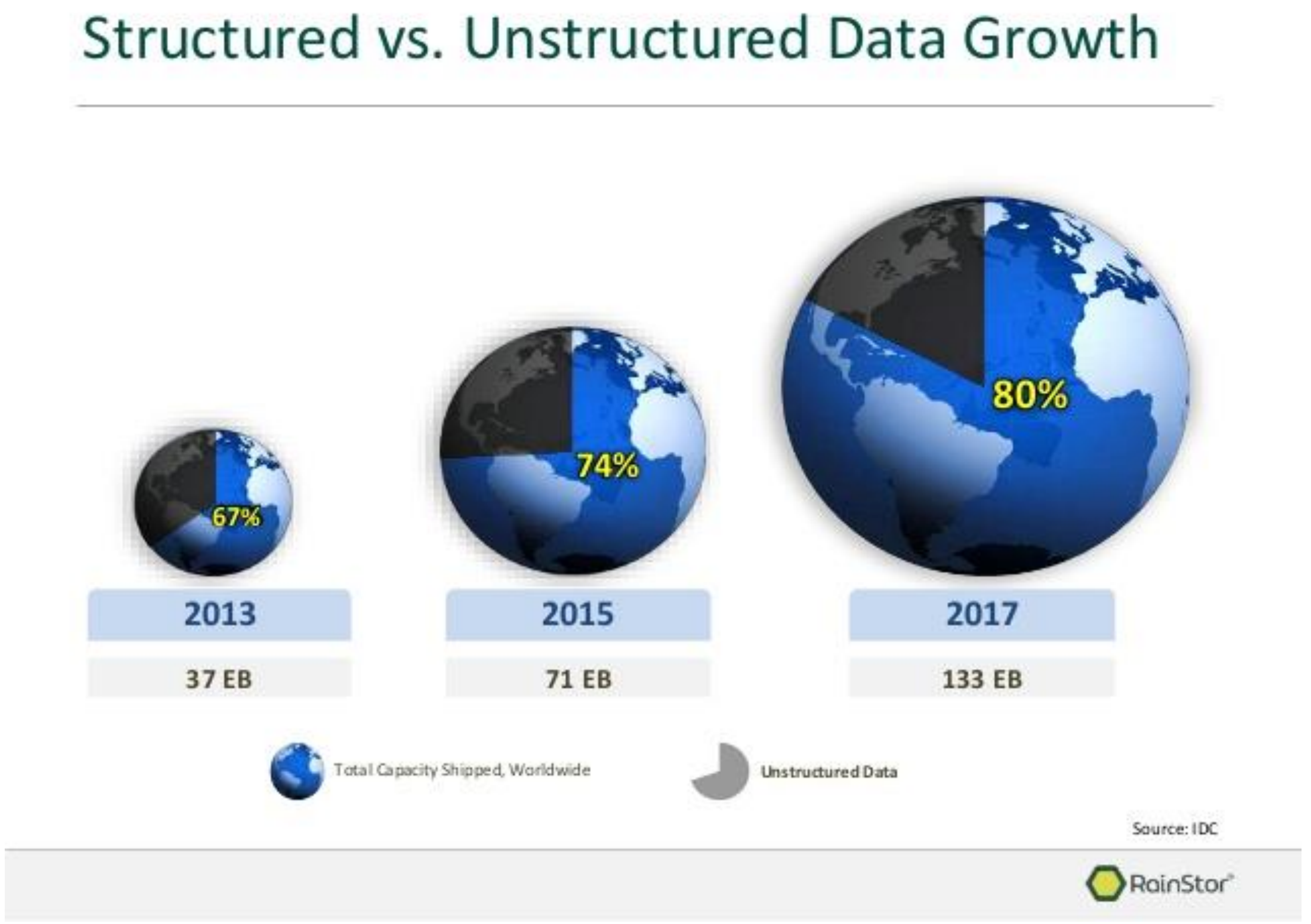
Data File + Metadata File/DB = Structured Data

Why Hive?

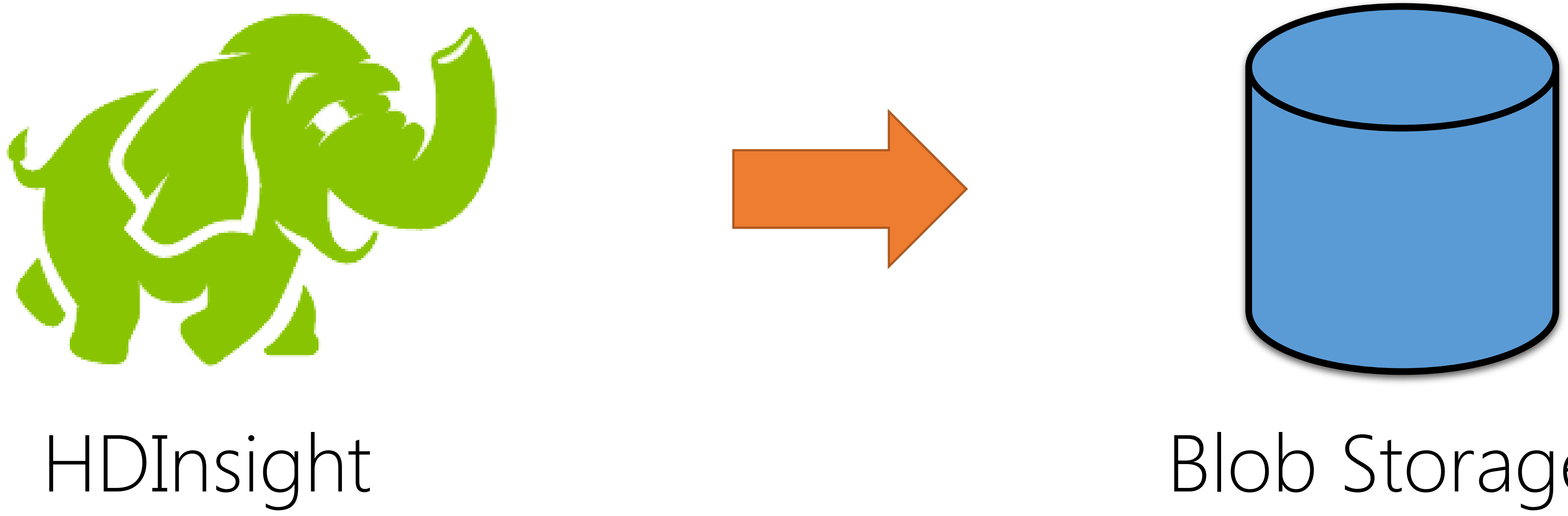
- SQL spoken here (HiveQL)
- ODBC driver
- BI Integration
- Supports only Structured Data

Limitations

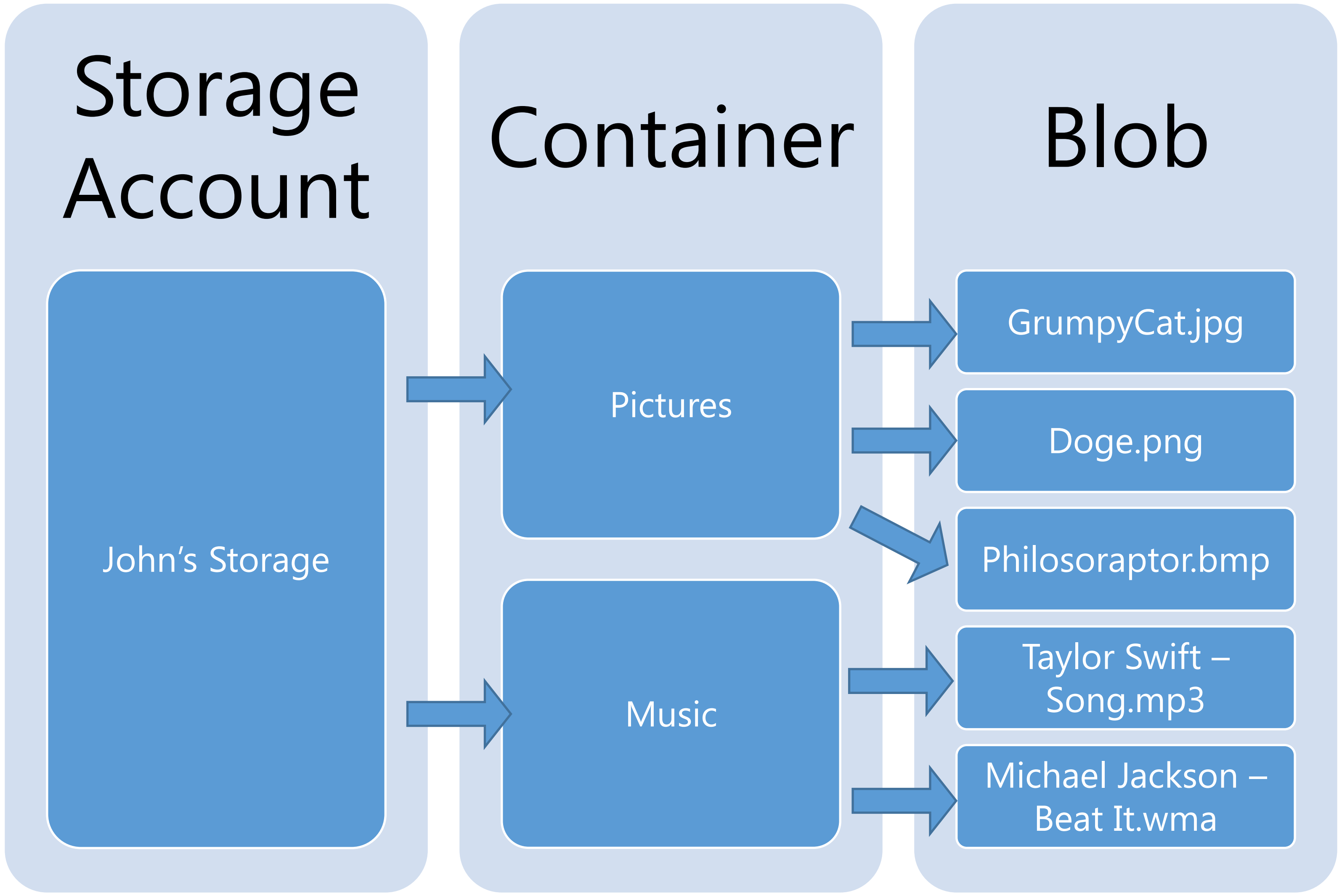




Azure Blob Storage

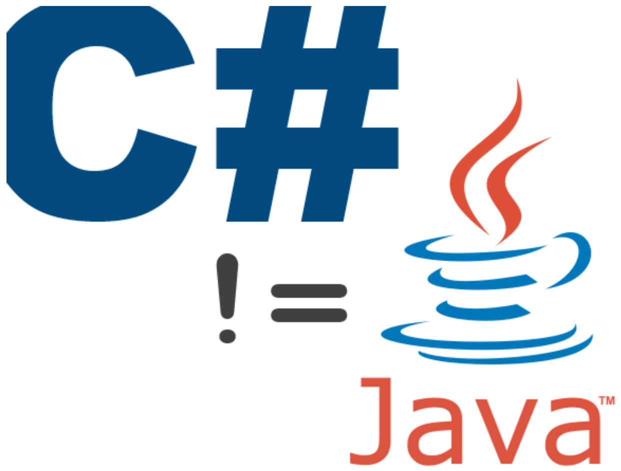


Azure Blob Storage



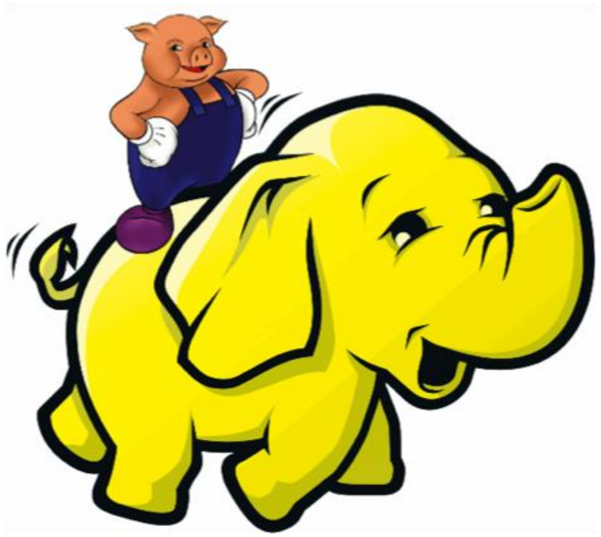
datasciencedojo
unleash the data scientist in you

When to Use Each



VS

VS



C#
Java
MapReduce

Hive

Pig

Questions?

datasciencedojo
unleash the data scientist in you