BIG DATA FUNDAMENTALS

Intel Costa Rica

Data Science Community of Practice

Alexandra Aguirre

Yu Hui Wen

Jose L Fernandez

Information Technology

AGENDA

- Session #1
  - Data Universe & Big Data
  - Hadoop
  - HDFS
  - Map Reduce
  - Sqoop

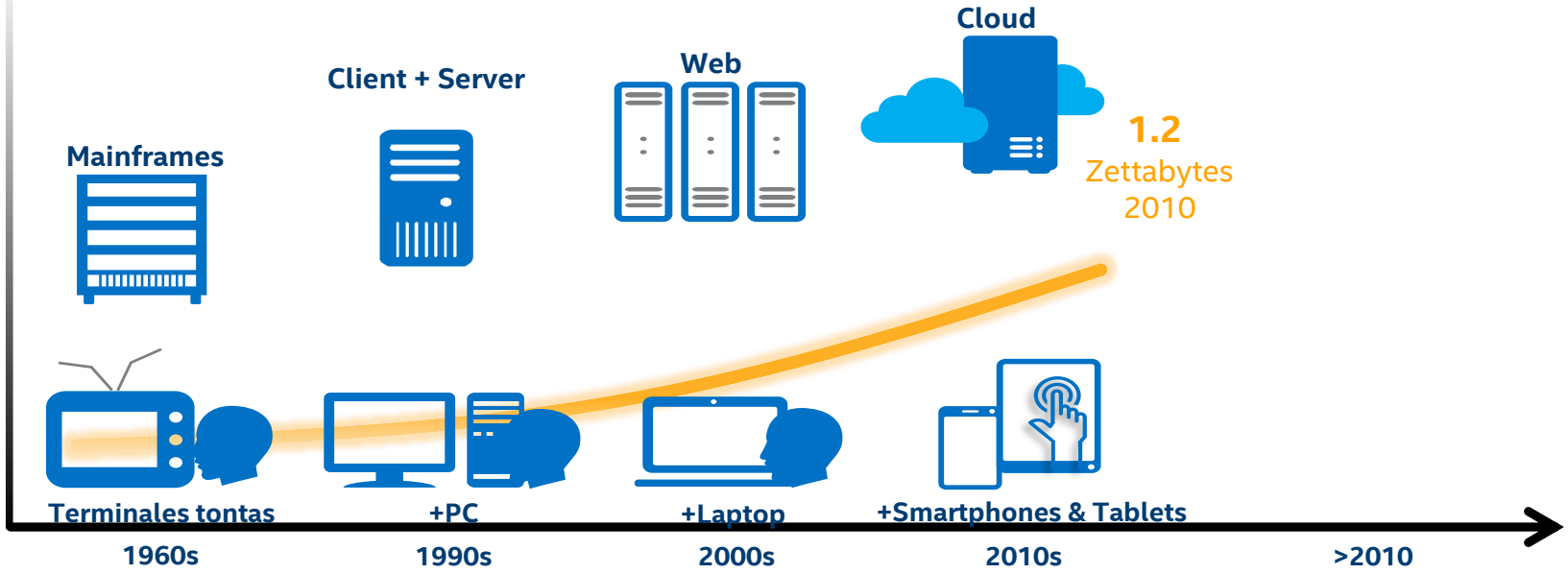
AGENDA

- Session #2
  - Hive & Impala
  - Apache Pig
  - Other tools for analysis

DATA UNIVERSE & BIG DATA

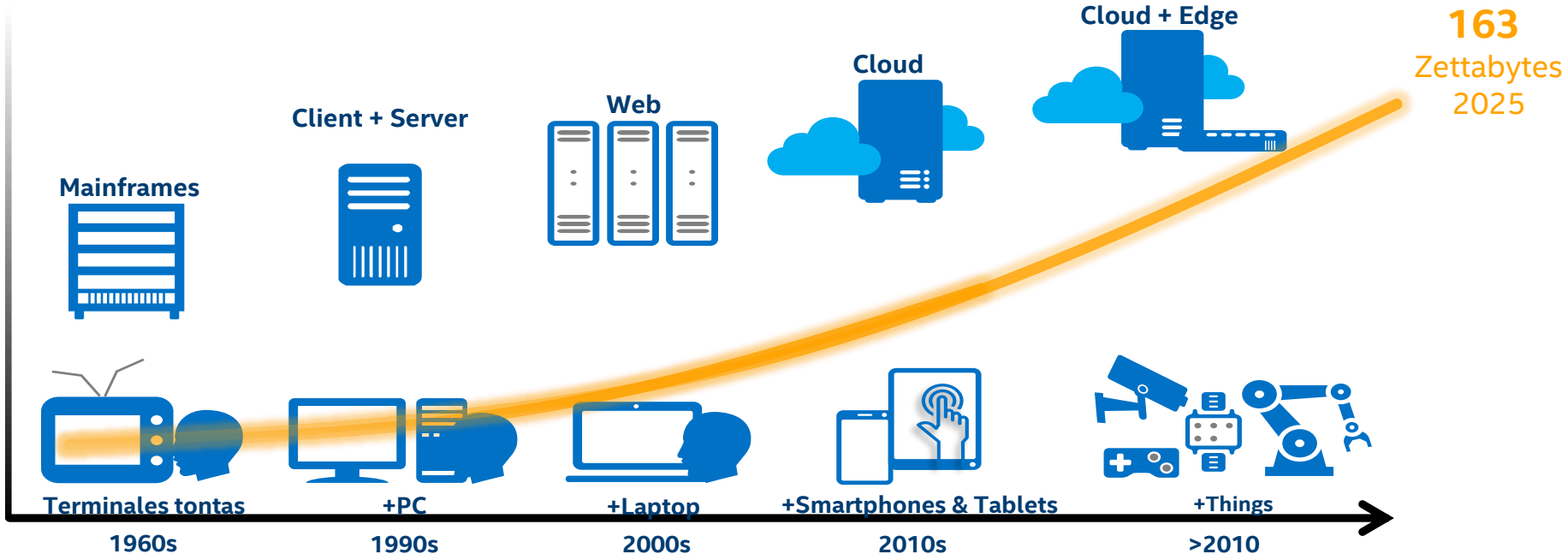
Information Technology

COMPUTATIONAL EVOLUTION



Fuente: Geetha Dabir VP/GM Intel

COMPUTATIONAL EVOLUTION



Fuente: Geetha Dabir VP/GM Intel

SMART PHONES AND SOCIAL NETWORKS

Web Traffic

2017

+50%

Smart phones

75%

Smart phones

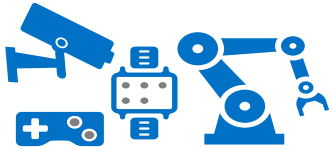
In one minute

4M
LIKES

350K
TWEETS

$300K
VENTAS

SMART PHONES AND SOCIAL NETWORKS



0.9 BILLONS

Connected devices
2009

20 BILLONS

Connected devices
2020

BIG DATA

Data whose scale, distribution, diversity, and/or timeliness require the use of new technical architectures and analytics to enable insights that unlock new sources of business value

VOLUMEN

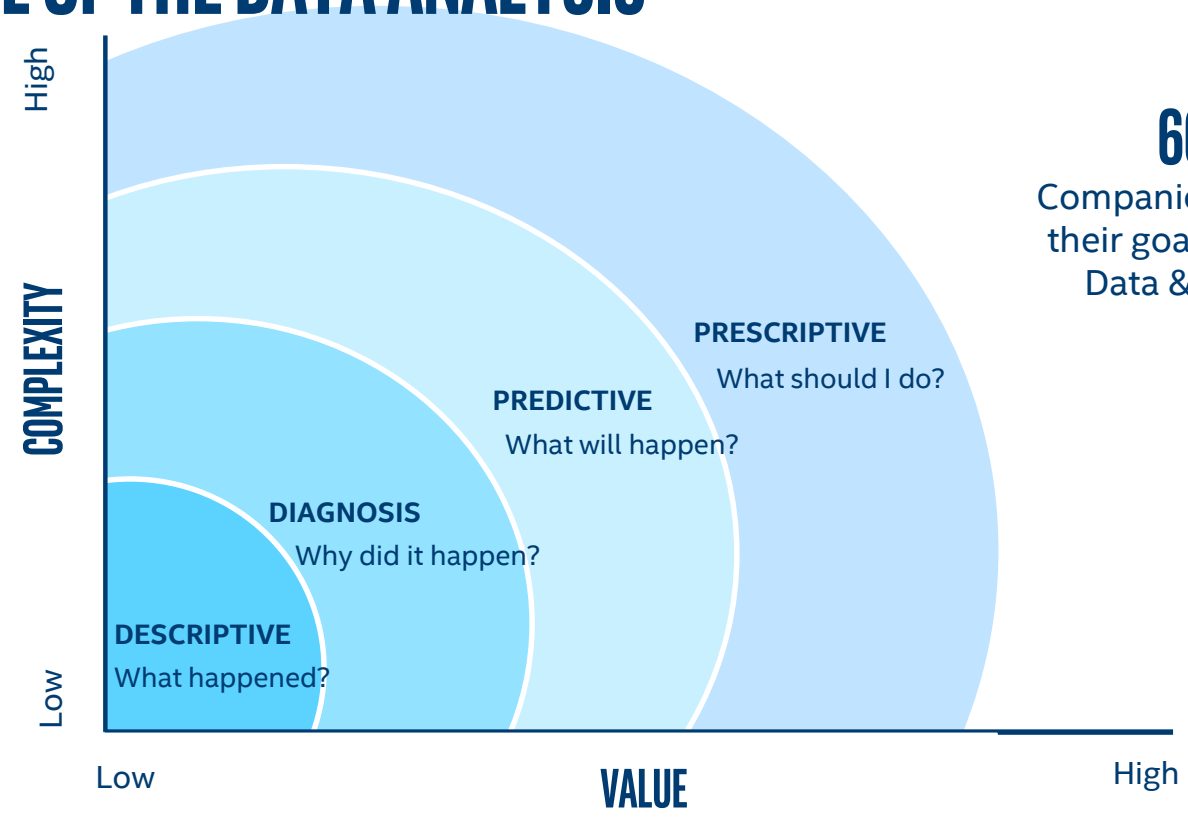
VARIETY

VELOCITY

VERACITY

VALUE

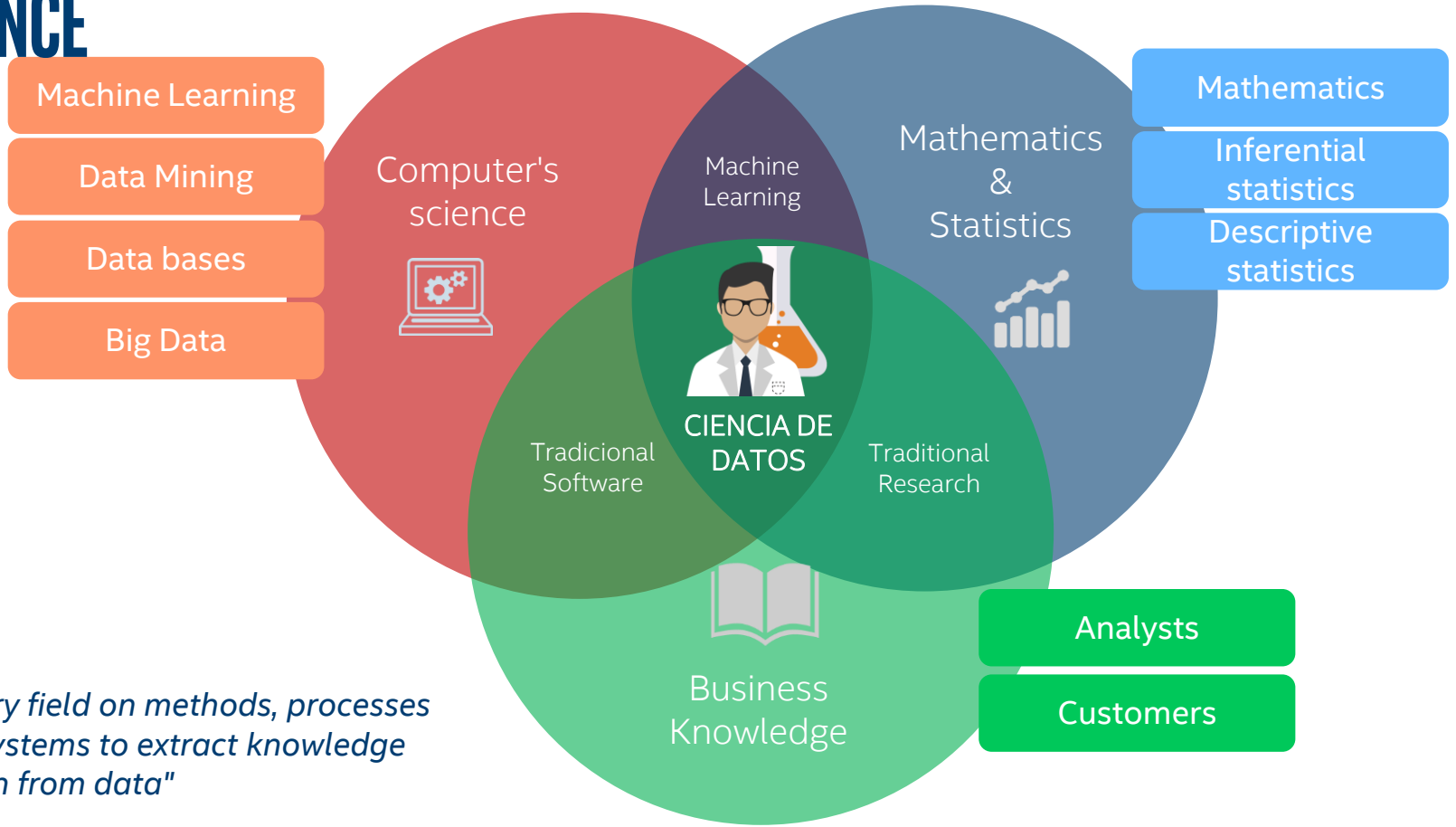
MATURITY MODEL OF THE DATA ANALYSIS



60%

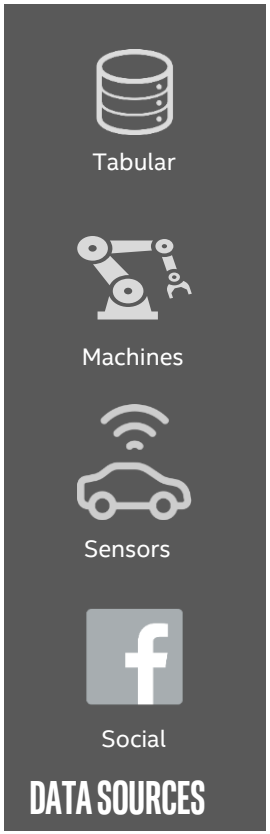
Companies exceeded their goals using Big Data & Analysis

DATA SCIENCE



"Interdisciplinary field on methods, processes and scientific systems to extract knowledge and information from data"

DATA LAKE

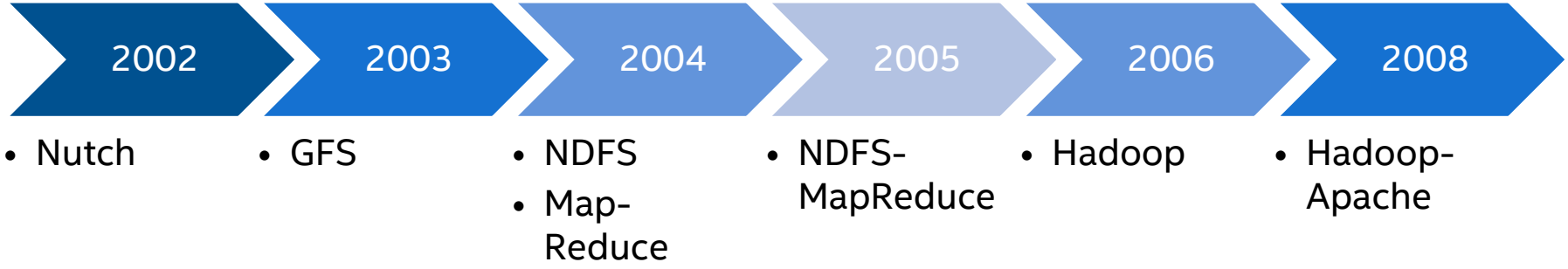


SECURITY, METADATA, GOVERNANCE, ADMINISTRATION

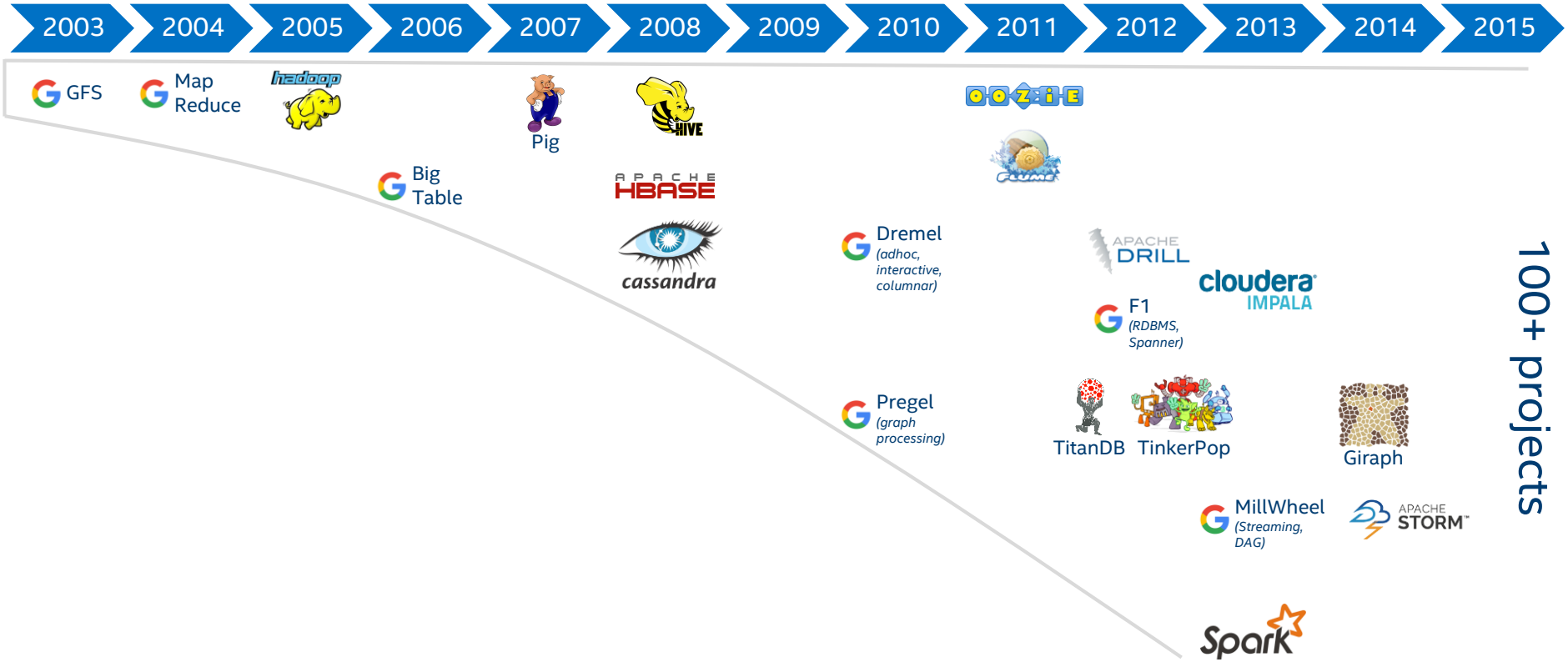
DISTRIBUTED STORAGE

HADOOP

Brief History



Evolution of the ecosystem



Hadoop

- Google has released several academic papers describing infrastructure:
  - Google File System, Map Reduce, Big Table, Dremel, Megastore, …
- Have been picked up by the Open Source community, and built their versions:
  - HDFS, Hadoop, Hbase, Hive, Pig, …

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

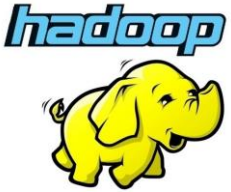
Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new

Doug Cutting

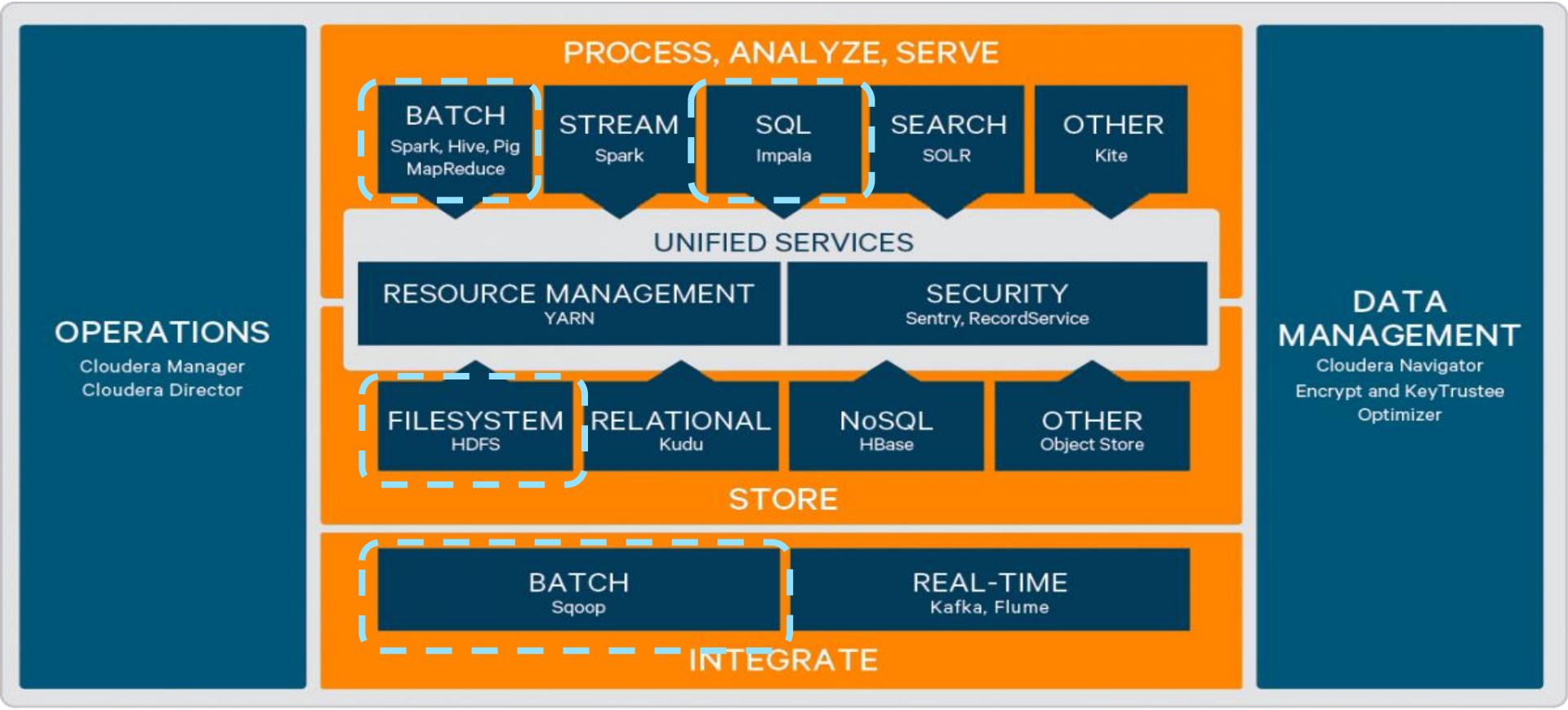


Hadoop in a nut shell

1. Build a cluster of machines
2. Distribute your data 

3. Process your data in parallel 

cloudera® DISTRIBUTION HADOOP (CDH)



HDFS

Hadoop Distributed File System

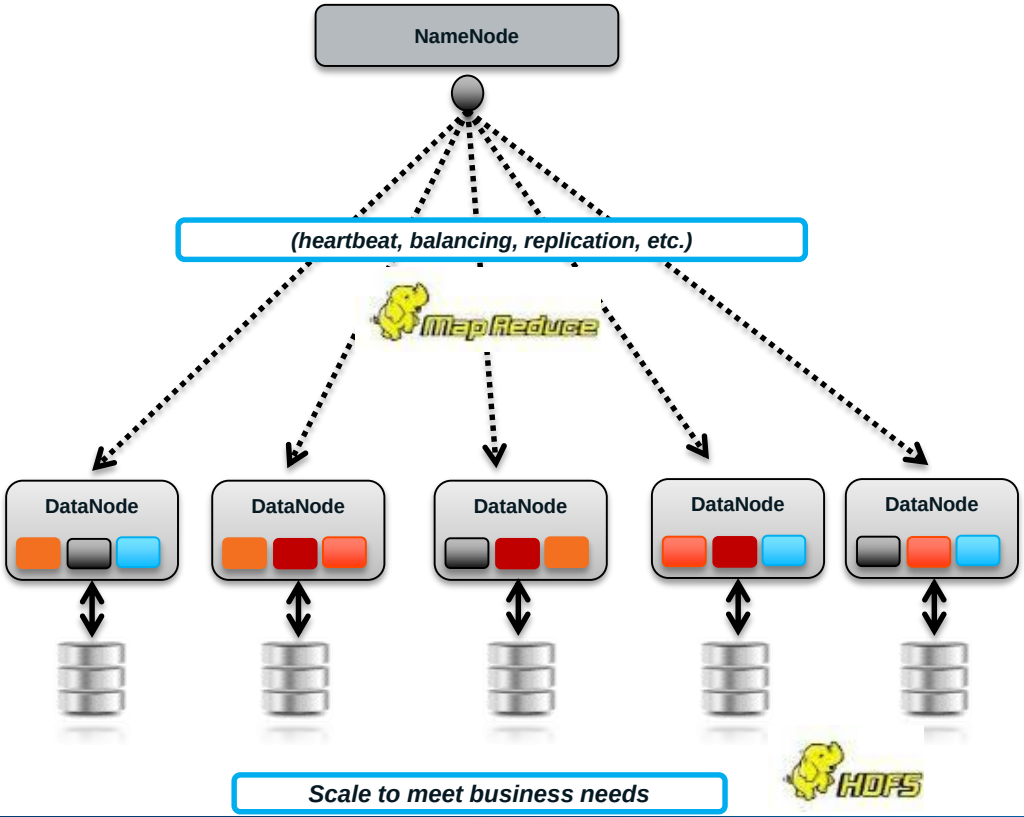
A Java-based file system that provides **scalable**, **fault-tolerant**, **distributed storage** system

HDFS stores large files across **multiple machines**

A **Hadoop cluster** has nominally a single **NameNode** plus a cluster of **DataNodes**

HDFS stores filesystem **metadata** and application **data** separately

Big Data Platform



42,000 NODES

YAHOO!

4,100 NODES

LinkedIn

1,400 NODES

facebook

Concepts

Splits files into large blocks and distributes them across nodes in a cluster

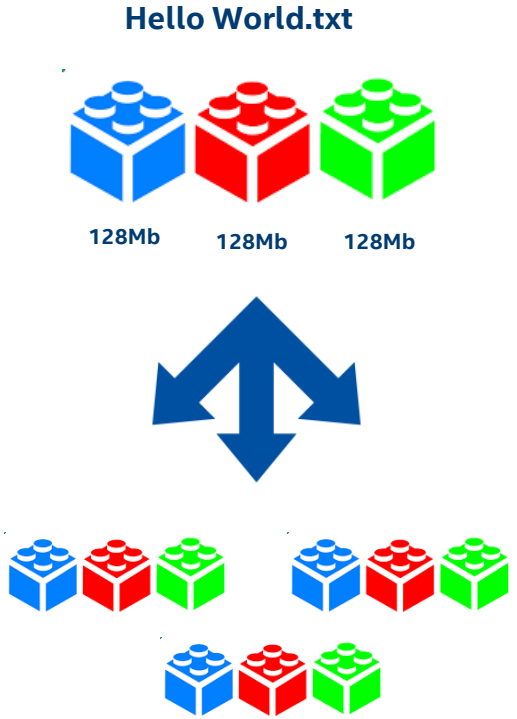
- Typically 128 megabytes

Each block of the file is independently replicated at multiple **DataNodes**

- Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high

Default replication value, 3, data is stored on three nodes

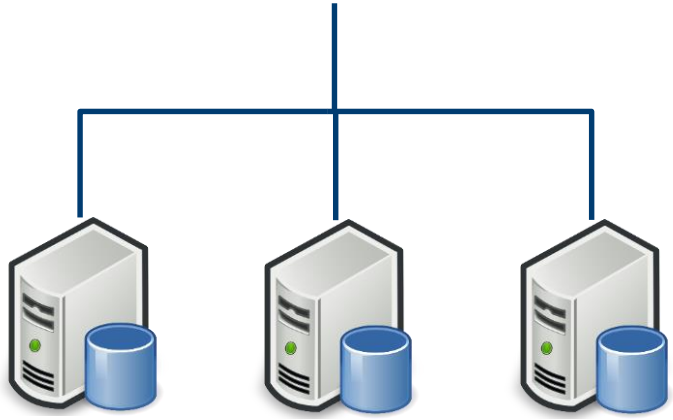
- Two on the same rack, and one on a different rack



Concepts

This approach takes advantage of **data locality**, nodes manipulating the data they have access to

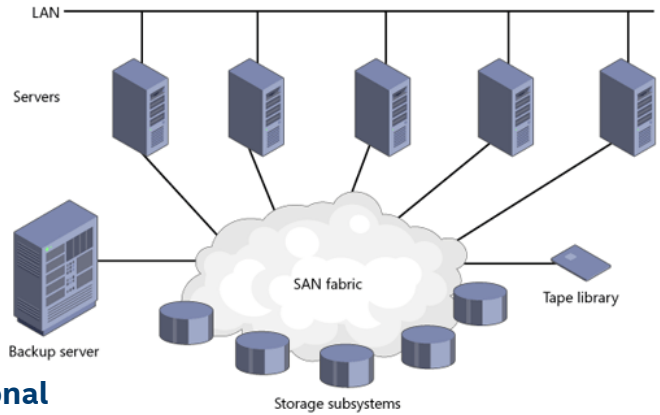
- Allow the dataset to be processed faster and more efficiently



HDFS

"Moving Computation is Cheaper than Moving Data"

- A computation requested by an application is much more efficient if it is executed near the data it operates on.



Traditional

Concepts

HDFS Client

- HDFS client interacts with **Namenode** and **Datanode** on behalf of user to fulfil user request.
- Establishes communication with HDFS through **File System API** and normal I/O operations

Namenode

- Namenode is the **masternode** of HDFS cluster.
- It stores **metadata** information and edit log in it.
- Metadata information contains addresses of **block locations** of Datanodes
- This information is used for file **read and write operation** to access the blocks in a HDFS cluster.
- The NameNode maintains the **namespace tree** and the mapping of **blocks** to DataNodes

Concepts

Datanode

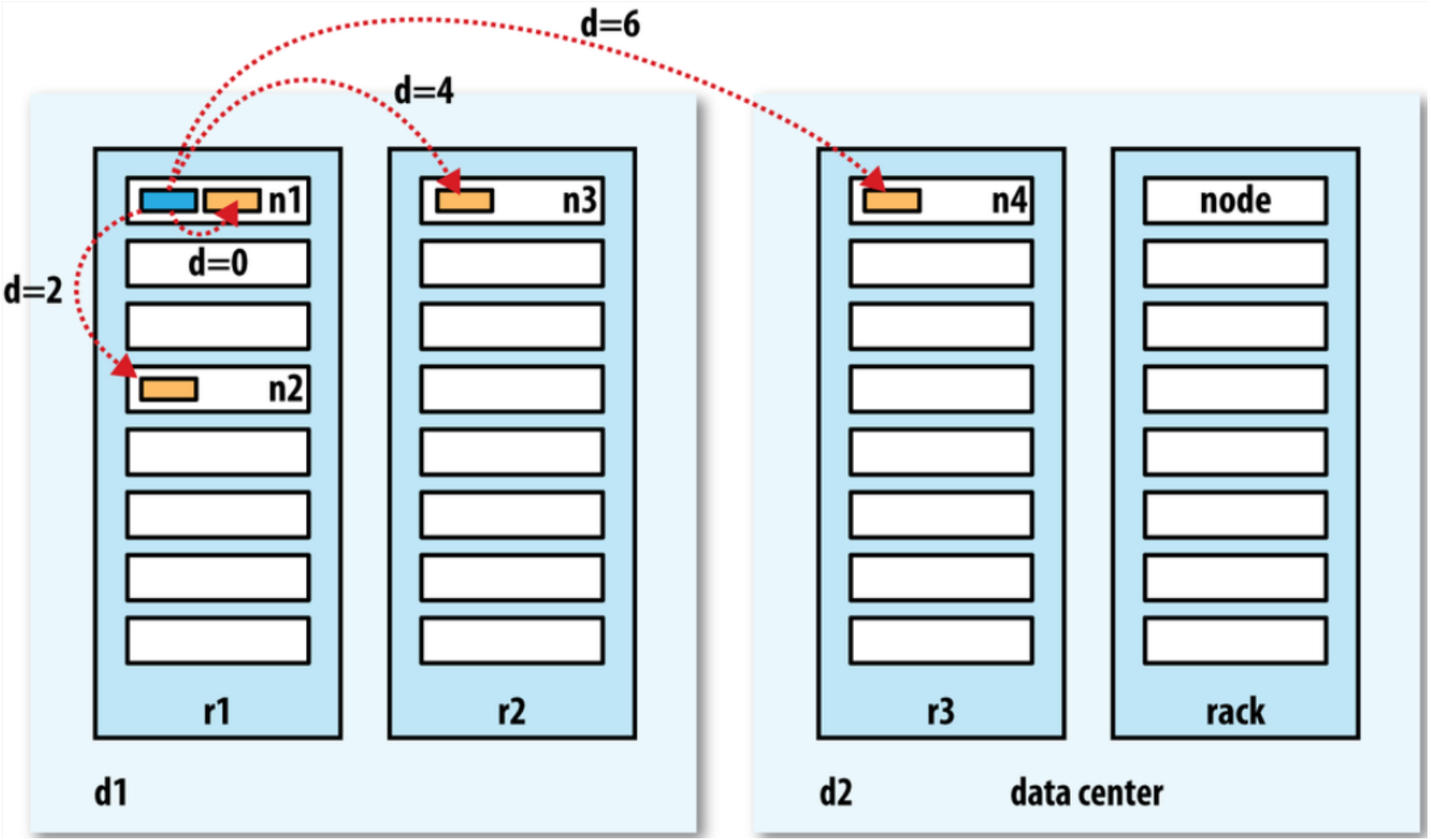
- Datanodes holds the actual **data**
- Datanode only **stores block**, a block is what is used to store and process the data.
- Datanode gives periodic **heartbeat signals** to Masternode to indicate that it is alive and can be used to store and retrieve data

Network Topology

In the context of high-volume data processing, the limiting factor is the rate at which we can transfer data between nodes—bandwidth is a scarce commodity

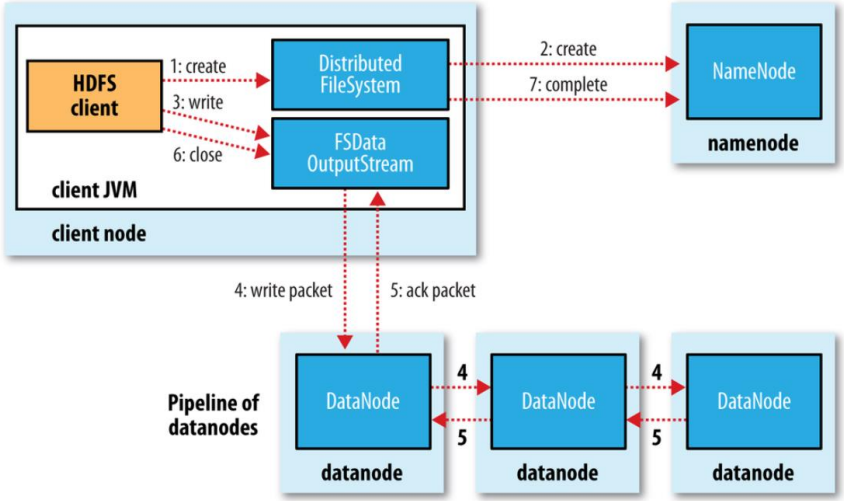
The idea is that the bandwidth available for each of the following scenarios becomes progressively less

- Processes on the same node
- Different nodes on the same rack
- Nodes on different racks in the same data center
- Nodes in different data centers


d=6
d=4
d=2
n1
d=0
n2
n3
n4
node
r1
r2
r3
rack
d1
d2
data center

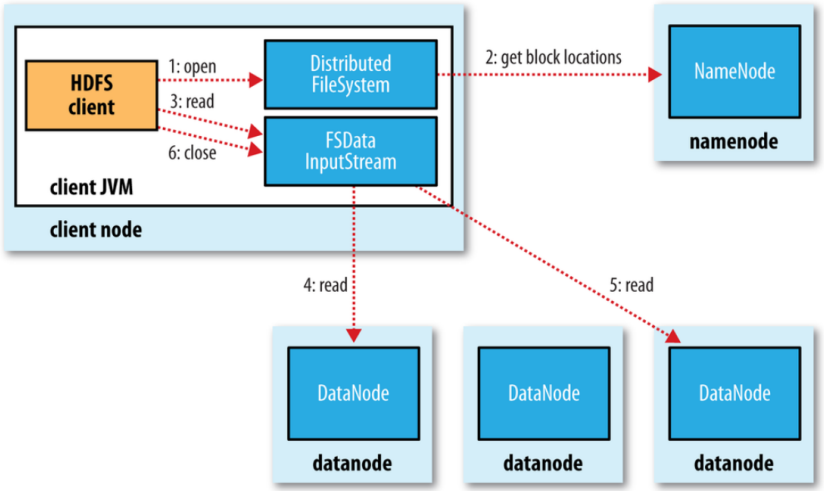
Anatomy of a File Write

1. User applications access the filesystem using the **HDFS client**
2. HDFS client asks the **NameNode** to choose **DataNodes** to host replicas of the first block of the file
3. The client organizes a **pipeline from node-to-node** and sends the data.
4. When the **first block is filled**, the client requests new DataNodes to be chosen to host replicas of the next block



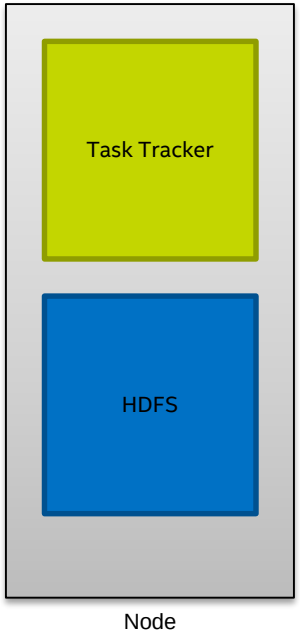
Anatomy of a File Read

1. User applications access the filesystem using the **HDFS client**
2. HDFS Client get the **block locations** of file to be read from **Namenode** metadata.
   - The list is sorted by the **network topology** distance from the client
3. For each block, **Namenode** returns the sorted **address of Datanode** that holds the copy of that block.
4. The client contacts a **DataNode directly** and requests the transfer of the desired block

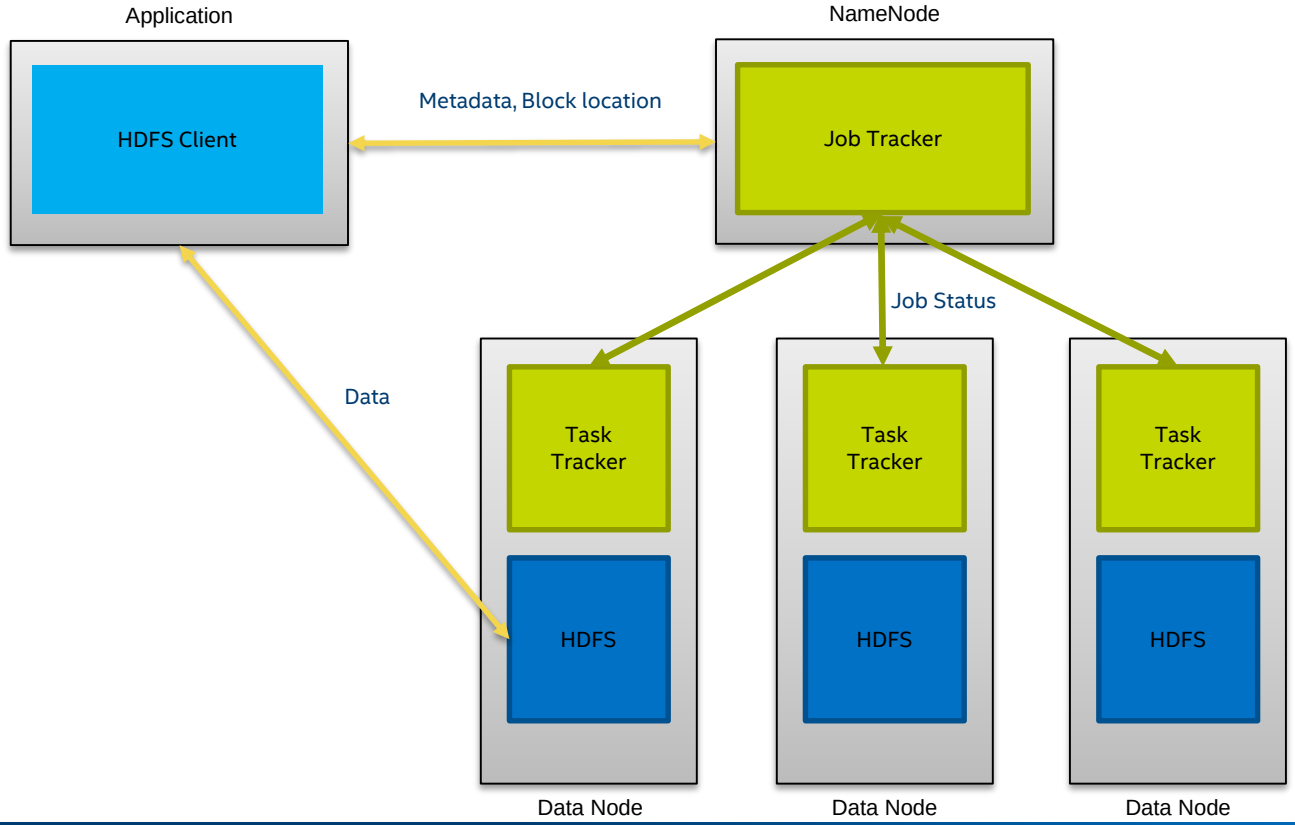


HDFS ARCHITECTURE

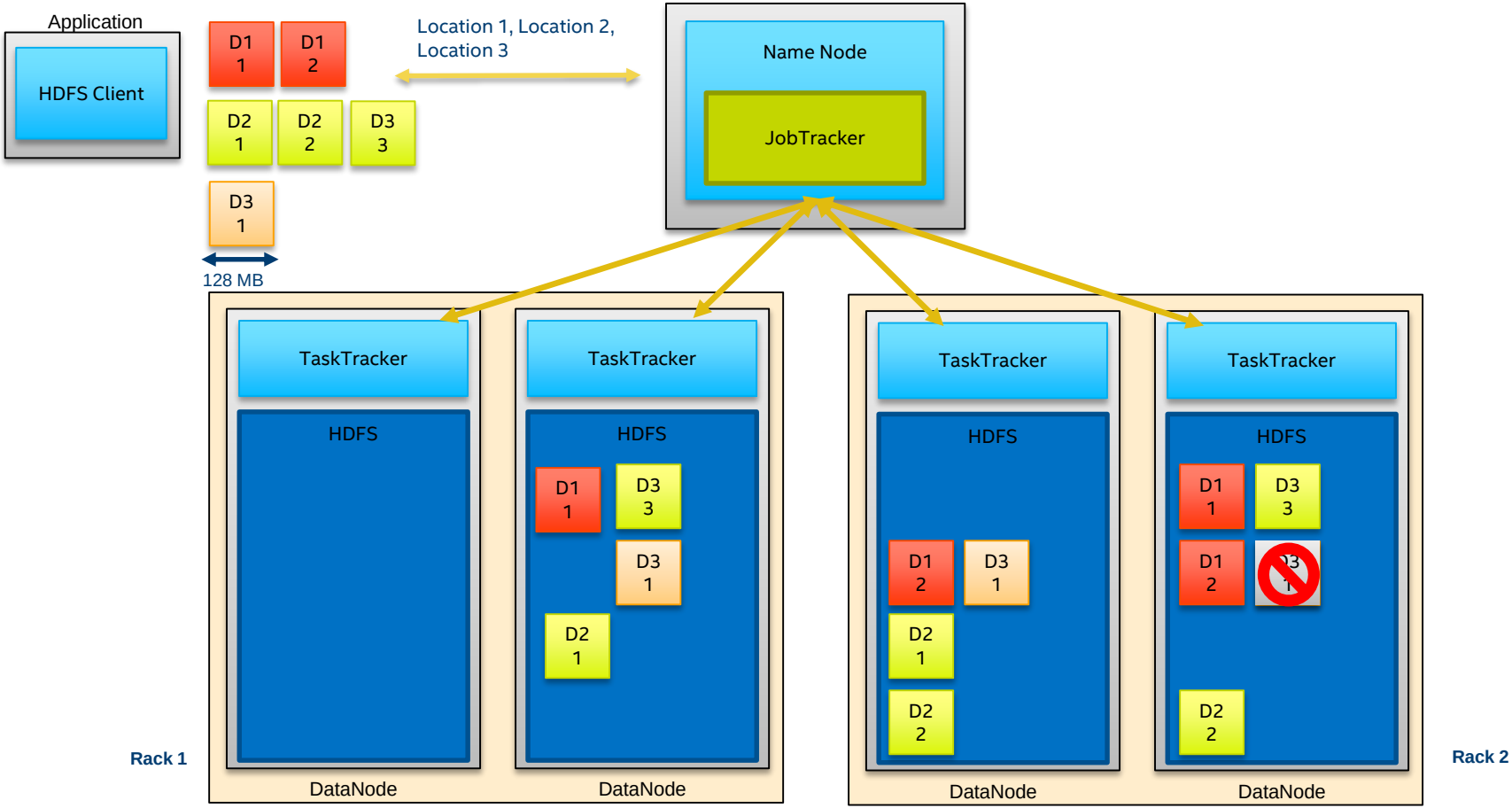
Hadoop Architecture



Hadoop Architecture



HDFS


Application
HDFS Client
D1 1
D1 2
D2 1
D2 2
D3 3
D3 1
128 MB
Location 1, Location 2, Location 3
Name Node
JobTracker
TaskTracker
HDFS
DataNode
Rack 1
Rack 2

HDFS COMMANDS & DEMO

HDFS commands

Print the Hadoop version	`hadoop version`
Help	`hadoop fs -help`
Report the amount of space used and available on currently mounted file system The -h option will format file sizes in a "human-readable" fashion	*`hadoop fs -df [-h] URI [URI ...]`* `hadoop fs -df hdfs:/` `hadoop fs -df -h hdfs:/`
List the contents of the root directory in HDFS	*`hadoop fs -ls [-d] [-h] [-R] [-t] [-S] [-r] [-u] <args>`* `hadoop fs -ls /`
List files in HDFS	`hadoop fs -ls`
Recursively list directory contents	`hadoop fs -ls -R`

http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html

HDFS commands

Create a new directory in HDFS	*hadoop fs -mkdir [-p] <paths>* `hadoop fs -mkdir temp`
Create a empty file	*hadoop fs -touchz URI* `hadoop fs -touchz temp/empty`
Copy files from source to destination	*hadoop fs -cp [-f] [-p | -p[topax]] URI [URI ...] <dest>* `hadoop fs -cp temp/empty temp/empty2`
Delete the file	*hadoop fs -rm [-f] [-r |-R] [-skipTrash] URI [URI ...]* `hadoop fs -rm temp/empty`
Move the directory to trash	*hadoop fs -rmdir [--ignore-fail-on-non-empty] URI [URI ...]* `hadoop fs -rmdir temp`

http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html

HDFS commands

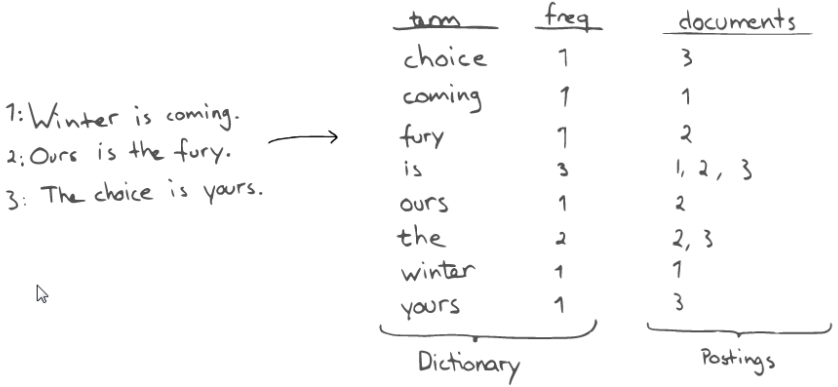
Copy single src, or multiple srcs from local file system to the destination file system.	*hadoop fs -put [-f] [-p] [-l] [-d] [- | <localsrc1> ..]. <dst>* `hadoop fs -put purchasesLocal.txt purchasesLocal.txt`
Copy files to the local file system.	*hadoop fs -get [-ignorecrc] [-crc] [-p] [-f] <src> <localdst>* `hadoop fs -get purchasesLocal.txt purchases.txt`
Takes a source directory and a destination file as input and concatenates files in src into the destination local file.	*hadoop fs -getmerge <src> <localdst>* `hadoop fs -getmerge hdfs_lab/output1 AliceResult.txt`

http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/FileSystemShell.html

MAP-REDUCE

Motivation & History

- Google wanted to index the whole internet
  - That is a lot of data…
  - To index the web you need to build a table of all the words, in all the websites, with their respective counts



https://www.found.no/foundation/elasticsearch-from-the-bottom-up/

Even simple things, like counting words become complex at scale…

- Map Reduce was developed for this purpose
  - But they soon realized that this paradigm could be used for a lot of other tasks

What is Map Reduce?

- Programming paradigm
  - Divide and Conquer:
    - Work is performed by nodes of a cluster
    - *Moving processing is better than moving data*
- Provides a simple API for the programmer:

	Input	Output
Map	(k1, v1)	list(k2, v2)
Reduce	(k2, list(v2))	list(k3, v3)

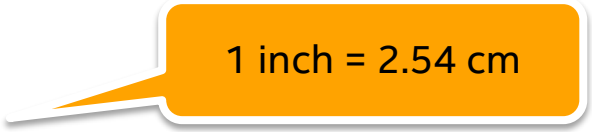
- Framework for parallel computing
  - Abstracts the user from issues of:
    - Parallelization
    - Remote execution
    - Data Distribution
    - Load Balancing
    - Fault Tolerance

Activity: What is the average height of those present?

Instructions:

Go to:

Enter your height in inches



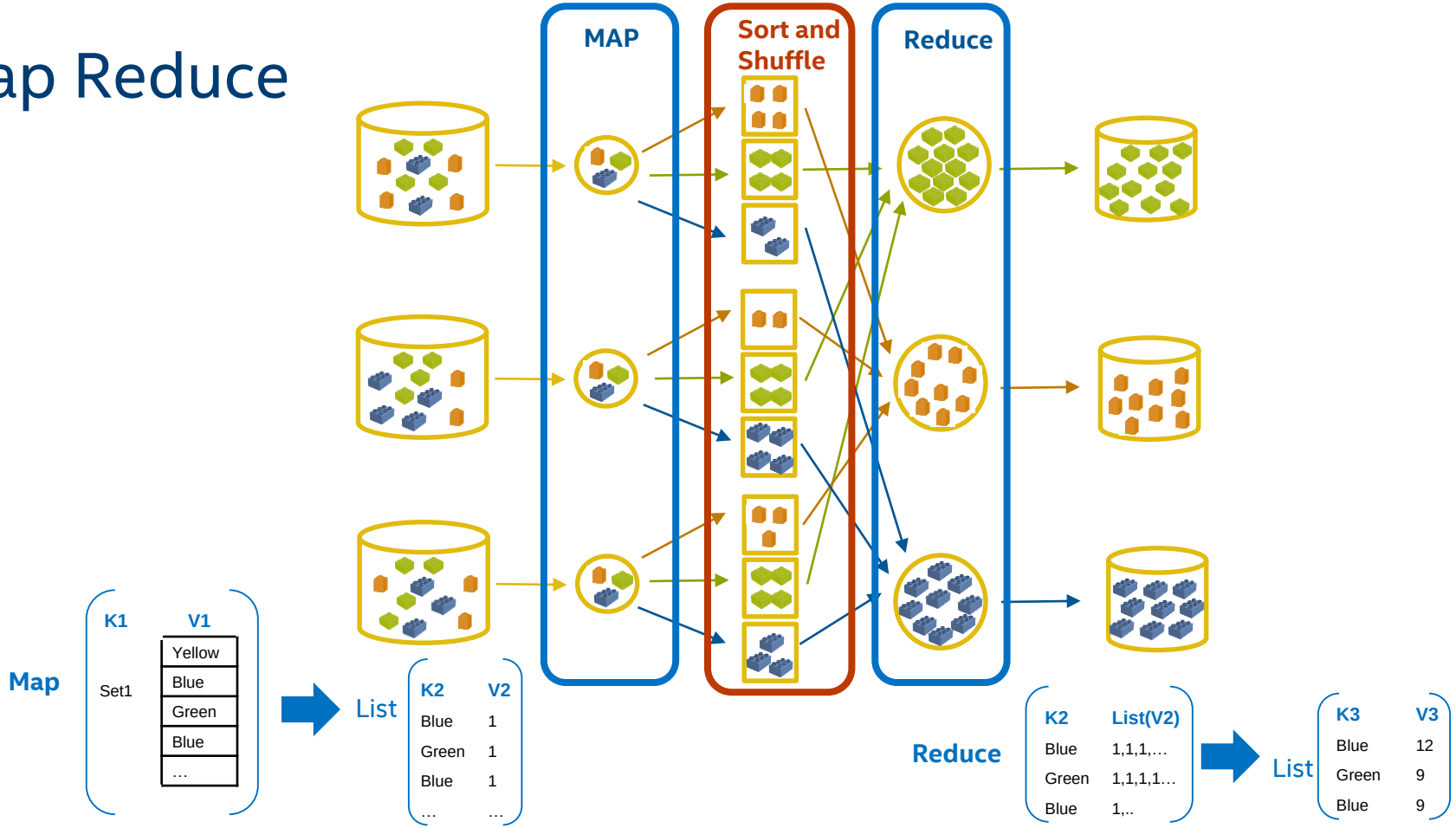
You just made a Map Reduce!

Each one acted as a **Mapper** node, in a distributed cluster

Each one had a small amount of data (their height), and each one was part of the processing (convert to inches)

They sent their results to a **Reducer**, who calculated the statistics

Map Reduce


MAP
Sort and Shuffle
Reduce
Map
K1
V1
Set1
Yellow
Blue
Green
Blue
…
List
K2
V2
Blue
1
Green
1
Blue
1
…
…
Reduce
K2
List(V2)
Blue
1,1,1,…
Green
1,1,1,1…
Blue
1,..
List
K3
V3
Blue
12
Green
9
Blue
9

NOW LETS SEE IT IN CODE

Canonical example: Word Counting

```
function map(String name, String document):
  // name: document name
  // document: document contents
  for each word w in document:
    emit (w, 1)
```

```
function reduce(String word, Iterator partialCounts):
  // word: a word
  // partialCounts: a list of aggregated partial counts
  sum = 0
  for each pc in partialCounts:
    sum += ParseInt(pc)
  emit (word, sum)
```

Mapper

```
package com.intel.aa.wc;

import java.io.IOException;

public class WordMapper extends MapReduceBase implements
    Mapper<LongWritable, Text, Text, IntWritable> {

  public void map(LongWritable key, Text value,
      OutputCollector<Text, IntWritable> output, Reporter reporter)
      throws IOException {
    String s = value.toString();
    for (String word : s.split("\\W+")) {
      if (word.length() > 0) {
        output.collect(new Text(word), new IntWritable(1));
      }
    }
  }
}
```

Reducer

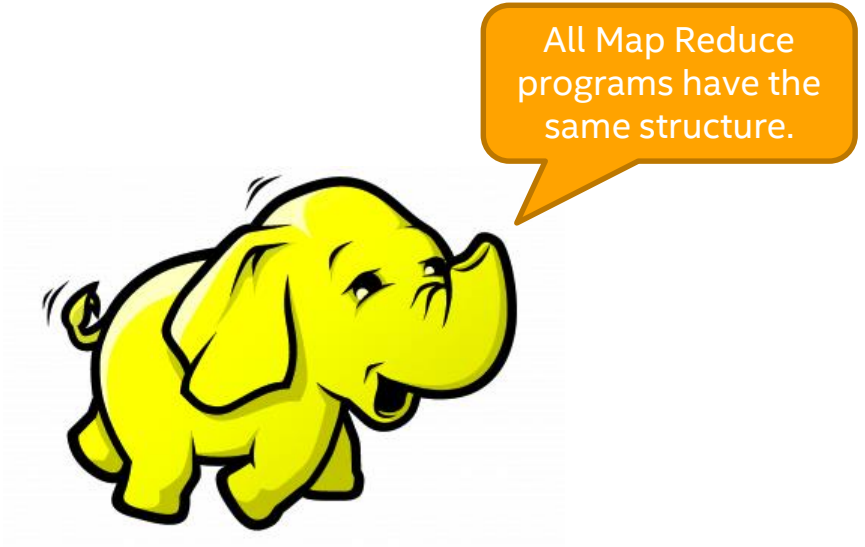
```
package com.intel.aa.wc;

import java.io.IOException;

public class SumReducer extends MapReduceBase implements
    Reducer<Text, IntWritable, Text, IntWritable> {

  public void reduce(Text key, Iterator<IntWritable> values,
      OutputCollector<Text, IntWritable> output, Reporter reporter)
      throws IOException {

    int wordCount = 0;
    while (values.hasNext()) {
      IntWritable value = values.next();
      wordCount += value.get();
    }
    output.collect(key, new IntWritable(wordCount));
  }
}
```

Driver

```
package com.intel.aa.wc;

import org.apache.hadoop.fs.Path;

public class WordCountMapOnly extends Configured implements Tool {

  public int run(String[] args) throws Exception {

    if (args.length != 2) {
      System.out.printf(
          "Usage: %s [generic options] <input dir> <output dir>\n", getClass()
              .getSimpleName());
      ToolRunner.printGenericCommandUsage(System.out);
      return -1;
    }

    JobConf conf = new JobConf(getConf(), WordCountMapOnly.class);
    conf.setJobName(this.getClass().getName());

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    conf.setMapperClass(WordMapper.class);
    //conf.setReducerClass(SumReducer.class);

    conf.setMapOutputKeyClass(Text.class);
    conf.setMapOutputValueClass(IntWritable.class);
```

Three classes

- Mapper
  - Implements the mapper algorithm
  - Must inherit from Mapper
- Reducer
  - Implements the reducer algorithm
  - Must inherit from Reducer
- Driver
  - Does the set up of the program
  - Specifies classes and types to use
  - Specifies input



MAP REDUCE DEMO

TMDB 5000 Movie Dataset

What can we say about the success of a movie before it is released? Are there certain companies (Pixar?) that have found a consistent formula? Given that major films costing over $100 million to produce can still flop, this question is more important than ever to the industry. Can we predict which films will be highly rated, whether or not they are a commercial success?

Datasets

- tmdb_5000_credits
- tmdb_5000_movies
- Cornell Movie--Dialogs Corpus



Source: https://www.kaggle.com/tmdb/tmdb-movie-metadata/data

Word Count Analysis

Running a Map Reduce Job

```
hadoop jar [jarFile] [ClassWithMain] [args…]
```

Mapper

```
hadoop jar wc-0.0.1-SNAPSHOT.jar com.intel.aa.wc.WordCountMapOnly mr-exercise/movie_lines.txt mr-exercise/wc-m-mv-lines
```

Mapper & Combiner

```
hadoop jar wc-0.0.1-SNAPSHOT.jar com.intel.aa.wc.WordCountMapCombiner mr-exercise/movie_lines.txt mr-exercise/wc-mc-mv-lines
```

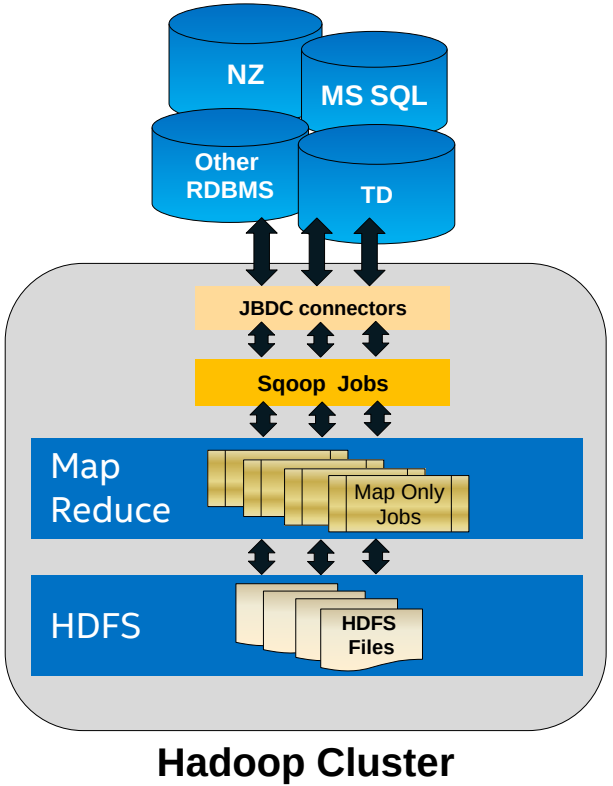
Mapper, Combiner & Reducer

```
hadoop jar wc-0.0.1-SNAPSHOT.jar com.intel.aa.wc.WordCountMapCombinerReducer mr-exercise/movie_lines.txt mr-exercise/wc-mcr-mv-lines
```

SQOOP

Sqoop Introduction

- Sqoop stands for "SQL-to-Hadoop."
- It was originally developed by Cloudera and it is now an open source project at Apache.
- It is designed for efficiently transferring bulk data between Hadoop and structured datastores such as relational databases.
- It supports mySql, PostGreSQL, HSQLDB, Oracle, Netezza (NZ), MS SQL, Teradata (TD), etc.
- It allows import data into Hive and Hbase.
- Sqoop uses MapReduce (map only job), which provides parallel operation and fault tolerance.



SQOOP IMPORT

Import to HDFS

Import to Hive

Sqoop Import

- Extract data out of RDBMS and load into HDFS storage
- Import into HDFS or Hive table
- Cannot import into local Linux directory
- Map tasks run in parallel and extract data independently
- By default, four tasks are used
- Number of map tasks (parallel processes) to use to import can be changed by using the -m or --num-mappers
- Sqoop uses a *splitting column* to split the workload
- By default, Sqoop will identify the primary key column (if present) in a table and use it as the splitting column

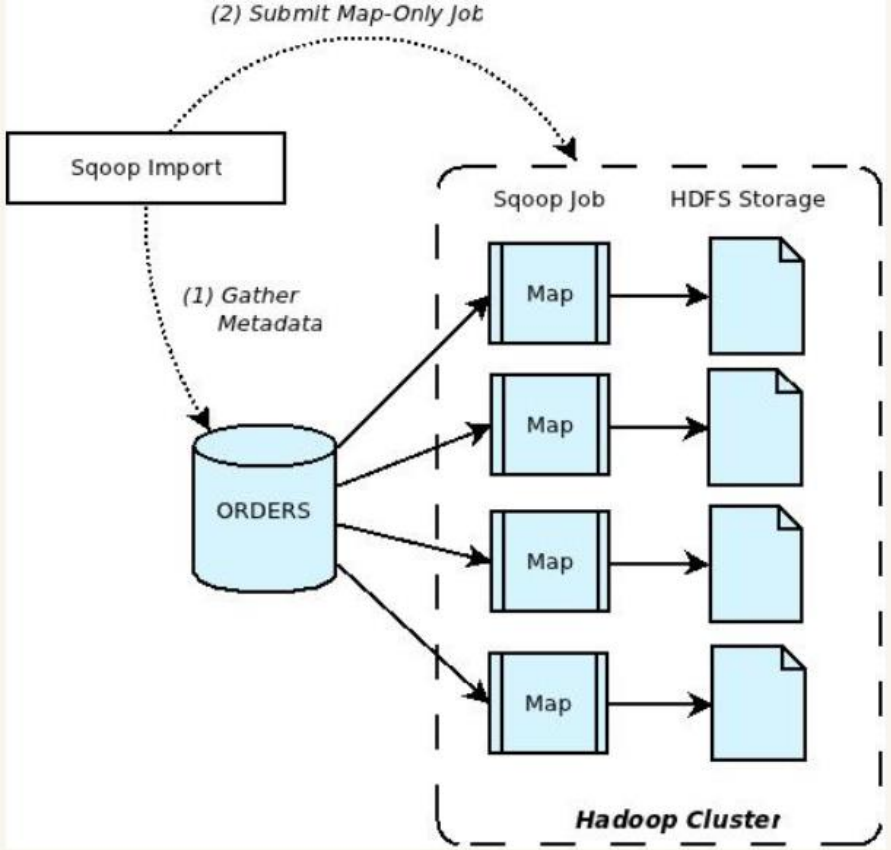


Figure 1: Sqoop Import Overview

Sqoop Import to HDFS - **Table**


```
sqoop import
--driver com.microsoft.sqlserver.jdbc.SQLServerDriver
--connect 'jdbc:sqlserver://<server>:<port>;DATABASE=<DB name>'
--table movies
--split-by movie_id
--target-dir '/user/<user>/movies_import'
--fields-terminated-by '\t'
--username <User name>
--password <Secret>
```

JDBC driver class to use
Note: You can use Sqoop with any other JDBC-compliant database.

Field used to split the table for parallel import

JDBC connect string

Lab - Sqoop Import into HDFS - **Table**

1. Verify the target folder does not exist before executing command.

```
hadoop fs -ls /user/<user>/movies_import
```

2. What message do you get?

```
sqoop import --driver com.microsoft.sqlserver.jdbc.SQLServerDriver --connect
'jdbc:sqlserver://<server>:<port>;DATABASE=<db name>' --table movies --target-dir
'/user/<user>/movies_import' --split-by movie_id  --fields-terminated-by '\t' --username <user name> --
password <secret>
```

3. Validate data in HDFS

Sqoop Import to HDFS – **Columns and Filter**


```
sqoop import
--driver com.microsoft.sqlserver.jdbc.SQLServerDriver
--connect 'jdbc:sqlserver://<server>:<port>;DATABASE=<db name>'
--table movies
--columns “movie_id,Name”
--where “category = 'Comedy'”
--split-by movie_id
--target-dir '/user/<user>/movies_import_filter'
--fields-terminated-by '\t'
--username <User name>
--password <Secret>
```

Subset of columns to extract and which rows are imported

Lab - Sqoop Import into HDFS – **Columns and Filters**

1. Verify the target folder does not exist before executing command.

```
hadoop fs -ls /user/<user>/movies_import_filter
```

What message do you get?

```
sqoop import --driver com.microsoft.sqlserver.jdbc.SQLServerDriver --connect 'jdbc:sqlserver://
<server>:<port>;DATABASE=<db name>' --table movies --columns "movie_id,Name" --where "category =
'Comedy'" --split-by movie_id --target-dir '/user/<user>/movies_import_filter' --fields-terminated-by '\t' --
username <user name> --password <secret>
```

3. Validate data in HDFS

Sqoop Import to HDFS – **Free-form Query**


```
sqoop import
--driver com.microsoft.sqlserver.jdbc.SQLServerDriver
--connect 'jdbc:sqlserver:// <server>:<port>;DATABASE=<db name>'
--query 'SELECT [movie_id] ,[name] FROM [sqoopdb].[dbo].[movies] WHERE movie_id
BETWEEN category='comedy' AND $CONDITIONS'
--split-by movie_id
--target-dir '/user/<user>/movies_import_query'
--fields-terminated-by '\t'
--username <User name>
--password <Secret>
```

Specify the SQL to execute
Note: Must include $CONDITIONS token to enable parallelism

Lab - Sqoop Import into HDFS – **Free-form Query**

1. Verify the target folder does not exist before executing command.

```
hadoop fs -ls /user/<user>/movies_import_query
```

What message do you get?

```
sqoop import --driver com.microsoft.sqlserver.jdbc.SQLServerDriver --connect 'jdbc:sqlserver://
<server>:<port>;DATABASE=<db name>' --query 'SELECT [movie_id] ,[name] FROM [sqoop].[dbo].[movies]
WHERE movie_id BETWEEN 0 AND 100 AND $CONDITIONS' --split-by movie_id  --target-dir
'/user/<user>/movies_import_query' --fields-terminated-by '\t' --username <user name> --password <secret>
```

3. Validate data in HDFS

Sqoop Import into Hive


```
sqoop import
--driver com.microsoft.sqlserver.jdbc.SQLServerDriver
--connect 'jdbc:sqlserver:// <server>:<port>;DATABASE=<db name> '
--table movies
--split-by movie_id
--target-dir movies_sqoop_temp
--hive-import
--hive-overwrite
--hive-table movies_sqoop_<user>
--fields-terminated-by '\t'
--username <User name>
--password <Secret>
```

Enable import data to Hive and Overwrite table if already exist

You can use any of the other forms of Import

Lab - Sqoop Import into Hive

1. Create hive table

```
hive -e "CREATE EXTERNAL TABLE movies_sqoop_<user> (id INT, name STRING, category String) ROW FORMAT
DELIMITED FIELDS TERMINATED BY '\t' STORED AS TEXTFILE LOCATION '/user/<user>/movies_sqoop';"
```

2. Import data into hive table

```
sqoop import --driver com.microsoft.sqlserver.jdbc.SQLServerDriver --connect 'jdbc:sqlserver://
<server>:<port>;DATABASE=<db name>' --table movies  --split-by movie_id  --target-dir
movies_sqoop_temp_<user> --hive-import --hive-overwrite --hive-table movies_sqoop_<user>  --fields-
terminated-by '\t' --username <user name> --password <secret>
```

```
hive -e "SELECT * FROM movies_sqoop_<user> LIMIT 5;"
```

3. Verify that the data has been loaded successfully

SQOOP EXPORT

Export to MSSQL

Sqoop Export

- Load data into RDBMS from HDFS storage
- Export from HDFS folder only.
- Cannot export Hive table or local Linux directory
- Map tasks run in parallel and load data independently
- Number of map tasks is configurable using –m option.
- Target table must exist
- By default sqoop-export appends new rows to a table
- Constraints must be handled by developer
- Sqoop offers the option to update specific records

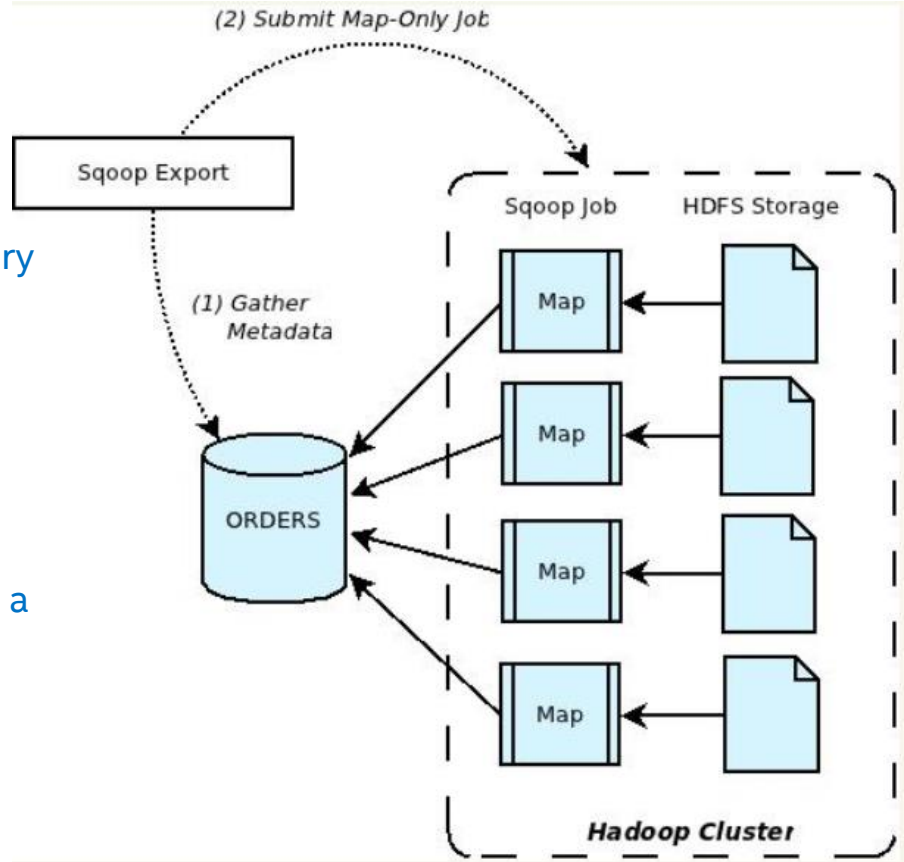


Figure 2: Sqoop Export Overview

Sqoop Export to MSSQL


```
sqoop export
--driver com.microsoft.sqlserver.jdbc.SQLServerDriver
--connect 'jdbc:sqlserver:// <server>:<port>;DATABASE=<db name>'
--table movies<user#>
--export-dir '/user/<user>/movies_import'
--username <User name>
--password <Secret>
--input-fields-terminated-by '\t'
```

Specify DB connection properties and destination table

Lab – Sqoop Export

1. Execute the command to export the data

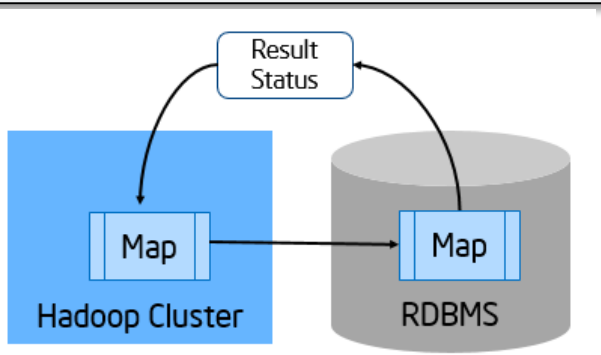
```
sqoop export --driver com.microsoft.sqlserver.jdbc.SQLServerDriver --connect 'jdbc:sqlserver://
<server>:<port>;DATABASE=<db name>' --table movies<user#>  --export-dir '/user/<user>/movies_import' --
username <user name> --password <secret> --input-fields-terminated-by '\t'
```

How to verify the data was exported successfully?

Sqoop Eval

Allows to execute SQL statements against a database on remote RDBMS.

```
sqoop eval
--driver com.microsoft.sqlserver.jdbc.SQLServerDriver
--connect 'jdbc:sqlserver:// <server>:<port>;DATABASE=<db name> '
--query "SELECT TOP 10 * FROM movies<user#>;"
--username <User name>
--password <Secret>
```



Lab - Sqoop eval

1. Query top 10 records in Movies table

```
sqoop eval --driver com.microsoft.sqlserver.jdbc.SQLServerDriver -connect 'jdbc:sqlserver://
<server>:<port>;DATABASE=<db name>' --query "SELECT TOP 10 * FROM movies<user#>;" --username <user
name> --password <secret>
```

2. Delete data in Movies table

```
sqoop eval --driver com.microsoft.sqlserver.jdbc.SQLServerDriver -connect 'jdbc:sqlserver://
<server>:<port>;DATABASE=<db name>' --query "delete from movies<user#>;" --username <user name> --
password <secret>
```

Resources

- Official Documentation Page:
  - http://sqoop.apache.org/docs/1.4.6/SqoopUserGuide.html

HIVE

Remember Map Reduce?

- Programming paradigm
  - Divide and Conquer:
    - Work is performed by nodes of a cluster
    - *Moving processing is better than moving data*
- Provides a simple API for the programmer:

	Input	Output
• Map:	(k1, v1)	list(k2, v2)
• Reduce:	(k2, list(v2))	list(k3, v3)

- Framework for parallel computing
  - Abstracts the user from issues of:
    - Parallelization,
    - Remote execution,
    - Data Distribution,
    - Load Balancing,
    - Fault Tolerance

It is very powerful, but…

```java
//All reducers must inherit from Reducer.  This class takes 4 types as parameters:
//Reducer<TypeOfKeyIn, TypeOfValueIn, TypeOfKeyOut, TypeOfValueOut>
public class ReducerSalesPerStore
    extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {

    @Override
    public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
        throws IOException, InterruptedException {
        // Each line that the reducer will get looks like:
        // key: storeLocation, such as San Jose
        // values: a set of sales in dollar amount
        try {
            // Iterate and accummulate the total sales for this store
            double totalSales = 0;
            for (DoubleWritable sale : values) {
                totalSales += sale.get();
            }

            // The output of the mapper is written to the context
            // Notice the conversion to the hadoop types
            context.write(key, new DoubleWritable(totalSales));

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
// All mappers must inherit from Mapper.  This class takes 4 types as parameters:
// Mapper<TypeOfKeyIn, TypeOfValueIn, TypeOfKeyOut, TypeOfValueOut>
// For this example, we don't get/care about a KeyIn, since it will read a line of text,
// so we can set it to Object
public class MapperSalesPerStore
    extends Mapper<Object, Text, Text, DoubleWritable> {

    @Override
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
        // Each line that the mapper will get looks like (tab separated):
        // 2012-01-01 09:00 San Jose Men's Clothing 214.05 Amex
        try {
            // Parse each line and get the store location and the sale value
            String line = value.toString().trim();
            String[] parts = line.split("\t");
            String storeLocation = parts[2];
            Double sale = Double.parseDouble(parts[4]);

            // The output of the mapper is written to the context
            // Notice the conversion to the hadoop types
            context.write(new Text(storeLocation), new DoubleWritable(sale));

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
public class DriverSalesPerStore {
    public static void main(String[] args) throws Exception {
        // Validate that two arguments were passed from the command line.
        String input, output;
        if (args.length == 2) {
            input = args[0];
            output = args[1];
        } else {
            System.err.println("Incorrect number of arguments.  Expected: input output");
            return;
        }

        // Create a instance of Job, based on a configuration, and give it a name (used in the reports and logs)
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf);
        job.setJobName("MR - Sales Per Store");

        // Specify the jar file.  This will be copied to all the nodes
        job.setJarByClass(DriverSalesPerStore.class);

        // Specify the mapper and reducer classes
        job.setMapperClass(MapperSalesPerStore.class);
        job.setReducerClass(ReducerSalesPerStore.class);

        // Specify the format of the output of the mapper and the reducer
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(DoubleWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);

        // Specify the paths for the input and output
        FileInputFormat.setInputPaths(job, new Path(input));
        FileOutputFormat.setOutputPath(job, new Path(output));

        // Start the job and wait for it to finish. If it finishes successfully return a 0, otherwise a 1.
        boolean success = job.waitForCompletion(true);
        System.exit(success ? 0 : 1);
    }
}
```

It is intended for software engineers!

What is Hive?

- A data warehousing infrastructure based on Hadoop
  - Leverages the benefits of the Hadoop infrastructure:
    - Massive scale, fault tolerance, commodity hardware
- Designed to enable easy data summarization, ad-hoc queries and analysis of large volumes of data
- Provides a simple query language: Hive QL, which is based on SQL
  - But also allows engineers to plug in their own scripts and programs
- Query execution via MapReduce, Tez or Spark

Hive: A higher level abstraction over Map Reduce

```java
//All reducers must inherit from Reducer.  This class takes 4 types as parameters:
//Reducer<TypeOfKeyIn, TypeOfValueIn, TypeOfKeyOut, TypeOfValueOut>
public class ReducerSalesPerStore
    extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {

    @Override
    public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
        throws IOException, InterruptedException {
        // Each line that the reducer will get looks like:
        // key: storeLocation, such as San Jose
        // values: a set of sales in dollar amount
        try {
            // Iterate and accummulate the total sales for this store
            double totalSales = 0;
            for (DoubleWritable sale : values) {
                totalSales += sale.get();
            }

            // The output of the mapper is written to the context
            // Notice the conversion to the hadoop types
            context.write(key, new DoubleWritable(totalSales));

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
// All mappers must inherit from Mapper.  This class takes 4 types as parameters:
// Mapper<TypeOfKeyIn, TypeOfValueIn, TypeOfKeyOut, TypeOfValueOut>
// For this example, we don't get/care about a KeyIn, since it will read a line of text,
// so we can set it to Object
public class MapperSalesPerStore
    extends Mapper<Object, Text, Text, DoubleWritable> {

    @Override
    public void map(Object key, Text value, Context context)
        throws IOException, InterruptedException {
        // Each line that the mapper will get looks like (tab separated):
        // 2012-01-01 09:00 San Jose Men's Clothing 214.05 Amex
        try {
            // Parse each line and get the store location and the sale value
            String line = value.toString().trim();
            String[] parts = line.split("\t");
            String storeLocation = parts[2];
            Double sale = Double.parseDouble(parts[4]);

            // The output of the mapper is written to the context
            // Notice the conversion to the hadoop types
            context.write(new Text(storeLocation), new DoubleWritable(sale));

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```java
public class DriverSalesPerStore {
    public static void main(String[] args) throws Exception {
        // Validate that two arguments were passed from the command line.
        String input, output;
        if (args.length == 2) {
            input = args[0];
            output = args[1];
        } else {
            System.err.println("Incorrect number of arguments.  Expected: input output");
            return;
        }

        // Create a instance of Job, based on a configuration, and give it a name (used in the reports and logs)
        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf);
        job.setJobName("MR - Sales Per Store");

        // Specify the jar file.  This will be copied to all the nodes
        job.setJarByClass(DriverSalesPerStore.class);

        // Specify the mapper and reducer classes
        job.setMapperClass(MapperSalesPerStore.class);
        job.setReducerClass(ReducerSalesPerStore.class);

        // Specify the format of the output of the mapper and the reducer
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(DoubleWritable.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(DoubleWritable.class);

        // Specify the paths for the input and output
        FileInputFormat.setInputPaths(job, new Path(input));
        FileOutputFormat.setOutputPath(job, new Path(output));

        // Start the job and wait for it to finish. If it finishes successfully return a 0, otherwise a 1.
        boolean success = job.waitForCompletion(true);
        System.exit(success ? 0 : 1);
    }
}
```

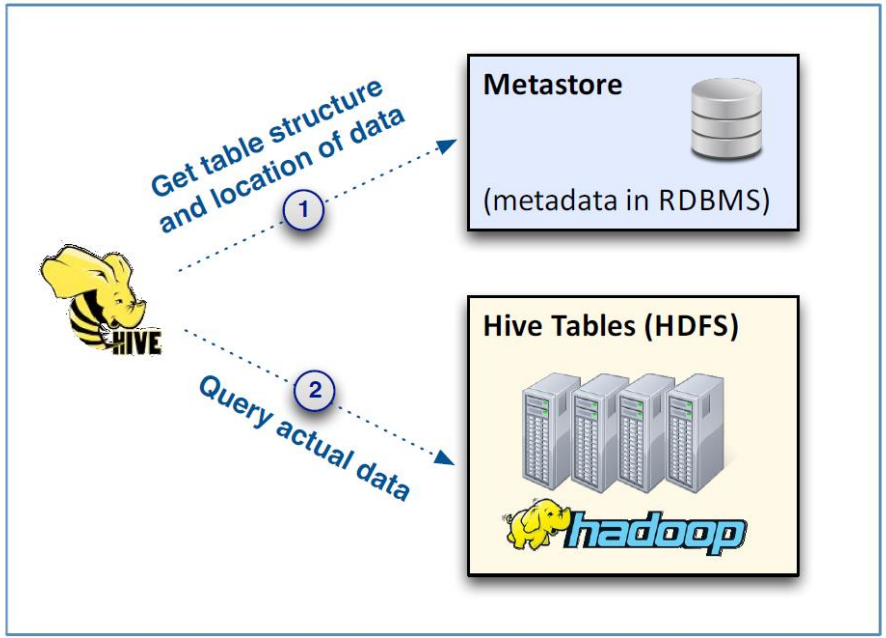
```
SELECT storeName, SUM(sales)
FROM purchases
GROUP BY storeName;
```

History of Hive

- Initially developed at Facebook
  - In 2008 it became an Apache project
- Wanted to bring the power of Hadoop to more people, not just developers
  - But keep the programmability exposed by Hadoop
- Leverage the ubiquity of SQL among engineers and analysts
- Store centralized meta-data about the datasets kept in Hadoop
- Current contributors come from:
  - Facebook, Qubole, LinkedIn, Hortonworks, Cloudera, DropBox, Microsoft, Intel, Yahoo, Twitter, …

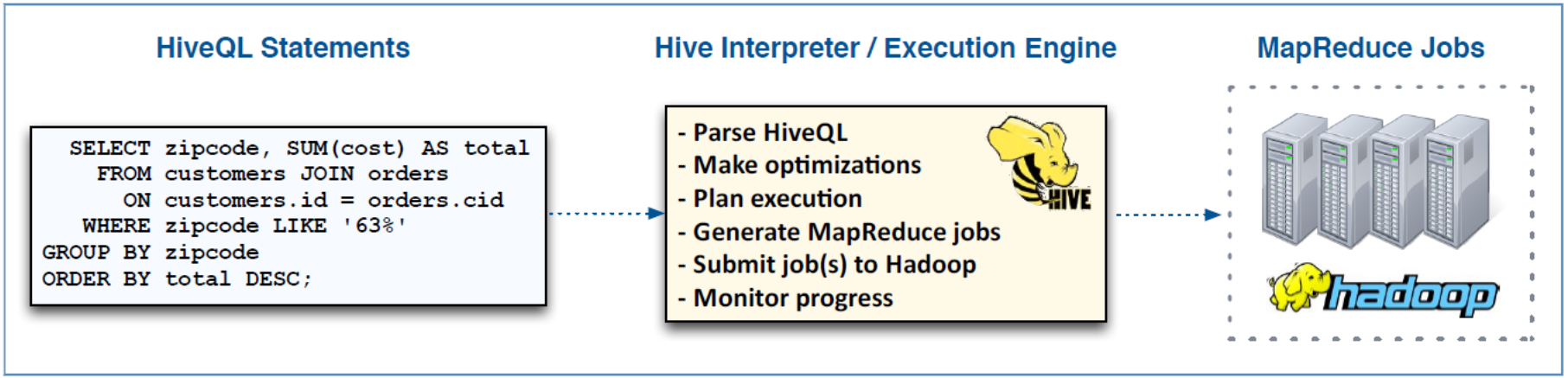
Metastore (HCatalog)

- Metastore to keep track of:
  - Structure of the data:
    - Databases, Tables, Columns
  - Location of the data
- Kept in a Relational Database
- Structure is independent of data
  - Schema on read
- Data and Metastore can be used by other Hadoop products



Execution model (simplified)

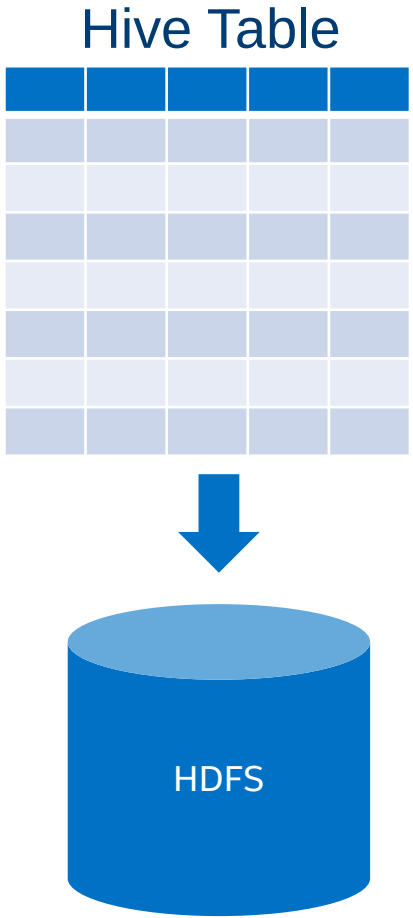
- Users write queries in HQL, and Hive transforms those into MapReduce jobs
  - Sent to the cluster for execution



CREATING TABLES IN HIVE DEMO

Creating table

- Two steps process:
  - CREATE TABLE
  - Move data to HDFS



Creating a table on HIVE

- **Basic syntax for creating a table:**

```
CREATE TABLE t (colname DATATYPE, ...)
    ROW FORMAT DELIMITED
    FIELDS TERMINATED BY char
    STORED AS {TEXTFILE|SEQUENCEFILE}
```

- **This creates a subdirectory under the `/user/hive/warehouse` directory in HDFS**
  - This is Hive's *warehouse directory*

Loading data from files

- **Loading data into a Hive table involves moving the data files into the Hive table warehouse directory in HDFS**
  - Can be done directly using `hadoop fs` commands
  - This example loads data from HDFS into Hive's `sales` table

```
$ hadoop fs -mv /depts/finance/salesdata \
/user/hive/warehouse/sales/
```

  - Alternatively, use Hive's `LOAD DATA INPATH` command

```
hive> LOAD DATA INPATH '/depts/finance/salesdata'
      INTO TABLE sales;
```

HIVE Table Data Types

- **Each type maps to a native data type in Java**

Type	Description
TINYINT	1 byte
SMALLINT	2 bytes
INT	4 bytes
BIGINT	8 bytes
FLOAT	Single precision
DOUBLE	Double precision
STRING	Sequence of characters
BOOLEAN	True/false
TIMESTAMP	YYYY-MM-DD HH:MM:SS.fffffffff

HIVE Complex Data Types

- **Hive also has three complex data types**
  - Extensions to standard SQL
- **Maps**
  - Key-value pairs
- **Arrays**
  - Lists of elements
- **Struct**
  - User defined structures

HIVE Complex Data Types – Basic Syntax

```
CREATE TABLE IF NOT EXISTS users_complex (
        id int, name string, salary bigint, phones array<string>,
        deductions map<string,int>,
        address struct<city:string,state:string,pin:bigint>)
ROW FORMAT DELIMITED
        FIELDS TERMINATED BY ','
        COLLECTION ITEMS TERMINATED BY '$'
        MAP KEYS TERMINATED BY '#'
    LINES TERMINATED BY '\n';
```

EXTERNAL VS INTERNAL TABLES DEMO

External vs Internal Tables

Use EXTERNAL tables when:

- The data is also used outside of Hive. For example, the data files are read and processed by an existing program that doesn't lock the files.
- Data needs to remain in the underlying location even after a DROP TABLE. This can apply if you are pointing multiple schemas (tables or views) at a single data set or if you are iterating through various possible schemas.
- Hive should not own data and control settings, dirs, etc., you have another program or process that will do those things.
- You are not creating table based on existing table (AS SELECT).

Use INTERNAL tables when:

- The data is temporary.
- You want Hive to completely manage the lifecycle of the table and data.

HIVE JOINS

HIVE Joins

- **A frequent requirement is to join two or more tables together**
- **Hive supports:**
  - Inner joins
  - Left outer joins
  - Right outer joins
  - Full outer joins
  - Left semi joins
    - Returns rows from the first table where one or more matches are found in the second table
    - E.g,. List the departments in a table that have at least one employee
  - **Note that:**
    - Before version 2.2.0, only equality joins are supported
    - a.id = b.id is supported

HIVE Joins Syntax

- **Hive uses the following syntax to join tables:**

```
SELECT cols FROM t1 JOIN t2 ON (condition)
JOIN t3 ON (condition)
```

- **JOIN can be replaced with LEFT OUTER JOIN, RIGHT OUTER JOIN, FULL OUTER JOIN or LEFT SEMI JOIN**
- **Note that the alternative syntax often used by popular RDBMSs is not supported before 0.13.0**

```
SELECT cols FROM t1, t2 WHERE …
```

HIVE Joins Optimization

- **Map-side Join**
  - Reads the smaller table to an in-memory hash table
  - Each mapper gets its own copy of the hash table
  - The join is performed in the mapper
  - No reduce or sort and shuffle phases are required!
  - As of 0.7.0 Hive no longer needs the hint and can perform this automatically

```
SELECT /*+mapjoin(B)*/ * FROM A JOIN B ON
A.Value = B.Value;
```

HIVE OPERATORS AND FUNCTIONS

Hive Operators and Functions

Operators - Built into Hive

- Relational Operators
- Arithmetic Operators
- Logical Operators
- Collection/Complex Type Constructors
- Operators on Collection/Complex Types
- Functions – Added and supported in Hive distribution
- Mathematical Functions

Hive Operators and Functions - continued

Operators - Built into Hive

- Collection/Complex Functions
- Type Conversion Functions
- Date Functions
- Conditional Functions
- String Functions
- Miscellaneous Functions

Aggregate Functions/UDAF

Relational Operators

Operator	Operand types	Description
A = B	All primitive types	TRUE if expression A is equal to expression B otherwise FALSE
A <=> B	All primitive types	Returns same result with EQUAL(=) operator for non-null operands, but returns TRUE if both are NULL, FALSE if one of the them is NULL
A == B	None!	Fails because of invalid syntax. SQL uses =, not ==
A <> B	All primitive types	NULL if A or B is NULL, TRUE if expression A is NOT equal to expression B otherwise FALSE
A != B	All primitive types	a synonym for the <> operator

Relational Operators (continued)

Operator	Operand types	Description
A < B	All primitive types	NULL if A or B is NULL, TRUE if expression A is less than expression B otherwise FALSE
A <= B	All primitive types	NULL if A or B is NULL, TRUE if expression A is less than or equal to expression B otherwise FALSE
A > B	All primitive types	NULL if A or B is NULL, TRUE if expression A is greater than expression B otherwise FALSE
A >= B	All primitive types	NULL if A or B is NULL, TRUE if expression A is greater than or equal to expression B otherwise FALSE
A [NOT] BETWEEN B AND C	All primitive types	NULL if A, B or C is NULL, TRUE if A is greater than or equal to B AND A less than or equal to C otherwise FALSE. This can be inverted by using the NOT keyword.

Relational Operators (continued)

Operator	Operand types	Description
A IS NULL	All types	TRUE if expression A evaluates to NULL otherwise FALSE
A IS NOT NULL	All types	FALSE if expression A evaluates to NULL otherwise TRUE
A LIKE B	strings	NULL if A or B is NULL, TRUE if string A matches the SQL simple regular expression B, otherwise FALSE. The comparison is done character by character. The _ character in B matches any character in A(similar to . in posix regular expressions) while the % character in B matches an arbitrary number of characters in A(similar to .* in posix regular expressions) e.g. 'foobar' like 'foo' evaluates to FALSE where as 'foobar' like 'foo_ _ _' evaluates to TRUE and so does 'foobar' like 'foo%'

Relational Operators (continued)

Operator	Operand types	Description
A RLIKE B	strings	NULL if A or B is NULL, TRUE if string A matches the Java regular expression B(See Java regular expressions syntax), otherwise FALSE e.g. 'foobar' rlike 'foo' evaluates to FALSE where as 'foobar' rlike '^f.*r$' evaluates to TRUE
A REGEXP B	strings	Same as RLIKE

Arithmetic Operators

The following operators support various common arithmetic operations on the operands. All return number types; if any of the operands are NULL, then the result is also NULL.

Operator	Operand types	Description
A + B	All number types	Gives the result of adding A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. e.g. since every integer is a float, therefore float is a containing type of integer so the + operator on a float and an int will result in a float.
A - B	All number types	Gives the result of subtracting B from A. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands.
A * B	All number types	Gives the result of multiplying A and B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands. Note that if the multiplication causing overflow, you will have to cast one of the operators to a type higher in the type hierarchy.

Arithmetic Operators (continued)

Operator	Operand types	Description
A / B	All number types	Gives the result of dividing B from A. The result is a double type.
A % B	All number types	Gives the remainder resulting from dividing A by B. The type of the result is the same as the common parent(in the type hierarchy) of the types of the operands.
A & B	All number types	Gives the result of bitwise AND of A and B. The type of the result is the same as the common parent (in the type hierarchy) of the types of the operands.
A | B	All number types	Gives the result of bitwise OR of A and B. The type of the result is the same as the common parent (in the type hierarchy) of the types of the operands.
A ^ B	All number types	Gives the result of bitwise XOR of A and B. The type of the result is the same as the common parent (in the type hierarchy) of the types of the operands.
~A	All number types	Gives the result of bitwise NOT of A. The type of the result is the same as the type of A.

Logical Operators

The following operators provide support for creating logical expressions. All of them return boolean TRUE, FALSE, or NULL depending upon the boolean values of the operands. NULL behaves as an "unknown" flag, so if the result depends on the state of an unknown, the result itself is unknown.

Operator	Operand types	Description
A AND B	boolean	TRUE if both A and B are TRUE, otherwise FALSE. NULL if A or B is NULL
A && B	boolean	Same as A AND B
A OR B	boolean	TRUE if either A or B or both are TRUE; FALSE OR NULL is NULL; otherwise FALSE
A || B	boolean	Same as A OR B
NOT A	boolean	TRUE if A is FALSE or NULL if A is NULL. Otherwise FALSE.
! A	boolean	Same as NOT A

Mathematical Functions

Return Type	Name(Signature)	Description
bigint	round(double a)	Returns the rounded BIGINT value of the double
double	round(double a, int d)	Returns the double rounded to d decimal places
bigint	floor(double a)	Returns the maximum BIGINT value that is equal or less than the double
bigint	ceil(double a), ceiling(double a)	Returns the minimum BIGINT value that is equal or greater than the double
double	rand(), rand(int seed)	Returns a random number (that changes from row to row) that is distributed uniformly from 0 to 1. Specifiying the seed will make sure the generated random number sequence is deterministic.
double	exp(double a)	Returns e^a where e is the base of the natural logarithm
double	ln(double a)	Returns the natural logarithm of the argument
double	log10(double a)	Returns the base-10 logarithm of the argument
double	log2(double a)	Returns the base-2 logarithm of the argument
double	log(double base, double a)	Return the base "base" logarithm of the argument
double	pow(double a, double p), power(double a, double p)	Return a^p
double	sqrt(double a)	Returns the square root of a

Mathematical Functions (continued)

Return Type	Name(Signature)	Description
string	bin(BIGINT a)	Returns the number in binary format (see [http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_bin])
string	hex(BIGINT a) hex(string a)	If the argument is an int, hex returns the number as a string in hex format. Otherwise if the number is a string, it converts each character into its hex representation and returns the resulting string. (see [http://dev.mysql.com/doc/refman/5.0/en/string-functions.html#function_hex])
string	unhex(string a)	Inverse of hex. Interprets each pair of characters as a hexidecimal number and converts to the character represented by the number.
string	conv(BIGINT num, int from_base, int to_base), conv(STRING num, int from_base, int to_base)	Converts a number from a given base to another (see [http://dev.mysql.com/doc/refman/5.0/en/mathematical-functions.html#function_conv])
double	abs(double a)	Returns the absolute value
int double	pmod(int a, int b) pmod(double a, double b)	Returns the positive value of a mod b

Mathematical Functions (continued)

Return Type	Name(Signature)	Description
double	sin(double a)	Returns the sine of a (a is in radians)
double	asin(double a)	Returns the arc sin of x if -1<=a<=1 or null otherwise
double	cos(double a)	Returns the cosine of a (a is in radians)
double	acos(double a)	Returns the arc cosine of x if -1<=a<=1 or null otherwise
double	tan(double a)	Returns the tangent of a (a is in radians)
double	atan(double a)	Returns the arctangent of a
double	degrees(double a)	Converts value of a from radians to degrees
double	radians(double a)	Converts value of a from degrees to radians
int double	positive(int a), positive(double a)	Returns a
int double	negative(int a), negative(double a)	Returns -a
float	sign(double a)	Returns the sign of a as '1.0' or '-1.0'
double	e()	Returns the value of e
double	pi()	Returns the value of pi

Aggregate Functions/UDAF

Return Type	Name(Signature)	Description
bigint	count(*)	count(*) - Returns the total number of retrieved rows, including rows containing NULL values; count(expr) - Returns the number of rows for which the supplied expression is non-NULL; count(DISTINCT expr[, expr]) - Returns the number of rows for which the supplied expression(s) are unique and non-NULL.
	count(expr)	
	count(DISTINCT expr)	
double	sum(col)	Returns the sum of the elements in the group or the sum of the distinct values of the column in the group
	sum(DISTINCT col)	
double	avg(col)	Returns the average of the elements in the group or the average of the distinct values of the column in the group
	avg(DISTINCT col)	
double	min(col)	Returns the minimum of the column in the group
double	max(col)	Returns the maximum value of the column in the group
double	variance(col)	Returns the variance of a numeric column in the group
	var_pop(col)	
double	var_samp(col)	Returns the unbiased sample variance of a numeric column in the group
double	stddev_pop(col)	Returns the standard deviation of a numeric column in the group
double	stddev_samp(col)	Returns the unbiased sample standard deviation of a numeric column in the group

Aggregate Functions/UDAF (continued)

Return Type	Name(Signature)	Description
double	covar_pop(col1, col2)	Returns the population covariance of a pair of numeric columns in the group
double	covar_samp(col1, col2)	Returns the sample covariance of a pair of a numeric columns in the group
double	corr(col1, col2)	Returns the Pearson coefficient of correlation of a pair of a numeric columns in the group
double	percentile(BIGINT col, p)	Returns the exact p^{th} percentile of a column in the group (does not work with floating point types). p must be between 0 and 1. NOTE: A true percentile can only be computed for integer values. Use PERCENTILE_APPROX if your input is non-integral.
array<double>	percentile(BIGINT col, array(p_1 [, p_2]...))	Returns the exact percentiles p_1, p_2, ... of a column in the group (does not work with floating point types). p_i must be between 0 and 1. NOTE: A true percentile can only be computed for integer values. Use PERCENTILE_APPROX if your input is non-integral.

Aggregate Functions/UDAF (continued)

Return Type	Name(Signature)	Description
double	percentile_approx(DOUBLE col, p [, B])	Returns an approximate pth percentile of a numeric column (including floating point types) in the group. The B parameter controls approximation accuracy at the cost of memory. Higher values yield better approximations, and the default is 10,000. When the number of distinct values in col is smaller than B, this gives an exact percentile value.
array<double>	percentile_approx (DOUBLE col, array(p1 [, p2]...) [, B])	Same as above, but accepts and returns an array of percentile values instead of a single one.
array<struct {'x','y'}>	histogram_numeric(col, b)	Computes a histogram of a numeric column in the group using b non-uniformly spaced bins. The output is an array of size b of double-valued (x,y) coordinates that represent the bin centers and heights
array	collect_set(col)	Returns a set of objects with duplicate elements eliminated

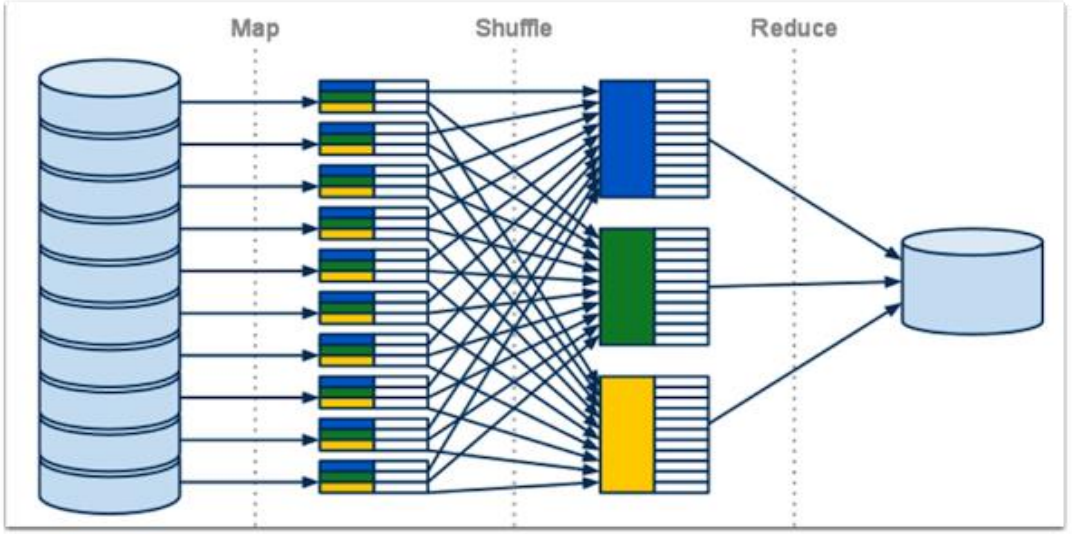
IMPALA

What is Impala ?

- SQL engine for processing the data stored in HBase and HDFS.
  - Impala uses Hive metastore and can query the Hive tables directly.
- Unlike Hive, Impala does not translate the queries into MapReduce jobs but executes them natively.
  - Faster execution and response time !

What is wrong with Map Reduce ?

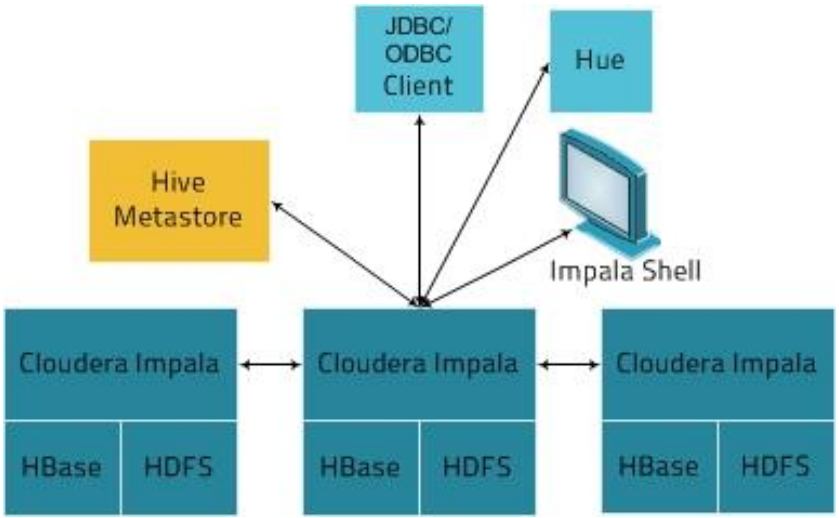
- Batch Oriented
- High latency
- Complex, only for developers !



Impala

- Inspired by Google Dremel project
  - Distributed system for interactively querying large datasets
- Very low latency measured in milli-seconds
- Ideal for interactive real time analysis
- Impala supports a dialect of SQL
  - Provides a high degree of compatibility with HiveQL
- Impala was developed by cloudera
  - Donated to Apache foundation

Impala high level architecture



- **Clients** - Entities including Hue, ODBC clients, JDBC clients, and the Impala Shell can all interact with Impala

- **Hive Metastore** - Stores information about the data available to Impala.

- **Cloudera Impala** - This process, which runs on DataNodes, coordinates and executes queries.

- **HBase and HDFS** - Storage for data to be queried.

Impala and Hive Comparison

HIVE

- High latency due to map reduce as execution engine
- Fault-tolerance, materializes all intermediate results
- Batch processing for long running, memory intensive, big data operations

IMPALA

- Much better performance for querying
- No fault tolerance, no checkpoints in intermediate steps
- An execution engine designed for low runtime overhead

Recommendations

Impala is faster than Apache Hive but its not the GO TO SQL solution for every need.

Impala is memory intensive and does not run efficiently on heavy operations since everything has to be pushed to memory (like joins)

HIVE

If there are batch processing over big data or if fault tolerance is very important then HIVE is a better solution

IMPALA

For real time processing of ad-hoc queries on subset of data

Impala Supported Data Types

- BIGINT
- BOOLEAN
- CHAR
- DECIMAL
- DOUBLE
- FLOAT
- INT
- REAL
- SMALLINT
- STRING
- TIMESTAMP
- TINYINT
- VARCHAR
- ARRAY Complex Type (Impala 2.3 and higher)
- MAP Complex Type (Impala 2.3 and higher)
- STRUCT Complex Type (Impala 2.3 and higher)

Accessing a table containing any columns with unsupported types causes an error.

IMPALA DEMO

PIG

Remember Map Reduce?

- Programming paradigm
  - Divide and Conquer:
    - Work is performed by nodes of a cluster
    - *Moving processing is better than moving data*
- Framework for parallel computing
  - Abstracts the user from issues of:
    - Parallelization
    - Remote execution
    - Data Distribution
    - Load Balancing
    - Fault Tolerance
- Provides a simple API for the programmer:

	Input	Output
• Map:	(k1, v1)	list(k2, v2)
• Reduce:	(k2, list(v2))	list(k3, v3)

It is very powerful, but…

```java
//All reducers must inherit from Reducer.  This class takes 4 types as parameters:
//Reducer<TypeOfKeyIn, TypeOfValueIn, TypeOfKeyOut, TypeOfValueOut>
public class ReducerSalesPerStore
     extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {

     @Override
     public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
          throws IOException, InterruptedException {
          // Each line that the reducer will get looks like:
          // key: storeLocation, such as San Jose
          // values: a set of sales in dollar amount
          try {
               // Iterate and accummulate the total sales for this store
               double totalSales = 0;
               for (DoubleWritable sale : values) {
                    totalSales += sale.get();
               }

               // The output of the mapper is written to the context
               // Notice the conversion to the hadoop types
               context.write(key, new DoubleWritable(totalSales));

          } catch (Exception e) {
               e.printStackTrace();
          }
     }
}
// All mappers must inherit from Mapper.  This class takes 4 types as parameters:
// Mapper<TypeOfKeyIn, TypeOfValueIn, TypeOfKeyOut, TypeOfValueOut>
// For this example, we don't get/care about a KeyIn, since it will read a line of text,
// so we can set it to Object
public class MapperSalesPerStore
     extends Mapper<Object, Text, Text, DoubleWritable> {

     @Override
     public void map(Object key, Text value, Context context)
          throws IOException, InterruptedException {
          // Each line that the mapper will get looks like (tab separated):
          // 2012-01-01 09:00 San Jose Men's Clothing 214.05 Amex
          try {
               // Parse each line and get the store location and the sale value
               String line = value.toString().trim();
               String[] parts = line.split("\t");
               String storeLocation = parts[2];
               Double sale = Double.parseDouble(parts[4]);

               // The output of the mapper is written to the context
               // Notice the conversion to the hadoop types
               context.write(new Text(storeLocation), new DoubleWritable(sale));

          } catch (Exception e) {
               e.printStackTrace();
          }
     }
}
```

```java
public class DriverSalesPerStore {
     public static void main(String[] args) throws Exception {
          // Validate that two arguments were passed from the command line.
          String input, output;
          if (args.length == 2) {
               input = args[0];
               output = args[1];
          } else {
               System.err.println("Incorrect number of arguments.  Expected: input output");
               return;
          }

          // Create a instance of Job, based on a configuration, and give it a name (used in the reports and logs)
          Configuration conf = new Configuration();
          Job job = Job.getInstance(conf);
          job.setJobName("MR - Sales Per Store");

          // Specify the jar file.  This will be copied to all the nodes
          job.setJarByClass(DriverSalesPerStore.class);

          // Specify the mapper and reducer classes
          job.setMapperClass(MapperSalesPerStore.class);
          job.setReducerClass(ReducerSalesPerStore.class);

          // Specify the format of the output of the mapper and the reducer
          job.setMapOutputKeyClass(Text.class);
          job.setMapOutputValueClass(DoubleWritable.class);
          job.setOutputKeyClass(Text.class);
          job.setOutputValueClass(DoubleWritable.class);

          // Specify the paths for the input and output
          FileInputFormat.setInputPaths(job, new Path(input));
          FileOutputFormat.setOutputPath(job, new Path(output));

          // Start the job and wait for it to finish. If it finishes successfully return a 0, otherwise a 1.
          boolean success = job.waitForCompletion(true);
          System.exit(success ? 0 : 1);
     }
}
```

It is intended for software engineers!

Motivation and History

- MapReduce is very powerful, but:
  - It requires a Java Programmer
  - Programmer has to re-invent common functionality (join, filter, etc.)
  - It is pretty low level

- Around 2006 the folks at Yahoo! Research Lab created Pig Latin as a high level data flow language for creating MapReduce programs
  - In 2007 it was moved into the Apache Software Foundation

[SIGMOD 2008]

What is Pig?

- It's a data flow query language
- Data analysis tasks are expressed in terms of set-oriented transformations:
  - Apply a function to every record or group of records
  - **Everything** is a 2-dimensional table
- Pig scripts are compiled into MapReduce code and executed in the cluster
  - One pig scripts will produce one or more MRs
- It can consume any format, and produce any format
- Designed to:
  - *"fit in a sweet spot between the declarative style of SQL, and the low-level, procedural style of map-reduce"*

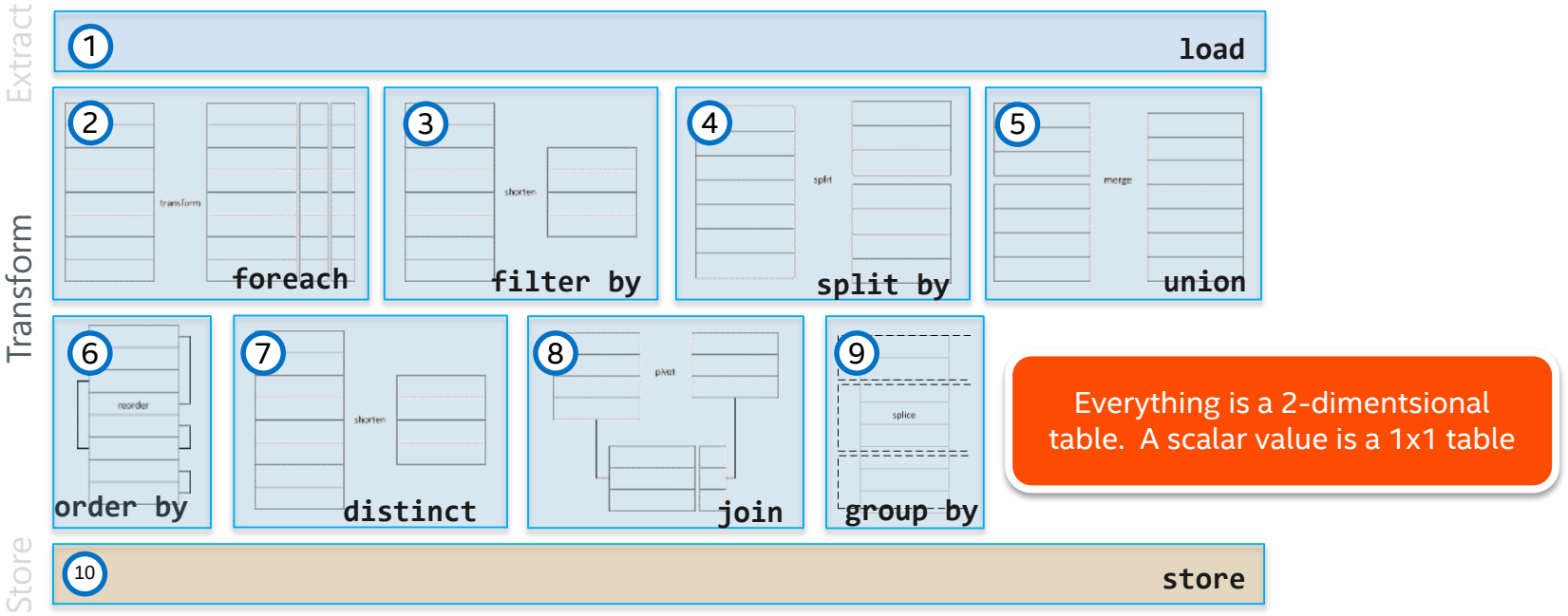
Pig: A higher level abstraction over Map Reduce

```java
//All reducers must inherit from Reducer.  This class takes 4 types as parameters:
//Reducer<TypeOfKeyIn, TypeOfValueIn, TypeOfKeyOut, TypeOfValueOut>
public class ReducerSalesPerStore
     extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {

     @Override
     public void reduce(Text key, Iterable<DoubleWritable> values, Context context)
          throws IOException, InterruptedException {
          // Each line that the reducer will get looks like:
          // key: storeLocation, such as San Jose
          // values: a set of sales in dollar amount
          try {
               // Iterate and accummulate the total sales for this store
               double totalSales = 0;
               for (DoubleWritable sale : values) {
                    totalSales += sale.get();
               }

               // The output of the mapper is written to the context
               // Notice the conversion to the hadoop types
               context.write(key, new DoubleWritable(totalSales));

          } catch (Exception e) {
               e.printStackTrace();
          }
     }
}
// All mappers must inherit from Mapper.  This class takes 4 types as parameters:
// Mapper<TypeOfKeyIn, TypeOfValueIn, TypeOfKeyOut, TypeOfValueOut>
// For this example, we don't get/care about a KeyIn, since it will read a line of text,
// so we can set it to Object
public class MapperSalesPerStore
     extends Mapper<Object, Text, Text, DoubleWritable> {

     @Override
     public void map(Object key, Text value, Context context)
          throws IOException, InterruptedException {
          // Each line that the mapper will get looks like (tab separated):
          // 2012-01-01 09:00 San Jose Men's Clothing 214.05 Amex
          try {
               // Parse each line and get the store location and the sale value
               String line = value.toString().trim();
               String[] parts = line.split("\t");
               String storeLocation = parts[2];
               Double sale = Double.parseDouble(parts[4]);

               // The output of the mapper is written to the context
               // Notice the conversion to the hadoop types
               context.write(new Text(storeLocation), new DoubleWritable(sale));

          } catch (Exception e) {
               e.printStackTrace();
          }
     }
}
```

```java
public class DriverSalesPerStore {
     public static void main(String[] args) throws Exception {
          // Validate that two arguments were passed from the command line.
          String input, output;
          if (args.length == 2) {
               input = args[0];
               output = args[1];
          } else {
               System.err.println("Incorrect number of arguments.  Expected: input output");
               return;
          }

          // Create a instance of Job, based on a configuration, and give it a name (used in the reports and logs)
          Configuration conf = new Configuration();
          Job job = Job.getInstance(conf);
          job.setJobName("MR - Sales Per Store");

          // Specify the jar file.  This will be copied to all the nodes
          job.setJarByClass(DriverSalesPerStore.class);

          // Specify the mapper and reducer classes
          job.setMapperClass(MapperSalesPerStore.class);
          job.setReducerClass(ReducerSalesPerStore.class);

          // Specify the format of the output of the mapper and the reducer
          job.setMapOutputKeyClass(Text.class);
          job.setMapOutputValueClass(DoubleWritable.class);
          job.setOutputKeyClass(Text.class);
          job.setOutputValueClass(DoubleWritable.class);

          // Specify the paths for the input and output
          FileInputFormat.setInputPaths(job, new Path(input));
          FileOutputFormat.setOutputPath(job, new Path(output));

          // Start the job and wait for it to finish. If it finishes successfully return a 0, otherwise a 1.
          boolean success = job.waitForCompletion(true);
          System.exit(success ? 0 : 1);
     }
}
```

```
A = LOAD 'purchases.txt' USING PigStorage() AS
    (date:chararray, time:chararray, storeName:chararray,
     category:chararray, sales:double, payment:chararray);
B = GROUP A BY storeName;
C = FOREACH B GENERATE group, SUM(A.sales);
STORE C INTO 'salesPerStore' USING PigStorage();
```

Main Pig Operators



Pig Data Types

Simple Types	Description	Example
int	Signed 32-bit integer	10
long	Signed 64-bit integer	10L
float	32-bit floating point	10.5F
double	64-bit floating point	10.5
chararray	Character array (string) in Unicode UTF-8	hello world
bytearray	Byte array (blob)	*0xcafebabe*
boolean	boolean	true/false (case insensitive)
datetime	datetime	1970-01-01T00:00:00.000+00:00
biginteger	Java BigInteger	200000000000
bigdecimal	Java BigDecimal	33.4567833213234412334 42
Complex Types	**Description**	**Example**
tuple	An ordered set of fields.	(19,2)
bag	An collection of tuples.	{(19,2), (18,1)}
map	A set of key value pairs.	[open#apache, apache#hadoop]

CLOUDERA ODBC DRIVER

Information Technology

Cloudera ODBC Driver

The Cloudera ODBC Driver enables your enterprise users to access Hadoop data through Business Intelligence (BI) applications with ODBC support.

Hive

https://www.cloudera.com/downloads/connectors/hive/odbc/2-5-12.html

Impala

https://www.cloudera.com/downloads/connectors/impala/odbc/2-5-41.html

R + SPARK

Sparklyr

It is a package that provides an interface between R and Apache Spark

- Interactively manipulate Spark data using both dplyr and SQL (via DBI).
- Filter and aggregate Spark datasets then bring them into R for analysis and visualization.
- Integrated support for establishing Spark connections and browsing Spark data frames within the RStudio IDE.

https://spark.rstudio.com/

THANK YOU!

Questions?