Aurelius Corporate Solutions

Course- Big Data Hadoop (Basic)

Aurelius Corporate Solutions
A-125, Sector 63, Noida, Uttar Pradesh – 201307, India
1000+ Unique Technologies Projects Delivered | 500+ Corporate Customers Worldwide | 50000+ Professionals Trained on 40+ Domains in Over 30 Countries

Course Topics

- ✓ Welcome to Big Data World
  - ✓ Understanding Big Data
  - ✓ Hadoop Architecture
- ✓ HDFS
  - ✓ Deep dive in HDFS Architecture
  - ✓ HDFS APIs
  - ✓ Introduction to HDP Sandbox
  - ✓ HDFS Hands – 1 Hour
- ✓ Introduction to YARN & MR
  - ✓ Hadoop MapReduce framework
  - ✓ Programming in Map Reduce
- ✓ Advance Map Reduce
  - ✓ Understanding Counters
  - ✓ Differences between MR1 & MR2
  - ✓ Introduction to MR API
  - ✓ Overview of Map Side Join
  - ✓ Overview of Reduce Side Join
  - ✓ Map Reduce Hands On – 1 hour
- ✓ Hive
  - ✓ Analytics using Hive
  - ✓ Understanding HIVE QL
- ✓ Advanced Hive
  - ✓ Advance Hive
  - ✓ Hive Hands On – 1 Hour
- ✓ NoSQL & HBase
  - ✓ CAP Theorem
  - ✓ NoSQL Databases and HBASE
  - ✓ HBase Architecture
  - ✓ HBase Schema Design
  - ✓ Difference between Hive & Hbase
  - ✓ Hbase Hands On – 1 Hour
- ✓ Apache Spark
  - ✓ Introduction to Spark
  - ✓ Why Spark?
  - ✓ Spark Stack Overview
  - ✓ Overview of RDD, Data Frame & Data Set
  - ✓ Spark Actions & Transformation Overview

Aurelius™ CORPORATE SOLUTIONS

Topics for Day 1: Covered

- ✓ Team Introduction
- ✓ Introduction to Big Data – Why and What?
- ✓ Characteristics of Big Data (4Vs)
- ✓ Overview of Big Data Ecosystem
- ✓ What is Hadoop?
- ✓ History of Hadoop

Tea Break

- ✓ Components of Hadoop
- ✓ Introduction to HDFS
- ✓ HDFS Architecture – Name Node / Data Node, Concept of Blocks
- ✓ File Formats in Hadoop
- ✓ HDFS API walk through
- ✓ Anatomy of a File Write and Read

Lunch Break

- ✓ Overview of Lab environment – HDP sandbox etc.
- ✓ HDFS Hands on – Getting Familiar with HDFS most commonly used commands
- ✓ Introduction to Map Reduce
- ✓ Map Reduce Phases – Map, Shuffle-Sort and Reduce
- ✓ Map Reduce Job Submission Flow

Topics for Today (Day 2)

- ✓ Any question from Day 1
- ✓ Understanding Counters
- ✓ Difference Between MR1 & MR2

Tea Break

- ✓ Job Class, GenericOptionsParser, Mapper & Reducer
- ✓ Distributed Cache
- ✓ Custom Input Format
- ✓ Overview of Map Side Join & Reduce Side Join

Lunch Break

- ✓ Map Reduce Hands On
- ✓ Data Integration Choices – Sqoop, Flume
- ✓ Introduction to Hive
  - ✓ Hive Architecture
  - ✓ Working with Schema
  - ✓ Introduction to Hive QL
  - ✓ Partitioning & Bucketing

Counters

Counters are lightweight objects in Hadoop that allow you to keep track of system progress in both the map and reduce stages of processing.

Counters are used to gather information about the data we are analysing, like how many types of records were processed, how many invalid records were found while running the job, etc.

```
17/12/13 09:29:11 INFO mapreduce.Job:  map 0% reduce 0%
17/12/13 09:29:18 INFO mapreduce.Job:  map 100% reduce 0%
17/12/13 09:29:32 INFO mapreduce.Job:  map 100% reduce 100%
17/12/13 09:29:32 INFO mapreduce.Job: Job job_1512999122732_0045 completed successfully
17/12/13 09:29:32 INFO mapreduce.Job: Counters: 53
        File System Counters
                FILE: Number of bytes read=93
                FILE: Number of bytes written=531461
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=153
                HDFS: Number of bytes written=21
                HDFS: Number of read operations=12
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=6
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=3
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=3566
                Total time spent by all reduces in occupied slots (ms)=33134
                Total time spent by all map tasks (ms)=3566
                Total time spent by all reduce tasks (ms)=33134
                Total vcore-milliseconds taken by all map tasks=3566
                Total vcore-milliseconds taken by all reduce tasks=33134
                Total megabyte-milliseconds taken by all map tasks=14606336
                Total megabyte-milliseconds taken by all reduce tasks=135716864
```

Counters

Counters

Two types of counters:

1. Hadoop Built-In counters: There are some built-in counters which exist per job. Below are built-in counter groups-

- **MapReduce Task Counters** - Collects task specific information (e.g., number of input records) during its execution time.
- **FileSystem Counters** - Collects information like number of bytes read or written by a task
- **FileInputFormat Counters** - Collects information of number of bytes read through FileInputFormat
- **FileOutputFormat Counters** - Collects information of number of bytes written through FileOutputFormat
- **Job Counters -** These counters are used by MRAppMaster. Statistics collected by them include e.g., number of task launched for a job.

2. User Defined Counters

In addition to built-in counters, user can define his own counters using similar functionalities provided by programming languages. For example, in Java 'enum' are used to define user defined counters.

WordCount Program Execution Revisit: Counter

hadoop jar hadoop-mapreduce-examples-2.6.0.jar wordcount /user/root/samplemr/input.txt /user/root/samplemr/wordcountOuput4

```
[root@sandbox-hdp ~]# hadoop jar hadoop-mapreduce-examples-2.6.0.jar wordcount /user/root/samplemr/input.txt /user/root/samplemr/wordcountOuput4
18/03/15 13:43:07 INFO client.RMProxy: Connecting to ResourceManager at sandbox-hdp.hortonworks.com/172.17.0.2:8032
18/03/15 13:43:07 INFO client.AHSProxy: Connecting to Application History server at sandbox-hdp.hortonworks.com/172.17.0.2:10200
18/03/15 13:43:08 INFO input.FileInputFormat: Total input paths to process : 1
18/03/15 13:43:08 INFO mapreduce.JobSubmitter: number of splits:1
18/03/15 13:43:09 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1521118455171_0006
18/03/15 13:43:09 INFO impl.YarnClientImpl: Submitted application application_1521118455171_0006
18/03/15 13:43:09 INFO mapreduce.Job: The url to track the job: http://sandbox-hdp.hortonworks.com:8088/proxy/application_1521118455171_0006/
18/03/15 13:43:09 INFO mapreduce.Job: Running job: job_1521118455171_0006
18/03/15 13:43:16 INFO mapreduce.Job: Job job_1521118455171_0006 running in uber mode : false
18/03/15 13:43:16 INFO mapreduce.Job:  map 0% reduce 0%
18/03/15 13:43:22 INFO mapreduce.Job:  map 100% reduce 0%
18/03/15 13:43:30 INFO mapreduce.Job:  map 100% reduce 100%
18/03/15 13:43:31 INFO mapreduce.Job: Job job_1521118455171_0006 completed successfully
18/03/15 13:43:31 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=42
                FILE: Number of bytes written=305435
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=158
                HDFS: Number of bytes written=24
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=3517
                Total time spent by all reduces in occupied slots (ms)=4948
                Total time spent by all map tasks (ms)=3517
                Total time spent by all reduce tasks (ms)=4948
                Total vcore-milliseconds taken by all map tasks=3517
                Total vcore-milliseconds taken by all reduce tasks=4948
                Total megabyte-milliseconds taken by all map tasks=879250
                Total megabyte-milliseconds taken by all reduce tasks=1237000
```

WordCount Program Execution Revisit: Counter Cont.

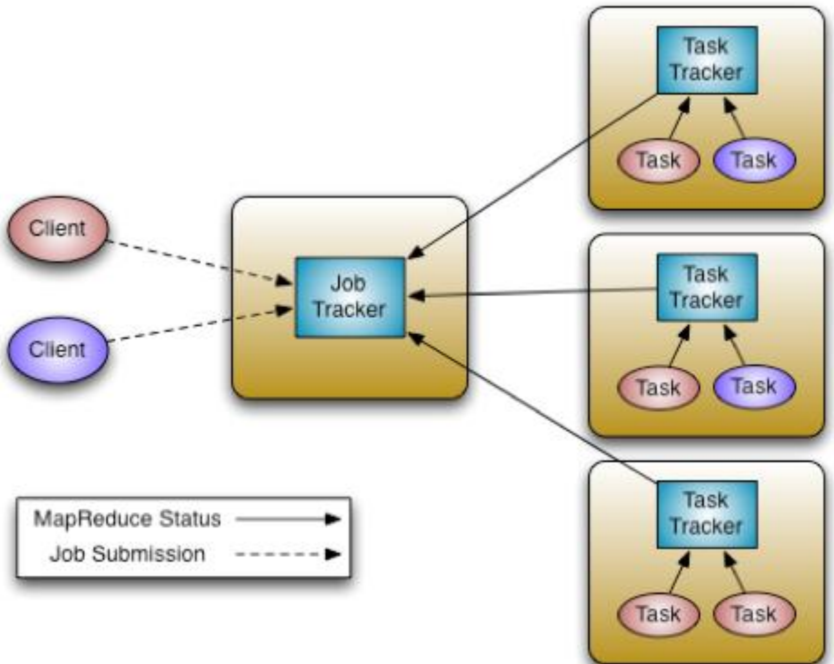
```
        Map-Reduce Framework
                Map input records=3
                Map output records=4
                Map output bytes=40
                Map output materialized bytes=42
                Input split bytes=133
                Combine input records=4
                Combine output records=3
                Reduce input groups=3
                Reduce shuffle bytes=42
                Reduce input records=3
                Reduce output records=3
                Spilled Records=6
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=271
                CPU time spent (ms)=1510
                Physical memory (bytes) snapshot=341209088
                Virtual memory (bytes) snapshot=4283138048
                Total committed heap usage (bytes)=151519232
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=25
        File Output Format Counters
                Bytes Written=24
[root@sandbox-hdp ~]#
```

User defined Counters

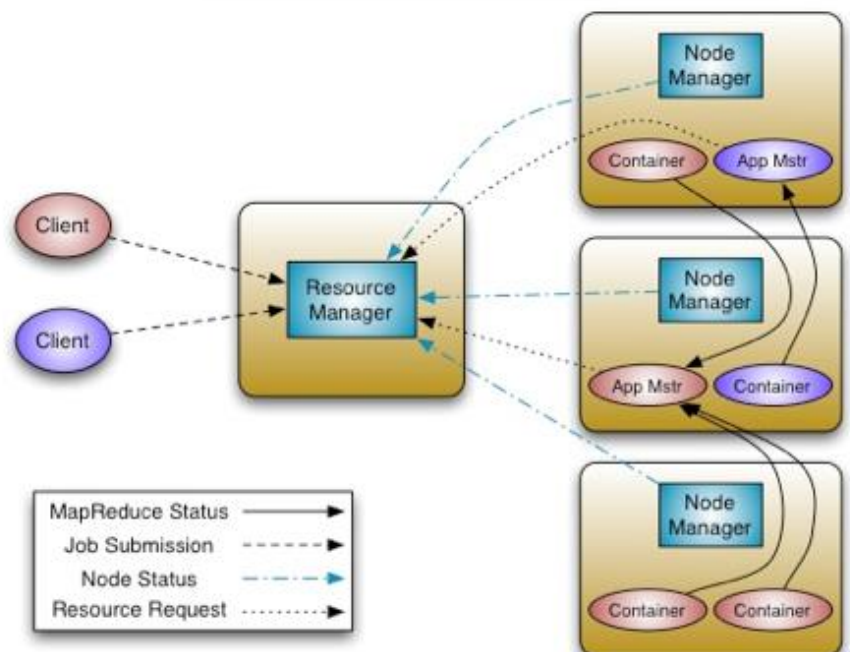
```
public static class MapClass
            extends MapReduceBase
            implements Mapper<LongWritable, Text, Text, Text>
{
    static enum SalesCounters { MISSING, INVALID };
    public void map ( LongWritable key, Text value,
                 OutputCollector<Text, Text> output,
                 Reporter reporter) throws IOException
    {

        //Input string is split using ',' and stored in 'fields' array
        String fields[] = value.toString().split(",", -20);
        //Value at 4th index is country. It is stored in 'country' variable
        String country = fields[4];

        //Value at 8th index is sales data. It is stored in 'sales' variable
        String sales = fields[8];

        if (country.length() == 0) {
            reporter.incrCounter(SalesCounters.MISSING, 1);
        } else if (sales.startsWith("\"")) {
            reporter.incrCounter(SalesCounters.INVALID, 1);
        } else {
            output.collect(new Text(country), new Text(sales + ",1"));
        }
    }
}
```

Difference between MR1 & MR2

Classic MapReduce (MR-1)



YARN (MR-2)
High Availability



Job Class, GenericOptionsParser, Mapper & Reducer

- Code Walk through of Map Reduce Programming:
- http://grepcode.com/file/repo1.maven.org/maven2/org.apache.hadoop/hadoop-mapreduce-examples/2.6.0/org/apache/hadoop/examples/WordCount.java

Distributed Cache

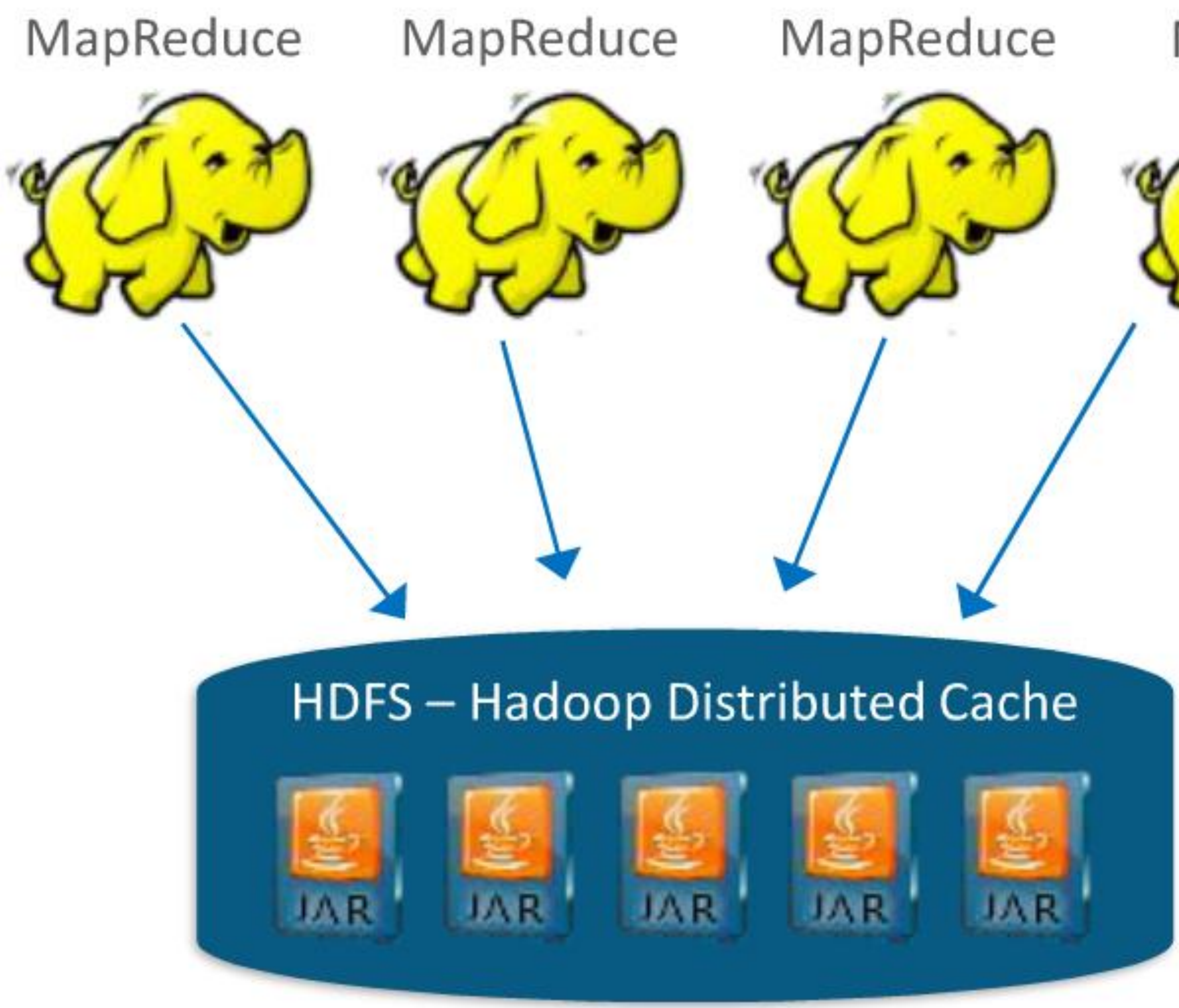
Rather than serializing side data in the job configuration, it is preferable to distribute datasets using Hadoop's distributed cache mechanism. This provides a service for copying files and archives to the task nodes in time for the tasks to use them when they run. To save network bandwidth, files are normally copied to any particular node once per job.

Distributed Cache is a facility provided by the Map-Reduce framework to cache files (text, archives, jars etc.) needed by applications

Files are copied only once per job and should not be modified by the application or externally while the job is executing.

Distributed Cache can be used to distribute simple, read-only data/text files and/or more complex types such as archives, jars etc via the JobConf.

Distributed Cache Cont.



Input Format

- As a MapReduce application writer, you don’t need to deal with InputSplits directly, as they are created by an InputFormat
- Hence an InputFormat is responsible for creating the input splits and dividing them into records).

```
public abstract class InputFormat<K, V> {
  public abstract List<InputSplit> getSplits(JobContext context)
      throws IOException, InterruptedException;

  public abstract RecordReader<K, V>
      createRecordReader(InputSplit split, TaskAttemptContext context)
          throws IOException, InterruptedException;
}
```

Mapper phase revisit: What InputFormat Does

- The client running the job calculates the splits for the job by calling getSplits(), then sends them to the application master, which uses their storage locations to schedule map tasks that will process them on the cluster.
- The map task passes the split to the createRecordReader() method on InputFormat to obtain a RecordReader for that split. A RecordReader is little more than an iterator over records, and the map task uses one to generate record key-value pairs, which it passes to the map function. We can see this by looking at the Mapper's run() method:

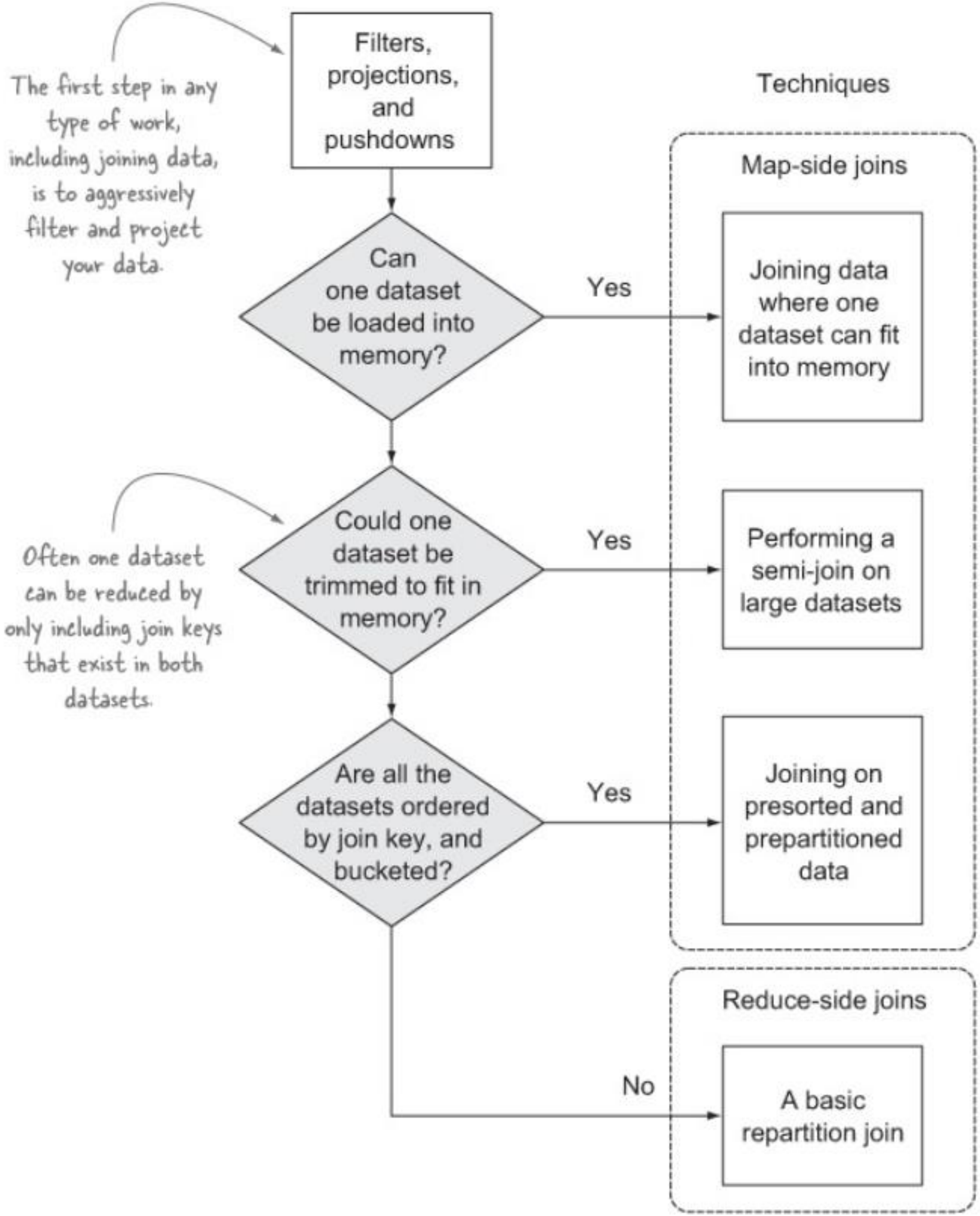
```
public void run(Context context) throws IOException, InterruptedException {
  setup(context);
  while (context.nextKeyValue()) {
    map(context.getCurrentKey(), context.getCurrentValue(), context);
  }
  cleanup(context);
}
```

Custom InputFormat

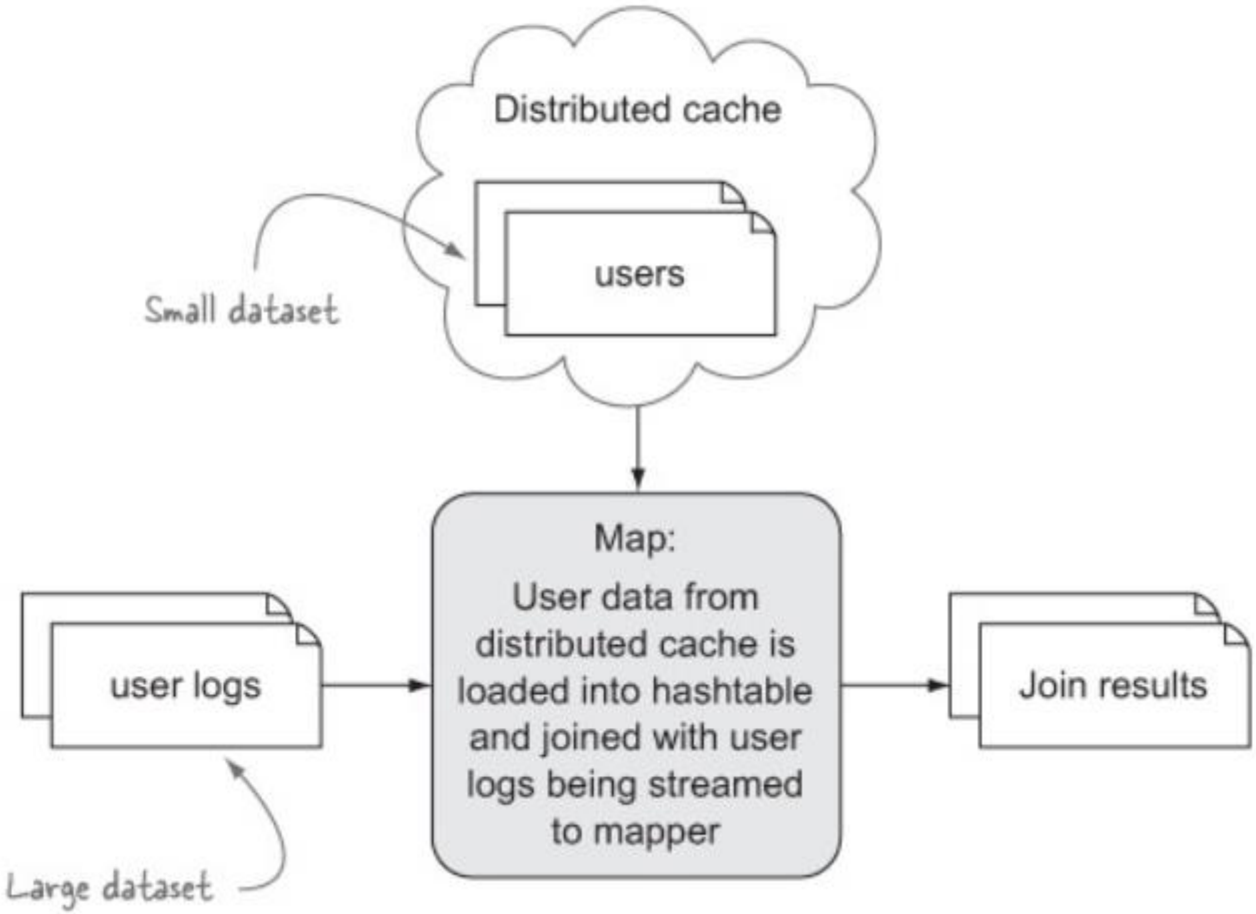
- Hence if you would like to receive different Key, Value pair in map() method, create your own custom InputFormat
- An example of Custom Input Format:
- http://blog.enablecloud.com/2014/05/writing-custom-hadoop-writable-and.html

Joins in Big Data

- *Replication join*
  - A map-side join that works in situations where one of the datasets is small enough to cache
- *Semi-join*
  - Another map-side join where one dataset is initially too large to fit into memory, but after some filtering can be reduced down to a size that can fit in memory
- Joining on Presorted & Prepartitioned data
  - Another map-side join where one dataset is initially too large to fit into memory, but after some filtering can be reduced down to a size that can fit in memory
- *Repartition join*
  - A reduce-side join for situations where you're joining two or more large datasets together


The first step in any type of work, including joining data, is to aggressively filter and project your data.
Filters, projections, and pushdowns
Techniques
Map-side joins
Can one dataset be loaded into memory?
Yes
Joining data where one dataset can fit into memory
Often one dataset can be reduced by only including join keys that exist in both datasets.
Could one dataset be trimmed to fit in memory?
Yes
Performing a semi-join on large datasets
Are all the datasets ordered by join key, and bucketed?
Yes
Joining on presorted and prepartitioned data
Reduce-side joins
No
A basic repartition join

Map Side Join



```
public void run(Path usersPath, Path userLogsPath, Path outputPath) {

  Configuration conf = super.getConf();

  Job job = new Job(conf);

  job.setJarByClass(ReplicatedJoin.class);
  job.setMapperClass(JoinMap.class);

  job.addCacheFile(usersPath.toUri());
  job.getConfiguration().set(
    JoinMap.DISTCACHE_FILENAME, usersPath.getName());

  job.setNumReduceTasks(0);

  FileInputFormat.setInputPaths(job, userLogsPath);
  FileOutputFormat.setOutputPath(job, outputPath);

  job.waitForCompletion(true);
}

public static class JoinMap
                extends Mapper<LongWritable, Text, Text, Text> {
  public static final String DISTCACHE_FILENAME = "distcachefile";
  private Map<String, User> users = new HashMap<String, User>();

  @Override
  protected void setup(Context context)
      throws IOException, InterruptedException {

    URI[] files = context.getCacheFiles();

    final String distributedCacheFilename =
        context.getConfiguration().get(DISTCACHE_FILENAME_CONFIG);

    for (URI uri: files) {
      File path = new File(uri.getPath());

      if (path.getName().equals(distributedCacheFilename)) {
        loadCache(path);
        break;
      }
    }
  }

  private void loadCache(File file) throws IOException {
    for(String line: FileUtils.readLines(file)) {
      User user = User.fromString(line);
      users.put(user.getName(), user);
    }
  }

  @Override
  protected void map(LongWritable offset, Text value, Context context)
      throws IOException, InterruptedException {

    UserLog userLog = UserLog.fromText(value);
    User user = users.get(userLog.getName());
    if (user != null) {
      context.write(
          new Text(user.toString()),
          new Text(userLog.toString()));
    }
  }
}
```

Add the users file to the distributed cache.

Save the users filename to the job config.

The larger user log file is the job input.

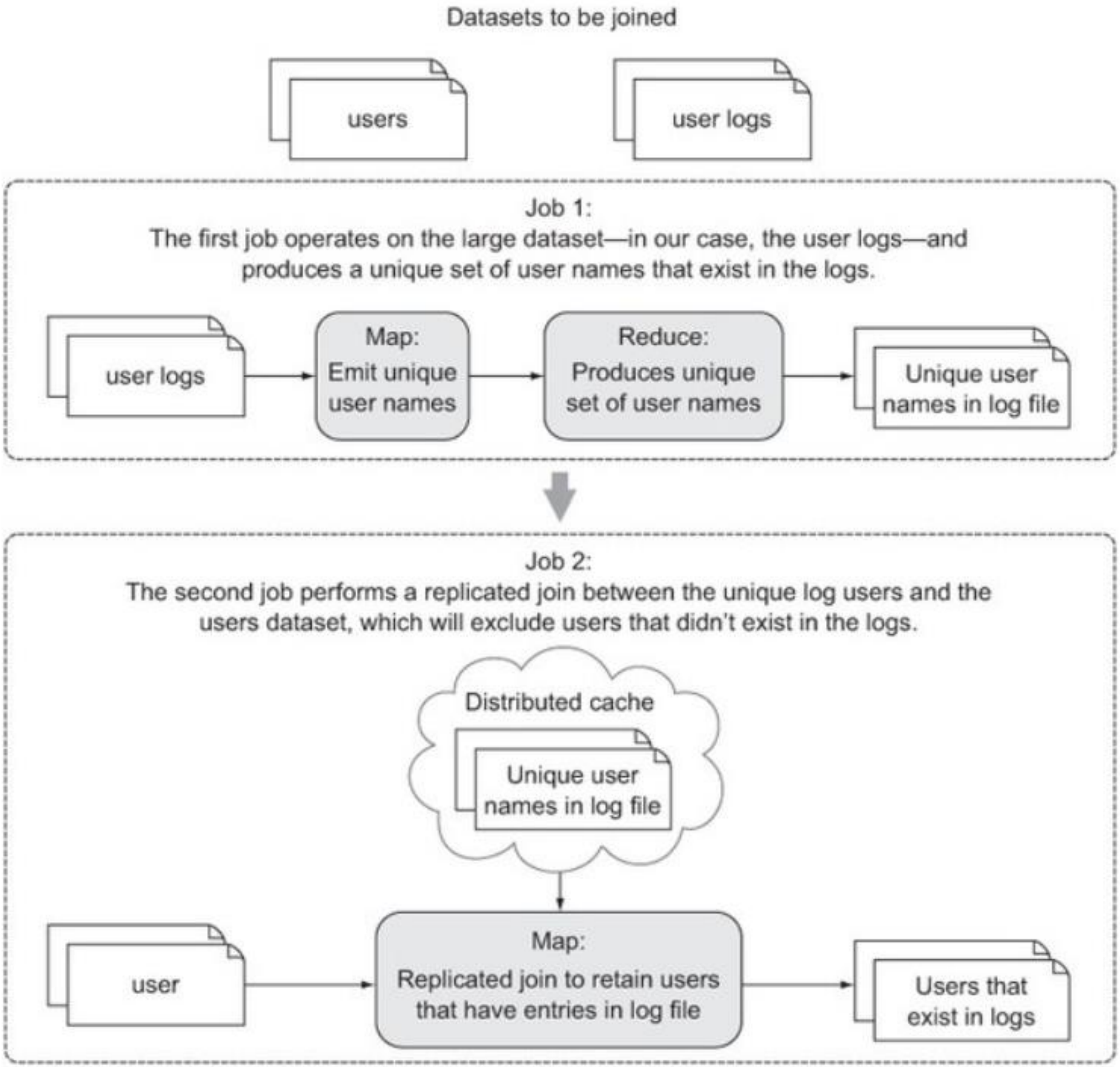
Extract the user filename from the job config.

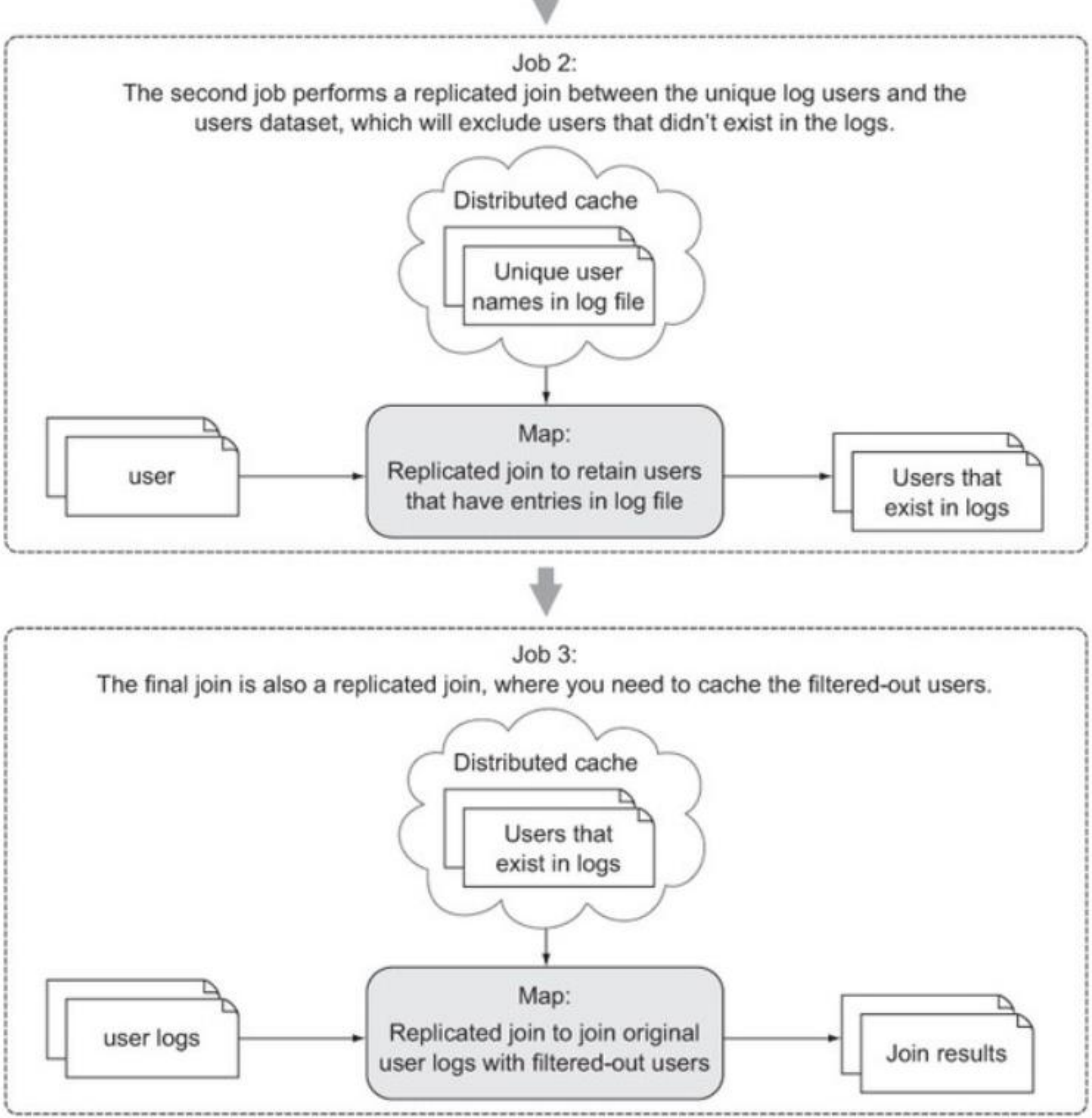
Loop through all the files in the distributed cache searching for your file.

When your file is found, load the users into memory.

If the user exists in both datasets, emit the combined records.

Semi-Join on large datasets

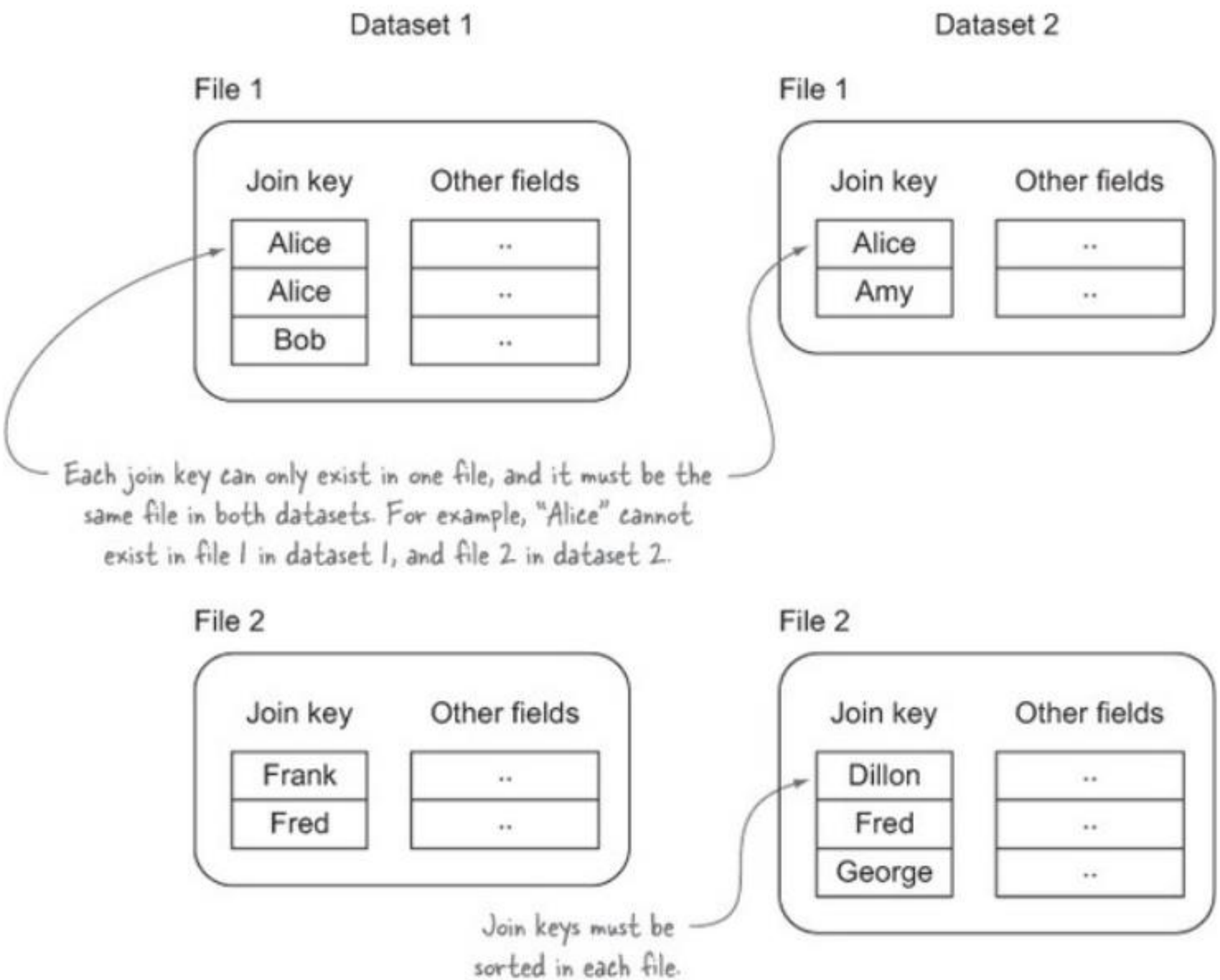



Job 2:
The second job performs a replicated join between the unique log users and the users dataset, which will exclude users that didn't exist in the logs.
Distributed cache
Unique user names in log file
user
Map:
Replicated join to retain users that have entries in log file
Users that exist in logs
Job 3:
The final join is also a replicated join, where you need to cache the filtered-out users.
Distributed cache
Users that exist in logs
user logs
Map:
Replicated join to join original user logs with filtered-out users
Join results

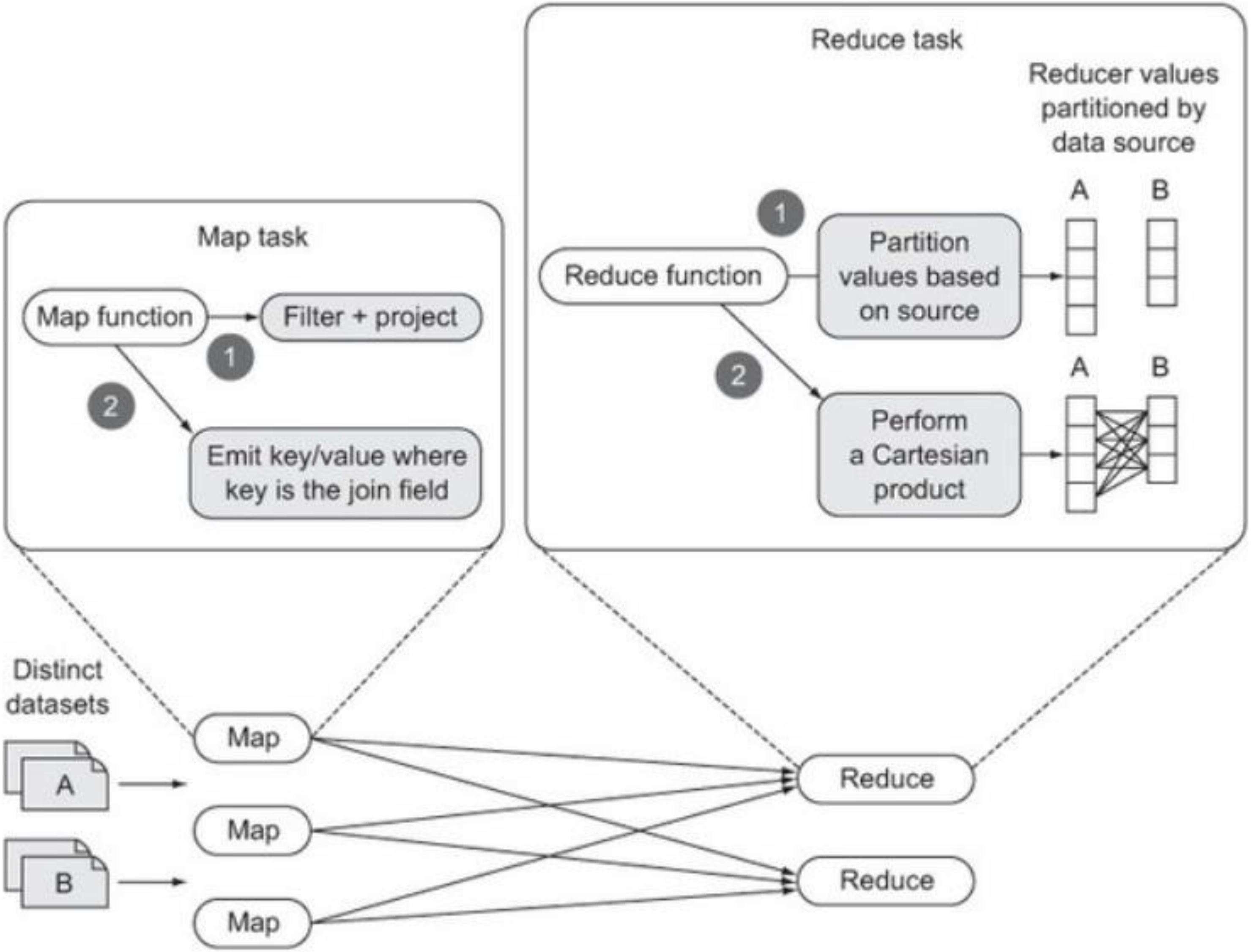
Joining on Presorted & Prepartitioned data

- None of the datasets can be loaded in memory in its entirety.
- The datasets are all sorted by the join key.
- Each dataset has the same number of files.
- File *N* in each dataset contains the same join key *K*.
- Each file is less than the size of an HDFS block, so that partitions aren't split. Or alternatively, the input split for the data doesn't split files.

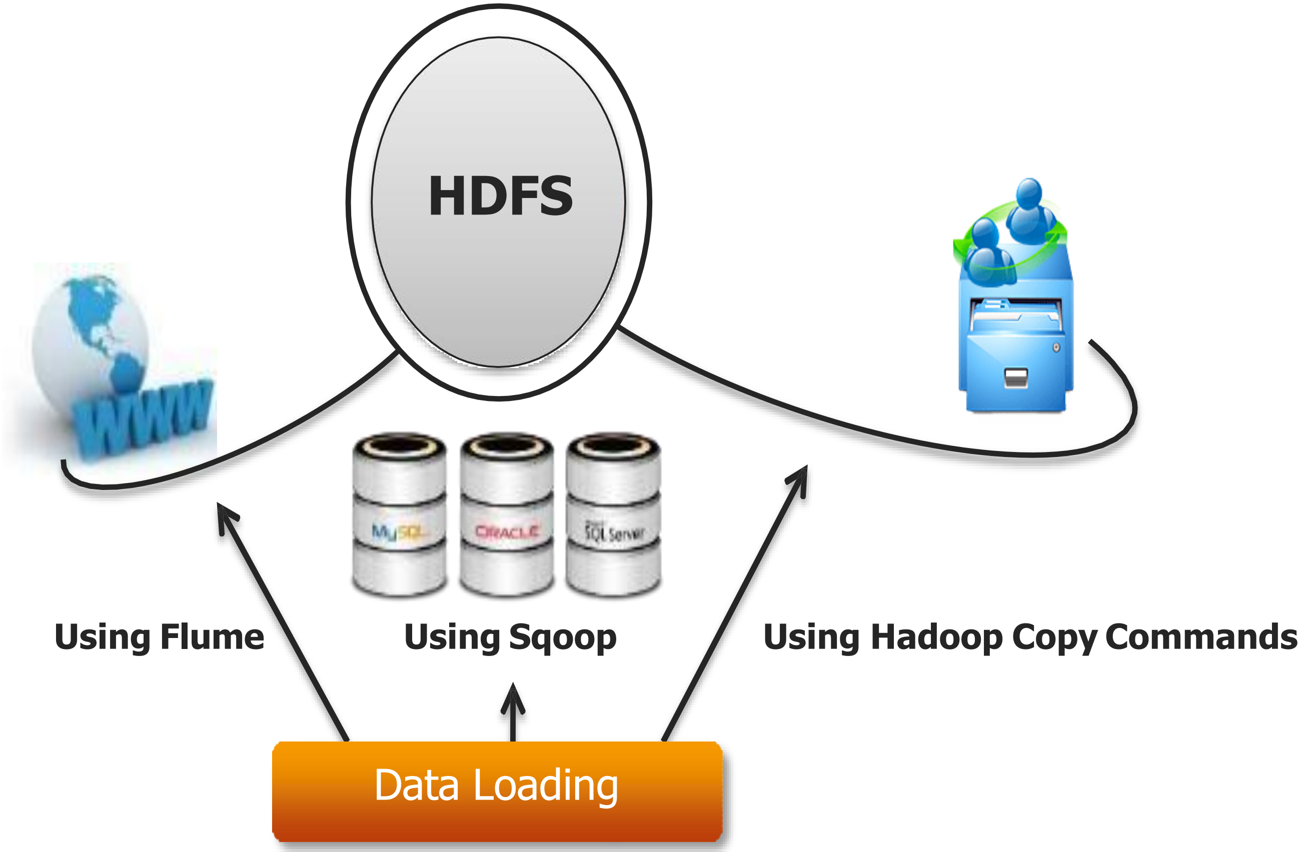
Joining on Presorted & Prepartitioned data Cont.



Reduce Side Join (Repartitioned Join)



Data Loading Techniques



Data Loading Using Sqoop

- Apache Sqoop (TM) is a tool designed for efficiently transferring bulk data between Apache Hadoop and
- structured data stores such as relational databases.

✓Imports individual tables or entire databases to HDFS.

✓Generates Java classes to allow you to interact with your imported data.

✓Provides the ability to import from SQL databases straight into your Hive data warehouse.

Question

Your website is hosting a group of more than 300 sub-websites. You want to have an analytics on the shopping patterns of different visitors? What is the best way to collect those information from the weblogs?

-SQOOP

-FLUME

Answer

FLUME.

Question

You want to join data collected from two sources. One source of data collected from a big database of call records is already available in HDFS. The another source of data is available in a database table.

The best way to move that data in HDFS is:

-SQOOP import

-PIG script

-Hive Query

Answer

Sqoop – How to run sqoop

Example:

```
sqoop import \
--connect jdbc:oracle:thin:@devdb11-s.cern.ch:10121/devdb11_s.cern.ch \
--username hadoop_tutorial \
-P \
--num-mappers 1 \
--target-dir visitcount_rfidlog \
--table VISITCOUNT.RFIDLOG
```

Sqoop – how to parallelize

```
-- table table_name

-- query select * from table_name where  CONDITIONS

-- table table_name
-- split-by primary key
-- num-mappers n

-- table table_name
-- split-by primary key
-- boundary-query select range from dual
-- num-mappers n
```

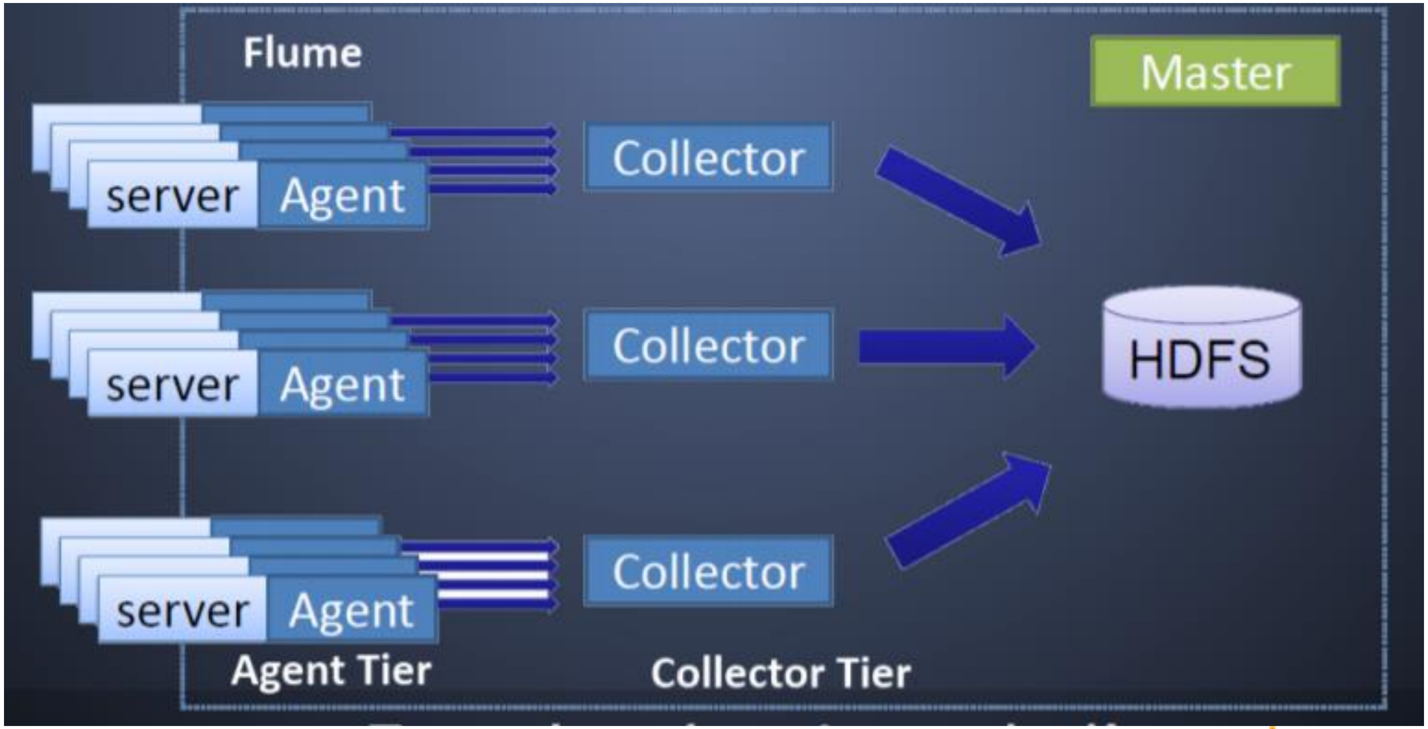
Reading Log data from File Systems

- **Situation**: You have hundreds of services running in different servers that produce lots of large logs which should be analyzed altogether. You have Hadoop to process them.

- **Problem**: How do I send all my logs to a place that has Hadoop? I need a reliable, scalable, extensible and manageable way to do it!

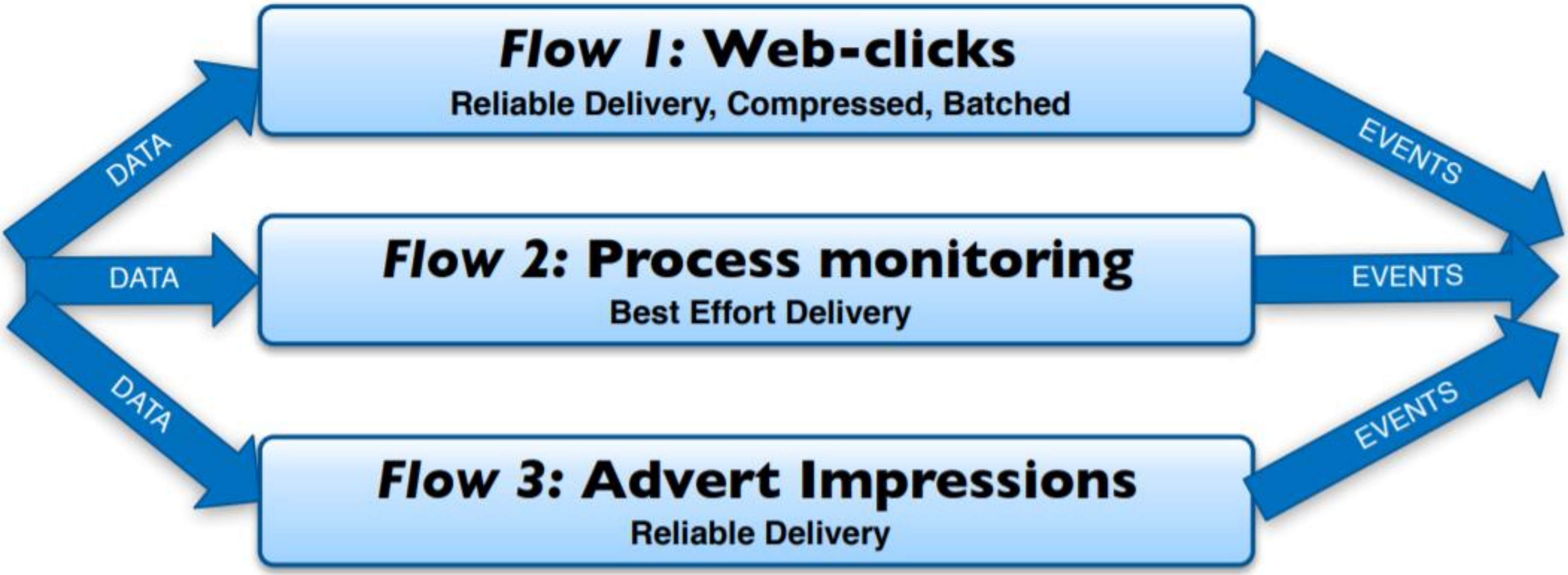
What is Apache Flume?

- It is a distributed data collection service that gets flows of data (like logs) from their source and aggregates them to where they have to be processed.
- Goals: reliability, scalability, extensibility, manageability.



Flume Flows

- Three typical flows, all on the same Flume service



Agents & Collectors

- Nodes that receive data from an application are called *agents*
- Flume supports many sources for agents, including:
  - Syslog
  - Tailing a file
  - Unix processes
  - Scribe API
  - Twitter
- Nodes that write data to permanent storage are called *collectors*
  - Most often they write to HDFS

HIVE

Motivation

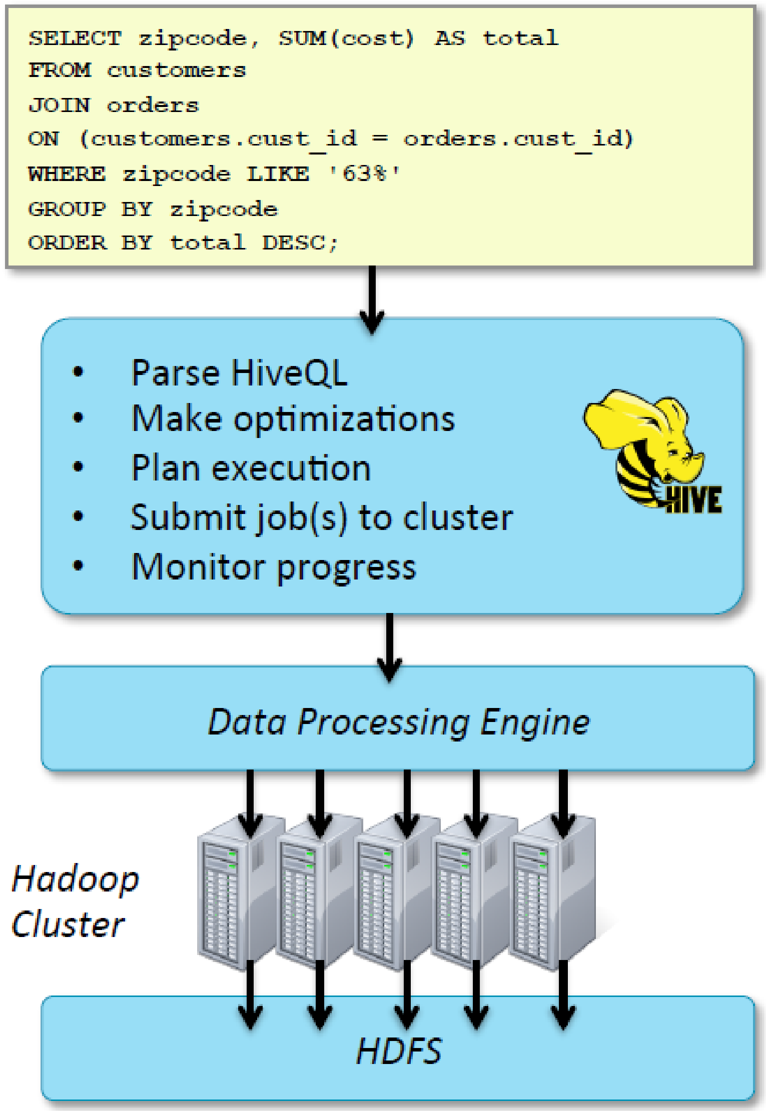
- Limitation of MR
  - Have to use M/R model
  - Not Reusable
  - Error prone
  - For complex jobs:
    - Multiple stage of Map/Reduce functions
    - Just like ask dev to write specify physical execution plan in the database

Overview

- Intuitive
  - Make the unstructured data looks like tables regardless how it really lay out
  - SQL based query can be directly against these tables
  - Generate specify execution plan for this query
- What's Hive
  - A data warehousing system to store structured data on Hadoop file system
  - Provide an easy query these data by execution Hadoop MapReduce plans

Hive"High/Level"Overview

- **Hive**
  - Turns HiveQL queries into data processing jobs
  - Submits those jobs to the data processing engine (MapReduce) to execute on the cluster



Why use Hive?

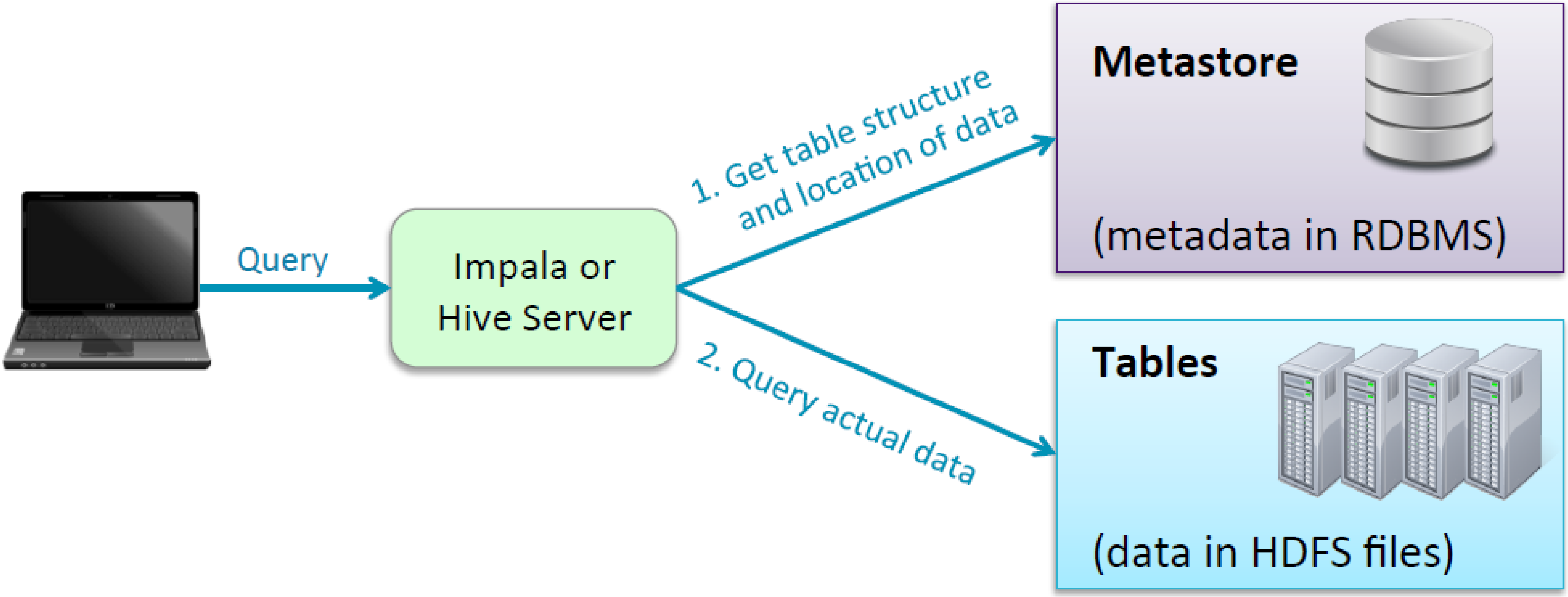
- **More productive than writing MapReduce directly**
  - Five lines of HiveQL/Impala SQL might be equivalent to 200 lines or more of Java
- **Brings large-scale data analysis to a broader audience**
  - No software development experience required
  - Leverage existing knowledge of SQL
- **Offers interoperability with other systems**
  - Extensible through Java and external scripts
  - Many business intelligence (BI) tools support Hive and/or Impala

How Hive and Impala Load and Store Data (1)

- **Queries operate on tables, just like in an RDBMS**
  - A table is simply an HDFS directory containing one or more files
  - Default path: `/user/hive/warehouse/`*`<table_name>`*
  - Supports many formats for data storage and retrieval
- **What is the structure and location of tables?**
  - These are specified when tables are created
  - This metadata is stored in the *Metastore*
    - Contained in an RDBMS such as MySQL
- **Hive and Impala work with the same data**
  - Tables in HDFS, metadata in the Metastore

How Hive and Impala Load and Store Data (2)

- **Hive and Impala use the Metastore to determine data format and location**
  - The query itself operates on data stored on a filesystem (typically HDFS)



Your Cluster is Not a Database Server

- **Client-server database management systems have many strengths**
  - Very fast response time
  - Support for transactions
  - Allow modification of existing records
  - Can serve thousands of simultaneous clients
- **Your Hadoop cluster is not an RDBMS**
  - Hive generates processing engine jobs (MapReduce) from HiveQL queries
    - Limitations of HDFS and MapReduce still apply
  - Impala is faster but not intended for the throughput speed required for an OLTP database
  - No transaction support

Comparison with RDBMS

	Relational Database	Hive
Query language	SQL (full)	SQL (subset)
Update individual records	Yes	Yes
Delete individual records	Yes	Yes
Transactions	Yes	No
Index support	Extensive	Limited
Latency	Very low	High
Data size	Terabytes	Petabytes

ACID support in Hive started from 0.14 version

Hive Query Language

Syntax Basics

- **Keywords are not case-sensitive**
  - Though they are often capitalized by convention
- **Statements are terminated by a semicolon**
  - A statement may span multiple lines
- **Comments begin with -- *(double hyphen)***
  - Only supported in scripts
  - There are no multi-line comments

myscript.sql

```
SELECT cust_id, fname, lname
   FROM customers
   WHERE zipcode='60601';     -- downtown Chicago
```


tusind tak
ngiyabonga
dziękuję
谢谢
dakujem vám
ありがとう
merci
suksema
danke
thank
baie dankie
धन्यवाद
molte grazie
gracias
takk
you
dank u
teşekkür edire
mahalo
obrigada
obrigado
gràcies
teşekkür ederim
شكرا
tänan
tack så mycket