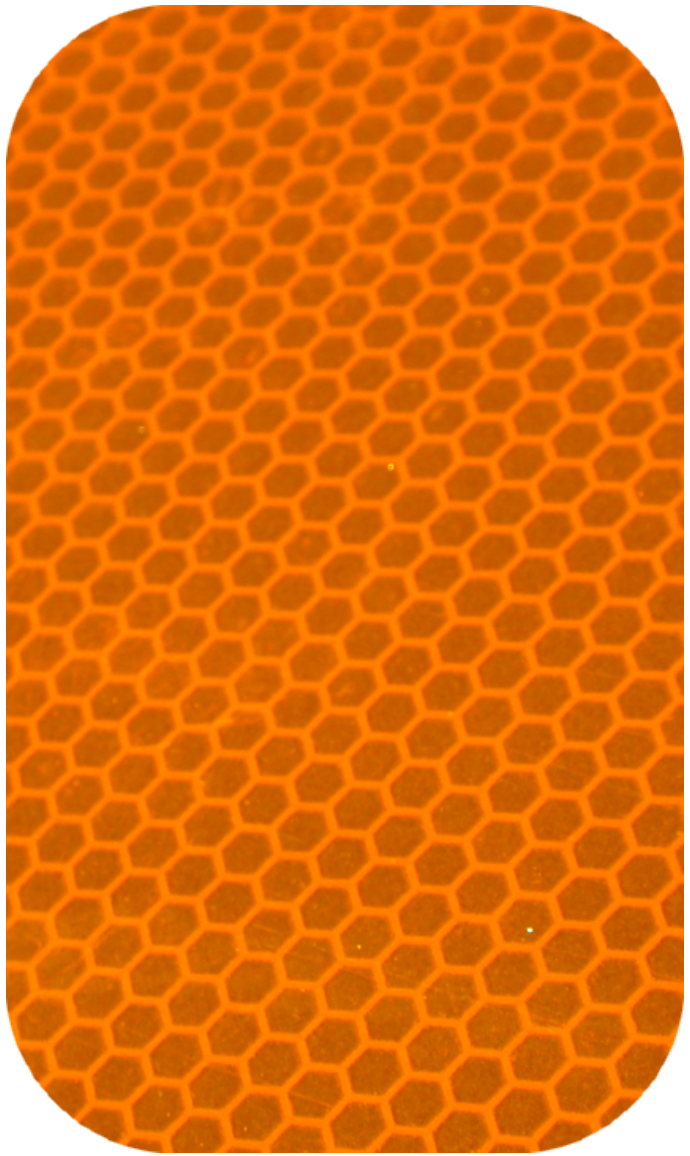
Hive

Agenda

- What is Hive?
  - Java vs Hive
  - Components
  - Configuration
- Hive Queries
  - Tables
  - Explain
  - Join
- Hive and HBase
  - HBase Table Mapping
  - Multiple columns to MAP



SQL for Hadoop

- **Data warehouse augmentation is a very common use case for Hadoop**
- **While highly scalable, MapReduce is notoriously difficult to use**
  - Java API is tedious and requires programming expertise
  - Unfamiliar languages (e.g. Pig) also requiring expertise
  - Many different file formats, storage mechanisms, configuration options, etc.
- **SQL support opens the data to a much wider audience**
  - Familiar, widely known syntax
  - Common catalog for identifying data and structure
  - Clear separation of defining the *what* (you want) vs. the *how* (to get it)

What is Hive?

- **A system for managing and querying structured data built on top of Hadoop**
  - Map-Reduce for execution
  - HDFS for storage
  - Metadata on raw files

- **Key Building Principles:**
  - SQL is a familiar data warehousing language
  - Extensibility – Types, Functions, Formats, Scripts
  - Scalability and Performance

Java versus Hive: The Word Count Algorithm

```
package org.myorg;

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

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

public class WordCount {

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

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

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

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

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

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

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

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

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

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

    job.waitForCompletion(true);
  }
}
```

Hive and Word Count!

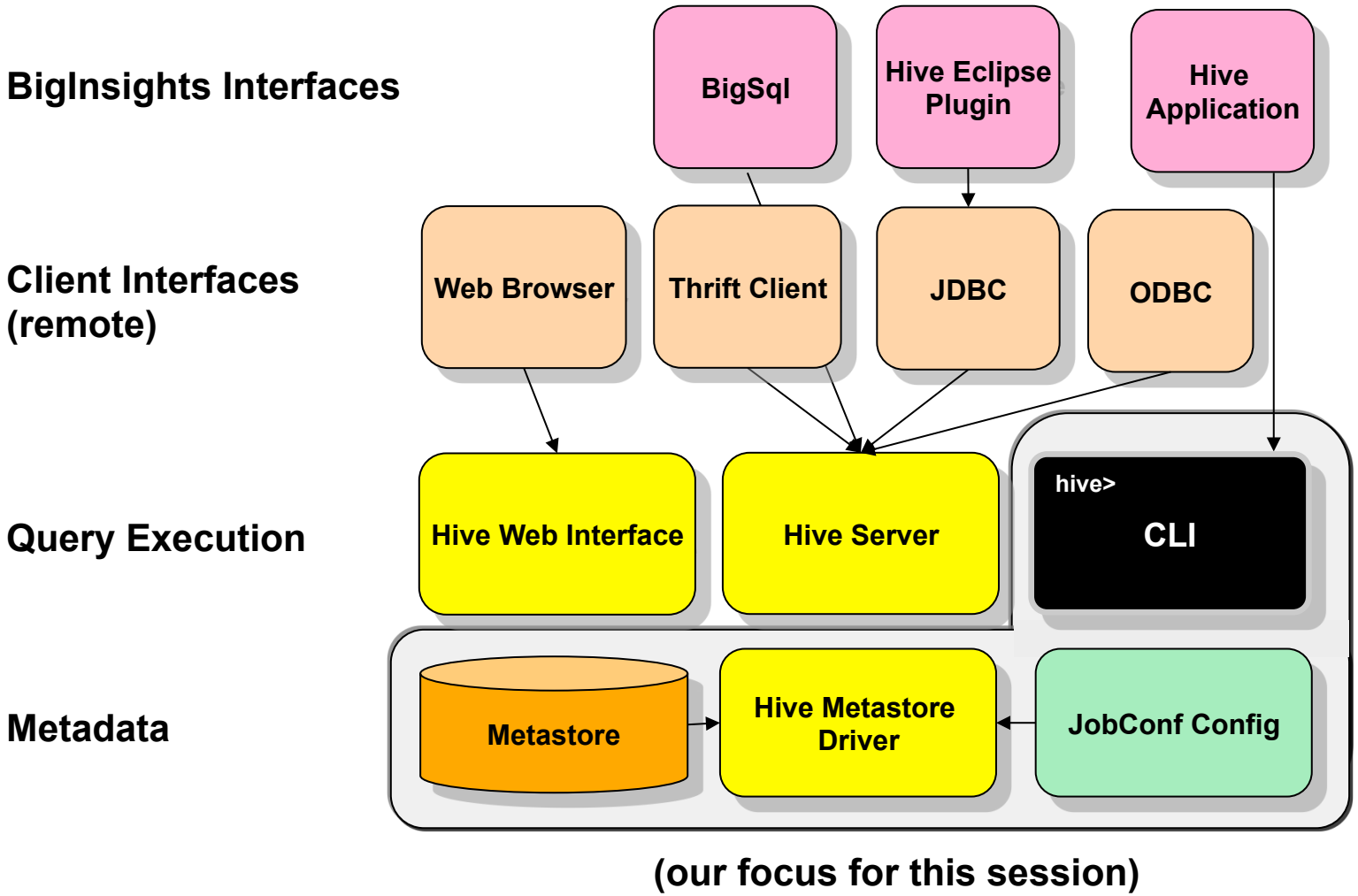
```
CREATE TABLE docs (line STRING);

LOAD DATA INPATH 'docs' OVERWRITE INTO TABLE docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
          (SELECT explode(split(line, '\s')) AS word FROM docs)
GROUP BY word
ORDER BY word;
```

*See “Programming Hive”, *Capriolo,Wampler & Rutherglen*

Hive - Components



(our focus for this session)

Starting Hive – The Hive Shell

- **The Hive shell is located in**
  $HIVE_HOME/bin/hive
- **From the shell you can**
  - Perform queries, DML, and DDL
  - View and manipulate table metadata
  - Retrieve query explain plans (execution strategy)

```
$ $HIVE_HOME/bin/hive
2013-01-14 23:36:52.153 GMT : Connection obtained for host: master-
Logging initialized using configuration in file:/opt/ibm/biginsight
Hive history file=/var/ibm/biginsights/hive/query/biadmin/hive_job

hive> show tables;
mytab1
mytab2
mytab3
OK
Time taken: 2.987 seconds
hive> quit;
```

Data Types and Models

- **Supports a number of scalar and structured data types**
  - tinyint, smallint, int, bigint, float, double
  - boolean
  - string, binary
  - timestamp
  - array – e.g. `array<int>`
  - struct – e.g. `struct<f1:int,f2:array<string>>`
  - map – e.g. `map<int,string>`
  - union – e.g. `uniontype<int,string,double>`

- **Partitioning**
  - Can partition on one or more columns
  - Value partitioning only, range partitioning is not yet supported

- **Bucketing**
  - Sub-partitioning/grouping of data by hash within partitions
  - Useful for sampling and improves some join operations

Data Model - Partition

- **Value partition based on partition columns**
- **Nested sub-directories in HDFS for each combination of partition column values**
- **Example**
  - Partition columns : ds, ctry
  - HDFS subdirectory for ds = 20090801, ctry = US
    - …/hive/warehouse/pview/ds=20090801/ctry=US
  - HDFS subdirectory for ds = 20090801, ctry = CA
    - …/hive/warehouse/pview/ds=20090801/ctry=CA

Data Model - Bucket

- **Split data based on hash of a column – mainly for parallelism**
- **One HDFS file per bucket within partition sub-directory**
- **Example**
  - Bucket column : user into 32 buckets
  - HDFS file for user hash 0
    - …/hive/warehouse/pview/ds=20090801/ctry=US/part-00000
  - HDFS file for user hash bucket 20
    - …/hive/warehouse/pview/ds=20090801/ctry=CA/part-00020

Data Model – External Table

- **Point to existing data directories in HDFS**
- **Can create tables and partitions – partition columns just become annotations to external directories**
-

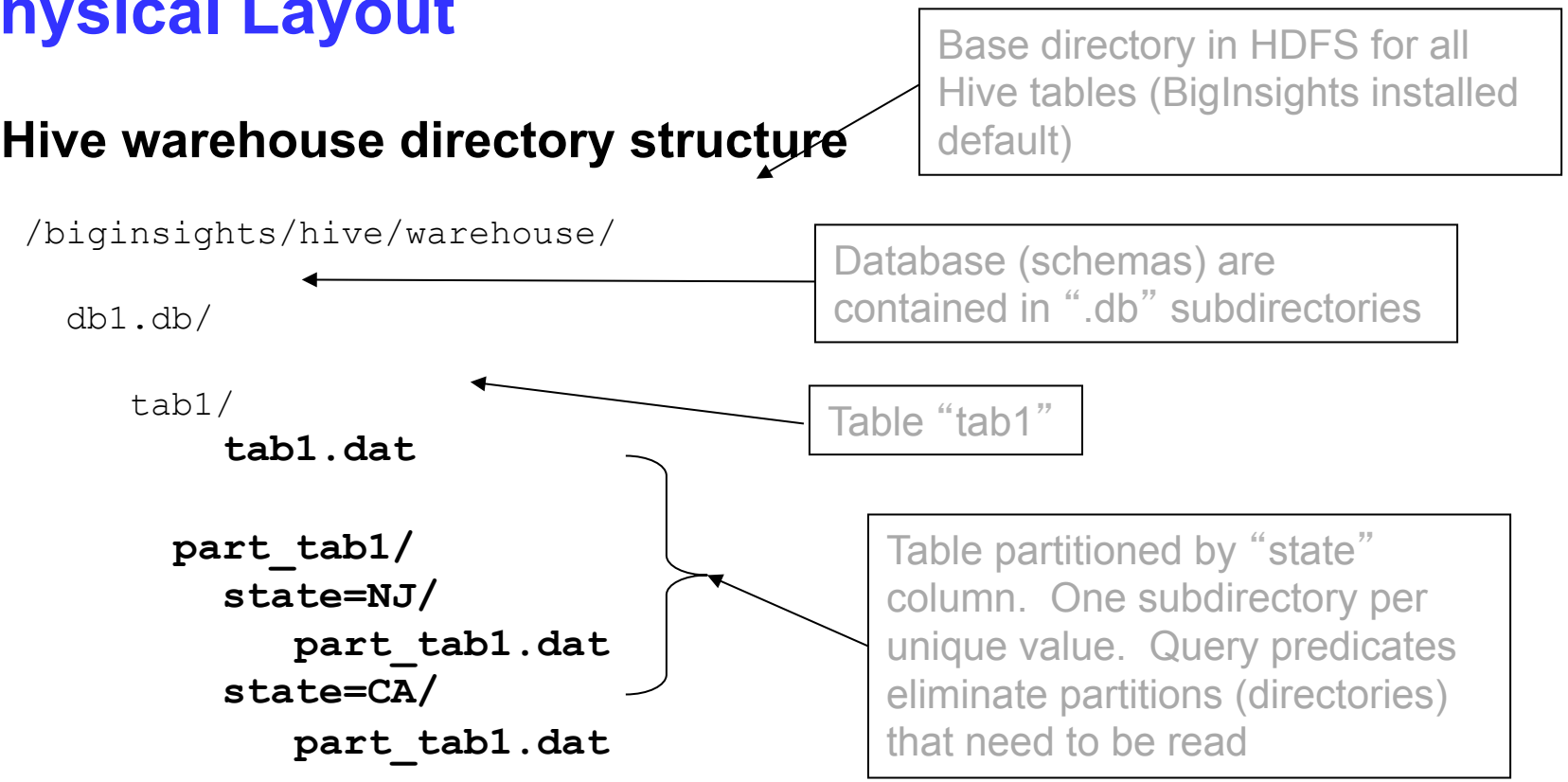
```
CREATE EXTERNAL TABLE pview (userid int, pageid int, ds string, ctry string)
PARTITIONED ON (ds string, ctry string)
STORED AS textfile
LOCATION ‘/path/to/existing/table’
```

```
ALTER TABLE pview
ADD PARTITION (ds= ‘20090801’ , ctry= ‘US’ )
LOCATION ‘/path/to/existing/partition’
```

- **Example : add a partition to external table**

Physical Layout

- **Hive warehouse directory structure**



- **Data files are just regular HDFS files**
  - Internal format can vary table-to-table (delimited, sequence, etc.)

- **Supports "external" tables**

Creating a table

- **Creating a delimited table**

```
hive> create table users
(
  id         int,
  office_id  int,
  name       string,
  children   array<string>
)
row format delimited
   fields terminated by '|'
   collection items terminated by ':'
stored as textfile;
```

file: users.dat

```
1|1|Bob Smith|Mary
2|1|Frank Barney|James:Liz:Karen
3|2|Ellen Lacy|Randy:Martin
4|3|Jake Gray|
5|4|Sally Fields|John:Fred:Sue:Hank:Robert
```

- **Inspecting tables:**

```
hive> show tables;
OK
users
Time taken: 2.542 seconds

hive> describe users;
OK
id          int
office_id   int
name        string
children    array<string>
Time taken: 0.129 seconds
```

Table population and querying

- **Loading data from input file**

```
hive> load data local inpath 'users.dat' into table users;
Copying data from file:/home/biadmin/hive_demo/users.dat
Copying file: file:/home/biadmin/hive_demo/users.dat
Loading data to table default.users
OK
Time taken: 0.276 seconds
```

- **The "local" indicates the source data is on the local (unix) filesystem**
  - Otherwise file is assumed to be on HDFS

- **Our first query:**

```
hive> select * from users;
1       1       Bob Smith       ["Mary"]
2       1       Frank Barney    ["James","Liz","Karen"]
3       2       Ellen Lacy      ["Randy","Martin"]
4       3       Jake Gray       []
5       4       Sally Fields    ["John","Fred","Sue","Hank",…
```

Tables derived from queries

- **Tables may be created using queries on other tables**

```
create table emps_by_state
as
select o.state as state, count(*) as employees
  from office o left outer join users  u
     on u.office_id = o.office_id
group by o.state;
```

- **or…**

```
create table emps_by_state ... stored as textfile;

insert overwrite table emps_by_state
select o.state as state, count(*) as employees
  from office o left outer join users  u
     on u.office_id = o.office_id
group by o.state;
```

Storage file formats

- **The `STORED AS` clause indicates the storage file/record format on HDFS**
  - `TEXTFILE` – Stored as a text line (one line per record)
  - `SEQUENCEFILE` – Stored as a Hadoop sequence file
  - `RCFILE` – Semi-columnar data storage with good compression. Best performance of the built-in storage formats
  - `INPUTFORMAT`/`OUTPUTFORMAT` – Can provide a specific Hadoop input or output format class

```
create table foo ( … )
row format delimited fields terminated by ‘,’
storage format sequencefile;
```

Record Format

- **Values are written/read into file using a Hive SerDe**
  - Serializer/Deserializer class – encodes and decodes values
- **Default SerDe is Hive's LazySimpleSerDe**
  - Delimited format
  - Delimiters specified with `DELIMITED` clause
  - "Lazy" indicates values are only read if accessed in the query
  - Re-uses objects across rows
  - `TIMESTAMP` must be formatted as yyyy-mm-dd hh:mm:ss.ffffff
  - Null value can be provided as `\N`
- **SerDe defined with the `STORED BY` clause**

```
create table foo ( … )
storage format sequencefile
stored by 'org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe'
```

Record Format (cont.)

- **Other built-in SerDe implementations**
  - Thrift (ThriftSerDe) – Thrift binary encoding
  - Regular Expressions (RegexSerDe) – Use regex to extract fields
  - Hive Binary format (LazyBinarySerDe) – Binary storage
- **You can implement your own SerDe**
- **Others available freely on the internet**
  - JSON
  - Avro
  - Etc.

Explain

- **The explain keyword generates a Hive explain plan for the query**

```
hive> explain select o.state as state, count(*) as employees
  from office o left outer join users  u
     on u.office_id = o.office_id
group by o.state
order by state;

STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-3 depends on stages: Stage-2
  Stage-0 is a root stage
...
```

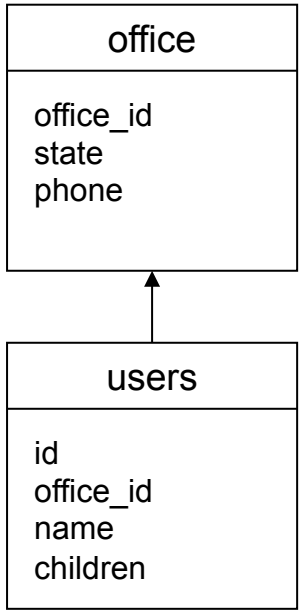
- **Query requires three MapReduce jobs to implement**
  - One to join the two inputs
  - One to perform the GROUP BY
  - One to perform the final sort

Joins

- **Hive supports joins via ANSI join syntax only**

```
select ...
  from T1, T2
 where T1.a = T2.b
```

```
select ...
  from T1 join T2
       on T1.a = T2.b
```



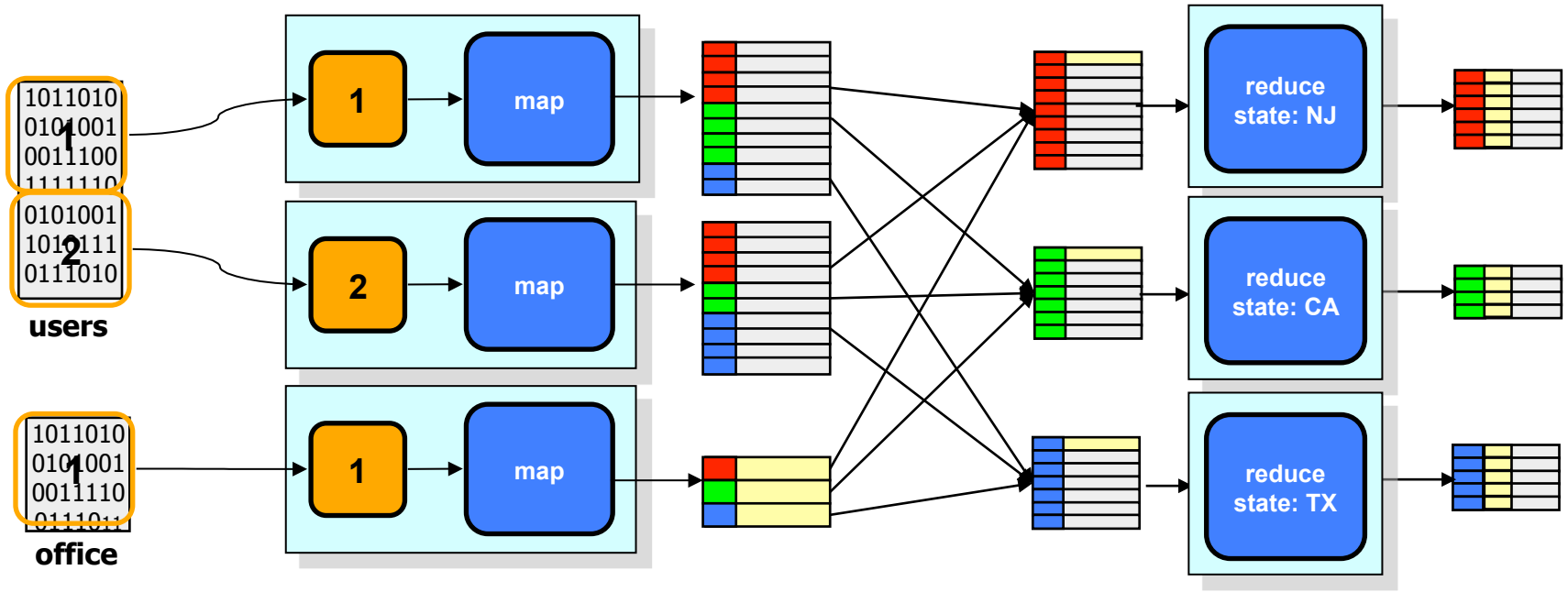
- **Example join**

```
hive> select o.state as state, count(*) as employees
  from office o left outer join users  u
     on u.office_id = o.office_id
group by o.state
order by state;
```

- **This takes 67 seconds (**On my (crummy) demo environment (!**)**
  - Only five offices and five users (< 1KB of data)
  - What's going on?

Understanding distributed joins

- **Not entirely necessary, but very useful to understand:**



- **Hive only supports equi-joins (t1.c1 = t2.c2)**
  - E.g. it cannot perform non-equi-joins - t1.c1 > t2.c2
  - From the above diagram, do you understand why?

User Defined Function

- **Java Code**

```
package com.example.hive.udf;

import org.apache.hadoop.hive.ql.exec.UDF;
import org.apache.hadoop.io.Text;

public final class Lower extends UDF {
  public Text evaluate(final Text s) {
    if (s == null) { return null; }
    return new Text(s.toString().toLowerCase());
  }
}
```

- **Registering the Class**
  - **CREATE FUNCTION** my_lower **AS** 'com.example.hive.udf.Lower';

- **Using the Function**
  - **SELECT** my_lower(title), sum(freq) **FROM** titles **GROUP BY** my_lower(title);

Hive and HBase

- **Hive comes with an HBase *storage handler***
- **Allows MapReduce queries and loading of HBase tables**
- **Uses predicate pushdown to optimize query**
  - Scans only necessary regions based upon table key
  - Applies predicates as HBase row filters (if possible)
- **Usually Hive must be provided additional jars and configuration in order to work with HBase**

```
$ hive \
    --auxpath \
       $HIVE_SRC/build/dist/lib/hive-hbase-handler-0.9.0.jar,\
       $HIVE_SRC/build/dist/lib/hbase-0.92.0.jar,\
       $HIVE_SRC/build/dist/lib/zookeeper-3.3.4.jar,\
       $HIVE_SRC/build/dist/lib/guava-r09.jar \
    -hiveconf hbase.master=hbase.yoyodyne.com:60000
```

- **BigInsights' Hive is preconfigured for HBase integration, so this is unecessary**

HBase table mapping

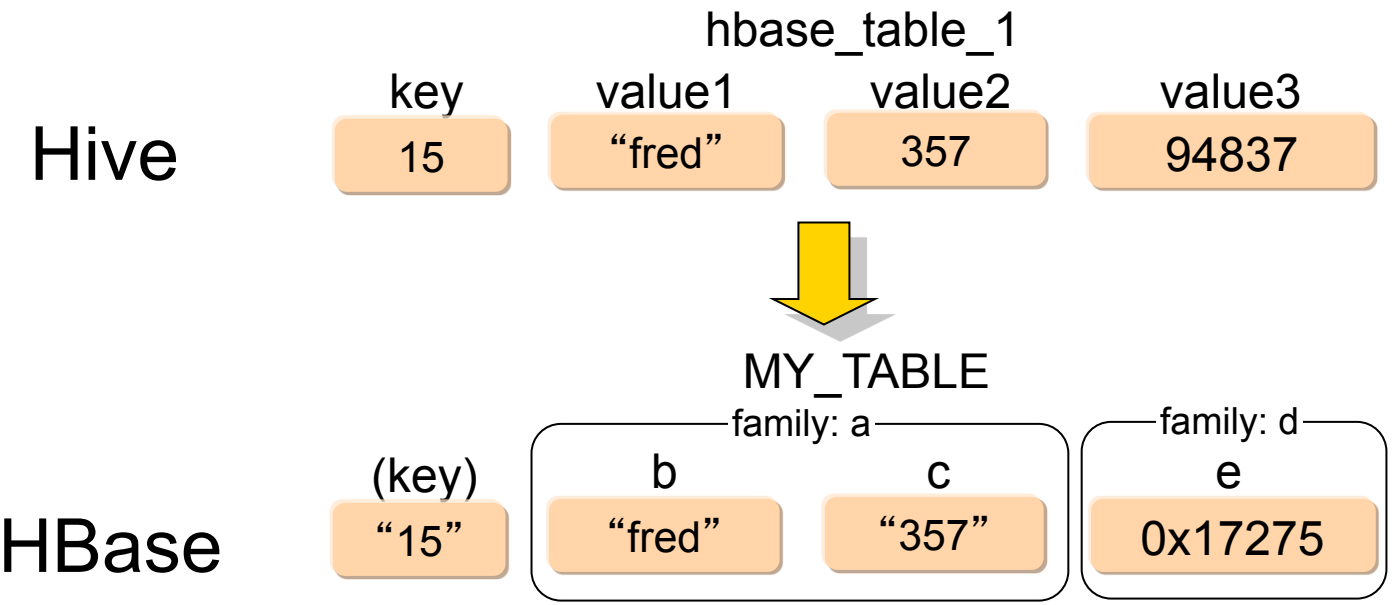
- **Creating an HBase table**

```
CREATE TABLE hbase_table_1 (
   key    int,
   value1 string,
   value2 int,
   value3 int)
STORED BY 'org.apache.hadoop.hive.hbase.HBaseStorageHandler‘
WITH SERDEPROPERTIES (
   "hbase.columns.mapping" = ":key,a:b,a:c,d:e“
)
TBLPROPERTIES(
   "hbase.table.name" = “MY_TABLE“
); ;
```

- **The `hbase.table.name` property provides the table name in HBase**
  - Optional. If not provided, the Hive name is assume

HBase table mapping

```
CREATE TABLE hbase_table_1 (
   key    int,
   value1 string,
   value2 int,
   value3 int)
…
WITH SERDEPROPERTIES ("hbase.columns.mapping"= ":key,a:b,a:c,d:e“)
TBLPROPERTIES("hbase.table.name" = “MY_TABLE“);
```



Configuring Hive

- **Very little configuration is necessary to get started with Hive**
- **Minimum configuration identifies where to find the metastore**
  - If no configuration is provided a local Derby database is used
  - BigInsights installs pre-configured to use it's Derby server
  - Can be configured to use a wide variety of storage options (DB2, MySQL, Oracle, XML files, etc.)

- **Configuration file is located in:**

  `$HIVE_HOME``/conf/hive-site.xml`

- **Configuration Example :**

```
<property>
        <name>hive.exec.scratchdir</name>
        <value>/tmp/mydir</value>
        <description>Scratch space for Hive jobs</description>
    </property>
```

Configuration Option Settings

Variable Name	Description
hive.ddl.output.format	The data format to use for DDL output (e.g. DESCRIBE table). One of "text" (for human readable text) or "json" (for a json object).
hive.exec.script.wrapper	Wrapper around any invocations to script operator e.g. if this is set to python, the script passed to the script operator will be invoked as python <script command>. If the value is null or not set, the script is invoked as <script command>.
hive.exec.scratchdir	This directory is used by hive to store the plans for different map/ reduce stages for the query as well as to stored the intermediate outputs of these stages.
hive.exec.submitviachild	Determines whether the map/reduce jobs should be submitted through a separate jvm in the non local mode.
hive.exec.script.maxerrsize	Maximum number of serialization errors allowed in a user script invoked through TRANSFORM or MAP or REDUCE constructs.
hive.exec.compress.output	Determines whether the output of the final map/reduce job in a query is compressed or not.
hive.default.fileformat	Default file format for CREATE TABLE statement. Options are TextFile, SequenceFile, RCFile, and Orc.

Conclusion

- **There are plenty of more interesting topics and features**
  - Settings to control hive behavior
  - Hints to control query execution strategy
  - Built-in functions and user defined functions (and aggregates)
  - Views
  - Performance tuning
  - Etc.
- **Limitations**
  - Many limitations have been addressed in Hive 0.11 and 0.12
    - Added data type support, added window functions, better JDBC drivers, better Hive server, …
  - Still serious limitations
    - No subqueries
    - Limited HBase support
    - …

Questions?