THE UNIVERSITY of CHICAGO
GRAHAM SCHOOL
of CONTINUING LIBERAL AND PROFESSIONAL STUDIES

BIG DATA SESSION 3

NICK KADOCHNIKOV

AGENDA

- Assignment 1 due: selecting Big Data infrastructure for Smart Meters
  - Hadoop Shell Commands
  - Hue
  - Pig
  - Hive
  - Hbase
  - Zookeeper
  - Sqoop
- Assignment 2: load data into Hadoop and create summary statistics using Hive

HADOOP FRAMEWORK

Apache Oozie
Workflow Scheduler for Hadoop

Apache Chukwa
HDFS Monitoring

Apache Zookeper
HDFS Management

Apache Mahout
Machine learning algorithms and data mining within HDFS framework

Apache HBase
Column oriented database to read/write into HDFS

Apache Flume
Tools to move streaming data to Hadoop Cluster

Apache Pig
Scripting language for data analysis

Apache Hive
SQL like language to query data from HDFS

Apache Sqoop
Scripting language for data manipulation

MapReduce Framework
Programming model for distributed data processing

Hadoop Distributed File System (HDFS)
A Distributed file system designed to run on commodity hardware

HADOOP FRAMEWORK COMPONENTS

- **Pig** – a platform for manipulating data stored in HDFS. It consists of a compiler for MapReduce programs and a high-level language called Pig Latin. It provides a way to perform data extractions, transformations and loading, and basic analysis without having to write MapReduce programs.
- **Hive** – a data warehousing and SQL-like query language that presents data in the form of tables. Hive programming is similar to database programming. (It was initially developed by Facebook.)
- **HBase** – a nonrelational, distributed database that runs on top of Hadoop. HBase tables can serve as input and output for MapReduce jobs.
- **Zookeeper** – an application that coordinates distributed processes.
- **Ambari** – a web interface for managing, configuring and testing Hadoop services and components.
- **Flume** – software that collects, aggregates and moves large amounts of streaming data into HDFS.
- **Sqoop** – a connection and transfer mechanism that moves data between Hadoop and relational databases.
- **Oozie** – a Hadoop job scheduler.

Hadoop Components
(so you are ready for this job interview…)

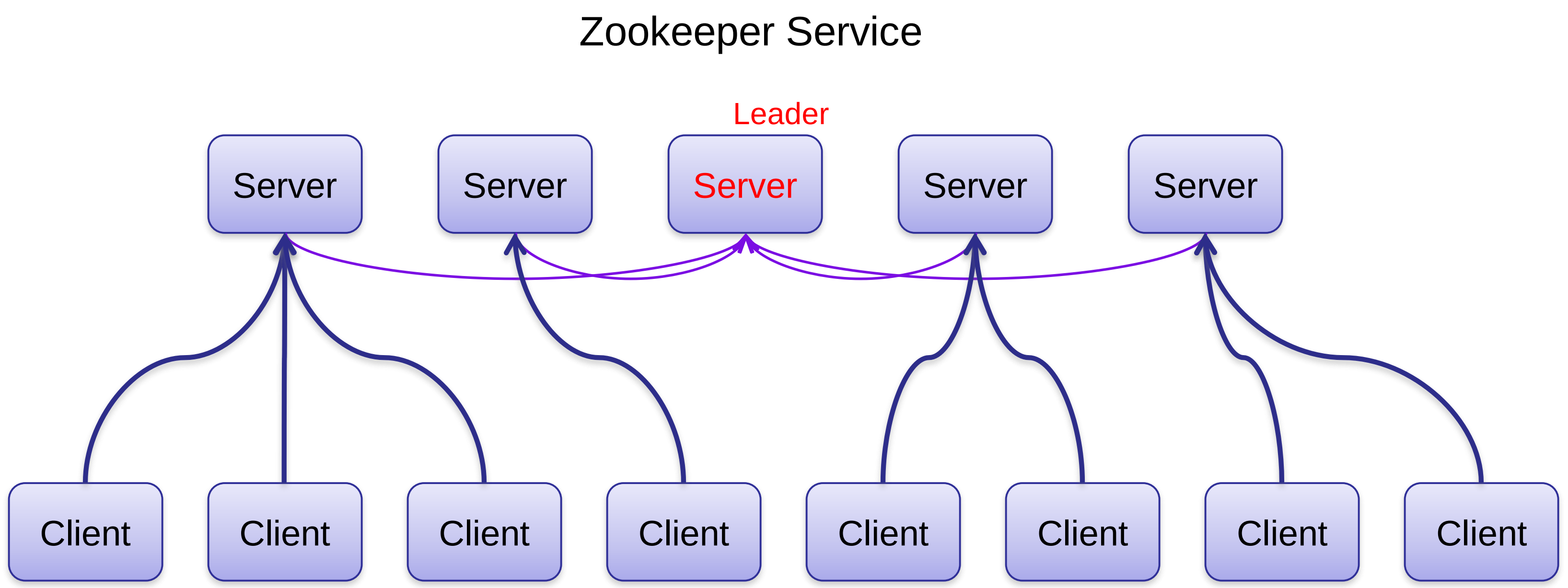
ZOOKEEPER

- Zookeeper is a stripped down filesystem that exposes a few simple operations and abstractions that enable you to build distributed queues, configuration service, distributed locks, and leader election among a group of peers.
    - Configuration Service – store and allows applications to retrieve or update configuration files
    - Distributed Lock – is a mechanism for providing mutual exclusion between a collection of processes. At any one time, only a single process may hold the lock. They can be utilized for leader election, where the leader is the process the holds the lock at any point of time.
- Zookeeper is highly available running across a collection of machines.

How ZooKeeper works

- Provides centralized management of the entire cluster in terms of name services, group services, synchronization services, configuration management, and more.
- ZooKeeper provides an infrastructure for cross-node synchronization and can be used by applications to ensure that tasks across the cluster are serialized or synchronized.
- It does this by maintaining status type information in memory on ZooKeeper servers. A ZooKeeper server is a machine that keeps a copy of the state of the entire system and persists this information in local log files.
- A very large Hadoop cluster can be supported by multiple ZooKeeper servers (in this case, a master server synchronizes the top-level servers).
- Each client machine communicates with one of the ZooKeeper servers to retrieve and update its synchronization information. distributed set of servers.

ZOOKEEPER SERVICE

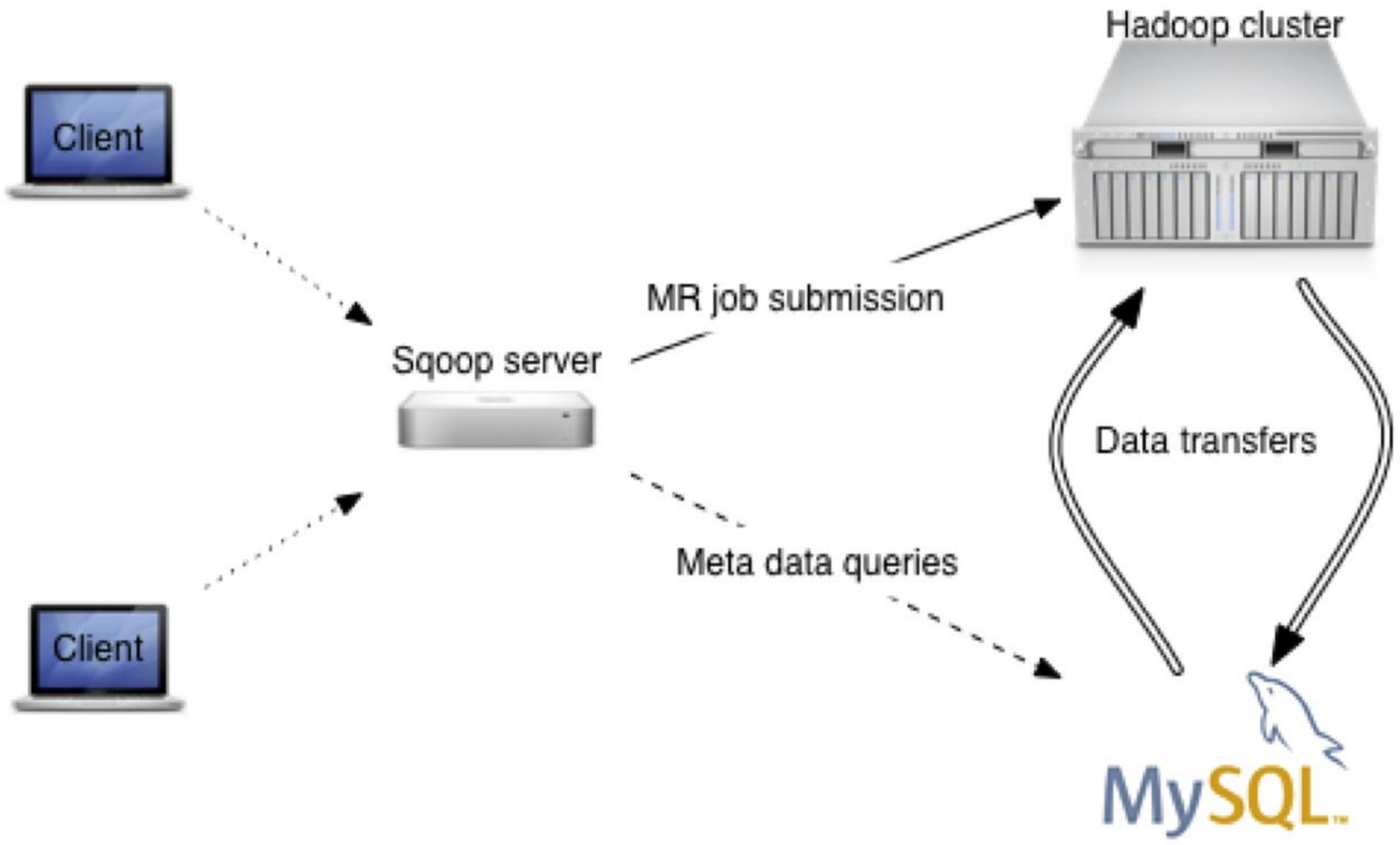


- Data is stored in-memory in all the servers
- A leader is elected at start-up
- Followers service clients, all updates go through leader
- Update responses are sent when a majority of servers have persisted the change

What is Sqoop?

- Apache Top-Level Project
- SQl and hadOOP
- Transfer a large bulk of data
  - **From** relational data warehouses: Teradata, MySQL, PostgreSQL, Oracle, Netezza
  - **To** Hadoop ecosystem: HDFS, Hive, HBase, Avio
  - Vice versa

SQOOP HELPS TRANSFER DATA BETWEEN HDFS AND THE TRADITIONAL RDBMS.



Sqoop Objectives

- While it is sometimes necessary to move the data in real time, it is most often necessary to load or unload data in bulk. Like Pig, Sqoop is a command-line interpreter. You type Sqoop commands into the interpreter and they are executed one at a time. Four key features are found in Sqoop:
- **Bulk import:** Sqoop can import individual tables or entire databases into HDFS. The data is stored in the native directories and files in the HDFS file system.
- **Direct input:** Sqoop can import and map SQL (relational) databases directly into Hive and HBase.
- **Data interaction:** Sqoop can generate Java classes so that you can interact with the data programmatically.
- **Data export:** Sqoop can export data directly from HDFS into a relational database using a target table definition based on the specifics of the target database.

SQOOP REAL WORD EXAMPLES

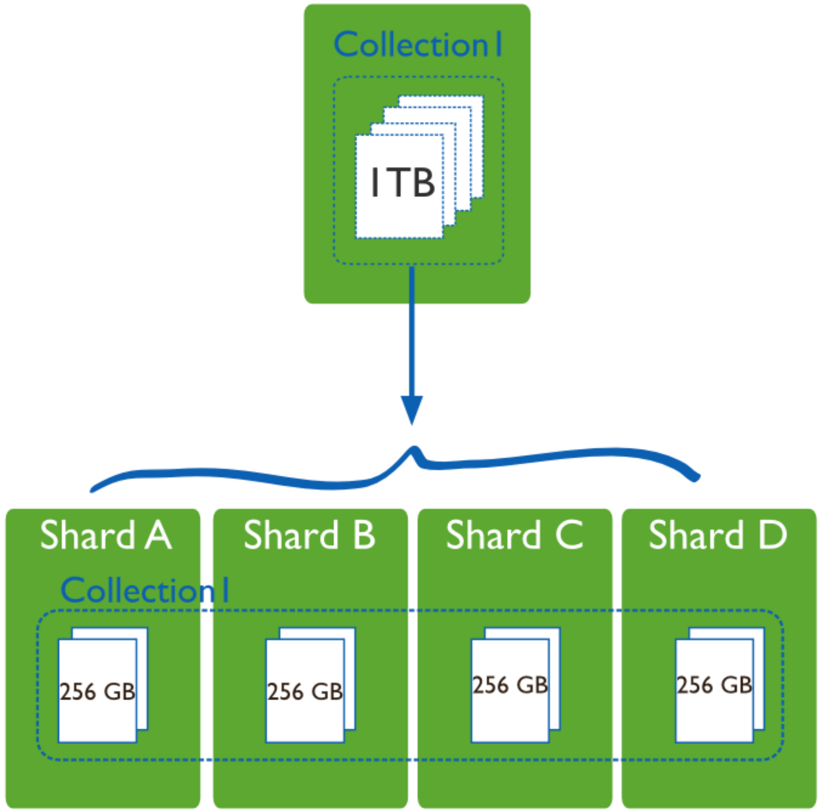
- ./sqoop import --connect jdbc:netezza://netezza-1.boulder.ibm.com:5480/BACC_DEV_BAT_AGGRACTIONS --driver org.netezza.Driver --username <userhere> --password <passwordhere> --table baccadm.dm_part_2 --split-by sequence_nbr --target-dir /sqoop/dm_part2

- $ bin/sqoop import --connect jdbc:mysql://localhost:3306/hadoop --driver com.mysql.jdbc.Driver --username root --password root --table bse30

- $ bin/sqoop import --connect jdbc:mysql://localhost:3306/hadoop --driver com.mysql.jdbc.Driver --username root --password root --table bsefmcg

DATABASE SHARDING

- Sharding is a type of database partitioning that separates very large databases the into smaller, faster, more easily managed parts called data shards. The word shard means a small part of a whole.



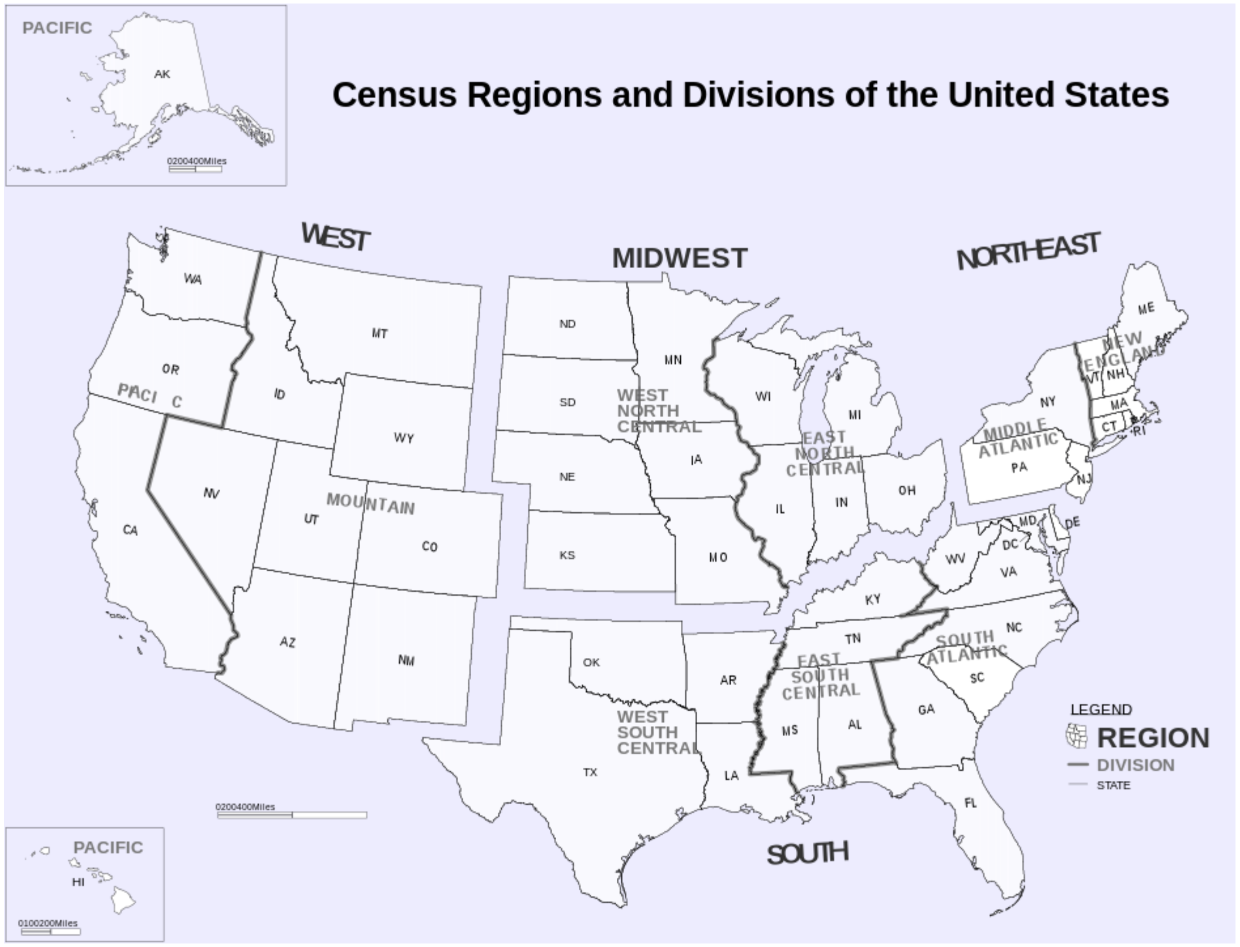
THE UNIVERSITY of CHICAGO GRAHAM SCHOOL of CONTINUING LIBERAL AND PROFESSIONAL STUDIES

SHARDING EXPLAINED

- Sharding your database involves breaking up your big database into many, much smaller databases that share nothing and can be spread across multiple servers
  - Technically, sharding is a synonym for horizontal partitioning.
  - But can also refer to any database partitioning that makes very large database more manageable
- Reasons behind sharding:
  - Data shards can be distributed across a number of inexpensive commodity servers.
  - Data shards have comparatively little restriction as far as hardware and software requirements are concerned.

SIMPLE EXAMPLE OF SHARDING: SPLIT US CUSTOMERS BY REGION

Assuming there is only one single location for each customer



THE UNIVERSITY of CHICAGO GRAHAM SCHOOL of CONTINUING LIBERAL AND PROFESSIONAL STUDIES

DISADVANTAGES OF SHARDING

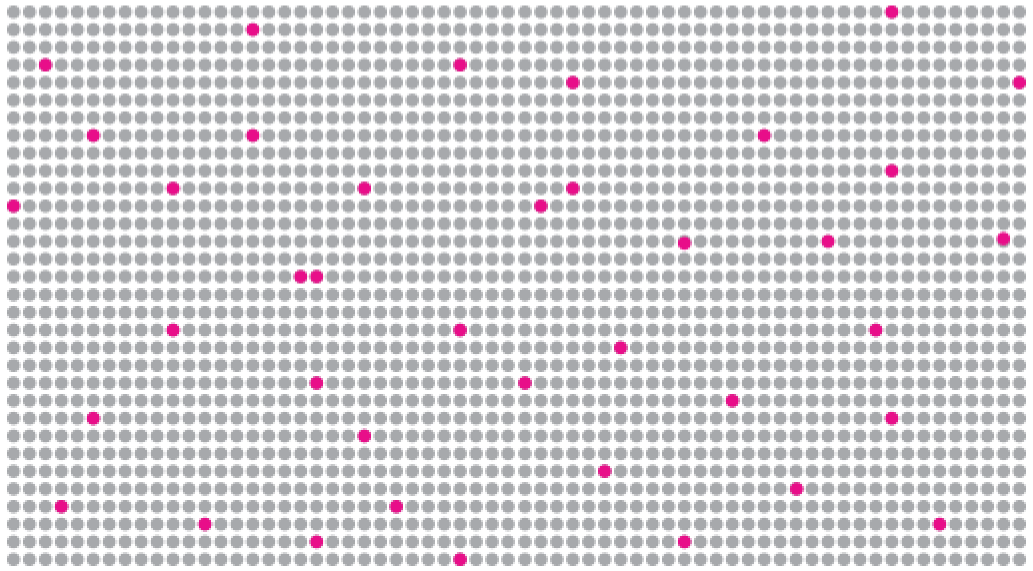
- Sharding a database that holds less structured data, can be very complicated, and the resulting shards may be difficult to maintain
- Difficult to optimize sharded database due to introduced complexity
- The sharding software that partitions, balances, coordinates, and ensures integrity can fail
- Complex backups and failover procedures
- Increased operational complexity:
    - Adding/removing indexes
    - Adding/deleting columns
    - Modifying the schema

HBASE: DISTRIBUTED COLUMN-ORIENTED DATABASE BUILT ON TOP OF HDFS.

HBase (short) History

- Began as project by Powerset to process massive amounts of data for natural language search
- 2006: Google releases paper on BigTable
- Open-source implementation of Google’s BigTable
  - Lots of semi-structured data
  - Commodity Hardware
  - Horizontal Scalability
  - Tight integration with MapReduce
- Developed as part of Apache’s Hadoop project and runs on top of HDFS (Hadoop Distributed Filesystem)
  - Provides fault-tolerant way of storing large quantities of sparse data.

HBase is well suited for sparse data sets



- **Cells with missing values** remain **empty** and **consume no storage**

Each table in HBase contains rows and columns, similar to a traditional database.

HBase is not a relational database and does not support a structured query language like SQL

	Primary Key	CF1			CF2				…
		colA	colB	colC	colA	colB	colC	colD	
R1	axxx	val		val	val			val	
	…								
	gxxx	val			val	val	val		
R2	hxxx	val	val	val	val	val	val	val	
	…								
	jxxx	val							
R3	kxxx	val		val	val			val	
	…								
	rxxx	val	val	val	val	val	val		
…	sxxx	val						val	

UNIVERSITY of CHICAGO GRAHAM SCHOOL of CONTINUING LIBERAL AND PROFESSIONAL STUDIES

EACH TABLE IN HBASE MUST HAVE A COLUMN, DEFINED AS A PRIMARY KEY

Values in HBase are stored sorted by Primary Keys and can be accessed only using a Primary Key (or by the range of Primary Keys”)

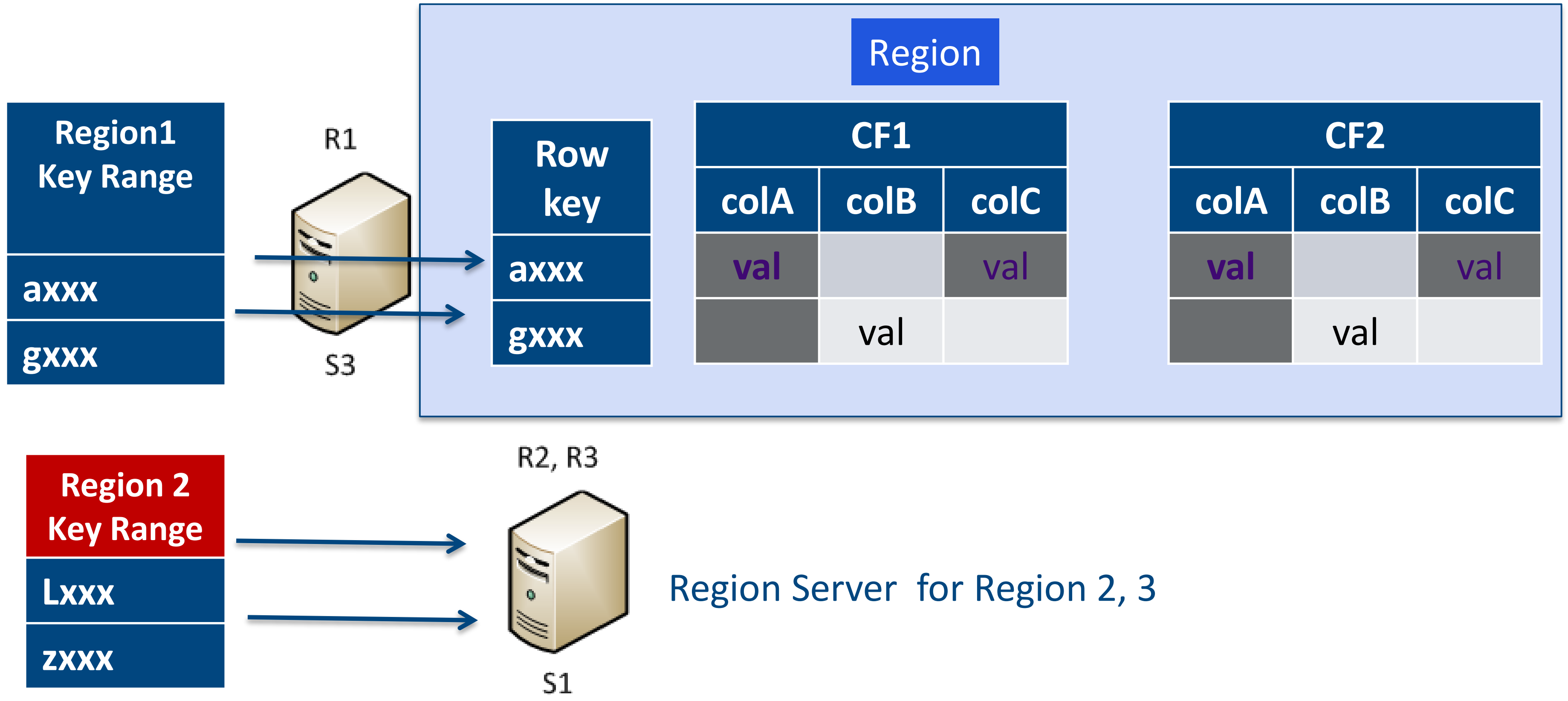
	Primary Key	CF1			CF2				…
		colA	colB	colC	colA	colB	colC	colD	
	axxx	val		val	val			val	
R1	…								
	gxxx	val			val	val	val		
	hxxx	val	val	val	val	val	val	val	
R2	…								
	jxxx	val							
	kxxx	val		val	val			val	
R3	…								
	rxxx	val	val	val	val	val	val		
…	sxxx	val						val	

HBase allows grouping of attributes into column families, so they are stored together.

This is different from a row-oriented relational database, where all the columns of a given row are stored together.

	Primary Key	CF1			CF2				…
		colA	colB	colC	colA	colB	colC	colD	
R1	axxx	val		val	val			val	
	…								
	gxxx	val			val	val	val		
R2	hxxx	val	val	val	val	val	val	val	
	…								
	jxxx	val							
R3	kxxx	val		val	val			val	
	…								
	rxxx	val	val	val	val	val	val		
…	sxxx	val						val	

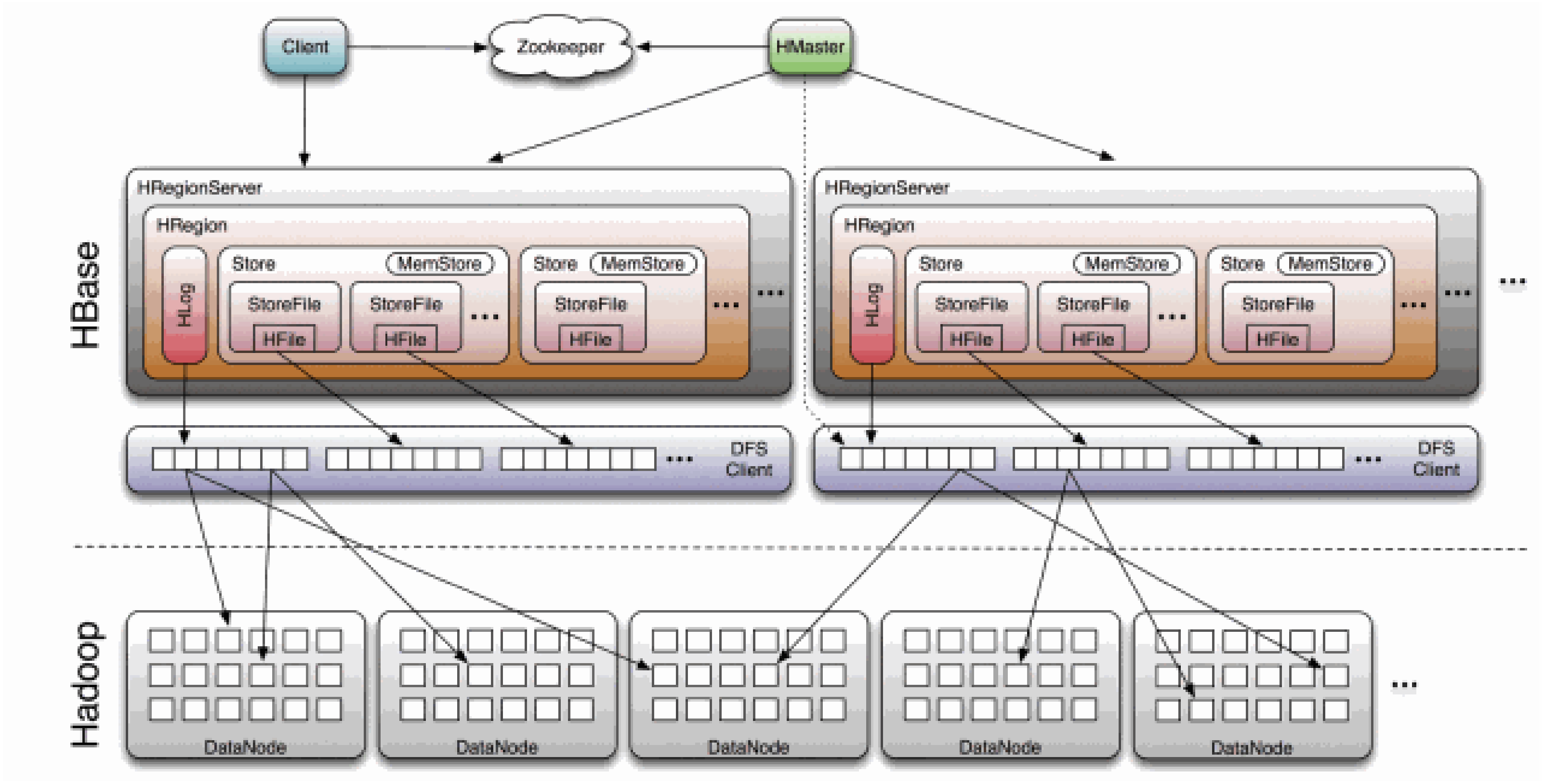
Tables are automatically sharded into regions and spread across cluster



Source: MapR, Introduction to Apache HBase, MapR Tables, and Security

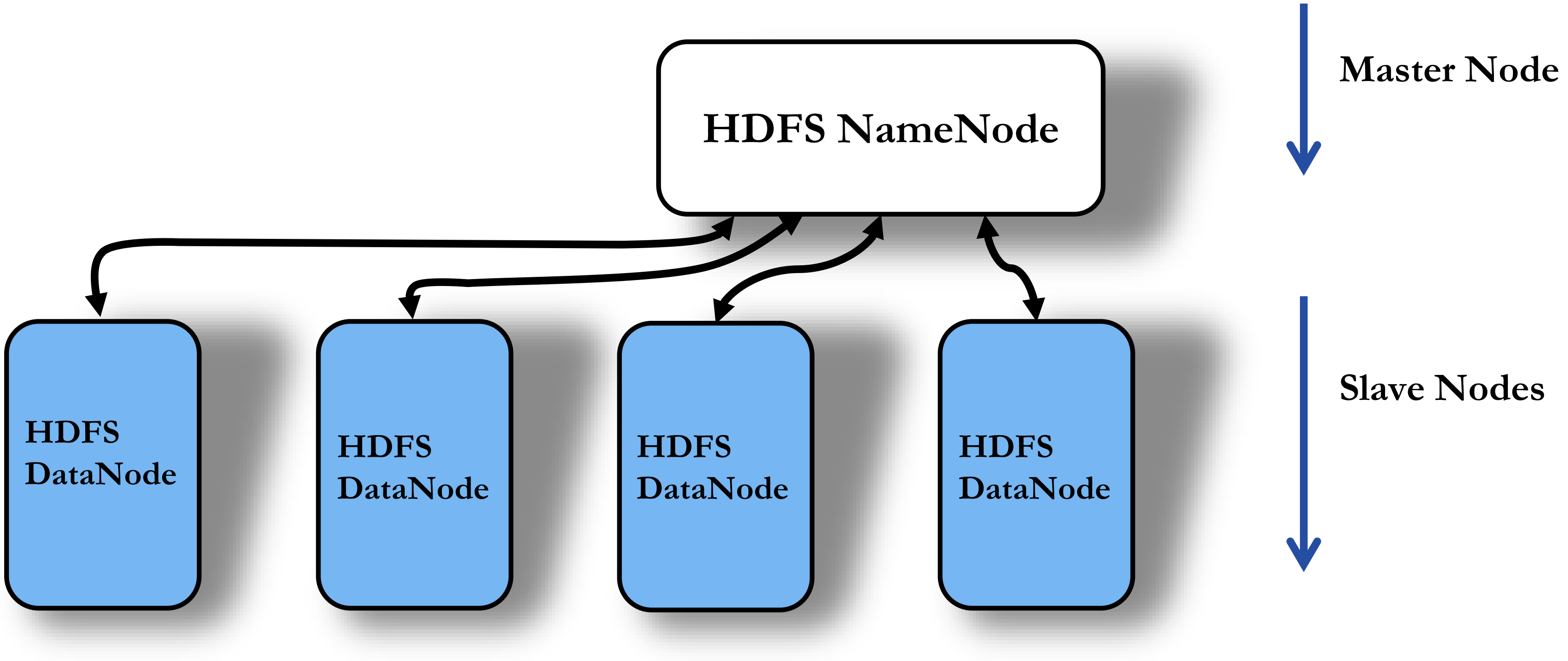
HBase is stored in HDFS

- HBase is made of a set of tables, where each table is stored in HDFS

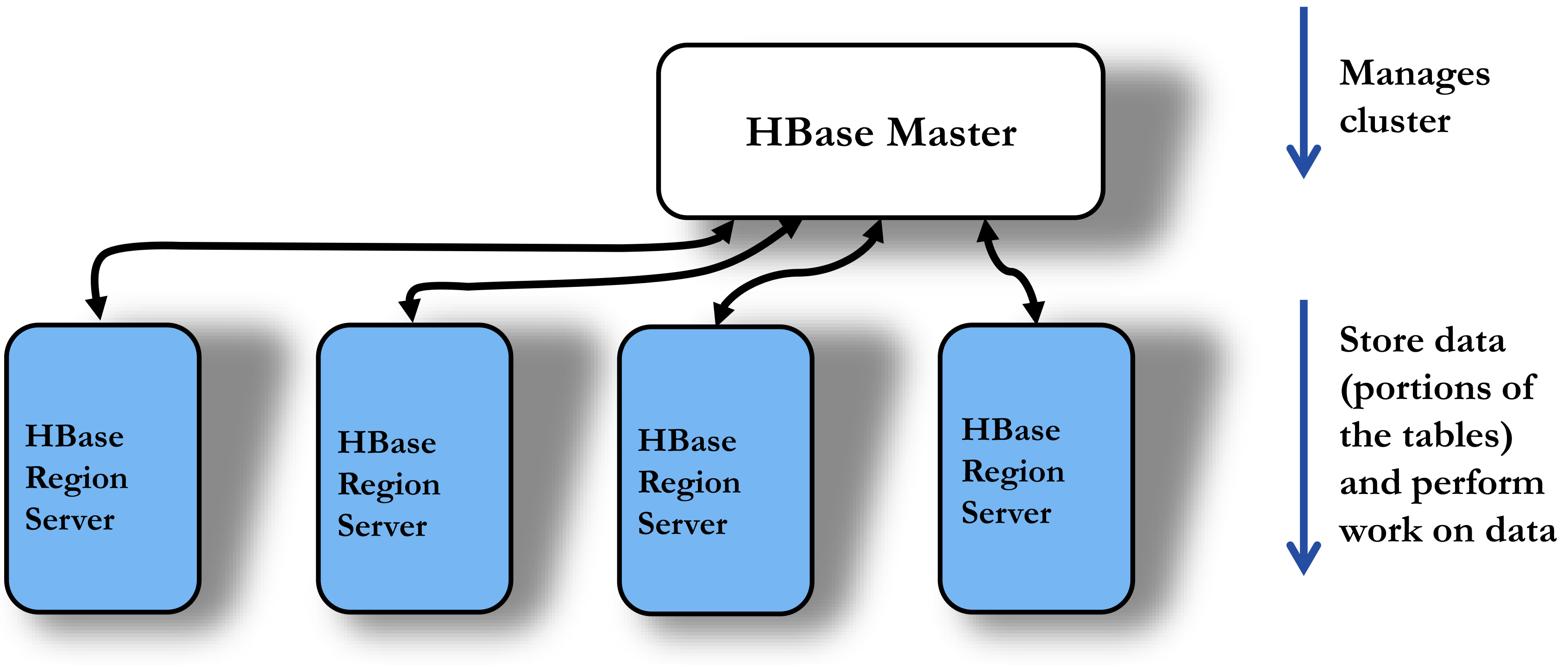


THE UNIVERSITY of CHICAGO GRAHAM SCHOOL of CONTINUING LIBERAL AND PROFESSIONAL STUDIES

Similar to HDFS Model with NameNode and DataNodes



HBase has HBase Master and RegionServers



HBase Data Model- Cells

- Value for each cell is specified by complete coordinates:
  - RowKey → Column Family → Column → Version: Value
  - Key:CF:Col:Version:Value
    - Where each column consists of any number of versions
    - Version is often a timestamp

RowKey	CF:Qualifier	version	value
Nick K	Data:street	2015-10-06:22:04:15	Main street

THE UNIVERSITY of CHICAGO GRAHAM SCHOOL of CONTINUING LIBERAL AND PROFESSIONAL STUDIES

HBase internals

- HBase writing:
  - It is first written to a commit log (write ahead log)
  - Then the row is written into memstore (in memory), where the data is ordered
  - Once memstore gets full, the data is persisted into HFile on disk
    - Minor compactions are triggered each time a memstore is flushed
    - Major compactions run about every 24 hours (after the currently oldest store file was written), and merge together all store files into one.
- HBase reading:
  - Single region might contain multiple HFiles
  - All HFiles must be merged to get a result of read
  - HFiles are combined during compactions (minor and major)

HBase Is Not …

- Tables have one primary index, the *row key*.
- No join operators
  - As of today cannot use SQL or Hive with HBase
- Limited atomicity and transaction support.
  - HBase supports multiple batched mutations of single rows only.
  - Data is unstructured and untyped.
- Cannot be accessed or manipulated via SQL.
  - Programmatic access via Java, REST, or Thrift APIs.
  - Scripting via JRuby.

HBASE OPERATIONS

- Operations are based on row keys

- Single-row operations:
  - Put
  - Get
  - Scan

- Multi-row operations:
  - Scan
  - MultiPut
  - No built-in joins (use MapReduce)

HBase Summary

HBase pros:

- Good for sparse data, since there is no schema
- The data is sharded into regions and automatically split, when a region becomes too big
- Scalable clustering

HBase cons:

- Takes a lot of disk space, because it is very verbose and there is no schema

Hive and Pig

Query Languages for Hadoop

- **Java:** Hadoop's Native Language
- **Pig:** Query and Workflow Language
- **Hive:** SQL-Based Language
- **HBase:** Column-oriented Database for MapReduce

Need for High-Level Languages

- Hadoop is great for large-data processing!
  - But writing Java programs for everything is verbose and slow
  - Not everyone wants to (or can) write Java code
- Solution: develop higher-level data processing languages
  - Hive: HQL is like SQL
  - Pig: Pig Latin is a bit like Perl

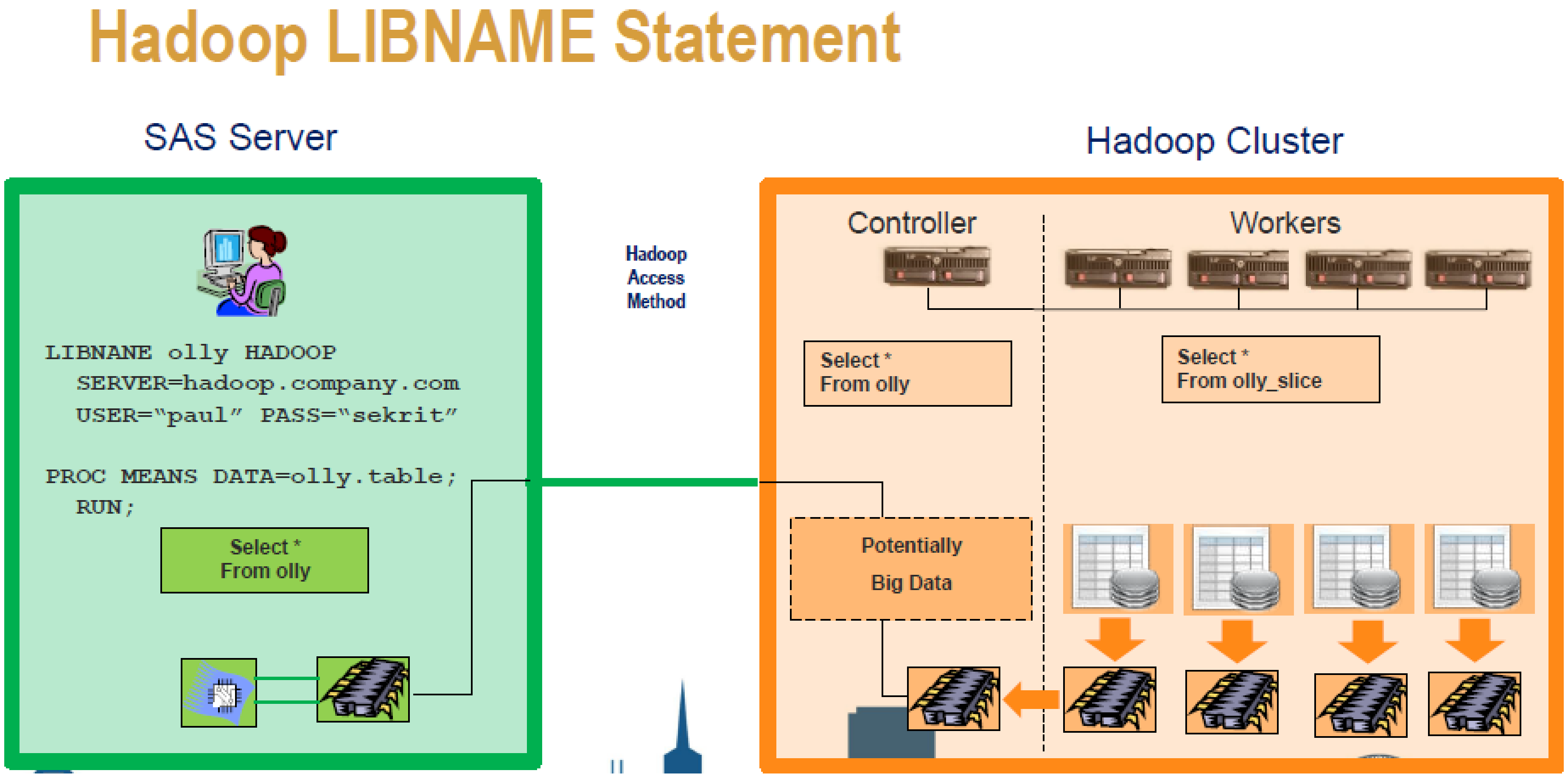
Hive and Pig

- Hive: data warehousing application in Hadoop
  - Query language is HQL, variant of SQL
  - Tables stored on HDFS as flat files
  - Developed by Facebook, now open source

- Pig: large-scale data processing system
  - Scripts are written in Pig Latin, a dataflow language
  - Developed by Yahoo!, now open source
  - Roughly 1/3 of all Yahoo! internal jobs
- Common idea:
  - Provide higher-level language to facilitate large-data processing
  - Higher-level language "compiles down" to Hadoop jobs

Using Hadoop with SAS and SPSS

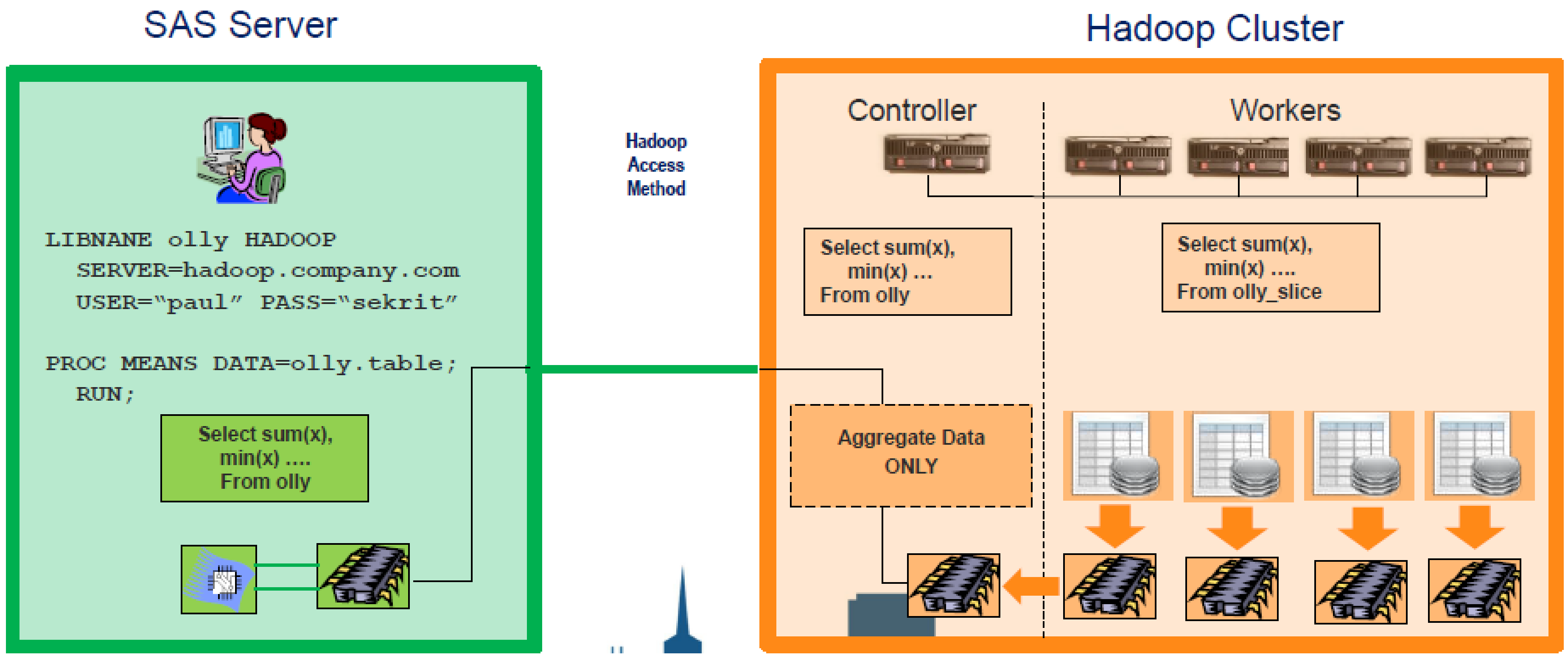
SAS Access Connection to HDFS (native)



```
LIBNANE olly HADOOP
   SERVER=hadoop.company.com
   USER="paul" PASS="sekrit"

PROC MEANS DATA=olly.table;
   RUN;
```

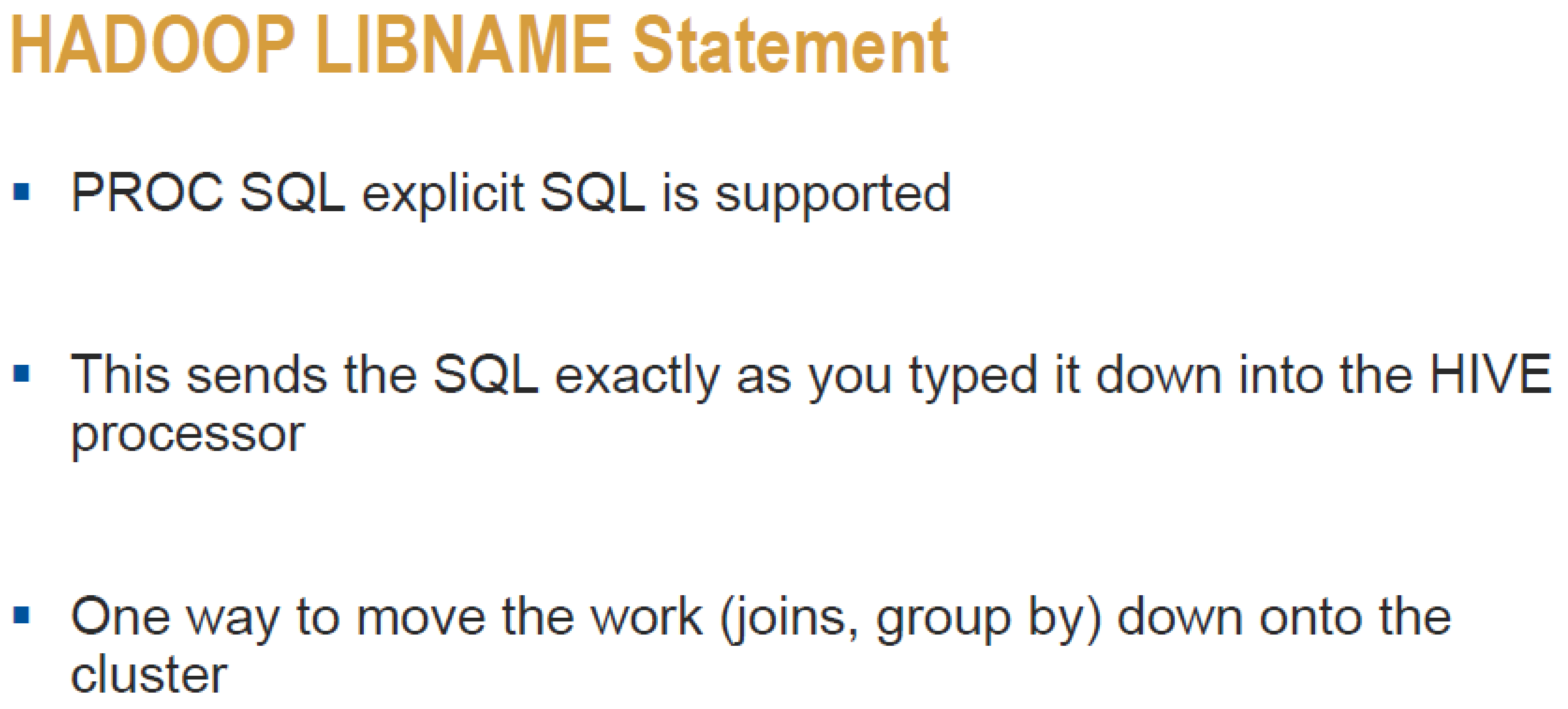
SAS Access Connection to HDFS (SQL Pasthru)

Hadoop LIBNAME Statement – with SQL Pasthru


SAS Server
Hadoop Cluster
Hadoop Access Method
Controller
Workers
Select sum(x), min(x) ... From olly
Select sum(x), min(x) .... From olly_slice
Aggregate Data ONLY
Select sum(x), min(x) .... From olly

```
LIBNANE olly HADOOP
  SERVER=hadoop.company.com
  USER="paul" PASS="sekrit"

PROC MEANS DATA=olly.table;
  RUN;
```

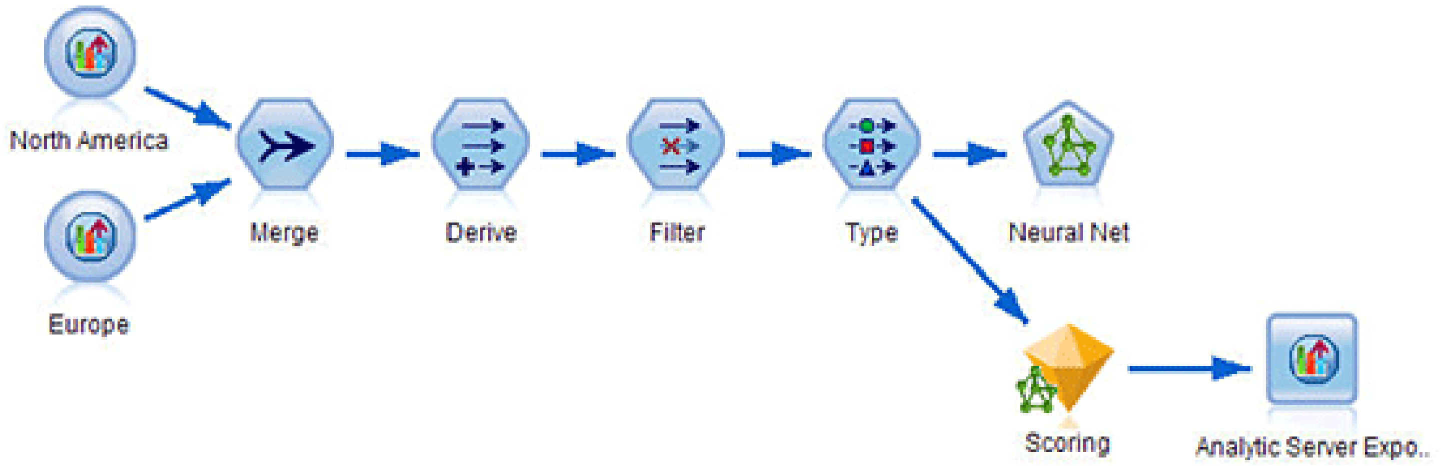
SAS Access Connection to HDFS (Proc SQL)



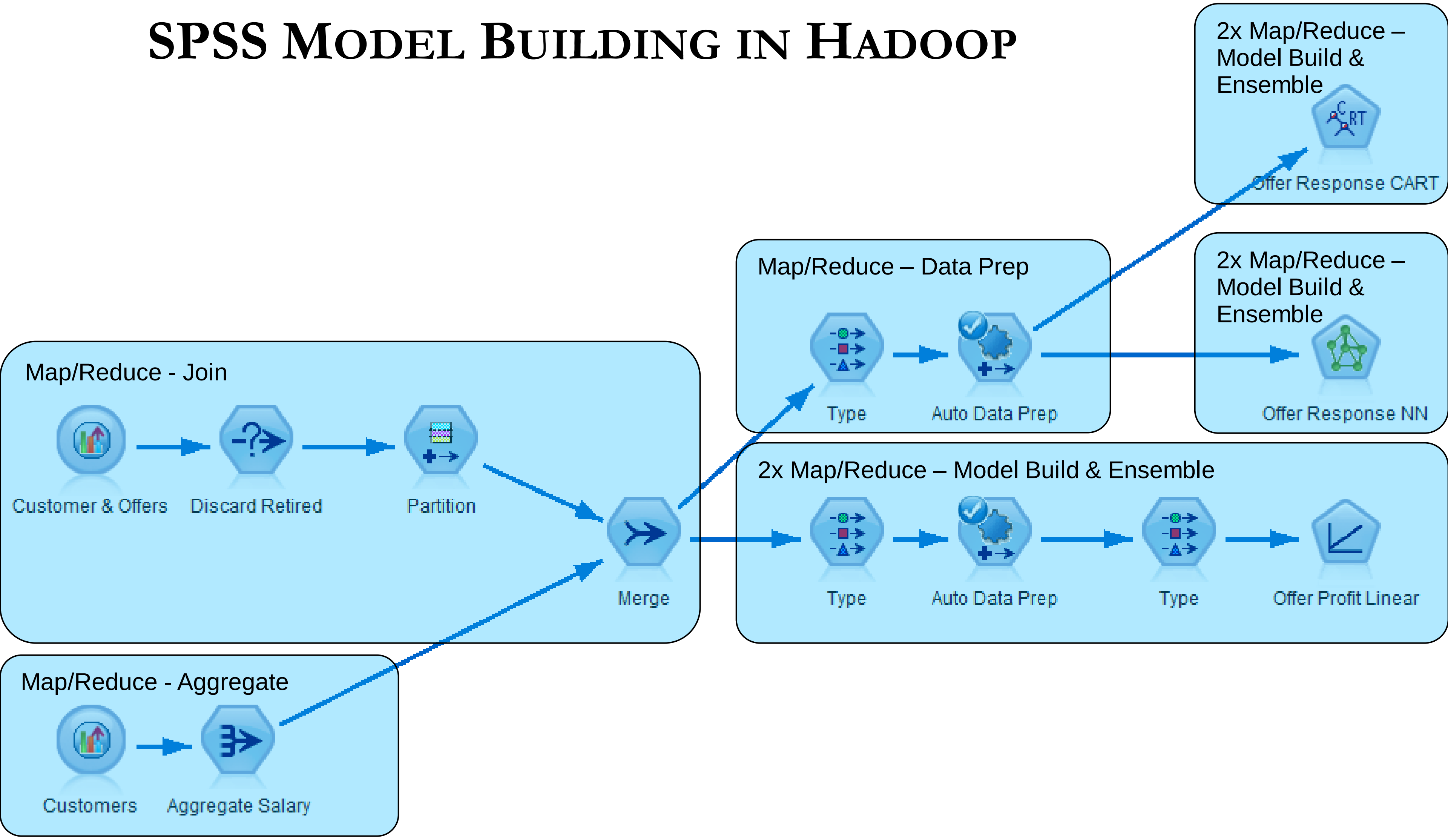
HADOOP LIBNAME Statement

- PROC SQL explicit SQL is supported
- This sends the SQL exactly as you typed it down into the HIVE processor
- One way to move the work (joins, group by) down onto the cluster

SPSS Modeler with SPSS Analytic Server

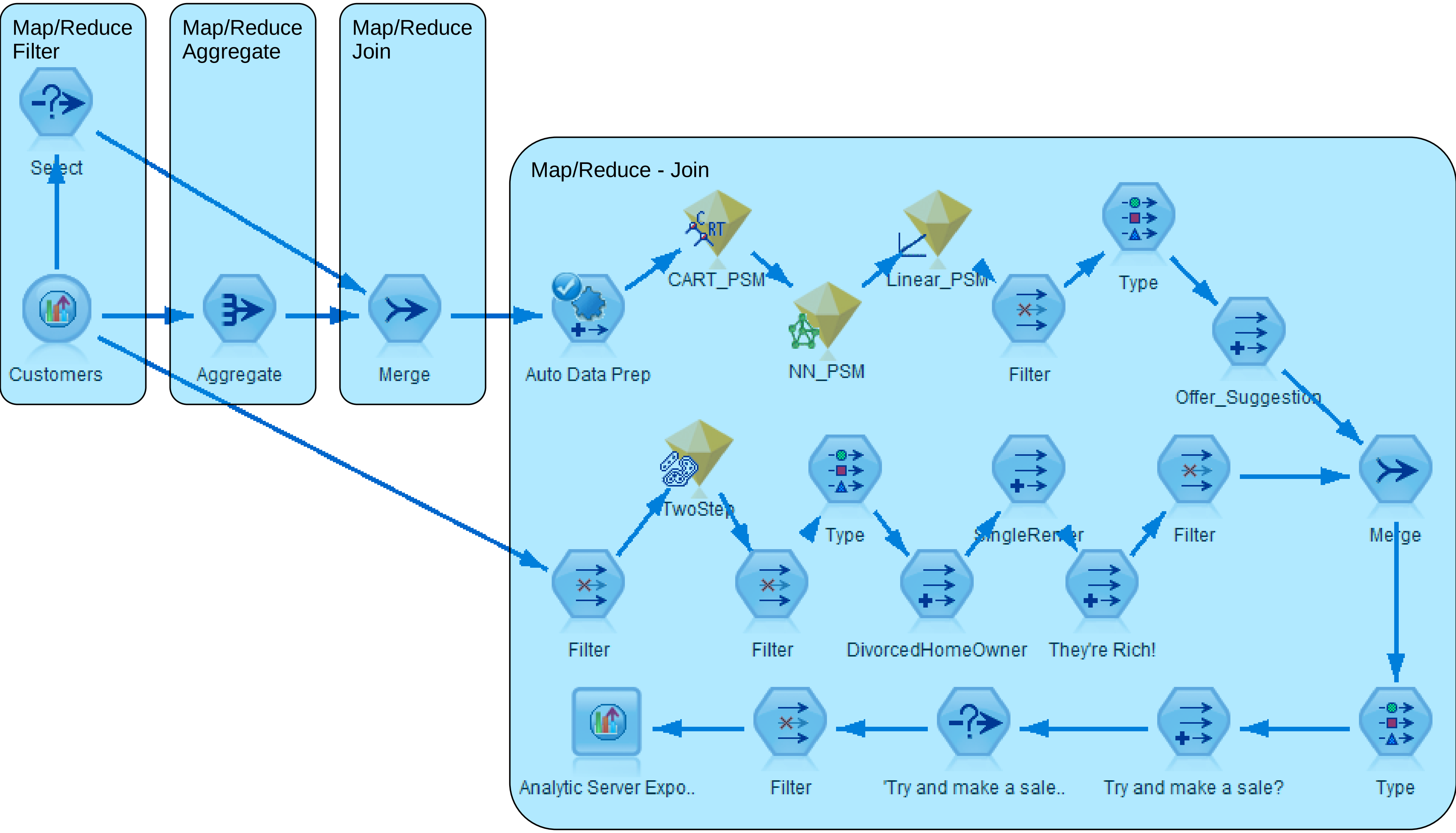


SPSS Model Building in Hadoop



THE UNIVERSITY of CHICAGO GRAHAM SCHOOL of CONTINUING LIBERAL AND PROFESSIONAL STUDIES

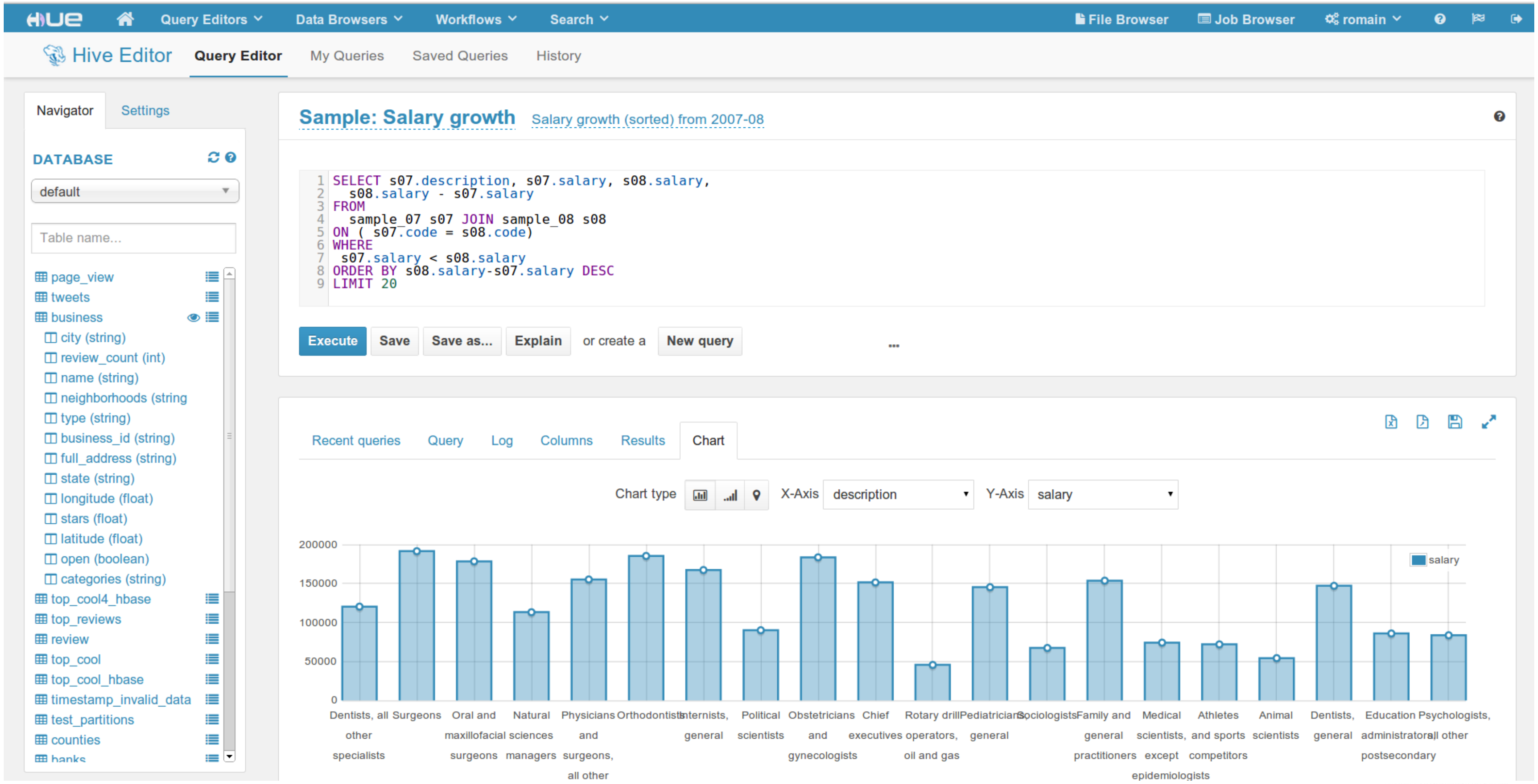
SPSS Model Scoring in Hadoop



THE UNIVERSITY *of* CHICAGO
GRAHAM SCHOOL
of CONTINUING LIBERAL AND PROFESSIONAL STUDIES

HUE

Hue is a web interface making it easier to interact with key Hadoop components



Sample: Salary growth Salary growth (sorted) from 2007-08

```
SELECT s07.description, s07.salary, s08.salary,
  s08.salary - s07.salary
FROM
  sample_07 s07 JOIN sample_08 s08
ON ( s07.code = s08.code)
WHERE
 s07.salary < s08.salary
ORDER BY s08.salary-s07.salary DESC
LIMIT 20
```

HIVE

APACHE HIVE

- A data warehouse infrastructure built on top of Hadoop for providing data summarization, query, and analysis
- **Hive Provides**
  - ETL
  - Structure
  - Access to different storage (HDFS or HBase)
  - Query execution via MapReduce
- **Key Building Principles**
  - SQL is a familiar language
  - Extensibility – Types, Functions, Formats, Scripts
  - Performance

DATA MODEL

- Tables
  - Typed columns (int, float, string, boolean)
  - Also, list: map (for JSON-like data)
- Partitions
  - For example, range-partition tables by date
- Buckets
  - Hash partitions within ranges (useful for sampling, join optimization)

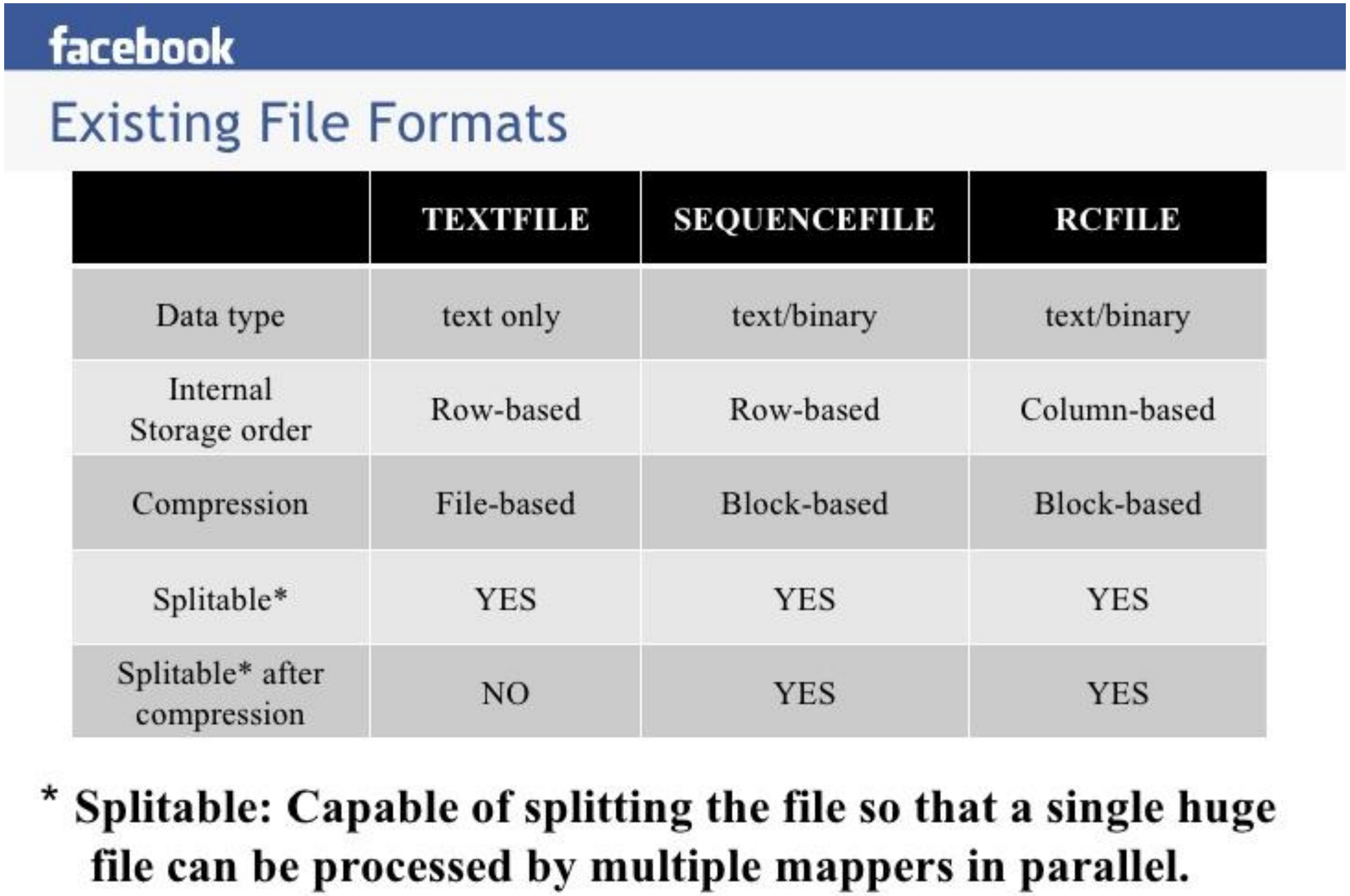
Metastore

- Database: namespace containing a set of tables
- Holds table definitions (column types, physical layout)
- Holds partitioning information
- Can be stored in Derby, MySQL, and many other relational databases

Physical Layout

- Warehouse directory in HDFS
  - E.g., /user/hive/warehouse
- Tables stored in subdirectories of warehouse
  - Partitions form subdirectories of tables
- Actual data stored in flat files
  - Control char-delimited text, SequenceFiles or RCFiles
  - With custom SerDe, can use arbitrary format

File format comparison



facebook

Existing File Formats

	TEXTFILE	SEQUENCEFILE	RCFILE
Data type	text only	text/binary	text/binary
Internal Storage order	Row-based	Row-based	Column-based
Compression	File-based	Block-based	Block-based
Splitable*	YES	YES	YES
Splitable* after compression	NO	YES	YES

* **Splitable: Capable of splitting the file so that a single huge file can be processed by multiple mappers in parallel.**

FASTER BIG DATA ON HADOOP WITH HIVE USING SEQUENCE FILE AND RCFILE

Sequence file

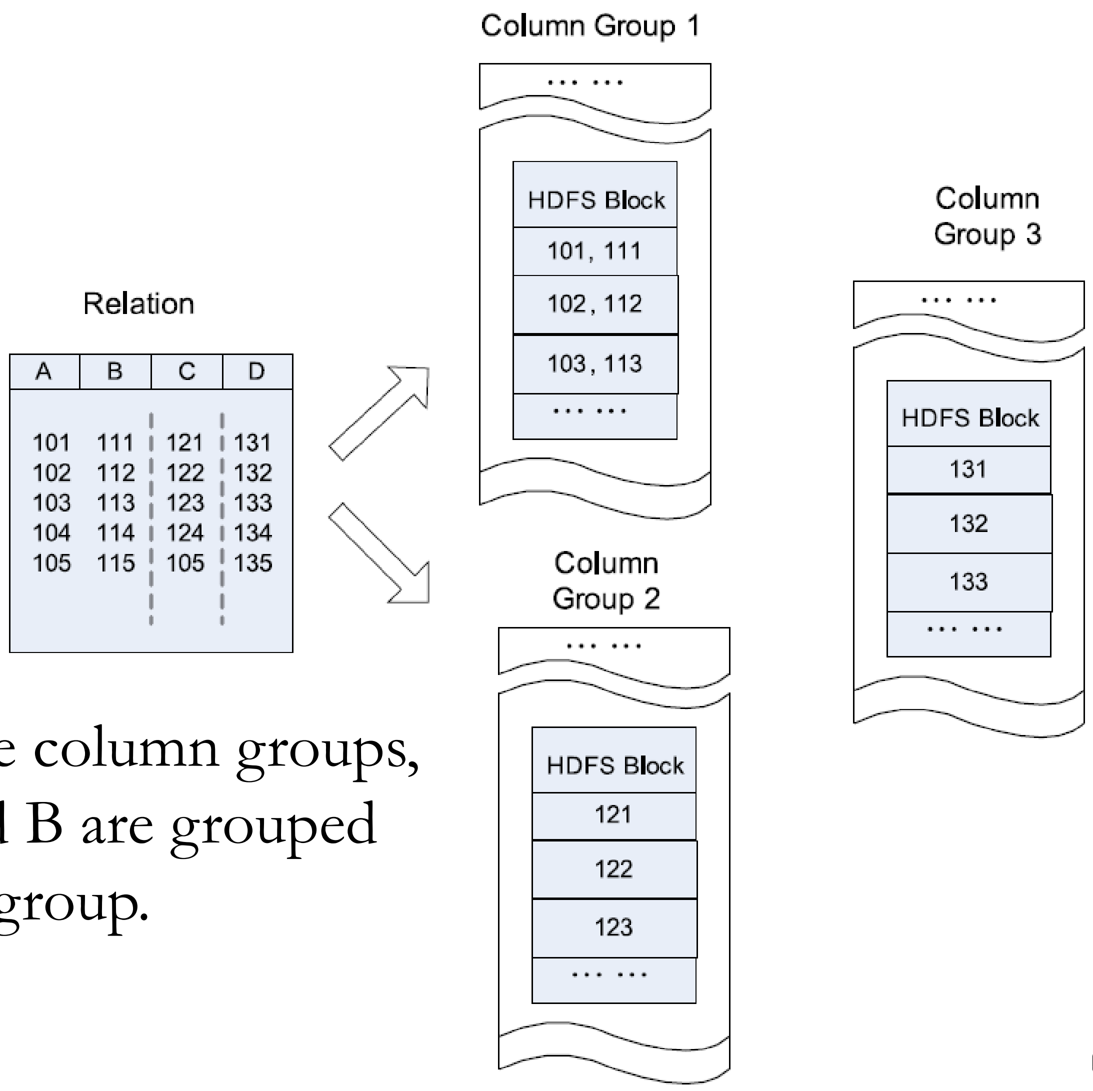
- Sequence Files are flat files, which are binary storage format for key value pairs. They have benefit of being more compact than text and fits well the map-reduce output format.
- Internally, the temporary outputs of maps are stored using Sequence Files
- Sequence files are also used as a container to store the small files. A very common use case when designing ingestion systems is to use Sequence files as containers and store any file related metadata(filename, path, creation time etc.) as the key and the file contents as the value.

RCFILE (RECORD COLUMNAR FILE)

RCFile

- RCFile format has been co-developed by Facebook (which is running the largest Hadoop and Hive installation in the world) to enable faster data loading and query processing
- The RCFile splits data horizontally into row groups.
  - For example, rows 1 to 100 are stored in one group and rows 101 to 200 in the next and so on. One or several groups are stored in a HDFS file.
  - The RCFile saves the row group data in a columnar format. So instead of storing row one then row two, it stores column one across all rows then column two across all rows and so on.
- The benefit of this data organization is that Hadoop's parallelism still applies since the row groups in different files are distributed redundantly across the cluster and processable at the same time.
- Subsequently, each processing node reads only the columns relevant to a query from a file and skips irrelevant ones
- Column-based compression is also more efficient. It can take advantage of similarity of the data in a column.

COLUMN-GROUP IN AN HDFS BLOCK.

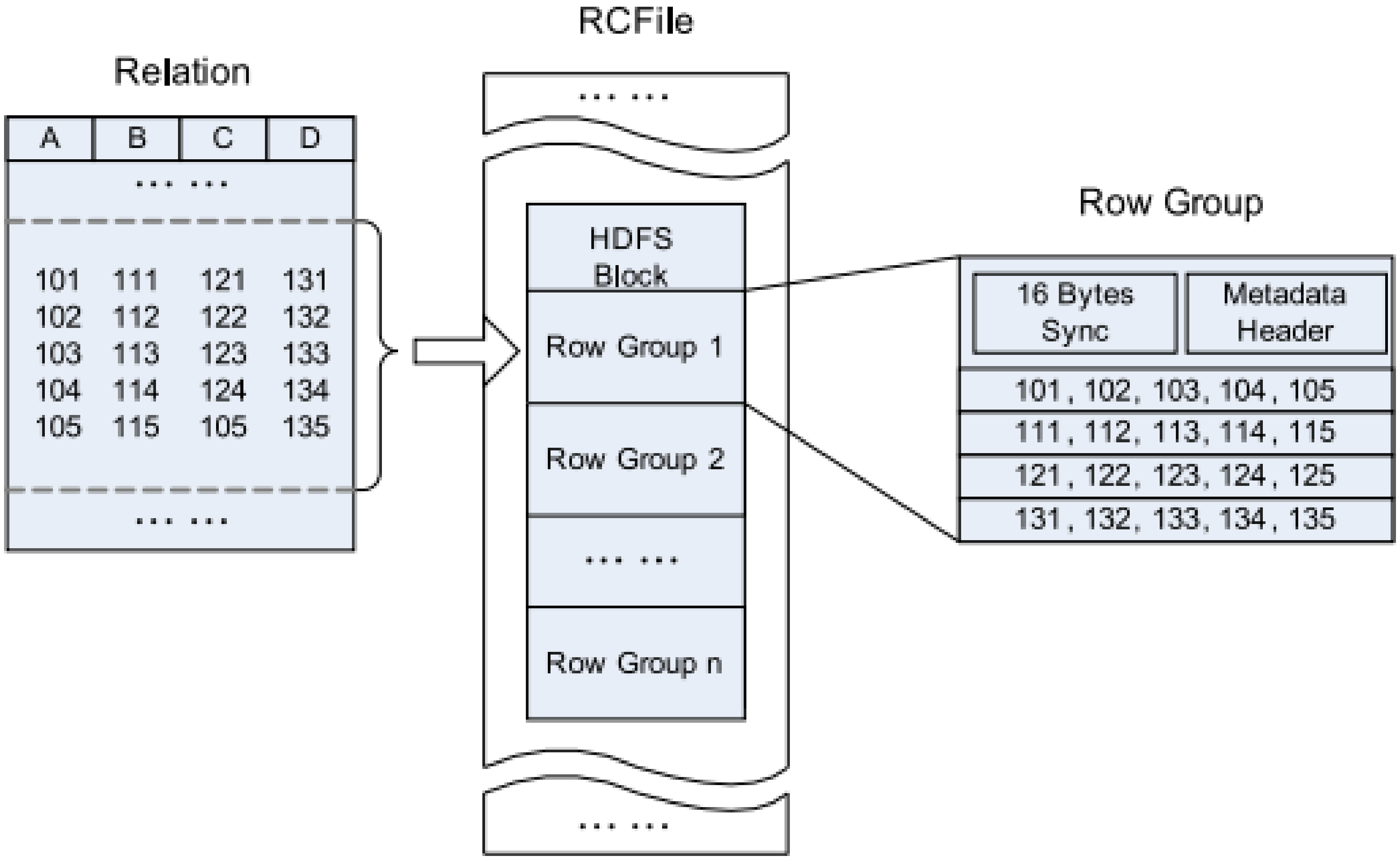


Four columns
are stored into three column groups,
since column A and B are grouped
in the first column group.

RCFile structure

- Based on the HDFS structure, a table can have multiple HDFS blocks
- All the records stored in an HDFS block are partitioned into row groups
- For a table, all row groups have the same size. Depending on the row group size and the HDFS block size, an HDFS block can have only one or multiple row groups
- A row group contains three sections
    1. Sync marker that is placed in the beginning of the row group to separate two continuous row groups in an HDFS block.
    2. Metadata header for the row group, which stores the information items on how many records are in this row group, how many bytes are in each column, and how many bytes are in each field in a column.
    3. Table data section that is actually a column-store, where all the fields in the same column are stored continuously together

RCFILE (RECORD COLUMNAR FILE)



THE UNIVERSITY of CHICAGO GRAHAM SCHOOL of CONTINUING LIBERAL AND PROFESSIONAL STUDIES

HIVE: EXAMPLE

- Hive looks similar to an SQL database
- Relational join on two tables:
  - Table of word counts from Shakespeare collection
  - Table of word counts from the bible

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

```
the   25848    62394
I     23031    8854
and   19671    38985
to    18038    13526
of    16700    34654
a     14170    8057
you   12702    2720
my    11297    4135
in    10797    12445
is    8882     6884
```

HIVE: BEHIND THE SCENES

```
SELECT s.word, s.freq, k.freq FROM shakespeare s
  JOIN bible k ON (s.word = k.word) WHERE s.freq >= 1 AND k.freq >= 1
  ORDER BY s.freq DESC LIMIT 10;
```

(Abstract Syntax Tree)

```
(TOK_QUERY (TOK_FROM (TOK_JOIN (TOK_TABREF shakespeare s) (TOK_TABREF bible k) (= (. (TOK_TABLE_OR_COL s) word) (.
(TOK_TABLE_OR_COL k) word)))) (TOK_INSERT (TOK_DESTINATION (TOK_DIR TOK_TMP_FILE)) (TOK_SELECT
(TOK_SELEXPR (. (TOK_TABLE_OR_COL s) word)) (TOK_SELEXPR (. (TOK_TABLE_OR_COL s) freq)) (TOK_SELEXPR (.
(TOK_TABLE_OR_COL k) freq))) (TOK_WHERE (AND (>= (. (TOK_TABLE_OR_COL s) freq) 1) (>= (. (TOK_TABLE_OR_COL k) freq)
1))) (TOK_ORDERBY (TOK_TABSORTCOLNAMEDESC (. (TOK_TABLE_OR_COL s) freq))) (TOK_LIMIT 10)))
```

(one or more of MapReduce jobs)

HIVE: BEHIND THE SCENES

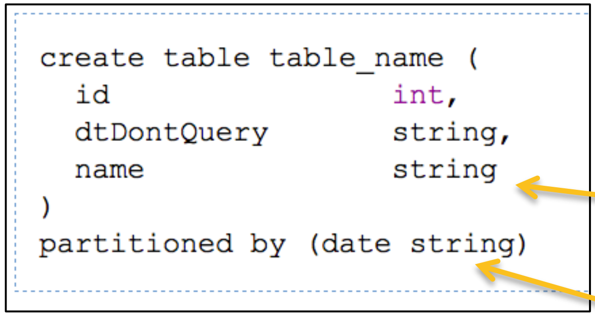
```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-2 depends on stages: Stage-1
  Stage-0 is a root stage

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Alias -> Map Operator Tree:
        s
          TableScan
            alias: s
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 0
                value expressions:
                      expr: freq
                      type: int
                      expr: word
                      type: string
        k
          TableScan
            alias: k
            Filter Operator
              predicate:
                  expr: (freq >= 1)
                  type: boolean
              Reduce Output Operator
                key expressions:
                      expr: word
                      type: string
                sort order: +
                Map-reduce partition columns:
                      expr: word
                      type: string
                tag: 1
                value expressions:
                      expr: freq
                      type: int
      Reduce Operator Tree:
        Join Operator
          condition map:
               Inner Join 0 to 1
          condition expressions:
            0 {VALUE._col0} {VALUE._col1}
            1 {VALUE._col0}
          outputColumnNames: _col0, _col1, _col2
          Filter Operator
            predicate:
                expr: ((_col0 >= 1) and (_col2 >= 1))
                type: boolean
            Select Operator
              expressions:
                    expr: _col1
                    type: string
                    expr: _col0
                    type: int
                    expr: _col2
                    type: int
              outputColumnNames: _col0, _col1, _col2
              File Output Operator
                compressed: false
                GlobalTableId: 0
                table:
                    input format: org.apache.hadoop.mapred.SequenceFileInputFormat
                    output format: org.apache.hadoop.hive.ql.io.HiveSequenceFileOutputFormat

  Stage: Stage-2
    Map Reduce
      Alias -> Map Operator Tree:
        hdfs://localhost:8022/tmp/hive-training/364214370/10002
            Reduce Output Operator
              key expressions:
                    expr: _col1
                    type: int
              sort order: -
              tag: -1
              value expressions:
                    expr: _col0
                    type: string
                    expr: _col1
                    type: int
                    expr: _col2
                    type: int
      Reduce Operator Tree:
        Extract
          Limit
            File Output Operator
              compressed: false
              GlobalTableId: 0
              table:
                  input format: org.apache.hadoop.mapred.TextInputFormat
                  output format: org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat

  Stage: Stage-0
    Fetch Operator
      limit: 10
```

THE UNIVERSITY of CHICAGO GRAHAM SCHOOL of CONTINUING LIBERAL AND PROFESSIONAL STUDIES

Hive DDL Commands

- **CREATE TABLE** sample (foo INT, bar STRING) **PARTITIONED BY** (ds STRING);
- **SHOW TABLES** '.*s';
- **DESCRIBE** sample;
- **ALTER TABLE** sample **ADD COLUMNS** (new_col INT);
- **DROP TABLE** sample;

A table in Hive is an HDFS directory in Hadoop



```
create table table_name (
  id                  int,
  dtDontQuery         string,
  name                string
)
partitioned by (date string)
```

Schema is known at creation time (like DB schema)

Partitioned tables have “sub-directories”, one for each partition

HIVE DML

Load data from local file system

Delete previous data from that table

```
LOAD DATA LOCAL INPATH './sample.txt' OVERWRITE INTO TABLE sample;
```

```
LOAD DATA INPATH '/user/falvariz/hive/sample.txt' INTO TABLE partitioned_sample
PARTITION (ds='2012-02-24');
```

Load data from HDFS

Augment to the existing data

Must define a specific partition for partitioned tables

Loaded data are files copied to HDFS under the corresponding directory

QUERY EXAMPLES I: SELECT & FILTER

- **SELECT** foo **FROM** sample **WHERE** ds='2012-02-24';
- **INSERT OVERWRITE DIRECTORY** '/tmp/hdfs_out' **SELECT** * **FROM** sample **WHERE** ds='2012-02-24';
- **INSERT OVERWRITE LOCAL DIRECTORY** '/tmp/hive-
  SELECT * **FROM** sample;

Create HDFS dir for the output

Create local dir for the output

QUERY EXAMPLES II: AGGREGATION & GROUPING

- **SELECT MAX**(foo) **FROM** sample;

- **SELECT** ds, **COUNT**(*), **SUM**(foo) **FROM** sample **GROUP BY** ds;

Hive allows the From clause to come first !!!

Store the results into a table

- **FROM** sample s **INSERT OVERWRITE TABLE** bar **SELECT** s.bar, count(*) WHERE s.foo > 0 **GROUP BY** s.bar;

This new syntax is to facilitate the "Multi-Insertion"

Example III: Joins

```
CREATE TABLE customer (id INT,name STRING,address STRING)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '#';

CREATE TABLE order_cust (id INT,cus_id INT,prod_id INT,price INT)
  ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t';
```

- `SELECT * FROM customer c JOIN order_cust o ON (c.id=o.cus_id);`

-
  ```
  SELECT c.id, c.name, c.address, ce.exp
  FROM customer c JOIN (SELECT cus_id,sum(price) AS exp
                        FROM order_cust
                        GROUP BY cus_id) ce ON (c.id=ce.cus_id);
  ```

HIVE DDL

- Drop table
- Create DDL
- Load data from HDFS
- Load data from Linux
- Load into the same table vs. overwrite
- Indexes: create, show and drop

Hive Aggregations and Joins

- Aggregate operations:
  - count
  - sum
  - avg
  - min
  - max
- Joins:
  - Hive is optimized to have the largest table on the right in your Join query. The table on the right will be streamed from disk and the other Join tables will attempt to be read from memory

Hive sorting

- In Hive the ORDER BY clause is similar to other versions of SQL.
  - The ORDER BY clause performs a total ordering of the result set – all through a single reducer. For large data sets this can mean high latency.
- We can contrast that with Hive's SORT BY clause.
  - Using the SORT BY causes the data to be ordered in EACH reducer – so each reducers output is sorted. If there are multiple reducers then the final results may not be sorted as desired.
- Hive supports *SORT BY* which sorts the data per reducer
  - The difference between "order by" and "sort by" is that the former guarantees total order in the output while the latter only guarantees ordering of the rows within a reducer. If there are more than one reducer, "sort by" may give partially ordered final results.

Casting in Hive

- Hive will do implicit conversions when you try to compare one type to another,

- You can also use the cast() function to explicitly convert a value of one type to another.

  - Cast (salary as float)

Thank You!

THE UNIVERSITY *of* CHICAGO
GRAHAM SCHOOL
of CONTINUING LIBERAL AND PROFESSIONAL STUDIES