CARLSON SCHOOL OF MANAGEMENT

UNIVERSITY OF MINNESOTA

Optimize Hive Query Performance

MSBA 6330 Prof Liu

Slides credits: Cloudera Academic Partners Program

Hive Optimization

- In this chapter, you will learn
  - Use faster engines for Hive
  - How to choose storage formats for Hive tables
  - How to partition tables to reduce amount of data read for a query
  - How to understand and write better performing Hive queries
  - Why/how to use bucketing

Optimize Hive Query Performance

USE FASTER PROCESSING ENGINES

Uses a Faster Execution Engine

- Hive uses MapReduce (MR) engine by default, but it can also leverage faster engines
  - – Tez: `set hive.execution.engine=tez;`
    - Tez also offers a customizable execution architecture, permitting dynamic performance optimizations, and dramatically improving the speed of execution.
  - – Spark: `set hive.execution.engine=spark;`
    - In memory computing, more customizable execution architecture.
    - Expect a long delay as Spark initializes after you submit the first query
    - Subsequent queries run without a delay
    - Hive on Spark requires more memory on cluster than Hive on MapReduce
- To see the current engine, use `SET hive.execution.engine;`

Optimize Hive Query Performance

USE FASTER STORAGE FORMATS

Choose a File Format

- Hive supports multiple storage formats

```
CREATE TABLE tablename (colname DATATYPE, …)
ROW FORMAT DELIMITED FIELDS TERMINATED BY char
STORED AS format;
```

- Format Options
  - TEXTFILE
  - SEQUENCEFILE
  - AVRO
  - PARQUET
  - RCFILE
  - ORCFILE

Considerations for Choosing a File Format

- Hadoop and its ecosystem support many file formats
  - You can ingest data in one format and convert to another as needed
- Selecting the format for your dataset involves several considerations
  - Ingest pattern
  - Tool compatibility
  - Expected lifetime
  - Storage and performance requirements

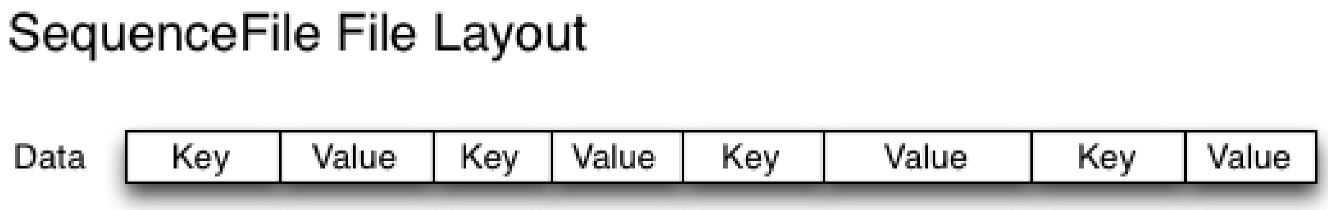
TEXTFILE

- Text files are the most basic file type in Hadoop
  - Can be read or written from virtually any programming language
  - Comma- and tab-delimited files are compatible with many applications
- Text files are human-readable
  - All values are represented as strings
  - Useful when debugging
- At scale, this format is inefficient
  - Representing numeric values as strings wastes storage space
  - Difficult to represent binary data such as images
    - Often resort to techniques such as Base64 encoding
  - Conversion to/from native types adds performance penalty
- **Verdict: Good interoperability, but poor performance**

How many bytes to store `108125150` as text (9 bytes) or integer (4 bytes)?

SequenceFile Format

- SequenceFiles store key-value pairs in a **binary** container format
  - Less verbose and more efficient than text files
  - Capable of storing binary data such as images
  - Format is **Java-specific** and tightly coupled to Hadoop
    - Use internally for temporal outputs of mappers
  - Support splitable compression
- **Verdict: Good performance, but poor interoperability**



SequenceFile File Layout

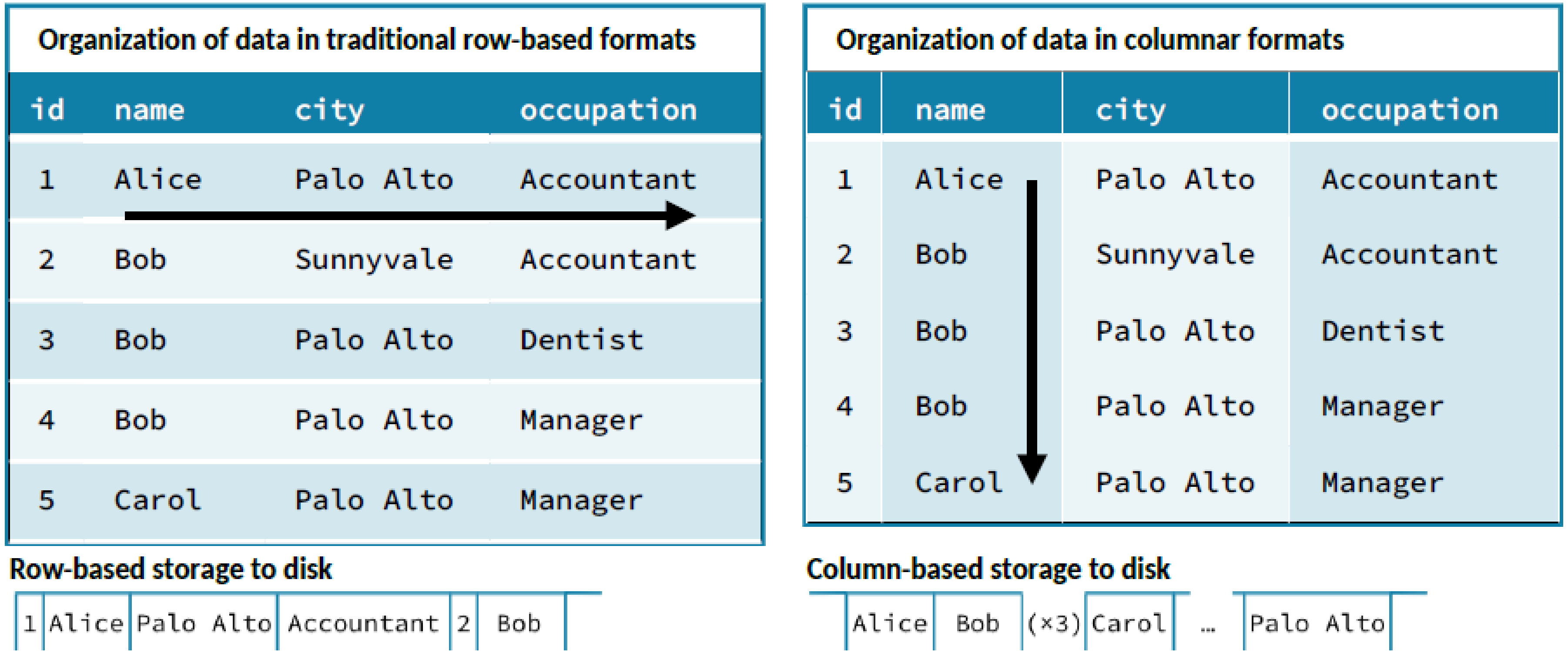
Data	Key	Value	Key	Value	Key	Value	Key	Value

Apache Avro File Format

- Apache Avro is an efficient data serialization framework
- Avro also defines a **binary** data file format for storing Avro records
  - Similar to SequenceFile format
- Efficient storage due to optimized binary encoding
  - Support compression
- Widely supported throughout the Hadoop ecosystem
  - Can also be used outside of Hadoop
- Ideal for long-term storage of important data
  - Many languages can read and write Avro files
  - Embeds schema in the file, so will always be readable
  - Schema evolution can accommodate changes
- **Verdict: Excellent interoperability and performance**
  - Best choice for general-purpose storage in Hadoop

Columnar formats

- Organize data storage on disk by column, rather than by row
  - Very efficient when selecting only a subset of a table’s column
  - Including Apache Parquet, RCFile, and ORCFile



Organization of data in traditional row-based formats

id	name	city	occupation
1	Alice	Palo Alto	Accountant
2	Bob	Sunnyvale	Accountant
3	Bob	Palo Alto	Dentist
4	Bob	Palo Alto	Manager
5	Carol	Palo Alto	Manager

Row-based storage to disk

1	Alice	Palo Alto	Accountant	2	Bob

Organization of data in columnar formats

id	name	city	occupation
1	Alice	Palo Alto	Accountant
2	Bob	Sunnyvale	Accountant
3	Bob	Palo Alto	Dentist
4	Bob	Palo Alto	Manager
5	Carol	Palo Alto	Manager

Column-based storage to disk

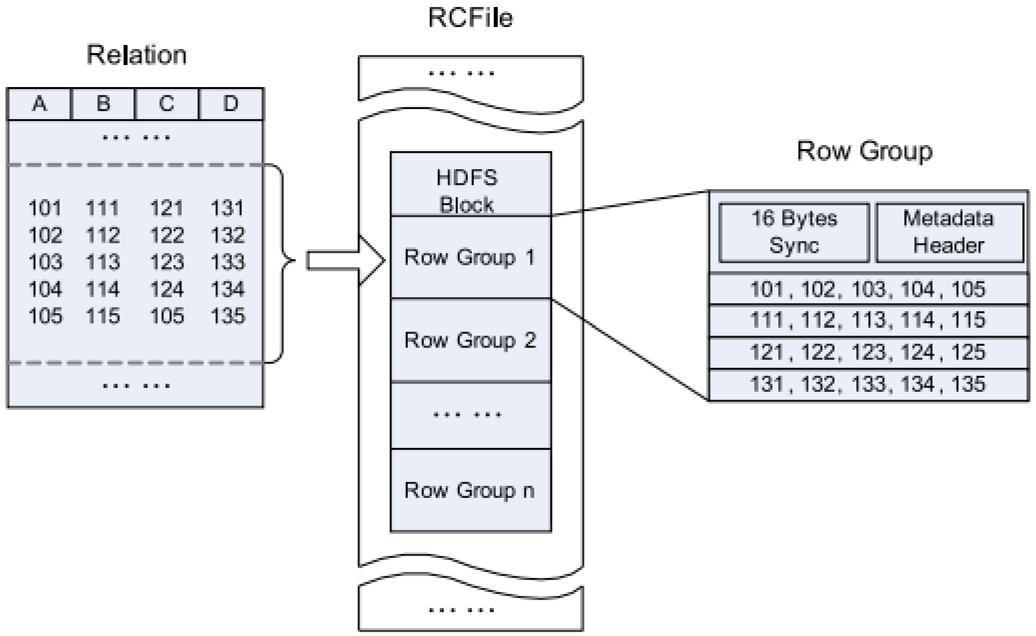
Alice	Bob	(×3)	Carol	…	Palo Alto

Columnar File Format: Apache **Parquet**

- Apache Parquet is an open source columnar format
  - Originally developed by engineers at Cloudera and Twitter
  - Now an Apache Software Foundation project
  - Supported in MapReduce, Hive, Pig, Impala, Spark, and others
  - Schema is embedded in the file
  - Stores **binary-encoded** records
- Uses advanced optimizations described in Google's Dremel paper
  - Reduces storage space
  - Increases performance
- Most efficient when adding many records at once
  - Some optimizations rely on identifying repeated patterns
- **Verdict: Excellent interoperability and performance**
  - Good choice for column-based access patterns

Columnar File Formats: RCFile and ORCFile

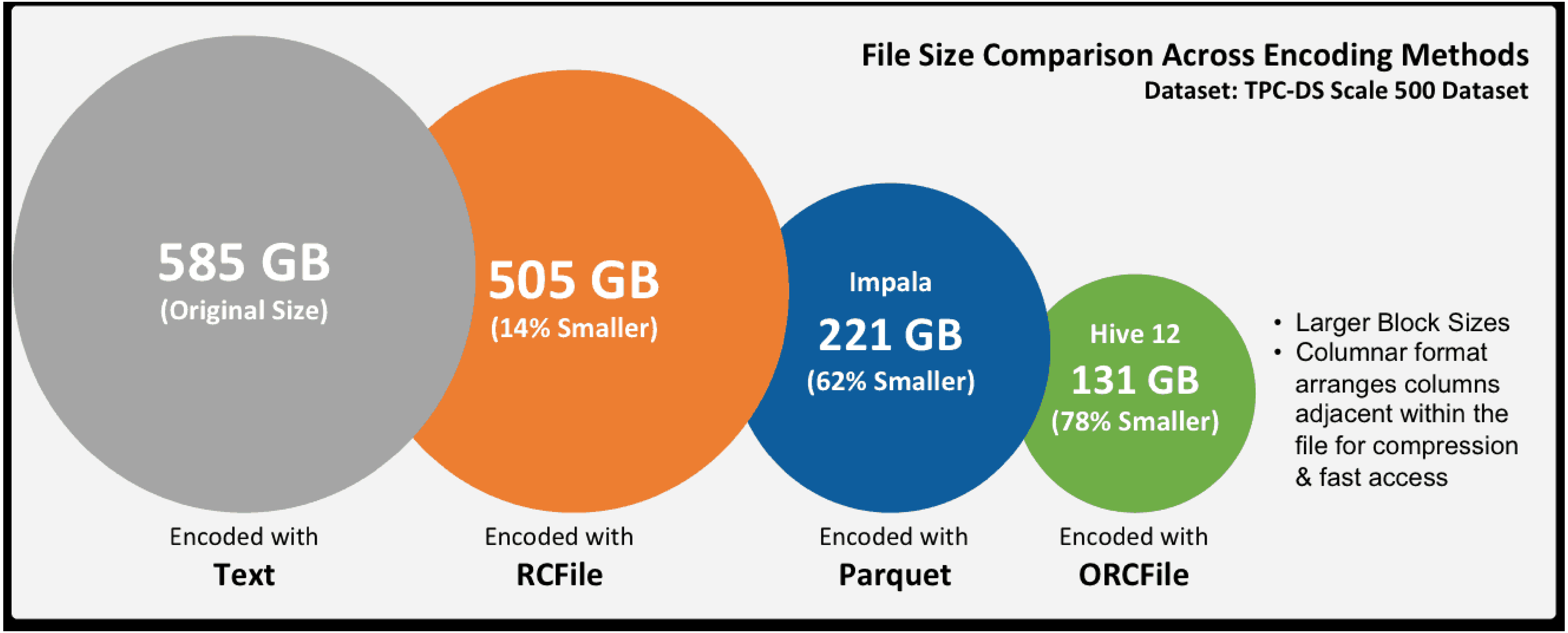
- RCFile
  - Splits data horizontally into row groups, each row group saves data in a columnar format
  - All data stored as strings (inefficient)
  - **Verdict: Poor performance and limited interoperability**
- ORCFile – [Optimized RCFile]
  - An improved version of RCFile
  - Currently supported in Hive, Spark, but limited in Impala
  - **Verdict: Good performance but limited interoperability**



Relation

A	B	C	D
… …			
101	111	121	131
102	112	122	132
103	113	123	133
104	114	124	134
105	115	105	135
… …			

HortonWorks (2013) Study of Storage Format and File Sizes



- Larger Block Sizes
- Columnar format arranges columns adjacent within the file for compression & fast access

https://hortonworks.com/blog/orcfile-in-hdp-2-better-compression-better-performance/

Convert between formats

- Load data in the original format, then use INSERT INTO TABLE to convert it into a new format.
- Create a new table stored in Parquet format

```
CREATE TABLE order_details_parquet (
order_id INT,
prod_id INT)
STORED AS PARQUET;
```

- Load data from another table into a Parquet table

```
INSERT OVERWRITE TABLE order_details_parquet
SELECT * FROM order_details;
```

Optimize Hive Query Performance

TABLE PARTITIONS

Motivation for Partitions

- When you perform lots of repetitive FILTER based queries and the FILTER conditions lend themselves to partitioning

```
SELECT event_type, COUNT(event_type)
FROM call_log
WHERE call_date = '2013-06-03'
GROUP BY event_type;
```

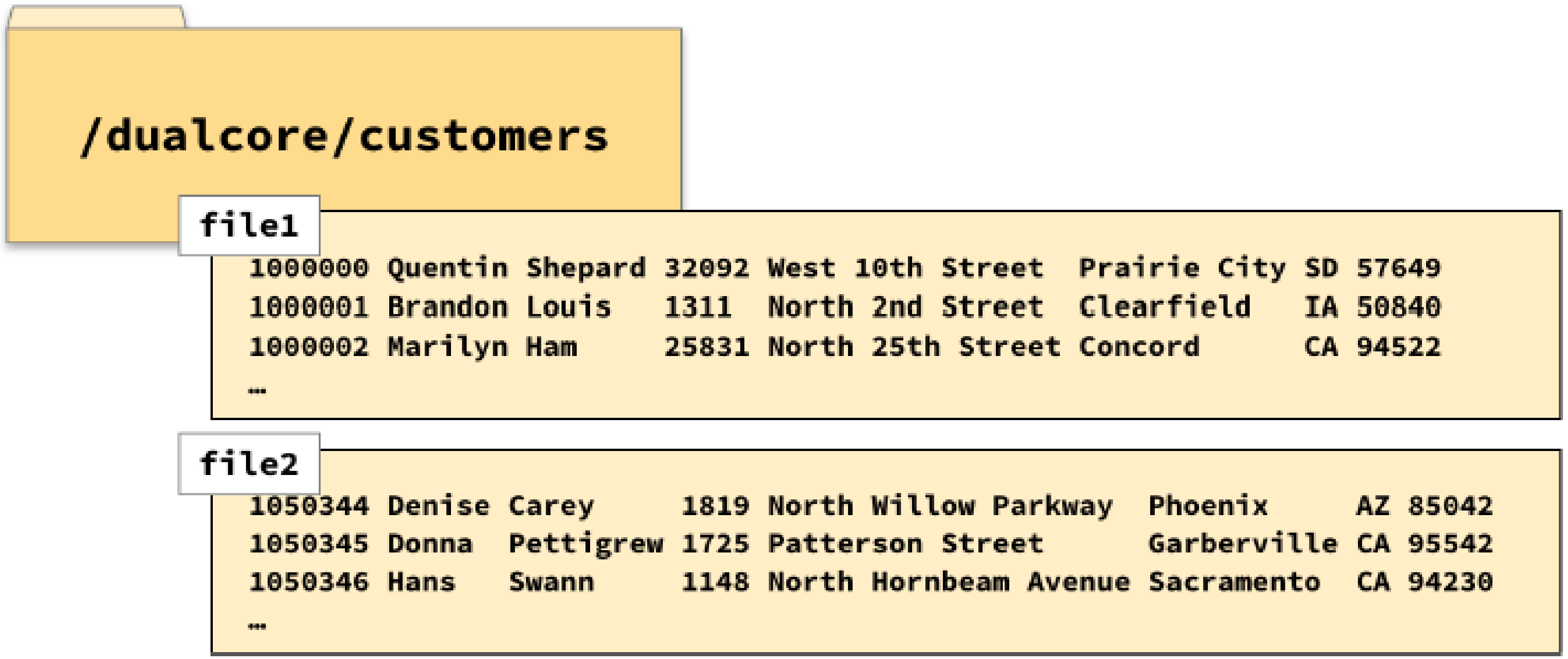
- It does not make sense to read and scan the whole dataset when you only need a small part of it.
  - An appropriate table partition allows Hive to fetch only a portion of the data.

Table Partitioning

- By default, all data files for a table are stored in a single directory
  - All files in the directory are read during a query
- Partitioning subdivides the data
  - Data is physically divided during loading, based on values from one or more columns
- Speeds up queries that filter on partition columns
  - Only the files containing the selected data need to be read
  - Does not prevent you from running queries that span multiple partitions

Example: Partitioning Customers by State

- `customers` is a non-partitioned table
- Data files are stored in a single directory
- All files are scanned for every query



/dualcore/customers

file1

```
1000000 Quentin Shepard 32092 West 10th Street   Prairie City SD 57649
1000001 Brandon Louis   1311  North 2nd Street   Clearfield   IA 50840
1000002 Marilyn Ham     25831 North 25th Street Concord      CA 94522
…
```

file2

```
1050344 Denise Carey     1819 North Willow Parkway  Phoenix     AZ 85042
1050345 Donna  Pettigrew 1725 Patterson Street      Garberville CA 95542
1050346 Hans   Swann     1148 North Hornbeam Avenue Sacramento  CA 94230
…
```

Creating a Partitioned Table

- Using PARTITIONED BY

```
CREATE EXTERNAL TABLE customers (
cust_id INT,
fname STRING,
lname STRING,
address STRING,
city STRING,
state STRING,
zipcode STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED
BY '\t'
LOCATION '/dualcore/customers';
```

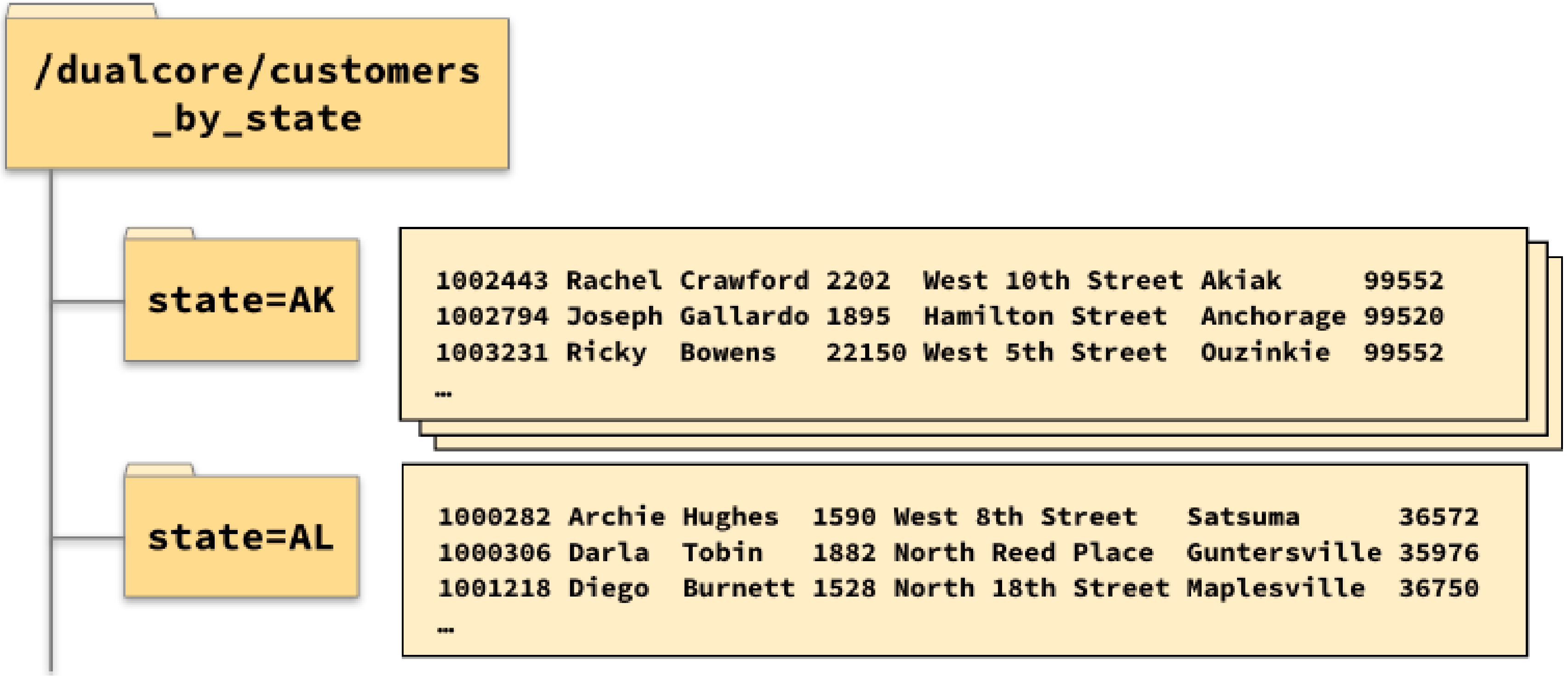
Non-partitioned

```
CREATE EXTERNAL TABLE customers_by_state(
cust_id INT,
fname STRING,
lname STRING,
address STRING,
city STRING,
zipcode STRING)
PARTITIONED BY (state STRING)
ROW FORMAT DELIMITED FIELDS TERMINATED BY
'\t'
LOCATION '/dualcore/customers';
```

Partitioned

Partitioning File Structure

- Partitioned tables store data in subdirectories
  - Queries that filter on partitioned fields limit amount of data read



```
1002443 Rachel Crawford 2202  West 10th Street Akiak     99552
1002794 Joseph Gallardo 1895  Hamilton Street  Anchorage 99520
1003231 Ricky  Bowens   22150 West 5th Street  Ouzinkie  99552
…
```

```
1000282 Archie Hughes  1590 West 8th Street   Satsuma      36572
1000306 Darla  Tobin   1882 North Reed Place  Guntersville 35976
1001218 Diego  Burnett 1528 North 18th Street Maplesville  36750
…
```

Partition fields

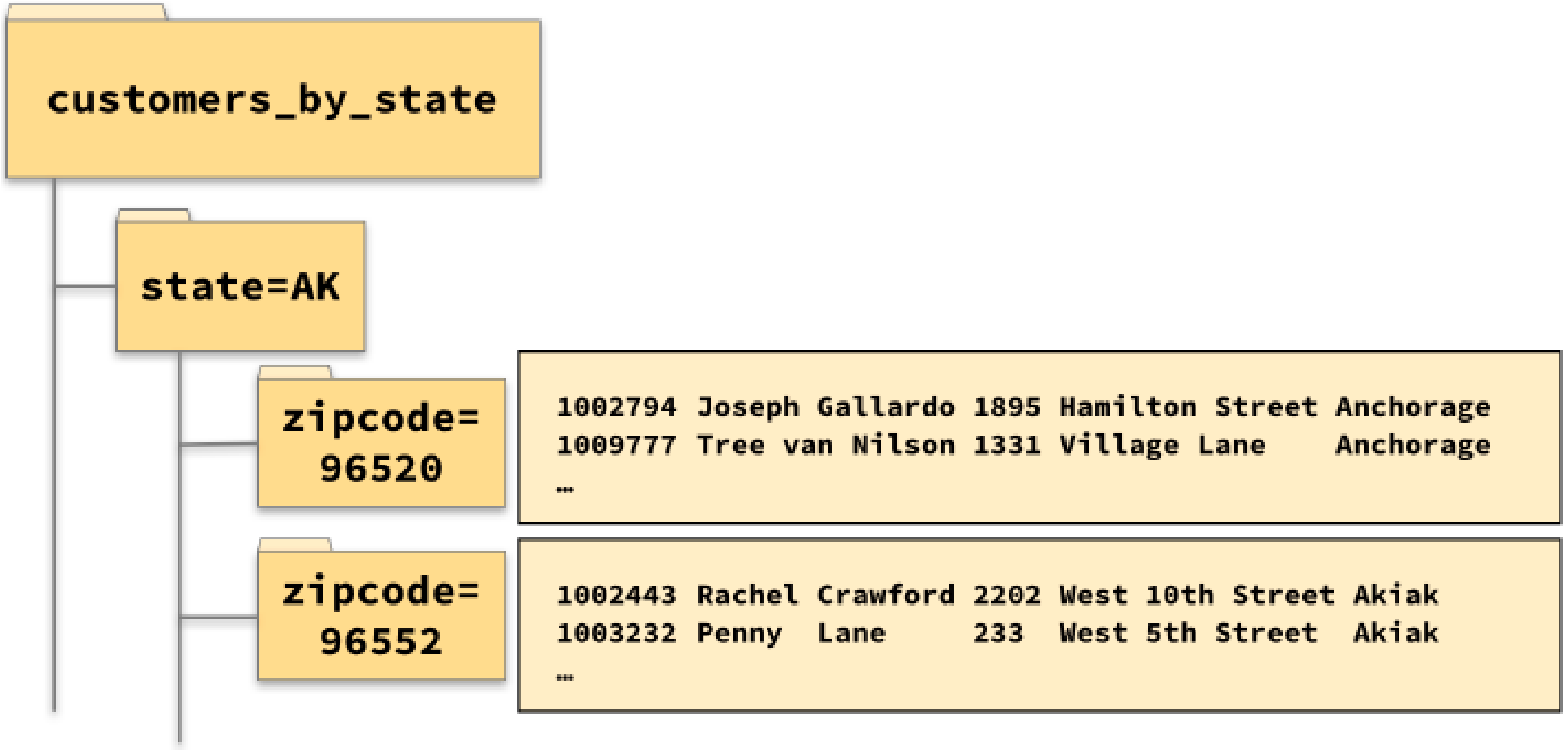
- The partition field(s) is removed from the table, since it is redundant, given the directory name in which the data is stored
- Each subdirectory is not limited to a single file
- A partition field is a *virtual field*. Column values are not stored in the files but the column is displayed if you DESCRIBE the table

```
DESCRIBE customers_by_state;
+-----------+---------+---------+
| name      | ty      | comment |
+-----------+---------+---------+
| cust_id   | int     |         |
| fname     | string  |         |
| lname     | string  |         |
| address   | string  |         |
| city      | string  |         |
| zipcode   | string  |         |
| state     | string  |         |
+-----------+---------+---------+
```

Nested Partitions

- You can also create nested partitions

```
… PARTITIONED BY (state STRING, zipcode STRING)
```



```
1002794 Joseph Gallardo 1895 Hamilton Street Anchorage
1009777 Tree van Nilson 1331 Village Lane     Anchorage
…
```

```
1002443 Rachel Crawford 2202 West 10th Street Akiak
1003232 Penny  Lane     233  West 5th Street  Akiak
…
```

When to Use Partitioning

- Use partitioning for tables when
  - Reading the entire dataset takes too long
  - Queries almost always filter on the partition columns
  - There are a reasonable number of different values for partition columns
  - Data generation or ETL process splits data by file or directory names
  - Partition column values are not in the data itself

When *Not* to Use Partitioning

- Avoid partitioning data into numerous small data files
  - Partitioning on columns with too many unique values
- Caution: This can happen easily when using dynamic partitioning!
  - For example, partitioning customers by first name could produce thousands of partitions

Loading Data into a Partitioned Table

- Dynamic partitioning
  - Hive/Impala automatically creates partitions
  - Inserted data is stored in the correct partitions based on column values
- Static partitioning
  - You manually create new partitions using `ADD PARTITION`
  - When loading data, you specify which partition to store it in

Static Partitioning

- With static partitioning, you create new partitions as needed

```
ALTER TABLE customers_by_state
ADD PARTITION (state='NY');
```

  - Adds the partition to the table’s metadata if it does not already exist.
  - Creates subdirectory `state=NY` in `/user/hive/warehouse/customers_by_state/`

- Then add data one partition at a time, e.g.

```
INSERT OVERWRITE TABLE customers_by_state
PARTITION(state='NY')
SELECT cust_id, fname, lname, address,
city, zipcode FROM customers WHERE state='NY';
```

Dynamic Partitioning

- With dynamic partitioning, you use an INSERT statement to load data
  - The partition column(s) must be included in the PARTITION clause
  - The partition column(s) must be specified last in the SELECT list

```
INSERT OVERWRITE TABLE customers_by_state
     PARTITION(state)
     SELECT cust_id, fname, lname, address, city,
          zipcode, state FROM customers
```

- Hive automatically creates partitions and inserts data into them based on the values of partition column(s)
  - If the partition does not already exist, it will be created
  - If the partition does exist, it will be overwritten

https://cwiki.apache.org/confluence/display/Hive/Tutorial#Tutorial-Dynamic-PartitionInsert

Dynamic Partition Inserts (2 Of 3)

- Dynamic partitioning is not enabled by default
  - Enable it by setting these two properties

Property Name	Value
`hive.exec.dynamic.partition`	true
`hive.exec.dynamic.partition.mode`	nonstrict

- Remember: Avoid creating an excessive number of partitions
  - This can happen when your data contains many unique values
- Caution: If the partition column has many different values, many partitions will be created
  - Partitioning by date is a popular, but common example

Viewing, Adding, And Removing Partitions

- To view the current partitions in a table

```
SHOW PARTITIONS call_logs;
```

- Use ALTER TABLE to add or drop partitions

```
ALTER TABLE call_logs
     ADD PARTITION (call_date='2018-06-05');
ALTER TABLE call_logs
     DROP PARTITION (call_date='2018-06-06');
```

  – Provides a very efficient way to drop data
    - Rather than filtering an entire table, Hive just deletes a subdirectory
  – Drops actual data whether or not it is a Hive-managed table.

Optimize Hive Query Performance

UNDERSTAND AND WRITE BETTER PERFORMING HIVE QUERIES

Write better Performing Hive Queries

- Generally speaking, Hive query performance improves if you can
  - Compress intermediate output
    - List columns you need, avoid `SELECT *`
  - Filter early in workflows consisting of several steps
    - e.g. if you can, use `WHERE` in the sub-query instead of in the main query
  - Write better queries that reduce the number of processing steps (e.g. avoid a costly join)

- This requires you to have a better understanding of how Hive executes a query.

Viewing The Execution Plan

- Prefix your query with EXPLAIN to view Hive’s execution plan

```
hive>  EXPLAIN SELECT zipcode, COUNT(cust_id) AS num
               FROM customers
              GROUP BY zipcode;
```

- The output of EXPLAIN can be very long and complex
  - Useful for determine, e.g. how many MapReduce phases it would require?
  - However, fully understanding it requires in-depth knowledge of MapReduce
  - We will cover the basics here

Viewing The Query Plan With EXPLAIN

- Query plan has stages, but stage numbers do not represent the execution sequence
  - They are merely identifiers
  - A root stage always runs first
  - Dependencies define the order of execution
- In the previous example, the stages run in the following order:
  - Stage-1 (MapReduce) runs first
  - Stage-0 (HDFS operation "ListSink") runs next
  - Hive is smart enough to run stages in parallel when there are no dependencies among them

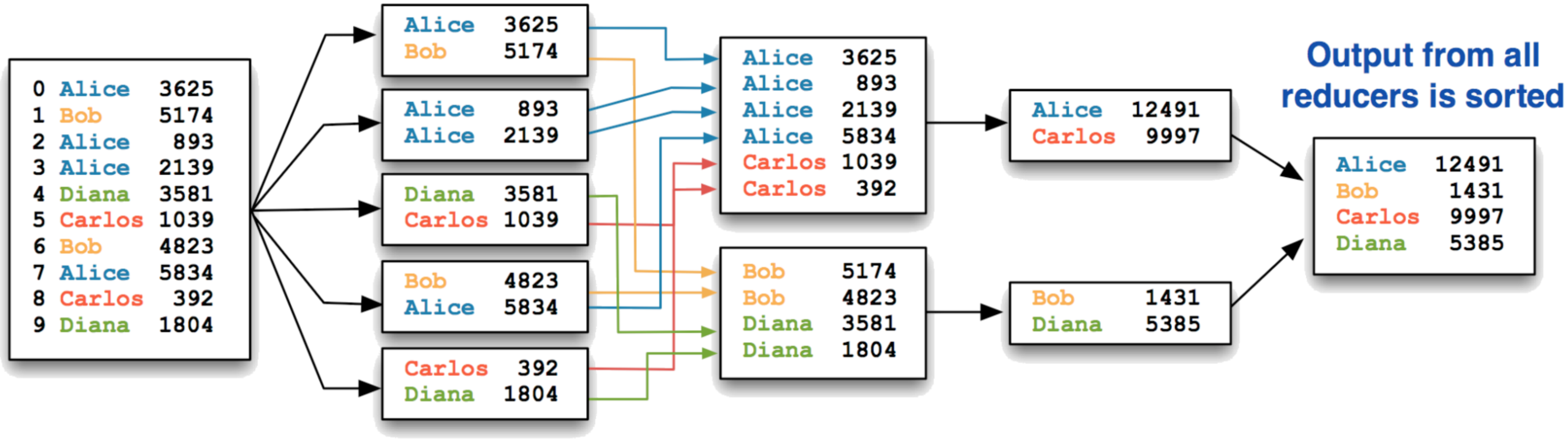
```
STAGE DEPENDENCIES:
  Stage-1 is a root stage
  Stage-0 depends on stages: Stage-1

STAGE PLANS:
  Stage: Stage-1
    Map Reduce
      Map Operator Tree:...
```

(for more details) https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Explain

Sorting Results

- As in SQL, ORDER BY sorts specified fields in HiveQL
  - Consider the result from the following query, utilizing 5 Mappers and 2 Reducers

```
hive> SELECT name, SUM(total)
            FROM order_info
            GROUP BY name
            ORDER BY name;
```



```
0 Alice   3625
1 Bob     5174
2 Alice    893
3 Alice   2139
4 Diana   3581
5 Carlos  1039
6 Bob     4823
7 Alice   5834
8 Carlos   392
9 Diana   1804
```

```
Alice   3625
Bob     5174
```

```
Alice    893
Alice   2139
```

```
Diana   3581
Carlos  1039
```

```
Bob     4823
Alice   5834
```

```
Carlos   392
Diana   1804
```

```
Alice   3625
Alice    893
Alice   2139
Alice   5834
Carlos  1039
Carlos   392
```

```
Bob     5174
Bob     4823
Diana   3581
Diana   1804
```

```
Alice   12491
Carlos   9997
```

```
Bob      1431
Diana    5385
```

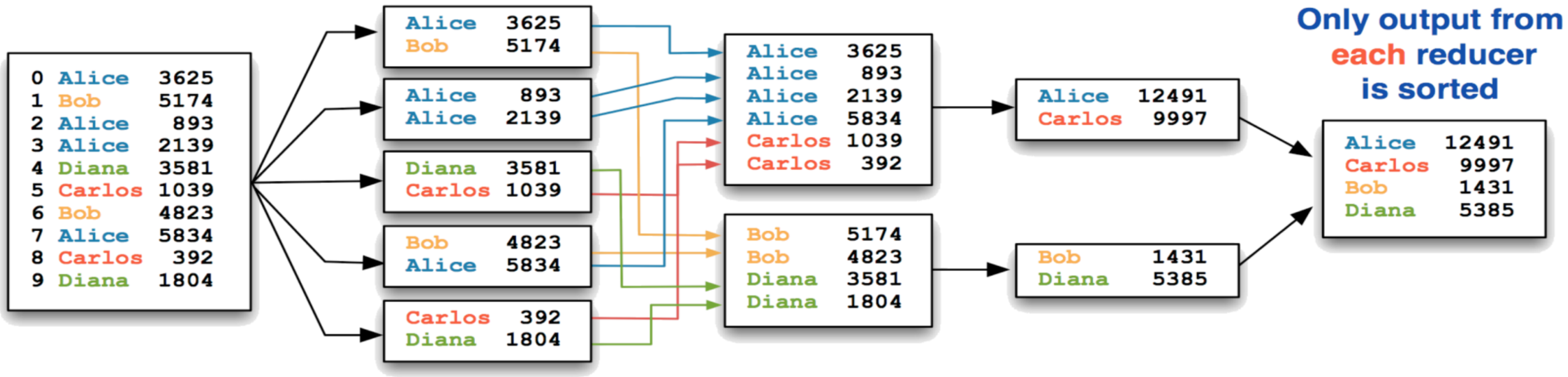
```
Alice   12491
Bob      1431
Carlos   9997
Diana    5385
```

Output from all reducers is sorted

Using SORT BY For Partial Ordering (1 Of 2)

- HiveQL also supports partial ordering via SORT BY
  - Offers much better performance if global order isn't required
    - e.g. You want ordering within each Reducer, but not across multiple Reducers

```
hive>SELECT name, SUM(total)
           FROM order_info
           GROUP BY name
           SORT BY name;
```



```
0 Alice   3625
1 Bob     5174
2 Alice    893
3 Alice   2139
4 Diana   3581
5 Carlos  1039
6 Bob     4823
7 Alice   5834
8 Carlos   392
9 Diana   1804
```

```
Alice   3625
Bob     5174
```

```
Alice    893
Alice   2139
```

```
Diana   3581
Carlos  1039
```

```
Bob     4823
Alice   5834
```

```
Carlos   392
Diana   1804
```

```
Alice   3625
Alice    893
Alice   2139
Alice   5834
Carlos  1039
Carlos   392
```

```
Bob     5174
Bob     4823
Diana   3581
Diana   1804
```

```
Alice   12491
Carlos   9997
```

```
Bob      1431
Diana    5385
```

```
Alice   12491
Carlos   9997
Bob      1431
Diana    5385
```

Only output from each reducer is sorted

Using SORT BY For Partial Ordering (2 Of 2)

- Behind the scene
  - ORDER BY uses a single reducer to ensure global sorting
  - SORT BY sorts the input before feeding them into reducers
    - So that output of each reducer is sorted (partial sorting), but collective output is not.
- SORT BY can improve performance on large queries in 2 ways:
  - Bringing the benefits of parallelism to the reduce phase
  - In some case, it can eliminate the need for a second MapReduce job dedicated to global ordering of results.

- https://cwiki.apache.org/confluence/display/Hive/LanguageManual+SortBy

Optimize Hive Query Performance

BUCKETING

Bucketing Data in Hive

- *Bucketing* data is another way of subdividing data
  - Stores data in separate files
  - Divides data into buckets in **an effectively random** way
  - Calculates **hash codes** based on column values
  - Use hash codes to assign records to a bucket
- Goal: Distribute rows across a predefined number of buckets
  - Useful for jobs that need samples of data.
  - Joins may be faster if all tables are bucketed on the join column

Example of Bucketing

- Each bucket should contain roughly 5% of the table’s data

```
CREATE TABLE orders_bucketed  (order_id INT,
    cust_id INT,  order_date TIMESTAMP)
  CLUSTERED BY (order_id) INTO 20 BUCKETS;

-- enforce bucketing when inserting data
SET hive.enforce.bucketing=true;
INSERT OVERWRITE TABLE orders_bucketed
   SELECT * FROM orders;

-- sample 10% of data
SELECT * FROM orders_bucketed
  TABLESAMPLE (BUCKET 1 OUT OF 10 ON order_id);
```

Essential Points

- Tez/Spark engine can lead to superior performance than the MR engine.
- ORC/PARQUET are significantly faster than the text format.
- Partitioning may reduce the amount of data a query must read
- Understanding Hive execution using the Explain command.
- SORT BY (partial ordering) is faster than ORDER BY
- Bucketing may also improve performance of certain types of queries