Understanding Map Tasks and Reduce Tasks

If you've been using the Hive engine, you may have noticed that some types of queries are significantly slower than others. Hive is designed to hide the complexity of distributed data processing from the user. To use Hive, you need only issue SQL queries. But to understand why some types of queries finish faster than others, you need to know what happens when you run a Hive query.

Hive Query Process

Hive does not have its own data processing engine; instead, it converts a query into one or more jobs that run on the cluster using a different engine. Originally, MapReduce was the exclusive data processing engine for Hive. Newer versions of Hive include support for Apache Spark or Apache Tez as an alternative engine. When you use MapReduce as Hive's execution engine, this is called *Hive on MapReduce*. There are similar terms for Spark and Tez. In this reading, and as the default execution engine in the course VM, we're using Hive on MapReduce.

A Hive client application, such as Beeline or Hue, connects to a Hive server. When you run a query from one of these clients, the Hive server performs several operations. It parses the SQL, retrieves metadata from the metastore, and plans the execution of the query. These steps are relatively fast; they might take only a small fraction of a second for a simple query. See the first column, "Steps Run by Hive Server," of Figure 1.

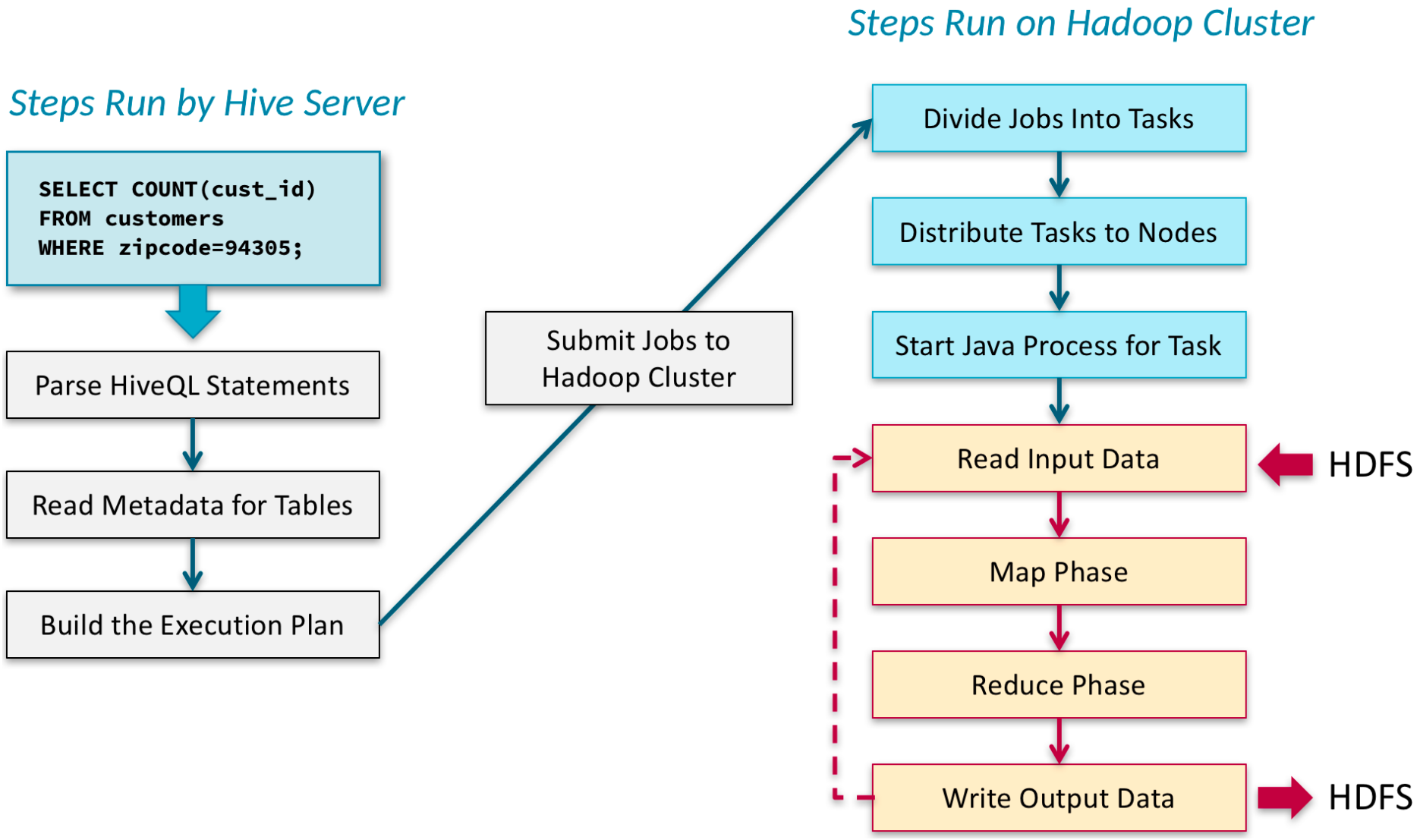

Figure 1: Hive process overview

Then the Hive server submits one or more jobs to the cluster. On the cluster, the jobs are divided into tasks, the tasks are distributed across nodes, and processes are started to execute the tasks (“Steps Run on Hadoop Cluster” in Figure 1). Tasks are executed in a specific sequence. First the input data is read, then the data is processed through one or more map and reduce phases, and finally the result is generated. The steps that run on the cluster account for a large majority of a query’s total running time.

Map and Reduce Tasks

To understand how a job on the cluster is divided up into tasks, you need to understand how the map-reduce data processing model works. The MapReduce engine used by Hive is an implementation of the map-reduce data processing model.

This model provides a way to divide a large data processing job into a sequence of smaller tasks that can run in parallel across a large number of computers. A MapReduce job is divided into two types of tasks: map tasks and reduce tasks. These tasks are sequenced in phases.

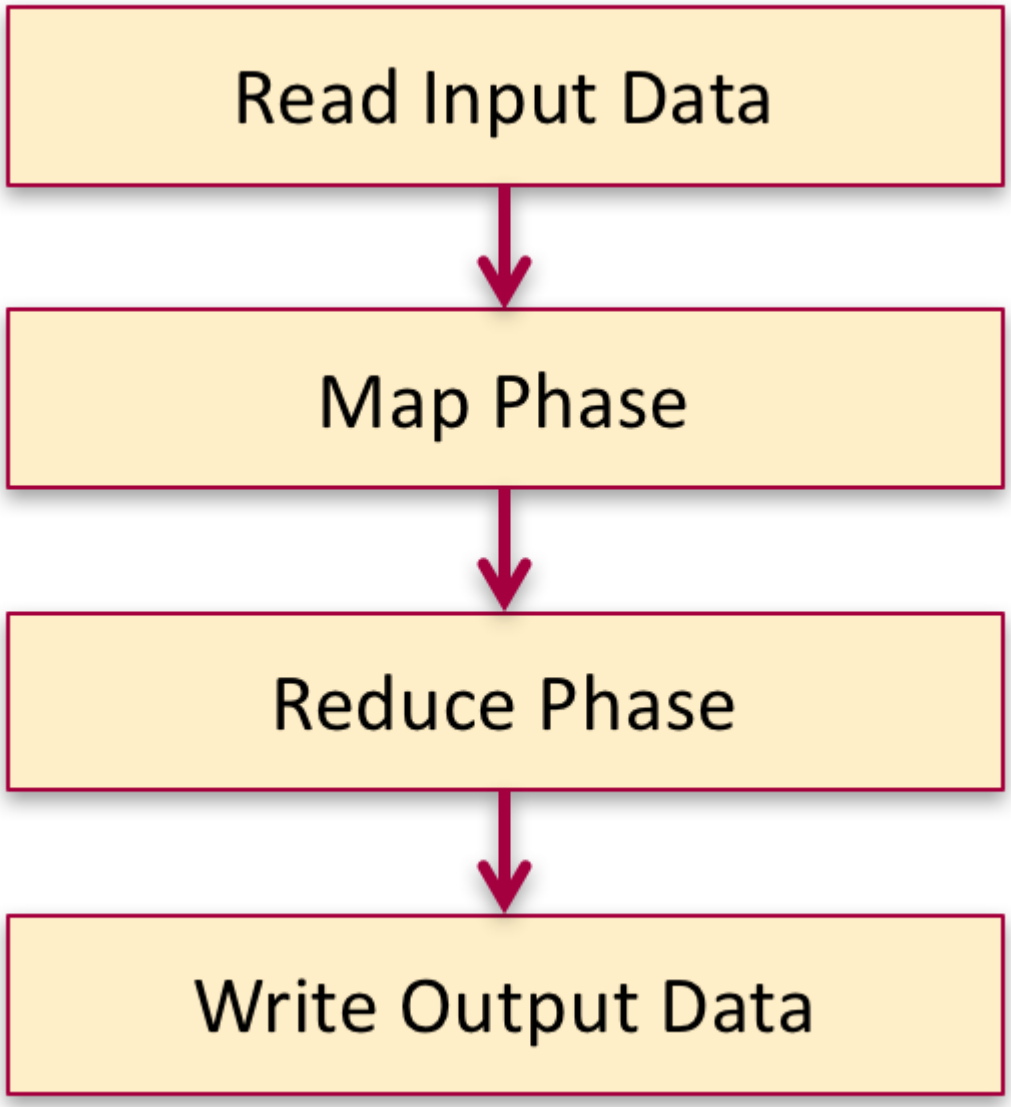
Read Input Data
Map Phase
Reduce Phase
Write Output Data

Figure 2: Map-reduce job within the Hive process

A map phase runs first. It is used to filter, transform, or parse data. In a map phase, each record of data is processed independently. The output from a map phase becomes the input to a reduce phase. A reduce phase is used to summarize or aggregate data, combining multiple records together. Some types of jobs don’t perform any aggregation so they don’t require a reduce phase; these are called map-only jobs.

Example MapReduce Job

This example illustrates how a Hive query executes as a MapReduce job. The query in this example selects data from a table named order_info. This table has three columns representing the order ID, the name of the salesperson, and the order amount.

```
SELECT upper(sales_rep), SUM(amount) AS high_sales
    FROM order_info
    WHERE amount > 1000
    GROUP BY upper(sales_rep);
```

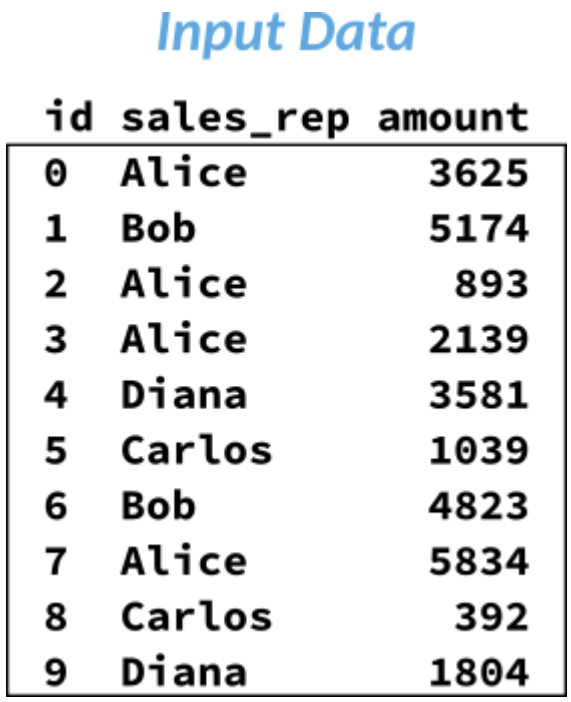

Figure 3: Example input data

Notice that salespeople can have multiple orders. The query groups by the sales_rep column, adjusted for case sensitivity, and calculates the sum of the order amounts for each salesperson. Executing this query requires both a map phase and a reduce phase.

In the map phase of the MapReduce job, the individual map tasks each receive a portion of the input data. The number of map tasks is determined primarily by the total size of the input data. The example in Figure 4 shows five parallel map tasks, but with a very large input dataset, there could be hundreds or thousands.

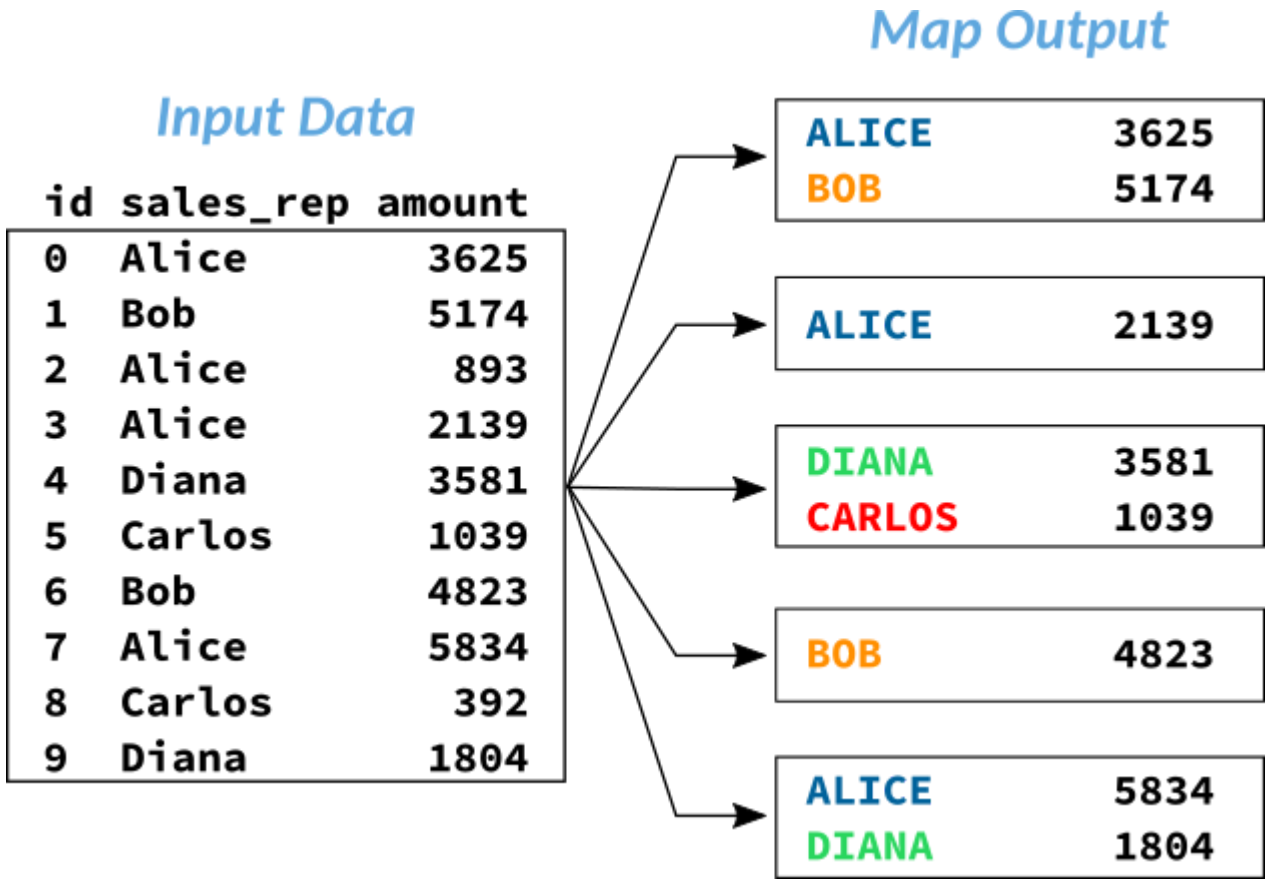

Figure 4: Map output

The map tasks process the input records. They may filter, transform, or parse the input data. The map tasks can also project the input data, which means returning only a subset of the columns. That's what the map tasks do in this example; each map task simply reads a portion of the input data and outputs the sales_rep and amount fields, discarding the order ID field because it's not needed.

The output from the map tasks goes through an intermediate process called *shuffle and sort*. This process merges together the output from all the map tasks to create the input to the reduce phase, one input dataset for each reduce task. (See Figure 5.) The process also sorts the data by the column or columns that the data is grouped by, which in this example is the sales_rep column. Notice here that the records for Alice are grouped together, the records for Carlos are grouped together, and so on. But the result is not globally ordered—notice here that Carlos comes before Bob.

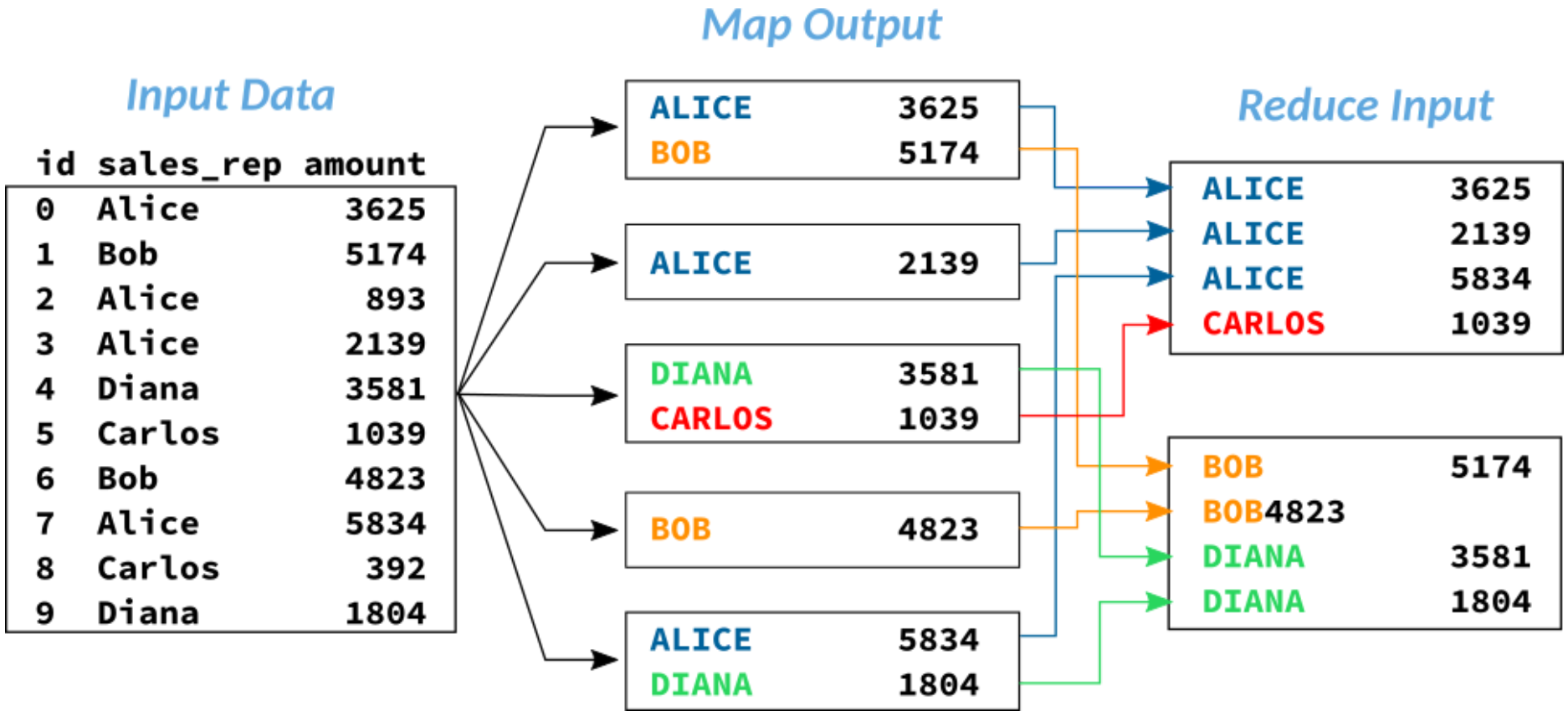

Figure 5: Shuffle and sort

The reduce tasks are where aggregation is performed; in this example, they compute the sum of the order amounts for each salesperson. The number of reduce tasks is determined by the configuration of Hive or MapReduce, and it's almost always much smaller than the number of map tasks. This example (Figure 6) shows two reduce tasks. The outputs from the reduce tasks are appended together to produce the query result.

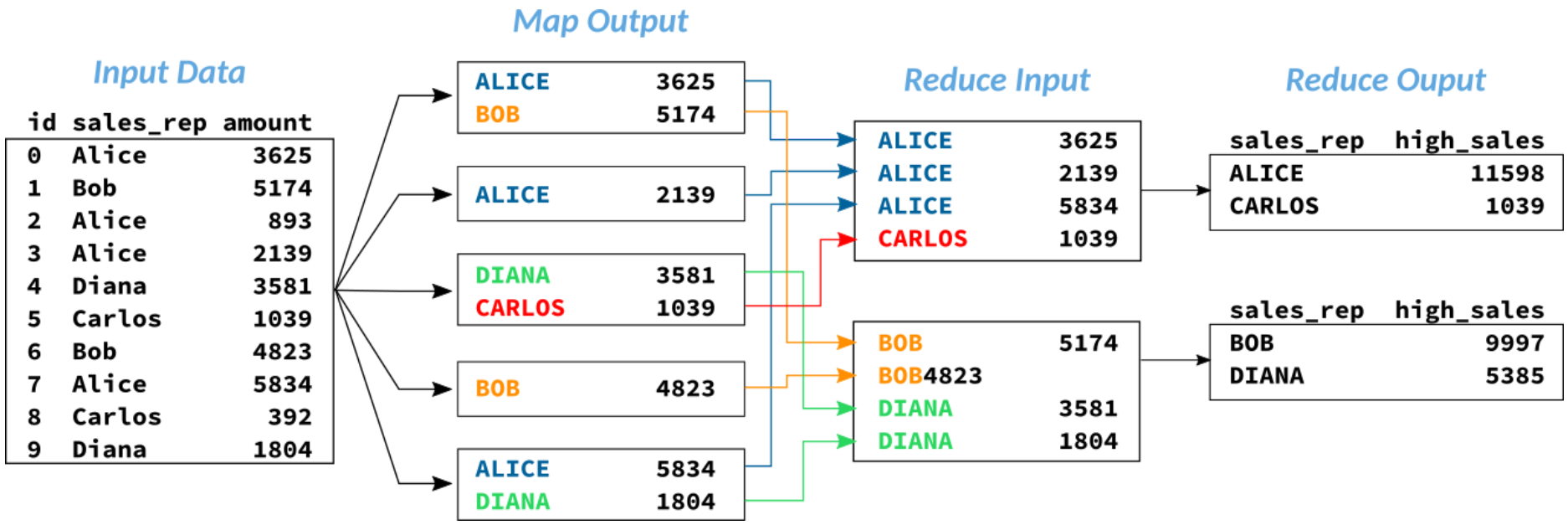

Figure 6: Reduce output