Hadoop and SAS Integration

SAS Foundation Interfaces for Hadoop

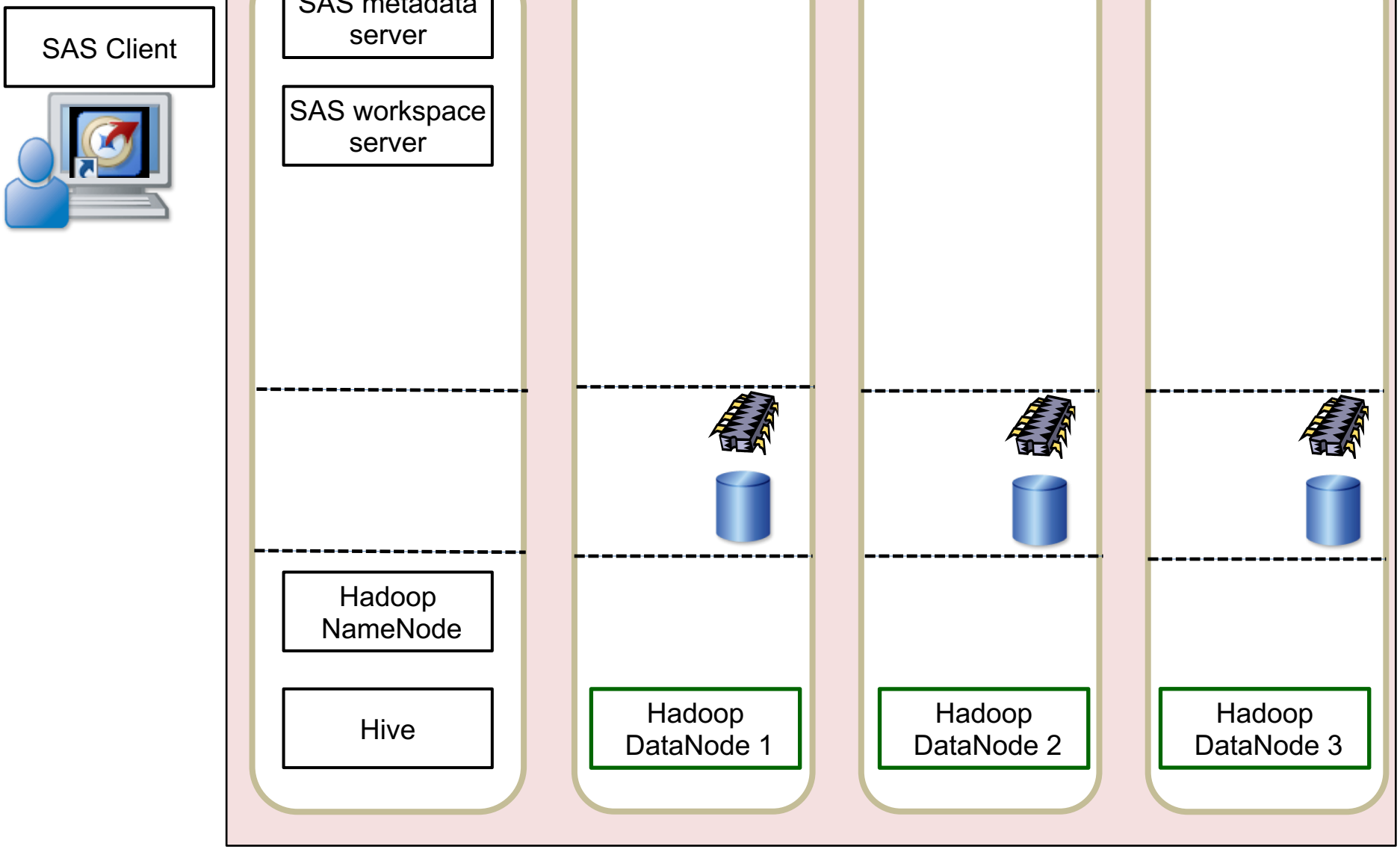
Tool	Purpose	Product
FILENAME statement	Allow the DATA step to read and write HDFS data files.	Base SAS
PROC HADOOP	Copy or move files between SAS and Hadoop. Execute MapReduce and Pig code. Execute Hadoop file system commands to manage files and directories.	Base SAS
SQL Pass-Through	Submit HiveQL queries and other HiveQL statements from SAS directly to Hive for Hive processing. Query results are returned to SAS.	SAS/ACCESS Interface to Hadoop
LIBNAME Statement For Hadoop	Access Hive tables as SAS data sets using the SAS programming language. SAS/ACCESS engine translates SAS language into HiveQL and attempts to convert the processing into HiveQL before returning results to SAS.	SAS/ACCESS Interface to Hadoop

SAS In-Memory Analytics Interfaces for Hadoop

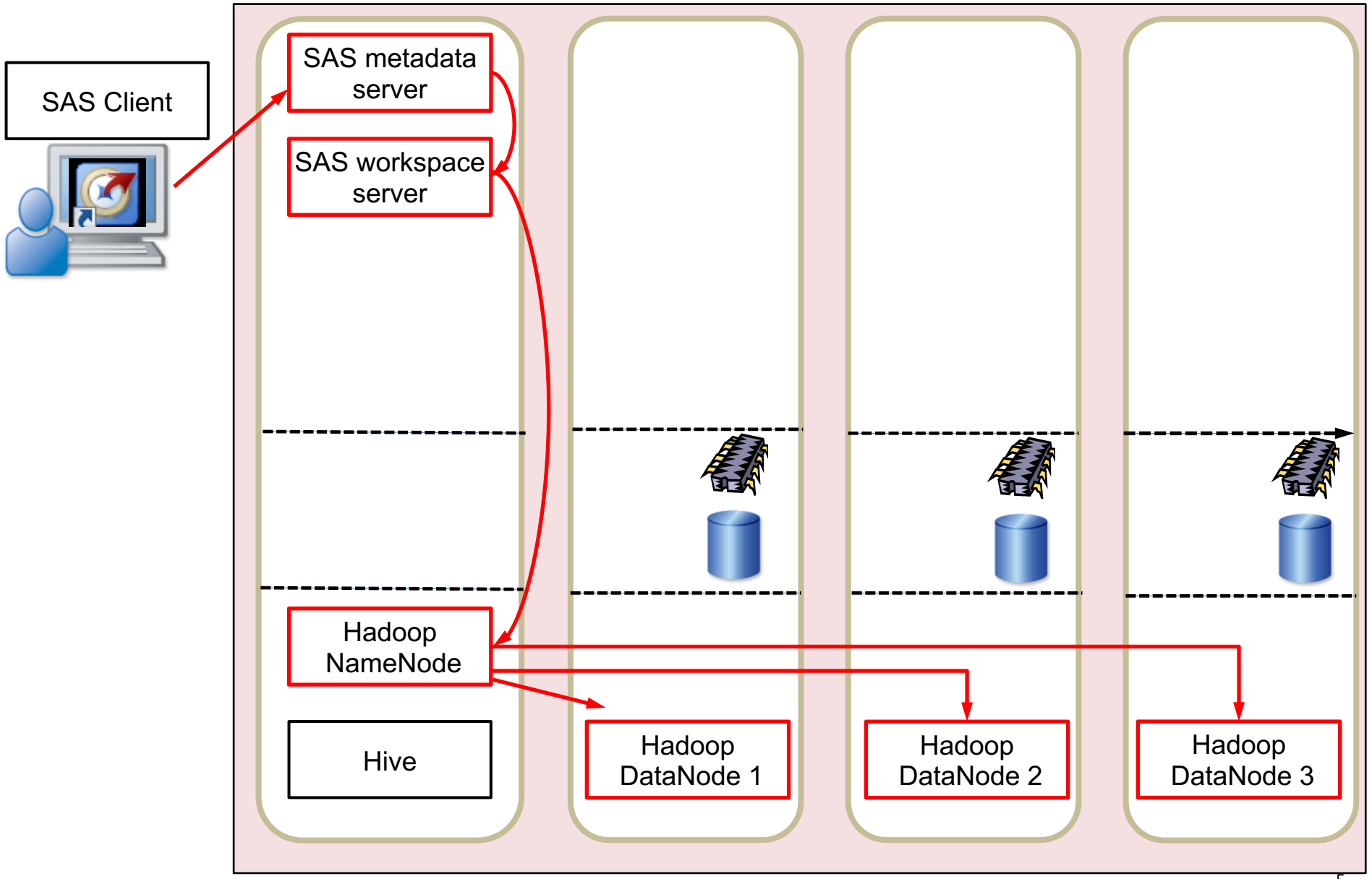
Hadoop is now one of the file storage systems that SAS uses for SAS In-Memory Analytics product solutions:

- SAS High-Performance Analytics products
- SAS Visual Analytics
- SAS In-Memory Statistics
- SAS Code Accelerator for Hadoop (DS2)

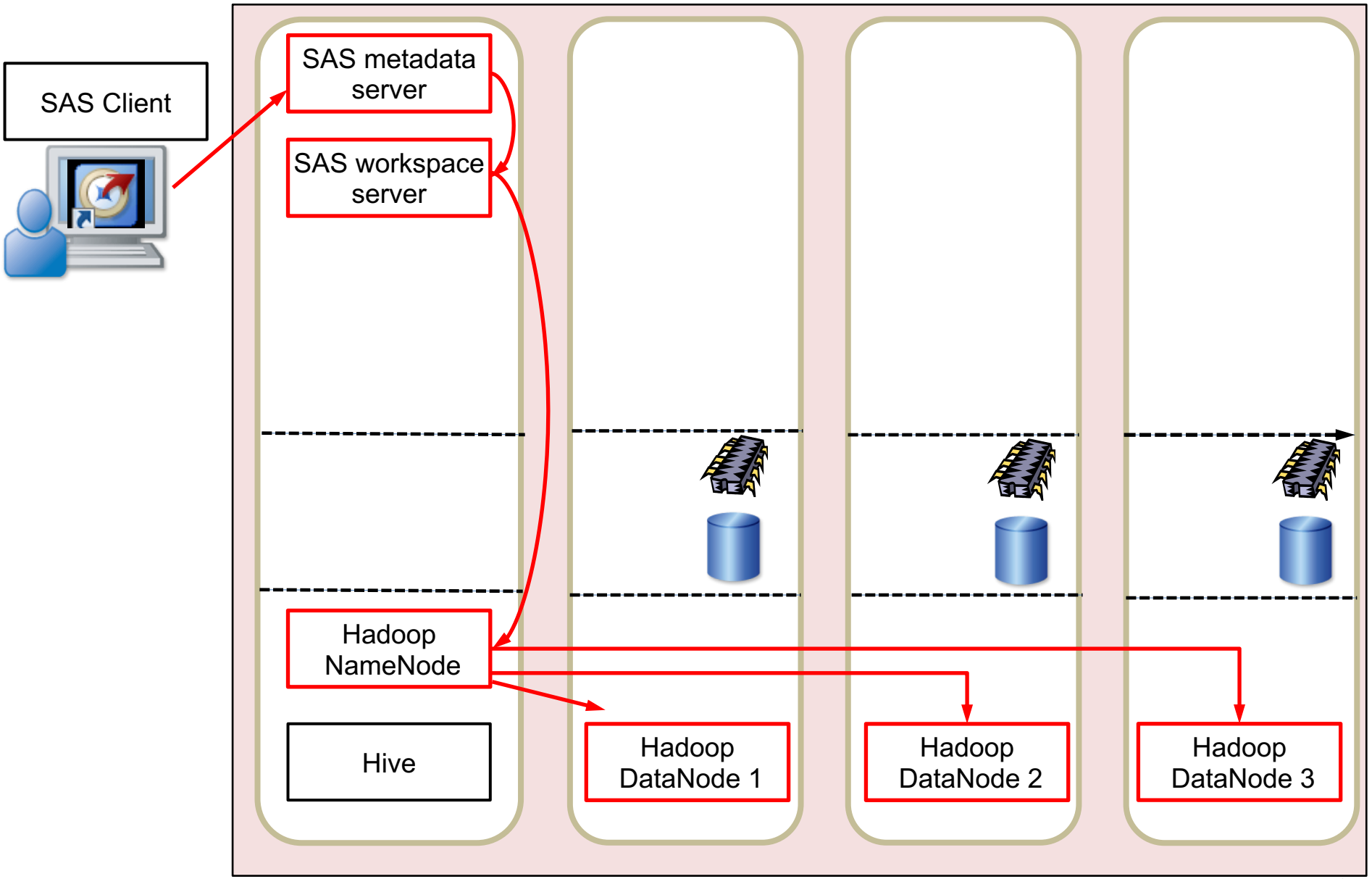
A Hadoop Cluster to Run SAS



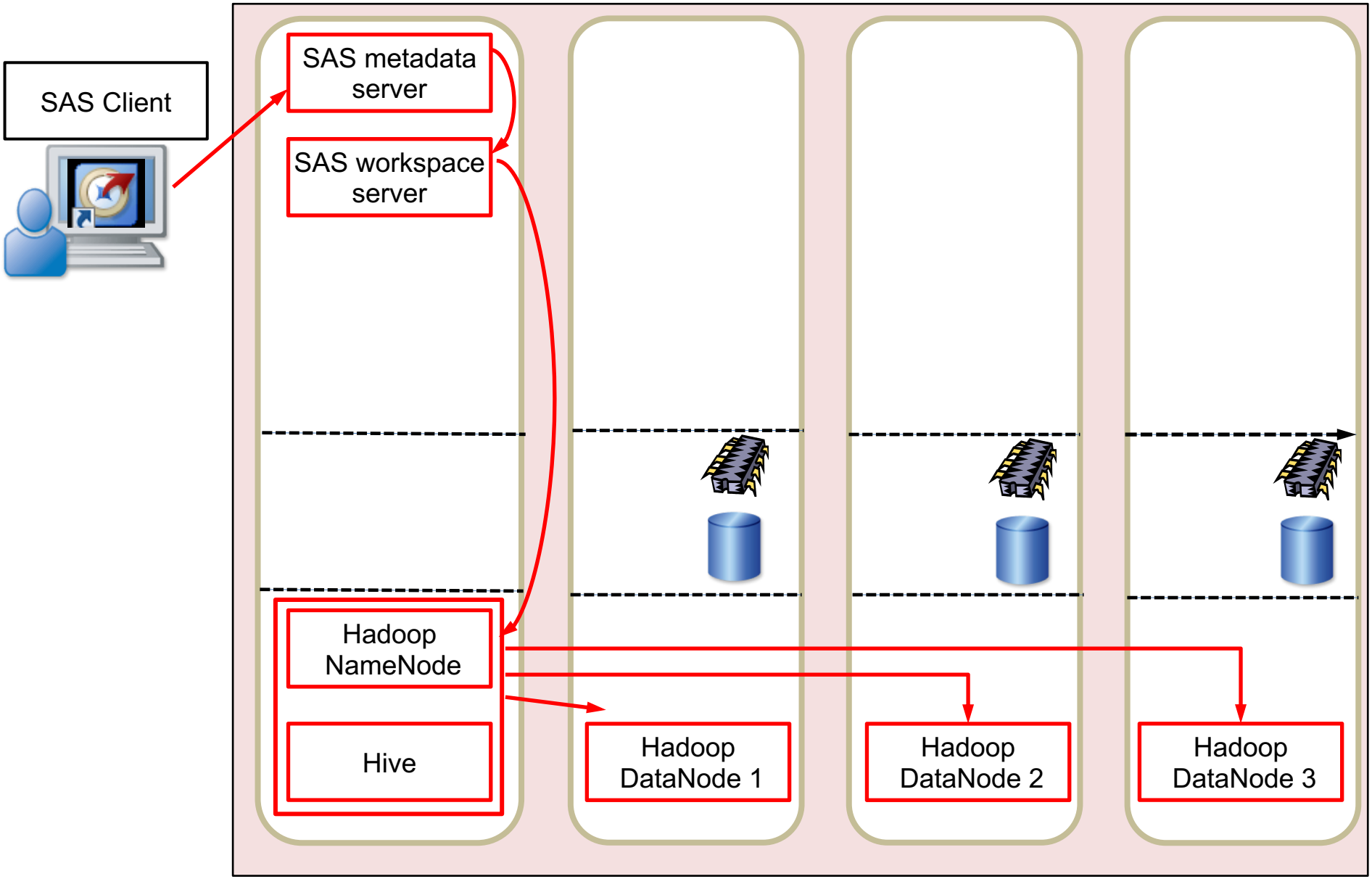
Base SAS: FILENAME for Hadoop



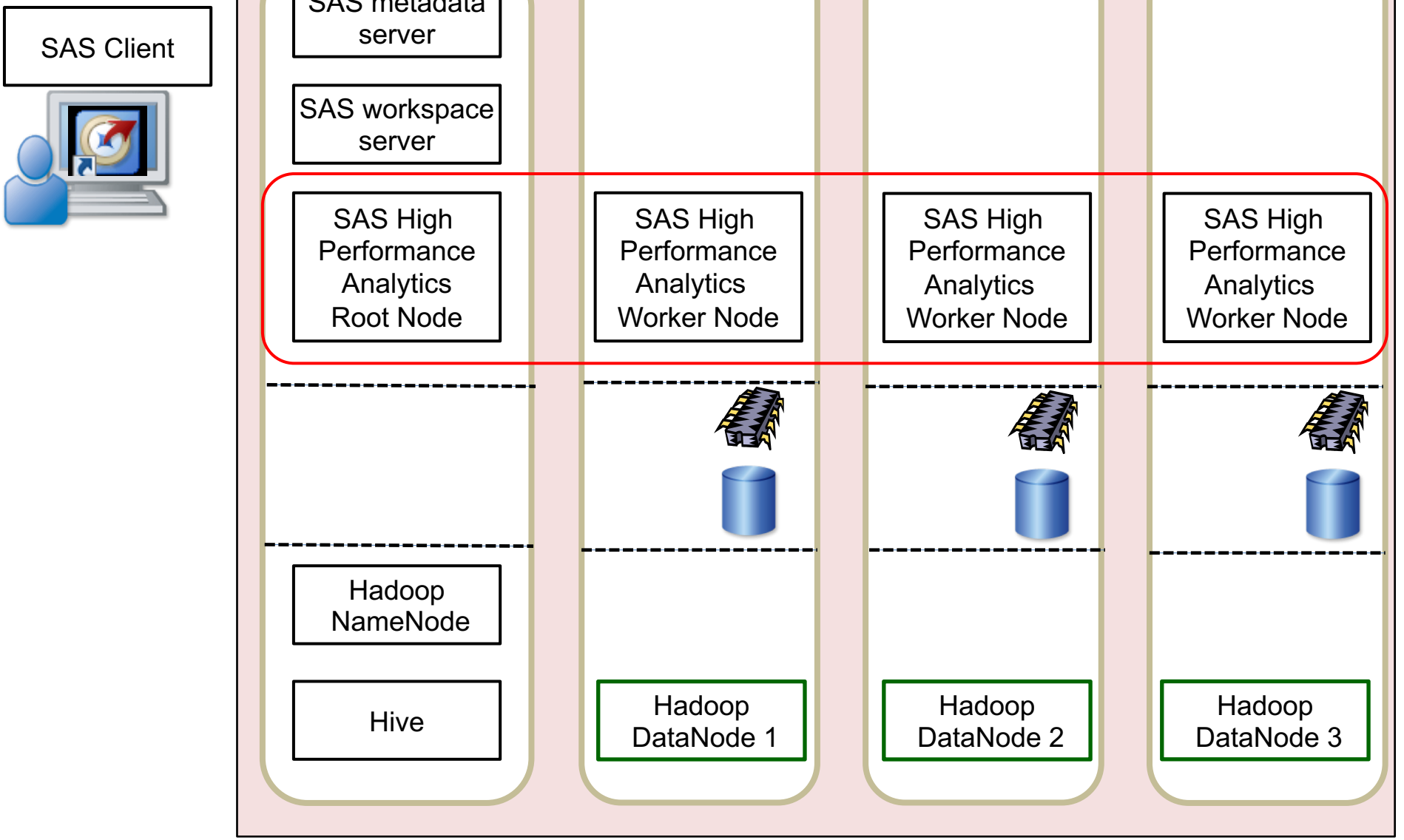
Base SAS: The Same Process for PROC HADOOP



SAS/ACCESS: SQL Pass-Through and LIBNAME

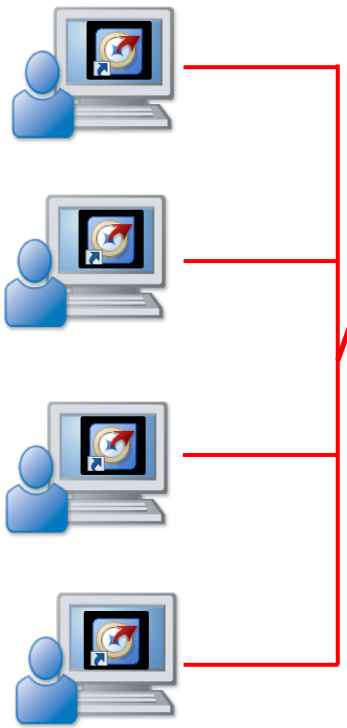


In-Memory Analytics



Computing Environment

MS WINDOWS CLIENT: e.g., individual machine for each student



LINUX SERVER: Shared SAS Hadoop system with individual accounts in Linux, HDFS, and Hive

SAS metadata server

SAS workspace server

SAS High Performance Analytics Root Node

Hadoop NameNode

Hive

SAS High Performance Analytics Worker Node

SAS High Performance Analytics Worker Node

SAS High Performance Analytics Worker Node

Hadoop DataNode 1

Hadoop DataNode 2

Hadoop DataNode 3

Managing Files and Executing Hadoop Commands, Map-Reduce, and Pig

- Introduction to Base SAS Methods for Hadoop
- The HADOOP FILENAME Statement and PROC HADOOP
- Executing Pig Code With PROC HADOOP

Introduction to Base SAS Methods for Hadoop

Base SAS Tools for Hadoop

- The Hadoop FILENAME statement enables you to do the following:
  - upload local data to Hadoop using the DATA step
  - read data from Hadoop using the DATA step

- PROC HADOOP enables you to do the following:
  - submit Hadoop file system (HDFS) commands
  - submit MapReduce programs
  - submit Pig script

The Hadoop Config.xml File

The FILENAME statement for Hadoop and PROC HADOOP require an option that specifies a Hadoop configuration file (config.xml).

- The configuration file defines how to connect to Hadoop.
- The file must be accessible to the SAS client application.
- A SAS administrator commonly manages this configuration for the SAS users.
- This file is often referred to as the Hadoop core-site.xml file.

Hadoop Config.xml File

hadoop_config.xml 〉 No Selection

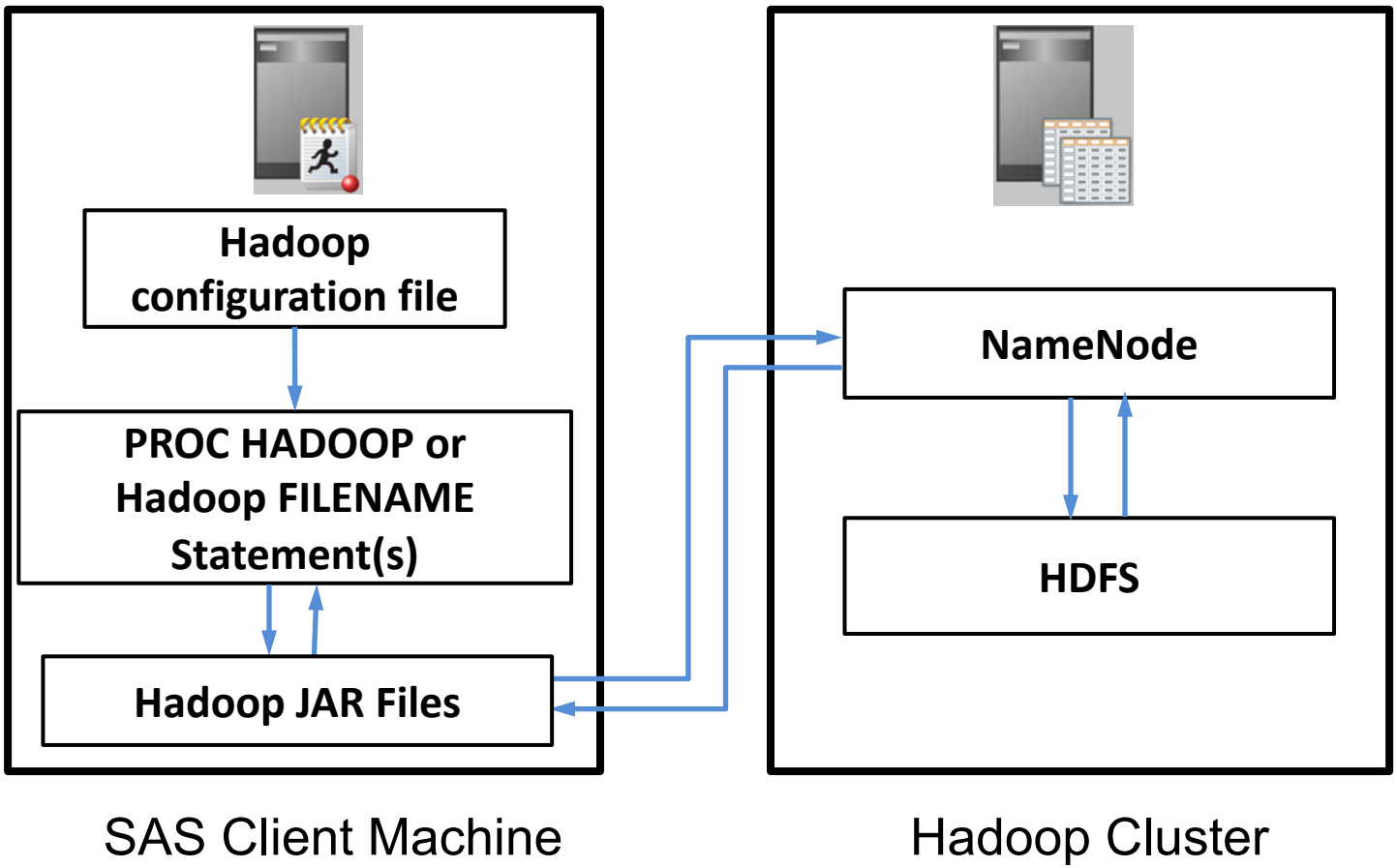
```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
<property>
    <name>fs.default.name</name>
    <value>hdfs://sasserver.demo.sas.com:8020</value>
</property>
<property>
    <name>mapred.job.tracker</name>
    <value>sasserver.demo.sas.com:8021</value>
</property>
<property>
    <name>dfs.http.address</name>
    <value>sasserver.demo.sas.com:50070</value>
</property>
<property>
    <name>dfs.secondary.http.address</name>
    <value>sasserver.demo.sas.com:50090</value>
</property>
<property>
    <name>mapred.job.tracker.http.address</name>
    <value>sasserver.demo.sas.com:50030</value>
</property>
</configuration>
```

Hadoop JAR Files

A collection of Hadoop JAR files is also required on the SAS client machine.

- An **environment variable** SAS_HADOOP_JAR_PATH on the SAS client machine defines the location of the Hadoop JAR files
- The Hadoop JAR files must be compatible with the specific Hadoop implementation and *can be copied from the Hadoop server*.
- A Hadoop system administrator commonly manages the configuration of the Hadoop JAR files for the SAS users.

Base SAS Interface to Hadoop



The HADOOP FILENAME Statement and PROC HADOOP

Business Scenario

We want to develop a prototype for a process that uses SAS to orchestrate the following scenario:

1. Move unstructured text files into the Hadoop file system.
2. Invoke MapReduce programs developed by Java programmers in order to:
   - read and process the text files to perform various analyses
   - output results as text files in the Hadoop file system
3. Read the summarized text analysis results back into SAS for further analysis and reporting purposes.

continued...

Business Scenario

We want to develop a prototype for a process that uses SAS to orchestrate the following scenario:

1. Move unstructured text files into the Hadoop file system.
2. Invoke MapReduce programs developed by Java programmers in order to:
   - read and process the text files to perform various analyses
   - output results as text files in the Hadoop file system.
3. Read the summarized text analysis results back into SAS for further analysis and reporting purposes.

PROC HADOOP and the HDFS COPYFROMLOCAL statement

continued …

Business Scenario

We want to develop a prototype for a process that uses SAS to orchestrate the following scenario:

1. Move unstructured text files into the Hadoop file system.
2. Invoke MapReduce programs developed by Java programmers in order to:
   - read and process the text files to perform various analyses (example: word counts)
   - output results as text files in the Hadoop file system.
3. Read the summarized text analysis results back into SAS for further analysis and reporting purposes.

PROC HADOOP and the MAPREDUCE statement

Business Scenario

- We want to develop a prototype for a process that uses SAS to orchestrate the following scenario:
  1. Move unstructured text files into the Hadoop file system.
  2. Invoke MapReduce programs developed by Java programmers in order to:
     - read and process the text files to perform various analyses (example: word counts)
     - output results as text files in the Hadoop file system.
  3. Read the summarized text analysis results back into SAS for further analysis and reporting purposes.

FILENAME statement for Hadoop and DATA step

Business Scenario Pseudocode

```
proc hadoop…;
  hdfs copyfromlocal='local file'
  out='hdfs file';
run;

proc hadoop…;
  mapreduce input='hdfs file'
            output='hdfs outfile'
            …;
run;

filename fileref 'hdfs outfile'…;
data somedata;
 infile fileref
 input …;
 …
run;
```

1. Move unstructured text files into the Hadoop file system
2. Input the hdfs file to MapReduce program and output results to hdfs file
3. Read the MapReduce output with SAS for further processing

PROC HADOOP

- PROC HADOOP submits:
  - Hadoop file system (HDFS) commands
  - MapReduce programs
  - PIG language code

```
PROC HADOOP <Hadoop-server-option(s)>;
     HDFS <Hadoop-server-option(s)> <hdfs-command-option(s)>;
     MAPREDUCE <Hadoop-server-option(s)> <mapreduce-option(s)>;
     PIG <Hadoop-server-option(s)> <pig-code-option(s)>;
     PROPERTIES <configuration-properties>;
RUN;
```

HDFS Statements

HDFS COPYFROMLOCAL='*local-file*' OUT='*output-location*'
<DELETESOURCE> < OVERWRITE>;

HDFS COPYTOLOCAL='*HDFS-file*' OUT='*output-location*'
<DELETESOURCE> < OVERWRITE> < KEEPCRC>;

HDFS DELETE='*HDFS-file*' <NOWARN>;

HDFS MKDIR='*HDFS-path*';

HDFS RENAME='*HDFS-file*' OUT='*new-name*';

Moving a File from SAS to Hadoop

This program creates a directory in the Hadoop file system (HDFS) and copies a file from the SAS server to the new HDFS directory.

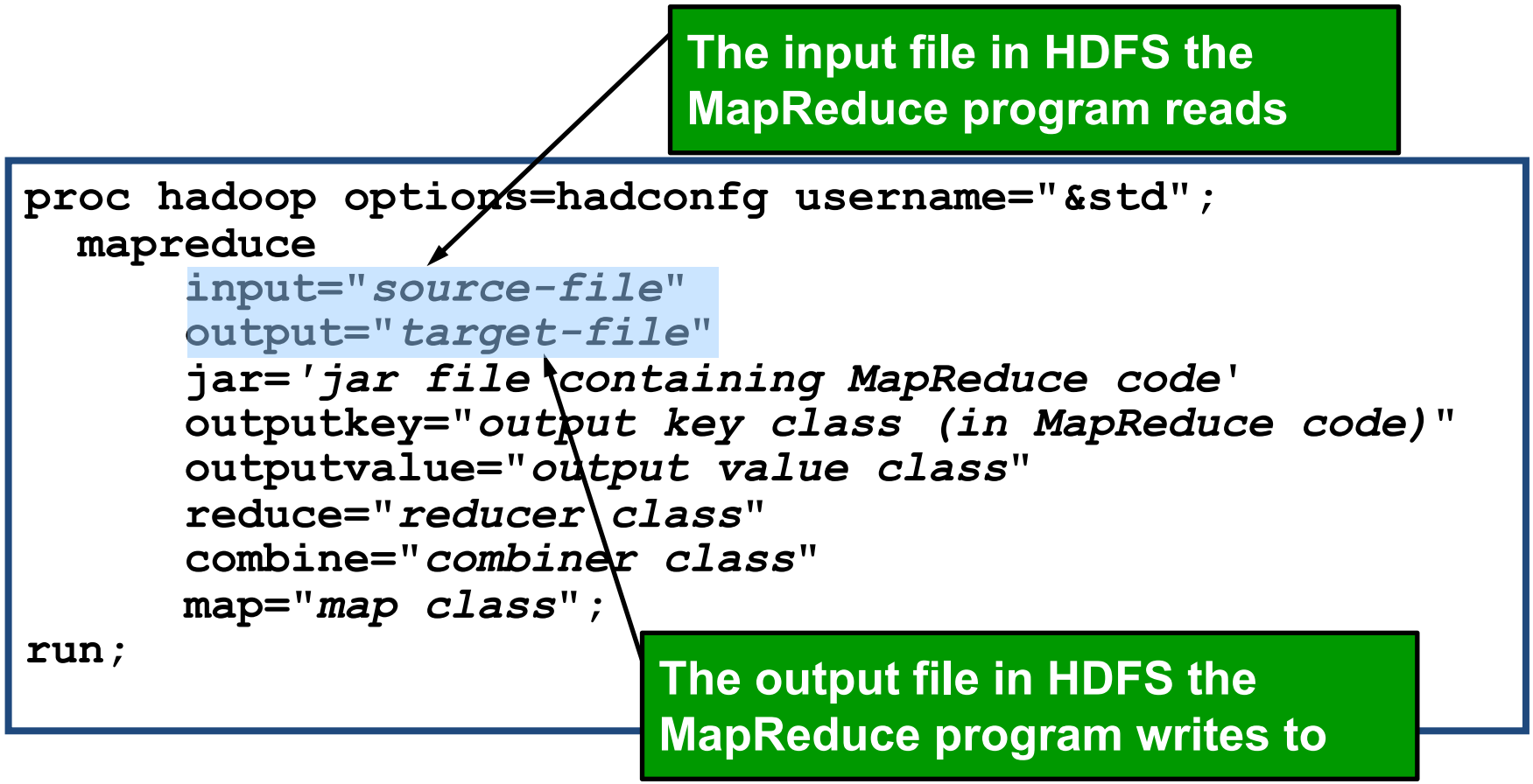
```
filename hadconfg '/work/hadoop_config.xml';
proc hadoop options=hadconfg username="&std"
                verbose;
    hdfs mkdir="/user/&std/data";
    hdfs copyfromlocal =
       '/work/DIACCHAD/data/moby_dick_via_sas.txt'
        out="/user/&std/data";
run;
```

Execute MapReduce Code

MAPREDUCE <*Hadoop-server-option(s)*> <*mapreduce-option(s)*>;

```
proc hadoop options=hadconfg username="&std";
  mapreduce
       input="source-file"
       output="target-file"
       jar='jar file containing MapReduce code'
       outputkey="output key class (in MapReduce code)"
       outputvalue="output value class"
       reduce="reducer class"
       combine="combiner class"
       map="map class";
run;
```

Execute MapReduce Code



Execute MapReduce Code

The JAR file containing the MapReduce program and named classes

```
proc hadoop options=hadconfg username="&std";
  mapreduce
       input="source-file"
       output="target-file"
       jar='jar file containing MapReduce code'
       outputkey="output key class (in MapReduce code)"
       outputvalue="output value class"
       reduce="reducer class"
       combine="combiner class"
       map="map class";
run;
```

Execute MapReduce Code

```
proc hadoop options=h                                    ;
  mapreduce
       input="source-file"
       output="target-file"
       jar='jar file containing MapReduce code'
       outputkey="output key class (in MapReduce code)"
       outputvalue="output value class"
       reduce="reducer class"
       combine="combiner class"
       map="map class";
run;
```

The name of the output key class in dot notation

The name of the output value class in dot notation

Execute MapReduce Code

The Java classes in the map reduce program that execute the map, reduce and combine steps

```
proc hadoop options=hadconfg username="&std";
  mapreduce
       input="source-file"
       output="target-file"
       jar='jar file containing MapReduce code'
       outputkey="output key class (in MapReduce code)"
       outputvalue="output value class"
       reduce="reducer class"
       combine="combiner class"
       map="map class";
run;
```

MapReduce Example

In the demonstration, PROC HADOOP will be used to invoke a MapReduce program that will do the following:

- read a text file containing free unstructured text. This file can be distributed in multiple data nodes in Hadoop
- parse the text into the individual words in each data node
- count up the number of instances of each unique word found in each data node
- combine total counts for each unique word across nodes
- write the results to an HDFS output file

MapReduce Example

```
proc hadoop options=hadconfg username="&std" verbose;
   mapreduce
       jar=          "<hdfs path>/hadoop-mr1-cdh.jar"

       input=        "<hdfs path>/moby_dick_via_sas.txt"

       map=          "org.apache.hadoop.examples.
                      WordCount$TokenizerMapper"

       reduce=       "org.apache.hadoop.examples.
                      WordCount$IntSumReducer"

       combine=      "org.apache.hadoop.examples.
                      WordCount$IntSumReducer"

       outputkey=    "org.apache.hadoop.io.Text"
       outputvalue= "org.apache.hadoop.io.IntWritable"

       output=       "<hdfs path>/mapoutput"
   ;
run;
```

MapReduce Example

```
proc hadoop options=hadconfg username="&std" verbose;
   mapreduce
      jar=          "<hdfs path>/hadoop-examples-2.0.0-
                     mr1-cdh4.4.0.jar"

      input=        "<hdfs path>/moby_dick_via_sas.txt"

      map=          "org.apache.hadoop.examples.
                     WordCount$TokenizerMapper"

      reduce=       "org.
                     Wor

      combine=      "org.apache.hadoop.examples.
                     WordCount$IntSumReducer"

      outputkey=    "org.apache.hadoop.io.Text"
      outputvalue=  "org.apache.hadoop.io.IntWritable"

      output=       "<hdfs path>/mapoutput"
   ;
run;
```

1. Read a text file containing free unstructured text

MapReduce Example

```
proc hadoop options=hadconfg username="&std" verbose;
   mapreduce
      jar=          "<hdfs path>/hadoop-examples-2.0.0-
                     mr1-cdh4.4.0.jar"

      input=        "<hdfs path>/moby_dick_via_sas.txt"

      map=          "org.apache.hadoop.examples.
                     WordCount$TokenizerMapper"

      reduce=       "org.apache.hadoop.examples.
                     Wo

      combine=      "org
                     Wo

      outputkey=    "org.apache.hadoop.io.Text"
      outputvalue= "org.apache.hadoop.io.IntWritable"

      output=       "<hdfs path>/mapoutput"
   ;
run;
```

2. In parallel in each data node, parse the text into the individual words

MapReduce Example

```
proc hadoop options=hadconfg username="&std" verbose;
   mapreduce
      jar=          "<hdfs path>/hadoop-examples-2.0.0-
                     mr1-cdh4.4.0.jar"

      input=        "<hdf

      map=          "org
                     Word

      reduce=       "org.apache.hadoop.examples.
                     WordCount$IntSumReducer"

      combine=      "org.apache.hadoop.examples.
                     WordCount$IntSumReducer"

      outputkey=    "org.apache.hadoop.io.Text"
      outputvalue=  "org.apache.hadoop.io.IntWritable"

      output=       "<hdfs path>/mapoutput"
   ;
run;
```

3. In each data node, in parallel, count up the number of instances of each unique word found

MapReduce Example

```
proc hadoop options=hadconfg username="&std" verbose;
   mapreduce
      jar=          "<hdfs path>/hadoop-examples-2.0.0-
                     mr1-cdh4.4.0.jar"

      input=        "<hdfs path>/moby_dick_via_sas.txt"

      map=          "org.
                     WordCount$TokenizerMapper"

      reduce=       "org.
                     WordCount$IntSumReducer"

      combine=      "org.apache.hadoop.examples.
                     WordCount$IntSumReducer"

      outputkey=    "org.apache.hadoop.io.Text"
      outputvalue=  "org.apache.hadoop.io.IntWritable"

      output=       "<hdfs path>/mapoutput"
   ;
run;
```

4. Combine the counts for each unique word across data nodes to find final counts

MapReduce Example

```
proc hadoop options=hadconfg username="&std" verbose;
   mapreduce
      jar=          "<hdfs path>/hadoop-examples-2.0.0-
                     mr1-cdh4.4.0.jar"

      input=        "<hdfs path>/moby_dick_via_sas.txt"

      map=          "org.apache.hadoop.examples.

      reduce=       "

      combine=      "
                     WordCount$IntSumReducer"

      outputkey=    "org.apache.hadoop.io.Text"
      outputvalue= "org.apache.hadoop.io.IntWritable"

      output=       "<hdfs path>/mapoutput"
   ;
run;
```

5. The output contains each unique word (outputkey) as Text and the number of times it occurred in the input file as Integer

MapReduce Example

```
proc hadoop options=hadconfg username="&std" verbose;
   mapreduce
      jar=          "<hdfs path>/hadoop-examples-2.0.0-
                     mr1-cdh4.4.0.jar"

      input=        "<hdfs path>/moby_dick_via_sas.txt"

      map=          "org.apache.hadoop.examples.
                     WordCount$TokenizerMapper"

      reduce=       "org.apache.hadoop.examples.
                     WordCount$IntSumReducer"

      combine=      "org.apach
                     WordCoun

      outputkey=    "org.apach
      outputvalue= "org.apache.hadoop.io.IntWritable"

      output=       "<hdfs path>/mapoutput"
   ;
run;
```

5. The HDFS location the output is written to

The FILENAME Statement for Hadoop

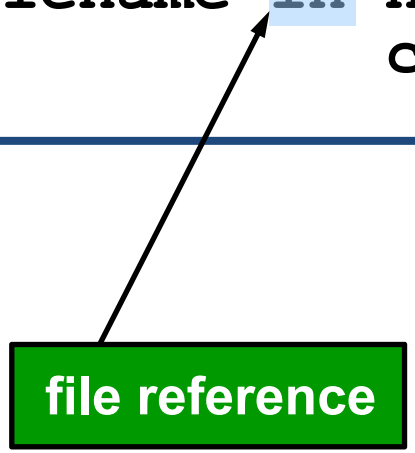
In SAS, the FILENAME statement associates a fileref with an external file and an output device or access method.

FILENAME *fileref <device type or access method> 'external file' <options>*;

The FILENAME Statement for Hadoop

FILENAME *fileref <device type or access method> 'external file' <options>*;

```
filename in hadoop "Hadoop-file-path" concat
           cfg=xml-config-file user="&std";
```



The FILENAME Statement for Hadoop

FILENAME *fileref <device type or access method> 'external file' <options>;*

```
filename in hadoop "Hadoop-file-path" concat
            cfg=xml-config-file user="&std";
```

The FILENAME Statement for Hadoop

FILENAME *fileref* <*device type or access method*> '*external file*' <*options*>;

```
filename in hadoop "Hadoop-file-path" concat
               cfg=xml-config-file user="&std";
```

The directory containing the concatenated files to read

The FILENAME Statement for Hadoop

FILENAME *fileref <device type or access method> 'external file' <options>*;

```
filename in hadoop "Hadoop-file-path" concat
                cfg=xml-config-file user="&std";
```

Specifies to read each file in the directory defined by the Hadoop file path

Note: This option is valid only for reading (not writing) Hadoop files with the FILENAME statement.

The FILENAME Statement for Hadoop

FILENAME *fileref <device type or access method> 'external file' <options>;*

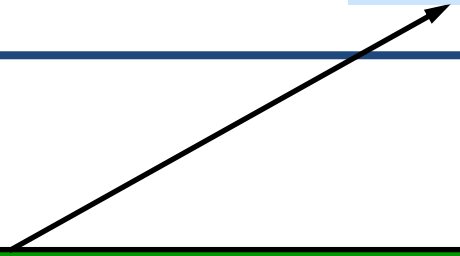
```
filename in hadoop "Hadoop-file-path" concat
                cfg=xml-config-file user="&std";
```

Points to the location of the Hadoop configuration file on the machine where SAS is executing.

The FILENAME Statement for Hadoop

FILENAME *fileref <device type or access method> 'external file' <options>;*

```
filename in hadoop "Hadoop-file-path" concat
               cfg=xml-config-file user="&std";
```



The user ID to connect to Hadoop

Reading a Hadoop File with a DATA Step

```
filename hadconfg
          "/work/hadoop_config.xml';

filename mapres hadoop
     "/user/&std/data/mapoutput" concat
     cfg=hadconfg user="&std";

data work.commonwords;
  infile mapres dlm='09'x;
  input word $ count;
  …
run;
```

Reading a Hadoop File with a DATA Step

```
filename hadconfg
      "/workshop/hadoop_config.xml';

filename mapres hadoop
      "/user/&std/data/mapoutput" concat
      cfg=hadconfg user="&std";

data work.commonwords;
   infile mapres dlm='09'x;
   input word $ count;
   …
run;
```

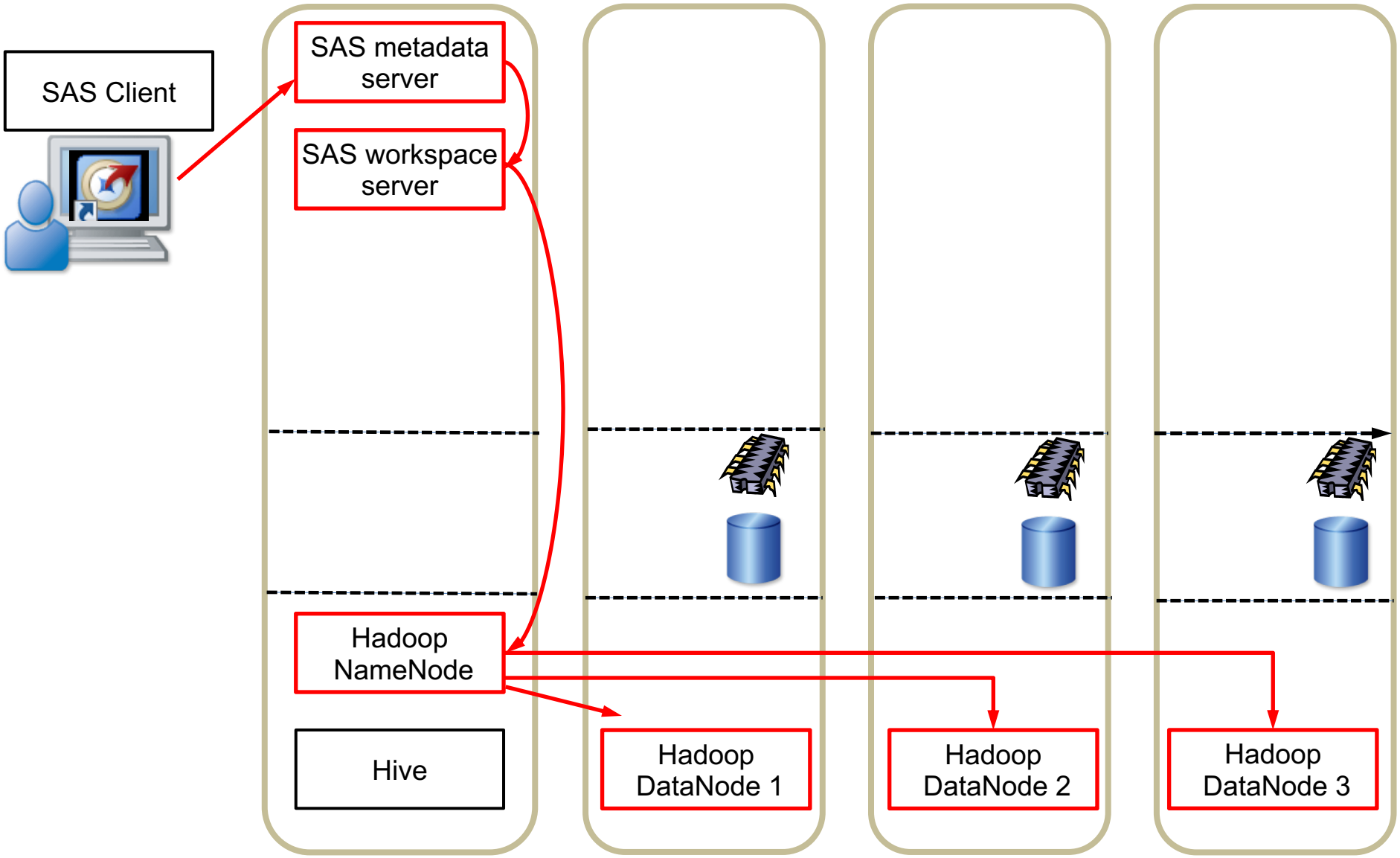
'09'x is the hex code constant for the tab character

Another Example

```
filename hadconfg
      '/work/hadoop_config.xml';
filename orders hadoop
      "/user/shared/data/custorders.txt"
      cfg=hadconfg user="&std";

data work.custorders;
 infile orders;
 input @1 customer_id 8. ...;
run;
proc print data=work.custorders;
run;
```

Does the DATA step read a single HDFS file or a concatenated directory?

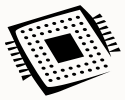
Base SAS: FILENAME for Hadoop (Review)



Reading a Hadoop File with a DATA Step

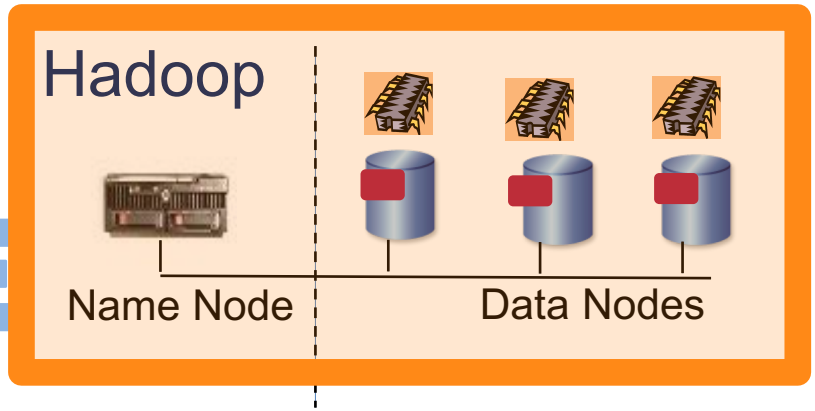
SAS Workspace Server

```
filename in hadoop
  "hadoop file" …;
data commonwords;
  infile in …;
  input …;
run;
```



Hadoop FILENAME Engine

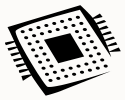
Threaded Read:
Data is read in parallel from disk in Hadoop into memory on SAS Workspace server.



Reading a Hadoop File with a DATA Step

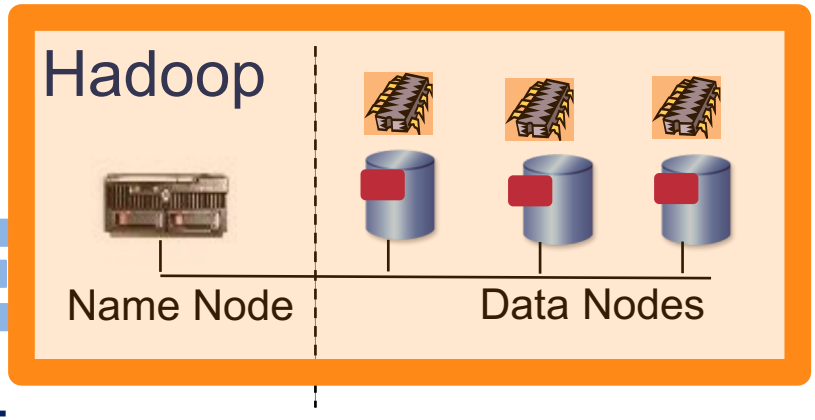
SAS Computer

```
filename in hadoop
  "hadoop file" …;
data commonwords;
  infile in …;
  input …;
run;
```

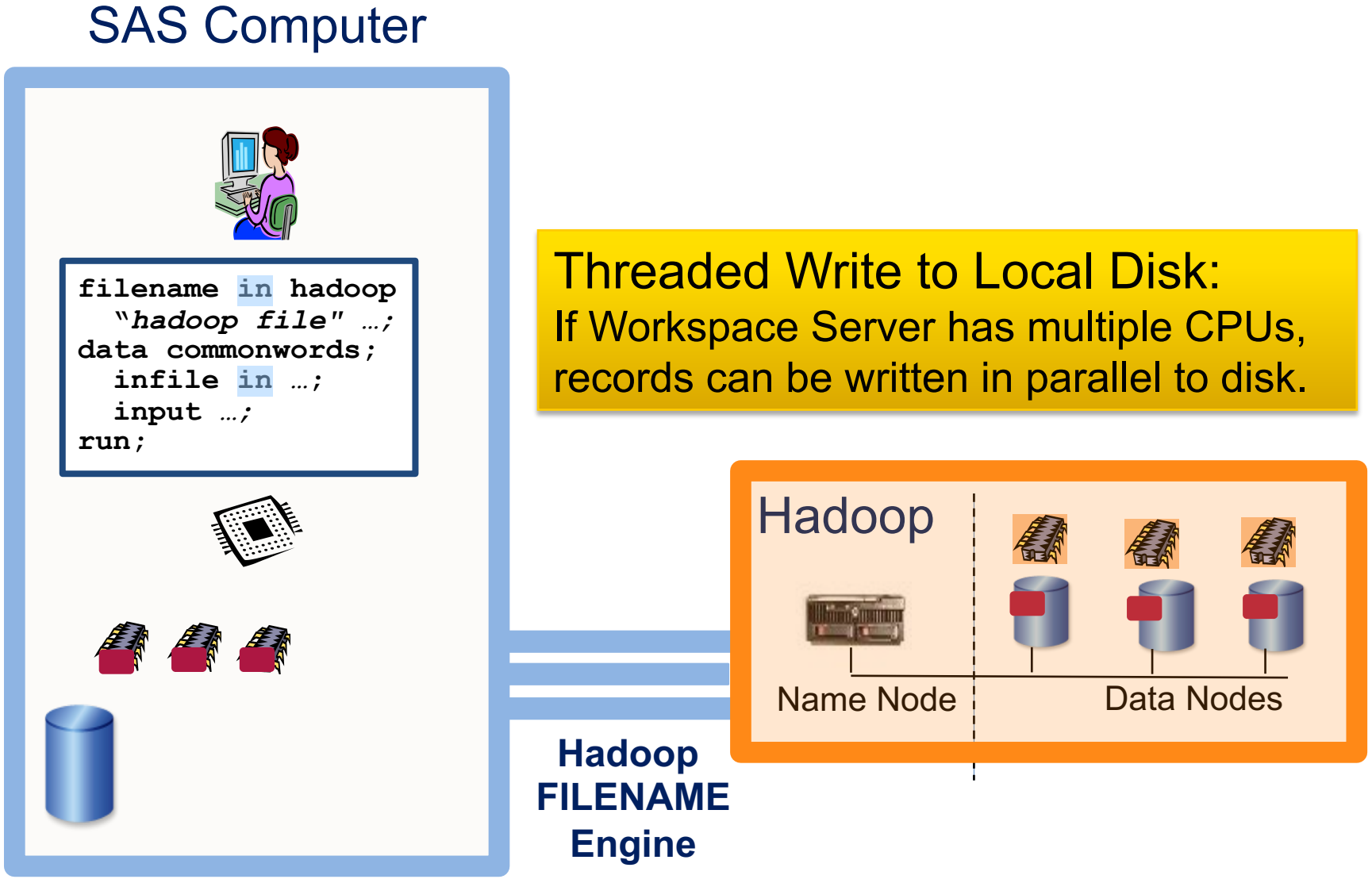


Hadoop FILENAME Engine

Sequential Processing:
SAS DATA step reads each record one at a time, sequentially.



Reading a Hadoop File with a DATA Step



Threaded Write to Local Disk:
If Workspace Server has multiple CPUs, records can be written in parallel to disk.

Writing a Hadoop File with a DATA Step

```
filename hadconfg
      '/work/hadoop_config.xml';
filename out hadoop
      "/user/&std/data/custord" dir
      cfg=hadconfg user="&std";

data _null_;
  set work.custorders;
  file out(corders) dlm=',';
  put customer_id country gender birth_date
      product_id order_date
      quantity costprice_per_unit;
run;
```

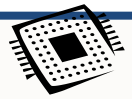
To write to files in the directory specified by the Hadoop file path

Create a file called corders in the HDFS directory "/user/&std/data/custord"

Writing a Hadoop File with a DATA Step

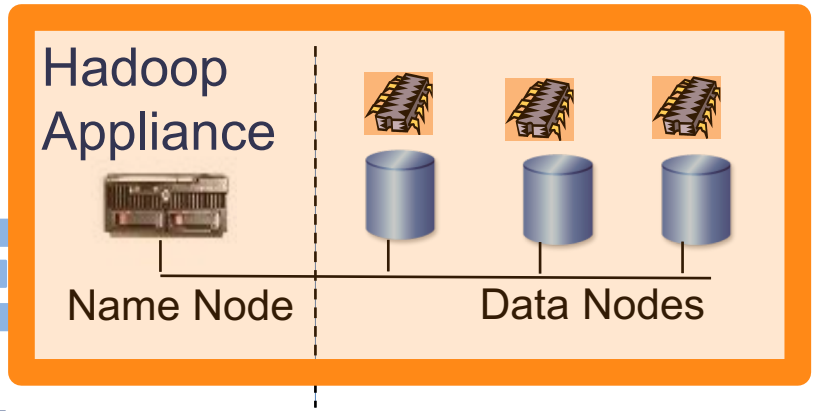
SAS Computer

```
filename out hadoop
  "hadoop file" …;
data _null_;
  set sasdataset;
  file out …;
  put …;
run;
```

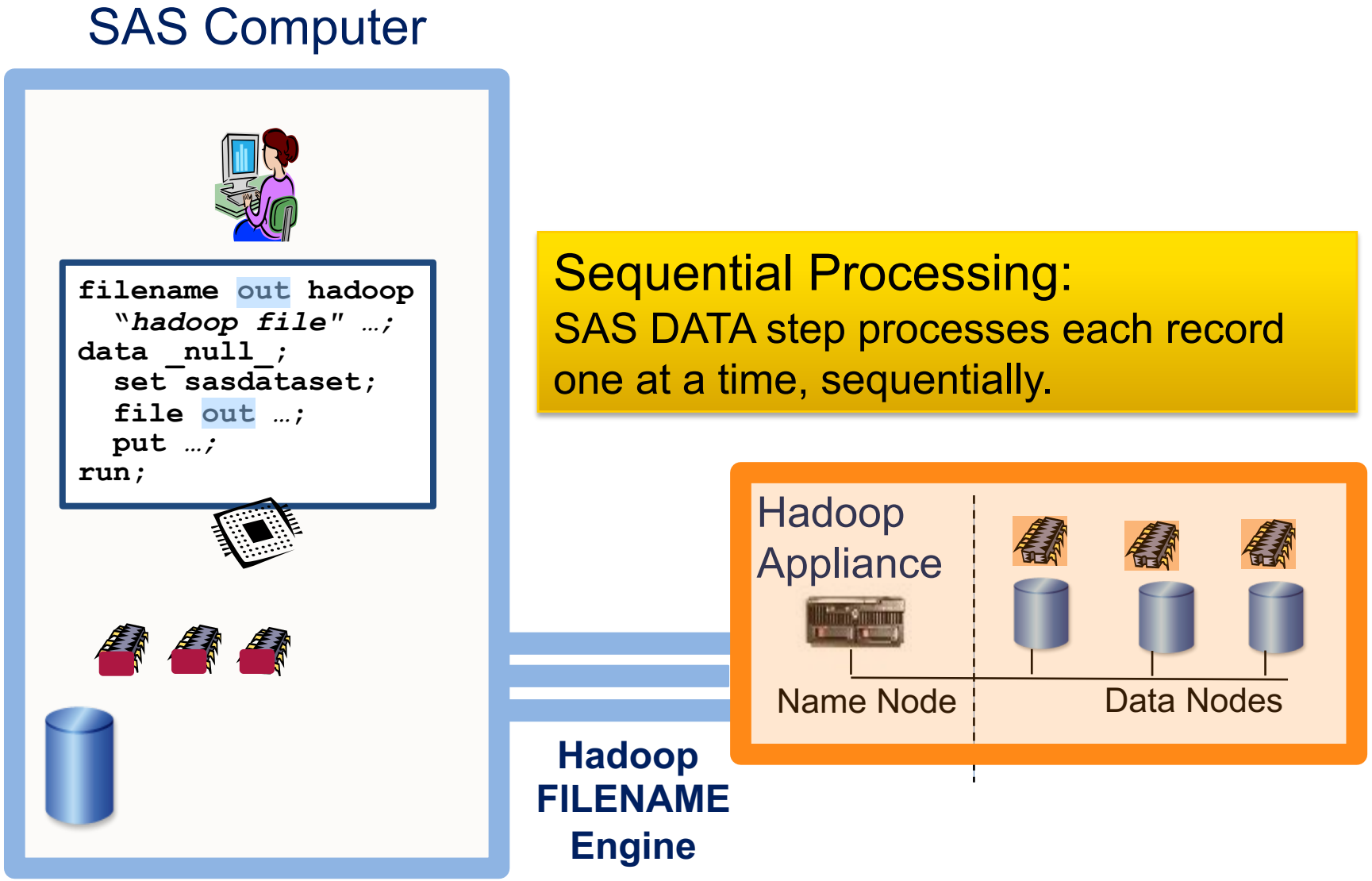


Hadoop FILENAME Engine

Threaded Read from Local Disk:
If Workspace Server has multiple CPUs, records read from disk into memory in parallel.



Writing a Hadoop File with a DATA Step

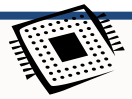


Sequential Processing:
SAS DATA step processes each record one at a time, sequentially.

Writing a Hadoop File with a DATA Step

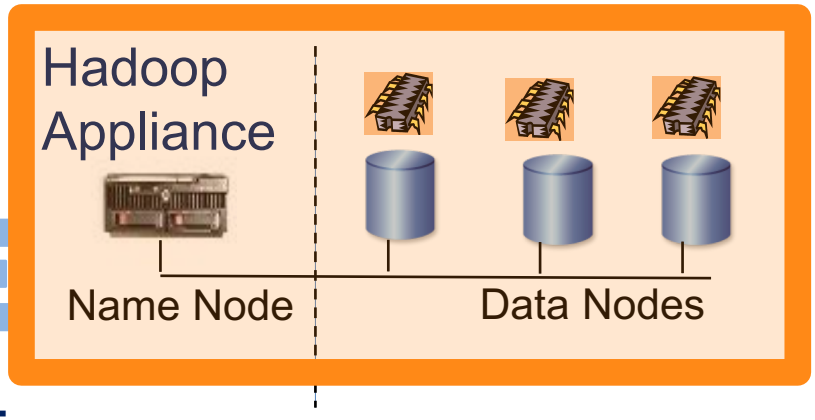
SAS Computer

```
filename out hadoop
  "hadoop file" …;
data _null_;
  set sasdataset;
  file out …;
  put …;
run;
```

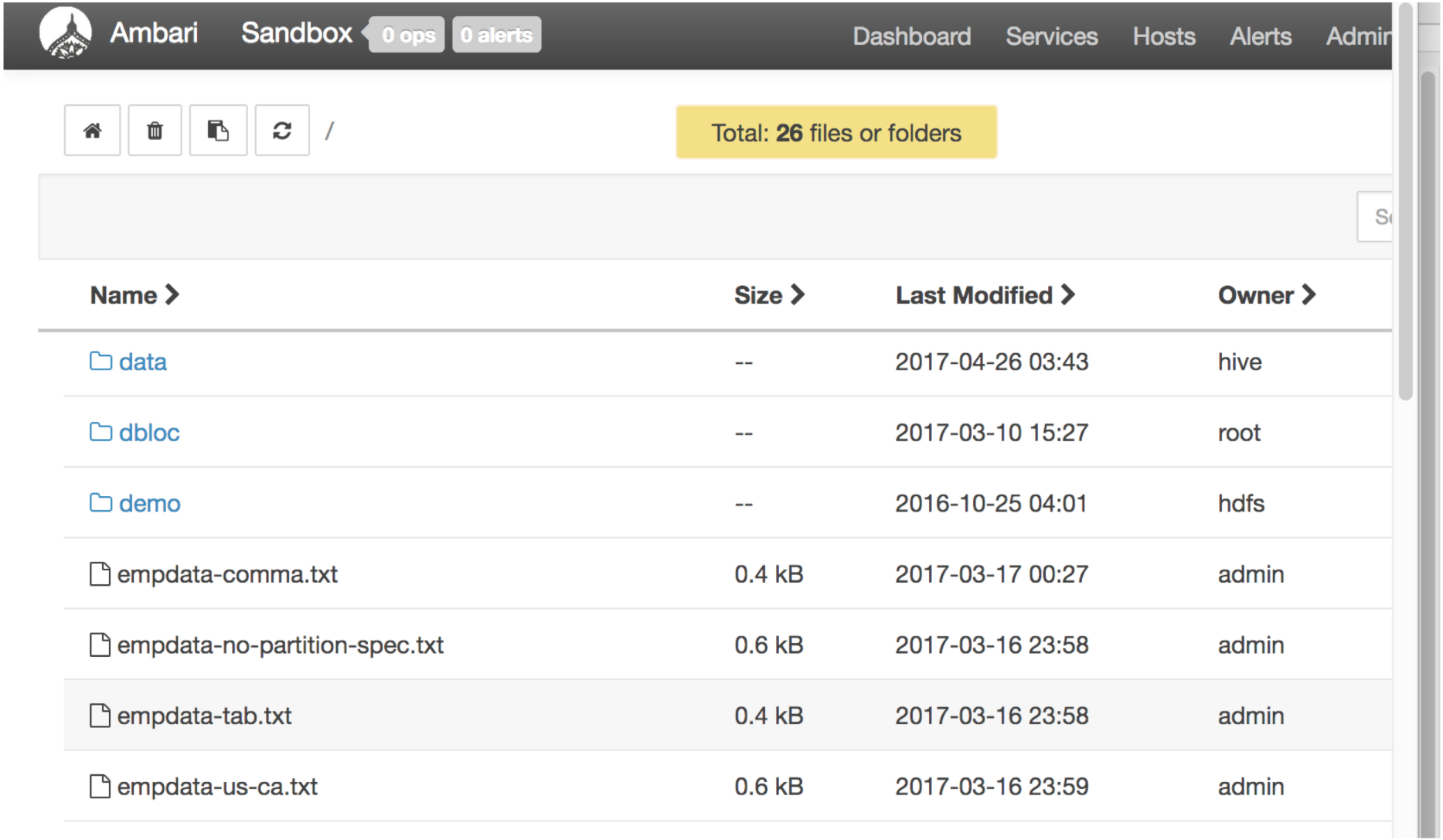


Hadoop FILENAME Engine

Parallel Write to Hadoop File System:
Records sent back to Hadoop file system and written in parallel to disks in the data nodes.



Use Ambari to Browse the Hadoop File System



Executing Pig Code With PROC HADOOP

Executing Pig Code with PROC HADOOP

```
PIG CODE=fileref | 'external-file'
      PARAMETERS=fileref | 'external-file'
      REGISTERJAR='external-file(s)'
;
```

Executing Pig Code with PROC HADOOP

```
filename pigcode
  '/workshop/DIACCHAD/pigcode.txt';

proc hadoop options=hadconfg
             username="hdfs" verbose;
   pig code=pigcode;
run;
```

Executing Pig Code with PROC HADOOP

```
filename pigcode '/workshop/DIACCHAD/pigcode.txt';

proc hadoop options=hadconfg username="hdfs" verbose;
   pig code=pigcode;
run;
```

pigcode.txt:

```
A = LOAD '/user/shared/data/custord'
        USING PigStorage (',')
          AS (customer_id, country, gender,
              birth_date, product_id, order_date,
              quantity, costprice_per_unit);

B = FILTER A BY gender == 'F';

store B into '/user/shared/data/student1';
```

Executing Pig Code with PROC HADOOP

```
A = LOAD '/user/shared/data/custord'
       USING PigStorage (',')
         AS (customer_id, country, gender,
             birth_date, product_id, order_date,
             quantity, costprice_per_unit);

B = FILTER A BY gender == 'F';

store B into '/user/shared/data/student1';
```

Load the comma delimited file and name each field

Executing Pig Code with PROC HADOOP

```
A = LOAD '/user/shared/data/custord'
       USING PigStorage (',')
         AS (customer_id, country, gender,
             birth_date, product_id, order_date,
             quantity, costprice_per_unit);

B = FILTER A BY gender == 'F';

store B into '/user/shared/data/student1';
```

Subset the loaded file for records where gender = 'F'

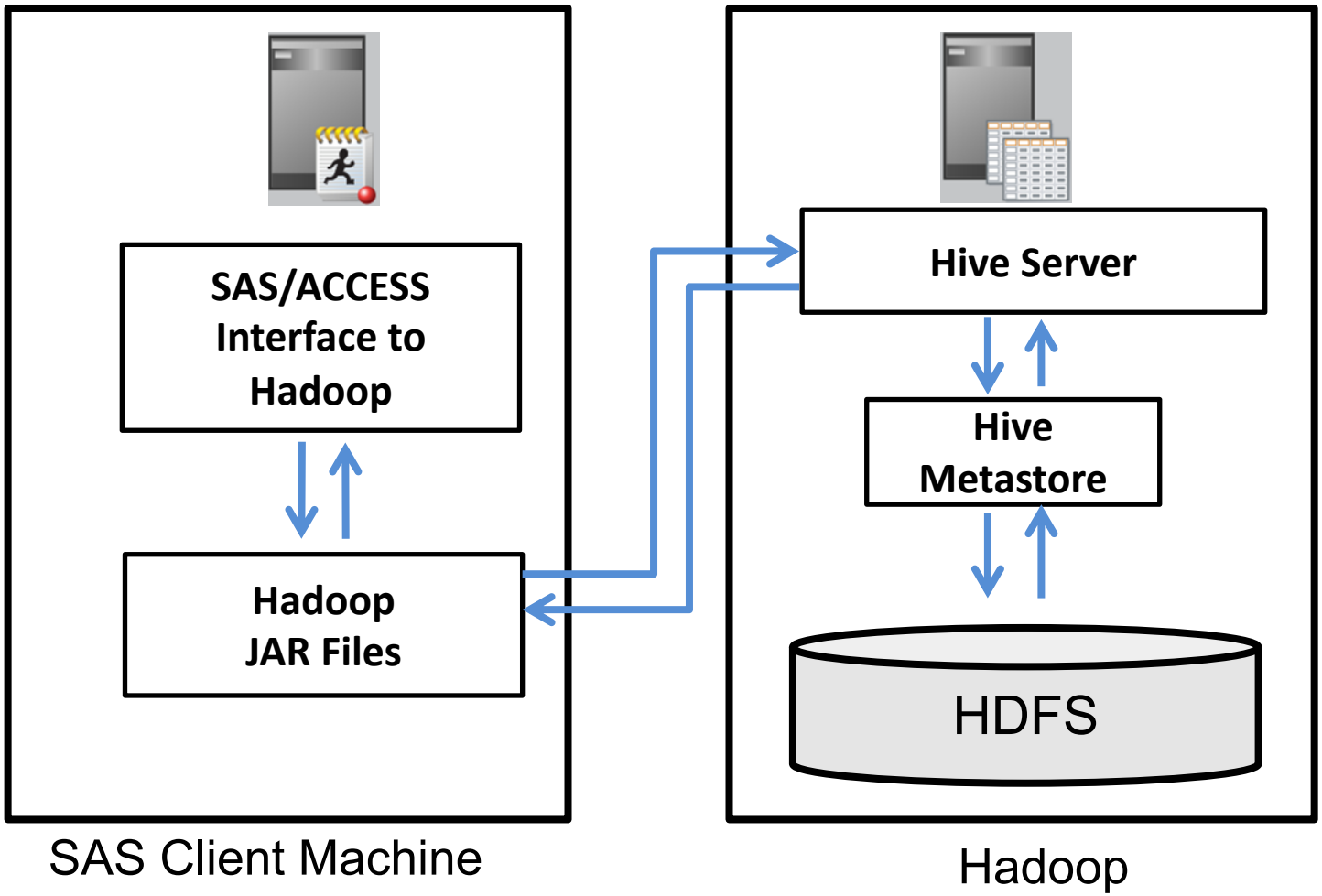
Executing Pig Code with PROC HADOOP

```
A = LOAD '/user/shared/data/custord'
       USING PigStorage (',')
         AS (customer_id, country, gender,
             birth_date, product_id, order_date,
             quantity, costprice_per_unit);

B = FILTER A BY gender == 'F';

store B into '/user/shared/data/student1';
```

Store the results in the Hadoop file system directory indicated

Using the SQL Pass-Through Facility

SAS/ACCESS Interface to Hadoop



SQL Pass-Through Query Example

```
proc sql;
 connect to hadoop
      (server=namenode port=10000 subprotocol=hive2
       schema=diacchad user="&std");
  select *
    from connection to hadoop
     (select employee_name,salary
       from salesstaff
         where emp_hire_date between
                '2011-01-01' and '2011-12-31'
      );
  disconnect from hadoop;
quit;
```

Note: The highlighted query is sent directly to Hive and is executed as a HiveQL query by Hive.

Joining Multiple Tables from Different Databases

```
proc sql;
 connect to hadoop
     (server=namenode subprotocol=hive2
      port=10000 schema=diacchad user="&std");
  select * from connection to hadoop
    (select st.employee_id, employee_name,
       st.job_title, emp_hire_date, st.salary,
       concat(ltrim(last_name), ', ' ,
              ltrim(first_name)) as mgrname
      from salesstaff st join
           diacch2.sales s
      where manager_id=s.employee_id and
            emp_hire_date between
            '2011-01-01' and  '2011-12-31');
  disconnect from hadoop;
quit;
```

Creating a SAS File from Hive Results

Partial PROC SQL code

```
proc sql;
   connect to hadoop (server=namenode
                      subprotocol=hive2 … );
   create table manager_rep_list as
   select MgrName as Manager_Name
                      label='Manager Name',
          Employee_ID label='ID Number',
          Employee_Name label='Name',
          Job_Title label='Job Title',
          input(Emp_Hire_Date, yymmdd10.)
                as Emp_Hire_Date
                format=mmddyyp10.
                label='Hired Date',
          Salary format=dollar12.
from connection to hadoop
   (select ... );
```

Using HiveQL DDL Statements in SAS

With SQL pass-through EXECUTE statements, Hive tables can be defined using HiveQL Data

```
proc sql;
   connect to hadoop (connection options);
     execute (create table customer
           (customer_id int,
            country string
            gender  string,
            birthdate string)
            row format delimited
            fields terminated by '\001'
       stored as textfile
       location "/user/&std/data/customer")
     by hadoop;
   disconnect from Hadoop;
quit;
```

Using the SAS/ACCESS LIBNAME Engine

- Using the LIBNAME Statement for Hadoop
- Using Data Set Options
- Creating Views
- Combining Tables
- Transferring Data Sets from SAS to Hive

The LIBNAME Statement (Review)

- The LIBNAME statement assigns a libref to a SAS library.

LIBNAME *libref* '*SAS-data-library*' <*options*>;

- Rules for naming a libref:
  - The name must be eight characters or less.
  - The name must begin with a letter or underscore.
  - The remaining characters must be letters, numbers, or underscores.

The SAS/ACCESS LIBNAME Statement

The SAS/ACCESS LIBNAME statement does the following:

- establishes a libref, which acts as an alias or nickname to Hive.
- permits a Hive table to be referenced by a two-level name.
- enables the use of the SAS/ACCESS LIBNAME statement **options** to specify how Hive objects are processed by SAS.
- enables you to customize how to connect to Hive.

SAS/ACCESS LIBNAME Statement

```
libname hivedb hadoop server=namenode
        subprotocol=hive2
        port=10000 schema=diacchad
        user=studentX pw=StudentX;
```

LIBNAME ***libref*** ***engine-name*** *<connection-options>*
<LIBNAME-options>;

- SAS Log

```
23      libname hivedb hadoop server=namenode
24              subprotocol=hive2
25              port=10000 schema=diacchad
26              user="&std" pw="&stdpw";
NOTE: Libref HIVEDB was successfully assigned as follows:
    Engine:       HADOOP
    Physical Name: jdbc:hive2://namenode:10000/diacchad
```

LIBNAME Statement Connection Options

Option	Specifies
USER=	Hive user name
PW=	Hive password associated with the Hive user
SERVER=	Hadoop server machine name or IP address
SUBPROTOCOL=	The version of Hive that is running
SCHEMA=	The Hive schema to access
PORT	Port for connection to server

Note: This is not a full list of the possible options.

Listing of Hive Tables in the Schema

```
proc contents data=hivedb._all_  nods;
run;
```

- Partial PROC CONTENTS Output

#	Name	Member Type
1	CUSTOMER	DATA
2	CUSTOMERORDERS	DATA
3	CUSTOMER_DIM_MORE	DATA
4	EMPLOYEE_ADDRESSES	DATA
5	EMPLOYEE_DONATIONS	DATA
6	EMPLOYEE_ORGANIZATION	DATA
7	EMPLOYEE_PAYROLL	DATA
8	EMPLOYEE_PHONES	DATA
9	ORDER_FACT	DATA

Contents of a HiveTable

```
proc contents data=hivedb.customerorders;
run;
```

Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Informat	Label
11	costprice_per_unit	Num	8			costprice_per_unit
1	customer_id	Num	8			customer_id
5	delivery_date	Num	8	DATE9.	DATE9.	delivery_date
12	discount	Num	8			discount
2	employee_id	Num	8			employee_id
4	order_date	Num	8	DATE9.	DATE9.	order_date
6	order_id	Num	8			order_id
7	order_type	Num	8			order_type
8	product_id	Num	8			product_id
9	quantity	Num	8			quantity
3	street_id	Num	8			street_id
10	total_retail_price	Num	8			total_retail_price

Listing of a Hive Table

```
proc print data=hivedb.customerorders;
    var customer_id order_type
        order_date product_id quantity;
    where order_type=3 and
          order_date between '01Jan2011'd
          and '15Jan2011'd;
run;
```

- PROC PRINT Output:

Obs	customer_id	order_type	order_date	product_id	quantity
1	24	3	02JAN2011	240800200021	2
2	53	3	06JAN2011	210200500002	3

SAS/ACCESS Engine Implicit Pass-through

- Behind the scenes, the SAS/ACCESS engine attempts to convert your SAS code to a HiveQL query that is implicitly passed to Hive. This maximizes as much as possible the amount of processing done by Hive in the Hadoop system.

- By default, there is no indication regarding how much of the processing was passed by the SAS/ACCESS engine to Hive.

Implicit versus Explicit SQL Pass-through

- When you send database-specific query code to the database via SQL pass-through syntax, it is called *explicit SQL pass-through*.

- When the SAS/ACCESS engine translates SAS language code into database-specific query code on your behalf to retrieve data from the database, it is called *implicit SQL pass-through*. This occurs when you access database tables via the SAS/ACCESS LIBNAME statement.

Optimizing Implicit Pass-through

- When using implicit pass-through, it is critical to develop SAS code that maximizes the amount of data processing done in Hadoop via HiveQL and minimizes the movement of data from Hadoop to SAS. In particular, the programmer perform the following processes in the Hadoop cluster as much as possible:
  - summarization
  - subsetting
  - table joins

In addition, unless output tables are small, store the results as Hive tables in HDFS rather than SAS data sets.

The SASTRACE= SAS System Option

- You can use the SASTRACE system option to assess how much of the data processing occurs in the database before results are returned to SAS.

SASTRACE= 'Value'	
',,,d'	specifies that all SQL statements sent to the DBMS are sent to the log.
',,,s'	specifies that a summary of timing information for calls made to the DBMS is sent to the log.

- Note: You can combine both options using SASTRACE=',,,ds'.

SASTRACE= SAS System Option

- The SASTRACE= option enables you to examine the HiveQL that the SAS/ACCESS engine submits to the Hive.

OPTIONS SASTRACE='*value*' <*options*>;

```
options sastrace=',,,d' nostsuffix
        sastraceloc=saslog;
proc print data=hivedb.customerorders;
run;
options sastrace=off;
```

Turns Off SASTRACE

NOSTSUFFIX SAS System Option

- The NOSTSUFFIX option limits the amount of information displayed in the log.

```
OPTIONS NOSTSUFFIX;
```

```
options sastrace=',,,d' nostsuffix
        sastraceloc=saslog;
proc print data=hivedb.customerorders;
run;
options sastrace=off;
```

SASTRACELOC= SAS System Options

- The SASTRACELOC= option defines where to send the SASTRACE information.

OPTIONS SASTRACELOC=STDOUT|SASLOG|FILE '*filename*';

```
options sastrace=',,,d' nostsuffix
        sastraceloc=saslog;
proc print data=hivdb.customerorders;
run;
options sastrace=off;
```

SASTRACE= Messages

- Partial SAS Log

```
24          proc print data=hivedb.customerorders;
HADOOP_1: Executed: on connection 6
USE `diacchad`

HADOOP_2: Prepared: on connection 6
SHOW TABLE EXTENDED LIKE `CUSTOMERORDERS`

HADOOP_3: Prepared: on connection 6
SELECT * FROM `CUSTOMERORDERS`

HADOOP_4: Prepared: on connection 6
DESCRIBE FORMATTED CUSTOMERORDERS
25          run;

HADOOP_5: Executed: on connection 6
SELECT * FROM `CUSTOMERORDERS`

NOTE: PROCEDURE PRINT used (Total process time):
      real time           5.71 seconds
      cpu time            1.65 seconds
```

Checks if the table exists

Determines column names and types

Retrieves SASFMT information in TBLPROPERTIES

User query executed

Using the MEANS and FREQ Procedures

```
options sastrace=',,,d' sastraceloc=saslog
        nostsuffix;

proc means data=hivedb.order_fact sum mean;
   var total_retail_price;
run;

proc freq data=hivedb.order_fact;
   tables order_type;
run;

options sastrace=off;
```

Using the MEANS and FREQ Procedures

Partial SAS Log – PROC MEANS

```
NOTE: SQL generation will be used to perform the initial summarization.

HADOOP_41: Executed: on connection 7
 select T1.ZSQL1, T1.ZSQL2, T1.ZSQL3, T1.ZSQL4 from
              ( select COUNT(*) as ZSQL1, COUNT(*) as ZSQL2,
                COUNT(TXT_1.`total_retail_price`) as ZSQL3,
                SUM(TXT_1.`total_retail_price`) as ZSQL4
                from `ORDER_FACT` TXT_1 ) T1
                          where T1.ZSQL1 > 0

ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.
```

- Hive returns the count and sum
- SAS calculates the mean

continued...

Using the MEANS and FREQ Procedures

Partial SAS Log – PROC FREQ

```
NOTE: SQL generation will be used to construct frequency and crosstabulation tables.

HADOOP_53: Executed: on connection 7
 select COUNT(*) as ZSQL1, case  when COUNT(*) >      COUNT(TXT_1.`order_type`) then  NULL  else
              MIN(TXT_1.`order_type`) end as ZSQL2,
              MAX(TXT_1.`order_type`) as ZSQL3 from `ORDER_FACT` TXT_1
              group by TXT_1.`order_type`

ACCESS ENGINE:  SQL statement was passed to the DBMS for fetching data.
```

Note: The REPORT, SORT, SUMMARY, and TABULATE procedures also generate HiveQL implicit pass-through queries.

Supported SAS Language Functions

- Selected Functions

AVG	LENGTH	MINUTE	STD
COUNT	LOG	MONTH	SUM
DAY	LOWCASE	SECOND	DATEPART
EXP	MAX	STRIP (TRIM)	UPCASE
HOUR	MIN	SUBSTR	YEAR

- For a complete list of functions passed to Hive, see the SAS documentation *Passing SAS Functions to Hadoop*.

Using a Supported SAS Function

```
options sastrace=',,,d' sastraceloc=saslog
        nostsuffix;

/* Orders Placed in 2011 */
data orders2011;
   set hivedb.customerorders ;
   where year(order_date)=2011;
run;

options sastrace=off;
```

Using a Non-Supported Function

```
options sastrace=',,,d' sastraceloc=saslog
        nostsuffix;

/* Orders with Saturday Delivery */
data sat_deliveries;
   set hivedb.customerorders ;
   where weekday(delivery_date)=7;
run;

options sastrace=off;
```

SAS function that returns the day of the week, where 1=Sunday and 7=Saturday

Using SAS Data Set Options

Option	Function
KEEP= *column1 column2*	Lists the column names to include in processing or writing to output tables.
DROP= *column1 column2*	Lists the column names to exclude in processing or writing to output tables.
OBS= *n*	Specifies the number of the last observation to process.
FIRSTOBS= *n*	Specifies the first observation to begin reading. By default, FIRSTOBS=1.
RENAME= (*oldname=newname*)	Enables you to change the name of columns in output data sets.

Selected SAS Data Set Options

```
data salesrep_i;
   set hivedb.salesstaff
       (drop=ssn birth_date
        rename=(job_title=jobcode));
   where jobcode='Sales Rep. I';
run;
```

SASDATEFMT= SAS/ACCESS Data Set Option

- The SASDATEFMT= SAS/ACCESS data set option converts Hive date values to the specified SAS date, time, or datetime format.

```
proc print data=hivedb.salesstaff
             (sasdatefmt=(birth_date="mmddyyp10."
              emp_hire_date="date11."));
   var employee_id employee_name birth_date
       emp_hire_date;
   where birth_date < '01jan1955'd;
run;
```

Quoted String

SASDATEFMT= SAS/ACCESS Data Set Option

- Partial SAS Output

Obs	employee_id	employee_name	birth_date	emp_hire_date
1	120148	Zubak, Michael	09.13.1953	01-JUN-1982
2	121025	Cassey, Barnaby	10.10.1953	01-SEP-1979
3	121066	Norman, Ceresh	08.23.1948	01-JAN-1978
4	120134	Shannan, Sian	06.06.1953	01-JAN-1978
5	121053	Mcdade, Tywanna	09.23.1948	01-FEB-1978
6	120172	Comber, Edwin	04.01.1948	01-JAN-1978
7	120151	Phaiyakounh, Julianna	11.21.1948	01-JAN-1978

Creating a SAS PROC SQL View

General form of the CREATE VIEW statement:

```
PROC SQL;
      CREATE VIEW view-name AS
            SELECT column-1, column-2,…column-n
                FROM table-1<,table-n>
                 …;
QUIT;
```

Note: The underlying tables in the view can be Hive tables.

Creating a SAS PROC SQL View

```
libname hivedb hadoop server=namenode
        subprotocol=hive2
        port=10000 schema=diacchad
        user="&std" pw="&stdpw";
proc sql;
   create view mysasdat.US_F_customers as
      select customer_id, customer_name,
        birth_date format=year4.,
        customer_type_id label='Customer Type'
       from hivedb.customer
       where country='US' and gender='F';
quit;
```

SAS Log

```
NOTE: SQL view mysasdat.US_F_CUSTOMERS has been defined.
```

Using a SAS PROC SQL View

You can reference the PROC SQL view the same way you reference a SAS data set.

```
proc freq data=mysasdat.US_F_Customers;
   tables customer_type_id birth_date/nocum;
run;
```

SAS Output:

Customer Type

customer_type_id	Frequency	Percent
1020	1	7.69
1030	2	15.38
1040	2	15.38
2010	1	7.69
2020	3	23.08
2030	2	15.38
3010	2	15.38

birth_date

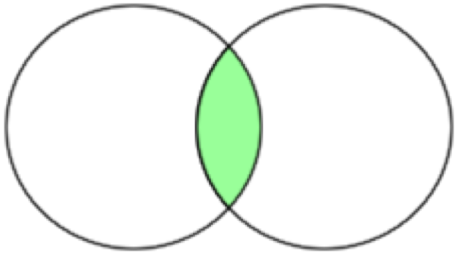
birth_date	Frequency	Percent
1938	1	7.69
1963	2	15.38
1968	1	7.69
1973	3	23.08
1983	3	23.08
1988	2	15.38
1990	1	7.69

Ways to Combine Data

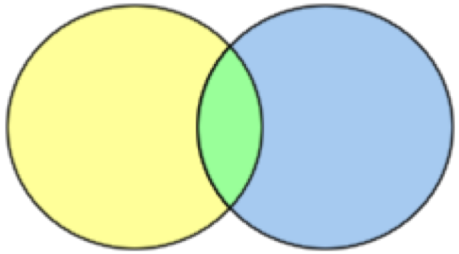
- SAS supports several ways to combine tables or views, including the following:
  - joining via the SQL procedure
  - merging via the DATA step
  - concatenating via the DATA step
  - set operators via the SQL procedure

Passing Joins to Hive

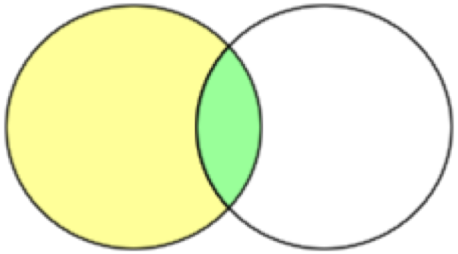
- When you join two Hive tables accessed via the same SAS/ACCESS library, PROC SQL can often pass the join to Hive for processing.



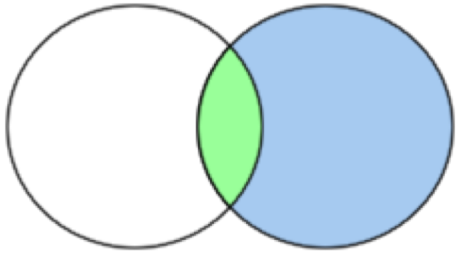
Inner Join



Full Join



Left Join



Right Join

Joining Tables from a Single Connection

```
libname hivedb hadoop subprotocol=hive2
      server=namenode port=10000
      schema=diacchad user="&std" pw="&stdpw";
proc sql;
 create table manager_rep_list as
  select st.employee_iD, employee_name,
      st.job_title, emp_hire_date, st.salary,
      trim(last_name)|| ', ' ||
      trim(first_name) as mgrname
   from hivedb.salesstaff st,
        hivedb.sales s
    where manager_id=s.employee_id and
      emp_hire_date between '01JAN2011'd
            and '31DEC2011'd;
quit;
```

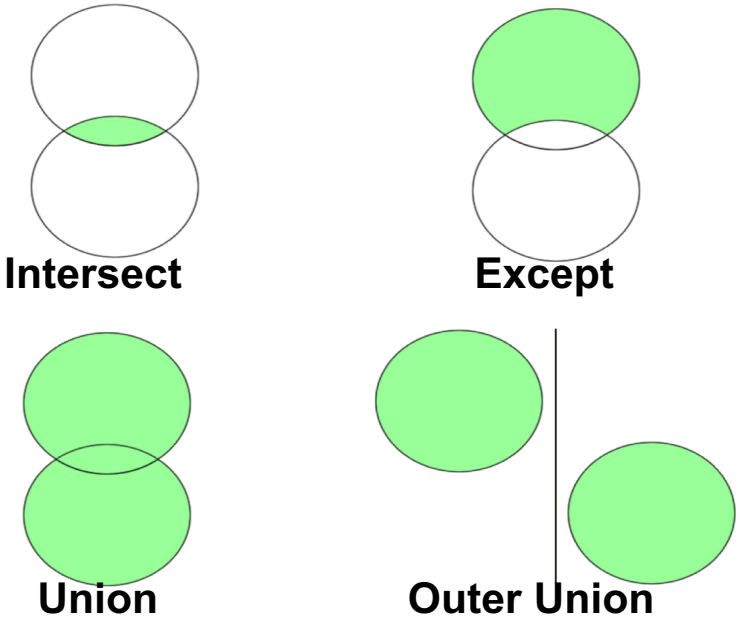
same library

Joining Tables from Multiple Connections

```
libname hivedb hadoop subprotocol=hive2 server=namenode
      port=10000 schema=diacchad user="&std" pw="&stdpw";
libname hived2 hadoop subprotocol=hive2 server=namenode
      port=10000 schema=diacch2 user="&std" pw="&stdpw";
proc sql;
 create table manager_rep_list as
  select st.employee_iD, employee_name,
      st.job_title, emp_hire_date, st.salary,
      trim(last_name)|| ', ' ||
      trim(first_name) as mgrname
   from hivedb.salesstaff st,
        hived2.sales s
    where manager_id=s.employee_id and
      emp_hire_date between '01JAN2011'd
            and '31DEC2011'd;
quit;
```

SQL Set Operators

- When you perform a set operation using SAS/ACCESS libref to Hive connection, PROC SQL passes a SELECT statement for each of the tables, retrieves the results from each, and performs the final set operation.

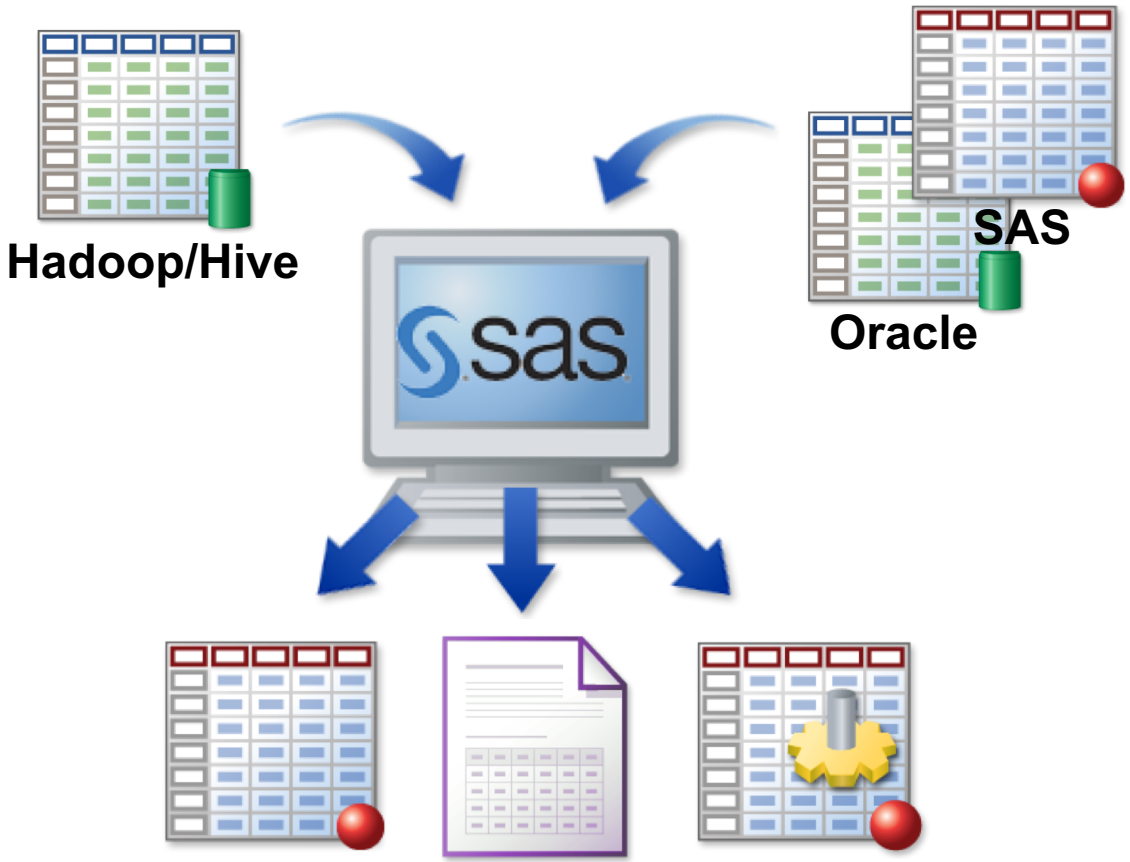


Stacking Tables Using Set Operators

```
proc sql number;
  create table type2 as
   select order_id, customer_id, order_type,
          order_date, delivery_date
       from hivedb.qtr1
       where order_type=2
             union
   select order_id, customer_id, order_type,
          order_date, delivery_date
       from hivedb.qtr2
       where order_type=2;

  select * from type2;
quit;
```

Combining Hive Tables with Other Sources

- The SAS System facilitates combining tables not only from Hive, but also from different data sources to create tables, views, or reports.



Combining SAS Data Set and Hive Table

```
Libname sasdata "/workshop/DIACCHAD";
libname hivedb hadoop server=namenode
        subprotocol=hive2
        port=10000 schema=diacchad
        user="&std" pw="&stdpw";
data phonelist;
  merge sasdata.employee_phones(in=p)
        hivedb.salesstaff(in=s);
  by employee_id;
  if p=1 and s=1;
  keep employee_id employee_name phone_type
       phone_number;
run;
```

Copying Data Sets to Hive

- Using SAS libraries, data can be copied to Hive using standard SAS language syntax.

```
libname sasdat "/workshop/DIACCHAD";
libname myhive hadoop server=namenode
        subprotocol=hive2
        port=10000 schema=&std
        user="&std" pw="&stdpw";
proc copy in=sasdat out=myhive;
     select customer_dim;
run;
```

- In this example, the customer_dim data set is copied from a local SAS library (**sasdat**) to the Hive schema (**myhive**).