DISTRIBUTED AND CLOUD COMPUTING

LAB 9: MAP-REDUCE STREAMING & INTRO TO SPARK

Plan for today

A more detailed look into YARN + YARN set up

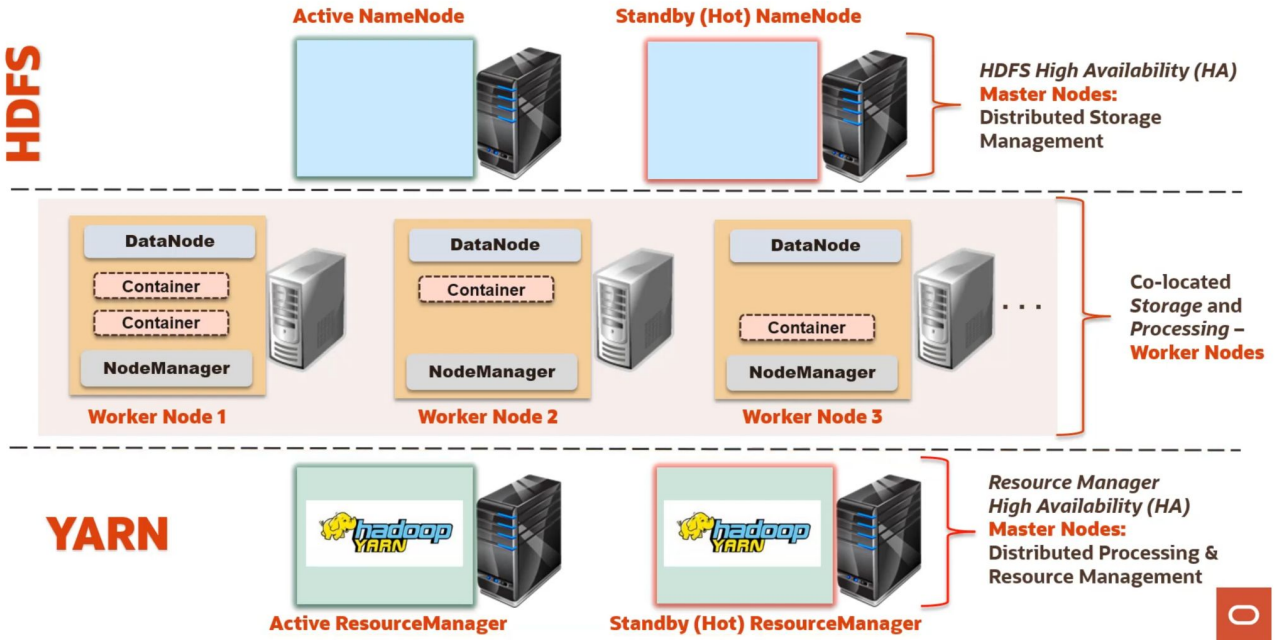
Mapreduce streaming & implementation of more complex mapreduce program

YARN intuition

YARN is to computation what HDFS is to data storage

In HDFS a master node (NameNode) listens to client requests for file system operations and orchestrates the cluster to achieve them in a distributed fault-tolerant way

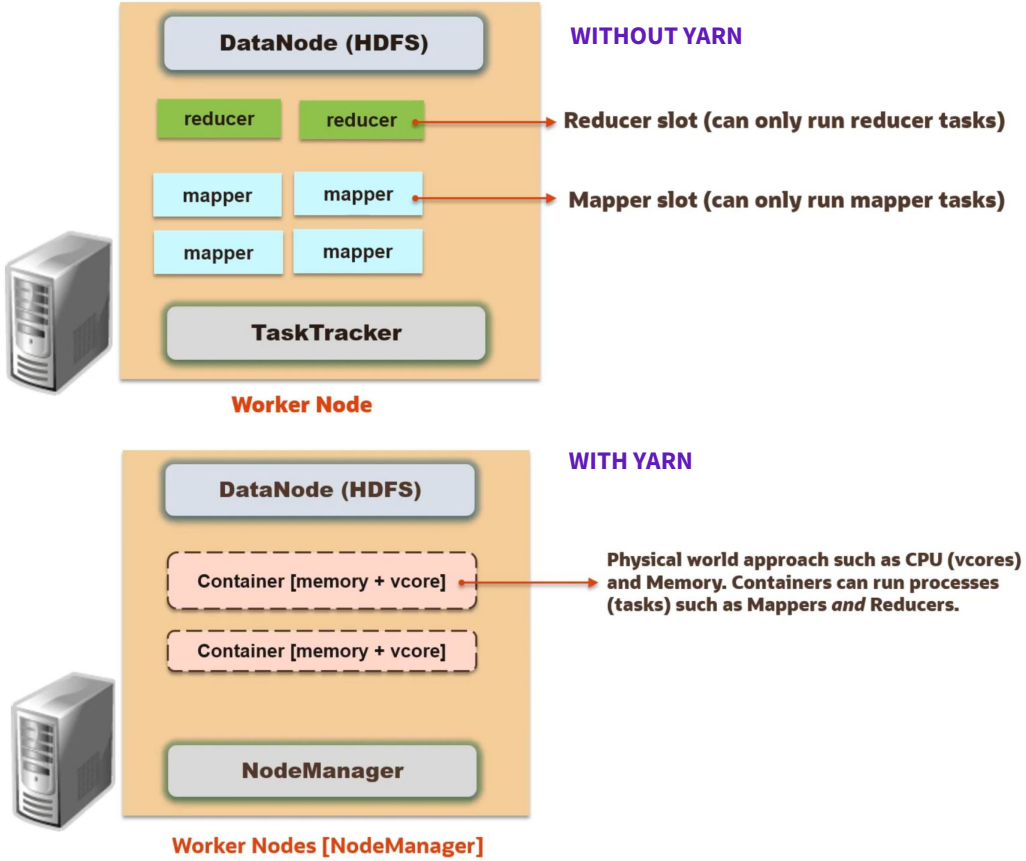
In **YARN** a master node **(ResourceManager)** listens to client requests for computation **(jobs)** and orchestrates the cluster to achieve them in a distributed fault-tolerant way



In HDFS the cluster computers run a DataNode daemon that listens to the NameNode

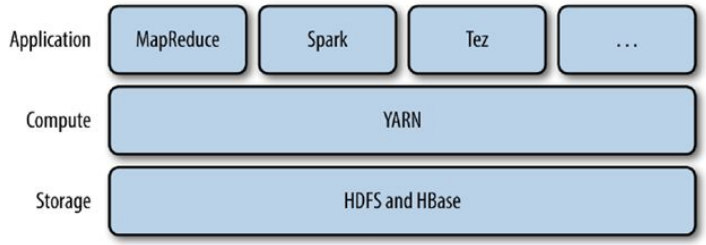
In YARN the **SAME** cluster computers run a **NodeManager** daemon that listens to the ResourceManager

Hadoop cluster computation with and without YARN

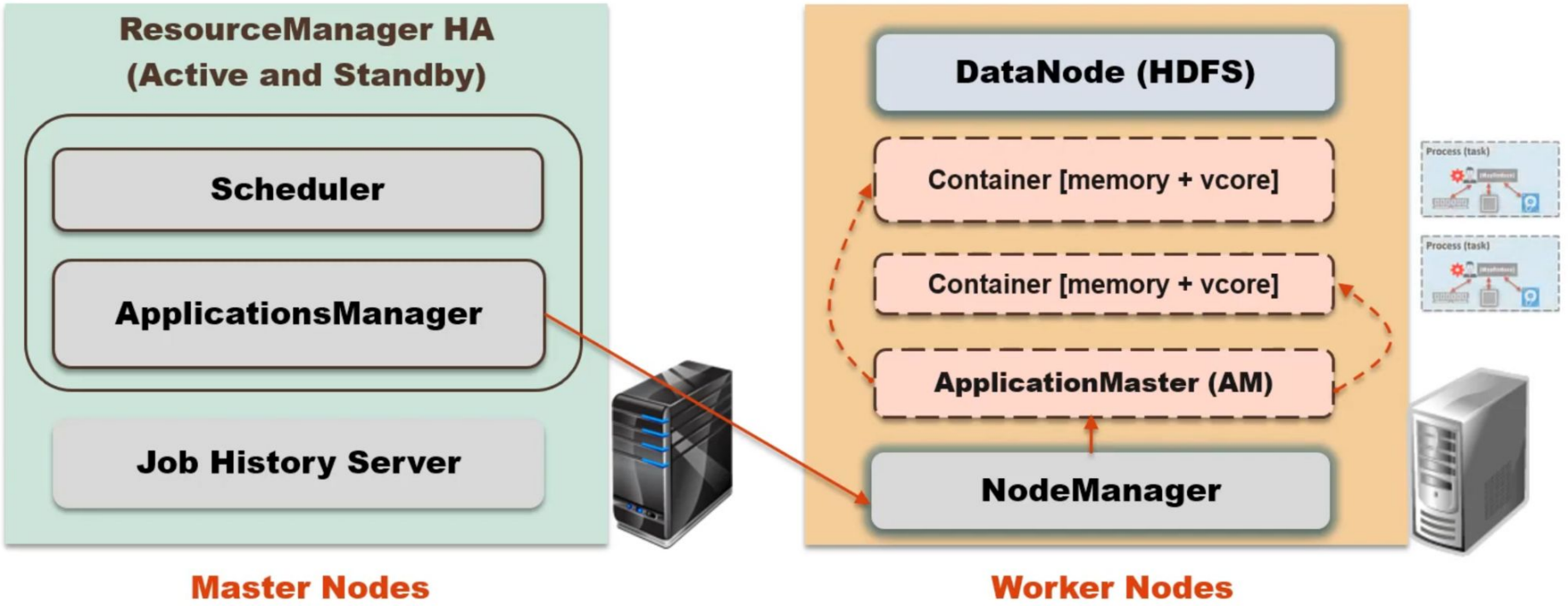


YARN containers allow for execution of other applications besides Map Reduce

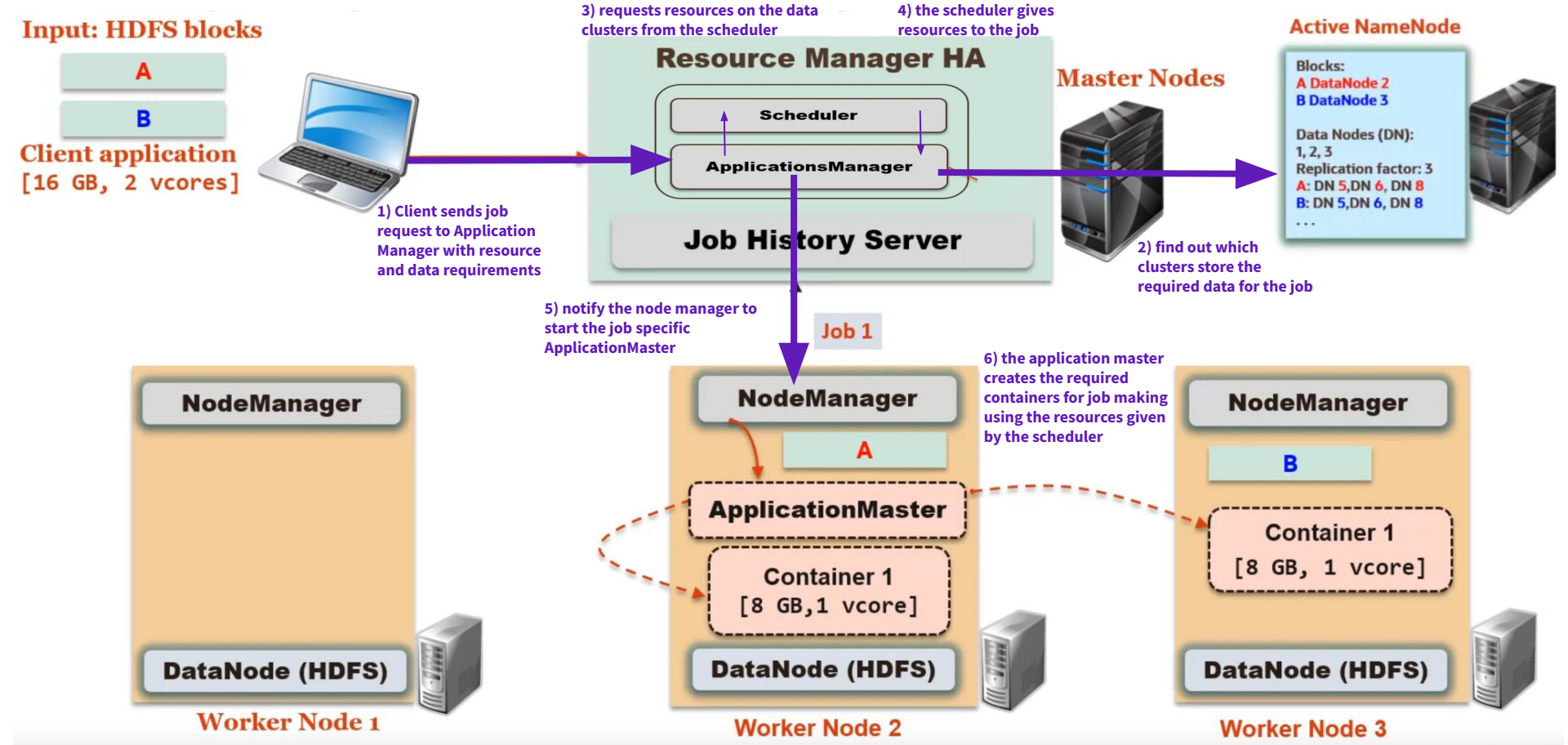
Spark is the one we are interested in



YARN: Resource Manager and NodeManager



YARN: Example of a YARN job execution



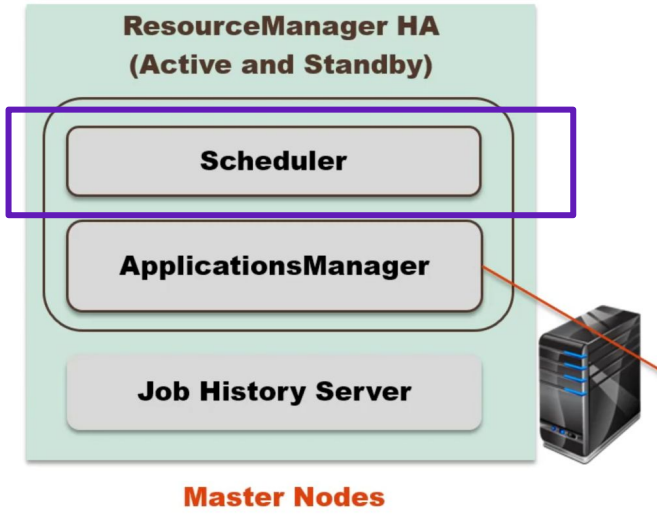
YARN: Scheduling

- The YARN scheduler is in charge of determining **where** and **when** submitted jobs will be executed
- The following strategies are available for **when**

First In, First out (FIFO) Scheduler
- Allocated resources to tasks in arrival order

Capacity Scheduler
- Tasks are assigned to priority queues
- Resources are assigned to tasks waiting in higher priority queues
- Within the same queue, FIFO is used

Fair Scheduler
- Tasks are assigned to queues (NOT priority queues)
- The scheduler assinges resources to tasks in all queues FAIRLY



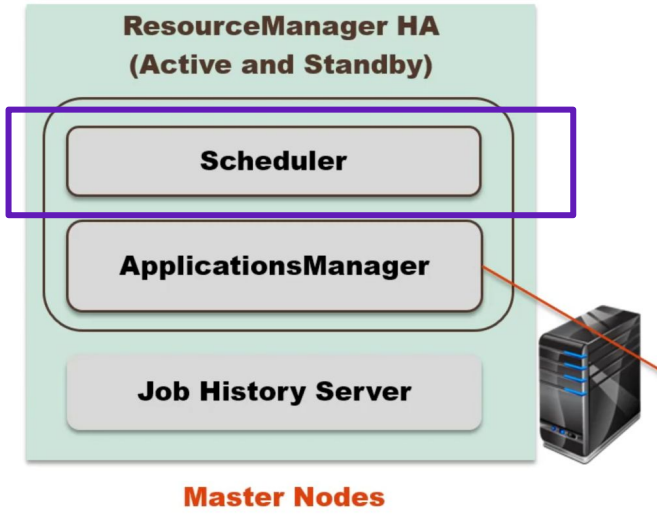
- The **where** is determined by **where** the data is stored in HDFS

YARN: Why is scheduling important

- Scheduling is extremely important
- An effective scheduling strategy can maximises the resource utilisation and minimises the task waiting time
- Bad resource utilisation → increased energy usage
  - 10 clusters running at 10% > 1 cluster at 100%
- However scheduling everything to one cluster will lead to massive waiting times

Finding an optimal scheduling strategy for a given workload is an NP-Complete problem

Fancy way of saying super hard



TASK 1: Set up YARN in your local Hadoop installation

STEP 1: Start HDFS in pseudo-distributed mode

- Use this link to remember the commands that we used last time
- If you were not here last time follow the tutorial in last weeks

STEP 2: In the same link from STEP 1 navigate to the section:

"YARN on a Single Node"

1) Copy the configuration to the appropriate file xml configuration files
2) Execute $ sbin/start-yarn.sh
   a) You must be in the hadoop home directory
3) Navigate to localhost:8088
   a) You should see something like the lower image

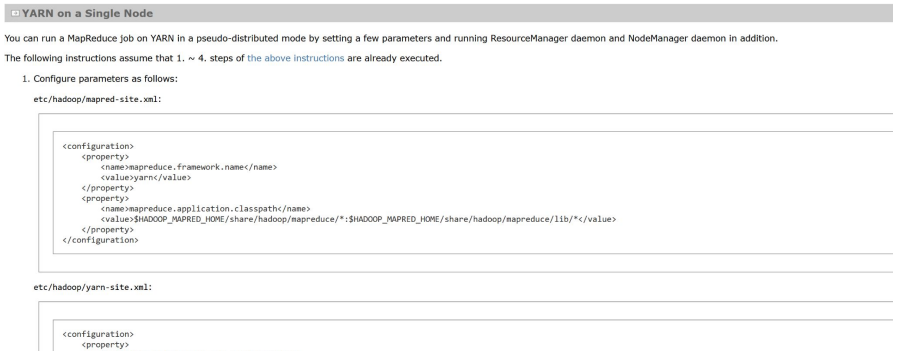
STEP 3: Execute a mapreduce operation and look around the YARN endpoint to see the execution details. You can do this by:

A) Modifying this example in standalone operation to work in the pseudo distributed mode by

- Creating the ***HDSF://user/<YOU>*** directory (if not there)
- Using hdfs commands to mimic ***$ cp etc/hadoop/*.xml input*** (place these files in ***HDSF://user/<YOU>***)
- Execute the mapreduce command using the correct hadoop version and paths

B) Compiling and running the word count java program example here

- Be careful to modify the command with the correct paths etc..



YARN on a Single Node

You can run a MapReduce Job on YARN in a pseudo-distributed mode by setting a few parameters and running ResourceManager daemon and NodeManager daemon in addition.

The following instructions assume that 1. ~ 4. steps of the above instructions are already executed.

1. Configure parameters as follows:

etc/hadoop/mapred-site.xml:

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
    <property>
        <name>mapreduce.application.classpath</name>
        <value>$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/*:$HADOOP_MAPRED_HOME/share/hadoop/mapreduce/lib/*</value>
    </property>
</configuration>
```

etc/hadoop/yarn-site.xml:

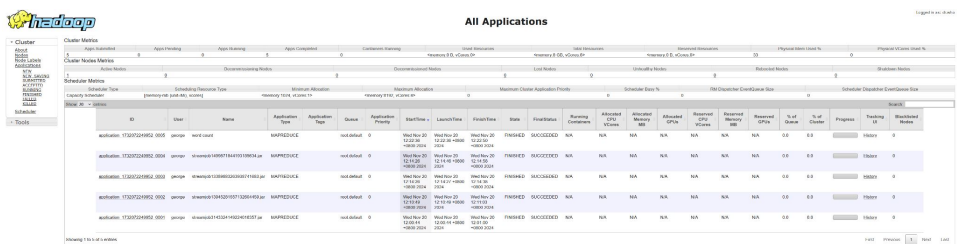
```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
    <property>
        <name>yarn.nodemanager.env-whitelist</name>
        <value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,HADOOP_YARN_HOME,HADOOP_HOME,PATH,LANG,TZ,HADOOP_MAPRED_HOME</value>
    </property>
</configuration>
```

2. Start ResourceManager daemon and NodeManager daemon:

```
$ sbin/start-yarn.sh
```

3. Browse the web interface for the ResourceManager; by default it is available at:

- ResourceManager - http://localhost:8088/



HDFS commands to manipulate the file system : https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-common/FileSystemShell.html

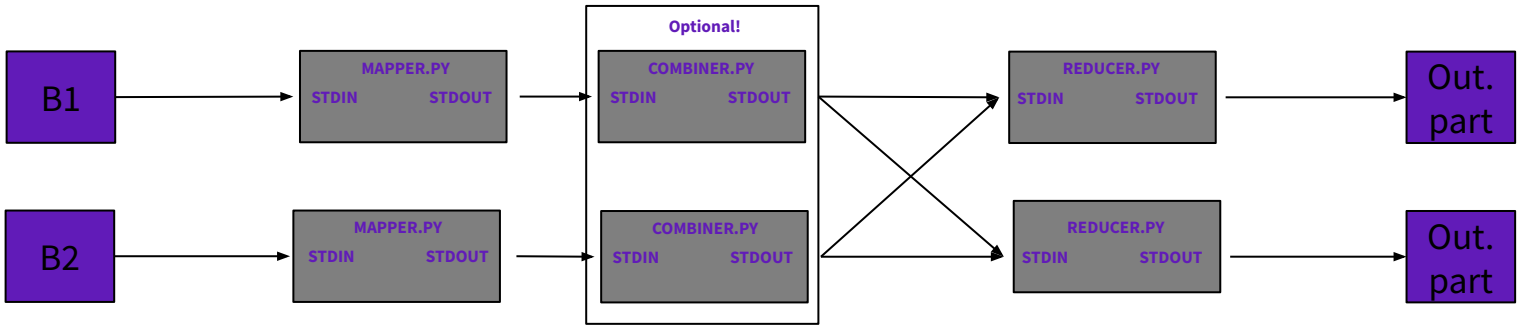
MapReduce Streaming

- So far all the examples of MapReduce we have seen used Java binaries for the mapper and reducer processes
  - Using the pre-compiled examples that shipped with hadoop (grep)
  - Using mappers and reducers written jars compiled by us (word count)
- MapReduce streaming offers the ability to use **ANY** kind of executable for the mapper and the reducer process
  - These can be shell scripts, python scripts or any type of executable binary

Gives the programmer much more freedom when writing mapreduce programs

MapReduce Streaming

- MapReduce streaming jobs utilise the **stdin** and **stdout** buffers
- Raw data are fed to the stdin of the mappers which output <k1,v1> to stdout
- The stdout of the mappers is sorted and fed to the stdin of the reducer scripts which output <k2, v2> to stdout
- The stdout of the reducers is the written to back HDFS as parts
  - New Lines (‘\n’) are used to separate the key value pairs
  - The first tab (‘\t’) separates the key from the value
    - ***<key:text up to first tab, value:remaining text until new line>***
  - This can be customized through -inputformat



Task 2: Mapreduce streaming with Python scripts

TASK DESCRIPTION: Write the **mapper.py** and **reducer.py** python scripts to be used by a mapreduce streaming job that searches 'commerce_data_cleaned.csv' for the most bought products by country (the format of the csv file can be seen below)

STEP 0: Start the NameNode, DataNodes and Yarn daemons and create ***HDFS://user/<user_name>*** if not already there
STEP 1: Download the 'commerce_data_cleaned.csv' and streaming_word_count example from blackboard (dataset is from here)
STEP 2: Move the .csv and .txt files to HDFS
STEP 3: Execute the providence mapper and reducer scripts with the input.txt to test that streaming works and its behaviour (also YARN)
STEP 3: Implement **mapper.py** and **reducer.py** (don't forget to include *#!/bin/python* at the first line in each file) for the taks
STEP 4: Execute the mapreduce process on the 'commerce_data_cleaned.csv' and check out the results

Command for mapreduce streaming (assuming current working dir is the hadoop home directory)

$ /bin/mapred streaming -input <path_to_data> -output <path_to_output> -mapper <path_to_mapper.py> -file <path_to_mapper.py> -reducer <path_to_reducer.py> -file <path_to_reducer.py>

CSV column labels

```
InvoiceNo,StockCode,Description,Quantity,InvoiceDate,UnitPrice,CustomerID,Country
536365,85123A,WHITE HANGING HEART T-LIGHT HOLDER,6,12/1/2010 8:26,2.55,17850,United Kingdom
536365,71053,WHITE METAL LANTERN,6,12/1/2010 8:26,3.39,17850,United Kingdom
```

Snippet of the expected output

```
(base) george@DESKTOP-E24BUDU:~/DistSys24/hadoop/st
hdfs dfs -cat /user/george/output_commerce/*
Australia       ('MINI PAINT SET VINTAGE ', 2916)
Austria ('SET 12 KIDS COLOUR  CHALK STICKS', 288)
Bahrain ('ICE CREAM SUNDAE LIP GLOSS', 96)
Belgium ('PACK OF 72 RETROSPOT CAKE CASES', 480)
```