COSI 129a

Introduction to Big Data Analysis

Fall 2016

Map Reduce- Hadoop Tutorial

Configuration Notes

- Not a substitute for documentation and experimentation, but these notes will give you an idea of what you are doing when you are configuring and starting Hadoop

Modes of Operation

- Single-node, single process (standalone)
  - Useful for debugging
- Single-node, multi-process (pseudo-cluster)
  - Processes run independently on single node and communicate via the network abstraction
- Multi-node (real cluster)
  - Processes may be on separate nodes and communicate via the network

Prerequisites

- Supported Platforms
  - GNU/Linux & Windows is supported
  - CS department supports only GNU/Linux
- Required Software
  - Java (recommended version for different Hadoop versions)
  - ssh must be installed
    - sshd must be running
    - Need to run Hadoop scripts and manage the daemons

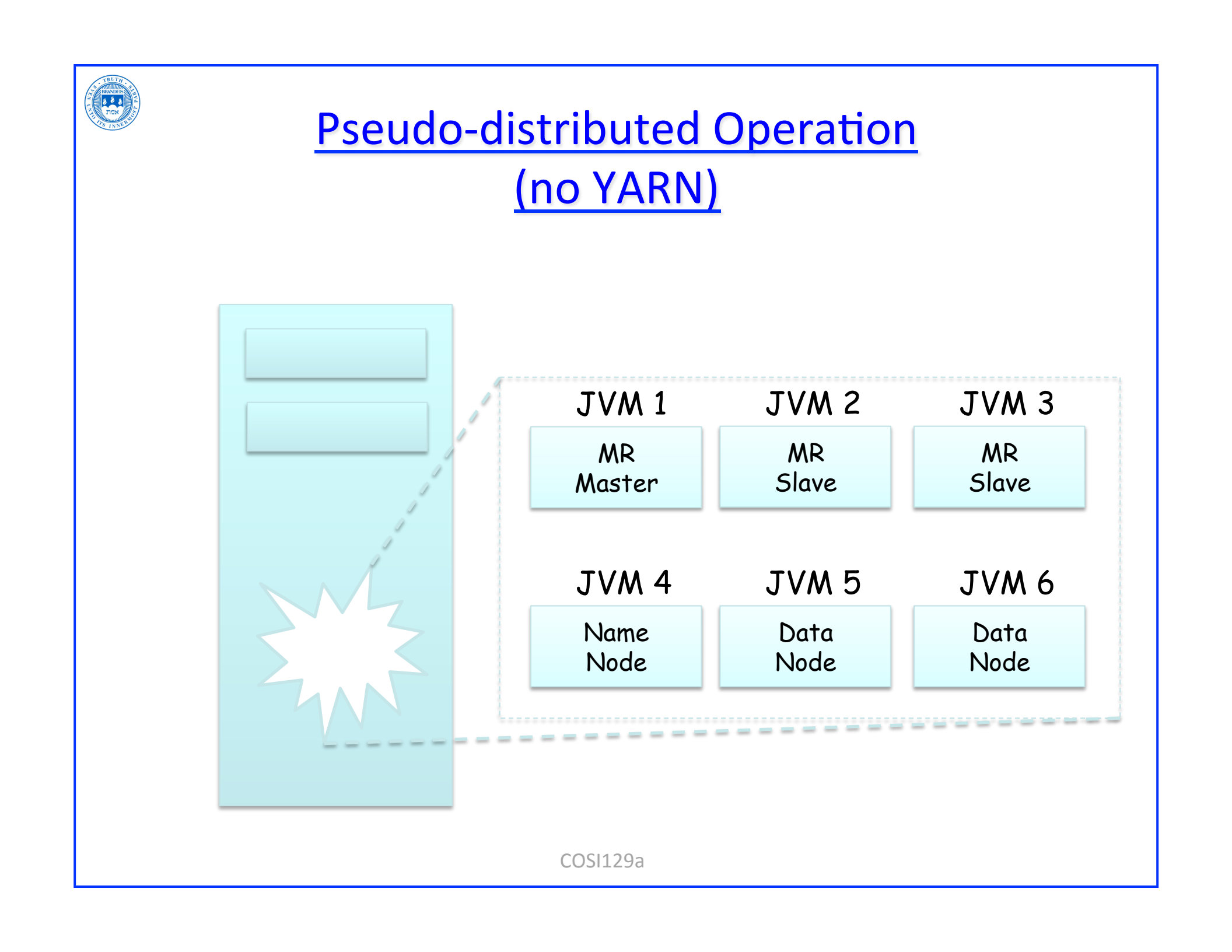
How to get started

- Download the latest Hadoop distribution
  - Unpack it in $HADOOP_HOME dir
- Update configuration parameters
  - File: $HADOOP_HOME/etc/hadoop/hadoop-env.sh
    - Set JAVA_HOME to your latest version
    - Set HADOOP_PREFIX to your $HADOOP_HOME
- Check for the available Hadoop commands
  - bin/hadoop

Standalone Operation

- Default setup used for debugging
- Example: the following commands copy files to an input directory, executes the grep MR program and writes the output to an output directory

```
> mkdir input
>$ cp etc/hadoop/*.xml input
>$ bin/hadoop jar share/hadoop/mapreduce/hadoop-
mapreduce-examples-2.5.0.jar grep input output 'dfs[a-z.]+'
>$ cat output/*
```



Pseudo-distributed Operation (no YARN)

Pseudo Distributed - How to set it up

1. Specify the location of HDFS

conf/etc/hadoop/core-site.xml:

```
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://localhost:9000</value>
    </property>
</configuration>
```

conf/etc/hadoop/hdfs-site.xml:

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
</configuration>
```

Pseudo Distributed - How to set it up

- Hadoop uses ssh to start daemons on nodes in the cluster (even if you are starting all the daemons on a single node!)
- Check whether you can connect without a passphrase:
  - **`$> ssh localhost`**
- If not, setup by executing the following:
  - **`$> ssh-keygen -t dsa -P '' -f ~/.ssh/id_dsa`**
  - **`$> cat ~/.ssh/id_dsa.pub >> ~/.ssh/authorized_keys`**

Pseudo Distributed - How to set it up

- Execution
  1. Format the file system (once on new cluster)

     ```
     >bin/hdfs namenode –format
     ```

  2. Start HDFS (NameNode & DataNode daemon)

     ```
     >sbin/start-dfs.sh
     ```

  3. Check NameNode status

     ```
     http://localhost:50070/
     ```

  4. Create HDFS dirs you need for your job

     ```
     >bin/hdfs dfs -mkdir /user
     >bin/hdfs dfs -mkdir /user/<username>
     ```

Pseudo Distributed - How to set it up

- Execution
  5. Copy files into HDFS

     > `bin/hdfs dfs -put etc/hadoop input`

  6. Run your MR code

     >`bin/hadoop jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.5.0.jar grep input output 'dfs[a-z.]+'`

  7. Examine your output

     >`bin/hdfs dfs -cat output/*`

  8. Stop the daemons

     >`sbin/stop-dfs.sh`

Pseudo Distributed w YARN- How to set it up

HDFS is up and running (previous steps 1-4)

1. Configure YARN to execute MR applications

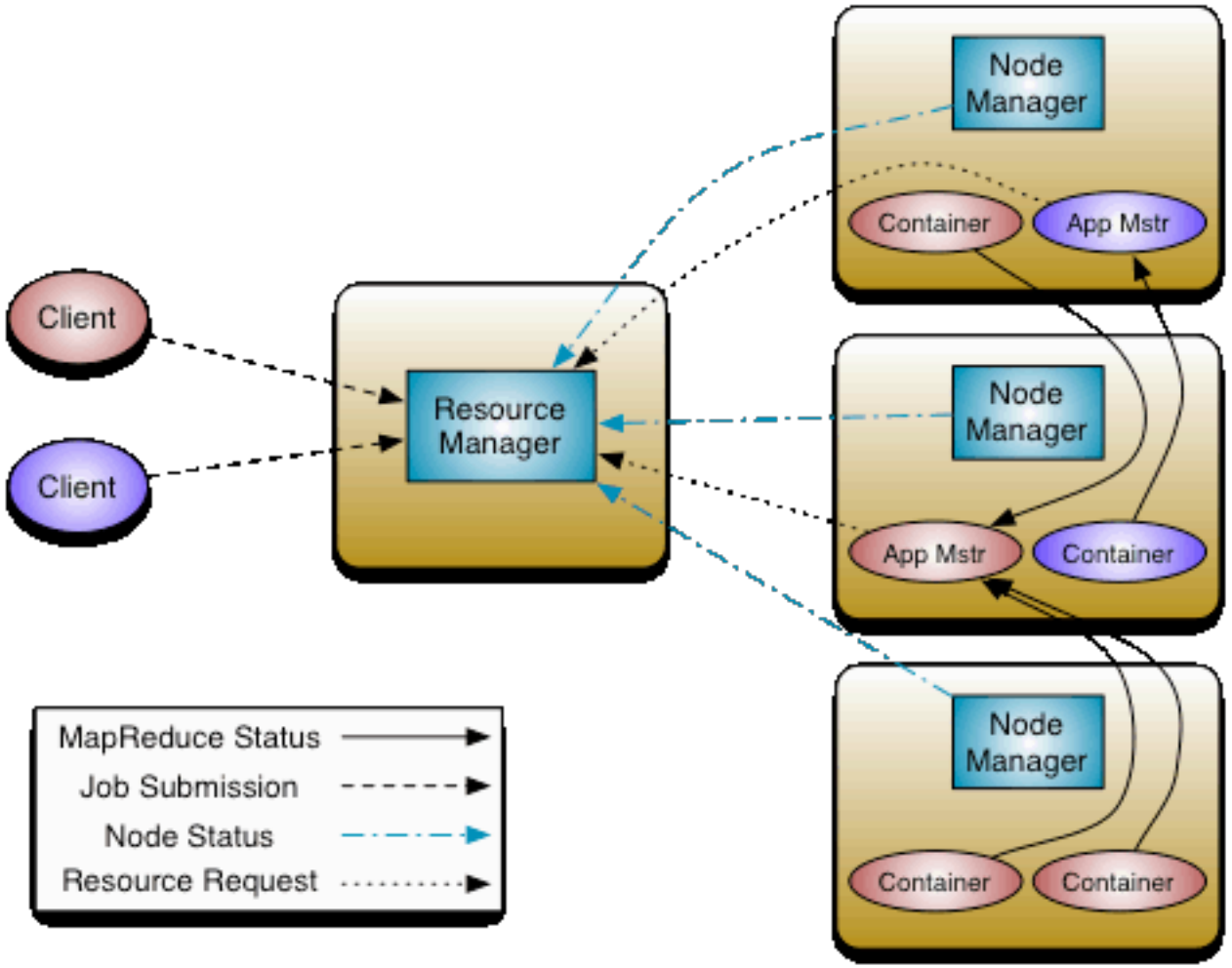
conf/etc/hadoop/mapred-site.xml:

```
<configuration>
    <property>
      <name>mapreduce.framework.name</
name>
      <value>yarn</value>
    </property>
</configuration>
```

conf/etc/hadoop/yarn-site.xml:

```
<configuration>
    <property>
       <name>yarn.nodemanager.aux-
services</name>
      <value>mapreduce_shuffle</value>
    </property>
</configuration>
```

YARN


Client
Client
Resource Manager
Node Manager
Container
App Mstr
Node Manager
App Mstr
Container
Node Manager
Container
Container
MapReduce Status
Job Submission
Node Status
Resource Request

Pseudo Distributed w YARN- How to set it up

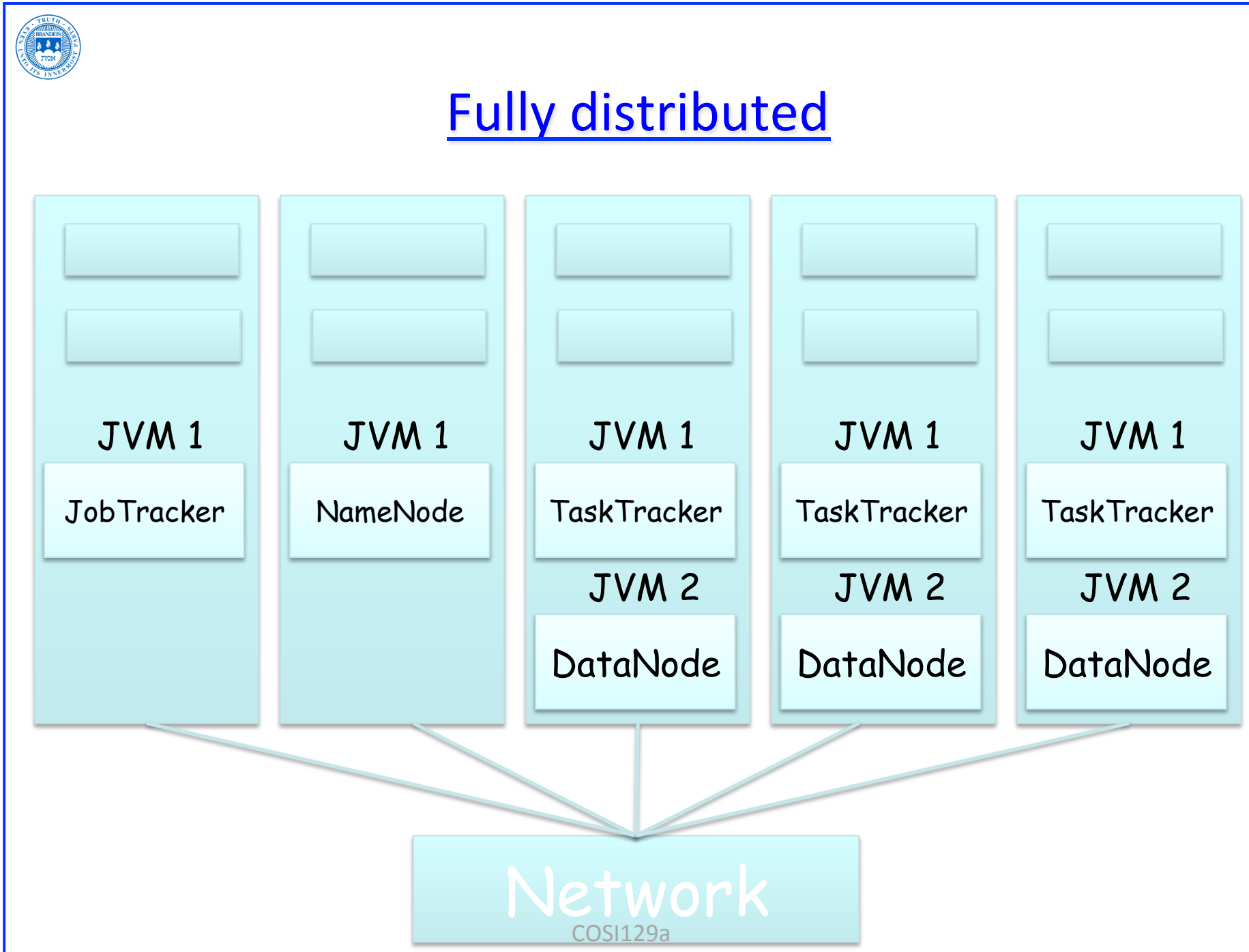
- Execution
  1. Start ResourceManager and Node Manager daemons

     `>sbin/start-yarn.sh`
  2. Check ResourceManager status

     http://localhost:8088/
  3. Run an MR job

     `steps 5-6`
  4. Stop the daemons

     `>sbin/stop-yarn.sh`



Fully distributed

JVM 1
JobTracker

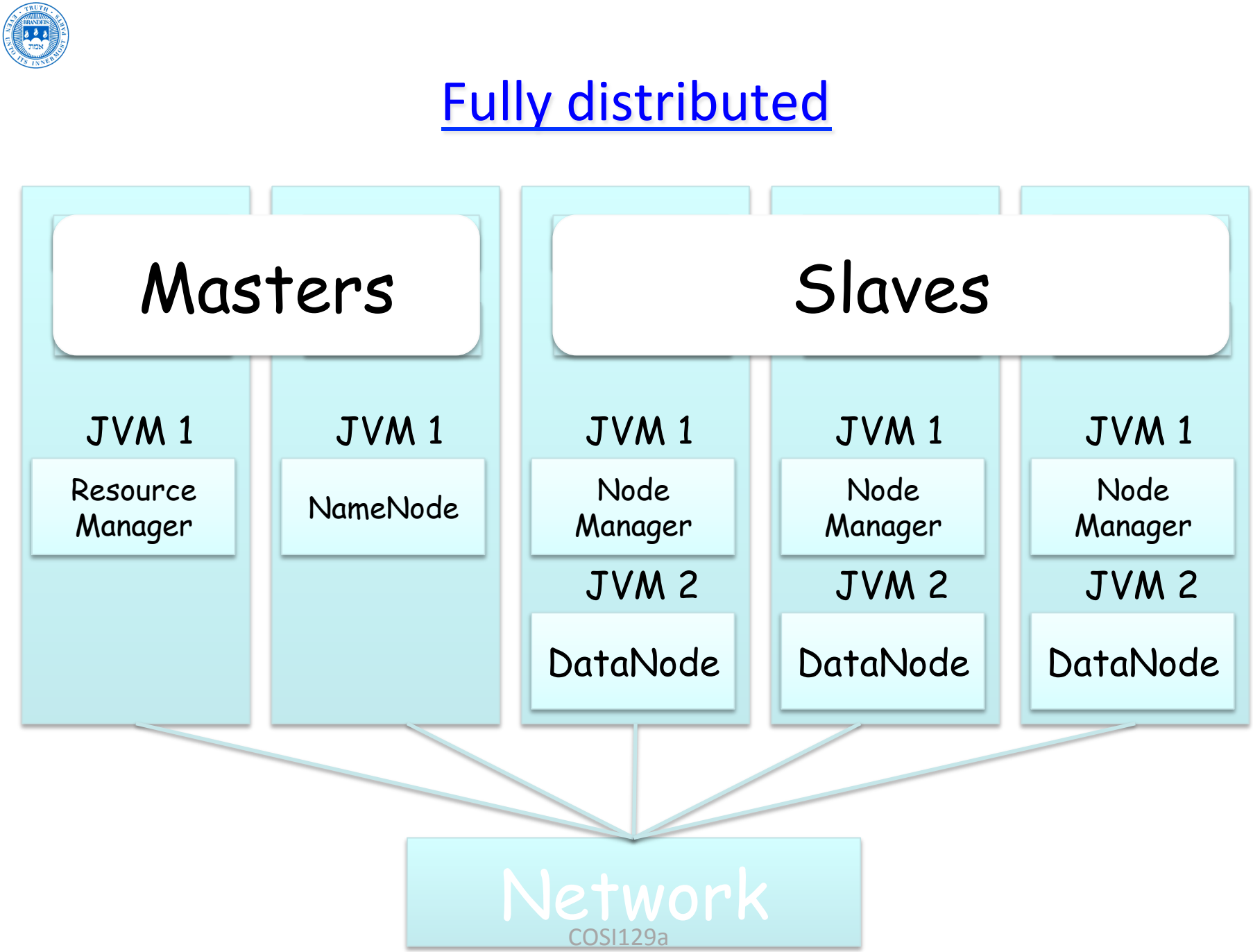
JVM 1
NameNode

JVM 1
TaskTracker
JVM 2
DataNode

JVM 1
TaskTracker
JVM 2
DataNode

JVM 1
TaskTracker
JVM 2
DataNode

Network


Fully distributed
Masters
Slaves
JVM 1
Resource Manager
JVM 1
NameNode
JVM 1
Node Manager
JVM 2
DataNode
JVM 1
Node Manager
JVM 2
DataNode
JVM 1
Node Manager
JVM 2
DataNode
Network
COSI129a

Cluster Configurations

- Two clusters (each having 3 nodes):
  - Deerstalker: hadoop01 - hadoop10
  - Akubra: hadoop11 - hadoop20
- Hadoop is setup and ready to use
- Refer to the Hadoop tutorial handout for more details

Queuing System

- Each group has its own queue for submitting its jobs
  - Queue is specified in your code or via command line
- You can manage your own queue (send/kill jobs)
  - Cannot interfere with other queues/teams
- You can have only one job per queue
  - Avoid cluster overloading
  - To submit another job you have to wait or kill the running one

Running a Program

$ bin/yarn jar share/hadoop/mapreduce/hadoop-mapreduce-examples-2.3.0-cdh5.0.2.jar wordcount -Dmapreduce.job.queuename=hadoop01 myinput myoutput

11/09/20 10:55:26 INFO mapred.FileInputFormat: Total input paths to process : 1

11/09/20 10:55:26 INFO mapred.JobClient: Running job: job_201109201045_0001

11/09/20 10:55:27 INFO mapred.JobClient: map 0% reduce 0%

11/09/20 10:55:42 INFO mapred.JobClient: map 66% reduce 0%

11/09/20 10:55:51 INFO mapred.JobClient: map 100% reduce 0%

11/09/20 10:55:54 INFO mapred.JobClient: map 100% reduce 22%

11/09/20 10:56:06 INFO mapred.JobClient: map 100% reduce 100%

11/09/20 10:56:11 INFO mapred.JobClient: Job complete: job_201109201045_0001

$ bin/hadoop fs -cat myoutput/part-*

<WORD COUNTS>

$ bin/hadoop fs -rm –r myinput

HDFS Commands

- **`$> ~/hadoop-dist> bin/hdfs dfs`**
  - [-cat [-ignoreCrc] <src> ...]
  - [-checksum <src> ...]
  - [-chgrp [-R] GROUP PATH...]
  - [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
  - [-chown [-R] [OWNER][:[GROUP]] PATH...]
  - [-copyFromLocal [-f] [-p] <localsrc> ... <dst>]
  - [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
  - [-count [-q] <path> ...]
  - [-cp [-f] [-p] <src> ... <dst>]
  - [-createSnapshot <snapshotDir> [<snapshotName>]]
  - [-deleteSnapshot <snapshotDir> <snapshotName>]
  - [-df [-h] [<path> ...]]
  - [-du [-s] [-h] <path> ...]
  - [-help [cmd ...]]
  - [-ls [-d] [-h] [-R] [<path> ...]]
  - [-mkdir [-p] <path> ...]
  - [-moveFromLocal <localsrc> ... <dst>]
  - [-moveToLocal <src> <localdst>]
  - [-mv <src> ... <dst>]
  - [-put [-f] [-p] <localsrc> ... <dst>]
  - [-renameSnapshot <snapshotDir> <oldName> <newName>]
  - [-rm [-f] [-r|-R] [-skipTrash] <src> ...]
  - [-tail [-f] <file>]
  - ...

Writing Your Own MapReduce

- General workflow:
  - Implement Mapper and Reducer Interface
  - In Mapper, override the map method
  - In Reducer, override the reduce method
- map and reduce both consume and emit pairs
- Let's see how to this in Java

Mapper in Java

```
public static YourClass {
   public static class YourMapper extends MapReduceBase implements
         Mapper<InputKeyType, InputValueType, OutputKeyType, OutputValueType> {

      public void map(InputKeyType key, InputValueType value,
                      OutputCollector<OutputKeyType, OutputValueType> output,
                       Reporter reporter) throws IOException {
        // …
        output.collect(k, v); // can be called 0 or more times
      }
}
```

These need to be Hadoop types that implement Writable. E.g., Text, IntWriteable, FloatWriteable

Reducer in Java

```
public static YourClass {
   public static class YourReducer extends MapReduceBase
    implements Reducer<InKeyType, InValueType, OutKeyType, OutValueType> {

     public void reduce(InKeyType key, Iterator<InValueType> values,
                        OutputCollector<OutKeyType, OutValueType> output,
                        Reporter reporter)
       throws IOException {
        while (values.hasNext()) {
           // ..
        }
        output.collect(key, v); // can be called 0 or more times
         // typically v is generated during the while loop
         // key won't necessarily be the same key that was passed in
      }
   }
}
```

Compiling

```
>$ mkdir yourclass_classes

>$ javac -classpath ${HADOOP_HOME}/hadoop-${HADOOP_VERSION}-
    core.jar -d yourclass_classes YourClass.java

>$ jar –cvf yourclass.jar -C yourclass_classes/ .
```