Big Data Architectures
For Machine Learning and Data Mining

Winter Semester 2019

Web Technology and Information Systems Group

Michael Voelske
<firstname>.<lastname>@uni-weimar.de

Today's Agenda

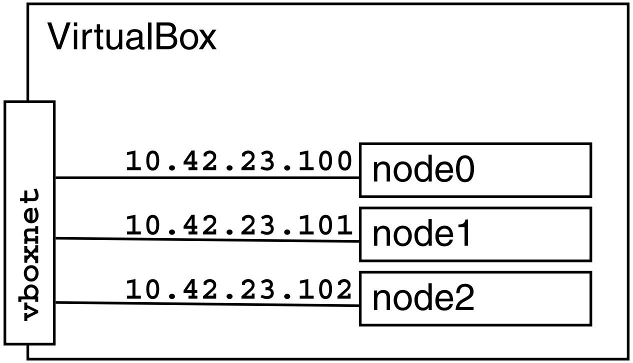
- Linux Command Line and Shell Scripting Basics II
- Introduction to Apache Hadoop: YARN and HDFS
- Installing Hadoop on the Virtual Cluster
- Distributed Processing with MapReduce

Next week: loose ends, discussion of talk topics.

A Few More Shell Scripting Basics

Virtual Cluster Revisited

After increasing the node count to 3, your virtual cluster looks like this:



Let’s log into the first node:

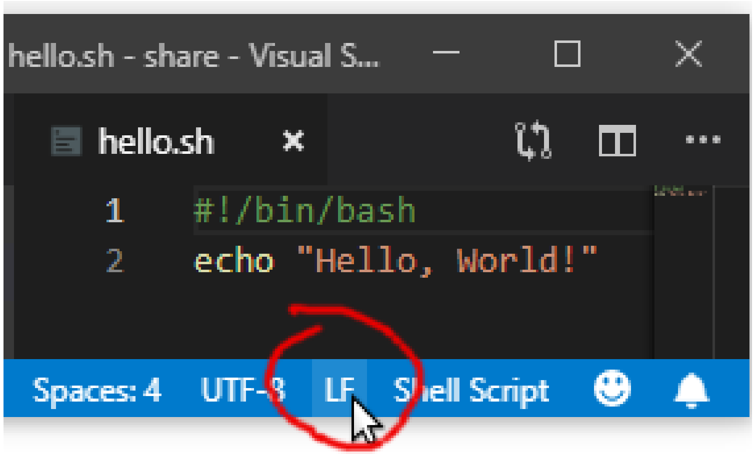
```
vagrant ssh node0
```

A Few More Shell Scripting Basics

Scripts and the Command Line

Generally, everything you can type into the command line, you can also write in a shell script, and vice versa.

Let's write a simple shell script to illustrate: Using your text editor, create a file `hello.sh` in the `share` folder.



Everything from `#` to the end of the line is a comment. The special comment `#!` ("she-bang") line tells your shell which interpreter to use.

Important note for Windows users: always make sure your editor is configured to use `LF` line terminators, not `CR+LF`.

We must make the script executable before we can run it:

```
chmod +x ~/share/hello.sh
~/share/hello.sh
```

A Few More Shell Scripting Basics

Variables

`=` defines variables, and `$` expands them. The variable name can be enclosed in curly braces if needed.

```
MYVAR="hello"
echo $MYVAR
echo ${MYVAR}world
```

Variables can be recursively composed of other variables. The syntax `$()` inserts the output of a subprocess. `$`-expansion happens in double-quoted strings, but not in single-quoted strings.

```
MYVAR="${MYVAR}, world.  The current time is $( date -R )"
echo ${MYVAR} 'single-quoted:  ${MYVAR} $( date )'
```

A Few More Shell Scripting Basics

Special Variables, and the Environment

Some special variables control aspects of your shell's behavior. For example: `$PATH` controls where the shell will look for executable commands.

```
echo ${PATH}
```

Separate new search paths with the colon character:

```
PATH=${PATH}:${HOME}/share
```

Now we can call our earlier script from anywhere:

```
hello.sh
```

The `export` command makes a variable part of the *environment*, and thus visible to sub-processes (you can set the value simultaneously). The `env` command prints all currently defined environment variables

```
export MYVAR
export MYVAR2=test
env
```

A Few More Shell Scripting Basics

Functions

Functions are defined like this:

```
function_name() {
    echo "Do this"
    echo "Then do this..."
}
```

Without arguments, functions are called simply by writing the name:

```
function_name
```

A Few More Shell Scripting Basics

Functions

Functions are defined like this:

```
function_name() {
   echo "Do this"
   echo "Then do this..."
}
```

Without arguments, functions are called simply by writing the name:

```
function_name
```

Functions can access their arguments using the special variables `$1`, `$2`, ...

```
say_hello() { echo "Hello, $1!"; }
say_hello Bob
```

A Few More Shell Scripting Basics

“Here” Documents

A *Here Document* is a file literal—a section of your code or command line that describes the contents of a separate file or input stream. You can use it as shorthand syntax for writing a command along with the input it should receive (e.g. on stdin):

```
command-with-options    <<END_MARKER
```
(arbitrary input lines go here...)
```
END_MARKER
```

A Few More Shell Scripting Basics

“Here” Documents

A *Here Document* is a file literal—a section of your code or command line that describes the contents of a separate file or input stream. You can use it as shorthand syntax for writing a command along with the input it should receive (e.g. on stdin):

```
command-with-options    <<END_MARKER
```
(arbitrary input lines go here...)
```
END_MARKER
```

For example:

```
wc -l <<EOF
I wonder
how many lines
these are?
EOF
```

Note: The “End Marker” can be any token, but EOF (for end of file) is a common convention.

A Few More Shell Scripting Basics

“Here” Documents (continued)

A common application in shell scripts, along with the `cat` command, makes a readable shorthand for creating a file with specific contents defined in-place:

File: <heredoc-test.sh>

```
#!/bin/bash

# Make the script create a new file with specific contents:
cat > /tmp/my-new-file.txt <<EOF
these are the contents
of the new file
written right here
EOF
```

A Few More Shell Scripting Basics

Exit Codes and Booleans

Every shell command has an exit code. By convention, exit code zero means success, everything else indicates an error. The special variable `$?` always contains the exit code of the previous command.

```
echo hello ; echo $?
cat /no/such/file ; echo $?
```

Which error a specific non-zero exit code corresponds to can differ between commands. In your own scripts, you can use the `exit` command to return an exit code of your choice.

A Few More Shell Scripting Basics

Exit Codes and Booleans

Every shell command has an exit code. By convention, exit code zero means success, everything else indicates an error. The special variable `$?` always contains the exit code of the previous command.

```
echo hello ; echo $?
cat /no/such/file ; echo $?
```

Which error a specific non-zero exit code corresponds to can differ between commands. In your own scripts, you can use the `exit` command to return an exit code of your choice.

Exit codes also have an interpretation as boolean values. By convention, zero means “true,” and everything else means “false.” Boolean operators like `||` and `&&` are defined accordingly.

```
cat unknown-file || echo "not found :-("
touch unknown-file # (this creates the file)
cat unknown-file && echo "found :-)"
```

A Few More Shell Scripting Basics

Conditionals and `test`

This can be used in control structures like `if`

```
if cat unknown-file ; then
  echo yes
else
  echo no
fi
```

With the `[]` syntax you can write a variety of conditional expressions; this is shorthand for the `test` command. See `man test` for what else it can do.

```
A=2 ; B=1
if [ "$A" -gt "$B" ] ; then
  echo "A is bigger." ;
fi
```

A Few More Shell Scripting Basics

Sourcing versus Executing Scripts

Let's say we have a file with the following contents:

File: <test.sh>

```
#!/bin/bash
export TESTVARIABLE=1234
echo "test script ends."
```

Executing it spawns a subprocess (another `bash`, as determined by the shebang line) which executes your script, which then exits and returns the control flow to your original shell. Changes in the environment do not propagate from subprocess to parent process:

```
./test.sh
echo $TESTVARIABLE
```

On the other hand, *sourcing* the file causse your current shell to interpret it, without spawning a subshell. Changes to the environment thus do affect your shell.

```
source test.sh
echo $TESTVARIABLE
```

Note: a single `.` is shorthand for `source`.

A Few Useful Networking Utilities

Setup

For the next few slides we use two terminals, each connected to a different node. To this end, open two terminals in the folder with the `Vagrantfile`, and run:

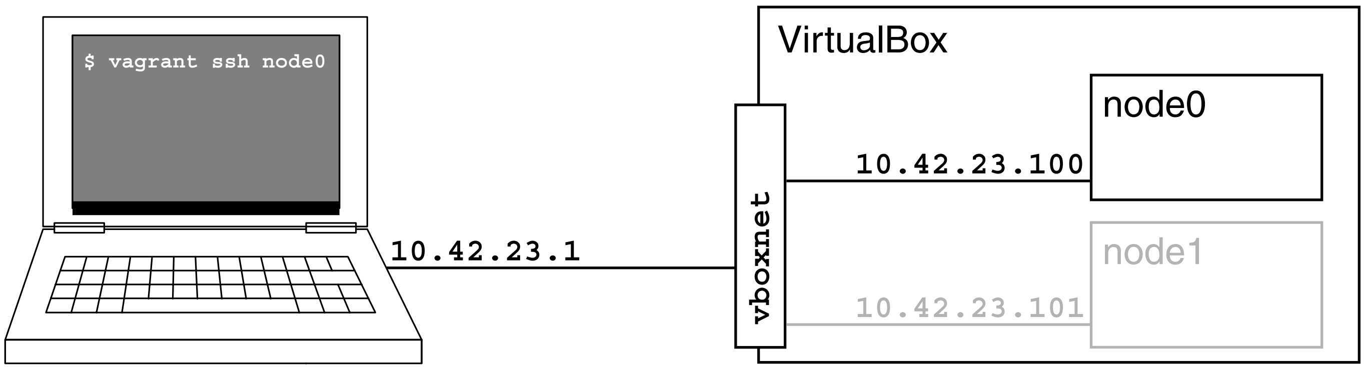
First terminal:

```
vagrant ssh node0
```

Second terminal:

```
vagrant ssh node1
```

Recall the virtual cluster network:



A Few Useful Networking Utilities

ICMP Ping: Is That Host Up?

The `ping` command uses the low-level (IP layer) Internet Control Message Protocol to

1. send a simple network packet to some other host
2. ask it to send a packet back

Example:

On `node0`:

```
ping node1
```

Ping may be unsuccessful if

- The other host is down
- There's something wrong with the network connection on either host, or somewhere in between
- Either host filters ICMP packets
- Some router along the way filters ICMP packets...

A Few Useful Networking Utilities

Netcat: the TCP / UDP Swiss Army Knife

Netcat operates on the higher-level Transport layer, and can send arbitrary data over TCP (the default) or UDP streams. It is typically used to test if a server is reachable on some specific port, but it can function both as client and as server.

Example:

On `node0`, start netcat in server mode.

```
nc -v -l -p 12345
```

On `node1`, connect to the server.

```
nc -v node0 12345
```

Whatever you type in either terminal now gets sent to the other side.

A Few Useful Networking Utilities

Netcat: the TCP / UDP Swiss Army Knife

Netcat operates on the higher-level Transport layer, and can send arbitrary data over TCP (the default) or UDP streams. It is typically used to test if a server is reachable on some specific port, but it can function both as client and as server.

Example:

On `node0`, start netcat in server mode.

```
nc -v -l -p 12345
```

On `node1`, connect to the server.

```
nc -v node0 12345
```

Whatever you type in either terminal now gets sent to the other side.

You can even use this to copy files over the network:

Server sends a file:

```
echo "test data" > test-file
nc -l -p 1234 < test-file
```

Client receives it:

```

nc -w0 node0 1234 > received
```

A Few Useful Networking Utilities

Curl: Testing HTTP Connections

The `cURL` ("client URL") program can transfer data to and/or from a server, e.g. to test connections or download or upload files. Curl supports a wide range of application layer protocols to do this (see the `man` page), but is most commonly used for testing HTTP servers.

To try this out, let's start up a simple HTTP server (the Python standard library happens to provide one):

First, on `node0`:

```
python3 -m http.server
```

```
Serving HTTP on 0.0.0.0 port 8000 (http:// 0.0.0.0:8000
/) ...
10.42.23.102 - - [...]  "GET / HTTP/1.1" 200 -
```

Then, on `node1`:

```
curl node0:8000
```

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html> [...]
```

As you can see in the server's output, `curl` sends a GET request by default, but all kinds of other things are possible. For example, `curl -I` sends a HEAD request, and displays the response headers; `curl -X FOO` submits a request with arbitrary method (in this case the nonexistant method "`FOO`," which most servers should answer with an error message).

A Few Useful Networking Utilities
Summary

The three aforementioned utilities allow us to test our network connections across the internet protocol suite:

Utility	IP Layer	Supported Protocols	Typical Use Case
`curl`	Application	HTTP(S), (S)FTP, SMB(S), IMAP(S), SMTP(S), ...	Is the server program working correctly?
`nc`	Transport	TCP & UDP	Is the server program running?
`ping`	Internet	IP	Is the machine even on?

FoxyProxy

Connecting Your Web Browser to the Virtual Network

To actually see the web page produced by that Python `http.server`, you'll have to connect the browser running on your host operating system to the server running inside VirtualBox. We'll achieve this with the help of:

1. The SSH client, which can act as a dynamic SOCKS proxy, and route connections from your browser into the VirtualBox network (see [en.wikipedia.org/wiki/SOCKS] and [en.wikibooks.org/wiki/OpenSSH/Cookbook/Proxies_and_Jump_Hosts]).
   Run the `connect-proxy.sh` script in the seminar repository to establish an SSH connection with the appropriate parameters.
2. The FoxyProxy browser extension, which can tells your browser to use a specific proxy based on pattern-matching on the URL.
   After installing the extension, find the "Import" button in the FoxyProxy settings, and select the appropriate settings for your browser (Firefox or Chrome).

With the Python `http.server` still running on `node0`, and after doing both steps above, type "`node0:8000`" into your browser's address bar.


Hadoop.

Hadoop

[hadoop.apache.org]

- Started in 2004 by Yahoo
- Open-Source implementation of Google MapReduce, Google Filesystem and Google BigTable
- Apache Software Foundation top level project
- Written in Java

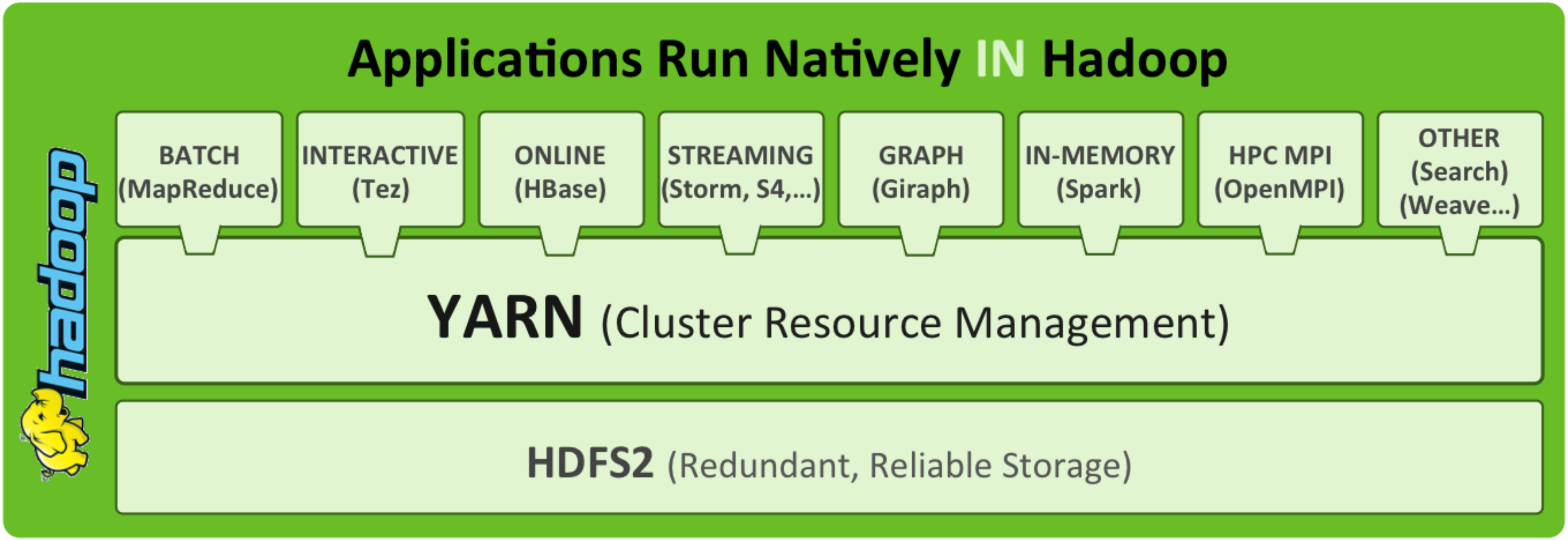
Hadoop

Basic Ideas

- Scale out, not up.
  - many individual nodes
  - cheap commodity hardware instead of supercomputers
  - fault-tolerance, redundancy
- Bring the program to the data.
  - storage and data processing on the same node
  - local processing (network is the bottleneck)
- Process large datasets sequentially.
  - Limited random access capability
  - Simpler distributed file system semantics
- Hide system-level details
  - User doesn't need to know what code runs on which machine

Hadoop

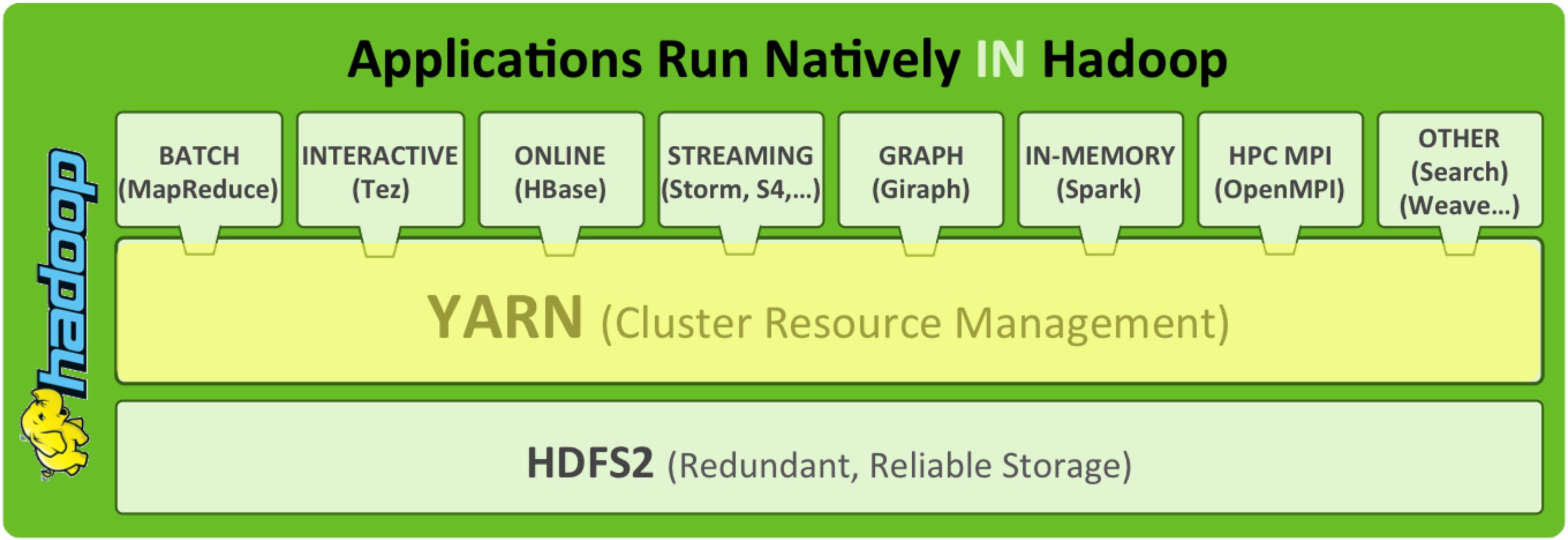
Hadoop 2.0 Ecosystem



[http://hortonworks.com/hadoop/yarn/]

Hadoop

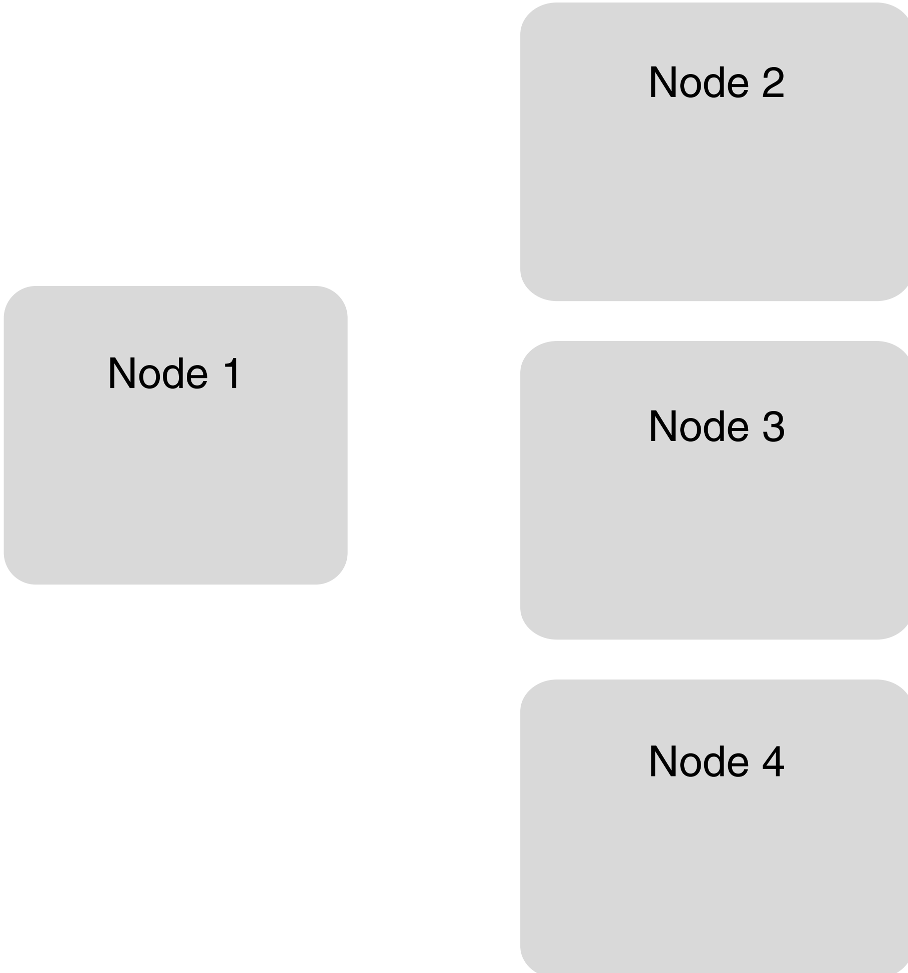
Hadoop 2.0 Ecosystem



[http://hortonworks.com/hadoop/yarn/]

Hadoop YARN

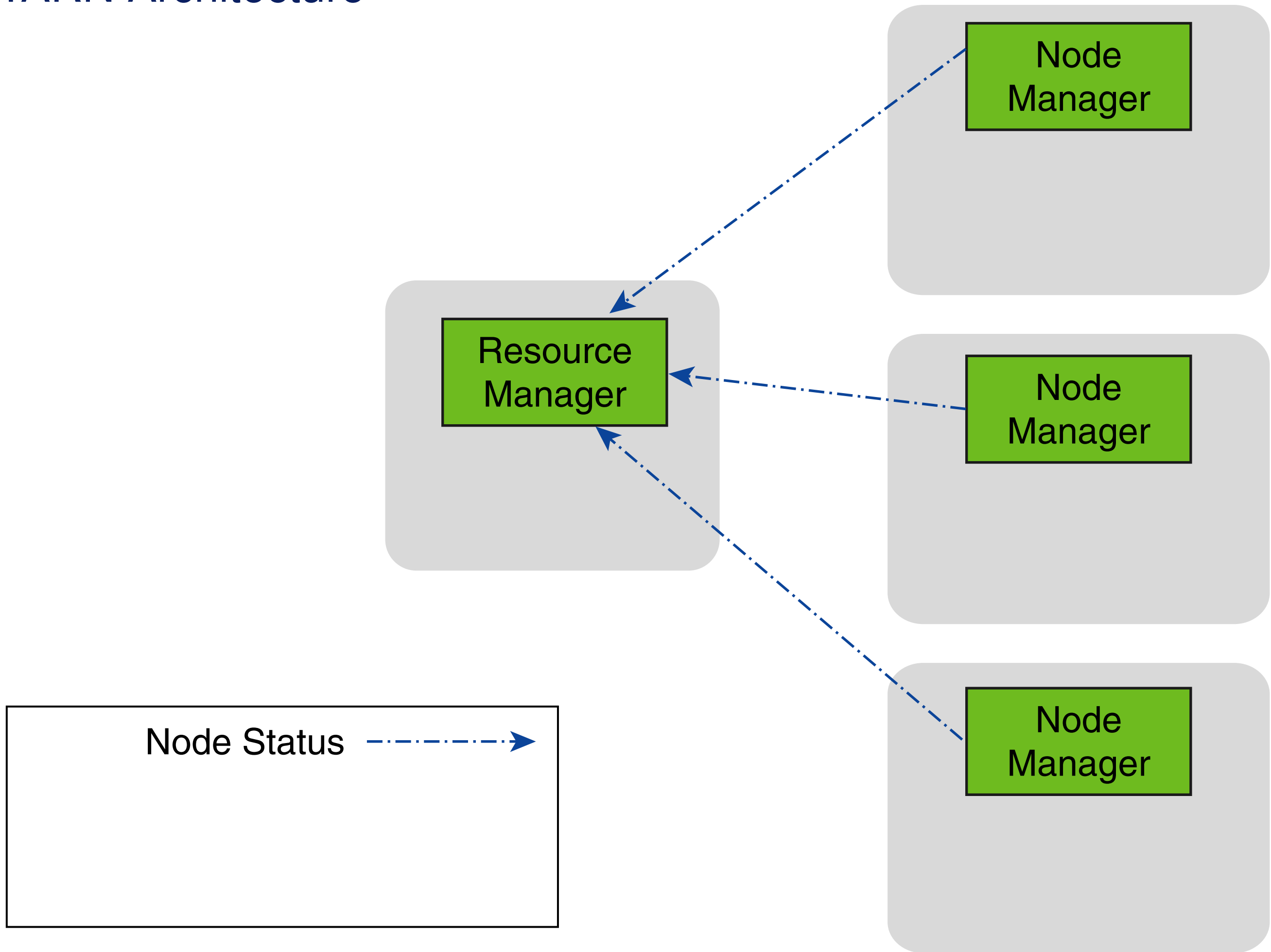
YARN Architecture



[https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html]

Hadoop YARN

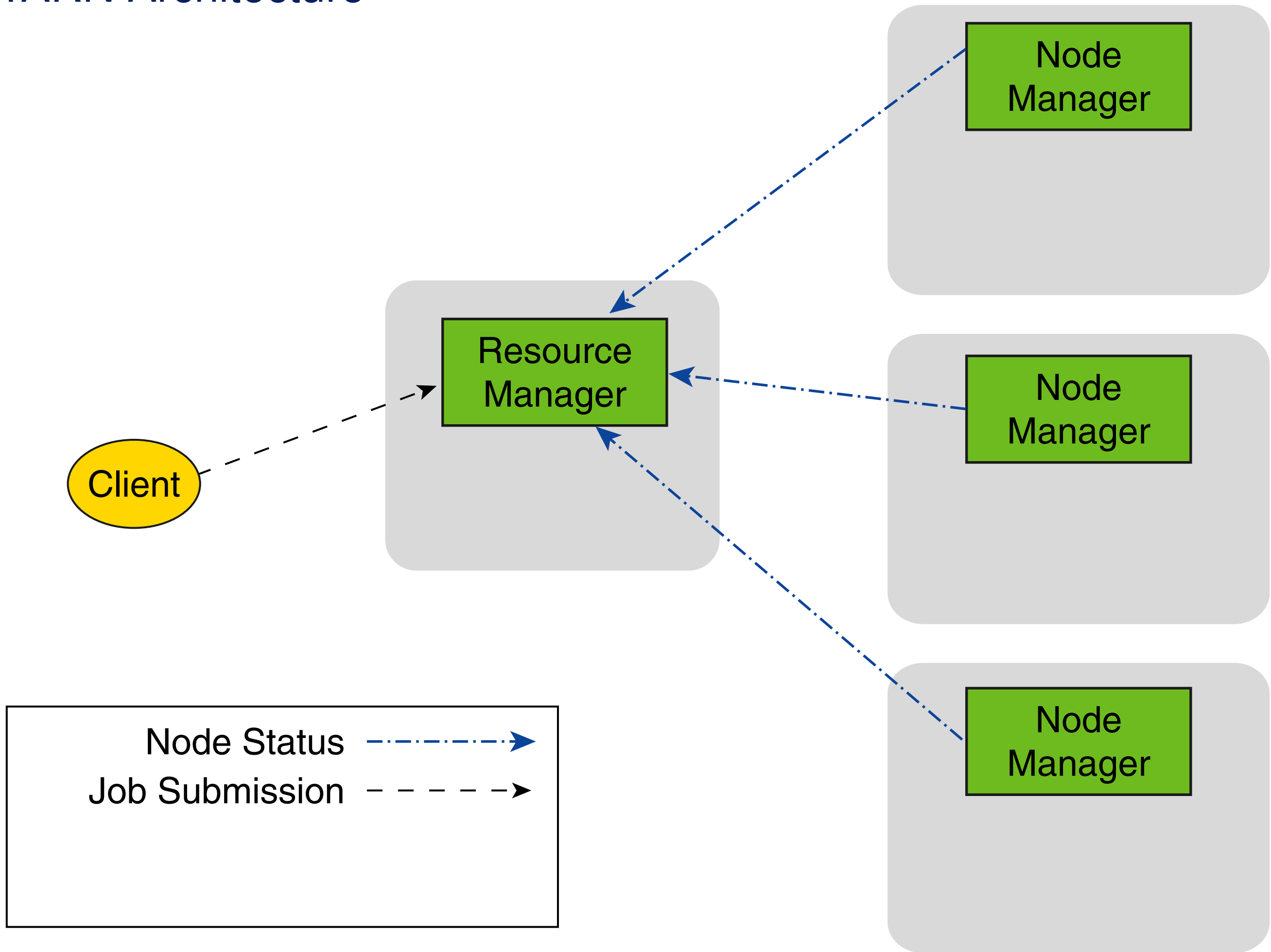
YARN Architecture



[https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html]

Hadoop YARN

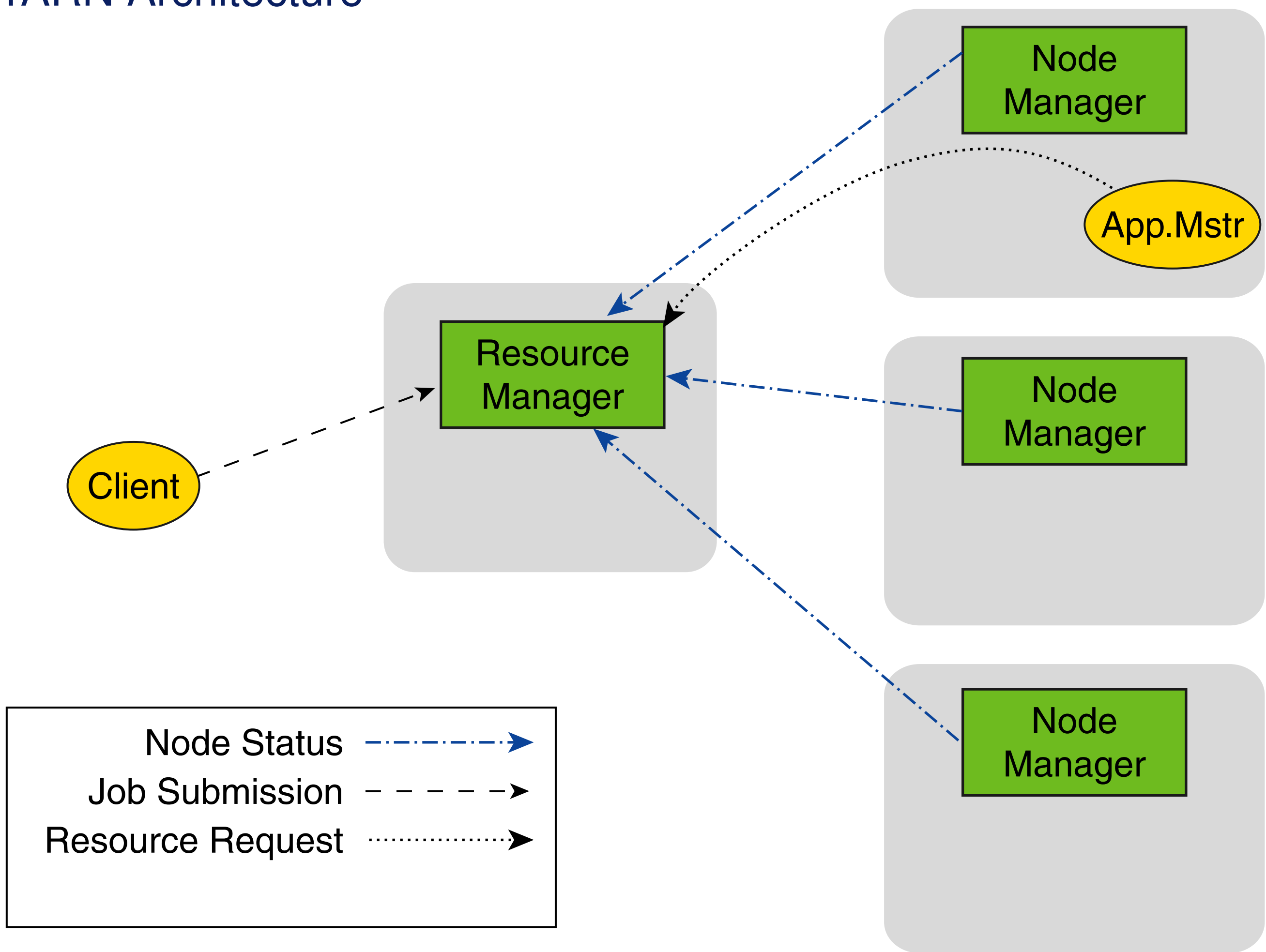
YARN Architecture



[https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html]

Hadoop YARN

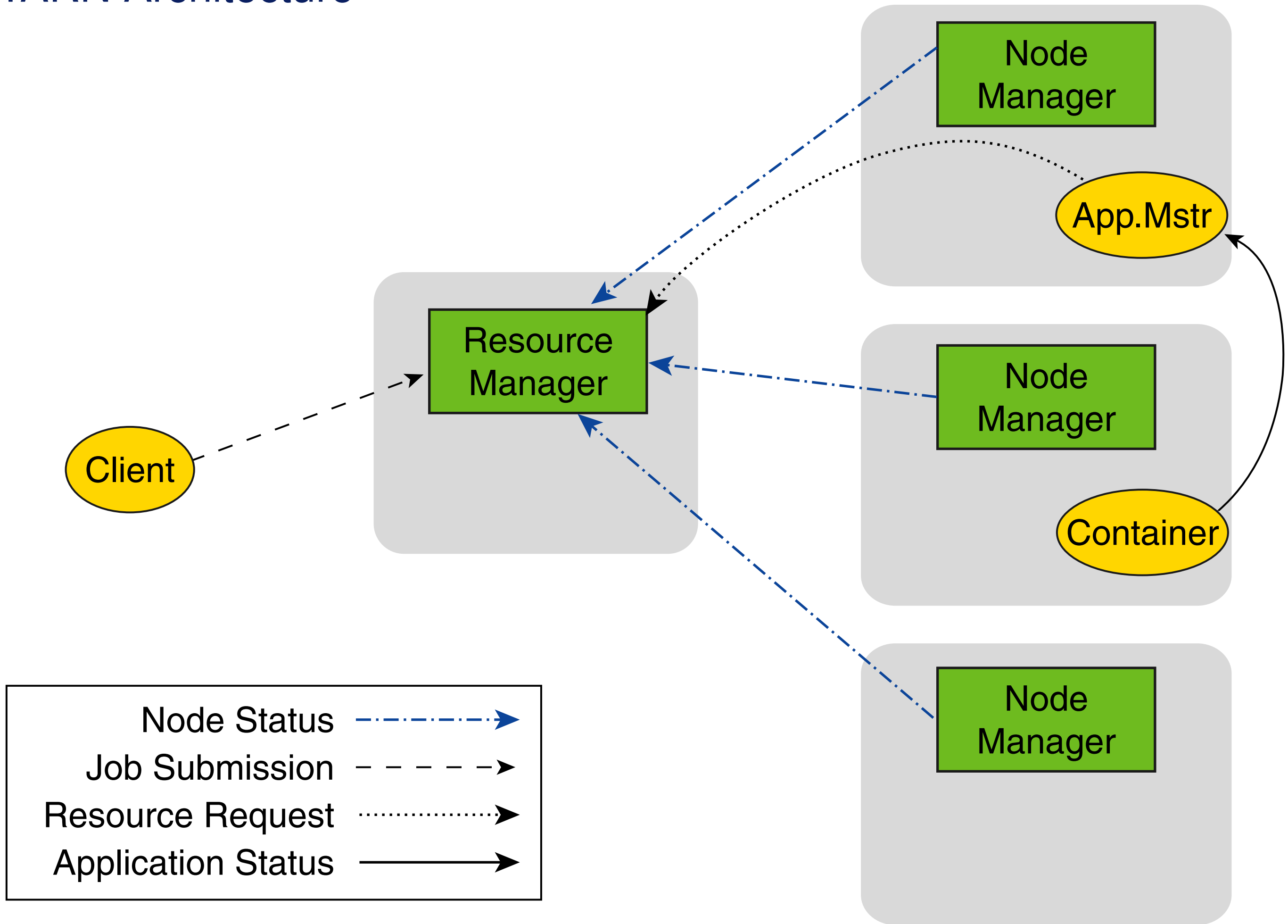
YARN Architecture



[https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html]

Hadoop YARN

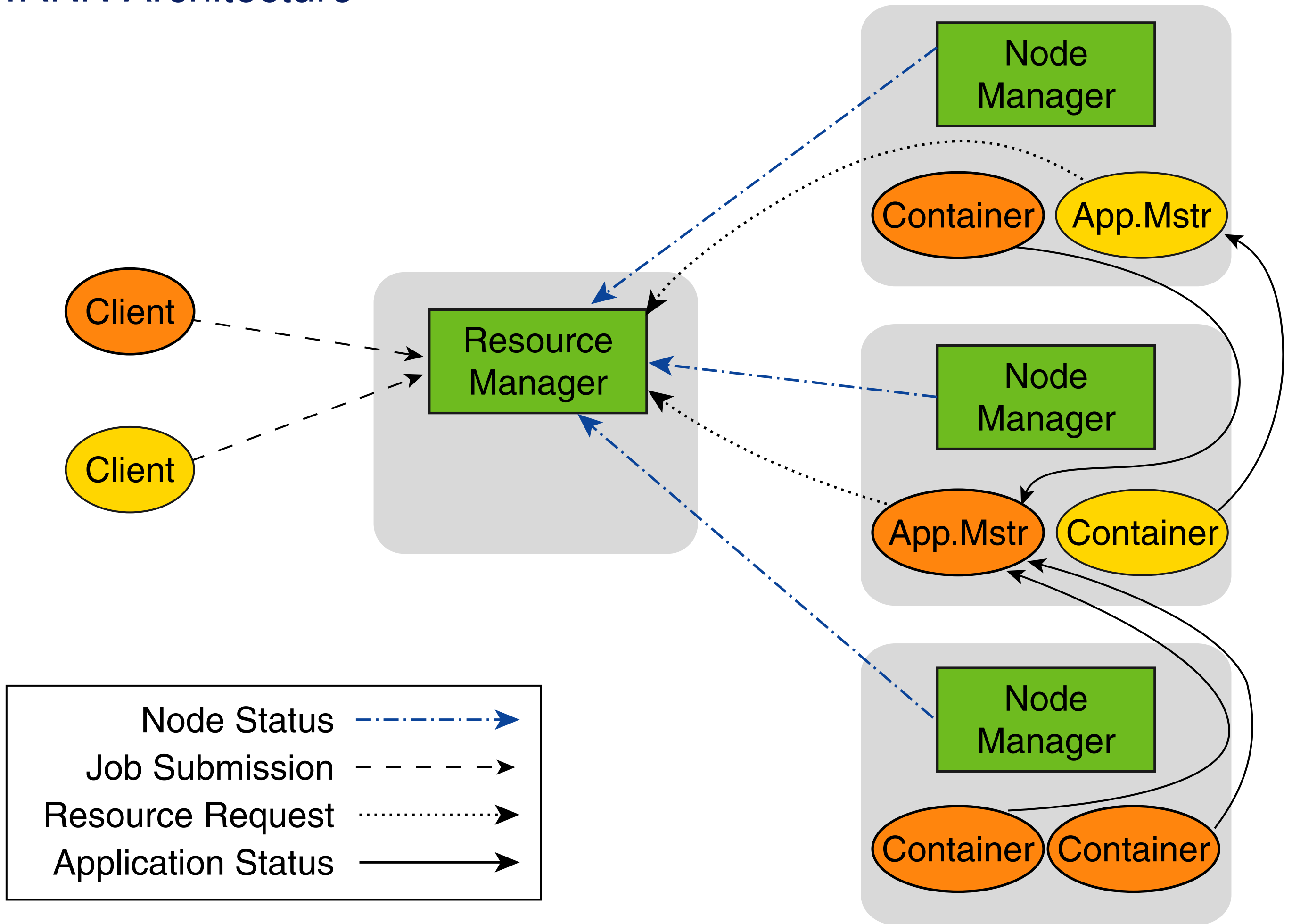
YARN Architecture



[https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html]

Hadoop YARN

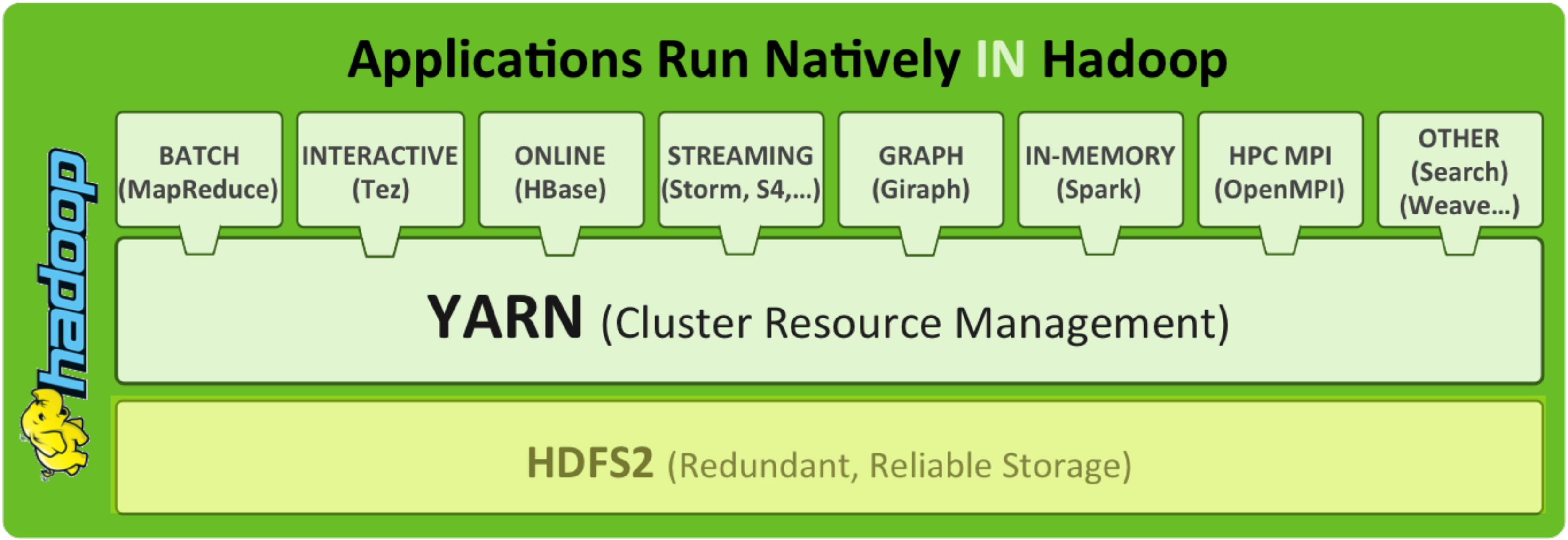
YARN Architecture



[https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html]

Hadoop HDFS

Distributed File System



[http://hortonworks.com/hadoop/hdfs/]

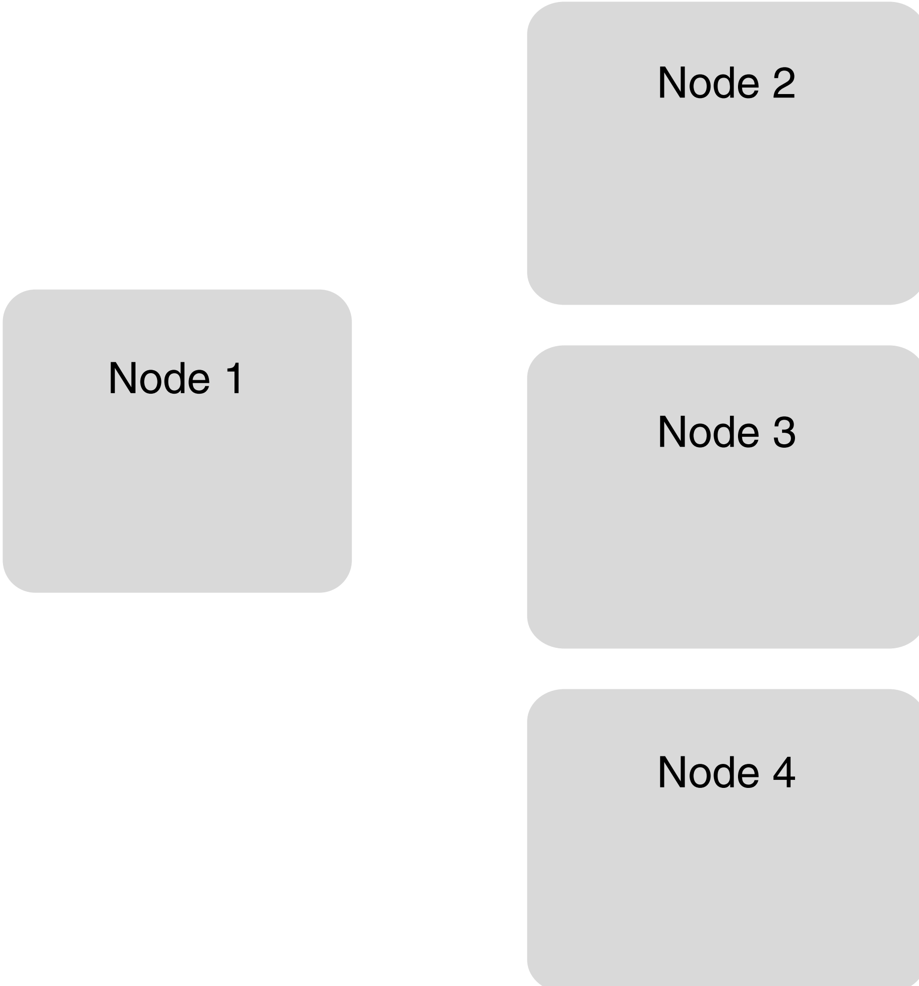
Hadoop HDFS

HDFS Overview

- Designed for storing large files
- Files are split in blocks
- Integrity: Blocks are checksummed
- Redundancy: Each block stored on multiple machines
- Optimized for sequentially reading whole blocks
- Daemon processes:
  - NameNode: Central registry of block locations
  - DataNode: Block storage on each node

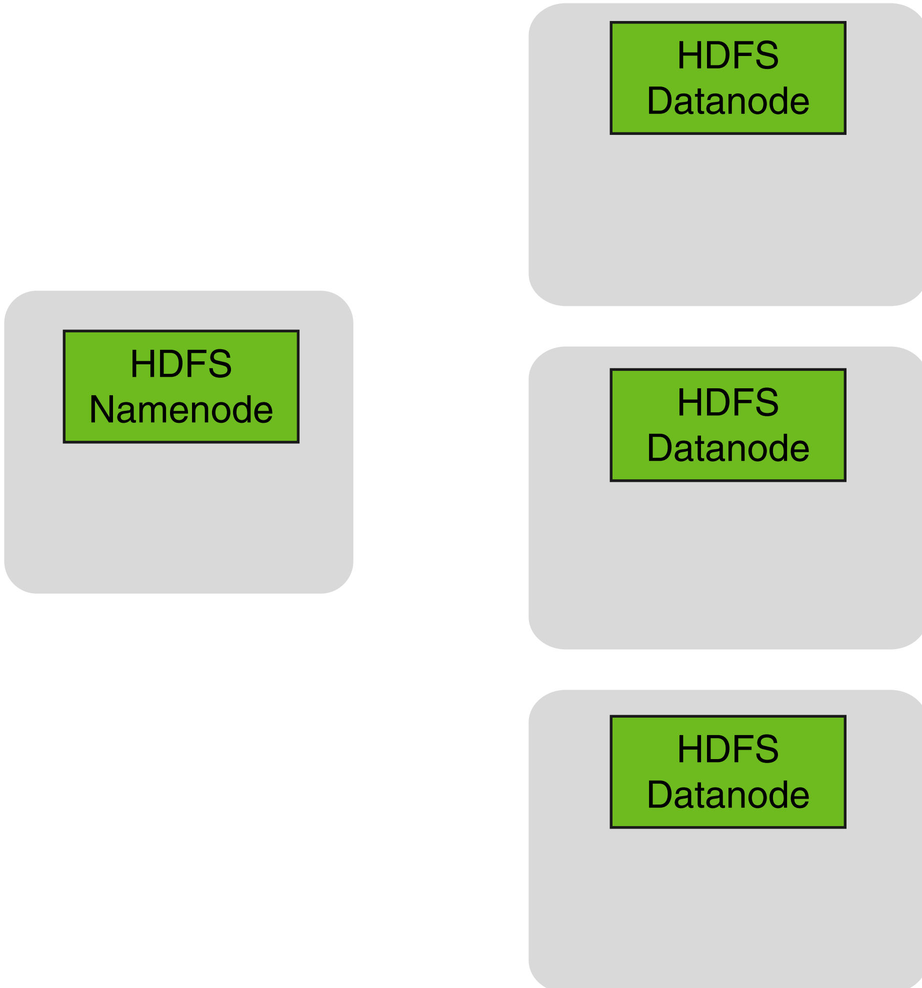
Hadoop HDFS

HDFS Architecture and Reading Files



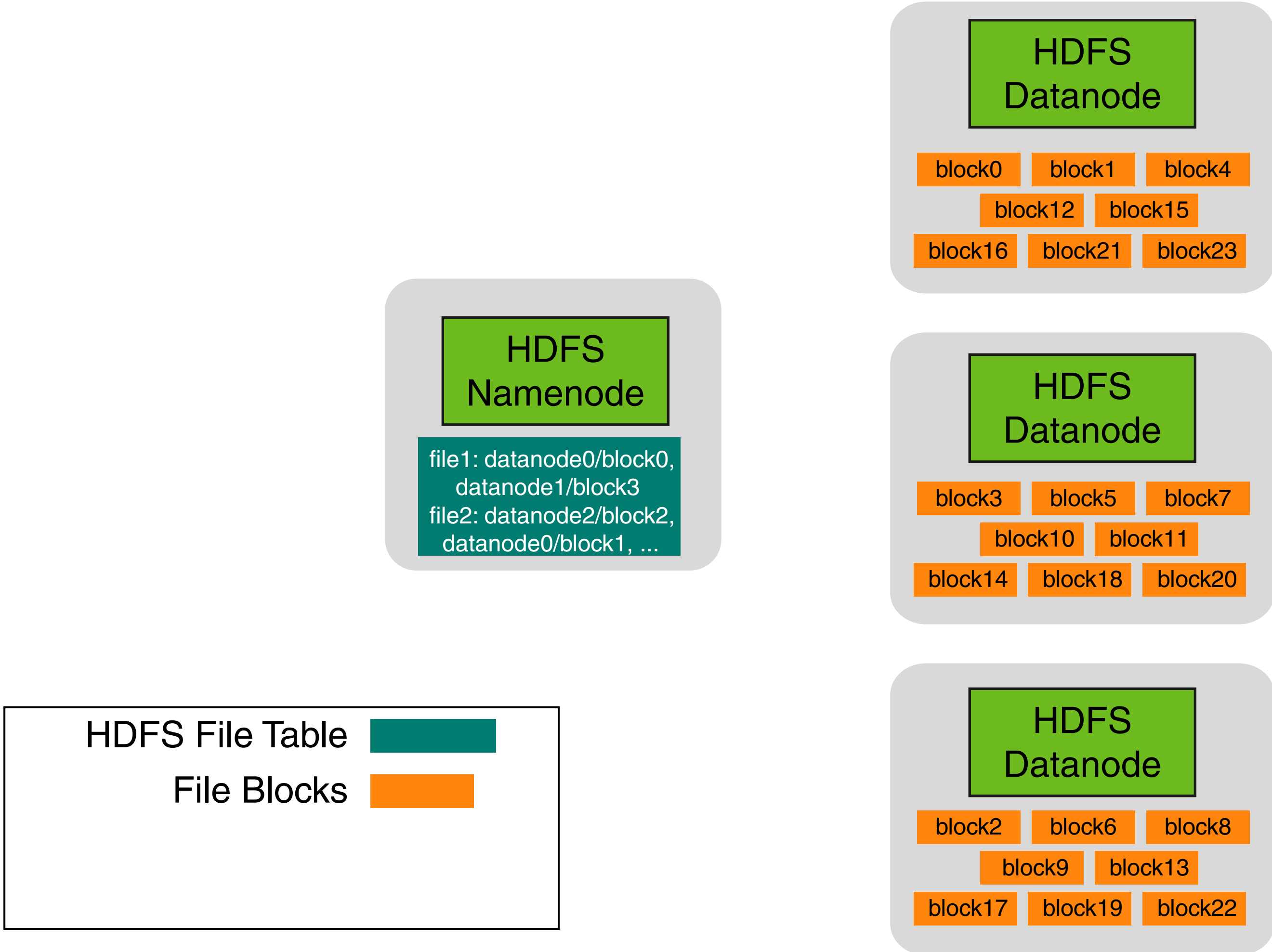
Hadoop HDFS

HDFS Architecture and Reading Files



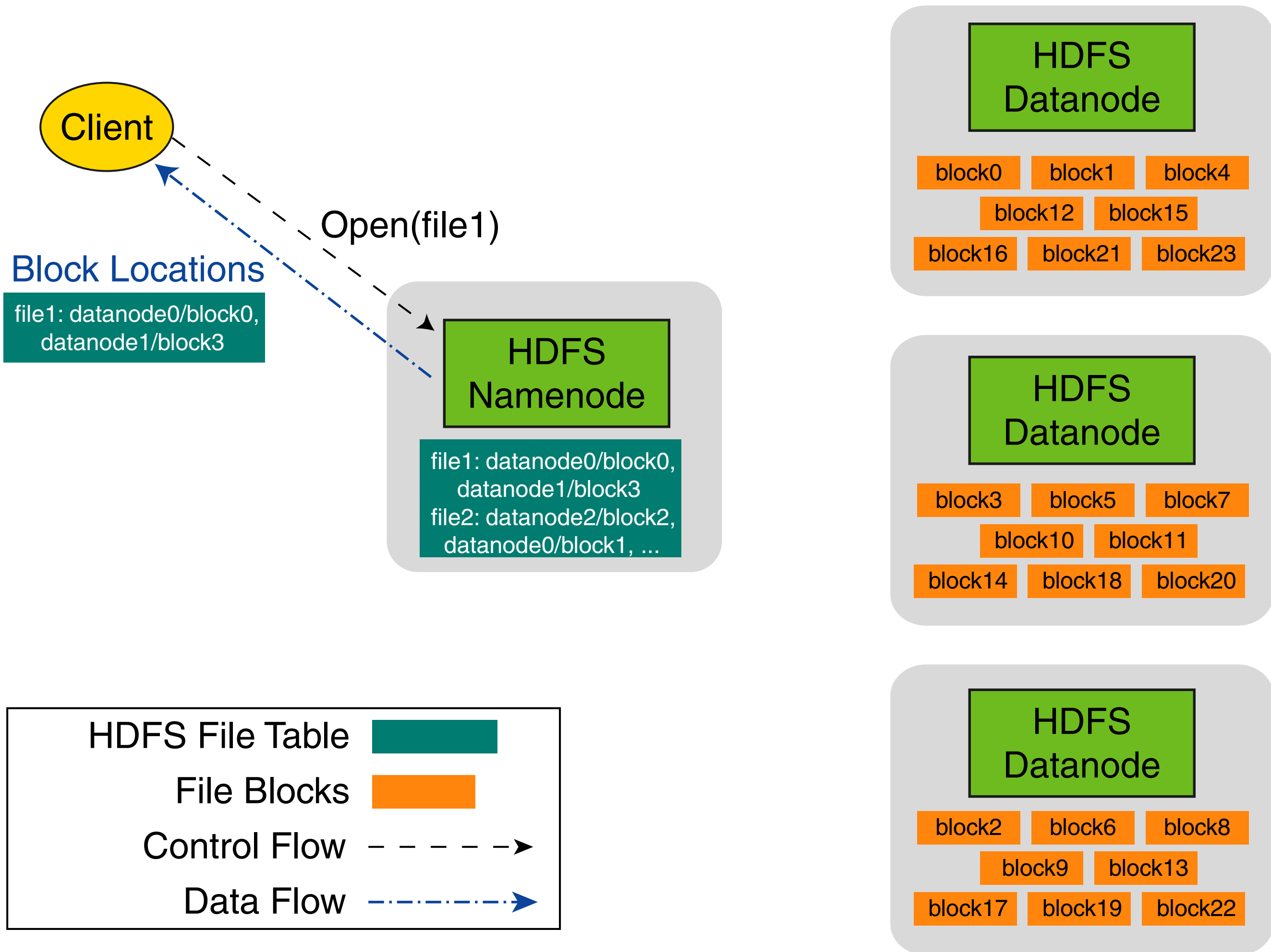
Hadoop HDFS

HDFS Architecture and Reading Files



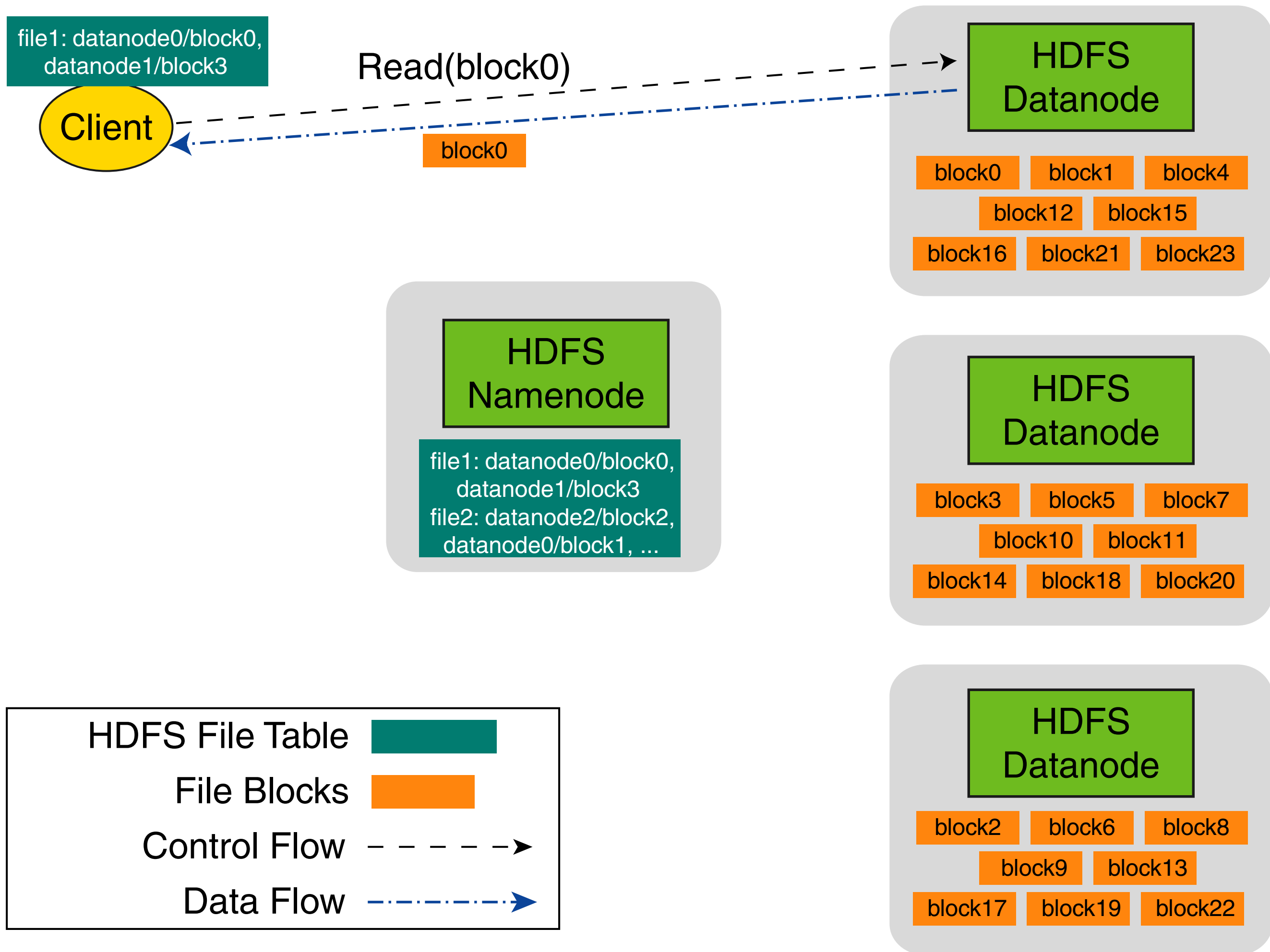
Hadoop HDFS

HDFS Architecture and Reading Files



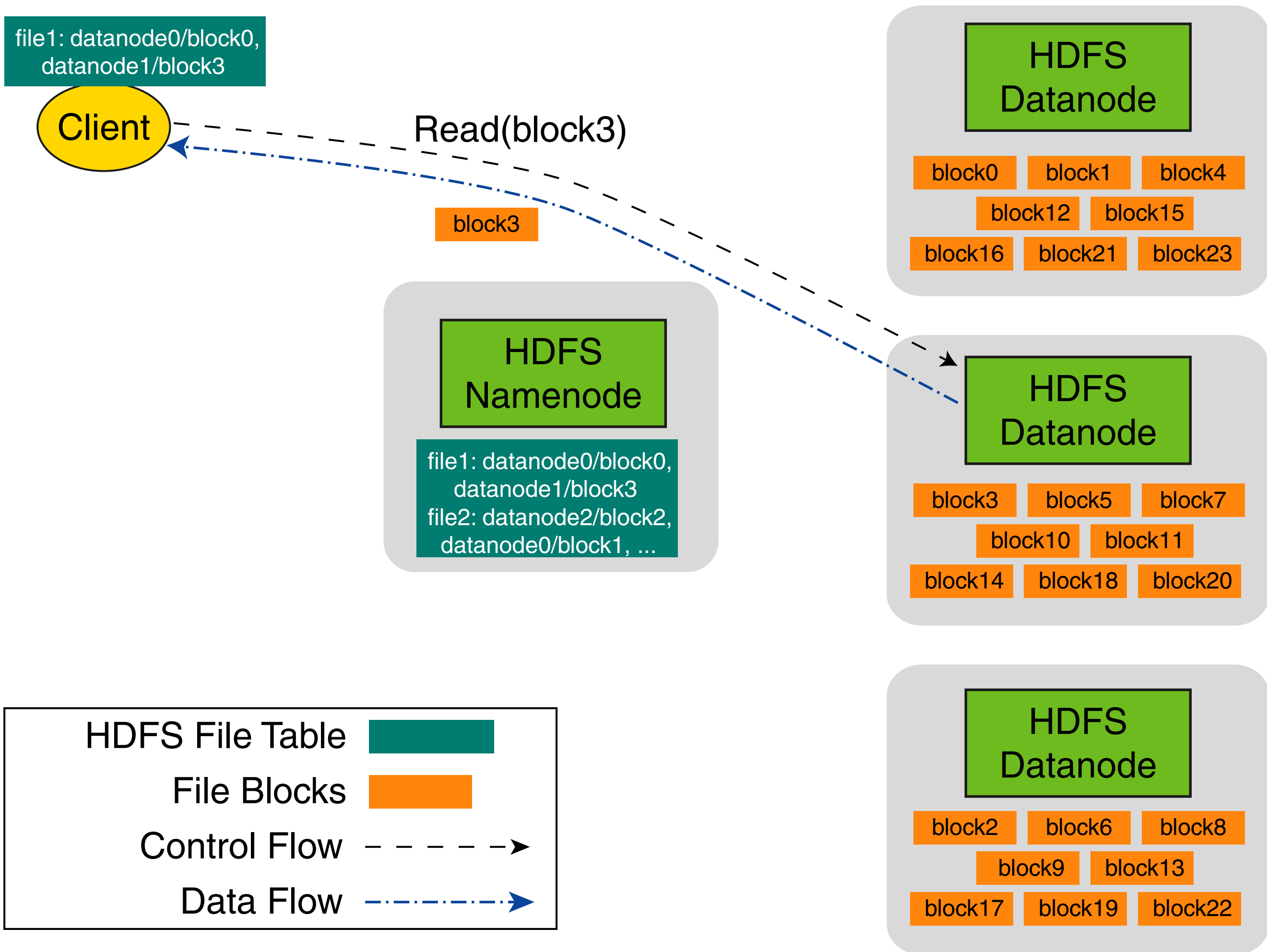
Hadoop HDFS

HDFS Architecture and Reading Files



Hadoop HDFS

HDFS Architecture and Reading Files




Installing Hadoop.

Hadoop Installation

Update the Vagrantfile

The Vagrantfile provided with the seminar repository already contains a facility for running scripts on the virtual machines, but it must first be activated:

```
# If set to true, run
# provisioning
$run_scripts = true
```

Once done, every file ending in `.sh` inside the `scripts` folder will be run on every virtual machine during provisioning, in alphabetical order. You can manually force the scripts to run using:

```
vagrant provision
```

Hadoop Installation

Update the Vagrantfile

The Vagrantfile provided with the seminar repository already contains a facility for running scripts on the virtual machines, but it must first be activated:

```
# If set to true, run
# provisioning
$run_scripts = true
```

Once done, every file ending in `.sh` inside the `scripts` folder will be run on every virtual machine during provisioning, in alphabetical order. You can manually force the scripts to run using:

```
vagrant provision
```

A very simple example script is already there:

`scripts/00-test-script.sh`

```
#!/bin/bash
echo "This is a setup script that will be run on all nodes."
echo "Currently running on $( hostname )"
```

Hadoop Installation

Hadoop Installation Script

Download the Hadoop installation script from the seminar web page, and place it into the scripts folder.

We'll walk through step by step. In a nutshell, the script:

1. Installs Java
2. Downloads an archive with the hadoop distribution and unpacks it in a standard location in the VM's file system
3. Creates a bunch of configuration files with settings appropriate to the virtual cluster
4. Starts the hadoop daemon services:
   - HDFS Namenode (with a new, empty file table) and Yarn ResourceManager on `node0`
   - HDFS Datanode and Yarn Nodemanager on all nodes (including `node0`)

Hadoop Installation

Hadoop Web UIs

Once the installation is complete, and all the Hadoop services are running, you can access the web-based user interfaces with your browser (with the help of the SSH-based proxy).

The **HDFS Web UI** [node0:9870] shows the state of the distributed file system, including the number of blocks and files, available space, and missing blocks.

The **YARN Web UI** [node0:8088] shows the state of the cluster resource manager, including available CPUs and memory, and currently running jobs and their resource usage.

Note: default ports may differ between Hadoop versions; when in doubt, check the installation script for which version is being installed, and refer to the corresponding documentation online.


HADOOP IS INSTALLED...
NOW WHAT?
@hadoopmemes

MapReduce

Problem

- Collecting data is easy and cheap
- Evaluating data is difficult

Solution

- Divide and Conquer
- Parallel Processing

MapReduce

MapReduce Steps

1. **Map** Each worker applies the `map()` function to the local data and writes the output to temporary storage. Each output record gets a key.

2. **Shuffle** Worker nodes redistribute data based on the output keys: all records with the same key go to the same worker node.

3. **Reduce** Workers apply the `reduce()` function to each group, per key, in parallel.

The user specifies the `map()` and `reduce()` functions.

MapReduce

Example: Counting Words

Mary had a
little lamb

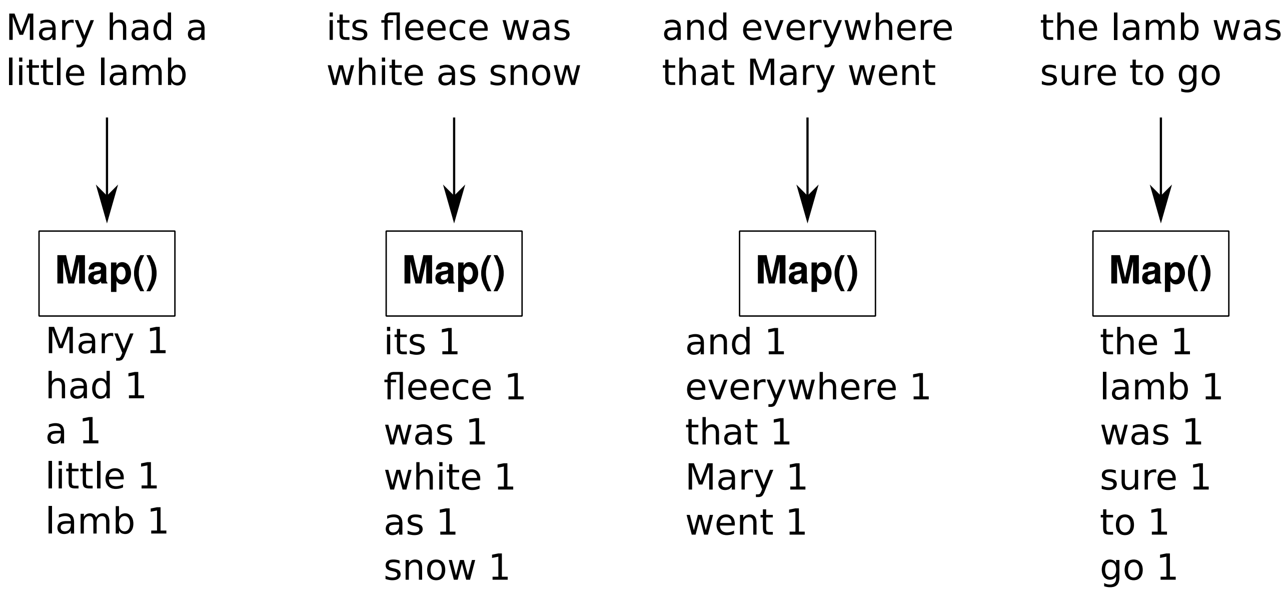
its fleece was
white as snow

and everywhere
that Mary went

the lamb was
sure to go

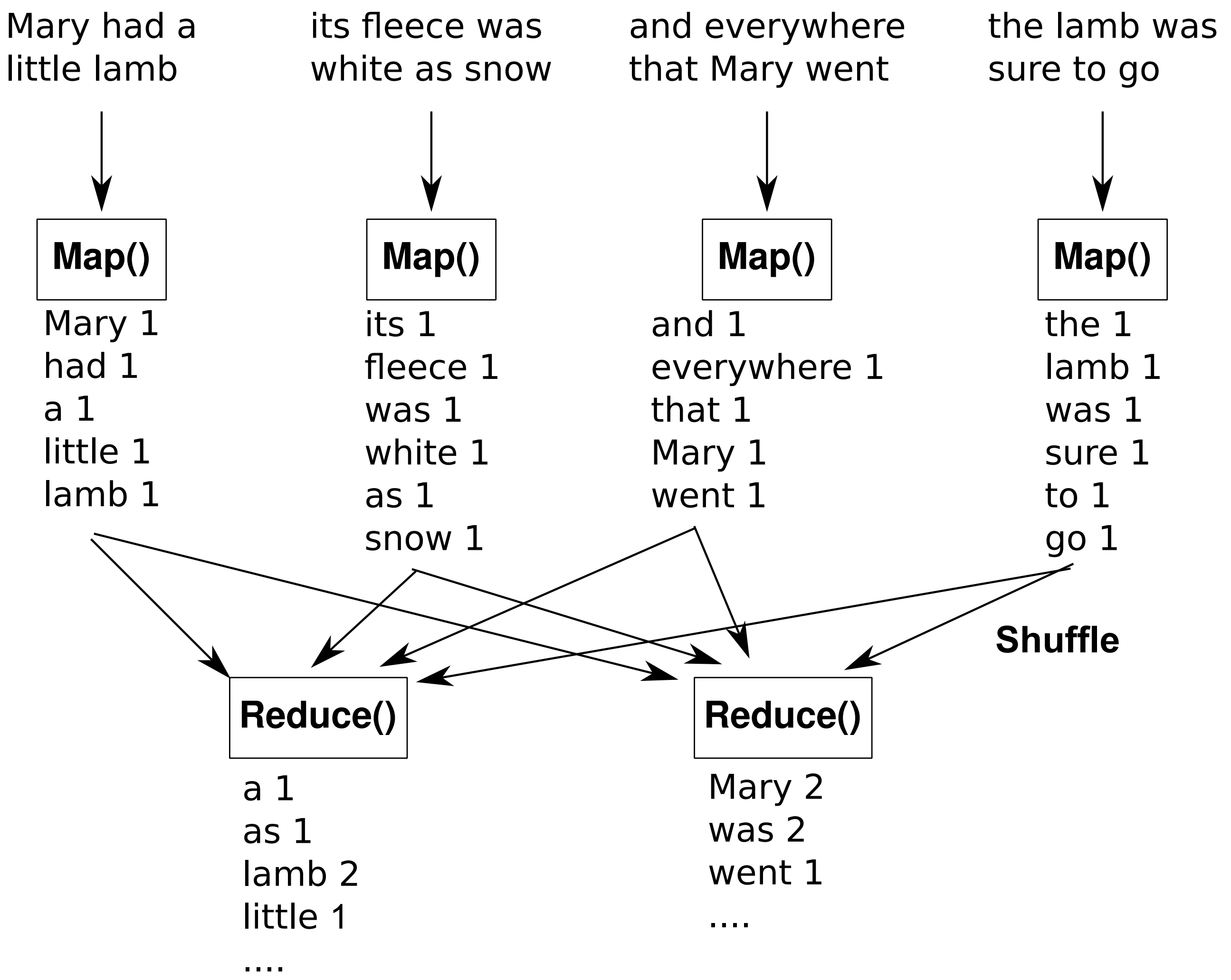
MapReduce

Example: Counting Words



MapReduce

Example: Counting Words



MapReduce

Data Representation with Key-Value Pairs

Map Step:

Each mapper independently produces a list of key-value pairs.

`Map(k1,v1)` $\rightarrow$ `list(k2,v2)`

Sorting and Shuffling:

Across all mapper outputs, all pairs with the same key are grouped together, forming one group per key. Groups are re-distributed such that all values in the same group go to the same reducer.

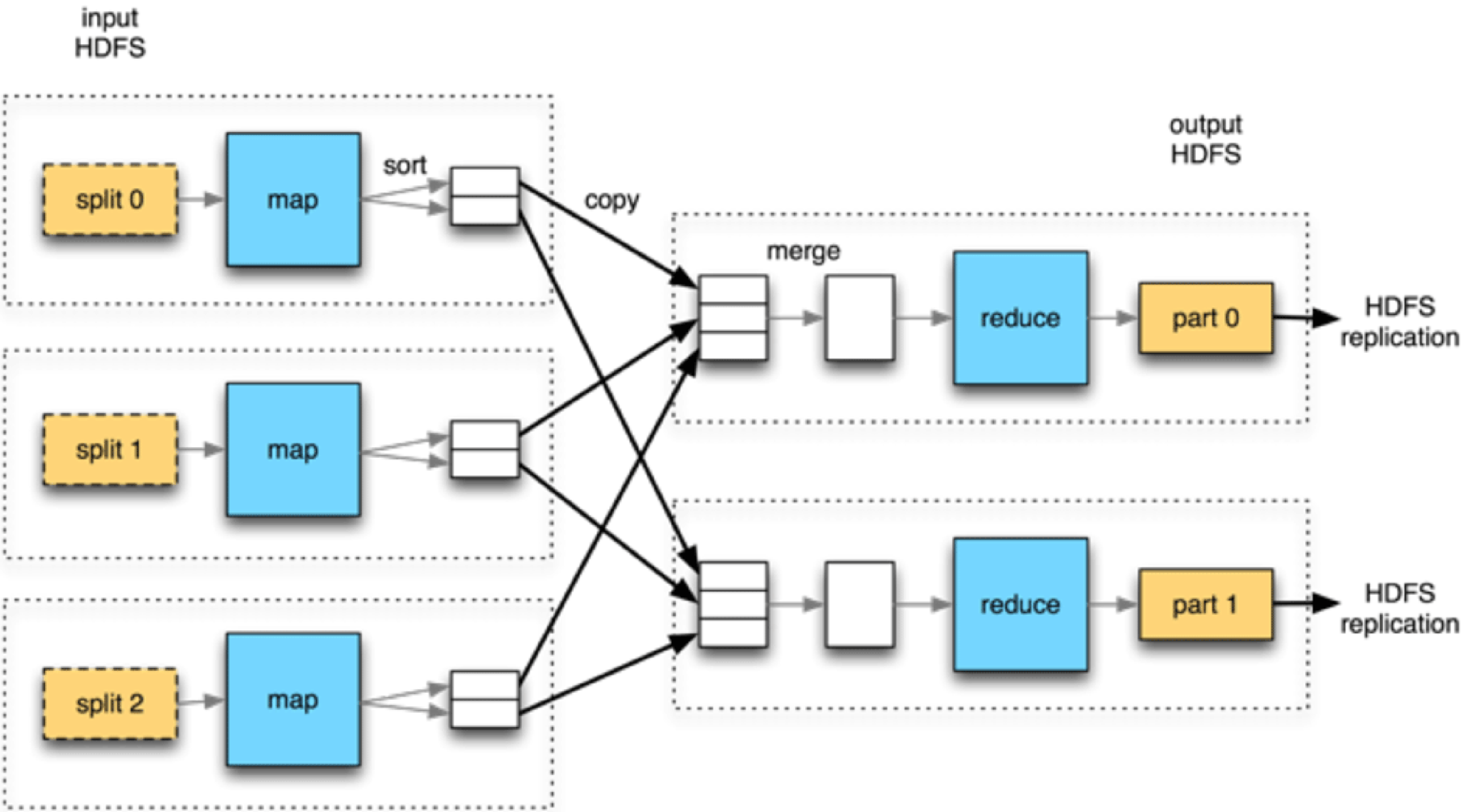
Reduce Step:

Each reducer independently processes its group.

`Reduce(k2, list(v2))` $\rightarrow$ `list(v3)`

MapReduce

MapReduce on YARN



MapReduce

MapReduce on YARN

Recap: Components of the YARN Framework

- **ResourceManager** Single instance per cluster, controls container allocation
- **NodeManager** Runs on each cluster node, provides containers to applications

Components of a YARN MapReduce Job

- **ApplicationMaster** Controls execution on the cluster (one for each YARN application)
- **Mapper** Processes input data
- **Reducer** Processes (sorted) Mapper output

Each of the above runs in a YARN Container

MapReduce

MapReduce on YARN

Basic process:

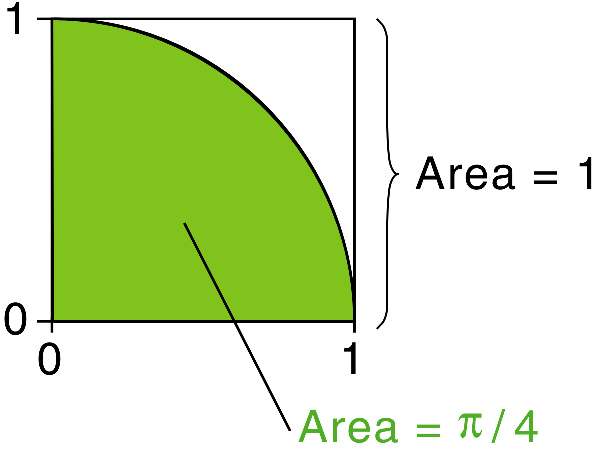
1. Client application requests a container for the ApplicationMaster
2. ApplicationMaster runs on the cluster, requests further containers for Mappers and Reducers
3. Mappers execute user-provided `map()` function on their part of the input data
4. The `shuffle()` phase is started to distribute map output to reducers
5. Reducers execute user-provided `reduce()` function on their group of map output
6. Final result is stored in HDFS

See also: [Anatomy of a MapReduce Job]

MapReduce Examples

Quasi-Monte-Carlo Estimation of π

Idea:

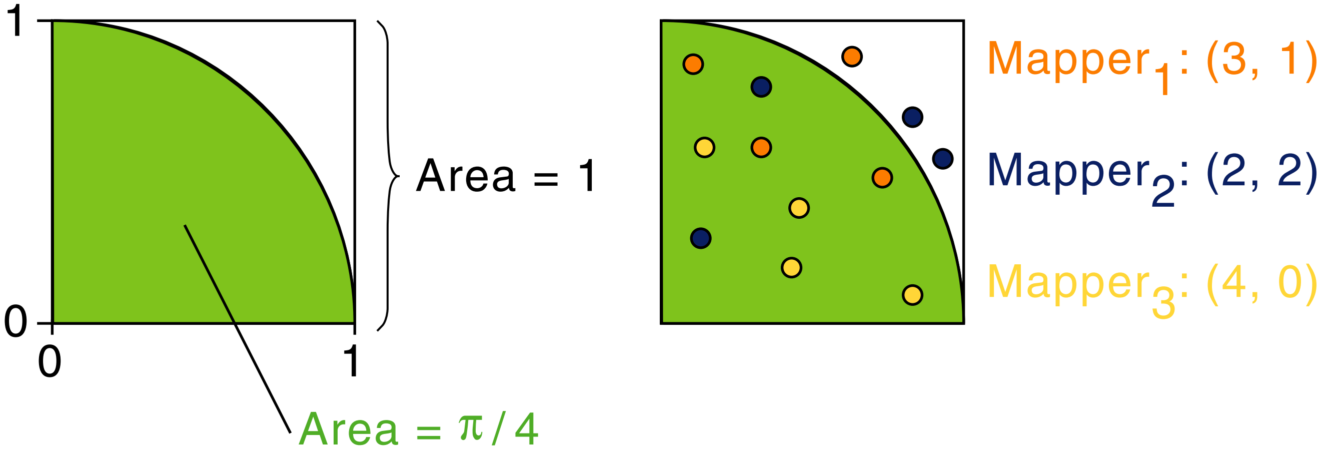


- The area of a circle segment inside the unit square is $\frac{\pi}{4}$

MapReduce Examples

Quasi-Monte-Carlo Estimation of π

Idea:

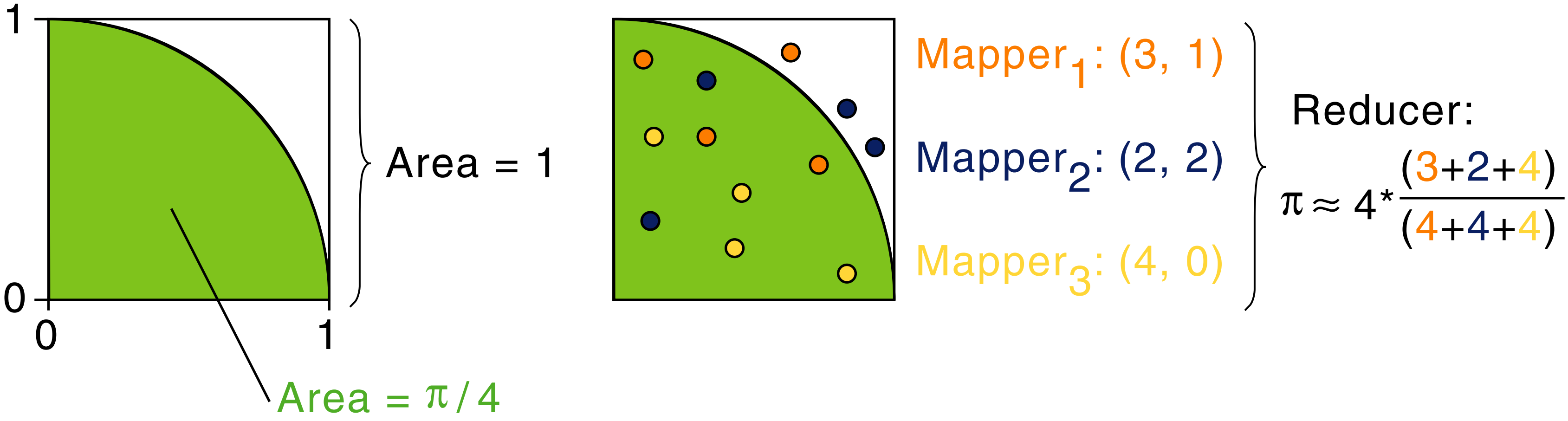


- The area of a circle segment inside the unit square is $\frac{\pi}{4}$
- Each mapper generates some random points inside the square, and counts how many fall inside/outside the circle segment.

MapReduce Examples

Quasi-Monte-Carlo Estimation of π

Idea:



- The area of a circle segment inside the unit square is $\frac{\pi}{4}$
- Each mapper generates some random points inside the square, and counts how many fall inside/outside the circle segment.
- The reducer sums up points inside and points total, to compute our estimate of π.

MapReduce Examples

Monte-Carlo Estimation of π

This is already included as an example program in Hadoop.

Connect to a node and run:

```
cd /opt/hadoop-*/share/hadoop/mapreduce
```

then:

```
hadoop jar hadoop-mapreduce-examples-*.jar pi 4 100000
```

The output should look like this:

```
Number of Maps = 4
Samples per Map = 1000000
Wrote input for Map #0
...
Job Finished in 13.74 seconds
Estimated value of Pi is 3.14160400000000000000
```

MapReduce Examples

Parallellizing Shell Scripts with Hadop Streaming

Let’s say we want to know which of the words “you” and “thou” occurs more frequently in Shakespeare’s works.

Last time, we answered this using a simple shell pipeline on a single node.

MapReduce Examples

Quick Recap From Last Time

`cat FILE` — outputs contents of FILE

`A | B` — the output of command A becomes input of command B

`grep PATTERN` — outputs all input lines containing PATTERN

MapReduce Examples

Quick Recap From Last Time

`cat FILE` — outputs contents of FILE

`A | B` — the output of command A becomes input of command B

`grep PATTERN` — outputs all input lines containing PATTERN

```
cat shakespeare.txt | grep ' you '
```

MapReduce Examples

Quick Recap From Last Time

`cat FILE` — outputs contents of FILE

`A | B` — the output of command A becomes input of command B

`grep PATTERN` — outputs all input lines containing PATTERN

```
cat shakespeare.txt | grep ' you '
```

`grep -o PATTERN` — outputs only the matching part of each input line.

`\|` — inside the PATTERN, marks an alternative (“or”)

MapReduce Examples

Quick Recap From Last Time

`cat FILE` — outputs contents of FILE

`A | B` — the output of command A becomes input of command B

`grep PATTERN` — outputs all input lines containing PATTERN

```
cat shakespeare.txt | grep ' you '
```

`grep -o PATTERN` — outputs only the matching part of each input line.

`\|` — inside the PATTERN, marks an alternative ("or")

```
cat shakespeare.txt | grep -o ' you \| thou '
```

MapReduce Examples

Quick Recap From Last Time

`cat FILE` — outputs contents of FILE

`A | B` — the output of command A becomes input of command B

`grep PATTERN` — outputs all input lines containing PATTERN

```
cat shakespeare.txt | grep ' you '
```

`grep -o PATTERN` — outputs only the matching part of each input line.

`\|` — inside the PATTERN, marks an alternative ("or")

```
cat shakespeare.txt | grep -o ' you \| thou '
```

`sort` — sorts the input alphabetically

`uniq -c` — counts consecutive identical lines in the input

MapReduce Examples

Quick Recap From Last Time

`cat FILE` — outputs contents of FILE

`A | B` — the output of command A becomes input of command B

`grep PATTERN` — outputs all input lines containing PATTERN

```
cat shakespeare.txt | grep ' you '
```

`grep -o PATTERN` — outputs only the matching part of each input line.

`\|` — inside the PATTERN, marks an alternative (“or”)

```
cat shakespeare.txt | grep -o ' you \| thou '
```

`sort` — sorts the input alphabetically

`uniq -c` — counts consecutive identical lines in the input

So, finally: [full explanation]

```
cat shakespeare.txt | grep -o ' you \| thou ' | sort | uniq -c
```

MapReduce Examples

Parallellizing Shell Scripts with Hadop Streaming

We have our answer, but we only used one machine. Hadoop Streaming lets us easily parallelize such shell scripts over the entire cluster.

MapReduce Examples

Parallellizing Shell Scripts with Hadop Streaming

We have our answer, but we only used one machine. Hadoop Streaming lets us easily parallelize such shell scripts over the entire cluster.

1. Put the input file in HDFS:

```
hadoop fs -put shakespeare.txt shakespeare-hdfs.txt
```

2. Go to the directory with the Hadoop Streaming Jar file:

```
cd /opt/hadoop-*/share/hadoop/tools/lib
```

3. Run our shellscript as a Streaming job:

```
hadoop jar hadoop-streaming-*.jar \
    -input shakespeare-hdfs.txt \
    -output my-word-counts \
    -mapper "grep -o ' you \| thou '" \
    -reducer "uniq -c"
```

Notes: \ means “continue the previous line”; Hadoop already does the sorting for us.

MapReduce Examples

Parallellizing Shell Scripts with Hadop Streaming

Let's look at the results:

```
hadoop fs -ls my-word-counts
```

```
Found 2 items
-rw-r-r- 3 vagrant supergroup  0 2018-04-21 13:41 my-word-counts/_SUCCESS
-rw-r-r- 3 vagrant supergroup 31 2018-04-21 13:41 my-word-counts/part-00000
```

```
hadoop fs -cat my-word-counts/part-00000
```

```
4159 thou
8684 you
```