Hadoop Architecture

Prof. Hyuk-Yoon Kwon
https://sites.google.com/view/seoultech-bigdata

Hadoop Architecture and HDFS

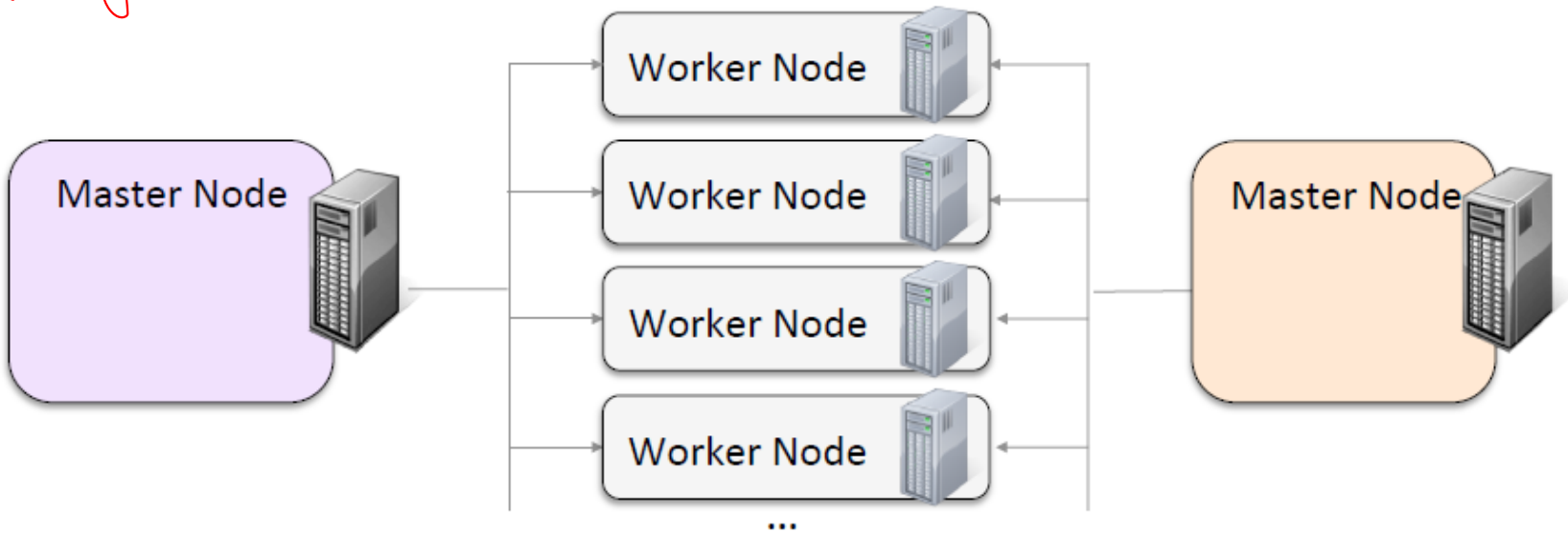
In this chapter you will learn

- How Hadoop Distributed File System stores data across a cluster
- How to use HDFS using the Hue File Browser or the `hdfs` command
- How Hadoop YARN provides cluster resource management for distributed data processing nodes, data, jobs
- How to use Hue, the YARN Web UI or the yarn command to monitor your cluster

Hadoop Cluster Terminology

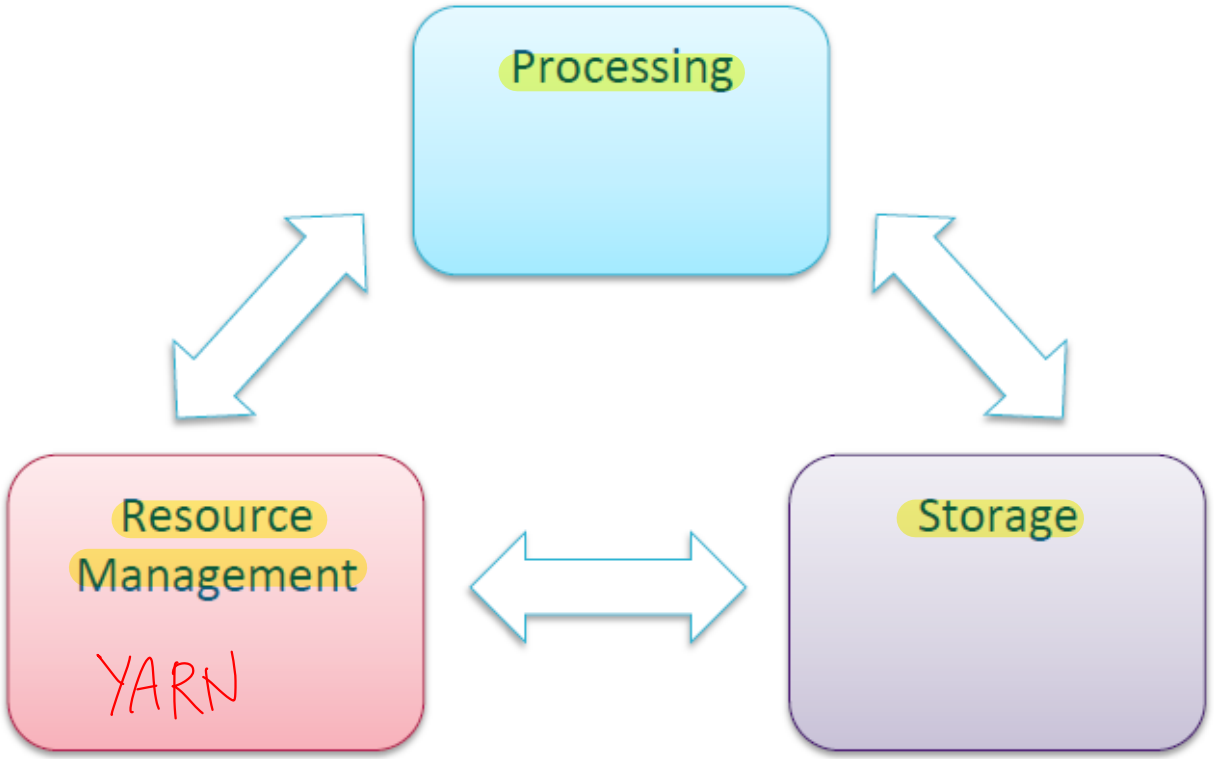
- A ***cluster*** is a **group of computers working together**
  - Provides data storage, data processing, and resource management
- A ***node*** is an **individual computer in the cluster**
  - *Master* nodes manage distribution of work and data to *worker* nodes
- A ***daemon*** is a **program running on a node**
  - Each Hadoop daemon performs a specific function in the cluster

always running



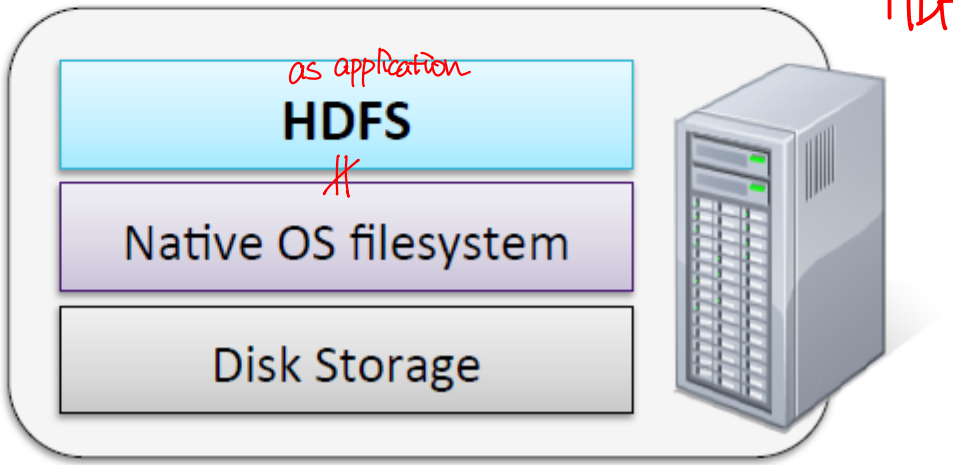
Cluster Components

- Three main components of a cluster
- Work together to provide distributed data processing
- We will start with the Storage component
  - HDFS



HDFS Basic Concepts (1) in user space (not kernel)

- **HDFS is a filesystem written in Java**
  - Based on Google's GFS
- **Sits on top of a native filesystem**
  - Such as ext3, ext4, or xfs
- **Provides redundant storage for massive amounts of data**
  - Using readily-available, industry-standard computers



HDFS ⇒ { logical : file system
physical : not actual file system

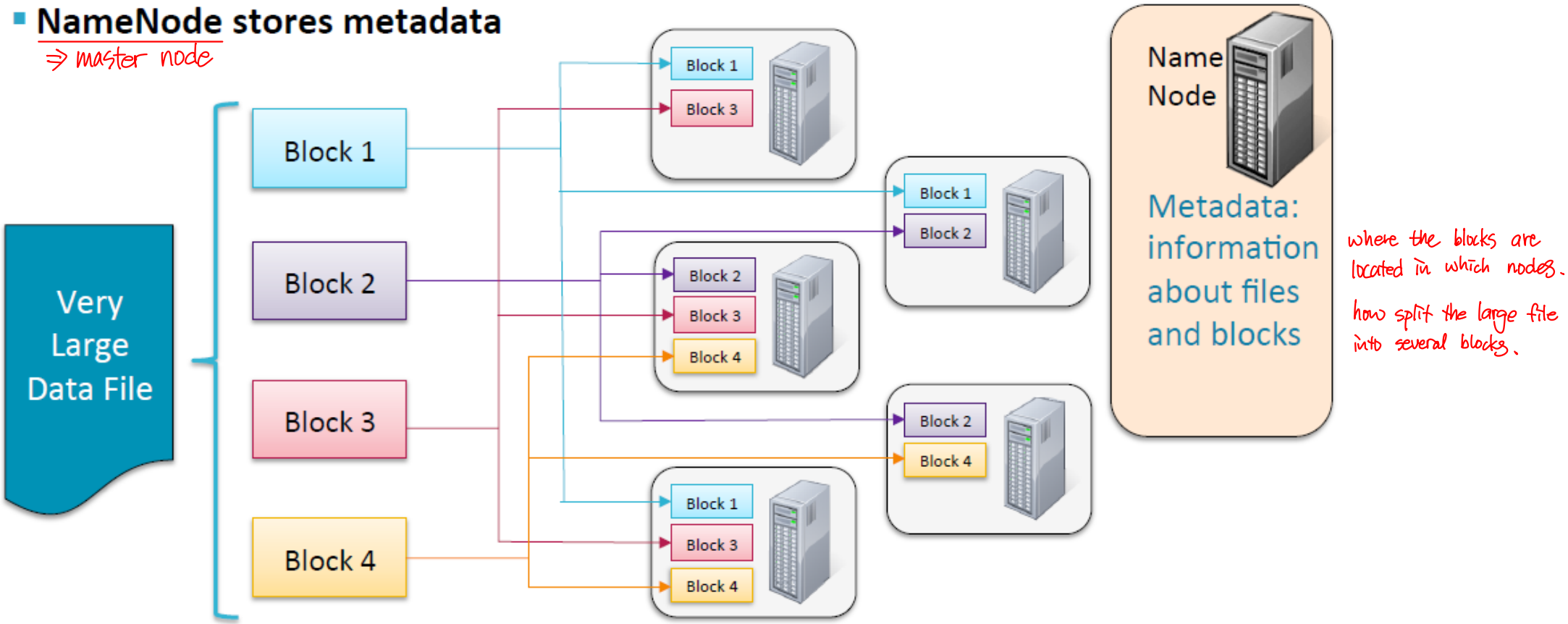
HDFS Basic Concepts (2)

- **HDFS performs best with a 'modest' number of large files**
  - Millions, rather than billions, of files
  - Each file typically 100MB or more
- **Files in HDFS are 'write once'** modifying file = write new file
  - No random writes to files are allowed
- **HDFS is optimized for large, streaming reads of files** merge smaller files to large files for performance
  - Rather than random reads

How Files Are Stored

minimizing the overhead to find the starting point of the file

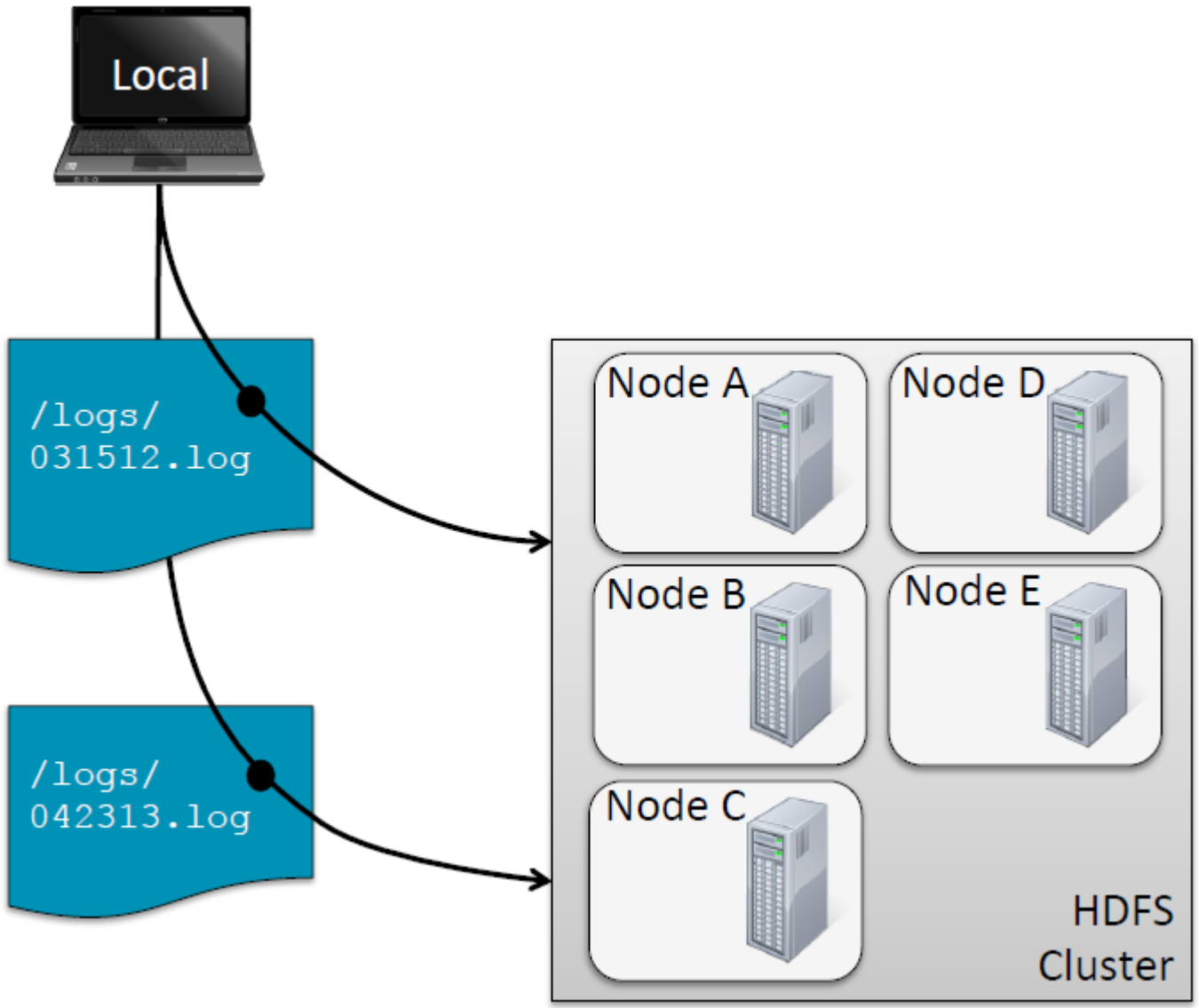
- Data files are split into 128MB blocks which are distributed at load time
  - huge size
  - default
- Each block is replicated on multiple data nodes (default 3x) ⇒ { increase reliability of service; provide efficient service to user } maximize availability
- NameNode stores metadata
  - ⇒ master node



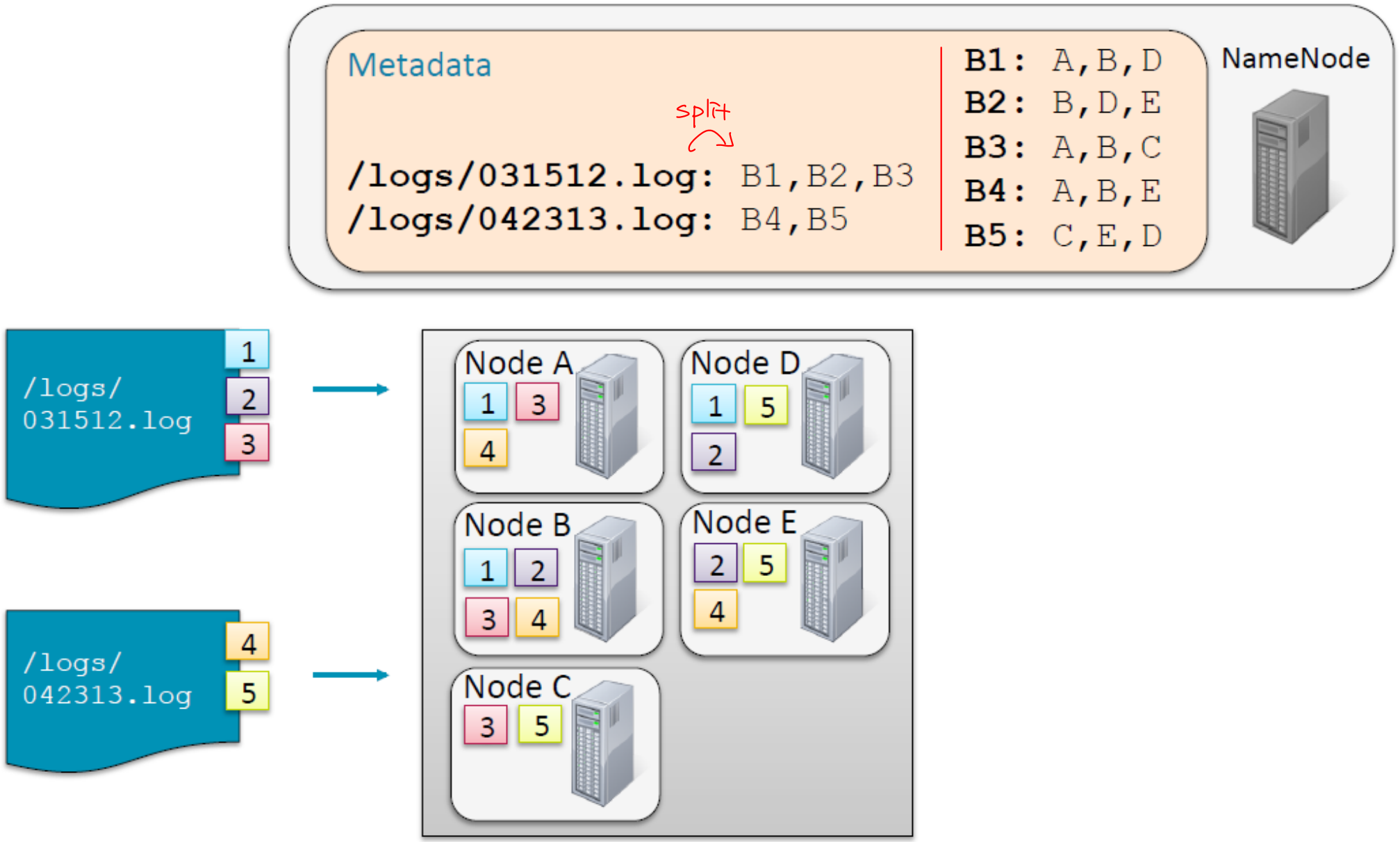
where the blocks are located in which nodes.

how split the large file into several blocks.

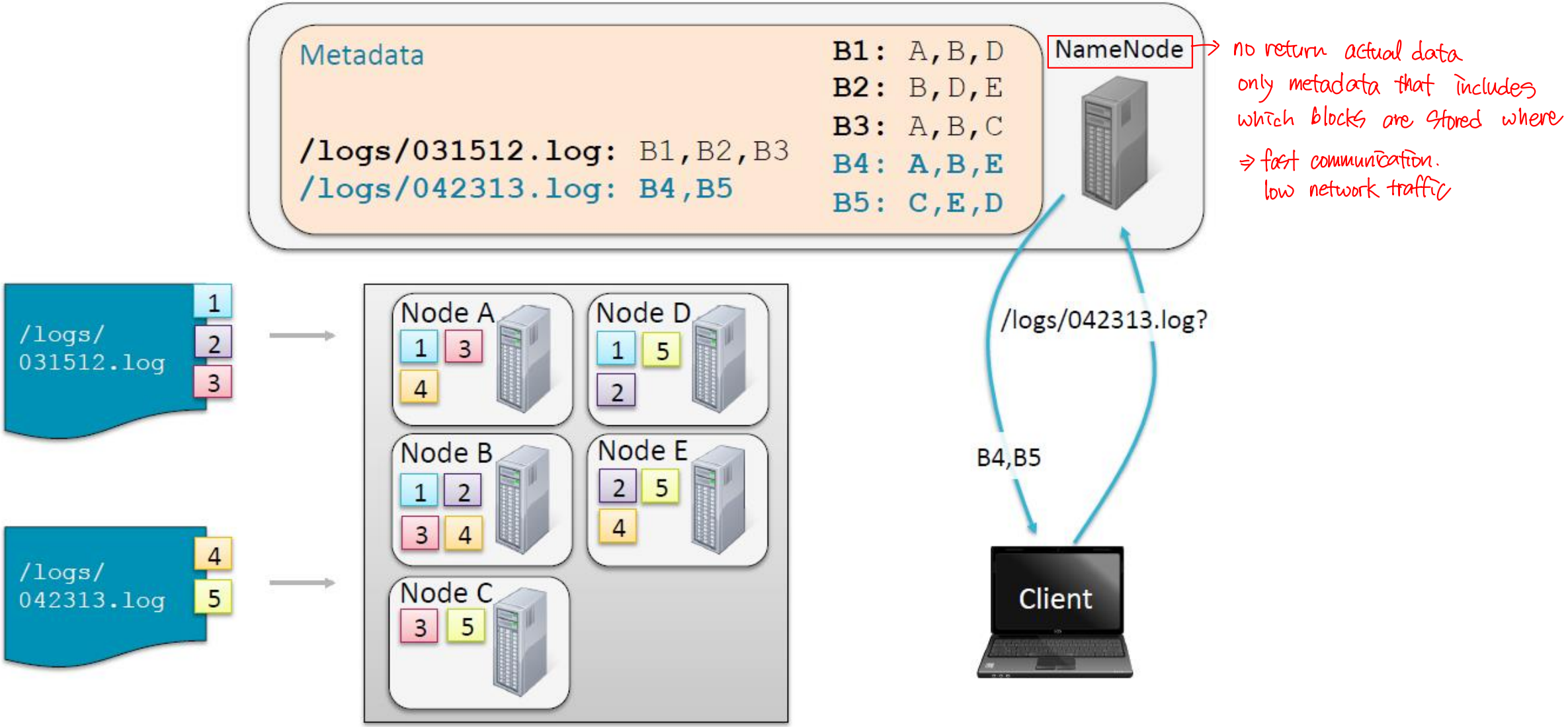
Example: Storing and Retrieving Files (1)



Example: Storing and Retrieving Files (2)

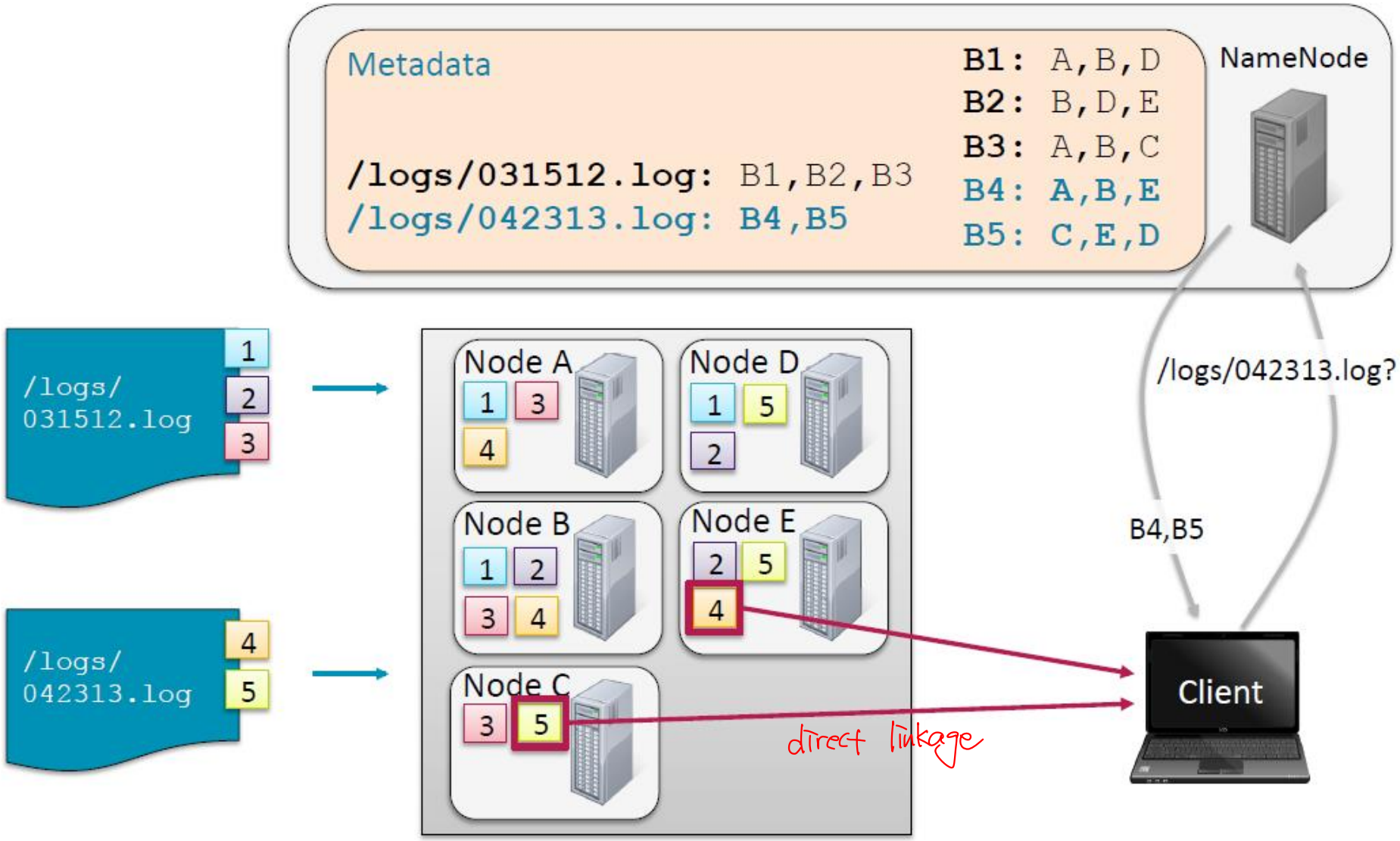


Example: Storing and Retrieving Files (3)



no return actual data
only metadata that includes which blocks are stored where
⇒ fast communication.
low network traffic

Example: Storing and Retrieving Files (4)



HDFS NameNode Availability

- **The NameNode daemon must be running at all times**
  - If the NameNode stops, the cluster becomes inaccessible

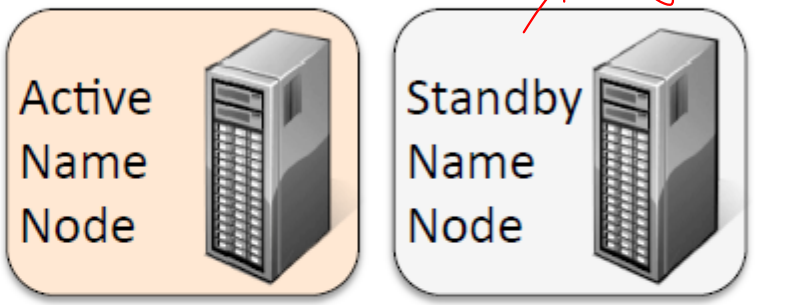
Name Node ⇒ managing all metadata for all datasets

- **HDFS is typically set up for High Availability**
  - Two NameNodes: Active and Standby

cost ↑

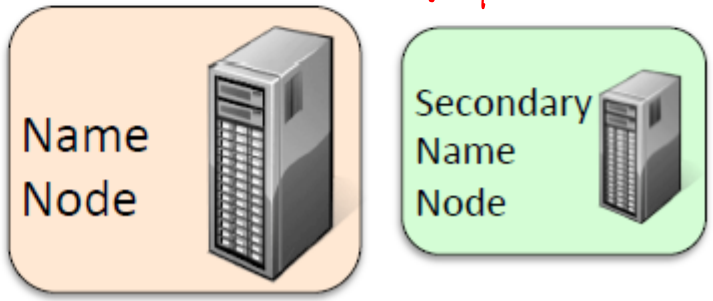
hot swap

ready state



- **Small clusters may use 'Classic mode'**
  - One NameNode
  - One "helper" node called the Secondary NameNode
    - Bookkeeping, not backup

direct replacement



Options for Accessing HDFS

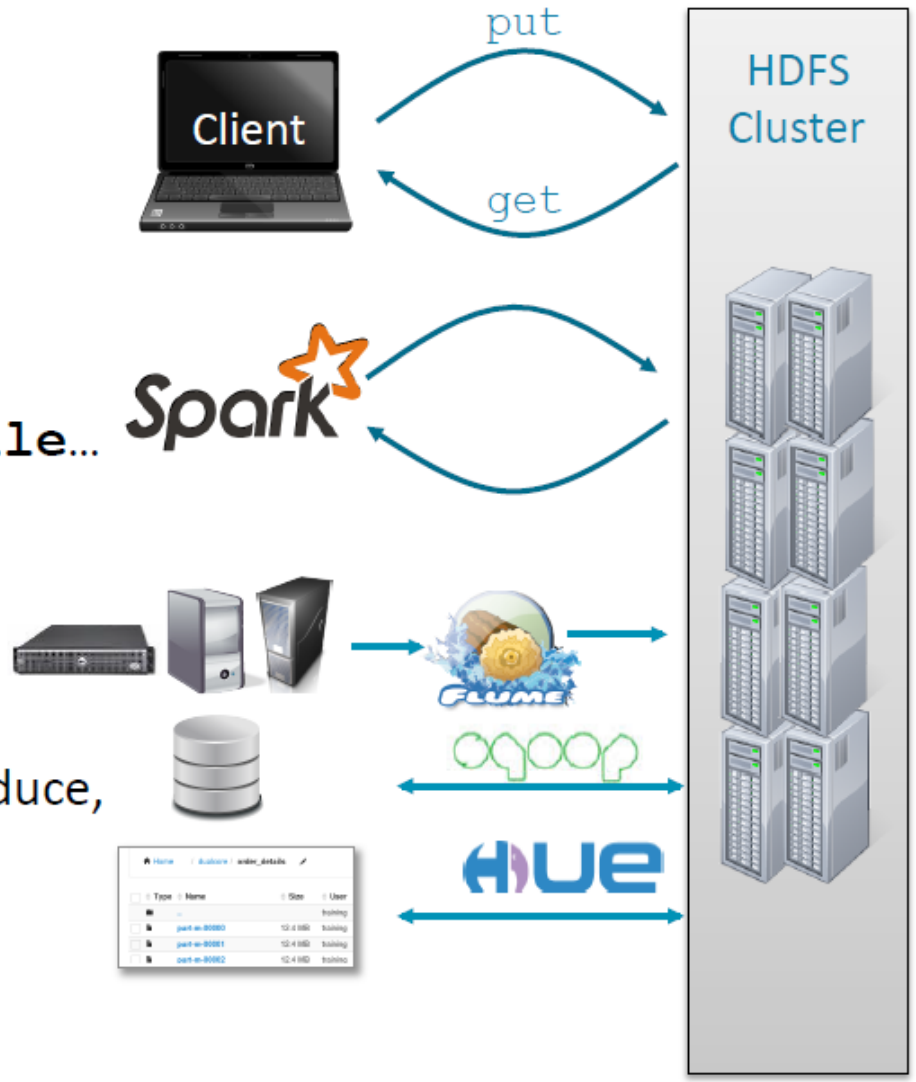
- **From the command line**
  - FsShell:

```
$ hdfs dfs
```

- **In Spark**
  - By URI, e.g.

```
hdfs://nnhost:port/file…
```

- **Other programs**
  - Java API
    - Used by Hadoop MapReduce, Impala, Hue, Sqoop, Flume, etc.
  - RESTful interface



HDFS Command Line Examples (1)

- Copy file `foo.txt` from local disk to the user's directory in HDFS

local hdfs

```
$ hdfs dfs -put foo.txt foo.txt
```

  - This will copy the file to `/user/username/foo.txt`

- Get a directory listing of the user's home directory in HDFS

```
$ hdfs dfs -ls
```

- Get a directory listing of the HDFS root directory

```
$ hdfs dfs -ls /
```

HDFS Command Line Examples (2)

- Display the contents of the HDFS file `/user/fred/bar.txt`

```
$ hdfs dfs -cat /user/fred/bar.txt
```

- Copy that file to the local disk, named as `baz.txt`

```
$ hdfs dfs -get /user/fred/bar.txt baz.txt
```

- Create a directory called `input` under the user's home directory

```
$ hdfs dfs -mkdir input
```

Note: `copyFromLocal` is a synonym for `put`; `copyToLocal` is a synonym for `get`

HDFS Command Line Examples (3)

- Delete the directory `input_old` and all its contents

```
$ hdfs dfs -rm -r input_old
```

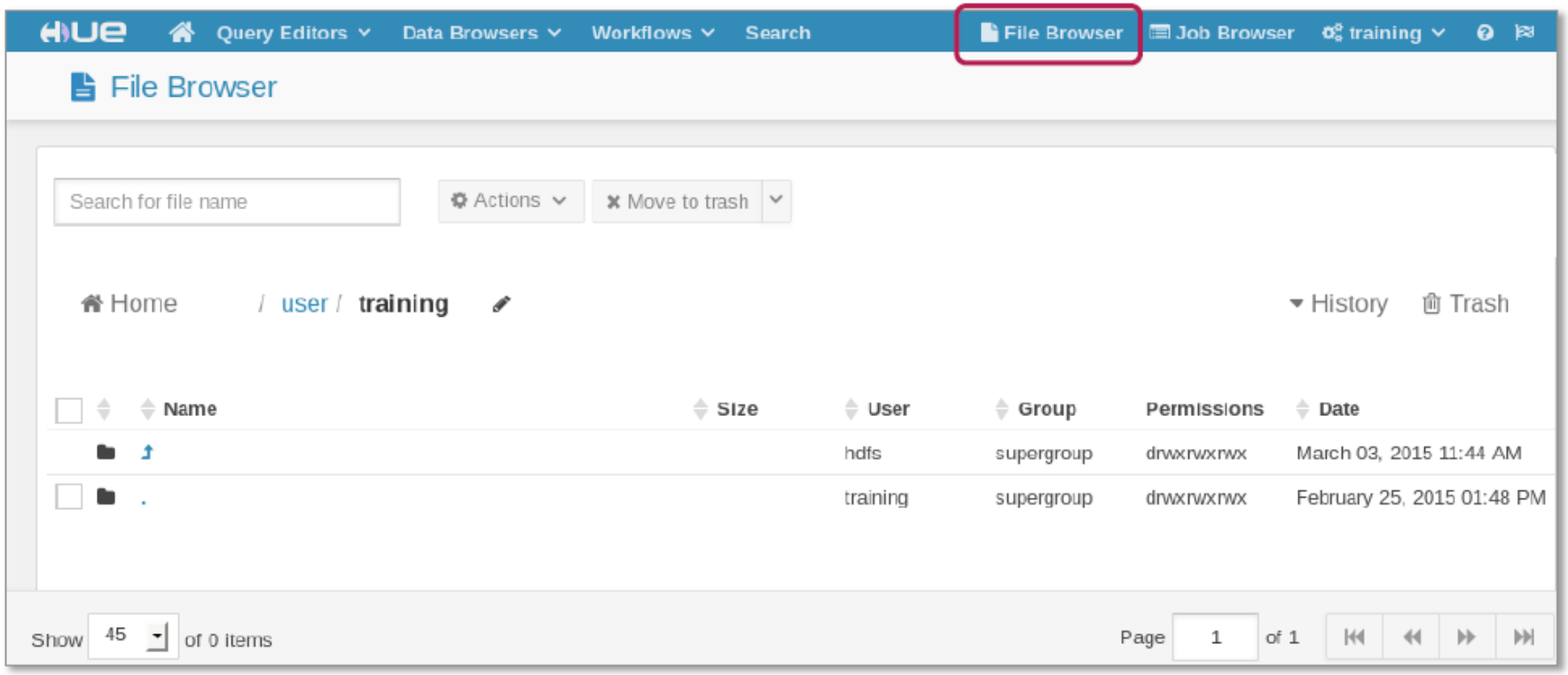
recursively

Practice1: Access HDFS with Command Line

- **In this homework lab you will**
  - Create a `/loudacre` base directory for course homework
  - Practice uploading and viewing data files
- **Please refer to the Homework description**

The Hue HDFS File Browser

- **The File Browser in Hue lets you view and manage your HDFS directories and files**
  - Create, move, rename, modify, upload, download and delete directories and files
  - View file contents



HDFS Recommendations

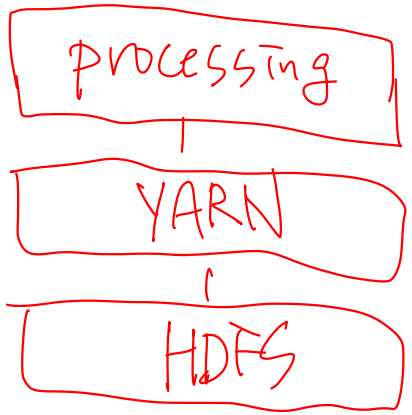
- **HDFS is a repository for all your data**
  - Structure and organize carefully!
- **Best practices include**
  - Define a standard directory structure
  - Include separate locations for staging data
- **Example organization**
  - **`/user/`**… – data and configuration belonging only to a single user
  - **`/etl`** – Work in progress in Extract/Transform/Load stage
  - **`/tmp`** – Temporary generated data shared between users
  - **`/data`** – Data sets that are processed and available across the organization for analysis
  - **`/app`** – Non-data files such as configuration, JAR files, SQL files, etc.

Practice2: Access HDFS with Hue

- **In this homework lab you will**
  - Create a `/loudacre` base directory for course homework
  - Practice uploading and viewing data files
- **Please refer to the Homework description**

What is YARN?

- YARN = Yet Another Resource Negotiator
- YARN is the Hadoop processing layer that contains
  - A resource manager
  - A job scheduler
- YARN allows multiple data processing engines to run on a single Hadoop cluster
  - Batch programs (e.g. Spark, MapReduce)
  - Interactive SQL (e.g. Impala)
  - Advanced analytics (e.g. Spark, Impala)
  - Streaming (e.g. Spark Streaming)

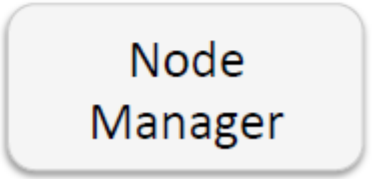


YARN Daemons

responsibility to maintain all resources. in distributed env.

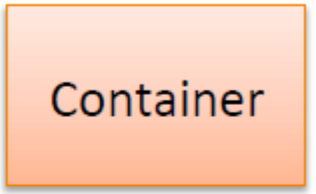
- **Resource Manager (RM)**
  - Runs on master node
  - Global resource scheduler
  - Arbitrates (경재.갈등) system resources between competing applications ⇒ optimize overall cluster utilization considering node status
  - Has a pluggable scheduler to support different algorithms (capacity, fair scheduler, etc.)

- **Node Manager (NM)**
  - Runs on slave nodes
  - Communicates with RM

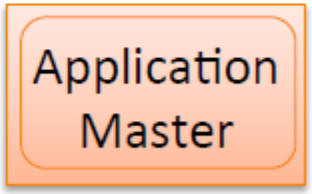


Running an Application in YARN

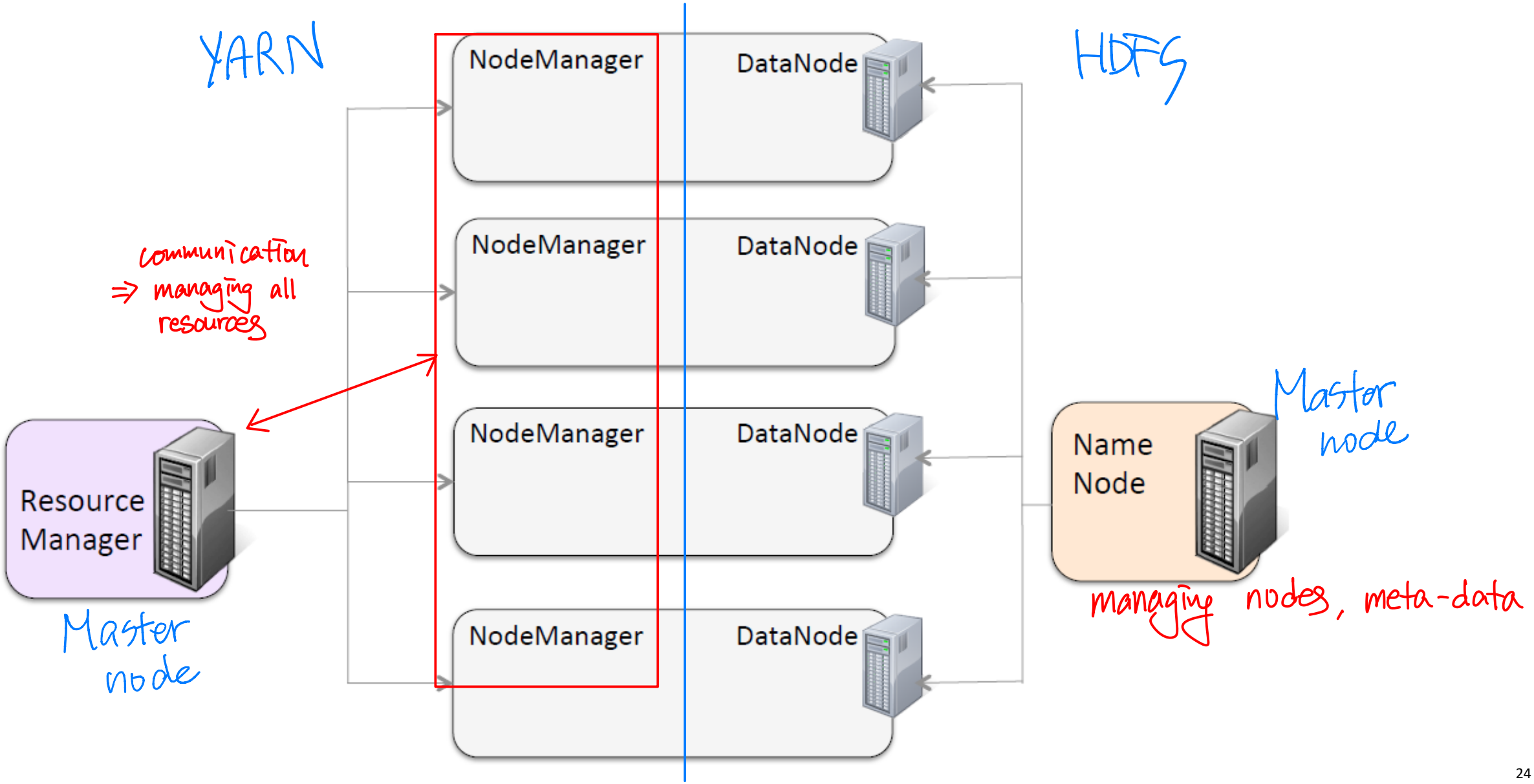
- **Containers** execute App in Hadoop ⇒ creating container (RM)
  - Created by the RM upon request
  - Allocate a certain amount of resources (memory, CPU) on a slave node
  - Applications run in one or more containers



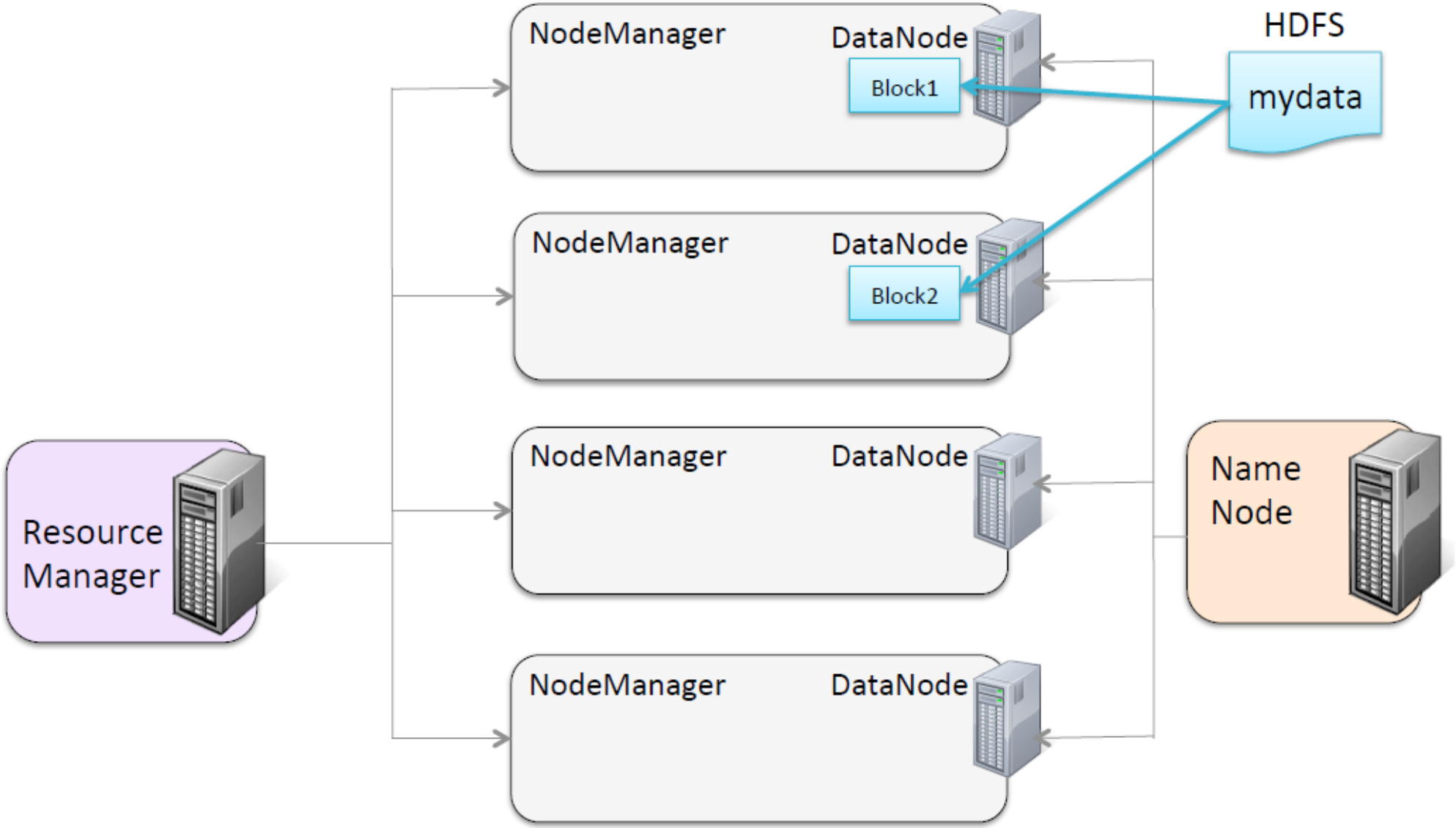
- **Application Master (AM)** managing overall app process (start ~ end)
  - One per application (for each)
  - Framework/application specific
  - Runs in a container
  - Requests more containers to run application tasks



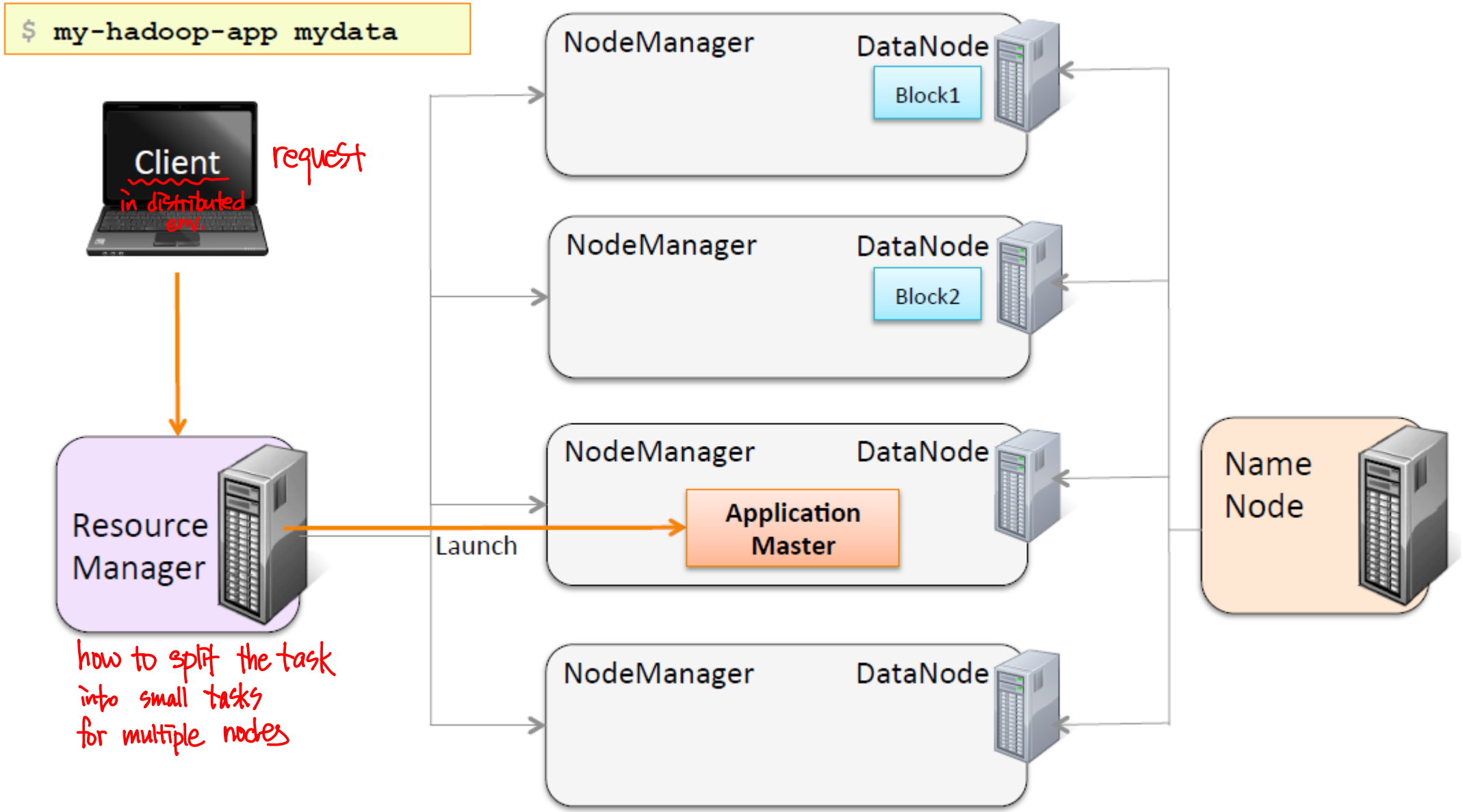
Running an Application on YARN (1)



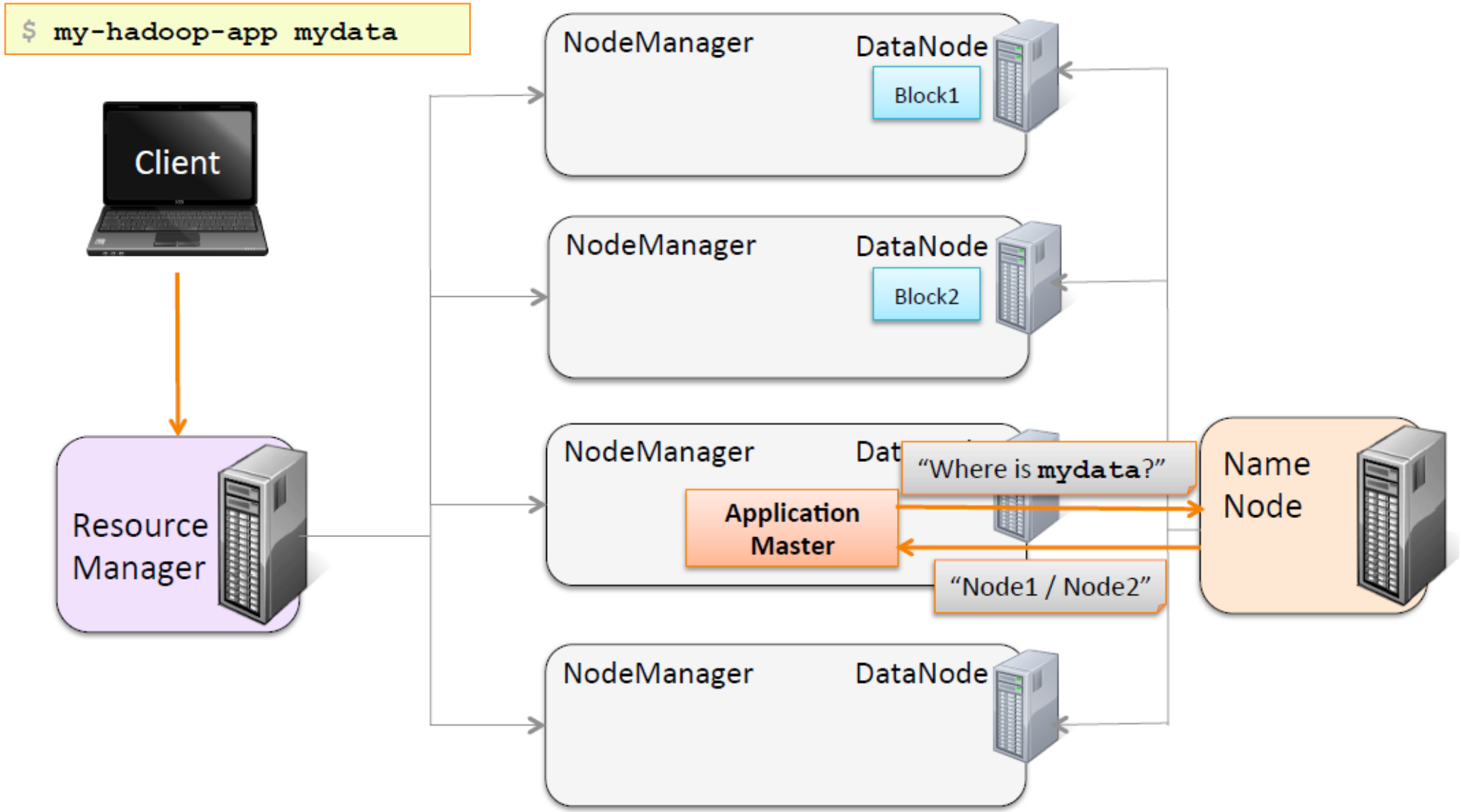
Running an Application on YARN (2)



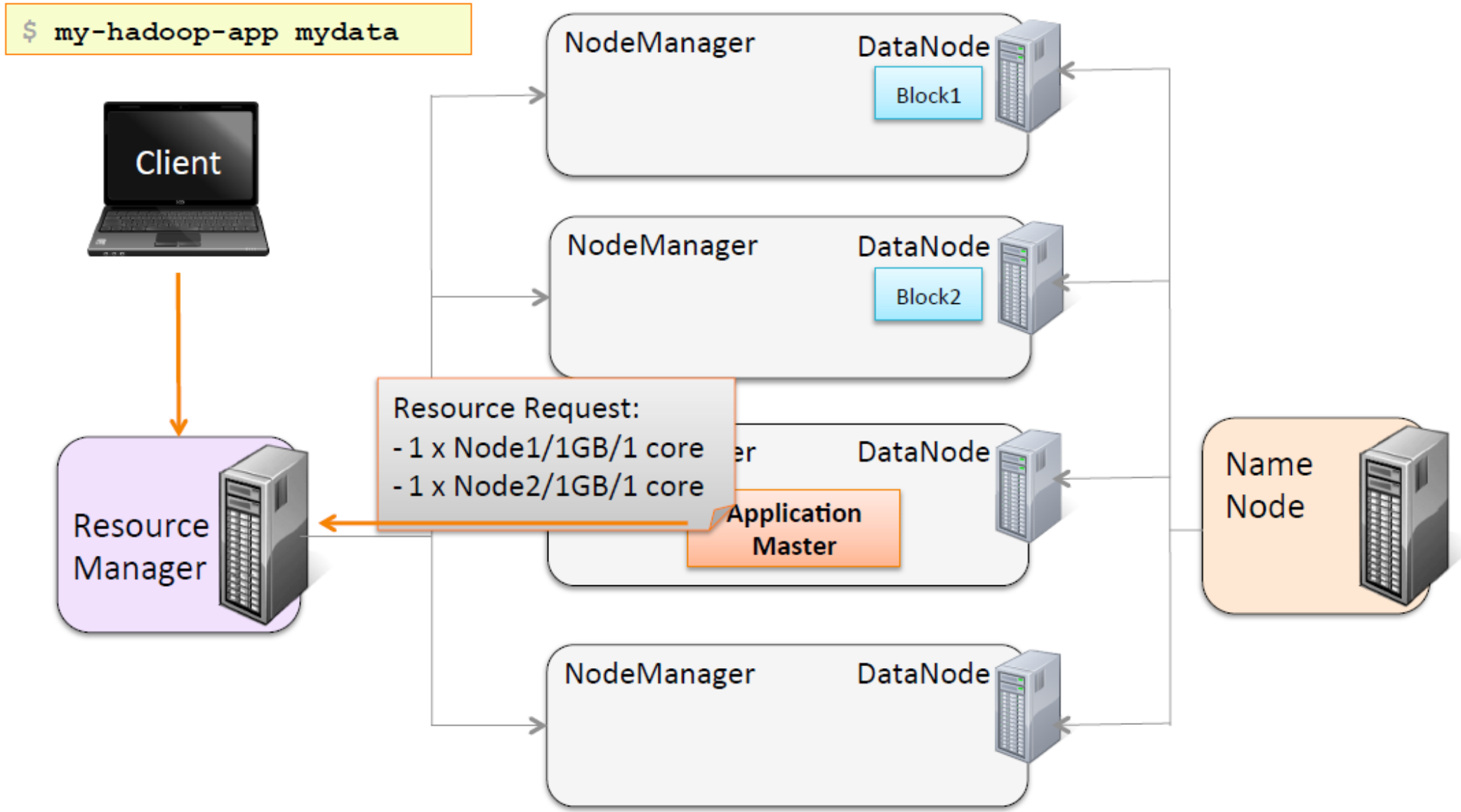
Running an Application on YARN (3)



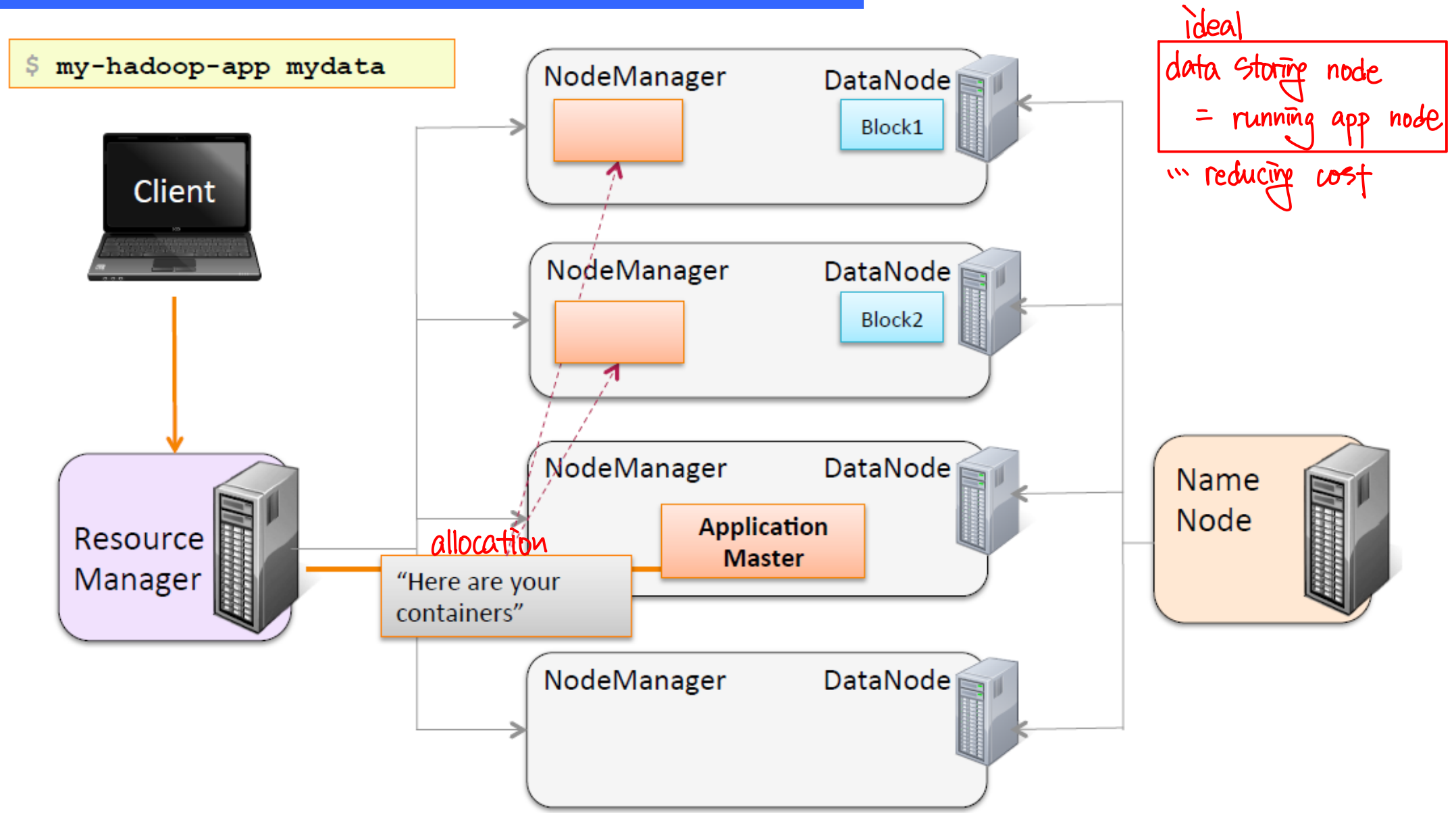
Running an Application on YARN (4)



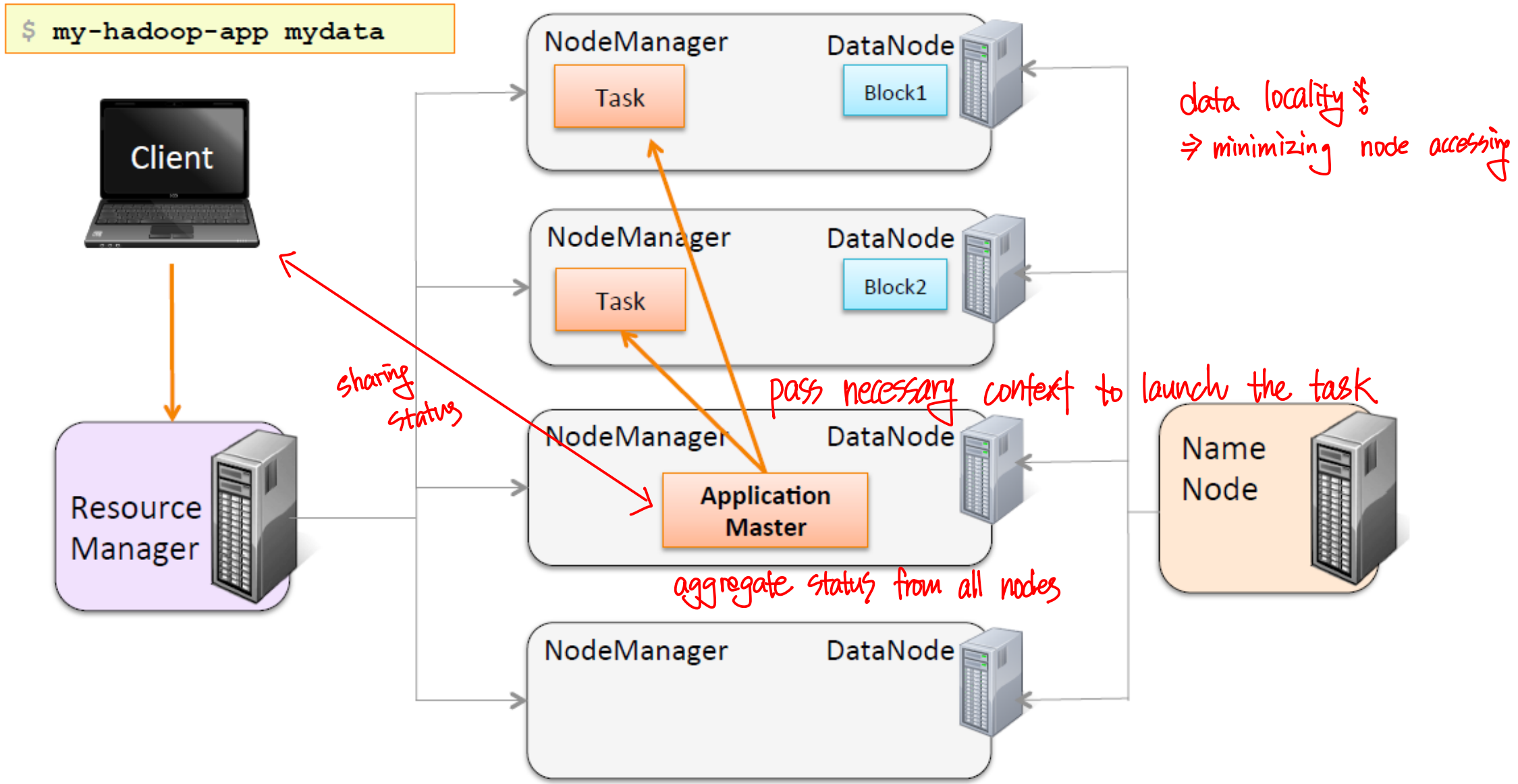
Running an Application on YARN (5)



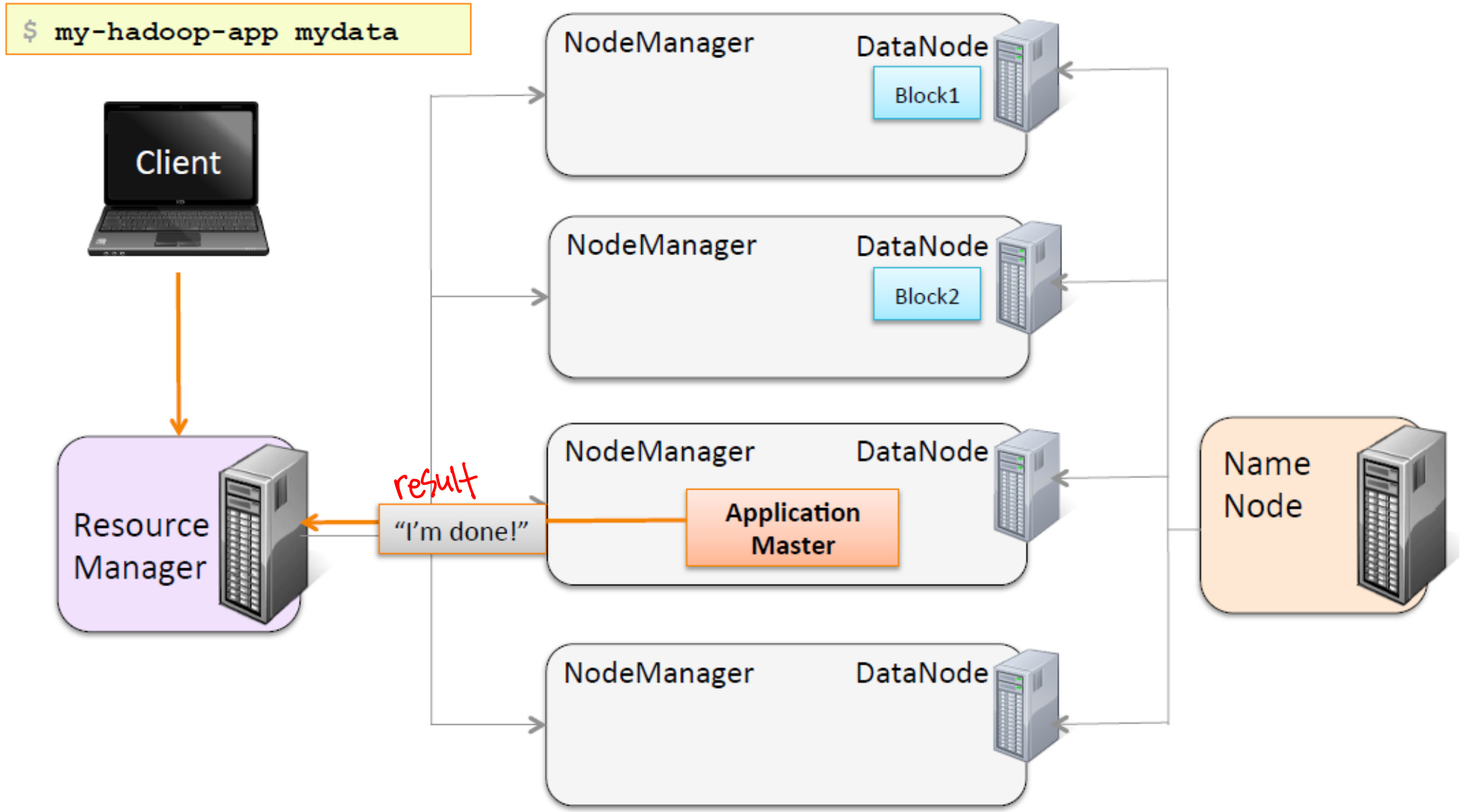
Running an Application on YARN (6)



Running an Application on YARN (7)



Running an Application on YARN (8)



Working with YARN

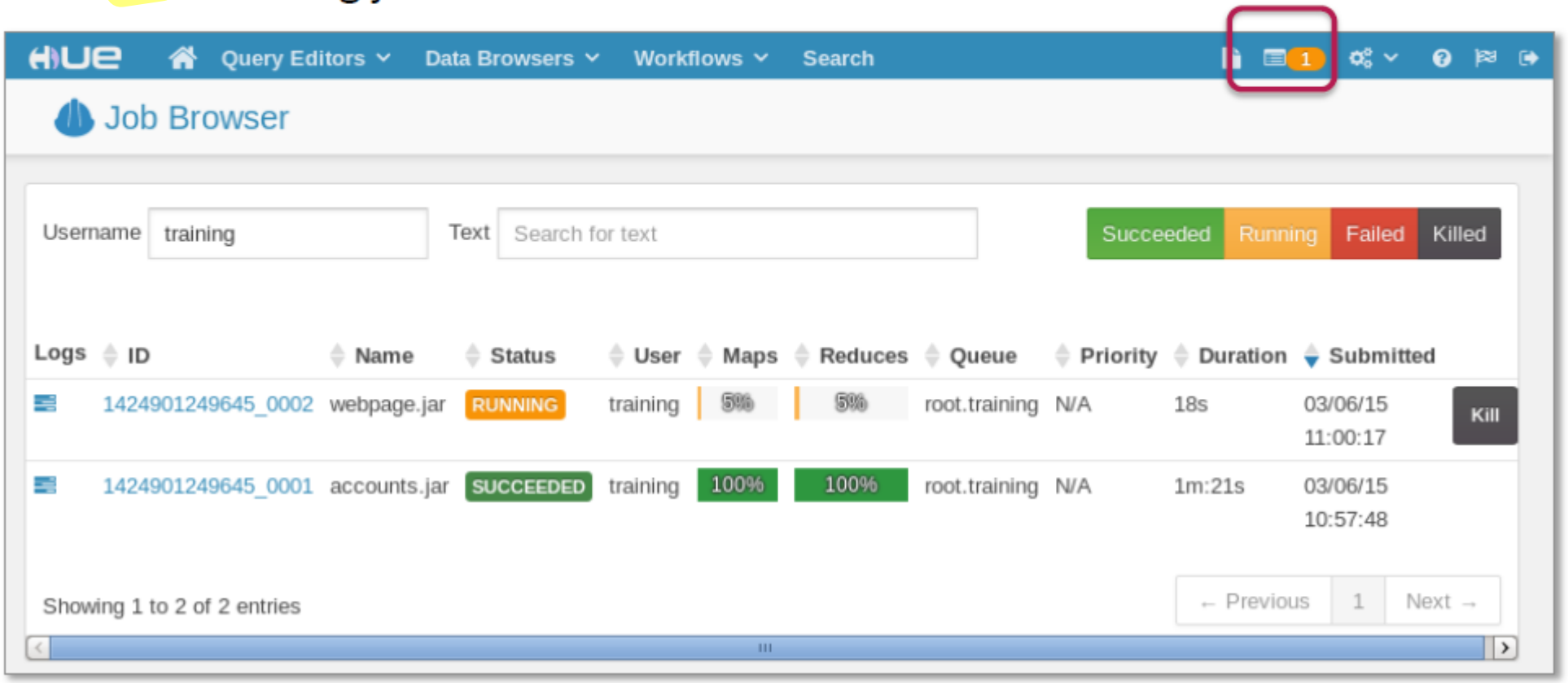
- **Developers need to be able to**
  - Submit jobs (applications) to run on the YARN cluster
  - Monitor and manage jobs
- **Hadoop includes three major YARN tools for developers**
  - The Hue Job Browser
  - The YARN Web UI
  - The YARN command line

Master node : Resource Manager

Worker node : Node Manager } Containers (Application Master)

The Hue Job Browser

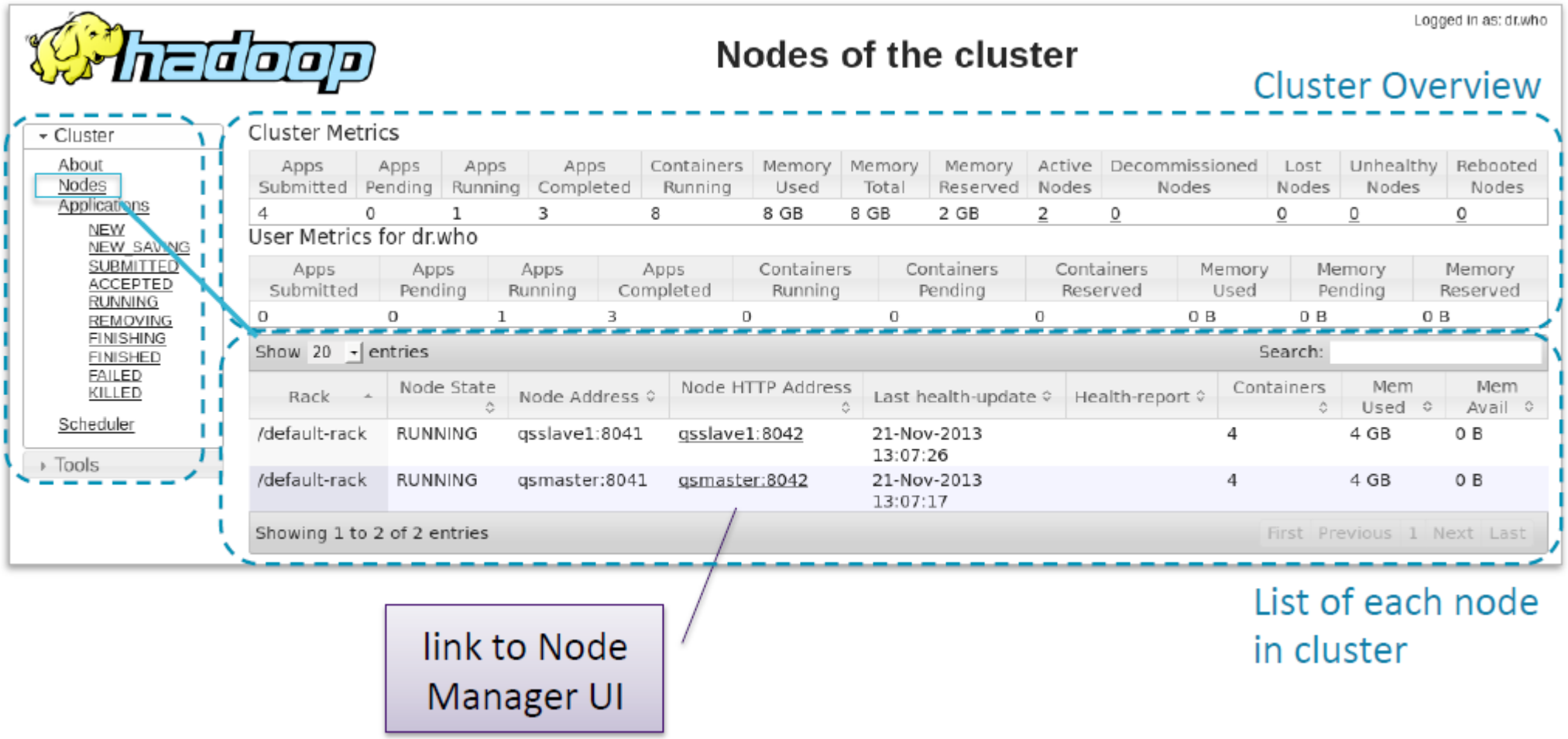
- **The Hue Job Browser allows you to**
  - Monitor the status of a job
  - View the logs
  - Kill a running job



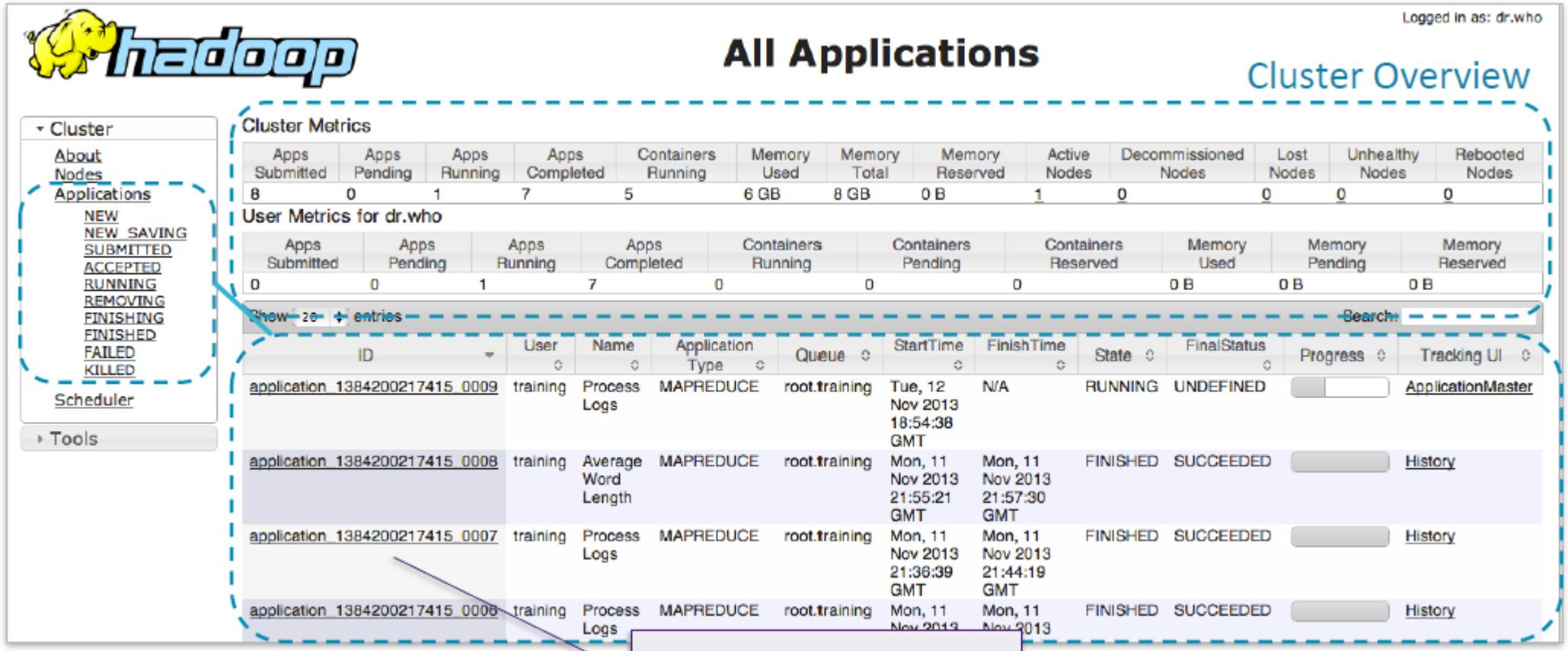
The YARN Web UI

- **Resource Manager UI is the main entry point**
  - Runs on the RM host on port 8080 by default
- **Provides more detailed view than Hue**
- **Does not provide any control or configuration**

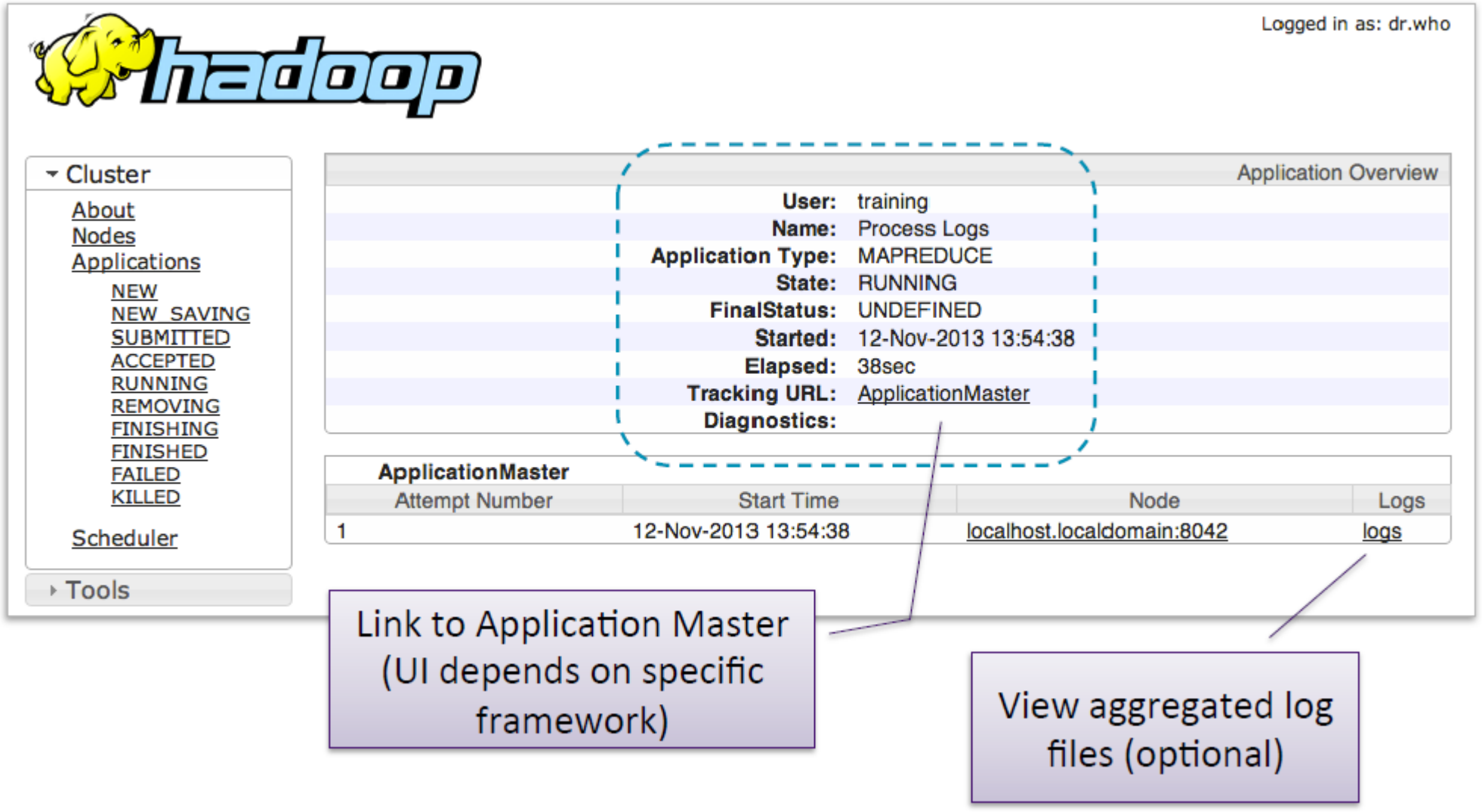
Resource Manager UI: Nodes



Resource Manager UI: Applications

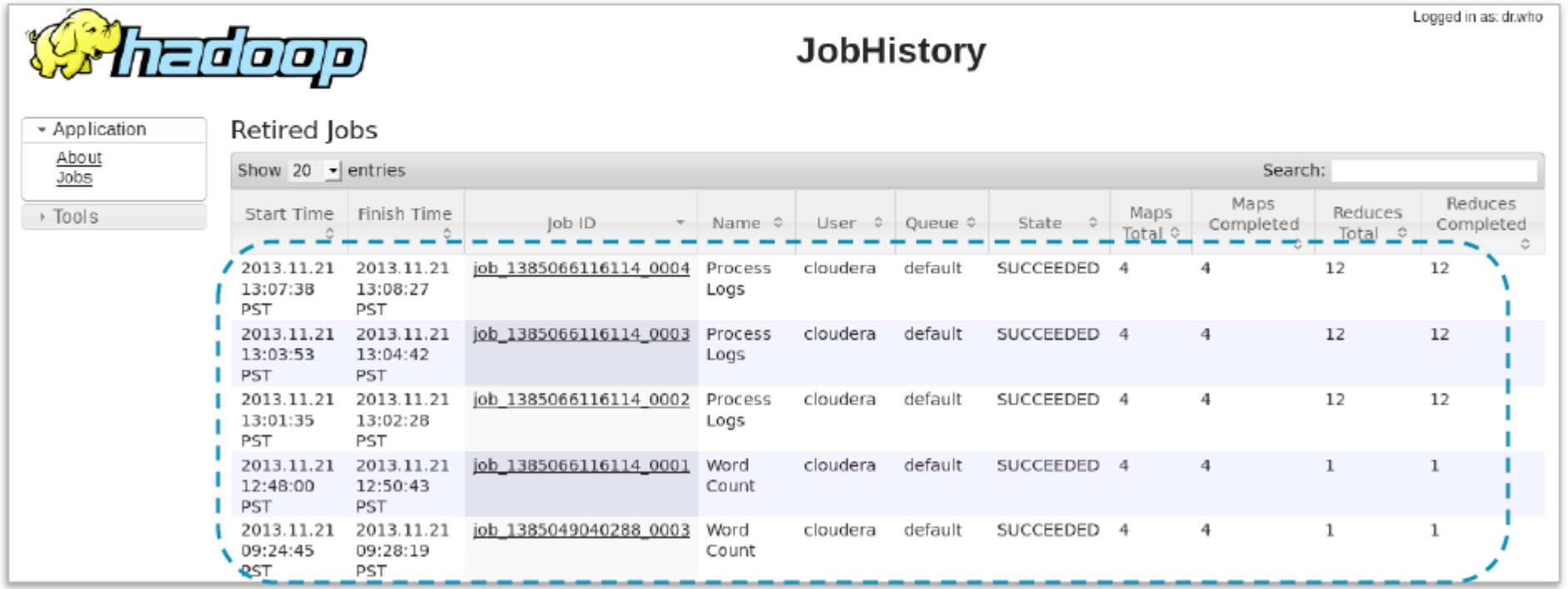


Resource Manager UI: Application Detail



Job History Server

- YARN does not keep track of job history
- Spark and MapReduce each provide a Job History Server
  - Archives job's metrics and metadata
  - Can be accessed through Job History UI or Hue



YARN Command Line

- Command to configure and view information about the YARN cluster
  - `yarn <command>`
- Most YARN command line tools are for administrators rather than developers
- Some helpful commands for developers
  - `yarn application`
    - Use `-list` to see running applications
    - Use `-kill` to kill a running application
  - `yarn logs -applicationId <app-id>`
    - View the logs of the specified application

Practice3: Run a YARN Job

- **In this homework, you will**
  - Use the YARN Web UI to view your YARN cluster “at rest”
  - Submit an application to run on the cluster
  - Monitor the job using both the YARN UI and Hue
- **Please refer to the Homework description**

Essential Points

- HDFS is the storage layer for Hadoop
- Chunks data into blocks and distributes them across the cluster when data is stored
- Slave nodes run DataNode daemons, managed by a single NameNode on a master node
- Access HDFS using Hue, the `hdfs` command or via the HDFS API
- YARN manages resources in a Hadoop cluster and schedules jobs
- YARN works with HDFS to run tasks where the data is stored
- Slave nodes run NodeManager daemons, managed by a ResourceManager on a master node
- Monitor jobs using Hue, the YARN Web UI or the yarn command