



**LESSON INTRODUCTION SLIDE**  
**DO NOT USE NUMBERING**

# SLIDE HEADER (56 POINT TEXT)

- First level text is 42 point
  - Second level text is 38 point
    - Third level is 38 point
- Slides should contain keywords and phrases
- Use only the bullets, fonts and sizing in this template
- Use the new slide option to maintain pre-configured layouts

# HEADERS ARE ALWAYS ALL CAPS

- First letter in a sentence is always capitalized
- Commands are almost always capitalized
  - Example: INSERT, SUM, CONCATENATE
- Proper nouns are capitalized
  - Exception: program names should follow official branding
    - Example: iPhone, GitHub
- Everything else should be left in lower case

# USE IMAGES WHEREVER POSSIBLE



# INTRODUCTION TO HADOOP YARN: THE NEXT GENERATION HADOOP APPLICATION AND JOB FRAMEWORK



O'REILLY®

# COURSE AGENDA

- Basic overview of the core Hadoop components: HDFS and YARN
- What is MapReduce?
- HDFS overview - getting to know the basics
- The "old" Hadoop architecture - What existed before YARN?



# COURSE AGENDA

- What is YARN?
- The YARN architecture and YARN daemons
- The concept of containers, applications and tasks in YARN
- How does Hadoop execute YARN Applications - a step by step guide
- Viewing YARN log files using the command line





# COURSE AGENDA

- YARN cluster resource allocation concepts
- Requesting resources in YARN: CPU and memory
- How does YARN handle job failures?
- Monitoring YARN Using the YARN web-UI and command line



# A LITTLE ABOUT ME

- CTO at NAYA Technologies - database consulting
- Oracle and Clouder certified
- Originally: Oracle DBA consultant and architect - 14 years
- Working with NoSQL / big data databases - 4 years
- Teaching Oracle and Hadoop - 5 years
- Experienced with Oracle, SQL Server, MySQL, PostgreSQL, Hadoop, MongoDB, etc...
- Living my passion...

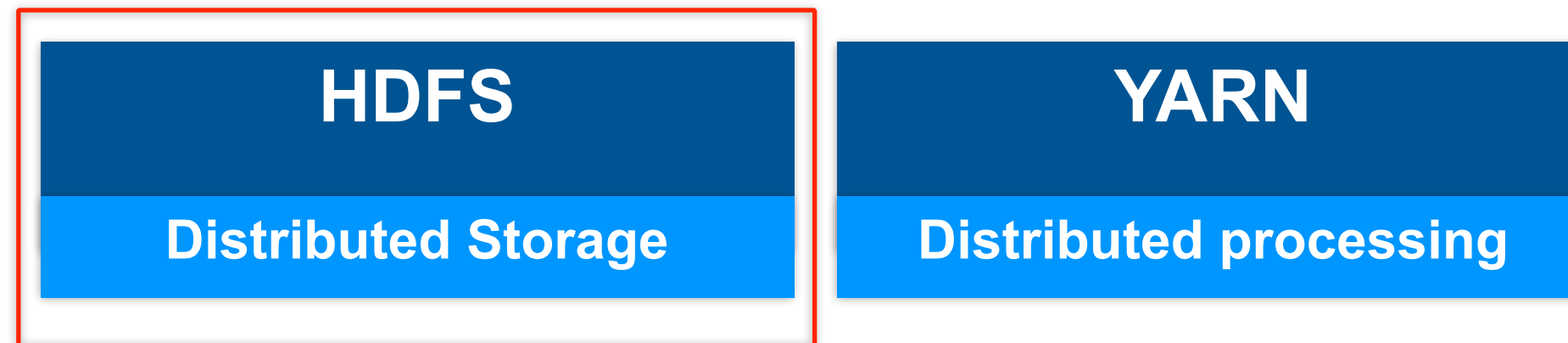
# CORE HADOOP 2.0



**“Core” Hadoop**

# HDFS IN A NUTSHELL

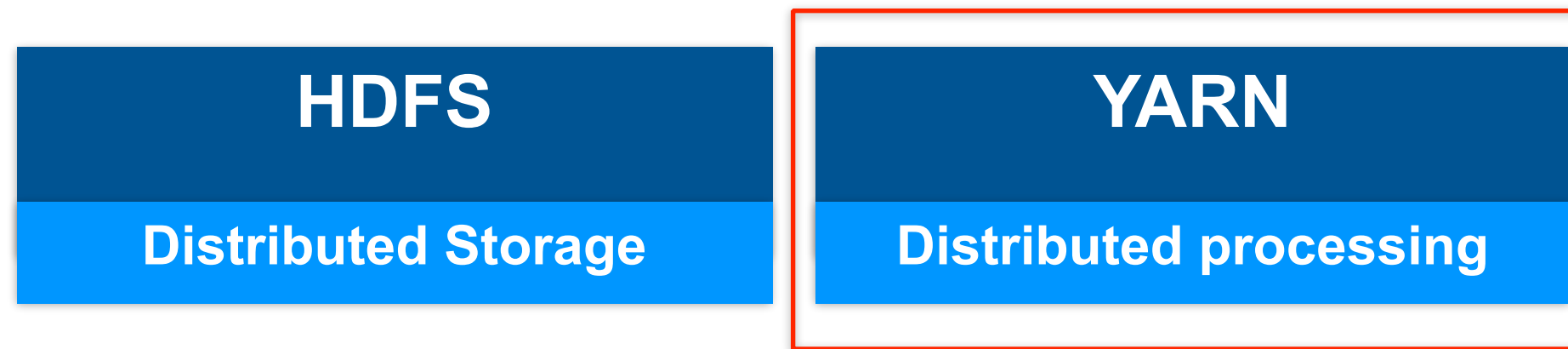
- HDFS facilitates data storage in Hadoop
- Provides redundant storage for massive amounts of data
- Provides parallel read performance when accessing data



**“Core” Hadoop**

# YARN IN A NUTSHELL

- YARN is the architecture for managing and controlling resources in an Hadoop cluster
- YARN allows for running jobs / applications



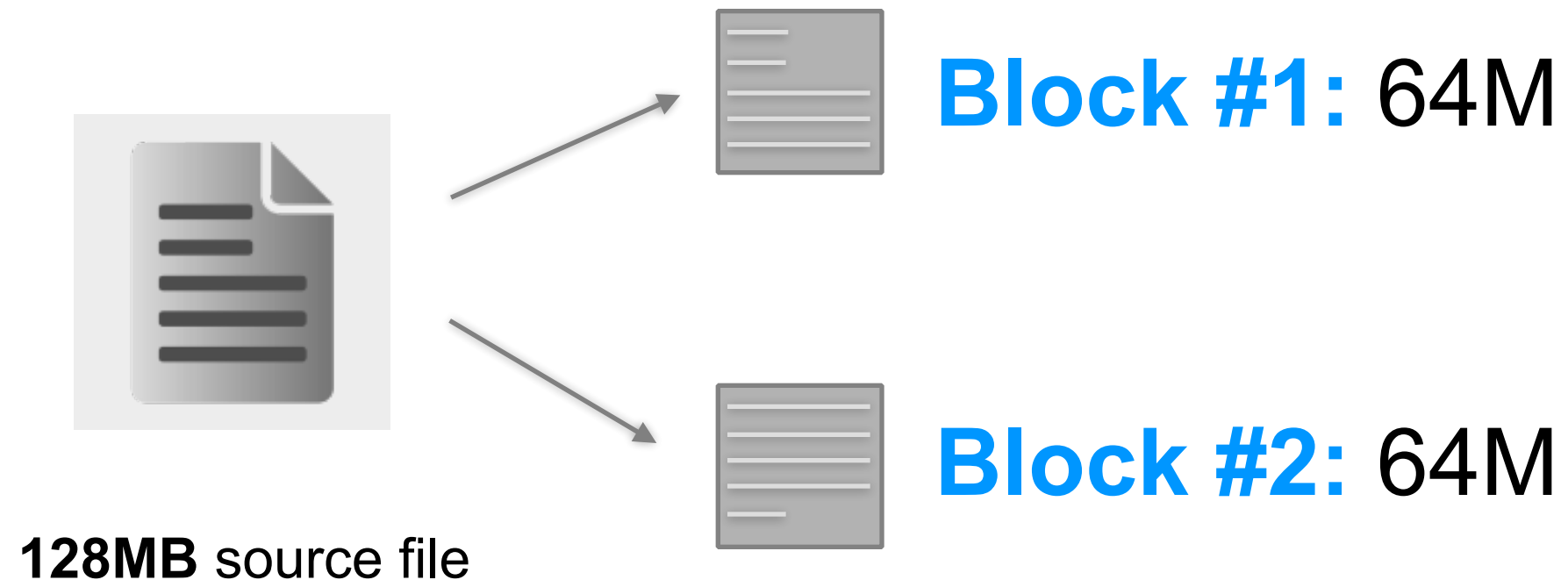
**“Core” Hadoop**

# HDFS: DIGGING DEEPER

- High performance
- Fault tolerant
- Simple management: master / slave architecture
- Optimized for distributed processing of data
- Allows for easy scalability

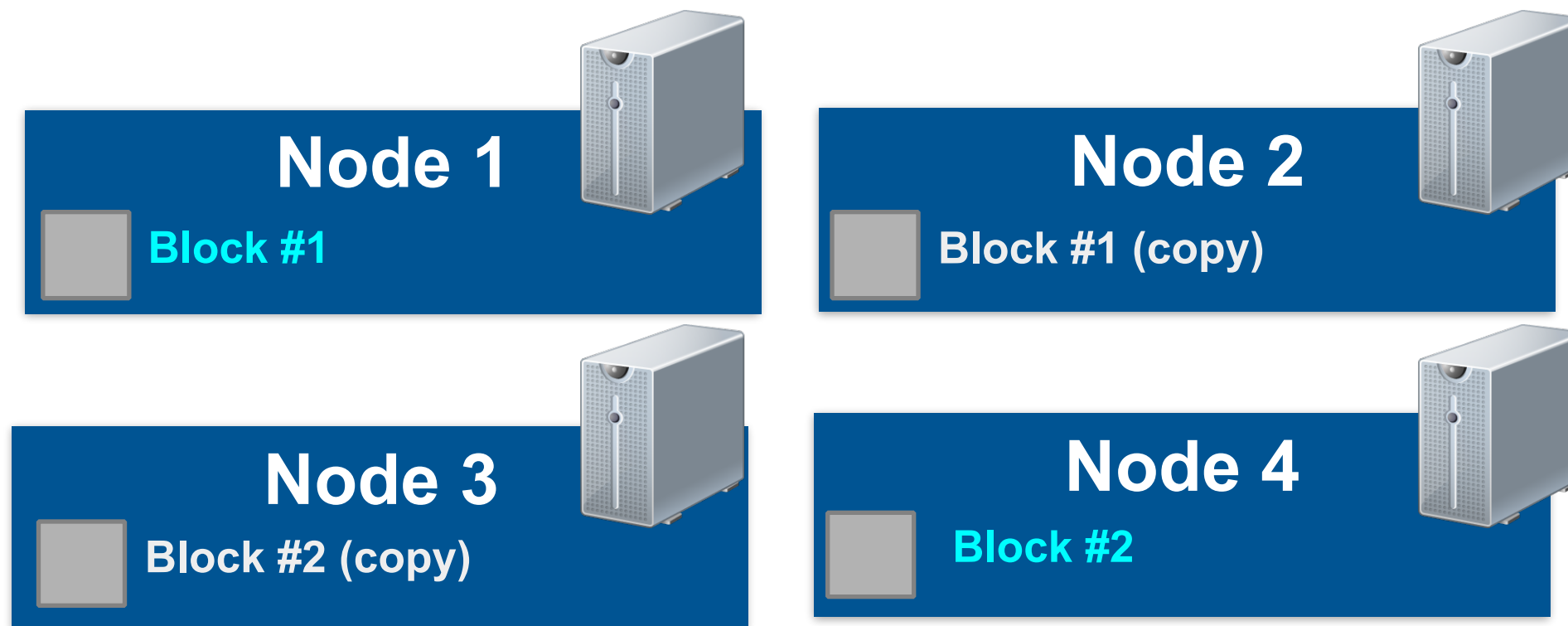
# HDFS: DIGGING DEEPER

- When a file is uploaded to HDFS, it is stripped into “blocks”



# HDFS: DIGGING DEEPER

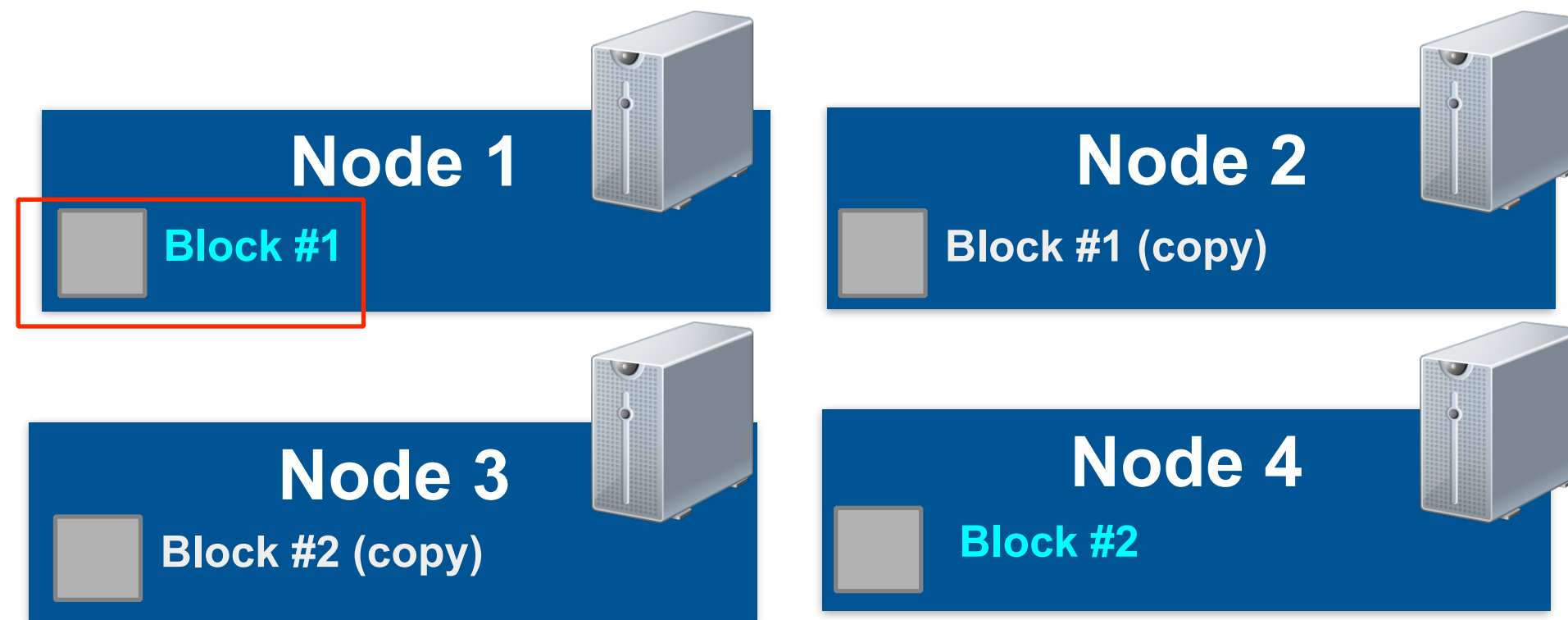
- Each block is stored on a different Hadoop node + replicas:





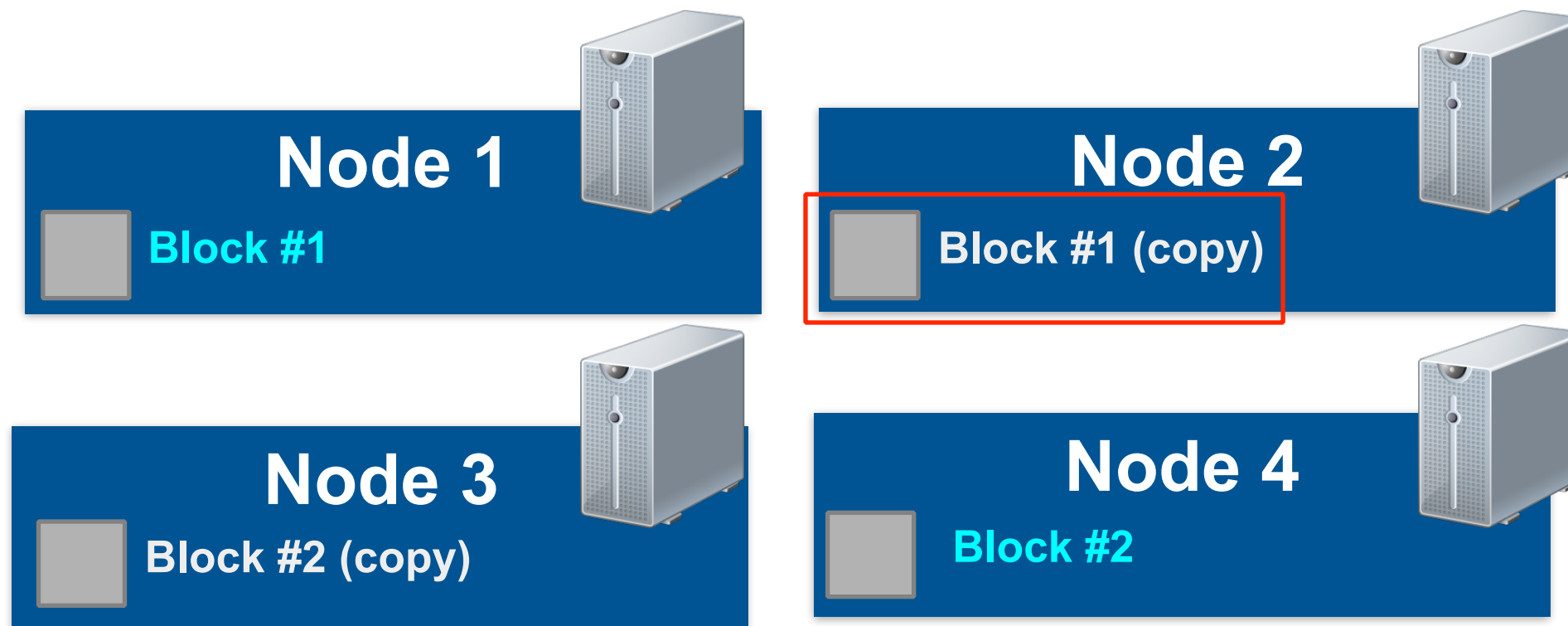
# HDFS: DIGGING DEEPER

- Each block is stored on a different Hadoop node + replicas:



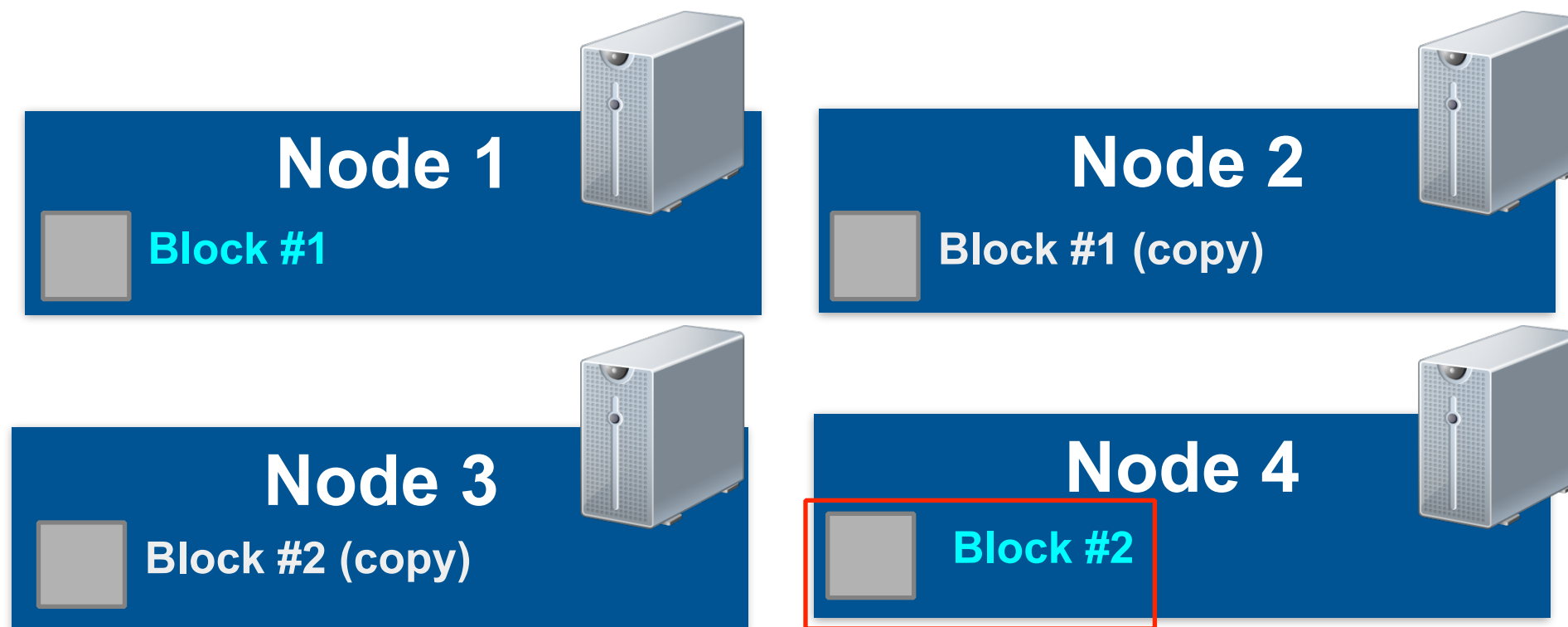
# HDFS: DIGGING DEEPER

- Each block is stored on a different Hadoop node + replicas:



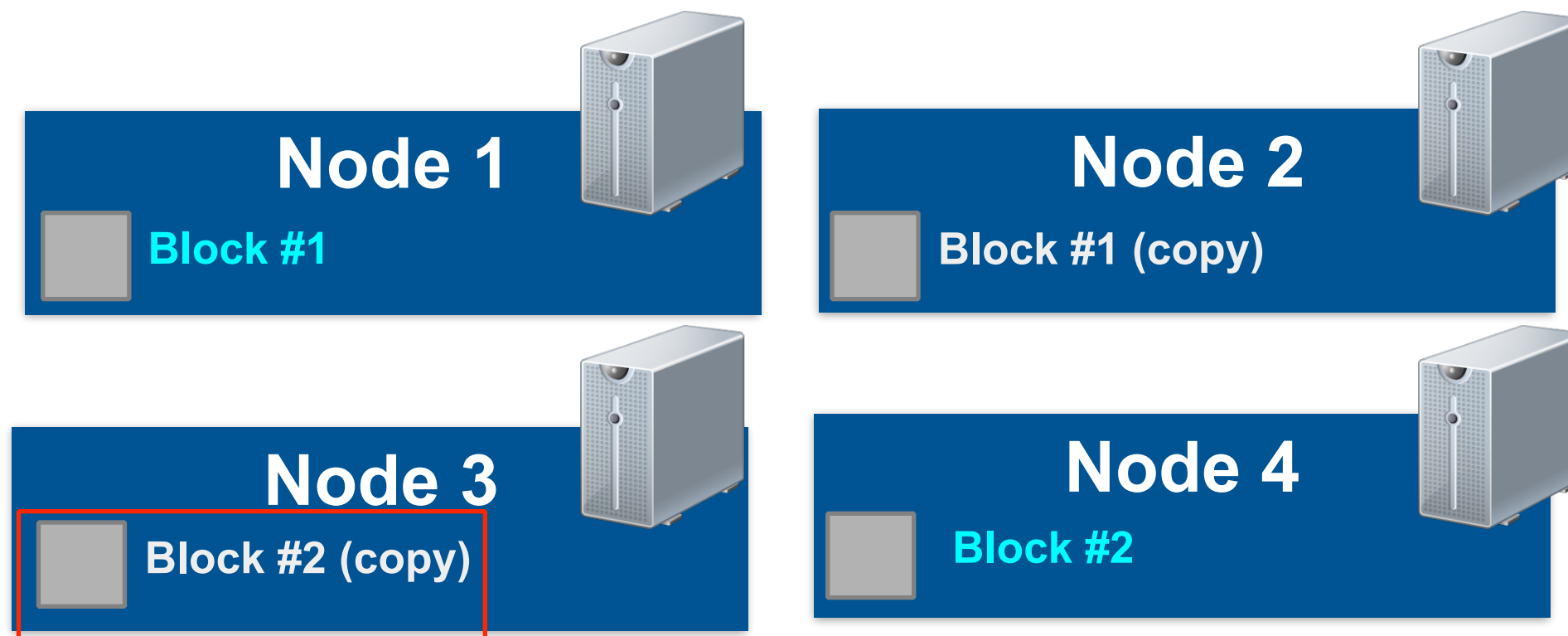
# HDFS: DIGGING DEEPER

- Each block is stored on a different Hadoop node + replicas:



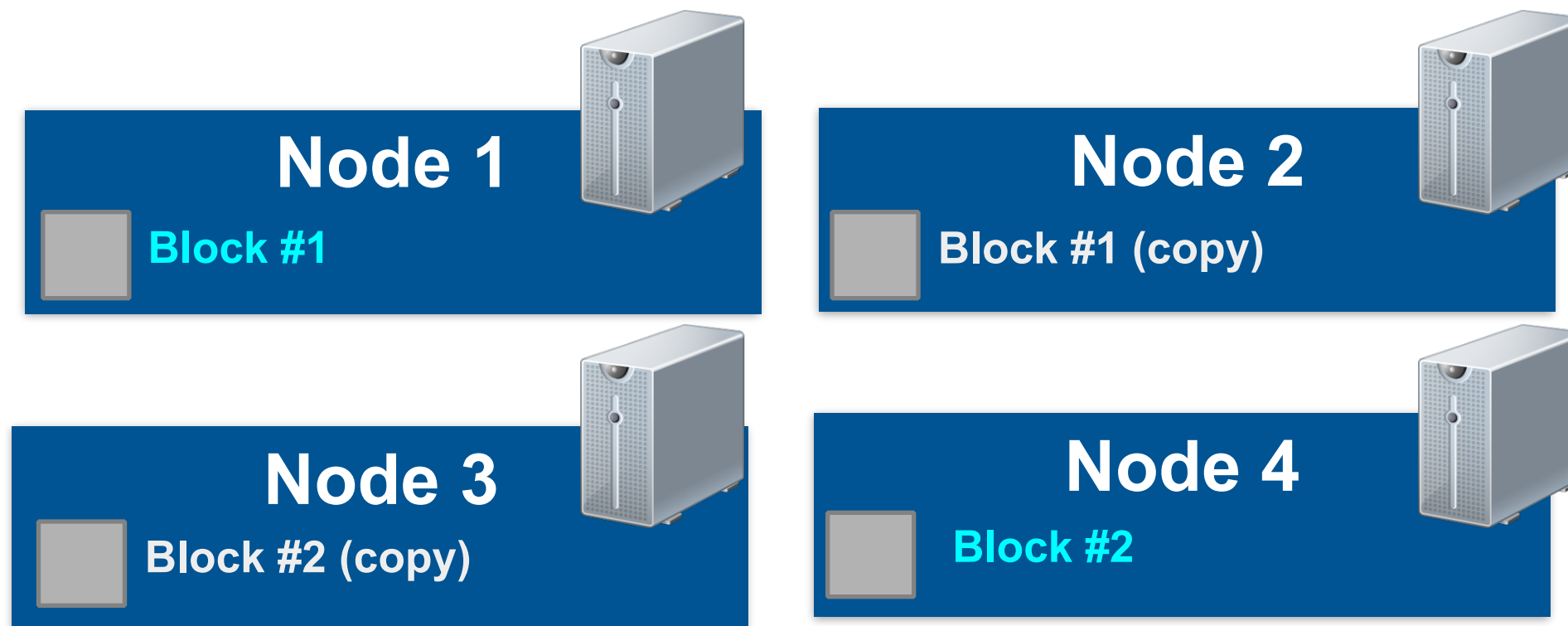
# HDFS: DIGGING DEEPER

- Each block is stored on a different Hadoop node + replicas:



# HDFS: DIGGING DEEPER

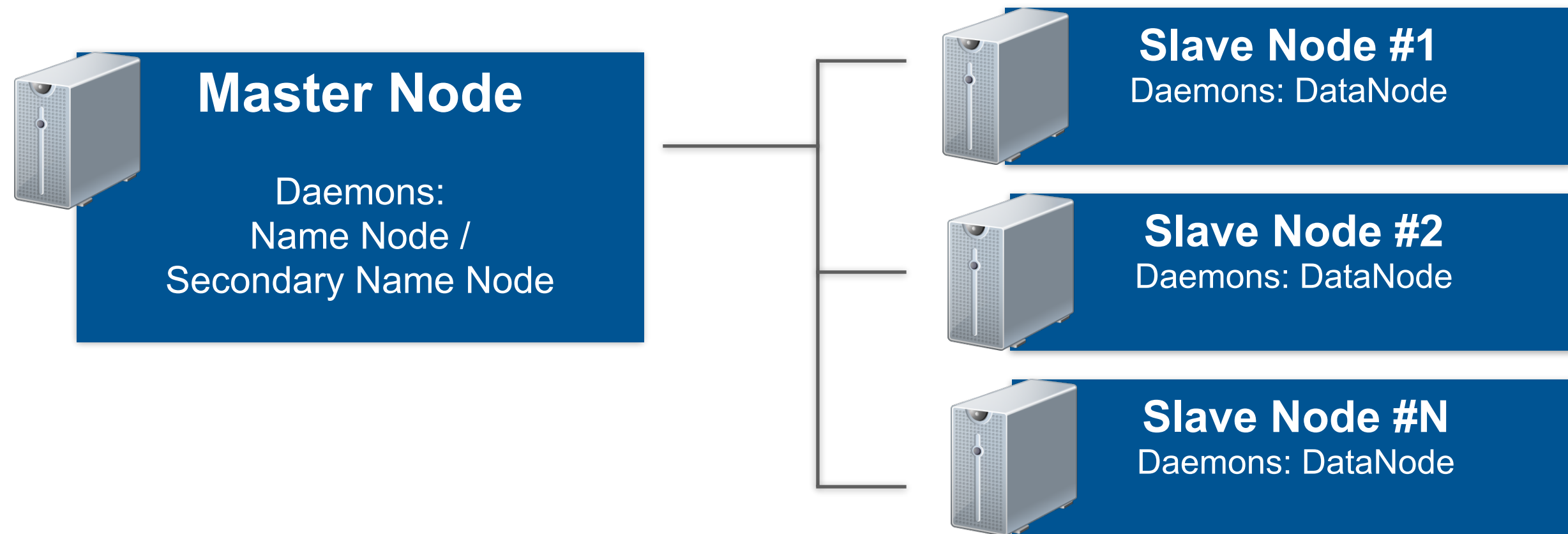
- Each block is stored on a different Hadoop node + replicas:



# HDFS: DIGGING DEEPER

- Master node (Name Node / Secondary Name Node daemons): stores HDFS metadata.
- Slave Node (DataNode daemons): store actual data blocks (which make up HDFS files)

# HDFS: DIGGING DEEPER

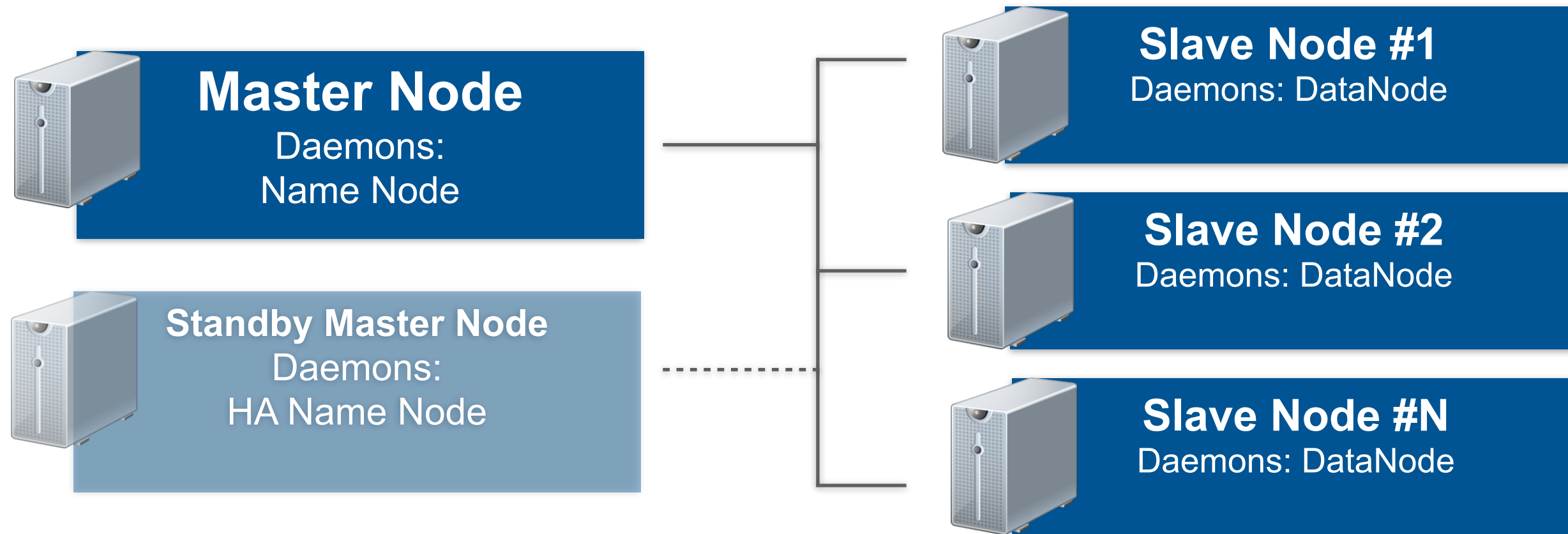


# HDFS: DIGGING DEEPER

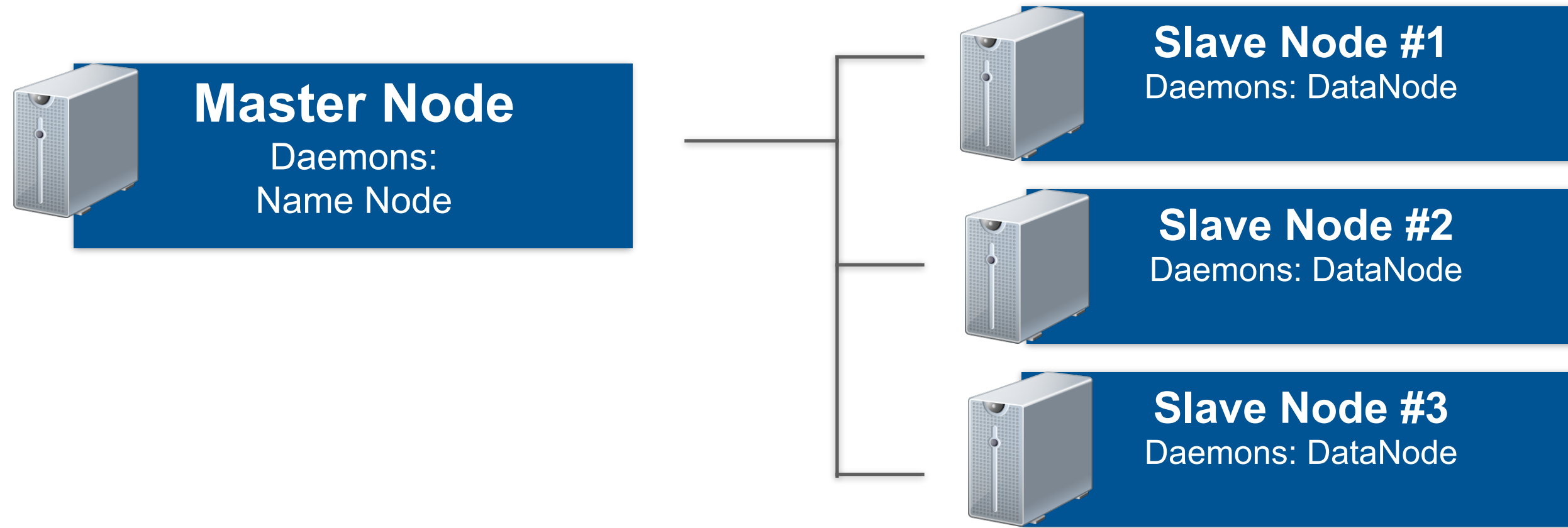
- Master node (Name Node daemon): stores HDFS metadata.
- Standby Master node (HA Name Node daemon): high availability for master node.
- Slave Node (DataNode daemons): store actual data blocks (which make up HDFS files)



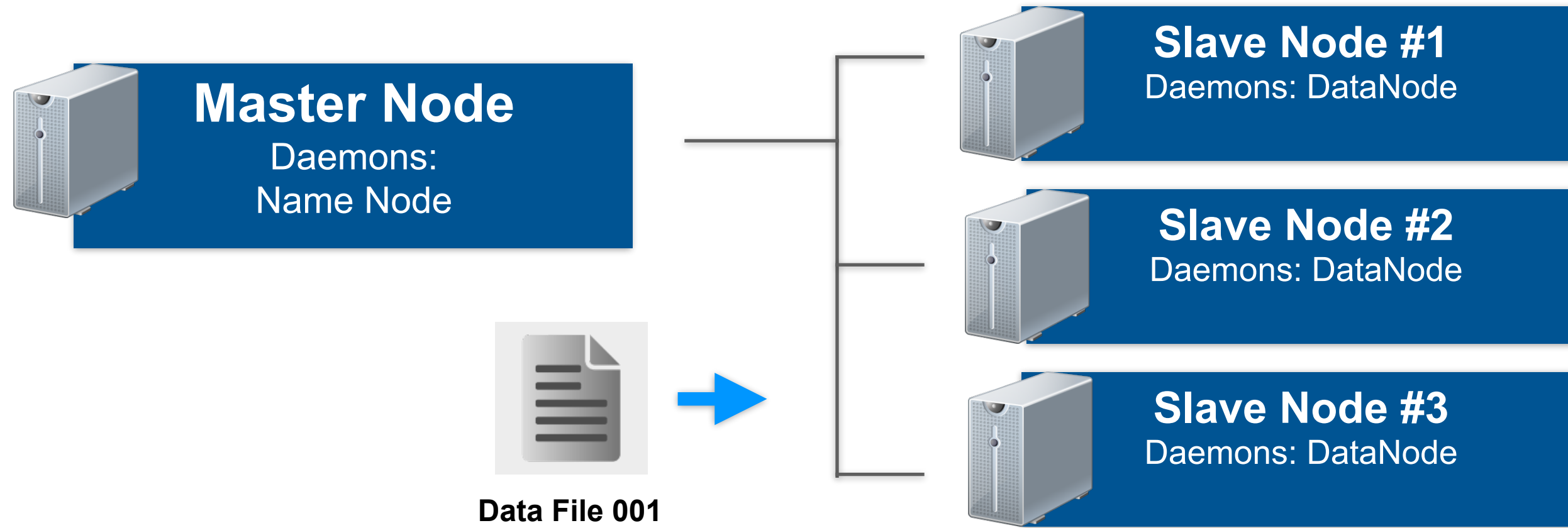
# HDFS: DIGGING DEEPER



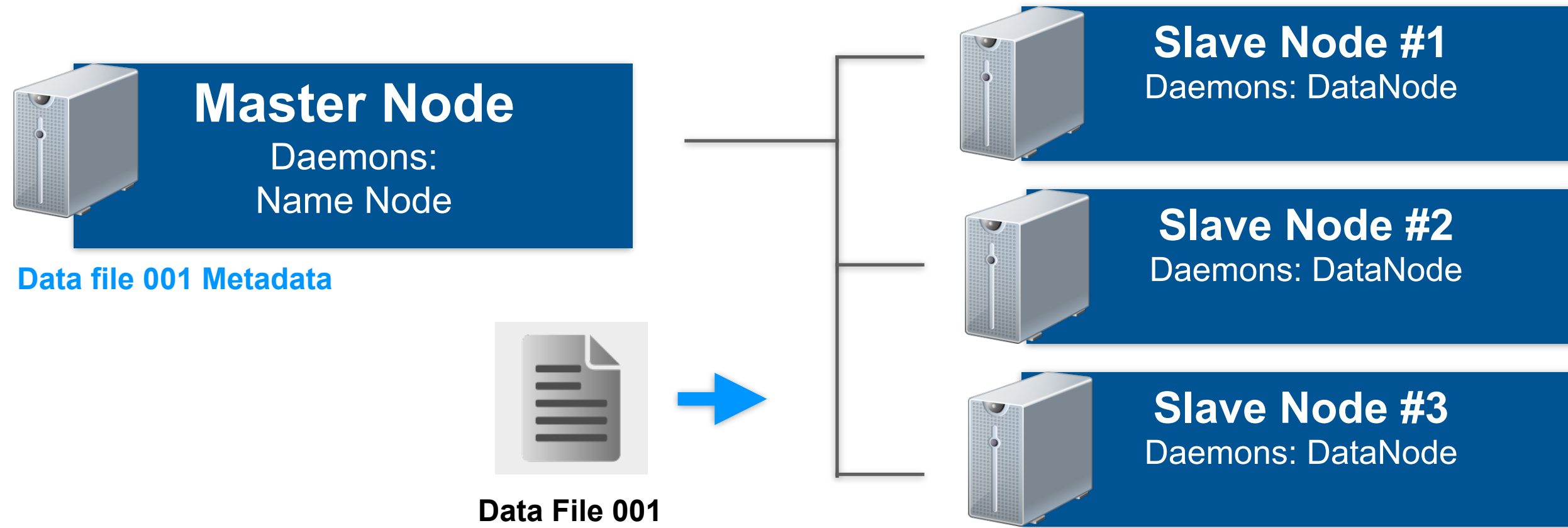
# HDFS: SCALABILITY



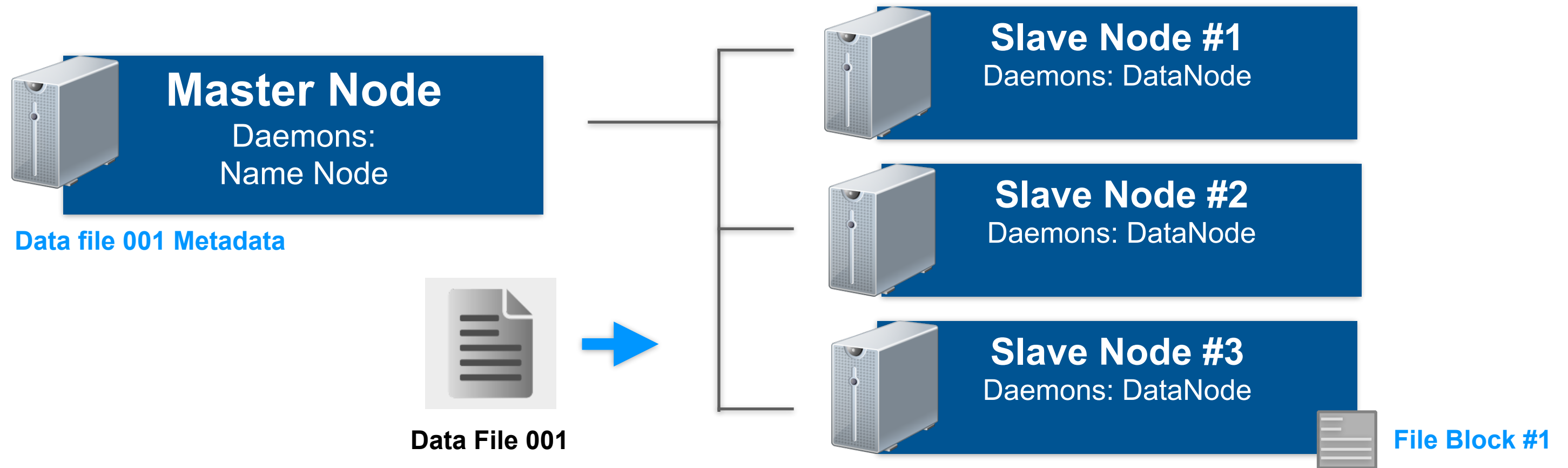
# HDFS: SCALABILITY



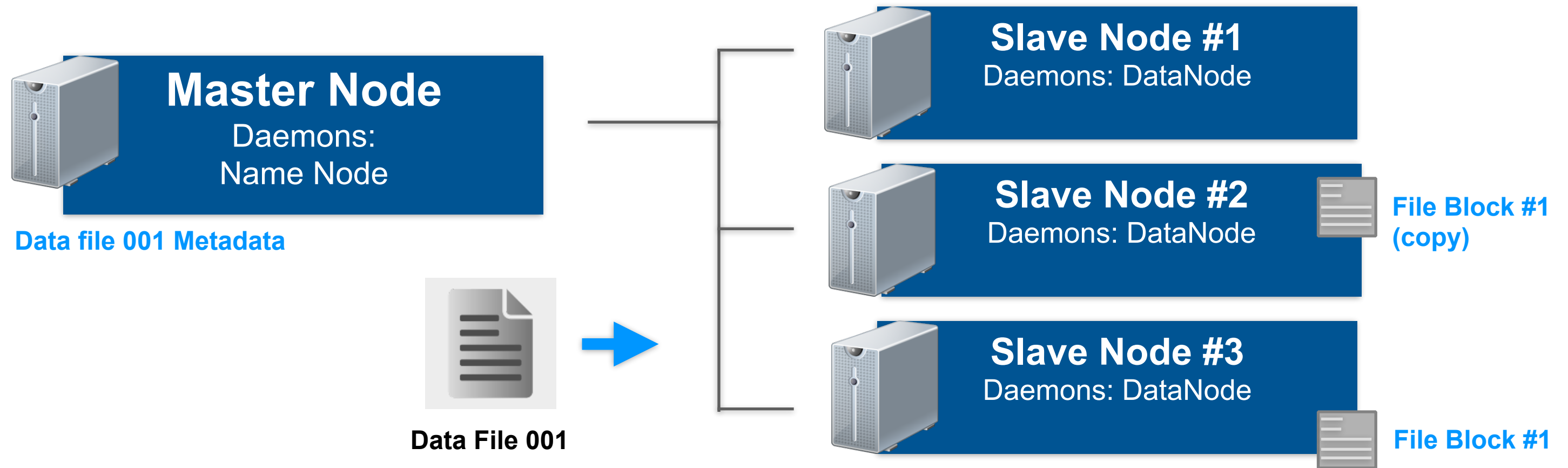
# HDFS: SCALABILITY



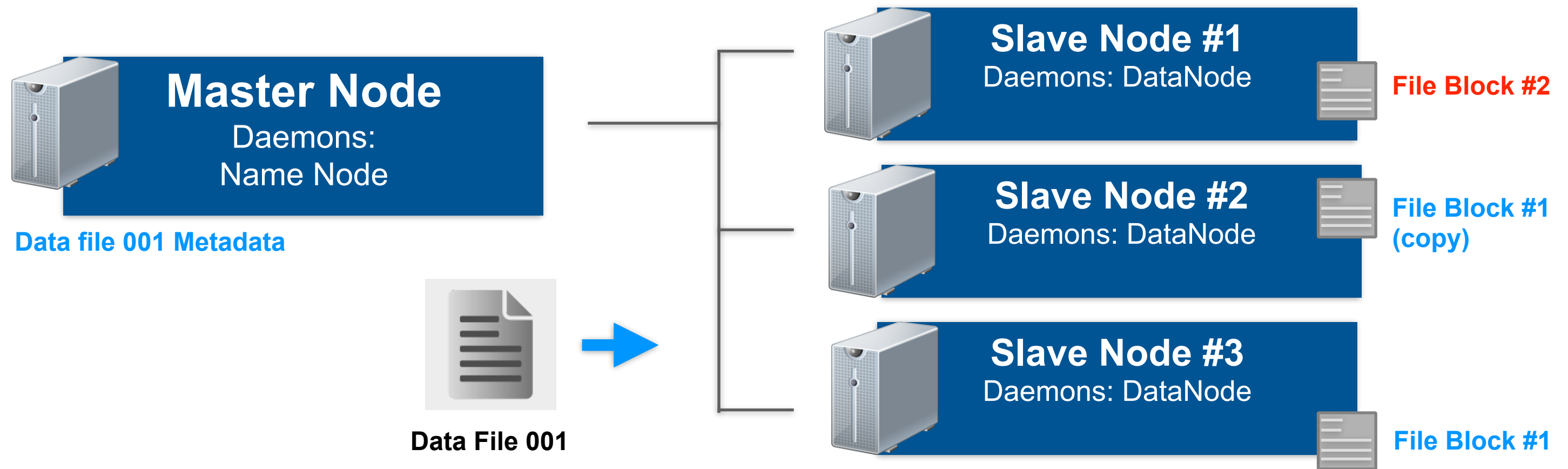
# HDFS: SCALABILITY



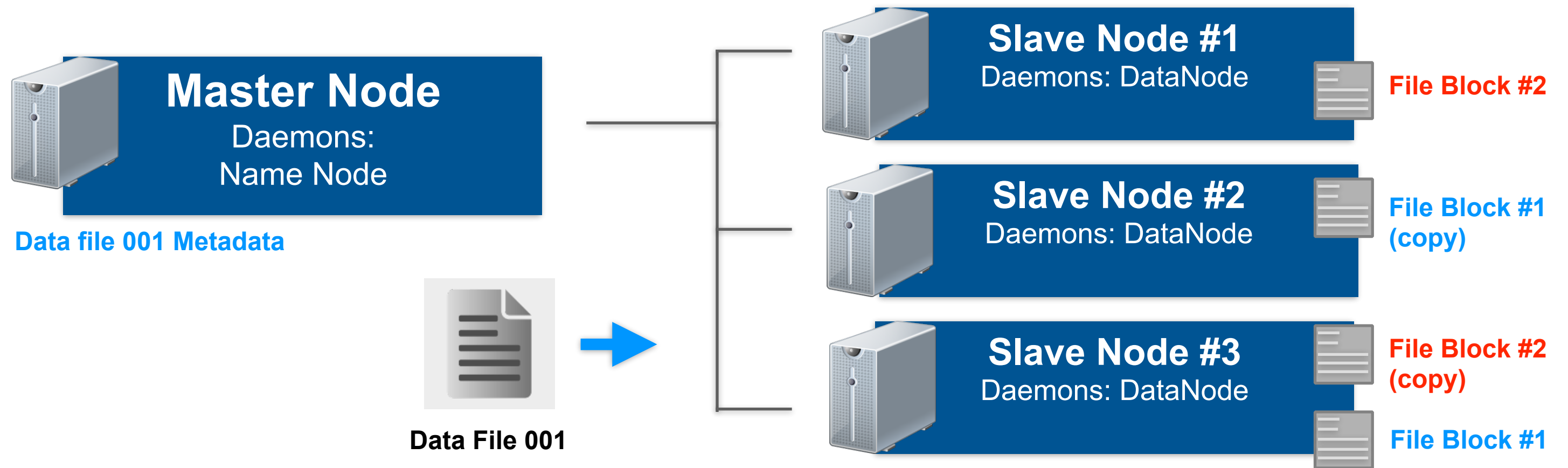
# HDFS: SCALABILITY



# HDFS: SCALABILITY

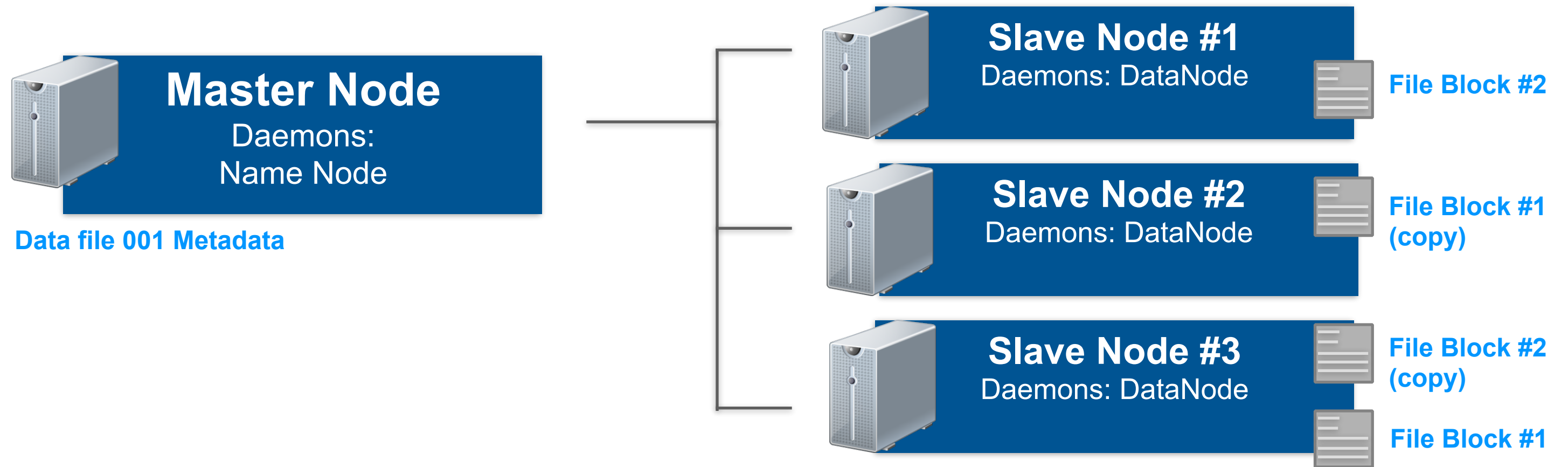


# HDFS: SCALABILITY

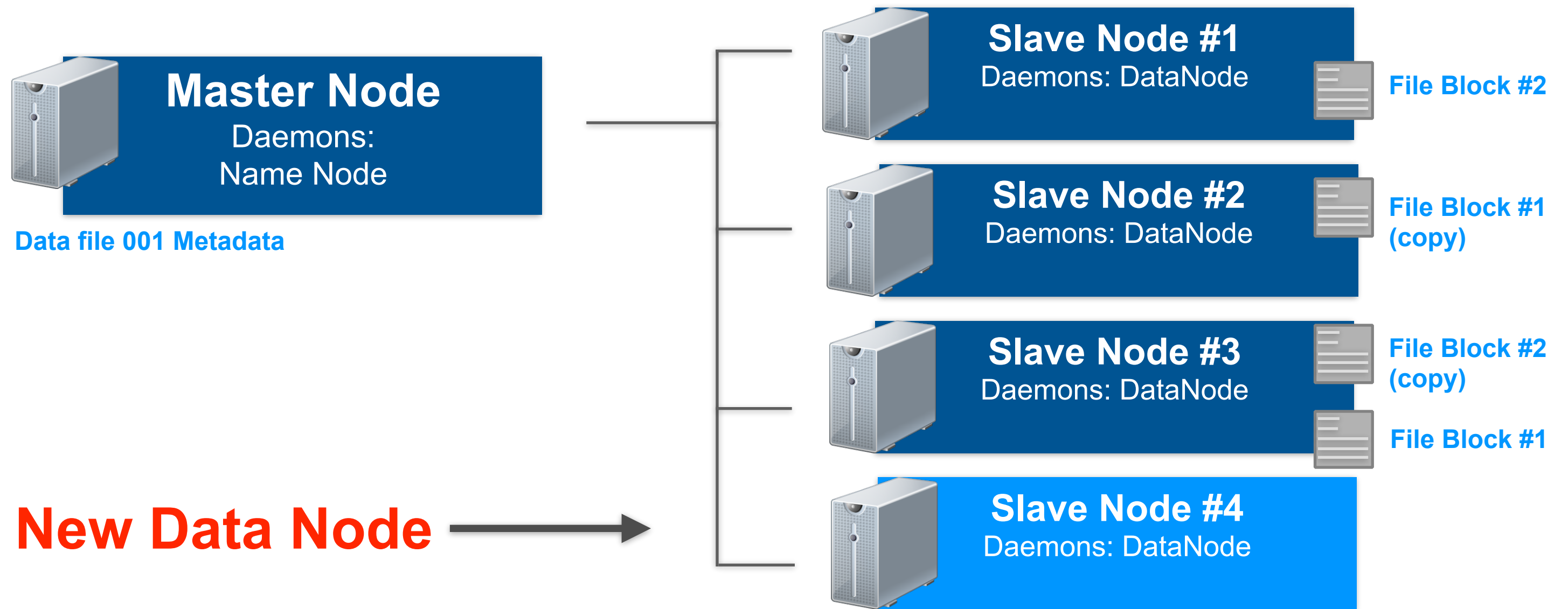




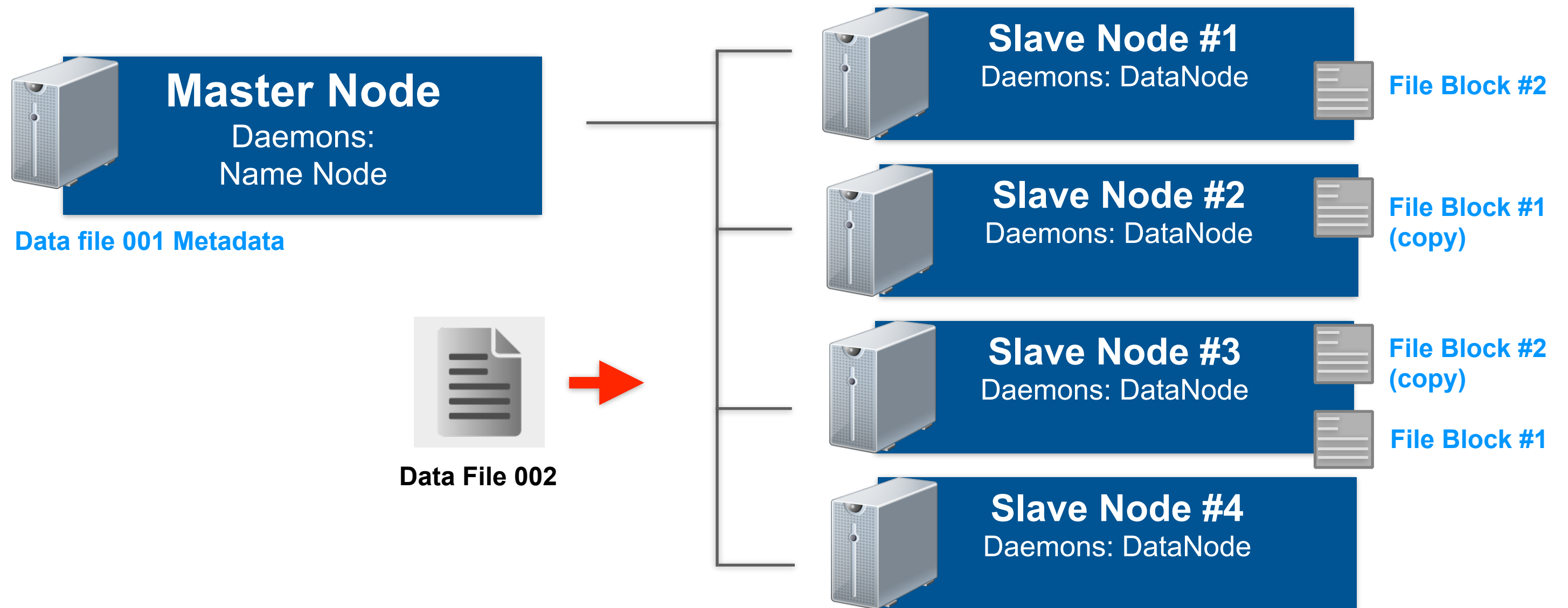
# HDFS: SCALABILITY



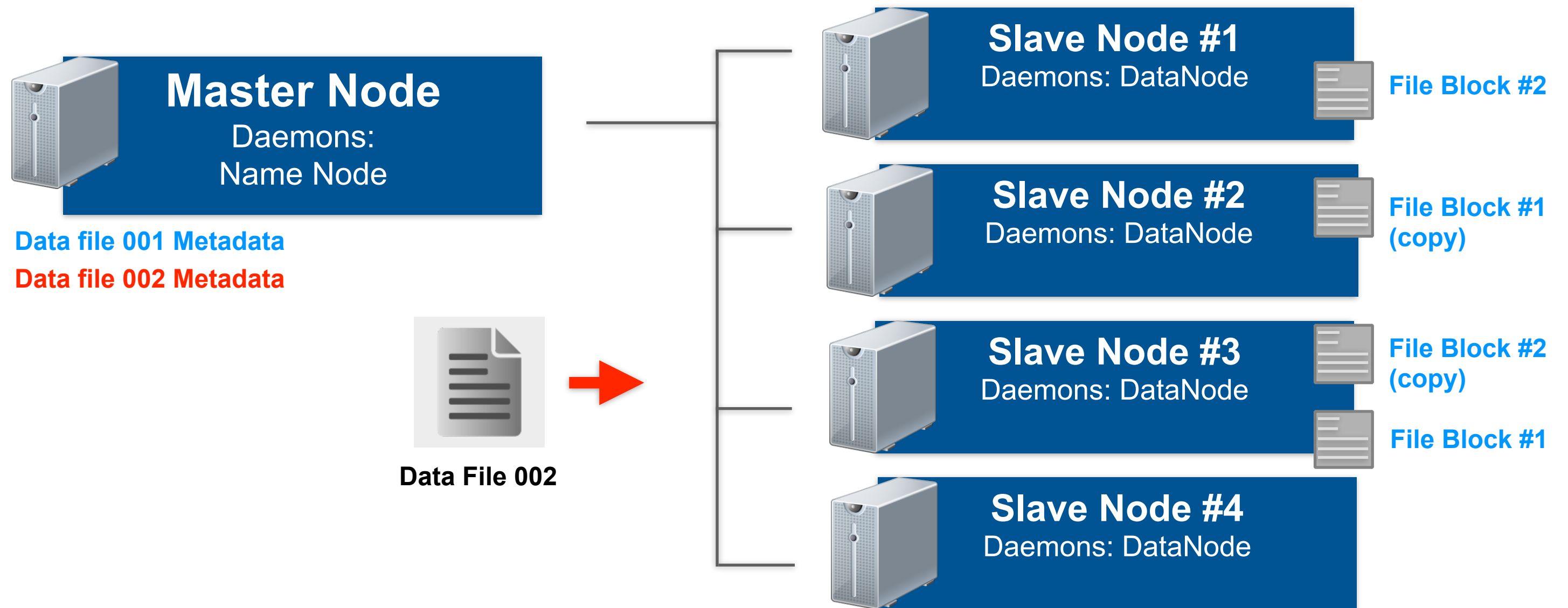
# HDFS: SCALABILITY



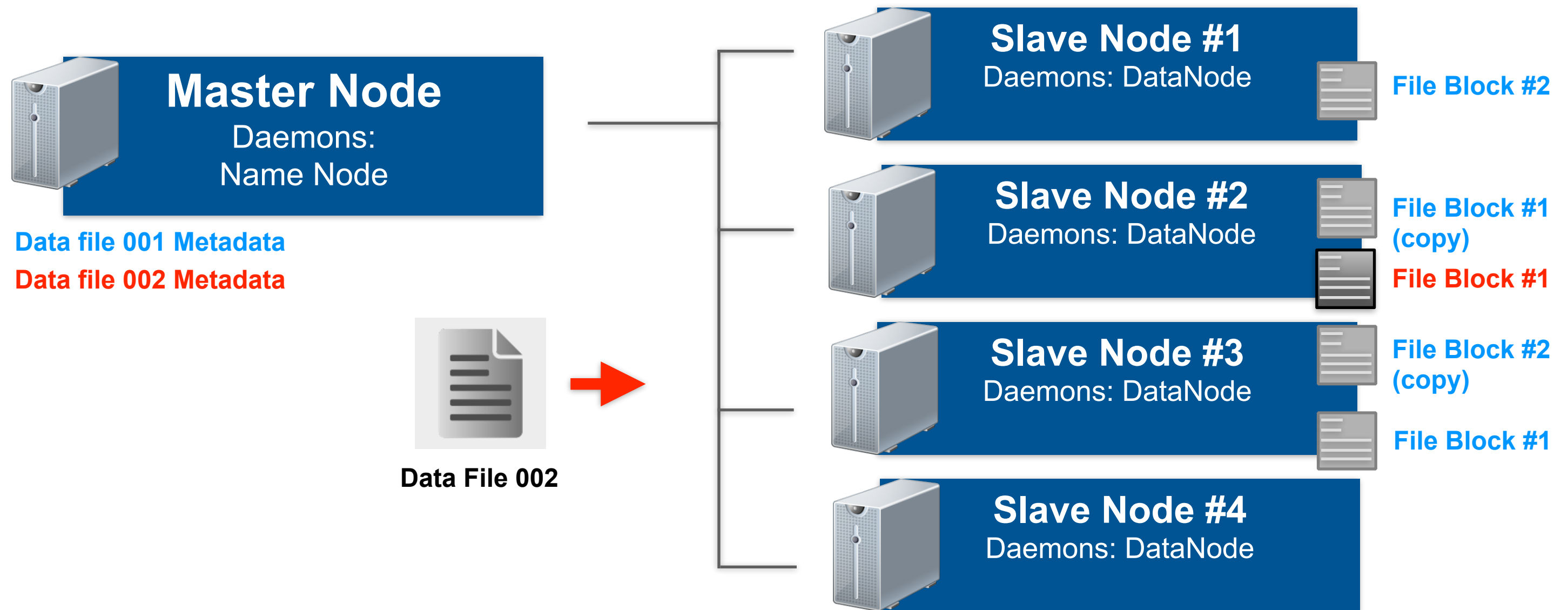
# HDFS: SCALABILITY



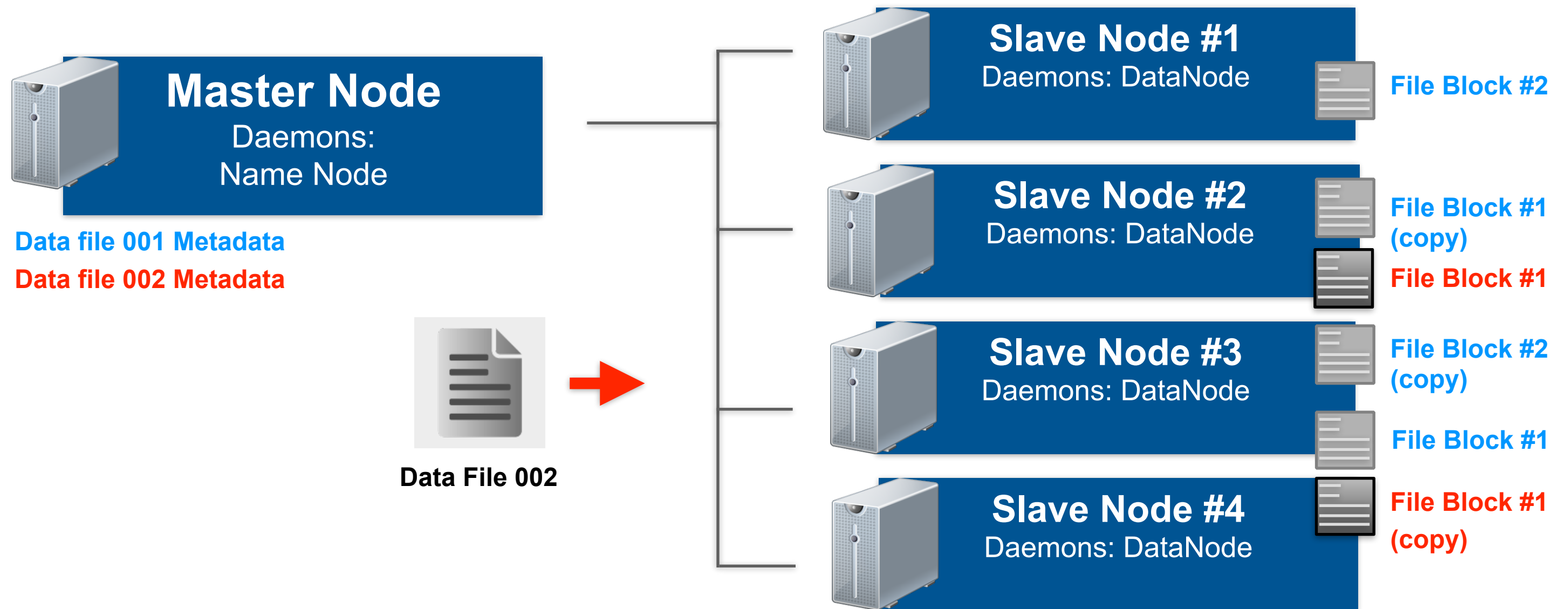
# HDFS: SCALABILITY



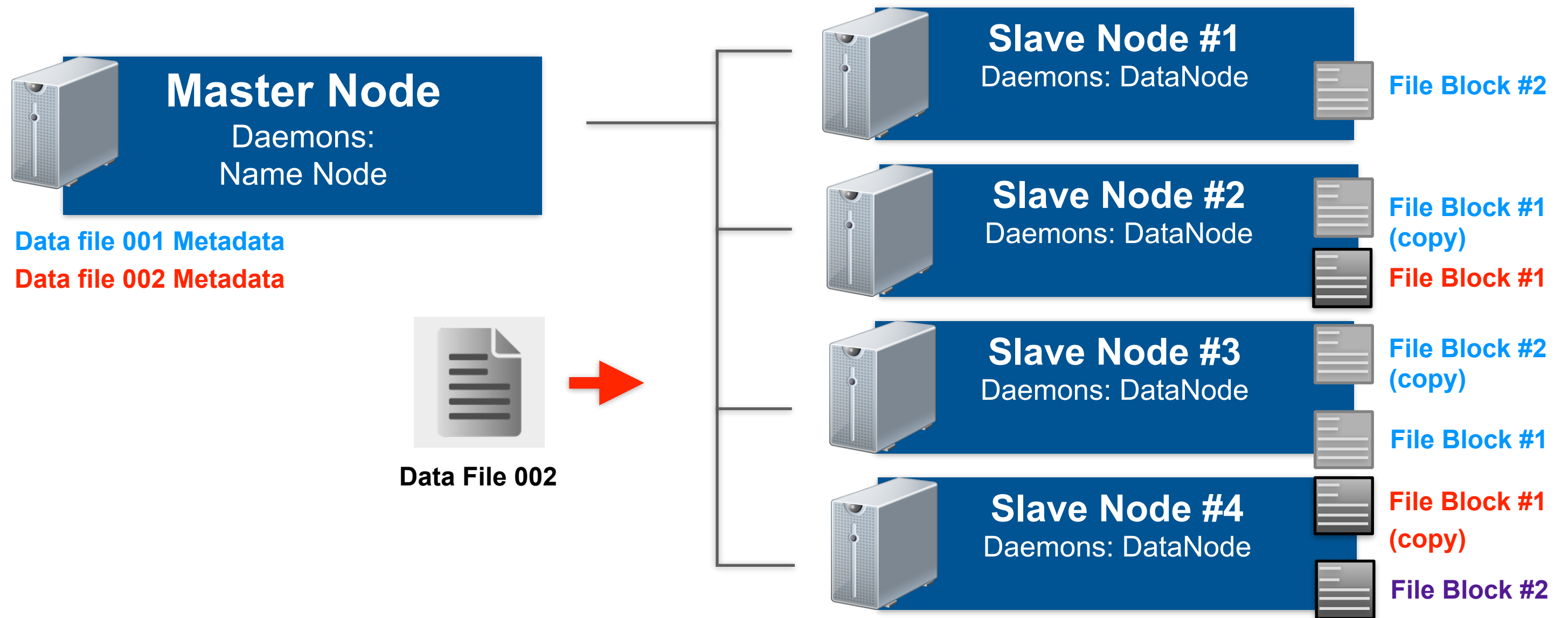
# HDFS: SCALABILITY



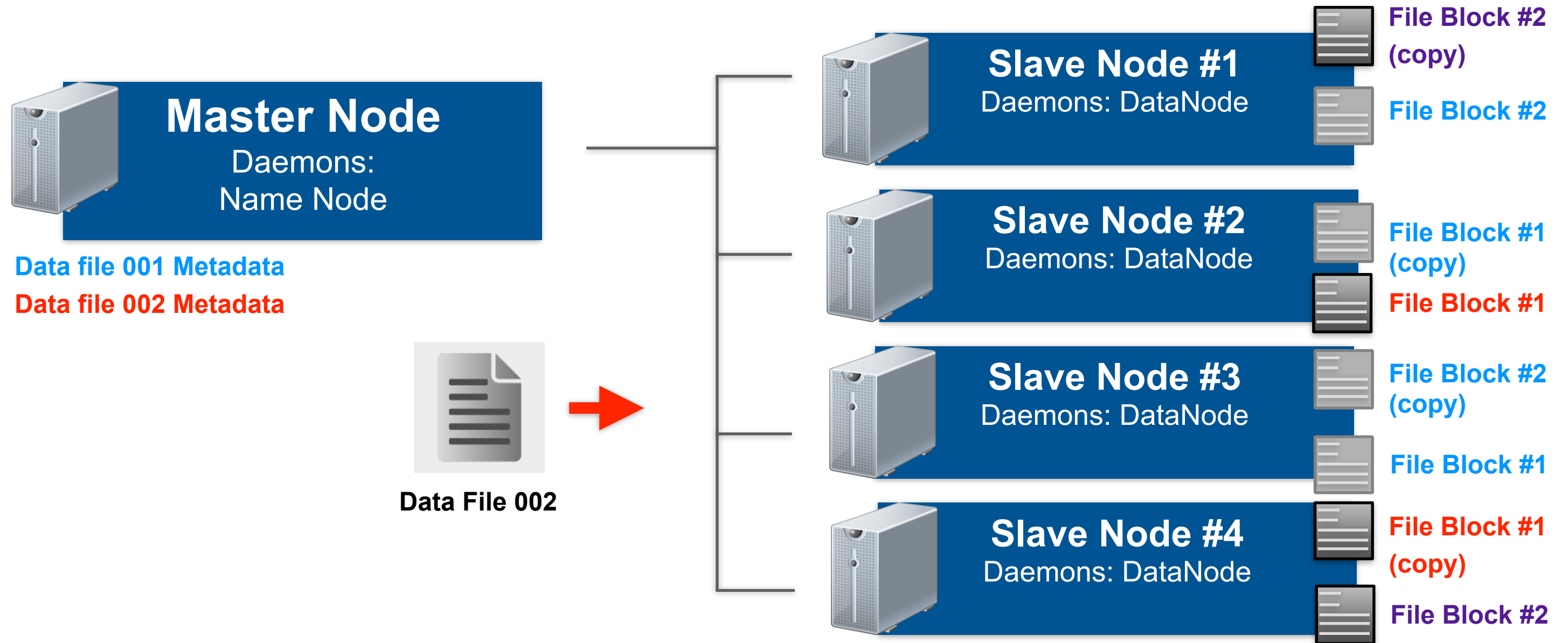
# HDFS: SCALABILITY



# HDFS: SCALABILITY

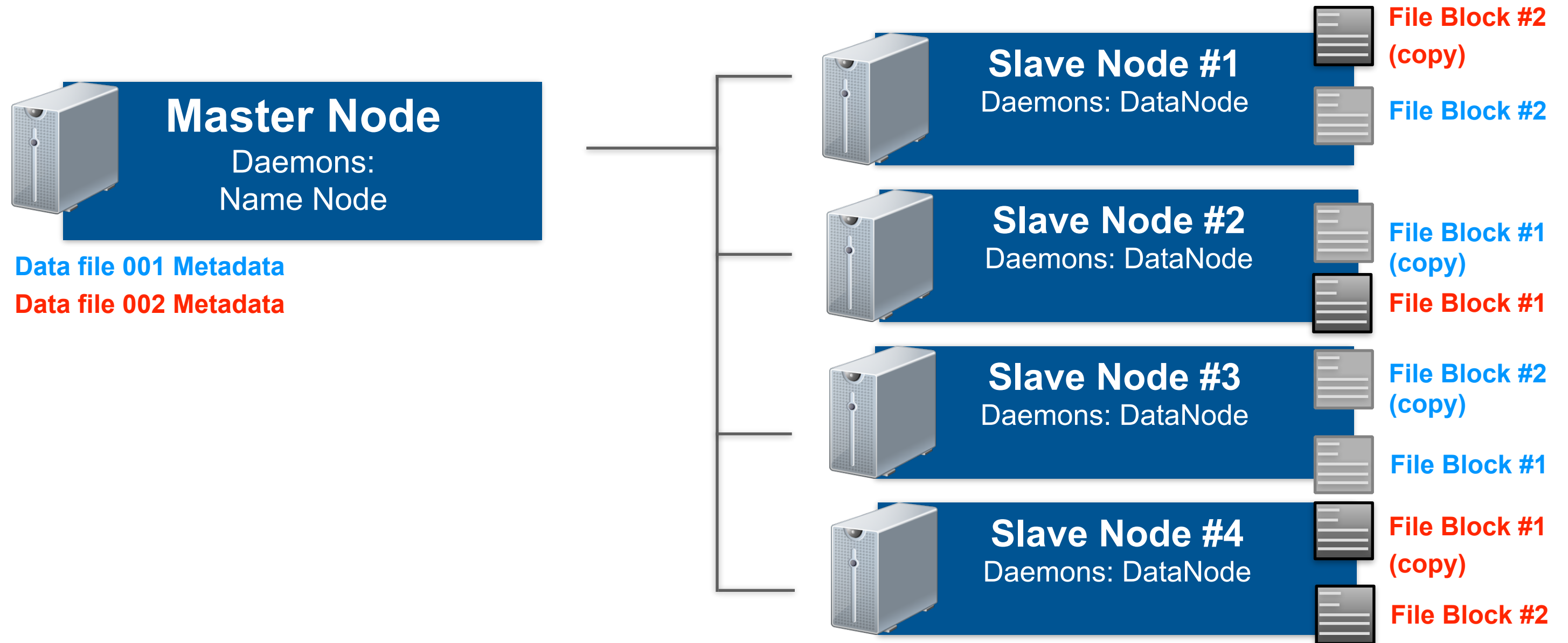


# HDFS: SCALABILITY





# HDFS: SCALABILITY



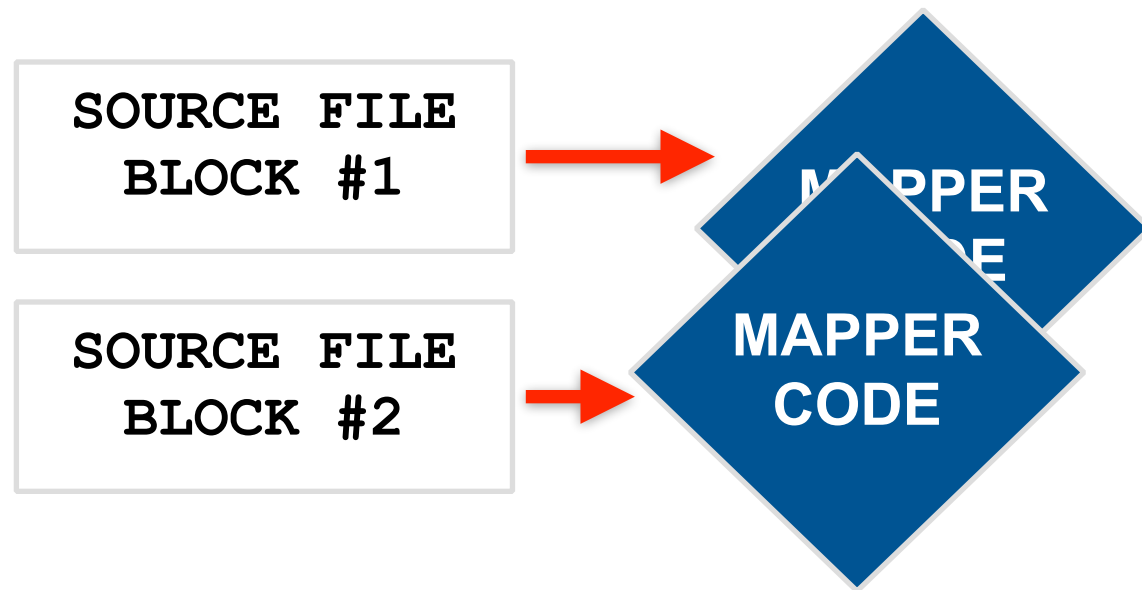
# MAPREDUCE OVERVIEW

- Provides the **framework** for developers to write scalable code easily
- MapReduce is a ***programming model*** in Hadoop and not a specific language
- Always processes records in ***key/value format***
- Allows for data distribution between nodes

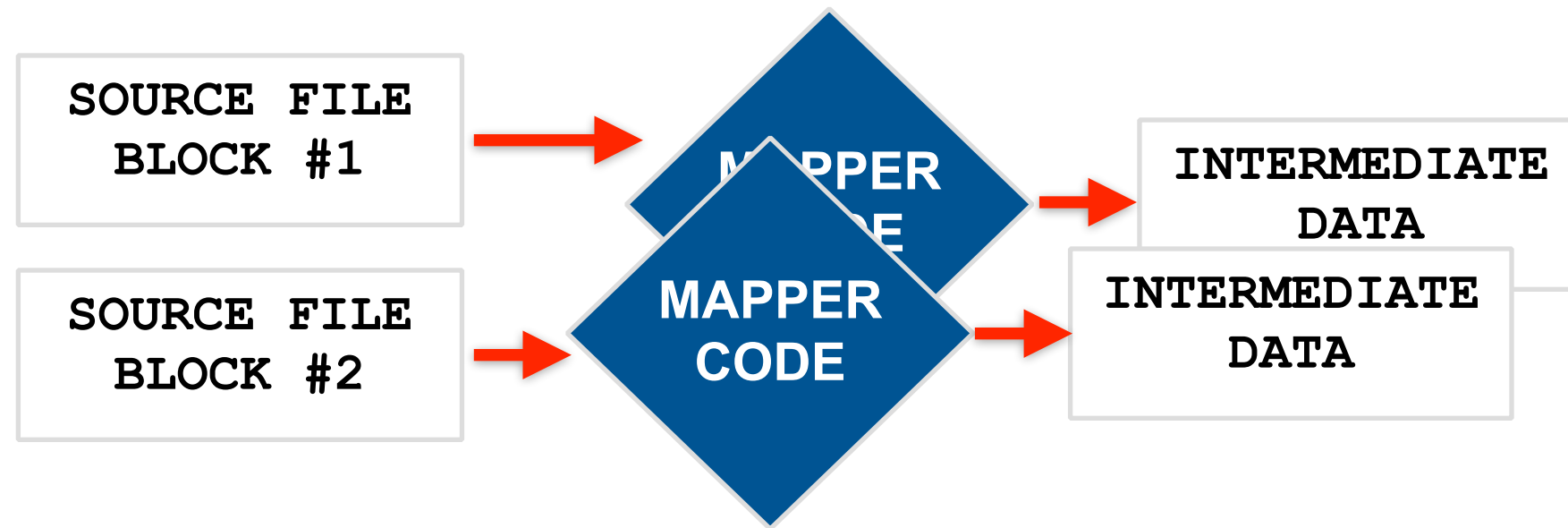
# MAPREDUCE OVERVIEW

- Requires a developer to ***only*** write two phases in a given program:
  - (1) Map
  - (2) Reduce
- Allows the developer to focus on ***business logic***

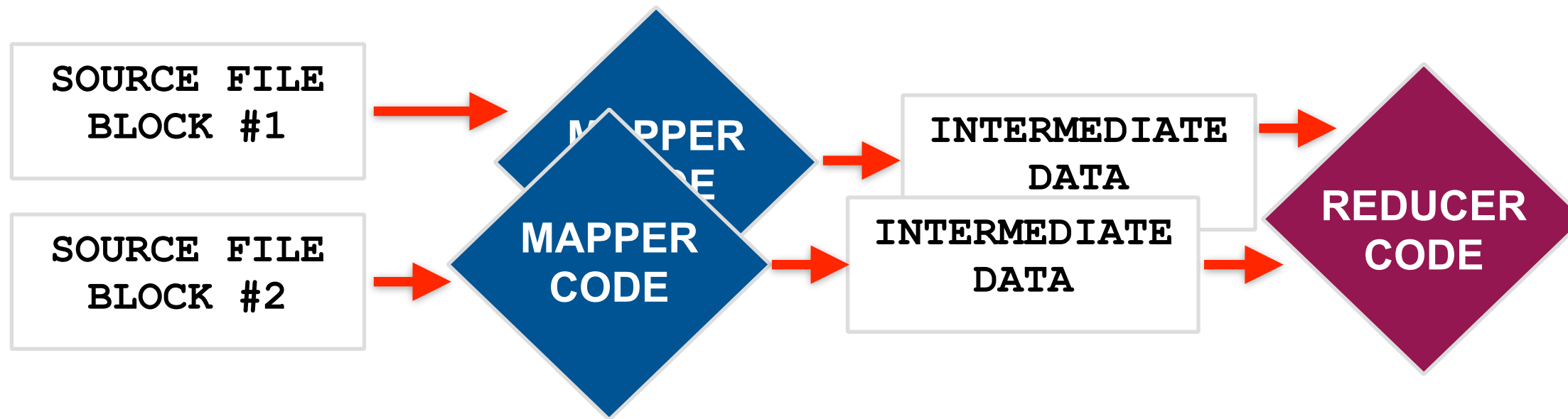
# MAPREDUCE OVERVIEW



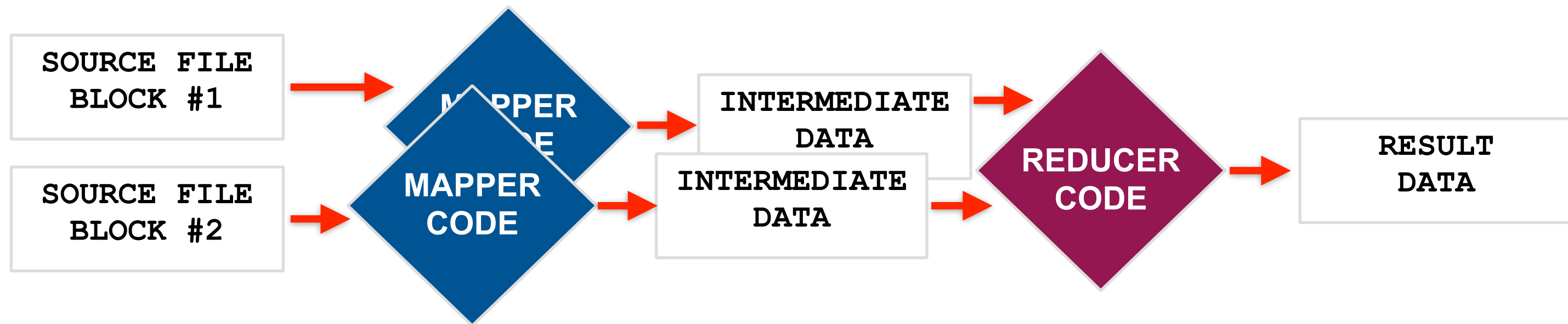
# MAPREDUCE OVERVIEW



# MAPREDUCE OVERVIEW



# MAPREDUCE OVERVIEW



# MAPREDUCE CAPABILITIES

- Provides automatic *parallelization* of data processing
- Provides *fault tolerance* for MapReduce tasks and handles failures
- Takes care of *job monitoring*
- *Abstraction for programs* - takes care of all the “nitty gritty” and hides it from developers
- MapReduce is *usually written in JAVA*



# MAPREDUCE DEMO

```
001 The car has four wheels  
002 The bike has two wheels
```

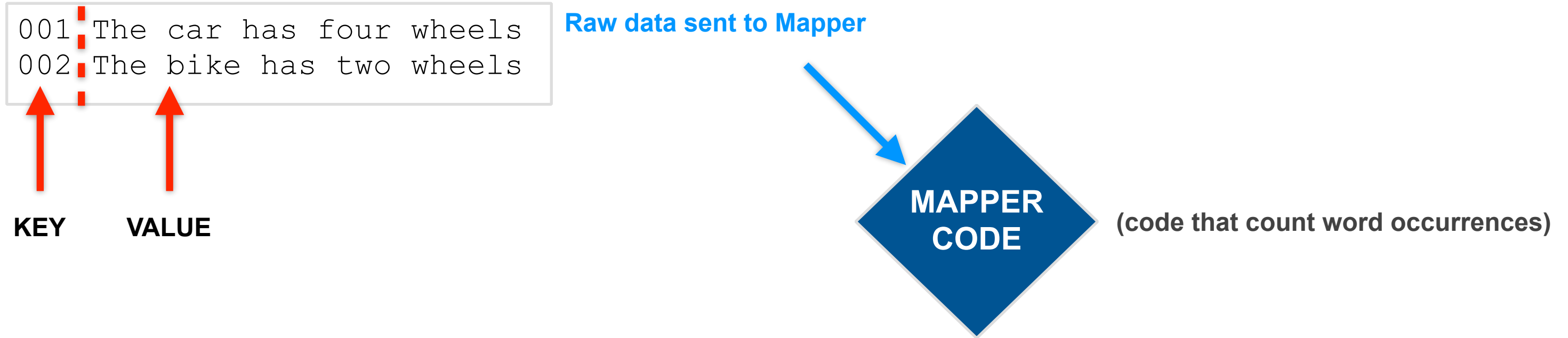
Raw data sent to Mapper



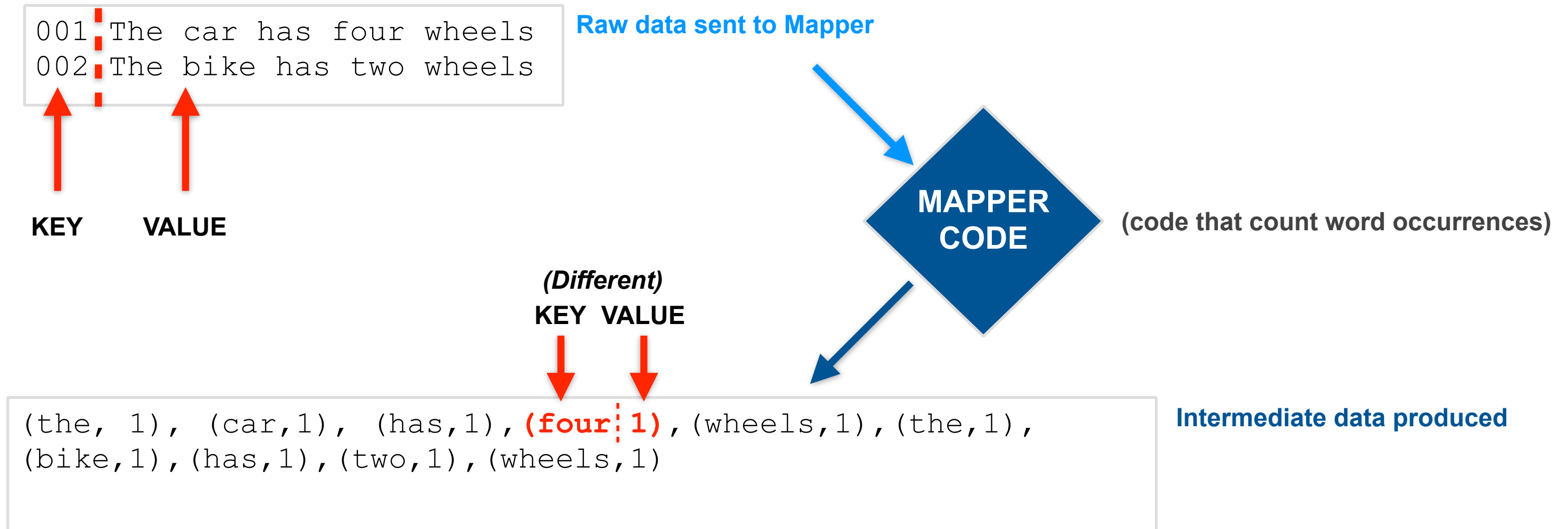
MAPPER  
CODE

(code that count word occurrences)

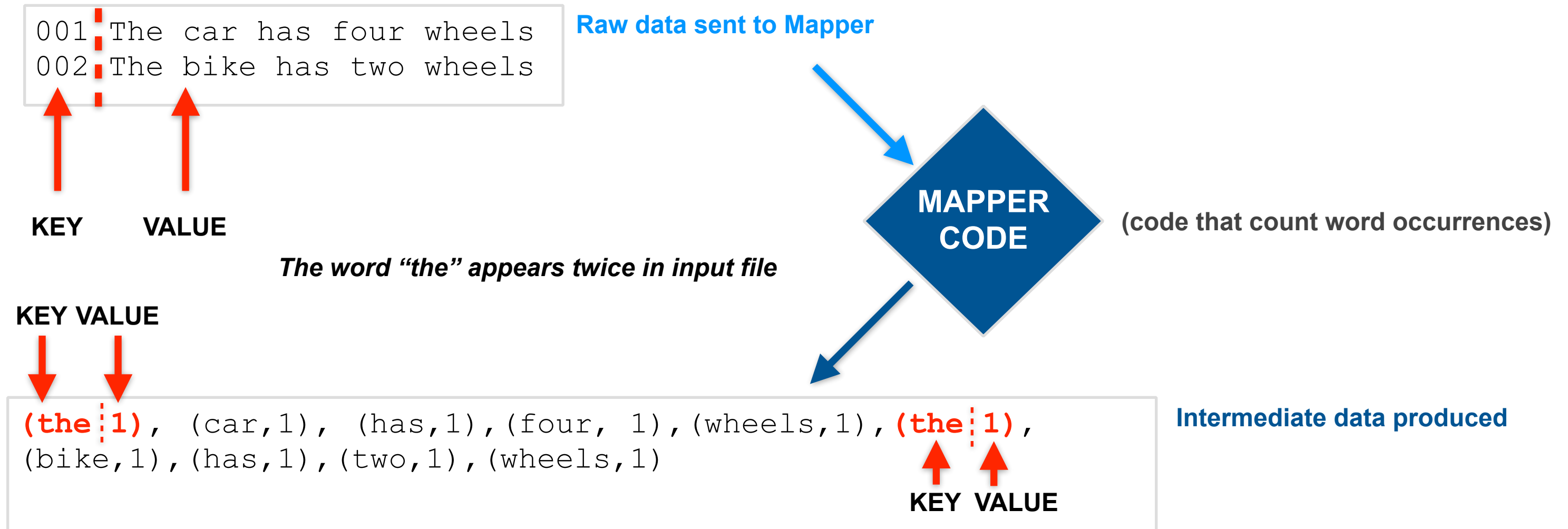
# MAPREDUCE DEMO



# MAPREDUCE DEMO



# MAPREDUCE DEMO



# MAPREDUCE DEMO

```
(the, 1), (car, 1), (has, 1), (four, 1), (wheels, 1), (the, 1),  
(bike, 1), (has, 1), (two, 1), (wheels, 1)
```

Intermediate data produced

KEY VALUE

```
(the 2)  
(car, 1)  
(has, 2)  
(four, 1)  
(wheels, 2)  
(bike, 1)  
(two, 1)
```

REDUCER  
CODE

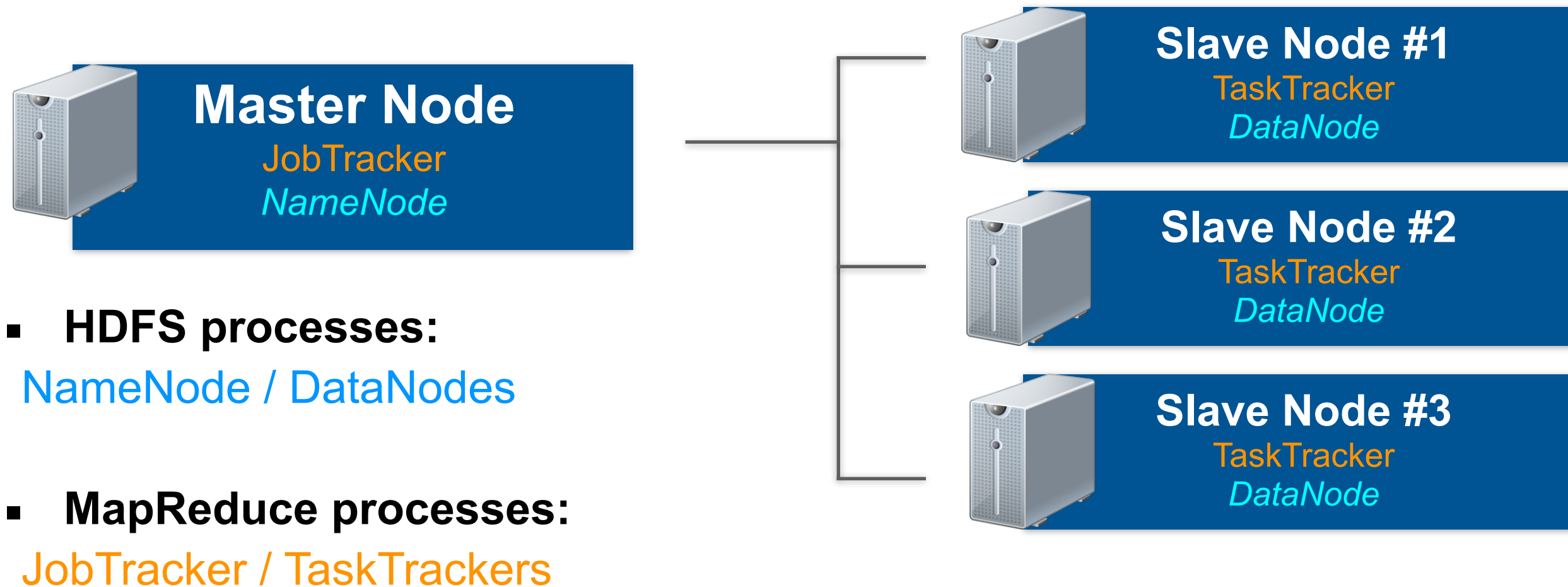
(code that aggregates word occurrences)

Reducer output written to HDFS

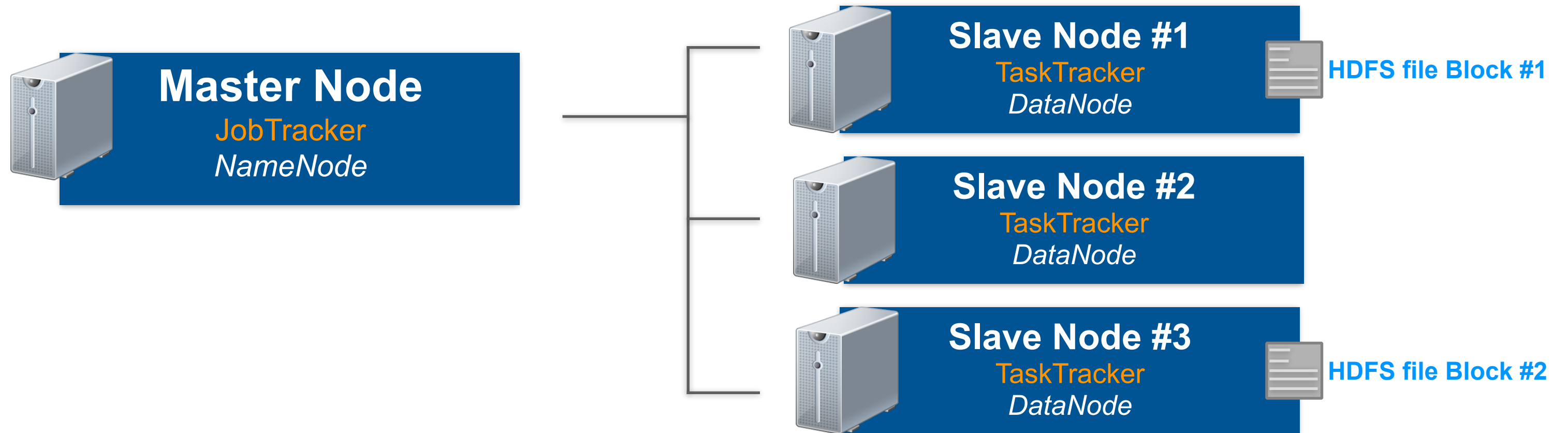
# BEFORE YARN EXISTED

- **MapReduce V1**
  - Single **JobTracker** process- *one per cluster*
    - Manages MapReduce jobs
    - Distributes tasks to JobTrackers
  - Multiple **TaskTracker** process- *one per slave node*
    - Starts and monitors Map/Reduce tasks

# MAPREDUCE V1 ARCHITECTURE

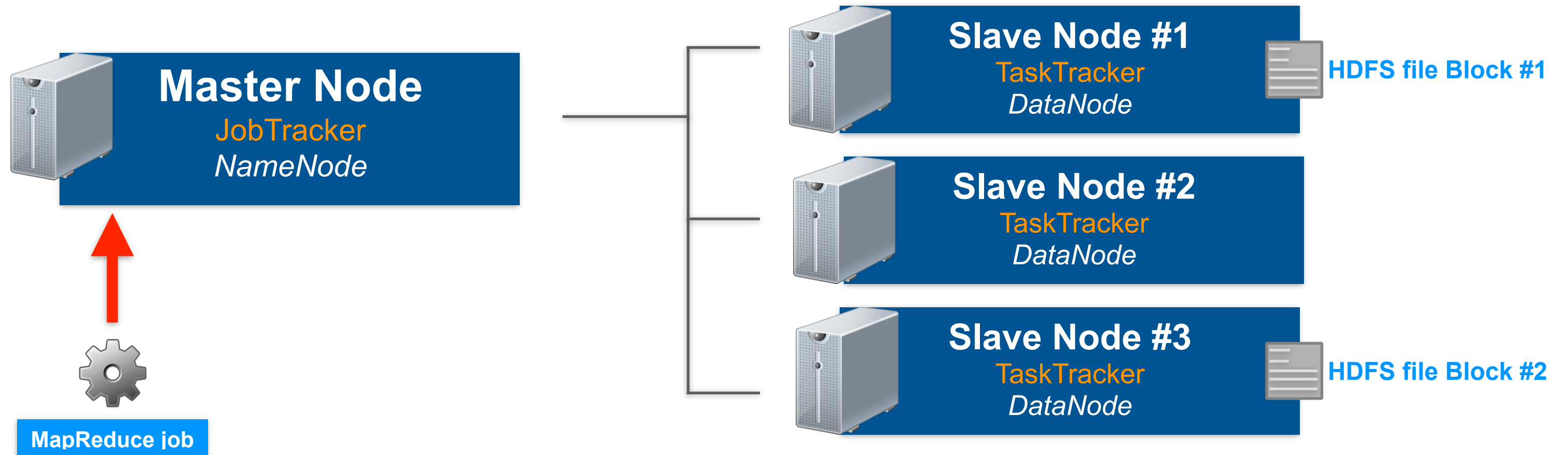


# RUNNING JOBS IN MAP/REDUCE V1

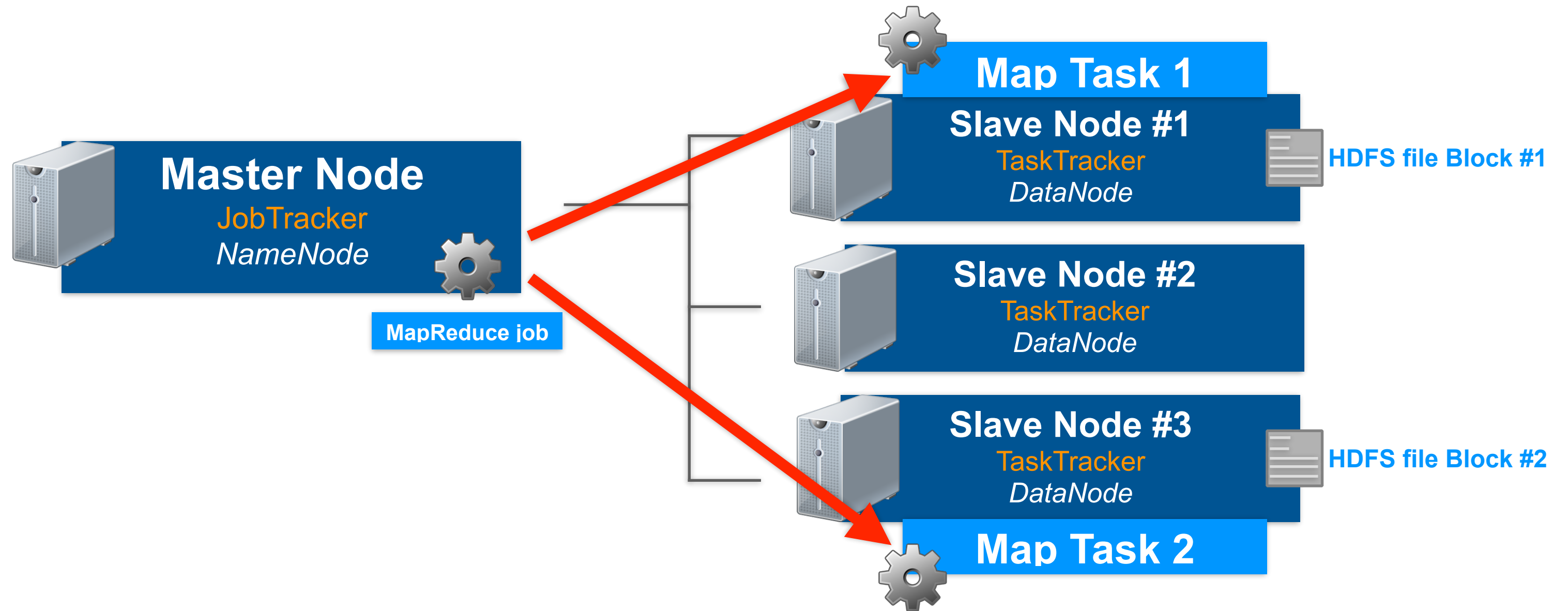




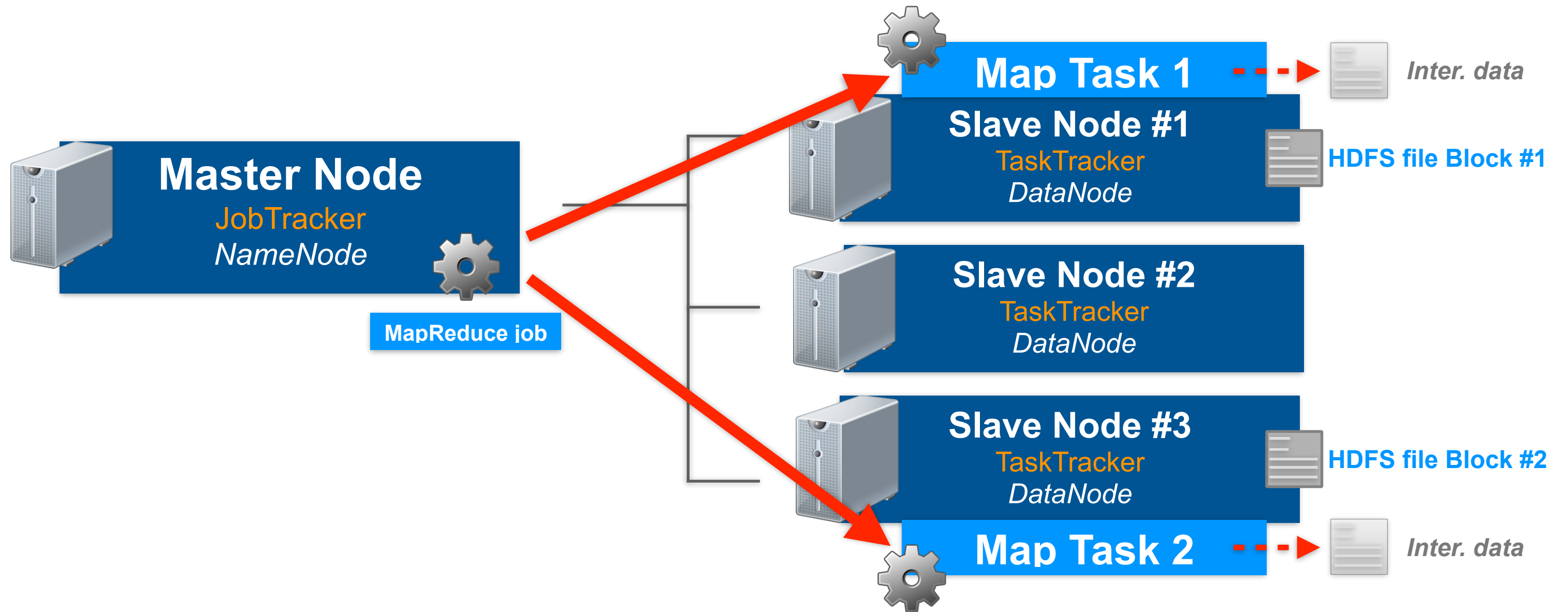
# RUNNING JOBS IN MAPREDUCE V1



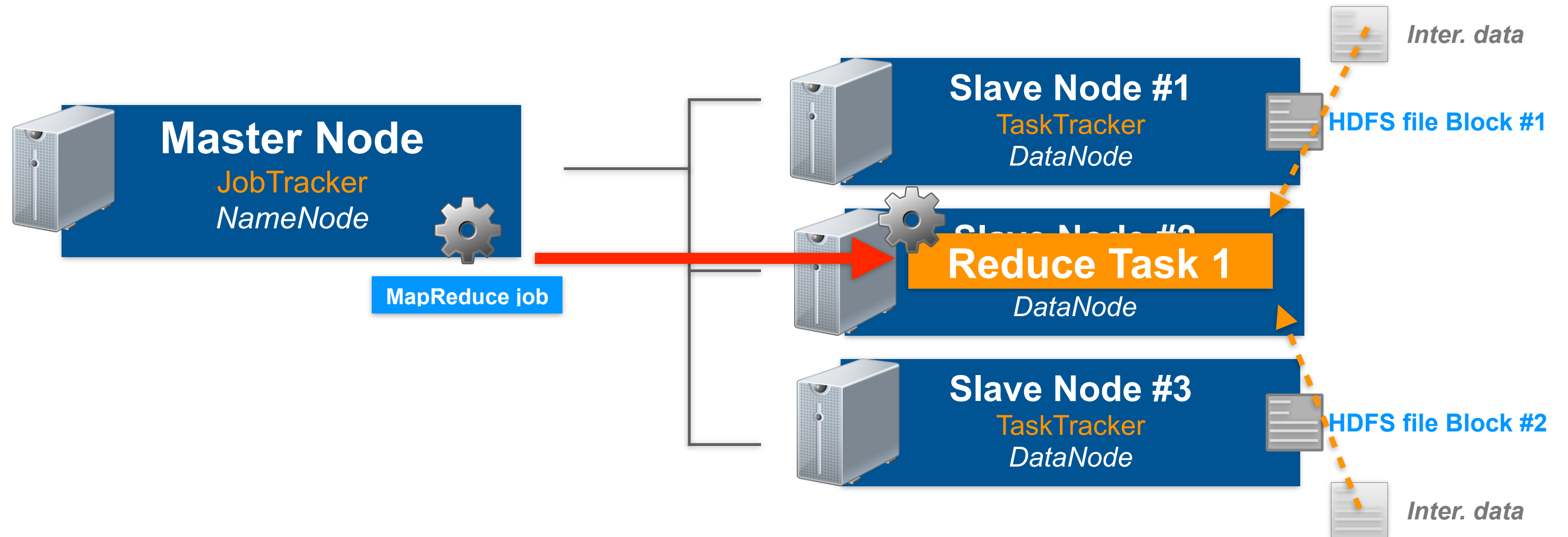
# RUNNING JOBS IN MAPREDUCE V1



# RUNNING JOBS IN MAPREDUCE V1



# RUNNING JOBS IN MAPREDUCE V1



# THE LIMITATIONS OF MAPREDUCE V1

- **Resource allocation:**
  - Slave nodes are configured with a fixed number of “slots” to run Map and Reduce tasks
- **Resource management process limitation:**
  - One JobTracker per cluster
- **Job types:**
  - Limited to MapReduce jobs only

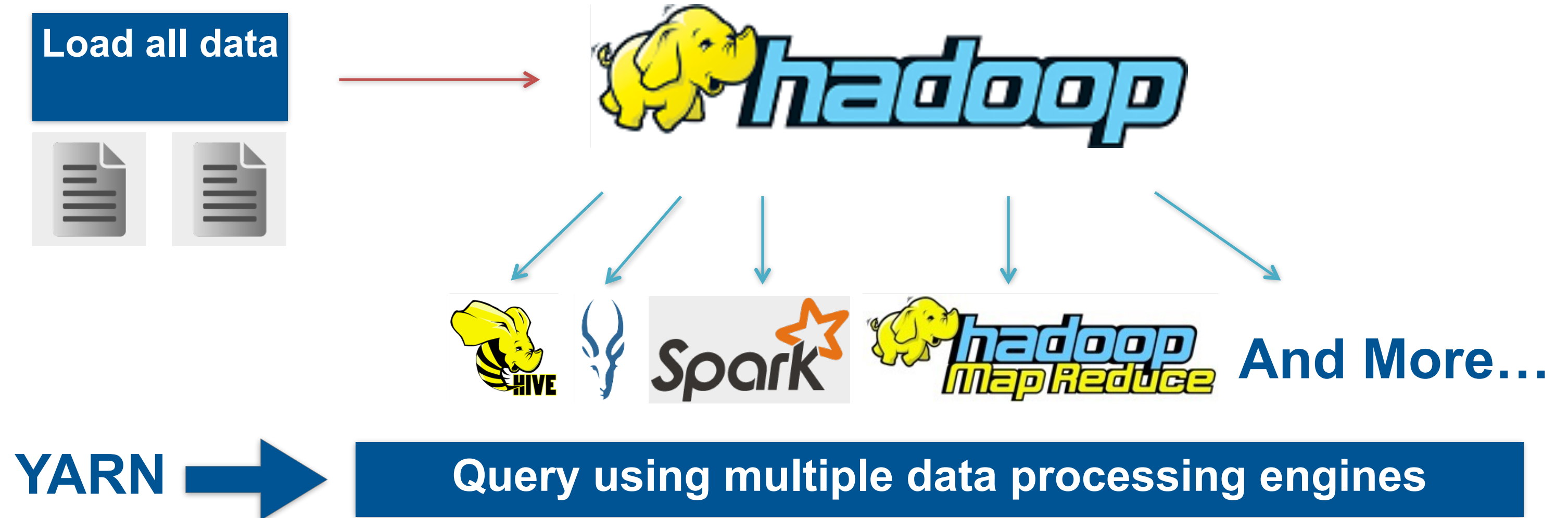
# GETTING TO KNOW YARN

- YARN is new for Hadoop V2 and provides many improvements over MapReduce in Hadoop V1
- YARN support *multiple* distributed processing frameworks:
  - MapReduce V2
  - Impala
  - Spark
  - Etc...



And More...

# YARN ARCHITECTURE

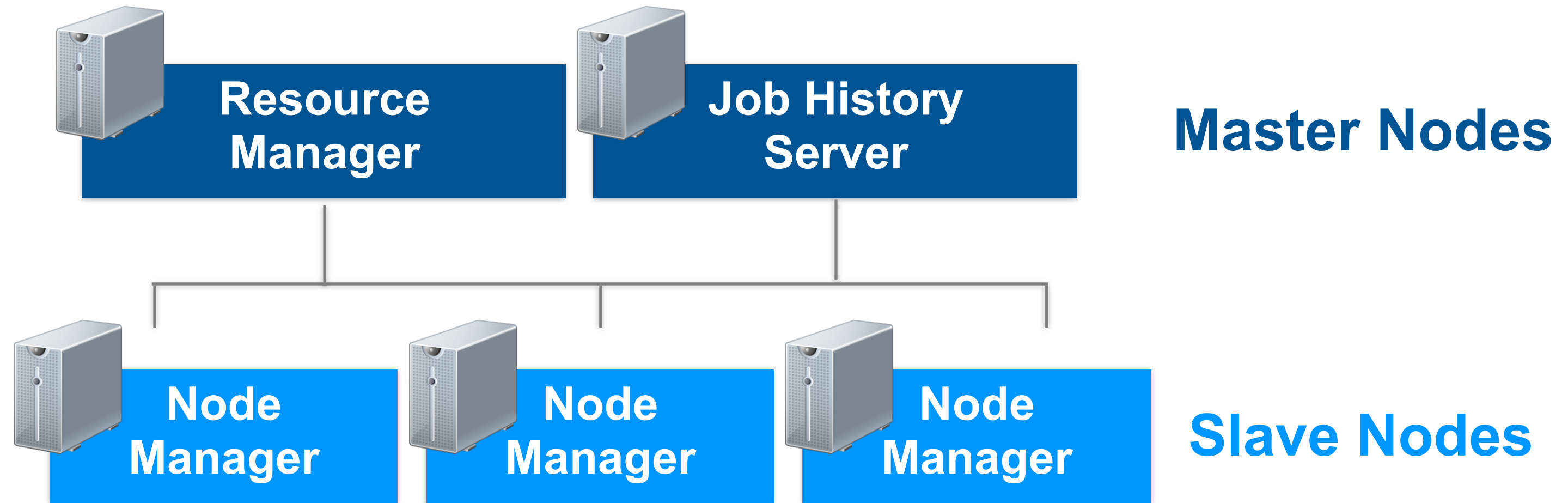


# YARN DAEMONS

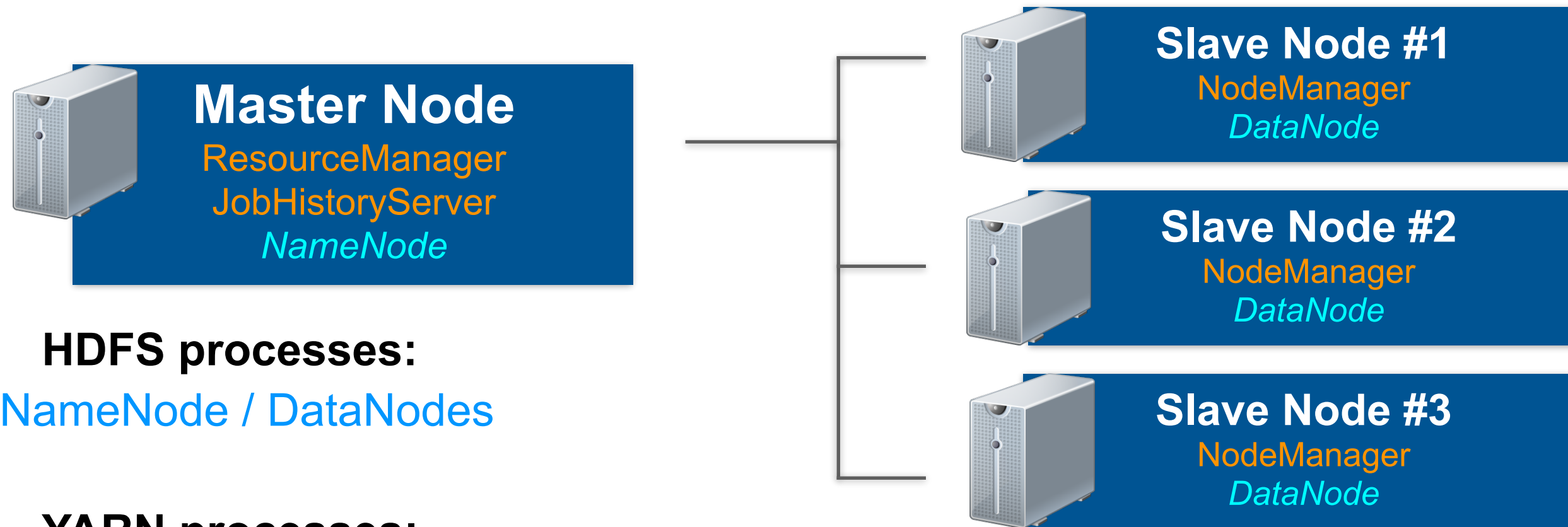
- **Resource Manager** - one per cluster
  - Controls application startup
  - Schedule resources on the slave nodes
- **Node Manager** - one per slave node
  - Starts all processes for a running application
  - Manages resources on the slave nodes
- **Job History Server** - one per cluster
  - Archival of job log files



# THE YARN ARCHITECTURE



# THE YARN ARCHITECTURE



- **HDFS processes:**

NameNode / DataNodes

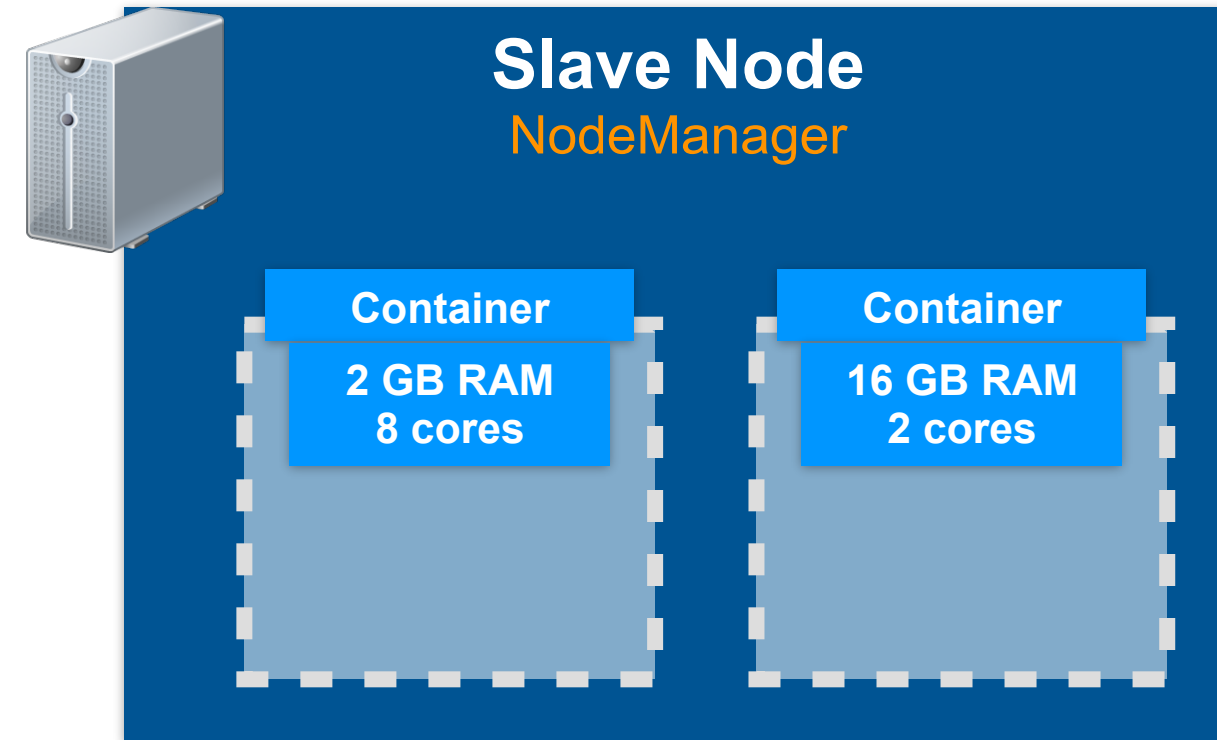
- **YARN processes:**

ResourceManager & JobHistoryServer / NodeManagers

# THE YARN ARCHITECTURE

- **Containers**

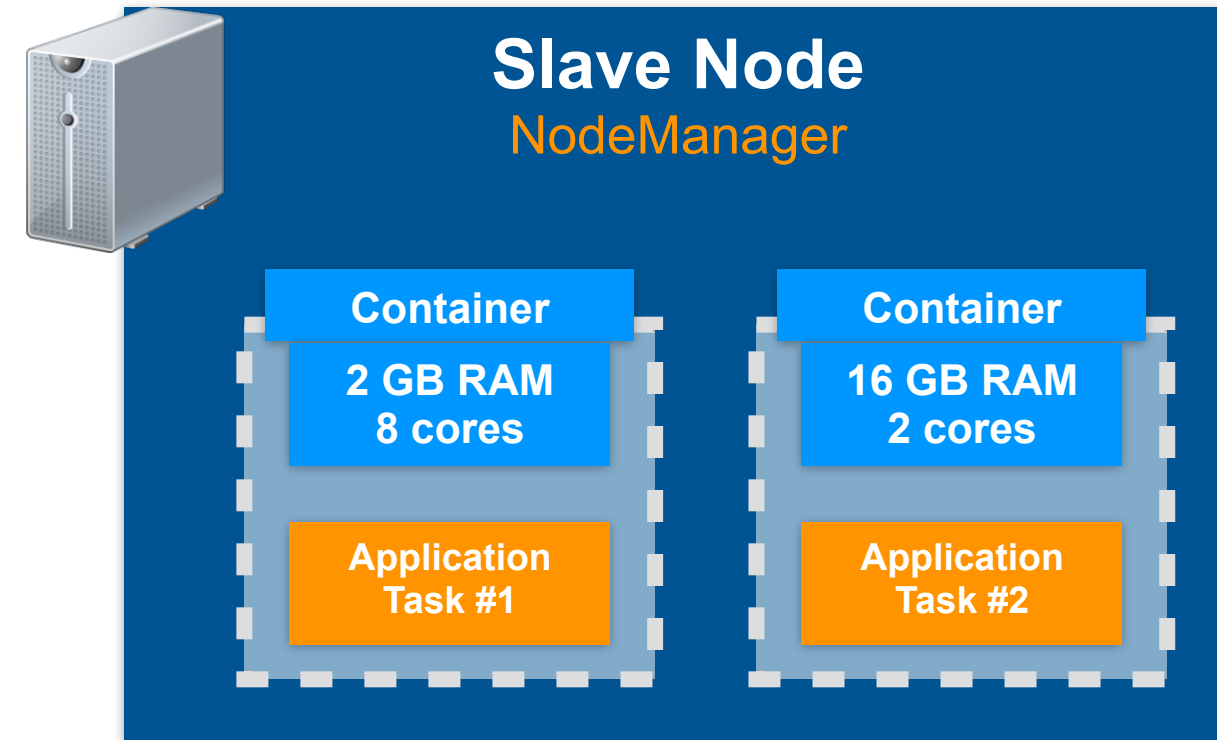
- Allocation of Containers is done by the ResourceManager process
- Containers allocate CPU and memory on a slave node
- Containers run an application task



# THE YARN ARCHITECTURE

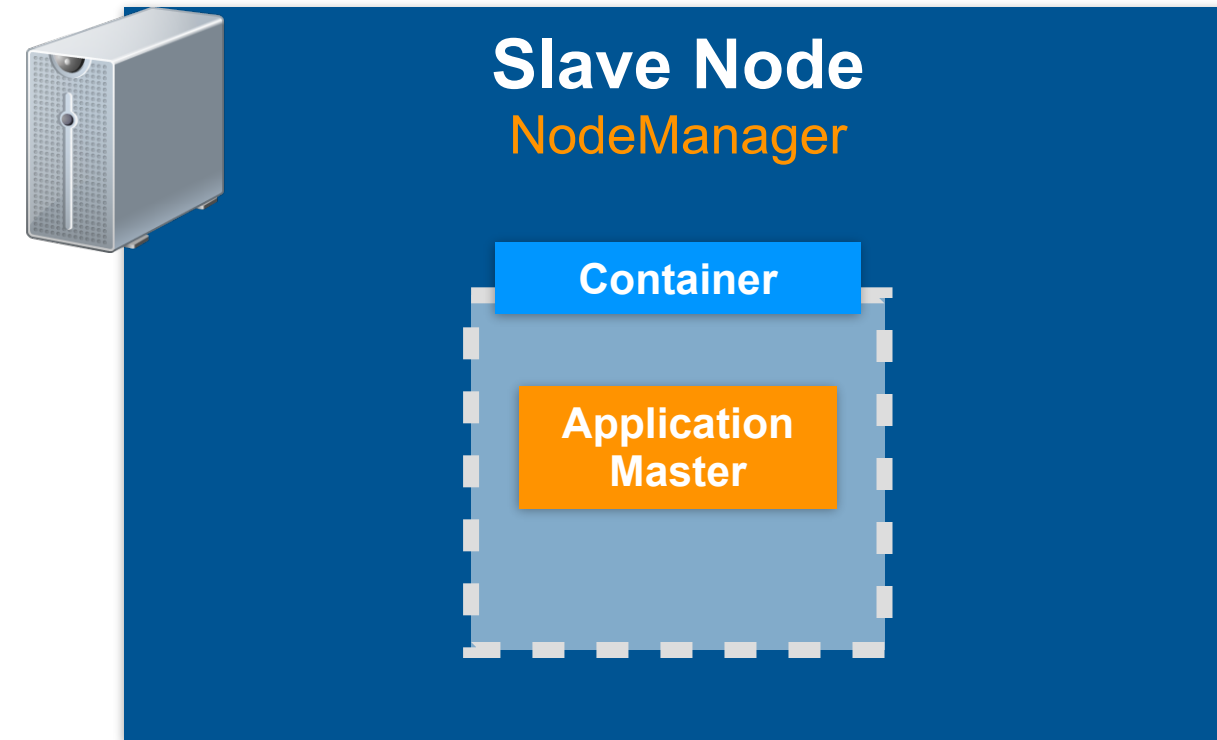
- **Containers**

- Allocation of Containers is done by the ResourceManager process
- Containers allocate CPU and memory on a slave node
- Containers run an application task



# THE YARN ARCHITECTURE

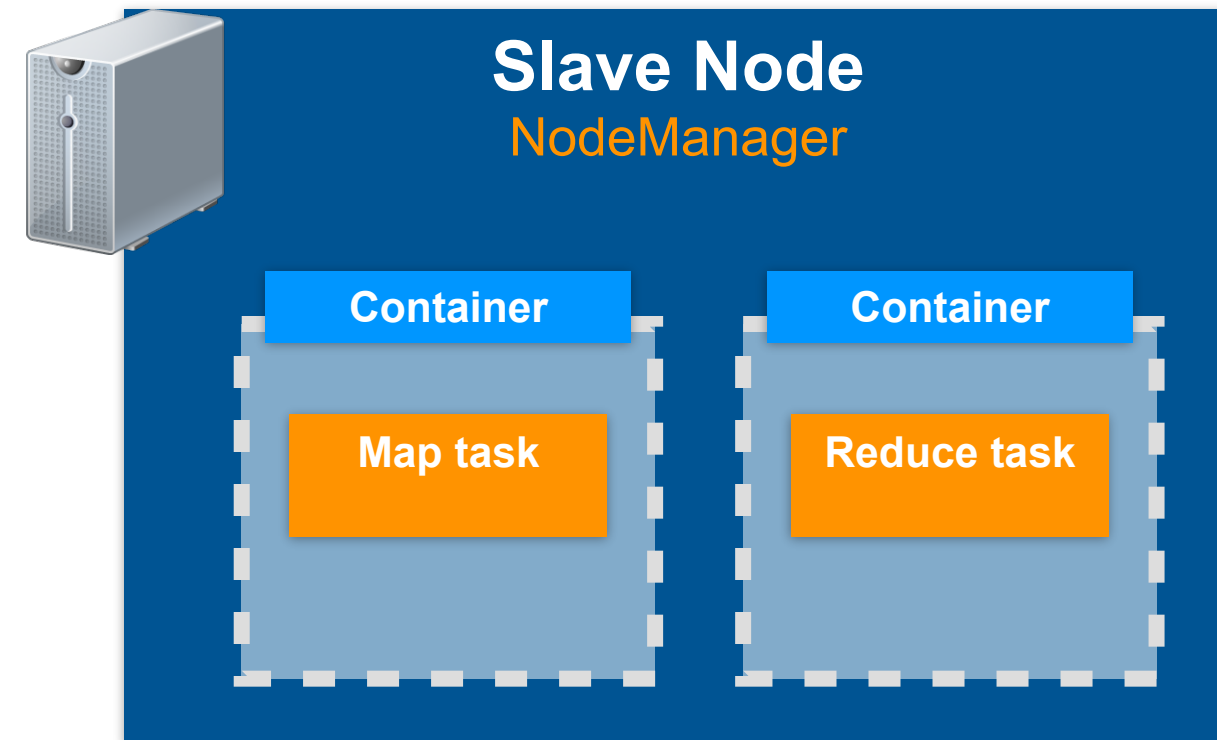
- **Application Masters**
  - Single Application Master per application
  - Runs inside a Container
  - Responsible for requesting additional containers on behalf of the application itself



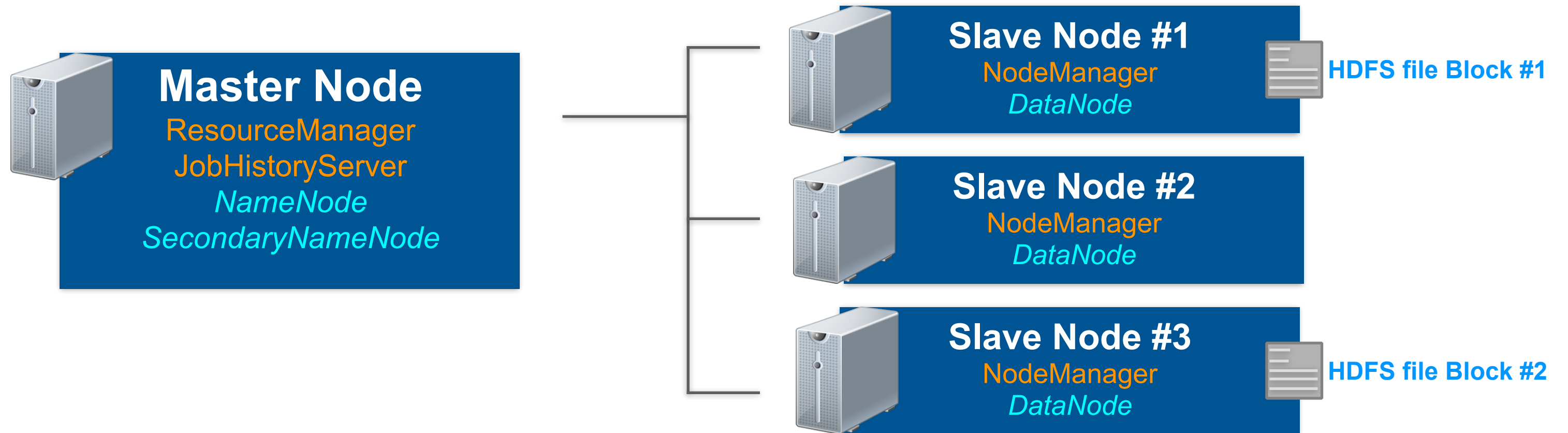
# THE YARN ARCHITECTURE

## ■ Task

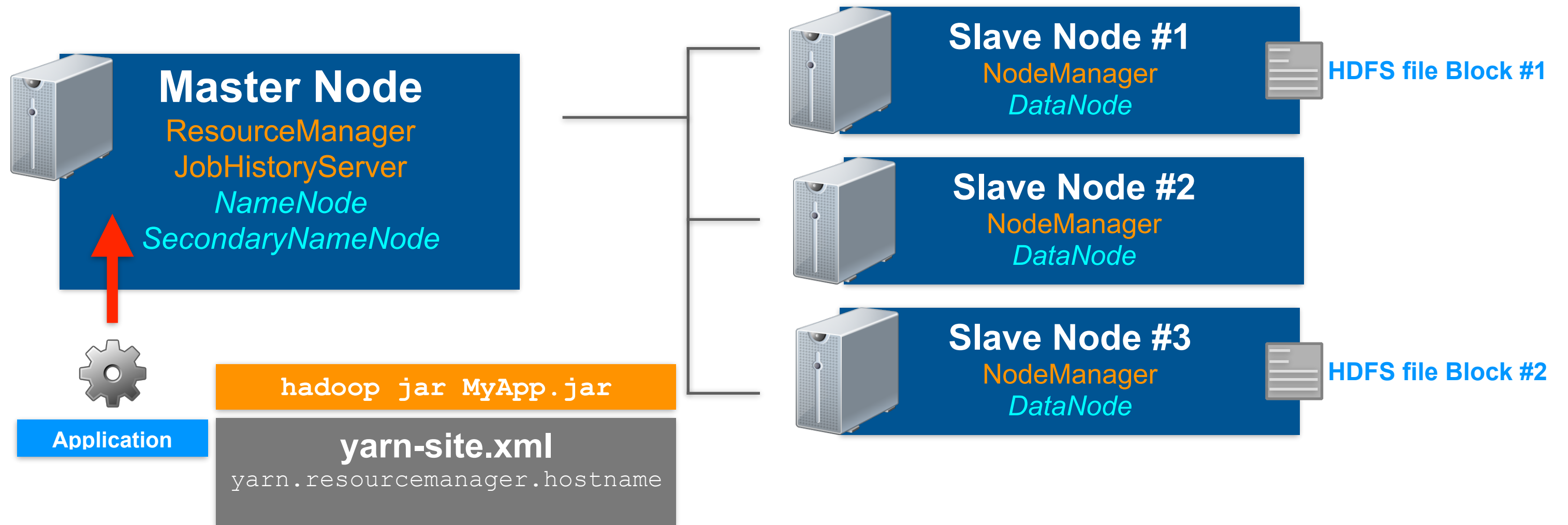
- A single user-submitted job or application is “broken down” to multiple tasks
- Each task runs in a container in Hadoop slave node



# RUNNING JOBS IN YARN

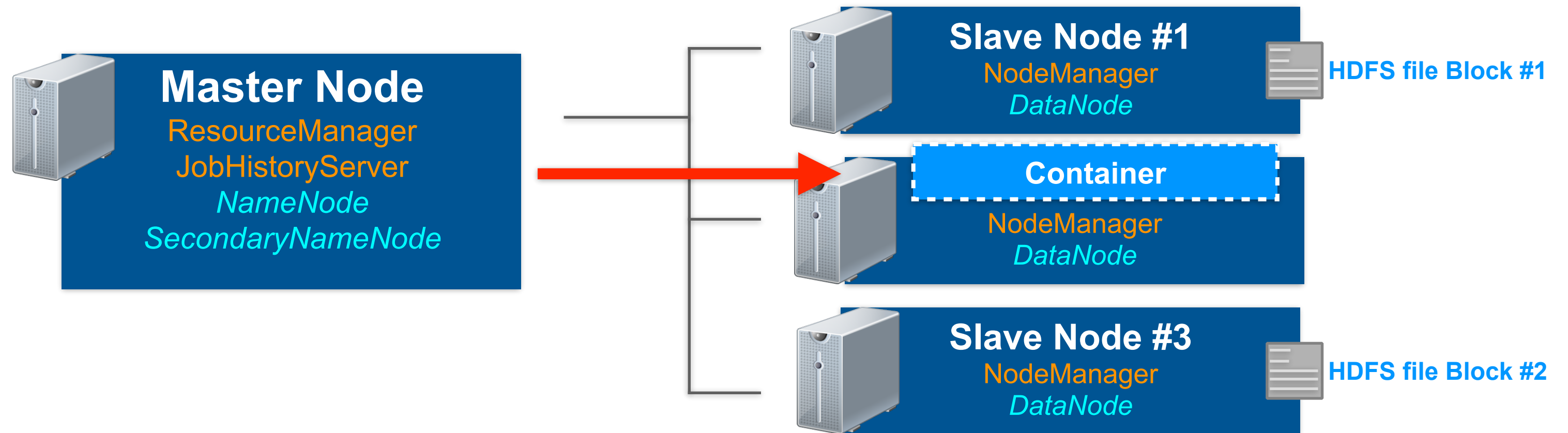


# RUNNING JOBS IN YARN

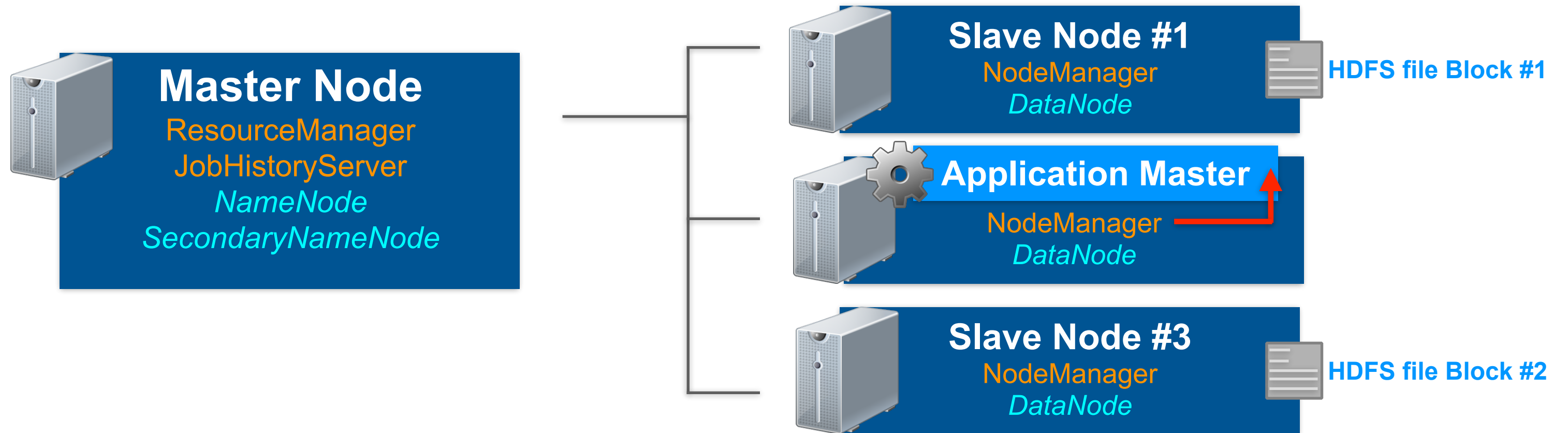




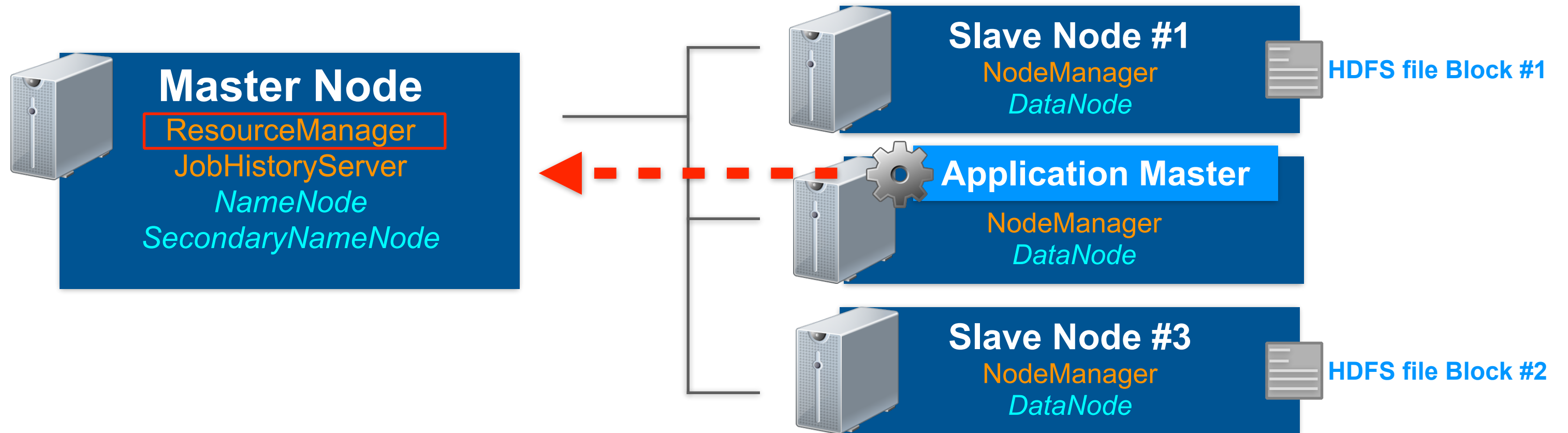
# RUNNING JOBS IN YARN



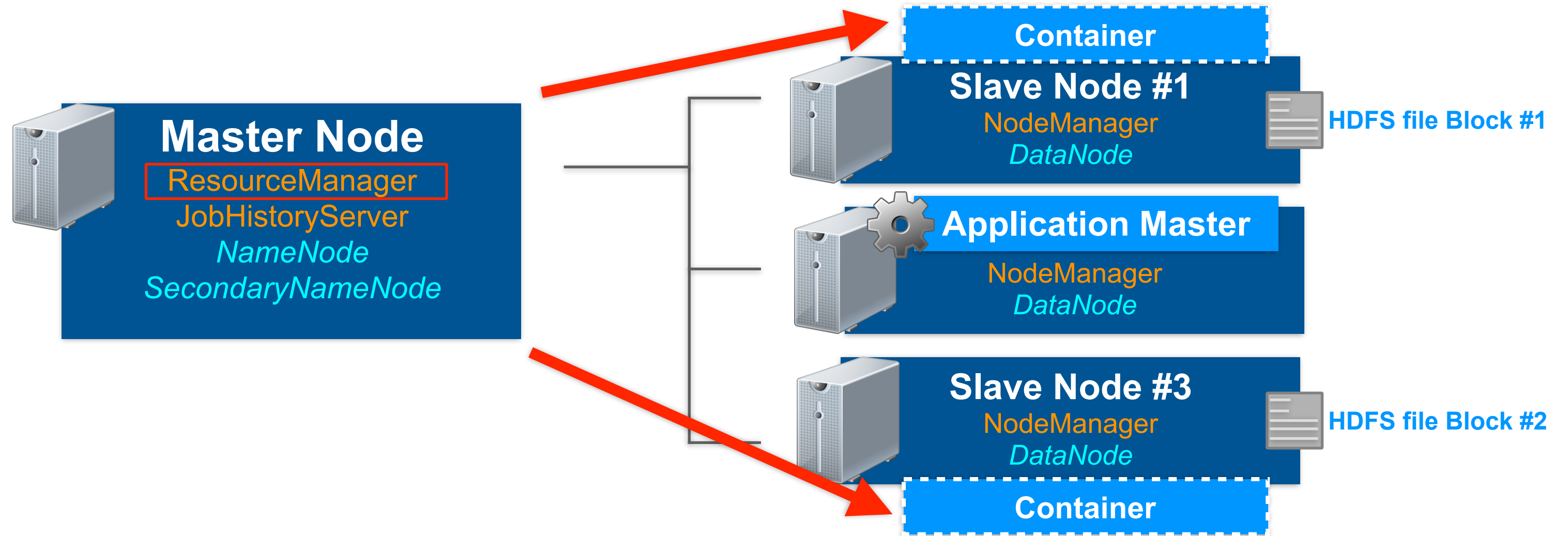
# RUNNING JOBS IN YARN



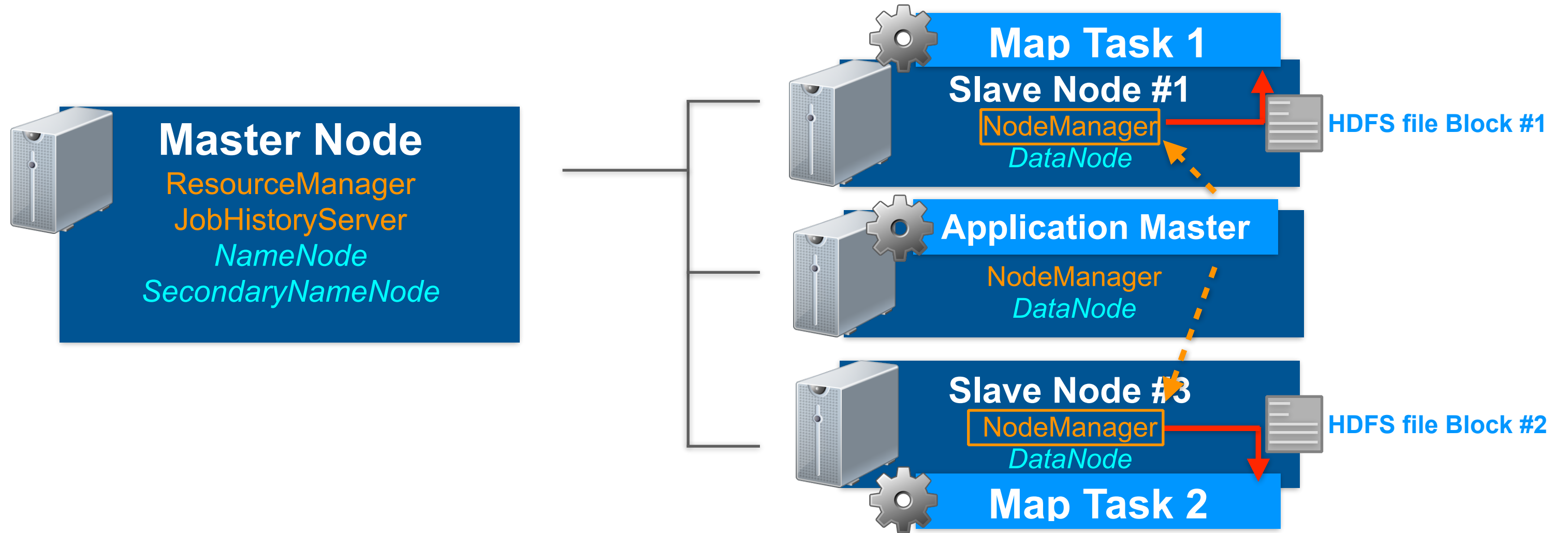
# RUNNING JOBS IN YARN



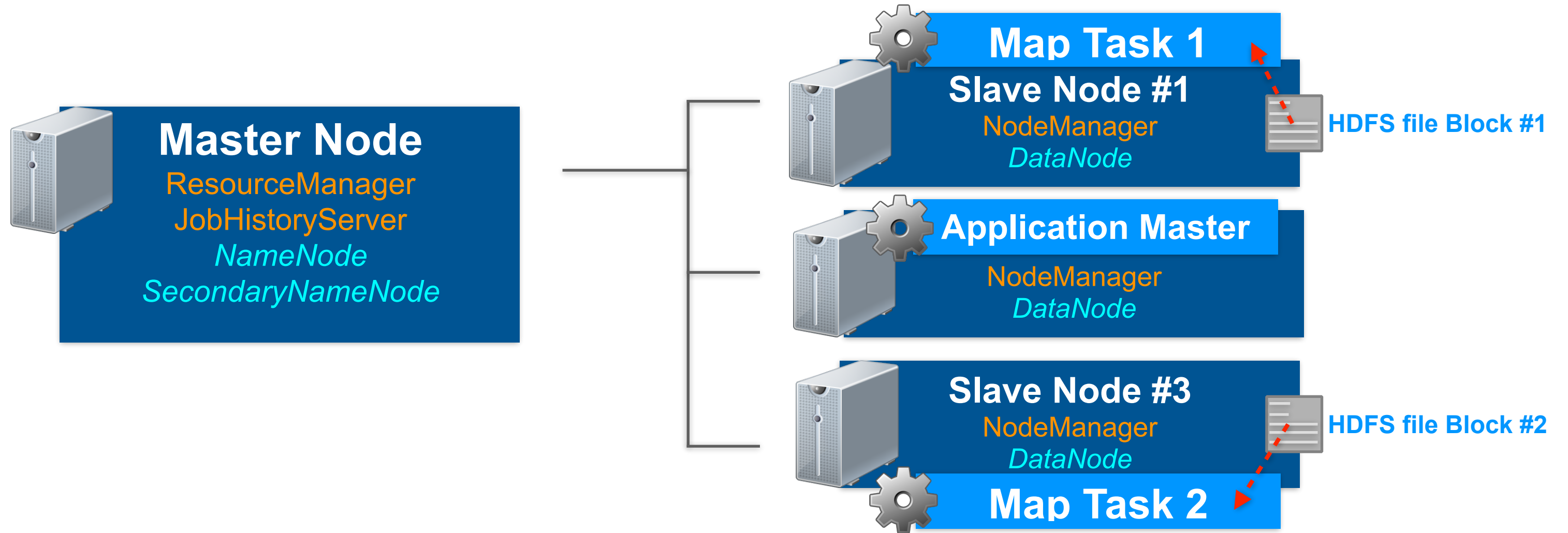
# RUNNING JOBS IN YARN



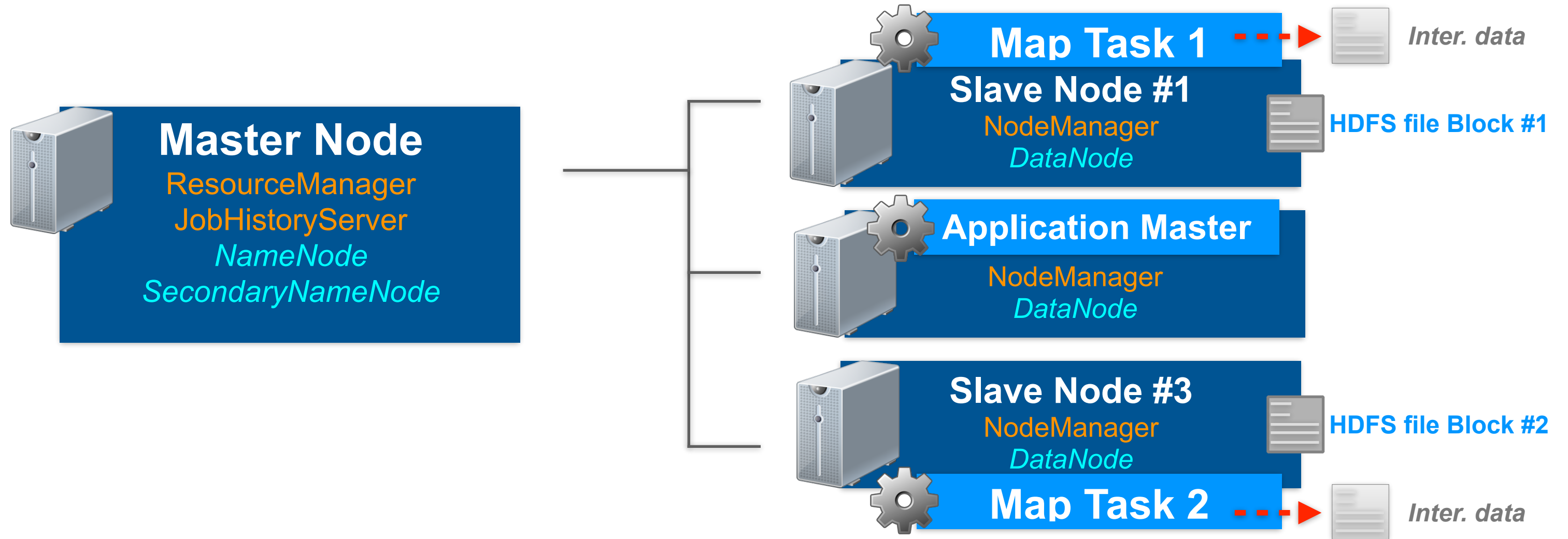
# RUNNING JOBS IN YARN



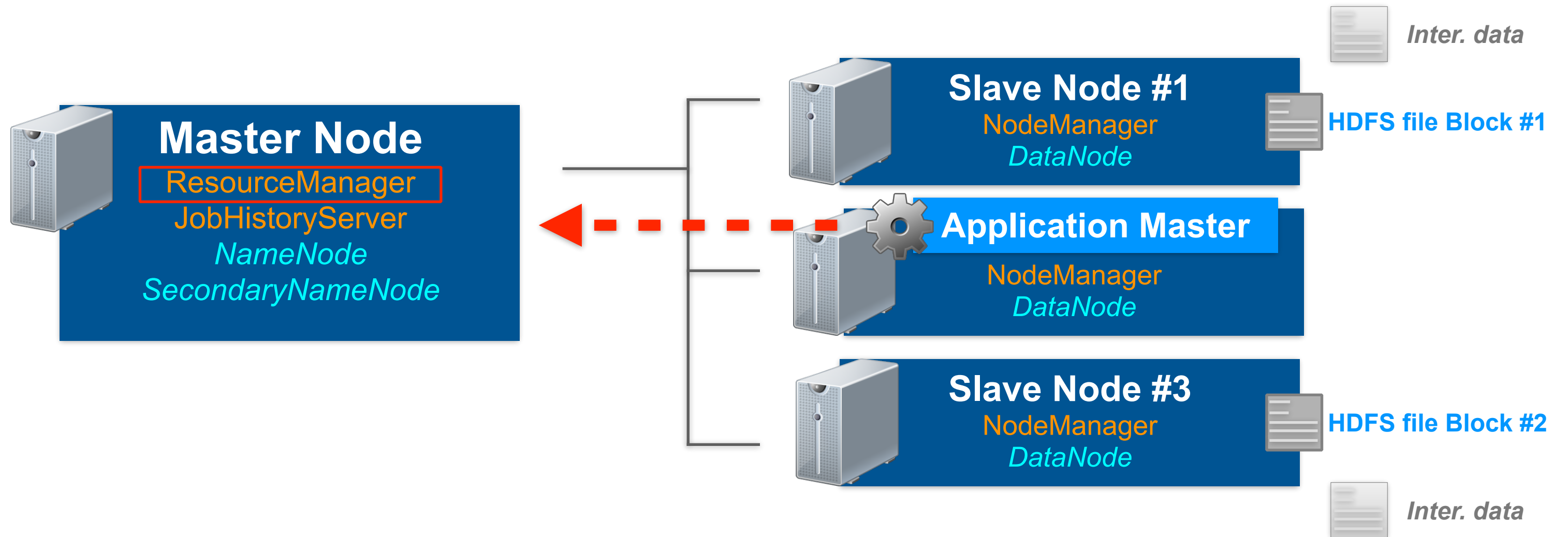
# RUNNING JOBS IN YARN



# RUNNING JOBS IN YARN

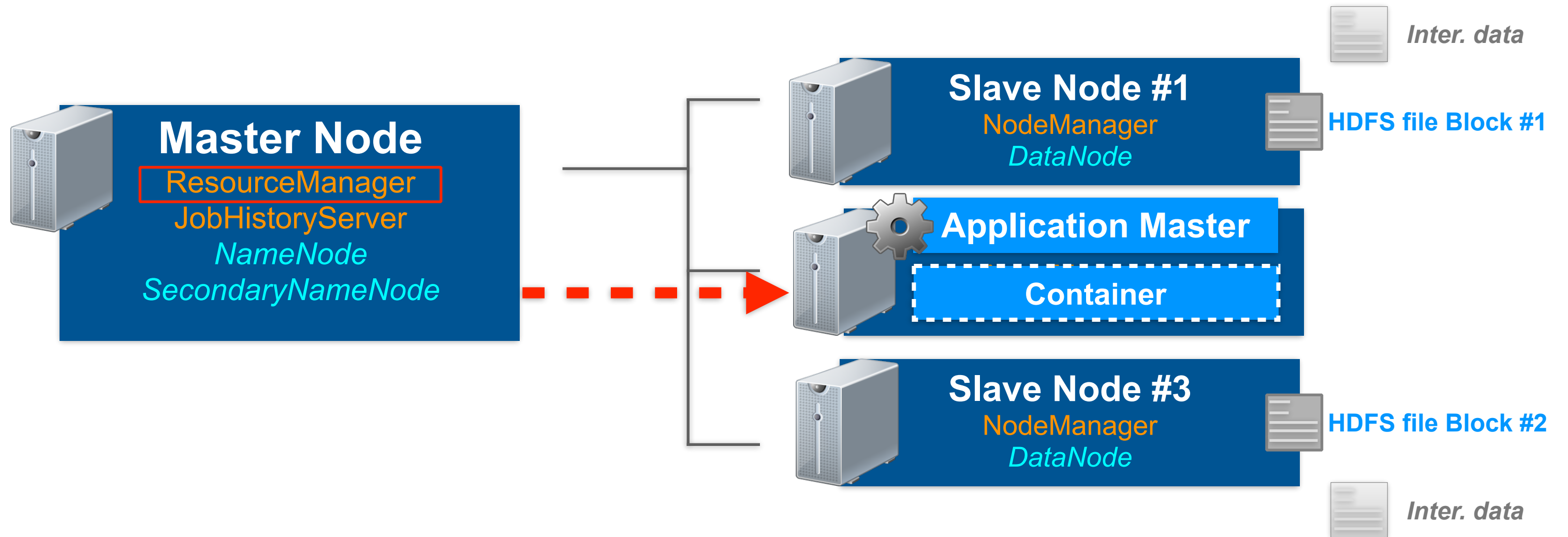


# RUNNING JOBS IN YARN

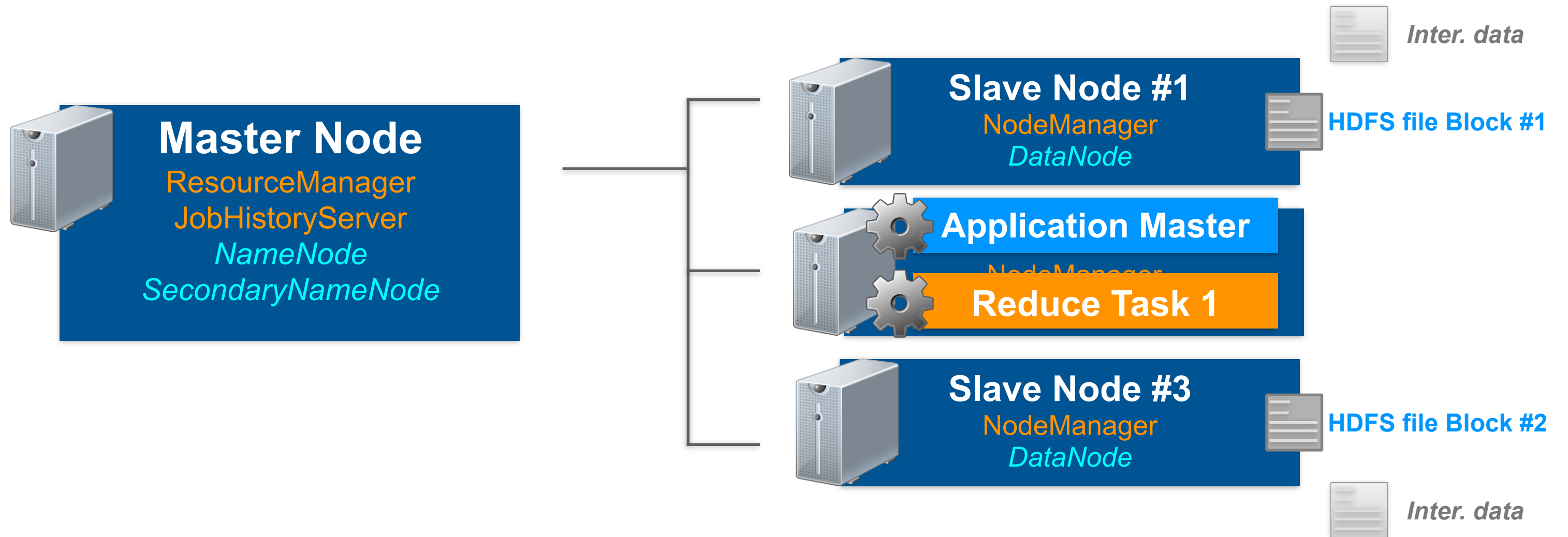




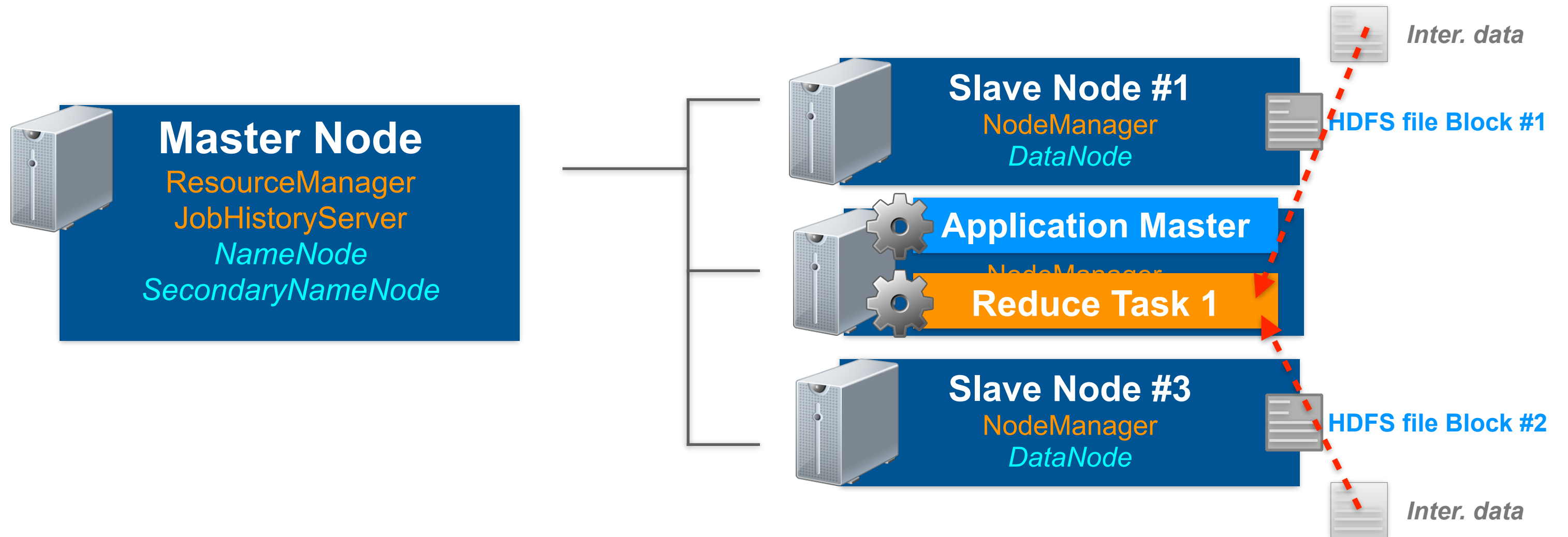
# RUNNING JOBS IN YARN



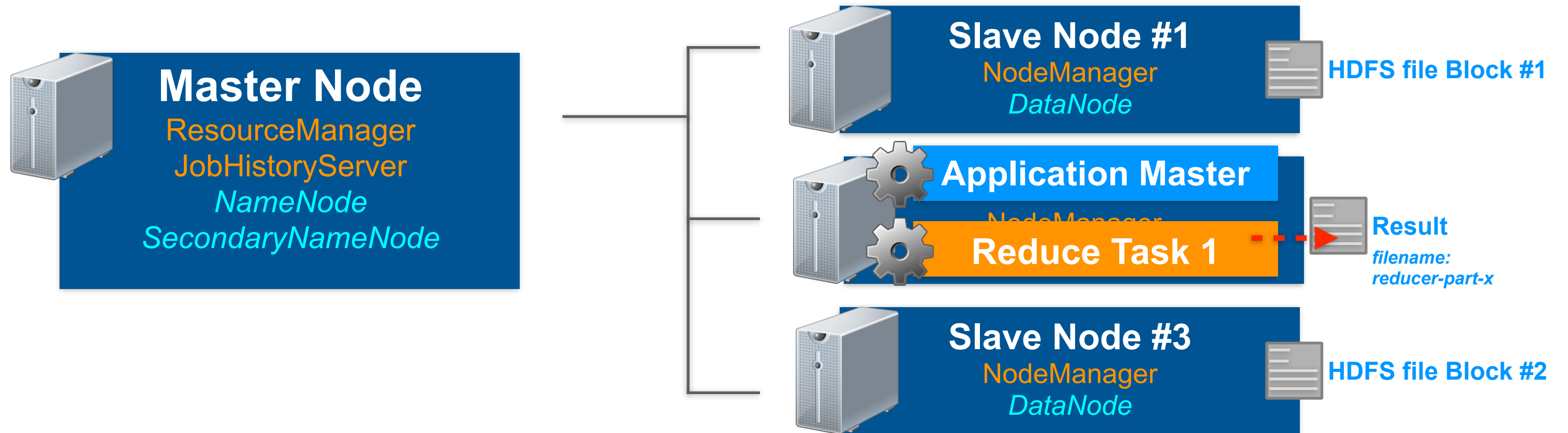
# RUNNING JOBS IN YARN



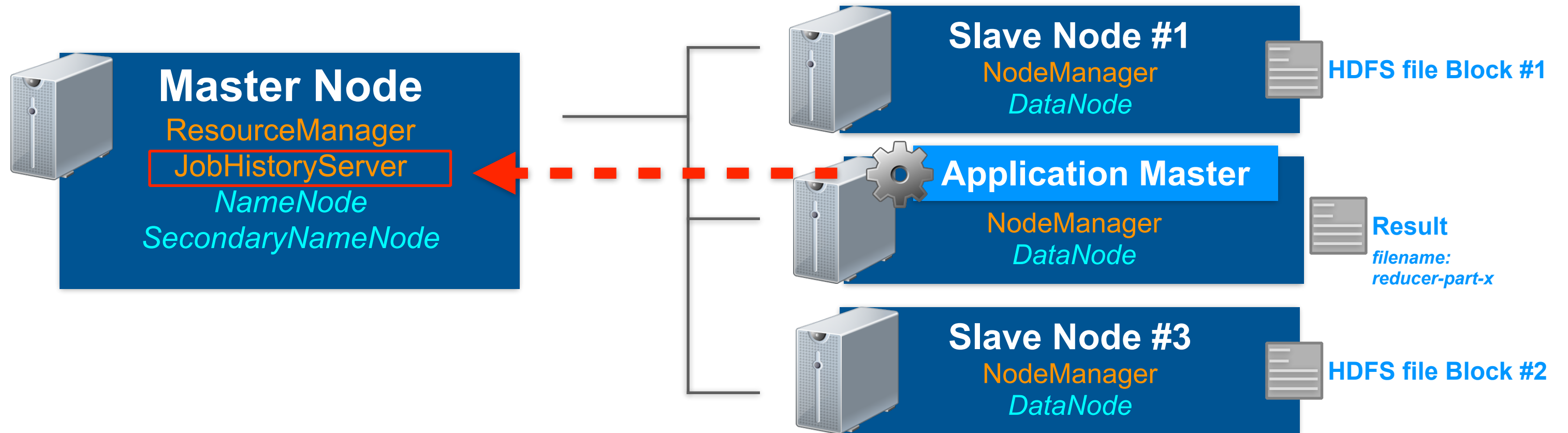
# RUNNING JOBS IN YARN



# RUNNING JOBS IN YARN



# RUNNING JOBS IN YARN



# YARN DAEMONS



- **Resource Manager** - one per cluster
  - Manages slave nodes and tracks heartbeats from NodeManagers
  - Running a resource scheduler which determines how resources are allocated
  - Responsible for the ApplicationMaster: allocates an Application Master container
  - Managers containers: both accepts request from Application Masters to startup new containers and releases containers when they finished executing tasks

# YARN DAEMONS



Node Manager

- **Node Manager** - one for each slave node
  - Communicates with the Resource Manager, registers with the ResourceManager to provide details on available node resources
  - Starts ApplicationMasters on request from ResourceManager
  - Starts job tasks on request from ApplicationMaster
  - Manages tasks in Containers
  - Monitors resource usage and “kills” tasks over-consuming resources
  - Stores applications logs on HDFS

# YARN CONFIGURATION

- Configuration files for YARN & MapReduce in Hadoop

`yarn-site.xml`

`mapred-site.xml`



# YARN CONFIGURATION

- Configuration file for YARN: **yarn-site.xml**

`yarn.resourcemanager.hostname`

Server name for the YARN ResourceManager.  
Used by the ResourceManager, the NodeManagers and clients

```
<name>yarn.resourcemanager.hostname</name>  
  <value>HADOOPM1:8032</value>
```

# YARN CONFIGURATION

- Configuration file for YARN: **yarn-site.xml**

```
yarn.nodemanager.local-dirs
```

The directory in which local resource files are stored such as intermediate output files and the distributed cache

```
<name>yarn.nodemanager.local-dirs</name>  
  <value>/disks/yarn/nm</value>
```

# YARN CONFIGURATION

- Configuration file for YARN: **yarn-site.xml**

```
yarn.nodemanager.aux-services
```

One or more auxiliary services that support application frameworks running under YARN

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

# YARN CONFIGURATION

- Master configuration file for YARN: **yarn-site.xml**

```
yarn.log-aggregation-enable
```

```
yarn.nodemanager.remote-app-log-dir
```

```
yarn.nodemanager.log-dirs
```

Parameter which control YARN log aggregation

We will discuss these in the YARN log chapter of the course...

# YARN CONFIGURATION

- MapReduce configuration file: **mapred-site.xml**

`mapreduce.framework.name`

MapReduce execution framework. Default is **local** should be changed to **yarn**

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

# YARN CONFIGURATION

- MapReduce configuration file: **mapred-site.xml**

```
yarn.app.mapreduce.am.staging-dir
```

The HDFS directory in which users job files (jar files) needed by applications are stored. Should usually be set to “/user”. Used mainly by the clients and Appellation Master

```
<name>yarn.app.mapreduce.am.staging-dir</name>  
    <value>/user</value>
```

# YARN CONFIGURATION

- MapReduce configuration file: **mapred-site.xml**

`mapreduce.jobhistory.address`

The address and port used for cluster internal communication with the JobHistoryServer

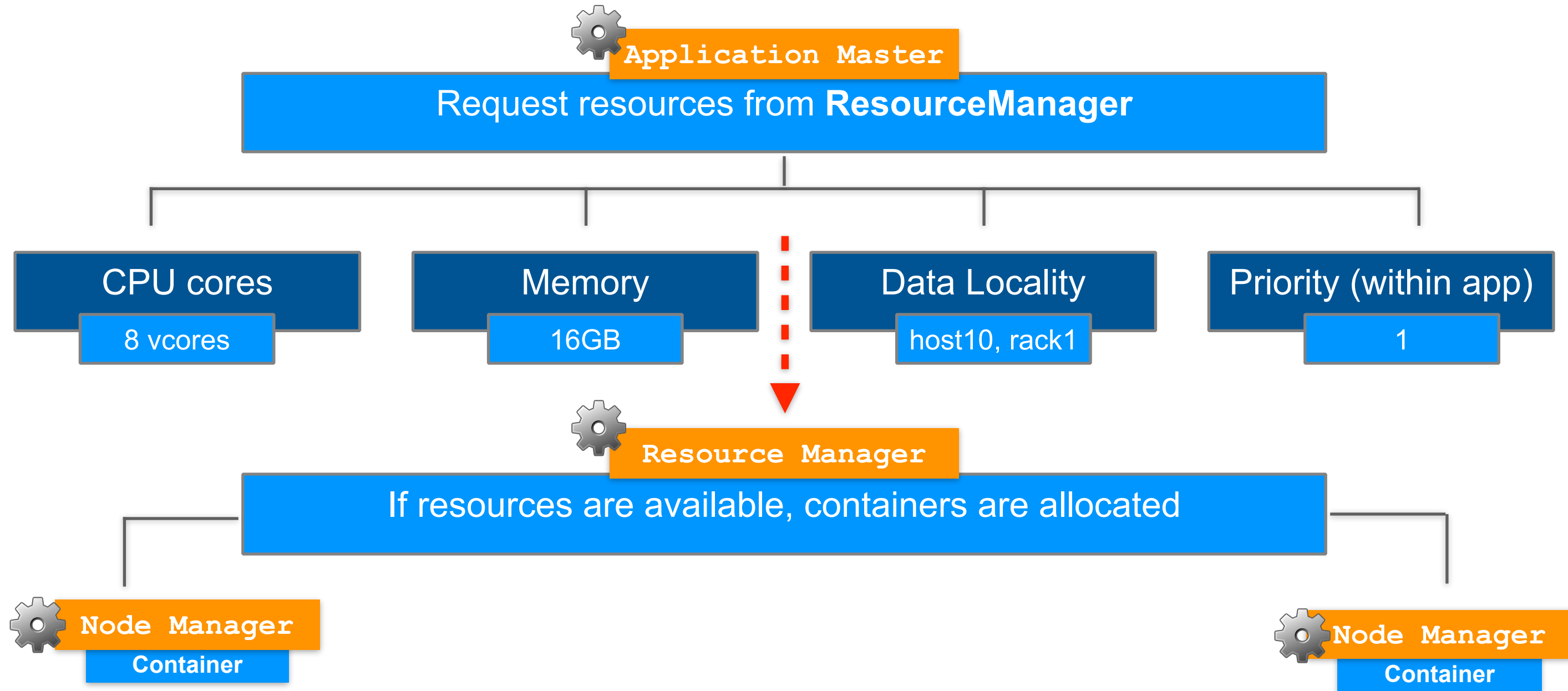
```
<name>mapreduce.jobhistory.address</name>  
  <value>HADOOPM1:10020</value>
```

# YARN RESOURCE ALLOCATION

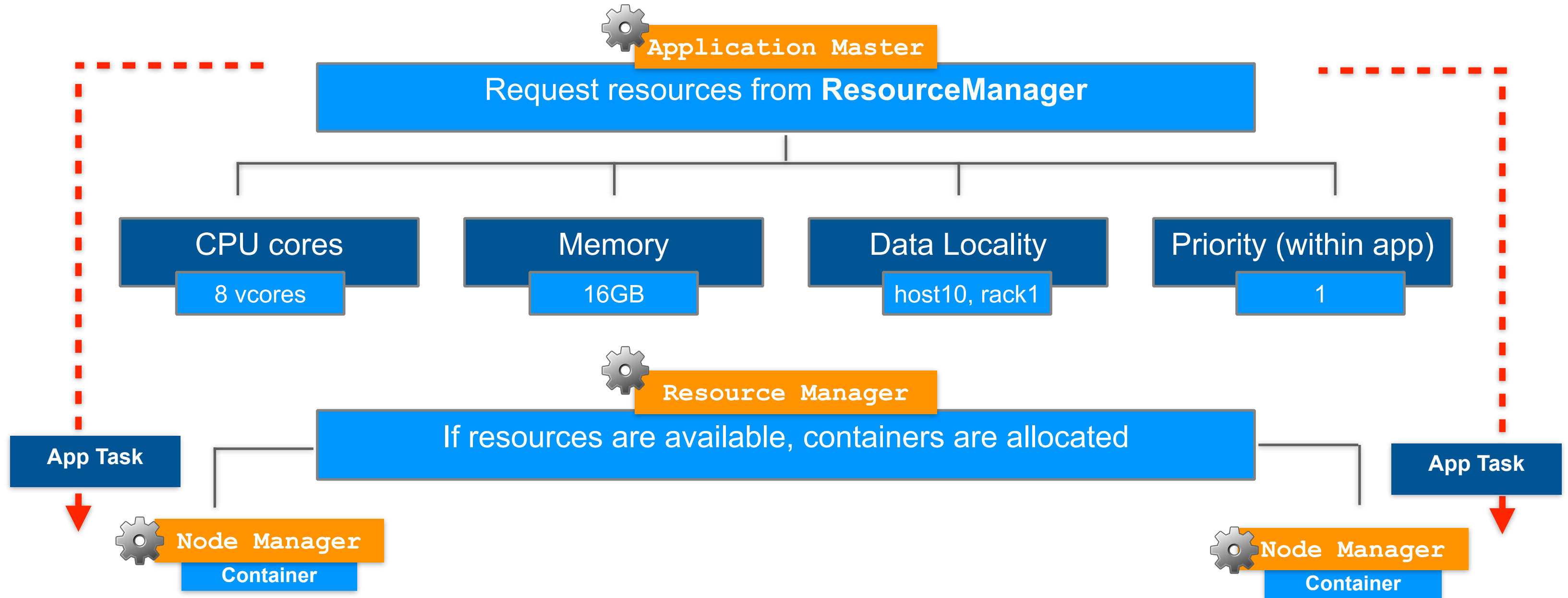
- **Application Master** negotiates with ResourceManager to obtain resources for the job and presents the containers to the NodeManagers
- **Resource Manager** is responsible for granting containers in the YARN-enabled Hadoop cluster
- **Node Manager** is responsible for managing the containers, running MapReduce jobs in the containers and monitors resource usage



# YARN RESOURCE ALLOCATION



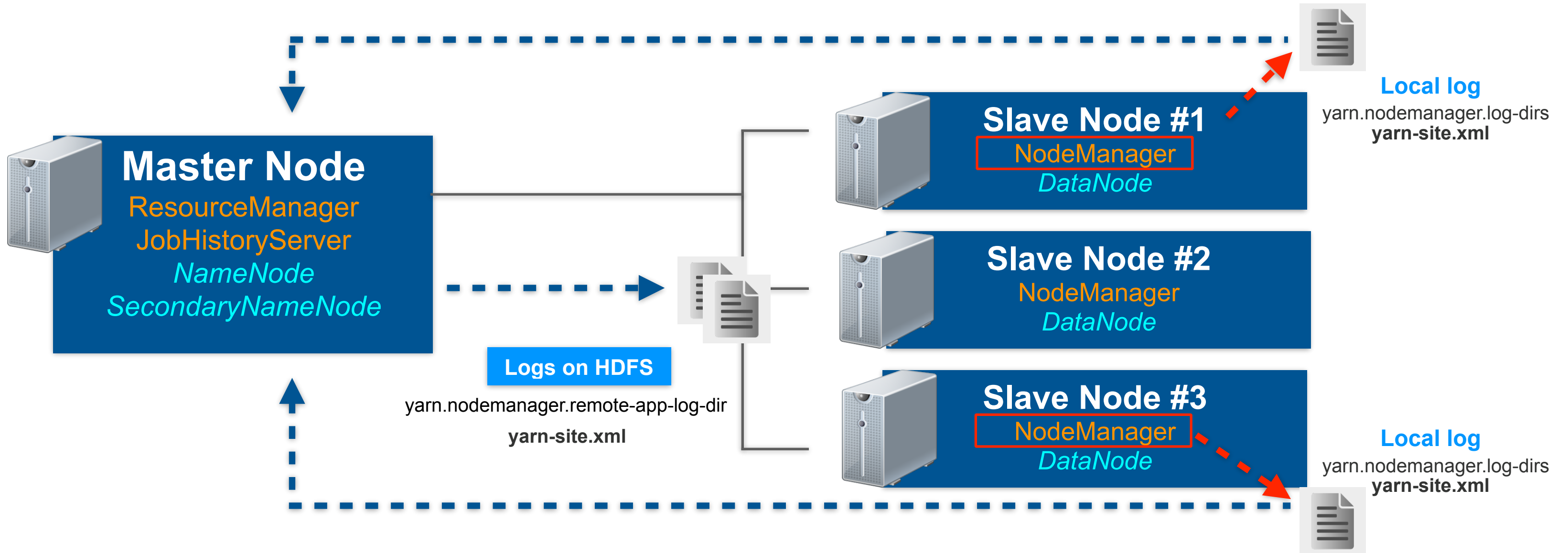
# YARN RESOURCE ALLOCATION



# YARN LOGS

- Debugging jobs running in **distribution** isn't easy.
- Multiple processes running different nodes.
- YARN aggregates local logs from ***NodeManagers*** to a central location - HDFS.
- Logs are ordered by application / job.
- Accessible on HDFS or using a WEB interface.
- **Disabled** by default, usually enabled after installation

# YARN LOGS



# YARN LOGS

- Parameters to enable log aggregation:

Parameter	Configuration file	Comments
yarn.log-aggregation-enable	yarn-site.xml	Enable for log aggregation
yarn.nodemanager.remote-app-log-dir	yarn-site.xml	The HDFS location to which NodeManagers should aggregate logs
yarn.nodemanager.log-dirs	yarn-site.xml	Local directory where NodeManagers store local logs
yarn.log-aggregation.retain-seconds	yarn-site.xml	How long to wait before deleting aggregated log files. Disabled with -1

# YARN FAULT TOLERANCE

- **If a task running in a container fails**
  - Application Master will retry the task again.
  - If too many application tasks have failed the entire application is considered as failed.
  - Job Recovery can be configured so that Application Master can retry all tasks or only tasks which have not completed.
- **If an Application Master fails**, the Resource Manager will restart the entire application - up to two times by default.

# YARN FAULT TOLERANCE

- **If a NodeManager fails** - ResourceManager will remove it from the list of “active nodes”.
- Tasks on that node will be treated as failed by Application Master.
- If the node running the Application Master fails, the Resource Manager will treat it as a failed application.

# YARN FAULT TOLERANCE

- **If the Resource Manager fails**
  - No applications or tasks can be executed until Resource Manager is back online.
  - We can configure high availability for the resource manager to handle such failures.
- Tasks on that node will be treated as failed by Application Master.
- If the node running the Application Master fails, the Resource