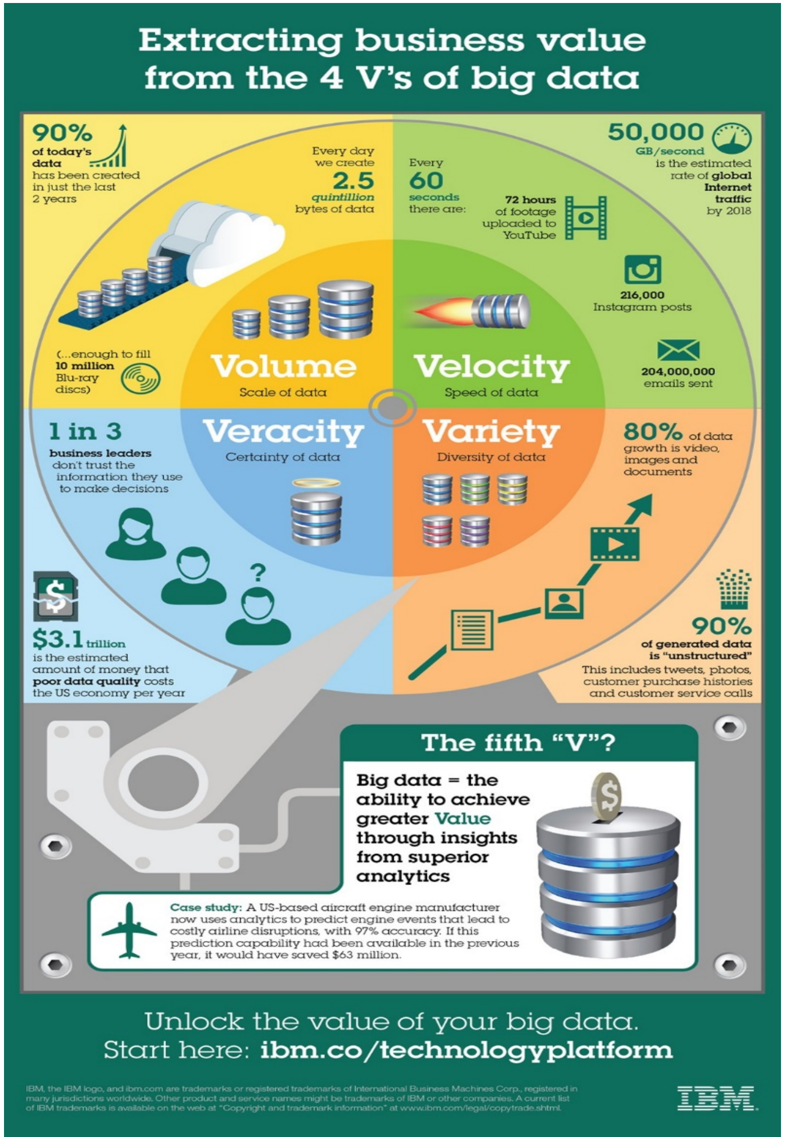
BIG DATA with HADOOP

What is big data?

- “Every day, we create 2.5 quintillion bytes of data — so much that 90% of the data in the world today has been created in the last two years alone. This data comes from everywhere: sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records, and cell phone GPS signals to name a few.

This data is “big data.”

The 5V's revisited



Extracting business value from the 4 V's of big data

90% of today's data has been created in just the last 2 years

Every day we create **2.5** *quintillion* bytes of data

(...enough to fill **10 million** Blu-ray discs)

Volume
Scale of data

Every **60** seconds there are:

72 hours of footage uploaded to YouTube

216,000 Instagram posts

204,000,000 emails sent

50,000 GB/second is the estimated rate of **global Internet traffic** by 2018

Velocity
Speed of data

Veracity
Certainty of data

1 in 3 business leaders don't trust the information they use to make decisions

$3.1 trillion is the estimated amount of money that **poor data quality** costs the US economy per year

Variety
Diversity of data

80% of data growth is video, images and documents

90% of generated data is "unstructured"

This includes tweets, photos, customer purchase histories and customer service calls

The fifth "V"?

Big data = the ability to achieve greater Value through insights from superior analytics

Case study: A US-based aircraft engine manufacturer now uses analytics to predict engine events that lead to costly airline disruptions, with 97% accuracy. If this prediction capability had been available in the previous year, it would have saved $63 million.

Unlock the value of your big data.
Start here: **ibm.co/technologyplatform**

Big Data Database Types

- Key Value Stores
  - Riak(Open Source based on DynamoDB) , Redis, MemcacheD DB, Amazon DynamoDB(not open source), Project Voldemort (Open Source based on DynamoDB)
- Document Databases
  - MongoDB, CouchDB, Terrastore, OrientDB, RavenDB, Lotus Notes storage Facility
- Column Family Databases
  - Cassandra, HBase(Open Source based on Big Table) , Hypertable, Google Big Table(not open source),
- Graph Databases
  - neo4J, Inifinite Graph, OrientDB, FlockDB
- **Hadoop Distributed File System (HDFS) – Hadoop**
  - **Does not fit the NoSQL categories as it is a file system but important in this Big Data World**

Apache Hadoop Project

- The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using a simple programming model
- http://hadoop.apache.org/
- Two main components
  - YARN (for managing processing)
  - Distribute File System (HDFS) (for managing storage)

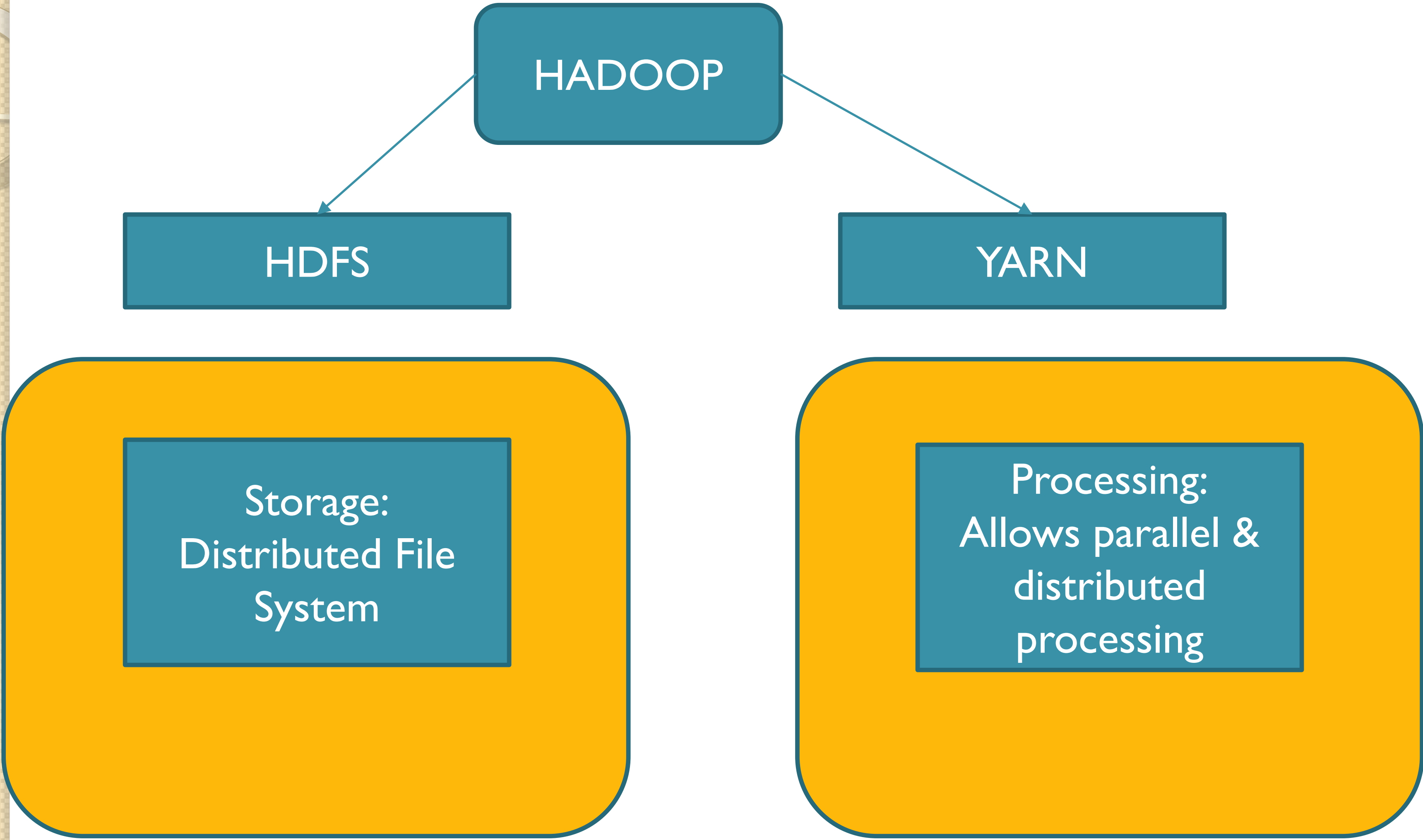
RDBMS Vs Hadoop

	RDBMS	Hadoop
Data Types	RDBMS relies on the structured data and the schema of the data is always known.	Any kind of data can be stored into Hadoop i.e. Be it structured, unstructured or semi-structured.
Processing	RDBMS provides limited processing capabilities.	Hadoop allows us to process the data which is distributed across the cluster in a parallel fashion.
Schema on Read Vs. Write	RDBMS is based on **'schema on write'** where schema validation is done before loading the data. CRUD supported	Hadoop follows the **'schema on read'** policy. Write once read many environment (**WORM**)
Read/Write Speed	In RDBMS, **reads are fast** because the schema of the data is already known.	The **writes are fast** in HDFS because no schema validation happens during HDFS write.
Cost	Typically licensed software, therefore, you have to pay for the software.	Hadoop is an open source framework.
Best Fit Use Case	RDBMS is used for OLTP (Online Transactional Processing) system. Is also used for OLAP\ data warehousing	Hadoop is used for Data discovery, ELT, data analytics or OLAP systems.

HADOOP Use Cases

- Data Warehouse Offload
  - Use Hadoop for long-term storage due to lower cost of storage and processing
  - Performing pre-integration ETL routines
  - Storing unprocessed data prior to being staged\integrated into a Data Warehouse
- Event Processing
  - Ingestion and processing of streaming data sources e.g. sensor data (e.g. RFID, CCTV cameras, temperature), log data , message data
  - Utilities include Storm, Flume, Spark Streaming, Kafka
- Advanced Analytics (Data Science)
  - Allow for machine learning\AI at scale

Hadoop-2.x



HADOOP

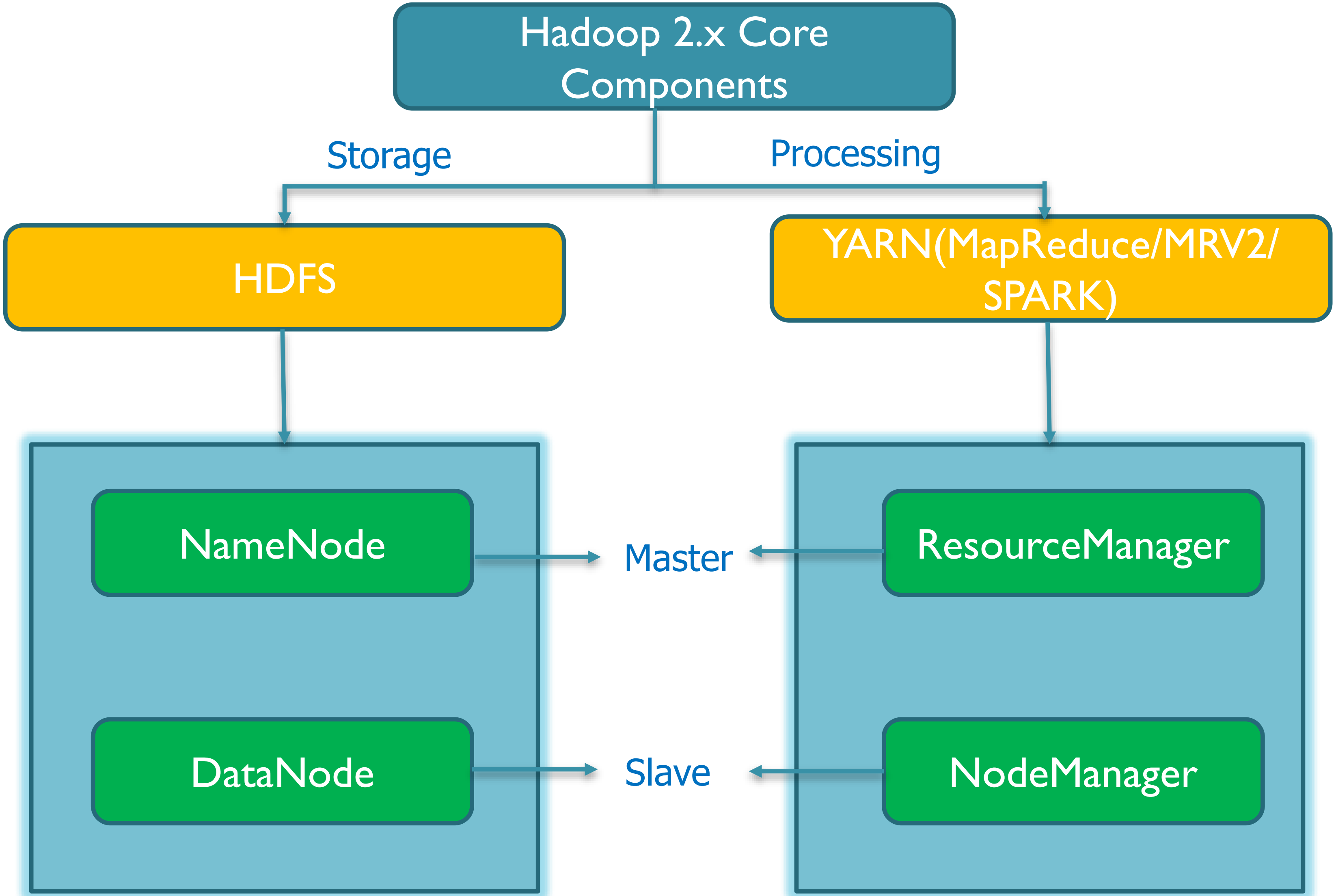
HDFS

YARN

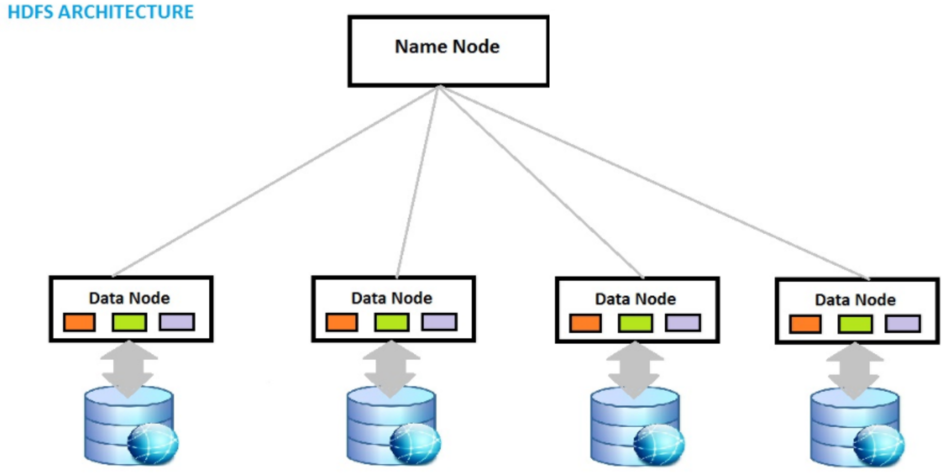
Storage:
Distributed File System

Processing:
Allows parallel & distributed processing

Hadoop2.x Main Daemons



NameNode and DataNode in HDFS



Data Nodes
(Commodity Hardware)

- NameNode
  - Master Daemon
  - Maintains and manages DataNodes
  - Records Metadata e.g. location of blocks stored, the size of the files, permissions, hierarchy; for performance held in memory)
  - Receives heartbeat and block report from all the DataNodes

- DataNodes
  - Slave Daemons
  - Stores actual data
  - Serves read and write requests from the clients
  - Sends a heartbeat to the NameNode

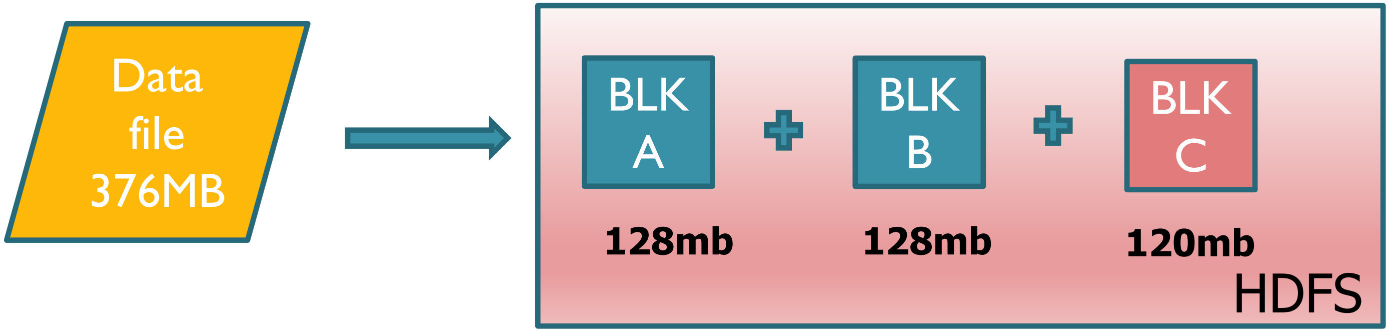
NameNode Metadata

Object	Block_id	Seq	Location	ACL	Checksum
/data/file.txt	Blk_00121	1	[DN1,DN2,DN3]	-rwxrwxrwx	8708b09….
/data/file.txt	Blk_00122	2	[DN2,DN3,DN4]	-rwxrwxrwx	cd786a87..
/data/file.txt	Blk_00123	3	[DN2,DN4,DN5]	-rwxrwxrwx	cd786a87..

Conceptual Representation of In_Memory Metadata

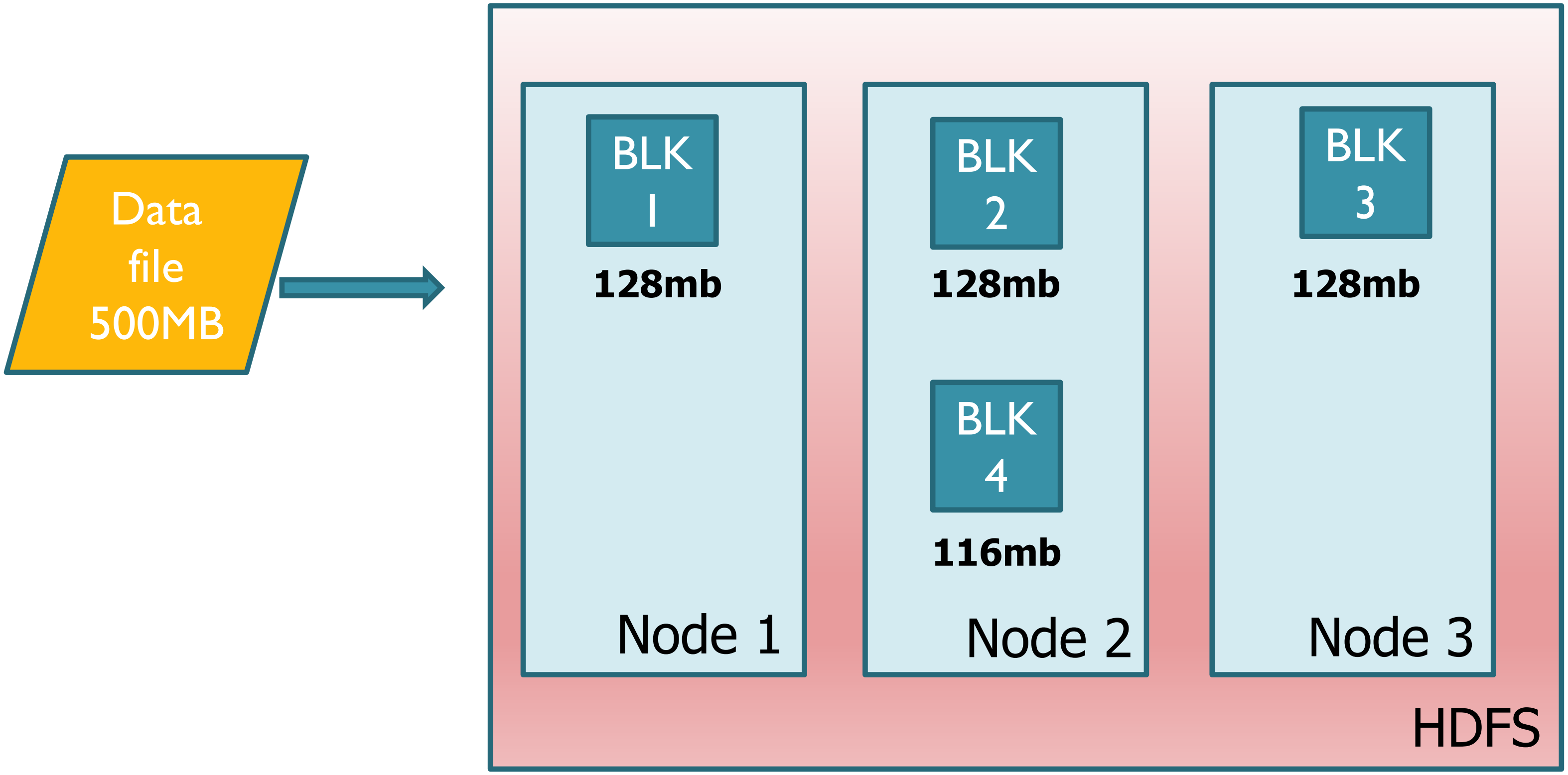
HDFS Blocks

- Each File is stored on HDFS Blocks
- The default size of each block is 128mb in Apache Hadoop 2.x (64mb in Apache Hadoop 1.x)
- Let us say we have a file 376mb it will be split into chunks as follows:



Blocks are Distributed

- If cluster contains more that one node, blocks are distributed



Data file 500MB

BLK I
128mb

BLK 2
128mb

BLK 4
116mb

BLK 3
128mb

Node 1

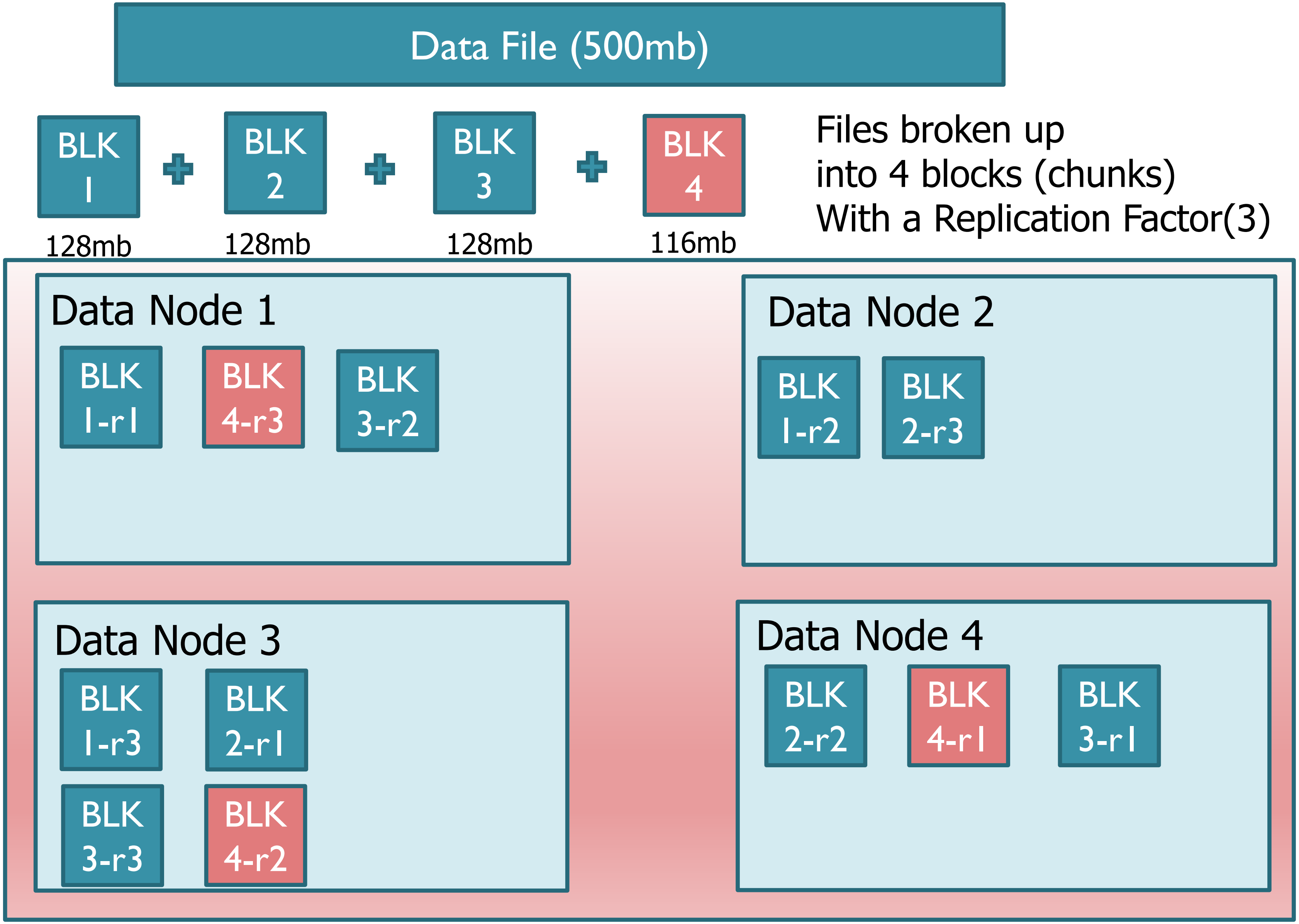
Node 2

Node 3

HDFS

Note: Each Block is also replicated in the DataNodes (Not shown in the diagram)

HADOOP Block Replication



Data File (500mb)

BLK 1 + BLK 2 + BLK 3 + BLK 4

128mb 128mb 128mb 116mb

Files broken up
into 4 blocks (chunks)
With a Replication Factor(3)

Data Node 1

BLK 1-r1 BLK 4-r3 BLK 3-r2

Data Node 2

BLK 1-r2 BLK 2-r3

Data Node 3

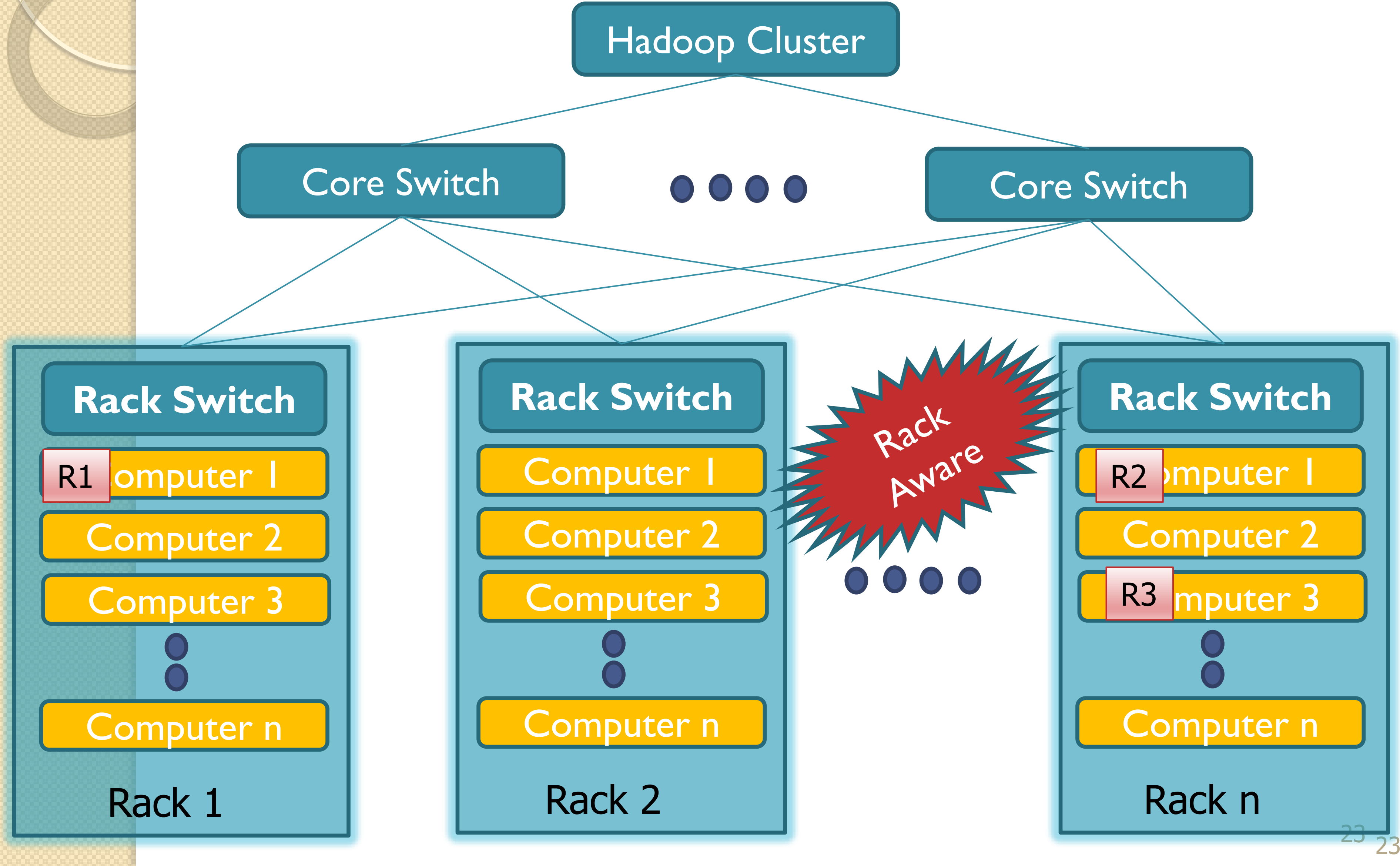
BLK 1-r3 BLK 2-r1

BLK 3-r3 BLK 4-r2

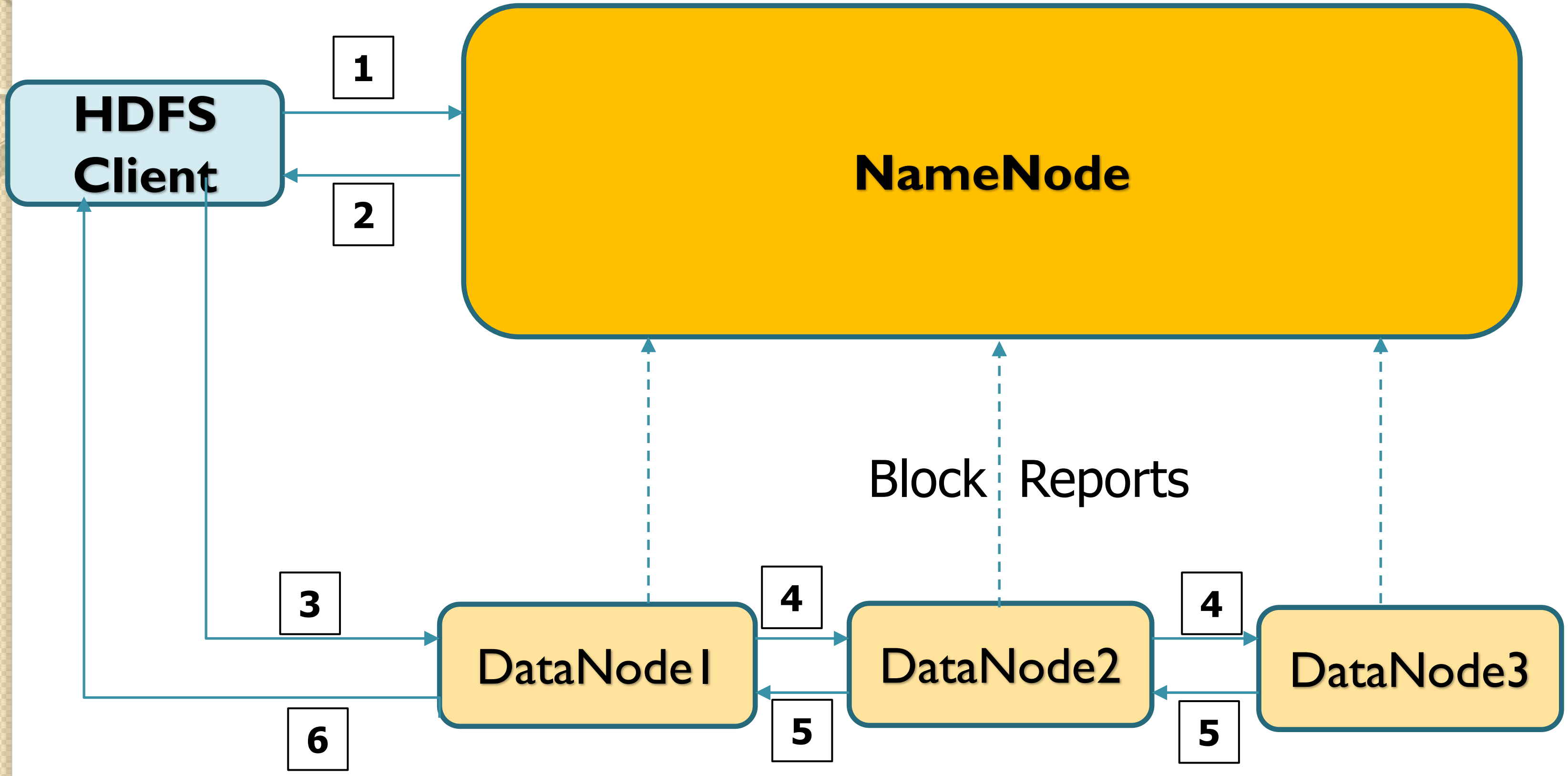
Data Node 4

BLK 2-r2 BLK 4-r1 BLK 3-r1

Hadoop Cluster Architecture

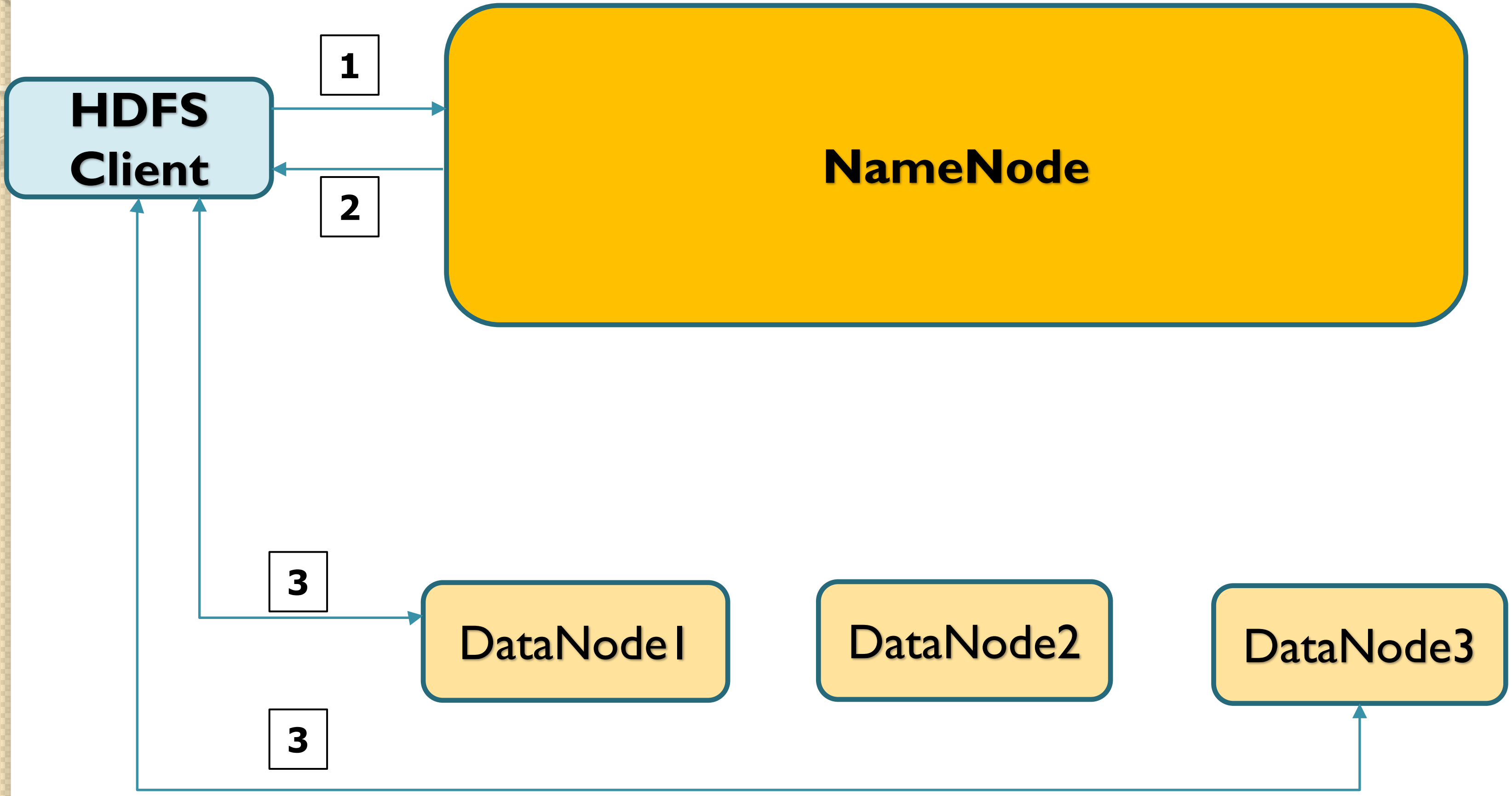


A Write Operation

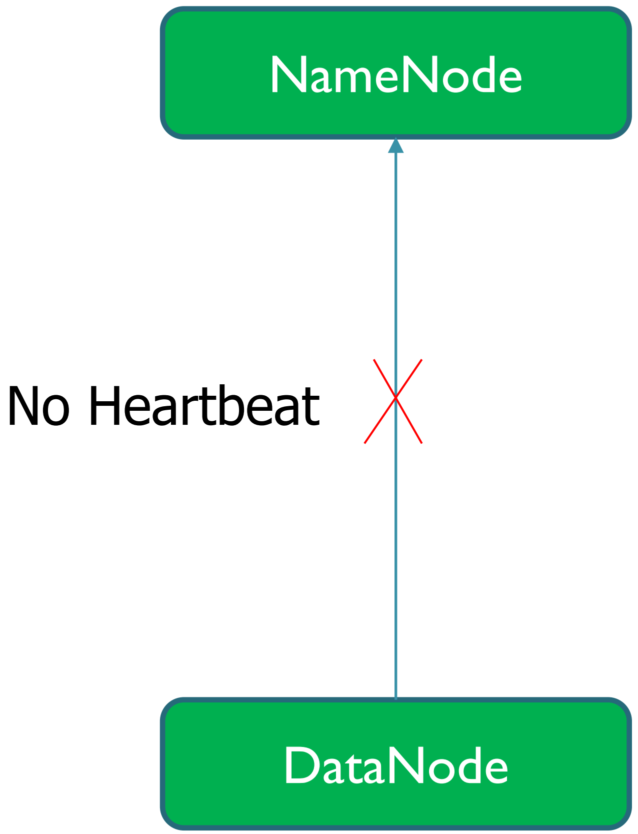


"Block Replication Pipeline"

A Read Operation



Data Node Failure



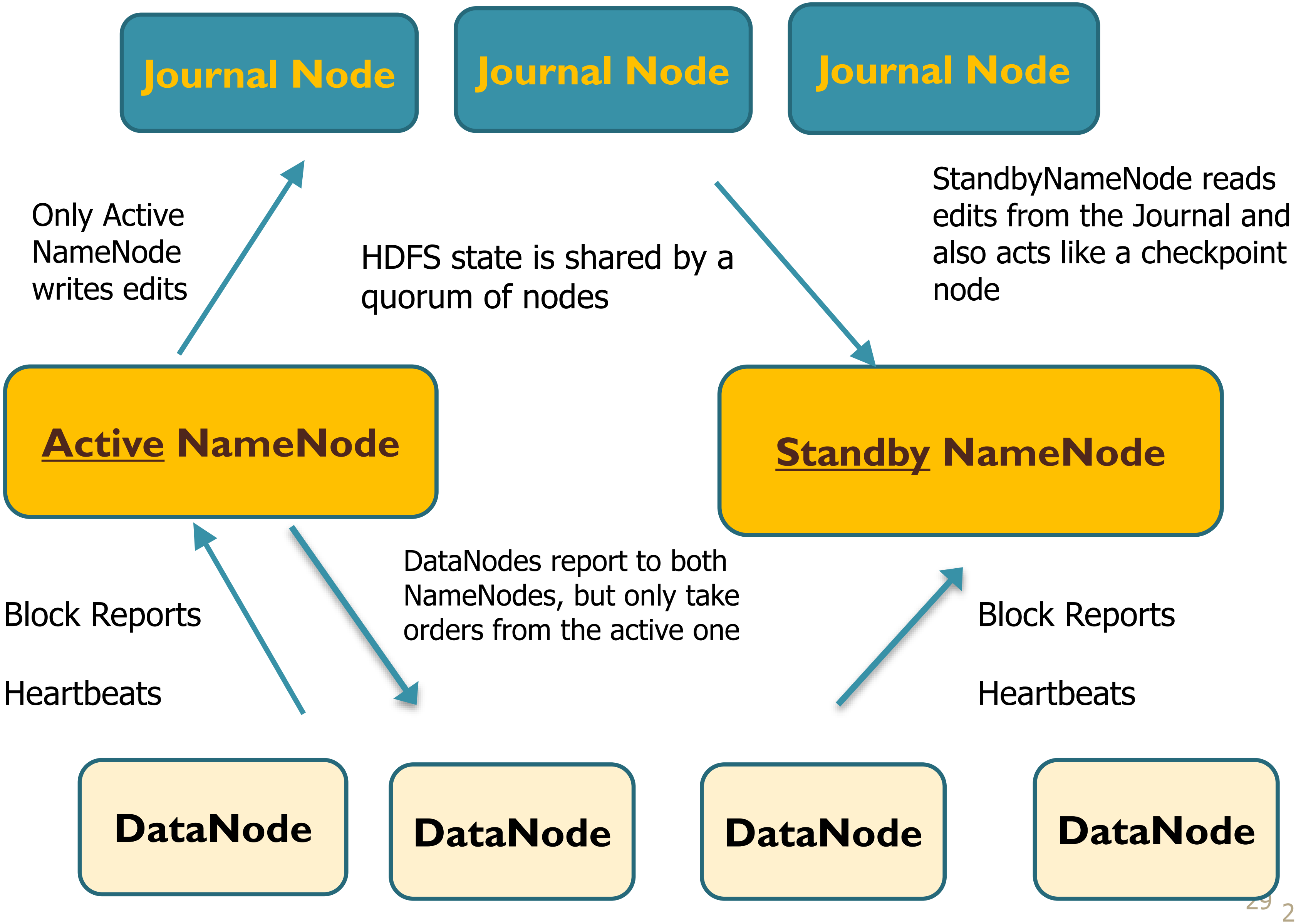
- NameNode detects no heartbeat from DataNode
- Marked as **dead** after specific period of time
- NameNodes replicates blocks to another Datanode (using the replicas created earlier)
- **Note** NameNode has a list of all the blocks on the Datanode(known as a block report)

Secondary NameNode(2NN) - Optional

- Performs periodic **checkpoints** that evaluate the status of the NameNode
- Main aim is to **improve NameNode restarts**
- NameNode keeps Metadata in memory. MetaData contained on disk in
  - **fsimage_**
    - An image snapshot of the HDFS file state when the Name-node was started
  - **edit_***
    - A series of modifications made to HDFS during the running of the NameNode (like a transaction log in RDB) or REDO log in Oracle
- Secondary NameNode
  - Periodically downloads these files from the Namenode
  - Merges them and loads an image back to the NameNode

***Secondary NameNode not an active failover node**

Standby NameNode for HA -Optional



Journal Node

Journal Node

Journal Node

Only Active NameNode writes edits

HDFS state is shared by a quorum of nodes

StandbyNameNode reads edits from the Journal and also acts like a checkpoint node

Active NameNode

Standby NameNode

Block Reports

Heartbeats

DataNodes report to both NameNodes, but only take orders from the active one

Block Reports

Heartbeats

DataNode

DataNode

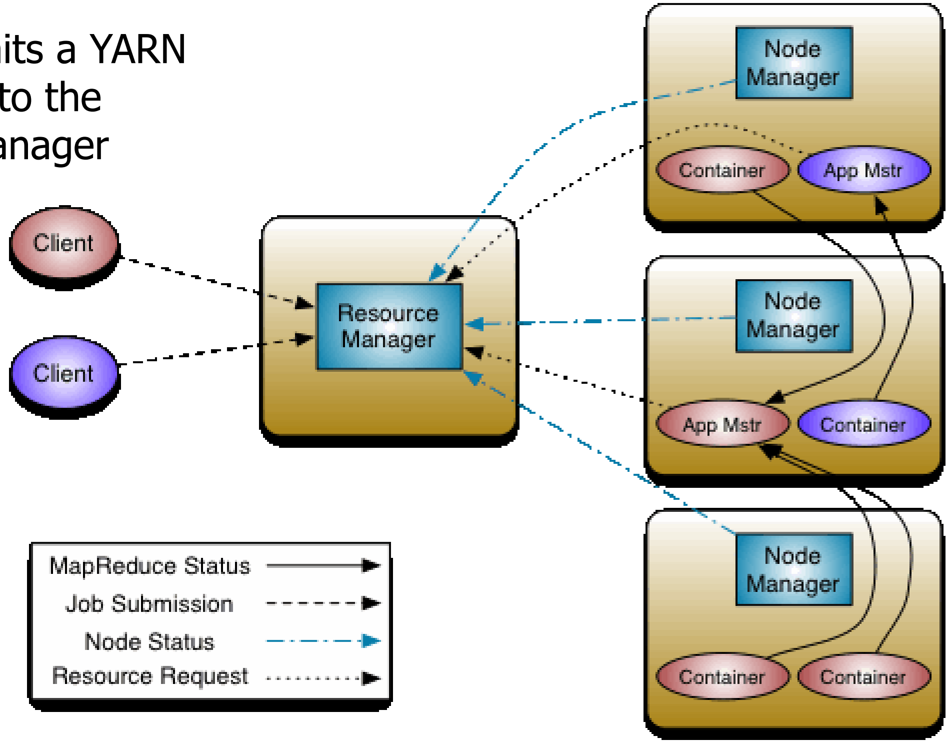
DataNode

DataNode

YARN (Yet Another Resource Negotiator)

The ResourceManager assigns an Application Master (a container) for the Application

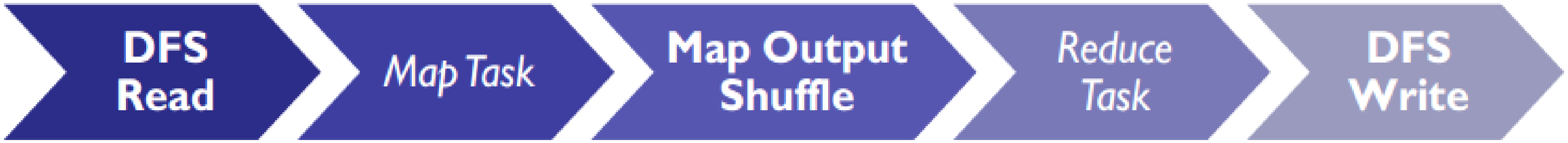
Client submits a YARN application to the ResourceManager



Tasks for an application report their progress to the ApplicationMaster for the application

Map/Reduce

- **MapReduce**
  - Distributed computation framework
  - Optimized for batch processing
  - Typical I/O profile:



Count words in docs – The "Hello World" Of Map Reduce

- Input consists of (filename, file-contents) pairs

- map(key=filename val=file-contents):
  - For each word *w* in file-contents, emit (w, "1")

- reduce(key=word, values=uniq_counts):

  Sum all "1"s in values list

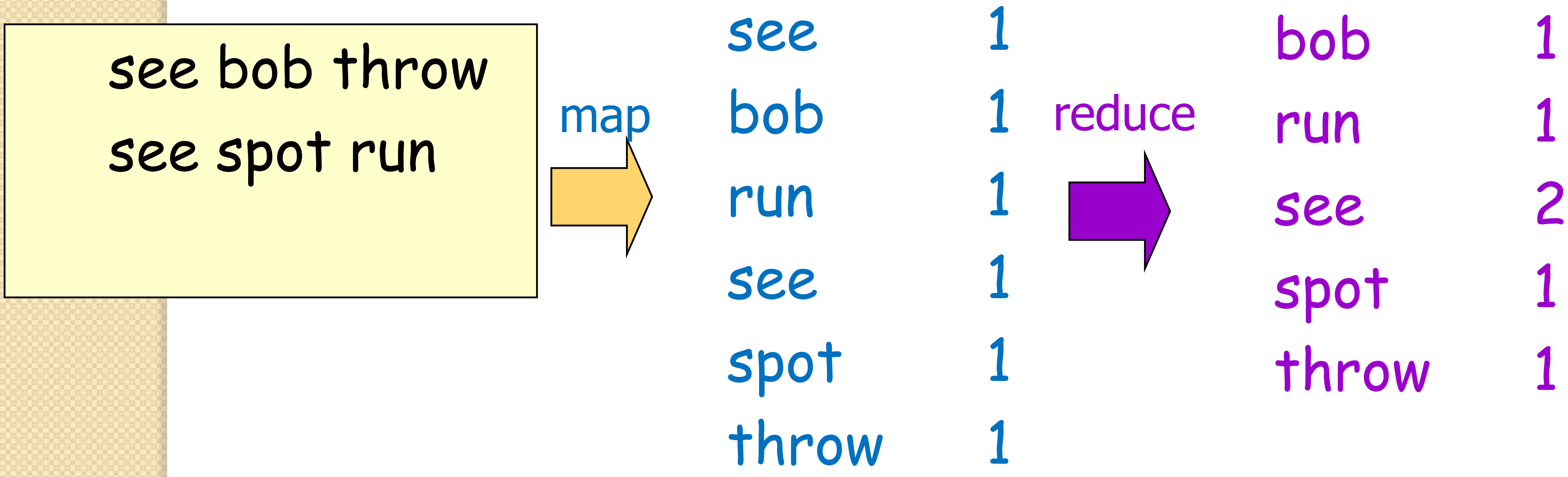
```
sum = 0
For each value in values:
Sum = sum + value
Emit result "(word, sum)"
```

Word Count, Illustrated

- map(key=filename val=file-contents):
  - For each word *w* in file-contents, emit (w, "1")

- reduce(key=word, values=uniq_counts):

```
Sum all "1"s in values list
     sum = 0
     For each value in values:
     Sum = sum + value
     Emit result "(word,  sum)"
```



see bob throw
see spot run

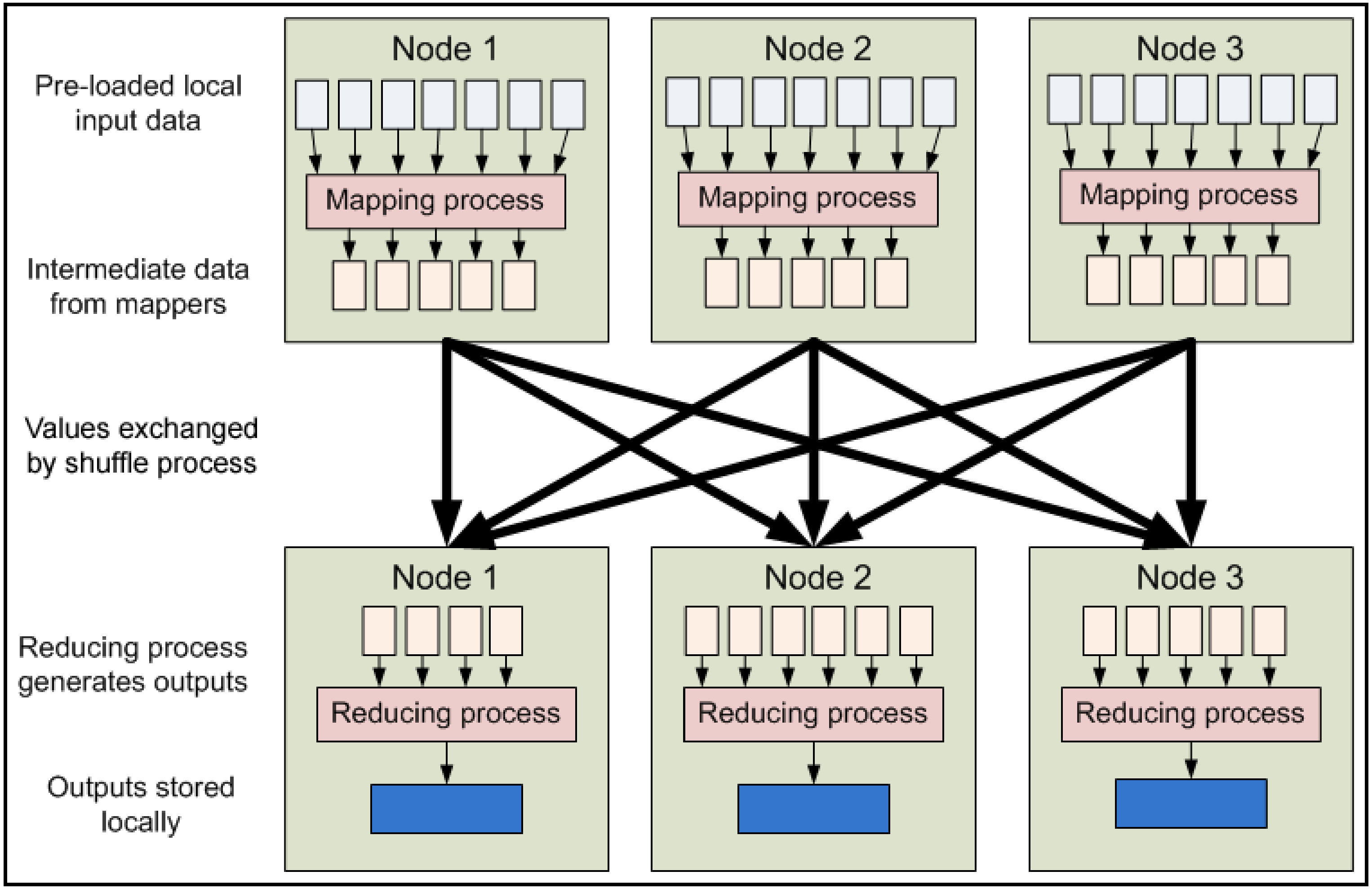
map

see	1
bob	1
run	1
see	1
spot	1
throw	1

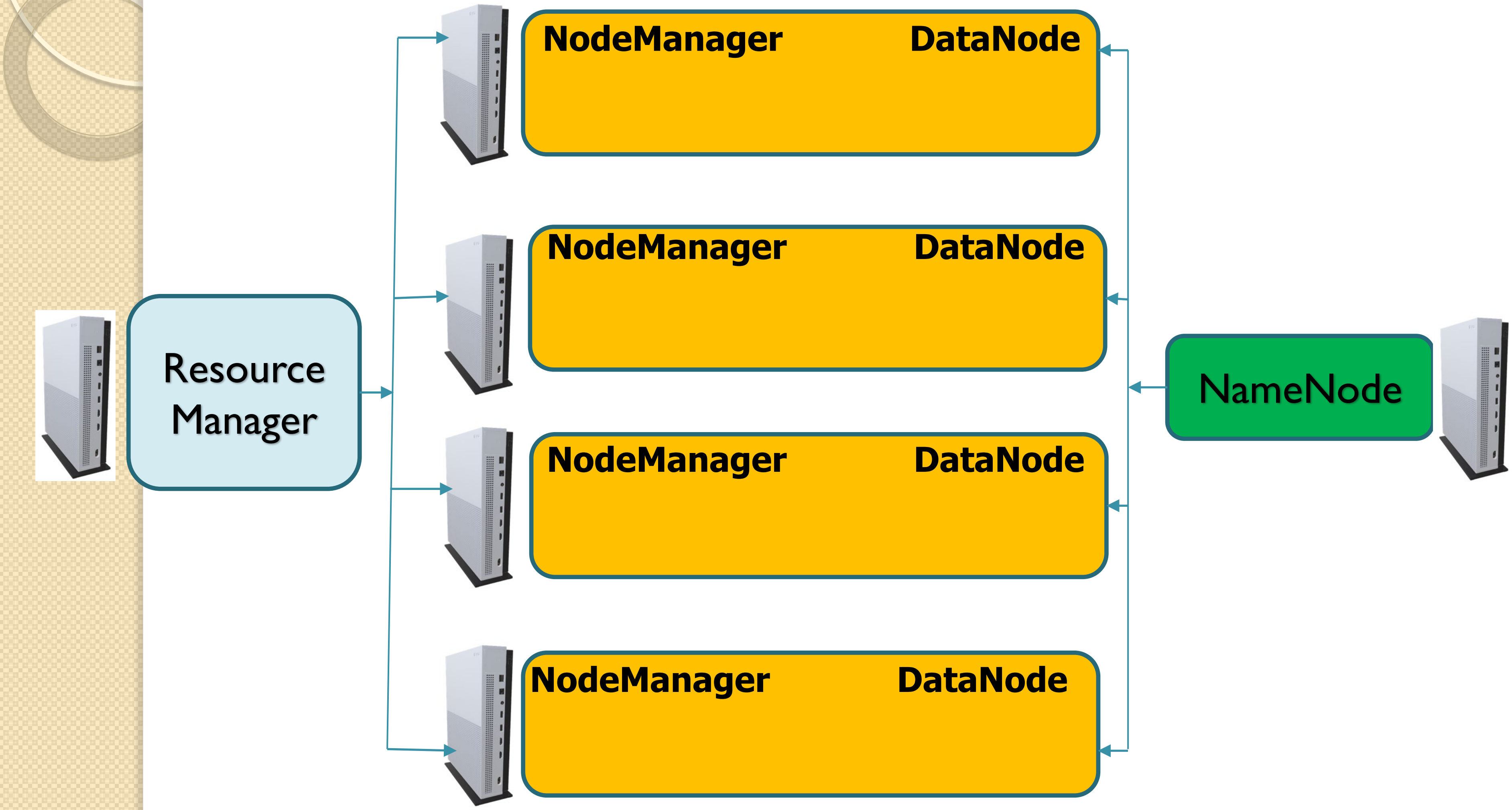
reduce

bob	1
run	1
see	2
spot	1
throw	1

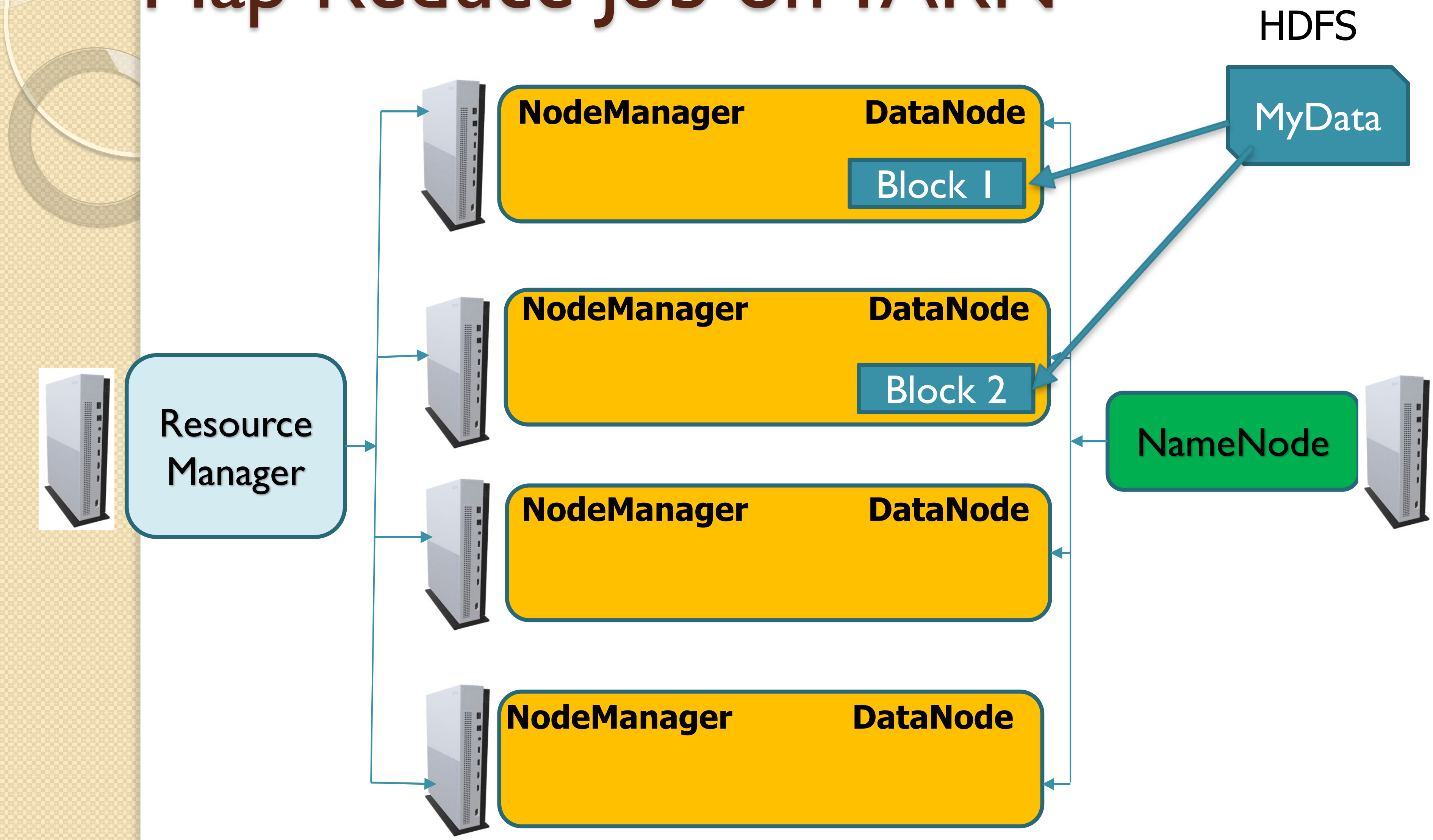
Mapping and Reducing tasks run on data nodes containing individual records of data



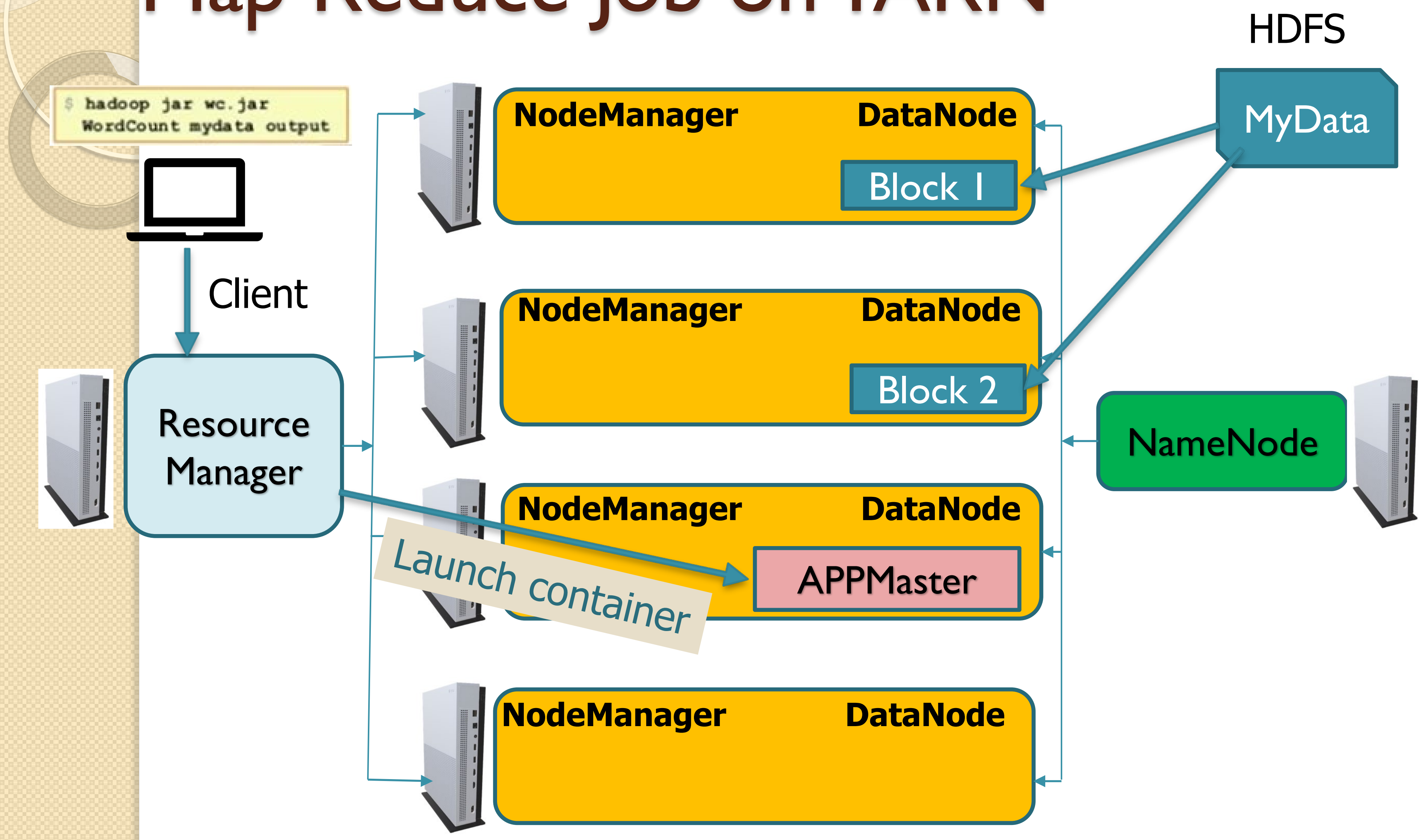
Map Reduce Job on YARN



Map Reduce Job on YARN

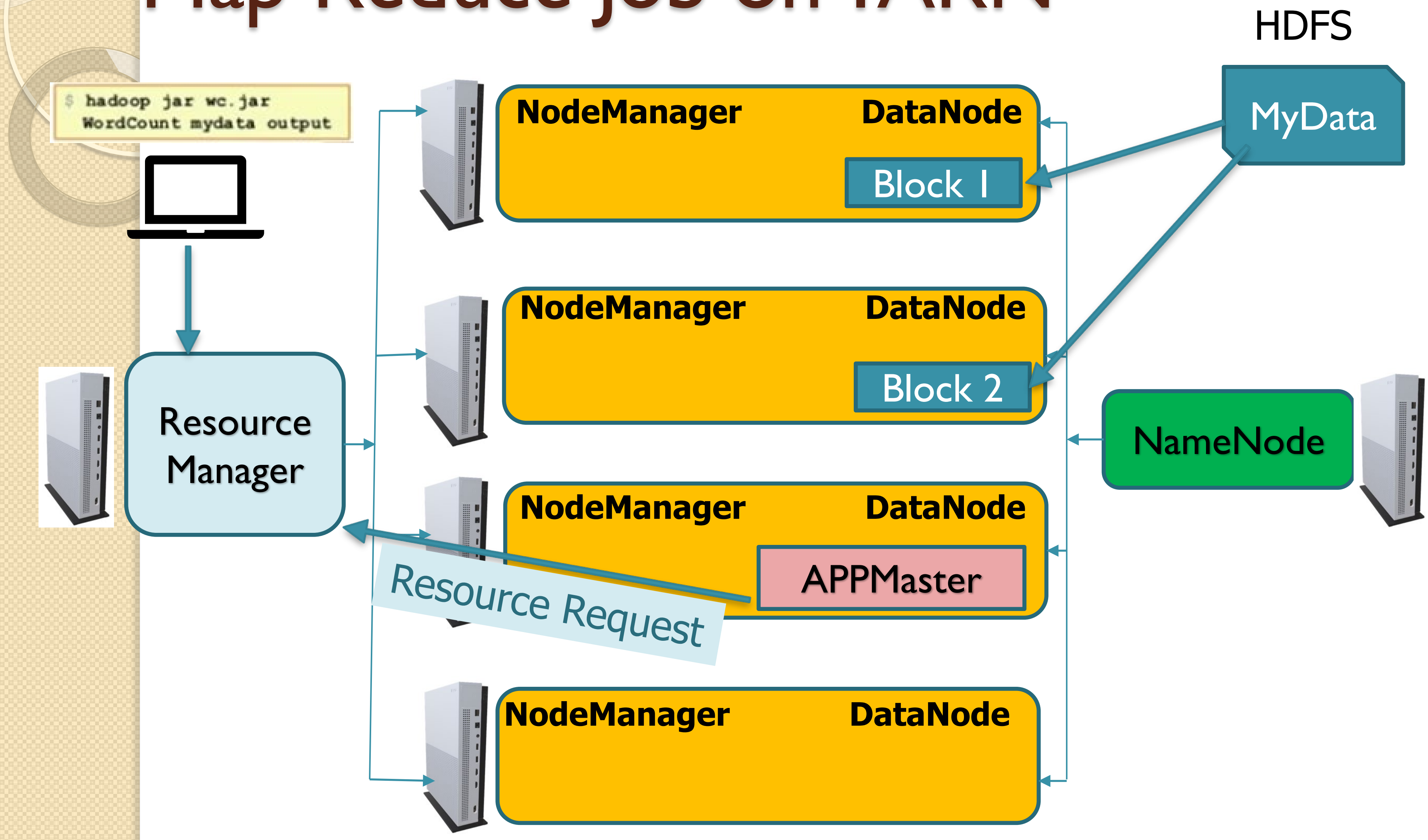


Map Reduce Job on YARN



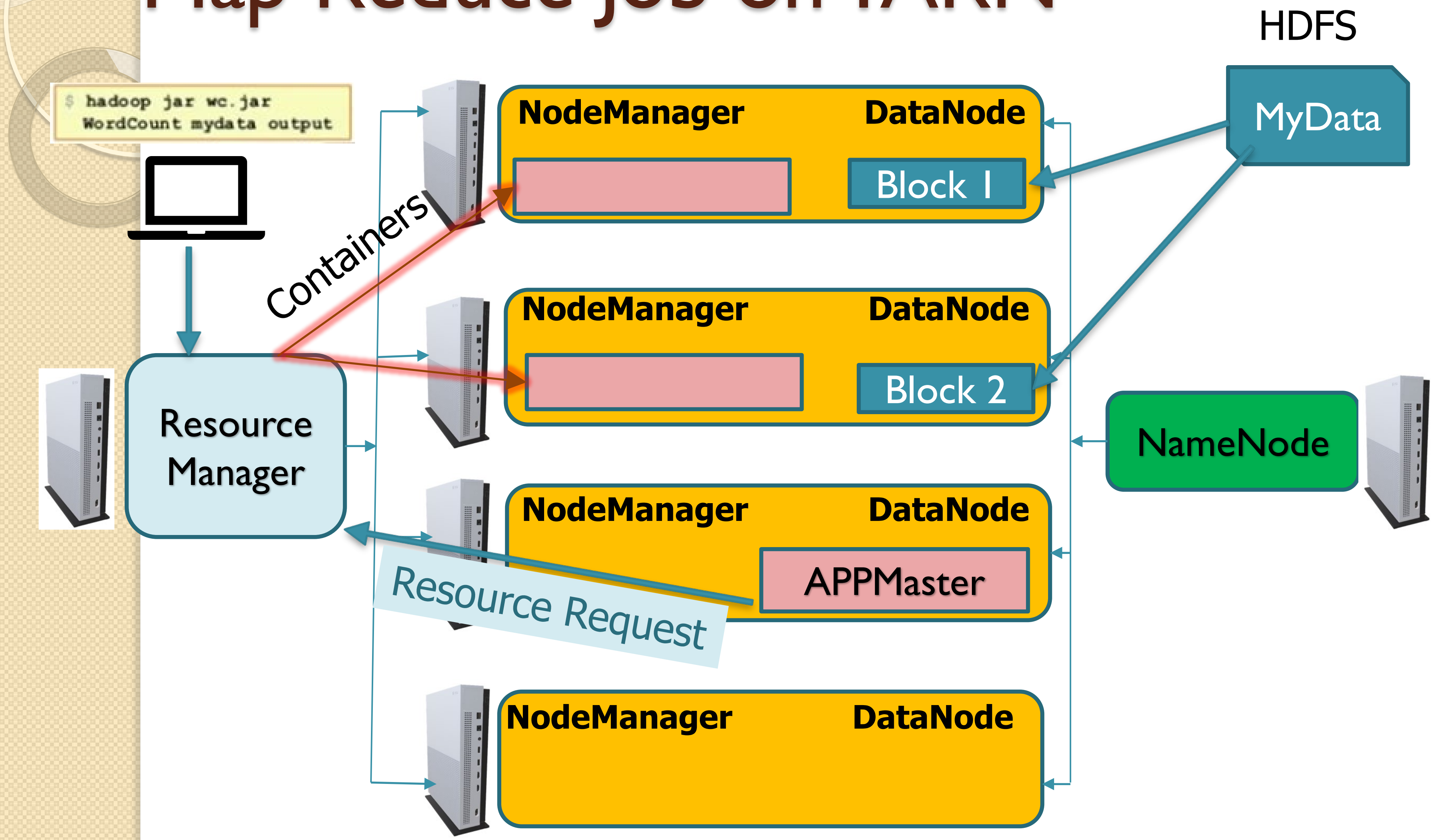
```
$ hadoop jar wc.jar
  WordCount mydata output
```

Map Reduce Job on YARN



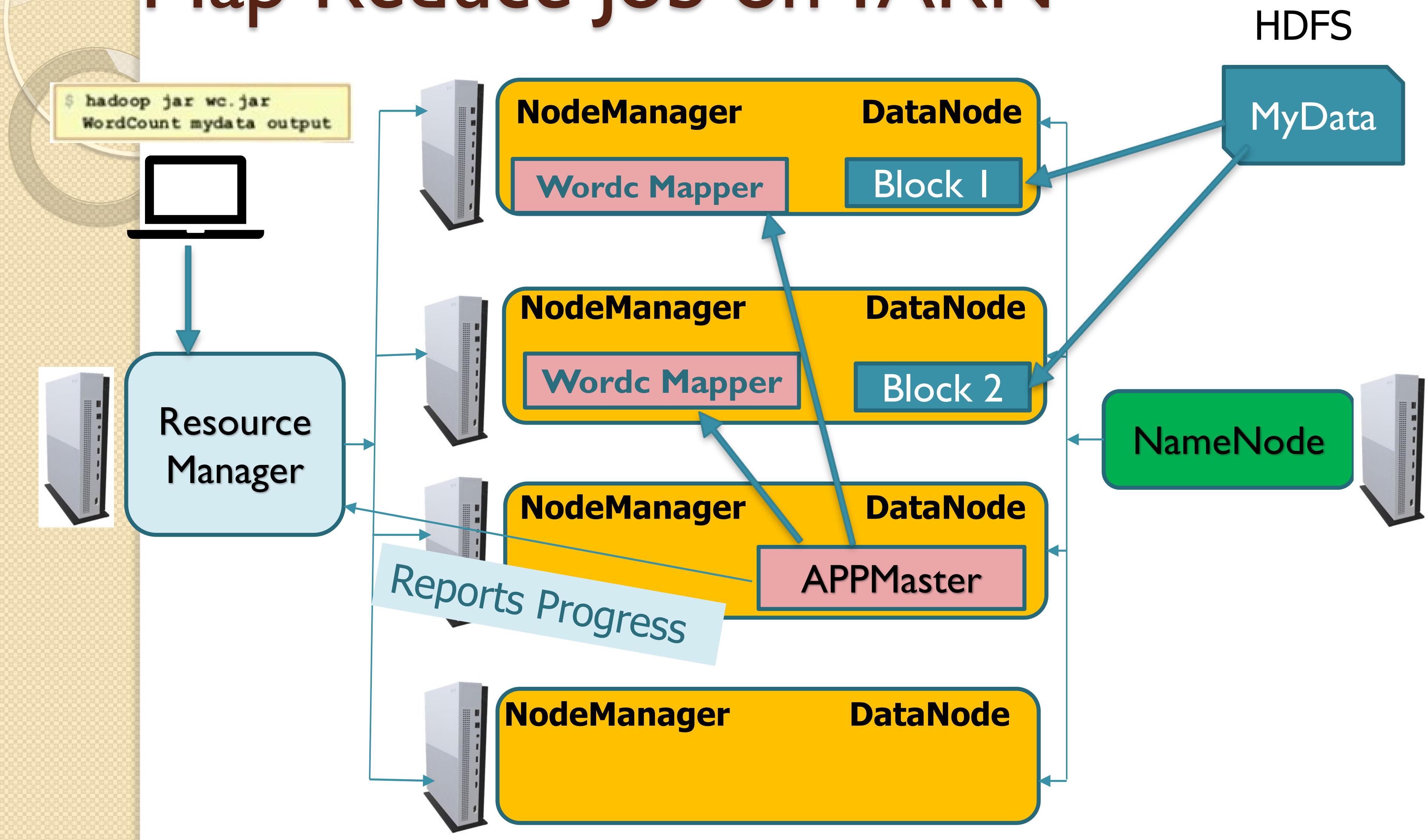
```
$ hadoop jar wc.jar
  WordCount mydata output
```

Map Reduce Job on YARN



```
$ hadoop jar wc.jar
WordCount mydata output
```

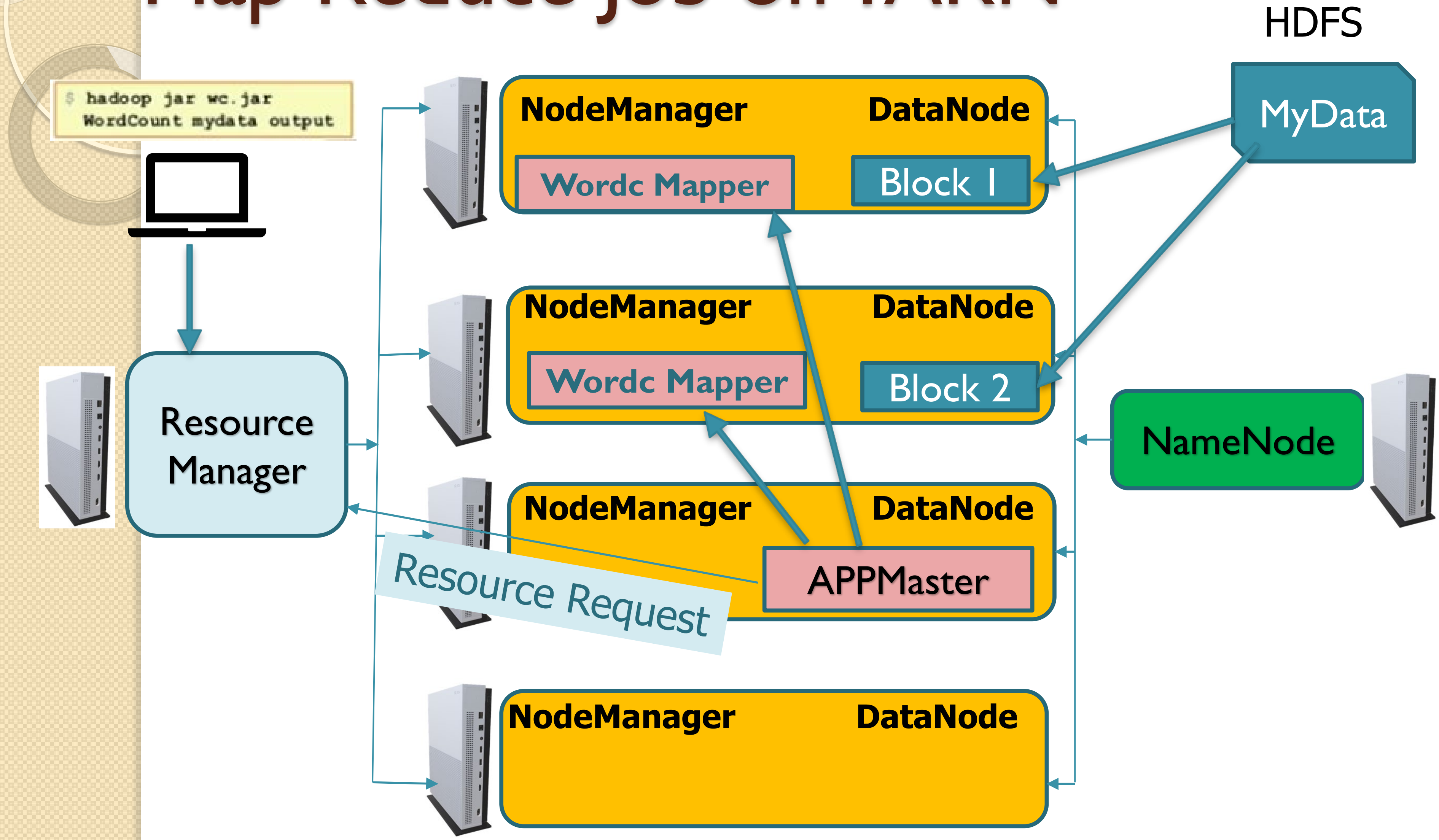
Map Reduce Job on YARN



```
$ hadoop jar wc.jar
 WordCount mydata output
```

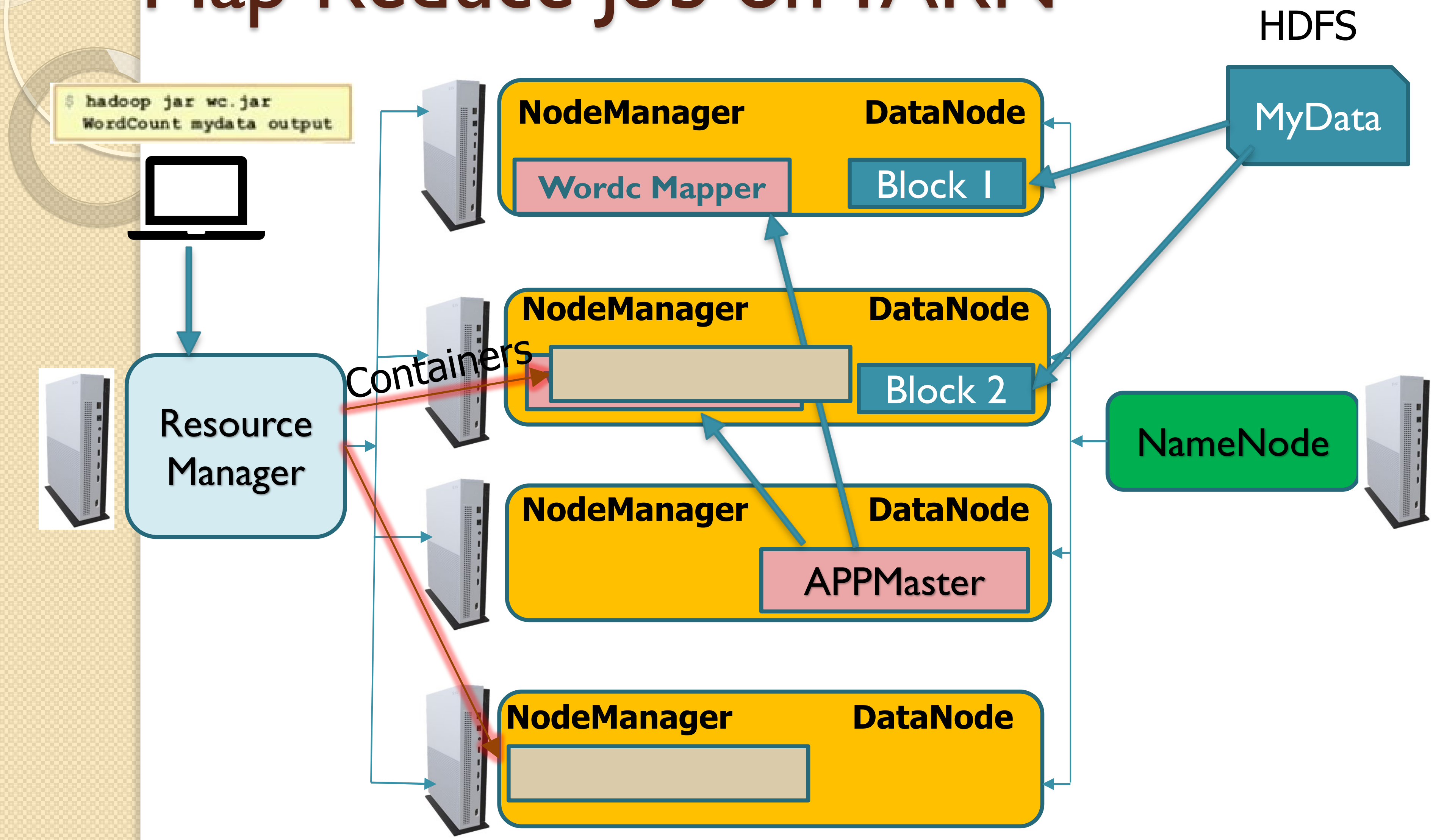
Note: Reducer Containers required

Map Reduce Job on YARN



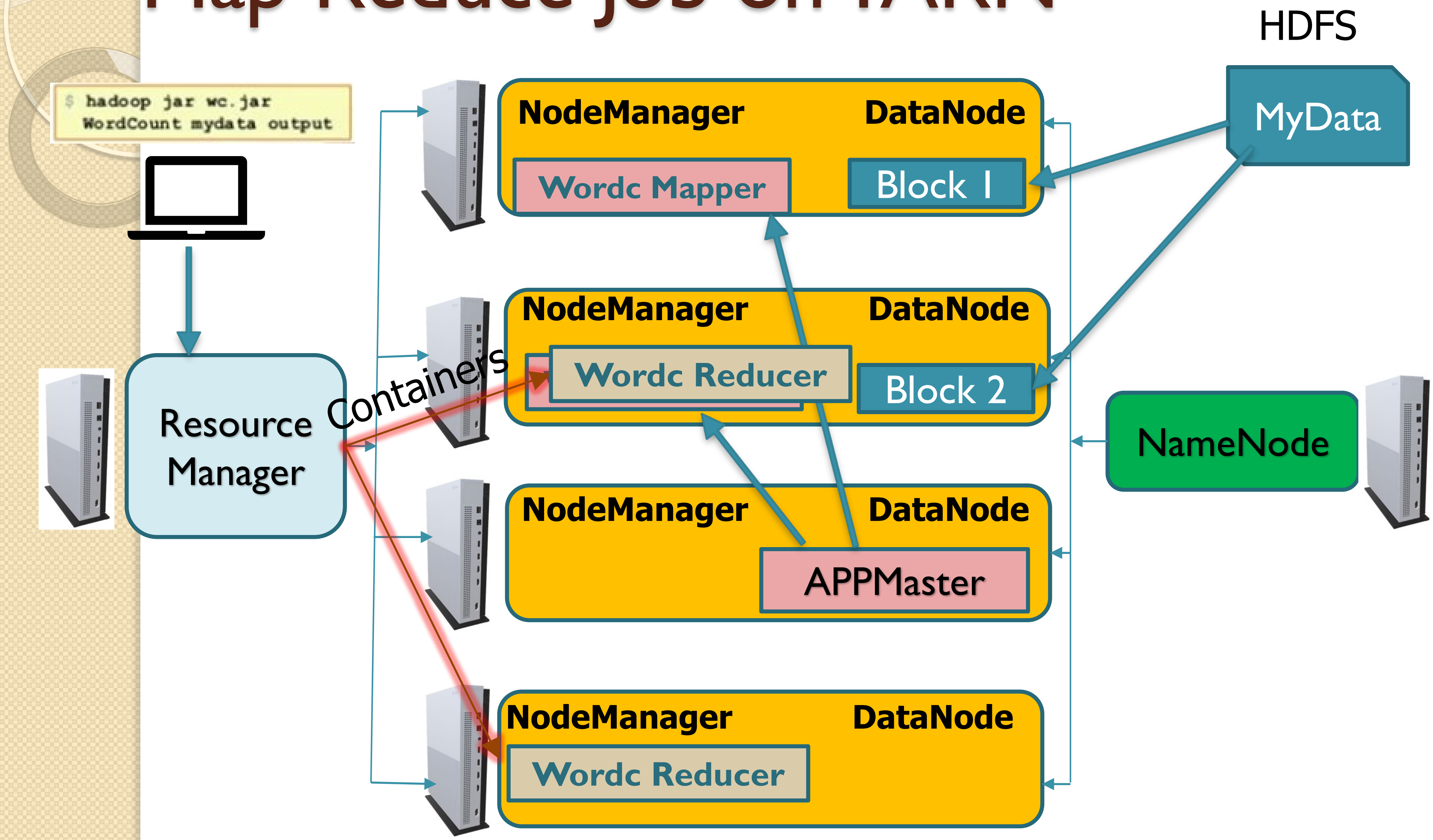
```
$ hadoop jar wc.jar
  WordCount mydata output
```

Map Reduce Job on YARN



```
$ hadoop jar wc.jar
  WordCount mydata output
```

Map Reduce Job on YARN



```
$ hadoop jar wc.jar
WordCount mydata output
```