The Hadoop Distributed Filesystem

An Introduction

Summary from White, T. *Hadoop: The definitive guide*. "O'Reilly Media, Inc.", 2014.

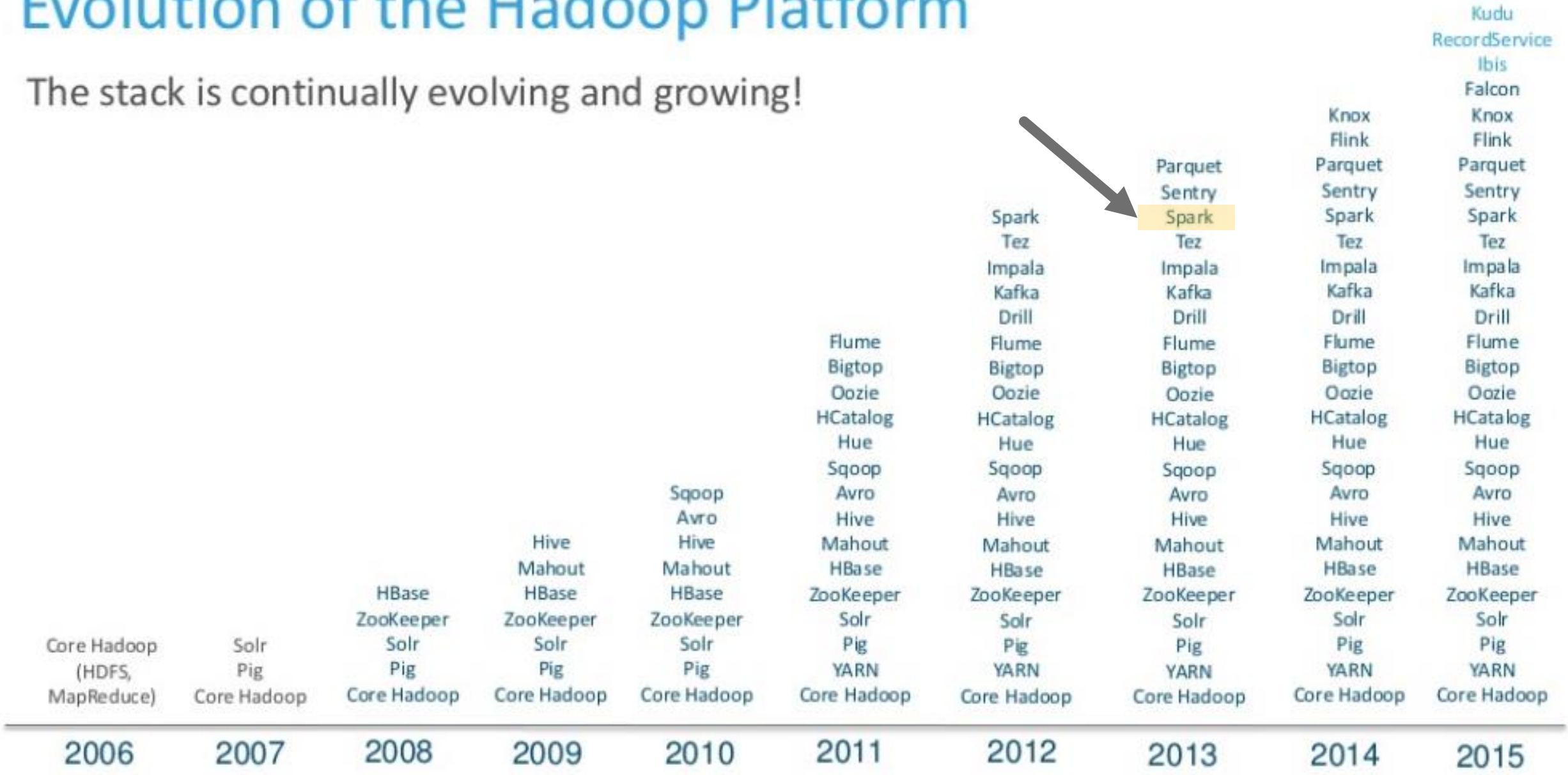

What is
Hadoop?

Hadoop's Original Architecture

cloudera

Evolution of the Hadoop Platform

The stack is continually evolving and growing!



HDFS (HaDoop File System)

- Replicates portions of a file (blocks) over cluster
- Automatically
  - Recovers from data failures
  - De/Compresses files (when configured to do so)
- HDFS is a core component of Hadoop
  - MapReduce and most other components use it
  - However, can substitute other files systems like Amazon S3

Object storage (S3) is very popular alternative. https://www.druva.com/blog/object-storage-versus-block-storage-understanding-technology-differences/

HDFS designed for analysis of large files

- Very large files (>100 MB – terabytes)
- Streaming access (most or all data is read for each job)
- Not low-latency, random access (it may take time to get the system going before it produces the first data byte)
  - Thus may not be good for exploratory analysis of data subsets
- Not lots of (small) files
  - Design limits the number of files (stored in a NameNode)
- Not lots of clients with arbitrary file modifications
  - It's not a transaction processing system!

In general, big data storage
(HDFS distributed file system, S3 object store)
is intended for large files that do not change
and having reliable, parallel access

update is much slower than standard file system

Let’s assume that we are doing
parallel processing

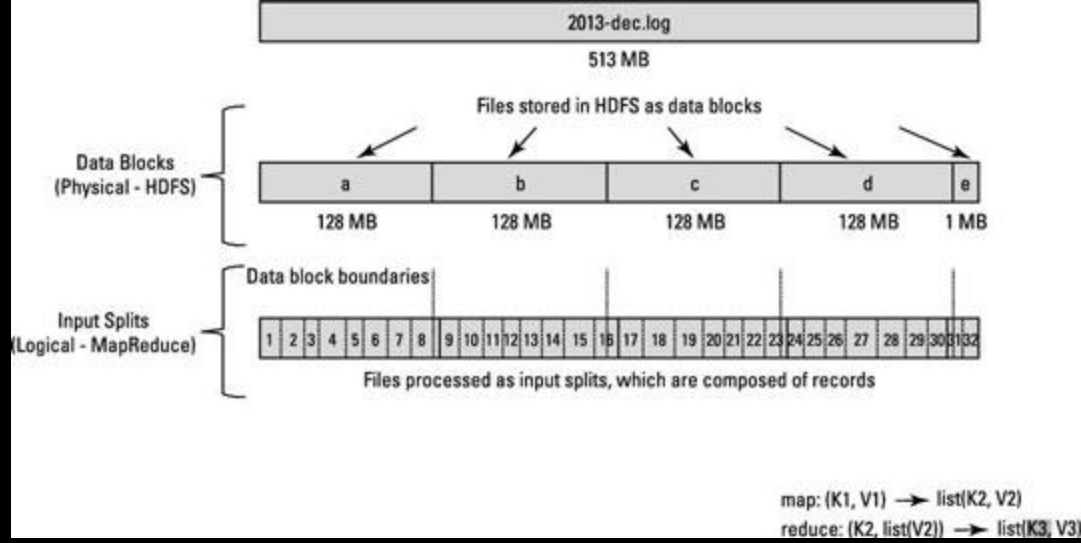
How can we **distribute** our **data** to all the processors?

Let’s assume that we have a **data node grid**

Cluster terminology

- A **computer cluster** is a set of loosely or tightly connected computers that work together so that, in many respects, they can be viewed as a single system—Wikipedia
- A **data center** is a facility used to house computer systems and associated components, such as telecommunications and storage systems.—Wikipedia
- Distinction
  - Cluster is collection of computers under an architecture (e.g., Hadoop)
  - Datacenter is a business function for processing data, almost always at a single location (i.e., not a virtual DC connected via Internet).
  - Hadoop clusters do not interconnect through the Internet, but instead are locally networked (LAN) computers.

Job data terminology

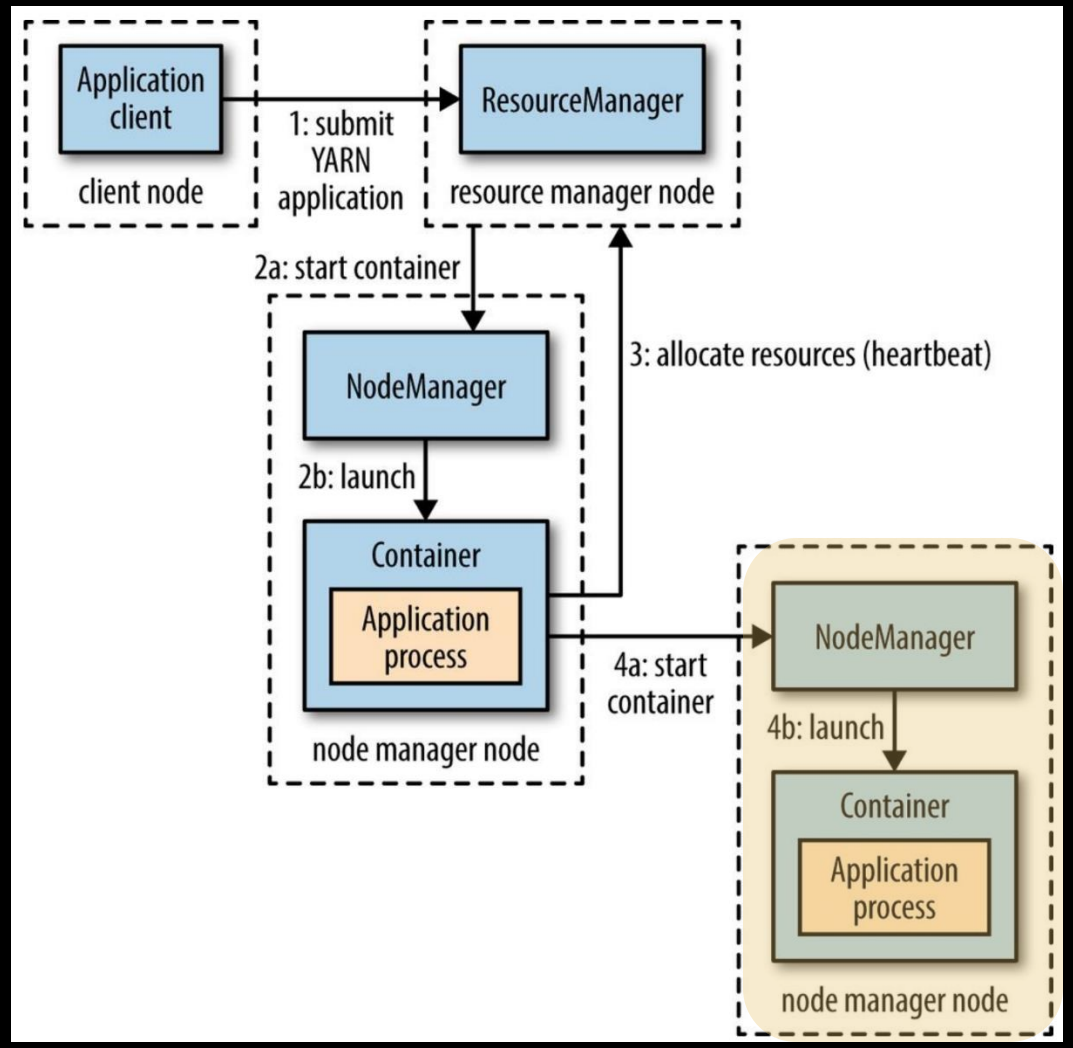


- Job
  - A unit of work that processes client input and provides results to the client
- Task
  - Smaller units of work, derived from the job, such as map or reduce (kinds of) tasks. Each task processes a partition of data.
- Data
  - Information stored on a disk and provided by a data node. Data is stored in various file formats, which include support for SQL and NoSQL databases.
- File system block (physical)
  - The smallest unit of bytes tracked by the file system. By default becomes a partition.
  - HDFS block is 128 MB by default. (Windows FS is typically 4K)
- Partition
  - A portion of the data being processed, created during write. It is possible to repartition data.
- Split (logical records for input to a task *for MapReduce*)
  - Input divided into smaller pieces for task processing.
  - A MapReduce Split is filled with data from an HDFS block

Data Node

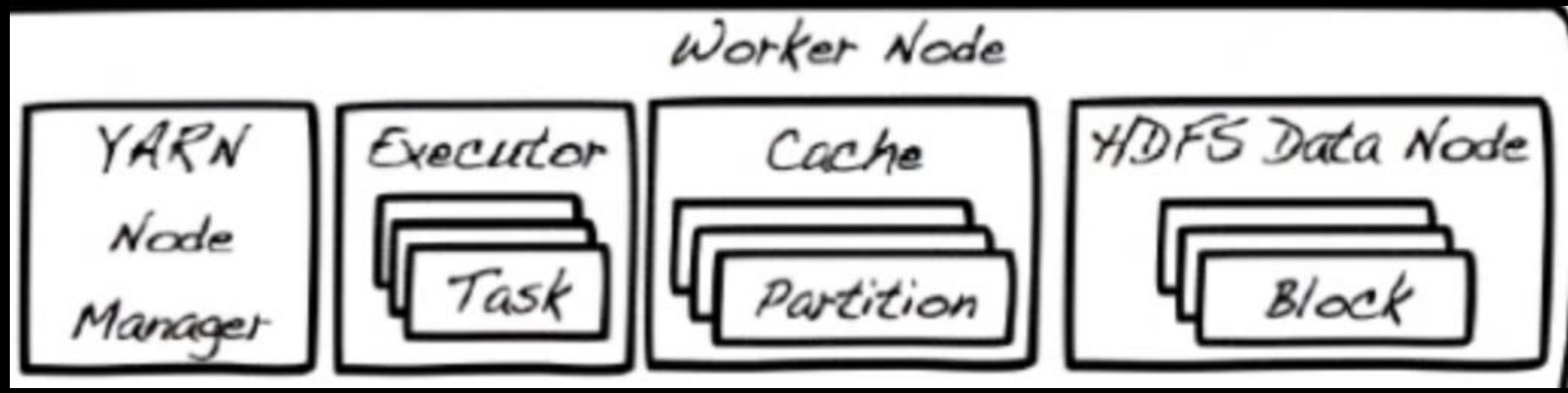
stores **data** in the Hadoop file system;
contains **tasks** that process data

- Data Node has
  - Node manager, which manages all requests to the node and its internal processing
  - Container for application **tasks**
  - **Data** stored on local file system



Data node

- aka worker node
  - Node manager, which communicates with Resource Manager (YARN)
  - Task Executor
  - HDFS data
  - Cache for data

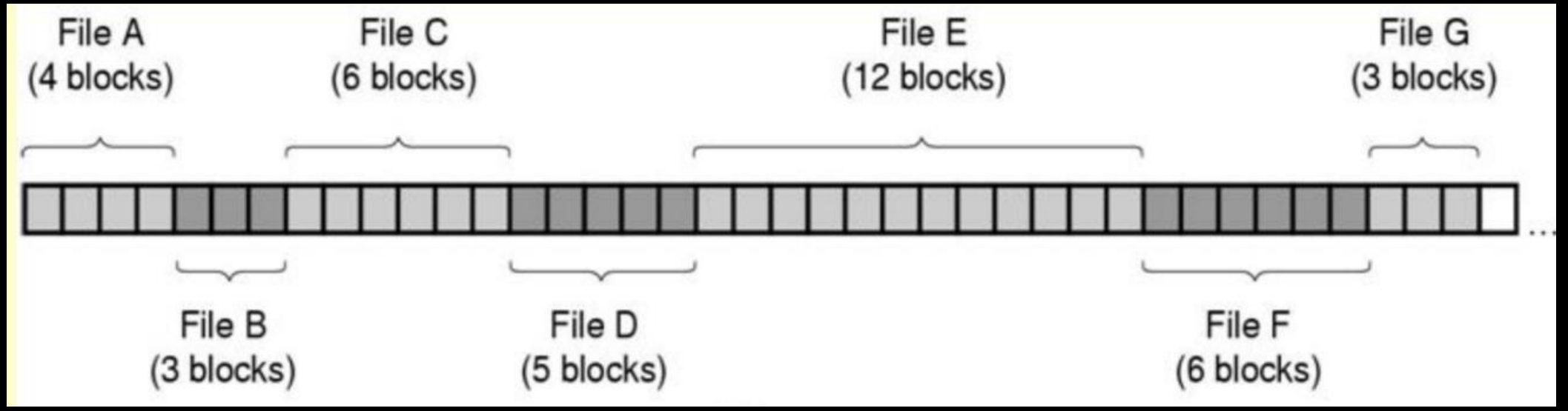


Yarn and HDFS are closely integrated

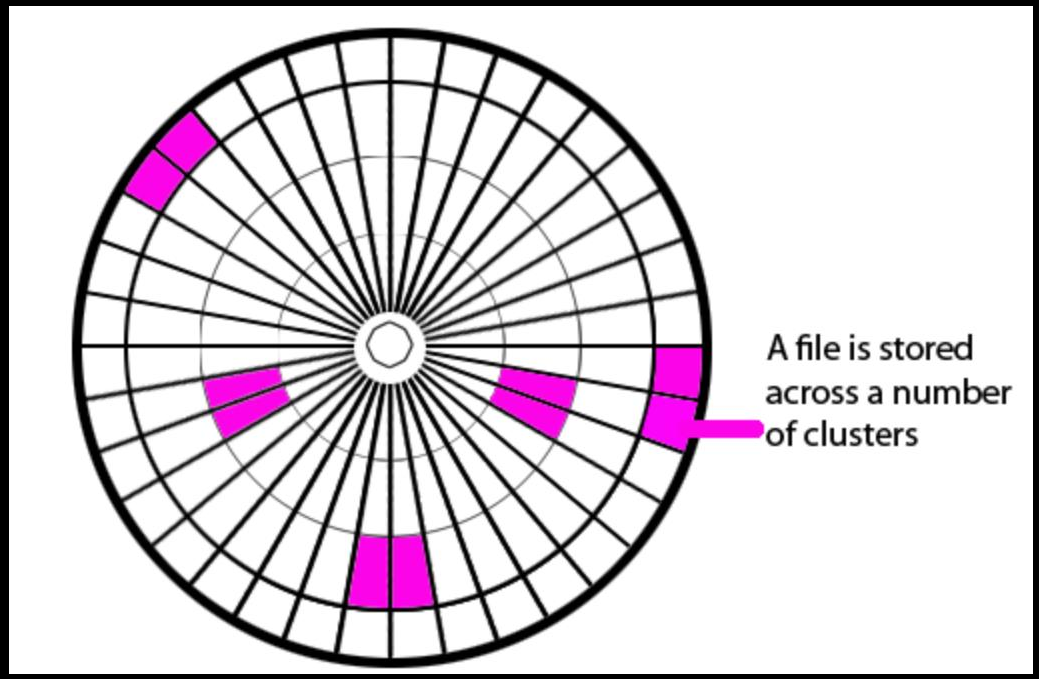
How can we **distribute** our **data** to all the processors?

Consider a very large file, having **Gigabytes** of data

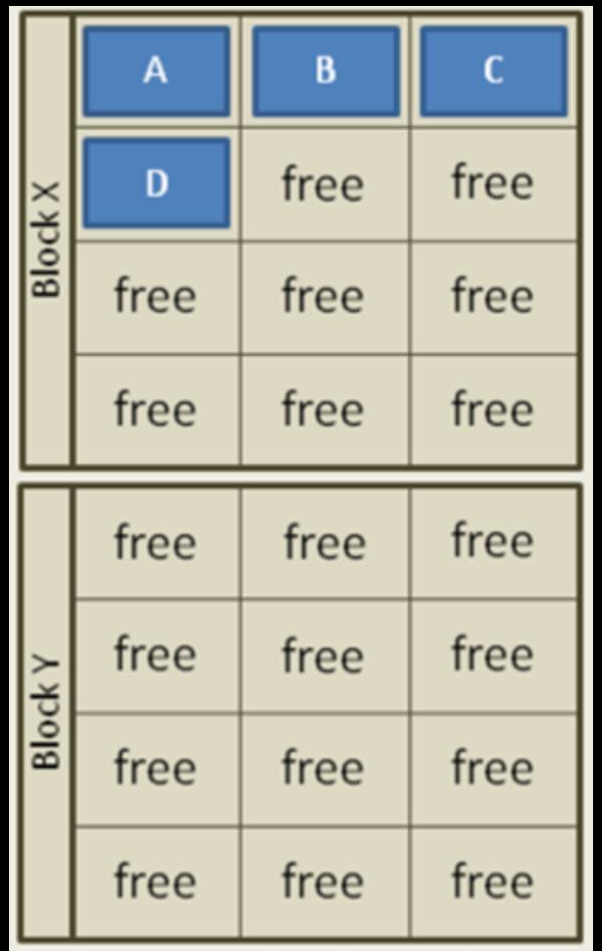
A file of **data** is stored on a medium (disk) as a **sequence of blocks**



Traditional disk drives store files as a **sequence of blocks**



Solid State Drives (SSD) store files as a **sequence of blocks**



1. Four pages (A-D) are written to a block (X). Individual pages can be written at any time if they are currently free (erased).

Why do we need the “Blocks” in addition to “Files”?

- File can be larger than a single disk
- Block is of fixed size, easy to manage and manipulate
- Easy to replicate and do more fine-grained load balancing

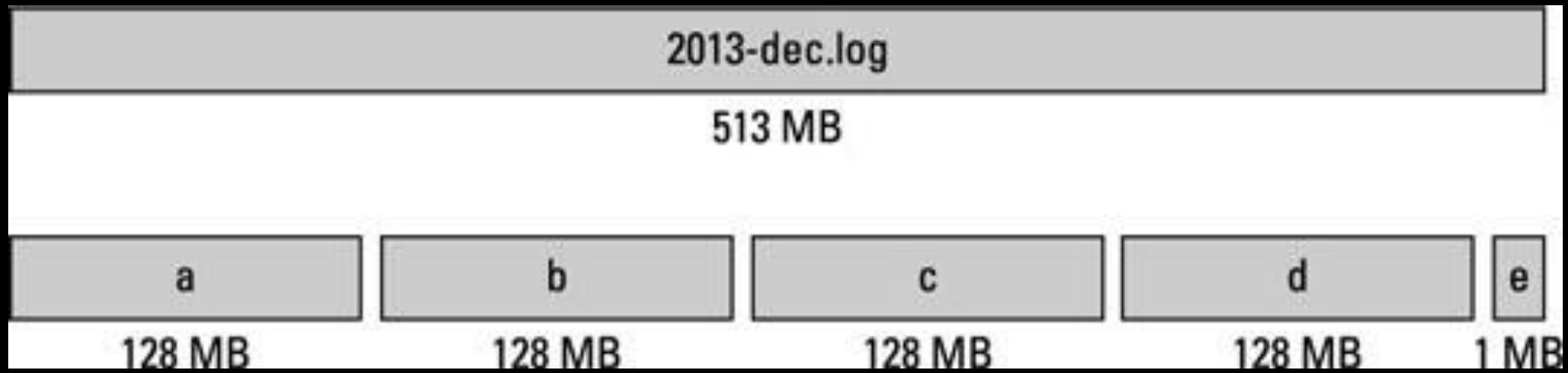
Why block size is much larger than regular (Windows) file system block?

- Large because of hardware constraints
  - Data seek (finding on disk) is slower than (large) data transfer
  - Assume seek time is ~10 ms (10/1000s = 1/100s) and transfer rate is ~100 MB/s
  - Then, to make seek time 1% of the transfer time (1% of one second),
  - we need to make the block size around 100 MB (transferred in a second), which is close to the default HDFS block size of 128 MB (64MB for Cloudera)
    - If seek time is too long, then may be nearly as quick just to look through all the data (no seek!)

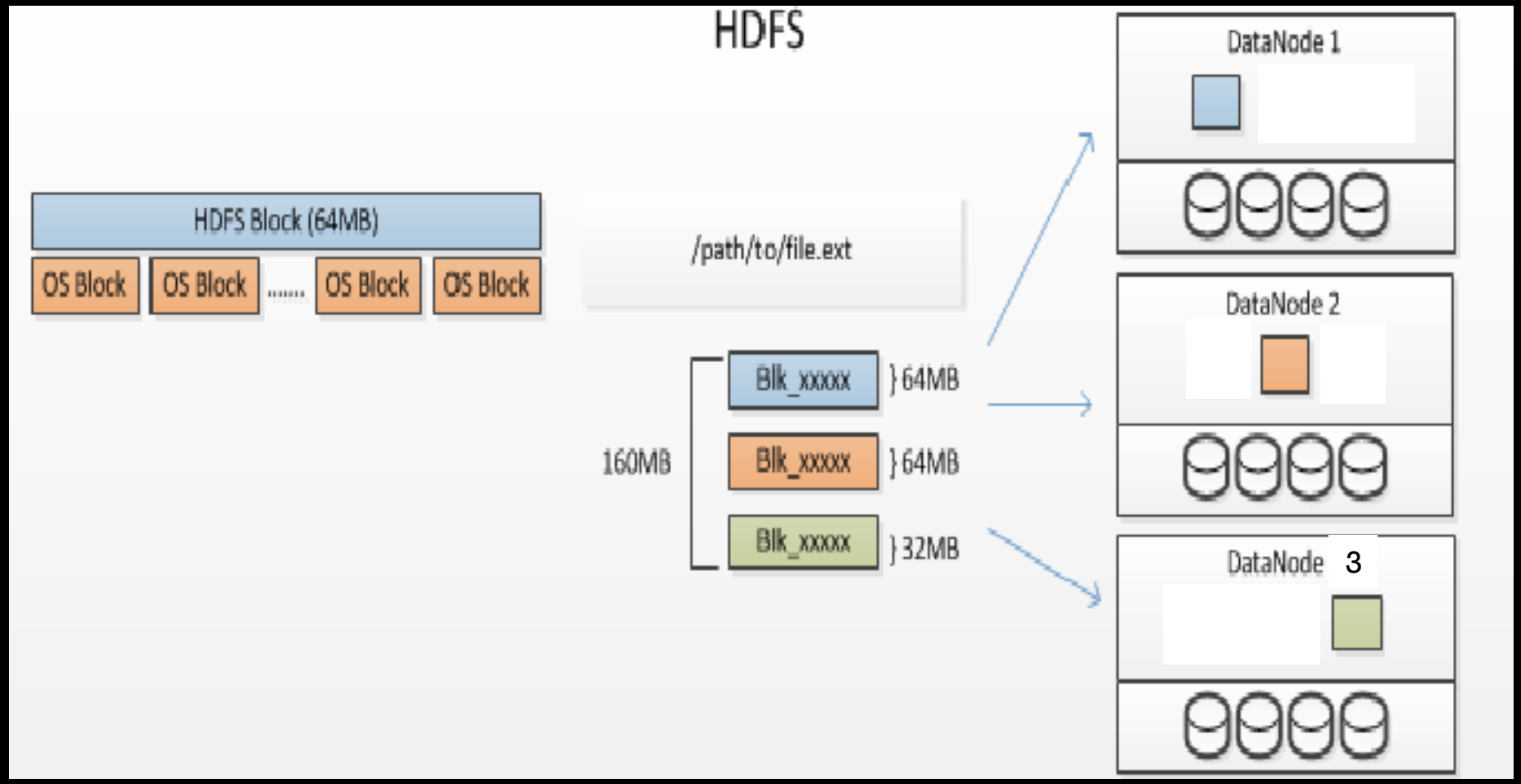
Using the concept of file **blocks**, how can a file of **data** be **distributed** over a network of data **nodes**?

HDFS key concept:
Distribute **file blocks over data nodes**

Given a file of **data**, divide it into a **sequence of blocks**, and **store the blocks**



The **blocks** of a file are **distributed** over the data **nodes**



What advantages does this **simple** block distribution provide?

Advantages that this **simple** block distribution provides

- When a node fails, only the data it has is lost (the rest is OK)
  - Fault tolerance
- Each node processes its data independent of the other nodes
  - Parallel processing
- Node access to its own data is local (and not via network access)
  - Fast data access

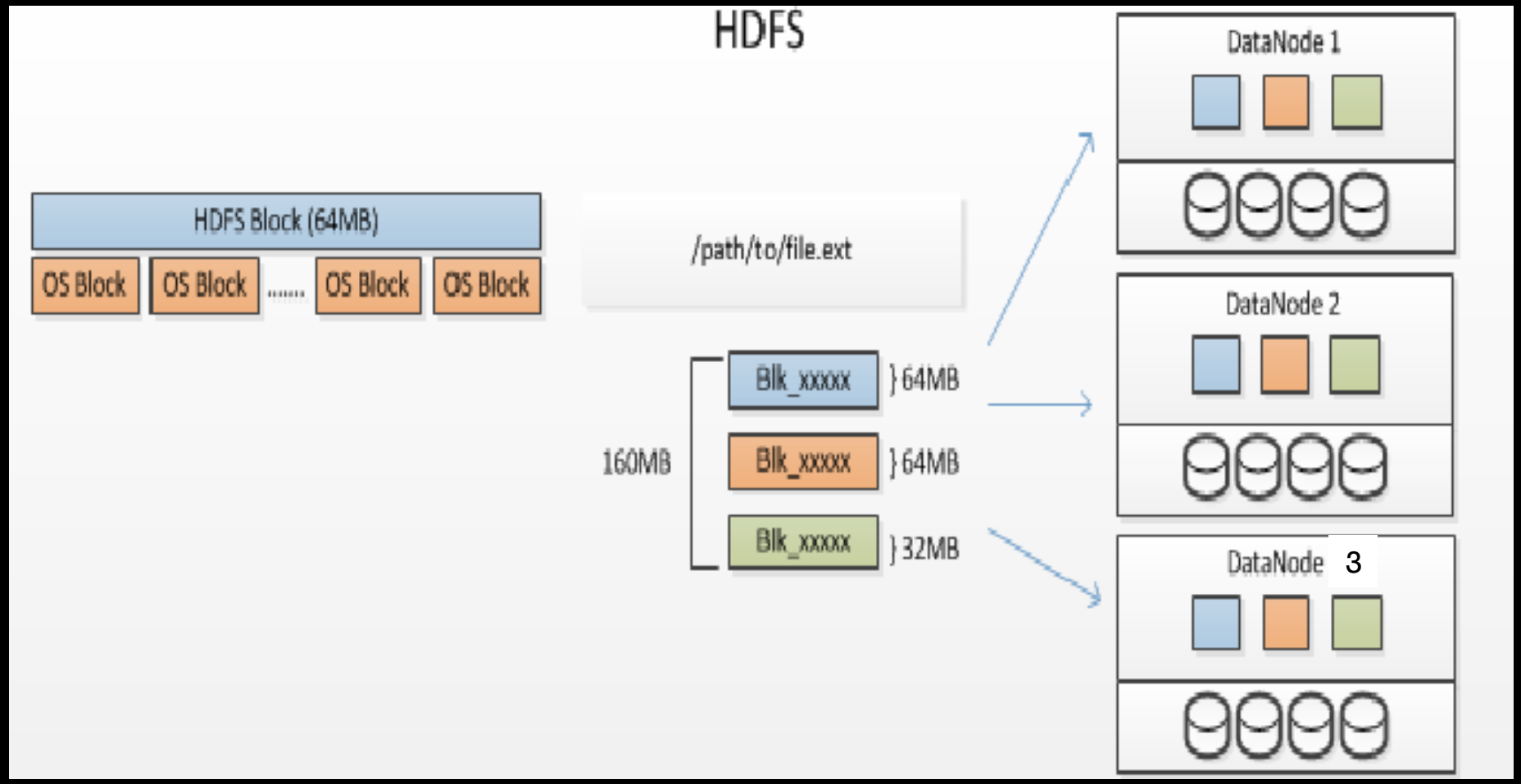
Can we **improve** this **simple** block distribution ?

Commodity hardware will fail!

- HDFS: Software is intelligent enough to handle hardware failure!

Let's provide **multiple copies** of the **blocks**
that are distributed

Each **block** has 3 **copies distributed** over the data **nodes**



What advantages does this distributed and replicated block system (HDFS) provide?

Advantages of this distributed and replicated block system (HDFS)

- When a node fails, the data can be found on the other replicated nodes
  - Fault tolerance
- Each node replica can process its data independent of the other nodes
  - Redundant, parallel processing
- Node access ideally is local, but can access replicas via the network
  - Fast data access
  - When a node's memory is full, a task can run from a nearby node, accessing the data over the network

Data locality

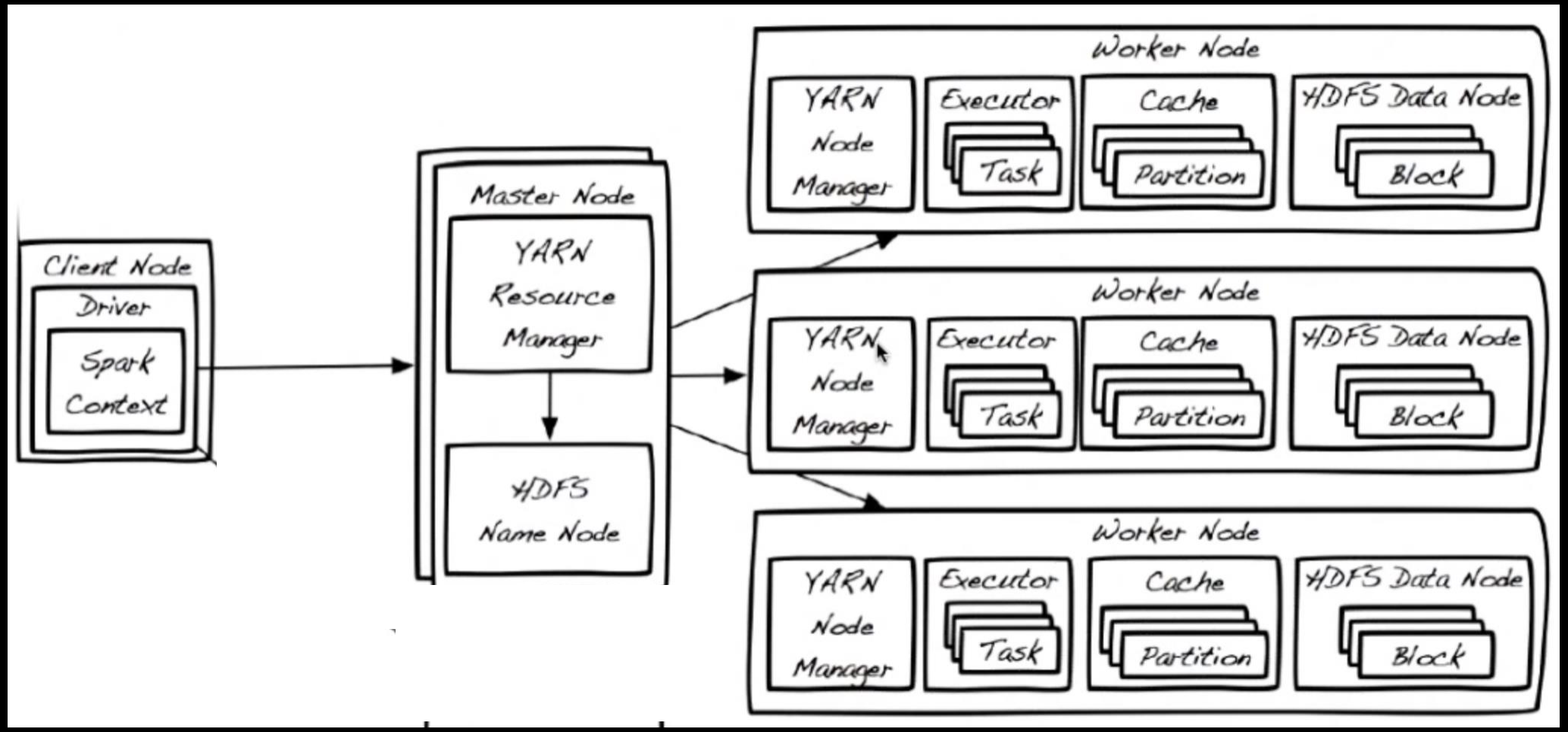
- Yarn and HDFS attempt to maximize data locality
- Move program copies (PySpark) to the data on the cluster
- Execute the program on a DataNode that has direct access to the data, which is locally stored
- Object stores (like Amazon's S3) do not address locality directly
  - HDFS can yield 6X higher read throughput than S3
  - However, S3 is less expensive
  - On Amazon
    - HDFS can run on S3
    - S3 is RACK_LOCAL to EMR spark clusters

https://databricks.com/blog/2017/05/31/top-5-reasons-for-choosing-s3-over-hdfs.html; https://stackoverflow.com/questions/44304104/s3-and-emr-data-locality

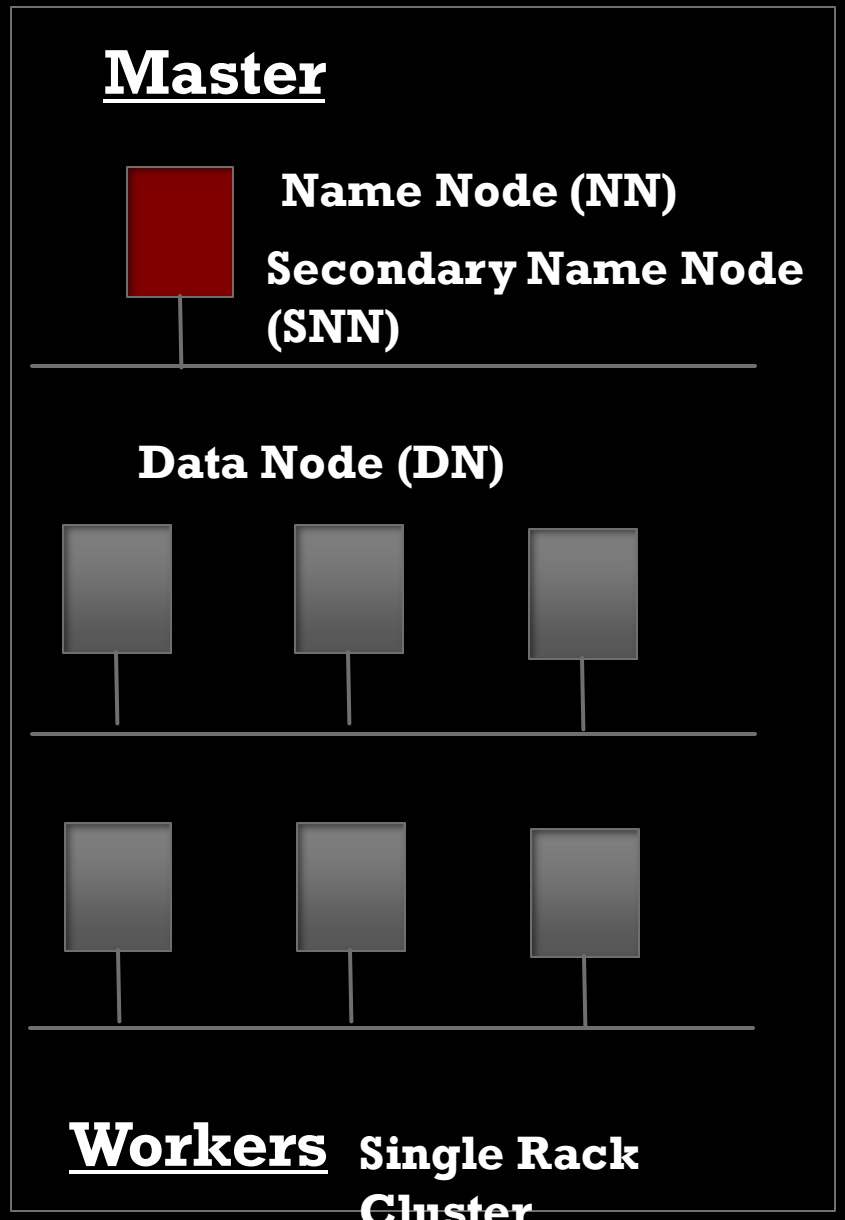
What's the HDFS architecture?

It works with Yarn

Hadoop Master/Worker Architecture



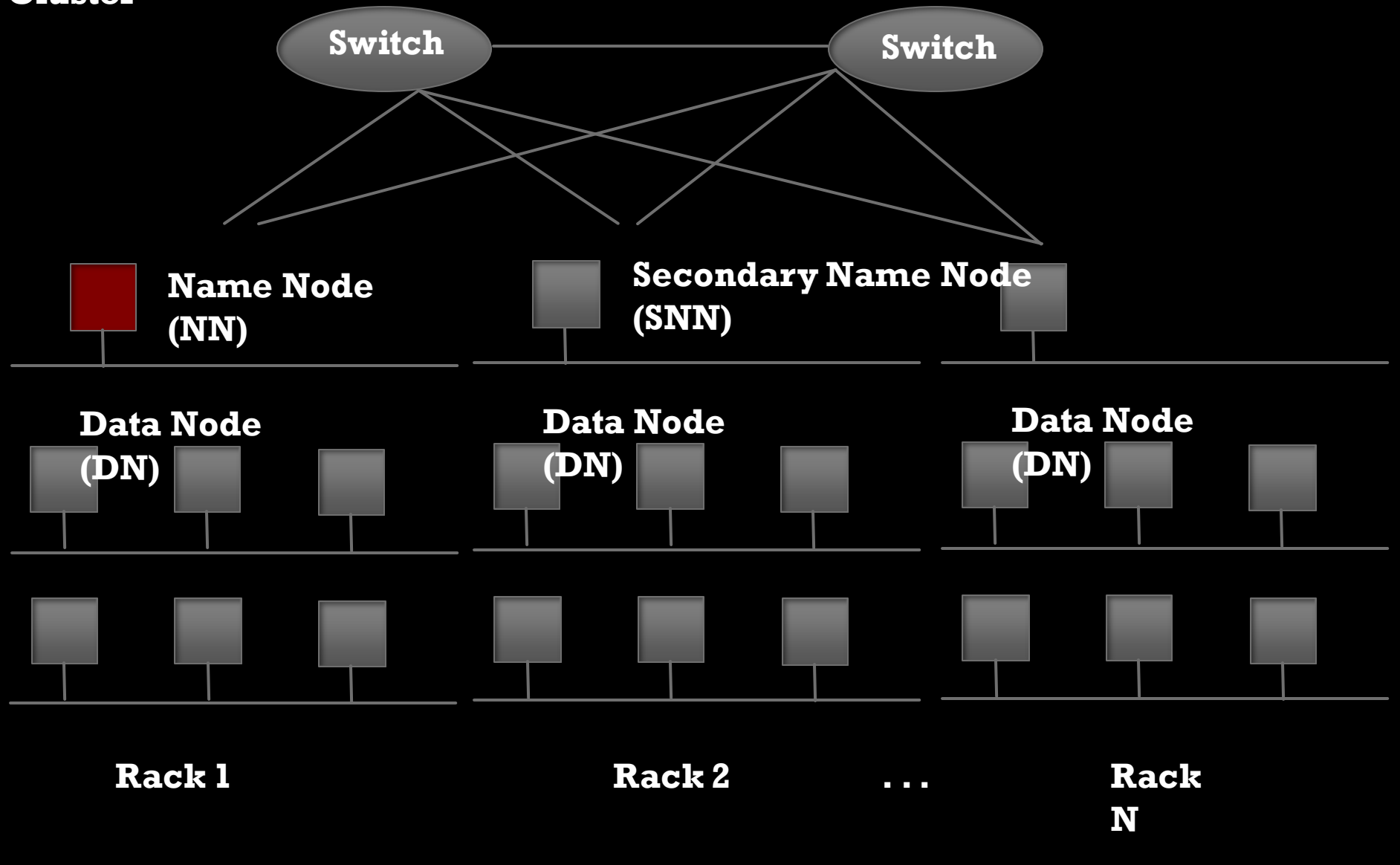
Master-Worker



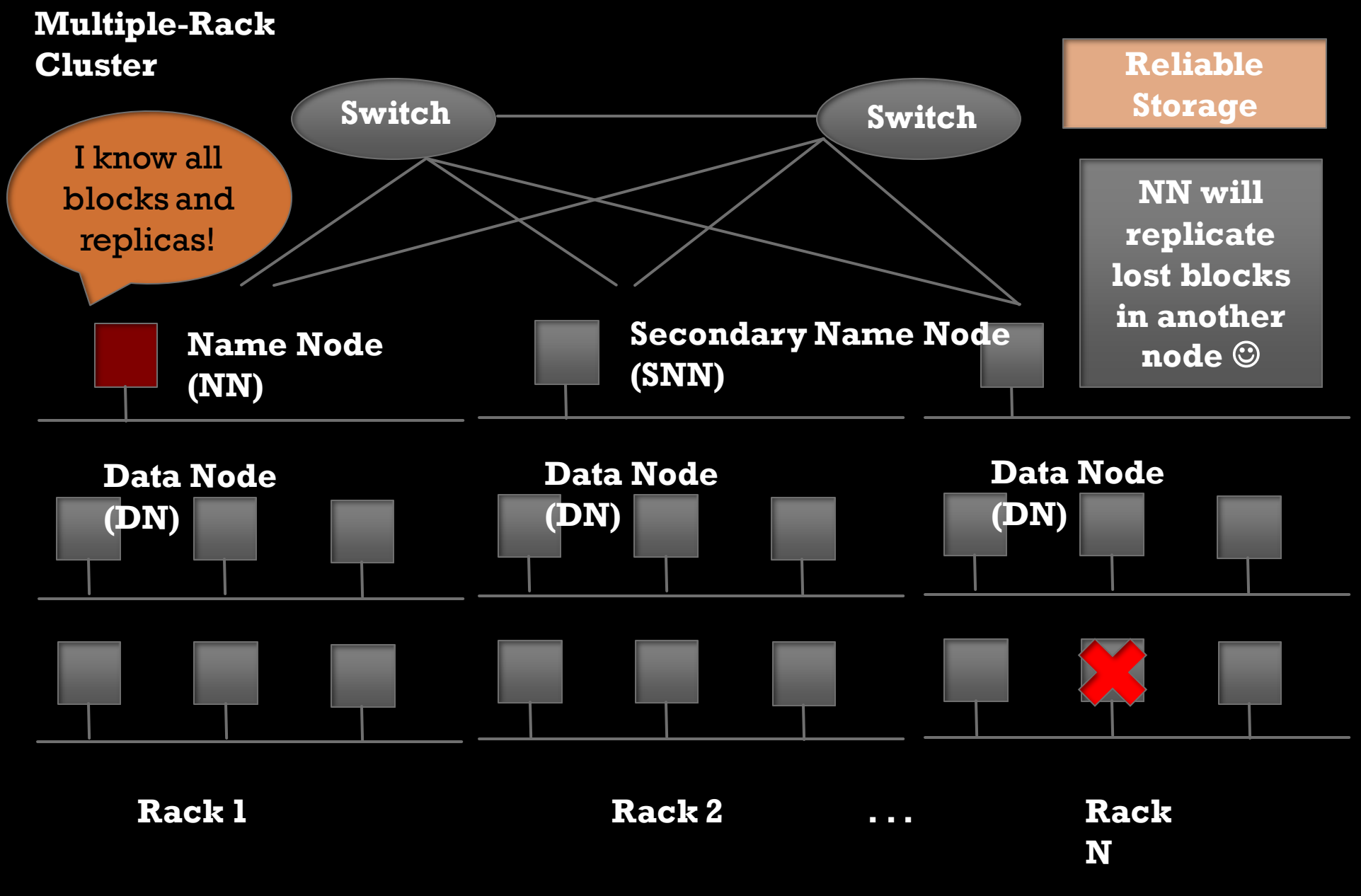
- Name Node: Controller
  - File System Name Space Management
  - Block Mappings
- Data Node: Work Horses
  - Block Operations
  - Replication
  - A block processes with one task
- Secondary Name Node:
  - Checkpoint node

Master-Worker Architecture

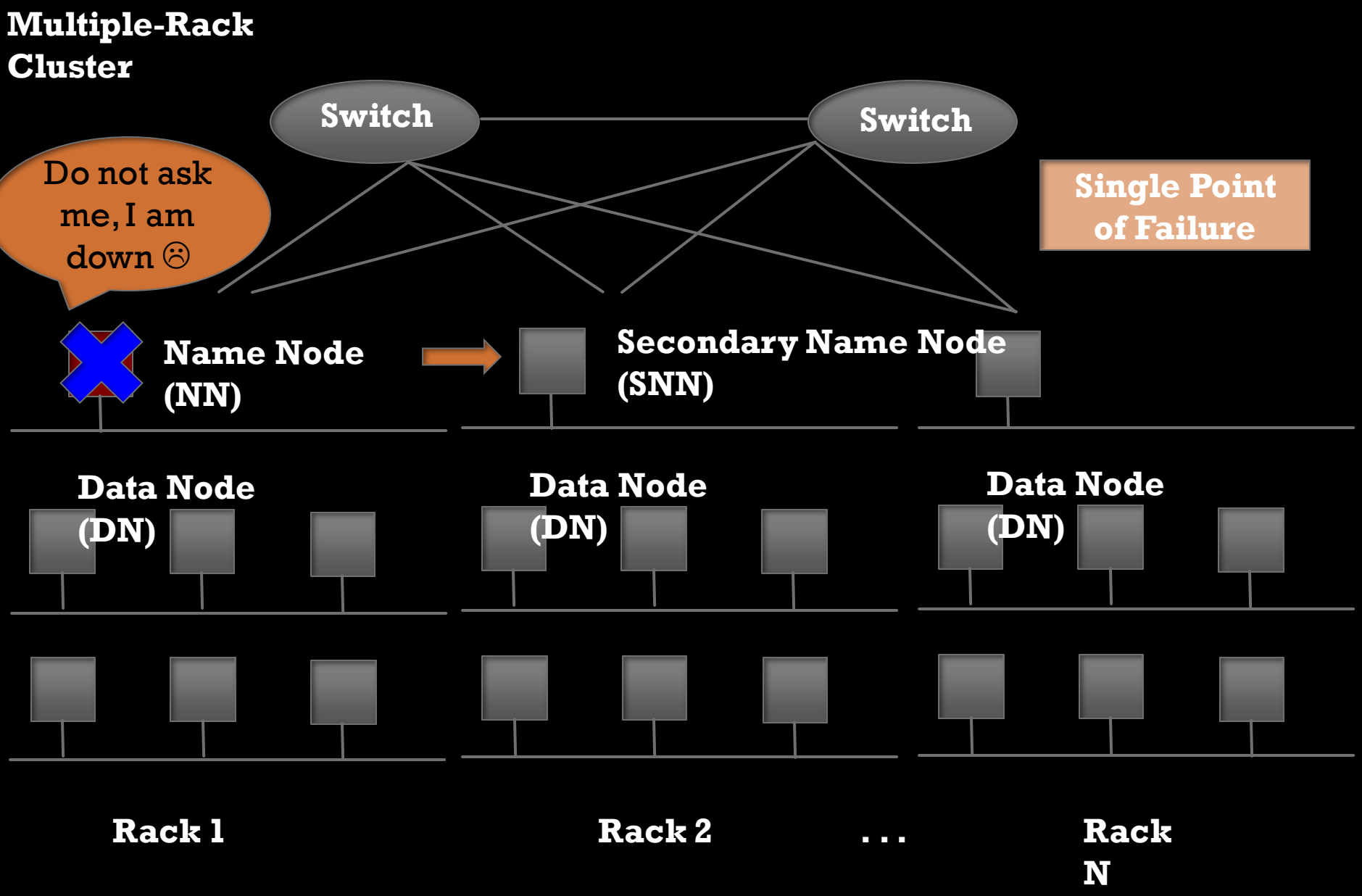
Multiple-Rack Cluster



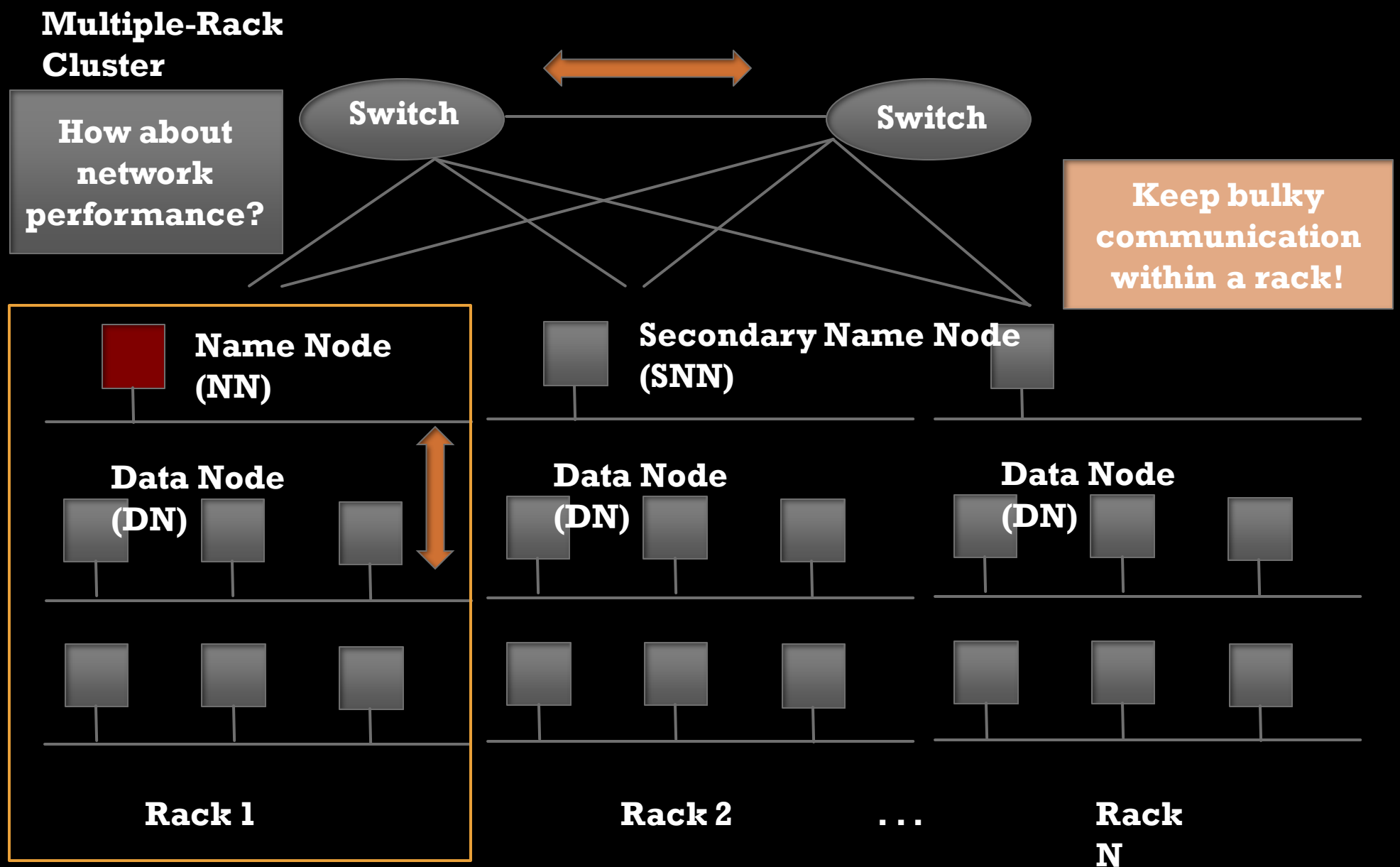
Master-Worker Architecture



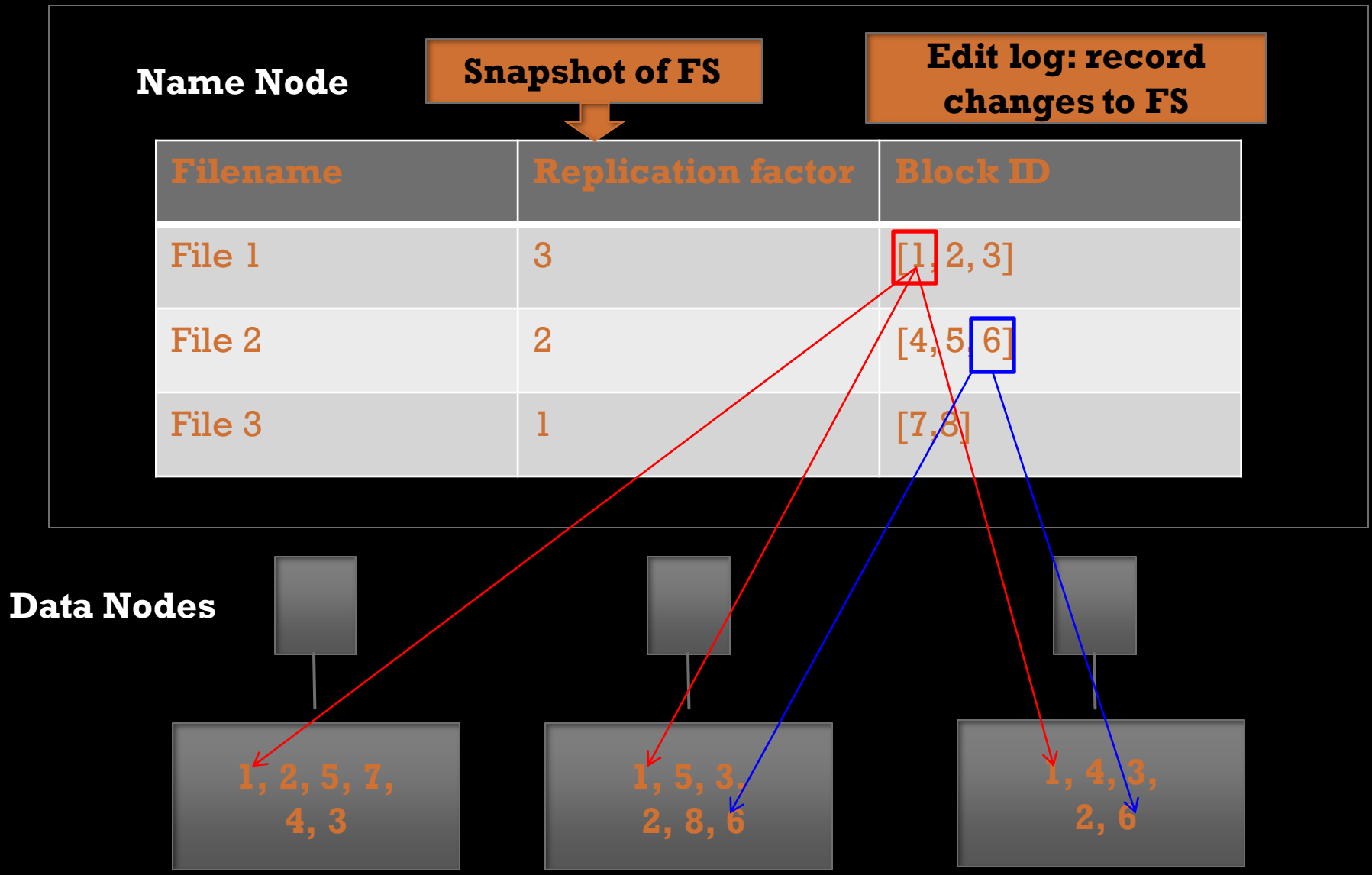
Master-Worker Architecture



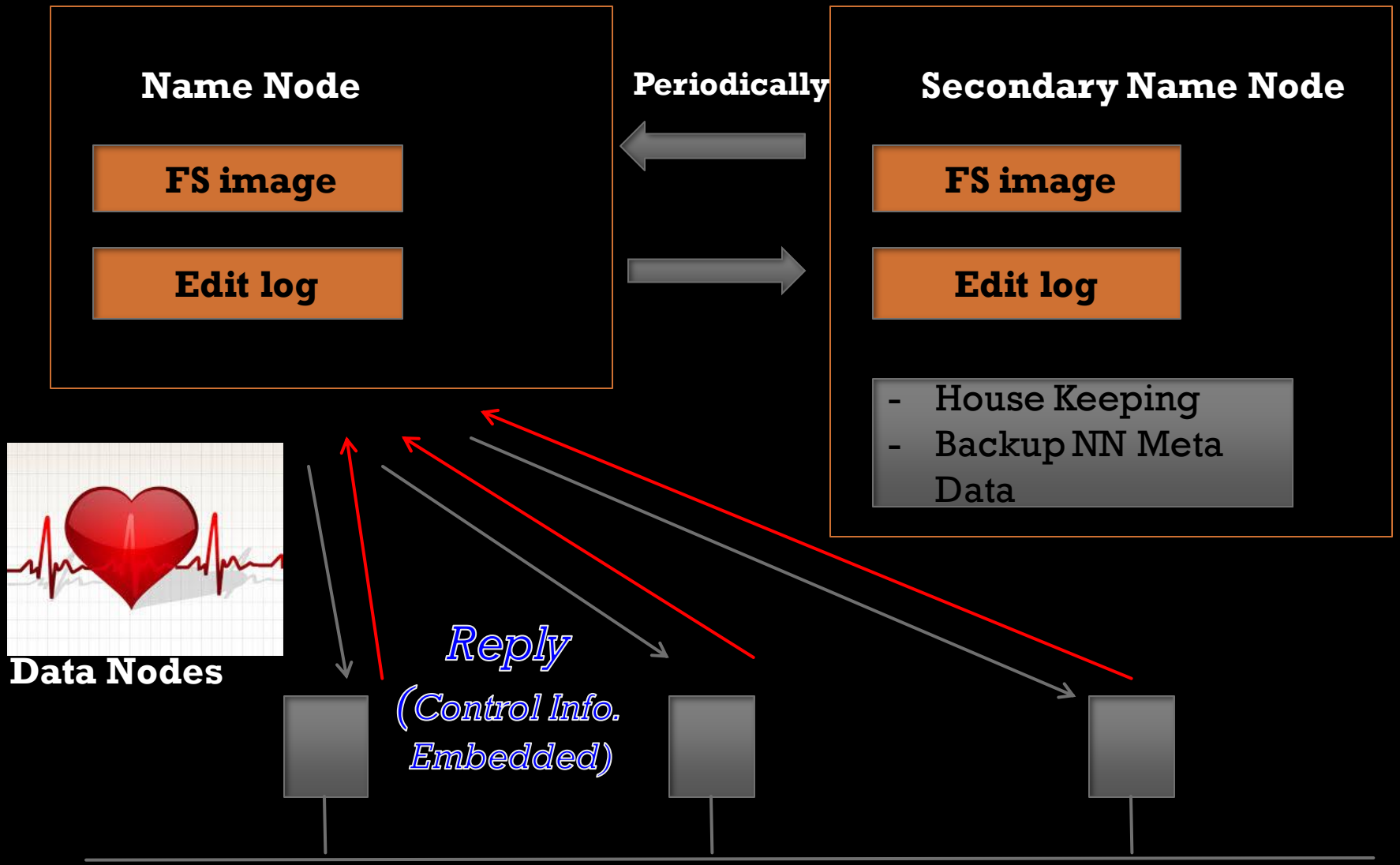
Master-Worker Architecture



Name Node HDFS Meta-Data

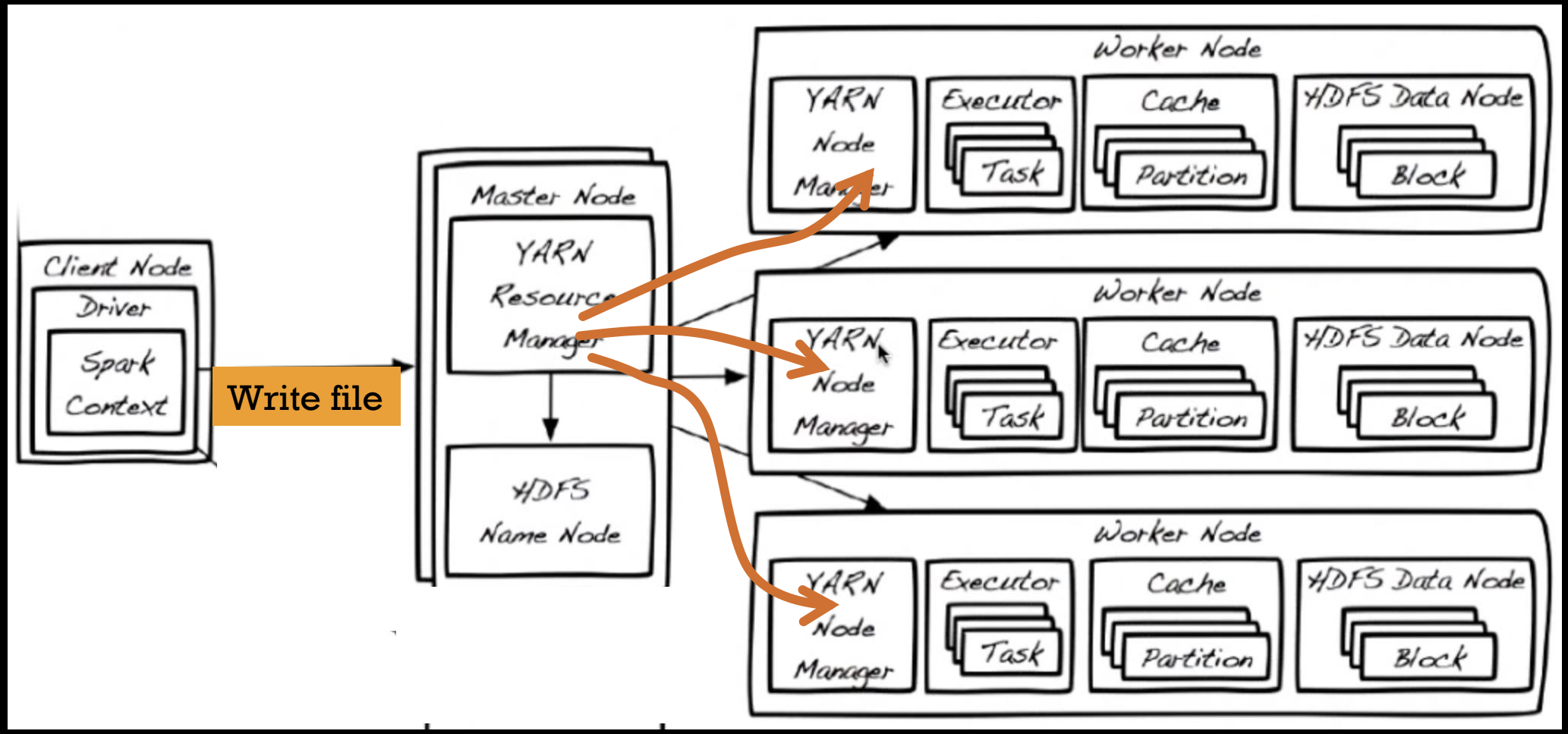


Name Node: Worker Heart Beat



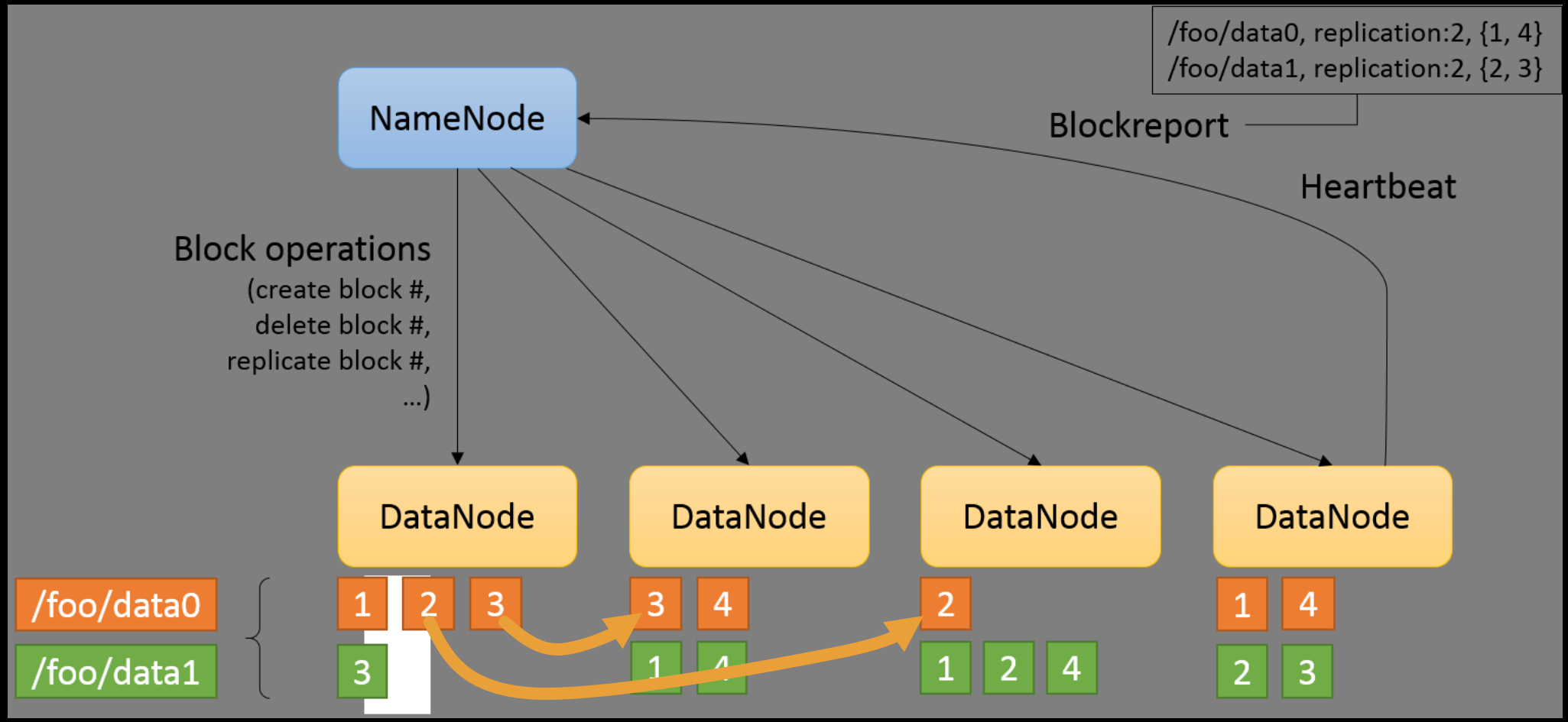
How are the file **blocks written** to the data network?

If the Resource Manager does all the writes, then it will be a bottleneck (for the many clients)

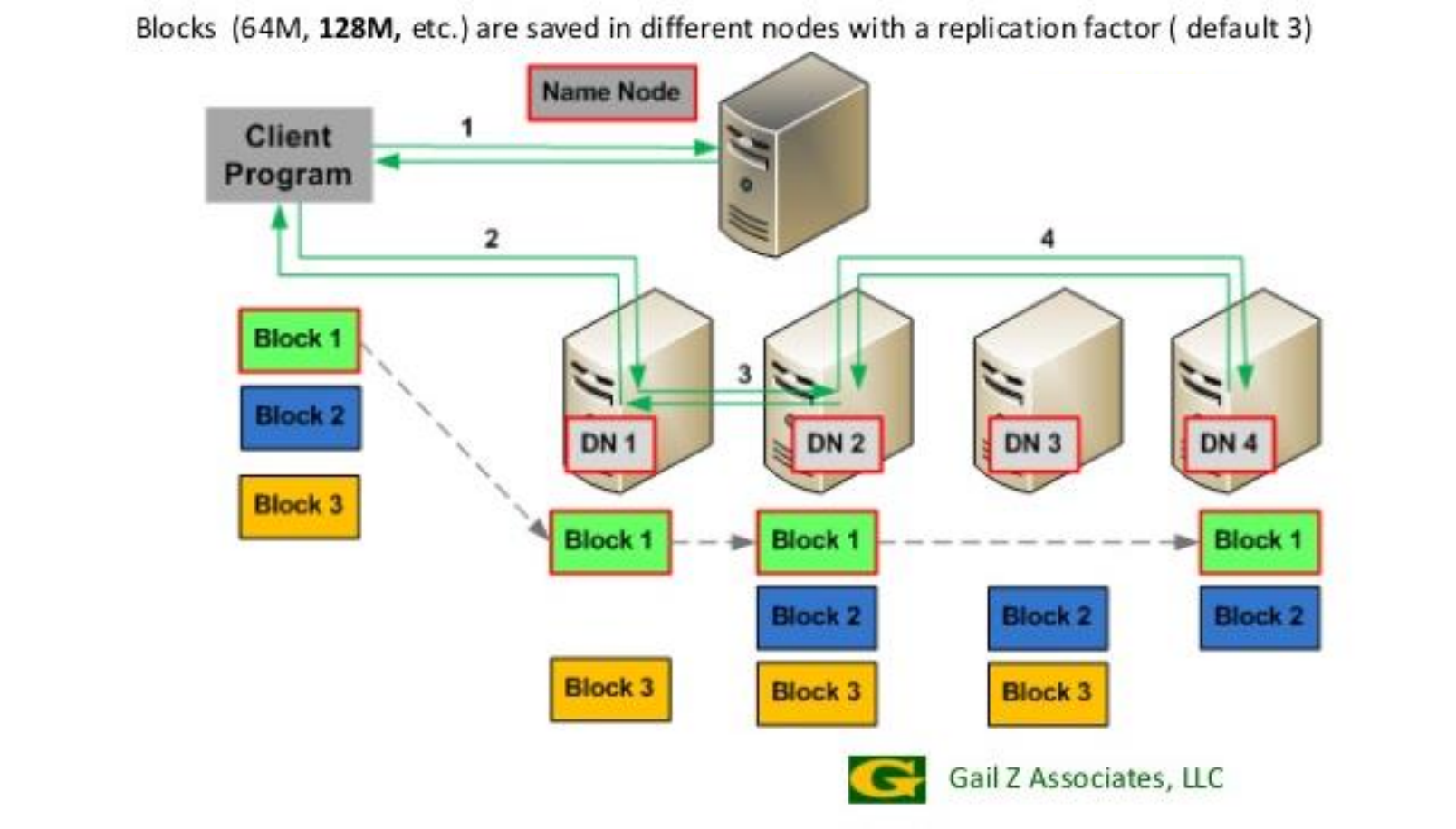


Replication

Data node sends copy to next data node, according to Name Node plan



Replication
Data node sends copy to next data node, according to Name Node plan

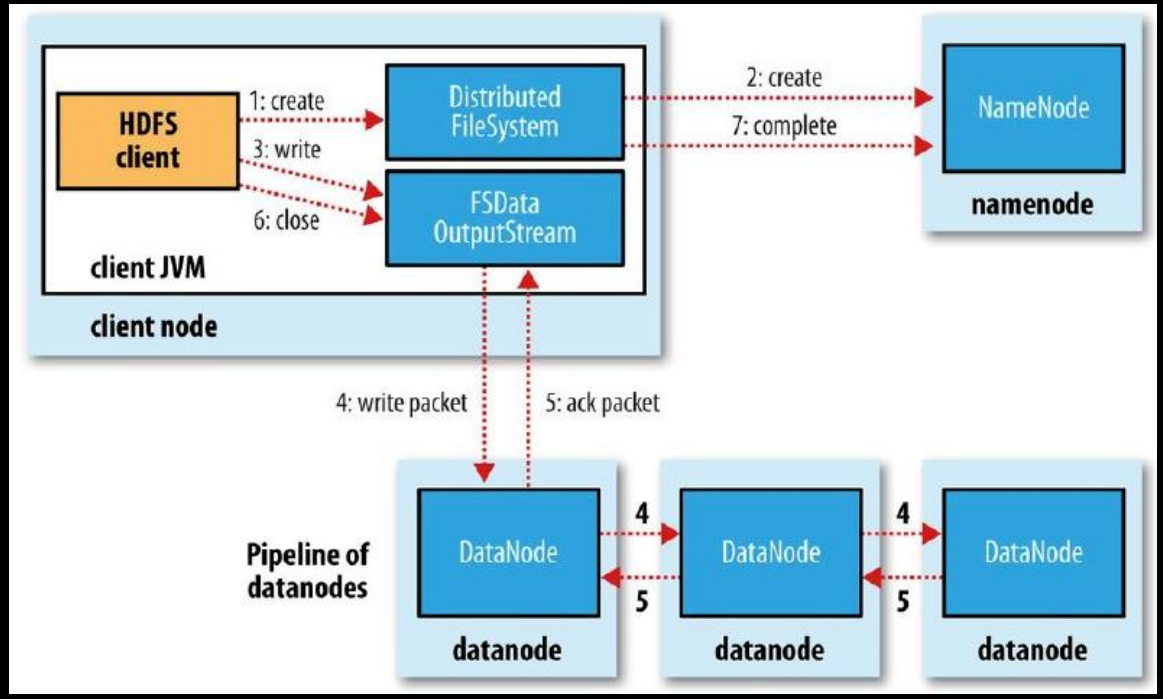


File Write

3: DFSOutputStream splits it into packets,

4…: DataNodes forward packets onto other DataNodes for replication

If write failure, then NameNode is informed, and it schedules latter replication



How are the file **blocks read** from the data network?

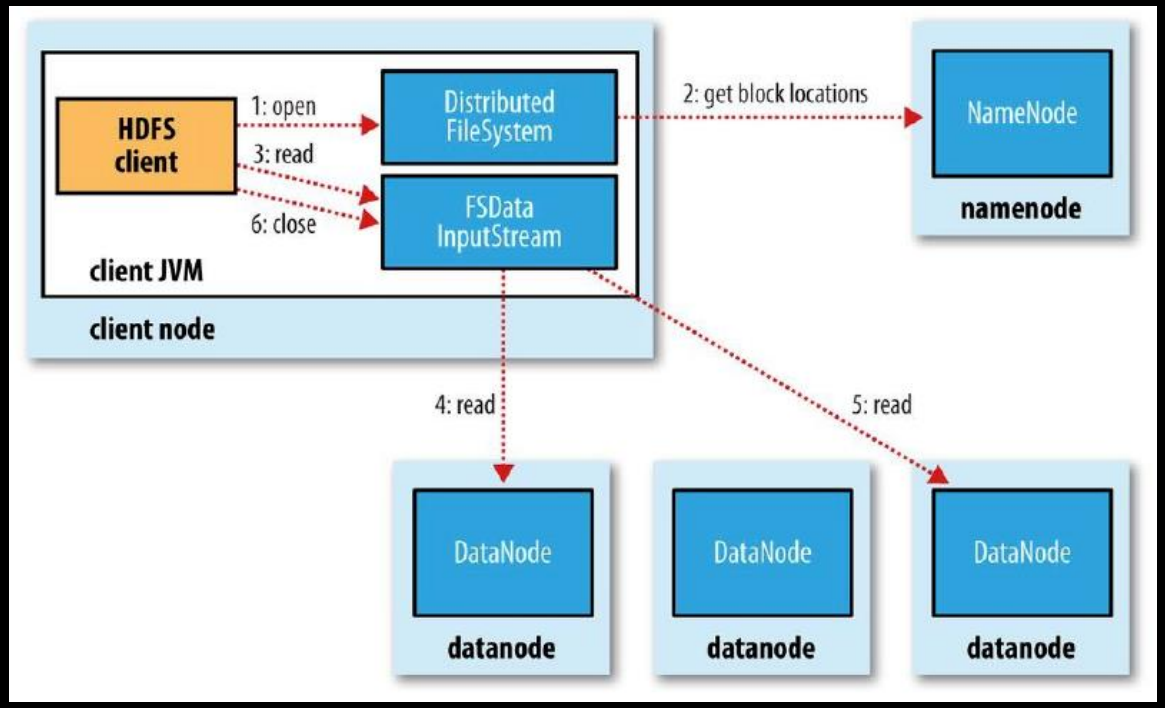
File Read

2: block locations returned in order of proximity to client

3: data is streamed from the **nodes to the client** (managed by the DFSInputStream object)

4,5…: reads occur on data stream

DFSInputStream handles faults by getting next replicated block



Why not ask client to read blocks through Name Node?

- Prevents NN from being the bottleneck of the cluster
- Allow HDFS to scale to large number of concurrent clients
- Spread the data traffic across the cluster

Where should the **block replicas be stored**,
to ensure efficiency and fault tolerance ?

Place **replicas**

close to client (which initiates processing)

close to other replicas

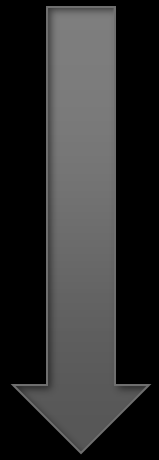
Both of these schemes **reduces network traffic**

Files consist of block sequences
Entire file will be processed
Block processed **results will be combined**

So, **blocks should be close together**
But, blocks should be on separate racks
(in case of failure)

HDFS bandwidth is a critical resource
(Estimate real bandwidth by distance)

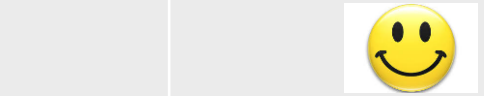
- Slower with network distance
  - Processes on the same node
  - Different nodes on the same rack
  - Nodes on different racks in the same data center (cluster)
  - Nodes in different data centers



Bandwidth becomes less

Where to place replicated blocks?

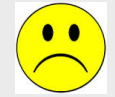
- Replication strategy tradeoffs

	Reliability	Write Bandwidth	Read Bandwidth
Put all replicas on one node			
Put all replicas on different racks			

Node failure or read contention is costly

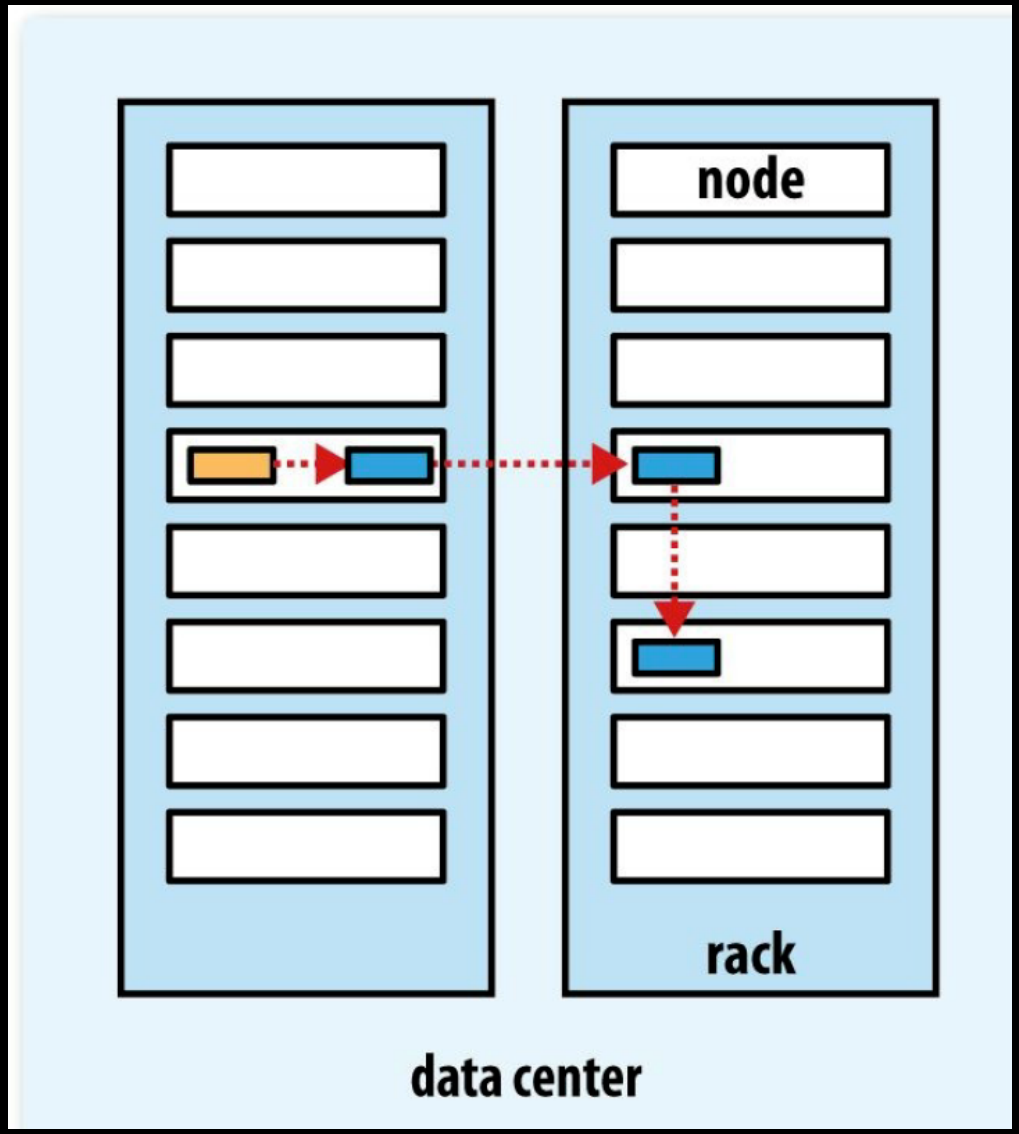
Network data transfer is costly

- Replication strategy tradeoffs

	Reliability	Write Bandwidth	Read Bandwidth
Put all replicas on one node			
Put all replicas on different racks			
HDFS: 1-> same node as client 2-> a node on different rack 3-> a different node on the same rack as 2			

Block replication

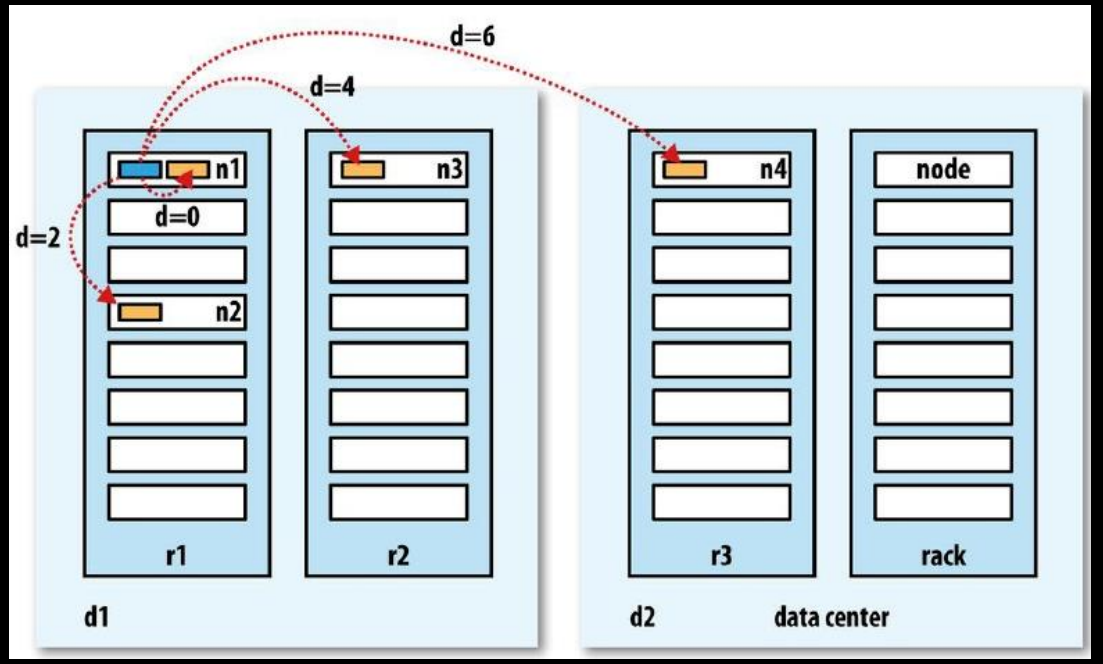
- Replication is typically:
  - Rack x (of client or at random)
  - Rack y (at random)
  - Rack y, but another node of rack y



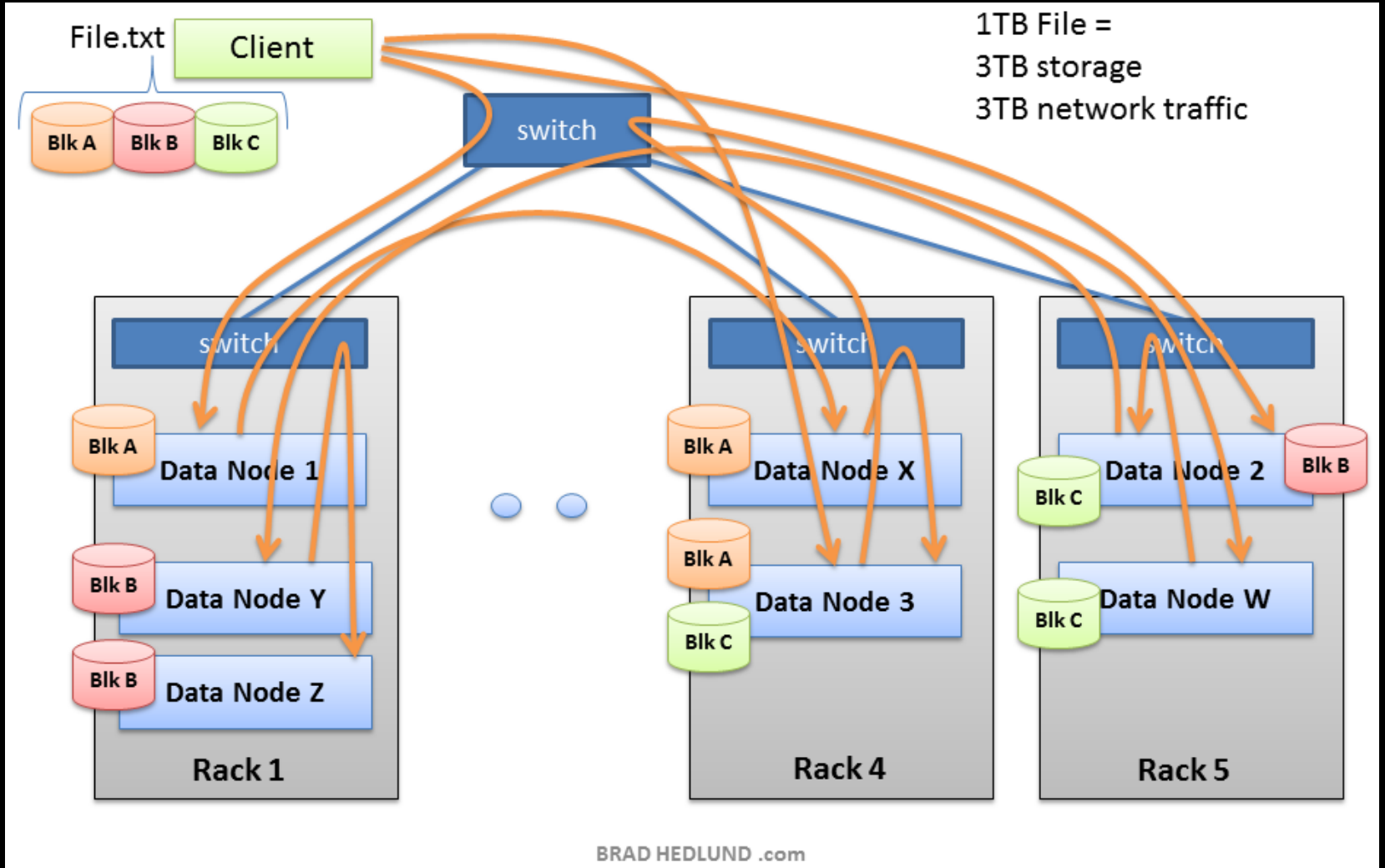
Exam question

Network aware **block distance metric**

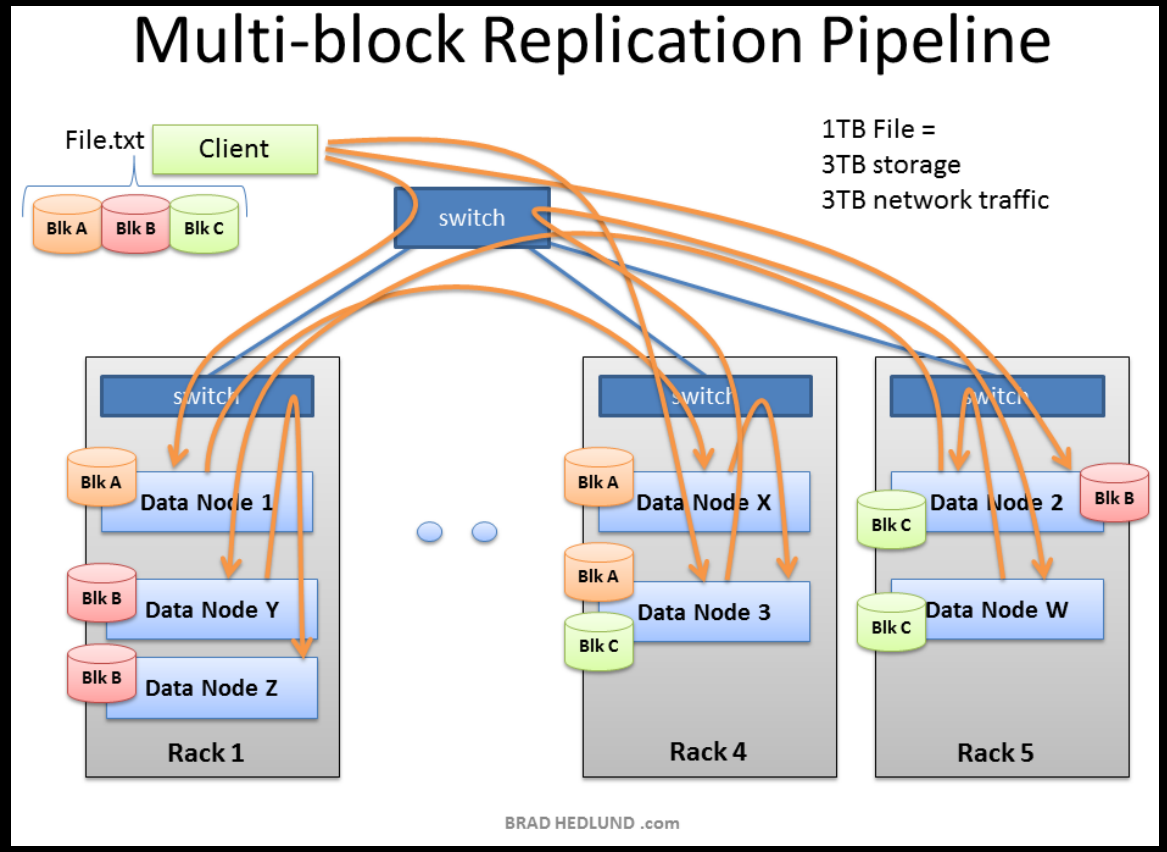
- distance(/d1/r1/n1,/d1/r1/n1) = 0
  - processes on the **same node**
- distance(/d1/r1/n1,/d1/r1/n2) = 2
  - different nodes on the **same rack**
- distance(/d1/r1/n1,/d1/r2/n3) = 4
  - nodes on **different racks** in the same data center
- distance(/d1/r1/n1,/d2/r3/n4) = 6
  - nodes in **different data centers**



File blocks: both same & different racks



- Transfers through switch (of rack) are slower
- Results of processing a block are often combined, so have blocks nearby
  - Blocks A & B can be combined within rack
  - Blocks B & C could be combined within rack



How do **tasks execute** over the data network?

MapReduce is the original method for executing over HDFS

Map Reduce over HDFS

1. Client requests Resource Manager execute a job
2. Resource Manager
    1. Obtains file block locations (meta-data)
    2. Computes input splits
        1. entire file for small files, otherwise (file/block-size) splits
    3. Requests Data Node Manager start job (with specified resources)
3. Application Manager (in data node)
    1. Starts a map task for each split, and some reduce tasks
        1. Map task placed close to its data (according to meta-data)
    2. Monitors execution, restating tasks as necessary

Both Spark and Map Reduce can use YARN, so the task distribution nearly the same

MapReduce Execution

1. Client (your program) submits job to (Yarn) RM

5a. RM plans resources, starts application in NM

5b. NM starts AM, which manages tasks created in NM's

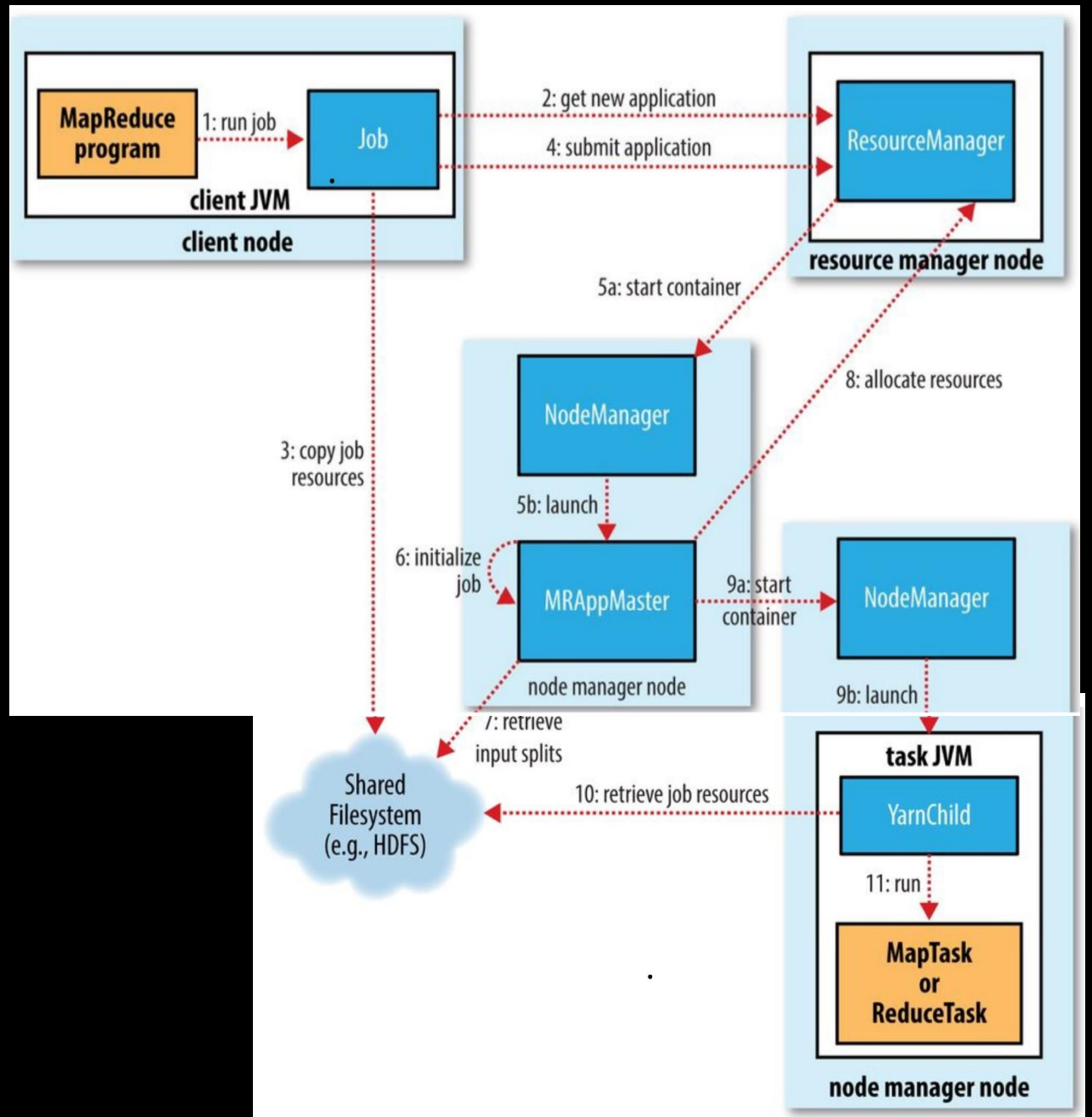
9b. YarnChild preps resources and runs (map or reduce) task

12. Status is sent back to each manager

YarnChild -> AM -> RM -> client

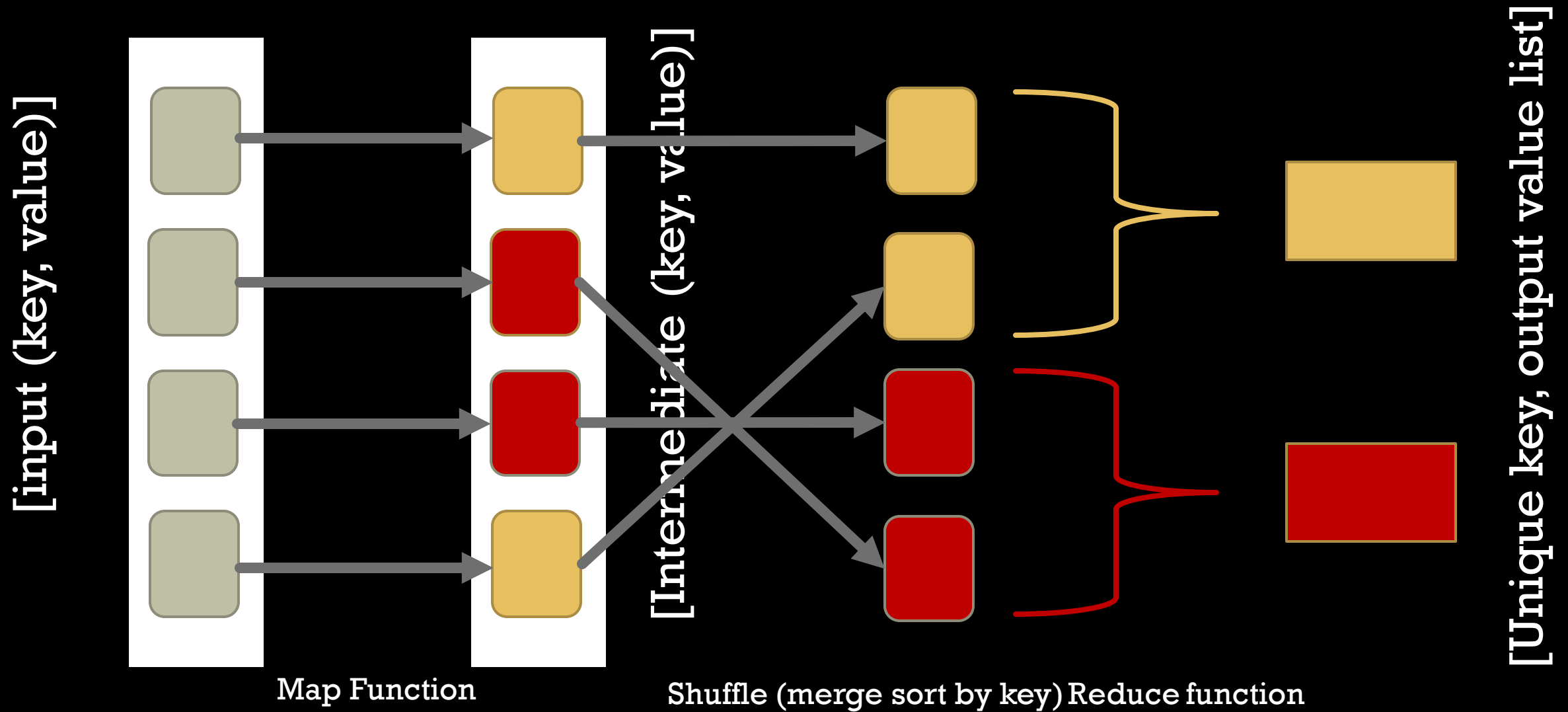
Restarts occur at various levels as necessary

Yarn is this RM, thus the AM and YarnChild are Yarn components



Classic Map Reduce example

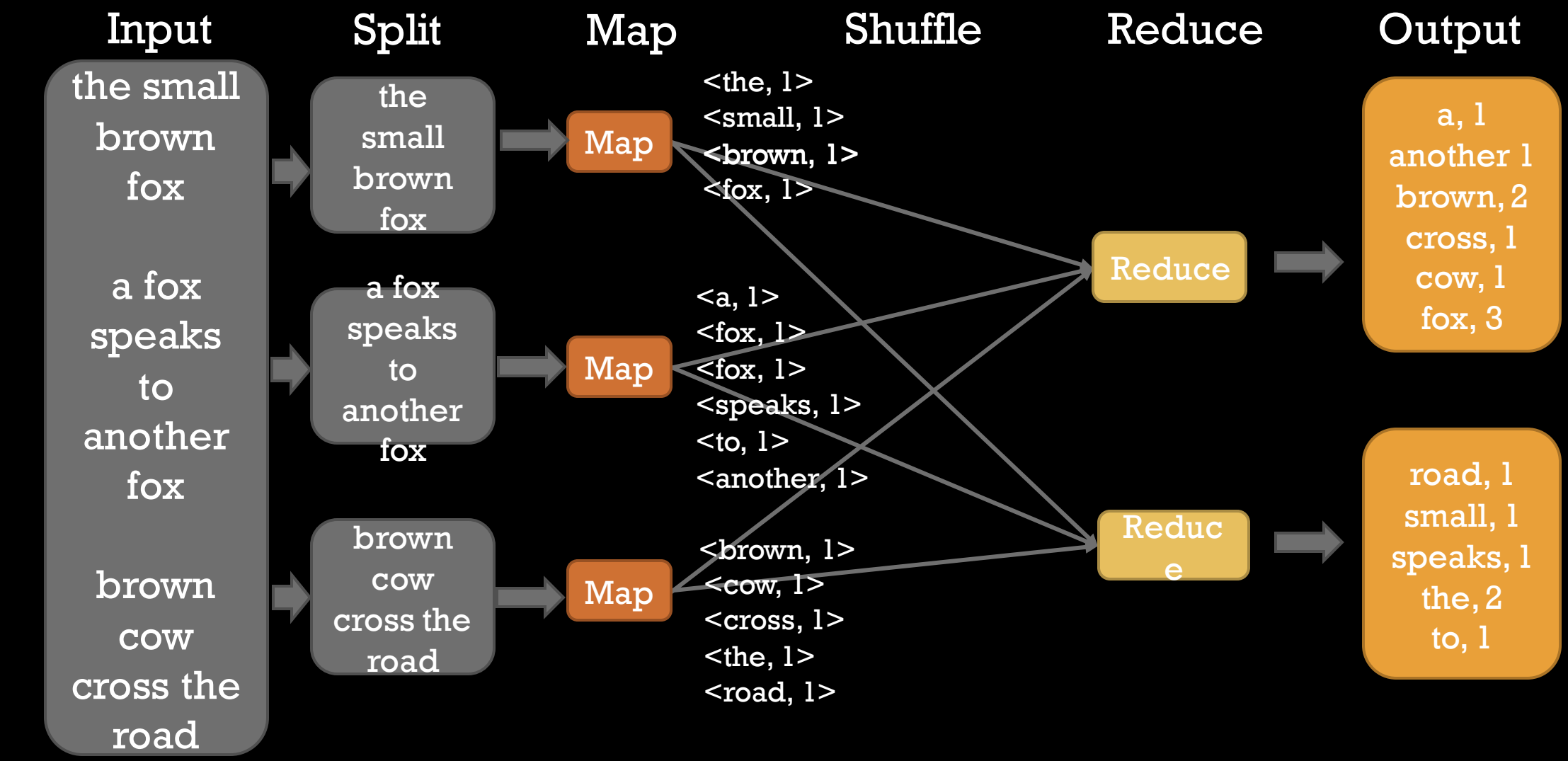
Programming Model



Example: WordCount

Block split by lines of text

Combine data blocks



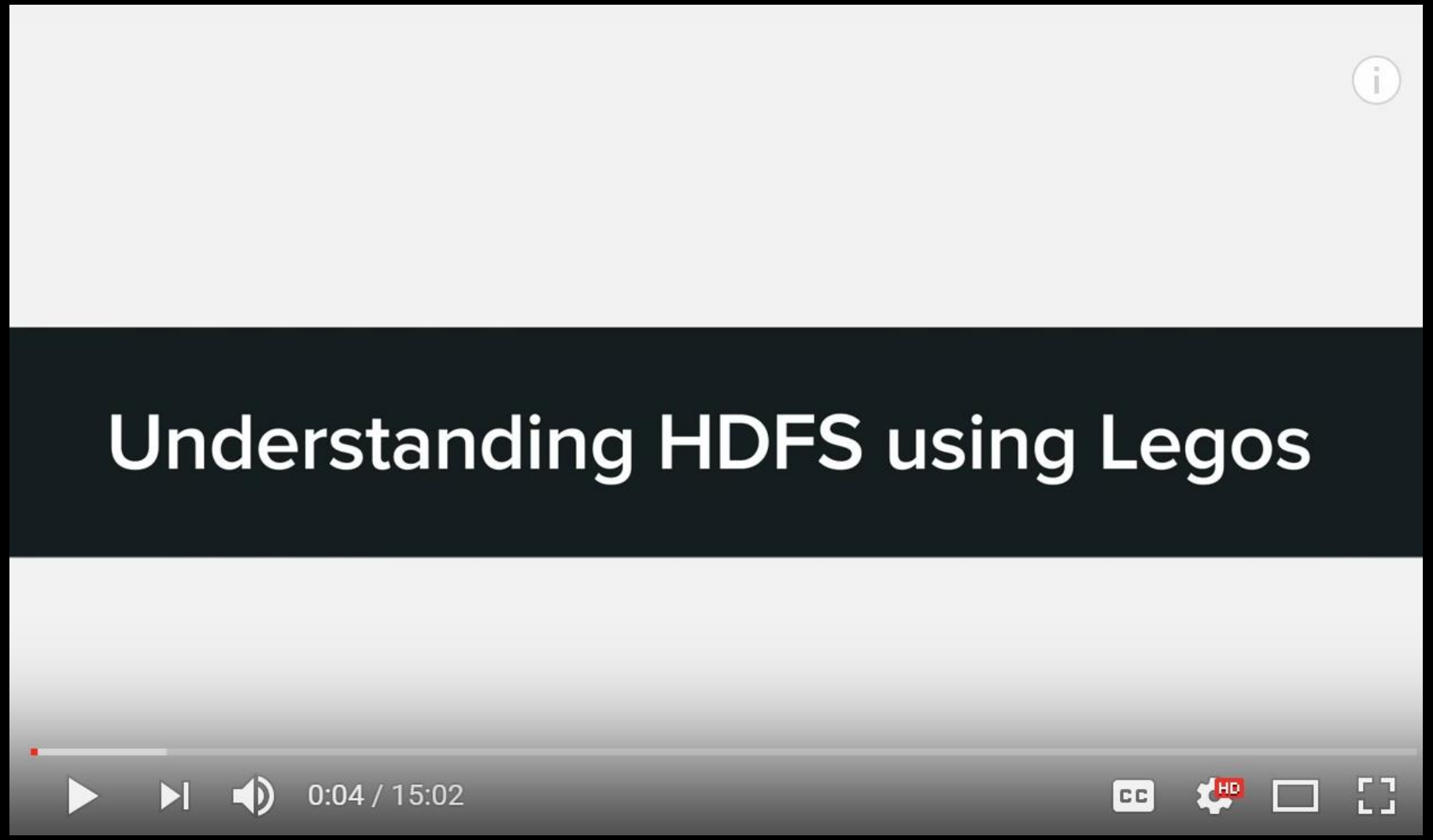
Summary

HDFS (HaDoop File System)

- Replicates portions of a file (blocks) over cluster
- Automatically
  - Recovers from data failures
  - De/Compresses files (when configured to do so)
- HDFS is a core component of Hadoop
  - MapReduce and most other components use it
  - However, can substitute other files systems like Amazon S3

Object storage (S3) is very popular alternative. https://www.druva.com/blog/object-storage-versus-block-storage-understanding-technology-differences/


BIG DATA REPUBLIC
TRANSFORM YOUR BUSINESS WITH DATA
www.bigdatarepublic.com
@bigdatarepublic
facebook.com/bigdatarepublic
0:01 / 1:05
CC
HD



Key is: 2:00m – 9:00m