CSP554
BIG DATA TECHNOLOGIES

Module 2b

Hadoop Distributed File System (HDFS)

Distributed File Systems

- When a dataset outgrows the storage capacity of a single physical machine or storage device…
- It becomes necessary to partition it across a number of separate machines or devices
- File systems that manage storage across a network of machines are called distributed file systems
- Since they are network based, all the complications of a cluster kicks in
  - So distributed file systems are more complex (subject to more failure modes) than centralized ones
  - The file system must tolerate node, storage and other failures without suffering data loss

HDFS

- Hadoop comes with a standard distributed file system implementation
  - Hadoop Distributed File System
  - Provides scalable, fault-tolerant, rack-aware data storage
- But Hadoop supports a general purpose file system abstraction (API)
  - Allows Hadoop to use other file system implementations
  - Examples
    - Amazon Web Services S3 Buckets
    - Microsoft Azure Blob Storage
  - Offers a range of powerful but nonstandard functionality

HDFS Characteristics

- Designed for storing very large files reliably
  - Gigabytes to Terabytes in size
- Provides those data sets at high bandwidth to distributed user applications
- These capabilities both are achieved by replicating file contents on multiple machines
- Optimized for write once, read many access pattern
  - No updates to already written data
  - But, appends to files are permitted
  - Useful for analysis of large and sometimes growing datasets of mostly unchanging records

HDFS Characteristics

- Assumes use of commodity hardware (or cloud VMs) for which the chance of failure is (relatively) high
- Designed to carry on working with no loss of data or interruption noticeable to the user in the face of such failures
- High throughput streaming data access
  - Applications requiring low latency (10ms) access to data will not work well with Hadoop
  - Applications that require high data read performance (throughput) are exactly those for which HDFS is optimized
  - In memory caching and other advanced techniques may offer improved latency over time

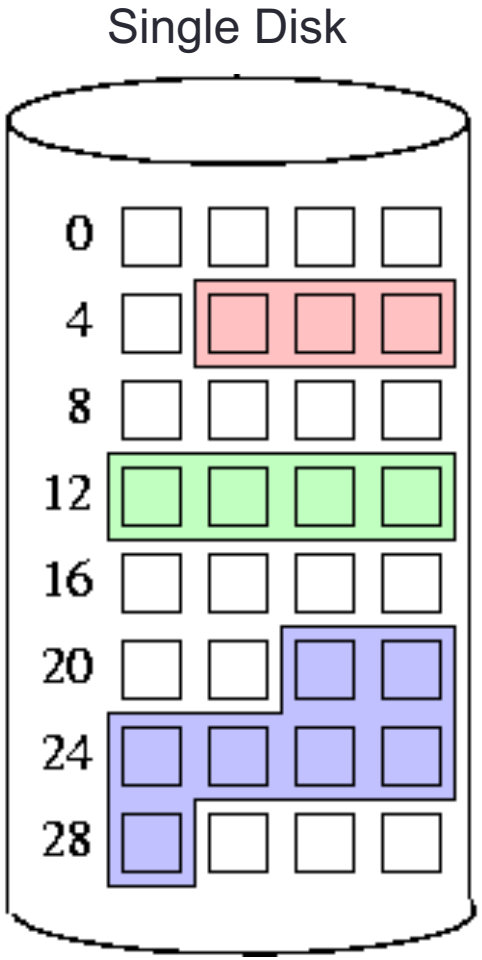
HDFS Characteristics

- Offers good performance for a relatively small number of large files and not many small ones
  - File system metadata is stored in memory and not distributed across nodes
  - Puts an upper bound on the total number of files about which information can be stored
  - Recent work supports having such metadata split across multiple machines
  - Allows more files and also improve metadata lookup performance

HDFS Concepts

- Blocks and Standard (Centralized) File Systems
  - Disks and the file systems which use them divide up space into fixed sized groupings of bytes
  - When a file is written to disk it is split by the file system into the blocks needed to hold all data
  - The file system reserves some space on each disk to record metadata of which blocks store data for which files
  - Disk physical blocks and block sizes that most file systems impose range up to some kilobytes

Blocks and Standard (Centralized) File Systems



Directory:

file	start	length
moo	5	3
snow	22	7
fall	12	4

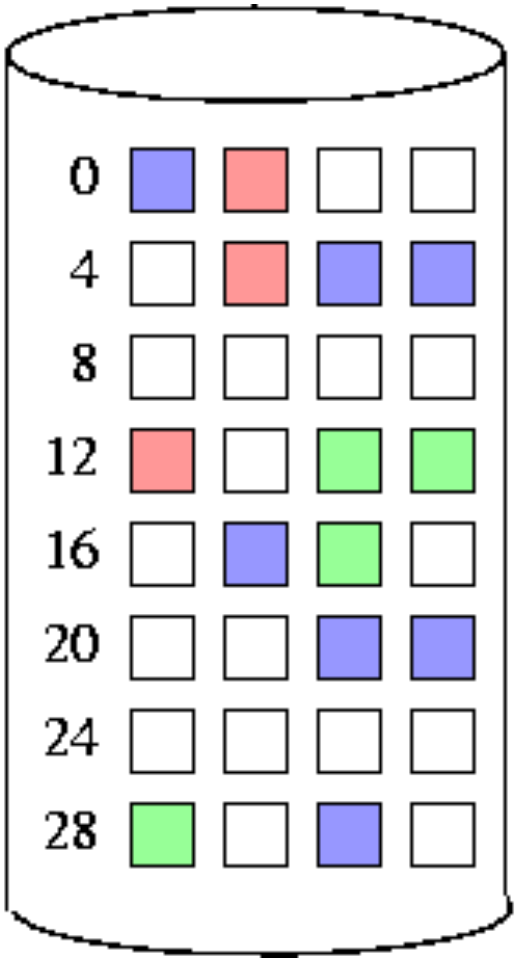
Disk Block Storage w/ Continous Blocks

File System Metadata

Blocks and Standard (Centralized) File Systems

Single Disk

Disk Block Storage w/ Discontinuous Blocks



File System Metadata

Directory

File	Start
moo	5
snow	30
fall	14

0	EOF	EOF		
4		12	0	17
8				
12	1		18	EOF
16		22	28	
20			6	7
24				
28	15		23	

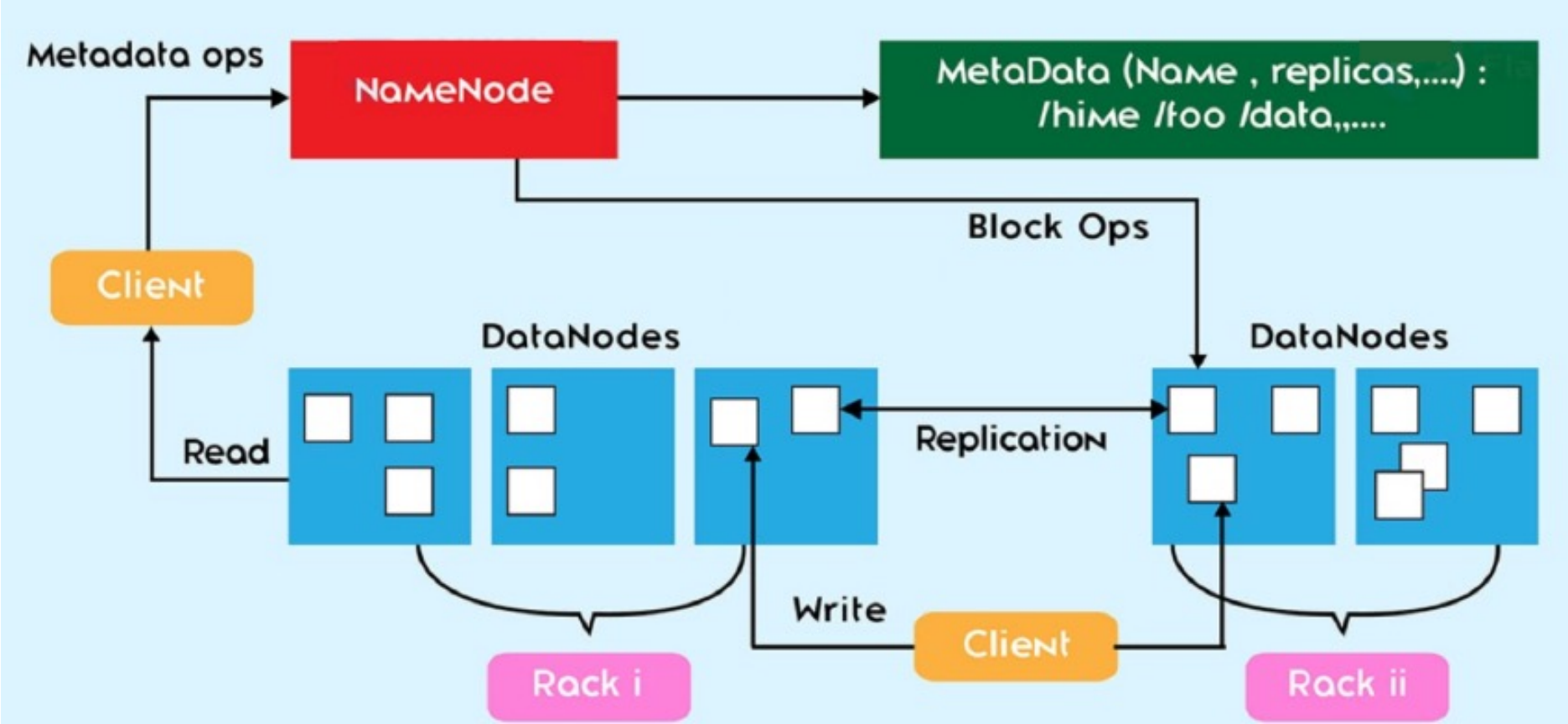
HDFS Concepts

- Blocks and the Hadoop Distributed File System
  - HDFS also stores files as one or more blocks
  - But these blocks are much larger, generally defaulting to 64Mb to 128Mb or even more
  - Rather than store the blocks of a file on one disk the blocks are distributed across disks
  - So HDFS can support files larger than the largest single disk available in the cluster
  - And file blocks distributed across the cluster can also be read in parallel increasing total I/O throughput

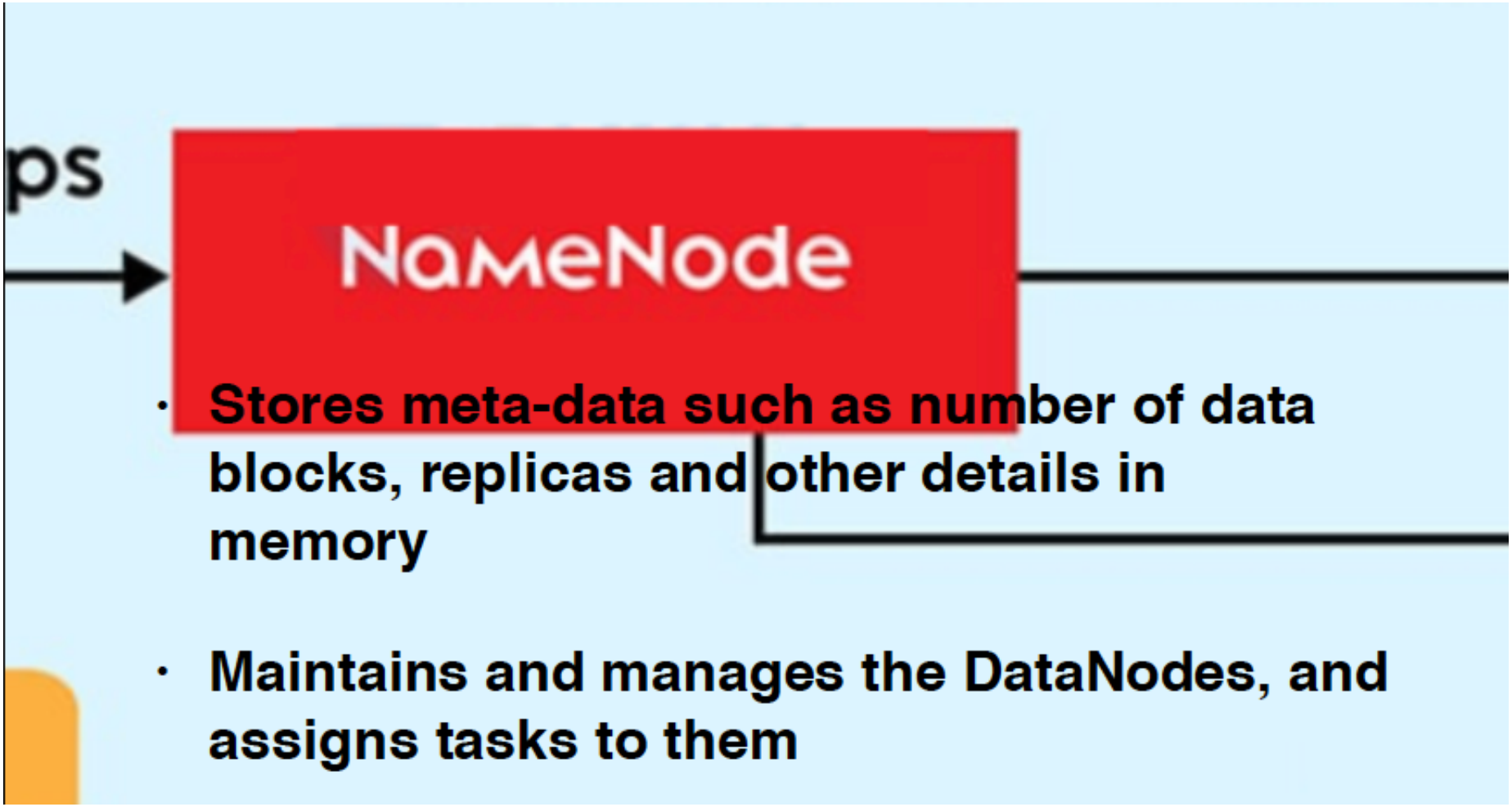
HDFS Concepts

- Blocks and the Hadoop Distributed File System
  - Each file block is usually duplicated twice for a total of three copies (so called default replication factor of 3)
  - With the two replicas stored on different racks where possible
  - Replicated blocks are part of the mechanism allowing HDFS high availability and fault-tolerance

HDFS Architecture

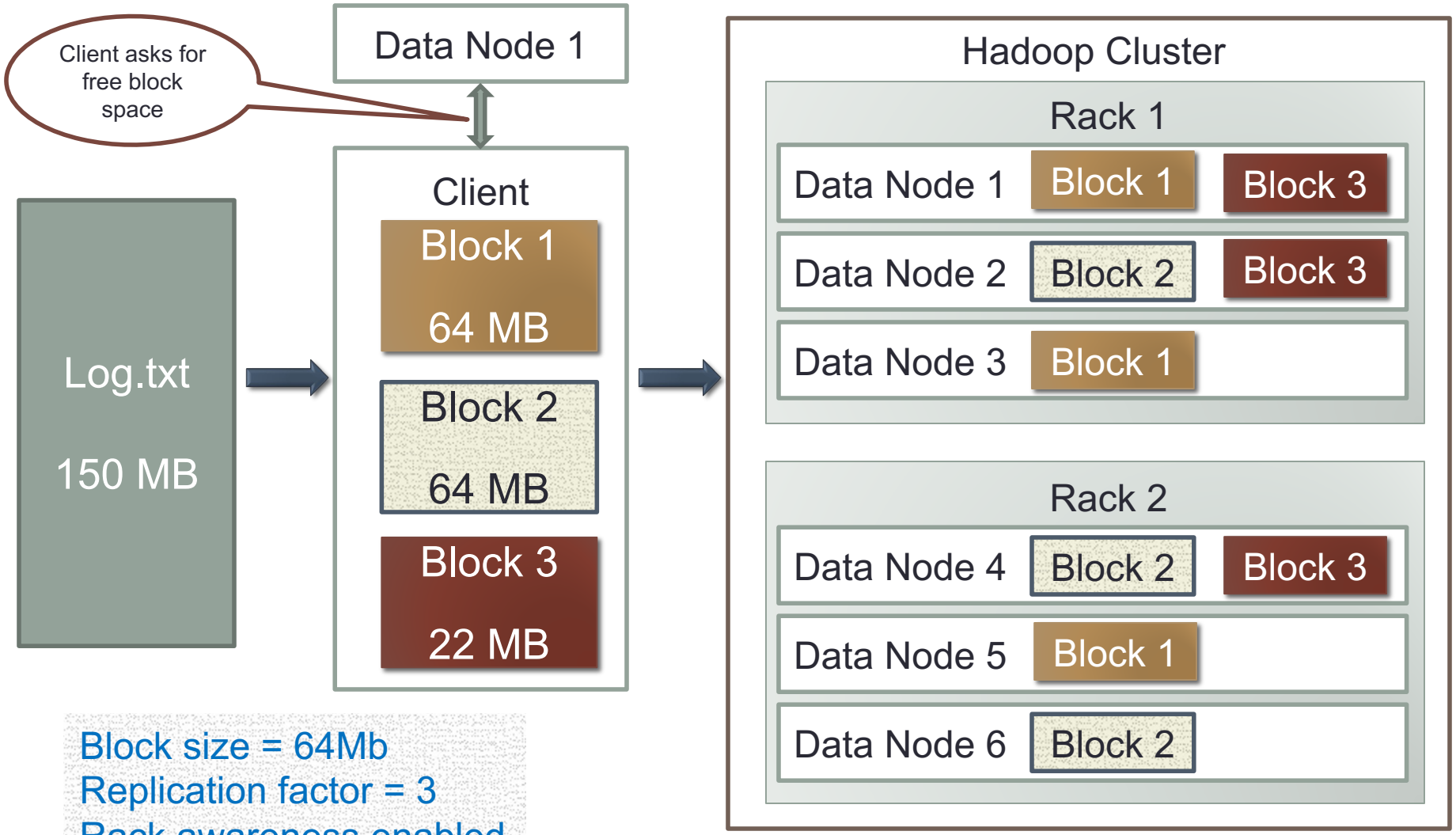


HDFS Architecture



HDFS Architecture

Blocks and the Hadoop Distributed File System



Block size = 64Mb
Replication factor = 3
Rack awareness enabled

HDFS Concepts

- Block Caching
  - Normally a data node reads blocks from persistent storage
  - But frequently accessed blocks may be cached explicitly in memory, for improved data access performance

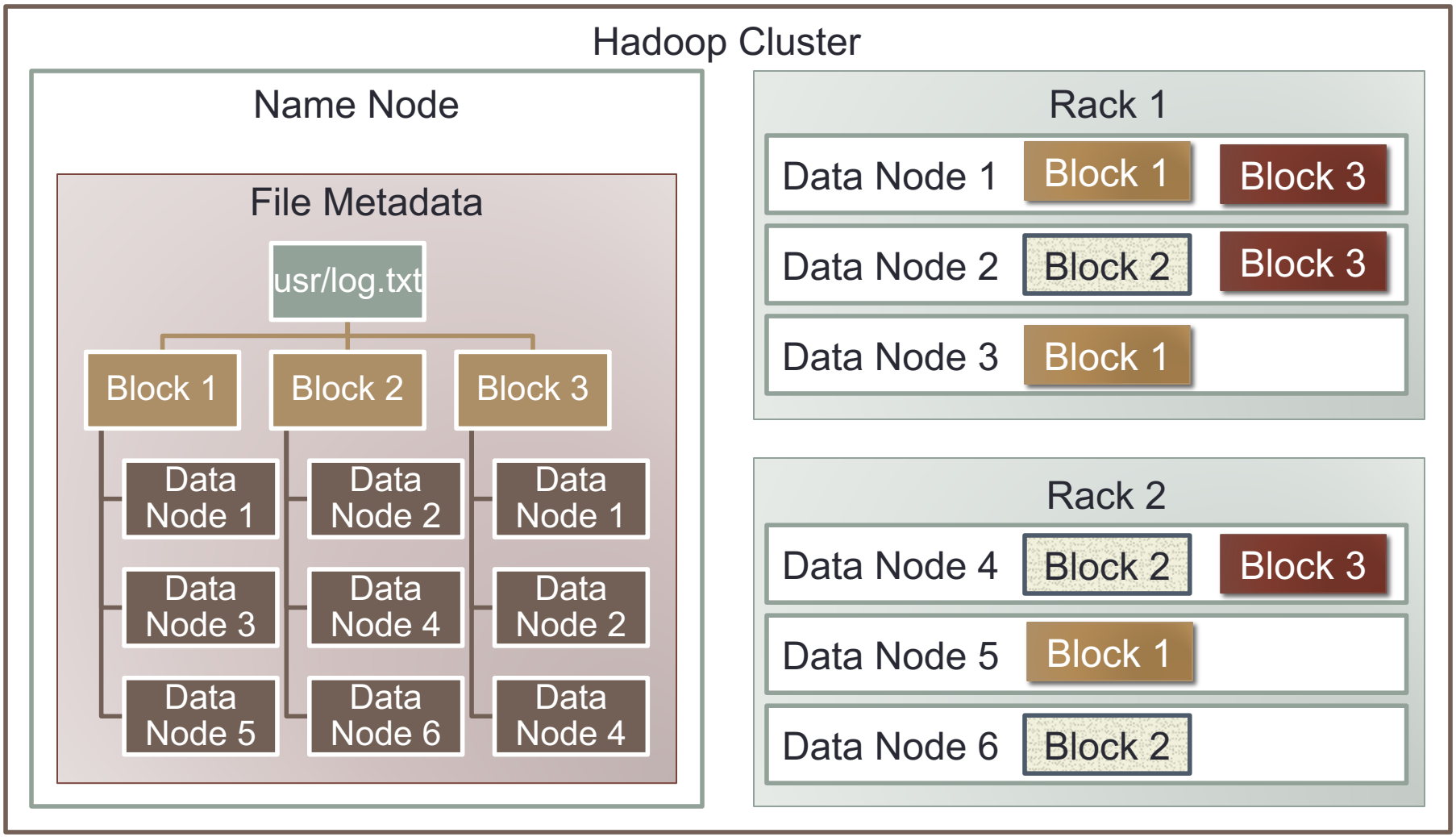
HDFS Concepts

- Name Nodes and Data Nodes
  - HDFS generalizes two facets of a centralized file system
    - File storage and file metadata
  - A HDFS cluster has two categories of nodes operating in a supervisor/worker pattern
    - One or more Name Nodes (supervisor)
    - Many Data Nodes (workers)
  - Name Nodes manage file system metadata
    - File system directory tree
    - File system access permissions
    - File system block location data
    - Keeps metadata in memory for fast access
    - Metadata also stored on disk for reliability

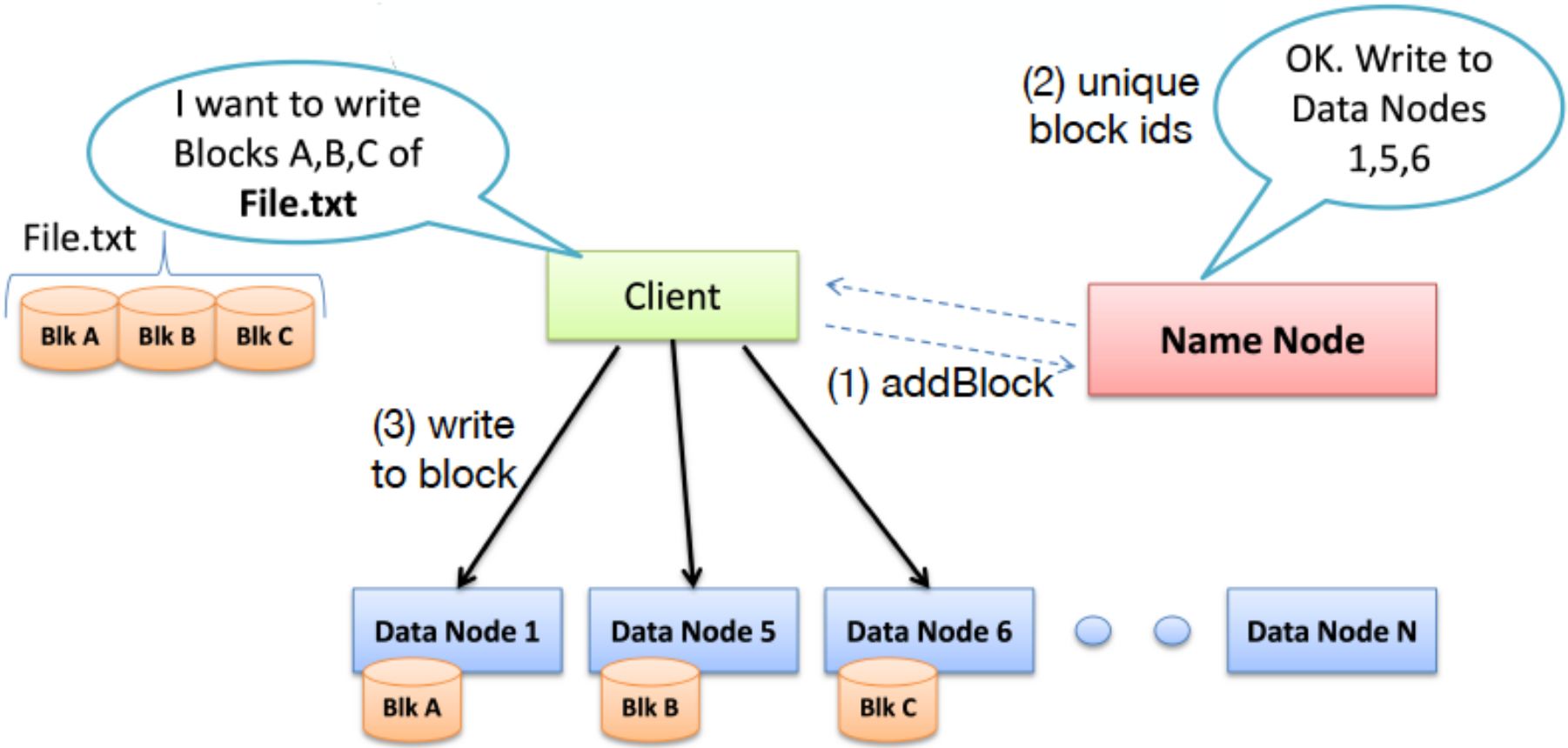
HDFS Concepts

- Name Nodes and Data Nodes
  - Name nodes broker access to files by clients
    - Provides metadata including block location on data nodes
    - Then client communicate directly with data nodes for files
  - Data Nodes store and retrieve (and process) file blocks
    - Blocks are stared as regular files on the local file system
    - A block itself does not know which file it belongs to
    - No data node holds more than one replica of any block
    - Report back to the name node as to what blocks they are holding
    - Sends a heartbeat message to the name node to say that it is still alive
  - Racks of data nodes (rack awarness)
    - Network topology is described to the cluster by an admin
    - Racks often share local switches and power conditioning units
      - Can be a single point of failure
    - If possible no rack holds more than two replicas of a block

Name Nodes and Data Nodes

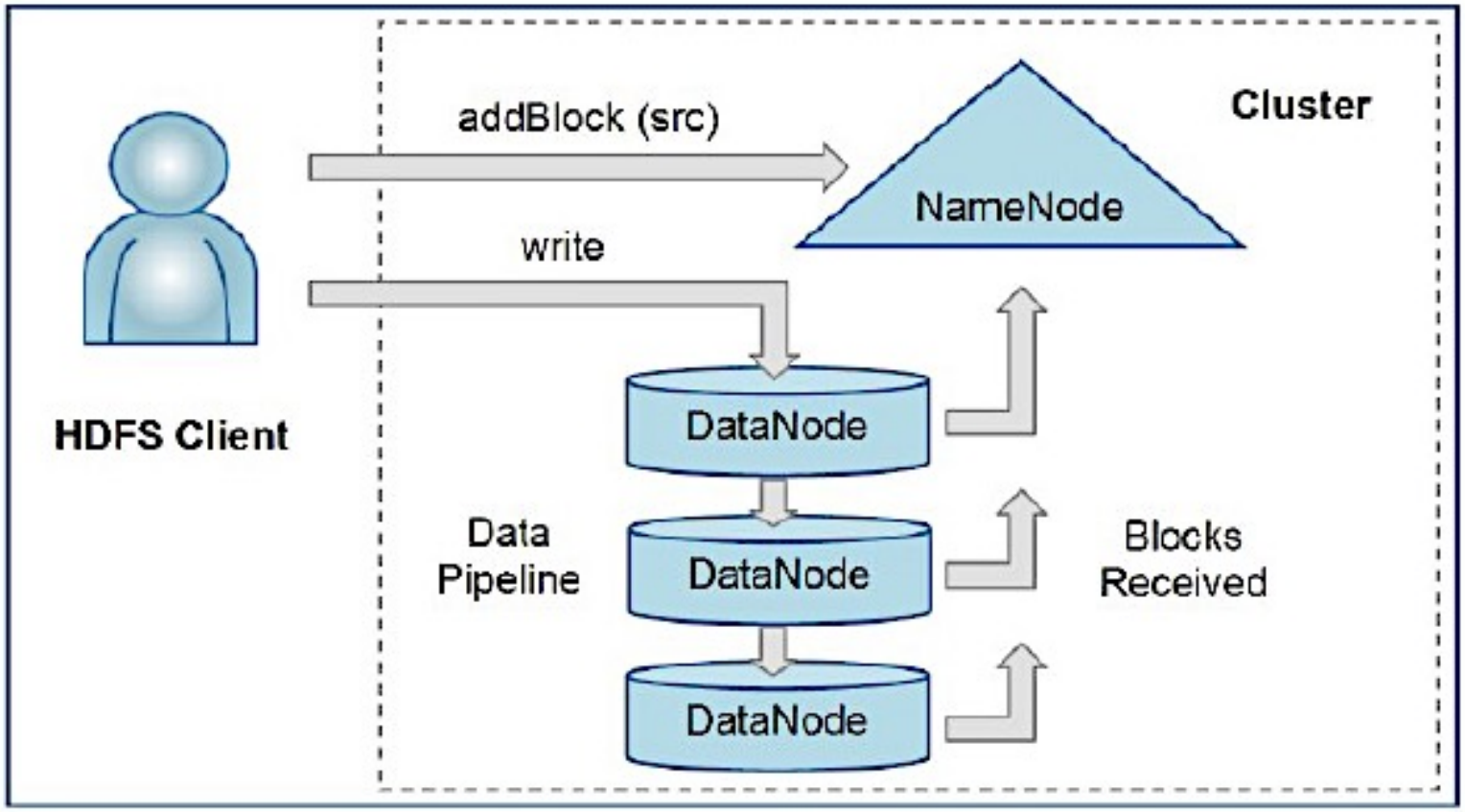


HDFS File Write

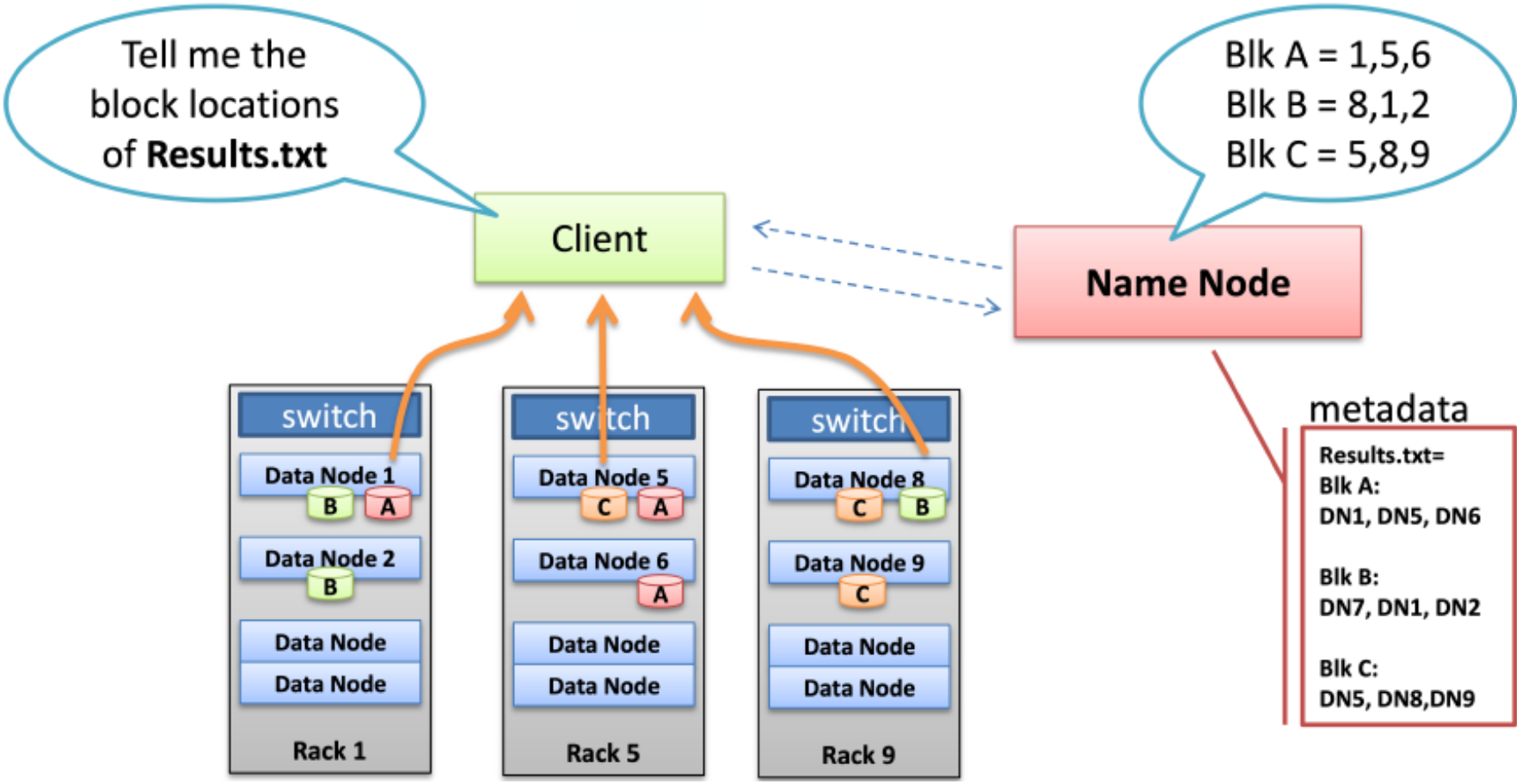


HDFS File Write

Block Replication



HDFS File Read



Common HDFS Failures

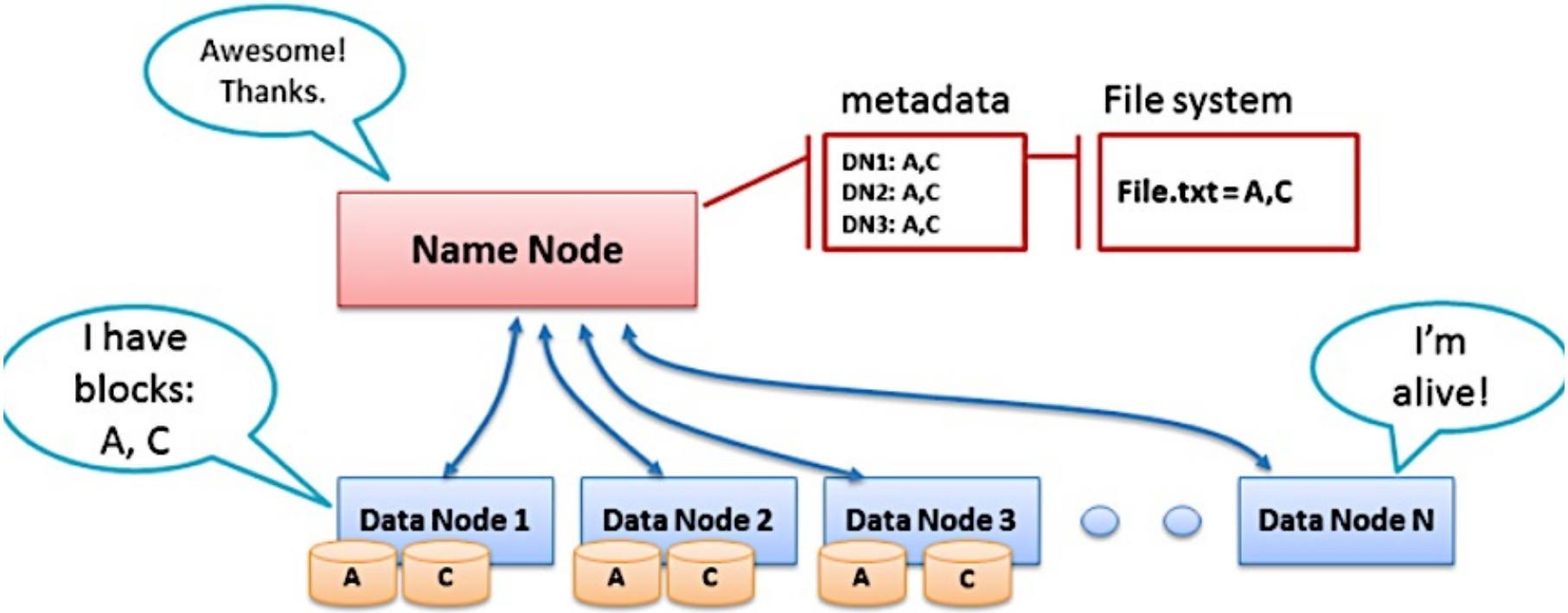
- Node can fail
- Software can fail
- Disk can crash
- Data corruption

Fault Tolerance

- How does this architecture achieve high fault-tolerance?
- Data Nodes Failure
- Name Node Failure

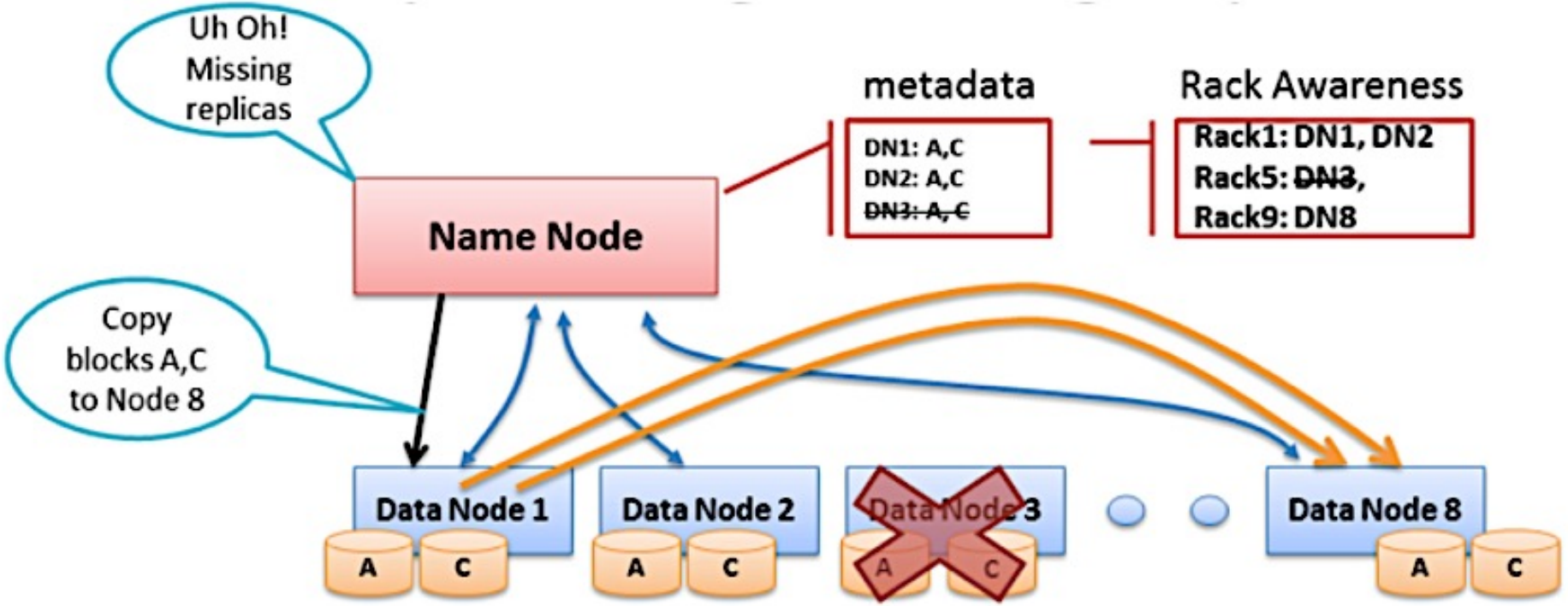
HDFS Architecture

Failure Recovery for Data Nodes



HDFS Architecture

Failure Recovery for Data Nodes



Data Node Fault Tolerance

- Periodic heartbeat—from data node to name node
- Data nodes without recent heartbeat
  - Marked dead, no new I/O sent
  - Blocks below replication factor re-replicated to other nodes

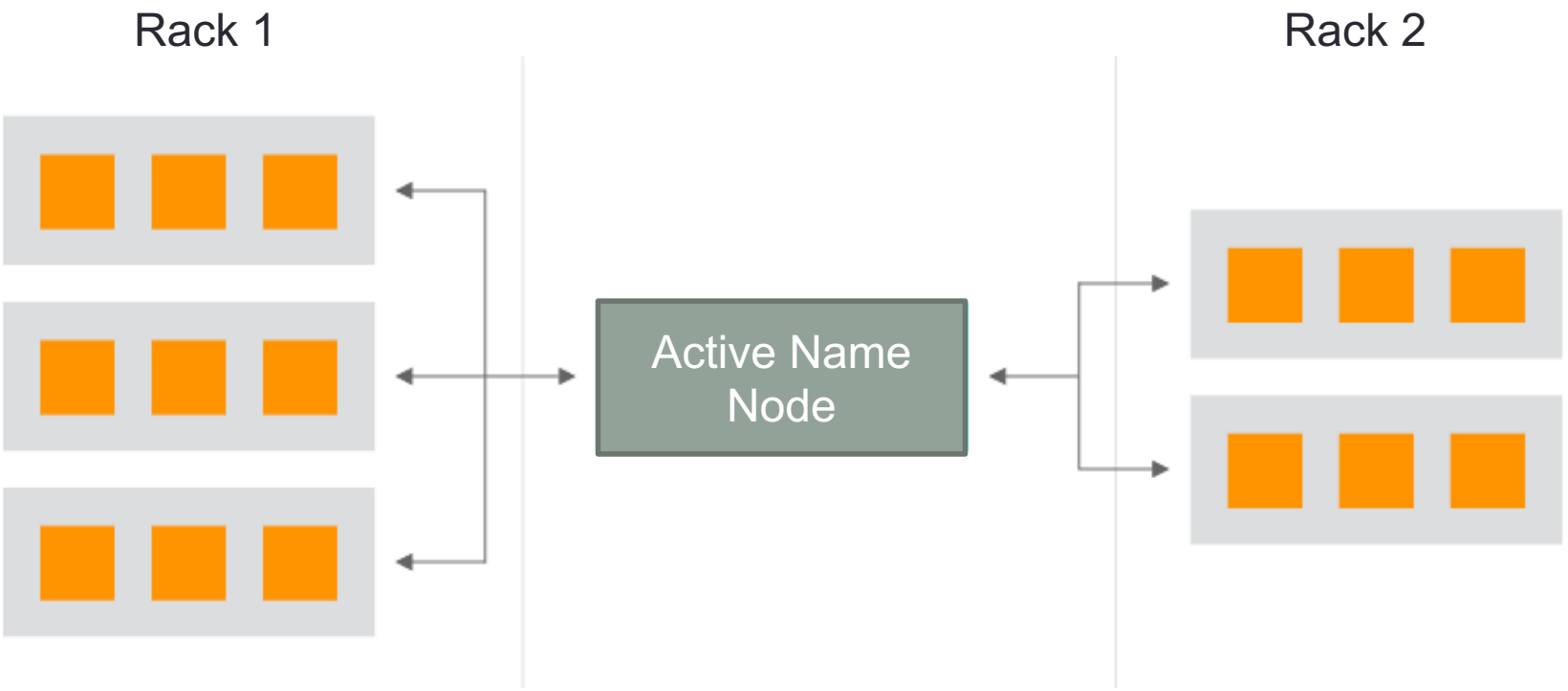
Data Corruption Fault Tolerance

- Checksum computed on file block creation
- Checksums are stored in HDFS metadata
- Used to check retrieved data
  - Re-read from alternate replica if needed

Name Node High Availability (HA)

- Phase 0
  - Single name node maintains all metadata
  - Single point of failure

Single Name Node



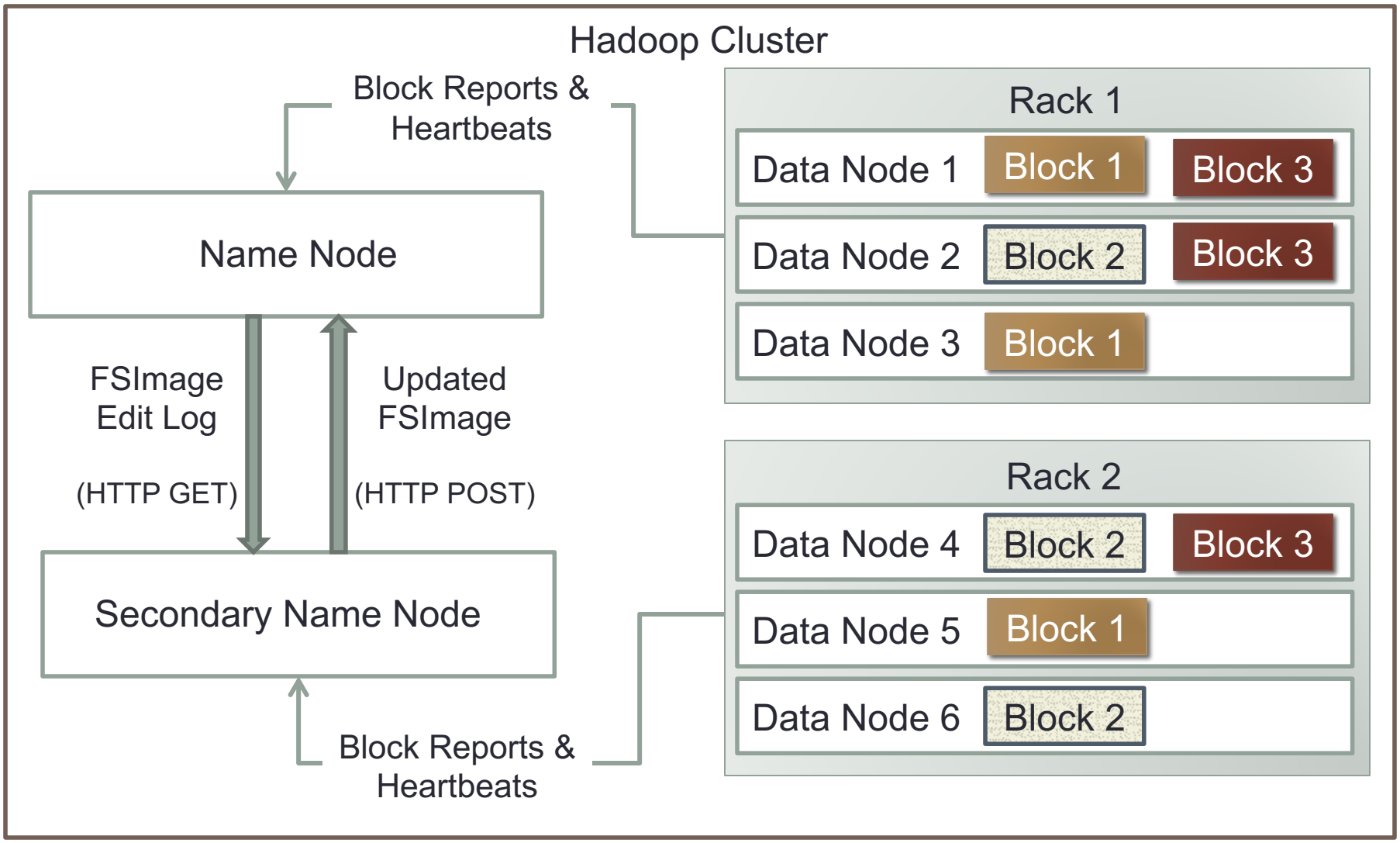
How the Name Node Backs Up Metadata

- Recall the name node keeps its file metadata in memory but is must be made persistent to survive failures
  - The name node stores file system metadata in two separate files: fsimage and edit log
  - These are not stored on HDFS but in the local file system of the Name Node
- The fsimage stores a complete snapshot of system metadata at a specific moment in time
- Incremental changes are then stored in the edit log for durability
  - This rather than create a new fsimage snapshot each time the file metadata is modified
- The Name Node can restore its state by loading the fsimage and performing all the transformations in the edit log
  - But this restore operation can take many minutes

Name Node High Availability (HA)

- Phase 1
  - Primary and Secondary Name Node
  - Communicate using HTTP protocol (no shared storage)
  - Manual failover to secondary
  - Avoids planned cluster downtime for primary name node maintenance
  - Insufficient to protect against unplanned downtime of primary name node

Primary and Secondary Name Nodes



How the Name Node Backs Up Metadata

- If present a Secondary Name Node updates the fsimage each time a change is made to the edit log
  - If the Name Node goes down and is then recovered it does not need to rebuild the fsimage as this was done by the Secondary Name Node
- The Secondary Name Node provides two advantages
  - It unloads the Name Node from the computational burden of updating the fsimage
  - It also allows a much faster recovery by the Name Node by assuring an up-to-date fsimage is always available

Name Node High Availability (HA)

- Phase 2
  - Automatically detect failure of active name node
  - Does not require operator intervention to initiate failover
  - Removes single point of failure

HA Name Node Support

- The Name Node and Secondary Name Node do not support automatic failure recovery
  - The Name Node is still a single point of cluster failure in this arrangement
- Moving from a Secondary Name Node to a Standby Name Node provides this availability mechanism
- Beyond a Standby Name Node two other system related capabilities are required to support high availability
  - Shared storage of the fsimage and edit log across name nodes using a highly availability storage scheme
  - Support to automatically have the Standby Name Node become the (Primary) Name Node using a cluster coordination service

HA Name Node Support

- The Quorum Journal Manager to used achieve a high availability metadata filesystem
  - Reachable by Primary and Standby Name Nodes
  - Of course, we can't use HDFS itself… why is this?
- When the active name node modifies its metadata it logs a record of change to the majority of journal nodes
- The Standby name node watches the journal nodes for changes to the edit logs
  - And applies them to its local copy of the fsimage

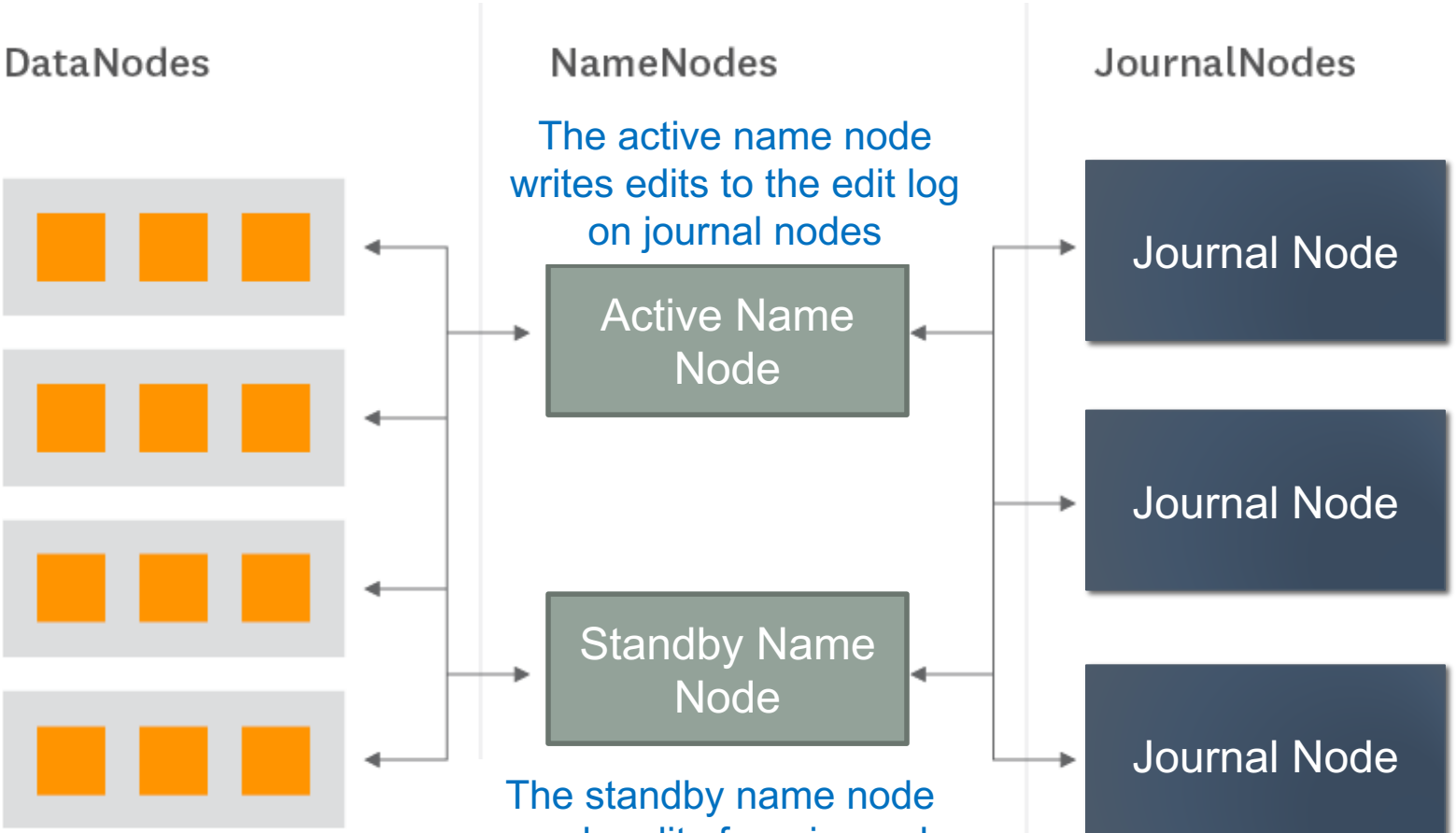
HA Name Node Support

- Automatic failover supports requires two components
  - Zookeeper (Quorum)—used for coordination across cluster nodes
    - Multiple instances to support HA
  - ZKFailoverController—a process that runs on each Name Node maintaining contact with Zookeeper
- If a Name Node loses contact with Zookeeper the other Name Node becomes aware of this and then takes over as the primary Name Node
- The switch over from one name node to another can take just a minute or so
  - This is much less time than a manual failover to the Secondary Name Node

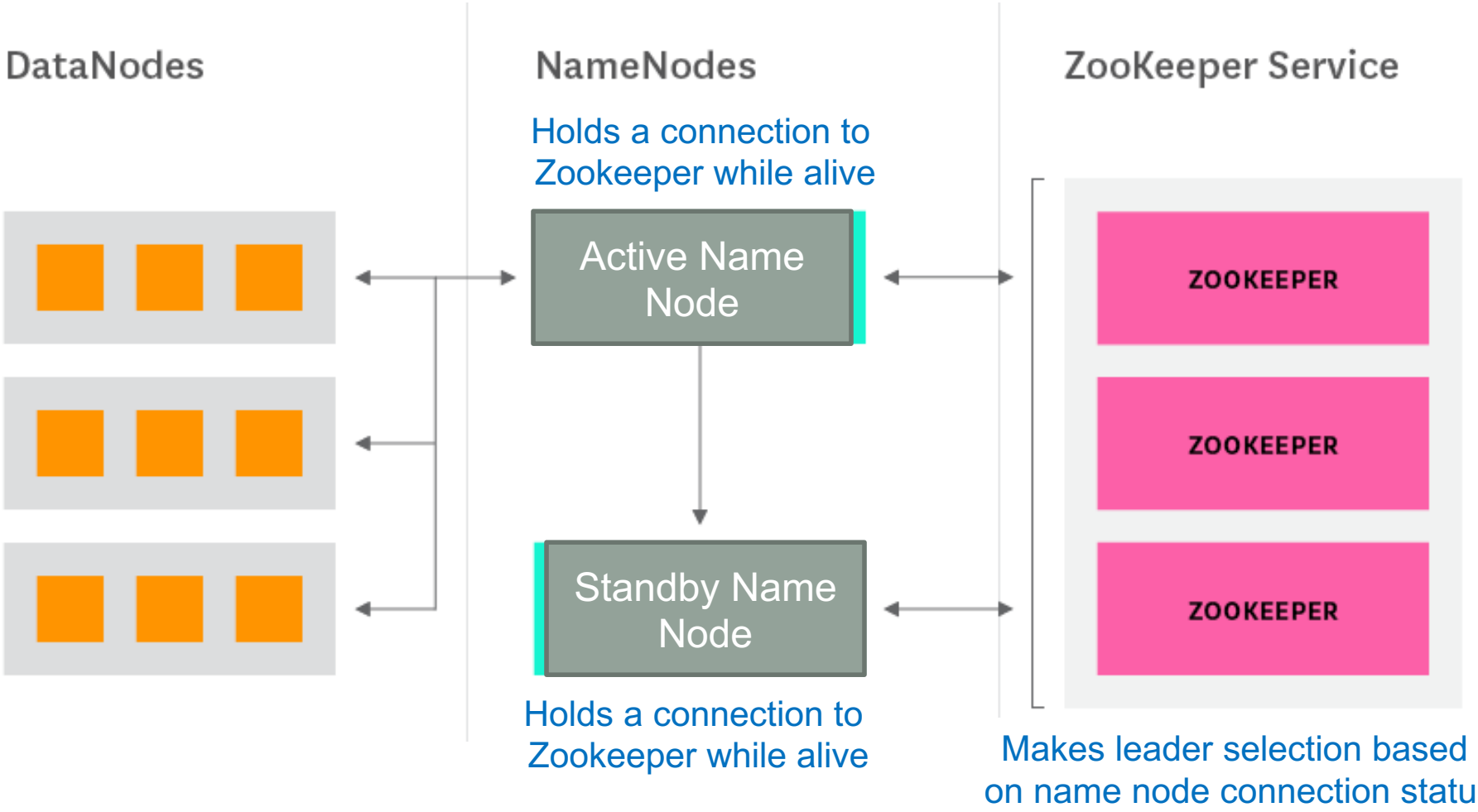
HA Name Node Support

- ZKFC has three simple responsibilities
  - Monitors health of associated name Nodes
  - Participates in leader election of Name Nodes
  - Fences other name nodes if it wins election
  - Removes requirement for shared name node storage

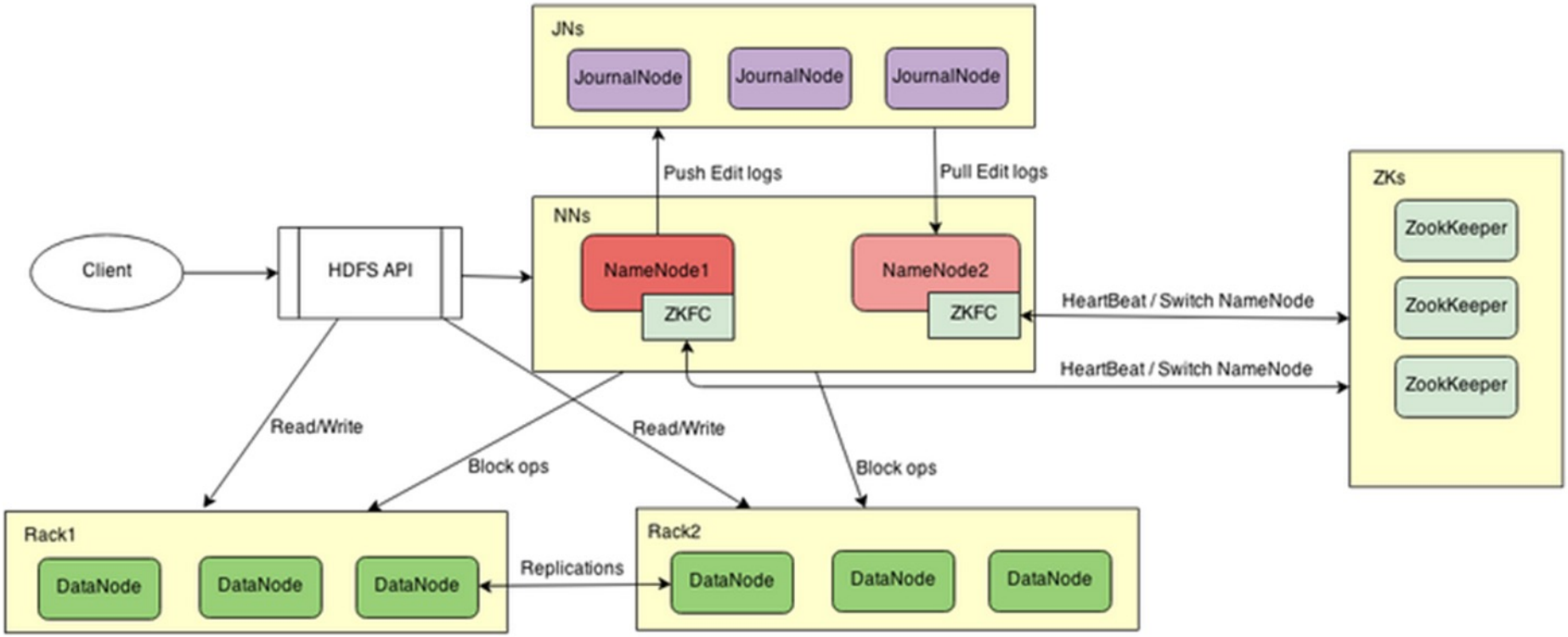
High Availability Configuration



High Availability Configuration



Full HDFS HA Architecture



HDFS Access Options

- Via Command Line
- Application Programing Interface
- NFS Gateway
- Other Options

HDFS Command Line

- $HADOOP_HOME/bin/hadoop fs <args>
- See online command reference
  - https://hadoop.apache.org/docs/r2.7.2/hadoop-project-dist/hadoop-common/FileSystemShell.html
  - Also as PDF in “Books” section of the course Blackboard account
- Operations are somewhat similar to what you might use to manipulate Linux files and directories
  - ls, mv, mkdir, chggrp, chgown, cat, cp, …
- Some specialized operations for moving data between the local (Linux, Windows) file system and HDFS
  - copyFromLocal (and put)
  - copyToLocal (and get)

“hadoop fs” versus “hdfs dfs”

- Two command line processors to manipulate Hadoop files
- **hadoop fs**
  - More “generic” command that allows you to interact with multiple file systems
    - Local files (linux, windows, etc.)
    - Simple Storage Service (S3) and other cloud object stores
    - HDFS (includes EMR, and other cloud hosted Hadoop systems)
- **hdfs dfs**
  - Command that is specific to the HDFS file system

- Note that **hdfs dfs** and **hadoop fs** commands become synonymous if the filing system which is used is HDFS

hadoop fs

Accessing Different File Systems

File System	Example
HDFS (local)	hdfs:///path/to/local/hdfs//file.json This works on the EMR master node. Amazon EMR resolves paths that do not specify a prefix in the URI to the local HDFS, so the above cold also be written as: /path/to/local/hdfs//file.json
HDFS (remote)	hdfs://name-node-ip-addr:port/path/to/remote/hdfs/file.json
S3	s3://bucket-name/object_name.json
Local (Linux, etc.)	file:///path/to/local/os/file.json For certain “hadoop fs” commands, you don’t need ‘file://’ when a command argument must be a local file

hadoop fs

Accessing Different File Systems

File System	Example
List the contents of an HDFS directory	hadoop fs –ls hdfs:///hdfs-path/dir-name Ex. List the root HDFS directory: *hadoop fs –ls /* <= *'hdfs:///' assumed by default*
Create a new HDFS directory	hadoop fs –mkdir hdfs:///hdfs-path/dir-name Ex. Create an HDFS directory 'joe' as a sub-directory of the existing HDFS directory '/users': *hadoop fs –mkdir /users/joe*
Display the contents of an HDFS file or S3 object	hadoop fs –cat s3://bucket-name/object_name.json
Delete an HDFS file or S3 object	*hadoop fs –rm* hdfs:///hdfs-path/file.json

hadoop fs

Accessing Different File Systems

File System	Example
Copy a file from the local file system to the destination file system	hadoop fs –put /usr/local/file.txt hdfs:///hdfs-path/dir-name hadoop fs –put /usr/local/file.txt hdfs:///hdfs-path/file.txt
Copy a file from the local file to HDFS	hadoop fs -copyFromLocal localfile /user/hadoop/hadoopfile
Copy a file to the local file system from HDFS	hadoop fs –get hdfs:///hdfs-path/file.txt /to/path/flie.txt
Display the contents of an HDFS file or S3 object	hadoop fs –cat s3://bucket-name/object_name.json
Delete an HDFS file or S3 object	*hadoop fs –rm* hdfs:///hdfs-path/file.json

hadoop fs

Accessing Different File Systems

File System	Example
Copy a file from source to destination	hadoop fs –cp hdfs:///some/path/to/file.txt hdfs:///other-path Now there are two copies of the file
Move a file from source to destination	hadoop fs –mv hdfs:///some/path/to/file.txt hdfs:///other-path Now there is still one copy of the file, but in a new directory hadoop fs –mv hdfs:///path/file.txt hdfs:///path/renamed.txt Now there is still one copy of the file but with a new name

HDFS Java API

- Base Class
  - org.apache.hadoop.fs.FileSystem
- Important Classes
  - FSDataInputStream
  - FSDataOutputStream
- Operations
  - get, open, create, read …

HDFS

Key Ideas

- Write Once, Read Many times (WORM)
- Divide files into big blocks and distribute across the cluster
- Store multiple replicas of each block for reliability
- Programs can ask "where do the pieces of my file live?”

Object Versus HDFS

	S3	HDFS
Storage Type	Buckets and Objects No Bucket Hierarchy	Directories and Files Subdirectory Hierarchy
Write Once Read Many	Yes	Yes
Can Update Existing (Stored) Data	No	No
Can Append to Existing (Stored) Data	No	Yes