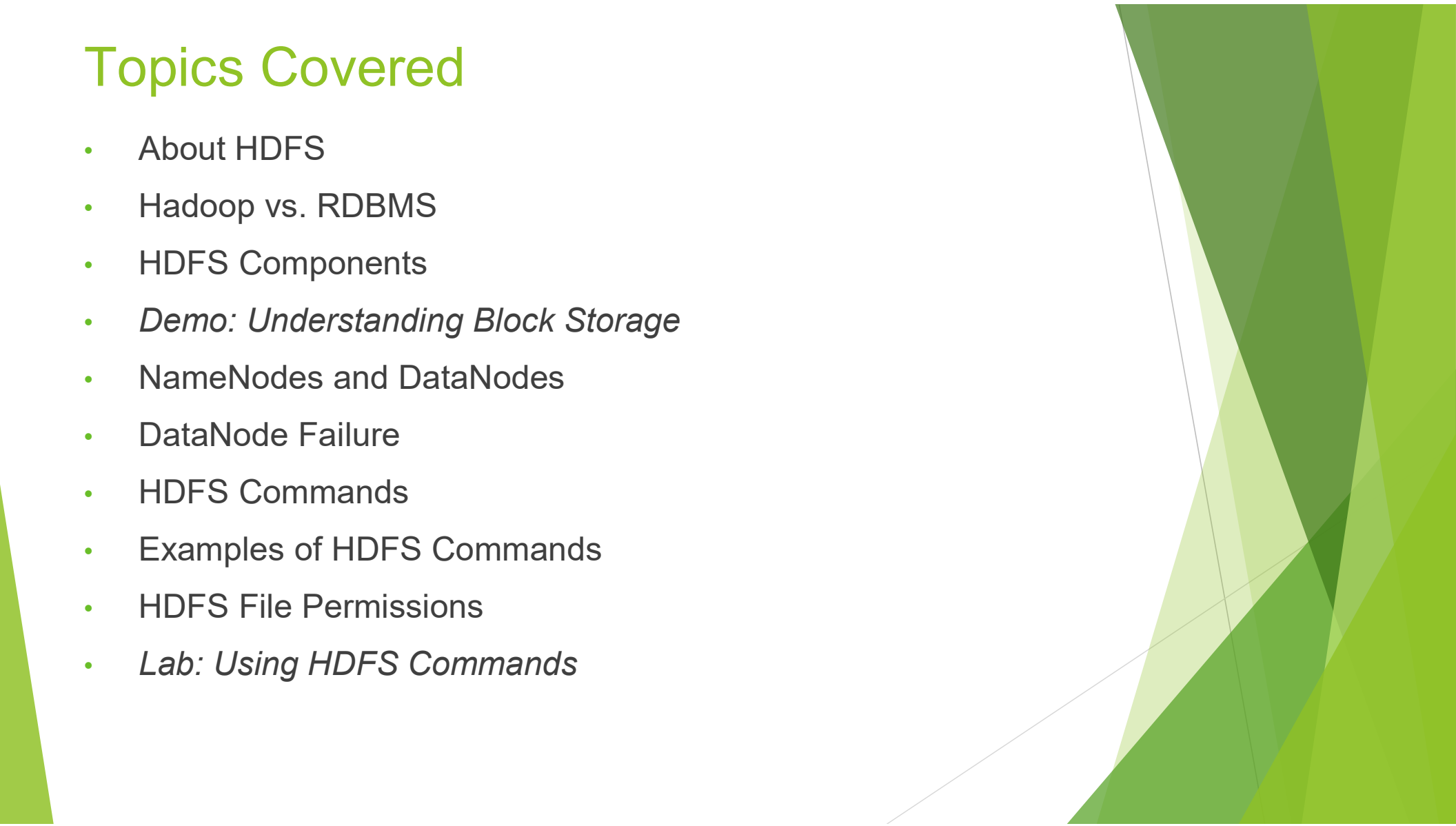
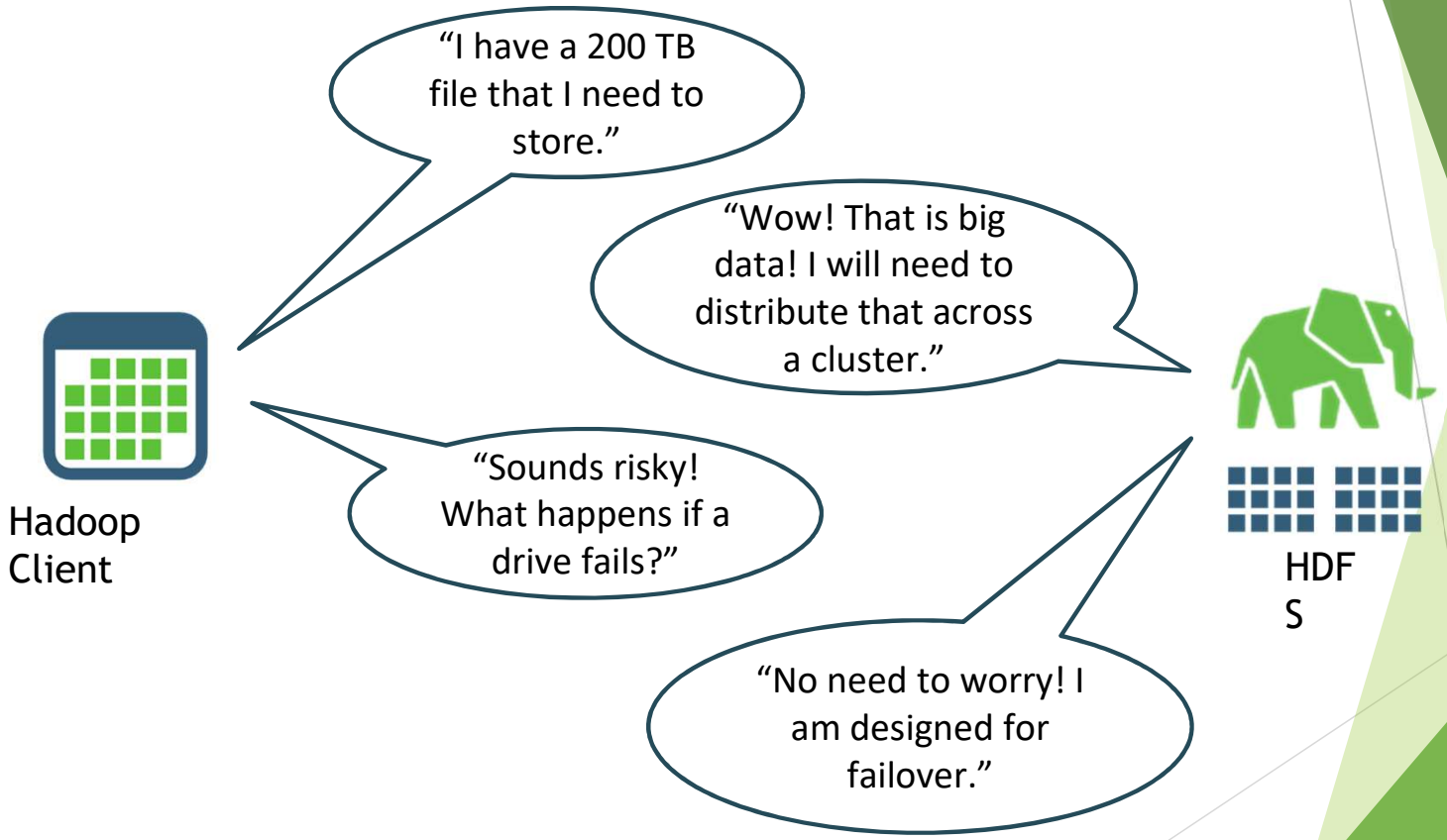
HDFS

Topics Covered

- About HDFS
- Hadoop vs. RDBMS
- HDFS Components
- *Demo: Understanding Block Storage*
- NameNodes and DataNodes
- DataNode Failure
- HDFS Commands
- Examples of HDFS Commands
- HDFS File Permissions
- *Lab: Using HDFS Commands*



About HDFS


"I have a 200 TB file that I need to store."
"Wow! That is big data! I will need to distribute that across a cluster."
"Sounds risky! What happens if a drive fails?"
"No need to worry! I am designed for failover."
Hadoop Client
HDFS

Hadoop RDBMS

- Assumes a task will require reading a significant amount of data off of a disk
- Does not maintain any data structure
- Simply reads the entire file
- Scales well (increase the cluster size to decrease the read time of each task)
  - 2,000 blocks of size 256MB
  - 1.9 seconds of disk read for each block
  - On a 40 node cluster with eight disks on each node, it would take about 14 seconds to read the entire 500 GB

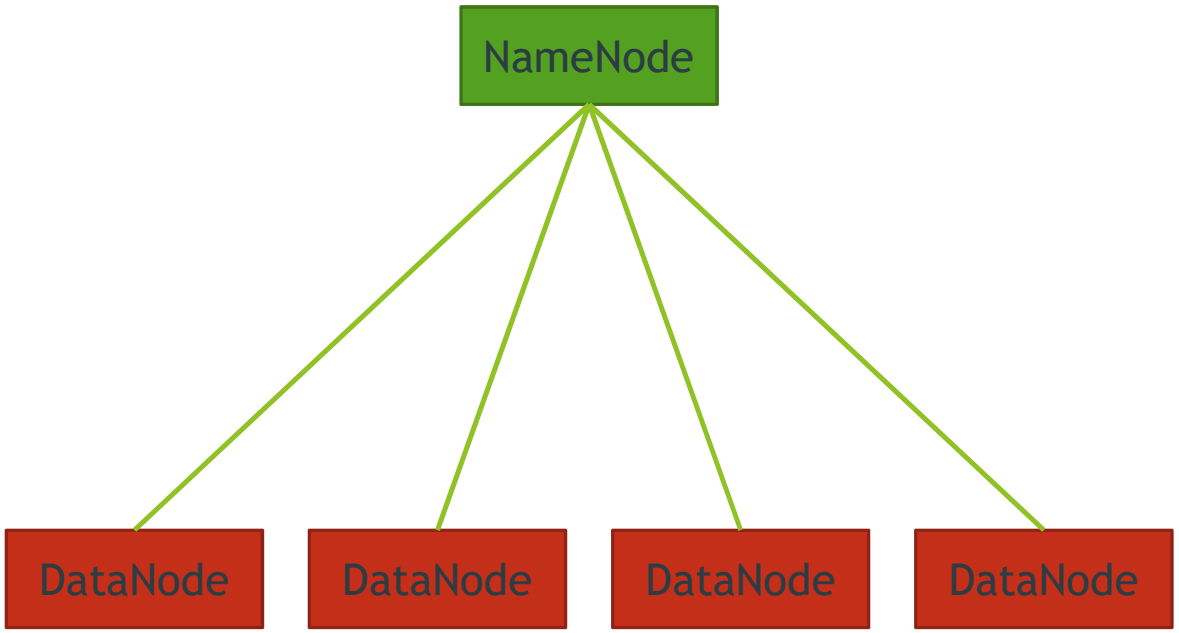
61 minutes to read this data off of a disk (assuming a transfer rate of 1,030 Mbps)

- Uses indexes to avoid reading an entire file (very fast lookups)
- Maintains a data structure in order to provide a fast execution layer
- Works well as long as the index fits in RAM

HDFS Characteristics

Characteristic	Description
Hierarchical	Directories containing files are arranged in a series of parent-child relationships.
Distributed	File system storage spans multiple drives and hosts.
Replicated	The file system automatically maintains multiple copies of data blocks.
Write-once, read-many optimized	The file system is designed to write data once but read the data multiple times.
Sequential access	The file system is designed for large sequential writes and reads.
Multiple readers	Multiple HDFS clients may read data at the same time.
Single writer	To protect file system integrity, only a single writer at a time is allowed.
Append-only	Files may be appended, but existing data not updated.

HDFS Components - NameNode and DataNodes Introduction



- **Master node** maintaining file system namespace and metadata including:
  - File names
  - Directory names
  - File system hierarchy
  - Permissions and ownerships
  - Last modification times
  - ACLs
- **Worker nodes** containing only file data blocks.

*More detail about NameNode and DataNode operation and management is provide in another lesson.

HDFS Components

NameNode

- Is the “master” node of HDFS
- Determines and maintains how the chunks of data are distributed across the DataNodes

DataNode

- Stores the chunks of data, and is responsible for replicating the chunks across other DataNodes

HDFS Architecture

- The NameNode and DataNodes are daemons running in a Java virtual machine.
- This lesson provides details about these components and their operation.

Primary NameNode - memory-based service

Namespace
- Hierarchy
- Directory names
- File names

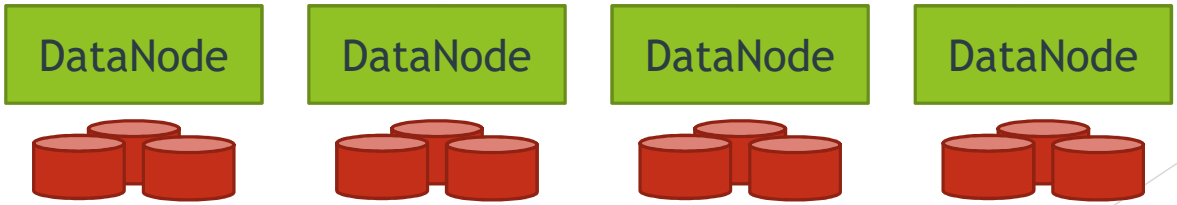
Metadata
- Permissions and ownership
- ACLs
- Block size and replication level
- Access and last modification times
- User quotas

Journaling
- Safely records file system changes

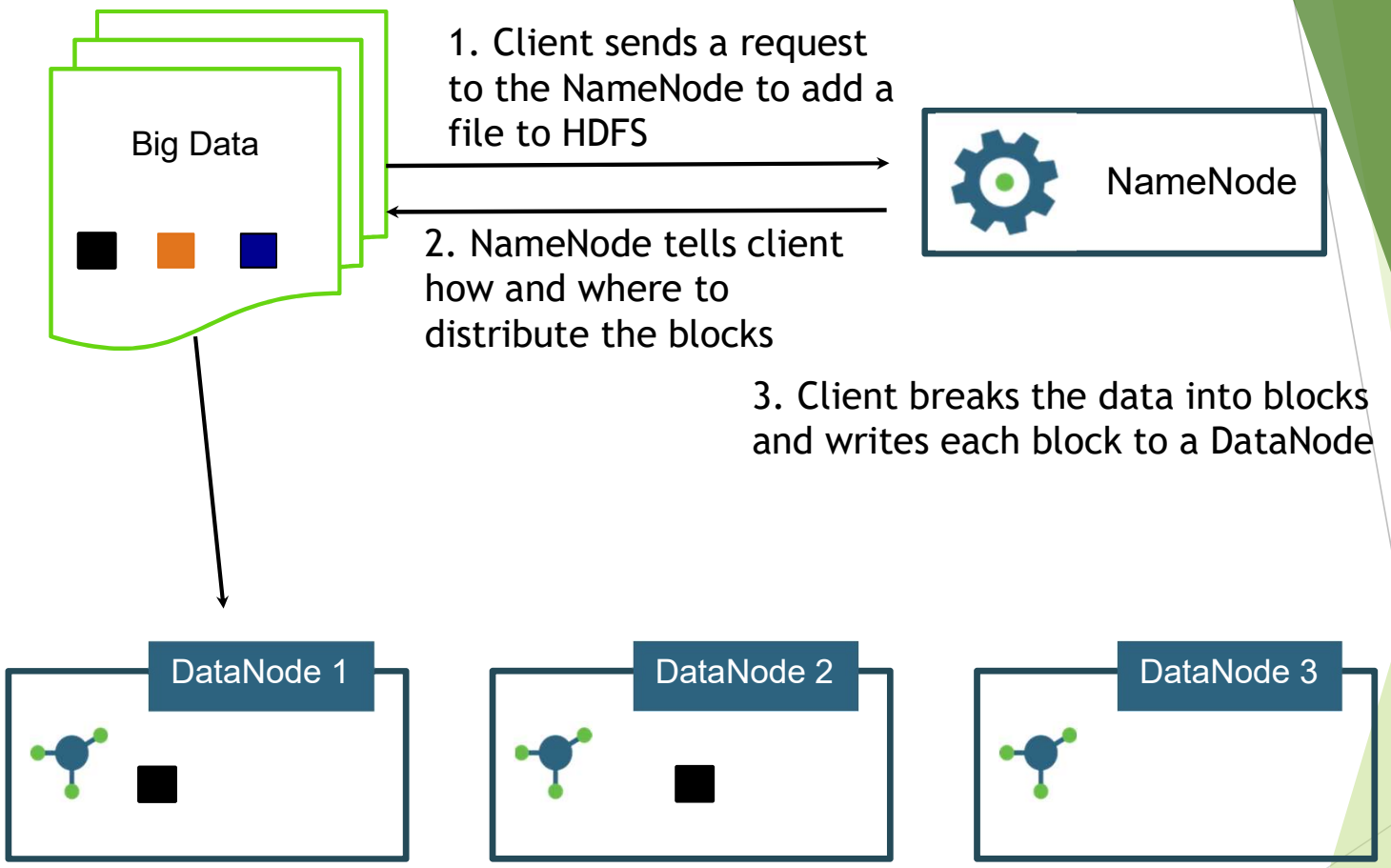
Block Map
- File names > block IDs

Secondary/Standby NameNode

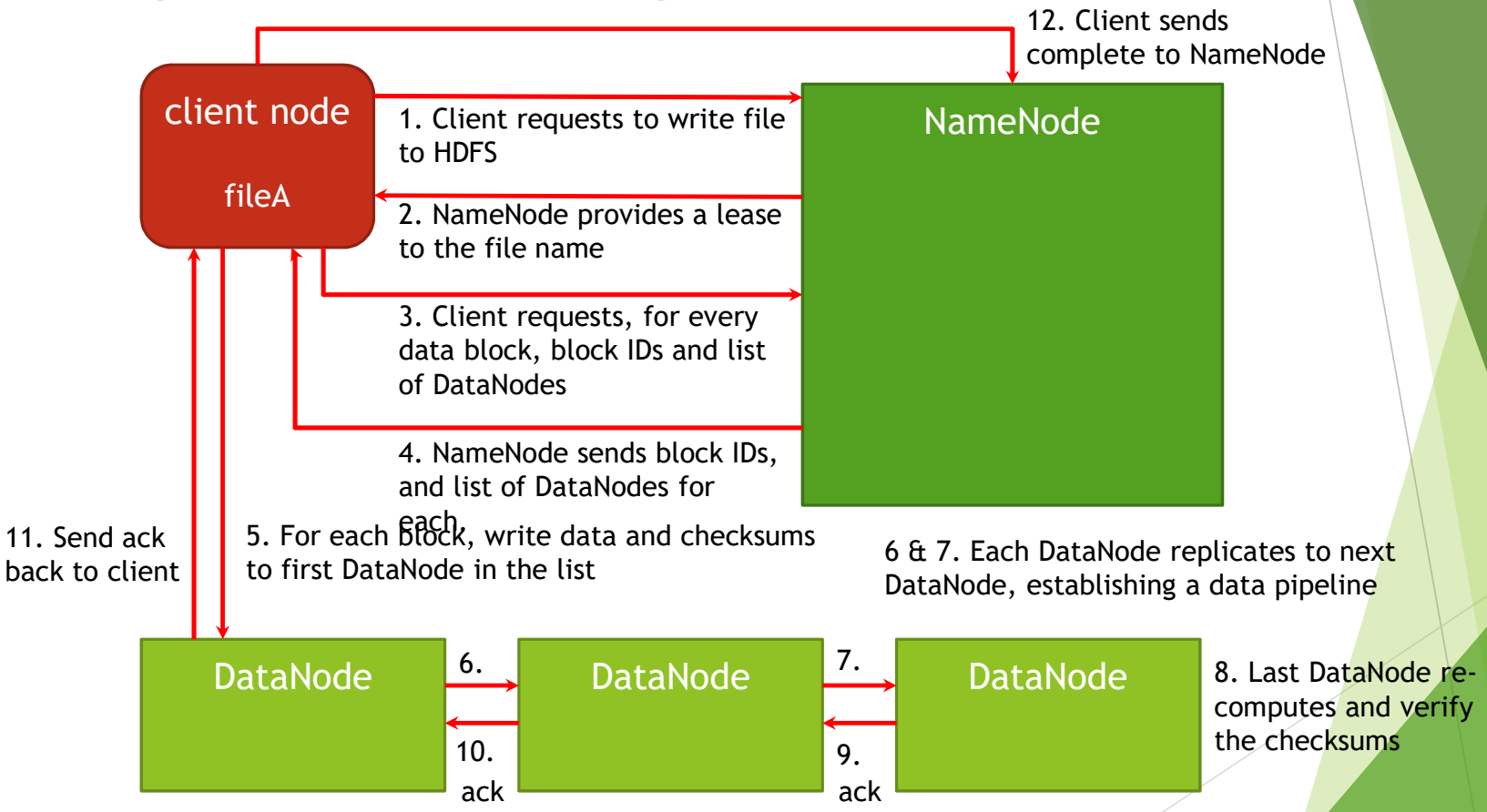
Checkpointing
- Merges the disk-based files used to persist in-memory file system state information



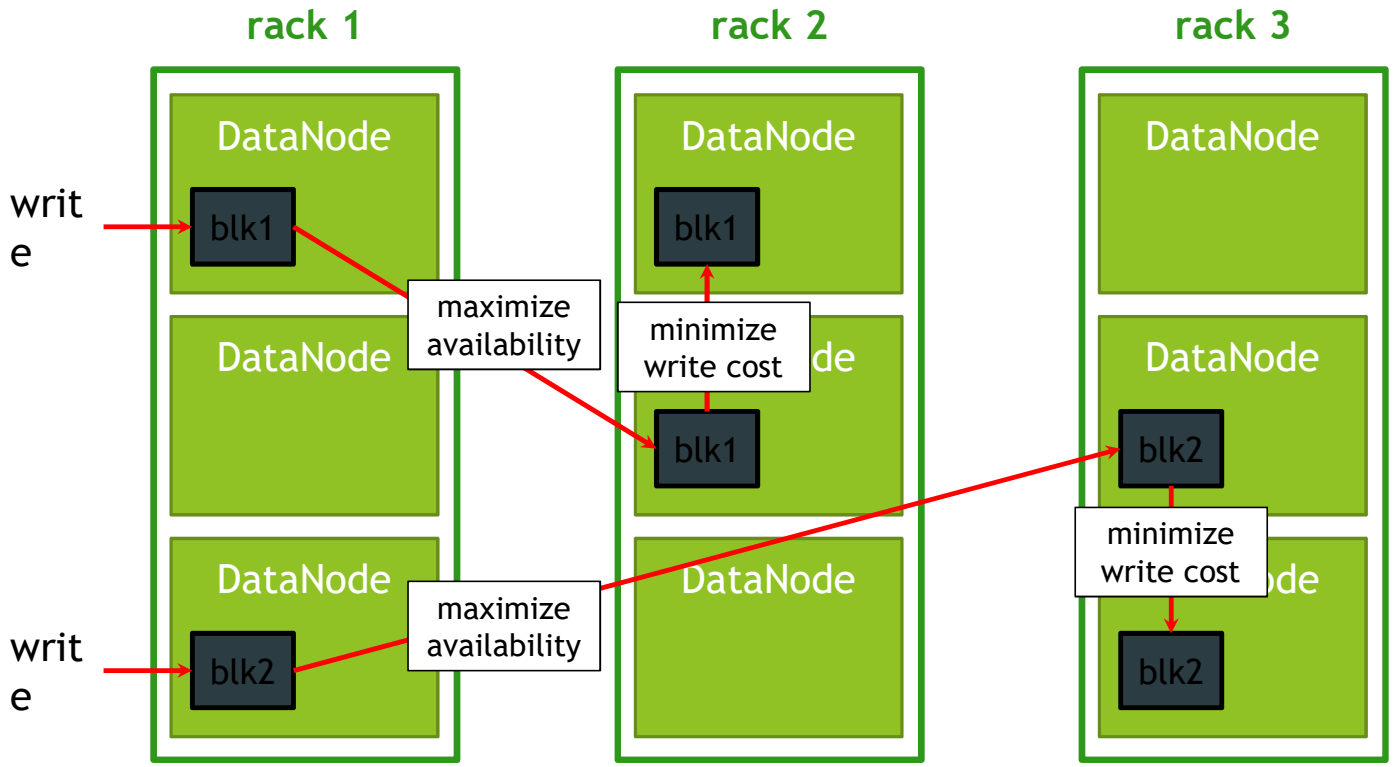
Block Storage
- Data blocks


Big Data
1. Client sends a request to the NameNode to add a file to HDFS
NameNode
2. NameNode tells client how and where to distribute the blocks
3. Client breaks the data into blocks and writes each block to a DataNode
DataNode 1
DataNode 2
DataNode 3
4. The DataNode replicates each block to two other DataNodes (as chosen by the NameNode)

Writing to HDFS Storage – Detailed view



Replication and Block Placement



HDFS is designed to assume, and handle, disk and system failures.

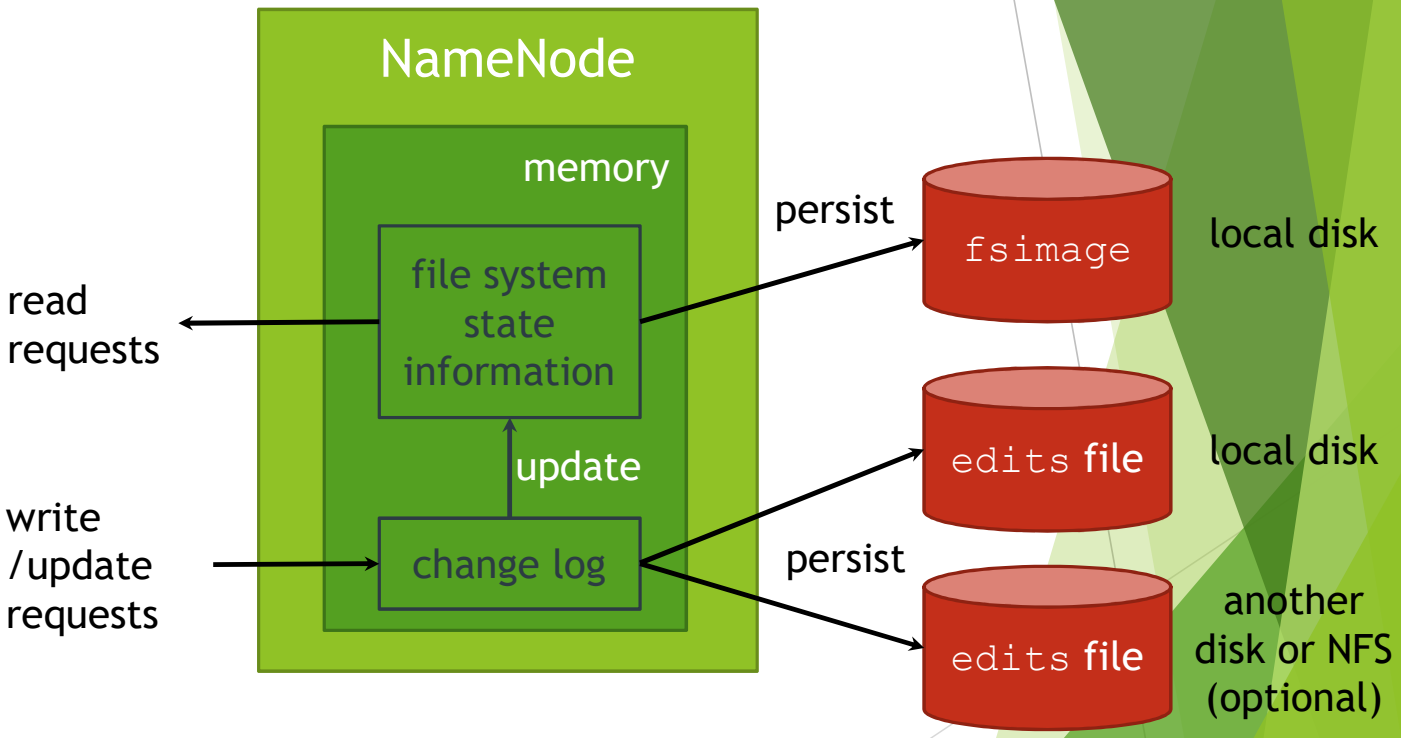
Minimize write cost

Maximize availability and read performance

Demo: Understanding Block Storage

Persisting File System Information on the NameNode

- File system state is maintained and served from memory.
- Memory is fast but volatile.
- File system state is regularly persisted to disk.



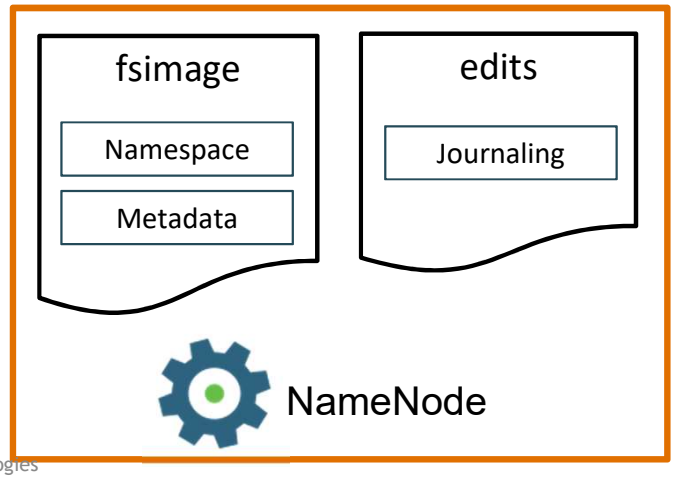
The NameNode Startup

1. When the NameNode starts, it reads the **fsimage_N** and **edits_N** files.

1. The transactions in **edits_N** are merged with **fsimage_N.**

1. A newly created **fsimage_N+1** is written to disk, and a new, empty **edits_N+1** is created.

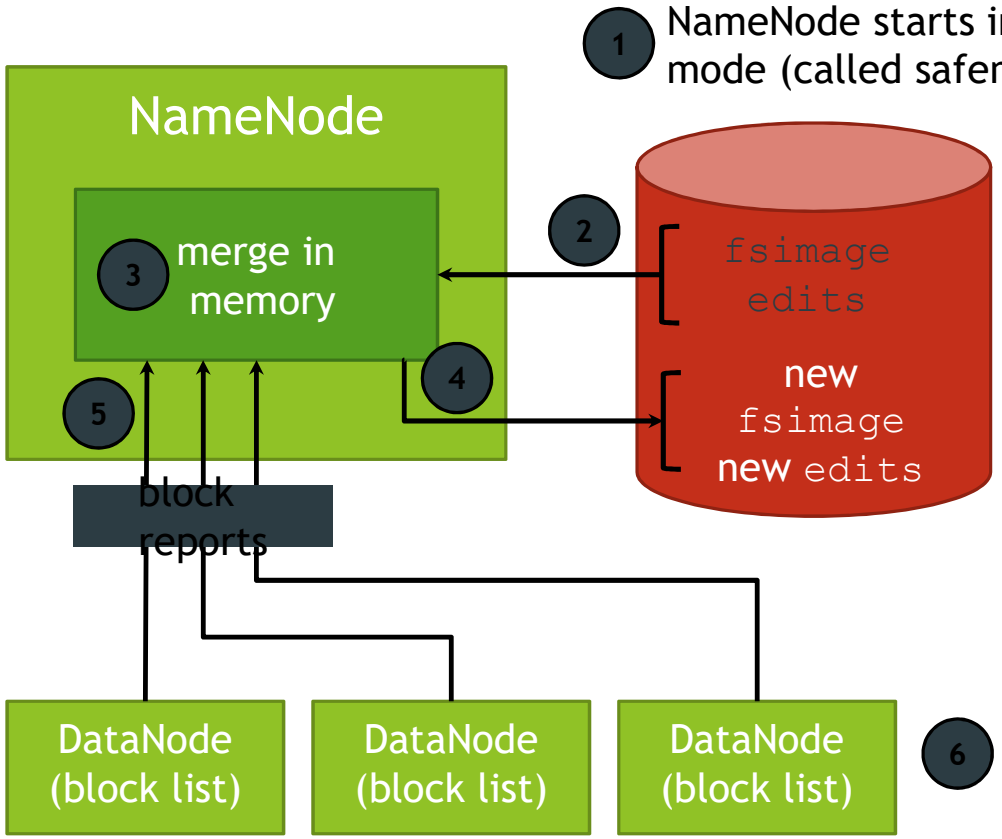
The NameNode will be in *safemode*, a read-only mode.



4. Now a client application can create a new file in HDFS

4. The NameNode journals that create transaction in the **edits_N+1** file

NameNode Startup – Detailed View

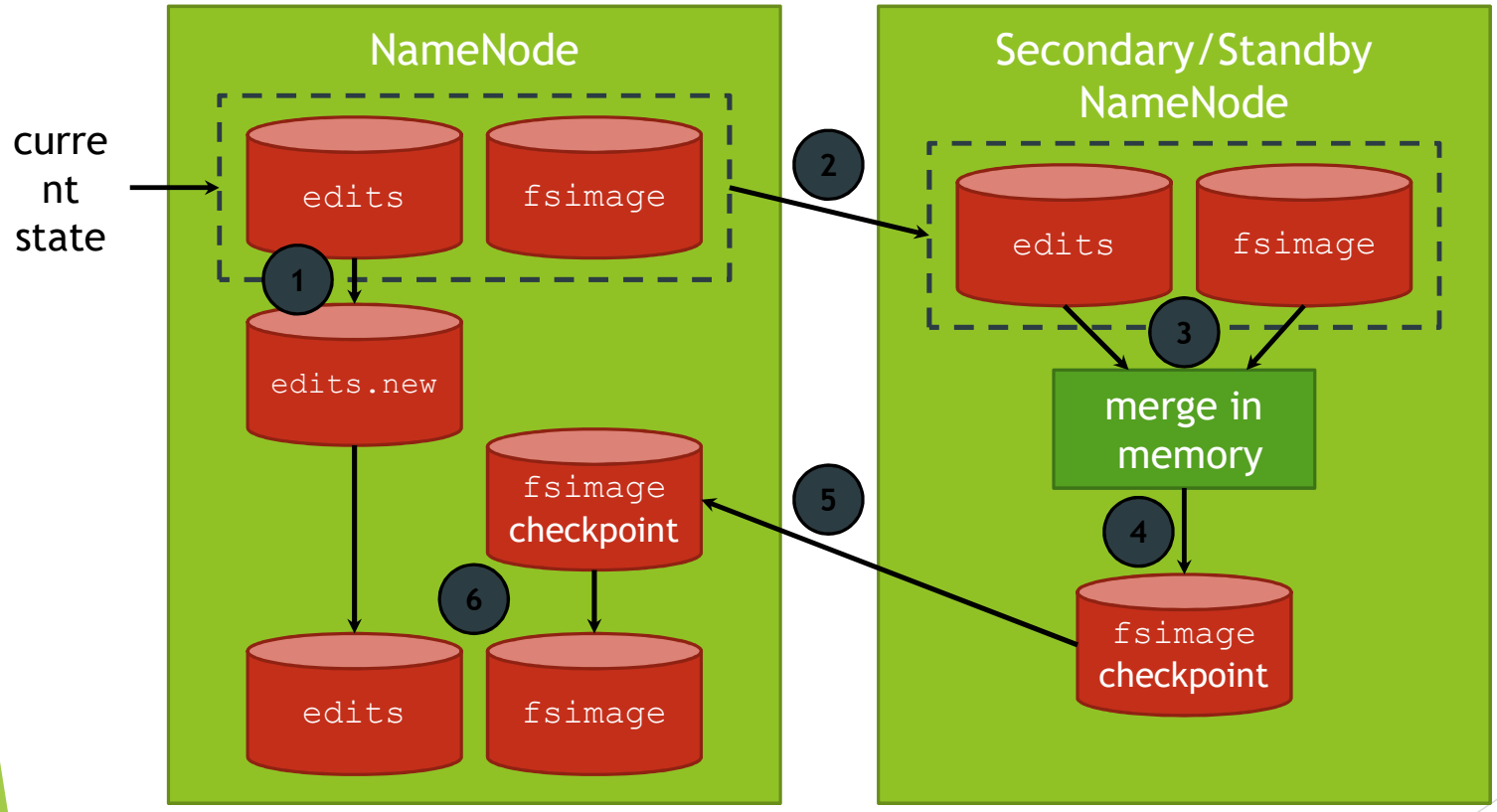


1. NameNode starts in safemode
1. Latest `fsimage` and `edits` read from disk
1. `edits` and `fsimage` files merged in memory
1. New `fsimage` and `edits` files created
1. Block lists retrieved from DataNodes and block map rebuilt
1. NameNode exits safemode

NameNode Checkpoint Operation

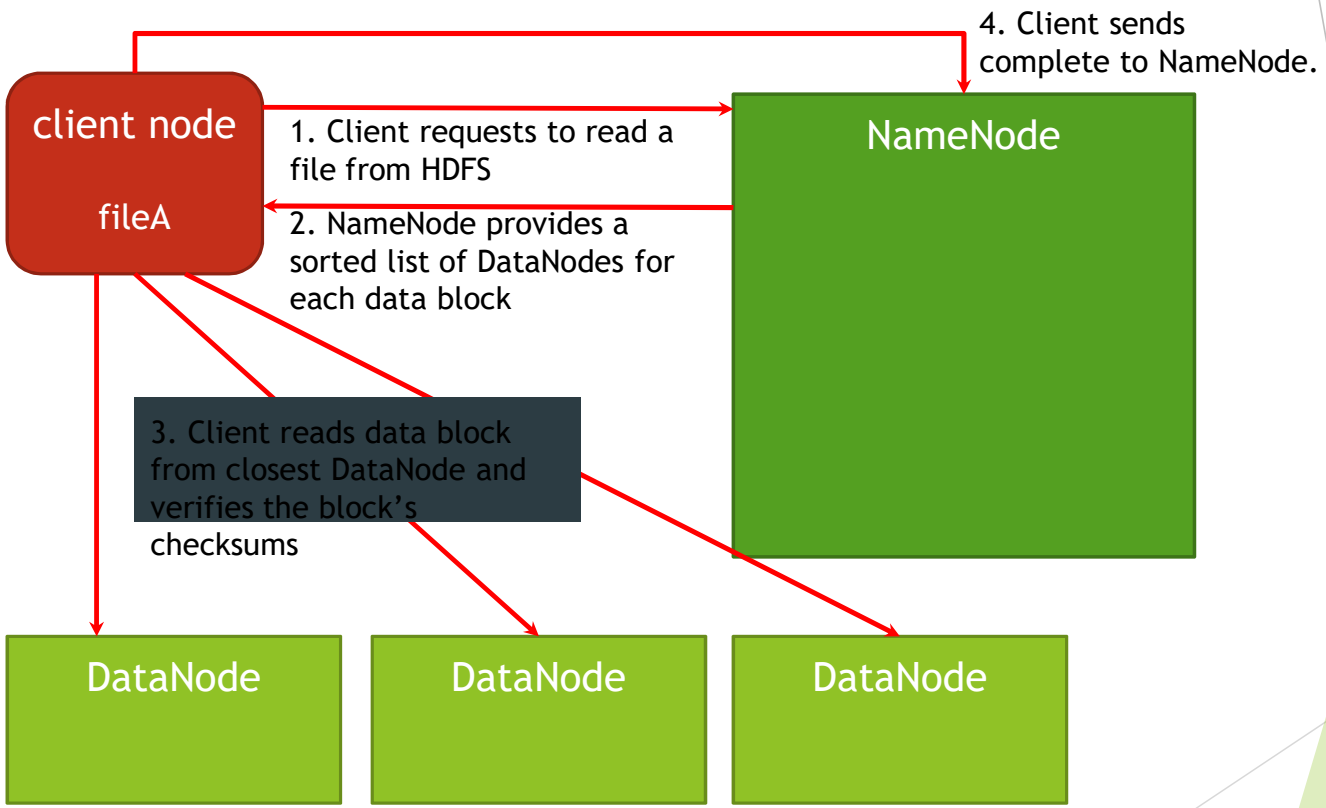
- NameNodes must periodically perform a checkpoint operation or the edits file would continue to grow without bounds.
- A checkpoint operation merges the changes recorded in the current edits file with the information in the current fsimage file, and then replaces the edits and fsimage files with a new files.
- The new edits file will initially be empty

NameNode Checkpoint Operation

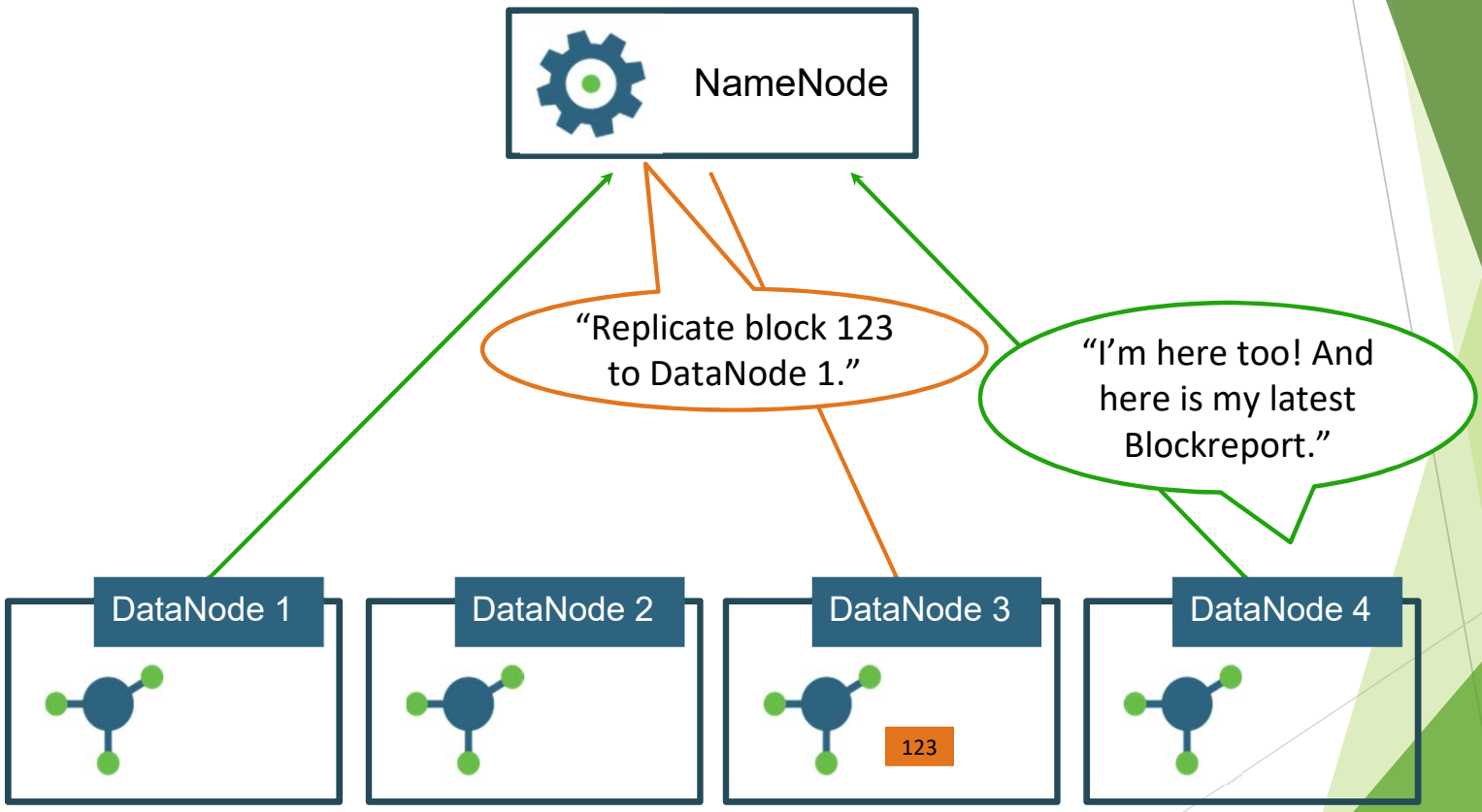


1. Primary creates and uses new `edits` file
1. Secondary/Standby retrieves `edits` and `fsimage` files
1. The `edits` and `fsimage` files merged in memory
1. New `fsimage` created
1. New `fsimage` sent to Primary
1. Primary saves new `fsimage` and continues using new `edits` file

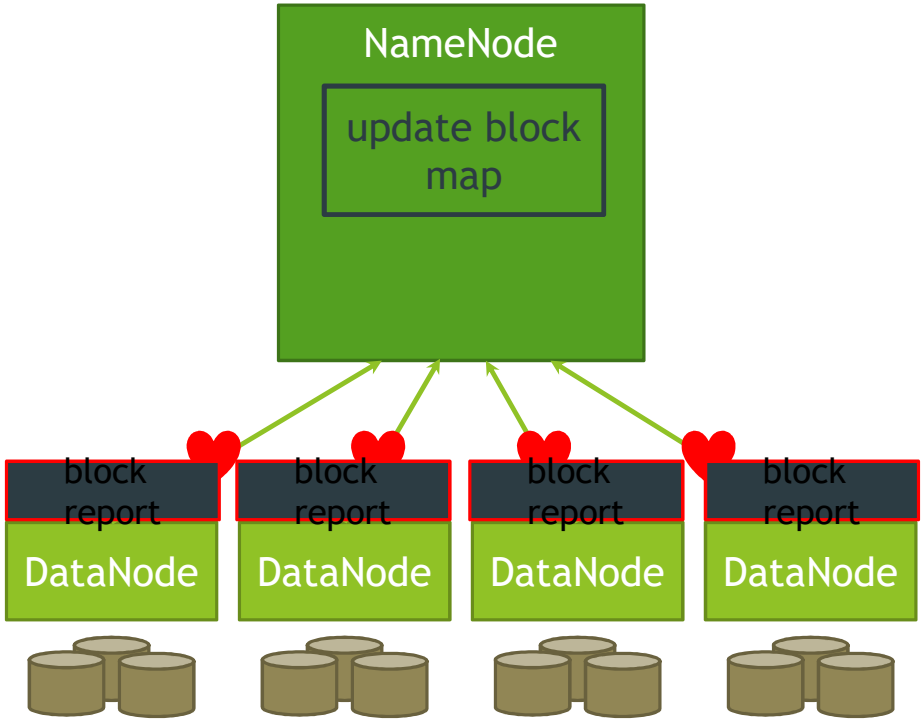
Reading Data



The DataNode Block Reports

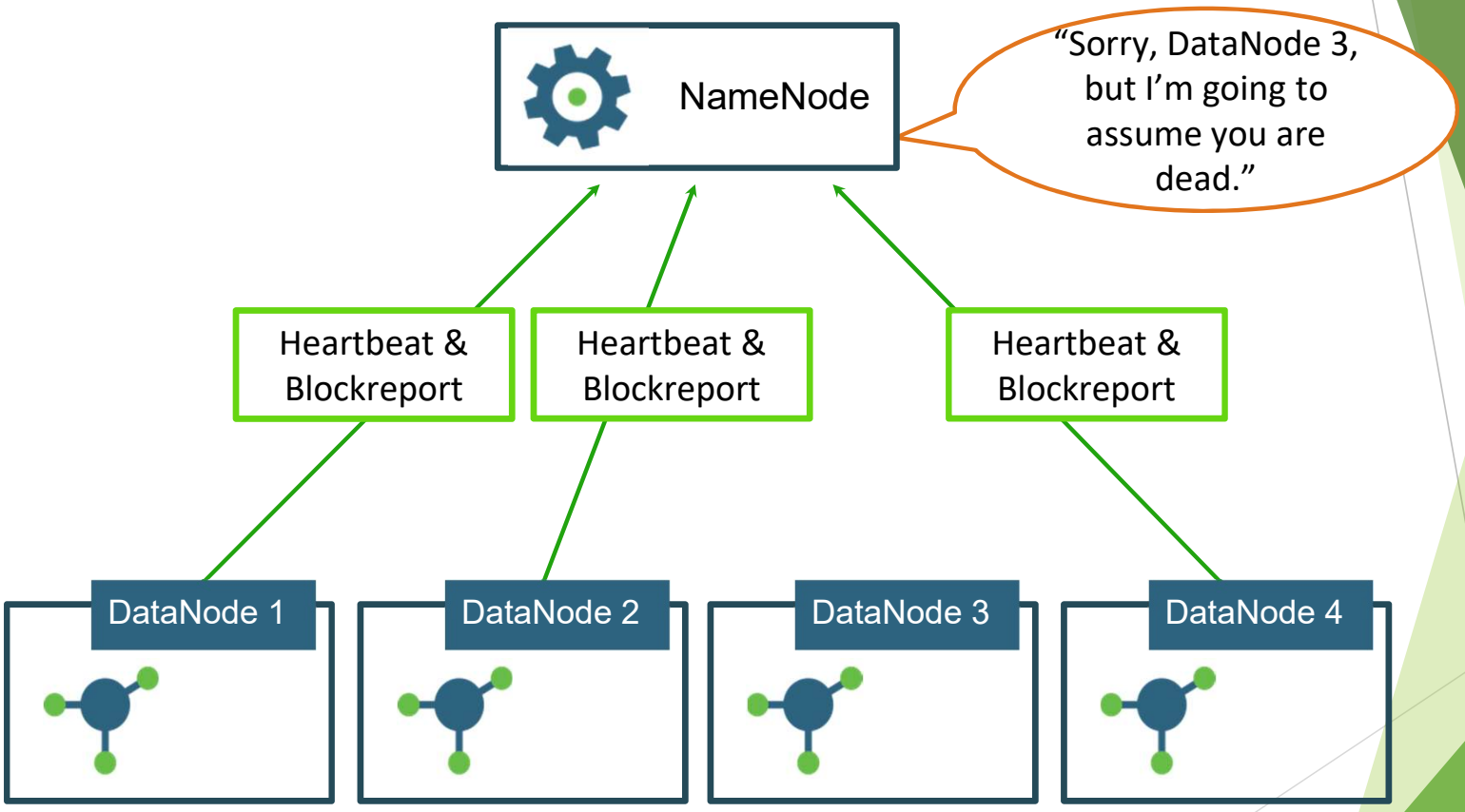


DataNode Block Reports – Detailed View

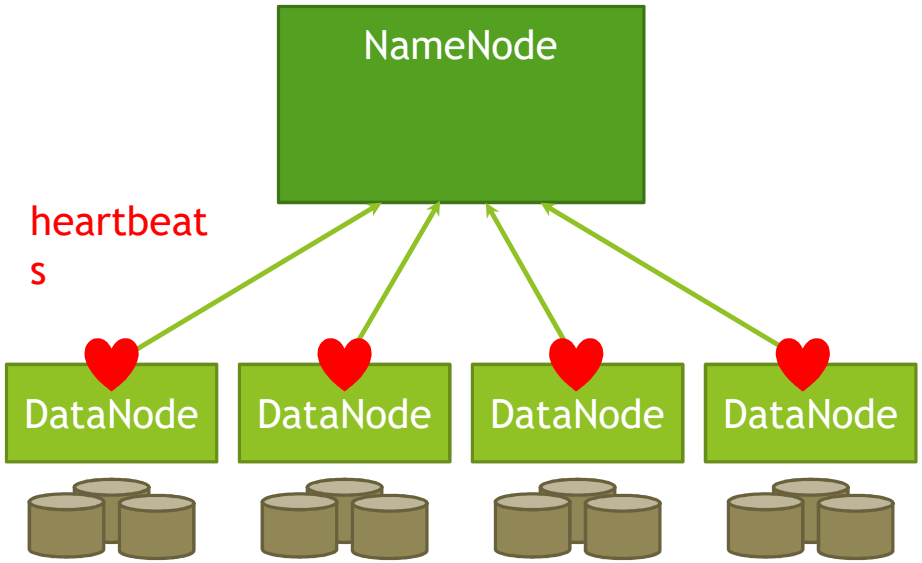


- At DataNode startup, a block report is sent to the NameNode after 3 minutes.
  - Determined by:
    - `dfs.blockreport.initialDelay = 120`
- Updated block reports are set every 6 hours at part of a heartbeat:
  - Determined by:
    - `dfs.blockreport.intervalMsec = 21600000`
- If the number of blocks is large, the report is split across multiple heartbeats.
  - `dfs.blockreport.split.threshold = 1000000`

DataNode Failure

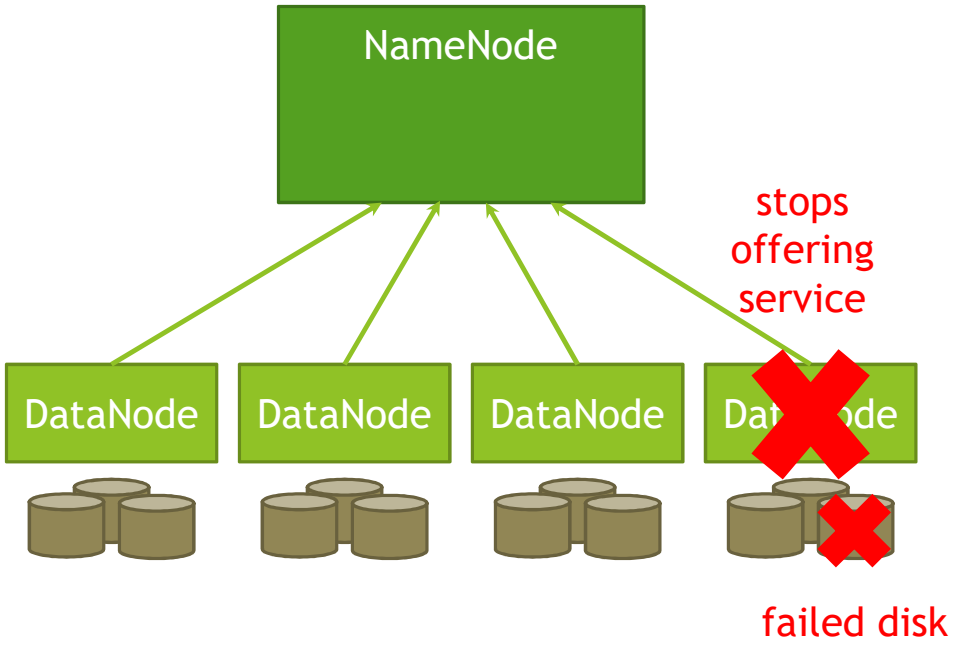


DataNode Failure – Detailed View



- A NameNode listens for DataNode heartbeats to determine availability.
  - A DataNode heartbeats every 3 seconds.
    - `dfs.heartbeat.interval`
- If heartbeats are not received, a DataNode is:
  - Declared stale after 30 seconds and used last
    - `dfs.namenode.stale.datanode.interval`
  - Declared dead after 10.5 minutes and not used
    - `dfs.namenode.heartbeat.recheck-interval` and `dfs.heartbeat.interval`
- A dead DataNode forces the NameNode to re-replicate the data blocks.

Failed DataNode Disks



- A DataNode typically has multiple disks to:
  - Enhance I/O performance
  - Create more available HDFS storage space
- More disks create more opportunity for failure.
- By default, a failed disk will cause a DataNode to stop offering service.
- Can modify `dfs.datanode.failed.volumes.tolerated` to make a DataNode tolerant of one or more failed disks.
  - `0` by default

HDFS Commands

```
hdfs dfs -command [args]
```

Here are a few (of the almost 30) HDFS commands:

`-cat`: display file content (uncompressed)

`-text`: just like cat but works on compressed files

`-chgrp,-chmod,-chown`: changes file permissions

`-put,-get,-copyFromLocal,-copyToLocal`: copies files from the local file system to the HDFS and vice versa.

`-ls, -ls -R:` list files/directories

`-mv,-moveFromLocal,-moveToLocal`: moves files

`-stat`: statistical info for any given file (block size, number of blocks, file type, etc.)

Examples of HDFS Commands

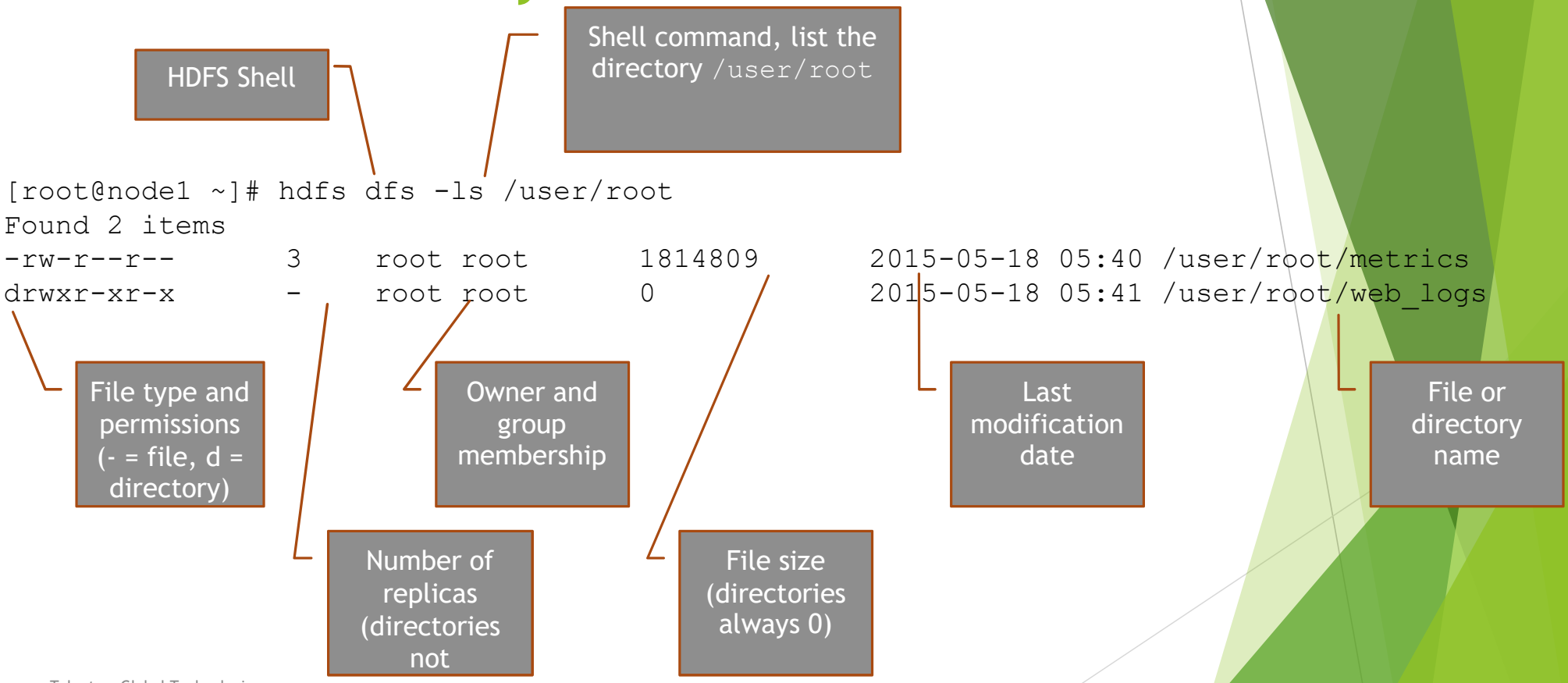
```
hdfs dfs -mkdir mydata

hdfs dfs -put numbers.txt
mydata/

hdfs dfs -ls mydata
```

HDFS File Permissions

- Files and directories have owners and groups
- r = read
- w = write
- x = permission to access the contents of a directory

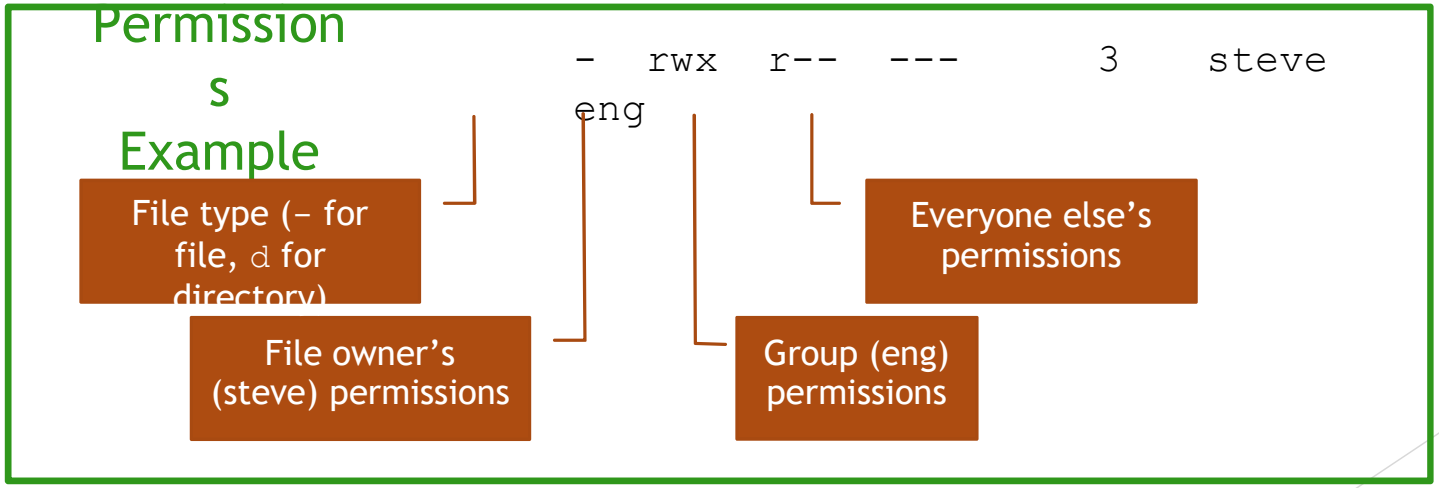
```
drwxr-xr-x   - hue      hue          0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/shell
-rwxr-xr-x   3 hue      hue         77 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/shell/hello.py
drwxr-xr-x   - hue      hue          0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sleep
-rwxr-xr-x   3 hue      hue          0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sleep/empty
drwxr-xr-x   - hue      hue          0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sqoop
-rwxr-xr-x   3 hue      hue       7175 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sqoop/TT.java
-rwxr-xr-x   3 hue      hue        420 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sqoop/db.hsqldb.properties
-rwxr-xr-x   3 hue      hue        276 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/sqoop/db.hsqldb.script
drwxr-xr-x   - hue      hue          0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/ssh
-rwxr-xr-x   3 hue      hue          0 2013-08-27 23:00 /user/hue/oozie/workspaces/unmanaged/ssh/empty
drwxr-xr-x   - root     root         0 2013-08-29 03:22 /user/root
drwxr-xr-x   - root     root         0 2013-08-29 03:23 /user/root/mydata
-rw-r--r--   3 root     root      2549 2013-08-29 03:23 /user/root/mydata/numbers.txt
-rw-r--r--   3 root     root   3613198 2013-08-28 21:55 /user/root/stocks.csv
[root@sandbox demos]#
```

File and Directory Attributes



HDFS Permissions

Permission	Authorized Directory Actions	Authorized File Actions
r = read	View (list) directory contents	View file contents
w = write	Create or delete files or subdirectories	Write, or append to, file contents
x = execute	Access a directory	Ignored for HDFS



Permissions are applied according to the most specific user class applicable to a user.

HDFS Home Directories

- Users and applications might have a home directory.
- Home directories are used in concert with permissions to control data access.

Only the Hive application can write to its home directory.

Only Saad can write to his home directory.

```
[hdfs@node1 ~]$ hdfs dfs -ls /user
Found 9 items
drwxrwx---   - ambari-qa hdfs          0 2015-05-15 12:40 /user/ambari-qa
drwxr-xr-x   - hcat      hdfs          0 2015-05-15 12:41 /user/hcat
drwx------   - hive      hdfs          0 2015-05-15 12:37 /user/hive
drwxr-xr-x   - jason     eng           0 2015-05-18 07:58 /user/jason
drwxr-xr-x   - may       sales         0 2015-05-18 07:58 /user/may
drwxrwxr-x   - oozie     hdfs          0 2015-05-15 12:38 /user/oozie
drwxr-xr-x   - root      root          0 2015-05-18 05:43 /user/root
drwxr-xr-x   - saad      sales         0 2015-05-18 07:58 /user/saad
drwxr-xr-x   - steve     eng           0 2015-05-18 07:58 /user/steve
```

Members of the sales group have read-only access to Saad's home directory.

Lab: Using HDFS Commands

HDFS Management Options

- There are several options for managing HDFS:

Option	Description
Ambari Web UI	Browser-based, HDFS configuration and service management interface
NameNode UI	Browser-based interface for basic status monitoring and directory browsing
DataNode UI	Browser-based interface, most commonly used to get block scanner reports (a scanner report is shown later)
HDFS command-line tools	Various command-line tools to interact with the HDFS service and its files, directories, and metadata (described later)
Manual configuration	Manually editing configuration files (**not** compatible with Ambari administration)

Command-Line Management

- Introduction to command-line management tools:

Command	Description
`hdfs dfs`	HDFS Shell to manage files, directories, and their metadata
`hdfs fsck`	Checks and reports on file system inconsistencies (does not repair)
`hdfs dfsadmin`	Reports basic file system information and statistics and performs various file system administration tasks

Determining Storage Space Consumed

- The HDFS Shell `du` command reports the number of bytes consumed by a file or directory. (Does not account for replication)
- Syntax: `hdfs dfs -du [-s] [-h] [path]`
- Examples:

```
[root@node1 ~]# hdfs dfs -du
1520  dir1
0     dir3
1520  passwd
12    textfile.txt
[root@node1 ~]# hdfs dfs -du dir1
0     dir1/dir2
1520  dir1/passwd
[root@node1 ~]# hdfs dfs -du -s dir1
1520  dir1
[root@node1 ~]# hdfs dfs -du -h dir1
0      dir1/dir2
1.5 K  dir1/passwd
[root@node1 ~]#
```

Summary-only of `dir1`

"Human" readable format, K, M, G, and T bytes instead of bytes

Monitoring File System Space

- The HDFS Shell `df` command reports the file system's total capacity, along with the current amount of free and used storage space.
- Syntax: `hdfs dfs –df [-h]`
- Examples:

```
[root@node1 ~]# hdfs dfs -df
Filesystem                Size       Used    Available  Use%
hdfs://node1:8020  100000174080  461144064  85128048640    0%
[root@node1 ~]# hdfs dfs -df -h
Filesystem           Size     Used  Available  Use%
hdfs://node1:8020  93.1 G  439.8 M     79.3 G    0%
[root@node1 ~]#
```

"Human" readable format, K, M, G, and T bytes instead of bytes

Checking File System Consistency

- The HDFS `fsck` command checks file system consistency.
- Run `fsck` when:
  - There is concern about possible file (data block) corruption
    - After an HDFS or hardware malfunction
  - Prior to upgrading HDFS to a newer version
- `fsck` does not repair data blocks.
  - It only reports, unlike Linux `fsck`
- An `fsck` reads block and metadata information from only the NameNode.
  - DataNodes are never contacted by `fsck`.
- Must have access permissions to the directories and files being checked
  - The HDFS superuser has access to all files and directories.

`fsck` Syntax

- Syntax:
  - `hdfs fsck [path] [options] [> <output_file>]`

Options	Description
`-files`	Reports a list of file and directories checked
`-blocks`	Reports block ID numbers checked (requires `-files -blocks` syntax)
`-locations`	Reports a list of DataNodes locations for each block ID number (requires `-files -blocks -locations` syntax)
`-racks`	Prepends the rack name on each reported DataNode location (requires at least `-files -blocks -racks` syntax). Really only useful if HDFS rack awareness has been configured (described in another lesson).
`-move`	Moves files with corrupted data blocks to the `/lost+found` directory
`-delete`	Deletes files with corrupted data blocks
`-openforwrite`	List files open for write during `fsck` (open files are not checked)

Understanding `fsck` Output

`fsck` reports:

- **Minimally replicated blocks:**
  - Blocks having at least one good replica
- **Over-replicated blocks:**
  - Blocks that exceed the file's replication factor (NameNode will delete)
- **Under-replicated blocks:**
  - Blocks that do not meet the file's replication factor (NameNode will replicate)
- **Mis-replicated blocks:**
  - Blocks replicated more than once on the same DataNode (NameNode will move)
- **Corrupt blocks:**
  - Blocks where all replicas report checksums errors (NameNode will **not** repair)
  - User action required!

The Primary Output

- `hdfs fsck /user/root`

```
Status: HEALTHY
 Total size:    4829660 B
 Total dirs:    6
 Total files:   5
 Total symlinks:                0
 Total blocks (validated):      7 (avg. block size 689951 B)
 Minimally replicated blocks:   7 (100.0 %)
 Over-replicated blocks:        0 (0.0 %)
 Under-replicated blocks:       0 (0.0 %)
 Mis-replicated blocks:         0 (0.0 %)
 Default replication factor:    3
 Average block replication:     3.0
 Corrupt blocks:                0
 Missing replicas:              0 (0.0 %)
 Number of data-nodes:          3
 Number of racks:               1
FSCK ended at Tue May 19 14:25:40 EDT 2015 in 2 milliseconds
```

HEALTHY status

No corrupt blocks

The `-files` Option

- `hdfs fsck /user/root -files`

```
/user/root <dir>
/user/root/.Trash <dir>
/user/root/ambari-metrics 1814809 bytes, 1 block(s):  OK
/user/root/big1 3011212 bytes, 3 block(s):  OK
/user/root/dir1 <dir>
/user/root/dir1/hosts 173 bytes, 1 block(s):  OK
/user/root/dir1/passwd 1733 bytes, 1 block(s):  OK
/user/root/dir2 <dir>
/user/root/dir2/dir3 <dir>
/user/root/passwd 1733 bytes, 1 block(s):  OK
/user/root/web_logs <dir>
```

Prepends a list of files and directories to the primary output.

The `-blocks` Option

- `hdfs fsck /user/root -files -blocks`
- The file `big1` has three blocks, each with a unique block ID.
  - HFDS generated block pool ID: BP-1472918407-172.17.0.2-1431707688874
    - The same block pool across all DataNodes
  - Data block ID: blk_1073742266_1442, and two others
    - The same ID for all of a block's replicas

```
/user/root/big1 3011212 bytes, 3 block(s):  OK
0. BP-1472918407-172.17.0.2-1431707688874:blk_1073742266_1442 len=1048576 repl=3
1. BP-1472918407-172.17.0.2-1431707688874:blk_1073742267_1443 len=1048576 repl=3
2. BP-1472918407-172.17.0.2-1431707688874:blk_1073742268_1444 len=914060 repl=3
```

For each file listed by `-files`, append a block ID number, size, and the replication factor.

The `-locations` Option

- `hdfs fsck /user/root -files -blocks -locations`

```
/user/root/big1 3011212 bytes, 3 block(s):  OK
0. BP-1472918407-172.17.0.2-1431707688874:blk_1073742266_1442 len=1048576 repl=3
 [172.17.0.2:50010, 172.17.0.3:50010, 172.17.0.4:50010]
1. BP-1472918407-172.17.0.2-1431707688874:blk_1073742267_1443 len=1048576 repl=3
 [172.17.0.2:50010, 172.17.0.3:50010, 172.17.0.4:50010]
2. BP-1472918407-172.17.0.2-1431707688874:blk_1073742268_1444 len=914060 repl=3
[172.17.0.2:50010, 172.17.0.4:50010, 172.17.0.3:50010]
```

For each data block, list the DataNodes that contain a replica.

The `-racks` Option

- `hdfs fsck /user/root -files -blocks -locations -racks`
- Rack name is `/default-rack` if rack awareness is not configured.
  - Rack awareness and rack naming are described in another lesson.

```
/user/root/big1 3011212 bytes, 3 block(s):  OK
0. BP-1472918407-172.17.0.2-1431707688874:blk_1073742266_1442 len=1048576 repl=3
 [/default-rack/172.17.0.2:50010, /default-rack/172.17.0.3:50010, /default-rack/
172.17.0.4:50010]
1. BP-1472918407-172.17.0.2-1431707688874:blk_1073742267_1443 len=1048576 repl=3
 [/default-rack/172.17.0.2:50010, /default-rack/172.17.0.3:50010, /default-rack/
172.17.0.4:50010]
2. BP-1472918407-172.17.0.2-1431707688874:blk_1073742268_1444 len=914060 repl=3
[/default-rack/172.17.0.2:50010, /default-rack/172.17.0.4:50010, /default-rack/1
72.17.0.3:50010]
```

Prepend a rack name to each DataNode listed.

Distributed File System Administration Command

- `dfsadmin` is a set of HDFS administration tools.
  - Ambari is gaining more and more of `dfsadmin` functionality.
- Syntax: `hdfs dfsadmin [options]`
  - Over 30 options, only a few options are shown here.
  - Getting more information and help: `hdfs dfsadmin -help`
- You must be the HDFS superuser.

dfsadmin Examples

- Transition a NameNode into safe mode:
  - `hdfs dfsadmin -safemode enter`
- Force a NameNode checkpoint (generates new `fsimage` and `edits` files)
  - `hdfs dfsadmin -saveNamespace`
- Or create only a new `edits` file:
  - `hdfs dfsadmin -rollEdits`
- Exit NameNode safe mode:
  - `hdfs dfsadmin -safemode leave`
- Download the latest `fsimage` file (useful for doing remote backups):
  - `hdfs dfsadmin -fetchImage`

Some of these commands are required when configuring NameNode HA.

Heath, Status, and Usage Reports

- `hdfs dfsadmin -report` can display status and usage information similar to the NameNode UI.

The summary section:

```
[hdfs@node1 ~]$ hdfs dfsadmin -report
Configured Capacity: 300000522240 (279.40 GB)
Present Capacity: 221331869696 (206.13 GB)
DFS Remaining: 219495591936 (204.42 GB)
DFS Used: 1836277760 (1.71 GB)
DFS Used%: 0.83%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0

-------------------------------------------------
```

A section for each DataNode:

```
Live datanodes (3):

Name: 172.17.0.4:50010 (node3)
Hostname: node3
Decommission Status : Normal
Configured Capacity: 100000174080 (93.13 GB)
DFS Used: 612110336 (583.75 MB)
Non DFS Used: 26222866432 (24.42 GB)
DFS Remaining: 73165197312 (68.14 GB)
DFS Used%: 0.61%
DFS Remaining%: 73.17%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 2
Last contact: Tue May 19 19:51:27 EDT 2015
```

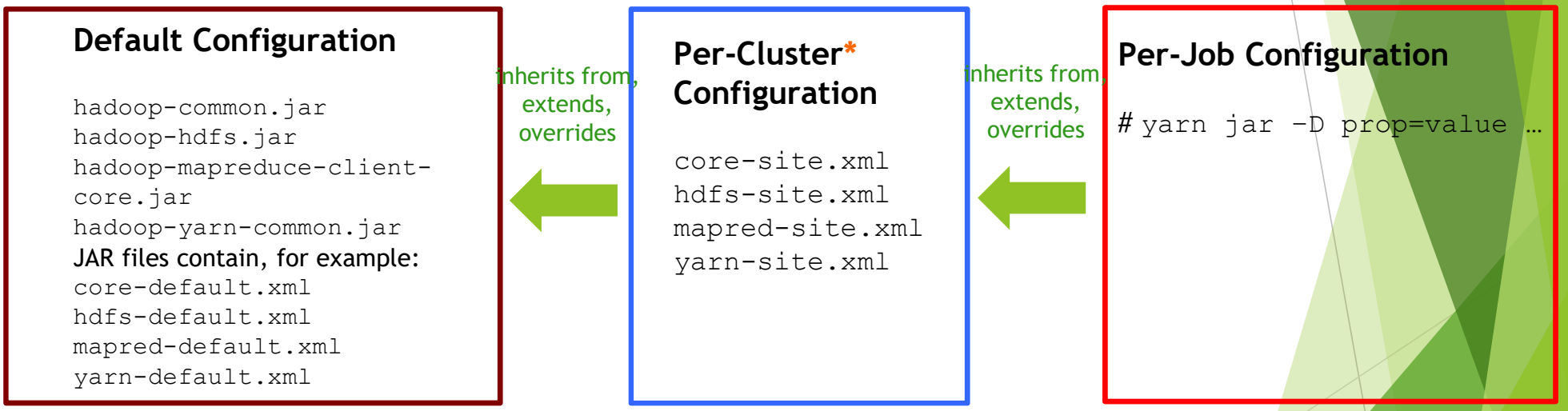
Core Hadoop Configuration Files

- Ambari installs the core Hadoop configuration files in `/etc/hadoop/conf`.

File Name	File Format	File Purpose
`core-site.xml`	Hadoop configuration XML	Hadoop core configuration settings that can be used by HDFS, YARN, MapReduce, and others
`hdfs-site.xml`	Hadoop configuration XML	HDFS configuration settings (NameNode and DataNode)
`yarn-site.xml`	Hadoop configuration XML	YARN configuration settings
`mapred-site.xml`	Hadoop configuration XML	MapReduce configuration settings
`hadoop-env.sh`	Bash script	Environment variables used by various Hadoop scripts and programs
`log4j.properties`	Java properties	System log file configuration settings

Configuration Precedence

- A running job's actual configuration is a combination of the default, per-site, possibly per-node, and per-job configuration.



*Cluster nodes with different hardware configurations commonly need different `*-site.xml` files.

Final Properties

- *Final* properties cannot be overridden by user applications.
  - For example, no `-D prop=value`

Using Ambari Web UI

Click to toggle on or off (dark gray or light gray)

Editing the Configuration File

```
<property>
        <name>dfs.datanode.data.dir<
/name>
        <value>/hadoop/hdfs/data</va
lue>
        <final>true</final>
</property>
```

*Using Ambari to view and modify property settings is described in more detail later in this lesson.

Other Framework Configuration Files

- Other Hadoop frameworks often use configuration files with similar formats and naming conventions.
  - Examples: `*-default.xml`, `*-site.xml`, `*-env.sh`, `*-log4j.properties`
- Other frameworks use their own dedicated configuration directories:
  - `/etc/ambari-server/conf`
  - `/etc/ambari-agent/conf`
  - `/etc/hive/conf`
  - `/etc/pig/conf`
  - `/etc/zookeeper/conf`
  - and so on…

Configuration Management Options

- Hadoop includes several options for configuration management:

Option	Description	Benefit
Ambari Web UI	Browser-based graphic user management interface	Ease of use, pre-built and ready to go
REST APIs: Ambari, WebHDFS, YARN, etc.	HTTP verb (GET, PUT, POST, DELETE) management interface	Integration with other web-based management interfaces, can be used for testing and troubleshooting cluster
Manual editing	Manually edit and distribute configuration files, manually restart services	No reliance on graphic user interface, no need to install Ambari, *not* compatible with Ambari management
Command-line	Per-framework command-line management utilities	Scriptable, no reliance on a graphic user interface

Lesson Review

1. Which component of HDFS is responsible for maintaining the namespace of the distributed filesystem?

1. What is the default file replication factor in HDFS?

1. **True or False:** To input a file into HDFS, the client application passes the data to the NameNode, which then divides the data into blocks and passes the blocks to the DataNodes.

1. Which property is used to specify the block size of a file stored in HDFS?

1. The NameNode maintains the namespace of the filesystem using which two sets of files?

1. What does the following command do? **hdfs dfs -ls -R /user/thomas/**

1. What does the following command do? **hdfs dfs -ls /user/thomas/**

Ingesting Data into HDFS

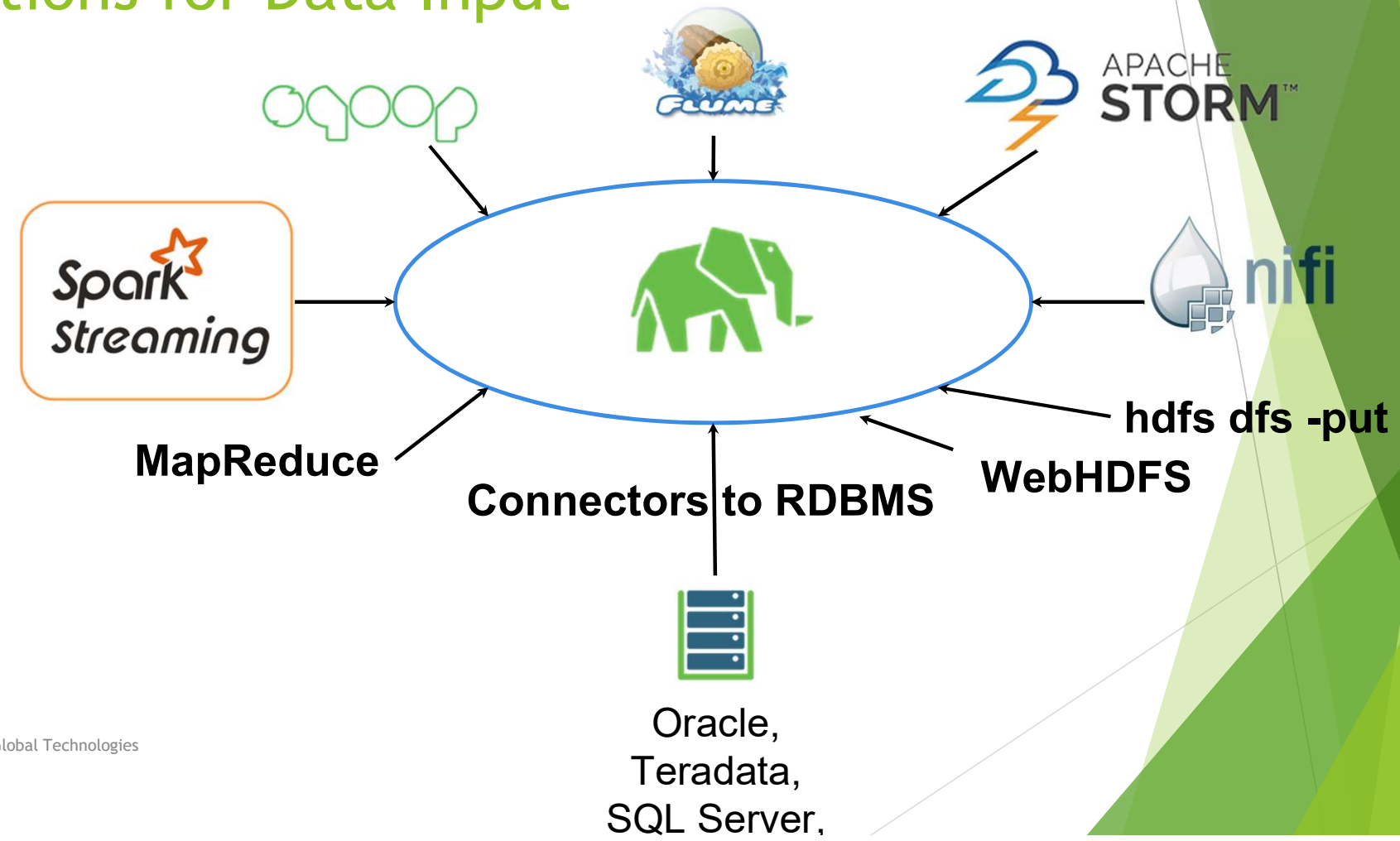
Topics Covered

- Options for Data Input
- The Hadoop Client
- WebHDFS
- Overview of Sqoop
- Importing a Data
- The Sqoop Export Tool
- Exporting to a Table
- *Labs: HDFS Importing/Exporting from/to RDBMS using Sqoop*
- *Lab: Importing Log Data into HDFS using Flume*

What is Ingestion in Big Data?

Big Data Ingestion involves connecting to various data sources, extracting the data, and detecting the changed data.

Options for Data Input



The Hadoop Client

The **put** Command

- Same as **copyFromLocal**

Perfect for inputting local files into HDFS

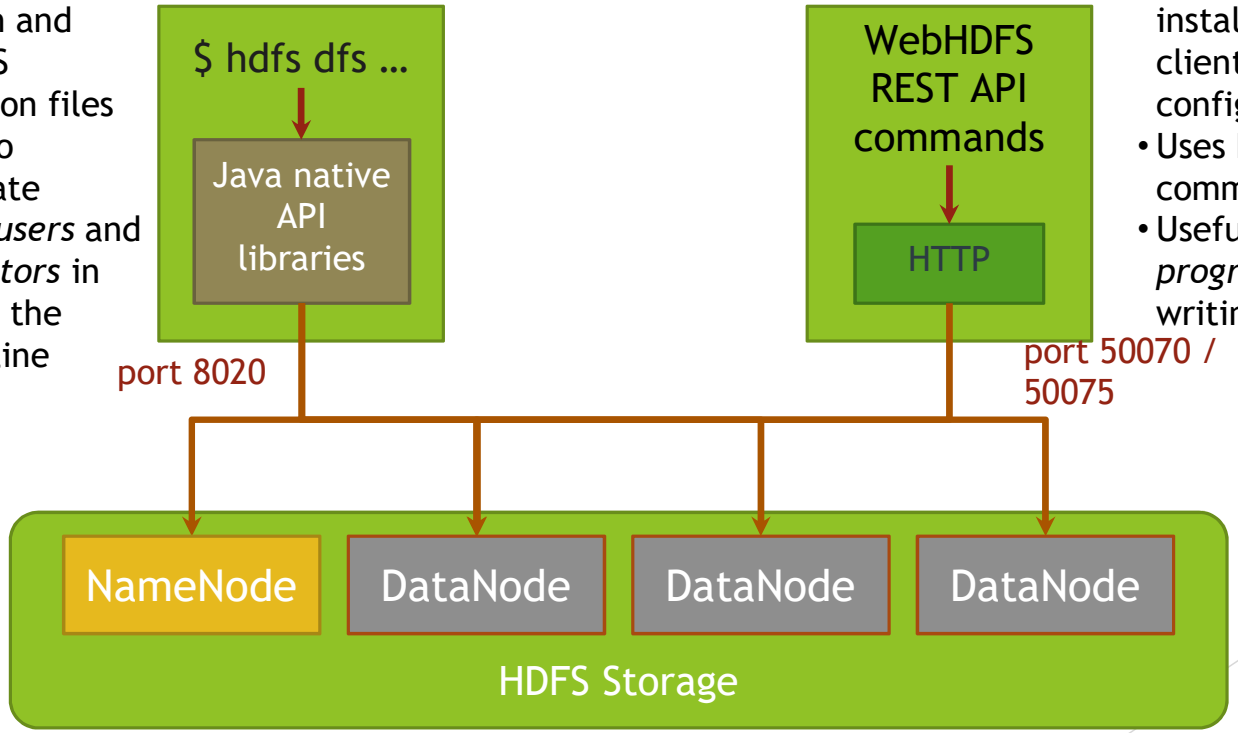
- Useful in batch scripts

Usage:

```
hdfs dfs -put <localsrc> ... <dst>
```

Java Native API Versus WebHDFS Access

- Requires installation and client HDFS configuration files
- Uses RPC to communicate
- Useful for *users* and *administrators* in scripts and the command line



- Requires no installation or client configuration files
- Uses HTTP to communicate
- Useful for *programmers* writing Web apps

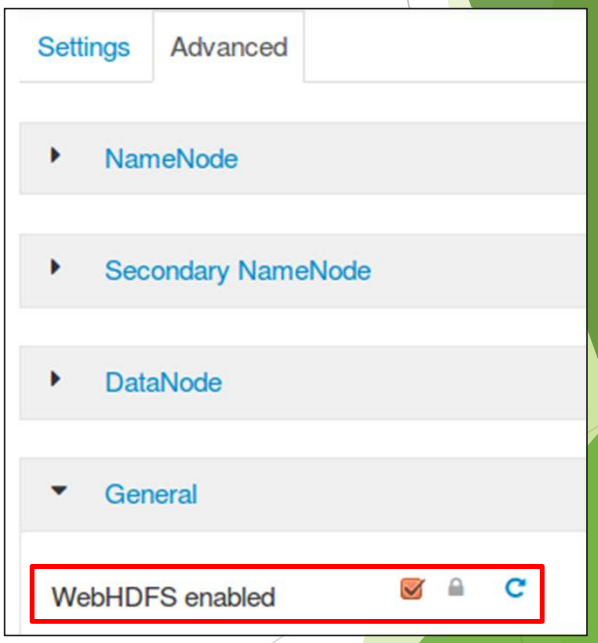
WebHDFS Features

- Supports all HDFS file administration operations
- Enables access to HDFS from programming languages other than Java
  - API access is through Java only.
- Enables faster access than `hdfs dfs` when the client is remote to the cluster
- Requires no additional servers
  - WebHDFS is built into the NameNode and DataNode
- Uses the full bandwidth of the Hadoop cluster for moving data
  - Read and write operations are redirected to the appropriate DataNodes.
- Is compatible with Kerberos authentication
  - Uses Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO), which extends Kerberos to Web applications
- Is completely open source

WebHDFS Enabled by Default

- To verify that WebHDFS is enabled, check either the `hdfs-site.xml` file or Ambari.

In Ambari



In `hdfs-site.xml`

```
<property>
      <name>dfs.webhdfs.enabled</name>
      <value>true</value>
   </property>
```

WebHDFS Operations

- The following WebHDFS operations, formatted using the proper URIs, enable HDFS file access and administration.

HTTP GET	HTTP PUT	HTTP POST	HTTP DELETE
OPEN	CREATE	APPEND	DELETE
GETFILESTATUS	MKDIRS		
LISTSTATUS	RENAME		
GETCONTENTSUMMARY	SETREPLICATION		
GETFILECHECKSUM	SETOWNER		
GETHOMEDIRECTORY	SETPERMISSION		
GETDELEGATIONTOKEN	SETTIMES		
	RENEWDELEGATIONTOKEN		
	CANCELDELEGATIONTOKEN		

WebHDFS Examples (1)

- All programs and applications performing WebHDFS operations use the URI syntax:
  - http://<NameNode>:50070/webhdfs/v1/<path>?op=<operation_and_arguments>

WebHDFS API prefix

- The `curl` command can be used to test WebHDFS operations.
- Creating a directory named `mydata`:
  - `curl -i -X PUT "http://<NameNode>:50070/webhdfs/v1/web/mydata?op=MKDIRS&user.name=jason"`
- Listing a directory named `mydata`:
  - `curl -i "http://<NameNode>:50070/webhdfs/v1/web/mydata?op=LISTSTATUS&user.name=jason"`
- Reading a file named `webdata`:
  - `http://<NameNode>:50070/webhdfs/v1/web/mydata/webdata?op=OPEN&user.name=jason"`

WebHDFS Examples (2)

- Writing a file is a two-step process.
  1. Create a file name on the NameNode.
  2. Write the file contents to a DataNode.
     - WebHDFS ensures that files larger than an HDFS block are written across multiple DataNodes.
- Create a file by creating a file name on the NameNode:
  - `curl -i -X PUT "http://<NameNode>:50070/webhdfs/v1/web/mydata/largefile.json?op=CREATE"`
  - The output from this command includes the URI used to write data to the file.
- Write to the file by sending data to the DataNodes:
  - `curl –i –X PUT –T largefile.json "http://<DataNode>:50075/webhdfs/v1/web/mydata/largefile.json?op=CREATE&user.name=root&namenoderpcaddress=node1:8020&overwrite=false"`
- Curl can perform a write operation using a single command that performs both steps:
  - `curl –i –X PUT largefile.json –L "http://<NameNode>:50070/webhdfs/v1/web/mydata/largefile.json?op=CREATE&user.name=root"`

WebHDFS

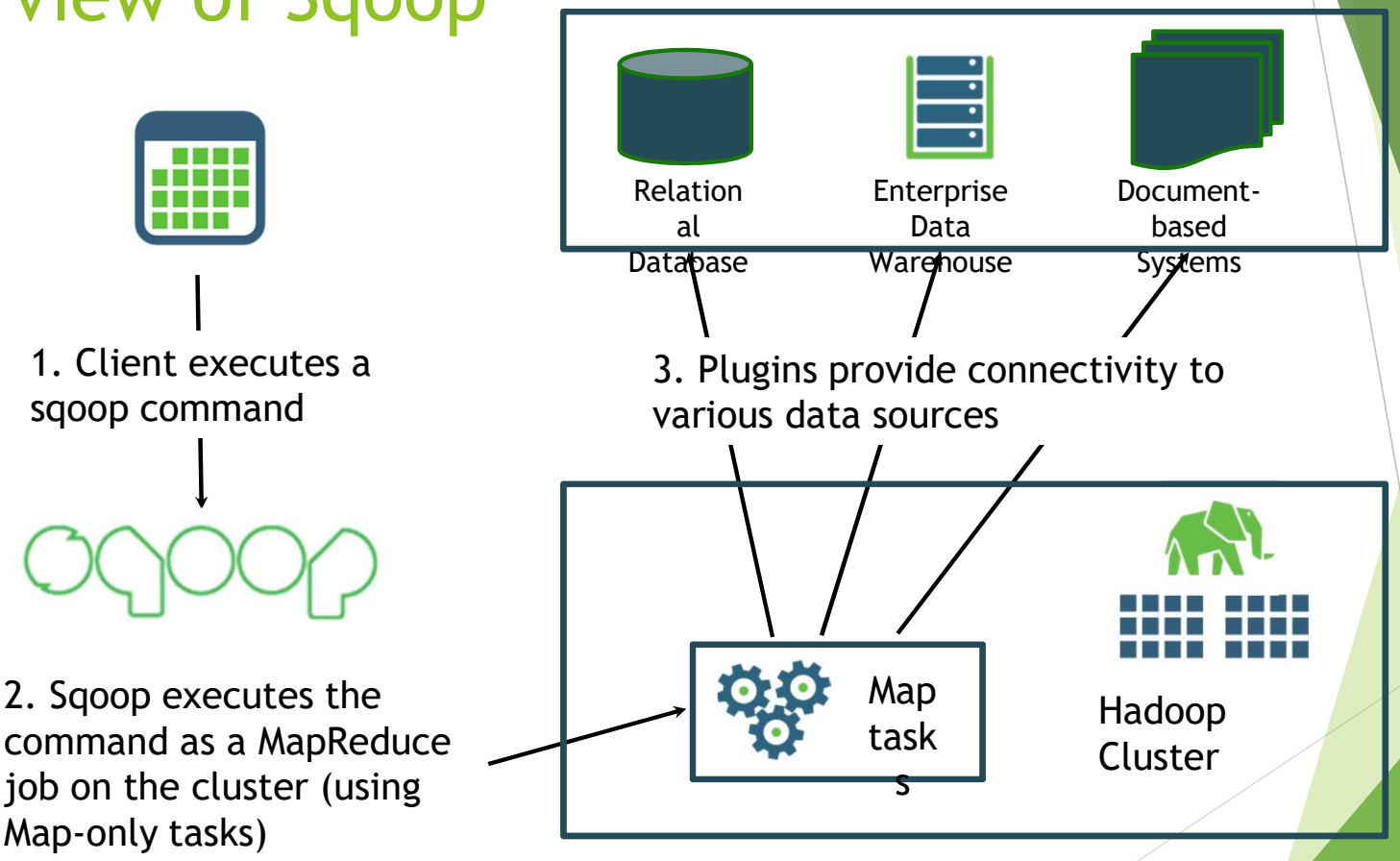
REST API for accessing all of the HDFS file system interfaces:

- http://*host*:*port*/webhdfs/v1/test/mydata.txt?op=OPEN
- http://*host*:*port*/webhdfs/v1/user/root/data?op=MKDIRS
- http://*host*:*port*/webhdfs/v1/test/mydata.txt?op=APPEND

Lab: Using WebHDFS Commands

Demo: Putting Files in HDFS with Java

Overview of Sqoop



The Sqoop Import Tool

The **import** command has the following requirements:

- Must specify a connect string using the **--connect** argument
- Credentials can be included in the connect string, so use the **--username** and **--password** arguments
- Must specify either a table to import using **--table** or the result of an SQL query using **--query**

Importing a Table

```
sqoop import
--connect jdbc:mysql://host/nyse
--table StockPrices
--target-dir /data/stockprice/
--as-textfile
```

Importing Specific Columns

```
sqoop import
--connect jdbc:mysql://host/nyse
--table StockPrices
--columns StockSymbol,Volume,
High,ClosingPrice
--target-dir /data/dailyhighs/
--as-textfile
--split-by StockSymbol
-m 10
```

Importing from a Query

```
sqoop import
--connect jdbc:mysql://host/nyse
--query "SELECT * FROM StockPrices s
WHERE s.Volume >= 1000000
AND \$CONDITIONS"
--target-dir /data/highvolume/
--as-textfile
--split-by StockSymbol
```