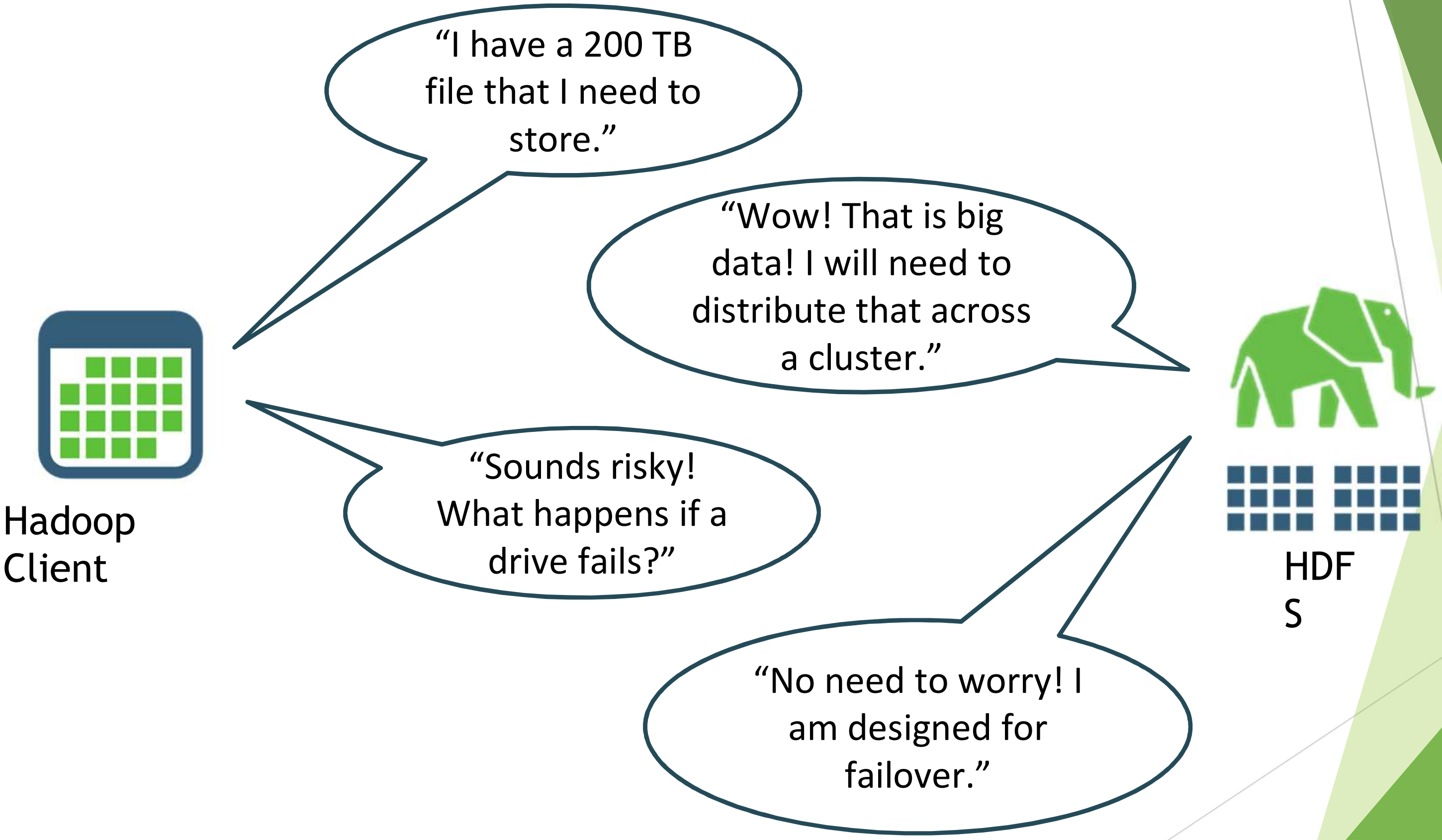
HDFS

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



Hadoop Client

"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."

HDF
S

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

500 GB
data file

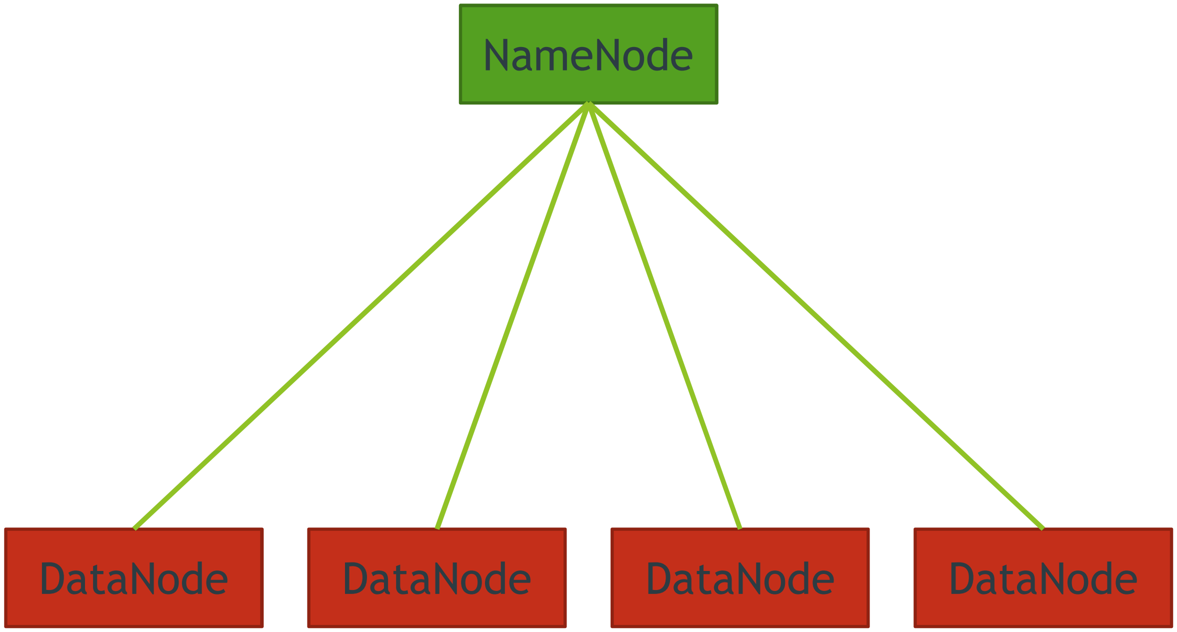
- Uses indexes to avoid reading an entire file (very fast lookups)
- Maintains a data structure in order to provide a fast execution layer
- Works well as long as the index fits in RAM

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

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

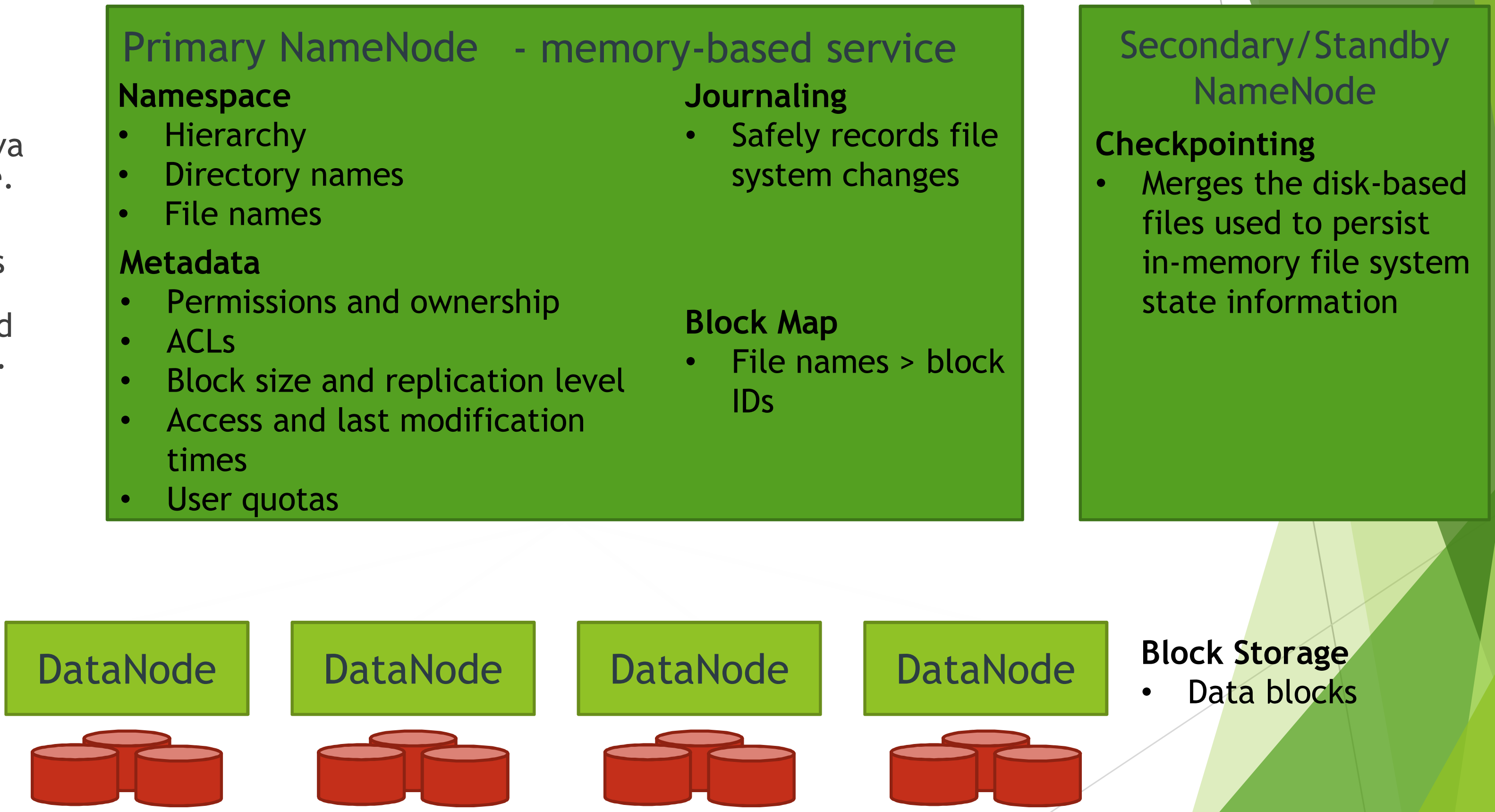
- Is the “master” node of HDFS
- Determines and maintains how the chunks of data are distributed across the DataNodes

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

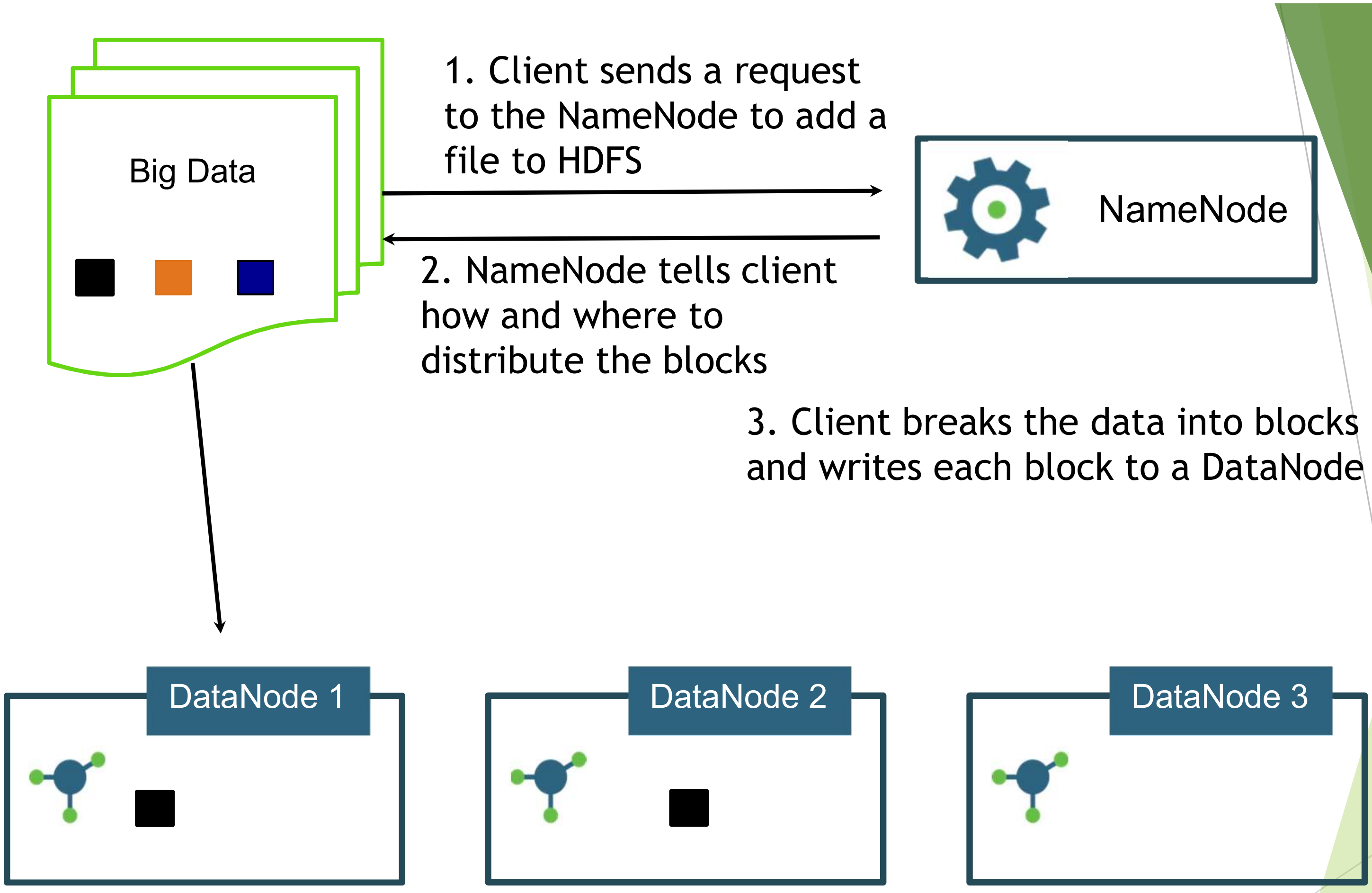
Secondary/Standby NameNode

Checkpointing

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

Block Storage

- Data blocks



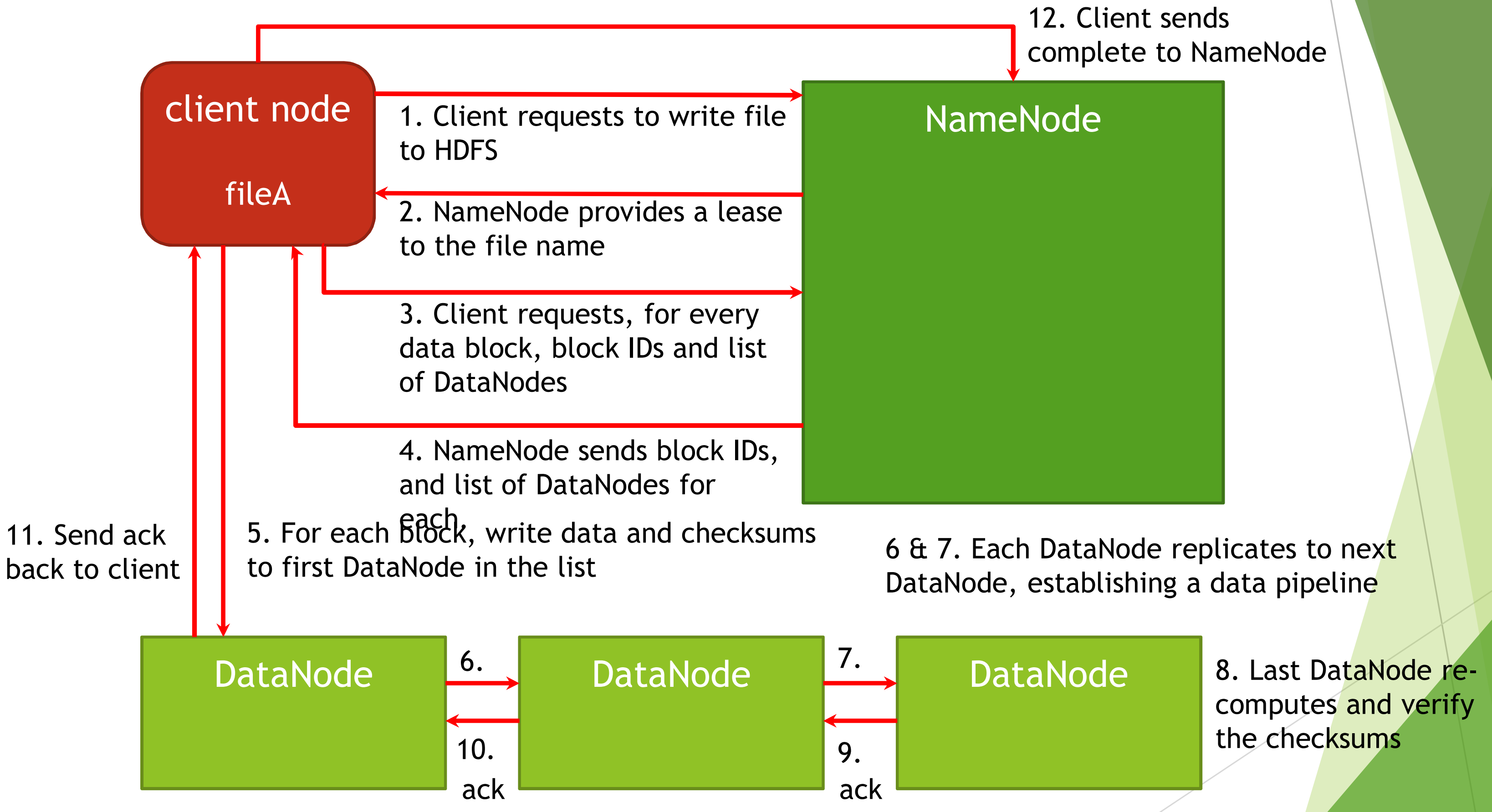
1. Client sends a request to the NameNode to add a file to HDFS

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

4. The DataNode replicates each block to two other DataNodes (as chosen by the NameNode)

Writing to HDFS Storage – Detailed view



12. Client sends complete to NameNode

client node

fileA

NameNode

1. Client requests to write file to HDFS

2. NameNode provides a lease to the file name

3. Client requests, for every data block, block IDs and list of DataNodes

4. NameNode sends block IDs, and list of DataNodes for each

11. Send ack back to client

5. For each block, write data and checksums to first DataNode in the list

6 & 7. Each DataNode replicates to next DataNode, establishing a data pipeline

DataNode

6.

10. ack

DataNode

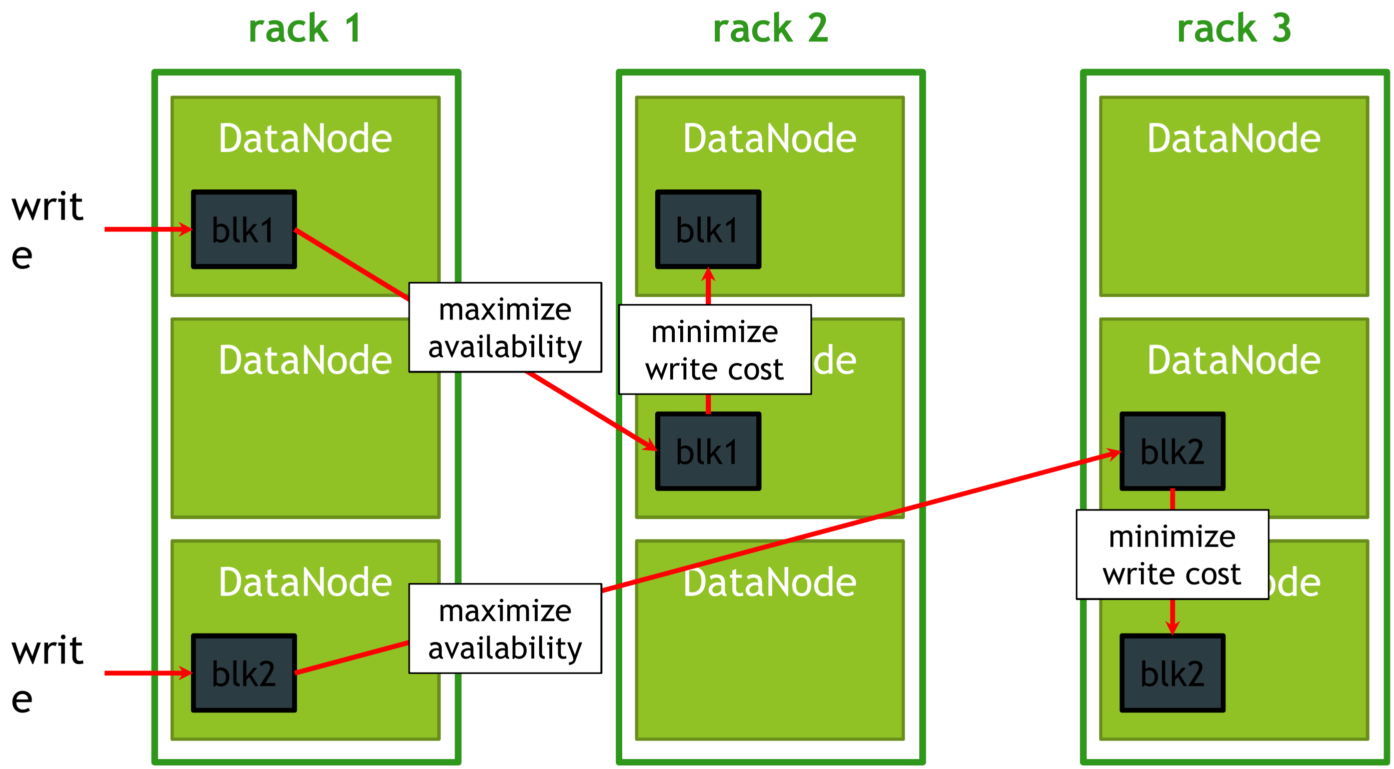
7.

9. ack

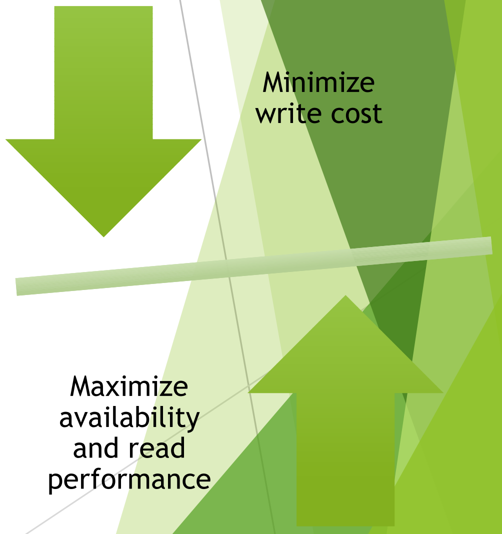
DataNode

8. Last DataNode re-computes and verify the checksums

Replication and Block Placement



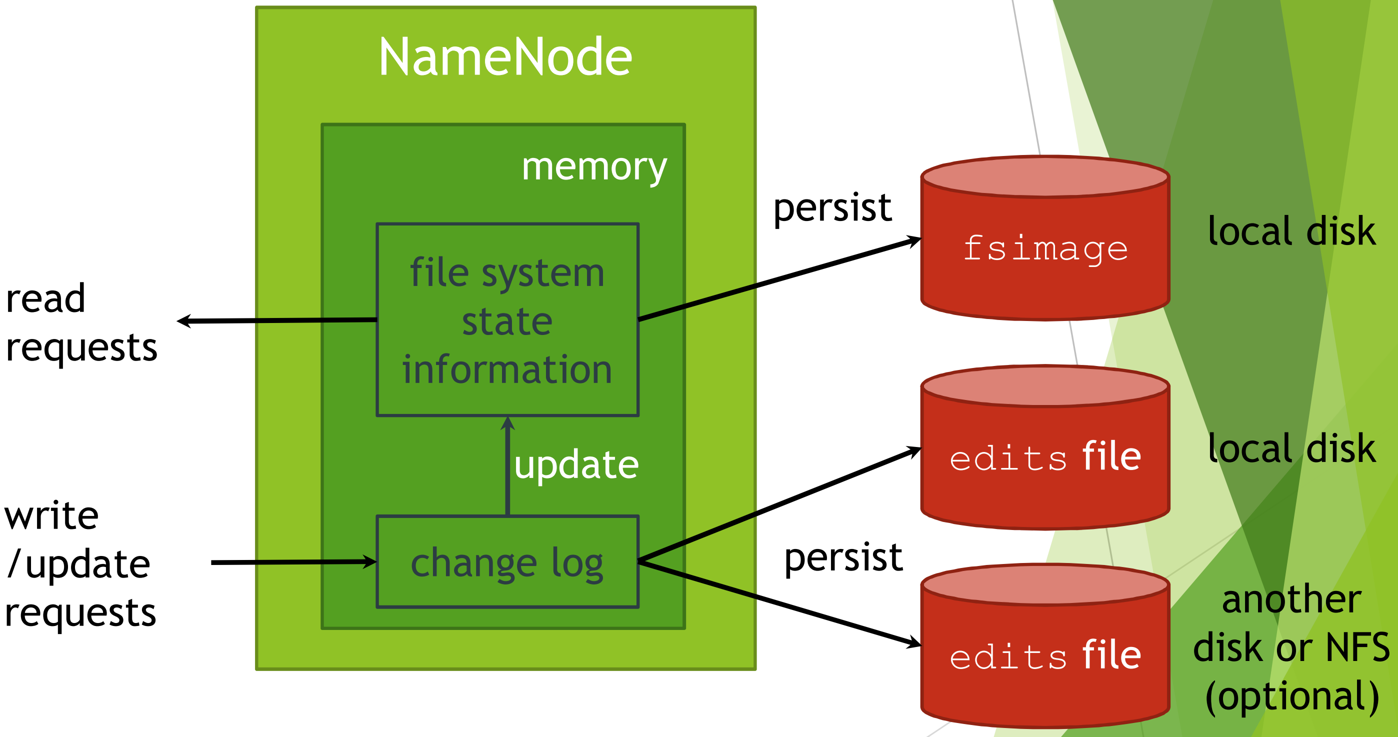
HDFS is designed to assume, and handle, disk and system failures.



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.



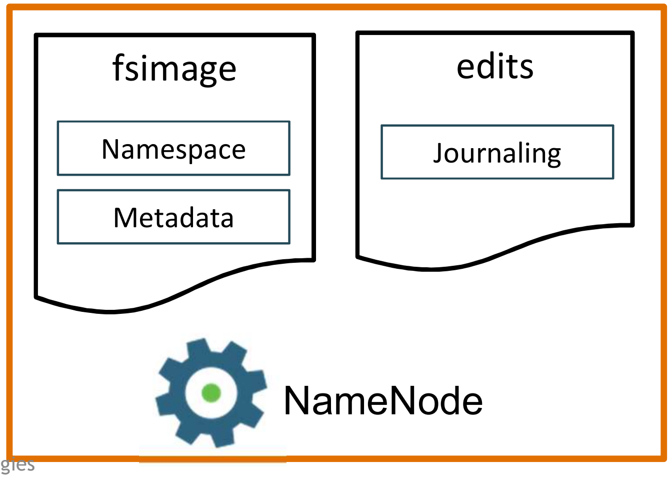
The NameNode Startup

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.

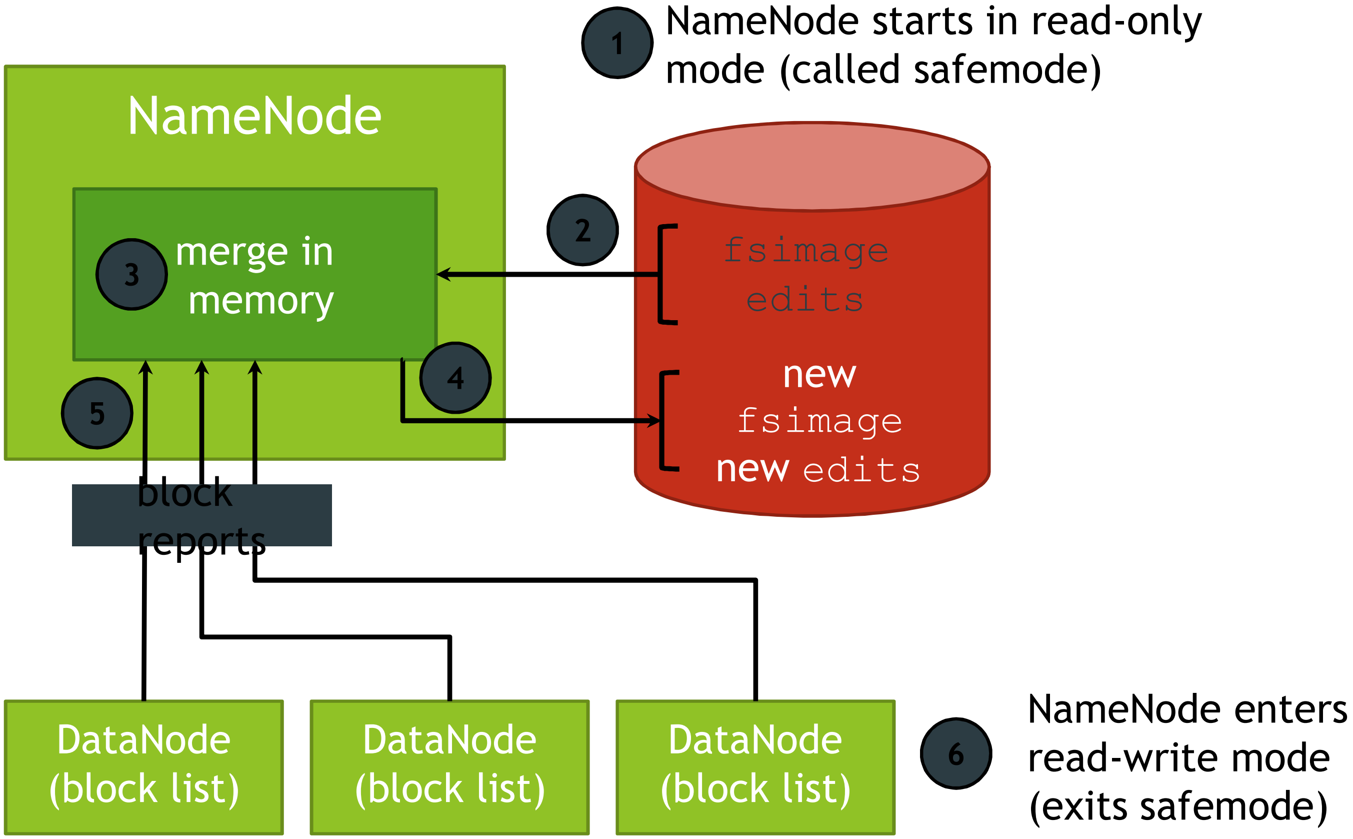
The NameNode will be in *safemode*, a read-only mode.



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

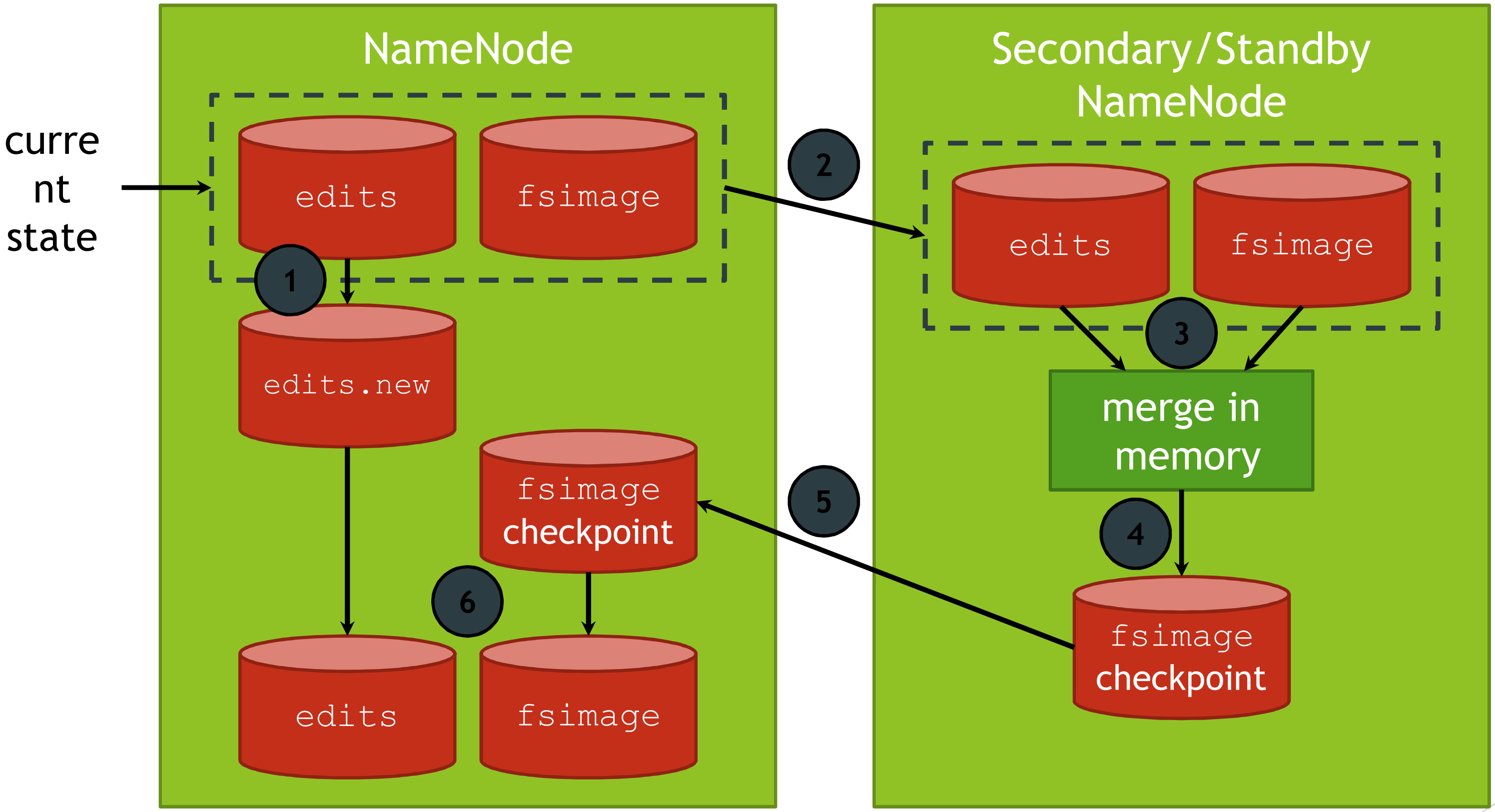


1. NameNode starts in safemode
1. Latest `fsimage` and `edits` read from disk
1. `edits` and `fsimage` files merged in memory
1. New `fsimage` and `edits` files created
1. Block lists retrieved from DataNodes and block map rebuilt
1. NameNode exits safemode

NameNode Checkpoint Operation

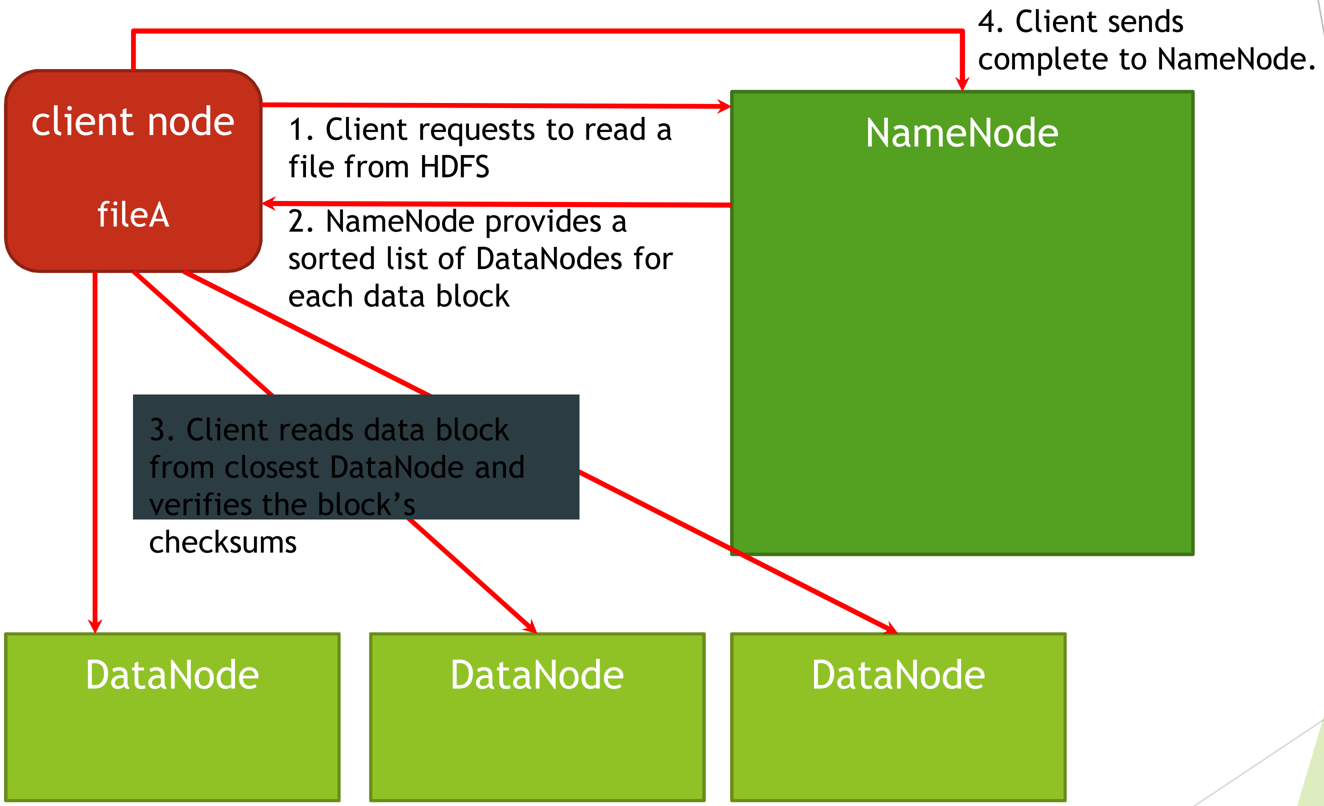
- NameNodes must periodically perform a checkpoint operation or the edits file would continue to grow without bounds.
- A checkpoint operation merges the changes recorded in the current edits file with the information in the current fsimage file, and then replaces the edits and fsimage files with a new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



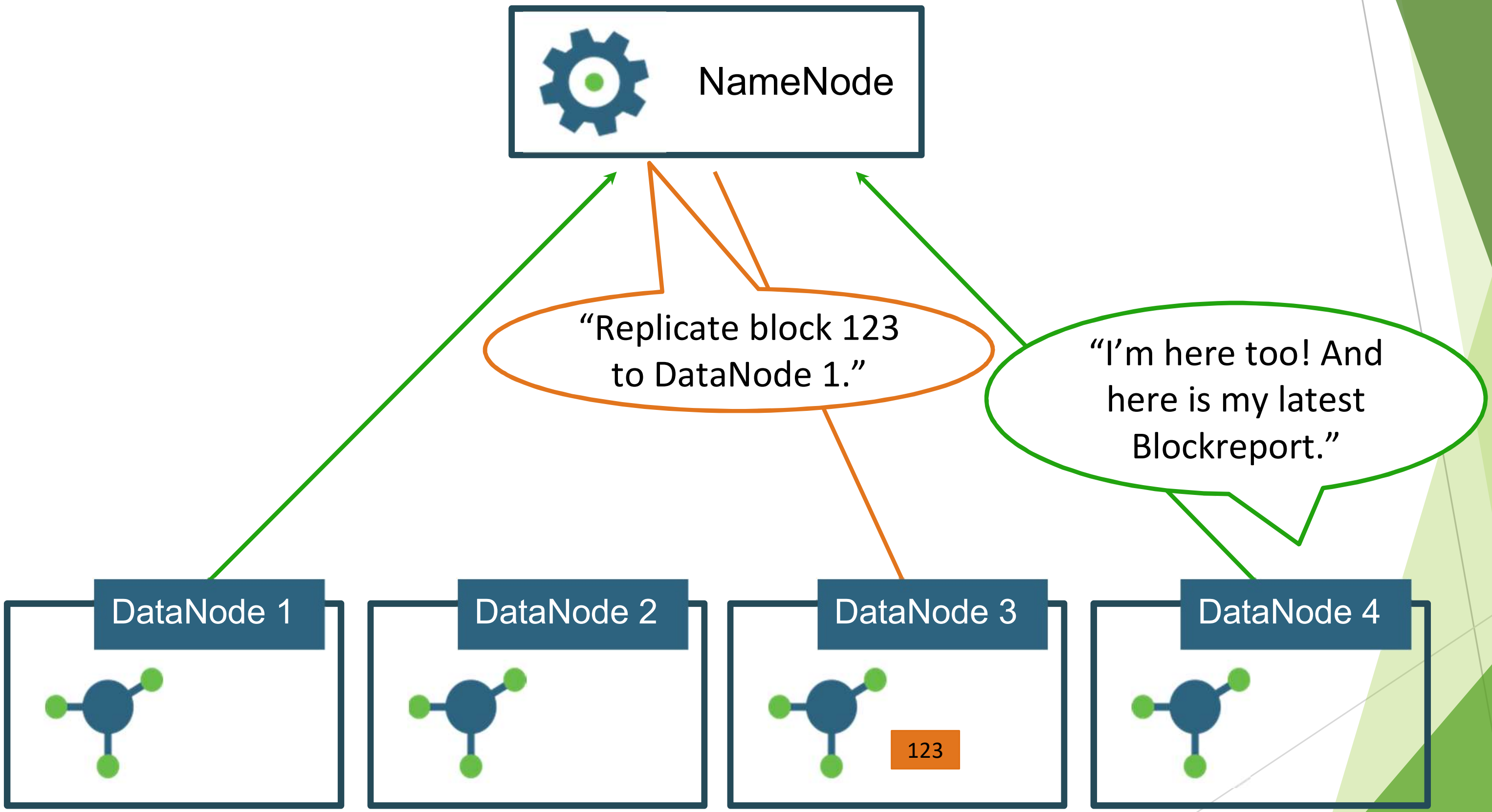
1. Client requests to read a file from HDFS

2. NameNode provides a sorted list of DataNodes for each data block

3. Client reads data block from closest DataNode and verifies the block's checksums

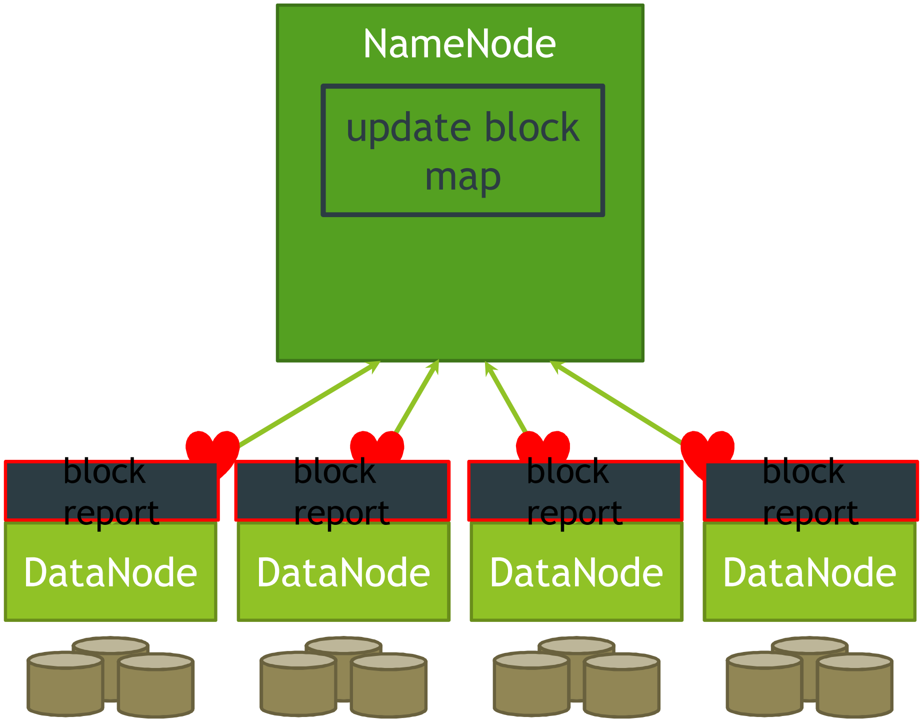
4. Client sends complete to NameNode.

The DataNode Block Reports



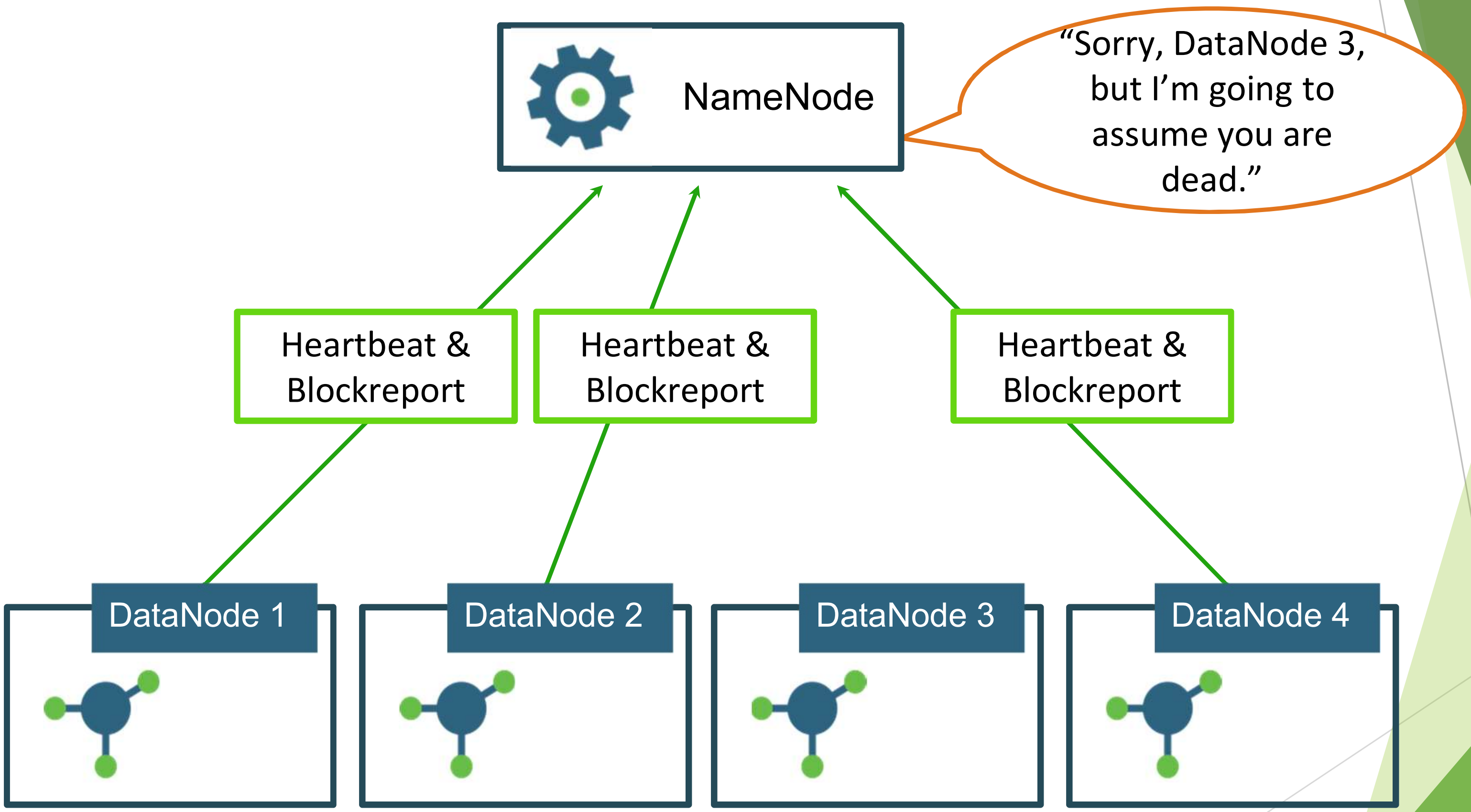
NameNode
"Replicate block 123 to DataNode 1."
"I'm here too! And here is my latest Blockreport."
DataNode 1
DataNode 2
DataNode 3
123
DataNode 4

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



NameNode

"Sorry, DataNode 3, but I'm going to assume you are dead."

Heartbeat & Blockreport

Heartbeat & Blockreport

Heartbeat & Blockreport

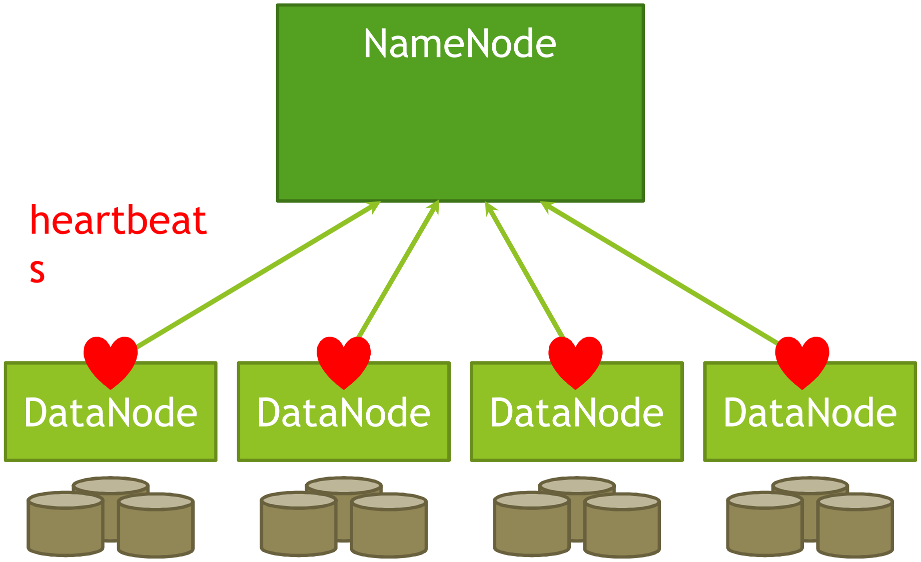
DataNode 1

DataNode 2

DataNode 3

DataNode 4

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.