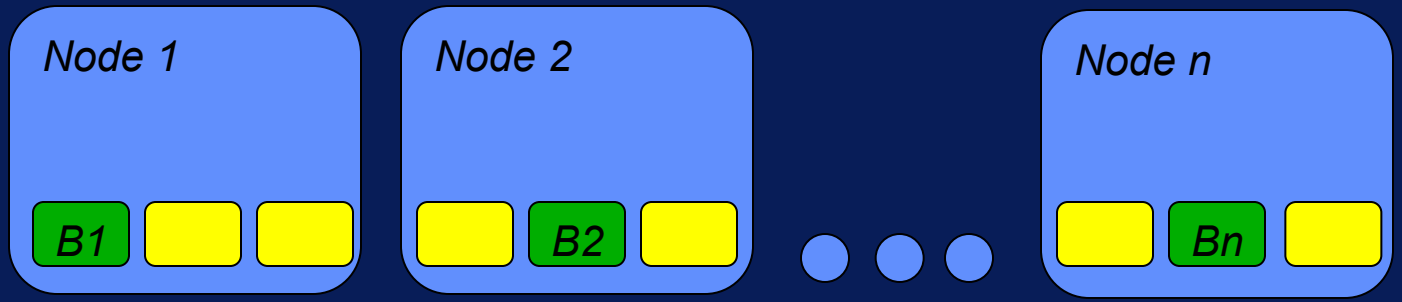
Overview of HDFS Architecture

- *Able to explain HDFS design goals*
- *Factors impacting design*
- *Design approach*
- *Explain basic HDFS architecture*

HDFS Design Concept

- *Scalable distributed filesystem*
- *Distribute data on local disks on several nodes*
- *Low cost commodity hardware*



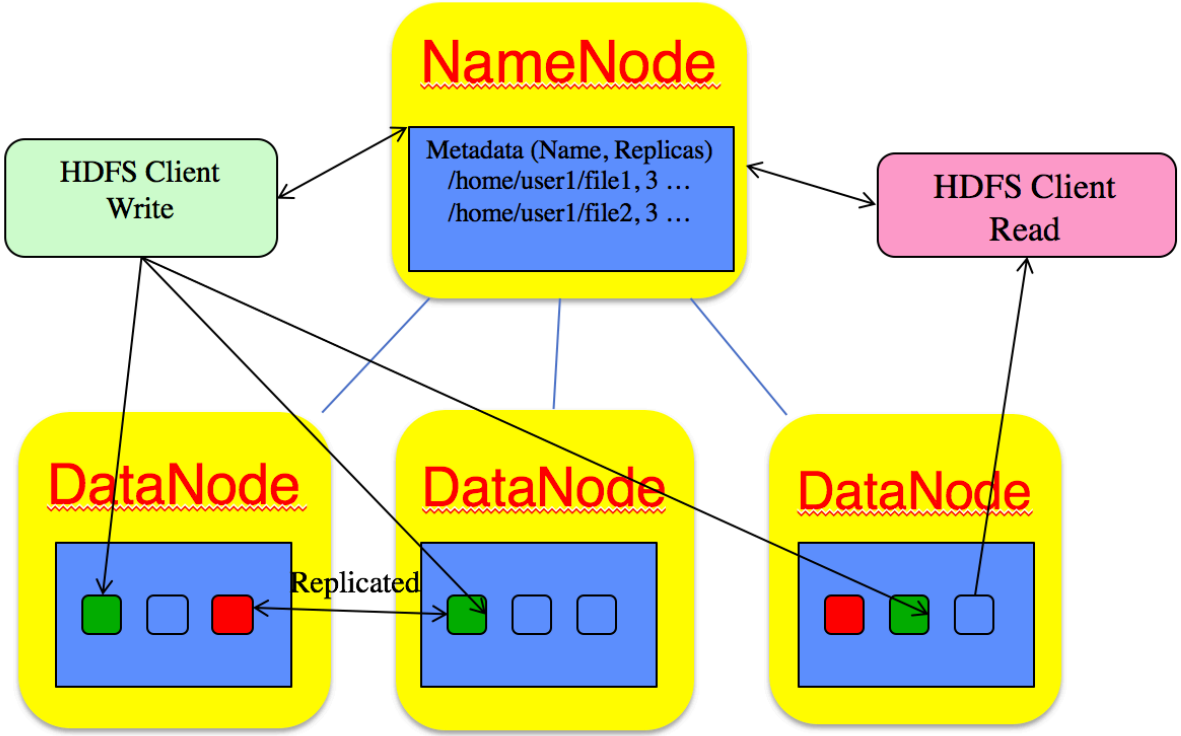
HDFS Design Factors

- ***Hundreds/Thousands of nodes*** **=>**
  - Need to handle node/disk failures
- ***Portability across heterogeneous hardware/software***
- ***Handle large data sets***
- ***High throughput***

Approach to meet HDFS design goals

- ***Simplified coherency model*** – write once read many.
- ***Data Replication*** – helps handle hardware failures
- ***Move computation close to data***
- ***Relax POSIX requirements*** – increase throughput

HDFS Architecture



Summary of HDFS Architecture

- ***Single NameNode -*** a master server that manages the file system namespace and regulates access to files by clients.
- ***Multiple DataNodes –*** typically one per node in the cluster. Functions:
  - Manage storage
  - Serving read/write requests from clients
  - Block creation, deletion, replication

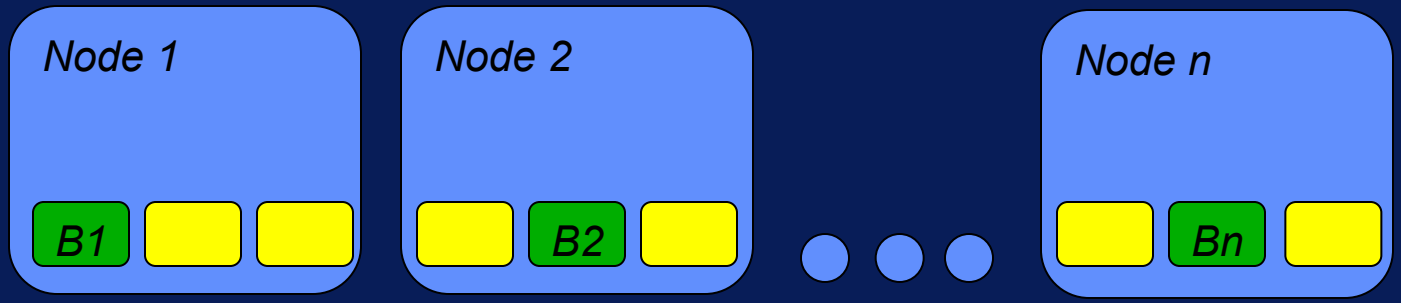
    based on instructions from NameNode

Performance Envelope of HDFS

- *Able to determine number of blocks for a given file size*
- *Key HDFS and system components impacted by block size*
- *Impact of small files on HDFS and system*

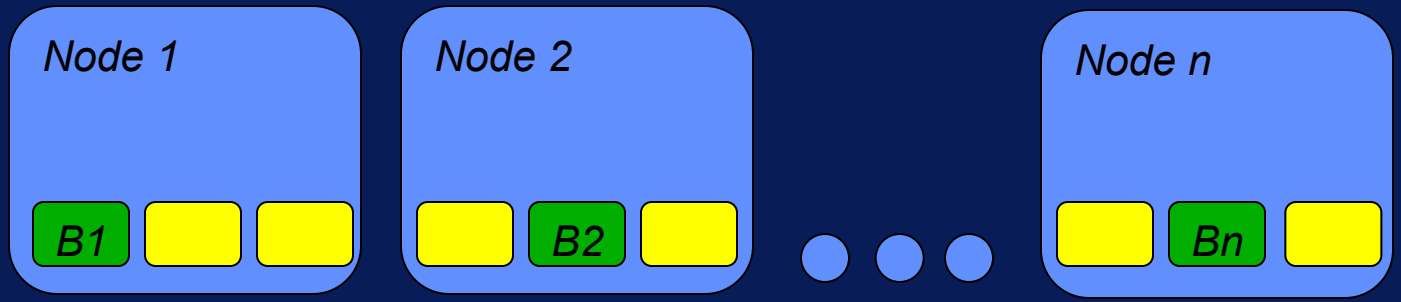
Recall: HDFS Architecture

- ***Distribute data on local disks on several nodes.***



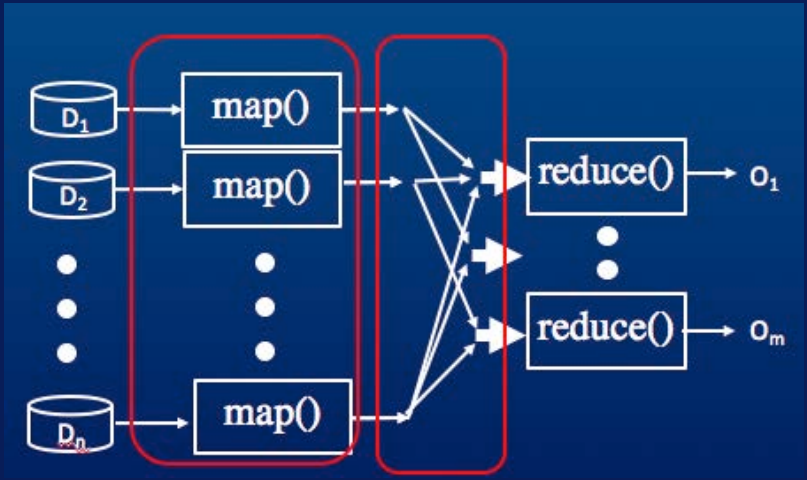
HDFS block size

- *Default block size is 64MB*
- *Good for large files!*
- *So a 10GB file will be broken into: 10x1024/64 = 160 blocks.*



Importance of #blocks in a file

- *NameNode memory usage:* Every block represented as object (default replication this will be further increased 3X)
- *Number of map tasks*: data typically processed block at a time



Large #small files: Impact on NameNode

- *Memory usage* ~ 150 bytes per object
  1 billion objects => 300GB memory!

- *Network load* – Number of checks with datanodes proportional to number of blocks.

Large #small files: Performance Impact

- ***Map tasks*** **– depends on #blocks**
  10GB of data, 32k file size => 327680 map tasks

  ⇒lots of queued tasks

  ⇒large overhead of spin up/tear down for each task

  ⇒Inefficient disk I/O with small sizes

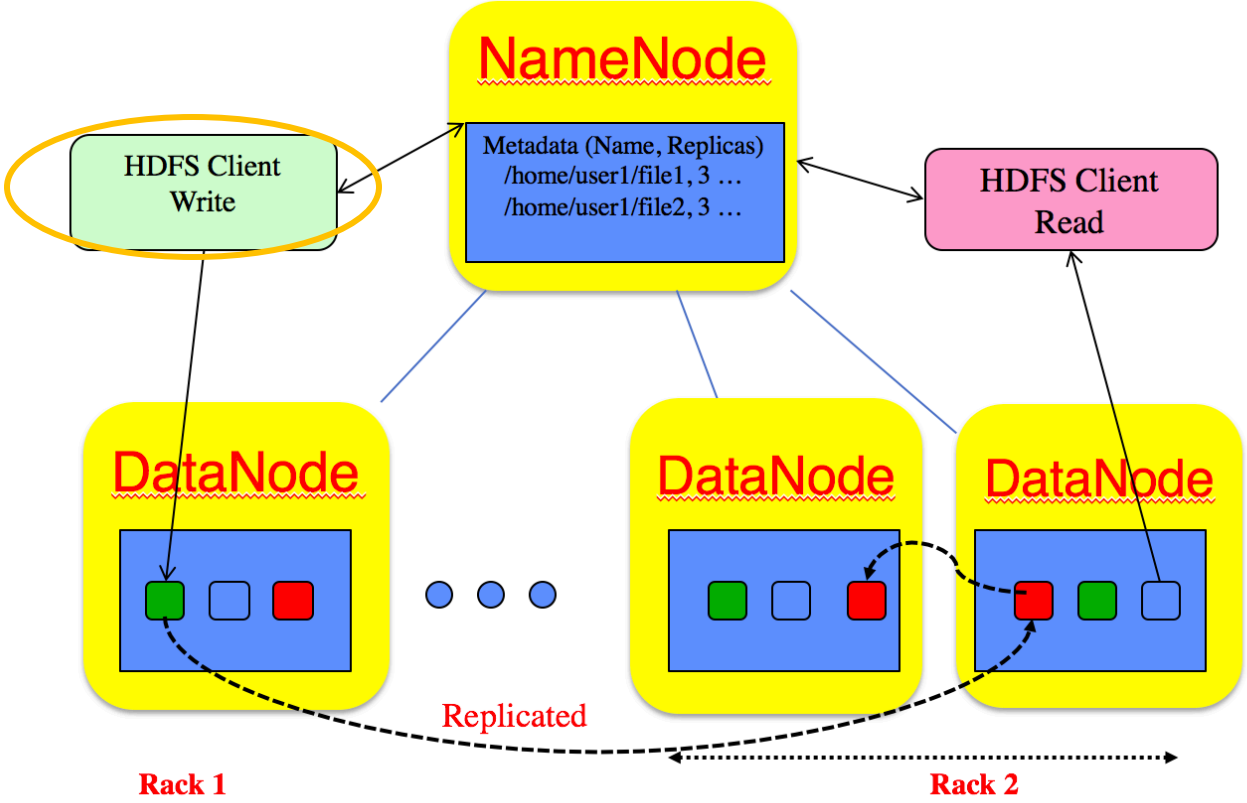
HDFS optimized for large files

- ***Key takeaway – lots of small files is bad!***
- **Solutions:**
  - Merge/Concatenate files
  - Sequence files
  - HBase, HIVE configuration
  - CombineFileInputFormat

Write/Read processes on HDFS

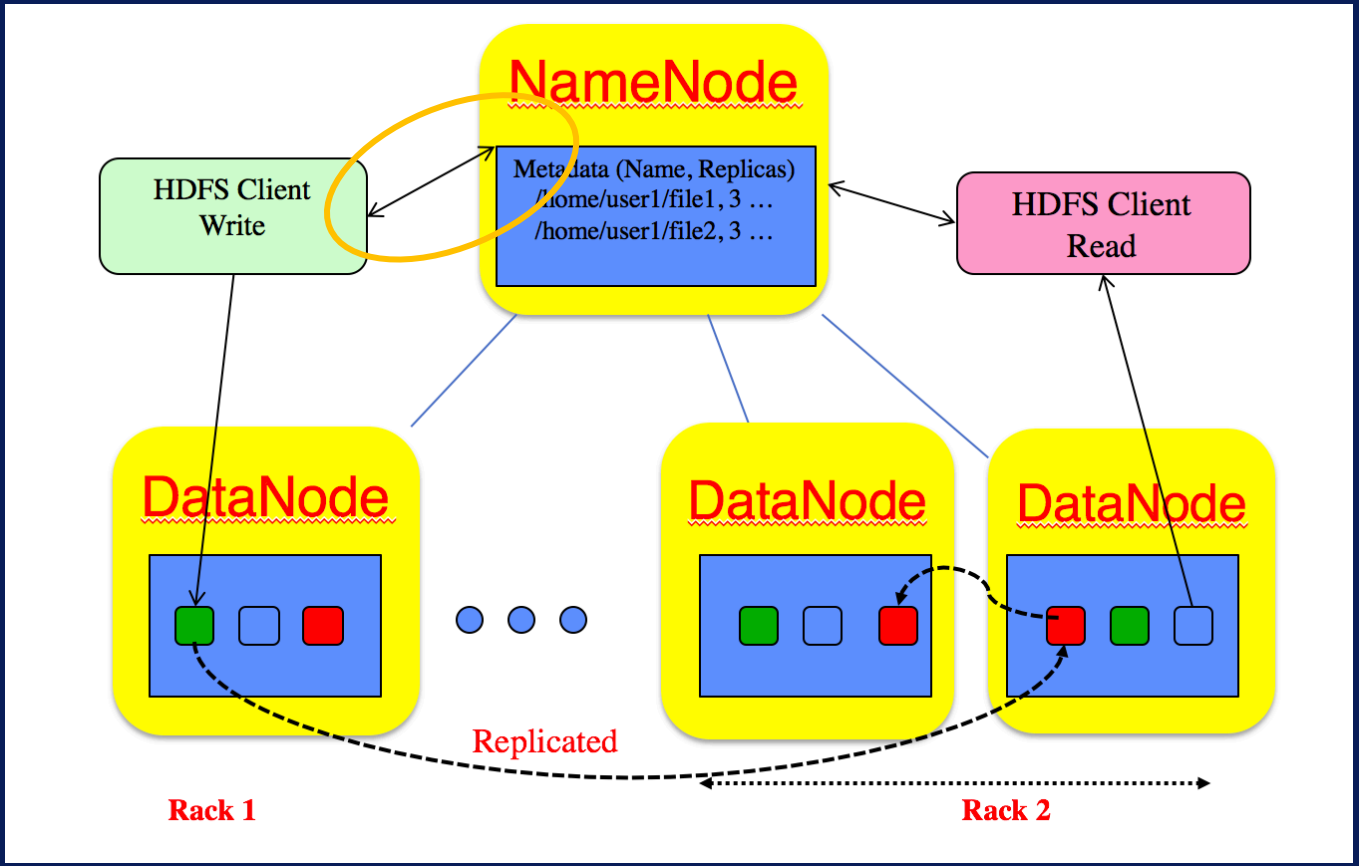
- *Able to explain write process in HDFS*
- *Detail the replication pipeline process*
- *Explain read process in HDFS*

Write Process in HDFS



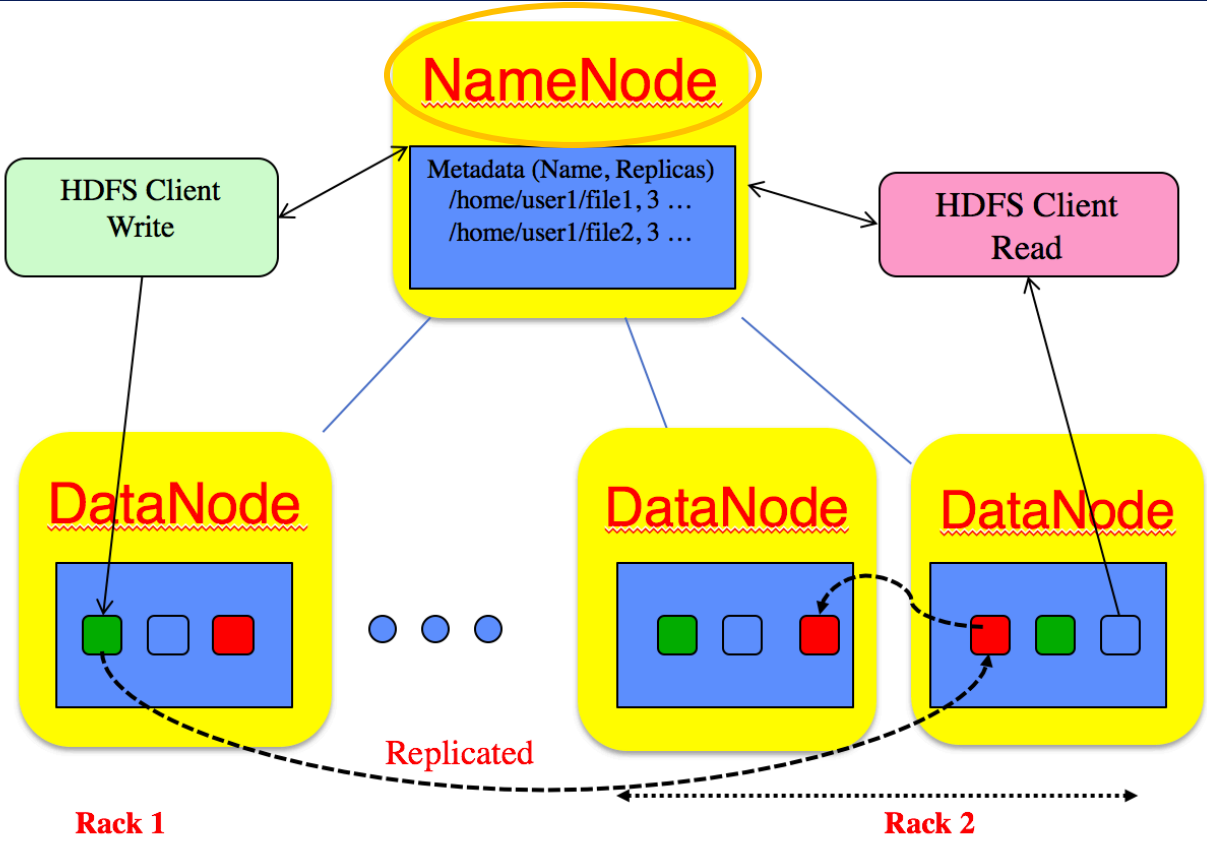
Client request to create file

Write Process in HDFS



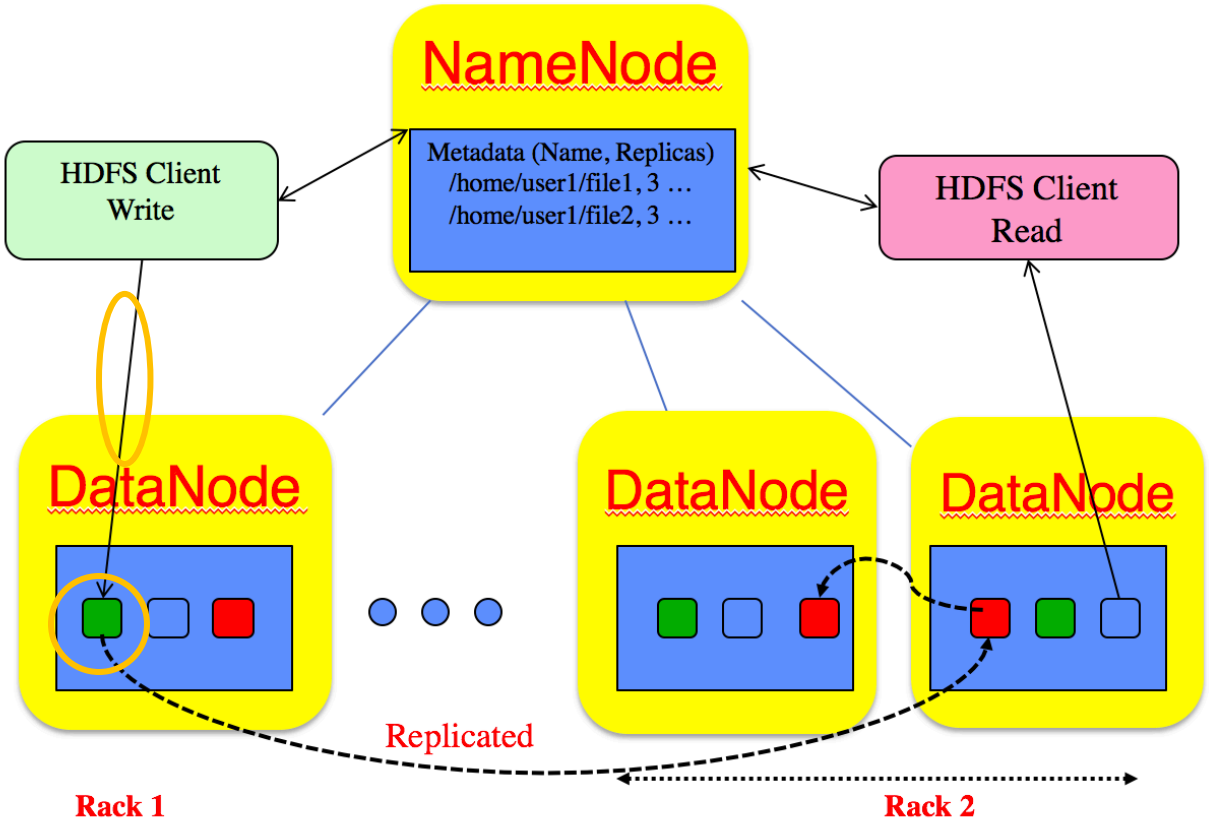
NameNode contacted once a block of data is accumulated

Write Process in HDFS



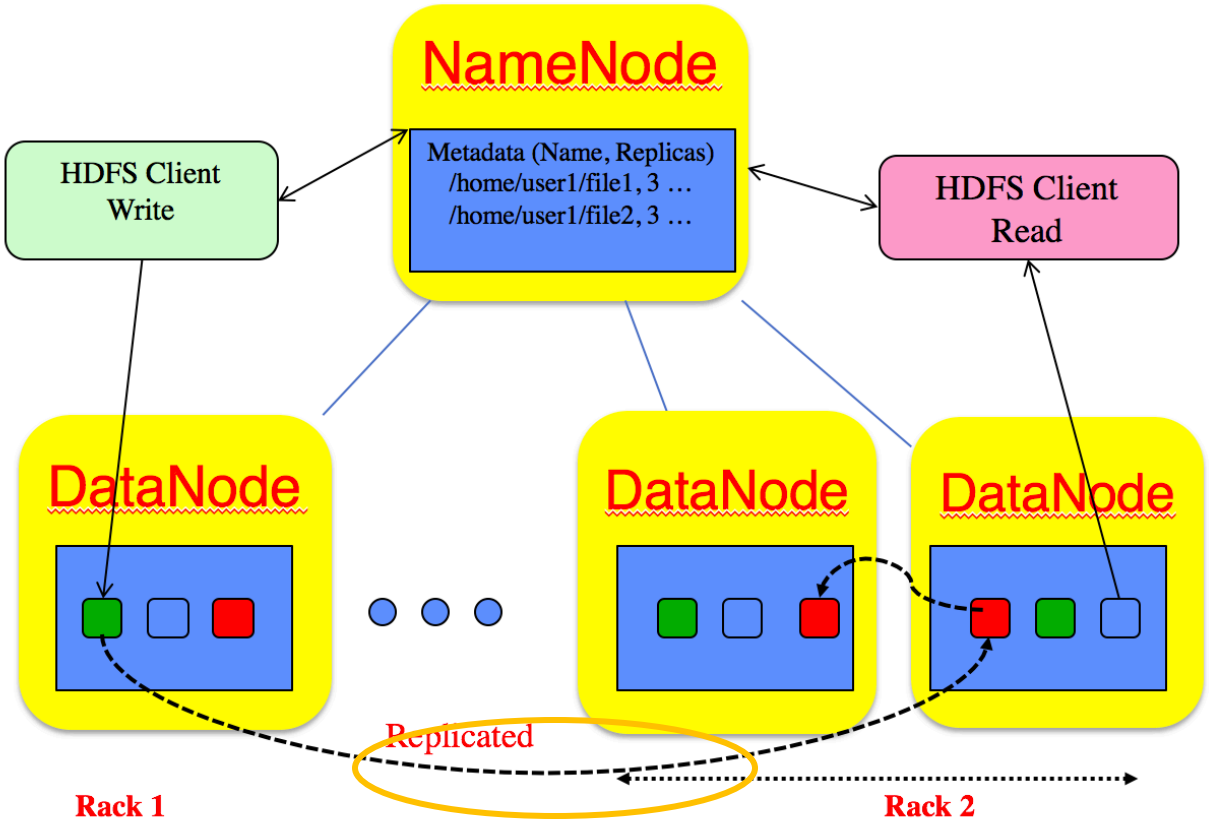
- *NameNode responds with list of DataNodes*
- *Rack aware*

Write Process in HDFS



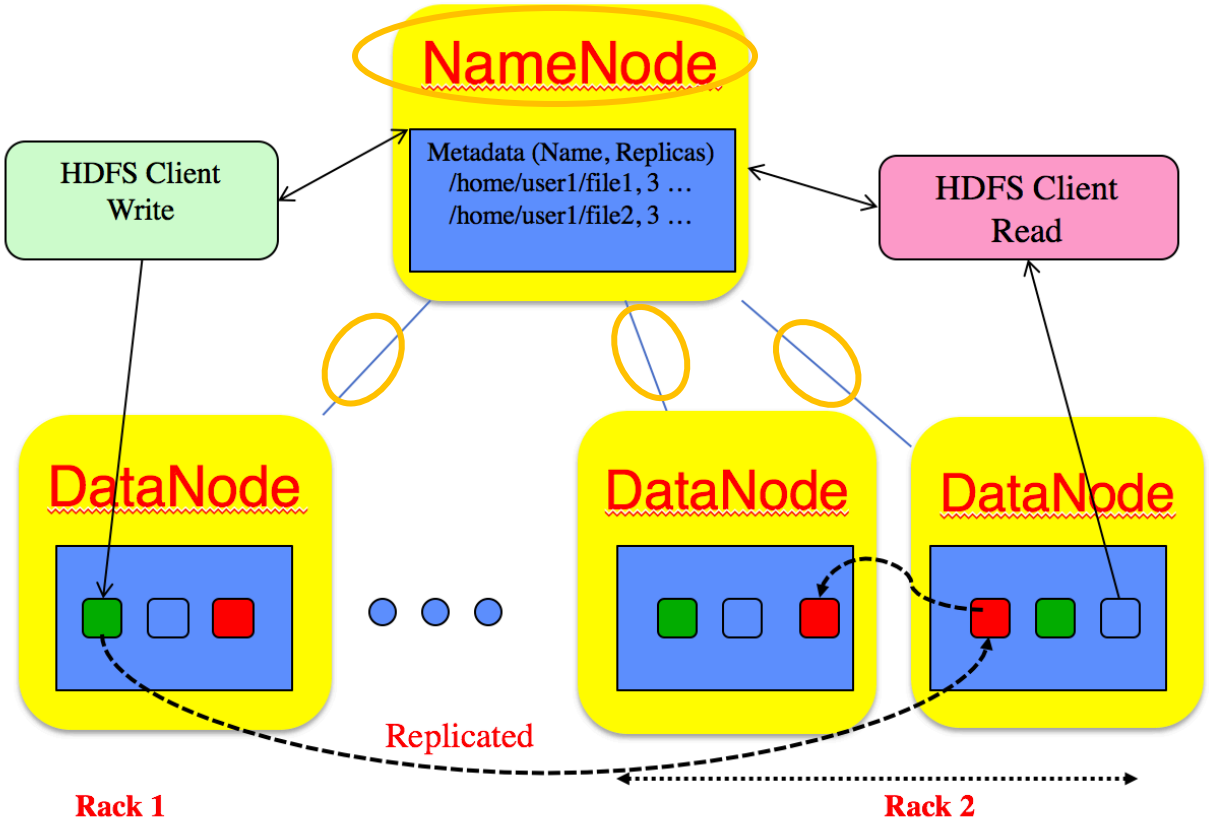
- *First DataNode receives data, writes to local and forwards to second DataNode*

Write Process in HDFS



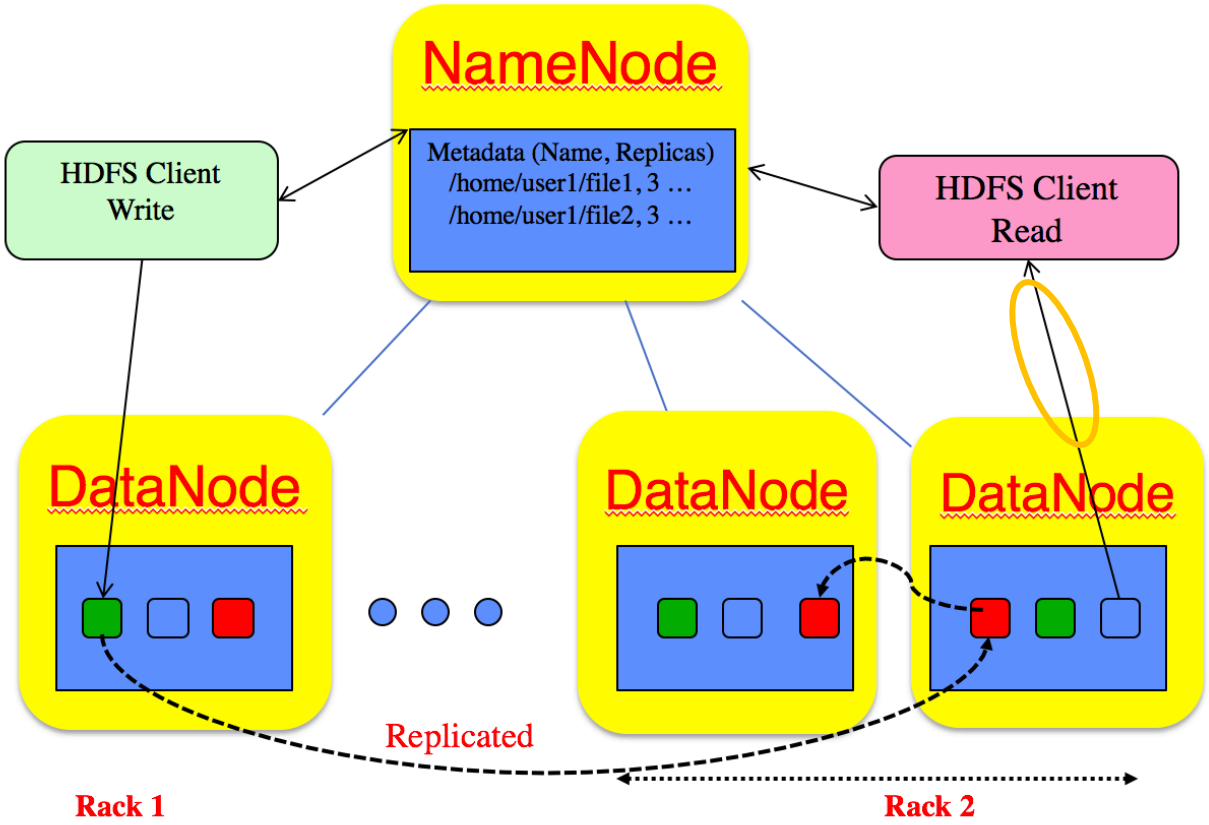
- *First DataNode receives data, writes to local and forwards to second DataNode*

Write Process in HDFS



- *NameNode commits file creation into persistent store.*
- *Receives heartbeat and block reports*

Read Process in HDFS



- *Client gets DataNode list from NameNode*
- *Read from replica closest to reader.*