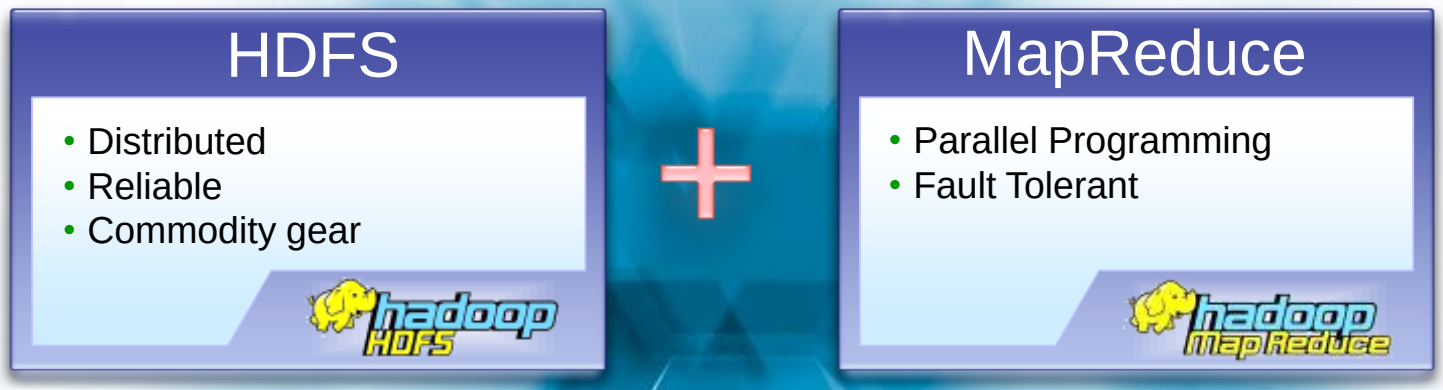
Hadoop Distributed File System (HDFS)

Agenda

- **Overview**
- **Architecture**
  - NameNode
  - DataNode
- **Blocks and Replication**
- **File System Shell**
- **Web Console**

Two Key Aspects of Hadoop

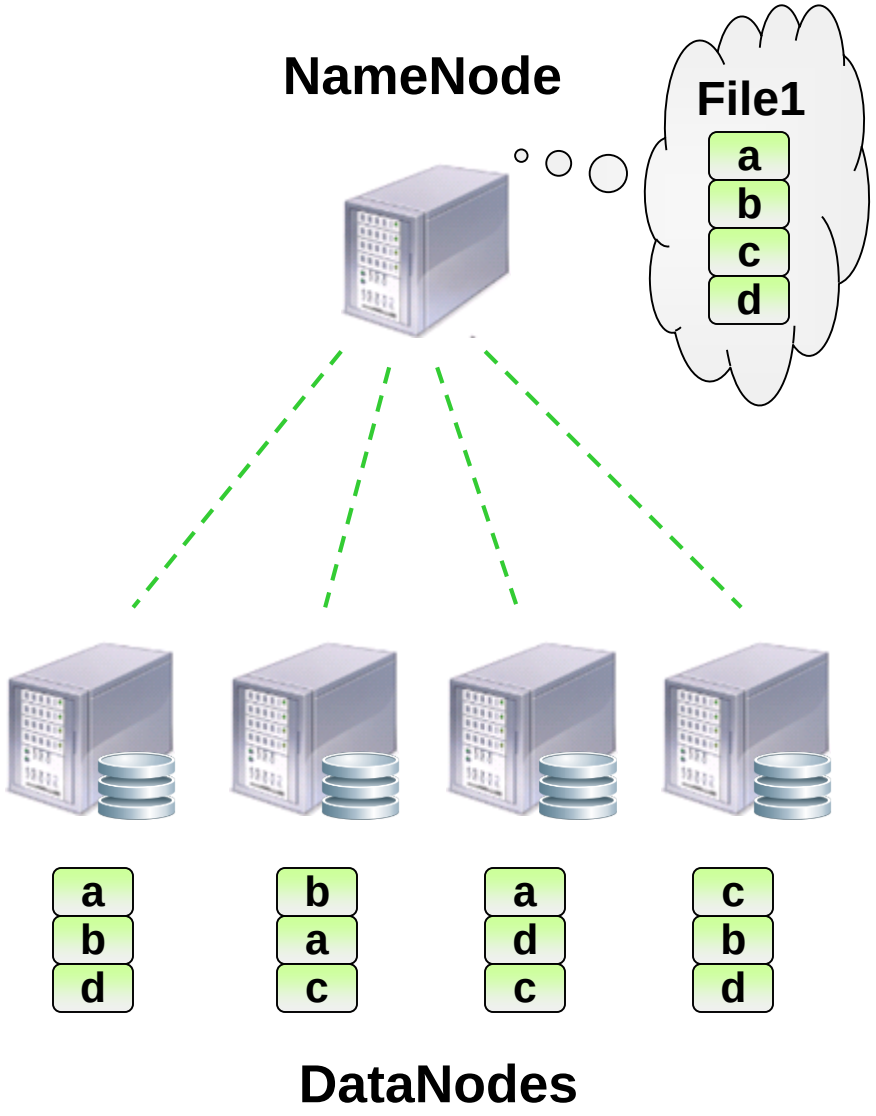


Hadoop Distributed File System (HDFS)

- Distributed, scalable, fault tolerant, high throughput
- Data access through MapReduce
- Files split into blocks
- 3 replicas for each piece of data by default
- Can create, delete, copy, but NOT update
- Designed for streaming reads, not random access
- Data locality: processing data on or near the physical storage to decrease transmission of data

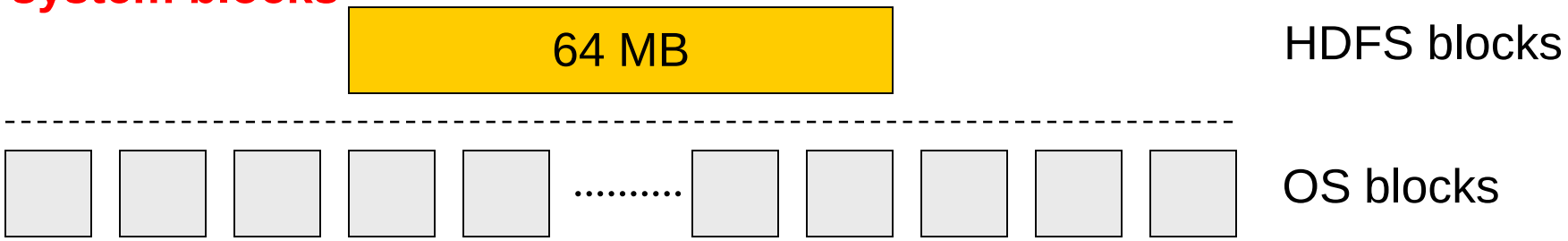
HDFS – Architecture

- **Master / Slave architecture**
- **Master: NameNode**
  - manages the file system namespace and metadata
    - FsImage
    - EditLog
  - regulates client access to files
- **Slave: DataNode**
  - many per cluster
  - manages storage attached to the nodes
  - periodically reports status to NameNode



HDFS – Blocks

- **HDFS is designed to support very large files**
- **Each file is split into blocks**
  - Hadoop default: 64MB
  - BigInsights default: 128MB
- **Blocks reside on different physical DataNode**
- **Behind the scenes, 1 HDFS block is supported by multiple operating system blocks**



- **If a file or a chunk of the file is smaller than the block size, only needed space is used. E.g.: a 210MB file is split as**

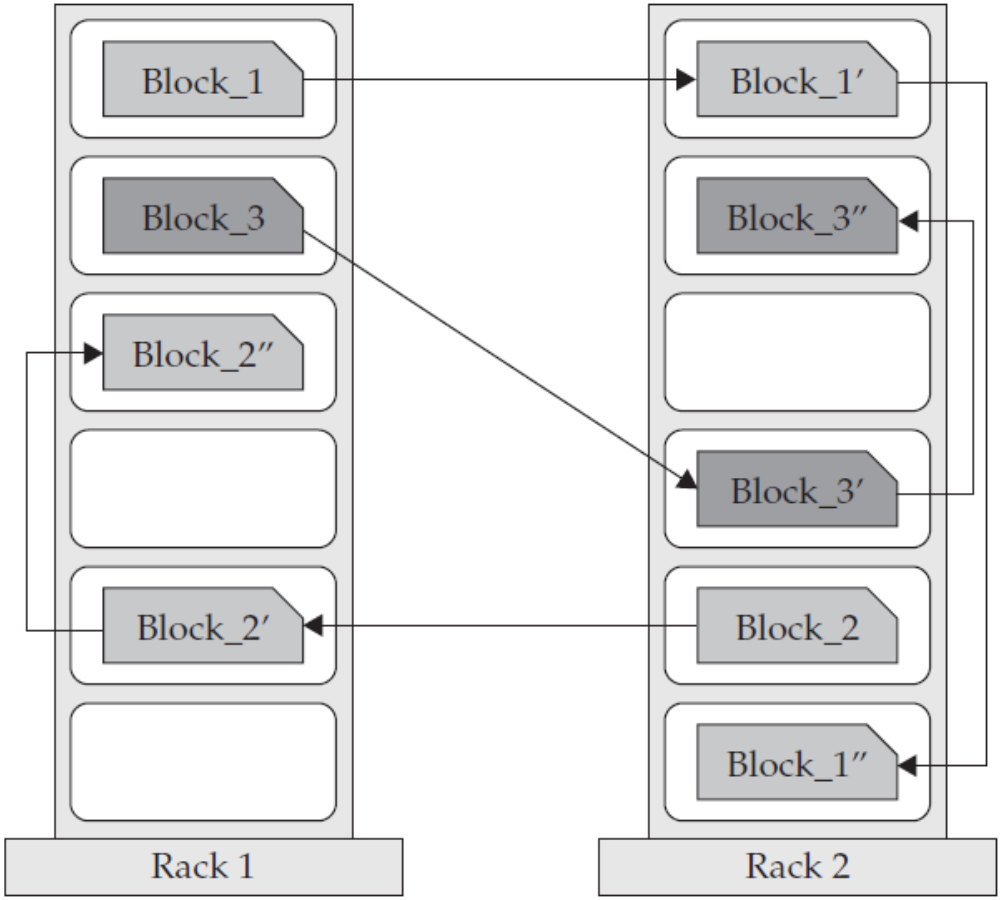
HDFS – Replication

- **Blocks of data are replicated to multiple nodes**
  - Behavior is controlled by replication factor, configurable per file
  - Default is 3 replicas

Common case:

- **one replica on one node in the local rack**
- **another replica on a different node in the local rack**
- **and the last on a different node in a different rack**

This cuts inter-rack network bandwidth, which improves write performance



Setting Rack Topology (Rack Awareness)

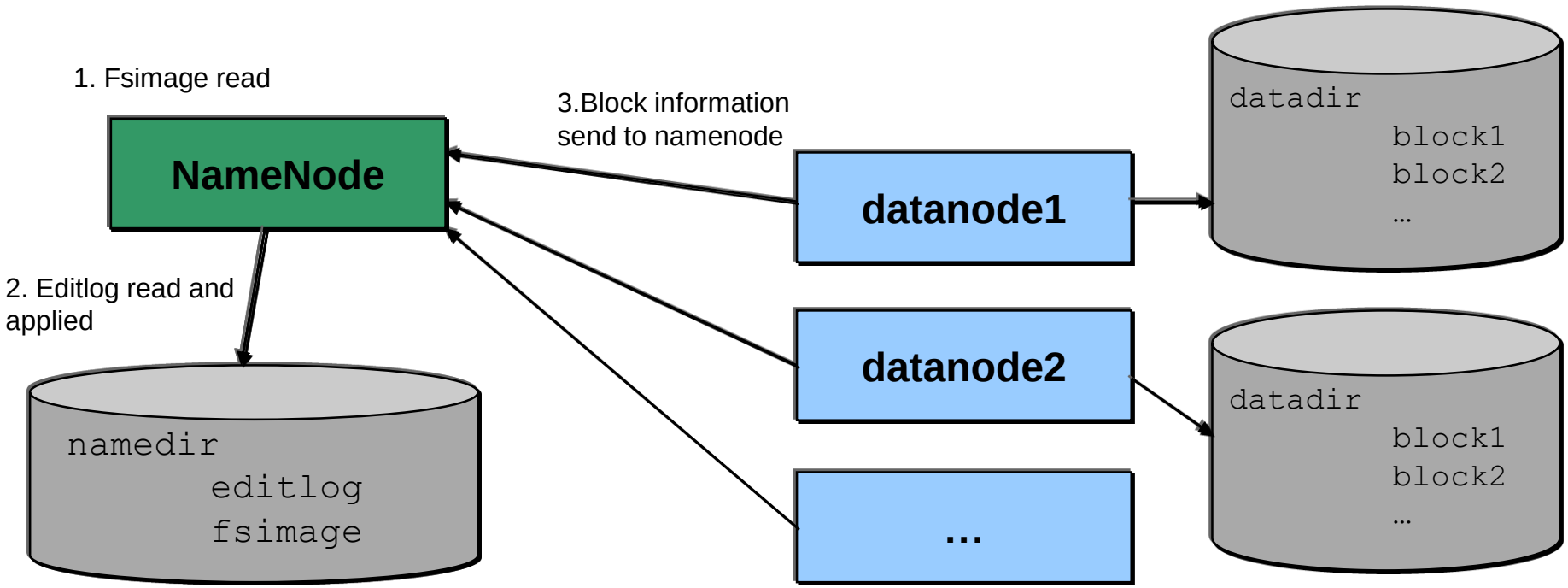
- **Can be defined by script which specifies which node is on which rack.**
- **Script is referenced in** topology.script.property.file **in** core-site.xml**.**
  - Example of property:

```
<property>
    <name>topology.script.file.name</name>
    <value>/opt/ibm/biginsights/hadoop-conf/rack-aware.sh</value>
</property>
```

- **The *network topology script* (**topology.script.file.name **in the above example) receives as arguments one or more IP addresses of nodes in the cluster. It returns on stdout a list of rack names, one for each input. The input and output order must be consistent.**

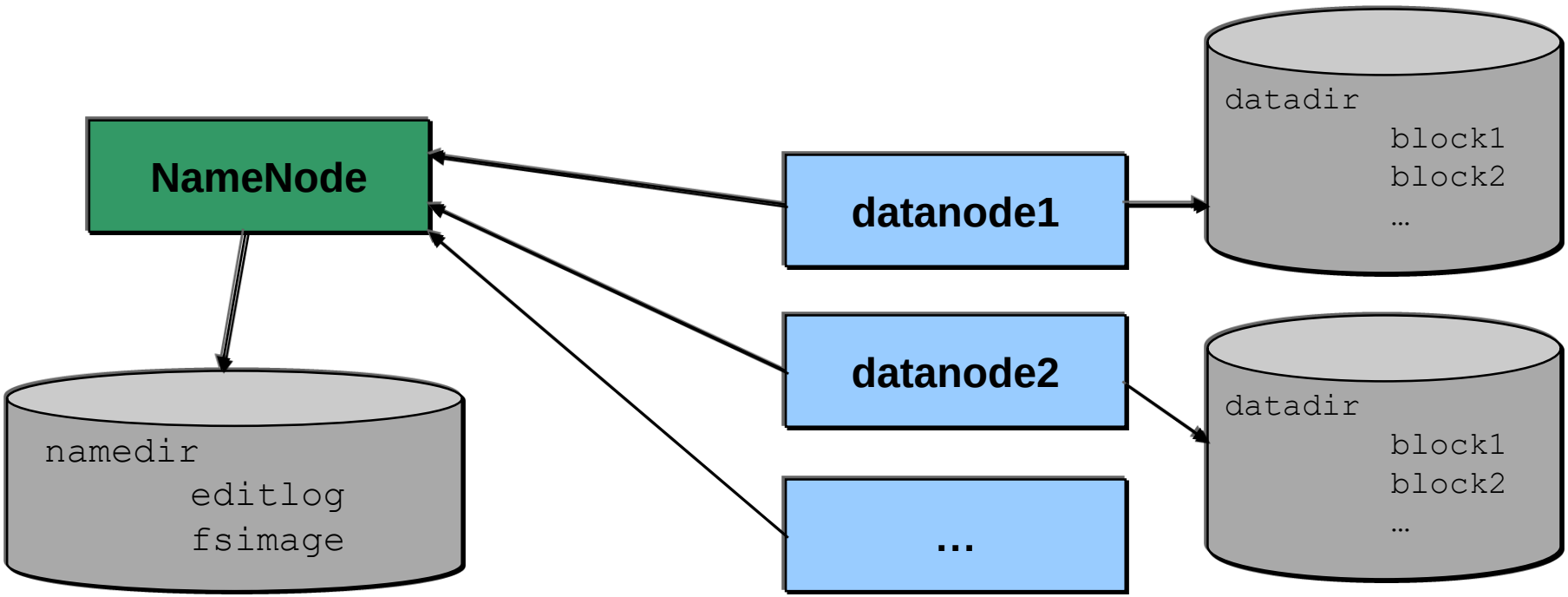
Namenode Startup

1. **NameNode reads fsimage in memory**
2. **NameNode applies editlog changes**
3. **NameNode waits for block data from data nodes**
   - Namenode doesn't store block information
   - Namenode exits safemode when 99.9% of blocks have at least one copy accounted for



Adding file

1. **File is added to NameNode memory and persisted in editlog**
2. **Data is written in blocks to datanodes**
   - Datanode starts chained copy to two other datanodes
   - If at least one write for each block succeeds, write is successful



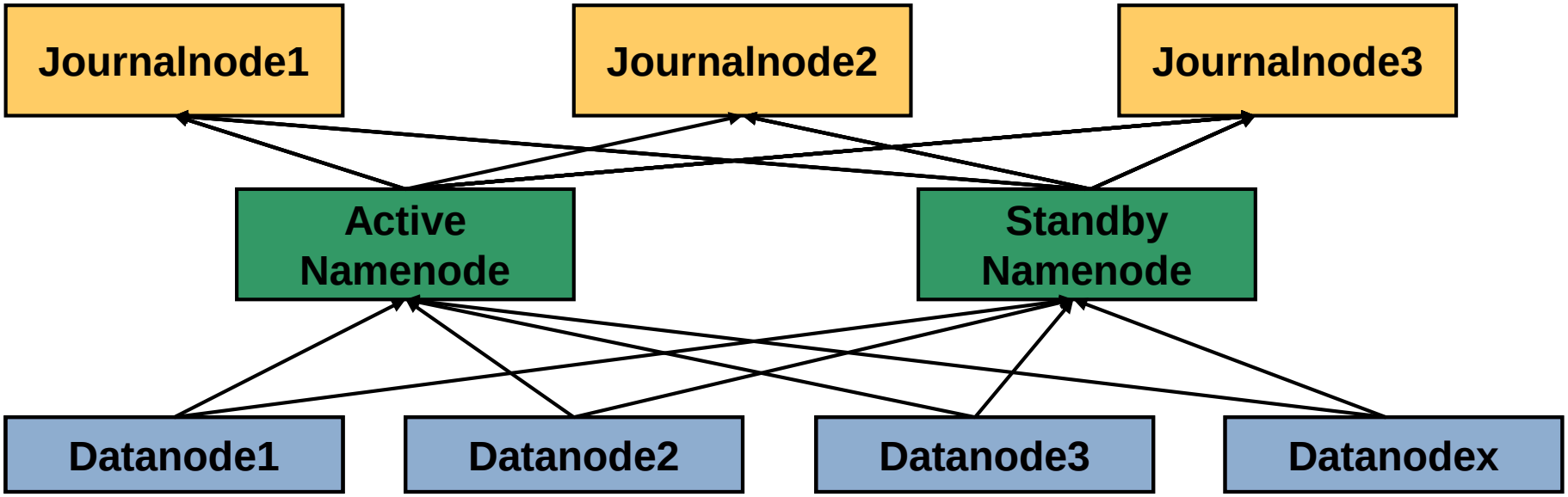
Managing Cluster

- **Adding Data Node**
  - Start new datanode (pointing to namenode)
  - If required run balancer (`hadoop balancer`) to rebalance blocks
- **Remove Node**
  - Simply remove datanode
  - Better: Add node to exclude file and wait till all blocks have been moved
  - Can be checked in server admin console server:50070
- **Checking filesystem health**
  - Use `hadoop fsck`

HDFS-2 Namenode HA

- **HDFS-2 adds Namenode High Availability**
- **Standby Namenode needs filesystem transactions and block locations for fast failover**
- **Every filesystem modification is logged to at least 3 quorum journal nodes by active Namenode**
  - Standby Node applies changes from journal nodes as they occur
  - Majority of journal nodes define reality
  - Split Brain is avoided by Journalnodes (They will only allow one Namenode to write to them)
- **Datanodes send block locations and heartbeats to both Namenodes**
- **Memory state of Standby Namenode is very close to Active Namenode**

➔ Much faster failover than cold start



Federated Namenode (HDFS2)

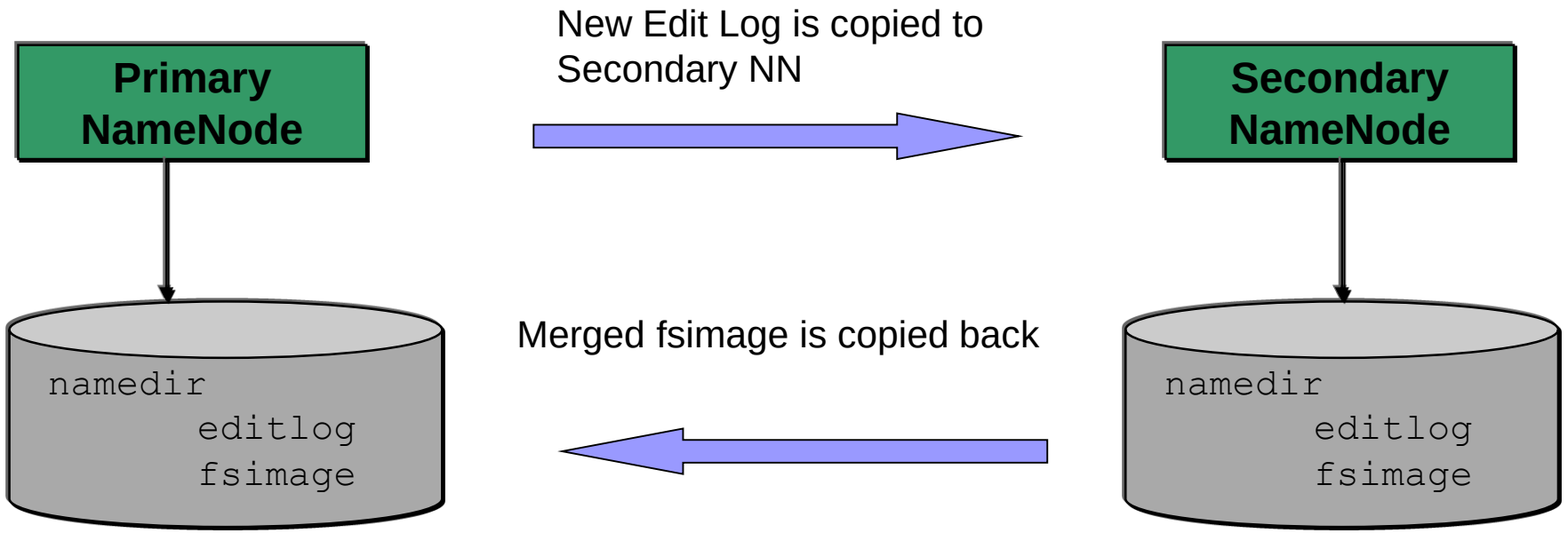
- **New in Hadoop2 Namenodes can be federated**
  - Historically Namenodes would become a bottleneck on huge clusters
  - One million blocks or ~100TB of data require roughly one GB of RAM in Namenode

- **Blockpools**
  - Administrator can create separate blockpools/namespaces with different namenodes
  - Datanodes register on all Namenodes
  - Datanodes store data of all blockpools (otherwise you could setup separate clusters)
  - New **ClusterID** identifies all namenodes in a cluster.
  - A Namespace and its block pool together are called Namespace Volume
  - You define which blockpool to use by connecting to a specific Namenode
  - Each Namenode still has its own separate backup/secondary/checkpoint node

- **Benefits**
  - One Namenode failure will not impact other Blockpools
  - Better scalability for large numbers of file operations

Secondary NameNode

- **During operation primary Namenode cannot merge fsImage and editlog**
- **This is done on the secondary namenode**
  - Every couple minutes, secondary namenode copies new edit log from primary NN
  - Merges editLog into fsimage
  - Copies the new merged fsImage back to primary namenode
- **Not HA but faster startup time**
  - Secondary NN does not have complete image. In-flight transactions would be lost
  - Primary Namenode needs to merge less during startup
- **Was temporarily deprecated because of Namenode HA but has some advantages**
  - (no need for Quorum nodes, less network traffic, less moving parts)



Possible FileSystem Setup

- **GPFS**
  - No single point of failure
  - Posix compliance
  - Advanced features like cold storage, backup and restore

- **Hadoop 2 with HA**
  - No single point of failure
  - Wide community support

- **Hadoop 2 without HA (or Hadoop 1.x in older versions)**
  - Copy namedir to NFS (RAID)
  - Have virtual IP for backup namenode
  - Still some failover time to read blocks, no instant failover but less overhead

fs – file system shell

- **File System Shell (fs)**
  - Invoked as follows:

```
hadoop fs <args>
```

- **Example:**
  - Listing the current directory in hdfs

```
hadoop fs –ls .
```

fs – file system shell

- **FS shell commands take URIs as argument**
  - URI format: scheme://authority/path
- **Scheme:**
  - For the local filesystem, the scheme is *file*
  - For HDFS, the scheme is *hdfs*
- **Authority is the hostname and port of the NameNode**

```
hadoop fs –copyFromLocal
        file:///myfile.txt
        hdfs://localhost:9000/user/keith/myfile.txt
```

- **Scheme and authority are optional**
  - Defaults are taken from configuration file core-site.xml

fs – file system shell

- **Many POSIX-like commands**
  - cat, chgrp, chmod, chown, cp, du, ls, mkdir, mv, rm, stat, tail
- **Some HDFS-specific commands**
  - copyFromLocal, put, copyToLocal, get, getmerge, setrep

HDFS – FS shell commands

- **copyFromLocal / put**
  - Copy files from the local file system into fs

```
hadoop fs -copyFromLocal <localsrc> .. <dst>
```

Or

```
hadoop fs -put <localsrc> .. <dst>
```

HDFS – FS shell commands

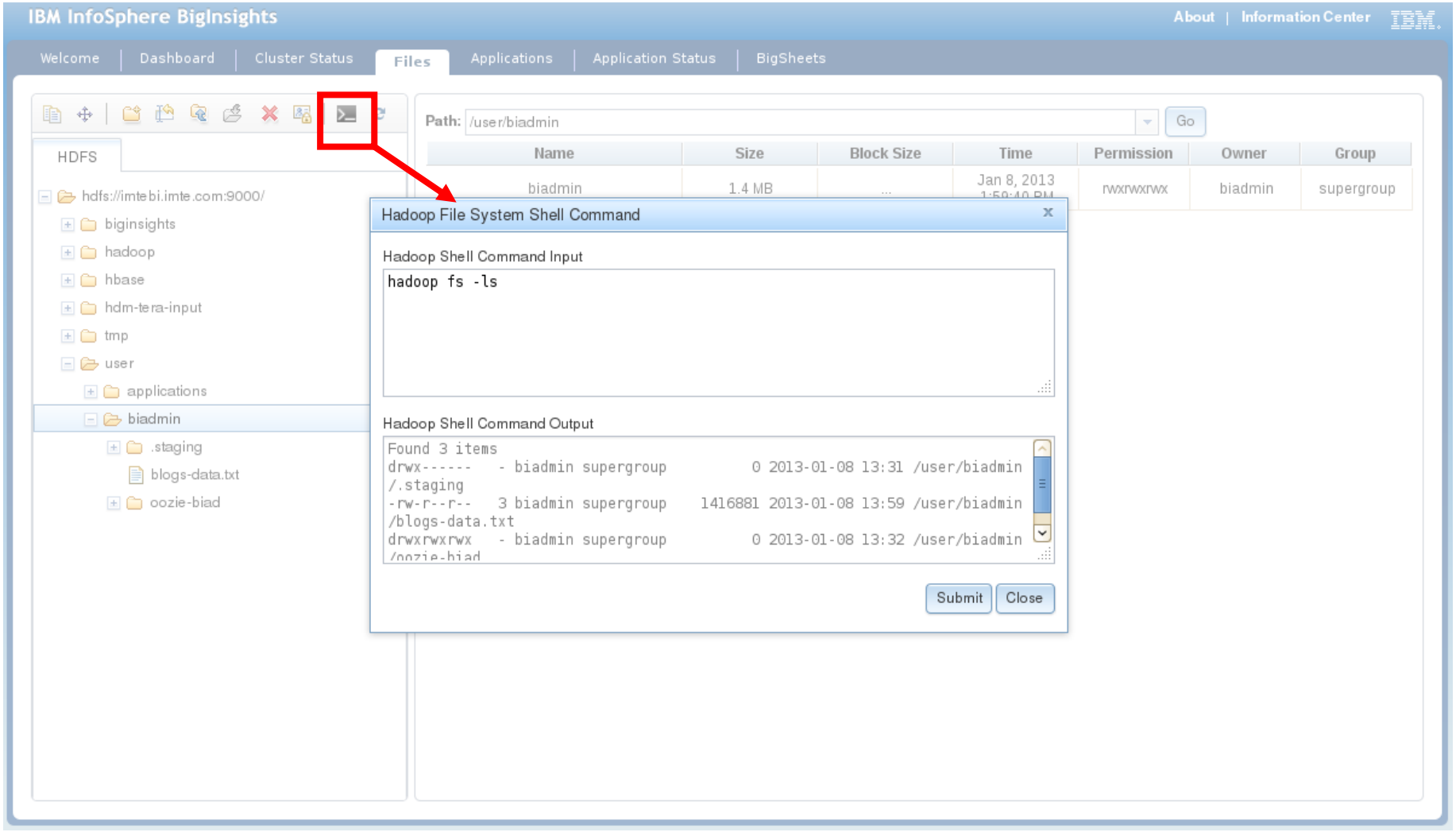
- **copyToLocal / get**
  - Copy files from fs into the local file system

```
hadoop fs -copyToLocal [-ignorecrc] [-crc]
                                    <src> <localdst>
```

Or

```
hadoop fs -get [-ignorecrc] [-crc]
                                    <src> <localdst>
```

Files Tab – hadoop shell command



Questions?