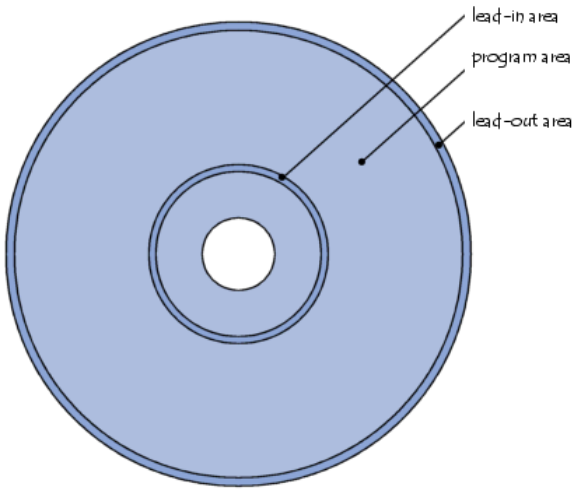
Hadoop Distributed File System (HDFS)

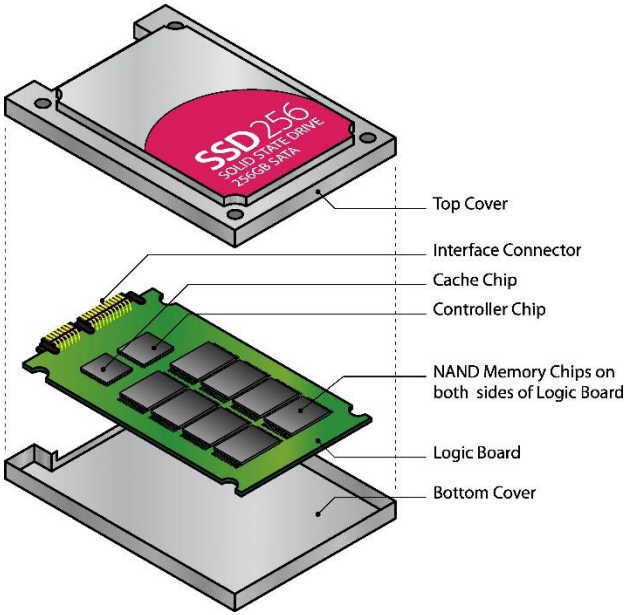
HDFS Overview

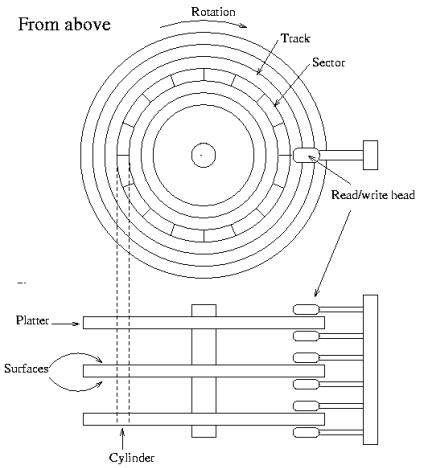
- A distributed file system
- Built on the architecture of Google File System (GFS)
- Shares a similar architecture to many other common distributed storage engines such as Amazon S3 and Microsoft Azure
- HDFS is a stand-alone storage engine and can be used in isolation of the query processing engine

Background on Disk Storage

- What are file systems and why do we need them?
- A file is a logical sequence of bits/bytes
- A physical disk stores data in sectors, tracks, tapes, blocks, … etc.







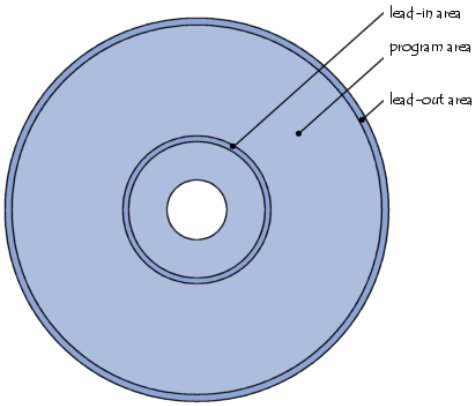
File System

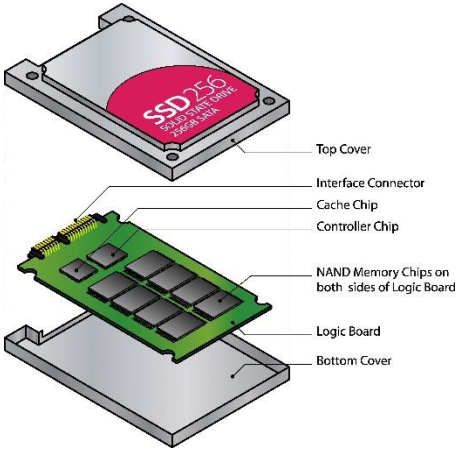
- Any file system, is a method to provide a high-level abstraction on physical disk to make it easier to store files

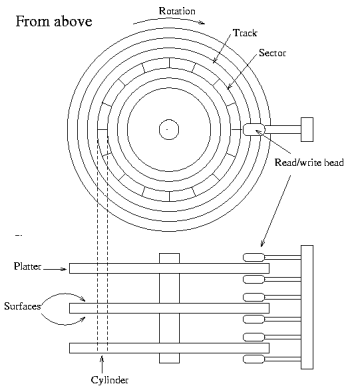
Files

Folders

File System





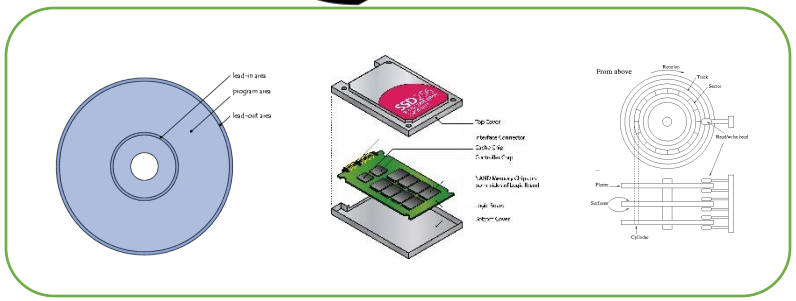


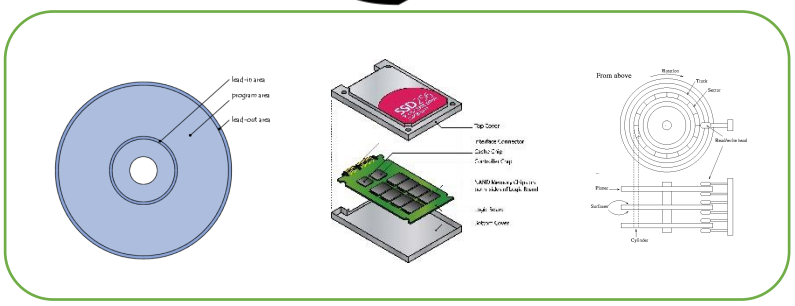
Distributed File System

Files

Folders

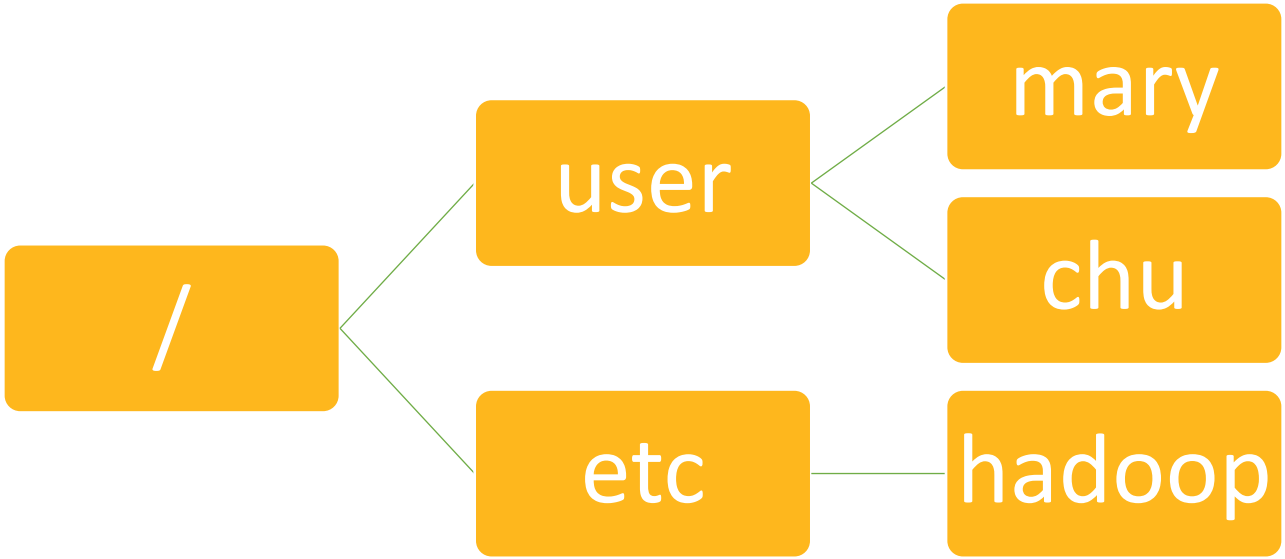
Distributed File System





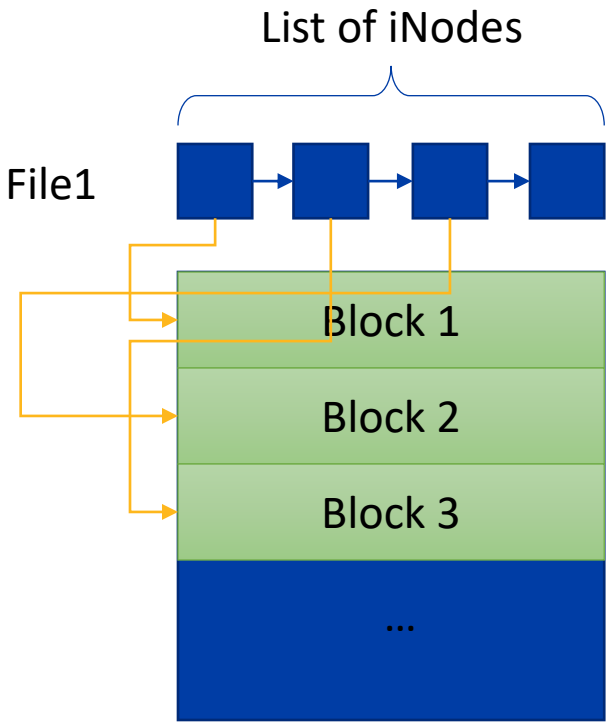
Analogy to Unix FS

The logical view is similar

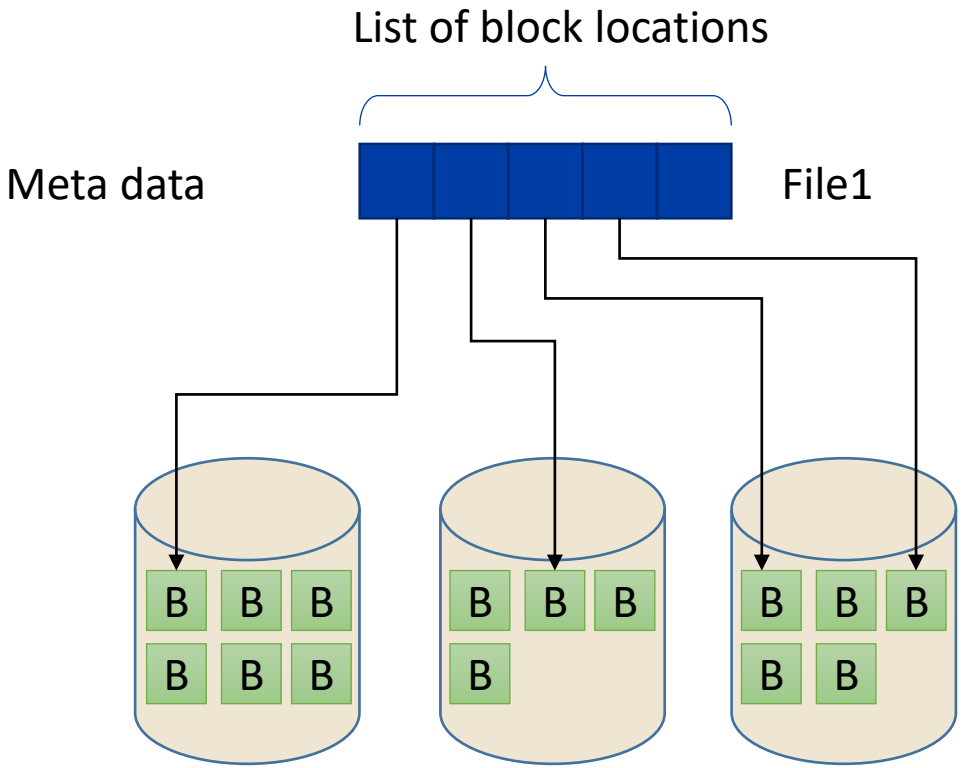


Analogy to Unix FS

The physical model is comparable

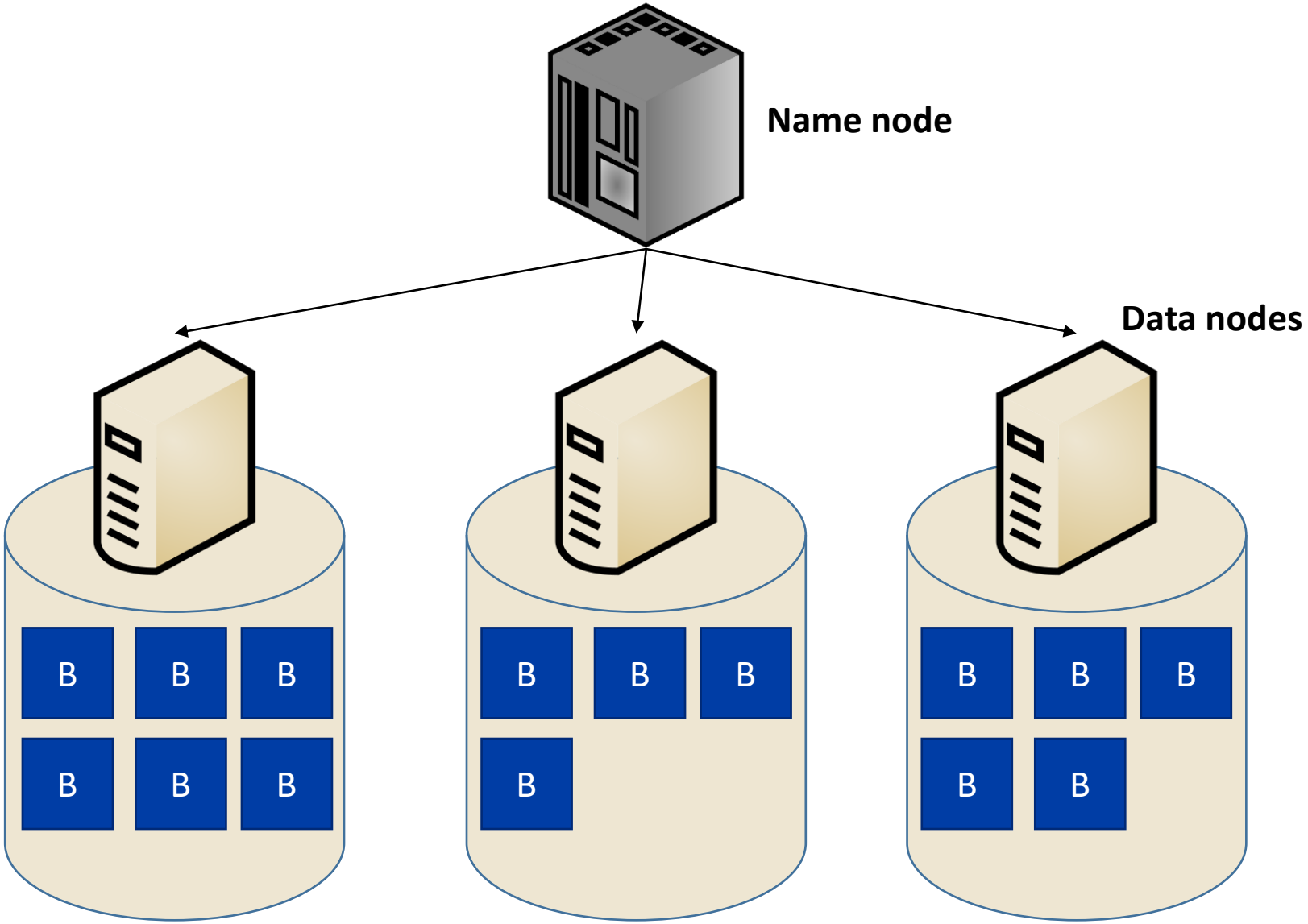


Unix

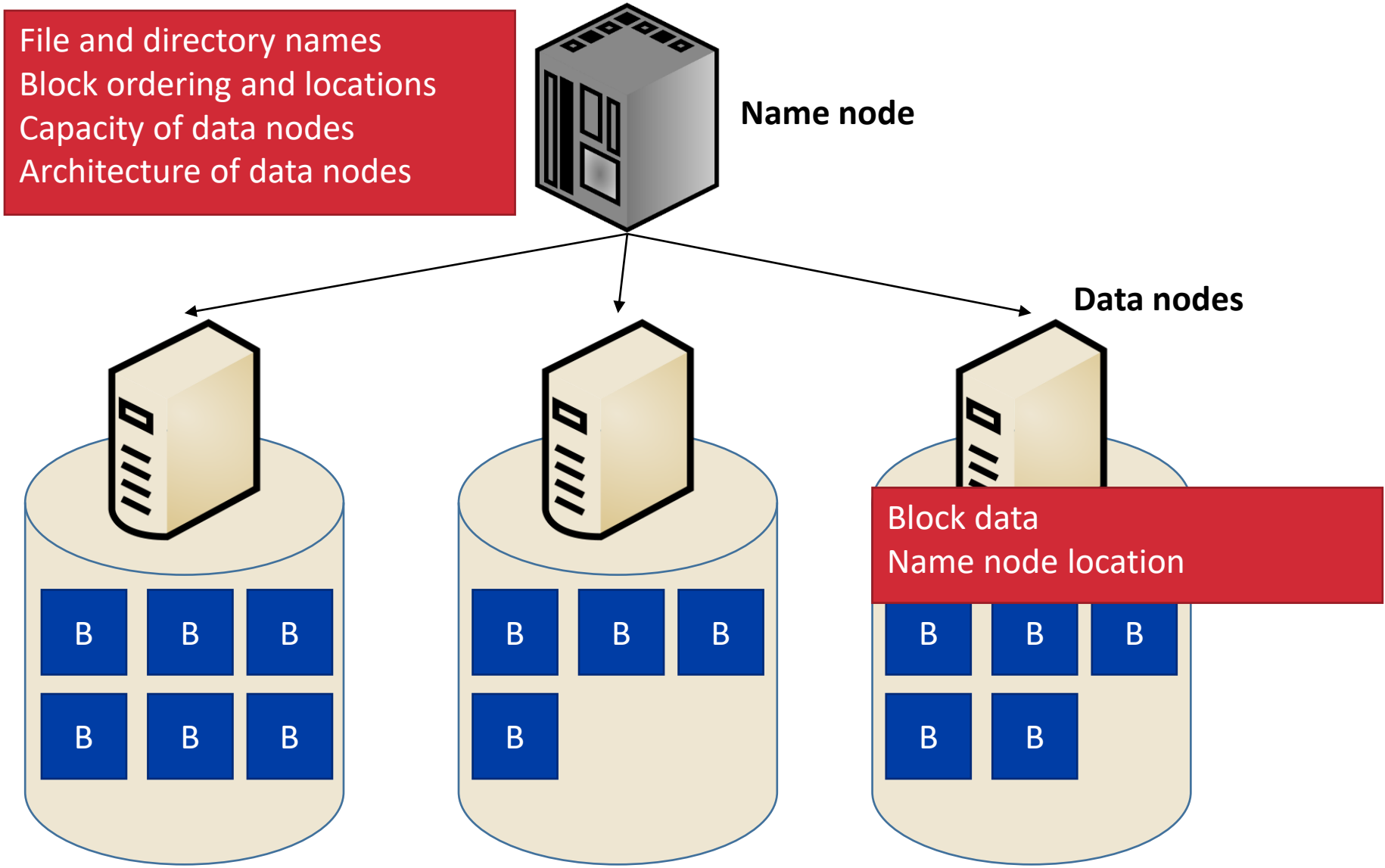


HDFS

HDFS Architecture

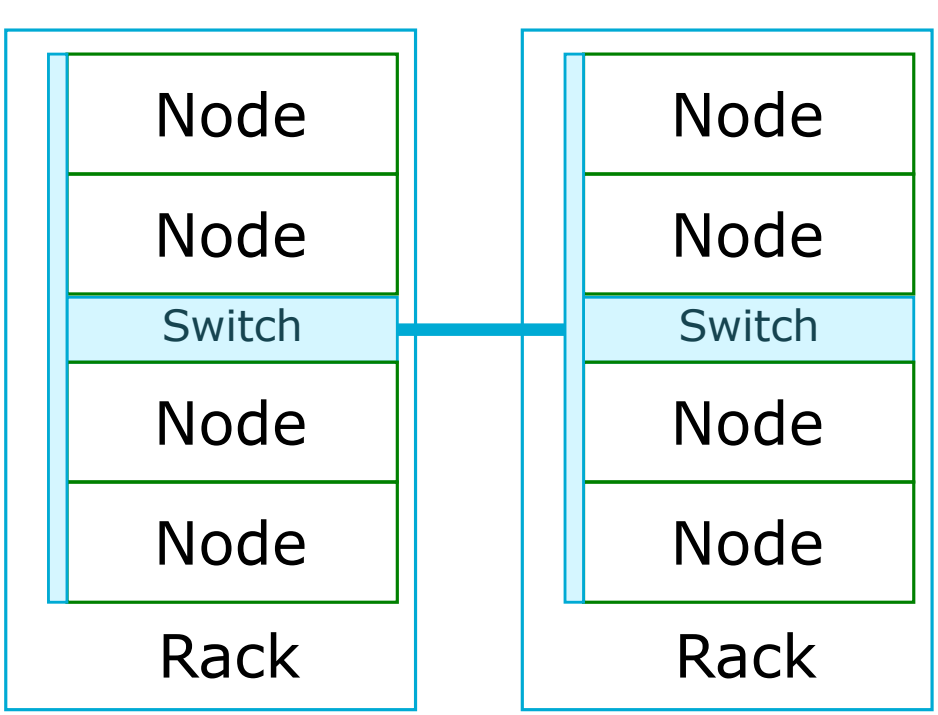


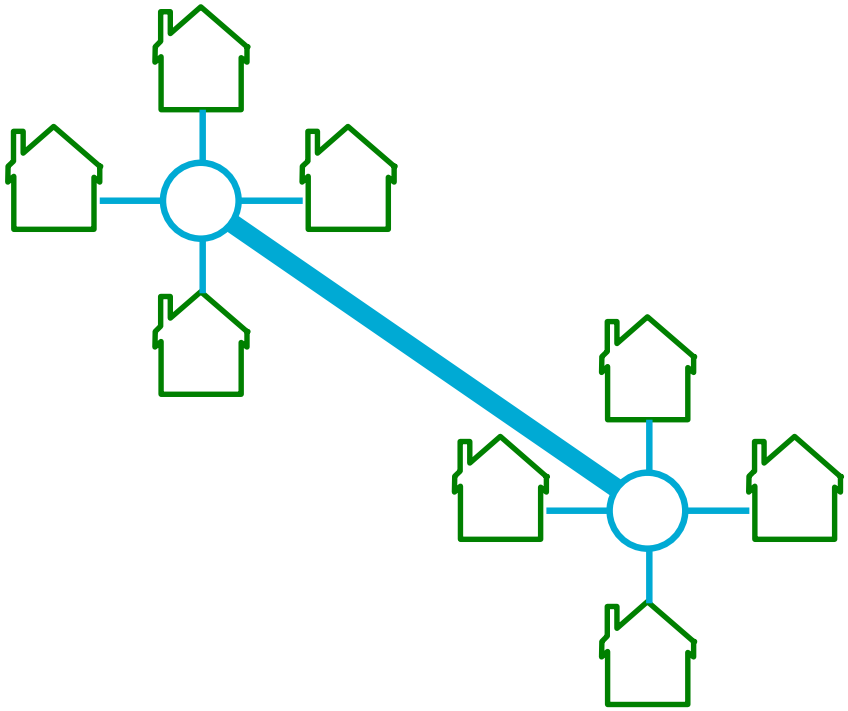
What is where?



Physical Cluster Layout

Analogy of racks





HDFS Shell

Manage the files from command line

HDFS Shell

- The easiest way to deal with HDFS is through its shell
- The commands are very similar to the Linux shell commands
- General format

```
hdfs dfs -<cmd> <arguments>
```

- So, instead of

```
mkdir –p myproject/mydir
```

- You will write

```
hdfs dfs -mkdir –p myproject/mydir
```

HDFS Shell

- In addition to regular commands, there are special commands in HDFS
  - `copyToLocal/get` Copies a file from HDFS to the local file system
  - `copyFromLocal/put` Copies a file from the local file system to HDFS
  - `setrep` Changes the replication factor
- A list of shell commands with usage
  - https://hadoop.apache.org/docs/r3.2.2/hadoop-project-dist/hadoop-common/FileSystemShell.html

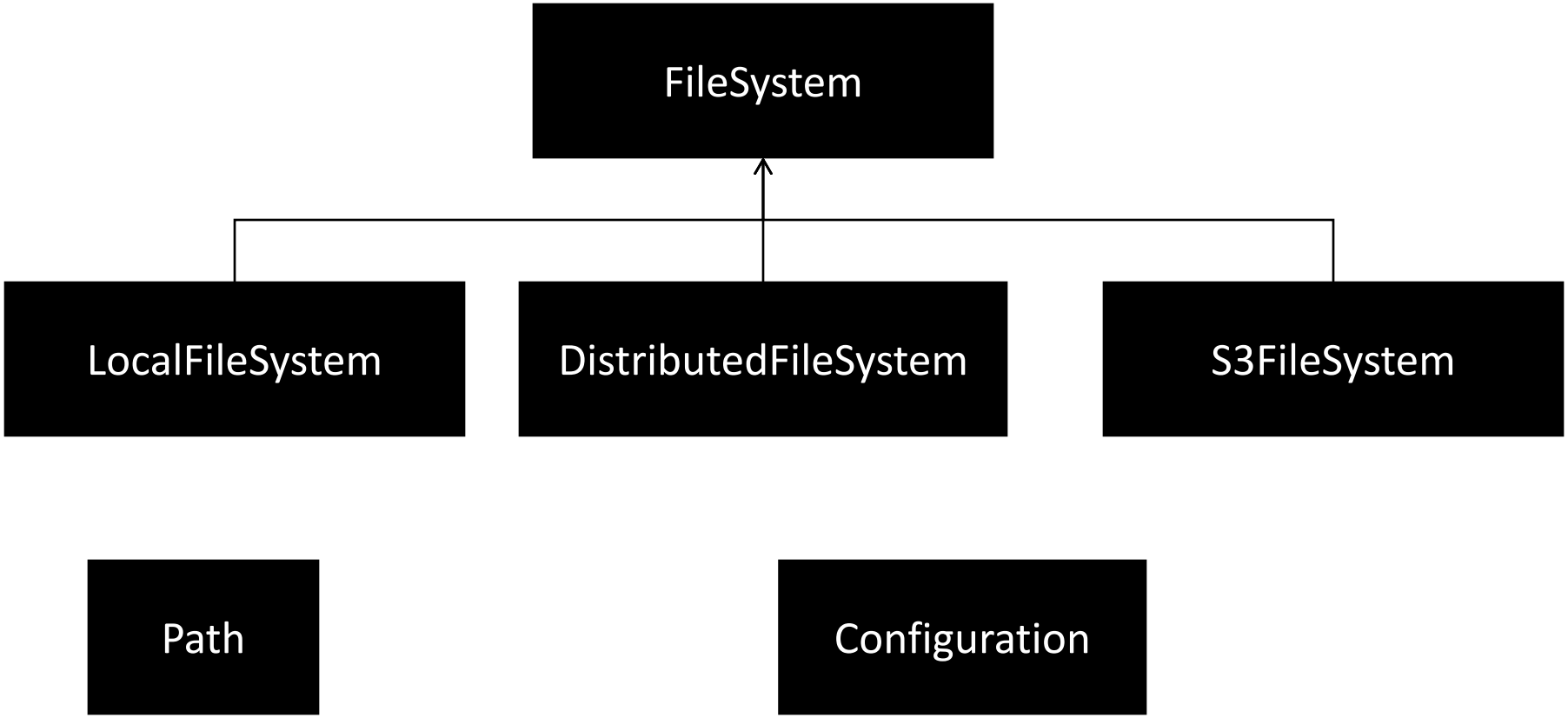
HDFS API

Mange the file system programmatically

FileSystem API

- HDFS provides a Java API that allows your programs to manage the files similar to the shell. It is even more powerful.
- For interoperability, the FileSystem API covers not only HDFS, but also the local file system and other common file systems, e.g., Amazon S3
- If you write your program in Hadoop FileSystem API, it will generally work for those file systems

HDFS API Basic Classes



HDFS API Classes

- Configuration: Holds system configuration such as where the master node is running and default system parameters
- Path: Stores a path to a file or directory
- FileSystem: An abstract class for file system commands

Fully Qualified Path

hdfs://masternode:9000/path/to/file

hdfs: the file system scheme. Other possible values are file, ftp, s3, …

masternode: the name or IP address of the node that hosts the master of the file system

9000: the port on which the master node is listening

/path/to/file: the absolute path of the file

Shorter Path Forms

- file: relative path to the current working directory in the default file system
- /path/to/file: Absolute path to a file in the default* file system (as configured)
- hdfs://path/to/file: Use the default* values for the master node and port
- hdfs://masternode/path/tofile: Use the given masternode name or IP and the default* port

*All the defaults are in the Configuration object

HDFS API

Create the file system

```
Configuration conf = new Configuration();
Path path = new Path("…");
FileSystem fs = path.getFileSystem(conf);

// To get the local FS
fs = FileSystem.getLocal(conf);

// To get the default FS
fs = FileSystem.get(conf);
```

HDFS API

Create a new file

```
FSDataOutputStream out = fs.create(path, …);
```

Delete a file

```
fs.delete(path, recursive);

fs.deleteOnExit(path); // For temporary files
```

Rename/Move a file

```
fs.rename(oldPath, newPath);
```

HDFS API

Open a file for reading

```
FSDataInputStream in = fs.open(path, …);
```

Seek to a different location for random access

```
in.seek(pos);
in.seekToNewSource(pos);
```

HDFS API

Concatenate

```
fs.concat(destination, src[]);
```

Get file metadata

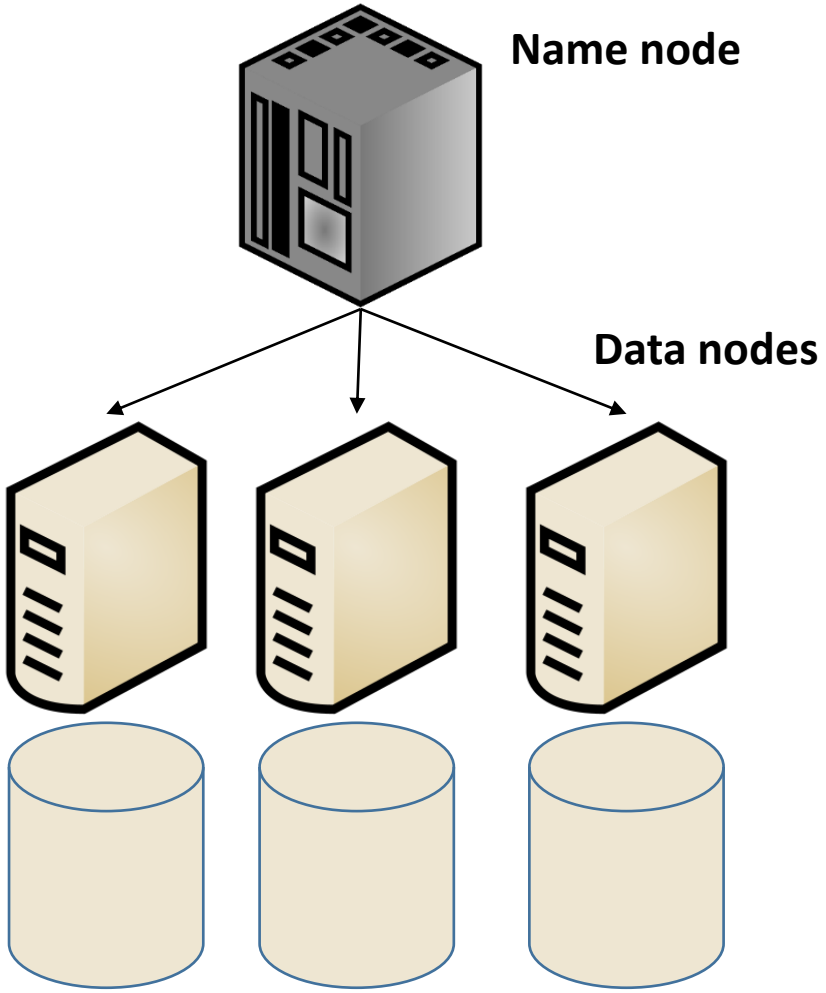
```
fs.getFileStatus(path);
```

Get block locations

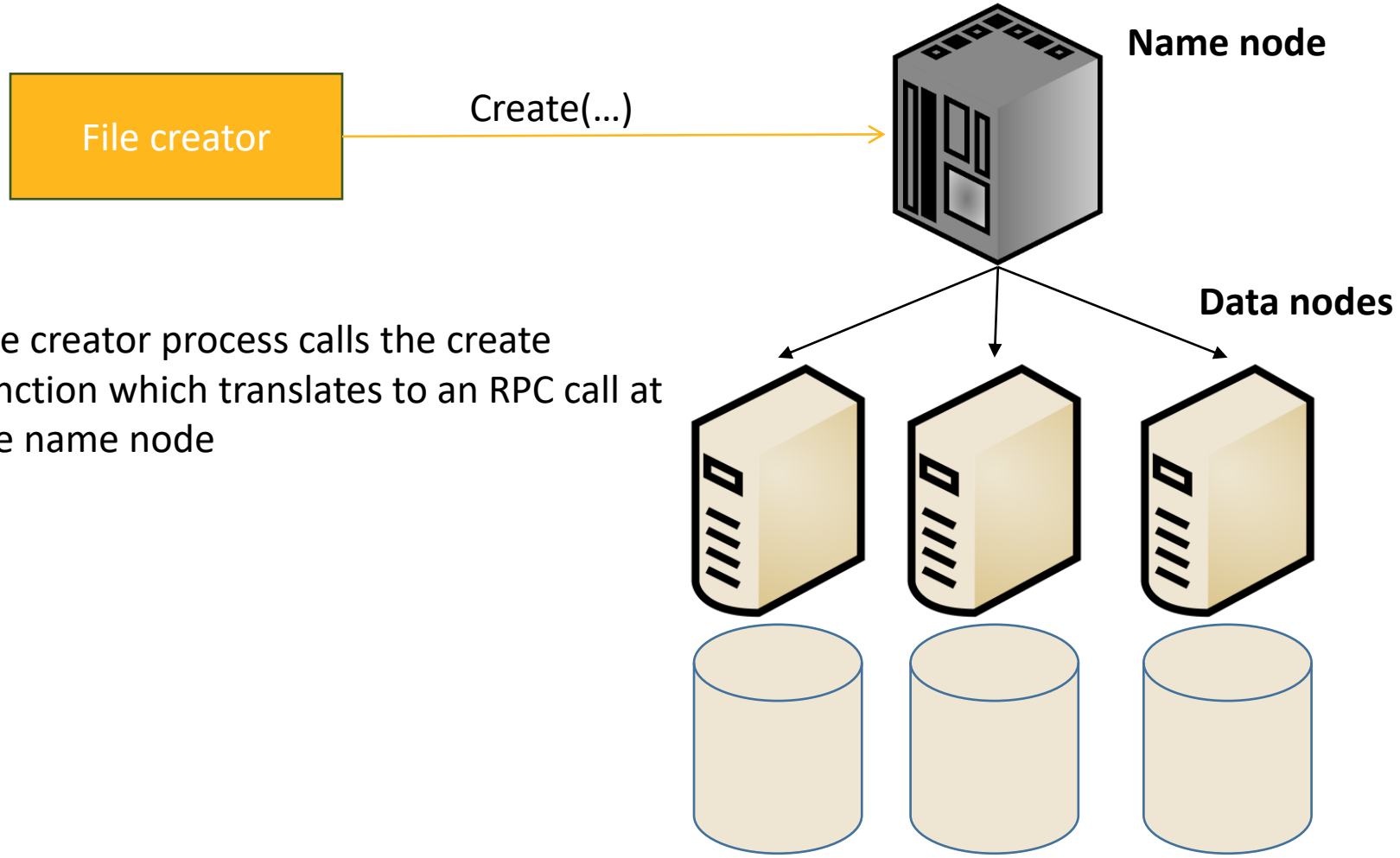
```
fs.getFileBlockLocations(path, from, to);
```

HDFS Writing Process

File creator

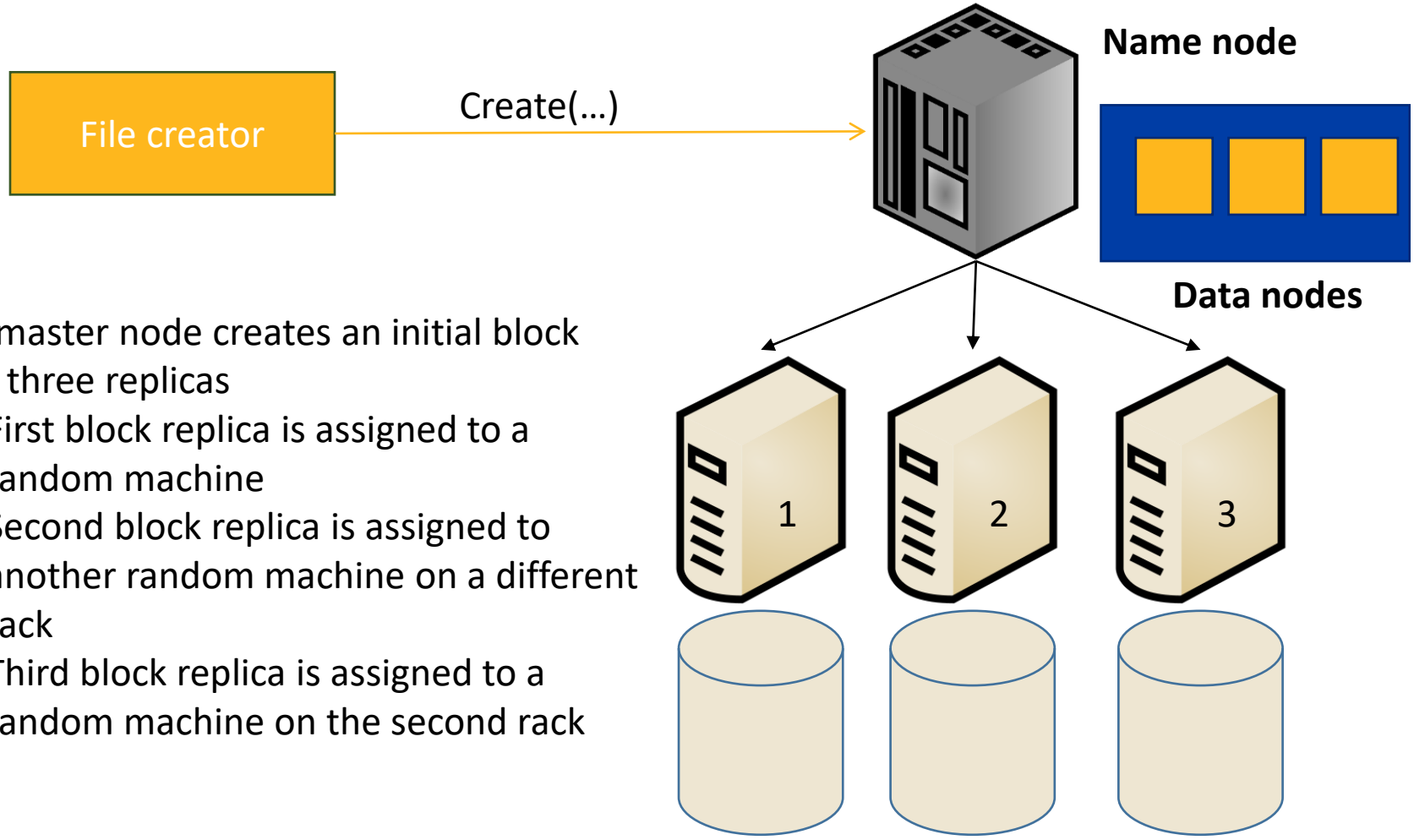


HDFS Writing Process



The creator process calls the create function which translates to an RPC call at the name node

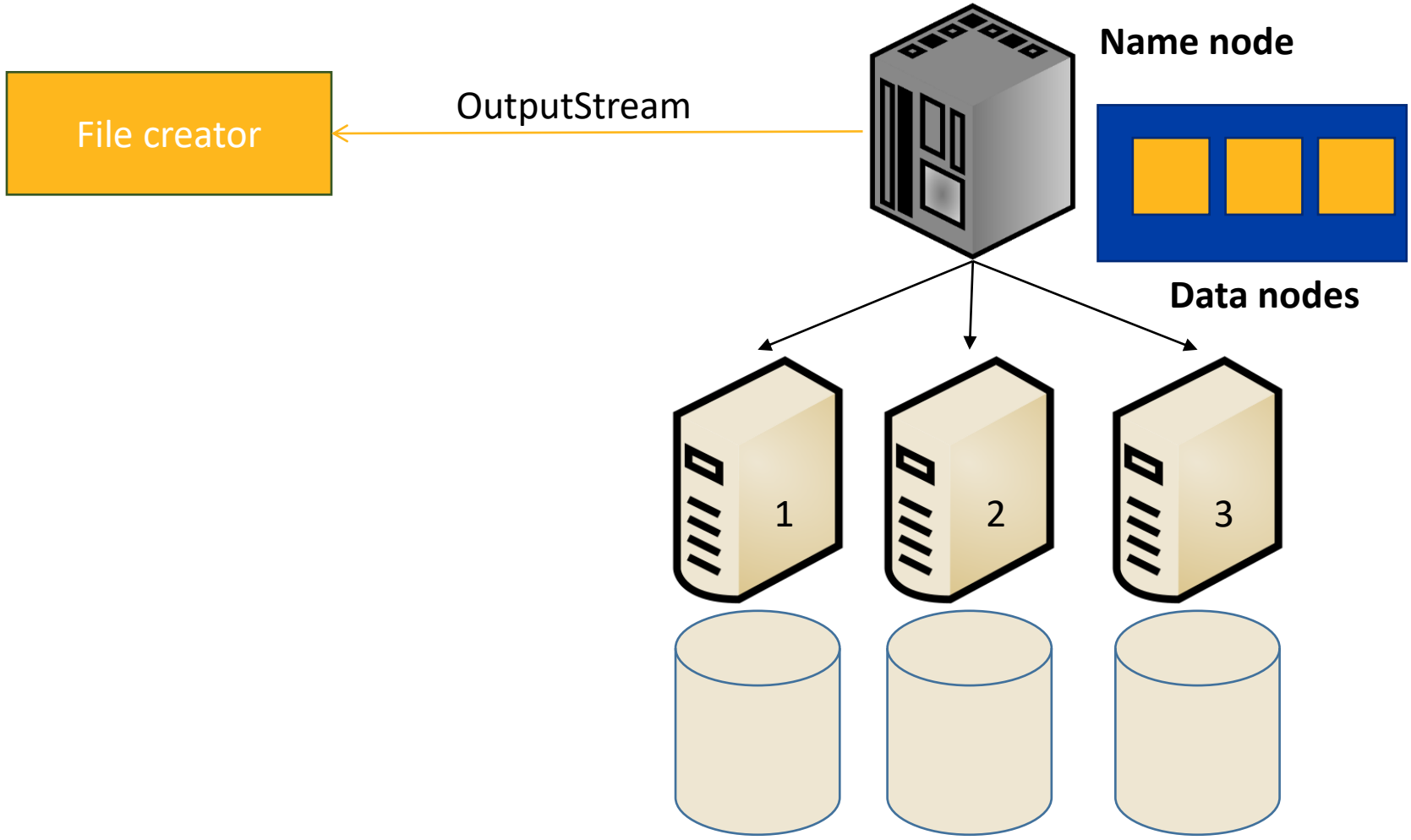
HDFS Writing Process



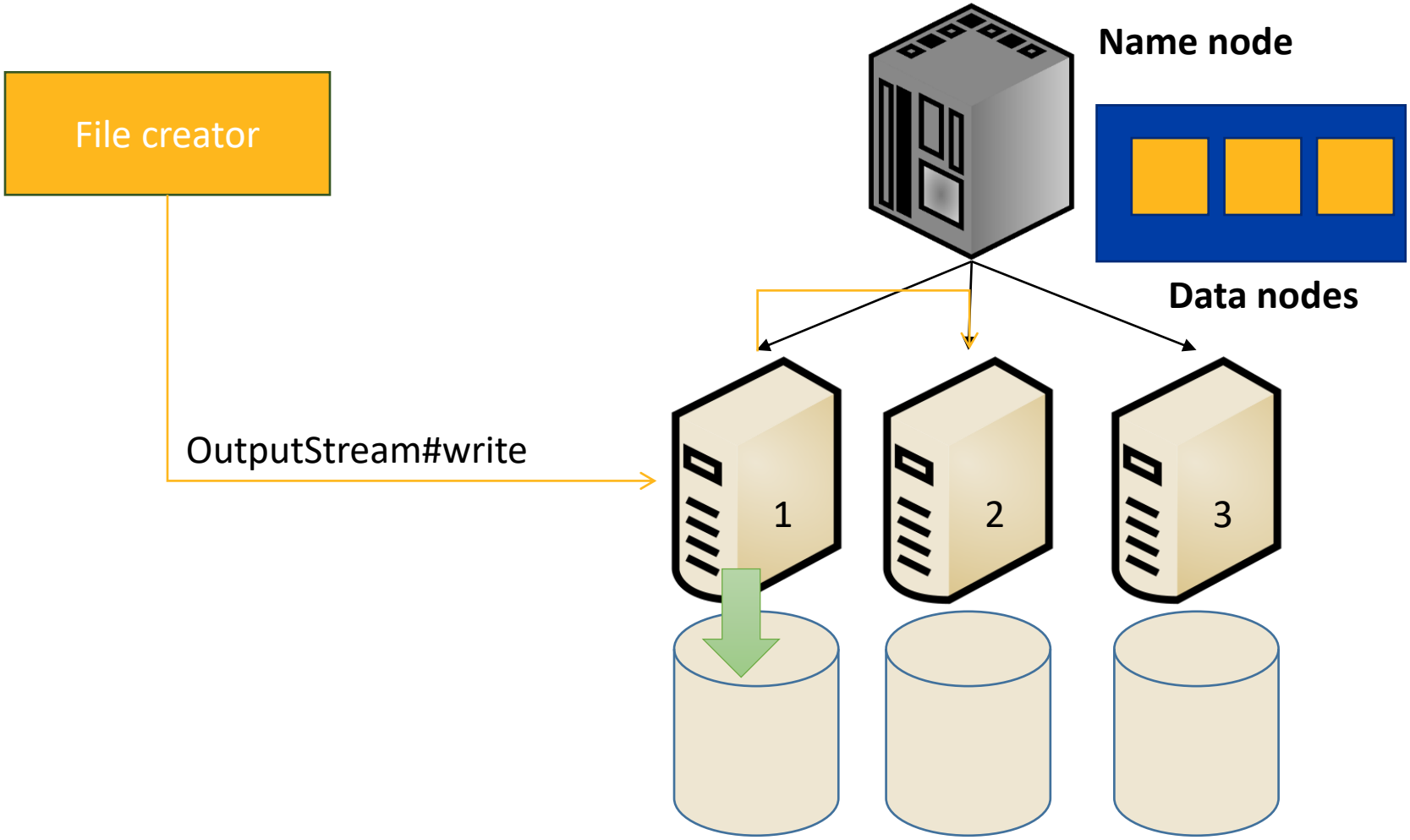
The master node creates an initial block with three replicas

1. First block replica is assigned to a random machine
2. Second block replica is assigned to another random machine on a different rack
3. Third block replica is assigned to a random machine on the second rack

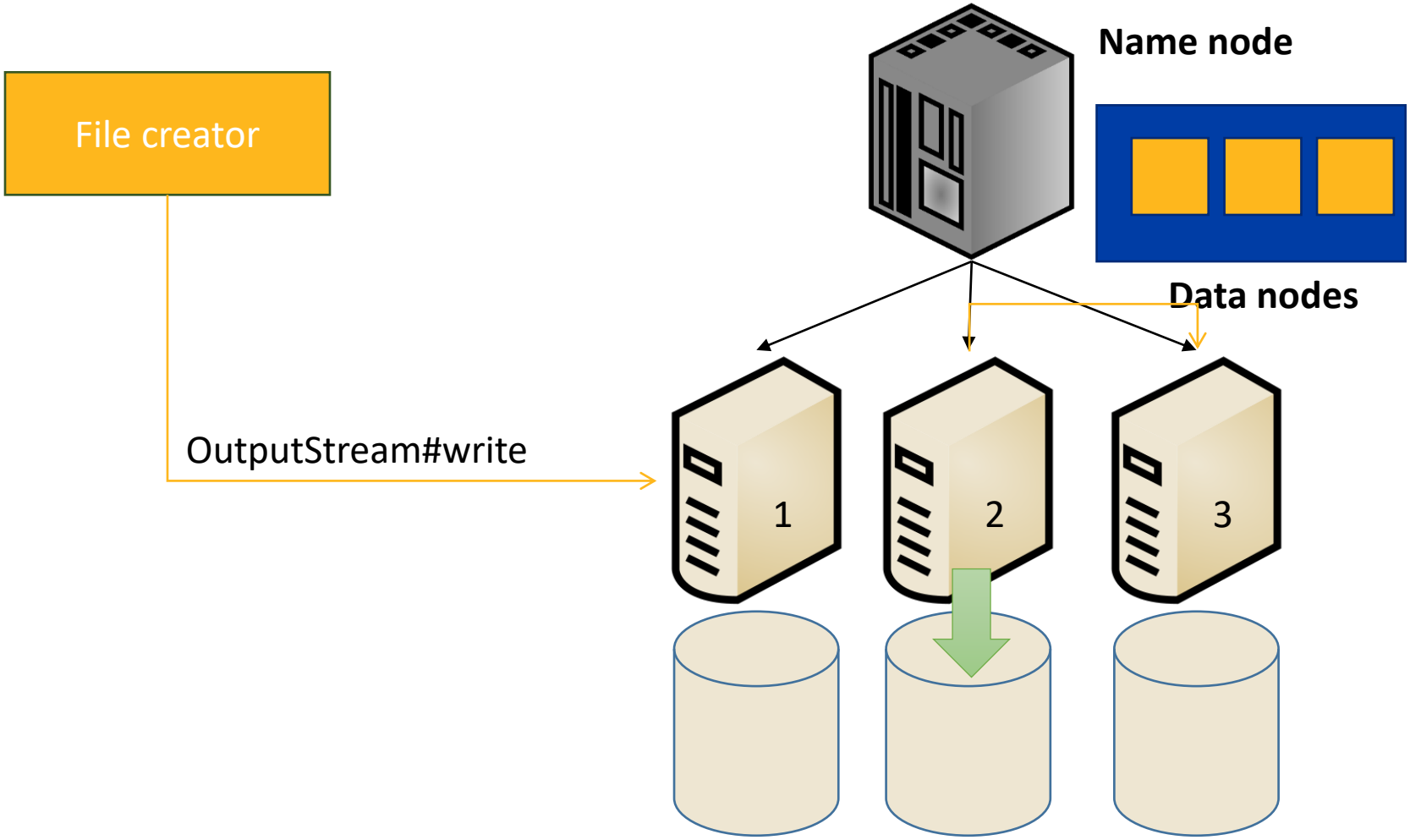
HDFS Writing Process



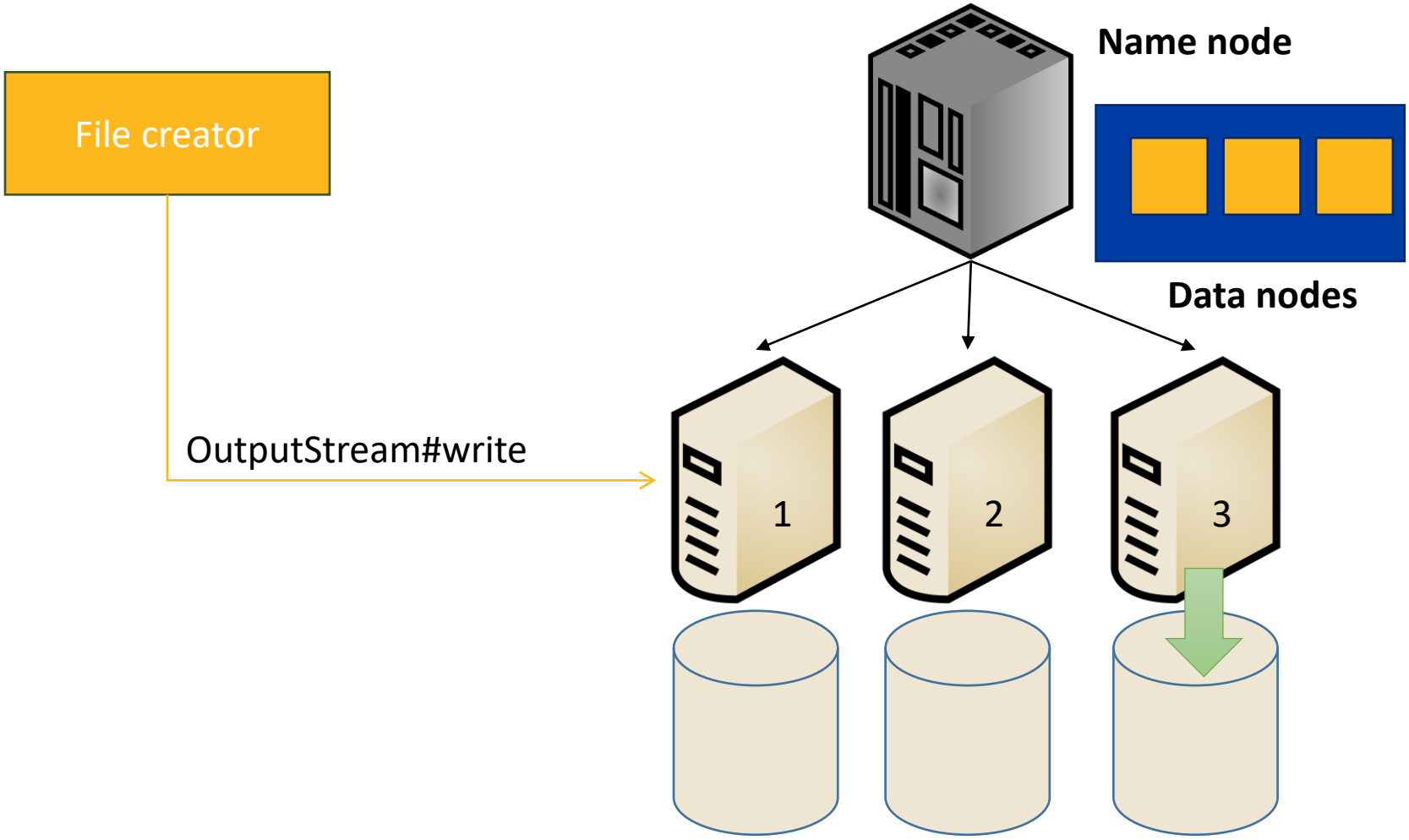
HDFS Writing Process



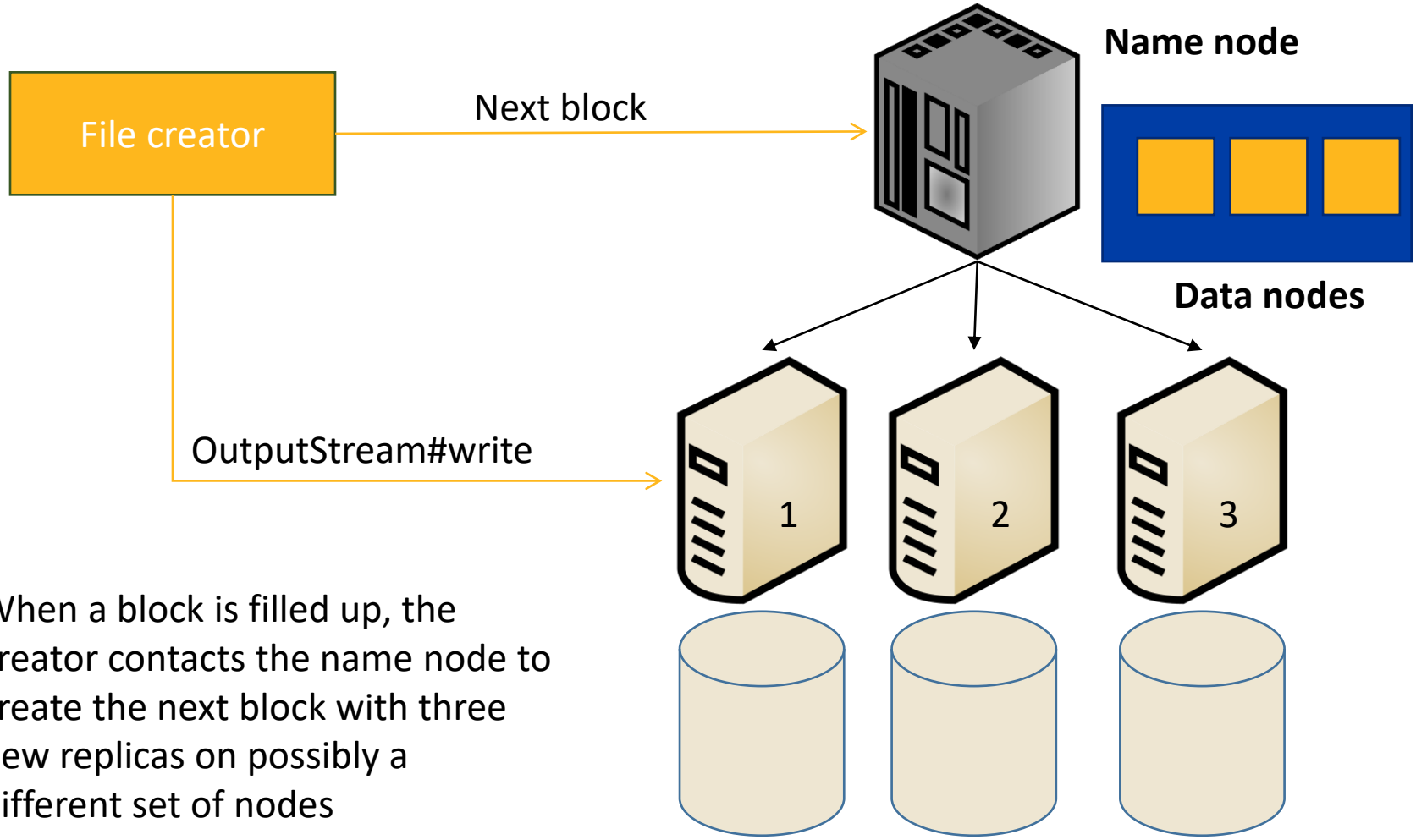
HDFS Writing Process



HDFS Writing Process



HDFS Writing Process



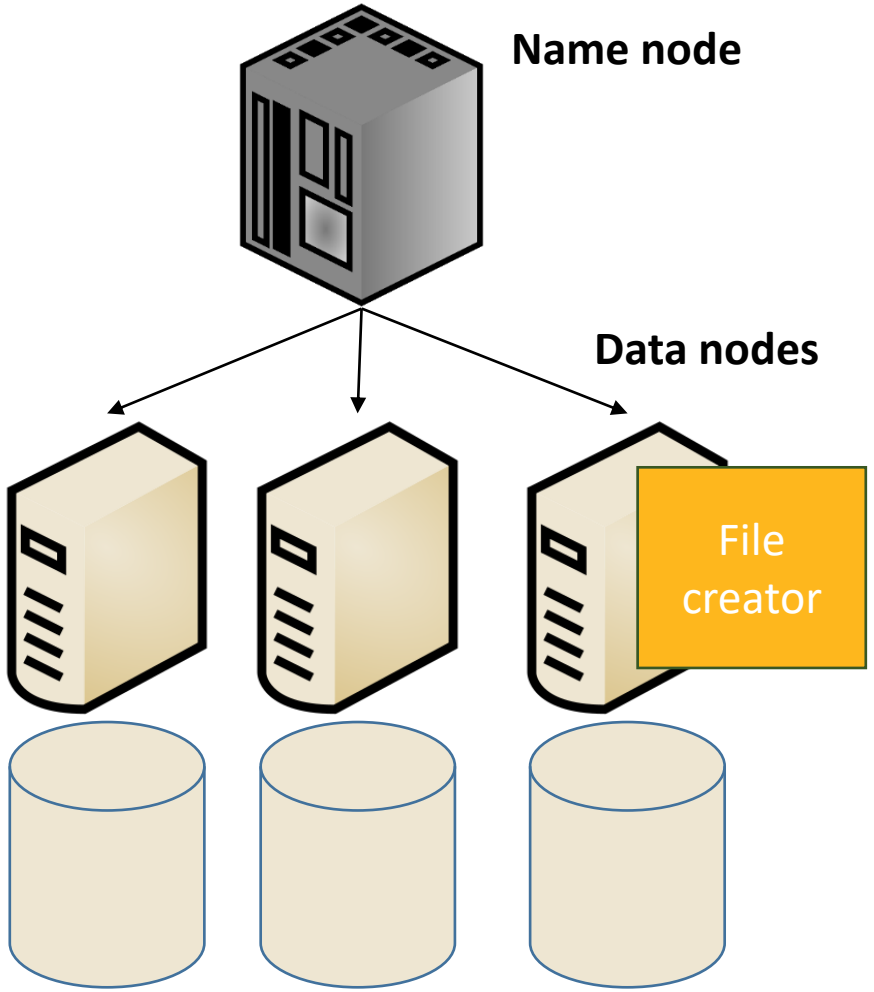
When a block is filled up, the creator contacts the name node to create the next block with three new replicas on possibly a different set of nodes

Notes about writing to HDFS

- Data transfers of replicas are pipelined
- The data does **not** go through the name node
- Random writing is **not** supported
- Appending to a file is supported but it creates a new block

Writing from a datanode

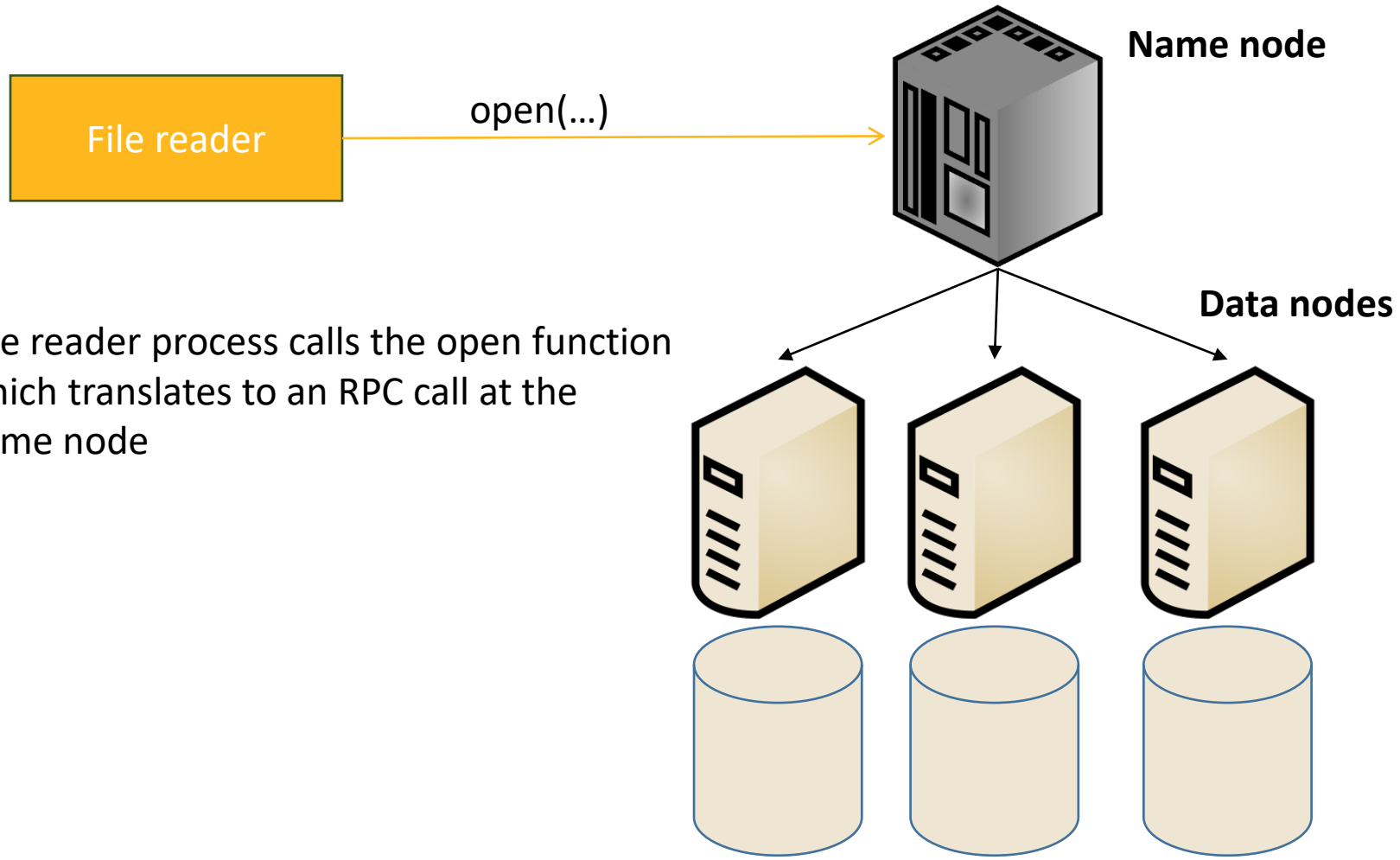
If the file creator is running on one of the data nodes, the first replica is always assigned to that node
The second and third replicas are assigned as before



Reading from HDFS

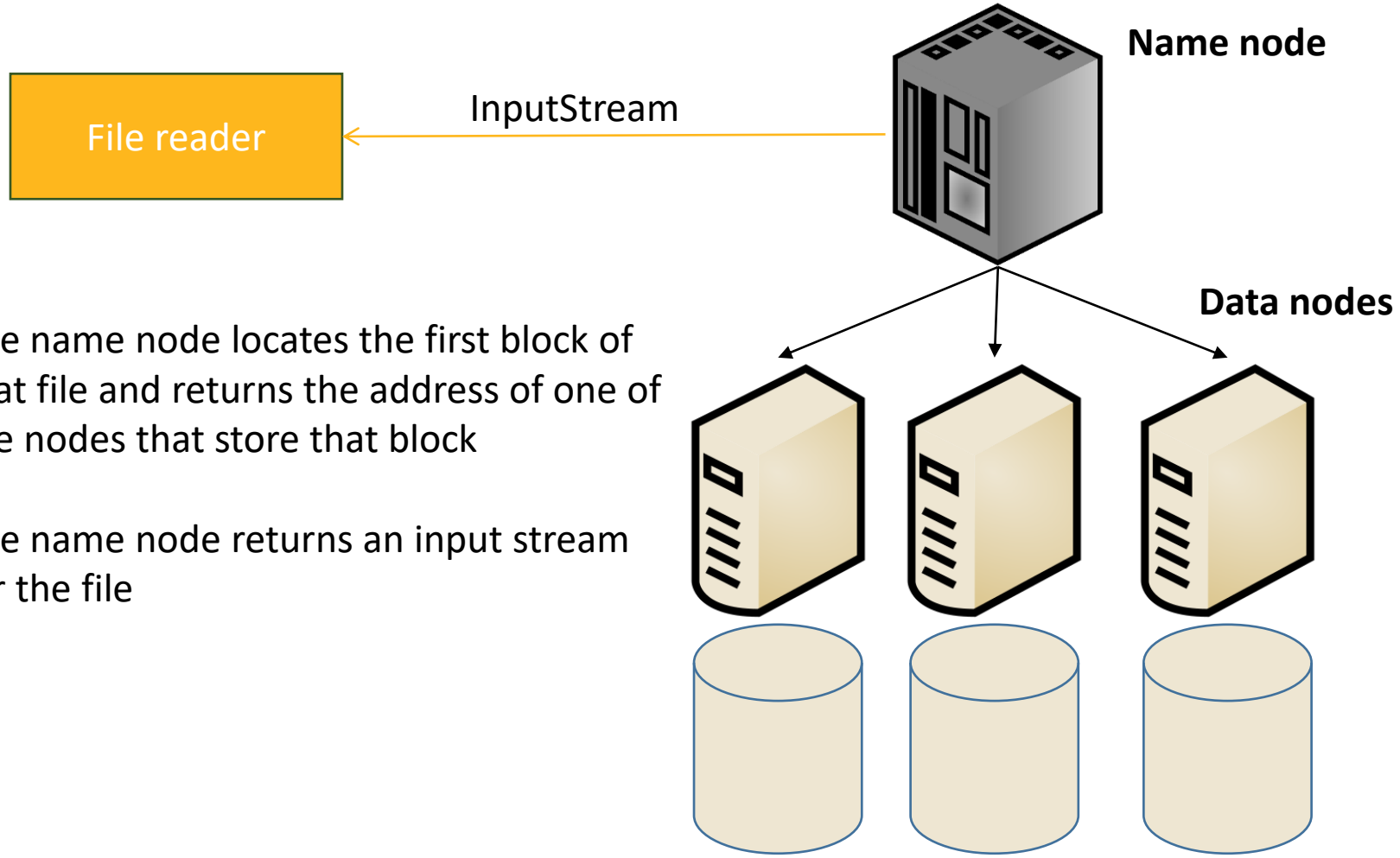
- Reading is relatively easier
- No replication is needed
- Replication can be exploited
- Random reading *is* allowed

HDFS Reading Process



The reader process calls the open function which translates to an RPC call at the name node

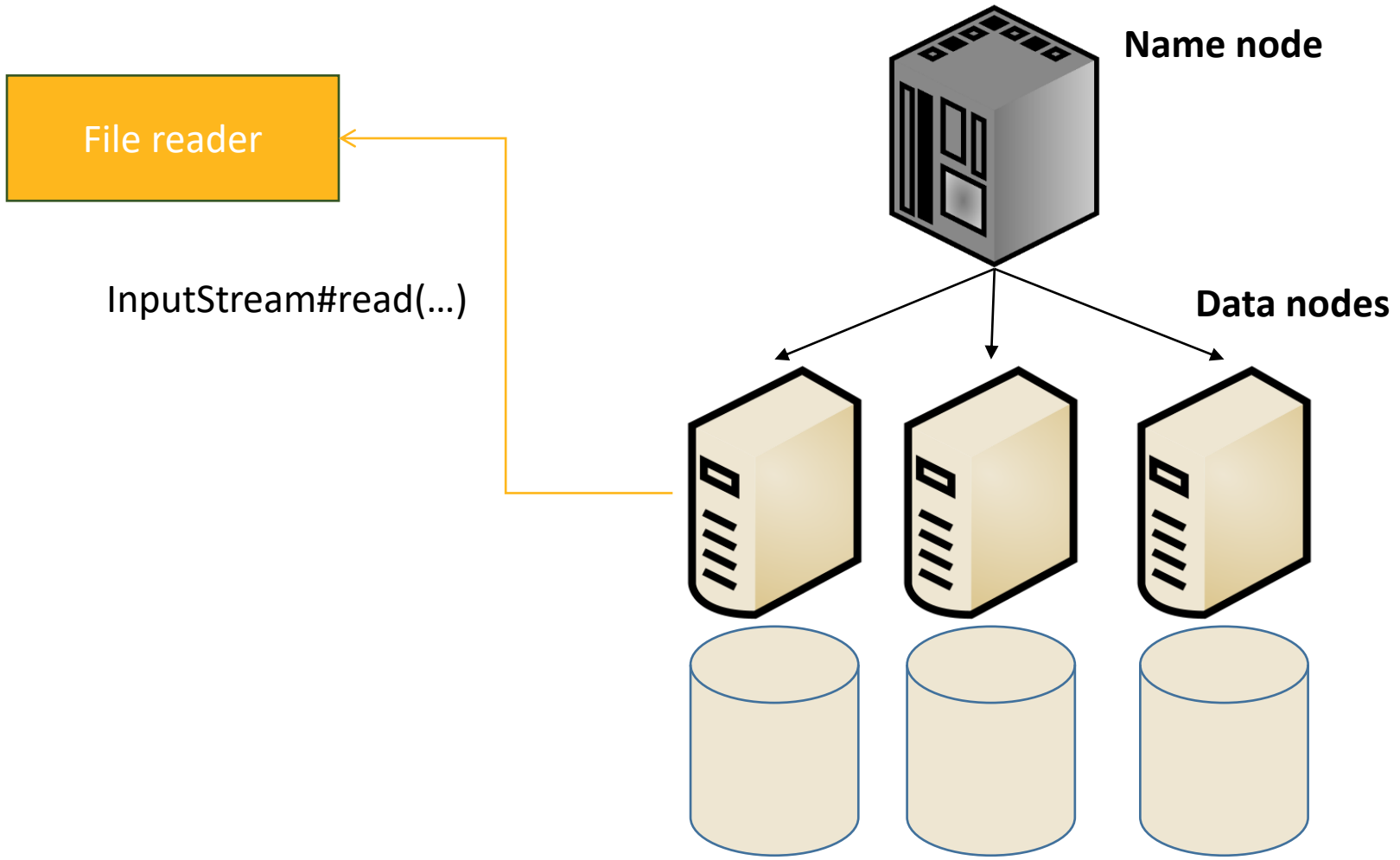
HDFS Reading Process



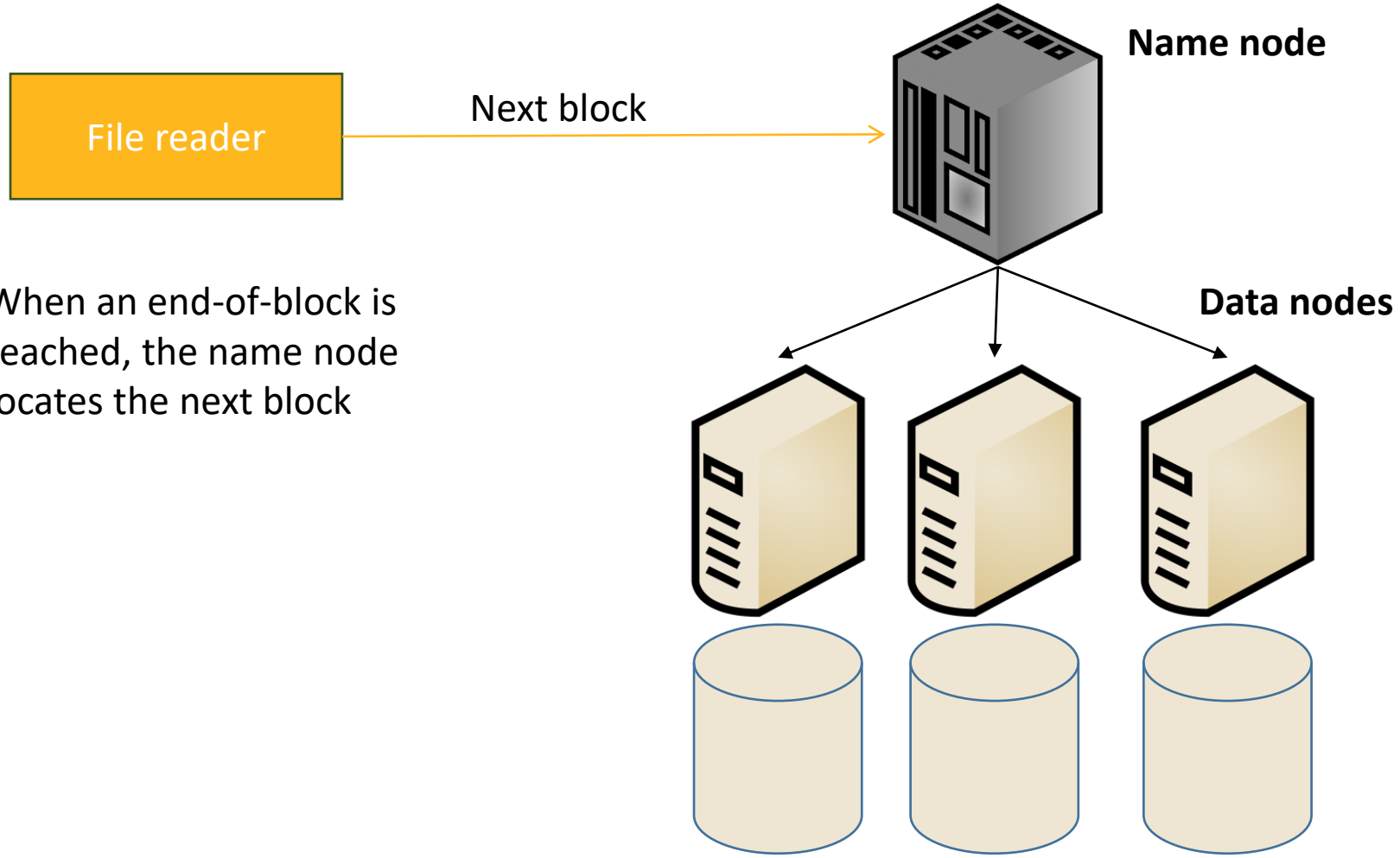
The name node locates the first block of that file and returns the address of one of the nodes that store that block

The name node returns an input stream for the file

HDFS Reading Process

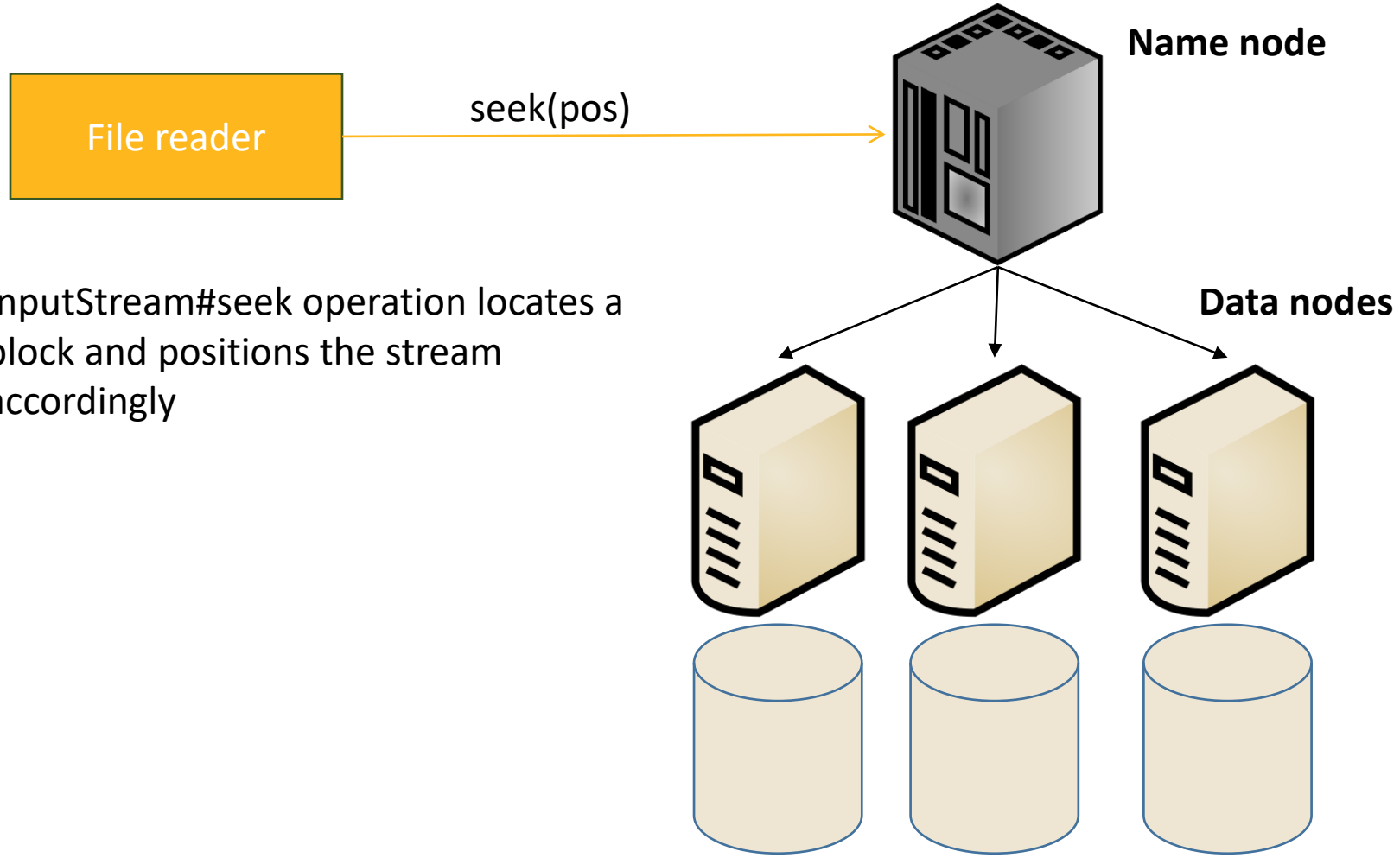


HDFS Reading Process



When an end-of-block is reached, the name node locates the next block

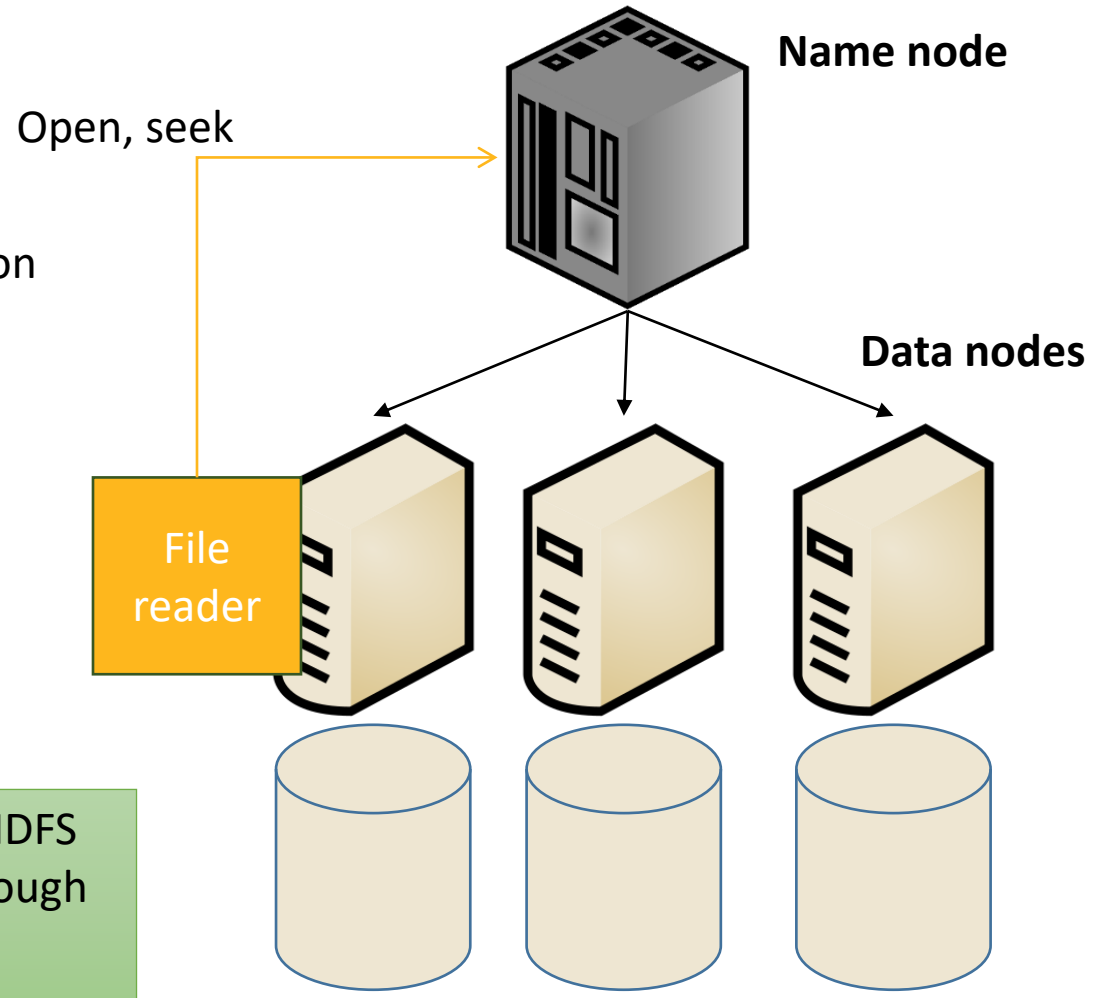
HDFS Reading Process



InputStream#seek operation locates a block and positions the stream accordingly

Reading from a datanode

1. If the block is locally stored on the reader, this replica is chosen to read
2. If not, a replica on another machine in the same rack is chosen
3. Any other random block replica is chosen



When self-reading occurs, HDFS can make it much faster through a feature called short-circuit

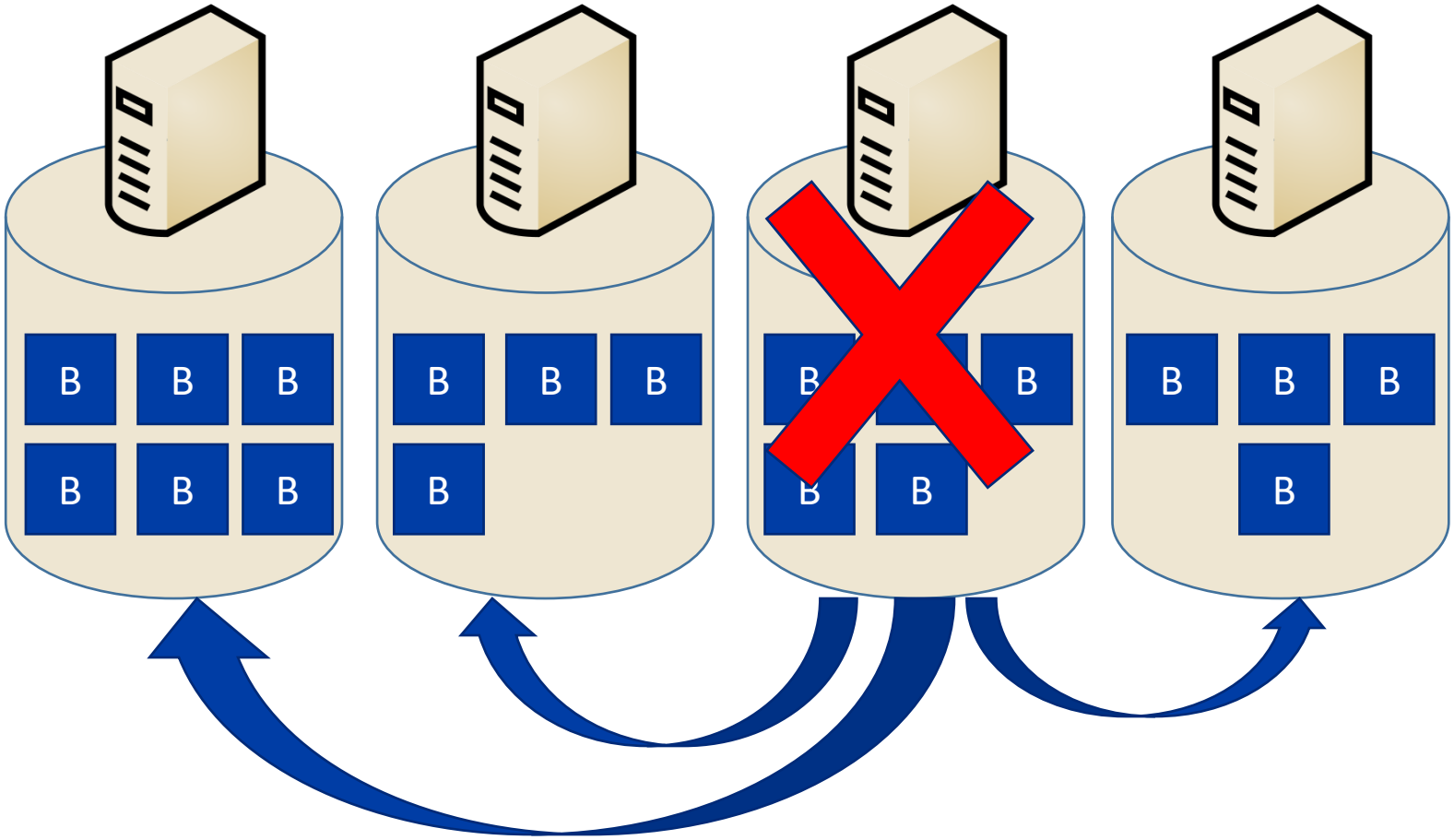
Notes About Reading

- The API is much richer than the simple open/seek/close API
  - You can retrieve block locations
  - You can choose a specific replica to read
- The same API is generalized to other file systems including the local FS and S3
- Review question: Compare random access read in local file systems to HDFS

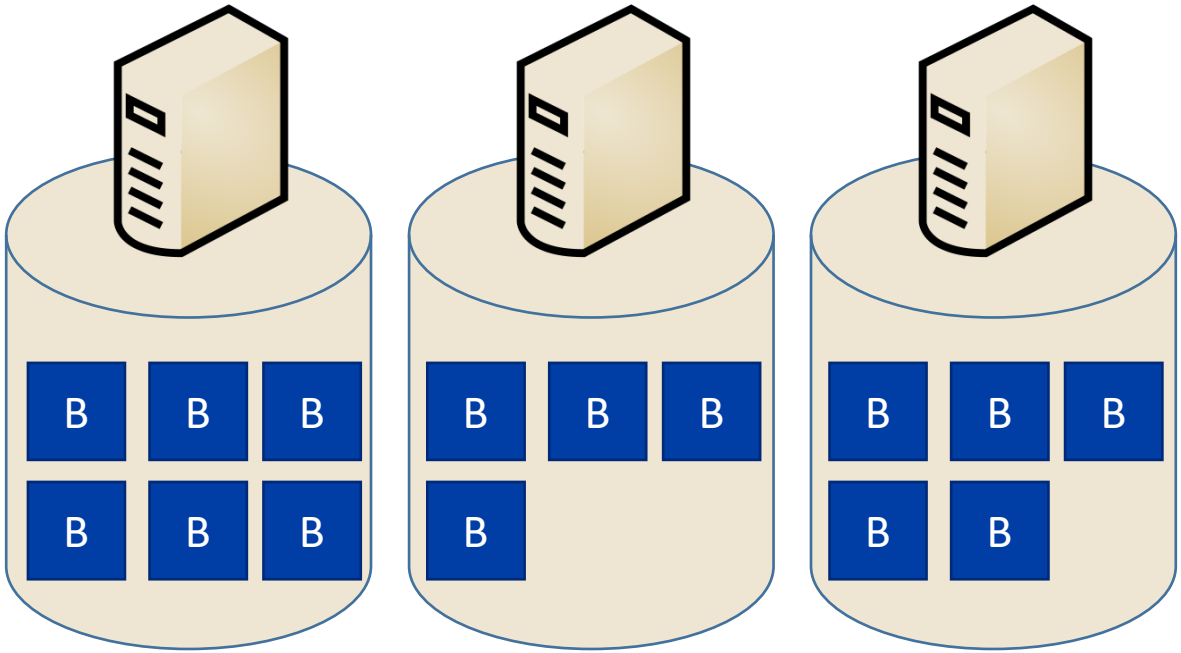
HDFS Special Features

- Node decommission
- Load balancer
- Cheap concatenation

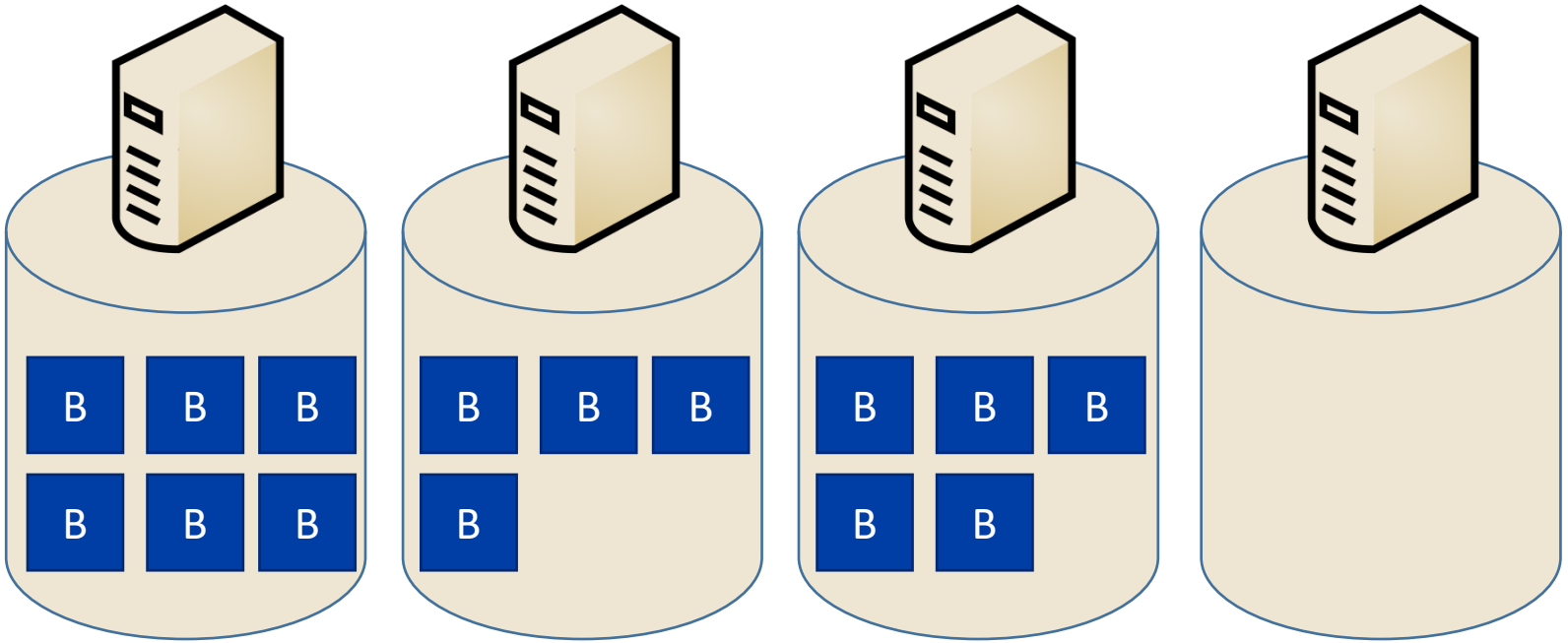
Node Decommission



Load Balancing

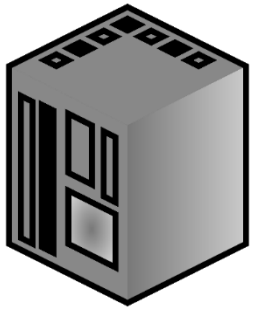


Load Balancing

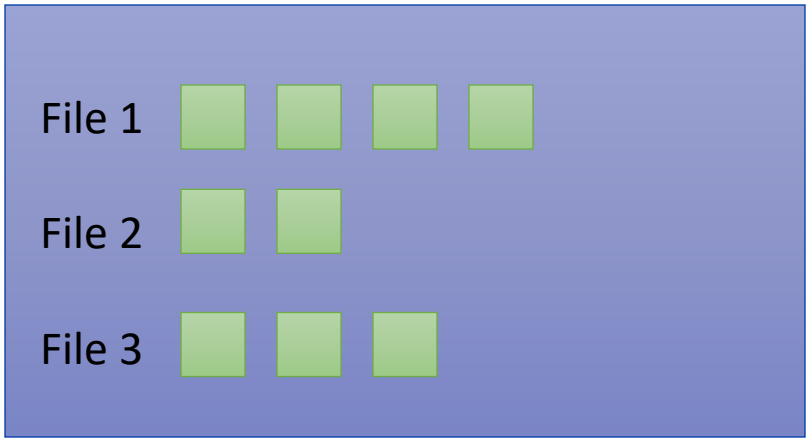


Start the load balancer

Cheap Concatenation



Name node



Concatenate File 1 + File 2 + File 3 ➔ File 4

Rather than creating new blocks, HDFS can just change the metadata in the name node to delete File 1, File 2, and File 3, and assign their blocks to a new File 4 in the right order.

Conclusion

- HDFS is a general-purpose distributed file system
- Provides a similar abstraction to other file systems
- HDFS provides two interfaces
  - Shell script. Similar to Linux and MacOS
  - Java API: For programmatic access
- The FileSystem API applies to other file systems including the local file system and Amazon S3

Further Readings

- HDFS Architecture
  - https://hadoop.apache.org/docs/r3.2.2/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html
- Shell commands
  - https://hadoop.apache.org/docs/r3.2.2/hadoop-project-dist/hadoop-common/FileSystemShell.html
- FileSystem API
  - https://hadoop.apache.org/docs/r3.2.2/api/org/apache/hadoop/fs/FileSystem.html