Hadoop Distributed File System (HDFS)

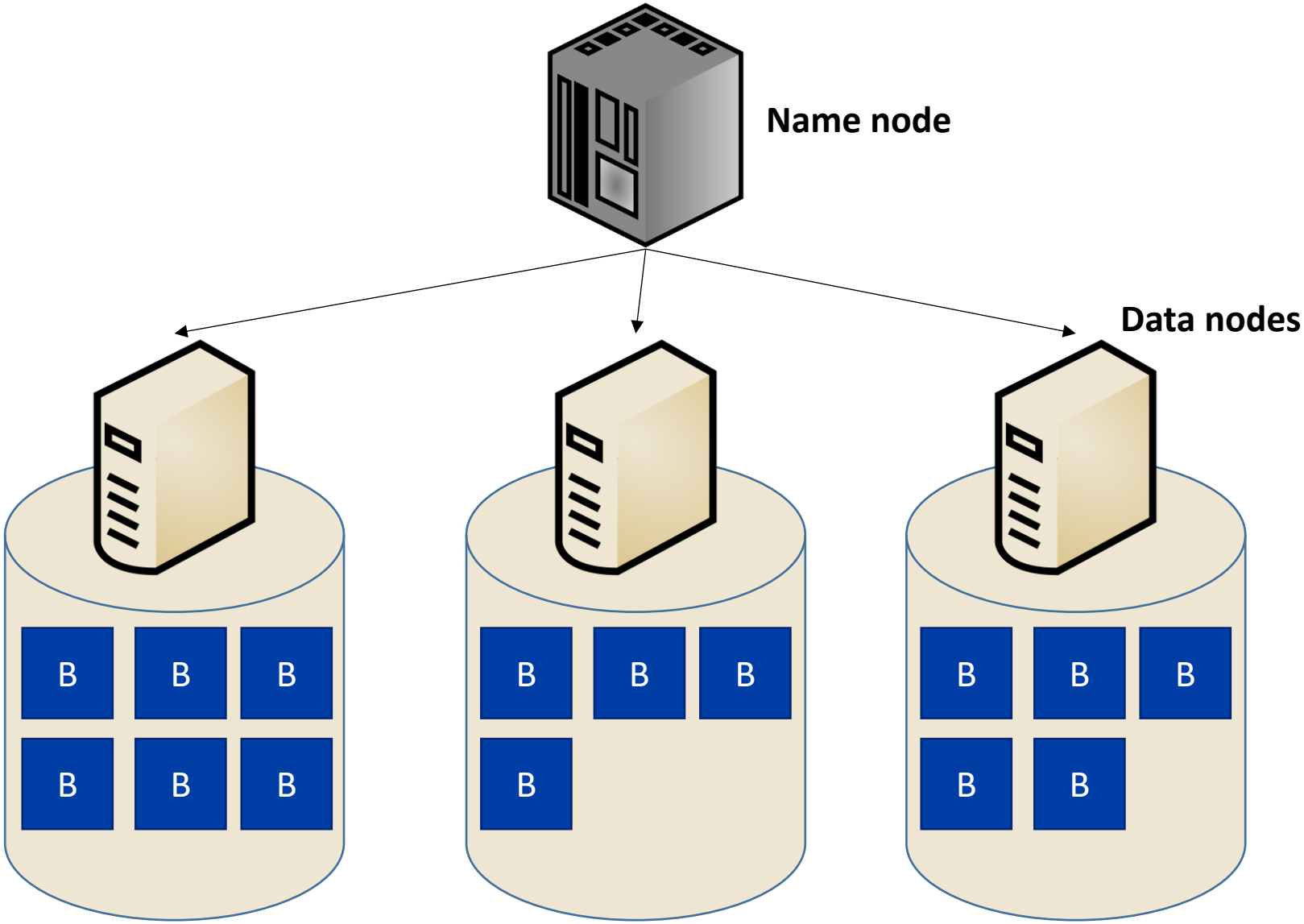
HDFS Overview

- 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
- Even if you do not use Hadoop MapReduce, you will probably still use HDFS

HDFS Topics

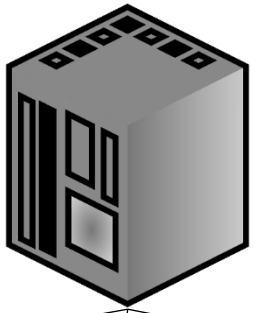
- HDFS design and architecture
- Fault tolerance in HDFS
- Create (Write) a file
- Stream reading
- Structured reading
- Java API
- Command-line interface (HDFS Shell)

HDFS Architecture



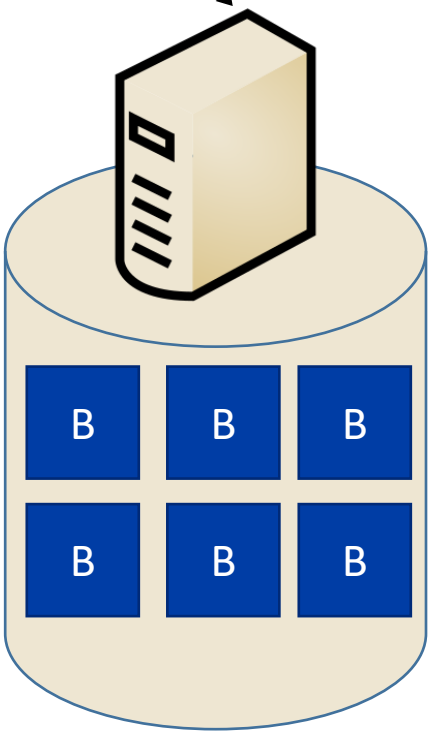
What is where?

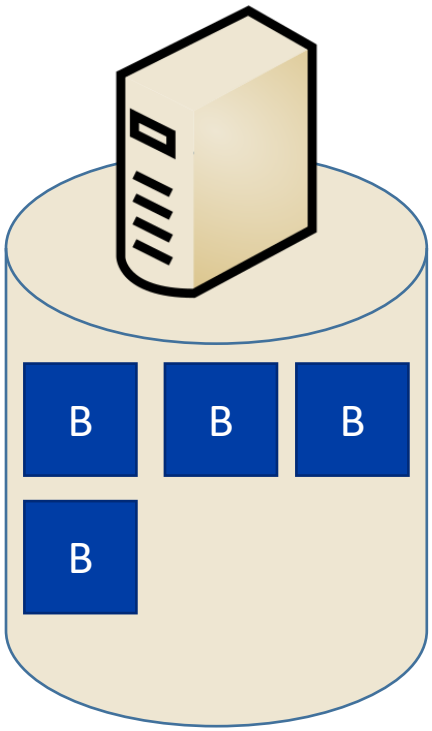
File and directory names
Block ordering and locations
Capacity of data nodes
Architecture of data nodes

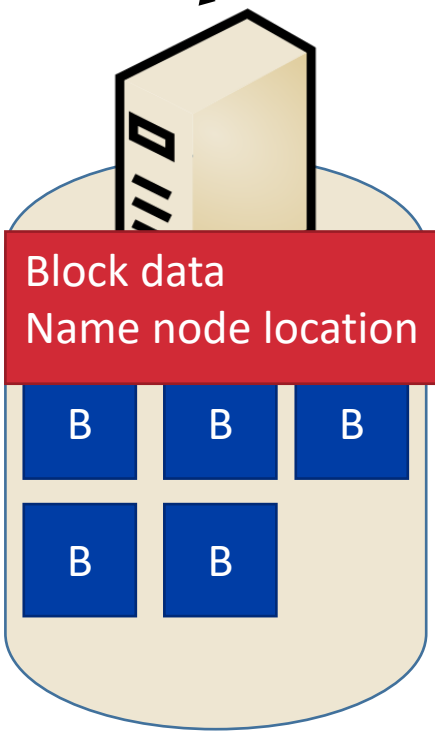


Name node

Data nodes

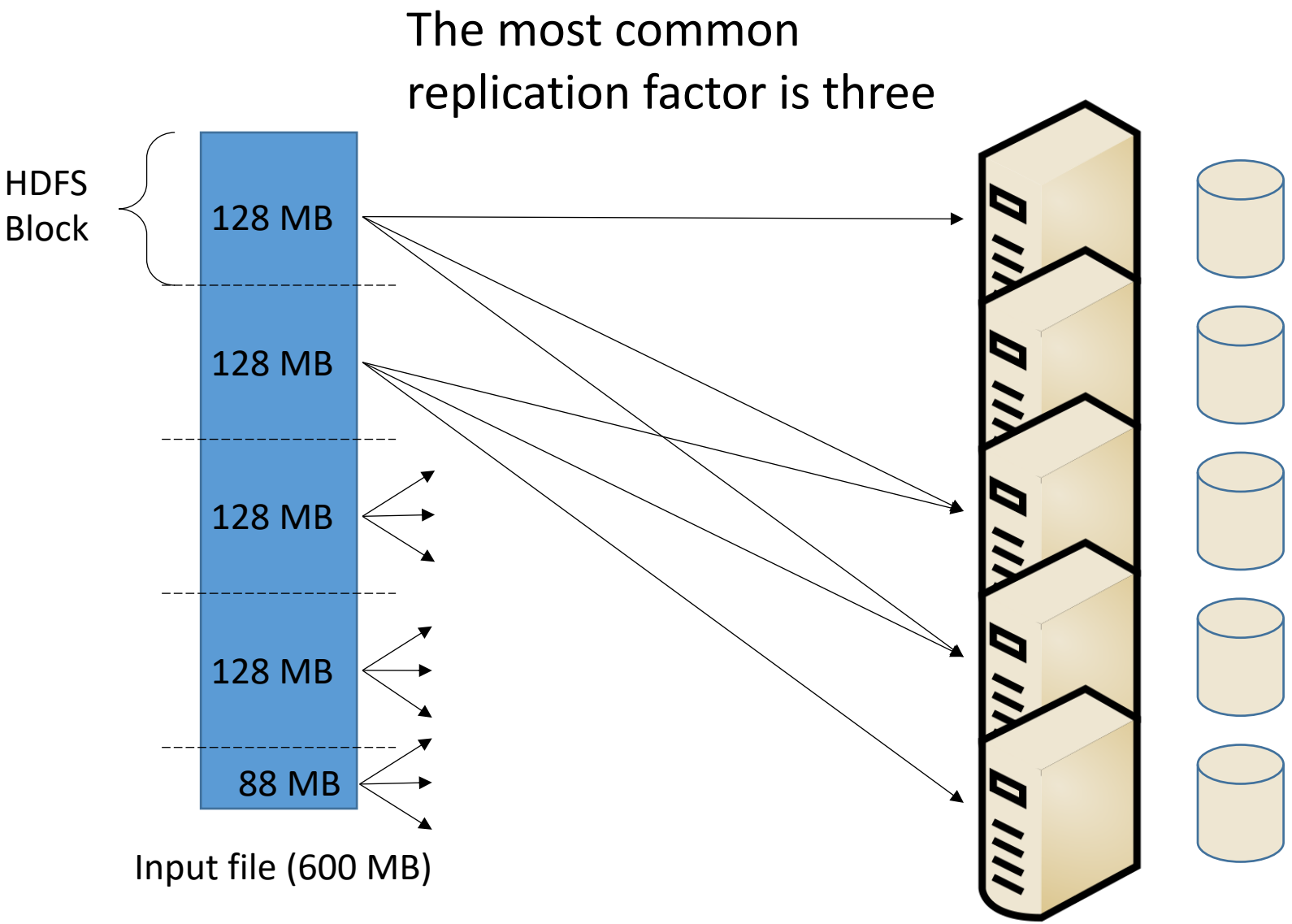




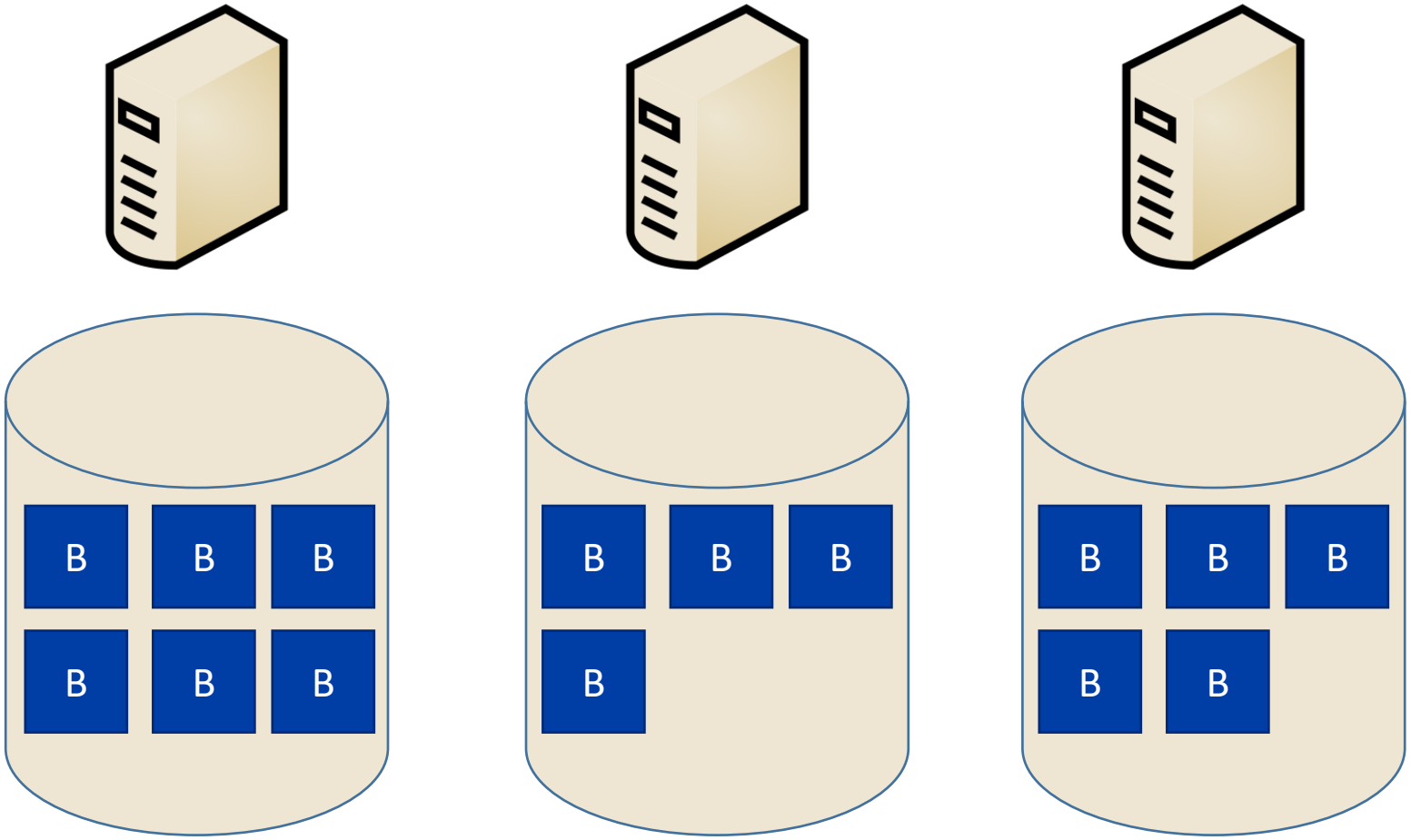


Block data
Name node location

Data Loading

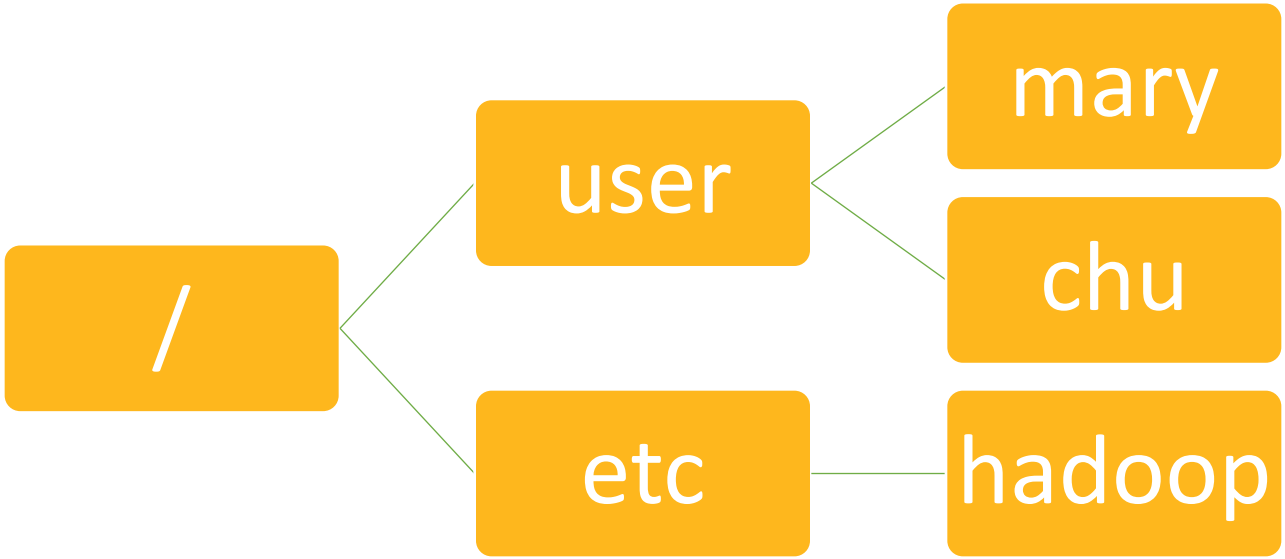


HDFS Storage



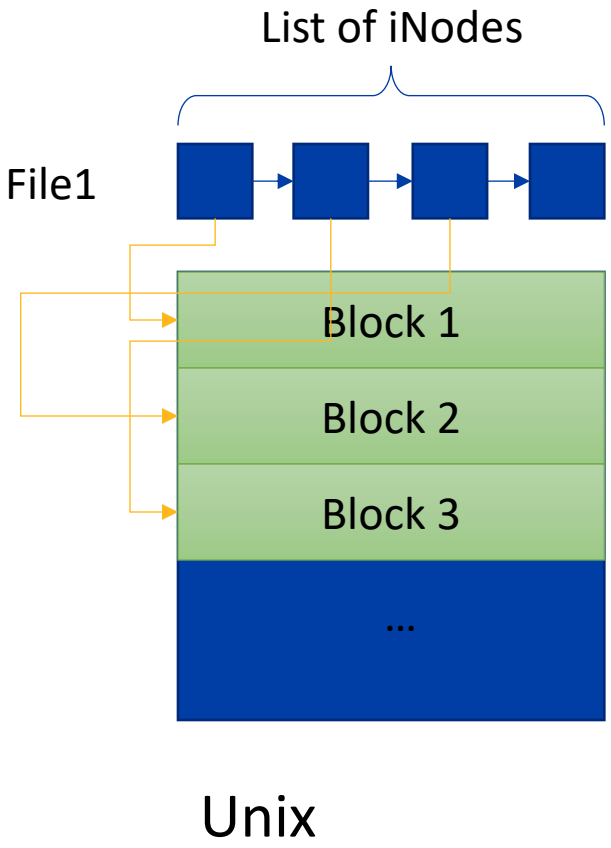
Analogy to Unix FS

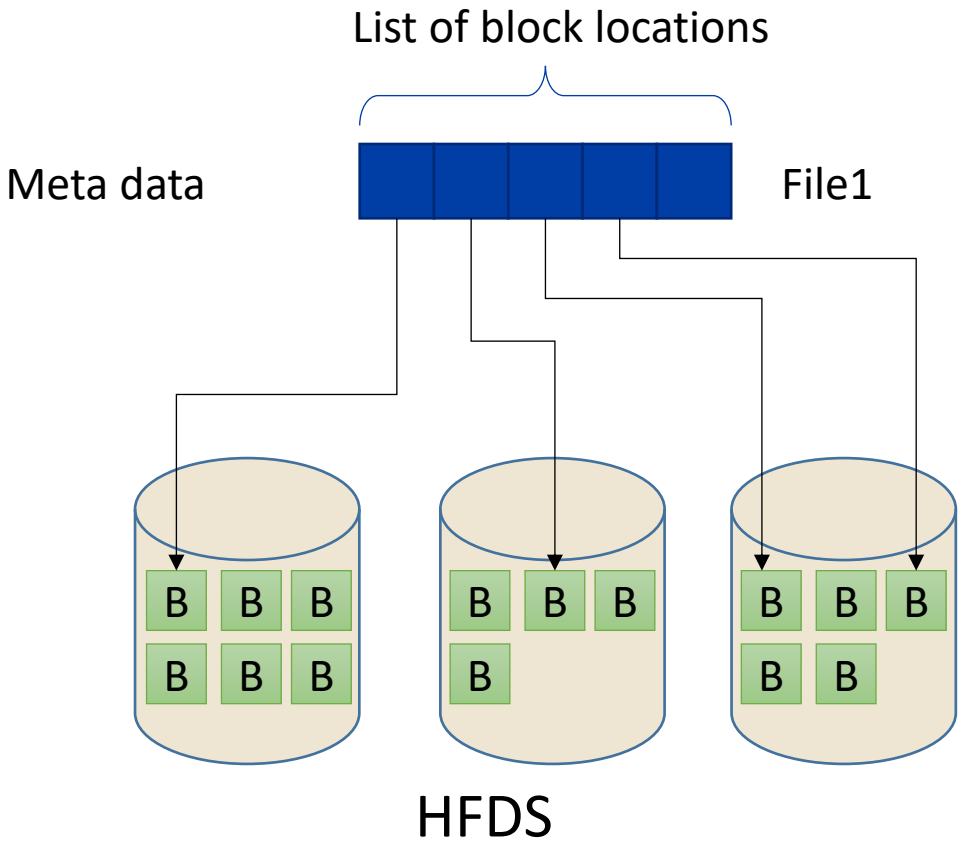
The logical view is similar



Analogy to Unix FS

The physical model is comparable





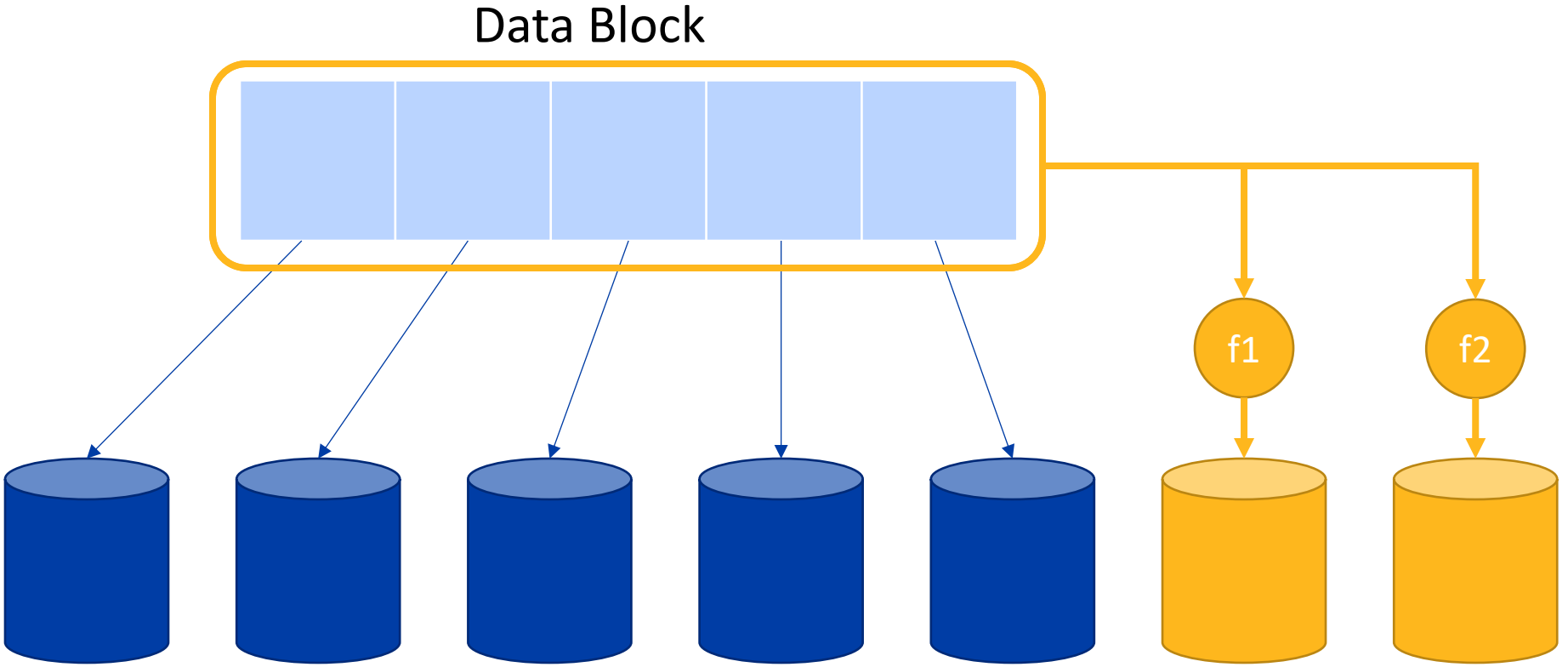
Fault Tolerance in HDFS

Replication

- The default fault tolerance mechanism in HDFS is replication
- The most common replication factor is three
- If one or two nodes are temporarily unavailable, the data is still accessible
- If one or two nodes permanently fail, the master node replicates the under-replicated blocks to reach the desired replication factor
- Drawback: reduced disk capacity

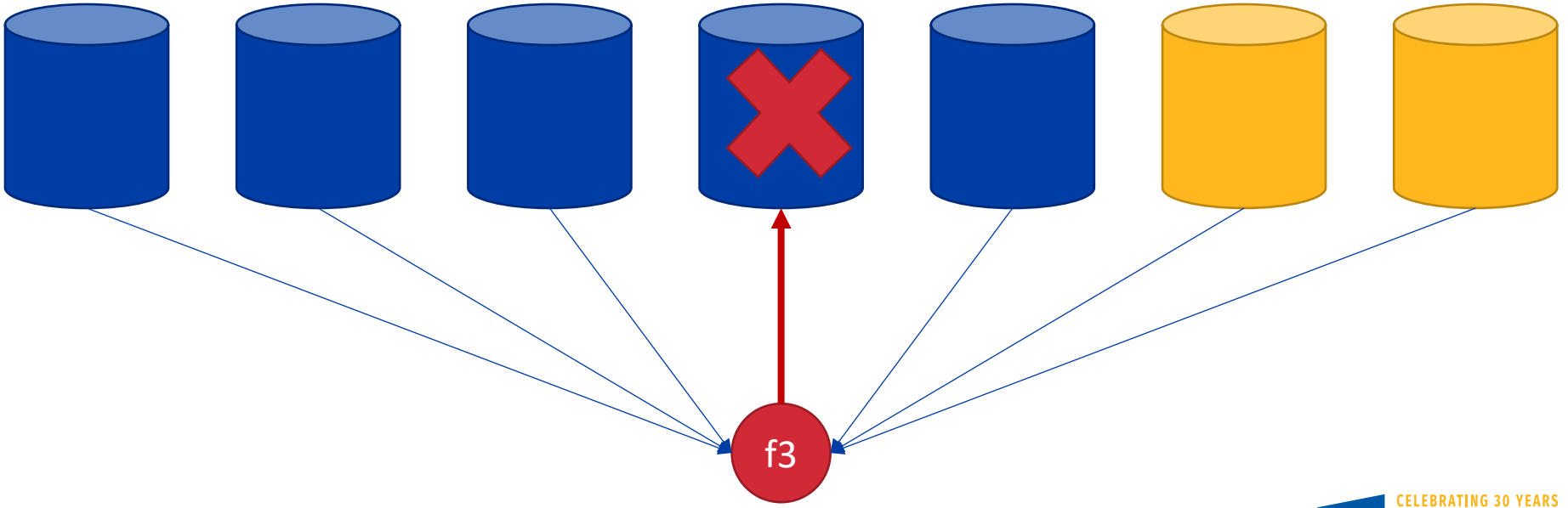
Erasure Coding

- Uses advanced algorithms for recovery, e.g., Reed-Solomon, XOR



Erasure Coding

- Uses advanced algorithms for recovery, e.g., Reed-Solomon, XOR



Overhead

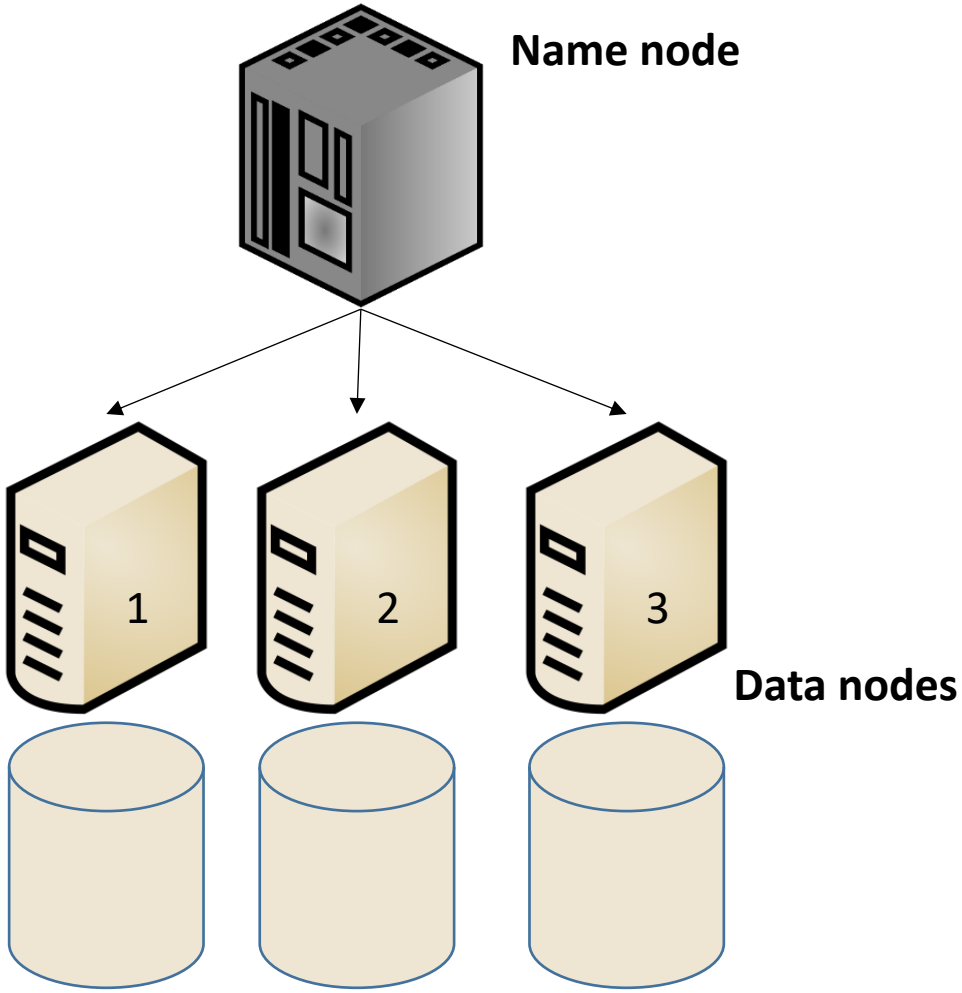
- Three-way replication
- Overhead$= \frac{2}{1}$
$= 200\%$

- Erase coding
- If we use 5+2 scheme, as in the previous example
- Overhead$= \frac{2}{5}$
$= 40\%$

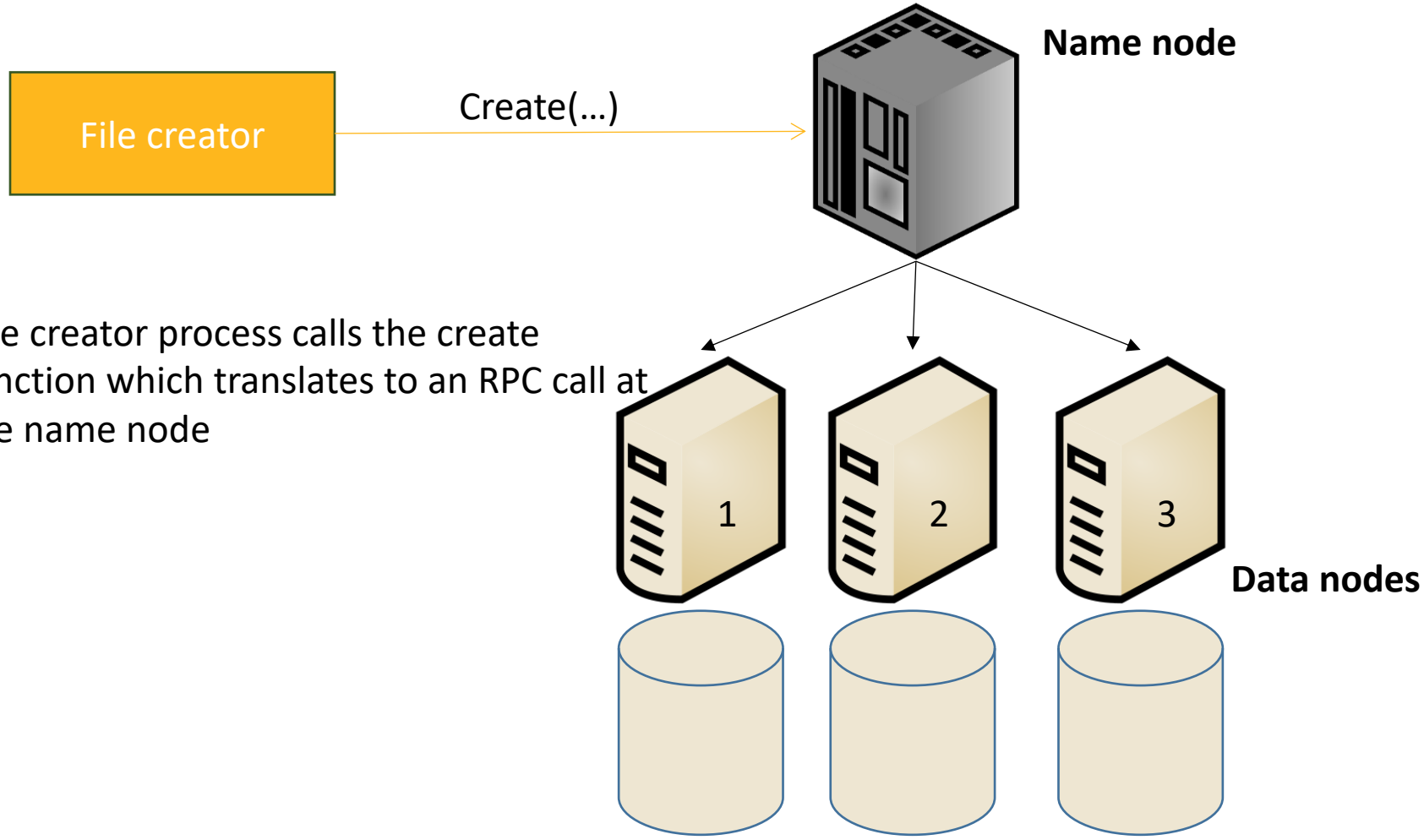
Writing to HDFS

HDFS Create

File creator

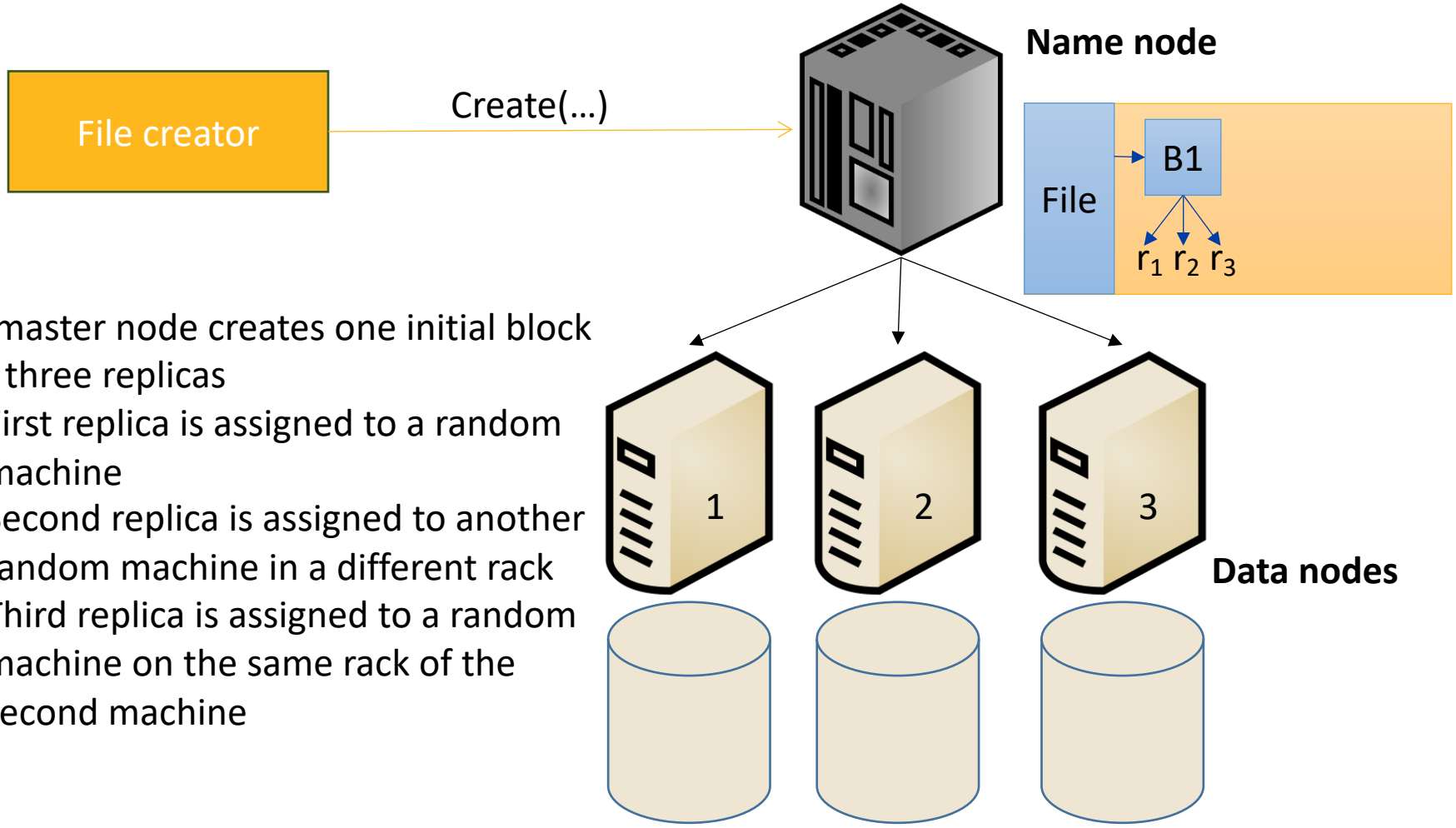


HDFS Create



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

HDFS Create

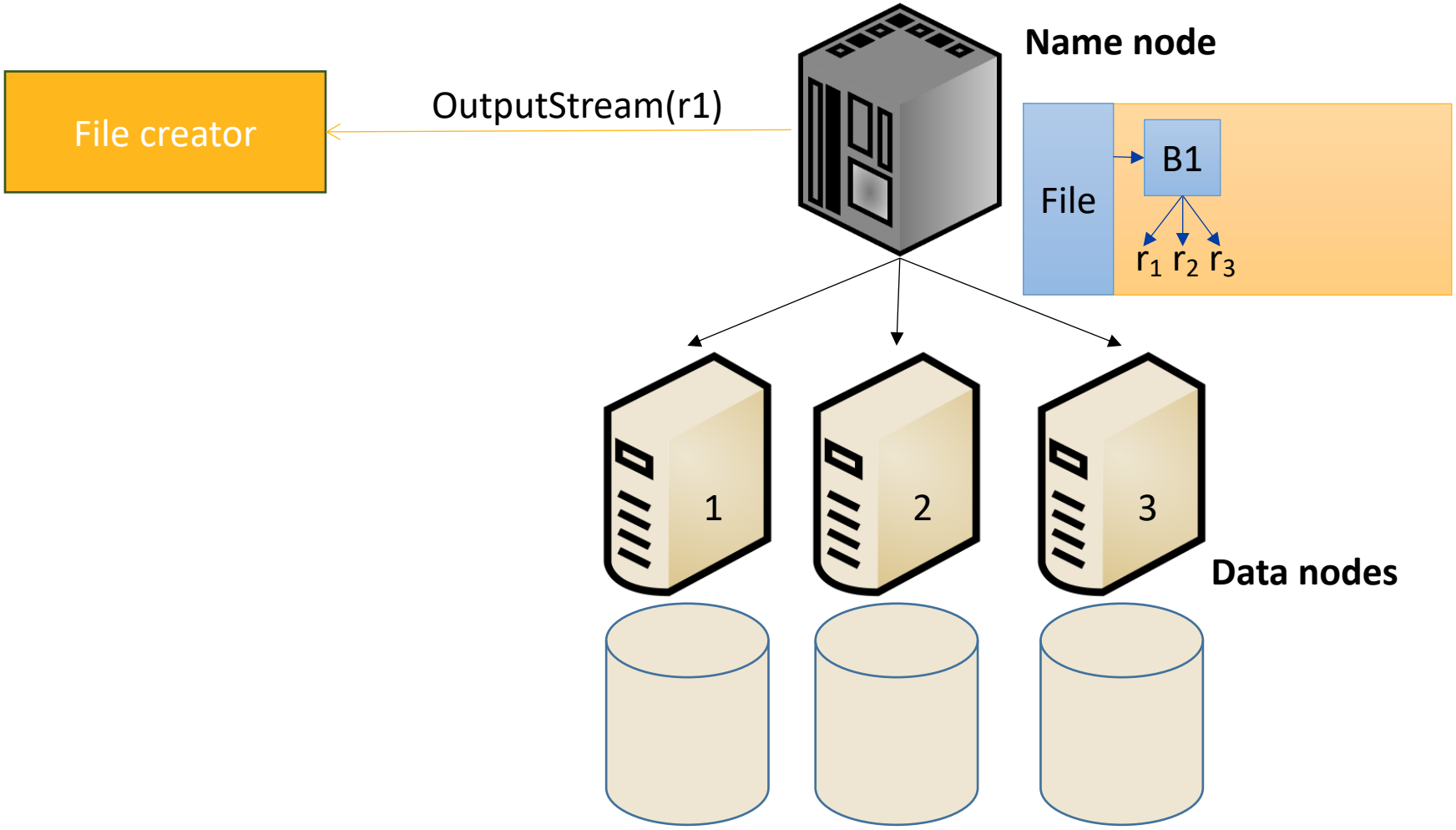


The master node creates one initial block with three replicas

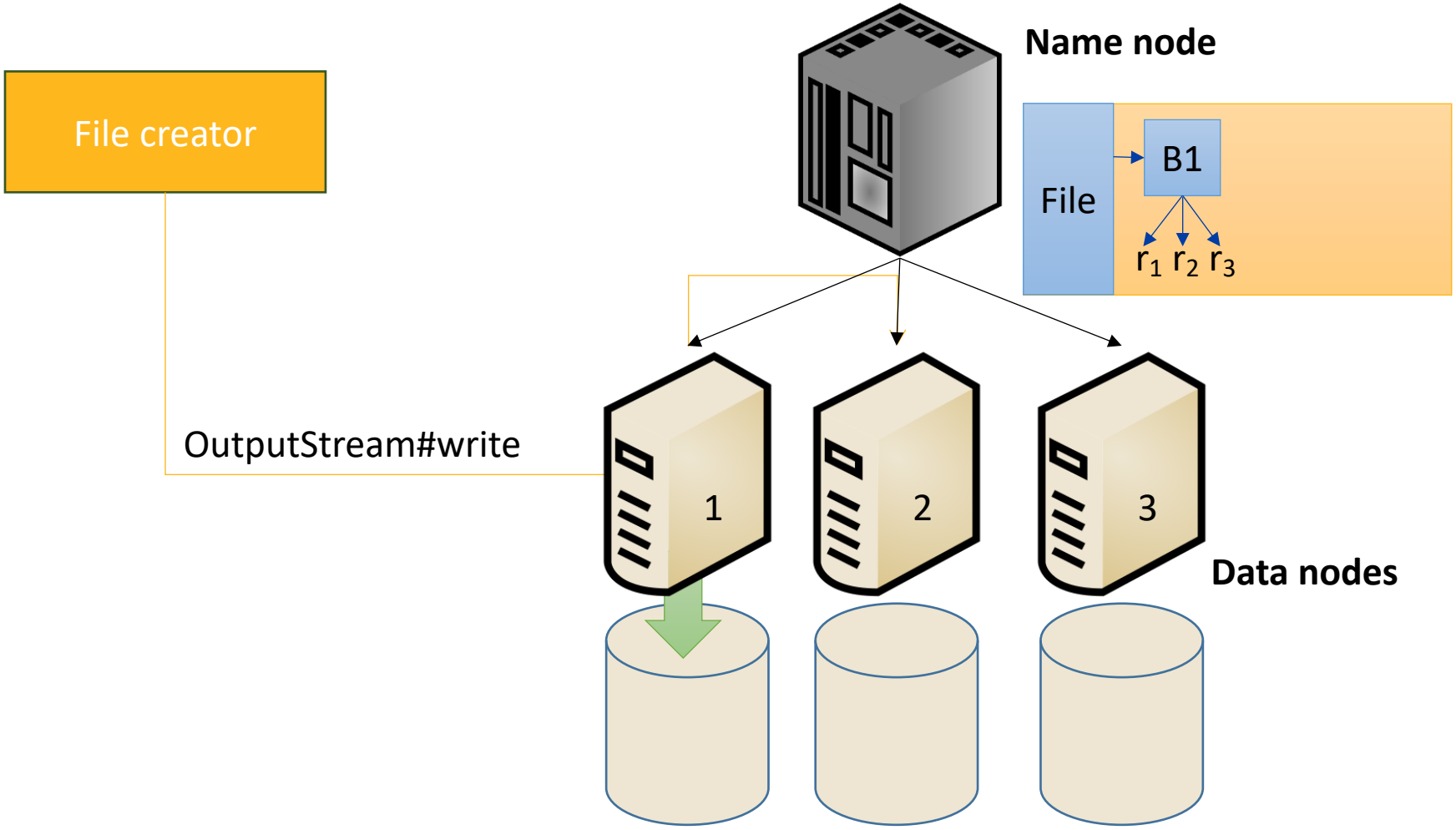
1. First replica is assigned to a random machine
2. Second replica is assigned to another random machine in a different rack
3. Third replica is assigned to a random machine on the same rack of the second machine

Physical Cluster Layout

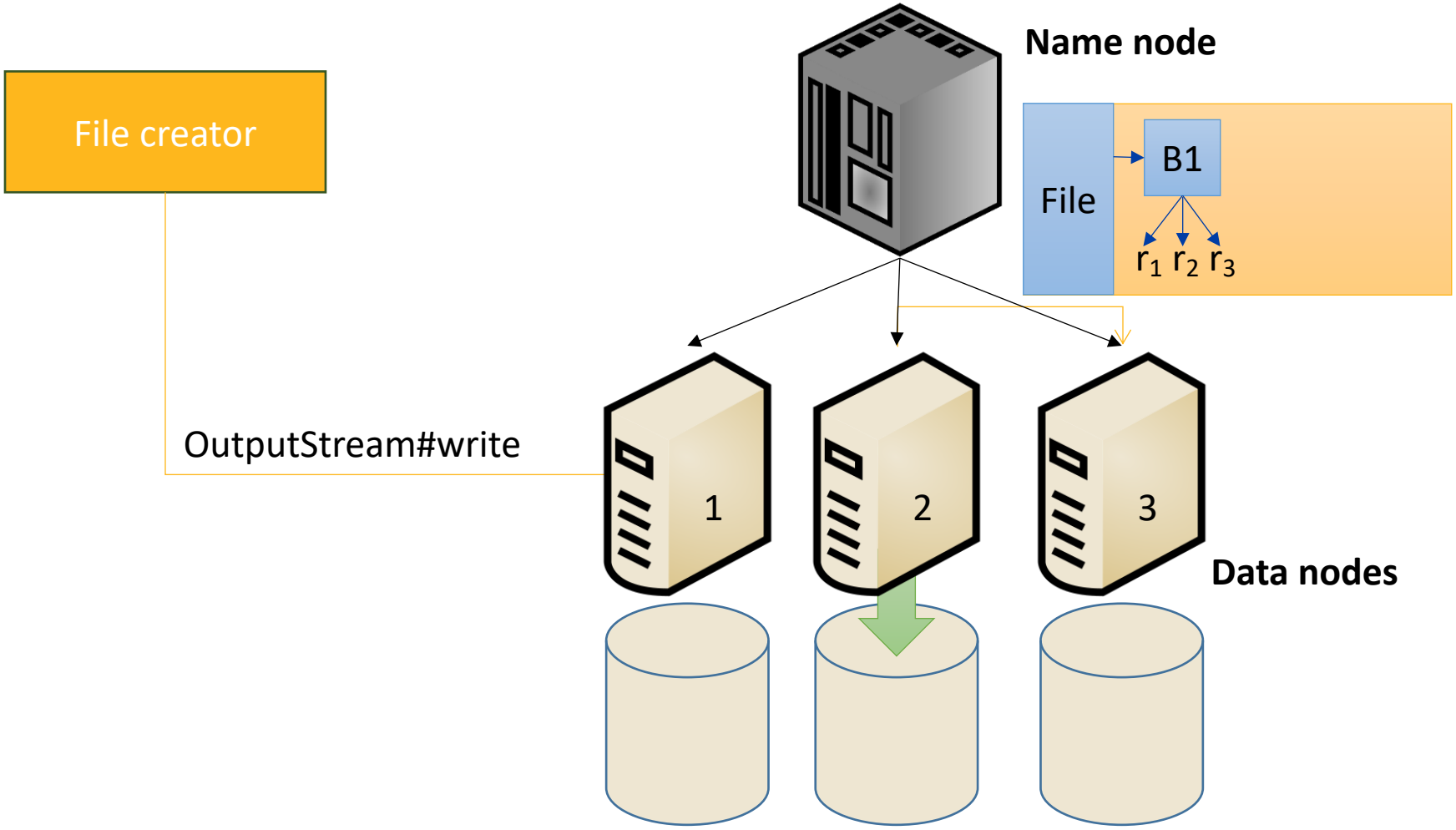
HDFS Create


File creator
OutputStream(r1)
Name node
File
B1
r1 r2 r3
1
2
3
Data nodes

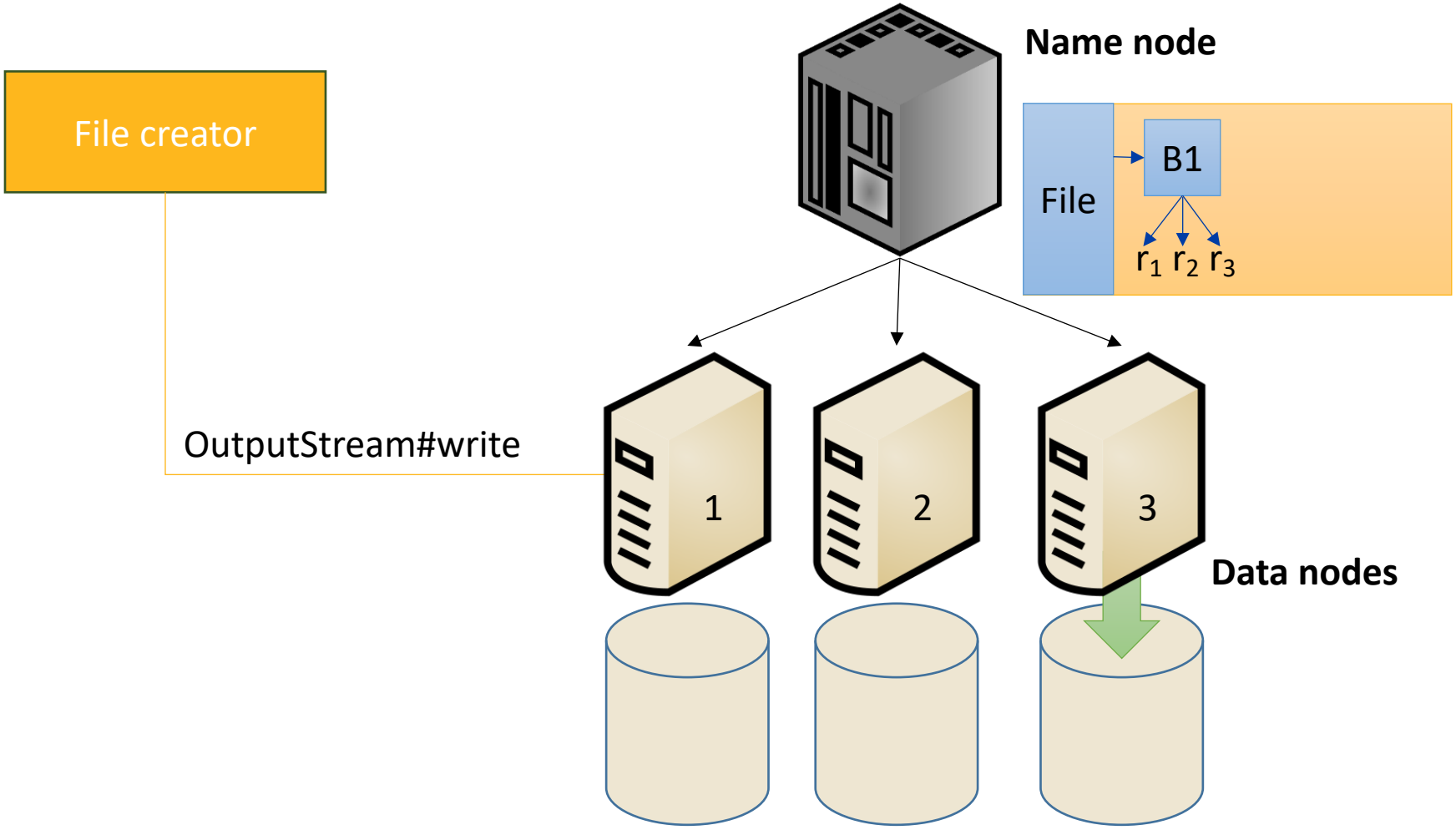
HDFS Create


File creator
OutputStream#write
Name node
File
B1
r1 r2 r3
1
2
3
Data nodes

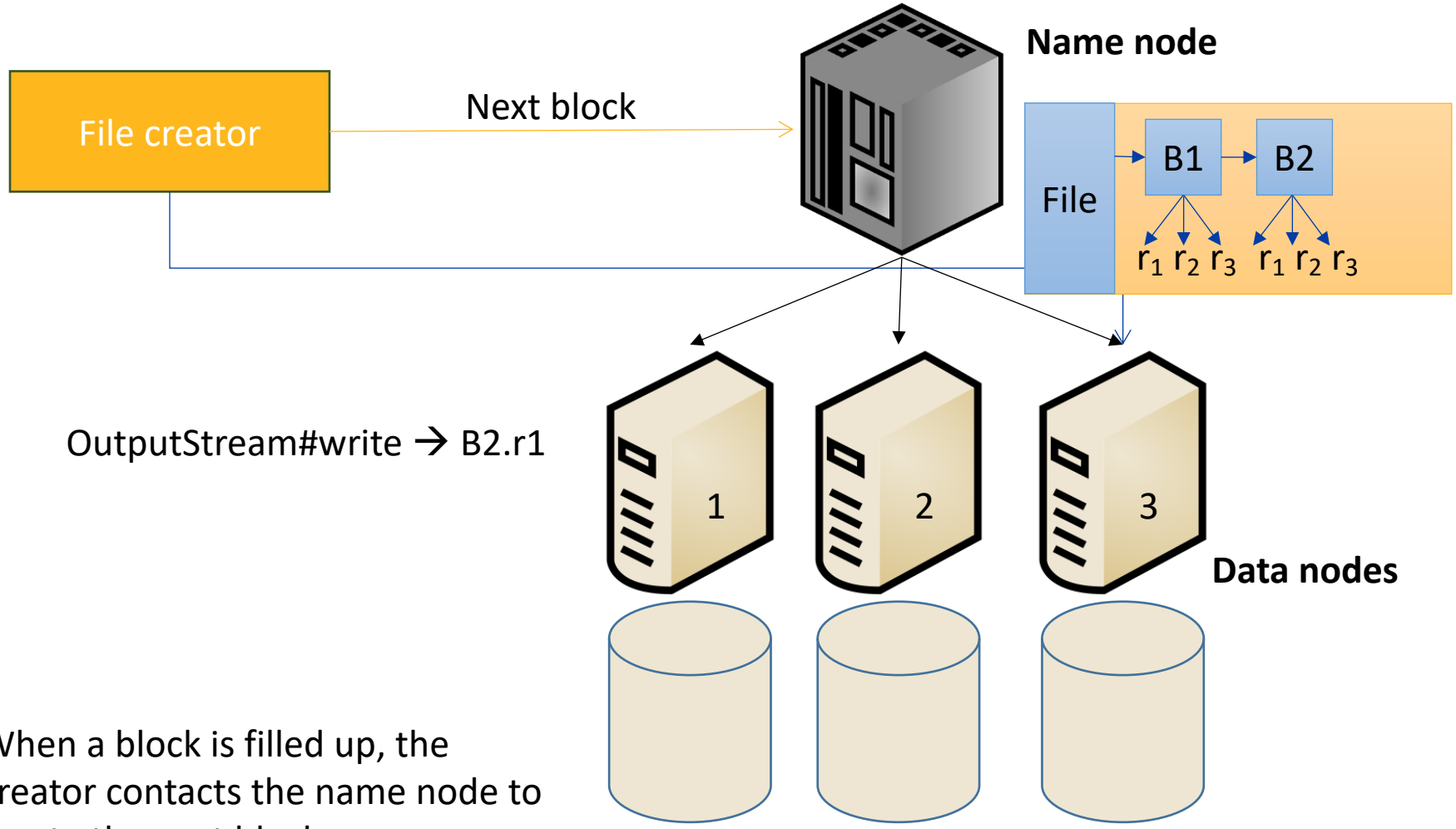
HDFS Create


File creator
Name node
File
B1
r_1 r_2 r_3
OutputStream#write
1
2
3
Data nodes

HDFS Create


File creator
OutputStream#write
Name node
File
B1
r_1 r_2 r_3
1
2
3
Data nodes

HDFS Create



When a block is filled up, the creator contacts the name node to create the next block

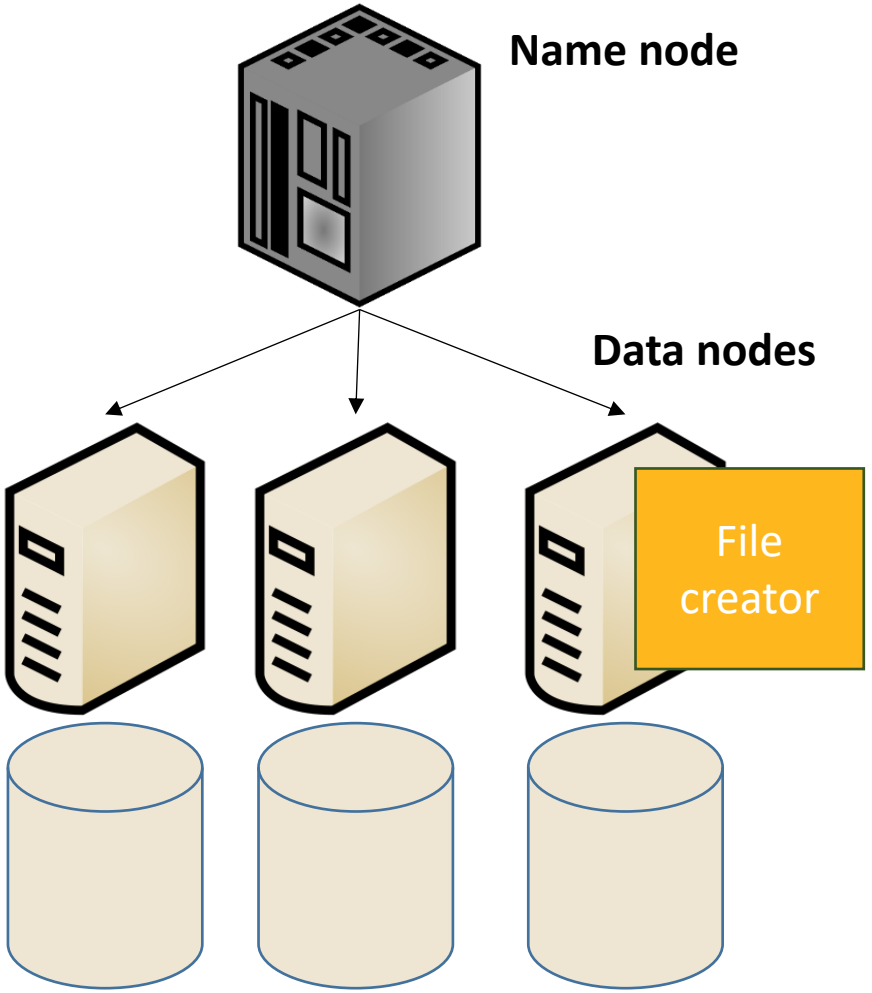
Notes about writing to HDFS

- 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

Self-writing

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, i.e., the second replica on a different rack and the third replica on the same rack as the second one.

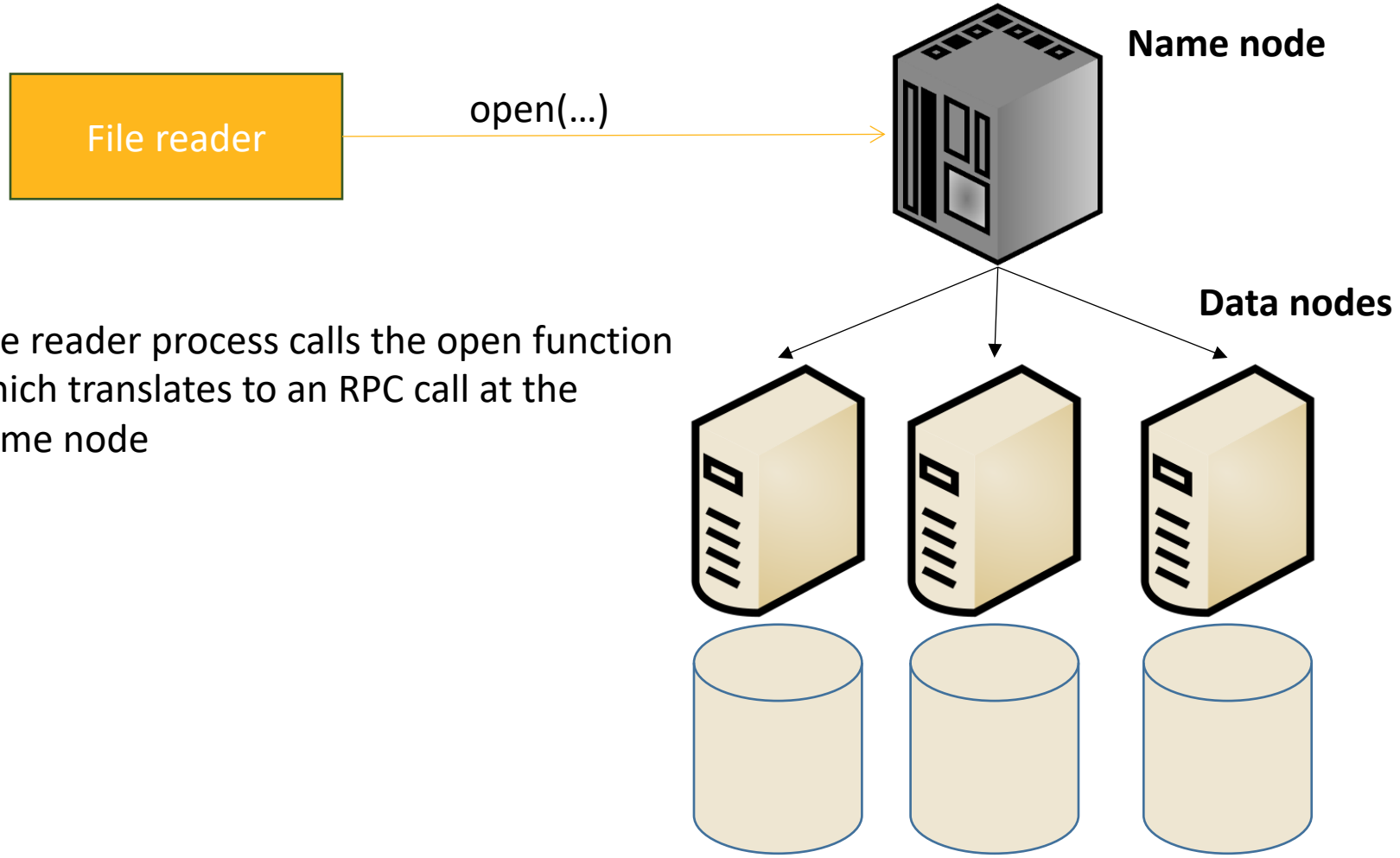


Stream reading from HDFS

Reading from HDFS

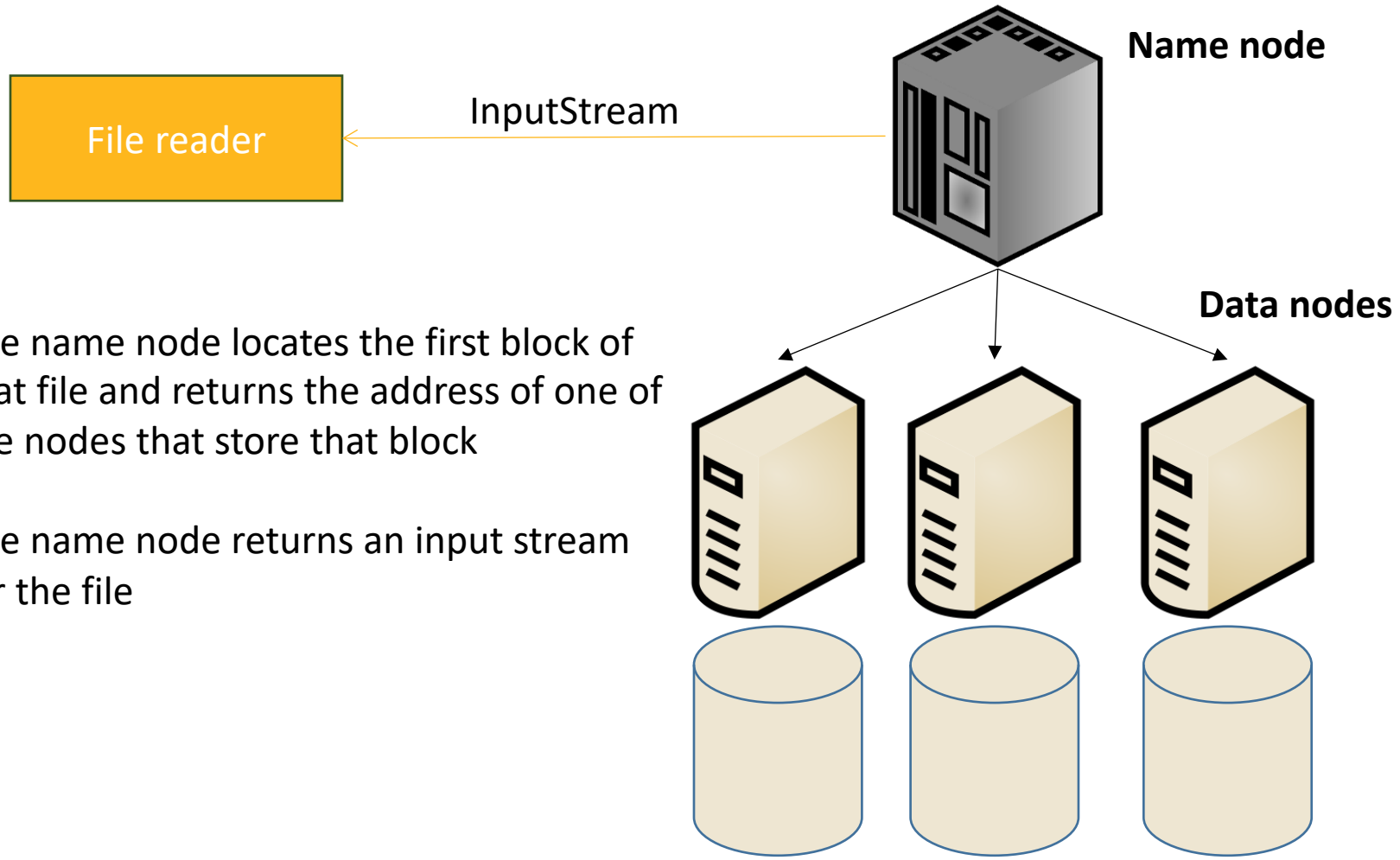
- Reading is relatively easier
- No replication is needed
- Replication can be exploited
- Random reading *is* allowed

HDFS Read



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

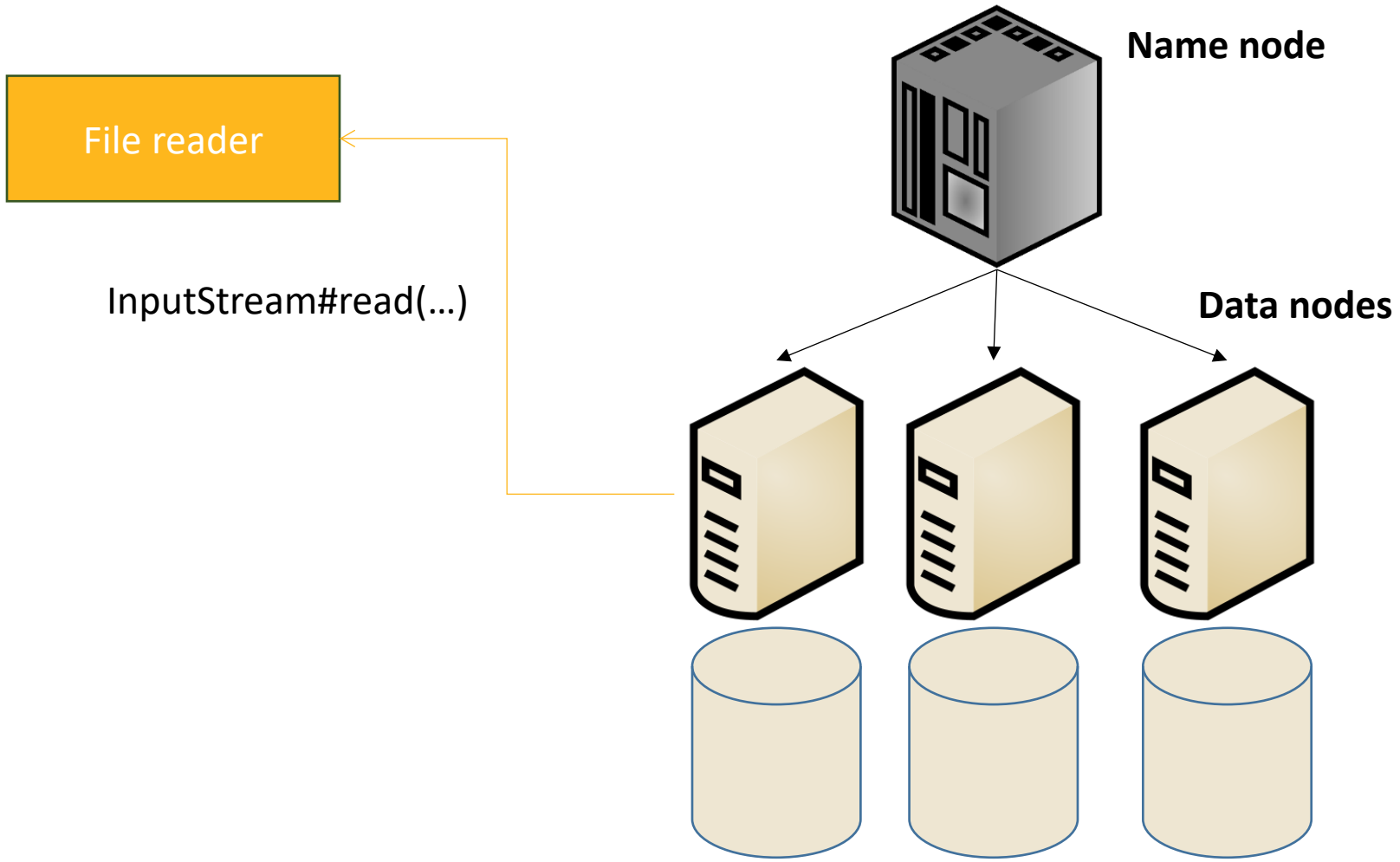
HDFS Read



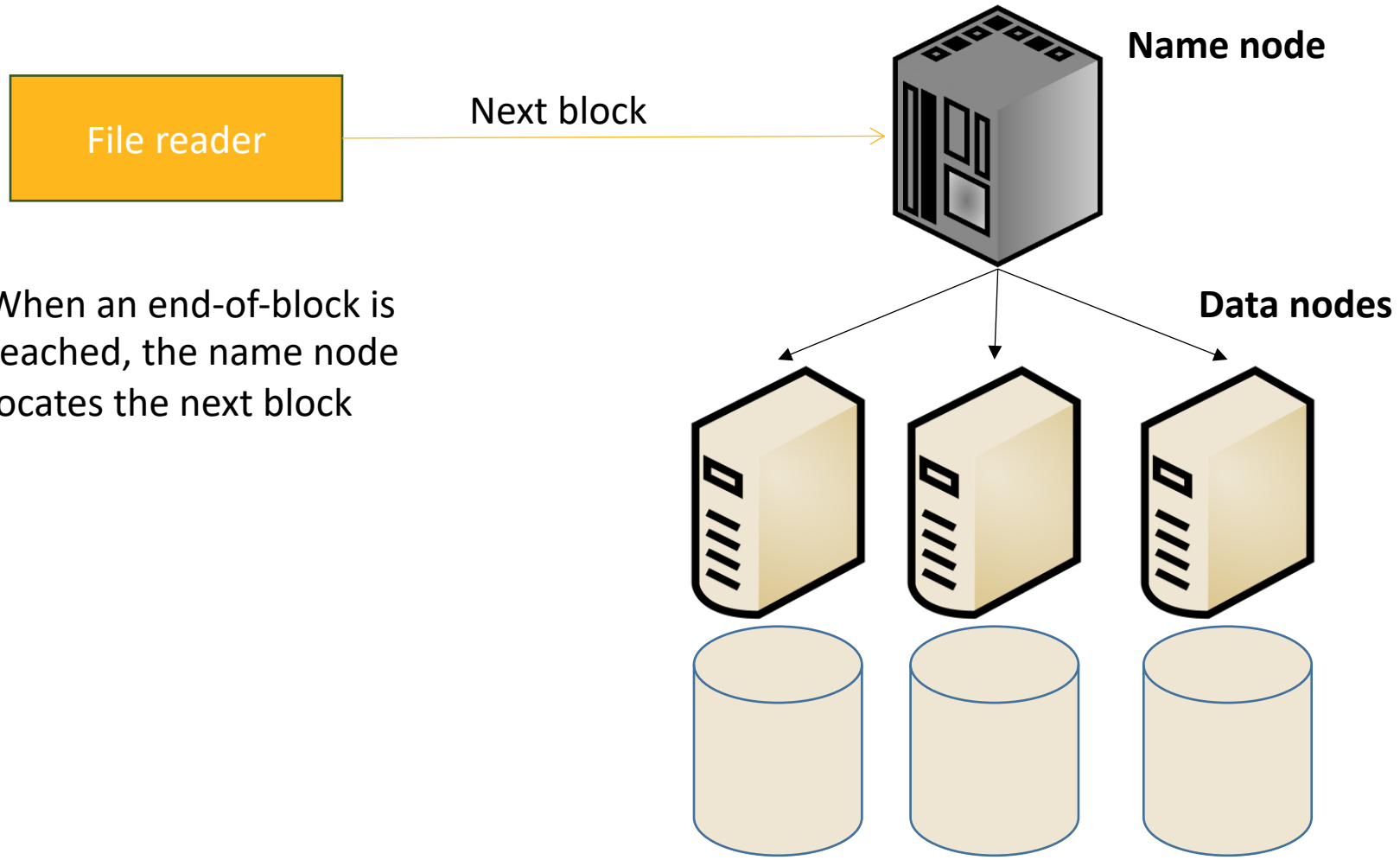
The name node locates the first block of that file and returns the address of one of the nodes that store that block

The name node returns an input stream for the file

HDFS Read

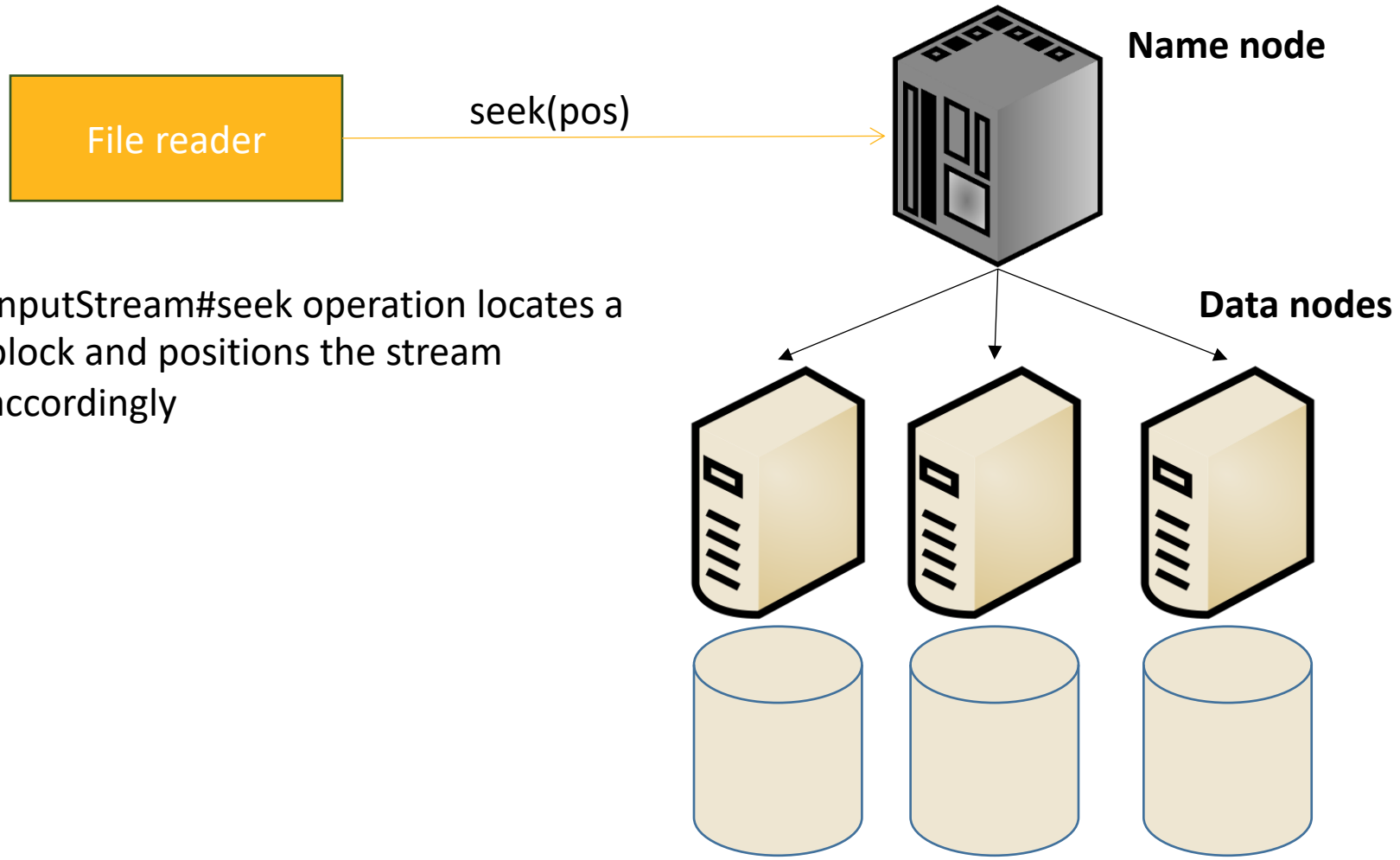


HDFS Read



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

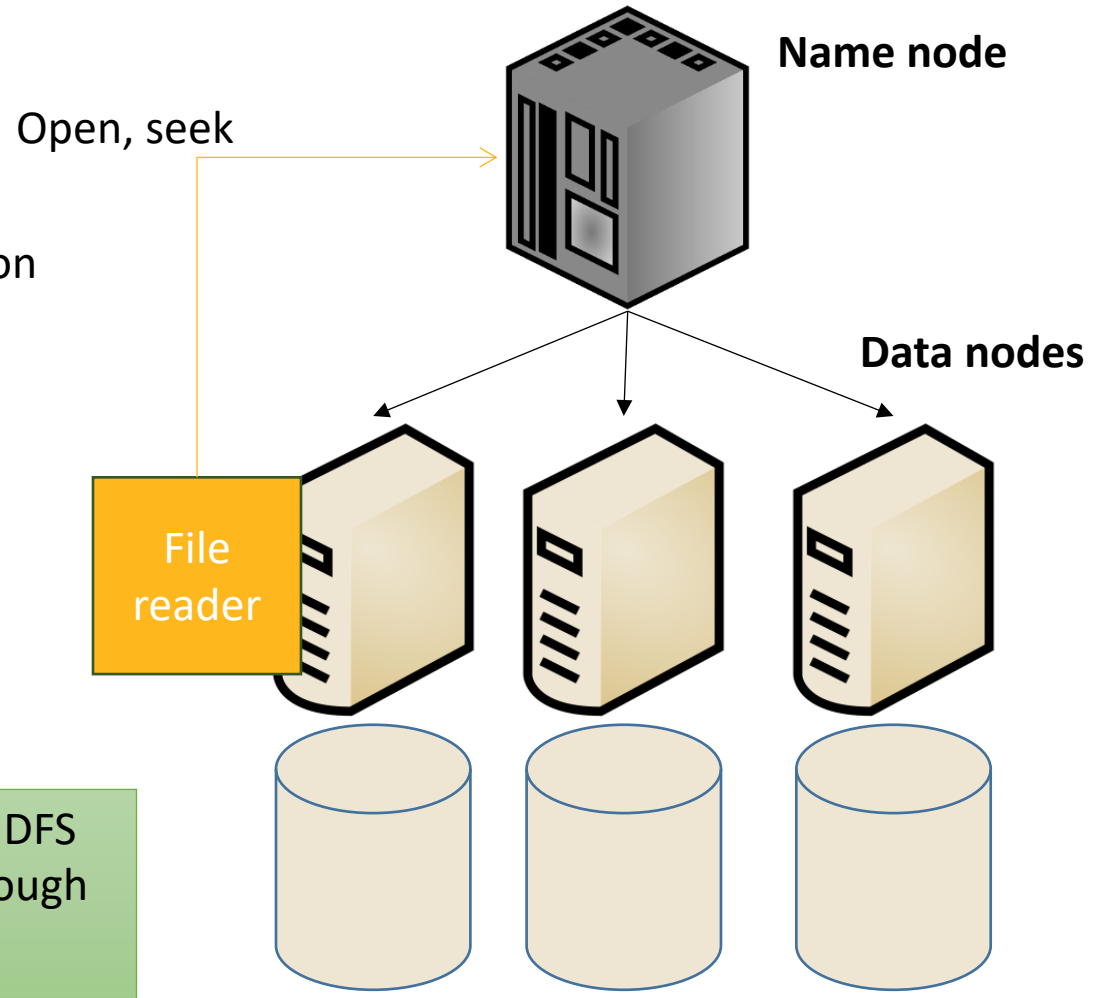
HDFS Read



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

Self-reading

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 is chosen



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

Notes About Reading

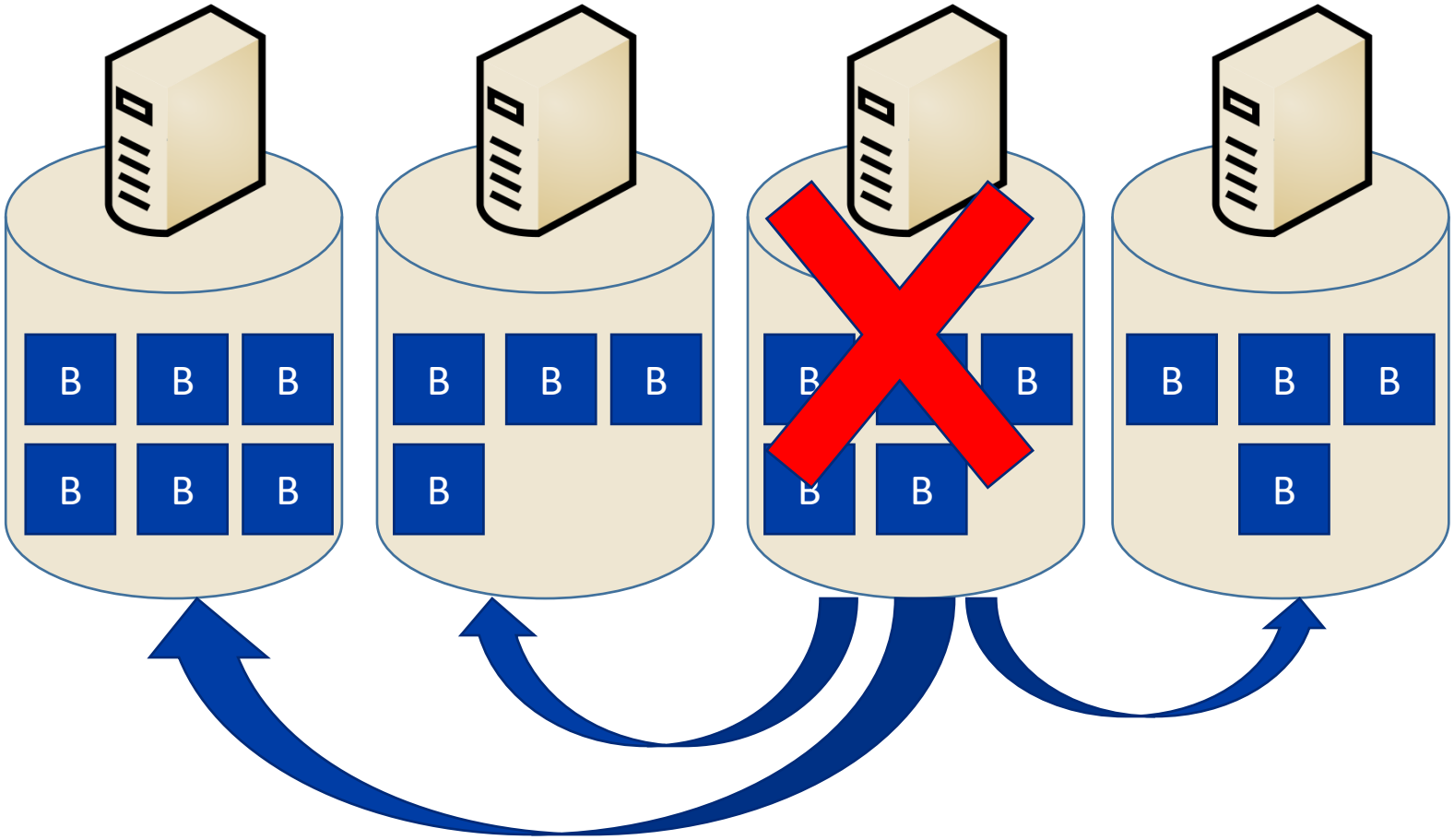
- The API is much richer than the simple open/seek/close API
  - You can retrieve block locations
  - You can choose a specific replica to read
- The same API is generalized to other file systems including the local FS and S3
- Review question: Compare random access read in local file systems to HDFS

Special Features in HDFS

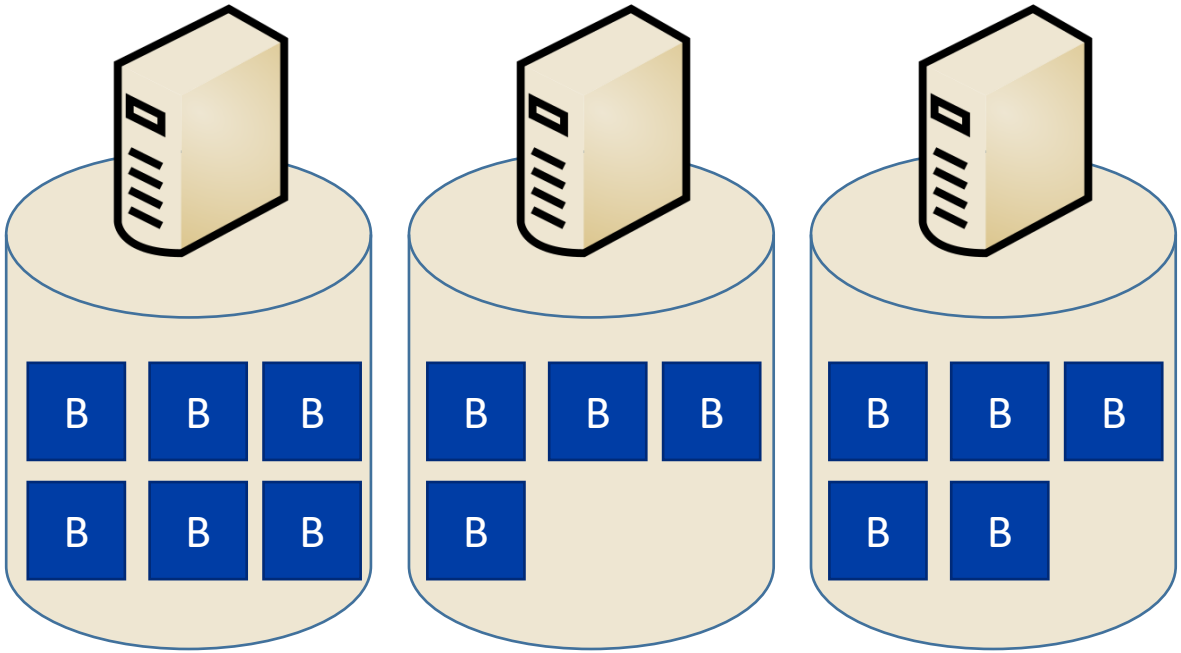
HDFS Special Features

- Node decomission
- Load balancer
- Cheap concatenation

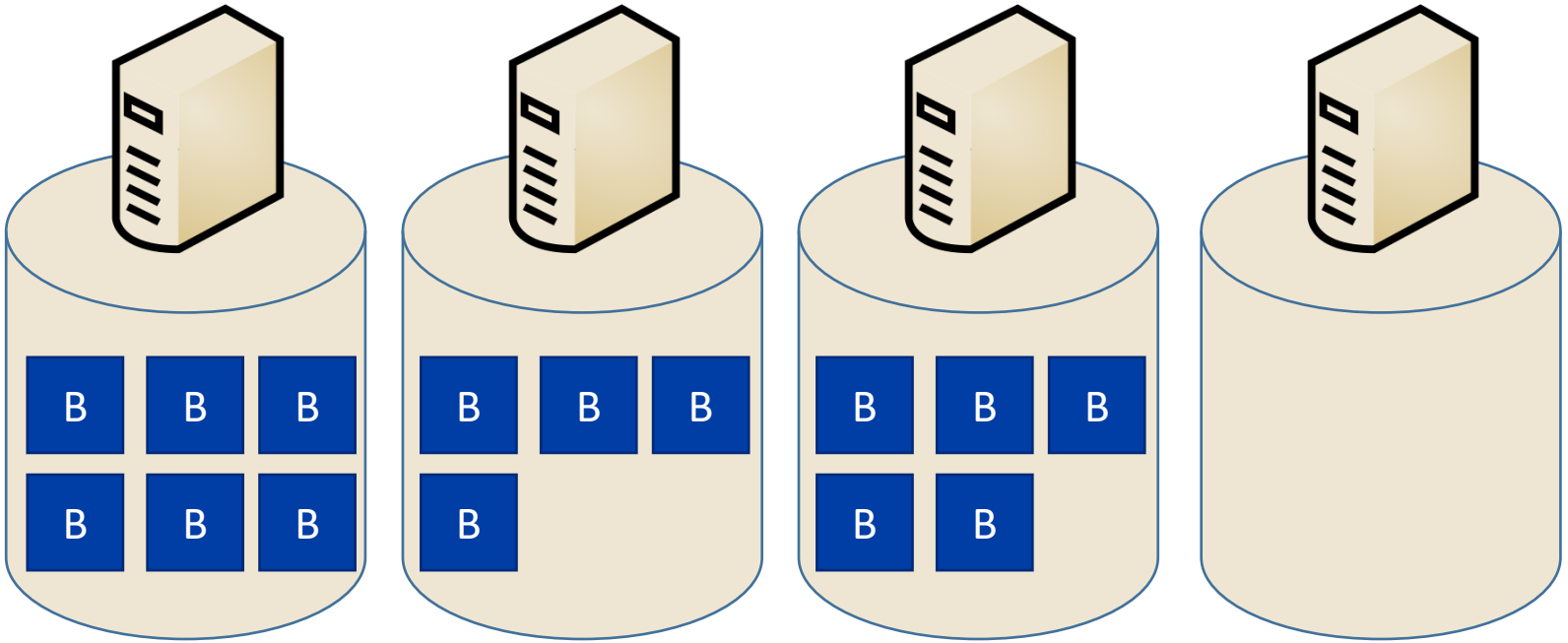
Node Decommission



Load Balancing

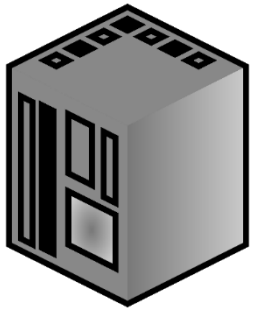


Load Balancing

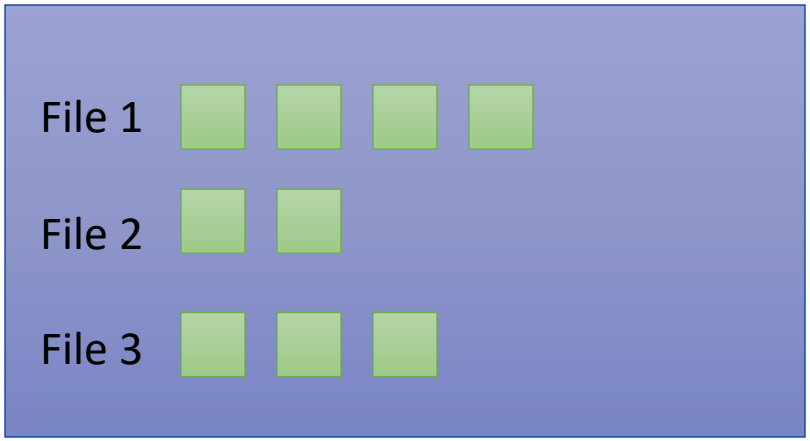


Start the load balancer

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.

HDFS Shell

Command-line Interface (CLI)

HDFS Shell

- Used for common operations
- Its usage is similar to Unix shell commands
- Basic operations include
  - ls, cp, mv, mkdir, rm, …
- HDFS-specific operations include
  - copyToLocal, copyFromLocal, setrep, appendToFile, …

HDFS Shell

- General format

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

- So, instead of

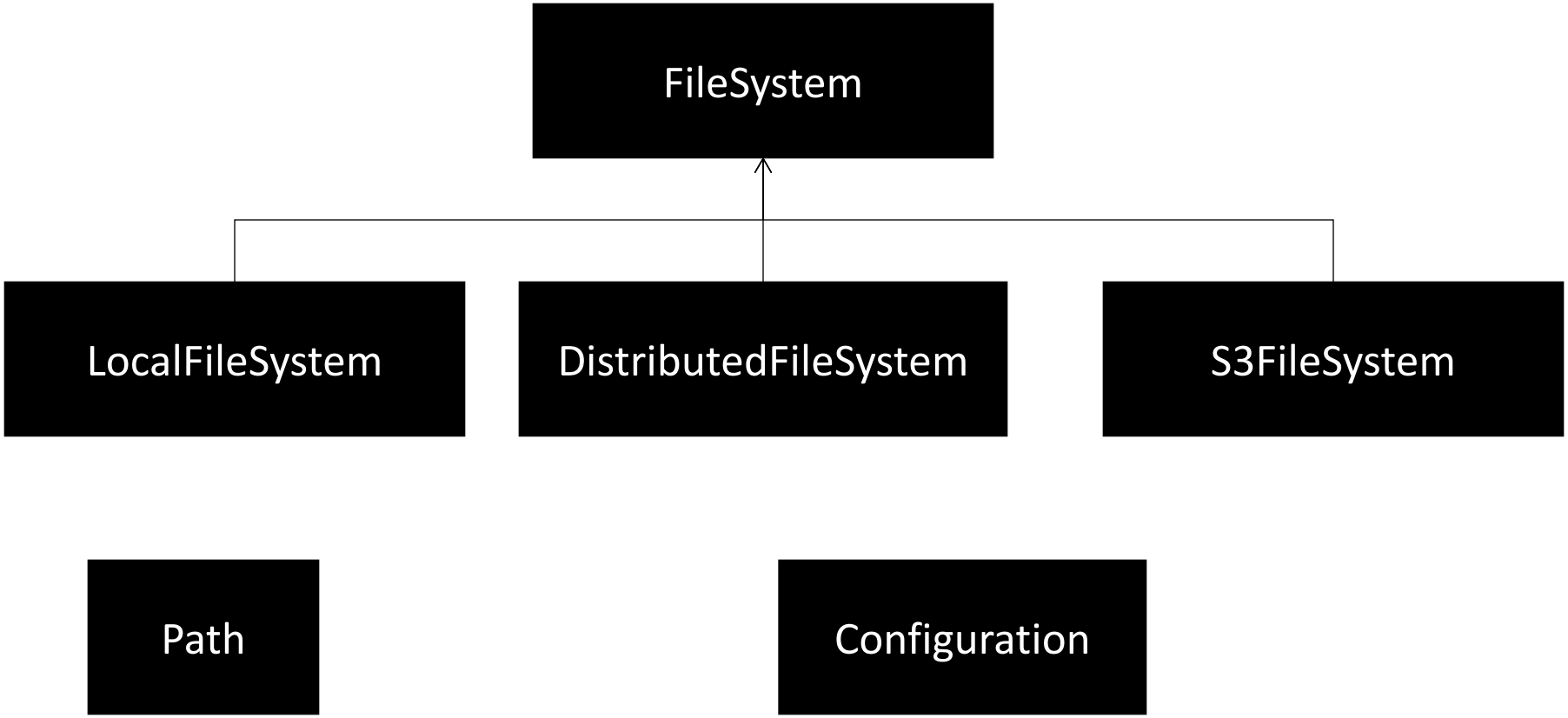
```
mkdir –p myproject/mydir
```

- You will write

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

- A list of shell commands with usage
  - https://hadoop.apache.org/docs/r3.3.0/hadoop-project-dist/hadoop-common/FileSystemShell.html

HDFS API


FileSystem
LocalFileSystem
DistributedFileSystem
S3FileSystem
Path
Configuration

HDFS API Classes

- Configuration: Holds system configuration such as where the master node is running and default system parameters, e.g., replication factor and block size
- Path: Stores a path to a file or directory
- FileSystem: An abstract class for file system operations

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 (Java)

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);
```

Rename a file

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

HDFS API

Open a file

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

Seek to a different location

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

HDFS API

Concatenate

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

Get file metadata

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

Get block locations

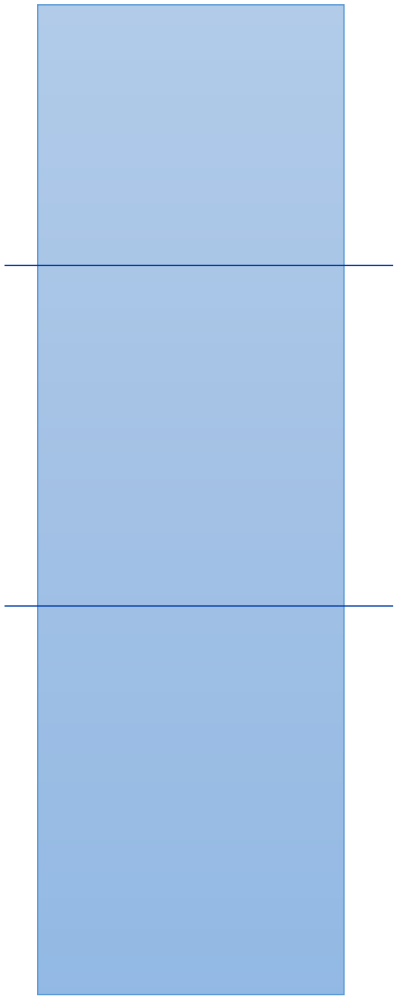
```
fs.getFileBlockLocations(path, from, to);
```

Structured Reading

Data files

- In distributed big data processing, input files contain records not just raw bytes
- We need to a way to read records from files in HDFS
- For efficiency, we should split the file and read it in parallel

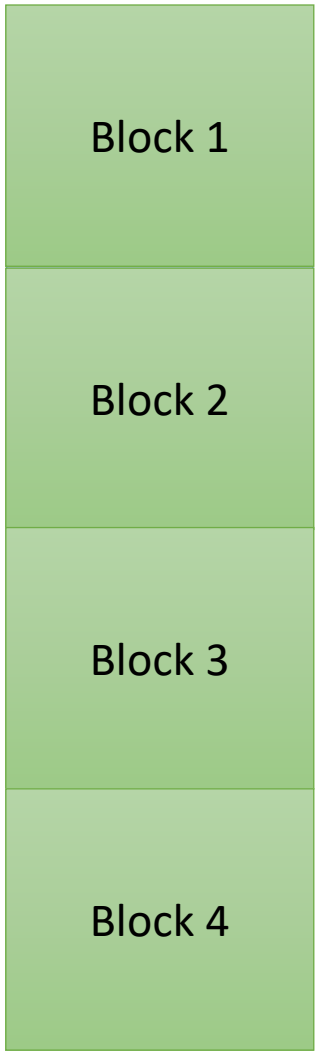
File Splitting



Input File

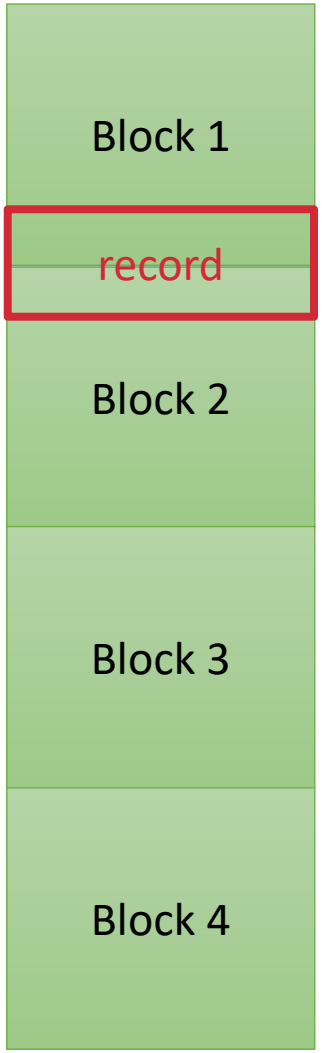
- How to split the file?
  - By record: For fixed-size records
  - By size: For variable-size records
- Considerations, splitting the file should be fast
- Should not need to read the entire file to split it

Default File Splitting



- A split is created for each block
- Advantages
  - Data locality
  - Efficiency
- If the file is too small, a single block might be further split

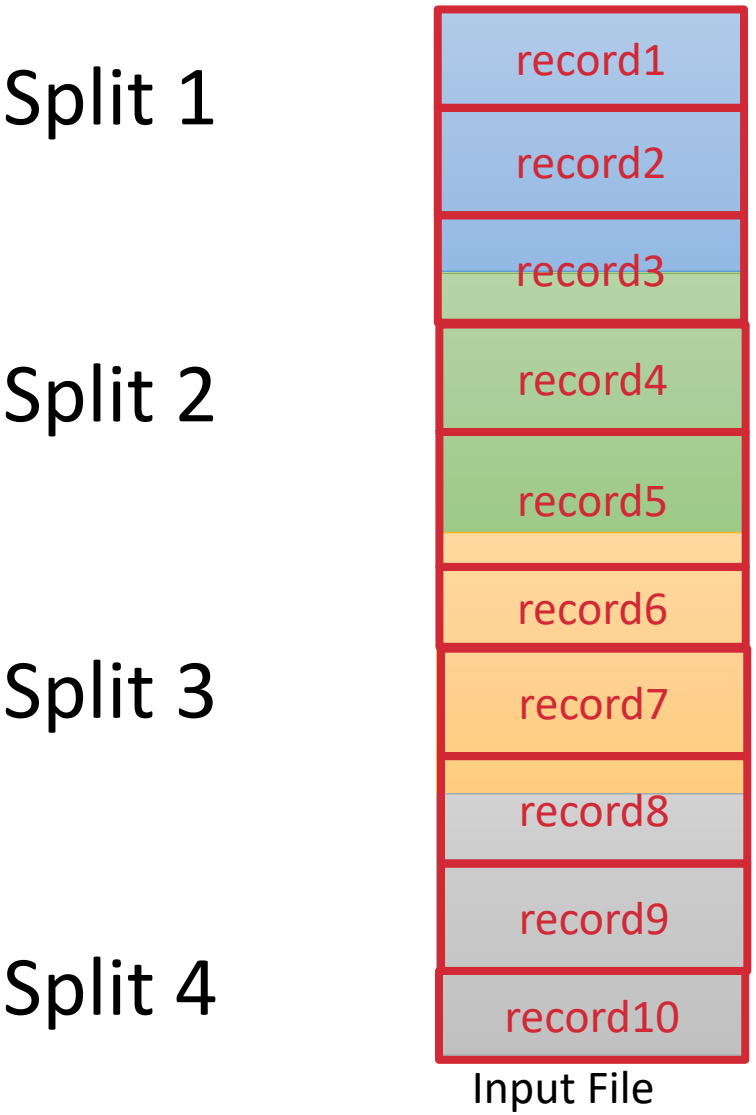
Read data in one split



Input File

- Read all the records that are inside the split
- How to deal with records that overlap two splits?
- In each split, we should read the records that **start** in that split

Read data in every split



- Which records will be read for each of the four splits?

Reading process

- Split the file based on the file metadata
  - File size, block sizes, # of nodes
- Each split is defined by:
  - File name, Start offset, Length
- For each split:
  - Seek to the start offset
  - Skip the first record (except for the first split)
  - Read until the beginning of the record goes beyond the start + length

Conclusion

- HDFS is a general-purpose distributed file system
- It provides a write-once ready-many access interface
- Supports random reading which can be used for stream reading and structured reading
- Provides a Unix-like shell
- Provides a Java API for programming