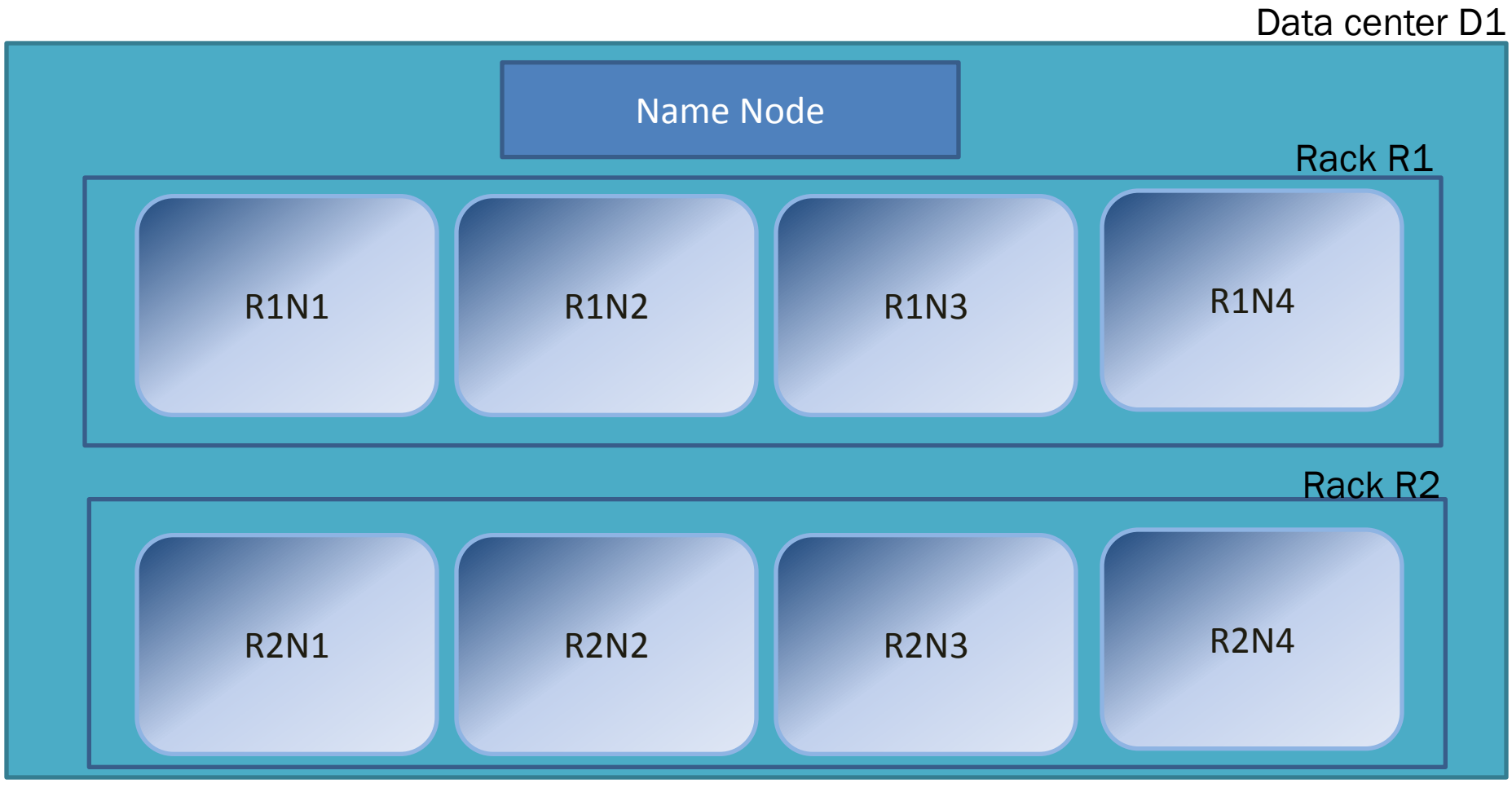
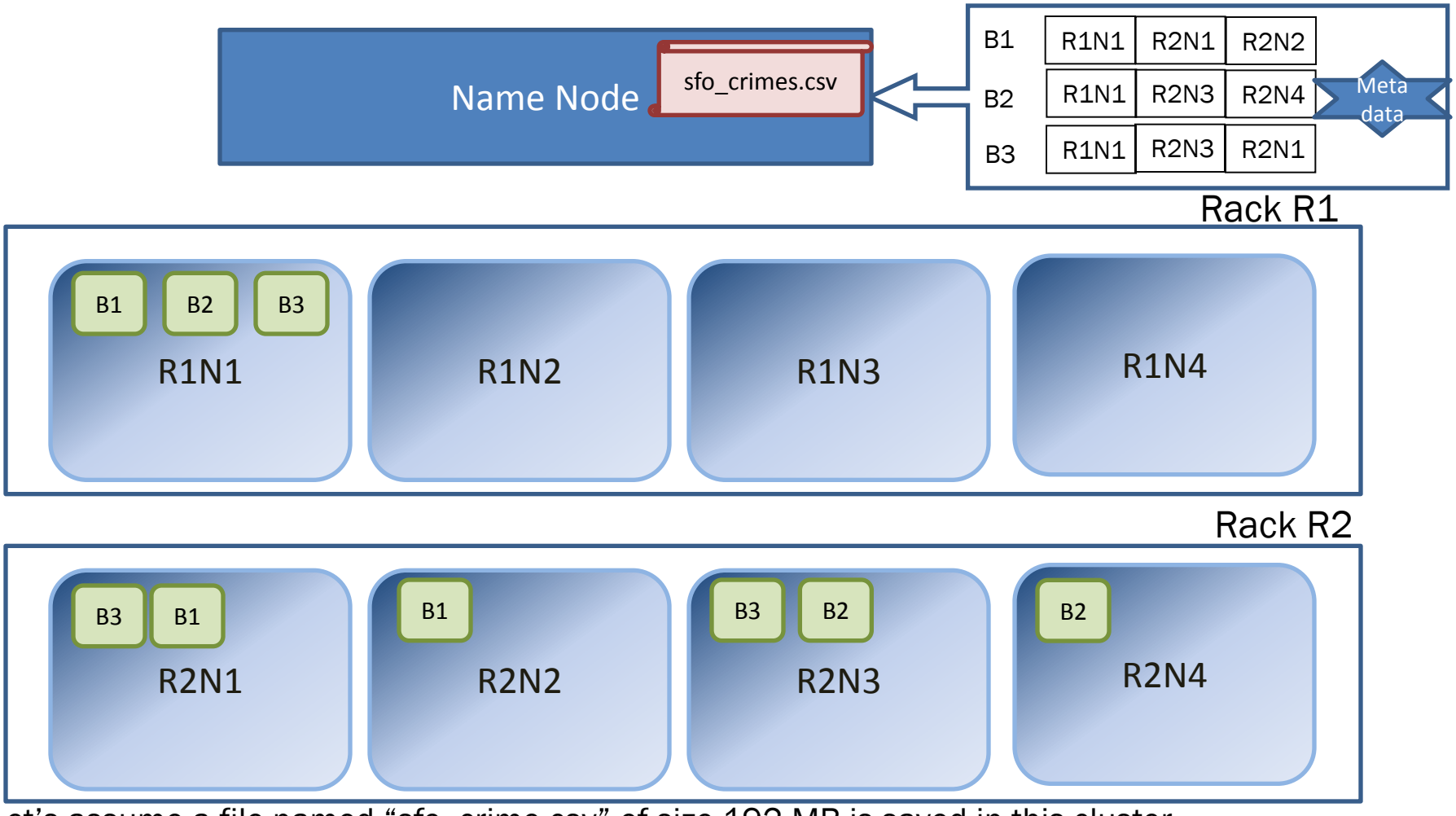
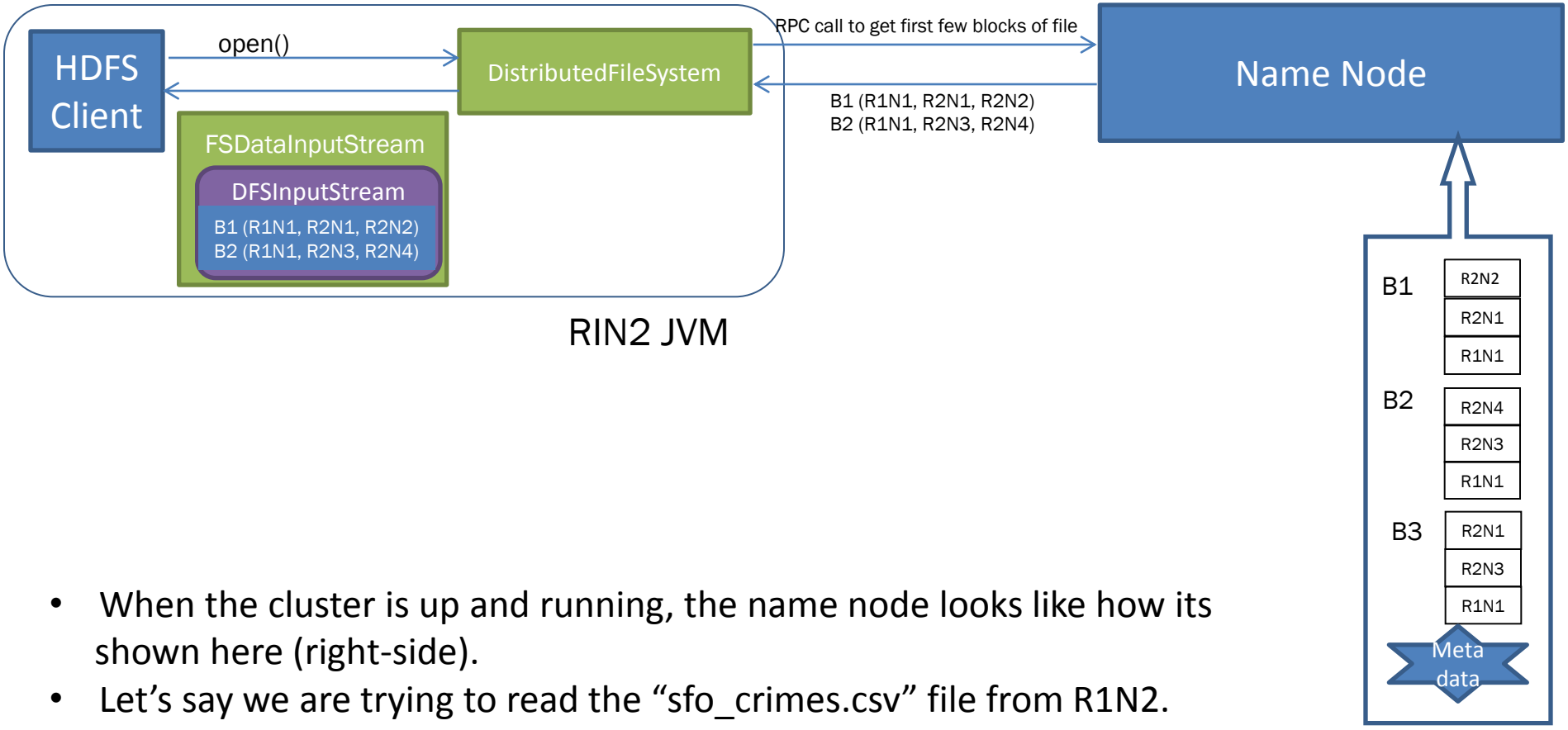
Anatomy of file read in Hadoop



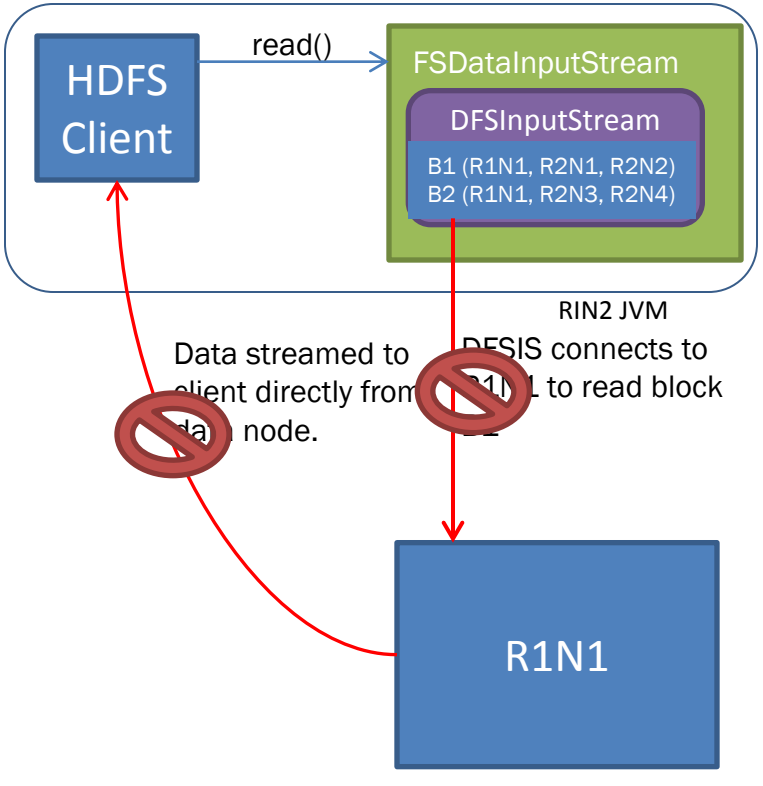
1. This is a Hadoop cluster with one name node and two racks named R1 and R2 in a data center D1. Each rack has 4 nodes and they are uniquely identified as R1N1, R1N2 and so on.
2. Replication factor is 3.
3. HDFS block size is 64 MB.
4. This cluster is used as an example to explain the concepts.



- Let's assume a file named "sfo_crime.csv" of size 192 MB is saved in this cluster.
- Also assume that the file was written from node R1N1.
- Metadata is written in name node.
- The file is split into 3 blocks each of size 64 MB. And each block is copied 3 times in the cluster.
- Along with data, a checksum will be saved in each block. This is used to ensure the data read from the block is read with out error.
- When cluster is started, the metadata will look as shown on top right corner.

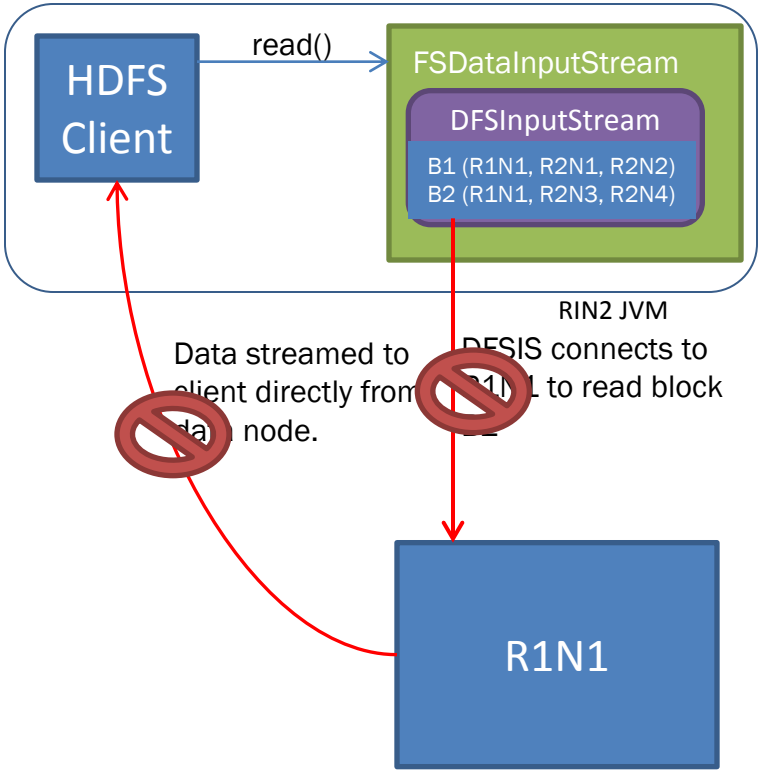


- When the cluster is up and running, the name node looks like how its shown here (right-side).
- Let's say we are trying to read the "sfo_crimes.csv" file from R1N2.
- So a HDFS Client program will run on R1N2's JVM.
- First the HDFS client program calls the method open() on a Java class DistributedFileSystem (subclass of FileSystem).
- DFS makes a RPC call returns first few blocks on the file. NN returns the address of the DN ORDERED with respect to the node from where the read is performed.
- The block information is saved in DFSInputStream which is wrapped in FSDataInputStream.
- In response to 'FileSystem.open()', HDFS Client receives this FSDataInputStream.



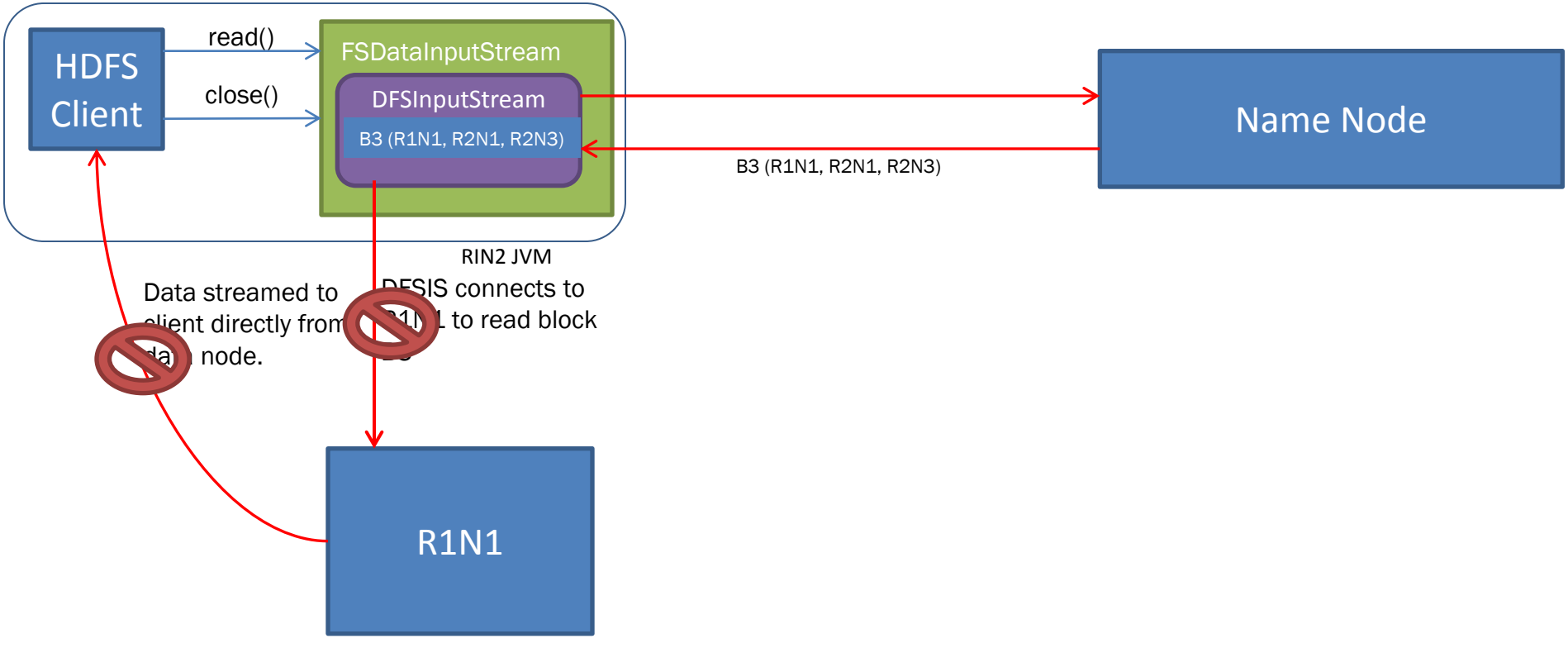
Name Node

- From now on HDFS Client deals with FSDataInputStream (FSDIS).
- HDFS Client invokes read() on the stream.
- Blocks are read in order. DFSIS connects to the closest node (R1N1) to read block B1.
- DFSIS connects to data node and streams data to client, which calls read() repeatedly on the stream. DFSIS verifies checksums for the data transferred to client.
- When the block is read completely, DFSIS closes the connection.



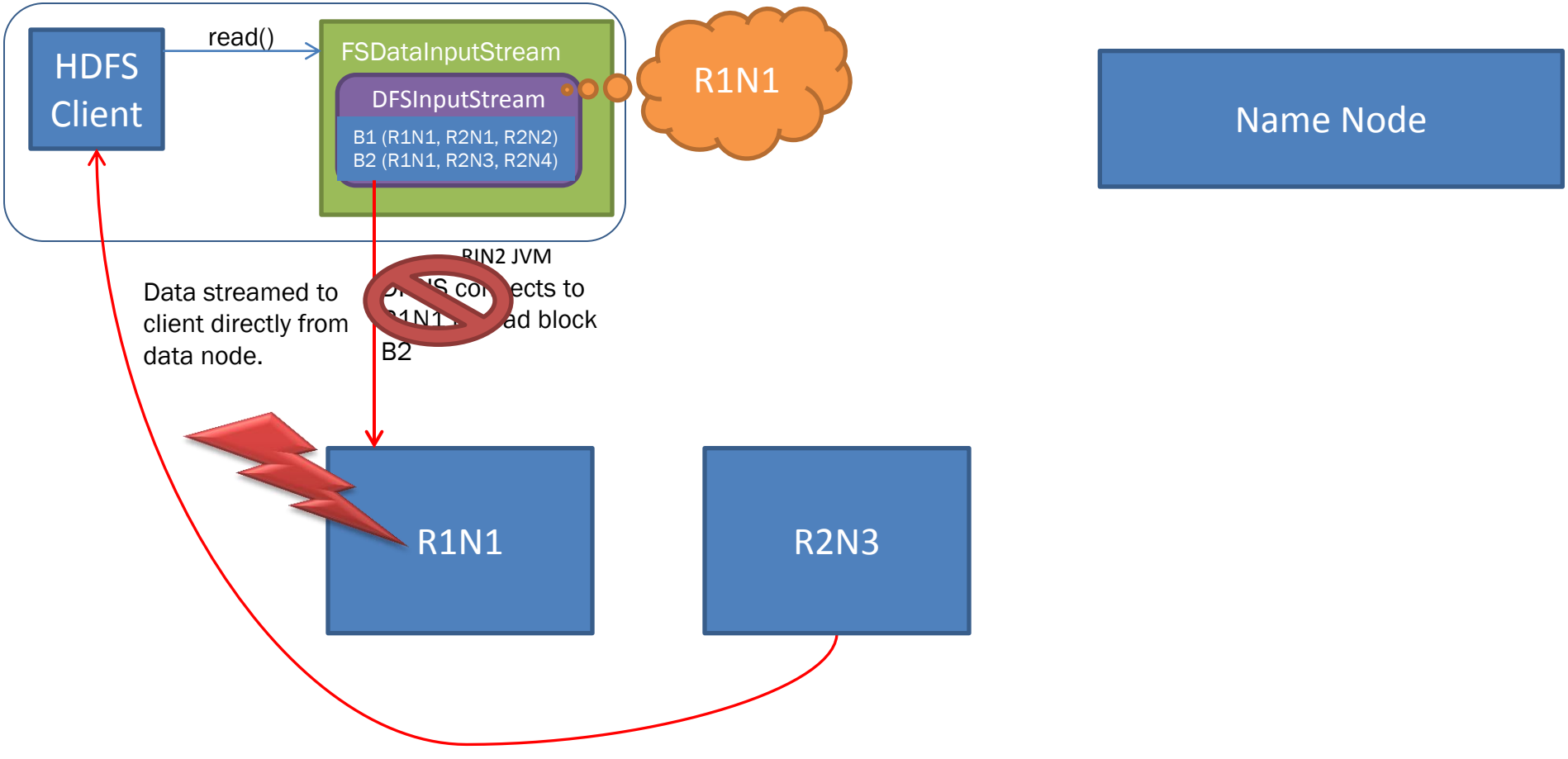
Name Node

- Next DFSIS attempts to read block B2. As mentioned earlier, the previous connection is closed and a fresh connection is made to the closest node (R1N1) of block B2.



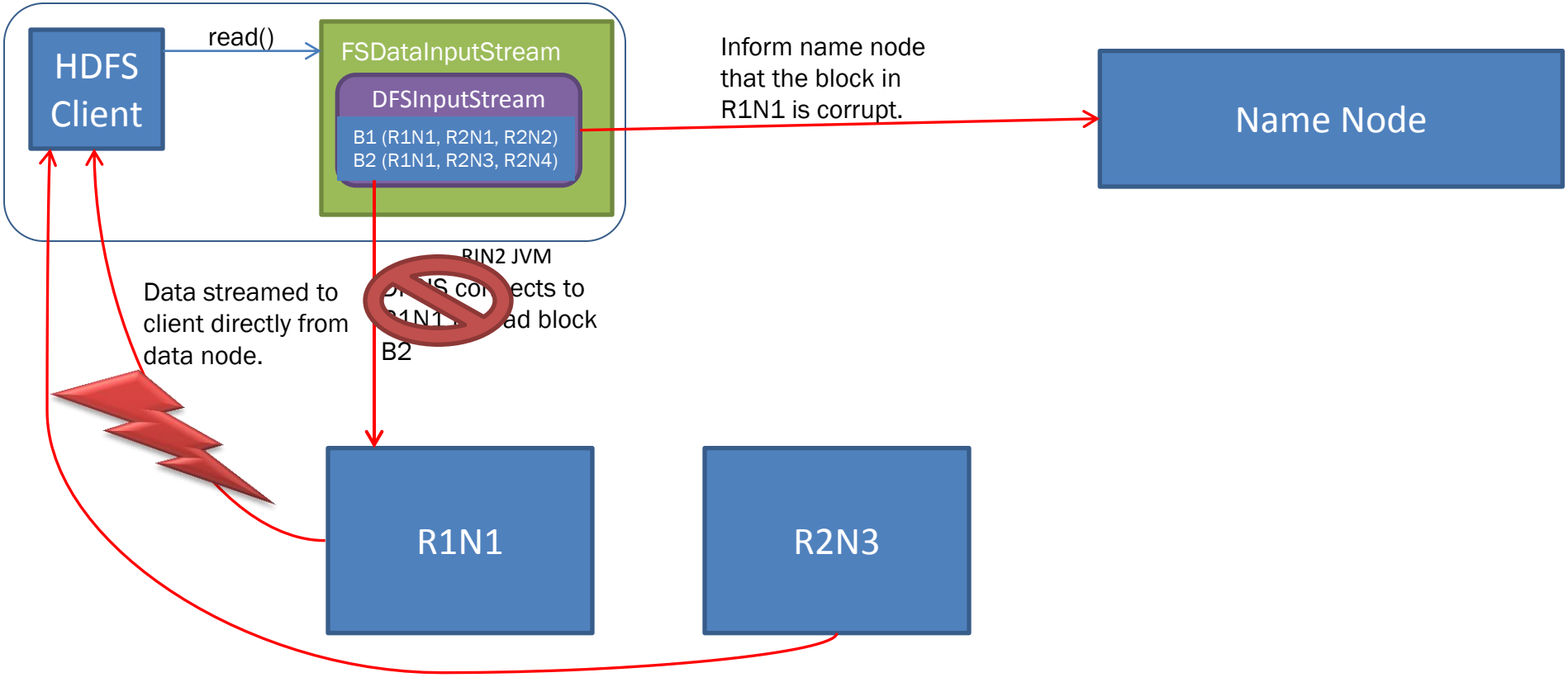
- Now DFSIS has read all blocks returned by the first RPC call (B1 & B2). But the file is not read completely. In our case there is one more block to read.
- DFSIS calls name node to get data node locations for next batch of blocks as needed.
- After the complete file is read for the HDFS client call close().

Anatomy of file read – Data Node Connection Error



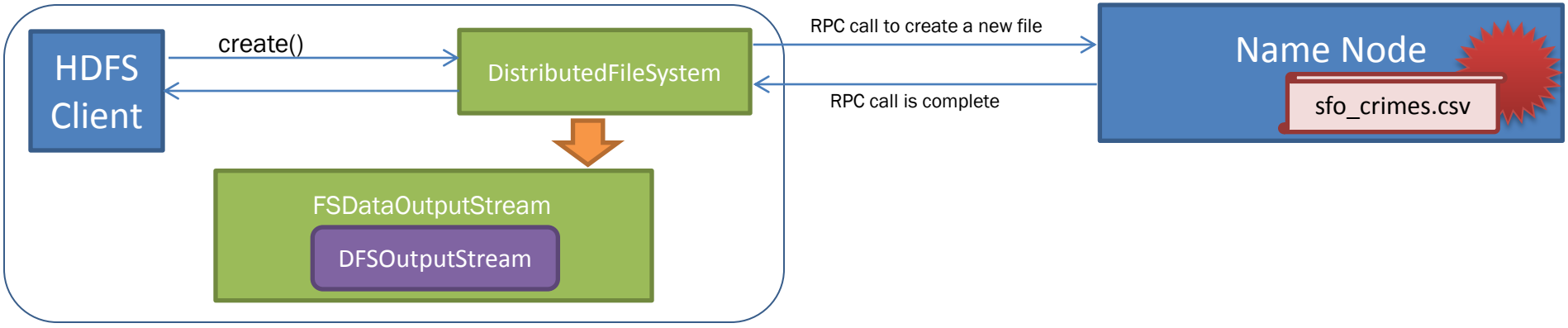
- Let's say there is some error while connecting to R1N1.
- DFSIS remembers this info, so it won't try to read from R1N1 for future blocks. Then it tries to connect to next closest node (R2N3).

Anatomy of file read – Data Node Checksum Error



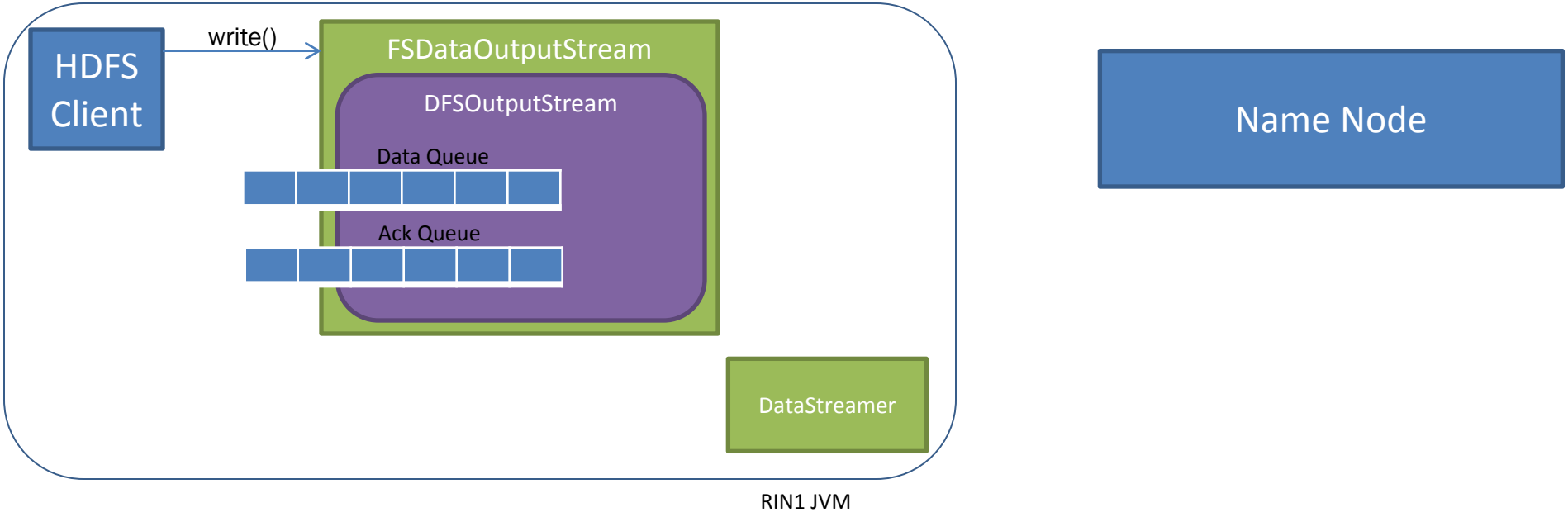
- Let's say there is a checksum error. This means the block is corrupt.
- Information about this corrupt block is sent to name node. Then DFSIS tries to connect to next closest node (R2N3).

Anatomy of file write

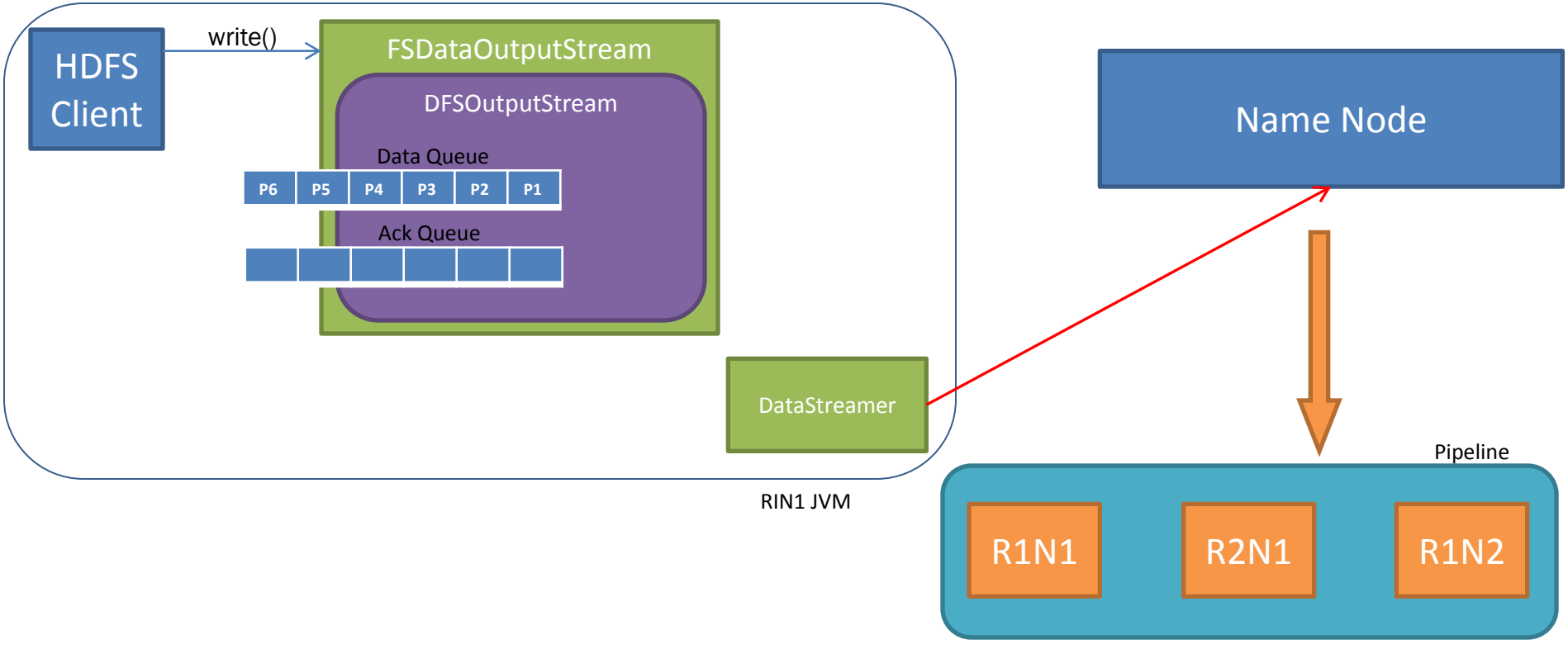


- Let's say we are trying to write the "sfo_crimes.csv" file from R1N1.
- So a HDFS Client program will run on R1N1's JVM.

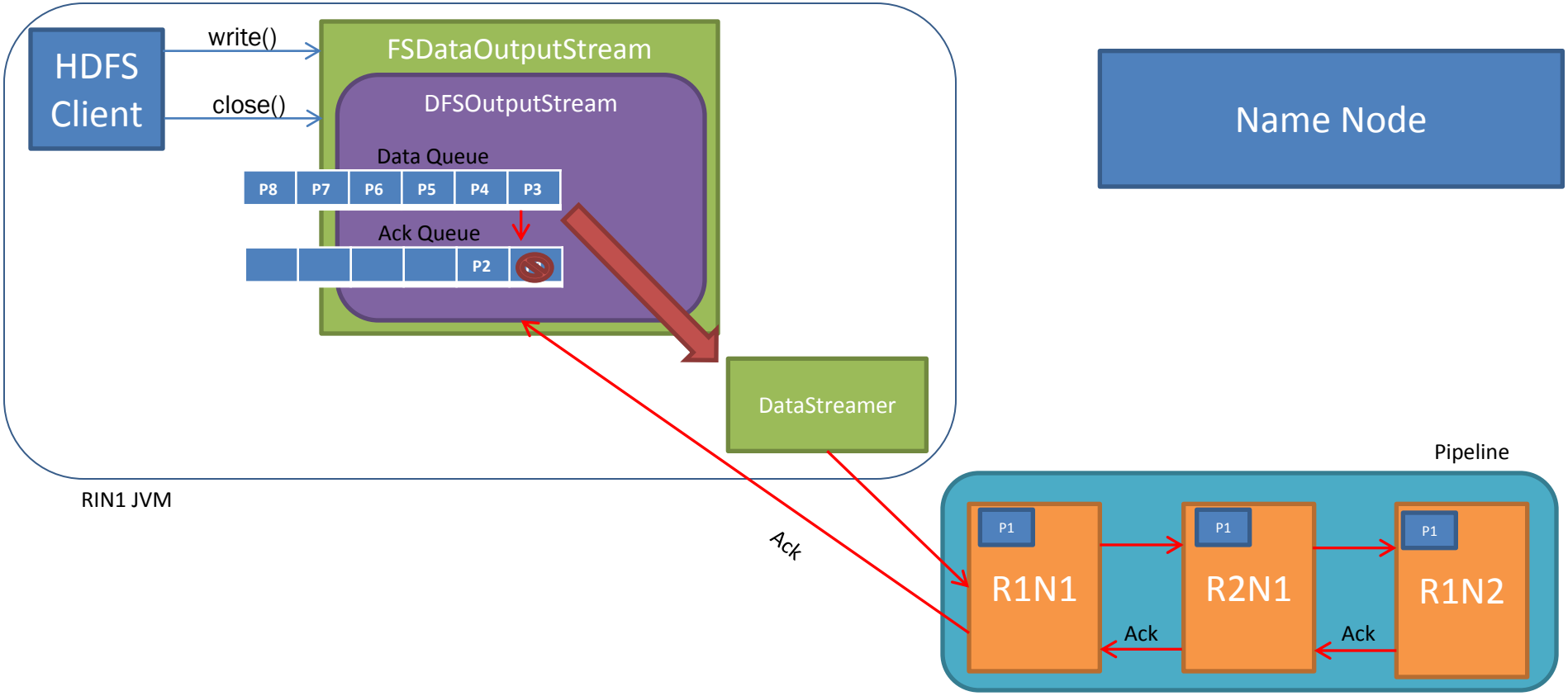
- First the HDFS client program calls the method create() on a Java class DistributedFileSystem (subclass of FileSystem).
- DFS makes a RPC call to name node to create a new file in the file system's namespace. No blocks are associated to the file at this stage.
- Name node performs various checks; ensures the file doesn't exists, the user has the right permissions to create the file. Then name node creates a record for the new file.
- Then DFS creates a FSDataOutputStream for the client to write data to. FSDOS wraps a DFSOutputStream, which handles communication with DN and NN.
- In response to 'FileSystem.create()', HDFS Client receives this FSDataOutputStream.



- From now on HDFS Client deals with FSDataOutputStream.
- HDFS Client invokes write() on the stream.
- Following are the important components involved in a file write;
  - **Data Queue:** When client writes data, DFDOS splits into packets and writes into this internal queue.
  - **DataStreamer**: The data queue is consumed by this component, which also communicates with name node for block allocation.
  - **Ack Queue:** Packets consumed by DataStreamer are temporaroly saved in an this internal queue.

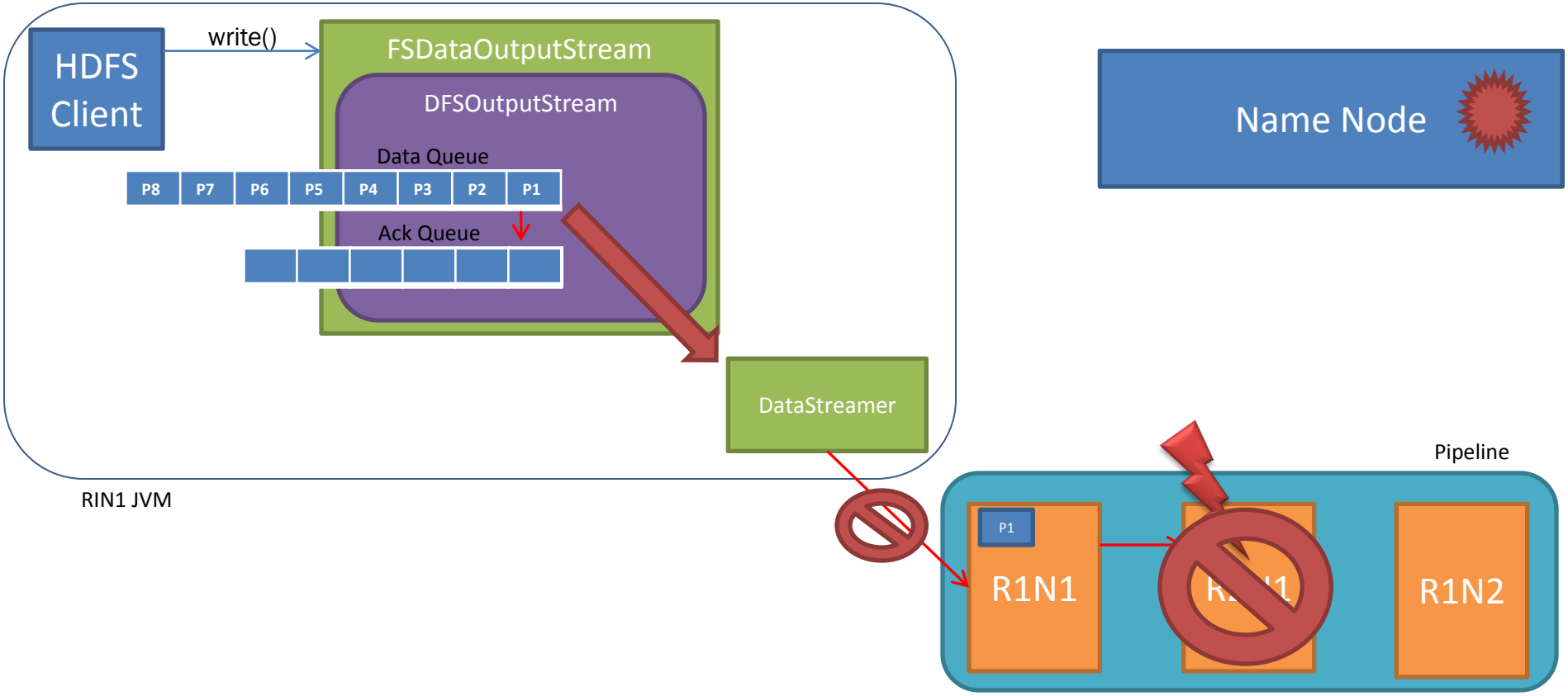


- As said, data written by client will be converted into packets and stored in data queue.
- DataStreamer communicates with NN to allocate new blocks by picking a list of suitable DNs to store the replicas. NN uses ‘Replica Placement’ as a strategy to pick DNs for a block.
- The list of DNs form a pipeline. Since the replication factor is assumed as 3, there are 3 nodes picked by NN.



- DataStreamer consumes few packets from data queue. A copy of the consumed data is stored in 'ack queue'.
- DataStreamer streams the packet to first node in pipeline. Once the data is written in DN1, the data is forwarded to next DN. This repeats till last DN.
- Once the packet is written to the last DN, an acknowledgement is sent from each DN to DFSOS. The packet P1 is removed from Ack Queue.
- The whole process continues till a block is filled. After that, the pipeline is closed and DataStreamer asks NN for fresh set of DNs for next block. And the cycle repeats.
- HDFS Client calls the close() method once the write is finished. This would flush all the remaining packets to the pipeline & waits for ack before informing the NN that the write is complete.

Anatomy of file WRITE – Data Node WRITE Error



- A normal write begins with a write() method call from HDFS client on the stream. And let's say an error occurred while writing to R2N1.
- The pipeline will be closed.
- Packets in ack queue are moved to front data queue.
- The current block on good DNs are given a new identity and its communicated to NN, so the partial block on the failed DN will be deleted if the failed DN recovers later.
- The failed data node is removed from pipeline and the remaining data is written to the remaining two DNs.
- NN notices that the block is under-replicated, and it arranges for further replica to be created on another node.

Thank you