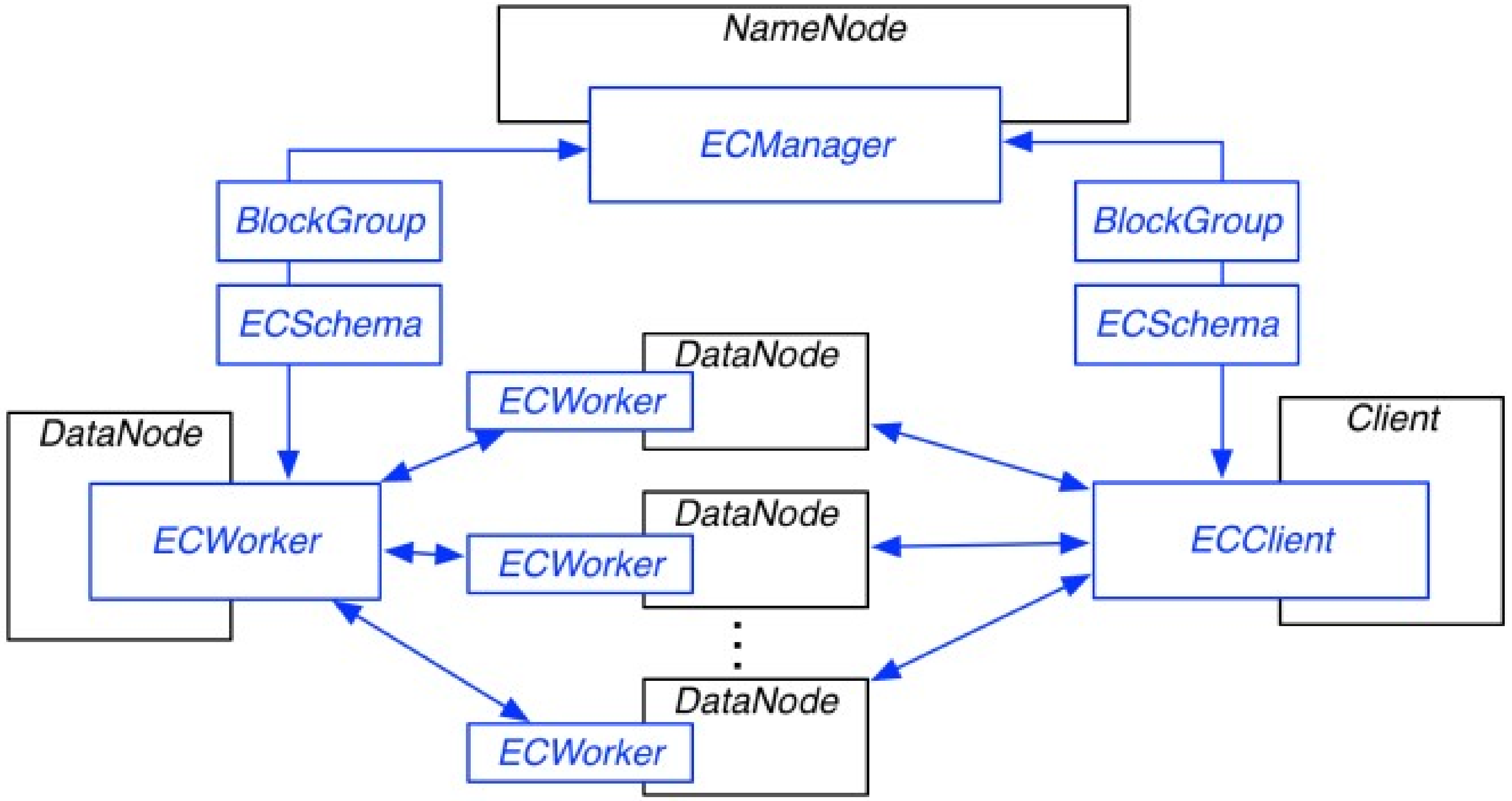
Erasure Coding module in Hadoop-3.0.0: Architecture, implementation details, striping, encode and decode

Online and offline encoding

- Encoding is done both online and offline
- Offline
  - Replicated initially, converted to erasure coded form when conditions are met
- Codec processing is done by both the client and DataNode
  - Client calculates parity data for the initial online encoding, and reconstructs lost original data during read requests.
  - DataNode builds and stores coded blocks in proactive/background recovery, as well as converts from replicated to EC forms.

Architecture



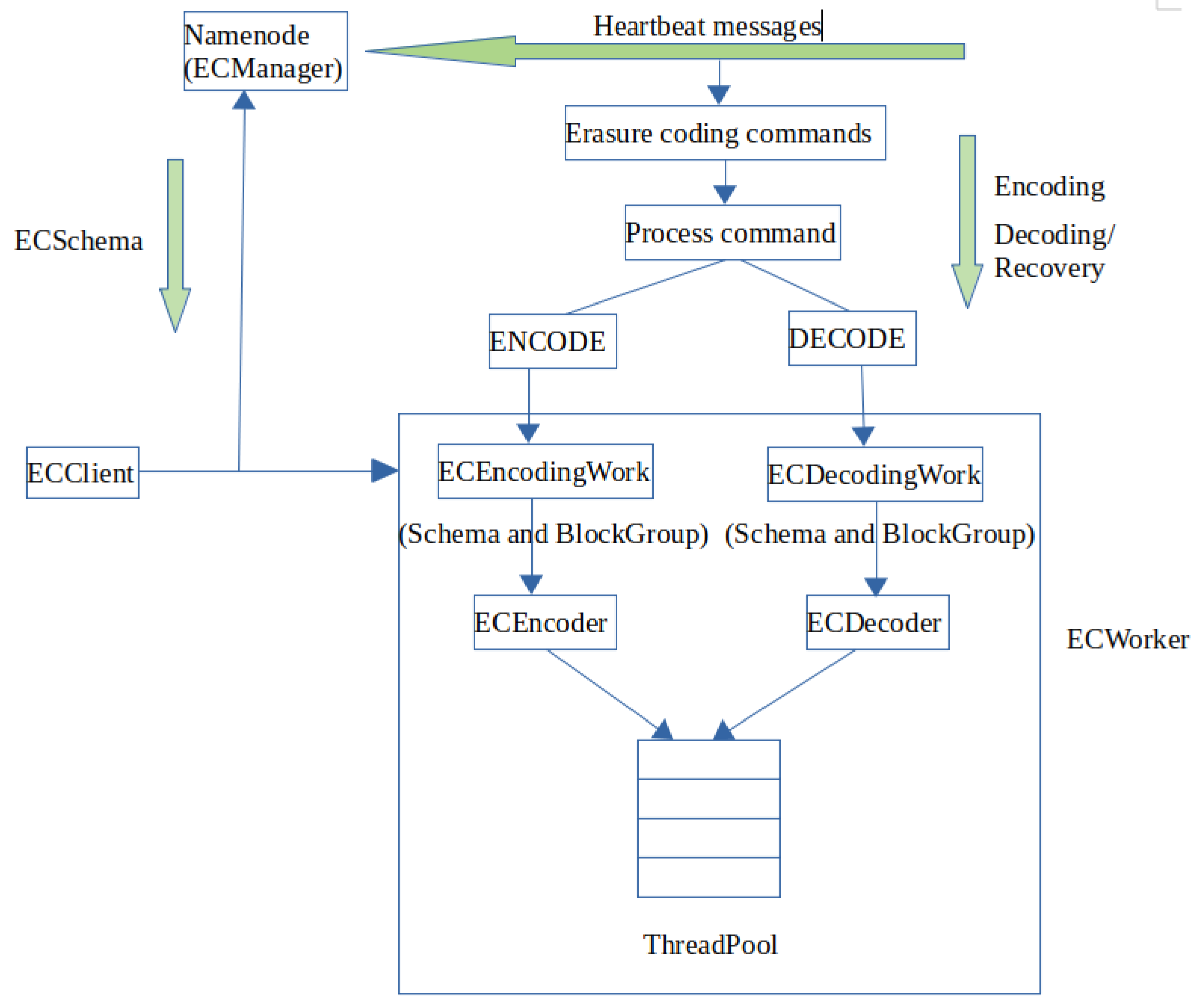
Components

- ECClient
  - extension to the HDFS client which stripes data to and from multiple DataNodes
- ECManager
  - resides on the NameNode and manages EC block groups
  - takes care of group allocation, placement, health monitoring, and coordination of recovery work
- ECWorker
  - resides at the DataNode
  - daemon thread that listens for recovery or conversion commands from ECManager
  - serves recovery requests by reading data from peer DataNodes and carrying out codec calculation

ECClient

- Connects to Hadoop Filesystem and perform EC tasks on a file
- Communicates with the NameNode daemon, and connects directly to DataNodes to read(decoding) or write(encoding) block data.

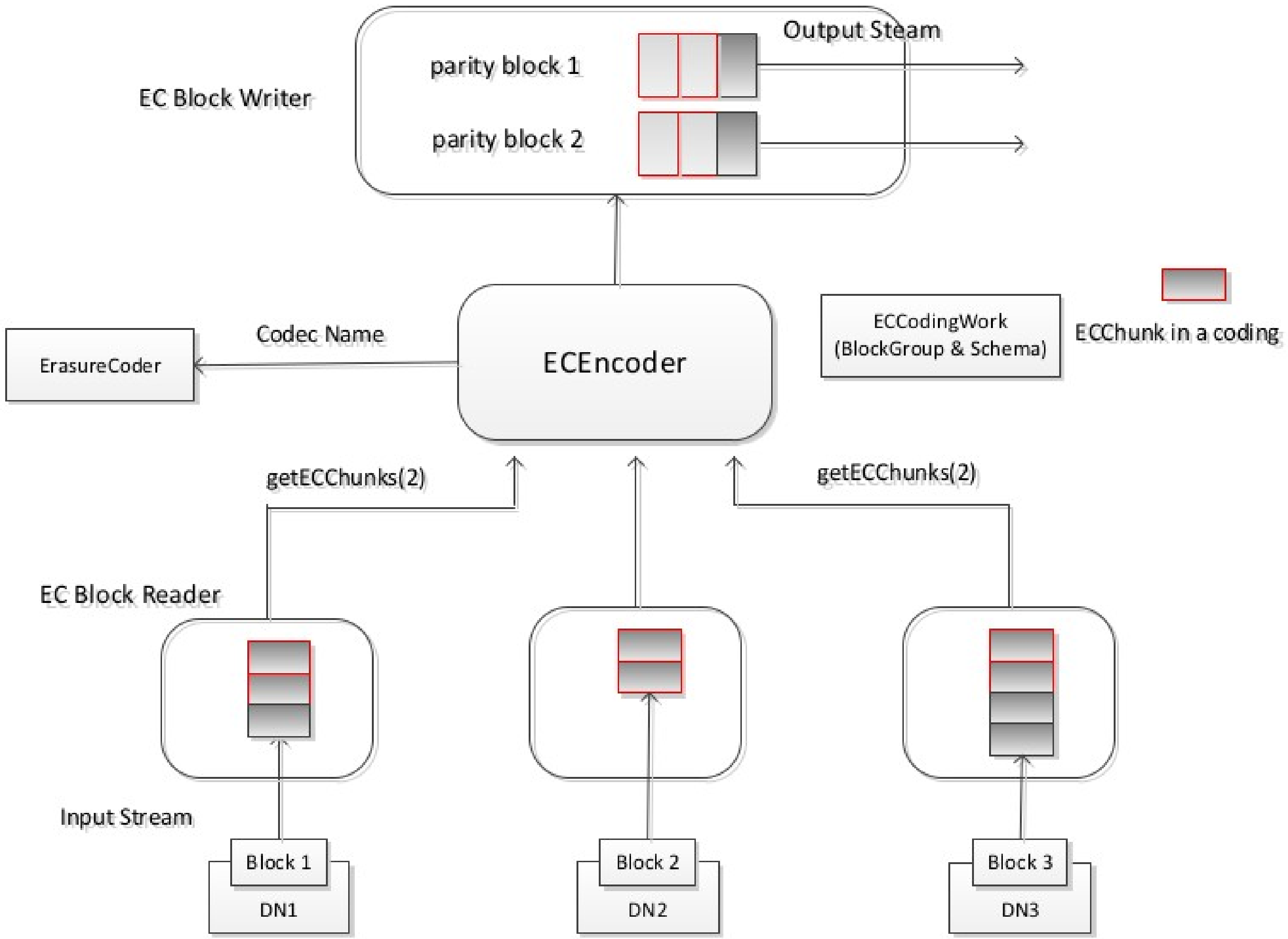
ECWorker design



ECWorker

- NameNode assigns encoding/decoding work to DataNodes via heartbeat messages.
- NameNode commands are parsed and ECEncodingWork or ECDecodingWork are constructed accordingly (wrapping BlockGroup plus schema info)
- ECWorker processes ECWork (either ECEncodingWork or ECDecodingWork) by creating ECEncoder or ECDecoder accordingly.
- The resultant ECEncoder or ECDecoder will be put into ECCoderQueue and processed by a thread pool.
- Decoding work can also occur in the background (passive/offline) recovery or to serve clients for erased blocks on demand (active/online recovery).
- Assigned priorities

ECEncoder



20/08/20

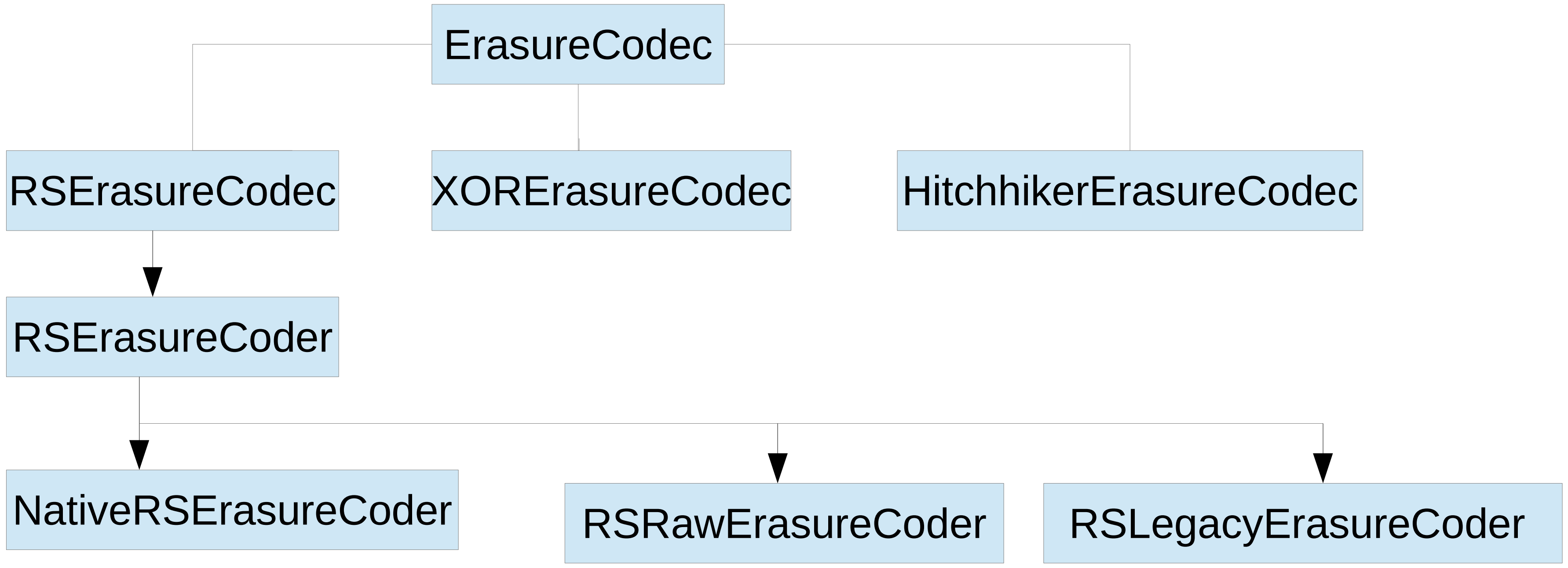
ECEncoder

- Utilizes BlockReader to read source input blocks (locally or remotely), and BlockWriter to write resultant output blocks (locally or remotely)
- Creates and utilizes an ErasureCoder with the codec name specified in the schema for EC computation
- Once done, an ACK is sent to the namenode to perform follow-up cleaning work (eg: delete redundant replicas in conversion)

Implementation framework hierarchy

- ErasureCodec
  - a high level construct
  - covers all the essential aspects of a code
  - allows users to customize all the aspect behaviors in a central place
- ErasureCoder
  - in the middle level
  - to encode or decode a block group, supports multiple erasure codecs
  - A codec can have multiple implementations
    - rs_native is the native implementation that leverages ISAL library, rs_java is the pure java implementation and rs-legacy_java is the implementation in java ported from HDFS-RAID
- RawErasureCoder
  - to perform the concrete algorithm calculation at the lowest level
  - ErasureCoder can combine and make use of different RawErasureCoders for tradeoff
- All these levels can be configured through schema definition

Multiple codecs and coders

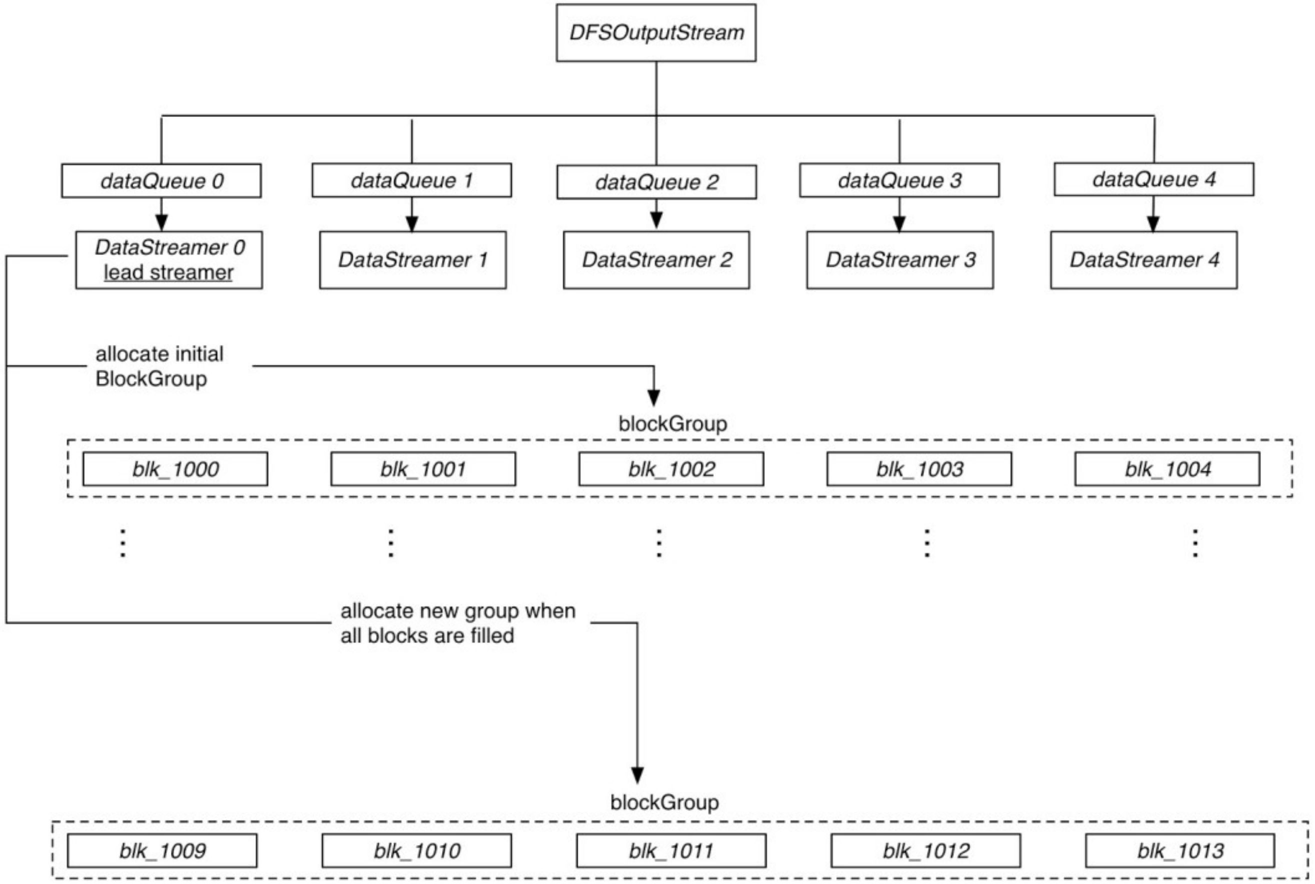


(Uses Intel ISAL)

BlockGroup

- A BlockGroup is formed during initial encoding and during the conversion from replication to EC
- It records the original and parity blocks in a coding group
- It is looked up during recoveries
- BlockGroups are not applicable for “traditional” HDFS files that have contiguous block layout
- Entire BlockGroup is represented with its first Block
  - Only the ID of the first block and the shared generation stamp are stored in a BlockGroup.
  - All the data blocks and parity blocks in a BlockGroup share the same generation stamp

Writing of a BlockGroup



Lead Streamer

- Dataqueues get filled in a round robin fashion
- Each DataStreamer works independently on its queue
- Lead streamer: commits block groups (after other streamers finish writing their blocks)
- When all blocks in the group is filled up, the lead streamer requests a new group from the NameNode
- When the datanode of the leading streamer fails, the other steamers cannot continue since a block group cannot be requested from the NameNode anymore.
- The size of the last BlockGroup can be smaller than block.getNumBytes() * NUM_DATA_BLOCKS (partial stripe)

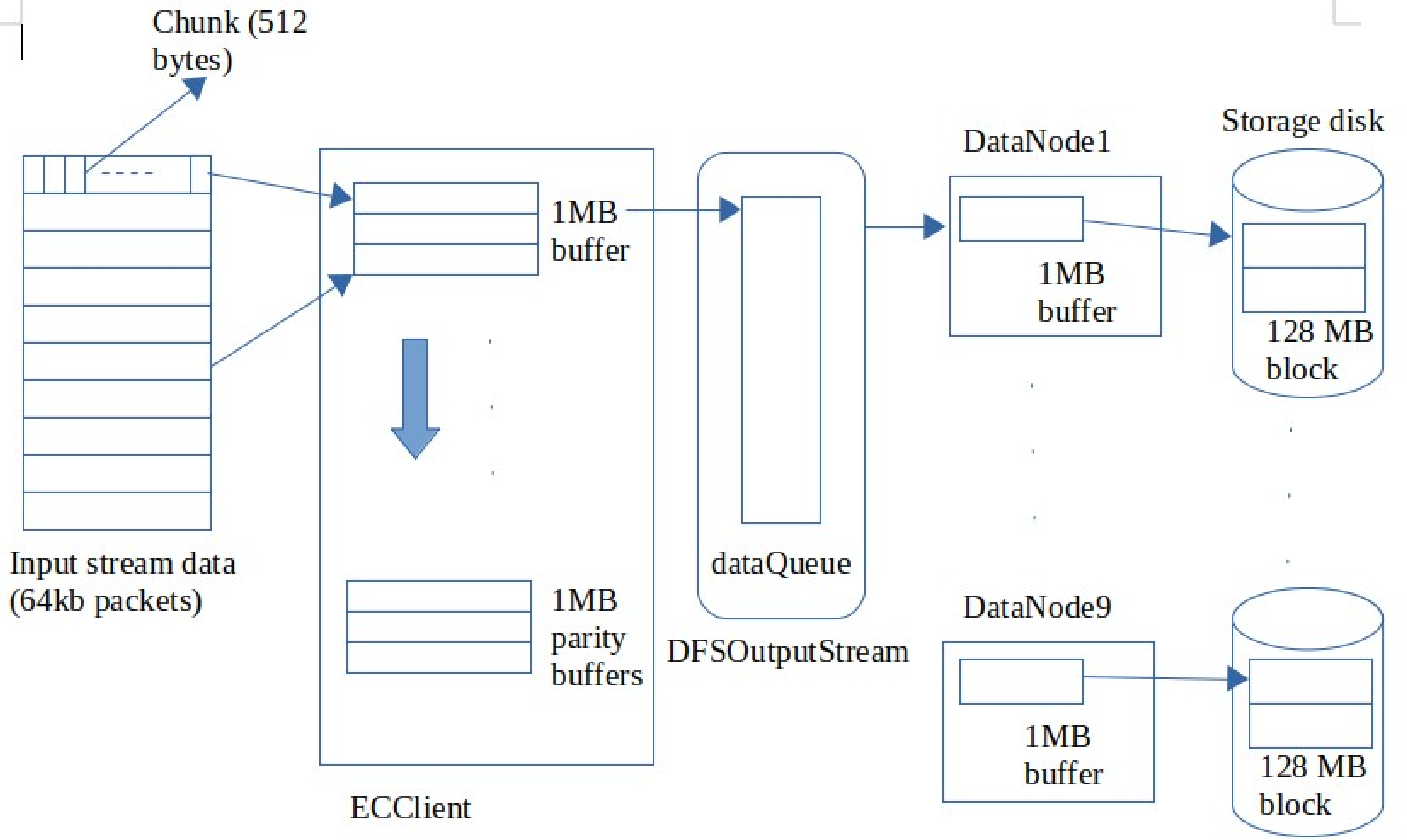
Implementation details: packets, chunks and buffers

- Data is broken up into packets, each packet is typically 64K in size.
- A packet comprises of chunks.
- Each chunk is typically 512 bytes and has an associated checksum with it.
- When a client fills up the currentPacket, it is enqueued into the dataQueue of DataStreamer.
- DataStreamer picks up packets from the dataQueue and sends it to the first datanode in the pipeline.

Buffers

- There are I/O transfer buffers between datanodes and the client.
- When a client writes some bytes to a datanode, the bytes are first copied to a buffer (e.g. 1MB) and then flushed later on.

Example of a (6,3) RS encoding

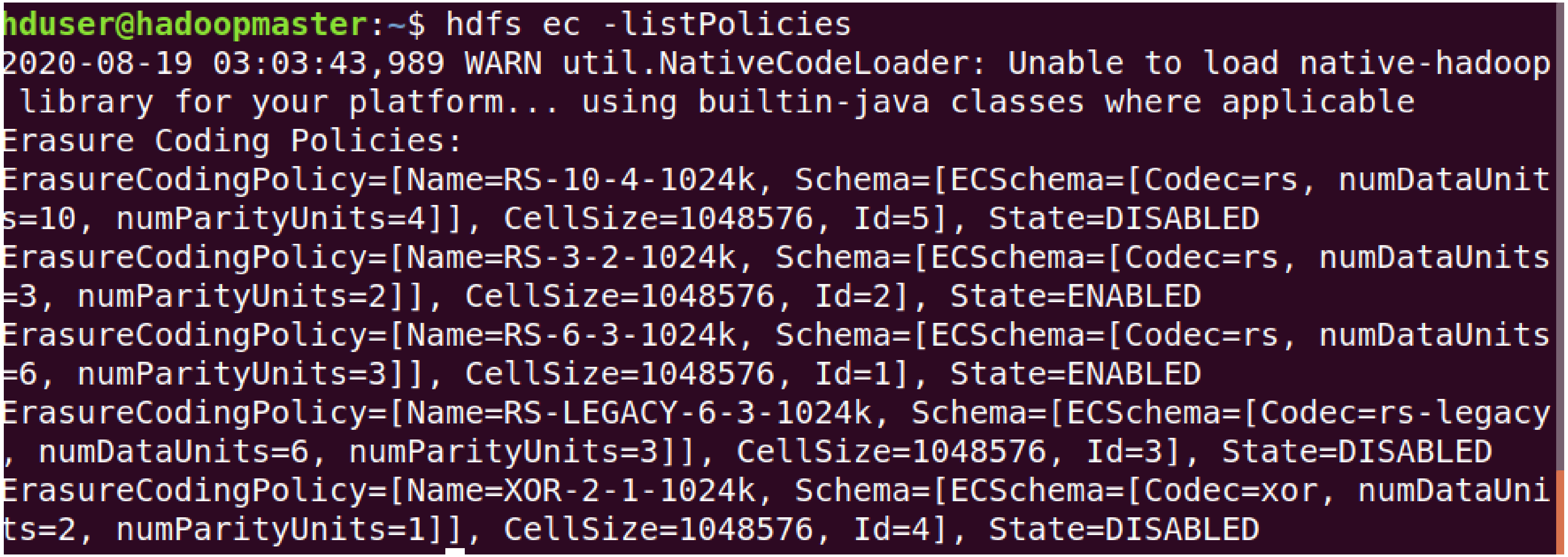


Schema and policy

- Defines and maintains common parameters for all codes or codecs
  - the EC algorithm, data block units and parity block units
- Recognized by the framework and used to initialize erasure coders
- The combination of schema and cell size forms an Erasure Coding Policy
- The EC storage policy as well as the codec schema, can be found in the file inode (meta-data)

Policies available by default

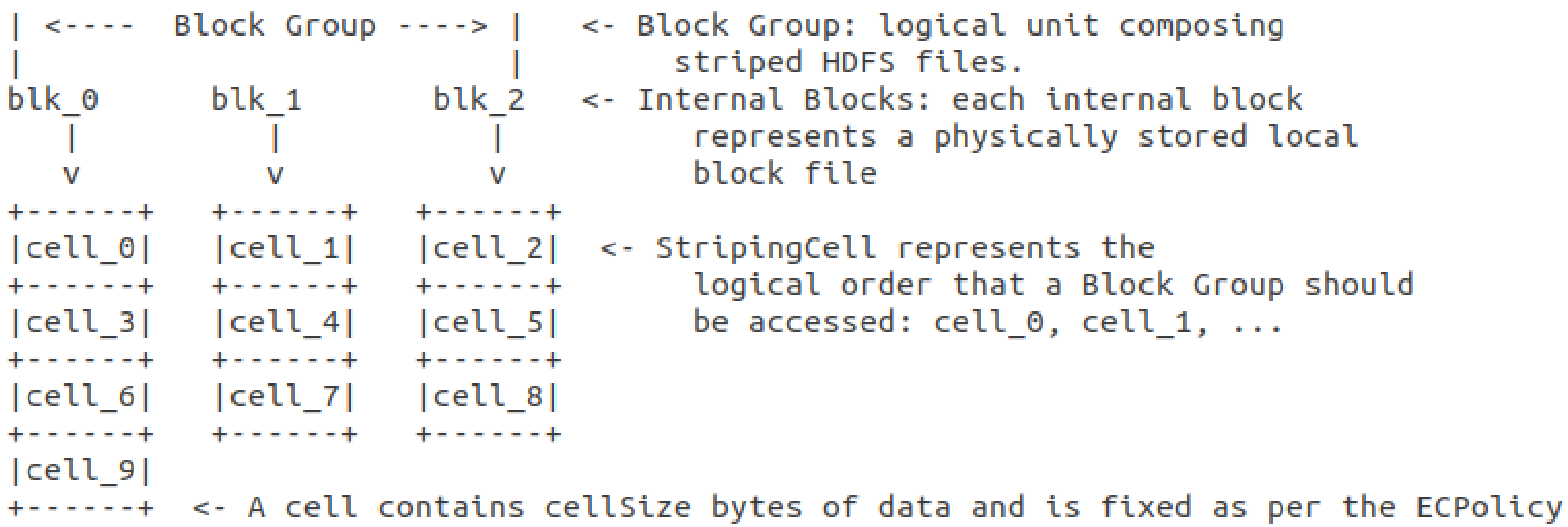
- Example from our Oracle cluster



```
hduser@hadoopmaster:~$ hdfs ec -listPolicies
2020-08-19 03:03:43,989 WARN util.NativeCodeLoader: Unable to load native-hadoop
 library for your platform... using builtin-java classes where applicable
Erasure Coding Policies:
ErasureCodingPolicy=[Name=RS-10-4-1024k, Schema=[ECSchema=[Codec=rs, numDataUnit
s=10, numParityUnits=4]], CellSize=1048576, Id=5], State=DISABLED
ErasureCodingPolicy=[Name=RS-3-2-1024k, Schema=[ECSchema=[Codec=rs, numDataUnits
=3, numParityUnits=2]], CellSize=1048576, Id=2], State=ENABLED
ErasureCodingPolicy=[Name=RS-6-3-1024k, Schema=[ECSchema=[Codec=rs, numDataUnits
=6, numParityUnits=3]], CellSize=1048576, Id=1], State=ENABLED
ErasureCodingPolicy=[Name=RS-LEGACY-6-3-1024k, Schema=[ECSchema=[Codec=rs-legacy
, numDataUnits=6, numParityUnits=3]], CellSize=1048576, Id=3], State=DISABLED
ErasureCodingPolicy=[Name=XOR-2-1-1024k, Schema=[ECSchema=[Codec=xor, numDataUni
ts=2, numParityUnits=1]], CellSize=1048576, Id=4], State=DISABLED
```

Note: ISAL is disabled

Striping cell in a BlockGroup



```
| <----  Block Group ----> |   <- Block Group: logical unit composing
|                          |        striped HDFS files.
blk_0      blk_1       blk_2   <- Internal Blocks: each internal block
   |          |           |          represents a physically stored local
   v          v           v          block file
+-------+  +-------+  +-------+
|cell_0|   |cell_1|   |cell_2|   <- StripingCell represents the
+-------+  +-------+  +-------+      logical order that a Block Group should
|cell_3|   |cell_4|   |cell_5|        be accessed: cell_0, cell_1, ...
+-------+  +-------+  +-------+
|cell_6|   |cell_7|   |cell_8|
+-------+  +-------+  +-------+
|cell_9|
+-------+  <- A cell contains cellSize bytes of data and is fixed as per the ECPolicy
```

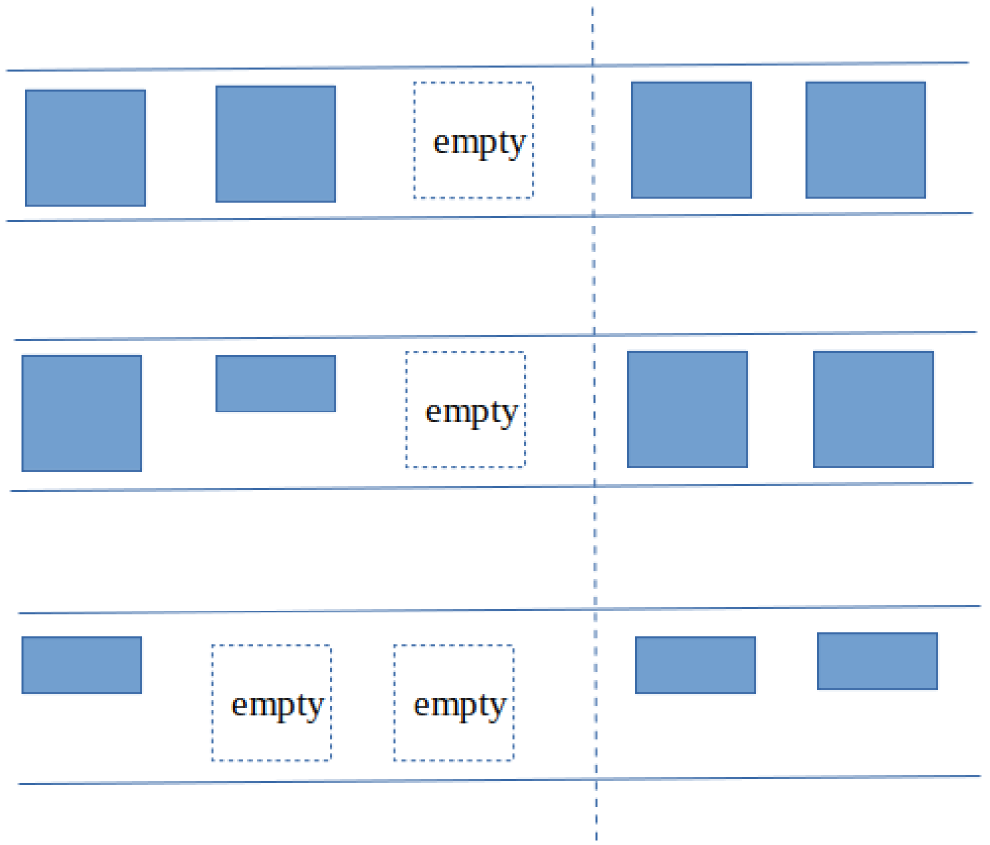
Encoding a stripe

- Striping greatly enhances sequential I/O performance by leveraging multiple disks in parallel
- When all data cells in a stripe are written successfully, they are encoded to generate parity cells
- These cells will be converted to packets and put to their DFSOutputStreamer's queue

Full and partial stripes acknowledgement

- A full stripe is ACKed when NUM_DATA_BLOCKS streamers have written the corresponding cells (including parity cells) of the stripe, and all previous full stripes are also ACKed.
- Partial stripes
  - write all parity cells
  - empty data cells are not written
  - parity cells are the length of the longest data cell(s).
  - For example, with RS(3,2), if we have data cells with lengths [1MB, 64KB, 0], the parity blocks will be length [1MB, 1MB]
  - Zero bytes are padded to the second data cell to make all cells of the same size (1MB).
- To be considered acked, a partial stripe needs at least NUM_DATA_BLOCKS empty or written cells
- For a partial stripe, written bytes of the block group is recorded.

Partial Stripe scenarios (3,2 RS)



Full data cells +
empty cell

Full data cells +
Parital data cells +
empty cell

Partial data cell +
empty cells

Reconstruction

- The minimum number of live stripe blocks should be no less than NUM_DATA_BLOCKS for a successful reconstruction to happen.
- 3 steps:
  - read bufferSize data from minimum number of sources required by reconstruction
  - decode data for targets
  - transfer data to targets
- Read is performed only once and reconstructs all unavailable blocks in a stripe
- If source blocks read are all data blocks, call encode; if there is a parity block, call decode
- Reconstructed blocks are forwarded to targets by constructing packets and sending them

Metrics

- Time taken to read from minimum source DNs required for reconstruction
- Time taken to reconstruct the targets
- Time taken to send/write reconstructed data to target DNs
- Remote bytes read

References

- Umbrella JIRA
  - Erasure Coding Support inside HDFS https://issues.apache.org/jira/browse/HDFS-7285 https://issues.apache.org/jira/browse/HDFS-7285
- Other related JIRAs
  - Striping
  - https://issues.apache.org/jira/browse/HDFS-7545
  - HDFS-7782, 8033, 7678, 8319, 7339, 8320
  - Codec and coders
  - HADOOP-11514, 11646, 11645, 11540, 11541, 11542, 11647
  - Configurable Policies
  - HDFS-7337, 13200, 11649
  - Metrics of interest
  - HDFS-7674, 8410, 8411, 8529, 8449
- Also, the source code of the project