Big data platform (Spark) performance acceleration

*Mentors: Tony Tan, Ning Wu, Yong Wang and **Theo Gkountouvas***

By:

Grishma Atul Thakkar

Virat Goradia

Nipun Midha

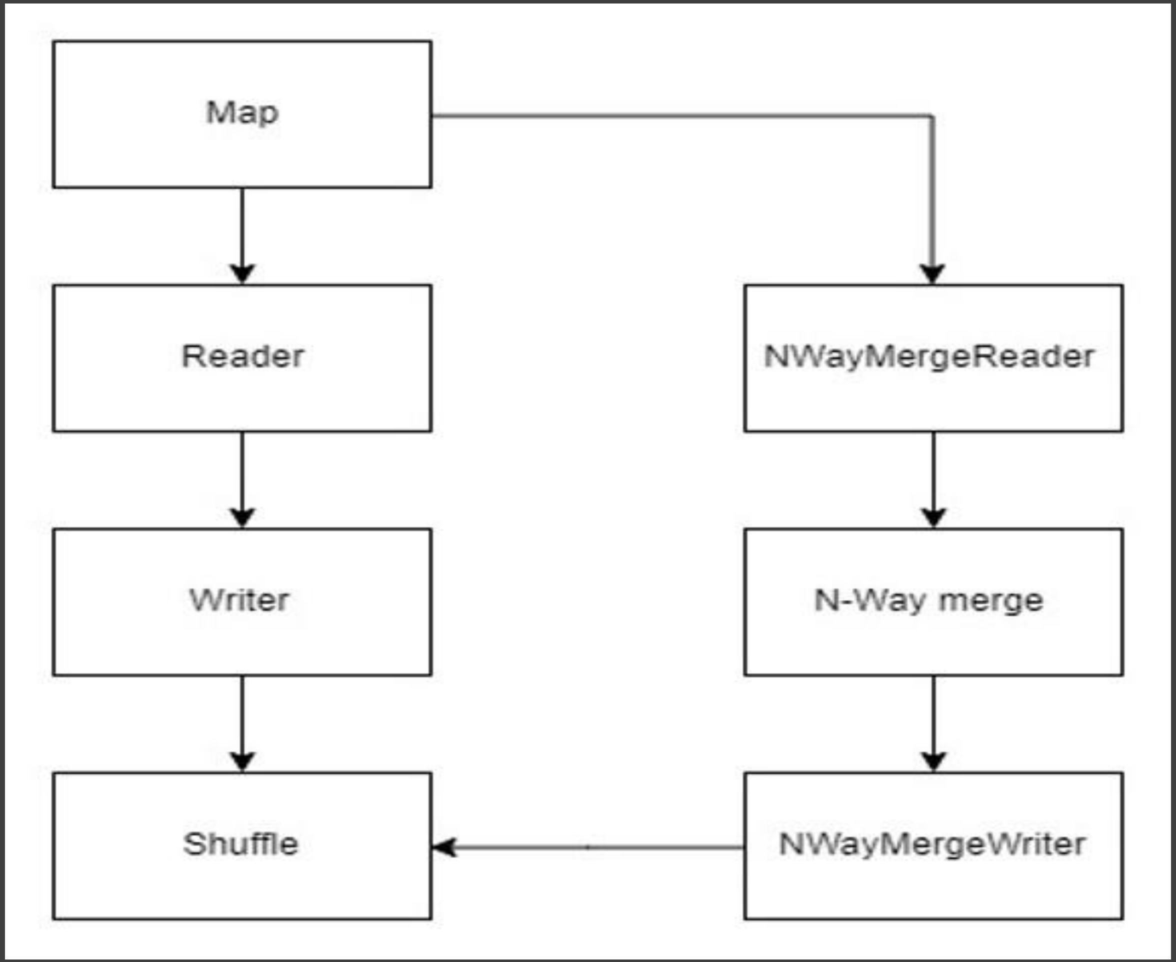
Baoshu Brady Qi

Recap

Created the following:

- NWayMergeReader
- NWayMergeWriter
- NWayMergeShuffleHandle
- isNWayMerge
- shouldNWayMerge
- registerNWayShuffle

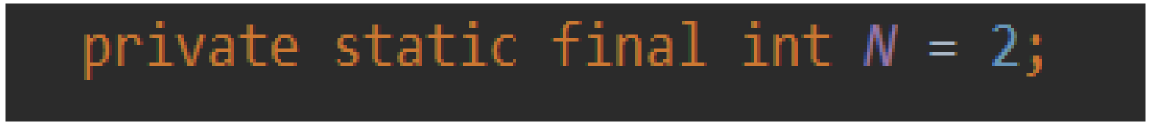
Recap(Continued)


$$\begin{tikzcd}
\text{Map} \arrow[d] \arrow[r] & \text{NWayMergeReader} \arrow[d] \\
\text{Reader} \arrow[d] & \text{N-Way merge} \arrow[d] \\
\text{Writer} \arrow[d] & \text{NWayMergeWriter} \arrow[dl] \\
\text{Shuffle} &
\end{tikzcd}$$

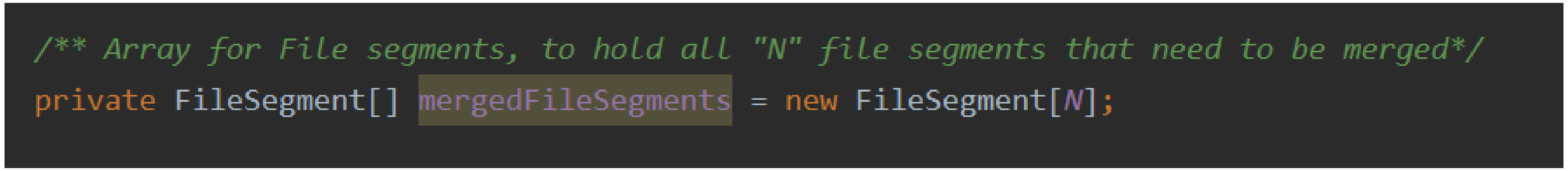
Sprint Goals

- Design strategies to implement N-Way merge algorithm.
- Implement the N-Way merge algorithm in parallel.

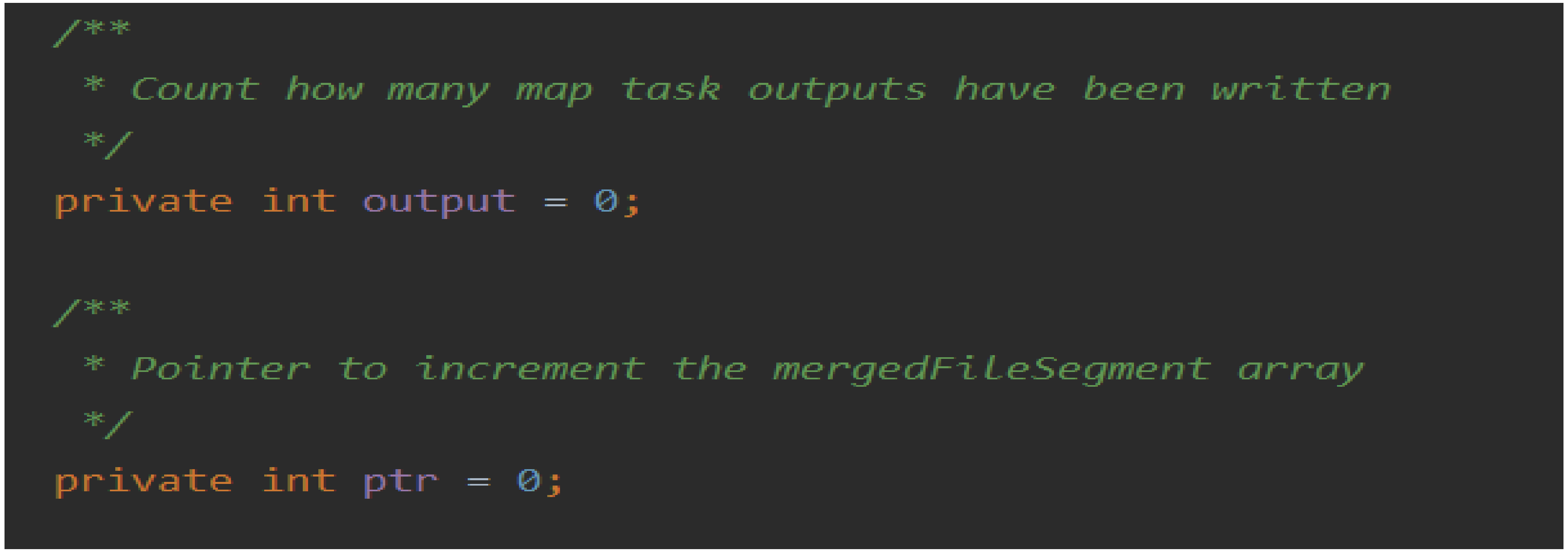
Approach 2



```
private static final int N = 2;
```

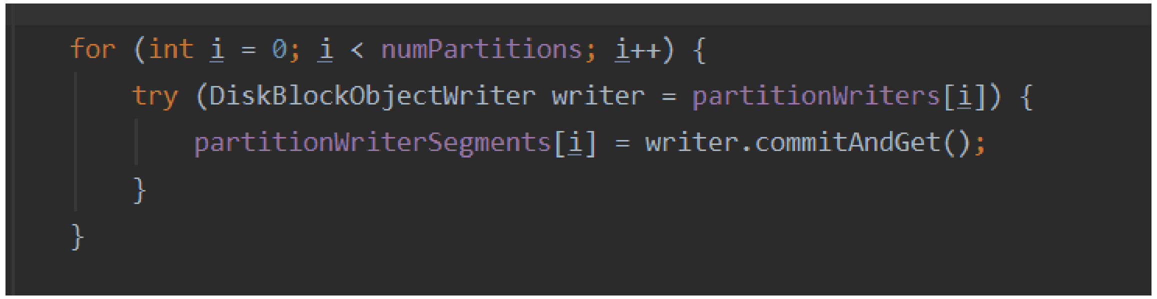


```
/** Array for File segments, to hold all "N" file segments that need to be merged*/
private FileSegment[] mergedFileSegments = new FileSegment[N];
```

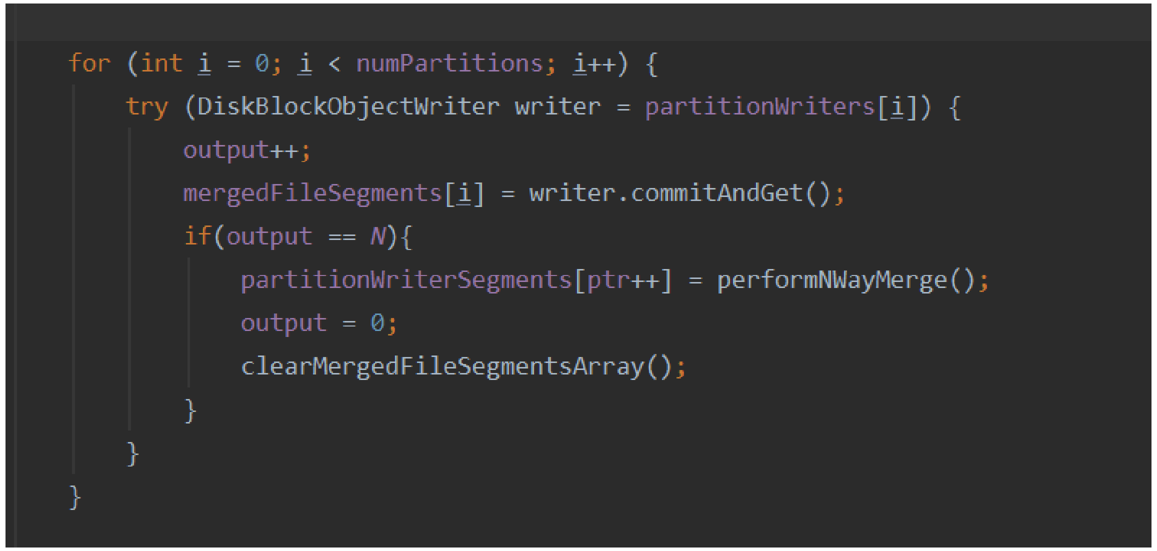


```
/**
 * Count how many map task outputs have been written
 */
private int output = 0;

/**
 * Pointer to increment the mergedFileSegment array
 */
private int ptr = 0;
```

Old Vs New



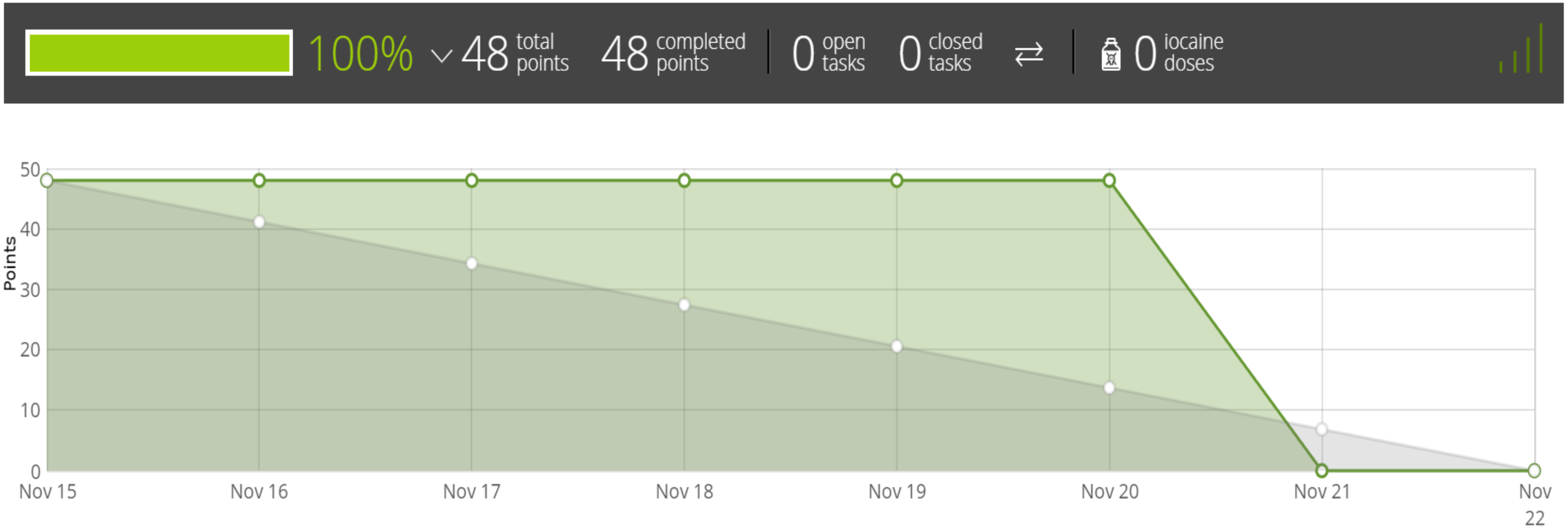
```java
for (int i = 0; i < numPartitions; i++) {
    try (DiskBlockObjectWriter writer = partitionWriters[i]) {
        partitionWriterSegments[i] = writer.commitAndGet();
    }
}
```



```java
for (int i = 0; i < numPartitions; i++) {
    try (DiskBlockObjectWriter writer = partitionWriters[i]) {
        output++;
        mergedFileSegments[i] = writer.commitAndGet();
        if(output == N){
            partitionWriterSegments[ptr++] = performNWayMerge();
            output = 0;
            clearMergedFileSegmentsArray();
        }
    }
}
```

Challenges

Burndown Chart


100% 48 total points 48 completed points 0 open tasks 0 closed tasks 0 iocaine doses

DEMO

Schedule Merge Task

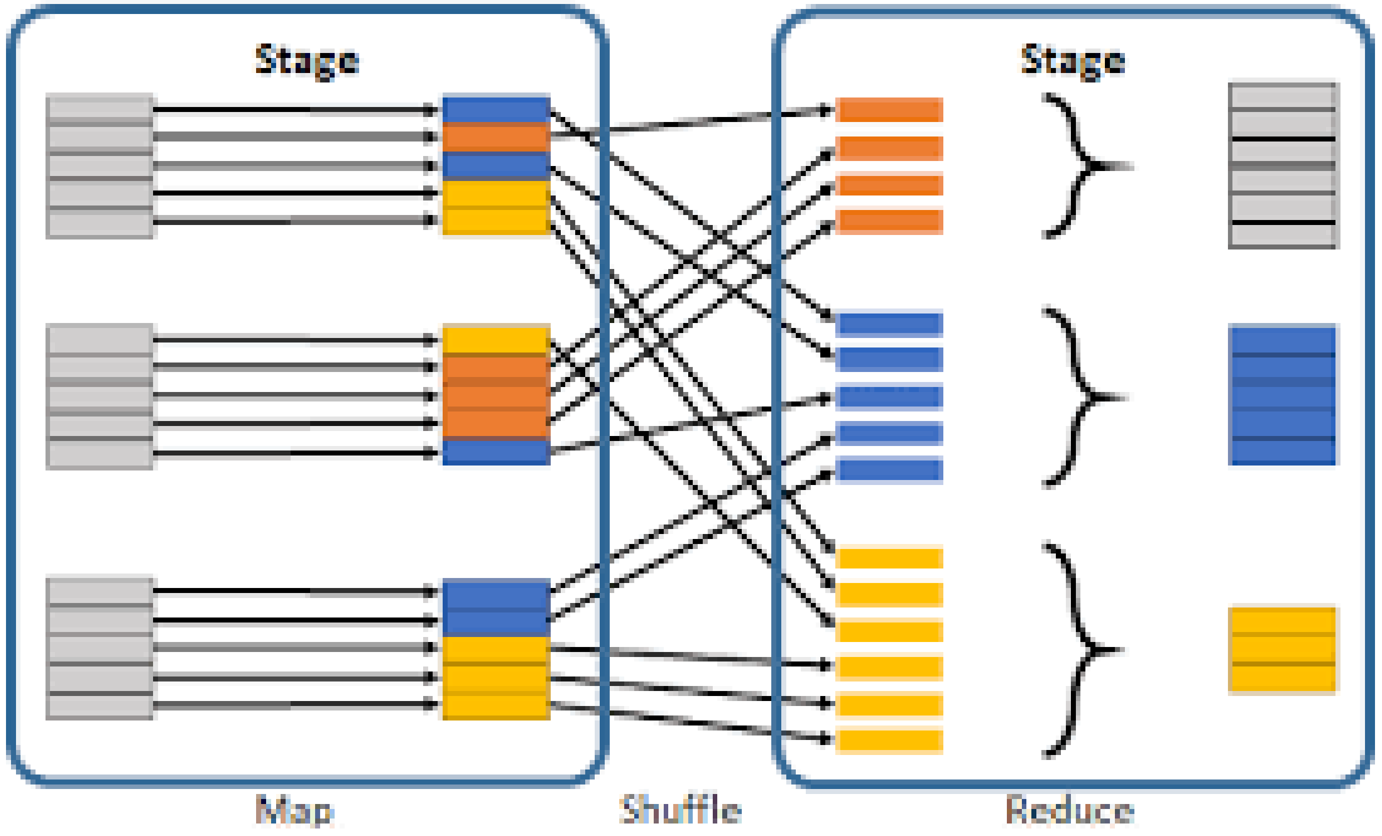
MergeTask

MergeReader

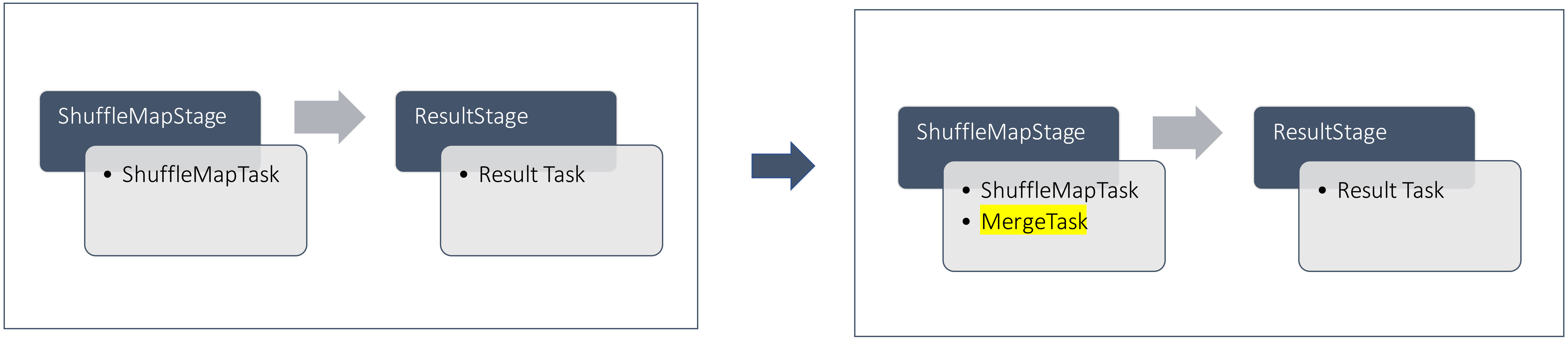
MergeWriter

Refactored ShuffleReader

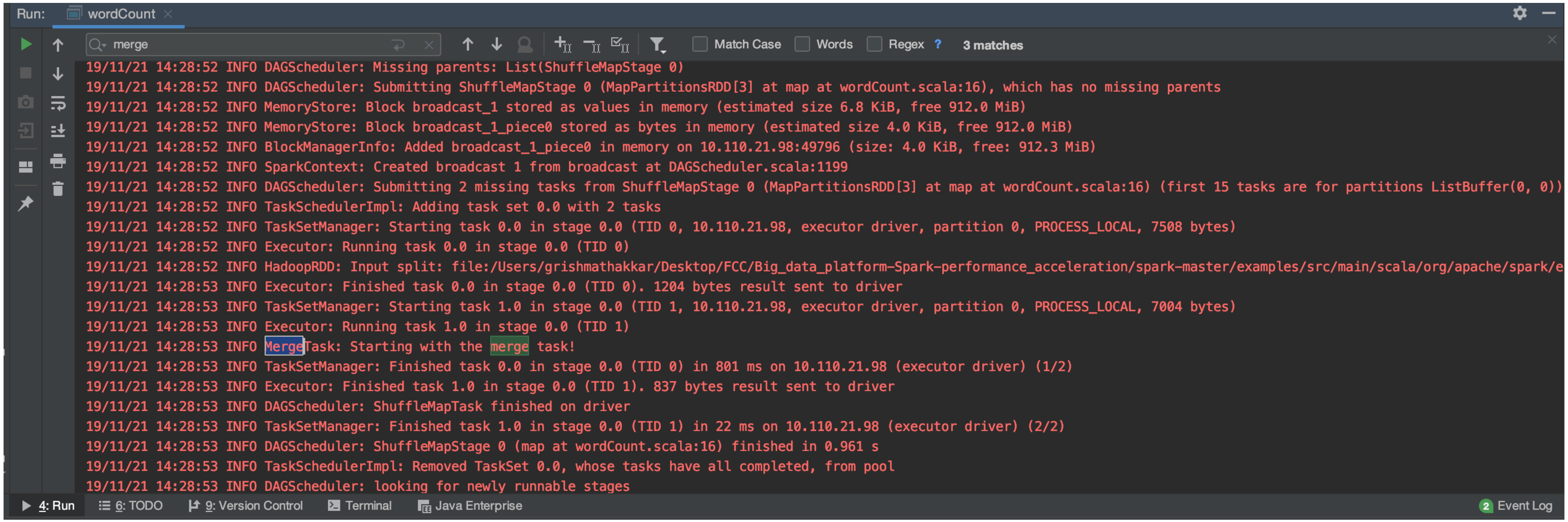
What is a ShuffleMapStage and ResultStage?



What did we do?

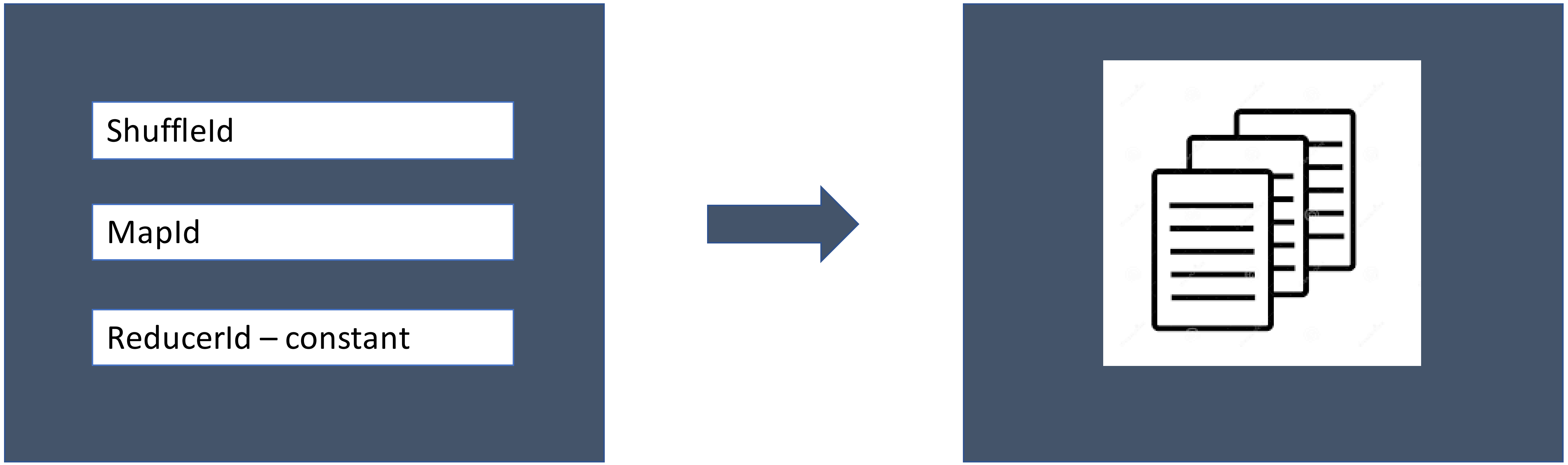


Schedule Merge Task



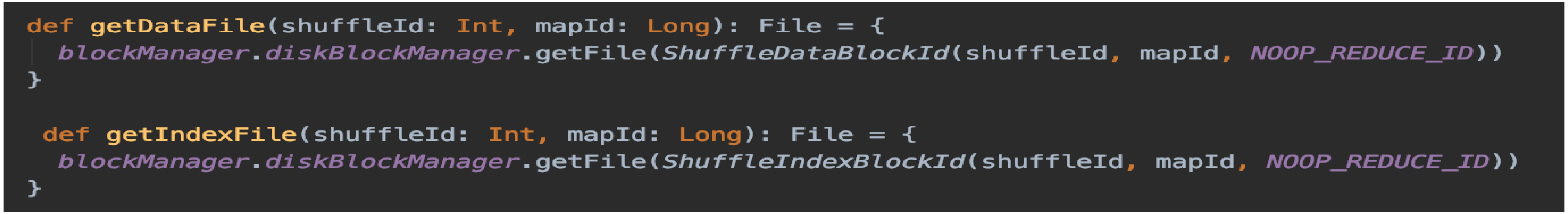
```
19/11/21 14:28:52 INFO DAGScheduler: Missing parents: List(ShuffleMapStage 0)
19/11/21 14:28:52 INFO DAGScheduler: Submitting ShuffleMapStage 0 (MapPartitionsRDD[3] at map at wordCount.scala:16), which has no missing parents
19/11/21 14:28:52 INFO MemoryStore: Block broadcast_1 stored as values in memory (estimated size 6.8 KiB, free 912.0 MiB)
19/11/21 14:28:52 INFO MemoryStore: Block broadcast_1_piece0 stored as bytes in memory (estimated size 4.0 KiB, free 912.0 MiB)
19/11/21 14:28:52 INFO BlockManagerInfo: Added broadcast_1_piece0 in memory on 10.110.21.98:49796 (size: 4.0 KiB, free: 912.3 MiB)
19/11/21 14:28:52 INFO SparkContext: Created broadcast 1 from broadcast at DAGScheduler.scala:1199
19/11/21 14:28:52 INFO DAGScheduler: Submitting 2 missing tasks from ShuffleMapStage 0 (MapPartitionsRDD[3] at map at wordCount.scala:16) (first 15 tasks are for partitions ListBuffer(0, 0))
19/11/21 14:28:52 INFO TaskSchedulerImpl: Adding task set 0.0 with 2 tasks
19/11/21 14:28:52 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, 10.110.21.98, executor driver, partition 0, PROCESS_LOCAL, 7508 bytes)
19/11/21 14:28:52 INFO Executor: Running task 0.0 in stage 0.0 (TID 0)
19/11/21 14:28:52 INFO HadoopRDD: Input split: file:/Users/grishmathakkar/Desktop/FCC/Big_data_platform-Spark-performance_acceleration/spark-master/examples/src/main/scala/org/apache/spark/e
19/11/21 14:28:53 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0). 1204 bytes result sent to driver
19/11/21 14:28:53 INFO TaskSetManager: Starting task 1.0 in stage 0.0 (TID 1, 10.110.21.98, executor driver, partition 0, PROCESS_LOCAL, 7004 bytes)
19/11/21 14:28:53 INFO Executor: Running task 1.0 in stage 0.0 (TID 1)
19/11/21 14:28:53 INFO MergeTask: Starting with the merge task!
19/11/21 14:28:53 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 801 ms on 10.110.21.98 (executor driver) (1/2)
19/11/21 14:28:53 INFO Executor: Finished task 1.0 in stage 0.0 (TID 1). 837 bytes result sent to driver
19/11/21 14:28:53 INFO DAGScheduler: ShuffleMapTask finished on driver
19/11/21 14:28:53 INFO TaskSetManager: Finished task 1.0 in stage 0.0 (TID 1) in 22 ms on 10.110.21.98 (executor driver) (2/2)
19/11/21 14:28:53 INFO DAGScheduler: ShuffleMapStage 0 (map at wordCount.scala:16) finished in 0.961 s
19/11/21 14:28:53 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
19/11/21 14:28:53 INFO DAGScheduler: looking for newly runnable stages
```

How do we get a Data/Index file?



How do we get a Data/Index file?

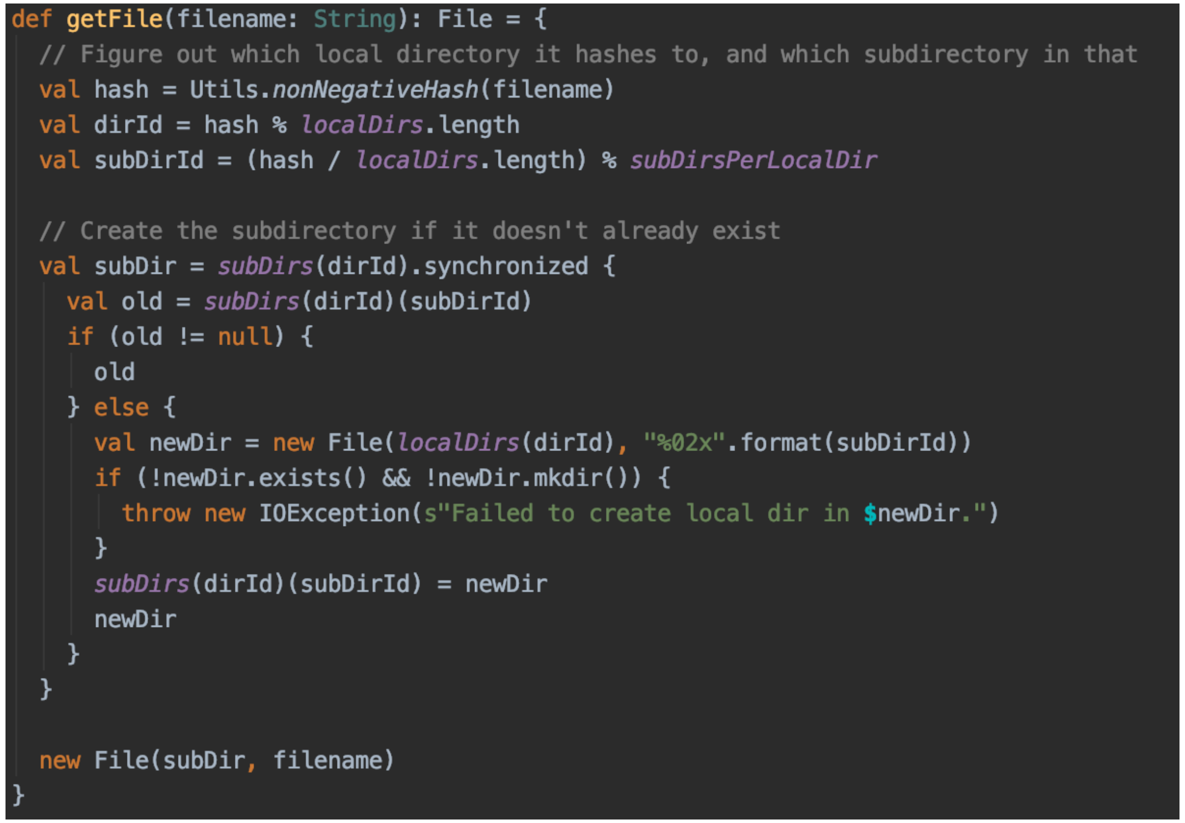
IndexShuffleBlockResolver



```
def getDataFile(shuffleId: Int, mapId: Long): File = {
  blockManager.diskBlockManager.getFile(ShuffleDataBlockId(shuffleId, mapId, NOOP_REDUCE_ID))
}

 def getIndexFile(shuffleId: Int, mapId: Long): File = {
  blockManager.diskBlockManager.getFile(ShuffleIndexBlockId(shuffleId, mapId, NOOP_REDUCE_ID))
}
```

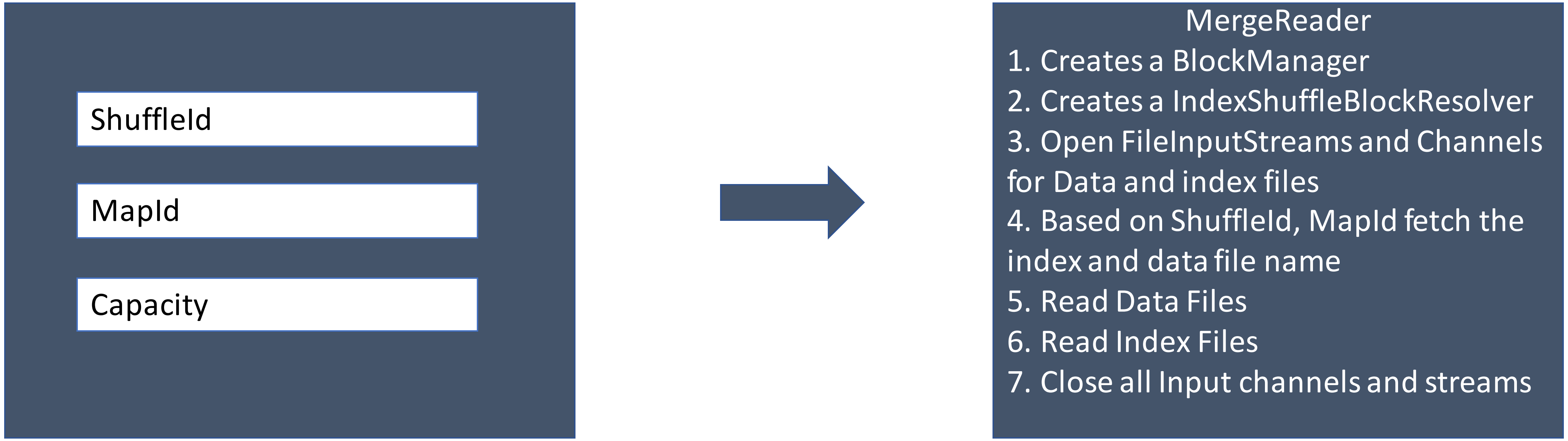
How do we get a Data/Index file?

DiskBlockManager



```scala
def getFile(filename: String): File = {
  // Figure out which local directory it hashes to, and which subdirectory in that
  val hash = Utils.nonNegativeHash(filename)
  val dirId = hash % localDirs.length
  val subDirId = (hash / localDirs.length) % subDirsPerLocalDir

  // Create the subdirectory if it doesn't already exist
  val subDir = subDirs(dirId).synchronized {
    val old = subDirs(dirId)(subDirId)
    if (old != null) {
      old
    } else {
      val newDir = new File(localDirs(dirId), "%02x".format(subDirId))
      if (!newDir.exists() && !newDir.mkdir()) {
        throw new IOException(s"Failed to create local dir in $newDir.")
      }
      subDirs(dirId)(subDirId) = newDir
      newDir
    }
  }

  new File(subDir, filename)
}
```

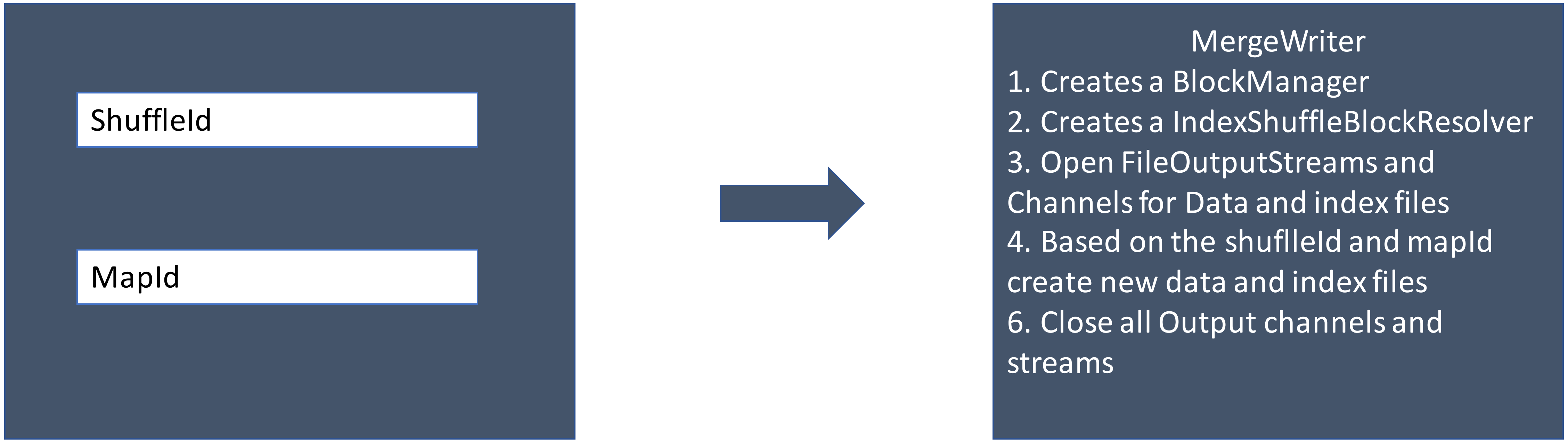
MergeReader



MergeReader

1. Creates a BlockManager
2. Creates a IndexShuffleBlockResolver
3. Open FileInputStreams and Channels for Data and index files
4. Based on ShuffleId, MapId fetch the index and data file name
5. Read Data Files
6. Read Index Files
7. Close all Input channels and streams

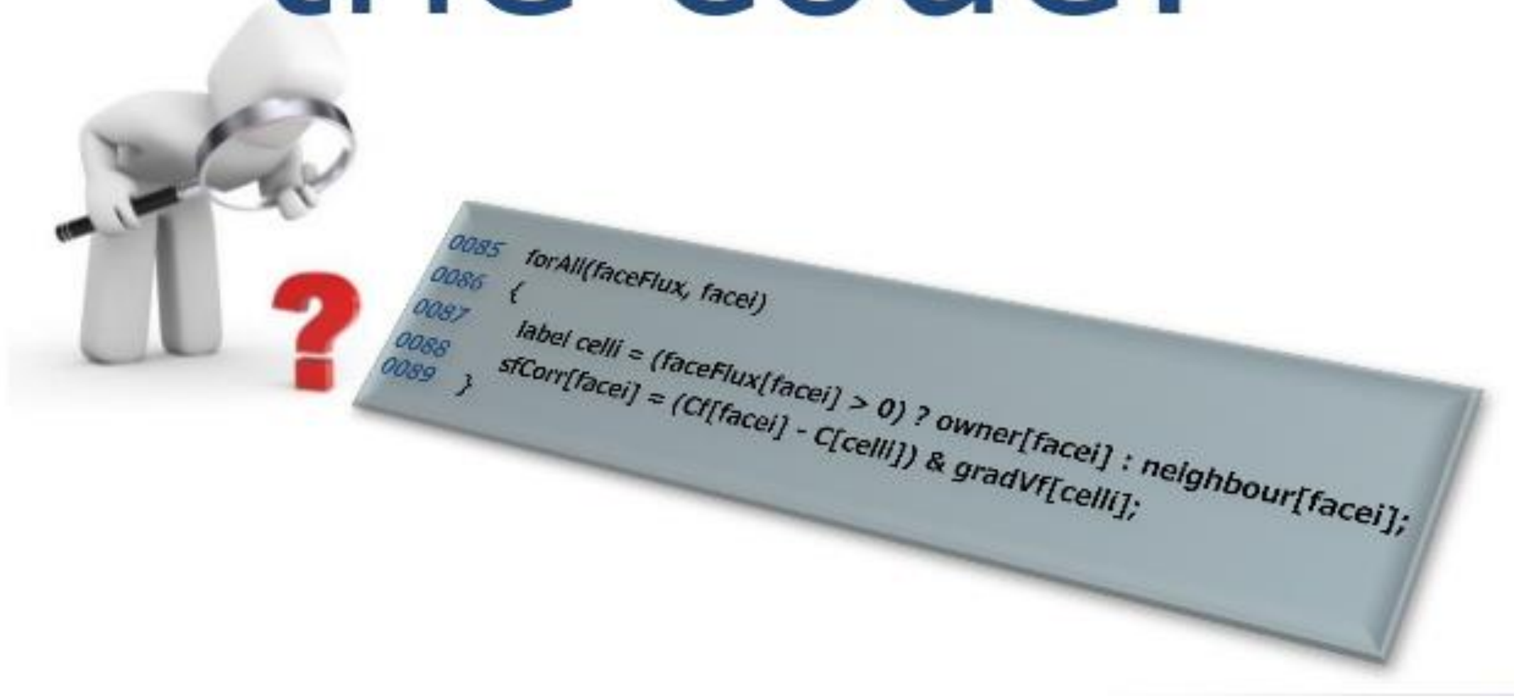
MergeWriter



MergeWriter

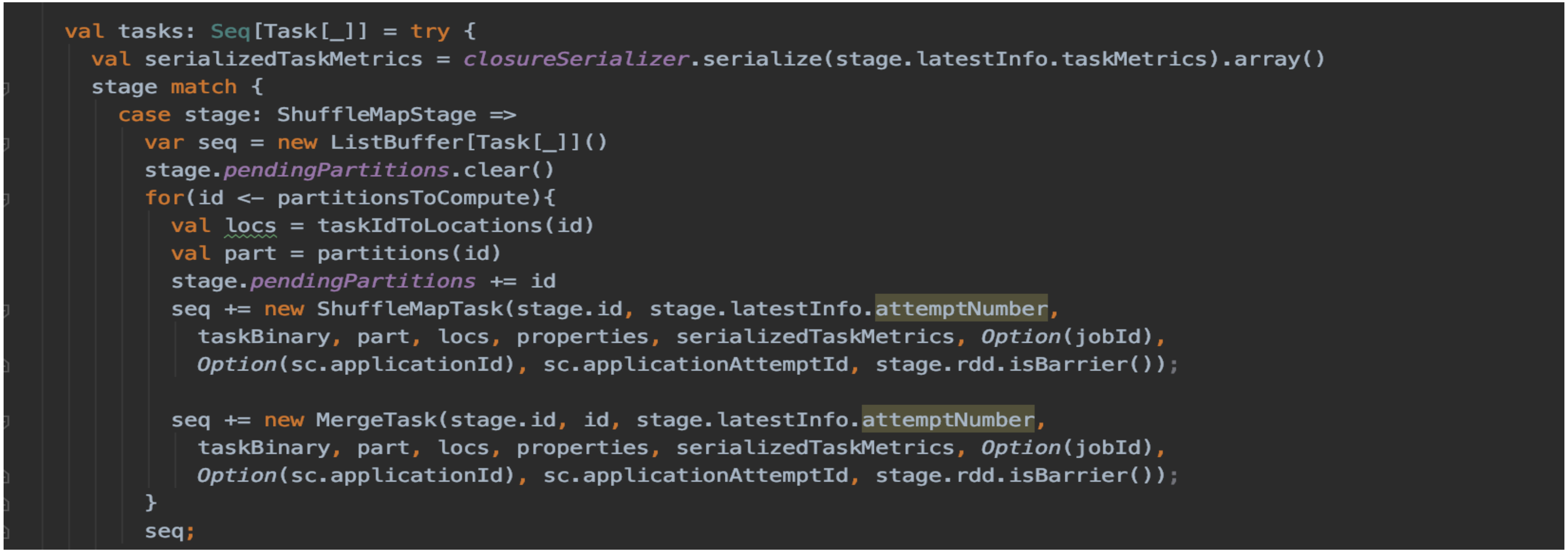
1. Creates a BlockManager
2. Creates a IndexShuffleBlockResolver
3. Open FileOutputStreams and Channels for Data and index files
4. Based on the shuflleId and mapId create new data and index files
6. Close all Output channels and streams

Let’s look into the code!



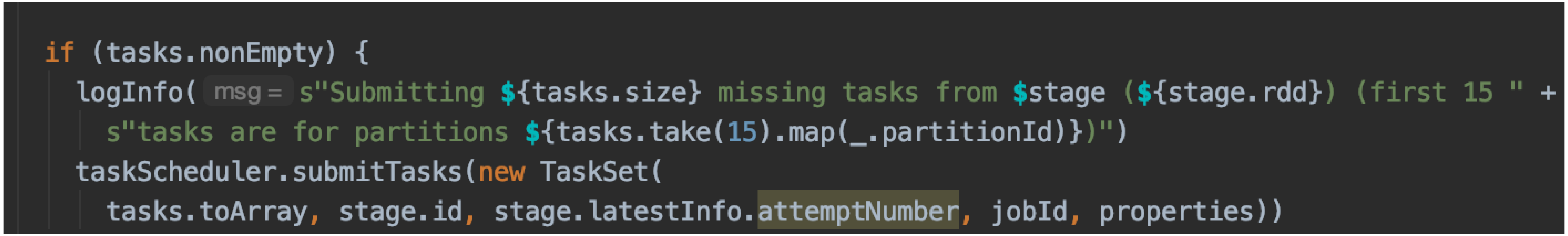
```
forAll(faceFlux, facei)
{
    label celli = (faceFlux[facei] > 0) ? owner[facei] : neighbour[facei];
    sfCorr[facei] = (Cf[facei] - C[celli]) & gradVf[celli];
}
```

Scheduling - DAG Scheduler



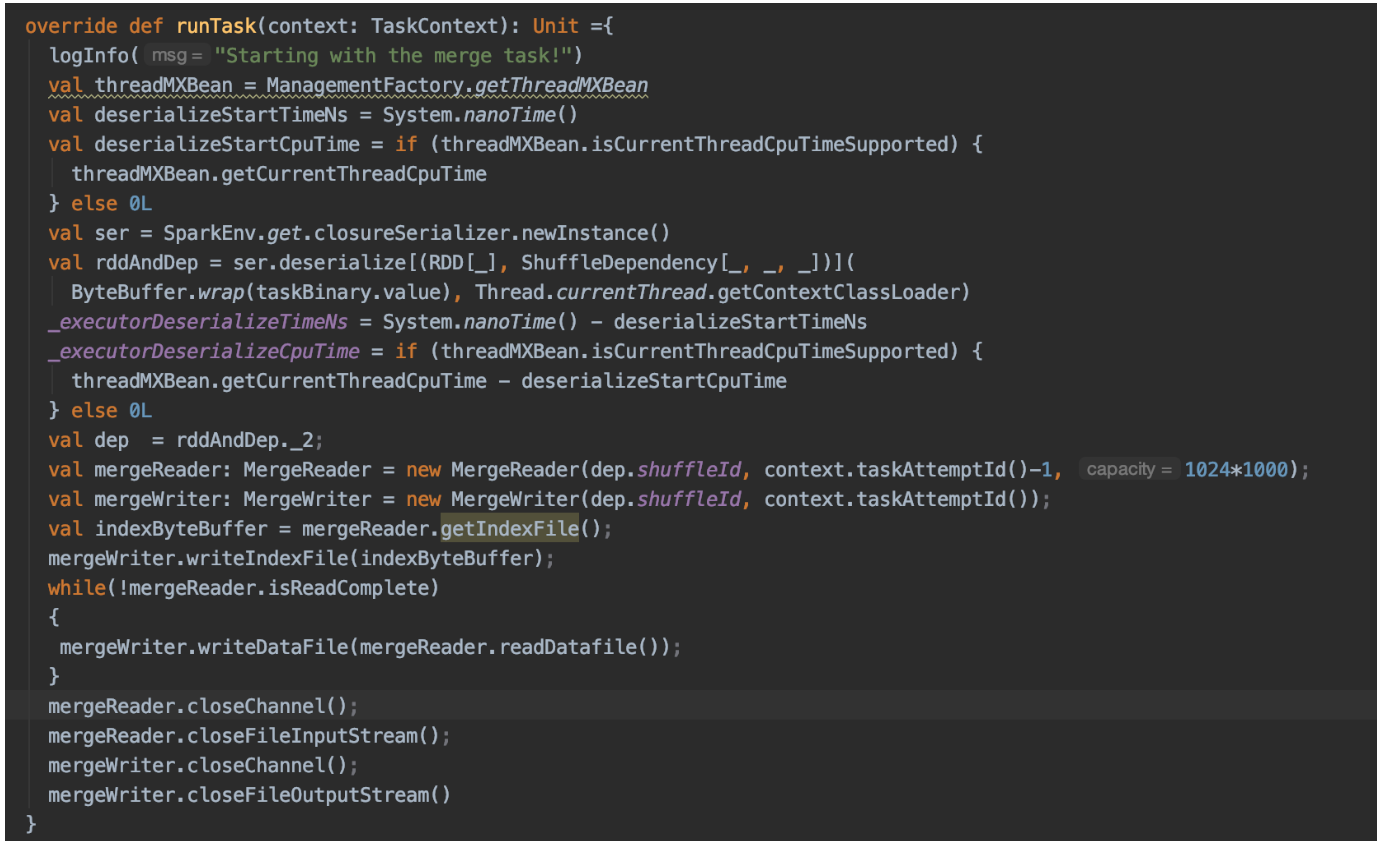
```scala
val tasks: Seq[Task[_]] = try {
  val serializedTaskMetrics = closureSerializer.serialize(stage.latestInfo.taskMetrics).array()
  stage match {
    case stage: ShuffleMapStage =>
      var seq = new ListBuffer[Task[_]]()
      stage.pendingPartitions.clear()
      for(id <- partitionsToCompute){
        val locs = taskIdToLocations(id)
        val part = partitions(id)
        stage.pendingPartitions += id
        seq += new ShuffleMapTask(stage.id, stage.latestInfo.attemptNumber,
          taskBinary, part, locs, properties, serializedTaskMetrics, Option(jobId),
          Option(sc.applicationId), sc.applicationAttemptId, stage.rdd.isBarrier());

        seq += new MergeTask(stage.id, id, stage.latestInfo.attemptNumber,
          taskBinary, part, locs, properties, serializedTaskMetrics, Option(jobId),
          Option(sc.applicationId), sc.applicationAttemptId, stage.rdd.isBarrier());
      }
      seq;
```

Parallel Tasks Scheduling



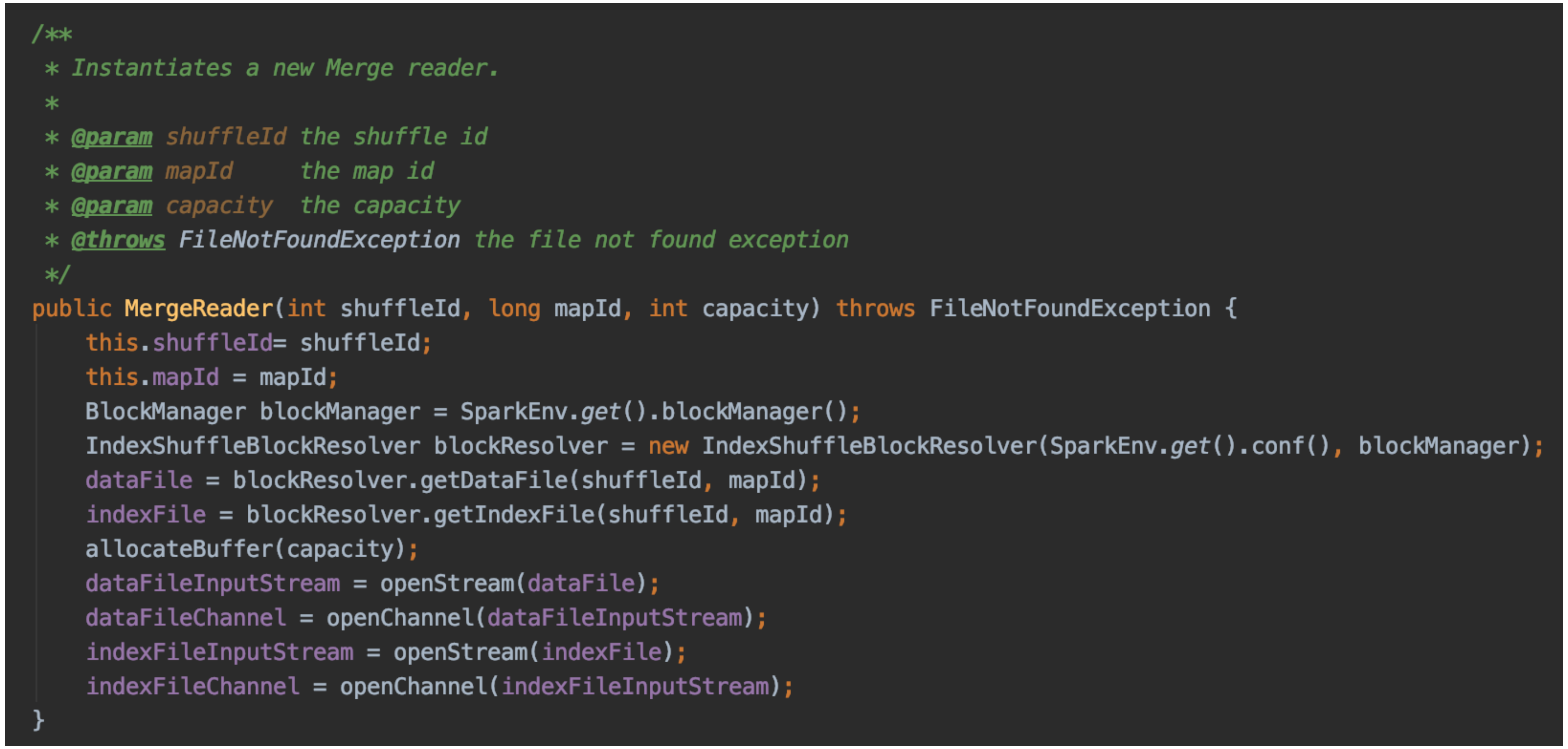
```
if (tasks.nonEmpty) {
  logInfo( msg = s"Submitting ${tasks.size} missing tasks from $stage (${stage.rdd}) (first 15 " +
    s"tasks are for partitions ${tasks.take(15).map(_.partitionId)})")
  taskScheduler.submitTasks(new TaskSet(
    tasks.toArray, stage.id, stage.latestInfo.attemptNumber, jobId, properties))
```

MergeTask



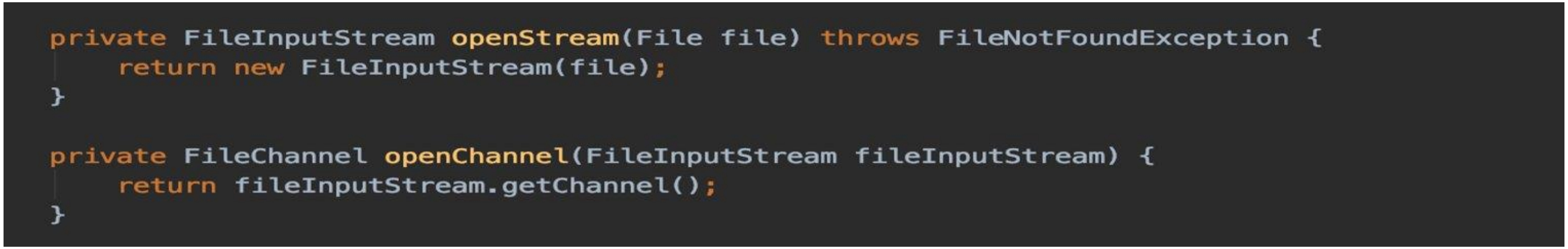
```
override def runTask(context: TaskContext): Unit ={
  logInfo( msg = "Starting with the merge task!")
  val threadMXBean = ManagementFactory.getThreadMXBean
  val deserializeStartTimeNs = System.nanoTime()
  val deserializeStartCpuTime = if (threadMXBean.isCurrentThreadCpuTimeSupported) {
    threadMXBean.getCurrentThreadCpuTime
  } else 0L
  val ser = SparkEnv.get.closureSerializer.newInstance()
  val rddAndDep = ser.deserialize[(RDD[_], ShuffleDependency[_, _, _])](
    ByteBuffer.wrap(taskBinary.value), Thread.currentThread.getContextClassLoader)
  _executorDeserializeTimeNs = System.nanoTime() - deserializeStartTimeNs
  _executorDeserializeCpuTime = if (threadMXBean.isCurrentThreadCpuTimeSupported) {
    threadMXBean.getCurrentThreadCpuTime - deserializeStartCpuTime
  } else 0L
  val dep  = rddAndDep._2;
  val mergeReader: MergeReader = new MergeReader(dep.shuffleId, context.taskAttemptId()-1,  capacity = 1024*1000);
  val mergeWriter: MergeWriter = new MergeWriter(dep.shuffleId, context.taskAttemptId());
  val indexByteBuffer = mergeReader.getIndexFile();
  mergeWriter.writeIndexFile(indexByteBuffer);
  while(!mergeReader.isReadComplete)
  {
   mergeWriter.writeDataFile(mergeReader.readDatafile());
  }
  mergeReader.closeChannel();
  mergeReader.closeFileInputStream();
  mergeWriter.closeChannel();
  mergeWriter.closeFileOutputStream()
}
```

MergeReader

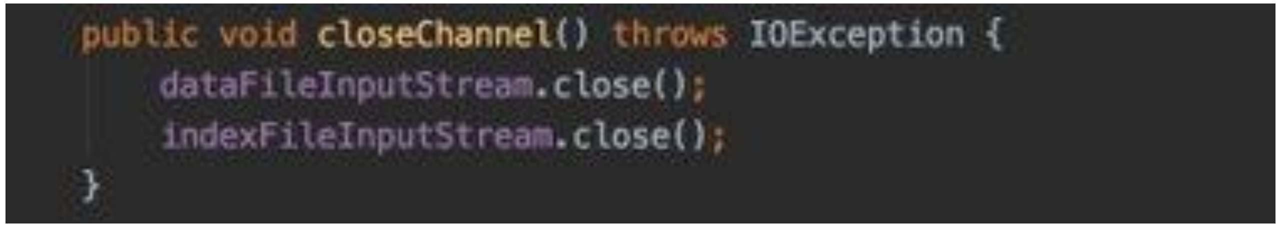


```java
/**
 * Instantiates a new Merge reader.
 *
 * @param shuffleId the shuffle id
 * @param mapId     the map id
 * @param capacity  the capacity
 * @throws FileNotFoundException the file not found exception
 */
public MergeReader(int shuffleId, long mapId, int capacity) throws FileNotFoundException {
    this.shuffleId= shuffleId;
    this.mapId = mapId;
    BlockManager blockManager = SparkEnv.get().blockManager();
    IndexShuffleBlockResolver blockResolver = new IndexShuffleBlockResolver(SparkEnv.get().conf(), blockManager);
    dataFile = blockResolver.getDataFile(shuffleId, mapId);
    indexFile = blockResolver.getIndexFile(shuffleId, mapId);
    allocateBuffer(capacity);
    dataFileInputStream = openStream(dataFile);
    dataFileChannel = openChannel(dataFileInputStream);
    indexFileInputStream = openStream(indexFile);
    indexFileChannel = openChannel(indexFileInputStream);
}
```

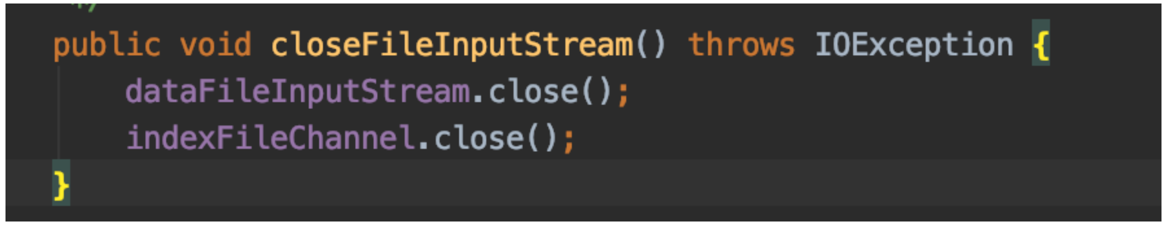
MergeReader



```java
private FileInputStream openStream(File file) throws FileNotFoundException {
    return new FileInputStream(file);
}

private FileChannel openChannel(FileInputStream fileInputStream) {
    return fileInputStream.getChannel();
}
```

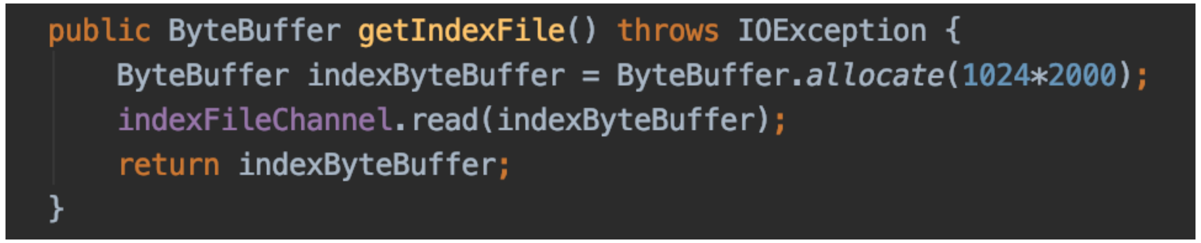


```java
public void closeChannel() throws IOException {
    dataFileInputStream.close();
    indexFileInputStream.close();
}
```

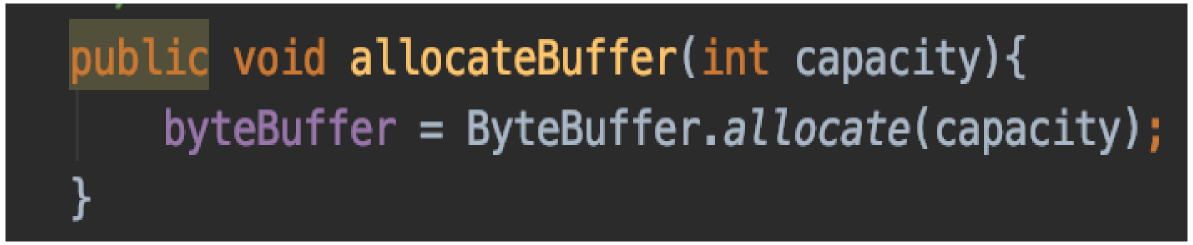


```java
public void closeFileInputStream() throws IOException {
    dataFileInputStream.close();
    indexFileChannel.close();
}
```

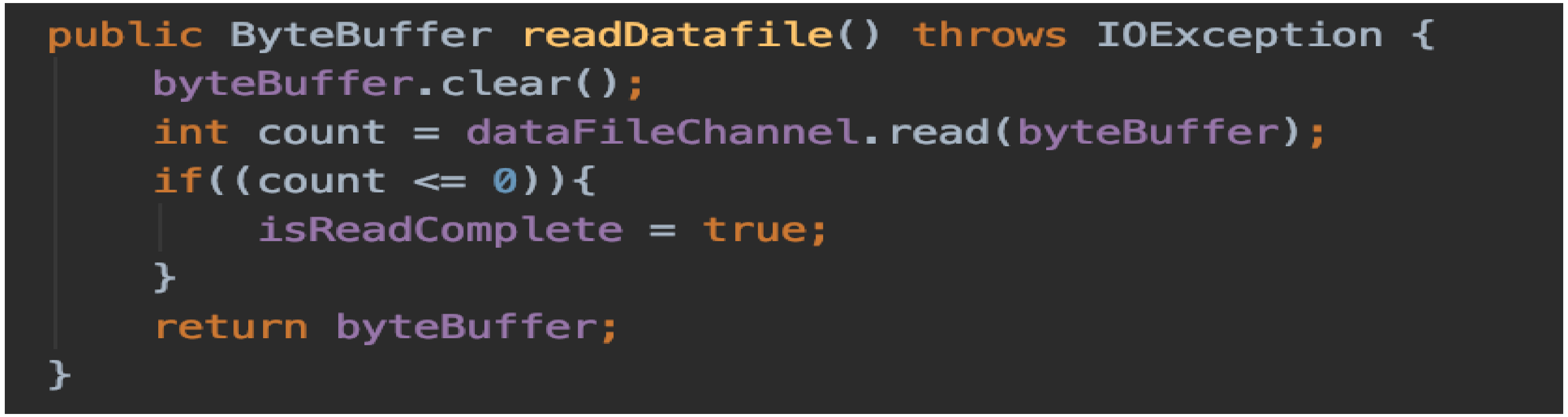
MergeReader



```java
public ByteBuffer getIndexFile() throws IOException {
    ByteBuffer indexByteBuffer = ByteBuffer.allocate(1024*2000);
    indexFileChannel.read(indexByteBuffer);
    return indexByteBuffer;
}
```

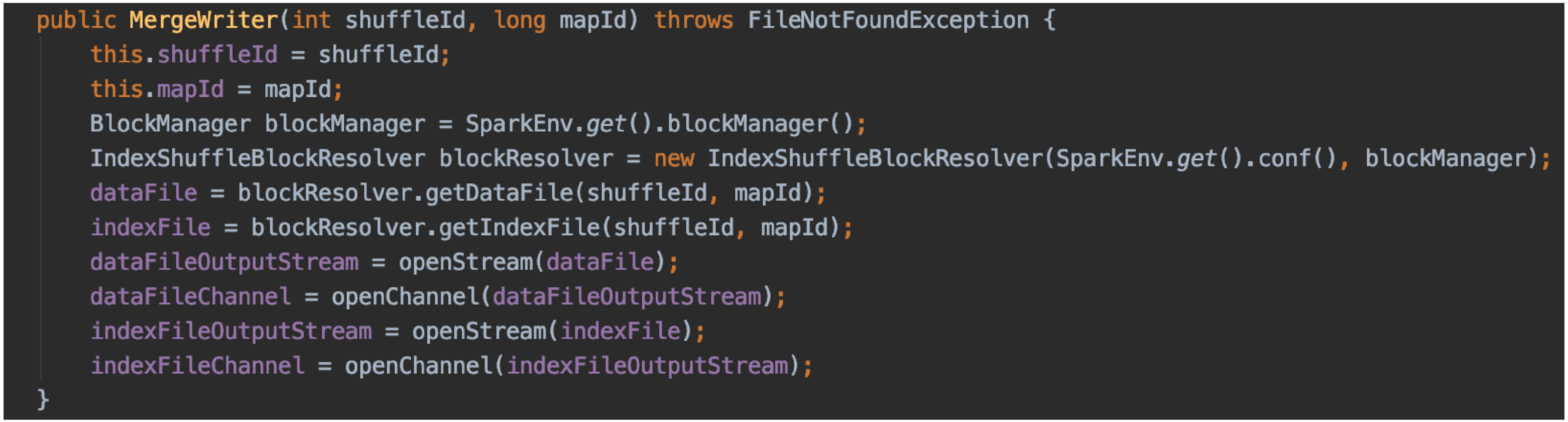


```java
public void allocateBuffer(int capacity){
    byteBuffer = ByteBuffer.allocate(capacity);
}
```



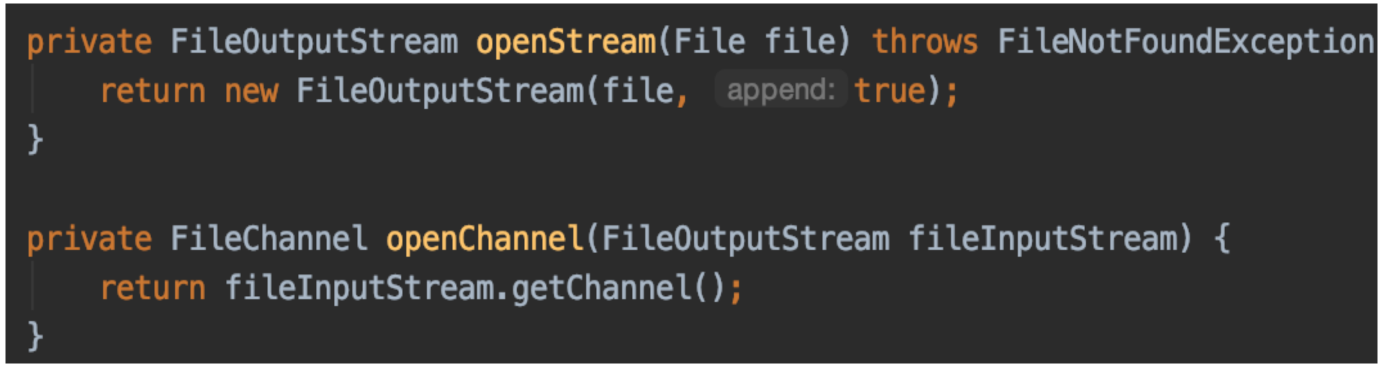
```java
public ByteBuffer readDatafile() throws IOException {
    byteBuffer.clear();
    int count = dataFileChannel.read(byteBuffer);
    if((count <= 0)){
        isReadComplete = true;
    }
    return byteBuffer;
}
```

MergeWriter

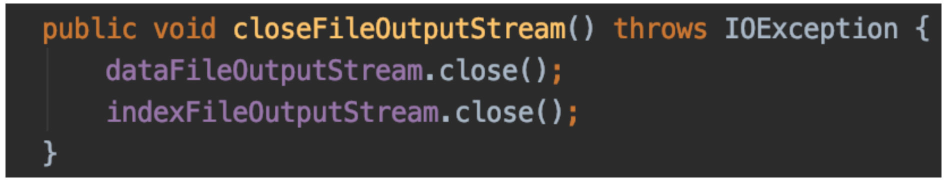


```java
public MergeWriter(int shuffleId, long mapId) throws FileNotFoundException {
    this.shuffleId = shuffleId;
    this.mapId = mapId;
    BlockManager blockManager = SparkEnv.get().blockManager();
    IndexShuffleBlockResolver blockResolver = new IndexShuffleBlockResolver(SparkEnv.get().conf(), blockManager);
    dataFile = blockResolver.getDataFile(shuffleId, mapId);
    indexFile = blockResolver.getIndexFile(shuffleId, mapId);
    dataFileOutputStream = openStream(dataFile);
    dataFileChannel = openChannel(dataFileOutputStream);
    indexFileOutputStream = openStream(indexFile);
    indexFileChannel = openChannel(indexFileOutputStream);
}
```

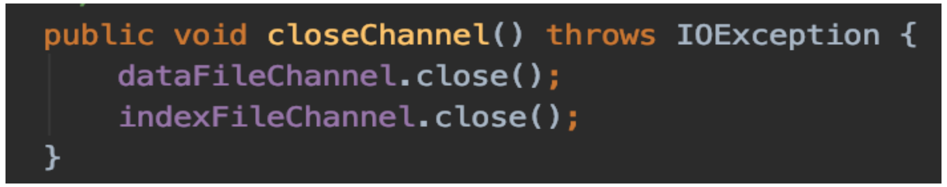
MergeWriter



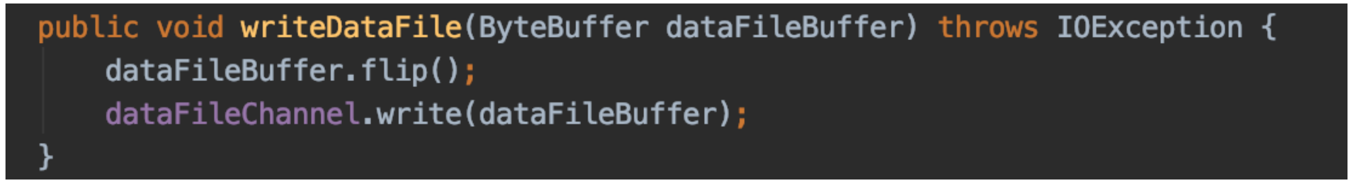
```java
private FileOutputStream openStream(File file) throws FileNotFoundException
    return new FileOutputStream(file,  append: true);
}

private FileChannel openChannel(FileOutputStream fileInputStream) {
    return fileInputStream.getChannel();
}
```



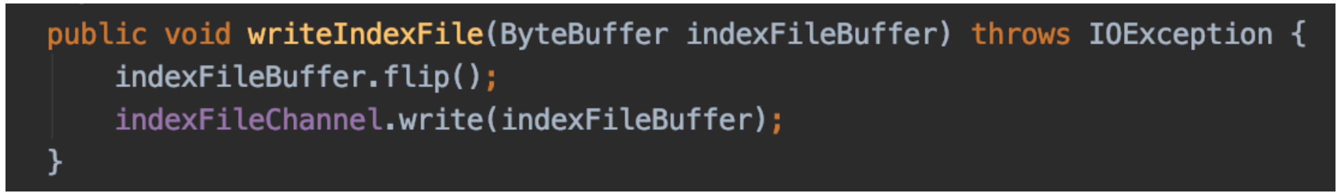
```java
public void closeFileOutputStream() throws IOException {
    dataFileOutputStream.close();
    indexFileOutputStream.close();
}
```



```java
public void closeChannel() throws IOException {
     dataFileChannel.close();
     indexFileChannel.close();
}
```

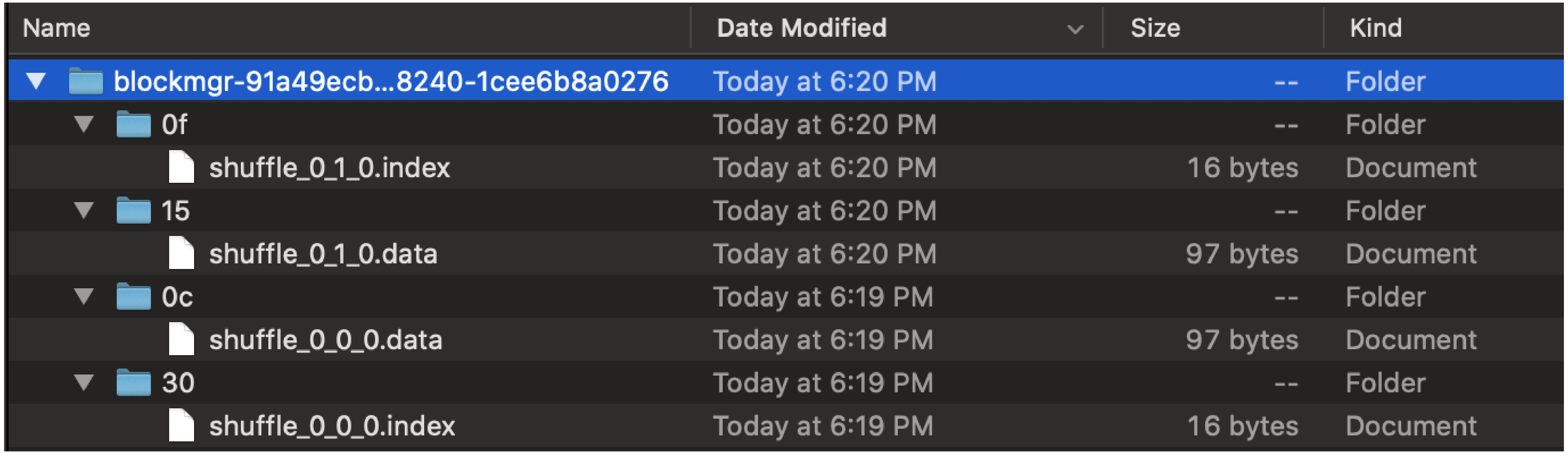


```java
public void writeDataFile(ByteBuffer dataFileBuffer) throws IOException {
    dataFileBuffer.flip();
    dataFileChannel.write(dataFileBuffer);
}
```



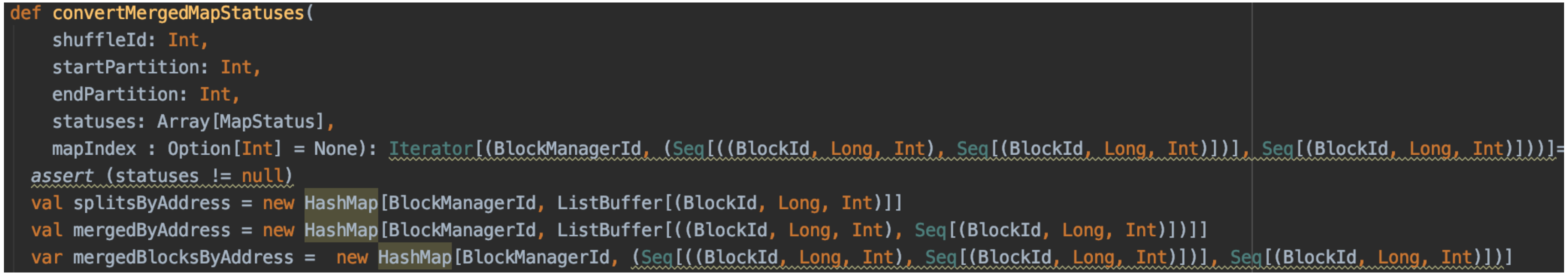
```java
public void writeIndexFile(ByteBuffer indexFileBuffer) throws IOException {
    indexFileBuffer.flip();
    indexFileChannel.write(indexFileBuffer);
}
```

Result

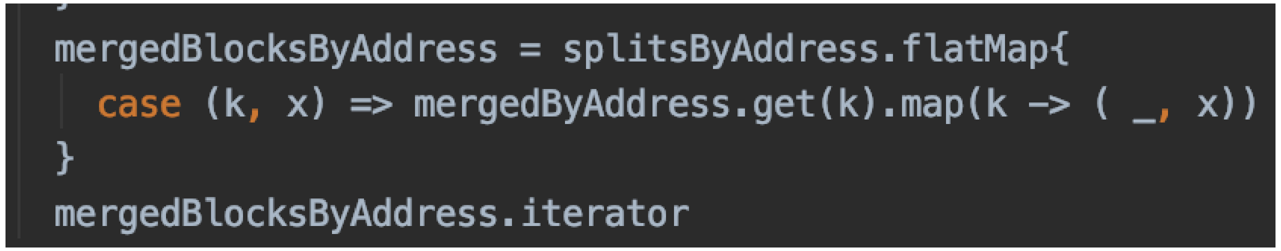


Name	Date Modified	Size	Kind
blockmgr-91a49ecb...8240-1cee6b8a0276	Today at 6:20 PM	--	Folder
0f	Today at 6:20 PM	--	Folder
shuffle_0_1_0.index	Today at 6:20 PM	16 bytes	Document
15	Today at 6:20 PM	--	Folder
shuffle_0_1_0.data	Today at 6:20 PM	97 bytes	Document
0c	Today at 6:19 PM	--	Folder
shuffle_0_0_0.data	Today at 6:19 PM	97 bytes	Document
30	Today at 6:19 PM	--	Folder
shuffle_0_0_0.index	Today at 6:19 PM	16 bytes	Document

Refactored ShuffleReader

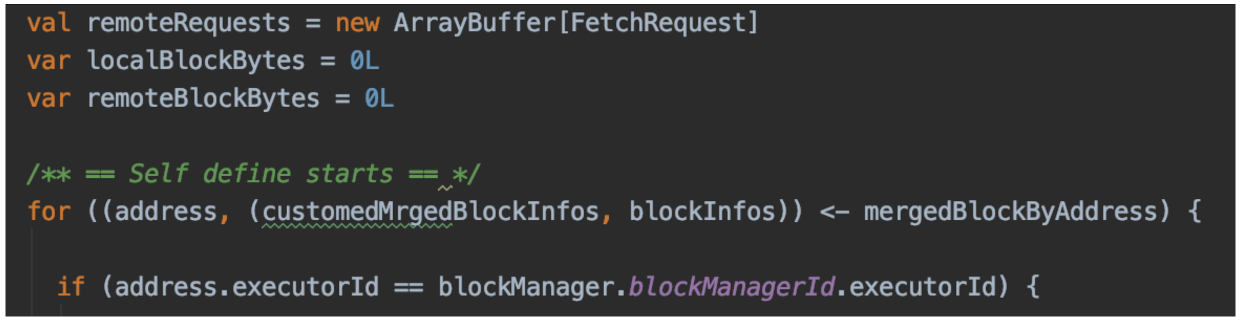


```scala
def convertMergedMapStatuses(
    shuffleId: Int,
    startPartition: Int,
    endPartition: Int,
    statuses: Array[MapStatus],
    mapIndex : Option[Int] = None): Iterator[(BlockManagerId, (Seq[((BlockId, Long, Int), Seq[(BlockId, Long, Int)])], Seq[(BlockId, Long, Int)]))]=
  assert (statuses != null)
  val splitsByAddress = new HashMap[BlockManagerId, ListBuffer[(BlockId, Long, Int)]]
  val mergedByAddress = new HashMap[BlockManagerId, ListBuffer[((BlockId, Long, Int), Seq[(BlockId, Long, Int)])]]
  var mergedBlocksByAddress =  new HashMap[BlockManagerId, (Seq[((BlockId, Long, Int), Seq[(BlockId, Long, Int)])], Seq[(BlockId, Long, Int)])]
```

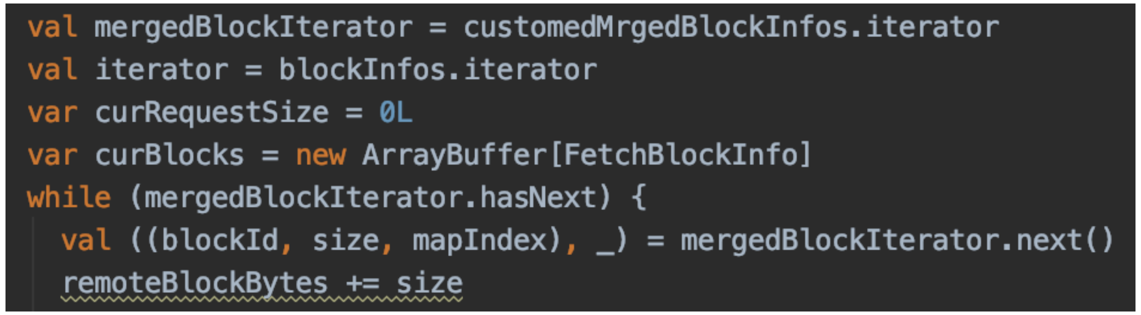


```scala
  mergedBlocksByAddress = splitsByAddress.flatMap{
    case (k, x) => mergedByAddress.get(k).map(k -> ( _, x))
  }
  mergedBlocksByAddress.iterator
```

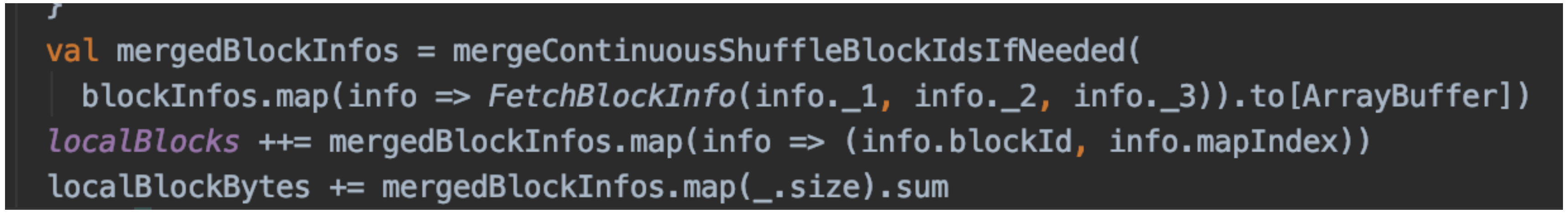
Refactored ShuffleReader



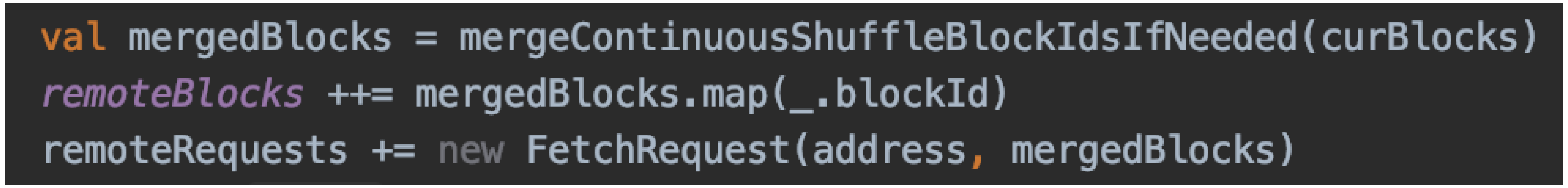
```scala
val remoteRequests = new ArrayBuffer[FetchRequest]
var localBlockBytes = 0L
var remoteBlockBytes = 0L

/** == Self define starts == */
for ((address, (customedMrgedBlockInfos, blockInfos)) <- mergedBlockByAddress) {

  if (address.executorId == blockManager.blockManagerId.executorId) {
```



```scala
val mergedBlockIterator = customedMrgedBlockInfos.iterator
val iterator = blockInfos.iterator
var curRequestSize = 0L
var curBlocks = new ArrayBuffer[FetchBlockInfo]
while (mergedBlockIterator.hasNext) {
  val ((blockId, size, mapIndex), _) = mergedBlockIterator.next()
  remoteBlockBytes += size
```

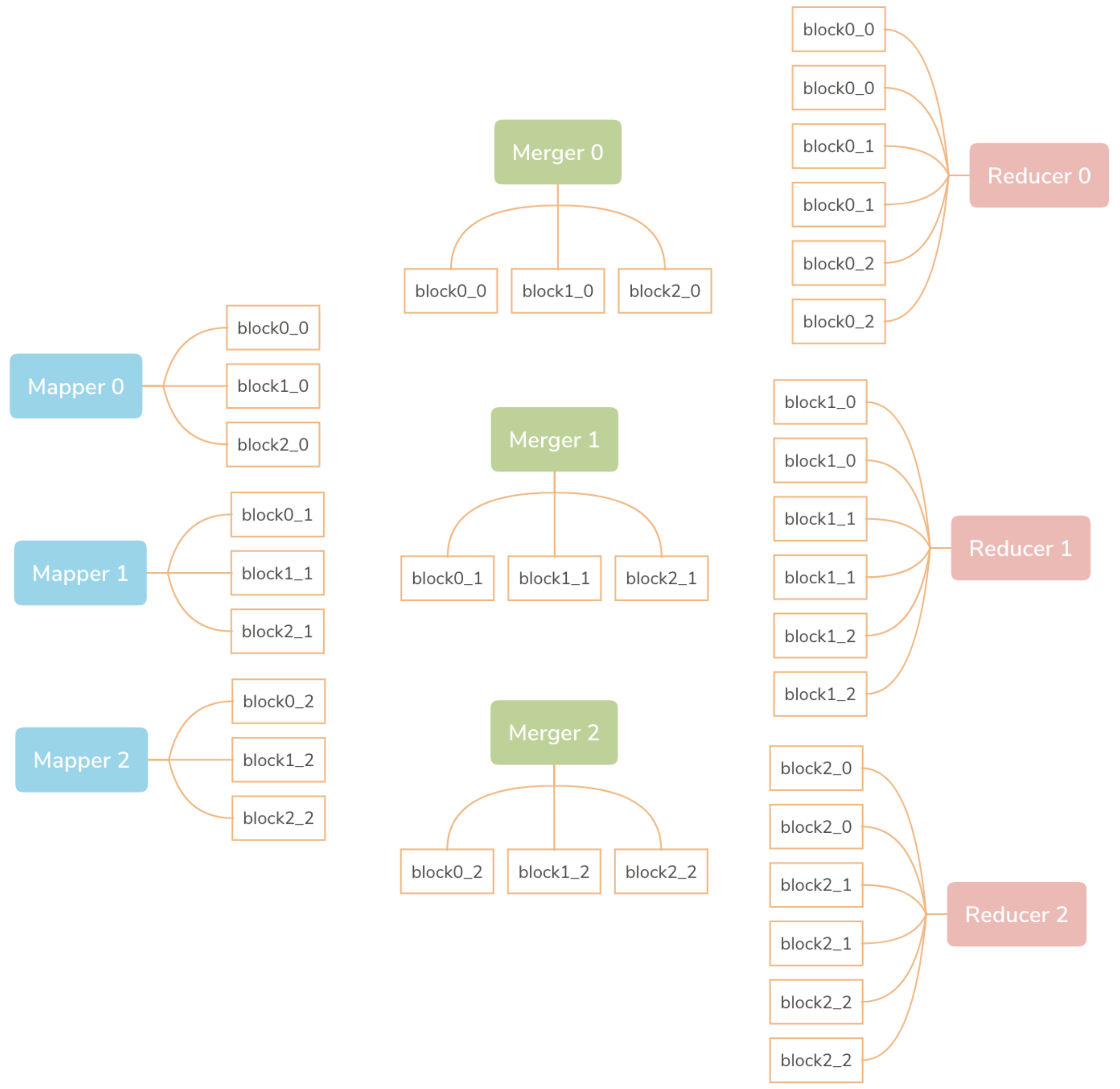


```scala
  }
  val mergedBlockInfos = mergeContinuousShuffleBlockIdsIfNeeded(
    blockInfos.map(info => FetchBlockInfo(info._1, info._2, info._3)).to[ArrayBuffer])
  localBlocks ++= mergedBlockInfos.map(info => (info.blockId, info.mapIndex))
  localBlockBytes += mergedBlockInfos.map(_.size).sum
```

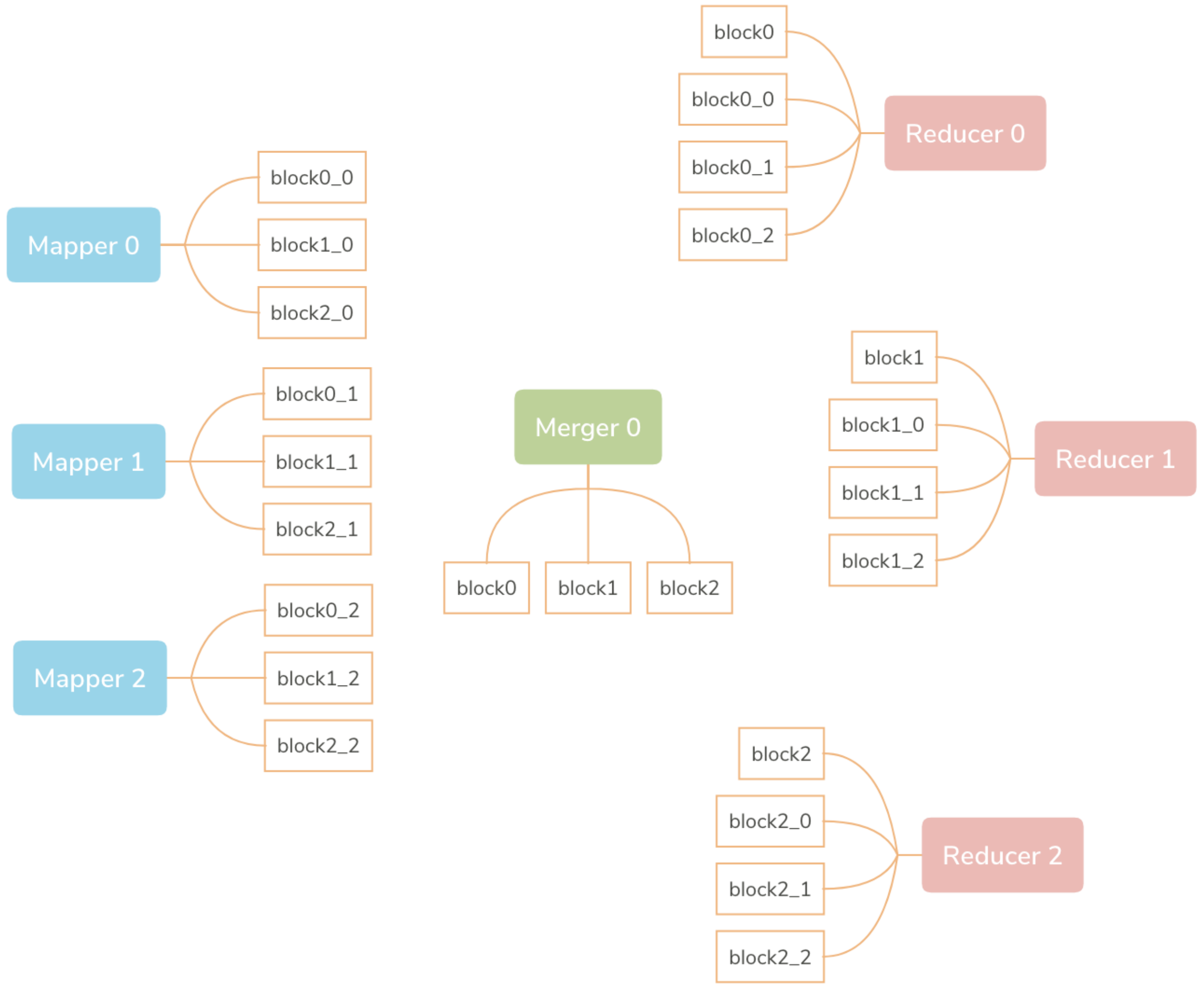


```scala
val mergedBlocks = mergeContinuousShuffleBlockIdsIfNeeded(curBlocks)
remoteBlocks ++= mergedBlocks.map(_.blockId)
remoteRequests += new FetchRequest(address, mergedBlocks)
```

Summary


Mapper 0
block0_0
block1_0
block2_0
Mapper 1
block0_1
block1_1
block2_1
Mapper 2
block0_2
block1_2
block2_2
Merger 0
block0_0
block1_0
block2_0
Merger 1
block0_1
block1_1
block2_1
Merger 2
block0_2
block1_2
block2_2
block0_0
block0_0
block0_1
block0_1
block0_2
block0_2
Reducer 0
block1_0
block1_0
block1_1
block1_1
block1_2
block1_2
Reducer 1
block2_0
block2_0
block2_1
block2_1
block2_2
block2_2
Reducer 2

Improvement



Next Sprint Goals

- Incorporate Merge Algorithm in exsisting flow
- Change mapping form 1 : 1 to N : 1

Any Questions?

Thank You!