Big data with Hadoop

Michael Enudi

Journey through the world of databases and data engineering

HADOOP CORE COMPONENTS

Storage

HDFS

Hadoop Distributed File System

Data distributed across nodes

Natively redundant

Processing

MapReduce v2

YARN

YARN

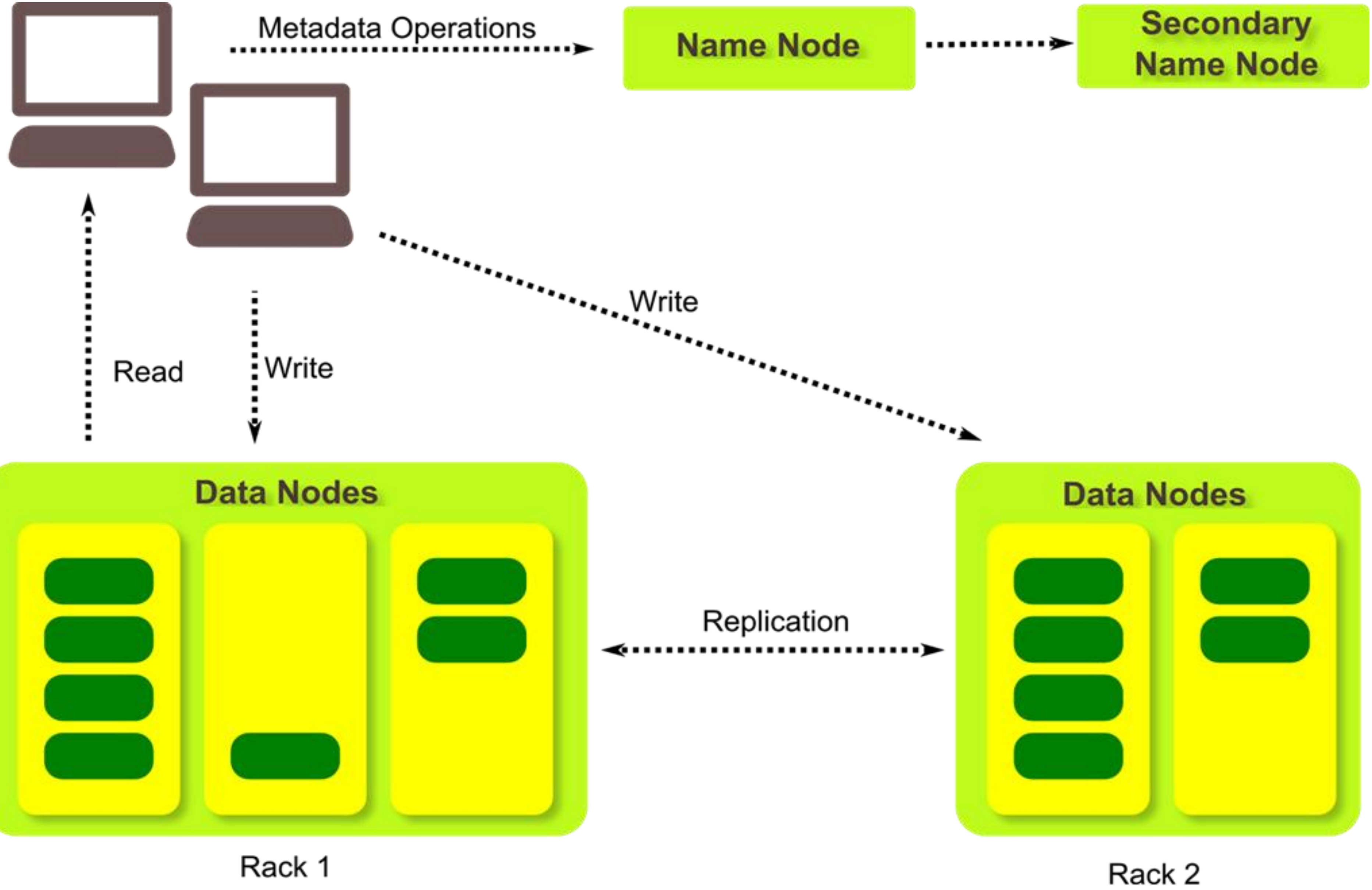
Splits the Job to tasks across nodes

Data Localization

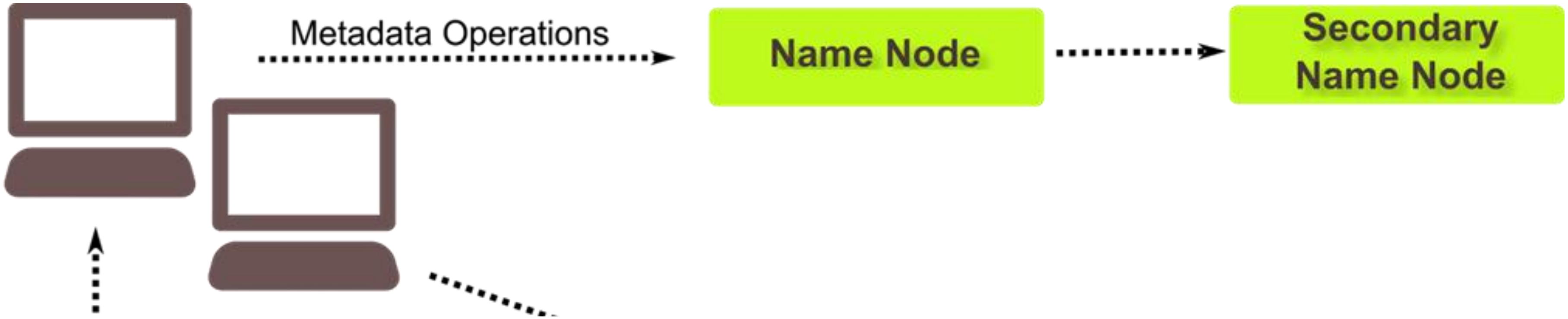
Fault Tolerant

Providesplugin system for more data processing frameworks

HDFS ARCHITECTURE

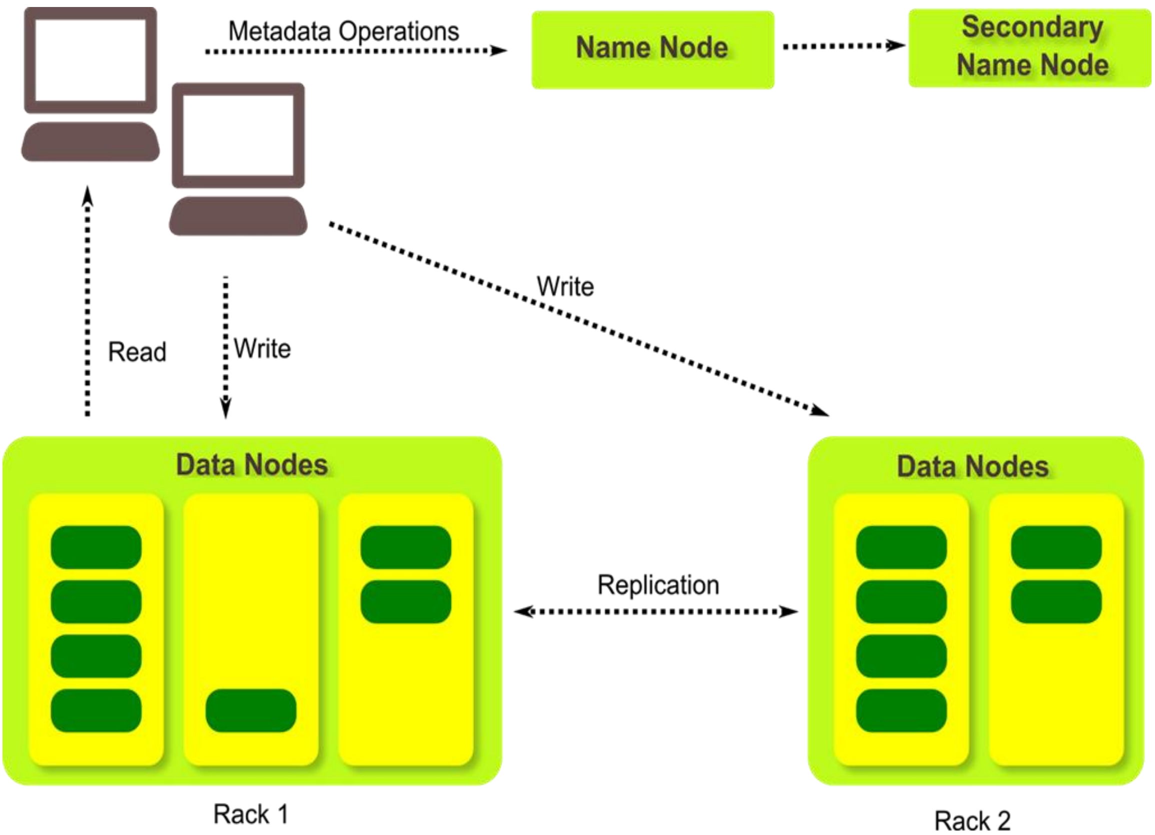


HDFS ARCHITECTURE



- Name Node (Master Node)
  - Keeps track of all active data nodes and the file blocks stored in them
  - Maintains meta data for the files that are stored on the Data Nodes
  - Manages the file system tree and other meta information
  - It is responsible for maintaining namespace image and edit log files
  - Ensures balancing of redundant copies of replicas in the case of failure of a data node.
  - Handles client request for data or metadata operations
  - ….

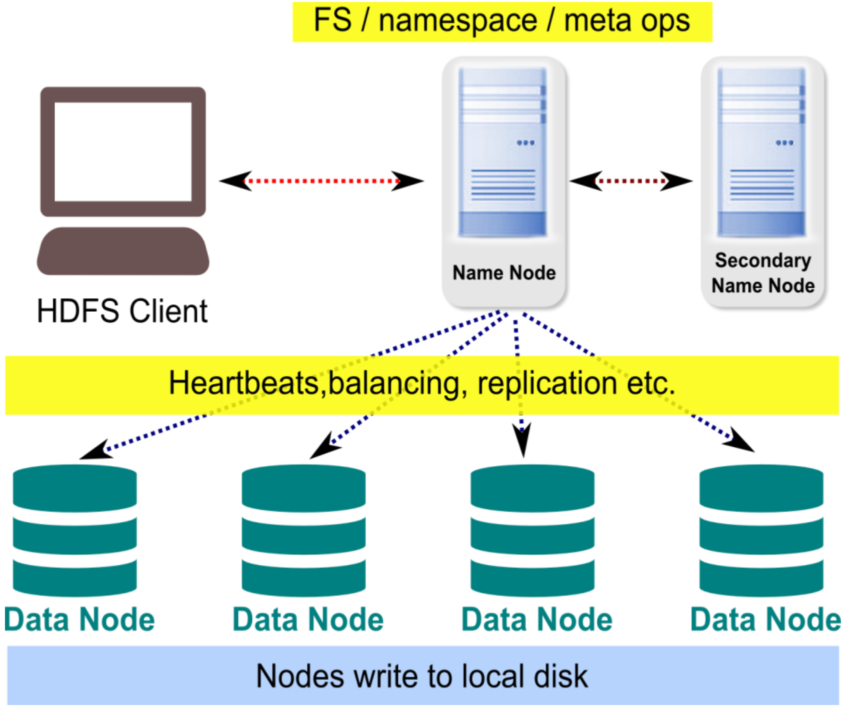
HDFS ARCHITECTURE



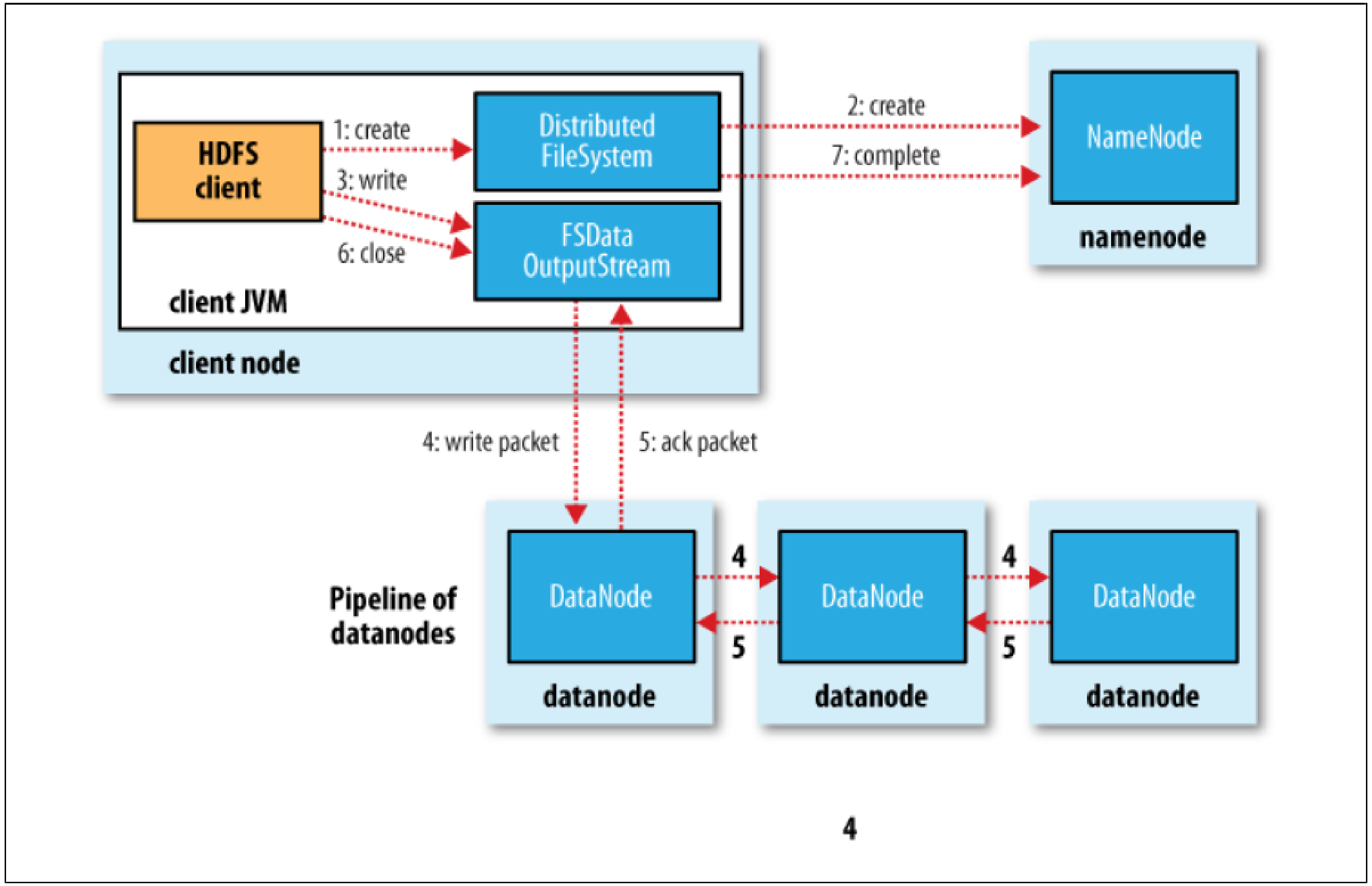
- Data Node (Slave Node)
  - The workhorses of HDFS. They are the block manager services.
  - Deployed on each machine and do the actual storage of file blocks
  - Responsible for serving read and write requests for the clients
  - Store and retrieve blocks when they are told to
  - Send heartbeat request periodically to the NN
  - They also inform NN with the list of blocks they have
  - DN are entirely unaware. They don’t keep state or know the details of the blocks they store. That is all done at the NN.

NAME NODE METADATA

- Meta-data in Memory
  - The entire metadata is in main memory
  - No demand paging of FS meta-data
- Types of Metadata
  - List of files
  - List of Blocks for each file
  - List of Data Node for each block
  - File attributes, e.g. access time, replication factor
- A Transaction Log
  - Records file creations, file deletions, etc.

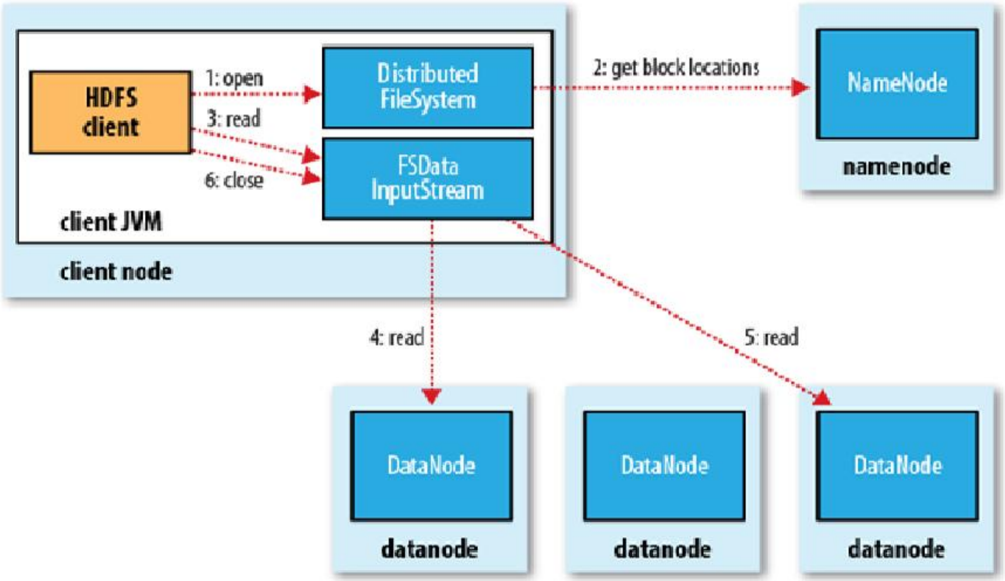

FS / namespace / meta ops
HDFS Client
Name Node
Secondary Name Node
Heartbeats,balancing, replication etc.
Data Node
Data Node
Data Node
Data Node
Nodes write to local disk

HDFS - ANATOMY OF FILE WRITE



- Client creates a new file by making a request through the file system internals to the NN with the file to be created.
- A decision is made at NN as to how many blocks the file can be split into.
- For Each Block, NN returns the list of data nodes to host its replicas
- The data is pipelined from the Client to the chosen DN with other information like the number of replicas as well as the nodes that will store the replicas.
- The DN confirms the creation of block replica to NN as well the result of other utilities like the block checksum
- The DN also passes the create request for the blocks for the replicas to the other DNs.

HDFS - ANATOMY OF FILE READ

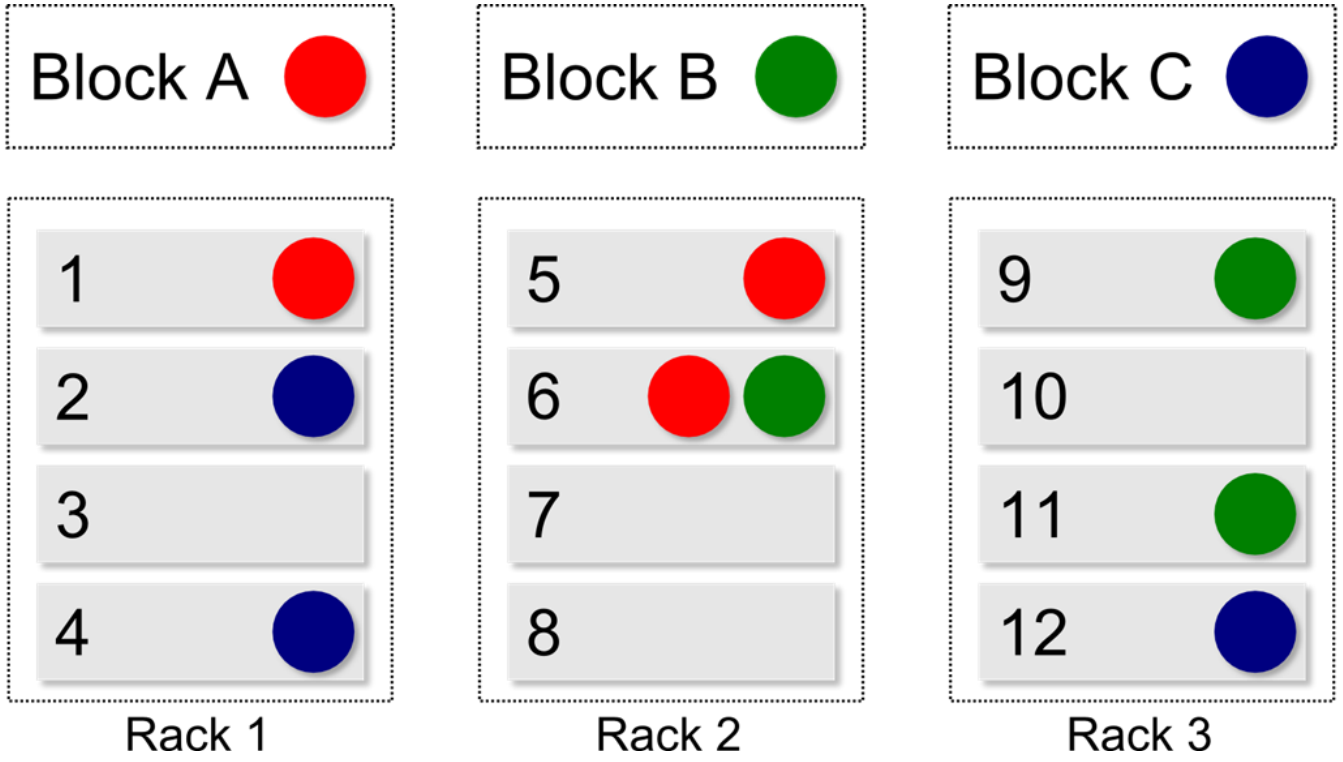


- Connects to NN
- The NN is request to give the list of DNs that host the file blocks
- The blocks for the files are then fetched from the DN without contacting NN again.
- Along with the data, checksum is also shipped for verifying the data integrity. Why?
  - If the replica is corrupt client intimates NN, and try to get the data from other data node
- The file blocks are then put to together which is served to the client.

Note:

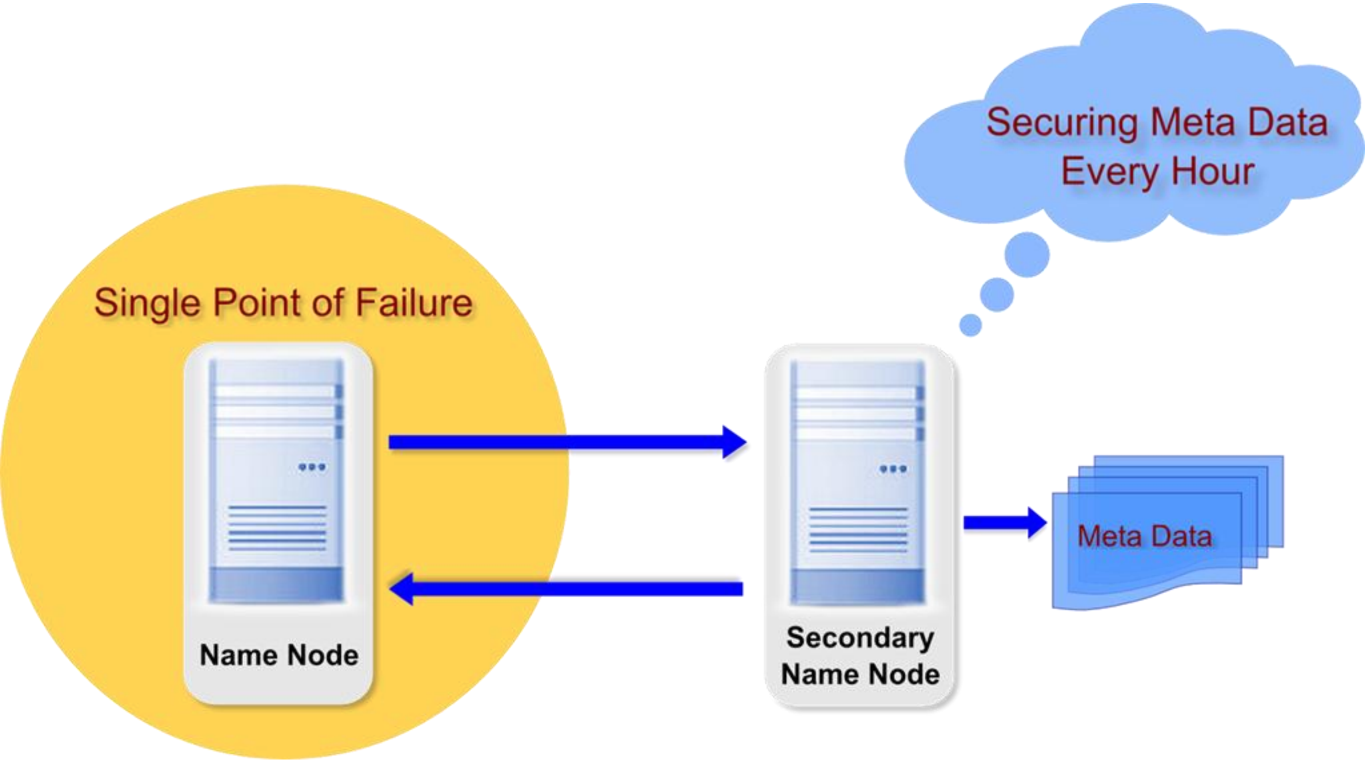
- Information requested from the NN include:
  - List of blocks
  - Location of each block from name node
  - Location is ordered by the distance from the reader

HDFS - REPLICATION AND RACK AWARENESS



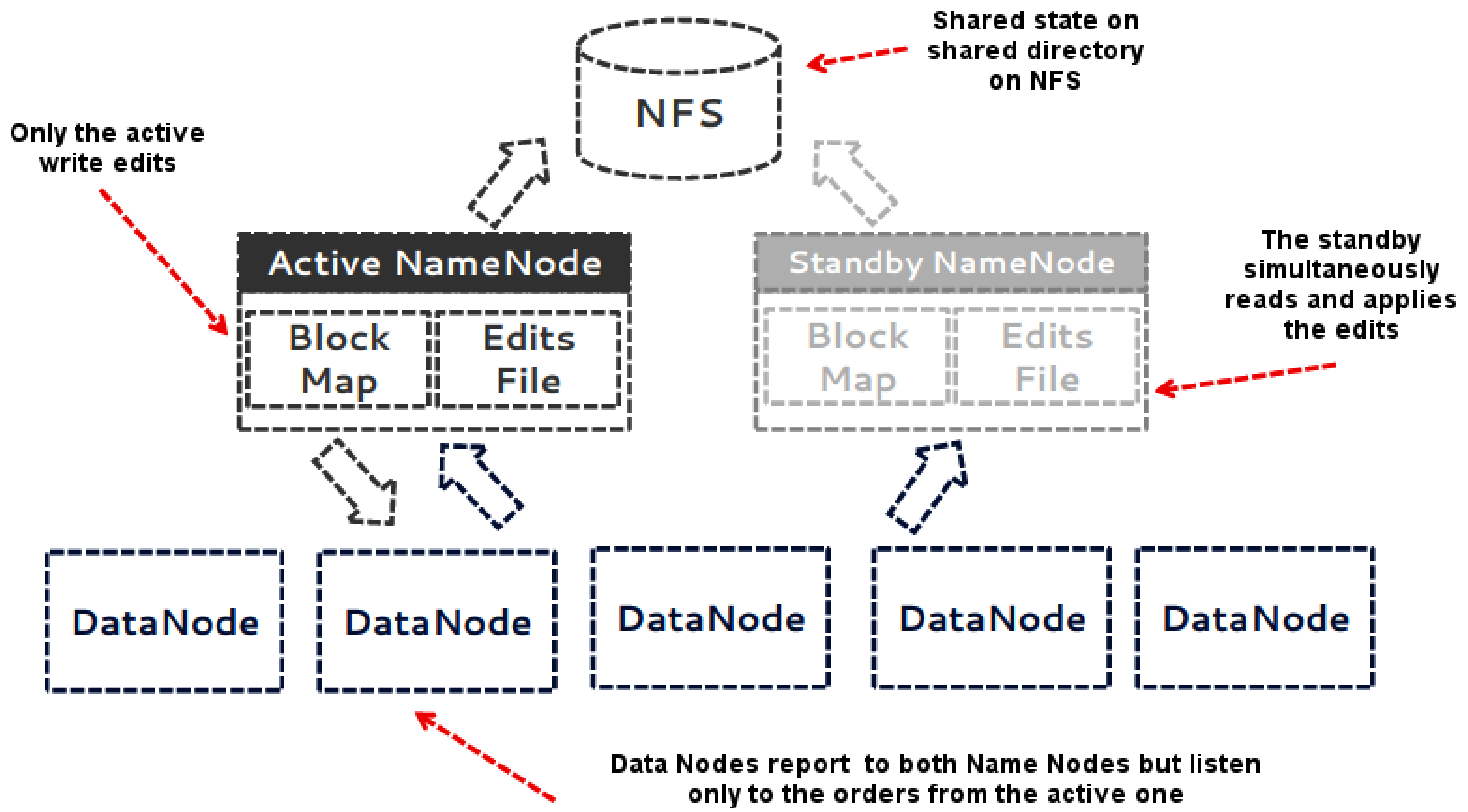
- **Replica placement policy**
  - 1st replica on one node in the local rack
  - 2nd replica on different node in different rack
  - 3rd replica on different node but same rack as 2nd replica
- **Improves read latency**
  - Block size and replication can be configured per file
  - Application can specify the replication of file

SECONDARY NAMENODE



- Not a hot standby for the Name Node
- Connects to Name Node every hour*
- Housekeeping, backup of Name Node metadata
- Saved metadata can build a failed Name Node
- Its primary role is to periodically merge the namespace image and edit log size within a limit
- Secondary Name Node usually runs on separate physical machine because it requires as much memory as Name Node to perform the merge

MULTI-MASTER HDFS ARCHITECTURE



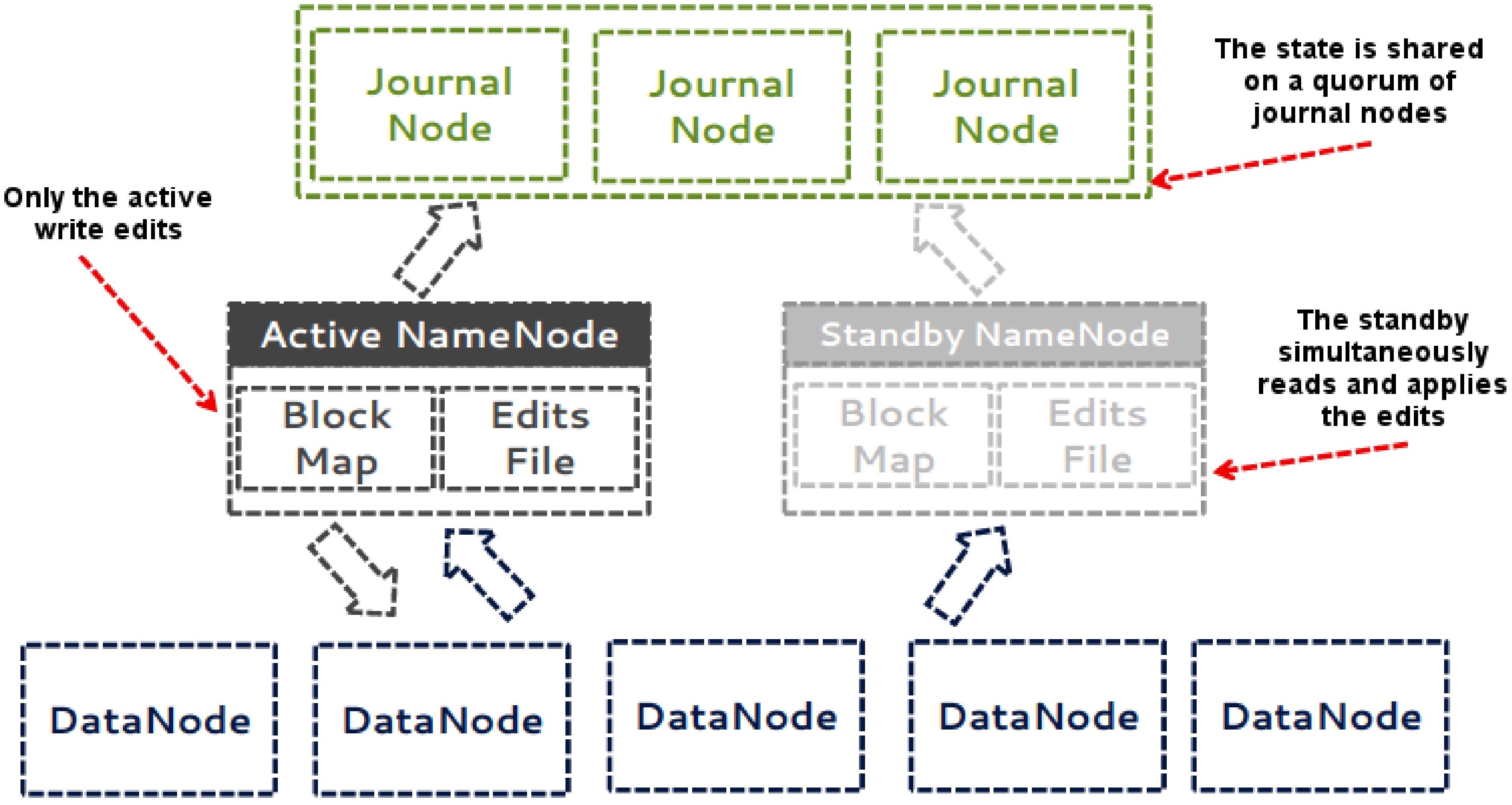
Shared state on shared directory on NFS

Only the active write edits

The standby simultaneously reads and applies the edits

Data Nodes report to both Name Nodes but listen only to the orders from the active one

HDFS ARCHITECTURE: QUORUM BASED STORAGE



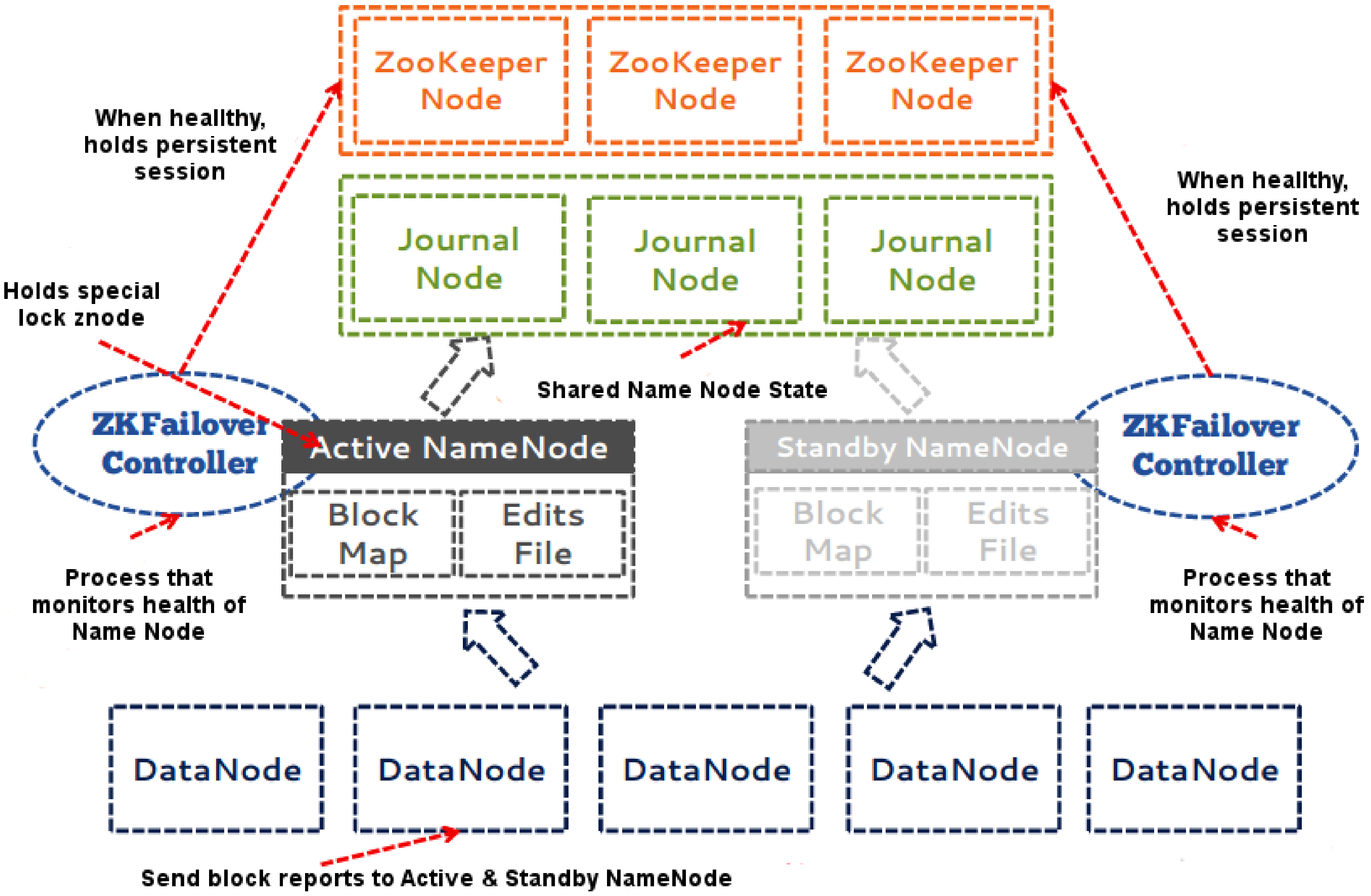
The state is shared on a quorum of journal nodes

Only the active write edits

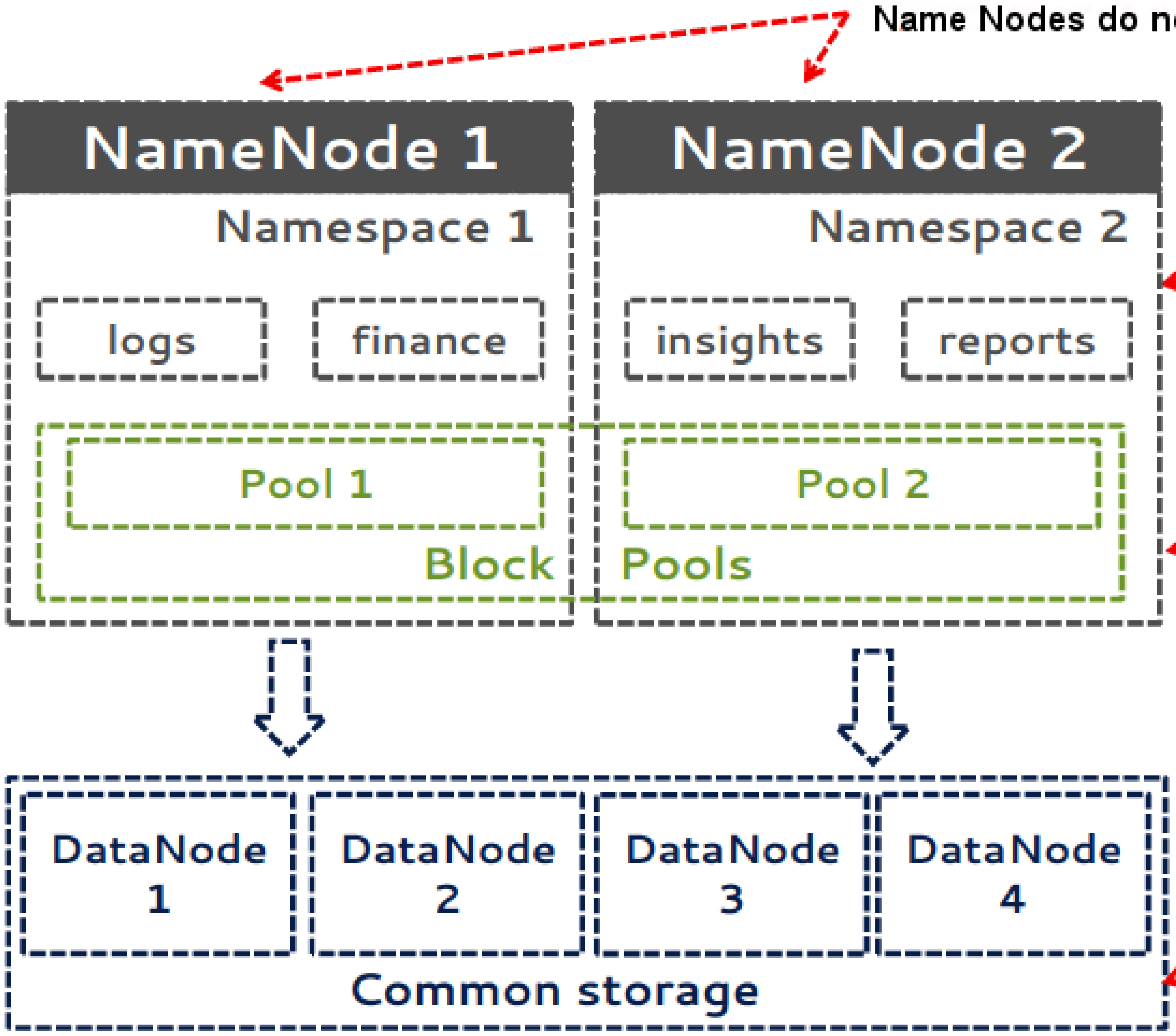
The standby simultaneously reads and applies the edits

Data Nodes report to both Name Nodes but listen only to the orders from the active one

HDFS ARCHITECTURE: HIGH AVAILABILITY



HDFS FEDERATION



Name Nodes do not talk to each other

Name Nodes manages only slice of namespace

Block Storage as generic storage service

Data Nodes can store blocks manually by any Name Node

HADOOP CORE COMPONENTS

Storage

HDFS

Hadoop Distributed File System

Data distributed across nodes

Natively redundant

Processing

MapReduce v2

YARN

YARN

Splits the Job to tasks across nodes

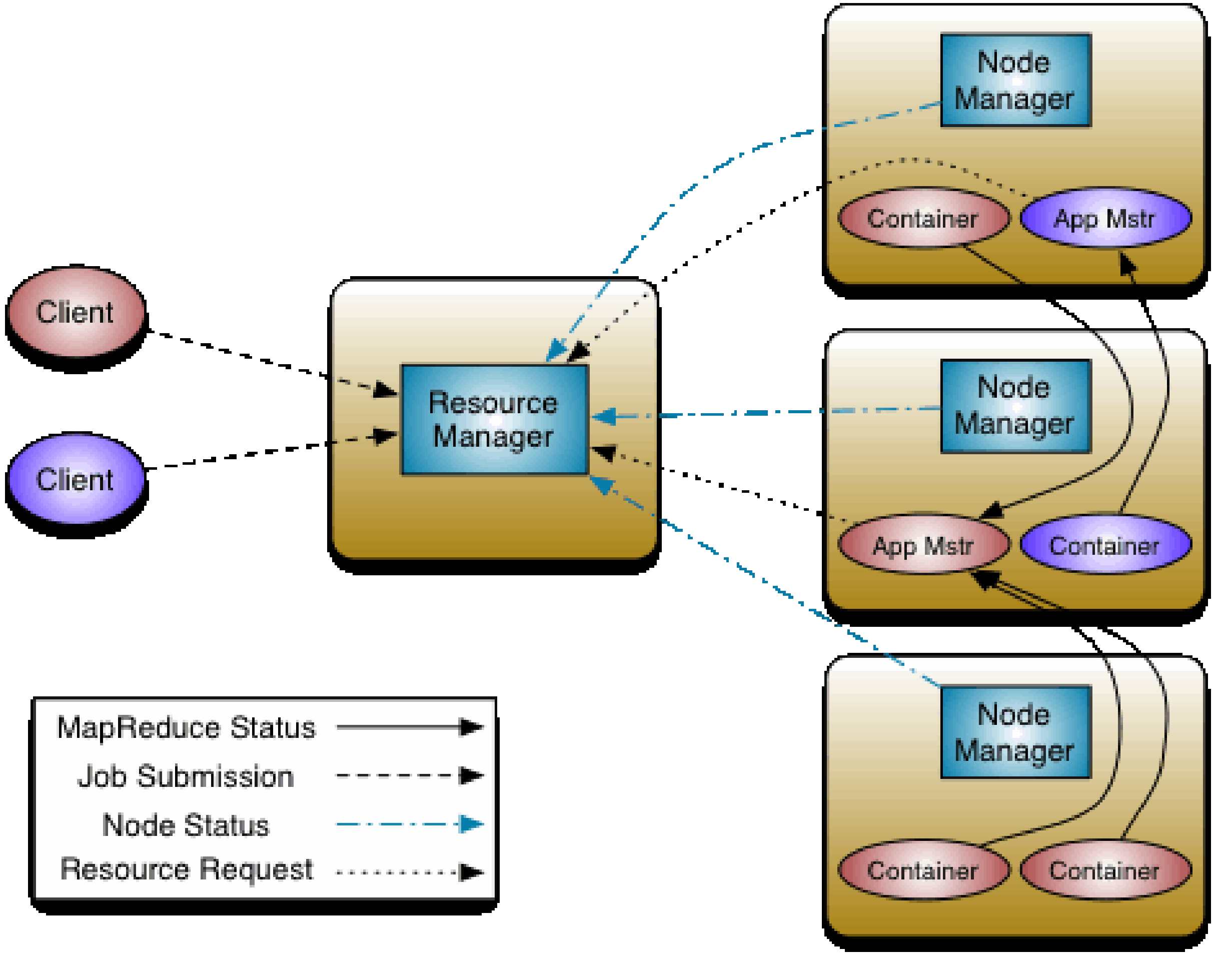
Data Localization

Fault Tolerant

Providesplugin system for more data processing frameworks

YARN ARCHITECTURE

- Central Resource manager
- per-Node manager
- per-Application master

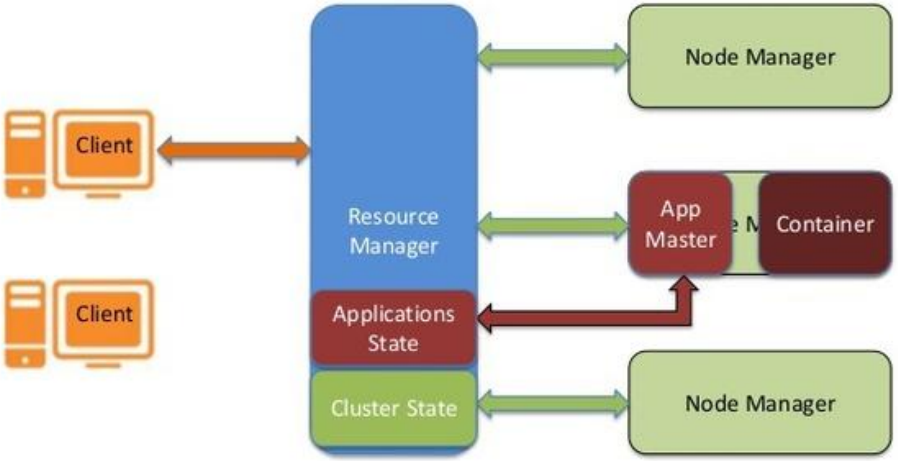


YARN: ARCHITECTURE

Resource Manager (Master)

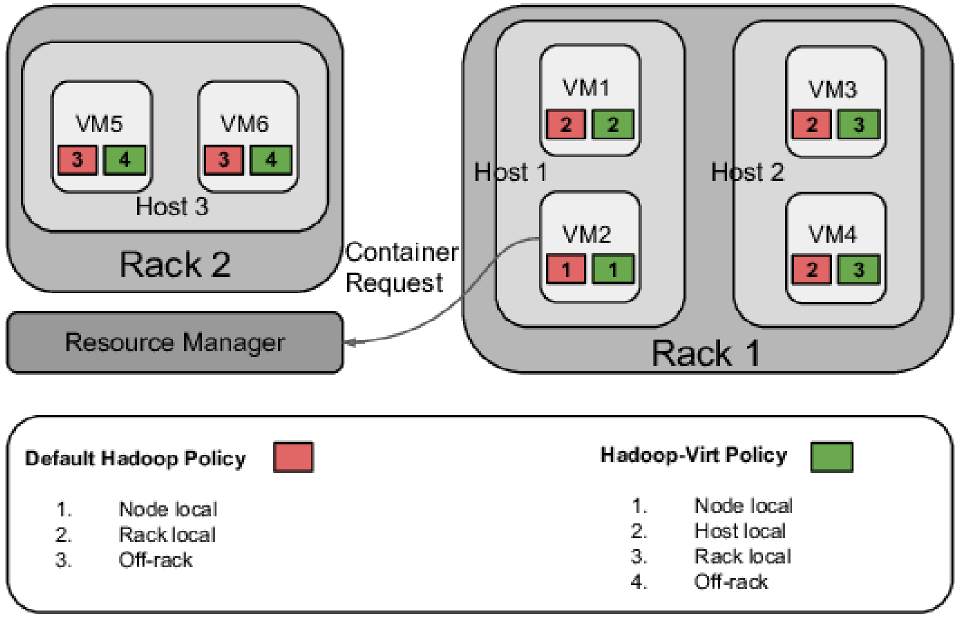
- ○ Handles all client requests for job submission
- ○ Global resource scheduler
- ○ Keeps track of Node managers (NM) and receives heartbeat request from them
- ○ Receives job progress reports from the per-Application Master (AM) for a running job
- ○ Enforces hierarchical queue definition

- **Node Manager**
  - ○ Per-machine agent
  - ○ Manages the life-cycle of container
  - ○ Sends heart beat messages to the RM.
  - ○ Reports current status of available resources to the RM.



YARN: ARCHITECTURE

- **Application Master**
  - Per-application
  - Runtime implementation of a Yarn Application.
  - Manages application scheduling and task execution
  - Sends Job progress report periodically to the RM
  - e.g. MapReduce Application Master
- **Container**
  - Basic unit of allocation
  - Fine-grained resource allocation across multiple resource types (RAM, CPU, Disk, Network, GPU, etc.)



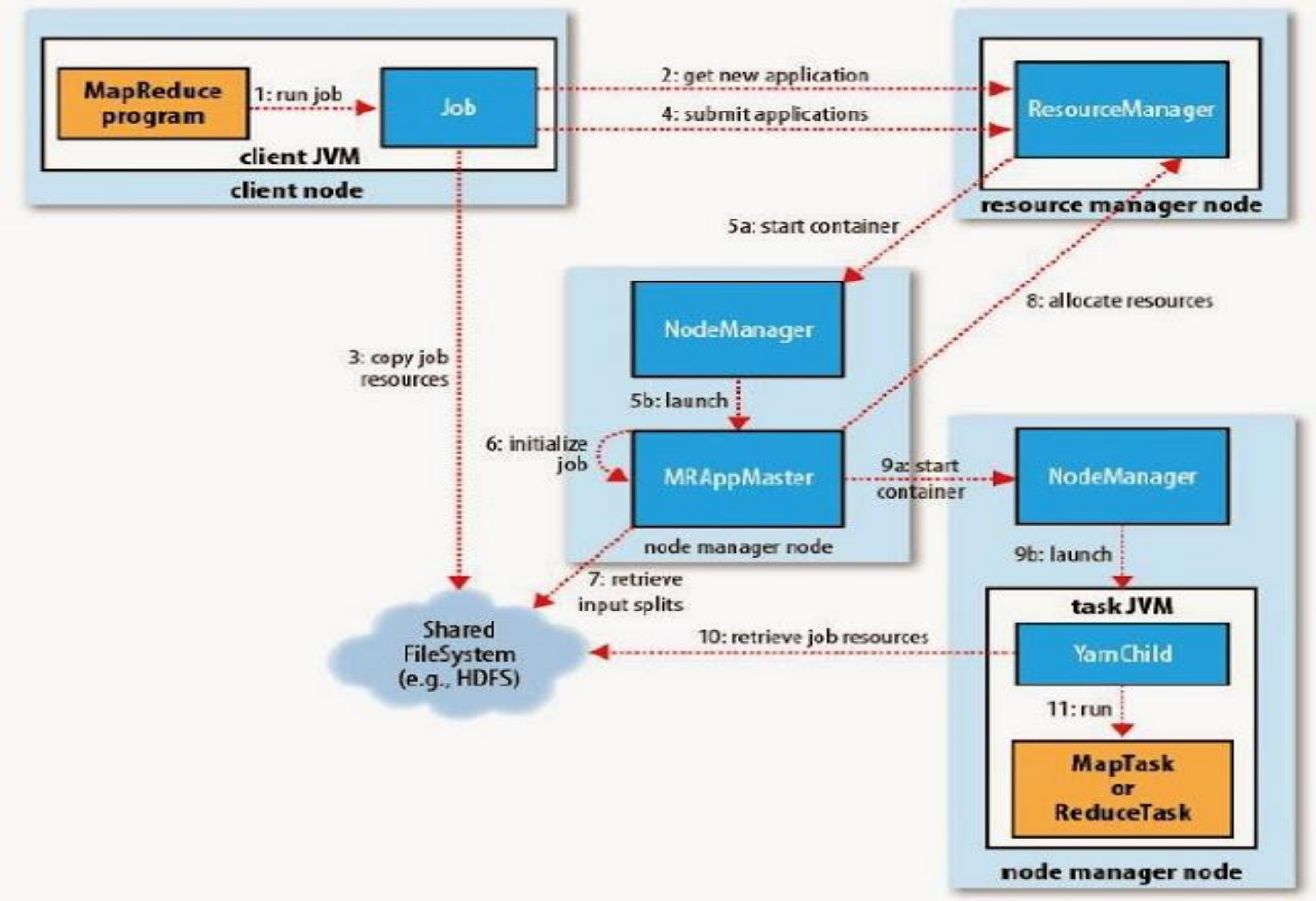
Default Hadoop Policy 🟥

1. Node local
2. Rack local
3. Off-rack

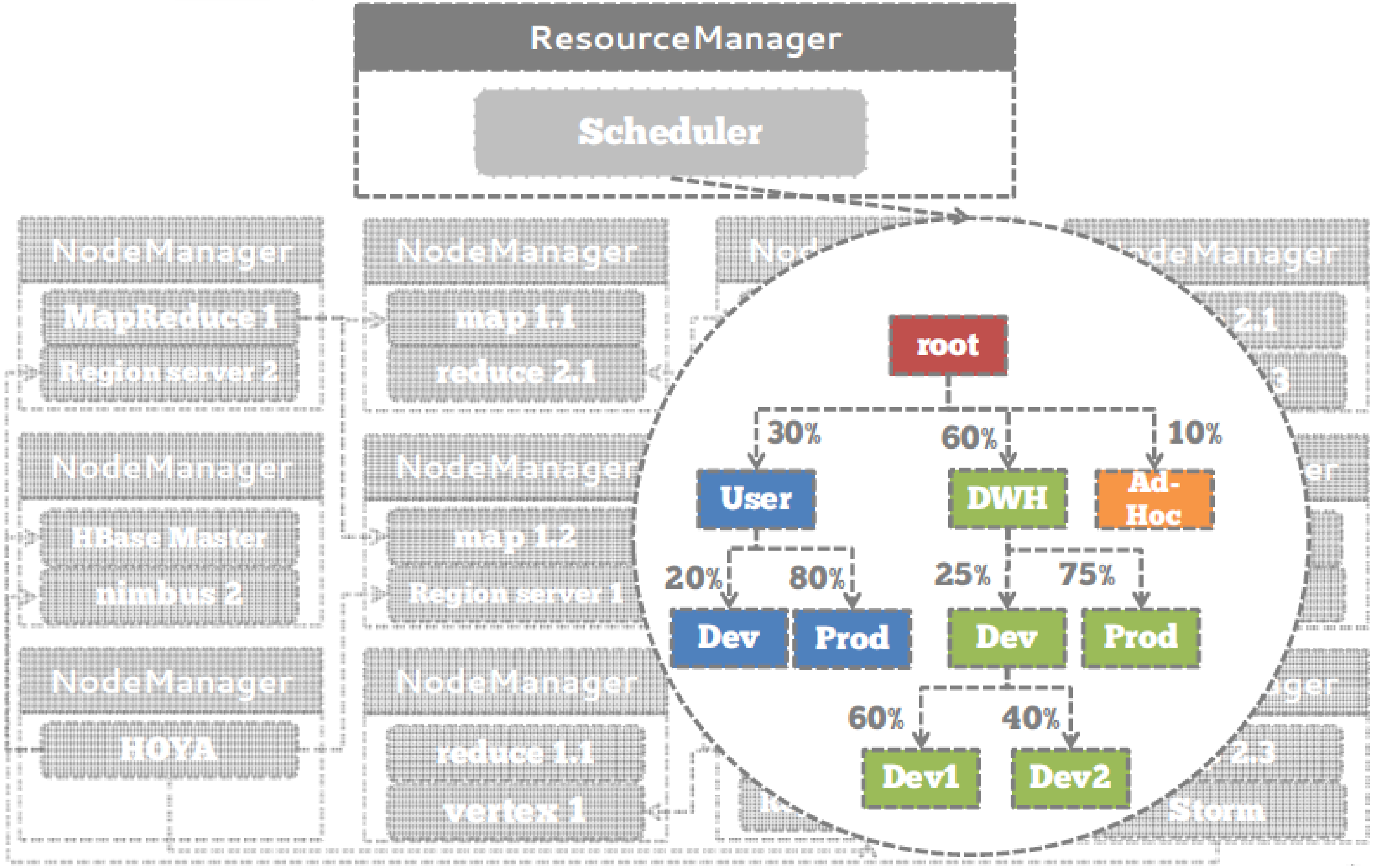
Hadoop-Virt Policy 🟩

1. Node local
2. Host local
3. Rack local
4. Off-rack

MAP-REDUCE FLOW WITH YARN

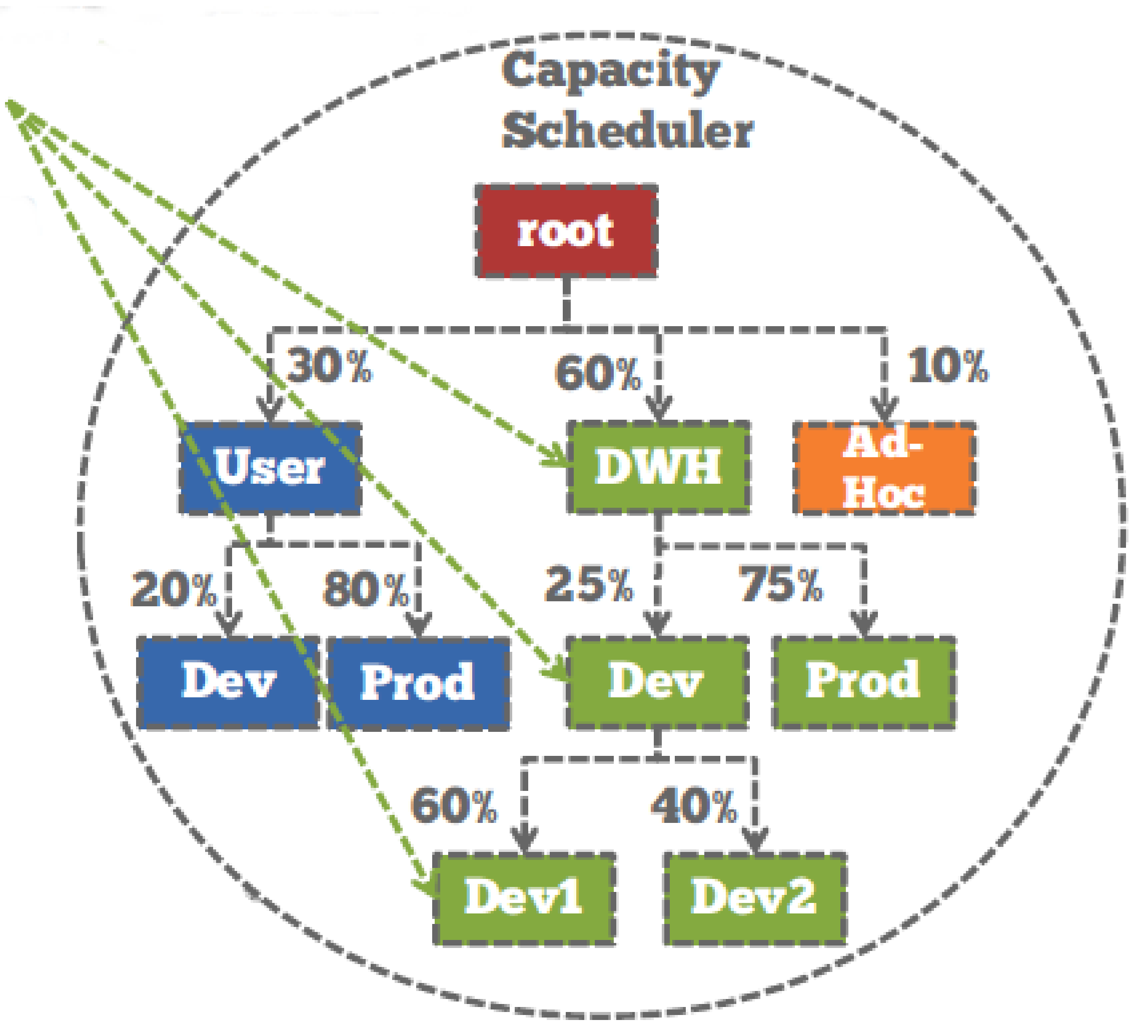


YARN: MULTI-TENANCY

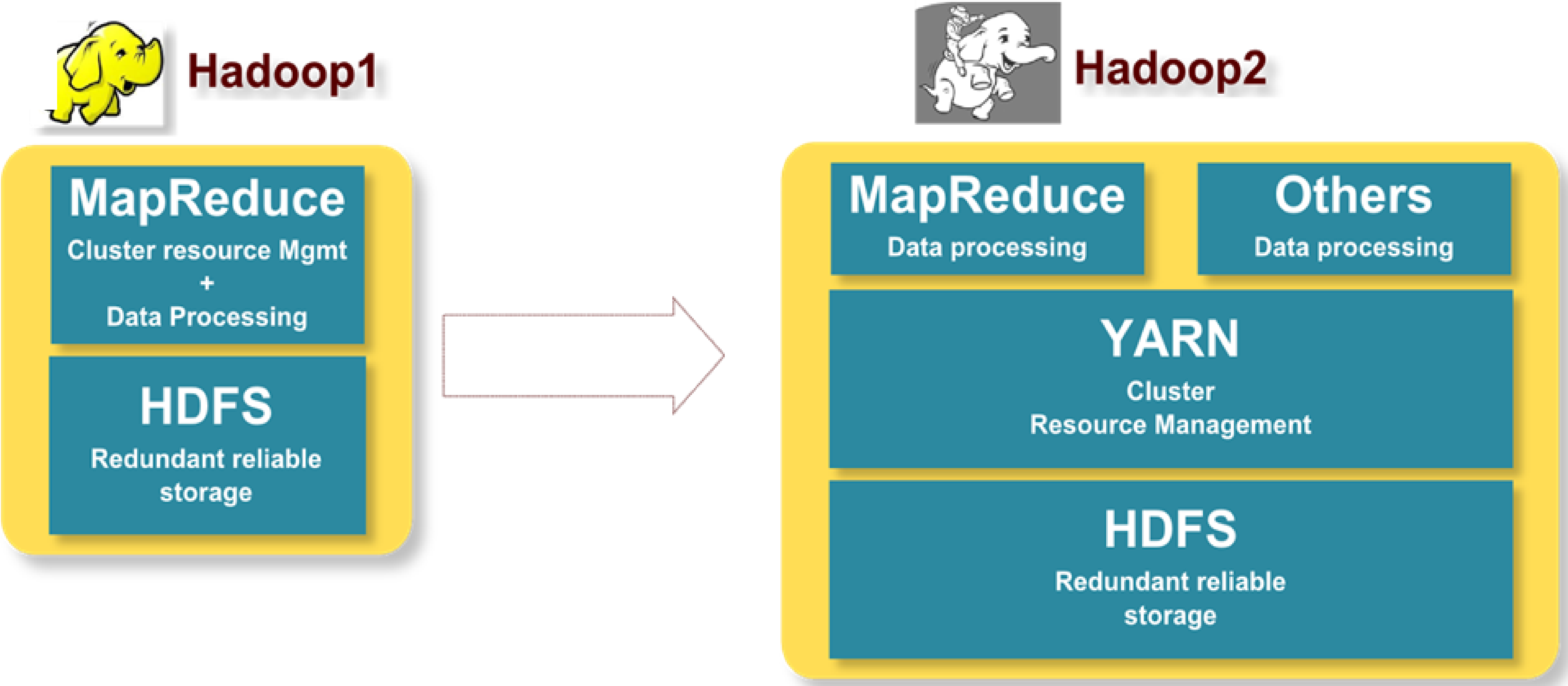


YARN: CAPACITY SCHEDULER

- Queues
- Economics as queue capacities
  - Hierarchical queues
  - SLA‘s via preemption
- Resource Isolation
- Administration
  - Queue ACL‘s
  - Runtime reconfig
  - Charge-back



A LITTLE BIT OF HISTORY



Hadoop1

MapReduce

Cluster resource Mgmt
+
Data Processing

HDFS

Redundant reliable storage

Hadoop2

MapReduce

Data processing

Others

Data processing

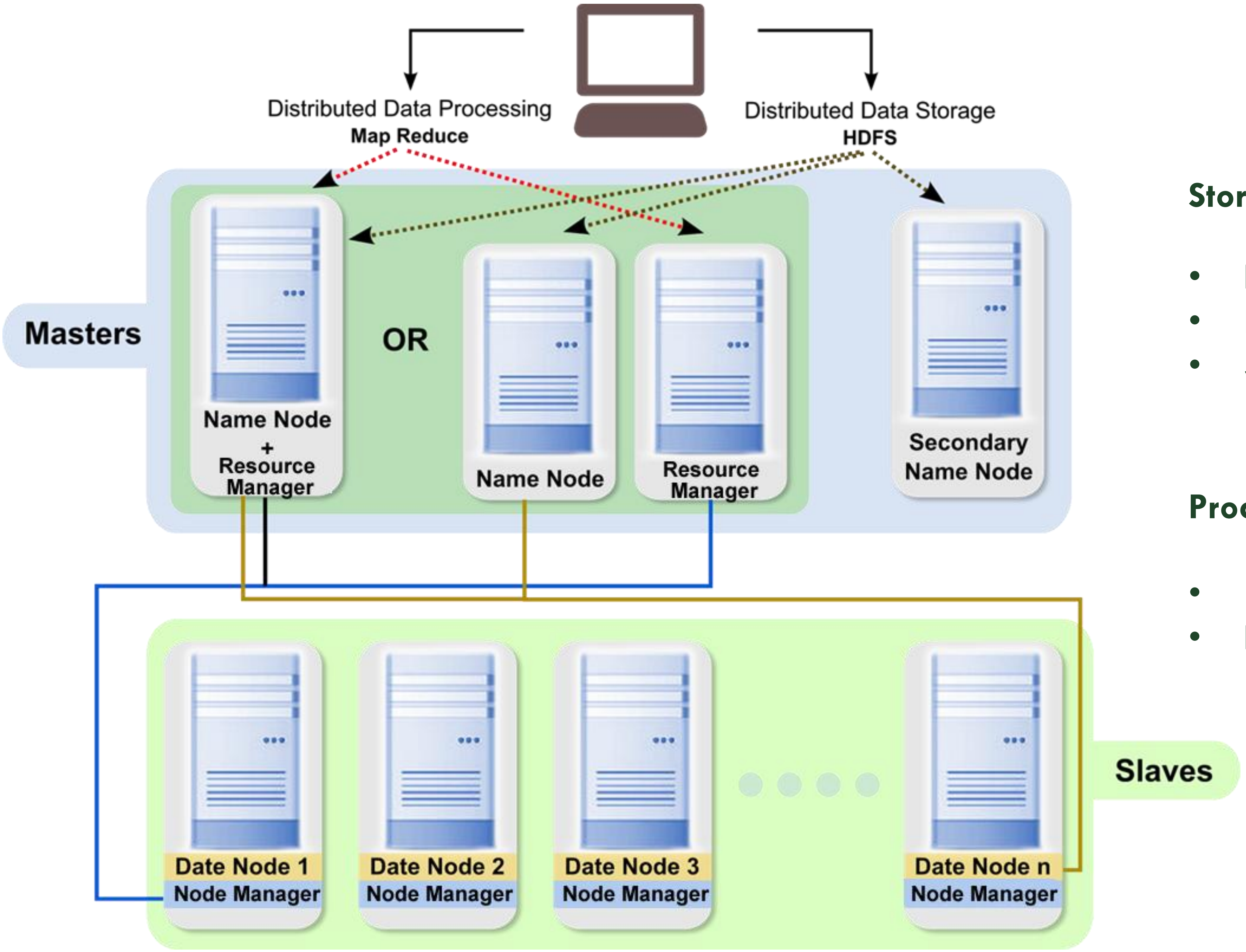
YARN

Cluster
Resource Management

HDFS

Redundant reliable storage

HADOOP DAEMONS



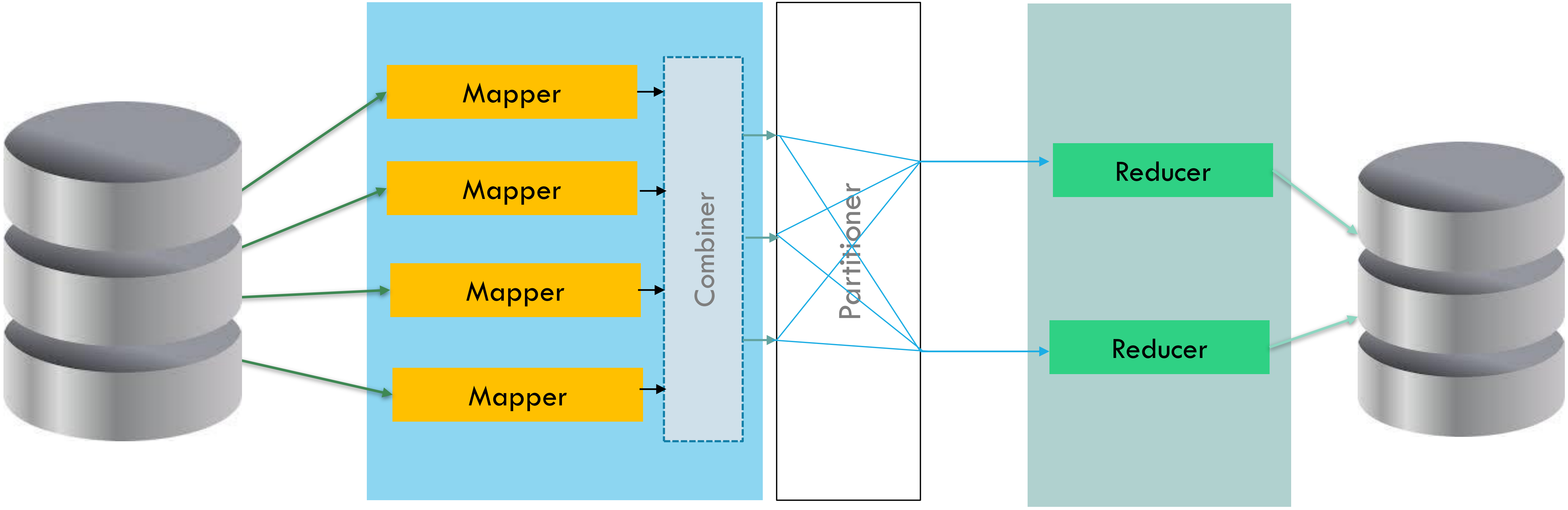
Storage

- Name Node
- Data Node
- Secondary Name Node

Processing

- Resource Manager
- Node Manager

MAPREDUCE



Input Data

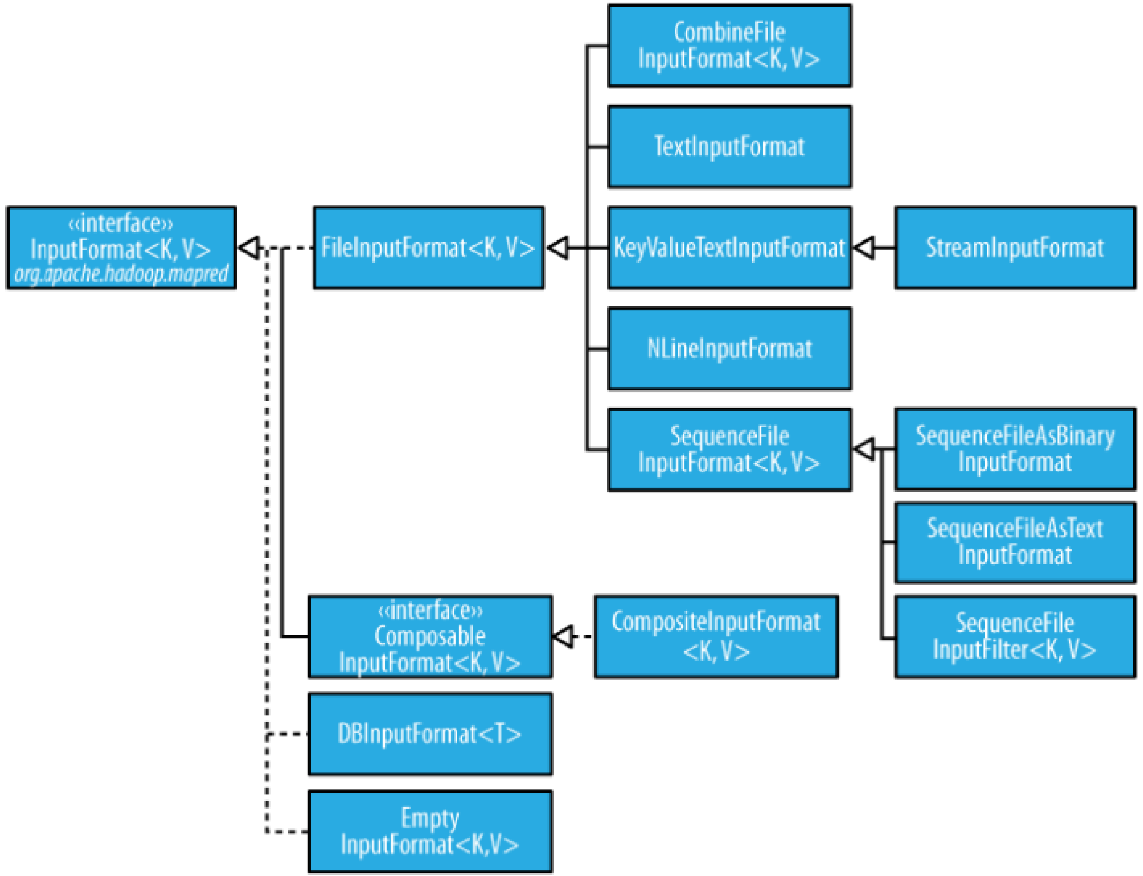
Map Phase

Shuffle Phase

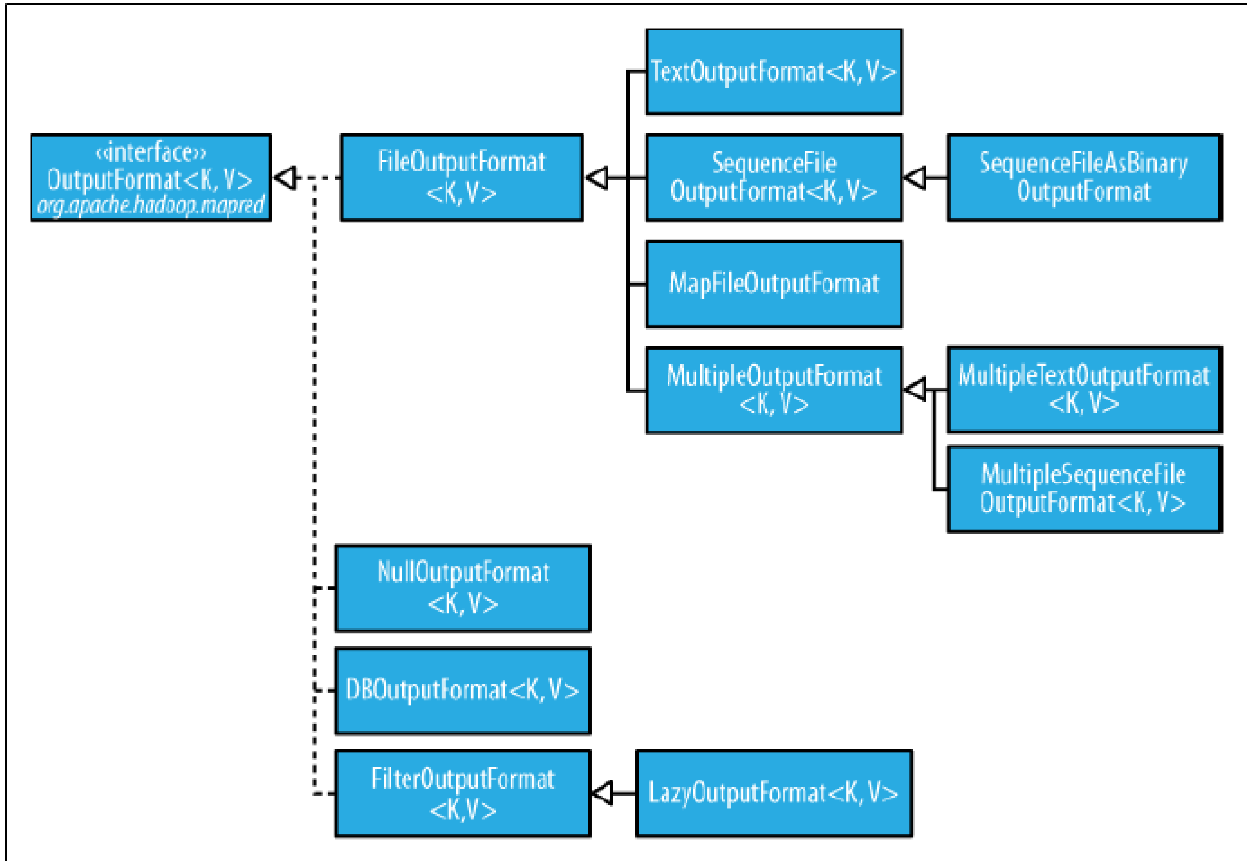
Reduce Phase

Output

MAPREDUCE


«interface»
InputFormat<K, V>
org.apache.hadoop.mapred
FileInputFormat<K, V>
CombineFile InputFormat<K, V>
TextInputFormat
KeyValueTextInputFormat
StreamInputFormat
NLineInputFormat
SequenceFile InputFormat<K, V>
SequenceFileAsBinary InputFormat
SequenceFileAsText InputFormat
SequenceFile InputFilter<K, V>
«interface» Composable InputFormat<K, V>
CompositeInputFormat <K, V>
DBInputFormat<T>
Empty InputFormat<K,V>

InputFormat

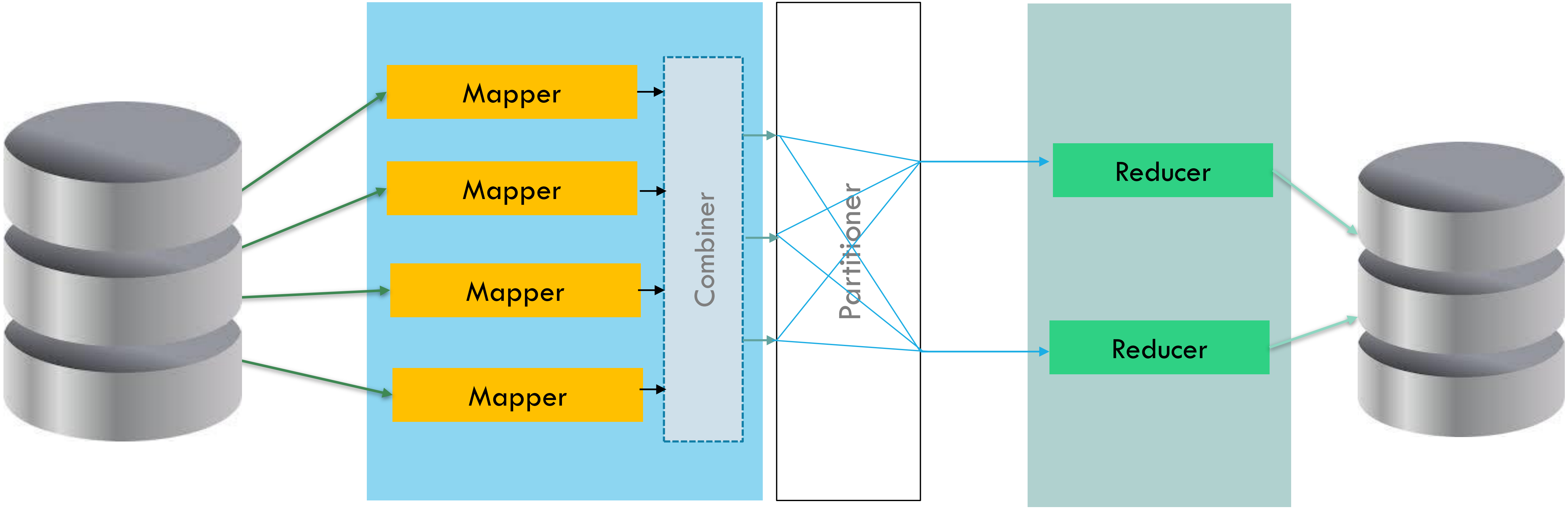


OutputFormat

MAPREDUCE PRIMITIVES

Hadoop	Java	Comment
BooleanWritable	Boolean; boolean	
BytesWritable	byte[]	
ByteWritable	Byte; byte	
DoubleWritable	Double; double	
IntWritable	Integer; int	
LongWritable	Long; long	
NullWritable	null	NullWritable != null reference. It is a safe null.
ShortWritable	Short; short	
Text	String	
Writable		The parent of all Hadoop IO primitive type
WritableComparable		A writable that has the compareTo method implementation. All implementation of this class can be used for keys in a key value pair

MAPREDUCE



Input Data

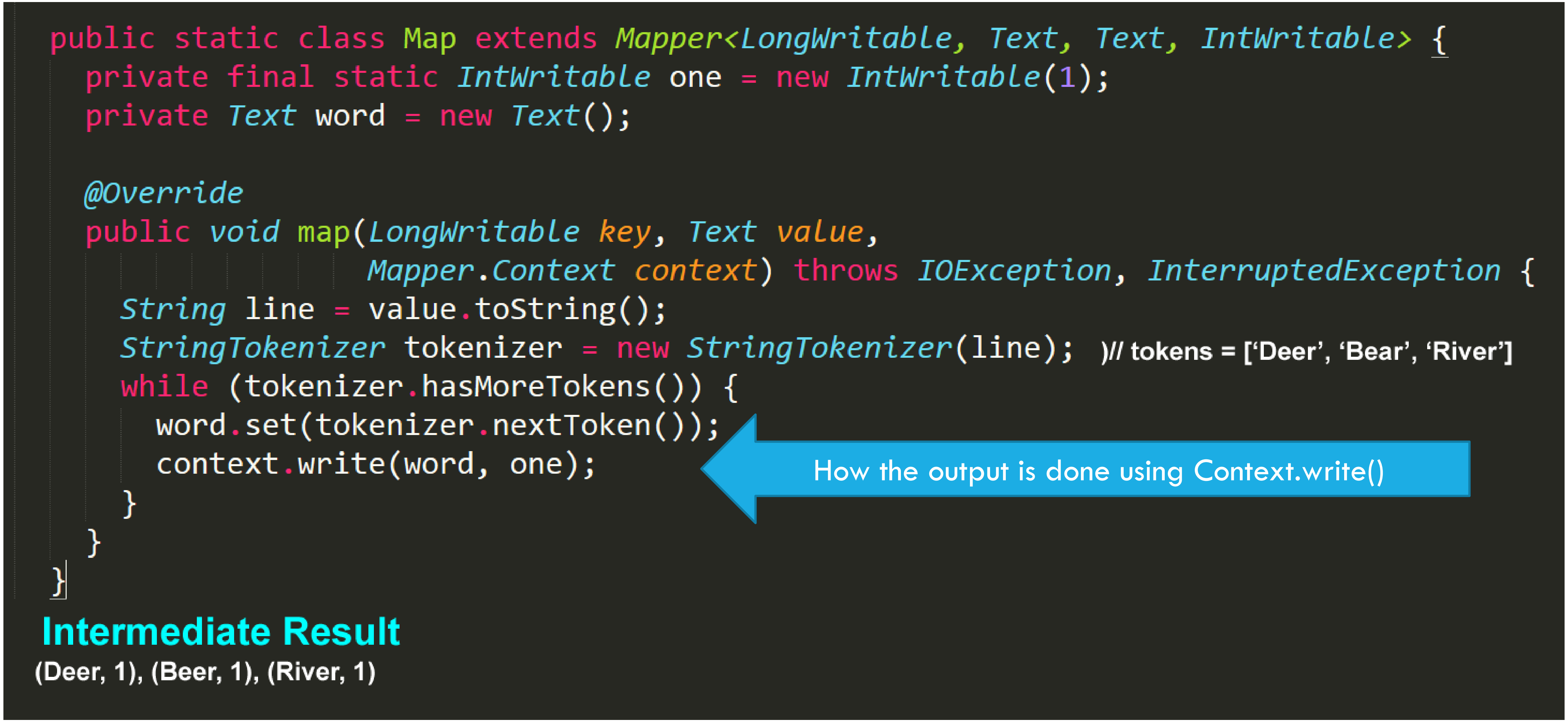
Map Phase

Shuffle Phase

Reduce Phase

Output

MAP



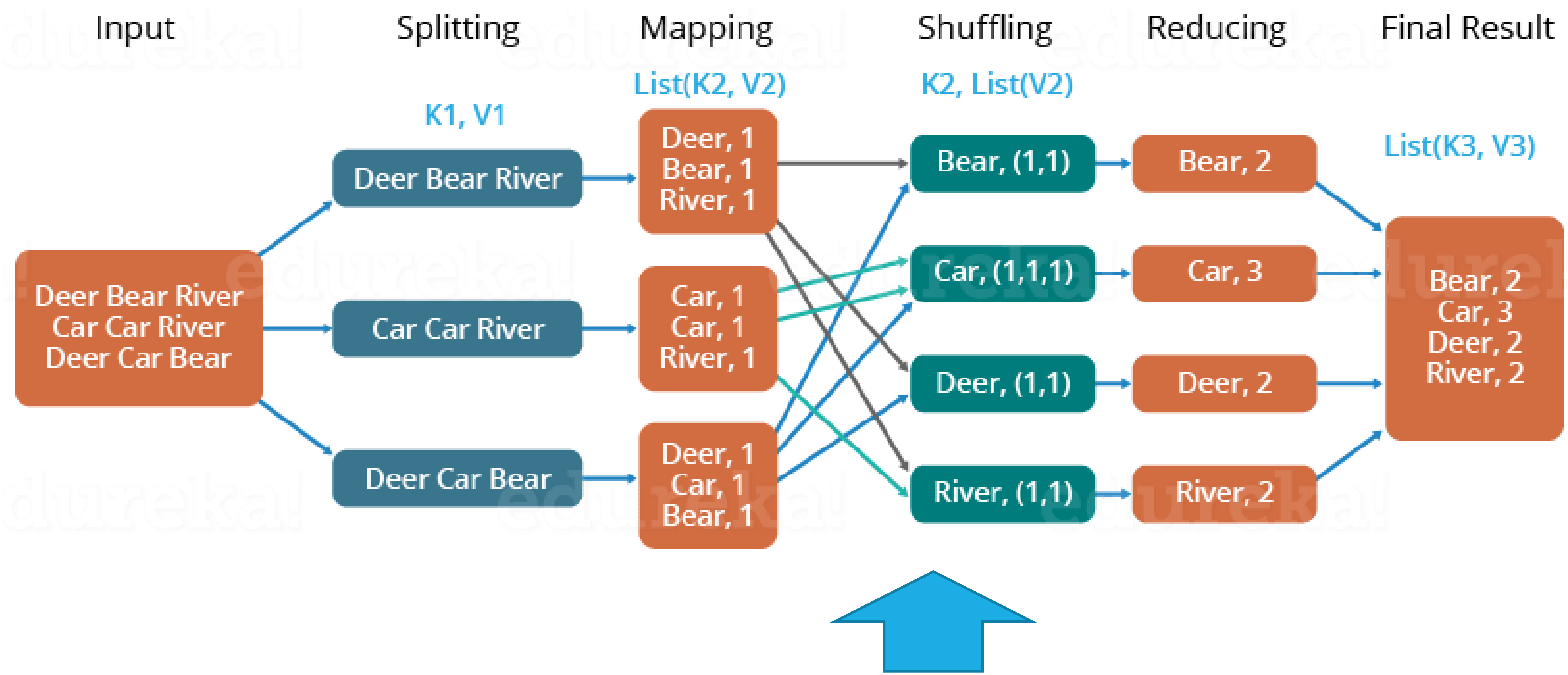
```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

  @Override
  public void map(LongWritable key, Text value,
                  Mapper.Context context) throws IOException, InterruptedException {
    String line = value.toString();
    StringTokenizer tokenizer = new StringTokenizer(line);  )// tokens = [‘Deer’, ‘Bear’, ‘River’]
    while (tokenizer.hasMoreTokens()) {
      word.set(tokenizer.nextToken());
      context.write(word, one);
    }
  }
}
```

How the output is done using Context.write()

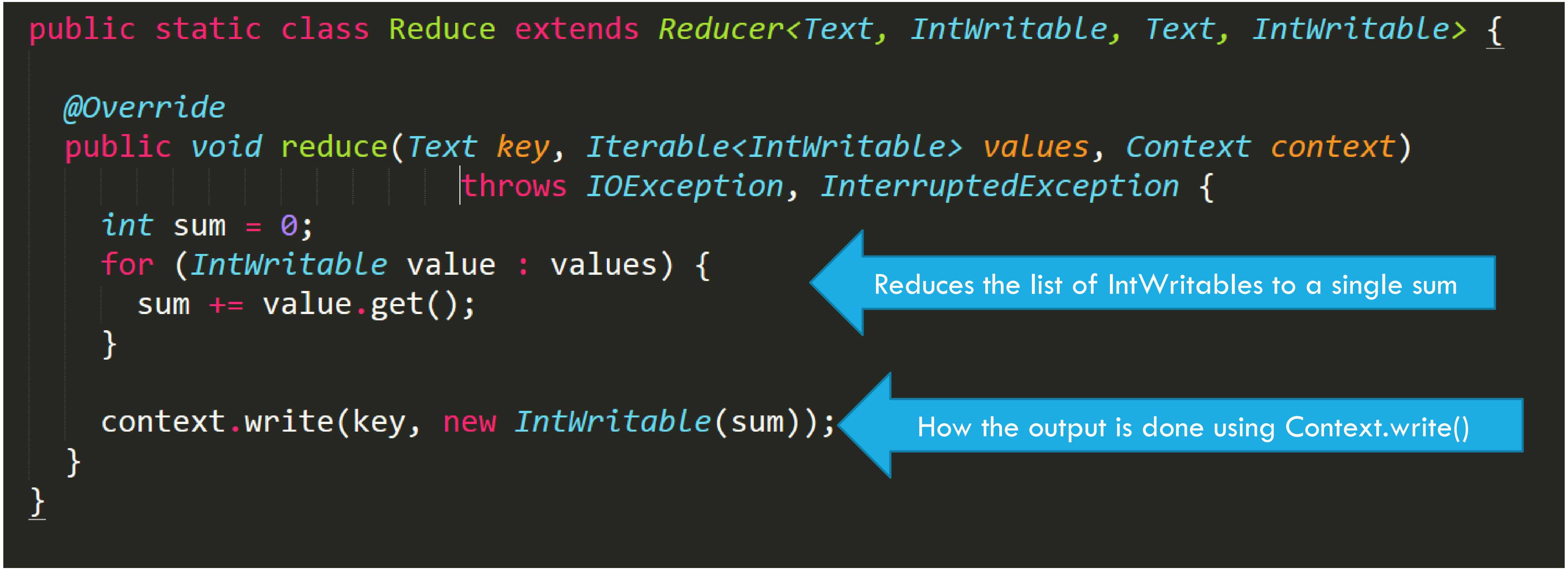
Intermediate Result

(Deer, 1), (Beer, 1), (River, 1)

SHUFFLE


Input
Splitting
Mapping
Shuffling
Reducing
Final Result
K1, V1
List(K2, V2)
K2, List(V2)
List(K3, V3)
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, (1,1)
Car, (1,1,1)
Deer, (1,1)
River, (1,1)
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

REDUCE

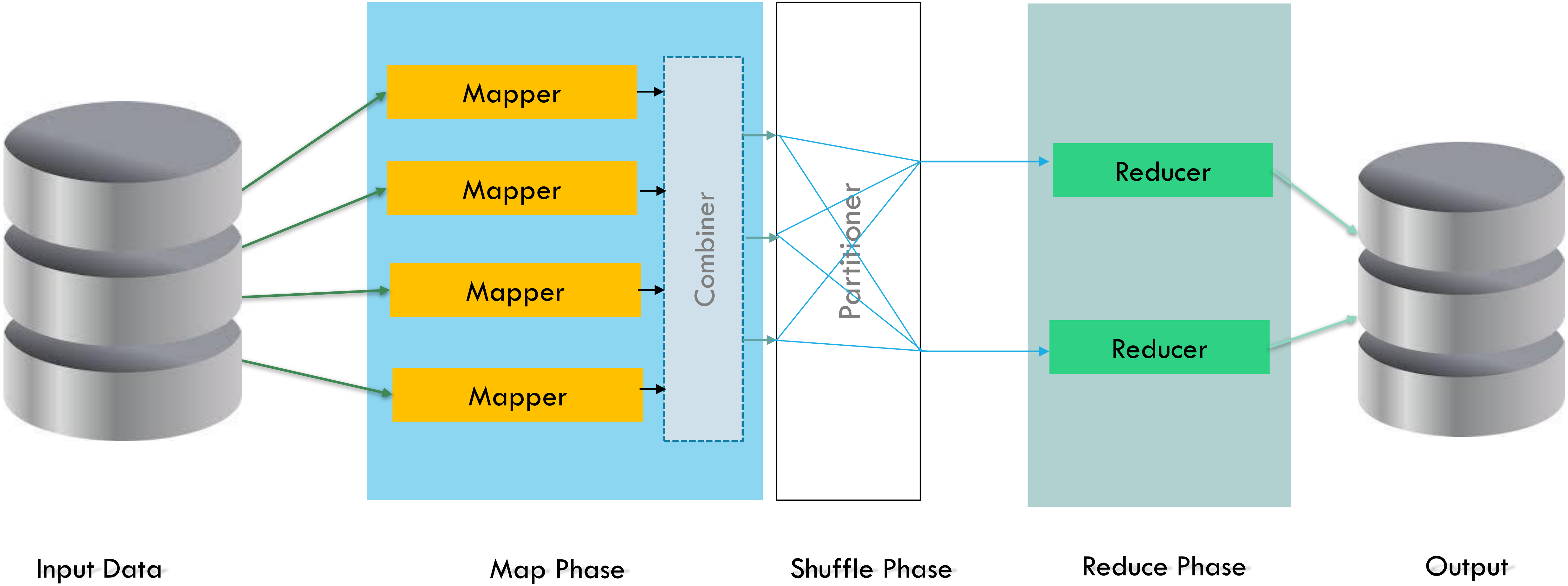


```java
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

  @Override
  public void reduce(Text key, Iterable<IntWritable> values, Context context)
                        throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable value : values) {
      sum += value.get();
    }

    context.write(key, new IntWritable(sum));
  }
}
```

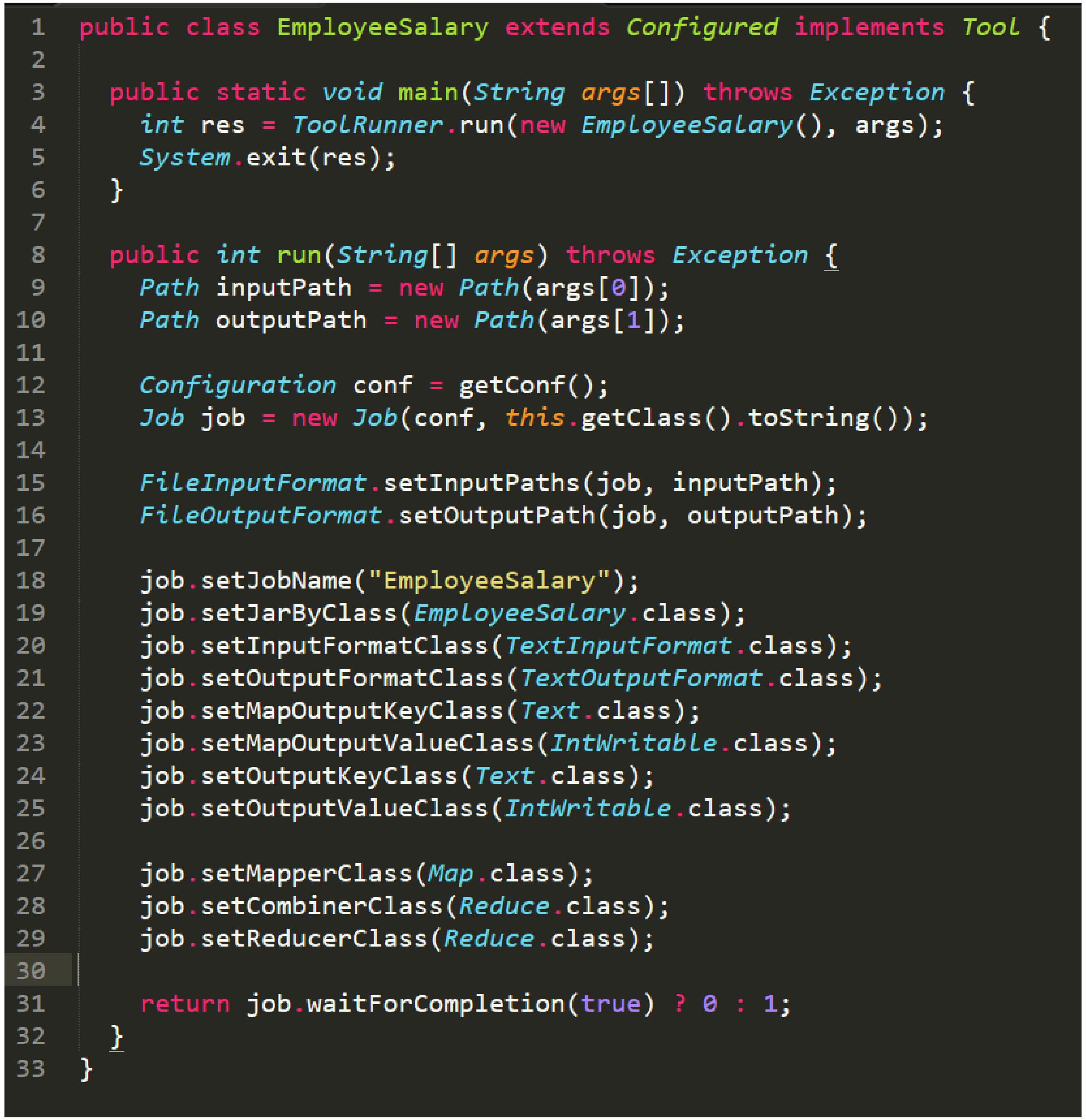
Reduces the list of IntWritables to a single sum

How the output is done using Context.write()

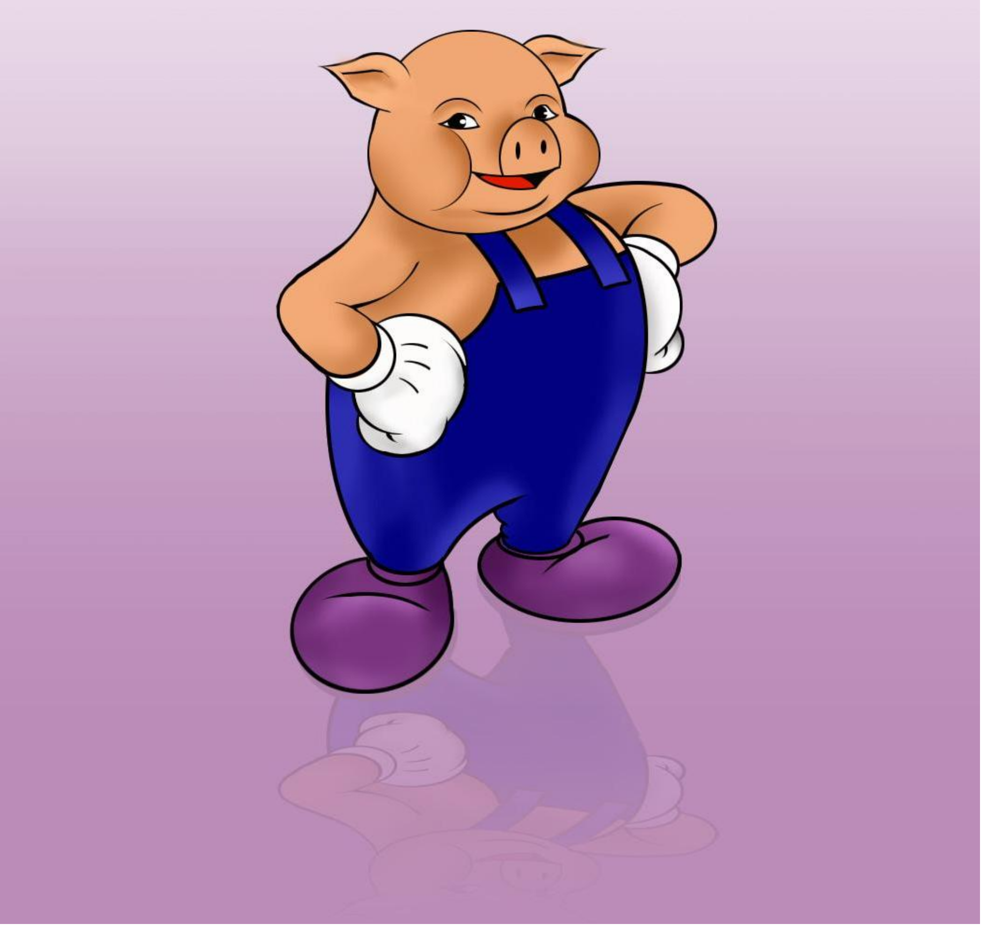
MAPREDUCE

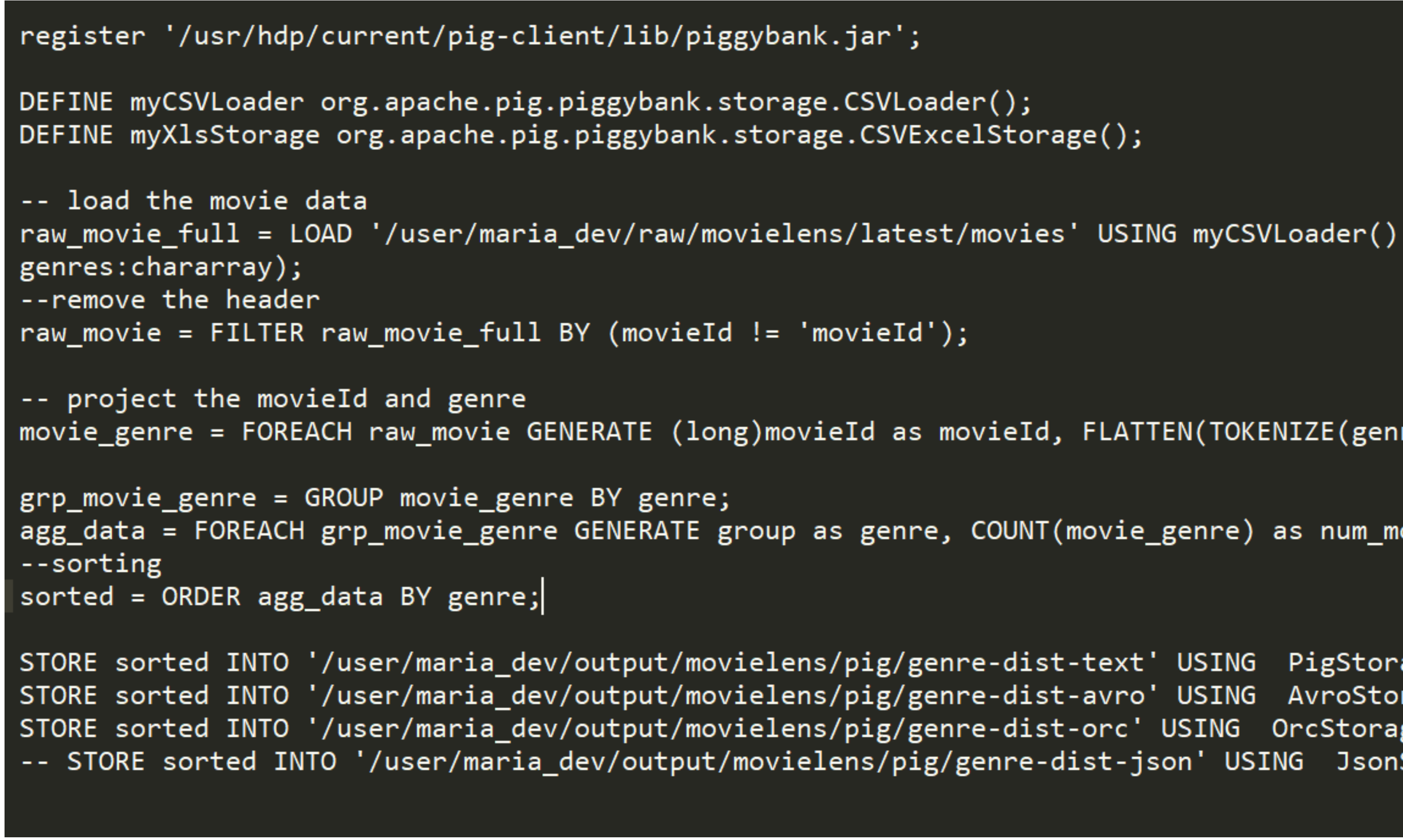

Mapper
Mapper
Mapper
Mapper
Combiner
Partitioner
Reducer
Reducer
Input Data
Map Phase
Shuffle Phase
Reduce Phase
Output

DRIVER CODE



```java
public class EmployeeSalary extends Configured implements Tool {

  public static void main(String args[]) throws Exception {
    int res = ToolRunner.run(new EmployeeSalary(), args);
    System.exit(res);
  }

  public int run(String[] args) throws Exception {
    Path inputPath = new Path(args[0]);
    Path outputPath = new Path(args[1]);

    Configuration conf = getConf();
    Job job = new Job(conf, this.getClass().toString());

    FileInputFormat.setInputPaths(job, inputPath);
    FileOutputFormat.setOutputPath(job, outputPath);

    job.setJobName("EmployeeSalary");
    job.setJarByClass(EmployeeSalary.class);
    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(IntWritable.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    job.setMapperClass(Map.class);
    job.setCombinerClass(Reduce.class);
    job.setReducerClass(Reduce.class);

    return job.waitForCompletion(true) ? 0 : 1;
  }
}
```

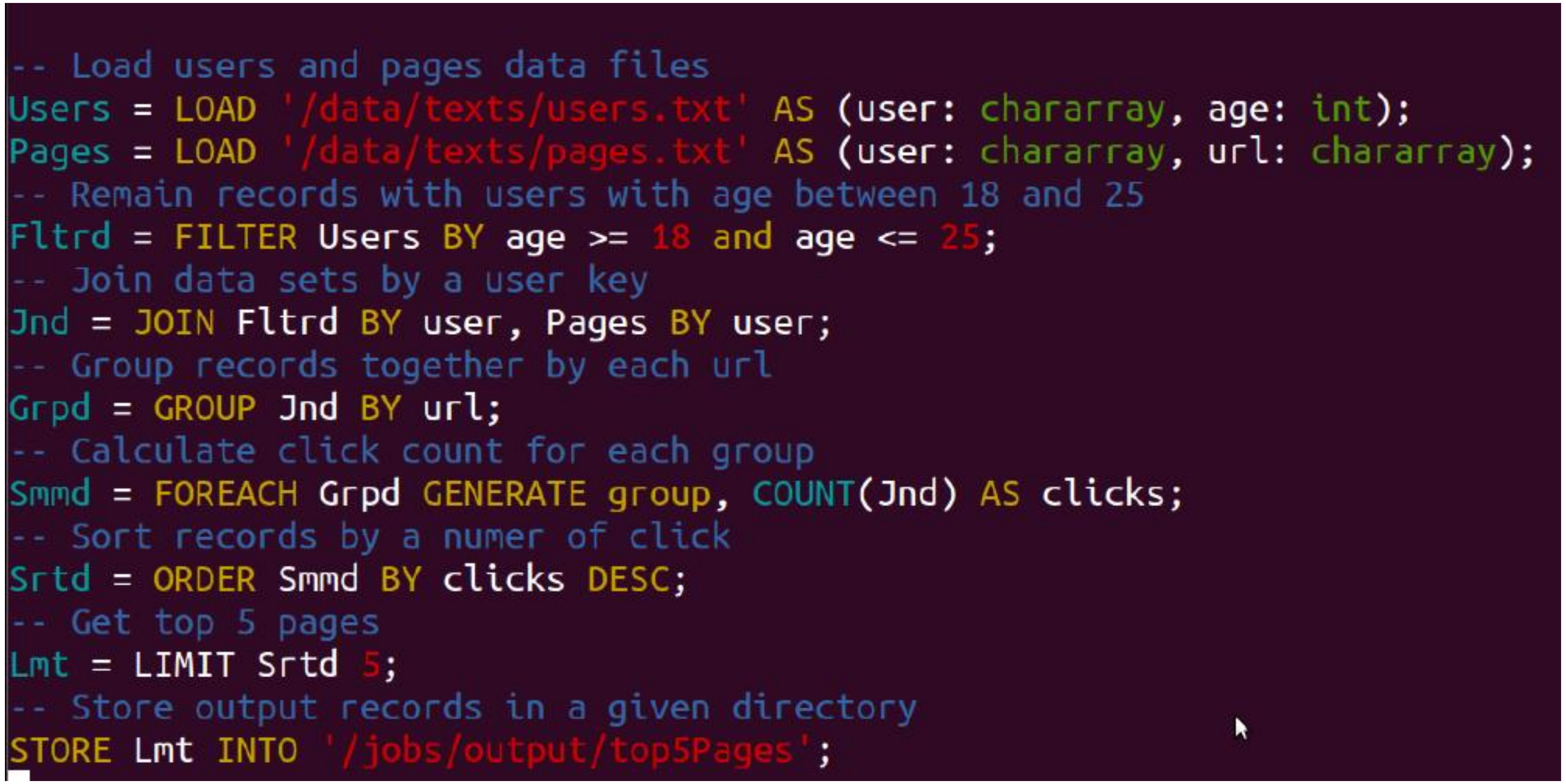




```
register '/usr/hdp/current/pig-client/lib/piggybank.jar';

DEFINE myCSVLoader org.apache.pig.piggybank.storage.CSVLoader();
DEFINE myXlsStorage org.apache.pig.piggybank.storage.CSVExcelStorage();

-- load the movie data
raw_movie_full = LOAD '/user/maria_dev/raw/movielens/latest/movies' USING myCSVLoader()
genres:chararray);
--remove the header
raw_movie = FILTER raw_movie_full BY (movieId != 'movieId');

-- project the movieId and genre
movie_genre = FOREACH raw_movie GENERATE (long)movieId as movieId, FLATTEN(TOKENIZE(gen

grp_movie_genre = GROUP movie_genre BY genre;
agg_data = FOREACH grp_movie_genre GENERATE group as genre, COUNT(movie_genre) as num_m
--sorting
sorted = ORDER agg_data BY genre;

STORE sorted INTO '/user/maria_dev/output/movielens/pig/genre-dist-text' USING  PigStor
STORE sorted INTO '/user/maria_dev/output/movielens/pig/genre-dist-avro' USING  AvroSto
STORE sorted INTO '/user/maria_dev/output/movielens/pig/genre-dist-orc' USING  OrcStora
-- STORE sorted INTO '/user/maria_dev/output/movielens/pig/genre-dist-json' USING  Json
```

PIG

PIG LATIN - ANATOMY



```
-- Load users and pages data files
Users = LOAD '/data/texts/users.txt' AS (user: chararray, age: int);
Pages = LOAD '/data/texts/pages.txt' AS (user: chararray, url: chararray);
-- Remain records with users with age between 18 and 25
Fltrd = FILTER Users BY age >= 18 and age <= 25;
-- Join data sets by a user key
Jnd = JOIN Fltrd BY user, Pages BY user;
-- Group records together by each url
Grpd = GROUP Jnd BY url;
-- Calculate click count for each group
Smmd = FOREACH Grpd GENERATE group, COUNT(Jnd) AS clicks;
-- Sort records by a numer of click
Srtd = ORDER Smmd BY clicks DESC;
-- Get top 5 pages
Lmt = LIMIT Srtd 5;
-- Store output records in a given directory
STORE Lmt INTO '/jobs/output/top5Pages';
```

Variables

- ✓ Relations
- ✓ Attributes

Operator

- ✓ Loader
- ✓ Transformations
- ✓ Actions

Function

- ✓ Built-in functions
- ✓ Custom UDF

PIG DATA TYPES

- **Scalar types**
  - int, long, float, double, chararray, bytearray, boolean
- **Complex types**
  - tuple - sequence of fields of any type
    - ('Poland', 2, 0.66)
  - bag - an unordered collection of tuples, possibly with duplicates
    - {('Poland', 2), ('Greece'), (3.14)}
  - map - a set of key-value pairs. Keys must be a chararray, values may be any type
    - ['Poland'#'Euro2012']

PIG LATIN – TRANSFORMING DATA USING OPERATORS

Category	Operator	Description
Loading and Storing	LOAD STORE DUMP	Loads data from the file system. (Loader) Saves a relation to the file system or other storage. (Action) Prints a relation to the console (Action)
Filtering	FILTER DISTINCT FOREACH...GENERATE	Removes unwanted rows from a relation. Removes duplicate rows from a relation. Adds or removes fields from a relation.
Grouping and Joining	JOIN COGROUP GROUP CROSS	Joins two or more relations. Groups the data in two or more relations. Groups the data in a single relation. Creates the cross product of two or more relations.
Sorting	ORDER LIMIT	Sorts a relation by one or more fields. Limits the size of a relation to a maximum number of tuples.
Combining and Splitting	UNION SPLIT	Combines two or more relations into one. Splits a relation into two or more relations.

PIG LATIN – USER DEFINED FUNCTIONS (UDF)

- Load/Store functions - determines how data is loaded into pig or returned back from the pig run. Examples include PigStorage(), BinStorage, JsonLoader(), JsonStorage(), AvroStorage(), OrcStorage(), PigDump(), etc

- Evaluation functions – preforms an evaluation on zero or more expression or arguments passed into it. Math, String, Date time and collection functions are evaluation functions.

- Filter functions – function that may take and expression and return true or false. Use in conjuction with the FILTER operator to remove records from the relation.

- Math functions – are evaluation functions that perform mathematical calculations. E.g. COS(), CEIL(), ABS(), LOG(), LOG10(), RANDOM(), etc.

- String functions – are evaluation functions that perform string manipulations task. E.g. STARTSWITH(), STRSPLIT(), etc.

- Date/time functions – perform and return values from either a datetime calculation or information about system date and/or time. E.g., AddDuration(), CurrentTime(), ToUnixTime(), etc.

- Complex data type functions – returns values after performing operations on collections data types. E.g. TOTUPLE(), TOMAP(), TOP(), etc.

PIG LATIN – CUSTOM UDF

In many cases, we will need functions that are not built into Pig. To get these functionalities into our pig dataflow application, we can either

Reuse third-party UDFs

A number of third party Pig function are open source and can be registered in our application to provide wider range of functionalities in addition to pig's built in functions.

Develop Custom UDFs

We can also build our own custom UDF using any of Java, JavaScript, groovy, Jython, Ruby and Python programming language.

Note that the above programming languages except Java have limited support.

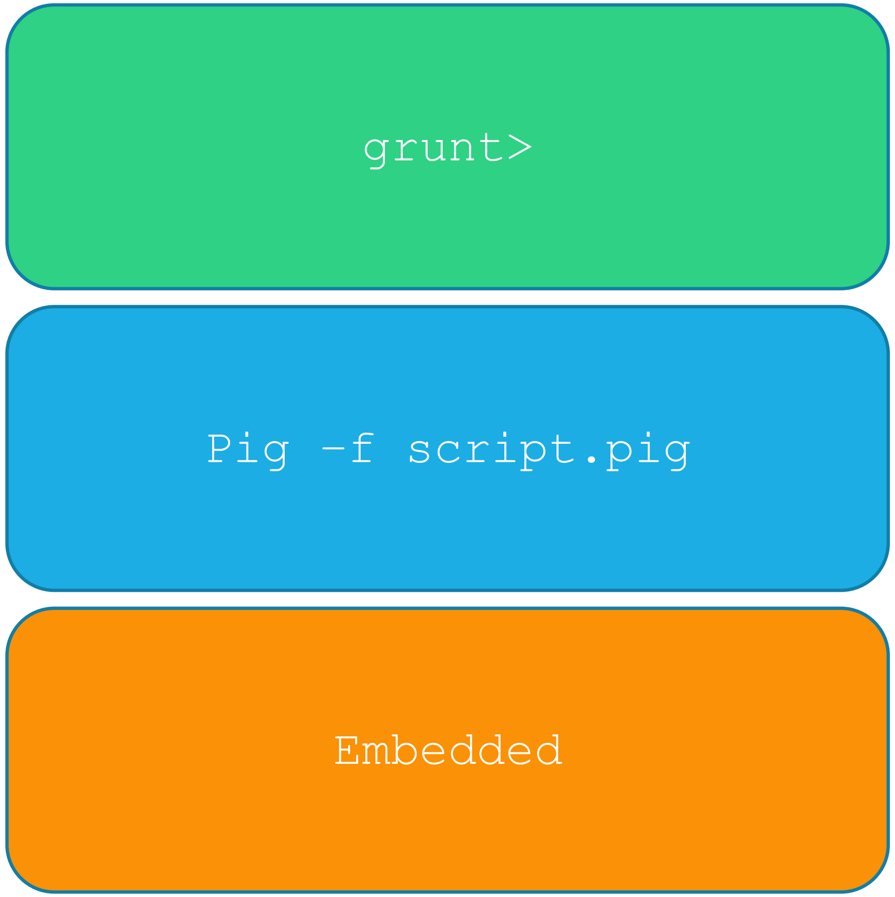
…… Avoid reinventing the wheel!

WHERE TO FIND USEFUL PIGLATIN UDFS?

- **PiggyBank** - Pig’s repository of user contributed functions
  - load/store functions (e.g. from XML)
  - datetime, text functions, math, stats functions
- **DataFu** - LinkedIn's collection of Pig UDFs
  - statistics functions (quantiles, variance etc.)
  - convenient bag functions (intersection, union etc.)
  - utility functions (assertions, random numbers, MD5, distance between lat/long pair), PageRank
- **Elephant Bird**

DEPLOYMENT

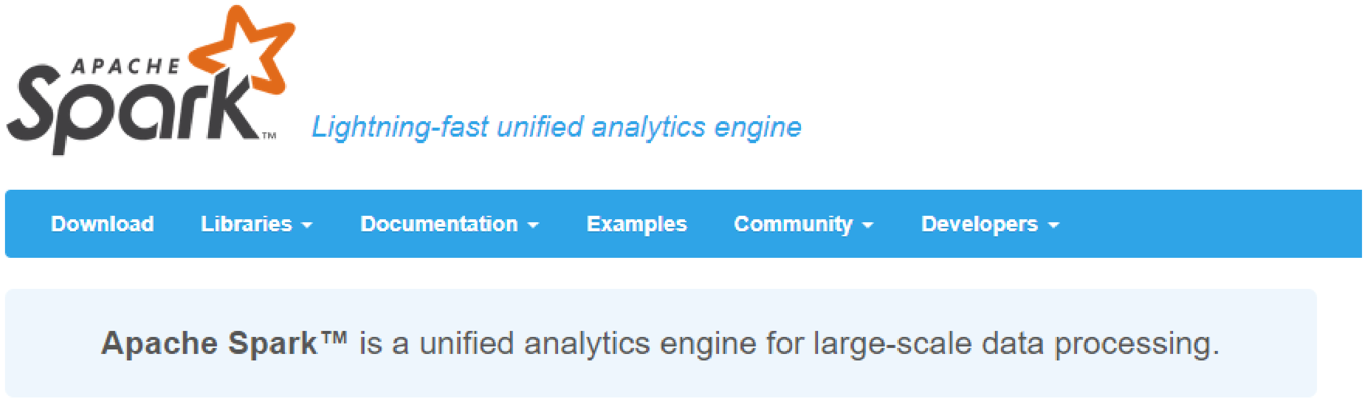
- **Grunt –** A pig REPL (read-eval-print-loop) shell that allow a developer to interactively write and execute pig instructions. It provides a shell for interactive ad hoc data analysis. If you execute the pig command with an argument, you get a grunt shell
- **Script –** Running a pig command with the –f <script_file> will enable pig to run in a batch mode. It will execute all commands in the script and terminate when complete.
- **Embedded –** Rare but possible. Pig can be run from within another application by calling some APIs.



grunt>

Pig –f script.pig

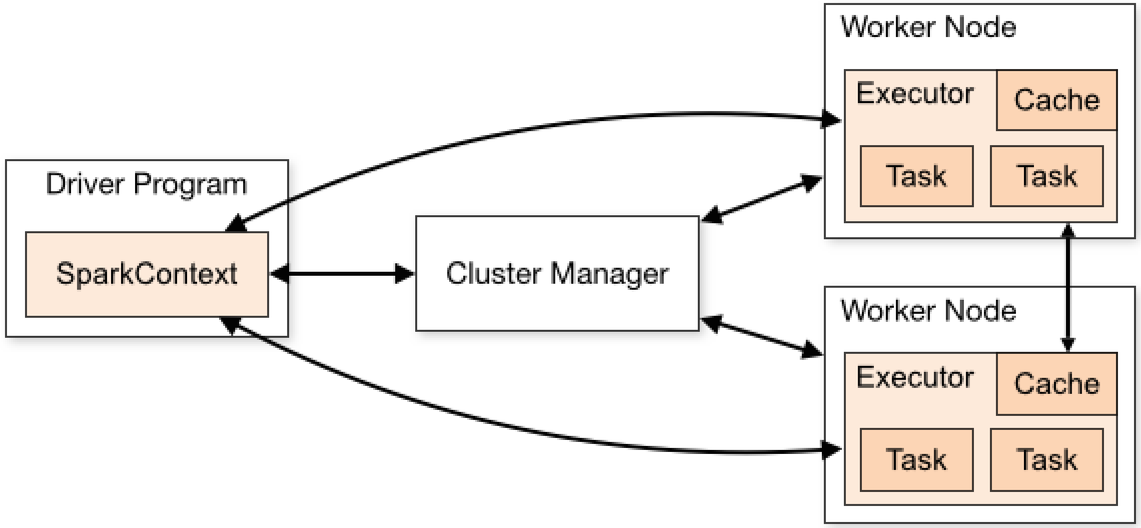
Embedded

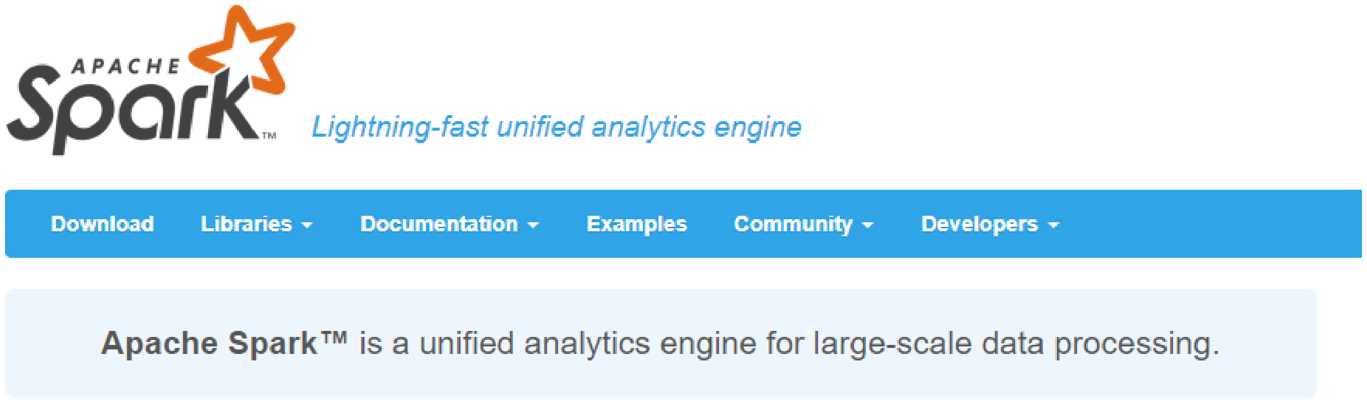


Apache Spark™ is a unified analytics engine for large-scale data processing.

- □ Start in 2009 at AMPLabs at UC Berkeley
- □ Open sourced in 2010
- □ Donated to ASF in 2013
- □ Current version is 2.4.4
- □ Most popular general-purpose big data computing platform today
- □ Flexible IO architecture
- □ Support 4 programming languages – Java, Scala, Python and R
- □ Basic data structure is the RDD.

Architecture





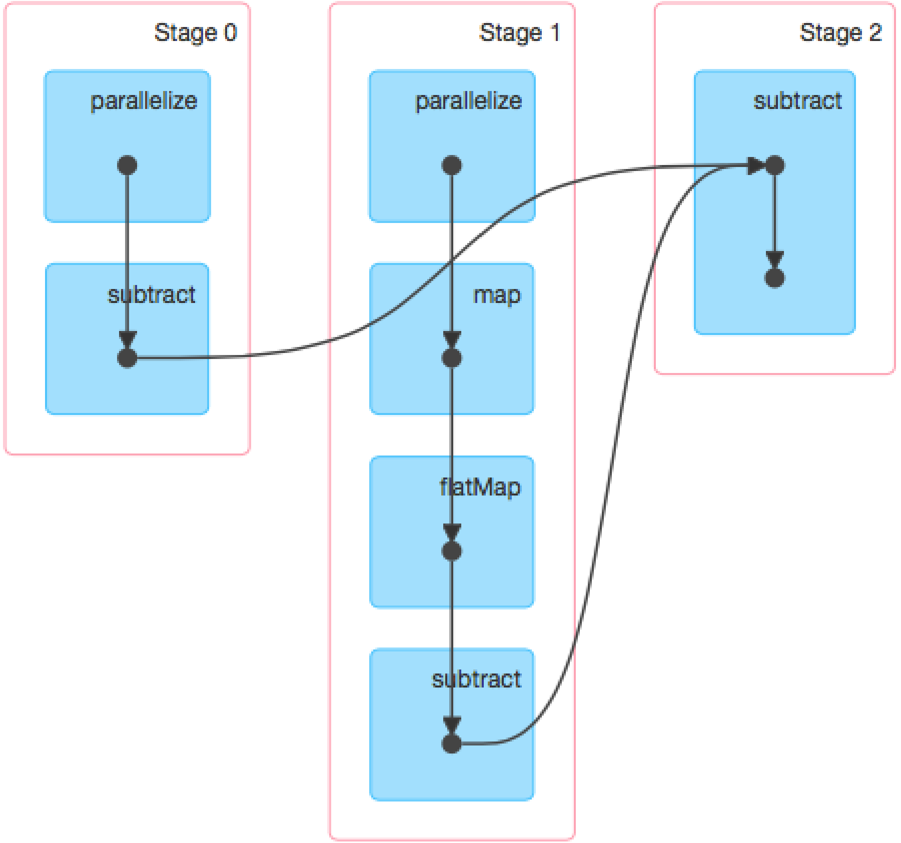
Lightning-fast unified analytics engine

Apache Spark™ is a unified analytics engine for large-scale data processing.

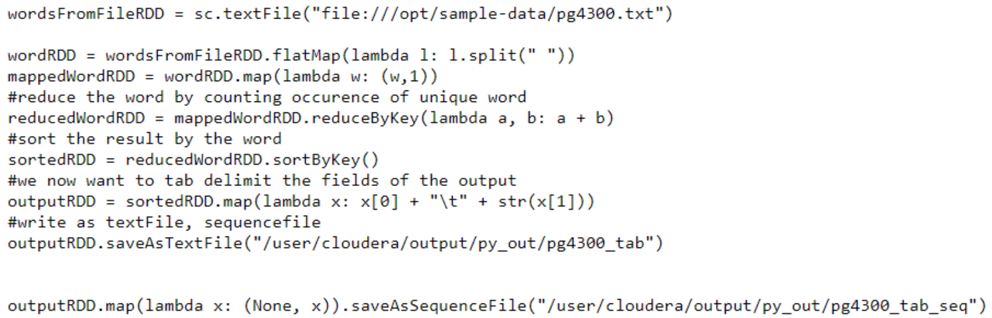
R – Resilient

D – Distributed

D - Dataset

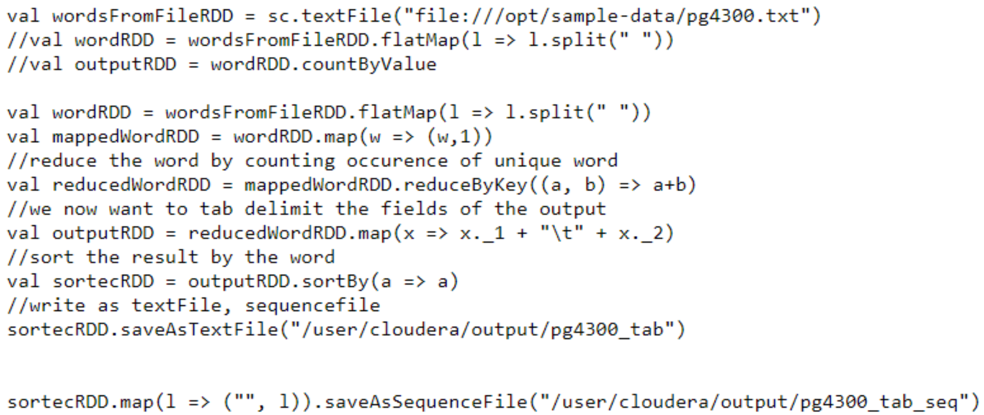


Python code



```
wordsFromFileRDD = sc.textFile("file:///opt/sample-data/pg4300.txt")

wordRDD = wordsFromFileRDD.flatMap(lambda l: l.split(" "))
mappedWordRDD = wordRDD.map(lambda w: (w,1))
#reduce the word by counting occurence of unique word
reducedWordRDD = mappedWordRDD.reduceByKey(lambda a, b: a + b)
#sort the result by the word
sortedRDD = reducedWordRDD.sortByKey()
#we now want to tab delimit the fields of the output
outputRDD = sortedRDD.map(lambda x: x[0] + "\t" + str(x[1]))
#write as textFile, sequencefile
outputRDD.saveAsTextFile("/user/cloudera/output/py_out/pg4300_tab")


outputRDD.map(lambda x: (None, x)).saveAsSequenceFile("/user/cloudera/output/py_out/pg4300_tab_seq")
```

Scala code

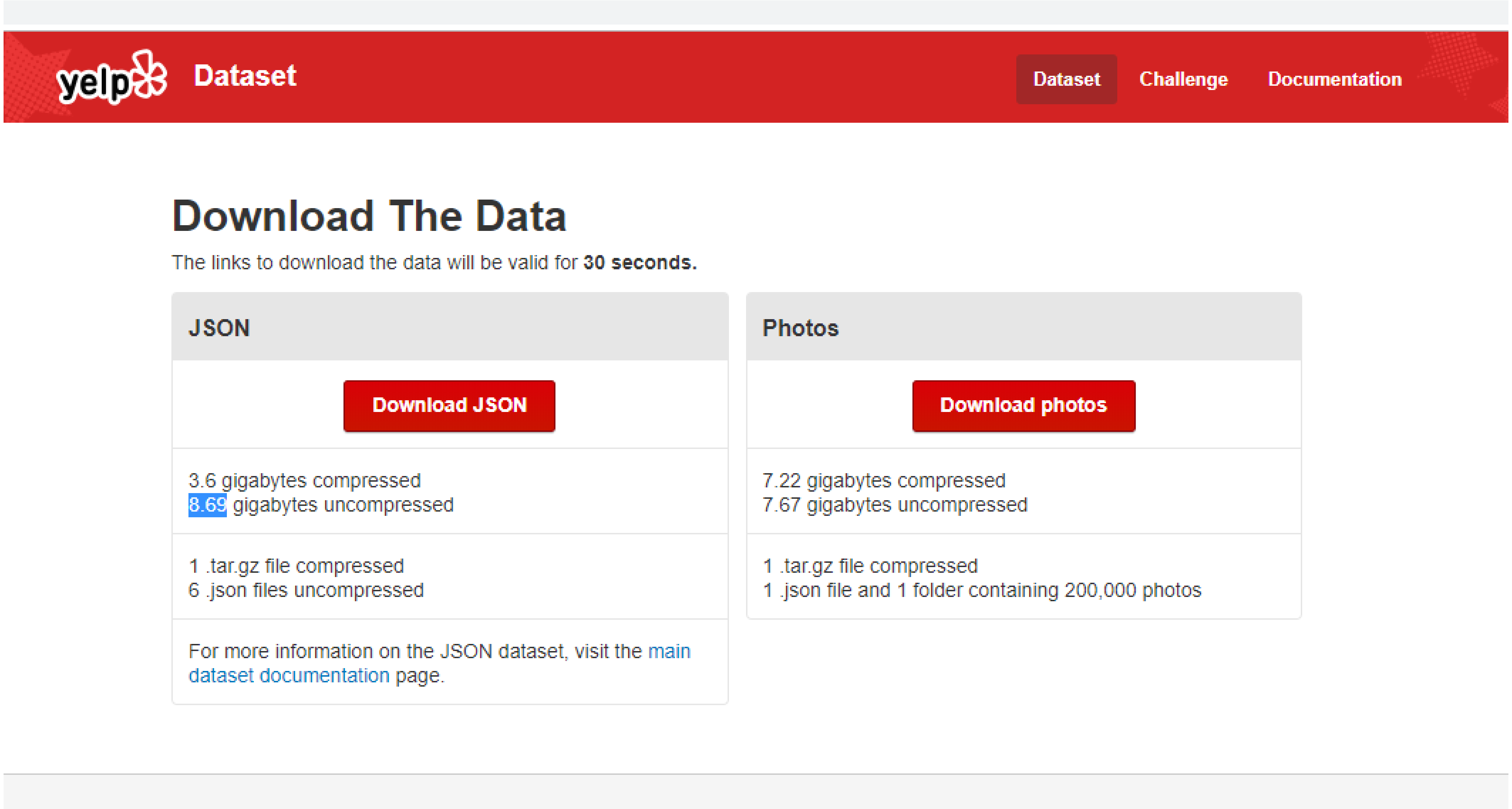


```
val wordsFromFileRDD = sc.textFile("file:///opt/sample-data/pg4300.txt")
//val wordRDD = wordsFromFileRDD.flatMap(l => l.split(" "))
//val outputRDD = wordRDD.countByValue

val wordRDD = wordsFromFileRDD.flatMap(l => l.split(" "))
val mappedWordRDD = wordRDD.map(w => (w,1))
//reduce the word by counting occurence of unique word
val reducedWordRDD = mappedWordRDD.reduceByKey((a, b) => a+b)
//we now want to tab delimit the fields of the output
val outputRDD = reducedWordRDD.map(x => x._1 + "\t" + x._2)
//sort the result by the word
val sortecRDD = outputRDD.sortBy(a => a)
//write as textFile, sequencefile
sortecRDD.saveAsTextFile("/user/cloudera/output/pg4300_tab")


sortecRDD.map(l => ("", l)).saveAsSequenceFile("/user/cloudera/output/pg4300_tab_seq")
```

DATA SERIALIZATION AND STORAGE FORMATS

- □ Compression
- □ Serialization
  - □ Sequence File
  - □ Avro
  - □ Parquet
  - □ Orc

COMPRESSION



Download The Data

The links to download the data will be valid for **30 seconds.**

JSON

3.6 gigabytes compressed
8.69 gigabytes uncompressed

1 .tar.gz file compressed
6 .json files uncompressed

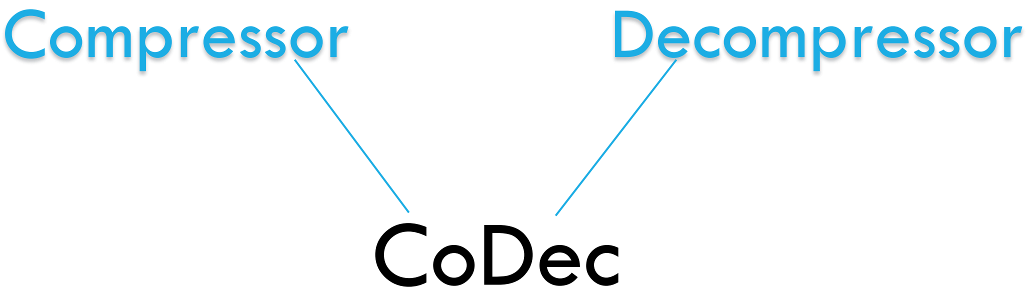
For more information on the JSON dataset, visit the main dataset documentation page.

Photos

7.22 gigabytes compressed
7.67 gigabytes uncompressed

1 .tar.gz file compressed
1 .json file and 1 folder containing 200,000 photos

COMPRESSION



- Deflate
- Snappy
- Bzip2
- Gzip
- Lz4
- Zlib
- …

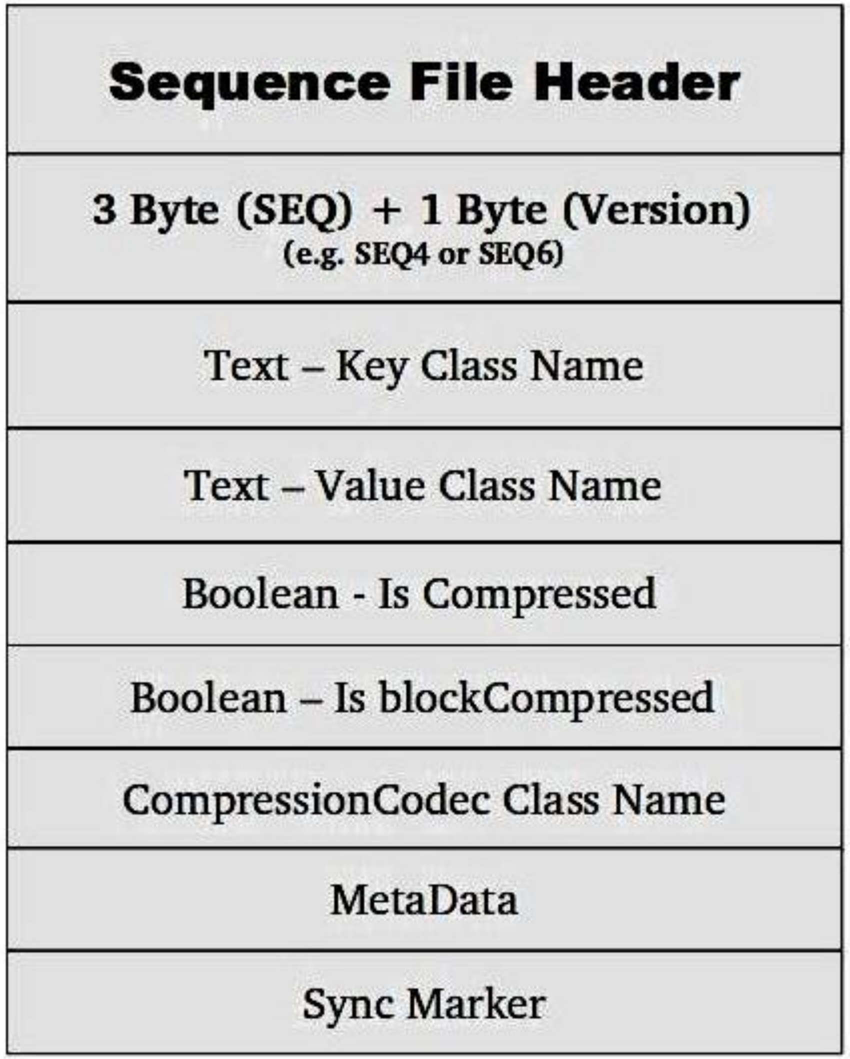
Codec	Splittable	Compression Ratio	Speed
Snappy		High	Medium
Gzip		High	Slow
Bzip2	Yes	Highest	Quite Slow
LZO	Yes	Low	Fastest
LZ4		Low	Fastest
Zlib		High	Slow

SERIALIZATION

- Row-oriented vs Column-oriented
- Compression ratio
- Read-write performance
- Data or schema evolution
- Language compatibility

Sequence File	Avro
Parquet	ORC

SEQUENCE FILE



Sequence File Header

3 Byte (SEQ) + 1 Byte (Version)
(e.g. SEQ4 or SEQ6)

Text – Key Class Name

Text – Value Class Name

Boolean - Is Compressed

Boolean – Is blockCompressed

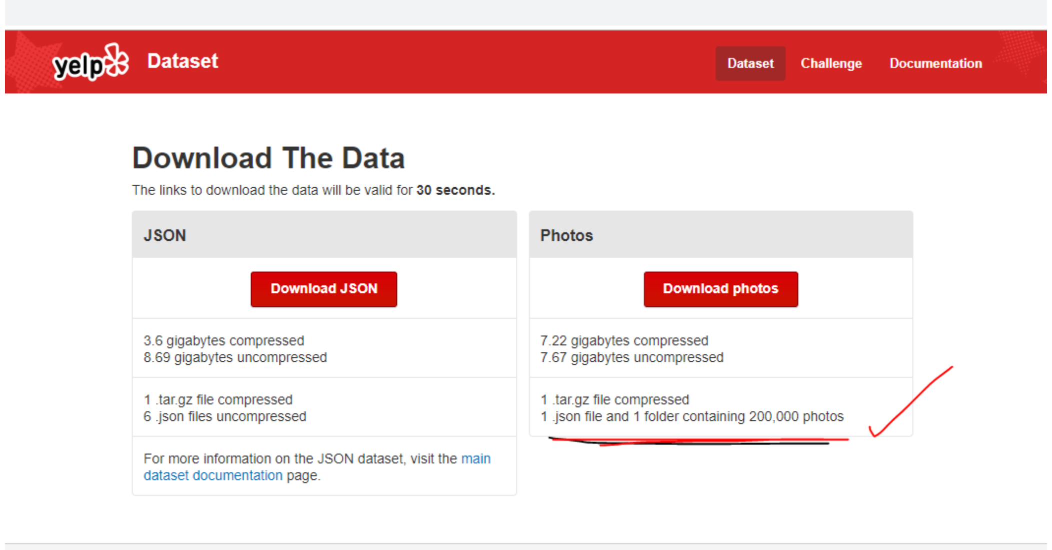
CompressionCodec Class Name

MetaData

Sync Marker

- Data serialization format for data storage that was built for the hadoop platform
- Flat binary file containing key-value pairs.
- Data type for either key or values must be a hadoop i/o primitive – writable.
- Very efficient for Map-Reduce applications.
- Support compression at either record or block level.
- Very good for storing small files.
- Supports only the java programming language
- Part of the Hadoop project

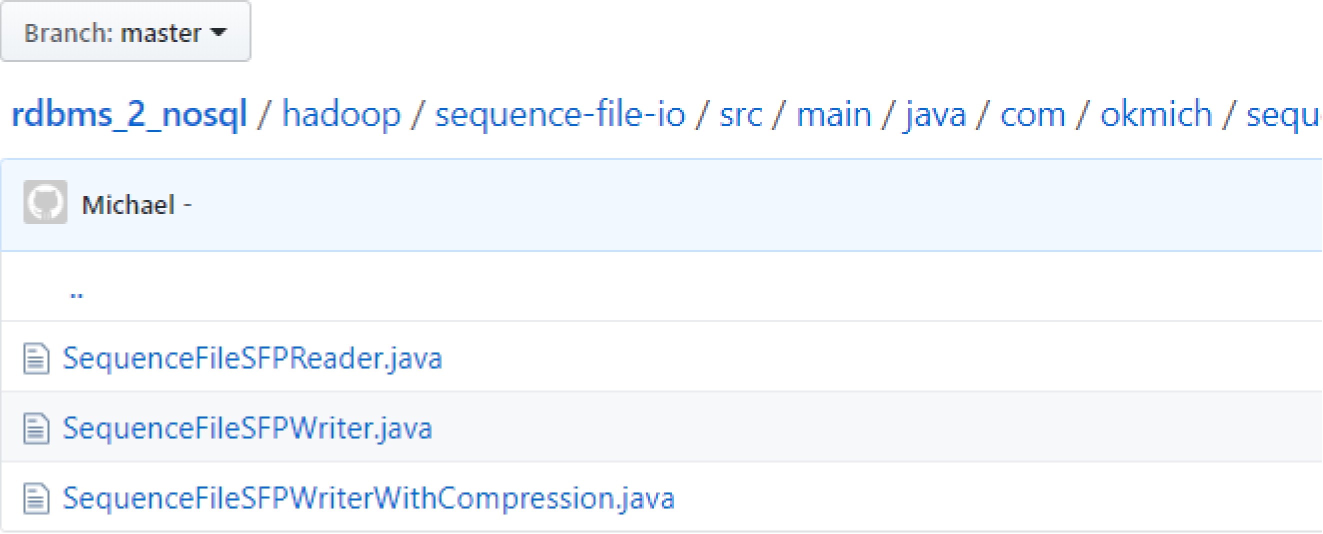
SEQUENCE FILE



Download The Data

The links to download the data will be valid for **30 seconds.**

JSON	Photos
Download JSON	Download photos
3.6 gigabytes compressed 8.69 gigabytes uncompressed	7.22 gigabytes compressed 7.67 gigabytes uncompressed
1 .tar.gz file compressed 6 .json files uncompressed	1 .tar.gz file compressed 1 .json file and 1 folder containing 200,000 photos
For more information on the JSON dataset, visit the main dataset documentation page.	



rdbms_2_nosql / hadoop / sequence-file-io / src / main / java / com / okmich / sequ

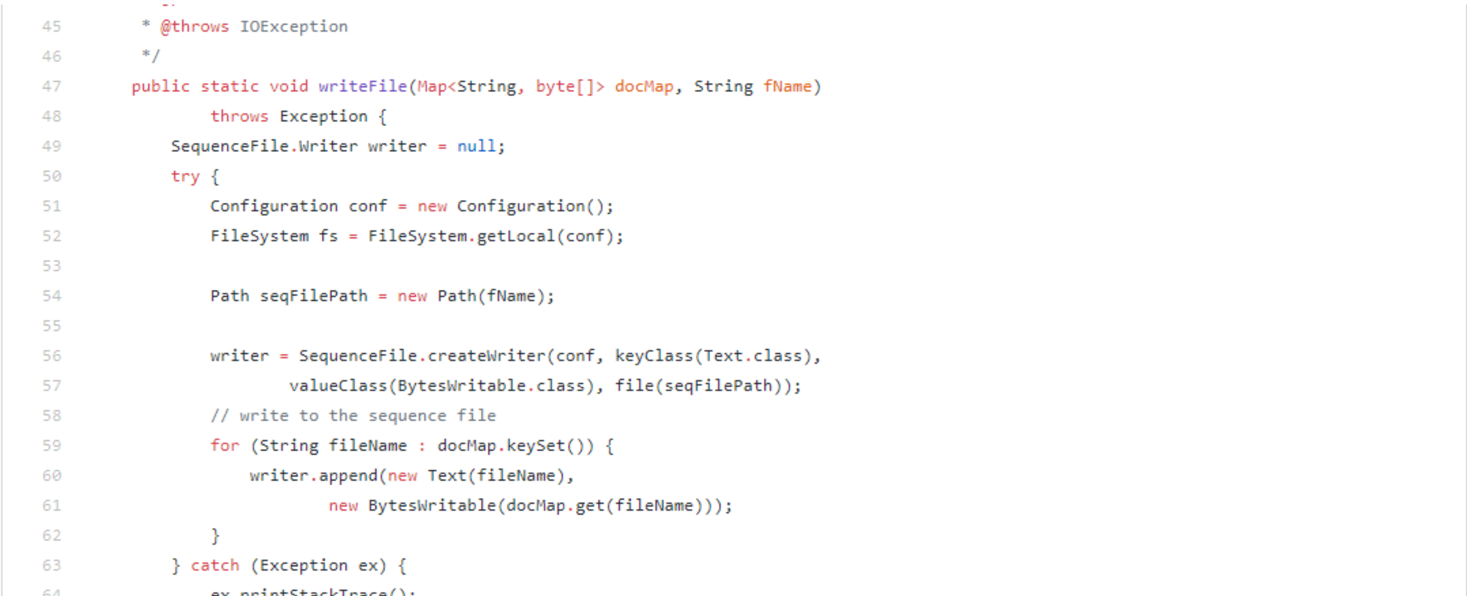
Michael -

..

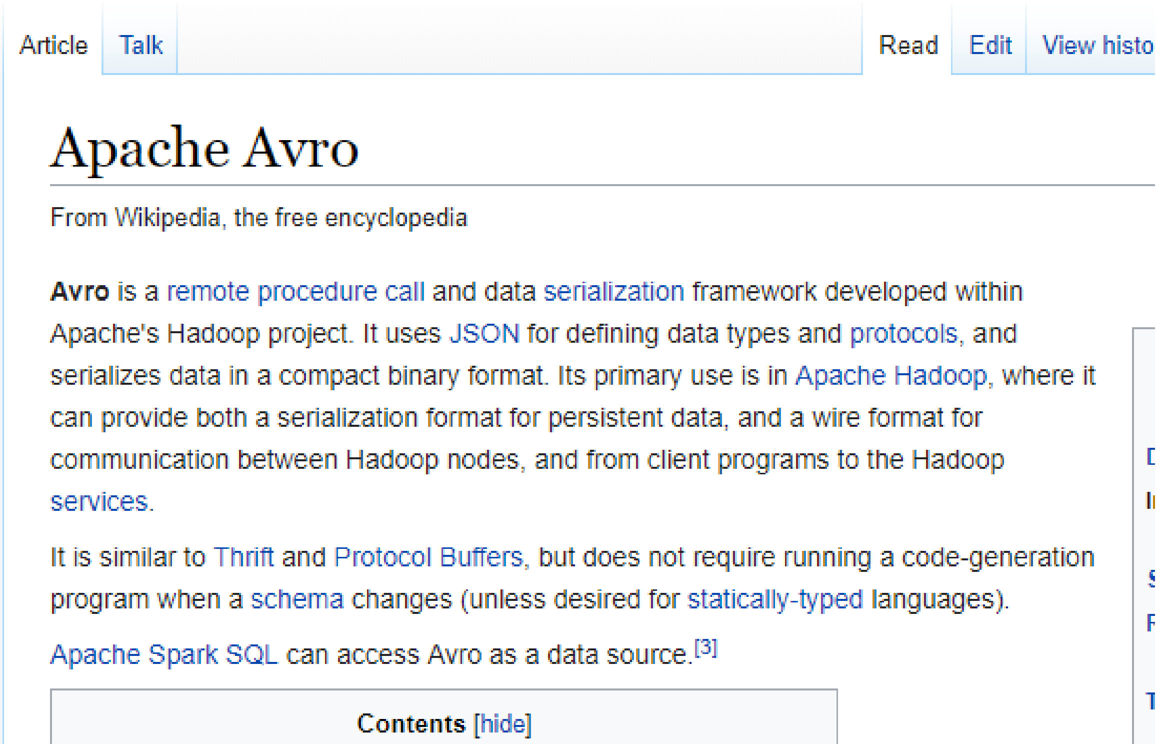
SequenceFileSFPReader.java

SequenceFileSFPWriter.java

SequenceFileSFPWriterWithCompression.java



```
     * @throws IOException
     */
    public static void writeFile(Map<String, byte[]> docMap, String fName)
            throws Exception {
        SequenceFile.Writer writer = null;
        try {
            Configuration conf = new Configuration();
            FileSystem fs = FileSystem.getLocal(conf);

            Path seqFilePath = new Path(fName);

            writer = SequenceFile.createWriter(conf, keyClass(Text.class),
                    valueClass(BytesWritable.class), file(seqFilePath));
            // write to the sequence file
            for (String fileName : docMap.keySet()) {
                writer.append(new Text(fileName),
                        new BytesWritable(docMap.get(fileName)));
            }
        } catch (Exception ex) {
            ex.printStackTrace();
```



Apache Avro

From Wikipedia, the free encyclopedia

Avro is a remote procedure call and data serialization framework developed within Apache's Hadoop project. It uses JSON for defining data types and protocols, and serializes data in a compact binary format. Its primary use is in Apache Hadoop, where it can provide both a serialization format for persistent data, and a wire format for communication between Hadoop nodes, and from client programs to the Hadoop services.

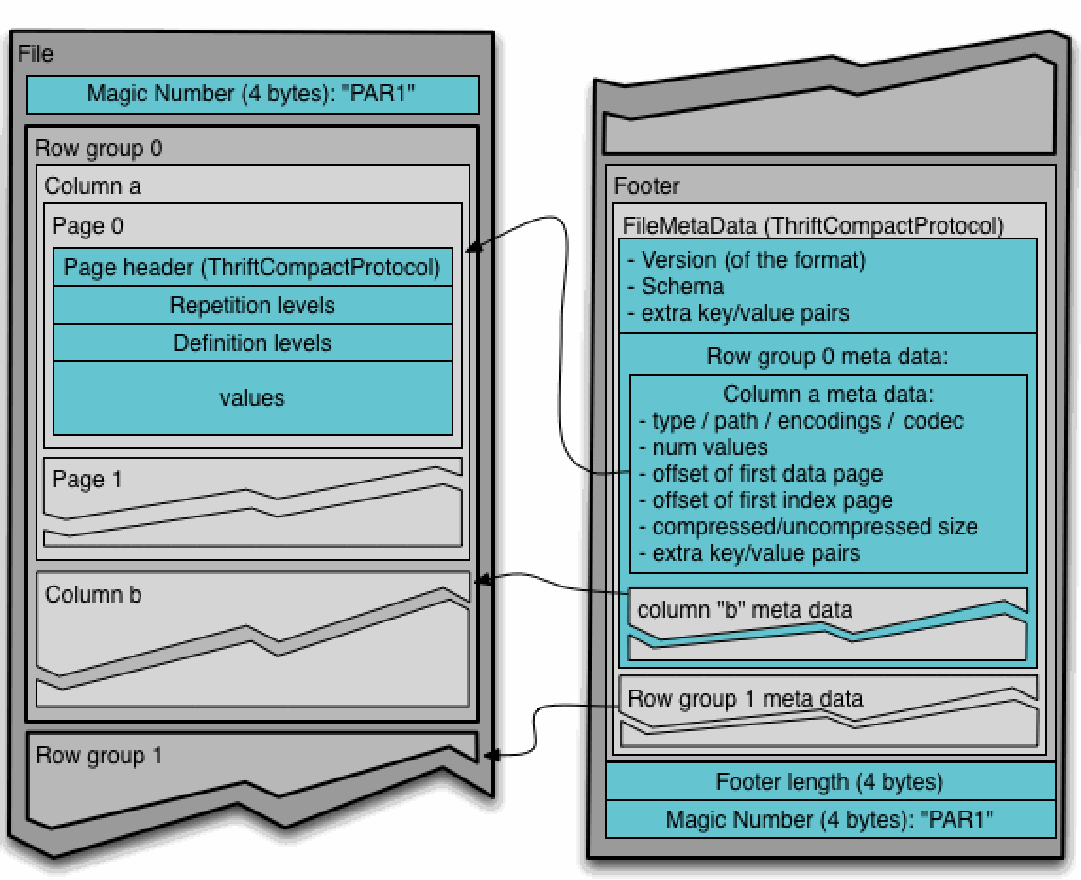
It is similar to Thrift and Protocol Buffers, but does not require running a code-generation program when a schema changes (unless desired for statically-typed languages).

Apache Spark SQL can access Avro as a data source.[3]

Contents [hide]

- Data serialization framework for data persistence as well as RPC.
- Self-describing container format
- Uses a json to describe its schema.
- Supports nested data structures as well as primitive data types.
- Language-neutral.
- Row-oriented.
- Supports internal sorting of records by fields.
- Supports deflate and snappy compression
- Has broad support in the data engineering world both as an RPC format and for storage.
- Top-level Apache project

Parquet



File

Magic Number (4 bytes): "PAR1"

Row group 0

Column a

Page 0

Page header (ThriftCompactProtocol)

Repetition levels

Definition levels

values

Page 1

Column b

Row group 1

Footer

FileMetaData (ThriftCompactProtocol)

- Version (of the format)
- Schema
- extra key/value pairs

Row group 0 meta data:

Column a meta data:

- type / path / encodings / codec
- num values
- offset of first data page
- offset of first index page
- compressed/uncompressed size
- extra key/value pairs

column "b" meta data

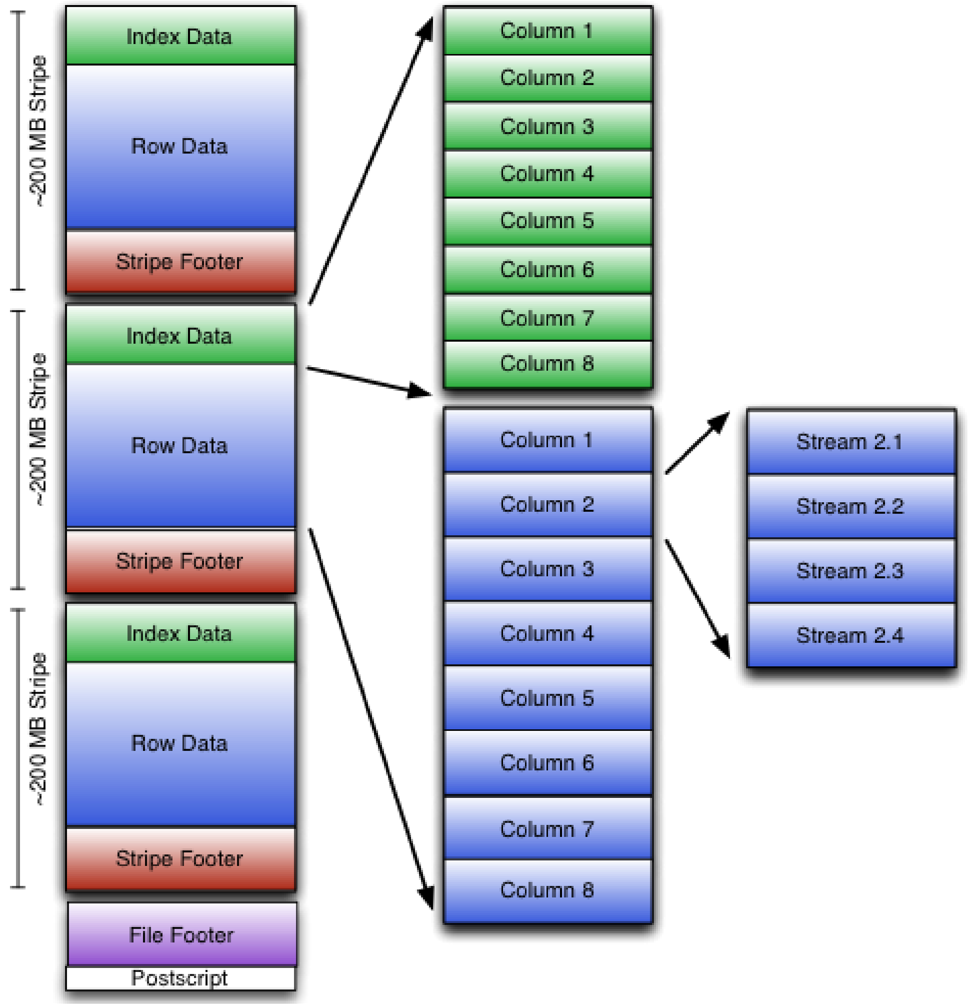
Row group 1 meta data

Footer length (4 bytes)

Magic Number (4 bytes): "PAR1"

- Data serialization format for data storage
- Self-describing container format
- Very good read performance at the cost of writes.
- Stores it schema at the footer
- Supports nested data structures as well as primitive data types.
- Column-oriented.
- Support predicate-push down.
- Can be used from various languages.
- Supports compression
- Has broad support in the data engineering world
- Top-level Apache project

Apache orc™



- Data serialization format for data storage
- Self-describing container format
- Very good read performance at the cost of writes.
- Supports nested data structures as well as primitive data types.
- Column-oriented.
- Supports internal sorting of records by fields.
- Support predicate-push down.
- Supports deflate and snappy compression
- Has broad support in the data engineering world
- Has support for in place updates.
- Top-level Apache project

END-TO-DATA BIG DATA ANALYTICS (SPARK)

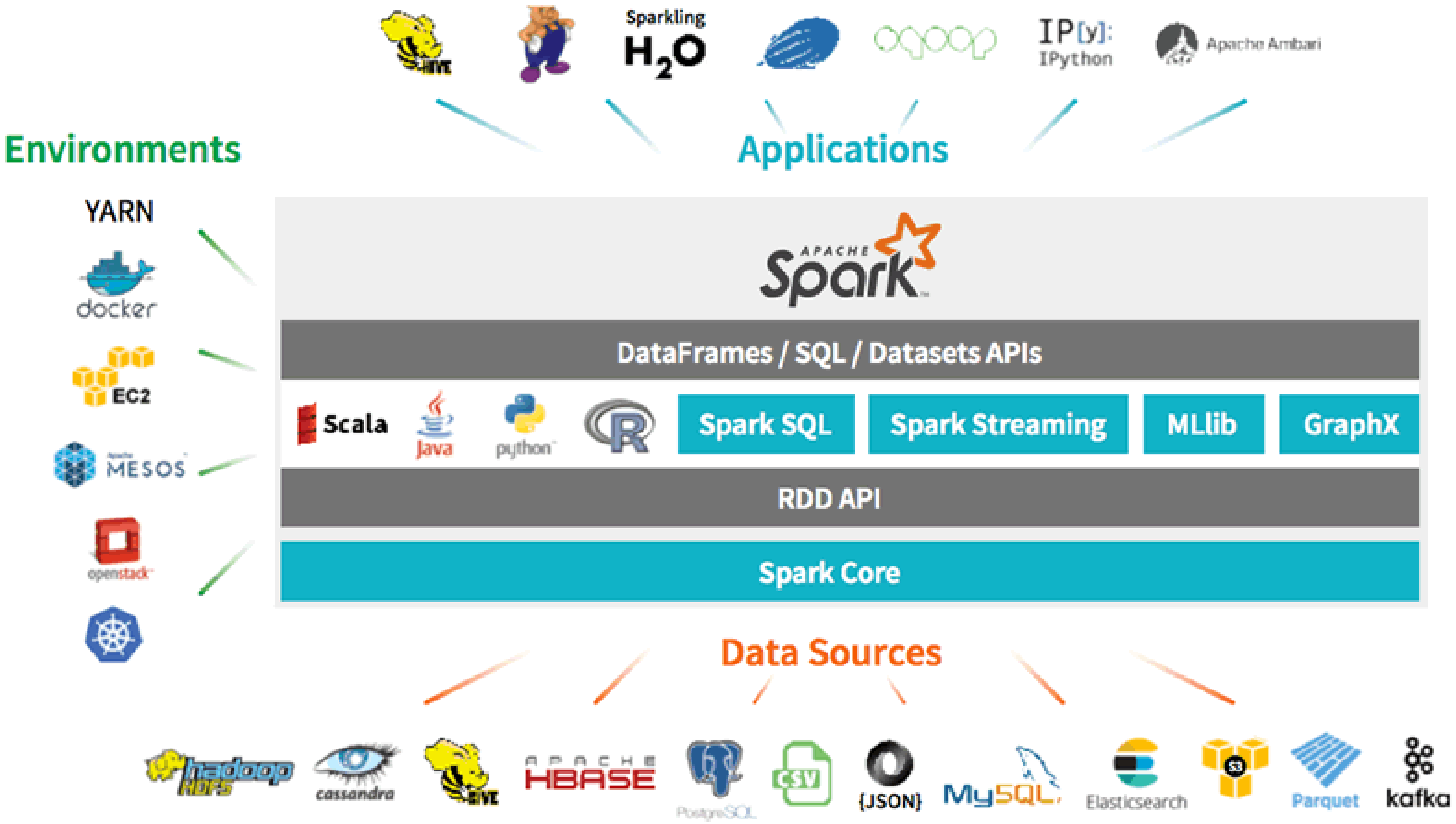
Spark SQL

Spark Streaming

Spark Structured Streaming

Spark GraphX and GraphFrames

Spark MLlib & SparkR



HADOOP: WRAP UP

Big Data Open source Ecosystem

Purpose/Tasks	Hadoop Open source tools.
Data ingestion (Batch)	HDFS Cli, Sqoop, …
Data ingestion (Real time or near real time)	Apache Flume, Kinesis, Kafka, Logstash, Splunk…
Data processing	MapReduce, Apache Pig, Spark, Crunch, Tez, Storm, Samza, Spark Streaming, Flink, Beam, Hama, Apex,…
Graph processing	Apache Giraph, Spark GraphX, Gelly, …
Pipeline Automation, scheduling & coordination	Apache Oozie, Airflow
Database	Apache HBase
Machine Learning (Data science)	Apache Mahout, SparkR, Spark ML, FlinkML, …
Cluster Management	YARN, Apache Mesos, Docker & Kubernetes
Security	Apache Sentry, Ranger, …
Collaboration & Visualization	Apache Zeppeline, Superset, …
Data serialization and storage format	Apache Avro, Parquet, Orc, Kyro
Cluster system administration	Apache Ambari, Cloudera Manager, …
OLAP	Apache Kylin, Druid, …
Data warehouse and SQL query engines	Presto, Apache Hive, Drill, Impala, HAWQ, Phoenix, Tajo, Kudu, Spark SQL, Google BigQuery…