Recap

- Data processing trend
- Big Data and its characteristics
- Applications of big data
- Various computing technologies
- History of Hadoop
- RDBMS vs Hadoop
- Major components of Hadoop cluster
- Unix commands

Agenda for today

- The Hadoop Distributed File System
- Accessing HDFS through CLI
- MapReduce detailed discussion
- Running your first MapReduce program

HDFS

- HDFS is a file system designed for storing very large files with streaming data access patterns, running on clusters of commodity hardware

Large Files

Streaming data access

Commodity hardware

HDFS Blocks

- Single unit of storage
- Default block size is 128MB
- Size of block will drive the ratio of time to read a block to the seek for a block

Exercise

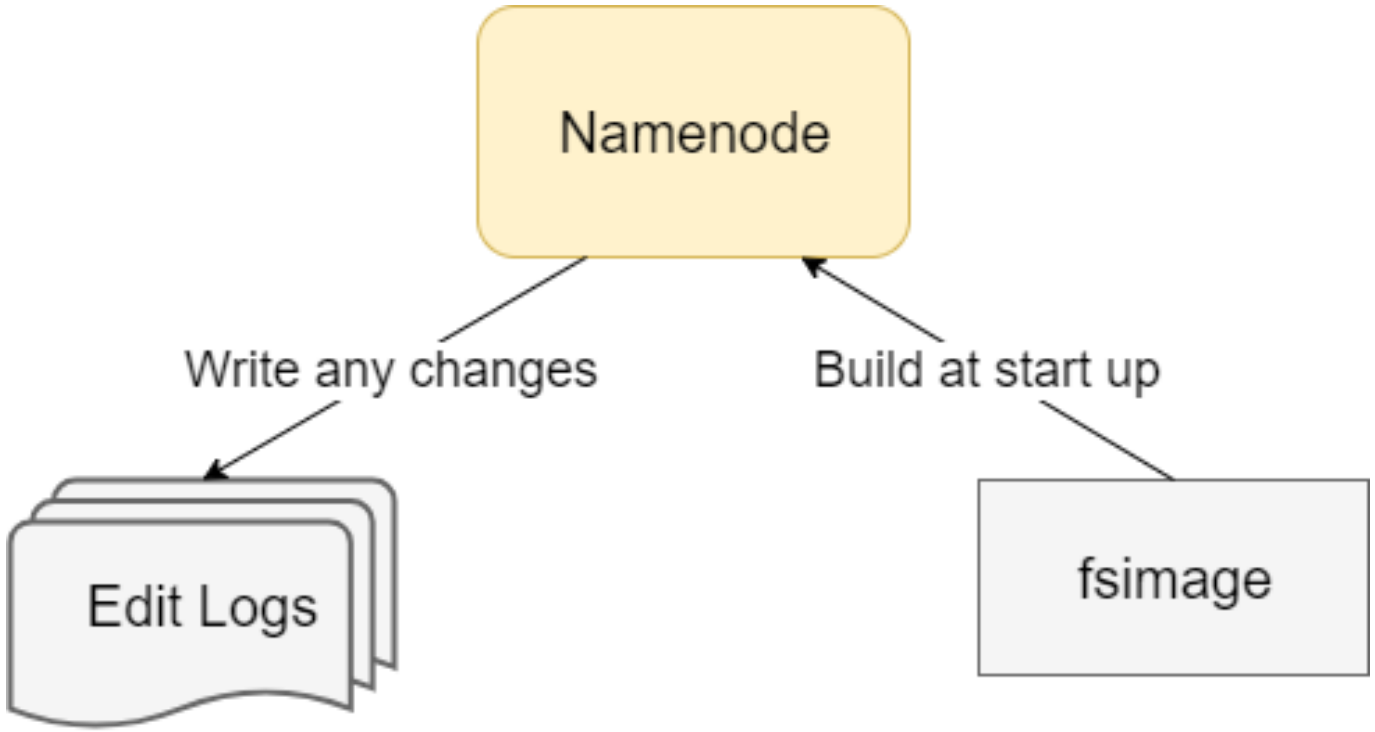
- What should be the block size to make seek time 1% of read time for given hardware configuration
  - Seek time: 10 ms
  - Data read rate: 100MB/s
- Solution:
  - Let say x MB is the block size then read time = x/100 seconds
  - To fulfill the given condition 1% of x/100 = 10 ms
  - Hence x=100 MB

Benefits of blocks

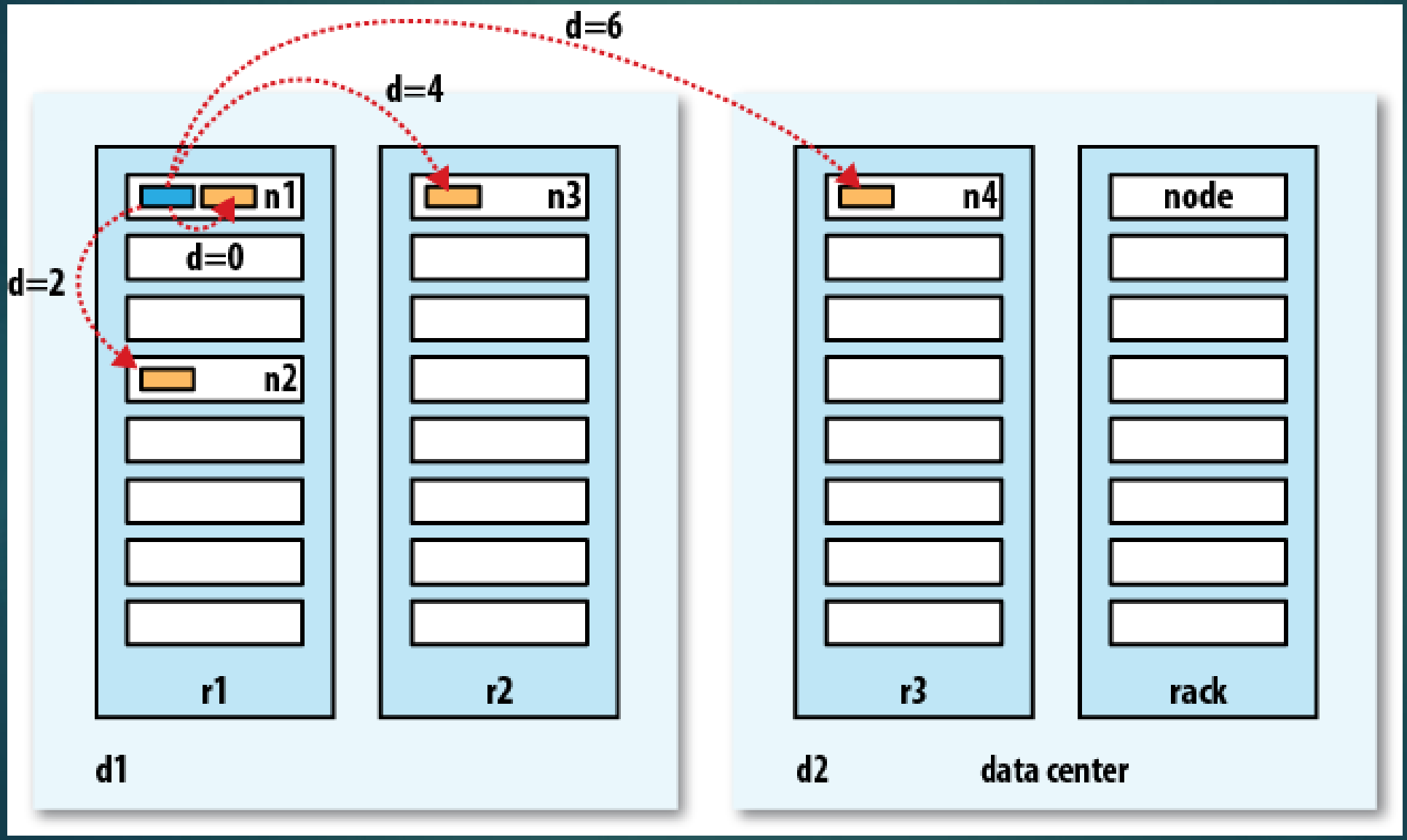
- Files can be larger than a single disk
- Simplicity at storage level as data node doesn't store any metadata
- Fault tolerance by replicating blocks

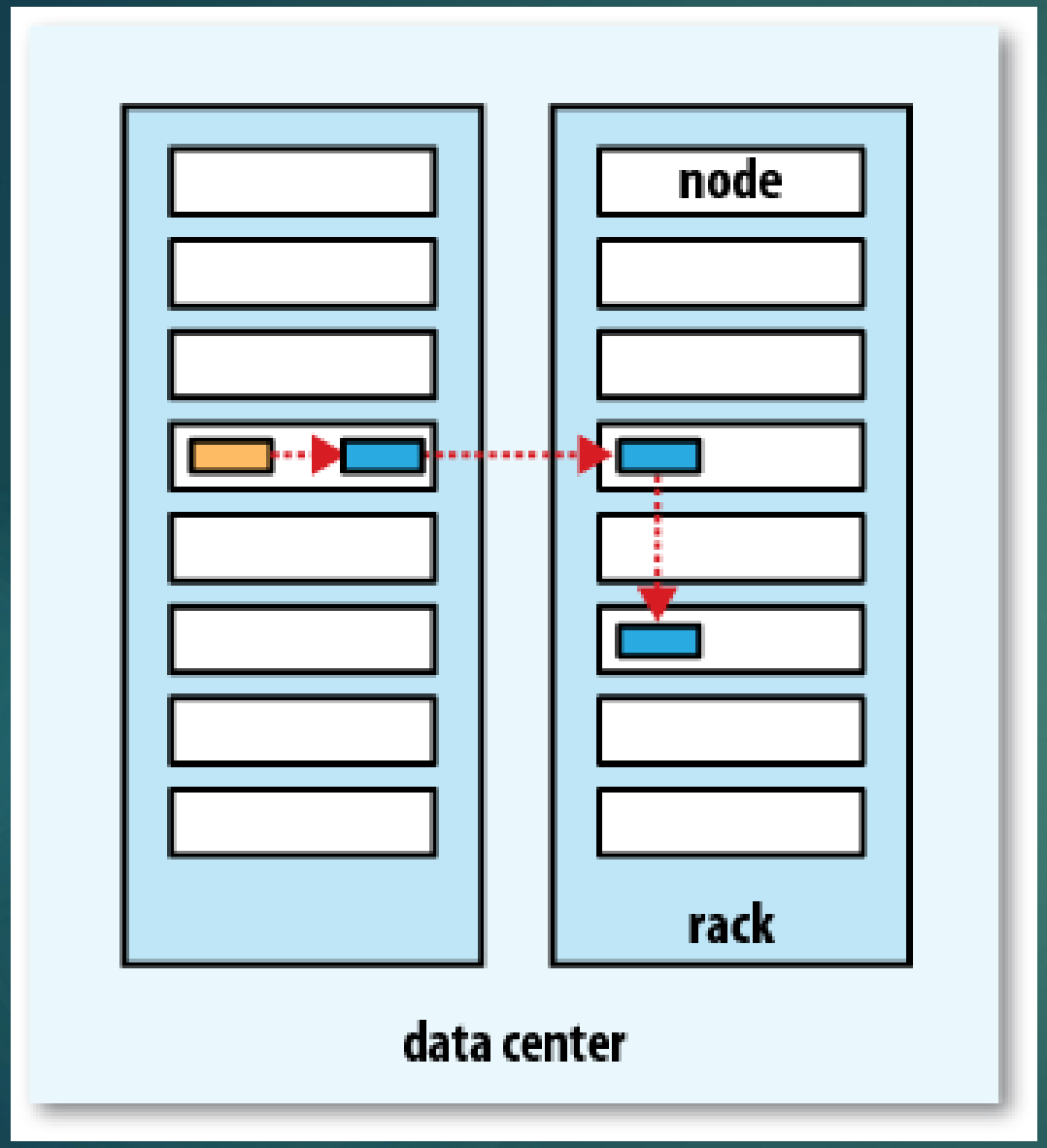
File system metadata

- Namenode stores the metadata
- Backup of metadata on secondary namenode



Network Topology





Rack awareness

HDFS CLI Copy-To-From command

hadoop fs –copyFromLocal <source> <target>

- To copy myTextFile.txt from local unix file system to HDFS under data directory

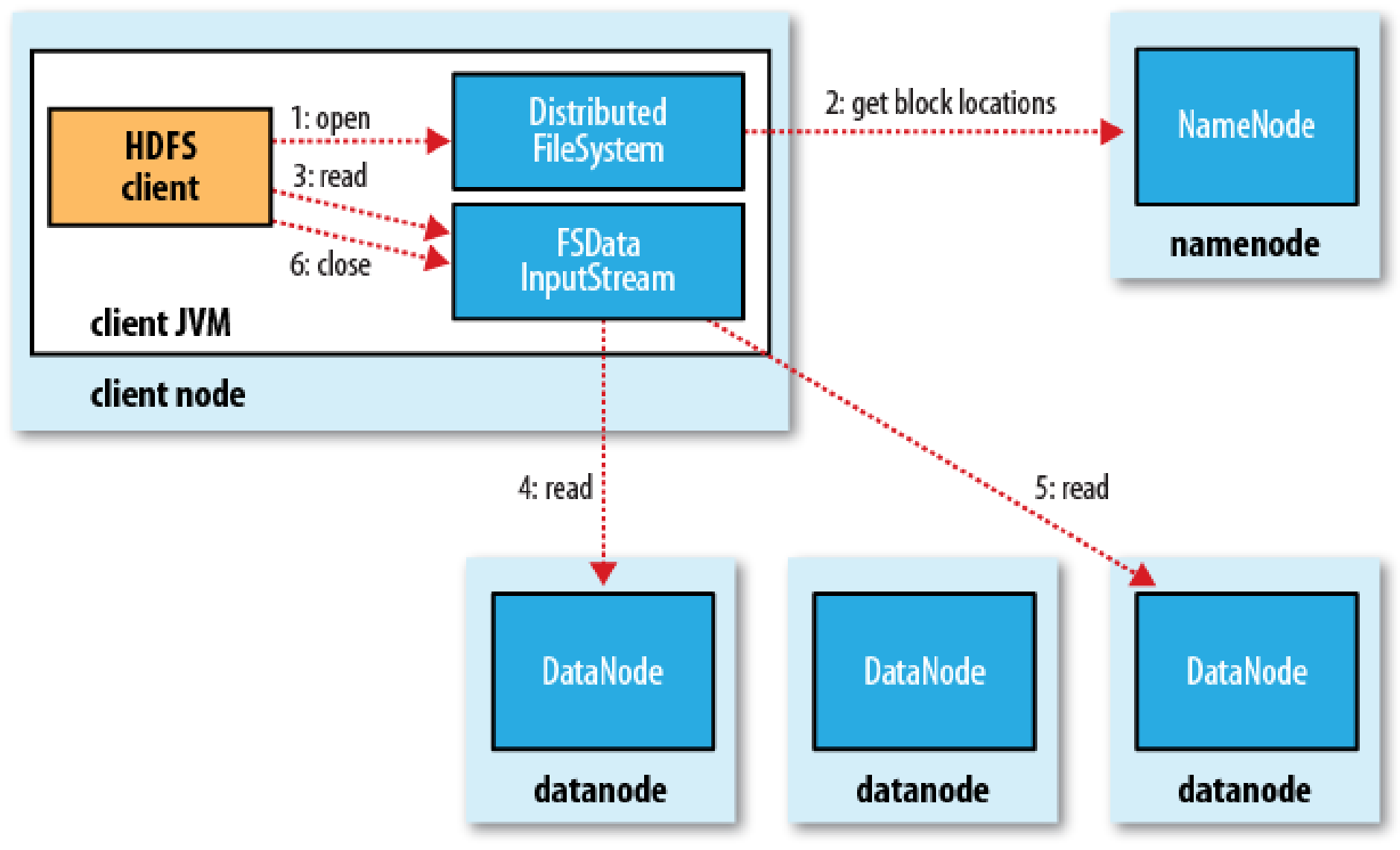
hadoop fs –copyFromLocal myTextFile.txt /data/

hadoop fs –copyToLocal <source> <target>

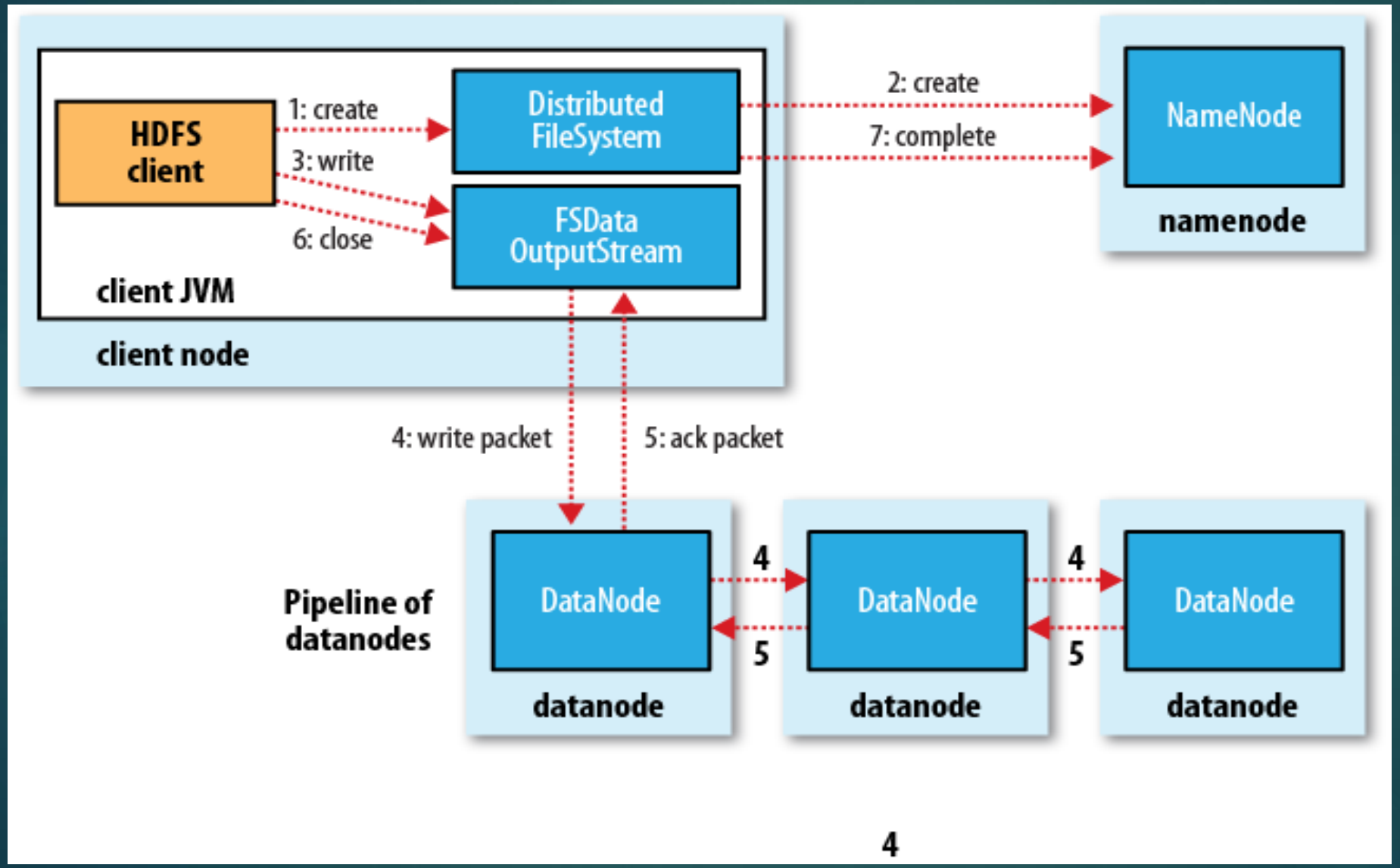
- To copy myTextFile.txt from data directory in HDFS to local file system on Desktop

hadoop fs –copyToLocal /data/myTextFile.txt ~/Desktop/

HDFS Read operation



HDFS Write operation



HDFS not made for

- Low-latency data access
  - T(N)=aN+b
- Lots of small files
  - Each file/block/directory stores around 150 bytes of metadata. Hence 1 million files each of one block will consume 300 MB of storage on Namenode
- Multiple writers, arbitrary file modifications

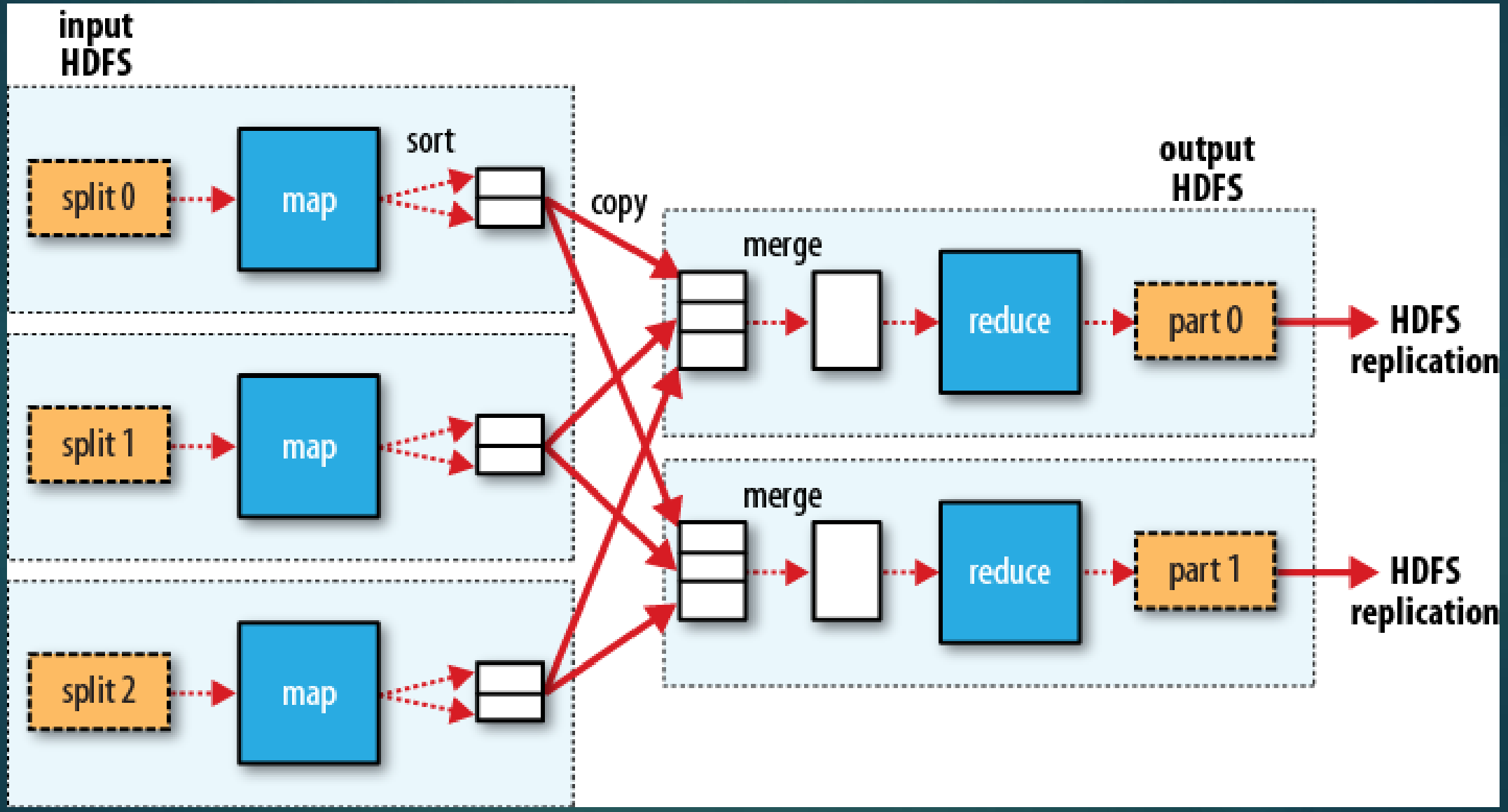
Exercise

- Calculate the memory(RAM) requirement on Namenode for given cluster configurations
  - Cluster size: 200 nodes
  - Storage capacity of each node: 24 TB
  - Block size: 128MB
  - Replication factor: 3
  - Metadata storage size for each block: 150 bytes
- Solution
  - (200*24*10^12*150)/(128*10^6*3)

MapReduce

- Two major phases: Map and Reduce
- Notion of <Key, Value> pairs
- Divides job into multiple tasks
- Map: extract important information from each record
- Reduce: Aggregate, Summarize, Filter, Transform

MapReduce Stages



Map Tasks

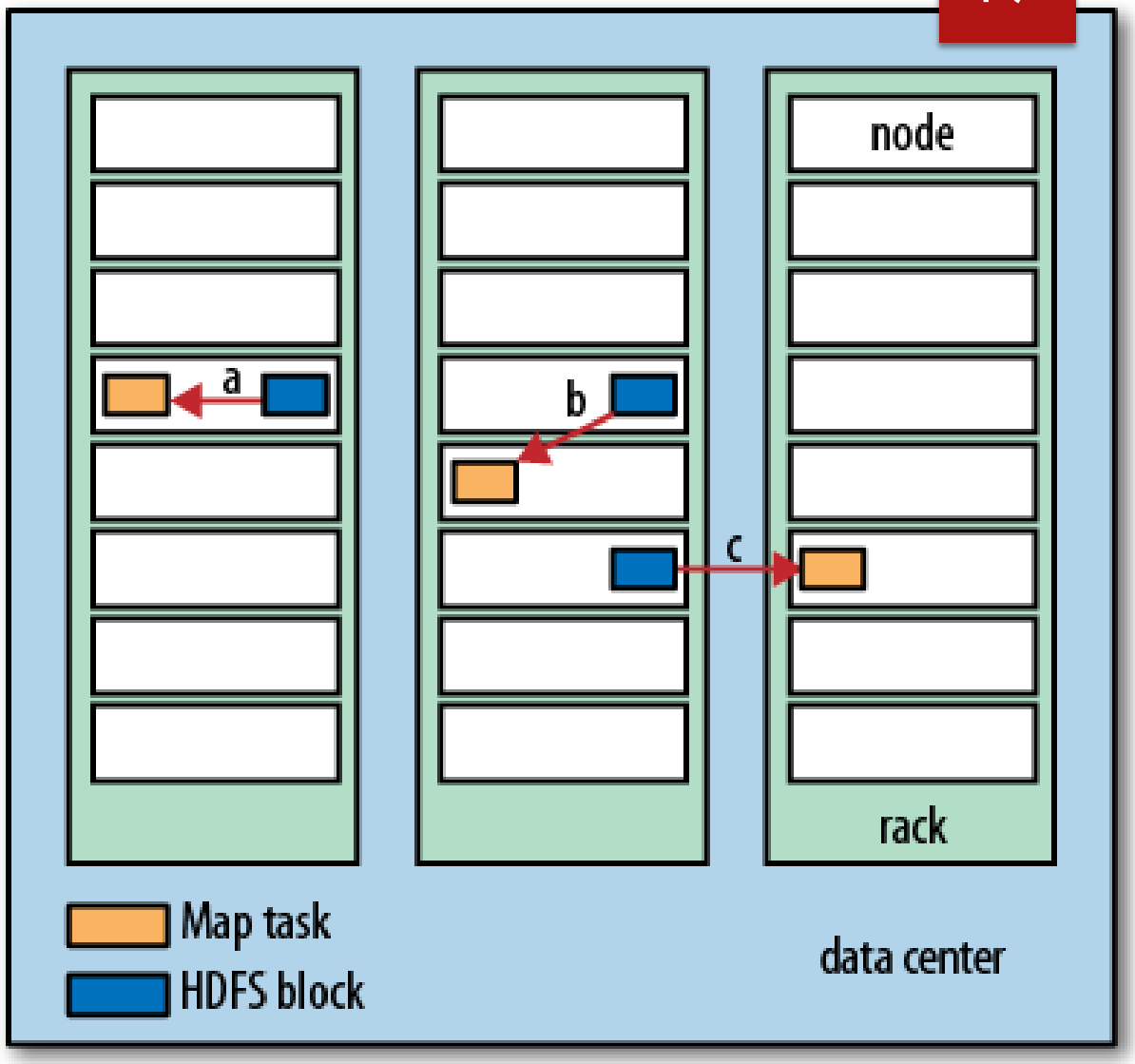
- What is a good approach to decide how many map tasks a job should launch?
- Less number of big tasks

  vs

  higher number of small tasks
- Normally same as input data blocks

Task to node mapping

- Notion of data locality

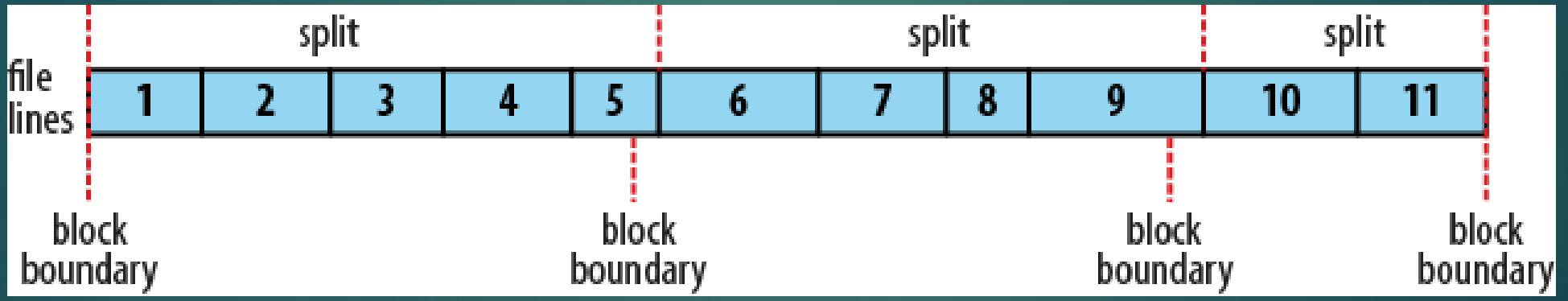


Input formats

Input format	Description
TextInputFormat	Read Text file line by line. Key is offset and value is record text
KeyValueTextInputFormat	Tab separated key values from a text file
SequenceFileInputFormat<K,V>	Hadoop's file format

Input Splits

- Blocks are of fixed size
- Good chances of records being split between two block



Reduce Tasks

- Can be configured by programmer
- Normally same as #datanodes participating in execution
- Input Key and Value type should be same as output type of mapper
- One output file per reducer under output directory
- Generates exception if output directory already exists. Any guess why?

Output Formats

Output Format	Description
TextOutputFormat<K,V>	Tab separated key value pairs in plain text format. One record per key value pair
SequenceFileOutputFormat<K,V>	Hadoop's Sequence file format
NullOutputFormat<K,V>	Nothing. Helps in map only job

MapReduce: Mapper code

```
public class WebHitCounterMapper extends
Mapper<Input Key, Input Value, Output Key, Output Value>
{

    public void map(Input Key, Input Value, Context context)
throws IOException, InterruptedException {

     <MAP Logic goes here>

     context.write(Output Key, Output Value)
    }
}
```

MapReduce: Reducer code

```
public class WebHitCounterReducer extends
Reducer<Input Key, Input Value, Output Key, Output Value>
{
    public void reduce(Input Key, Iterable<Value Data type>
values,Context context) throws IOException,
InterruptedException {

     <REDUCE logic goes here>

        context.write(Output Key, Output Value);
    }
}
```

MapReduce: Driver Code


```
public class WebHitCounterMain {
    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Daily Web Hit Counter");


    }
}
```

MapReduce: Driver Code


```
public class WebHitCounterMain {
    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Daily Web Hit Counter");

        job.setJarByClass(main.WebHitCounterMain.class);
        job.setMapperClass(mapper.WebHitCounterMapper.class);
        job.setReducerClass(reducer.WebHitCounterReducer.class);

    }
}
```

MapReduce: Driver Code


```
public class WebHitCounterMain {
    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Daily Web Hit Counter");

        job.setJarByClass(main.WebHitCounterMain.class);
        job.setMapperClass(mapper.WebHitCounterMapper.class);
        job.setReducerClass(reducer.WebHitCounterReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

    }
}
```

MapReduce: Driver Code


```
public class WebHitCounterMain {
    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Daily Web Hit Counter");

        job.setJarByClass(main.WebHitCounterMain.class);
        job.setMapperClass(mapper.WebHitCounterMapper.class);
        job.setReducerClass(reducer.WebHitCounterReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

    }
}
```

MapReduce: Driver Code


```
public class WebHitCounterMain {
    public static void main(String[] args) throws Exception {

        Configuration conf = new Configuration();
        Job job = Job.getInstance(conf, "Daily Web Hit Counter");

        job.setJarByClass(main.WebHitCounterMain.class);
        job.setMapperClass(mapper.WebHitCounterMapper.class);
        job.setReducerClass(reducer.WebHitCounterReducer.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

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

Programming Exercise

Challenges with Hadoop 1

- Applications were limited to MapReduce implementations only
- Namenode machine crash or maintenance activity
- Namespace scaling
- Backup and Recovery
- Batch oriented architecture
- Support for various file formats
- Dual responsibilities of Job tracker

Image Ref: https://www.greycampus.com/blog/big-data/top-differences-between-hadoop-1-0-and-hadoop-2-2

Hadoop 2

- Support for other data processing engines
- High Availability
- HDFS Federation
- HDFS Snapshot
- Introduced Streaming and Interactive analysis
- Support for various file formats
- Yarn

YARN

- Yet Another Resource Negotiator

MapReduce 1	YARN
Job Tracker	Resource Manager, Application Master and Timeline server
Task Tracker	Node Manager
Slot	Containers

YARN model

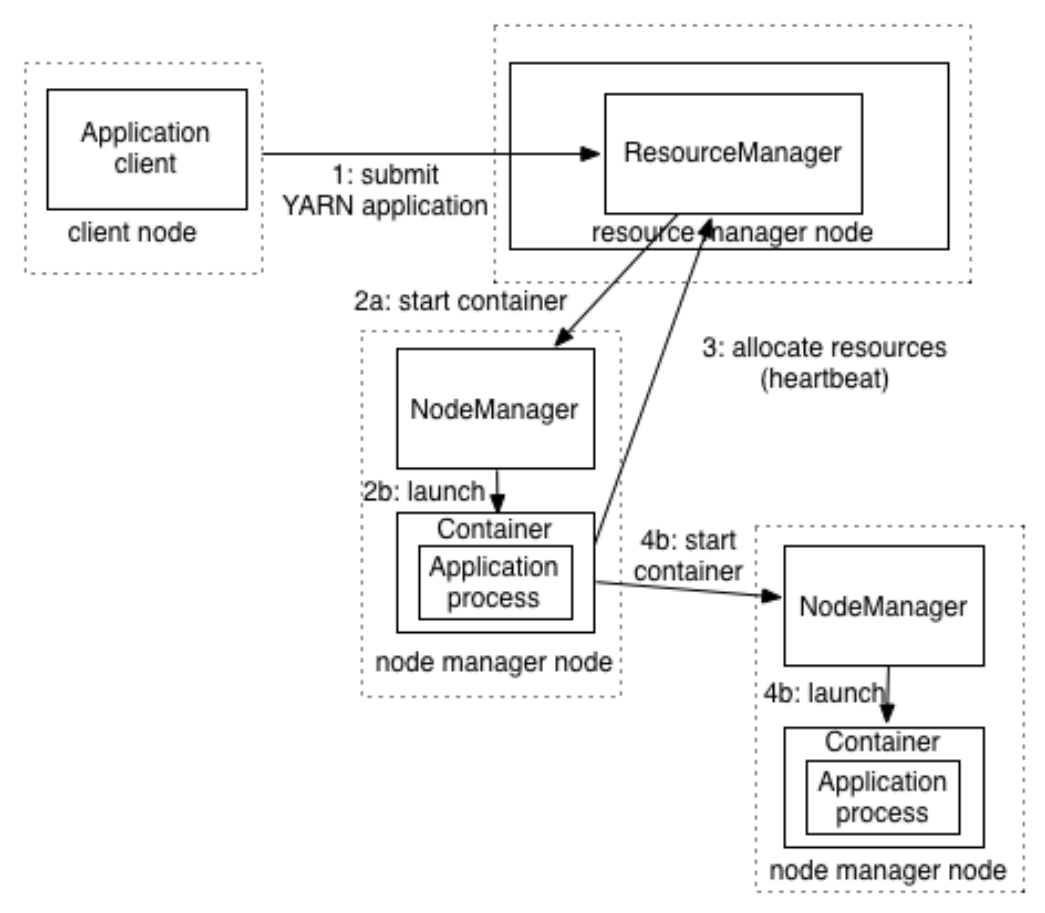


Image Ref: Hadoop definitive guide 4th edition

Pros of YARN

- Scalability
- Availability
- Utilization
- Multitenancy

Reference

- Image reference: Hadoop definitive guide 4th edition

Additional resources

- Reference Book
- Hadoop: The definitive guide by Tom White (Weblink)
- MapReduce google paper
- https://ai.google/research/pubs/pub62
- Filesystem metadata on Namenode
- https://hortonworks.com/blog/hdfs-metadata-directories-explained/
- Rack awareness
- https://data-flair.training/blogs/rack-awareness-hadoop-hdfs/
- https://community.hortonworks.com/articles/43057/rack-awareness-1.html
- Concurrency vs Parallelism
- https://stackoverflow.com/questions/1050222/what-is-the-difference-between-concurrency-and-parallelism
- Latency vs Throughput
- https://stackoverflow.com/questions/16718095/high-throughput-vs-low-latency-in-hdfs