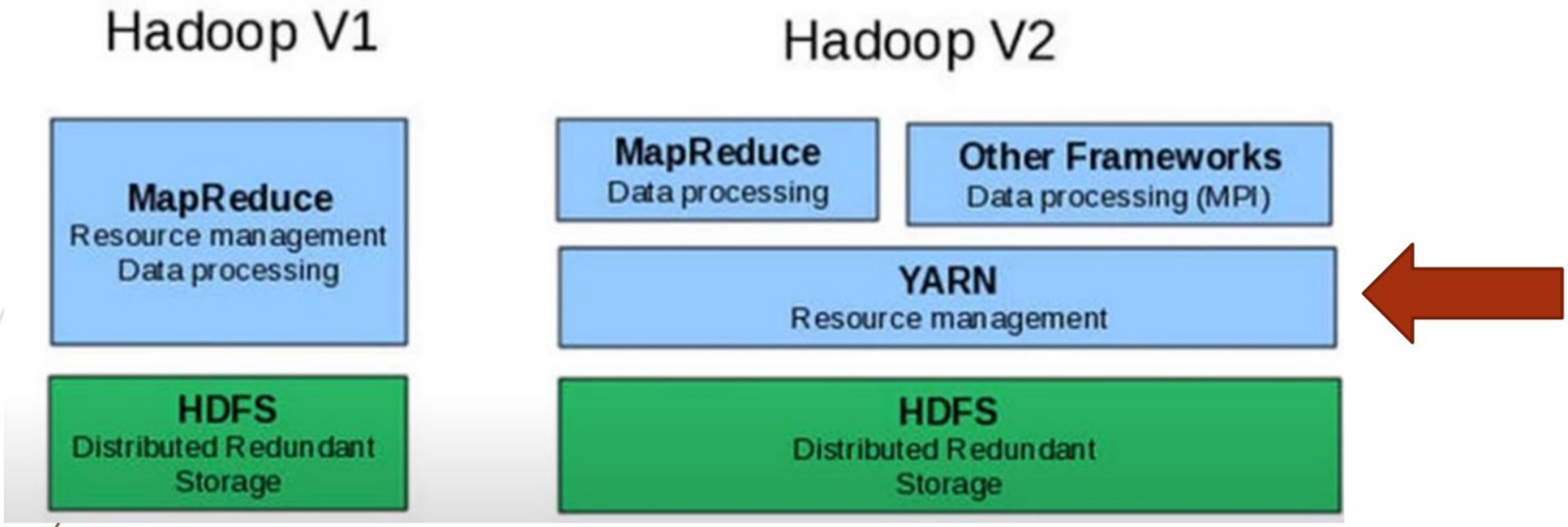
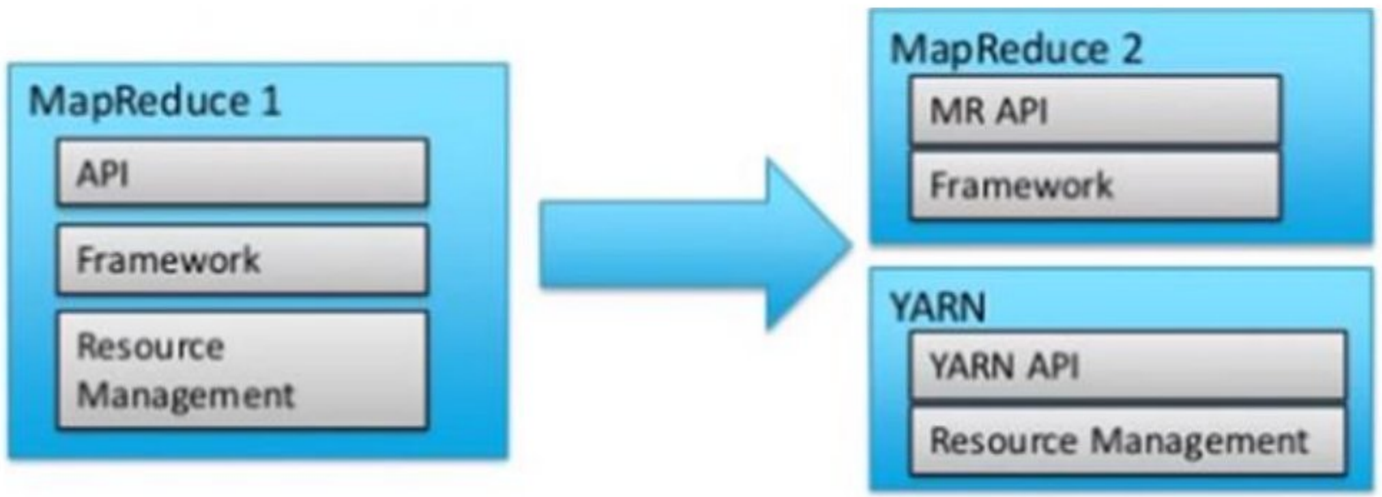
Chapter 3:

YARN

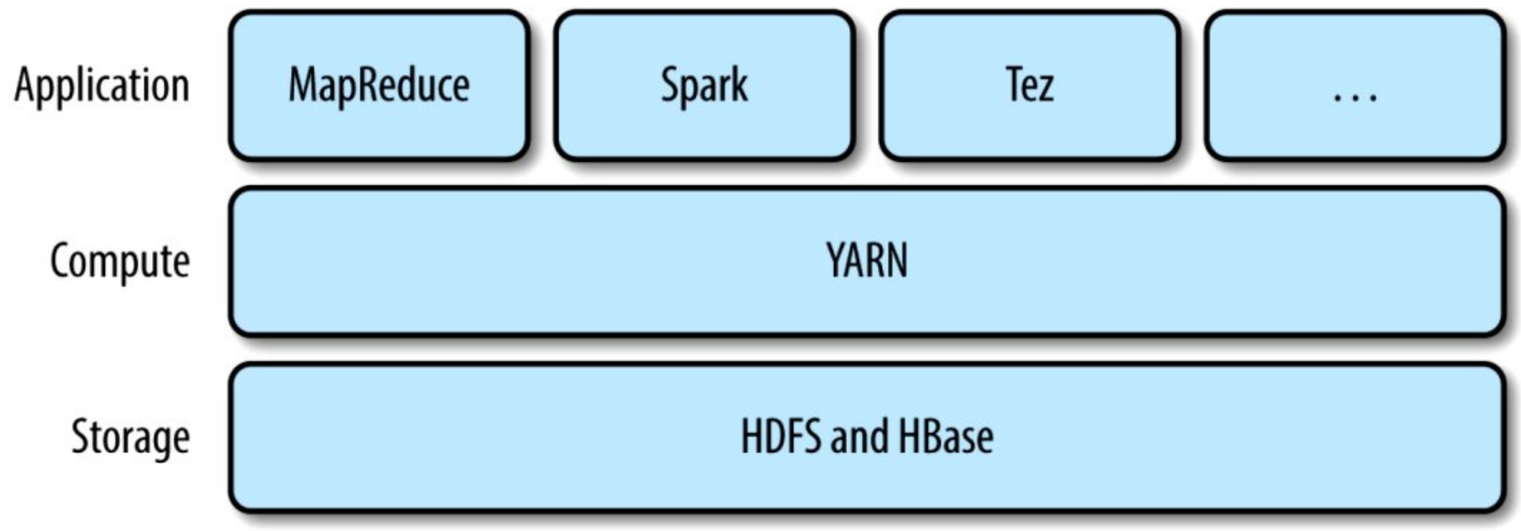
YARN


Hadoop V1
MapReduce
Resource management
Data processing
HDFS
Distributed Redundant
Storage
Hadoop V2
MapReduce
Data processing
Other Frameworks
Data processing (MPI)
YARN
Resource management
HDFS
Distributed Redundant
Storage


MapReduce 1
API
Framework
Resource
Management
MapReduce 2
MR API
Framework
YARN
YARN API
Resource Management

YARN

- Apache YARN: Yet Another Resource Negotiator
- It is Hadoop's cluster resource management system
- It was introduced in Hadoop 2 to improve the MapReduce implementation



YARN Application Run

- YARN core provides two services:
  - Resource manager (one per cluster): manage the use of resources across the cluster
  - Node managers: running on all the nodes in the cluster to launch and monitor containers.
- A container executes an application-specific process with a constrained set of resources (memory, CPU, and so on).

MapReduce1 vs YARN

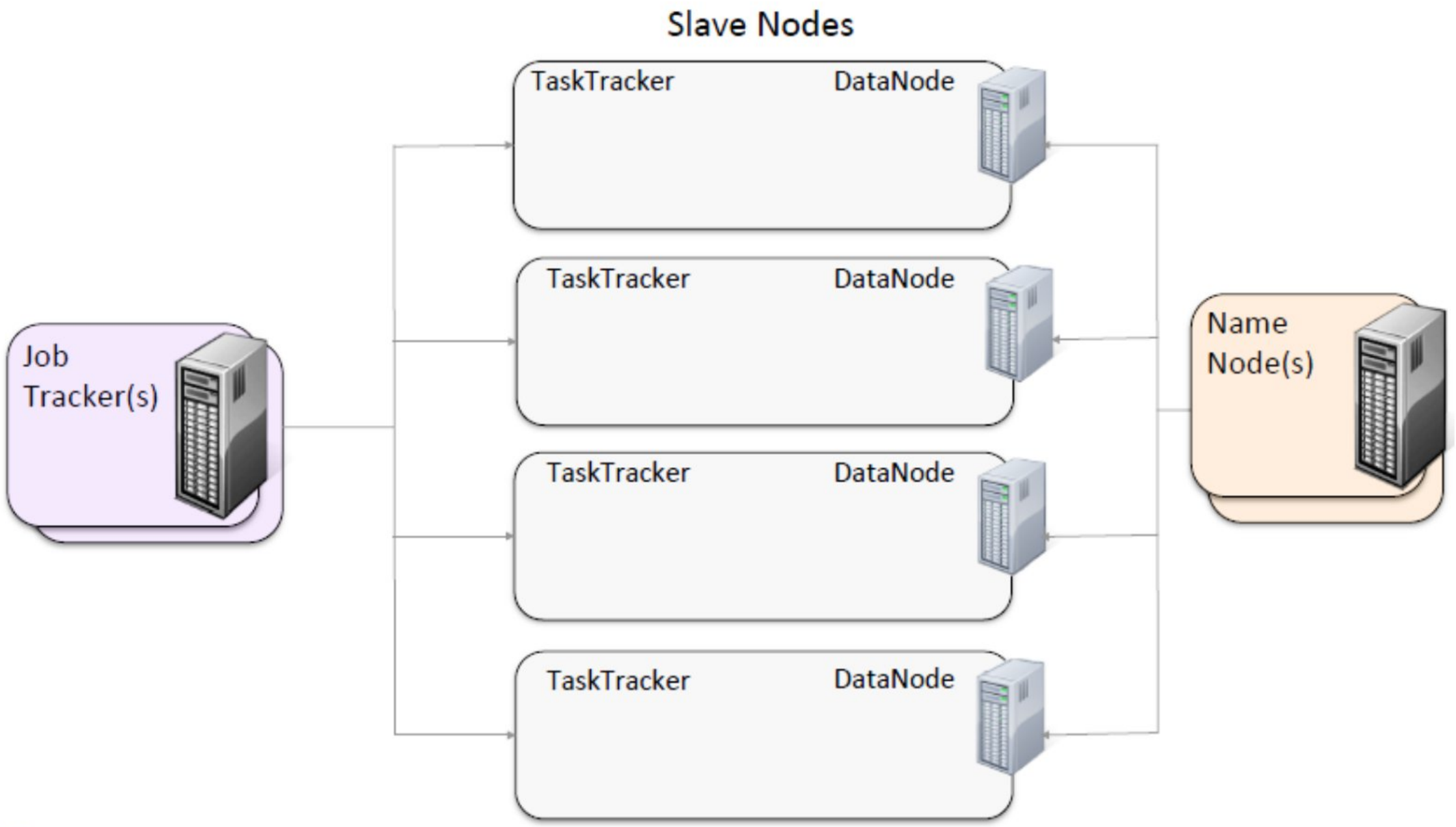
MapReduce 1	YARN
Jobtracker	Resource manager, application master, timeline server
Tasktracker	Node manager
Slot	Container

- **Benefits of using YARN:**
  - **Scalability: YARN:** scale up to 10,000 nodes and 100,000 tasks.

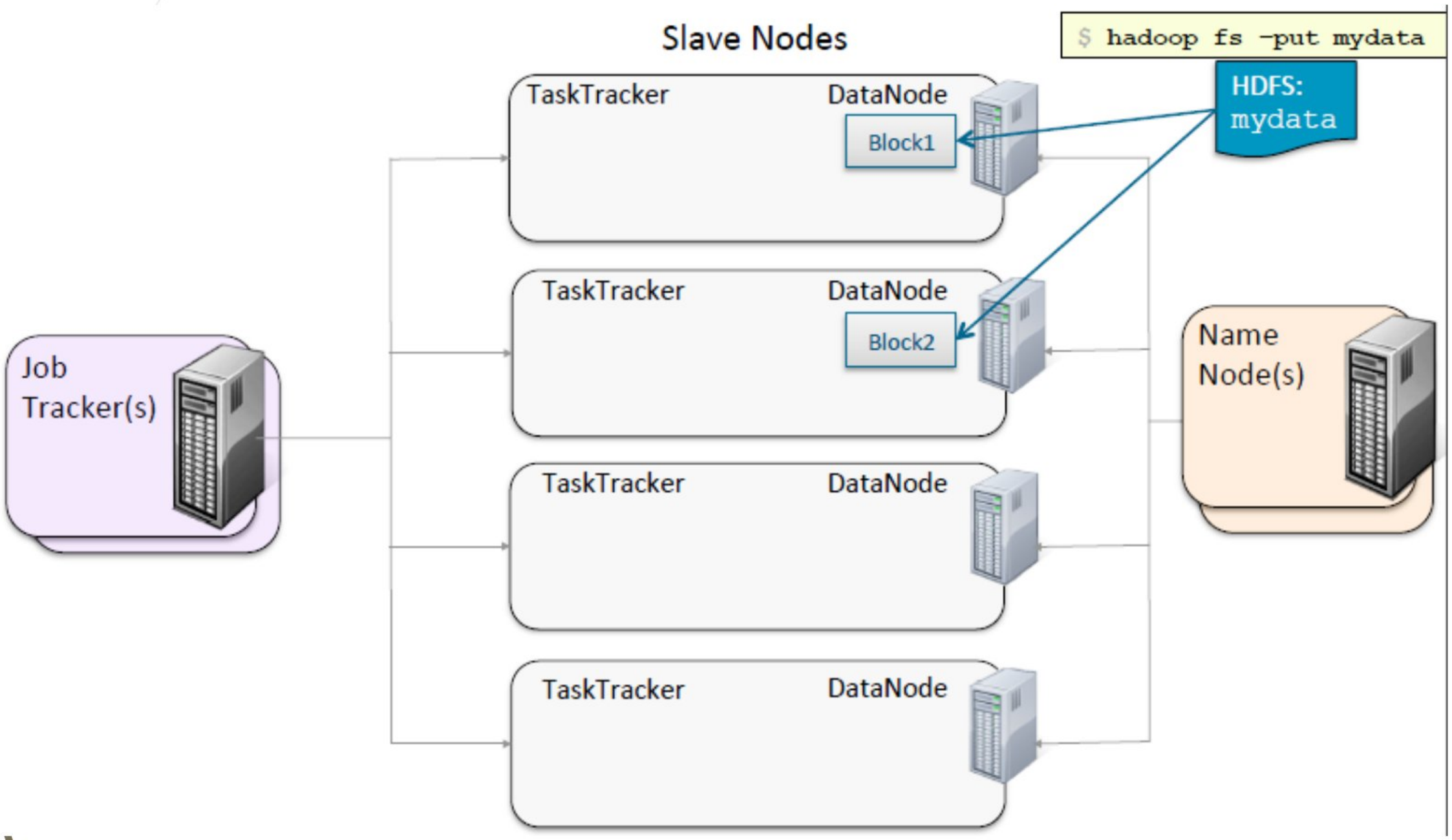
    MapReduce1: 4,000 nodes and 40,000 tasks
  - **Availability:** using YARN, ressource manager will not be overloaded thus, the cluster and the tasks will be more available
  - **Utilization: MapReduce1** allocates static slots (map and reduce slots)

    YARN allocates dynamic container (fine-grained)

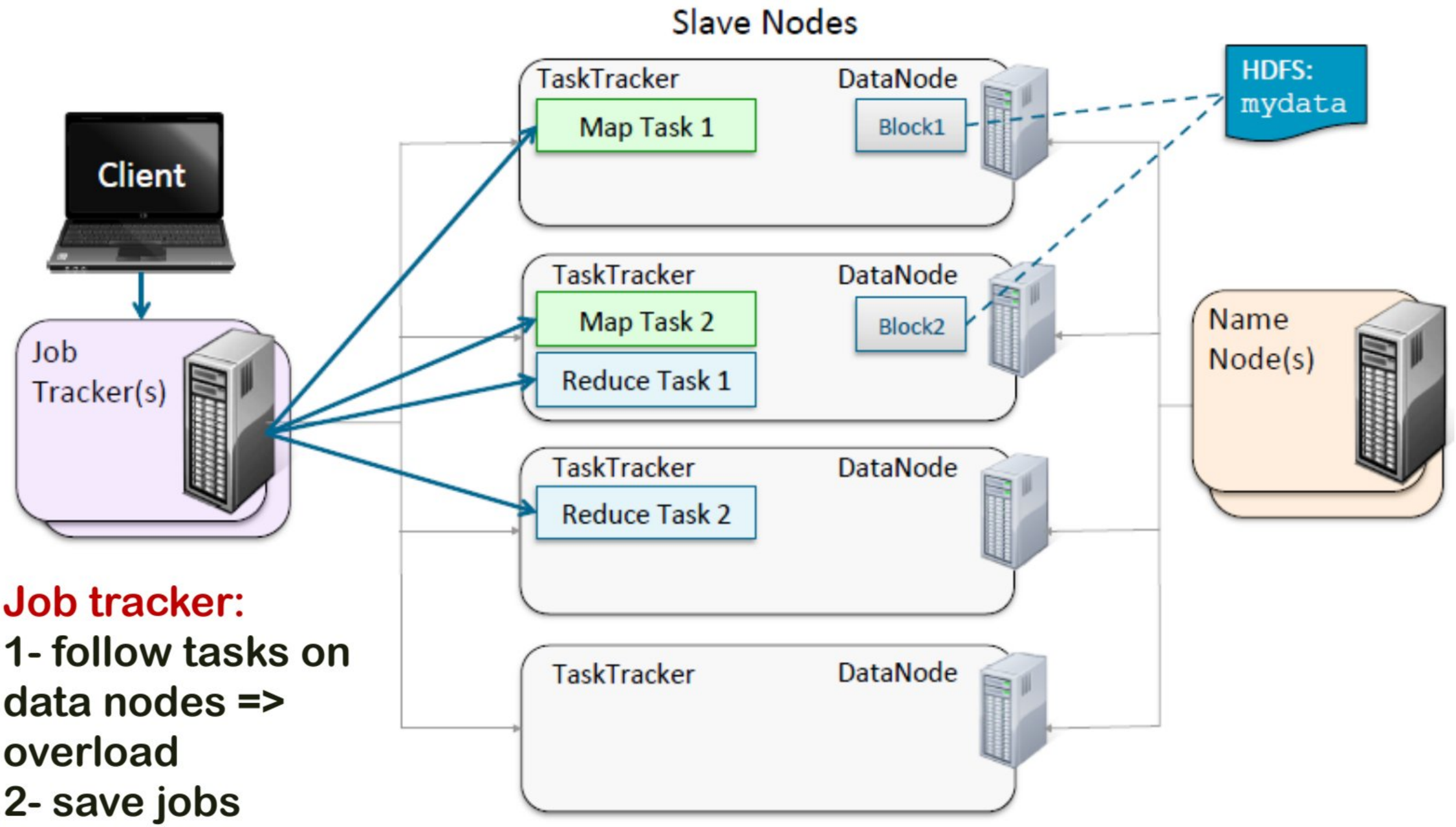
MapReduce v1



MapReduce v1



MapReduce v1

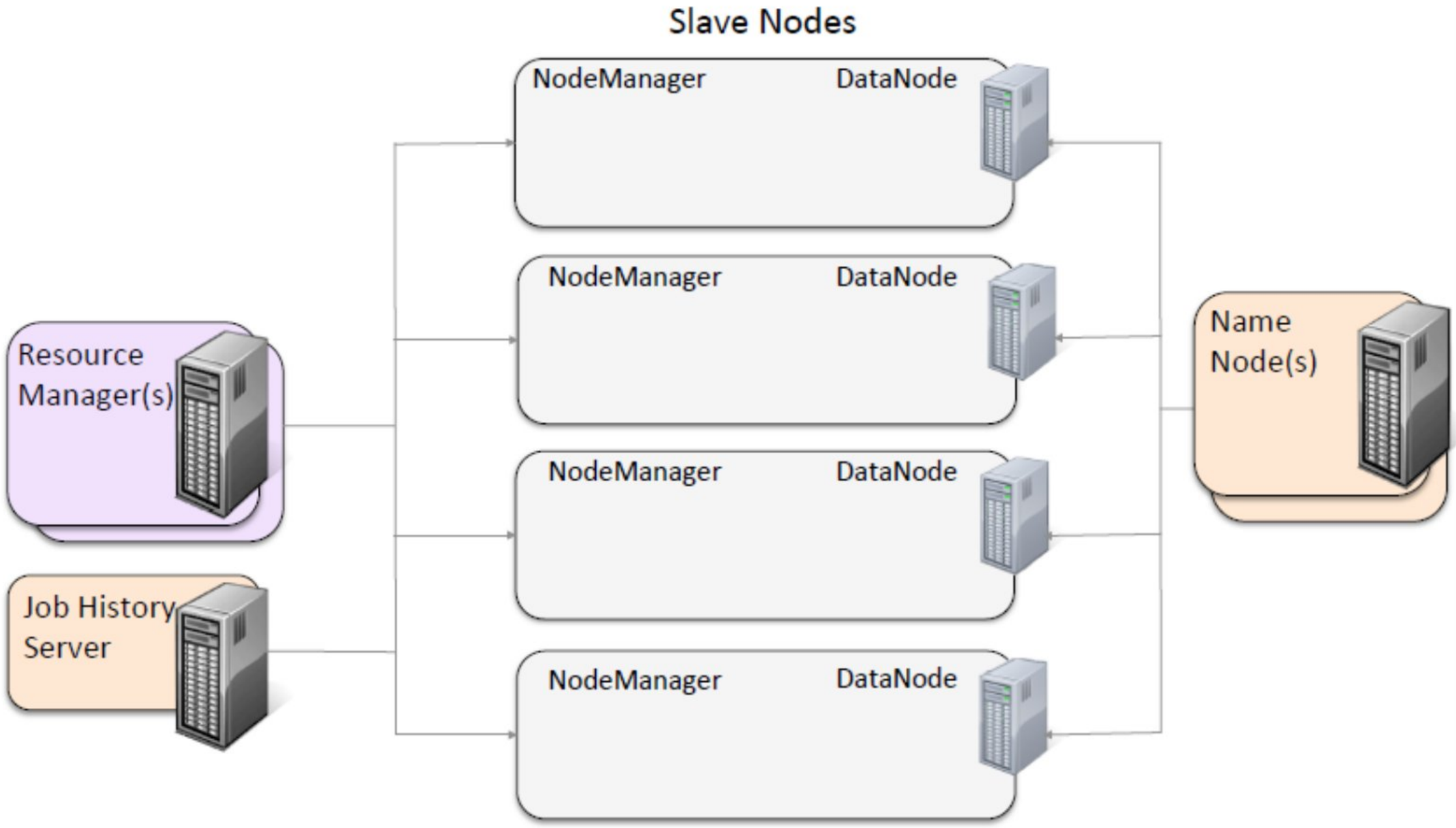


Job tracker:

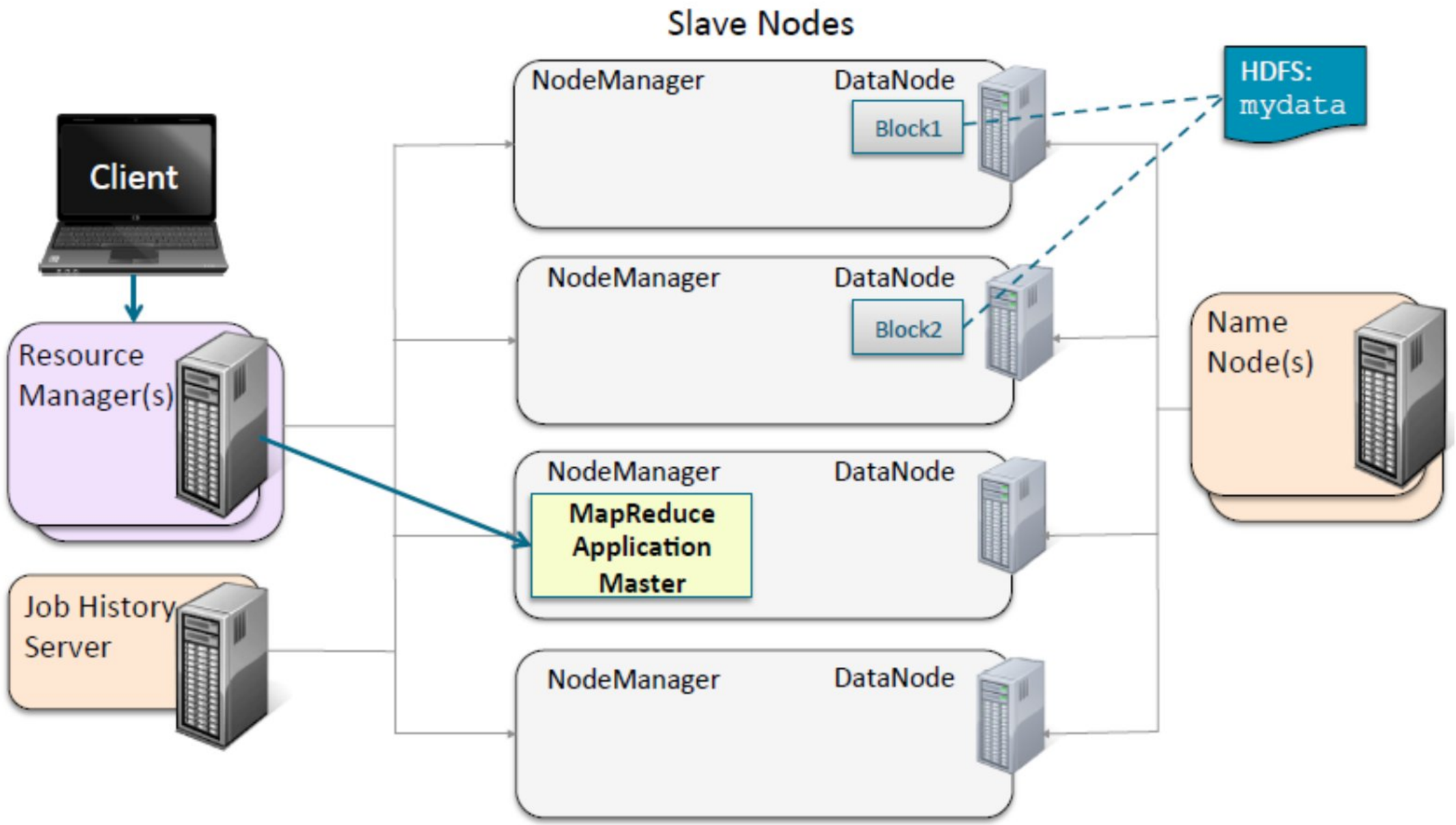
1- follow tasks on data nodes => overload

2- save jobs history

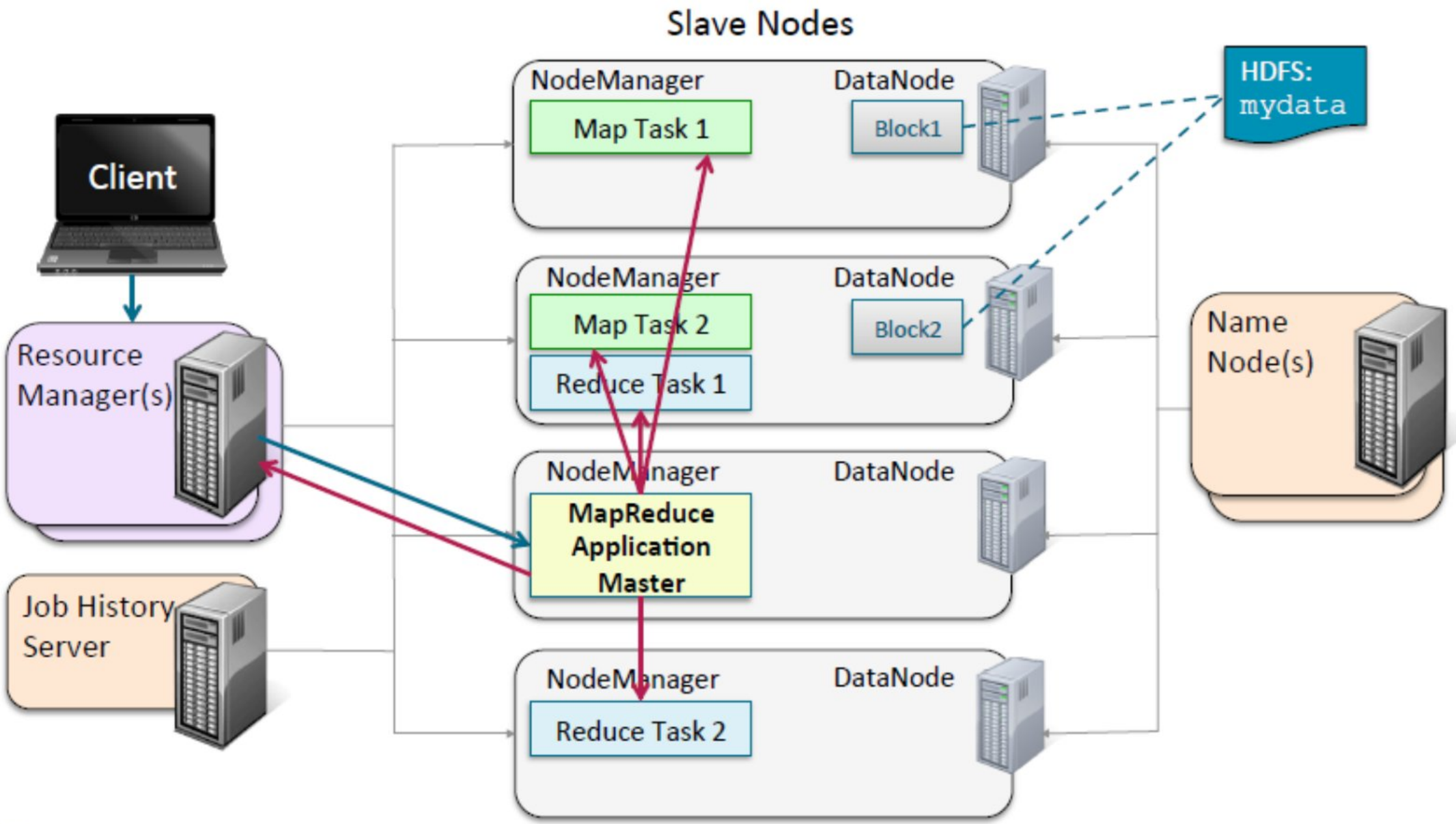
MapReduce v2



MapReduce v2



MapReduce v2



Node manager

- **Node Manager**

Manages single node resource allocations

Per-node agent

- **Container/Slot**

Basic unit of allocation

Ex:Container X= 2GB,1 CPU

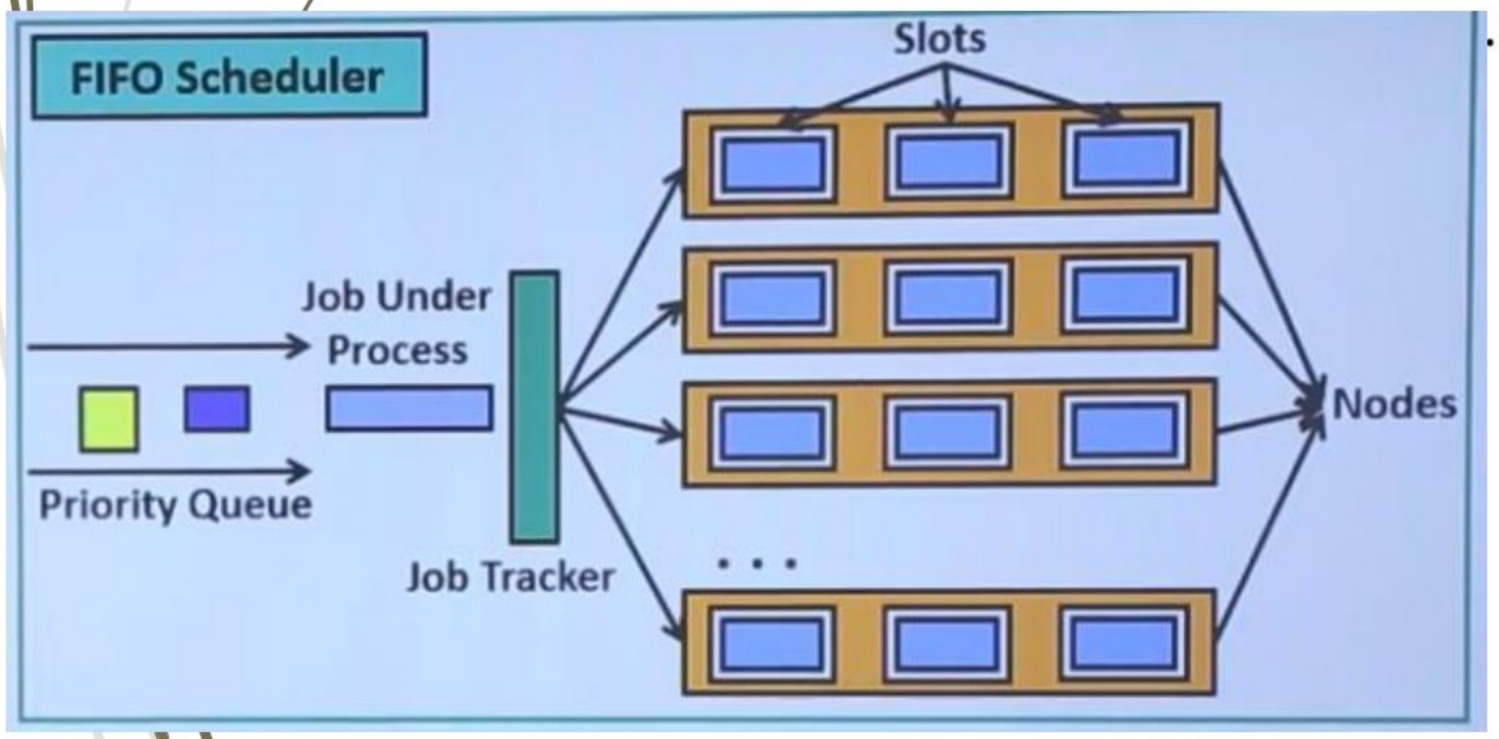
Fine grained resource

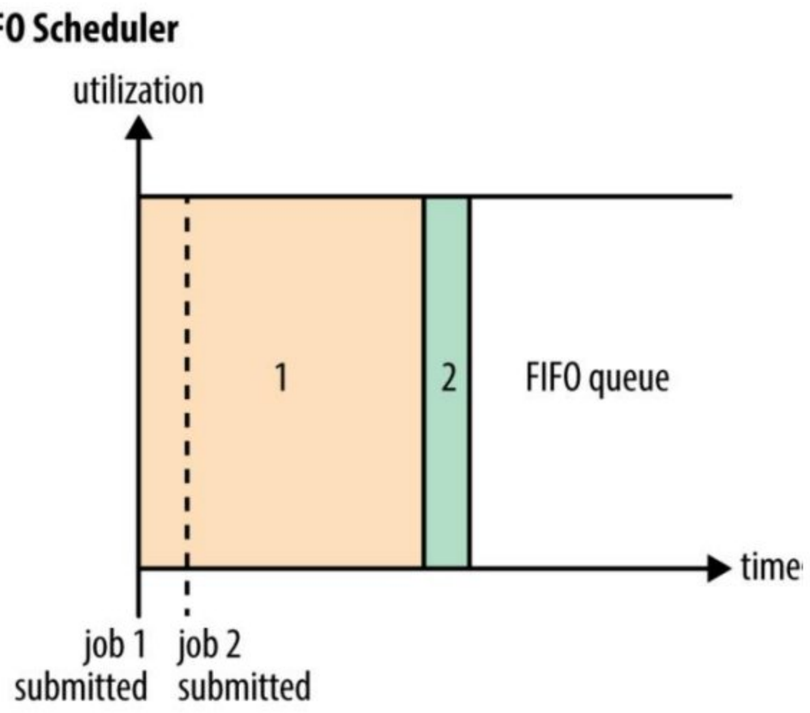
Scheduling in YARN

- It is the job of the YARN scheduler to allocate resources to applications according to some defined policy.
- Three schedulers are available in YARN:
  - FIFO
  - Capacity
  - Fair

FIFO Scheduler

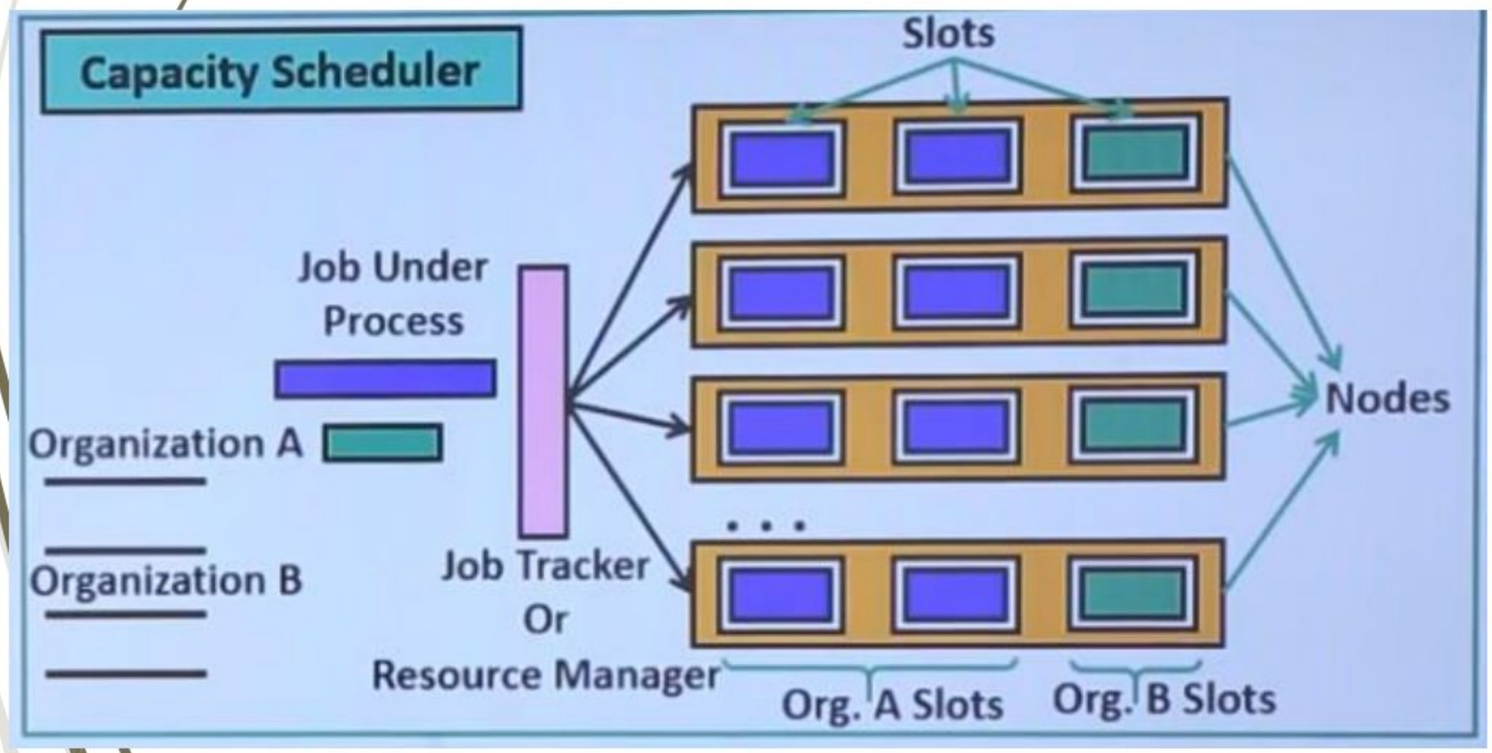
- First-in First-out
- **Problem:** Large applications will use all the resources in a cluster, so each application has to wait its turn.
- **Problem:** it's not suitable for shared clusters



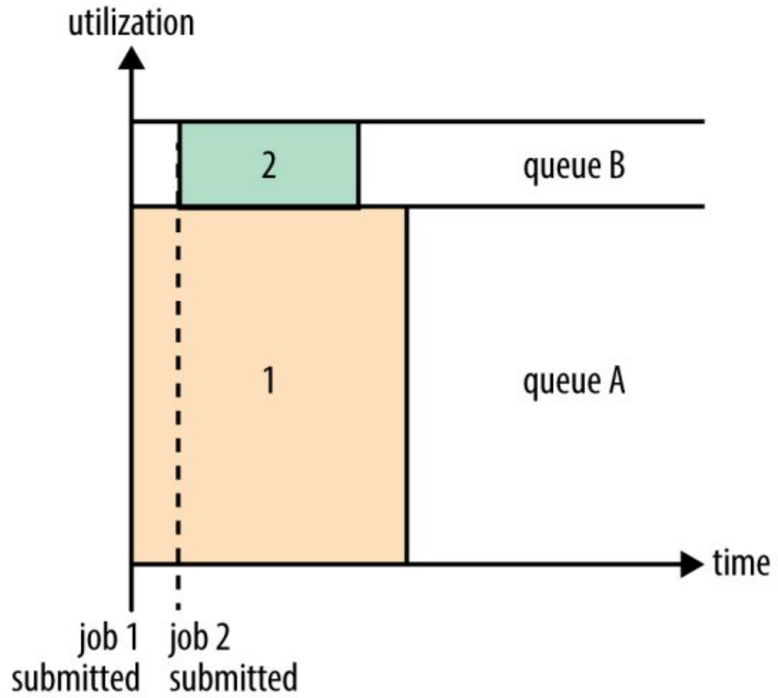


Capacity Scheduler

- A separate dedicated queue allows the small job to start as soon as it is submitted
- **Problem:**
  - the cost of overall cluster utilization since the queue capacity is reserved for jobs in that queue.
  - the large job finishes later than when using the FIFO Scheduler.

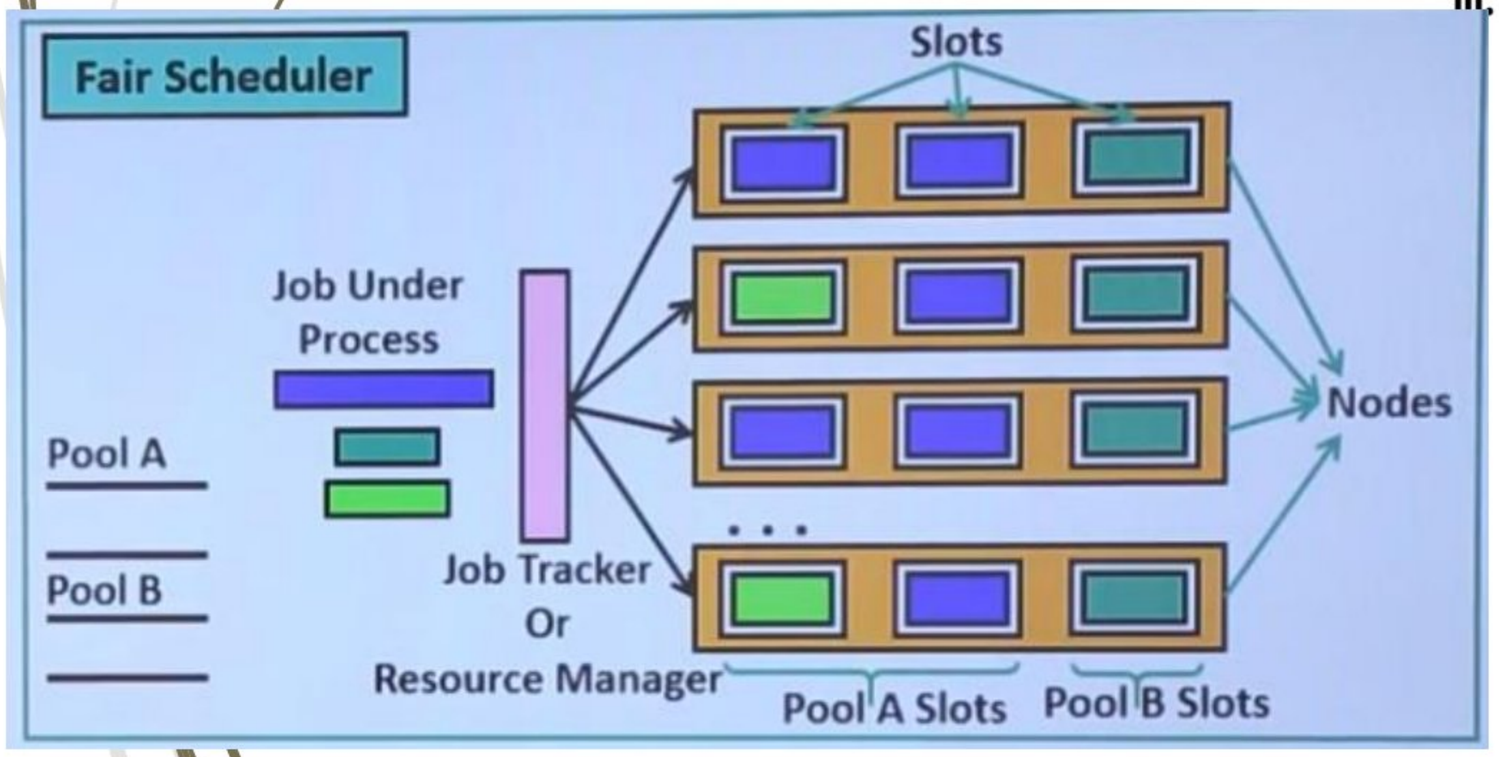


ii. Capacity Scheduler

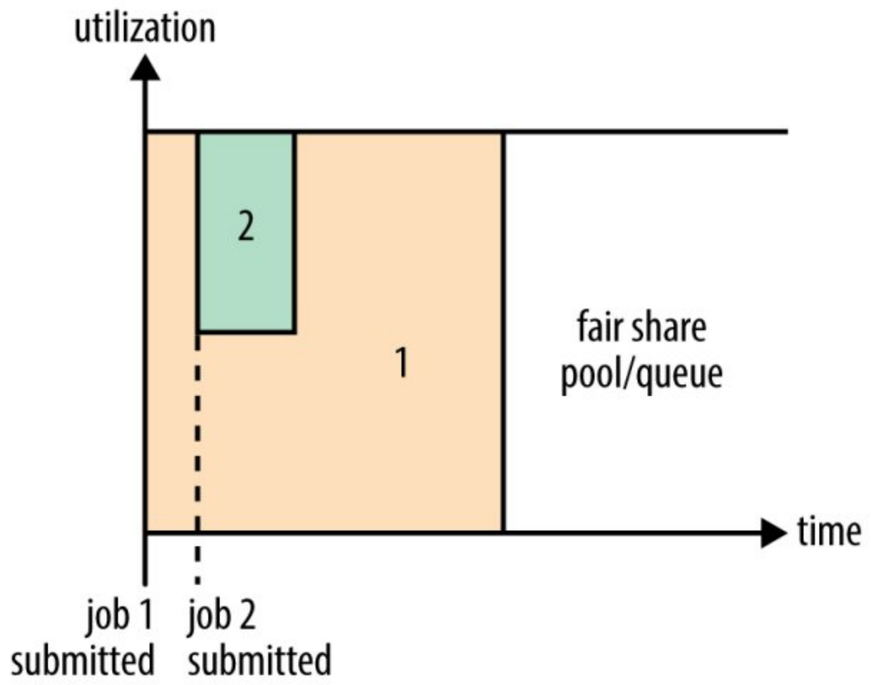


Fair Scheduler

- There is no need to reserve a set amount of capacity, since it will dynamically balance resources between all running jobs.
- Just after the first (large) job starts, it is the only job running, so it gets all the resources in the cluster.
- When the second (small) job starts, it is allocated half of the cluster resources so that each job is using its fair share of resources.



iii. Fair Scheduler



Queue Configuration

- *fair-scheduler.xml*

```
<?xml version="1.0"?>
<allocations>
  <defaultQueueSchedulingPolicy>fair</defaultQueueSchedulingPolicy>

  <queue name="prod">
    <weight>40</weight>
    <schedulingPolicy>fifo</schedulingPolicy>
  </queue>

  <queue name="dev">
    <weight>60</weight>
    <queue name="eng" />
    <queue name="science" />
  </queue>

  <queuePlacementPolicy>
    <rule name="specified" create="false" />
    <rule name="primaryGroup" create="false" />
    <rule name="default" queue="dev.eng" />
  </queuePlacementPolicy>
</allocations>
```

Hadoop: Data Integrity

- Users of Hadoop rightly expect that no data will be lost or corrupted during storage or processing
- It uses CRC-32 for checksumming in Hadoop's ChecksumFileSystem
- A separate checksum is created for every 512 bytes of data
- CRC is made for every read/write data from/to HDFS
- View checksum for every file in HDFS
  - hadoop fs -checksum /*.txt

Hadoop: Compression

- File compression brings two major benefits:
  - Reduces the space needed to store files
  - Speeds up data transfer across the network or to or from disk
- A summary of compression formats used with Hadoop (efficiency vs speed) (see file extension when decompression)

Compression format	Tool	Algorithm	Filename extension	Splittable?
DEFLATE[a]	N/A	DEFLATE	*.deflate*	No
gzip	*gzip*	DEFLATE	*.gz*	No
bzip2	*bzip2*	bzip2	*.bz2*	Yes
LZO	*lzop*	LZO	*.lzo*	No[b]
LZ4	N/A	LZ4	*.lz4*	No
Snappy	N/A	Snappy	*.snappy*	No

Hadoop: Compression

- Application to run the maximum temperature job producing compressed output

```
public class MaxTemperatureWithCompression {

  public static void main(String[] args) throws Exception {
    if (args.length != 2) {
      System.err.println("Usage: MaxTemperatureWithCompression <input path> " +
          "<output path>");
      System.exit(-1);
    }

    Job job = new Job();
    job.setJarByClass(MaxTemperature.class);

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

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

    FileOutputFormat.setCompressOutput(job, true);
    FileOutputFormat.setOutputCompressorClass(job, GzipCodec.class);

    job.setMapperClass(MaxTemperatureMapper.class);
    job.setCombinerClass(MaxTemperatureReducer.class);
    job.setReducerClass(MaxTemperatureReducer.class);

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

  }
}
```

Writable Classes

- Writable wrapper classes for Java primitives

Java primitive	Writable implementation	Serialized size (bytes)
boolean	BooleanWritable	1
byte	ByteWritable	1
short	ShortWritable	2
int	IntWritable	4
	VIntWritable	1–5
float	FloatWritable	4
long	LongWritable	8
	VLongWritable	1–9
double	DoubleWritable	8

Writable Classes: Example

Up to 2 GB

```
Text t = new Text("hadoop");
assertThat(t.getLength(), is(6));
assertThat(t.getBytes().length, is(6));

assertThat(t.charAt(2), is((int) 'd'));
assertThat("Out of bounds", t.charAt(100), is(-1));

Text t = new Text("hadoop");
assertThat("Find a substring", t.find("do"), is(2));
assertThat("Finds first 'o'", t.find("o"), is(3));
assertThat("Finds 'o' from position 4 or later", t.find("o", 4), is(4));
assertThat("No match", t.find("pig"), is(-1));
```

Sequence file: more suitable for storing file