DASCI

Instituto Andaluz Interuniversitario
en Ciencia de Datos e
Inteligencia Computacional

Ciencia de Datos a través del Big Data

Diego García (djgarcia@ugr.es)
Isaac Triguero (isaaktriguero@ugr.es)

Financiado por
la Unión Europea
NextGenerationEU

GOBIERNO
DE ESPAÑA

MINISTERIO
PARA LA TRANSFORMACIÓN DIGITAL
Y DE LA FUNCIÓN PÚBLICA

SECRETARÍA DE ESTADO
DE DIGITALIZACIÓN
E INTELIGENCIA ARTIFICIAL

Plan de
Recuperación,
Transformación
y Resiliencia

UNIVERSIDAD
DE GRANADA

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

REPSOL

CAMBRIDGE UNIVERSITY PRESS

Chapter 3

Hadoop I

Learning Outcomes

- How a big data application is distributed in Hadoop [KU]
- How a distributed file systems enables data locality for big data processing [KU]
- How transparency and fault tolerance are implemented in Hadoop [KU]
- Limitations of Hadoop for different data processing approaches [KU, IS]

Recap

- MapReduce as **programming model** for big data
  - Two main functions to program (`map` and `reduce`)
  - Key-values determine the movement across network
  - Why is appropriate for Big Data processing?
    - Data locality
    - Transparency (hide complexity away)
    - Fault-tolerance
- This is, let's say, a concept. We need an actual implementation to use it

Objective for today

- What is Hadoop?
- Word-Count with Hadoop MapReduce
- Hadoop in detail
- Executing MapReduce in a cluster

What is Hadoop?

- Hadoop is:
  - An **open-source** framework written in Java
  - Distributed storage of very large data sets
  - Distributed processing of very large data sets
- This framework consists of a **number of modules**
  - *Hadoop Common*
  - ***Hadoop Distributed File System (HDFS)***
  - *Hadoop YARN* – resource manager (from V2)
  - ***Hadoop MapReduce*** – programming model

Doug Cutting

Ghemawat, S., Gobioff, H., and Leung, S.-T. 2003. The Google file system. Pages 29–43 of: *Proceedings of the nineteenth ACM symposium on Operating systems principles*. SOSP '03. New York, NY, USA: ACM. Link
Dean, Jeffrey, and Ghemawat, Sanjay. 2004. MapReduce: Simplified Data Processing on Large Clusters. Pages 137–150 of: *OSDI'04: Sixth Symposium on Operating System Design and Implementation*. Link

Hadoop's birth

- **July 2008 - Hadoop Wins Terabyte Sort Benchmark**
  - One of Yahoo's Hadoop clusters sorted 1 terabyte of data in 209 seconds, which beat the previous record of 297 seconds in the annual general purpose (Daytona) **terabyte sort benchmark**. This is the first time that either a **Java** or an open-source program won.

2008, 3.48 minutes

Hadoop

910 nodes x (4 dual-core processors, 4 disks, 8 GB memory)

Owen OMalley, Yahoo

2007, 4.95 min

TokuSampleSort

tx2500 disk cluster

400 nodes x (2 processors, 6-disk RAID, 8 GB memory)

Bradley C. Kuszmaul , MIT

	Daytona
Gray	2013, 1.42 TB/min **Hadoop** 102.5 TB in 4,328 seconds 2100 nodes x (2 2.3Ghz hexcore Xeon E5-2630, 64 GB memory, 12x3TB disks) Thomas Graves Yahoo! Inc.

Explaining Hadoop MapReduce

WordCount using Hadoop MapReduce

Map(key,value): **key**: line number; **value**: text in line

```
public static class TokenizerMapper
        extends Mapper<Object, Text, Text, IntWritable>{
```

```
private final static IntWritable one = new IntWritable(1);
private Text word = new Text();
```

Member variables

```
public void map(Object key, Text value, Context context
) throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
        word.set(itr.nextToken());
        context.write(word, one);
    }
}
```

https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

WordCount using Hadoop MapReduce

Reduce(key,<values>): **key**: word; **<values>**: list of repetitions

```
public static class IntSumReducer extends
Reducer<Text,IntWritable,Text,IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(Text key, Iterable<IntWritable> values, Context
context) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      sum += val.get();
    }
    result.set(sum);
    context.write(key, result);
  }
}
```

https://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

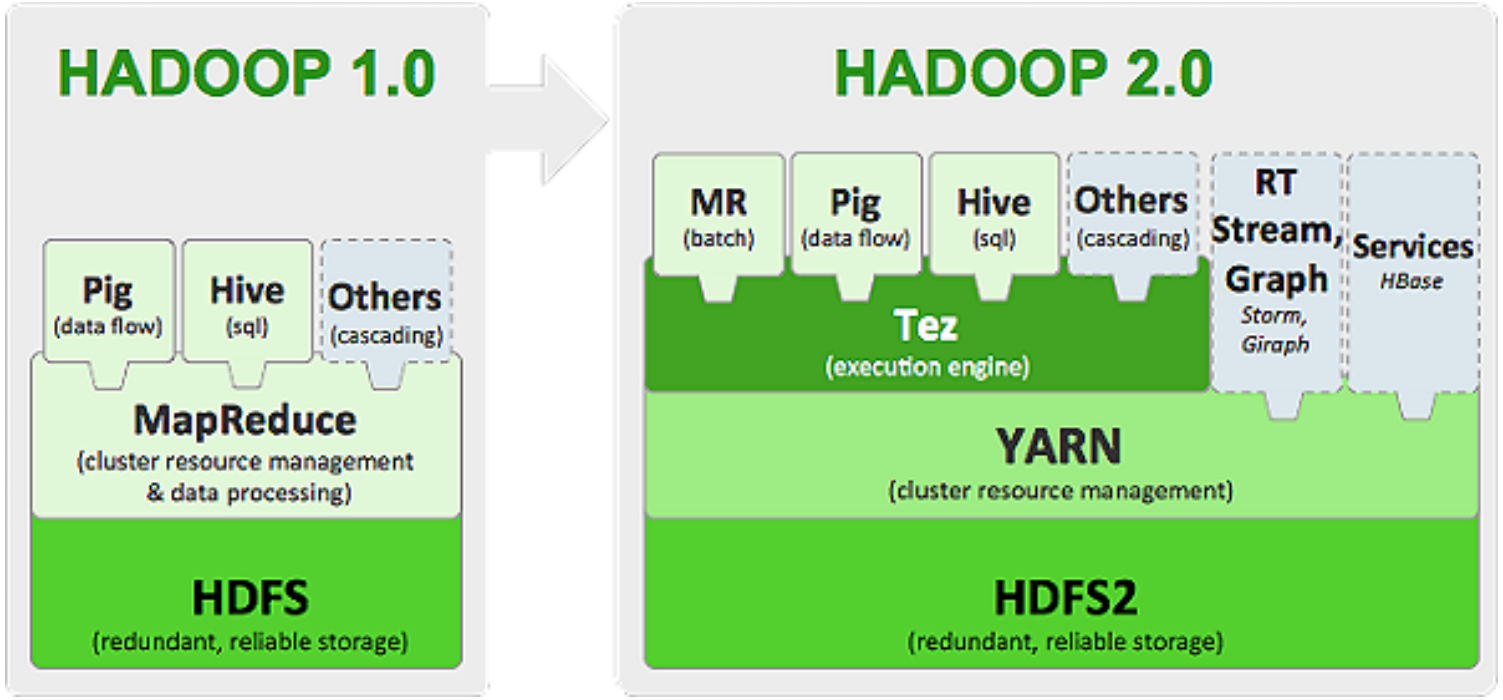
WordCount using Hadoop MapReduce

- The Main Function for the WordCount program

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    String[] otherArgs = new GenericOptionsParser(conf, \
                                        args).getRemainingArgs();
    if (otherArgs.length < 2) {
        System.err.println("Usage: wordcount <in> [<in>...] <out>");
        System.exit(2);
    }
    Job job = Job.getInstance(conf, "word count");
    job.setJarByClass(WordCount.class);
    job.setMapperClass(TokenizerMapper.class);
    job.setCombinerClass(IntSumReducer.class);
    job.setReducerClass(IntSumReducer.class);
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);
    for (int i = 0; i < otherArgs.length - 1; ++i) {
        FileInputFormat.addInputPath(job, new Path(otherArgs[i]));
    }
    FileOutputFormat.setOutputPath(job,
        new Path(otherArgs[otherArgs.length - 1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

Hadoop: Basic concepts

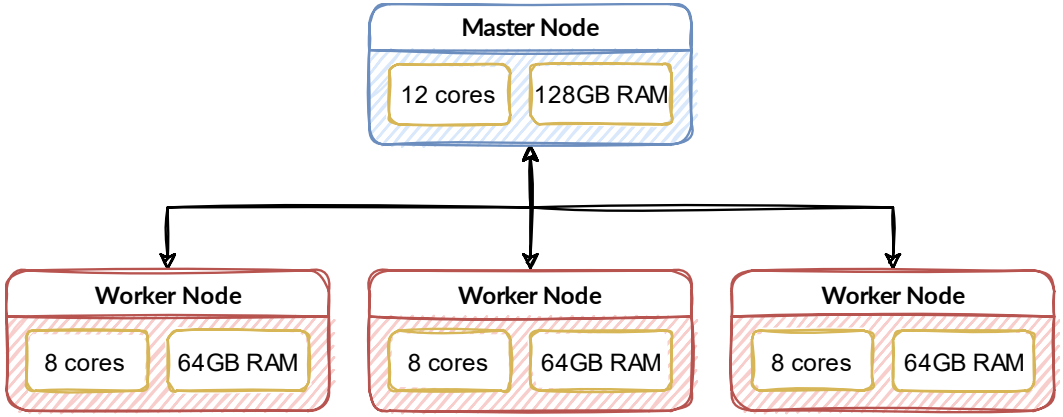
- **Yarn** (*Yet Another Resource Negotiator*) was included in version 2, taking away resource administration from the MapReduce module
- Advantageous for other projects allowing use Yarn and HDFS without using MapReduce from Hadoop



http://hadoop.apache.org/

Hadoop: Basic concepts

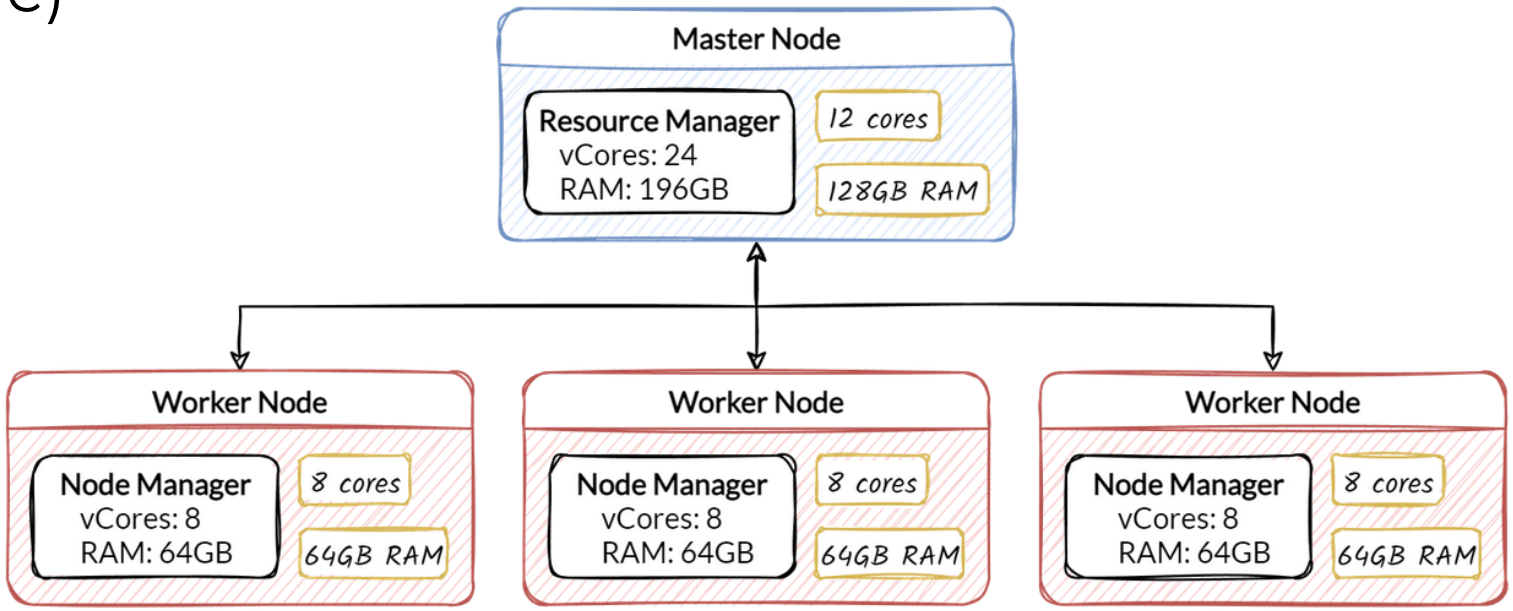
- YARN is organized around **4 key concepts**
  - Resource Manager
  - Node Manager
  - Container
  - Application Master



Master/Worker architecture

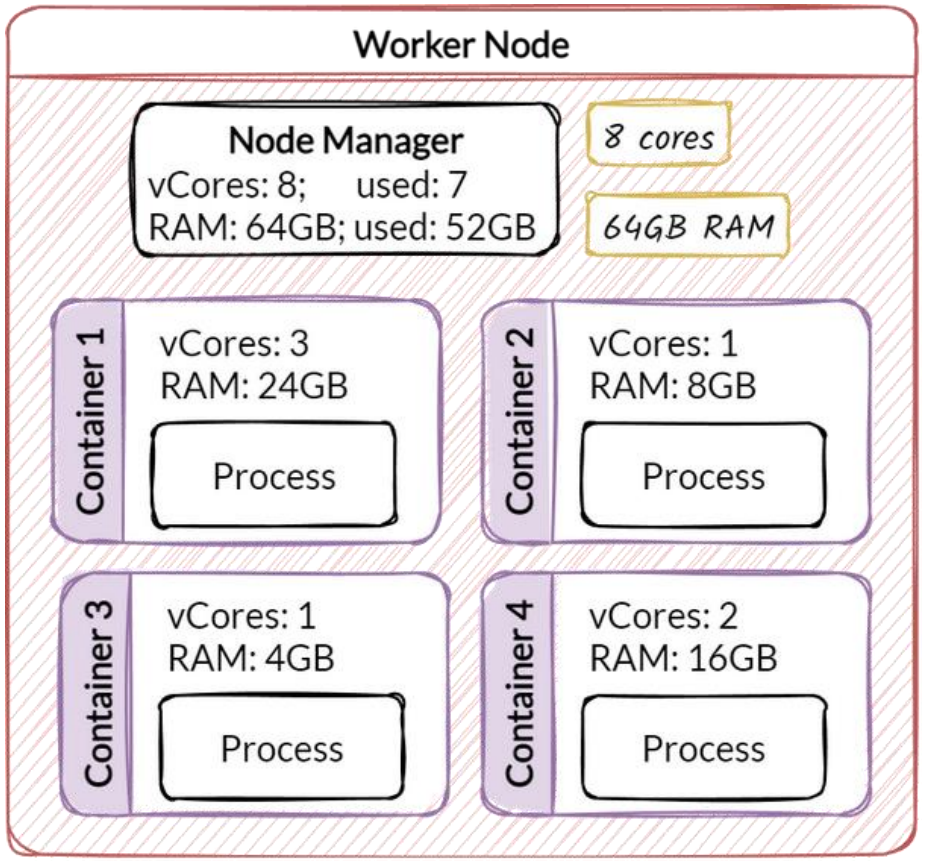
Hadoop: Basic concepts

- **Resource Manager**
  - Controls resources available in the cluster (for all applications). Takes part of the duty from the JobTracker
- **Node Manager**
  - Launches and track processes assigned to workers (one process per node)

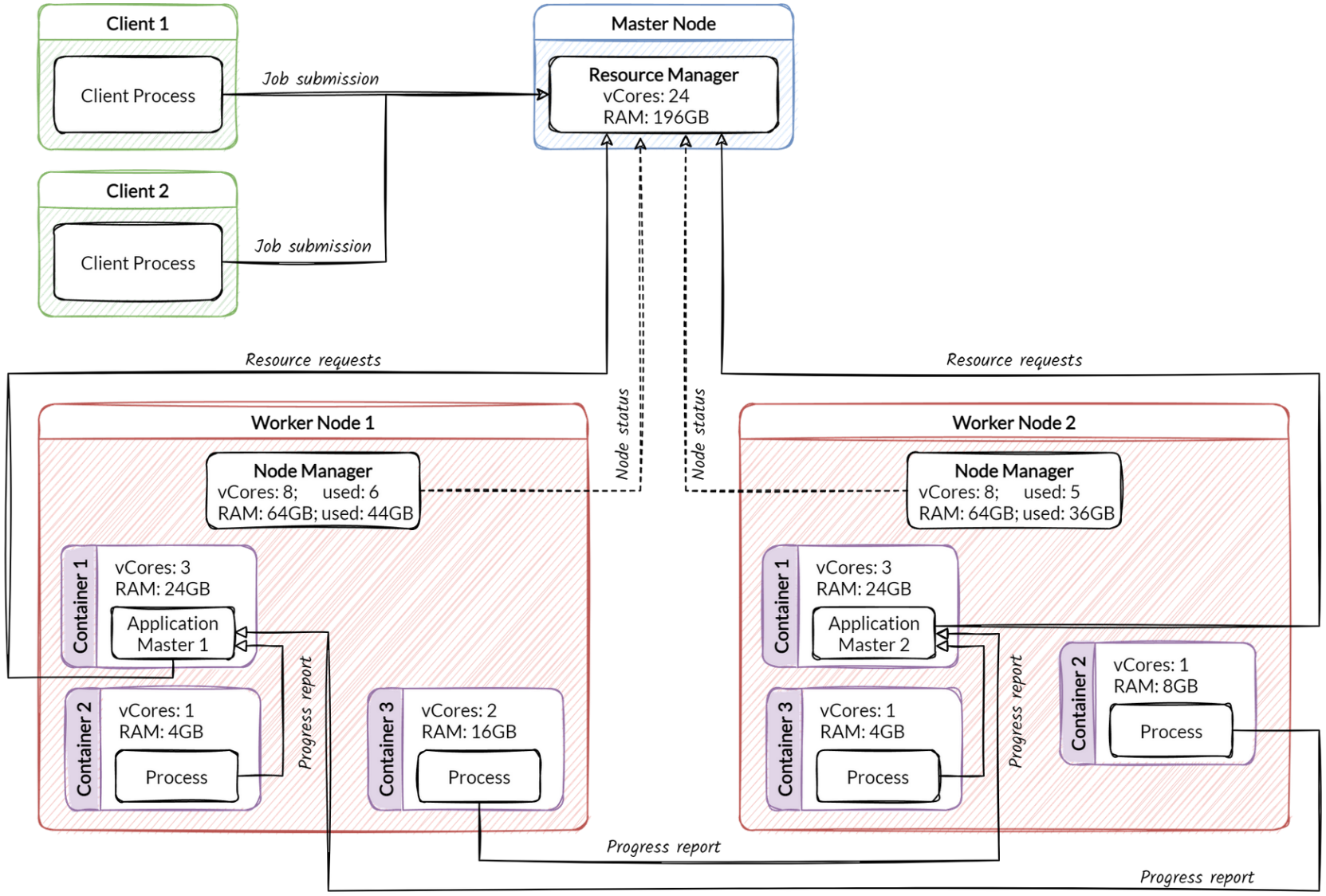


Hadoop: Basic concepts

- **Container**: It is a subset of resources of the cluster (key concept!)
- **Application master**: Manages a particular application and runs on a container. Responsible for fault tolerance



Hadoop: Basic concepts – Resource Negotiator Process

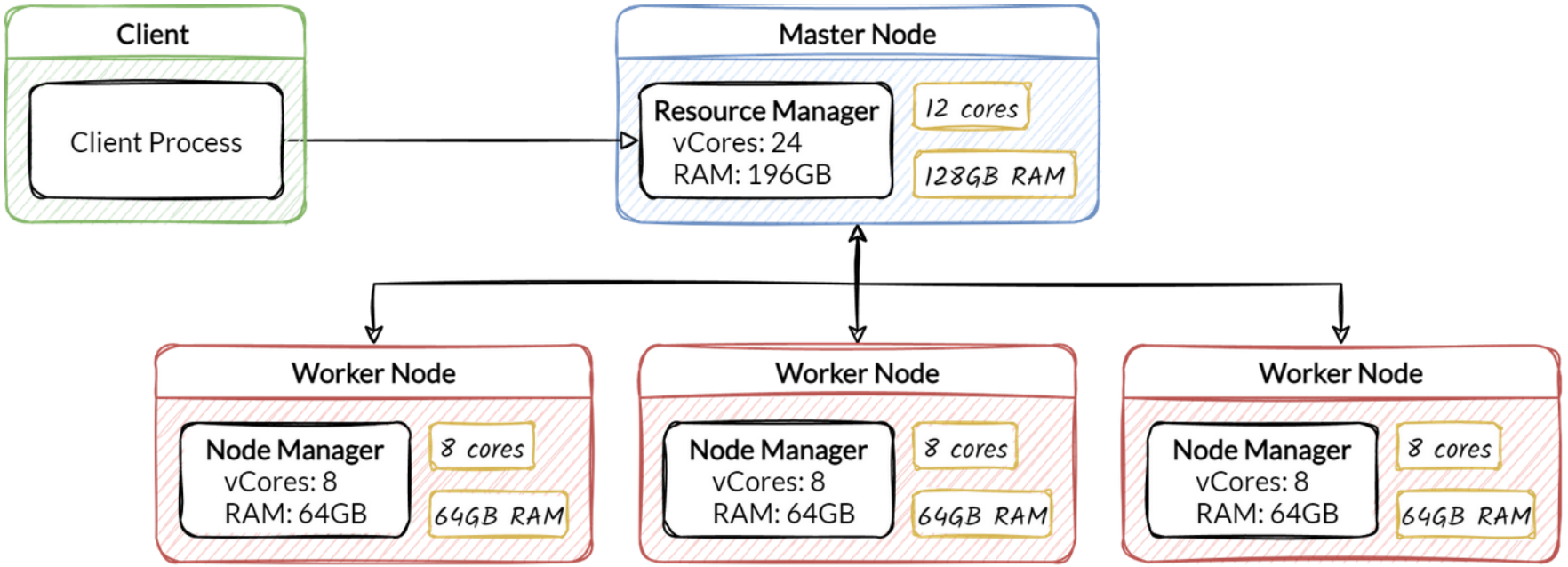


Hadoop: Basic concepts – Configurations

- Hadoop can be run with **3 different configurations**:
  1. **Local / Standalone**. It is run in a single JVM (Java Virtual Machine). *Very useful for debugging!*
  2. **Pseudo-distributed** (Cluster simulator)
  3. **Distributed** (Cluster)

Hadoop V2 – Executing a MapReduce process

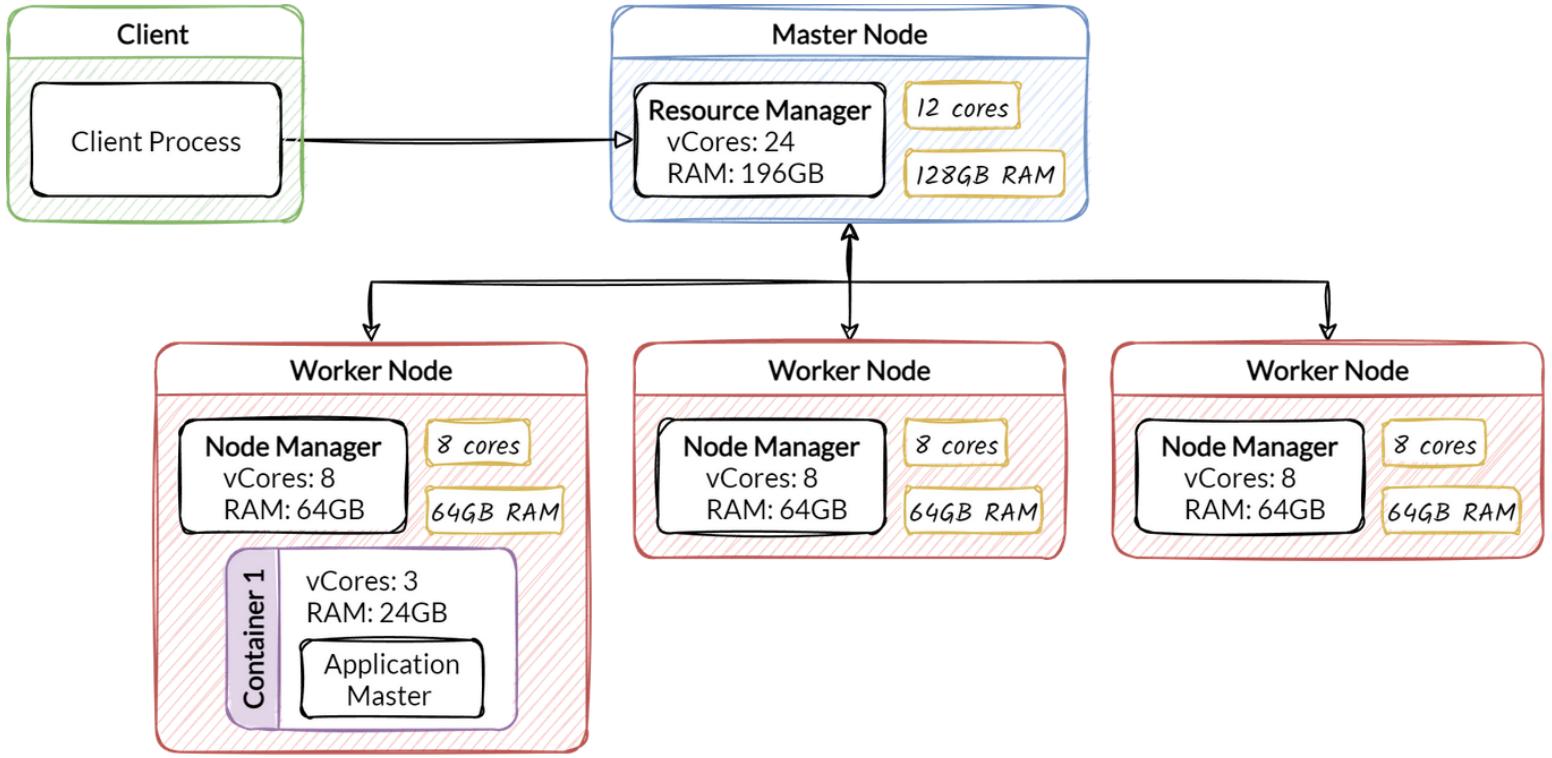
- **Execution** of a MapReduce process
  1) The client launches the process (connection with the `Resource Manager`)



Hadoop V2 – Executing a MapReduce process

- **Execution** of a MapReduce process

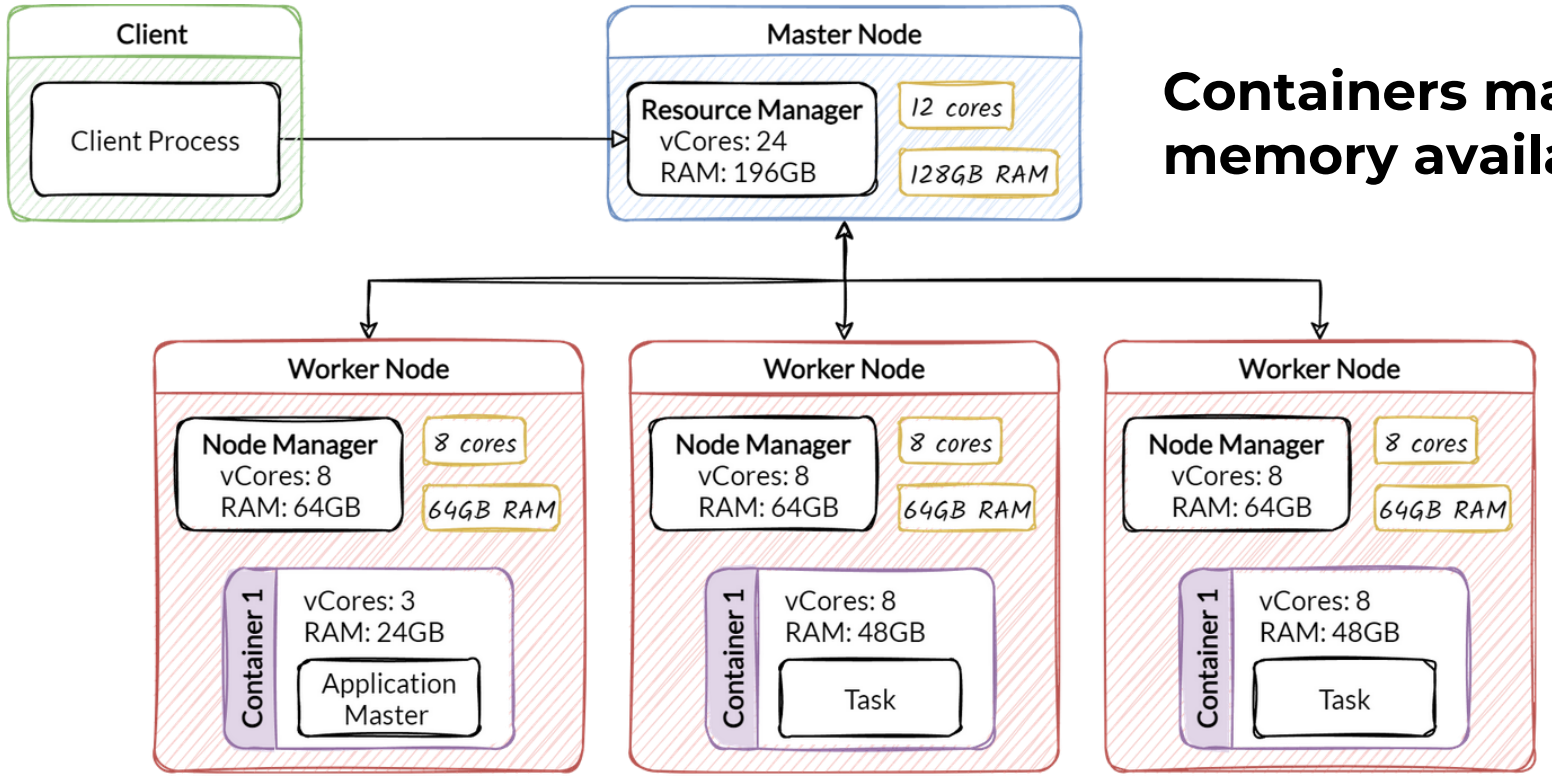
  2) The **`Resource Manager`** allocates a **single** container where the **`Application Master`** is executed



Hadoop V2 – Executing a MapReduce process

- **Execution** of a MapReduce process

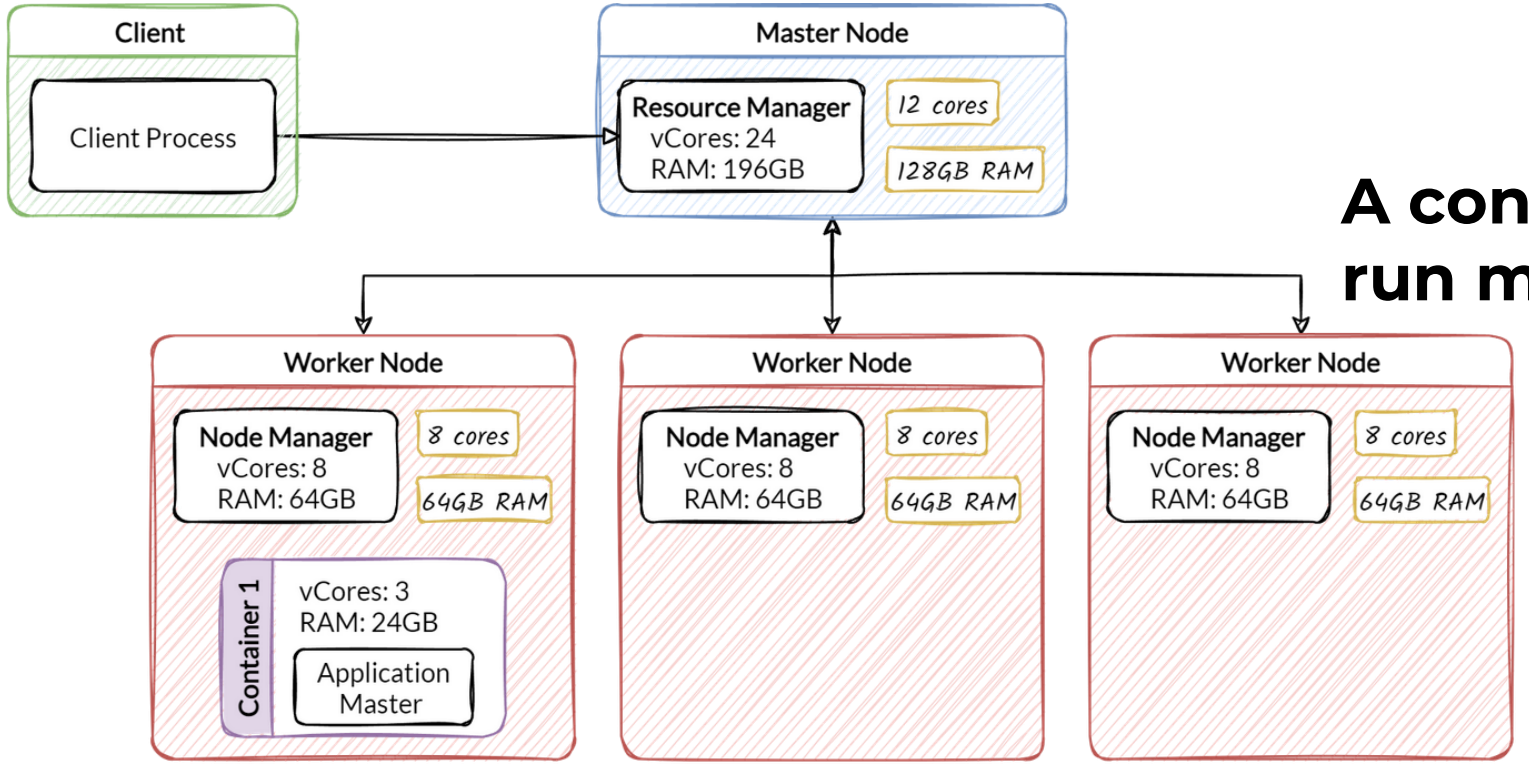
  3) The **`Application Master`** requests the containers to execute all the tasks (in different nodes)



Containers may not use all the memory available in a node

Hadoop V2 – Executing a MapReduce process

- **Execution** of a MapReduce process
  4) All the tasks are executed in the containers.
     Containers are released once its tasks are finished

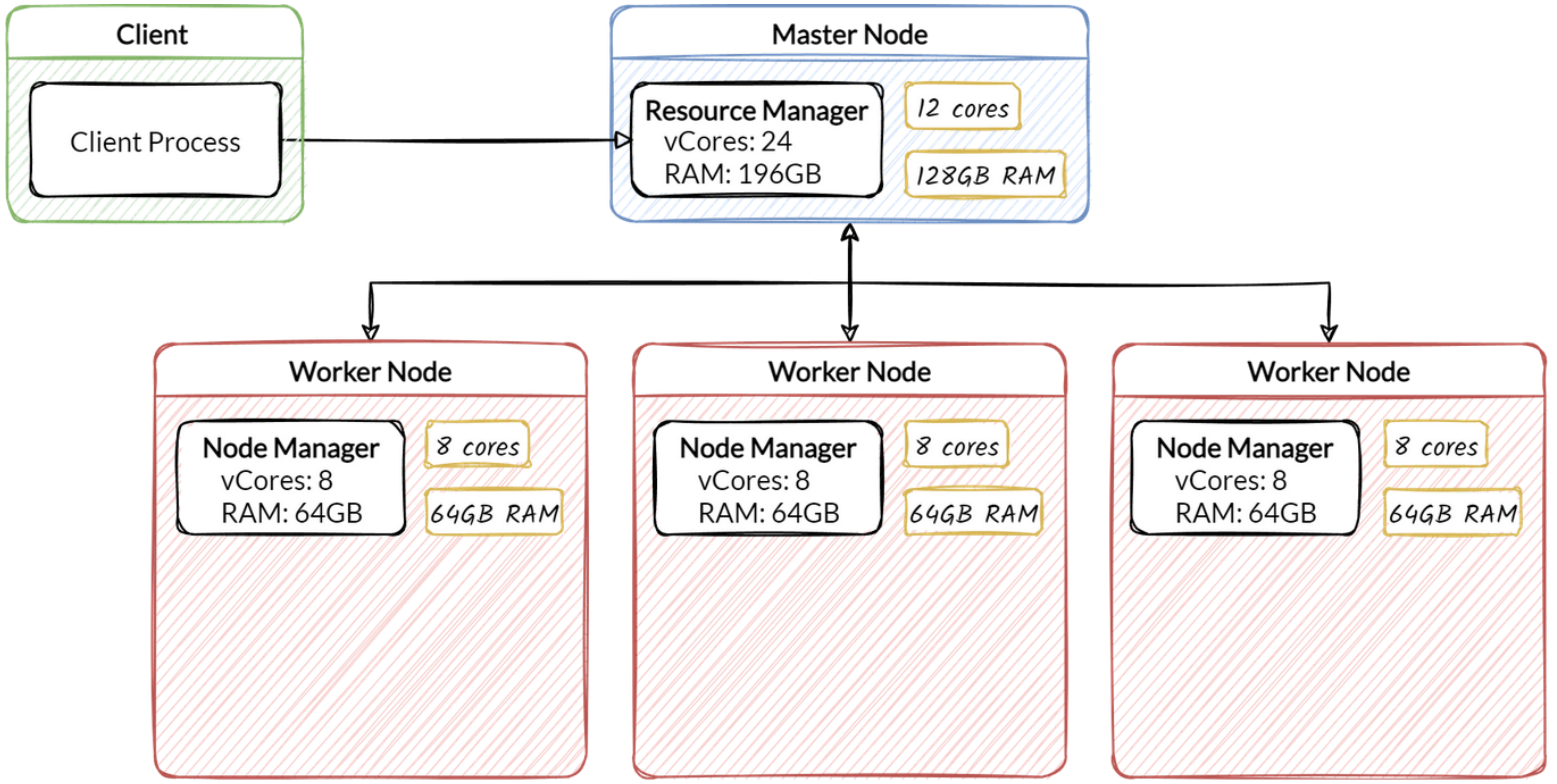


A container may run multiple tasks

Hadoop V2 – Executing a MapReduce process

- **Execution** of a MapReduce process

  5) The **`Application Master`** finishes when all tasks have been completed and release the container



Hadoop V2 – Executing a MapReduce process

- Summary:
  1) The client launches the process (connection with the **`Resource Manager`**)
  2) The **`Resource Manager`** requests a container where the **`Application Master`** is executed
  3) The **`Application Master`** requests the containers to execute all the tasks (in different nodes)
  4) All the tasks are executed in the containers. Containers are released once its tasks are finished
  5) The **`Application Master`** finishes when all tasks have been completed and release the container

Challenge

- If you have installed Hadoop in your machine, implement your solutions to the MapReduce exercises we discussed in the first lab using Hadoop MapReduce.

Take-home message

- Hadoop is not MapReduce, nor vice versa
- Hadoop is composed of multiple layers: HDFS, YARN and the MapReduce implementation
- Implementing MapReduce in Hadoop may be a bit cumbersome
- Master/worker architecture
  - Multiple processes involved in the execution of a MapReduce program
    - `Resource Manager`, `Node Manager`, and `Application Master`

What's next?

- The Hadoop File System
- Limitations of Hadoop

CAMBRIDGE UNIVERSITY PRESS

Chapter 3

Hadoop I