Infraestructura para Big Data

Introducción: Clúster Big Data

Tabla de contenidos

Contexto de la asignatura en el máster

Fundamentos de análisis de datos

Análisis de los datos utilizando herramientas de alto nivel librerías de algoritmos de machine learning

Ciclo de vida analítico del dato

Fundamentos: sistemas y arquitecturas

Infraestructura para Big Data

Contexto de la asignatura en el máster

Fundamentos de análisis de datos

Implementación de algritmos con diferentes herramientas disponibles
Hive, Hbase, Java, …

Ciclo de vida analítico del dato

Fundamentos: sistemas y arquitecturas

Infraestructura para Big Data

Contexto de la asignatura en el máster

Fundamentos de análisis de datos

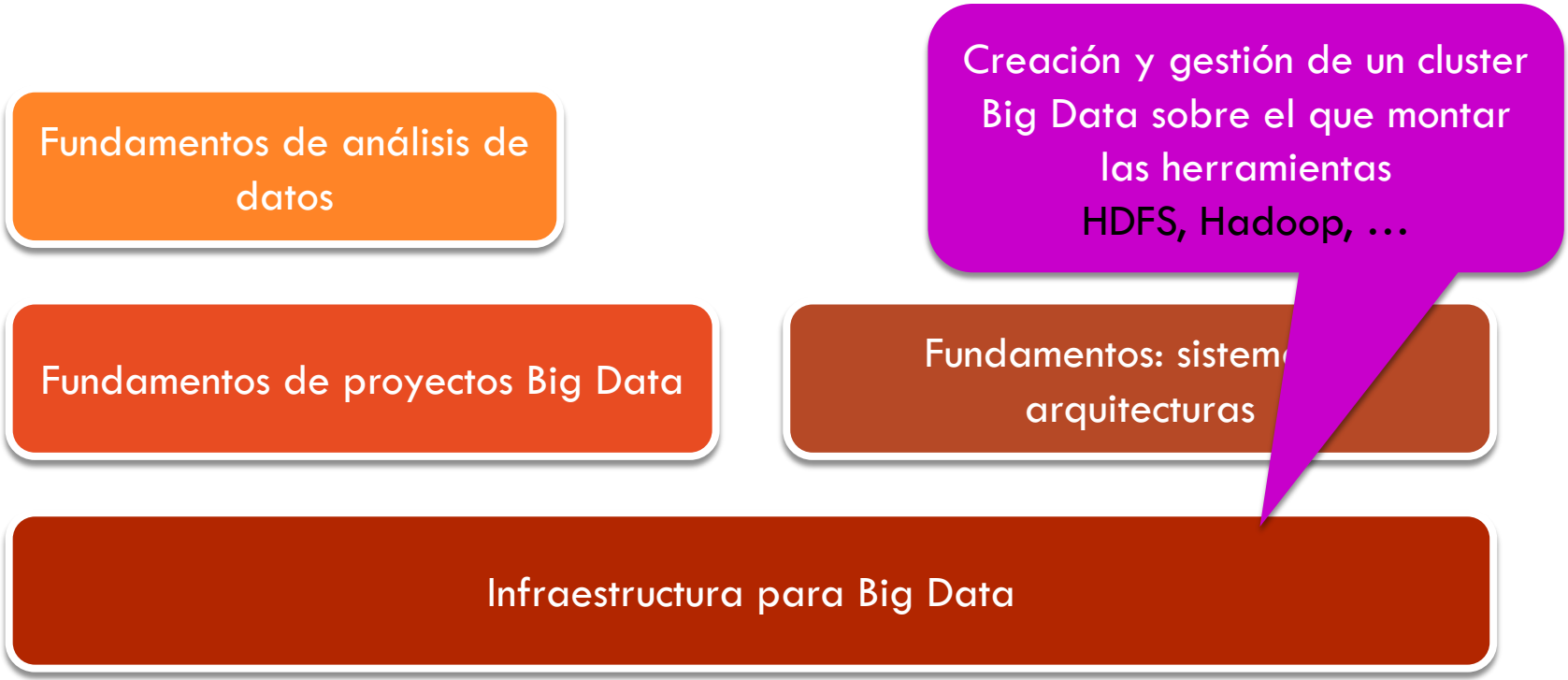
Fundamentos sobre Linux, Redes de ordenadores, Virtualización

Ciclo de vida analítico del dato

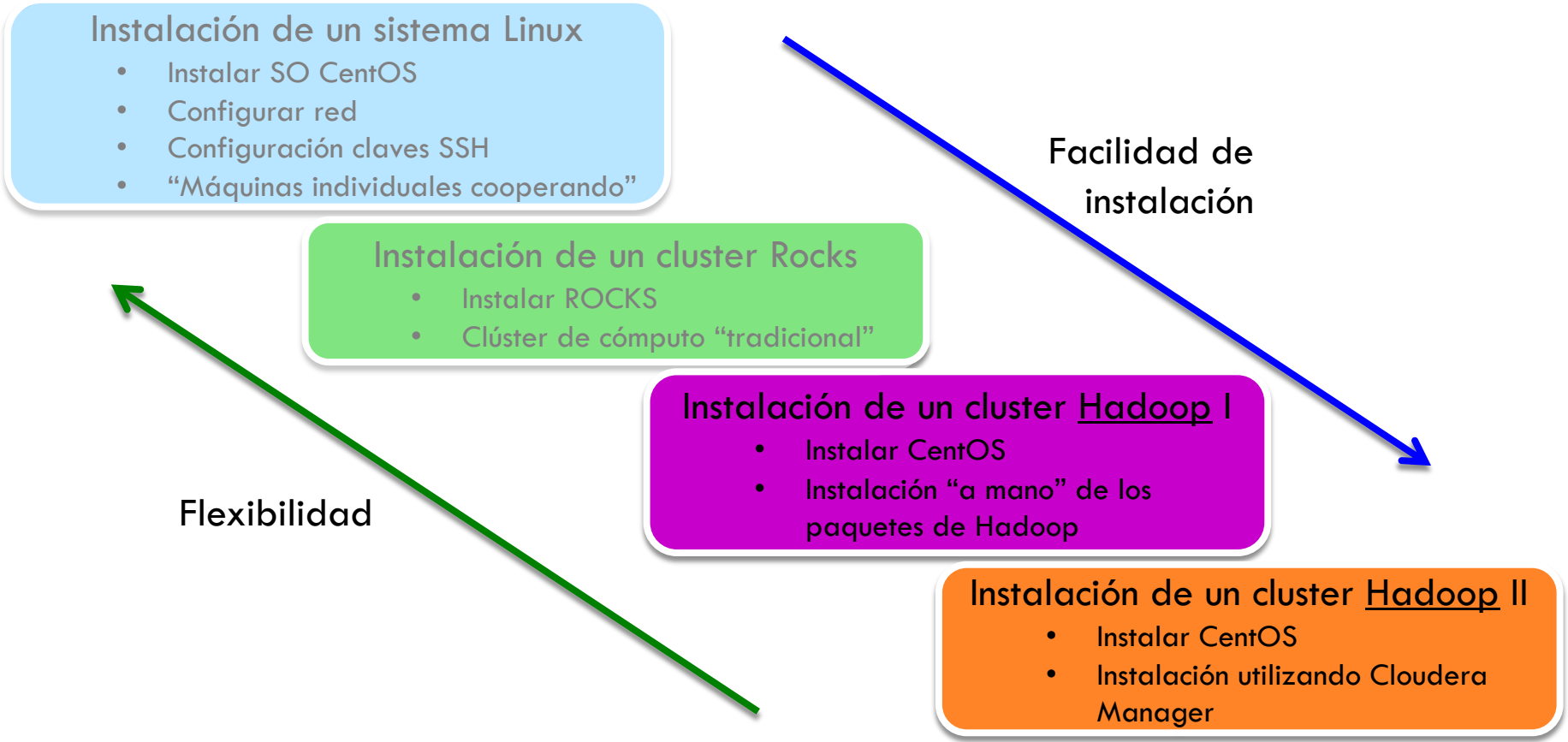
Fundamentos: sistemas y arquitecturas

Infraestructura para Big Data

Contexto de la asignatura en el máster



Contexto de la asignatura en el máster

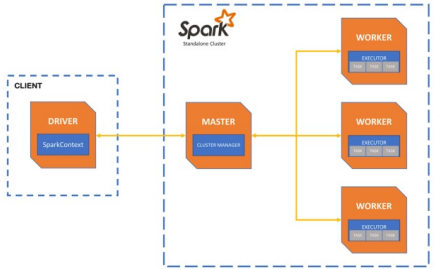


Algunos conceptos clave

- Clúster Big Data (Hadoop, Spark) vs Clúster HPC
  - HPC: computation-constrained

We need plenty of cores

  - Big Data: I/O constrained (acces to data)



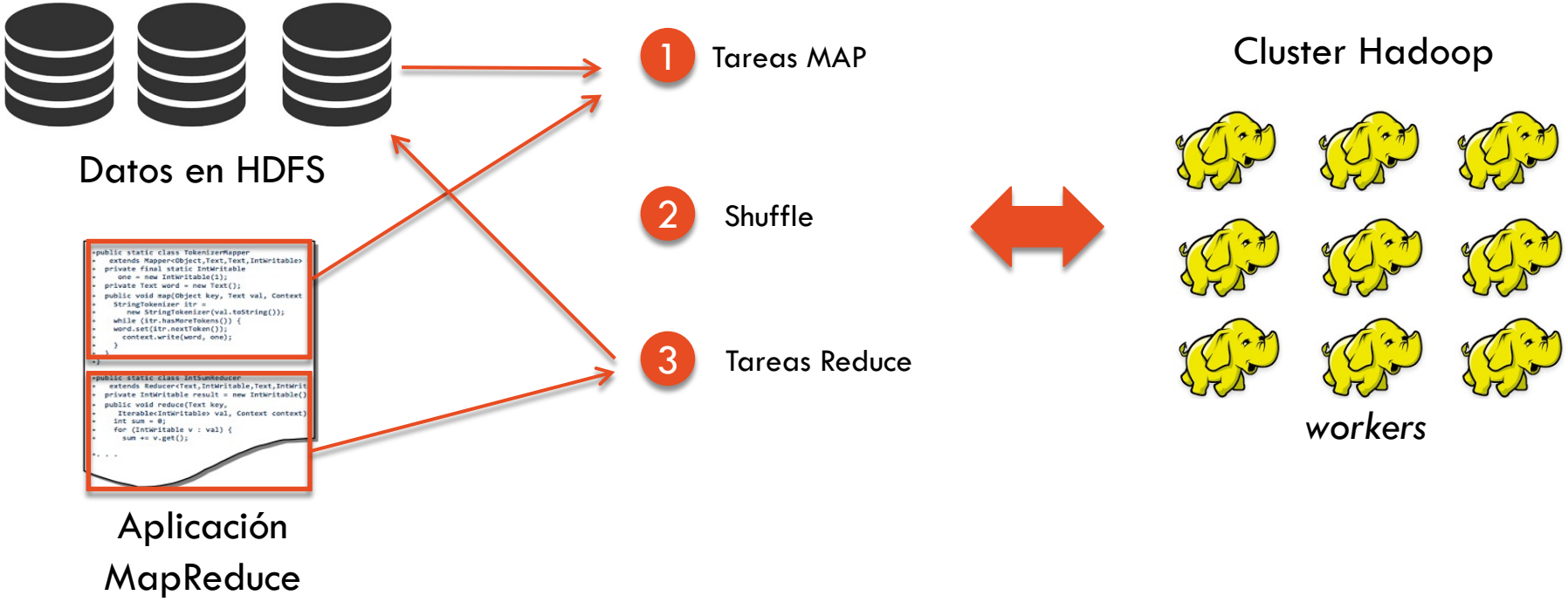
We need plenty of storage (disk or memory)

Algunos conceptos clave

- Big Data, Cloud computing y Virtualización
  - Disponer de un clúster Big Data 24x7 no está al alcance de todos…
  - La tecnología Cloud está cada vez más extendida:
    - Pago por uso (OPEX)
    - Flexibilidad
    - Escalado
    - CAPEX
  - Detrás de las tecnologías en la nube, estás las técnicas de virtualización (concretamente virtualización de plataformas)
    - IaaS: Máquinas (o contenedores) individuales, clústers de computación
    - PaaS: Servicios gestionados

Repaso: paradigma de ejecución de Hadoop

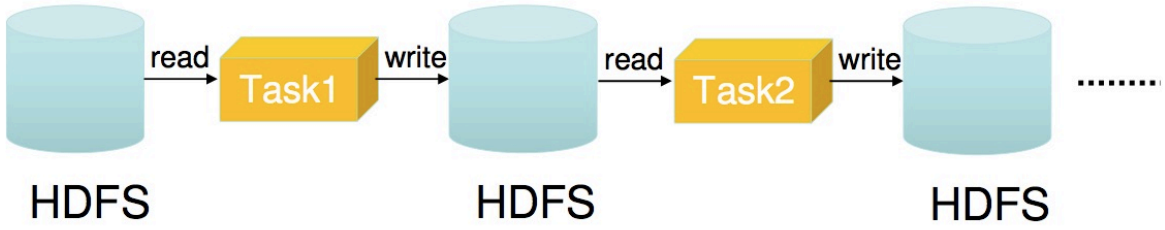
- ¿Cómo se ejecuta una aplicación Hadoop?



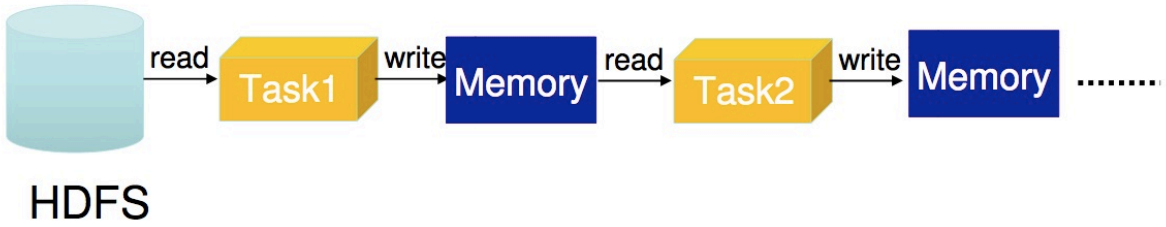
Repaso: paradigma de ejecución de Hadoop

- Hadoop vs Spark

Hadoop:

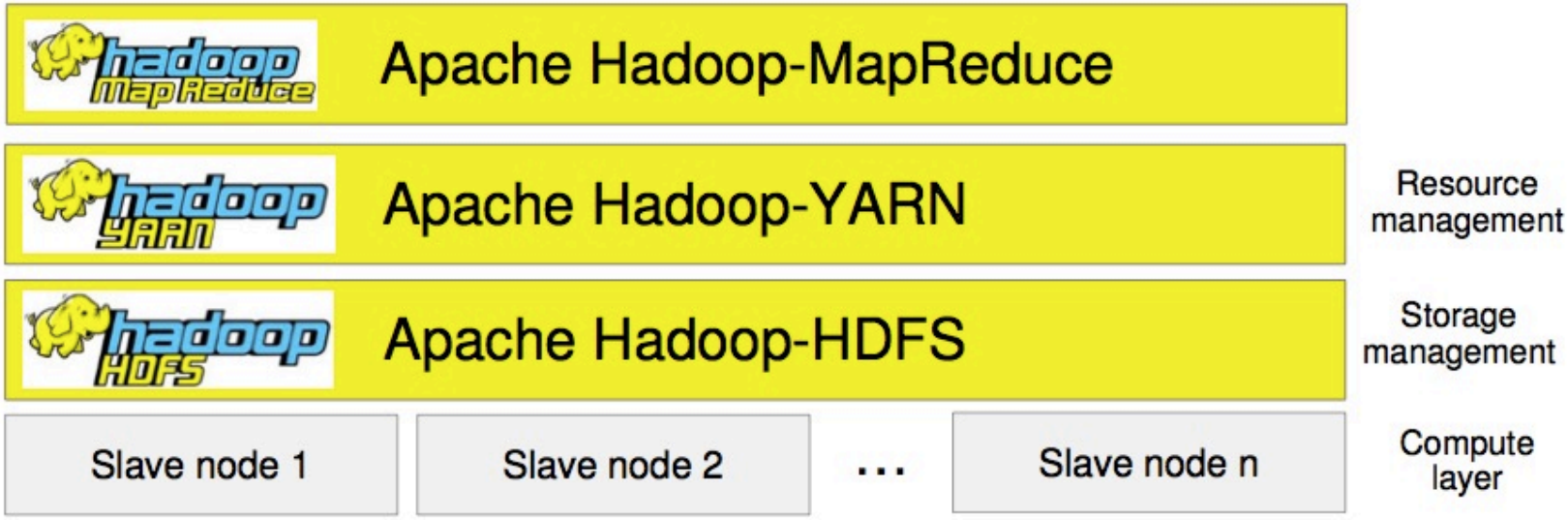


Spark:



Repaso: paradigma de ejecución de Hadoop

- Arquitectura de servicios en un clúster Hadoop

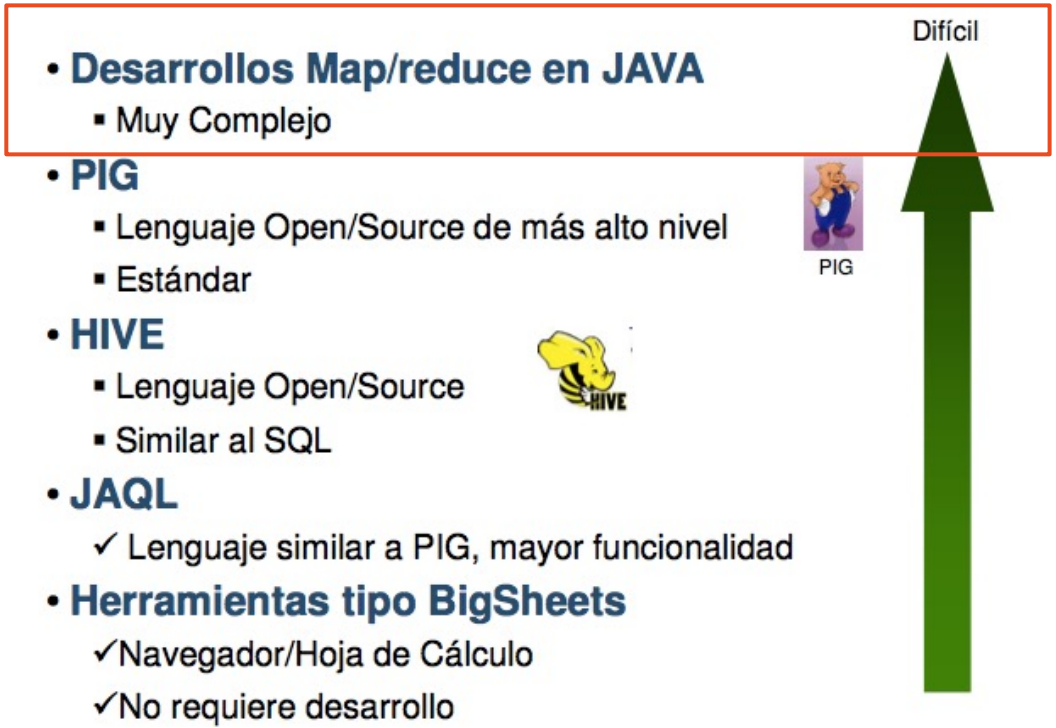


Repaso: paradigma de ejecución de Hadoop

- Hadoop es…
  - Framework de desarrollo para aplicaciones Big Data
    - Paradigma MapReduce
      - Google™
  - Entorno de ejecución
    - Aplicaciones tipo batch
    - Lectura intensiva
  - Se encarga de almacenar los datos generados
  - Escalable

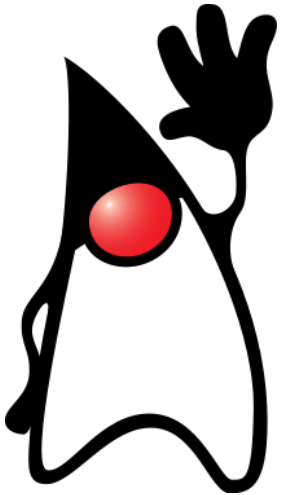
Repaso: paradigma de ejecución de Hadoop

- ¿Cómo desarrollar aplicaciones Hadoop?



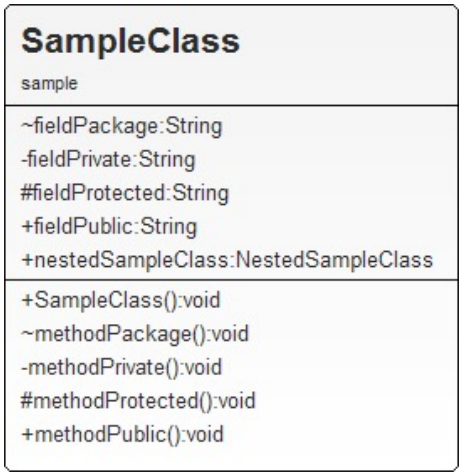
El lenguaje de programación Java

- Creado en el año 1991 (oak) por *Sun Microsystems*
- Objetivos de diseño:
  - Lenguaje orientado objetos
  - Máquina virtual de Java
  - Fácil de utilizar
  - Ejecución de código remoto y soporte de red



EL lenguaje de programación Java

- Lenguaje orientado a objetos
  - Clases
  - Objetos
  - Métodos
  - Parámetros por referencia



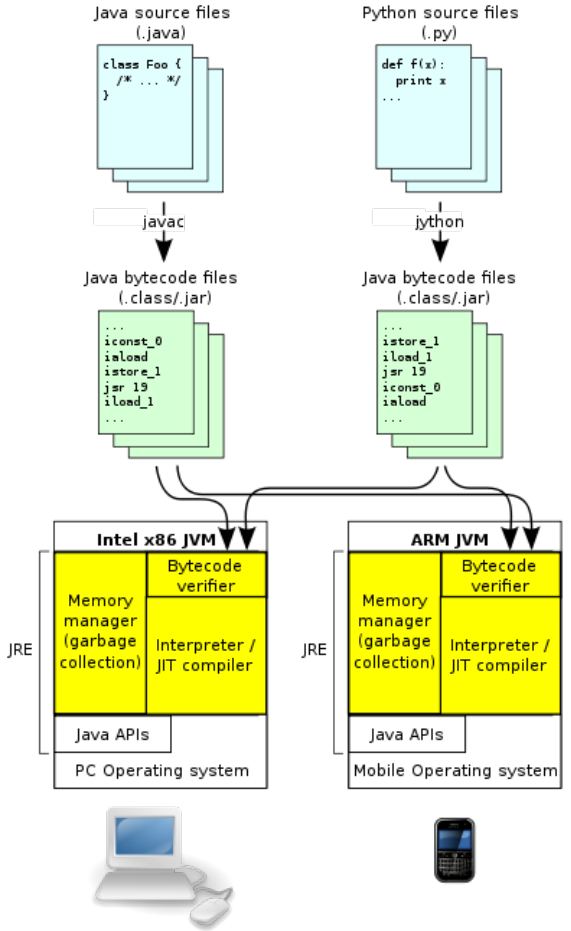
Definición del "tipo de variable" más completa que en otros lenguajes: campos y métodos (y accesibilidad de los mismos).

- ¿Qué operaciones puedes hacerse sobre un objeto?
- ¿Qué campos/atributos definen un objeto?
- ¿Qué métodos y atributos son accesibles y a quién?

static: campos comunes a todos los objetos de una misma clase

EL lenguaje de programación Java

- Máquina Virtual de Java (JVM)
  - Portabilidad
    - Cualquier dispositivo
    - Cualquier SO
    - Java > ByteCode > Código ejecutable
  - Menor rendimiento
    - Etapas intermedias de ejecución
    - Dificultad de programación MP/MC
  - Aislamiento



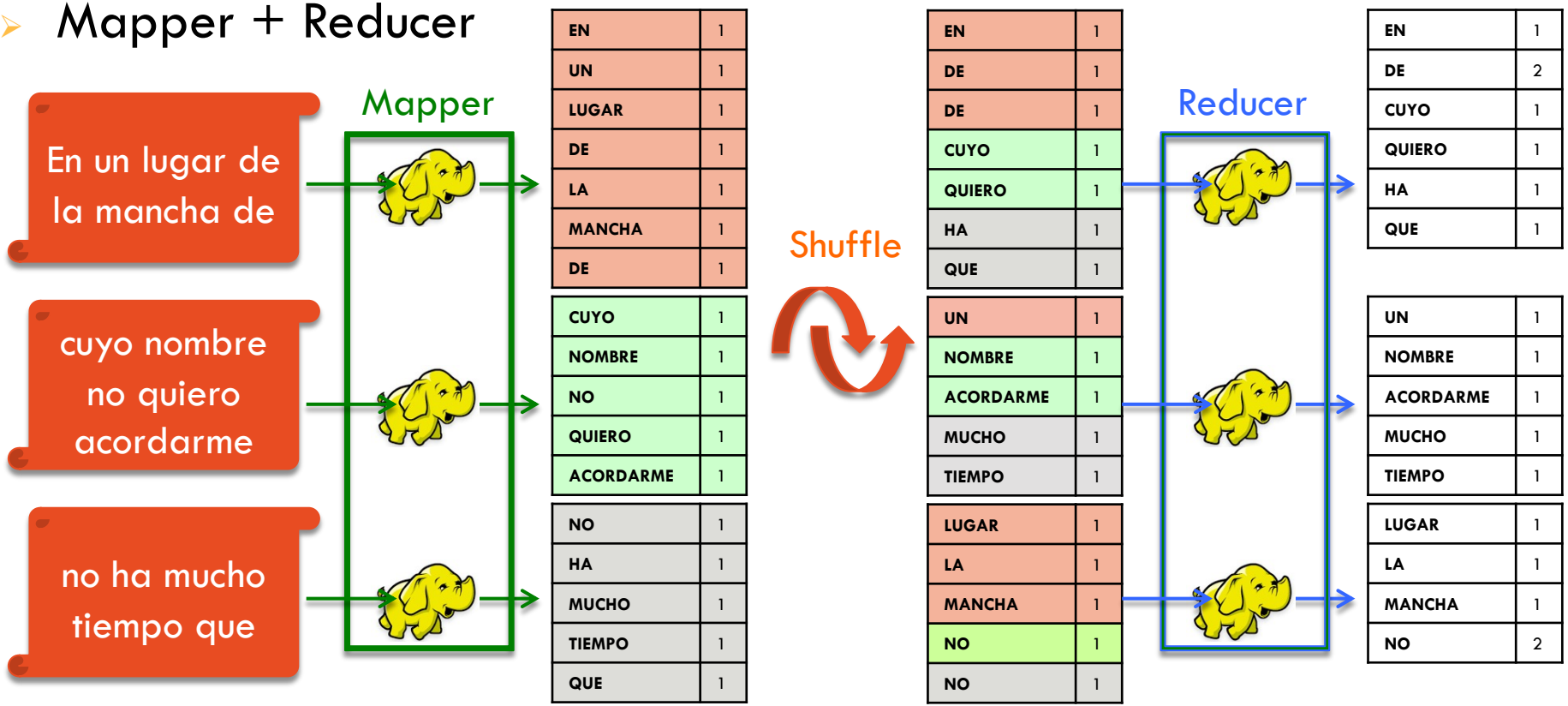
EL lenguaje de programación Java

- Fácil de utilizar
  - La comunidad Java dispone de innumerables paquetes con código listo para su reutilización
    - Java Beans
  - Excepciones
  - Sintaxis similar a C
  - Entornos de programación IDE: NetBeans, Eclipse

- Gestión de memoria
  - Garbage collector
    - El programador no se preocupa de liberar recursos una vez se ha terminado con ellos
    - Se almacena un número de referencias por objeto. Cuando llega a 0, el GC lo liberará la próxima vez que se ejecute

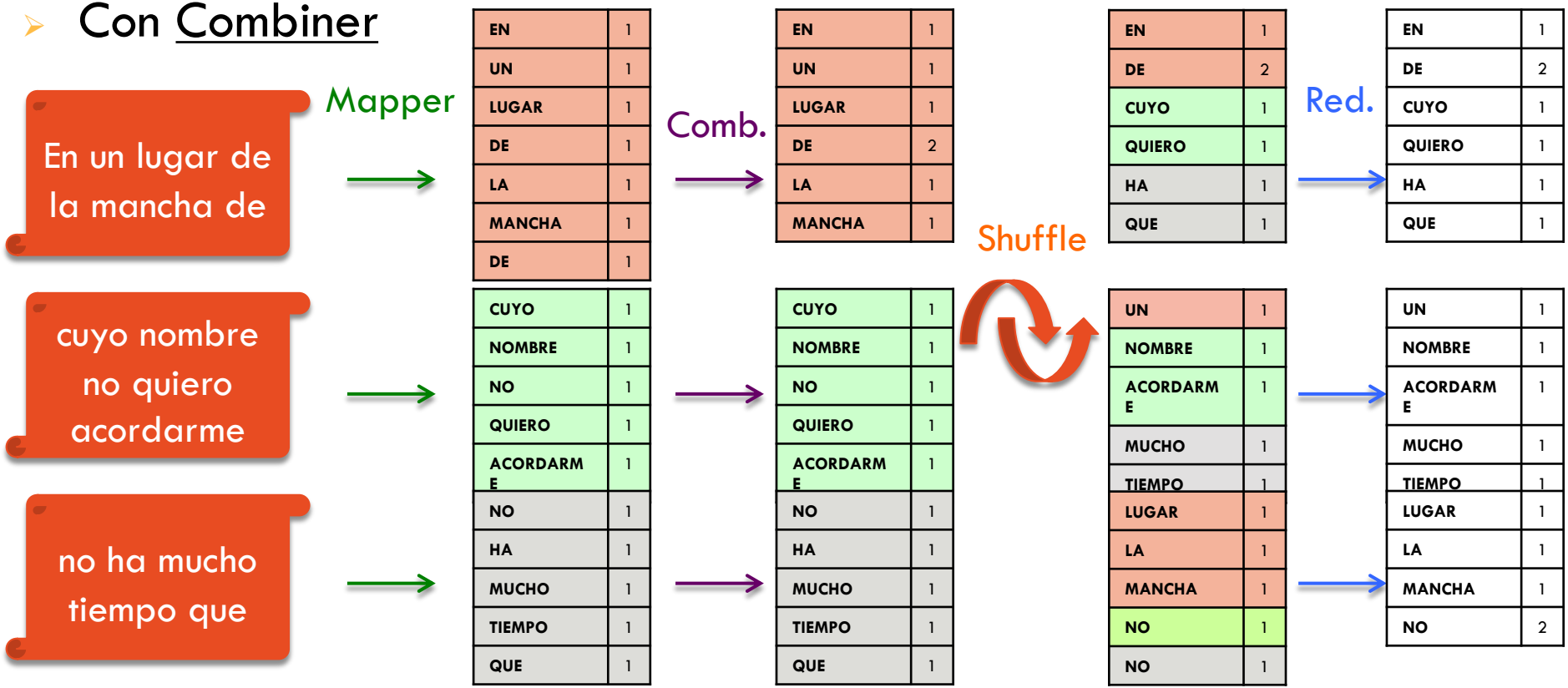
Caso de ejemplo: WordCount

- Mapper + Reducer



Caso de ejemplo: WordCount

- Con Combiner



Caso de ejemplo: WordCount

- Ejemplo de salida

(Y	1
(a	1
(al	1
(como	1
(creyendo	1
(de	2
…	
Anoche	1
Anteo,	1
Antequera	1
Antequera.	1
Antequera;	1
Antes	1
…	

Has	2
Haz	1
He	1
Hechas,	2
Hecho	5
Hechos	1
Henares	1
Henares,	1
Hermandad	2
Hermandad,	1
…	
has	21
has,	1
hasme	1
hasta	53
hato	1
hato.	1
hay	52
hay,	2
…	

Caso de ejemplo: WordCount

Extra

```
package org.apache.hadoop.examples;

import java.io.IOException;
import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;

public class WordCount {

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

  public static class IntSumReducer
       extends Reducer<Text,IntWritable,Text,IntWritable> { ... }

  public static void main(String[] args) throws Exception { ... }
}
```

Inclusión de paquetes necesarios

Definición de clase Mapper
(herencia de clase)

Definición de clase Reducer
(herencia de clase)

Configuración del entorno MapReduce

Caso de ejemplo: WordCount

Extra

Configuración del trabajo MR

```
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> <out>");
    System.exit(2);
  }
  Job job = new Job(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);
  FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
  FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
  System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

- Configuración de parseo de los argumentos de entrada

- Clases Mapper, Combiner y Reducer

- Clases de las claves y valores de salida

- Configuración de los parámetros de entrada y salida

Caso de ejemplo: WordCount

Extra

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

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

  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);
    }
  }
}
```

Separamos palabras de una misma línea, y las procesamos por separado

Clase Mapper

La clase abstracta Reducer de la que hereda, fuerza tipo de argumentos de entrada, y la implementación del método map.

Método map

Entrada:

- Clave (en este caso no utilizada, pero contemplada para MR multi-etapa)
- Valor (cada línea de texto)

Salida:

- Clave (palabra individual)
- Valor (en nuestro caso, 1)

Caso de ejemplo: WordCount

Extra

```
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);
  }
}
```

Clase Reducer

La clase abstracta Reducer de la que hereda, fuerza tipo de argumentos de entrada, y la implementación del método reduce.

Método reduce

Entrada: par clave, valor generado en la etapa map

Salida: nuevos pares clave, valor

- clave: la misma que antes
- valor: suma de todos los *valor* asociados a una misma clave.

Caso de ejemplo: WordCount

Extra

```
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);
  }
}
```

Clase Combiner

También implementa la interfaz Reucer. Reduce la cantidad de datos que se barajean para reducir movimiento de datos y "facilitar" las cosas al reducer. En este ejemplo las clases coinciden.

Combiner vs Reducer

- Los formatos tanto de entrada como de salida deben coincidir con los de salida del Mapper
- Un Combiner toma sus datos de entrada de un solo Mapper
- Sólo pueden usarse con funciones conmutativas y asociativas (o alterarían el resultado)

Modos de despliegue de Hadoop

- Hadoop se puede desplegar de tres maneras distintas:
  - Standalone
  - Pseudo-Distributed
  - Fully Distributed

¿Preguntas?