# Sistemas de almacenamiento y procesado distribuido

## Tema 3. Procesamiento en batch

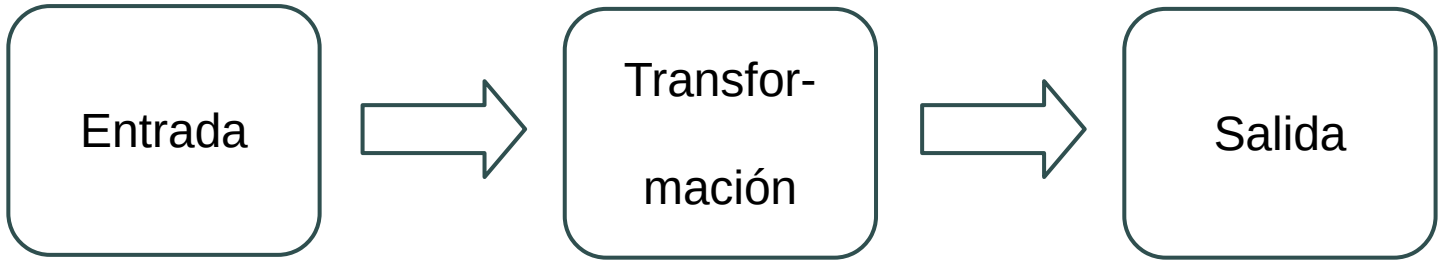
# Contenido

- Introducción
- MapReduce
- Hadoop
- Spark

# Introducción

## Procesamiento datos

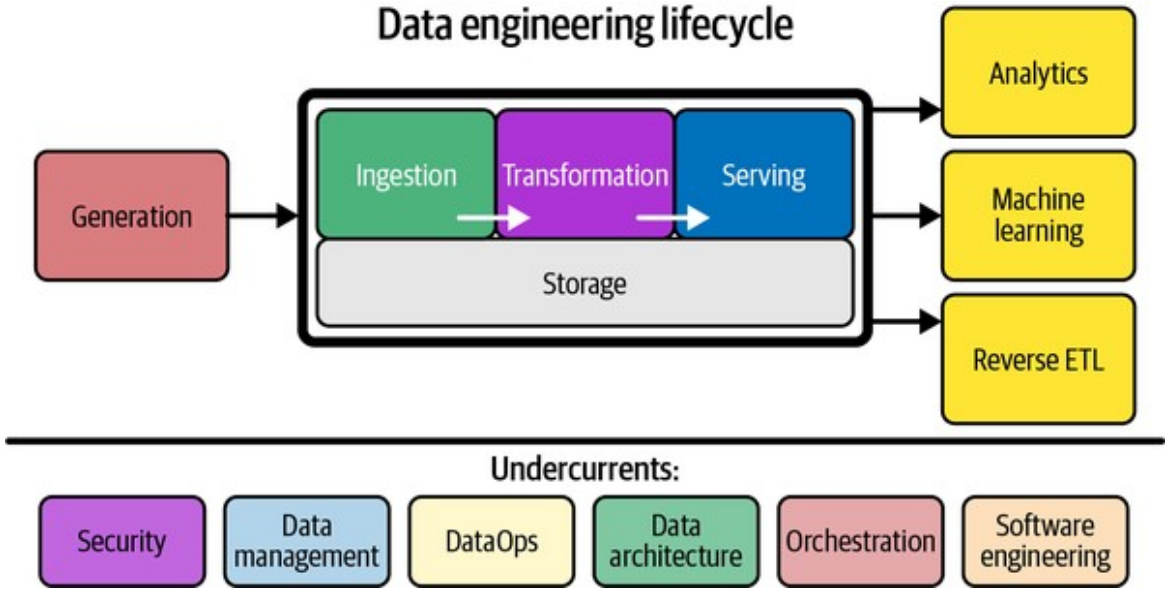
- Tiene lugar siempre que se produce una transformación de datos
- Las fases principales son siempre las mismas: obtener la entrada, transformación de datos, generar la salida
- La transformación suele incluir diversas subfases: limpieza, conversión, filtrado, agrupación, etc.



# Introducción

## Procesamiento datos

- Se enmarca en procesos más complejos: ciclo de vida de la ingeniería de datos




Fundamentals of Data Engineering. Joe Reis and Matt Housley. O'Reilly Media, Inc, 2022.

# Introducción

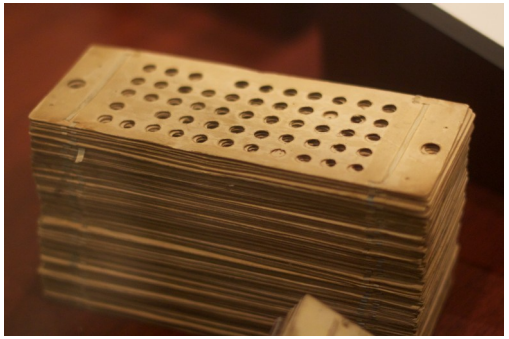
## Procesamiento datos > Tipos

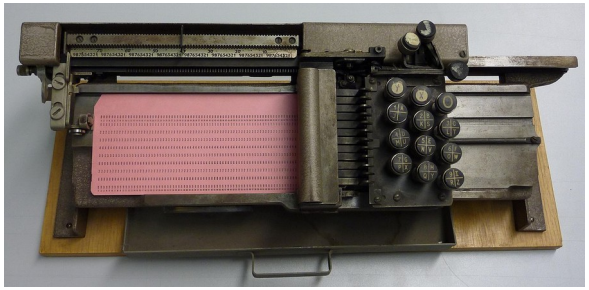
- Batch vs streaming
  - Batch: se procesan los datos en bloque
  - Streaming: se procesan los datos de manera contínua
- Online vs Offline
  - Online: el cliente envía consulta y espera respuesta de manera interactiva
  - Offline: el cliente no espera respuesta; se procesa en modo desconectado

# Introducción

## Procesamiento en batch

- El término *batch* (lote) procede de los tiempos de tarjetas perforadas.
- Cada programa estaba contenido en un lote (batch) de tarjetas.
- Los programas se ejecutaban uno tras otro.

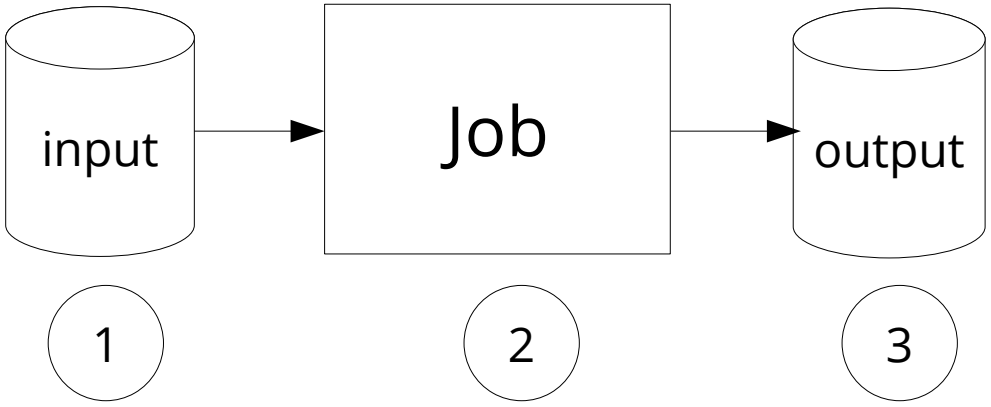




# Introducción

## Procesamiento en batch

- Un proceso batch (job)
  1. Lee grandes volúmenes de datos (TB)
  2. Los procesa y genera datos de salida
  3. Escribe los resultados



# Introducción

## Procesamiento en batch

- Un proceso batch (job)
  - No precisa interacción del usuario (off-line)
  - Se ejecuta de manera periódica/planificada
  - Medidas de rendimiento: tiempo de proceso, tareas/segundo

# Introducción

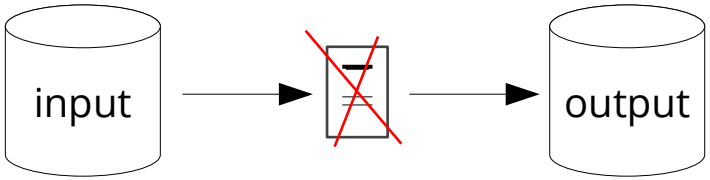
## Procesamiento en batch

- Ejemplos de procesos batch:
  - Integración de información obtenida de múltiples fuentes: procesos ETL (Extract-Transform-Load)
  - Actualización (construcción) de índices sobre grandes volúmenes de datos
  - Aplicación periódica que aplica reglas de negocio sobre grandes volúmenes de datos
  - Machine learning (iterativo)
  - etc.

# Introducción

## Procesamiento datos a escala

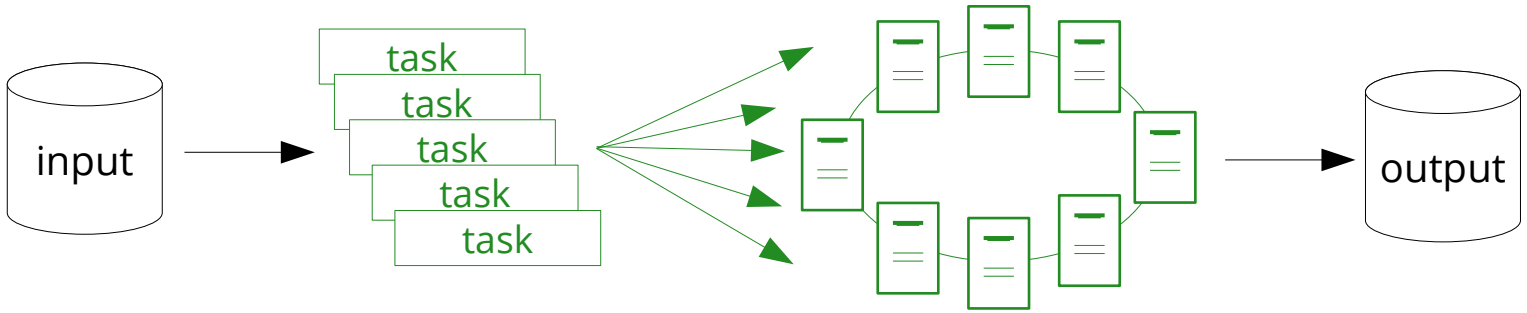
- Cuando el volumen de datos es muy grande
- Cuando el tiempo de proceso es muy grande
- Los cálculos no pueden tener lugar en una única máquina



# Introducción

## Procesamiento datos a escala

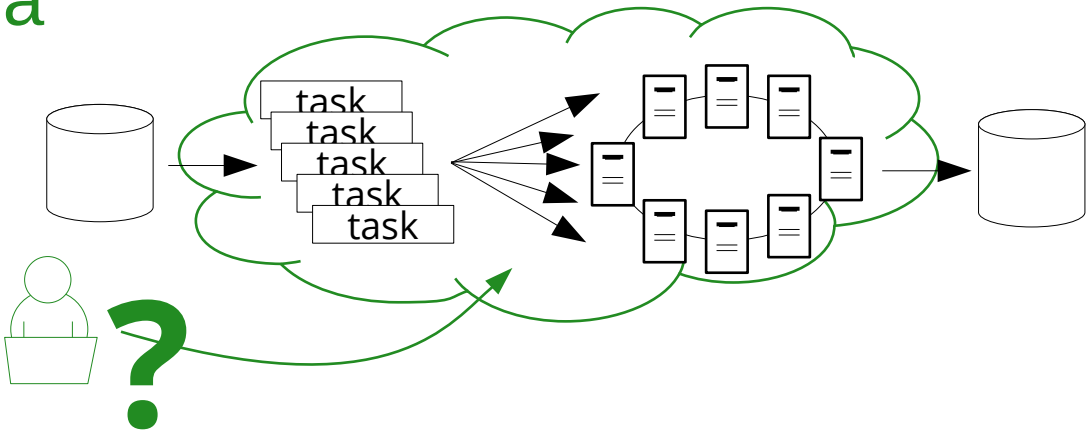
- Deben ejecutarse en un clúster de máquinas, con tareas en paralelo, que se comunican entre sí



# Introducción

## Procesamiento datos a escala

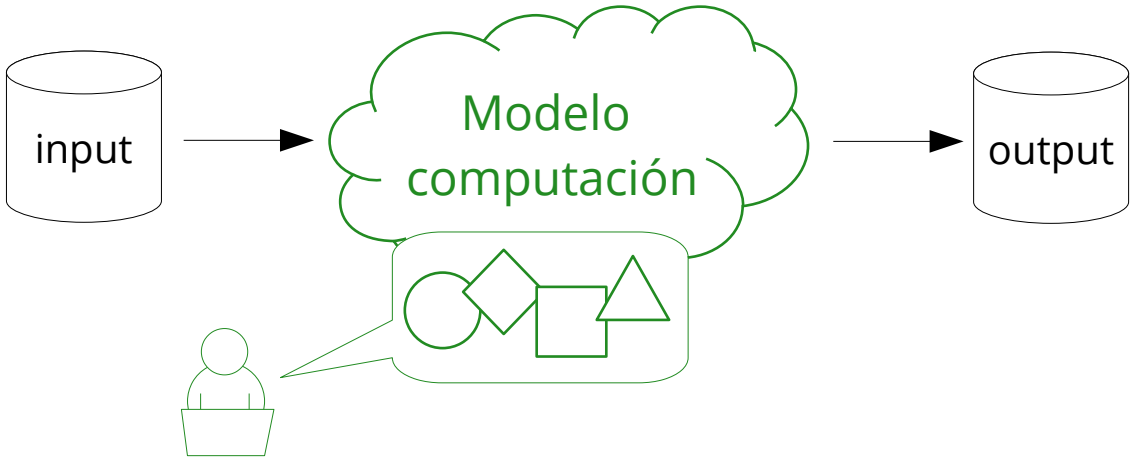
- Deben ejecutarse en un clúster de máquinas, con tareas en paralelo, que se comunican entre sí
- Es muy complejo: dividir en tareas, enviar y recibir datos, sincronizar, fallos, etc.
- Para simplificar, se utilizan modelos de computación distribuida



# Introducción

## Modelos de computación distribuida

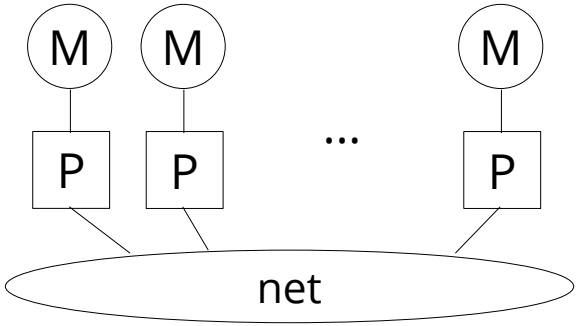
- Proporciona abstracciones que facilitan la programación de algoritmos distribuidos



# Introducción

## Modelos de computación distribuida

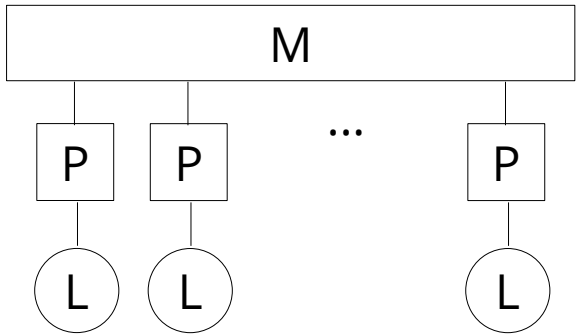
- Ejemplos
  - MPI (Message Passing Interface): mensajes



# Introducción

## Modelos de computación distribuida

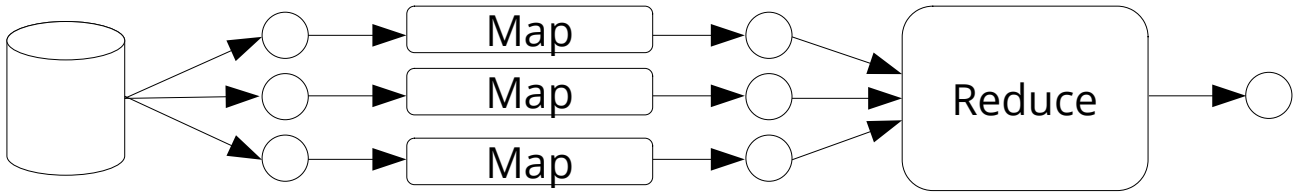
- Ejemplos
  - MPI (Message Passing Interface): mensajes
  - Memoria compartida: por debajo hay mensajes



# Introducción

## Modelos de computación distribuida

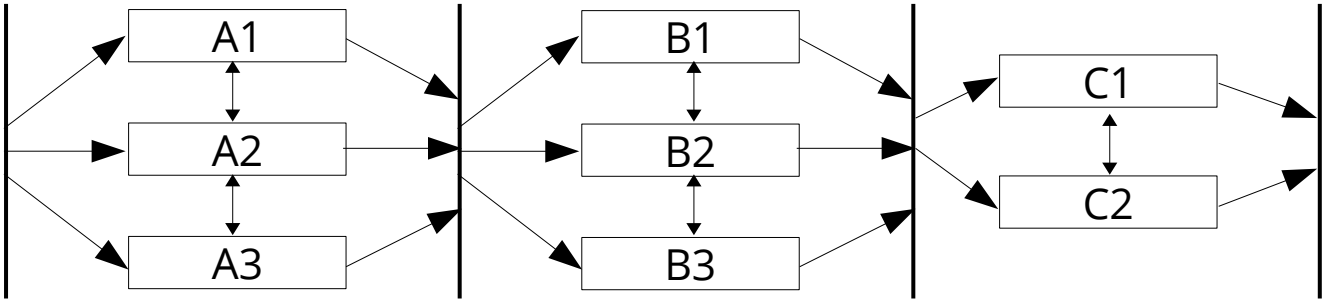
- Ejemplos
  - MPI (Message Passing Interface): mensajes
  - Memoria compartida: por debajo hay mensajes
  - MapReduce: dos fases, tareas independientes



# Introducción

## Modelos de computación distribuida

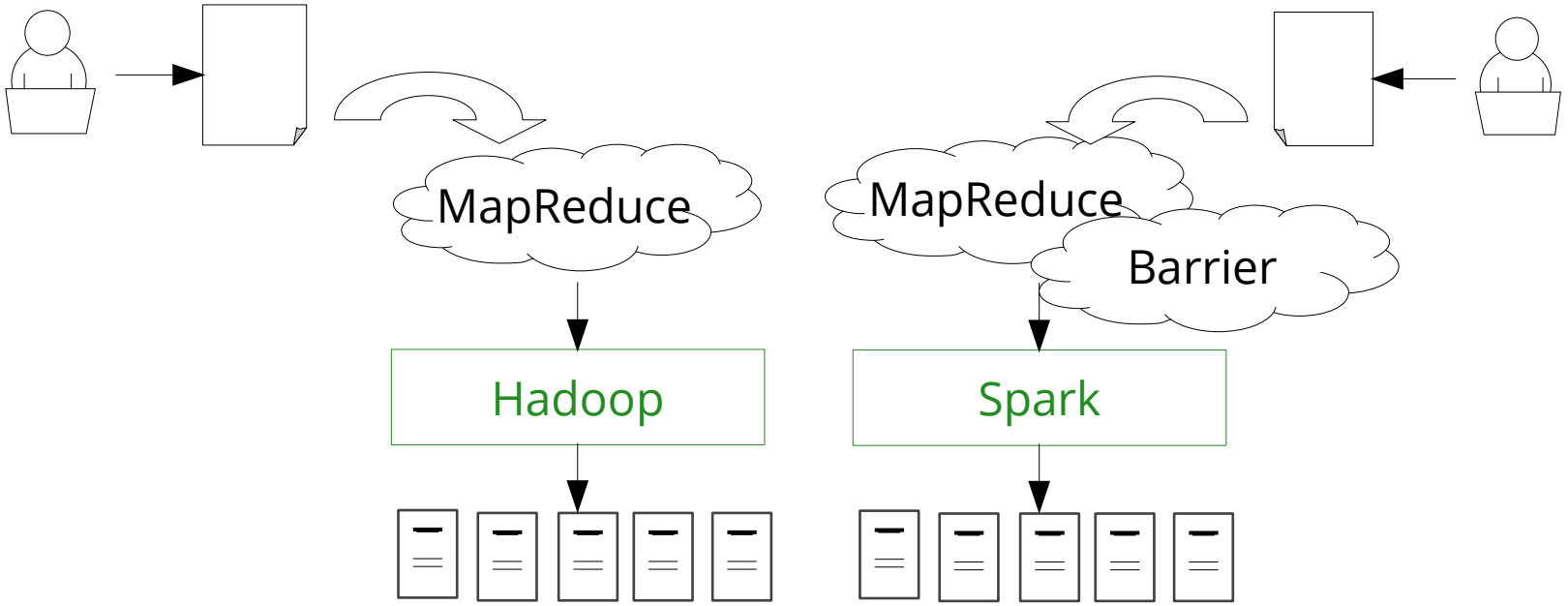
- Ejemplos
  - MPI (Message Passing Interface): mensajes
  - Memoria compartida: por debajo hay mensajes
  - MapReduce: dos fases, tareas independientes
  - Barrier: barreras de sincronización, tareas dependientes



# Introducción

## Plataforma de computación distribuida

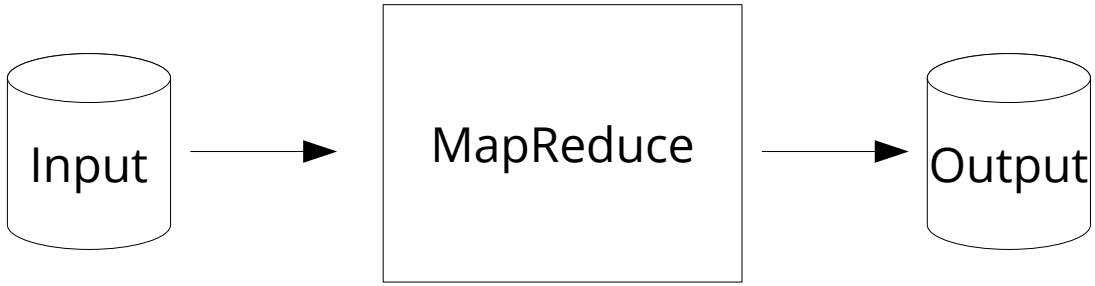
- El modelo es implementado por una plataforma de computación distribuida: Hadoop, Spark, etc.



# MapReduce

## ¿Qué es?

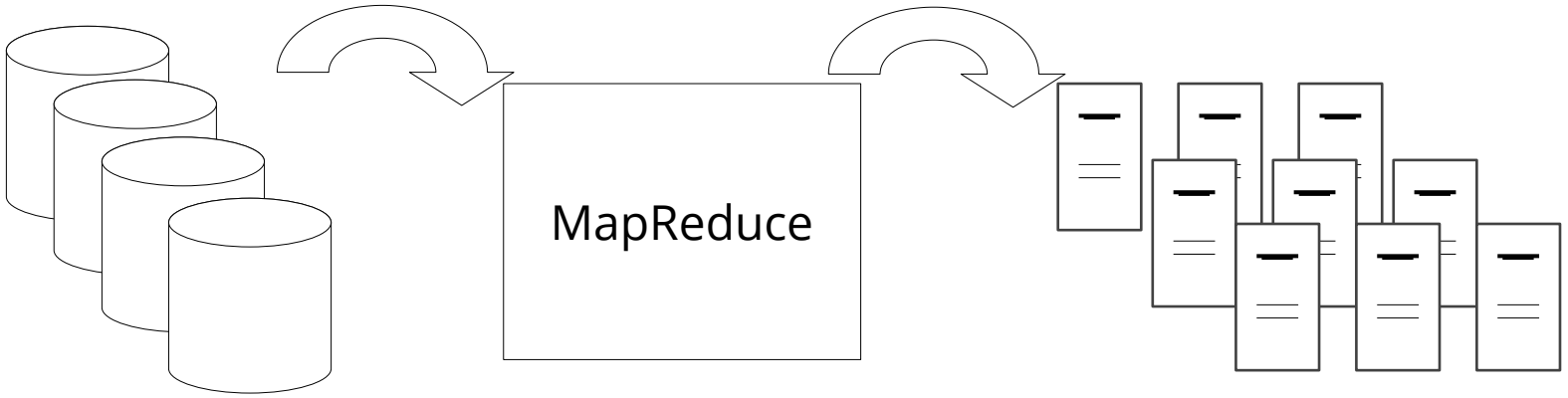
- Es un modelo de programación en batch



# MapReduce

## ¿Qué es?

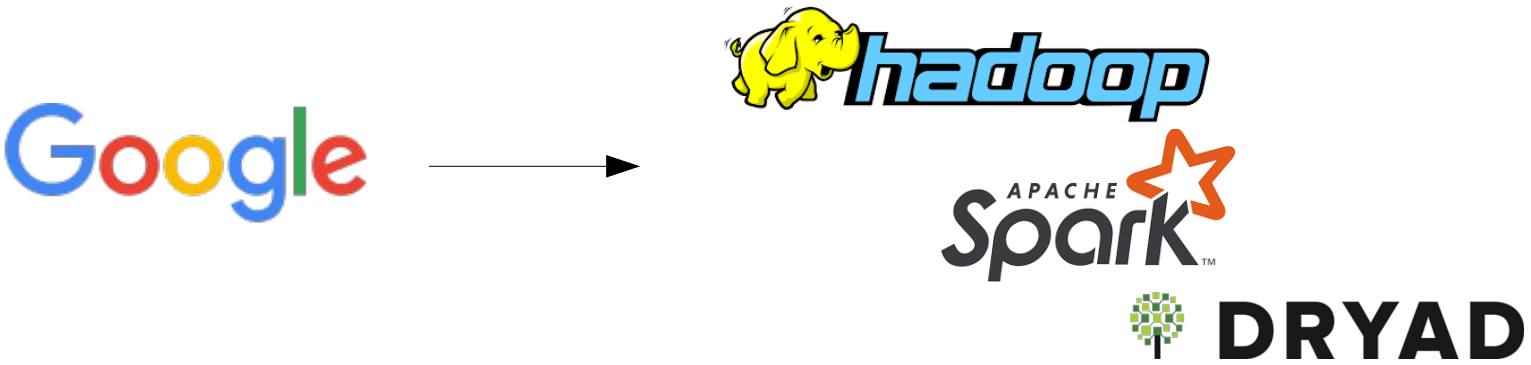
- Es un modelo de programación en batch
- Permite procesar grandes volúmenes de datos en paralelo en grandes clústers



# MapReduce

## ¿Qué es?

- Es un modelo de programación en batch
- Permite procesar grandes volúmenes de datos en paralelo en grandes clústers
- Original de Google; hoy en día existen múltiples implementaciones (Hadoop, Spark, Dryad, etc.)



# MapReduce

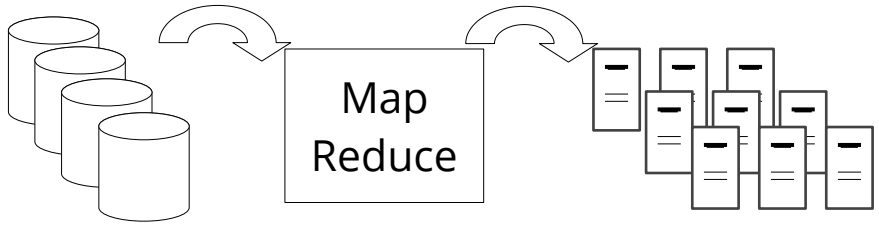
## ¿Qué es?

- Es un modelo de programación en batch
- Permite procesar grandes volúmenes de datos en paralelo en grandes clústers
- Original de Google; hoy en día existen múltiples implementaciones (Hadoop, Spark, Dryad, etc.)
- Supone el comienzo de la "era Big Data"
- A partir de este modelo han surgido diversas extensiones que conforman el panorama Big Data actual

# MapReduce

## ¿Qué es?

- Proporciona un esquema genérico aplicable a múltiples casos
- Abstrae las complejidades de bajo nivel (fraccionar datos, distribuir tareas a máquinas, gestionar fallos, equilibrar carga trabajo, etc.)



# MapReduce

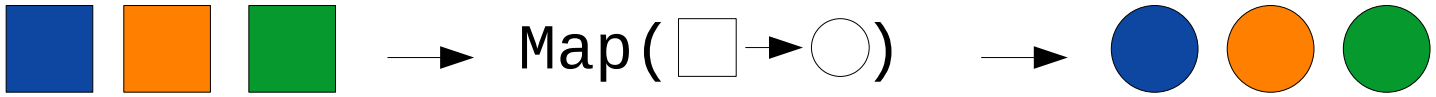
## ¿Qué es?

- Proporciona un esquema genérico aplicable a múltiples casos
- Abstrae las complejidades de bajo nivel (fraccionar datos, distribuir tareas a máquinas, gestionar fallos, equilibrar carga trabajo, etc.)
- Se basa en las primitivas funcionales Map y Reduce

# MapReduce

## Map

- Transforma datos de entrada en datos de salida

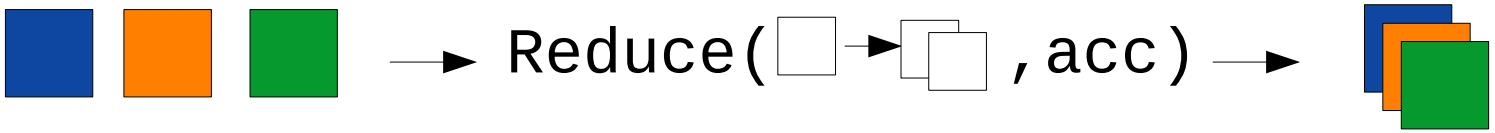



```
map(fun, iter): iter

def double(n): return n + n

input = [1, 2, 3, 4]
output = map(double, input)
output = map(lambda a: a+a, input)
print(list(output))    # [2, 4, 6, 8]
```

# MapReduce

## Reduce (fold)

- Aplica una función a todos los elementos, y acumula: agregación, búsquedas, etc.

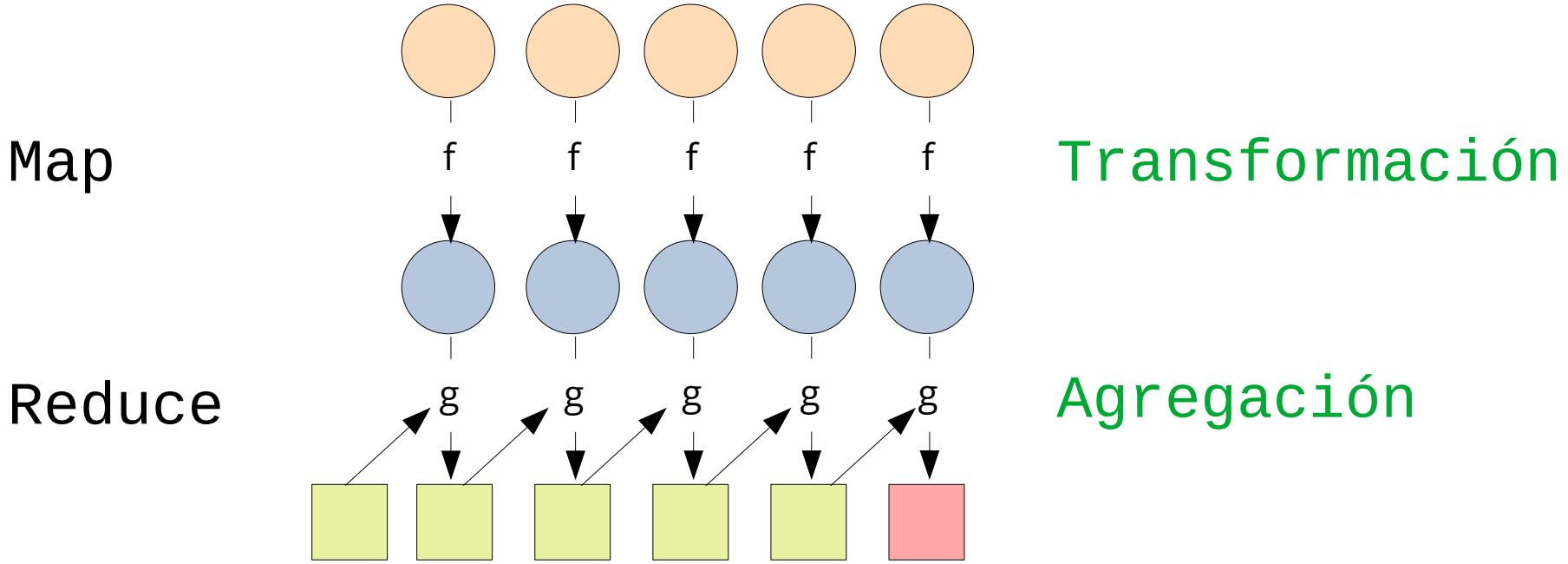



```
reduce(fun, iter, accum=None): any

def sum(a,b): return a+b

input = [1, 2, 3, 4]
output = reduce(sum, input, 0)
output = reduce(lambda a,b: a+b, input, 0)
print(output)    # 10
```

# MapReduce

## Map + Reduce

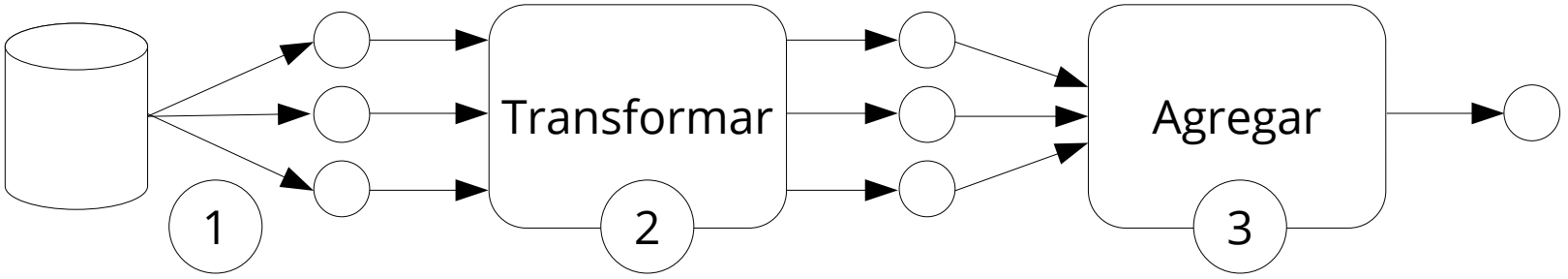
- ¿Por qué no combinarlos?



# MapReduce

## Map + Reduce

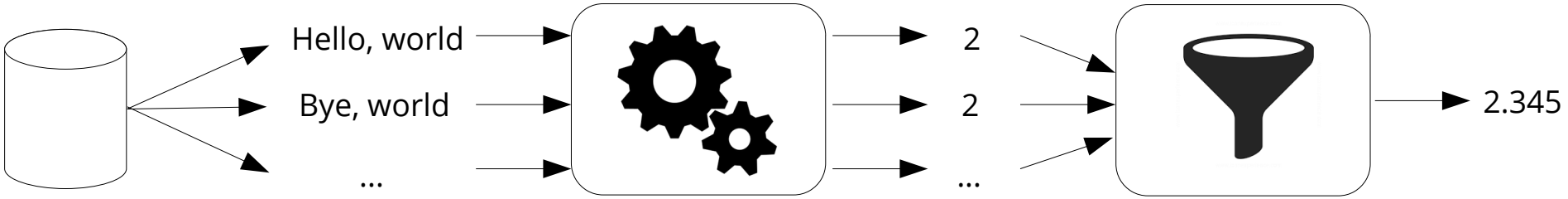
- Múltiples problemas de procesamiento de datos se pueden adaptar a este patrón
  1. Se extrae información de una fuente de datos
  2. Se transforma la información
  3. Se efectúa algún cálculo agregado



# MapReduce

## Map + Reduce

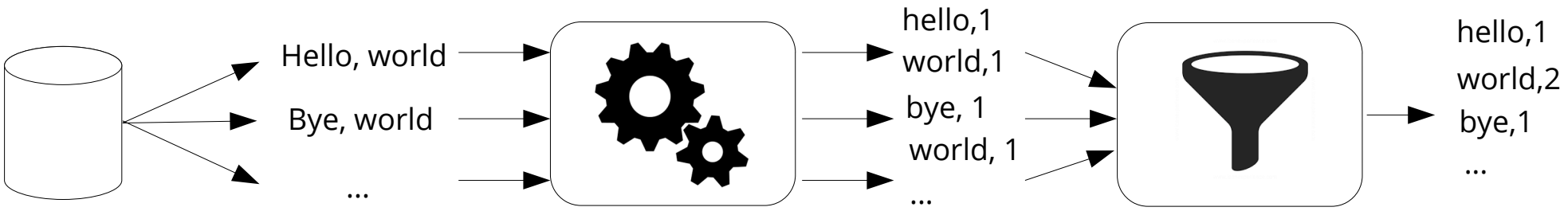
- Ejemplo: Contar nº palabras total en texto
  - Fuente de datos: líneas de texto
  - Transformación: parsear (split), contar, emitir suma
  - Agregación: sumar



# MapReduce

## Map + Reduce

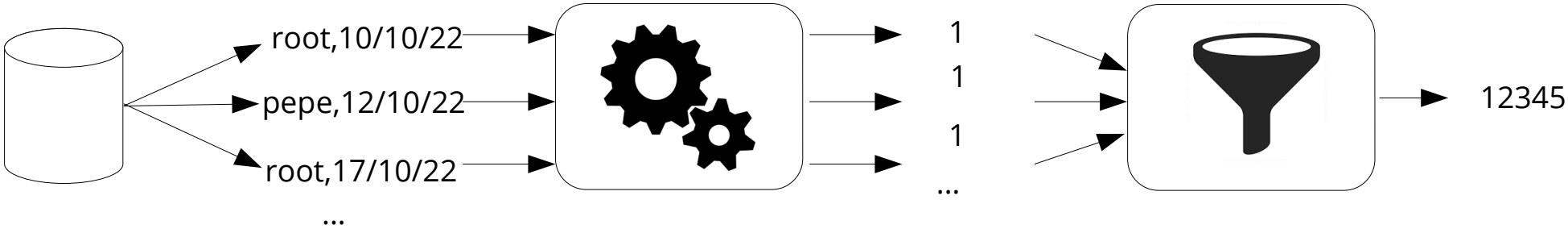
- Ejemplo: Contar nº palabras en texto
  - Fuente de datos: líneas de texto
  - Transformación: parsear (split) y emitir conteo individual (word,1)
  - Agregación: agrupar y sumar



# MapReduce

## Map + Reduce

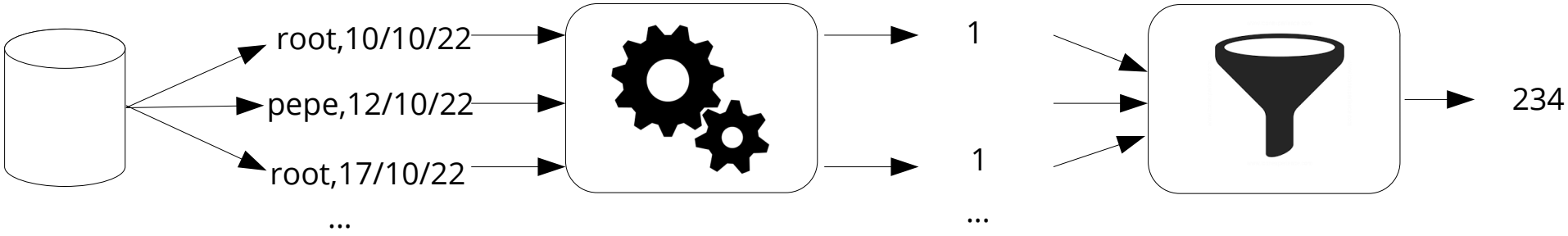
- Ejemplo: Número accesos total al sistema
  - Fuente de datos: entradas en log [user, fecha]
  - Transformación: emitir 1
  - Agregación: sumar



# MapReduce

## Map + Reduce

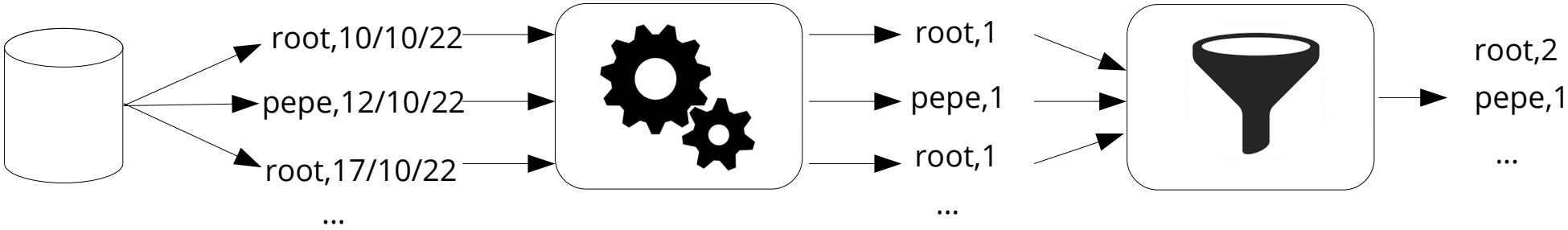
- Ejemplo: Número accesos de root al sistema
  - Fuente de datos: entradas en log [user, fecha]
  - Transformación: parsear, filtrar y emitir 1
  - Agregación: sumar



# MapReduce

## Map + Reduce

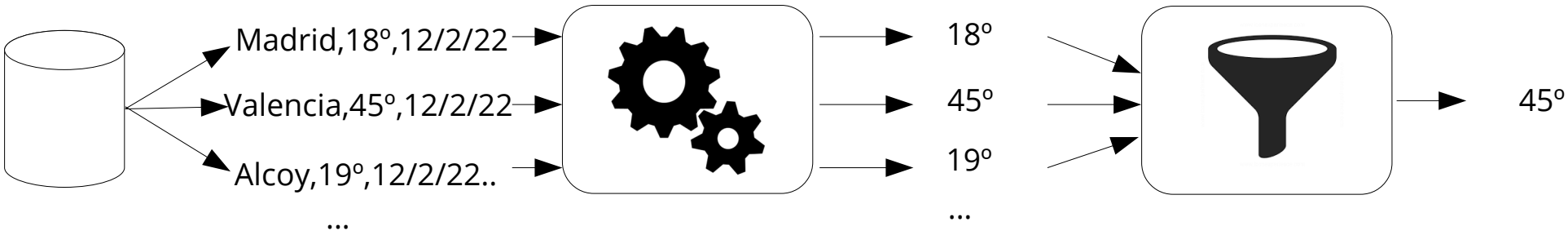
- Ejemplo: Número accesos al sistema por usuario
  - Fuente de datos: entradas en log [user, fecha]
  - Transformación: parsear, y emitir conteo individual (usr,1)
  - Agregación: agrupar y sumar



# MapReduce

## Map + Reduce

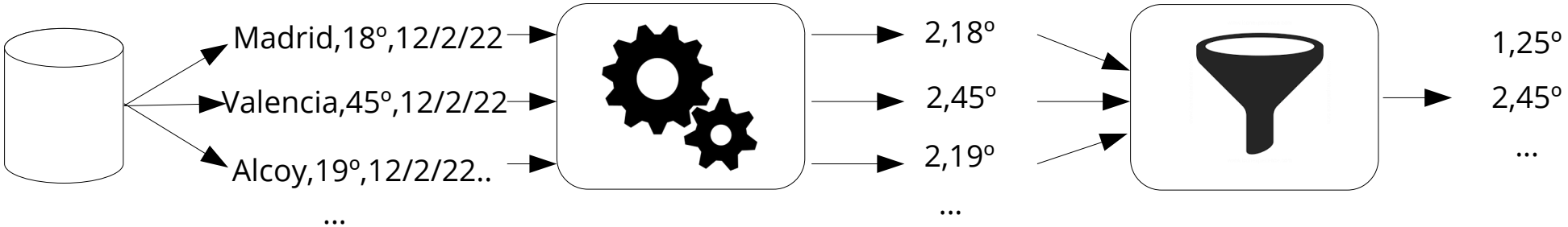
- Ejemplo: Temperatura máxima al año
  - Fuente de datos: entradas en log [ciudad, temp, fecha]
  - Transformación: parsear, extraer temperatura, emitir
  - Agregación: comparar



# MapReduce

## Map + Reduce

- Ejemplo: Temperatura máxima por mes
  - Fuente de datos: entradas en log [ciudad, temp, fecha]
  - Transformación: parsear, emitir (mes,temperatura)
  - Agregación: agrupar y comparar

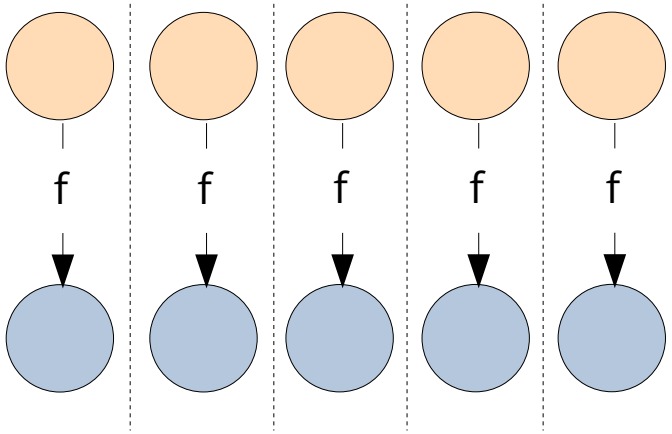


# MapReduce

## Propiedades > Map

- Las tareas Map son independientes: fácilmente paralelizables

Map

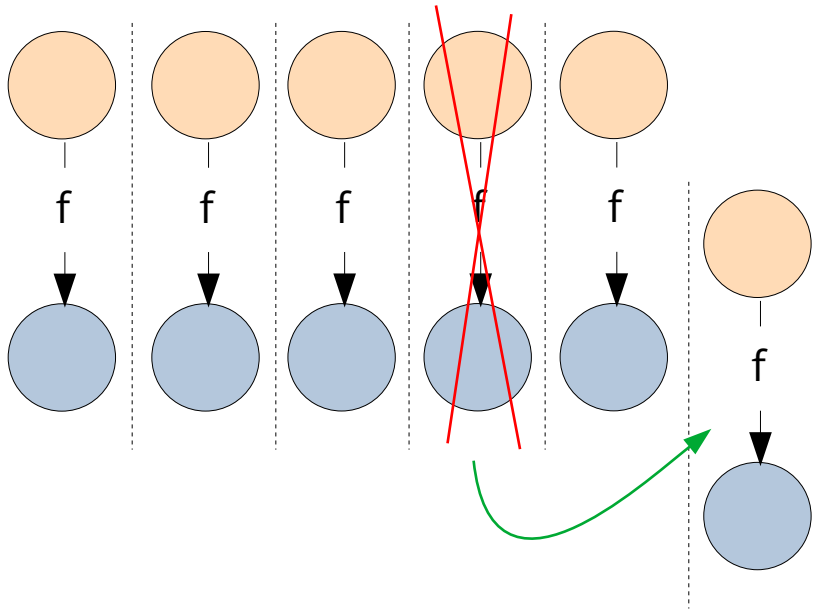


# MapReduce

## Propiedades > Map

- Las tareas Map son independientes: fácilmente paralelizables
- Si una tarea falla, se puede relanzar, con el mismo resultado

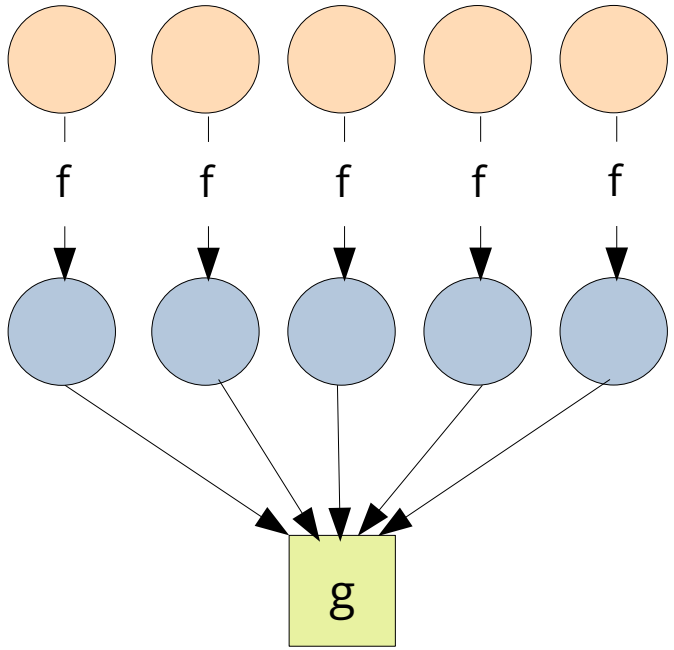
Map



# MapReduce

## Propiedades > Reduce

- Los resultados se acumulan en una tarea Reduce

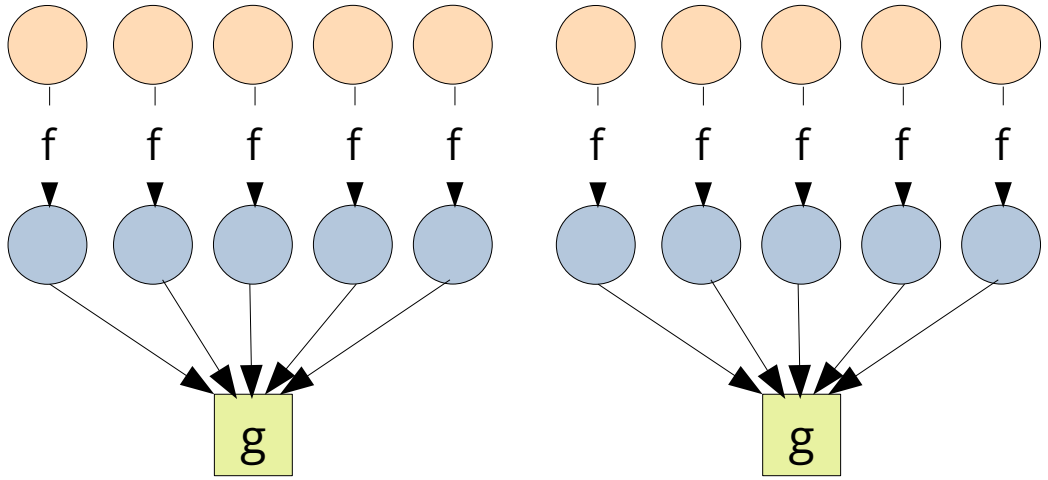



Reduce

# MapReduce

## Propiedades > Reduce

- Los resultados se acumulan en una tarea Reduce
- Se podría paralelizar en varias tareas Reduce, pero con resultados independientes

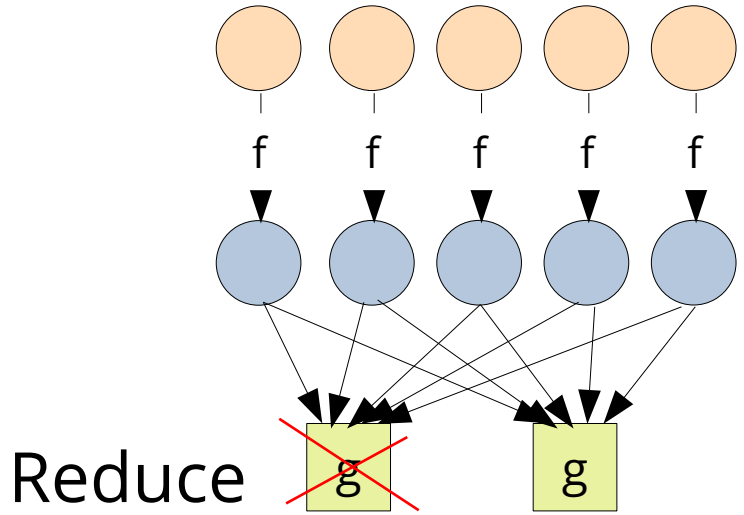



Reduce

# MapReduce

## Propiedades > Reduce

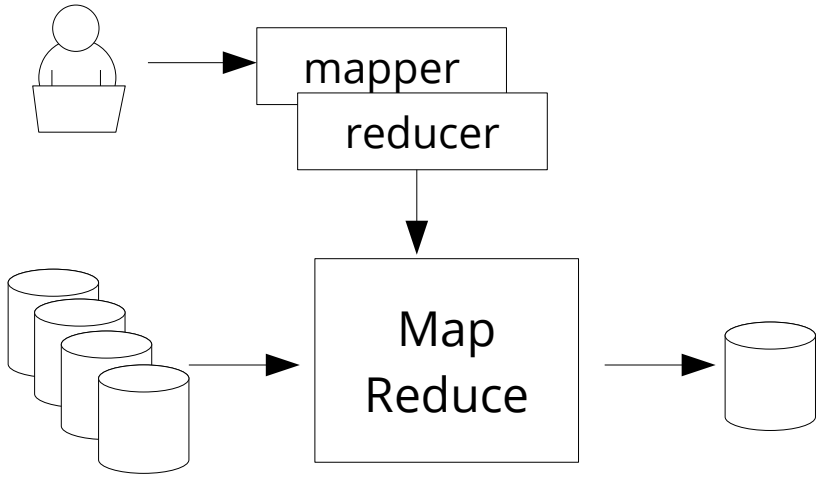
- Los resultados se acumulan en una tarea Reduce
- Se podría paralelizar en varias tareas Reduce, pero con resultados independientes
- Si falla, hay que redistribuir los resultados y recomputar



# MapReduce

## Propiedades > Abstracción

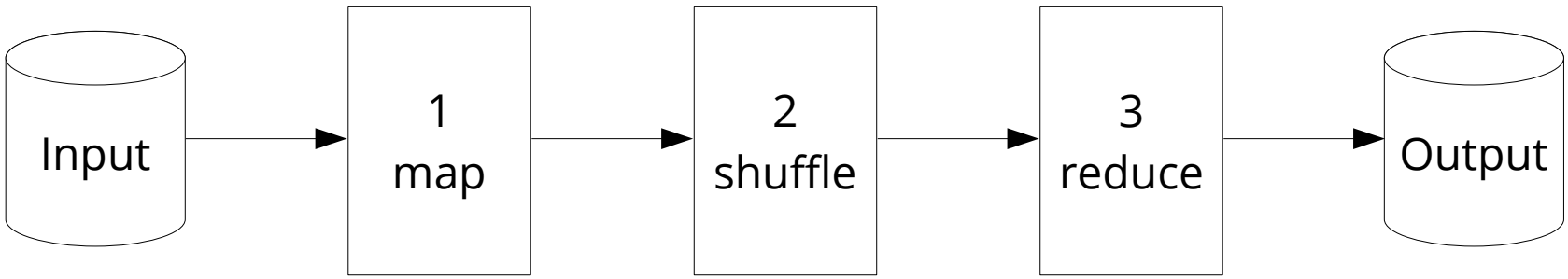
- El programador sólo tiene que codificar el mapper y el reducer
- De todo lo demás (distribución de datos y tareas, ordenación, fallos, etc.) se encarga “el sistema”



# MapReduce

## Implementación

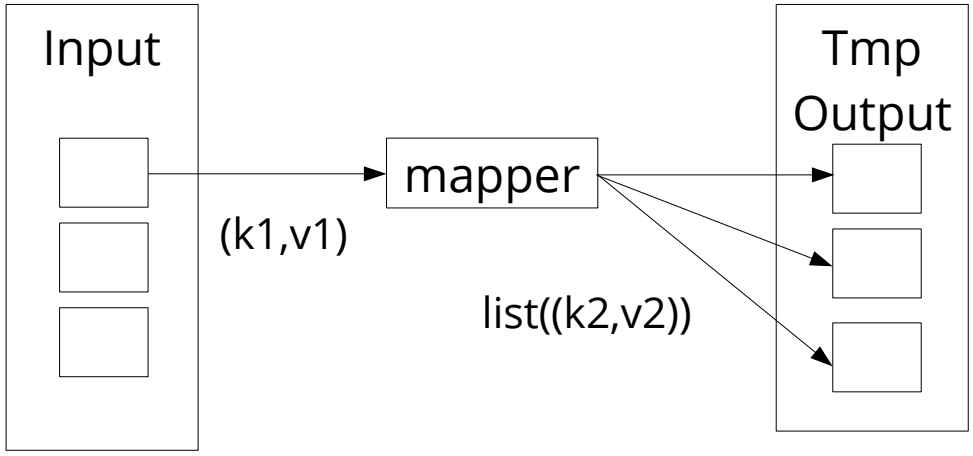
- Generalmente MapReduce se implementa como un proceso con tres fases:
  1. Fase *map*
  2. Fase *shuffle/sort*
  3. Fase *reduce*



# MapReduce

## Implementación > 1. Fase map

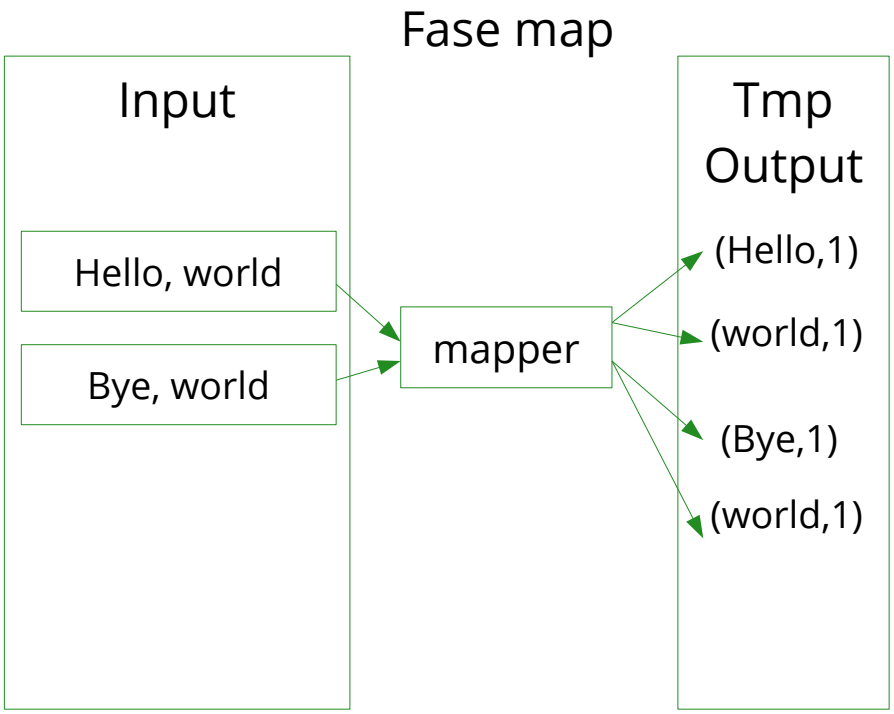
- Una función, el mapper, recibe pares (k1,v1)
- Para cada par, produce cero o más pares (k2,v2), es decir, list((k2,v2))



# MapReduce

## Implementación > 1. Fase map

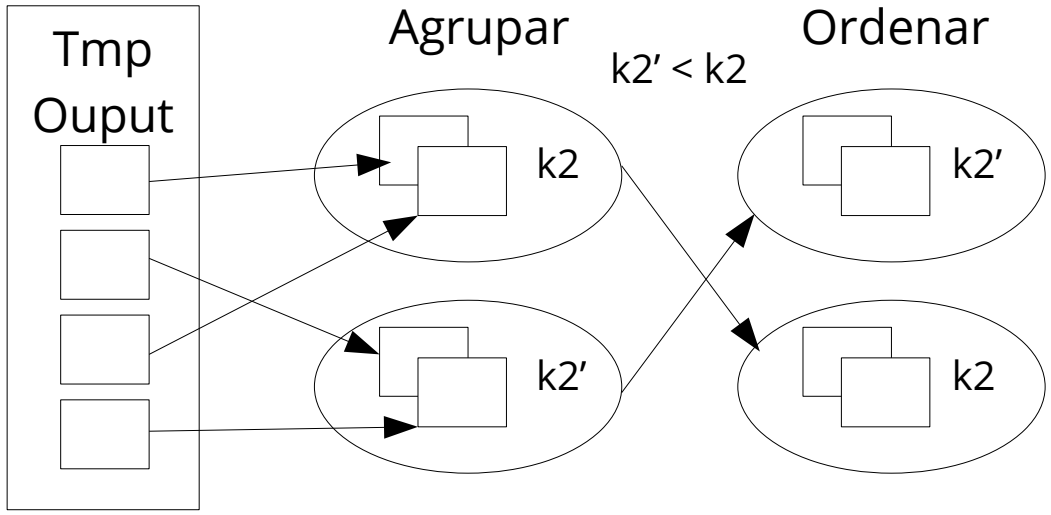
- Ejemplo: contar palabras



# MapReduce

## Implementación > 2. Fase shuffle/sort

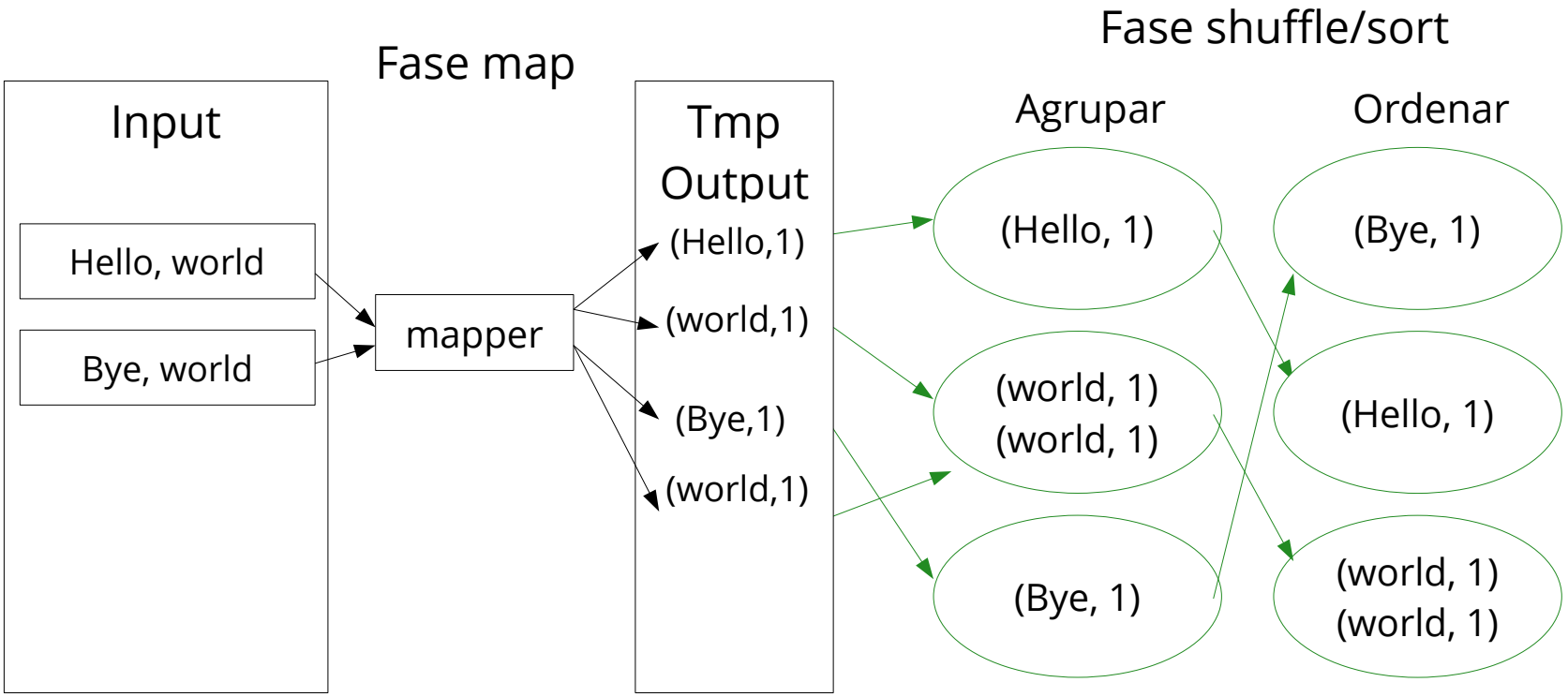
- Los resultados de map, list((k2,v2)), se agrupan (por clave) y ordenan (por clave), dando lugar a (k2, list(v2))



# MapReduce

## Implementación > 2. Fase shuffle/sort

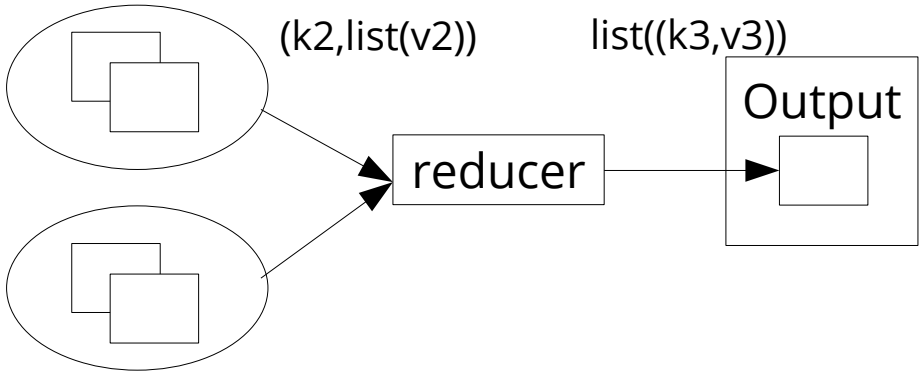
- Ejemplo: contar palabras



# MapReduce

## Implementación > 3. Fase reduce

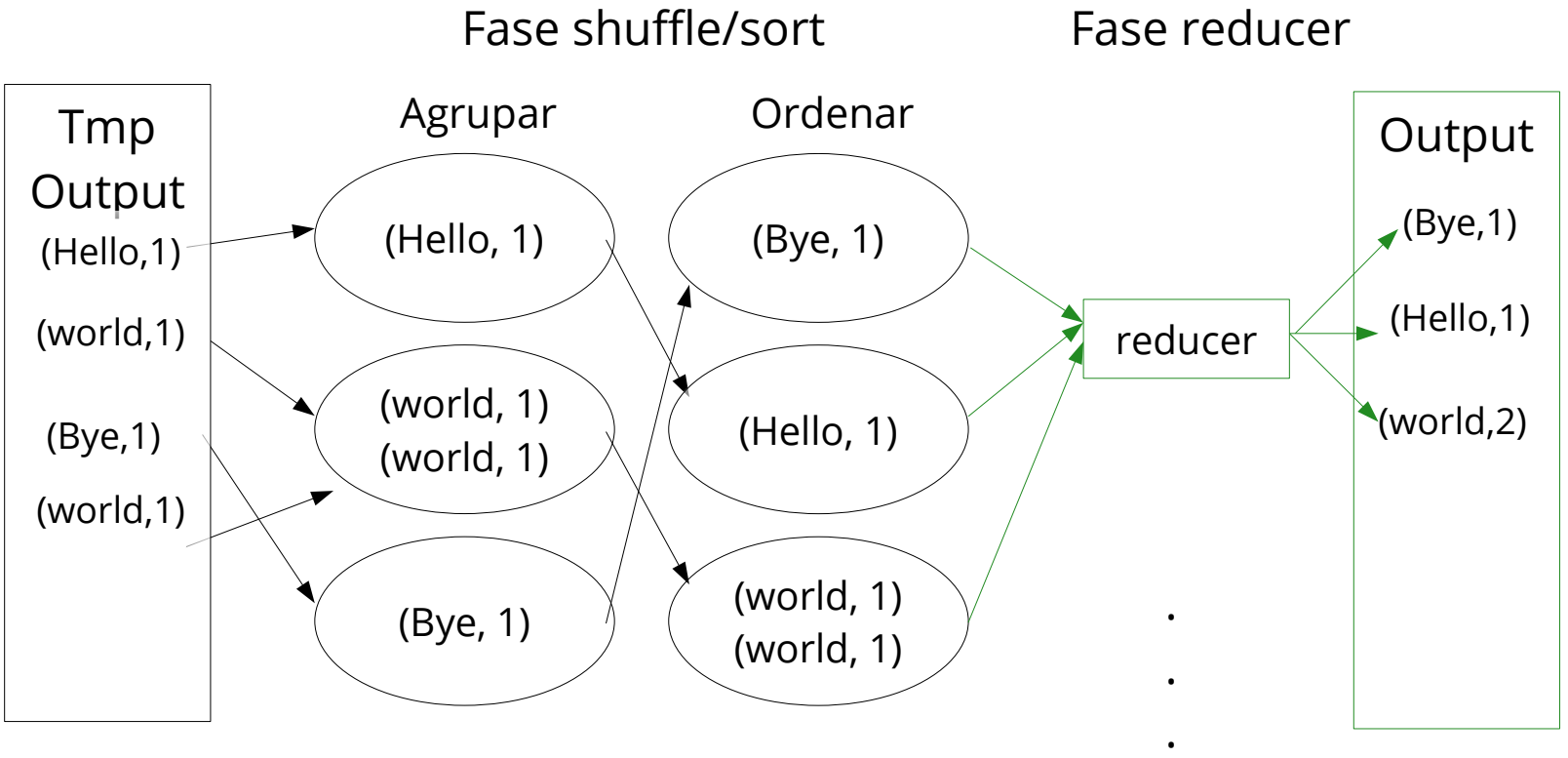
- Una función, el reducer, recibe la clave y valores asociados a dicha clave, (k2, list(v2)), en orden
- Y produce cero o más pares (k3,v3), es decir, list((k3,v3))



# MapReduce

## Implementación > 3. Fase reduce

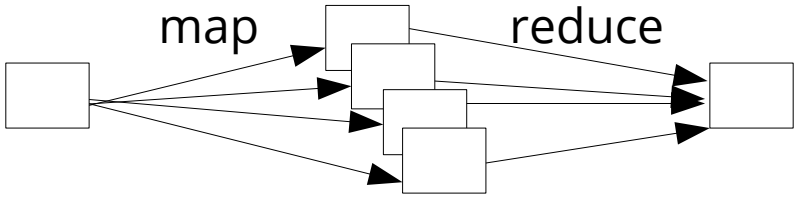
- Ejemplo: contar palabras

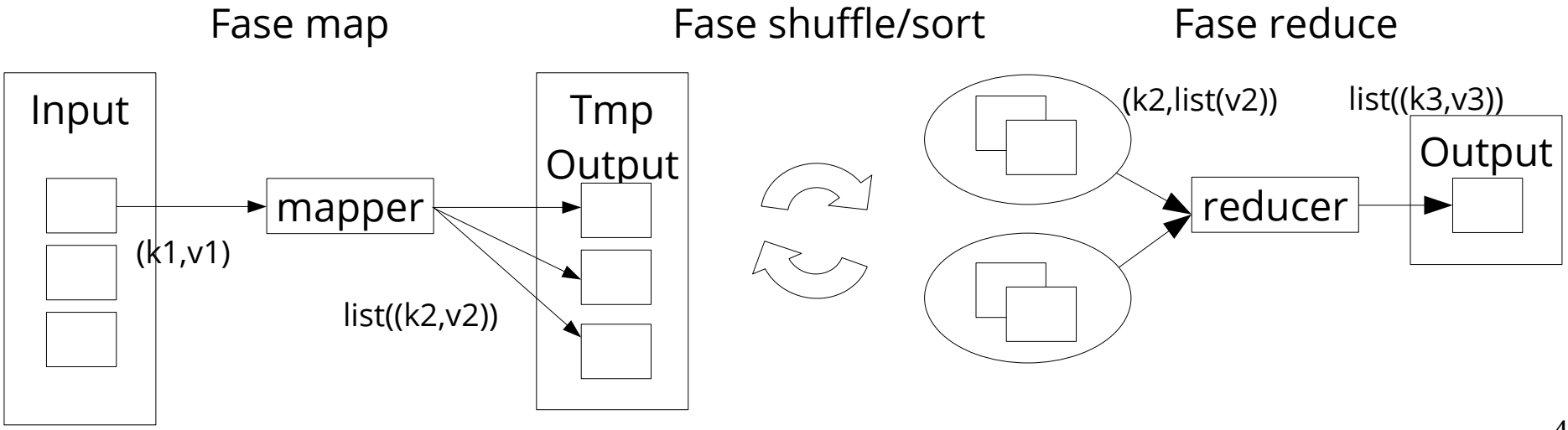


# MapReduce

## Implementación > Las tres fases

- Map abre los resultados, reduce los recoge



# MapReduce

## Diseñar mapper y reducer > Ejemplo I

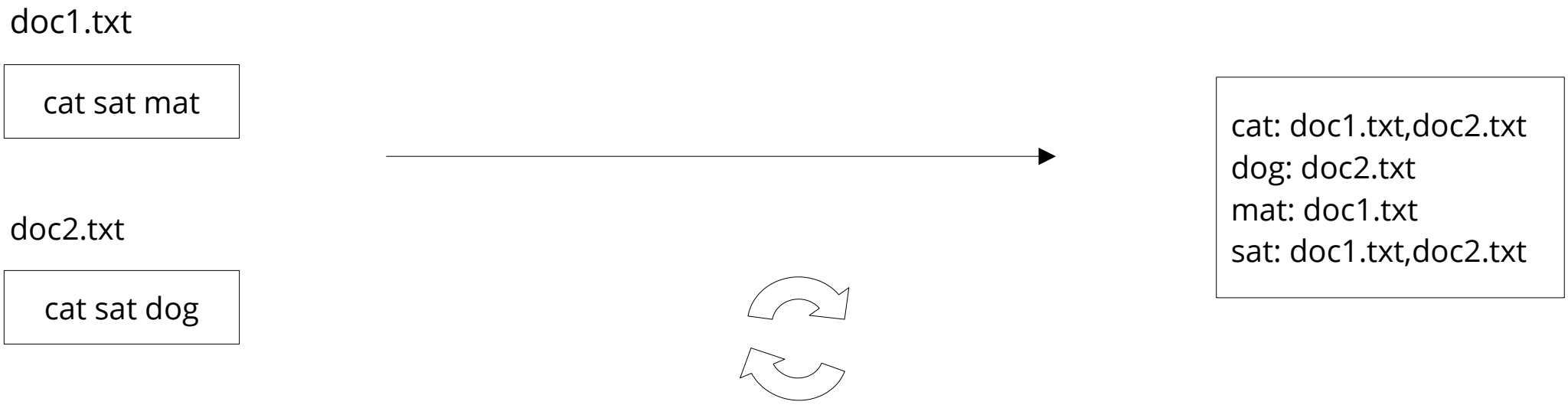
- El contador de palabras en pseudocódigo

```
def mapper(document):
  words = tokenize(document)
  for word in words:
    emit(word, 1)

def reducer(word, values):
  sum = 0
  for value in values: sum += value
  emit(word, sum)
```

# MapReduce

## Diseñar mapper y reducer > Ejemplo II

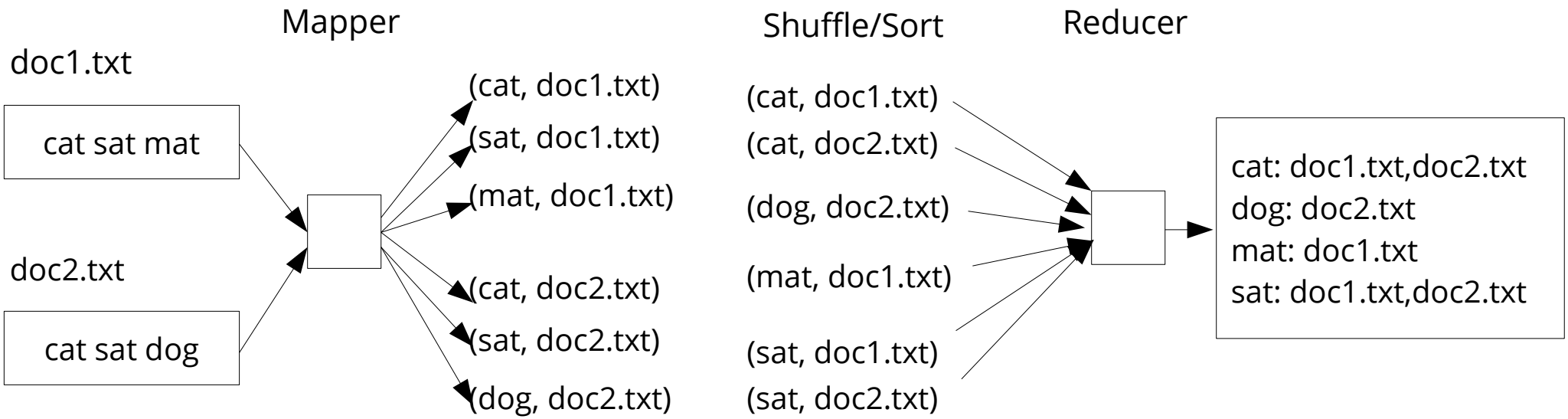
- Indexación de documentos



# MapReduce

## Diseñar mapper y reducer > Ejemplo II

- Indexación de documentos



# MapReduce

## Diseñar mapper y reducer > Ejemplo II

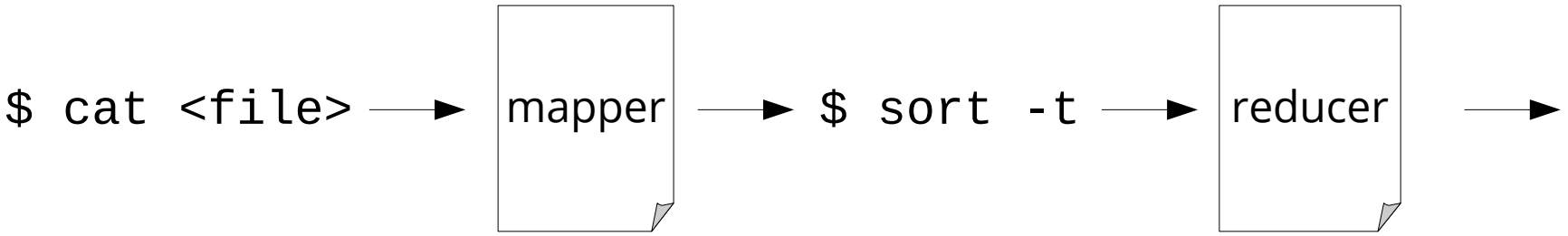
- Indexación de documentos en pseudocódigo

```
def mapper(document):
  words = tokenize(document)
  for word in words:
    emit(word, document.name)

def reducer(word, values):
  emit(word, str(values))
```

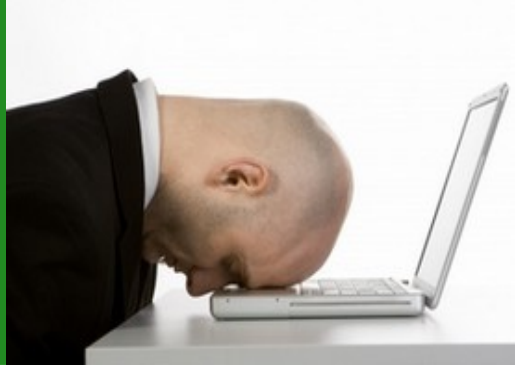
# MapReduce

## Simulación en una máquina

- Es sencillo simular un programa MapReduce con Unix en una única máquina
- Redirecciones de entrada/salida y comando sort
- Mapper y reducer leen/escriben en stdin/stdout

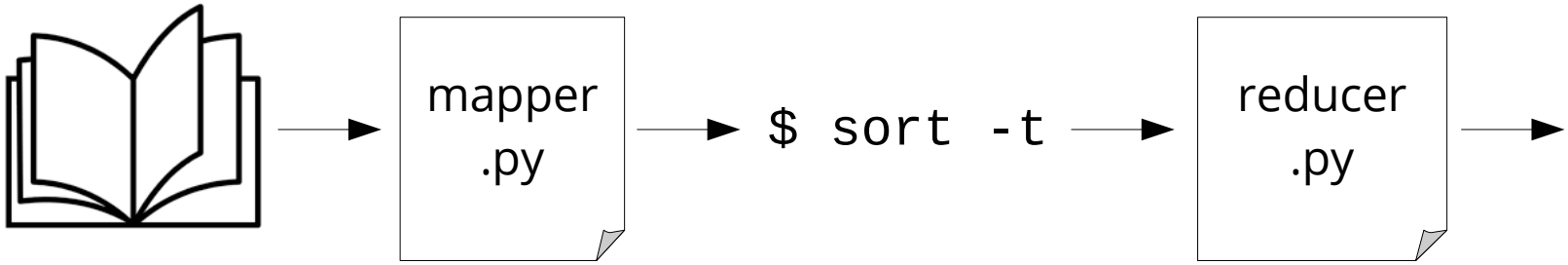


# MapReduce

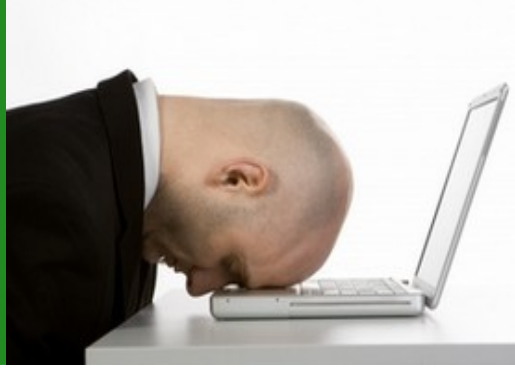


## Ejercicio 1

- Contador de palabras con Python y Unix



# MapReduce

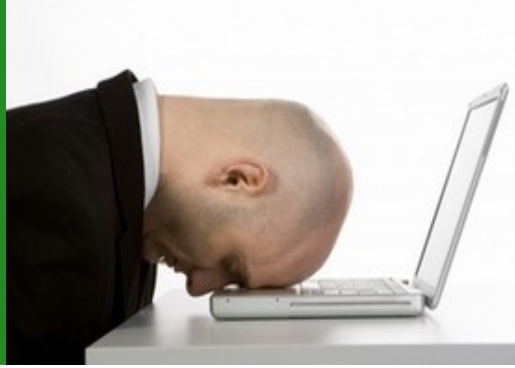


## Ejercicio 1

- Contador de palabras con Python y Unix

```
#!/bin/python3
import sys
def map(input):
  for line in input:
    words = line.split()
    for word in words:
      print(f"{word} 1")

map(sys.stdin)
```

# MapReduce

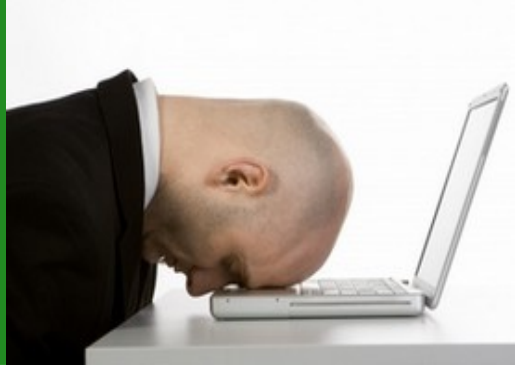


## Ejercicio 1

- Contador de palabras con Python y Unix

```
#!/bin/python3
import sys
def reduce(input):
  curr_word = None; curr_count = 0
  for line in input:
    word, count = line.split()
    if word == curr_word: curr_count += int(count)
    else:
      if curr_word: print(f"{curr_word} {curr_count}")
      curr_word = word; curr_count = int(count)
  if curr_word == word: print(f"{curr_word} {curr_count}")

reduce(sys.stdin)
```

# MapReduce



## Ejercicio 1

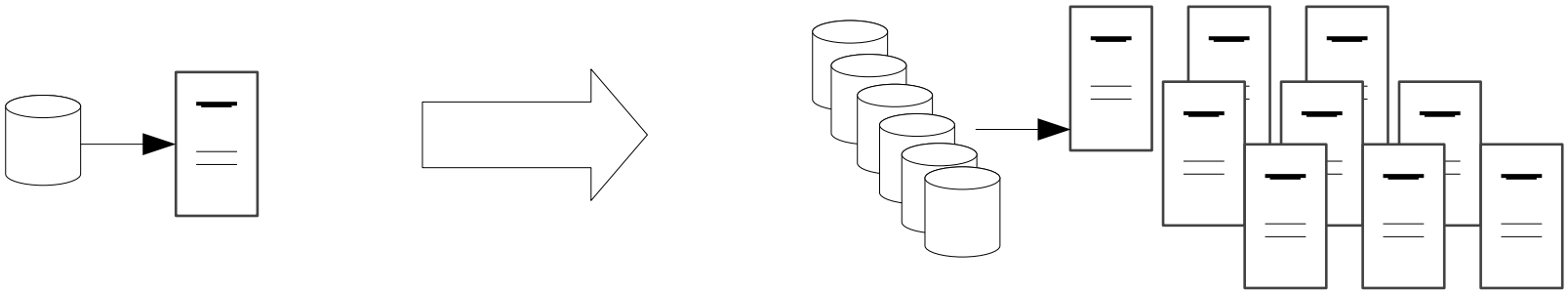
- Contador de palabras con Python y Unix

```
$ cat el_quijote.txt | python3 map.py | sort -t 1 |
python3 reduce.py
```

# MapReduce

## Escalar MapReduce

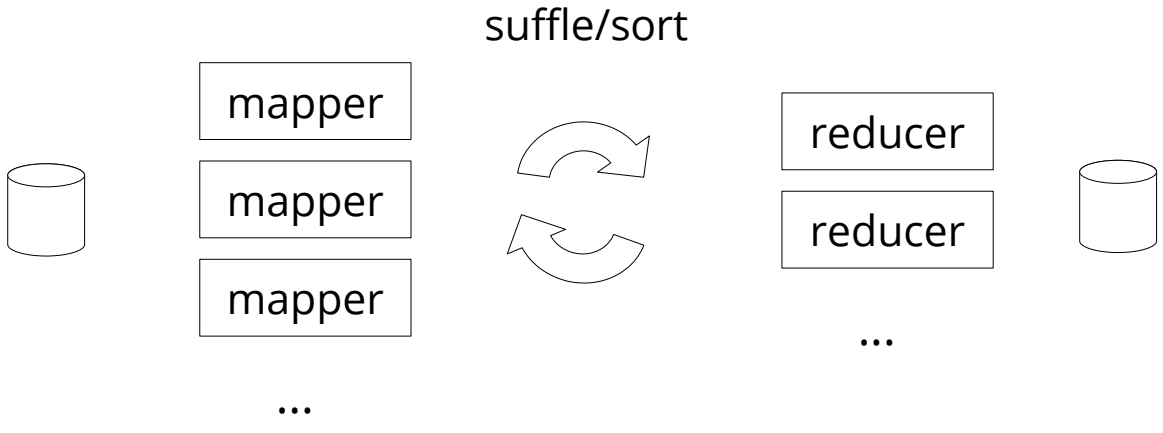
- MapReduce está diseñado para ser escalable
- Si se incrementa la infraestructura se obtienen mejores resultados



# MapReduce

## Escalar MapReduce

- MapReduce está diseñado para ser escalable
- Si se incrementa la infraestructura se obtienen mejores resultados
- Múltiples mappers, múltiples reducers, en paralelo



# MapReduce

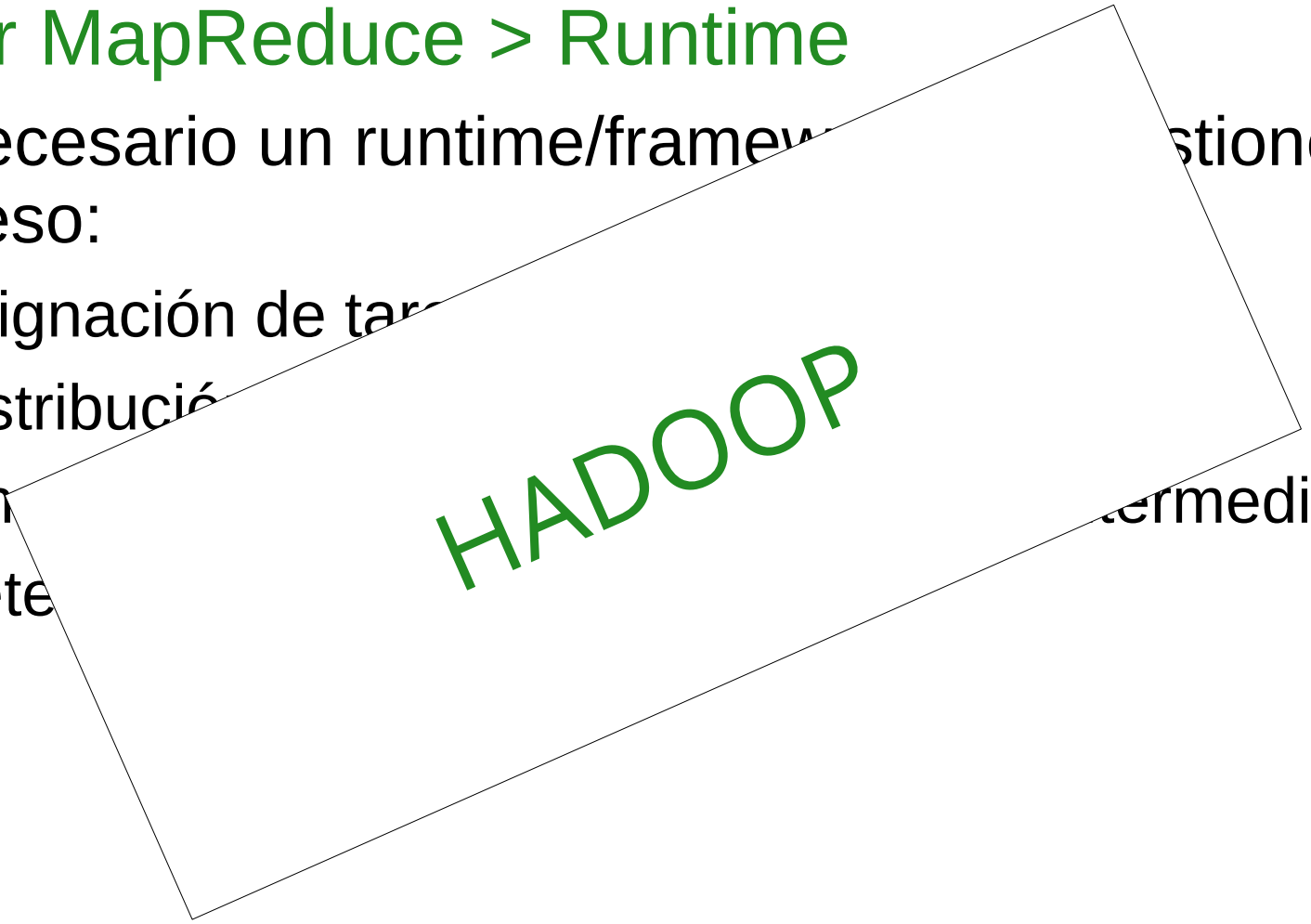
## Escalar MapReduce > Runtime

- Es necesario un runtime/framework que gestione todo el proceso:
  - Asignación de tareas map/reduce a nodos
  - Distribución de datos: mover tareas a los datos
  - Sincronización: agrupar, ordenar datos intermedios
  - Detección de errores y reinicios

# MapReduce

## Escalar MapReduce > Runtime

- Es necesario un runtime/framew… …stione todo el proceso:
  - Asignación de tar…
  - Distribuci…
  - Sin… …termedios
  - Dete…



# Hadoop

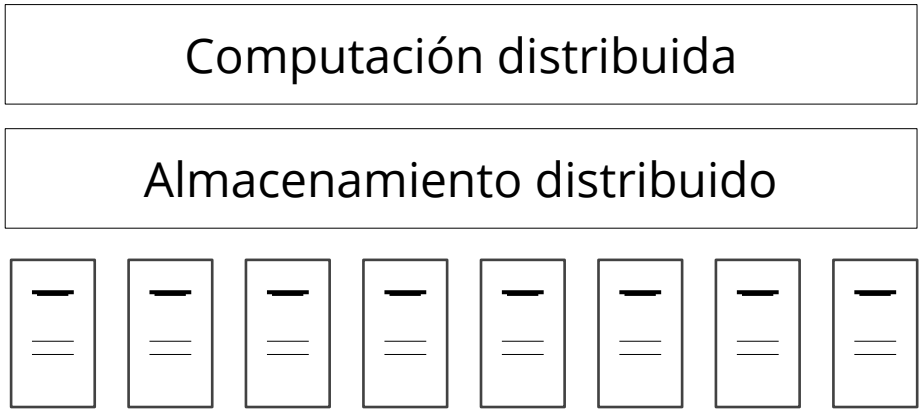
## ¿Qué es?

- https://hadoop.apache.org/
- Plataforma de computación open source distribuida, fiable y escalable
- Originalmente concebida para ejecutar trabajos batch (MapReduce) en clústers con miles de máquinas
- Gestiona volúmenes de PetaBytes
- Utilizado por Yahoo!, Facebook, Twitter, ...

# Hadoop

## ¿Qué es?

- En la actualidad: sistema ficheros distribuido + plataforma de computación distribuida con localidad de datos

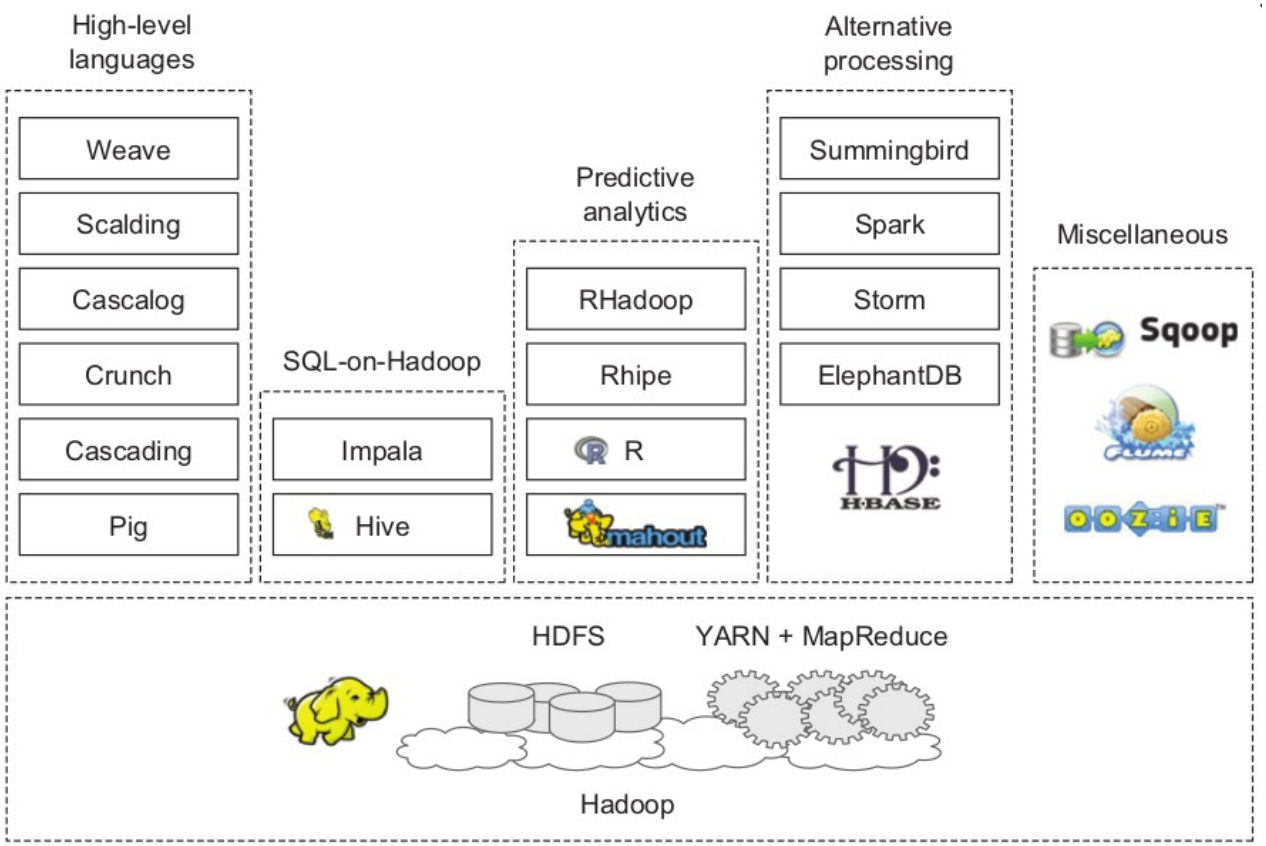



clúster

# Hadoop

## Ecosistema

- Infinidad de herramientas basadas en Hadoop




[Hadoop in practice. 2nd Ed. Alex Holmes]

# Hadoop

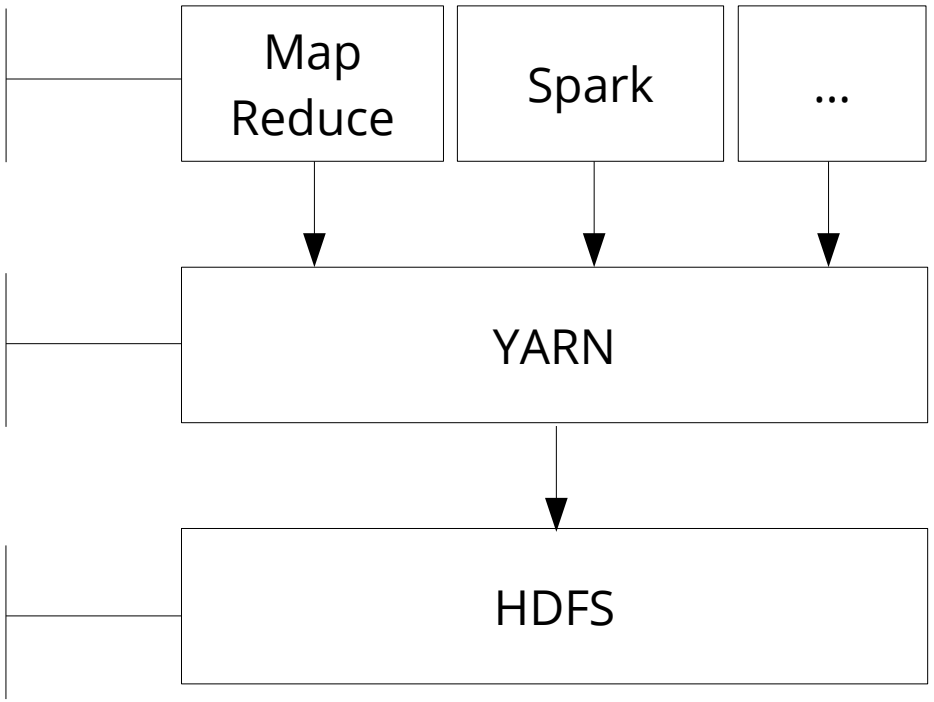
## Arquitectura

- HDFS + YARN + MapReduce/XXX

- MapReduce: modelo de procesamiento en batch
- Otros tipos de carga también

- Gestor de recursos del clúster
- Permite ejecutar cualquier programa distribuido

- Sistema de ficheros distribuido
- Eficiente, escalable, replicado



# Hadoop

## Instalación

- Requisitos Hadoop 3.3.x

```
> sudo apt install openjdk-8-jdk ssh pdsh
```

- Descargar hadoop-x.x.x.tar.gz y fijar variables en /etc/hadoop/hadoop-env.sh

```
> tar xfz hadoop-x.x.x.tar.gz
> export JAVA_HOME=/usr/java/latest (ubicación JDK)
> export HADOOP_HOME=/home/alumno/hadoop-x.x.x
> export PDSH_RCMD_TYPE=ssh
```

- Ejecutar hadoop

```
> bin/hadoop
```

# Hadoop

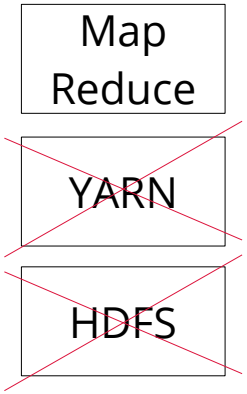
## Ejecución

- Tres modos
  - Modo local
  - Modo pseudo-distribuido
  - Modo distribuido
- + Modo docker :-)!

# Hadoop

## Ejecución > Modo local

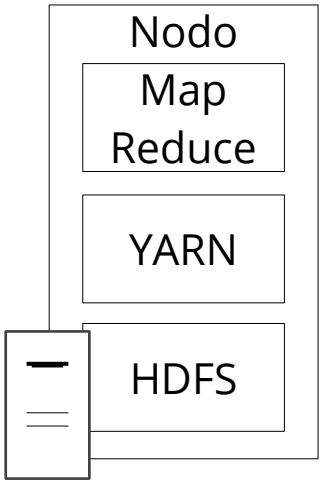
- Standalone: modo por defecto
- Hadoop se ejecuta en un único proceso Java
- Útil para depuración
- Se utiliza el sistema de ficheros local



# Hadoop

## Ejecución > Modo pseudo-distribuido

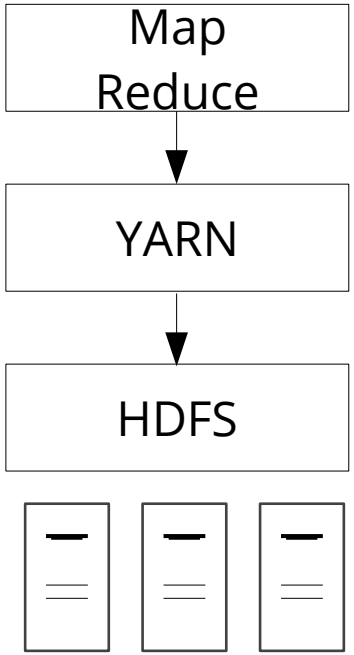
- Hadoop se ejecuta en una única máquina
- Cada demonio se ejecuta en su proceso Java
- Se pueden probar todos los componentes
- Se simula un clúster en pequeña escala



# Hadoop

## Ejecución > Modo distribuido

- Hadoop se ejecuta en un clúster de nodos
- Configuración de Hadoop en producción

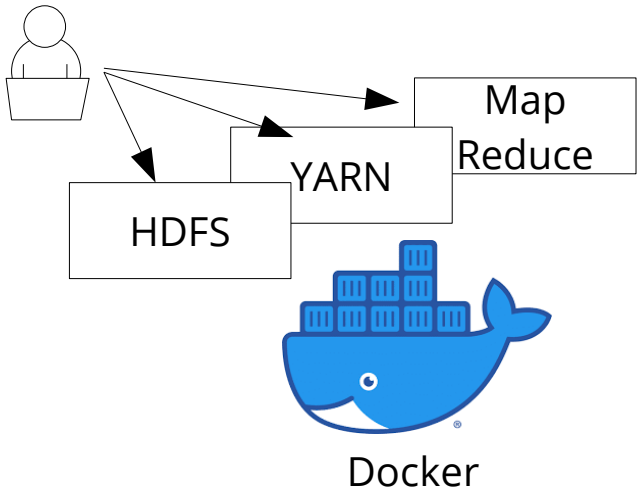


# Hadoop

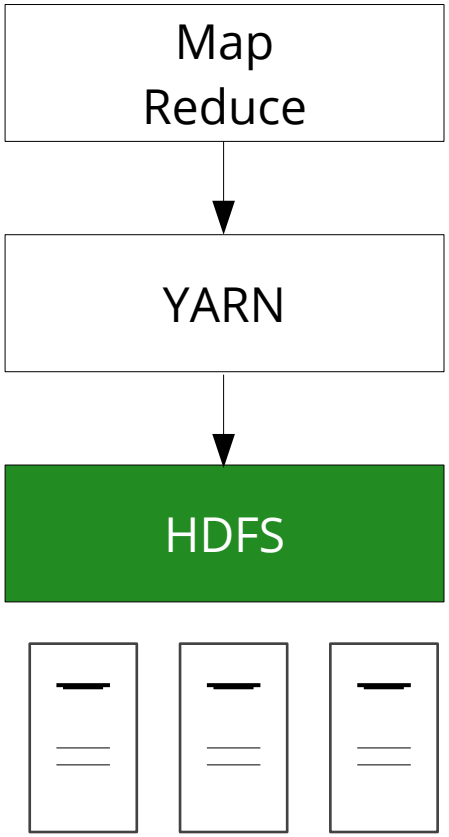
## Ejecución > Modo Docker

- Arranca todo el sistema en modo pseudo-distribuido en Docker, con un único comando
- Hay varias imágenes Docker disponibles; nos basaremos en la imagen oficial apache/hadoop
- Dos ficheros: docker-compose.yaml y config.env

```
> docker compose up -d
```



# Hadoop > HDFS



# Hadoop > HDFS

## ¿Qué es?

- Sistema de ficheros distribuido usado en Hadoop
- Implementación open-source Google File System
- Eficiente, escalable, altamente disponible
- Se ejecuta en hardware básico
  - Alta probabilidad de fallos
- Optimizado para leer/escribir ficheros grandes (TB)
  - Pocos ficheros, pero muy grandes
  - Escribir una vez, leer muchas veces

# Hadoop > HDFS

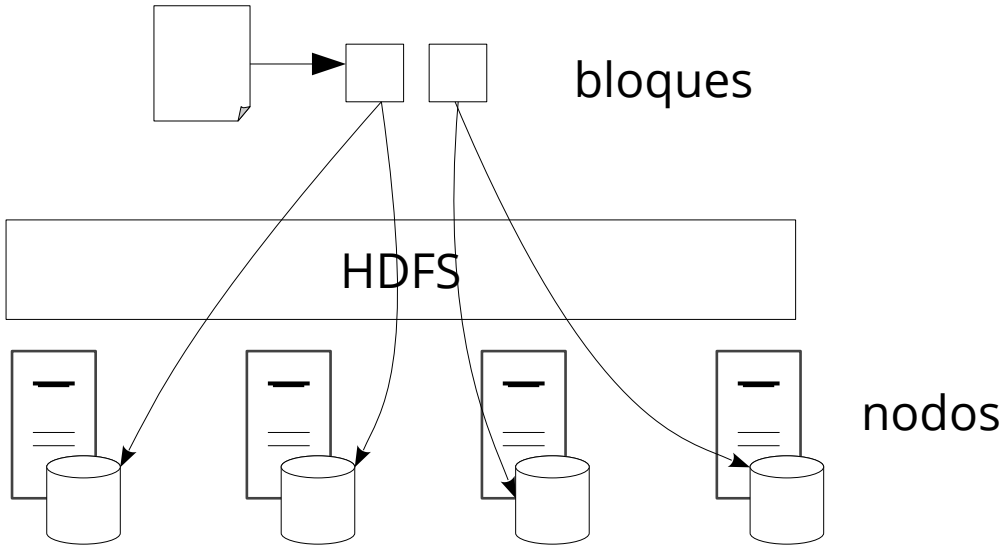
## ¿Qué es?

- No está pensado para:
  - Acceder a datos con baja latencia
  - Muchos ficheros pequeños
  - Múltiples escritores
  - Modificaciones arbitrarias

# Hadoop > HDFS

## Bloques

- Los ficheros se dividen en bloques grandes (+128MB)
- HDFS distribuye y replica los bloques en distintos nodos



# Hadoop > HDFS

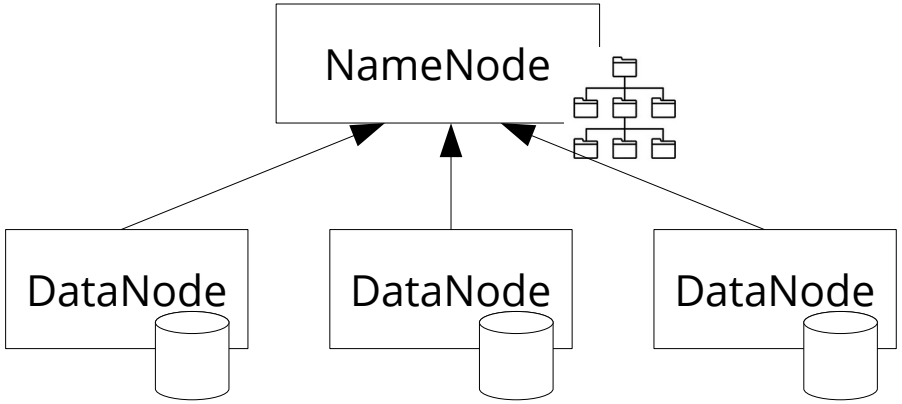
## Bloques

- Bloques grandes = minimizar búsquedas en disco
- El factor de réplica es configurable (por fichero!) (suele ser 3)
- Si una réplica falla
  - Se lee de otra réplica de manera transparente
  - Se vuelve a replicar en otro nodo

# Hadoop > HDFS

## Arquitectura

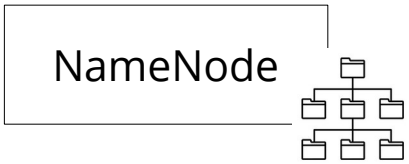
- Maestro/esclavo: NameNode vs DataNode



# Hadoop > HDFS

## Arquitectura

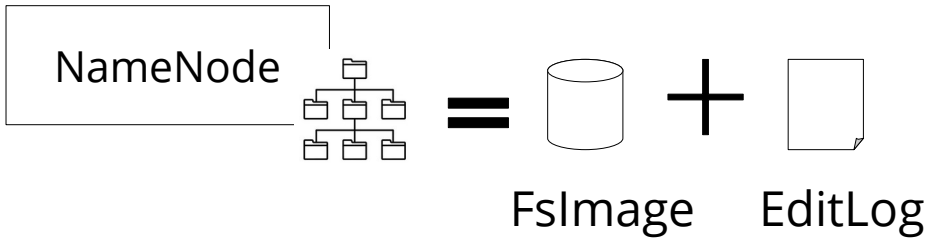
- NameNode
  - Gestiona el sistema de ficheros
  - Mantiene un mapa de toda la info en memoria



# Hadoop > HDFS

## Arquitectura

- NameNode
  - Gestiona el sistema de ficheros
  - Mantiene un mapa de toda la info en memoria
  - Además, registra toda la info en disco: FsImage + EditLog



# Hadoop > HDFS

## Arquitectura

- NameNode
  - Gestiona el sistema de ficheros
  - Mantiene un mapa de toda la info en memoria
  - Además, registra toda la info en disco: FsImage + EditLog
  - Al arrancar, carga en memoria FsImage y aplica EditLog
  - Periódicamente efectúa checkpoints de la imagen en memoria en disco y elimina EditLog

# Hadoop > HDFS

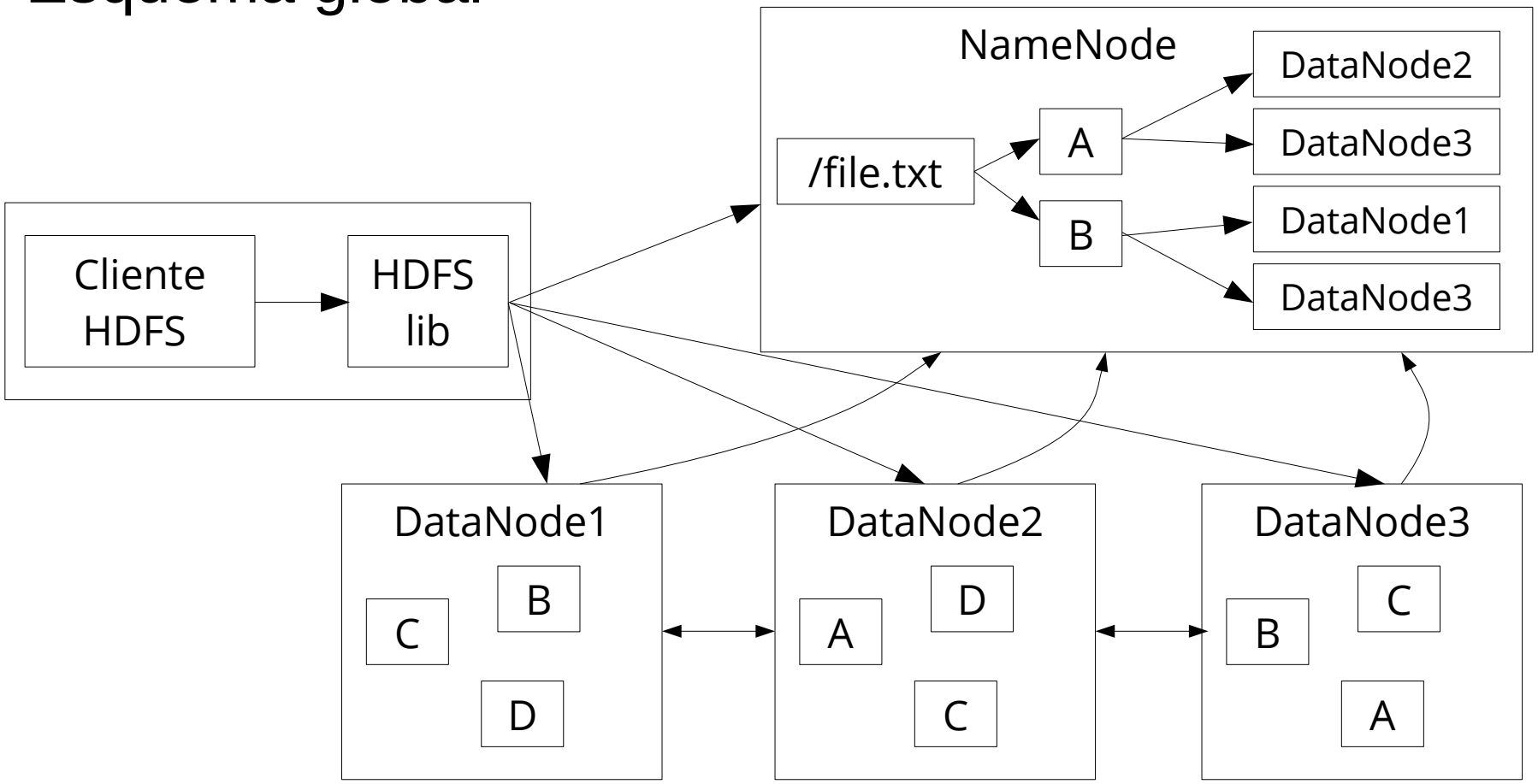
## Arquitectura

- DataNode
  - Almacena los bloques como ficheros independientes en el disco local
  - Crean, eliminan y replican bloques solicitados por NameNode
  - Leen/escriben los bloques solicitados por el cliente
  - Periódicamente envían heartbeat y reporte de bloques (*BlockReport*) al NameNode

# Hadoop > HDFS

## Arquitectura

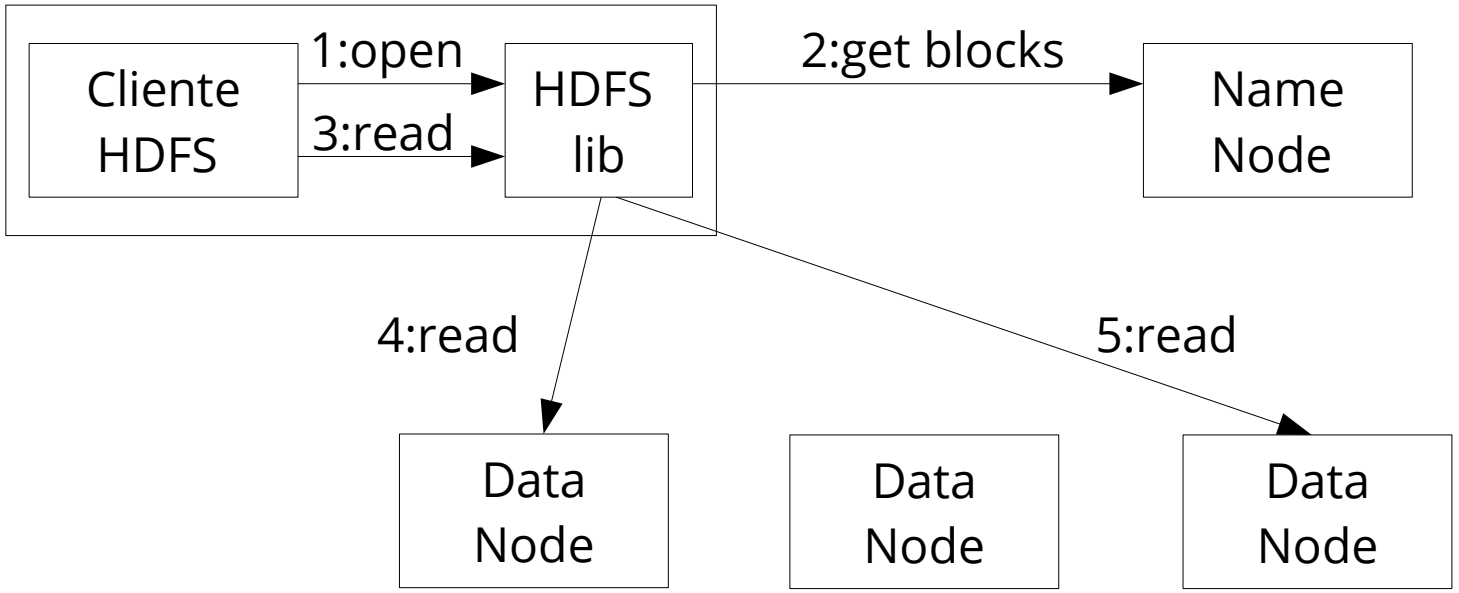
- Esquema global



# Hadoop > HDFS

## Funcionamiento

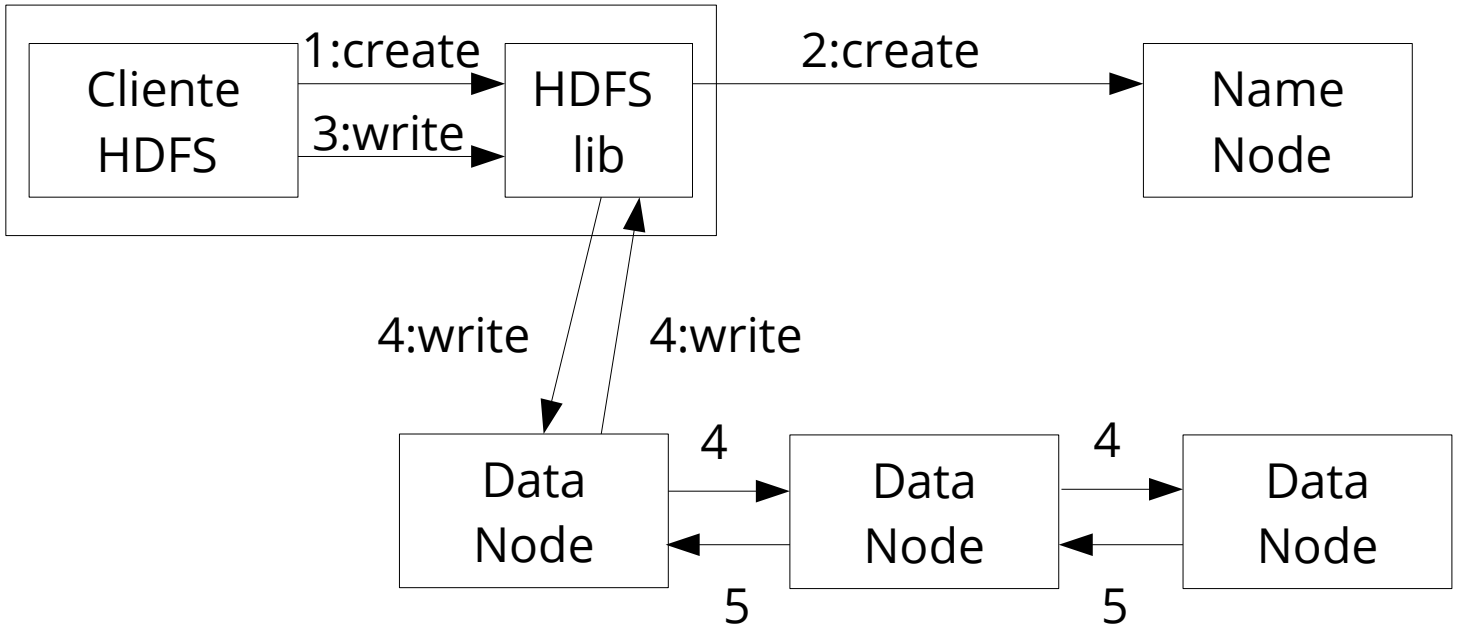
- Lectura de fichero



# Hadoop > HDFS

## Funcionamiento

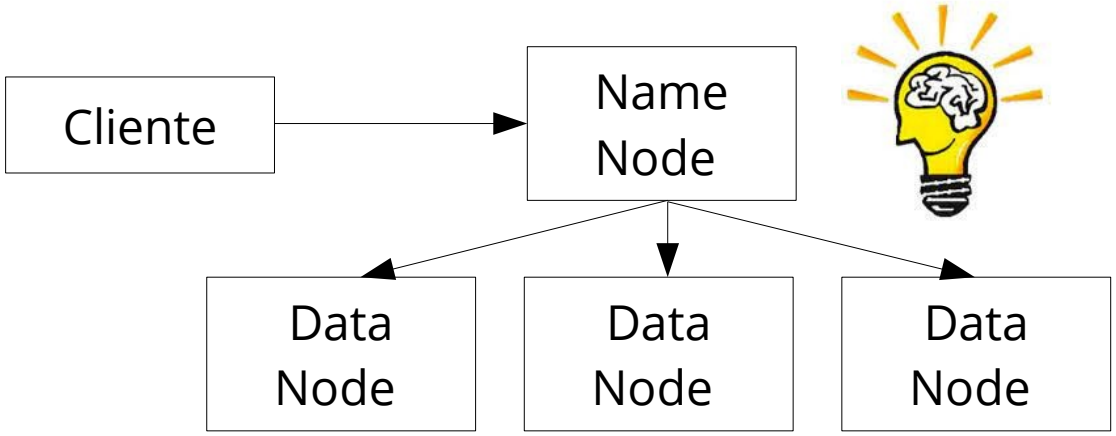
- Escritura en fichero



# Hadoop > HDFS

## Replicación

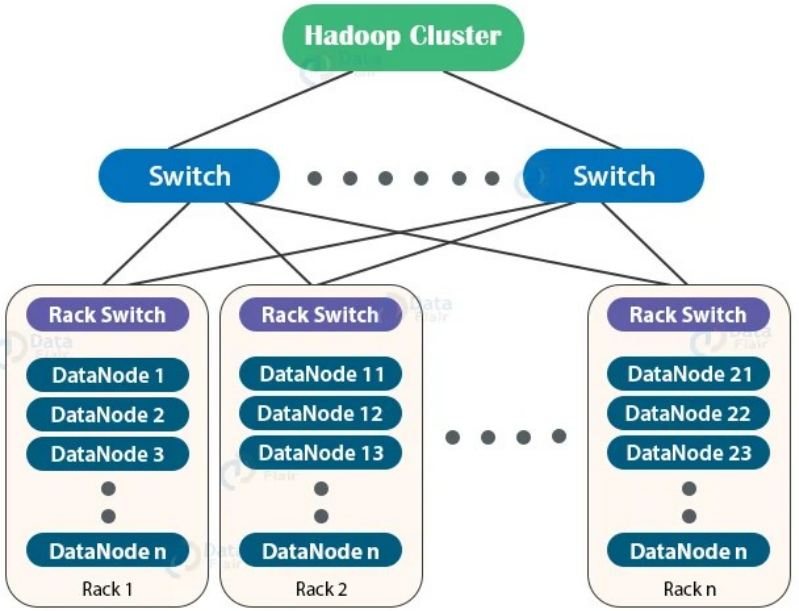
- HDFS distribuye los bloques “inteligentemente”, para asegurar mayor fiabilidad y rendimiento



# Hadoop > HDFS

## Replicación

- HDFS distribuye los bloques “inteligentemente”, para asegurar mayor fiabilidad y rendimiento
- Premisas: nodos en racks, ancho de banda en el mismo rack es mayor



# Hadoop > HDFS

## Replicación

- HDFS distribuye los bloques “inteligentemente”, para asegurar mayor fiabilidad y rendimiento
- Premisas: nodos en racks, ancho de banda en el mismo rack es mayor
- Distintas estrategias, por ejemplo:
  - 3 bloques en 3 racks distintos
  - Buena disponibilidad y tasa de lectura
  - Mala tasa de escritura (3 racks!)

# Hadoop > HDFS

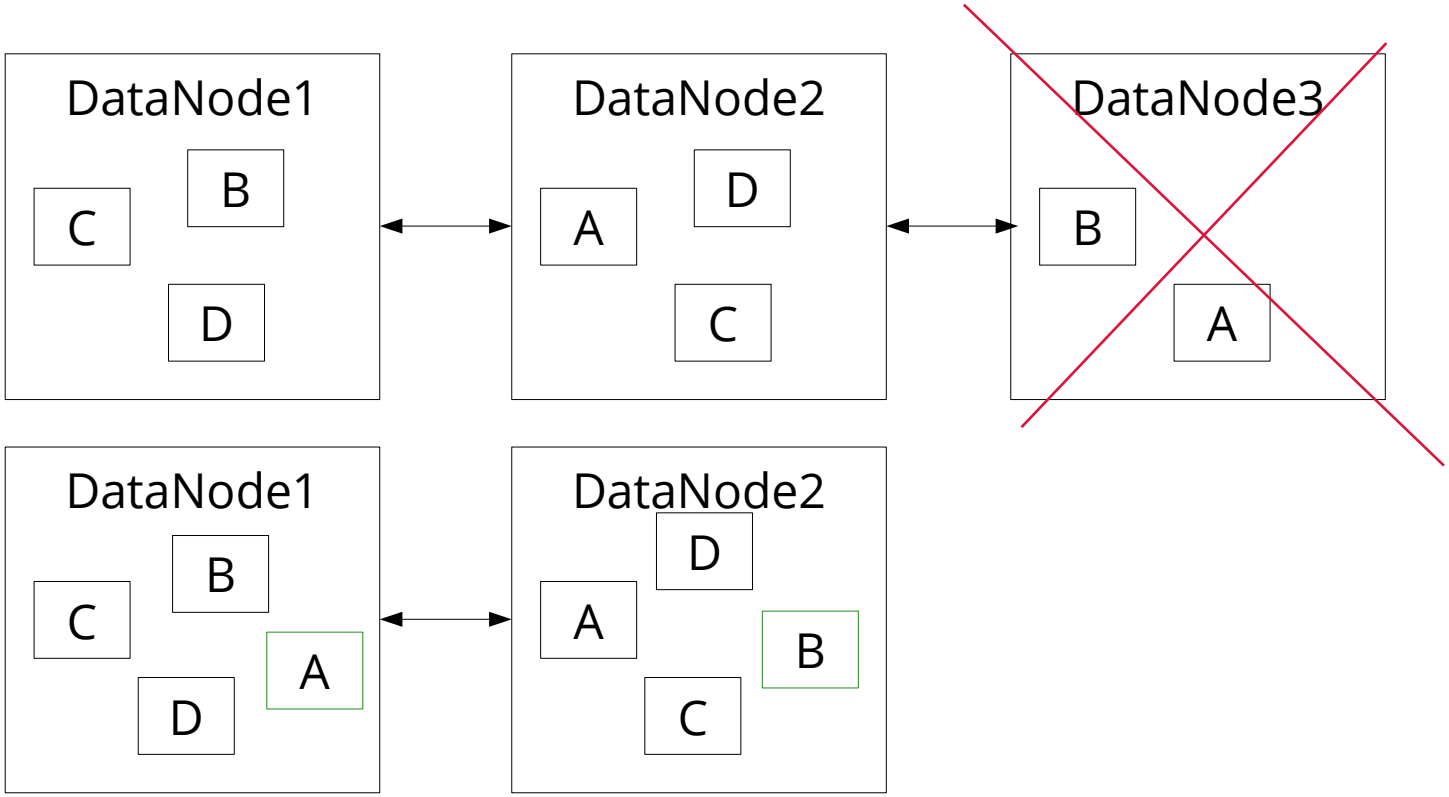
## Replicación

- HDFS distribuye los bloques “inteligentemente”, para asegurar mayor fiabilidad y rendimiento
- Premisas: nodos en racks, ancho de banda en el mismo rack es mayor
- Distintas estrategias, por ejemplo:
  - 1$^{er}$ bloque se escribe en nodo local (o arbitrario), 2$^{o}$ bloque en otro nodo en el mismo rack, 3$^{er}$ bloque en otro rack
  - Buena disponibilidad, tasa lectura y escritura

# Hadoop > HDFS

## HA (High Availability)

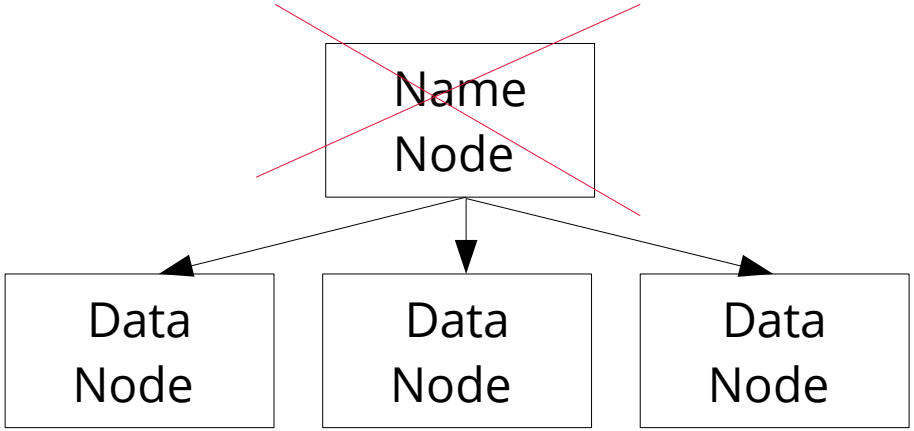
- Si falla un DataNode, los bloques que almacena vuelven a replicarse entre el resto de DataNodes



# Hadoop > HDFS

## HA (High Availability)

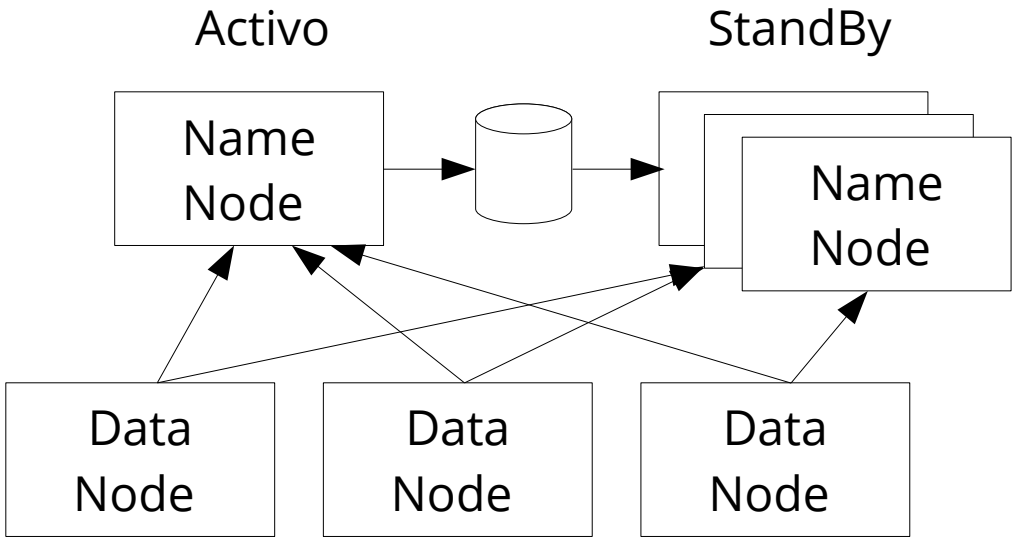
- ¿Y si falla el NameNode?



# Hadoop > HDFS

## HA (High Availability)

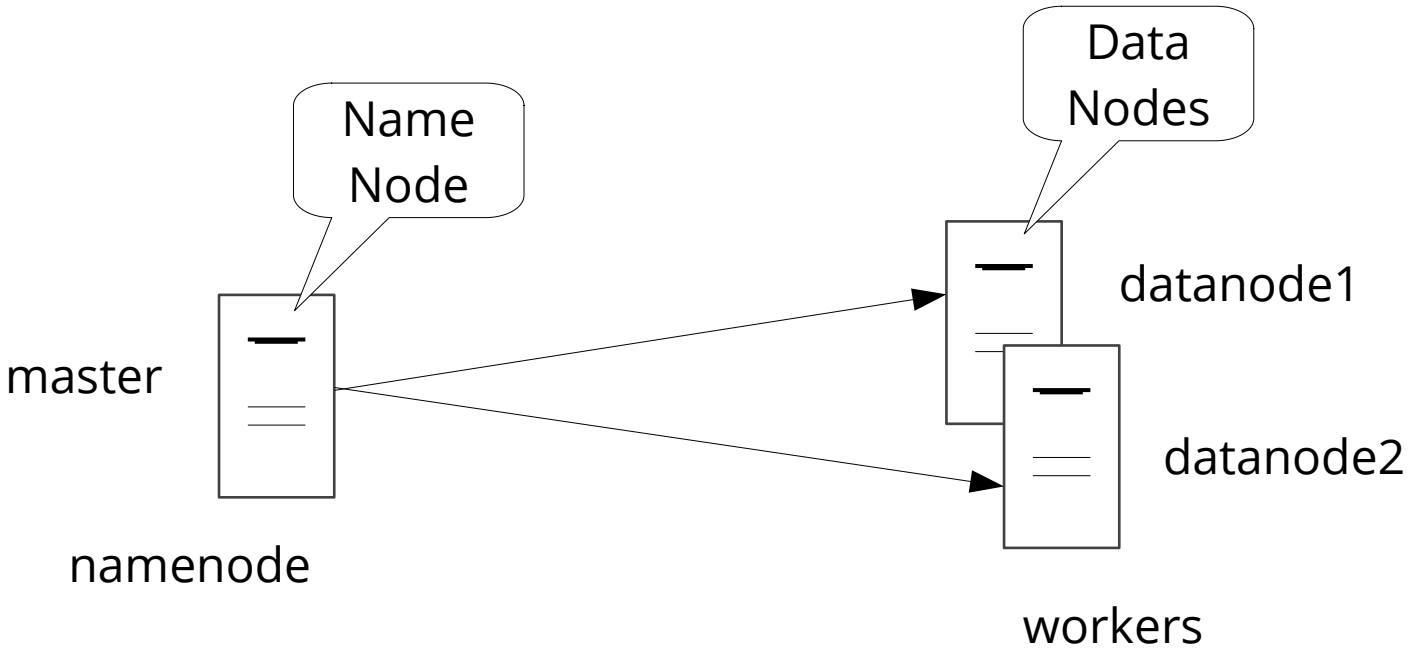
- ¿Y si falla el NameNode?
- Es necesario replicarlo: múltiples NameNodes que comparten el EditLog (NFS vs Journal)
- Los DataNodes remiten informes de estado a todos



# Hadoop > HDFS

## Puesta en marcha en Docker

- Definición del clúster



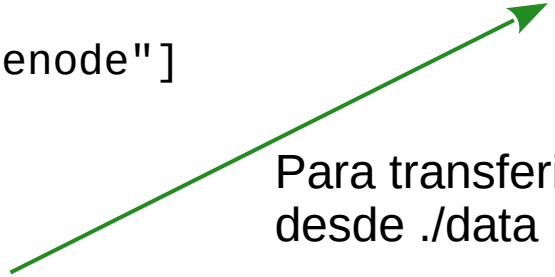
# Hadoop > HDFS

## Puesta en marcha en Docker

- Usando la imagen oficial [apache/hadoop]
- docker-compose.yaml

```
version: "2"
services:
   namenode:
      image: apache/hadoop:3
      hostname: namenode
      command: ["hdfs", "namenode"]
      ports:
        - 9870:9870
      env_file:
        - ./config.env
      environment:
          ENSURE_NAMENODE_DIR: "/tmp/hadoop-root/dfs/name"
   datanode:
      image: apache/hadoop:3
      command: ["hdfs", "datanode"]
      env_file:
        - ./config.env
```

```
volumes:
  - ./data:/data
```



Para transferir ficheros desde ./data

Puede replicarse

# Hadoop > HDFS

## Puesta en marcha en Docker

- Usando la imagen oficial apache/hadoop
- config.env

```
HADOOP_HOME=/opt/hadoop
CORE-SITE.XML_fs.default.name=hdfs://namenode
CORE-SITE.XML_fs.defaultFS=hdfs://namenode
HDFS-SITE.XML_dfs.namenode.rpc-address=namenode:8020
HDFS-SITE.XML_dfs.replication=1
```

```
> docker compose up
```

- El NameNode publica una GUI web en http://localhost:9870/
- Iniciar sesión en namenode:

```
> docker compose exec namenode /bin/bash
```

# Hadoop > HDFS

## Hadoop File System Shell

- Para interaccionar con HDFS en línea de comandos

```
> bin/hadoop fs <args>
```

- Existen multitud de comandos familiares
- cat, chmod, chown, cp, df, du, find,  head, ls, mkdir, mv, rm, rmdir, tail, touch, concat, ...

```
> bin/hadoop fs -mkdir -p /user/alumno
> bin/hadoop fs -touch /user/alumno/test.txt
> bin/hadoop fs -ls /user/alumno
> bin/hadoop fs -cp /user/alumno/test.txt /test2.txt
```

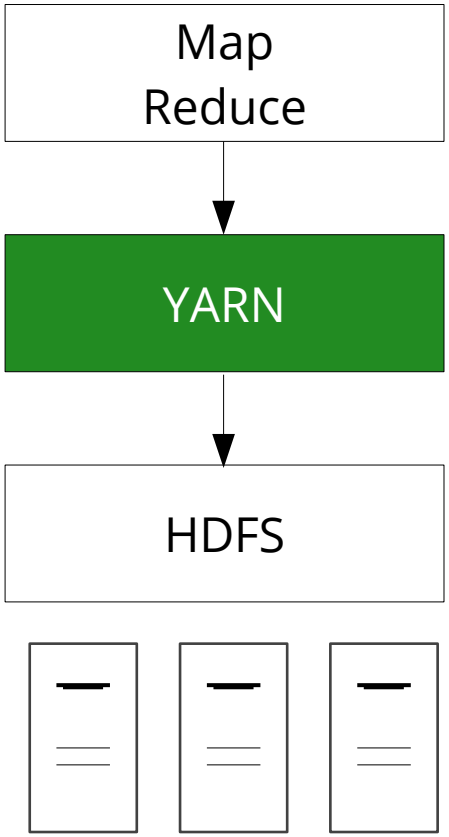
# Hadoop > HDFS

## Hadoop File System Shell

- Es habitual mover/copiar ficheros del sistema de ficheros local a HDFS y viceversa: appendToFile, copyFromLocal, copyToLocal, get, moveFromLocal, moveToLocal, put

```
> bin/hadoop fs -copyFromLocal test.txt /user/alumno
> bin/hadoop fs -copyToLocal /test2.txt test2.txt
> bin/hadoop fs -get /test2.txt test2.txt
> bin/hadoop fs -put test.txt /user/alumno
> echo "hello world!" | bin/hadoop fs -put - /test3.txt
> bin/hadoop fs -cat /test3.txt
```

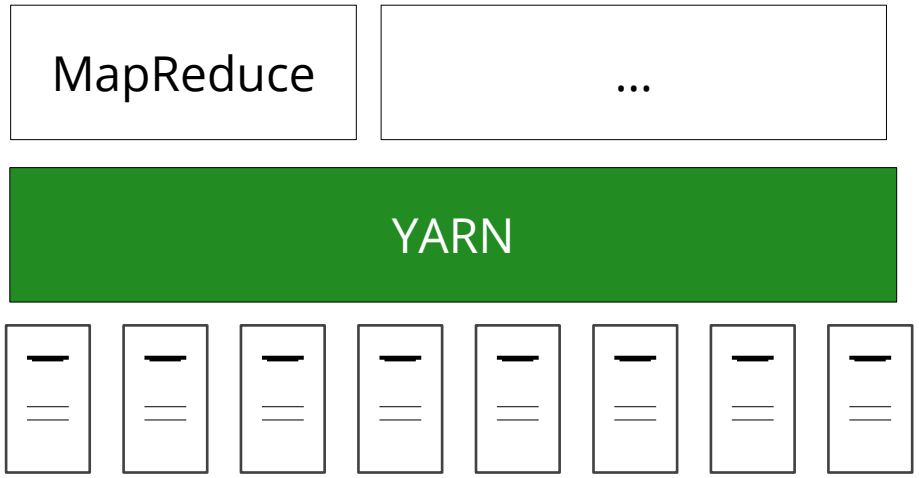
# Hadoop > YARN



# Hadoop > YARN

## ¿Qué es?

- Yet Another Resource Negotiator (Hadoop 2+)
- Sistema de gestión de recursos de clúster Hadoop
- Permite la ejecución de cualquier programa distribuido (MapReduce, machine learning, grafos, etc.)



# Hadoop > YARN

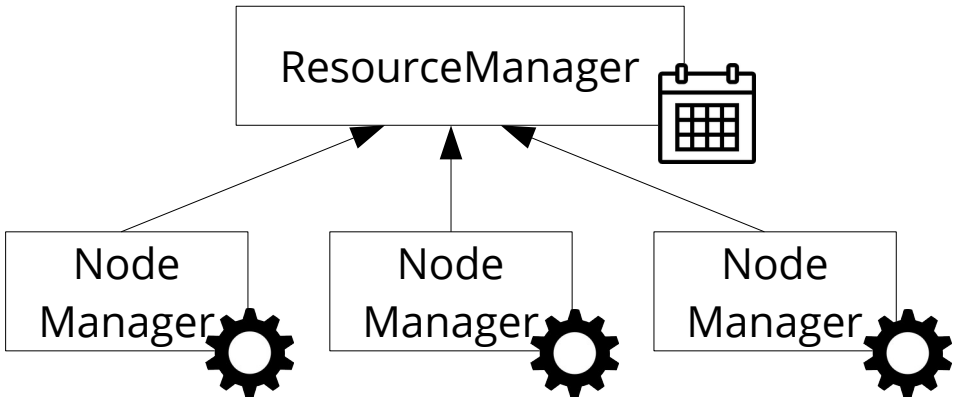
## Contenedores

- YARN gestiona los recursos del clúster a través de contenedores
- Son procesos con un contrato que determina los recursos físicos que está permitido usar
- YARN crea contenedores bajo demanda, planifica dónde se ejecutan y los monitoriza
- Si el contenedor utiliza más recursos, o YARN necesita liberar recursos, el contenedor puede ser finalizado

# Hadoop > YARN

## Arquitectura

- Maestro/esclavo: ResourceManager vs NodeManager



# Hadoop > YARN

## Arquitectura

- ResourceManager
  - Planifica los recursos del clúster
  - Recibe peticiones de creación de contenedores y decide dónde se ejecutan
  - Distintas políticas de planificación (plugins):
    - CapacityScheduler: se fracciona la capacidad del clúster en múltiples colas; no se puede exceder la capacidad de una cola
    - FairScheduler: los recursos se reparten equitativamente entre todas las aplicaciones

# Hadoop > YARN

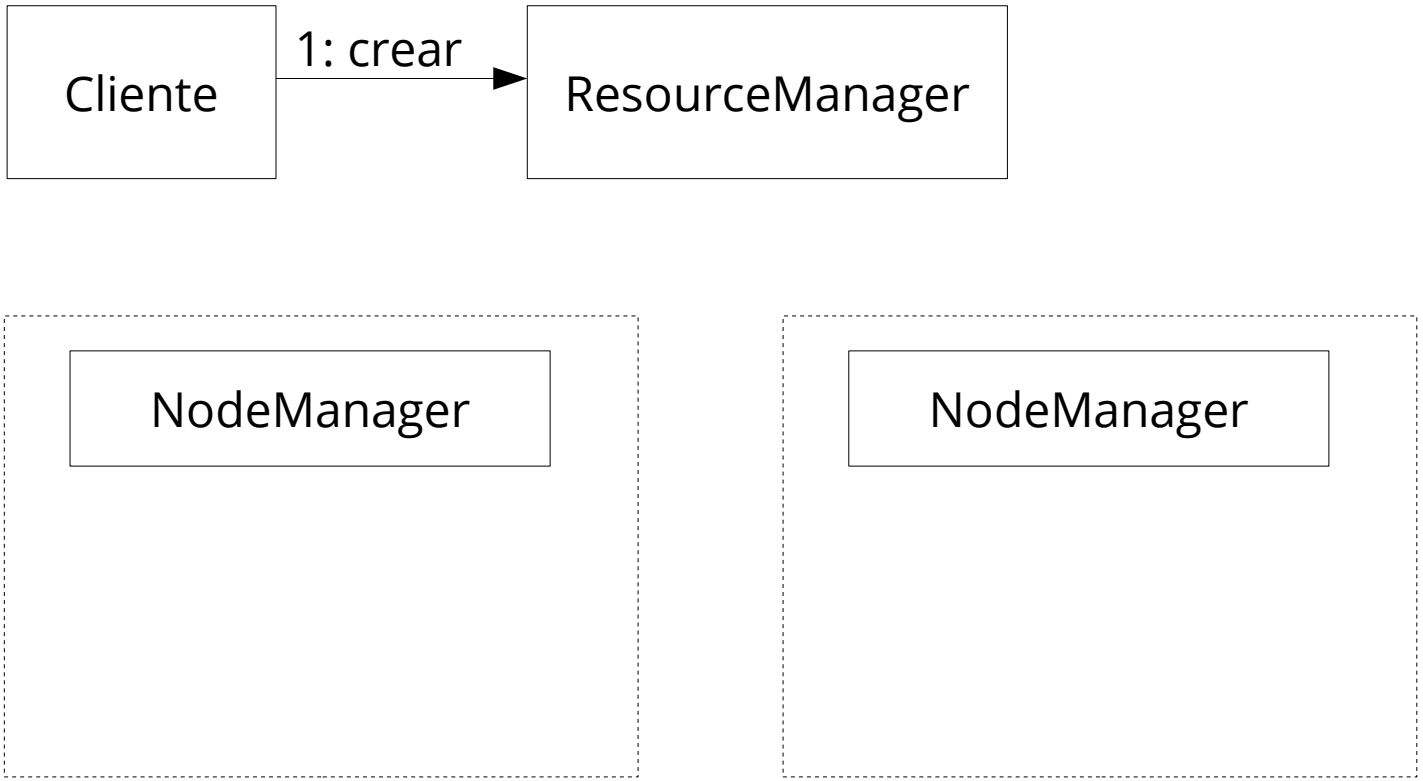
## Arquitectura

- NodeManager
  - Ejecuta los contenedores
  - Crea, monitoriza y mata contenedores, por orden del ResourceManager
  - Periódicamente proporciona informes del estado de los contenedores al ResourceManager

# Hadoop > YARN

## Funcionamiento

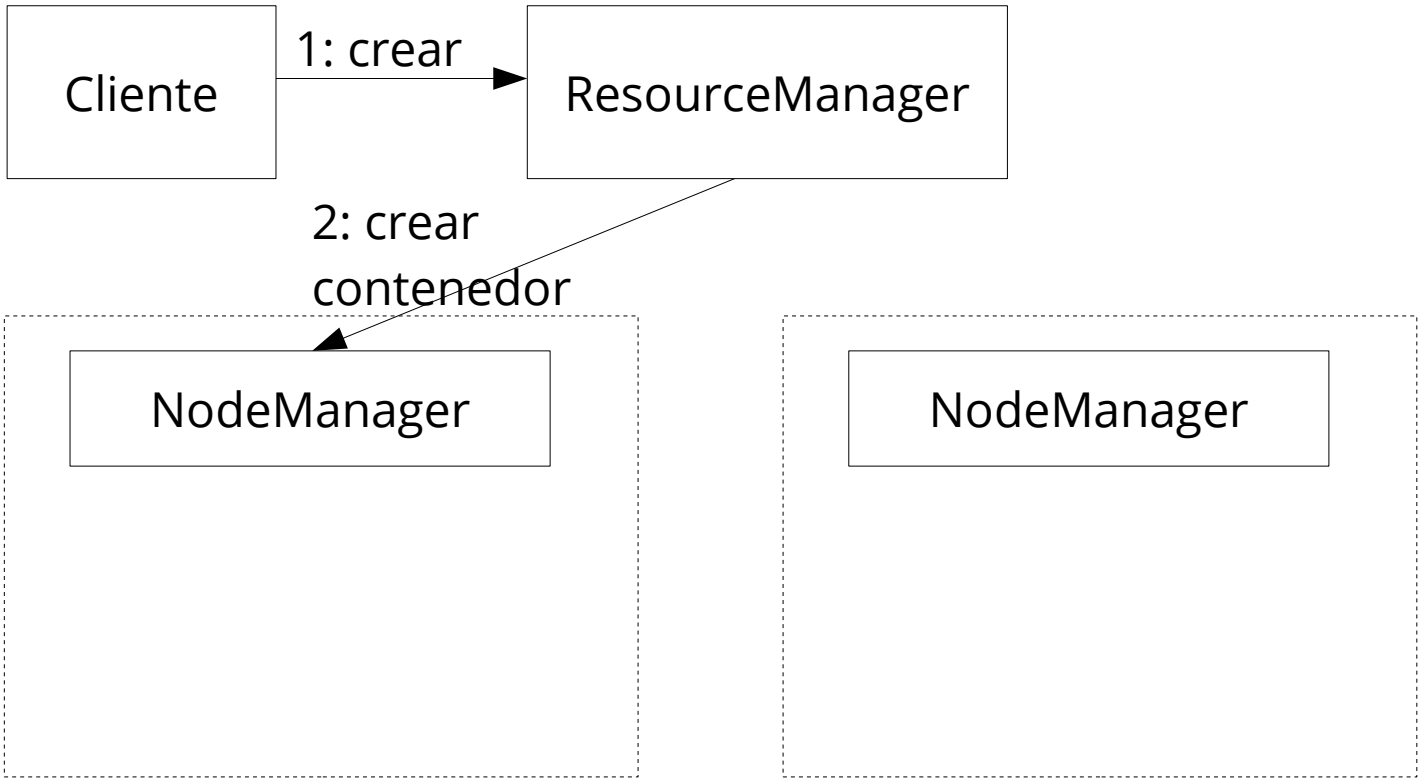
- Un cliente crea una aplicación YARN



# Hadoop > YARN

## Funcionamiento

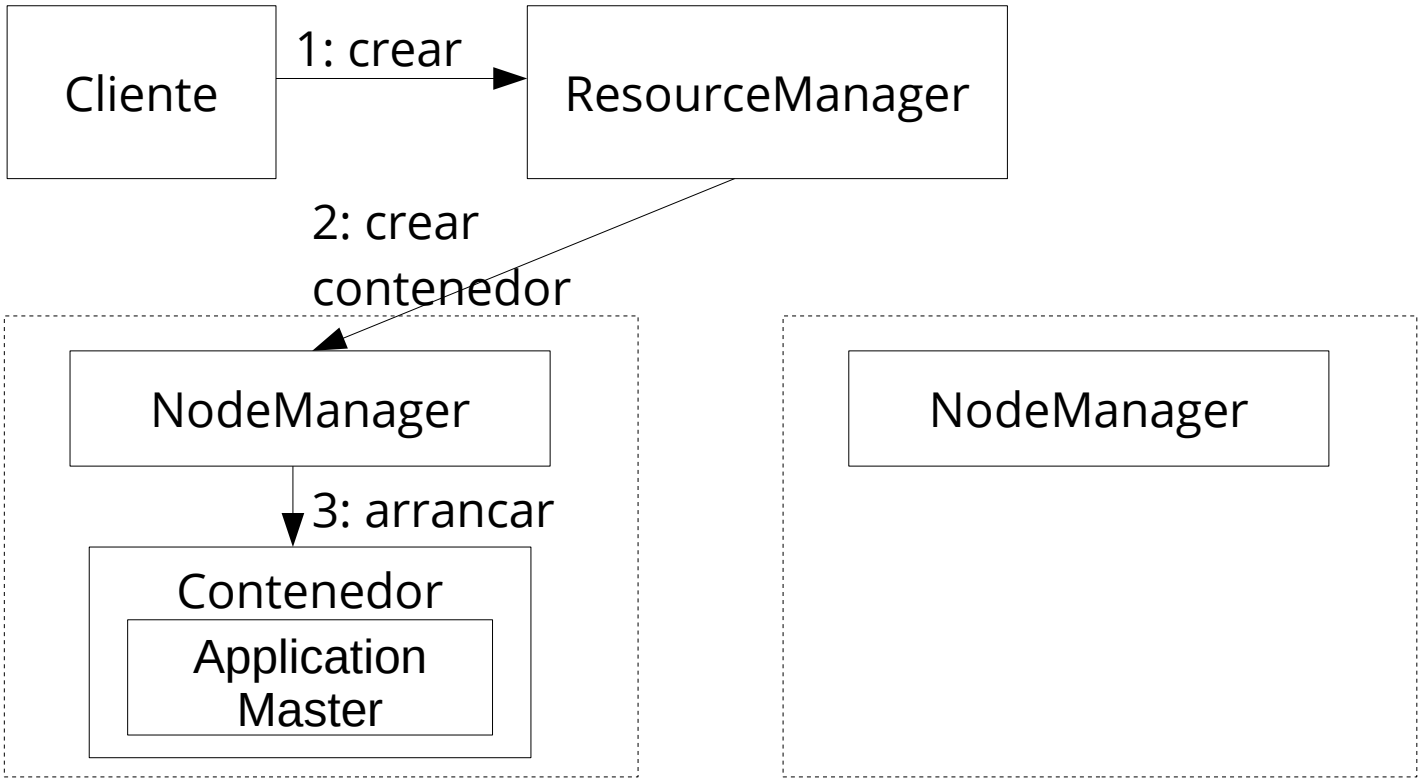
- Un cliente crea una aplicación YARN



# Hadoop > YARN

## Funcionamiento

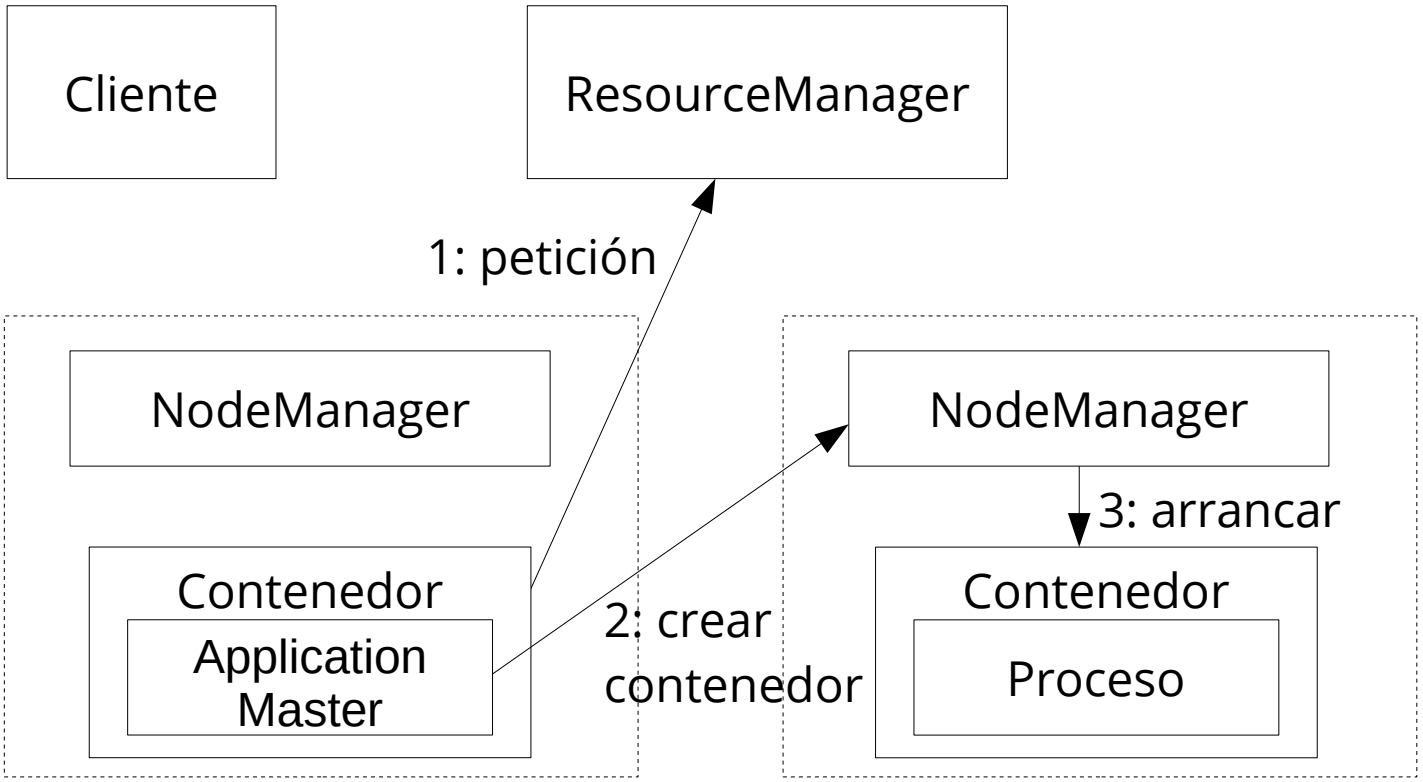
- Un cliente crea una aplicación YARN



# Hadoop > YARN

## Funcionamiento

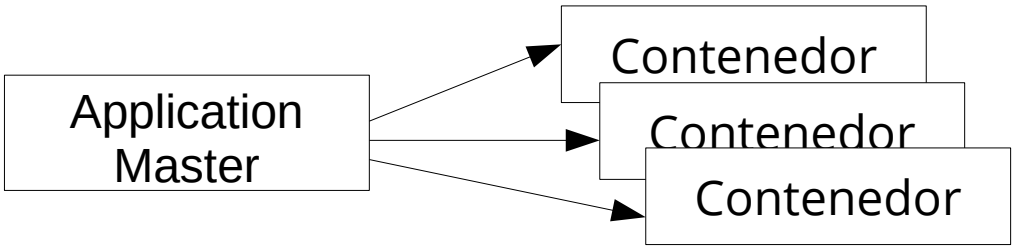
- ApplicationMaster solicita nuevos recursos



# Hadoop > YARN

## ApplicationMaster

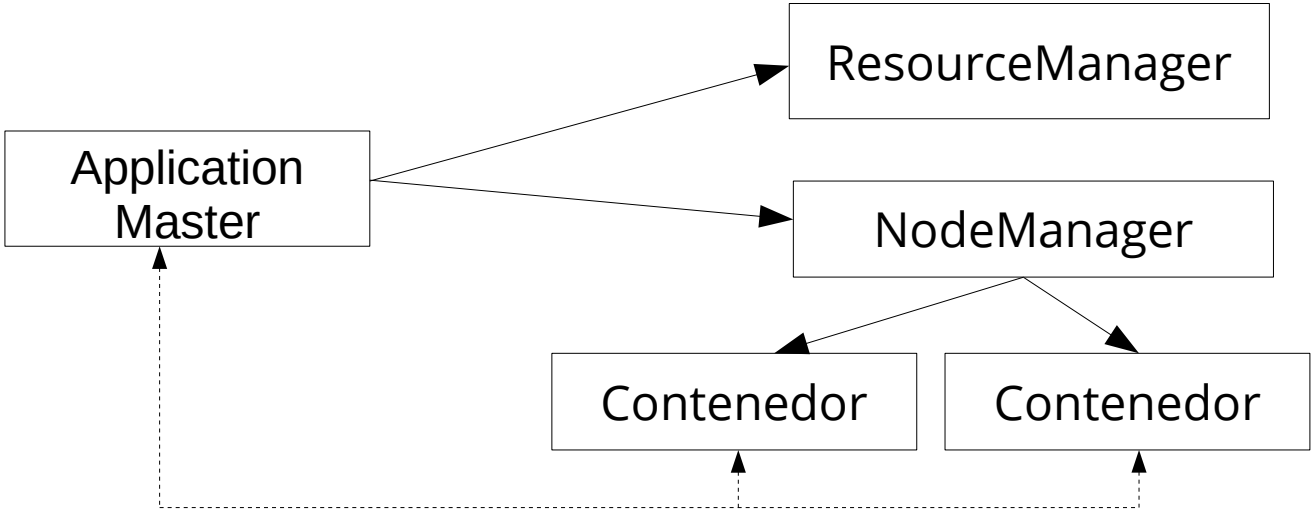
- Se encarga de distribuir el trabajo necesario para completar la tarea (un único proceso, cientos de procesos, etc.)



# Hadoop > YARN

## ApplicationMaster

- Se encarga de distribuir el trabajo necesario para completar la tarea (un único proceso, cientos de procesos, etc.)
- Solicita la creación de nuevos contenedores, los coordina y los reinicia si fallan



# Hadoop > YARN

## ApplicationMaster

- Se encarga de distribuir el trabajo necesario para completar la tarea (un único proceso, cientos de procesos, etc.)
- Solicita la creación de nuevos contenedores, los coordina y los reinicia si fallan
- Distintos modelos: una aplicación por trabajo (e.g. MapReduce), una aplicación por sesión de usuario, una aplicación compartida por varios usuarios, etc.

# Hadoop > YARN

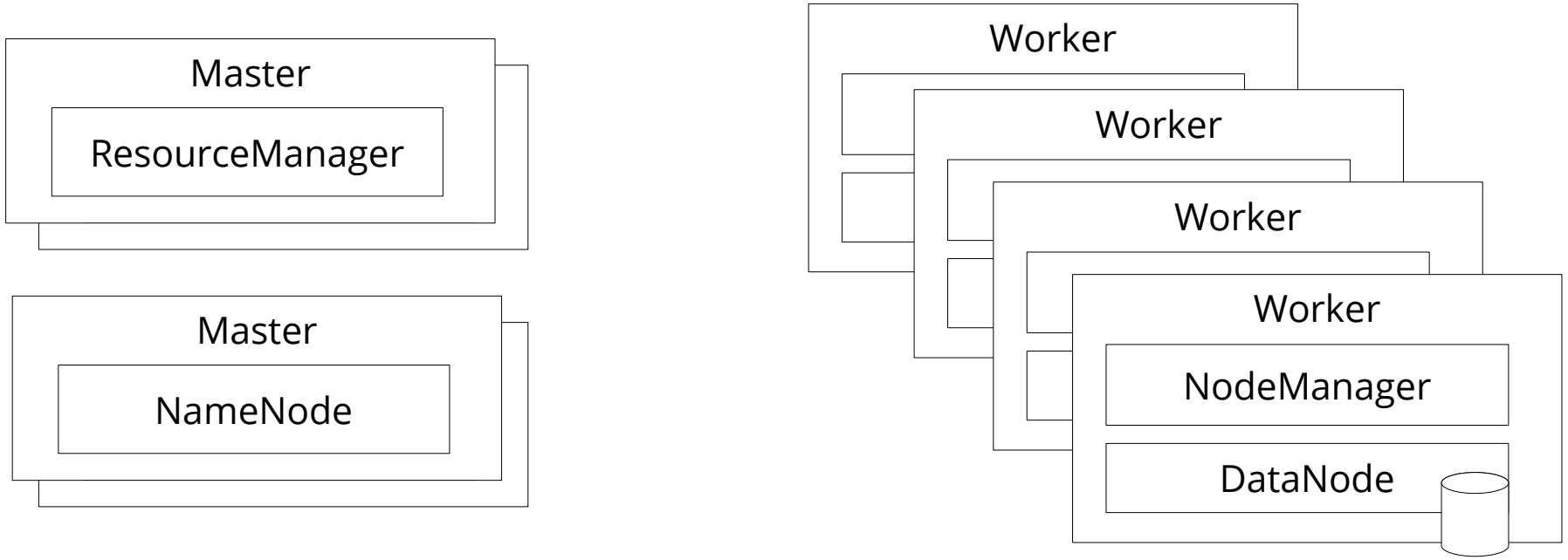
## Solicitud de contenedores

- Se pueden solicitar todos al principio, bajo demanda, etc.
- Se especifican los recursos necesarios (CPU, memoria, GPU, disco, etc.)
- Se pueden especificar restricciones de localidad: en qué nodo, rack, etc.
- Esto permite que un contenedor se pueda ejecutar en el mismo nodo en el que reside un bloque HDFS

# Hadoop > YARN

## HDFS + YARN

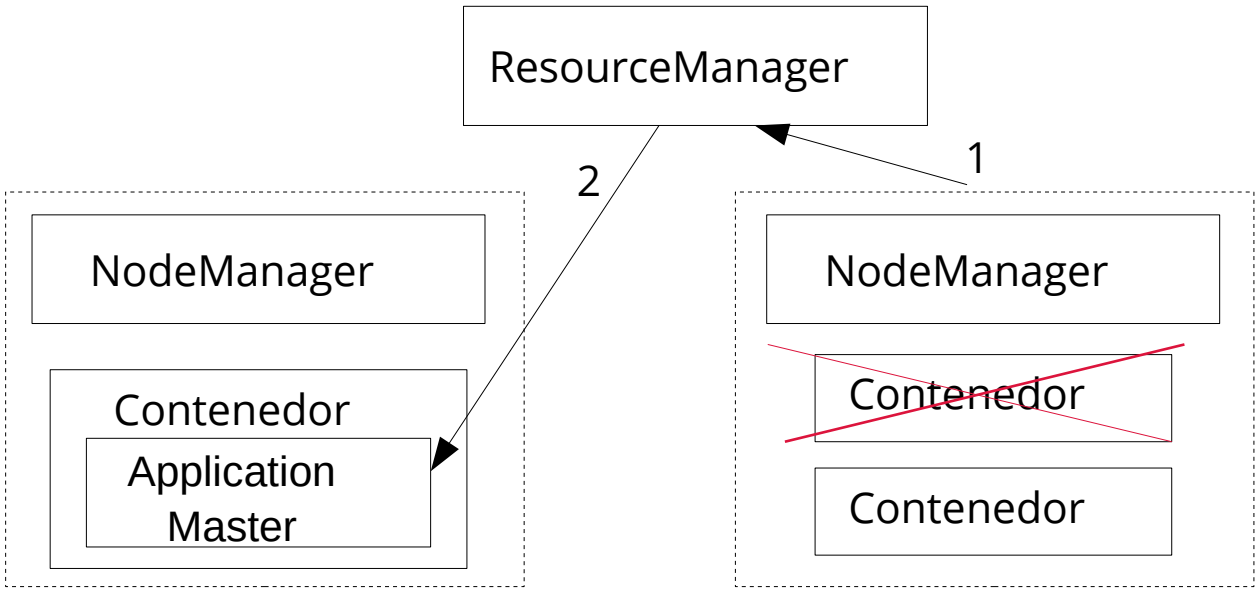
- En un clúster en producción: nodos maestro vs nodos trabajadores



# Hadoop > YARN

## HA (High Availability)

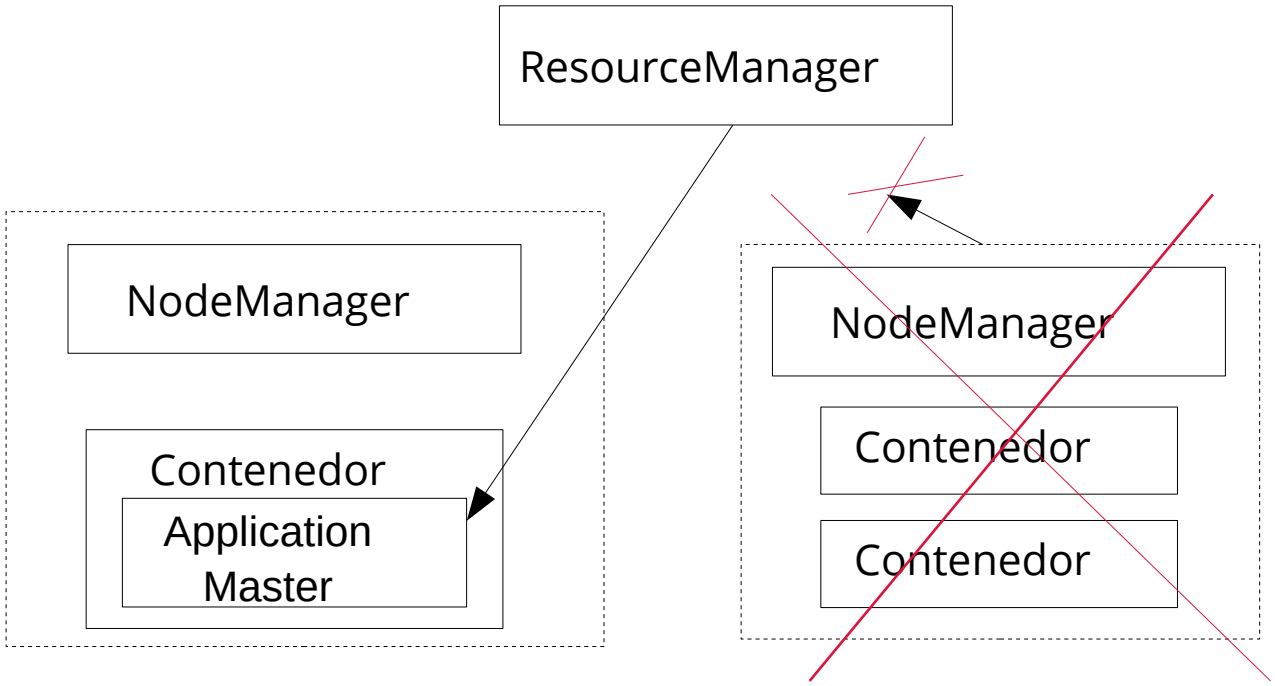
- Si un contenedor falla, el NodeManager notifica al ResourceManager, y éste al ApplicationMaster
- ApplicationMaster decide si el contenedor debe ser ejecutado otra vez



# Hadoop > YARN

## HA (High Availability)

- Si un NodeManager falla, todos sus contenedores se marcan como fallados, y se sigue el mismo procedimiento



# Hadoop > YARN

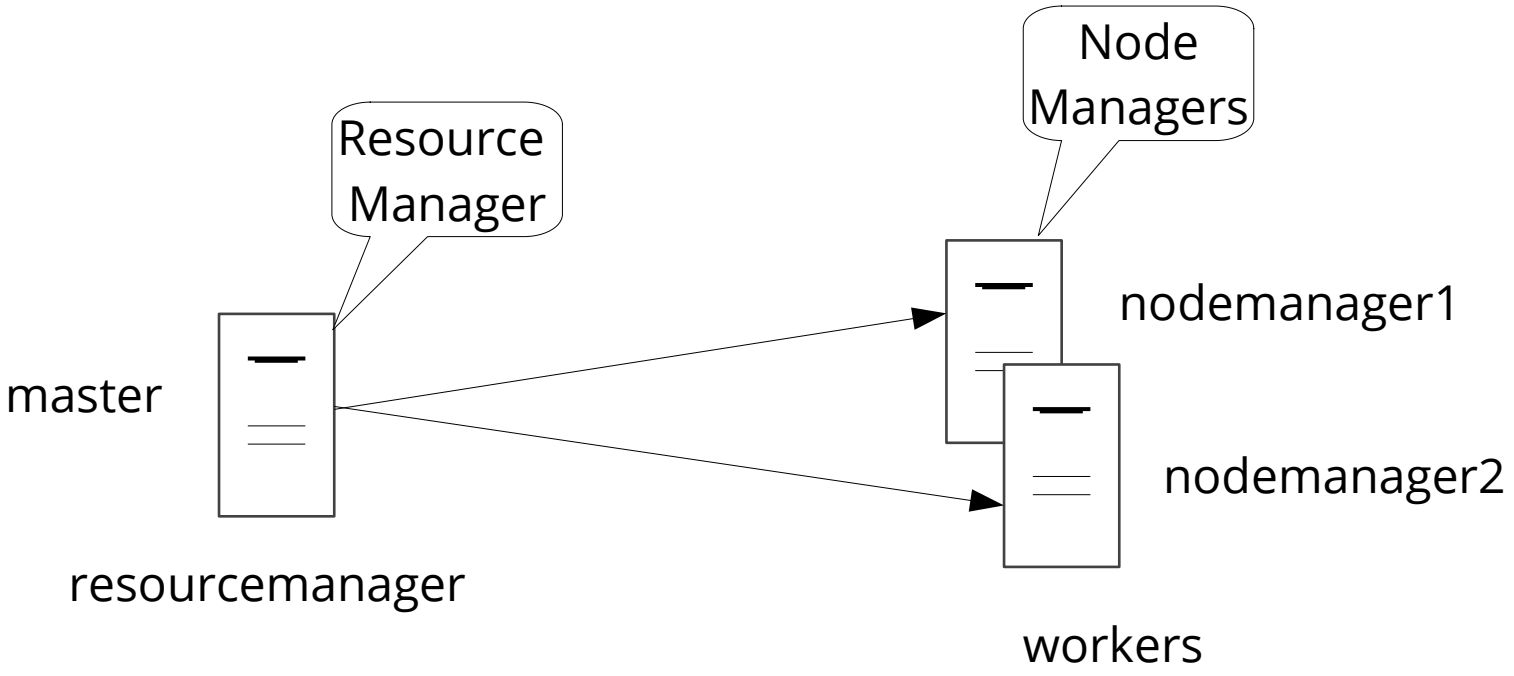
## HA (High Availability)

- ¿Y si falla el ResourceManager?
- Es necesario replicarlo: múltiples RMs comparten estado (clave-valor ZooKeeper)
- Los demás (cliente, NodeManagers, ApplicationMaster) se comunican con los RMs en round-robin (sólo el RM activo responde)

# Hadoop > YARN

## Puesta en marcha en Docker

- Definición del clúster



# Hadoop > YARN

## Puesta en marcha en Docker

- Usando la imagen oficial apache/hadoop
- config.env

```
HADOOP_HOME=/opt/hadoop
CORE-SITE.XML_fs.default.name=hdfs://namenode
CORE-SITE.XML_fs.defaultFS=hdfs://namenode
YARN-SITE.XML_yarn.resourcemanager.hostname=resourcemanager
YARN-SITE.XML_yarn.nodemanager.pmem-check-enabled=false
YARN-SITE.XML_yarn.nodemanager.delete.debug-delay-sec=600
YARN-SITE.XML_yarn.nodemanager.vmem-check-enabled=false
YARN-SITE.XML_yarn.nodemanager.aux-services=mapreduce_shuffle
YARN-SITE.XML_yarn.log-aggregation-enable=true
CAPACITY-SCHEDULER.XML_yarn.scheduler.capacity.maximum-
applications=10000
...
```

# Hadoop > YARN

## Puesta en marcha en Docker

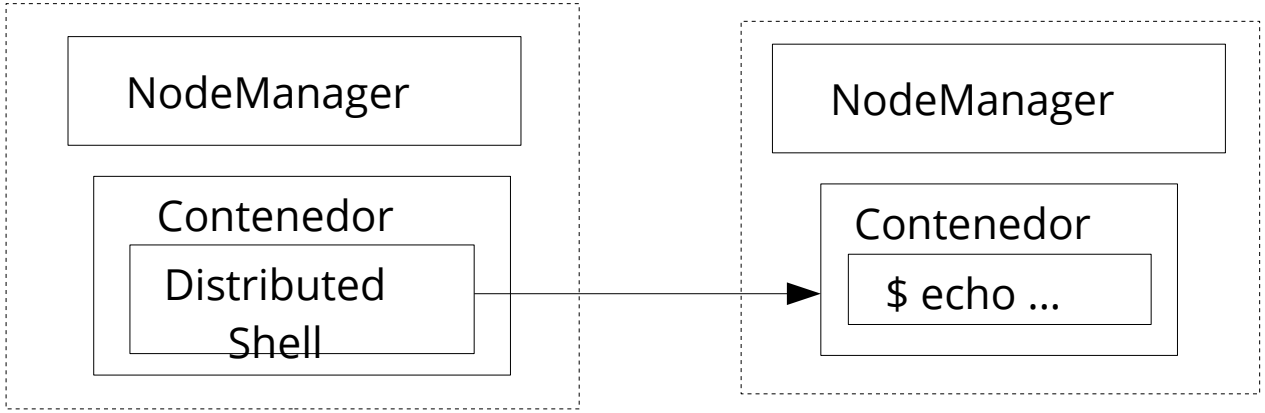
- El ResourceManager publica una GUI web en http://localhost:8088/
- Iniciar sesión en namenode:

```
> docker compose exec resourcemanager /bin/bash
```

# Hadoop > YARN

## DistributedShell

- YARN integra dos aplicaciones por defecto
  - DistributedShell: ejecuta un comando en el clúster
  - MapReduce: lo veremos después
- Ejecutar DistributedShell: echo “HelloWorld!”



# Hadoop > YARN

## DistributedShell

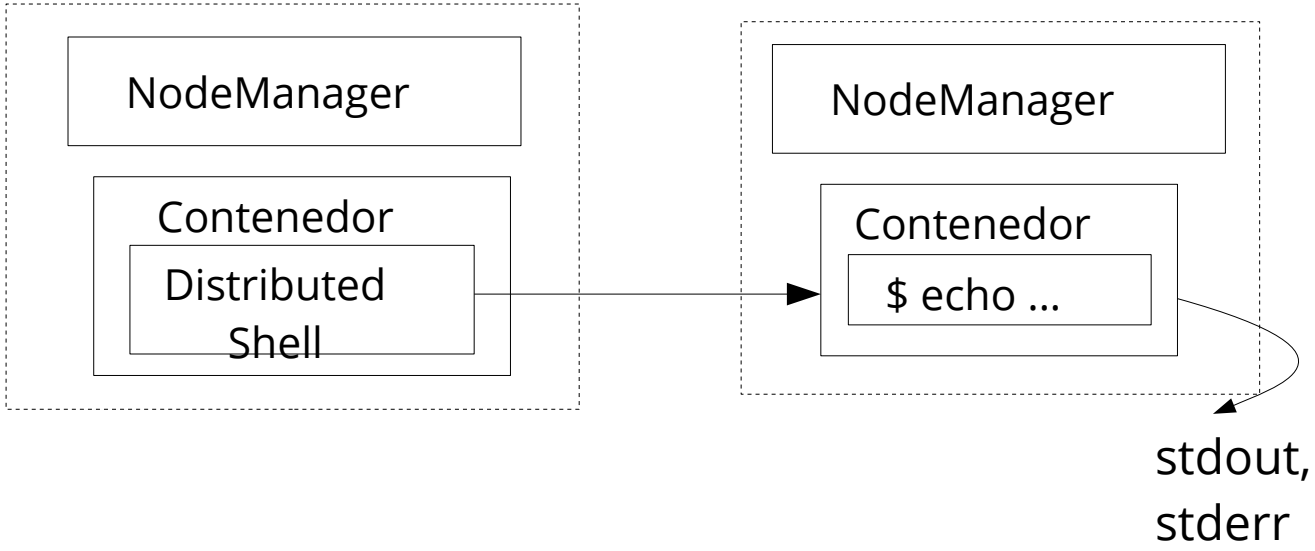
- YARN integra dos aplicaciones por defecto
  - DistributedShell: ejecuta un comando en el clúster
  - MapReduce: lo veremos después
- Ejecutar DistributedShell: echo “HelloWorld!”

```
> export HADOOP_HOME=/home/alumno/hadoop-xxx
> bin/hadoop
org.apache.hadoop.yarn.applications.distributedshell.Client
-shell_command echo -shell_args '"Hello world!"'
-jar ${HADOOP_HOME}/share/hadoop/yarn/*distributedshell*.jar
-container_memory 350 -master_memory 350
```

# Hadoop > YARN

## DistributedShell > Logs

- La tarea se ejecuta en un NodeManager
- Los logs (stdout, stderr) se guardan en local, y se eliminan tras cierto tiempo



# Hadoop > YARN

## DistributedShell > Logs

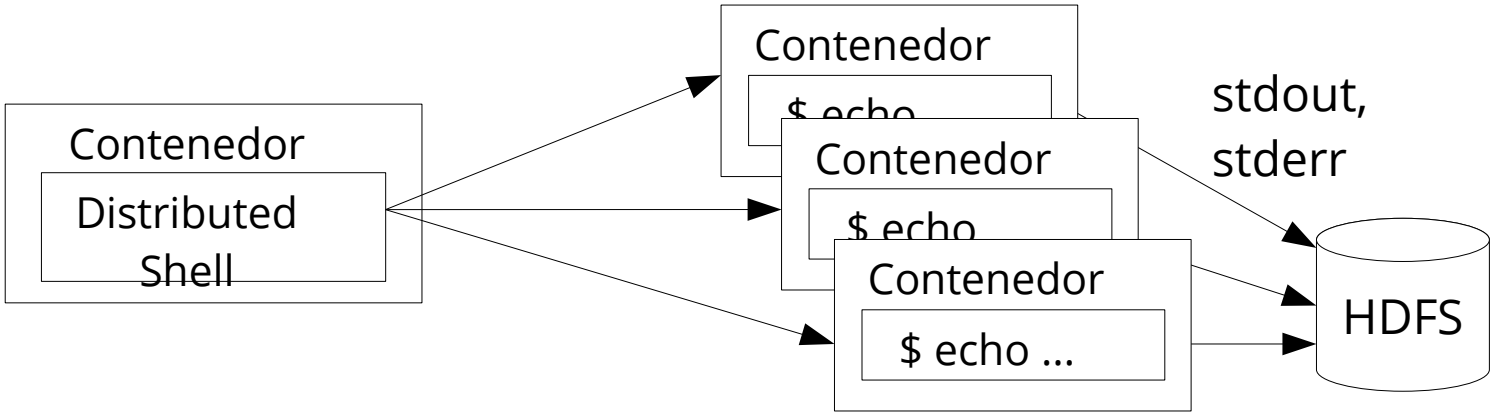
- La tarea se ejecuta en un NodeManager
- Los logs (stdout, stderr) se guardan en local, y se eliminan tras cierto tiempo
- En la GUI web se puede acceder a los logs del contenedor master (DistributedShell)
- Para ver el resto de logs debe publicarlos la ApplicationMaster; DistributedShell no lo hace
- Otra opción es habilitar agregación de logs

YARN-SITE.XML_yarn.log-aggregation-enable=true

# Hadoop > YARN

## Agregación de logs

- Los logs de todos los contenedores se guardan en HDFS



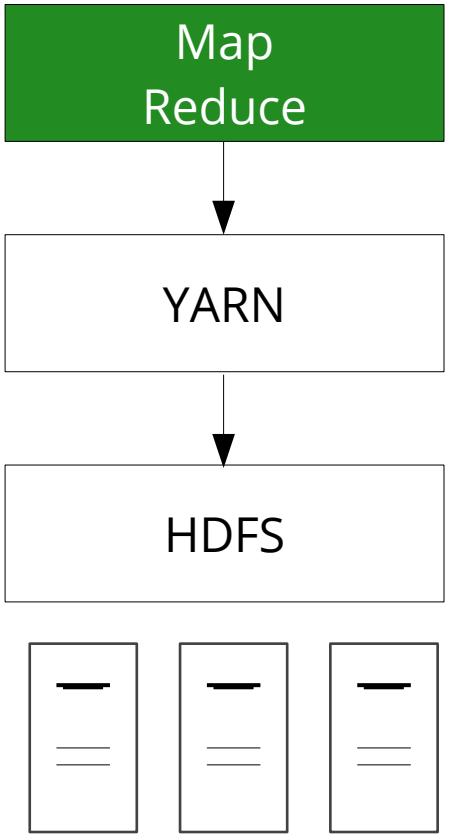
# Hadoop > YARN

## Agregación de logs

- Los logs de todos los contenedores se guardan en HDFS
- Se pueden consultar utilizando CLI

```
> bin/yarn logs -applicationId application_xxxx
```

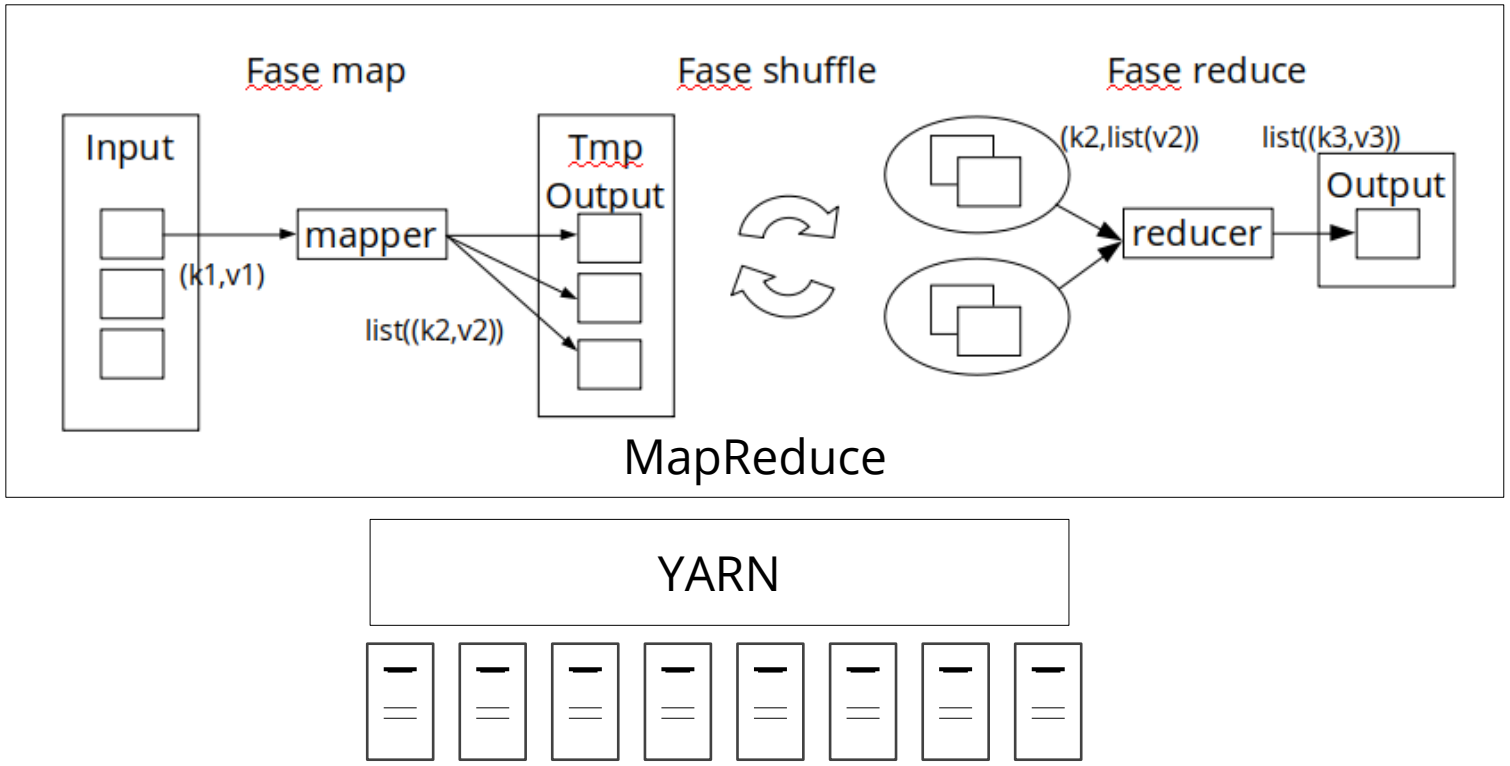
# Hadoop > MapReduce



# Hadoop > MapReduce

## MapReduce en Hadoop

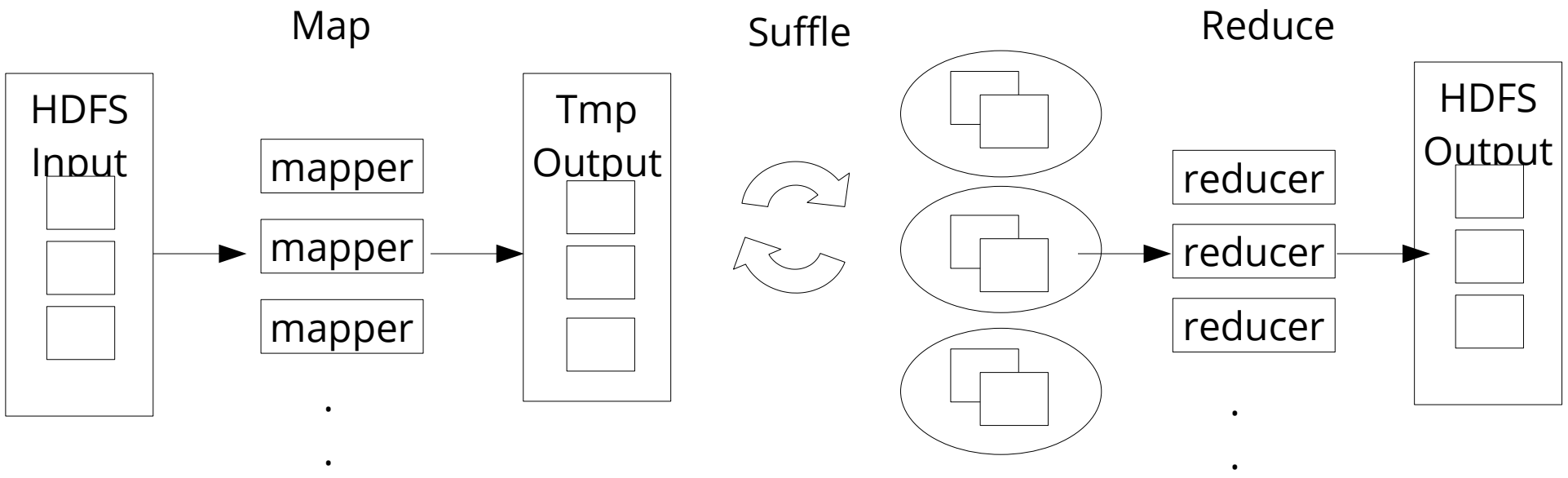
- MapReduce es una aplicación que se ejecuta sobre YARN



# Hadoop > MapReduce

## MapReduce en Hadoop

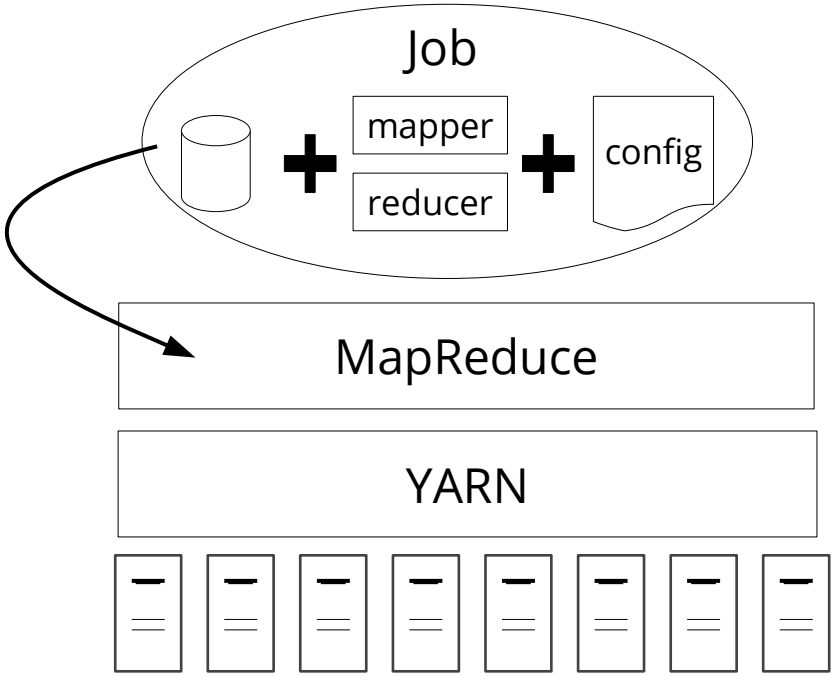
- 1 Job ejecuta múltiples mappers/reducers en paralelo



# Hadoop > MapReduce

## MapReduce en Hadoop

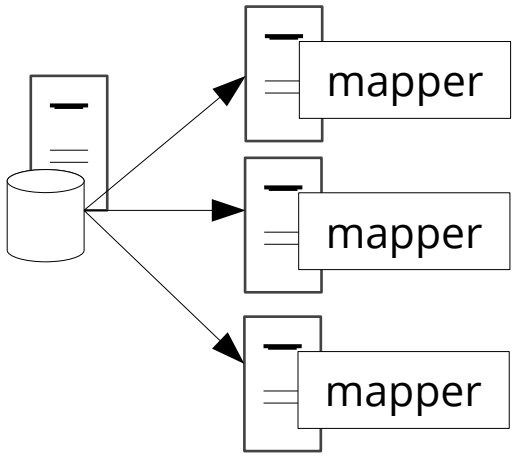
- 1 Job ejecuta múltiples mappers/reducers en paralelo
- Job = entrada + mapper/reducer + config



# Hadoop > MapReduce

## MapReduce en Hadoop > Entrada

- Los datos están en una máquina, cada mapper se ejecuta en otra: coste alto en la distribución de datos

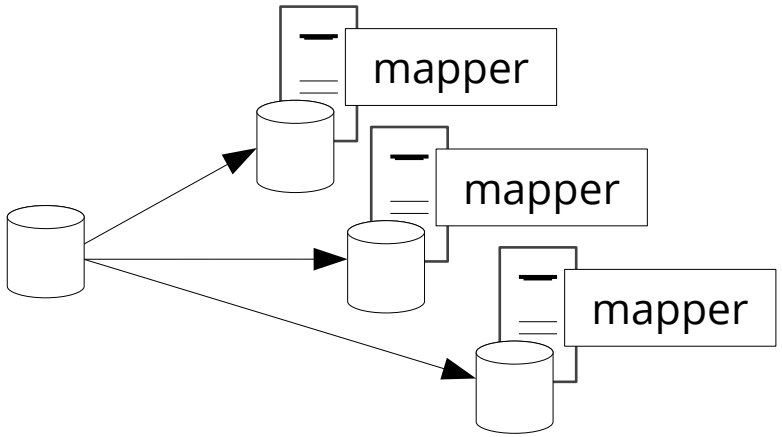



...

# Hadoop > MapReduce

## MapReduce en Hadoop > Entrada

- Los datos están en una máquina, cada mapper se ejecuta en otra: coste alto en la distribución de datos
- Localidad de datos: particionar datos primero y ejecutar tareas donde estén los datos después

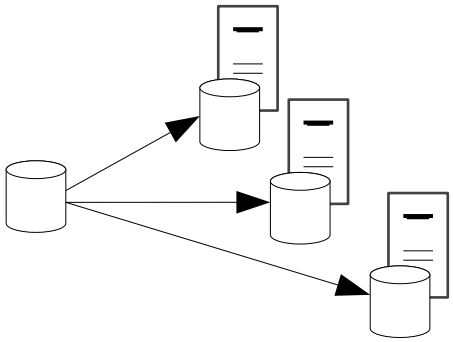



...

# Hadoop > MapReduce

## MapReduce en Hadoop > Entrada

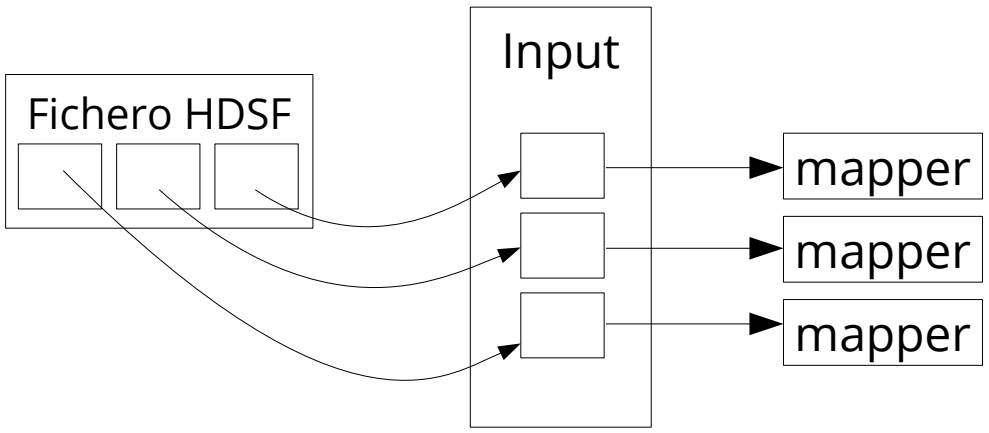
- Los datos están en una máquina, cada mapper se ejecuta en otra: coste alto en la distribución de datos
- Localidad de datos: particionar datos primero y ejecutar tareas donde estén los datos después
- Es necesario contar con sistemas de ficheros distribuidos: HDFS!!!



# Hadoop > MapReduce

## MapReduce en Hadoop > Entrada

- Generalmente en HDFS: varios ficheros en dir
- Se divide en splits. Cada split es procesado por un mapper



# Hadoop > MapReduce

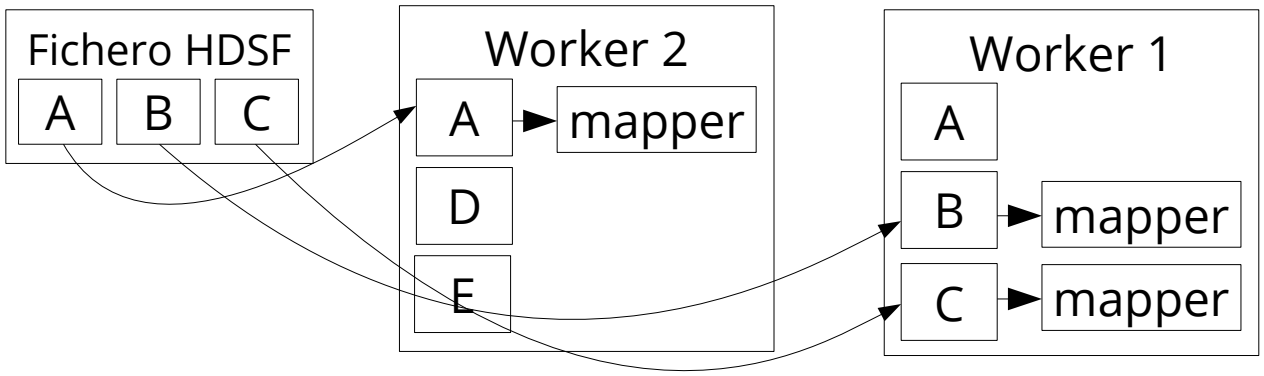
## MapReduce en Hadoop > Entrada

- Número de splits = número de mappers
- Si split pequeño/grande
  - Muchos/pocos mappers en paralelo
  - Mucho/poco trabajo de coordinación
- Habitualmente 1 split = 1 bloque (128MB?)

# Hadoop > MapReduce

## MapReduce en Hadoop > Map

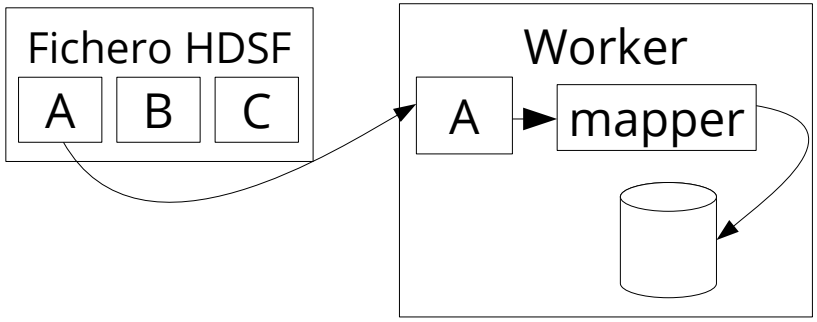
- Localidad de datos: cada mapper se distribuye al nodo donde está el bloque, o al mismo rack



# Hadoop > MapReduce

## MapReduce en Hadoop > Map

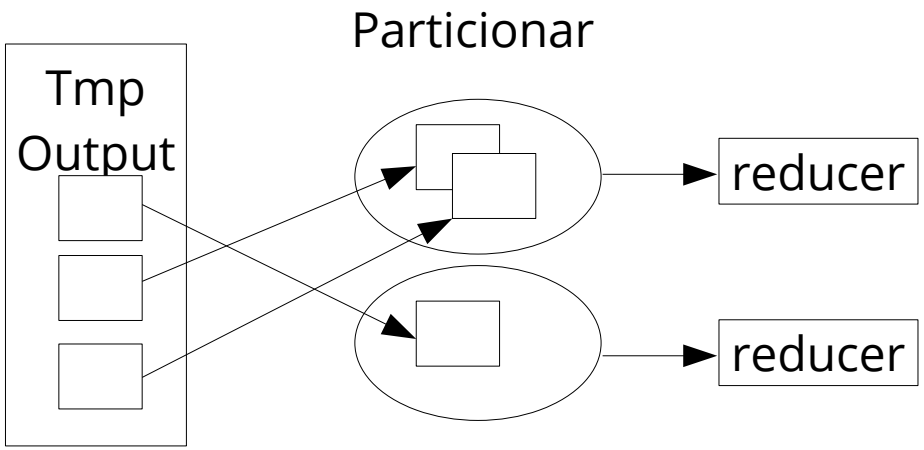
- Localidad de datos: cada mapper se distribuye al nodo donde está el bloque, o al mismo rack
- La salida del mapper se escribe en local
  - Es una salida intermedia; se desecha
  - Guardarla en HDFS sería demasiado costoso
  - Si el mapper falla, se vuelve a computar



# Hadoop > MapReduce

## MapReduce en Hadoop > Shuffle/Sort

- Agrupa los valores por clave y ordena las claves
- El número de reducers *r* es configurable
- Se crean *r* particiones de claves (con sus valores)
- Cada partición es procesada por un reducer



# Hadoop > MapReduce

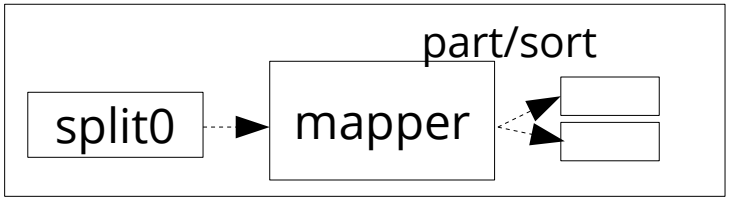
## MapReduce en Hadoop > Shuffle/Sort

- Las particiones las crea una función de particionado: por defecto, se usan técnicas de hashing (balanceo); es configurable
- En una partición
  - Las claves se ordenan
  - Puede haber muchas claves (y los valores asociados)
  - Es mucho trabajo para el reducer

# Hadoop > MapReduce

## MapReduce en Hadoop > Shuffle/Sort

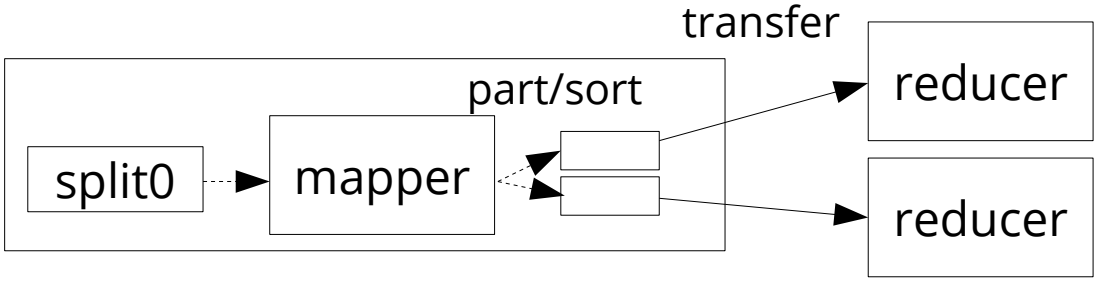
- El particionado y ordenado (parcial) se produce en el nodo que ejecutó el mapper



# Hadoop > MapReduce

## MapReduce en Hadoop > Shuffle/Sort

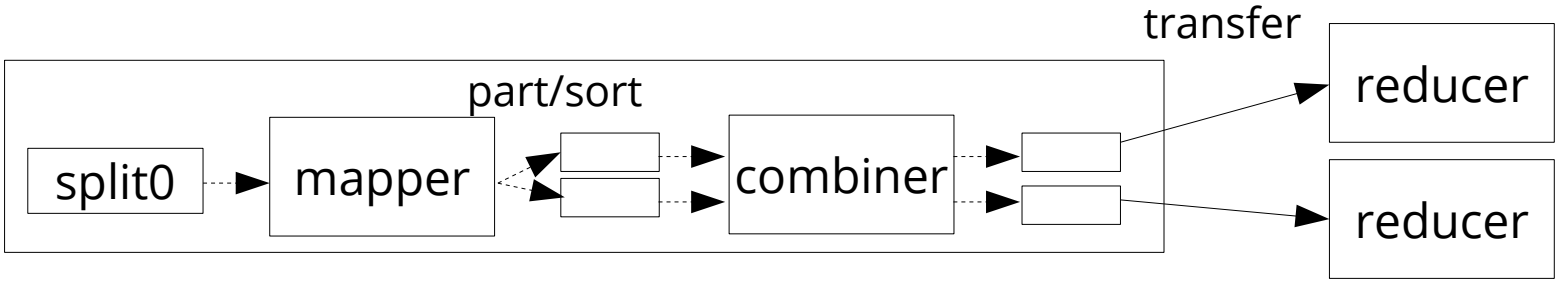
- El particionado y ordenado (parcial) se produce en el nodo que ejecutó el mapper
- Las particiones (parciales) se envían a los reducers
- No se puede aprovechar localidad de datos; se transfieren muchos datos



# Hadoop > MapReduce

## MapReduce en Hadoop > Shuffle/Sort

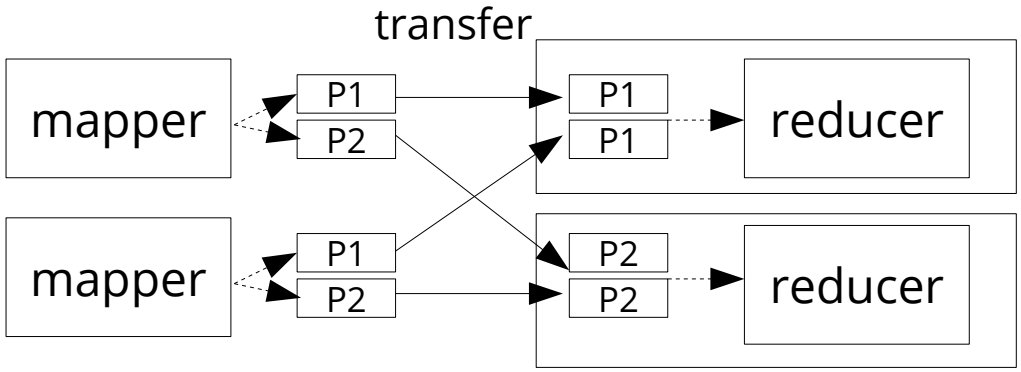
- El particionado y ordenado (parcial) se produce en el nodo que ejecutó el mapper
- Las particiones (parciales) se envían a los reducers
- No se puede aprovechar localidad de datos; se transfieren muchos datos
- Es habitual efectuar un reduce local en el mapper para transferir menos datos: combiner



# Hadoop > MapReduce

## MapReduce en Hadoop > Reduce

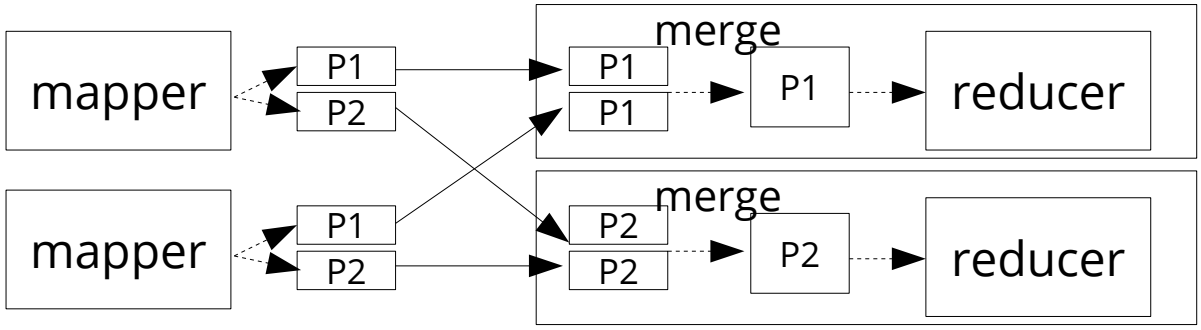
- Cada reducer recibe las particiones (parciales) de múltiples mappers



# Hadoop > MapReduce

## MapReduce en Hadoop > Reduce

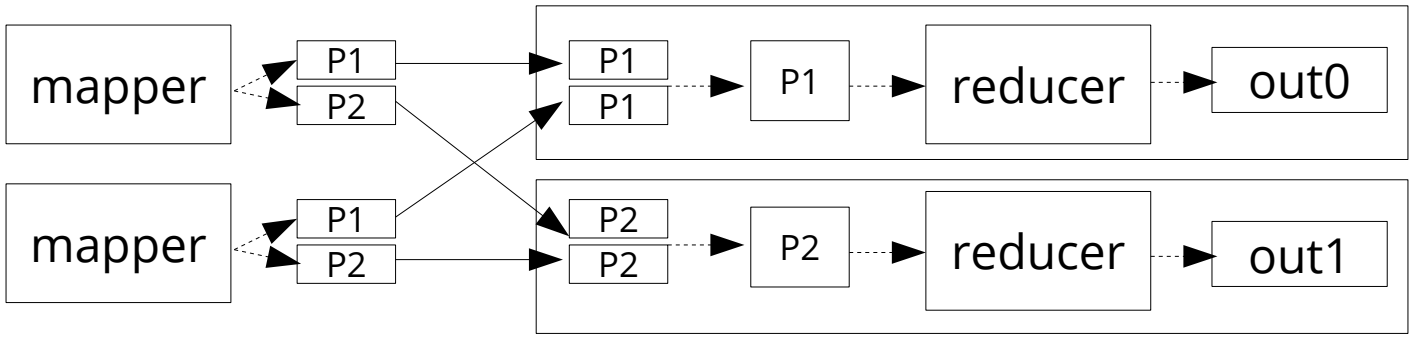
- Cada reducer recibe las particiones (parciales) de múltiples mappers
- El reducer debe fusionar (ordenadamente) las particiones (mergesort?)



# Hadoop > MapReduce

## MapReduce en Hadoop > Reduce

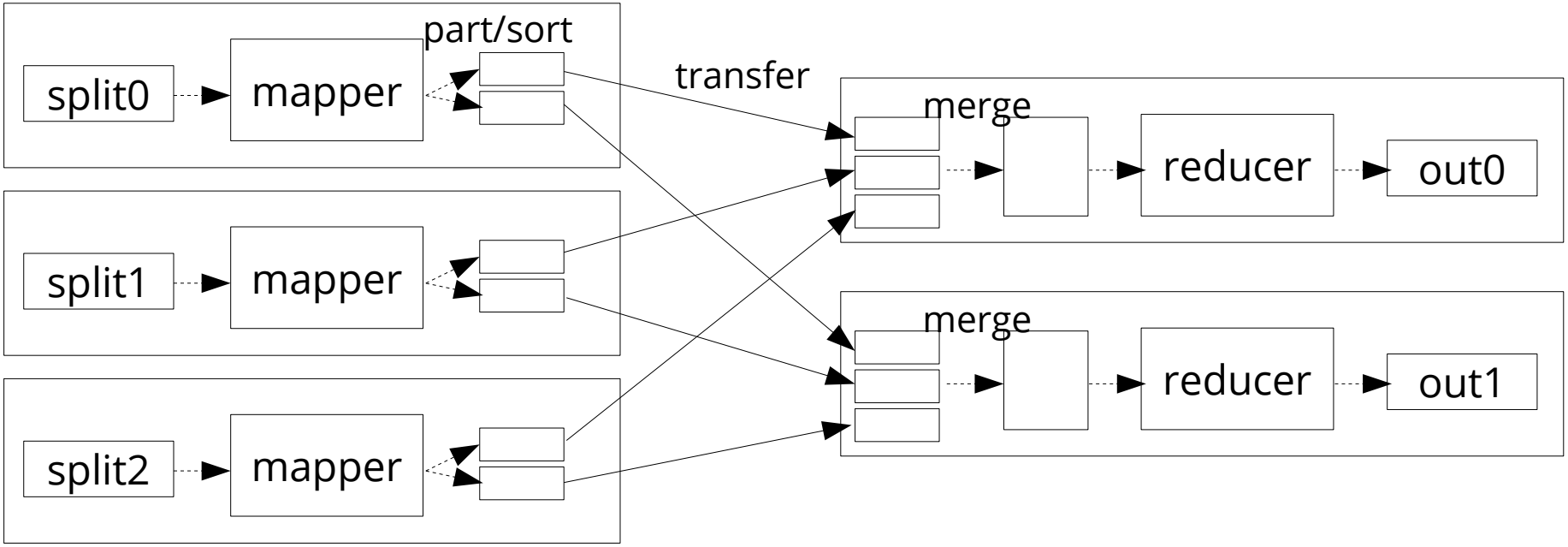
- Cada reducer recibe las particiones (parciales) de múltiples mappers
- El reducer debe fusionar (ordenadamente) las particiones (mergesort?)
- Cada reducer escribe en HDFS (local – mismo rack - otro rack) aprovechando la localidad de datos



# Hadoop > MapReduce

## MapReduce en Hadoop

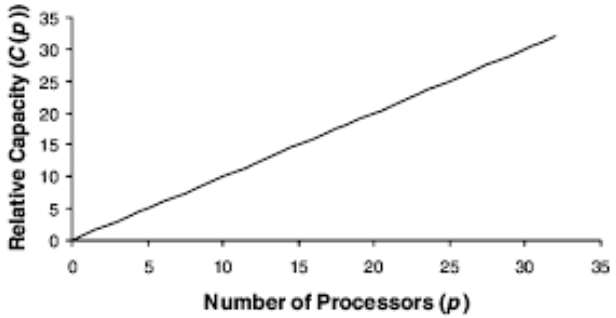
- El esquema completo



# Hadoop > MapReduce

## MapReduce en Hadoop > Propiedades

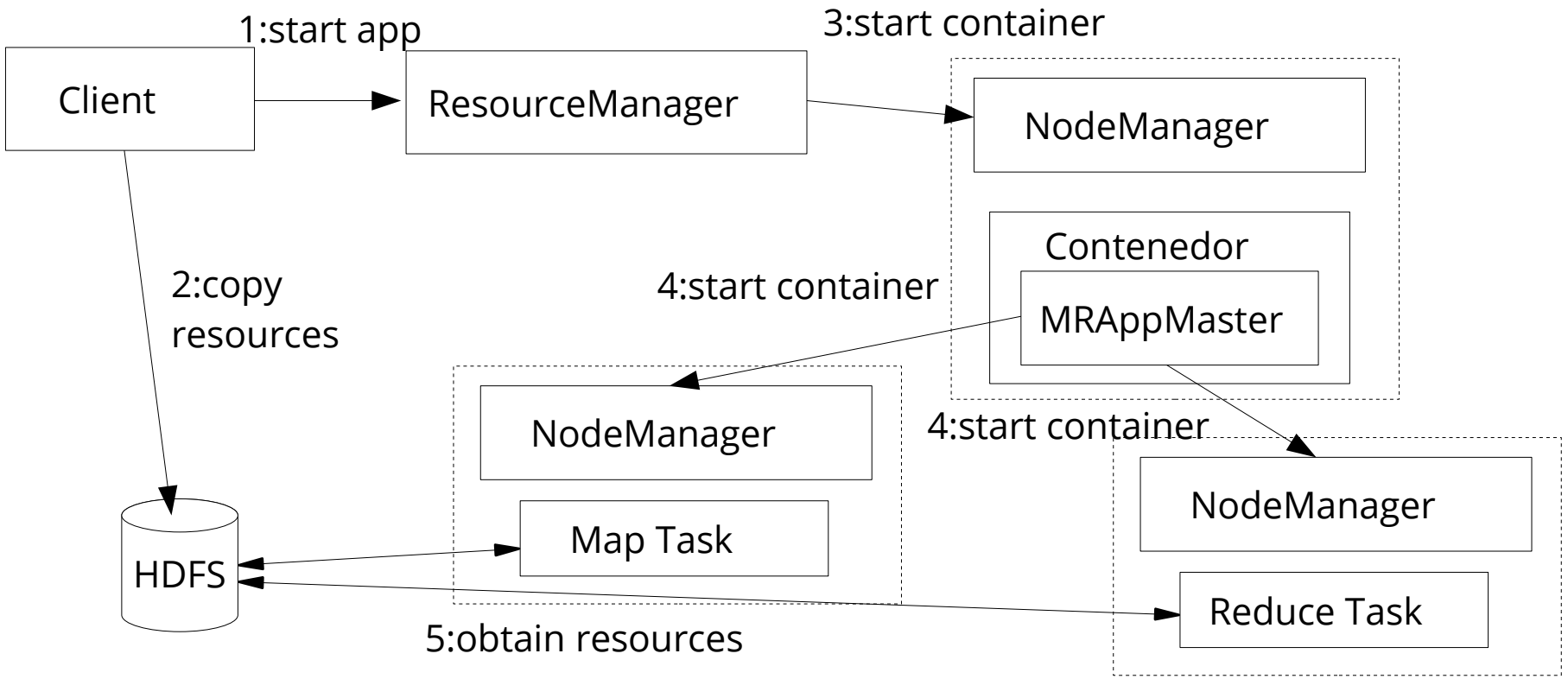
- Si la máquina donde se ejecuta un mapper/reducer falla, se puede relanzar en otra: idempotente
- Ejecución especulativa: ralentización, sospechas de fallos, etc.
- Escalabilidad lineal: la capacidad de proceso aumenta linealmente con respecto a los recursos disponibles



# Hadoop > MapReduce

## MapReduce en Hadoop > YARN

- Hadoop ejecuta trabajos MapReduce utilizando el ApplicationMaster incluido en su distribución



# Hadoop > MapReduce

## MapReduce en Hadoop > API

- Java es el lenguaje de programación nativo

```
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);
    }
  }
}
```

# Hadoop > MapReduce

## MapReduce en Hadoop > API

- Java es el lenguaje de programación nativo

```
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);
  }
}
```

# Hadoop > MapReduce

## MapReduce en Hadoop > API

- Java es el lenguaje de programación nativo

```
public class WordCount {
  public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    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);
    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));
    System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
}
```

# Hadoop > MapReduce

## MapReduce en Hadoop > API

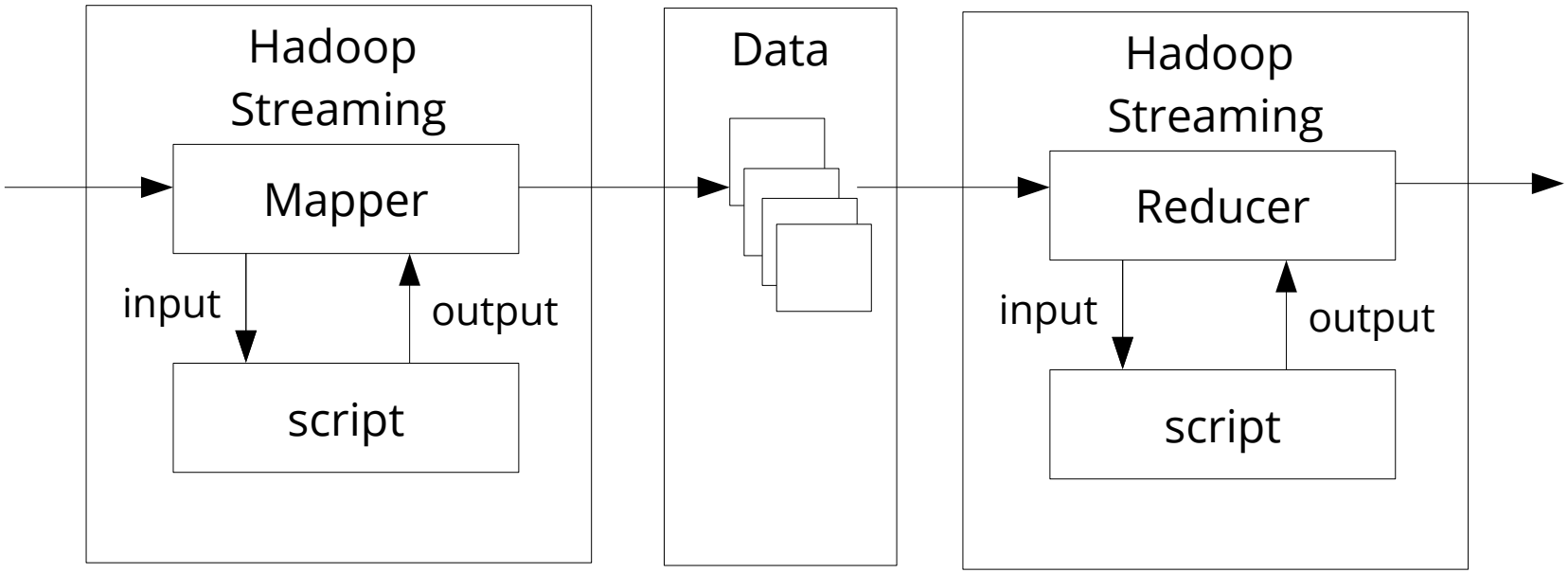
- Java es el lenguaje de programación nativo

```
% Copiar datos de entrada a HDFS
> bin/hadoop fs -cat /input.txt
Hello World Bye World

% Ejecutar MapReduce
> bin/hadoop jar wc.jar WordCount /input.txt /output

% Leer datos de salida en HDFS
> bin/hadoop fs -cat /output/part-r-00000
```

# Hadoop > MapReduce

## Hadoop Streaming

- Hadoop Streaming permite utilizar cualquier ejecutable/script como mapper/reducer



# Hadoop > MapReduce

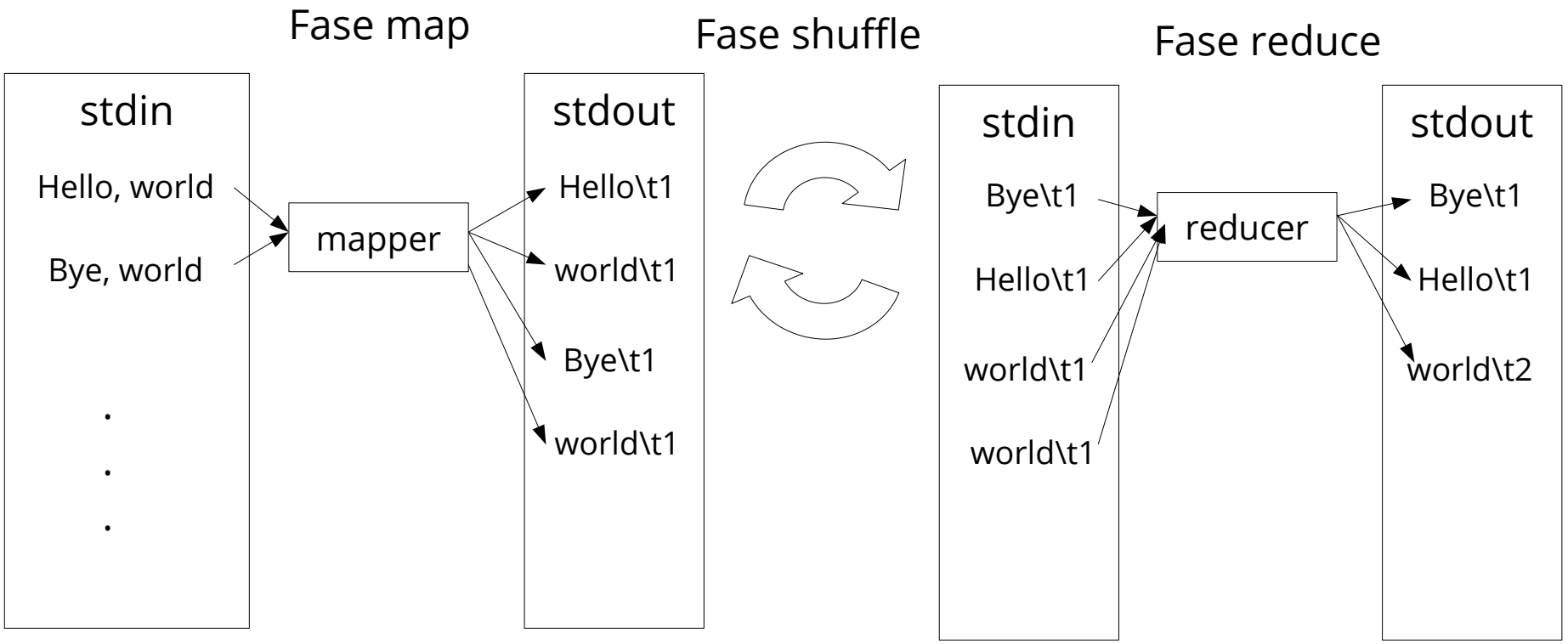
## Hadoop Streaming

- Hadoop Streaming permite utilizar cualquier ejecutable/script como mapper/reducer
- La interfaz de comunicación entre Hadoop y el mapper/reducer es a través de entrada/salida universal stdin/stdout
- La entrada se obtiene de la entrada estándar stdin, una línea cada vez
- La salida se escribe en la salida estándar stdout en el formato: clave TAB valor

# Hadoop > MapReduce

## Hadoop Streaming

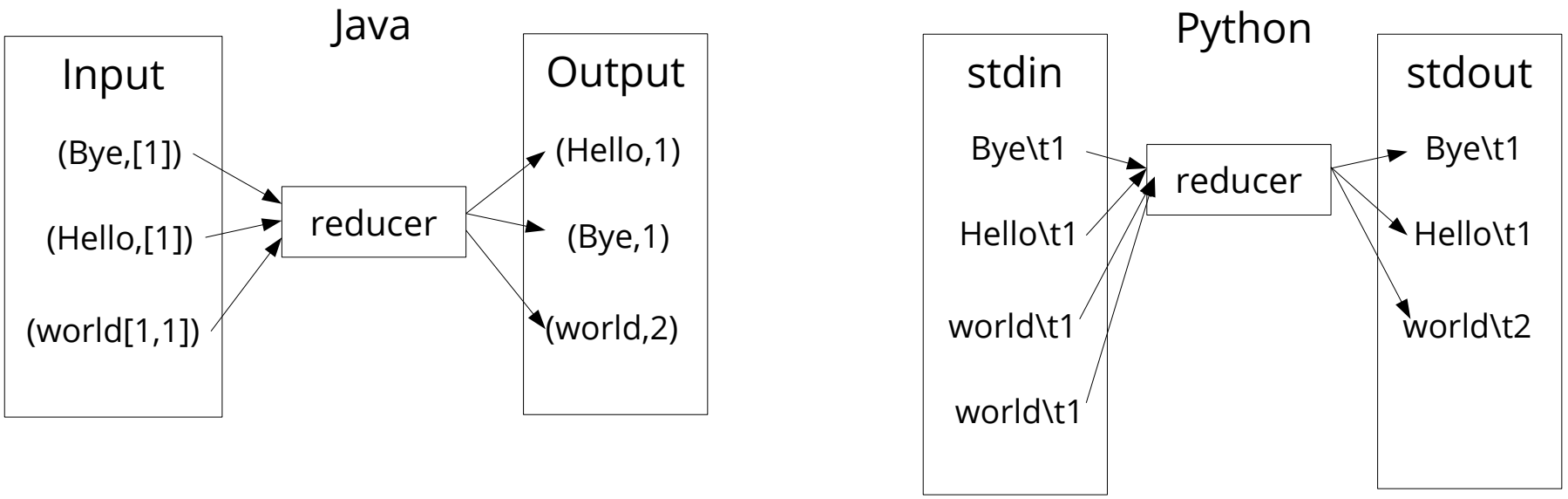
- Ejemplo: contar palabras



# Hadoop > MapReduce

## Hadoop Streaming

- Hay diferencias notables en el reducer
- En Java recibe (k2, list(v2)), en Streaming se recibe (k2,v2) en cada lectura de stdin



# Hadoop > MapReduce

## Hadoop Streaming

- Hay diferencias notables en el reducer
- En Java recibe (k2, list(v2)), en Streaming se recibe (k2,v2) en cada lectura de stdin
- Dificulta un poco la lógica de reducer: es necesario saber cuándo acaban los valores asociados a una clave y cuándo comienzan los de la siguiente

# Hadoop > MapReduce

## Ejemplo

- Contar palabras con Streaming (mapper.py)

```
#!/bin/python
import sys
for line in sys.stdin:
  words = line.split()
  for word in words:
    print("{0}\t{1}".format(word, 1))
```

# Hadoop > MapReduce

## Ejemplo

- Contar palabras con Streaming (reducer.py)

```
#!/bin/python
import sys
curr_word = None
curr_count = 0
for line in sys.stdin:
  word, count = line.split('\t')
  count = int(count)
  if word == curr_word:
    curr_count += count
  else:
    if curr_word:
      print("{0}\t{1}".format(curr_word, curr_count))
    curr_word = word
    curr_count = count
if curr_word == word:
  print("{0}\t{1}".format(curr_word, curr_count))
```

# Hadoop > MapReduce

## Ejemplo

- Contar palabras con Streaming

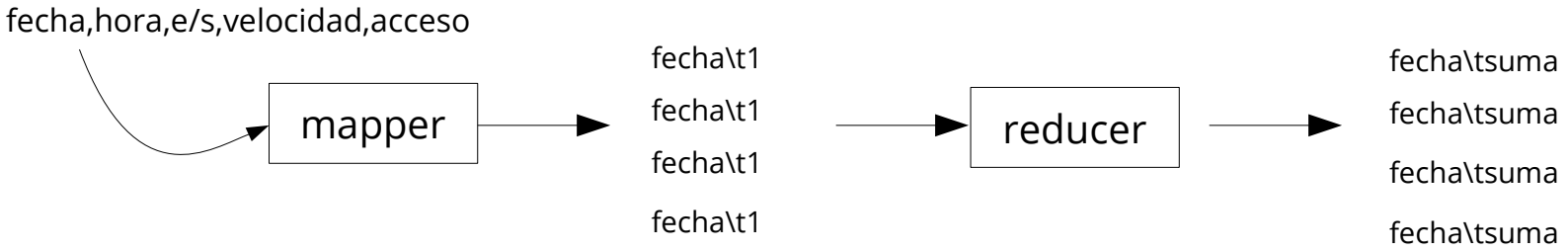
```
% Arrancar clúster e iniciar sesión en ResourceManager
> docker compose up -d
> docker compose exec resourcemanager /bin/bash

% Copiar datos de entrada a HDFS
> bin/hadoop fs -put /shared/el_quijote.txt /

% Ejecutar MapReduce
> bin/hadoop jar $HADOOP_HOME/share/hadoop/tools/lib/hadoop-
streaming-3.3.6.jar \
    -files /shared/mapper.py,/shared/reducer.py
    -input /el_quijote.txt -output /output \
    -mapper mapper.py -reducer reducer.py \
    -numReduceTasks 1

% Leer datos de salida en HDFS
> bin/hadoop fs -cat /output/part-00000
```

# Hadoop > MapReduce

## Ejercicio 2

- A partir del dataset “city_accesses.csv”, contar número de coches que entran en Alcoy cada día
  - https://opendata.alcoi.org/es/
  - Transporte > Entradas y salidas

```
"2020-01-01 00:56:27.537","00:56:28","Acercamiento",
"72","AF_Entrada_Cocentaina"
```

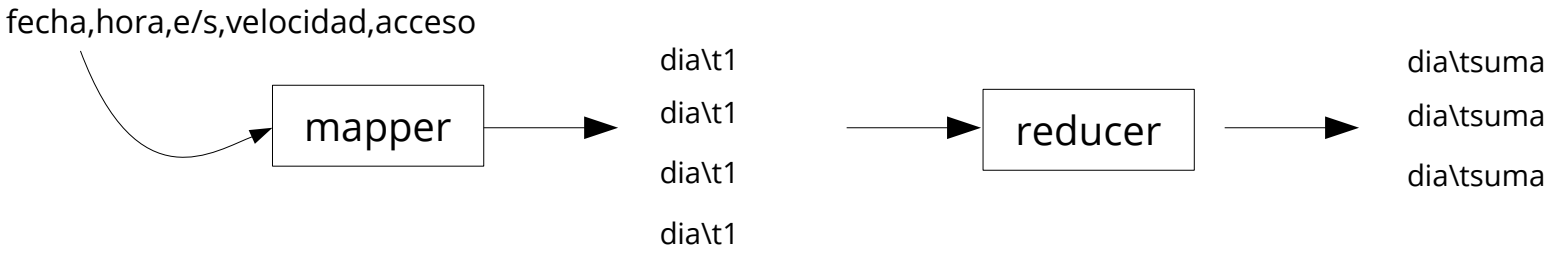


# Hadoop > MapReduce

## Ejercicio 3

- A partir del dataset “city_accesses.csv”, contar cuántos coches entran cada día de la semana

```
"2020-01-01 00:56:27.537","00:56:28","Acercamiento",
"72","AF_Entrada_Cocentaina"
```

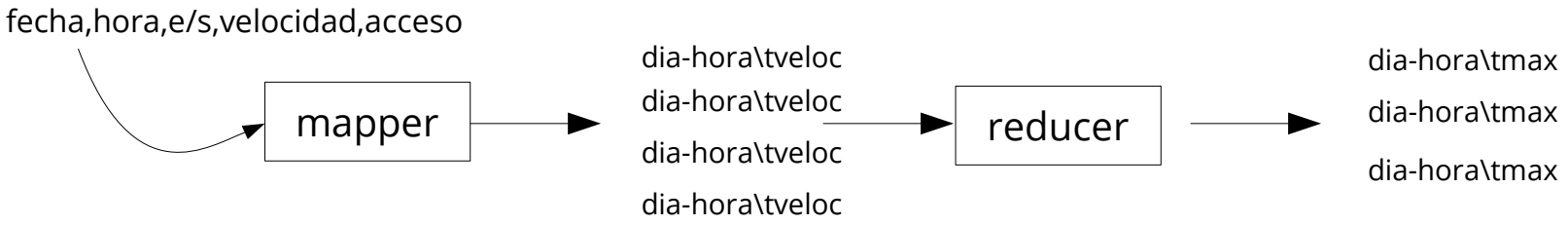


# Hadoop > MapReduce

## Ejercicio 4

- A partir del dataset “city_accesses.csv”, qué día de la semana y a qué hora se corre más

```
"2020-01-01 00:56:27.537","00:56:28","Acercamiento",
"72","AF_Entrada_Cocentaina"
```



# Hadoop > MapReduce

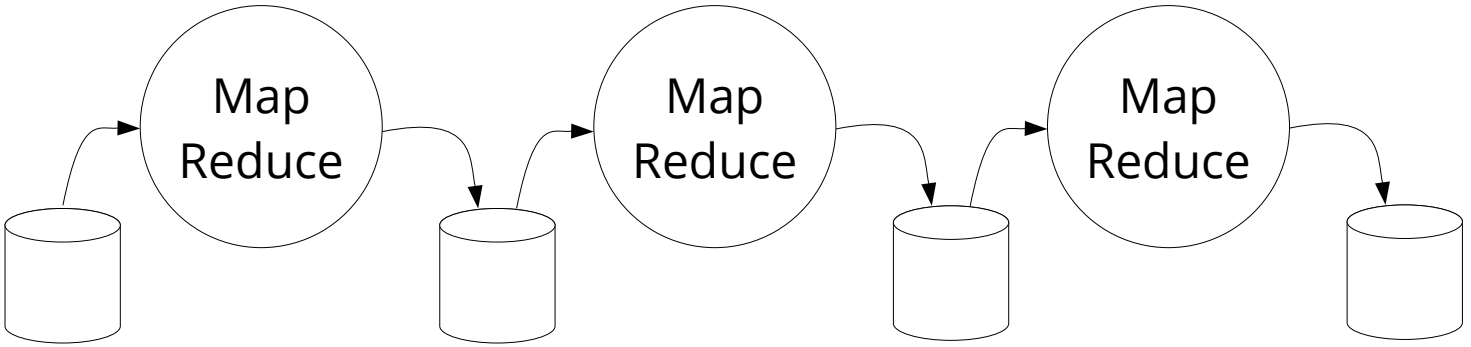
## Críticas a MapReduce

- No todo son luces ….

# Hadoop > MapReduce

## Críticas a MapReduce

- No todo son luces ….
- La salida se almacena en HDFS
  - Es costoso (replicación, etc.)
  - ¿Qué pasa si es un resultado intermedio, parte de un workflow?



# Hadoop > MapReduce

## Críticas a MapReduce

- No todo son luces ….
- La salida se almacena en HDFS
- El modelo MapReduce es muy específico
  - Surgió para indexar la web
  - Hay muchos problemas cuya solución no se adaptan bien al modelo MapReduce
  - ¿Procesos iterativos? Ej: fibonacci, grafos, ...

# Hadoop > MapReduce

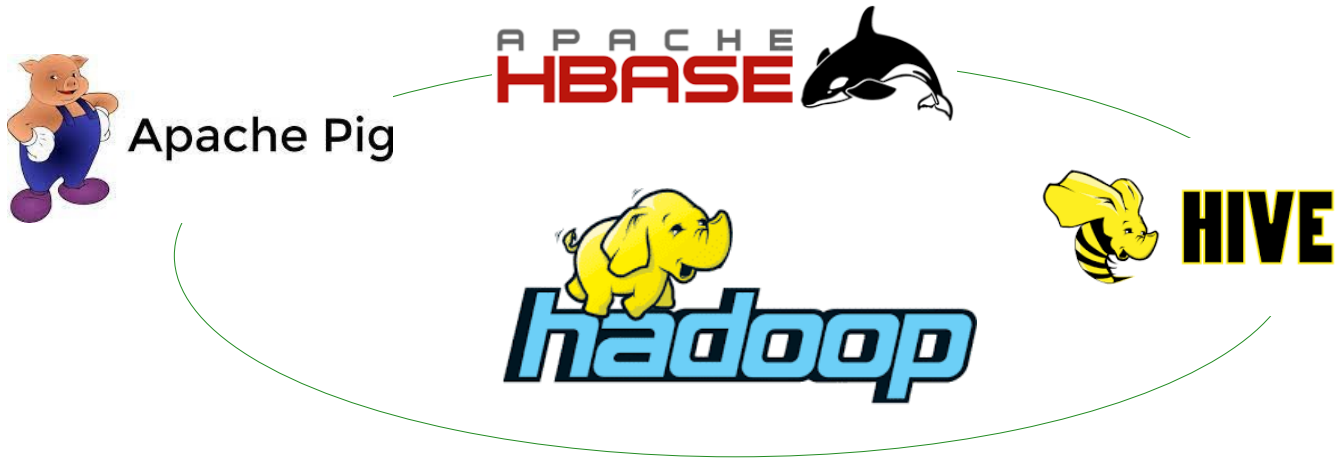
## Críticas a MapReduce

- No todo son luces ….
- La salida se almacena en HDFS
- El modelo MapReduce es muy específico
- MapReduce es difícil de programar
  - No es un modelo de procesamiento de datos natural: bucles y filtros
  - Algunas operaciones de procesamiento de datos son difíciles de traducir a tareas MapReduce

# Hadoop > MapReduce

## Críticas a MapReduce

- Aparecen extensiones de más alto nivel




- Aparecen otras plataformas

# Spark

# Spark

## Contenido

- Introducción
- RDDs
- Spark SQL

# Spark > Introducción

## ¿Qué es?

- http://spark.apache.org/
- Plataforma de computación open source distribuida, fiable y escalable
- Puede ejecutar trabajos en batch/streaming, iterativos, grafos, machine learning, etc.
- Muy rápida: 10x (disco) -100x (memoria) más rápida que Hadoop
- Múltiples lenguajes: Java, Scala, Python, R

# Spark > Introducción

## ¿Qué es?

- Permite efectuar múltiples operaciones sobre una colección de datos

# Spark > Introducción

## ¿Qué es?

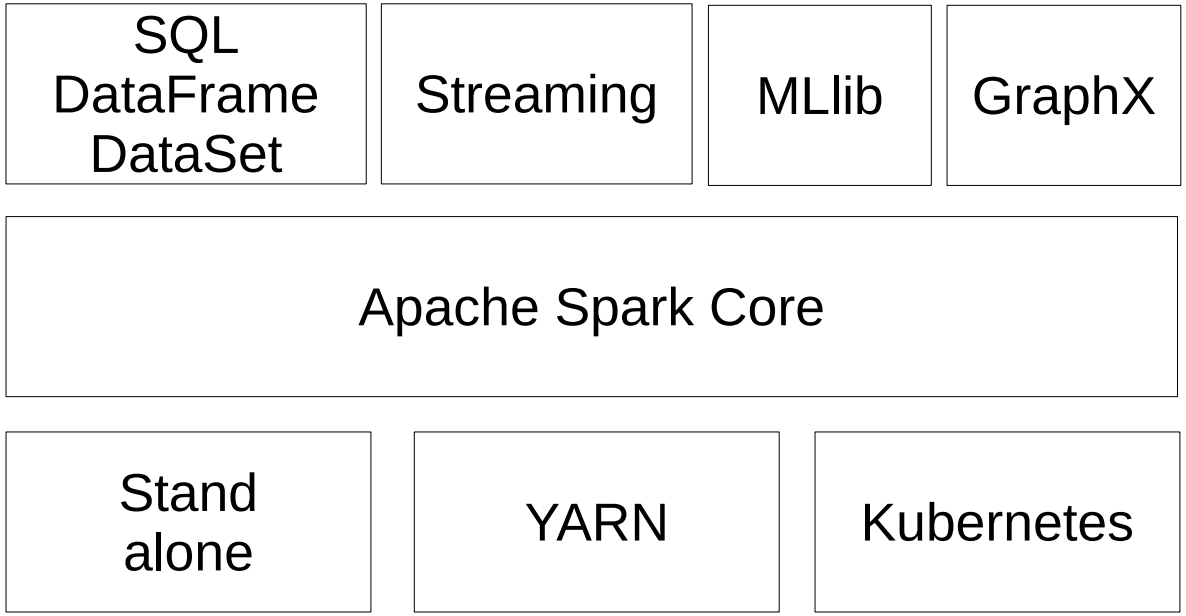
- Permite efectuar múltiples operaciones sobre una colección de datos
- Con una API muy sencilla (ej: wordcount)

```
sc = SparkContext()
file = sc.textFile("hdfs://...")
counts = file.flatMap(lambda line: line.split(" "))
  .map(lambda word: (word, 1)).countByKey()
counts.saveAsTextFile("hdfs://...")
```

# Spark > Introducción

## ¿Qué es?

- Es un framework muy genérico (SQL, stream, etc.)
- Se ejecuta sobre múltiples plataformas



# Spark > Introducción

## Componentes

- Cada componente publica cierta funcionalidad
  - Spark Core
  - Spark SQL
  - Spark Streaming
  - MLlib
  - GraphX

# Spark > Introducción

## Componentes

- Cada componente publica cierta funcionalidad
  - Spark Core
    - RDDs: colecciones básicas, la base de todo

# Spark > Introducción

## Componentes

- Cada componente publica cierta funcionalidad
  - Spark Core
  - Spark SQL
    - DataFrames: colecciones de datos organizadas en columnas (como una tabla)
    - Datasets: añaden tipado de datos (en Python no está disponible)
    - SQL: subconjunto de SQL

# Spark > Introducción

## Componentes

- Cada componente publica cierta funcionalidad
  - Spark Core
  - Spark SQL
  - Spark Streaming
    - Extensión de Spark Core para procesar streams de datos continuos

# Spark > Introducción

## Componentes

- Cada componente publica cierta funcionalidad
  - Spark Core
  - Spark SQL
  - Spark Streaming
  - MLlib
    - Contiene algoritmos (escalables) de Machine Learning

# Spark > Introducción

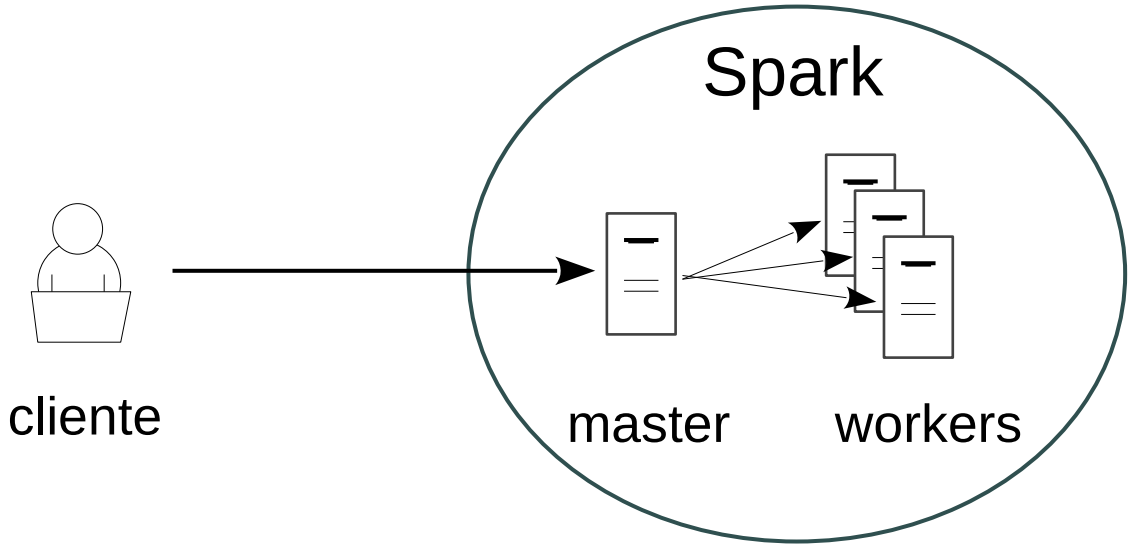
## Componentes

- Cada componente publica cierta funcionalidad
  - Spark Core
  - Spark SQL
  - Spark Streaming
  - MLlib
  - GraphX
    - Incluye algoritmos para procesamiento de grafos en paralelo

# Spark > Introducción

## Arquitectura global

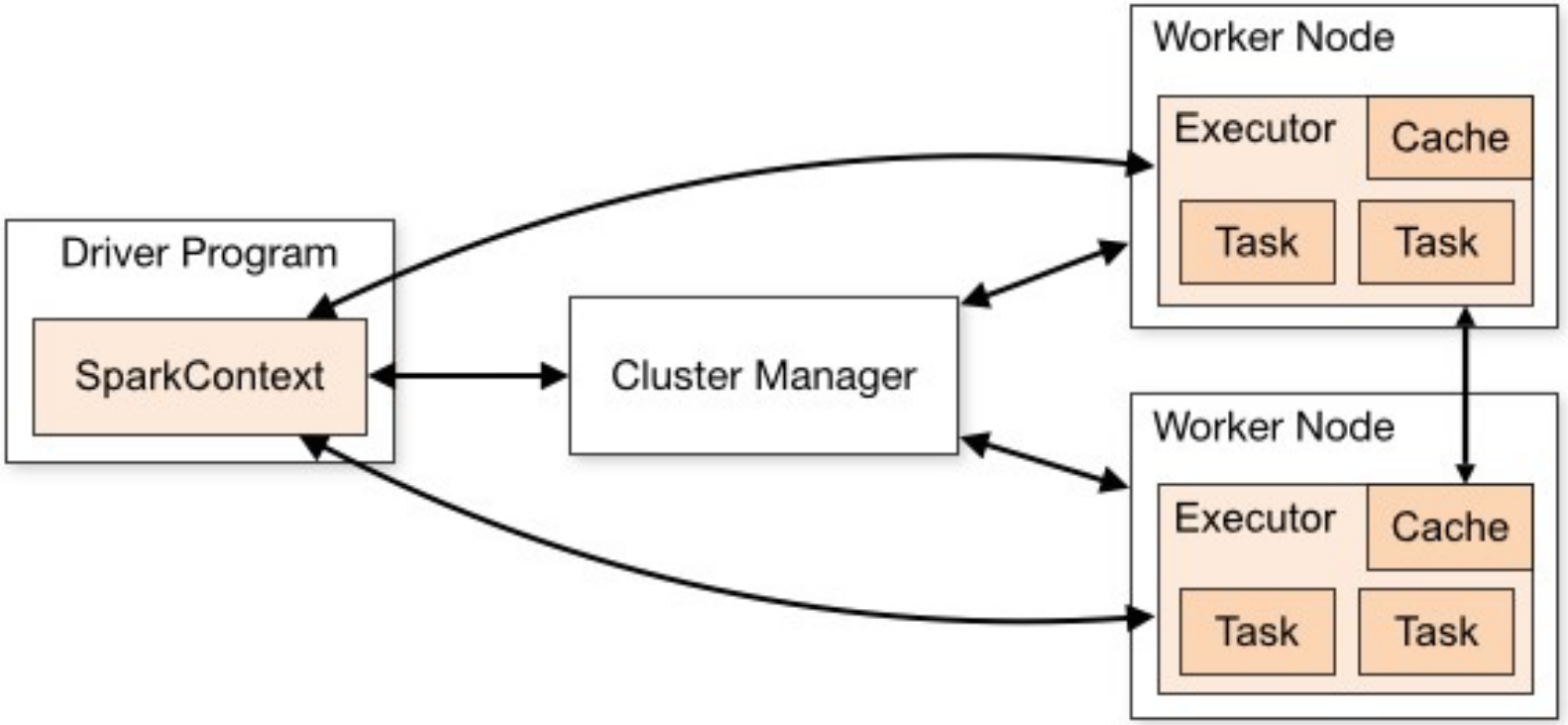
- Clúster Spark: master vs workers
- Cliente contacta con máster y envía trabajos



# Spark > Introducción

## Arquitectura en detalle

- Driver, SparkContext, Cluster Manager, Executor, Task



# Spark > Introducción

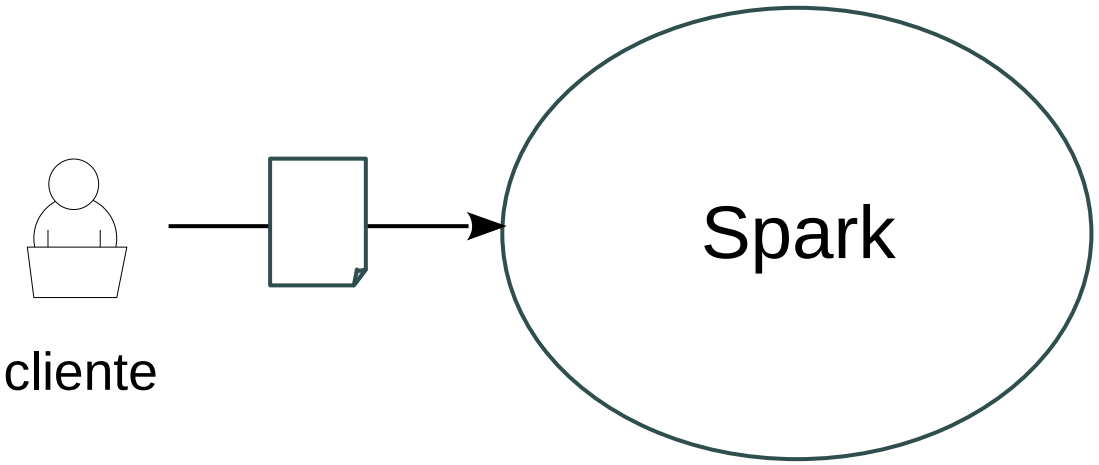
## Arquitectura en detalle

- El Driver dialoga con el Clúster Manager a través de SparkContext
- SparkContext solicita Executors al principio, y están disponibles durante toda la ejecución de la tarea
- El Executor ejecuta Tasks (threads) y mantiene datos
- El Driver se conecta con los Executors

# Spark > Introducción

## Puesta en marcha

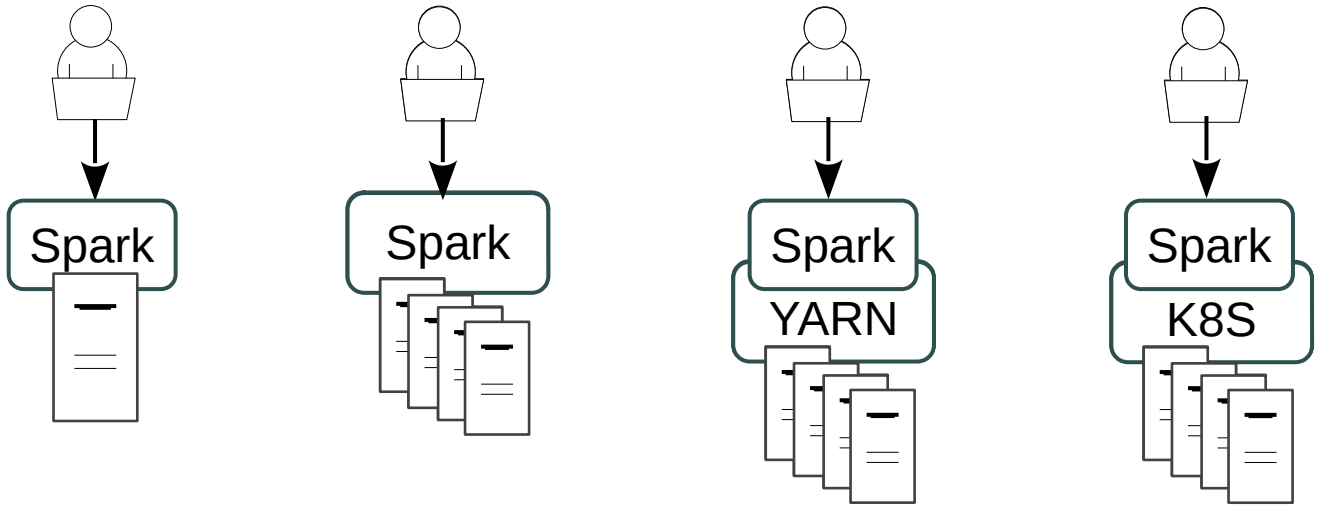
- Primero es necesario disponer de una instalación de Spark funcionando
- Entonces el cliente puede enviar trabajos a Spark



# Spark > Introducción

## Instalación de Spark

- En local (para aprender, desarrollo, pruebas) o Colab
- En un clúster standalone
- En un clúster YARN
- En un clúster Kubernetes (K8S)



# Spark > Introducción

## Instalación de Spark > En local

- https://spark.apache.org/downloads.html
- Versión 3.x pre-built for Hadoop 3.x
- > tar xvfz spark-3.x-bin-hadoop3.x.tgz
- > export SPARK_HOME=xxx

# Spark > Introducción

## Instalación de Spark > En local con Docker

- Un único contenedor

```
> docker run –d --name spark -v ./share:/data -p 8888:8888 -p 4040:4040 jupyter/pyspark-notebook
```

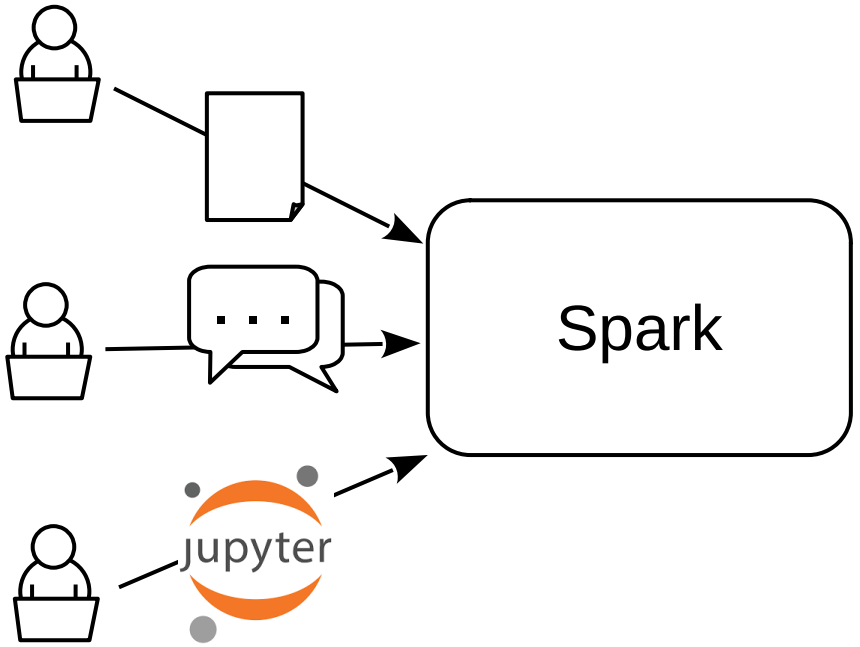
- En Google Colab (PoliformaT)
- Simulación de clúster con Docker Compose (PoliformaT)

```
> docker compose up -d
```

# Spark > Introducción

## Enviar trabajos

- En batch (spark-submit)
- Intérprete interactivo (pyspark shell)
- Jupyter (IDE interactivo)



# Spark > Introducción

## Enviar trabajos > En batch

- Con ./bin/spark-submit
- --master local [local[K]], --py-files, --xxx-memory, --xxx-cores , ...

```
> ./bin/spark-submit --master local --driver-memory 8g test.py
```

test.py

```
import pyspark
sc = pyspark.SparkContext()
data = sc.parallelize(["Hello world","Bye world"])
counts = data.flatMap(lambda line: line.split(" "))
  .map(lambda word: (word, 1)).countByKey()
print(counts)
```

# Spark > Introducción

## Enviar trabajos > Intérprete interactivo

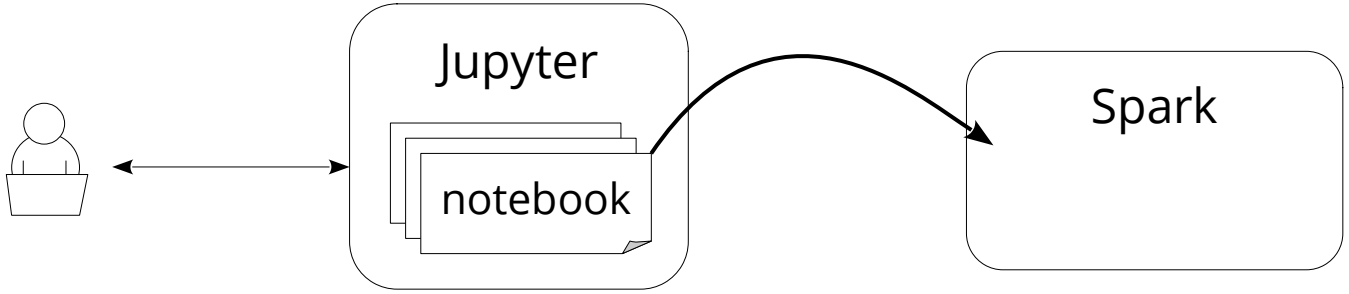
- Con ./bin/pyspark
- --master local [local[K]], --py-files, --xxx-memory, --xxx-cores, ...

```
> ./bin/pyspark --master local
>>> data = sc.parallelize(["Hello world","Bye world"])
>>> counts = data.flatMap(lambda line: line.split(" "))
  .map(lambda word: (word, 1)).countByKey()
>>> print(counts)
>>> quit()
```

# Spark > Introducción

## Enviar trabajos > Jupyter

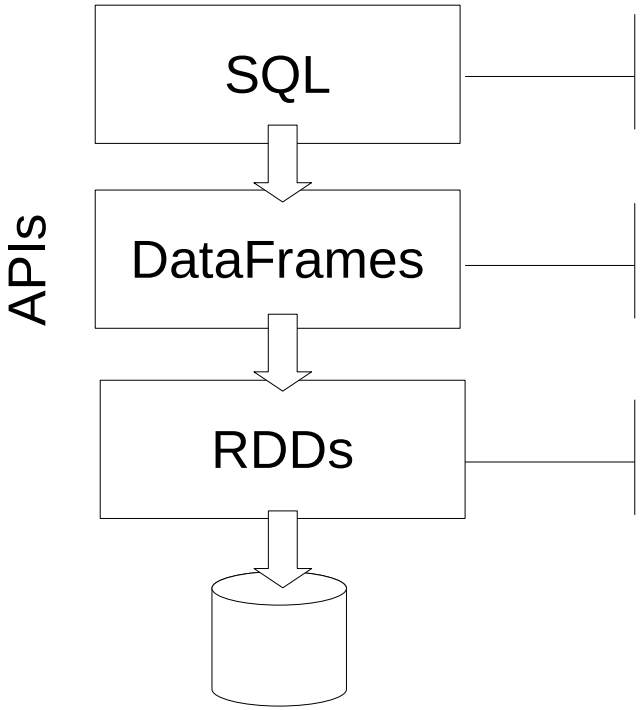
- Permite ejecutar comandos en entorno de desarrollo interactivo, a través de notebooks



# Spark > Introducción

## APIs

- Las distintas APIs suministran distintas abstracciones
- Cada abstracción de mayor nivel se basa en la anterior

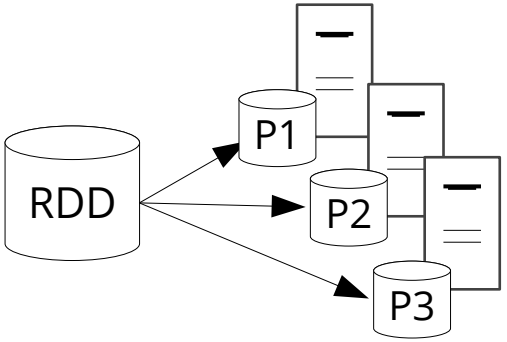



- Consulta de datos de alto nivel (SQL)
- Colección tabular de datos
- Añade estructura y optimización
- Colección de datos distribuida
- API de bajo nivel de procesamiento de datos

# Spark > RDDs

## ¿Qué es?

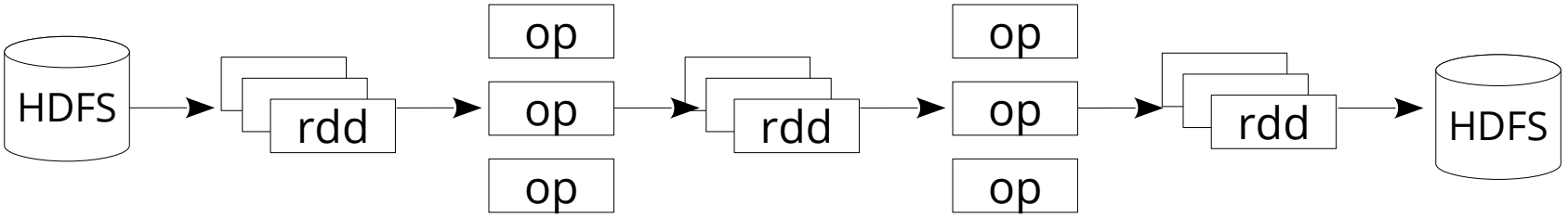
- Resilient Distributed Dataset
- Colección de datos
  - Distribuida: se particiona en múltiples nodos (en memoria)
  - Inmutable: no se puede modificar
  - Resiliente: si un nodo falla se reconstruye



# Spark > RDDs

## ¿Cómo funciona?

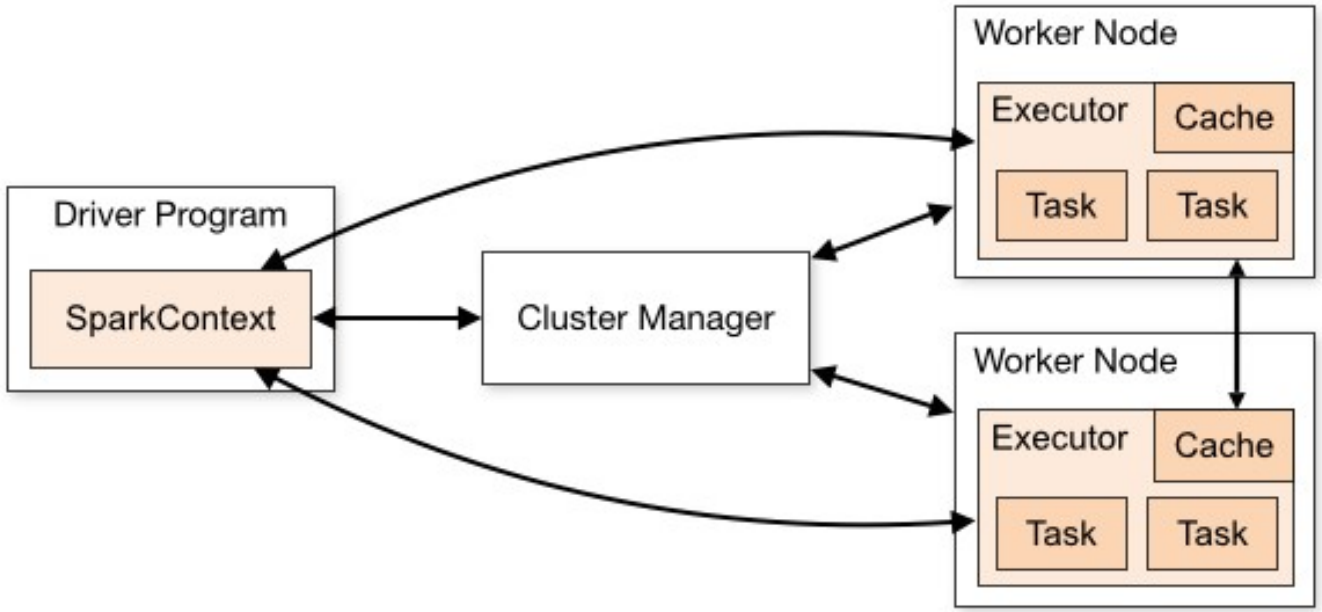
- Datos a partir de un origen (HDFS, otros), se particionan
- Cada partición se distribuye a un nodo (localidad de datos), y se mantiene en memoria
- Se aplican operaciones: en paralelo, eficientes
- Se generan nuevas particiones, en memoria
- Finalmente, se recuperan/guardan los resultados



# Spark > RDDs

## SparkContext

- Para empezar a trabajar con RDDs es necesario disponer de un contexto Spark: conexión con el clúster



# Spark > RDDs

## SparkContext

- Para empezar a trabajar con RDDs es necesario disponer de un contexto Spark: conexión con el clúster
- En PySpark ya está disponible en la variable sc; en otros clientes hay que crearlo
- El contexto se cierra al finalizar la ejecución: .stop()

```
from pyspark import SparkContext
sc = SparkContext(master='local',appName='test')
...
sc.stop()
```

# Spark > RDDs

## Creación de RDDs

- A partir de un iterable
- A partir de una fuente externa

# Spark > RDDs

## Creación de RDDs

- A partir de un iterable
  - Con sc.parallelize()
  - Los datos locales se distribuyen en particiones

```
data = [1,2,3,4,5]
rdd = sc.parallelize(data)
```

- A partir de una fuente externa

# Spark > RDDs

## Creación de RDDs

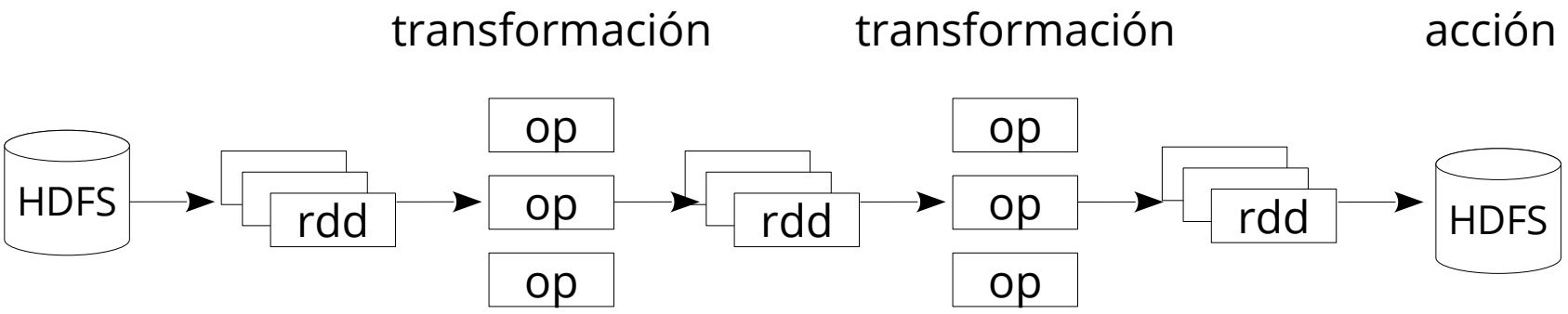
- A partir de un iterable
- A partir de una fuente externa
  - Cualquier origen soportado por Hadoop (HDFS, HBase, Amazon S3, etc.)
  - Ficheros con sc.textFile()

```
rdd = sc.textFile("file:///data.txt")
rdd = sc.textFile("/my/dir")
rdd = sc.textFile("/my/dir/*.txt")
rdd = sc.textFile("/my/dir/*.gz")
```

# Spark > RDDs

## Operaciones sobre RDDs

- Una vez se dispone de un RDD, se aplican operaciones
- Existen dos tipos de operaciones
  - Transformaciones: en cadena, generan nuevos RDDs
  - Acciones: finalizan la cadena de transformaciones

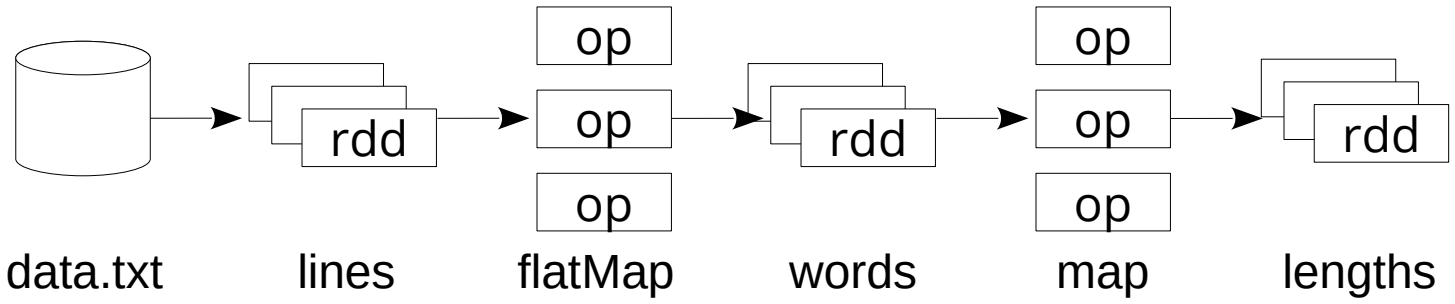


# Spark > RDDs

## Transformaciones

- A partir de una colección origen (en memoria), crea una nueva colección destino (en memoria)
- La transformación se aplica en paralelo en las distintas particiones

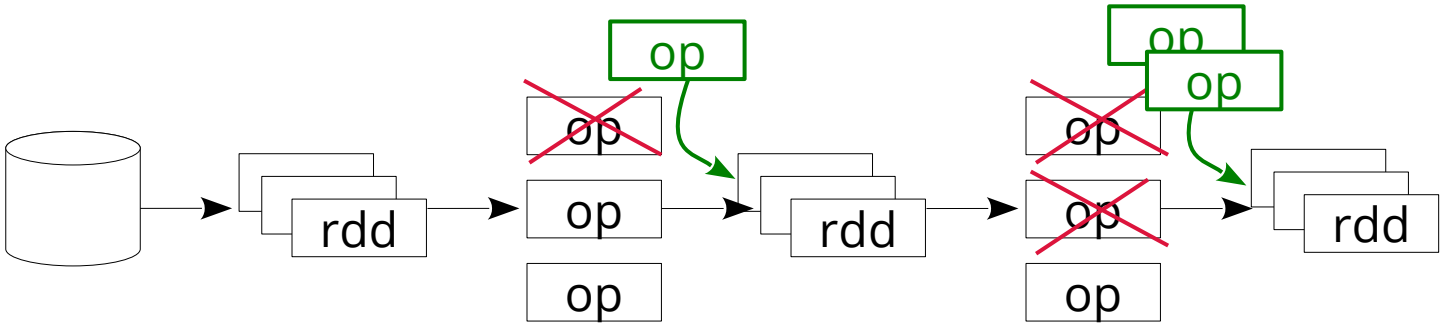
```
lines = sc.textFile("data.txt")
words = lines.flatMap(lambda line: line.split())
lengths = words.map(lambda word: len(word))
```



# Spark > RDDs

## Transformaciones

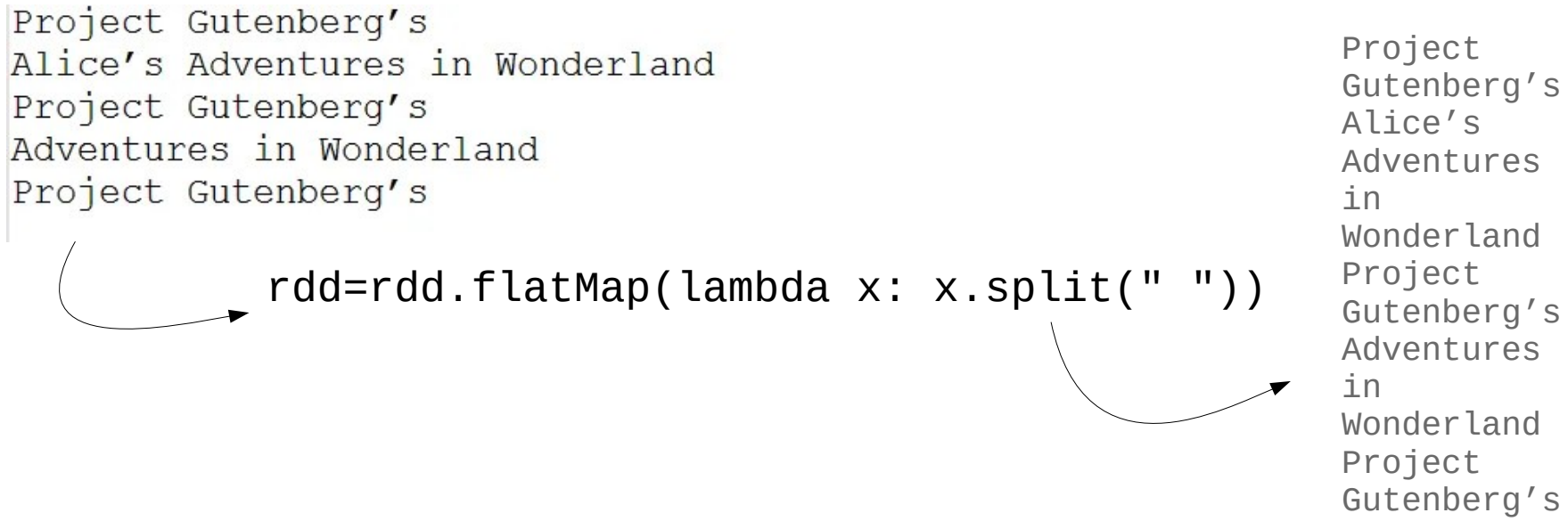
- A partir de una colección origen (en memoria), crea una nueva colección destino (en memoria)
- La transformación se aplica en paralelo en las distintas particiones
- En caso de fallo, es habitual recomputar las particiones



# Spark > RDDs

## Transformaciones

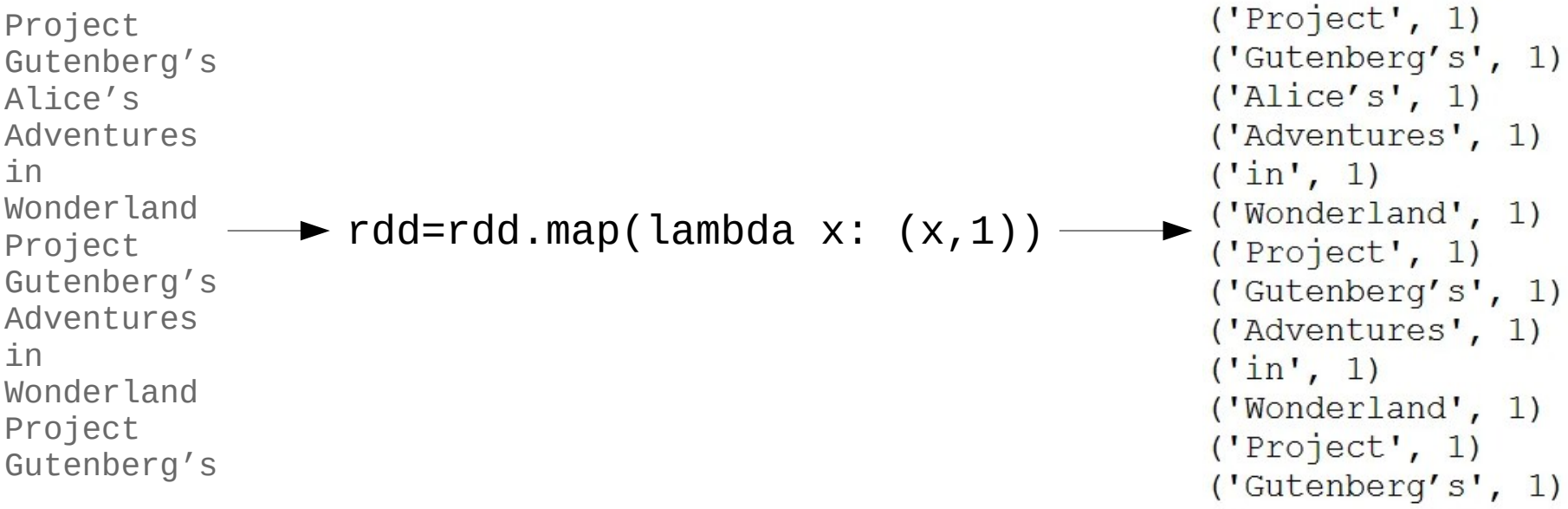
- Consultar la API
- flatMap(), map(), reduceByKey(), sortByKey(), filter(), union(), intersection(), distinct(), groupByKey(), aggregateByKey(), join(), ...



# Spark > RDDs

## Transformaciones

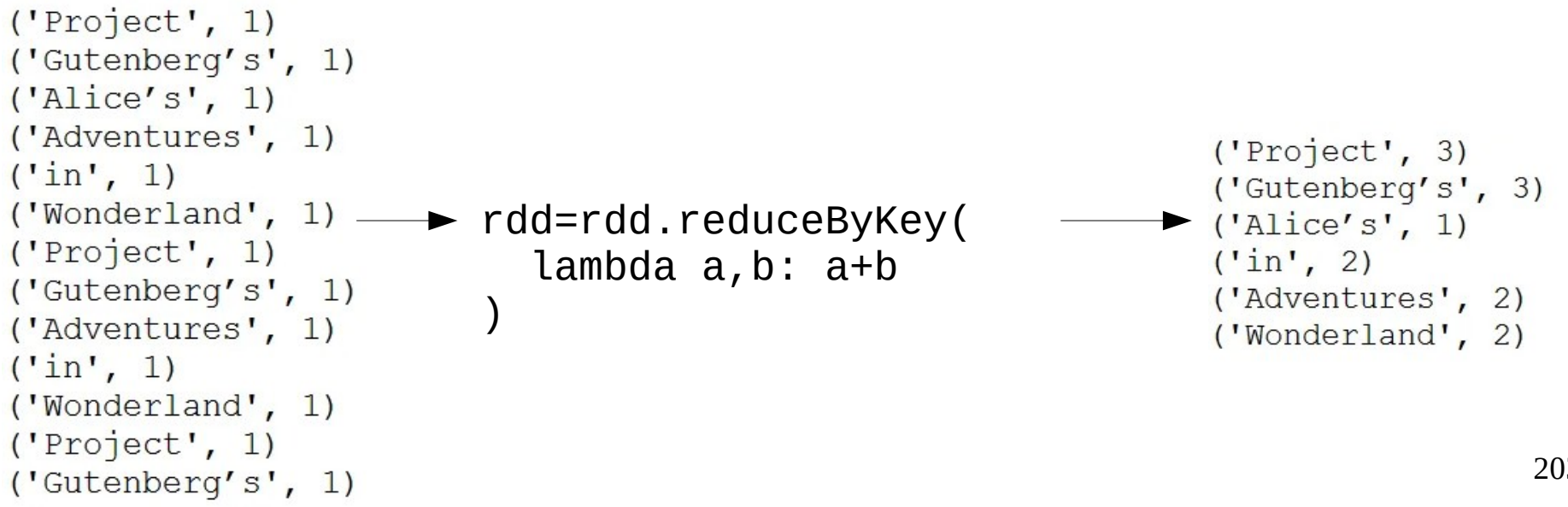
- Consultar la API
- flatMap(), map(), reduceByKey(), sortByKey(), filter(), union(), intersection(), distinct(), groupByKey(), aggregateByKey(), join(), ...



# Spark > RDDs

## Transformaciones

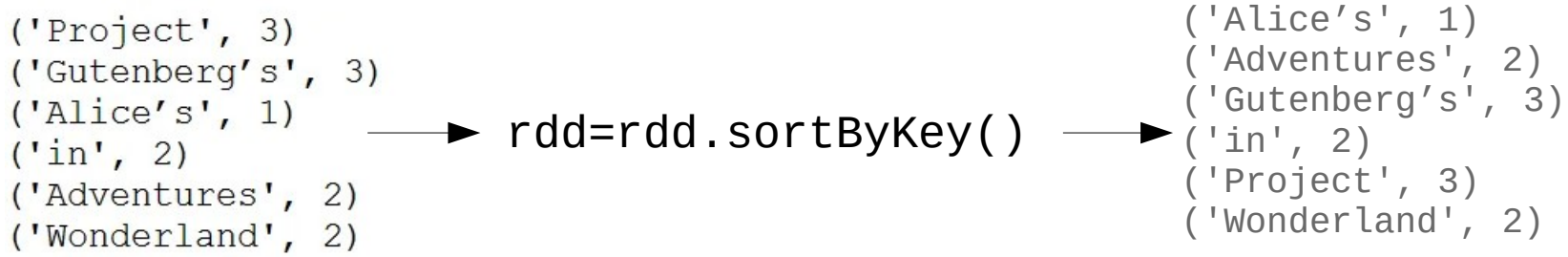
- Consultar la API
- flatMap(), map(), reduceByKey(), sortByKey(), filter(), union(), intersection(), distinct(), groupByKey(), aggregateByKey(), join(), ...



# Spark > RDDs

## Transformaciones

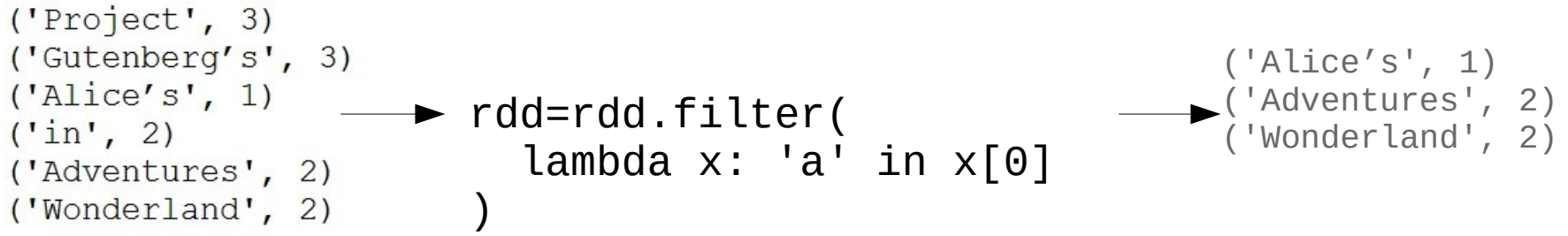
- Consultar la API
- flatMap(), map(), reduceByKey(), sortByKey(), filter(), union(), intersection(), distinct(), groupByKey(), aggregateByKey(), join(), ...



# Spark > RDDs

## Transformaciones

- Consultar la API
- flatMap(), map(), reduceByKey(), sortByKey(), filter(), union(), intersection(), distinct(), groupByKey(), aggregateByKey(), join(), ...



# Spark > RDDs

## Transformaciones

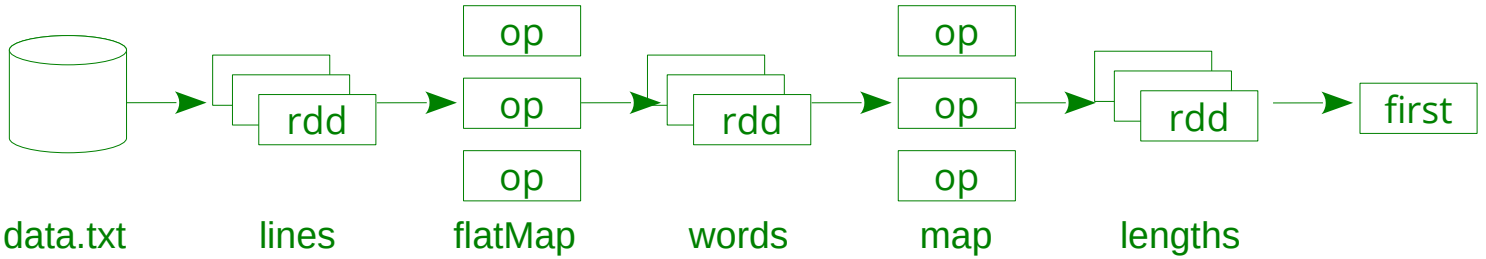
- Son perezosas: sólo se evalúan cuando es necesario, cuando se produce una acción

No hay ejecución

```
lines = sc.textFile("data.txt")
words = lines.flatMap(lambda line: line.split())
lengths = words.map(lambda word: len(word))
first = lengths.first()
print(first)
```

Necesidad de obtener un resultado

Se dispara la ejecución de las transformaciones




tiempo

# Spark > RDDs

## Transformaciones

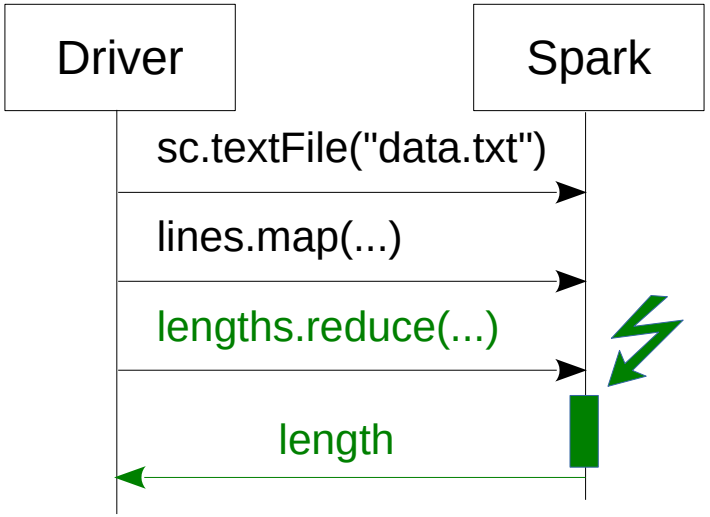
- Son perezosas: sólo se evalúan cuando es necesario, cuando se produce una acción
- Esta evaluación retardada permite analizar todo el proceso y diseñar un plan de ejecución
- El plan de ejecución permite incorporar diversas optimizaciones (reordenar evaluaciones, evitar cálculos duplicados, etc.)

# Spark > RDDs

## Acciones

- Disparan la ejecución de transformaciones

```
lines = sc.textFile("data.txt")
lengths = lines.map(lambda line: line.length)
length = lengths.reduce(lambda a, b: a + b)
```

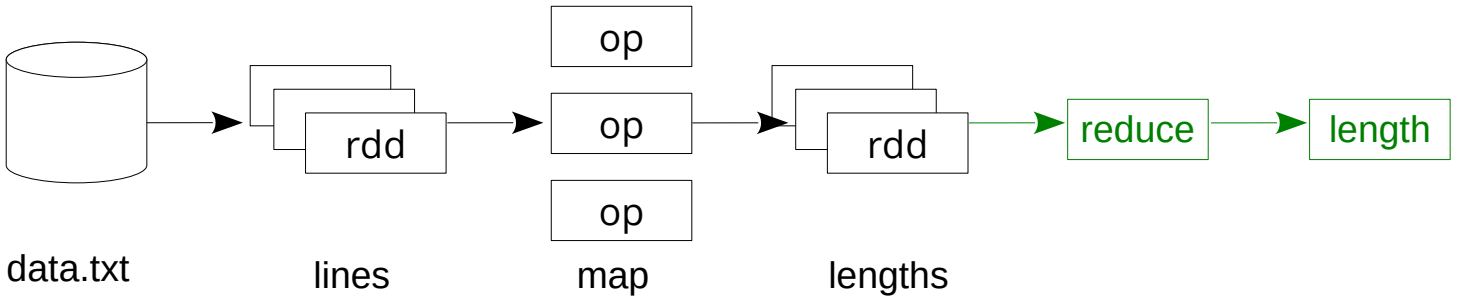


# Spark > RDDs

## Acciones

- Disparan la ejecución de transformaciones
- Computan una colección origen y devuelven un valor al Driver

```
lines = sc.textFile("data.txt")
lengths = lines.map(lambda line: line.length)
length = lengths.reduce(lambda a, b: a + b)
```



# Spark > RDDs

## Acciones

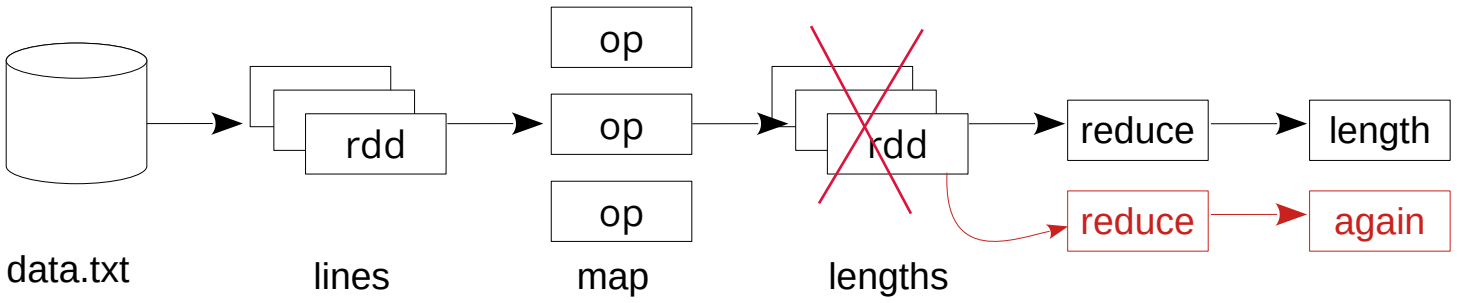
- Consultar la API
- reduce(func), collect(), count(), first(), take(n), saveAsTextFile(path), countByKey(), foreach(func), ...

# Spark > RDDs

## Persistencia

- Los RDDs por defecto se cachean en memoria
- Una vez computada la operación, los RDDs intermedios se desechan

```
lines = sc.textFile("data.txt")
lengths = lines.map(lambda line: line.length)
length = lengths.reduce(lambda a, b: a + b)
again = lengths.reduce(lambda a, b: a + b)
```

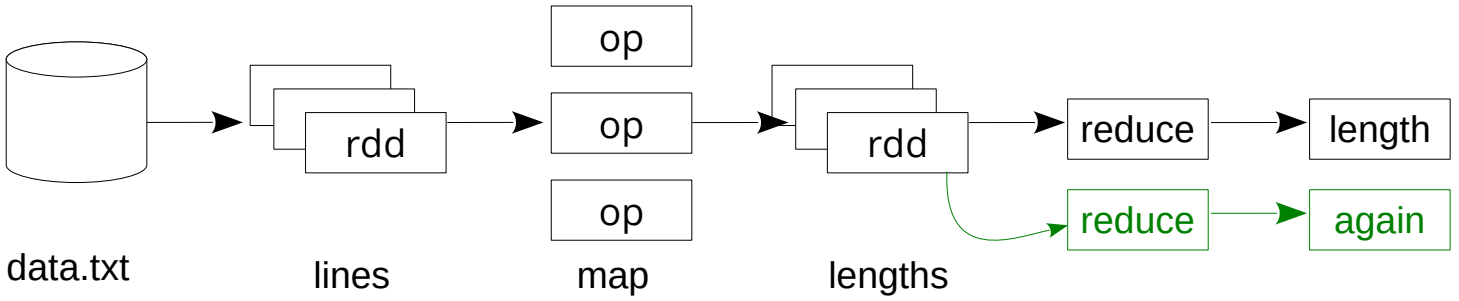


# Spark > RDDs

## Persistencia

- Se pueden mantener en el clúster con rdd.persist()/ rdd.cache()

```
lines = sc.textFile("data.txt")
lengths = lines.map(lambda line: line.length).persist()
length = lengths.reduce(lambda a, b: a + b)
again = lengths.reduce(lambda a, b: a + b)
```



# Spark > RDDs

## Ejercicio 5

- A partir del dataset “city_accesses.csv”, obtener qué día de la semana y a qué hora se corre más usando Spark

```
"2020-01-01 00:56:27.537","00:56:28","Acercamiento",
"72","AF_Entrada_Cocentaina"
```

# Spark > RDDs

## Ejercicio 6

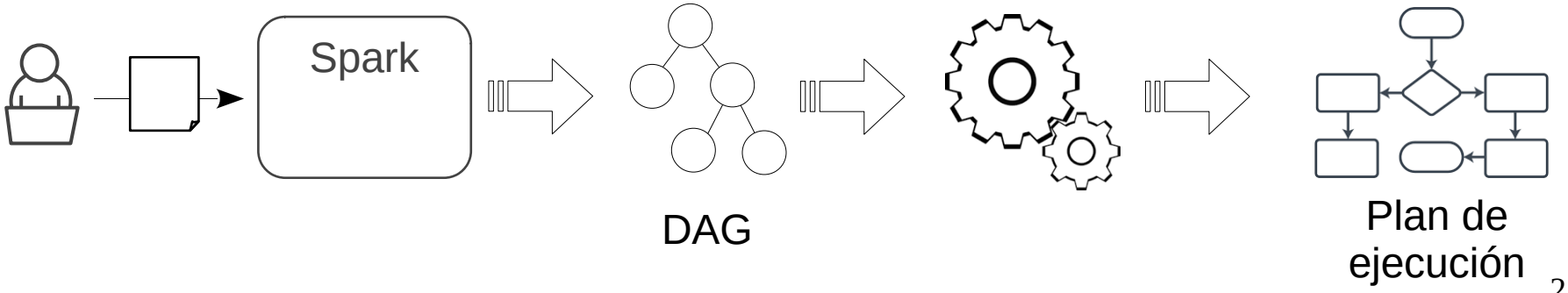
- A partir del dataset “climate.json”, obtener la temperatura media por mes usando Spark


```
[
...
{
"fecha" : "2000-01-01", "indicativo" : "8025",  "nombre" : "ALACANT/ALICANTE",
"provincia" : "ALICANTE",  "altitud" : "81", "tmed" : "9,0", "prec" : "0,0", "tmin" : "2,4",
"horatmin" : "08:00", "tmax" : "15,7", "horatmax" : "12:30", "dir" : "16", "velmedia" : "1,1",
"racha" : "2,5", "horaracha" : "15:33",  "sol" : "8,4", "presMax" : "1019,4", "horaPresMax" :
"11", "presMin" : "1017,5", "horaPresMin" : "06"
}
...
]
```

# Spark > RDDs

## Planificación de trabajos

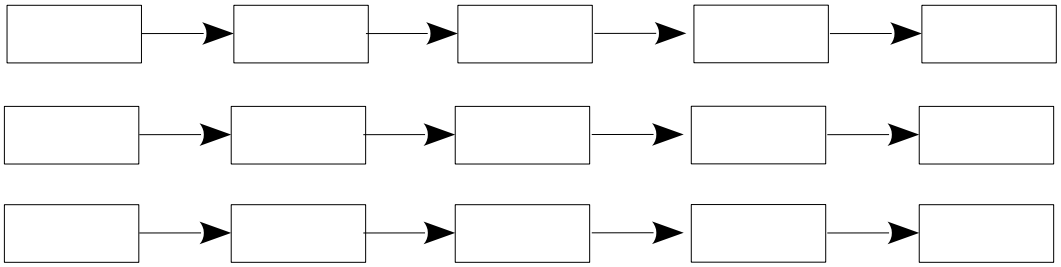
- Spark dispone de un planificador de trabajos: Spark Scheduler
- Transforma los trabajos en un grafo (DAG-Direct Acyclic Graph) de operaciones
- Analiza, reagrupa, reordena, optimiza el grafo para obtener un plan de ejecución



# Spark > RDDs

## Trabajos

- Cada secuencia de transformaciones que se ejecuta como consecuencia de una acción es un trabajo (job)



# Spark > RDDs

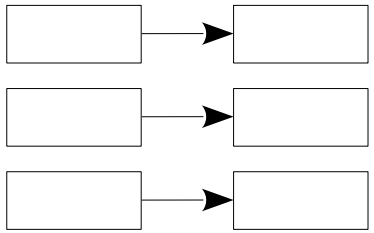
## Trabajos > Transformaciones

- Hay dos tipos de transformaciones
  - Narrow
  - Wide

# Spark > RDDs

## Trabajos > Transformaciones

- Hay dos tipos de transformaciones
  - Narrow (non-shuffle):
    - Cada partición de salida puede ser computada a partir de una única partición de entrada
    - No requiere comunicación entre nodos (e.g. map(), filter())

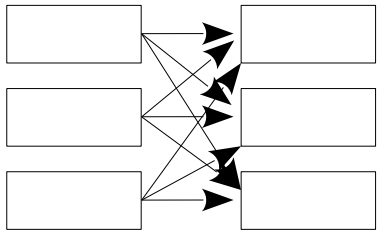


rdd op rdd

# Spark > RDDs

## Trabajos > Transformaciones

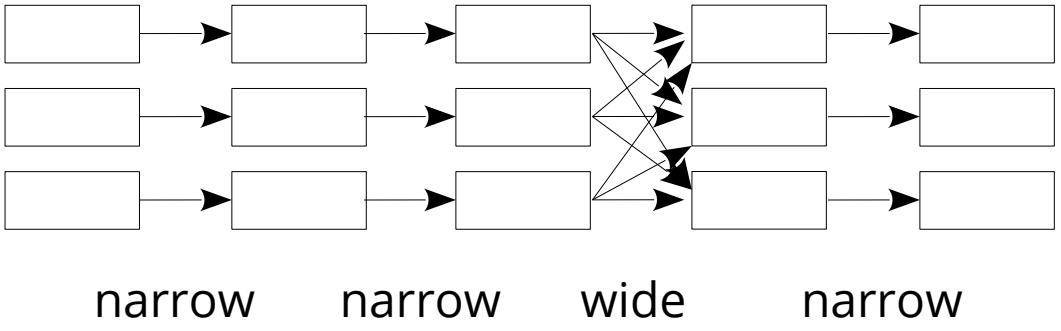
- Hay dos tipos de transformaciones
  - Wide (shuffle):
    - Cada partición de salida se computa a partir de varias particiones de entrada
    - Requiere comunicación entre nodos, redistribución de datos: partition shuffling (e.g. groupBy(), sortBy()):



# Spark > RDDs

## Trabajos > Transformaciones

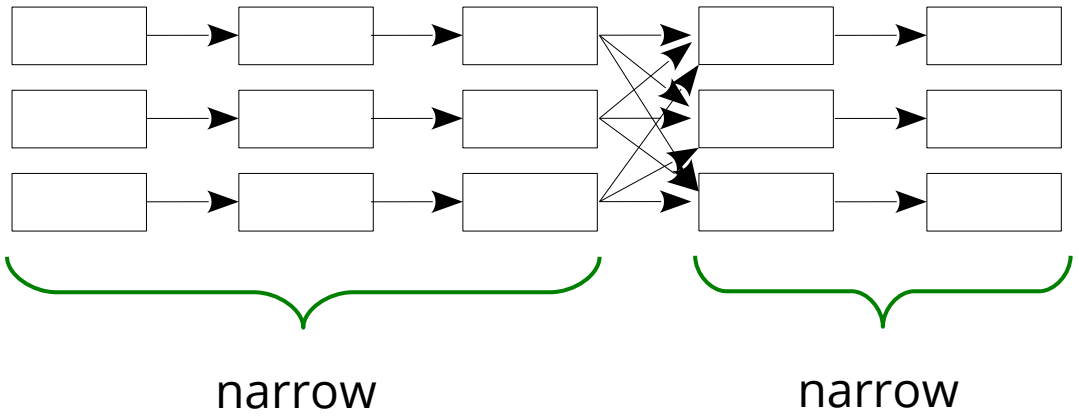
- Transformaciones narrow y wide se suceden en un flujo de trabajo



# Spark > RDDs

## Trabajos > Transformaciones

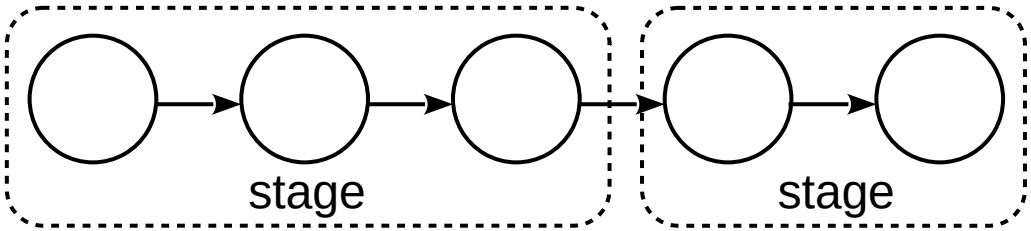
- Transformaciones narrow y wide se suceden en un flujo de trabajo
- Las transformaciones narrow se pueden ejecutar en un mismo nodo, y sin esperar a que acabe la anterior



# Spark > RDDs

## Trabajos > Transformaciones

- Transformaciones narrow y wide se suceden en un flujo de trabajo
- Las transformaciones narrow se pueden ejecutar en un mismo nodo, y sin esperar a que acabe la anterior
- Es posible definir un grafo (DAG), identificando grupos de operaciones narrow (stages), separadas por operaciones wide



# Spark > RDDs

## Trabajos > DAG

- Cada nodo identifica un RDD

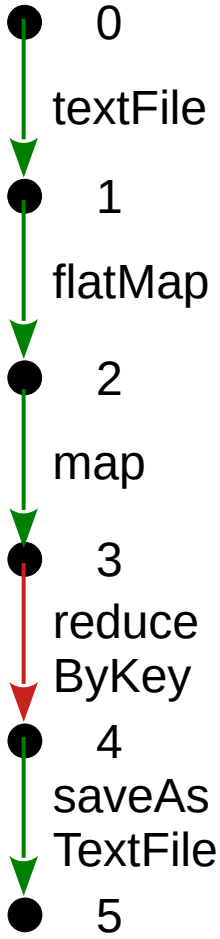
```
(0) sc.textFile("/data/el_quijote.txt") (1)
 (1) .flatMap(lambda line: line.split(" ")) (2)
 (2) .map(lambda word: (word, 1)) (3)
 (3) .reduceByKey(lambda a, b: a + b) (4)
 (4) .saveAsTextFile("/data/count")      (5)
```

# Spark > RDDs

## Trabajos > DAG

- Cada nodo identifica un RDD
- Cada arista identifica una operación sobre el RDD: non-shuffle (narrow) vs shuffle (wide)

```
(0) sc.textFile("/data/el_quijote.txt") (1)
 (1) .flatMap(lambda line: line.split(" ")) (2)
 (2) .map(lambda word: (word, 1)) (3)
 (3) .reduceByKey(lambda a, b: a + b) (4)
 (4) .saveAsTextFile("/data/count")      (5)
```

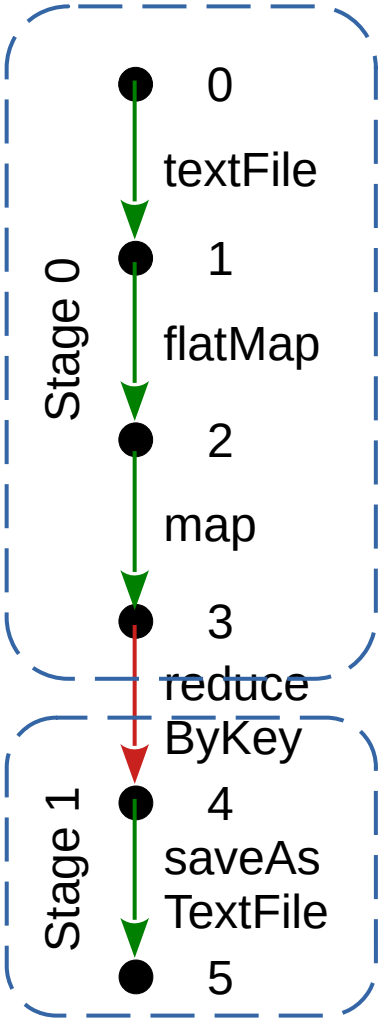


# Spark > RDDs

## Trabajos > DAG

- Cada nodo identifica un RDD
- Cada arista identifica una operación sobre el RDD: non-shuffle (narrow) vs shuffle (wide)
- Los nodos se agrupan en etapas cuyos límites son las operaciones shuffle

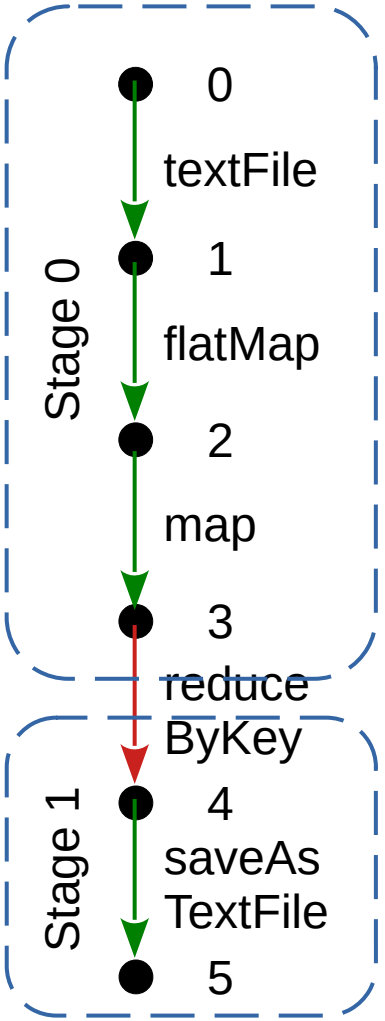
```
(0) sc.textFile("/data/el_quijote.txt") (1)
 (1) .flatMap(lambda line: line.split(" ")) (2)
 (2) .map(lambda word: (word, 1)) (3)
 (3) .reduceByKey(lambda a, b: a + b) (4)
 (4) .saveAsTextFile("/data/count")      (5)
```



# Spark > RDDs

## Trabajos > DAG

- Las operaciones incluidas en una misma etapa no requieren intercambio de datos entre nodos
- Se ejecutan en el mismo nodo y se efectúa pipelining (la salida de una operación se va pasando de manera contínua como entrada de la siguiente)
- Distintas etapas implican intercambio de datos entre nodos



# Spark > RDDs

## Problemática

- La naturaleza de los datos procesados por Spark es desconocida: Spark (de)serializa objetos Java en memoria
- La transformación que se aplica a los datos en un RDD es arbitraria y desconocida para Spark (funciones lambda soportadas por map, filter, reduce, etc.)
- Spark tiene poca información para optimizar las estructuras de datos y las operaciones: los trata como cajas negras

# Spark > Spark SQL

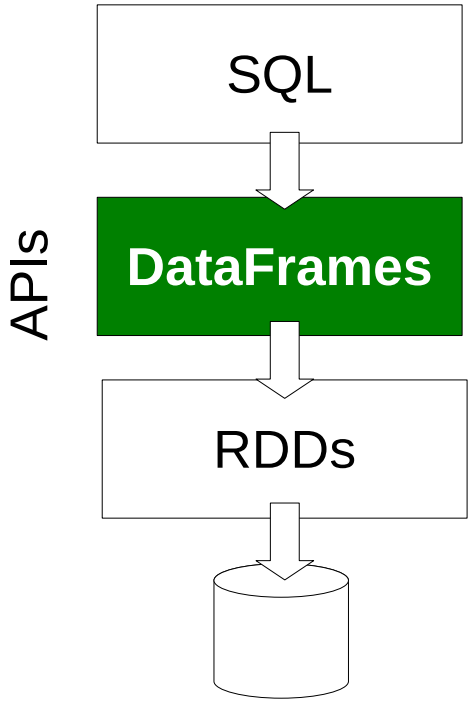
## SparkSQL

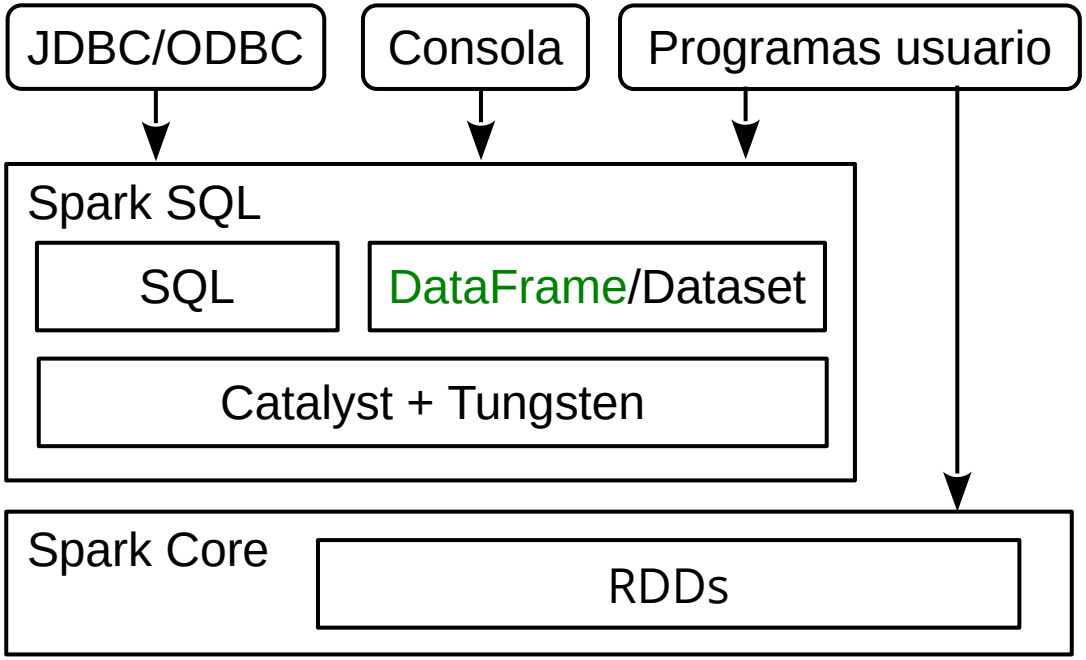
- La solución pasa por añadir estructura: definir esquemas y tipos de datos, y detallar las operaciones a aplicar
- De este modo, Spark puede optimizar el almacenamiento y transferencia de datos, así como compactar, reordenar operaciones
- Spark SQL representa las APIs estructuradas de Spark: DataFrames, Datasets, SQL

# Spark > Spark SQL

## DataFrames

- API de alto nivel (relacional)
- A partir de los RDDs, aportan estructura y optimización



# Spark > Spark SQL

## DataFrames > Estructura

- Modelo de datos tabular, inspirado por Pandas
- Cada columna tiene un tipo de datos
- Soportan operaciones relacionales de alto nivel (e.g. select, union, join, …)

Columnas

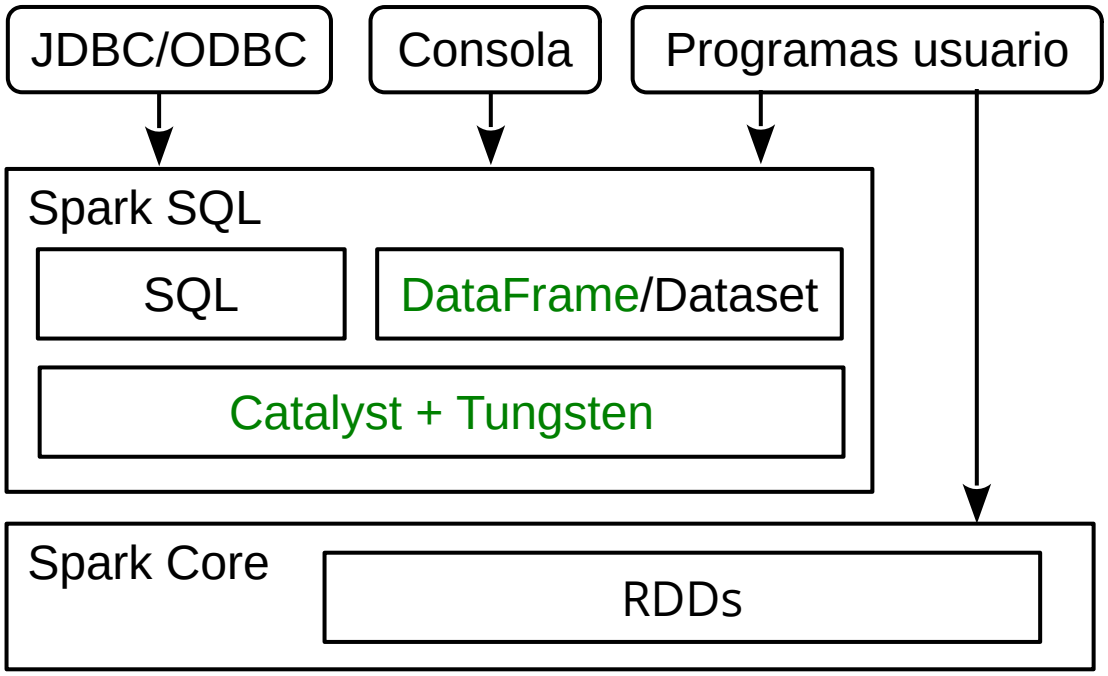
| str | int | str |
|---|---|---|
| **col1** | **col2** | **col3** |
| Joe | 1.3 | A |
| Nat | 0.4 | F |
| Harry | 4.3 | C |
| Sam | -1.6 | D |

Filas

# Spark > Spark SQL

## DataFrames > Optimización

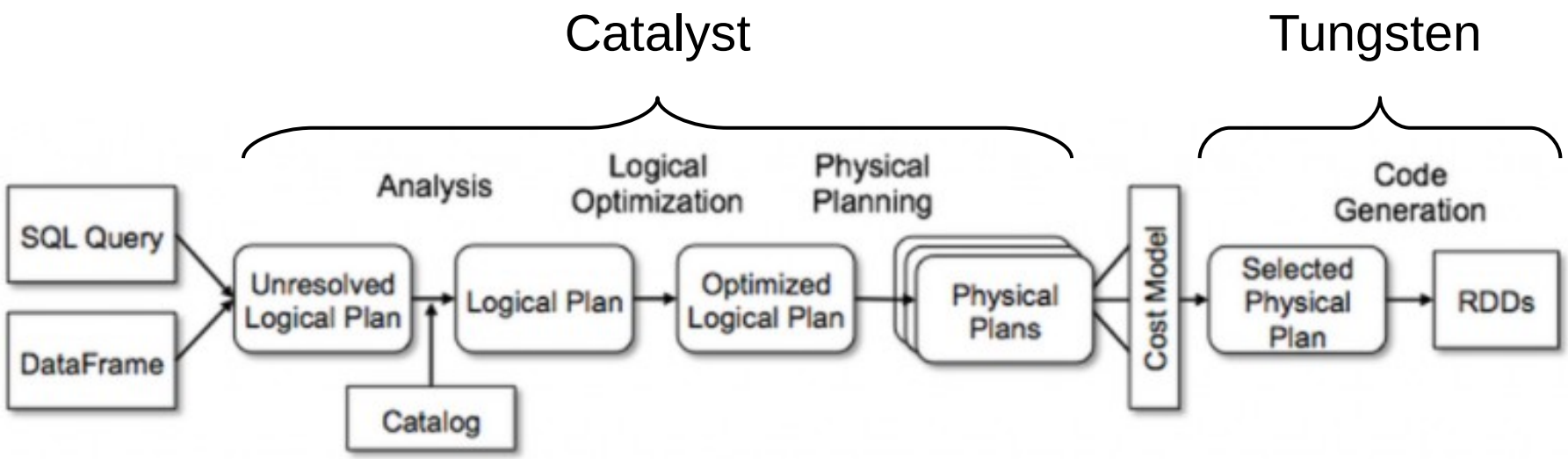
- Los trabajos son optimizados por Catalyst+Tungsten



# Spark > Spark SQL

## DataFrames > Optimización

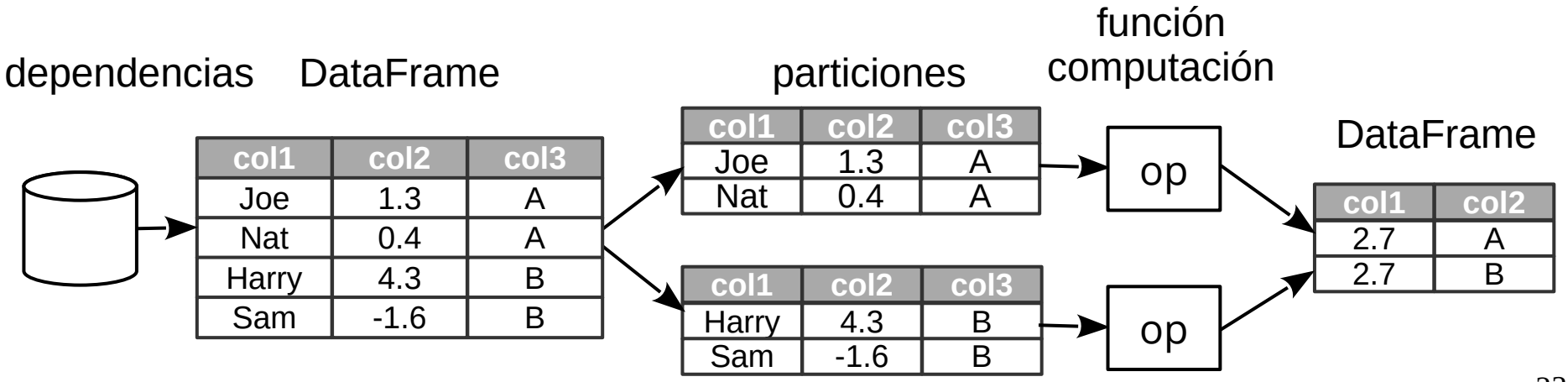
- Los trabajos son optimizados por Catalyst+Tungsten
- Catalyst analiza las operaciones y elabora un plan de ejecución optimizado; Tungsten genera el código final



# Spark > Spark SQL

## DataFrames > Optimización

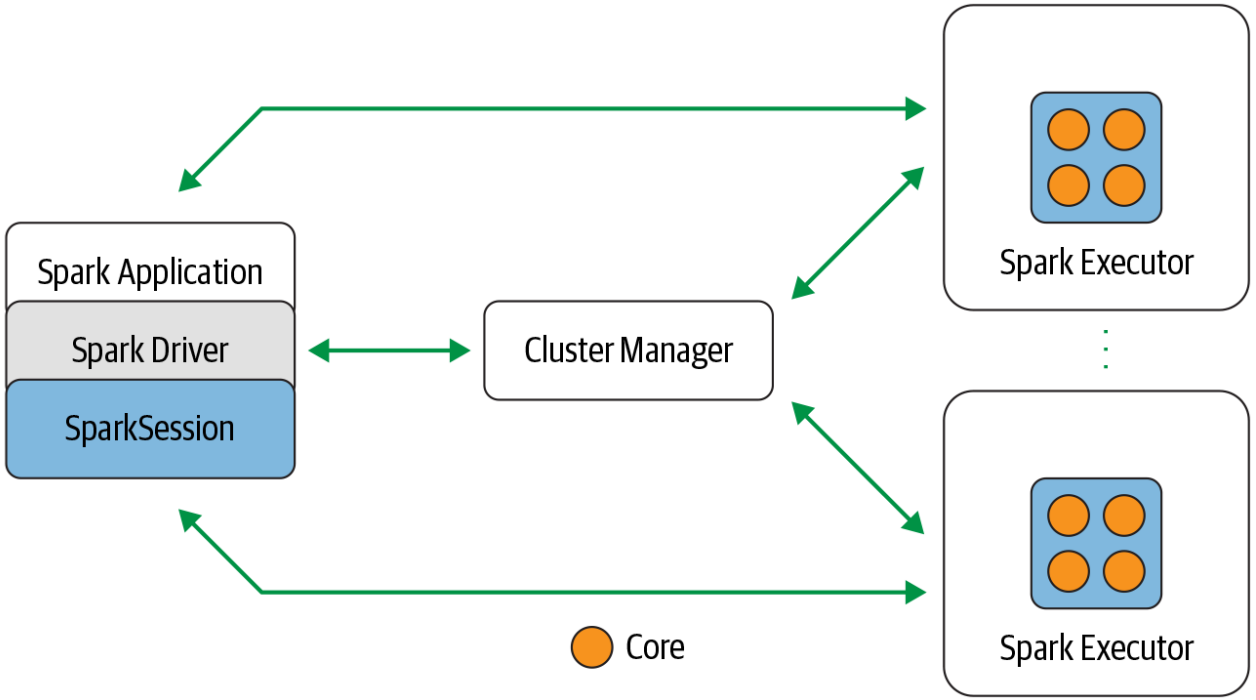
- Los trabajos son optimizados por Catalyst+Tungsten
- Catalyst analiza las operaciones y elabora un plan de ejecución optimizado
- Al final, se traduce a operaciones sobre RDDs (dependencias, particiones, función de computación)



# Spark > Spark SQL

## DataFrames > SparkSession

- Sustituye a SparkContext como punto de entrada para trabajar con Spark: modela una conexión con el clúster



# Spark > Spark SQL

## DataFrames > SparkSession

- Sustituye a SparkContext como punto de entrada para trabajar con Spark: modela una conexión con el clúster
- En PySpark ya está disponible en la variable spark; en otros clientes hay que crearla
- Contiene SparkContext en el atributo .sparkContext
- La sesión se cierra al finalizar el trabajo: .stop()

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local").appName("test")
       .config(key, value).getOrCreate()
...
spark.stop()
```

# Spark > Spark SQL

## DataFrames > Creación

- A partir de datos en memoria: spark.createDataFrame()
- A partir de una fuente externa con spark.read
- Tienen un esquema en .schema; Spark intenta inferirlo

```
df = spark.createDataFrame([("John",10),("Mary", 15)])
df = spark.read.option(key,value).csv("/data/datos.csv")
df = spark.read.option(key,value).json("/data/datos.json")
print(df.columns), print(df.schema), df.printSchema()

df = spark.createDataFrame([("John",10),("Mary", 15)],
     schema="name STRING, age INT")
df = spark.read.csv("/data/datos.csv", "name STRING, age INT")
df.show(), df.describe().show(), df.head(), df.tail()
```

# Spark > Spark SQL

## DataFrames > Operaciones

- Como con RDDs, existen dos tipos de operaciones: transformaciones y acciones
- Transformaciones: generan un nuevo DataFrame, son perezosas
- Acciones: disparan la ejecución de las transformaciones

# Spark > Spark SQL

## DataFrames > Transformaciones

- Proyecciones: .select()

```
> df.select("*"), df.select("name", df.name, (df.age+10))
```

- Filtrados: .filter(), .where()

```
> df.filter("age > 10"), df.filter(df.age > 10)
```

- Manipulación de columnas: .withColumn(), .drop()

```
> df.withColumn("age", df.age+10).drop("age")
```

- Operaciones conjuntos: .union(), .intersect(), .join()

```
> df.union(df2), df.intersect(df3), df.join(df4, on="id")
```

# Spark > Spark SQL

## DataFrames > Transformaciones

- Distintos, ordenar: .distinct(), .sort(),

```
> df.distinct(), df.sort("name")
```

- Agrupar (devuelve un grupo): .groupBy()

```
> df.groupBy("name")
```

- Agregar en un grupo: .count(), .max(), .avg(), ...

```
> df.groupBy("name").avg("age")
```

# Spark > Spark SQL

## DataFrames > Acciones

- Disparan la ejecución de las transformaciones
- Recuperar datos desdecliente: .show(), .collect(), .head(), first(), .tail(), .take(), .count()
- Escribir datos en disco o almacén externo

```
df = spark.read.csv("/data/datos.csv")
df.head(10), df.take(10), df.count()
df.groupBy("name").avg("age").show()
lst = df.groupBy("name").avg("age").collect()
print(lst)
```

# Spark > Spark SQL

## DataFrames vs RDDs

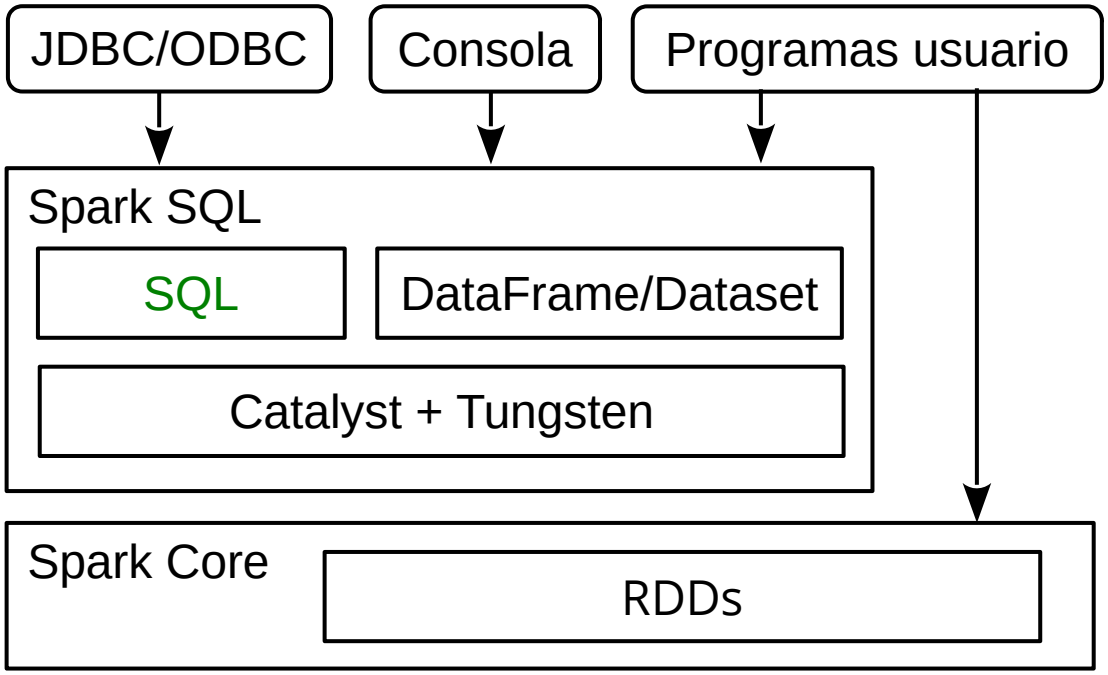
- Al final todo son RDDs
- DataFrame y RDDs son APIs compatibles

```
df = spark.read.csv("/data/datos.csv")

rdd = df.rdd        # df to rdd
rdd = rdd.map(lambda row: (row.date, row.speed))
print(rdd.collect())

df = rdd.toDF()     # rdd to df
df = spark.createDataFrame(rdd, schema="date STRING, speed INT")
df.show()
```

# Spark > Spark SQL

## SQL

- El motor Spark SQL soporta las APIs SQL y DataFrame



# Spark > Spark SQL

## SQL

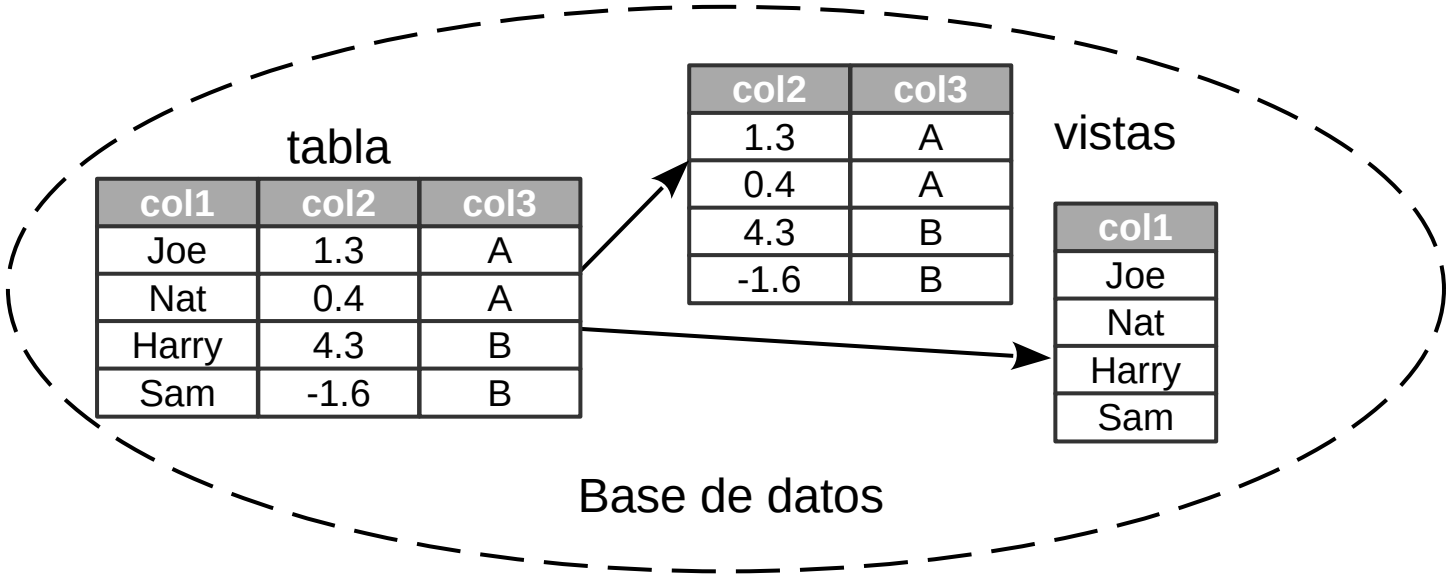
- El motor Spark SQL soporta las APIs SQL y DataFrame
- SQL es una API de alto nivel disponible en spark.sql()

```
df.select("date","type","speed")
  .where(df.speed > 50)
  .orderBy("speed", asceding=True)

df.sql("SELECT date, type, speed FROM table
        WHERE speed > 50 ORDER BY speed ASC")
```

# Spark > Spark SQL

## SQL

- El motor Spark SQL soporta las APIs SQL y DataFrame
- SQL es una API de alto nivel disponible en spark.sql()
- SQL trabaja sobre una base de datos, con tablas y vistas en memoria/disco



# Spark > Spark SQL

## SQL > Base de datos

- Creación de base de datos; por defecto existe “default”

```
spark = SparkSession.builder.master('local').appName('test')
    .config('spark.sql.catalogImplementation','hive')
    .getOrCreate()

spark.sql("CREATE DATABASE IF NOT EXISTS db")
spark.sql("USE db")

spark.sql("SHOW DATABASES").show()
dbs = spark.catalog.listDatabases()

spark.sql("DROP DATABASE db")
```

# Spark > Spark SQL

## SQL > Tablas

- Creación de tabla vacía

```
spark.sql("CREATE TABLE IF NOT EXISTS mytable (date STRING, type
STRING, speed INT")
spark.sql("SHOW TABLES").show()
```

- Creación de tabla a partir de DataFrame

```
df = spark.read.csv("/data/datos.csv",
     schema="date STRING, type STRING, speed INT")
df.write.saveAsTable("mytable", mode="overwrite")
spark.sql("SHOW TABLES").show()
tbls = spark.catalog.listTables()
```

- Eliminación de tabla

```
spark.sql("DROP TABLE mytable")
```

# Spark > Spark SQL

## SQL > Vistas

- A partir de un DataFrame
- Son temporales: sólo visibles en la sesión en curso
- Vista global vs local: visible en todas las sesiones de la aplicación en curso

```
df = spark.read.csv("/data/datos.csv",
     schema="date STRING, type STRING, speed INT")
df.createOrReplaceTempView("myview")
df.createOrReplaceGlobalTemptView("myview")
spark.sql("SHOW VIEWS").show()
spark.sql("DROP VIEW myview")
```

# Spark > Spark SQL

## SQL > Consultas

- Utilizan Ansi SQL, sobre tablas/vistas
- Devuelven un DataFrame

```
df = spark.sql("SELECT * FROM mytable WHERE speed > 50")
df = spark.sql("SELECT * FROM myview WHERE speed > 50")
```