

# Fuentes de Datos

José Incera

Abril – junio 2020

# Contenido

- Introducción. Conceptos Big Data
  - Hadoop MapReduce
- Datos no estructurados - texto
  - Principios básicos de análisis
- Internet de las cosas
  - Redes de sensores. Arquitecturas M2M
- Procesamiento de datos en tiempo (casi)real
- Redes
  - Conceptos básicos. Grafos
  - Redes sociales
- Datos geo-referenciados
- (¿Audio e imágenes?)

# Evaluación

- 25% Examen parcial
- 25% Examen final
- 30% Prácticas, controles
- 20% Proyecto final

# Ayúdame a responder:

¿Qué es Big Data, qué es BI, qué es Analítica?

¿Qué tan grande es un set de datos para que sea “big”?

¿De dónde vienen los datos?

¿Para qué se usa Big Data/Analytics en las organizaciones?

¿Cómo se integran los datos en (los sistemas de) las organizaciones?

¿Cuál es el potencial de Big Data en las organizaciones?

¿Qué ventajas, oportunidades, amenazas y desafíos puede tener Big Data y Analytics?

# Explosión de información

- Estimación datos digitales globales en 2012:  
1.6 Zettabytes ( $10^{21}$ )  
  
= toda la población mundial enviando tweets  
durante 140 años  
  
= 47,500 años de HDTV

# Ejemplo de casos de uso

	Fuente de Datos	Operaciones alta frecuencia	Operaciones baja frecuencia
Monitoreo Trans. Financieras	Mercados de capital	Escribe/indexa todas las transacciones, almacena data tickets	Muestra riesgo consolidado entre operadores
Gestión CDR telecomunicaciones	Solicita inicio de llamada	Autorización en tiempo real	Análisis y detección de fraudes
Analítica sitios web, detección fraudes	Solicitudes HTTP entrantes	Análisis, alerta y registro de visitantes	Análisis de patrones de tráfico
Juegos en línea, micro transacc.	Juegos en línea	Rank scores: <ul style="list-style-type: none"> <li>• Intervalos definidos</li> <li>• Top ten jugadores</li> </ul>	Diversos tipos de búsqueda
Serv. Intercambio, Ad. Digital	Sistemas anuncios en tiempo real	Match factor de forma, criterio de colocación, apuesta/consulta	Reporta desempeño Ads. de flujos de info
Serv. Basados en localización	Sensor localización dispositivo móvil	Actualización de coordenadas, QoS, transacciones	Análisis de transacciones

# Niveles de madurez en la absorción BD&A



## Innovadores

## Practicantes

## Amenazados

Cultura analítica desde la Alta Dirección

Trabajando para convertirse en *data driven*

Decisiones basadas más en experiencia y gestión que en análisis

Principal uso: estratégico

Principal uso: Operativo

Principal uso: reducción de costos

Datos tienen un alto valor

Tiene datos "suficientemente buenos"

Datos de mala calidad y difícil acceso

Sólidas habilidades para gestión y análisis de datos

Más información que analítica para toma de decisiones

No tiene habilidades para gestión ni análisis de datos

# Big Data y Analítica

- 95% de las empresas reconoce que la analítica es sumamente valiosa
  - Sólo 31% es capaz de medir ese valor y considera que cumple o excede sus expectativas
- La mayoría de las iniciativas sale de las unidades de negocio
  - Es imperativo tener una estrategia transversal, que comparta datos, visiones y hallazgos
- 64% de las empresas a nivel mundial ha invertido en iniciativas de BD&A
  - En América Latina, 30%
- México:
  - 66% reconoce que la recopilación de datos mejora la toma de decisiones, pero
  - 55% no considera a Big Data como estratégico



# Proyectos BD&A

	Ejemplo de proyecto	Objetivos globales	Estructura de proyecto	Competencias necesarias	Indicadores de éxito
TI Clásico	Instala ERP Optimiza proceso de negocio	Incrementa eficiencia Disminuye costos	Gestión clásica: define entregables, fechas, planes de desarrollo, recursos	PMP con conocimientos TI, matemáticas Conocimientos de la U. de negocio	Proyecto terminado en tiempo, forma y presupuesto
BD&A	Desarrolla una <b>visión compartida del cliente</b> Estima crecimiento de mercado	Cambia concepción y uso de datos en la empresa <b>Cambia paradigmas para toma de decisiones</b> Usa aprendizaje para transformar interacciones, predecir escenarios	<b>Gestión basada en descubrimiento.</b> Desarrolla teorías e hipótesis Identifica datos relevantes Experimenta Afina o desecha hipótesis	PMP con conocimientos TI, matemáticas Conocimientos de la U. de negocio <b>Estadística, ciencia de datos Ciencia de comportamiento y cognitiva</b>	<b>Empresa basa sus decisiones en evidencia arrojada por los datos</b> Empleados usan datos para generar conocimiento en nuevos contextos

# Big Data y Analítica

- La nueva era caracterizada por la abundancia de datos
  - Ha alcanzado todos los sectores en la economía mundial
  - Los datos son un nuevo factor de producción y de ventaja competitiva
- Oportunidad
  - Aprender sobre el comportamiento humano para diversos fines
  - Creación de valor vía innovación, eficiencia y competitividad
  - Aumento del excedente del consumidor y del bienestar del ciudadano
- Nuevas formas de competencia y nuevos negocios
  - Almacenamiento y gestión de datos
  - Análisis de datos empresariales

# Fuentes de datos



**Mobile  
Sensors**



**Social  
Media**



**Video  
Surveillance**



**Video  
Rendering**



**Smart  
Grids**



**Geophysical  
Exploration**

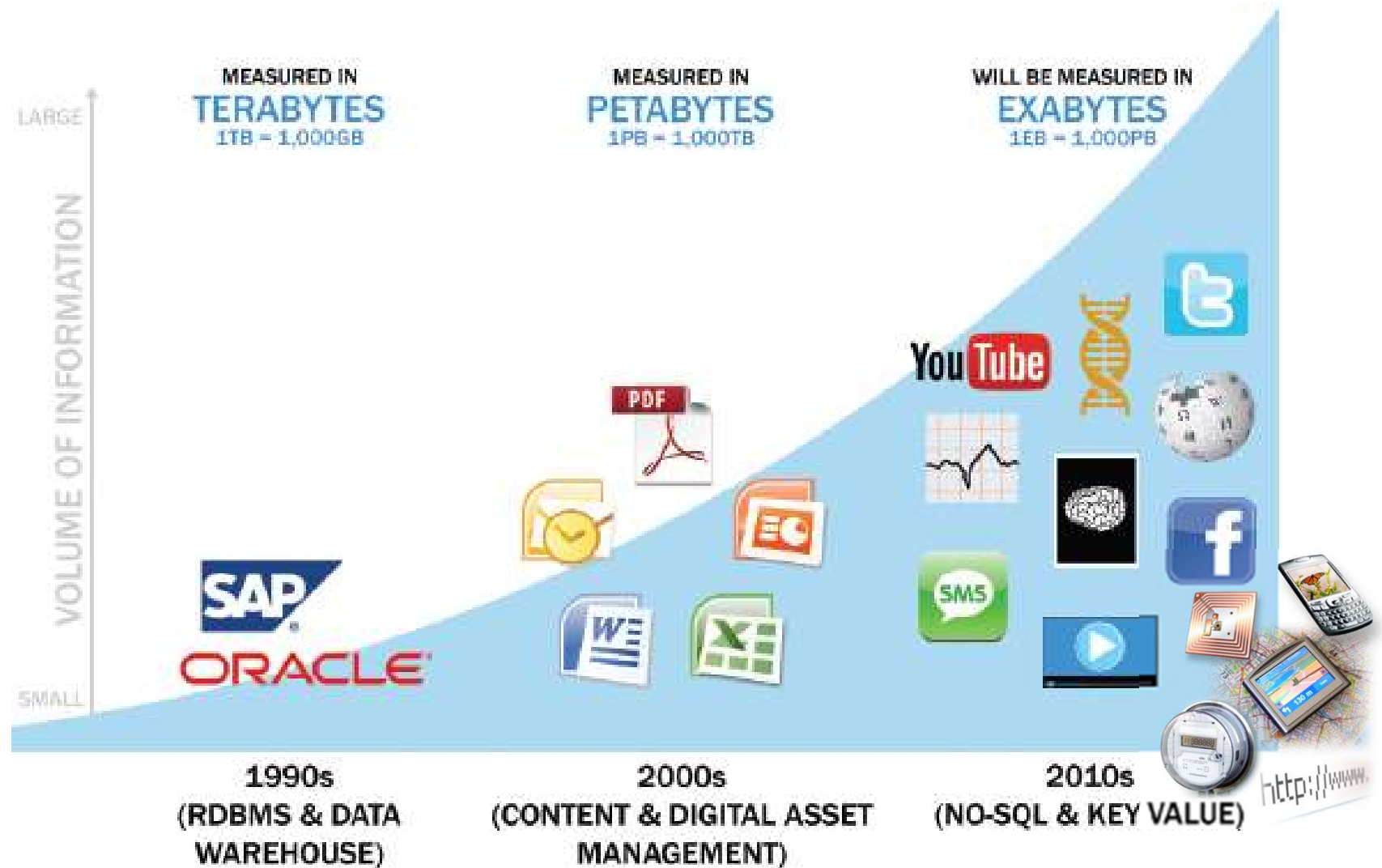


**Medical  
Imaging**



**Gene  
Sequencing**

# Fuentes de datos



# Explosión de información

- La cantidad de datos disponibles para análisis en una organización es muy superior a esa capacidad de análisis
- Las empresas se van volviendo mas “ingenuas” sobre su propio negocio



# Oportunidad tecnológica

- Incremento exponencial en capacidades de procesamiento (Ley de Moore), ancho de banda y almacenamiento, brindan capacidades de procesamiento mucho más poderosas a menor costo
- Podemos incorporar sensores con capacidad de procesamiento y comunicación en casi todo tipo de objeto

# Big Data

Al menos 80% datos  
no estructurados

Redes de sensores,  
redes sociales, hiper  
transacciones, ...



50x de 2010 a 2020

Veracidad: Uno de tres  
líderes toma decisiones  
sin confiar en sus datos

# Big Data

- Volumen
  - Creciente sociedad digital
  - Cantidad de datos generados duplicándose cada año
- Velocidad
  - Datos generados a gran velocidad, muchos de ellos deben ser analizados en tiempo real
- Variedad
  - Datos estructurados, semiestructurados, no estructurados
- Veracidad
  - Datos deben ser confiables para apoyar las decisiones y así crear Valor



# Ejemplo: Redes de sensores

- ▶ Volumen → Miles de millones de sensores
- ▶ Velocidad → ... deben ser procesados casi en tiempo real
- ▶ Variedad → ... gran variedad de tipos y de redes

## TIC clásicas

- Servidores
- BD relacionales
- Data Warehouse/  
Data Marts



- Muy poco soporte
- Costoso
- Procesamiento en tiempo real muy limitado

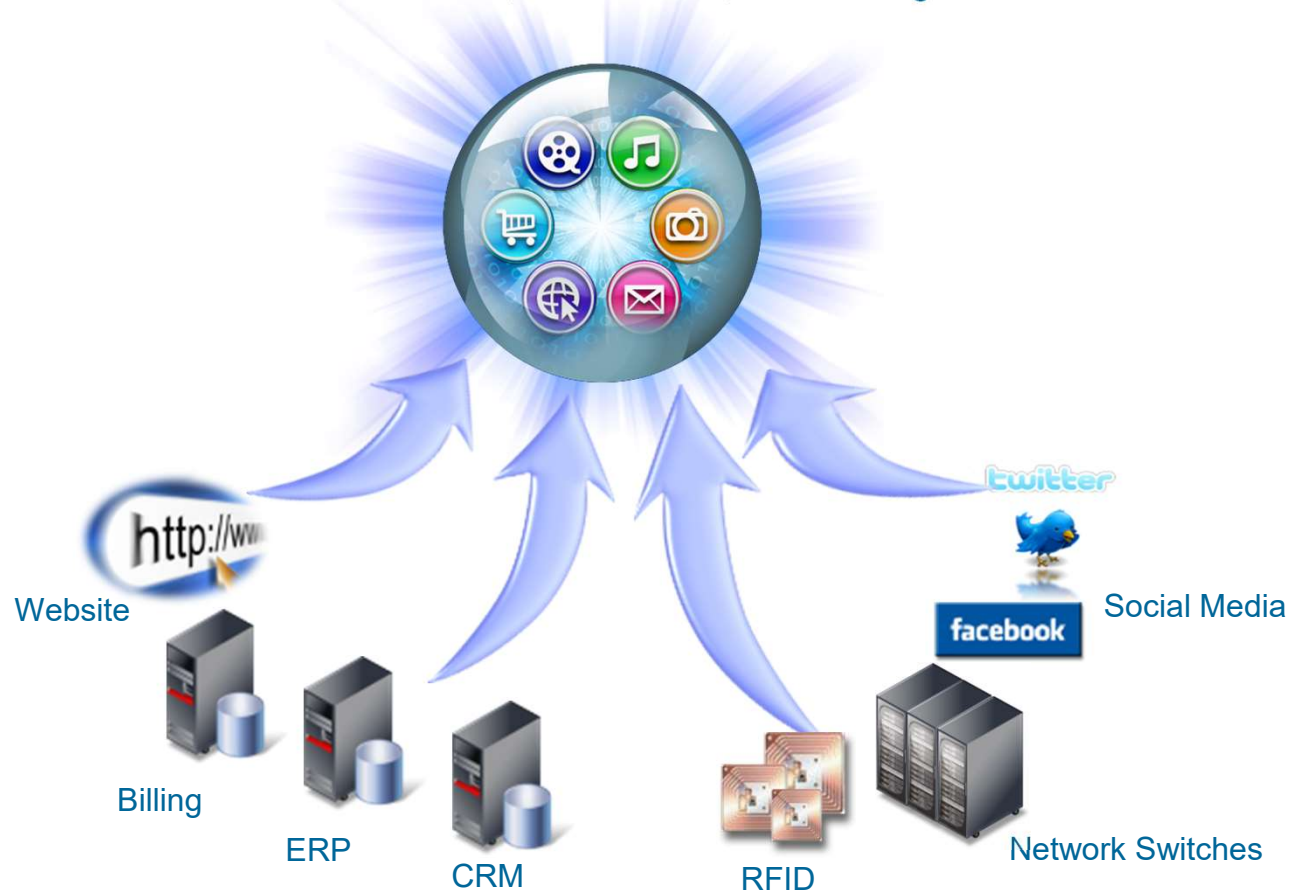
# Algunos beneficios

- Mucho mejor conocimiento del mercado y de todos los actores en el ecosistema
- Innovación en nuevos modelos de negocios, productos y servicios
  - Mejora de productos existentes
  - Desarrollo de nuevos productos (masa y personalización)
  - Nuevos modelos de servicio a nivel empresarial y gubernamental
- Apoyo a la toma de decisiones
  - Análisis de desempeño mejor y más oportuno.
  - Facilidad para redefinir la estrategia
- Transparencia y eficiencia al compartir datos

“El nuevo petróleo”

# Big Data is a Hot Topic Because Technology Makes it Possible to Analyze ALL Available Data

Cost effectively manage and analyze  
*all available data in its native form*  
*unstructured, structured, streaming*



# Una breve introducción a Hadoop y MapReduce

# ¿Cómo puedo transportar esta carga?





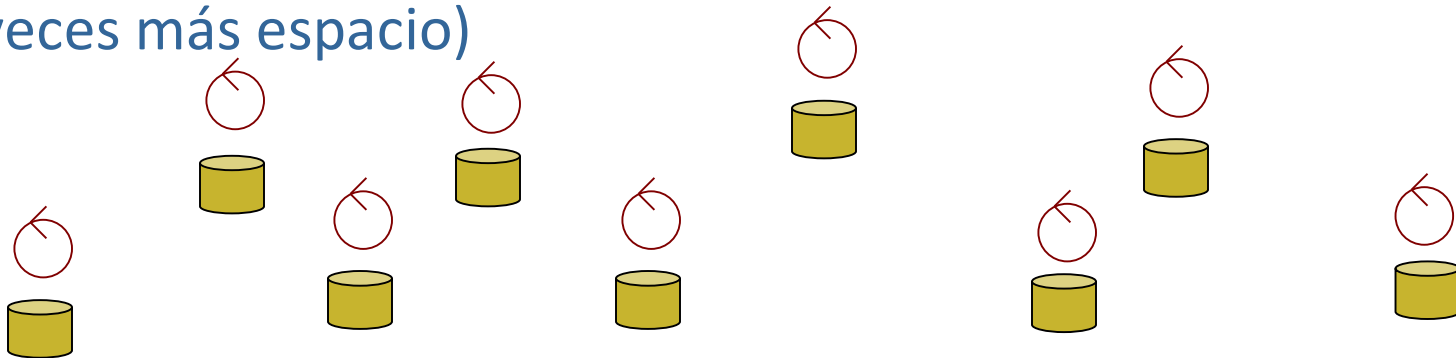


# HDFS y MapReduce

- Procesamiento de grandes volúmenes de información requiere de una gran capacidad de procesamiento y almacenamiento
- Mainframes, supercomputadoras, SANs del orden de Petabytes, excesivamente costosas
- Google observó que la gran mayoría de las operaciones requeridas eran bastante simples
  - *Sistema de archivos distribuido y librería de instrucciones relativamente simples*

# Hadoop y MapReduce

- Banco de datos de 1 TB.
  - UN disco duro, tasa de transferencia 100MB/s
    - Lectura del banco en 2.7 Hr
  - Cien discos duros, misma tasa
    - Lectura del banco en 2 min
- ... pero 100 discos aumenta drásticamente la probabilidad de fallos
  - Réplicas de información entre los discos (tenemos 99 veces más espacio)

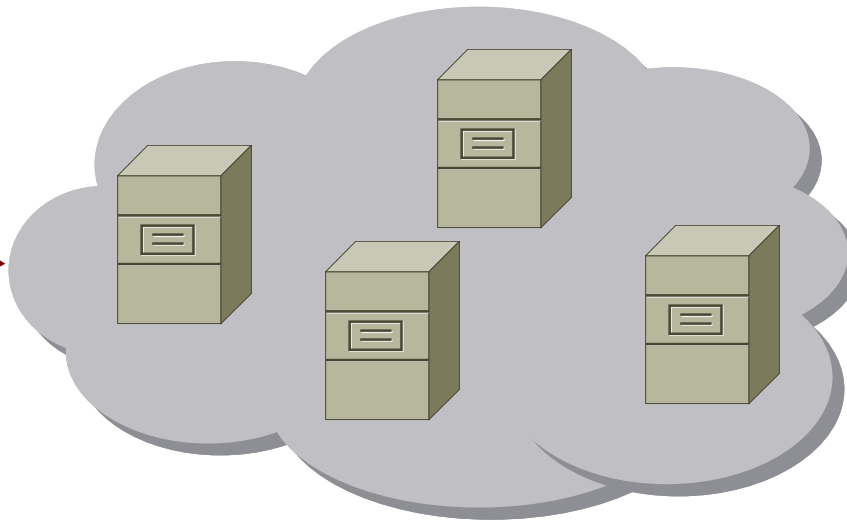




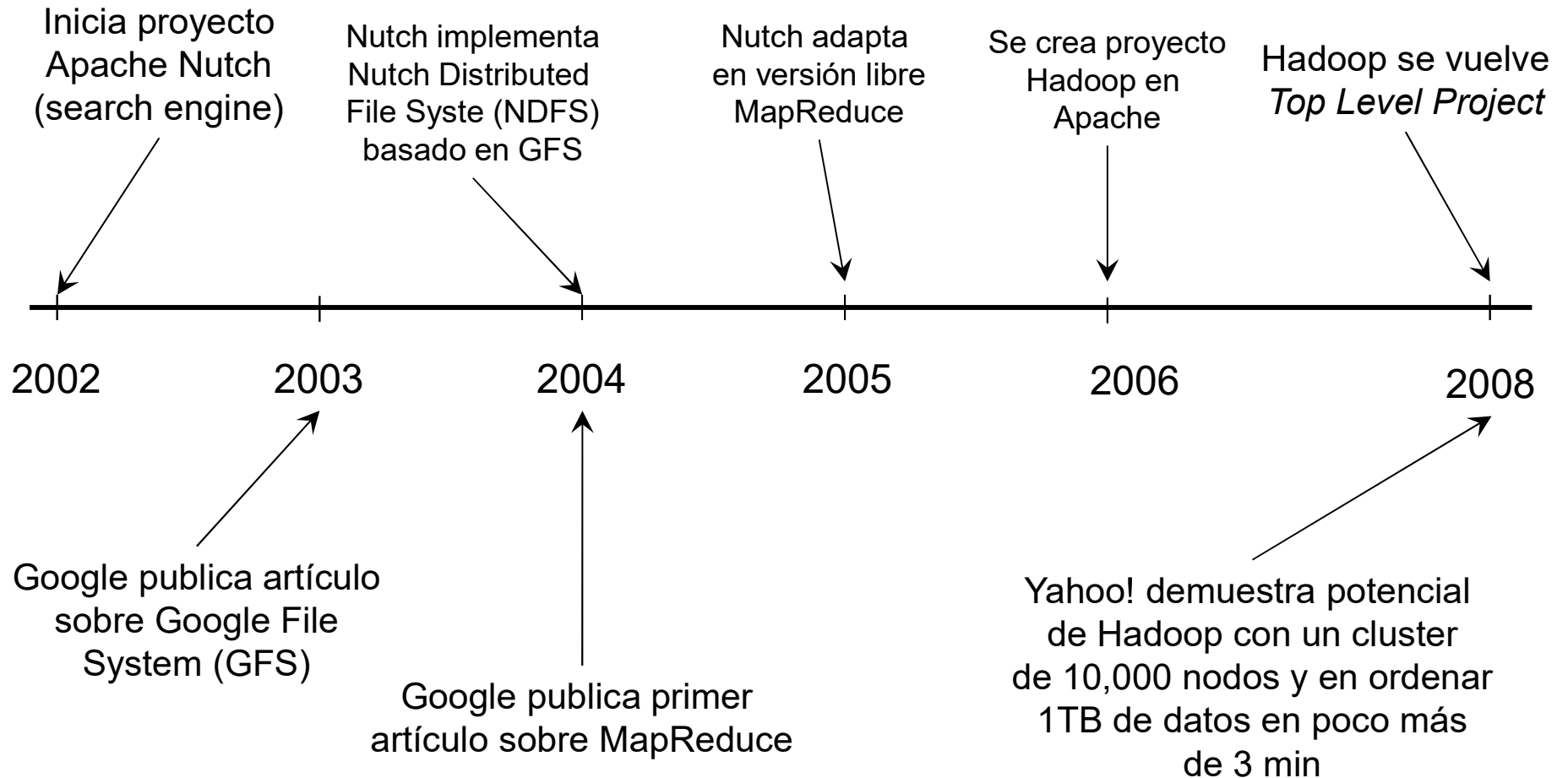


HDFS  
Almacenamiento  
confiable y de  
alta capacidad

MapReduce  
Procesamiento  
distribuido



# Algunos hitos



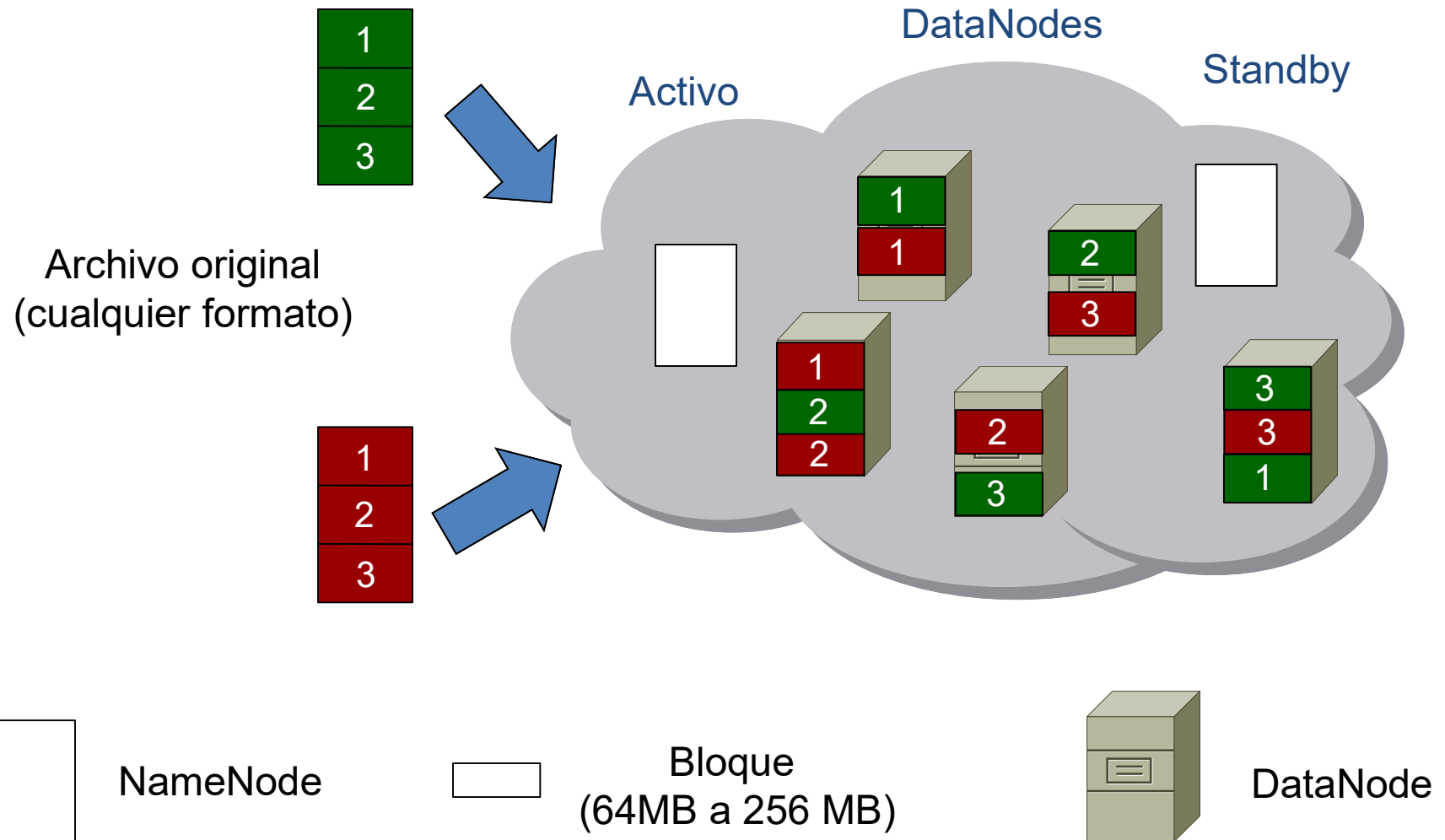


- Implementación de software libre (Apache Software Foundation) de la especificación GFS y MapReduce de Google
  - HDFS.- Sistema de archivos distribuido, redundante y escalable
    - A más nodos, más capacidad
  - Map Reduce.- Oculta la complejidad de paralelizar, sincronizar y garantizar la ejecución de tareas sobre los datos distribuidos en el HDFS



- Escrito en Java
- Proyecto software libre
- Utiliza hardware convencional de bajo costo
  - Confiabilidad basada en replicación
- Permite almacenar datos distintos (estructurados, semi/no estructurados)
- Sumamente escalable
- Optimizado para procesamiento en lotes
  - No para OLTP, OLAP, transacciones en t. Real

# Arquitectura HDFS



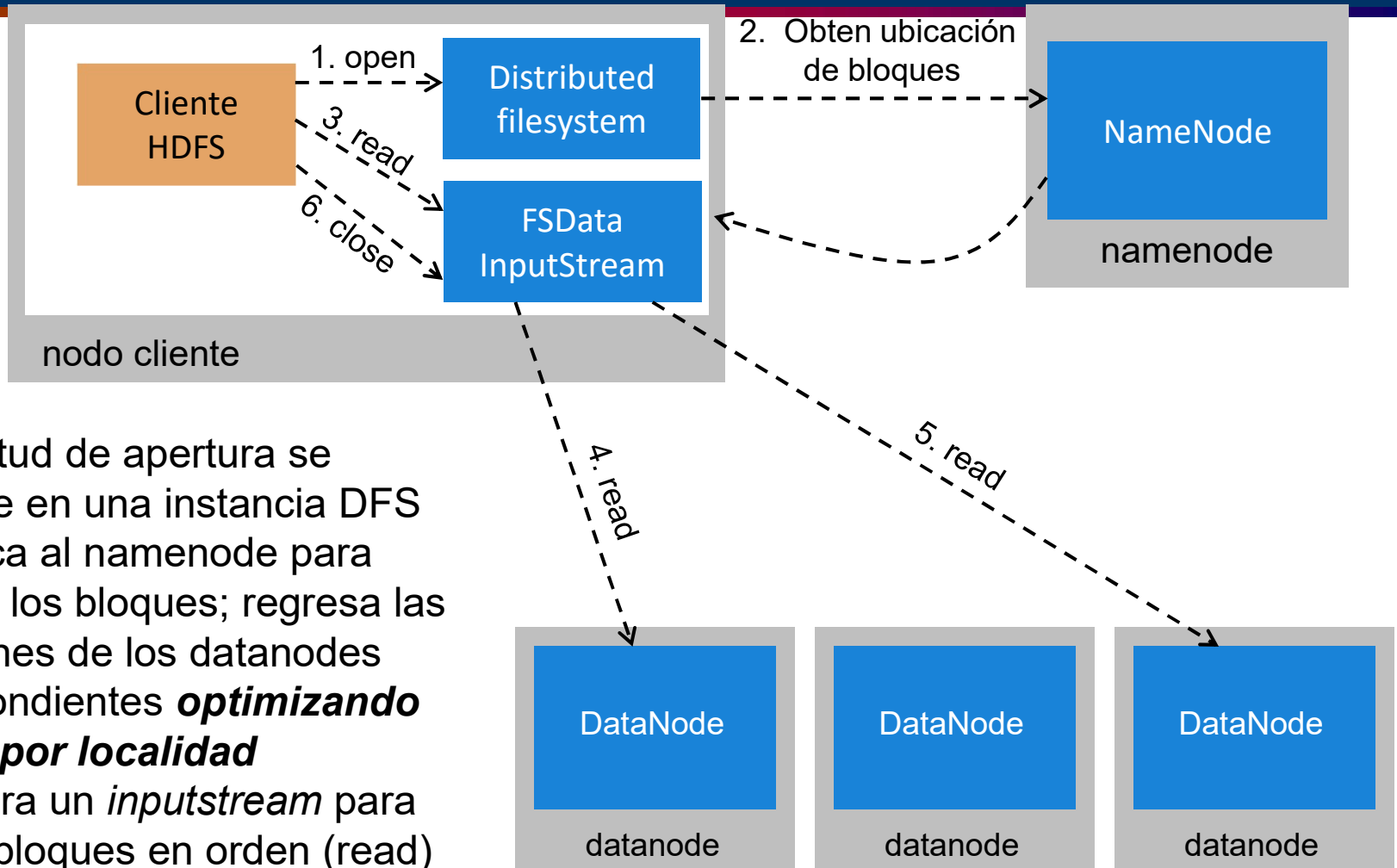
# HDFS

- Un solo *namespace* para el cluster
  - Administrado por un solo *NameNode (V1)*
  - Archivos son write-once, read-many. Solo se puede agregar información
  - Optimizado para flujos de lectura de grandes archivos (mejor pocos grandes que muchos pequeños)
- Archivos divididos en grandes bloques
  - Replicados en varios *DataNodes*
- Cliente interactúa con *NameNode* y con *DataNodes*
  - Desempeño escala casi linealmente con el número de *DataNodes*
  - Acceso desde Java, C, línea de comandos, lenguajes de scripts, ...

# Nodos en Hadoop V1

- NameNode
  - Uno por cluster. Maneja namespace y metadata
  - Es un elemento crítico. Sin él, no se puede acceder al sistema de archivos
- DataNodes
  - Almacenan bloques. Reportan periódicamente qué bloques tienen
- JobTracker
  - Uno por cluster. Recibe solicitudes de trabajos.
  - Dispara y monitorea tareas Map y Reduce en task trackers
- TaskTracker
  - Leen bloques de DataNodes y ejecutan tareas Map y reduce

# Anatomía de una lectura de archivo



- La solicitud de apertura se convierte en una instancia DFS
- Se invoca al namenode para localizar los bloques; regresa las direcciones de los datanodes correspondientes **optimizando acceso por localidad**
- Se genera un *inputstream* para leer los bloques en orden (read)
- Eventualmente se cierra este stream



# Comandos *shell* HDFS

- Varios comandos muy similares al manejo de archivos en POSIX (Unix/Linux)

`hadoop fs -ls`

`hadoop fs -put [-f] archivo <archivo>`

`hadoop fs -tail Archivo`

`hadoop fs -mkdir Dir`

`hadoop fs -mv Archivo Dir`

`hadoop fs -get Dir/Archivo`

`hadoop fs -rm Dir/Archivo`

`cat`  
`chgrp`  
`chmod`  
`stat`

`copyFromLocal`  
`copyToLocal`  
`getMerge`  
`Setrep`

Versión 2: `hdfs dfs -<cmd>`

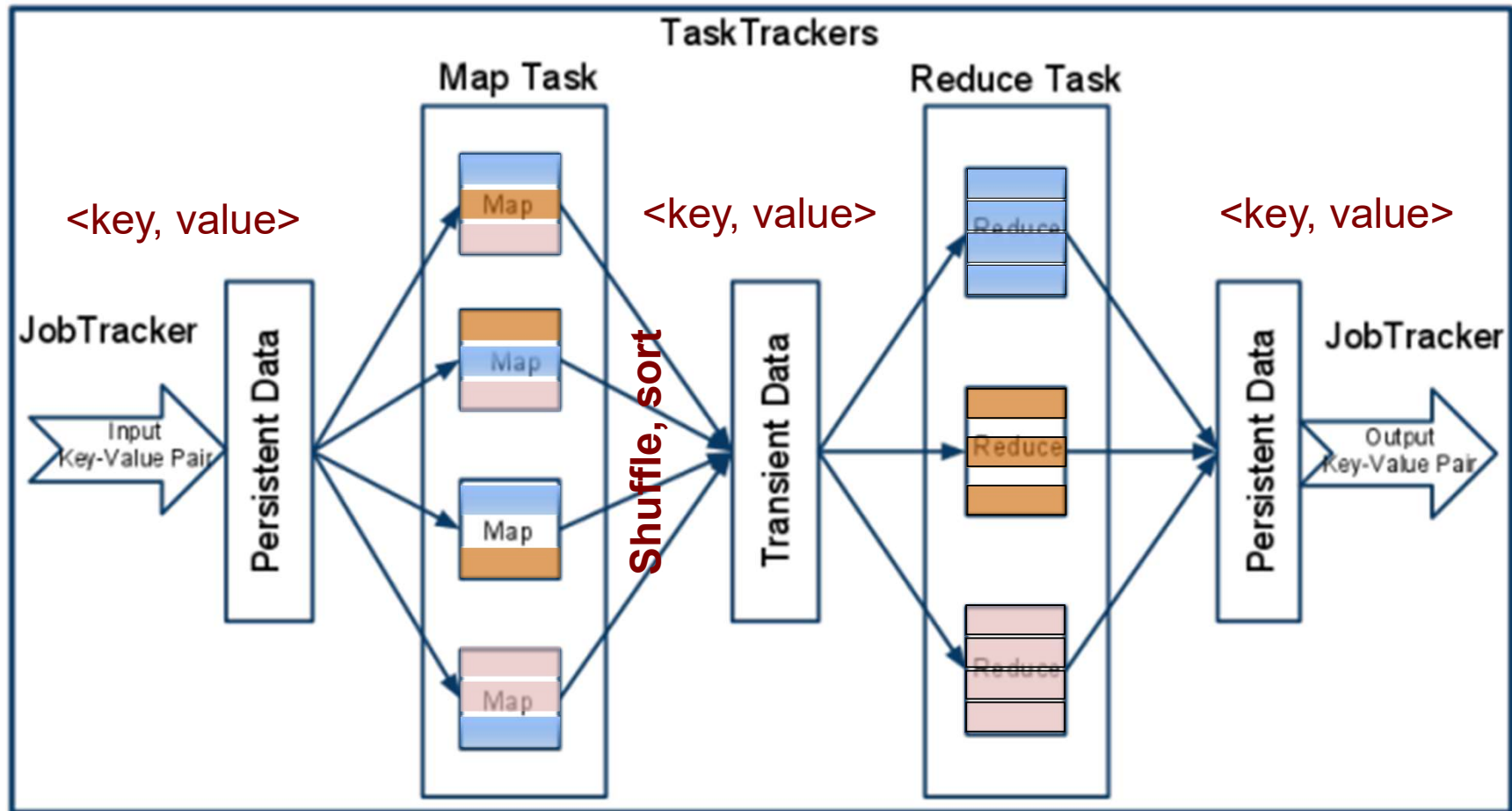
# Idea central MapReduce

- Los datos están dispersos en el cluster
- Lleva el procesamiento a los nodos donde están los datos, no al revés
- Las funciones Map tratan de asignarse al nodo donde se hospendan los datos que les toca procesar

# Conceptos básicos

- Un *job* es una unidad de trabajo de un cliente
  - Tiene datos de entrada, Programa MapReduce, información de configuración
  - Hadoop lo divide en tareas *map* y *reduce*
- Dos tipos de nodos para controlar la ejecución de *jobs*
  - JobTracker y TaskTracker
- Las entradas a un *job* se reducen en pedazos de tamaño fijo llamados *splits*. Se crea una tarea *map* para cada *split*
  - El tamaño del *split* es crítico. En la mayoría de los casos, es del tamaño de un bloque HDFS. Por omisión, hoy es de 128 MB
- El número de *reducers* se determina por configuración

# MapReduce



Los datos transitorios (las salidas de los map) se almacenan en el sistema de archivos local; los de los reducers, en HDFS

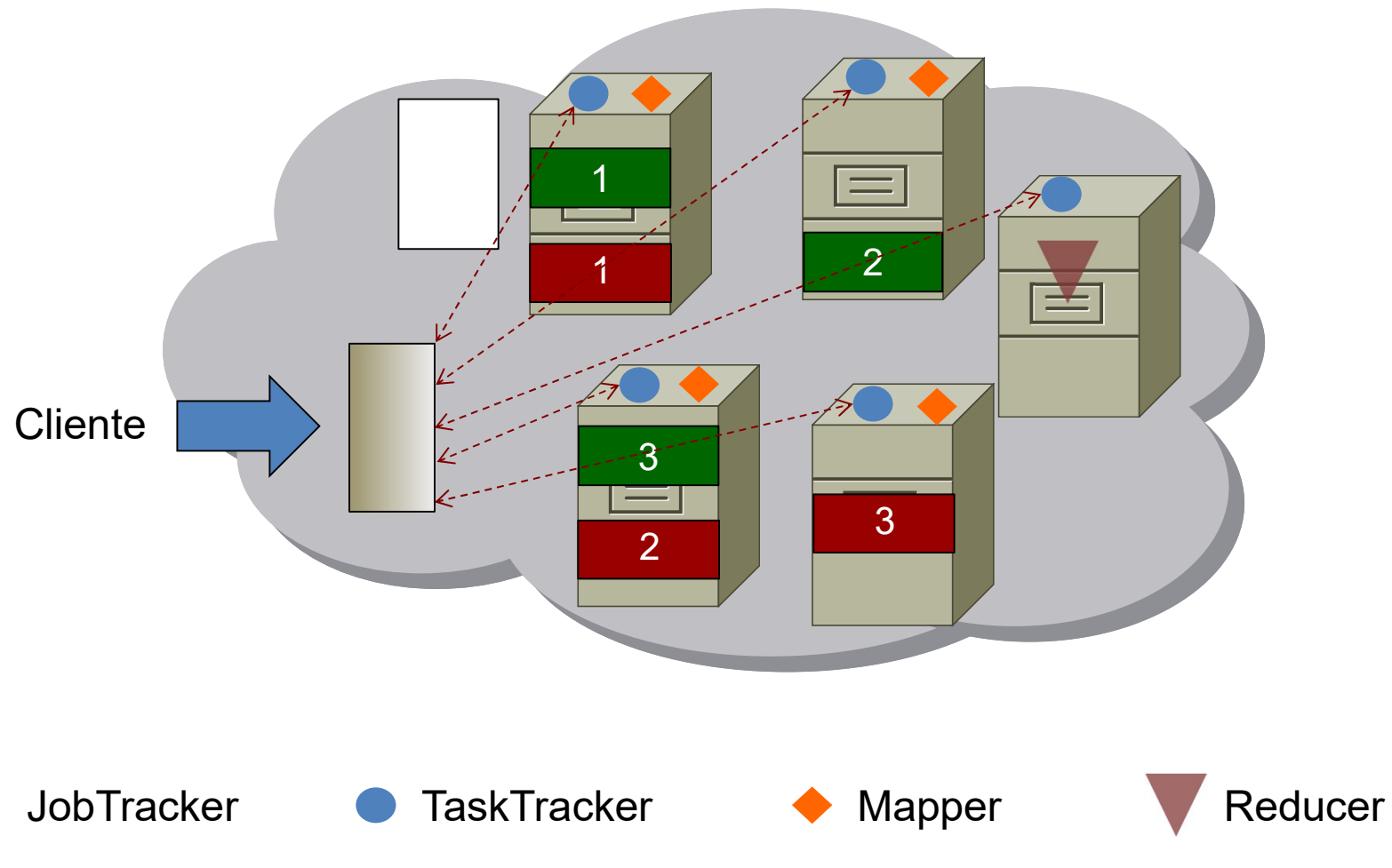
# MapReduce

- Rompe el procesamiento en dos fases:
  - Map.- Pre-procesamiento, limpieza de datos, filtrado
  - Reduce.- Procesamiento suplementario, resultados finales
- Cada fase tiene como entrada y salida tuplas *<key-value>*
  - Tanto la llave como el valor, pueden representar cualquier cosa (Hadoop los transforma en sus propios tipos de datos optimizados para ser fácilmente serializables)
- Entre las dos fases, hay un proceso de ordenamiento con base en la llave de salida de Map (y entrada a reduce)

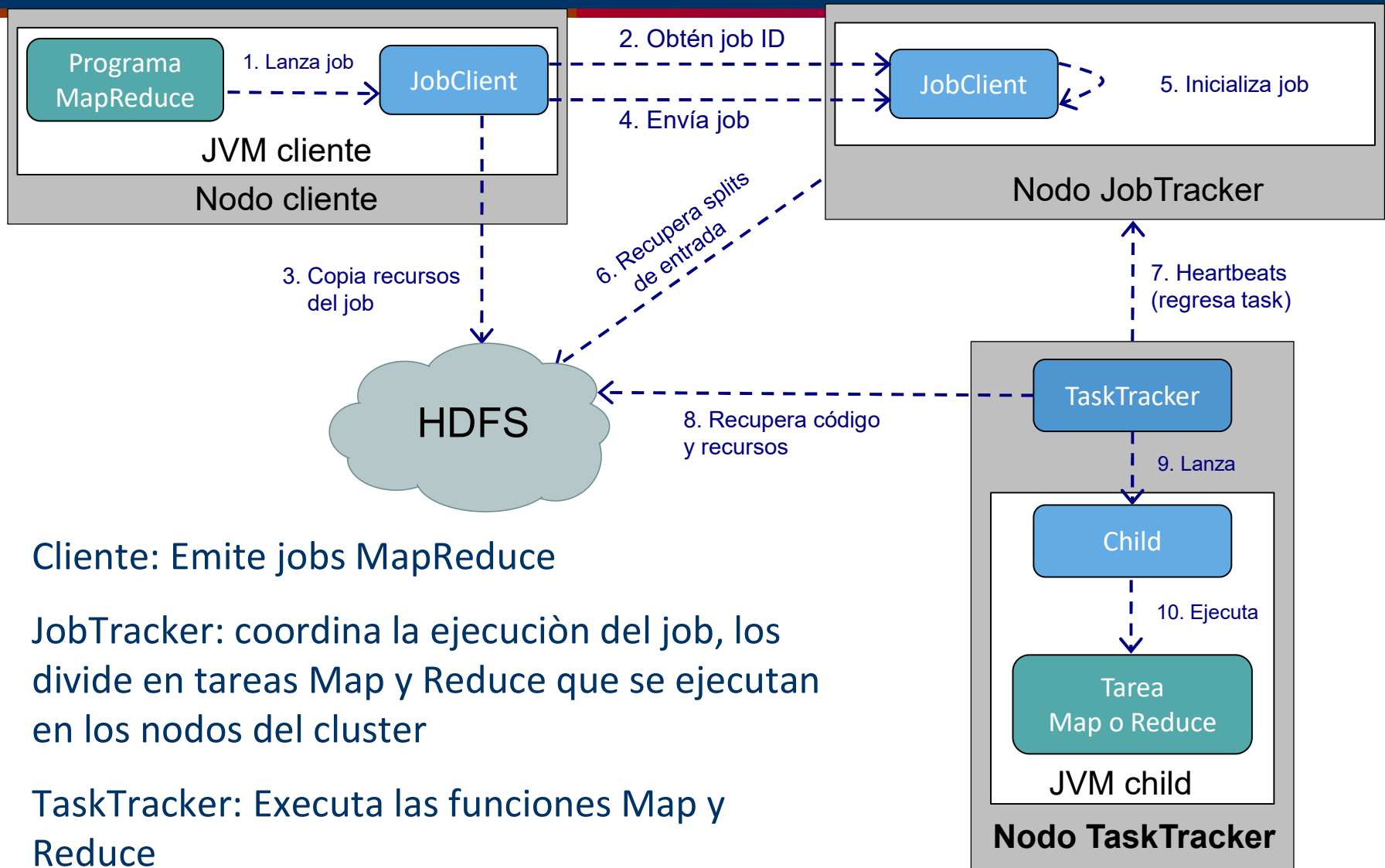
# Arquitectura MapReduce V1

- Arquitectura maestro/esclavo
  - Un solo maestro (JobTracker) controla la ejecución de múltiples esclavos (los TaskTrackers)
- JobTracker
  - Acepta jobs MapReduce enviados por los clientes
  - Lanza tareas map y reduce a los nodos Task Tracker
  - Monitorea las tareas y el estado de los Task Trackers
- TaskTracker
  - Ejecuta tareas map y reduce
  - Reporta estado a JobTracker
  - Gestiona almacenamiento y comunicación de salidas intermedias

# Procesos MapReduce V1



# Ejecución de un job (MapReduce V1)

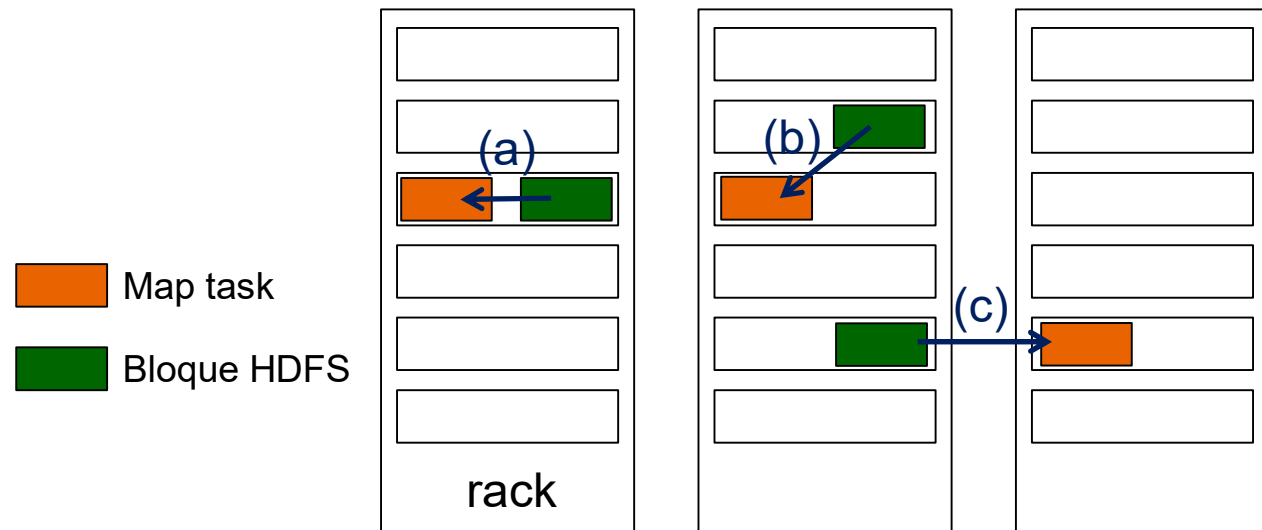


- Cliente: Emite jobs MapReduce
- JobTracker: coordina la ejecución del job, los divide en tareas Map y Reduce que se ejecutan en los nodos del cluster
- TaskTracker: Ejecuta las funciones Map y Reduce



# Optimización de ejecución

- En la medida de lo posible, una tarea *map* se ejecutará en el nodo donde residan los datos de entrada que ésta debe procesar (*data locality*)
- De no ser posible, buscará que los datos estén en el mismo *rack* (*rack local*).
- En última instancia, los tomará de un nodo en otro rack (*off-rack*)



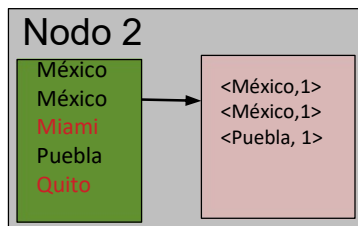
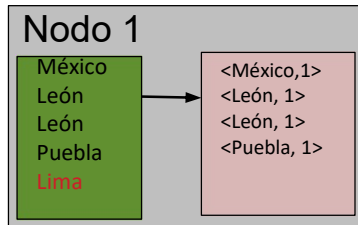
# Funciones *combiner*

- Dado que la comunicación entre *mappers* y *reducers* puede reducir el desempeño, conviene reducir la cantidad de información a transmitir
- Hadoop permite especificar funciones *combiner* a la salida de map para agrupar y minimizar los datos a ser transferidos
- Son relativamente pocas las operaciones que permiten la ejecución de una función combiner. Típicamente son operaciones de reducción (max, min, sum, ...).

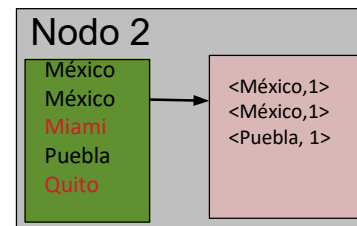
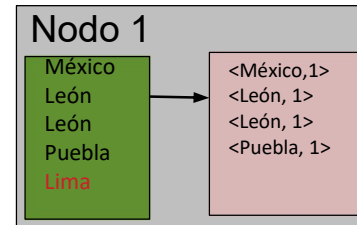
# Ejemplo MapReduce. Word count

Map

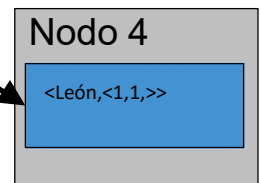
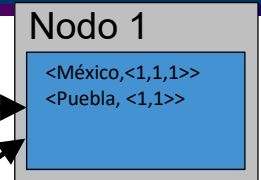
Shuffle



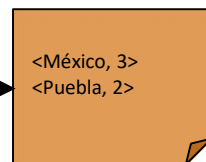
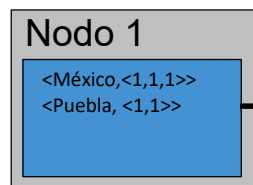
Las tareas Map se ejecutan localmente en cada split



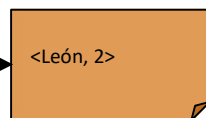
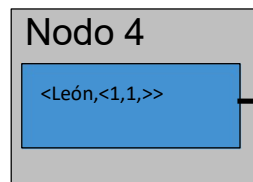
Claves iguales a mismos nodos para reducers



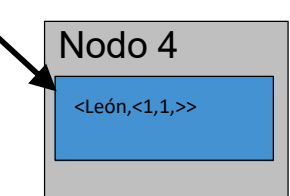
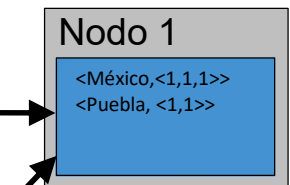
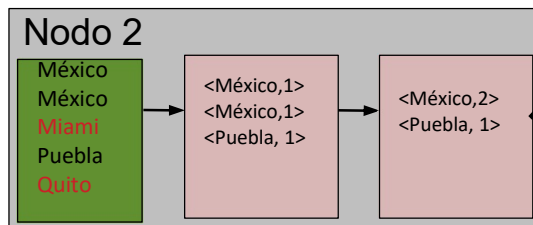
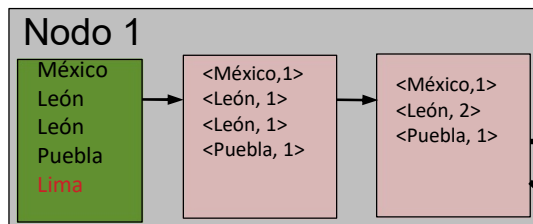
Reduce



Las salidas se escriben en HDFS



Combine (opcional)



# MapReduce en Java

- El código en Java tendría tres funciones:
  - Función map
    - Extiende la función genérica Mapper con cuatro parámetros formales: key-value para entrada y para salida
  - Función reduce
    - Extiende la función genérica Reducer también con cuatro parámetros formales
  - Función main
    - Controla ejecución del *job* MapReduce.
  - El código se empaqueta en un archivo JAR que Hadoop distribuirá para su ejecución en el cluster

**Sin embargo, en el curso utilizaremos un API (Hadoop Streaming) para especificar funciones map y reduce como código independiente, y en otros lenguajes (python)**

# MapReduce

- Modelo de programación para cómputo distribuido eficiente

- Flujo de datos similar a pipeline de Unix:

```
cat input | grep | sort | uniq -c | cat > output
```

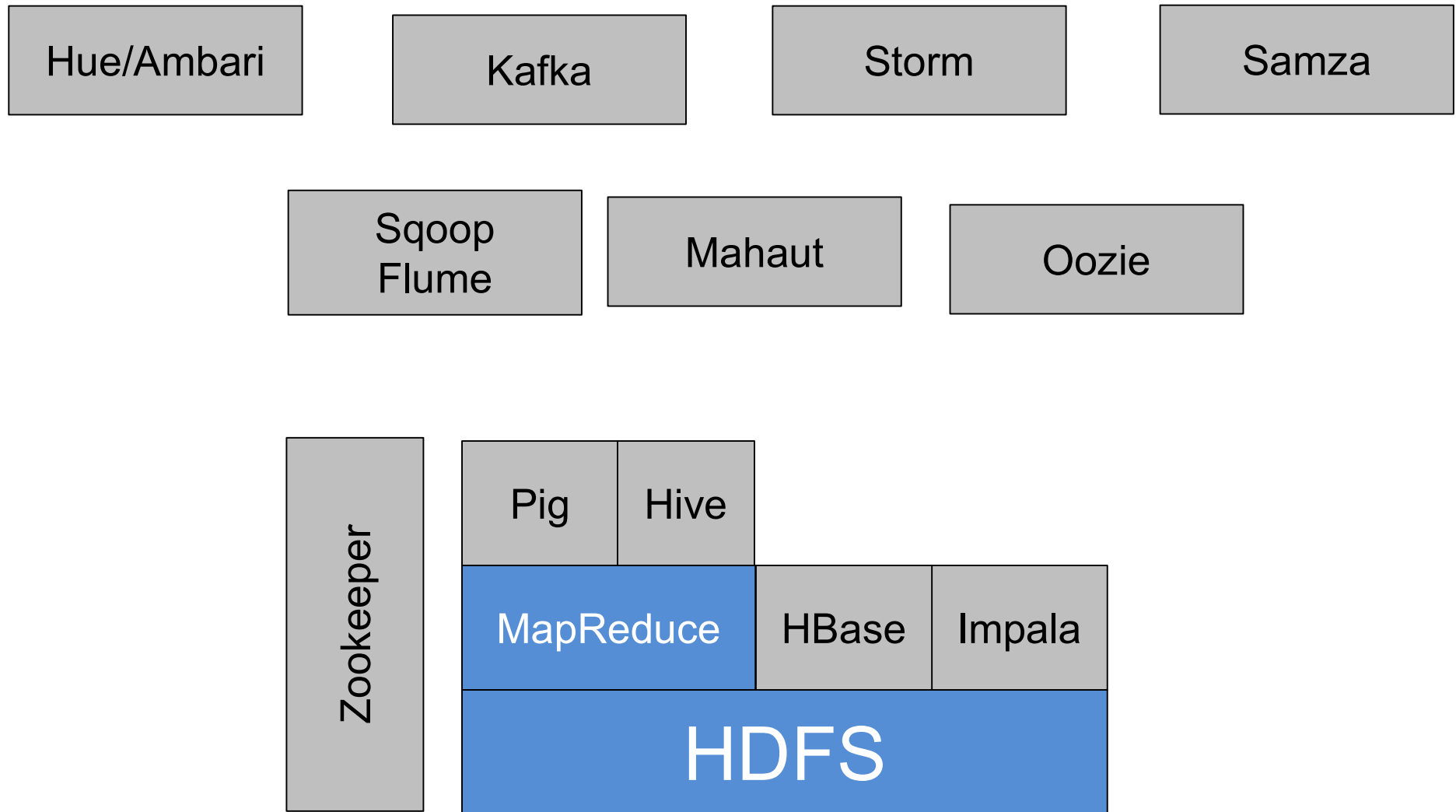
**Input** | **Map** | **Shuffle & Sort** | **Reduce** | **Output**

- La eficiencia se obtiene de:
  - Dividir tareas que se ejecutan en paralelo
  - Pipelining
- El reto es “paralelizar” el código. Muchas aplicaciones son muy difíciles de llevar a este modelo
  - Ideal para aplicaciones donde hay muchos datos que pueden ser procesados independientemente

# Desarrollo de un programa

1. Diseñar el código en términos de procesos Map y Reduce
  - Si el código es Java, diseñar funciones map, reduce y driver para crear contexto
  - Si el código es otro lenguaje, utilizar API Streaming
2. Pruebas locales con conjunto de datos pequeño y representativo
3. Pruebas en cluster bajo condiciones controladas
4. Profiling y optimización
5. Despliegue en producción

# Ecosistema Hadoop - BigData



- HBase – Base de datos columnar distribuida (NoSQL)
- Hive – Datawarehouse distribuido con lenguaje tipo SQL
- Pig – Lenguaje de alto nivel (oculta modelo MapReduce) para explorar grandes sets de datos
- ZooKeeper – Configuración y sincronización entre nodos
- Hue – Agrupa muchos de los componentes de Hadoop (HDFS, MapReduce/YARN, Hbase, Hive, Pig, Sqoop, Oozie) en una sola interfaz gráfica



- Oozie – Despachador y administrador de flujos de trabajo en MapReduce y Pig
- Sqoop – Interfaz en línea de comandos para transferencias masivas entre bases relacionales y Hadoop
- Mahout – Minería de datos/aprendizaje de máquina, algoritmos matemáticos
- Kafka – Gestor de colas distribuido para generar flujos (streams) en tiempo real
- Storm – Procesamiento en flujos en tiempo real
- Samza – Similar en concepto a Storm, pero más adaptado a la arquitectura Hadoop/YARN

# Hadoop/MapReduce NO ES para todo tipo de tareas

- No para procesos transaccionales (acceso aleatorio)
- No para trabajos que no pueden ser paralelizados
- No para procesos que requieren de baja latencia
- No para ambientes con muchos archivos pequeños
- No para cómputo intensivo con pocos datos

# YARN

## Hadoop/MapReduce v2

Yet Another Resource Negotiator

# Principales limitaciones MapReduce V1

- La centralización en el manejo de jobs y recursos en el JobTracker, presenta problemas de desempeño, confiabilidad y escalabilidad
- Acoplamiento rígido entre el modelo de programación (MapReduce) y los recursos de la infraestructura (HDFS) limita el desarrollo de otros paradigmas de programación

# Sobrecarga JobTracker

- El JobTracker es responsable de dos funciones muy distintas:
  - Gestión de los recursos de cómputo en el cluster
    - Lista de nodos activos, lista de slots disponibles para asignar tareas map y reduce, asignación de slots a tasks en función de las políticas de despacho, etc.
  - Coordinación de las tareas en ejecución
  - Iniciar tareas map y reduce, monitorerar su ejecución, reiniciar tareas en caso de fallos, actualizar contadores, etc.

En un cluster relativamente grande, el JobTracker puede terminar gestionando decenas de miles de tareas

# Subutilización TaskTrackers

- Un TaskTracker por datanode, simplemente mantiene comunicación regular con el JobTracker (heartbeats)
- Monitorea la ejecución de unas cuantas tareas (map o reduce) asignadas por el JobTracker

# YARN/ MapReduce V2

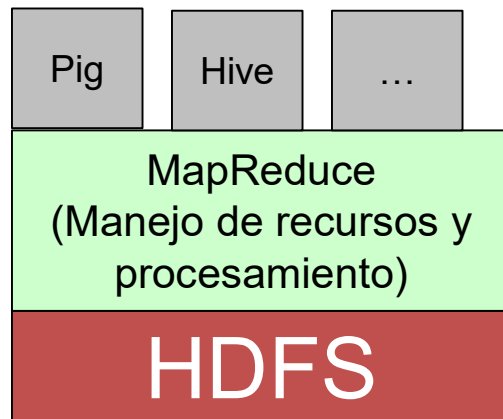
- Divide las funciones del JobTracker en dos actividades separadas:
  - ResourceManager (RM). Dos componentes
    - Un despachador global y un *agente* por nodo (NodeManager, NM) encargado de monitorerar el uso de recursos y reportarlos al despachador
    - Un ApplicationMaster (AM)
      - Un AM por aplicación (un job MapReduce o un grafo de jobs), encargada de negociar los recursos del RM y trabajar con los NM para ejecutar y monitorear las tareas

Se mantiene compatibilidad de APIs entre MRv1 y MRv2; todas las aplicaciones de la primera versión deberían poder ejecutarse sin modificación en el nuevo ambiente tras ser recompiladas

# Hadoop v1 y v2

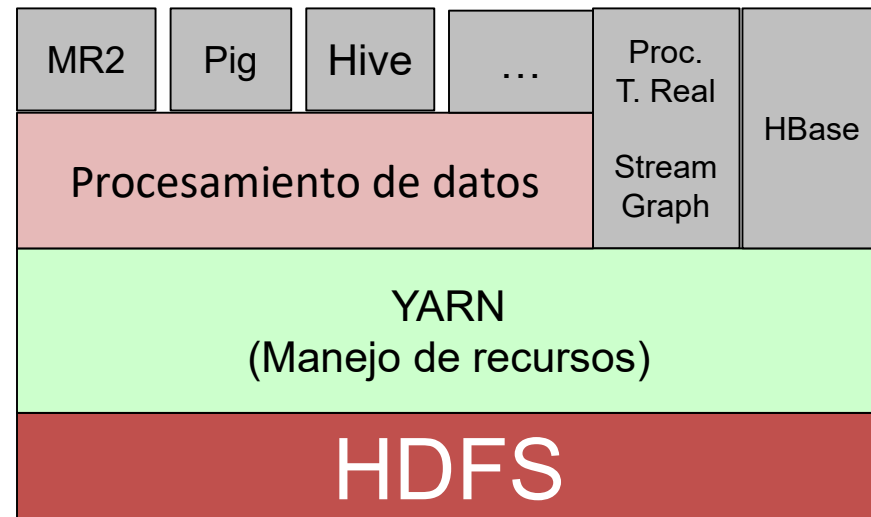
## Hadoop 1

Plataforma especializada



## Hadoop 2

Plataforma multipropósito



Con YARN, Hadoop V2 soporta distintos ambientes de ejecución. MapReduce es sólo uno de ellos

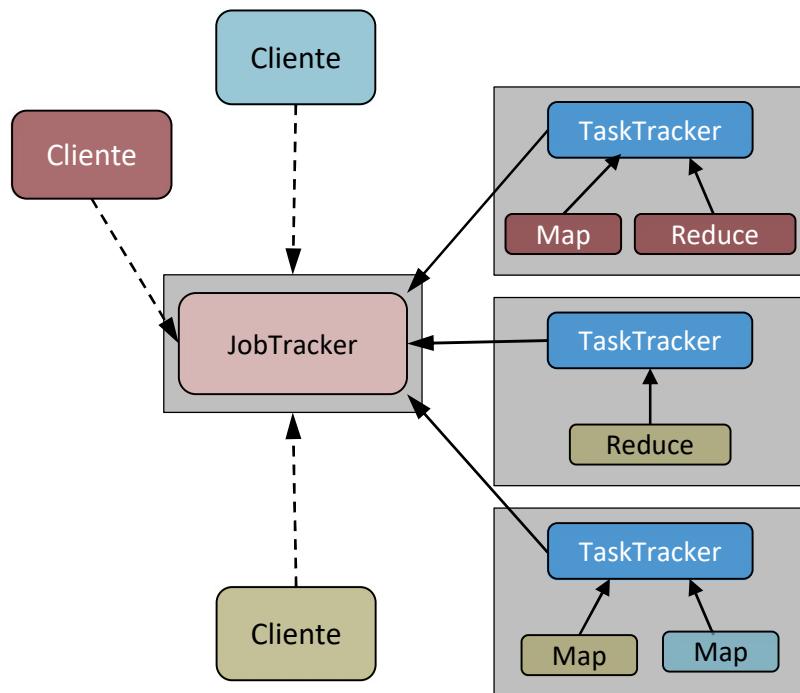


# Principales características

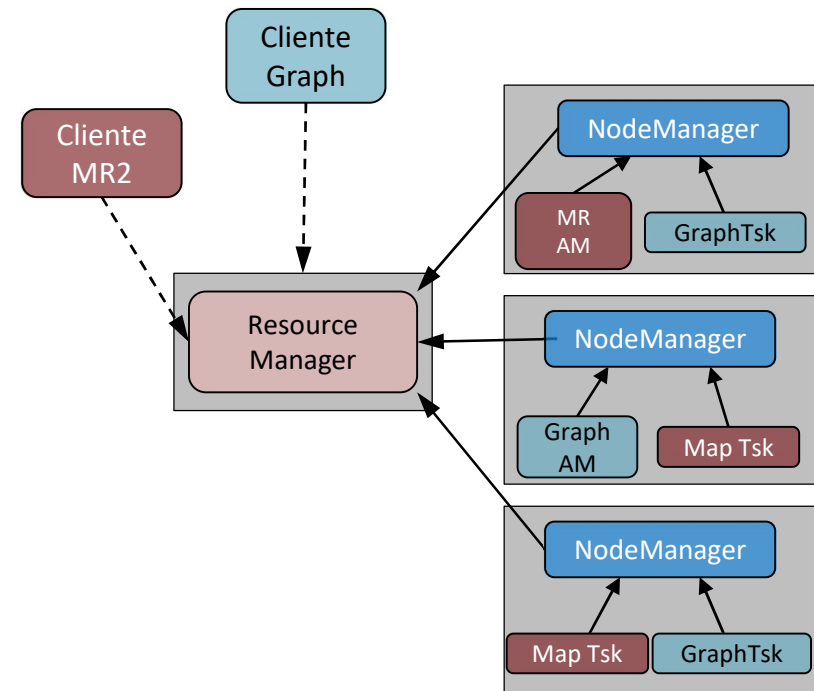
- Escalabilidad
  - Separar funcionalidades del JobTracker permite un incremento de 10x en el número de nodos y tareas
- Múltiples ambientes
  - Distintos ambientes de ejecución operando simultáneamente en la misma infraestructura con base en SLAs y políticas
- Compatibilidad
  - Misma API que Hadoop/MRv1
- Alta disponibilidad
  - Mecanismos para ofrecer NameNode de alta disponibilidad
- Mayor utilización de recursos
  - NodeManager es más eficiente que TaskTracker: permite creación dinámica de contenedores (y control de recursos)

# Diferencias en arquitectura

## Arquitectura Hadoop/MR1



## Arquitectura YARN

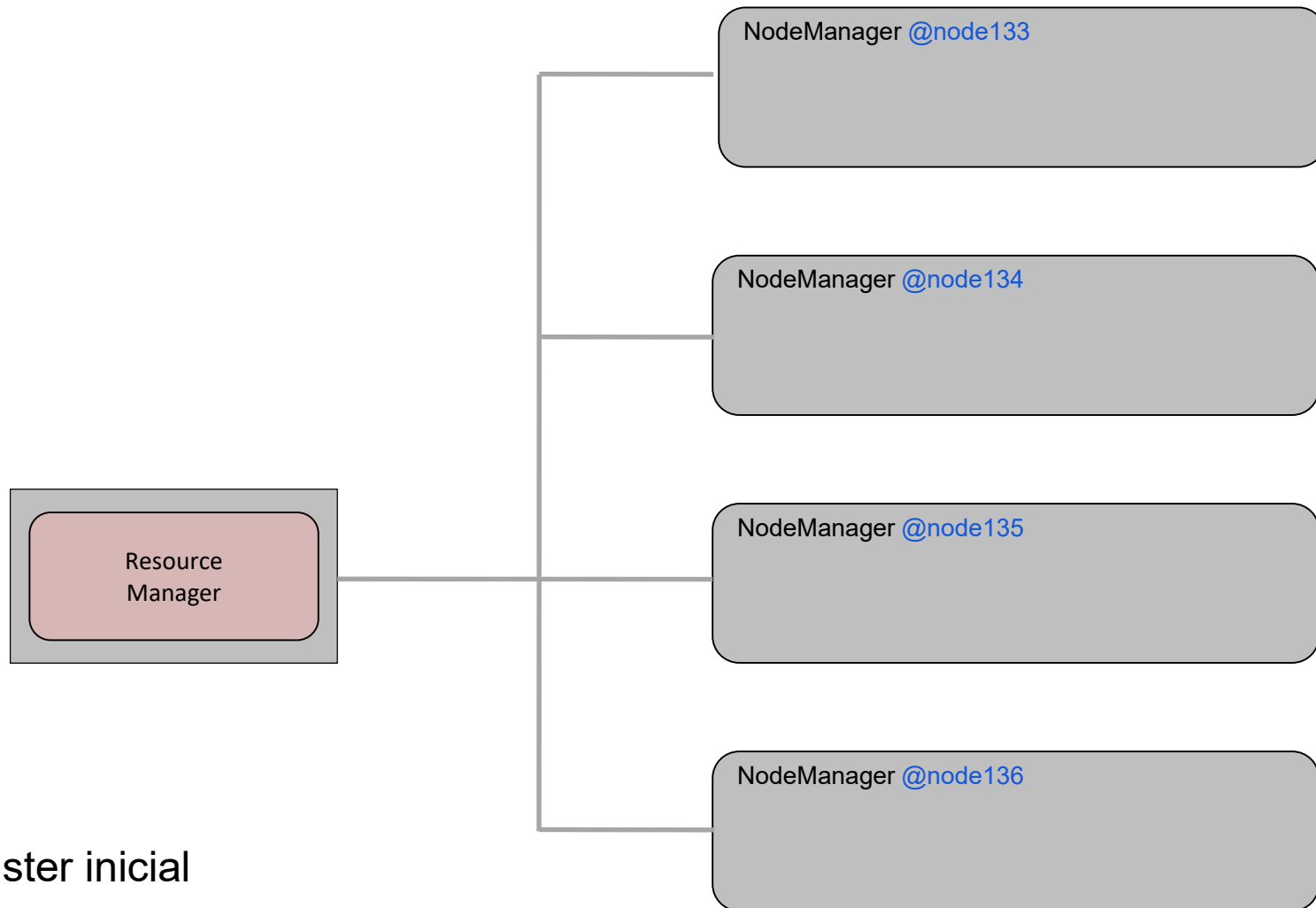


# Diferencias terminología

Hadoop/MR 1	YARN
Cluster Manager	ResourceManager
JobTracker	ApplicationMaster (dedicado y de corta vida)
TaskTracker	NodeManager
MapReduce Job (único tipo)	Distributed Application
Slot	Container

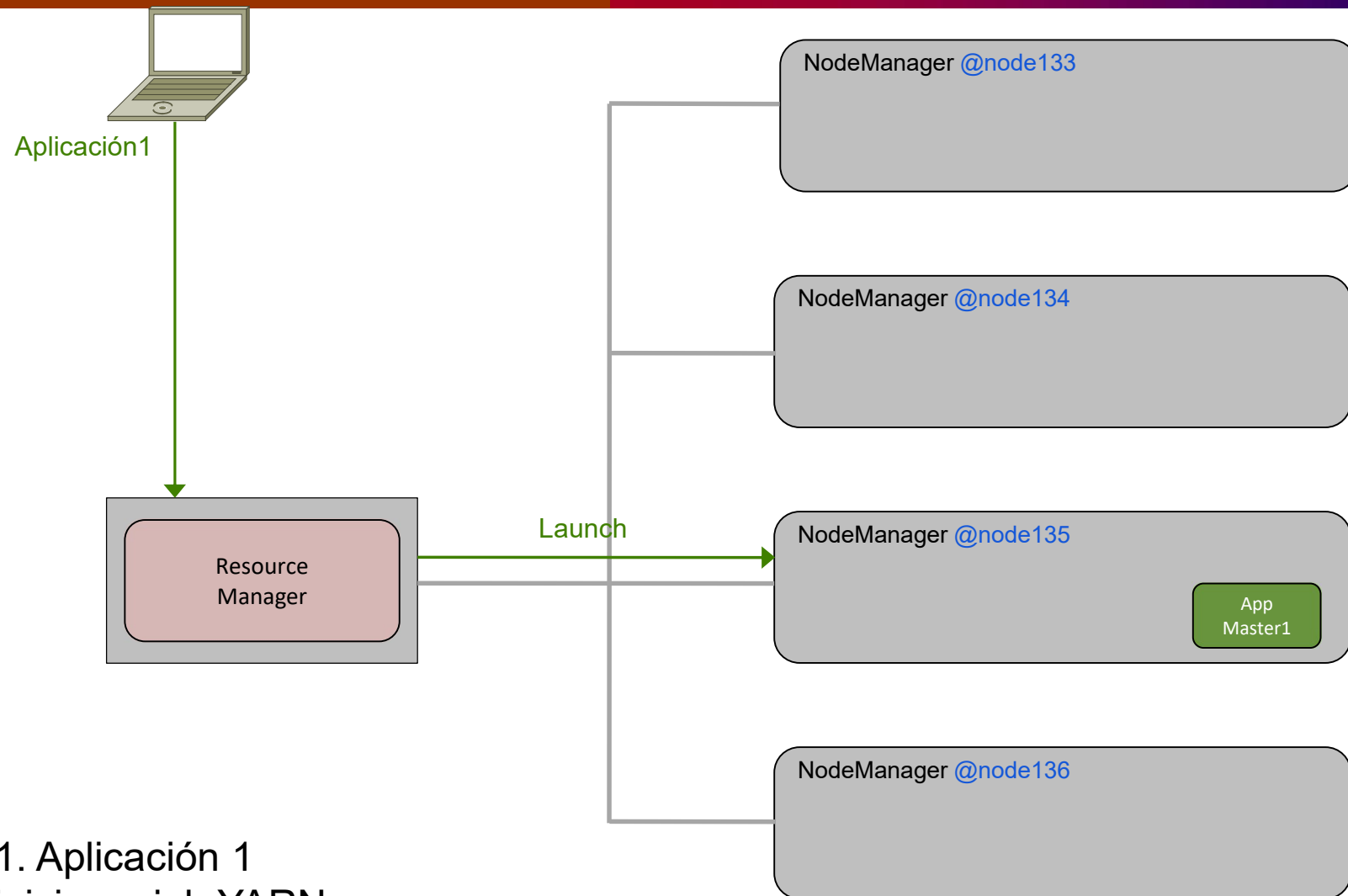
**En YARN los recursos se entienden en términos de contenedores supervisados por el NodeManager**

# Ejecutando una aplicación



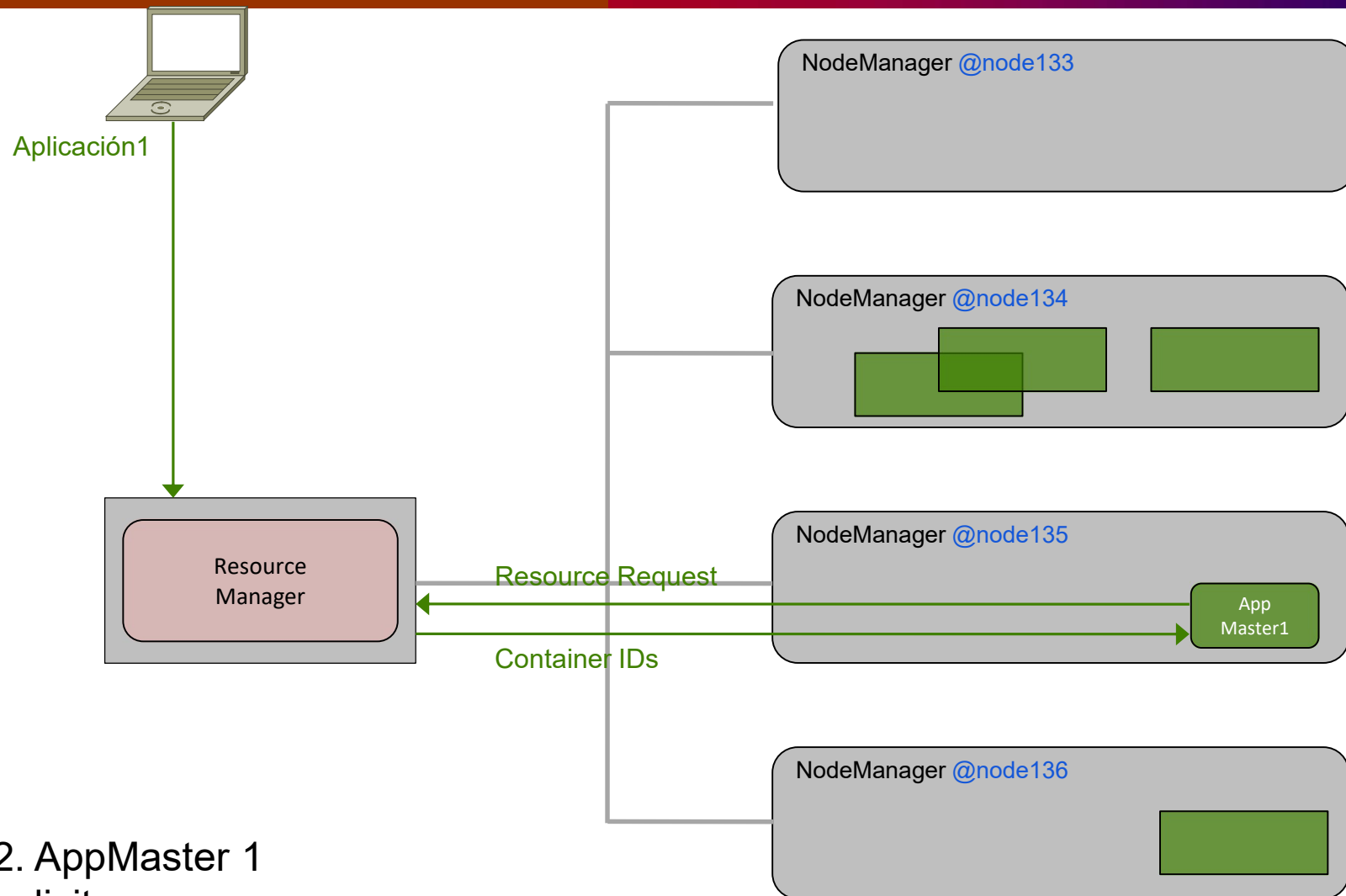
0. Cluster inicial

# Ejecutando una aplicación



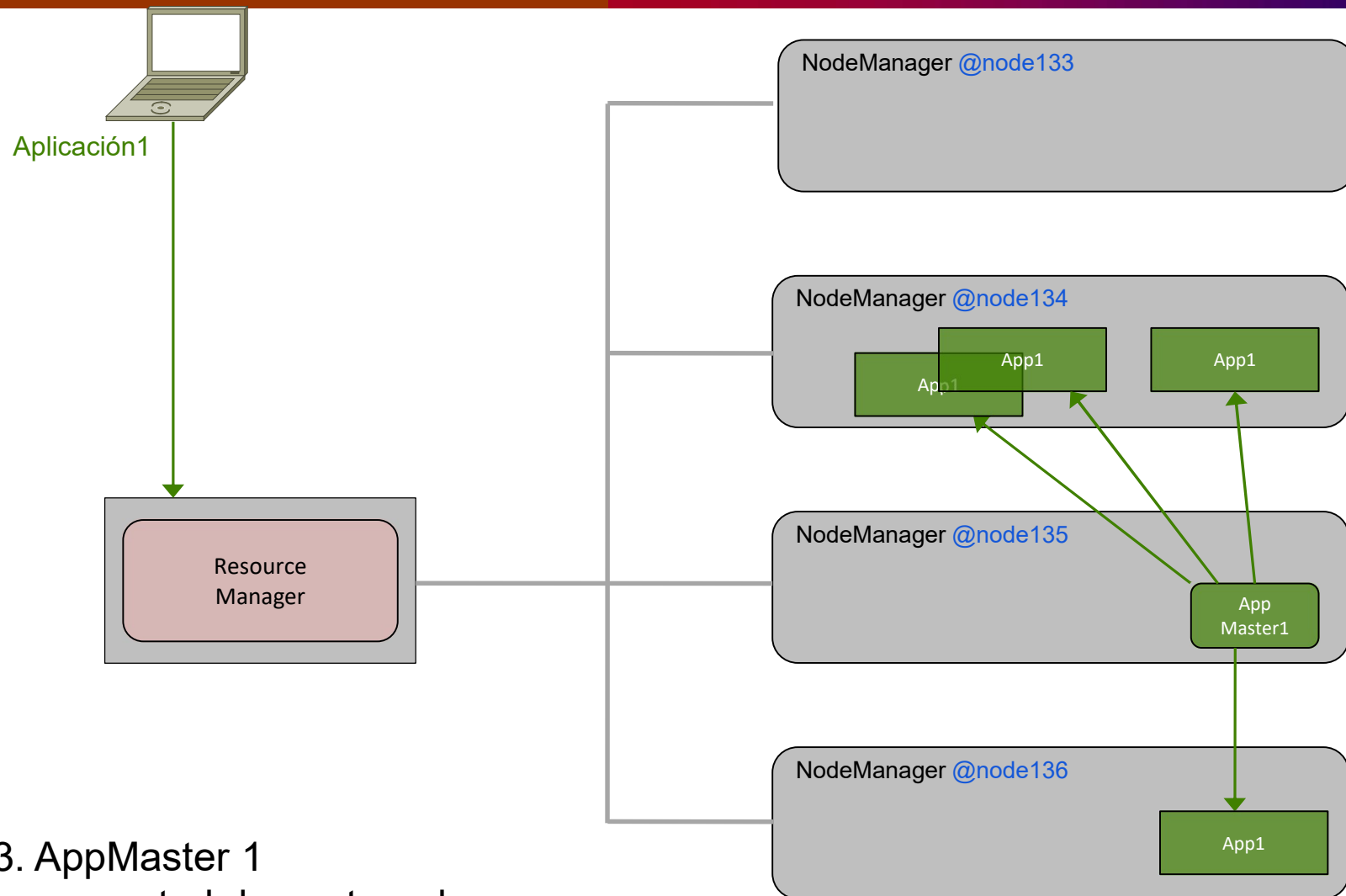
1. Aplicación 1  
inicia un job YARN

# Ejecutando una aplicación



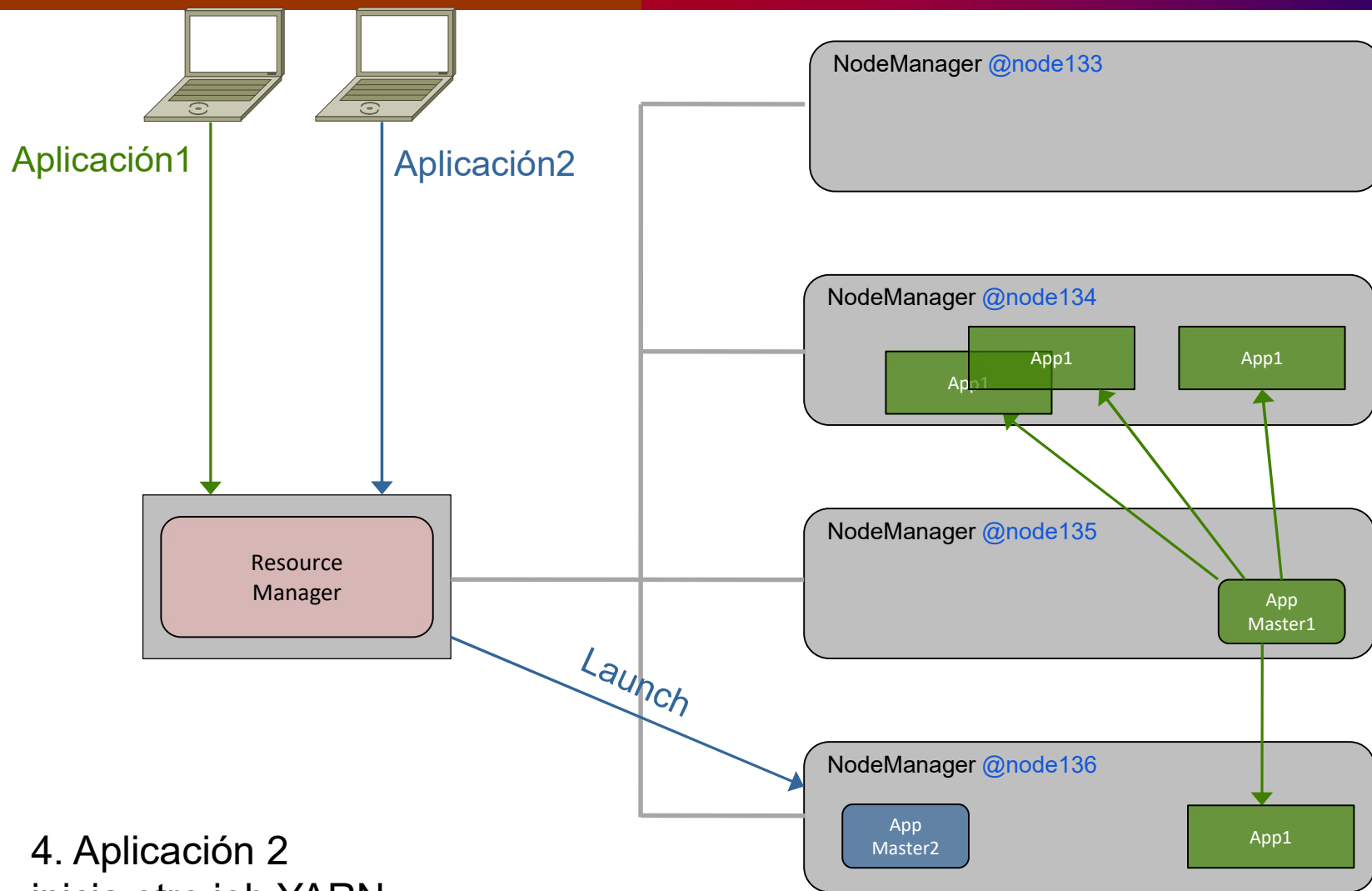
2. AppMaster 1  
solicita recursos

# Ejecutando una aplicación



3. AppMaster 1  
toma control de contenedores

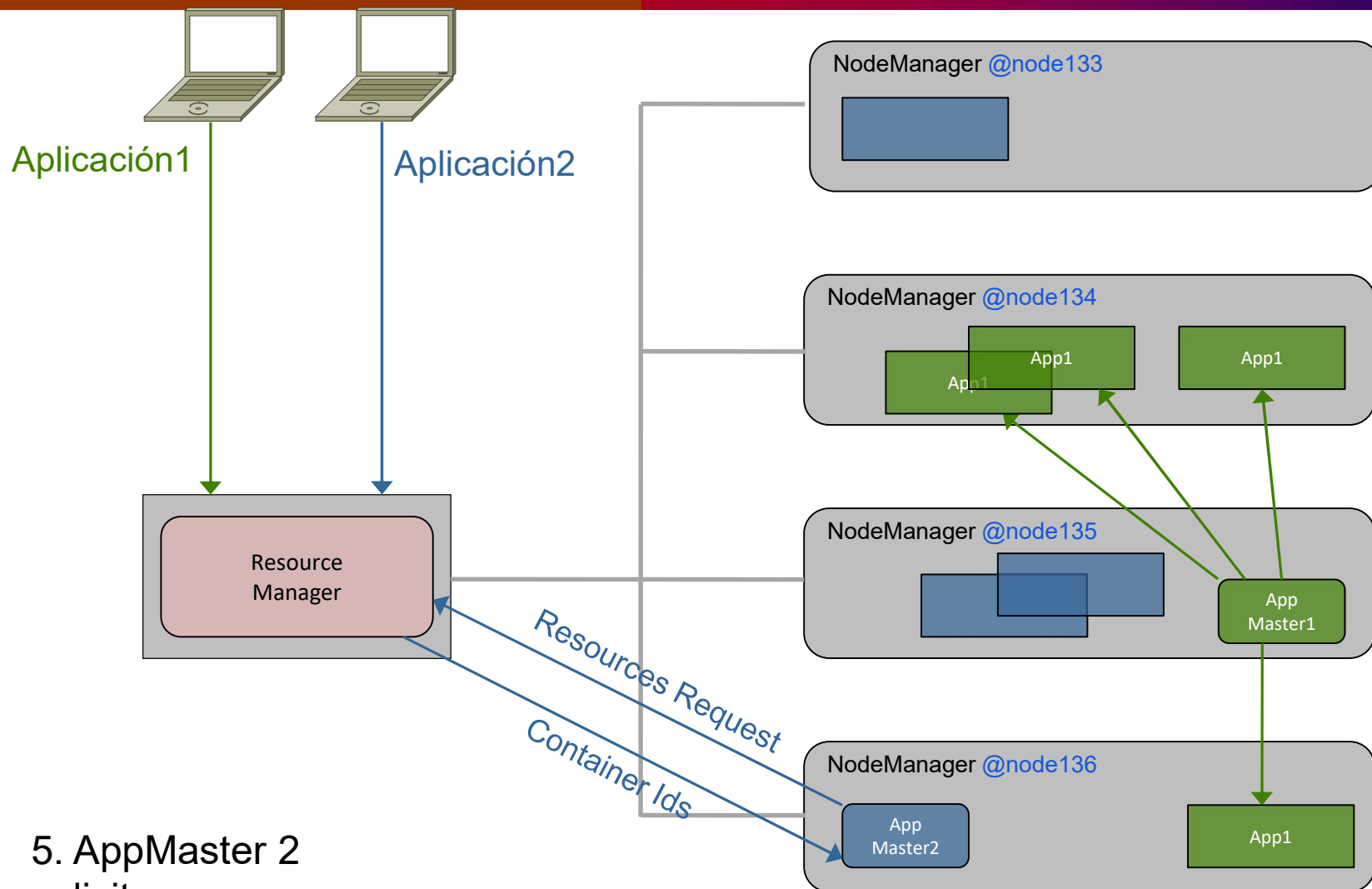
# Ejecutando una aplicación



4. Aplicación 2  
inicia otro job YARN

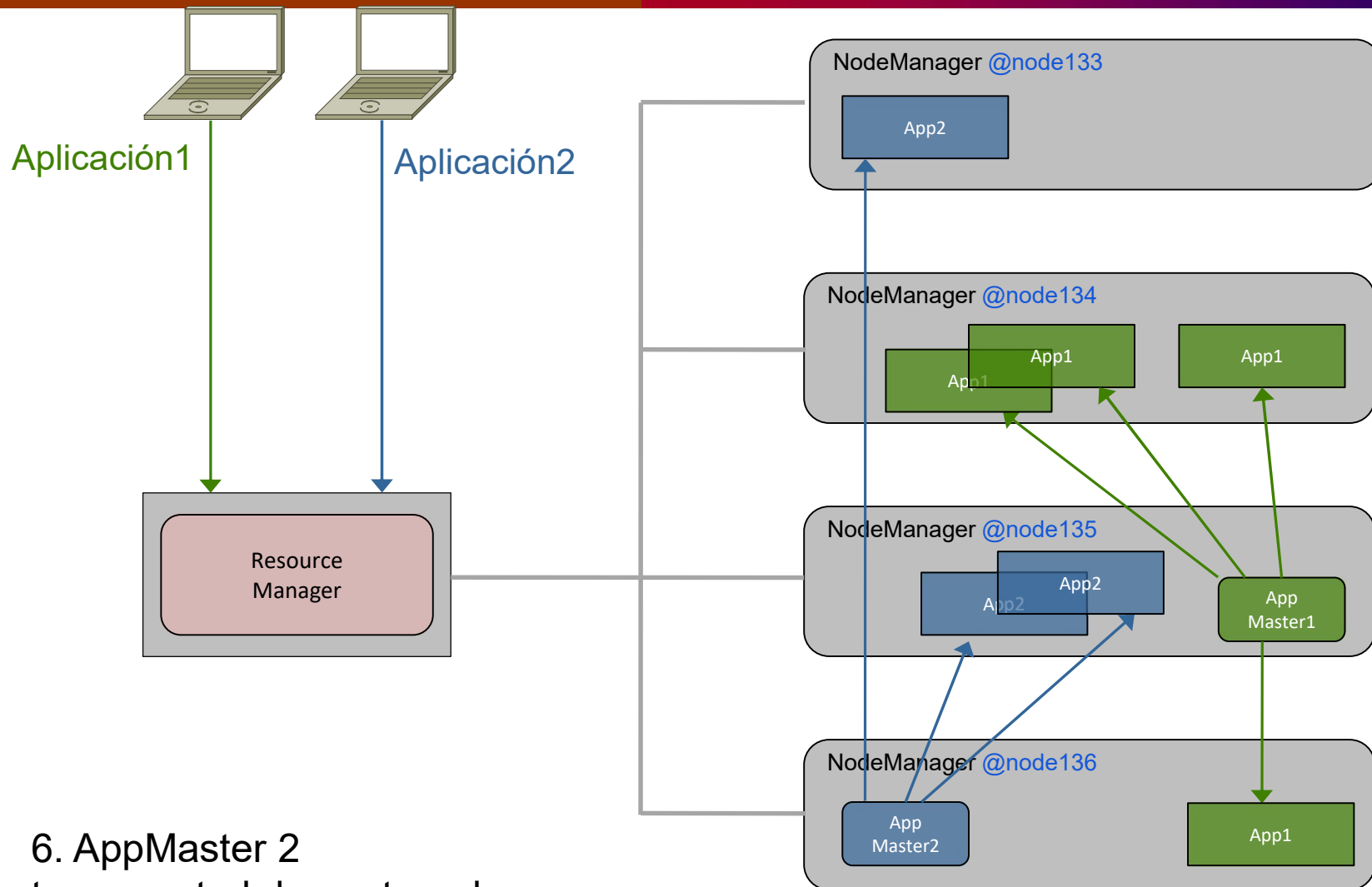


# Ejecutando una aplicación



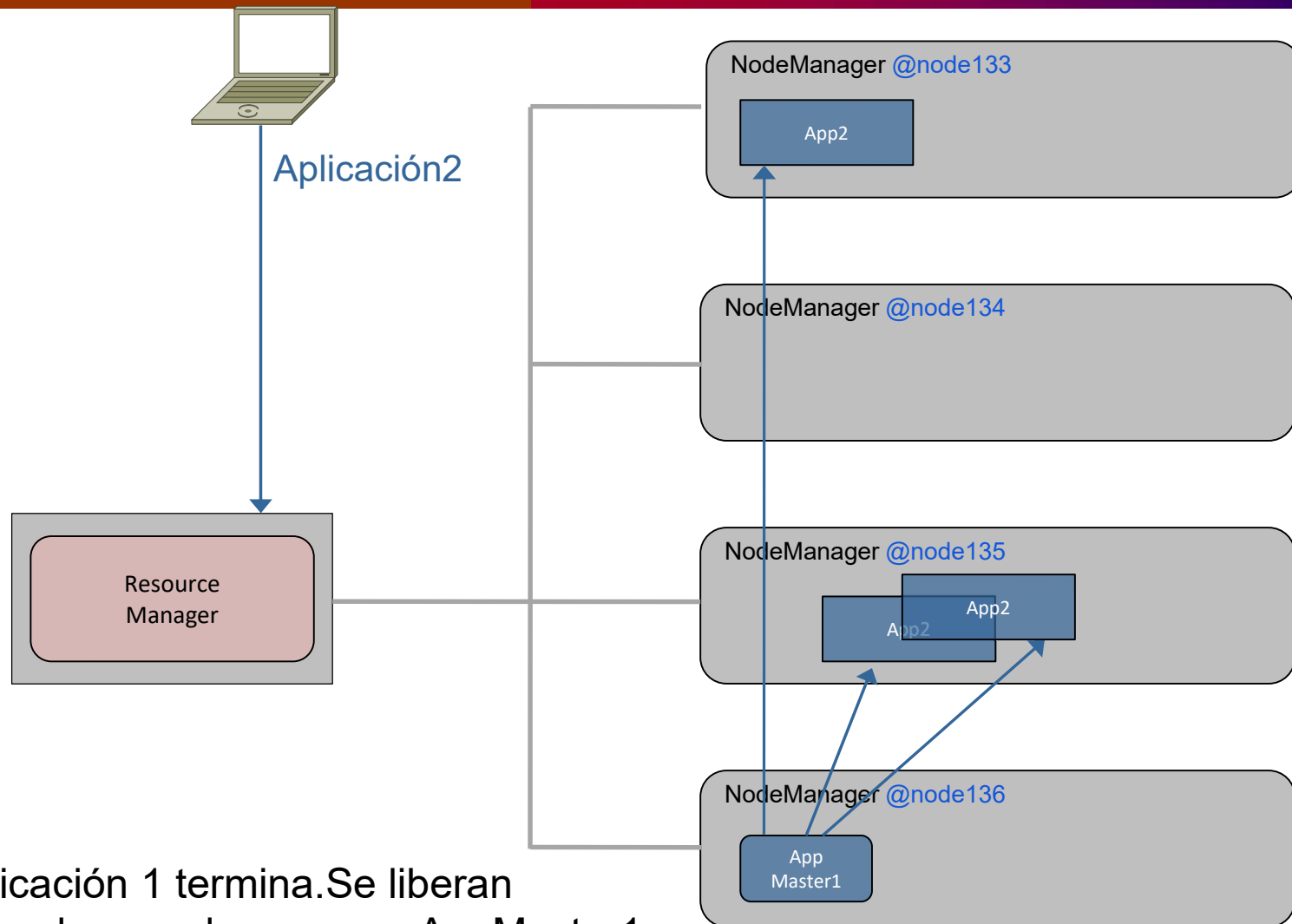
5. AppMaster 2  
solicita recursos

# Ejecutando una aplicación



6. AppMaster 2  
toma control de contenedores

# Ejecutando una aplicación



7. Aplicación 1 termina. Se liberan contenedores y desaparece AppMaster1

# Procesos complejos

- Un programa realista, requiere de un procesamiento de datos mucho más complejo que los ejemplos del curso.
  - No es recomendable hacer funciones map y reduce más elaboradas
  - Se diseña el programa como una secuencia de procesos map/reduce: Un flujo de trabajo
- Los flujos de trabajo suelen definirse con lenguajes más abstractos, como Pig y Hive
- El control de los flujos puede hacerse con cadenas lineales (chain mapper), o gráficas acíclicas dirigidas (DAG).
  - Lo más recomendable es utilizar OOZIE

# Referencias

- White, T., *Hadoop. The definitive guide*. O'Reilly, 3th Ed., 2012
- *Map Reduce and YARN*, Big Data University.  
<http://www.bigdatauniversity.com>

# Word Count Mapper

```
public static class Map extends MapReduceBase implements  
    Mapper<LongWritable,Text,Text,IntWritable> {  
    private static final IntWritable one = new IntWritable(1);  
    private Text word = new Text();  
  
    public static void map(LongWritable key, Text value,  
        OutputCollector<Text,IntWritable> output, Reporter reporter) throws  
        IOException {  
        String line = value.toString();  
        StringTokenizer = new StringTokenizer(line);  
        while(tokenizer.hasNext()) {  
            word.set(tokenizer.nextToken());  
            output.collect(word,one);  
        }  
    }  
}
```

# Word Count Reducer

```
public static class Reduce extends MapReduceBase implements  
    Reducer<Text,IntWritable,Text,IntWritable> {  
public static void map(Text key, Iterator<IntWritable> values,  
    OutputCollector<Text,IntWritable> output, Reporter reporter) throws  
    IOException {  
    int sum = 0;  
    while(values.hasNext()) {  
        sum += values.next().get();  
    }  
    output.collect(key, new IntWritable(sum));  
}  
}
```

# Word Count - Ejemplo

- Jobs controlados configurando *JobConfs*
- JobConfs son mapas de nombres de atributos a valores string
- El marco define atributos para controlar cómo se ejecuta un job
  - `conf.set("mapred.job.name", "MyApp");`
- Las aplicaciones pueden añadir valores arbitrarios al JobConf
  - `conf.set("my.string", "foo");`
  - `conf.set("my.integer", 12);`
- JobConf está disponible para todas las tareas



# Uniéndolo todo

- Se crea un programa *launch* para la aplicación
- Este programa configura:
  - Las funciones *Mapper* y *Reducer* que se utilizarán
  - El tipo de datos para los valores *key* y *value* de salida. Los tipos de entrada se infieren de *InputFormat*
  - La ubicación de los datos de entrada y salida
- El programa *launch* emite el job y típicamente espera que se haya completado

# Uniéndolo todo

```
JobConf conf = new JobConf(WordCount.class);  
conf.setJobName("wordcount");
```


```
conf.setOutputKeyClass(Text.class);  
conf.setOutputValueClass(IntWritable.class);
```

```
conf.setMapperClass(Map.class);  
conf.setCombinerClass(Reduce.class);  
conf.setReducer(Reduce.class);
```

```
conf.setInputFormat(TextInputFormat.class);  
Conf.setOutputFormat(TextOutputFormat.class);
```

```
FileInputFormat.setInputPaths(conf, new Path(args[0]));  
FileOutputFormat.setOutputPath(conf, new Path(args[1]));
```

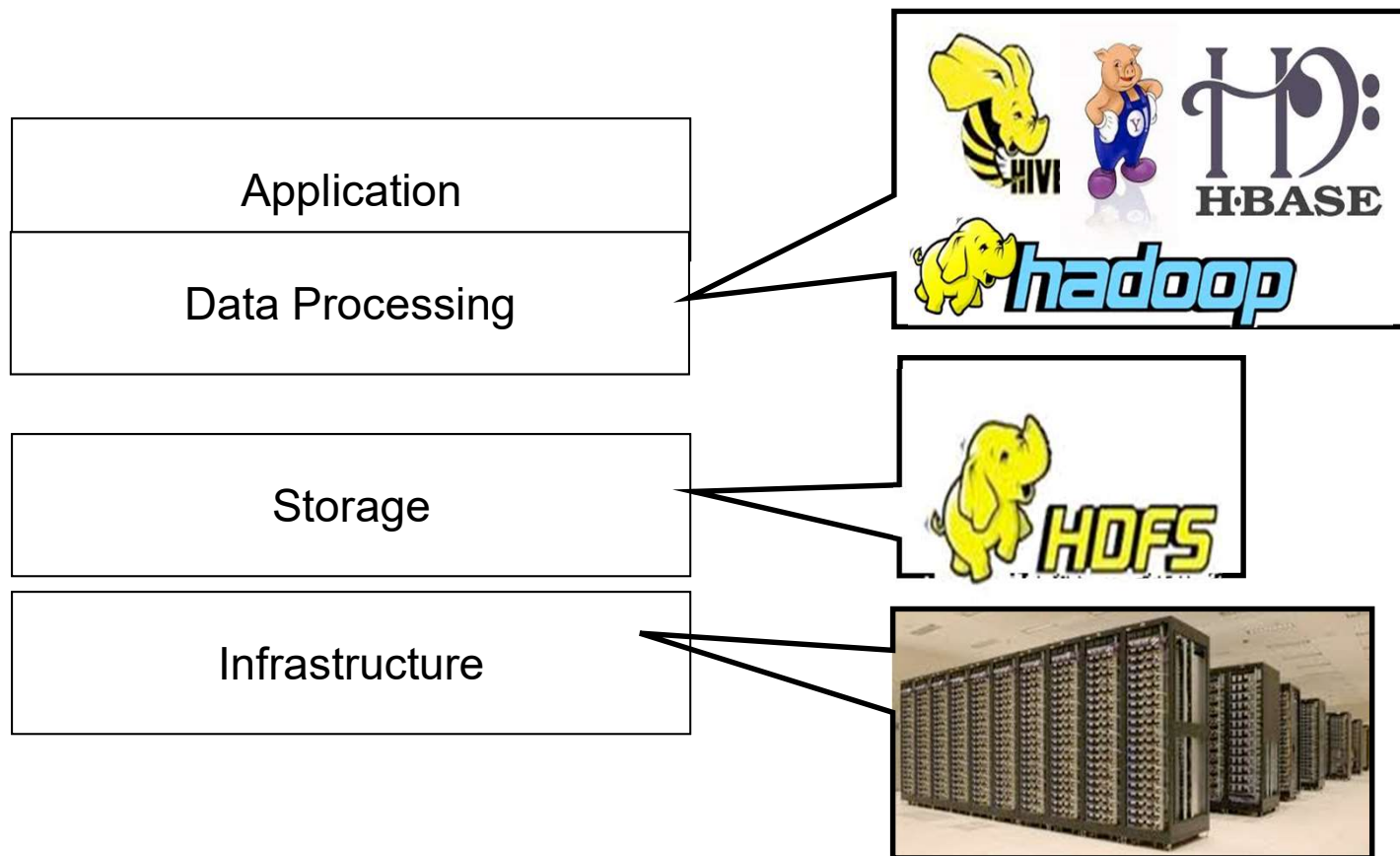
```
JobClient.runJob(conf);
```



Necesidad de procesamiento  
(casi) en tiempo real

# La infraestructura para Big Data y analítica hasta ahora:

... muy apropiada para procesamiento de grandes volúmenes *almacenados*. Buen desempeño pero alta latencia, **lenta**

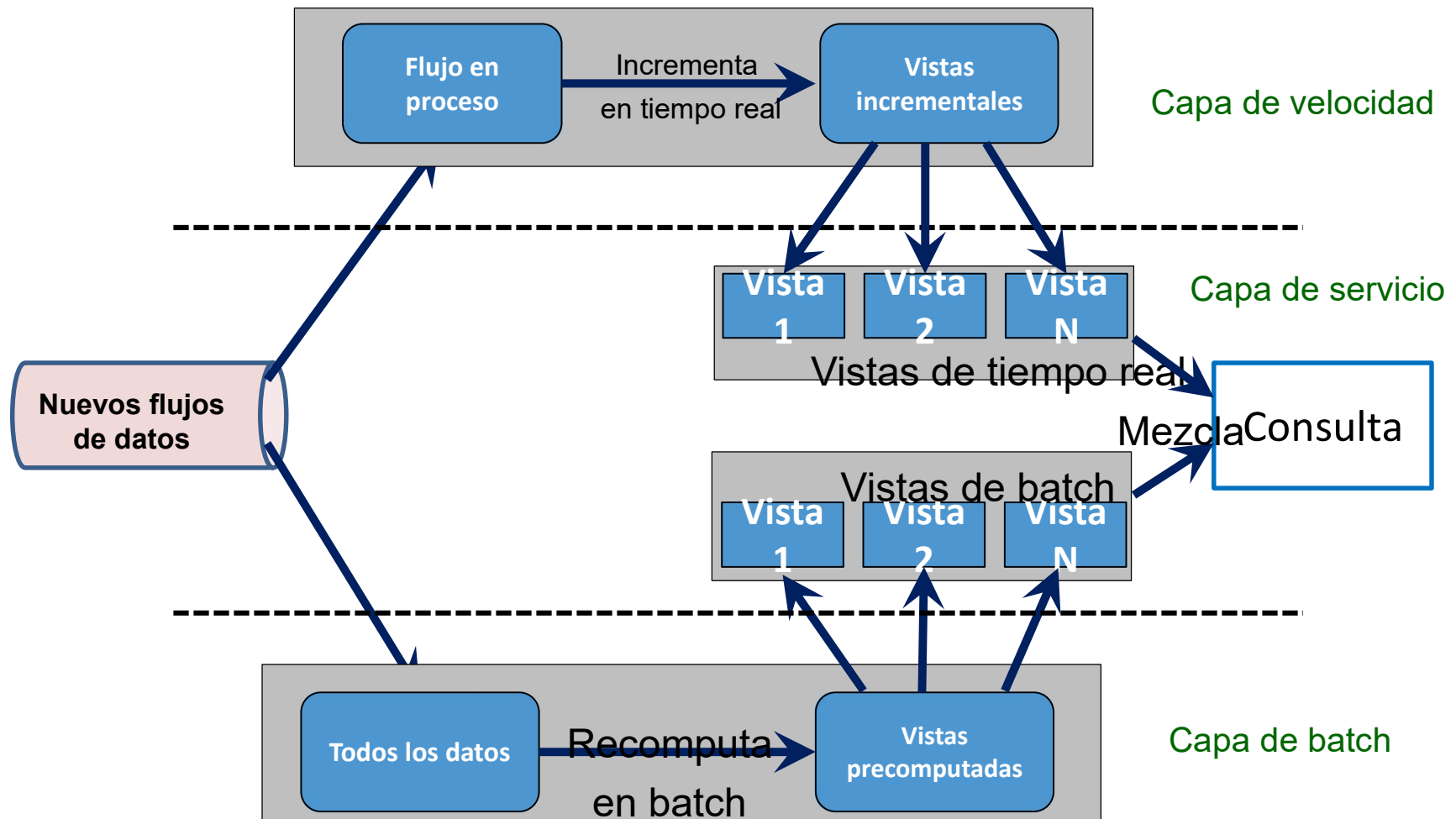


# El problema:

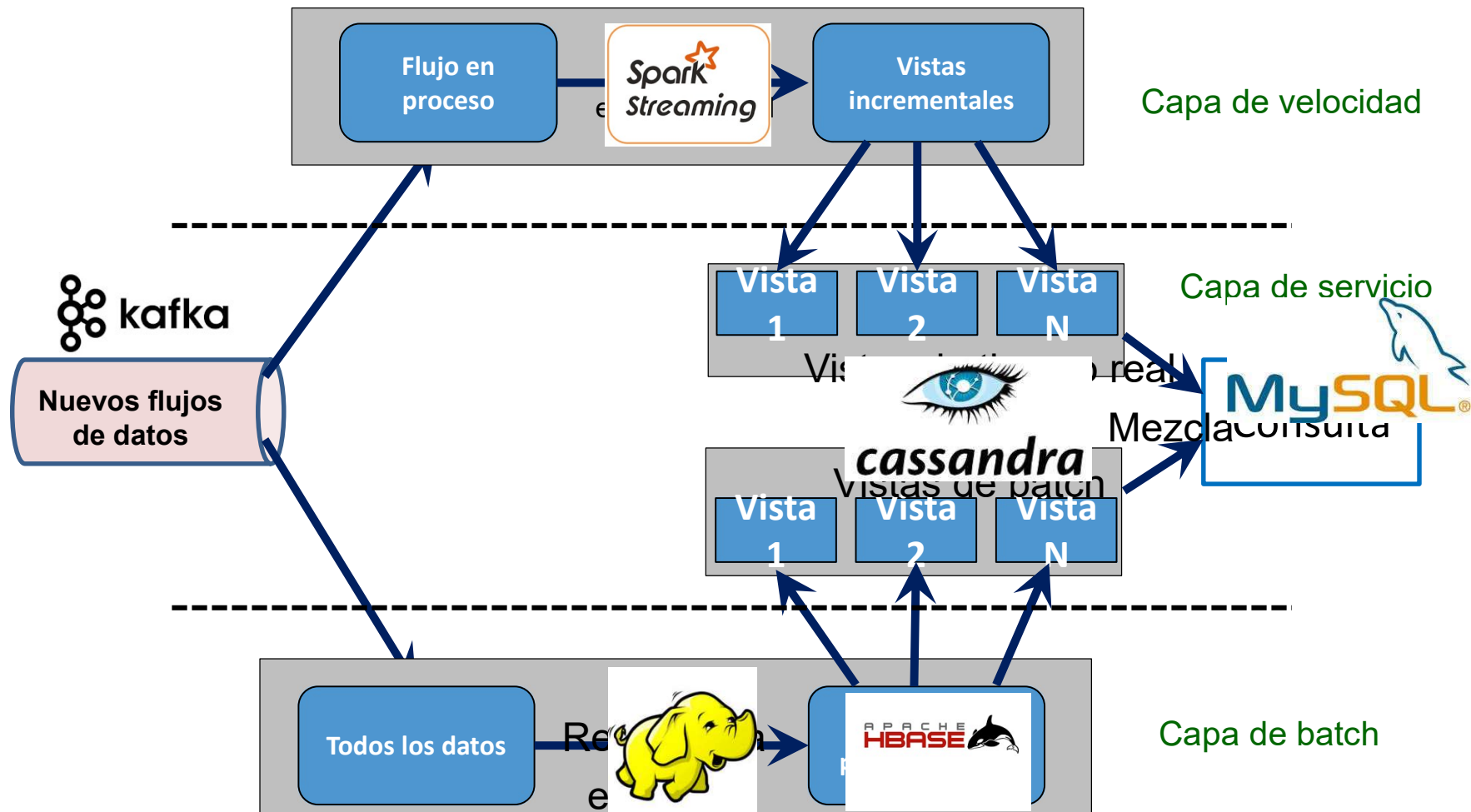
- Muchas aplicaciones importantes deben procesar *grandes volúmenes* de **flujos de datos en vivo** y ofrecer resultados casi en tiempo real
  - Tendencias en redes sociales
  - Analíticos en sitios web
  - Sistemas de detección de intrusiones
  - ...
- Aún requiere de clusters grandes para soportar la carga de trabajo
- .... Pero con latencias del orden de segundos



# Arquitectura Lambda



# Arquitectura Lambda



# Apache Spark

- Motor de procesamiento más versátil que tan solo “*mappers*” y “*reducers*”. Define un amplio conjunto de operaciones (transformaciones y acciones)
  - Las operaciones pueden combinarse en cualquier orden
- Software libre – es un proyecto Apache
- Soporta Java, Scala y Python
- Constructor clave: Resilient Distributed Dataset (RDD) que se ejecutan *en memoria* en cada nodo en el cluster
  - Representan datos o transformaciones sobre datos
  - Más apropiados cuando el modelo requiere aplicar las mismas operaciones a todos los elementos del dataset



# Competencia Gray sort

	Hadoop MR Record	Spark Record (2014)	Spark-based System 3x faster with 1/10 # of nodes
Data Size	102.5 TB	100 TB	
Elapsed Time	72 mins	23 mins	
# Nodes	2100	206	
# Cores	50400 physical	6592 virtualized	
Cluster disk throughput	3150 GB/s (est.)	618 GB/s	
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network	
<b>Sort rate</b>	<b>1.42 TB/min</b>	<b>4.27 TB/min</b>	
<b>Sort rate/node</b>	<b>0.67 GB/min</b>	<b>20.7 GB/min</b>	

Sort benchmark, Daytona Gray: sort of 100 TB of data (1 trillion records)

<http://databricks.com/blog/2014/11/05/spark-officially-sets-a-new-record-in-large-scale-sorting.html>

Spark  
SQL

Spark  
Streaming

MLlib  
(machine  
learning)

GraphX  
(graph)

Apache Spark

# Spark Streaming

- Ambiente para procesar flujos de datos a gran escala
- Puede escalar a cientos de nodos
- Latencia en el orden de los segundos
- Se integra con las plataformas Spark para procesamiento en batch e interactivas
- Provee una API sencilla para implementar algoritmos complejos
- Puede tomar flujos de conectores como Kafka, Flume y ZeroMQ