Diplomado en Big Data y Data Science

Fundamentos de Big Data

ITAM
Extensión Universitaria
y Desarrollo Ejecutivo

Temario

- Big Data
  - Conceptos básicos
  - Infraestructura para Big Data
- Hadoop, Map-Reduce
  - Hadoop V1
  - Ecosistema
- Arquitecturas para procesamiento de datos en tiempo real
- Hadoop v2. YARN
- Hive
- Introducción a la gestión de infraestructuras

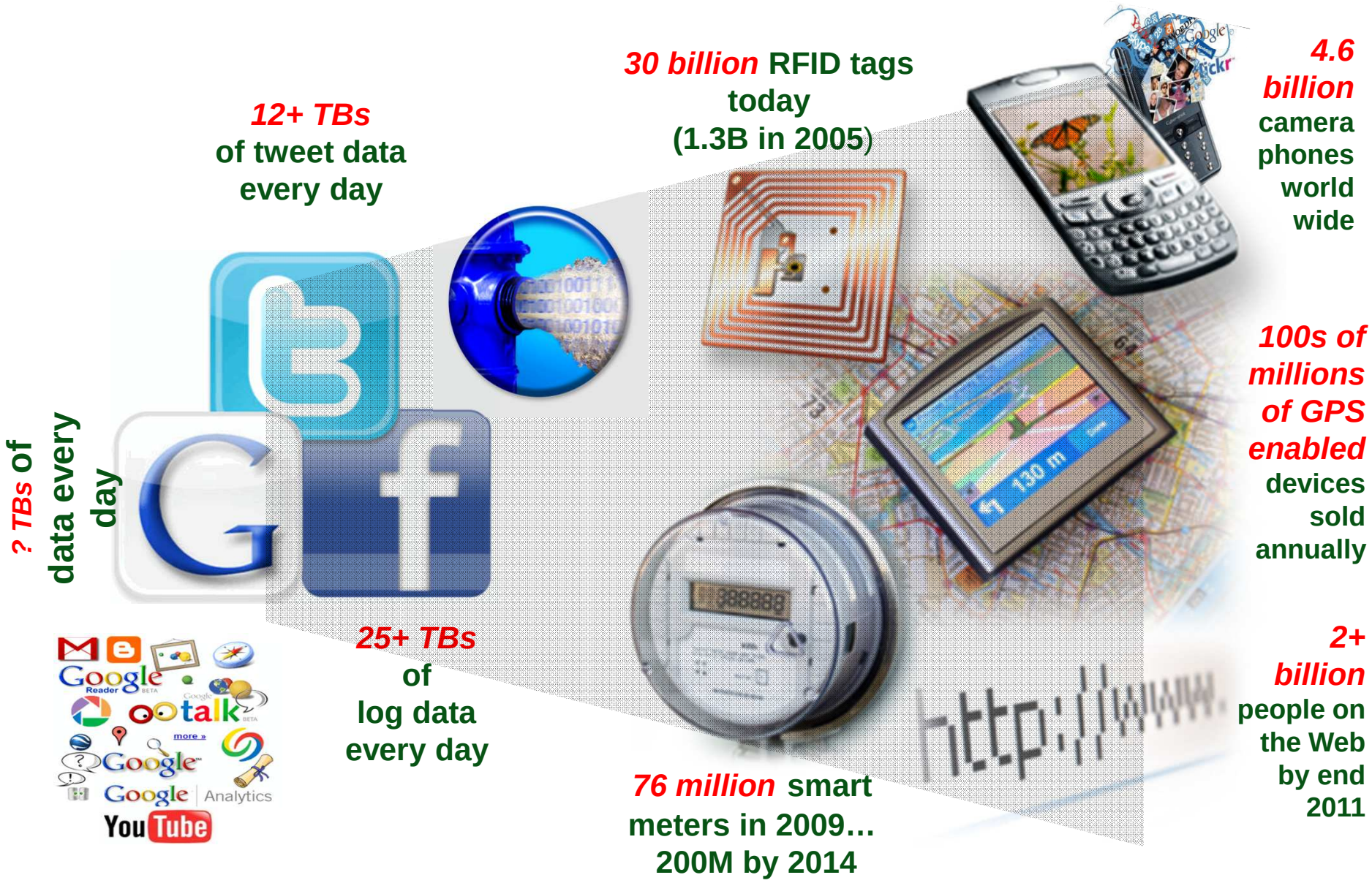
Big Data y Analítica

- La nueva era caracterizada por la abundancia de datos
  - Ha alcanzado todos los sectores de la economía
  - 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

Fuente: W. Peres, CEPAL

Fuentes de datos

Fuentes de datos



Fuente: IBM

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

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

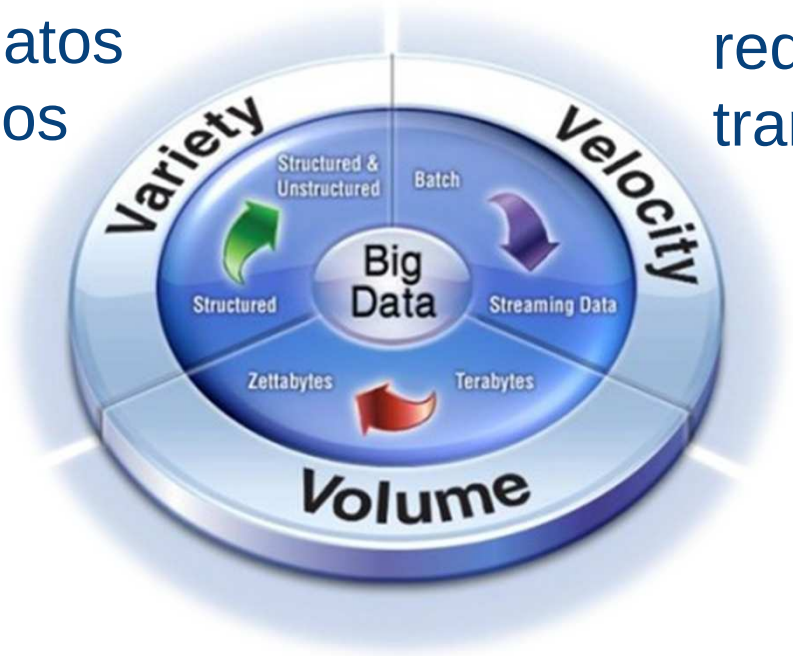
¿Qué datos usa su empresa?

Fuentes internas	
Transaccionales	88%
Bitácoras	73%
eMail	57%
Fuentes externas	
Redes sociales	43%
Audio	38%
Imagen y video	34%

Fuente: IBM IBV

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

Ejemplo: Redes de sensores

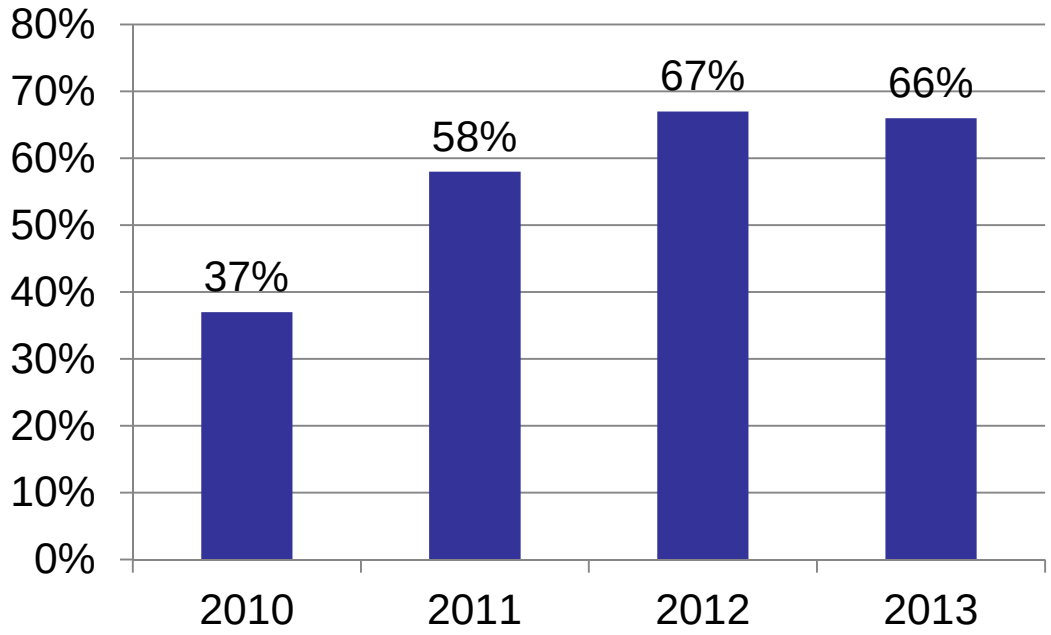
- Volumen → Miles de millones de sensores
- Velocidad → … deben ser procesados casi en tiempo real
- Variedad → … gran variedad de tipos y de redes

TIC clásicas

- Servidores
- BD relacionales
- Data Warehouse/ Data Marts

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

Evolución Big Data y Analítica



Porcentaje de empresas que ve en la analítica una fuente de ventaja competitiva

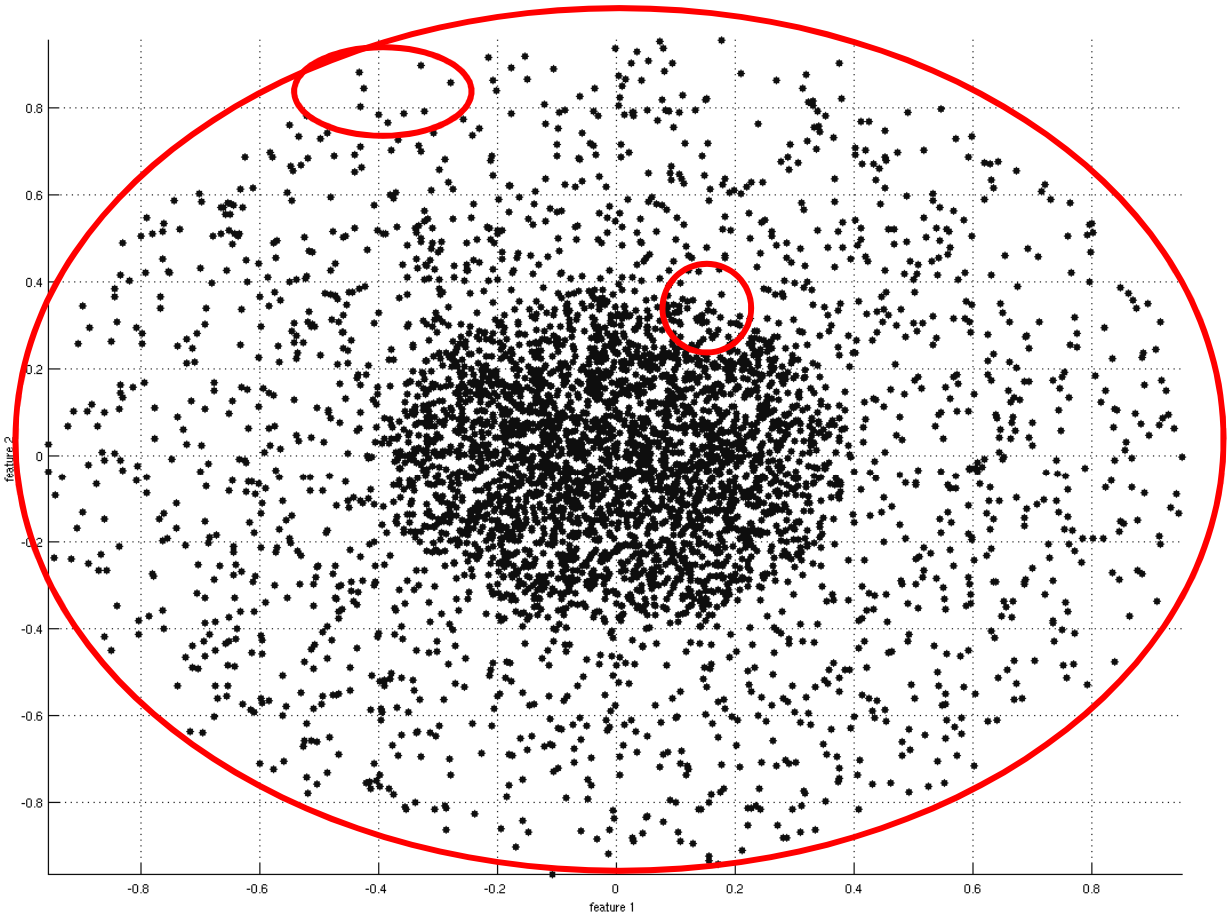
N = 2,037

91 de 100 ejecutivos de Fortune 1000 están invirtiendo en BD&A

88% considera una inversión de $ 1M USD para 2016

Fuente: MIT Sloan/SAS 2014

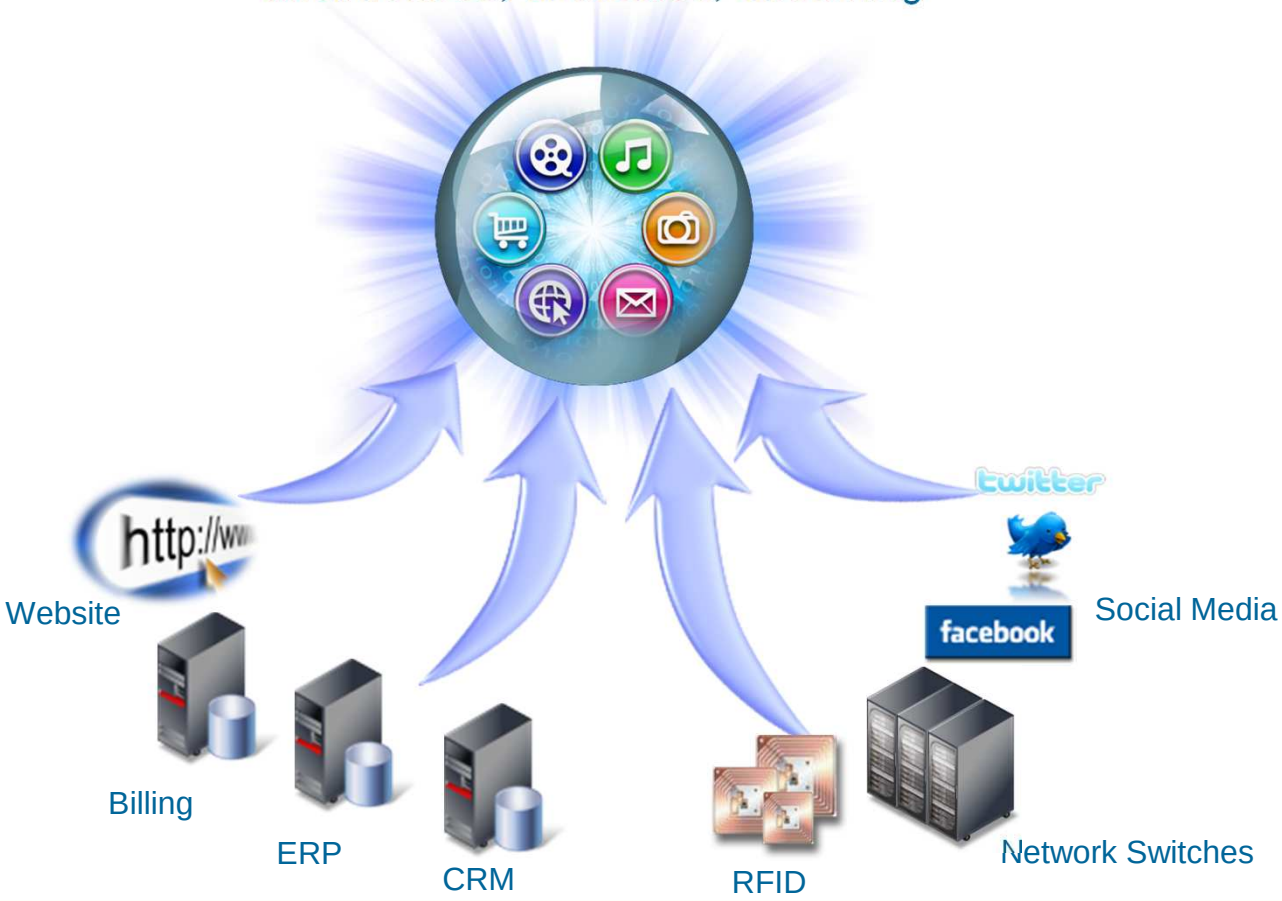
Una gran ventaja de Big Data



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



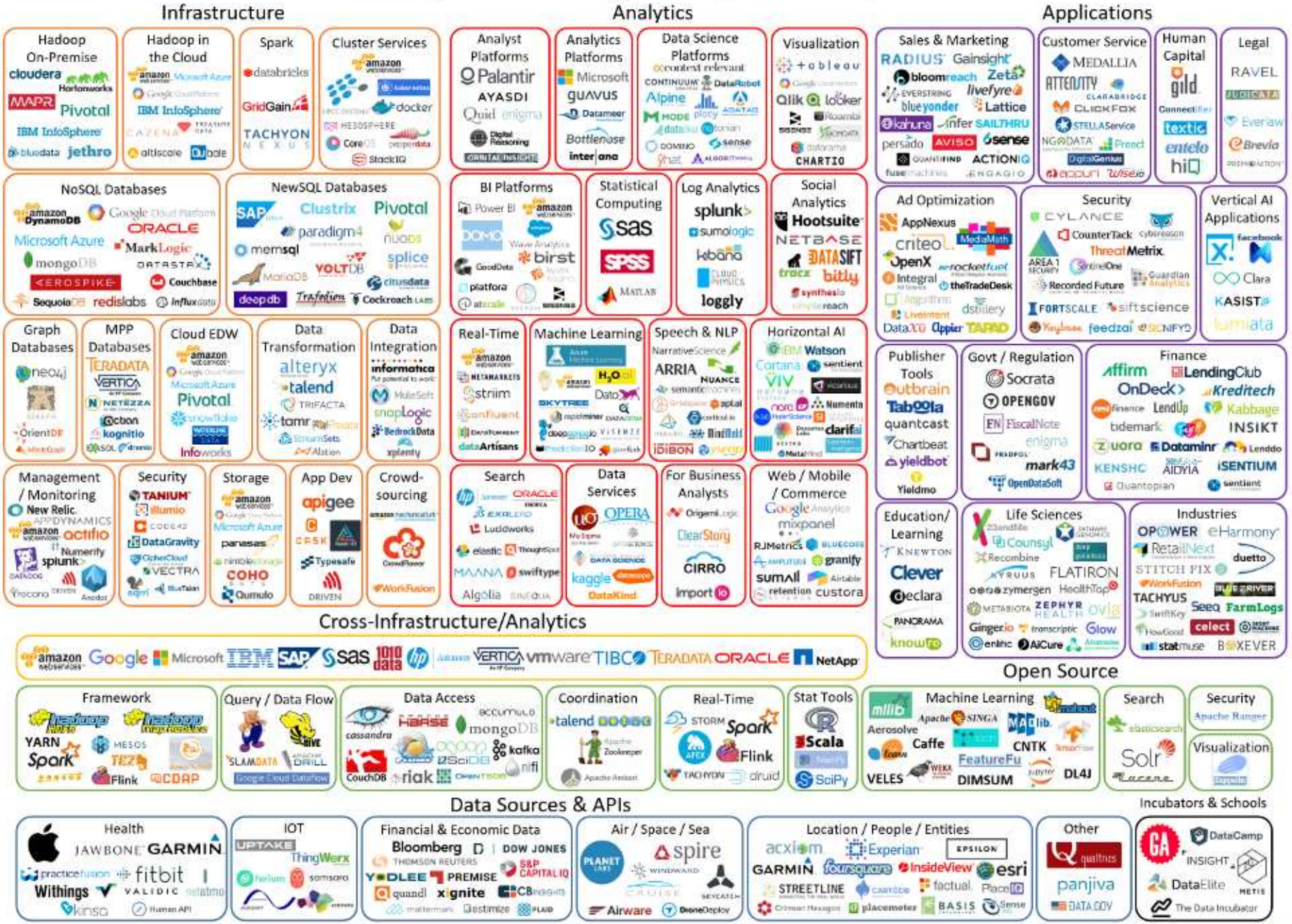
Fuente: IBM

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 Landscape 2016 (Version 2.0)



Last Updated 2/12/2016

FIRSTMARK

Una breve introducción a Hadoop y MapReduce

ITAM
Extensión Universitaria
y Desarrollo Ejecutivo

¿Cómo puedo transportar esta carga?

HDFS y MapReduce

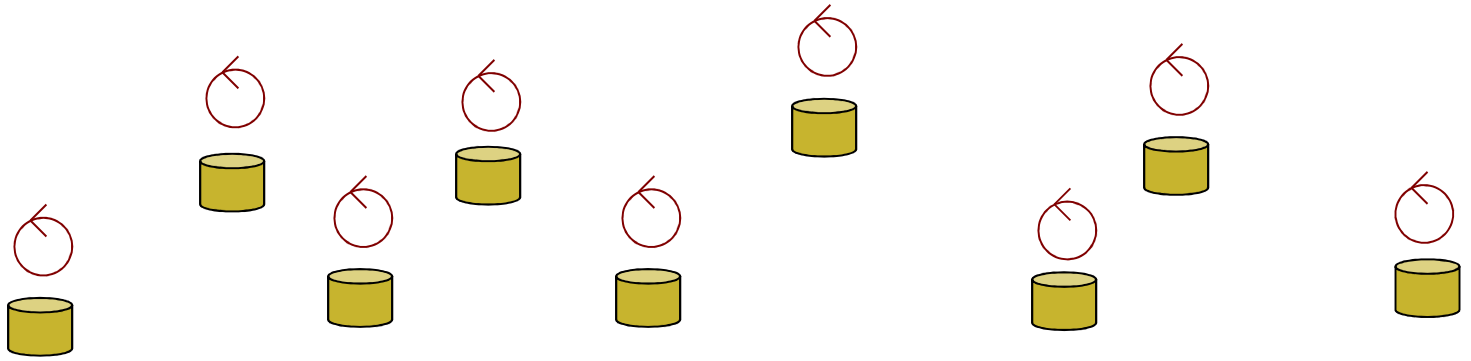
- Procesamiento de grandes volúmenes de información requiere de una gran capacidad de procesamiento y almacenamiento
- Mainframes, supercomputadoras, SANs del orden de Petabytes, excesivamente costosas

- Google observó que la gran mayoría de las operaciones requeridas eran bastante simples

→ *Sistema de archivos distribuido y librería de instrucciones relativamente simples*

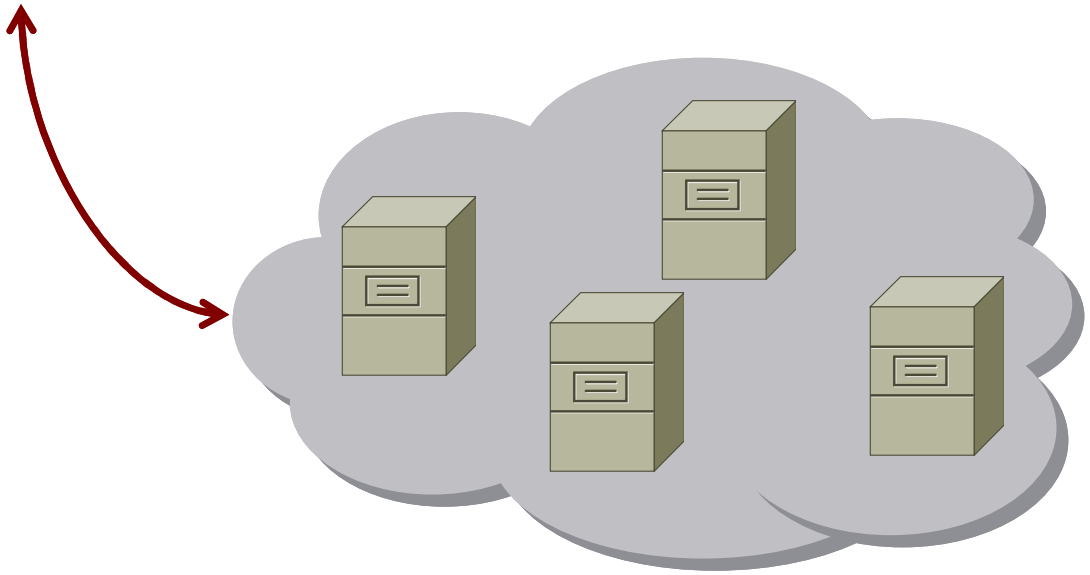
Hadoop y MapReduce

- 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)

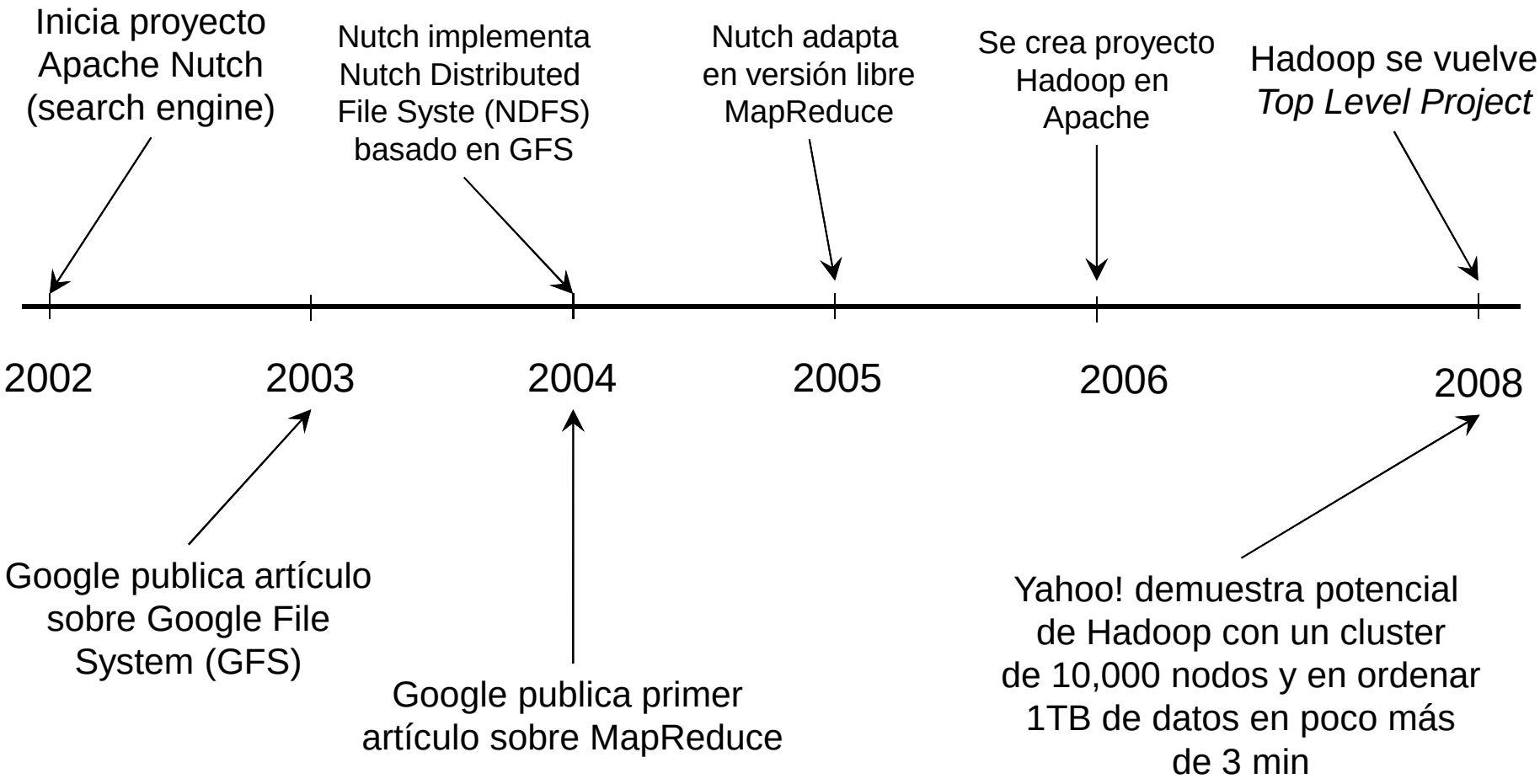


HDFS
Almacenamiento confiable y de alta capacidad

MapReduce
Procesamiento distribuido



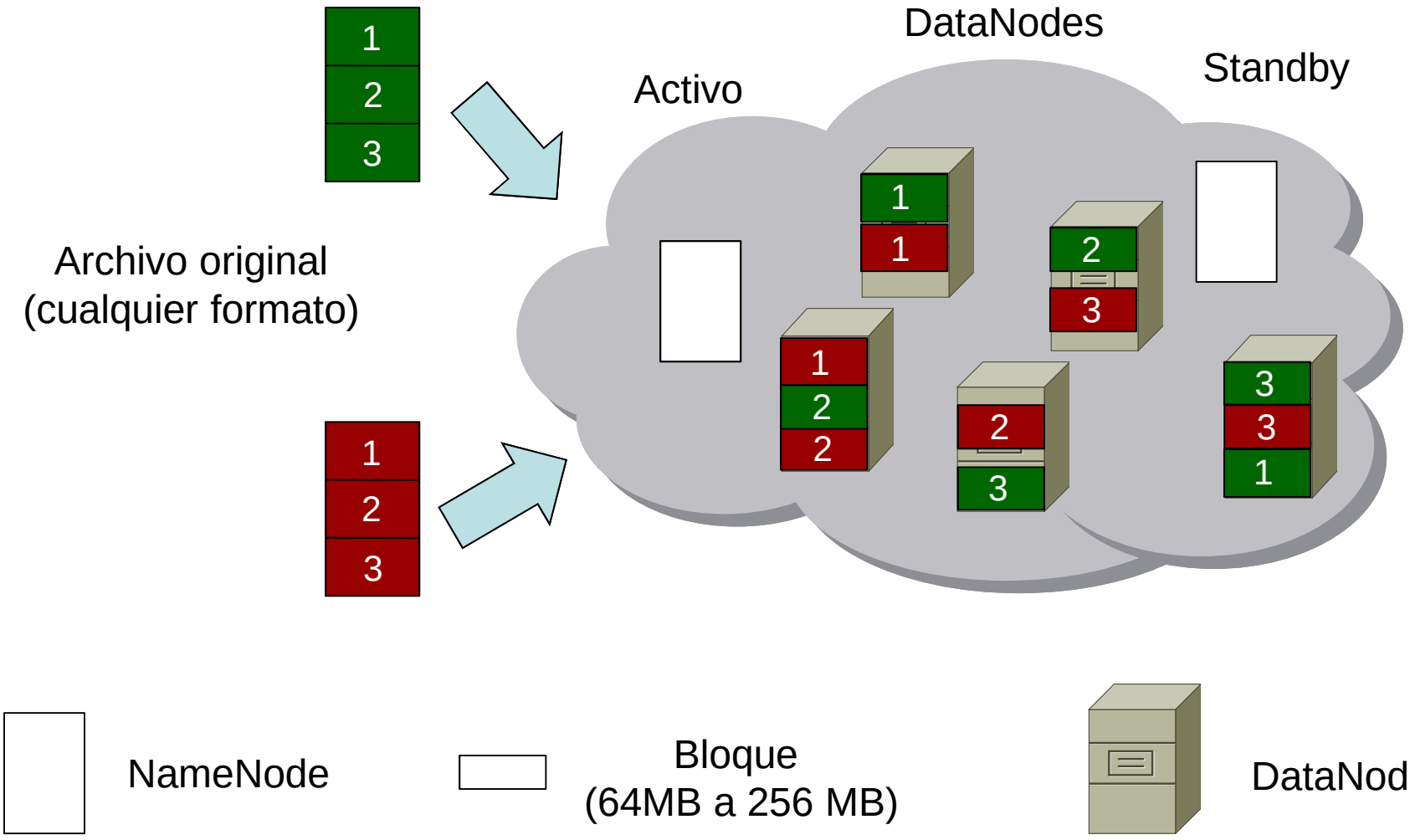
Algunos hitos



- 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 clusters de hardware convencional
  - Confiablidad basada en replicación
  - Procesamiento masivo a bajo costo
- Permite almacenar datos distintos (estructurados, semi/no estructurados)
  - Es un sistema de archivos
- Sumamente escalable
- Optimizado para procesamiento en lotes
  - Alta latencia. No adecuado para OLTP, OLAP, transacciones en t. Real
  - Modelo concebido para *write once-read many*

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*
  - Tres réplicas por omisión
- Cliente interactúa con NameNode y con DataNodes
  - Desempeño escala casi linealmente con el número de DataNodes
  - Accesso 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

Comandos *shell* HDFS

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

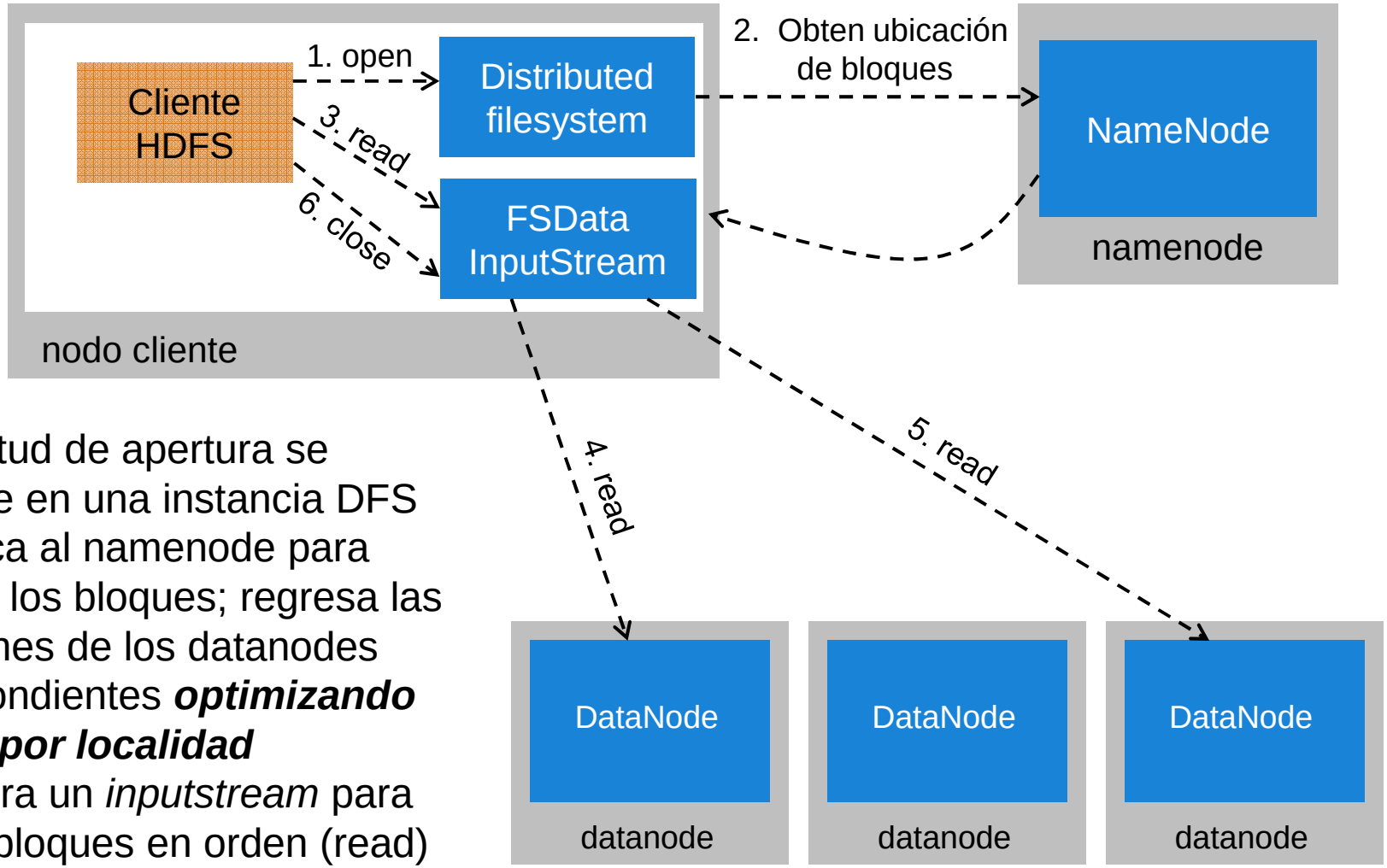
```
hadoop fs –ls
hadoop fs –put [-f] archivo <archivo>
hadoop fs –tail Archivo
hadoop fs –mkdir Dir
hadoop fs –mv Archivo Dir
hadoop fs –get Dir/Archivo
hadoop fs –rm Dir/Archivo
```

cat
chgrp
chmod
stat

copyFromLocal
copyToLocal
getMerge
Setrep

Versión 2: hdfs dfs -<cmd>

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

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

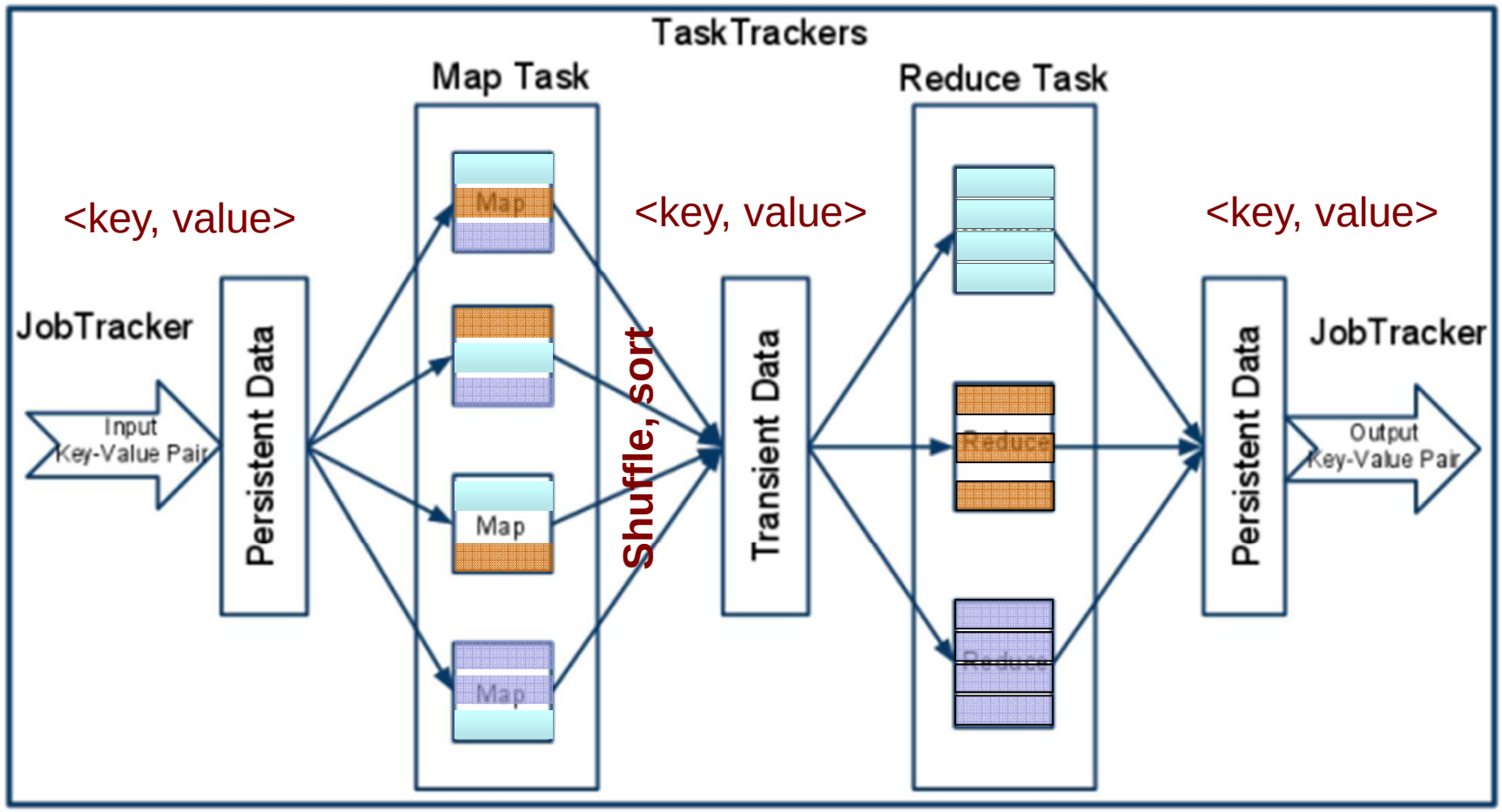
MapReduce

- 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)

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

- Modelo de programación para cómputo distribuido eficiente
- Flujo de datos similar a pipeline de Unix:

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

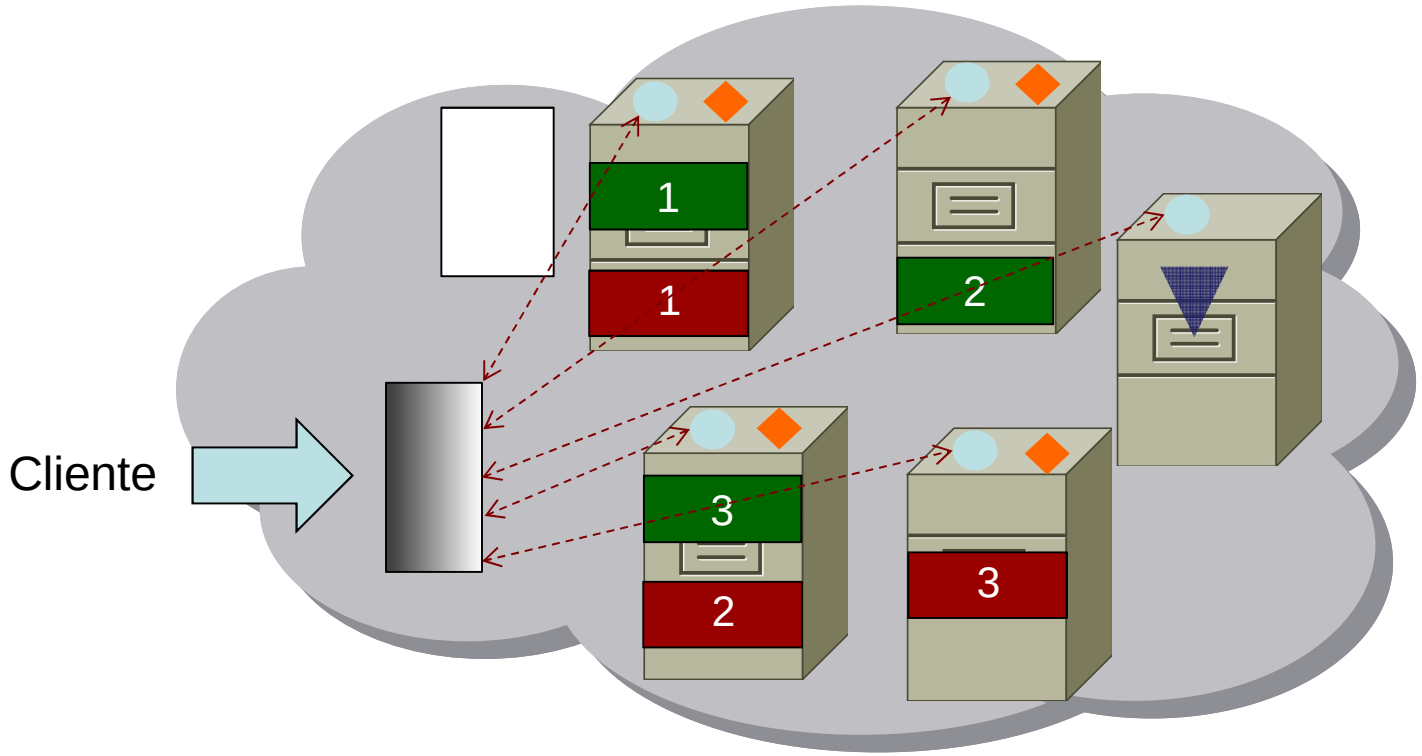
Input | **Map** |Shuffle & Sort | **Reduce** | **Output**

- 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

Arquitectura MapReduce V1

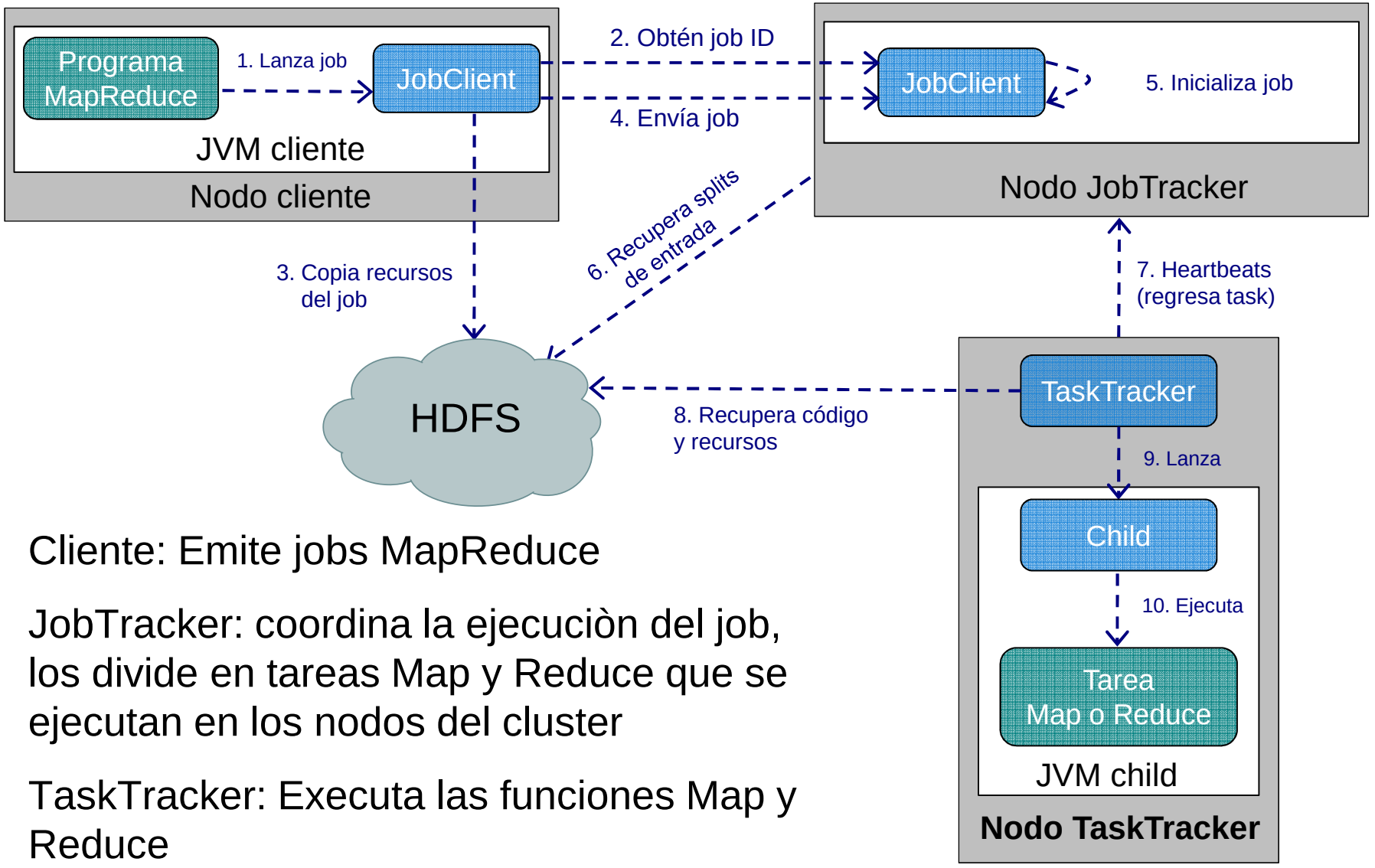
- Arquitectura maestro/esclavo
  - Un solo maestro (JobTracker) controla la ejecución de múltiples esclavos (los TaskTrackers)
- JobTracker
  - Acepta jobs MapReduce enviados por los clientes
  - Lanza tareas map y reduce a los nodos Task Tracker
  - Monitorea las tareas y el estado de los Task Trackers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

Procesos MapReduce V1



JobTracker TaskTracker Mapper Reducer

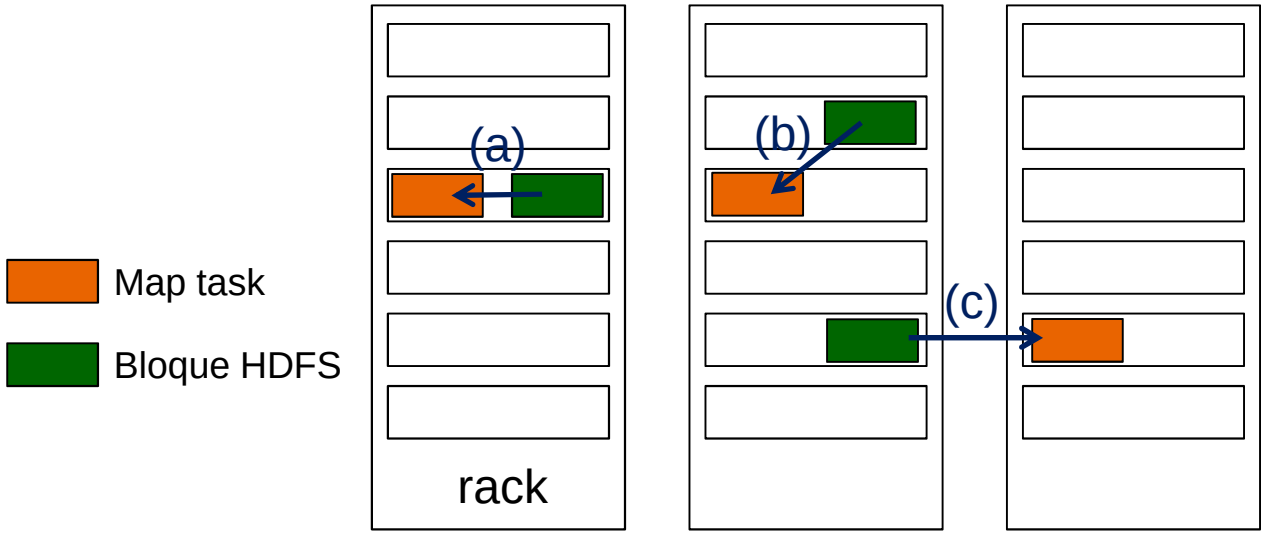
Ejecución de un job (MapReduce V1)



- Cliente: Emite jobs MapReduce
- JobTracker: coordina la ejecuciòn del job, los divide en tareas Map y Reduce que se ejecutan en los nodos del cluster
- TaskTracker: Executa 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 tomaría de un nodo en otro rack (*off-rack)*

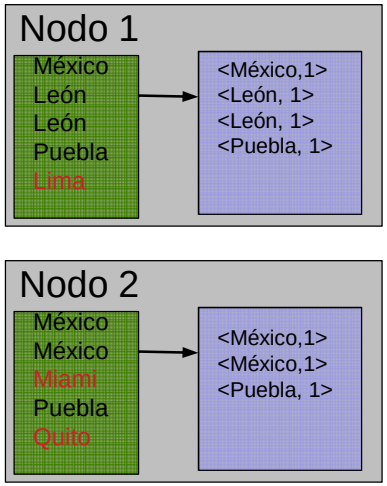


Funciones *combiner*

- 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, …).

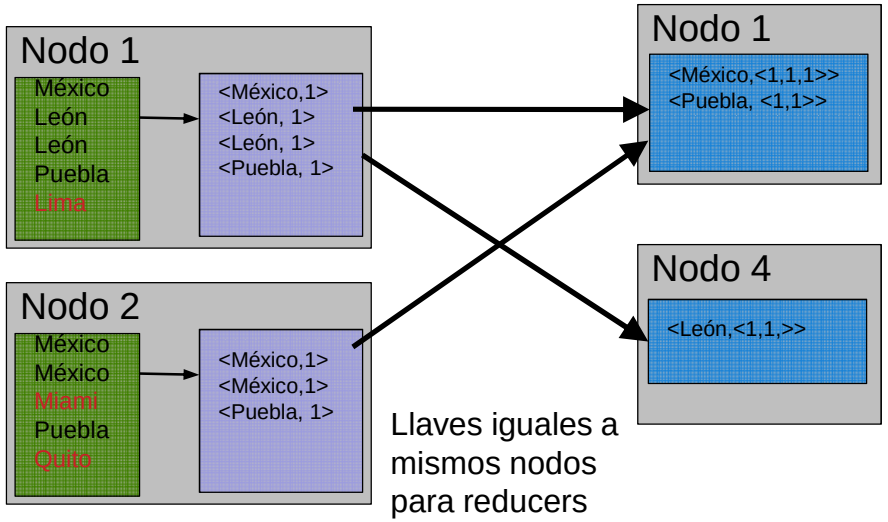
Ejemplo MapReduce. Word count

Map



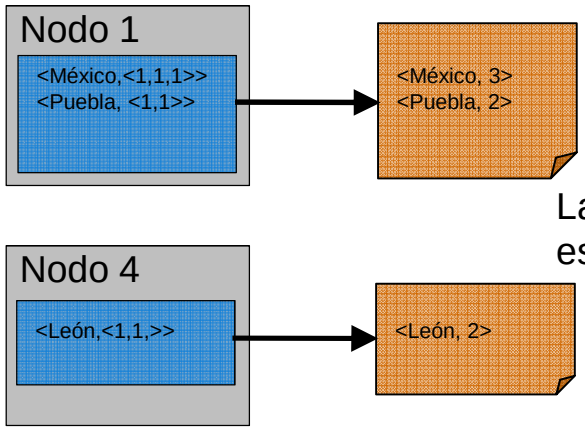
Las tareas Map se ejecutan localmente en cada split

Shuffle



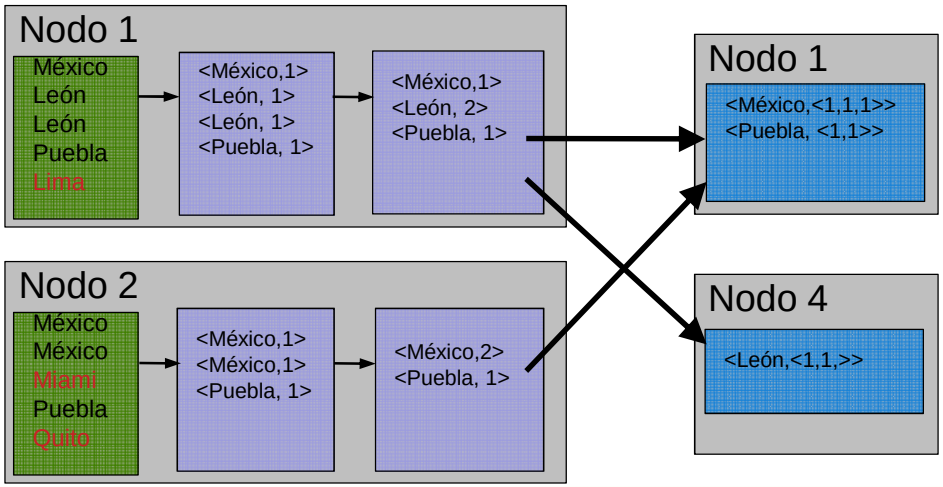
Llaves iguales a mismos nodos para reducers

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

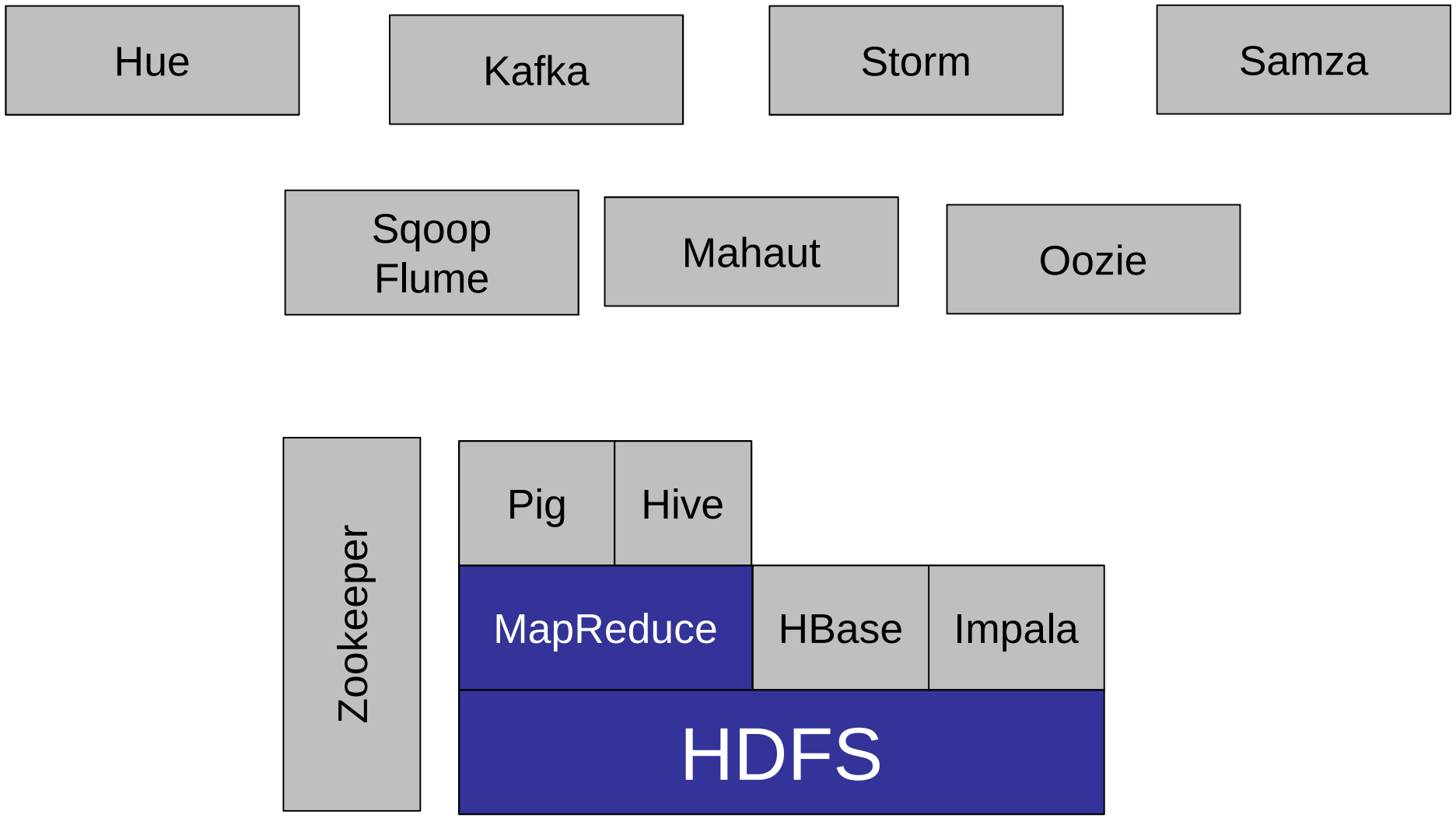
Sin embargo, en el curso utilizaremos un API (Hadoop Streaming) para especificar funciones map y reduce como código independiente, y en otros lenguajes (python)

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

Un ejemplo de código en Java

Ecosistema Hadoop - BigData



Fuente: Cloudera

- 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
- Mahaut – 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

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

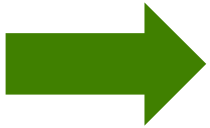
Uniéndolo todo

```
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);
```

Arquitecturas para procesamiento en tiempo (casi) real “Datos en movimiento”

Disminuir la latencia

- Hadoop fue concebido para procesamiento por lotes. La latencia no era relevante
- Grandes volúmenes de datos provienen de fuentes que los generan continuamente
  - sensores, redes sociales, datos geo-referenciados, …
  - Para muchas aplicaciones, el valor de estos datos depende de la oportunidad de explotarlos rápidamente
    - Sistemas de recomendaciones, análisis de sentimientos, procesamiento de eventos complejos



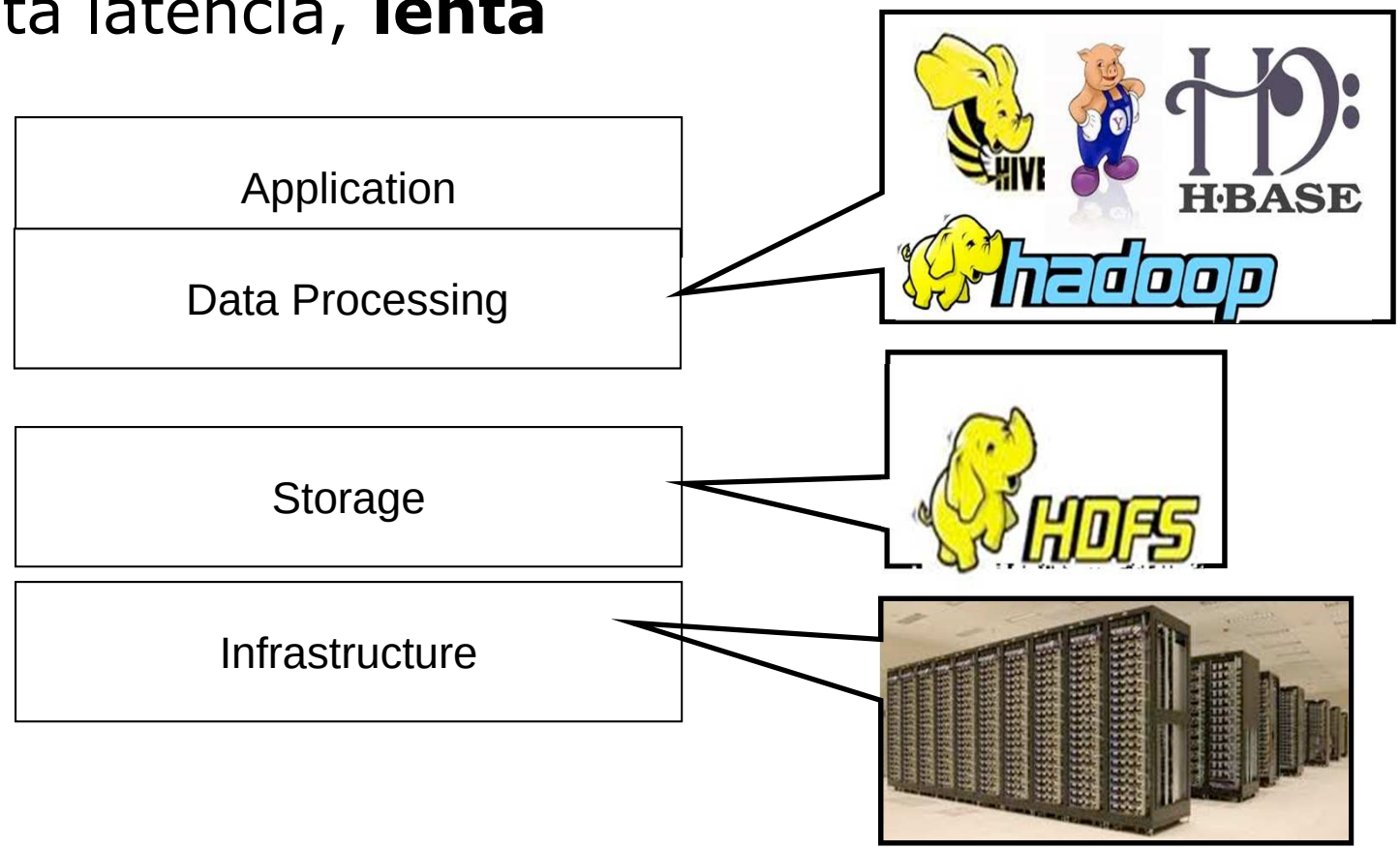
Procesamiento en memoria (Spark)

Procesamiento por flujos (streams)

Arquitecturas para Big Data

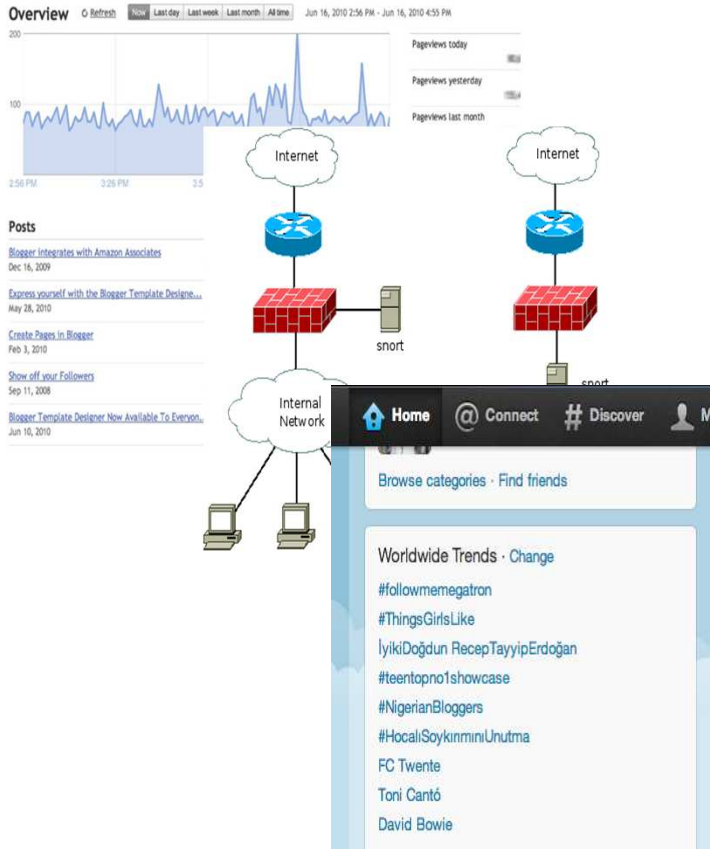
La infraestructura para Big Data y analítica hasta ahora:

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



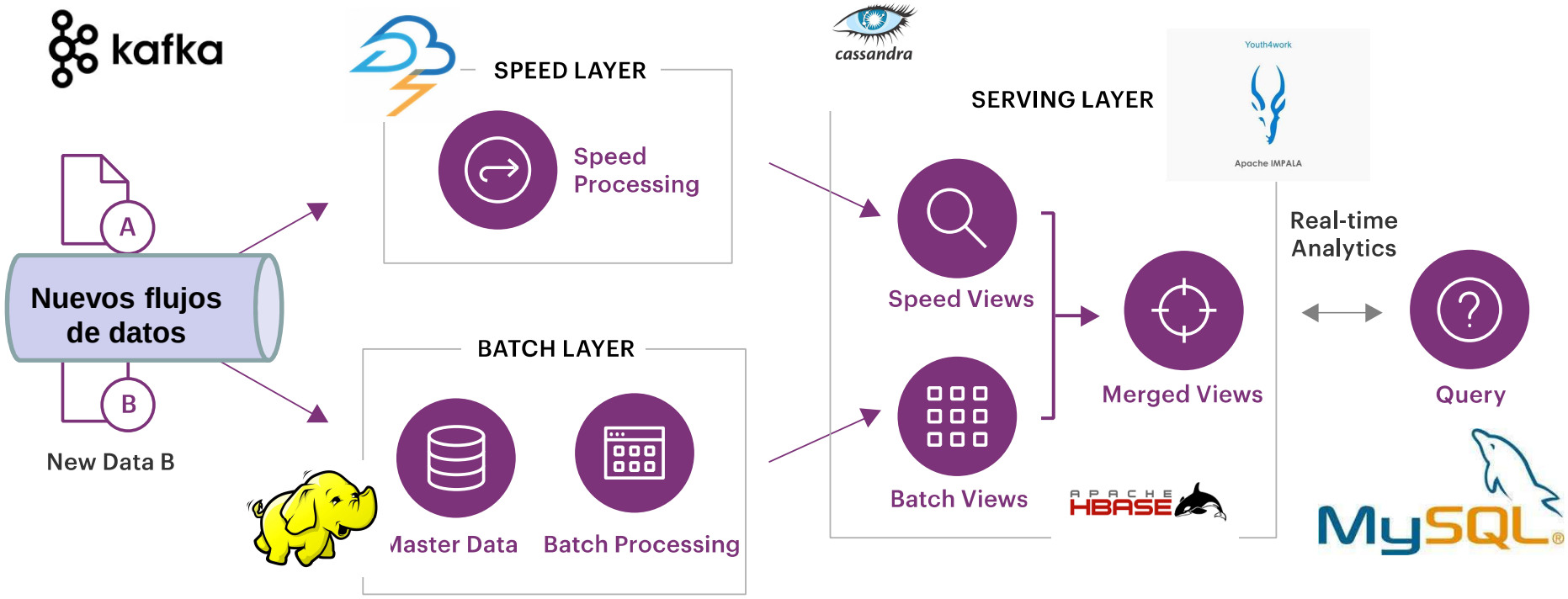
El problema:

- 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



Detección de fraudes

Atención a clientes

Ciber-crimen

Ofertas en tiempo real

Análisis de riesgos

Seguridad y cumplimiento

Retención de clientes

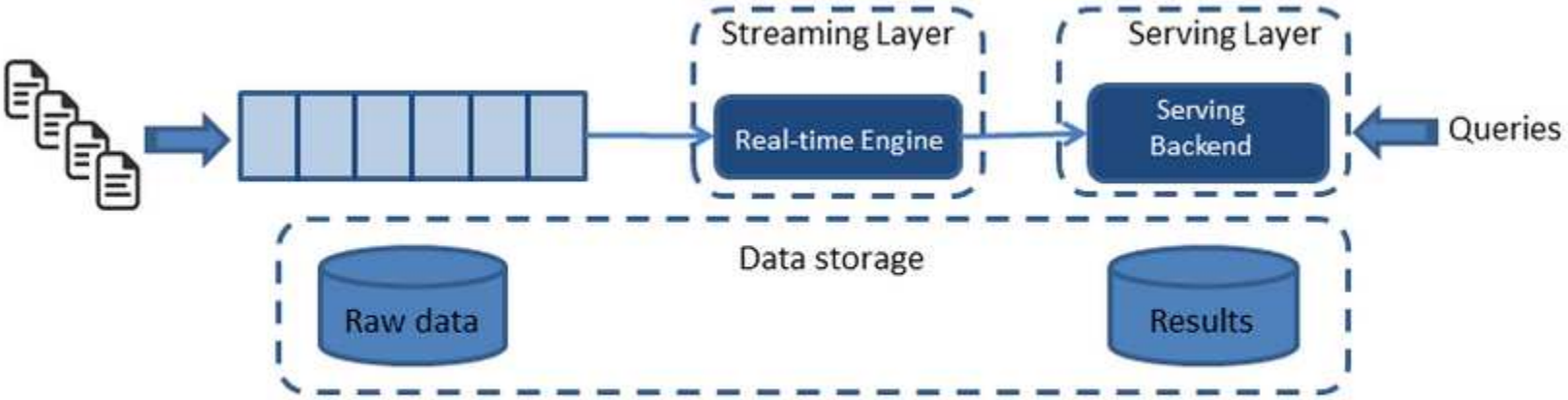
Auditoría y gobernanza

Arquitectura Lambda

- ✓ Los datos se almacenan en una capa persistente. *No se modifican*
- ✓ Dos niveles de procesamiento: Alta velocidad (streaming) y baja velocidad (batch)
  - Procesamiento en paralelo
- ✓ La capa de servicio "toma" información de las dos rutas.
  - Por ejemplo, streaming para tableros en tiempo real, batch para reportes

- x Es muy difícil mantener coordinación en el desarrollo de dos sistemas
- x Limita funcionalidad y escalabilidad

Arquitectura Kappa



- Si la capa batch sirve para almacenar datos históricos (que ahora hace el data lake) y para reprocesarlos (por ejemplo, para correr nuevos modelos de Machine Learning) puede ser mucho más eficiente reprocesar los datos cambiando el bloque de alta velocidad, que mantener dos sistemas

- Un batch es parte de un flujo con un principio y un fin. Si el flujo está almacenado, el lote puede reproducirse
- **Debe acotarse el tiempo de retención de los datos potenciales a ser reprocesados**

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

Apache Spark

- Procesamiento distribuido de bloques de datos (RDD, Resilient distributed datasets) *en memoria*
- Soporta modelo de procesamiento en lotes y en streaming (y modelos de grafos y de aprendizaje de máquina)
- Simplifica despliegue de arquitecturas Lambda

- Su penetración es tan grande, que dedicaremos una sección aparte en el Diplomado para conocer esta arquitectura

Competencia Gray sort

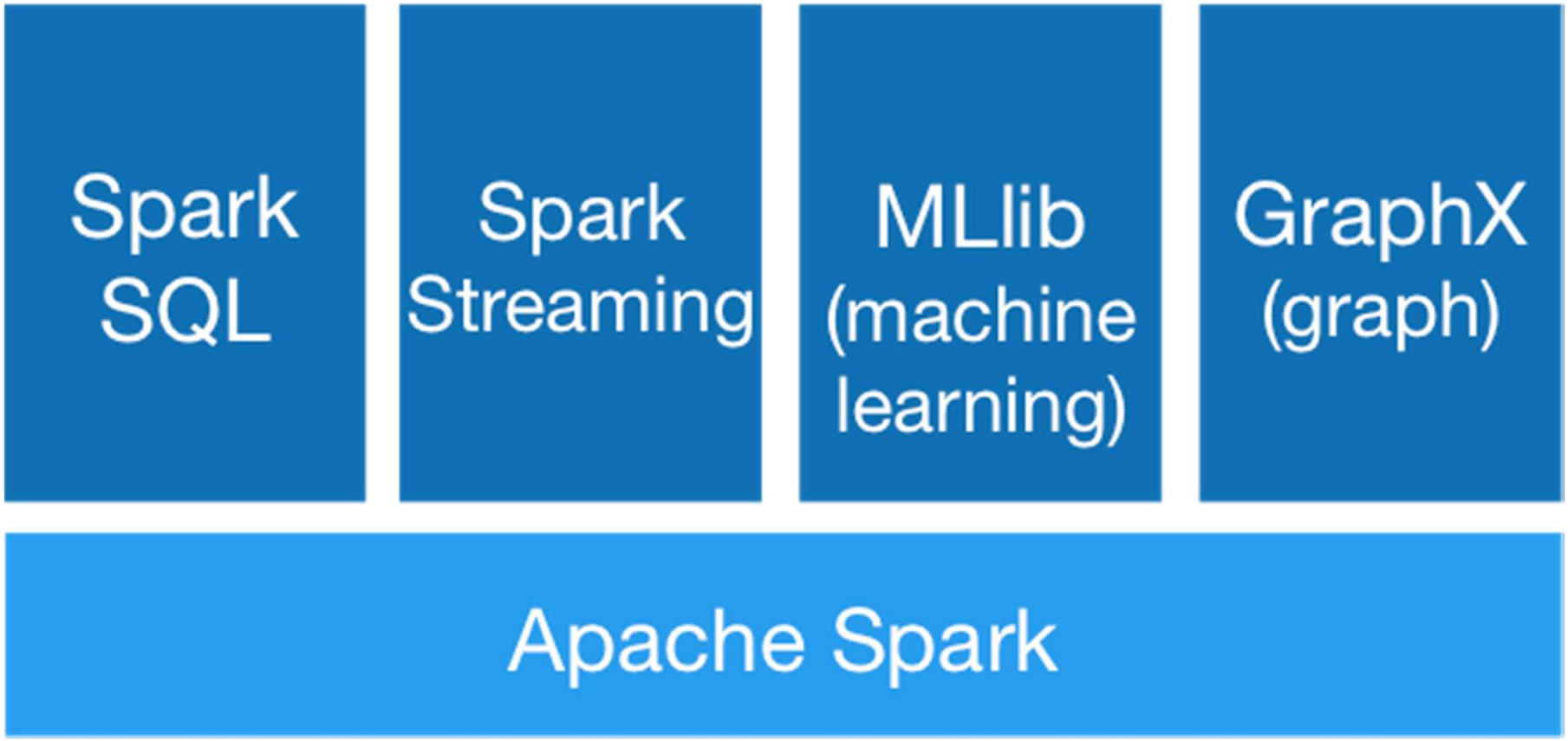
	Hadoop MR Record	Spark Record (2014)
Data Size	102.5 TB	100 TB
Elapsed Time	72 mins	23 mins
# Nodes	2100	206
# Cores	50400 physical	6592 virtualized
Cluster disk throughput	3150 GB/s (est.)	618 GB/s
Network	dedicated data center, 10Gbps	virtualized (EC2) 10Gbps network
Sort rate	**1.42 TB/min**	**4.27 TB/min**
Sort rate/node	**0.67 GB/min**	**20.7 GB/min**

Spark-based System 3x faster with 1/10 # of nodes

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 Ecosystem



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, asigancián 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

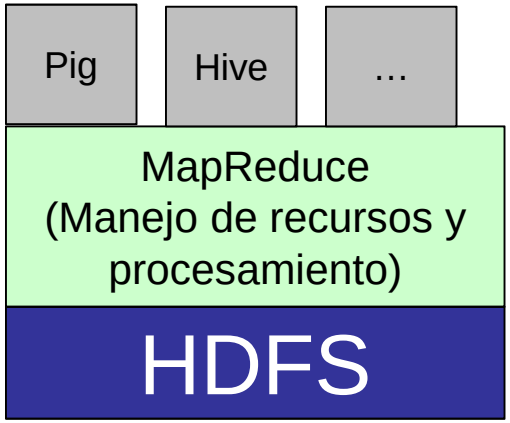
- Divide las funciones del JobTracker en dos actividades separadas:
  - ResourceManager (RM). Dos componentes
    - Un despachador global y un *agente* por nodo (NodeManager, NM) encargado de monitorerar el uso de recursos y reportarlos al despachador
    - Un ApplicationMaster (AM)
      - Un AM por 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

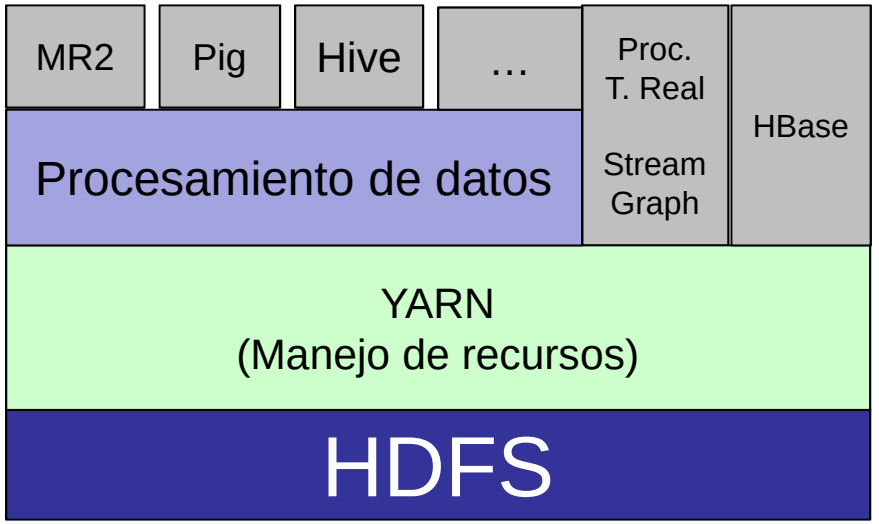
Hadoop 1

Plataforma especializada



Hadoop 2

Plataforma multipropósito



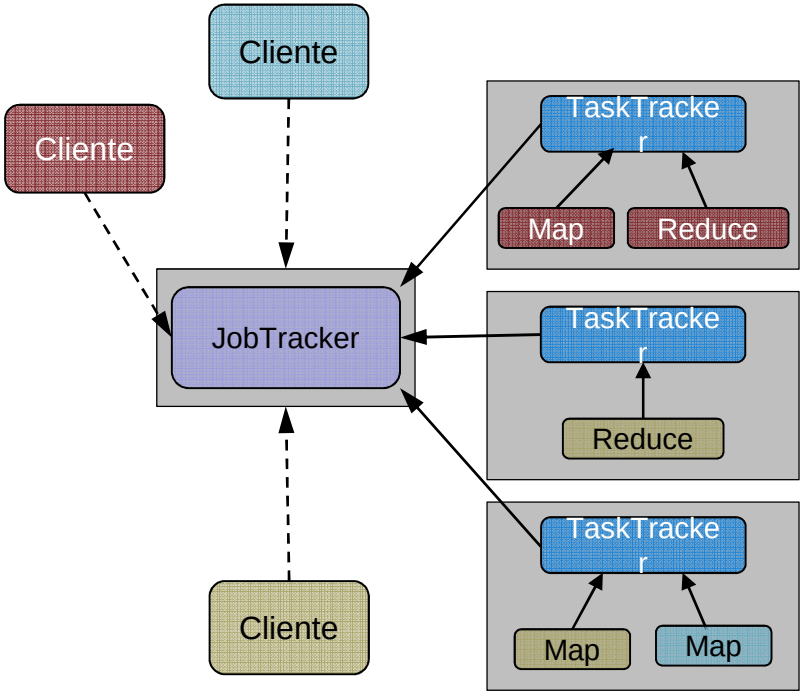
Con YARN, Hadoop V2 soporta distintos ambientes de ejecución. MapReduce es sólo uno de ellos

Principales características

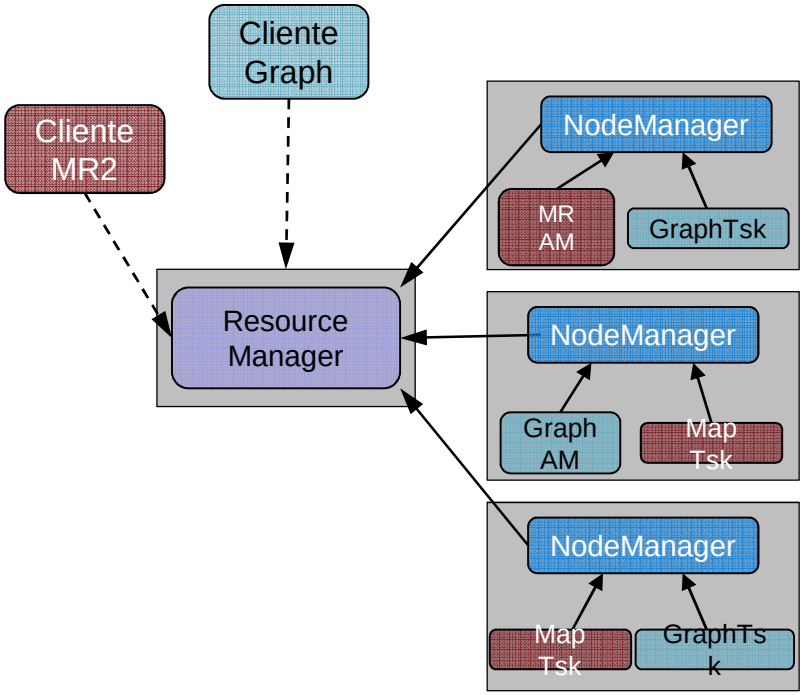
- Escalabilidad
  - Separar funcionalidades del JobTracker permite un incremento de 10x en el número de nodos y tareas
- Múltiples ambientes
  - Distintos ambientes de ejecución operando simultáneamente en la misma infraestructura con base en SLAs y políticas
- Compatibilidad
  - Misma API que Hadoop/MRv1
- 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

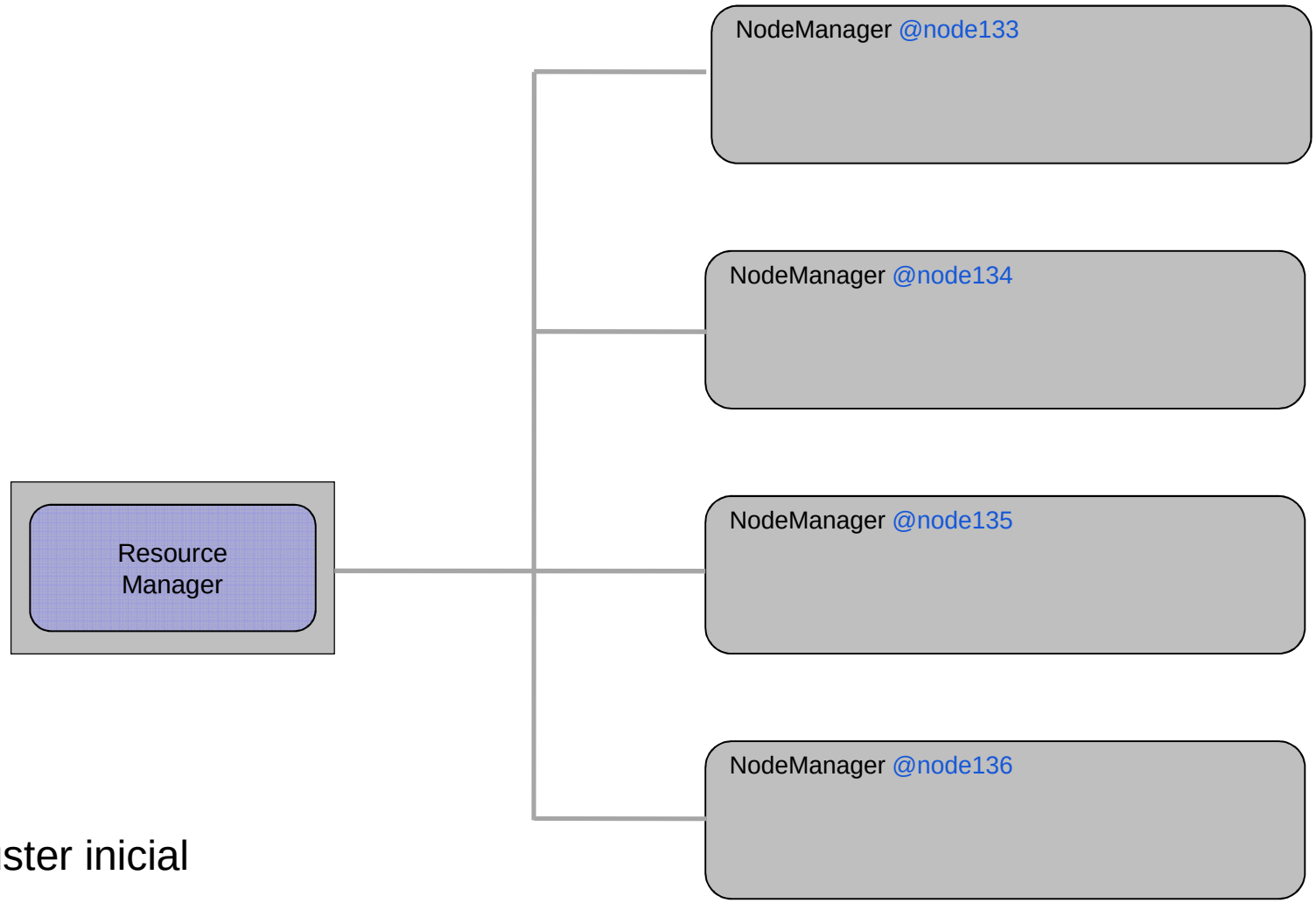


Diferencias terminología

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

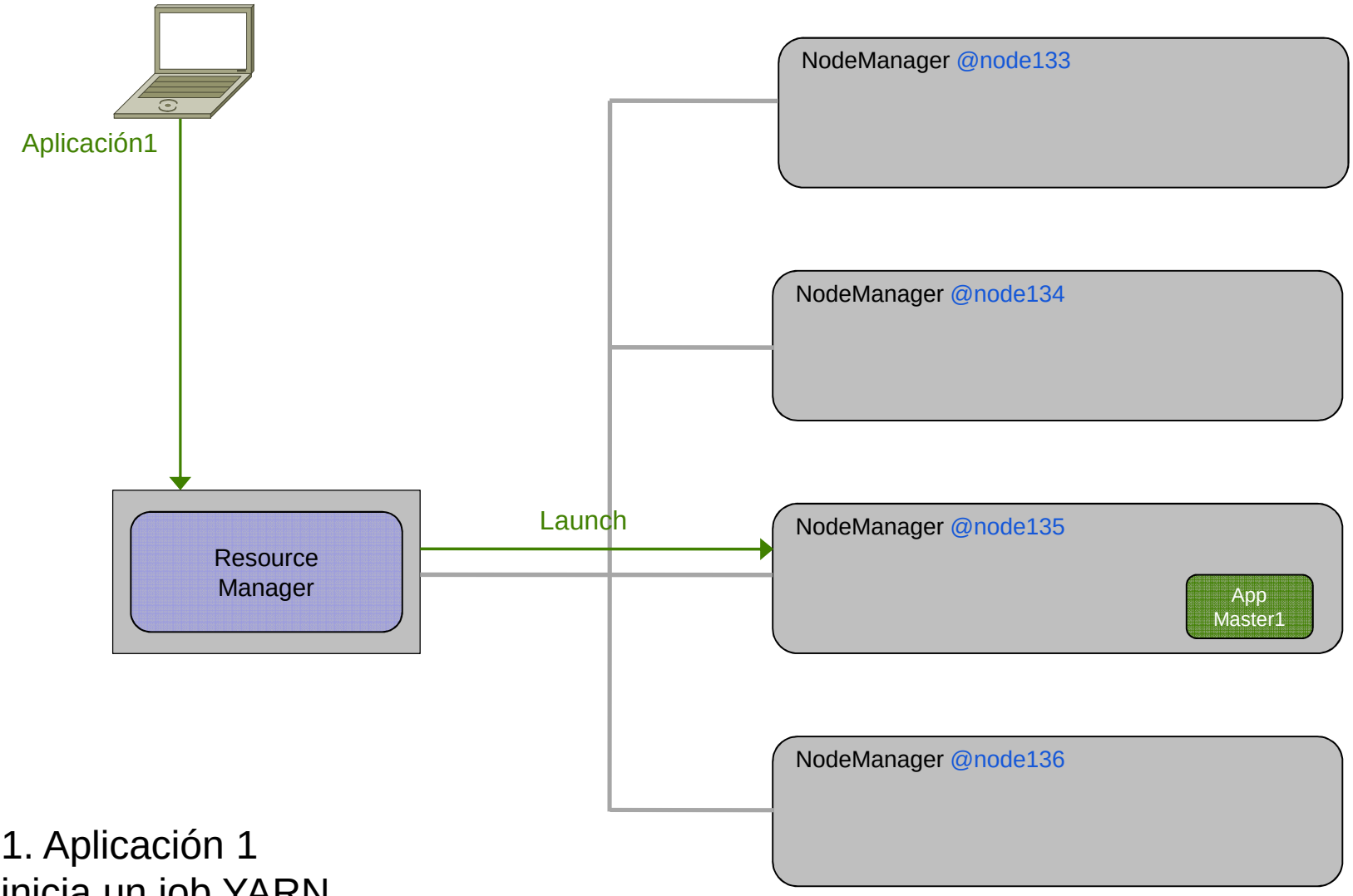
En YARN los recursos se entienden en términos de contenedores supervisados por el NodeManager

Ejecutando una aplicación



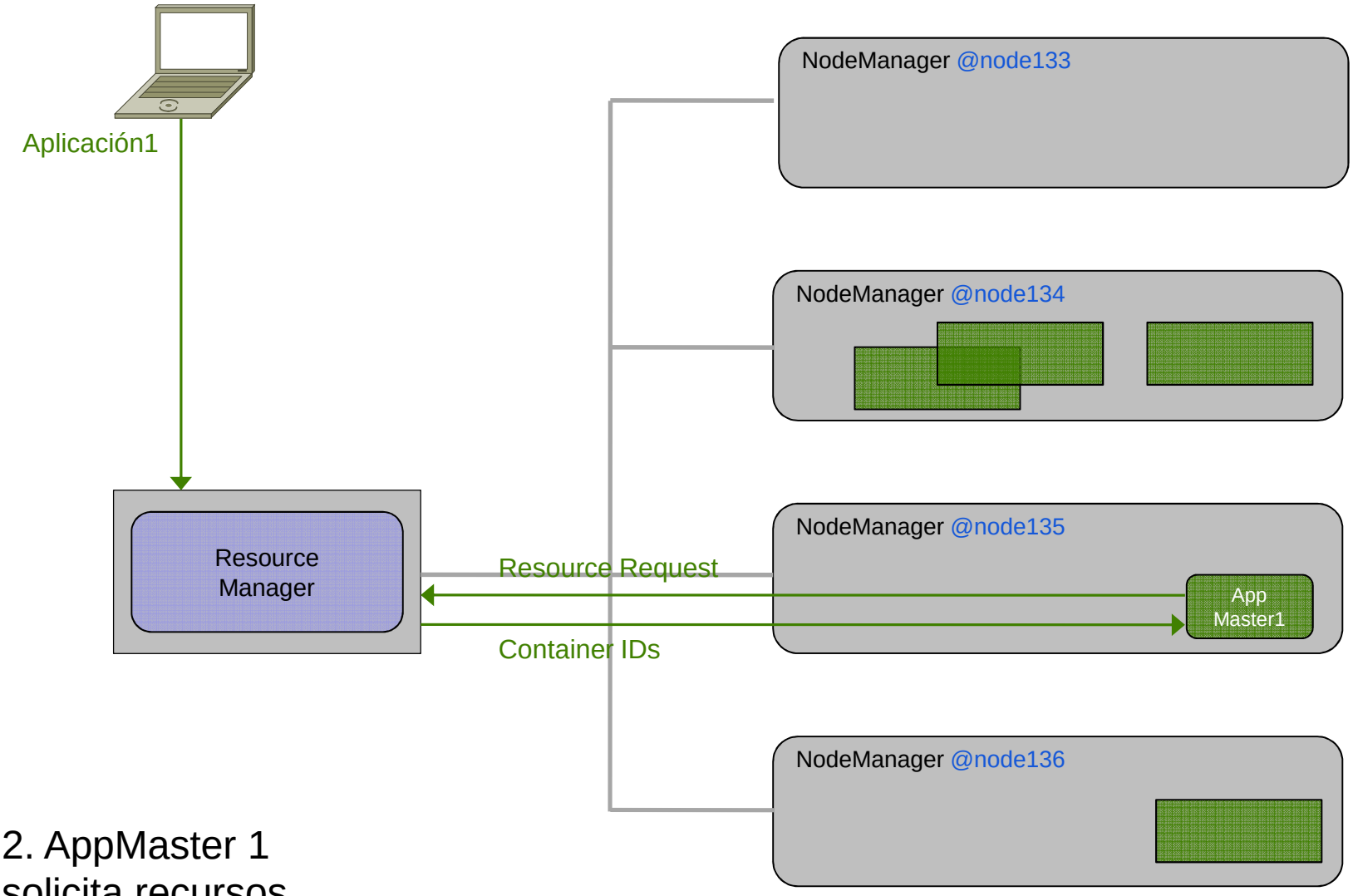
0. Cluster inicial

Ejecutando una aplicación



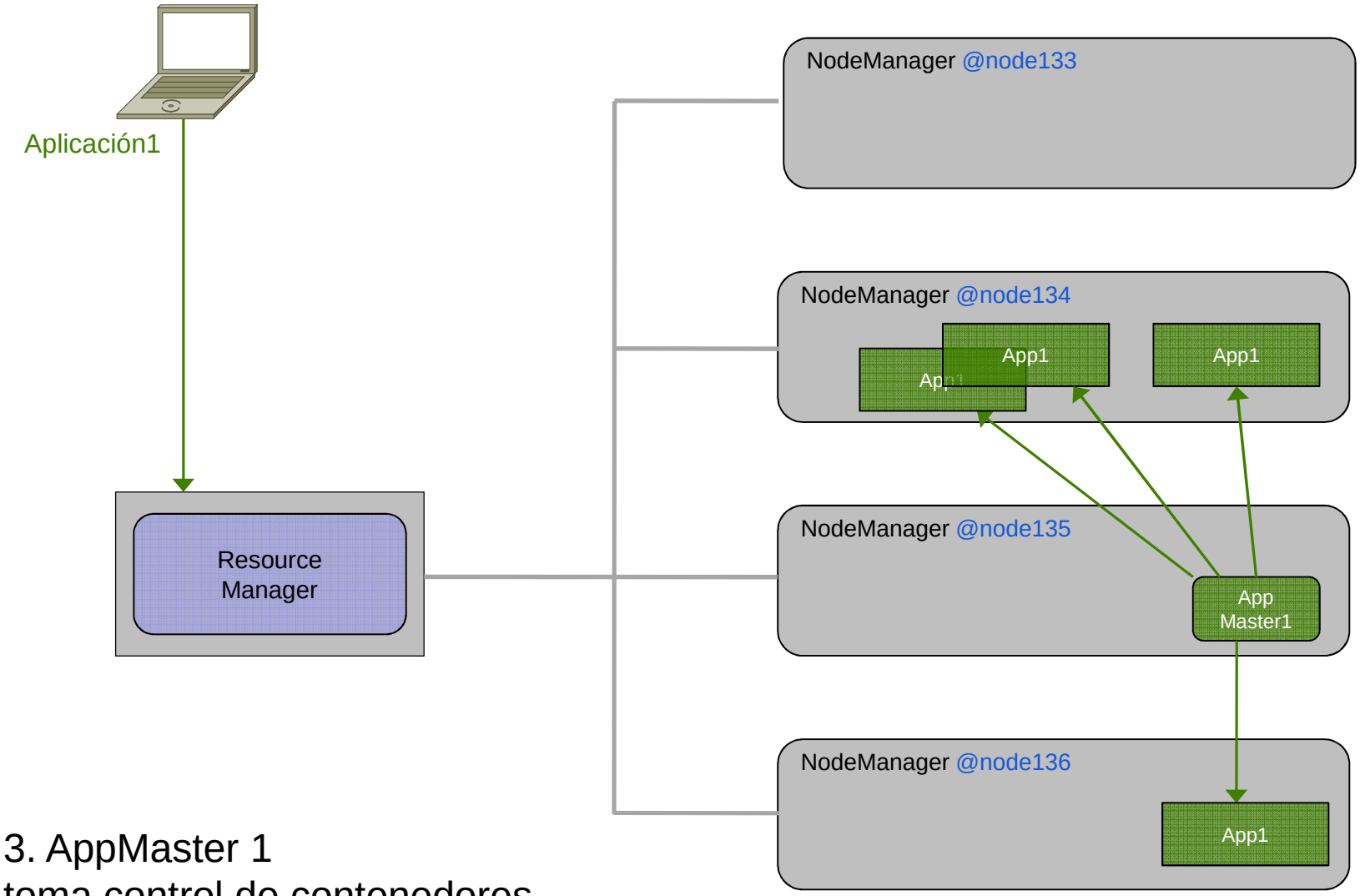
1. Aplicación 1
inicia un job YARN

Ejecutando una aplicación



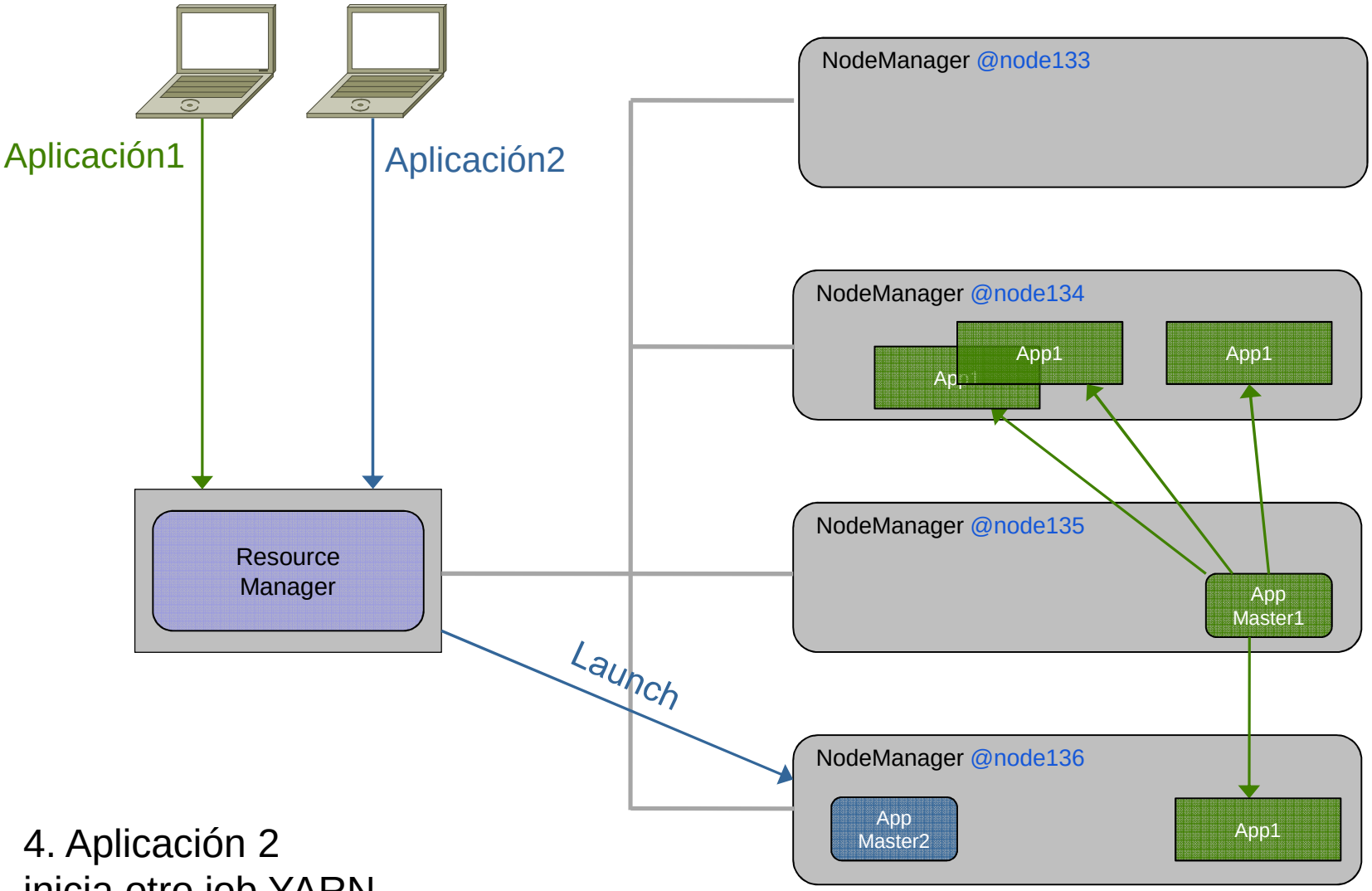
2. AppMaster 1 solicita recursos

Ejecutando una aplicación



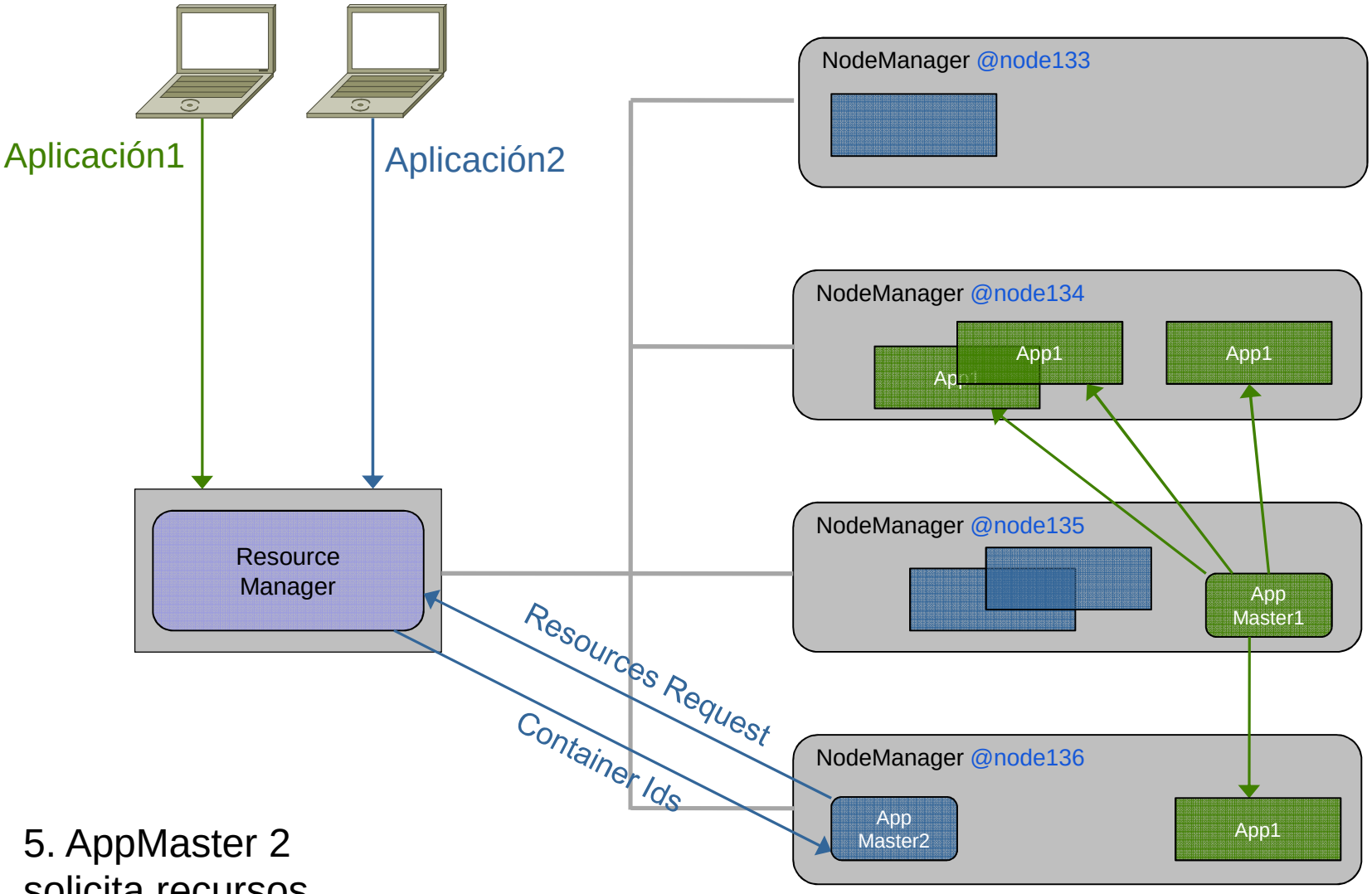
3. AppMaster 1
toma control de contenedores

Ejecutando una aplicación



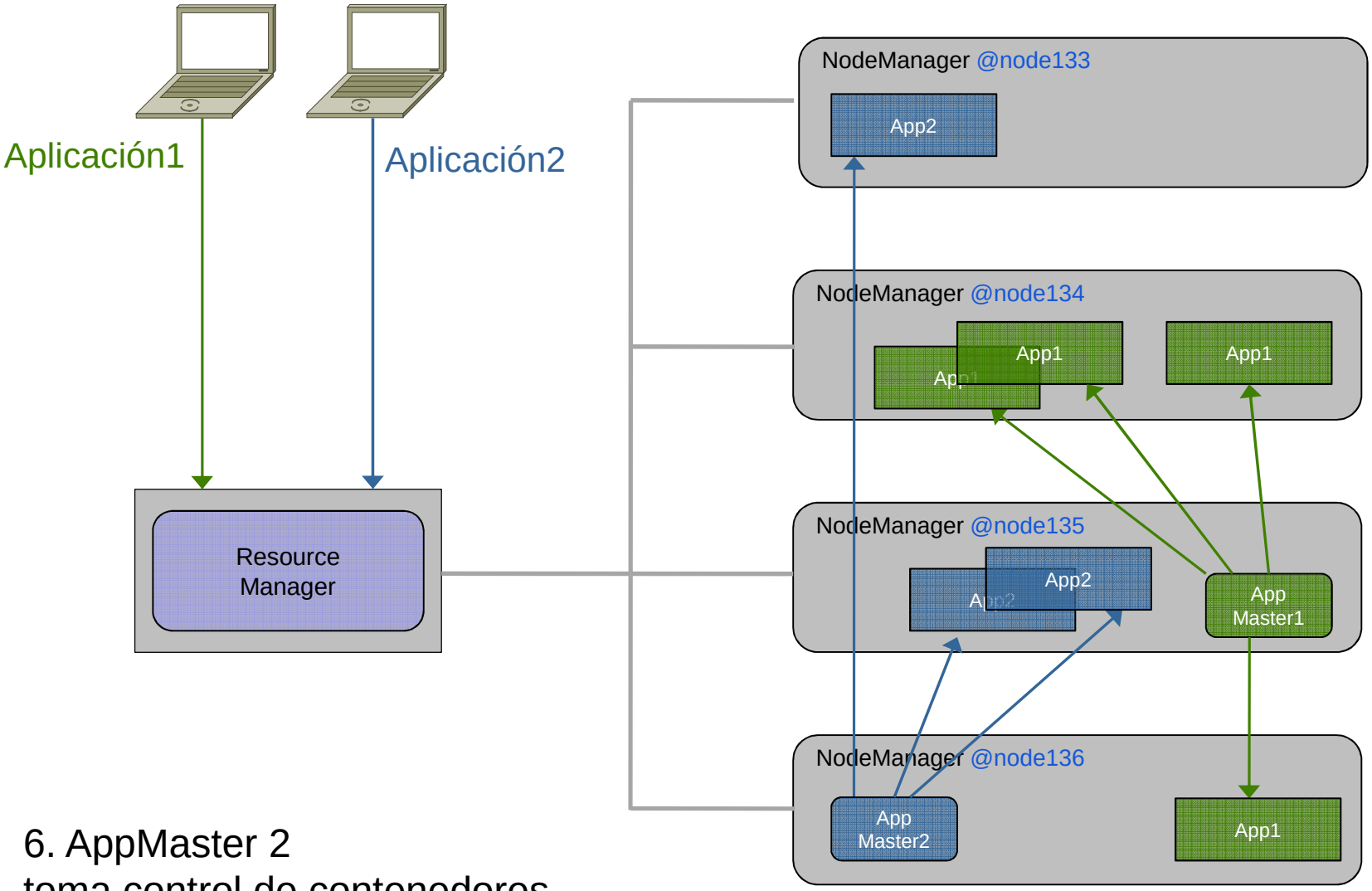
4. Aplicación 2
inicia otro job YARN

Ejecutando una aplicación



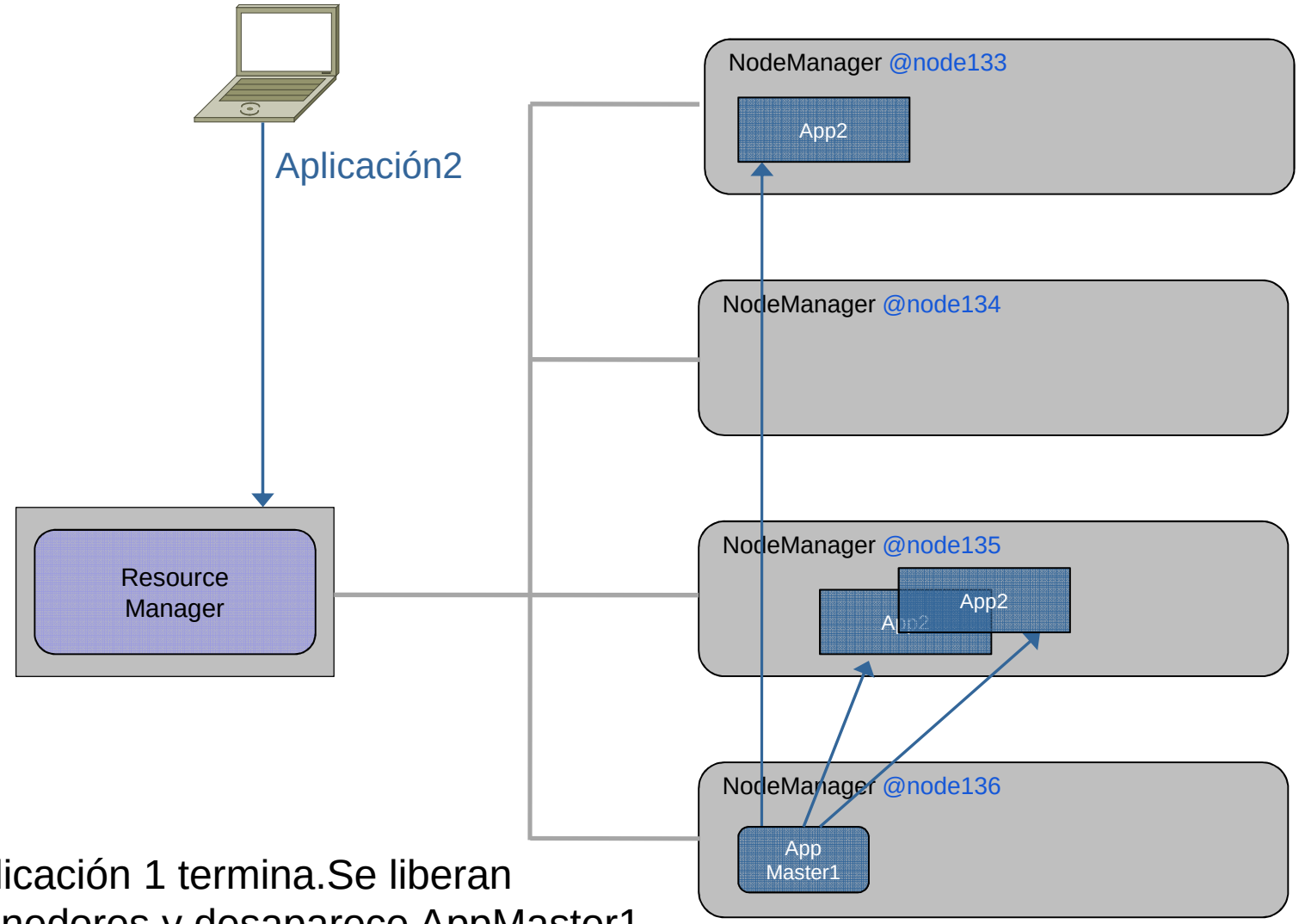
5. AppMaster 2 solicita recursos

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

Apache Hive

Introducción

- Fue creado en 2007 por Facebook (Pig fue creado por Yahoo!)
  - Base de datos de 700 TB muy lenta en procesarse en su data warehouse
  - Migraron a Hadoop pero el modelo MapReduce era muy poco amigable
- Principio de diseño
  - Retener los conceptos familiares para la explotación de bases de datos (y DWH) manteniendo la extensibilidad, flexibilidad y bajo costo de Hadoop

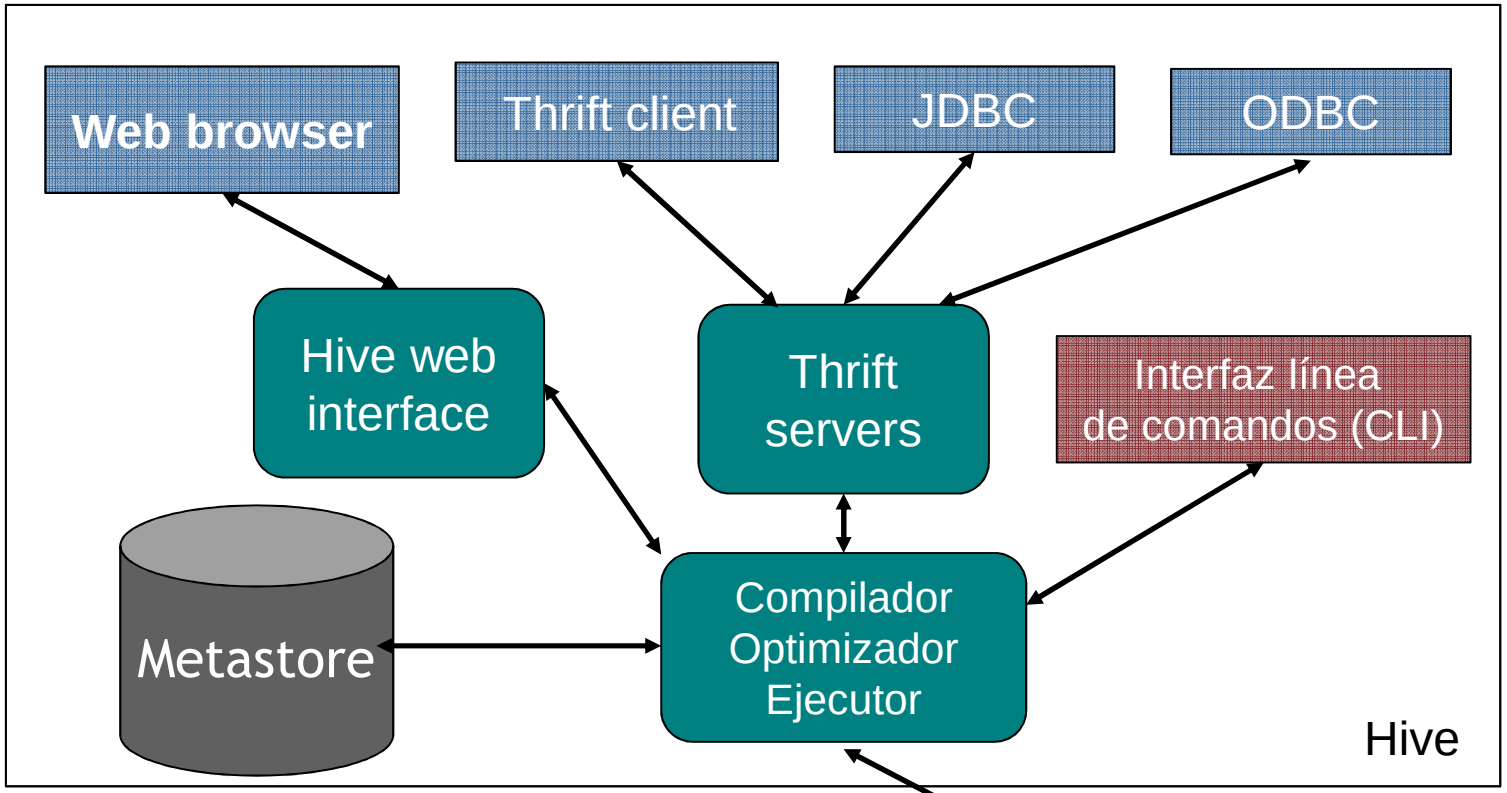
En sus orígenes, Pig era más útil para diseño de scripts interactivos, como lenguaje de flujo de datos y para manejo de datos semi-estructurados. Hive fue concebido para manejo eficiente de DWH con datos estructurados.

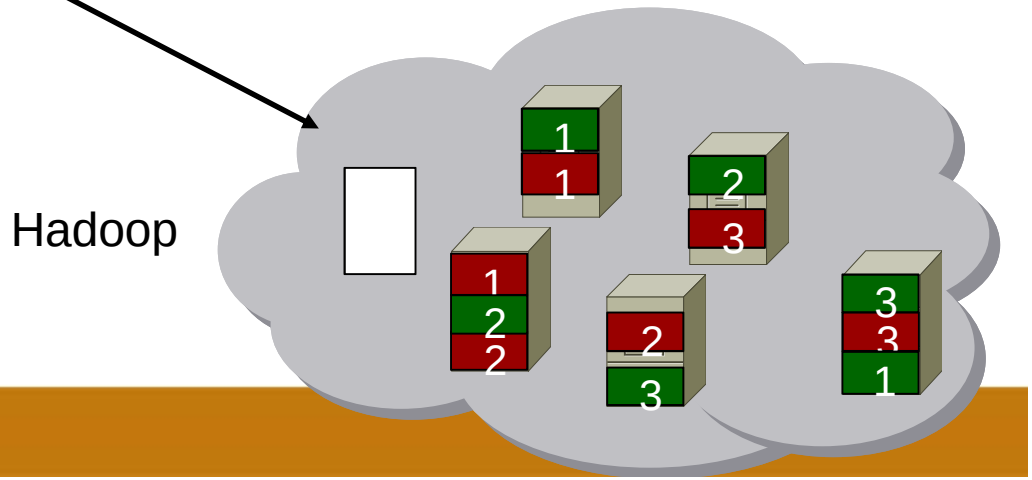
En su evolución se han ido traslapando, por lo que hay muchas discusiones sobre cuándo usar Pig o Hive y hasta por qué se tienen los dos.

Hive

- Hive es:
  - Un datawarehouse sobre Hadoop
  - Facilita sumarización, queries adhHoc, análisis de datasets almacenados en Hadoop
  - Interfaz SQL (HQL). Data definition language y data manipulation language
  - Permite proyectar estructura sobre datasets en Hadoop
  - Catalog metastore. Mapea estructura de archivos a forma tabular
- Hive NO ES:
  - Un Manejador de Base de Datos completo
  - Para procesamiento en tiempo real: Latencias mucho mayores que RDBMS. Schema on Read = carga rápida pero query costoso
  - No soporta modelos transaccionales

Componentes



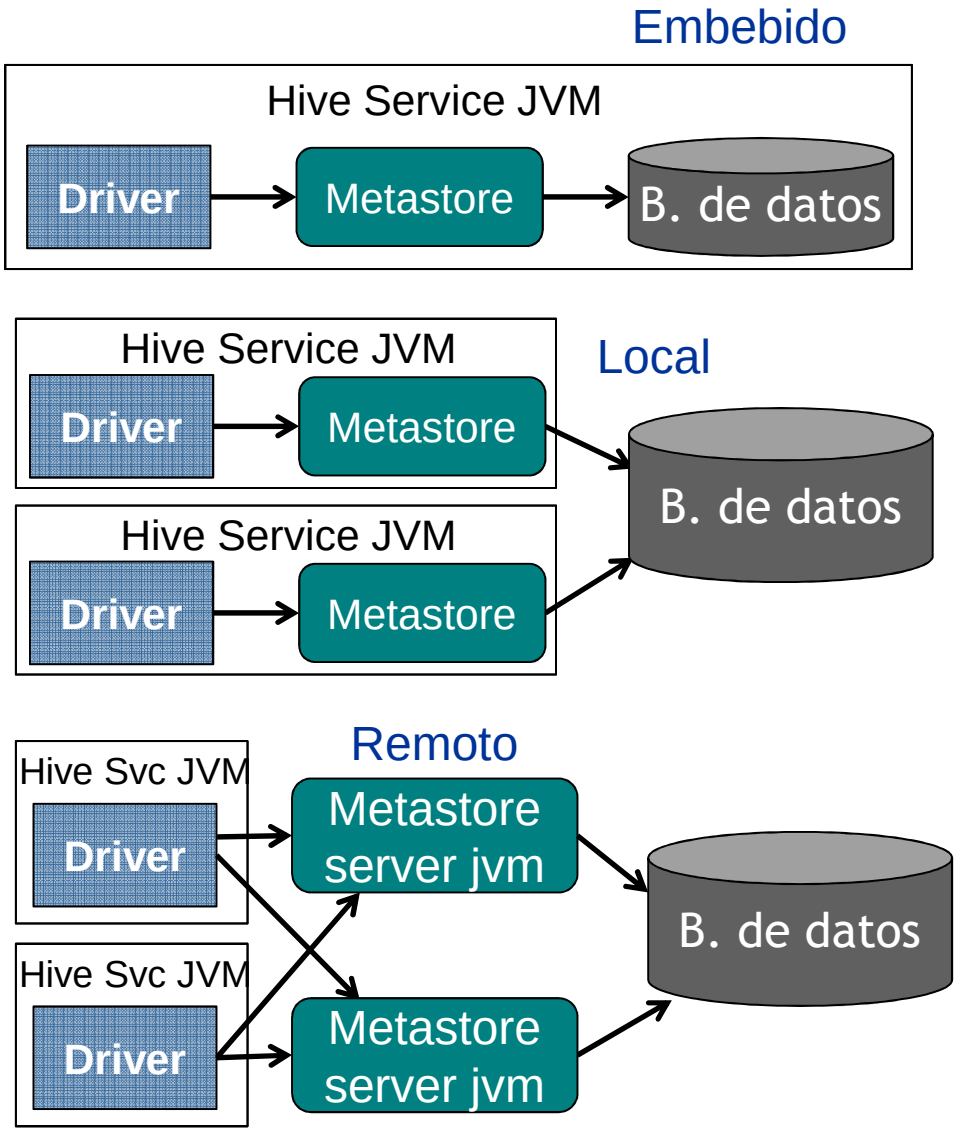


Metastore

Almacena el catálogo del sistema y metadata sobre tablas, columnas, particiones, etc.

Permite mapear estructuras de archivos a formas tabulares.

- Embebido. Ejecuta el código en el mismo proceso que Hive. La BD está en el mismo proceso. Típicamente para ambientes de pruebas
- Local. La BD está separada en otro proceso pero el código se ejecuta en el mismo proceso de Hive
- Remoto. Se ejecuta en un proceso independiente. Puede ser compartido por otros usuarios y procesos. Es el más común en ambientes de producción.



Componentes

HCatalog es un nuevo componente encima del metastore y ofrece interfaces de lectura y escritura para Pig y MapReduce. Usa CLI para emitir definición de datos y comandos para explorar metadata.
HCatalog facilita a usuarios de Pig, MapReduce y Hive leer y escribir datos en una malla.
El driver, compilador, Optimizador y Ejecutor de Hive trabajan juntos para convertir un query en un conjunto de jobs.

- El Driver maneja el ciclo de vida de un comando HQL. Mantiene un handle y estadísticas para la sesión.
- El compilador convierte queries en un grafo dirigido de tareas MapReduce que son monitoreadas por la *execution engine*.

Organización de datos. DDL

En Hive los datos se organizan jerárquicamente en las siguientes estructuras (definidas por orden de granularidad)

Database –Namespace que separa tablas y otras unidades de datos

Table –Unidades *homogéneas*, con el mismo *schema*

Partition –Opcional, es una columna virtual que define cómo se almacena en el HDFS. Muy útil para optimización

Bucket – Opcional, son divisiones con base en un valor hash de una columna permite optimizar *joins*

Almacenamiento

- Los archivos se almacenan directamente en HDFS. Se puede utilizar una amplia variedad de formatos para los registros. El formato interno en realidad cambiará de tabla en tabla dependiendo de cómo se configure.

  Por default se almacenan en la carpeta warehouse del archivo de configuración. Se crea un directorio para la BD y subdirectorio para cada tabla, subsubdir para partition, y el archivo se almacena ahí o en varios bucketfiles si se decidió usarlos.

Entidad	Ejemplo	Ubicación
B. de datos	Midb	/apps/midb.db
Tabla	T	/apps/midb.db/T
Partición	Date='230602017	/apps/midb.db/T/date=23062017
Bucket col.	userId	/apps/midb.db/T/date=2306201700000_0 … /apps/midb.db/T/date=2306201700017_0

Tipos de datos

- Integer
  - TINYINT – 1 byte
  - SMALLINT – 2 bytes
  - INT – 4 bytes
  - BIGINT – 8 bytes
- Boolean
  - TRUE/FALSE
- Float
  - FLOAT, DOUBLE, DECIMAL
- String
  - STRING
  - VARCHAR (especifica long)
- Date/Time
  - TIMESTAMP YYYY-MM-DD:MM:SS:ffffff
  - DATE –YYYY-MM-DD
- Binary
- Array
  - Cualquier tipo primitivo
  - Se indexan a partir de 0
- Struct
  - Colección de elementos de distinto tipo
  - Se acceden con notación punto
- Map
  - Colección de tuplas <key-value>
  - Se accede por Name[key]
  - Key debe ser tipo primitivo
- Union
  - Puede variar el tipo de dato que contiene pero en un momento determinado solo puede tener exactamente uno de los tipos definidos

Mapeo HQL a MapReduce: Join

page_view

pageid	**userid**	time
1	**111**	9:08:01
2	**111**	9:08:13
1	**222**	9:08:14

x

user

userid	age	gender
111	25	female
222	32	male

=

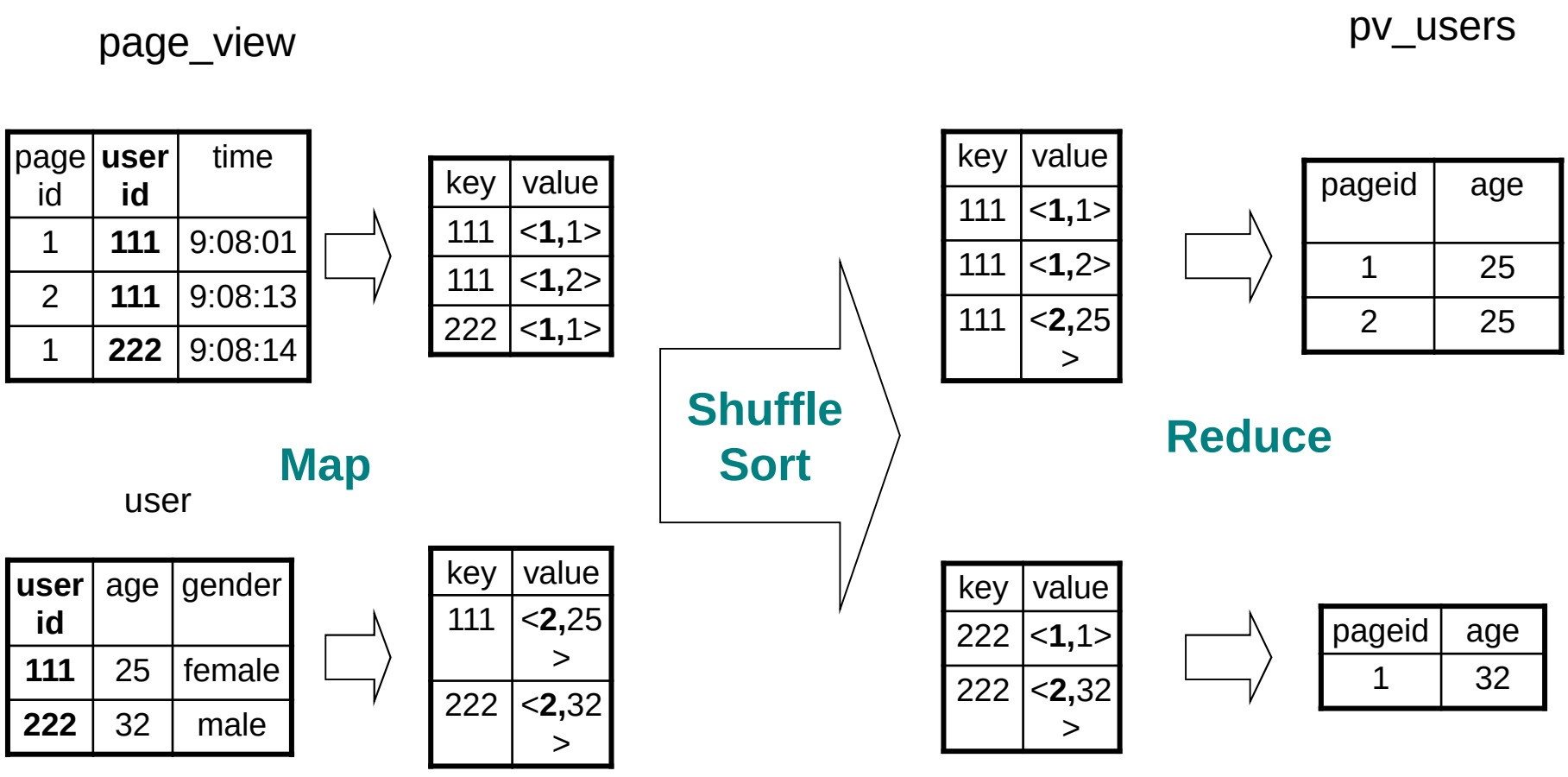
pv_users

pageid	age
1	25
2	25
1	32

HQL:

```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.userid);
```

Mapeo HQL a MapReduce: Join



Incremento en desempeño Hive

- How to optimize Hive Performance (www.bigdataanalust.in)
  - Cost based optimization – Optimiza plan de ejecución https://es.hortonworks.com/blog/hive-0-14-cost-based-optimizer-cbo-technical-overview/
  - Privilegia Tez sobre MapReduce como ambiente de ejecución
  - Skewed tables – separa valores que ocurren con mucha frecuencia en una tabla separada
  - Vectorización – Permite combinar varias filas en una sola
  - Comprime con ORC_table
  - Usa SORT by en vez de ORDER by
  - En JOIN, pon tabla grande a la derecha y usa Map_side join
  - Almacena bloques comprimidos de 256 MB (o el tamaño de los fragmentos de HDFS
- Practical Hive (http://www.apress.com/jp/book/9781484202722)
- 10 ways to optimize Hive queries http://sanjivblogs.blogspot.mx/2015/05/10-ways-to-optimizing-hive-queries.html