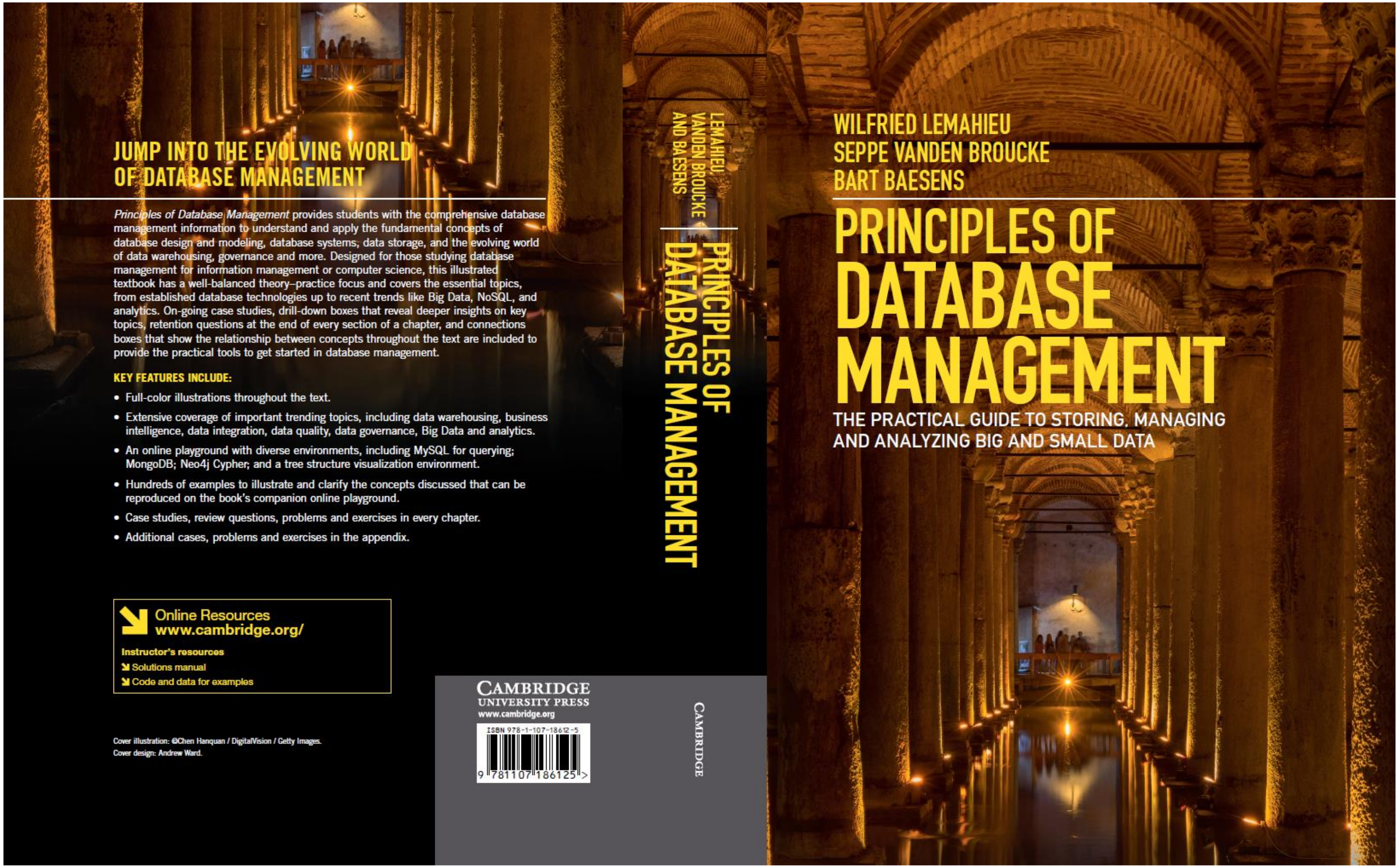
# Big Data



## JUMP INTO THE EVOLVING WORLD OF DATABASE MANAGEMENT

*Principles of Database Management* provides students with the comprehensive database management information to understand and apply the fundamental concepts of database design and modeling, database systems, data storage, and the evolving world of data warehousing, governance and more. Designed for those studying database management for information management or computer science, this illustrated textbook has a well-balanced theory–practice focus and covers the essential topics, from established database technologies up to recent trends like Big Data, NoSQL, and analytics. On-going case studies, drill-down boxes that reveal deeper insights on key topics, retention questions at the end of every section of a chapter, and connections boxes that show the relationship between concepts throughout the text are included to provide the practical tools to get started in database management.

**KEY FEATURES INCLUDE:**

- Full-color illustrations throughout the text.
- Extensive coverage of important trending topics, including data warehousing, business intelligence, data integration, data quality, data governance, Big Data and analytics.
- An online playground with diverse environments, including MySQL for querying; MongoDB; Neo4j Cypher; and a tree structure visualization environment.
- Hundreds of examples to illustrate and clarify the concepts discussed that can be reproduced on the book's companion online playground.
- Case studies, review questions, problems and exercises in every chapter.
- Additional cases, problems and exercises in the appendix.

Online Resources
**www.cambridge.org/**

**Instructor's resources**
Solutions manual
Code and data for examples

Cover illustration: ©Chen Hanquan / DigitalVision / Getty Images.
Cover design: Andrew Ward.

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org
ISBN 978-1-107-18612-5

LEMAHIEU,
VANDEN BROUCKE
AND BAESENS

PRINCIPLES OF
DATABASE MANAGEMENT

CAMBRIDGE

WILFRIED LEMAHIEU
SEPPE VANDEN BROUCKE
BART BAESENS

## PRINCIPLES OF DATABASE MANAGEMENT

THE PRACTICAL GUIDE TO STORING, MANAGING AND ANALYZING BIG AND SMALL DATA

[www.pdbmbook.com](http://www.pdbmbook.com)

# Introduction

- The 5 V’s of Big Data
- Hadoop
- SQL on Hadoop
- Apache Spark

# The 5 V’s of Big Data

- Every minute:
  - More than 300000 tweets are created
  - Netflix subscribers are streaming more than 70000 hours of video at once
  - Apple users download 30000 apps
  - Instagram users like almost 2 million photos
- Big Data encompasses both structured and highly unstructured forms of data

# The 5 V’s of Big Data

- **Volume**: the amount of data, also referred to the data “at rest”
- **Velocity**: the speed at which data comes in and goes out, data “in motion”
- **Variety**: the range of data types and sources that are used, data in its “many forms”
- **Veracity**: the uncertainty of the data, data “in doubt”
- **Value**: TCO and ROI of the data

# The 5 V’s of Big Data

- Examples:
    - Large scale enterprise systems (e.g., ERP, CRM, SCM)
    - Social networks (e.g., Twitter, Weibo, WeChat)
    - Internet of Things
    - Open data

# Hadoop

- Open-source software framework used for distributed storage and processing of big datasets
- Can be set up over a cluster of computers built from normal, commodity hardware
- Many vendors offer their implementation of a Hadoop stack (e.g. Amazon, Cloudera, Dell, Oracle, IBM, Microsoft)

# Hadoop

- History of Hadoop
- The Hadoop stack

# History of Hadoop

- Key building blocks:
  - Google File System: a file system that could be easily distributed across commodity hardware, whilst providing fault tolerance
  - Google MapReduce: a programming paradigm to write programs that can be automatically parallelized and executed across a cluster of different computers
- Nutch web crawler prototype developed by Doug Cutting
  - Later renamed to Hadoop
- In 2008, Yahoo! open-sourced Hadoop as “Apache Hadoop”

# The Hadoop Stack

- Four modules:
  - Hadoop Common: a set of shared programming libraries used by the other modules
  - Hadoop Distributed File System (HDFS): a Java-based file system to store data across multiple machines
  - MapReduce framework: a programming model to process large sets of data in parallel
  - YARN (Yet Another Resource Negotiator): handles the management and scheduling of resource requests in a distributed environment

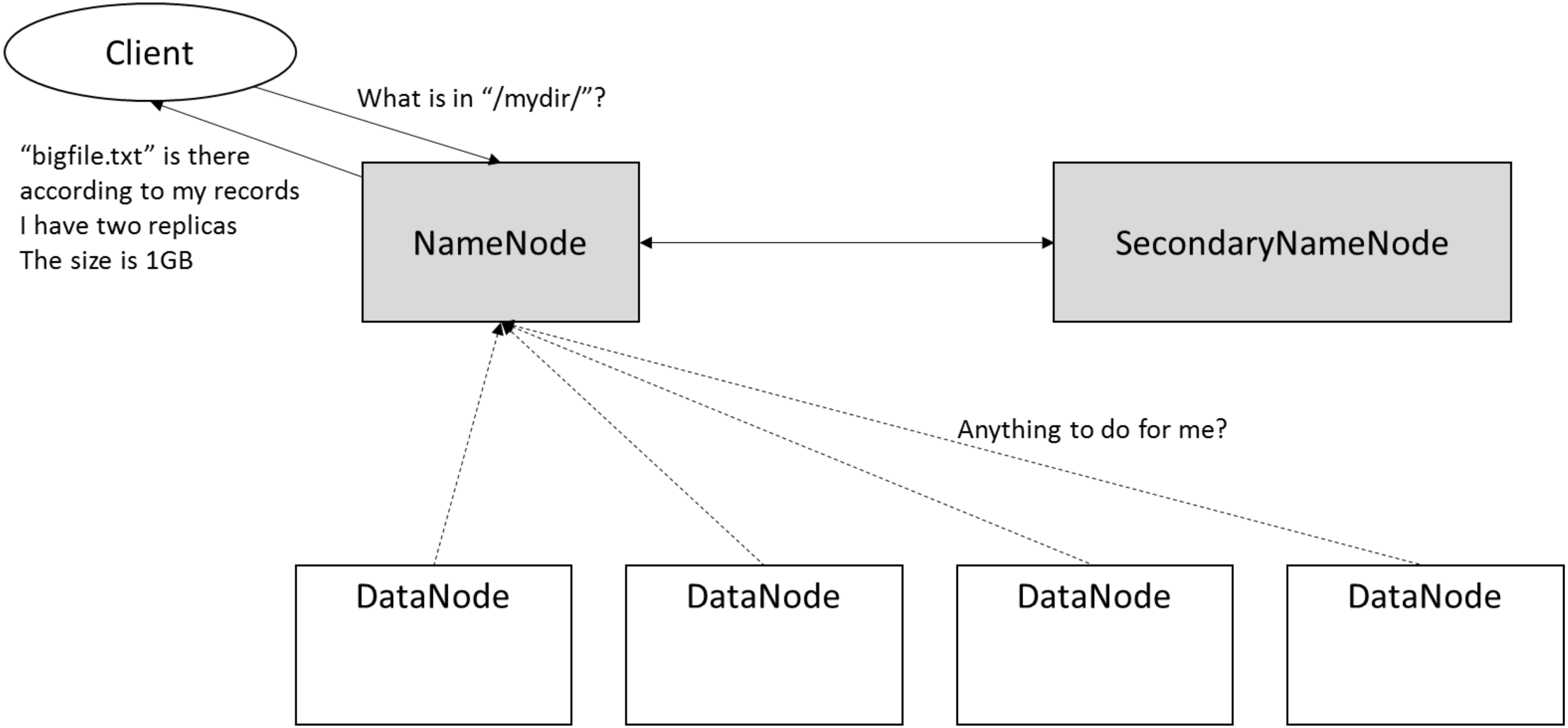
# Hadoop Distributed File System (HDFS)

- Distributed file system to store data across a cluster of commodity machines
- High emphasis on fault-tolerance
- HDFS cluster is composed of a NameNode and various DataNodes

# Hadoop Distributed File System (HDFS)

- NameNode
  - a server which holds all the metadata regarding the stored files
  - manages incoming file system operations
  - maps data blocks (parts of files) to DataNodes
- DataNode
  - handles file read and write requests
  - create, delete and replicate data blocks amongst their disk drives
  - continuously loop, asking the NameNode for instructions.
- Note: size of 1 data block is typically 64 megabytes

# Hadoop Distributed File System (HDFS)



Client

What is in “/mydir/”?

“bigfile.txt” is there
according to my records
I have two replicas
The size is 1GB

NameNode

SecondaryNameNode

Anything to do for me?

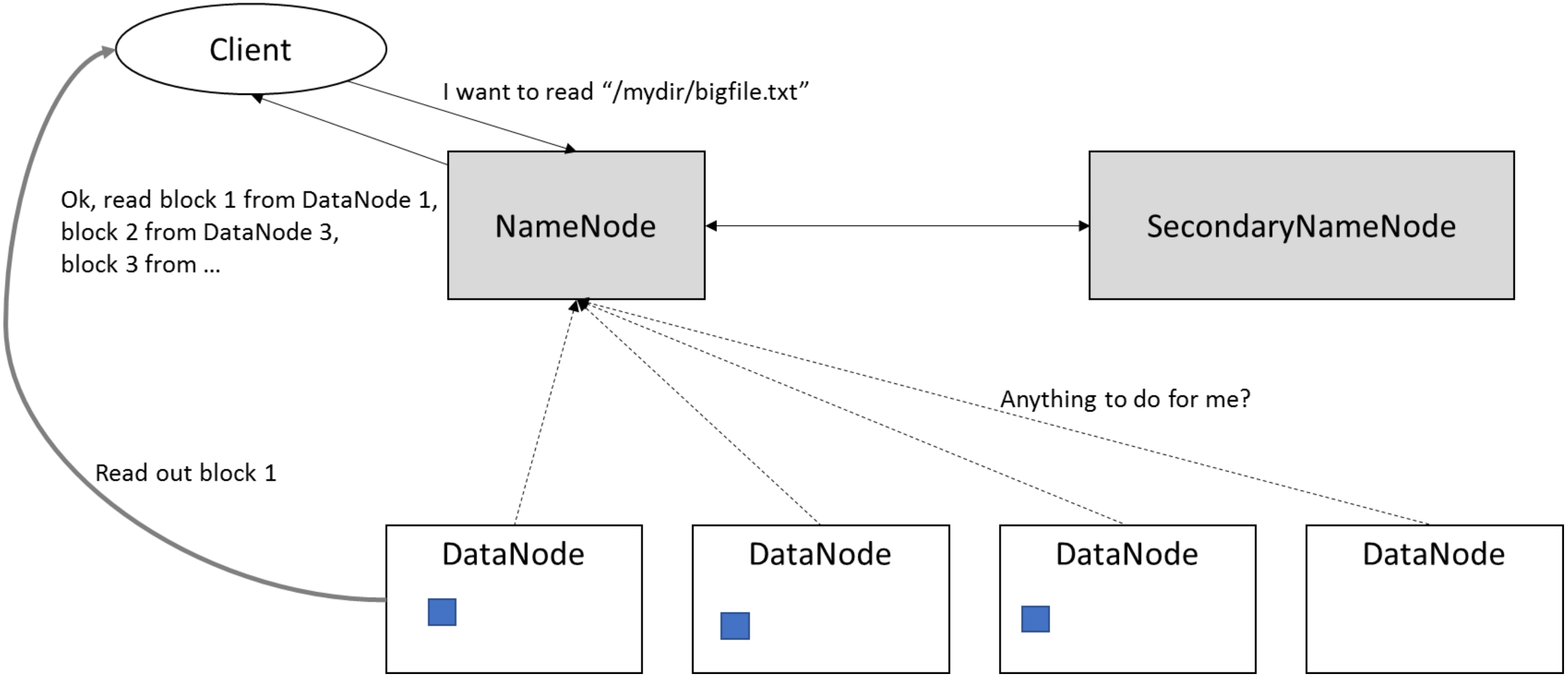
DataNode

DataNode

DataNode

DataNode

# Hadoop Distributed File System (HDFS)



Client

I want to read "/mydir/bigfile.txt"

Ok, read block 1 from DataNode 1,
block 2 from DataNode 3,
block 3 from …

NameNode

SecondaryNameNode

Anything to do for me?

Read out block 1

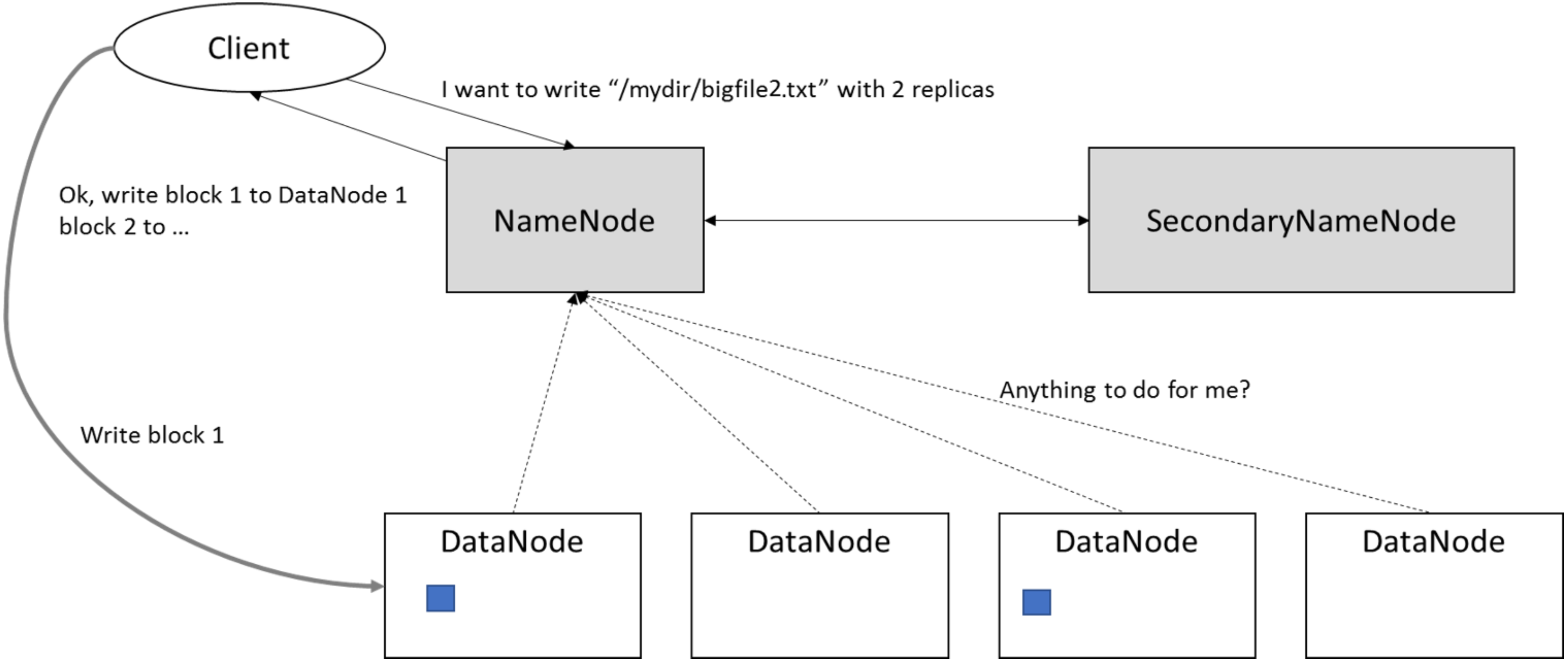
DataNode

DataNode

DataNode

DataNode

# Hadoop Distributed File System (HDFS)



Client

I want to write "/mydir/bigfile2.txt" with 2 replicas

Ok, write block 1 to DataNode 1
block 2 to ...

NameNode

SecondaryNameNode

Anything to do for me?

Write block 1

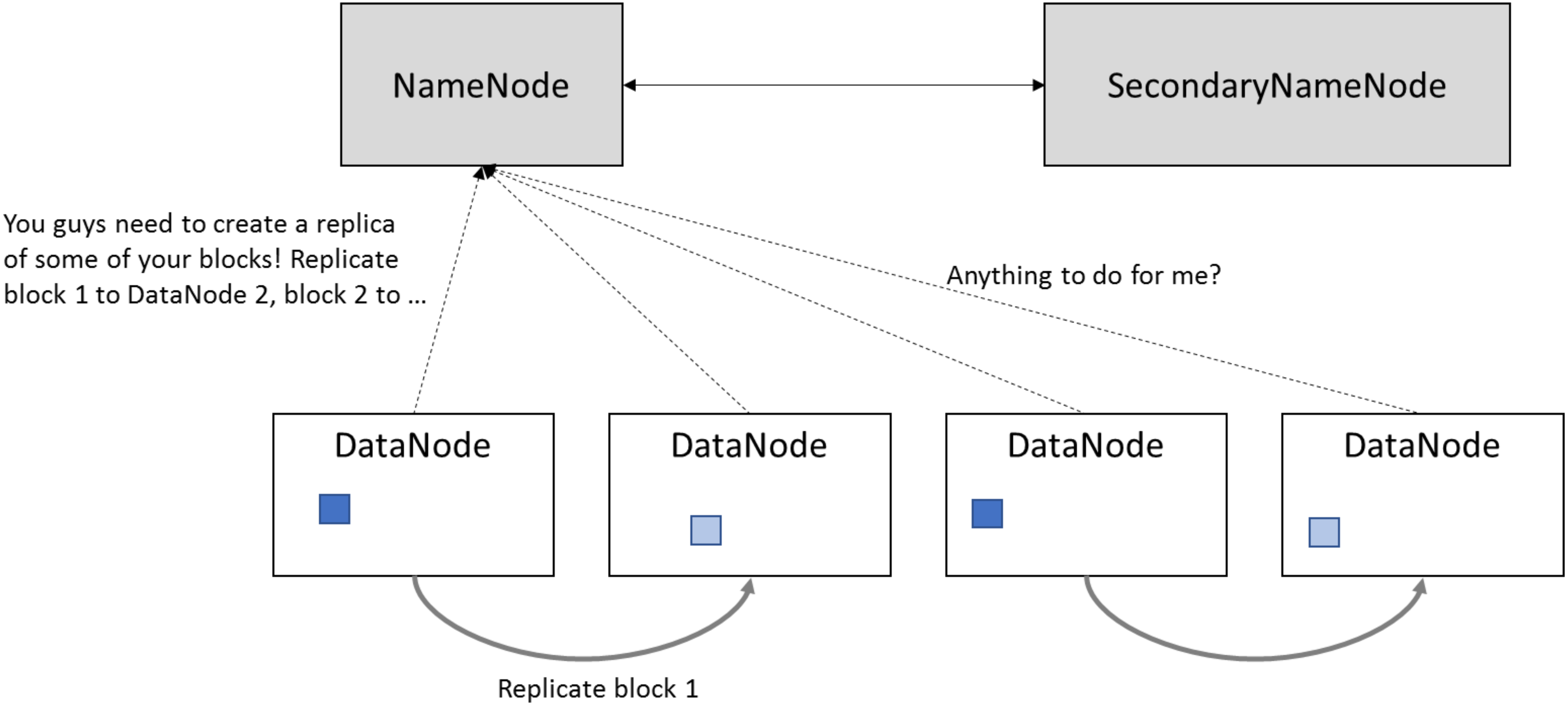
DataNode

DataNode

DataNode

DataNode

# Hadoop Distributed File System (HDFS)



You guys need to create a replica of some of your blocks! Replicate block 1 to DataNode 2, block 2 to …

Anything to do for me?

Replicate block 1

# Hadoop Distributed File System (HDFS)

- HDFS provides a native Java API to allow for writing Java programs that can interface with HDFS

```
String filePath = “/data/all_my_customers.csv”;
Configuration config = new Configuration();
org.apache.hadoop.fs.FileSystem hdfs =
org.apache.hadoop.fs.FileSystem.get(config);
org.apache.hadoop.fs.Path path = new
org.apache.hadoop.fs.Path(filePath);
org.apache.hadoop.fs.FSDataInputStream inputStream =
hdfs.open(path);
byte[] received = new byte[inputStream.available()];
inputStream.readFully(received);
```

# Hadoop Distributed File System (HDFS)

```
// ...
org.apache.hadoop.fs.FSDataInputStream inputStream = hdfs.open(path);
byte[] buffer=new byte[1024]; // Only handle 1KB at once
int bytesRead;
while ((bytesRead = in.read(buffer)) > 0) {
    // Do something with the buffered block here
}
```

# Hadoop Distributed File System (HDFS)

| | |
|---|---|
| `hadoop fs -mkdir mydir` | Create a directory on HDFS |
| `hadoop fs -ls` | List files and directories on HDFS |
| `hadoop fs -cat myfile` | View a file's content |
| `hadoop fs -du` | Check disk space usage on HDFS |
| `hadoop fs -expunge` | Empty trash on HDFS |
| `hadoop fs -chgrp mygroup myfile` | Change group membership of a file on HDFS |
| `hadoop fs -chown myuser myfile` | Change file ownership of a file on HDFS |
| `hadoop fs -rm myfile` | Delete a file on HDFS |
| `hadoop fs -touchz myfile` | Create an empty file on HDFS |
| `hadoop fs -stat myfile` | Check the status of a file (file size, owner, …) |
| `hadoop fs -test -e myfile` | Check if file exists on HDFS |
| `hadoop fs -test -z myfile` | Check if file is empty on HDFS |
| `hadoop fs -test -d myfile` | Check if myfile is a directory on HDFS |

# MapReduce

- Programming paradigm made popular by Google and subsequently implemented by Apache Hadoop
- Focus on scalability and fault tolerance
- A map-reduce pipeline starts from a series of values and maps each value to an output using a given mapper function

# MapReduce

- High-level Python example
  - **Map**

```
>>> numbers = [1,2,3,4,5]
>>> numbers.map(lambda x : x * x) # Map a
function to our list
[1,4,9,16,25]
```

  - **Reduce**

```
>>> numbers.reduce(lambda x : sum(x) + 1)
# Reduce a list using given function
16
```

# MapReduce

- A MapReduce pipeline in Hadoop starts from a list of key-value pairs, and maps each pair to one or more output elements
- The output elements are also key-value pairs
- Next, the output entries are grouped so all output entries belonging to the same key are assigned to the same worker  (e.g. physical machine)
- These workers then apply the reduce function to each group, producing a new list of key-value pairs
- The resulting, final outputs can then be sorted

# MapReduce

- Reduce-workers can already get started on their work even although not all mapping operations have finished yet
- Implications:
    - the reduce function should output the same key-value structure as the one emitted by the map function
    - the reduce function itself should be built in such a way so it provides correct results, even if called multiple times

# MapReduce

- In Hadoop, MapReduce tasks are written in Java
- To run a MapReduce task, a Java program is packaged as a JAR archive and launched as:

```
hadoop jar myprogram.jar TheClassToRun [args...]
```

# MapReduce

- Example: Java program to count the appearance of a word in a file

```
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class WordCount {
  // Following fragments will be added here
}
```

# MapReduce

- Define mapper function as a class extending the built-in `Mapper<KeyIn, ValueIn, KeyOut, ValueOut>` class, indicating which type of key-value input pair we expect and which type of key-value output pair our mapper will emit

# MapReduce

```
public static class MyMapper extends Mapper<Object, Text, Text, IntWritable> {
		// Our input key is not important here, so it can just be any generic object.  Our input value is a piece of text (a line)
		// Our output key will also be a piece of text (a word) . Our output value will be an integer.

		public void map(Object key, Text value, Context context)  throws IOException, InterruptedException {
				// Take the value, get its contents, convert to lowercase,
				// and remove every character except for spaces and a-z values:
				String document = value.toString().toLowerCase().replaceAll("[^a-z\\s]", "");
				// Split the line up in an array of words
				String[] words = document.split(" ");

				// For each word...
				for (String word : words) {
						// "context" is used to emit output values
						// Note that we cannot emit standard Java types such as int, String, etc. Instead, we need to use
						// a org.apache.hadoop.io.* class such as Text (for string values) and IntWritable (for integers)

						Text textWord = new Text(word);
						IntWritable one = new IntWritable(1);

						// ... simply emit a (word, 1) key-value pair:
						context.write(textWord, one);
				}
		}
}
```

# MapReduce

| **Input key-value pairs** | |
|---|---|
| Key <Object> | Value <Text> |
| 0 | This is the first line |
| 23 | And this is the second line, and this is all |

| **Mapped key-value pairs** | |
|---|---|
| Key <Text> | Value <IntWritable> |
| this | 1 |
| is | 1 |
| the | 1 |
| first | 1 |
| line | 1 |
| and | 1 |
| … | … |

# MapReduce

- reducer function is specified as a class extending the built-in `Reducer<KeyIn, ValueIn, KeyOut, ValueOut>` class

# MapReduce

```
public static class MyReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

        public void reduce(Text key, Iterable<IntWritable> values, Context context)
                        throws IOException, InterruptedException {
                int sum = 0;
                IntWritable result = new IntWritable();
                // Summarise the values so far...
                for (IntWritable val : values) {
                        sum += val.get();
                }
                result.set(sum);
                // ... and output a new (word, sum) pair
                context.write(key, result);
        }
}
```

# MapReduce

**Mapped key-value pairs**

| Key <Text> | Value <IntWritable> |
|---|---|
| this | 1 |
| is | 1 |
| the | 1 |
| first | 1 |
| line | 1 |
| and | 1 |
| this | 1 |
| is | 1 |

**Mapped key-value pairs for “this”**

| Key <Text> | Value <IntWritable> |
|---|---|
| this | 1 |
| this | 1 |

**Reduced key-value pairs for “this”**

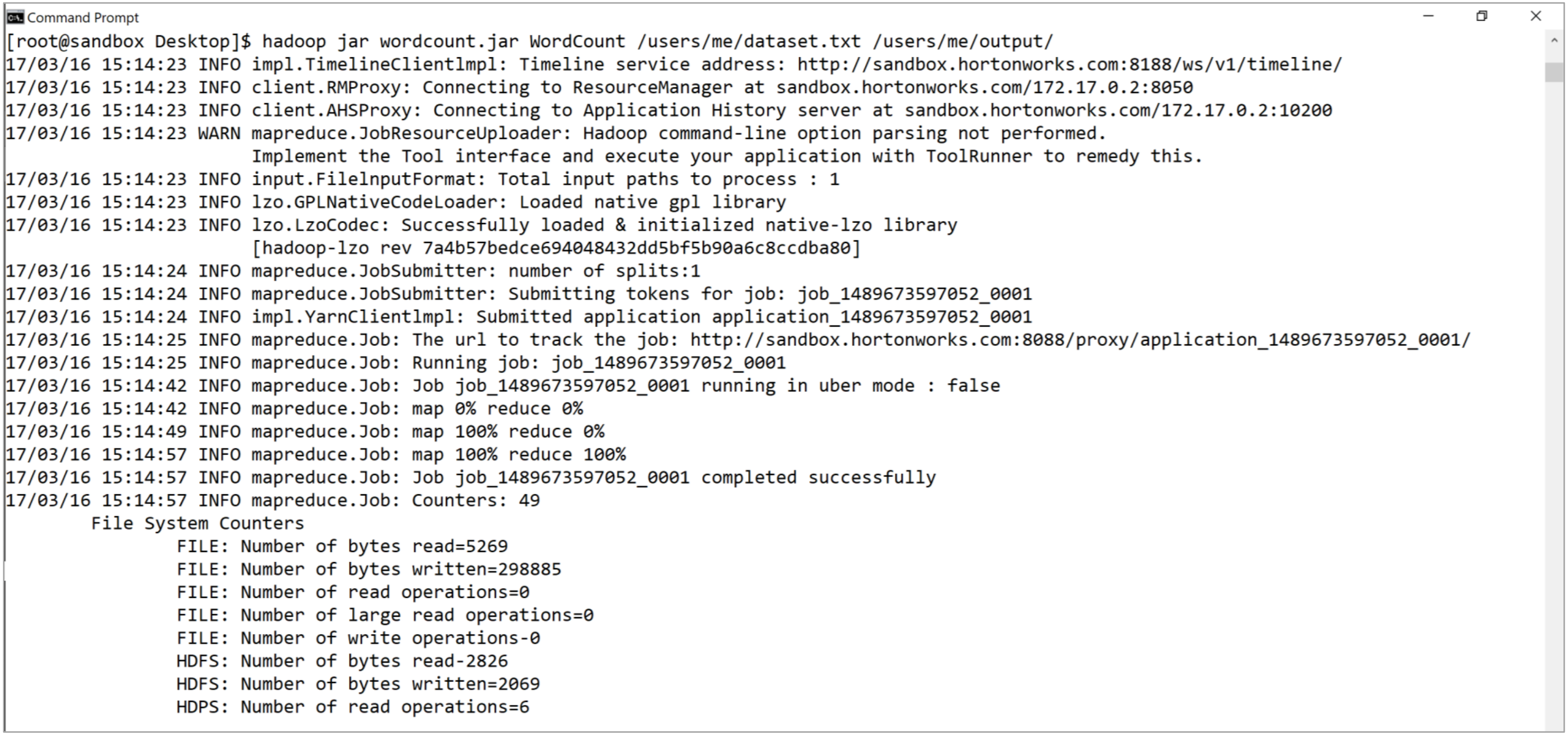
| Key <Text> | Value <IntWritable> |
|---|---|
| this | 1 + 1 = 2 |

# MapReduce

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

    // Set up a MapReduce job with a sensible short name:
    Job job = Job.getInstance(conf, "wordcount");

    // Tell Hadoop which JAR it needs to distribute
    // to the workers.
    // We can easily set this using setJarByClass
    job.setJarByClass(WordCount.class);

    job.setMapperClass(MyMapper.class);
    job.setReducerClass(MyReducer.class);

    // What does the output look like?
    job.setOutputKeyClass(Text.class);
    job.setOutputValueClass(IntWritable.class);

    // Our program expects two arguments, the first one is the input
    // file on HDFS
    // Tell Hadoop our input is in the form of TextInputFormat
    // (Every line in the file will become value to be mapped)
    TextInputFormat.addInputPath(job, new Path(args[0]));

    // The second argument is the output directory on
    // HDFS
    Path outputDir = new Path(args[1]);
    // Tell Hadoop what our desired output structure is
    FileOutputFormat.setOutputPath(job, outputDir);

    // Delete the output directory if it exists
    FileSystem fs = FileSystem.get(conf);
    fs.delete(outputDir, true);

    // Stop after our job has completed
    System.exit(job.waitForCompletion(true) ? 0 : 1);
}
```

# MapReduce

```
hadoop jar wordcount.jar WordCount /users/me/dataset.txt /users/me/output/
```



```
[root@sandbox Desktop]$ hadoop jar wordcount.jar WordCount /users/me/dataset.txt /users/me/output/
17/03/16 15:14:23 INFO impl.TimelineClientlmpl: Timeline service address: http://sandbox.hortonworks.com:8188/ws/v1/timeline/
17/03/16 15:14:23 INFO client.RMProxy: Connecting to ResourceManager at sandbox.hortonworks.com/172.17.0.2:8050
17/03/16 15:14:23 INFO client.AHSProxy: Connecting to Application History server at sandbox.hortonworks.com/172.17.0.2:10200
17/03/16 15:14:23 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed.
                       Implement the Tool interface and execute your application with ToolRunner to remedy this.
17/03/16 15:14:23 INFO input.FilelnputFormat: Total input paths to process : 1
17/03/16 15:14:23 INFO lzo.GPLNativeCodeLoader: Loaded native gpl library
17/03/16 15:14:23 INFO lzo.LzoCodec: Successfully loaded & initialized native-lzo library
                       [hadoop-lzo rev 7a4b57bedce694048432dd5bf5b90a6c8ccdba80]
17/03/16 15:14:24 INFO mapreduce.JobSubmitter: number of splits:1
17/03/16 15:14:24 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1489673597052_0001
17/03/16 15:14:24 INFO impl.YarnClientlmpl: Submitted application application_1489673597052_0001
17/03/16 15:14:25 INFO mapreduce.Job: The url to track the job: http://sandbox.hortonworks.com:8088/proxy/application_1489673597052_0001/
17/03/16 15:14:25 INFO mapreduce.Job: Running job: job_1489673597052_0001
17/03/16 15:14:42 INFO mapreduce.Job: Job job_1489673597052_0001 running in uber mode : false
17/03/16 15:14:42 INFO mapreduce.Job: map 0% reduce 0%
17/03/16 15:14:49 INFO mapreduce.Job: map 100% reduce 0%
17/03/16 15:14:57 INFO mapreduce.Job: map 100% reduce 100%
17/03/16 15:14:57 INFO mapreduce.Job: Job job_1489673597052_0001 completed successfully
17/03/16 15:14:57 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=5269
                FILE: Number of bytes written=298885
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations-0
                HDFS: Number of bytes read-2826
                HDFS: Number of bytes written=2069
                HDPS: Number of read operations=6
```

# MapReduce

```
$ hadoop fs -ls /users/me/output
Found 2 items
-rw-r—r-- 1 root   hdfs   0    2017-05-20  15:11         /users/me/output/_SUCCESS
-rw-r—r-- 1 root   hdfs   2069  2017-05-20  15:11       /users/me/output/part-r-00000


$ hadoop fs -cat /users/me/output/part-r-00000
and       2
first     1
is        3
line      2
second    1
the       2
this      3
```

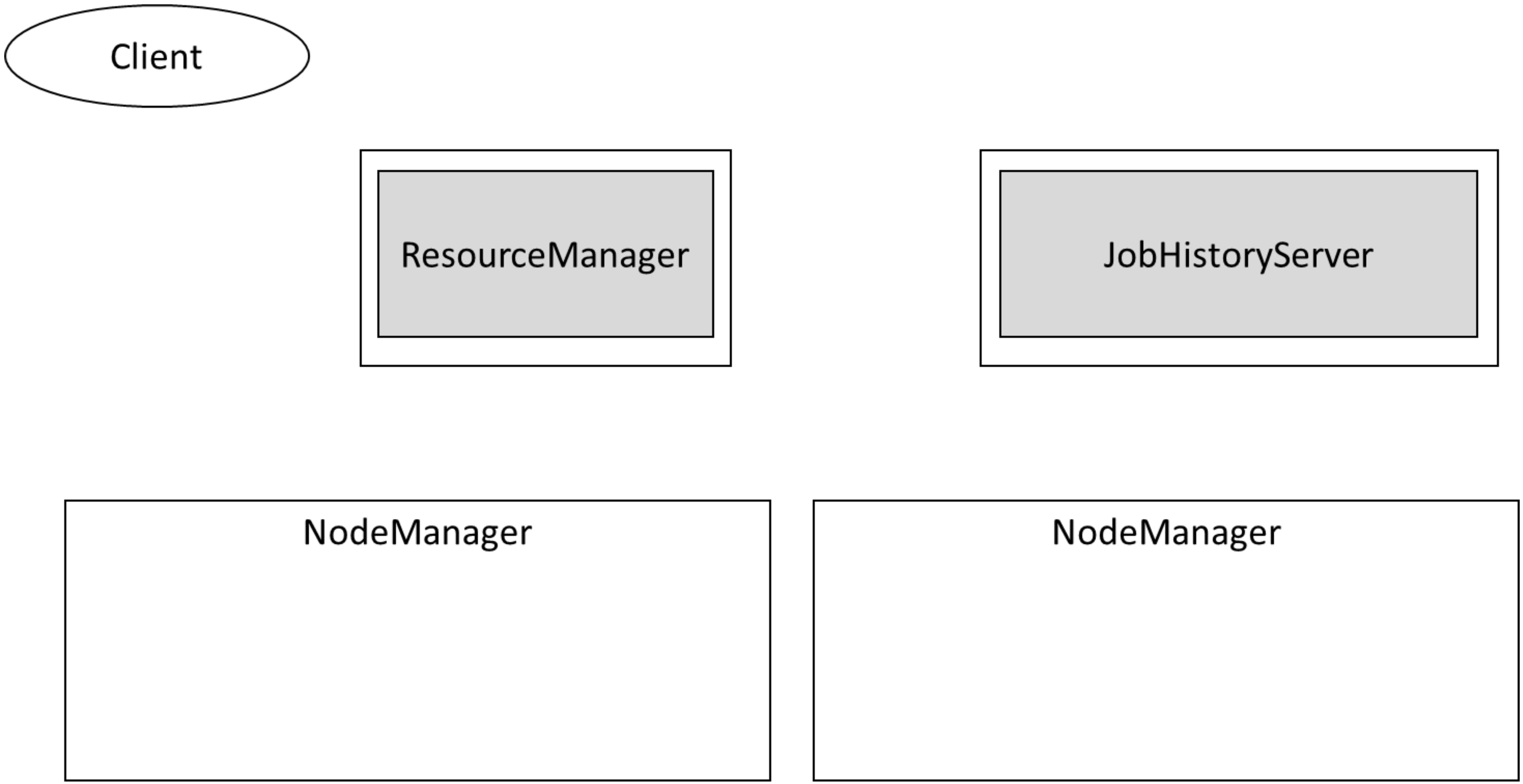
# MapReduce

- Constructing MapReduce programs requires a certain skillset in terms of programming
- Tradeoffs in terms of speed, memory consumption, and scalability

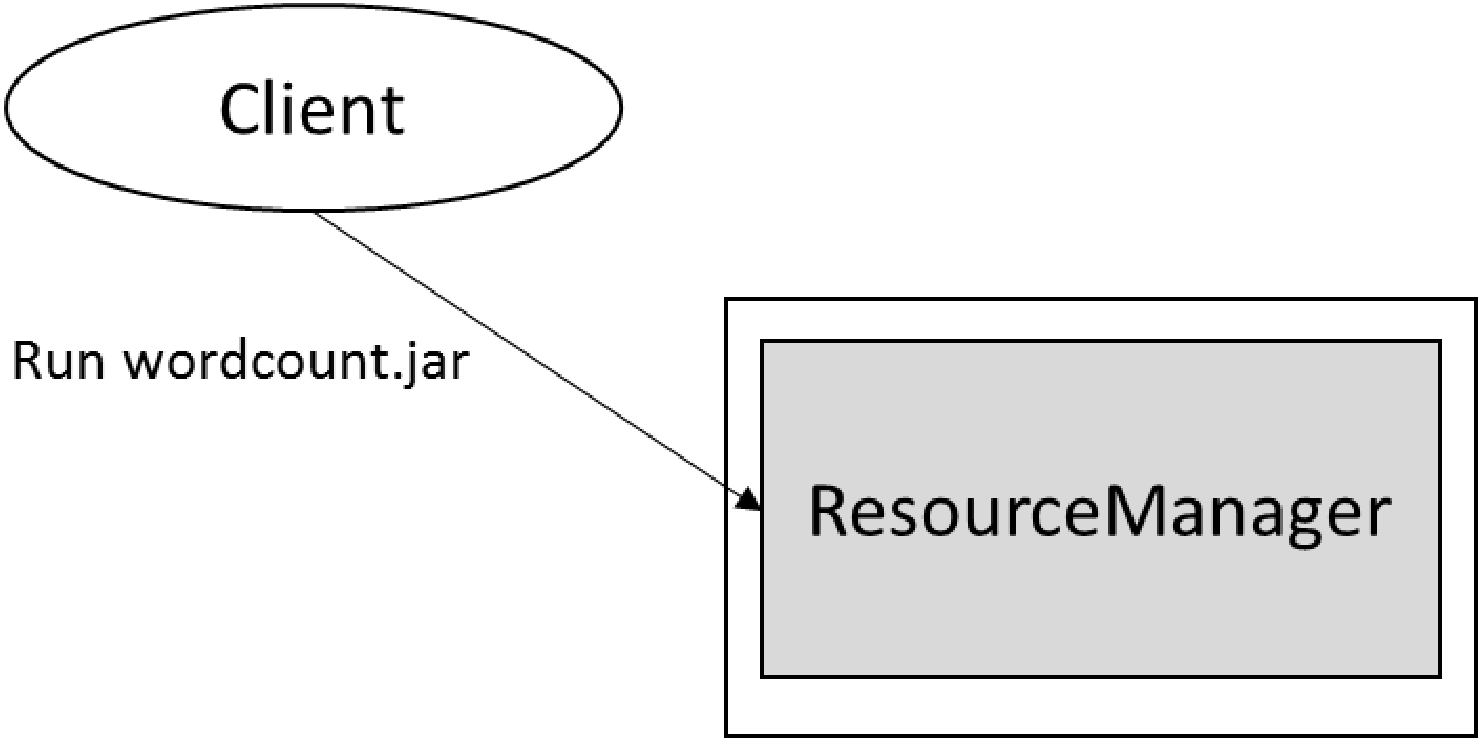
# Yet Another Resource Negotiator (YARN)

- Yet Another Resource Negotiator (YARN) distributes a MapReduce program across different nodes and takes care of coordination
- Three important services
  - ResourceManager: a global YARN service that receives and runs applications (e.g., a MapReduce job) on the cluster
  - JobHistoryServer: keeps a log of all finished jobs
  - NodeManager: responsible to oversee resource consumption on a node

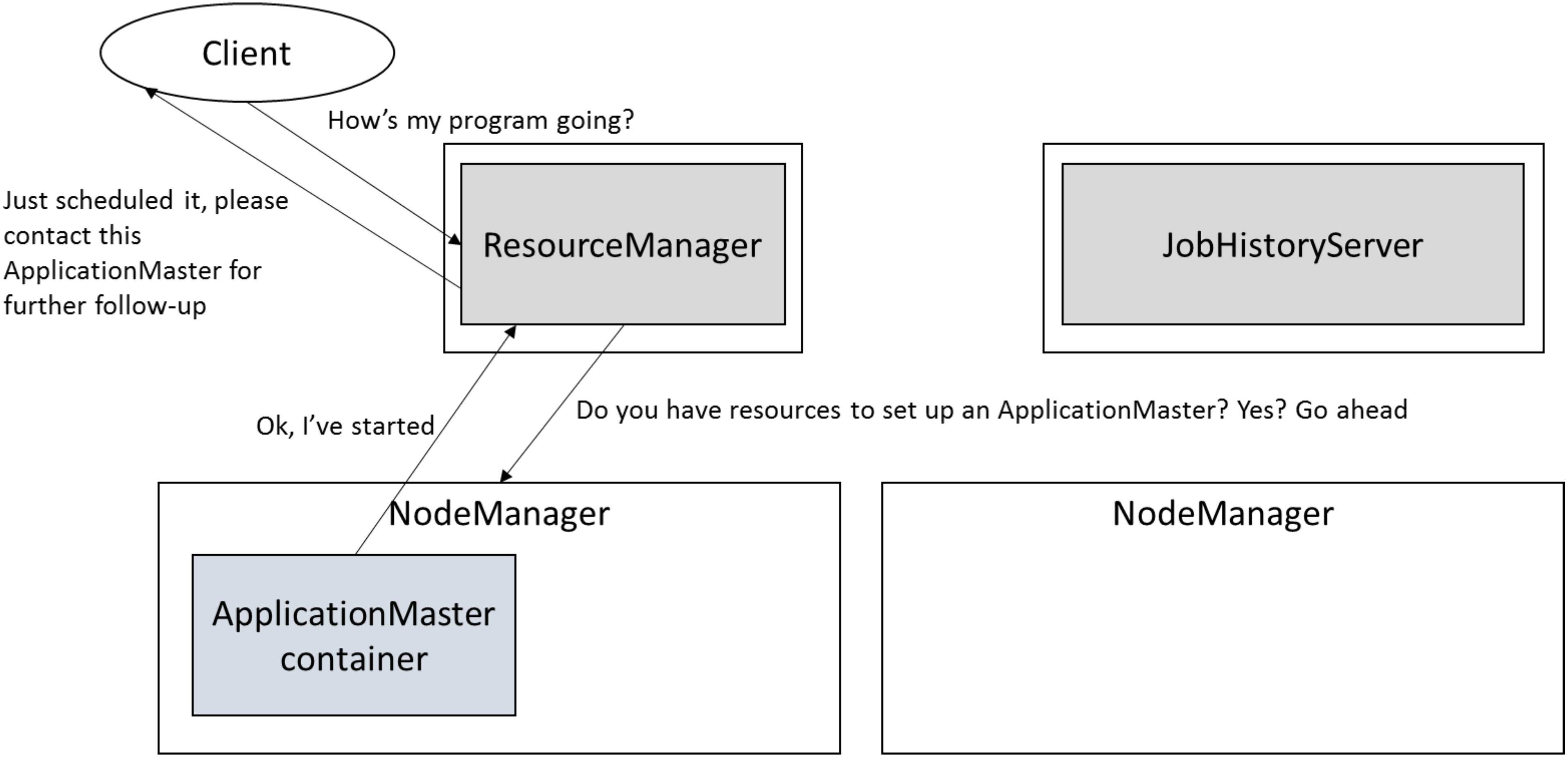
# Yet Another Resource Negotiator (YARN)



# Yet Another Resource Negotiator (YARN)



# Yet Another Resource Negotiator (YARN)



Client

How's my program going?

Just scheduled it, please contact this ApplicationMaster for further follow-up

ResourceManager

JobHistoryServer

Ok, I've started

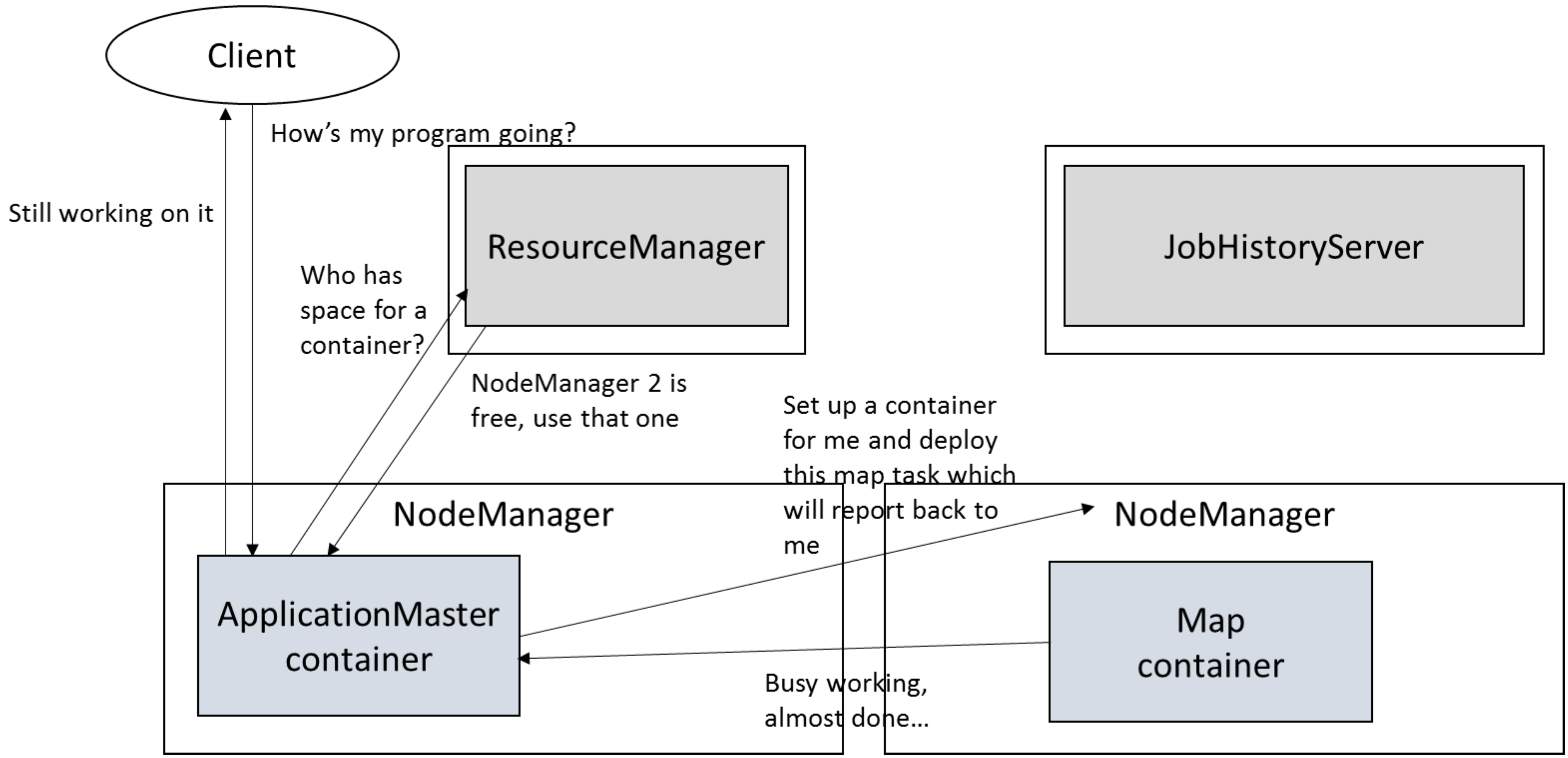
Do you have resources to set up an ApplicationMaster? Yes? Go ahead

NodeManager

ApplicationMaster container

NodeManager

# Yet Another Resource Negotiator (YARN)


Client
How's my program going?
Still working on it
ResourceManager
JobHistoryServer
Who has space for a container?
NodeManager 2 is free, use that one
Set up a container for me and deploy this map task which will report back to me
NodeManager
ApplicationMaster container
NodeManager
Map container
Busy working, almost done...

# Yet Another Resource Negotiator (YARN)

- Complex setup
- Allows to run programs and applications other than MapReduce

# SQL on Hadoop

- MapReduce very complex when compared to SQL
- Need for a more database-like setup on top of Hadoop

# SQL on Hadoop

- HBase
- Pig
- Hive

# HBase

- First Hadoop database inspired by Google’s Bigtable
- Runs on top of HDFS
- NoSQL alike data storage platform
  - No typed columns, triggers, advanced query capabilities, etc.
- Offers a simplified structure and query language in a way that is highly scalable and can tackle large volumes

# HBase

- Similar to RDBMSs, HBase organizes data in tables with rows and columns
- HBase table consists of multiple rows
- A row consists of a row key and one or more columns with values associated with them
- Rows in a table are sorted alphabetically by the row key

# HBase

- Each column in HBase is denoted by a column family and qualifier (separated by a colon, ‘:’)
- A column family physically co-locates a set of columns and their values
- Every row has the same column families, but not all column families need to have a value per row
- Each cell in a table is hence defined by a combination of the row key, column family and column qualifier, and a timestamp

# HBase

- Example: HBase table to store and query users
- The row key will be the user id
- column families:qualifiers
  - name:first
  - name:last
  - email (without a qualifier)

# HBase

```
hbase(main):001:0> create 'users', 'name', 'email'
0 row(s) in 2.8350 seconds

=> Hbase::Table - users



hbase(main):002:0> describe 'users'
Table users is ENABLED
users
COLUMN FAMILIES DESCRIPTION
{NAME => 'email', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', K
EEP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', C
OMPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '6
5536', REPLICATION_SCOPE => '0'}
{NAME => 'name', BLOOMFILTER => 'ROW', VERSIONS => '1', IN_MEMORY => 'false', KE
EP_DELETED_CELLS => 'FALSE', DATA_BLOCK_ENCODING => 'NONE', TTL => 'FOREVER', CO
MPRESSION => 'NONE', MIN_VERSIONS => '0', BLOCKCACHE => 'true', BLOCKSIZE => '65
536', REPLICATION_SCOPE => '0'}
2 row(s) in 0.3250 seconds
```

# HBase

```
hbase(main):003:0> list 'users'
TABLE
users
1 row(s) in 0.0410 seconds

=> ["users"]
```

# HBase

```
hbase(main):006:0> put 'users', 'seppe', 'name:first', 'Seppe'
0 row(s) in 0.0200 seconds

hbase(main):007:0> put 'users', 'seppe', 'name:last', 'vanden Broucke'
0 row(s) in 0.0330 seconds

hbase(main):008:0> put 'users', 'seppe', 'email', 'seppe.vandenbroucke@kuleuven'
0 row(s) in 0.0570 seconds


hbase(main):009:0> scan 'users'
ROW                   COLUMN+CELL
 seppe                column=email:, timestamp=1495293082872, value=seppe.vanden
                      broucke@kuleuven.be
 seppe                column=name:first, timestamp=1495293050816, value=Seppe
 seppe                column=name:firstt, timestamp=1495293047100, value=Seppe
 seppe                column=name:last, timestamp=1495293067245, value=vanden Broucke

1 row(s) in 0.1170 seconds
```

# HBase

```
hbase(main):011:0> get 'users', 'seppe'
COLUMN                          CELL
 email:                         timestamp=1495293082872,
value=seppe.vandenbroucke@kuleuven.be
 name:first                     timestamp=1495293050816, value=Seppe
 name:last                      timestamp=1495293067245, value=vanden Broucke
4 row(s) in 0.1250 seconds
```

```
hbase(main):018:0> put 'users', 'seppe', 'email', 'seppe@kuleuven.be'
0 row(s) in 0.0240 seconds
```

```
hbase(main):019:0> get 'users', 'seppe', 'email'
COLUMN                          CELL
 email:                         timestamp=1495293303079, value=seppe@kuleuven.be
1 row(s) in 0.0330 seconds
```

# HBase

- HBase’s query facilities are very limited
- Essentially a key-value, distributed data store with simple get/put operations
- Includes facilities to write MapReduce programs
- Hbase (similar to Hadoop) doesn’t perform well on less than 5 HDFS DataNodes with an additional NameNode
  - only makes the effort worthwhile when you can invest in, set up and maintain at least 6-10 nodes

# Pig

- Yahoo! Developed “Pig”, which was made open source as Apache Pig in 2007
- High-level platform for creating programs that run on Hadoop (in Pig Latin), which uses MapReduce underneath
- Somewhat resembles querying facilities of SQL

# Pig

```
timesheet = LOAD 'timesheet.csv' USING PigStorage(',');
raw_timesheet = FILTER timesheet by $0 > 100;
timesheet_logged = FOREACH raw_timesheet GENERATE $0 AS
driverId, $2 AS hours_logged, $3 AS miles_logged;
grp_logged = GROUP timesheet_logged by driverId;
sum_logged = FOREACH grp_logged GENERATE group as driverId,
SUM(timesheet_logged.hours_logged) as sum_hourslogged,
SUM(timesheet_logged.miles_logged) as sum_mileslogged;
```

# Pig

- Some have argued that RDBMSs and SQL are substantially faster than MapReduce – and hence Pig
- Pig Latin is relatively procedural versus declarative SQL
- No wide adoption

# Hive

- Initially developed by Facebook but open-sourced afterwards
- Data warehouse solution offering SQL querying facilities on top of Hadoop
- Converts SQL-like queries to a MapReduce pipeline
- Also offers a JDBC and ODBC interface
- Can run on top of HDFS, as well as other file systems

# Hive

- Hive Metastore stores metadata for each table such as its schema and location on HDFS (using an RDBMS)
- Driver service is responsible to receive and handle incoming queries
  - query is first converted to an abstract syntax tree, which is then converted to a directed acyclic graph representing an execution plan
  - the directed acyclic graph will contain a number of MapReduce stages and tasks
- Optimizer optimizes the directed acyclic graph
- Executer sends MapReduce stages to Hadoop’s resource manager (e.g. YARN) and monitor their progress

# Hive

- HiveQL does not completely follow the full SQL-92 standard
  - E.g., lacks strong support for indexes, transactions, materialized views, and only has limited subquery support
- Example:

```
SELECT genre, SUM(nrPages) FROM books
GROUP BY genre
```

- HiveQL also allows to query data sets other than structured tables

# Hive

```
CREATE TABLE docs (line STRING); -- create a docs table

-- load in file from HDFS to docs table, overwrite existing data:
LOAD DATA INPATH '/users/me/doc.txt' OVERWRITE INTO TABLE docs;

-- perform word count
SELECT word, count(1) AS count
FROM ( -- split each line in docs into words
  SELECT explode(split(line, '\s')) AS word FROM docs
) t
GROUP BY t.word
ORDER BY t.word;
```

# Hive

- One difference with traditional RDBMS is that Hive does not enforce the schema at the time of loading the data
  - Hive: schema-on-read
  - RDBMS: schema-on-write
- Recent versions of Hive support full ACID transaction management
- Performance and speed of SQL queries still forms the main disadvantage of Hive today
  - Solutions to bypass MapReduce (e.g. Apache Tez, Cloudera Impala, Facebook Presto)

# Apache Spark

- Open-source alternative for MapReduce
- New programming paradigm centered on a data structure called the resilient distributed dataset (RDD) which can be distributed across a cluster of machines and is maintained in a fault tolerant way
- RDDs can enable the construction of iterative programs that have to visit a data set multiple times, as well as more interactive or exploratory programs
- Many orders of magnitude faster than MapReduce implementations
- Rapidly adopted by many Big Data vendors

# Apache Spark

- Similar to Hadoop, Spark works with HDFS and requires a cluster manager (e.g. YARN)
- Key components
  - Spark Core
  - Spark SQL
  - MLib, Spark Streaming, GraphX

# Spark Core

- Foundation for all other components
- Provides functionality for task scheduling and a set of basic data transformations that can be used through many programming languages (e.g., Java, Python, Scala, and R)
- RDDs are the primary data abstraction in Spark
  - designed to support in-memory data storage and operations, distributed across a cluster

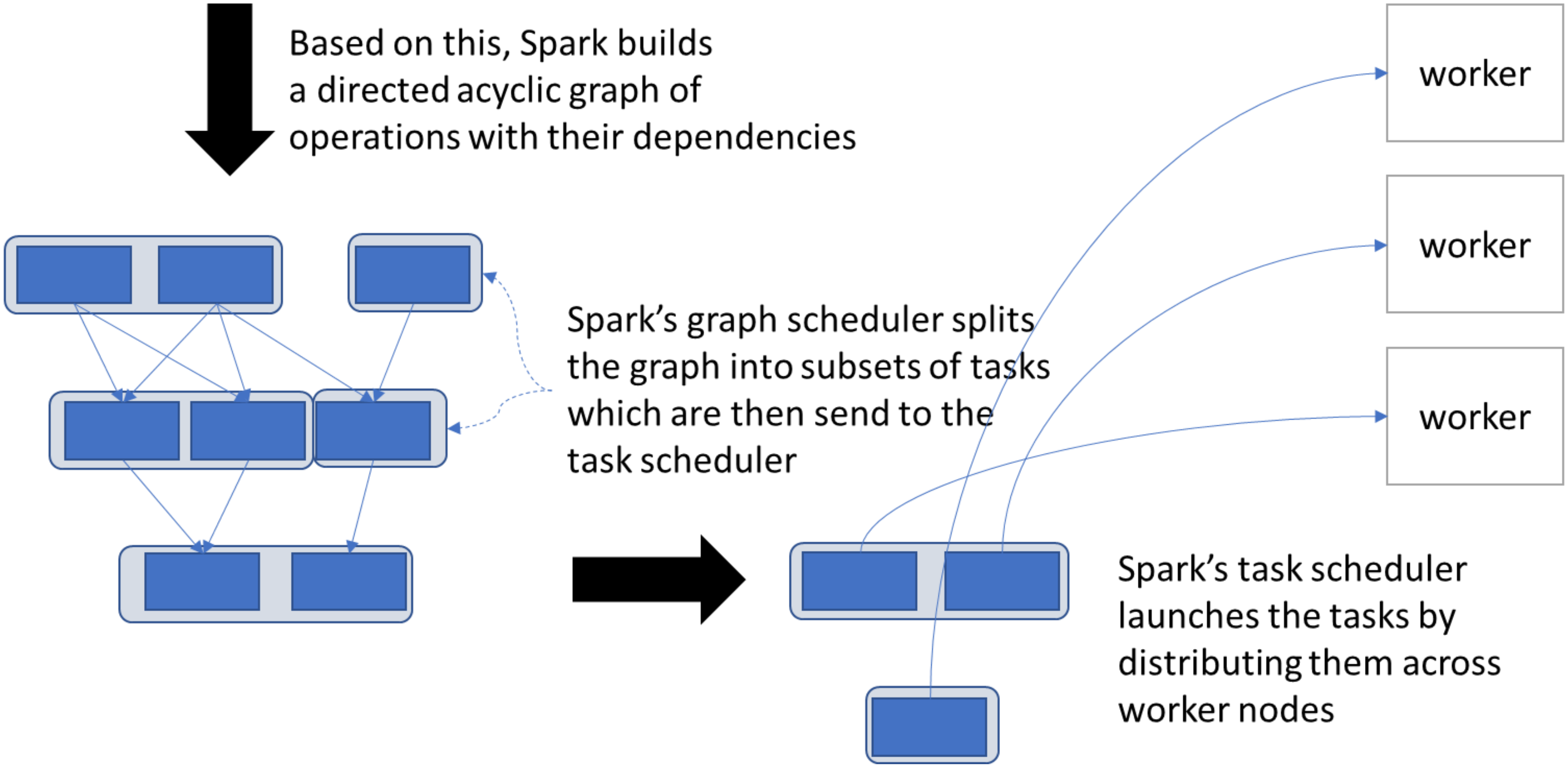
# Spark Core

- Once data is loaded into an RDD, two basic types of operations can be performed:
  - Transformation which creates a new RDD through changing the original one
  - Actions which measure but do not change the original data
- Transformations are lazily evaluated
  - executed when a subsequent action has a need for the result
- RDDs will also be kept as long as possible in memory
- A chain of RDD operations gets compiled by Spark into a directed acyclic graph but which is then spread out and calculated over the cluster

# Spark Core

A programmer writes a Spark program using its API:

```
rdd1.join(rdd2).groupBy(…).filter(…)
```



Based on this, Spark builds a directed acyclic graph of operations with their dependencies

Spark's graph scheduler splits the graph into subsets of tasks which are then send to the task scheduler

Spark's task scheduler launches the tasks by distributing them across worker nodes

# Spark Core

- Spark’s RDD API is relatively easy to work with compared to writing MapReduce programs

```
# Set up connection to the Spark cluster
sconf = SparkConf()
sc = SparkContext(master='', conf=sconf)

# Load in an RDD from a text file, the RDD will represent a collection of
# text strings (one for each line)
text_file = sc.textFile("myfile.txt")

# Count the word occurrences
counts = text_file.flatMap(lambda line: line.split(" ")) \
.map(lambda word: (word, 1)) \
.reduceByKey(lambda a, b: a + b)

print(counts)
```

# Spark SQL

- Spark SQL runs on top of Spark Core and introduces another data abstraction called DataFrames
- DataFrames can be created from RDDs by specifying a schema on how to structure the data elements in the RDD, or can be loaded in directly from various sorts of file formats
- Even although DataFrames continue to use RDDs behind the scenes, they represent themselves to the end user as a collection of data organized into named columns

# Spark SQL

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("Spark example").getOrCreate()

# Create a DataFrame object by reading in a file
df = spark.read.json("people.json")

df.show()
# | age|    name|
# +----+--------+
# |null|   Seppe|
# |  30|Wilfried|
# |  19|    Bart|
# +----+--------+

# DataFrames are structured in columns and rows:
df.printSchema()
# root
# |-- age: long (nullable = true)
# |-- name: string (nullable = true)
```

# Spark SQL

```
df.select("name").show()
# +--------+
# |    name|
# +--------+
# |   Seppe|
# |Wilfried|
# |    Bart|
# +--------+

# SQL-like operations can now easily be expressed:
df.select(df['name'], df['age'] + 1).show()
# +--------+---------+
# |    name|(age + 1)|
# +--------+---------+
# |   Seppe|     null|
# |Wilfried|       31|
# |    Bart|       20|
# +--------+---------+
```

# Spark SQL

```
df.filter(df['age'] > 21).show()
# +---+--------+
# |age|    name|
# +---+--------+
# | 30|Wilfried|
# +---+--------+

df.groupBy("age").count().show()
# +----+-----+
# | age|count|
# +----+-----+
# |  19|    1|
# |null|    1|
# |  30|    1|
# +----+-----+
```

# Spark SQL

- Spark implements a full SQL query engine which can convert SQL statements to a series of RDD transformations and actions

```
# Register the DataFrame as a SQL temporary view
df.createOrReplaceTempView("people")

sqlDF = spark.sql("SELECT * FROM people WHERE age > 21")
sqlDF.show()

# +---+--------+
# |age|    name|
# +---+--------+
# | 30|Wilfried|
# +---+--------+
```

# MLlib, Spark Streaming and GraphX

- MLlib is Spark’s machine learning library
  - offers classification, regression, clustering, and recommender system algorithms
- MLlib was originally built directly on top of the RDD abstraction
- New MLlib version works directly with SparkSQL’s DataFrames based API

# MLlib, Spark Streaming and GraphX

- Spark Streaming leverages Spark Core and its fast scheduling engine to perform streaming analytics
- Spark Streaming provides another high-level concept called the DStream, which represents a continuous stream of data
  - represented as a sequence of RDD fragments
- DStreams provide windowed computations

# MLlib, Spark Streaming and GraphX

- Example: Word counting

```
from pyspark import SparkContext
from pyspark.streaming import StreamingContext
sc = SparkContext("local[2]", "StreamingWordCount")
ssc = StreamingContext(sc, 1)

# Create a DStream that will connect to server.mycorp.com:9999 as a source
lines = ssc.socketTextStream("server.mycorp.com ", 9999)

# Split each line into words
words = lines.flatMap(lambda line: line.split(" "))

# Count each word in each batch
pairs = words.map(lambda word: (word, 1))
wordCounts = pairs.reduceByKey(lambda x, y: x + y)

# Print out first ten elements of each RDD generated in the wordCounts Dstream
wordCounts.pprint()

# Start the computation
ssc.start()
ssc.awaitTermination()
```

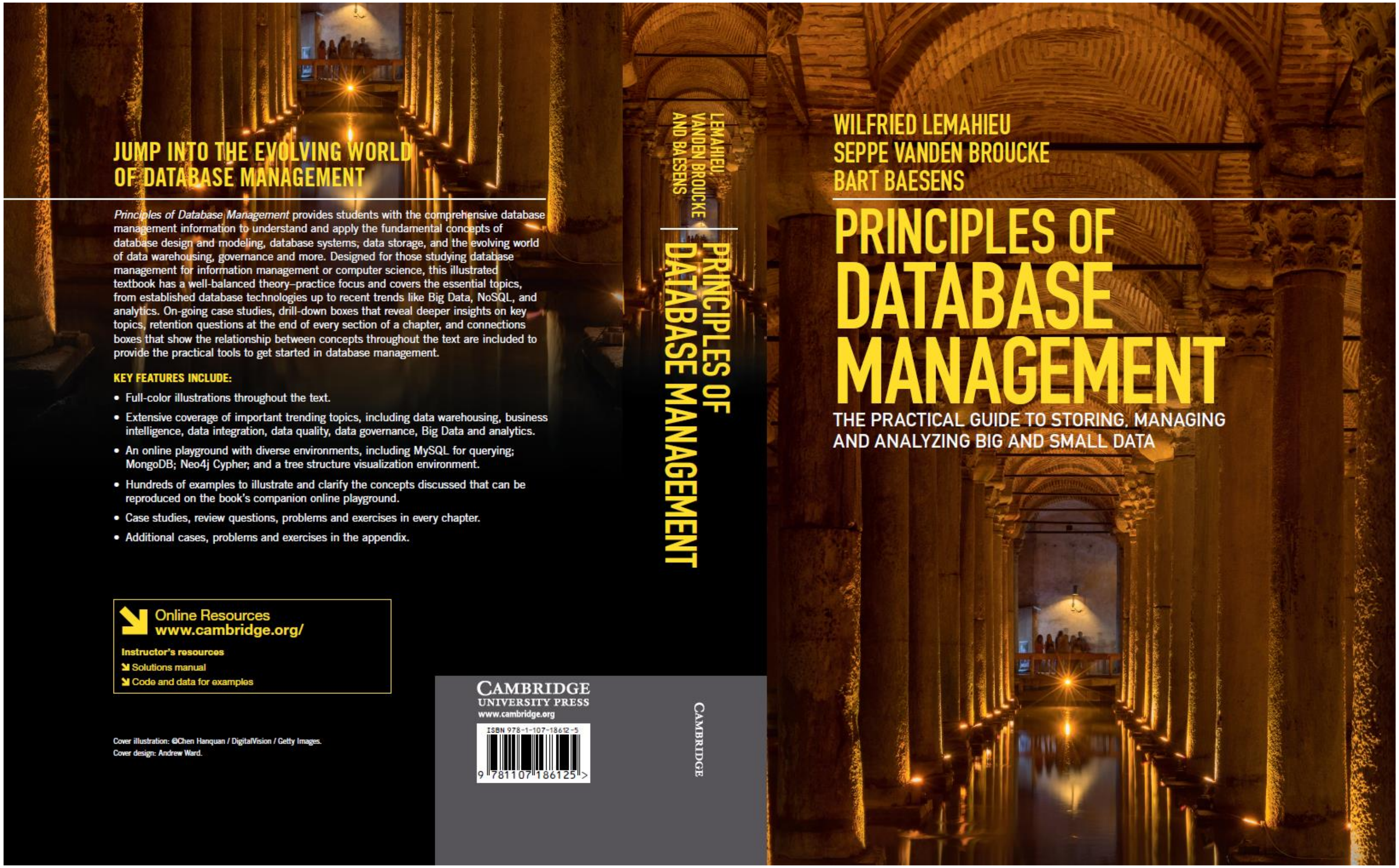
# MLlib, Spark Streaming and GraphX

- GraphX is Spark’s component implementing programming abstractions to deal with graph based structures, again based on the RDD abstraction
- GraphX comes with a set of fundamental operators and algorithms to work with graphs and simplify graph analytics tasks

# Conclusion

- The 5 V’s of Big Data
- Hadoop
- SQL on Hadoop
- Apache Spark

# More information?



JUMP INTO THE EVOLVING WORLD OF DATABASE MANAGEMENT

*Principles of Database Management* provides students with the comprehensive database management information to understand and apply the fundamental concepts of database design and modeling, database systems, data storage, and the evolving world of data warehousing, governance and more. Designed for those studying database management for information management or computer science, this illustrated textbook has a well-balanced theory–practice focus and covers the essential topics, from established database technologies up to recent trends like Big Data, NoSQL, and analytics. On-going case studies, drill-down boxes that reveal deeper insights on key topics, retention questions at the end of every section of a chapter, and connections boxes that show the relationship between concepts throughout the text are included to provide the practical tools to get started in database management.

**KEY FEATURES INCLUDE:**

- Full-color illustrations throughout the text.
- Extensive coverage of important trending topics, including data warehousing, business intelligence, data integration, data quality, data governance, Big Data and analytics.
- An online playground with diverse environments, including MySQL for querying; MongoDB; Neo4j Cypher; and a tree structure visualization environment.
- Hundreds of examples to illustrate and clarify the concepts discussed that can be reproduced on the book's companion online playground.
- Case studies, review questions, problems and exercises in every chapter.
- Additional cases, problems and exercises in the appendix.

Online Resources
**www.cambridge.org/**

**Instructor's resources**
Solutions manual
Code and data for examples

Cover illustration: ©Chen Hanquan / DigitalVision / Getty Images.
Cover design: Andrew Ward.

CAMBRIDGE
UNIVERSITY PRESS
www.cambridge.org

ISBN 978-1-107-18612-5

LEMAHIEU
VANDEN BROUCKE
AND BAESENS

PRINCIPLES OF
DATABASE MANAGEMENT

CAMBRIDGE

WILFRIED LEMAHIEU
SEPPE VANDEN BROUCKE
BART BAESENS

PRINCIPLES OF
DATABASE
MANAGEMENT

THE PRACTICAL GUIDE TO STORING, MANAGING
AND ANALYZING BIG AND SMALL DATA

[www.pdbmbook.com](www.pdbmbook.com)