Zurich University
of Applied Sciences

# Information Engineering 2

## Data Storage:
## HDFS & Parquet

Prof. Dr. Kurt Stockinger

# Semesterplan

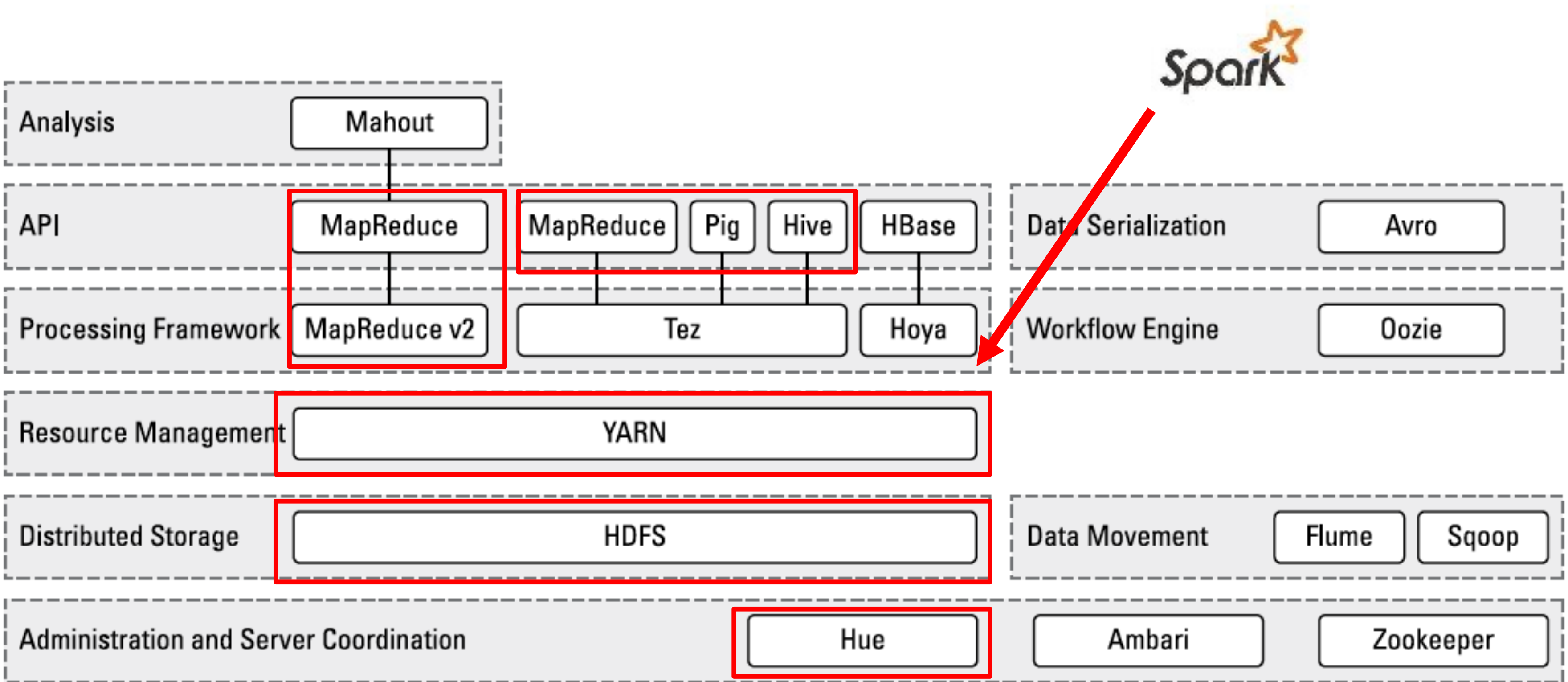
| SW | Datum | Vorlesungsthema | Praktikum |
|---|---|---|---|
| 1 | 23.02.2022 | Data Warehousing Einführung | Praktikum 1: KNIME Tutorial |
| 2 | 02.03.2022 | Dimensionale Datenmodellierung 1 | Praktikum 1: KNIME Tutorial (Vertiefung) |
| 3 | 09.03.2022 | Dimensionale Datenmodellierung 2 | Praktikum 2: Datenmodellierung |
| 4 | 16.03.2022 | Datenqualität und Data Matching | Praktikum 3: Star-Schema, Bonus: Praktikum 4: Slowly Changing Dimensions |
| 5 | 23.03.2022 | Big Data Einführung | DWH Projekt - Teil 1 |
| 6 | 30.03.2022 | Spark - Data Frames | DWH Projekt - Teil 2 (Abgabe: 4.4.2022 23:59:59) |
| 7 | 06.04.2022 | Data Storage: Hadoop Distributed File System & Parquet | Praktikum 1: Data Frames |
| 8 | 13.04.2022 | Query Optimization | Praktikum 2: Data Storage |
| 9 | 20.04.2022 | Spark Best Practices & Applications | Praktikum 3: Query Optimization & Performance Analysis |
| 10 | 27.04.2022 | Machine Learning mit Spark 1 | Praktikum 3: Query Optimization & Performance Analysis (Vertiefung) |
| 11 | 04.05.2022 | Machine Learning mit Spark 2 + Q&A | Praktikum 4: Machine Learning (Regression) |
| 12 | 11.05.2022 | NoSQL Systems | Big Data Projekt - Teil 1 |
| 13 | 18.05.2022 | Keine Vorlesung (Arbeit am Projekt) | Big Data Projekt - Teil 2 |
| 14 | 25.05.2022 | Keine Vorlesung (Arbeit am Projekt) | Big Data Projekt - Teil 3 (Abgabe: 30.5.2022 23:59:59) |

# Learning Objectives for Today

- Learn about concepts of Hadoop Distributed File System (HDFS)
- Understand data distribution and replication
- Learn about Parquet file storage
- Understand performance implications of using Parquet storage

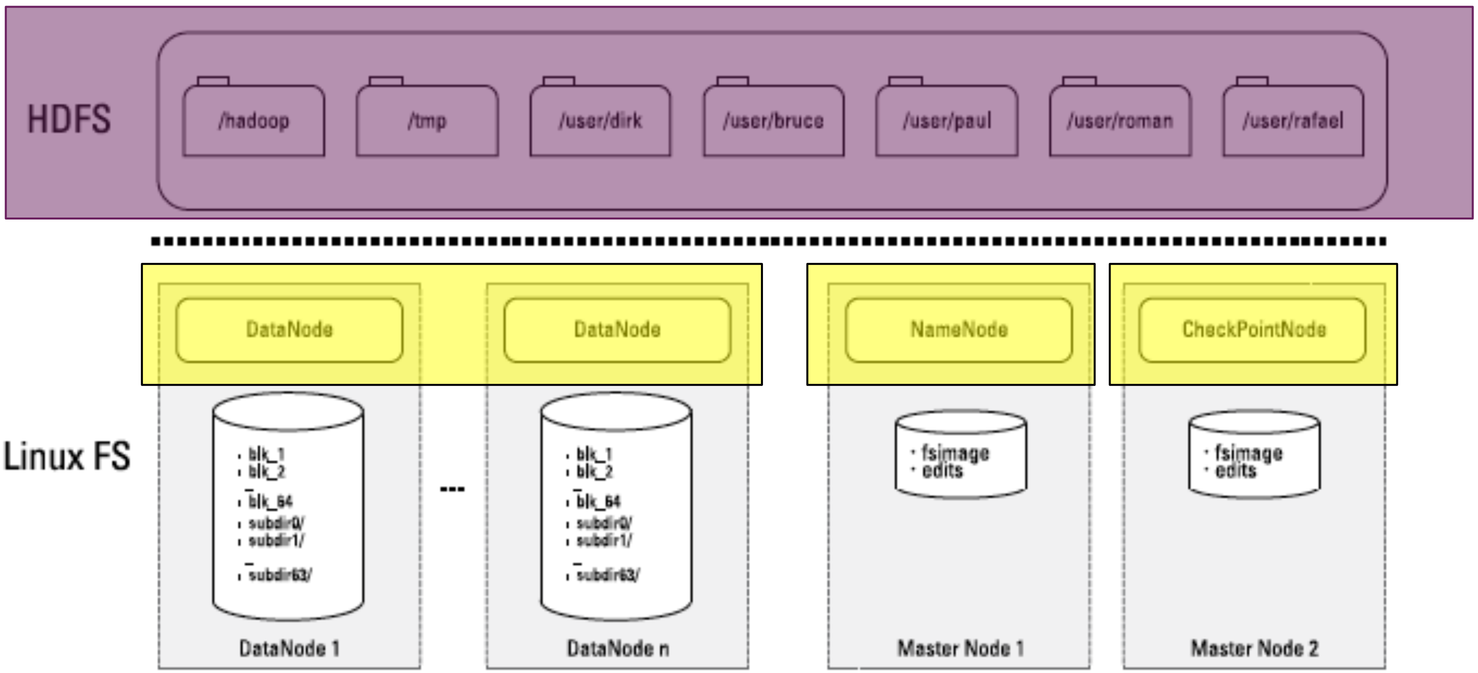
# Hadoop

# Architecture of Hadoop



# Hadoop Distributed File System

- Parallel file system for managing large amounts of data
- Runs on top of Linux file system
- Data is distributed onto n machines



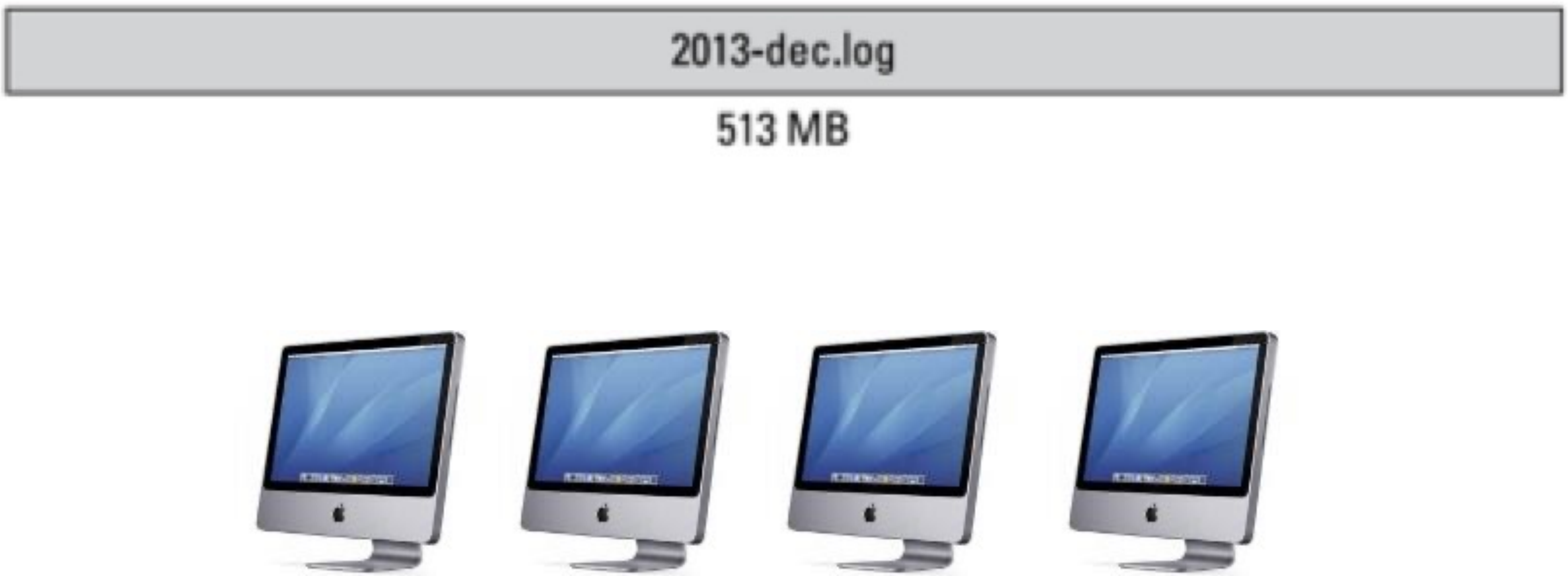
# Managing Data in HDFS

- Starting situation:
  - Large log file about click streams (access to web pages)
  - HDFS with 4 machines

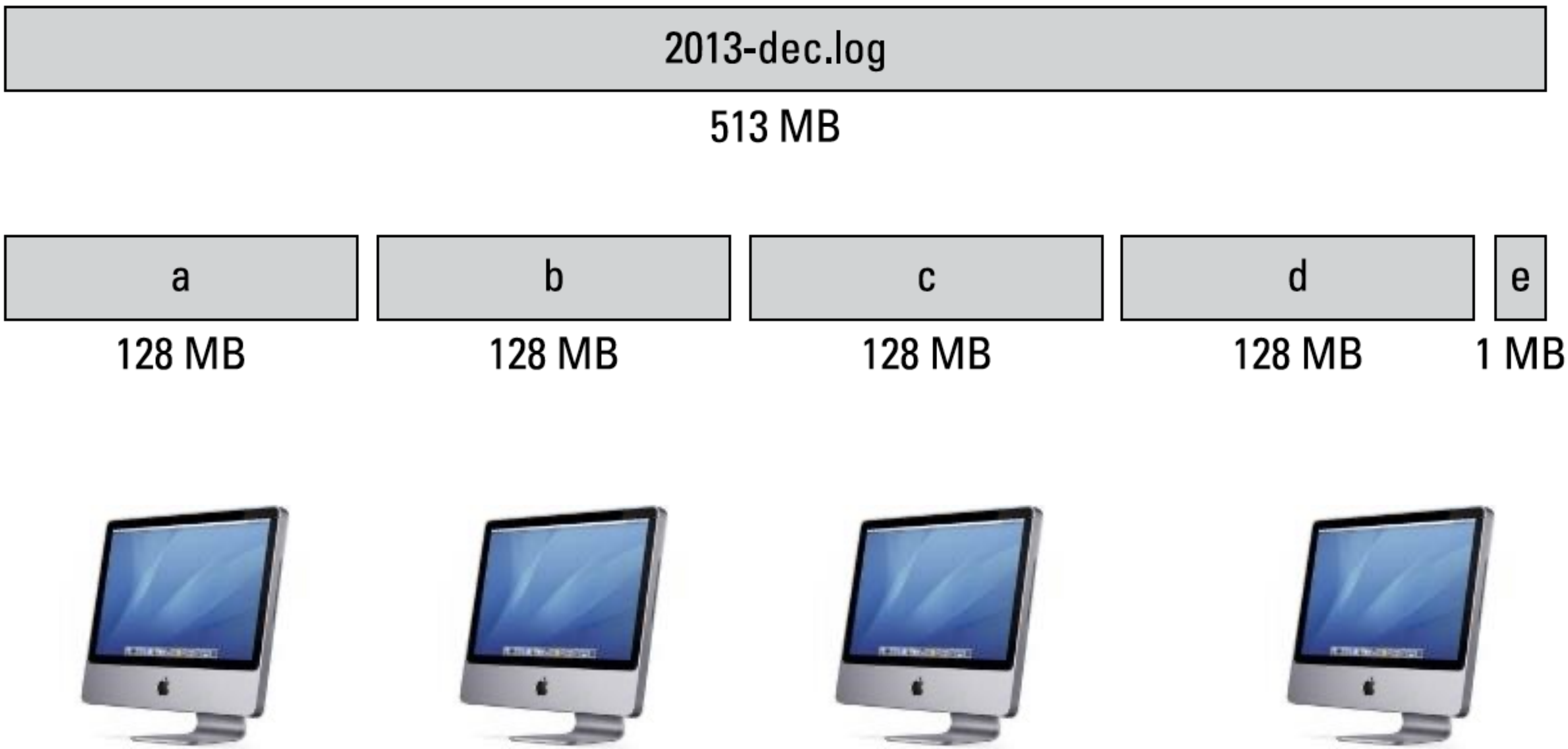
- How should the file be distributed onto the 4 machines?

# Starting Situation

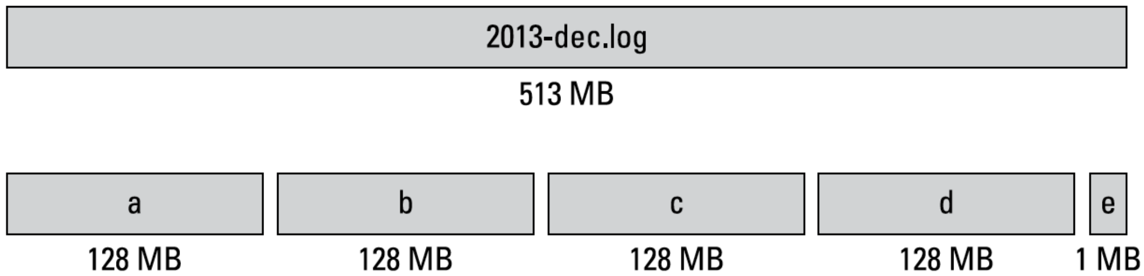
- File of 513 MB



# Distribution of the Data Blocks onto Machines



# Distribution of the Data Blocks onto Machines



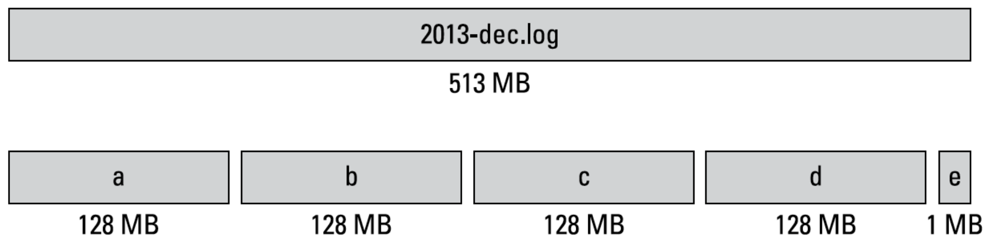

- HDFS block size: typically 128 MB
- Data blocks are distributed irrespective of content
- Goal:
  - Even distribution of blocks to yield highest possible parallelization factor

- Conflict?
  - Isn't it better to consider data content for distribution?

# Distribution of Data Blocks to Map-Tasks

- Input data: "… This is a very long text with a size of 513 MB. … "

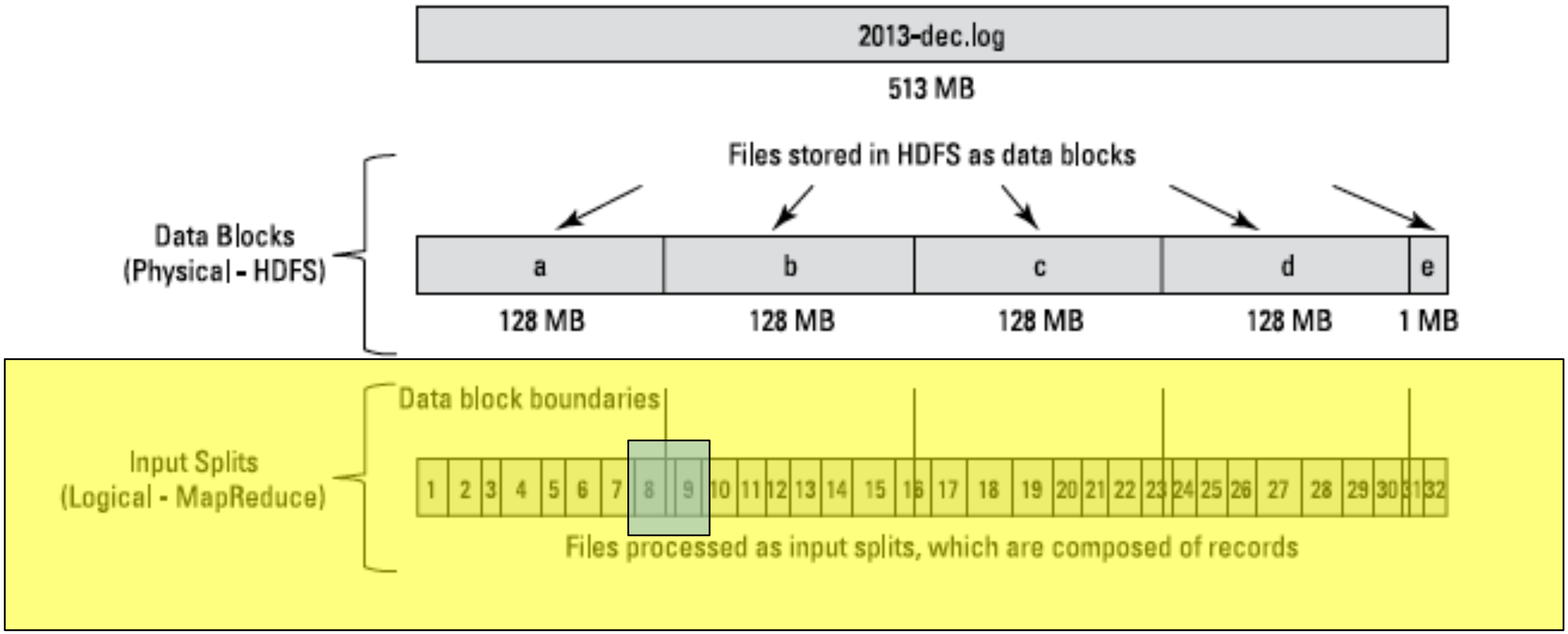
128 MB 128 MB




- How are the blocks distributed to mappers?
- Are words split (e.g. "text")?

# Distribution of Data Blocks to Map-Tasks

In MapReduce data is read record wise (key/value)



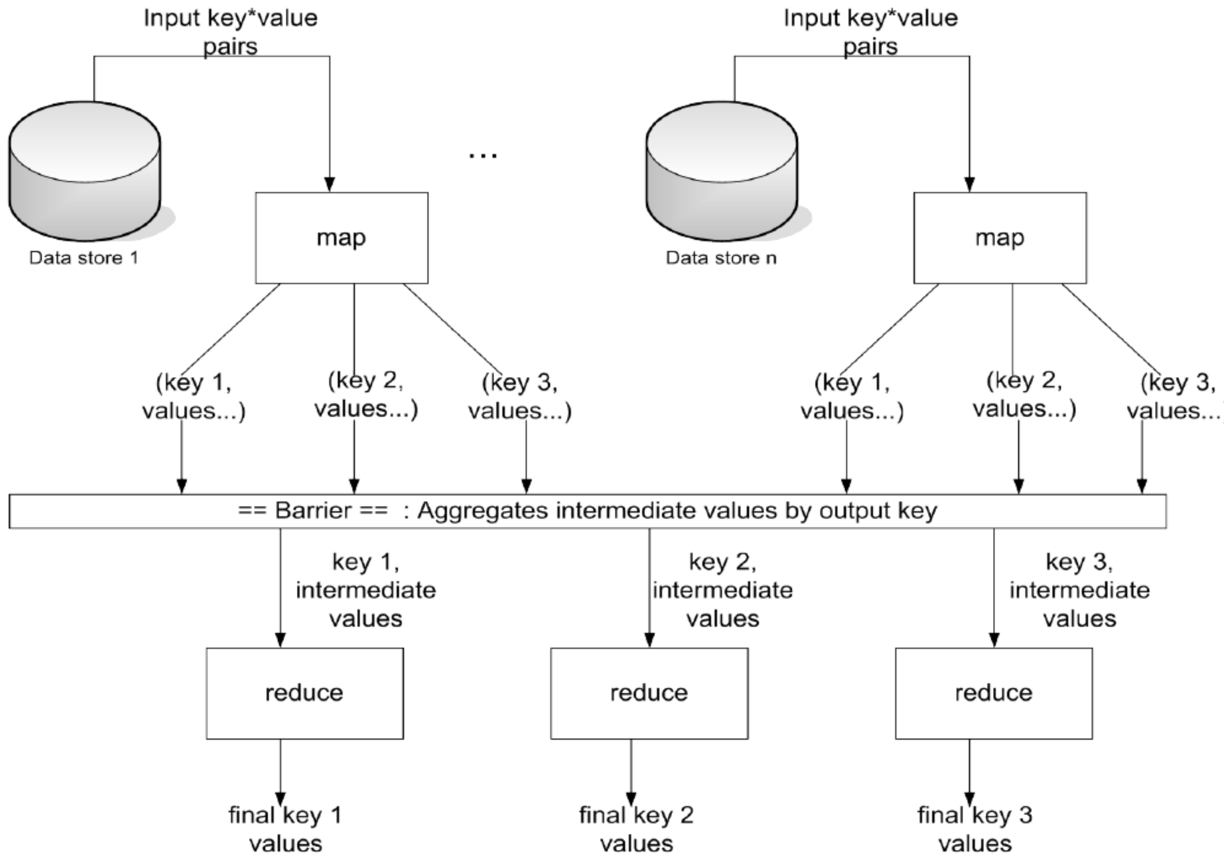

map: (K1, V1) → list(K2, V2)

reduce: (K2, list(V2)) → list(K3, V3)

# Mapper - Details

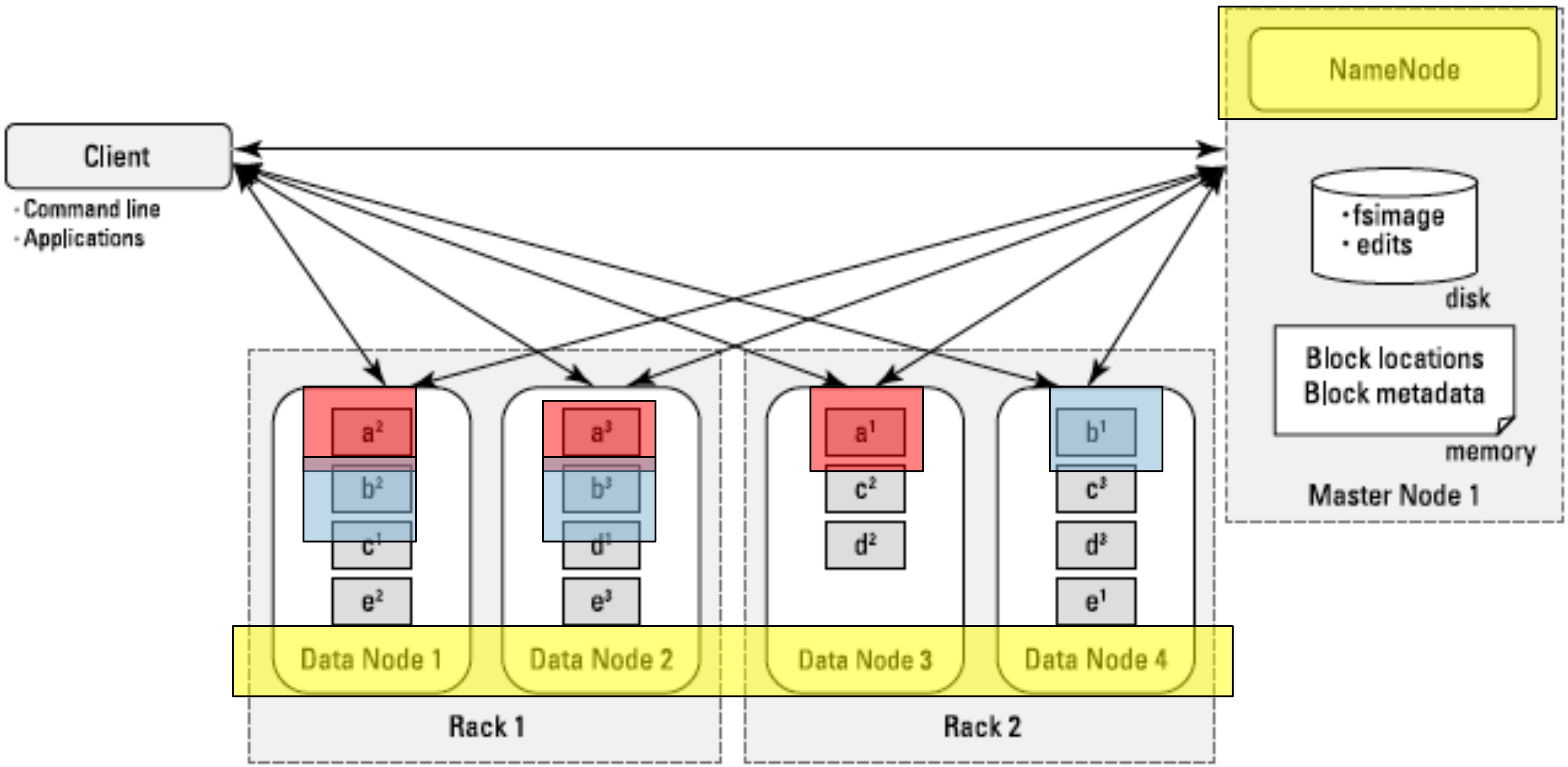
- Data is read line by line
- One mapper task per input block
- Logical separation if data is larger than size of input block:
    - No separation of single fields

# Processing Model of MapReduce




Source: Big Data Vorlesung, Kossmann, Tatbul 2012

# Data Distribution and Replication

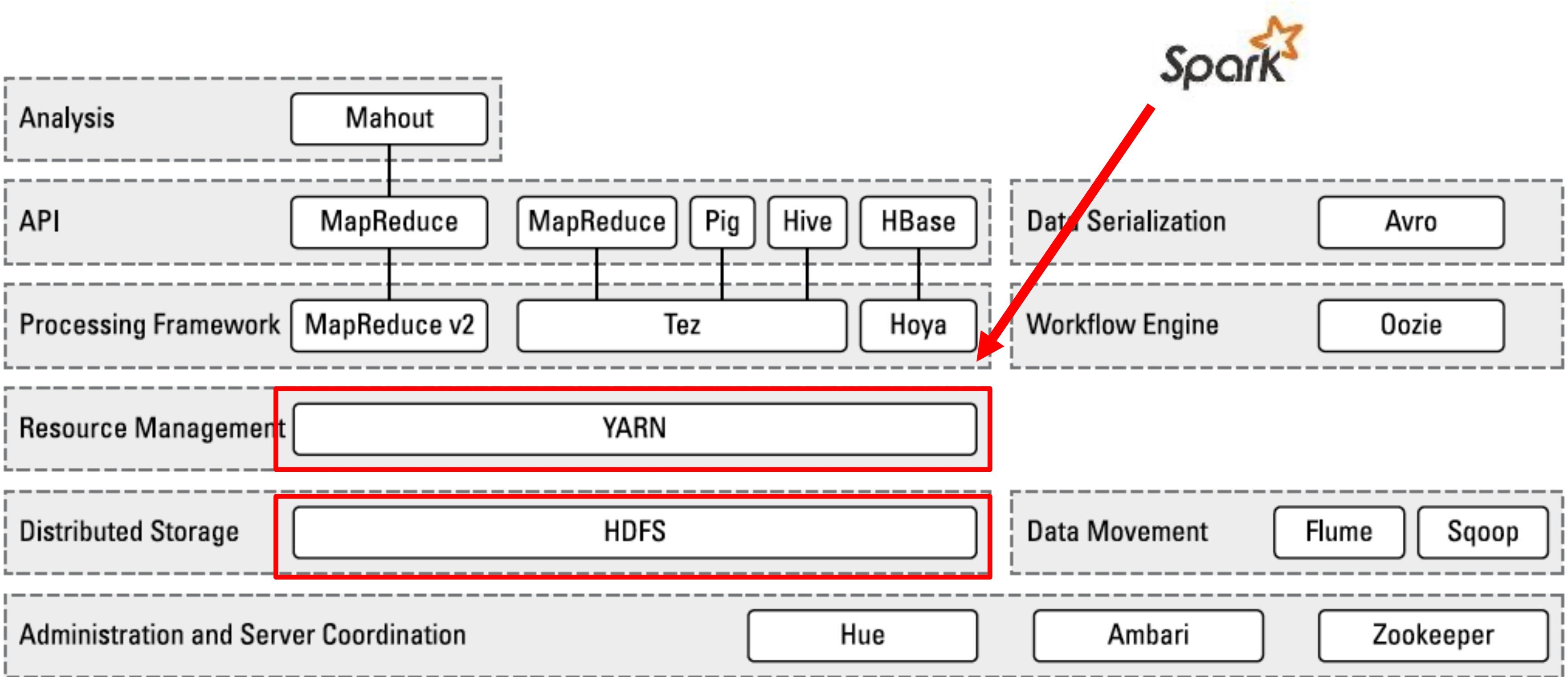



Default replication factor: 3

# Relevance for HDFS

- HDFS is the basis for parallel data processing in Hadoop
- Can easily be combined with Spark
- Can also be used as a file system without using MapReduce (e.g. via Spark)

# Reflection of Architecture of Hadoop: How does Spark fit in?



# Most important HDFS commands

# Copy Data from Linux to HDFS

- File: test.txt

- Copy from Linux to HDFS:
  - **hadoop fs –copyFromLocal** test.txt

- Result:
  - Data is copied to HDFS and distributed over all machines

# Inspect HDFS Folders

- Show root directory of HDFS:
  - **hadoop fs –ls**

- Explanation:
  - ls … list directory

- Result:

```
[root@hadoop-master ~]# hadoop fs -ls
Found 2 items
drwx------   - root hadoop          0 2015-04-01 18:22 .staging
-rw-r--r--   2 root hadoop         15 2015-04-01 18:31 test.txt
```

**readable and writable** **replication factor** **file size in bytes**

# Create Directory

- Command:
  - **hadoop fs –mkdir** output7
- Explanation:
  - mkdir … make directory
- Result:

```
[root@hadoop-master ~]# hadoop fs -mkdir output7
[root@hadoop-master ~]#
[root@hadoop-master ~]# hadoop fs -ls
Found 3 items
drwx------   - root hadoop          0 2015-04-01 18:22 .staging
drwxr-xr-x   - root hadoop          0 2015-04-01 18:36 output7
-rw-r--r--   2 root hadoop         15 2015-04-01 18:31 test.txt
```

**d... directory, rwx ... readable, writeable, executable**

# Show Content of File

- Command:
  - **hadoop fs –cat** test.txt

- Explanation:
  - cat ... concatenate file and print on the standard output

```
[root@hadoop-master ~]# hadoop fs -cat test.txt
Test Kurt 123.
```

# Copy Data from HDFS to Linux

- Command:
  - **hadoop fs -copyToLocal** output/simulation1.txt simulation2015.txt

# Help for Hadoop Commands

- General help:
  - **hadoop**

```
[root@hadoop-master ~]# hadoop
Usage: hadoop [--config confdir] COMMAND
       where COMMAND is one of:
  fs                   run a generic filesystem user client
  version              print the version
  jar <jar>            run a jar file
  checknative [-a|-h]  check native hadoop and compression libraries
  distcp <srcurl> <desturl> copy file or directories recursively
  archive -archiveName NAME -p <parent path> <src>* <dest> create a
  classpath            prints the class path needed to get the
  credential           interact with credential providers
                       Hadoop jar and the required libraries
  daemonlog            get/set the log level for each daemon
 or
  CLASSNAME            run the class named CLASSNAME

Most commands print help when invoked w/o parameters.
```

# More Details

- Command:
  - **hadoop fs**

```
[root@hadoop-master ~]# hadoop fs
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
        [-cat [-ignoreCrc] <src> ...]
        [-checksum <src> ...]
        [-chgrp [-R] GROUP PATH...]
        [-chmod [-R] <MODE[,MODE]... | OCTALMODE> PATH...]
        [-chown [-R] [OWNER][:[GROUP]] PATH...]
        [-copyFromLocal [-f] [-p] <localsrc> ... <dst>]
        [-copyToLocal [-p] [-ignoreCrc] [-crc] <src> ... <localdst>]
```

# Overview of Major Hadoop Commands #1

| Command | Usage | Description |
|---|---|---|
| **ls** | Usage: hadoop fs -ls <args><br><br>**Example**:<br>hadoop fs -ls /user/hive/warehouse | For a file returns information on the file<br><br>For a directory it returns list of its direct children as in unix |
| **lsr** | Usage: hadoop fs -lsr <args><br><br>**Example**:<br>hadoop fs -ls /user | Recursive version of ls. Similar to Unix ls -R |
| **mkdir** | Usage: hadoop fs -mkdir <paths><br><br>**Example**:<br><br>hadoop fs -mkdir /user/ dir1 | Takes path uri's as argument and creates directory |
| **moveFromLocal** | Usage: dfs -moveFromLocal <src> <dst> | Displays a "not implemented" message |
| **mv** | Usage: hadoop fs -mv URI [URI …] <dest><br><br>**Example:** | Moves files from source to destination. This command allows multiple sources as well in which case the destination needs to be a directory. Moving files across filesystems is not permitted. |
| **put** | Usage: hadoop fs -put <localsrc> ... <dst><br><br>**Example:**<br><br>hadoop fs -put localfile1 localfile2 /user/hadoop/hadoopdir | Copy single src, or multiple srcs from local file system to the destination filesystem. Also reads input from stdin and writes to destination filesystem. |

# Overview of Major Hadoop Commands #2

| | | |
|---|---|---|
| **copyToLocal** | Usage: hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst><br><br>**Example:**<br><br>hadoop fs –copyToLocal /temp/file.txt /tmp | Similar to get command, except that the destination is restricted to a local file reference |
| **cp** | Usage: hadoop fs -cp URI [URI …] <dest><br><br>**Example:**<br><br>hadoop fs -cp /user/hadoop/file1 /user/hadoop/file2 /user/hadoop/dir | Copy files from source to destination. This command allows multiple sources as well in which case the destination must be a directory. |
| **cat** | Usage: hadoop fs -cat URI [URI …]<br><br>**Example:**<br><br>hadoop fs -cat hdfs://nn1.example.com/file1 hdfs://nn2.example.com/file2 | Copies source paths to stdout. |
| **rm** | Usage: hadoop fs -rm URI [URI …]<br><br>**Example**:<br><br>hadoop fs -rm hdfs://nn.example.com/file /user/hadoop/emptydir | Delete files specified as args. Only deletes non empty directory and files. Refer to rmr for recursive deletes. |
| **rmr** | Usage: hadoop fs -rmr URI [URI …]<br><br>**Example**:<br><br>hadoop fs -rmr /user/hadoop/dir | Recursive version of delete |
| **du** | Usage: hadoop fs -du URI [URI …]<br><br>**Example**: | Disk usage: Displays aggregate length of files contained in the directory or the length of a file in case its just a file. |

# Parquet Storage Format & Spark

# How Shall We Store Data in a File/Table?

| Title | Released | Label | PeakChart.UK | Certification.BVMI | Certification.RIAA | (omitted for space...) |
|---|---|---|---|---|---|---|
| Led Zeppelin | 01/12/1969 | Atlantic | 6 | | 8x Platinum | ... |
| Led Zeppelin II | 10/22/1969 | Atlantic | 1 | Platinum | Diamond | ... |
| Led Zeppelin III | 10/05/1970 | Atlantic | 1 | Gold | 6x Platinum | ... |
| Led Zeppelin IV | 11/08/1971 | Atlantic | 1 | 3x Gold | Diamond | ... |
| Houses of the Holy | 03/28/1973 | Atlantic | 1 | Gold | Diamond | ... |
| Physical Graffiti | 02/24/1975 | Swan Song | 1 | Gold | Diamond | ... |
| Presence | 03/31/1976 | Swan Song | 1 | | 3x Platinum | ... |
| In Through The Out Door | 08/15/1979 | Swan Song | 1 | | 6x Platinum | ... |
| Coda | 11/19/1982 | Swan Song | 4 | | Platinum | ... |

# Data in Columns on Disk

| Title | Date | Chart |
| --- | --- | --- |
| | | |
| | | |
| | | |

## Row-Oriented data on disk

| | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Led Zeppelin IV | 11/08/1971 | 1 | Houses of the Holy | 03/28/1973 | 1 | Physical Graffiti | 02/24/1975 | 1 |

## Column-Oriented data on disk

| | | | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Led Zeppelin IV | Houses of the Holy | Physical Graffiti | 11/08/1971 | 03/28/1973 | 02/24/1975 | 1 | 1 | 1 |

# Goals for Data Lake Storage

- What is the requirement for a good storage system?

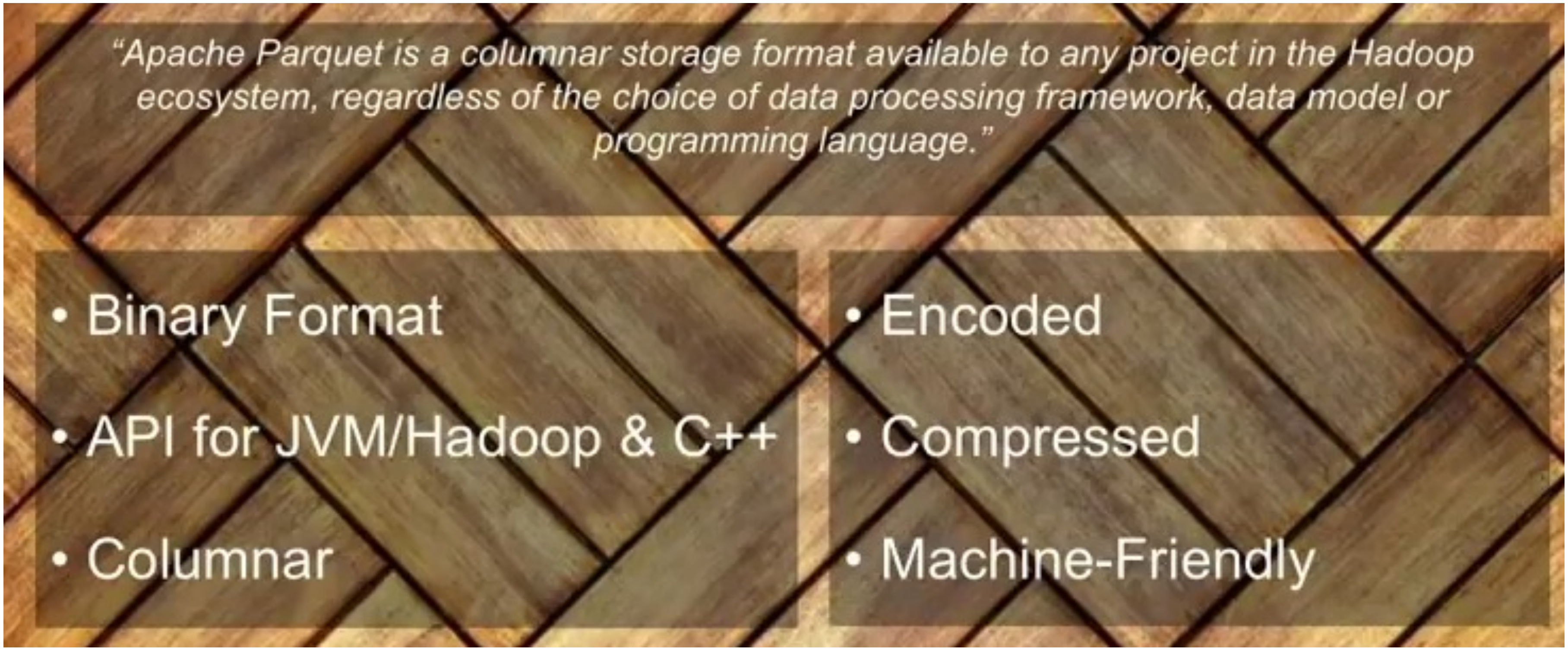
  - Easy to backup
  - Minimal learning curve
  - Easy integration with existing tools (Spark)
  - Resource efficient:
    - Disk space
    - Disk I/O
    - Network I/O

- Overall goal: fast queries

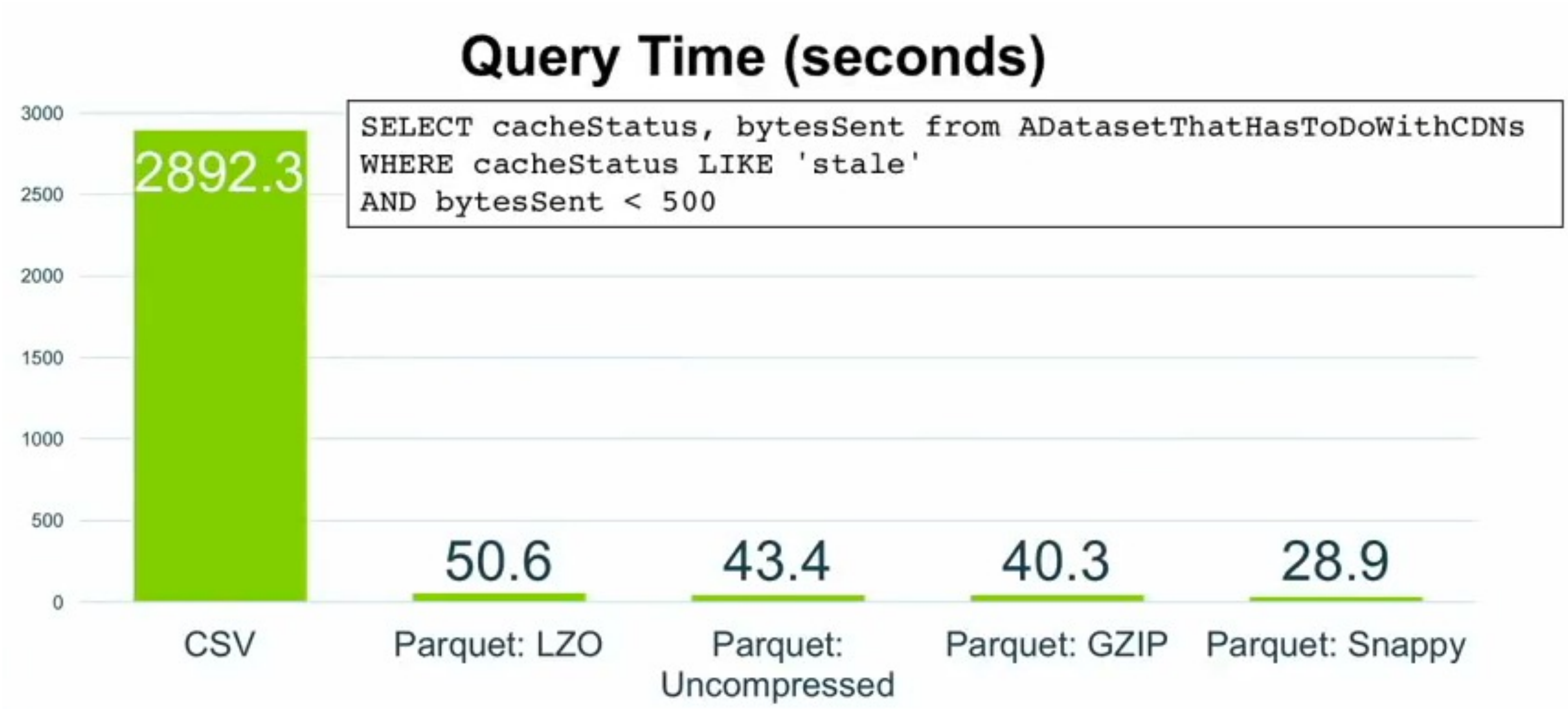
# Options for Multi-PB Data Lake Storage

| | Files | Compressed Files | Databases |
|---|---|---|---|
| Usability | Great! | Great! | OK to BAD (not as easy as a file!) |
| Administration | None! | None! | LOTS |
| Spark Integration | Great! | Great! | Varies |
| Resource Efficiency | BAD (Big storage, heavy I/O) | OK… (Less storage) | BAD (Requires storage AND CPU) |
| Scalability | Good-ish | Good-ish | BAD (For multi-petabyte!) |
| CO$$$$T | OK… | OK… | TERRIBLE |
| QUERY TIME | TERRIBLE | BAD | Good! |

# Parquet Format



# CSV vs. Parquet: SQL Query




**1 master, 3 worker nodes with data on HDFS, file size: 420 GB**

# CSV vs. Parquet: Cost Consideration

| Dataset | Size on Amazon S3 | Query Run time | Data Scanned | Cost |
|---|---|---|---|---|
| Data stored as CSV files | 1 TB | 236 seconds | 1.15 TB | $5.75 |
| Data stored in Apache Parquet format* | **130 GB** | **6.78 seconds** | **2.51 GB** | **$0.01** |
| Savings / Speedup | **87% less with Parquet** | **34x faster** | **99% less data scanned** | **99.7% savings** |

- Costs for just one query every day within a year
  - $2000 for the csv
  - $3.50 for the parquet file
- "unproductive waiting time" for an analyst for just one query every day within a year
  - 30 hours for the csv
  - 42 mins for the parquet file

Source: Andreas Weiler

# How can we Encode/Compress the Data?

| Title | Released | Label | PeakChart.UK | Certification.BVMI | Certification.RIAA | (omitted for space...) |
|---|---|---|---|---|---|---|
| Led Zeppelin | 01/12/1969 | Atlantic | 6 | | 8x Platinum | ... |
| Led Zeppelin II | 10/22/1969 | Atlantic | 1 | Platinum | Diamond | ... |
| Led Zeppelin III | 10/05/1970 | Atlantic | 1 | Gold | 6x Platinum | ... |
| Led Zeppelin IV | 11/08/1971 | Atlantic | 1 | 3x Gold | Diamond | ... |
| Houses of the Holy | 03/28/1973 | Atlantic | 1 | Gold | Diamond | ... |
| Physical Graffiti | 02/24/1975 | Swan Song | 1 | Gold | Diamond | ... |
| Presence | 03/31/1976 | Swan Song | 1 | | 3x Platinum | ... |
| In Through The Out Door | 08/15/1979 | Swan Song | 1 | | 6x Platinum | ... |
| Coda | 11/19/1982 | Swan Song | 4 | | Platinum | ... |

# Encoding: Incremental Encoding

Led_Zeppelin
Led_Zeppelin_II
Led_Zeppelin_III
Led_Zeppelin_IV

58 bytes*

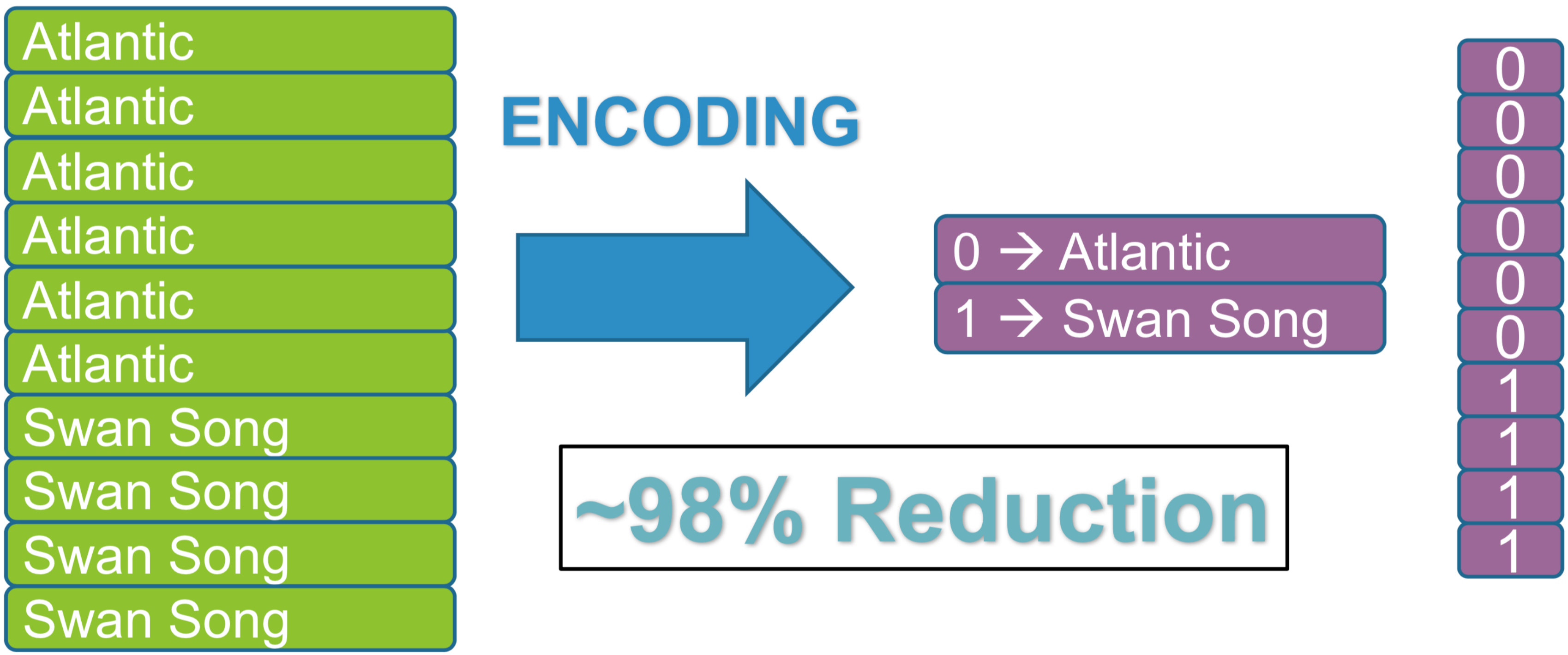
ENCODING

0 Led_Zeppelin
12 _II
15 I
14 V

24 bytes*

58% Reduction

# Encoding: Dictionary Encoding




84 bytes*

1.25 bytes + dictionary size

# Parquet File: Writing and Reading

```
# Read data from file (tab-separated)
customer = sqlContext.read.format("com.databricks.spark.csv")\
           .option("header","true")\
           .option("delimiter", "|")\
           .option("inferSchema", "true")\
           .load("/FileStore/tables/uqrziax61502358733901/customer.tbl")

# Write as parquet file
customer.write.parquet("customerPq")

# Read from parquet file
customerDF = spark.read.parquet("customerPq")
```

# More Encoding Schemes

- Plain (bit-packed, little endian)
- Dictionary-encoding
- Run length encoding
- Etc.

# Reading vs. Writing

- Writing a Parquet file is typically significantly slower than writing a CSV file:
  - More operations required (encoding)
- Reading a Parquet file is typically significantly faster than reading a CSV file:
  - Only specific columns need to be read
  - Columns are compressed
- However, data is often written once and read many times
- Hence, benefits of fast reads often outweigh performance penalty of slow writes

Zurich University
of Applied Sciences

# Conclusions

- HDFS distributes blocks onto machines in a round robin fashion
- Replication guarantees fault tolerance
- Using Parquet can be more performant than CSV due to encoding and compression
- However, Parquet format might also be slower:
  - e.g. if data is not “compressible” (i.e. no repetition in data)