Apache Hadoop Ecosystem

Other available tools

MUNI
ÚVT

hadoop – inspired by Google

- Apache Hadoop project
  - inspired by Google's MapReduce and Google File System papers.
- **Open sourced**, **flexible** and **available** architecture for **large scale** computation and data processing on a network of **commodity hardware**
- Open Source Software + Hardware Commodity
  - IT Costs Reduction

MUNI ÚVT

Hadoop Core

HDFS

- Hadoop Distributed File System
- Redundancy
- Fault Tolerant
- Scalable
- Self Healing
- Write Once, Read Many Times
- Java API
- Command Line Tool

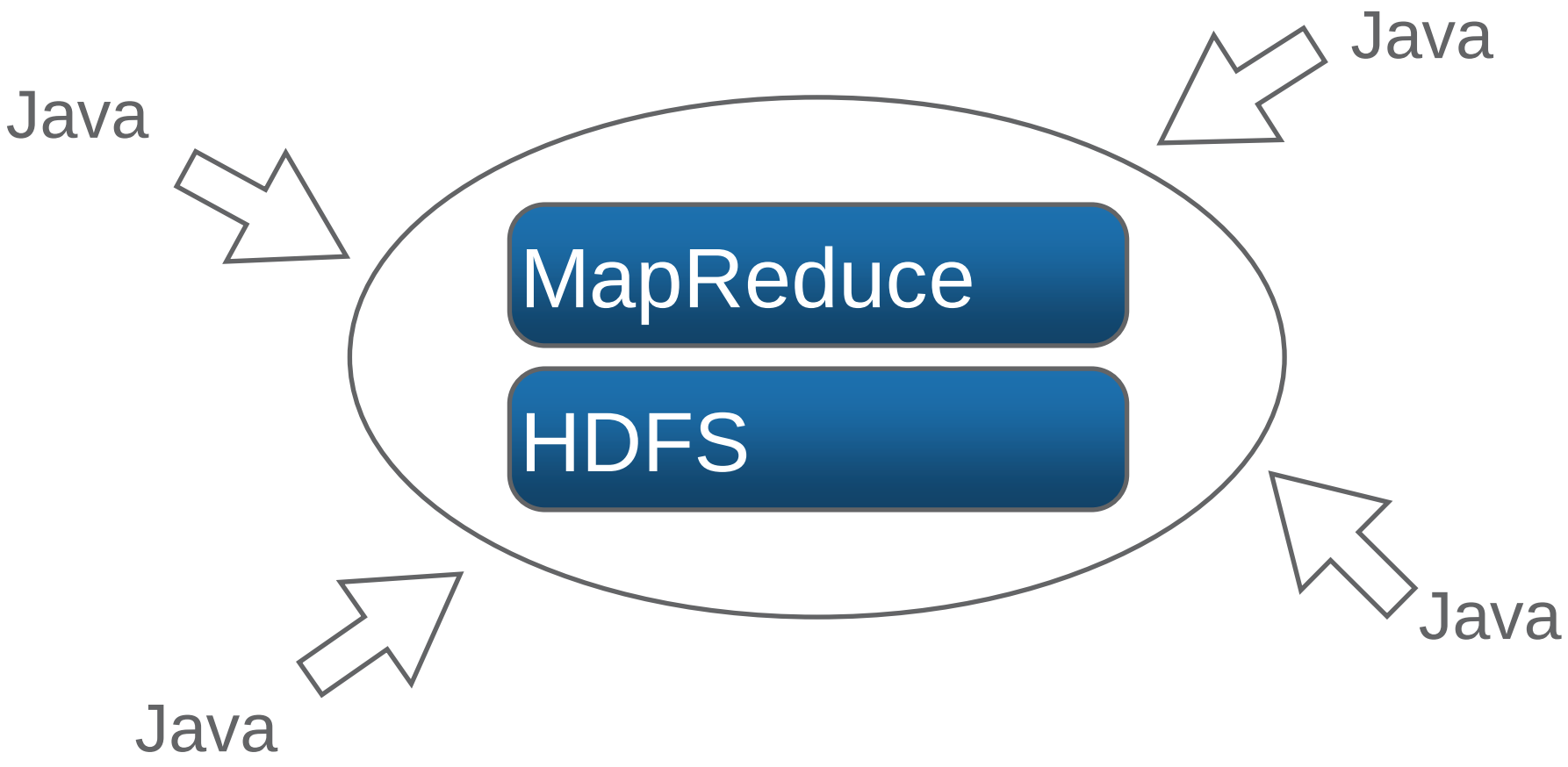
MUNI
ÚVT

MapReduce

- Two Phases of Functional Programming
- Redundancy
- Fault Tolerant
- Scalable
- Self Healing
- Java API

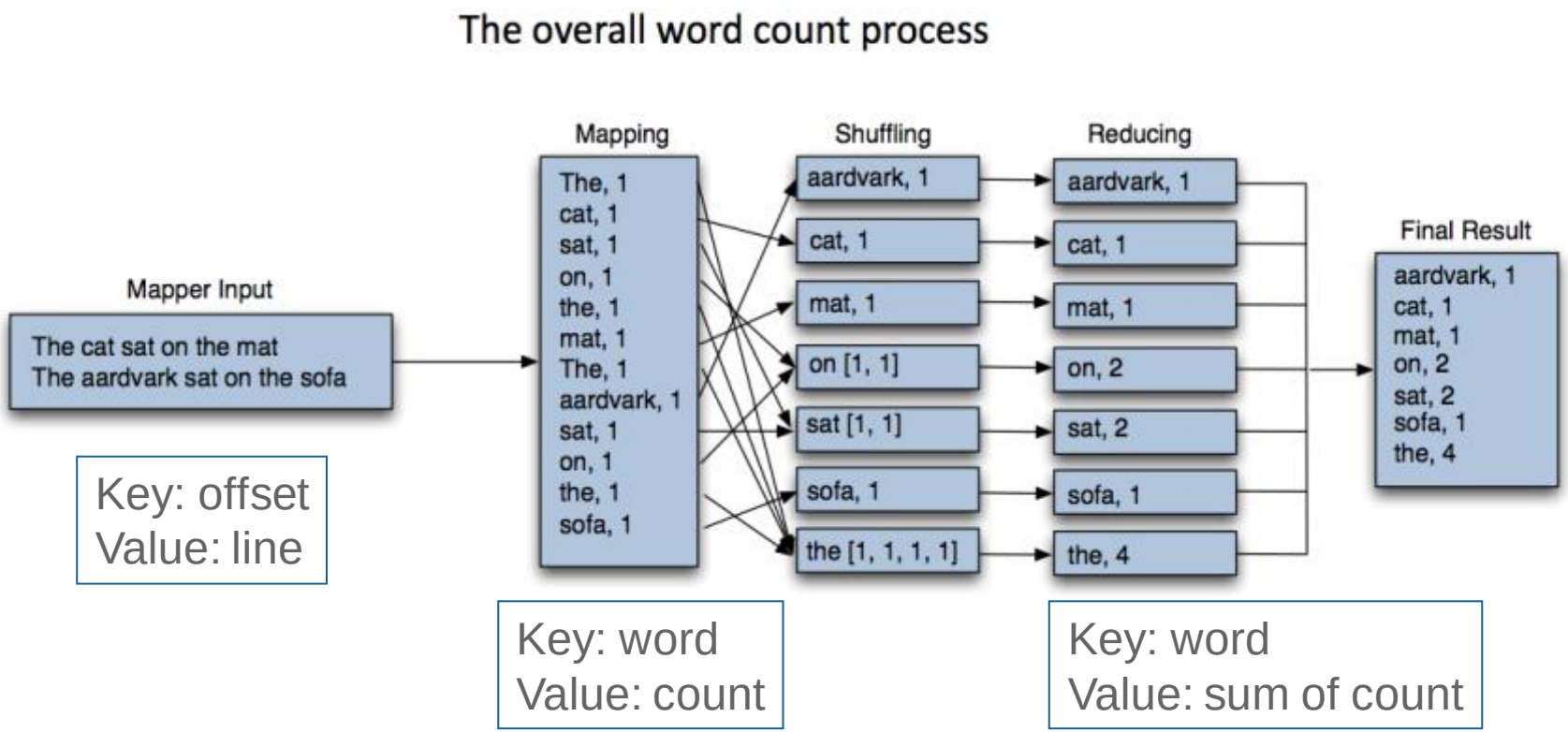
MUNI
ÚVT

Hadoop Core



MUNI ÚVT

Word Count Example



MUNI ÚVT

Apache Hadoop Ecosystem

Why do these tools exist?

- MapReduce is very powerful, but can be awkward to master

- These tools allow programmers who are familiar with other programming styles to take advantage of the power of MapReduce

MUNI
ÚVT

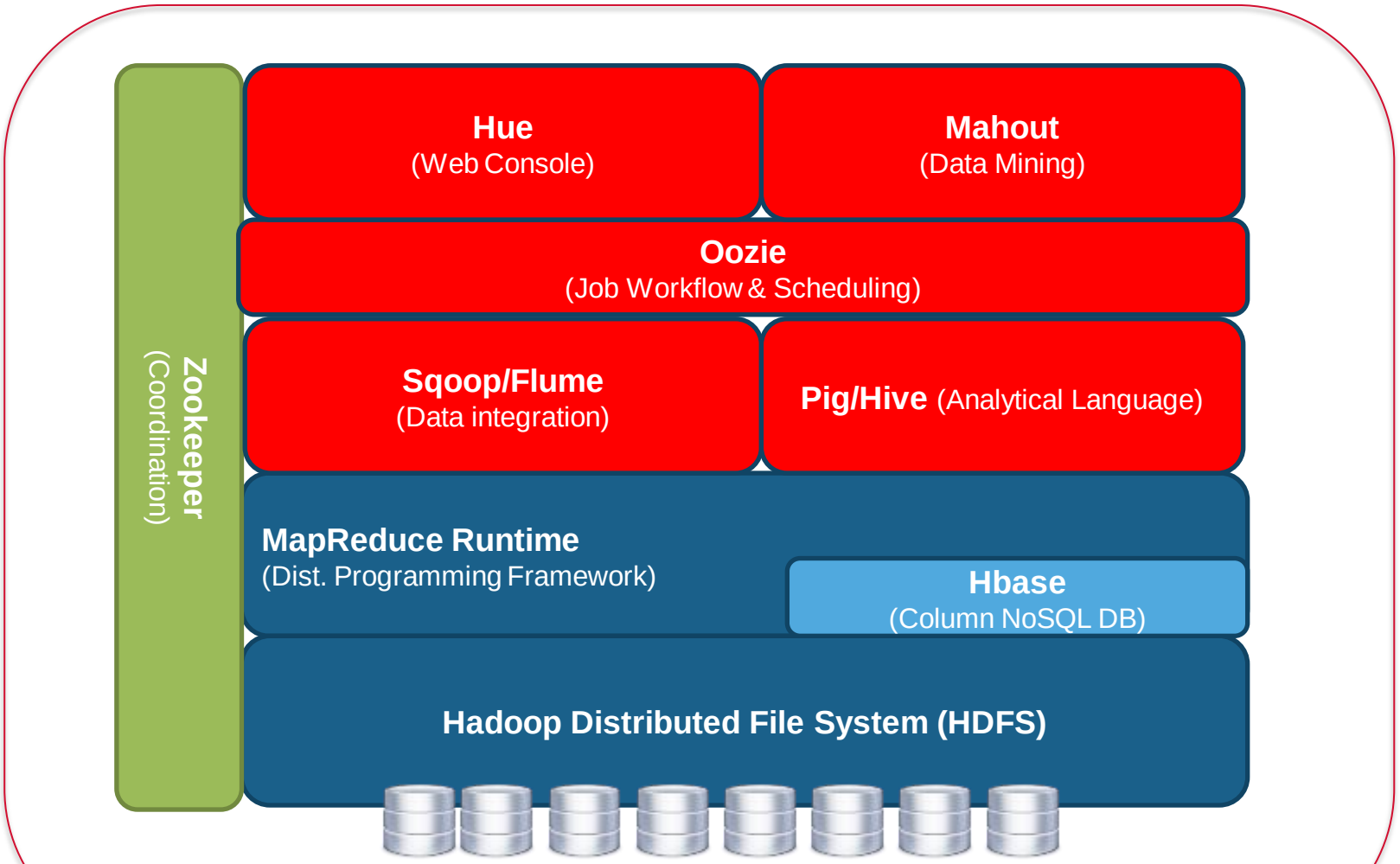
Apache Hadoop Ecosystem

The Ecosystem is the System

- Hadoop has become the kernel of the distributed operating system for Big Data
- No one uses the kernel alone
- **Hadoop Ecosystem** = a collection of projects at Apache

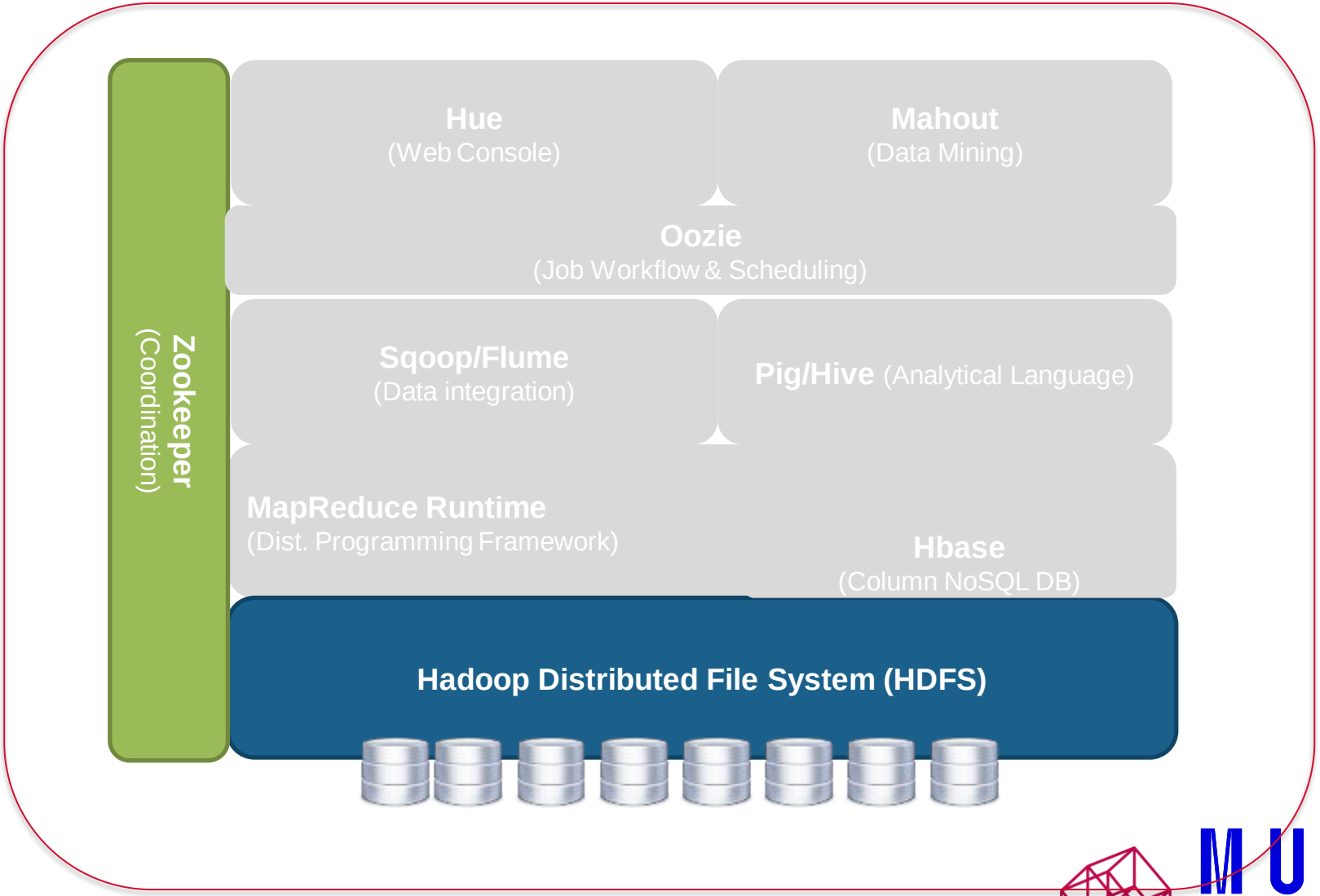
MUNI ÚVT

Apache Hadoop Ecosystem



MUNI
ÚVT

Zookeeper – Coordination Framework



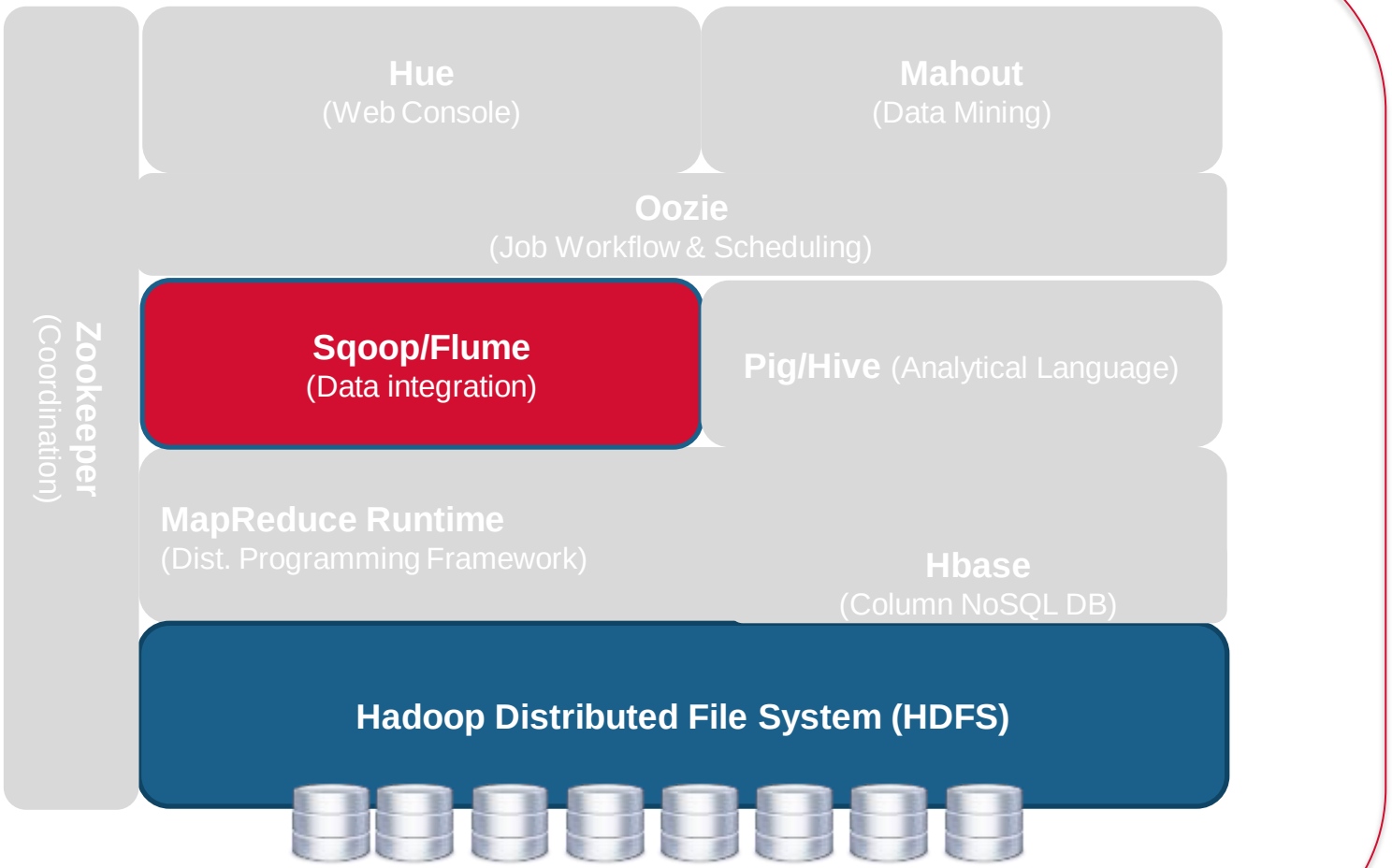
MUNI
ÚVT

What is ZooKeeper?

- A centralized service for maintaining
  - Configuration information
  - Naming
  - Providing distributed synchronization
  - etc.
- A **set of tools to build distributed applications** that can safely handle partial failures
  - They don‘t need to implement them on their own
- ZooKeeper was designed to store coordination data
  - Status information
  - Configuration
  - Location information

MUNI ÚVT

Flume / Sqoop – Data Integration Framework



MUNI
ÚVT

What's the problem for data collection?

- Collecting data (to analyze) from distributed systems requires many mechanisms to provide reliability, security, …
  - These don't need to be implemented by each application

- **Apache Flume**
  - a system used for moving massive quantities of streaming data into HDFS
    - eg., collecting log data present in log files from web servers and aggregating it in HDFS for analysis

MUNI
ÚVT

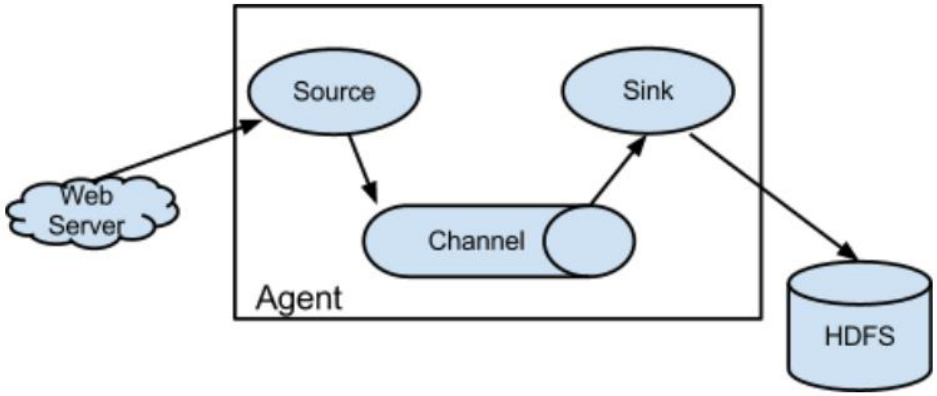
How can Flume help?

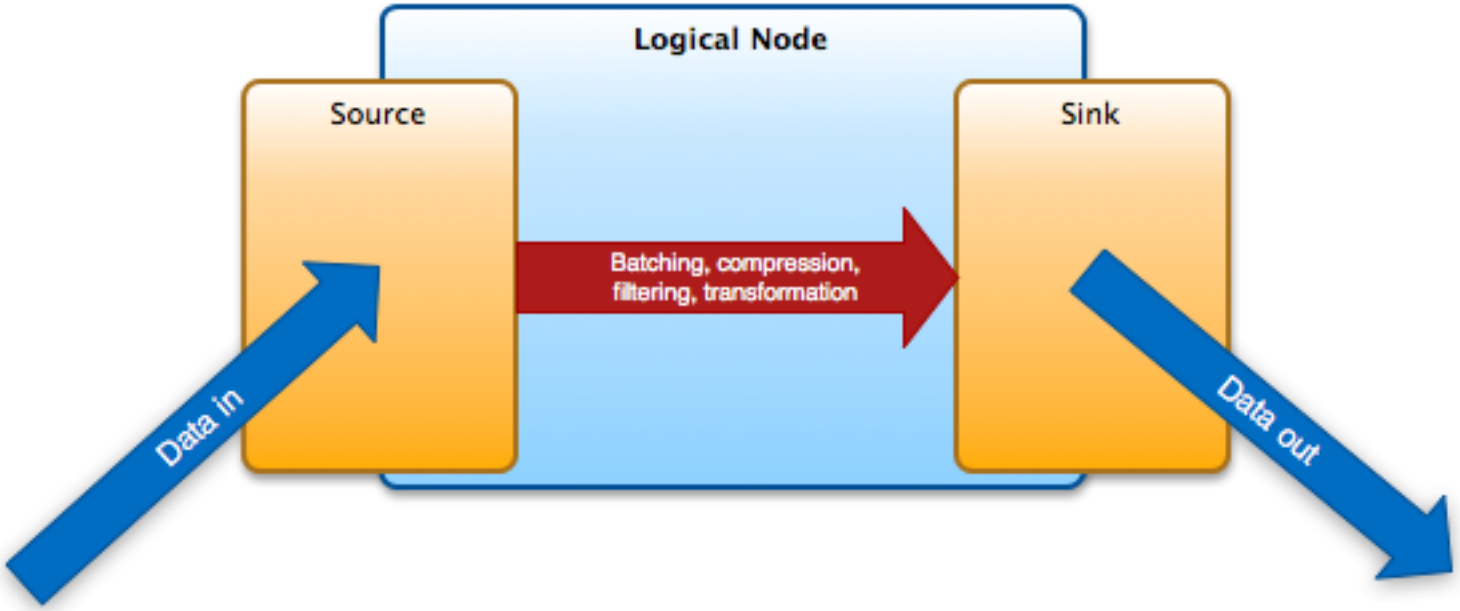
- A distributed data collection service
- It‘s efficiently collecting, aggregating, and moving large amounts of data
- Supports data encryption (SSL/TLS)
- Fault tolerant, many failover and recovery mechanisms
- One-stop solution for data collection of all formats

MUNI ÚVT

Flume: High-Level Overview

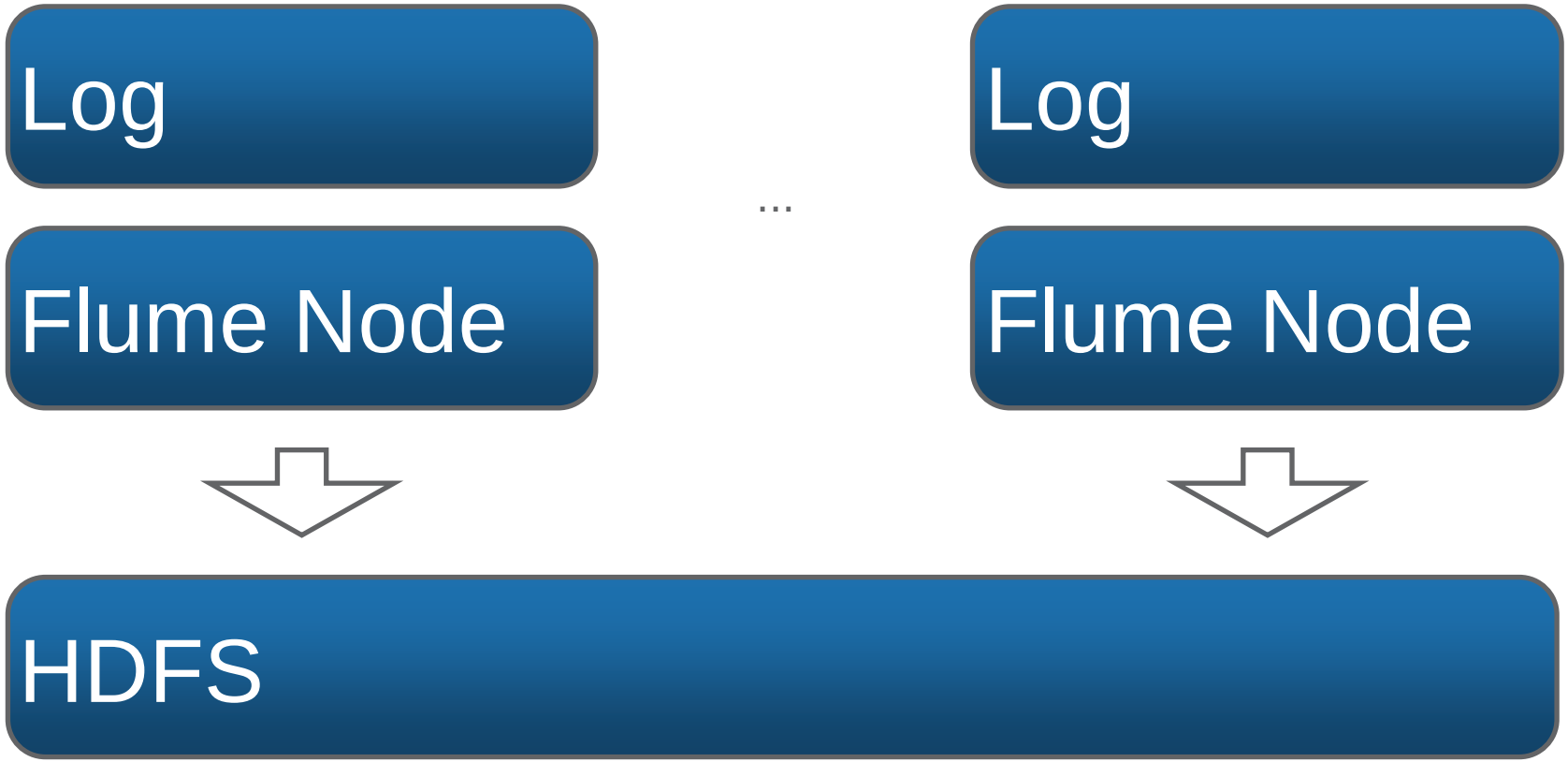
- **Logical Node**
- **Source**
- **Sink**
- **Channel** (passive store)





MUNI ÚVT

Flume Architecture



MUNI ÚVT

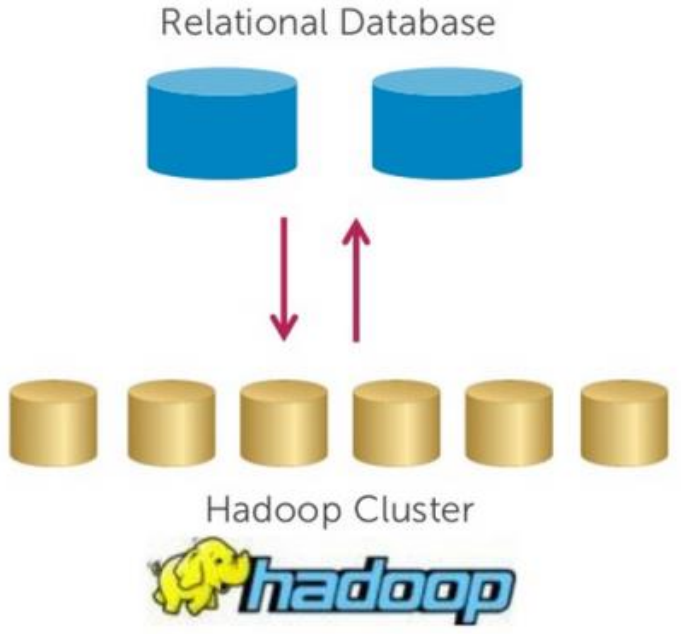
Flume Sources and Sinks

- Local Files
- HDFS
- Stdin, Stdout
- Twitter
- IRC
- IMAP
- NetCat UDP source
- Syslog
- Custom sources
- …

MUNI
ÚVT

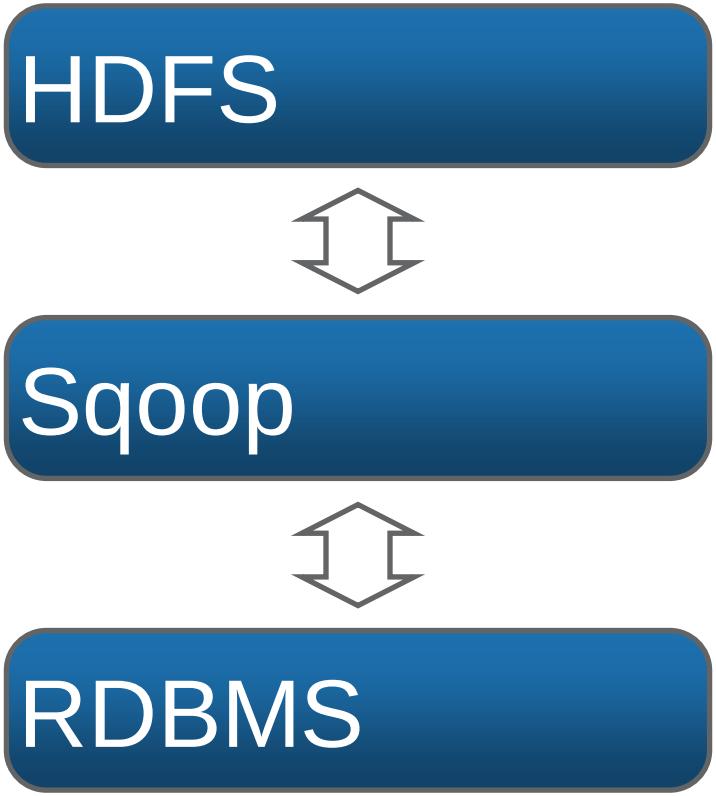
Sqoop

- **Apache Sqoop** (= Sql+Hadoop)
  - Tool for efficient transferring bulk data between Apache Hadoop and structured datastores such as relational databases
  - Easy, parallel database import/export
  - What you want to do?
    - Insert data from RDBMS to HDFS
    - Export data from HDFS back into RDBMS



MUNI ÚVT

Sqoop



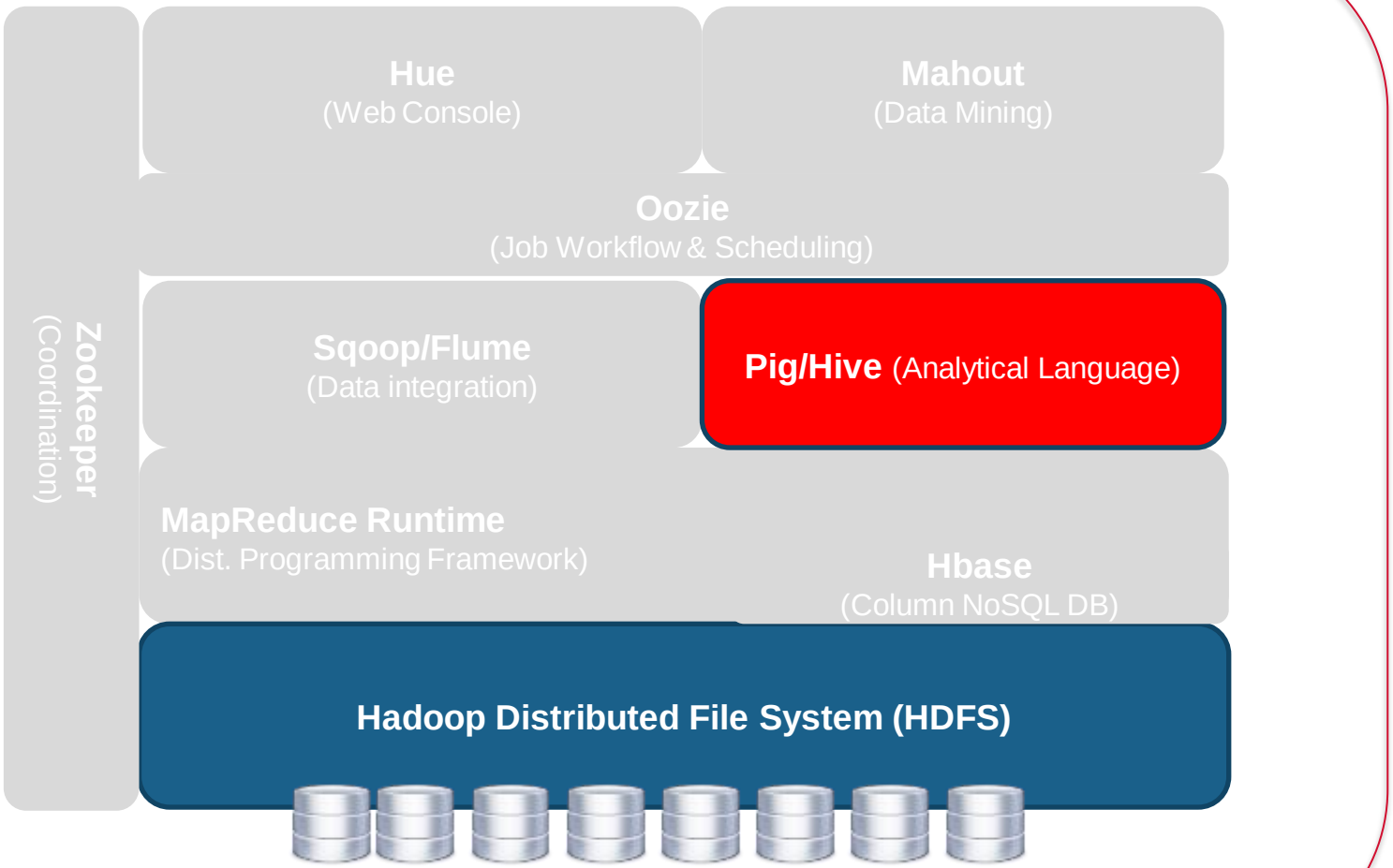
Sqoop Examples

```
$ sqoop import --connect jdbc:mysql://localhost/world --
username root --table City
...

$ hadoop fs -cat City/part-m-00000
1,Kabul,AFG,Kabol,17800002,Qandahar,AFG,Qandahar,2375003,He
rat,AFG,Herat,1868004,Mazar-e-
Sharif,AFG,Balkh,1278005,Amsterdam,NLD,Noord-Holland,731200
...
```

MUNI
ÚVT

Pig / Hive – Analytical Language



MUNI
ÚVT

Why Hive and Pig?

- Although MapReduce is very powerful, it can also be complex to master
- Many organizations have business or data analysts who are skilled at writing SQL queries, but not at writing Java code
- Many organizations have programmers who are skilled at writing code in scripting languages
- **Hive** and **Pig** are two projects which evolved separately to help such people analyze huge amounts of data via MapReduce
  - Hive was initially developed at Facebook, Pig at Yahoo!

MUNI
ÚVT

– Developed by facebook

Apache Hive

- An SQL-like interface to Hadoop

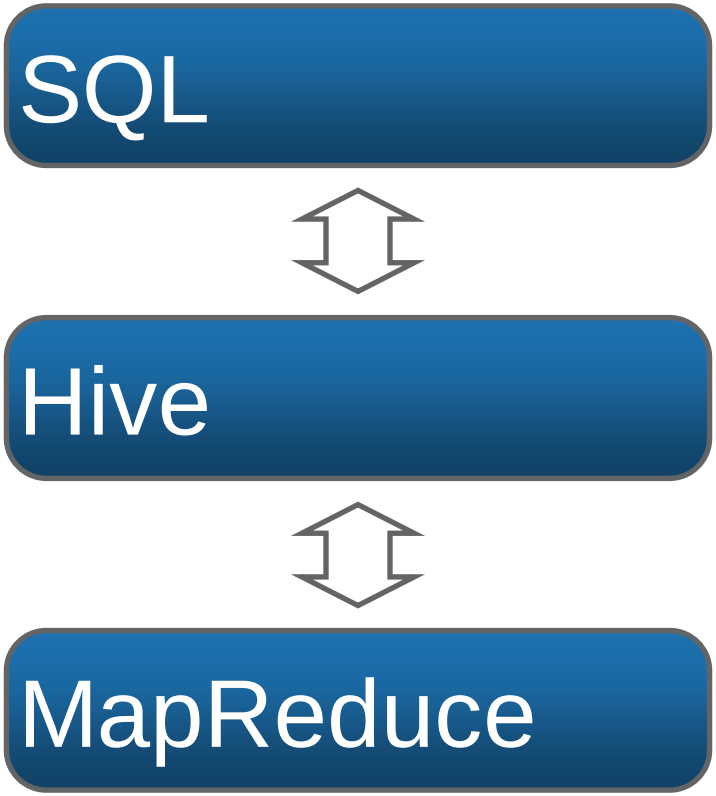
- Data Warehouse infrastructure that provides data summarization and ad hoc querying on top of Hadoop
  - MapRuduce for execution
  - HDFS for storage

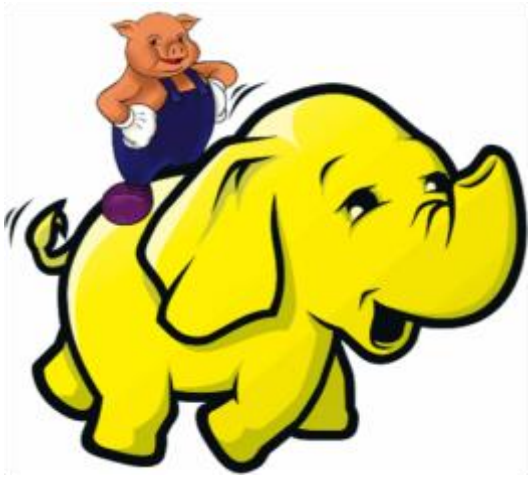
- Hive Query Language
  - Basic-SQL : Select, From, Join, Group-By
  - Equi-Join, Muti-Table Insert, Multi-Group-By
  - Batch query

```
SELECT * FROM purchases WHERE price > 100 GROUP BY storeid
```

MUNI
ÚVT

Hive





– Initiated by YAHOO!

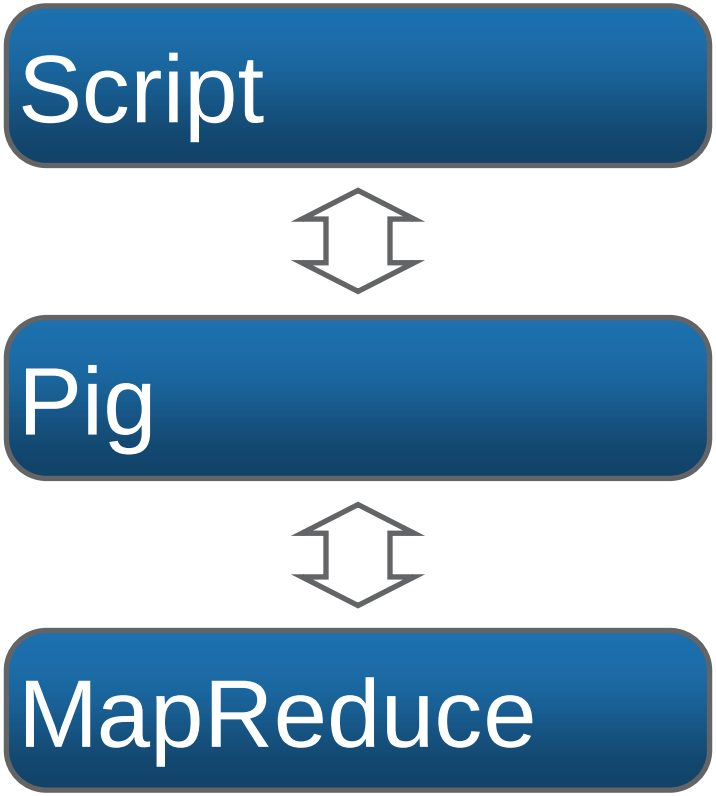
```
A = load 'a.txt' as (id, name, age, ...)
B = load 'b.txt' as (id, address, ...)
C = JOIN A BY id, B BY id;STORE C into 'c.txt'
```

Apache Pig

- A high-level scripting language (Pig Latin)
- Allows to simply write MapReduce programs
  - their structure is amenable to substantial parallelization
    - allow for large data processing
  - optimizes automatically
    - allows the user to focus on semantics rather than efficiency
- Crucial features
  - easy to understand, easy to debug
  - extendable (new functions)
  - allows for high optimilization

MUNI ÚVT

Pig



Hive vs. Pig

	Hive	Pig
Language	HiveQL (SQL-like)	Pig Latin, a scripting language
Schema	Table definitions that are stored in a metastore	A schema is optionally defined at runtime
Programmait Access	JDBC, ODBC	PigServer

MUNI
ÚVT

WordCount Example

- Input

```
Hello World Bye World
Hello Hadoop Goodbye Hadoop
```

- For the given sample input the map emits

```
< Hello, 1>
< World, 1>
< Bye, 1>
< World, 1>
< Hello, 1>
< Hadoop, 1>
< Goodbye, 1>
< Hadoop, 1>
```

-

```
< Bye, 1>
< Goodbye, 1>
< Hadoop, 2>
< Hello, 2>
< World, 2>
```

MUNI
ÚVT

WordCount Example In MapReduce

```
public class WordCount {
 public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
      String line = value.toString();
      StringTokenizer tokenizer = new StringTokenizer(line);
      while (tokenizer.hasMoreTokens()) {
         word.set(tokenizer.nextToken());
         context.write(word, one);
      }
   }
 }

 public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {
   public void reduce(Text key, Iterable<IntWritable> values, Context context)
     throws IOException, InterruptedException {
      int sum = 0;
      for (IntWritable val : values) {
         sum += val.get();
      }
      context.write(key, new IntWritable(sum));
   }
 }

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

   Job job = new Job(conf, "wordcount");
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(IntWritable.class);

   job.setMapperClass(Map.class);
   job.setReducerClass(Reduce.class);

   job.setInputFormatClass(TextInputFormat.class);
   job.setOutputFormatClass(TextOutputFormat.class);

   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));

   job.waitForCompletion(true);
 }
}
```

WordCount Example By Pig

```
A = LOAD 'wordcount/input' USING PigStorage as (token:chararray);

B = GROUP A BY token;

C = FOREACH B GENERATE group, COUNT(A) as count;

DUMP C;
```

MUNI
ÚVT

WordCount Example By Hive

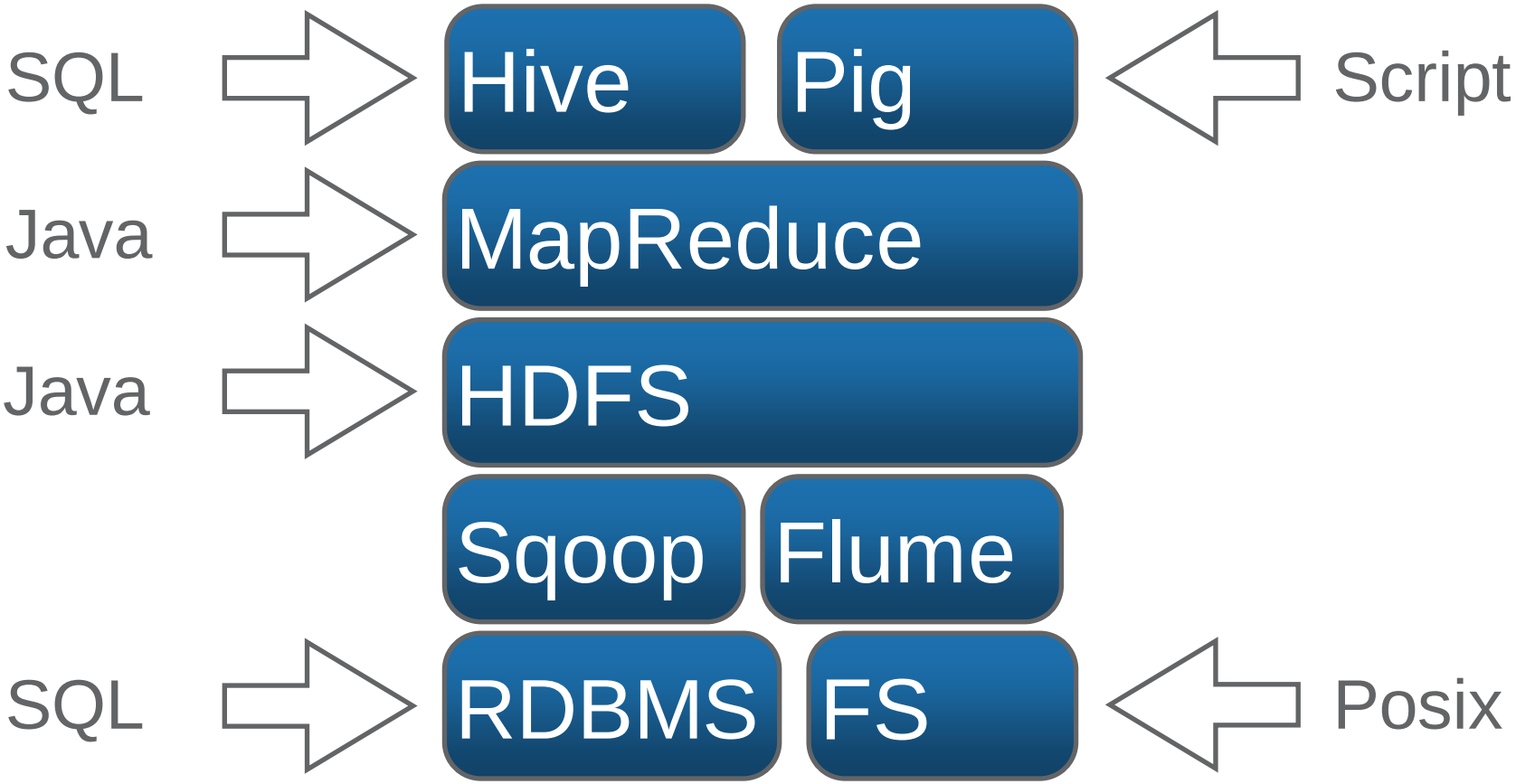
```
CREATE TABLE wordcount (token STRING);

LOAD DATA LOCAL INPATH ’wordcount/input'
OVERWRITE INTO TABLE wordcount;


SELECT count(*) FROM wordcount GROUP BY token;
```

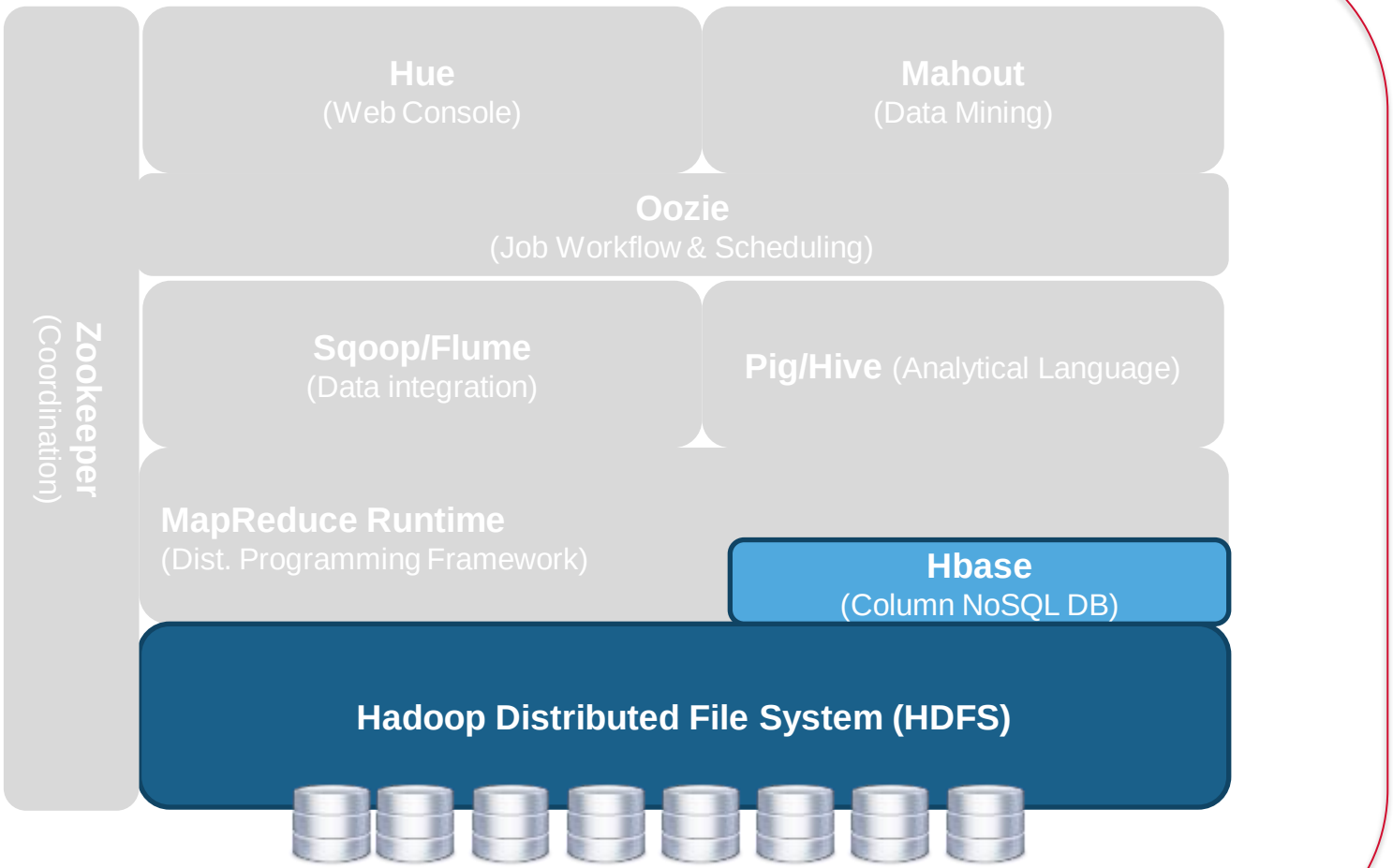
MUNI
ÚVT

The Story So Far …



MUNI
ÚVT

Hbase – Column NoSQL DB



MUNI
ÚVT

Structured-data vs Raw-data

MUNI
ÚVT

APACHE HBASE – Inspired by Google

Apache HBase

- Open-source Apache project
- Non-relational, distributed Database
- Runs on top of HDFS
- Modeled after Google's BigTable technology
- Written in Java
- NoSQL (Not Only SQL) Database
- Consistent and Partition tolerant
- Runs on commodity hardware
- Able to host Very Large Databases (terabytes to petabytes)
  - billions of rows & millions of columns atop clusters of commodity hardware
- Low latency random read / write to HDFS
- Many companies are using HBase
  - Facebook, Twitter, Adobe, Mozilla, Yahoo!, Trend Micro, and StumbleUpon

MUNI ÚVT

APACHE HBASE – Inspired by Google

Apache HBase is NOT

- A direct replacement for RDBMS
- ACID (Atomicity, Consistency, Isolation, and Durability) complaint
  - HBase provides row-level atomicity
  - a scan is NOT consistent view of a table (neither isolated)
  - all visible data is also durable data

MUNI ÚVT

Relational Database vs HBase

- Hardware
  - Expensive Enterprise multiprocessor systems
  - Same as Hadoop
- Fault Tolerance
  - RDBMS are configured with high availability. Server down time intolerable.
  - Built into the architecture. Individual Node failure does not impact overall performance.
- Database Size
  - RDBMS can hold upto TBs (Tera bytes)
  - Hbase can hold PBs (Peta bytes)
- Data Layout
  - RDBMS are rows and columns oriented
  - Hbase is Column oriented

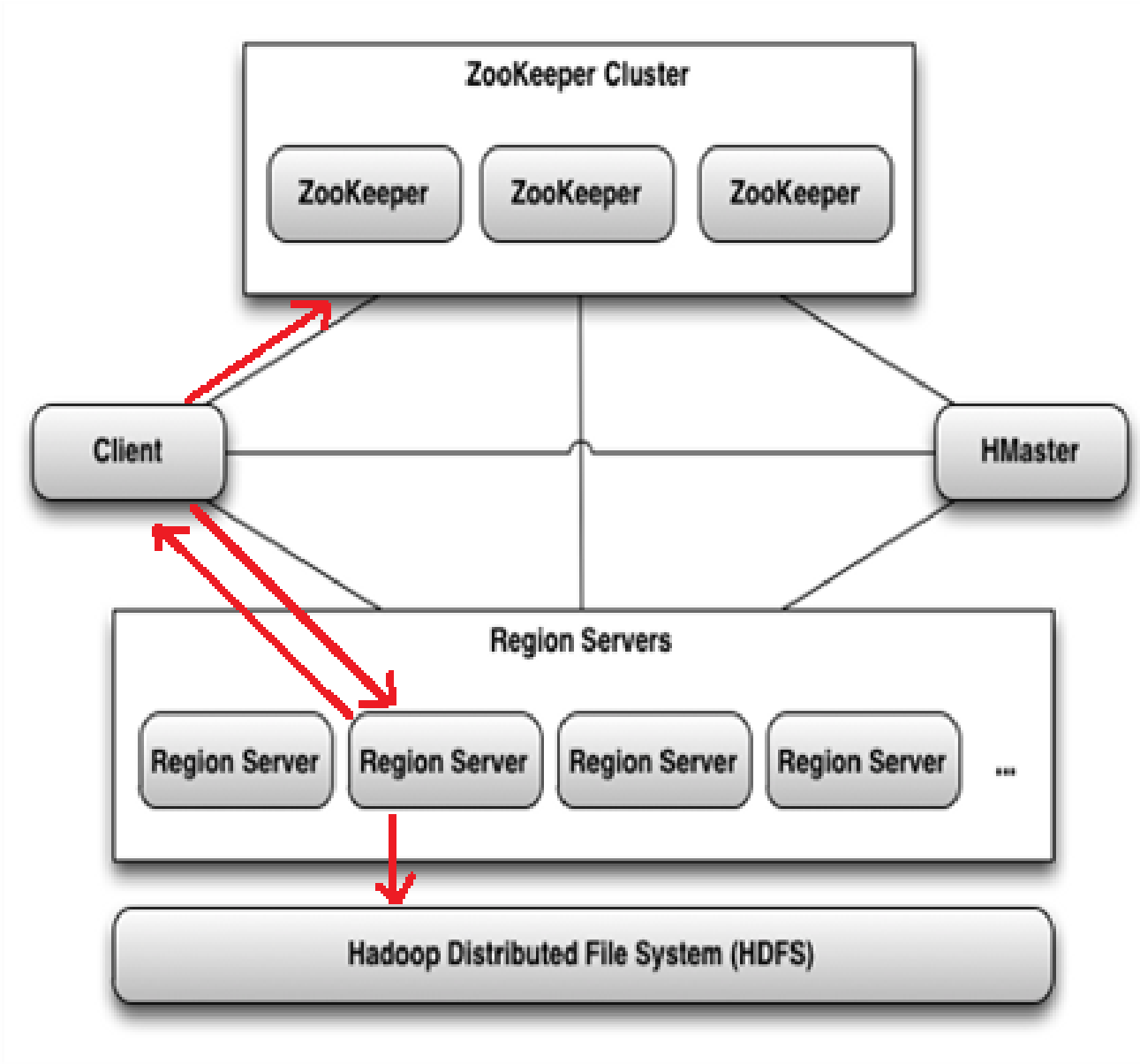
MUNI
ÚVT

Relational Database vs HBase

- Data Type
  - Rich data type.
  - Bytes
- Transactions
  - Fully ACID complaint.
  - ACID on single row only.
- Indexes
  - PK, FK and other indexes.
  - Sorted Row-key (not a real index)

MUNI
ÚVT

HBase – workflow



HBase – Fault Tolerance

- **What if region server dies?**
  - The HBase master will assign a new regionserver
- **What if master dies?**
  - The back up master will take over
- **What if the backup master dies?**
  - You are dead
- **Replication of Data**
  - HBase achieves this using HDFS replication mechanism
- **Failure Detection**
  - Zookeeper is used for identifying failed region servers

MUNI
ÚVT

HBase – Data Model

- No Schema
- Table
  - Row-key must be unique
  - Rows are formed by one or more columns
  - Columns are grouped into Column Families
  - Column Families must be defined at table creation time
  - Any number of Columns per column family
  - Columns can be added on the fly
  - Columns can be NULL
    - NULL columns are NOT stored (free of cost)
    - Column only exist when inserted (Sparse)
- Cell
  - Row Key, Column Family, Qualifier , Timestamp / Version
- Data represented in byte array
  - Table name, Column Family name, Column name

MUNI
ÚVT

HBase – Data Model

- Cells are “versioned”
- Table rows are sorted by row key
- Region – a row range [start-key:end-key]

Table

Row	Timestamp	Animal		Repair
		Type	Size	Cost
Enclosure1	12	Zebra	Medium	1000€
	11	Lion	Big	
Enclosure2	13	Monkey	Small	1500€

Region

Key

Column

Family

Cell

(Table, Row_Key, Family, Column, Timestamp) = Cell (Value)

MUNI
ÚVT

HBase – Logical View of Data

RDBMS View

ID (pk)	First Name	Last Name	tweet	Timestamp
1234	John	Smith	hello	20130710
5678	Joe	Brown	xyz	20120825
5678	Joe	Brown	zzz	20130916

Logical Hbase View

Row key	Value (Column Family, Qualifier, Version)
1234	Info{‘lastName’: ‘Smith’, ‘firstName’:’John’} pwd{‘tweet’:’hello’ @ts 20130710}
5678	Info{‘lastName’: ‘Brown’, ‘firstName’:’Joe’} pwd{‘tweet’:’xyz’ @ts 20120825, ‘tweet’:’zzz’ @ts 20130916}

HBase – Physical View of Data

Info column family

Row key	Column Family:Column	Timestamp	Value
1234	info:firstName	12345678	John
1234	Info:lastName	12345678	Smith
5678	Info:firstName	12345679	Joe
5678	Info:lastName	12345679	Brown

tweet column family

Row key	Column Family:Column	Timestamp	Value
1234	tweet:msg	12345678	Hello
5678	tweet:msg	12345679	xyz
5678	tweet:msg	12345999	zzz

KEY (ROW KEY, CF, QUALIFIER, TIMESTAMP) => VALUE

MUNI ÚVT

Hbase – Logical to Physical View

	CF1				CF2		
Row	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**
ROW1	V1		V3			V6	
ROW2	V4	V6		V7			
ROW3			V6			V5	
ROW4	V10		V11			V2	

Physical View

HFile for CF1

ROW1:CF1:C1:V1
ROW1:CF1:C3:V3
ROW2:CF1:C1:V4
ROW2:CF1:C2:V6
ROW2:CF1:C4:V7
ROW3:CF1:C3:V6
ROW4:CF1:C1:V10
ROW4:CF1:C3:V11

HFile for CF2

ROW1:CF2:C6:V6
ROW3:CF2:C6:V5
ROW4:CF2:C6:V2

MUNI
ÚVT

HBase DB Design Considerations

- **Row Key design**
  - To Leverage HBase system, row-key design is very important
  - Row Key must be designed based on how you access data
  - Salting rowkey (prefix)
  - Must be designed to make sure data are uniformly distributed
    - avoid hotspotting
- **Column Family design**
  - Designed based on grouping of like information (user base info, user tweets)
  - Short name for column family (every row in Hfile contains the name, in bytes)
  - Two to three column families per Table

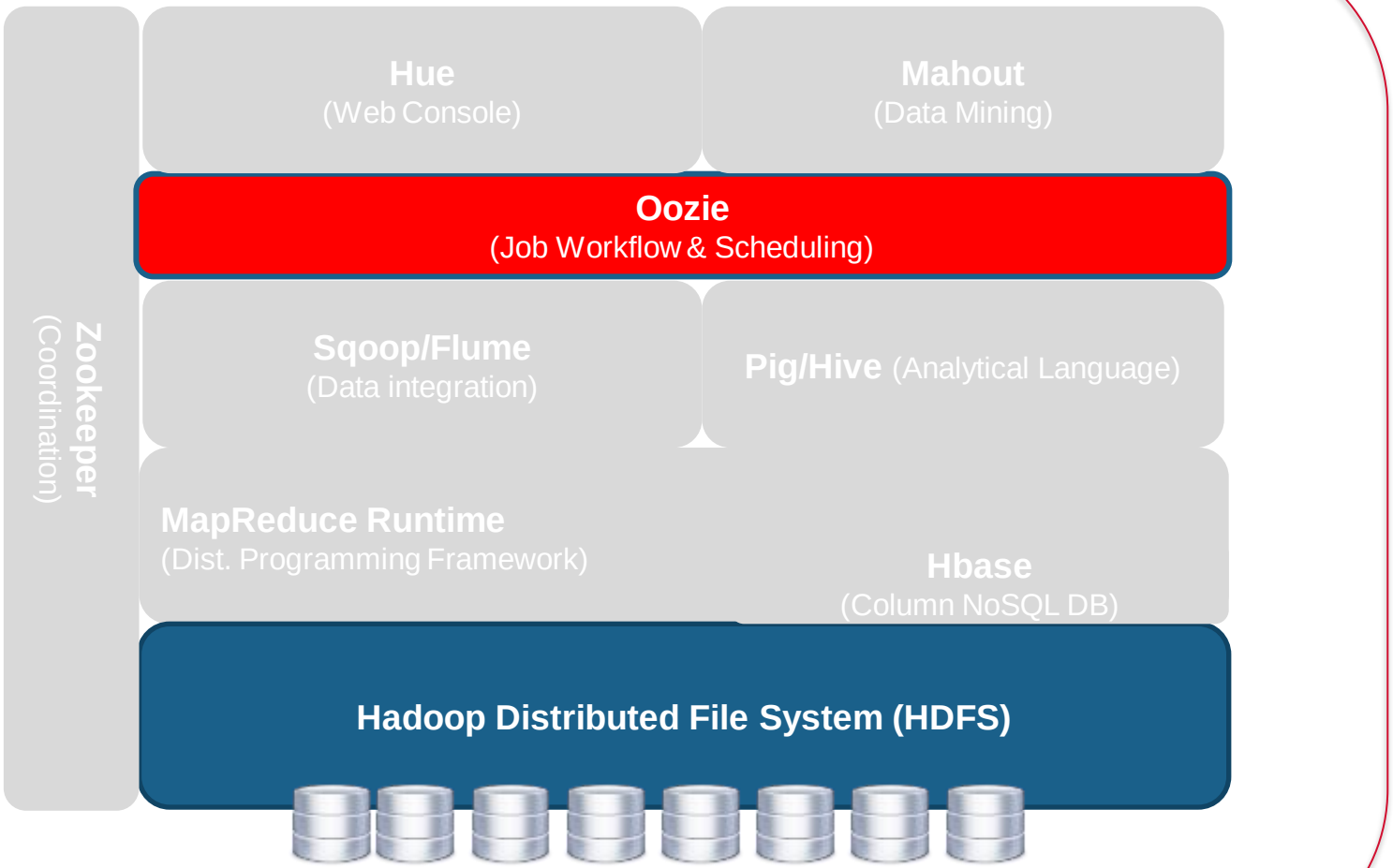
MUNI
ÚVT

HBase Examples

```
hbase> create 'mytable', 'mycf‘
hbase> list
hbase> put 'mytable', 'row1', 'mycf:col1', 'val1‘
hbase> put 'mytable', 'row1', 'mycf:col2', 'val2‘
hbase> put 'mytable', 'row2', 'mycf:col1', 'val3‘
hbase> scan 'mytable‘
hbase> disable 'mytable‘
hbase> drop 'mytable'
```

MUNI
ÚVT

Oozie – Job Workflow & Scheduling

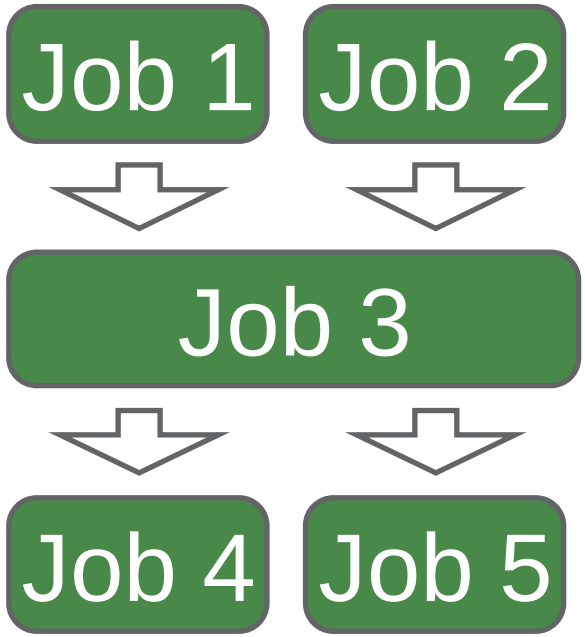


MUNI ÚVT

What is OOZIE™?

Apache Oozie

- A Java Web Application
  - scalable, reliable and extensible system
- Oozie is a workflow scheduler for Hadoop
  - ie., Crond for Hadoop
- Workflow jobs are Directed Acyclical Graphs (DAGs) of actions
- Triggered
  - Time (frequency)
  - Data (availability)



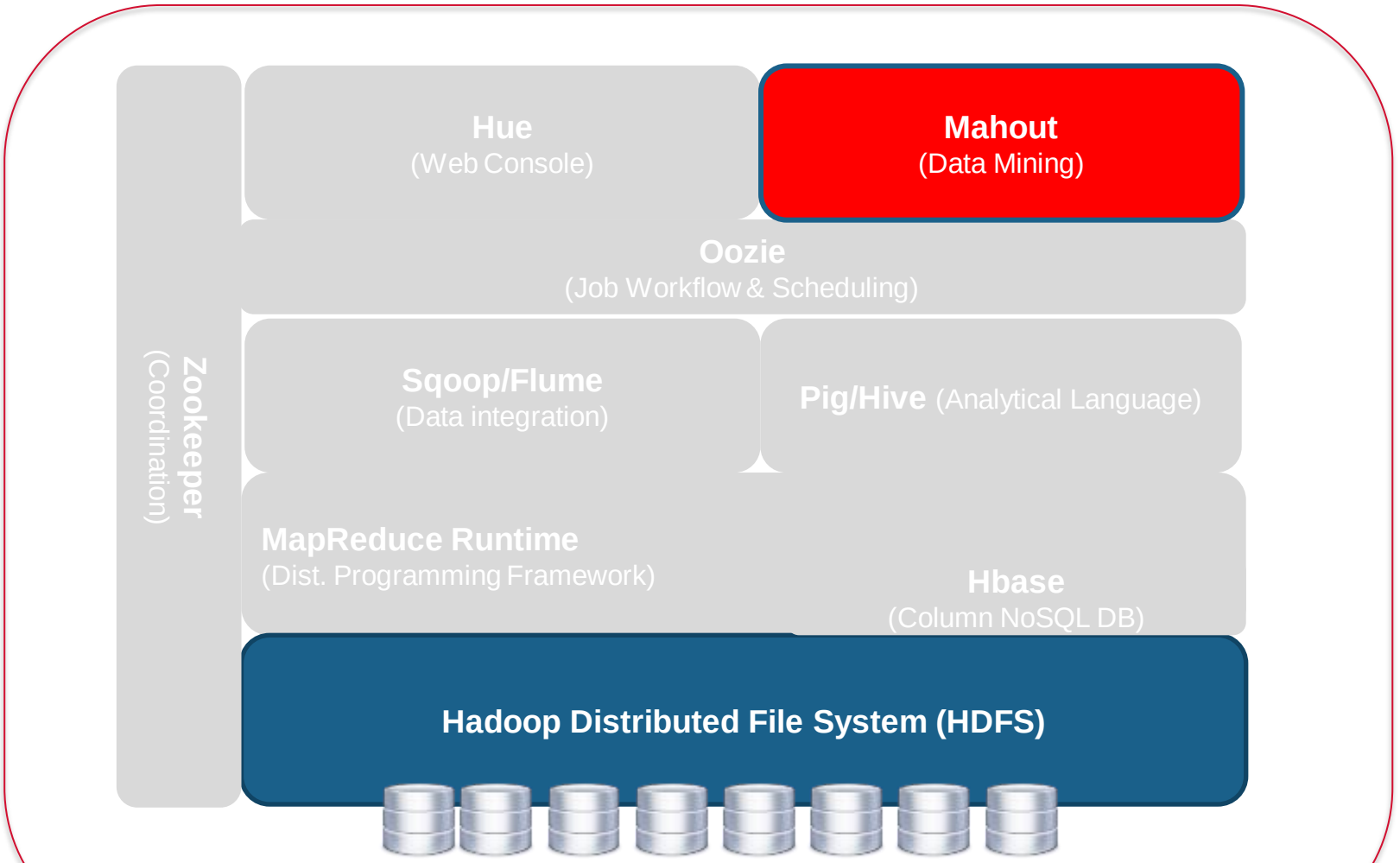
MUNI ÚVT

Oozie Features

- Execute and monitor workflows in Hadoop
- Periodic scheduling of workflows
- Trigger execution of data availability
- HTTP and command line interface and web console
- Component Independent
  - MapReduce
  - Hive
  - Pig
  - SqoopStreaming
  - HDFS
  - sub-workflow
  - Java (custom Java code)

MUNI
ÚVT

Mahout – Data Mining



MUNI
ÚVT

What is 

Apache Mahout

- Machine-learning tool
- Implements **distributed and scalable machine learning techniques/algorithms** on the Hadoop platform
  - Recommendation
  - Classification
  - Clustering
- Allows to build intelligent applications easier and faster

MUNI
ÚVT

What is 

Apache Mahout Features

- Algorithms are written on top of Hadoop
  - works well in distributed environment
- Offers a ready-to-use framework for doing data mining tasks on large volumes of data
- Lets applications to analyze large sets of data effectively and in quick time
- Includes several MapReduce enabled clustering implementations
  - such as k-means, fuzzy k-means, Canopy, Dirichlet, and Mean-Shift
- Supports Distributed Naive Bayes and Complementary Naive Bayes classification implementations
- Comes with distributed fitness function capabilities for evolutionary programming

MUNI
ÚVT

Mahout Use Cases

- Yahoo: Spam Detection
- Foursquare: Recommendations
- SpeedDate.com: Recommendations
- Adobe: User Targetting
- Amazon: Personalization Platform
- Twitter: User Interest Modelling

MUNI
ÚVT

Use case Example

- Predict what the user likes based on
  - His/Her historical behavior
  - Aggregate behavior of people similar to him

Customers Who Bought This Item Also Bought

Pattern Recognition and Machine Learning... by Christopher M. Bishop

(50)

$76.10

The Elements of Statistical Learning: Data Minin... by Trevor Hastie

(38)

$71.96

Pattern Classification (2nd Edition) by Richard O. Duda

(29)

$88.52

MUNI ÚVT

Conclusion

During last two lessons, we introduced:

- Why Hadoop is needed
- The basic concepts of HDFS and MapReduce
- What sort of problems can be solved with Hadoop
- What other projects are included in the Hadoop ecosystem

MUNI
ÚVT


thank you
danke
謝謝
ngiyabonga
рахмат
teşekkür ederim
Баярлалаа
спасибо
faafetai lava
dank je
misaotra
matondo
paldies
grazzi
gracias
tapadh leat
хвала
asante
manana
obrigada
tenki
kiitos
dankie
nanni
nandri
dhanyavad
bedankt
hvala
mauruuru
köszönöm
bayarlalaa
gràçie
enkosi
djiere dieuf
tau
mochchakkeram
mamnun
дякую
chokrane
murakoze
dziękuję
akun
dankon
ačiū
chnorakaloutioun
gratias ago
gràcies
sulpáy
go raibh maith agat
sobodi
dëkuji
sagolun
sukriya
kop khun krap
taiku
obrigado
mèsi
didi madloba
najis tuke
ありがとう
tanemirt
grazie
arigatô
takk
dakujem
trugarez
kam sah hamnida
rahmat
terima kasih
rahmet
diolch
dhanyavadagalu
shukriya
merci
мерси
তোমাকে ধন্যবাদ
감사합니다
xièxie
ευχαριστώ

MUNI
ÚVT