Business Intelligence

Week 12

Big Data and Emerging Trends

- Introduction
- Apache Hadoop Ecosystem
- Apache Spark
- Spark Streaming
- Spark Mllib

Tilahun Melak(PhD)

June, 2026

Objectives

At the end of this lecture students will be able to :

- Describe the **Apache Hadoop ecosystem**, including HDFS, MapReduce, and Spark
- Explain the key features and benefits of **Apache Spark**
- Compare Apache Spark programming languages, including **Scala, Java, and Python**
- Apply core Spark components, including **MLlib, Spark Streaming, and GraphX**
- Explain the concept and role of **Resilient Distributed Datasets (RDDs)** in Spark

Introduction

- Big Data is used for a collection of data sets so large and complex that it is difficult to process using traditional tools.
- A recent survey says that 80% of the data created in the world are unstructured.
- One challenge is how we can store and process this big amount of data. In this lecture, we will discuss the top technologies used to store and analyse Big Data.

John Gantz, & David Reinsel. (2012). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. IDC.

Apache Hadoop

- **Apache Hadoop** is an open-source software framework for big data.

It has two basic parts:

- **Hadoop Distributed File System (HDFS)** is the storage system of Hadoop which splits big data and distributes it across many nodes in a cluster.
  a. Scaling out of H/W resources
  b. Fault tolerant
- **MapReduce:** Programming model that simplifies parallel programming.
  a. Map → apply()
  b. Reduce → summarize()
  c. Google used MapReduce for indexing websites.

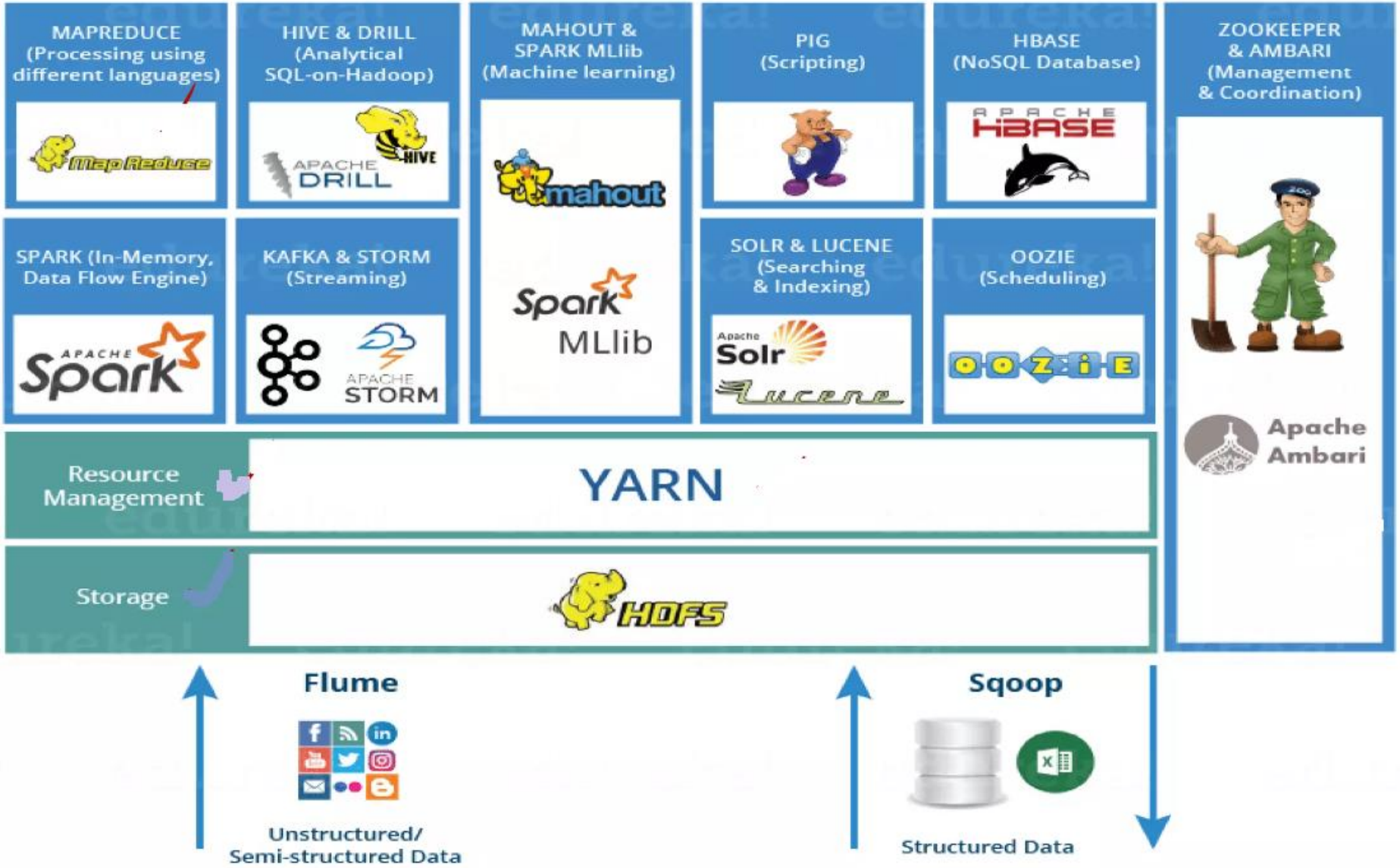
Apache Software Foundation. (n.d.). Apache Hadoop. Hadoop.

MapReduce

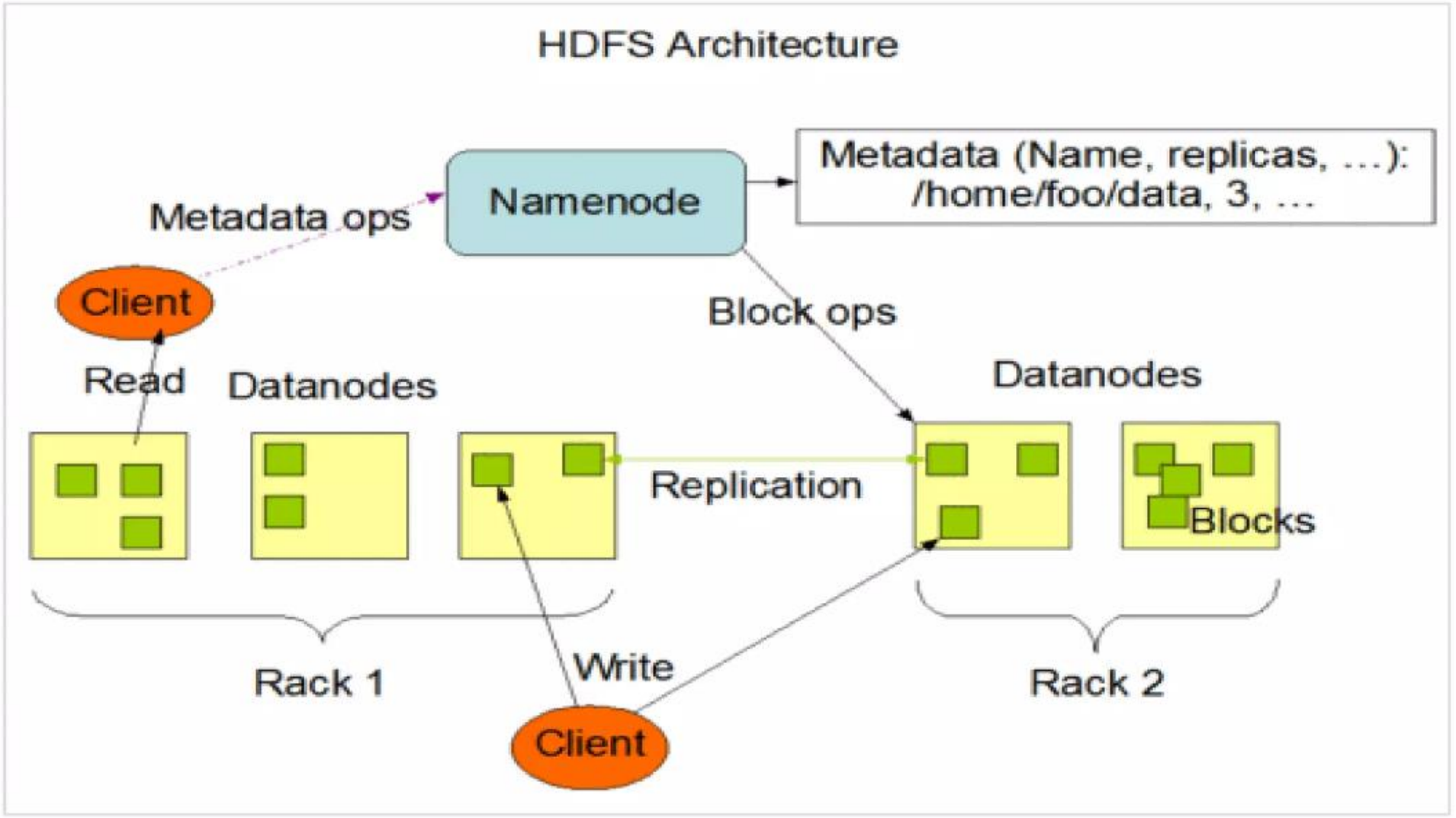
- **MapReduce** is a programming model and an associated implementation for **processing and generating large data sets**.
- Users specify a **map** function that processes a key/value pair to generate a set of intermediate key/value pairs, and a **reduce** function that merges all intermediate values associated with the same intermediate key.

Apache Software Foundation. (n.d.). Apache Hadoop. Hadoop.

Apache Hadoop Ecosystem



Apache Software Foundation. (2025). Apache Hadoop documentation. Apache Hadoop..



Basic architecture of Hadoop distributed file system (HDFS)

Apache Software Foundation. (2025). HDFS architecture guide. Apache Hadoop.

Apache Spark

- **Apache Spark** is a big data analytics framework that was originally developed at the University of California, Berkeley's AMPLab, in 2012. Since then, it has gained a lot of attraction both in academia and in industry.
- Apache Spark is a lightning-fast cluster computing technology, designed for fast computation.
- Apache Spark is a lightning-fast cluster computing technology, designed for fast computation.

Matei Zaharia, Mosharaf Chowdhury, Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing.

Apache Spark…

- **Apache Spark™** is a fast and general-purpose engine for large-scale data processing.
- **Spark** is a scalable data analytics platform that incorporates primitives for in-memory computing. As a result, it offers significant performance advantages over the traditional Hadoop cluster-storage approach. Spark is implemented using the Scala programming language and provides a powerful environment for data processing.

Matei Zaharia, Mosharaf Chowdhury, Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing.

Apache Spark…

- **Spark** is well suited for complex analytics and provides strong support for machine learning libraries.
- **Spark** is an open-source computing framework that was originally developed at the AMPLab (Algorithms, Machines, and People Lab) at the University of California, Berkeley. It was later donated to the Apache Software Foundation, where it continues to be actively developed and maintained.

Matei Zaharia, Mosharaf Chowdhury, Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing.

Apache Spark…

- Fast, expressive cluster computing system compatible with Apache Hadoop
  - Works with any Hadoop-supported storage system (HDFS, S3, SequenceFile, Avro, ...)
- Improves **efficiency** through:
  - In-memory computing primitives → ***Up to 100× faster***
  - General computation graphs
- Improves **usability** through:
  - Rich APIs in Java, Scala, Python
  - Interactive shell → ***Often 2–10× less code***

Matei Zaharia, Mosharaf Chowdhury, Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing.

Apache Spark Benefits

- In contrast to Hadoop's two-stage, disk-based MapReduce paradigm, Spark's multi-stage in-memory primitives provide performance improvements of up to **100 times faster** for certain applications.
- Spark allows user programs to load data into a cluster's memory and query it repeatedly.
- Spark is particularly well suited for **machine learning applications**, which often involve iterative computations and repeated in-memory processing.

Matei Zaharia, Mosharaf Chowdhury, Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing.

Apache Spark Benefits…

- Spark requires both a **cluster manager** and a **distributed storage system**. For cluster management, Spark supports:
  - Standalone Spark clusters
  - Hadoop YARN
  - Apache Mesos
- For distributed storage, Spark can interface with various storage systems, including:
  - Hadoop Distributed File System (HDFS)
  - Amazon S3
  - Other compatible distributed storage platforms

Matei Zaharia, Mosharaf Chowdhury, Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing.

How to Run Apache Spark?

- Local multicore: just a library in your program
- EC2: scripts for launching a Spark cluster
- Private cluster: Mesos, YARN, Standalone Mode

Matei Zaharia, Mosharaf Chowdhury, Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing.

Scala vs Java

- Spark originally written in Scala, which allows concise function syntax and interactive use
- APIs in Java, Scala and Python
- Interactive shells in Scala and Python

Matei Zaharia, Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), 15–28.

About Scala

High-level language for the Java VM

- Object-oriented + functional programming

Statically typed

- Comparable in speed to Java
- Often no need to explicitly write types due to type inference

Interoperates with Java

- Can use any Java class, inherit from it, and leverage existing Java libraries
- Scala code can also be called from Java applications

Matei Zaharia, Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), 15–28.

Best Way to Learn Scala

- Interactive shell: just type scala
- Supports importing libraries, tab completion, and all language constructs available in Scala

Matei Zaharia, Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), 15–28.

Quick Tour

Declaring variables:

```
var x: Int = 7
var x = 7  // type inferred
val y = "hi"  // read-only
```

Java equivalent:

```
int x = 7;

final String y = "hi";
```

Functions:

```
def square(x: Int): Int = x*x

def square(x: Int): Int = {
  x*x
}
```

Last expression in block returned

Java equivalent:

```
int square(int x) {
  return x*x;
}
```

```
def announce(text: String) {
  println(text)
}
```

```
void announce(String text) {
  System.out.println(text);
}
```

Quick Tour…

Generic types:

```
var arr = new Array[Int](8)

var lst = List(1, 2, 3)
// type of lst is List[Int]
```

Factory method

Java equivalent:

```
int[] arr = new int[8];
List<Integer> lst =
  new ArrayList<Integer>();
lst.add(...)
```

Can’t hold primitive types

Indexing:

```
arr(5) = 7

println(lst(5))
```

Java equivalent:

```
arr[5] = 7;

System.out.println(lst.get(5));
```

Quick Tour…

Processing collections with functional programming:

```
val list = List(1, 2, 3)
list.foreach(x => println(x))  // prints 1, 2, 3
list.foreach(println)          // same

list.map(x => x + 2)  // => List(3, 4, 5)
list.map(_ + 2)       // same, with placeholder notation

list.filter(x => x % 2 == 1)  // => List(1, 3)
list.filter(_ % 2 == 1)       // => List(1, 3)

list.reduce((x, y) => x + y)  // => 6
list.reduce(_ + _)            // => 6
```

Scala Closure Syntax

```
(x: Int) => x + 2   // full version
x => x + 2    // type inferred
_ + 2         // when each argument is used exactly once
x => {        // when body is a block of code
  val numberToAdd = 2
  x + numberToAdd
}

// If closure is too long, can always pass a function
def addTwo(x: Int): Int = x + 2

list.map(addTwo)
```

Other Collection Methods

- Scala collections provide many other functional methods; for example, Google for "Scala Seq"

Method on Seq[T]	Explanation
`map(f: T => U): Seq[U]`	Pass each element through f
`flatMap(f: T => Seq[U]): Seq[U]`	One-to-many map
`filter(f: T => Boolean): Seq[T]`	Keep elements passing f
`exists(f: T => Boolean): Boolean`	True if one element passes
`forall(f: T => Boolean): Boolean`	True if all elements pass
`reduce(f: (T, T) => T): T`	Merge elements using f
`groupBy(f: T => K): Map[K,List[T]]`	Group elements by f(element)
`sortBy(f: T => K): Seq[T]`	Sort elements by f(element)
. . .	

Spark Overview

- **Goal: Work with distributed collections as you would with local ones**

- Concept: resilient distributed datasets (RDDs)
  - Immutable collections of objects spread across a cluster
  - Built through parallel transformations (map, filter, etc)
  - Automatically rebuilt on failure
  - Controllable persistence (e.g. caching in RAM)

Matei Zaharia, Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12) (pp. 15–28).

Main Primitives

Resilient distributed datasets (RDDs)

- Immutable, partitioned collections of objects

- Transformations (e.g. map, filter, groupBy, join)
  - Lazy operations to build RDDs from other RDDs

Actions (e.g. count, collect, save)

- Return a result or write it to storage

Matei Zaharia, Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12) (pp. 15–28).

Example: Mining Console Logs

interactively search for patterns

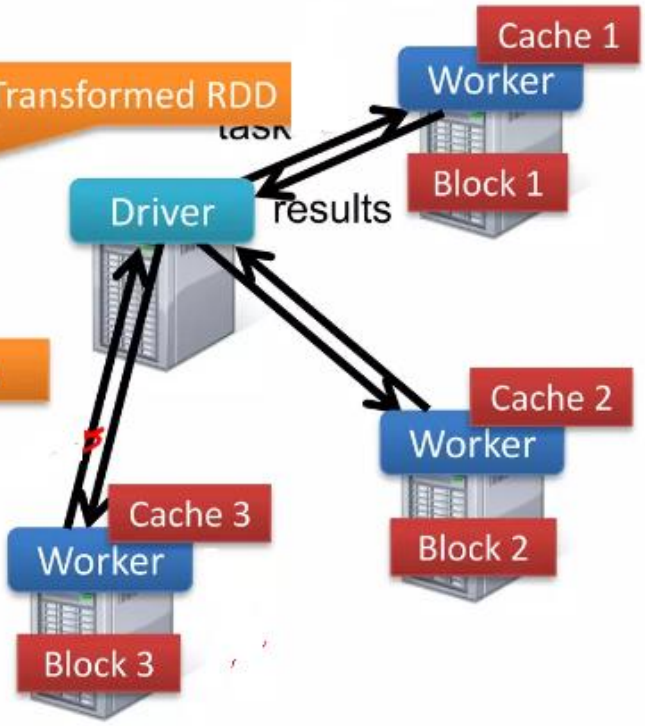
```
lines = spark textFile(“hdfs://...”)
errors = lines.filter(lambda s: s.startswith(“ERROR”))
messages = errors.map(lambda s: s.split(‘\t’)[2])
messages.cache()

messages.filter(lambda s: “foo” in s).count()
messages.filter(lambda s: “bar” in s).count()
. . .
```

Base RDD

Transformed RDD

Action

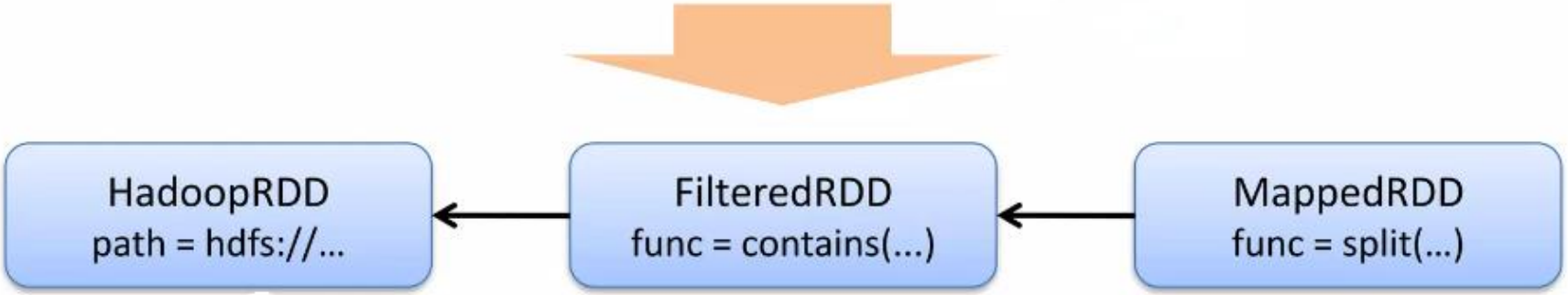


RDD Fault Tolerance

RDDs track the transformations used to build them (their *lineage*) to recompute lost data

E.g:

```
messages = textFile(...).filter(lambda s: s.contains("ERROR"))
                        .map(lambda s: s.split('\t')[2])
```



Spark Streaming

- **Spark Streaming** is an extension of the core Spark API that enables scalable, high-throughput, fault-tolerant stream processing of live data streams.
- Streaming data input from HDFS, Kafka, Flume, TCP sockets, Kinesis, etc.
- Spark ML (Machine Learning) functions and GraphX graph processing algorithms are fully applicable to streaming data.

Matei Zaharia, Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13), 423–438.

Matei Zaharia, Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13), 423–438.

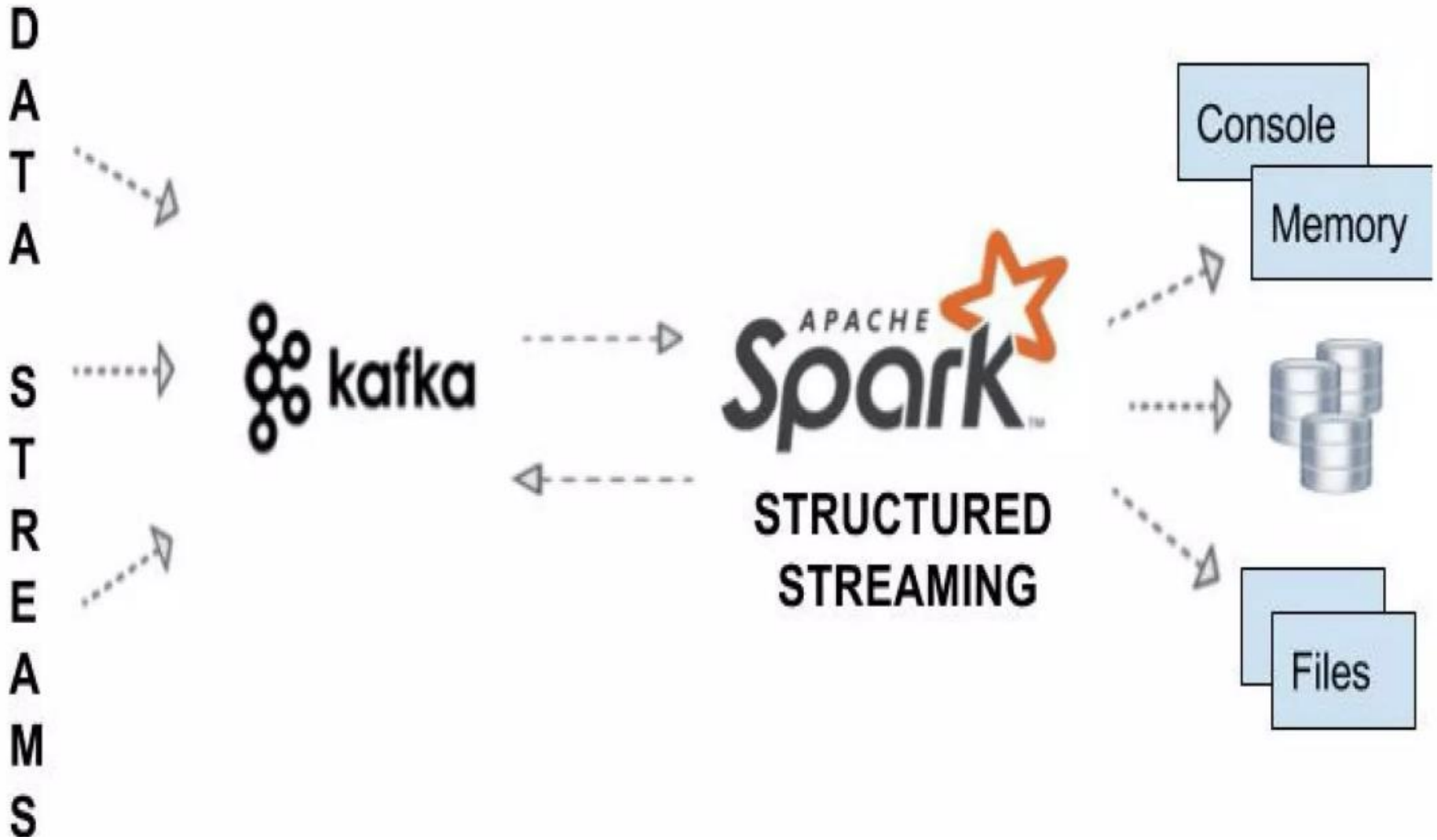
Kafka Streaming Ecosystem

• Apache Kafka is an open-source stream-processing software platform developed by the Apache Software Foundation, written in Scala and Java.

• Apache Kafka is an open-source distributed streaming platform capable of handling trillions of events a day. Kafka is based on an abstraction of a distributed commit log.

Matei Zaharia, Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13), 423–438.

Kafka



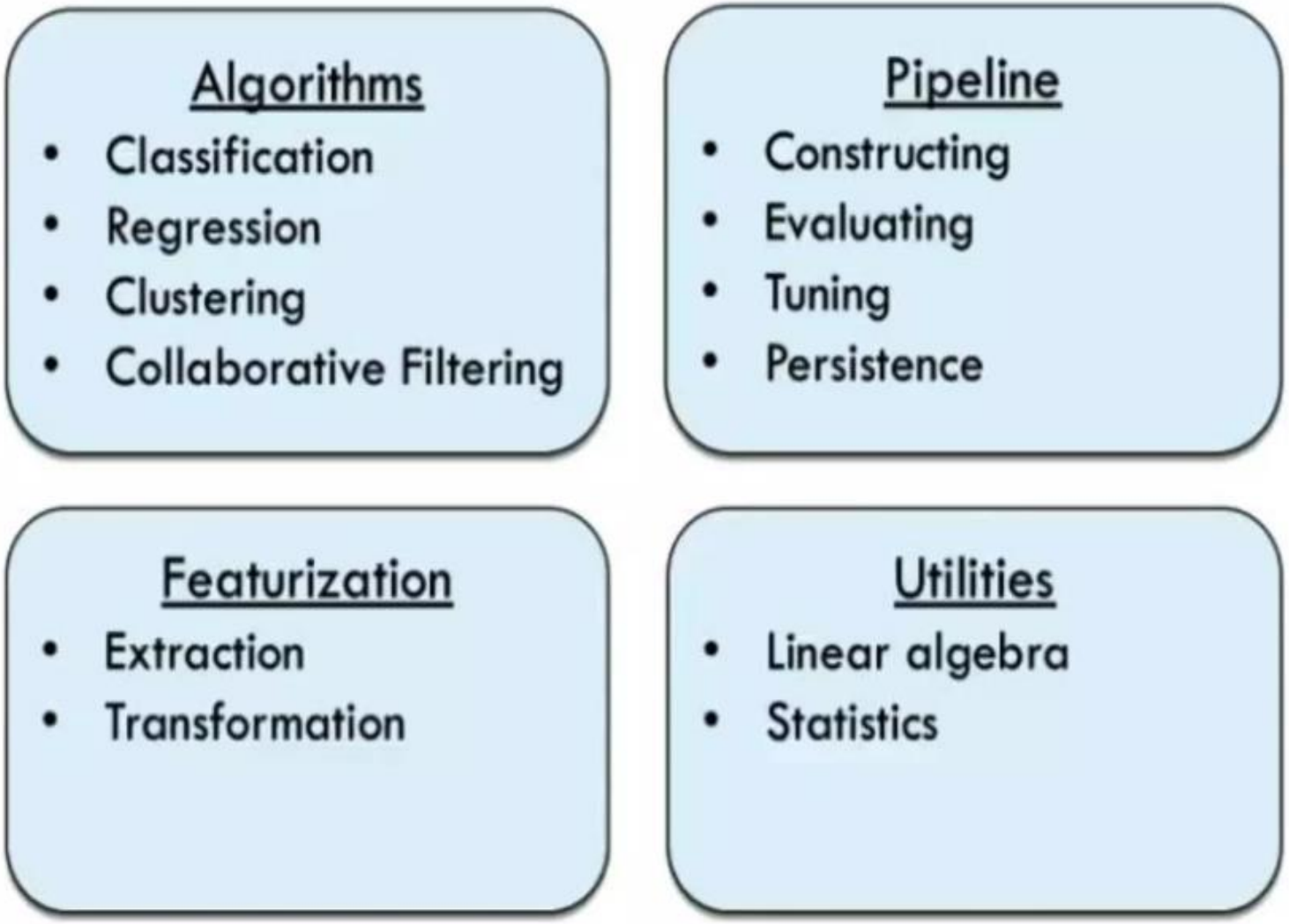
Matei Zaharia, Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13), 423–438.

Spark MLlib

- Spark MLlib is a distributed machine-learning framework on top of Spark Core.
- MLlib is Spark's scalable machine learning library consisting of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, and dimensionality reduction.

Matei Zaharia, Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2016). Apache Spark: A unified engine for big data processing. Communications of the ACM, 59(11), 56–65.

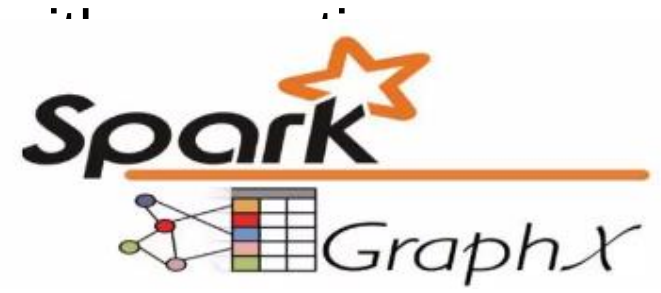
Spark MLlib Components



Matei Zaharia, Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2016). Apache Spark: A unified engine for big data processing. Communications of the ACM, 59(11), 56–65.

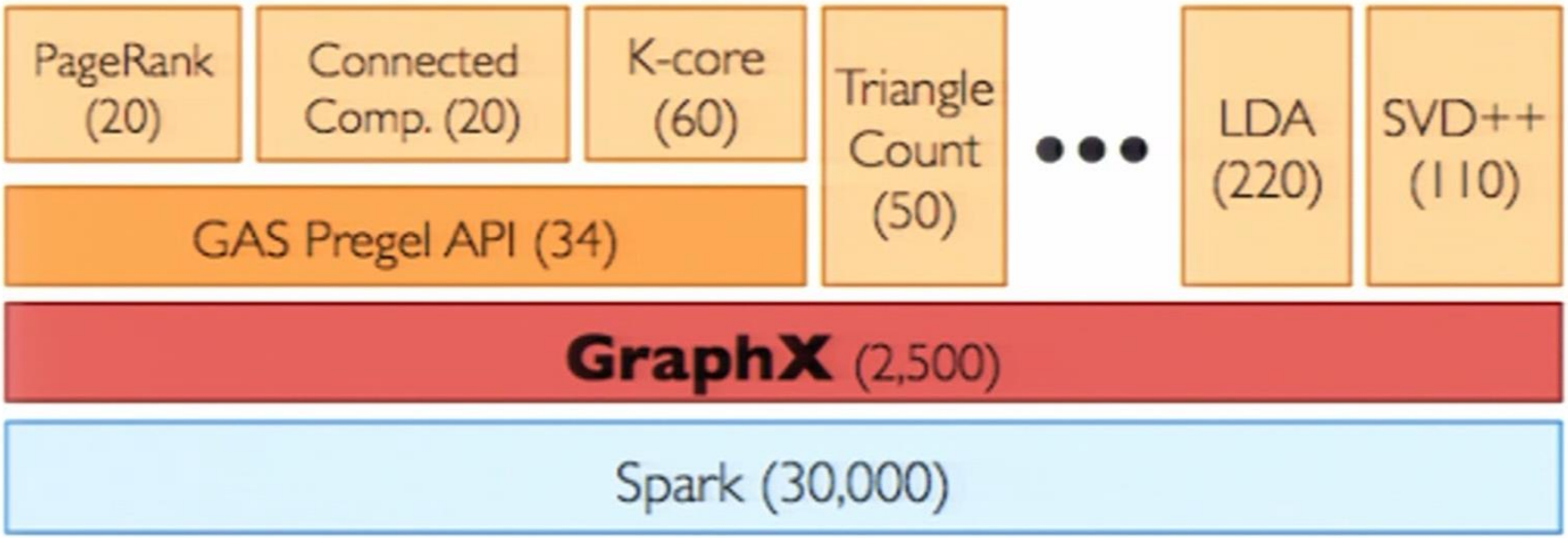
Spark GraphX

- **GraphX** is a new component in Spark for graphs and graph-parallel computation. At a high level, GraphX extends the Spark RDD by introducing a new graph abstraction.
- **GraphX** reuses the Spark RDD concept, simplifies graph analytics tasks, and provides the ability to perform operations on a directed multigraph attached to each vertex and edge.



Matei Zaharia, Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2016). Apache Spark: A unified engine for big data processing. Communications of the ACM, 59(11), 56–65.

Spark GraphX



- Graphx is a thin layer on the top of spark general-purpose-data flow framework (lines of code).

Matei Zaharia, Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2016). Apache Spark: A unified engine for big data processing. Communications of the ACM, 59(11), 56–65.

Summary

- Overview of the **Apache Hadoop ecosystem**, including HDFS, MapReduce, and Spark
- Introduction to **Apache Spark** and its key features and benefits for big data processing
- Comparison of Spark programming languages: **Scala, Java, and Python**
- Application of Spark components, including **MLlib, Spark Streaming, and GraphX**
- Explanation of **Resilient Distributed Datasets (RDDs)** as the core abstraction in Spark

References

- John Gantz, & David Reinsel. (2012). The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. IDC.
- Apache Software Foundation. (n.d.). Apache Hadoop. Hadoop.
- Apache Software Foundation. (2025). Apache Hadoop documentation. Apache Hadoop..
- Apache Software Foundation. (2025). HDFS architecture guide. Apache Hadoop.
- Matei Zaharia, Mosharaf Chowdhury, Franklin, M. J., Shenker, S., & Stoica, I. (2010). Spark: Cluster computing with working sets. Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing.
- Matei Zaharia, Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2012). Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), 15–28.
- Matei Zaharia, Das, T., Li, H., Hunter, T., Shenker, S., & Stoica, I. (2013). Discretized streams: Fault-tolerant streaming computation at scale. In Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13), 423–438.
- Matei Zaharia, Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., & Stoica, I. (2016). Apache Spark: A unified engine for big data processing. Communications of the ACM, 59(11), 56–65.