Dell - Data Engineering Training

Uday Kumar – Data Platform Architect

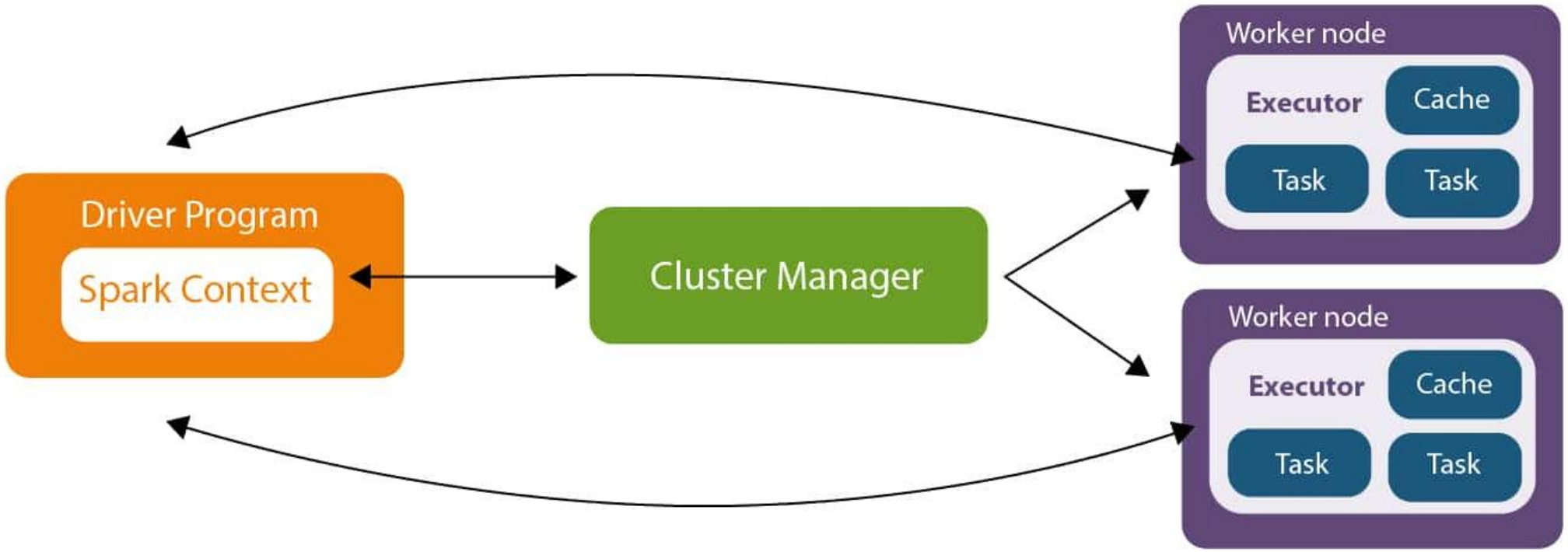
Day 7

Data Engineering Training

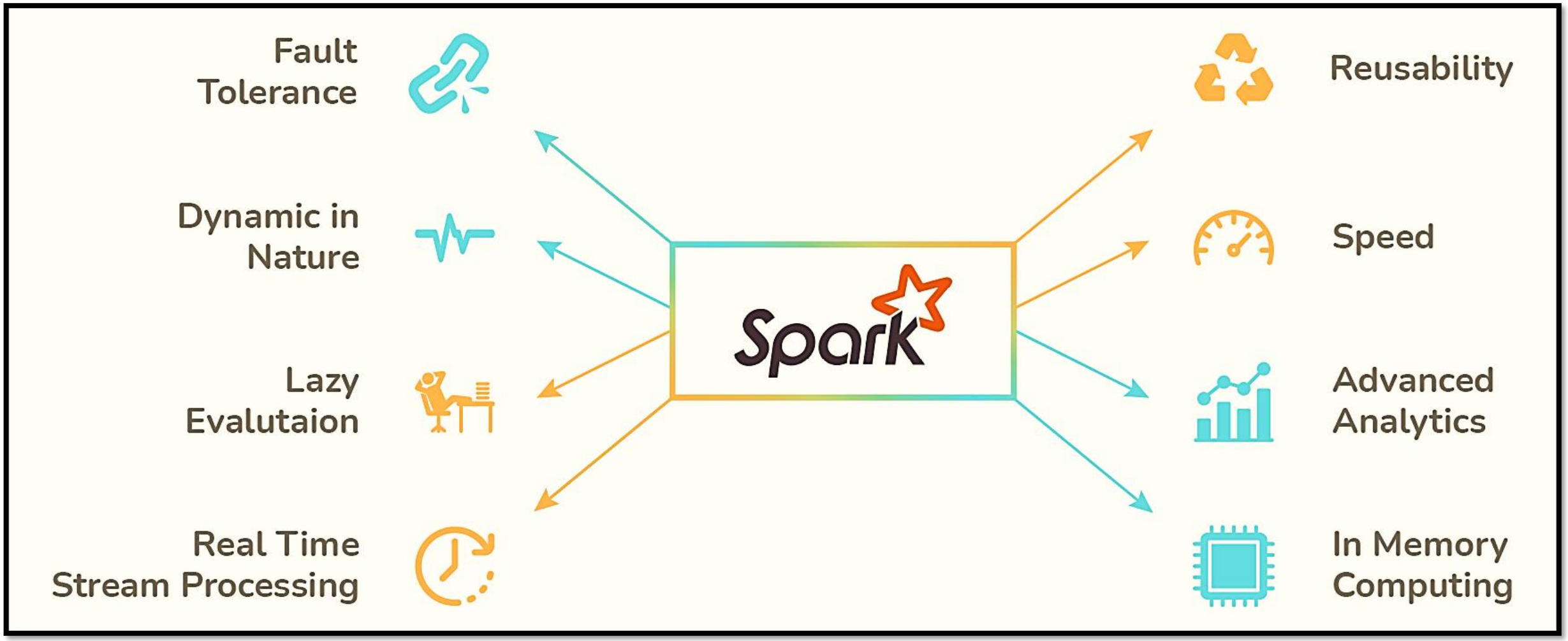
Agenda

1. Spark Architecture
2. Integration of Spark and Flink
3. Files to write to Kafka Producer
4. Kafka Consumer & Stream

Spark Cluster Architecture



APACHE SPARK FEATURES



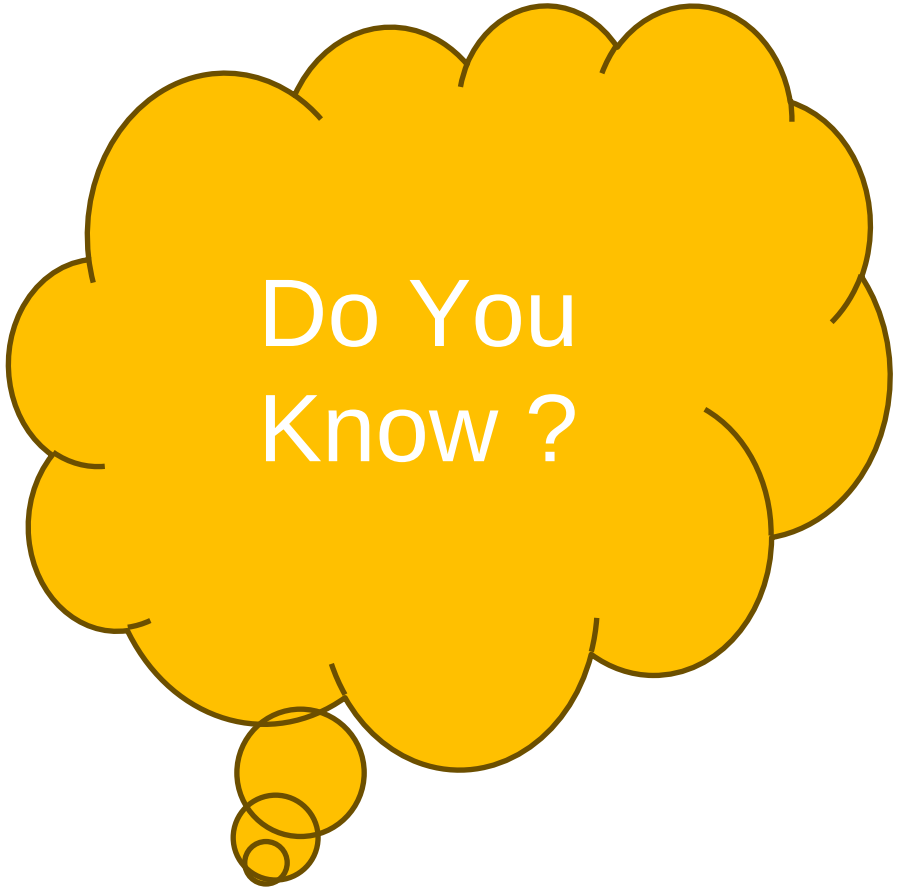
APACHE SPARK FEATURES

The most widely-used engine for scalable computing

Thousands of companies, including 80% of the Fortune 500, use Apache Spark™. Over 2,000 contributors to the open source project from industry and academia.

Ecosystem

Apache Spark™ integrates with your favorite frameworks, helping to scale them to thousands of machines.



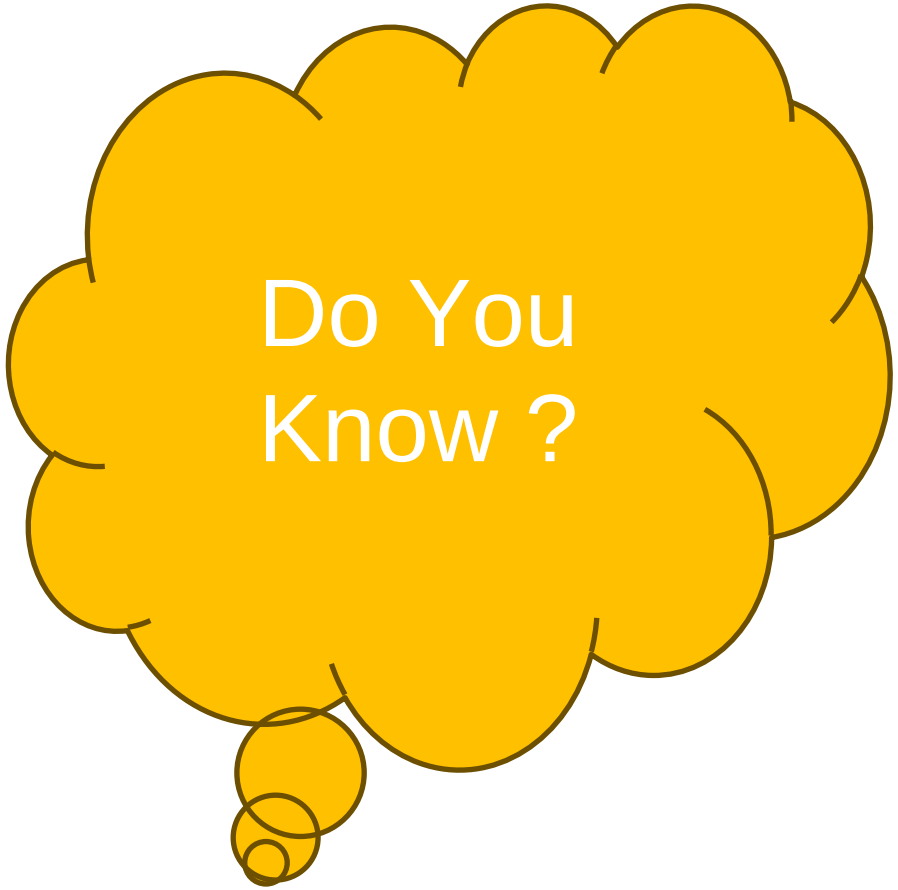
APACHE SPARK FEATURES

RDD stands for **Resilient Distribution Datasets**. It is a fault-tolerant collection of parallel running operational elements. The partitioned data of RDD is distributed and immutable.

There are two types of datasets:

Parallelized collections: Meant for running parallelly.

Hadoop datasets: These perform operations on file record systems on HDFS or other storage systems



APACHE SPARK ECO SYSTEM

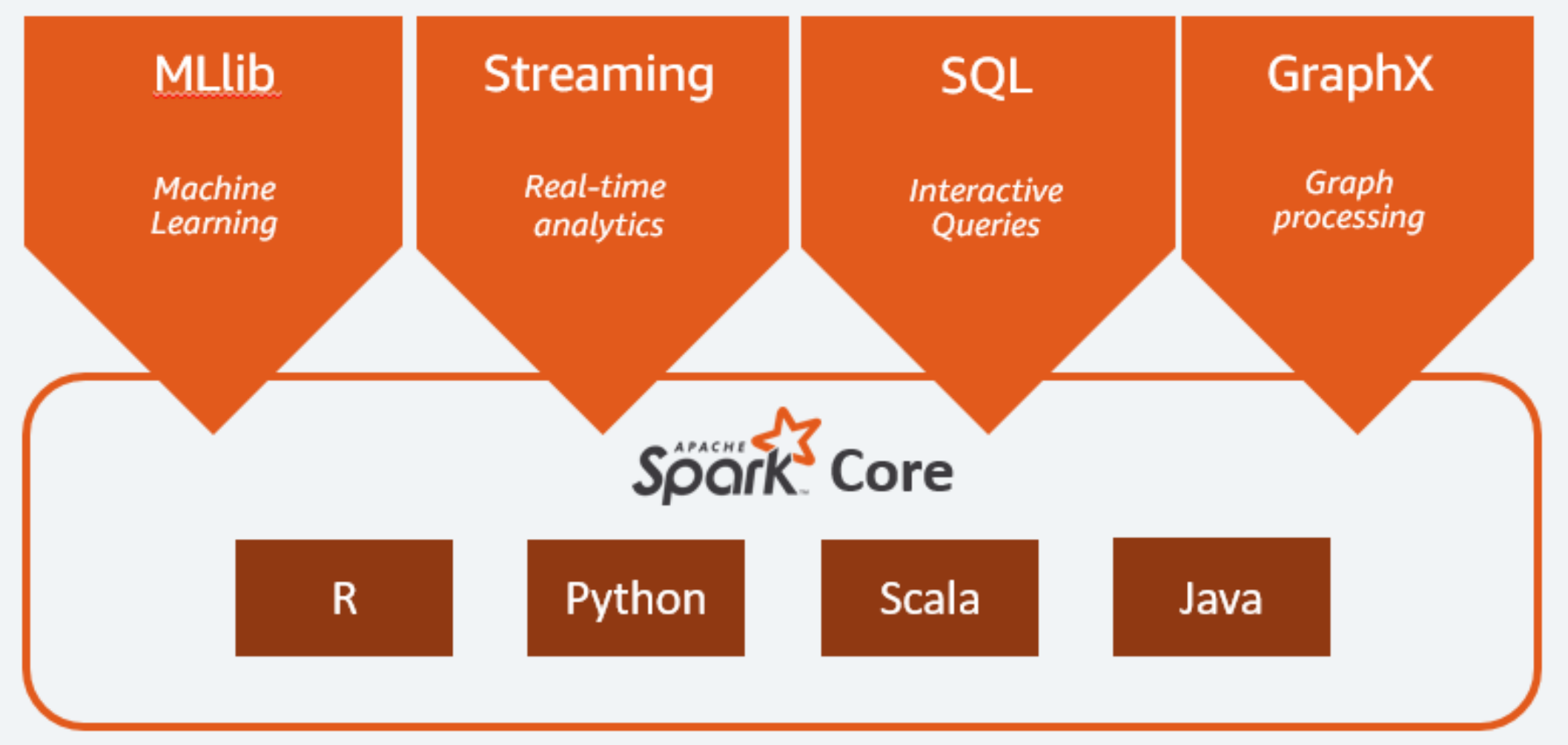
IntelliPaat

Data science and Machine learning

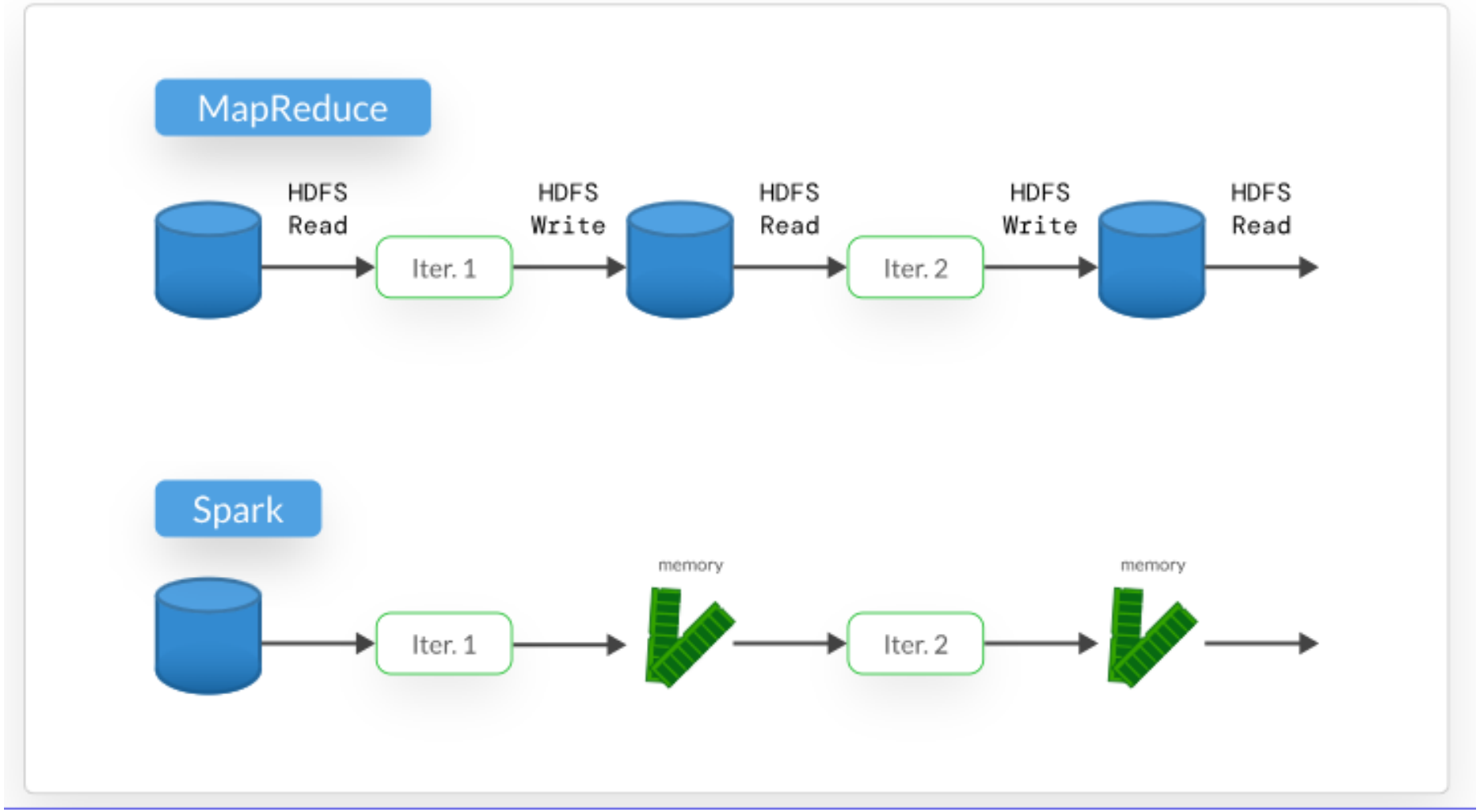
SQL analytics and BI

Storage and Infrastructure

APACHE SPARK WORKLOADS

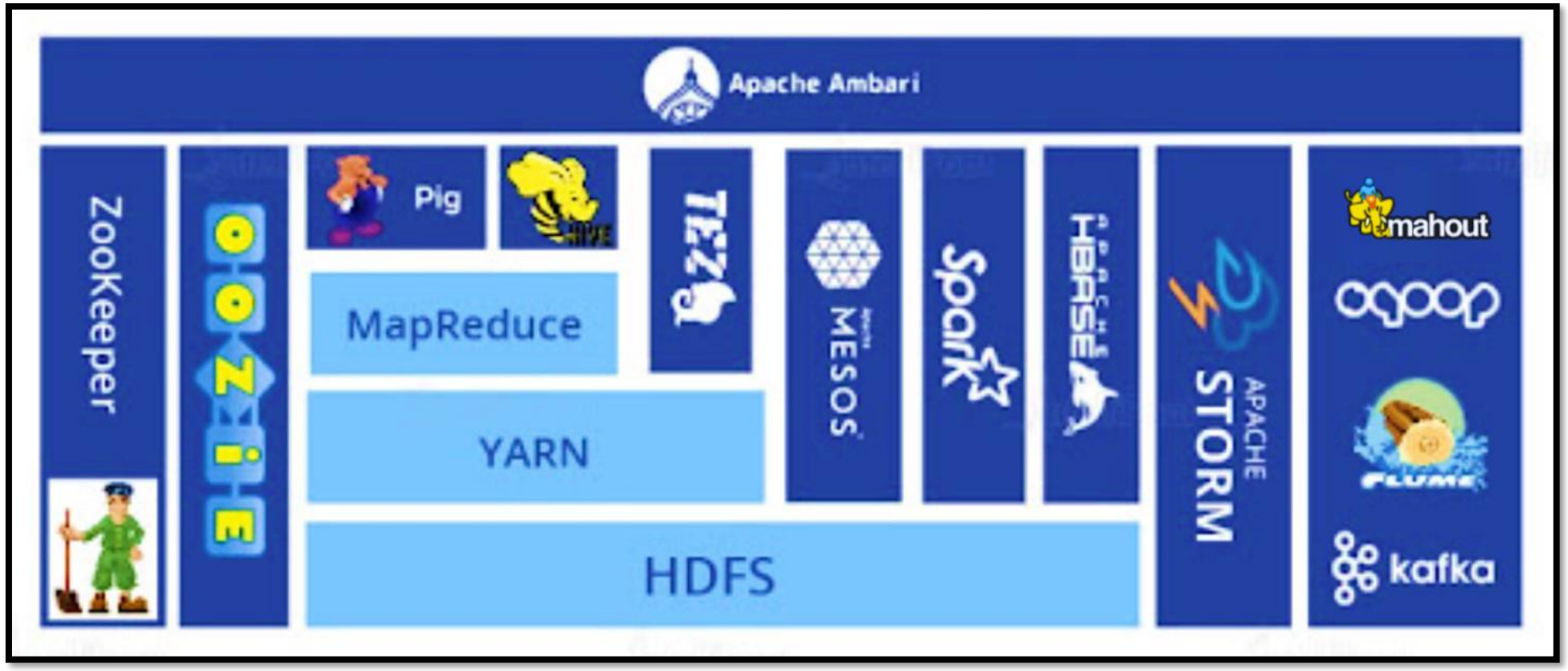


HADOOP ECOSYSTEM

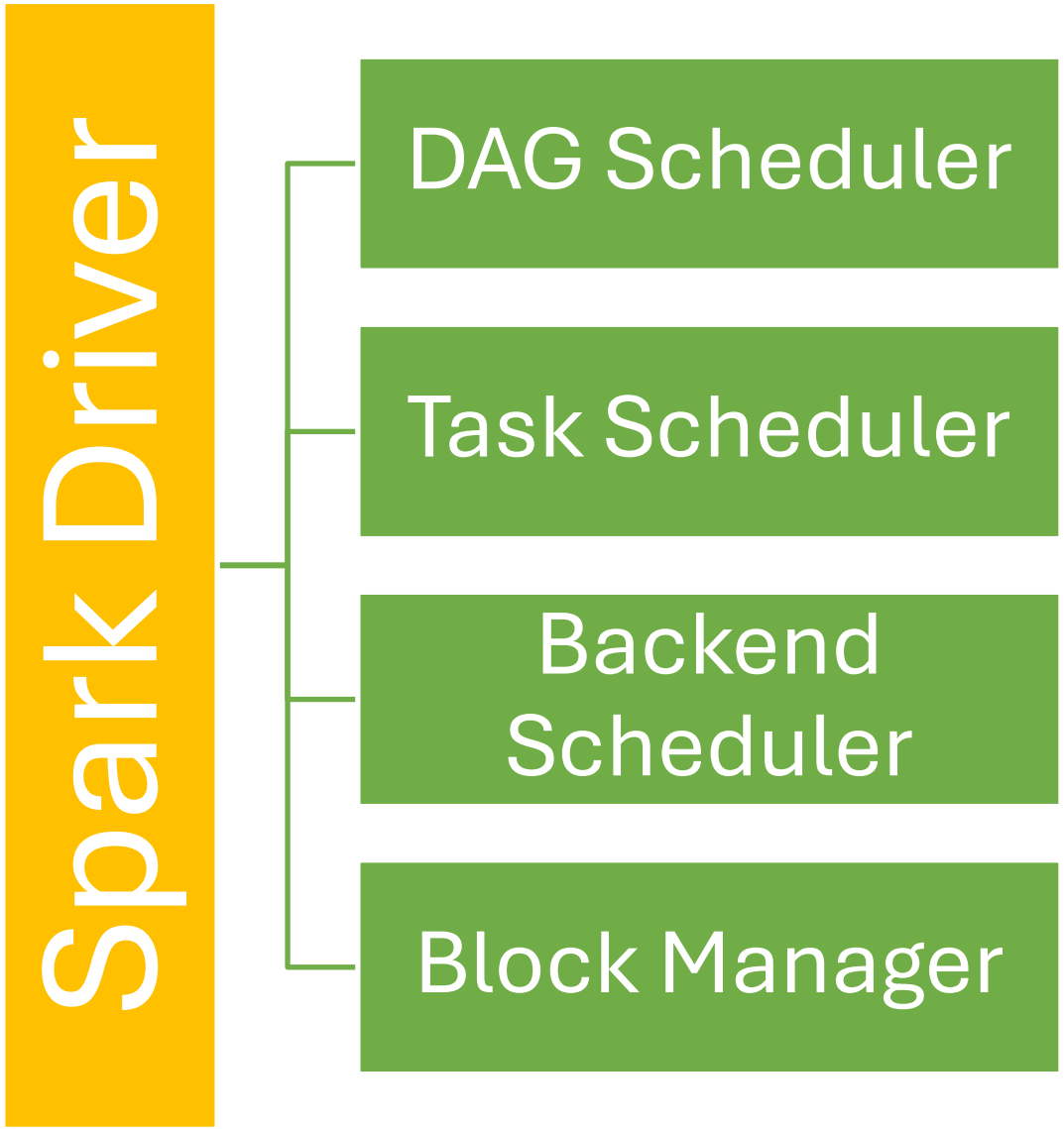


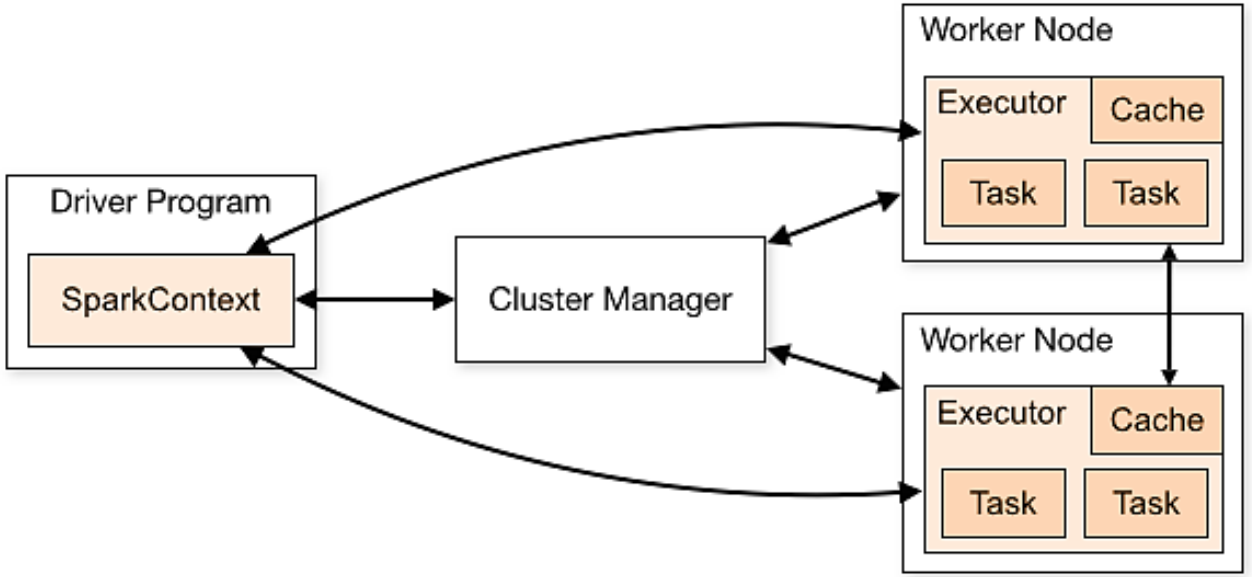
Traditional MapReduce writes to disk, but Spark can process in-memory

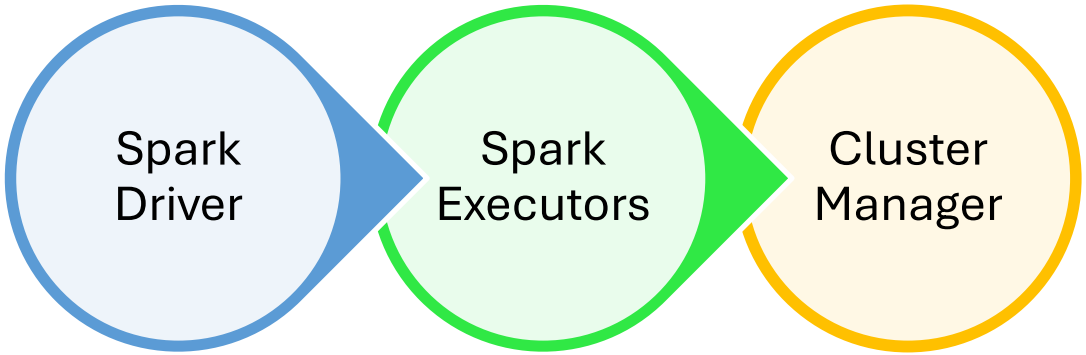
HADOOP ECOSYSTEM



APACHE SPARK ARCHITECTURE





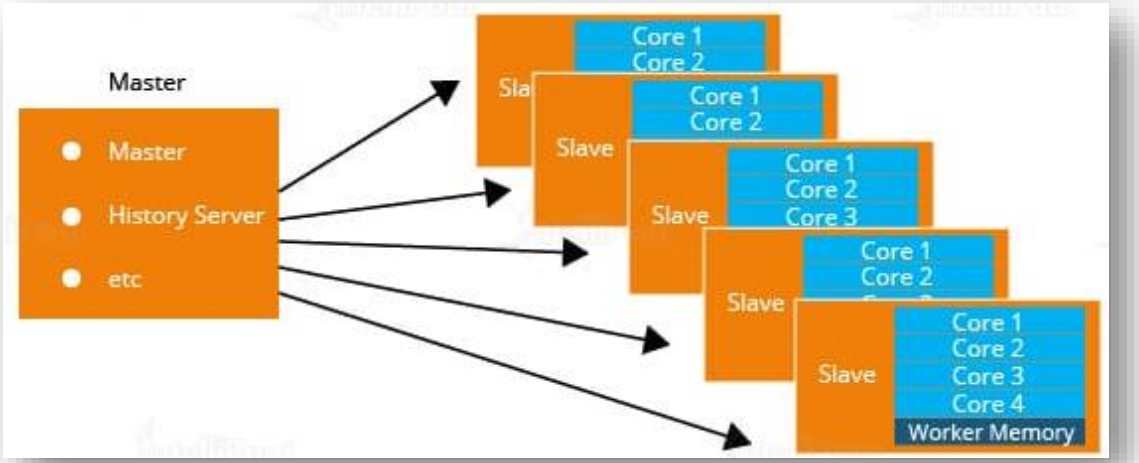


APACHE SPARK ARCHITECTURE – CLUSTER MANAGER

APACHE SPARK ARCHITECTURE – STANDALONE

The Spark Standalone Cluster comprises a **Standalone Master that functions as the Resource Manager**, along with Standalone Workers **serving as the worker nodes**. In this cluster mode, each worker node hosts a sole executor responsible for executing tasks.

To commence the execution process, **a client establishes a connection with the Standalone Master**, requesting the requisite resources. Acting as the application master, the client collaborates with the Resource Manager to procure the necessary resources.

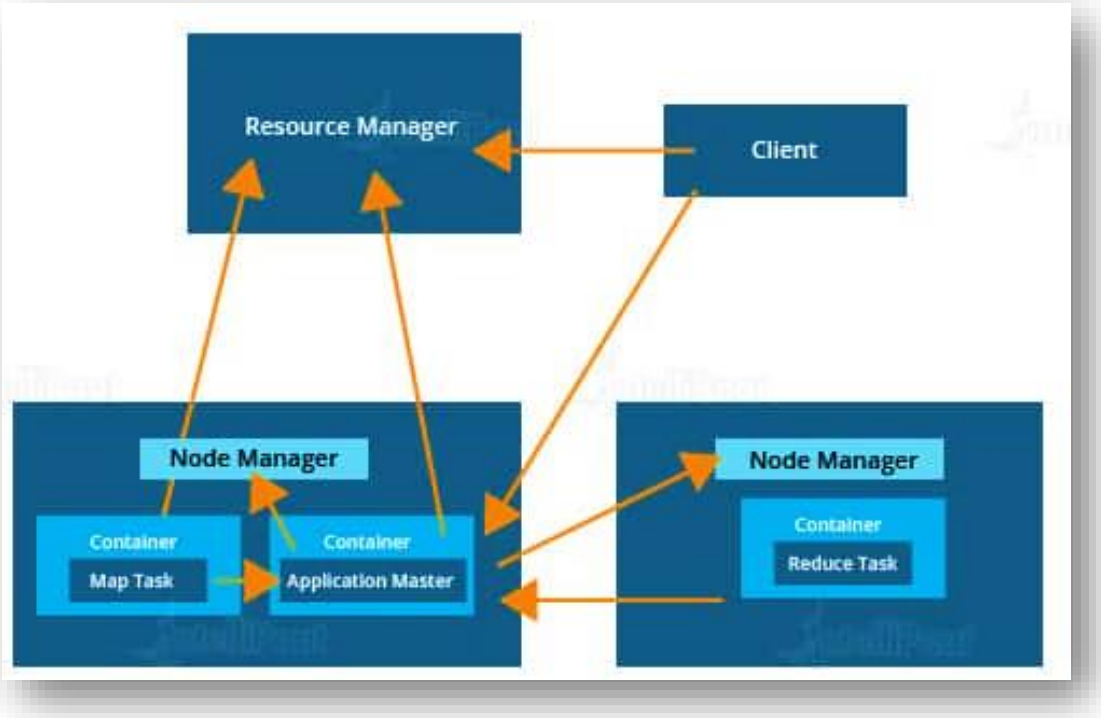


APACHE SPARK ARCHITECTURE – YARN

Yet Another Resource Negotiator(YARN) takes care of resource management for the Hadoop ecosystem. It has two components:

Resource Manager: It manages resources on all applications in the system. It consists of a Scheduler and an Application Manager. The Scheduler allocates resources to various applications.

Node Manager: Node Manager consists of an Application Manager and a Container. Each task of MapReduce runs in a container. An application or job thus requires one or more containers, and the Node Manager monitors these containers and resource usage. This is reported to the Resource Manager.



APACHE SPARK ARCHITECTURE

Two Main Abstractions of Apache Spark

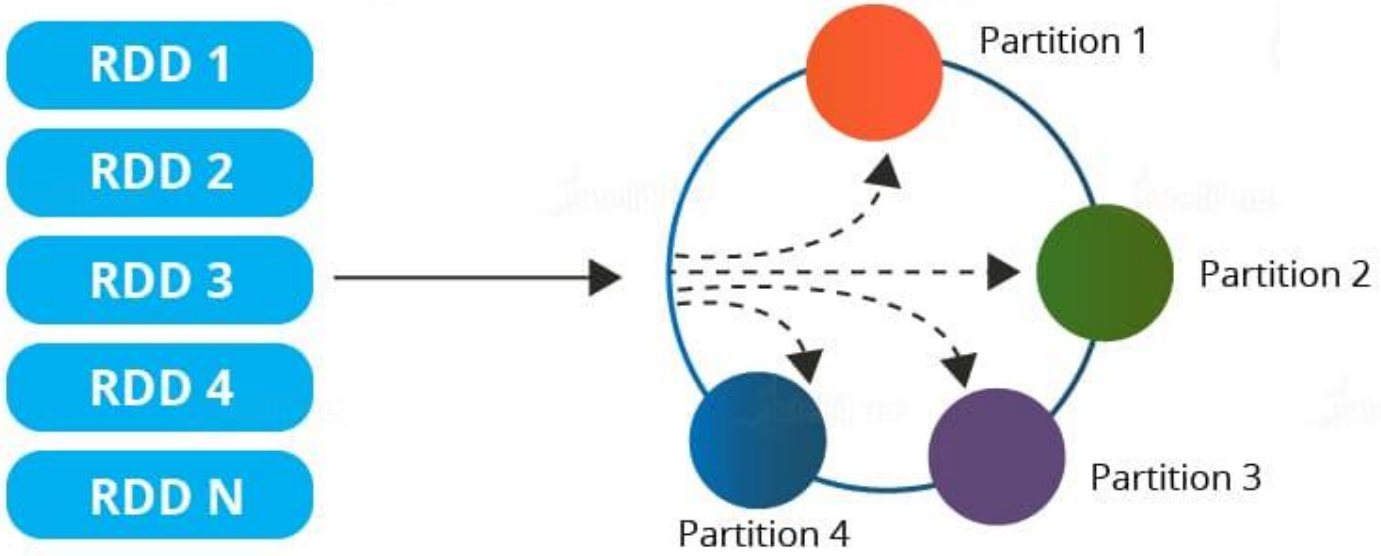
Resilient Distributed Dataset (RDD)

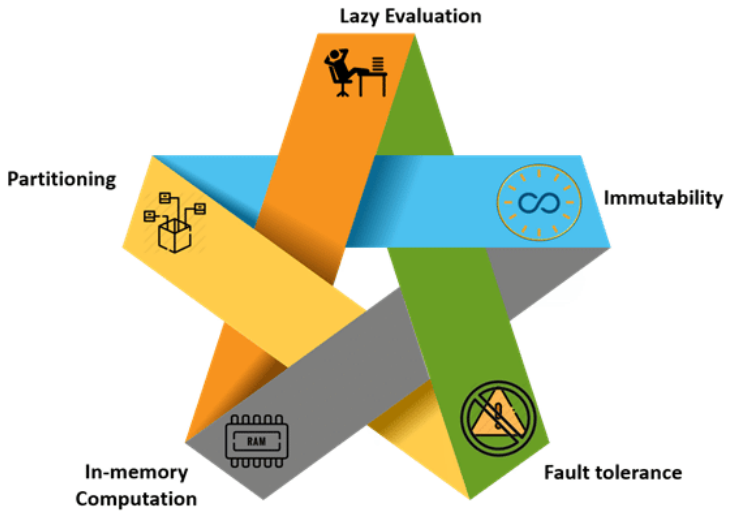
Directed Acyclic Graph (DAG)

APACHE SPARK ARCHITECTURE - RDD

RDDs are the main logical data units in Spark. They are a **distributed collection of objects**, which are stored in memory or on disks of different machines of a cluster. A **single RDD can be divided into multiple logical partitions** so that these partitions can be stored and processed on different machines of a cluster.

RDDs are immutable (read-only) in nature. You cannot change an original RDD, but you can create new RDDs by performing coarse-grain operations, like transformations, on an existing RDD.





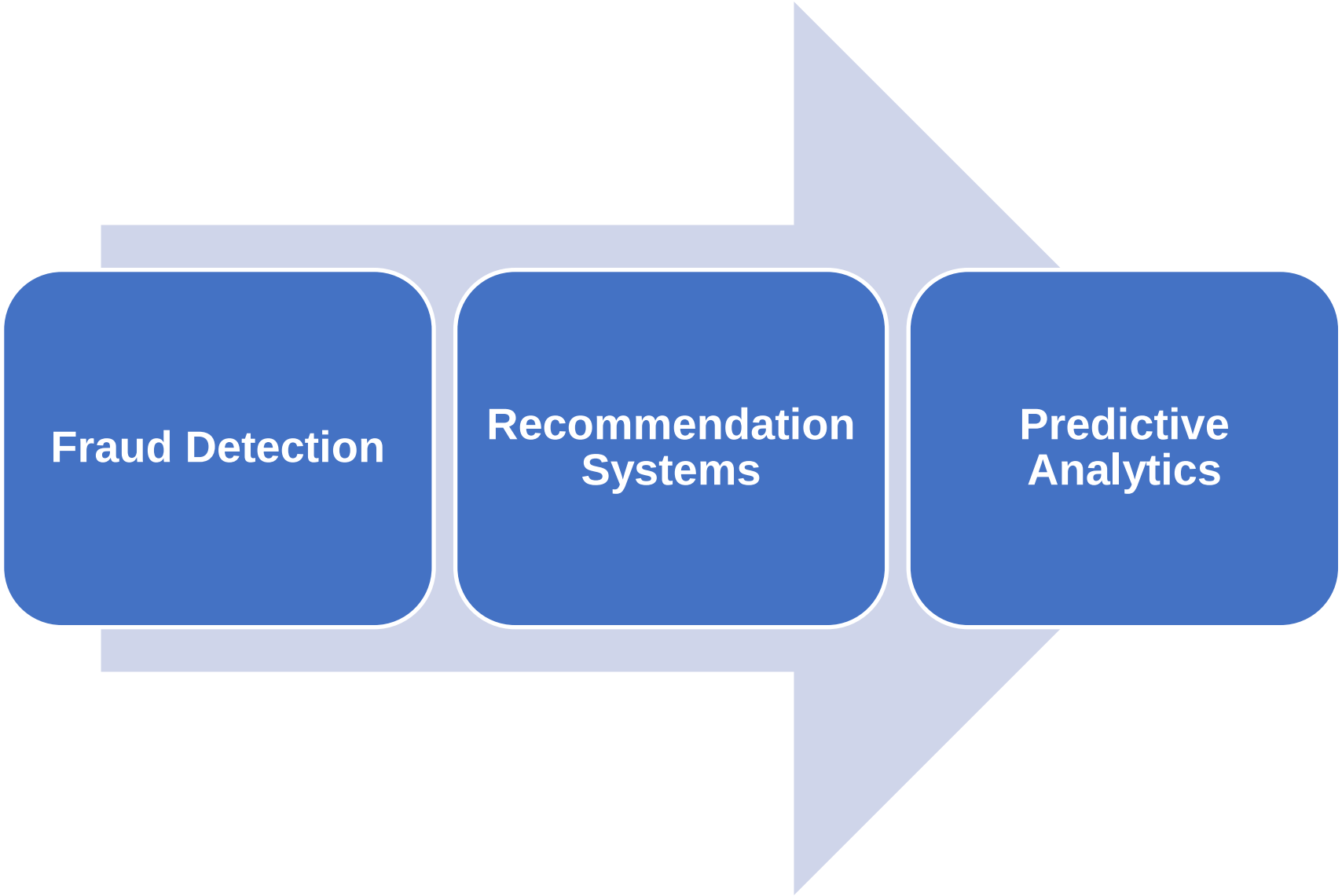
APACHE SPARK ARCHITECTURE - EXECUTION MODES

IntelliPaat

- Cluster mode
- Client mode
- Local mode

APACHE SPARK USE CASES

APACHE SPARK USE CASES



APACHE SPARK KEY CONCEPTS

Apache Spark: It is an open-source, Hadoop-compatible, fast and expressive cluster computing platform.

Resilient Distributed Datasets (RDDs): The core concept in Apache Spark is RDDs, which are the immutable distributed collections of data that are partitioned across machines in a cluster.

Transformation: It is an operation performed on an RDD, such as filter(), map(), or union(), which yields another RDD.

Action: It is an operation that triggers a computation such as count(), first(), take(n), or collect().

Partition: It is a logical division of data stored on a node in a cluster.

SparkContext: It holds a connection with Spark Cluster Management.

Driver: The process of running the main() function of an application and creating the SparkContext is managed by the driver.

Worker: Any node which can run the program on the cluster is called a worker.

Cluster Manager: A cluster manager allocates resources to each application in a driver program. There are three types of cluster managers supported by Apache Spark

PYSPARK

Pyspark is a connection between Apache Spark and Python.

It is a Spark Python API and helps you connect with Resilient Distributed Datasets (RDDs) to Apache Spark and Python. Let's talk about the basic concepts of Pyspark RDD, DataFrame, and spark files.

PySpark is considered an interface for Apache Spark in Python. Through PySpark, you can write applications by using Python APIs.

PYSPARK SQL

PySpark SQL establishes the connection between the RDD and relational table. It provides much closer integration between relational and procedural processing through declarative Dataframe API, which is integrated with Spark code.

PySpark SQL has a language combined **User-Defined Function (UDFs)**. UDF is used to define a new column-based function that extends the vocabulary of Spark SQL's DSL for transforming DataFrame.

pyspark.sql.SparkSession: It represents the main entry point for DataFrame and SQL functionality.

pyspark.sql.DataFrame: It represents a distributed collection of data grouped into named columns.

pyspark.sql.Column: It represents a column expression in a DataFrame.

pyspark.sql.Row: It represents a row of data in a DataFrame.

pyspark.sql.GroupedData: Aggregation methods, returned by DataFrame.groupBy().

pyspark.sql.DataFrameNaFunctions: It represents methods for handling missing data (null values).

pyspark.sql.DataFrameStatFunctions: It represents methods for statistics functionality.

pysark.sql.functions: It represents a list of built-in functions available for DataFrame.

pyspark.sql.types: It represents a list of available data types.

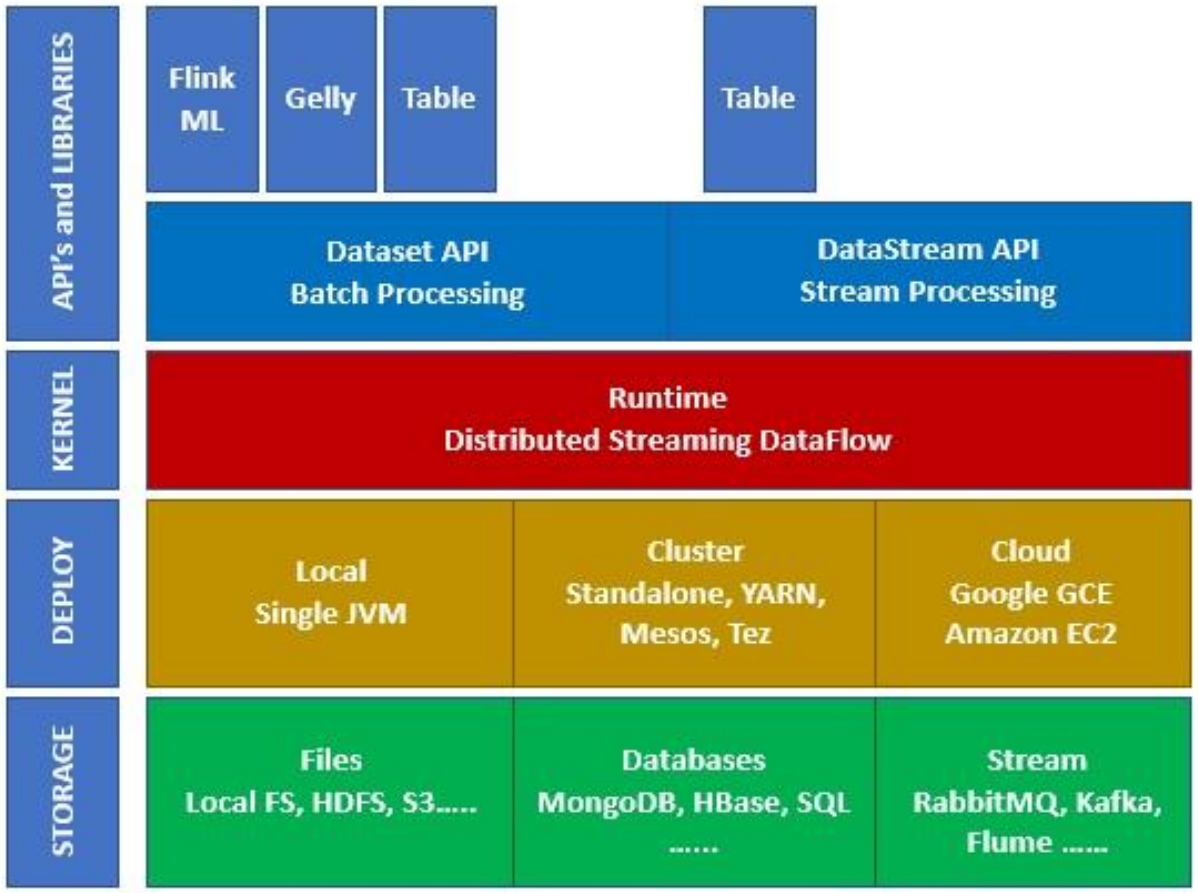
pyspark.sql.Window: It is used to work with Window functions.

Apache Flink

APACHE FLINK

Apache Flink is a **real-time processing framework** which can process streaming data. It is an open source stream processing framework for high-performance, scalable, and accurate real-time applications. It has true streaming model and does not take input data as batch or micro-batches.

Apache Flink was **founded by Data Artisans company** and is now developed under Apache License by Apache Flink Community. This community has **over 479 contributors and 15500 + commits** so far.



APACHE FLINK

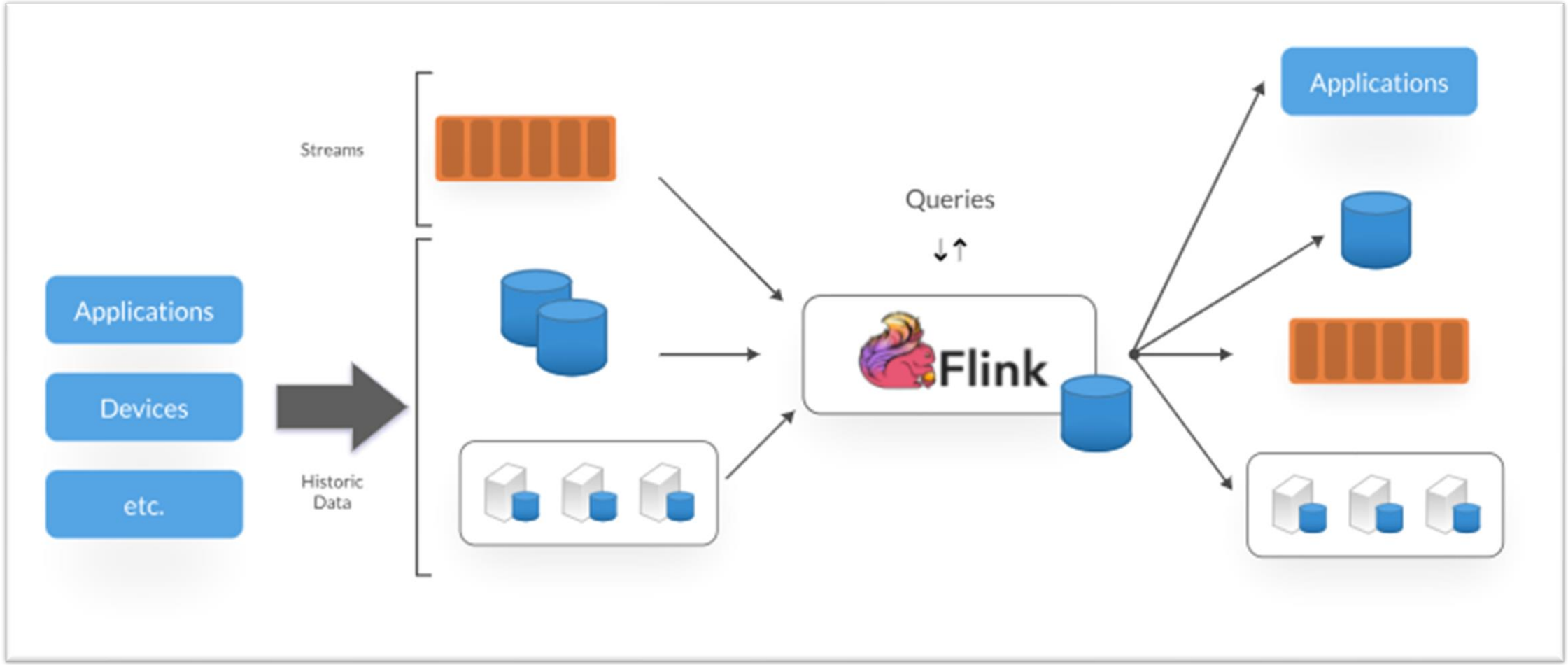
Flink is a fourth-generation data processing framework and is one of the more well-known Apache projects.

Flink supports batch and stream processing natively. It promotes continuous streaming where event computations are triggered as soon as the event is received.

Batch processing refers to performing computations on a fixed amount of data. This means that we already know the boundaries of the data and can view all the data before processing it, e.g., all the sales that happened in a week.

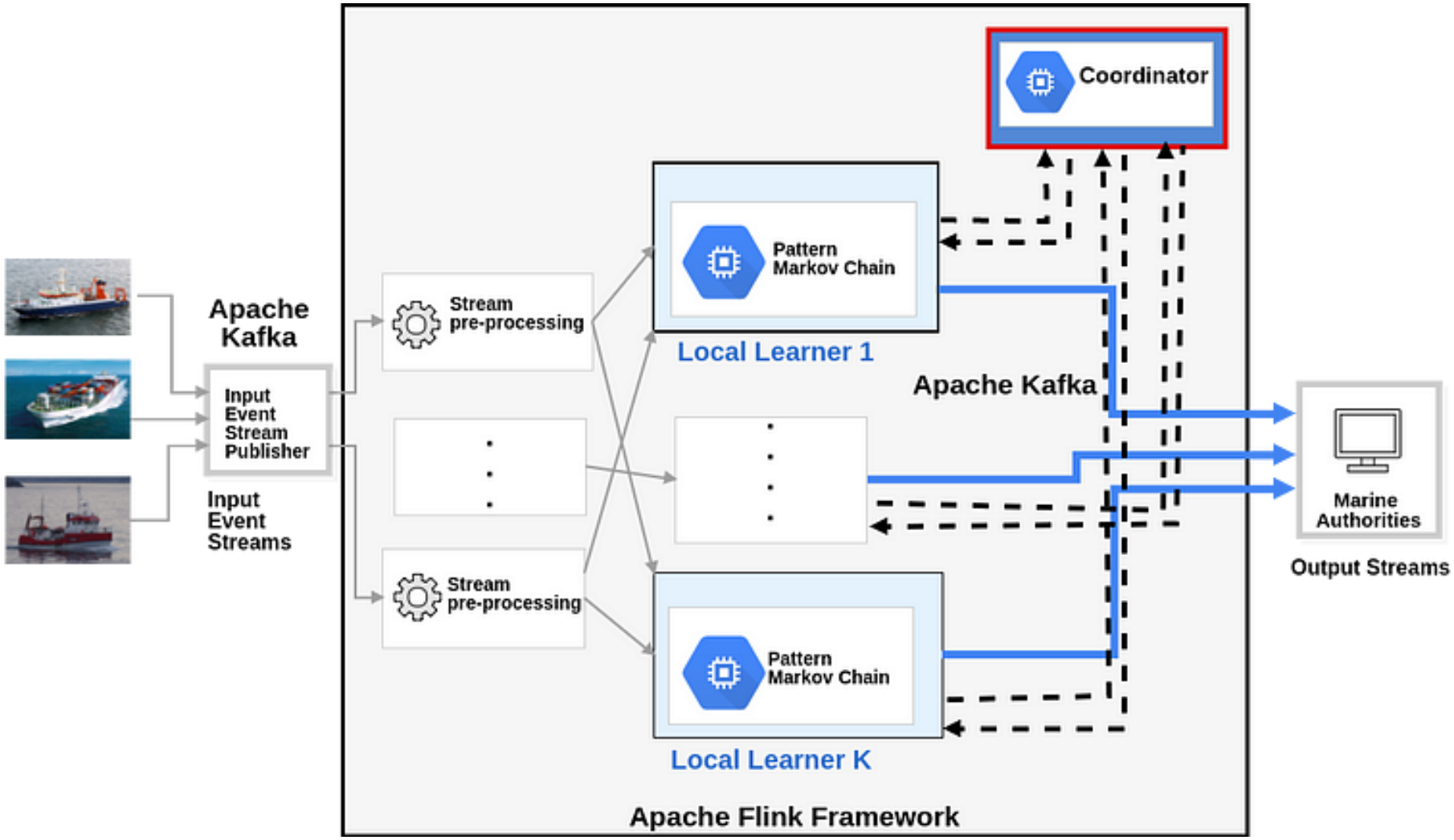
Stream processing is for "infinite" or unbounded data sets that are processed in real-time. Common use cases for stream processing include monitoring user activity, processing gameplay logs, and detecting fraudulent transactions.

APACHE FLINK



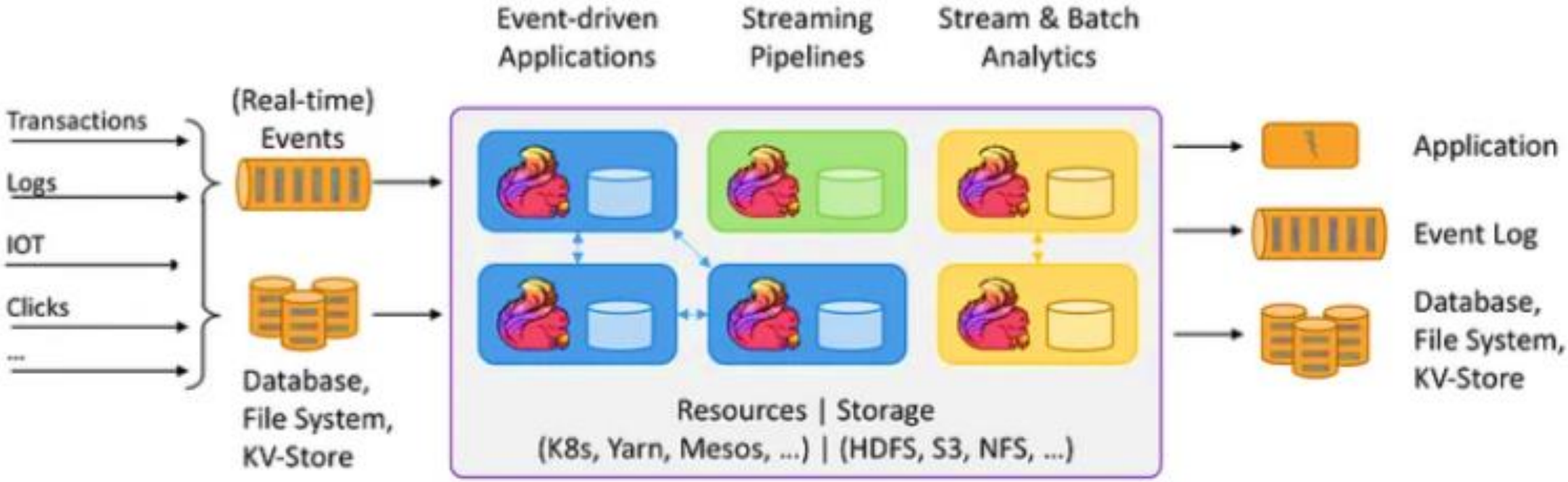
A high-level view of the Flink ecosystem

APACHE FLINK



A Distributed Online Learning Approach for Pattern Prediction over Movement Event Streams.

APACHE FLINK



Apache Flink® — Stateful Computations over Data Streams

APACHE SPARK VS FLINK

Data Processing	Batch/Stream (Micro Batch)	Batch/Stream (Native)
Iterations	No	Yes
SQL	Spark SQL	Table, SQL API
Fault tolerance	Yes (WAL)	Yes (Chandy-Lamport)
Optimization	Manual	Auto
Windowing	Timed	Time, Count
State Backend	HDFS	In-memory, file system, RocksDB
Language	Scala, Java, Python, R, C#, F#	Java, Scala, Python, SQL
Cost	Open source	Open source
Backpressure	Manual Configuration	Implicit, through architecture
Geo-Distribution	No	No
Latency	Seconds	Sub-second
Geo-fencing	No	No
Converged Platform	No	No

Flink optimizes jobs before execution on the streaming engine. The Flink optimizer is independent of the programming interface and works similarly to relational database optimizers by transparently applying optimizations to data flows.

APACHE SPARK VS FLINK

Iterative Processing

Spark offers iterative processing through its resilient distributed datasets (RDDs) and directed acyclic graph (DAG) execution model. Spark is well-suited for batch processing, but it can also handle iterative processing and streaming using micro-batching.

Flink was designed primarily for stream processing, with native support for iterative algorithms. Flink processes data using a continuous streaming model, offering lower latency and better handling of out-of-order events compared to Spark's micro-batching approach.

Fault Tolerance

Spark achieves fault tolerance through RDDs, which are immutable and partitioned data structures that can be recomputed in case of failures. Additionally, Spark stores lineage information to track dependencies and recover lost data.

Flink uses a distributed snapshot-based approach for fault tolerance, capturing the state of the application at specific checkpoints. This allows Flink to recover quickly and consistently from failures with minimal impact on performance.

Optimization

Spark employs the Catalyst optimizer, which is an extensible query optimizer for data transformation and processing. Spark also includes the Tungsten execution engine that optimizes the physical execution of operations for better performance.

Flink has a cost-based optimizer for batch processing, which analyzes the data flow and selects the most efficient execution plan based on available resources and data characteristics. Flink's stream processing also benefits from pipeline-based execution and low-latency scheduling.

APACHE SPARK VS FLINK

Windowing
Spark provides windowing functions for processing streaming data within fixed or sliding time windows. However, Spark's windowing is less flexible and efficient compared to Flink's, due to its reliance on micro-batching.

Flink has advanced support for windowing, including event-time and processing-time-based windows, session windows, and flexible custom window functions. Flink's windowing is more efficient and accurate for stream processing as it is designed specifically for continuous data streams.

Language Support
Spark supports multiple programming languages, such as Scala, Java, Python, and R. This broad language support makes Spark accessible to a wide range of developers and data scientists.

Flink also supports various programming languages, including Java, Scala, and Python. However, Flink's support for Python is less mature compared to Spark, which may limit its appeal to Python-centric data science teams.

Ecosystem and Community
Spark has a larger and more mature ecosystem, with a wide range of connectors, libraries, and tools available. This can make it easier to find resources, support, and third-party integrations for your project.

Flink, while growing in popularity, has a smaller ecosystem compared to Spark. However, it is continuously evolving and adding new features, making it a strong contender in the big data processing space.

APACHE SPARK VS FLINK – HOW TO CHOOSE

- Data processing requirements
- Performance
- Ease of use
- Integration with other tools
- Availability of resources:

APACHE SPARK VS FLINK – HOW TO CHOOSE

Conclusion

In conclusion, both Apache Spark and Apache Flink are powerful and versatile distributed data processing frameworks, each with its unique strengths and capabilities.

Spark excels in **batch processing** and offers mature support for various programming languages, making it suitable for a wide range of use cases. On the other hand, Flink **shines in stream processing**, providing low-latency performance and advanced windowing functions for real-time analytics.

The choice between Spark vs. Flink depends on your specific **use cases, requirements, and team expertise**

Contact Us

080-4524-9465

support@intellipaat.com