Spark has its own cluster management computation, it uses Hadoop for storage purpose only

Spark is designed to cover a wide range of workloads such as batch applications, iterative algorithms, interactive queries and streaming

**Iterative** Applications: Repetitive computations on the same data

Ex: Machine Learning Algorithms (e.g., Gradient Descent, K-Means Clustering)

**Interactive** Applications: On-demand queries and real-time data exploration

Ex: Ad Hoc Querying (one-time, **on-demand query** that is **not pre-defined** or scheduled)

* require faster data sharing across parallel jobs => Spark

**RDD**

Resilient Distributed Datasets

a fundamental data structure of Spark

**Key Features of RDDs:**

1. **Immutable** → Once created, they cannot be changed (only transformed into new RDDs).
2. **Distributed** → Data is split across multiple nodes in a cluster.
3. **Fault-Tolerant (Resilient)** → If a node fails, RDDs can be **recomputed** using lineage (history of transformations).
4. **Lazy Evaluation** → Transformations (like map, filter) are not executed immediately but only when an action (like collect, count) is triggered.
5. **In-Memory Processing** → RDDs store intermediate data in memory, making computations much faster than traditional disk-based frameworks like Hadoop MapReduce.

**RDD Operations:**

* **Transformations** (return new RDDs)
  + map(), filter(), flatMap(), groupByKey(), reduceByKey(), join()
* **Actions** (return final results)
  + collect(), count(), reduce(), take(), saveAsTextFile()

By default, **RDD transformations are lazy**, meaning Spark recomputes them every time you perform an action. To **avoid recomputation**, use **cache()** or **persist()**