BigData Analytics

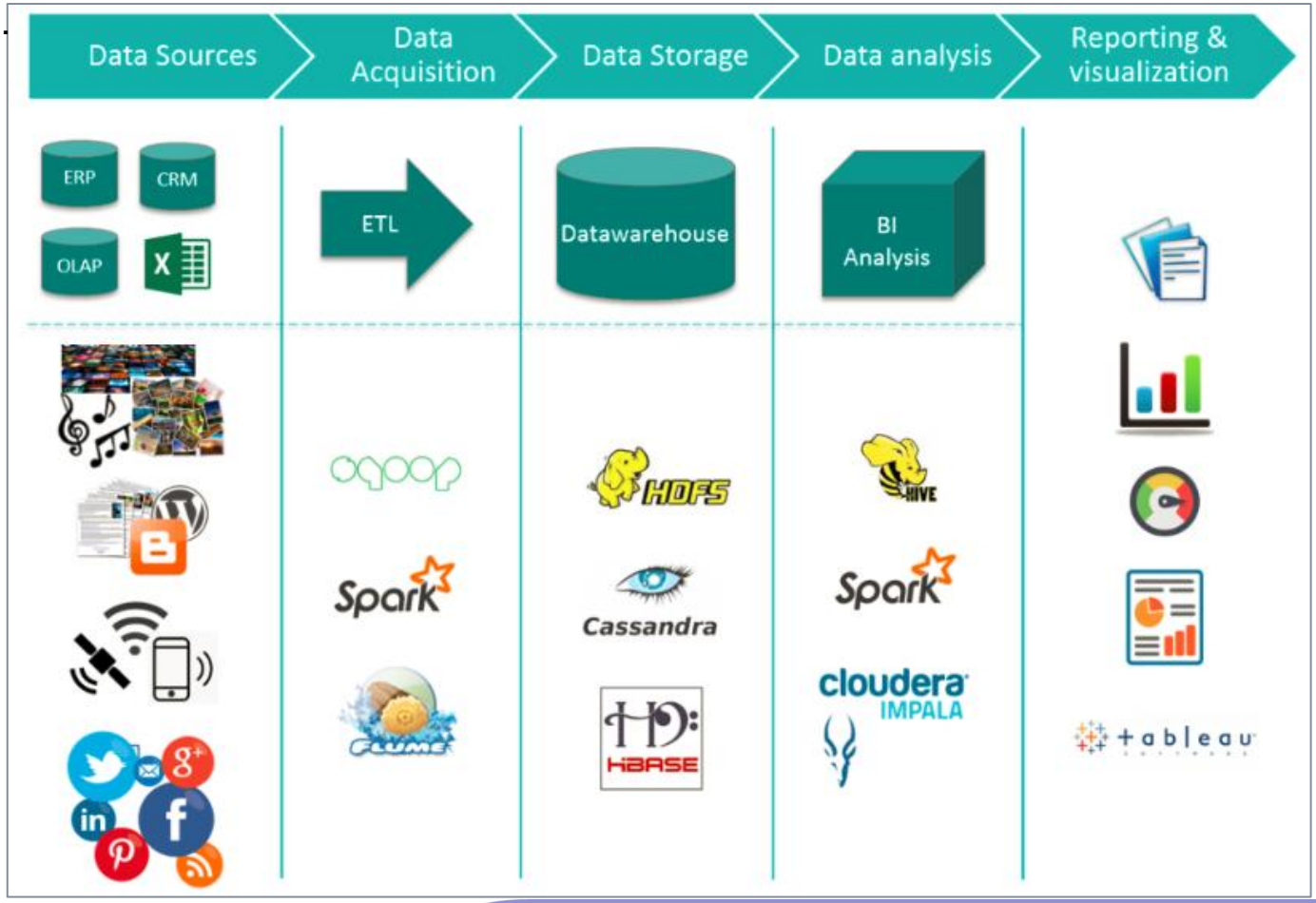
Ecosystem

sohailimran2@yahoo.com

Ref: Web

Intro

The big data ecosystem is a comprehensive **suite of technologies and tools** designed to handle the complexities of managing, processing, and analyzing vast amounts of data.
Central to this ecosystem are **distributed** storage systems like HDFS, scalable processing frameworks like Apache **Hadoop** and Apache **Spark**, real-time data streaming tools like Apache **Kafka**, and various data analytics and machine learning platforms.



Apache Hadoop is an **open-source** software framework for **distributed** storage and processing of big data sets. It uses a distributed computing model that allows for data processing across a **cluster** of computers. This framework is designed to handle structured and unstructured data, making it particularly useful for big data applications.

Traditional databases were not capable of handling the sheer volume and variety of data being generated by businesses. The significance of Apache Hadoop lies in its ability to provide a cost-effective solution for handling large amounts of data across different machines.

At its core, Apache Hadoop consists of three main components: the Hadoop Distributed File System (**HDFS**), **Yarn**, and **MapReduce**.

HDFS: is responsible for **storing** and managing large amounts of data across a cluster of machines.

MapReduce: is responsible for **processing** that data.

Yarn: is the **Resource Management** unit of Hadoop.

One key aspect of the architecture behind Apache Hadoop is its reliance on **commodity** hardware. By using widely available, inexpensive hardware components instead of specialized equipment, businesses can reduce costs while still achieving high performance. Another crucial aspect of the architecture is its focus on **fault tolerance**.

Apache Hadoop can replicate data across multiple nodes automatically, so there's no danger of losing important information if one node goes down.

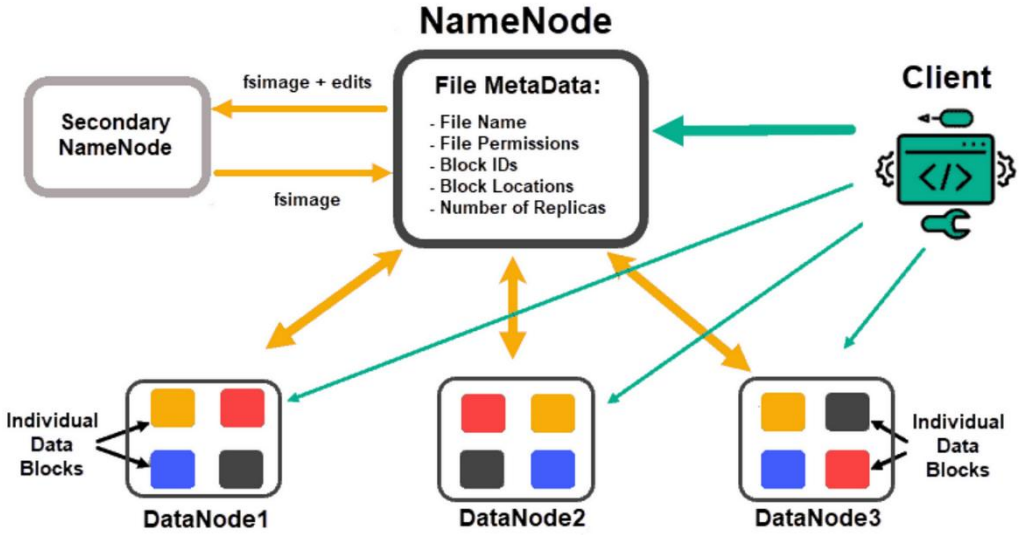
Designed to answer: "**How to process big data with reasonable cost and time?**".

hadoop HDFS

At its most basic level, HDFS allows users to store files that are too large to be stored on a single machine by breaking them up into smaller chunks and storing those chunks across multiple machines in a cluster. This approach provides several benefits over traditional file systems:

- **Scalability**: Because HDFS can store data across multiple machines, it can handle petabytes of data with ease.
- **Fault tolerance**: As mentioned earlier, because HDFS replicates data across multiple nodes, there's no risk of losing important information if one machine goes down. — High throughput: Because HDFS is designed to read and write large files quickly, it's well-suited for big data applications.

Architecture



HDFS architecture

Blocks

- Data stored in HDFS is split into **blocks** (by default, 128 MB each).
- Blocks are replicated across multiple DataNodes to ensure data availability and fault tolerance.

Namenode

- Maintaining the filesystem tree and **metadata**.
- Managing the **mapping** of file blocks to DataNodes.
- Ensuring data **integrity** and **coordinating replication** of data blocks.

Datanodes

- **Storing** data blocks and serving read/write requests from clients.
- **Performing** block creation, deletion, and replication upon instruction from the NameNode.
- Periodically **sending** block reports and **heartbeats** to the NameNode to confirm its **status**.

Secondary NameNode

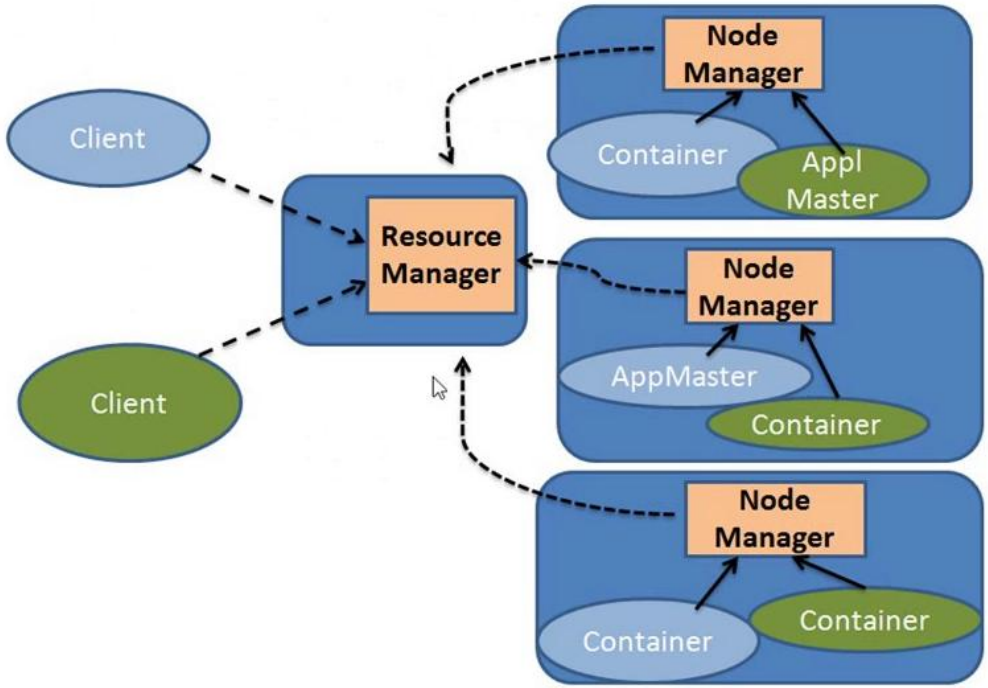
- **Merging EditLogs** with **FsImage** to create a new checkpoint.
- Helping to **manage** the NameNode's namespace metadata.

Replication

To ensure data reliability, HDFS replicates each block of data multiple times across different Datanodes (usually three copies by default). This replication ensures that even if one machine fails, the data is still available from the other machines.

Yet Another Resource Negotiator

The technology used for **job scheduling** and **resource management** and one of the main components in **Apache Hadoop**. This enables Hadoop to support different processing types. It runs **interactive** queries, **streaming** data, and **real-time** applications. Also, it supports a broader range of applications. Yarn **combines** a central resource manager with different **containers**. It can combine the resources dynamically to different applications, and the operations are **monitored** well.



✓Job tracker 1.0 responsibility is now split

- **Resource Manager** manages the resource allocation in the cluster
- **Application master** manages resource needs of individual applications

✓Node Manager is a generalized task tracker

✓A container executes an application specific process

YARN: major components

1. Resource Manager

- **Resource Management:** keeps an inventory of cluster-wide resources such as RAM, CPU, and Network usage.
- **Scheduling Management:** responsible for only allocating resources as requested by various Application Masters. It does not monitor or track the status of the job.
- **Application Management:** ensures the start-up and job completion. It monitors the activity and status of the Application Master and provides a restart in case of failure of the Application Master.
- **Containers in Hadoop:** It is a way to define requirements for memory, CPU, and network allocation by dividing the resources on the data server into a container.
- **Resource Containers:** The Resource Manager is responsible for scheduling resources by allocating containers.

2. Node Manager

The primary goal of the Node Manager is memory management. Node Manager tracks the usage and status of the cluster inventories, such as CPU, memory, and network on the local data server, and reports the status regularly to the Resource Manager. This holds the parallel programming in place.

3. Application Master

Responsible for the execution of parallel computing jobs. Its daemons are started by the resource manager at the start of a job. It negotiates with the scheduler function in the Resource Manager for the containers of resources throughout the cluster. The application master reports the job status both to the Resource Manager and the client.

The MapReduce processing framework is responsible for processing the data stored in HDFS. It does this by **breaking** down large datasets into **smaller chunks** and processing those chunks in parallel across a cluster of machines.

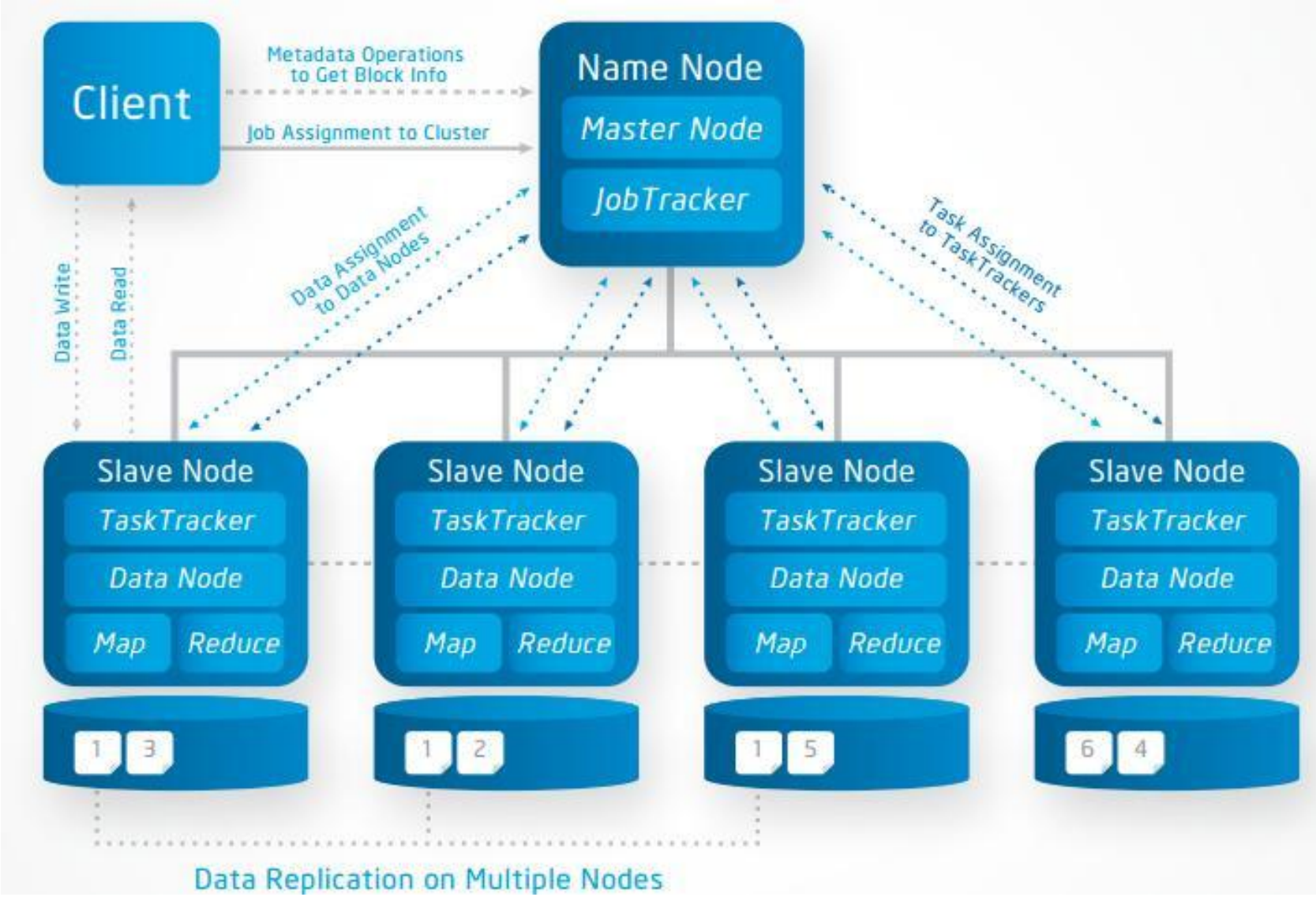
At its core, MapReduce consists of two main functions:

- **Map**: This function takes a dataset and breaks it down into smaller sub-datasets that can be processed in parallel.
- **Reduce**: This function takes the output from the Map function and combines it into a single result. Together, these functions allow businesses to process vast amounts of structured or unstructured data quickly and efficiently.

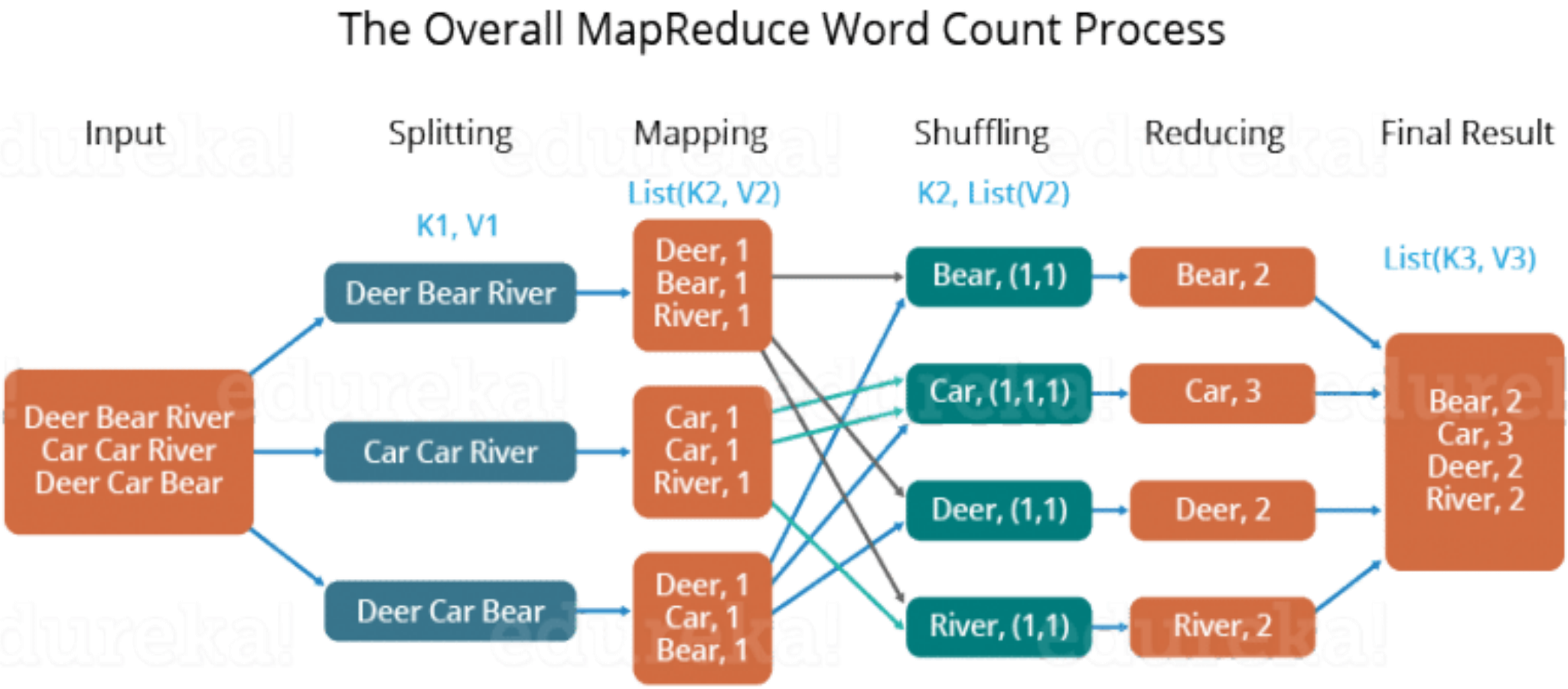
MapReduce Engine

- **JobTracker** & **TaskTracker**
- JobTracker **splits** up data into smaller tasks("**Map**") and **sends** it to the TaskTracker process in each node.
- TaskTracker **reports back** to the JobTracker node and reports on **job progress**, sends data ("Reduce"), or requests new jobs.

MapReduce: Processing

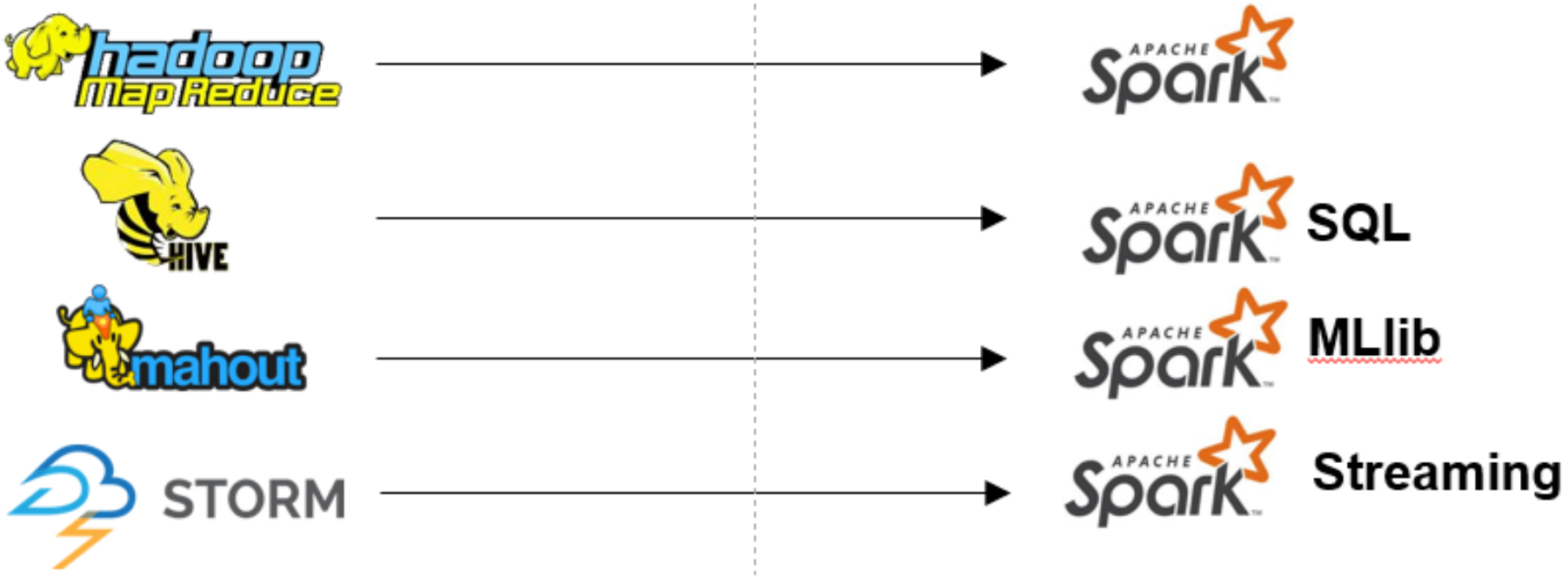


MapReduce: Example

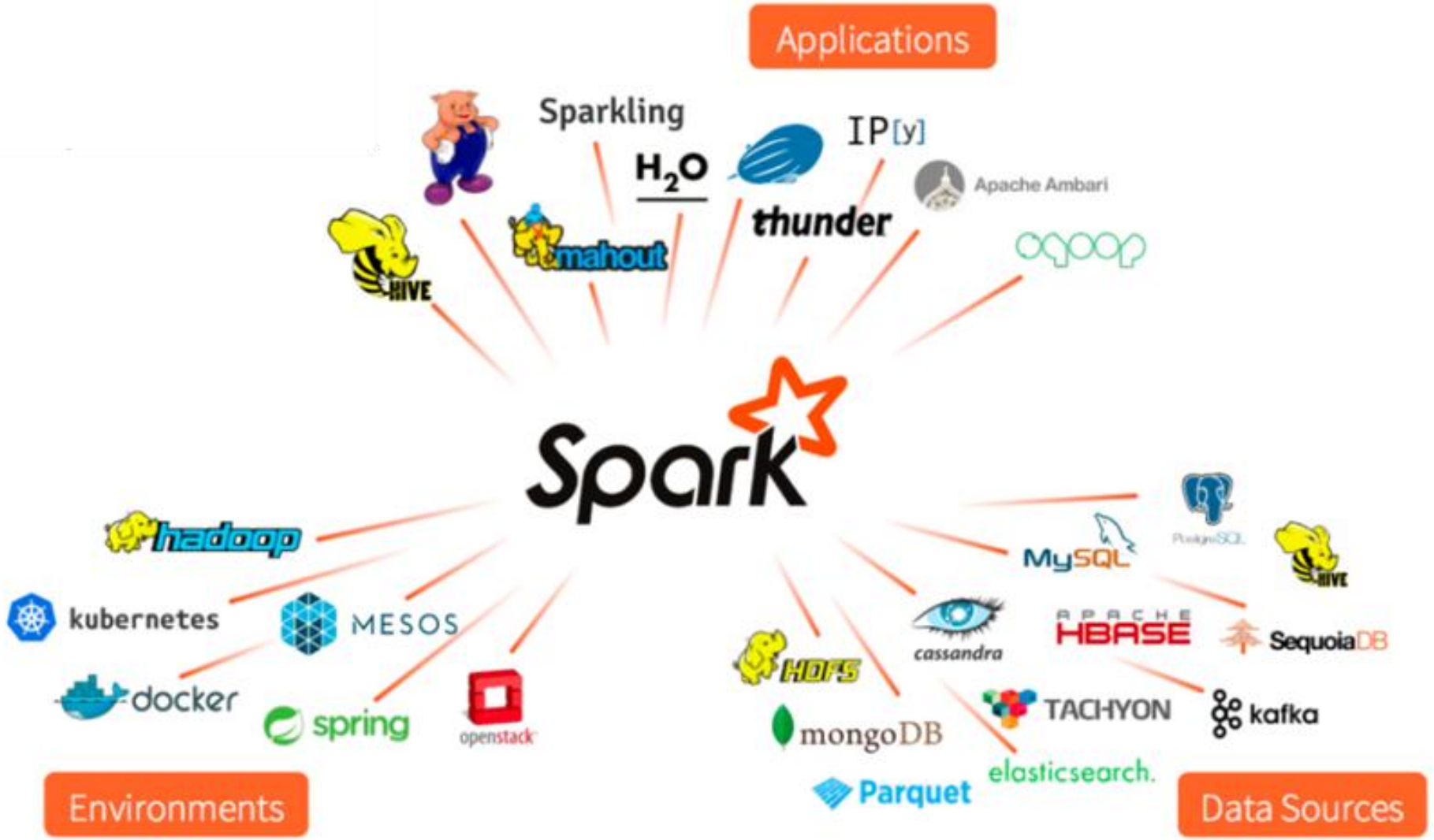


Spark is an Apache project advertised as “**lightning-fast cluster computing**”. It has a thriving open-source community and is the most active Apache project at the moment.
The Spark core is complemented by a set of powerful, higher-level libraries that can be seamlessly used in the same application. These libraries currently include **SparkSQL**, Spark **Streaming**, **MLlib**, and **GraphX**, each of which is further detailed in this article. Additional Spark libraries and extensions are currently under development as well.

Apache Spark: a unified API



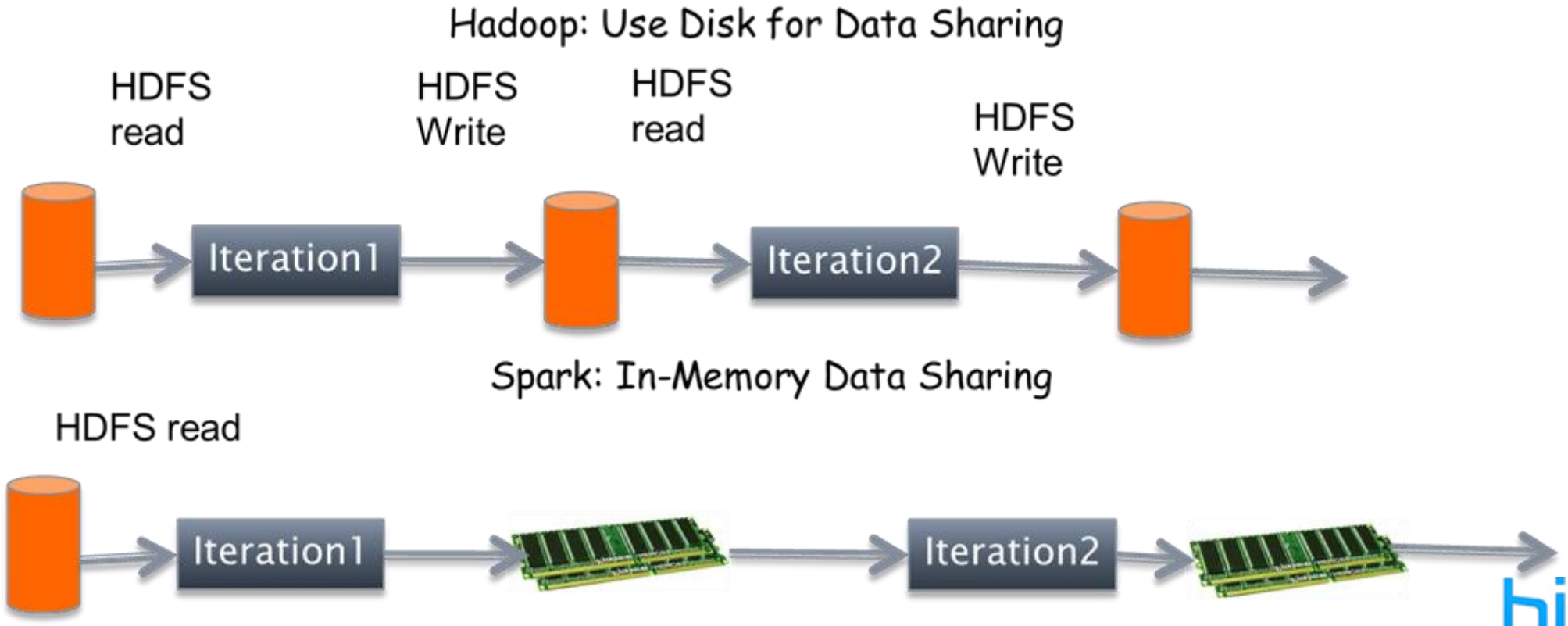
Spark Integration



Spark Ecosystem

Spark uses Memory instead of Disk

Spark provides a faster and more general data processing platform.
Spark lets you run programs up to **100x** faster in memory, or **10x** faster on disk, **than Hadoop**.
Spark and Hadoop serve different niches within the big data processing landscape. Spark is often preferred when low-latency and iterative processing are crucial, while Hadoop is still widely used for batch processing and storage of massive datasets.

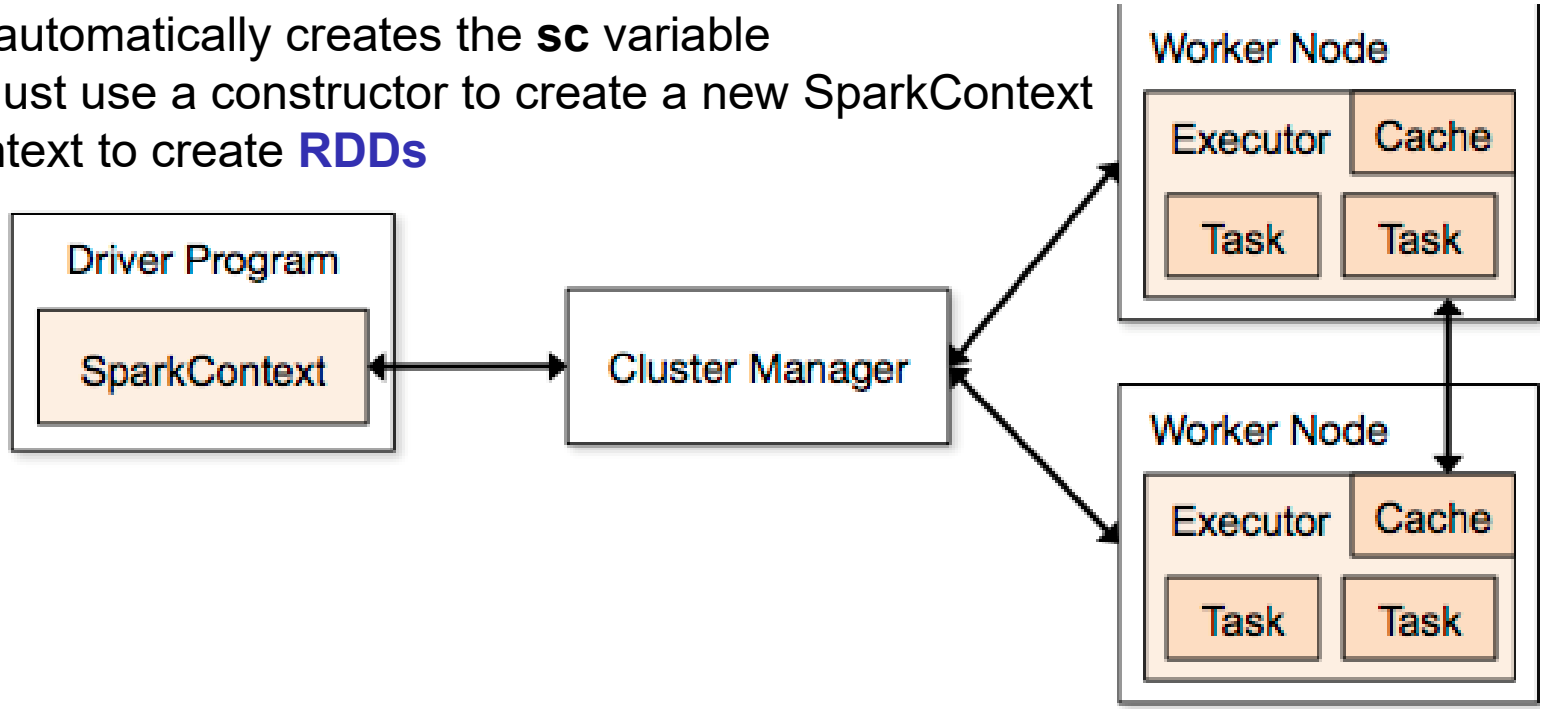


Spark Context

SparkContext is the **object** that manages the **connection** to the clusters in Spark and coordinates running processes on the clusters themselves. SparkContext connects to cluster managers, which manage the actual executors that run the specific computations.

The SparkContext object is usually referenced as the variable **sc**.

- A Spark program first creates a SparkContext object
  - » Tells Spark how and where to access a cluster
  - » Spark shell automatically creates the **sc** variable
  - » Programs must use a constructor to create a new SparkContext
- Use SparkContext to create **RDDs**



Spark Core

Spark Core is the base engine for large-scale parallel and distributed data processing. It is responsible for:

- **memory management** and **fault recovery**
- **scheduling**, **distributing**, and **monitoring** jobs on a cluster
- **interacting** with storage systems

Spark introduces the concept of an RDD (**Resilient Distributed Dataset**), an immutable, fault-tolerant, distributed collection of objects that can be operated on in **parallel**. An RDD can contain any type of object and is created by loading an external dataset or distributing a collection from the driver program.

RDDs support two types of operations:

Transformations are operations (such as **map**, **filter**, **join**, **union**, and so on) that are performed on an RDD and which yield a new RDD containing the result.

Actions are operations (such as **reduce**, **count**, **first**, and so on) that return a value after running a computation on an RDD.

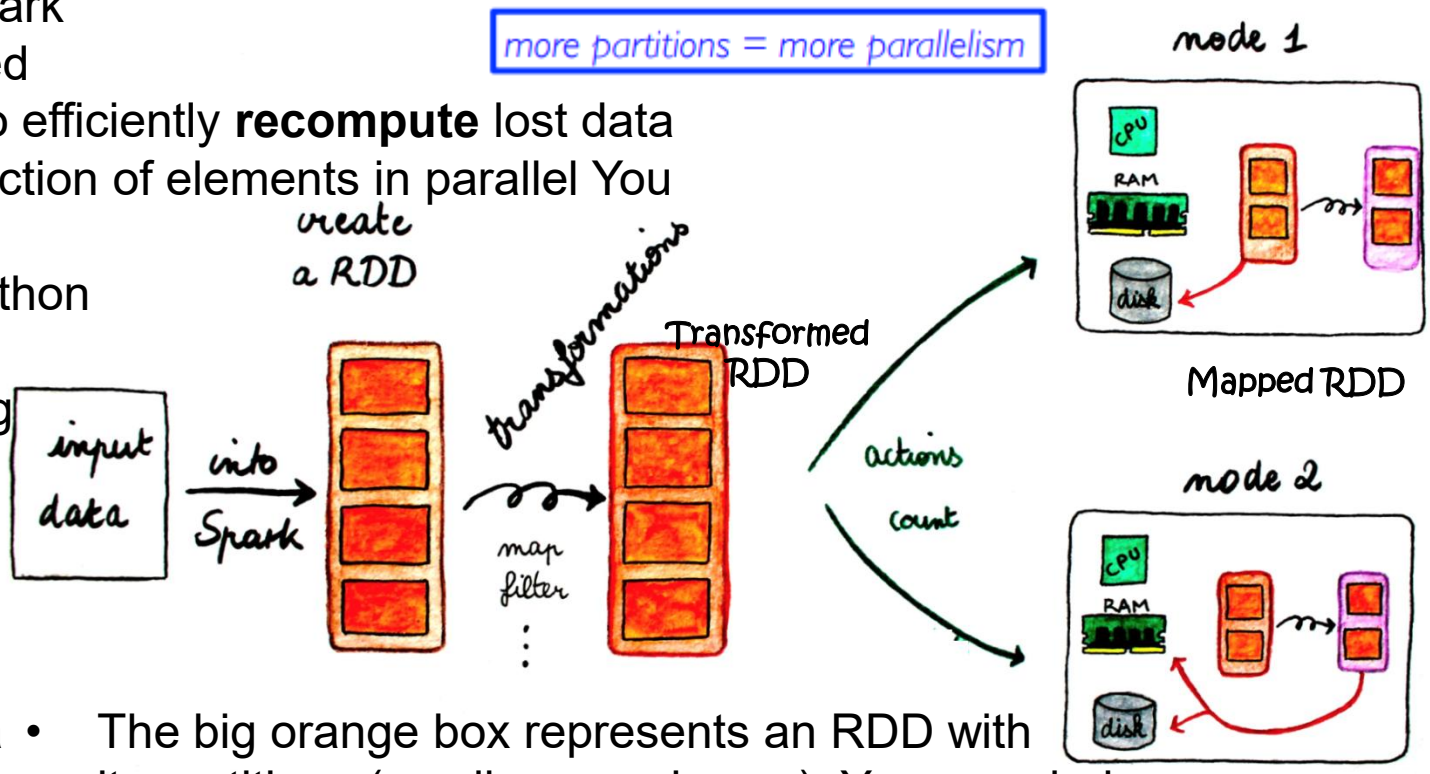
Transformations in Spark are "**lazy**", meaning that they **do not compute** their results right away. Instead, they just "remember" the **operation to be performed** and the dataset (e.g., **file**) to which the operation is to be performed. The transformations are only actually **computed when an action is called,** and the result is returned to the driver program. This design enables Spark to run more efficiently.

For example, if a big file was transformed in various ways and passed to the first action, Spark would **only process and return the result** for the first line, rather than do the work for the entire file.

By default, each transformed RDD may be **recomputed each time** you run an action on it. However, you may also persist an RDD in memory using the persist or cache method, in which case Spark will keep the elements around on the cluster for much faster access the next time you query it.

Operation on Resilient Distributed Datasets

The primary abstraction in Spark
» **Immutable** once constructed
» **Track** lineage information to efficiently **recompute** lost data
» Enable **operations** on collection of elements in parallel You construct RDDs
» by **parallelizing** existing Python collections (lists)
» by **transforming** an existing RDDs
» from files in HDFS or any other **storage** system



- To the left, the **input data** comes from an external storage. The data are loaded into Spark and an Resilient Distributed Dataset (RDD) is created.
- The big orange box represents an RDD with its partitions (small orange boxes). You can chain **transformations** on RDDs. As the transformations are lazy, the partitions will be sent across the nodes of the cluster when you will call an **action** on the RDD.
- Once a partition is located on a node, you can continue to operate on it.

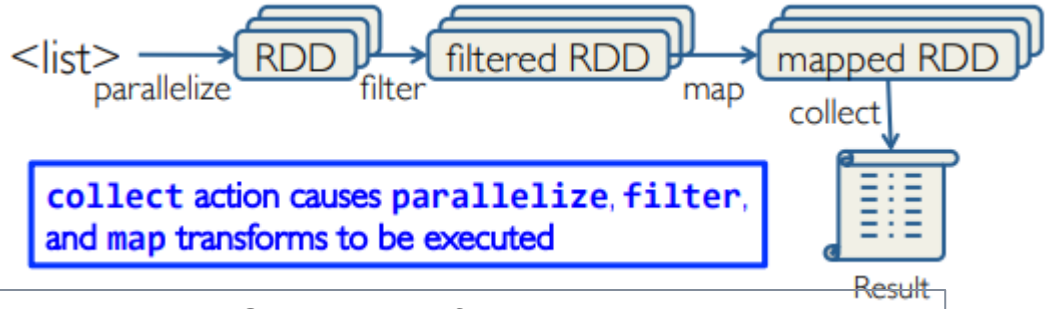
Working with RDDs

Create an RDD from a data source:

<list>

Programmer specifies number of partitions for an RDD

- Apply transformations to an RDD: map filter
- Apply actions to an RDD: collect count



Some Transformations

Transformation	Description
map(func)	return a new distributed dataset formed by passing each element of the source through a function func
filter(func)	return a new dataset formed by selecting those elements of the source on which func returns true
distinct ([numTasks]))	return a new dataset that contains the distinct elements of the source dataset
flatMap(func)	similar to map, but each input item can be mapped to 0 or more output items (so func should return a Seq rather than a single item)0

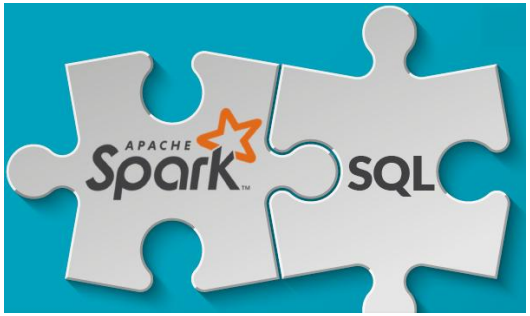
Some Key-Value Transformations

Key-Value Transformation	Description
reduceByKey(func)	return a new distributed dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function func, which must be of type (V,V) -> V
sortByKey()	return a new dataset (K, V) pairs sorted by keys in ascending order
groupByKey()	return a new dataset of (K, Iterable) pairs

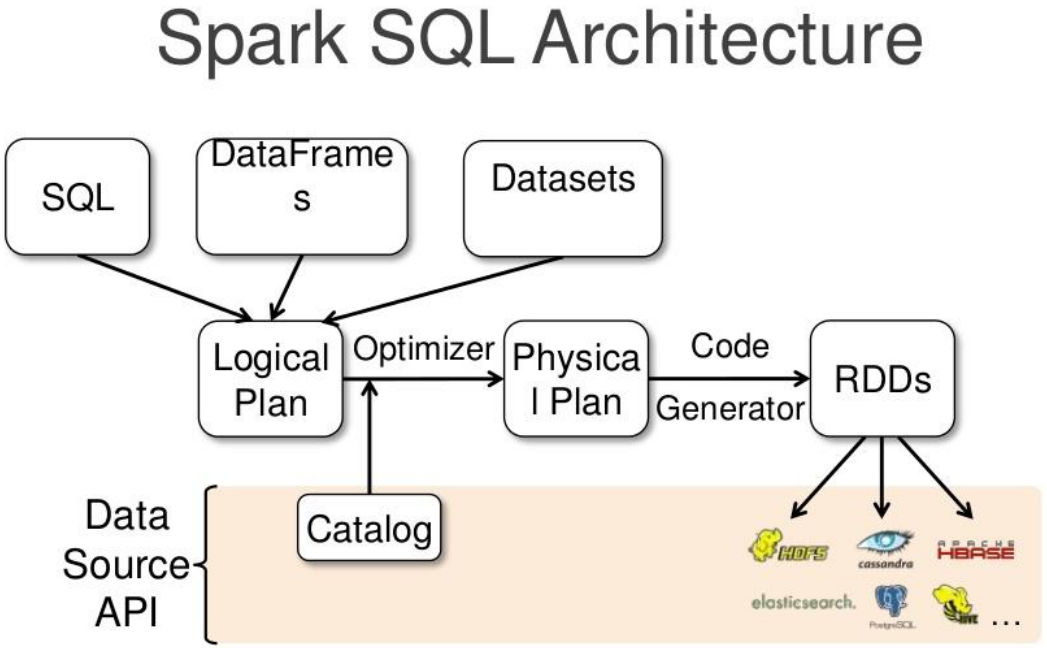
Some Actions

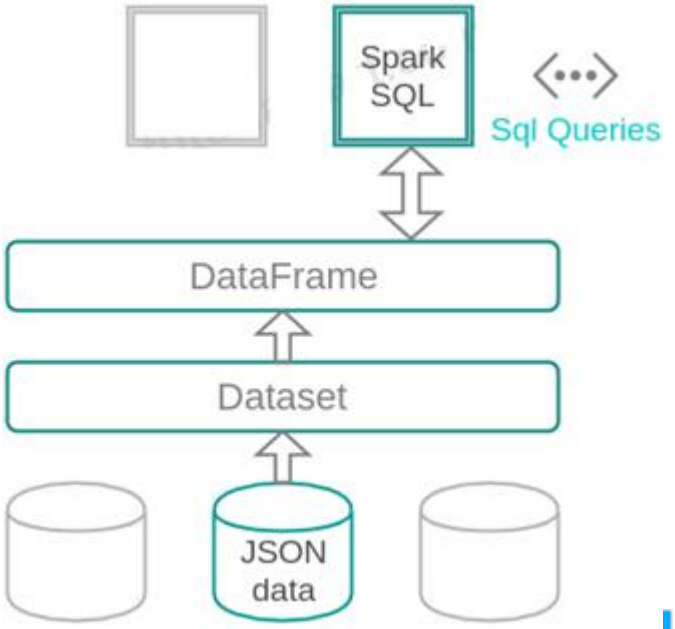
Action	Description
reduce(func)	aggregate dataset's elements using function func. func takes two arguments and returns one, and is commutative and associative so that it can be computed correctly in parallel
take(n)	return an array with the first n elements
collect()	return all the elements as an array WARNING: make sure will fit in driver program
takeOrdered (n, key=func)	return n elements ordered in ascending order or as specified by the optional key function

Spark SQL



SparkSQL is a Spark component that supports querying data either via **SQL** or via the **Hive Query Language**. It originated as the Apache Hive port to run on top of Spark (in place of MapReduce) and is now integrated with the Spark stack. In addition to providing support for various data sources, it makes it possible to weave SQL queries with code transformations which results in a very powerful tool.





Streaming analytics with Spark Streaming

The Spark Streaming API closely matches that of the Spark Core, making it easy for programmers to work in the worlds of both batch and streaming data.
Spark Streaming supports real time processing of streaming data, such as production web server log files (e.g. Apache Flume and HDFS/S3), social media like Twitter, and various messaging queues like Kafka. Under the hood, Spark Streaming receives the input data streams and divides the data into batches.

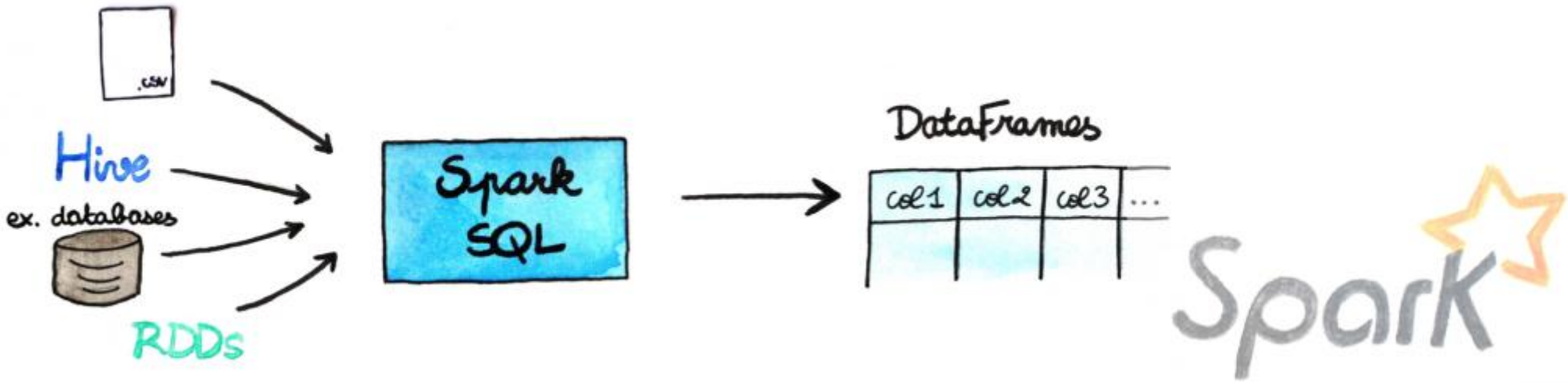
Spark Streaming receives live input data streams and divides the data into batches, which are then processed by the Spark engine to generate the final stream of results in batches.

The abstraction, which represents a continuous stream of data is the DStream (discretized stream).

Structured data with the DataFrame

A DataFrame is a distributed collection of data organized into named columns. It is conceptually equivalent to a table in a relational database or a data frame in R/Python, but with richer optimizations under the hood.
The Spark SQL/dataframe component provides a SQL-like interface. So, you can apply SQL-like queries directly on the RDDs.

DataFrames can be constructed from different sources such as structured data files, tables in Hive, external databases, or existing RDDs.



Spark MLlib

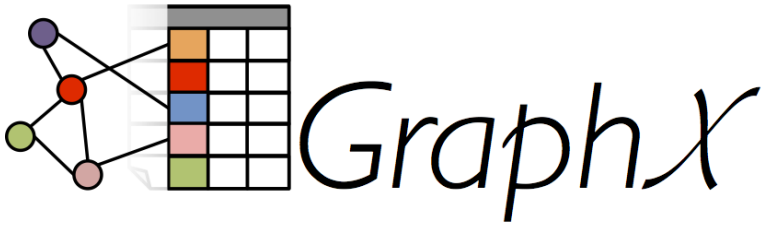
Spark MLLib is one of the elements of the Apache Spark framework and it utilizes all of its advantages. It allows to apply machine learning to large data sets with no scalability concerns. The system has dozens of built-in machine learning algorithms which may be applied depending on a particular business case. These include:

- **Classification**: logistic regression, naive Bayes
- **Regression**: generalized linear regression, isotonic regression
- **Decision trees**: random forests, and gradient-boosted trees
- **Recommendation**: alternating least squares (ALS)
- **Clustering**: K-means, Gaussian mixtures (GMMs)
- **Topic modeling**: latent Dirichlet allocation (LDA)
- **Feature transformations**: standardization, normalization, hashing
- Model **evaluation** and hyper-parameter **tuning**
- **ML Pipeline** construction
- **ML persistence**: saving and loading models and pipelines
- **Survival analysis**: accelerated failure time model
- Frequent itemset and sequential **pattern mining**: FP-growth, association rules, PrefixSpan
- **Distributed linear algebra**: singular value decomposition (SVD), principal component analysis (PCA)
- **Statistics**: summary statistics, hypothesis testing

Spark GraphX

GraphX is a library for manipulating graphs and performing graph-parallel operations.
It provides a uniform tool for ETL, exploratory analysis and iterative graph computations.
At a high-level, GraphX extends the Spark RDD abstraction by introducing the Resilient Distributed Property Graph: a directed multigraph with properties attached to each vertex and edge. To support graph computation, GraphX exposes a set of fundamental operators (e.g., subgraph, joinVertices, and mapReduceTriplets) as well as an optimized variant of the Pregel API. In addition, GraphX includes a growing collection of graph algorithms and builders to simplify graph analytics tasks.
Apart from built-in operations for graph manipulation, it provides a library of common graph algorithms such as PageRank.

GraphX library provides graph operators like **subgraph**, **joinVertices**, and **aggregateMessages** to transform the graph data. It provides several ways of building a graph from a collection of **vertices** and **edges** in an **RDD** or on **disk**. GraphX also includes a number of graph **algorithms** and **builders** to perform graph analytics tasks.



Hive is a **data warehouse** tool optimized for analytics.

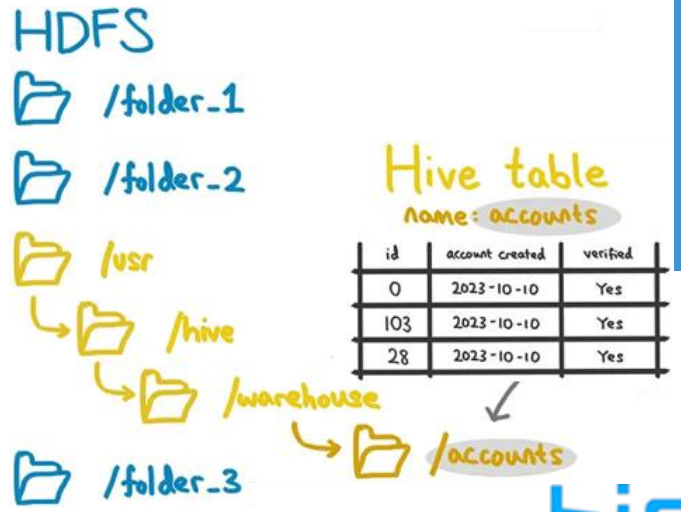
Hive sits on top of a data platform compatible with Hadoop (HDFS/Amazon's S3).

By 'sit', it means that the data accessed by Hive is stored in **HDFS**.

Types of Hive Tables

Depending on your needs for a Hive table, there are different categories to choose from:

- **Transactional:** Able to **perform** ACID/CRUD operations
- **Non-Transactional:** **Not** able to **perform** ACID/CRUD operations
- **Managed:** Stored in Hive-created subfolders, namely /usr/hive/warehouse/* (still stored in HDFS). This location means they're under the Hive Metastore, giving you much more control: This includes ACID operability and the ability to delete tables.
- **External:** Stored in the HDFS file system, but not within the Hive subfolders. Because this is not under the Hive metastore, these tables don't have ACID operations. The tables also require additional settings to be properly deleted.



Note: There are different folders you can store your Hive tables in. These different options exist since there are different types of Hive tables.

HBase is an open-source **non-relational distributed database** modeled after Google's Bigtable, providing a fault-tolerant way of storing large quantities of sparse data, which is suitable for use cases where low-latency random access and massive storage up to multi-PetaBytes are both required.

HBase employs a shared-storage architecture, separating the computation layer from the storage layer, which typically resides on **HDFS**. This architecture choice has enabled attracting features like smooth scaling-up and cost optimization etc. Here are some use cases where HBase may be a good candidate to consider.

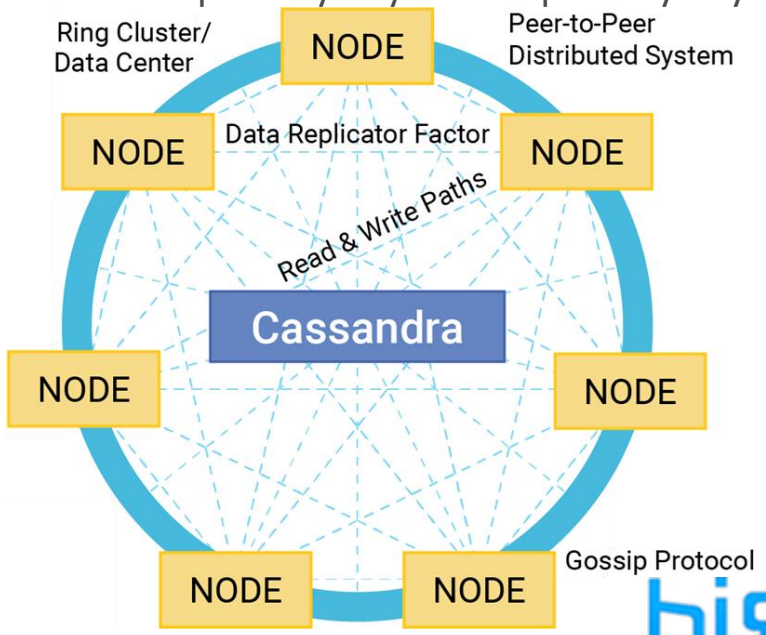
Tables in HBase are semi-structured, which means the **column families** must be defined before use. A **data cell** is uniquely identified by the specific **rowkey**, the specific column family, the exact **column qualifier**, and then the **timestamp** (for a specific version in the multi-versioned cells). Data is **sparsely** stored in HBase, and missing columns take no storage space at all.

HBase supports **multi-versioned data**, meaning that "a cell" actually consists of multiple cells, one per version, ordered by timestamp descending.

	emp_contact_data				emp_professional_data	
row key	first_name	last_name	city	county	designation	salary
1	James	Butt	New Orleans	Orleans	manager	100000
2	Josephine	Darakjy	Brighton	Livingston	associate	50000
3	Art	Venere	Bridgeport	Gloucester	developer	60000
4	Lenna	Paprocki	Anchorage	Anchorage	Sr Developer	65000
5	Simona	Morasca	Ashland	Ashland	executive	45000

The architecture of Apache Cassandra generally looks like this:

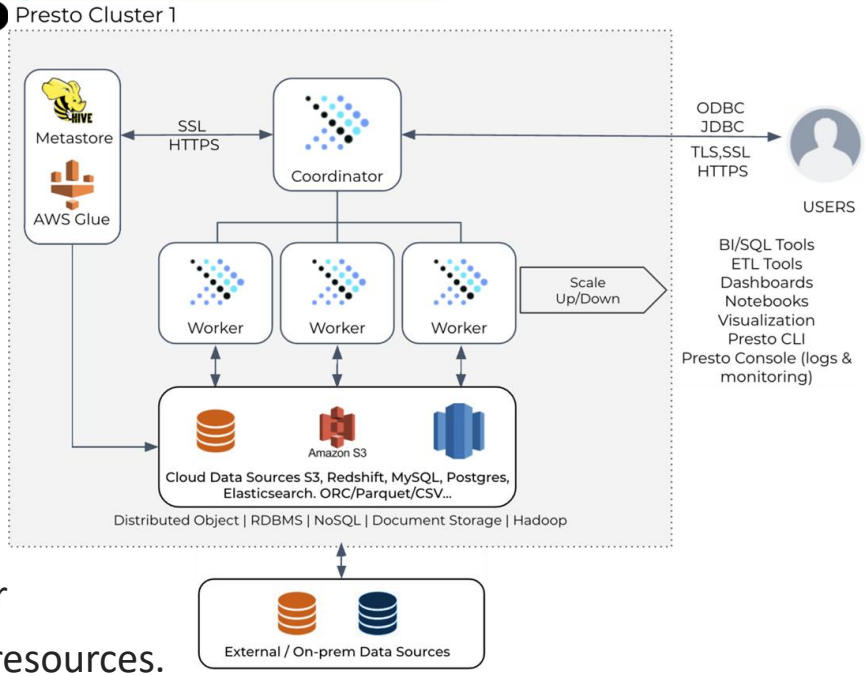
- **Peer-to-peer distributed system.** All Cassandra nodes in the cluster are equal and communicate with each other without the need for a centralized coordinator, avoiding a single point of failure.
- **Ring clusters.** Apache Cassandra design architectures distribute data across the cluster in a ring, with each of the nodes in a cluster responsible for a range of data determined by a hashing algorithm.
- **Data replication factor.** Cassandra replicates data files across several nodes and can be configured within a single data center or across multiple sites.
- **Data model.** Cassandra is schema-optional, using a partitioned row store data model that is flexible. In this model, data is stored and organized into rows based on a partition key—a subset of the primary key or the primary key itself if it's simple.
- **Read and write paths.** Cassandra supports fast write and read operations by employing a distributed commit log and memtable for write requests, and a distributed read operation across replicas for reads.
- **Gossip protocol.** Cassandra's gossip protocol for internode communication allows nodes to discover and share information about the cluster's state and network topology.



presto

Presto is an open-source distributed **SQL query engine for big data** created by Facebook.

- It is developed for **fast analytic queries on large datasets.**
- SQL query engine → **You can use SQL syntax!**
- It is designed for high performance
- It can query data from various sources, such as Hadoop Distributed File System (HDFS), relational databases, etc.



Key features :

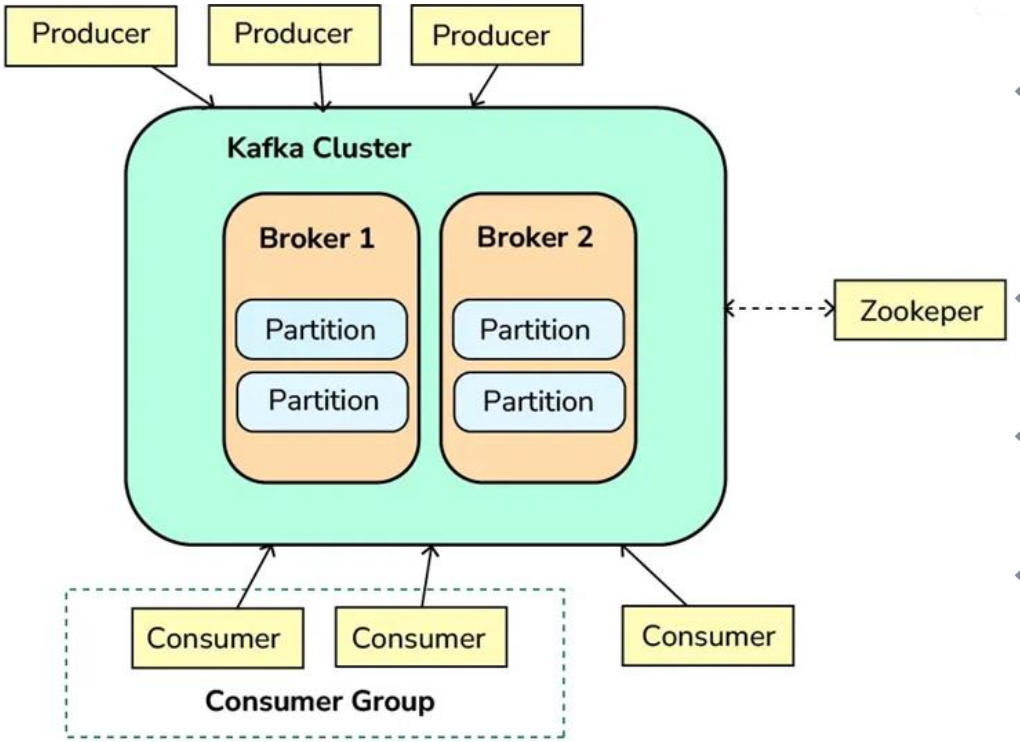
1. **Distributed Architecture:** It is designed to run queries on a cluster of machines, allowing for parallel processing and efficient use of resources.
1. **Versatile** : It can query data from multiple sources, including Hadoop, relational databases (like MySQL, PostgreSQL), and other data stores. It provides a unified interface for querying diverse datasets.
2. **SQL Compatibility**: It supports SQL syntax, making it familiar to users who are accustomed to relational databases.
3. **High Performance**: It is optimized for low-latency interactive queries, making it suitable for ad-hoc data analysis and business intelligence.
4. **Complex queries, joins, subqueries** : It supports complex queries, joins, subqueries, and aggregations, making it suitable for a wide range of analytical tasks.

APACHE DRILL

Apache Drill is an innovative **schema-free SQL query engine** for Hadoop, NoSQL, and cloud storage. This open-source software enables users to analyze large-scale datasets from numerous sources directly, without needing to shift data across systems. Apache Drill provides the **flexibility** of on-the-fly data discovery, making it significantly adaptable to changing data formats and structures.

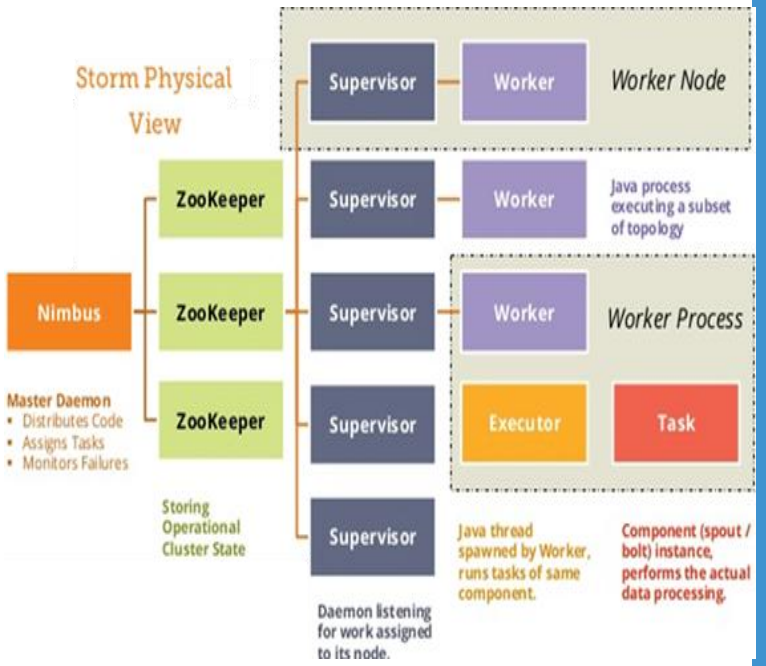
Drill supports a variety of NoSQL databases and file systems, including HBase, MongoDB, MapR-DB, HDFS, MapR-FS, Amazon S3, Azure Blob Storage, Google Cloud Storage, Swift, NAS, and local files. A single query can join data from multiple datastores. For example, you can join a user profile collection in MongoDB with a directory of event logs in Hadoop.

Drill's datastore-aware optimizer automatically restructures a query plan to leverage the datastore's internal processing capabilities. In addition, Drill supports data locality, so it's a good idea to co-locate Drill and the datastore on the same nodes.



- Kafka is a messaging system that is designed to be fast, scalable, and durable.
- A **Producer** is an entity/application that publishes data to a Kafka cluster, which is made up of **brokers.**
- A **Broker** is responsible for receiving and storing the data when a producer publishes.
- A **Consumer** then consumes data from a broker at a specified offset, i.e., position.
- A **Topic** is a category/feed name to which records are stored and published. Topics have partitions and order guaranteed per partition.
- All Kafka **records** are organized into topics. Producer applications write data to topics, and consumer applications read from topics.

Apache Storm is a processing engine in **big data** used for **real-time** analytics and computation. It is an easily available, open-source, and distributed data framework.



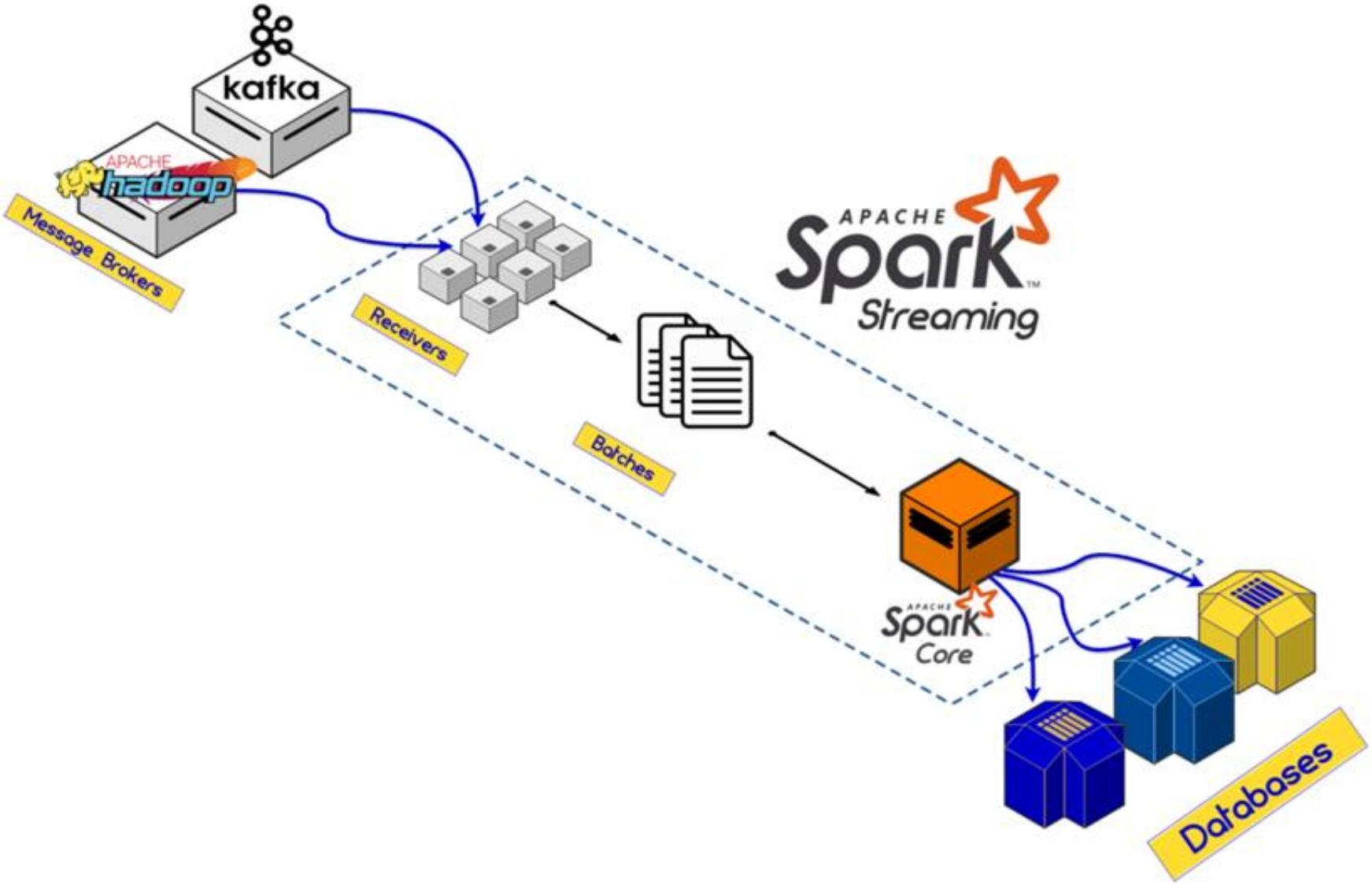
Components of Apache Storm

Topology is the real-time computational and graphical representation data structure. The topology consists of bolts and spouts, where a spout determines how the output is fixed to the inputs of bolts, and output from a single bolt is linked to the inputs of other bolts. A storm cluster gets input as topology, the nimbus daemon in the master node seeks information from the supervisor daemon in the Worker node, and accepts the topology.

Spout acts as an initial point-step in topology, and data from unlike sources is acquired by the spout. It ingests the data as a stream of tuples and sends it to Bolt for processing of stream as data. A single spout can generate multiple outputs of streams as tuples, these tuples of streams are further consumed by one or many bolts. Spout gets data from various databases, file system distribution or messages like Kafka consistently, converts them in streams of tuples and sends them to bolts for processing.

Bolts are responsible for the processing of data, their work includes filtering, functioning, aggregations, & handing databases, etc. Bolts consume multiple streams as input, process them, & generate new streams for processing of data.

Streaming


kafka
APACHE
hadoop
Message Brokers
Receivers
Batches
APACHE
Spark
Streaming
APACHE
Spark
Core
Databases

Real-Time Dashboard Reference Architecture