Leandro Batista de Almeida

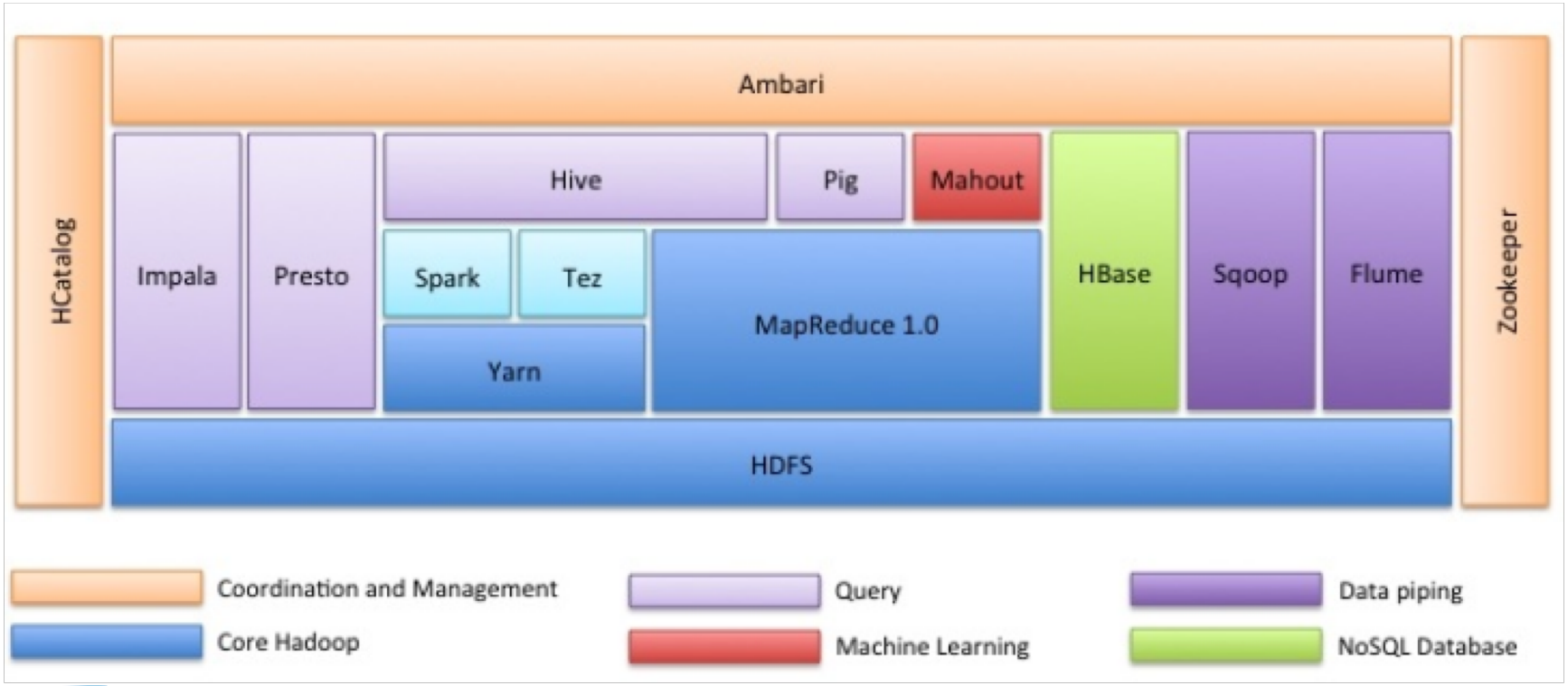
Anthony Ventresque

Spark - Concepts

School of Computer Science, UCD

Scoil na Ríomheolaíochta, UCD

Spark and Hadoop Ecosystem



Big Data Programming

- Hadoop used extensively by industry
  - Simple programming model (MapReduce)
  - Scalable, flexible, fault-tolerant, cost-effective
- But some concerns about maintain speed in processing large datasets
  - Waiting time between queries
  - Waiting time to run the programs
  - Google moved from MapReduce to BigTable for this reason

Spark

- Introduced for speeding up Hadoop process
  - Developed in 2009, in UC Berkeley's AMPLab
    - By Matei Zaharia
    - https://amplab.cs.berkeley.edu/
    - One of the Hadoop's sub projects
    - Open-sourced in 2010 under BSD license
    - Donated to Apache Software Foundation in 2013
      - Top-level Apache project from 2014

Spark – Seminal Papers

- Spark: Cluster Computing with Working Sets
  - Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica
  - University of California, Berkeley
- Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing
  - Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, Ion Stoica
  - University of California, Berkeley

What is Spark?

- Fast cluster computing technology, designed for fast computation
  - "lightning-fast"
- Based on Hadoop MapReduce
  - Extends MapReduce model for efficiency
    - Interactive queries, stream processing, etc
- Main feature is "in-memory cluster computing"
- Designed to cover a wide range of workloads
  - Batch applications, interactive algorithms, interactive queries, streaming, machine learning
  - Reduces the management burden of maintaining several separate tools

What Spark is NOT?

- Spark IS NOT a modified version of Hadoop
  - Actually, is not dependent on Hadoop
    - Has its own cluster management
      - Based on Mesos
    - Can be implemented in other clusters managers
- Can use Hadoop (or parts of) in two ways
  - Storage (HDFS)
  - Processing (YARN)

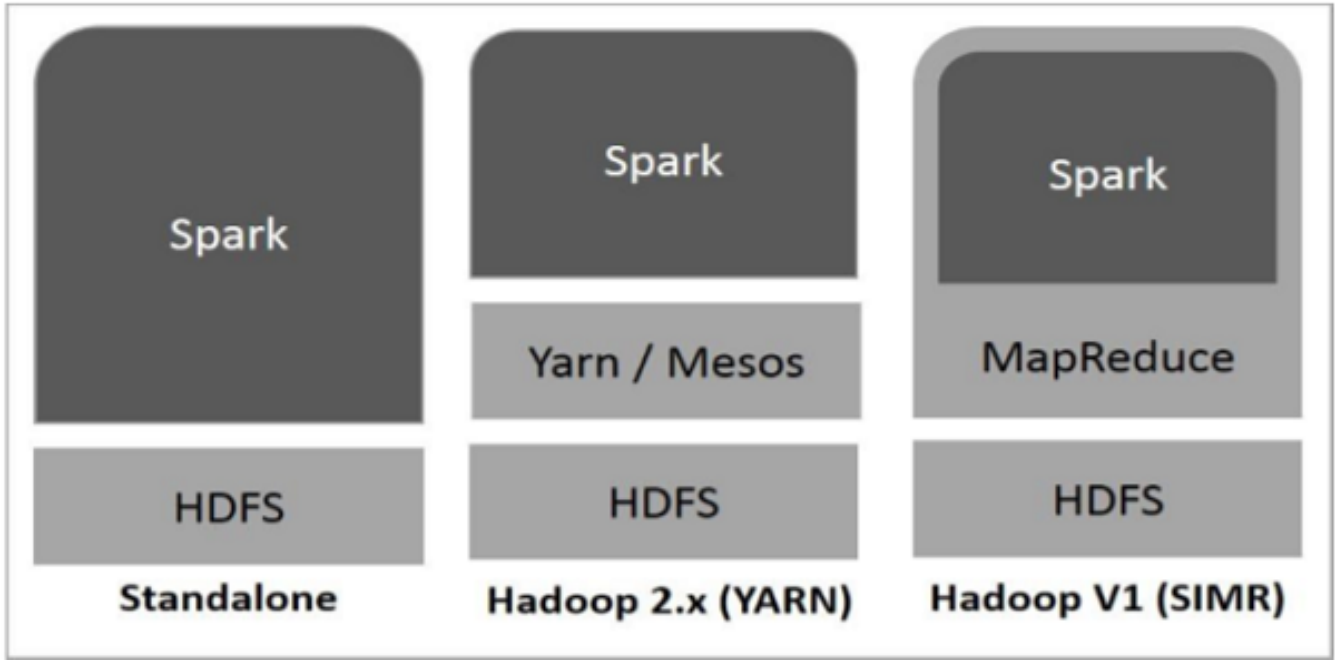
Spark – features

- Speed
  - Runs applications 100x faster in memory and 10x faster when running in disk
    - Using a Hadoop cluster
  - Possible by reducing the number of read/write operations to disk
    - Intermediate processing data is stored in memory
- Supports multiple languages
  - Built-in APIs in Java, Scala or Python
  - 80 high-level operators for interactive querying
- Advanced analytics
  - Beyond Map and Reduce
  - SQL queries, streaming data, machine learning, graph algorithms

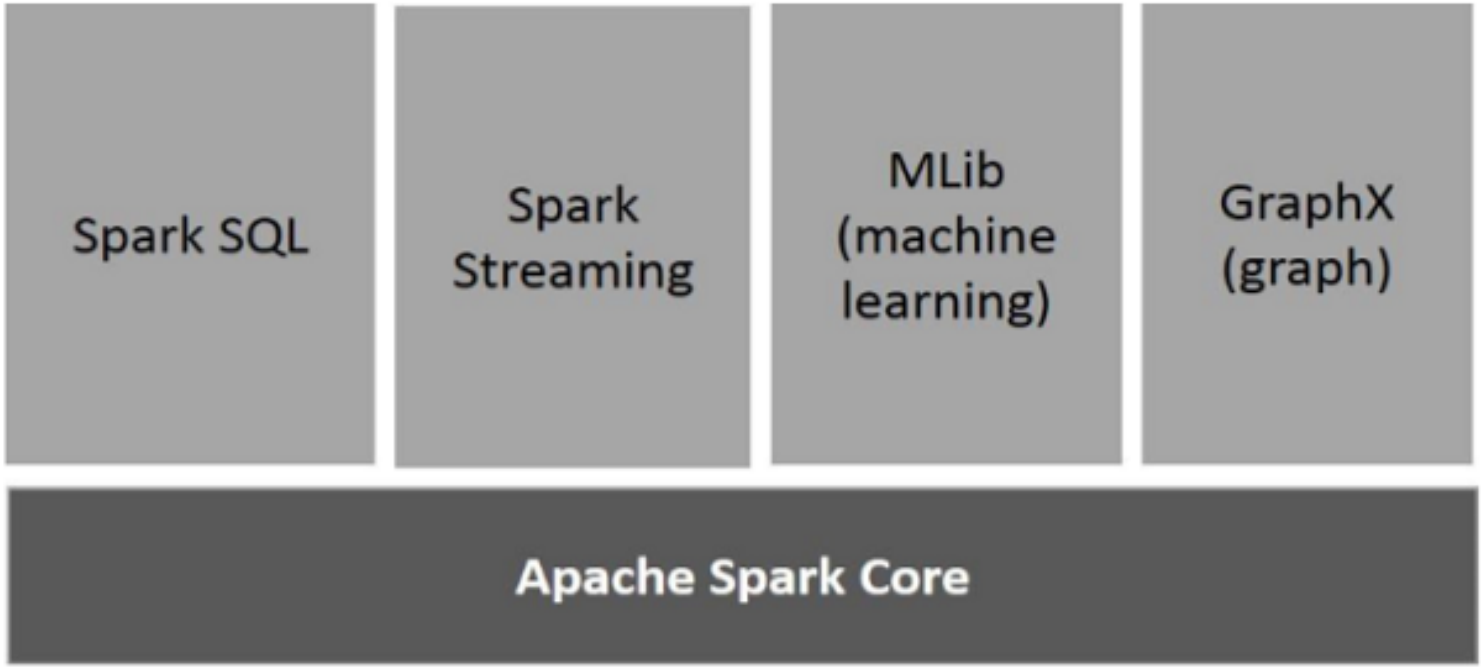
Spark – deployment

- With Hadoop (storage and/or processing)
  - HDFS
  - HDFS + YARN/Mesos
  - HDFS + MapReduce
- Without Hadoop
  - Storage
    - Cassandra, HBase, S3
  - Processing
    - Standalone, Mesos

Spark – deployment



Spark – Components



Spark – Components

- Apache Spark Core
  - General execution engine
  - Provides in-memory computing
  - Referencing datasets in external storage system
  - RDD (Resilient Distributed Datasets)
- Spark SQL
  - Provides support for structured and semi-structured data
    - SQL again...
  - SchemaRDD

Spark – Components

- Spark Streaming
  - Streaming analytics
  - Ingests data in mini-batches
    - Perform RDD transformations on those
- MLib (Machine Learning Library)
  - Distributed machine learning framework
    - Distributed memory-based architecture
  - 9x faster than Apache Mahout over Hadoop
- GraphX
  - Distributed graph-processing framework
  - API for expressing graph computation
    - Model user-defined graphs using Pregel Abstraction API

Spark RDD

- RDD – Resilient Distributed Datasets
  - Fundamental data structure of Spark
- Immutable distributed collection of objects
  - Can contain any type of Scala, Java or Python objects
    - Including user-defined classes
  - Each dataset in RDD is divided into logical partitions
  - Each partition may be computed on diferent nodes of the cluster
- Formally: read-only, partitioned collection of records
  - Created through deterministic operations on either data on stable storage of other RDDs
  - Fault-tolerant collection of elements that can be operated in parallel

Spark RDD

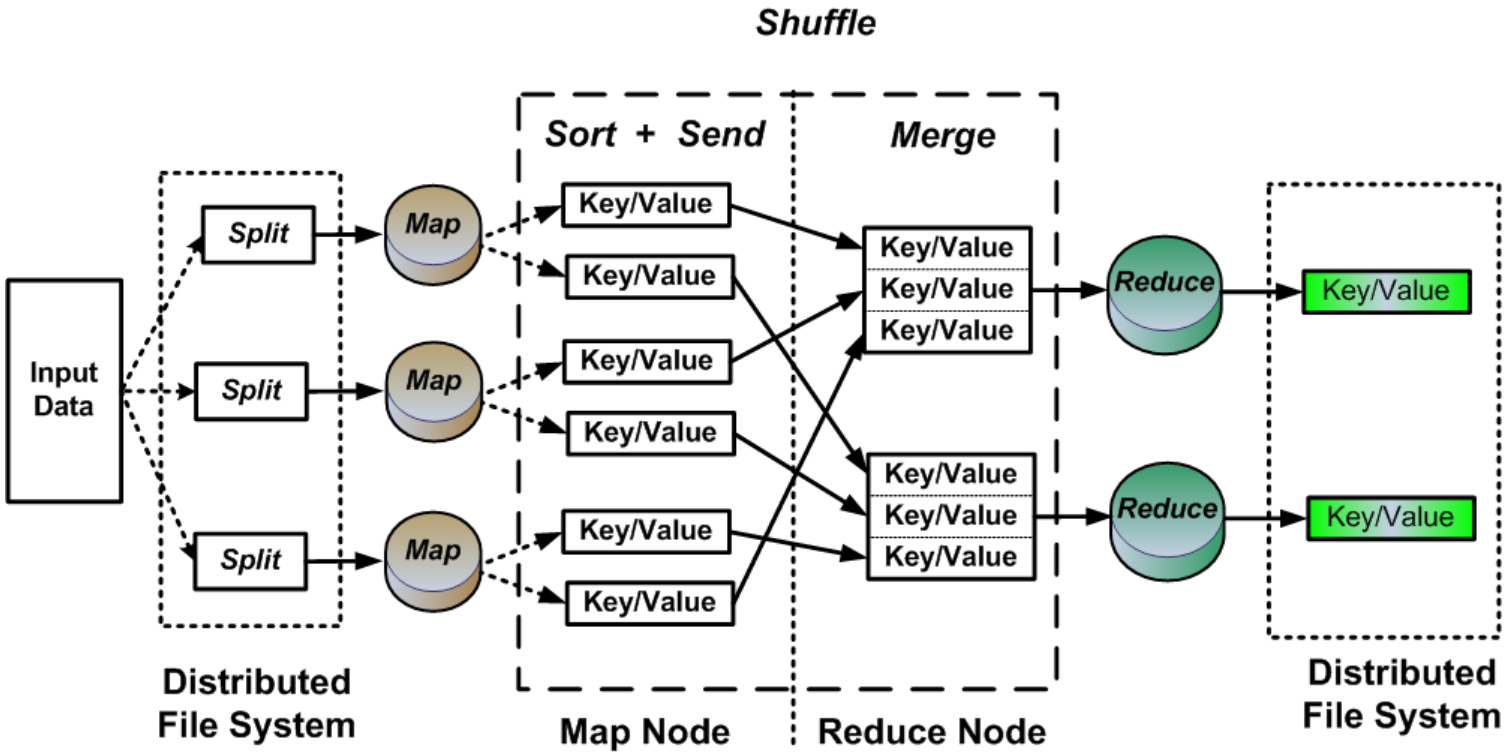
- Two ways to create RDDs
  - Parallelizing an existing collection in your driver program
  - Referencing a dataset in a external storage
    - Shared file system, HDFS, HBase, etc
- RDDs concept allows faster and efficient operations
  - MapReduce operations
  - Analisys operations

Why RDDs?

- Or any other kind of collections...
  - Don't we have too many options already?
- MapReduce (and similar solutions) is widely adopted
  - Processing large datasets with a parallel, distributed algorith in a cluster
  - High-level API
    - No need to worry about distribution, faul-tolerance, etc
- But...
  - In most current frameworks, the only way to reuse data between operations is to write it to an external stable storage system (like HDFS)
    - When you concatenate MapReduce jobs, for instance

Why RDDs?

MapReduce workflow

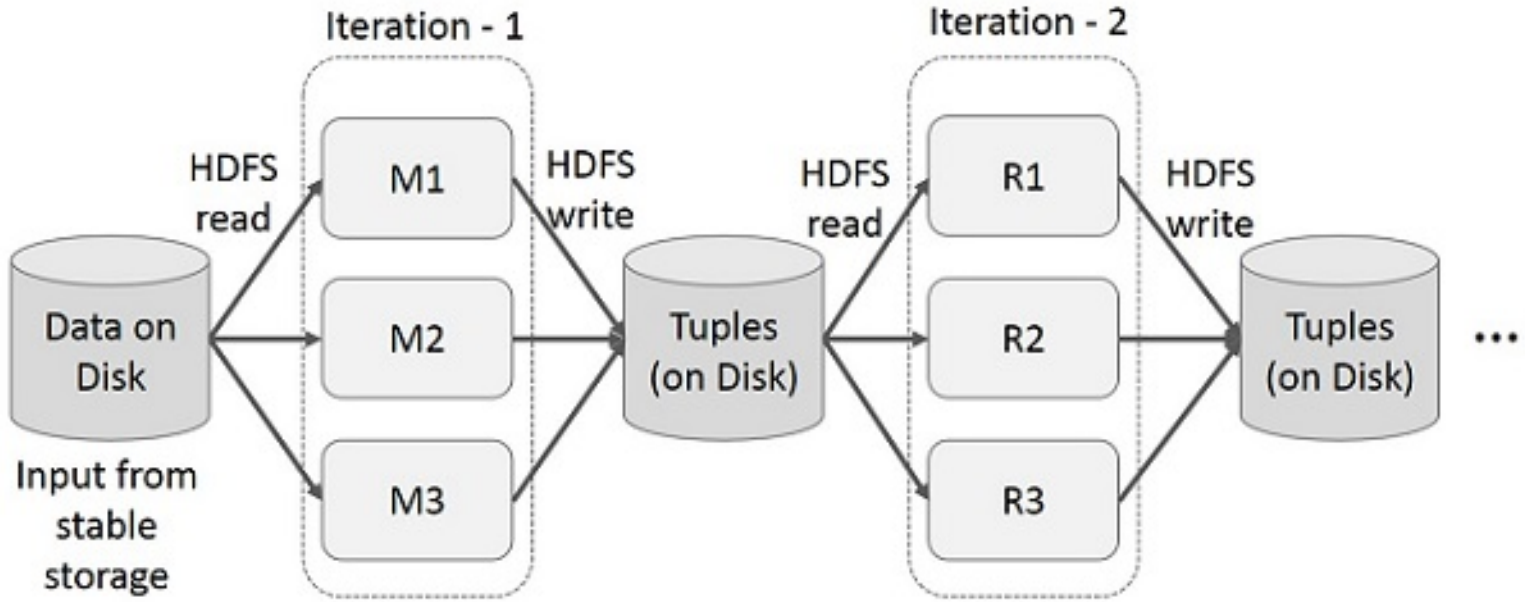


Why RDDs?

- Several frameworks already provides faster access to data in transit
  - But with limited sucess
- Data sharing is slow in MapReduce due to the same features that make the framework useful
  - Replication
  - Serialization
  - Disk IO
- Hadoop applications spend more than 90% of the time in HDFS read/write operations
  - Regarding system operations

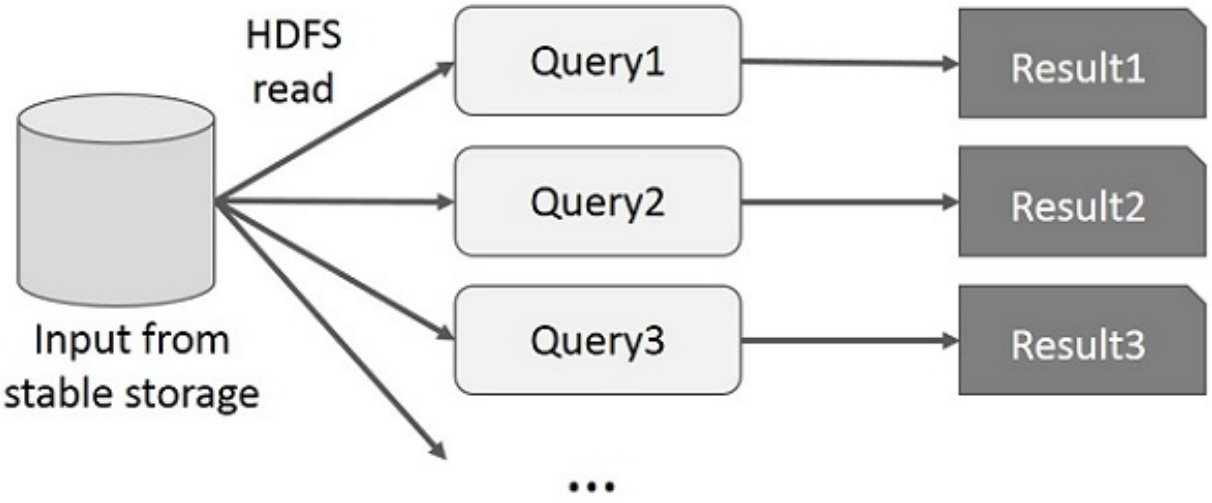
Iterative Operations on MapReduce

Reusing intermediate results across multiple operations in multi-stage applications



Interactive Operations on MapReduce

Ad-hoc queries on the same subset of data

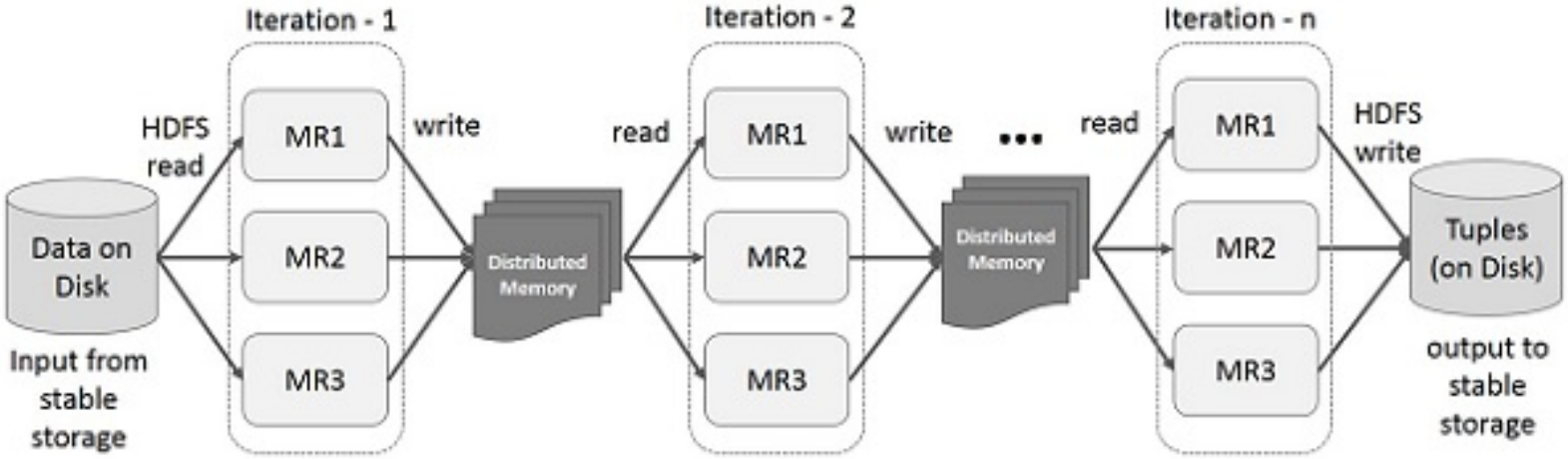


Using Spark RDD

- Data sharing is slow in MapReduce
  - Replication, serialization, disk IO
- Spark aims to these particular issues
- Resilient Distributed Dataset
  - Key idea
  - Supports in-memory processing computation
- Stores the state of memory as an object across the jobs
  - Object is shareable between those jobs
- Data sharing in memory is 10 to 100 times faster than network and disk

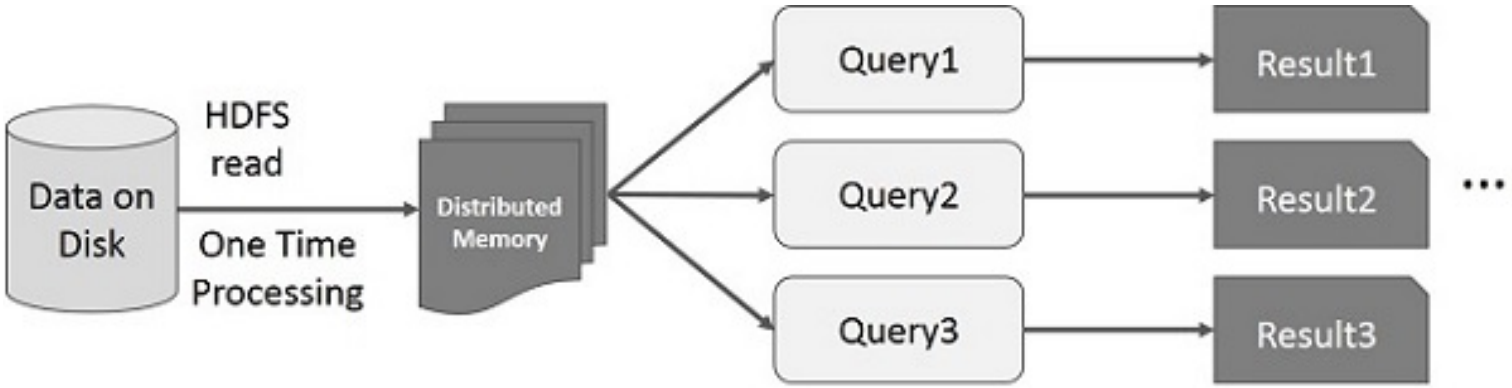
Iterative Operations on RDDs

Intermediate results stored in a distributed memory



Interactive Operations on RDDs

If different queries are run on the same dataset repeatedly, this data can be kept in memory



Operations on RDDs

- Distributed Memory (RAM) may not be sufficient to store intermediate results
    - Results will be stores on disk
- Each transformed RDD may be recomputed each time you run an action on it
    - The actual computation only occours when an action is performed
        - Lazy computation
- RDD can be persisted in memory
    - Spark will keep elements in the cluster
        - Faster access
- Spark support persisting RDDs on disk and/or replicate across multiple nodes

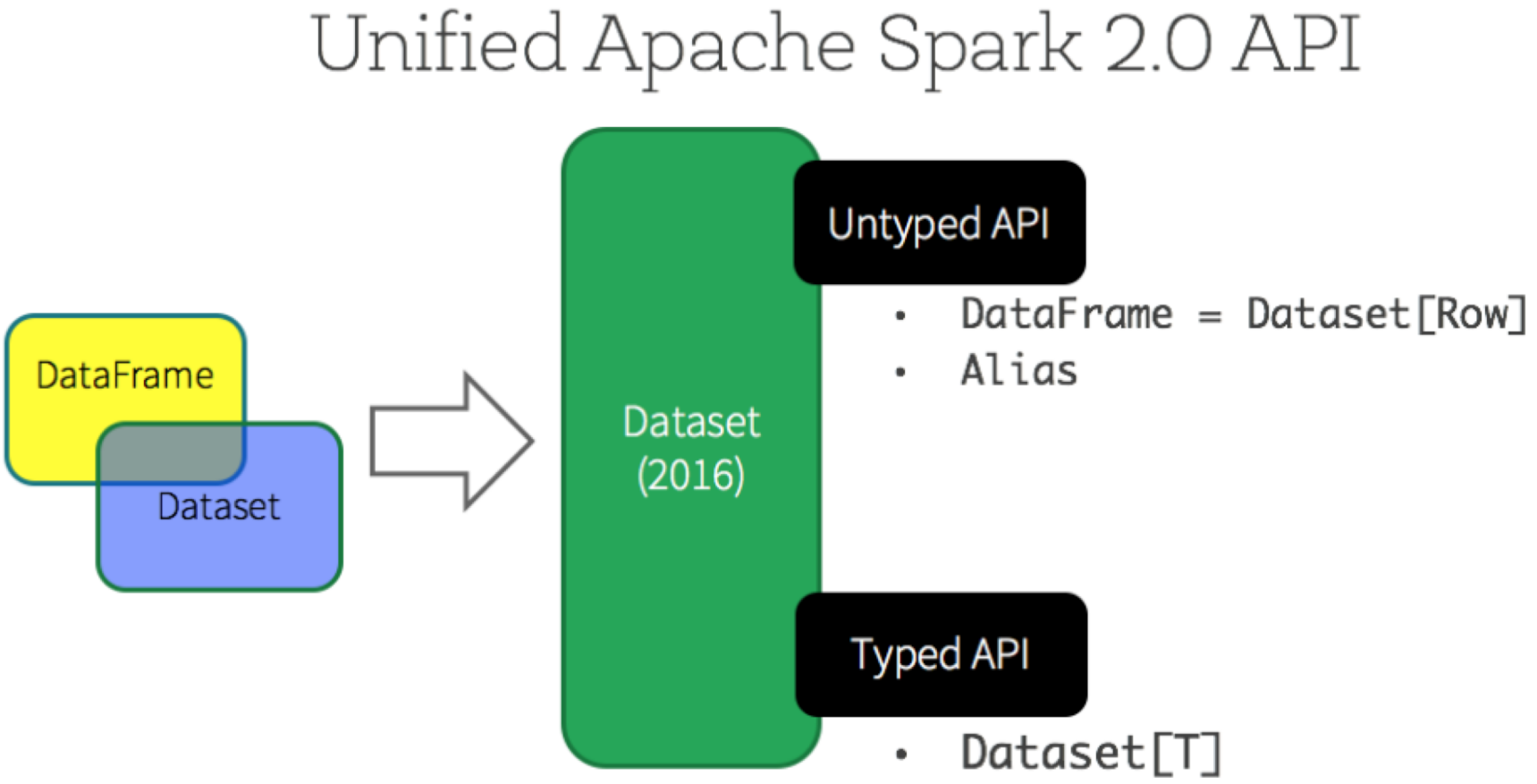
RDDs, DataFrames, DataSets

- RDD
  - Primary API in Spark
  - Similar to Scala collections
  - Very flexible
    - Unstructured data
- But…
  - Eventually, your data IS structured
    - XML, JSON
  - And you want high-level operations
    - SQL

RDDs, DataFrames, DataSets

- DataFrames
  - Collection organized in columns
    - Imposes structure
  - Provides domain specific language API
- DataSets
  - Spark 2.0
  - Strongly-typed API
    - Collection of strongly-typed JVM objects
  - Integrate DataFrame as DataSet[Row]

RDDs, DataFrames, DataSets



databricks

DUBLIN