# Introduction:



* Spark is general purpose cluster execution engine. It has no storage system.
* It is faster because it does in-memory execution. Which overcomes disadvantage of map reduce.
* Disadvantage of map reduce:
  1. Intermediate results in map reduce is stored in hard disk where as spark does everything in memory

**Map Reduce**:

HDFS -> map reduce -> storage -> map reduce -> storage

Here storage is Hadoop distributed file system

**Spark**:

HDFS -> RAM -> Spark -> RAM -> Spark -> Storage

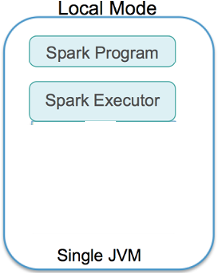
In case there is no RAM available than it starts using hard disk

* 1. Map reduce cannot optimize entire code whereas spark can optimize entire code. Ex: project Tungsten & catalyst SQL optimizer.

# Spark Architecture:

**Local mode**:

* In local mode entire processing is done on single server.
* **But it still benefits from parallelization across all cores in your server.**



**Cluster mode**:



**Flow:**

Submit Program 🡪 Driver starts (APP master) 🡪 Request cluster manager for resource 🡪 cluster manager allocates executors 🡪 executors will run their own tasks 🡪 results are returned to driver table 🡪 dump the final output to storage system.

**Driver Program:**

* Process running main function & creating spark context.
* Sparks runs independent tasks on worker node coordinated by spark context.
* Spark context can connect to many types of cluster manager.

**Cluster Manager:**

* Cluster manager is responsible for allocating resources
* There are many types of cluster manager is supported by Spark
  1. **Standalone**: included with spark
  2. **Apache Mesos**: general cluster manager that can also run Hadoop map reduce
  3. **YARN**: Resource manager in Hadoop 2
  4. **Kubernetes**: open source system for deployment, scaling & containerized application.

**Note**: when you ask for 10GB of RAM you will get only 54% of it as other 46% will be consumed by JVM & other processes

**Worker node:**

Any node can run application in the cluster

**Executor**:

Process launched for an application inside worker node that runs tasks & keeps data in memory or in disk storage across them. All programs run under JVM.

**Task**:

Unit of work sent to executor. It is created by job scheduler.

**Job**:

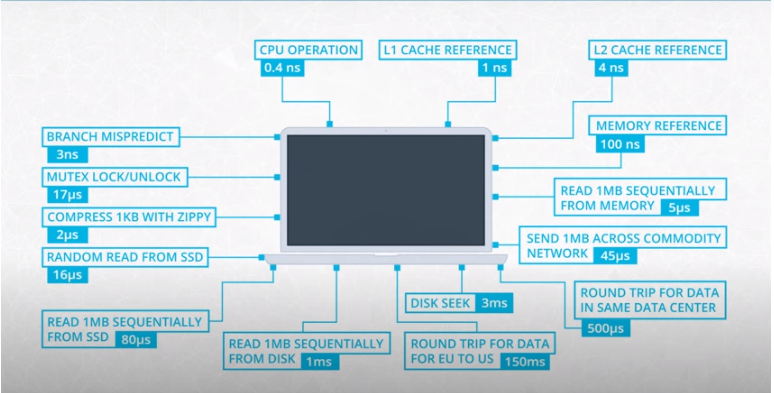
Job consist of multiple task happening in parallel in response to any action event (ex: collect).

Job is divided into smaller set of tasks called stages.

**Stage**:

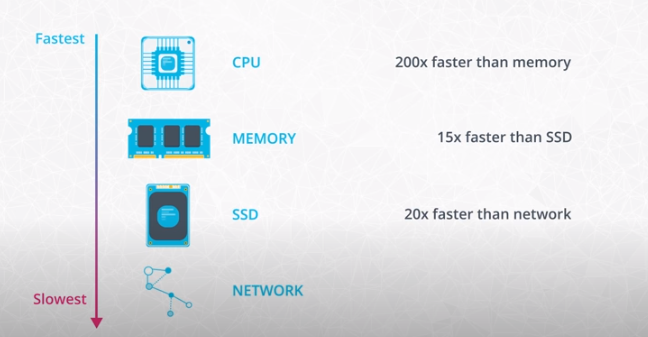
Each job is divided in to small tasks called stages similar to MapReduce job map & reduce are two stages

# Numbers everyone should know



Above is the exhaustive list of numbers developer should be aware of. But let’s focus on 4 key numbers.

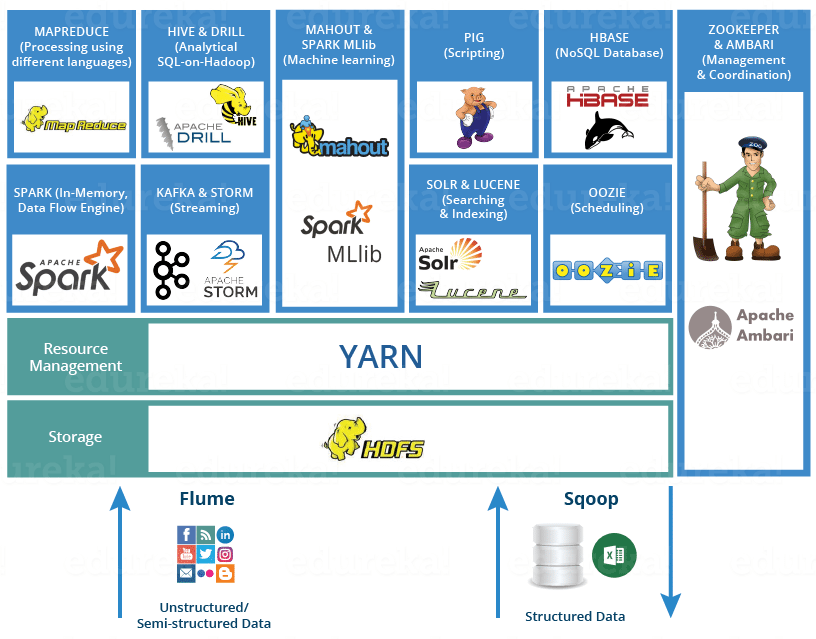
|  |  |  |
| --- | --- | --- |
| CPU | Brain of computer. Every operation is handled by CPU | 0.4 ns |
| Memory  (RAM) | Place to store data before it goes to CPU | 100ns |
| Storage (hard disk) | SSD or HDD, its used for long term storage. Your file cabinet | 16 micro s |
| Network | Connection to outside world | 150 ms |



* CPU has its own storage which is used store data temporarily. Lading data from that storage is 200 times faster than loading data from memory.
* Slowest component is the network reason why big data system avoid shuffling between clusters.
* Spark is optimized at storage level. Map reduce always store data in hard disk where as spark keep data in memory between the process.

**Note**: network within datacenter is really fast. Roundtrip in same datacenter takes around 500 micro seconds. That is the reason in cloud computing storage could be separated from execution engine to save cost.

# Hadoop ecosystem



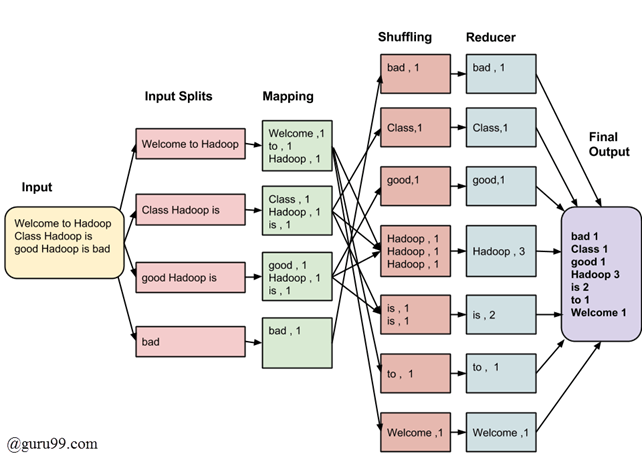
Hadoop framework consist of 3 main components

1. **HDFS**: storage system, it stores data on commodity machines
2. **Map reduce**: Data processing
3. **YARN**: resource manager
4. Apache Pig: a SQL-like language that runs on top of Hadoop MapReduce developed by yahoo
5. Apache Hive: a SQL-like language that runs on top of Hadoop MapReduce
6. Apache storm & Apache flink: for streaming data

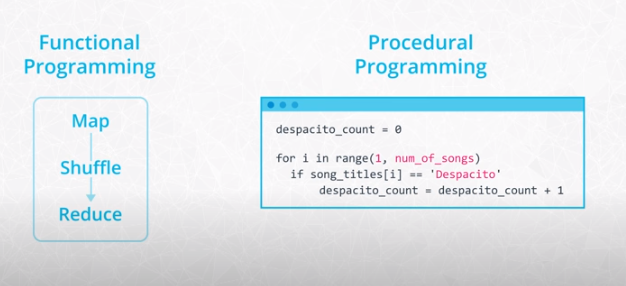
# Map Reduce

Map reduce consist of 4 steps

1. Input & split: input can be HDFS storage system in which data is chunked into smaller batch called partitions
2. Mapper function: creates key value pair
3. Shuffle: shuffles data across cluster to have same key together
4. Reducer: reduces data on the basis of key



# Functional VS Procedural VS object-oriented programing



Mainly there are three types of programing

1. Functional programing:
   1. Under the hood spark is written in functional programing language called Scala
   2. It is perfect for distributed systems
   3. In functional programing functions are treated as data meaning we can pass function as parameter
   4. Same input always results in same output & not depend on local or global state.
   5. Objects are immutable.
   6. Which makes them fault tolerant meaning if one of the nodes is broken we have to start process again & it will return same output always.
2. Procedural programming:
   1. Methods inside OOP is procedural programing
   2. Its mutable object.
3. Object oriented programing:
   1. Collection of objects
   2. Class can be considered as blueprint of object.

# RDD (resilience distributed dataset):

**Resilience**: withstand failure

**Distributed**: across multiple machine

**Read-only**, **partitioned** collection of **immutable** records.

There are two ways to create RDD:

1. Parallelizing existing collection in your driver program using **sc. parallelize**
2. Or reading from external data storage like HDFS, Hbase any other data source offering Hadoop input format.

RDD is made up of four main part:

* Partitions
* Dependencies
* Functions: computing new RDD based on parent RDD
* Metadata about partition scheme & data placement

# Partitions:

* Each RDD data split into multiple partition consider RDD as a pointer.
* Data in same partition will be on same machine. By default, no of partition is equal to number of CPU cores of machine.
* Spark runs 1 task for each partition. In general, more partition allows more parallelism.
* How to choose number of partitions?
  + Lower bound = 2 times more than number of cores in cluster
  + Upper bound = task should take more than 100+ ms to complete else application will take much more time to distribute than execute

**Types of partitions in Spark:**

1. Hash partitioning:
   1. concept of hash partitioning is keys having same hash value should be in same partition
   2. its default partition in spark
2. Range partitioning:
   1. First range partition will sort the record based on the key & than it will divide data in number of partitions.

**Note**: one of the major issues while partitioning is that data might be skewed. This means majority of data will be with one executor and less data on other executor. Overall time taken to complete would increase so partition wisely.

**Note**: Data localization is not guaranteed when you work with spark. Hadoop does good job here as mapper will be sent where data is. When spark application in run & ask resource from cluster manager, cluster manager does not know where is the data. So data localization could not be achieved.

# Transformation & Action:

There are two type of apache spark RDD operations are

1. **Transformation:** 
   1. allows you to create new RDD from existing RDD with some condition
   2. when you apply transformation dependency graph will be created
   3. it could be narrow or wide transformation.
2. **Action:**
   1. actions trigger computation & output of that is retuned to driver node

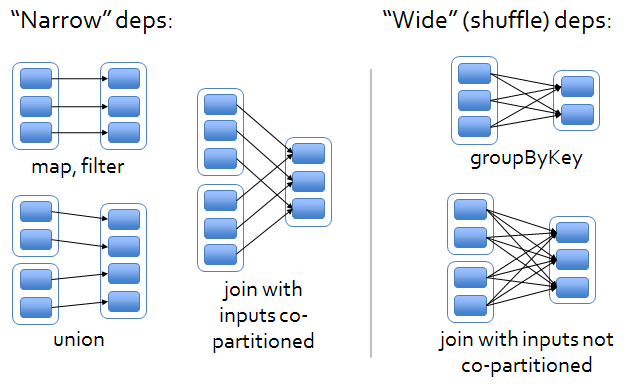
# Narrow & Wide dependencies:

**Narrow dependencies:**

* When each partition of parent RDD is used by at most one child RDD
* Data shuffling is not required.
* Ex: Filter, map

**Wide dependencies:**

* When each partition of parent RDD is used by multiple child partitions.
* Data shuffling is required
* Ex: groupbykey



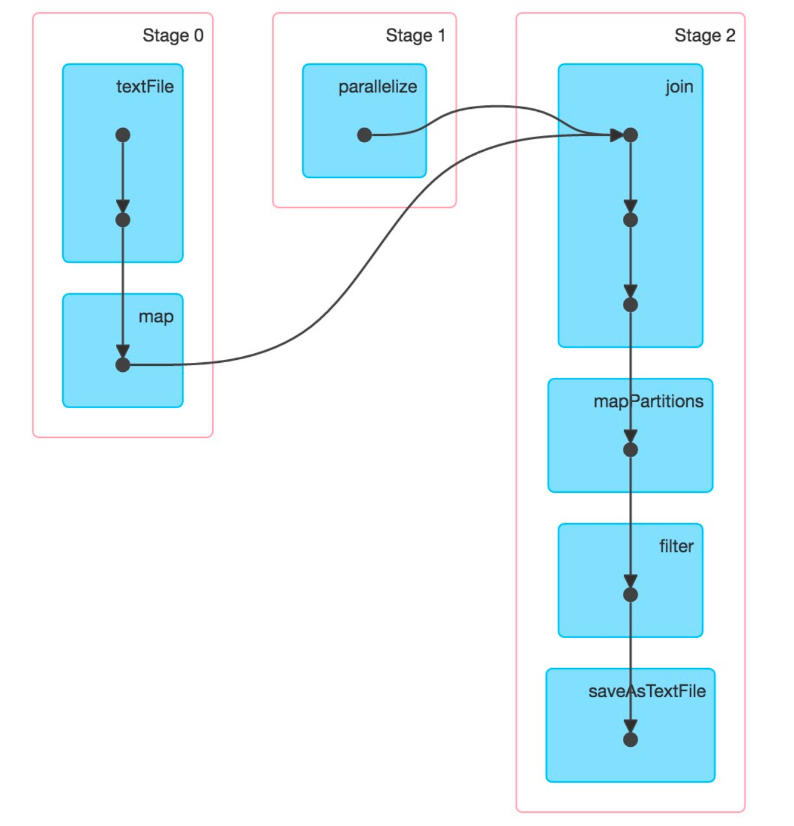
**Note**: Join operation could be narrow as well as wide dependencies

# DAG (Directed Acyclic Graph):

DAG in spark is set of vertices connected by edges where vertices represents the RDD & edges the operations to be applied on RDD

How it works?

* Spark creates graph when you write a code (sequence of transformation)
* When action is called on RDD. Spark submits the graph to DAG scheduler
* DAG scheduler divides graph into stages on the basis of transformation. All narrow transformation is grouped together in one stage. This is how it optimized the performance.
* Shuffle operation i.e. wide transformation defines boundary of two stage.
* DAG scheduler will submit stages to task scheduler depends on number of partitions for ex is no of partition is 4 than 4 tasks will be submitted.
* If stage can be run parallel than both stage can be submitted together in below example stage 0 & 1 can run together to increase parallelism.



# Spark session v/s Spark context:

* Spark session is a unified entry point of a spark application from Spark 2.0. It provides a way to interact with various spark’s functionality with a lesser number of constructs.
* Instead of having a spark context, hive context, SQL context, now all of it is encapsulated in a Spark session

# Coalesce vs Repartition:

**Coalesce**:

* Reduces number of partitions in spark data frame
* Faster than partition function because minimizing data movement between partitions

**Partition**:

* Can reduce/increase partitions
* It will do complete reshuffle. So slower than coalesce

# Caching & Persistence:

One of the mechanisms to speed up application is to cache the RDD which is used multiple times. If RDD is not cached it will be re-evaluated again if we call an action. Two function is available cache & Persist.

**cache()** will cache the RDD in memory & **persist()** can catch in memory , on disk or off-heap memory according to caching strategy. Persist without argument is cache.

Freeing up space can be done with **unpersist()**

**levels in persistence**:

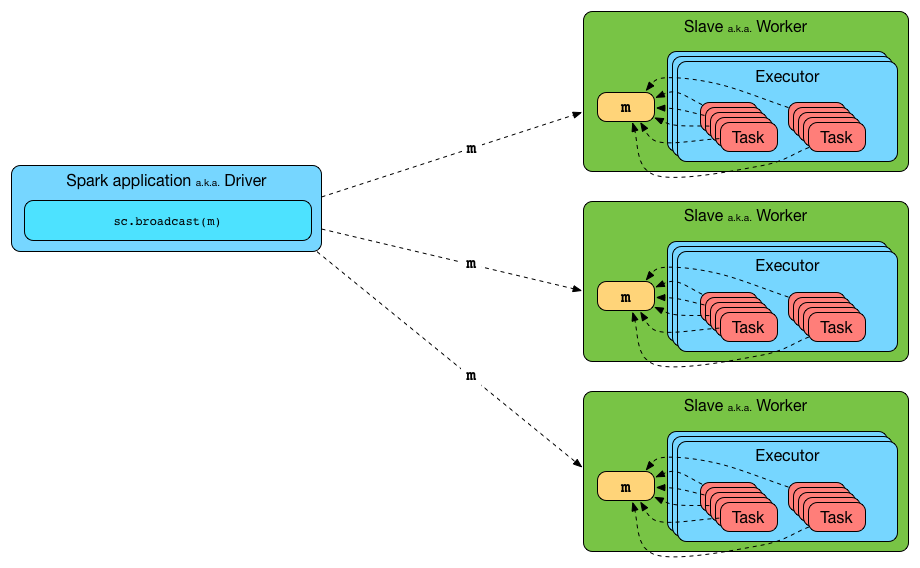
1. disk\_only: save data to disk
2. memory\_only: saves data in memory
3. memory\_and\_disk: keeps data in memory when out of memory saves to disk
4. disk\_only\_2: same as memory\_and\_disk but makes two replicas.

**When to use caching**:

1. RDD re-use in iterative machine learning application
2. RDD re-use for standalone spark applications
3. When RDD computation is expensive caching can help to reduce cost of recovery when it fails

# Broadcast variable:

Broadcast variable allow the programmer to keep read-only variable cached on each machine rather than shipped with the task. Without broadcast variable these variable would be shipped to each executor every time transformation or action is performed which causes network overhead .



Use case:

1. In code if we want to do lookup on large table of zip code, it is not feasible to share such a large dataset across network on each transformation.
2. Sharing database connection file.

# Accumulator:

Accumulator are write-only variables for executors. They can be added by executors & read by the driver only.

Use case:

1. They can be used to implement counters or sums
2. Only associative operations can be used in accumulator.
3. Natively supports numeric types but custom accumulator can also be created.

# Other resources:

<https://mallikarjuna_g.gitbooks.io/spark/spark-broadcast.html>