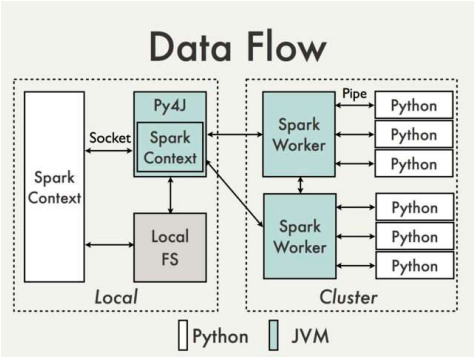
Notes: Apache Spark, Hadoop, Big Data & Distributed ML

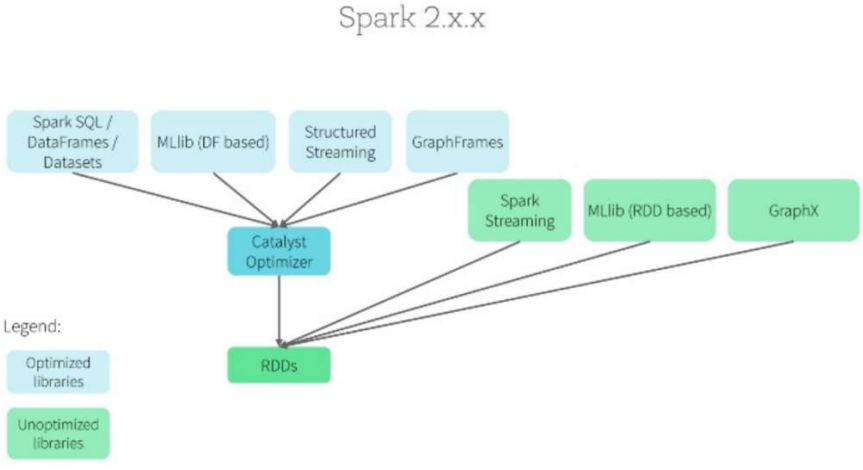
Refer to this comprehensive account of Apache Spark latest release 2.3 from its authoritative owner DataBricks https://www.slideshare.net/databricks/spark-saturday-spark-sql-dataframe-workshop-with-apache-spark-23.

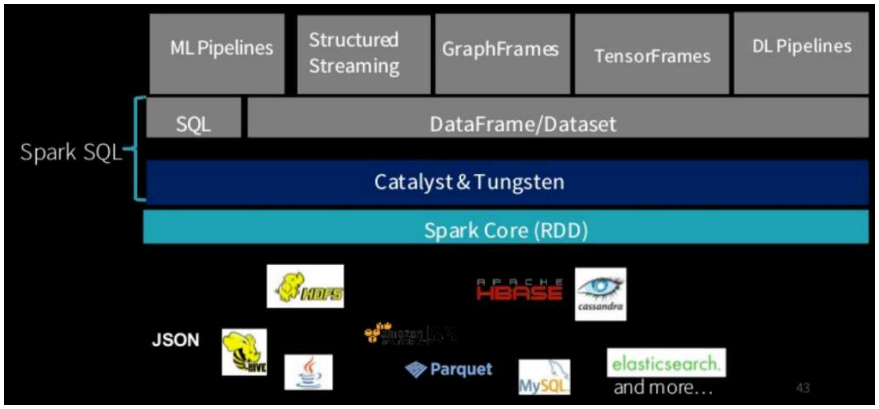
Memory management optimization in JVM environment is key to Spark efficiency and this reference https://devenderprakash.wordpress.com/2017/02/26/serialization-in-bigdata/ captures the crux really well!

- Summary: Use Spark DataFrame & Spark SQL for ML/DL in Scala or Python. Note that the most efficient Spark DataSet is not available in non-JVM languages. Even performance wise, one need not work directly with RDD unless one wants to control own optimization.
- Anyone coming with background in Python, Panda, SkLearn based ML can attempt DataFrame based Spark.ml in Python. But NOTE that one must scale up to Scala to come out of the mould of Data Scientist into the world of Data Engineering!
- Ref to https://www.youtube.com/watch?v=fNtyDpWOA2k Big Data beyond JVM, i.e., what is the overhead of Python ← →Py4J & extra layer of de/serialization & how Spark DataFrames help avoid some of these overheads!



- Most of the open source Stats/AI/ML framework for local powerful multicore CPU or GPU machines are available in Python/R which are implemented in low level C/C++. But it is just Java & JVM at the core and hence Scala as functional programming language rules the roost in Big Data & distributed ML!
- Unified API (SparkSession, DataSet, DataFrame & Mlib)
- Dataframe based Mlib is usually called spark.ml and observe such namespace in Spark framework
- Apache Spark Tungsten & Catalyst Optimizer are at the core of cost-based-optimization & code generation in terms of DataSet/Frame transformed into optimized RDD, transformations & actions https://jaceklaskowski.gitbooks.io/mastering-spark-sql/spark-sql-tungsten.html
- Tungsten Off-Heap Memory Management
    - Own off-heap binary memory management
    - Reduces the usage of JVM objects (and therefore JVM garbage collection)
    - No Java objects… uses sun.misc.Unsafe to manipulate raw memory
    - As DataSet/DataFrame have known schema, it properly and in a more compact and efficient way lays out the objects on its own
- Tungsten Cache Locality: It uses algorithms and cache-aware data structures that exploit the physical machine caches at different levels - L1, L2, L3
- Observe blue -vs- green shaded blocks in below hierarchy diagram for Un/Semi-Structured-RDD -vs- Structured-DataFrame-Dataset based services where later ones are Catalyst & Tungsten optimized

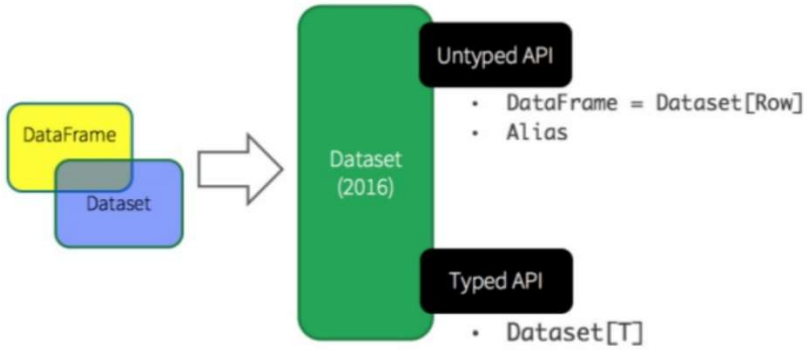




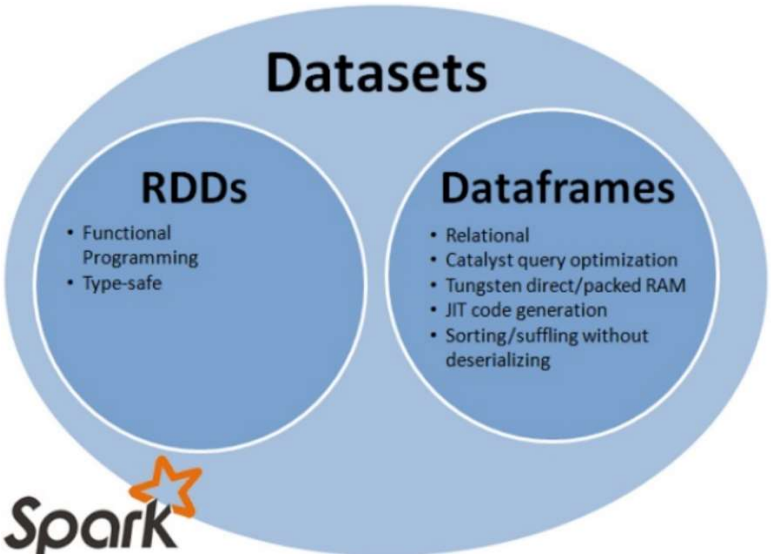
- SparkSession (all contexts & conf in single context… SparkContext, SQLContext, HiveContext, StreamingContext)

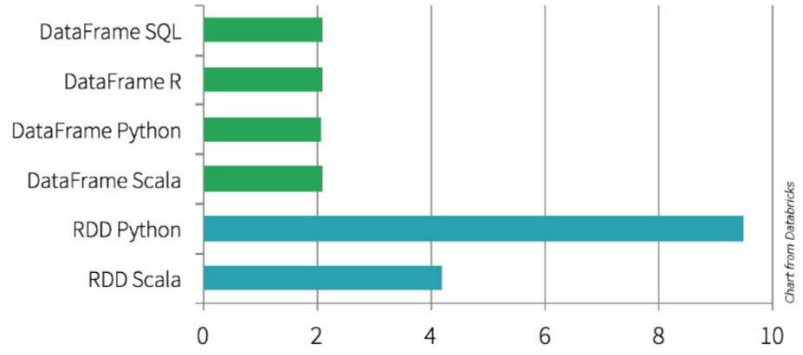
RDD -vs- DataFrame -vs- DataSet

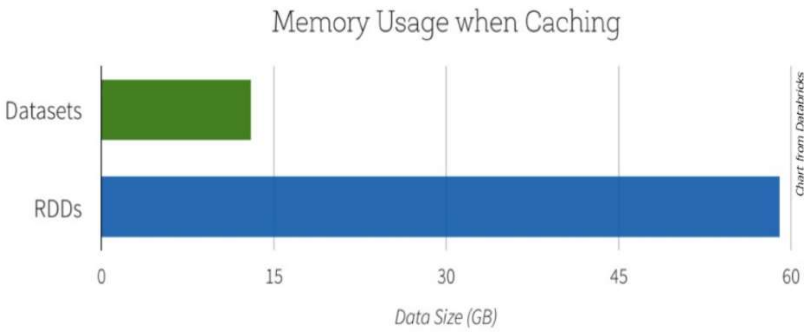
- DataSet is STATICALLY TYPED as Java object, i.e., it is close to bare metal, hence not available in non-JVM languages



- Observe below that Spark DataSet delivers benefits of both Spark RDD & Spark DataFrame. Spark DataSet being Java objects avoids extra de/serialization overhead and hence it consumes a lot less memory!

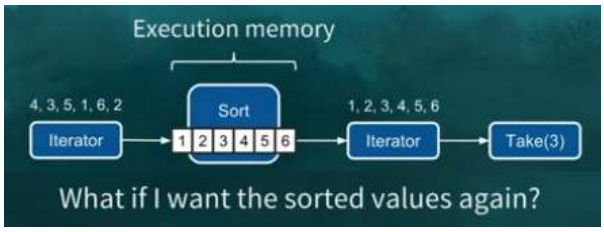




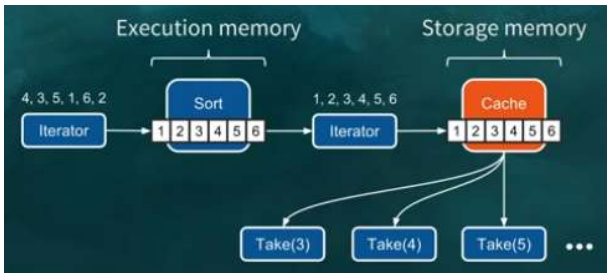


Unified Memory Manager (ref Apache Spark Memory Management https://www.youtube.com/watch?v=dPHrykZL8Cg)

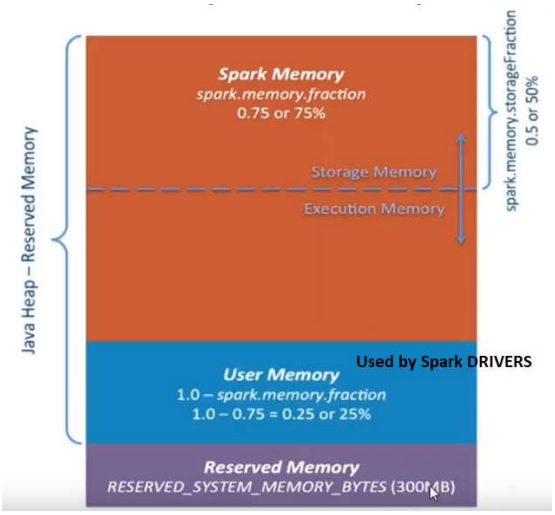
- Shared memory from nodes… Used for execution & storage
- Execution Memory: Used for INTERMEIDATE results during computation in
    - Shuffle
    - Join
    - Sorts
    - Aggregation



Spark ITERATOR are fundamentally read-once => hence we need to CACHE the result of sorting if we need that sorted data again & again



- Storage Memory: For future computation… Used for CACHING & PROPAGATING data across nodes in cluster
- Default mem mgr w/
    - onHeapExecutionMemory
    - onHeapStorageMemory
- Below settings through JVM config… below diagram looks like STATIC settings but UNIFIED MEMORY MANAGER allowed for dynamically crossing Execution/Storage boundary and in both cases Storage LRU (least recently used) Blocks are spilled over to disk



- How to arbitrate memory contention b/w…
    - … execution & storage? ➔ Dynamically use whole memory as long as it is available but if other party (Execution or Storage) needs some space then Storage LRU is spilled to disk
    - … tasks running in parallel? ➔ Instead of keeping as many SLOTS as CORES count on the worker node, keep as many SLOTS as actively running tasks
    - … operators running within same task? ➔ Different code modules can ask other code module to release PAGE to relieve some PAGE for it
    - Common theme of these solution is to AVOID STATICALLY RESERVING MEMORY
- Off-heap memory… avl for both Execution & Storage from Spark 2.0 onwards

Spark ML on Azure HDInsight

- Azure HDInsight is Apache Hadoop distribution for Azure
- HDInsight cluster *processing* node types available are Apache Hadoop, Hbase, Spark, Kafka, Storm & R-server
- HDInsight storage can be either Azure Blob Storage -or- Azure Data Lake
- Azure DataBricks is a feature packed alternative to Azure HDInsight for managed Apache Spark-as-a-service on cloud
    - It is actually Spark-as-a-service, where the first benefit you get is pause/resume, auto-termination & auto-scaling for your cluster
    - Native Azure-AD integration
    - Wide variety of IO option (Azure SQL DW, Azure Cosmos DB, Azure Event Hub, Apache Kafka for HDInsight & PowerBI) => These IO accessibility can be achieved in Azure HDInsight but indirectly through Azure Data Factory

Spark ML on Databricks Community

Databricks -vs- base Apache Spark https://databricks.com/product/comparing-databricks-to-apache-spark.

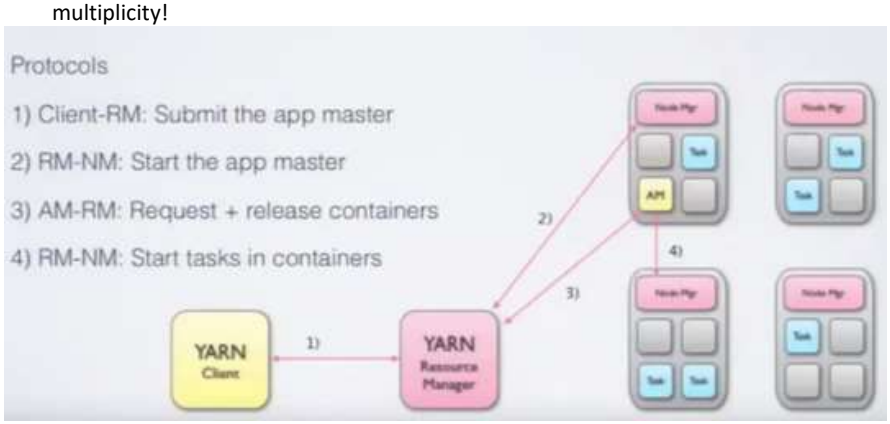
Databricks Community edition gives a micro-cluster, a cluster manager and the notebook environment (Interactive Cluster). It is not time-limited and no AWS cost. It exposes useful interfaces for viewing Spark Master UI, Driver Logs, Metrics, event logs , Jobs DAG graph, etc. This link https://databricks.com/blog/2015/06/22/understanding-your-spark-application-through-visualization.html has quite a few details of visualization on Databricks platform.

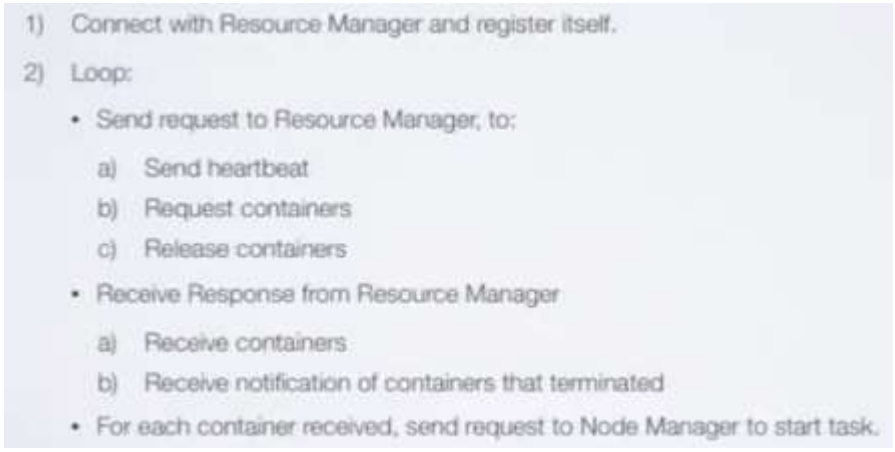
It comes with the **limitation** of max 6GB memory, no GPU, Driver & Worker(s) are community optimized, and it works in Driver-only Local Mode (master = local[8]). Databricks Runtime ML which packs popular ML libraries TF/Keras/XgBoost/distributed-TF is not available in Community edition.

YARN

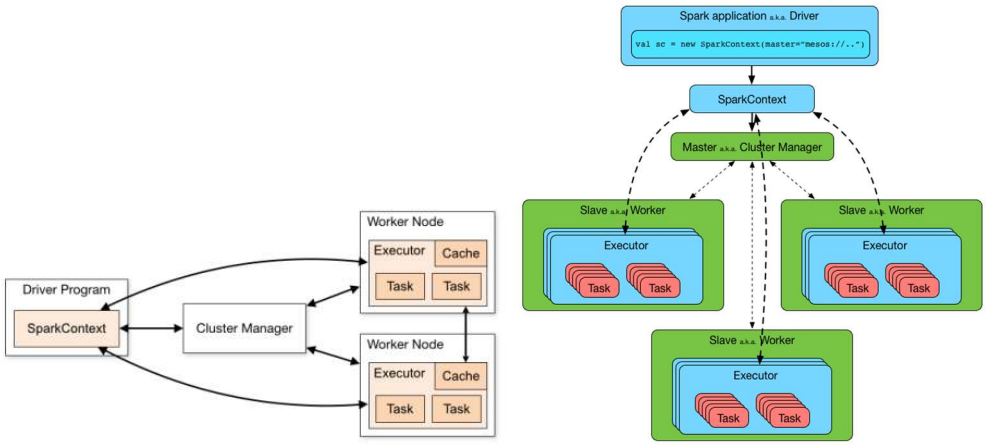
- Ultimate precise explanation about **YARN: Hadoop Beyond MapReduce** https://www.youtube.com/watch?v=HHv2pkIJjR0&t=22s
- Motivation
    - Until Hadoop 2.x, its native RM was tied into Hadoop system & RM could understand just Map/Reduce applications
    - If cluster has idle or partially used resources, then running other kind of applications was not possible
    - YARN separates Resource Management from Application Management

- Multi-tenant, Secure, Scalable
- RM (master daemon): YARN Adds 1 mode/machine as YARN Resource Manager (RM)
- AM: Every Applications has its Application Master residing on one of the cluster node slot (communicates with YARN RM to get available resources & requests NM to starts tasks… NM monitors the tasks)
- NM: Machine === Node === Has several SLOTS (Slot === CONTAINER with JVM to run TASK) + Node Manager (to manage its Slots, i.e., start & monitor Tasks)
- Slot: CONTAINER with JVM to run JVM task or run even Shell command. It is extremely important to have CONTAINERs optimized for memory & load time as the whole scalability managed by Resource Orchestrator (YARN/K8) demands its multiplicity!





- Hadoop, YARN, Spark coupling



  - Configs for memory/cores/etc for YARN & Spark https://stackoverflow.com/questions/43214762/how-to-solve-yarn-container-sizing-issue-on-spark => Note that Spark or any other service can consume full or subset of resources that YARN has available
  - Note from above link that each Node has to advertise its offers, hence requiring some bootstrap sequence => https://docs.qubole.com/en/latest/user-guide/clusters/node-bootstrap.html node bootstrap script is invoked after the HDFS daemons have been bought up (the NameNode on the Master and DataNodes on Slaves) but

before MapReduce and YARN daemons have been initialized. This means that Hadoop applications are run only after the node bootstrap completes.

- YARN RM -vs- Hadoop NameNode: They can run on different nodes.
- YARN NM -vs- Hadoop DataNode: Every slave nodes should have both running.