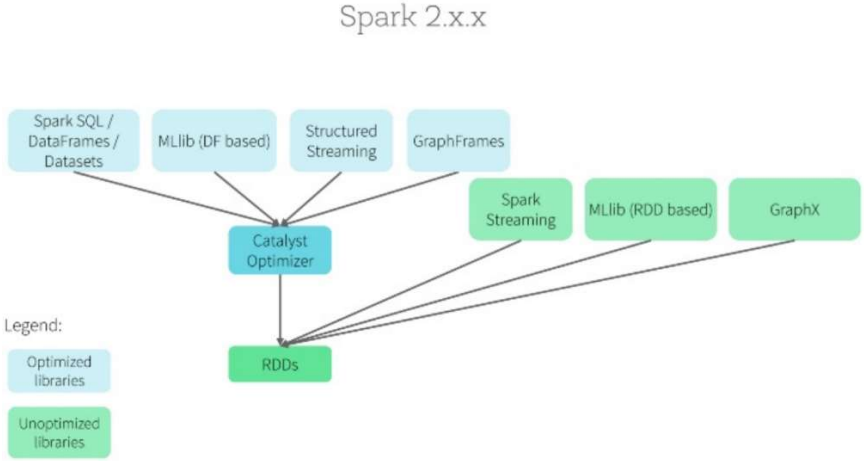
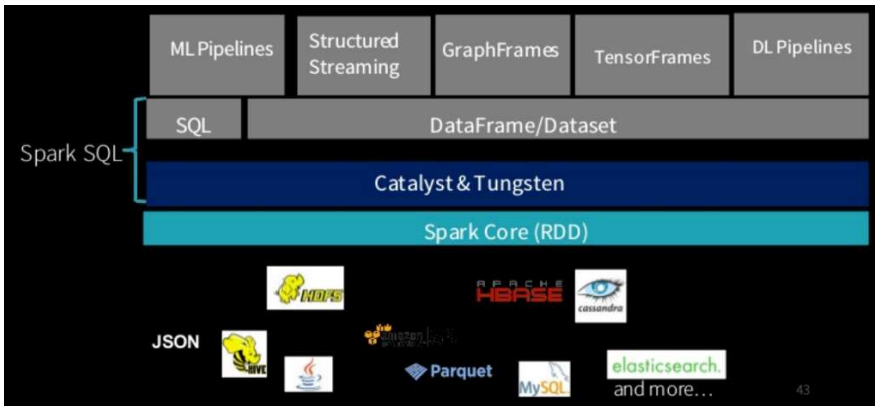
Notes: Apache Spark for Machine Learning & Deep Learning

This analysis was captured while exploring **if PySpark would be sufficient & efficient enough for Machine Learning & Deep Learning**. 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

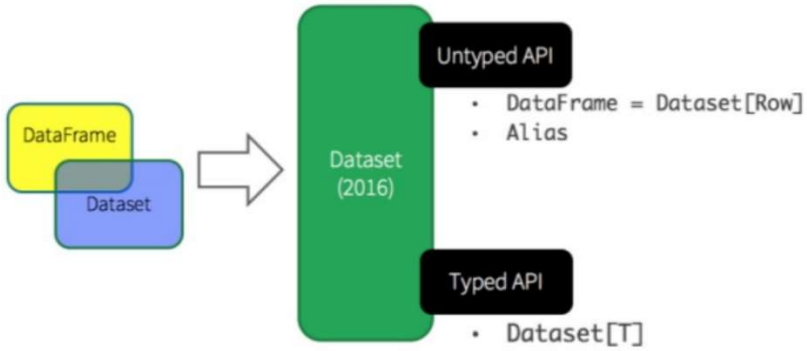
- Summary: Use Spark DataFrame & Spark SQL for ML/DL in PySpark. Note that the most efficient Spark DataSet is not available in non-JVM languages. We will worry about using RDD for low level control or own optimization
- Unified API (SparkSession, DataSet, DataFrame & Mlib)
- Apache Spark Tungsten & 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



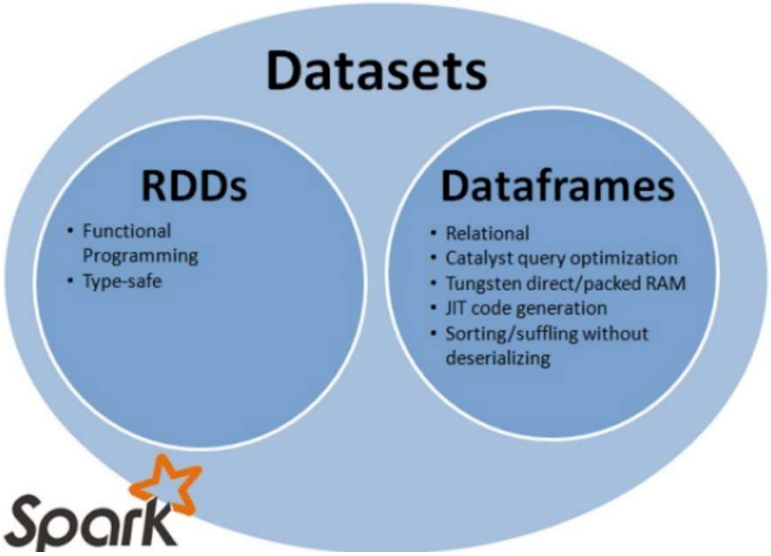


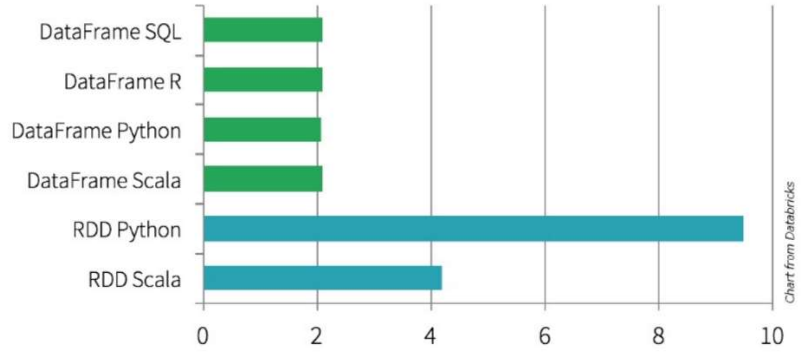
- SparkSession (all contexts & conf in single context… SparkContext, SQLContext, HiveContext, StreamingContext)

- 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



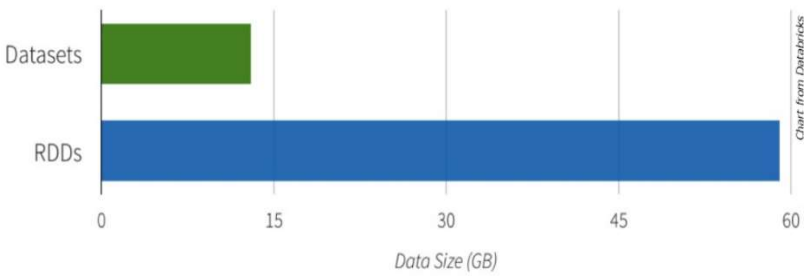
- Observe below that Spark DataSet delivers benefits of both Spark RDD & Spark DataFrame. Spark DataSet being Java objects avoids extra de/serialization overhead and hence it consumes a lot less memory!





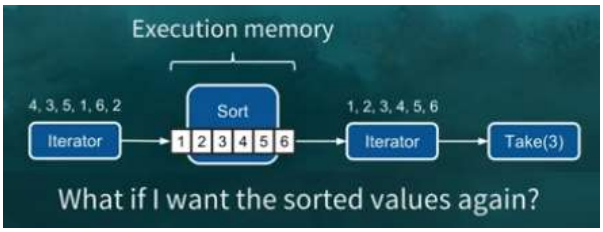
Time to aggregate 10 million integer pairs (in seconds)

Memory Usage when Caching

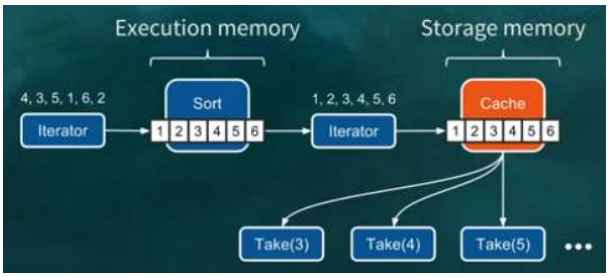


Unified Memory Manager (ref Apache Spark Memory Management https://www.youtube.com/watch?v=dPHrykZL8Cg)

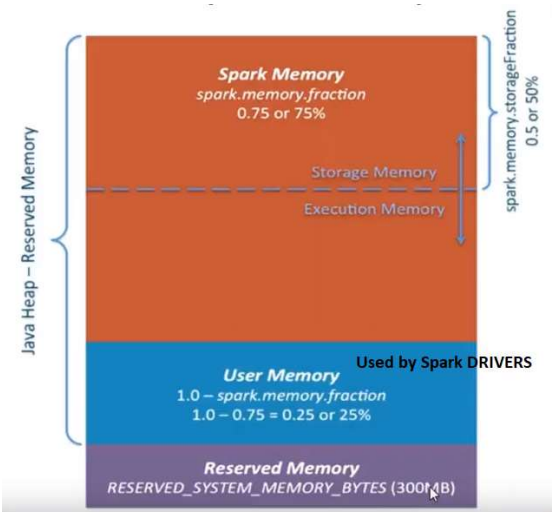
- Shared memory from nodes… Used for execution & storage
- Execution Memory: Used for INTERMEIDATE results during computation in
    - Shuffle
    - Join
    - Sorts
    - Aggregation



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