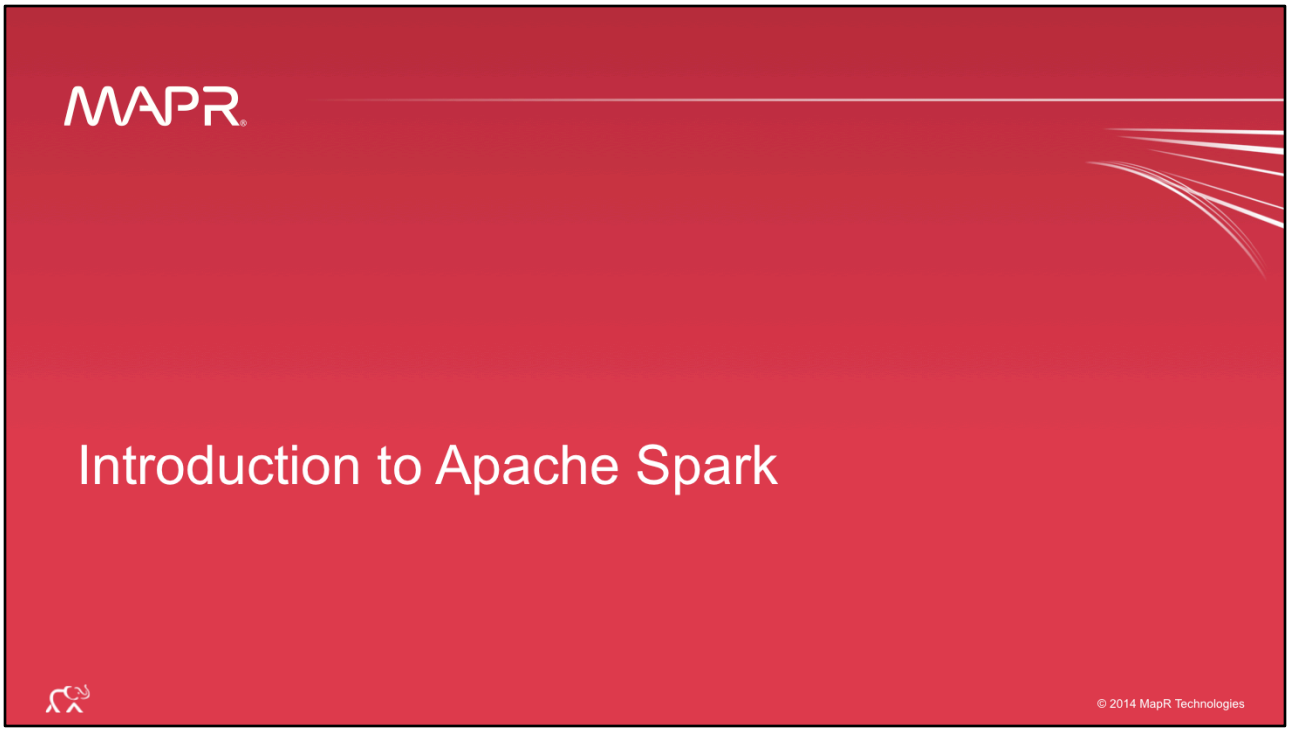
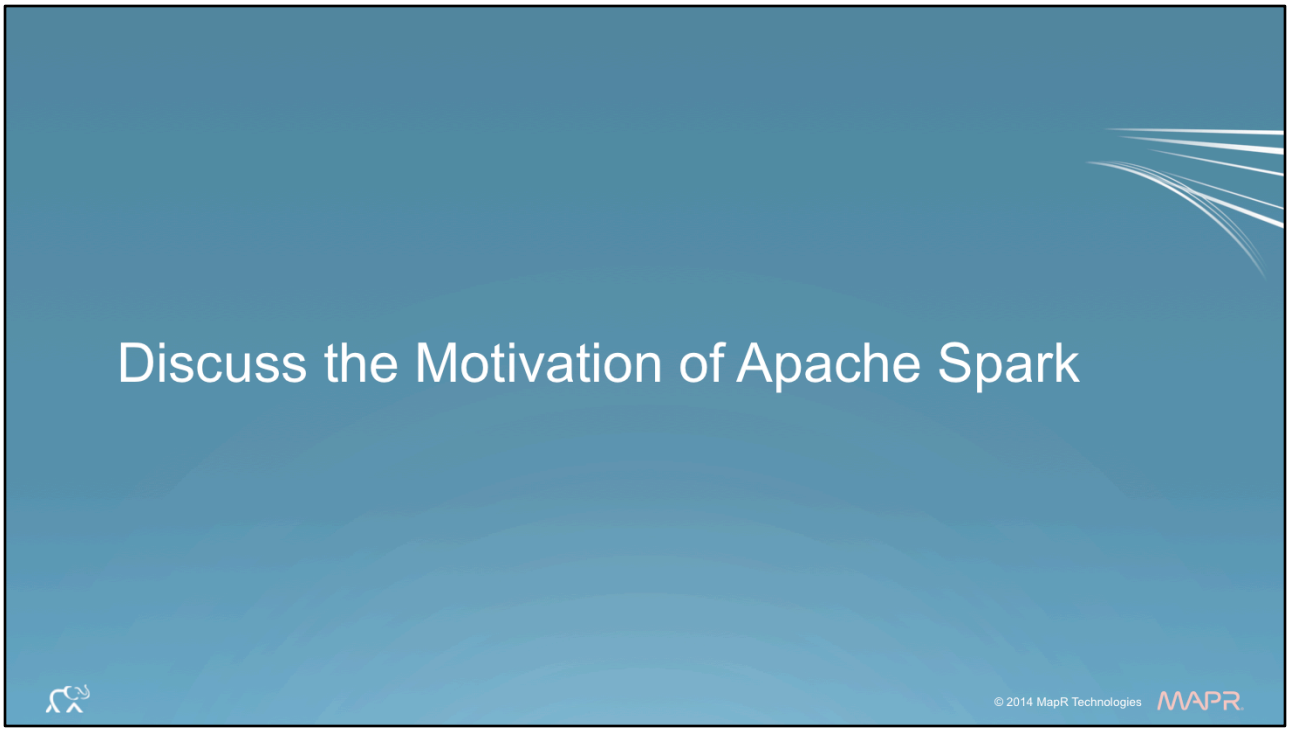
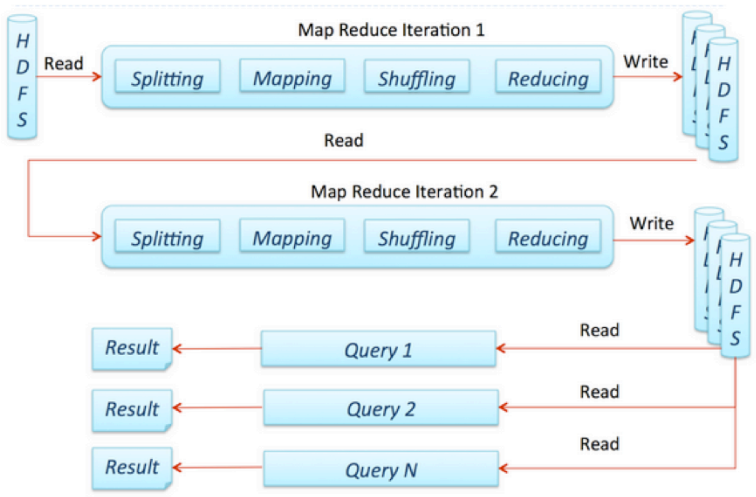


This lesson provides a brief introduction to Apache Spark.

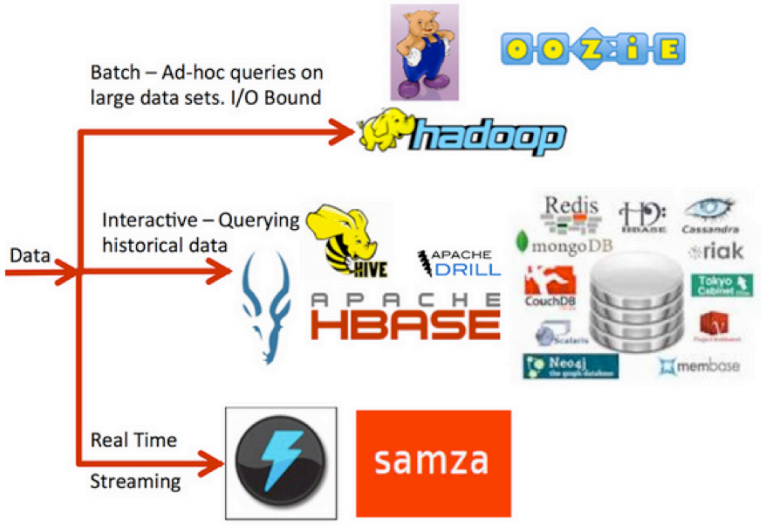


This section describes the motivation of Apache Spark.

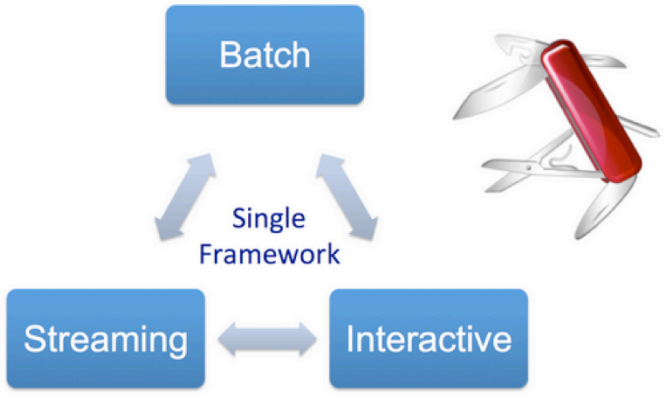
MapReduce is Too Slow for Some

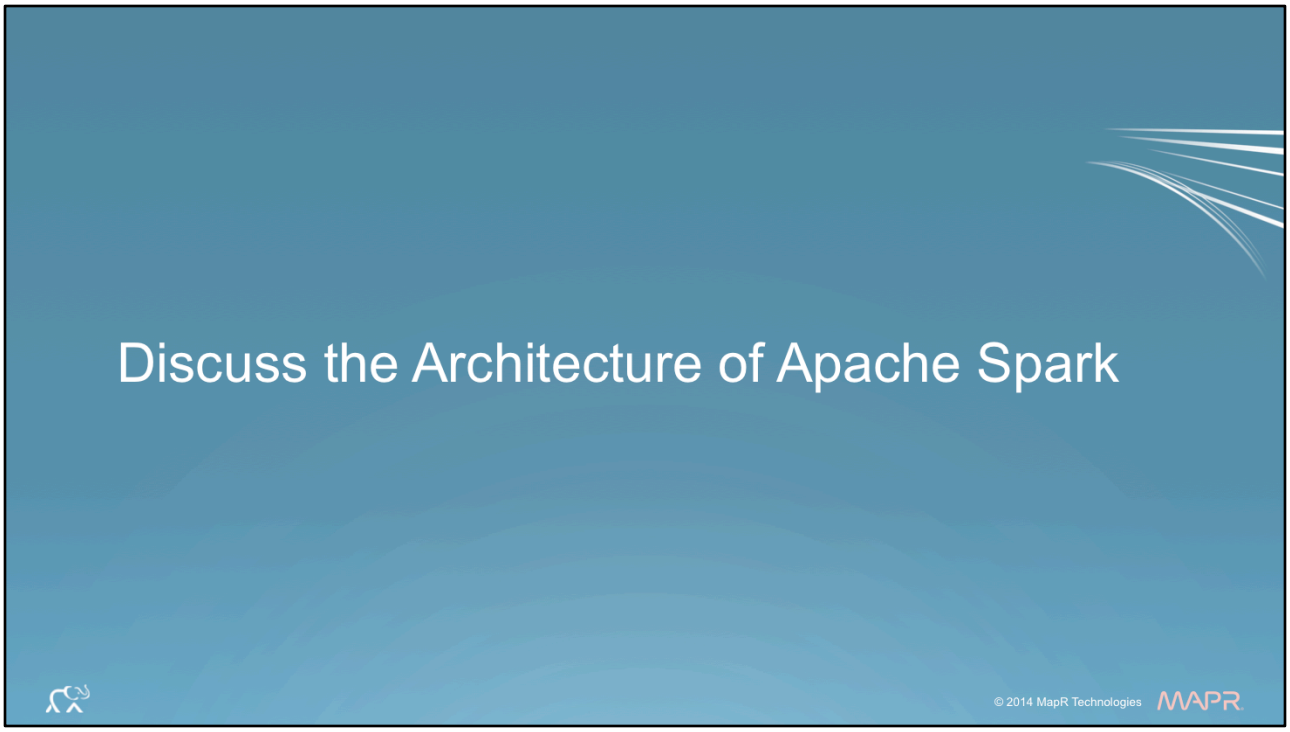


There are a lot of ecosystem tools to deal with



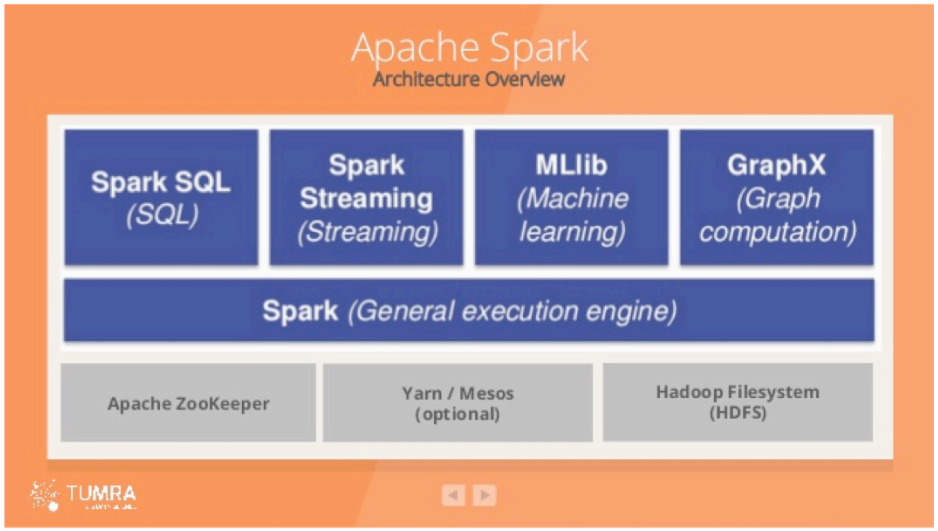
Wouldn't this be nice?



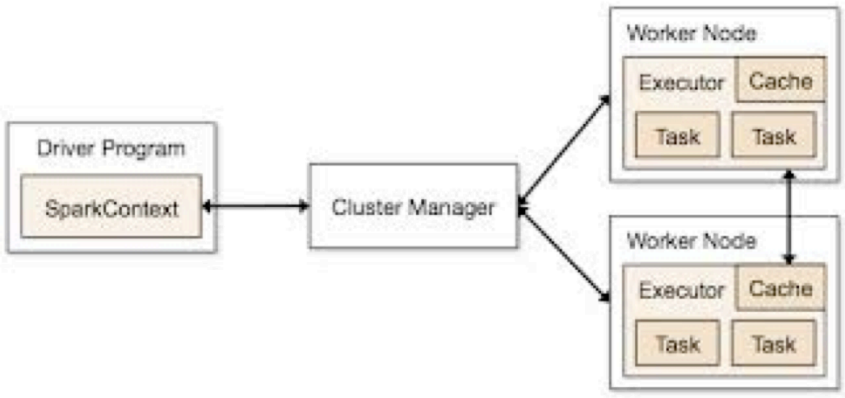


This section describes the architecture of Apache Spark.

High-Level Spark Architecture



Spark Job Execution



1. Each application gets its own executor processes which persist throughout the application and run in their own threads.
2. Spark is agnostic of the underlying cluster manager. Spark supports standalone mode, local mode, and running in a YARN cluster.
3. Jobs are submitted to Spark using the spark-submit script. One of the options to this script is the underlying cluster manager. The driver program runs the main() program and creates the spark context.

Spark is a polyglot platform

Deploying Spark Applications

```
./bin/spark-submit \
--class <main-class>
-- master <master-url> \
-- deploy-mode <deploy-mode> \
-- conf <key>=<value>
…
<application-jar> \
[application arguments]
```

--class : main entry point for application
--master : master url for the cluster (master, local, yarn, or mesos)
--deploy-mode : deploy driver on cluster nodes (cluster) or externally (local)
--application-jar : path to bundled jar file including driver and all job dependencies
application-arguments : args passed to main method, if any (e.g. executor-memory, total-executor-cores)

Spark job deployment options

- local
- local[n]
- spark://host:port
- yarn-client
- yarn-cluster

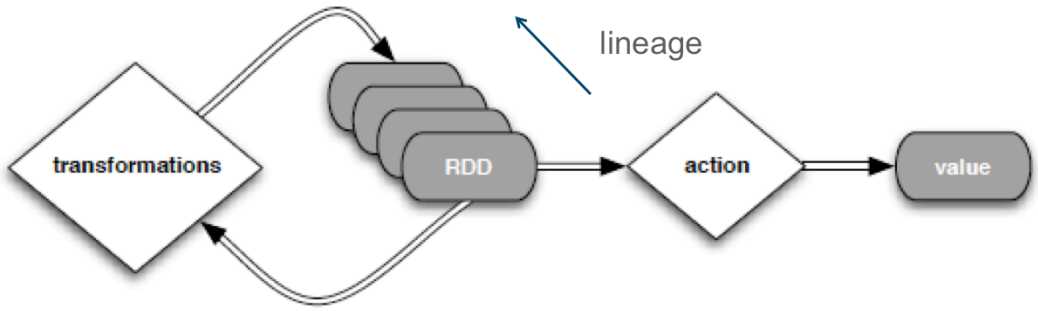
The spark master defines the context for the driver program which tells spark how to access the cluster.

This section discusses resilient distributed data sets (RDDs).

What is an RDD

- Primary abstraction in Spark
- Fault-tolerant collection of elements that can be operated on in parallel
- Can be persisted (to memory or disk)
- 2 types:
  - Parallelized collections: run parallel operations on existing Scala collection
  - Hadoop datasets: run functions on each record in HDFS file (or any file type supported in Hadoop like SequenceFile, Hbase table, S3 bucket,)

What can be done with RDD?



The Spark master/driver remembers transformations applied to an RDD. If a partition is lost (due to failure), that partition can easily be reconstructed on some other machine in the cluster. In fact, the entire lineage of the RDD is remembered, so however far back needs reconstruction will be supported.

What is a transformation?

- One type of operation on RDD
- Creates new data set from existing one
- Transformations are lazy (not computed immediately)
  - Optimized at runtime
  - Able to recover lost partitions
- Transformed RDD is computed when action is run on it

Examples of Transformations

- `map(func)` ➔ return new RDD by applying func on each element
- `flatMap(func)` ➔ returns new RDD (which is a sequence, not a value) by applying func on each element
- `filter(func)` ➔ create new RDD by selecting data on which func returns true
- `reduceByKey(func, numTasks)` ➔ aggregate values of a key using a function (number of reduce tasks is optional)
- `join(otherRDD, numTasks)` ➔ creates new RDD by joining this RDD to other RDD on common keys

What is an action?

- The other type of operation on an RDD
- Operation that does not transform the data
- Some operations return results while others persist RDD

Action Examples

- `saveAsTextFile(path)` ➔ write contents of RDD to text file
- `foreach(func)` ➔ execute func for each data element in RDD
- `reduce(func)` ➔ aggregate data elements in RDD using this function
- `collect()` ➔ return all data elements in RDD as an array
- `count()` ➔ return number of data elements in RDD

So what makes Spark faster than Hadoop MapReduce?

- Both Spark and Hadoop M/R jobs are run serially
- Hadoop M/R jobs store each intermediate result on disk ☹
- Spark jobs may store intermediate results in memory ☺