Spark (and Hadoop)

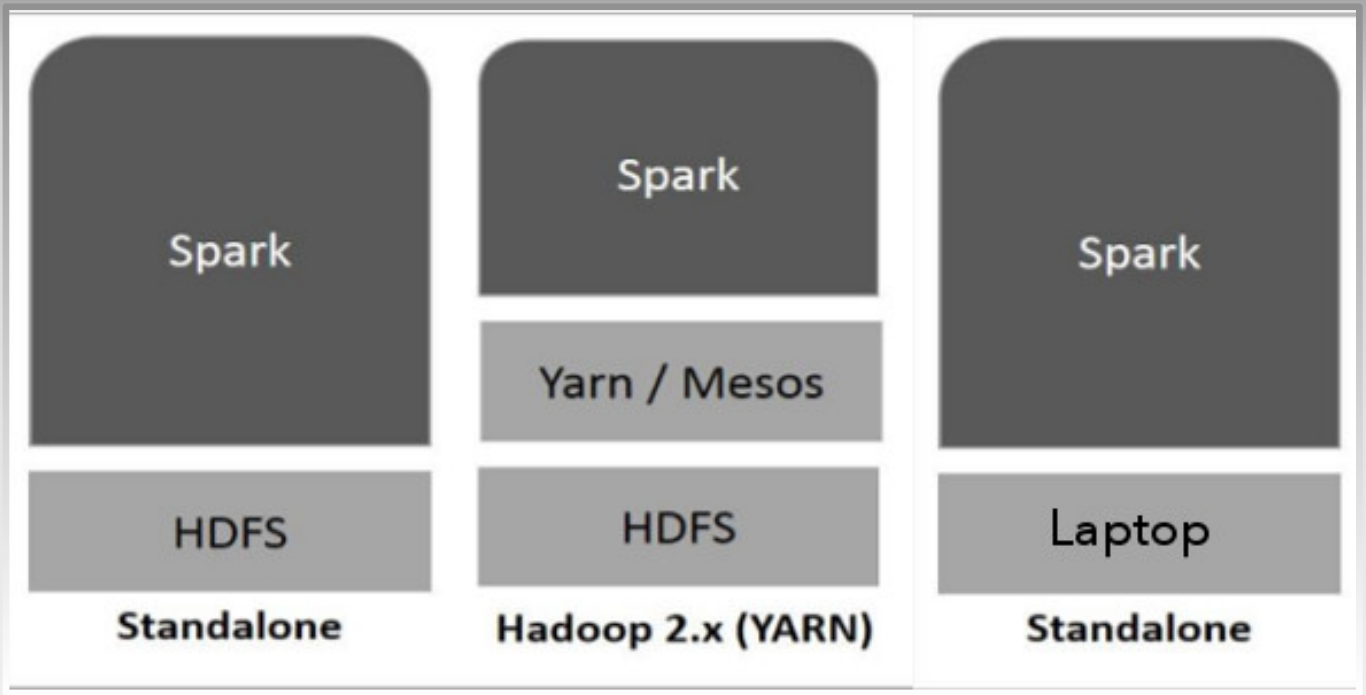
- Developed in 2009 in UC Berkeley's AMPLab by Matei Zaharia.
- It was Open Sourced in 2010 under a BSD license.
- It was donated to Apache software foundation in 2013
- Similar to Hadoop MapReduce, but is independent project

BASEMENT SUPERCOMPUTING

What is So Special About Spark?

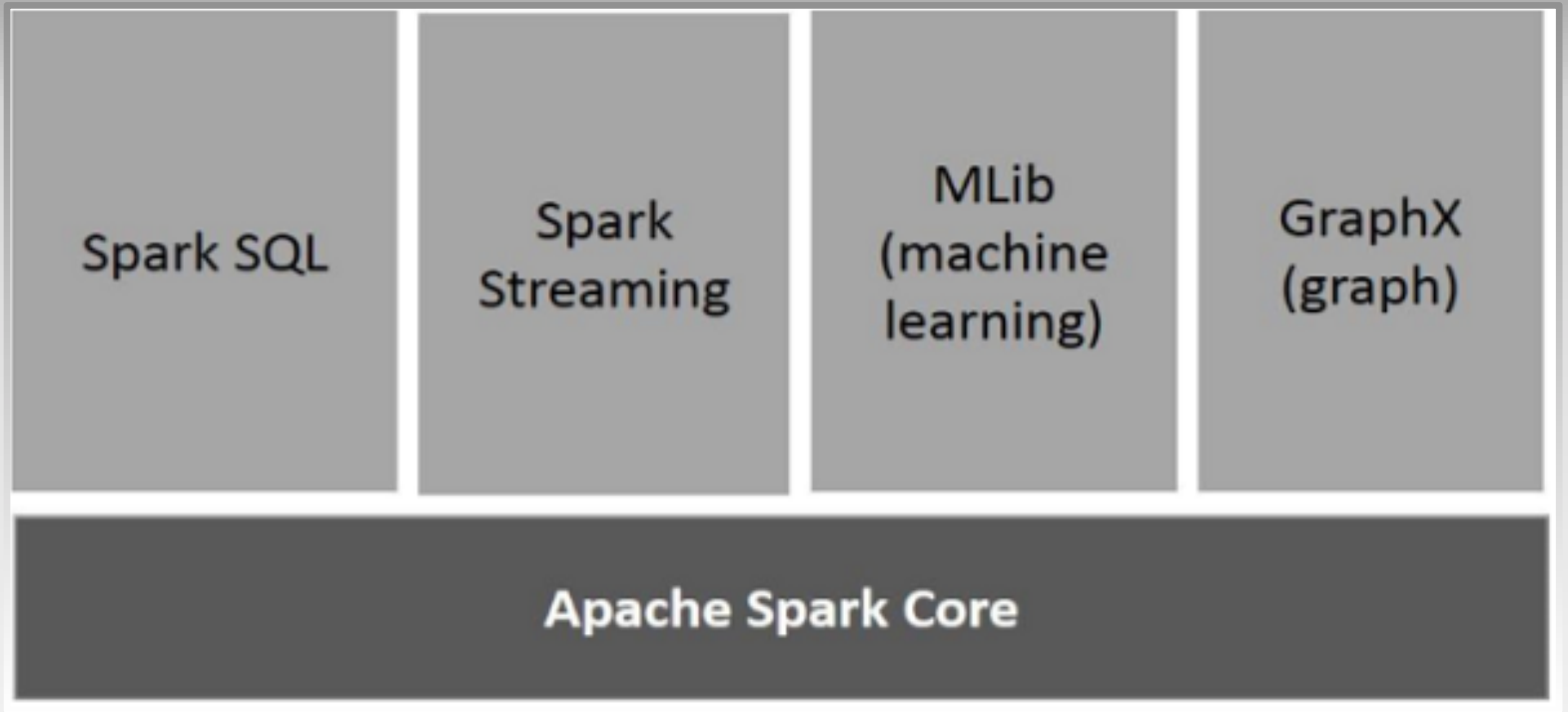
- **Speed** – Spark keeps intermediate results in memory. Many analytics jobs consist of stages, traditional MapReduce writes results to disk between stages. Spark stores the intermediate processing data in memory.
- **Supports Multiple Languages** – Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark includes 80 high-level operators for interactive querying.
- **Advanced Analytics** – Spark not only supports "Map" and "Reduce." It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms. It is higher level than native MapReduce, it can be used in place of Hive and/or Pig

Spark Deployment



BASEMENT SUPERCOMPUTING

Components of Spark



BASEMENT SUPERCOMPUTING

Spark Components

- **Apache Spark Core -** Spark Core is the underlying general execution engine for spark
- **Spark SQL -** Spark SQL is a component on top of Spark Core that introduces a new data abstraction called SchemaRDD, which provides support for structured and semi-structured data.
- **Spark Streaming -** Spark Streaming leverages Spark Core's fast scheduling capability to perform streaming analytics.
- **MLlib (Machine Learning Library) -** MLlib is a distributed machine learning framework
- **GraphX -** GraphX is a distributed graph-processing framework on top of Spark. It provides an API for expressing graph computation.

BASEMENT SUPERCOMPUTING

Spark RDDs

- Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark.
- It is an immutable distributed collection of objects.
- Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster.
- RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Spark Operations: Transformations and Actions

- **RDD Transformations** return a pointer to new RDD . The original RDD cannot be changed. Spark is lazy, so nothing will be executed unless a transformation or action is called.

  An RDD transformation is not a set of data, but is a step in a program (might be the only step) telling Spark how to get data and what to do with it.
- **RDD Actions** return values (e.g. collect, count, take, save-as).

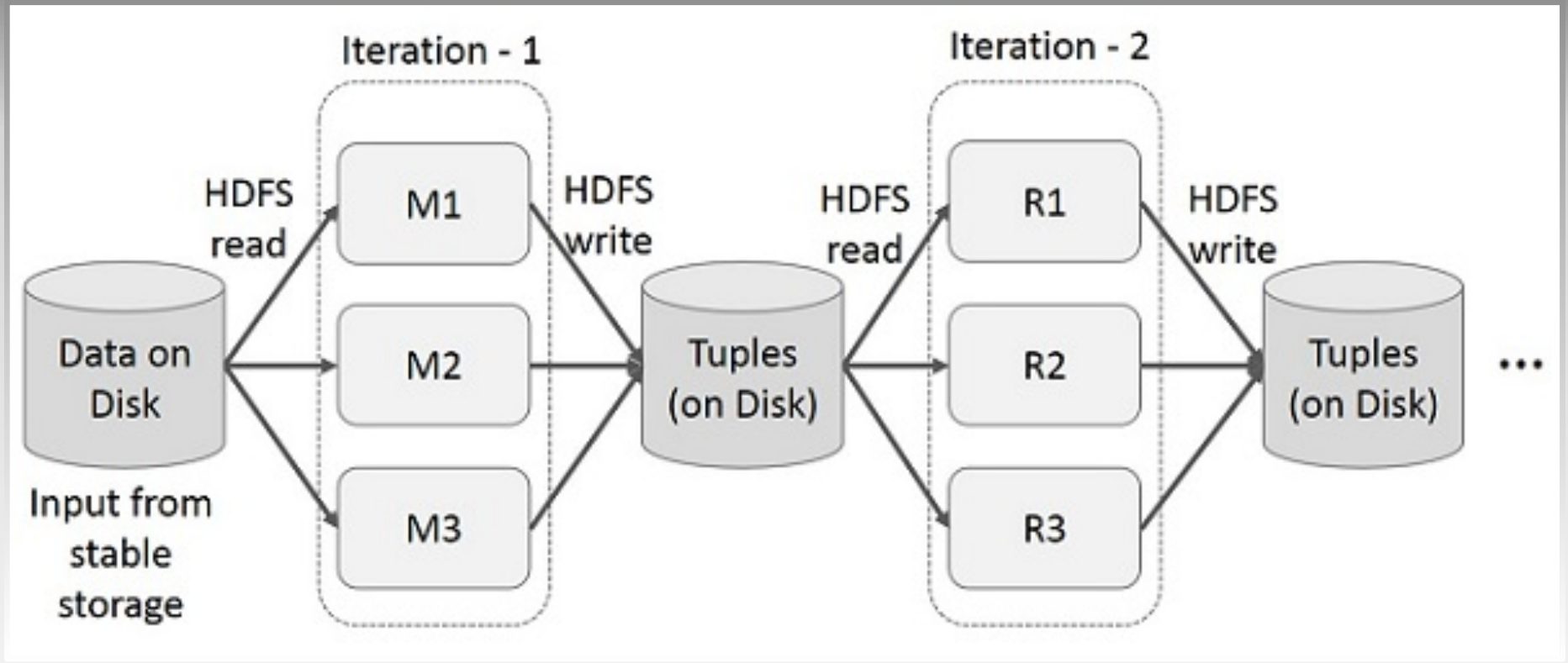
BASEMENT SUPERCOMPUTING

Spark RDD vs. Data Frame

- An **RDD** is blind structure partitioned across the nodes of the cluster and provides many transformation methods, such as map(), filter(), and reduce(). Each of these methods results in a new RDD representing the transformed data.
- The **DataFrame** introduces the concept of a schema to describe the data (named columns), allowing Spark to manage and optimize computation across nodes. Conceptually equivalent to a table in a relational database or a R/Python Dataframe.
- Think of a **DataFrame** as a distributed database table and an **RDD** as distributed raw data. (new: DataSets)

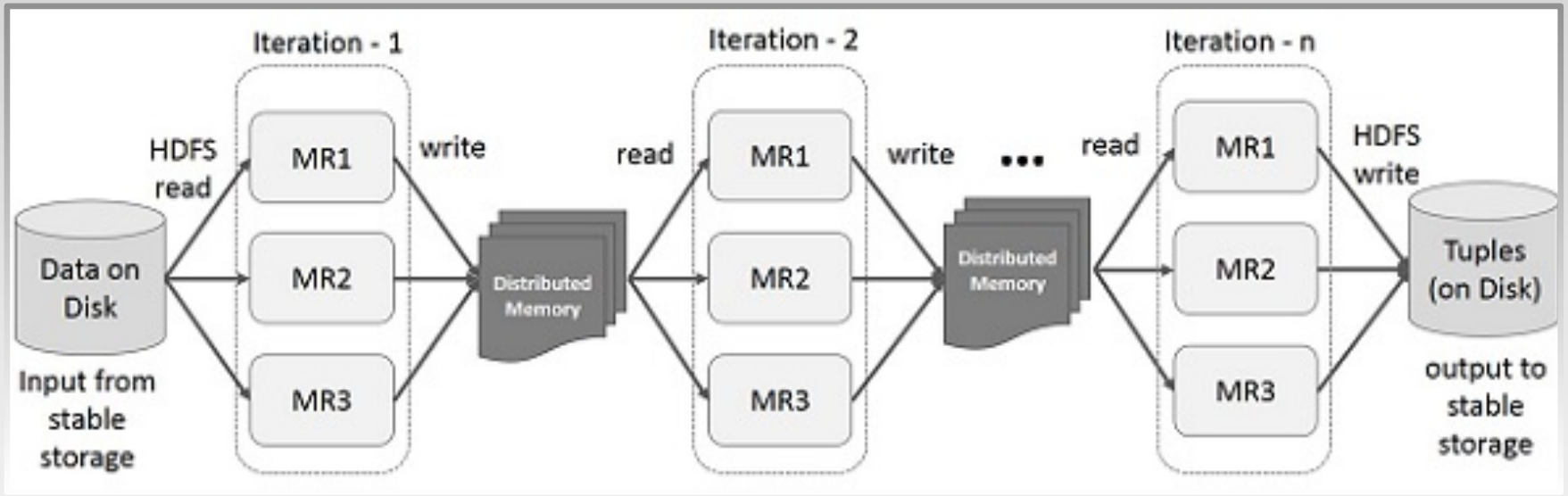
BASEMENT SUPERCOMPUTING

Iterative Operations Using Traditional MapReduce (Batch Mode)



BASEMENT SUPERCOMPUTING

Iterative Operations Using Spark RDD



BASEMENT SUPERCOMPUTING

In Reality, However

- Hadoop MapReduce has been optimized using Tez (keeps tuples in memory) Pig and Hive SQL have become more interactive (like Spark) and less like batch jobs.

- Spark uses “in memory” computing, but if it runs out of memory then intermediate results will spill to disk. And if the job does not fit in memory, then it is back to Hadoop MapReduce.

BASEMENT SUPERCOMPUTING

Spark Code Example (Pi estimation)

```
from pyspark import SparkContext
from numpy import random
n=5000000

def sample(p):
    x, y = random.random(), random.random()
    return 1 if x*x + y*y < 1 else 0

count = sc.parallelize(xrange(0,n)).map(sample) \
.reduce(lambda a, b: a + b)

print "Pi is roughly %f" % (4.0 * count / n)
```

MapReduce Java Pi Estimation

```
      /**
         * Mapper class for Pi estimation. /**
         * Mapper class for Pi estimation.
         * Generate points in a unit square
         * and then count points inside/outside of the inscribed circle of the squa
         */
        public static class QmcMapper extends
            Mapper<LongWritable, LongWritable, BooleanWritable, LongWritable> {

          /** Map method.
           * @param offset samples starting from the (offset+1)th sample.
           * @param size the number of samples for this map
           * @param context output {ture->numInside, false->numOutside}
           */
          public void map(LongWritable offset,
                          LongWritable size,
                          Context context)
              throws IOException, InterruptedException {

            final HaltonSequence haltonsequence = new HaltonSequence(offset.get());
            long numInside = 0L;
            long numOutside = 0L;

            for(long i = 0; i < size.get(); ) {
              //generate points in a unit square
              final double[] point = haltonsequence.nextPoint();

              //count points inside/outside of the inscribed circle of the square
              final double x = point[0] - 0.5;
              final double y = point[1] - 0.5;
              if (x*x + y*y > 0.25) {
                numOutside++;
              } else {
                numInside++;
              }

              //report status
              i++;
              if (i % 1000 == 0) {
                context.setStatus("Generated " + i + " samples.");
              }
            }

            //output map results
            context.write(new BooleanWritable(true), new LongWritable(numInside));
            context.write(new BooleanWritable(false), new LongWritable(numOutside))
          }
        }

        /**
         * Reducer class for Pi estimation.
         * Accumulate points inside/outside results from the mappers.
         */
        public static class QmcReducer extends
            Reducer<BooleanWritable, LongWritable, WritableComparable<?>, Writable>

          private long numInside = 0;
```

```
          /**
           * Reduce task done, write output to a file.
           */
          @Override
          public void cleanup(Context context) throws IOException {
            //write output to a file
            Path outDir = new Path(TMP_DIR, "out");
            Path outFile = new Path(outDir, "reduce-out");
            Configuration conf = context.getConfiguration();
            FileSystem fileSys = FileSystem.get(conf);
            SequenceFile.Writer writer = SequenceFile.createWriter(fileSys, conf,
                outFile, LongWritable.class, LongWritable.class,
                CompressionType.NONE);
            writer.append(new LongWritable(numInside), new LongWritable(numOutside));
            writer.close();
          }
        }
         * Generate points in a unit square
         * and then count points inside/outside of the inscribed circle of the square.
         */
        public static class QmcMapper extends
            Mapper<LongWritable, LongWritable, BooleanWritable, LongWritable> {

          /** Map method.
           * @param offset samples starting from the (offset+1)th sample.
           * @param size the number of samples for this map
           * @param context output {ture->numInside, false->numOutside}
           */
          public void map(LongWritable offset,
                          LongWritable size,
                          Context context)
              throws IOException, InterruptedException {

            final HaltonSequence haltonsequence = new HaltonSequence(offset.get());
            long numInside = 0L;
            long numOutside = 0L;

            for(long i = 0; i < size.get(); ) {
              //generate points in a unit square
              final double[] point = haltonsequence.nextPoint();

              //count points inside/outside of the inscribed circle of the square
              final double x = point[0] - 0.5;
              final double y = point[1] - 0.5;
              if (x*x + y*y > 0.25) {
                numOutside++;
              } else {
                numInside++;
```

• • •

BASEMENT SUPERCOMPUTING