These are the links in this lecture:

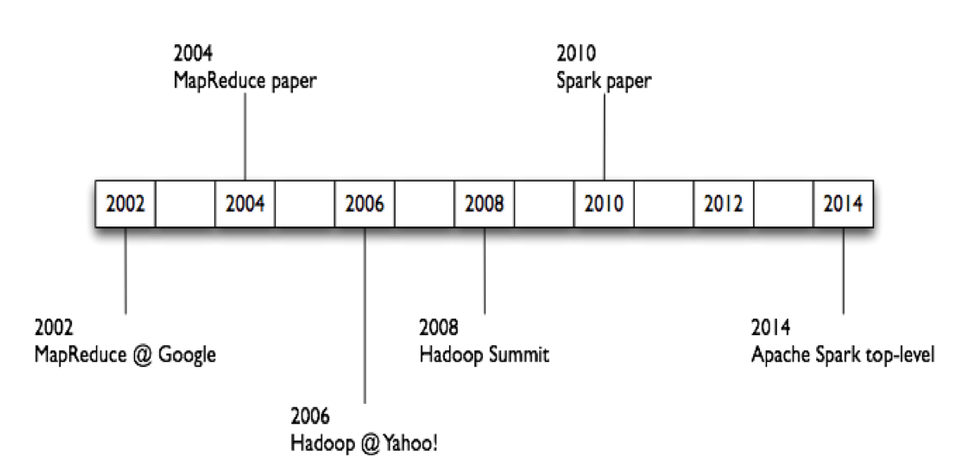
* + <http://www-formal.stanford.edu/jmc/history/lisp/lisp.html>
  + [Google: MapReduce: Simplified Data Processing on Large Clusters](http://research.google.com/archive/mapreduce.html)
  + [Apache Hadoop](http://research.yahoo.com/files/cutting.pdf)
  + [Yahoo! web scale search indexing](http://developer.yahoo.com/hadoop/)
  + [Amazon Web Services Elastic MapReduce](http://aws.amazon.com/elasticmapreduce/)
  + [Spark: Cluster Computing with Working Sets](http://people.csail.mit.edu/matei/papers/2010/hotcloud_spark.pdf)
  + [*Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing*](http://usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf)

Map Reduce deals with failures and slow tasks by re-launching the tasks on other machines. This functionality is enabled by the requirement that individual tasks in a Map Reduce job are idempotent and have no side effects. These two properties mean that given the same input, re-executing a task will always produce the same result and will not change other state. So, the results and end condition of the system are the same, whether a task is executed once or a thousand times.

Using memory instead of disks offers two huge benefits. The first benefit is that memory is much faster than disks. The time to read or write a value to memory is only a few nanoseconds, while the time to read or write is several milliseconds - that means memory is a million times faster than disks. The second benefit is that keeping intermediate results in memory means that they do not have to be converted into a format that can be stored on disks. The process of converting a memory object to a disk object is called serialization and the process of converting a disk object to a memory object is called deserialization. Serializing and deserializing objects is a very expensive and time consuming process. Keeping intermediate results in memory avoids this significant overhead.

Taken together, the faster access times and avoidance of serialization/deserialization overhead make Spark much faster than Map Reduce - up to 100 times faster!

History Review



* 1956-1979: Stanford, MIT, CMU, and other universities develop set/list operations in LISP, Prolog, and other languages for parallel processing (see <http://www-formal.stanford.edu/jmc/history/lisp/lisp.html>).
* Circa 2004: [Google: MapReduce: Simplified Data Processing on Large Clusters](http://research.google.com/archive/mapreduce.html)  
   by Jeffrey Dean and Sanjay Ghemawat
* Circa 2006: [Apache Hadoop](http://research.yahoo.com/files/cutting.pdf), originating from the Yahoo!’s Nutch Project  
   Doug Cutting
* Circa 2008: [Yahoo! web scale search indexing](http://developer.yahoo.com/hadoop/)  
   - Hadoop Summit, Hadoop User Group
* Circa 2009: Cloud computing with [Amazon Web Services Elastic MapReduce](http://aws.amazon.com/elasticmapreduce/) (AWS EMR)  
  , a Hadoop version modified for Amazon Elastic Cloud Computing (EC2) and Amazon Simple Storage System (S3), including support for Apache Hive and Pig.

Selected Research Papers

* [Spark: Cluster Computing with Working Sets](http://people.csail.mit.edu/matei/papers/2010/hotcloud_spark.pdf),   
  Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica.   
  USENIX HotCloud (2010)  
  .
* [Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing](http://usenix.org/system/files/conference/nsdi12/nsdi12-final138.pdf)*,*  
  Matei Zaharia, Mosharaf Chowdhury, Tathagata Das,   
  Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin,   
  Scott Shenker, Ion Stoica  
  . NSDI (2012)
* [MLlib: Machine Learning in Apache Spark](http://arxiv.org/pdf/1505.06807.pdf), X. Meng, J. Bradley, B. Yuvaz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, D. Xin, R. Xin, M. Franklin, R. Zadeh, M. Zaharia, A. Talwalkar. Preprint (2015).

These are the links in this lecture:

* + [Spark Documentation](https://spark.apache.org/documentation.html)
  + [Spark Programming Guide](https://spark.apache.org/docs/latest/programming-guide.html)
  + [pySpark API documentation](https://spark.apache.org/docs/latest/api/python/index.html)
  + [Python Documentation](https://docs.python.org/2/)
  + [Download Python Documentation](https://docs.python.org/2/download.html)

# Apache Spark References

For more information about Apache Spark, you should refer to the online [Spark Documentation](https://spark.apache.org/documentation.html). The documentation includes screencasts, training materials, and hands-on exercises. The [Spark Programming Guide](https://spark.apache.org/docs/latest/programming-guide.html) is anothe good starting point, and the [pySpark API documentation](https://spark.apache.org/docs/latest/api/python/index.html" \t "_blank) is a great reference to use when writing Spark programs.

A very useful reference when writing pySpark applications is the [Python Documentation](https://docs.python.org/2/) site. You can even[download](https://docs.python.org/2/download.html) the reference documentation for later reference.