

INTRODUCTION TO APACHE SPARK

1. DISTRIBUTED COMPUTING AND BIG DATA

CONTEXT

Moore's Law: computing power doubled every two years from 1975 to 2012. Nowadays, every two and a half year.

Rapid growth of datasets: internet activity, genomics, astronomy, censor networks, etc.

Data size trends: doubles every year according to [IDC executive summary](#).

Data grows faster than Moore's law. How do we scale the training of a statistical model?

SCALING-UP

Keywords: *High Performance Computing (HPC), parallel computing*

Scaling-up means using a bigger machine. Can lead to huge performance increase for medium scale problems.

Very expensive, requires specialized hardware able to handle lots of processors and memory.

Challenges: side effects.

An expression has a side effects if it modifies some state or has an observable interaction with calling functions or the outside world.

Solution? Locks to limit access to resources, which ensures that only one thread is accessing a specific resource all at once.

Some rare algorithms, such as [Hogwild!](#), exploit side effects to improve performance.

Does not scale at some point to very big datasets.

WAIT... WHAT IS BIG? SOME FIGURES. (FROM 2015)..

Big data examples:

- facebook daily logs: 60TB
- 1000 genomes project: 200TB
- Google web index: 100+ PB

Cost of 1Tb of storage: 35—50

Time to read 1Tb from disk: ~3hours (100MB/s)

Data is streamed from the disk to the different layers of memory.

Problem: disks cannot be read in parallel... Use several disks.

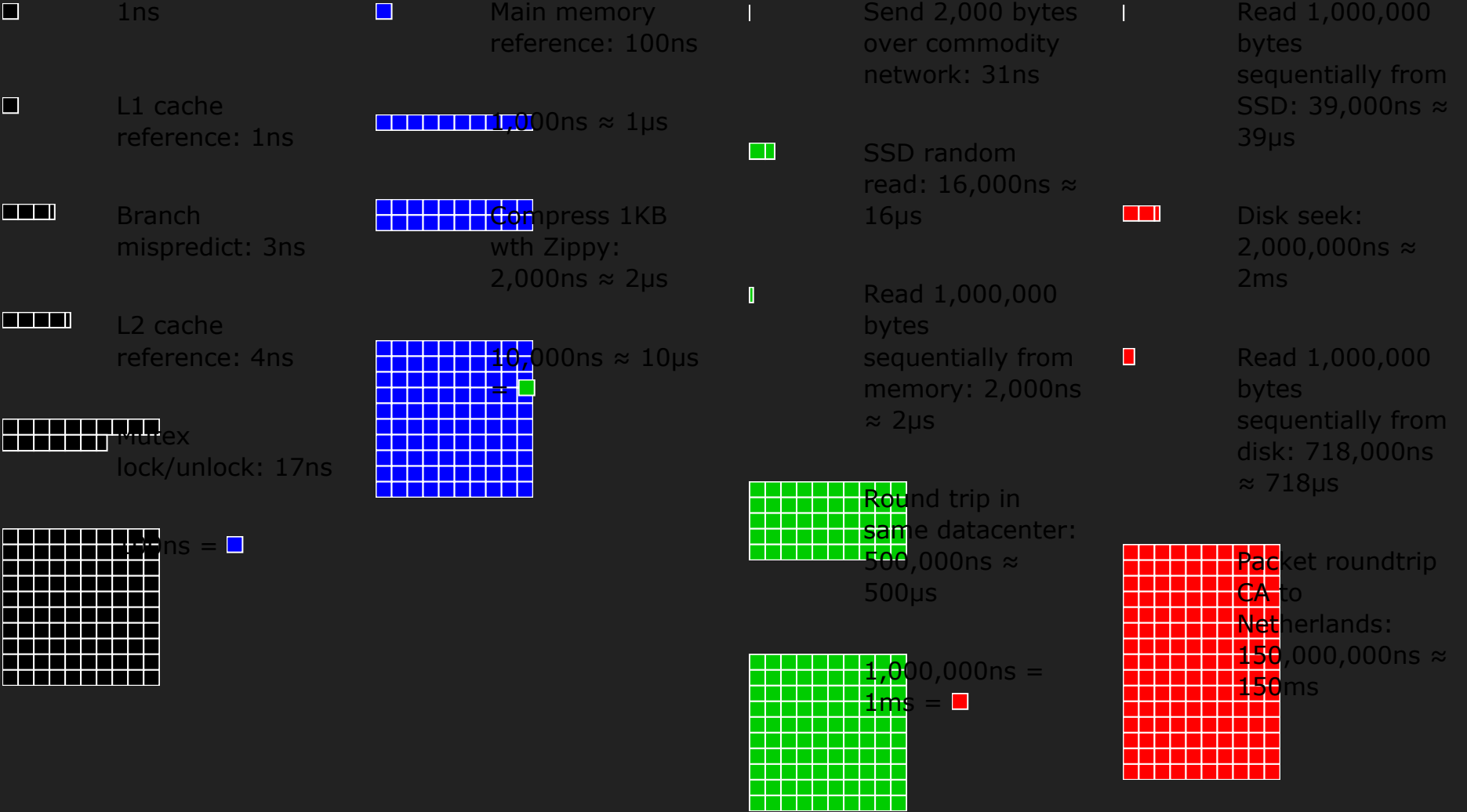
Limited number of disks inside one machine? Use several machines: distributed computing!

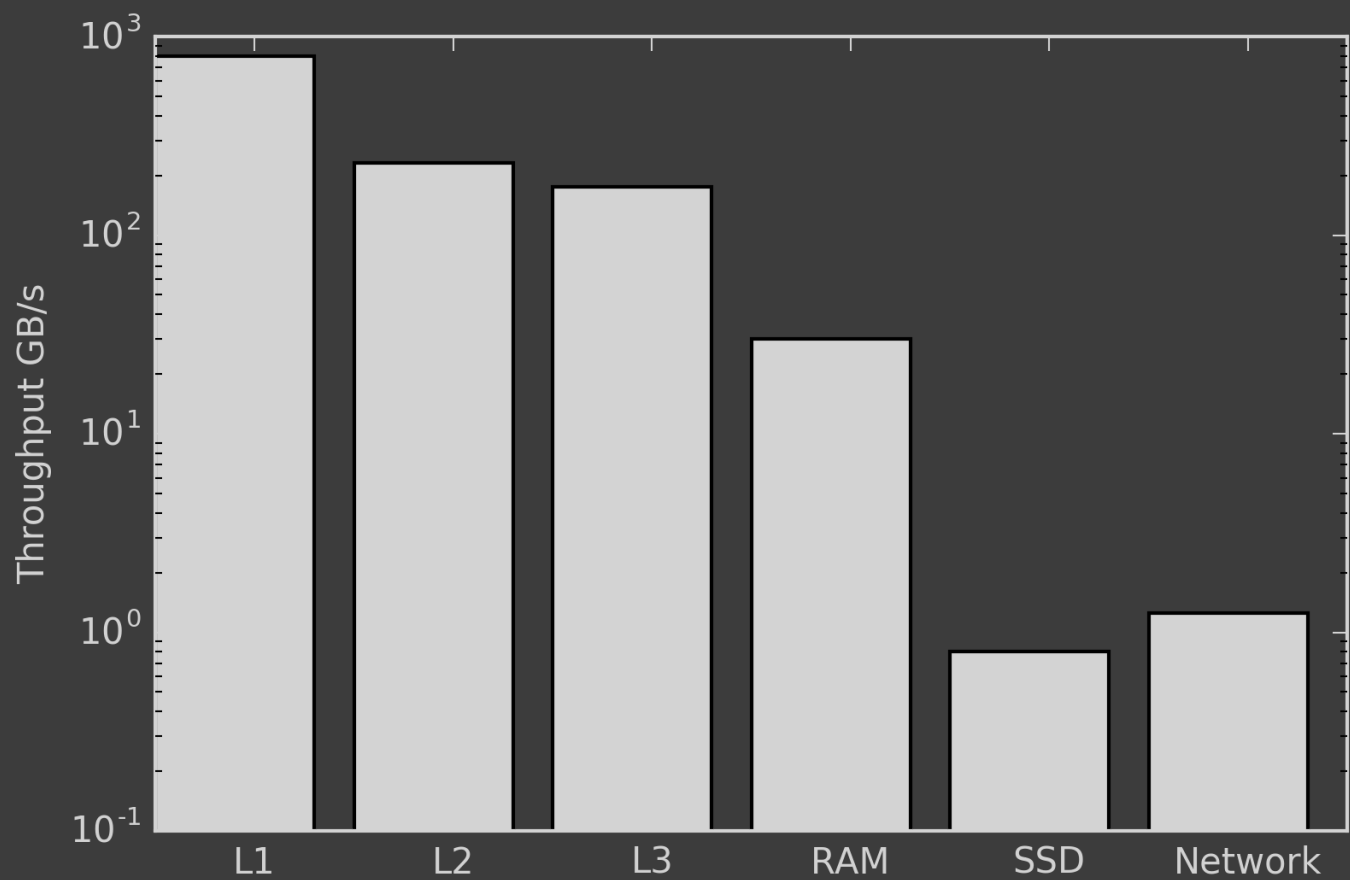
When one machine can no longer process or even store all the data, use distributed computing.

Depending on the task, distributed computing starts to be useful with 1TB datasets. Below that, a large multicore workstation or server might be able to handle it.

Latency Numbers Every Programmer Should Know

2020





SCALING-OUT

Keywords: *distributed computing, High-throughput computing*

Scale-out means using many small machines.

Uses commodity hardware: cheap, common architecture i.e. processor + RAM + disk.

Problem: dealing with network computation adds software complexity.

Challenges:

- Scheduling: How to split the work across machines? Must consider network, data locality as moving data may be very expensive.
- Reliability: How to deal with failures? Commodity (cheap) hardware fails more often. At Google, 1-5% hard drive failure per year. 0.2% DIMM failure/year.
- Uneven performance of the machines: some nodes (*stragglers*) are slower than others.

PERFORMANCE MEASURES

How to evaluate the performance of a scaling? Let T_J be the time complexity of an algorithm using J processing elements.

$$\text{Speedup} = \frac{T_1}{T_N} \leq N$$

Measures the benefits of using parallelism.

$$\text{Efficiency} = \frac{\text{Speedup}}{N} \leq 1$$

Provides an indication of the effective utilization of all the processing units.

Strong scaling:

How to compute faster? Relevant when the task is CPU-bound.

Fixed problem size, increased number of processing elements.

In such case, speedup and efficiency can be used to measure the performance gain and tune the number of processing units.

Weak scaling:

How to compute using more data? Relevant when the task is I/O-bound.

Problem size (workload) assigned to each processing element stays constant. Additional processing elements are used to solve a larger total problem.

In this case, efficiency does not make sense, as we assume a constant workload.

TOOLS

In practice, softwares such as [Spark](#) or [HadoopMR](#) are in charge of these problems.

They are *distributed compute engines*, i.e. softwares that ease the development of distributed algorithms.

They run on clusters, managed by a resource manager, such as [YARN](#) or [Mesos](#)

In short, resource managers ensures that the different tasks running on the cluster do not try to use the same resources all at once.

HADOOP MAPREDUCE

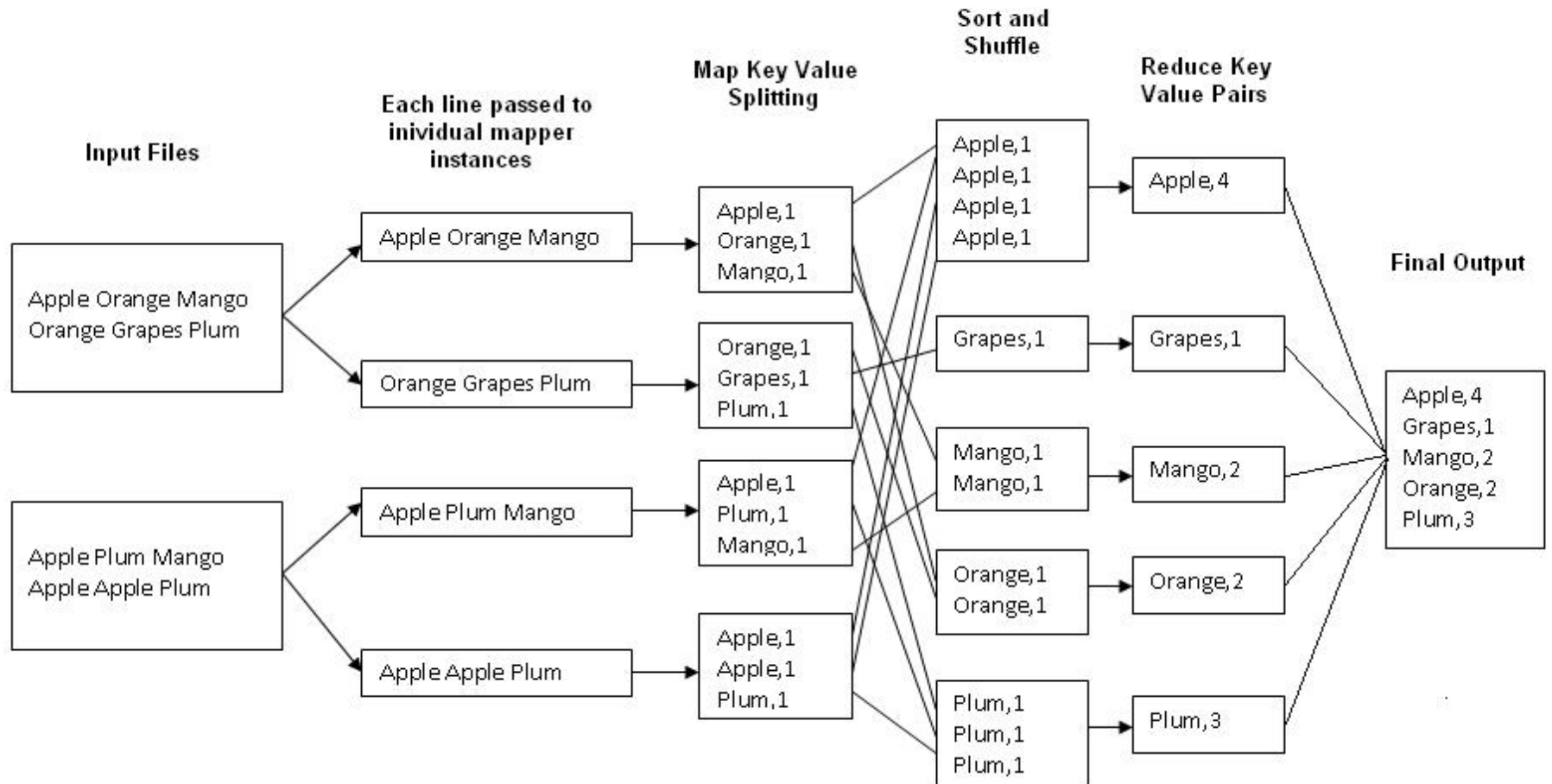
Hadoop is older, more enterprise-grade codebase (e.g. security with Kerberos)

Good for data crunching (e.g. data cleaning, ETL: Extract, Transform, Load).

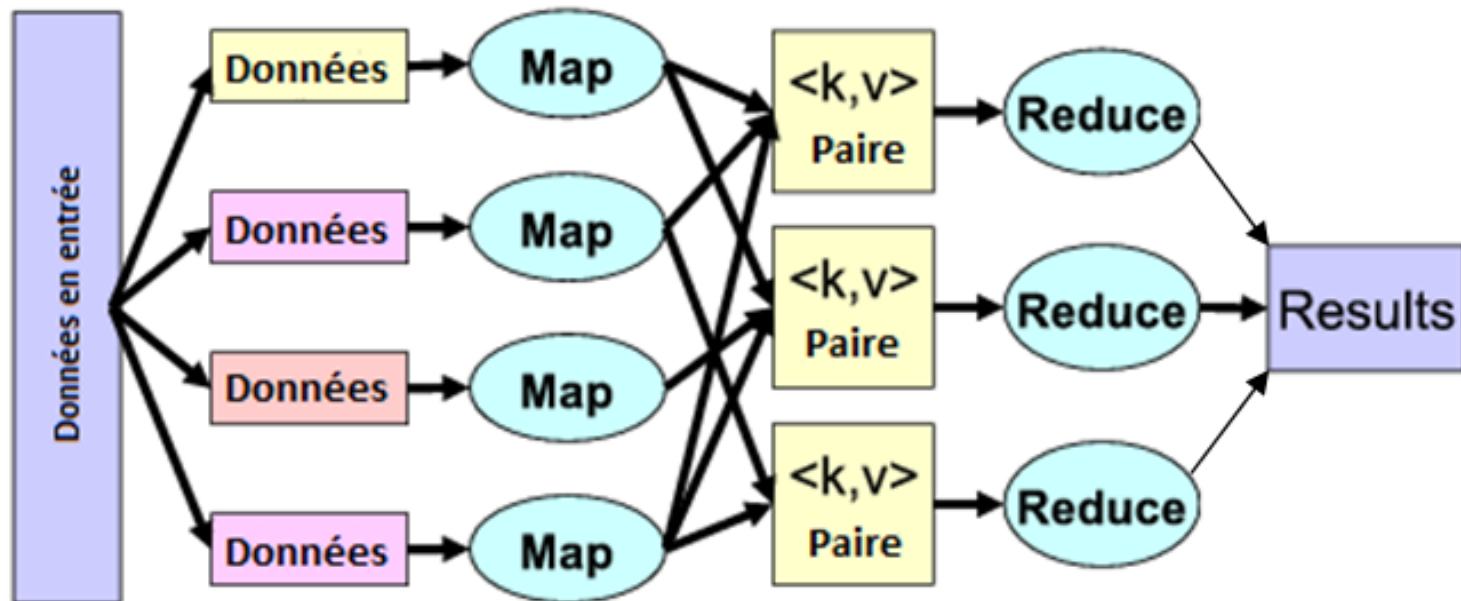
Problems: lots of disk I/O

For Machine Learning problems, this is a real problem as iterative algorithms need to access such as gradient values, parameters vector, etc. very often.

MAPREDUCE'S WORDCOUNT EXAMPLE



MAPREDUCE'S WORKFLOW



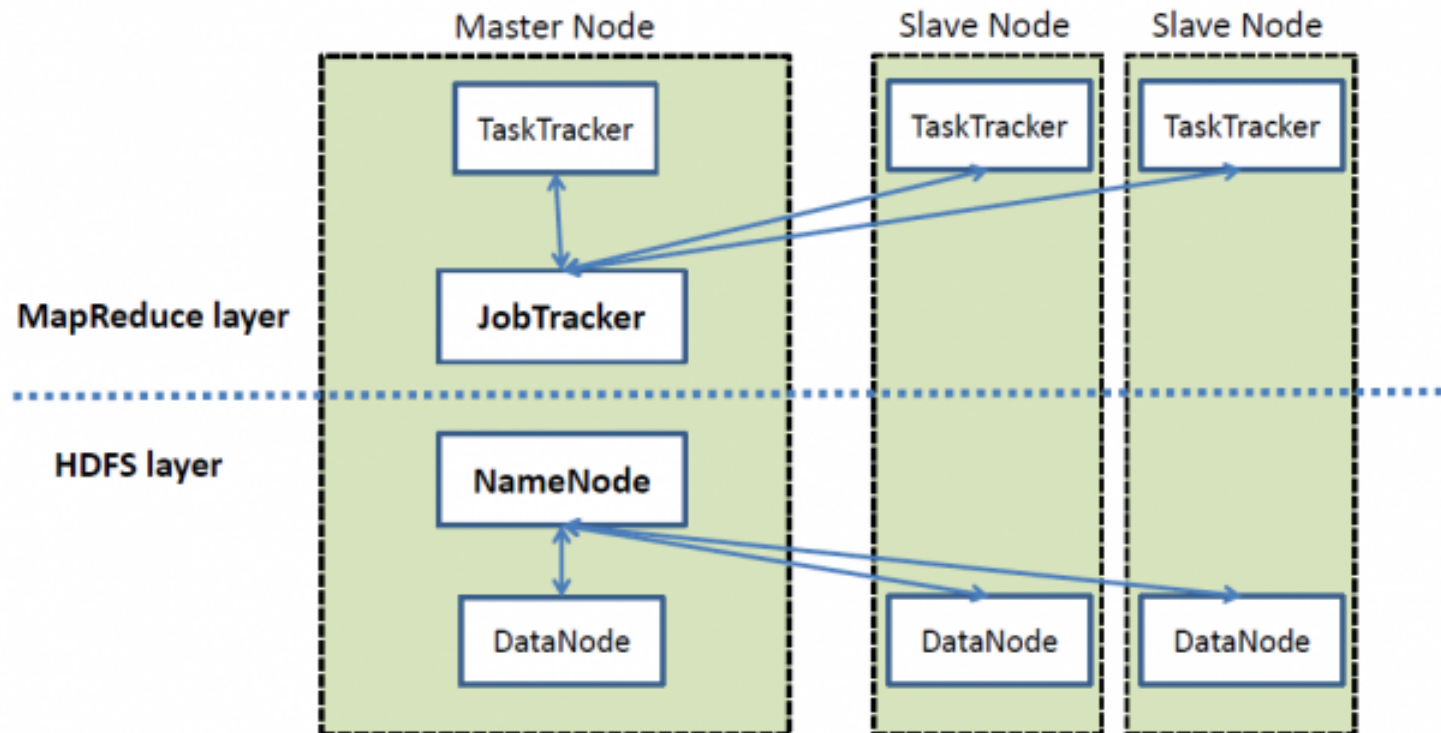
HADOOP ARCHITECTURE

Hadoop is made of three components :

- HDFS (Highly Distributed File System) inspired from [GoogleFileSystem](#)
- YARN (Yet Another Resource Negotiator)
- MapReduce inspired from [Google](#)

HADOOP ARCHITECTURE

High Level Architecture of Hadoop



SPARK

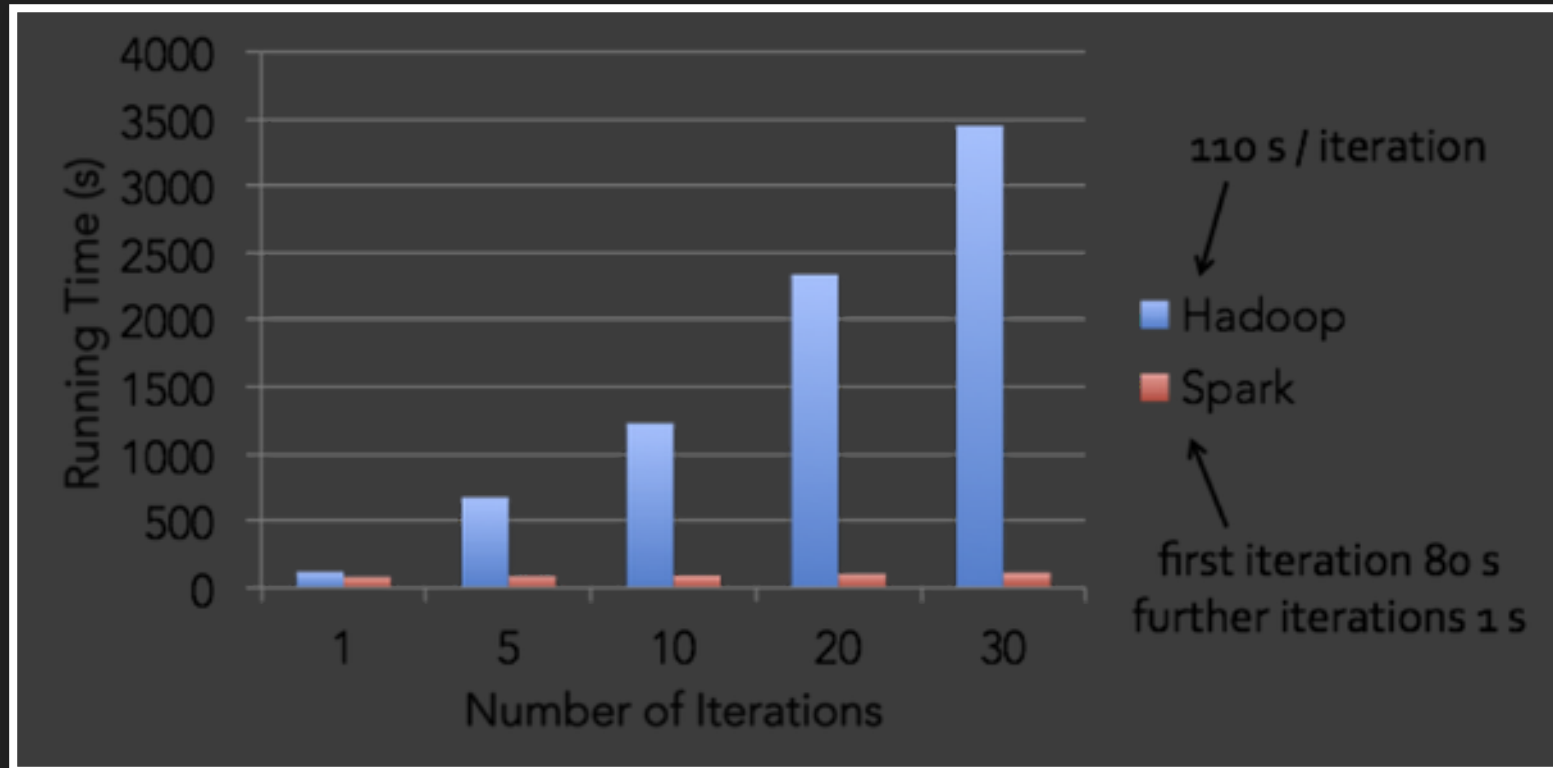
HadoopMR is designed for acyclic data flow models while Spark handles cyclic (e.g. iterative) data flows.

Advantage of Spark over HadoopMR ?

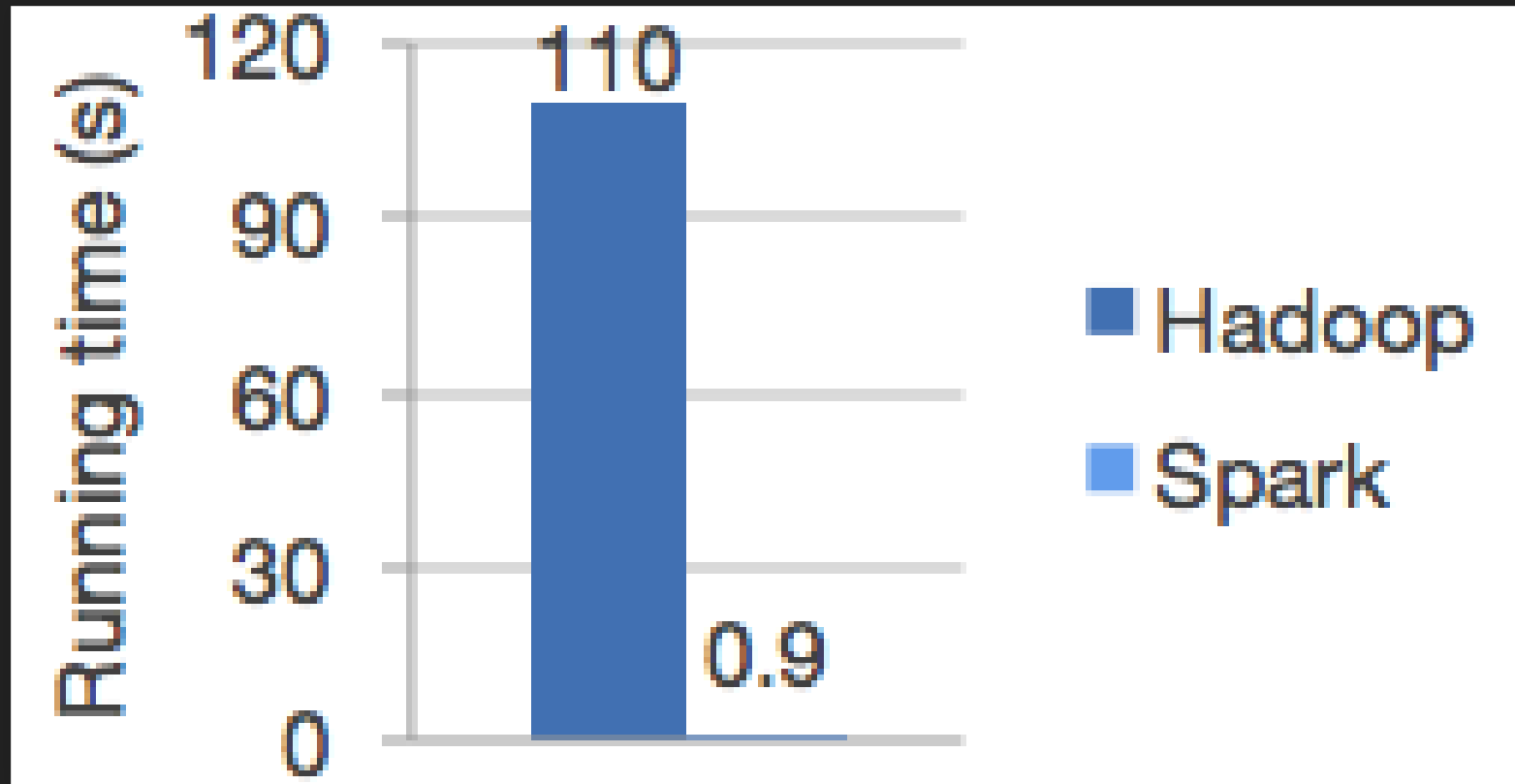
- Use RAM, i.e. fast iterative computations
- lower overhead for starting jobs
- simple & expressive (scala, python + interactive shell)
- higher level libraries (SparkSQL, SparkStreaming, MLlib, GraphX)

Requires servers with more CPU and more memory (more expensive), but still quite cheap compared to HPC.

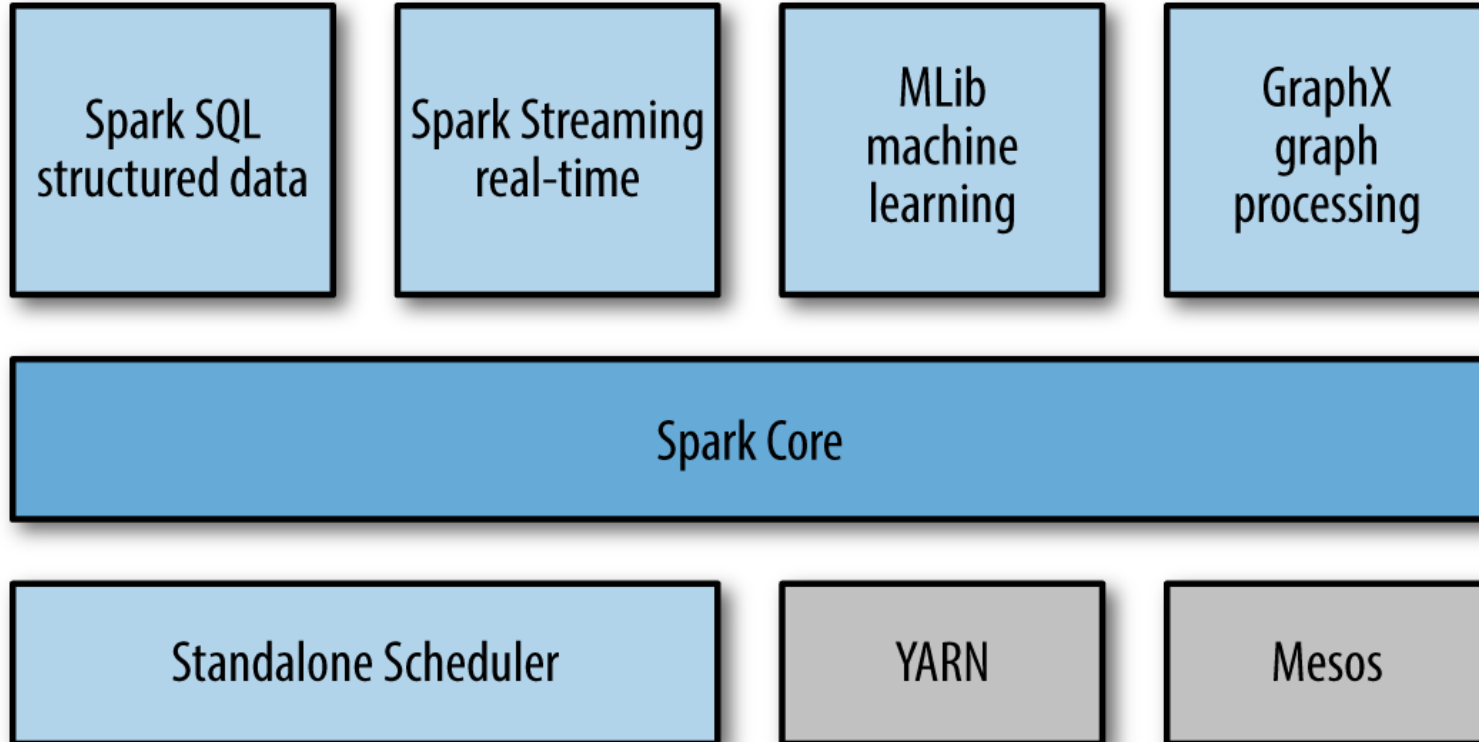
ITERATION SPEED COMPARISON



LOGISTIC REGRESSION SPEED COMPARISON



SPARK STACK



SPARK AND HADOOP COMPARISON:

	HadoopMR	Spark
storage	Disk	in-memory or disk
operations	Map, reduce	Map, reduce, join, sample...
execution model	Batch	Batch, interactive, streaming
Programming environments	Java	Scala, Java, Python, R

FIRST LAB: PYTHON RECAP

BONUS

Wonder what a datacenter **looks like?**

Explore a Google data center with Street View

