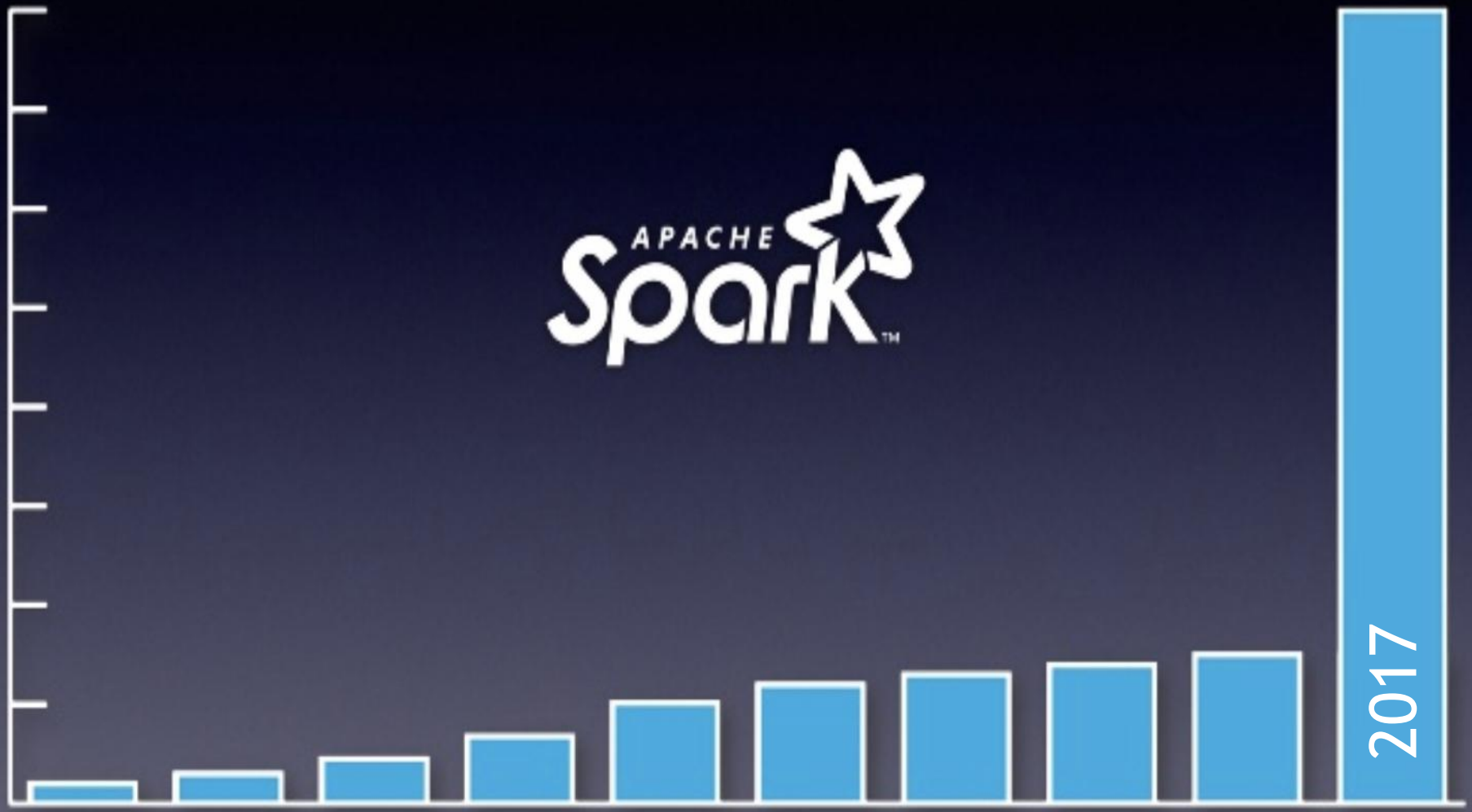

APACHE
Spark™
2017

Apache Spark

- **Fast** and general engine for large scale data processing.
- Initially started at UC Berkeley in 2009
- Written in Scala and lives in Github!
- Supports different file systems - HDFS, AWS S3 etc. [Easy data loading]
- Provides high level APIs in Java, Python, Scala and R
- Most popular for *stream processing*.

What’s different?

- **Unified Engine**
  - Creating data processing pipelines are easier because of unified APIs
  - Different tasks on same system - Core libraries for ML, SQL, Graph & Streaming

- **In-memory processing**
  - Uses RAM more than the hard disks
  - Lazy Loading - Intelligent data loading and persisting

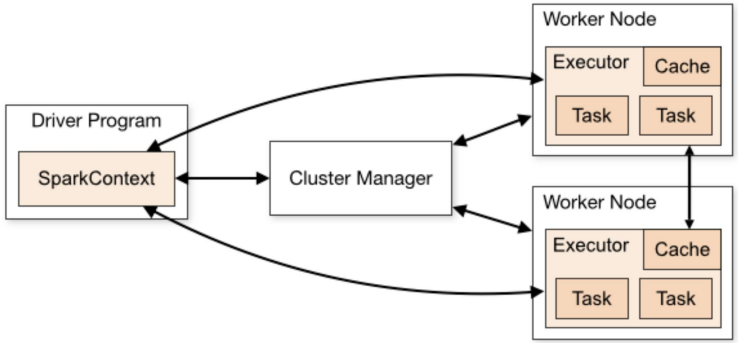
Why Use Apache Spark?

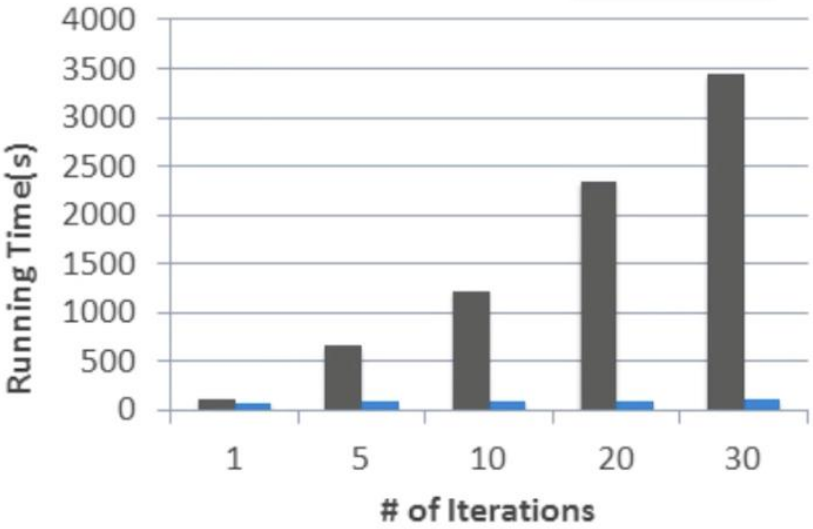
- Programmer friendly
- Simple workflow: Spark vs H20 (ML), SQL, Storm (Data Streaming)
- The paper likens Apache spark to having a smartphone, whereas other workflows are like having a phone, GPS, camera, map. More to keep track of and distract you from the task at hand.
- Apache spark makes life easier for the programmer, combining everything into a unified API, making applications easier to develop and reduce time spent configuring.
- Supports many different use cases: interactive, data streaming => widely used in scientific applications.
- Open Source

GREG : Speed Functionality: A Cluster Computing Tool

- Speed is very good preforming Machine Learning operations
  - **700 Queries Per Second!**
  - **Spark is 100 times faster than disk and Map-reduce!**
- In memory *processing* and *iterative computation*
  - Since it is in memory it can improve:
    - **Performance** on your daily tasks,
    - **Operations**
    - **Automation**
- In-memory storage
  - Hundreds of machines **working together**
    - with masters (Cluster Management)
    - slaves (Workers and Exeturors)

- Scalable, resilient and versatile
- Drawback - Speed is good but means **more data to cover**





Memory vs Drive Architecture

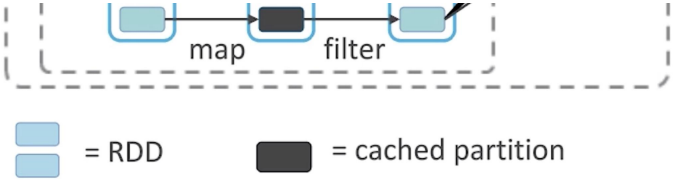
```
data = spark.textFile(...).map(readPoint).cache()

w = numpy.random.rand(D)

for i in range(iterations):
    gradient = data
     .map(lambda p: (1 / (1 + exp(-p.y * w.dot(p.x))))
          * p.y * p.x)
     .reduce(lambda x, y: x + y)
    w -= gradient

print "Final w: %s" % w
```

***** READS AND WRITES FROM RAM RATHER THAN DISK!!!**

- **Scheduling** OPTIMIZATION:
  - Fault tolerant
  - Mapping, joining, grouping, filters as well as
  - Resilient distributed dataset
- **MEMORY** Features
  - Memory, caching layer, stores in a distributed and fault tolerant way (nodes)
  - Falls back to disc when data does not fit in memory
- **Monte Carlo** Type Potential: Big Data
  - *DATA SETS ARE AND COME FROM ANYWHERE*
  - This speed is **good for:**
    - Big Data,
    - Data and Application Integration,
    - Cloud Data
    - Data Management

	Hadoop MapReduce	Spark
Data Size	102.5 TB	100 TB
Time Taken	72 min	23 min
No of nodes	2100	206
No of cores	50400 physical	6592 virtualized
Cluster disk throughput	3150 GBPS	618 GBPS
Network	Dedicated 10 Gbps	Virtualized 10 Gbps

RDD and What It Means?



- Cached RDDs + Cached Data Frames
    - Speeds up processes
    - WHY? :
    - Resilient : if data is **lost** it will be **recreated automatically**
    - Distributed : The Data is distributed **stored and processed**
    - Datasets Based:
        - Data can come from **different places** and data stores
- Features:
    - Parallel processing on datasets
    - **Immutable** and recomputable (machine learning and recovery)
    - Implicated and implemented on RDDs
- RDD, Traditional **SQL, NoSQL as a choice!**

YE: Explanation of Fault Tolerance

RDDs feature

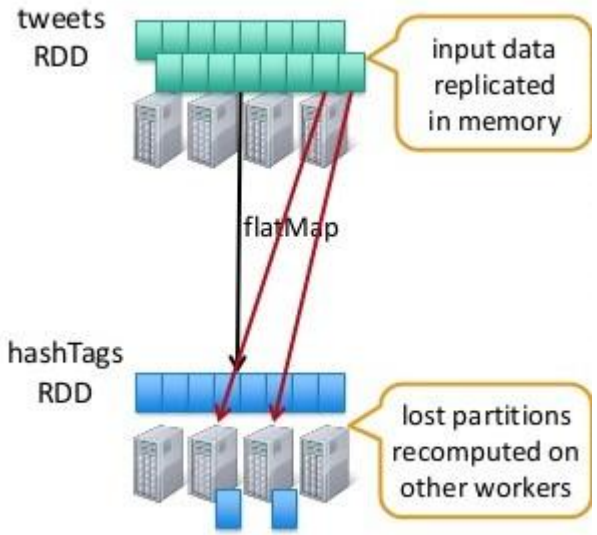
- Apache Spark RDD is an **immutable** dataset
- RDDs **track lineage** info to rebuild lost data

Recovery process

1. Retrieve lost data from **replicated memory**
2. **Recompute** replicated input data

Ideal result

The data in the **final** transformed RDD will always be the **same**



OPERATIONS: TRANSFORMATIONS + ACTIONS

- OPERATIONS
  - If you create an RDD and update it, it will create a new RDD
    - ***Map, filters, and so on***
    - Constant state of recreation of data

- ACTIONS
  - When trying to get a result or writing to storage
  - ***Count, collect, save***
    - Will look inside of the RDD and output a result

EXtra: Spark Speed Facts

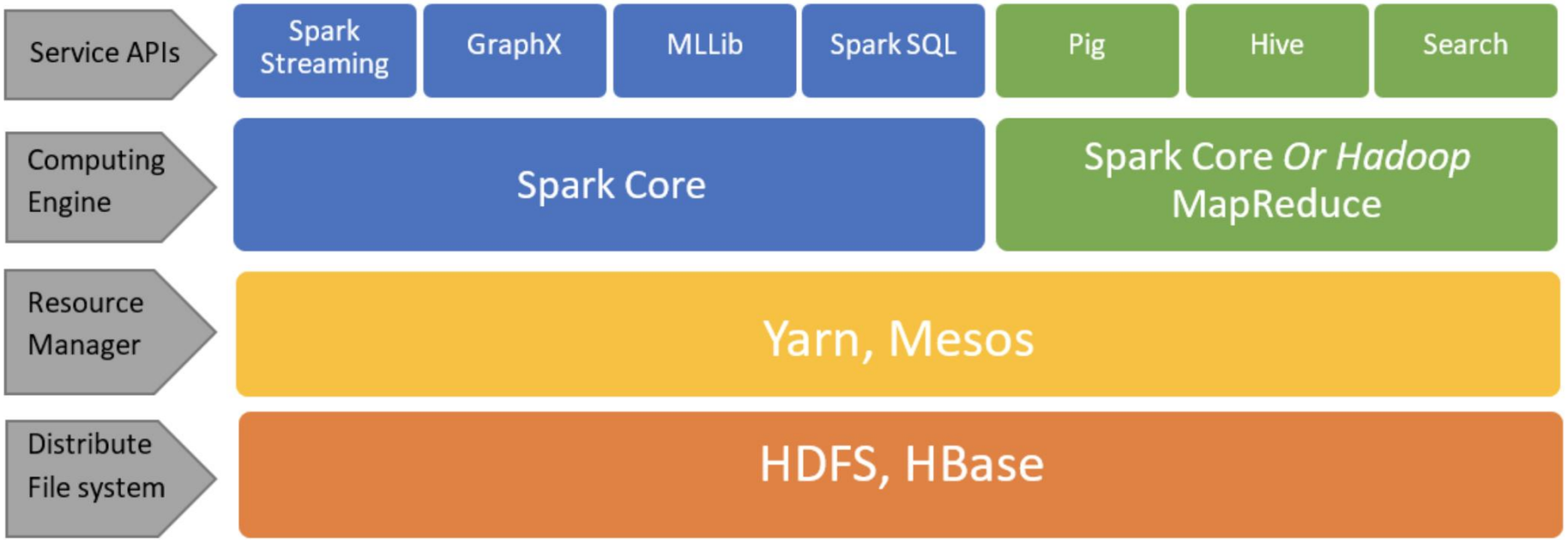
- Spark has the ability to *conjoin* with Hive to make your queries very fast
  - Providing an engine for **Hive Data** to move 1**00x faster than traditional databases**
  - **Monitoring** speed capabilities (**Dr Elephant**)
  - **Memory tuning** - **Scaling**

Speed reduces costs $

=

Issues!

Increase spark applications + **Increasing** Spark Users



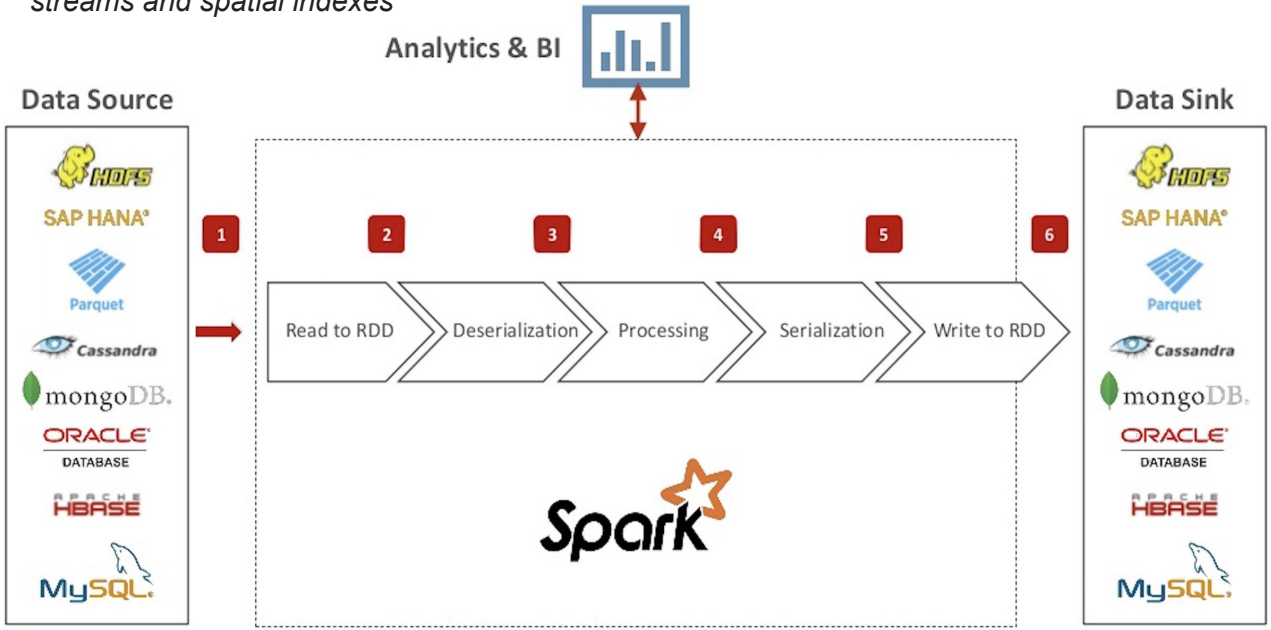
< Architecture >

Redis:

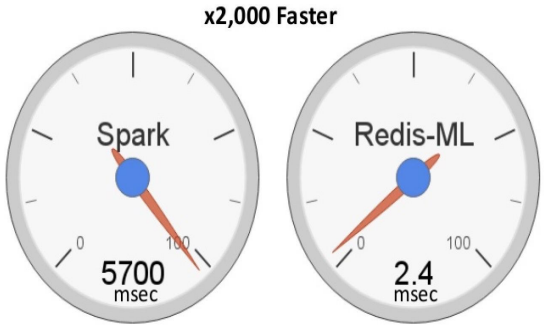
- open-source in-memory data structure
- project implementing a distributed
- in-memory key-value database
  - **optional durability**

=== > EXTRA FUNCTIONALITY

Capabilities: *strings, lists, maps, sets, sorted sets, hyperloglogs, bitmaps, streams and spatial indexes*



Redis X		Spark SQL X
Supported programming languages	C C# C++ Clojure Crystal D Dart Elixir Erlang Fancy Go Haskell Haxe Java JavaScript (Node.js) Lisp Lua MatLab Objective-C OCaml Pascal Perl PHP Prolog Pure Data Python R Rebol Ruby Rust Scala Scheme Smalltalk Swift Tcl Visual Basic	Java Python R Scala
In-memory capabilities	yes	no



Drawbacks