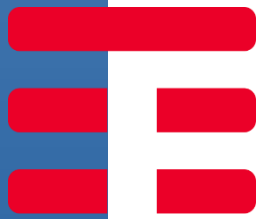

GRUPPO TELECOM ITALIA
Big Data Transformation
Big Data Transformation
TIM Academy



GRUPPO TIM

Big Data Transformation

Apache Spark Fundamentals & Query Fundamentals

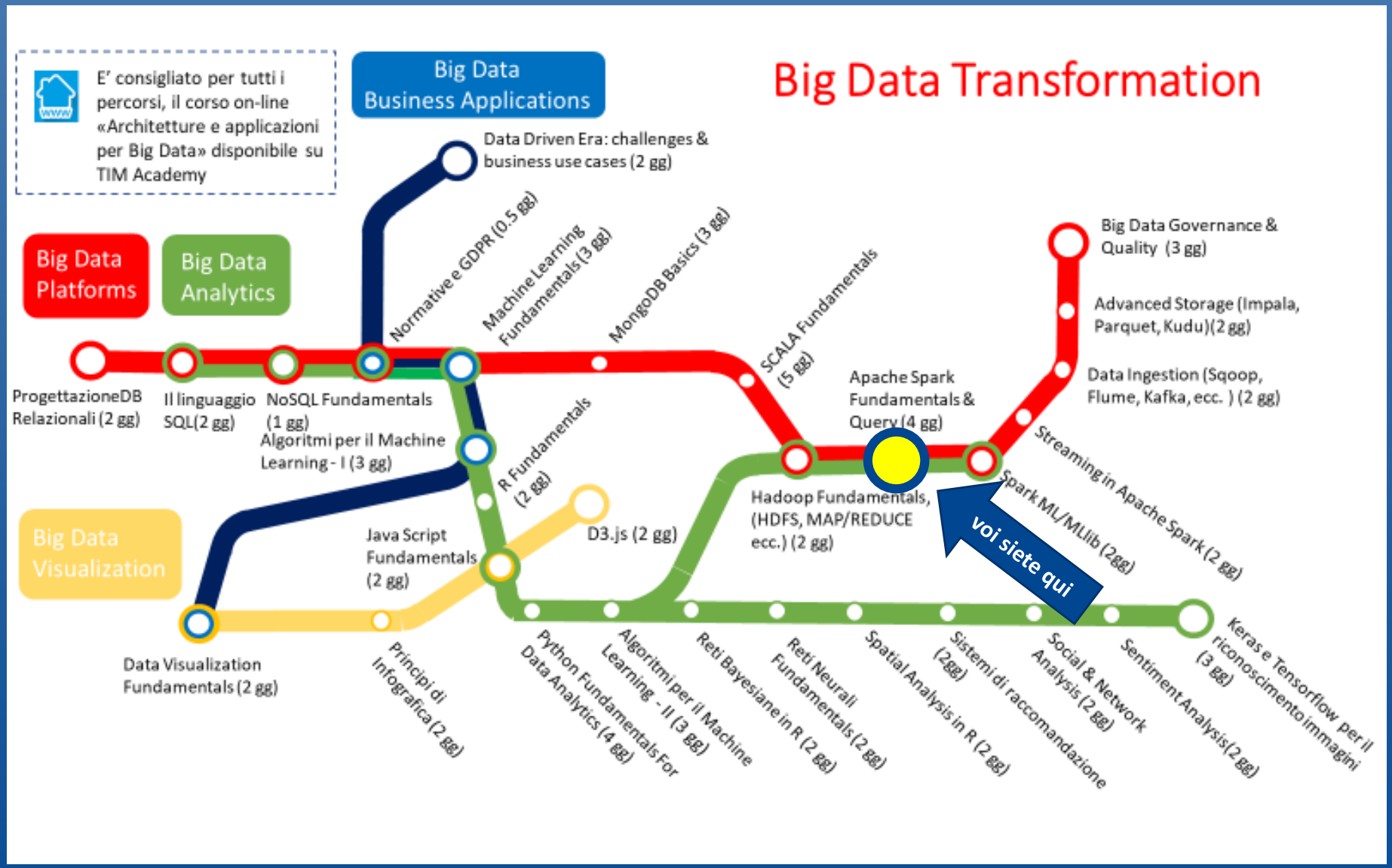
TIM Academy

GRUPPO TIM

Big Data Transformation

Apache Spark Fundamentals & Query

TIM Academy


E' consigliato per tutti i percorsi, il corso on-line «Architetture e applicazioni per Big Data» disponibile su TIM Academy
Big Data Business Applications
Big Data Transformation
Big Data Platforms
Big Data Analytics
Big Data Visualization
Data Driven Era: challenges & business use cases (2 gg)
Normative e GDPR (0.5 gg)
Machine Learning Fundamentals (3 gg)
MongoDB Basics (3 gg)
SCALA Fundamentals (5 gg)
Big Data Governance & Quality (3 gg)
Advanced Storage (Impala, Parquet, Kudu)(2 gg)
Data Ingestion (Sqoop, Flume, Kafka, ecc.) (2 gg)
Apache Spark Fundamentals & Query (4 gg)
Streaming in Apache Spark (2 gg)
Spark ML/MLlib (2gg)
voi siete qui
ProgettazioneDB Relazionali (2 gg)
Il linguaggio SQL(2 gg)
NoSQL Fundamentals (1 gg)
Algoritmi per il Machine Learning - I (3 gg)
R Fundamentals (2 gg)
Hadoop Fundamentals, (HDFS, MAP/REDUCE ecc.) (2 gg)
Java Script Fundamentals (2 gg)
D3.js (2 gg)
Data Visualization Fundamentals (2 gg)
Principi di Infografica (2 gg)
Python Fundamentals For Data Analytics (4 gg)
Algoritmi per il Machine Learning - II (3 gg)
Reti Bayesiane in R (2 gg)
Reti Neurali Fundamentals (2 gg)
Spatial Analysis in R (2 gg)
Sistemi di raccomandazione (2gg)
Social & Network Analysis (2 gg)
Sentiment Analysis(2 gg)
Keras e Tensorflow per il riconoscimento immagini (3 gg)

TIM Academy

Apache Spark

Uno dei tool più usati nel campo dei ***BIG DATA***

https://www.whizlabs.com/blog/big-data-tools/

1° APACHE hadoop

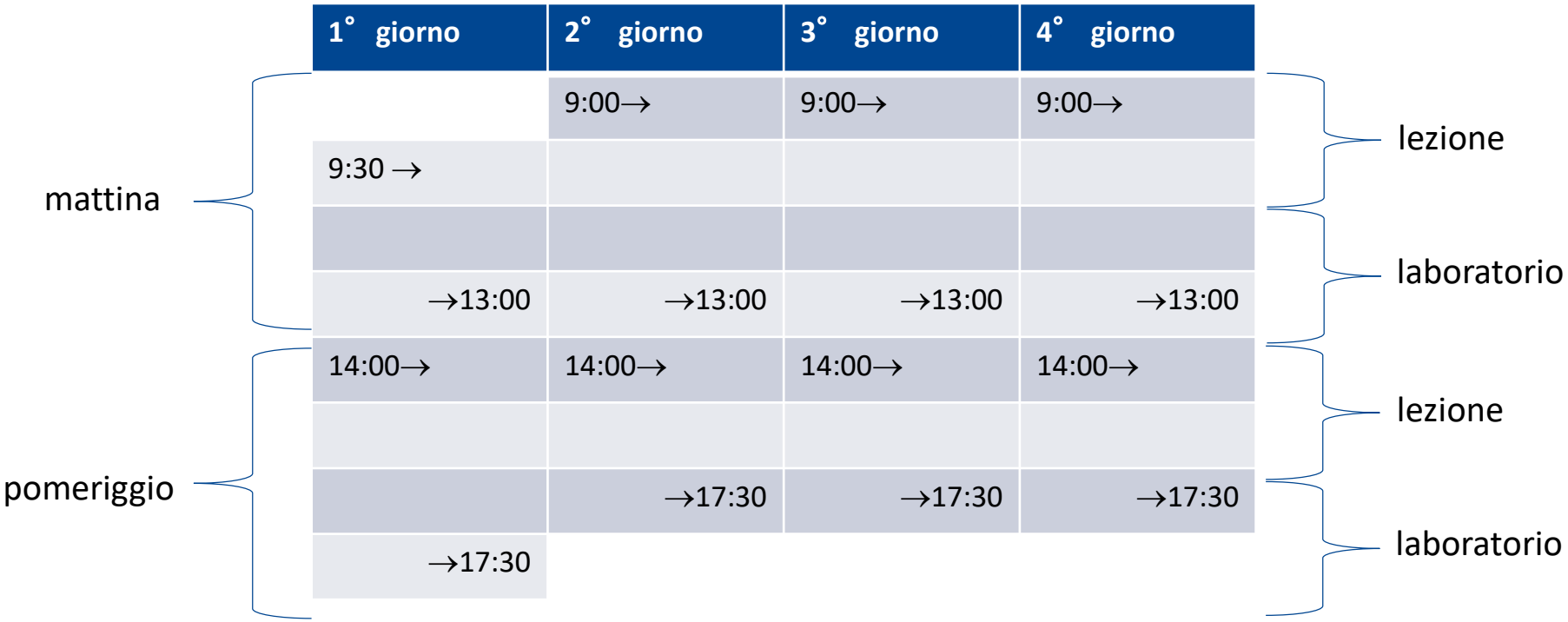
2° APACHE Spark™

Prerequisiti

- Conoscenza del linguaggio di programmazione Scala

- Aver seguito il corso Hadoop Fundamentals

Agenda del corso



Agenda 1/8

Introduzione

Presentazione del corso

Spark
Cos'è? A cosa serve? Chi lo usa? Punti di forza e debolezze

Contesto e storia
Come è nato? Chi l'ha fatto? Quale problema affronta?

Com'è fatto?
Una prima presentazione dell'architettura

E in pratica?
Come si usa? Come lo useremo noi in laboratorio?

Agenda 2/8

Le fondamenta

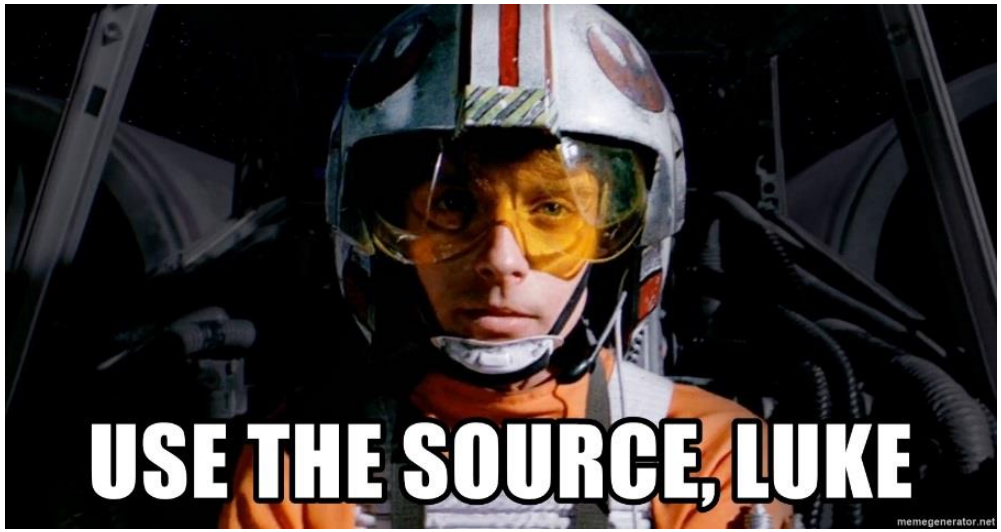
La programmazione funzionale
Un paradigma utile per il calcolo parallelo

Strutture dati
RDDs, Datasets e DataFrames

Operazioni sui dati
Transformazioni e Azioni; Valutazioni «pigre»; persistenza (cache)

Punti di riferimento
Documentazione; Cloudera, Hortonworks; Databricks;
Codice sorgente su github

Esercizi di laboratorio
Mappe, filtri e qualche azione



Agenda 3/8

- **API Strutturata**
  - DataFrames e Dataset in profondità
  - Data Source
  - Date, Stringhe ed Espressioi regolari
  - Union, Join e SQL

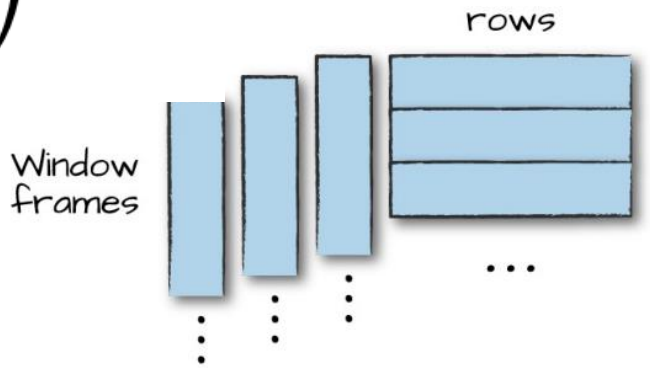
```
root
 |-- InvoiceNo: string (nullable = true)
 |-- StockCode: string (nullable = true)
 |-- Description: string (nullable = true)
 |-- Quantity: integer (nullable = true)
 |-- InvoiceDate: timestamp (nullable = true)
 |-- UnitPrice: double (nullable = true)
 |-- CustomerID: double (nullable = true)
 |-- Country: string (nullable = true)
```

```
+---------+---------+--------------------+--------+-------------------+----...
|InvoiceNo|StockCode|         Description|Quantity|        InvoiceDate|Unit...
+---------+---------+--------------------+--------+-------------------+----...
|   536365|   85123A|WHITE HANGING HEA...|       6|2010-12-01 08:26:00|    ...
|   536365|    71053| WHITE METAL LANTERN|       6|2010-12-01 08:26:00|    ...
...
|   536367|    21755|LOVE BUILDING BLO...|       3|2010-12-01 08:34:00|    ...
|   536367|    21777|RECIPE BOX WITH M...|       4|2010-12-01 08:34:00|    ...
+---------+---------+--------------------+--------+-------------------+----...
```

Agenda 4/8

- **Aggregazioni**
  - Tipi diversi di aggregazione
  - Window
  - Aggregazioni su tipi complessi

$$\left(\sum_{i=1}^{n} p_i x_i^2\right) - \mu^2$$



Agenda 5/8

- **RDD**

  La prima API e quella fondamentale

  Tipi diversi di RDD. RDD di coppie chiave-valore

  Partizioni

  Variabili condivise: accumulatori e variabili broadcast

RDD

Agenda 6/8

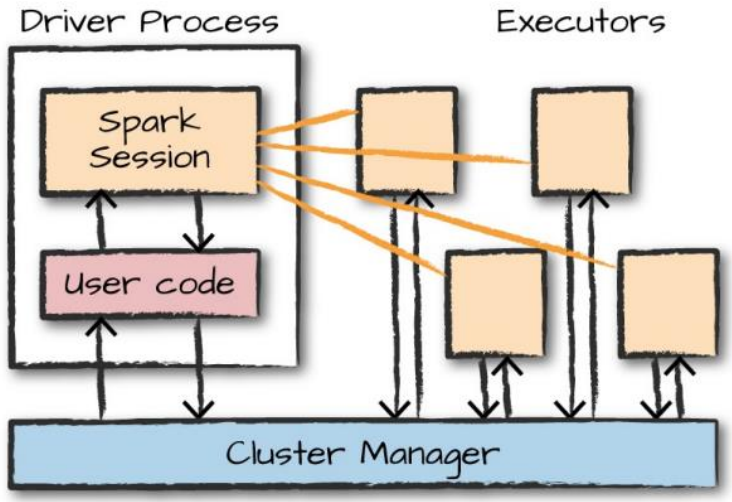
- **Il cluster**

  Struttura di un'applicazione Spark

  Modi di esecuzione

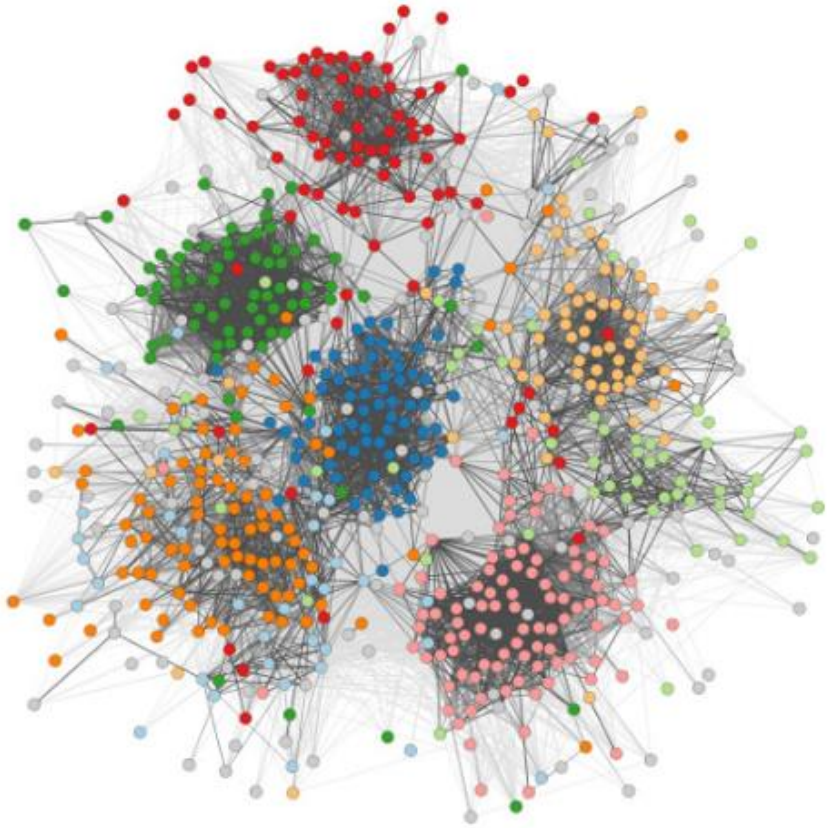
  Ciclo di vita di un'applicazione Spark

  Sessione, programma e configurazione



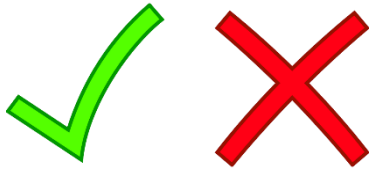
Agenda 7/8

- **GraphFrames**
  - Packages
  - Spark streaming, Spark ML
  - GraphX e GraphFrames
  - Cos'è un grafo? Vertici e Archi
  - Algoritmi principali



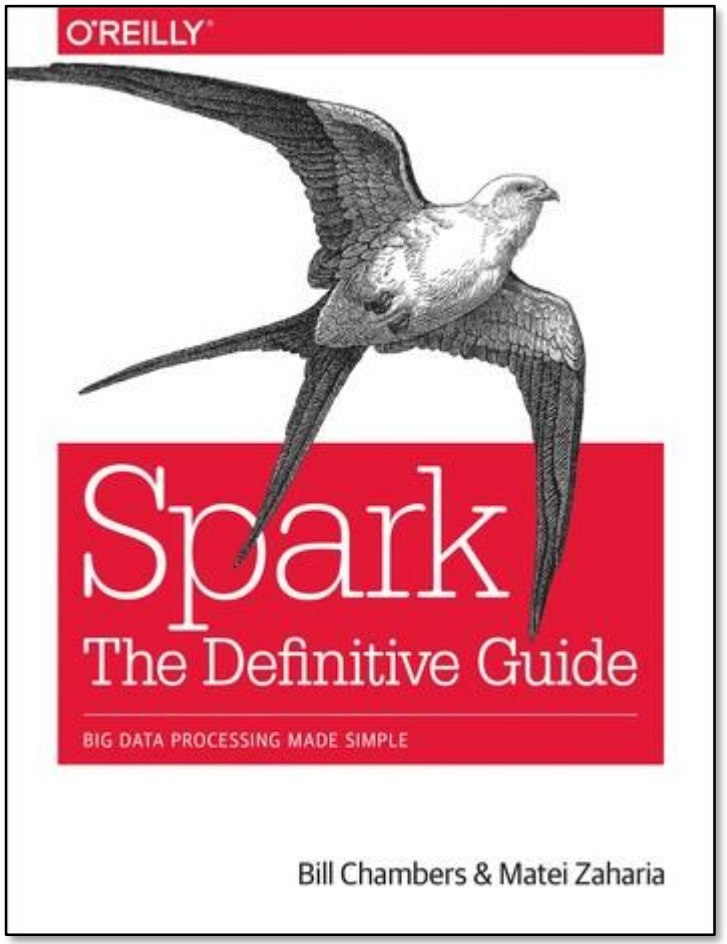
Agenda 8/8

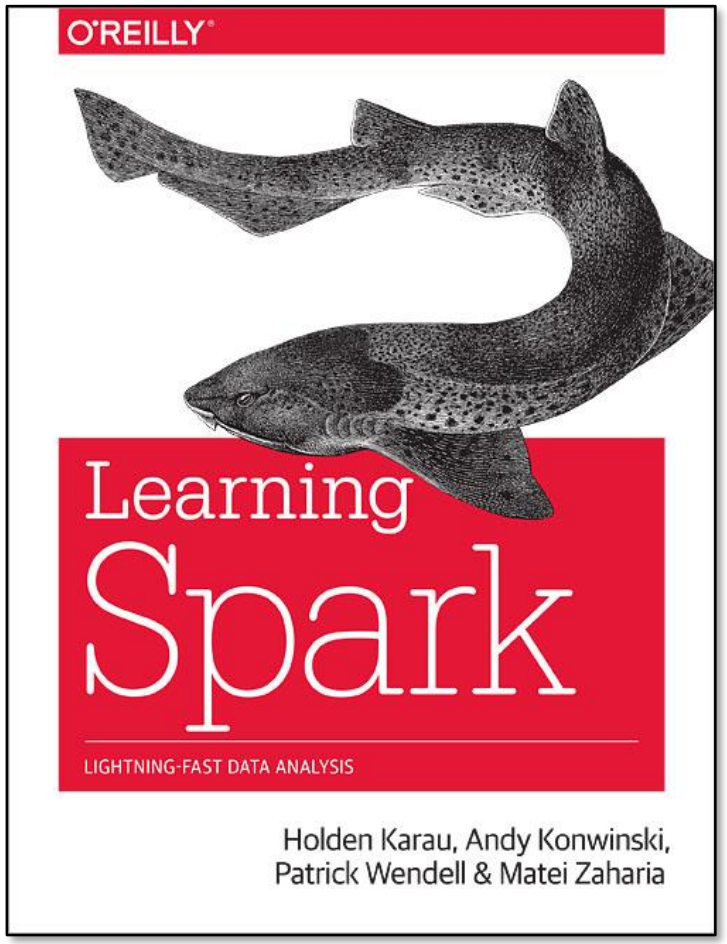
- **Conclusioni**
  - «Do» e «don't» in Spark
  - Recapitolazione e domande
  - **Compilazione del questionario**



Primo giorno: mattina

Riferimenti bibliografici





Cos'è

Apache Spark?

Cos'è Apache Spark?

"Apache Spark is an **open-source distributed general-purpose cluster-computing framework**".

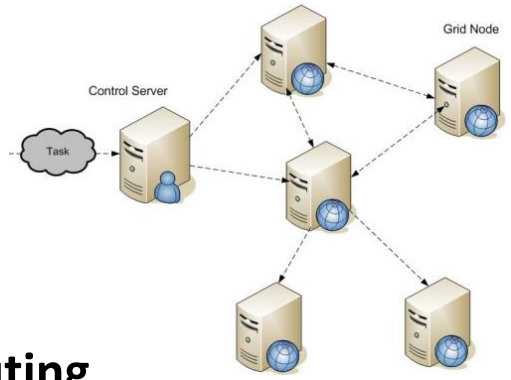
WIKIPEDIA
The Free Encyclopedia

- **Open-source**

Open Source Initiative

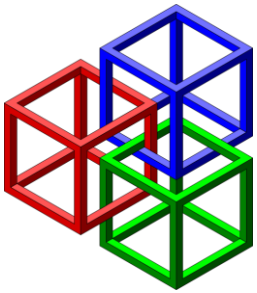
- **General-purpose**

- **Distributed**



- **Cluster-computing**

- **Framework**



Computing… ad esempio?

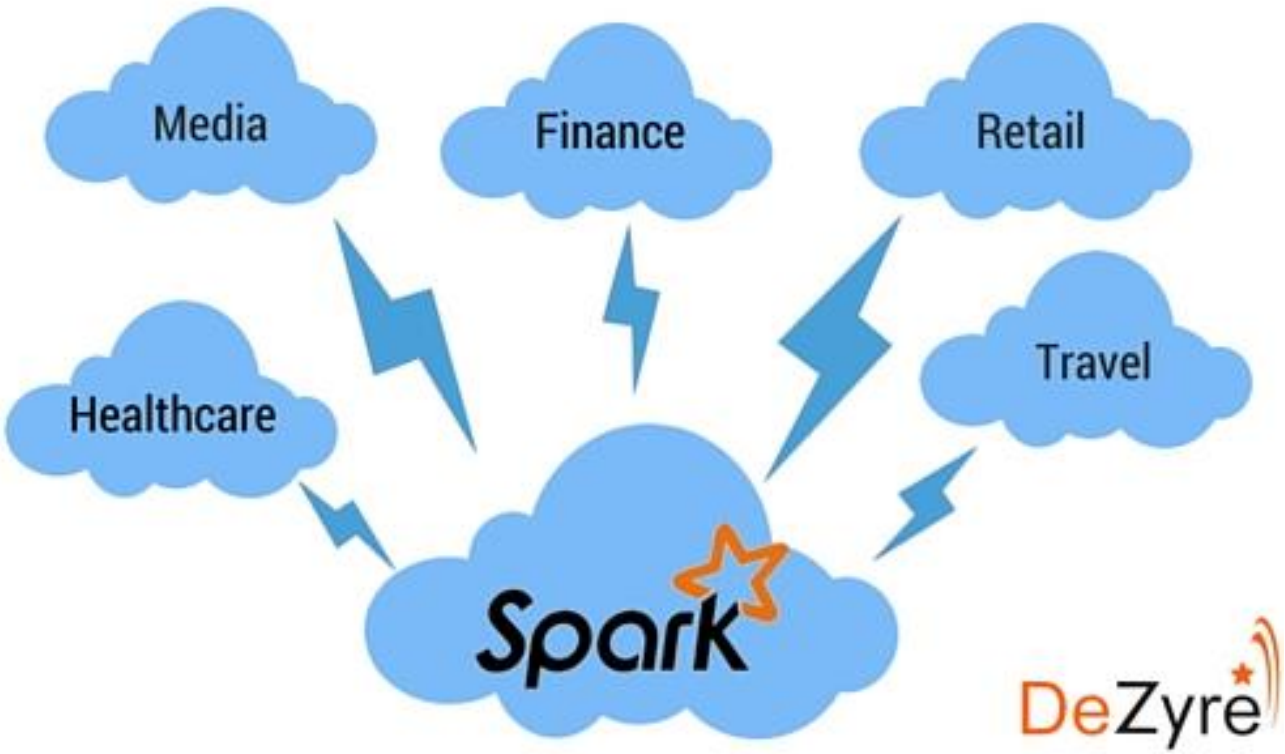
Vari utilizzi di Apache Spark

- Streaming
  - Streaming ETL (Extract, Transform, Load)
  - Data enrichment
  - Event detection (ad es. per individuare transizioni fraudolente, cambi potenzialmente pericolosi dei parametri vitali, ecc.)
  - Complex session analysis
- Machine learning
  - Marketing purposes and sentiment analysis
  - Network security
- Interactive analysis

https://www.qubole.com/blog/apache-spark-use-cases/

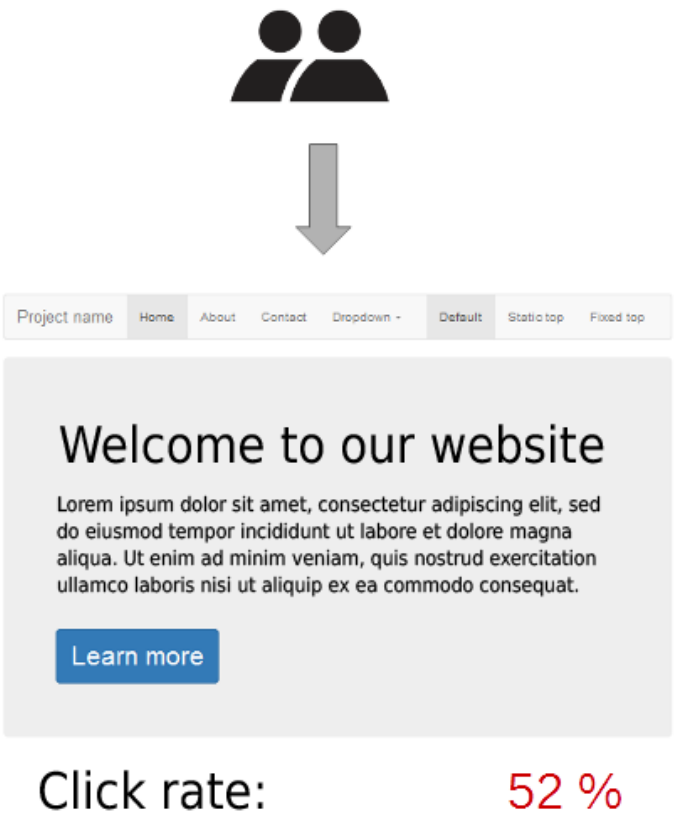
Computing… ad esempio?

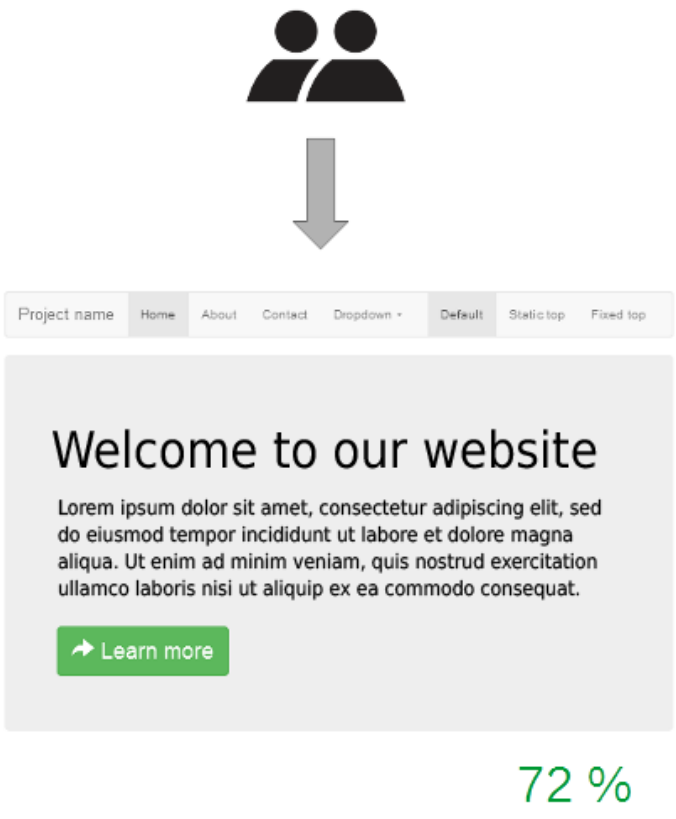
Vari utilizzi di Apache Spark



https://www.dezyre.com/article/top-5-apache-spark-use-cases/271

Test A/B


Project name
Home
About
Contact
Dropdown
Default
Static top
Fixed top
Welcome to our website
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Learn more
Click rate:
52 %


Project name
Home
About
Contact
Dropdown
Default
Static top
Fixed top
Welcome to our website
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
Learn more
72 %

Qualche esempio concreto

Alibaba.com™

ebay

NASA Jet Propulsion Laboratory
California Institute of Technology

YAHOO!

Rif: https://spark.apache.org/powered-by.html

Qualche esempio concreto

...

Fonte: https://databricks.com/blog/2018/09/05/a-guide-apache-spark-use-cases-streaming-and-research-talks-at-spark-ai-summit-europe.html
https://www.infoworld.com/article/3031690/analytics/why-you-should-use-spark-for-machine-learning.amp.html

Qualche esempio concreto

Massively Parallel Processing.
1 PetaByte = 1.000.000 GB

ebay

"[…] eBay is migrating its 30 PB MPP database to Apache Spark […]" [2018]

https://databricks.com/session/experience-of-optimizing-spark-sql-when-migrating-from-mpp-database

GE Aviation

monitor or detect or predict preventive maintenance [2018]

"[…] Analytic tools such as SQL Server and MATLAB were used until recently, when GE's data was moved to an Apache Spark environment. Consequently, our advanced analytics are now being migrated to Spark, […]"

https://databricks.com/session/ge-aviation-spark-application-experience-porting-analytics-into-pyspark-ml-pipelines

ING

[2015?]

"In the system presented, we run Spark to run the data analytics pipeline for anomaly detection. […] Our machine learning pipeline uses Spark decision tree ensembles and k-means clustering."

https://spark-summit.org/eu-2015/events/real-time-anomaly-detection-with-spark-ml-and-akka/

Fonte: https://databricks.com/blog/2018/09/05/a-guide-apache-spark-use-cases-streaming-and-research-talks-at-spark-ai-summit-europe.html
https://www.infoworld.com/article/3031690/analytics/why-you-should-use-spark-for-machine-learning.amp.html

Qualche esempio concreto

Uber

Dal 2015 usa *Hadoop* + *Kafka* + ***Spark*** *(ecc.)*
100+ Petabytes con un minuto di latenza
10 000 job **Spark** / giorno

Cfr: https://eng.uber.com/uber-big-data-platform/

Pinterest

175 milioni di utenti mensilmente attivi
Test A/B
Kafka + Pinball + ***Spark*** *Streaming + HBase &MemSQL + Presto*

Cfr: https://medium.com/@Pinterest_Engineering/building-a-new-experiment-pipeline-with-spark-b04e1a7d639a

conviva®

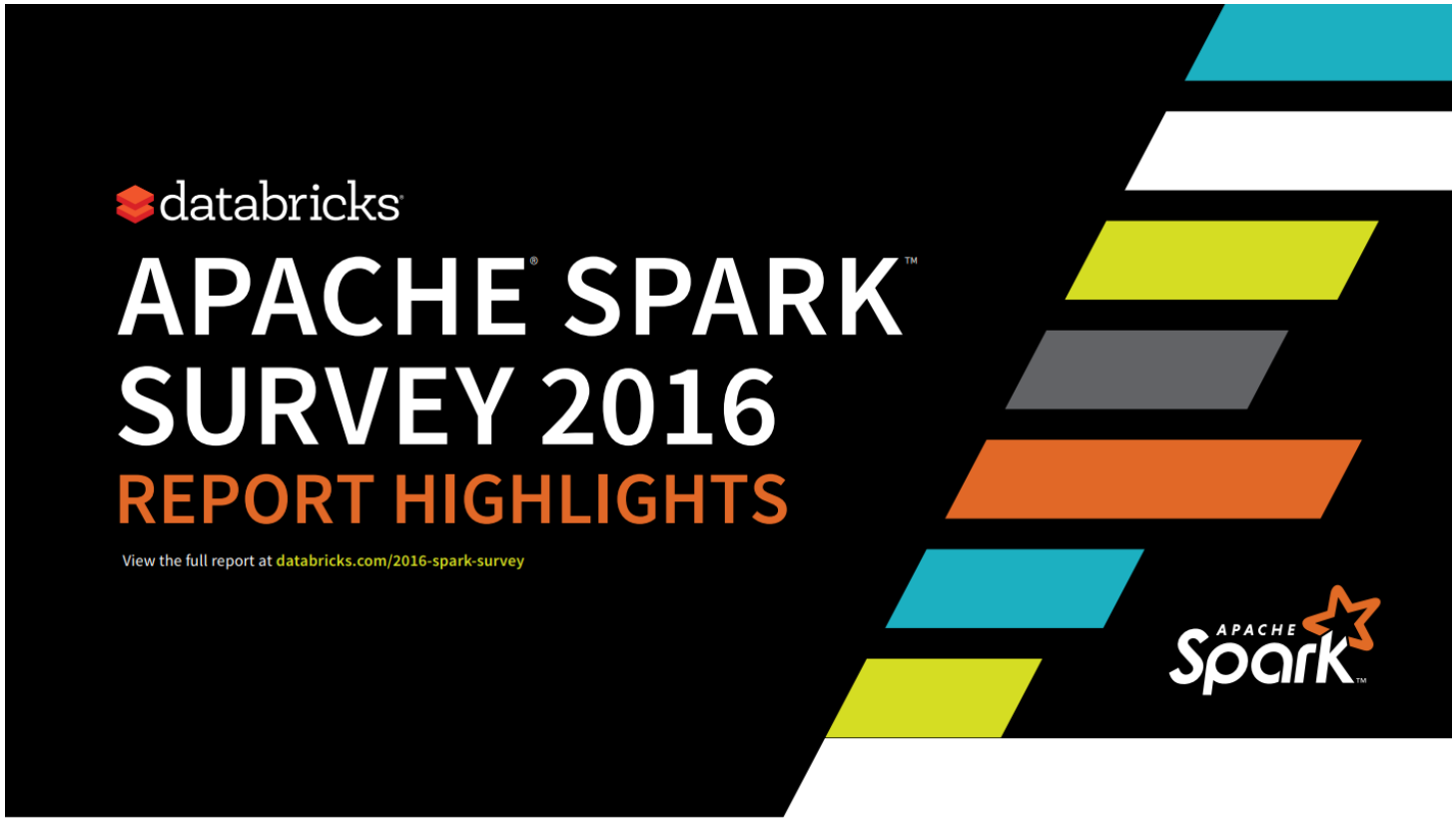
65 milioni di video stream / giorno

http://ampcamp.berkeley.edu/wp-content/uploads/2012/06/dilip-joseph-amp-camp-2012-conviva-using-spark.pdf

https://www.qubole.com/blog/apache-spark-use-cases/

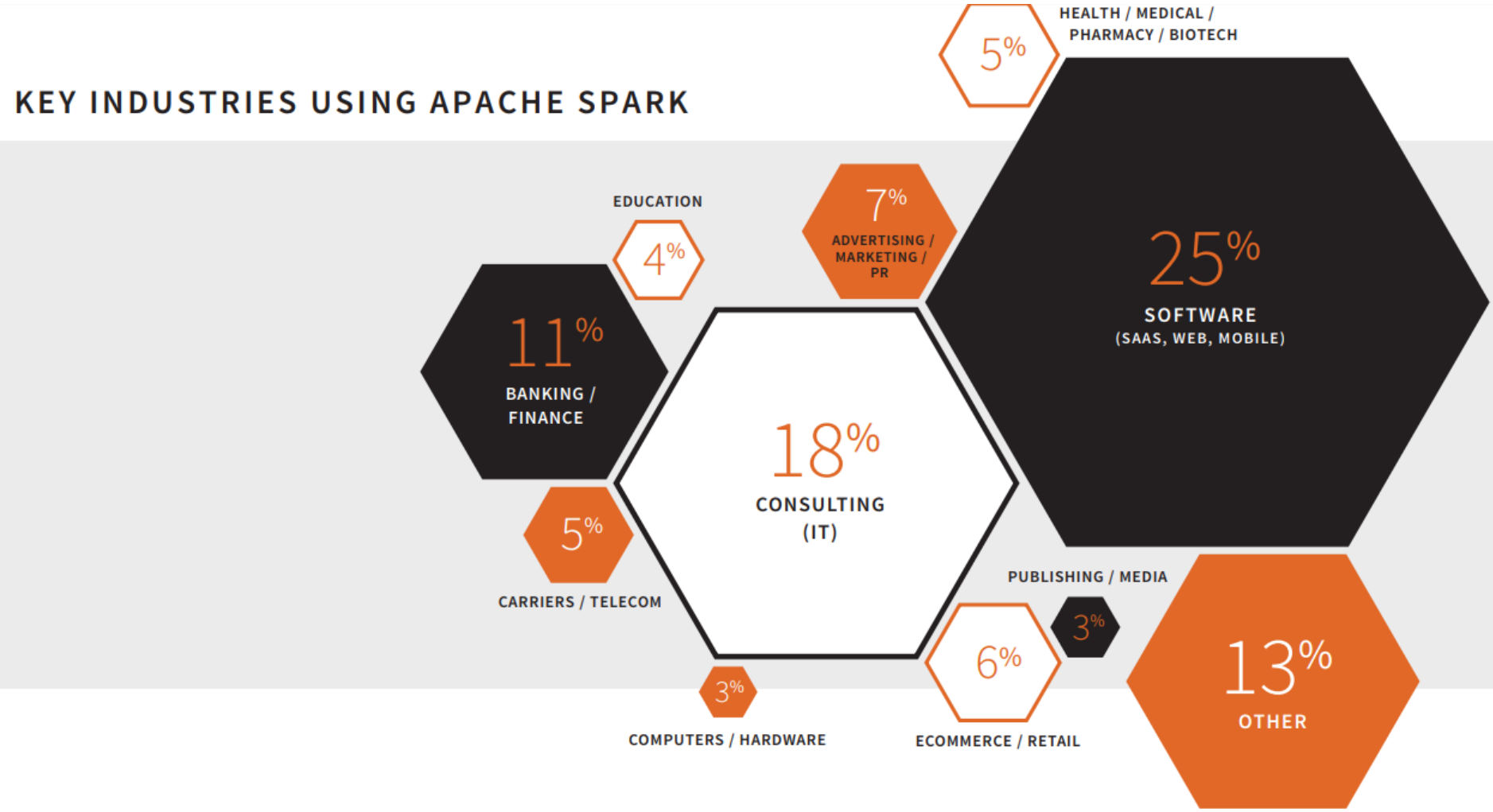
Chi usa Spark e perché

Nel 2016 i creatori di Spark hanno organizzato un sondaggio fra gli utenti.
(1615 risposte da parte di più di 900 organizzazioni)

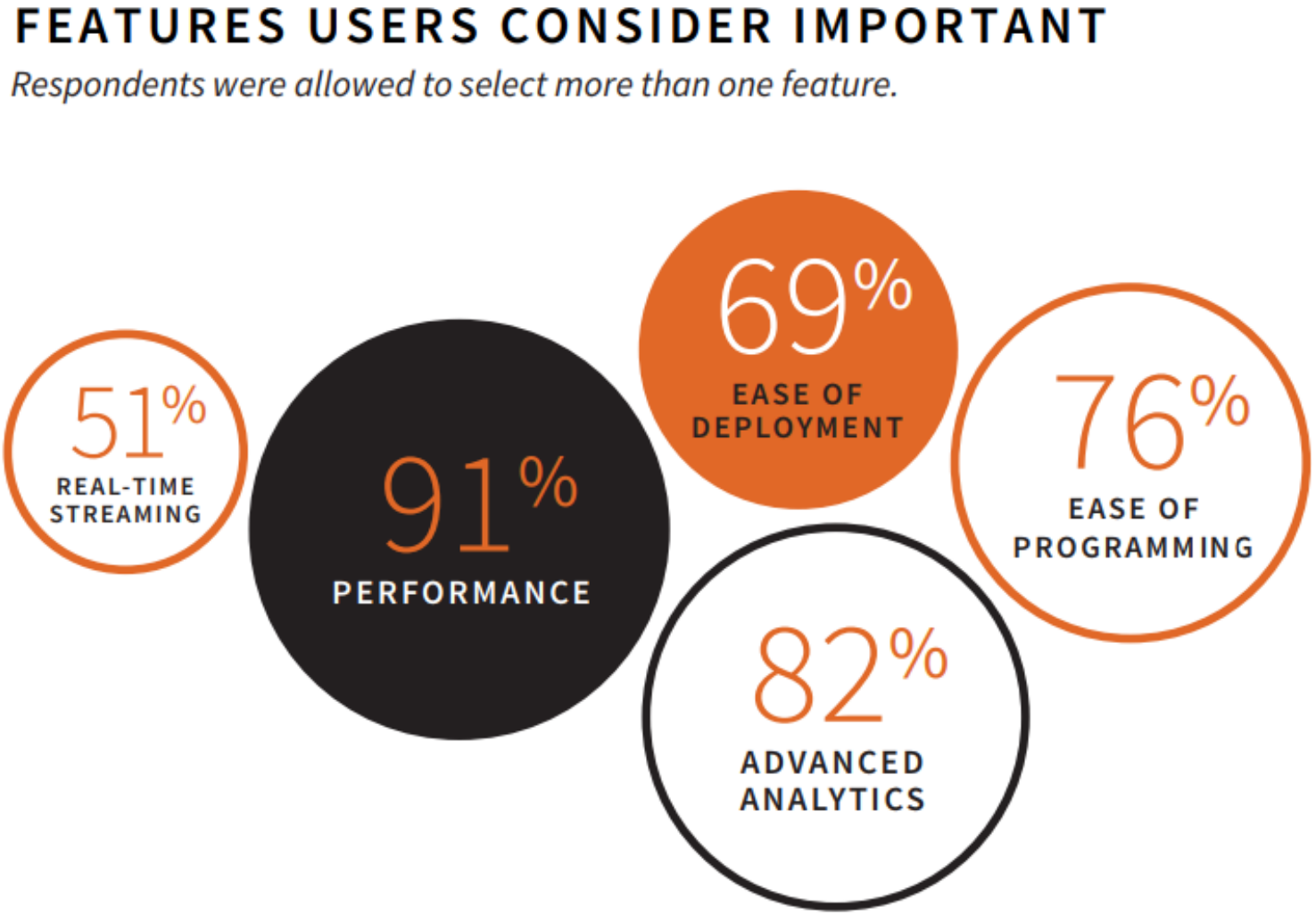


http://pages.databricks.com/rs/094-YMS-629/images/2016_Spark_Infographic.pdf

Chi usa Spark e perché



Chi usa Spark e perché



Perché Spark?

Principali punti di forza

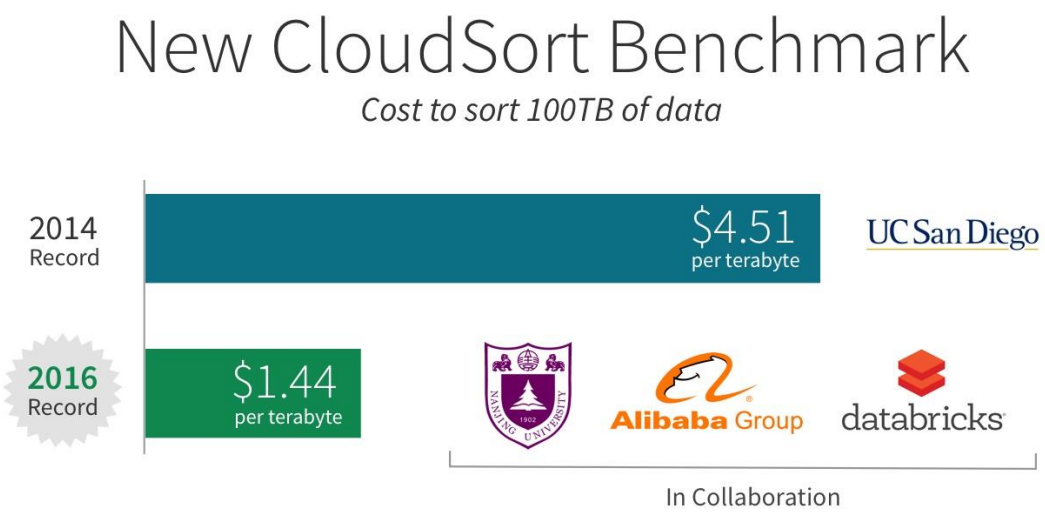
- Veloce
- Agnostico rispetto alla piattaforma
- Versatile
- Facile
- Capace di scalare

- Le «3 S» di Spark:
  - **Speed** = Velocità
  - **Semplicity** = Facilità di uso
  - **Support** = Vari linguaggi e sistemi di storage;
    Provider commerciali + comunità grande, attiva e internazionale

https://bigdata-madesimple.com/what-is-3-ss-of-spark-and-its-effect-on-big-data/

Punti di forza: Velocità

- Ingegnerizzato dal basso per ottenere le migliori performance
- 100x più veloce di *Hadoop MapReduce* grazie all'elaborazione in memoria (e altre ottimizzazioni)
- Veloce anche su disco.
- Record del mondo nel 2014 e 2016 per ordinamento su disco di grandi basi dati



https://databricks.com/blog/2016/11/14/setting-new-world-record-apache-spark.html

Punti di forza: Facilità d'uso

- Approccio dichiarativo di alto livello ed API per agire su grandi datasets
- >100 operatori per trasformare data
- API basate su *DataFrame* e *Dataset* che permettono di manipolare facilmente dati strutturati e semi-strutturati

```
val errorCount = spark.read
        .text(fn)
        .filter($"value".like("%ERROR%"))
        .count()
```

Punti di forza: Facilità d'uso

WordCount in *MapReduce (Java)*

```java
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

   public static class Map extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {
     private final static IntWritable one = new IntWritable(1);
     private Text word = new Text();

     public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
       String line = value.toString();
       StringTokenizer tokenizer = new StringTokenizer(line);
       while (tokenizer.hasMoreTokens()) {
         word.set(tokenizer.nextToken());
         output.collect(word, one);
       }
     }
   }
   public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable, Text, IntWritable> {
     public void reduce(Text key, Iterator values, OutputCollector<Text, IntWritable> output, Reporter reporter) throws IOException {
       int sum = 0;
       while (values.hasNext()) {
         sum += values.next().get();
       }
       output.collect(key, new IntWritable(sum));
     }
   }

   public static void main(String[] args) throws Exception {
     JobConf conf = new JobConf(WordCount.class);
     conf.setJobName("wordcount");
     conf.setOutputKeyClass(Text.class);
     conf.setOutputValueClass(IntWritable.class);
     conf.setMapperClass(Map.class);
     conf.setCombinerClass(Reduce.class);
     conf.setReducerClass(Reduce.class);
     conf.setInputFormat(TextInputFormat.class);
     conf.setOutputFormat(TextOutputFormat.class);
     FileInputFormat.setInputPaths(conf, new Path(args[0]));
     FileOutputFormat.setOutputPath(conf, new Path(args[1]));
     JobClient.runJob(conf);
   }
}
```

WordCount in *Spark (Scala)*

```scala
val file = spark.textFile("hdfs://...")
val counts = file
    .flatMap(line => line.split(" "))
    .map(word => (word, 1)).reduceByKey(_ + _)
counts.saveAsTextFile("hdfs://...")
```

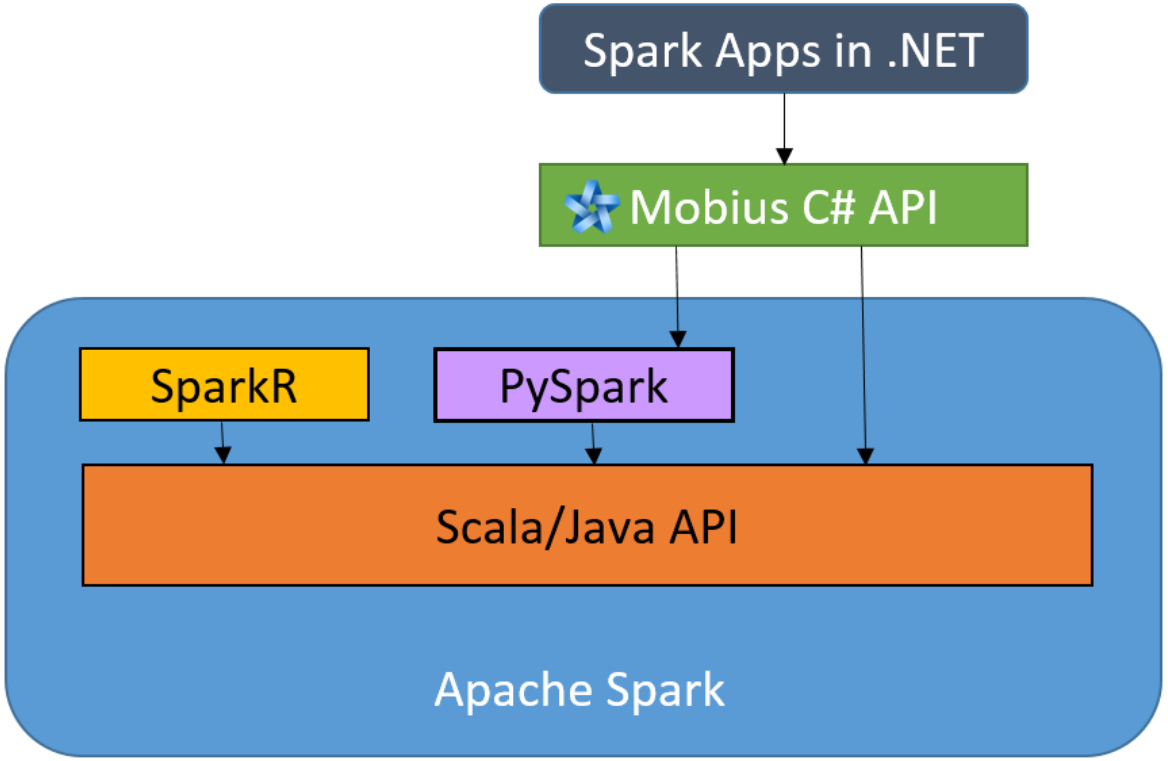
Punti di forza: Facilità d'uso

- Può essere eseguito interattivamente dalla shell o utilizzando un *NoteBook*
- Supporto per diversi linguaggi: **Scala**, **Java**, **Python**, **R**, **SQL**

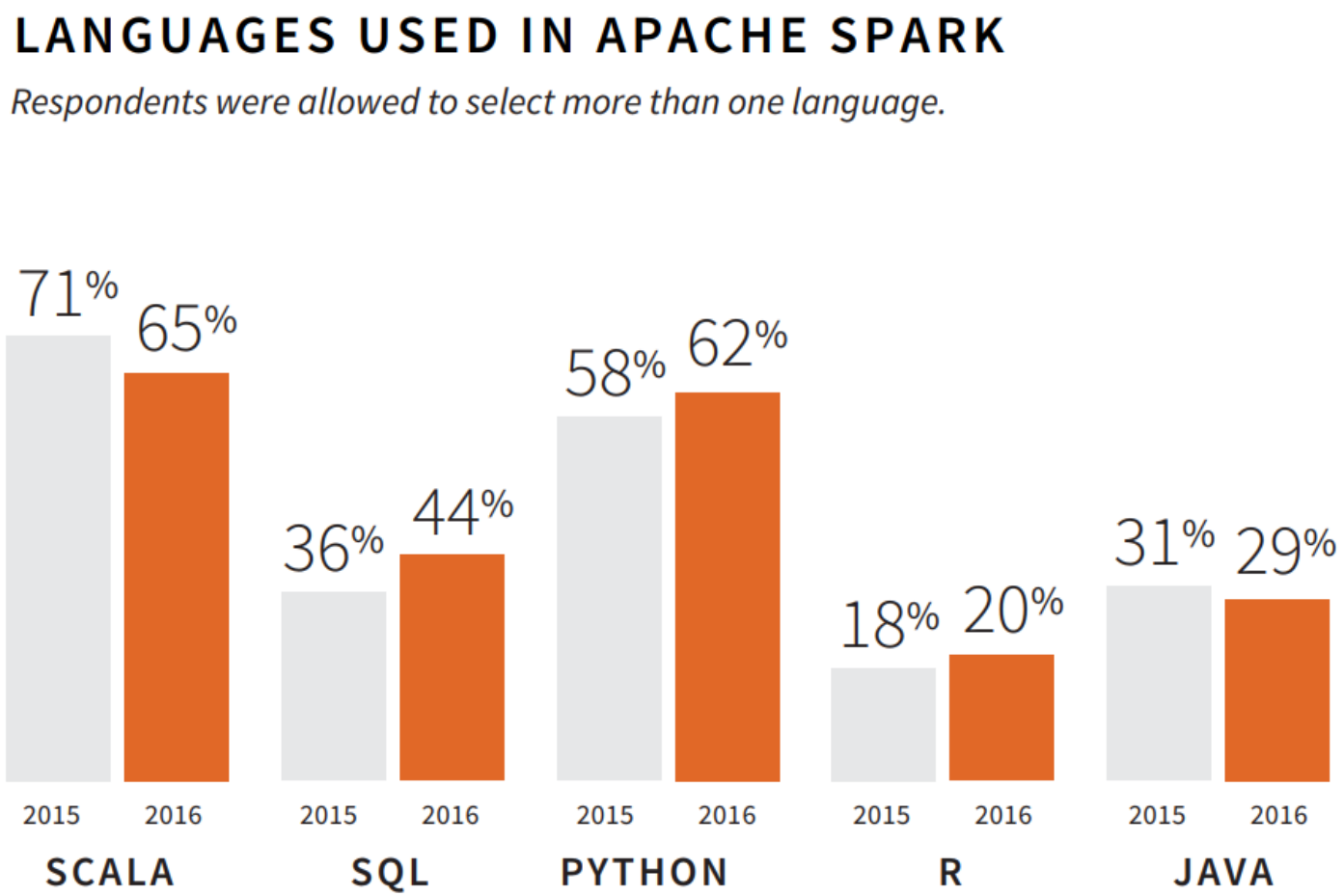
E stanno arrivando altri linguaggi!

(progetti di terze parti)

- *Clojure*
- *C#/.net* (progetto *Möbius*)
- *Groovy*
- *Julia*



Quale linguaggio usare?



http://pages.databricks.com/rs/094-YMS-629/images/2016_Spark_Infographic.pdf

Quale linguaggio usare?

Java

PRO:

- Potente e completo
- Molto diffuso, conosciuto e utilizzato: probabilmente il linguaggio più usato per i Big Data
- Alte prestazioni

CONTRO:

- Verboso
- Non supporta *Read-Evaluate-Print-Loop*

Quale linguaggio usare?

R

PRO:

- Il linguaggio di riferimento per Statistici e Analisti di Dati
- Buone librerie di visualizzazione

CONTRO:

- Non è considerato un linguaggio di utilizzo generale
- Prestazioni non alte
- C'è un certo ritardo nell'integrazione con Spark

Quale linguaggio usare?

SQL

PRO:

- Utilizzabile anche da non programmatori
- Trend di utilizzo in crescita

CONTRO:

- Inadatto per alcuni tipi di applicazioni

Quale linguaggio usare?

Python

PRO:

- Linguaggio molto noto e diffuso, anche nel mondo dei Big Data
- Facile da imparare
- Molto conciso ed espressivo
- Buone librerie di visualizzazione

CONTRO:

- Più lento
- «*Duck typing*». Controllo sui tipi a run-time

Quale linguaggio usare?

Scala

PRO:

- Spark è scritto in Scala: ogni feature del framework è immediatamente disponibile
- Compatibile con le librerie Java
- Controllo sui tipi durante la compilazione
- Alte prestazioni

CONTRO:

- Meno diffuso di Java o Python
- Marginalmente meno facile di Python

In questo corso useremo:


```
object HelloWorld {
        def main(args: Array[String]): Unit = {
                println("Hello, world!")
        }
}
```

Spark ha qualche punto debole

- Non ha un suo sistema di storage
- Grande consumo di risorse (in particolare la memoria) $\Rightarrow$ Spark è dispendioso
- Può gestire anche dataset molto grandi, ma richiede un po' di attenzione.
- Non ha supporto per il *real-time processing* (*Spark Streaming* usa micro-batches)
- Tende a generare tanti file relativamente piccoli. Questo è un male, ad es. per *Hadoop HDFS* o *Amazon S3*.
- Può richiedere ottimizzazione manuale
- Ha una latenza considerevole

Non solo Apache Spark!

Alcune alternative:

(creato da Twitter)

Apache Flink

TIBCO®

StreamBase

IBM InfoSphere Streams

Il contesto
(e un po’ di storia)

Contesto

La potenza di calcolo ha continuato a crescere ininterrottamente per ottant'anni

Univac I (1951)
~ 1 000 000$
1905 operazioni al secondo

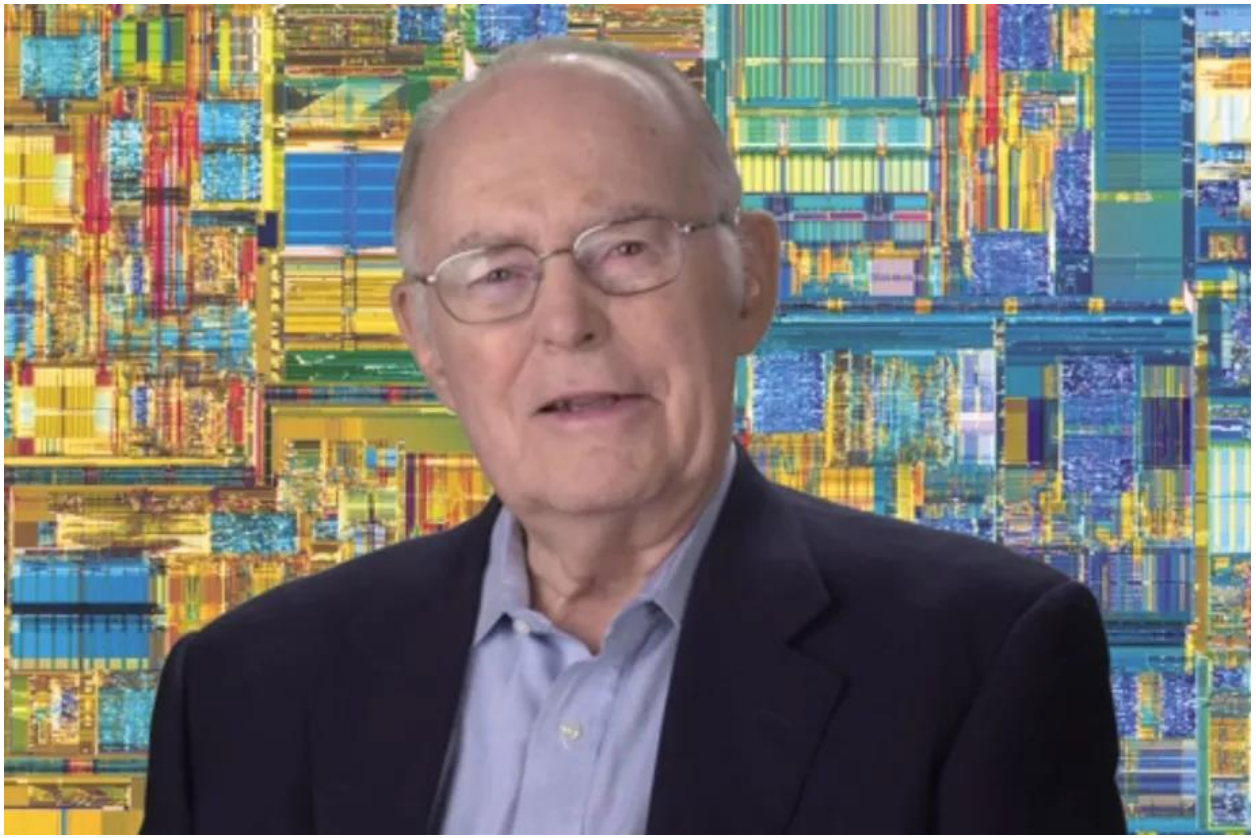
Cray-1 (1976)
~ 10 000 000$, 115 Kw
1.9 GFLOPS

iPad 2 (2011)
<1 000$
1.6 GFLOPS

La legge di Moore

Gordon Earle Moore, nato nel 1929

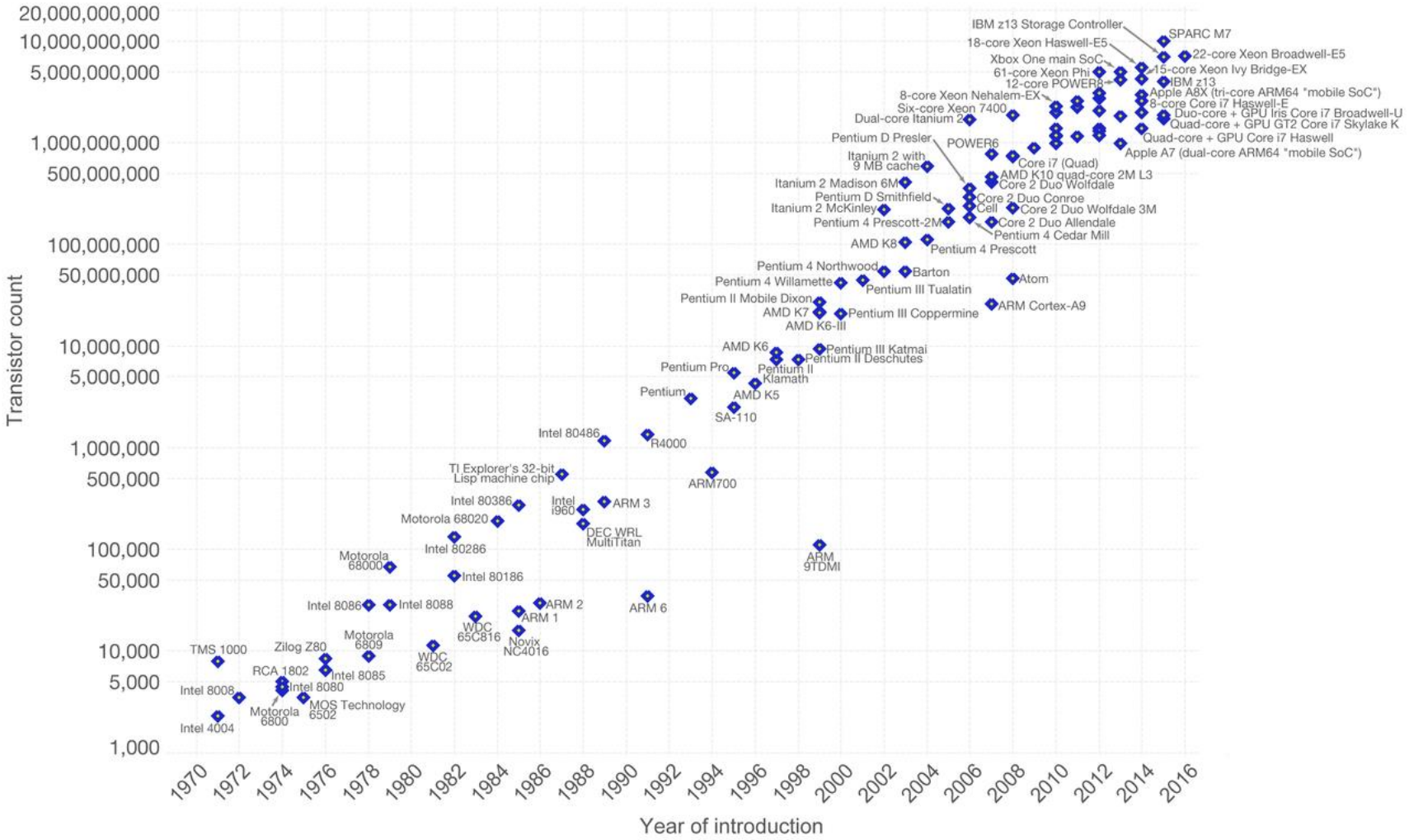
Co-fondatore di Intel



Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Our World in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

La legge di Moore è ancora valida?

DEATH NOTICE

Long beloved by both engineers and computer scientists because of ongoing performance benefits ceaselessly and seemingly effortlessly achieved. From the age of fifty, Moore's Law began to age rapidly, passing into senescence and then, at the beginning of this month, into oblivion. Moore's Law leaves a thriving multi-trillion dollar global semiconductor and electronics industry, along with a growing set of questions about how that industry will survive its passing. In lieu of flowers, donations should be lavished on Intel shares.

https://www.theregister.co.uk/2018/01/24/death_notice_for_moores_law/

Un’altra strategia per andare veloci: il parallelismo

Largest Amazon EC2 instance: 128 virtual CPUs

Azul Systems Vega 3
Cores per chip: 54
Cores per system: 864

The Future is Parallel

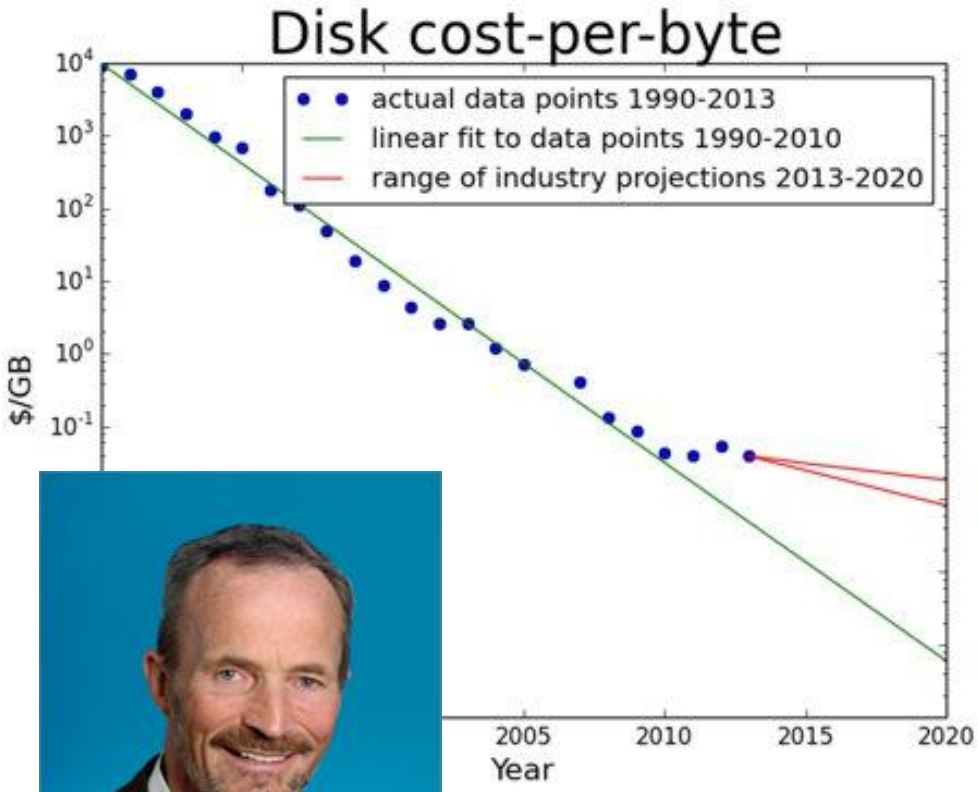
Intel Xeon
24 cores
48 threads

AMD Opteron
16 cores

Tilera Gx-3000
100 cores

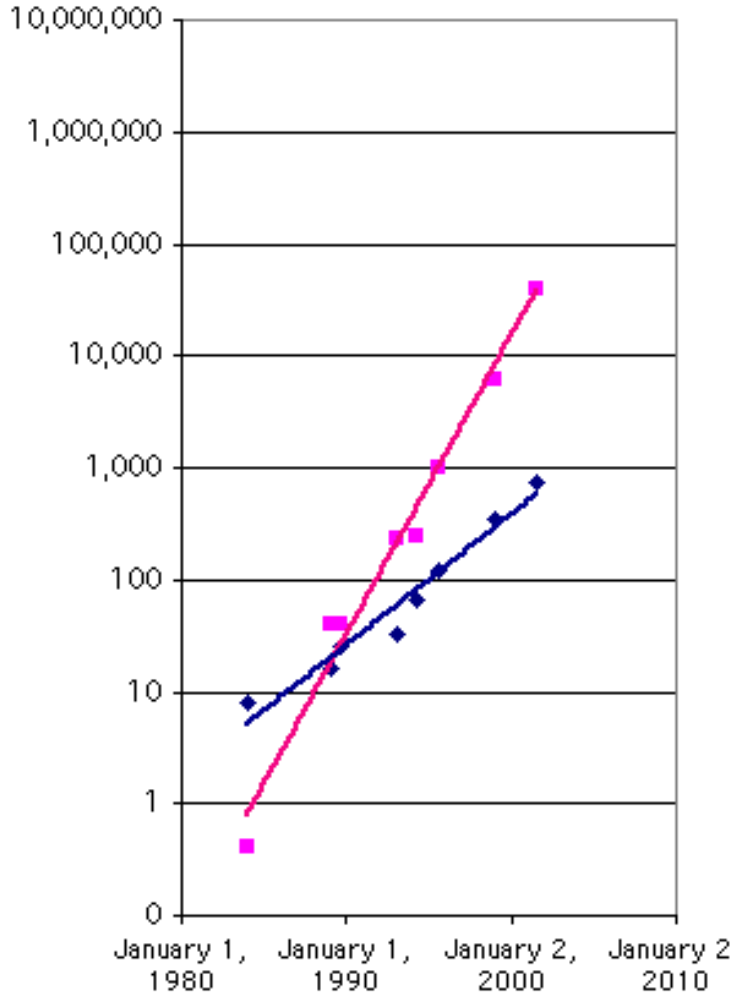
La legge di Kryder

Le prestazioni dei dischi crescono più velocemente della potenza di calcolo

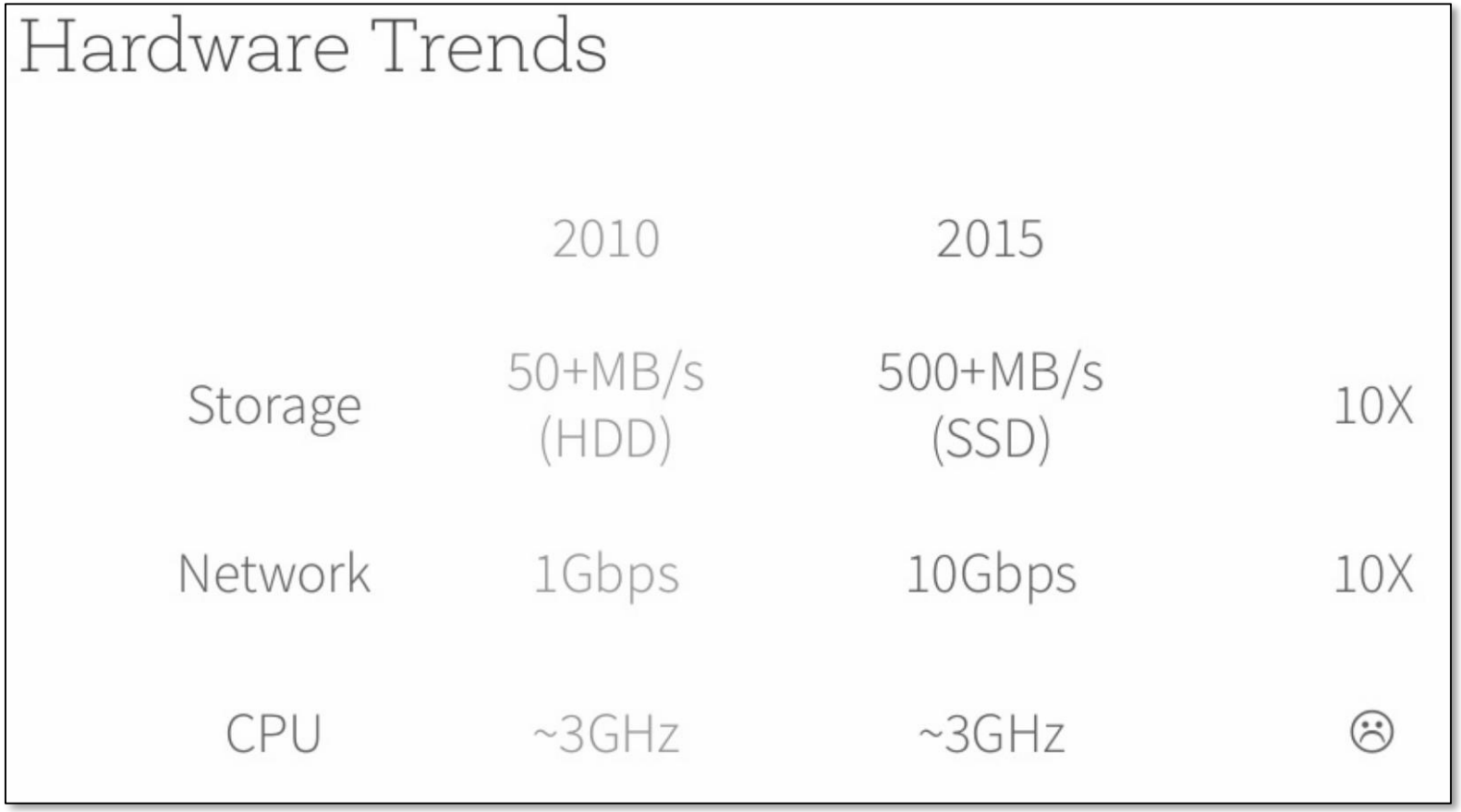


Mark H. Kryder

Legge di Kryder (viola)
vs.
Legge di Moore (blu)



Le prestazioni dei dischi crescono più velocemente della potenza di calcolo

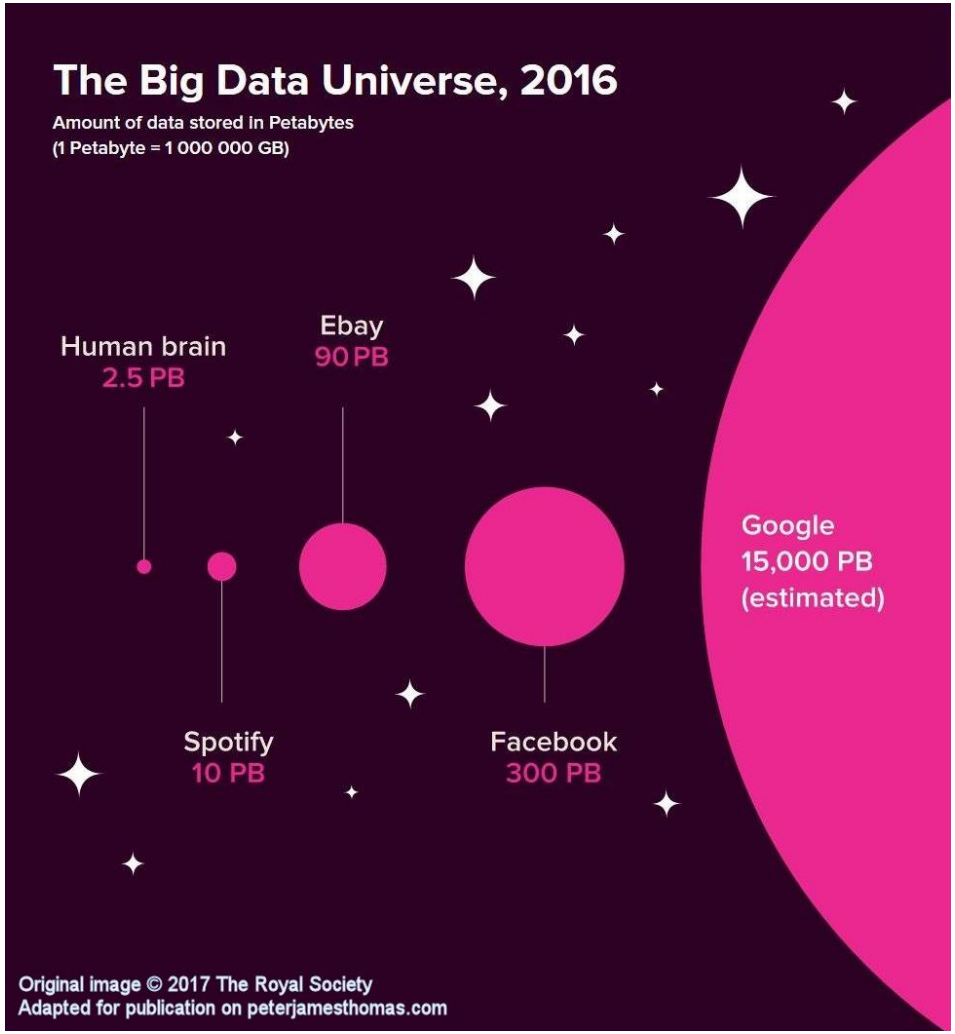


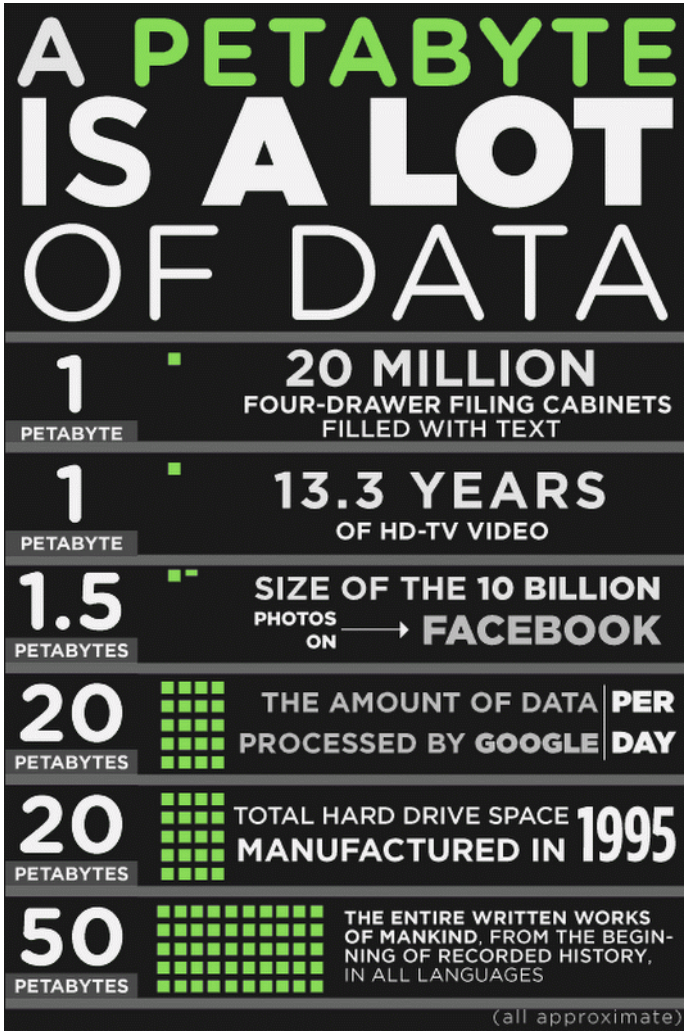
L’avvento dei BIG Data

BIG DATA:

- 500 milioni di tweets al giorno
- Traffico di quasi 2 GB/mese sui telefoni cellulari
- Amazon vende 600 prodotti al secondo
- 200 miliardi di email vengono scambiate ogni giorno
- MasterCard esegue 74 miliardi di transazioni ogni anno
- Ci sono quasi 6000 voli di linea al giorno.

Quanto sono «*BIG*» i Big Data?





Intermezzo : aspetta… quanti dati ha Google???

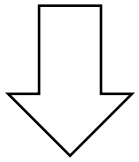
"If all digital data were stored on punch cards, how big would Google's data warehouse be?" ?

Randall Munroe

https://blog.ted.com/using-serious-math-to-answer-weird-questions-randall-munroe-at-ted2014/

Riassumendo

- La velocità di calcolo cresce, ma non tanto quanto vorremmo
- Lo storage cresce molto più velocemente
- Con uno storage economico e virtualmente illimitato la quantità di dati raccolti diventa terrificante



Il modo più efficace di analizzare questa enorme massa di dati è il *cluster computing*

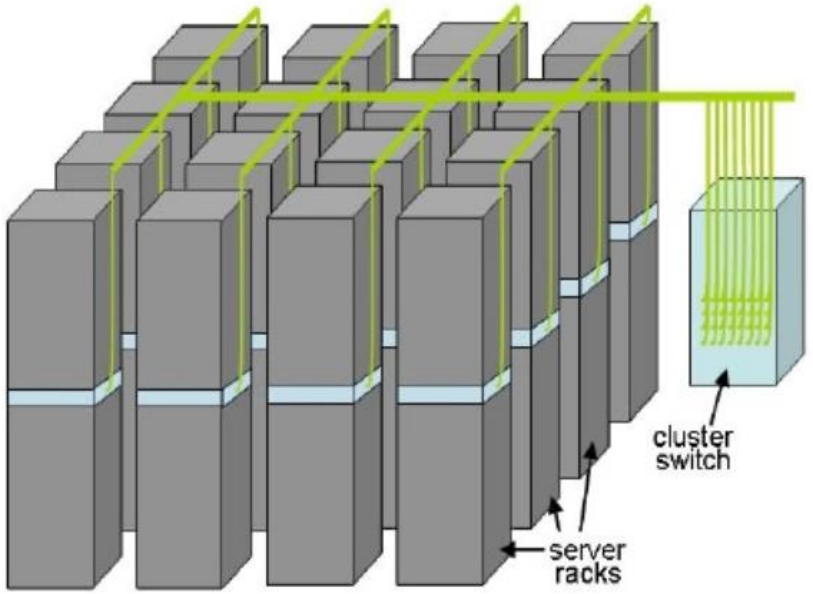
Esistono cluster con centinaia e anche migliaia di nodi!

Come si programma questa roba?

Come si programma questa roba?

Servers
- CPUs
- DRAM
- Disks

Racks
- 40-80 servers
- Ethernet switch



Clusters

Nel primo decennio del secolo Google pubblica tre articoli fondamentali

2003

GFS - Google File System

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google

ABSTRACT

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.

In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use.

Categories and Subject Descriptors

D [4]: 3—*Distributed file systems*

General Terms

Design, reliability, performance, measurement

Keywords

Fault tolerance, scalability, data storage, clustered storage

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application bugs, operating system bugs, human errors, and the failures of disks, memory, connectors, networking, and power supplies. Therefore, constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system.

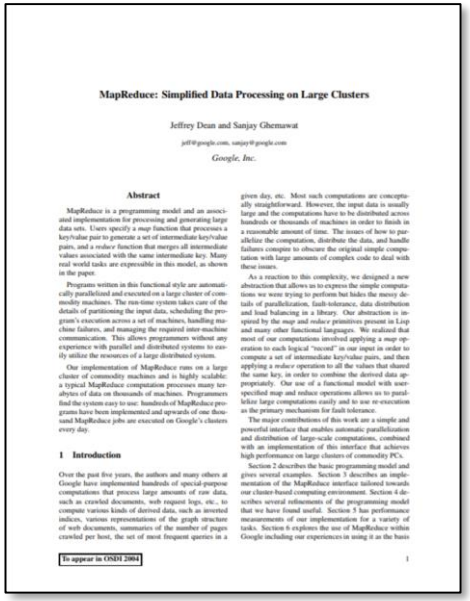
Second, files are huge by traditional standards. Multi-GB files are common. Each file typically contains many application objects such as web documents. When we are regularly working with fast growing data sets of many TBs comprising billions of objects, it is unwieldy to manage billions of approximately KB-sized files even when the file system could support it. As a result, design assumptions and parameters such as I/O operation and block sizes have to be revisited.

Third, most files are mutated by appending new data rather than overwriting existing data. Random writes within a file are practically non-existent. Once written, the files are only read, and often only sequentially. A variety of data share these characteristics. Some may constitute large repositories that data analysis programs scan through. Some may be data streams continuously generated by running applications. Some may be archival data. Some may be intermediate results produced on one machine and processed on another, whether simultaneously or later in time. Given this access pattern on huge files, appending becomes the focus of performance optimization and atomicity guarantees, while caching data blocks in the client loses its appeal.

Fourth, co-designing the applications and the file system API benefits the overall system by increasing our flexibility.

2004

MapReduce



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

Google, Inc.

Abstract

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a *map* function that processes a key/value pair to generate a set of intermediate key/value pairs, and a *reduce* function that merges all intermediate values associated with the same intermediate key. Many real world tasks are expressible in this model, as shown in the paper.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.

Our implementation of MapReduce runs on a large cluster of commodity machines and is highly scalable: a typical MapReduce computation processes many terabytes of data on thousands of machines. Programmers find the system easy to use: hundreds of MapReduce programs have been implemented and upwards of one thousand MapReduce jobs are executed on Google's clusters every day.

1 Introduction

Over the past five years, the authors and many others at Google have implemented hundreds of special-purpose computations that process large amounts of raw data, such as crawled documents, web request logs, etc., to compute various kinds of derived data, such as inverted indices, various representations of the graph structure of web documents, summaries of the number of pages crawled per host, the set of most frequent queries in a given day, etc. Most such computations are conceptually straightforward. However, the input data is usually large and the computations have to be distributed across hundreds or thousands of machines in order to finish in a reasonable amount of time. The issues of how to parallelize the computation, distribute the data, and handle failures conspire to obscure the original simple computation with large amounts of complex code to deal with these issues.

As a reaction to this complexity, we designed a new abstraction that allows us to express the simple computations we were trying to perform but hides the messy details of parallelization, fault-tolerance, data distribution and load balancing in a library. Our abstraction is inspired by the *map* and *reduce* primitives present in Lisp and many other functional languages. We realized that most of our computations involved applying a *map* operation to each logical "record" in our input in order to compute a set of intermediate key/value pairs, and then applying a *reduce* operation to all the values that shared the same key, in order to combine the derived data appropriately. Our use of a functional model with user-specified map and reduce operations allows us to parallelize large computations easily and to use re-execution as the primary mechanism for fault tolerance.

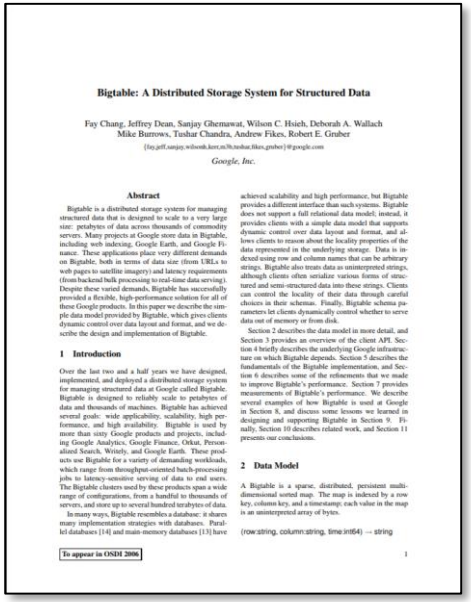
The major contributions of this work are a simple and powerful interface that enables automatic parallelization and distribution of large-scale computations, combined with an implementation of this interface that achieves high performance on large clusters of commodity PCs.

Section 2 describes the basic programming model and gives several examples. Section 3 describes an implementation of the MapReduce interface tailored towards our cluster-based computing environment. Section 4 describes several refinements of the programming model that we have found useful. Section 5 has performance measurements of our implementation for a variety of tasks. Section 6 explores the use of MapReduce within Google including our experiences in using it as the basis

To appear in OSDI 2004

2006

BigTable



Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

Google, Inc.

Abstract

Bigtable is a distributed storage system for managing structured data that is designed to scale to a very large size: petabytes of data across thousands of commodity servers. Many projects at Google store data in Bigtable, including web indexing, Google Earth, and Google Finance. These applications place very different demands on Bigtable, both in terms of data size (from URLs to web pages to satellite imagery) and latency requirements (from backend bulk processing to real-time data serving). Despite these varied demands, Bigtable has successfully provided a flexible, high-performance solution for all of these Google products. In this paper we describe the simple data model provided by Bigtable, which gives clients dynamic control over data layout and format, and we describe the design and implementation of Bigtable.

1 Introduction

Over the last two and a half years we have designed, implemented, and deployed a distributed storage system for managing structured data at Google called Bigtable. Bigtable is designed to reliably scale to petabytes of data and thousands of machines. Bigtable has achieved several goals: wide applicability, scalability, high performance, and high availability. Bigtable is used by more than sixty Google products and projects, including Google Analytics, Google Finance, Orkut, Personalized Search, Writely, and Google Earth. These products use Bigtable for a variety of demanding workloads, which range from throughput-oriented batch-processing jobs to latency-sensitive serving of data to end users. The Bigtable clusters used by these products span a wide range of configurations, from a handful to thousands of servers, and store up to several hundred terabytes of data.

In many ways, Bigtable resembles a database: it shares many implementation strategies with databases. Parallel databases [14] and main-memory databases [13] have achieved scalability and high performance, but Bigtable provides a different interface than such systems. Bigtable does not support a full relational data model; instead, it provides clients with a simple data model that supports dynamic control over data layout and format, and allows clients to reason about the locality properties of the data represented in the underlying storage. Data is indexed using row and column names that can be arbitrary strings. Bigtable also treats data as uninterpreted strings, although clients often serialize various forms of structured and semi-structured data into these strings. Clients can control the locality of their data through careful choices in their schemas. Finally, Bigtable schema parameters let clients dynamically control whether to serve data out of memory or from disk.

Section 2 describes the data model in more detail, and Section 3 provides an overview of the client API. Section 4 briefly describes the underlying Google infrastructure on which Bigtable depends. Section 5 describes the fundamentals of the Bigtable implementation, and Section 6 describes some of the refinements that we made to improve Bigtable's performance. Section 7 provides measurements of Bigtable's performance. We describe several examples of how Bigtable is used at Google in Section 8, and discuss some lessons we learned in designing and supporting Bigtable in Section 9. Finally, Section 10 describes related work, and Section 11 presents our conclusions.

2 Data Model

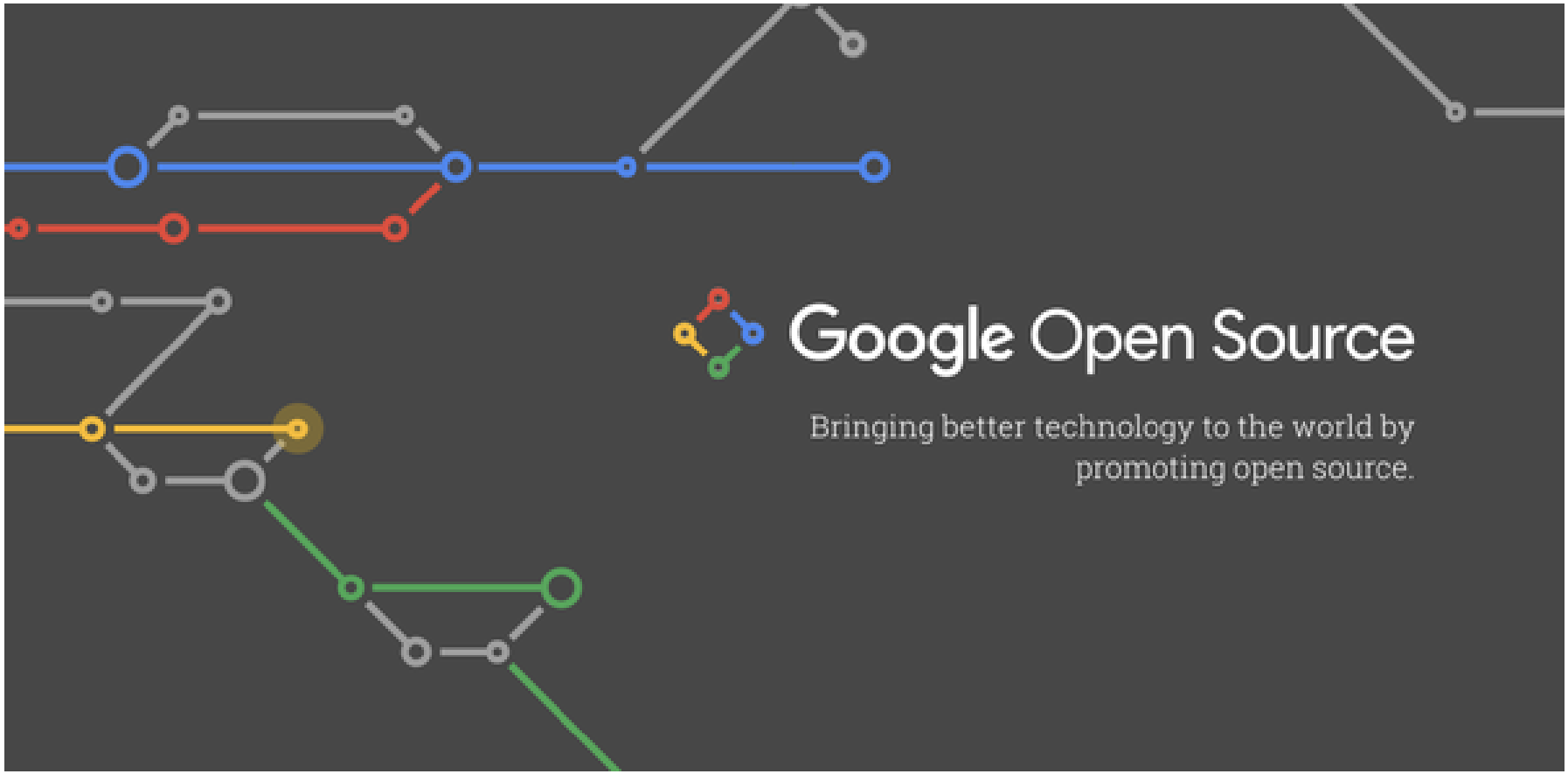
A Bigtable is a sparse, distributed, persistent multi-dimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes.

(row:string, column:string, time:int64) → string

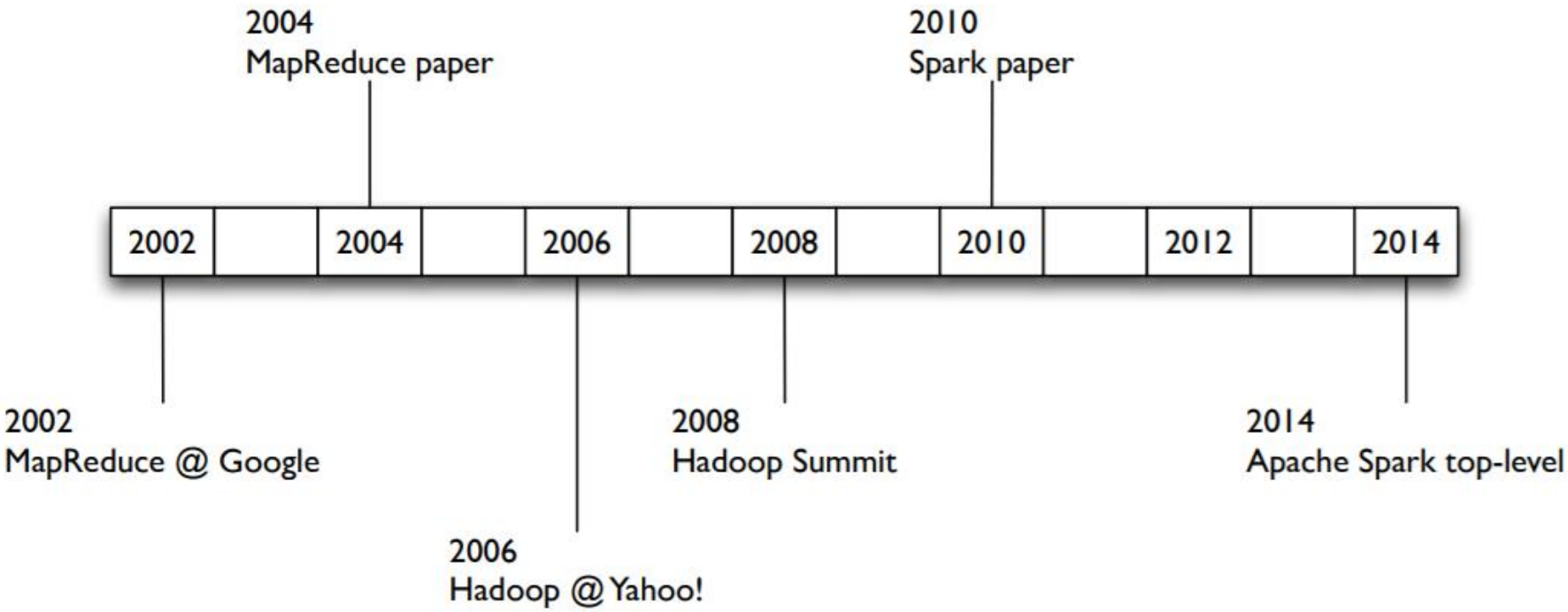
To appear in OSDI 2006

ma non pubblica il codice dell'implementazione!

… poi (un po’ in ritardo) abbraccia l’Open Source



Nel frattempo nascono Hadoop e Spark



Subito prima della nascita di Spark

Hadoop è la soluzione di riferimento per i Big Data:

- HDFS per lo storage e
- MapReduce per il calcolo

MapReduce ha delle limitazioni: è relativamente lento e difficile da programmare.

Inoltre gli utenti cominciano a richiedere:

- Algoritmi più complessi, basati su passi multipli
- Query ad-hoc interattive
- Processing in real time

Queste feature sono difficilmente implementabili in MapReduce

Subito prima della nascita di Spark

Per ovviare a queste limitazioni vengono sviluppati dei sistemi *specializzati*, uno per ogni tipo di esigenza:

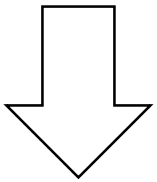
- Pregel
- Giraph
- Dremel
- Drill
- Impala
- Presto
- Storm
- S4
- ….

Subito prima della nascita di Spark

Problemi con i sistemi specializzati

- Molti sistemi da controllare, configurare e installare

- Difficile combinarli fra loro anche se spesso servirebbe (ad es. leggere dati con SQL e poi usare degli algoritmi di machine learning)
  $\Rightarrow$ In molti casi il trasferimento dei dati fra due sistemi diventa un costo dominante



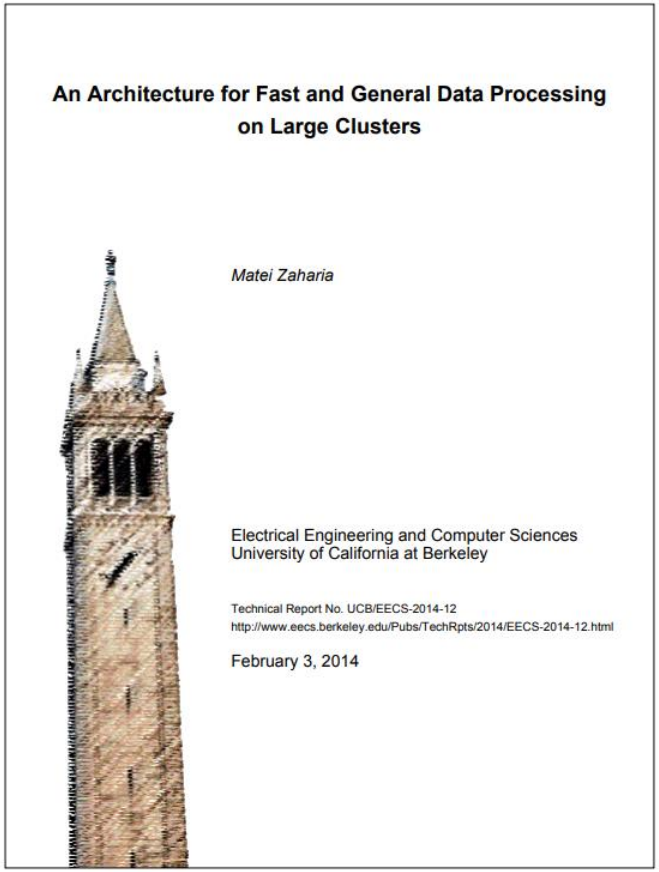
Spark nasce come un engine unificato

2009 : Nascita di Spark

Un gruppo di ricerca a UC Berkeley

Il gruppo lavora su un framework per il controllo di cluster che possa sostenere diversi tipi di cluster computing system. Spark nasce come esempio.

All'inizio focalizzato sul *machine learning*, poi diventa uno strumento generale



An Architecture for Fast and General Data Processing on Large Clusters

Matei Zaharia

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-12
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-12.html

February 3, 2014

https://www.kdnuggets.com/2015/05/interview-matei-zaharia-creator-apache-spark.html

2009 : Nascita di Spark

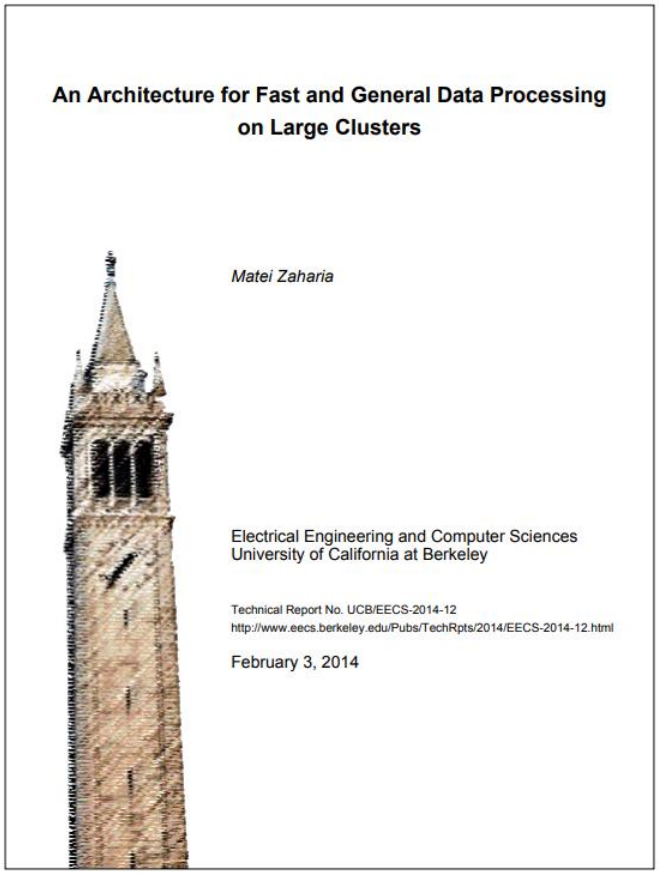
Un gruppo di ricerca a UC Berkeley

Due idee chiave:

- Tenere i risultati intermedi in memoria
- Fare uno strumento generale che possa essere utilizzato in contesti differenti (es. streaming e graph processing)

Nel 2013 si muove in Apache

Nel 2014 diventa un ***Top-Level Project*** (TLP)



An Architecture for Fast and General Data Processing on Large Clusters

Matei Zaharia

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2014-12
http://www.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-12.html

February 3, 2014

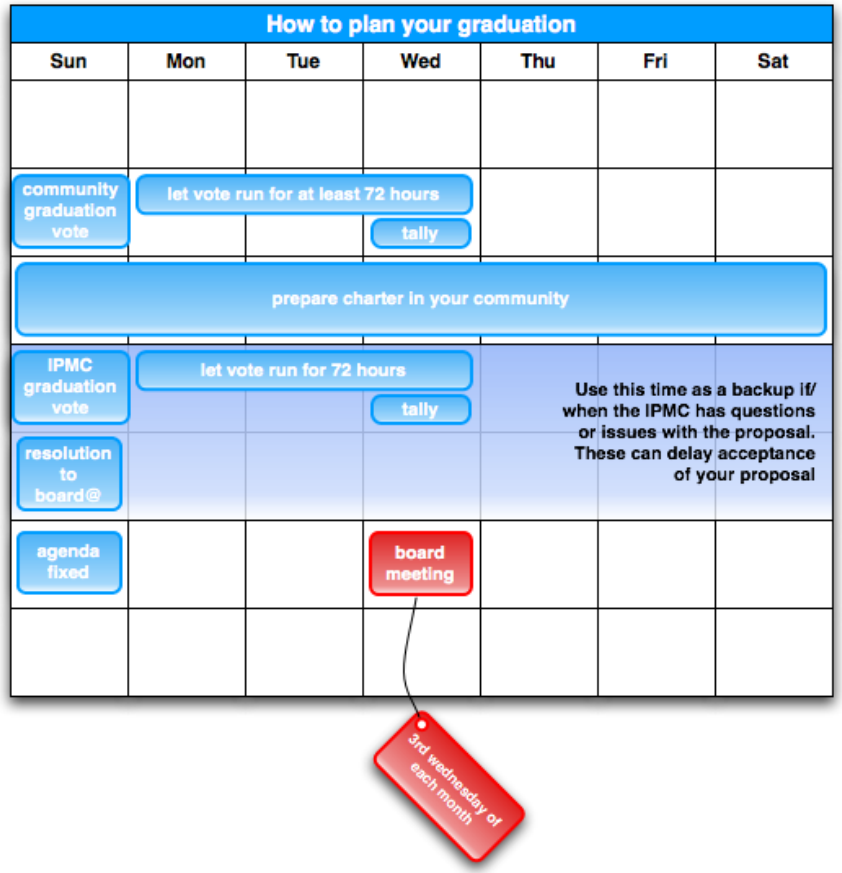
https://www.kdnuggets.com/2015/05/interview-matei-zaharia-creator-apache-spark.html

2014: Spark è Apache Top Level Project

APACHE INCUBATOR

Diventare TPL è un processo complicato e laborioso.

Quando un progetto diventa TLP può contare su una comunità grande e attiva e su una grande visibilità



Apache is a well recognized brand

GOVERNOR ARNOLD SCHWARZENEGGER

November 5, 2009

Apache Software Foundation

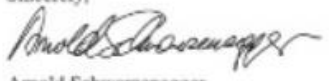
It is a great pleasure to extend my greetings to all those attending ApacheCon and congratulations on your tenth anniversary.

I applaud your incredible work over the past decade and appreciate you choosing California as the place to celebrate this fantastic milestone. Our state is a land of innovation, and you have likewise fostered great technological advancements that have touched the lives of millions of people around the world.

Whether managing financial systems, positioning satellites or powering websites through the Apache HTTP Server, your open source projects play key roles in making our information age possible. Thank you for your extraordinary accomplishments and commitment to discovery.

On behalf of all Californians, I send my gratitude to everyone in attendance for your participation, and I offer my best wishes for a rewarding conference and continued success.

Sincerely,



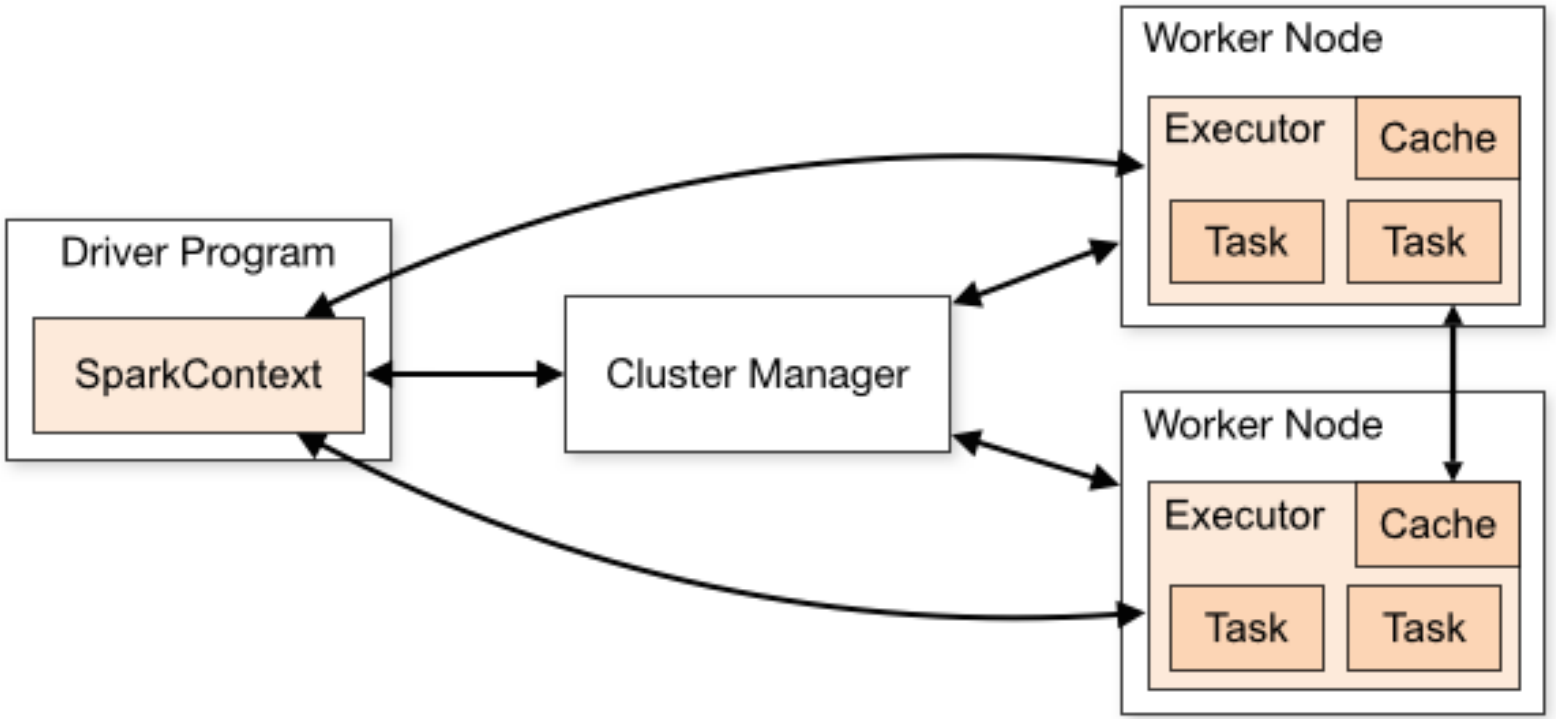
Arnold Schwarzenegger

Spark,

com’è fatto?

Struttura di un programma Spark

L'elaborazione è distribuita su più nodi

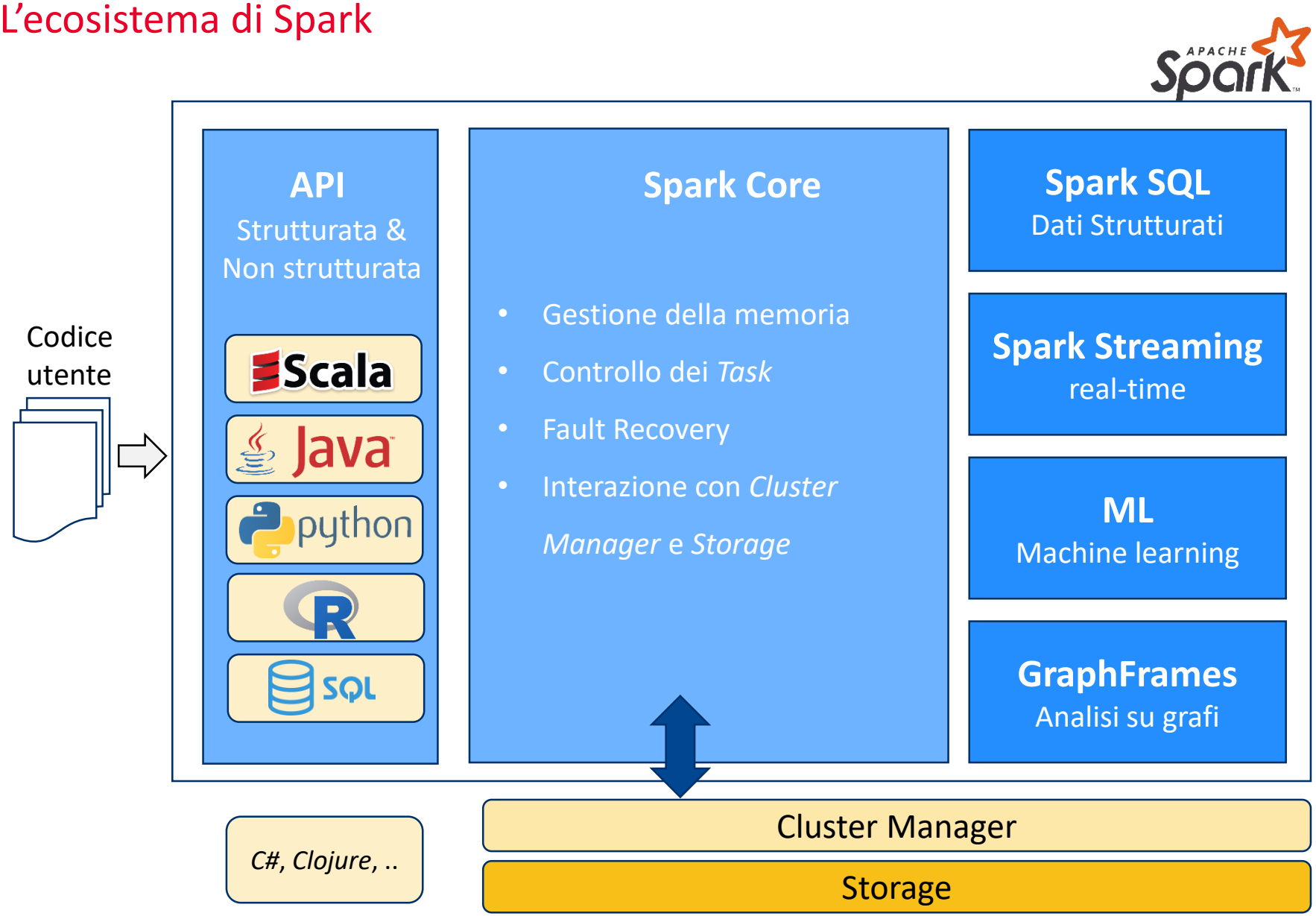


L’ecosistema di Spark

Cluster manager

Storage

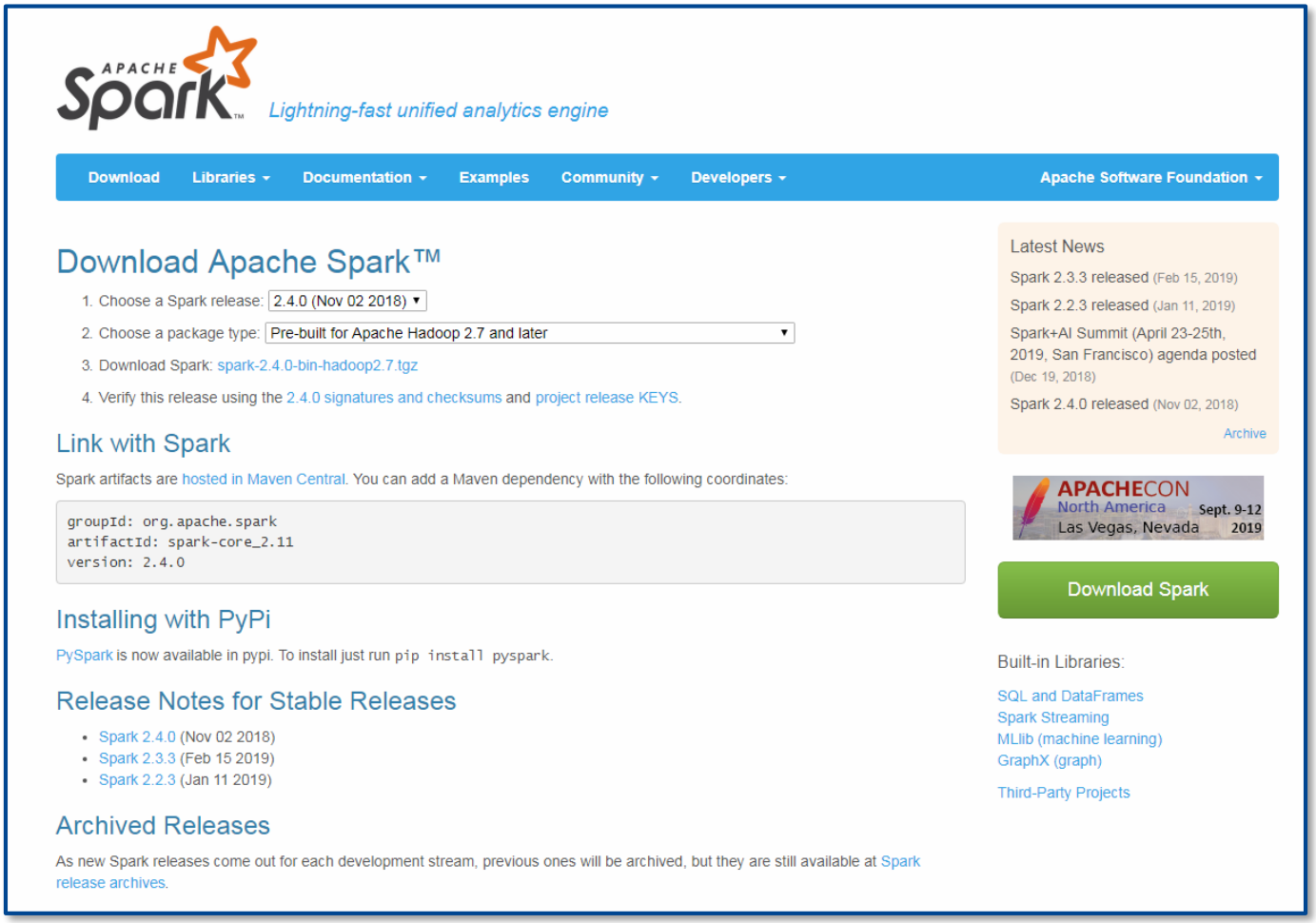
L'ecosistema di Spark



In pratica?

Per fare degli esperimenti a casa

https://spark.apache.org/downloads.html

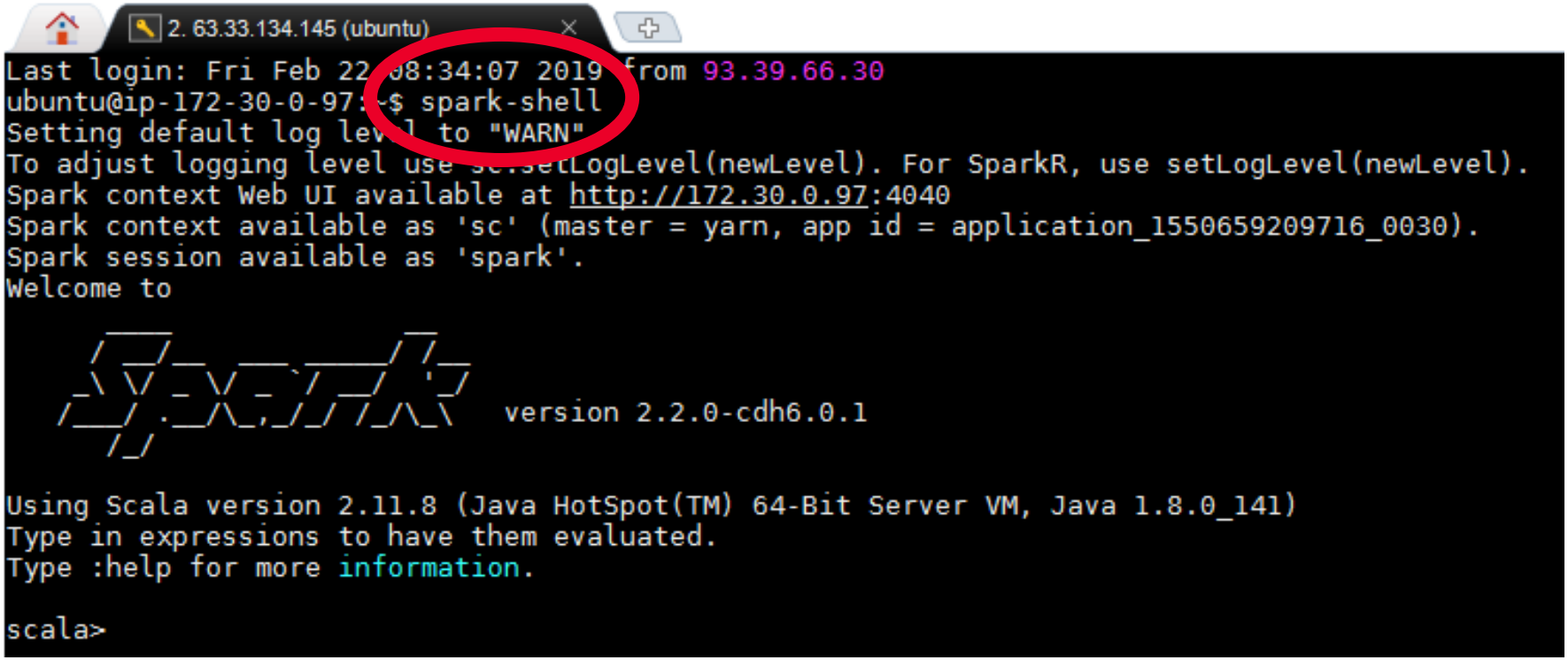


Spark gira anche su un singolo computer

Lo stesso codice funziona su un cluster di 1000 nodi!

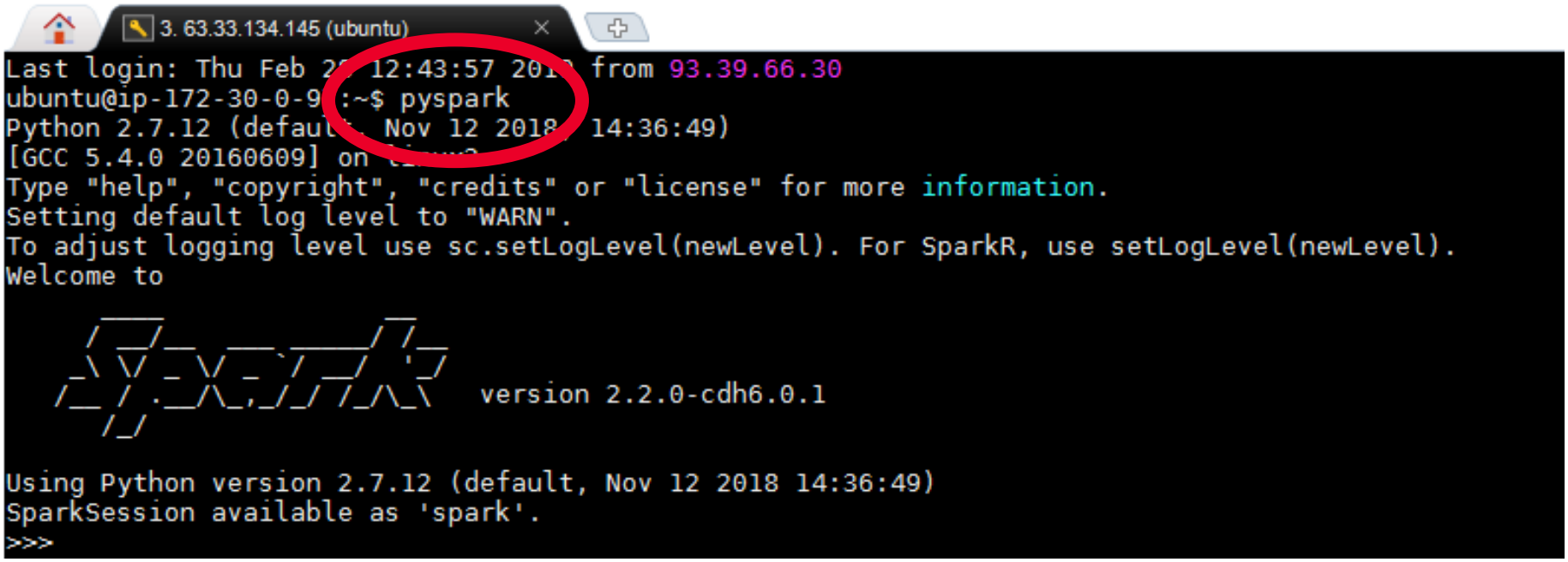
Come useremo Spark nel laboratorio di oggi?

La Shell di Spark



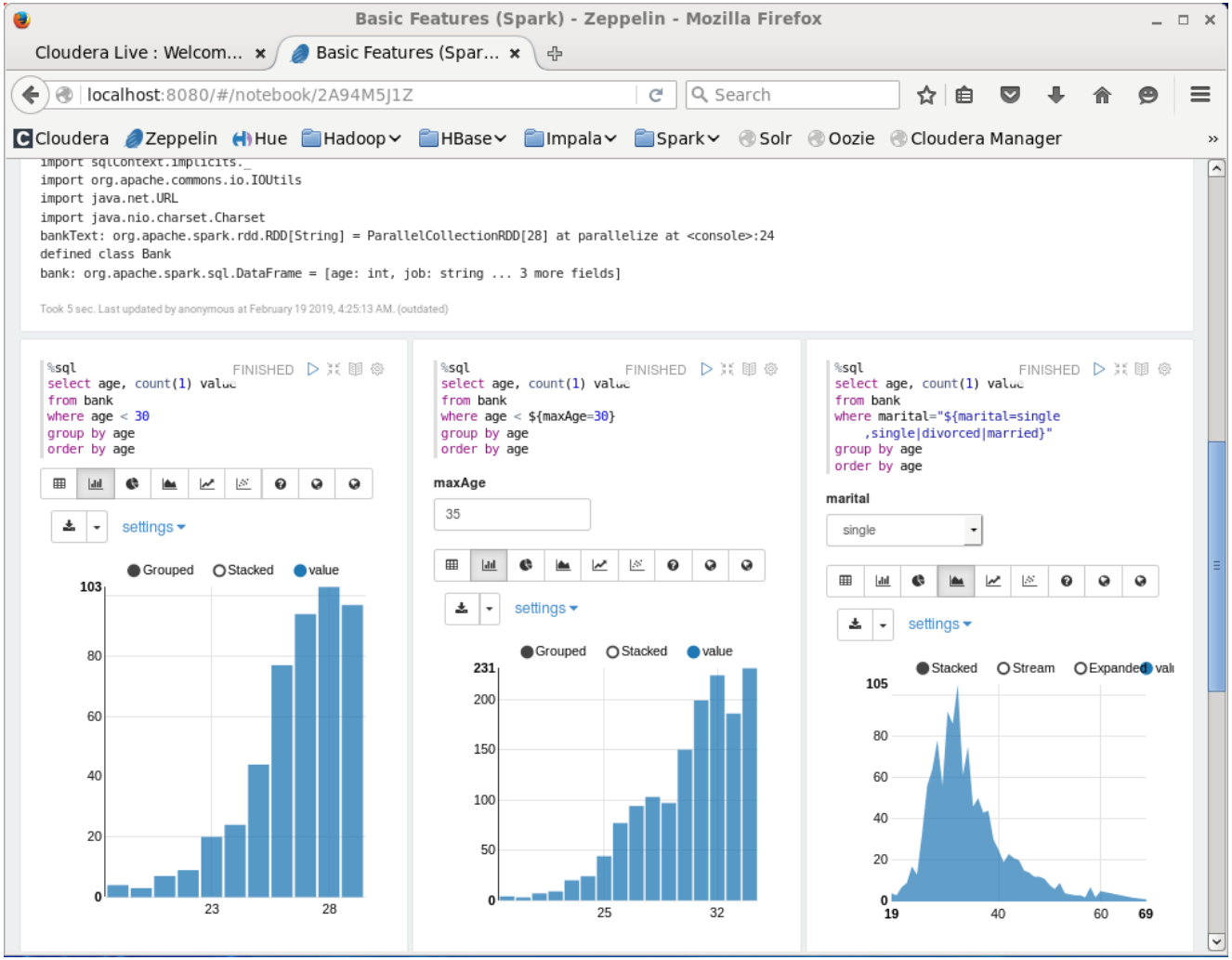
Come useremo Spark nel laboratorio di oggi?

Si può usare anche Python (pyspark)



Come useremo Spark nel laboratorio di oggi?

Zeppelin Notebook



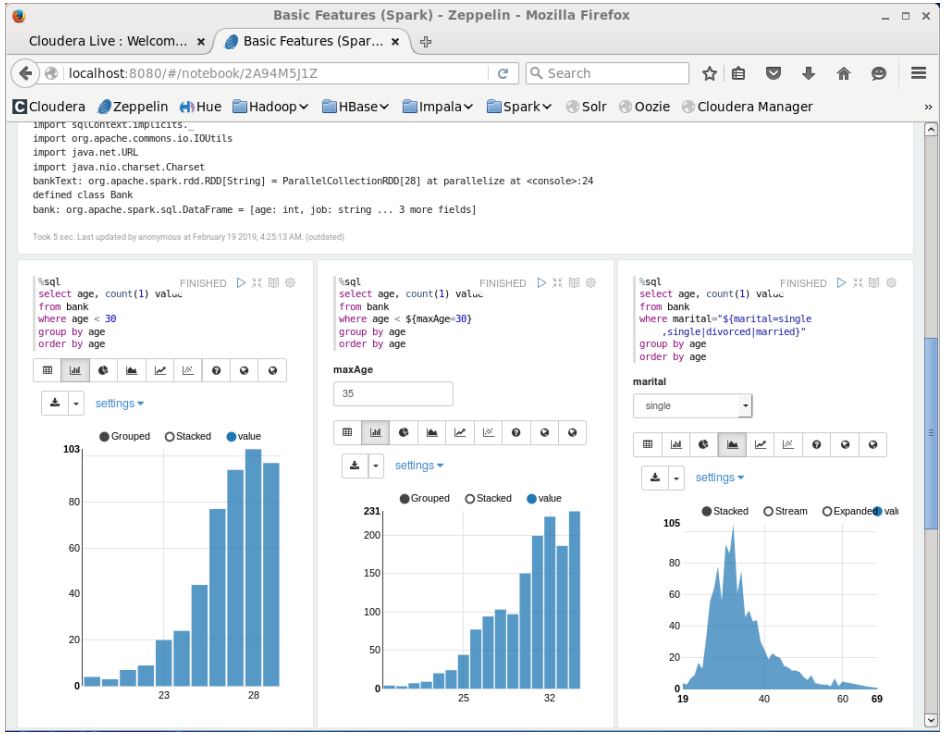
I Notebook

Un ambiente per il «literate programming»

Un’interfaccia che permette di mescolare frammenti eseguibili di programma con grafici, testo e altri contenuti.

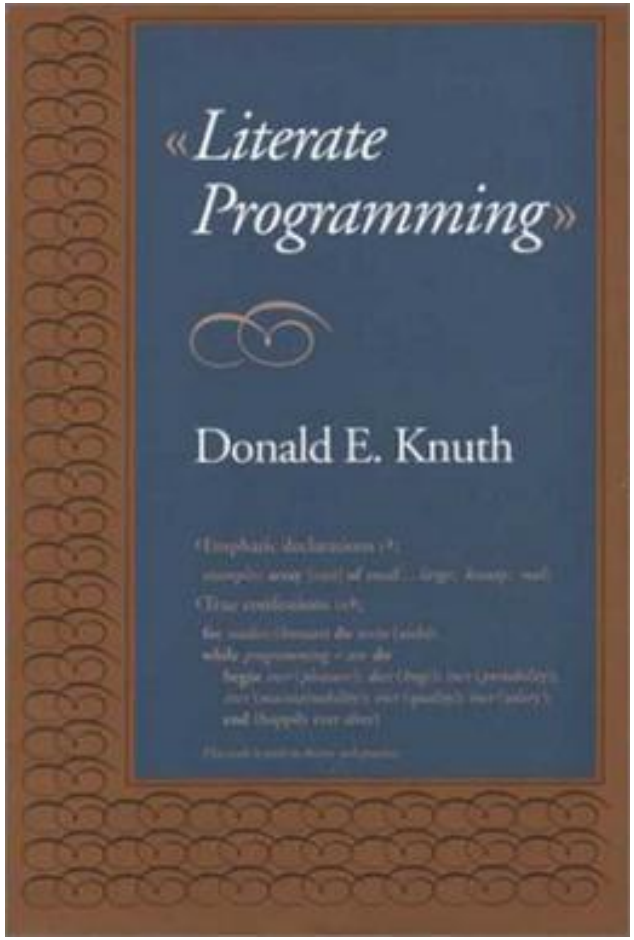
I notebook per Apache Spark sono:

- *Jupyter Notebook*
- *Apache Zeppelin*
- *Apache Spark Notebook*



Literate programming

Donald E. Knuth



Laboratorio

Obiettivi per questa mattina

- Ripasso di Scala
- Prendere familiarità con *Zeppelin Notebook*

Scala Cheat Sheet

variances

Given T <: P
When C is covariant then C[T] <: C[P]
When C is contravariant then C[T] >: C[P]

class C[+T]	C is covariant
class C[-T]	C is contravariant
class C[T]	C is invariant

control constructs

if (condition) this else that	if then else
while (x < 5) { println(x); x += 1}	while loop
do { println(x); x += 1} while (x < 5)	do while loop
for (x <- xs if x%2 == 0) yield x*10	for comprehension
for (i <- 1 to 5) { println(i) }	for loop

variables and functions

val x = 1	constant, evaluated immediately
var y = 2	variable
def z = 2	evaluated when called
lazy val w = 2	evaluated once when needed
def f(x: Int) = { x*x }	define function
def f(x: R)	call by value
def f(x: => R)	call by name (lazy parameters)
(1 to 5).map(_*2)	anonymous function (positionally matched arguments)
(1 to 5).map(2*)	anonymous function (bound infix method)
(1 to 5).map(x => x*x)	anonymous function (to use an arg twice, have to name it)
def mapmake[T](g:T=>T)(seq: List[T]) = seq.map(g)	generic type
def sum(args: Int*) = args.reduceLeft(_+_)	varargs

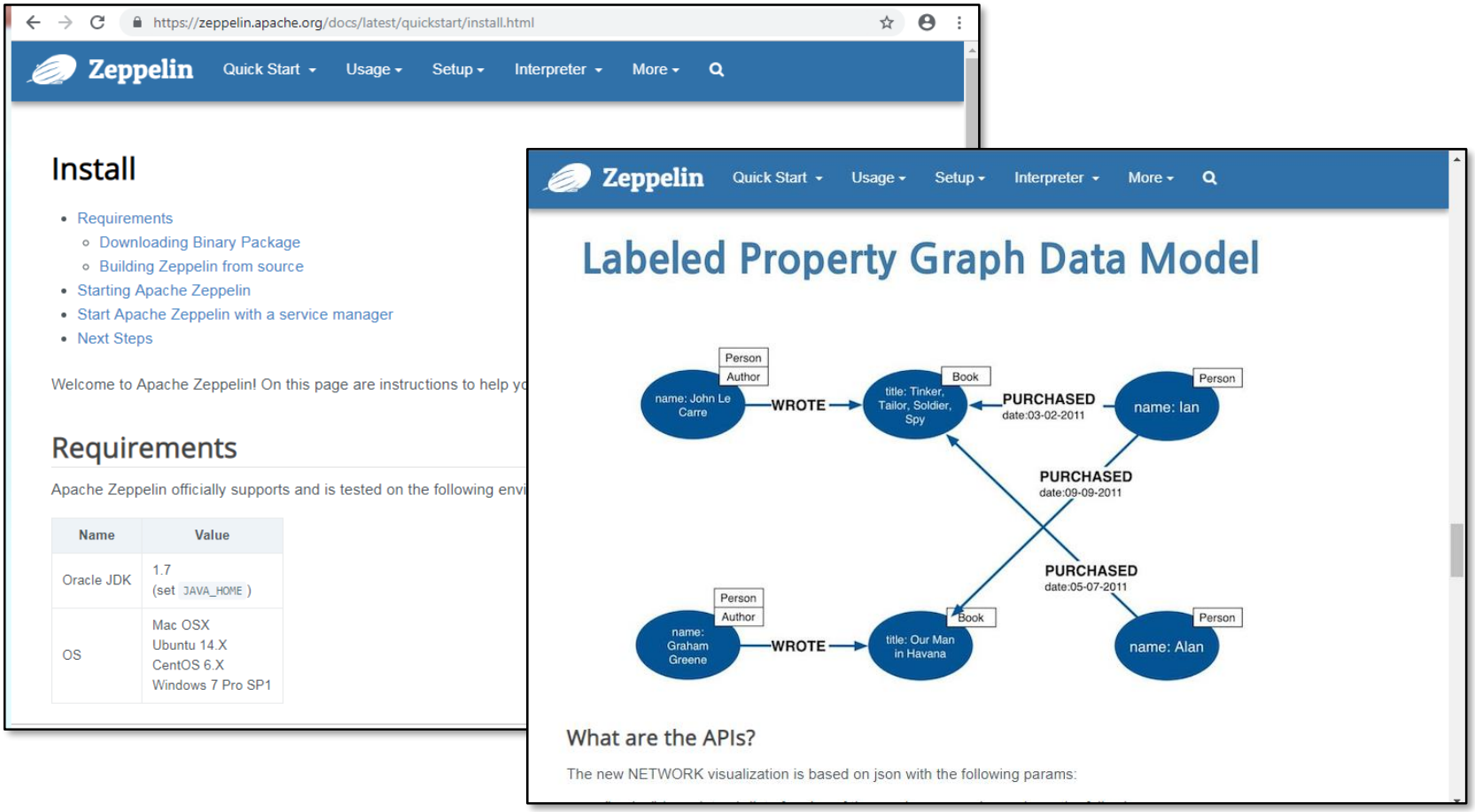
object orientation

class C(x: R)	class with private param
class C(val x: R)	class with public param
new { ... }	anonymous class
abstract class D { ... }	abstract class
class C extends D { ... }	inheritance
class C(x: R) extends D(x)	inheritance and constructor params
object O extends D { ... }	singleton
trait T { ... }	interfaces with implementation
class C extends D with T	using multiple traits

gutefrage.net

Zeppelin Notebook

Manuale online



https://zeppelin.apache.org/

KEEP CALM AND START CODING