TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Introduction to Hadoop MapReduce and Spark

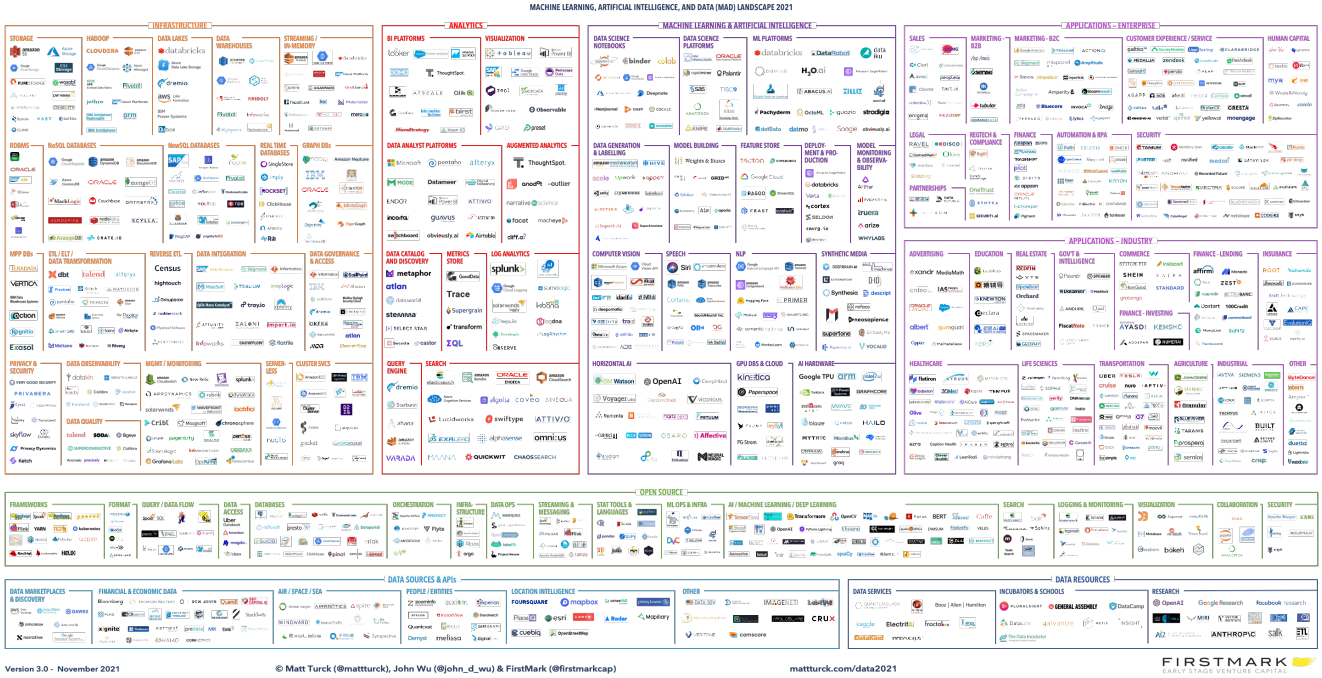
Corso di Sistemi e Architetture per Big Data

A.A. 2021/22

Valeria Cardellini

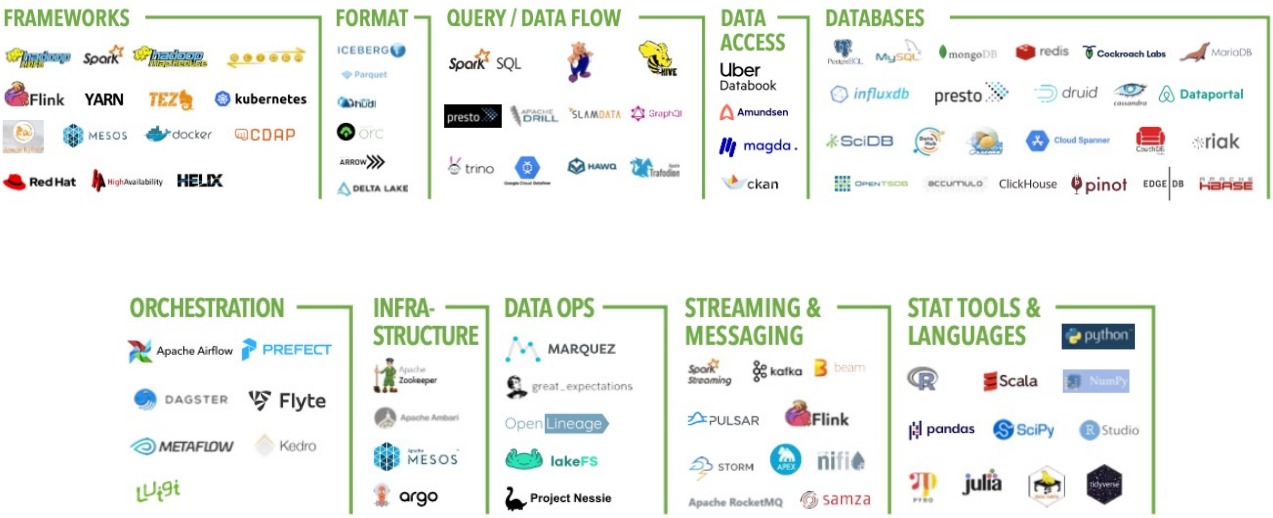
Laurea Magistrale in Ingegneria Informatica

The ML, AI and Data (MAD) landscape

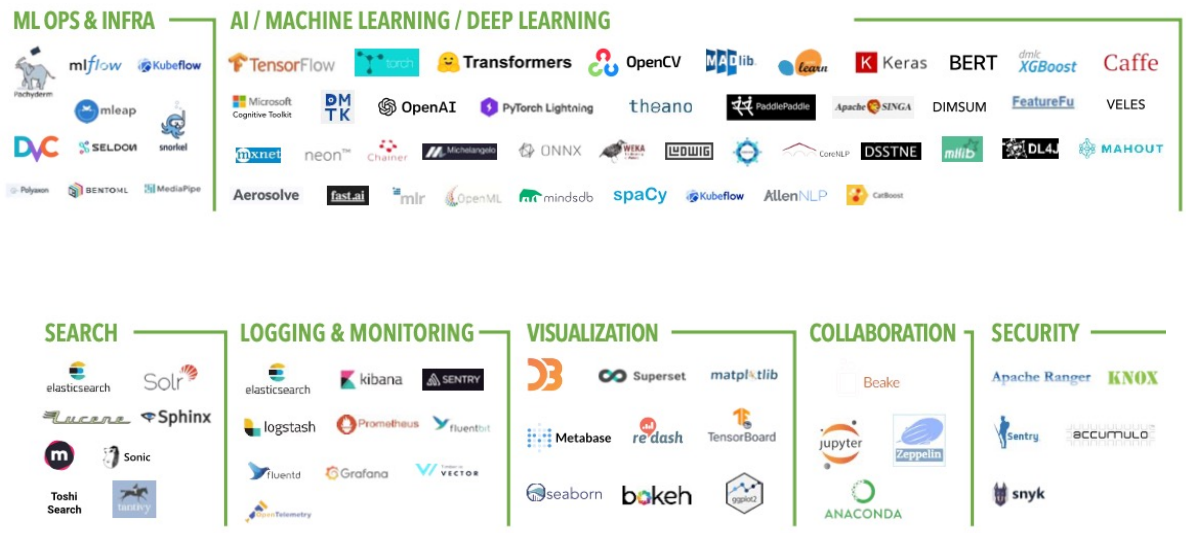


https://mattturck.com/data2021/

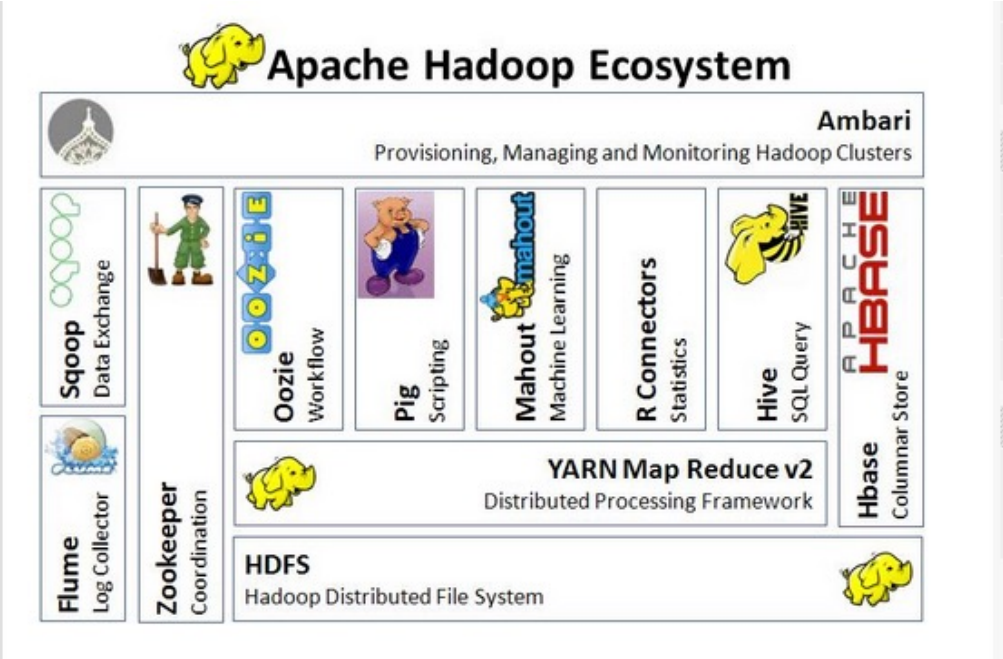
Zooming on open-source



Zooming on open-source

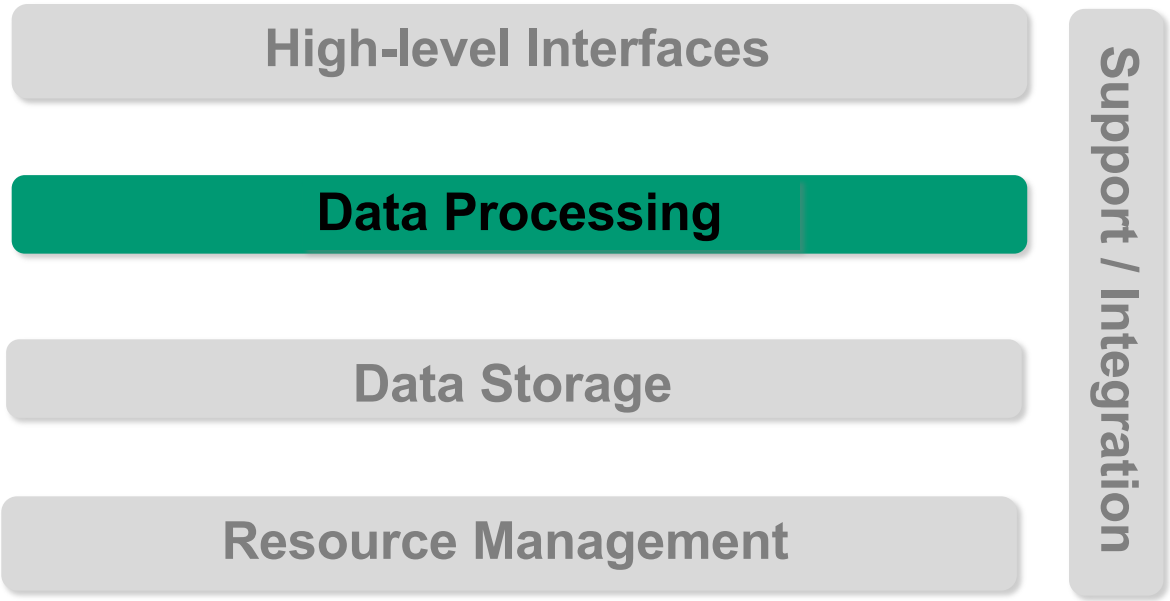


The basic Hadoop ecosystem



See https://hadoopecosystemtable.github.io for more products

The reference Big Data stack



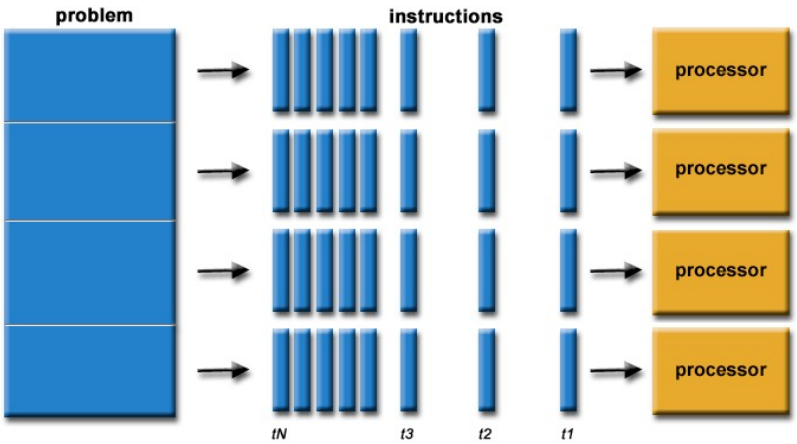
Processing Big Data

- Frameworks to process Big Data
  - Relational DBMS: old fashion
  - NoSQL data stores and NewSQL databases
  - **Batch processing**: store and process data sets at massive scale (especially Volume+Variety)
    - MapReduce and Hadoop
    - Spark
  - **Data stream processing**: process fast data (in real-time) as data is being generated, without storing

Our focus

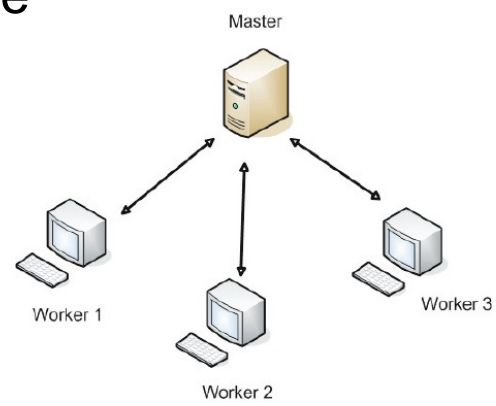
Parallel programming: background

- Parallel programming
  - Simultaneous use of multiple computing resources (e.g., processors) to solve a problem
  - How? Break processing into parts that can be executed concurrently on multiple computing resources



Parallel programming: background

- Simplest environment for parallel programming
  - **Master/worker** architecture
    - Master
      - Gets data and splits it into chunks according to the number of workers
      - Sends each worker equal number of chunks
      - Receives results from each worker
    - Workers:
      - Receive some chunks of data from master
      - Perform processing
      - Send back results to master



Parallel programming: background

- There are several styles of parallel programming
- Single Program, Multiple Data (**SPMD**) is the most commonly used
  - Single Program: all computing resources execute the same program simultaneously
  - Multiple Data: all computing resources may use different data

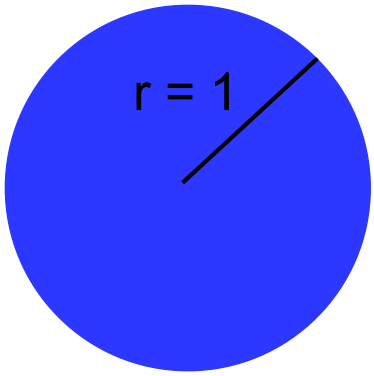
Example: Pi estimation

- Estimation algorithm for calculating π
  - Relies on Monte Carlo method

- Let's first consider the sequential version of the algorithm
- Then, how to realize a parallel and faster version

Example: Pi estimation

- By definition, π is the area of a circle with radius equal to 1



Example: Pi estimation

- How to estimate π?

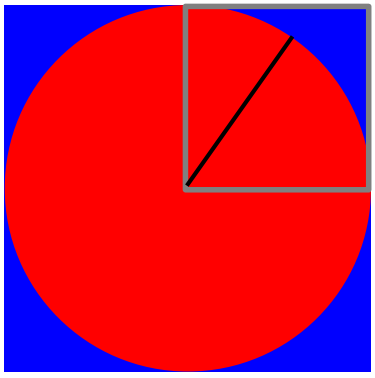
1. Pick a large number of points randomly inside the circumscribed unit square
   - A certain number of these points will end up inside the area described by the circle, while the remaining number of these points will lie outside of it (but inside the square)
2. Count the fraction of points that end up inside the circle out of a total population of points randomly thrown at the circumscribed square

Example: Pi estimation

- In formulas:

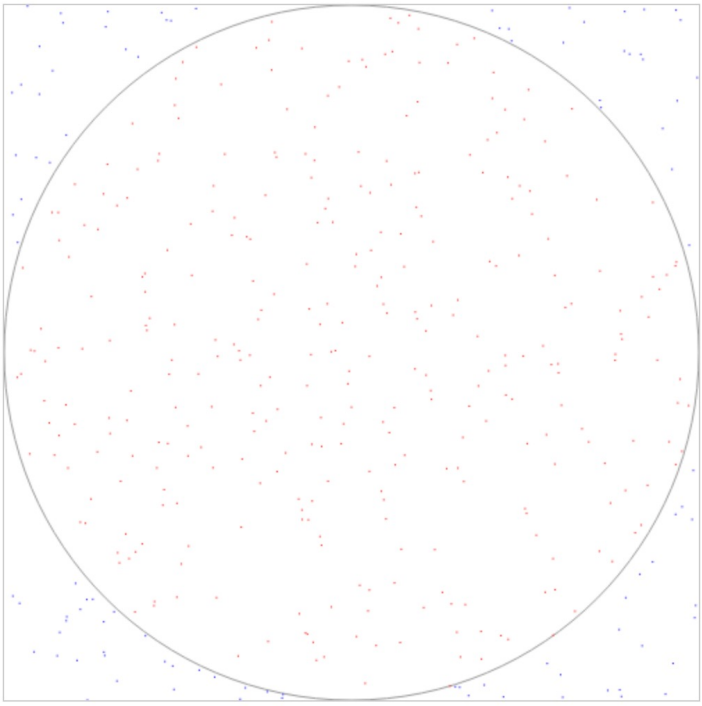
$$\frac{\pi}{4} \approx \frac{N_{inner}}{N_{total}}$$

$$\pi \approx 4\frac{N_{inner}}{N_{total}}$$



- The more points generated, the greater the accuracy of the estimation

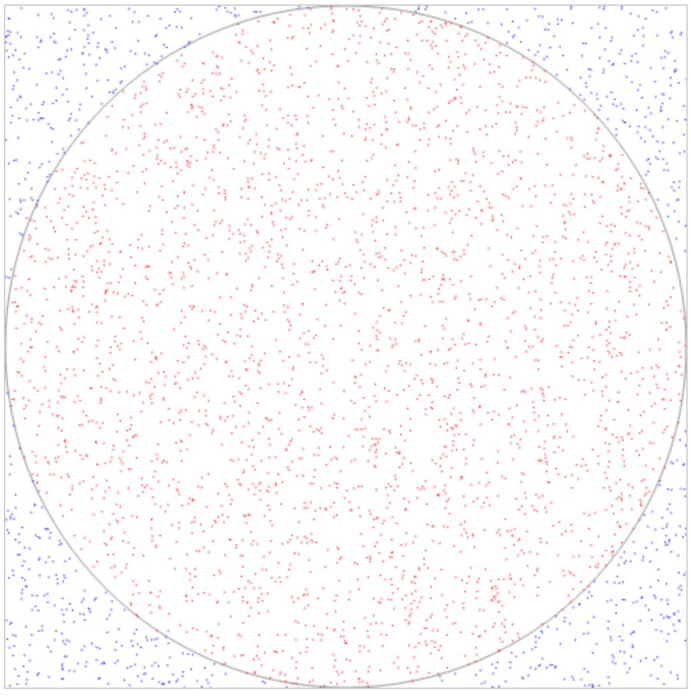
Example: Pi estimation



Total Number of points: 249
Points within circle: 185
Pi estimation: 2.97189

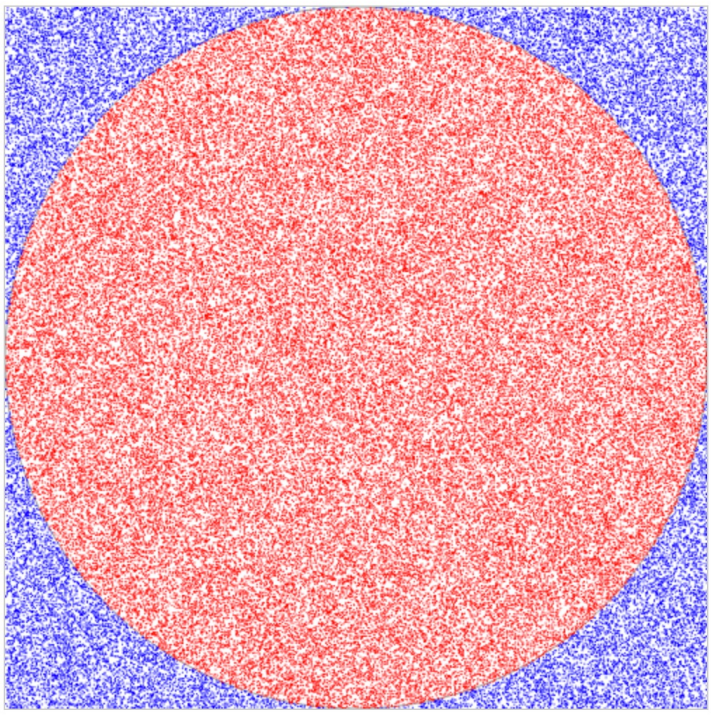
See animation at https://academo.org/demos/estimating-pi-monte-carlo/

Example: Pi estimation



Total Number of points: 4163
Points within circle: 3259
Pi estimation: 3.13140

Example: Pi estimation



Total Number of points: 95770
Points within circle: 75212
Pi estimation: 3.14136

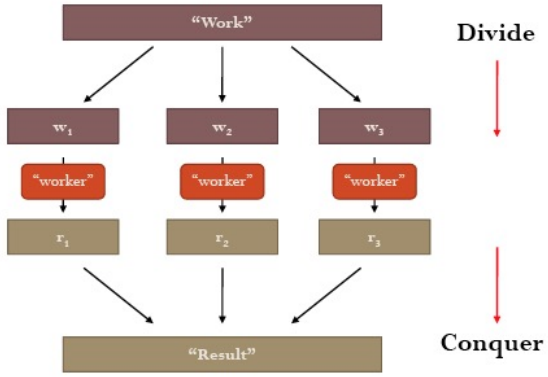
The more points generated, the greater the accuracy of the estimation

Example: Pi estimation

- How to get an accurate and faster estimation of π?
- From sequential to **parallel** computation
- Use master/worker approach
  - Each worker runs the algorithm to generate a set of random points, categorize them, and count how many end up inside the circle
  - The master collects from the workers the total number of generated points and total number of points in the circle. It calculates the ratio of the two numbers and multiplies it by 4 to get a more accurate estimation of π

Key idea behind MapReduce and Spark: **Divide and conquer**

- Feasible approach to tackle large-data problems
  - Partition a large problem into smaller sub-problems
  - Solve independent sub-problems in parallel
  - Combine intermediate results from each individual worker



Implementation details of divide and conquer are complex

Divide and conquer: how?

- Decompose the original problem in smaller, parallel tasks
- Schedule tasks on workers distributed in a cluster, keeping into account:
  - Data locality
  - Resource availability
- Ensure workers get the data they need
- Coordinate synchronization among workers
- Share partial results
- Handle failures

Key idea behind MapReduce and Spark: scale out, not up!

- For data-intensive workloads, a large number of commodity servers is preferred over a small number of high-end servers
  - Cost of super-computers is not linear
  - Data center efficiency
- Processing data is quick, I/O is slow
- Shared nothing is preferable over sharing
  - Shared nothing: each node is completely independent of other nodes in the system, no shared memory or storage
    - ✓ Scalability and fault tolerance
  - Sharing: nodes share a common/global state that must be managed
    - ✗ Requires synchronization, deadlocks can occur, shared resources can become bottlenecks (e.g., bandwidth to access stored data)