NESUS

cost IC1305

Network for Sustainable Ultrascale Computing

Models and systems for scalable Big Data analysis

Paolo Trunfio and Fabrizio Marozzo

University of Calabria

Italy

Before to start

- Download VM Workstation Player www.vmware.com/products/workstation-player/workstation-player-evaluation.html
- Copy the folder **Cloudera-Nesus-Handson**
- Run cloudera-quickstart-vm-5.12.0-0-vmware.**vmx**
- Run **Eclipse**

Outline

- ***Overview* on Big Data analysis and systems**
  - Main requirements
  - Classification criteria
  - Programming models
  - Representative systems
- ***Case study:* Trajectory mining from social media**
  - Motivations
  - Methodology
  - Applications
- ***Hands-on:* Trajectory mining using Hadoop**
  - Main steps
  - Configuration
  - Executions

Overview on Big Data analysis and systems

Big Data analysis and systems

- **Big Data analysis** refers to advanced and efficient data analysis techniques applied to large amount of data.
- Research work and results in the area of Big Data analysis are continuously rising, and more and more new and **efficient architectures**, **programming models**, **systems**, and **data mining algorithms** are proposed.
- Taking into account the **most popular programming models** for **Big Data analysis** (*MapReduce*, *Directed Acyclic Graphs*, *Message Passing*, *Bulk Synchronous Parallel*, *Workflows* and *SQL-like*), we discuss the features of the **main systems implementing them**.

Requirements of Big Data systems (1/2)

- ***Efficient data management and exchange***. Big Data sets often gather data from several heterogeneous sources. Programming systems should support efficient protocols for *data transfers* and *communication*, enable *local computation* of data sources, and provide *fusion mechanisms* to compose the results produced in distributed nodes.

- ***Interoperability***. A main issue in large-scale applications that involve distributed data and computing nodes. Programming systems should support interoperability by allowing the use of different data formats and tools. The National Institute of Standards and Technology (NIST) released the Big Data interoperability framework, a collection of documents aiming at defining some standards for Big Data.

Requirements of Big Data systems (2/2)

- ***Efficient parallel computation***. An effective approach for analyzing large volumes of data and obtaining results in a reasonable time is the exploitation of the inherent parallelism of most data analysis and mining algorithms. Thus, programming systems for Big Data analysis should allow *parallel data processing* and provide a way to easy *monitor* and *tune* the degree of parallelism.
- ***Scalability***. With the exponential increases in the volume of data to be processed, programming systems for Big Data analysis should accommodate *rapid changes* in the growth of data volume and traffic, by exploiting the increment of computational or storage resources efficiently.

Classification criteria (1/5)

- Big Data systems can be compared using four classification criteria:
  - **Level of abstraction**
  - **Type of parallelism**
  - **Infrastructure scale**
  - **Classes of applications**
- These criteria can help developers to select the best solution according to their skills, hardware availability, productivity and application needs.

Classification criteria (2/5)

- **Level of abstraction** of a system refers its programming capabilities to hide the low-level details of a solution:
  - ***Low***, when a programmer can exploit low-level APIs, mechanisms and instructions that are powerful but not trivial to use. A greater development effort is required compared to systems providing a higher level of abstraction, but code efficiency is higher because it can be fully tuned. It also requires a low-level understanding of the system, including working with files on distributed environments.
  - ***Medium***, when a programmer defines an application as a script or a visual representation of the program code, hiding the low-level details that are not fundamental for application design. It requires a medium development effort and code tuning capabilities.
  - ***High***, when developers can build applications using high-level interfaces, such as visual IDEs or abstract models with high-level constructs not related to the running architecture. Program development effort is low as well as code efficiency at run time because executable generation is harder and code mapping is not direct.

Classification criteria (3/5)

- **Type of parallelism** describes the way in which a programming model or system expresses parallel operations and how its runtime supports the execution of concurrent operations on multiple nodes or processors:
  - ***Data parallelism*** is achieved when the same code is executed in parallel on different data elements. Data parallelism is also known as SIMD (Single Instruction Multiple Data).
  - ***Task parallelism*** is achieved when different tasks that compose applications run in parallel. The presence of data dependencies can limit the benefits of this kind of parallelism.
  - ***Pipeline parallelism*** is obtained when data is processed in parallel at different stages, so that the (partial) output of a task is passed to the next task to be processed.

Classification criteria (4/5)

- **Infrastructure scale** refers to the capability of a programming system to efficiently execute applications on a infrastructure of a given size (i.e., number of computing nodes):
  - ***Small*** refers to a small enterprise cluster or Cloud platforms with up to hundreds of computational nodes.
  - ***Medium*** identifies a medium enterprise cluster consisting of up to thousands nodes.
  - ***Large*** refers to large HPC environments or high-level Cloud services with up to ten thousands of nodes.

Classification criteria (5/5)

- **Classes of applications** refer to the main purposes (or application fields) a programming system was designed for:
  - General purpose
  - Real-time stream processing
  - Predictive analytics and machine learning
  - Graph processing
  - Scientific data analytics
  - Visual and script-based analytics
  - Data querying and analysis

Programming models and systems

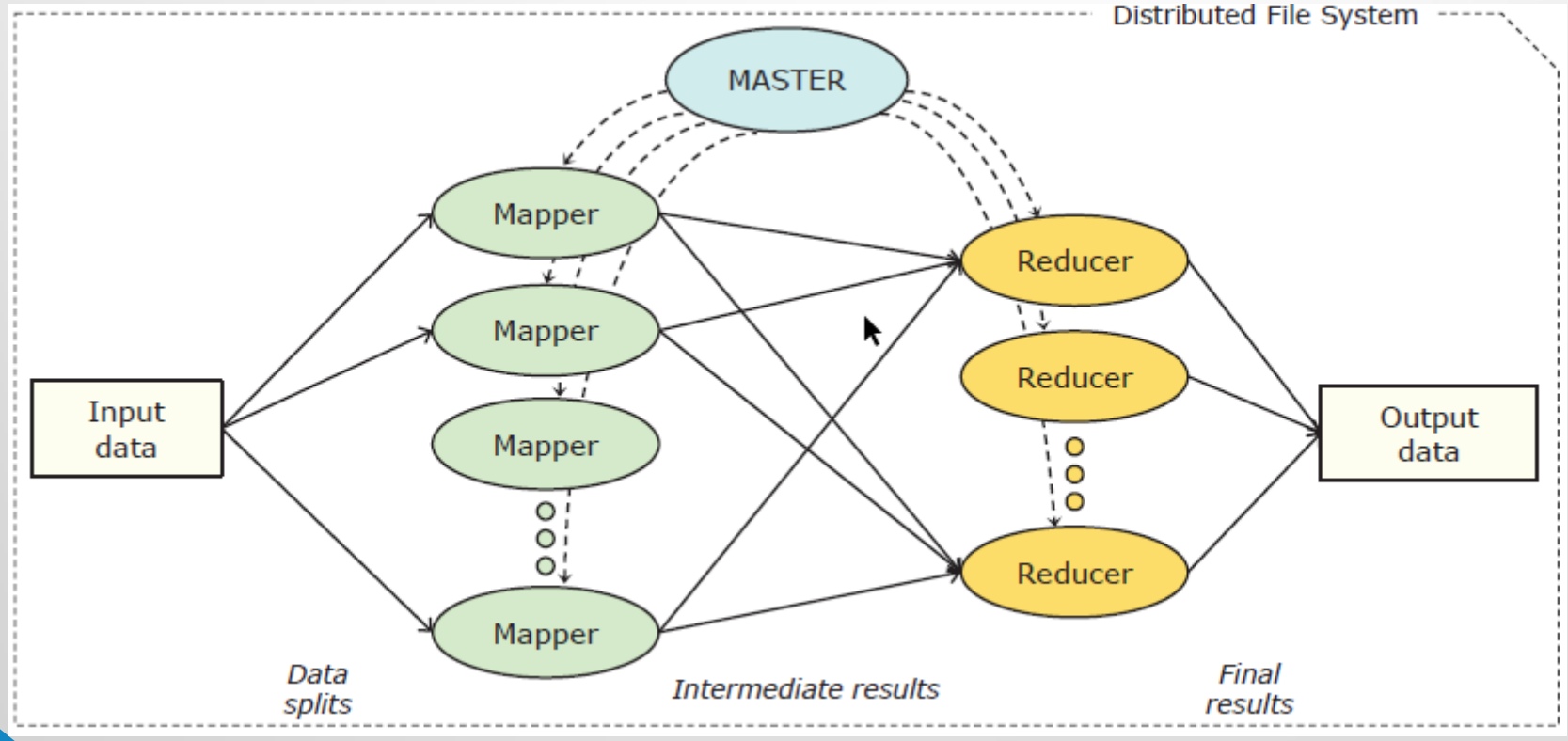
- **MapReduce**
  - *Apache Hadoop*
- **Directed Acyclic Graphs (DAGs)**
  - *Apache Spark*, Apache Storm, Apache Flink, Azure Machine Learning
- **Message Passing**
  - *MPI*
- **Bulk Synchronous Parallel (BSP)**
  - *Apache Giraph*, Apache Hama
- **Workflows**
  - Swift, COMPSs, DMCF
- **SQL-like**
  - Apache Pig, Apache Hive

MapReduce (1/2)

- MapReduce is a programming model developed by Google in 2004 for large-scale data processing.
- The model is inspired by the *map* and *reduce* functions commonly used in functional programming.
- The *map* function processes a (key, value) pair and returns a list of intermediate (key, value) pairs:
  - (k1,v1) -> list(k2,v2)
- The *reduce* function merges all intermediate values having the same intermediate key:
  - k2, list(v2) -> list(v3)

MapReduce (2/2)

- The MapReduce process can be described as follows:

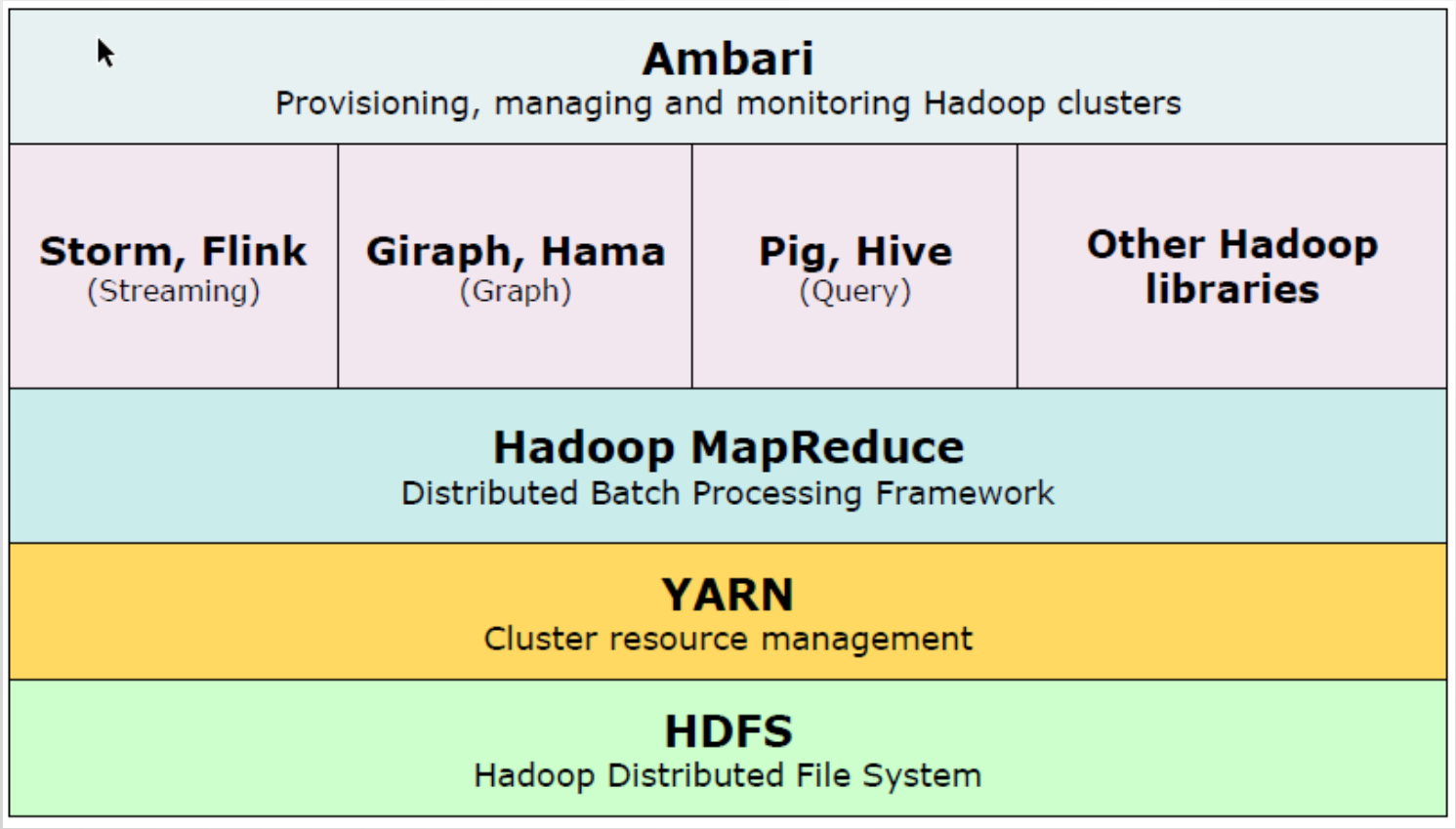


MapReduce: Apache Hadoop (1/2)

- Hadoop is the most used open source MapReduce implementation. It relieves developers from having to deal with classical distributed computing issues, such as load balancing, fault tolerance, data locality, and network bandwidth saving.
- The Hadoop project is not only about the MapReduce programming model (Hadoop MapReduce module), as it includes other modules such as:
  - ***Hadoop Distributed File System (HDFS)****:* a distributed file system providing fault tolerance with automatic recovery, portability across heterogeneous commodity hardware and operating systems, high-throughput access and data reliability.
  - ***Hadoop YARN****:* a framework for cluster resource management and job scheduling.
  - ***Hadoop Common****:* common utilities that support the other Hadoop modules.

MapReduce: Apache Hadoop (2/2)

- Hadoop software stack:



DAGs (1/2)

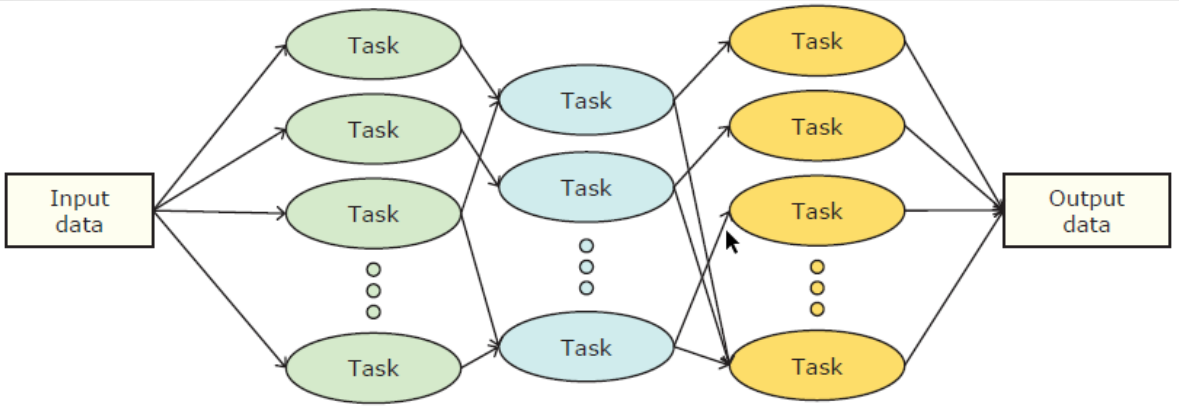
- Directed Acyclic Graph (DAG) is an effective paradigm to model complex data analysis processes, such as data mining applications, which can be efficiently executed on a distributed computing systems such as a Cloud platform.

- A DAG consists of a finite set of edges and vertices, with each edge directed from one vertex to another. DAGs are very close to workflows, but they do not include cycles (i.e., circular dependencies among vertices).

- DAGs can easily model many different types of applications, where the input, output, and tasks of an application depend on other tasks.

DAGs (2/2)

- The tasks dependencies in a DAG can be defined either:
  - *Explicitly*, if the programmer specifies them through explicit instructions (e.g., T_2 depends on T_1)
  - *Implicitly*, if the system analyzes the input/output of tasks to understand dependencies among them (e.g., T_2 reads input O_1, which is an output of T_1)
- DAG tasks can be composed following a number of different patterns (e.g., sequences, parallel constructs)

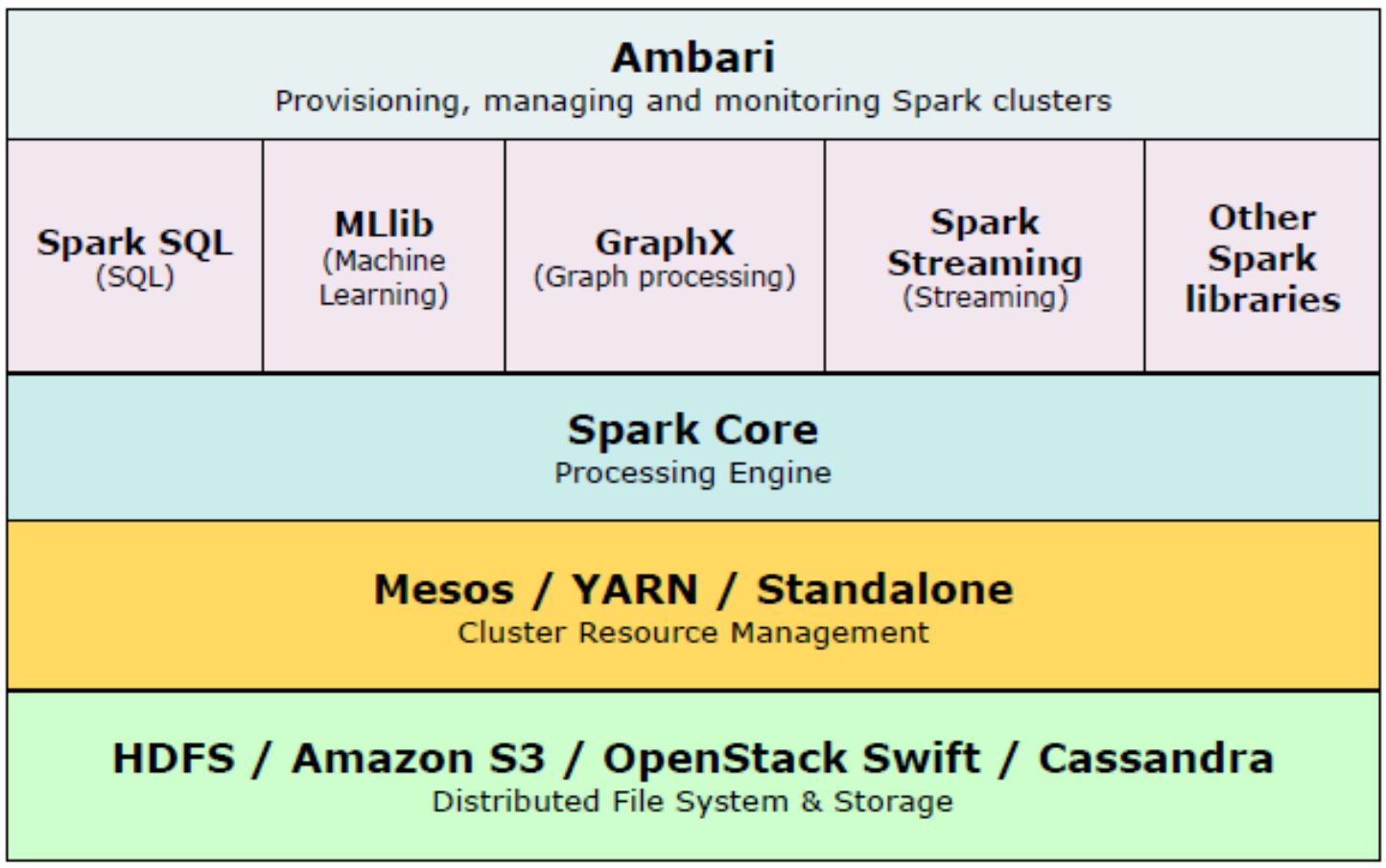


DAGs: Apache Spark (1/2)

- Spark is another top project of the Apache Software Foundation.
- Differently from Hadoop, in which intermediate data are always stored in distributed file systems, Spark stores data in RAM memory and queries it repeatedly so as to obtain better performance for some classes of applications compared to Hadoop (e.g., iterative machine learning algorithms).
- A Spark application in defined as a set of independent stages running on a pool of worker nodes. A stage is a set of tasks executing the same code on different partitions of input data.

DAGs: Apache Spark (2/2)

- Spark software stack:



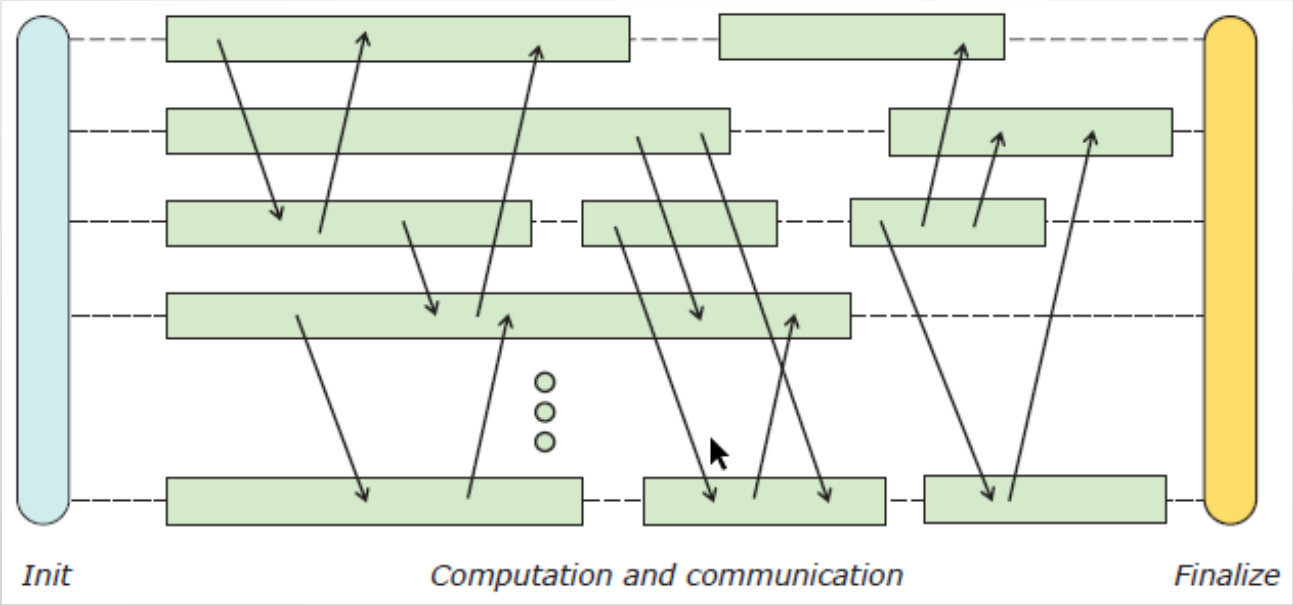
Message passing (1/2)

- The message passing model is a well-known paradigm that provides basic mechanisms for process-to-process communication in distributed computing systems where each processing element has its own private memory.

- In message passing, the sending process composes the message containing the data to be shared with the receiving process(es), including a header specifying to which processing element and process the data is to be routed, and sends it to the network.

Message passing (2/2)

- The primitives of the message passing model are basically two:
  - *Send*(destination, message): a process sends a *message* to another process identified as *destination*
  - *Receive*(source, message): a process receives a *message* from another process identified as *source*

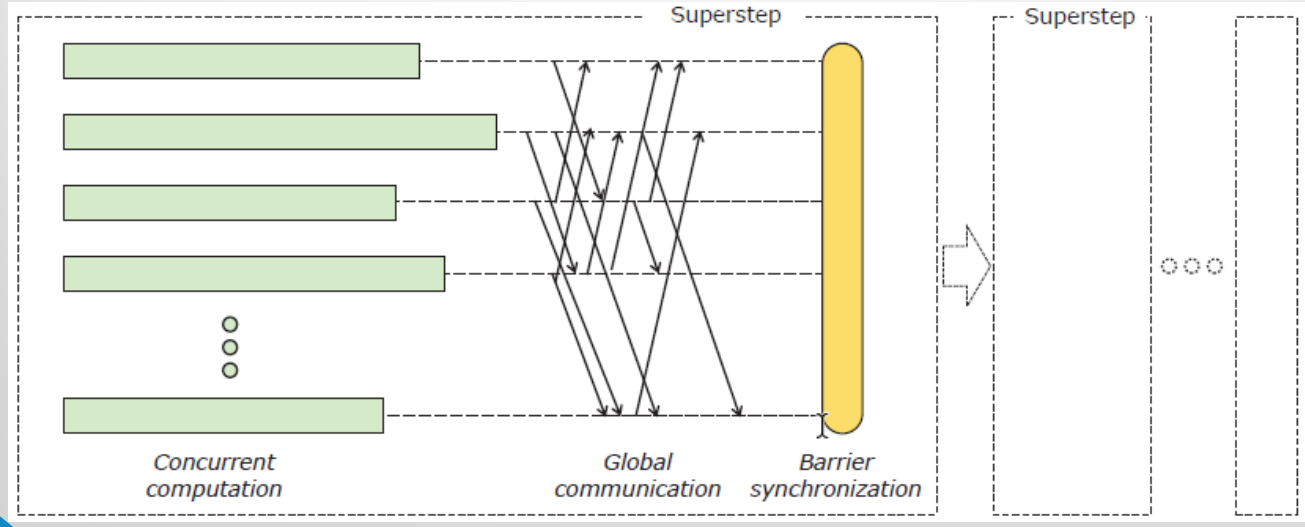


Message passing: MPI

- MPI is the de-facto standard message-passing interface for parallel applications.
- An MPI parallel program is composed of a set of processes running on different processors that use MPI functions for message passing.
- A single MPI process can be executed on each processor of a parallel computer and, according the SPMD (Single Program Multiple Data) model, all the MPI processes that compose a parallel program execute the same code on different data elements.

BSP

- Bulk Synchronous Parallel (BSP) is a parallel computation model in which computation is divided into a sequence of supersteps, each one can perform the following operations:
  - *Concurrent computation*: each processor performs computation using local data asynchronously;
  - *Global communication*: the processes exchange data among them according to requests made during the local computation;
  - *Barrier synchronization*: when a process reaches the barrier, it expects all other processes have reached the same barrier.



BSP: Apache Giraph

- Giraph is an iterative graph processing system for developing highly-scalable applications. It follows the BSP programming model and is implemented using Hadoop as resource manager.
- Specifically, Giraph runs supersteps as a set of computations executed by map-only jobs. Each vertex executes a mapper task by performing the following operations:
  1. receives messages sent to the vertex in the previous superstep;
  2. performs computation using the vertex data and received messages;
  3. sends messages to other connected vertices.
- After every vertex computation, synchronization is performed and Giraph prepares for the next superstep. The execution halts when there are no more messages to process and all work is done.

Workflows (1/2)

- A workflow is a well defined, and possibly repeatable, pattern or systematic organization of activities designed to achieve a certain transformation of data.
- Workflows provide a declarative way of specifying the high-level logic of different kinds of applications, hiding the low-level details that are not fundamental for application design.
- A workflow is programmed as a graph, which consists of a finite set of edges and vertices, with each edge directed from one vertex to another.

Workflows (2/2)

- For example, a data analysis workflow can be designed as a sequence of pre-processing, analysis, post-processing, and interpretation tasks.
- Differently from DAGs, workflows permit to define applications with cycles, which are circular dependencies among vertices.
- Workflow tasks can be composed together following a number of different patterns (e.g., loops, parallel constructs).

Workflows: Swift

- Swift is a parallel scripting language that runs workflows across several distributed systems, like clusters, Clouds, grids, and supercomputers.
- It is based on a C-like syntax and uses an implicit data-driven task parallelism. In fact, it looks like a sequential language, but all variables are futures, thus the execution is based on data availability.
- When the input data is ready, functions are executed in parallel. Its runtime comprises a set of services that implement the parallel execution of Swift scripts exploiting the maximal concurrency permitted by both data dependencies (within a script) and external resource availability.

SQL-like (1/2)

- With the exponential growth of data to be stored in distributed network scenarios, relational databases highlight scalability limitations that significantly reduce the efficiency of querying and analysis.
- Relational databases are not able to scale horizontally over many machines, which makes challenging storing and managing the huge amounts of data produced everyday by many applications.
- The NoSQL database approach became popular in the last years as an alternative or as a complement to relational databases, in order to ensure horizontal scalability of simple read/write database operations distributed over many servers.

SQL-like (2/2)

- NoSQL databases addresses several issues about storing and managing Big Data, but in many cases they are not suitable for analyzing data. For this reason, much effort has been spent to develop MapReduce solutions for querying and analyzing data in a more productively manner.
- Although Hadoop is able to address scalability issues and reduce querying times, it is not easy to be used by low-skilled people since it requires to write a complete MapReduce applications also for simple tasks (e.g., sum or average calculation, row selections or counts), with a considerable waste of time (and money) for companies.
- To cope with this lack, some systems have been developed for improving the query capabilities of Hadoop and easing the development of simple data analysis applications using an SQL-like language.

SQL-like: Apache Pig

- Pig is an Apache open source project for executing data flow applications on top of Hadoop.
- It provides a high-level scripting language, called Pig Latin, that allows users to define a script containing a sequence of operations.
- Each operation is defined in a SQL-like syntax for describing how data must be manipulated and processed.
- Pig is commonly used for developing Extract, Transform, Load applications, which are able to gather, transform and load data coming from several sources (e.g., streams, HDFS, files).
- Each Pig script is translated into a series of MapReduce jobs. The Pig interpreter tries to optimize the execution of these jobs on a Hadoop cluster.

Comparative table

	System	Level of Abstraction	Type of Parallelism	Infrastructure Scale	Classes of Applications
MapReduce	Apache Hadoop	Low	Data	Large	General purpose
DAG	Apache Spark	Low	Data/Task	Large	General purpose
	Apache Storm	Medium	Data/Task/Pipeline	Large	Real-time stream processing
	Apache Flink	Medium	Data/Task/Pipeline	Large	Real-time stream processing
	Azure ML	High	Task	Small	Predictive analytics and machine learning
MP	MPI	Low	Data	Medium	General purpose
BSP	Apache Giraph	Low	Data	Small	Graph processing
	Apache Hama	Low	Data	Small	Graph processing
Workflow	Swift	Medium	Task/Pipeline	Medium	Scientific data analytics
	COMPSs	Medium	Task	Small	Scientific data analytics
	DMCF	High	Task/Pipeline	Small	Visual and script-based analytics
SQL-like	Apache Pig	Medium	Data/Task	Large	Data querying and analysis
	Apache Hive	High	Data	Large	Data querying and reporting

Case study: Trajectory mining from social media

Motivations (1/4)

- Huge volumes of digital data are being generated by social media users
- For instance, *every minute**:
  - **Twitter** users send **456,000 tweets**
  - **Instagram** users post **46,740 photos**

- *Social media analysis* is a fast growing research area aimed at extracting useful information from this data.
- Analysis of collective sentiments, understanding the behavior of groups of people, studying the dynamics of public opinion….

*** https://www.domo.com/learn/data-never-sleeps-5**

Motivations (2/4)

- Social media posts are often tagged with **geographical coordinates** or other information (e.g., text, location fields) that allows identifying users' positions.
- Therefore, users moving through a set of locations produce a huge amount of geo-referenced data that embed extensive knowledge about **mobility behaviors**.
- In the latest years, there has been a growing interest in the extraction of trajectories from geotagged social data using **trajectory mining techniques**.

Motivations (3/4)

- We proposed a general methodology for discovering mobility patterns of users attending large-scale public events.
- Main goals:
  - **Discovery of most visited places and most attended events**. We analyze data to discover the places that have been most visited, and the events that have been most attended by visitors during the observed period.
  - **Discovery of most frequent sets of visited places and most frequent sets of attended events**. We extract the sets of places that are most frequently visited together, and the events that have been most attended by visitors.
  - **Discovery of most frequent mobility patterns among places and most frequent sequences of attended events**. We analyze data to discover mobility behaviors and extract patterns about the attended events.
  - **Discovery of the origin and destination of visitors**. We study which countries visitors came from or moved to after the events. This information can give insights about the touristic impact on the local territory.

Motivations (4/4)

- The methodology have been applied to two large-scale events occurred in 2014 and 2015:
  - **FIFA World Cup 2014**. Analysis of geotagged tweets sent by *Twitter* users who attended the World Cup matches to discover the mobility of fans during the competition.
  - **EXPO 2015**. Analysis of the geotagged posts published by the *Instagram* users who visited EXPO, from one month before to one month after their visit to EXPO.

Some definitions (1/3)

- $P = \{p_1, p_2, \ldots\}$ is a set of *Places-of-Interest* (PoIs), where p_i is a specific area that is of interest to a community during a given time period.
- A PoI may be a business location (e.g., shopping mall), a tourist attraction (theater, museum, park, bridge) or some other location (square, pavilion, stadium…) relevant to an event.
- A PoI is not limited to a single geographical point or a street, but it refers to an area (e.g., a polygon over a map).
- Thus, PoIs can also be referred to as *Regions-of-Interest* (RoIs), where a region represents the boundaries of a PoI's area.

Some definitions (2/3)

- $E = \{e_1, e_2, \ldots\}$ is a set of *events* involving a large presence of people, where $e_i = \langle p_i, [t_i^{begin}, t_i^{end}] \rangle$ takes place in p_i during a time interval $[t_i^{begin}, t_i^{end}]$.
- An event may be a match played in a stadium (as part of a sport competition), a showcase hosted in a pavilion (as part of a trade exposition), or a concert held in a theater or in a square (as part of an artist's tour).
- An event can have some additional descriptive properties, related to the specific domain.

Some definitions (3/3)

- $G = \{g_1, g_2, \ldots\}$ is a set of *geotagged items*, where g_i is a social media content (e.g. tweet, post, photo, video) posted by a user during an event e_i from the place p_i where e_i was held.
- A geotagged item g_i includes:
  - *userID*, containing the identifier of the user who posted g_i
  - *coordinates* (latit. and long.) of the place where g_i was sent from
  - *timestamp*, indicating when (date and time) g_i was posted
  - *text*, containing a textual description of g_i
  - *tags*, containing the tags associated to g_i
- $U = \{u_1, u_2, \ldots\}$ is a set of users, where u_i has published at least one geotagged item in G.

A 7-step methodology

1. Definition of the set of events ***E***.
2. Definition of the places-of-interests ***P*** where the events in ***E*** are held.
3. Collection and pre-processing of the geotagged items ***G*** related to the events in ***E***.
4. Identification of the users ***U*** who published at least one of the geotagged items in ***G***.
5. Creation of the input dataset ***D***.
6. Data analysis and trajectory mining on ***D***.
7. Results visualization.

A running example

- A large-scale cultural event in the center of Rome composed by free or paid events like concerts, guided tours, outdoor exhibits spread over three days.

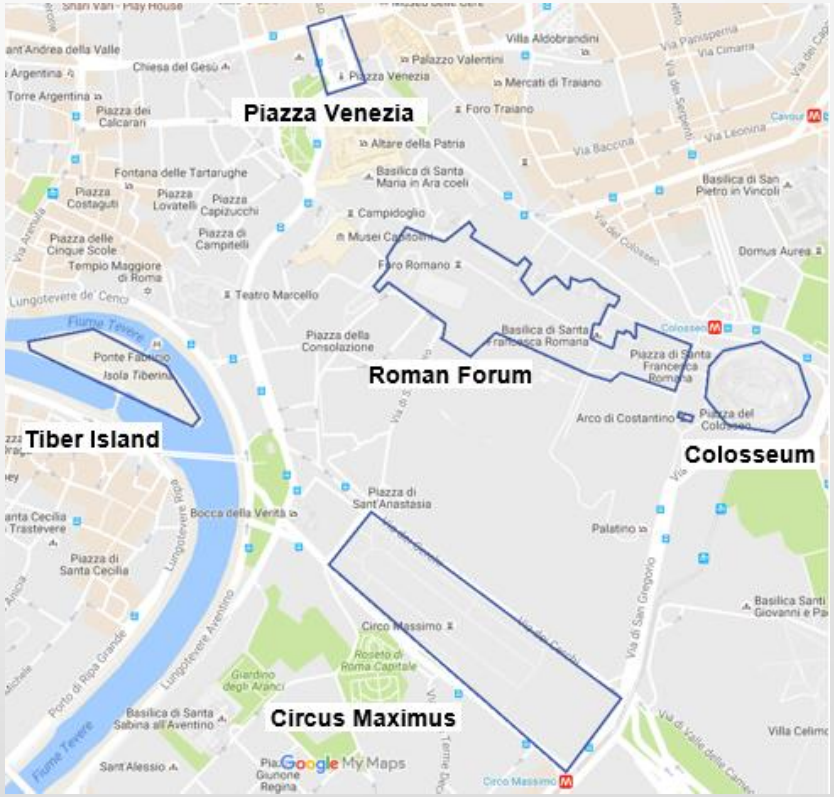
Step 1: Definition of events

- This step is aimed at defining the set of events ***E***.
- Each event is described by the *id* of the place-of-interest (PoI) where it is located, *starting/ending time* of the event, and other optional data (e.g., *free/paid* event, *type* of event, etc.).

$$
\begin{aligned}
e_1 &= \langle Colosseum, (\text{2016-11-01T19:00}, \text{2016-11-01T23:00}), \text{Paid}, \text{Concert} \rangle \\
e_2 &= \langle RomanForum, (\text{2016-11-02T10:00}, \text{2016-11-02T19:00}), \text{Free}, \text{Guided tour} \rangle \\
e_3 &= \langle TiberIsland, (\text{2016-11-02T10:00}, \text{2016-11-02T19:00}), \text{Free}, \text{Guided tour} \rangle \\
e_4 &= \langle PiazzaVenezia, (\text{2016-11-03T09:00}, \text{2016-11-03T18:00}), \text{Free}, \text{Outdoor exhibit} \rangle \\
e_5 &= \langle CircusMaximus, (\text{2016-11-03T19:00}, \text{2016-11-03T23:00}), \text{Paid}, \text{Concert} \rangle
\end{aligned}
$$

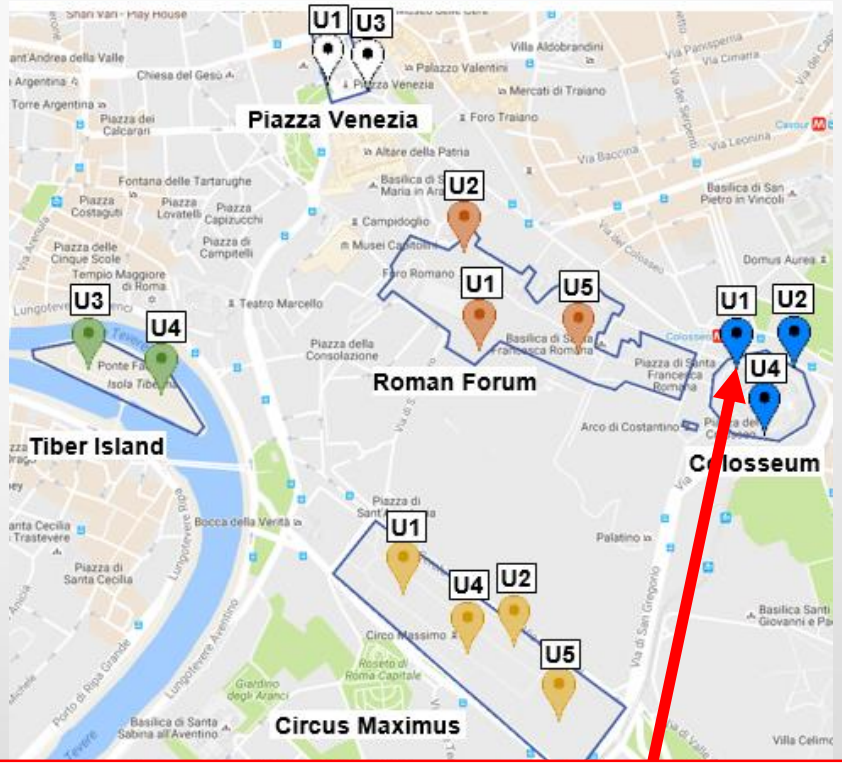
Step 2: Definition of places-of-interest

- This step defines the geographical boundaries of the PoIs in ***P***.
- This can be done:
  - *manually* defining the boundaries of the PoI (e.g., polygons on a map)
  - *automatically*, using external services (e.g., cadastral maps), or web services providing the boundaries of a place given its name (e.g., OpenStreetMap)



Step 3: Collection and pre-processing of geotagged items

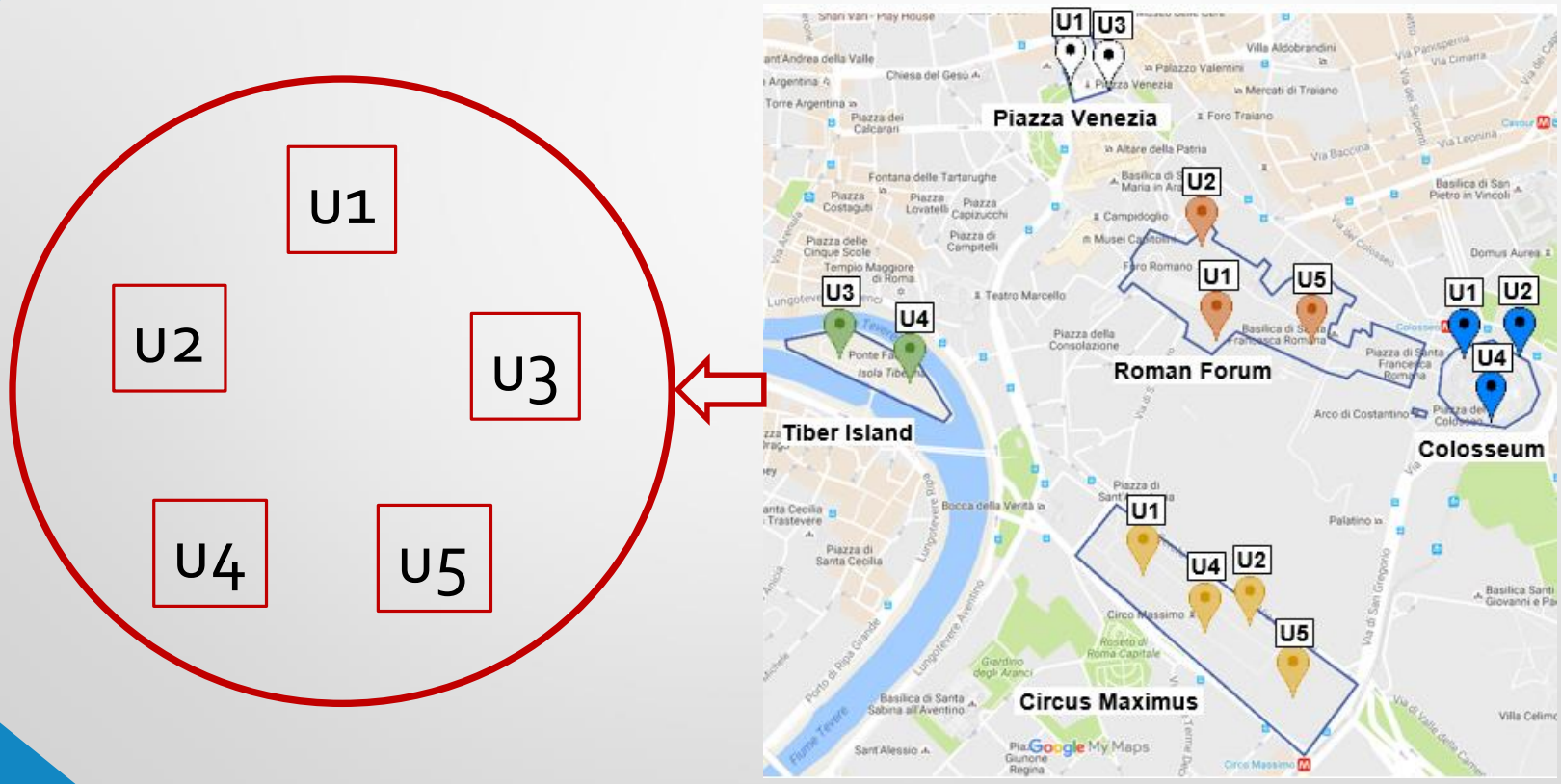
- The goal of this step is to collect the geotagged items ***G*** posted during each event e_i in ***E*** from the place p_i where e_i was held.
- *Data collection* using the APIs provided by most social media platforms.
- *Pre-processing* to clean, select and transform data to make it suitable for analysis.



$$g_1 = \langle \text{"}u1\text{"}, [41.889995, 12.492157], 2016\text{-}11\text{-}01\text{T}20{:}35, \text{"Unforgettable experience at the Coliseum."}, [\#Coliseum, \#Art, \#History] \rangle$$

Step 4: Identification of users

- This is done by extracting the set ***U*** of distinct users who published at least one geotagged item in ***G***.



Step 5: Creation of the input dataset

- This step creates the input datasets

$$D = \{T_1, T_2, \ldots\}$$

where T_i is a tuple

$$< u_i, \{e_{i1}, e_{i2}, \ldots, e_{ik}\}, optFields >$$

in which e_{ij} is the j^{th} event attended by user u_i and *optFields* are optional descriptive fields (e.g., nationality, interests).

$$< u_1, \{e_1, e_2, e_4, e_5\}, Italian, art >$$
$$< u_2, \{e_1, e_2\}, German, music >$$
$$< u_3, \{e_3, e_4, e_5\}, French, art >$$
$$< u_4, \{e_1, e_3, e_5\}, German, music >$$
$$< u_5, \{e_2, e_5\}, Italian, music >$$

Steps 6: Data analysis and trajectory mining (1/3)

- ***D*** can be analyzed through algorithms and techniques for *associative analysis* (associative pattern mining) and *sequential analysis* (sequential pattern mining).
- *Associative analysis* algorithms are used with the goal of discovering the item values that occur together with a high frequency.
- *Example*: Sets of places that are most frequently visited together by users.

Steps 6: Data analysis and trajectory mining (2/3)

- *Sequential analysis* algorithms are intended to discover the sequences of elements that occur most frequently in the data.
- Unlike associative analysis, in sequential analysis the time dimension and the chronological order in which the values appear in the data are fundamental.
- *Example*: Most frequent sequences of attended events (mobility patterns).

Steps 6: Data analysis and trajectory mining (3/3)

- Examples of rules:

$(e_1, e_5), 3$
$(e_2, e_5), 3$
$(e_1, e_2), 2$
$(e_1, e_2, e_5), 2$

3 users attended events e1 and e5

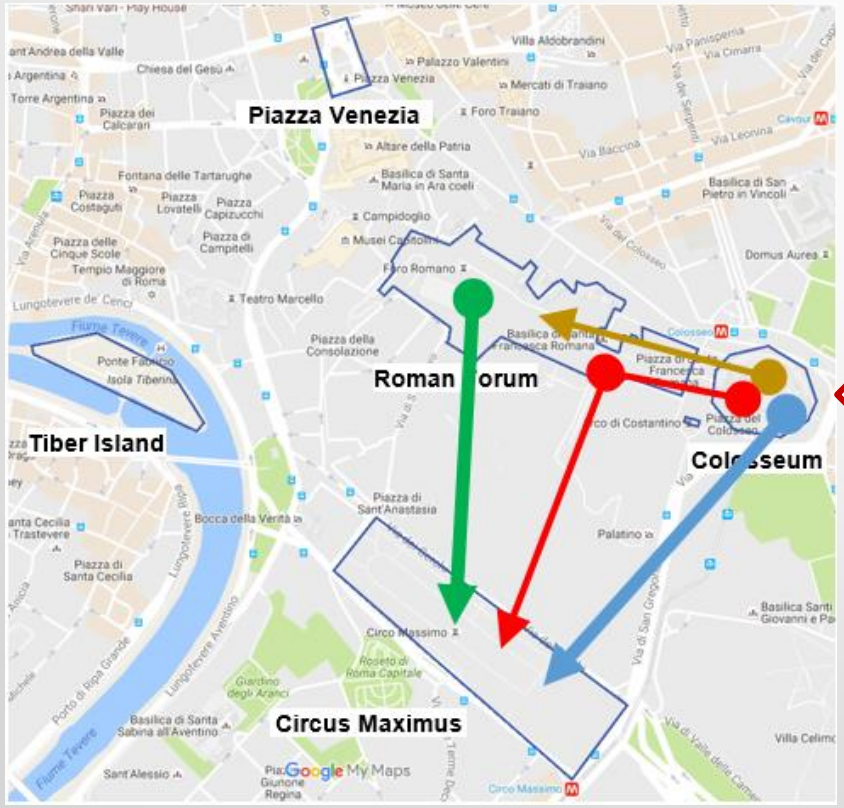
2 users attended events e1, e2 and e5

Steps 7: Results visualization (1/2)

- Info-graphics to presents results in a easy-to-understand way, avoiding complex statistical details that may be hard to understand to the general audience.
- Three main guidelines:
  - Preferring a visual representation of the quantitative information to the numeric one
  - Minimizing the cognitive efforts necessary for decoding the system of signs
  - Structuring the graphic elements into hierarchies

Steps 7: Results visualization (2/2)

- Example of visualization:



Different line sizes used to represent the support of each rule

Example 1: FIFA World Cup 2014

- 20th edition of the quadrennial world championship organized by FIFA.
- Took place in Brazil from 12 June to 13 July 2014.
- 32 national teams, with a total of 64 matches played in 12 stadiums distributed across Brazil.
- More than 5 million people attended soccer matches and related events.

Definition of events

- ***E*** is composed by the 64 matches played during the World Cup:

$$E = \{e_1, e_2, \ldots, e_{64}\}$$

where:

$$e_i = \langle p_i, [t_i^{begin}, t_i^{end}], team1, team2, phase \rangle$$

and:

p_i is the stadium

t_i^{begin} is 3 hours before the begin of the match

t_i^{end} is 3 hours after the end of the match

team1 is the first team

team2 is the second team

phase = 'opening match', 'group stage', 'round of 16' ....

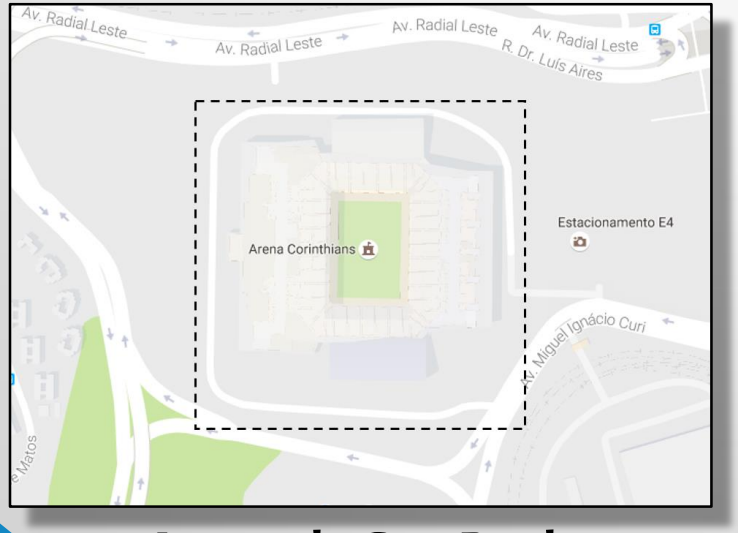
$$e_1 = \langle São Paulo, [2014\text{-}06\text{-}12T14{:}00, 2014\text{-}06\text{-}12T20{:}00], Brasil, Croatia, Opening Match \rangle$$
$$e_2 = \langle Natal, [2014\text{-}06\text{-}13T10{:}00, 2014\text{-}06\text{-}13T16{:}00], Mexico, Cameroon, Group Stage \rangle$$

Definition of places-of-interest

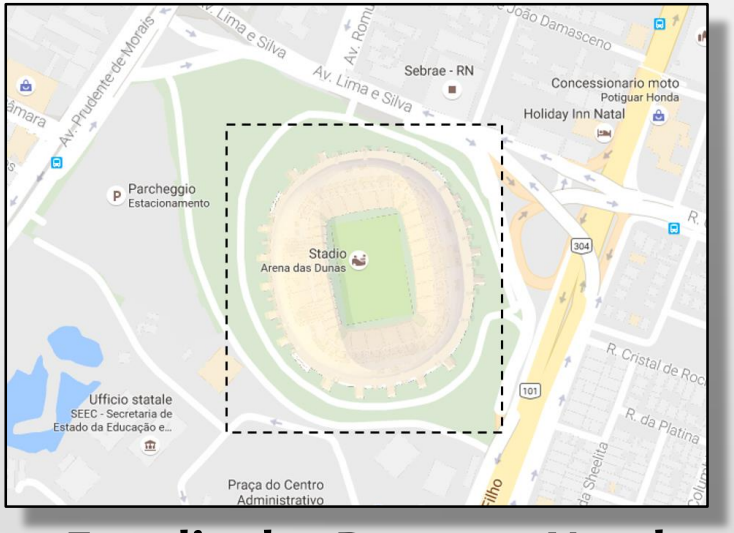
- ***P*** are the 12 stadiums that hosted at least one match during the World Cup :

$$P = \{p_1, p_2, \ldots, p_{12}\}$$

- The RoI of each p_i is the smallest rectangle that fully contains its boundaries.



Arena de Sao Paulo



Estadio das Dunas at Natal

Collection of geotagged items

- We collected 526,000 geotagged tweets posted from coordinates within the above defined RoIs during the matches:

$$G = \{g_1, g_2, \ldots\}$$

where each g_i contains $user_{ID}$, *coordinates*, *timestamp*, *text*, *tags*, and *application* (optional field) used to generate it.

$$g_i = \langle 11223344, [-23.545531, -46.473372], \text{2014-06-12T16:59} \\ \text{“Proud to be here.”}, [\#BRAvsCRO, \#Brasil2014], TwitterForAndroid \rangle$$

Data pre-processing

- Pre-processing on **G** to *clean*, *select* and *transform* data to make it suitable for analysis:
  1. *Cleaned* data by removing all the tweets with unreliable position (e.g., tweets with coordinates automatically set by other applications).
  2. *Selected* only tweets posted by users attending the matches, by removing re-tweets and favorites posted by other users.
  3. *Transformed* data by keeping one tweet per user per match, as we were interested to know only if a user attended a match or not.

Input dataset

- Input dataset ***D*** contains the list of matches attended by a single user:

$$D=\{T_1, T_2, \ldots, T_n\}$$

where

$$T_i=<u_i,\{m_{i1}, m_{i2}, \ldots, m_{ik}\}>$$

and $m_{i1}, m_{i2}, \ldots, m_{ik}$ are the matches attended by a Twitter user u_i.

Trajectory mining

- Trajectory pattern mining to extract the most frequent movements of fans starting from ***D***.
- **Trajectory pattern**: sequence of geographic regions that emerge as frequently visited in a given temporal order.
- The ***support*** of a trajectory pattern ***p*** (# of transactions containing ***p***) is a measure of its reliability.
- In this case study, a ***frequent pattern fp*** with support ***s***:

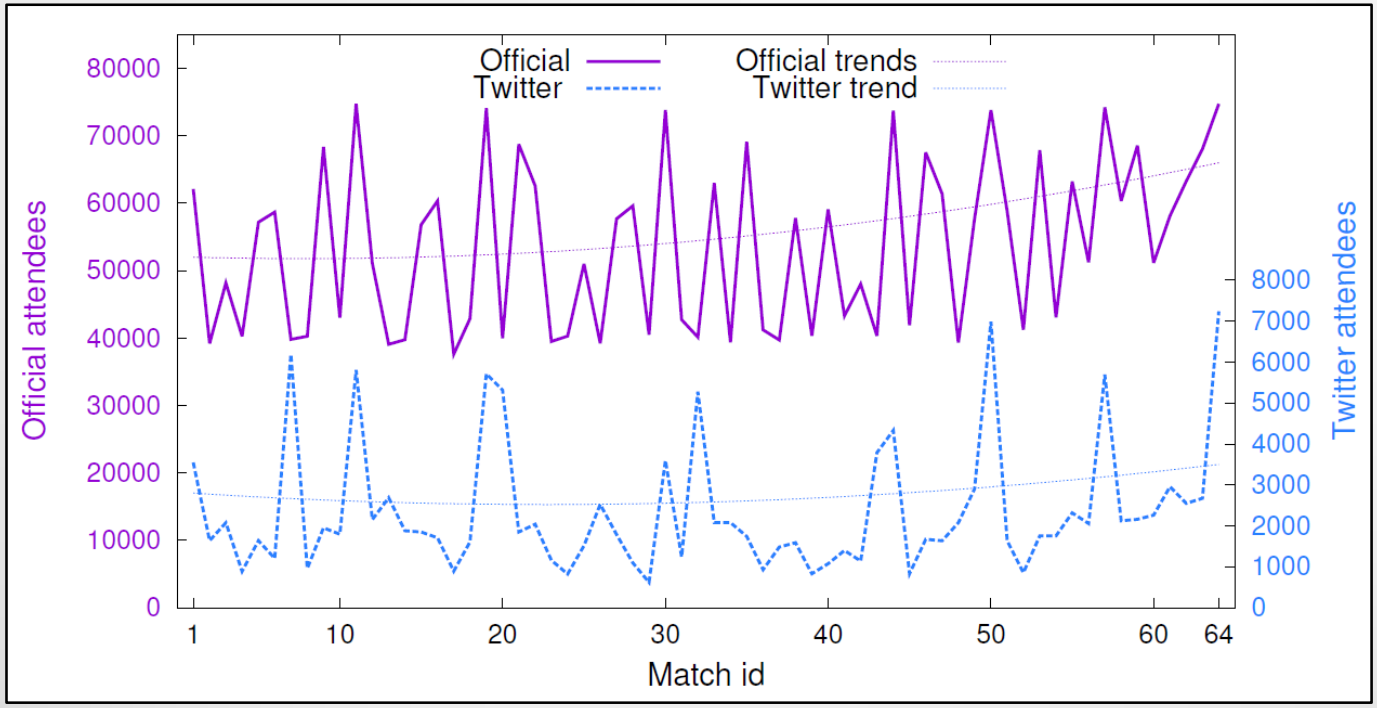
$$fp=<m_i, m_j,.., m_k>(s)$$

is an ordered sequence of matches $m_i, m_j,.., m_k$ where ***s*** is the percentage of transactions in ***D*** containing ***fp***

Results

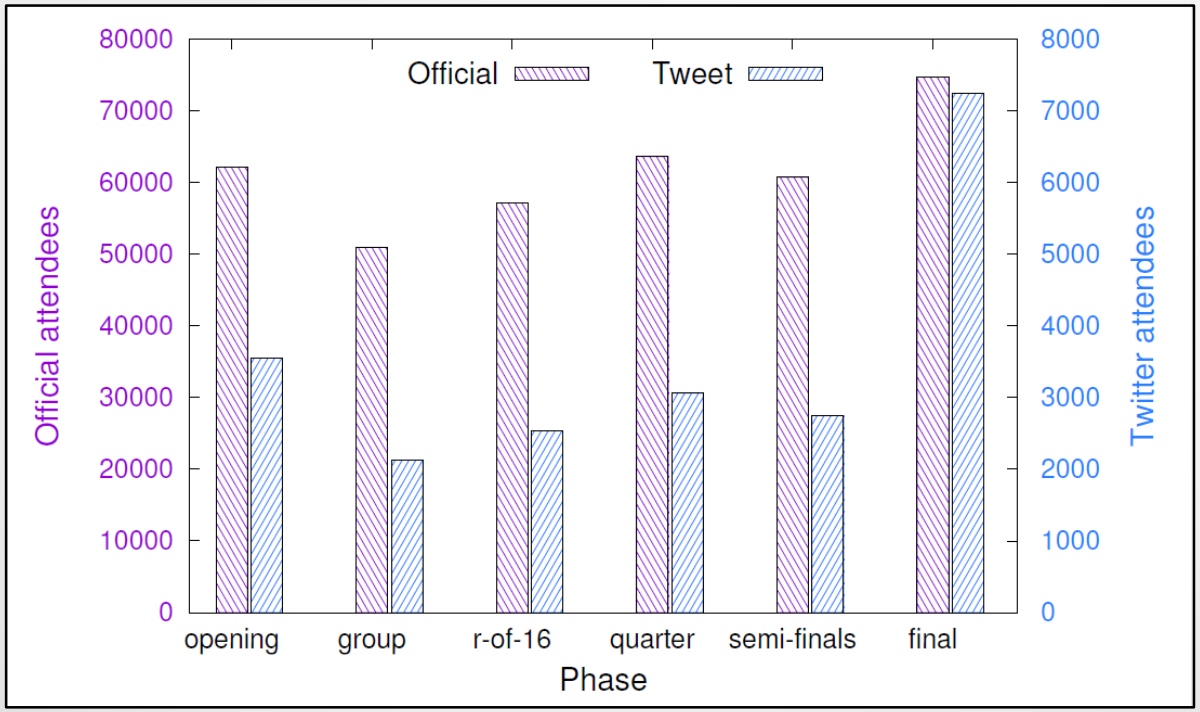
- **Number of people attending the matches** over time
- **Number of matches attended** by fans during the competition
- **Most frequent sequences of matches** attended by fans, either in the same stadium or to follow a given soccer team
- **Most frequent movement patterns** obtained by grouping matches based on the phase in which they were played

Number of people attending the matches (1/2)



- Number of attendees per match, according to both Twitter and official data
- In some cases Twitter data peaks are equivalent to the official attendance ones (e.g. matches n. 11, 19, 44, 50 and 57).

Number of people attending the matches (2/2)



- Average number of attendees per match, grouped by phases, for both Twitter and official data
- Pearson correlation between official attendee numbers and Twitter users equal to 0.9

Number of matches attended

No. of matches	Spectators
1	71.3%
2	16.0%
3	6.0%
4	3.0%
5 or more	3.7%

- 3.7% of the spectators attended **five or more** matches during the whole World Cup.
- Twitter profiles of those who attended several matches, show that many of them were journalists.

Frequent sequences (1/4)

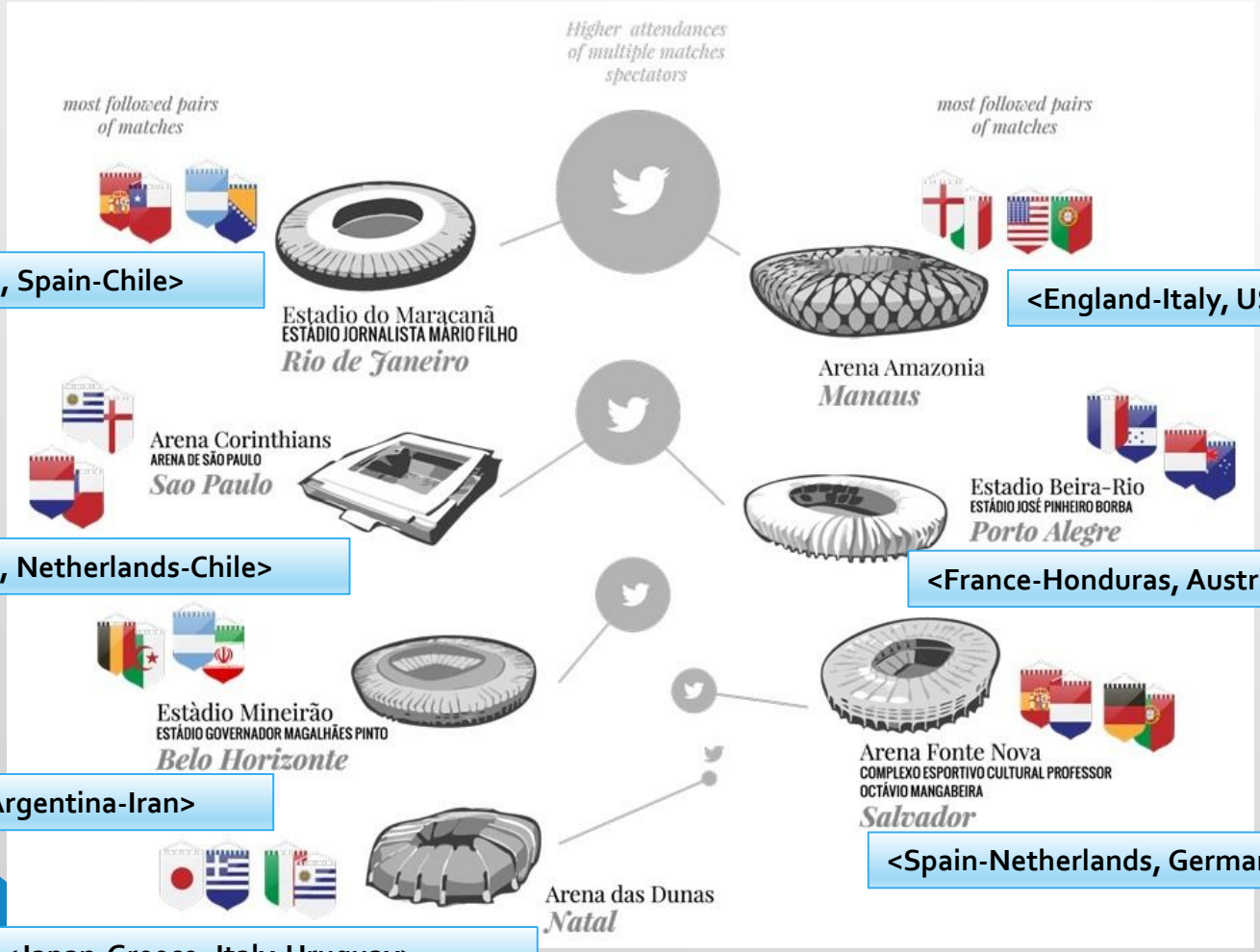
- General classification of the paths followed by fans who attended at least two matches:

No. of matches	Same stadium	Same team
2	62.9%	22.2%
3	48.8%	11.8%
4	41.0%	7.2%
5	37.0%	8.4%
6	33.7%	4.8%

- Results show that **most of those who attended multiple matches did it staying in the same city.**

Frequent sequences (2/4)

- Most frequent **2-match-sets** observed during the **group stage**, from June 12 to June 26, 2014



<Argentina-Bosnia, Spain-Chile>

<England-Italy, USA-Portugal>

<Uruguay-England, Netherlands-Chile>

<France-Honduras, Austria-Netherlands>

<Belgium-Algeria, Argentina-Iran>

<Spain-Netherlands, Germany-Portugal>

<Japan-Greece, Italy-Uruguay>

Frequent sequences (3/4)

- Most frequent paths of fans who attended **two** or **three** matches of the same team during the group stage

Most frequent 2-match sets:

<***Colombia***-Greece, ***Colombia***-Cote d'Ivoire>

<Brazil-***Mexico***, Croatia-***Mexico***>

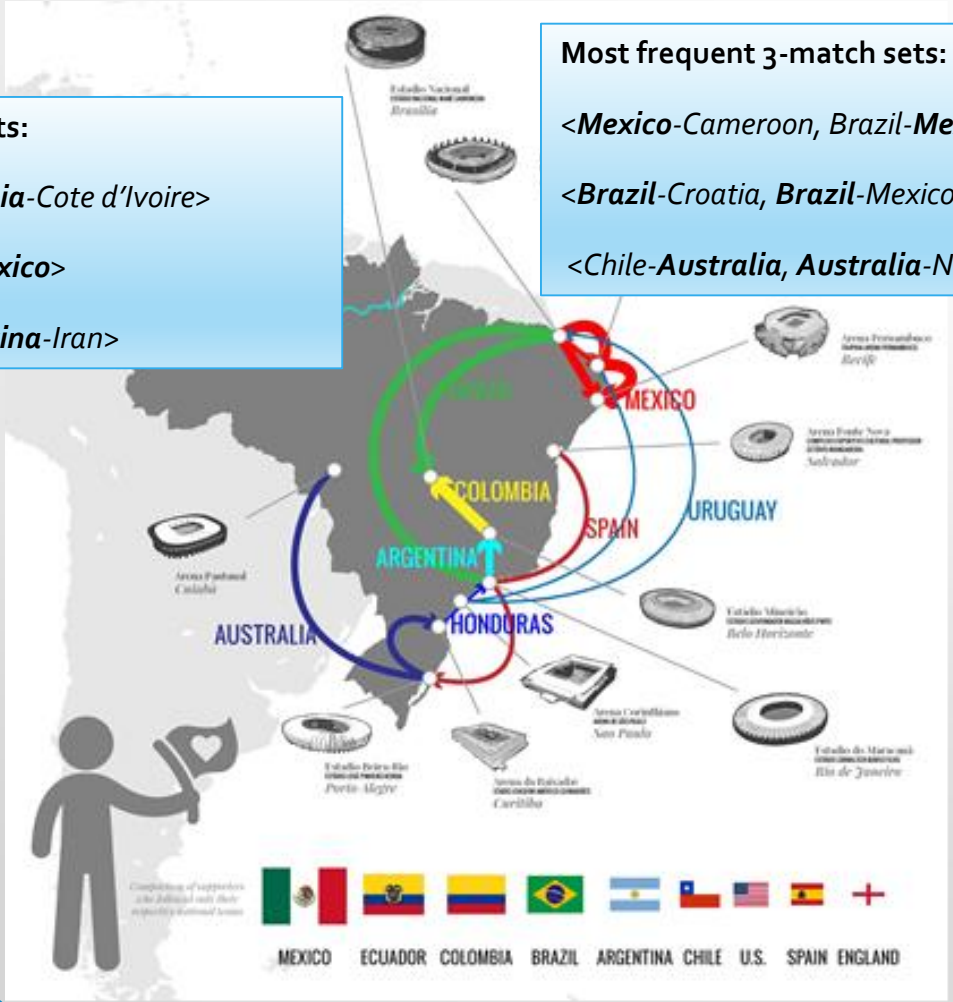
<***Argentina***-Bosnia, ***Argentina***-Iran>

Most frequent 3-match sets:

<***Mexico***-Cameroon, Brazil-***Mexico***, Croatia-***Mexico***>

<***Brazil***-Croatia, ***Brazil***-Mexico, Cameroon-***Brazil***>

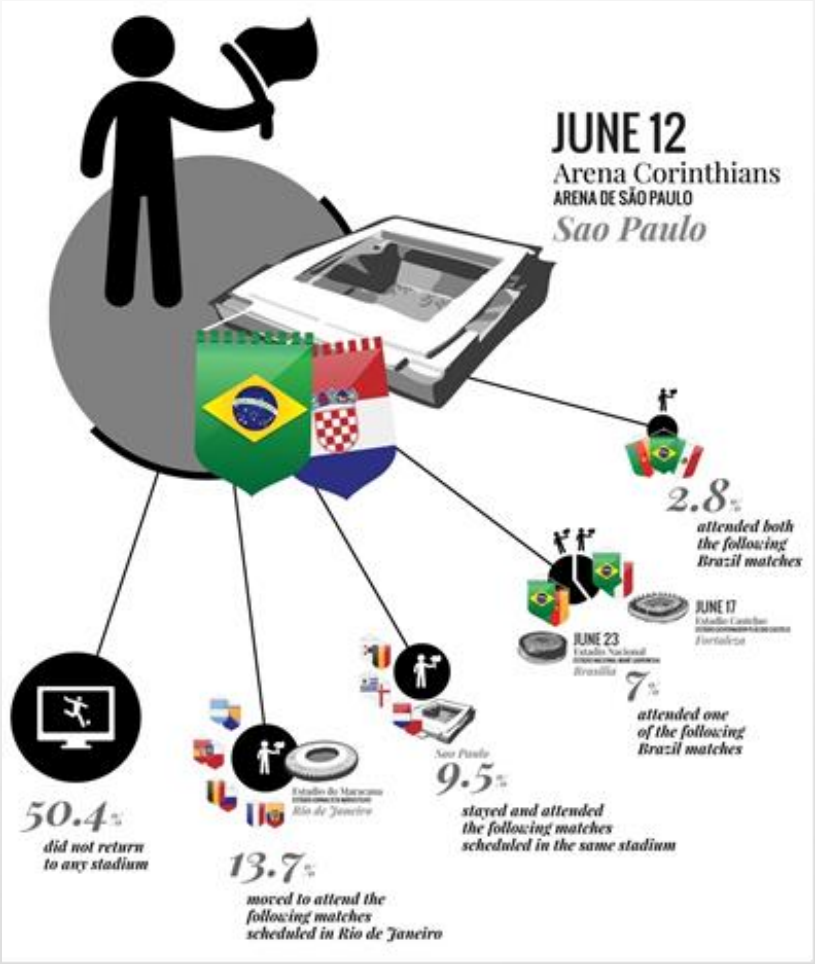
<Chile-***Australia***, ***Australia***-Netherlands, ***Australia***-Spain>



Frequent sequences (4/4)

- Specific analysis on the spectators of the opening match <*Brazil-Croatia*> played on in São Paulo

- At the end of group stage:
  - **50.4%** did not attend other matches
  - **13.7%** moved to Rio de Janeiro to attend other matches
  - **9.5%** attended other matches in the same stadium



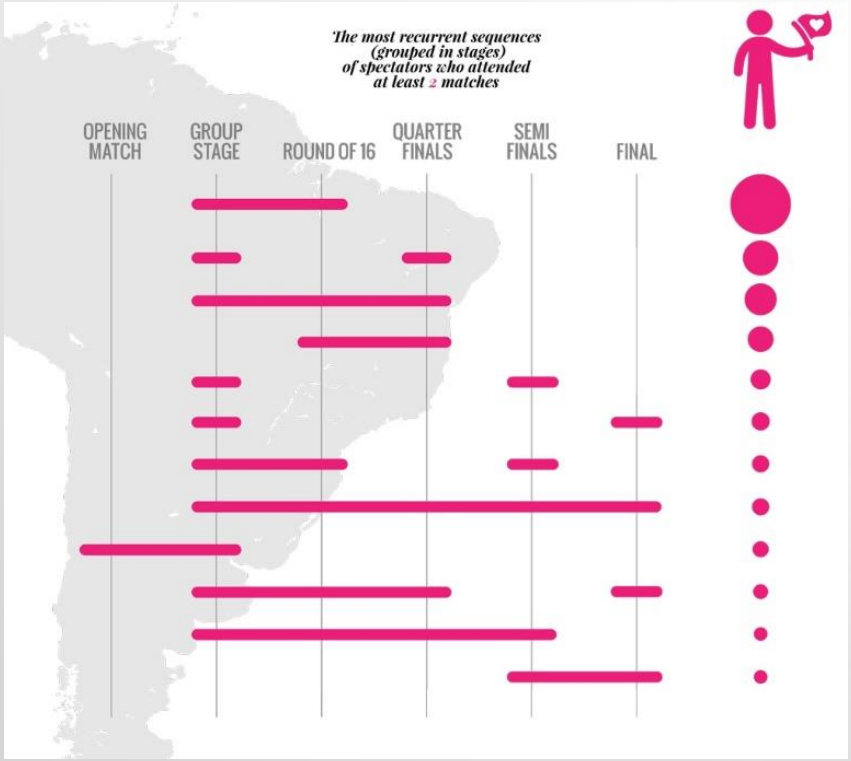
Aggregate analysis (1/2)

- *Goal*: studying the movements of fans during the different phases of the competition

- Matches were grouped into the following phases:
  - *Opening match* (match no. 1)
  - *Group stage* (matches no. 2-48)
  - *Round of 16* (matches no. 49-56)
  - *Quarter finals* (matches no. 57-60)
  - *Semi-finals* (matches no. 61-62)
  - *Final* (match no. 64)

Aggregate analysis (2/2)

- Patterns of movements based on the grouping above, and the relative frequency (support) of these patterns



Most frequent:

Group stage and **Round of 16**

2nd most frequent:

Group stage and **Quarter finals**

3rd most frequent:

Group stage and **Group of 16** and **Quarter finals**

Least frequent:

Semi-final and **Final**

Example 2: EXPO 2015

- Universal Exposition held under the theme "Feeding the Planet, Energy for Life"
- Hosted in Milan, Italy, from May 1st to October 31st, 2015.
- Exhibitors were countries, international organizations, civil society organizations and companies, for a total of 188 exhibition spaces.
- About 22 million people visited EXPO during the six months, making it the world-wide largest event in 2015.

Definition of events

- ***E*** is composed by the showcases (organized by a country or organization/company) exhibited in the pavilions:

$$E=\{e_1, e_2, \ldots, e_{188}\}$$

where:

$$e_i = \langle p_i, [t_i^{begin}, t_i^{end}] \rangle$$

and:

p_i is a pavilion

t_i^{begin} is May 1st, 2015

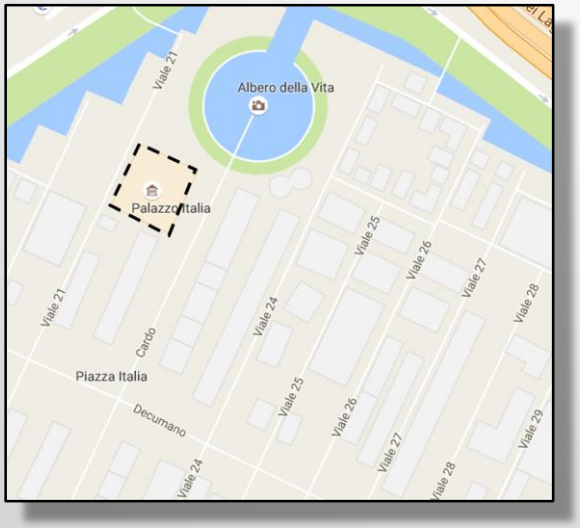
t_i^{end} is October 31st, 2015

Definition of places-of-interest

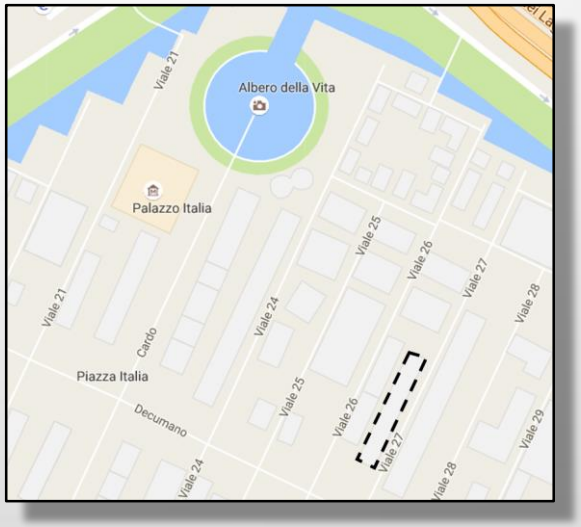
- ***P*** are the 188 pavilions that hosted the EXPO showcases:

$$P = \{p_1, p_2, \ldots, p_{188}\}$$

- For each p_i, the corresponding RoI was identified from the EXPO map



Italy pavilion



USA pavilion

Data collection and pre-processing (1/2)

- We collected geotagged posts published by Instagram users who visited at least one pavilion during the EXPO.
- We also gathered posts that were *pre-related* and *post-related* to the EXPO visits, i.e., posts published by users from 1 month before to 1 month after their visit to EXPO.
- Overall numbers:
  - **238,000** Instagram users who visited EXPO.
  - **570,000** posts published during the visits to EXPO.
  - **2.63 million** posts published by users 1 month before to 1 month after their visit to EXPO.

Data collection and pre-processing (2/2)

- Formally, the set of geotagged items is:

$$G = \{g_1, g_2, \ldots\}$$

where each g_i contains $user_{ID}$, *coordinates*, *timestamp*, *text* and *tags*.

$$g_i = \langle 111122222, [-23.545531, -46.473372], \text{2015-09-01T15:03}, \text{“Discovering the Italian excellence in food”}, [\#Italy, \#food, \#excellence] \rangle$$

- ***G*** was pre-processed by keeping one Instagram post per user per pavilion per day, as we were interested to know only if a user visited a pavilion (or not) in a given day.

Identification of users

- $U = \{u_1, u_2, \ldots\}$ is the set of Instagram users who have published at least one Instagram post at EXPO 2015.
- Each user u_i contains *userID* and *nationality*.
- For Italian visitors, we also recorded city and region of origin.
- The origin of a user u_i has been extracted by analyzing the location of the posts she/he published during the 30 days preceding her/his visit to EXPO.

Input dataset

- Input dataset ***D*** contains the list of PoIs visited by each user:

$$D=\{T_1, T_2, \ldots, T_n\}$$

where

$$T_i=<u_i,\{p_{i1}, p_{i2}, \ldots, p_{ik}\}>$$

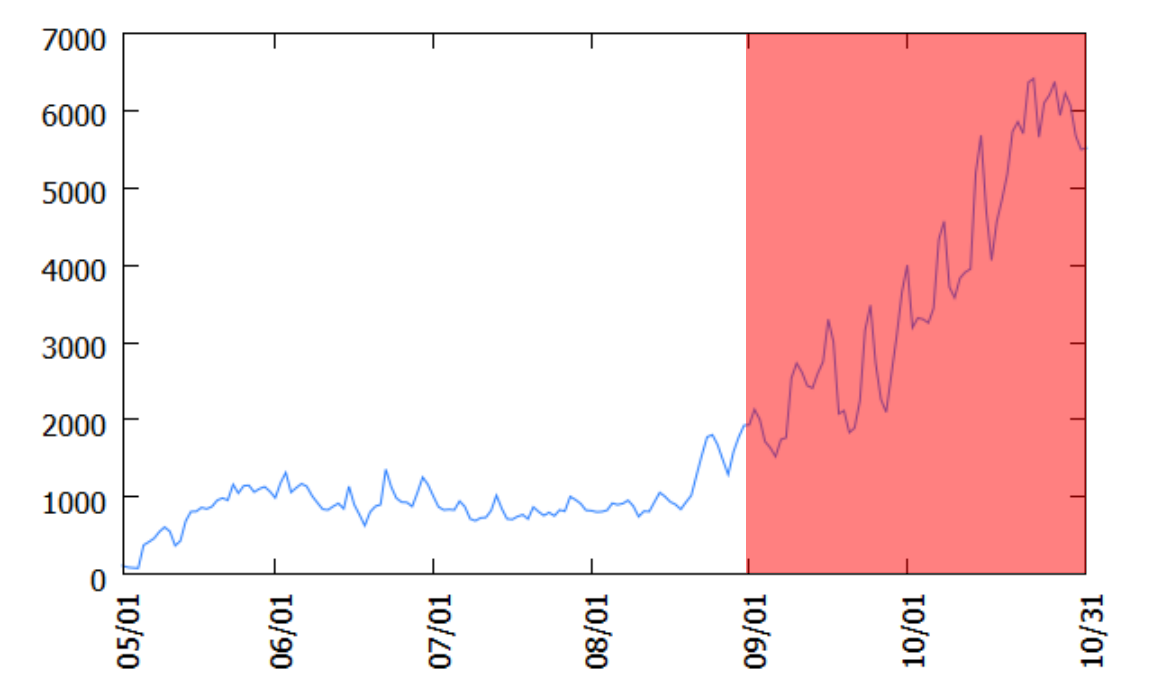
and p_{ij} contains *event* (e.g., visit to a given Pavilion) and *timestamp* of the j^{th} post published by Instagram user u_i.

- The PoI of a post sent during the EXPO visit corresponds to the visited pavilion, while the PoI of a post sent before or after the visit corresponds to a city/region/state (outside EXPO).

Results

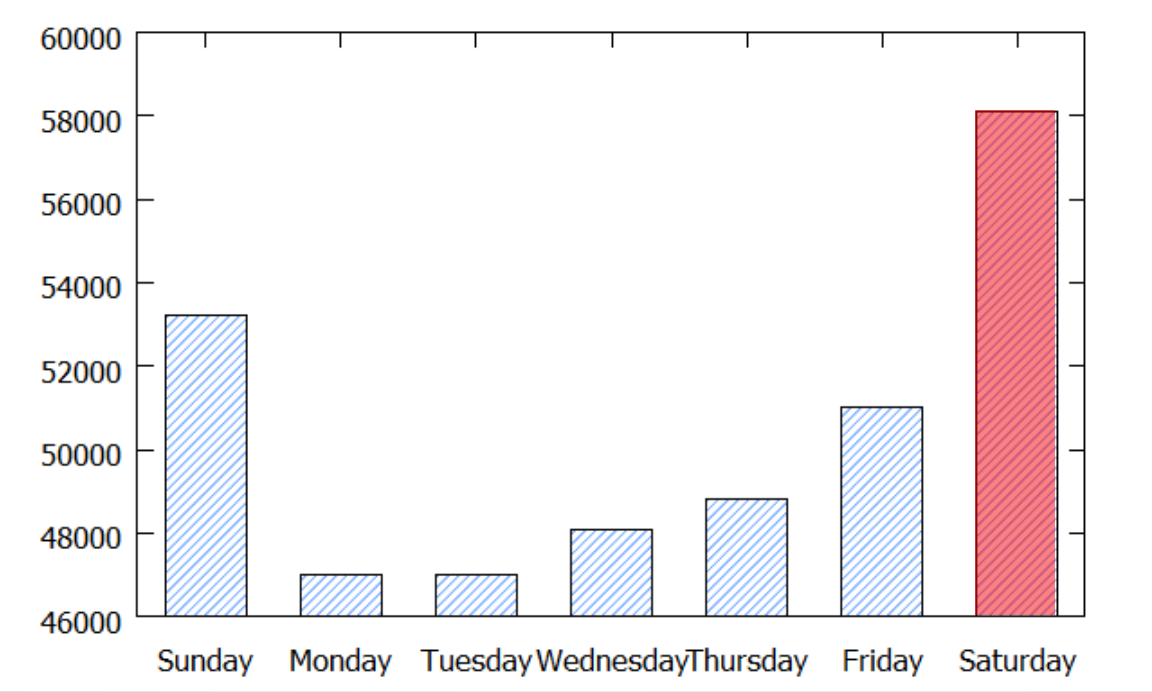
- **Visit trends**
- **Most visited pavilions**
- **Most frequent sets of visited pavilions**
- **Origin of visitors**
- **Destination of foreign visitors**

Visit trends (1/3)



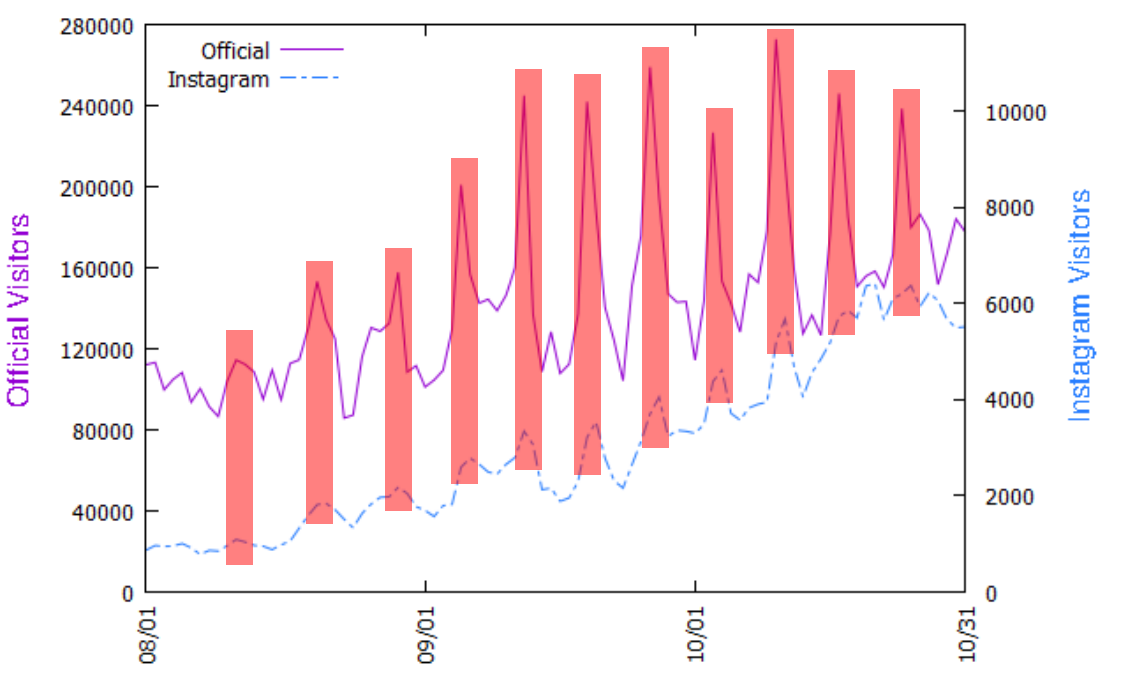
- Visitors increased significantly during the last two months (September and October 2015)

Visit trends (2/3)



- There is a peak of visits during the weekend days (the highest number of visitors is registered on Saturdays).

Visit trends (3/3)



- Strong correlation (Pearson coefficient 0.7) between official visitor numbers and Instagram users

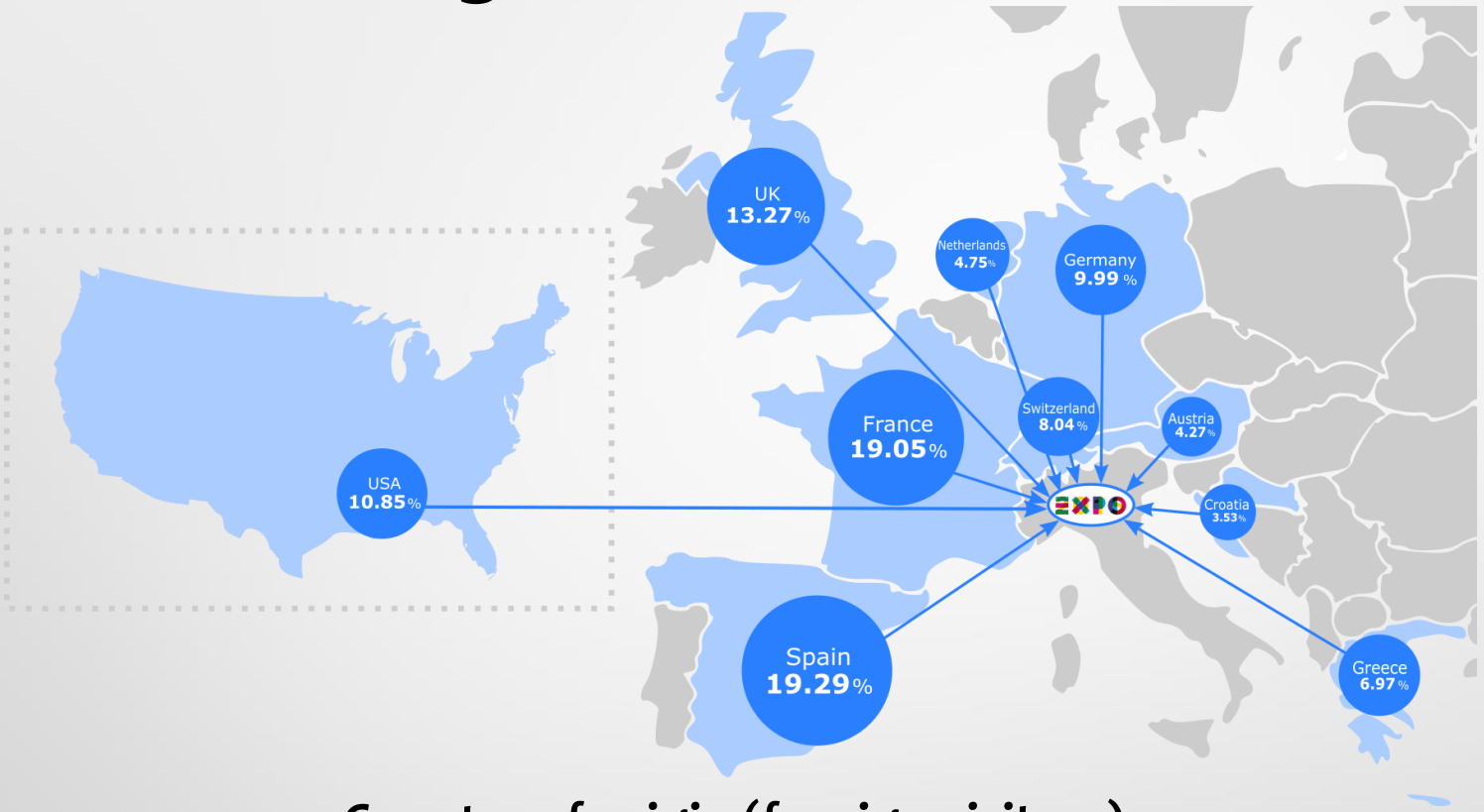
Most visited pavilions

Rank	Pavilion	Visitors
1	China	20.77%
2	Japan	16.38%
3	UK	14.40%
4	Korea	13.03%
5	Russia	13.02%
6	Brazil	12.56%
7	USA	11.75%
8	UAE	9.32%
9	Qatar	9.09%
10	Italy	8.59%
11	Netherlands	7.75%
12	Austria	7.72%
13	Spain	6.94%
14	Thailand	6.78%
15	Azerbaijan	6.48%
16	Poland	6.46%
17	Vietnam	6.38%
18	Nepal	6.35%
19	France	6.08%

Most frequent sets of visited pavilions

Rank	Pavilion 1	Pavilion 2	Support
1	China	Japan	3.77%
2	China	UK	3.75%
3	China	Korea	3.29%
4	China	Russia	3.04%
5	Brazil	China	3.03%
6	Japan	UK	2.92%
7	China	USA	2.76%
8	Japan	Russia	2.57%
9	Japan	Korea	2.51%
10	Korea	UK	2.42%
11	Japan	USA	2.39%
12	China	UAE	2.23%
13	Russia	USA	2.20%
14	Russia	UK	2.19%
15	Brazil	Japan	2.14%
16	Brazil	UK	2.13%
17	China	Qatar	2.09%
18	China	Thailand	2.04%
19	Japan	UAE	2.03%

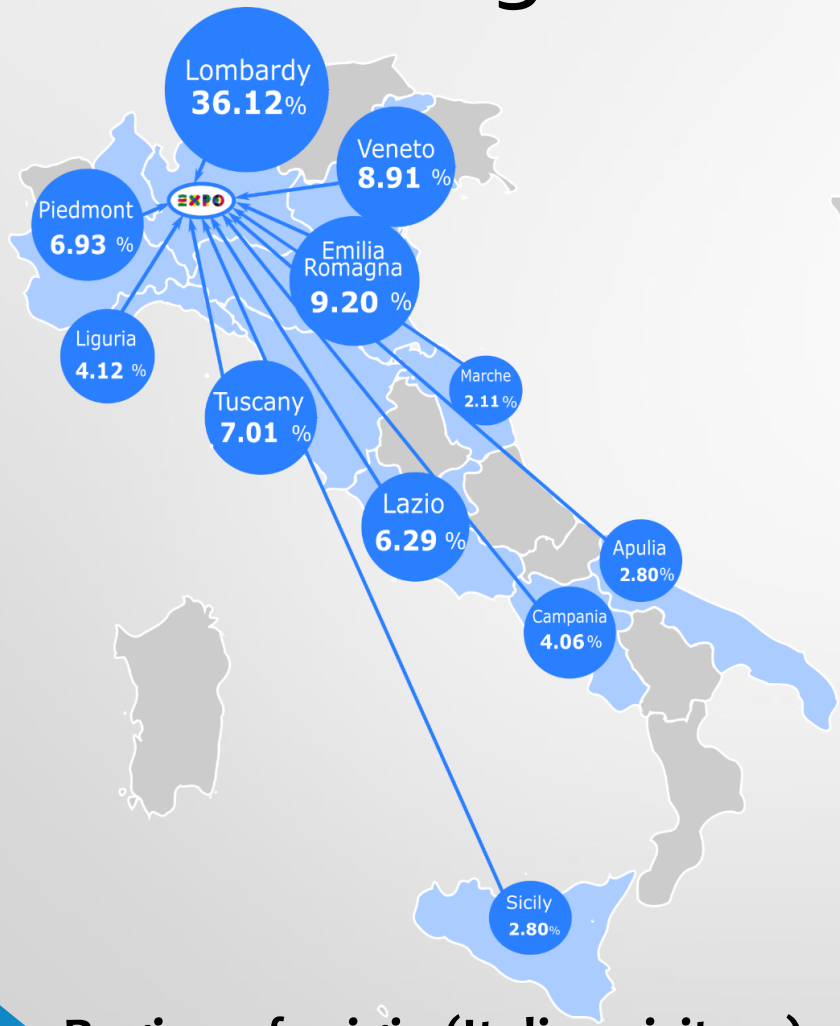
Origin of visitors (1/2)



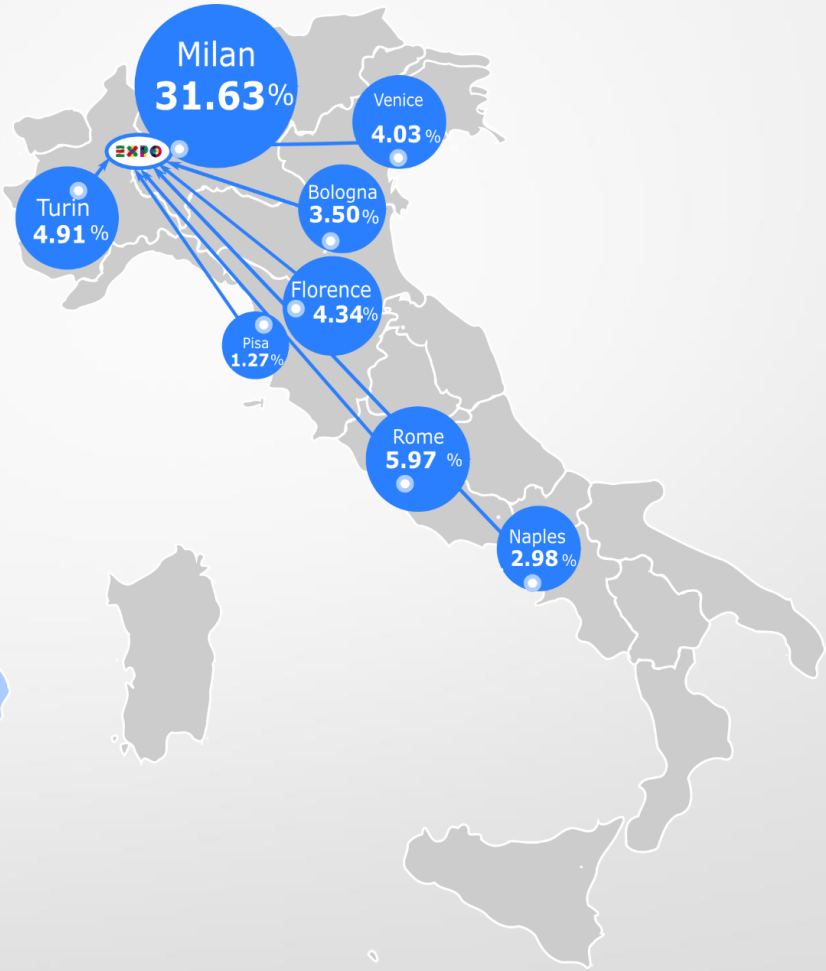
Country of origin (foreign visitors)

- Largest foreign inflows from Spain and France (19.3% and 19.1%, resp.), followed by UK (13.3%) and USA (10.9%)

Origin of visitors (2/2)

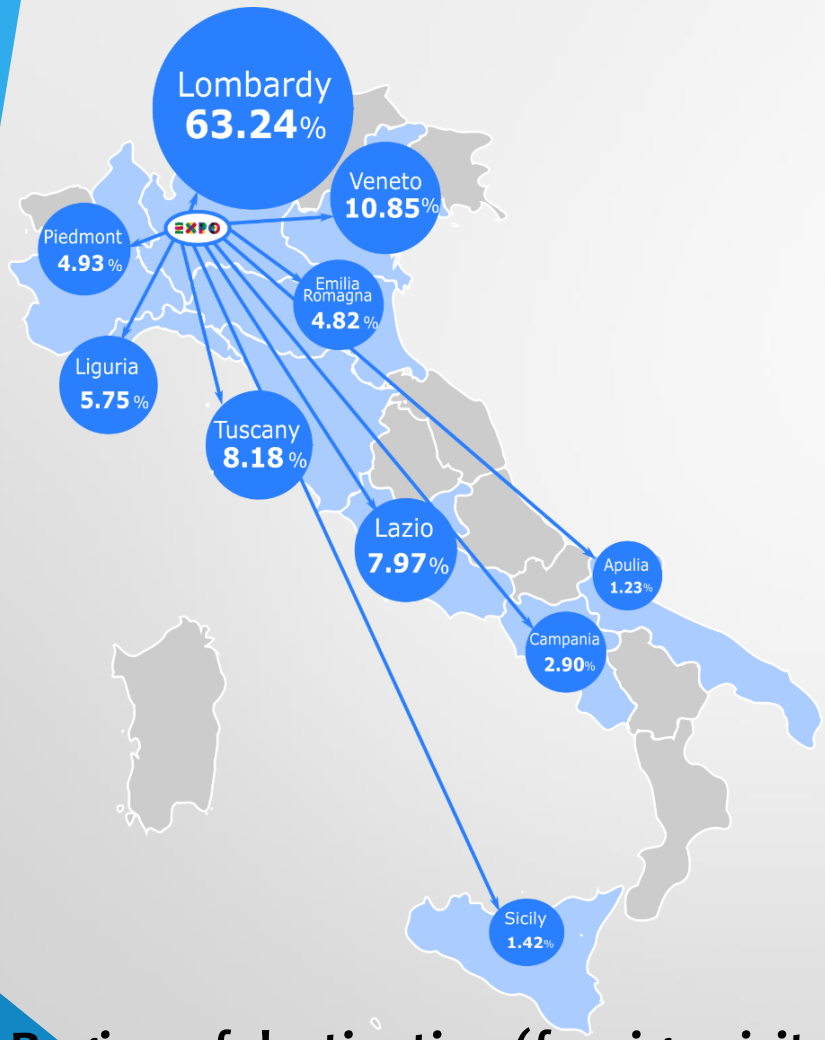


Region of origin (Italian visitors)

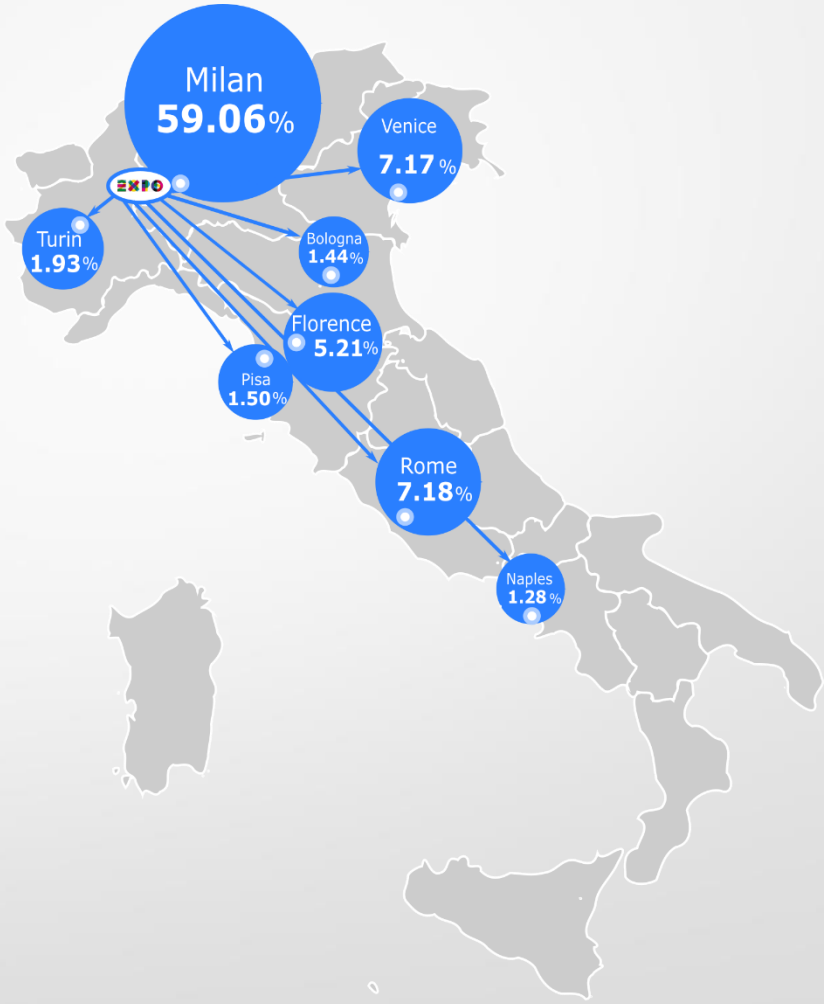


City of origin (Italian visitors)

Destination of foreign visitors



Region of destination (foreign visitors)



City of destination (foreign visitors)

Hands-on: Trajectory mining on social media data

Outline

- General structure of a **social data analysis application**
- Main steps of a **trajectory mining application**
- Definition of the trajectory mining application using real data (Flickr photos).
- Configuration of **Cloudera**
- Hands on (1): A trajectory mining application using **Java Stream** for small datasets
- Hands on (2): A trajectory mining application using **MapReduce (Hadoop)** for big datasets

Trajectory mining from social media

- Social media posts are often tagged with **geographical coordinates** or **other information** (e.g., text, location fields) that allows identifying users' positions.
- Therefore, users moving through a set of locations produce a huge amount of geo-referenced data that embed extensive knowledge about **mobility behaviors**.
- In the latest years, there has been a growing interest in the extraction of trajectories from geotagged social data using **trajectory mining** techniques (*).

(*) E. Cesario, F. Marozzo, D. Talia, P. Trunfio, "SMA4TD: A Social Media Analysis Methodology for Trajectory Discovery in Large-Scale Events". Online Social Networks and Media, vol. 3-4, pp. 49-62, October 2017.

A. Altomare, E. Cesario, C. Comito, F. Marozzo, D. Talia, "Trajectory Pattern Mining for Urban Computing in the Cloud". Transactions on Parallel and Distributed Systems, vol. 28, n. 2, pp. 586-599, 2017

Social data analysis applications - General structure (1/2)

- Often, a social data analysis application can be structured in 7 steps:

1. **Data acquisition:** to run multiple crawlers in parallel for collecting and storing social media items (e.g., on a distributed file system).
2. **Data filtering:** to filter social media items according to a set of filtering functions.
3. **Data mapping:** to transform the information contained in social media items by applying a set of map functions.

Social data analysis applications General structure (2/2)

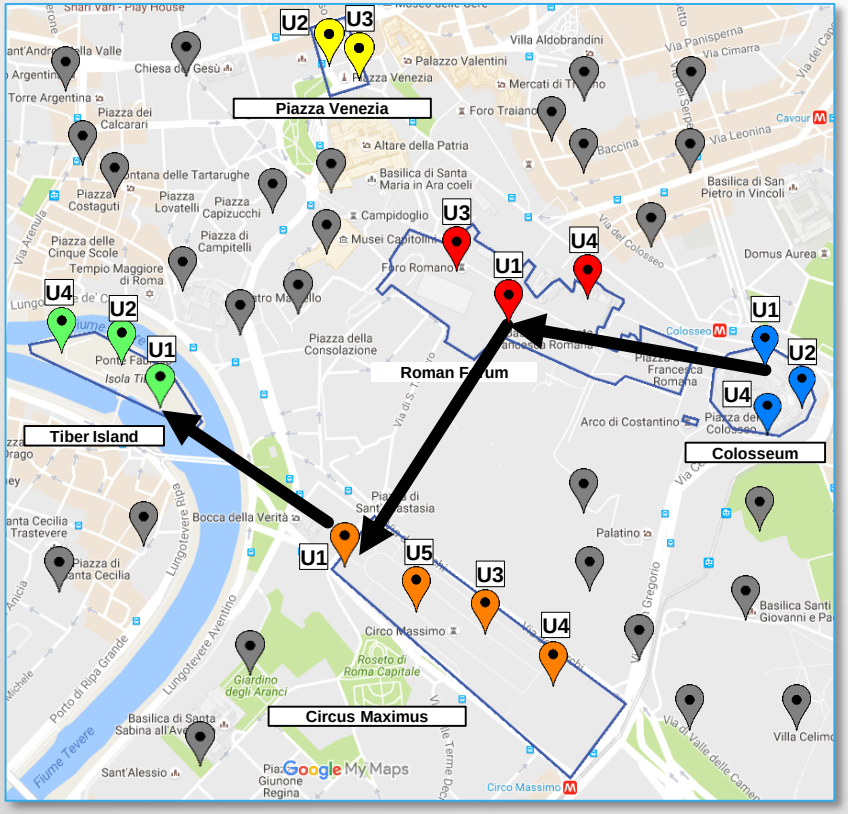
4. **Data partitioning:** here data is partitioned into shards by a primary key and then sorted by a secondary key.
5. **Data reduction:** this step aggregates all the data contained in a shard according to the provided reduce function.
6. **Data analysis:** given a data analysis or mining function, this step analyzes data to extract the knowledge of interest.
7. **Data visualization:** a visualization function is applied on the data analysis results to present them in the desired format.

Algorithms for trajectory mining

- Sequential pattern analysis (or trajectory mining) algorithms are intended to discover the sequences of elements that occur most frequently in the data.
- Unlike the frequent item set analysis, in sequential analysis are fundamental the time dimension and the chronological order in which the values appear in the data.
- In our case, this type of analysis is useful to discover the most frequent mobility patterns among some places of interest.
- We use a parallel version of the PrefixSpan algorithm (*)

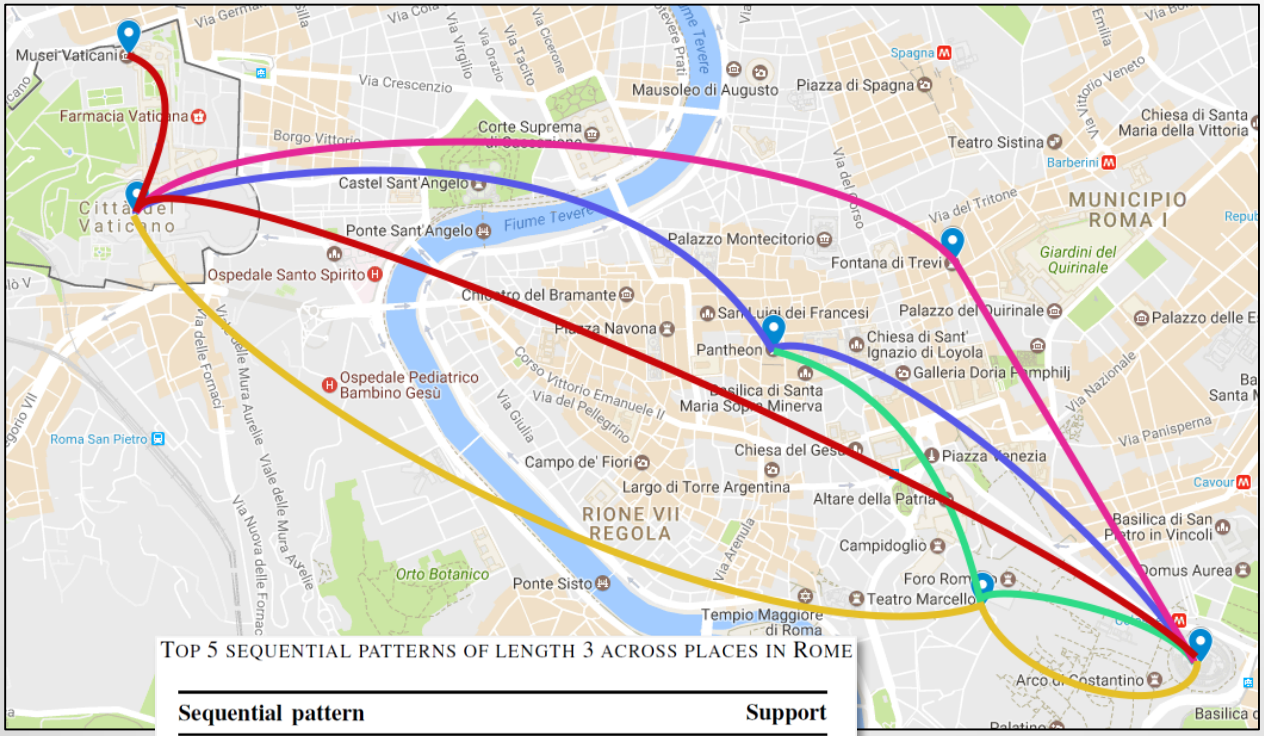
(*) Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.C.: Mining sequential patterns by pattern-growth: the prefixspan approach. IEEE Transactions on Knowledge and Data Engineering 16(11), 1424{1440 (Nov 2004)

Trajectory mining



Given a user (e.g. U1), we can extract his/her movements across places by grouping social media items by user id and sorting them by date and time.

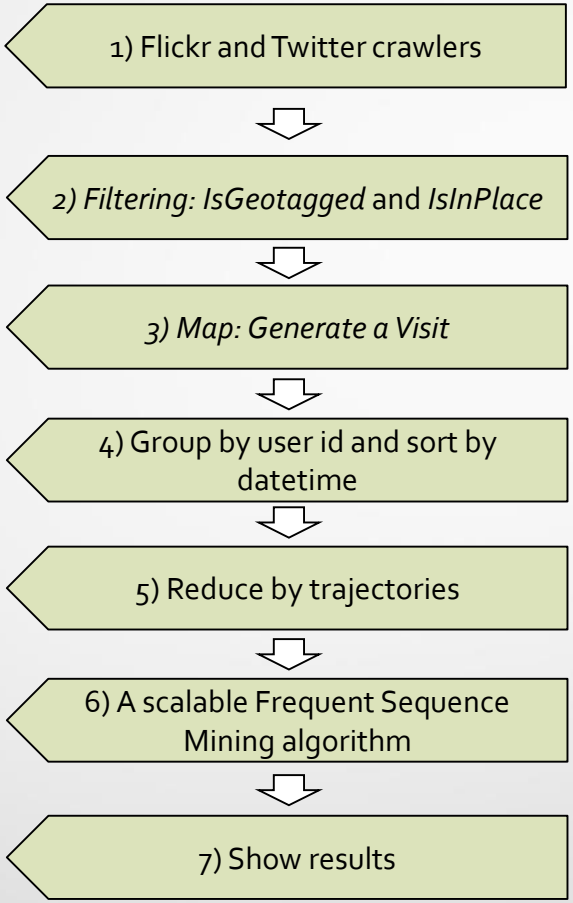
Trajectory mining



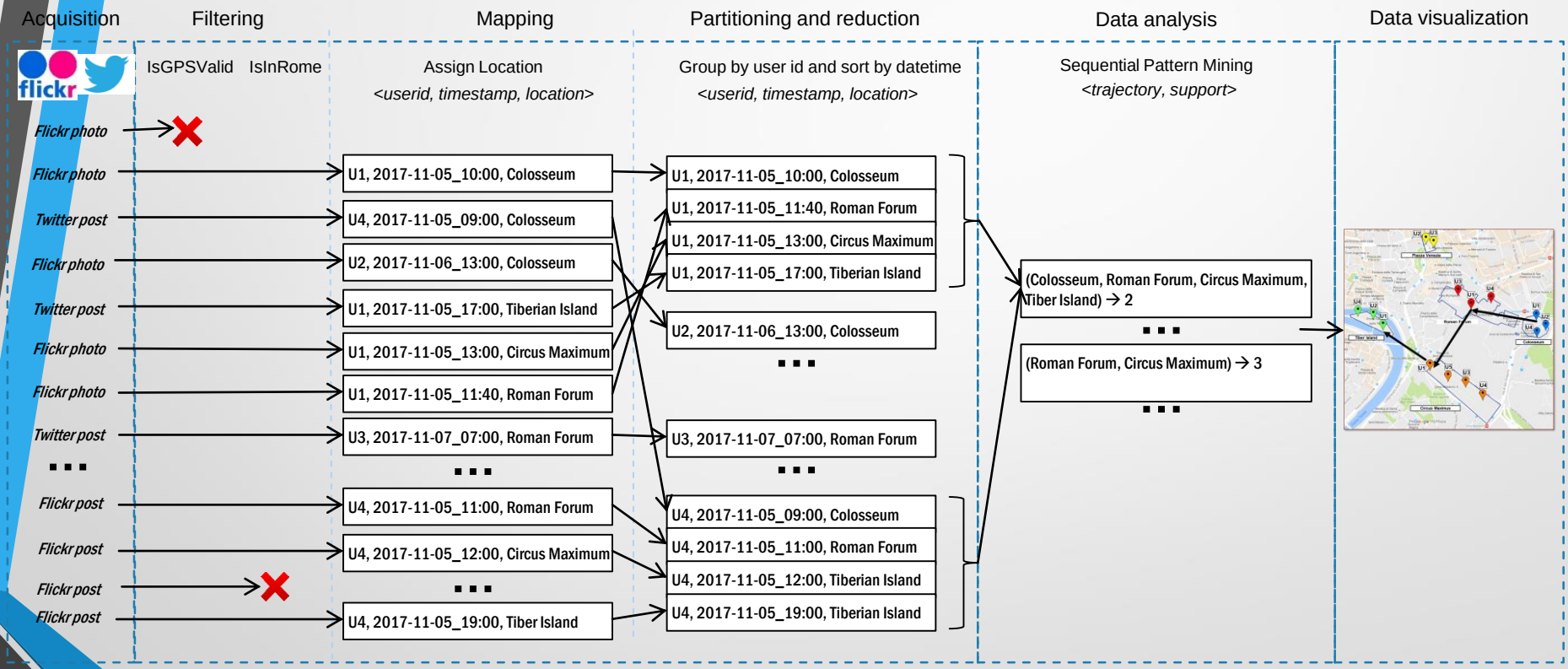
TOP 5 SEQUENTIAL PATTERNS OF LENGTH 3 ACROSS PLACES IN ROME

Sequential pattern	Support
Colosseum → Roman Forum → St. Peter's Basilica	4.4%
Vatican Museums → St. Peter's Basilica → Colosseum	3.9%
Colosseum → Trevi Fountain → St. Peter's Basilica	3.7%
Colosseum → Roman Forum → Pantheon	3.6%
Colosseum → Pantheon → St. Peter's Basilica	3.6%

Main steps of a trajectory mining application



Trajectory mining application– Execution flow

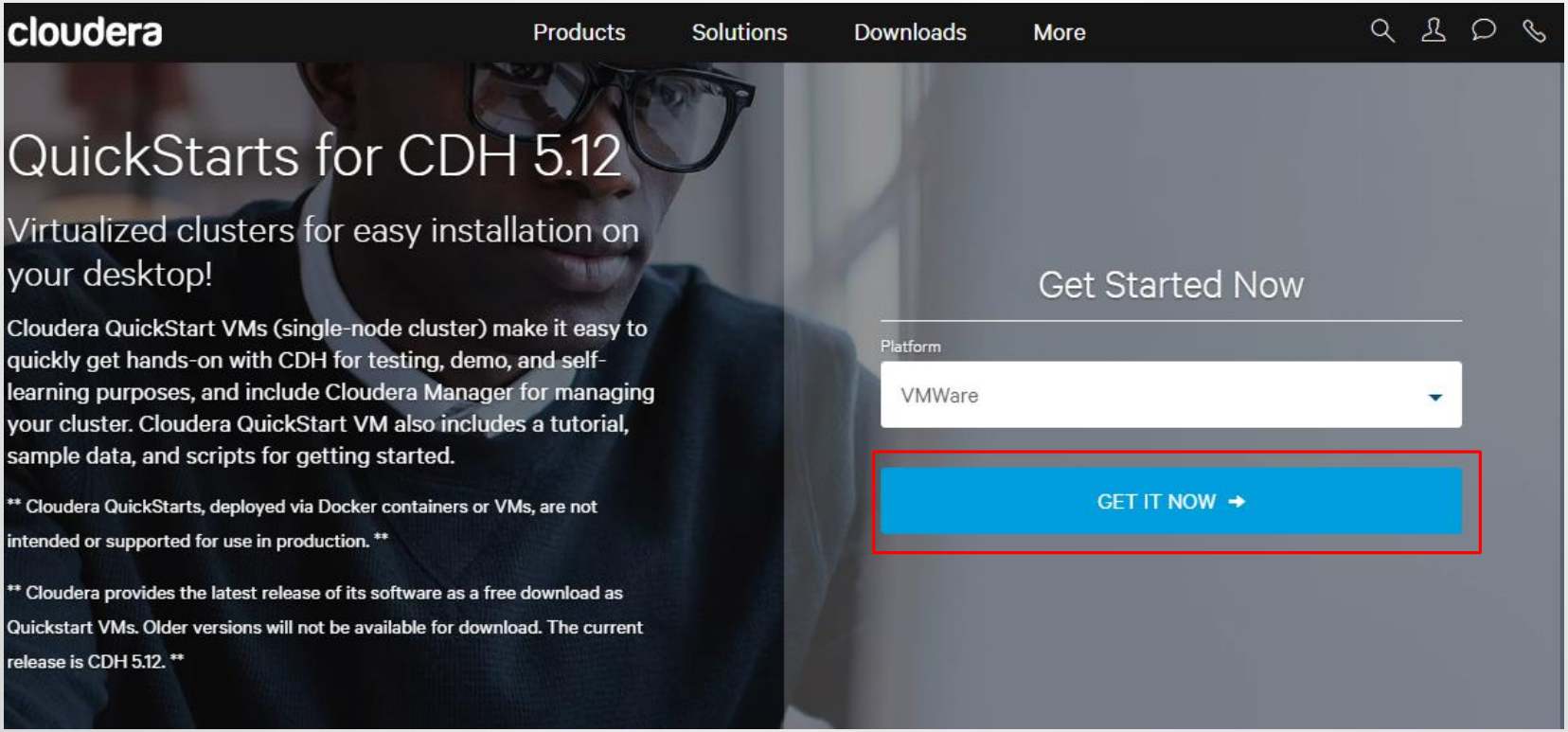


Configuration of Cloudera

Download VMware Workstation Player (Non-commercial use)

- https://www.vmware.com/products/workstation-player/workstation-player-evaluation.html

Download Cloudera



From https://www.cloudera.com/downloads/quickstart_vms/5-12.html the cloudera-quickstart-vm-5.12.0-0-vmware.zip file (5.5 GB)

Cloudera VM requirements

System Requirements | Installed Products | Getting Started

Cloudera QuickStart virtual machines (VMs) include everything you need to try CDH, Cloudera Manager, Cloudera Impala, and Cloudera Search. The VM uses a package-based install. This allows you to work with or without Cloudera Manager. Parcels do not work with the VM unless you first migrate your CDH installation to use parcels. On your production systems, Cloudera recommends that you use parcels.

Prerequisites

- These 64-bit VMs require a 64-bit host OS and a virtualization product that can support a 64-bit guest OS.
- To use a VMware VM, you must use a player compatible with WorkStation 8.x or higher:
  - Player 4.x or higher
  - Fusion 4.x or higher

  Older versions of WorkStation can be used to create a new VM using the same virtual disk (VMDK file), but some features in VMware Tools are not available.
- The amount of RAM required varies by the run-time option you choose:

CDH and Cloudera Manager Version	RAM Required by VM
CDH 5 (default)	4+ GiB*
Cloudera Express	8+ GiB*
Cloudera Enterprise (trial)	10+ GiB*

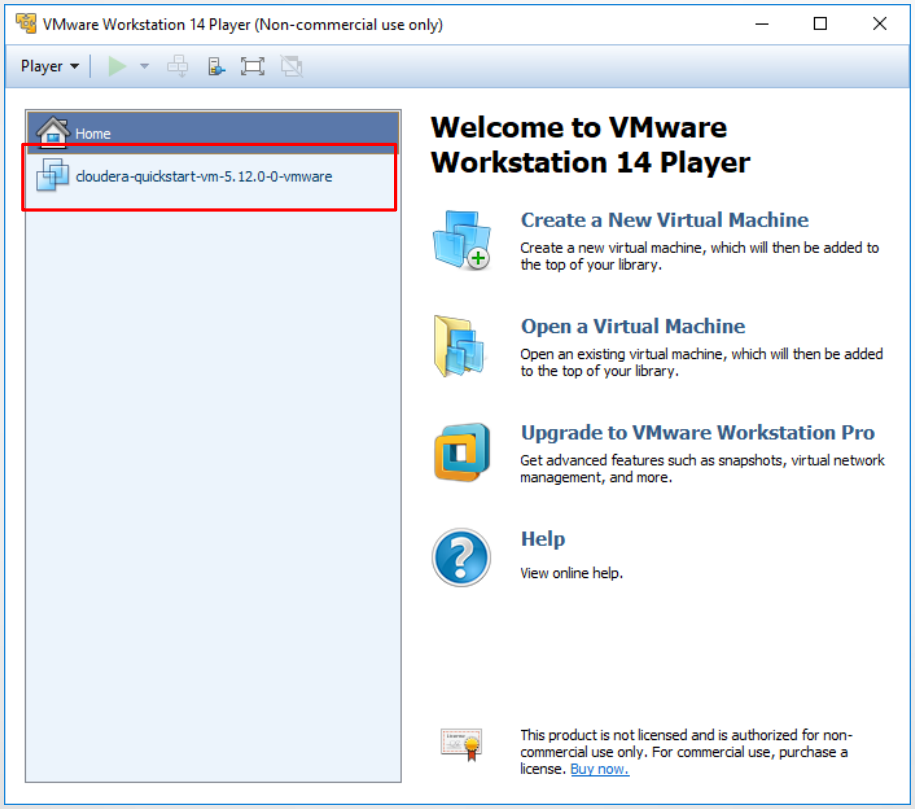
*Minimum recommended memory. If you are running workloads larger than the examples provided, consider allocating additional memory.

Hard disk space >15GB

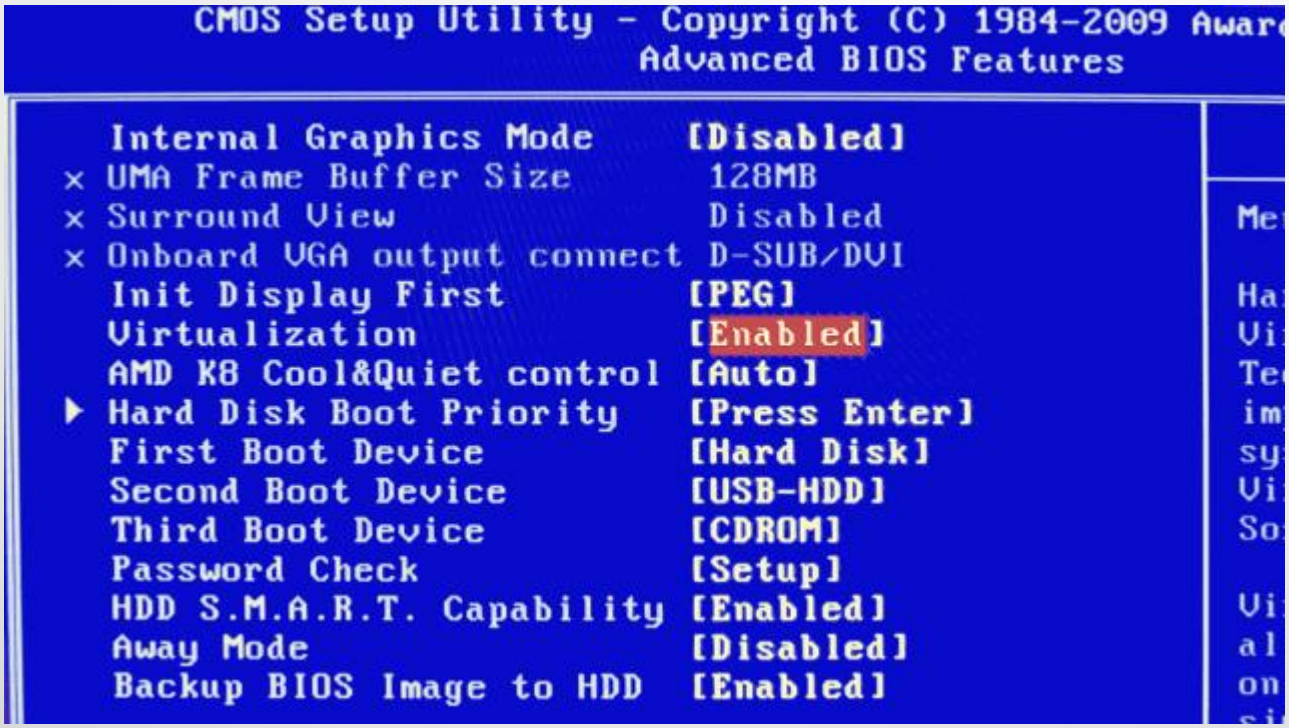
Datasets and code

- **NESUS2018-FabrizioMarozzo-Dataset+Code.tar.gz** (a zip file of 109 MB)
  - A sample dataset FlickrRomeSample.json (5MB)
  - A real dataset FlickRome2017.json (1.3 GB)
  - Source code (3 packages: common, javastream, mapreduce)
    - javastream.MainJavaStream: Main of the Java Stream application;
    - Mapreduce.MainMR: Main of the MapReduce application.

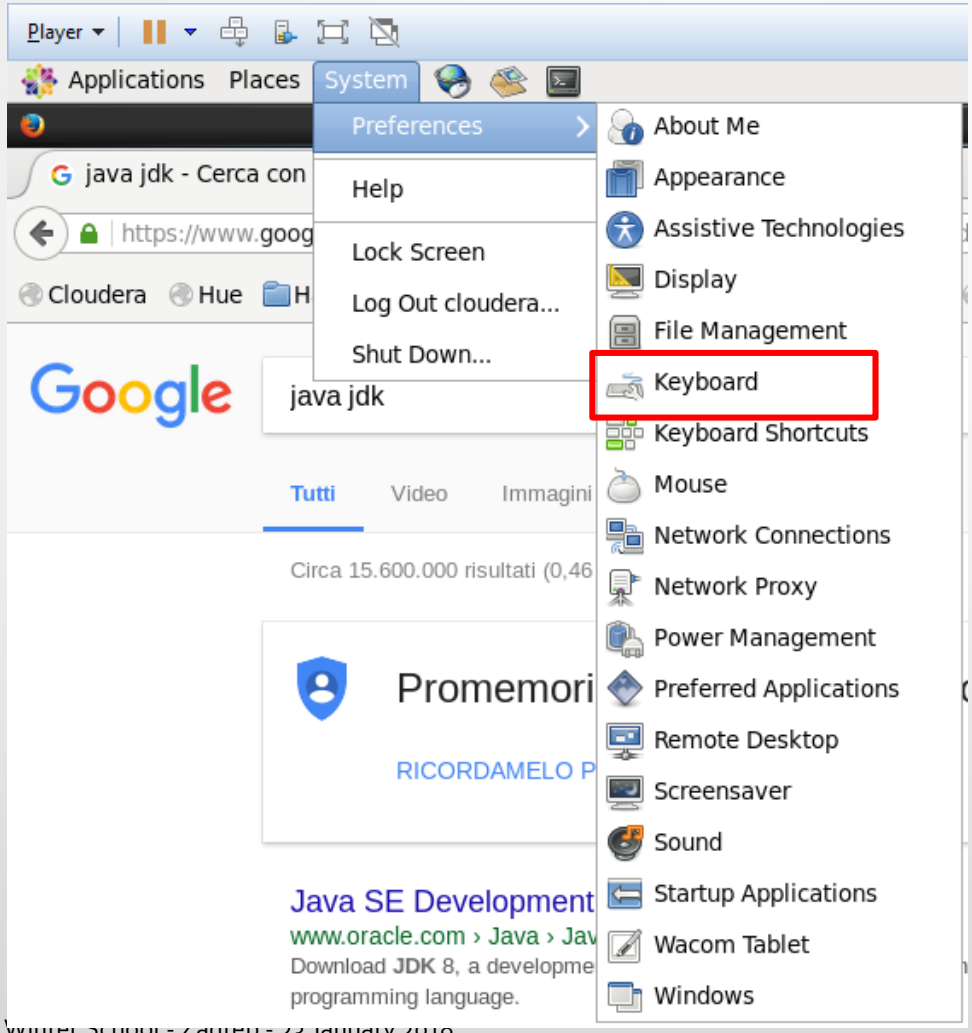
Launch Cloudera VM



Enabling Virtualization in your PC BIOS



Configure keyboard layout



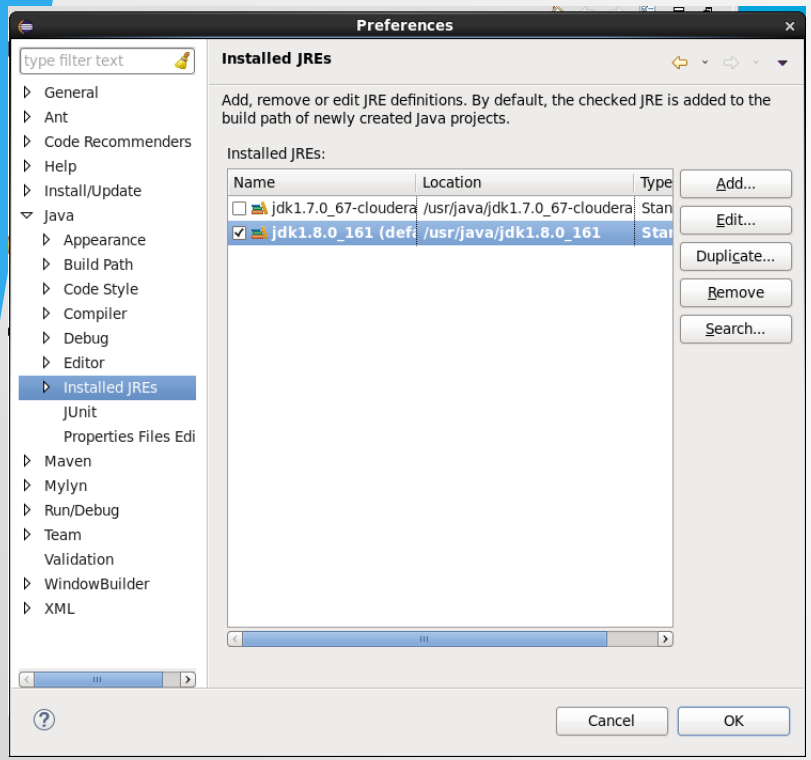
Cloudera VM Administrative Information

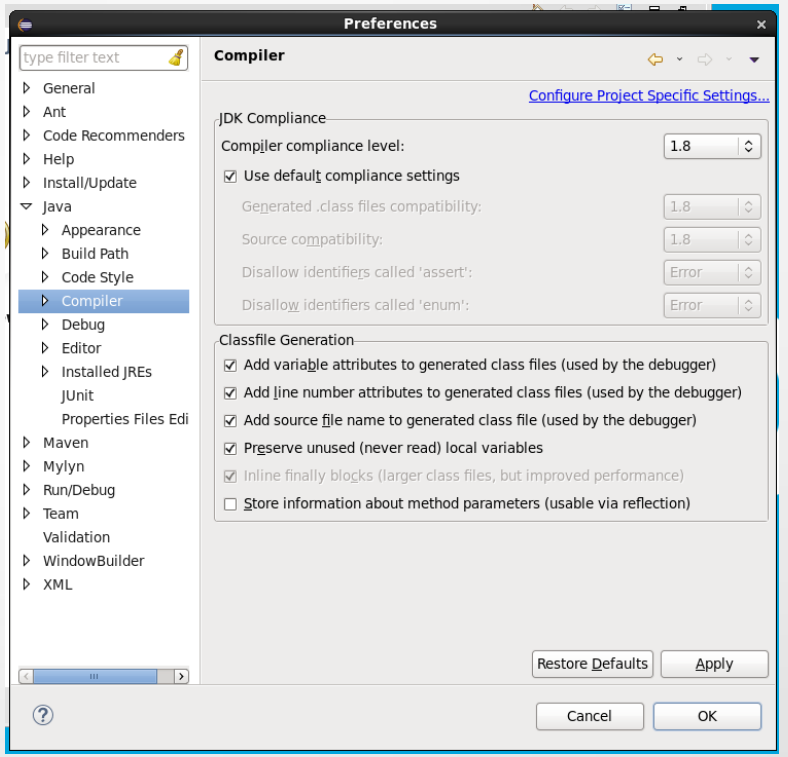
- OS: Red Hat Enterprise Linux 6 64-bit
- Once you launch the VM, you are automatically logged in as the cloudera user. The account details are:
  - username: ***cloudera***
  - password: ***cloudera***
- The cloudera account has *sudo* privileges in the VM. The root account password is cloudera.
- https://www.cloudera.com/documentation/enterprise/5-6-x/topics/quickstart_vm_administrative_information.html

Install JDK8

- Download a Linux x64 version of JDK8 from Oracle web site
- http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
- Run the command
  - yum install jdk-8u162-linux-x64.rpm

Configure Eclipse with JDK8





- Use JDK1_8 in /usr/java/jdk1.8.0_161 and set the compiler level to 1.8

Data stream

A JSON file (a list of String)

media	"photo"
mediaSt...	"ready"
notes	...
originalF...	"jpg"
originalH...	0
originalS...	""
original...	0
owner	...
pathAlias	""
placeId	"FphPyURWU7ux6h4"
primary	false
publicFlag	true
rotation	-1
secret	"eb4efe1bbc"
server	"5497"
tags	...
title	"#Colosseum under the moon #Colosseum ...
url	"https://flickr.com/photos/61240032@N05/31...
urls	...
views	21

A list of Geotagged Objects

```
public abstract class GeotaggedItem {

    public abstract User extractUserInfo(JSONObject obj);

    public abstract String extractId(JSONObject obj);

    public abstract boolean check(JSONObject obj);

    public abstract Point extractLocationInfo(JSONObject

    public abstract LocalDateTime extractDateTimeInfo(JSC

    public GeotaggedItem(String json) {
        this(new JSONObject(json));
    }

    public GeotaggedItem(JSONObject jsonObject) {

        if (!check(jsonObject))
            throw new IllegalArgumentException();

        JSONObject basicJson = new JSONObject();
```

A list of Visit Objects

```
public class Visit {

    private String userId;
    private LocalDateTime dateTime;
    private String location;

    public Visit(String userId, LocalD
        super();
        this.userId = userId;
        this.dateTime = dateTime;
        this.location = location;
    }

    public Visit(GeotaggedItem geoItem
        this(geoItem.getUserId(), geoI
    }

    public String getUserId() {
        return userId;
    }
    public void setUserId(String u) {
```

A file (a list of trajectories)

```
StPeterBasilica
Colosseum VaticanMuseums StPeterBasilica
CapitolineHill RomanForum
PiazzaNavona MausoleumOfHadrian
StPeterBasilica
CapitolineHill
VaticanMuseums StPeterBasilica
Colosseum Colosseum PiazzaVenezia
TreviFountain
Colosseum MausoleumOfHadrian Trastevere P
Pantheon Trastevere
TreviFountain
Colosseum
PiazzaNavona
CapitolineHill
Colosseum RomanForum Colosseum VaticanMus
VaticanMuseums StPeterBasilica MausoleumO
Colosseum TreviFountain StMaryMajor Colos
PiazzaVenezia
StPeterBasilica Colosseum PiazzaVenezia
StPeterBasilica Colosseum
CapitolineHill
```

A file (Sequential patterns)

```
1 *** SEQUENTIAL PATTERNS OF LENGTH 2 ***
2 Colosseum RomanForum --> 9
3 Colosseum StPeterBasilica --> 9
4 TreviFountain Colosseum --> 7
5 Pantheon Colosseum --> 7
6 RomanForum StPeterBasilica --> 7
7 Pantheon TreviFountain --> 7
8 RomanForum Colosseum --> 6
9 StPeterBasilica MausoleumOfHadrian --> 6
10 PiazzaDiSpagna Colosseum --> 6
11 StPeterBasilica Pantheon --> 6
12 Colosseum PiazzaVenezia --> 6
13 Colosseum VaticanMuseums --> 6
14 PiazzaNavona Pantheon --> 5
15 Pantheon PiazzaDiSpagna --> 5
16 StPeterBasilica PiazzaNavona --> 5
```

Java Stream API (for small dataset)

- Main operations
  - **.stream** or **.parallelStream:** To create a sequential or parallel operations from various data sources (e.g., Collections). A stream represents a sequence of elements and supports different kind of operations to perform computations upon those elements.
    - **.filter**: For filtering a stream of data.
    - **.map**: For transforming a stream of data.
    - **.collect**: for collecting the results of the operations

Java Stream API Main

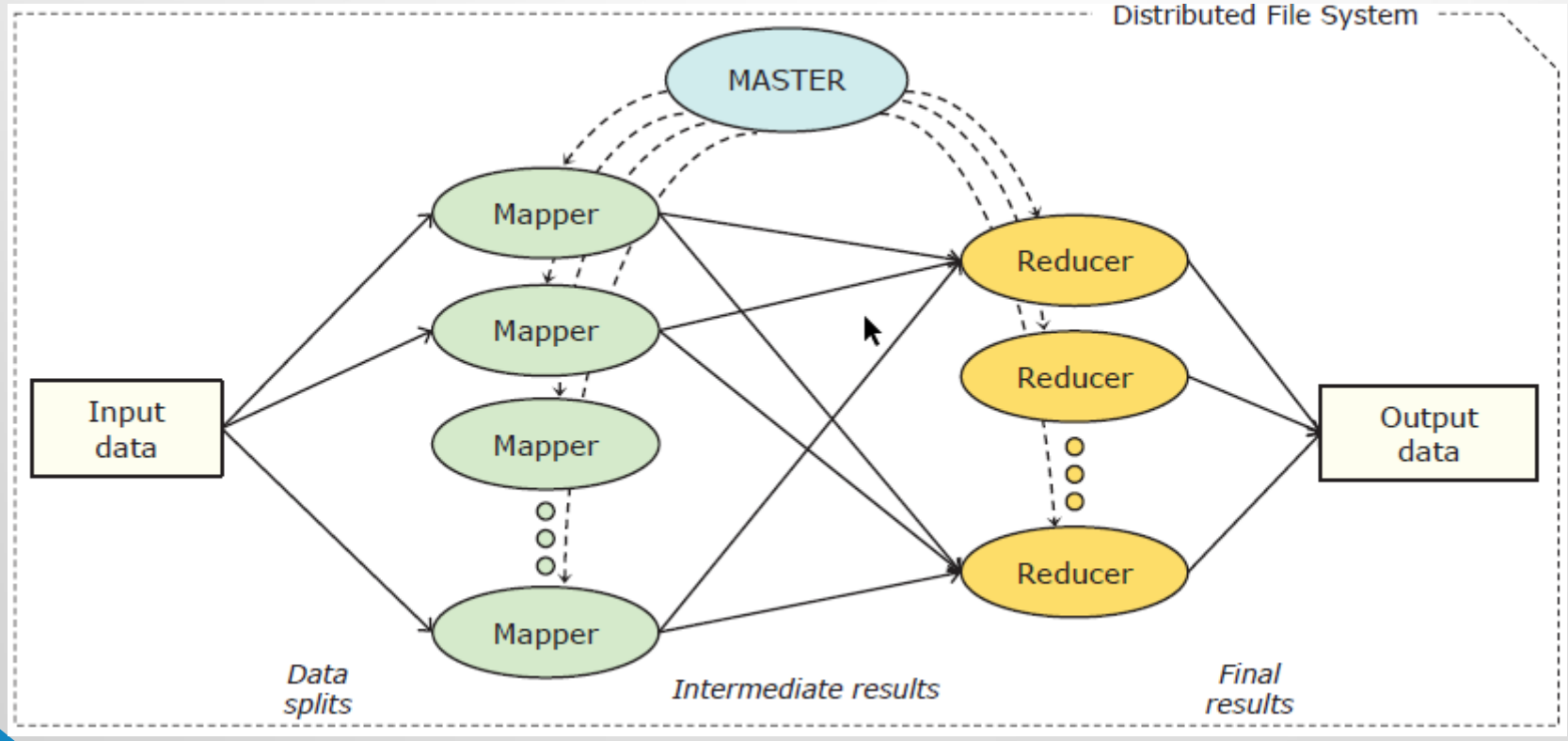
```
public static void main(String[] args) {

    String pathFlickrItems = "FlickrRomeSample.json"; String pathRoIs = "RomeRealShapes.kml";
    File input = new File(pathFlickrItems); List<Roi> rois = Utils.loadRoi(pathRoIs);

    try {
        List<String> data = Files
                .lines(Paths.get(input.toURI())).parallel()
                .map(x -> buildGeotaggedItem(x))
                //filtering
                .filter(Objects::nonNull).filter(x -> isGpsValid(x)).filter(x -> hasLocation(x, rois))
                //mapping
                .map(x -> new Visit(x))
                //partitioning
                .collect(Collectors.groupingByConcurrent(Visit::getUserId))
                //reduction
                .values().parallelStream().map(x -> getTrajectory(x))
                .collect(Collectors.toList());

        Files.write(Paths.get("input.txt"), data);

        //Data analysis
        String params = "-i input.txt -o output -m s -g 3 --tempDir /tmp";
        analyzeData(params.split(" "));

        String vis_params = "output/translatedFS output/sortedFS";
        VisualizeFSMResult.visualize(vis_params.split(" "));
```

MapReduce (1/2)

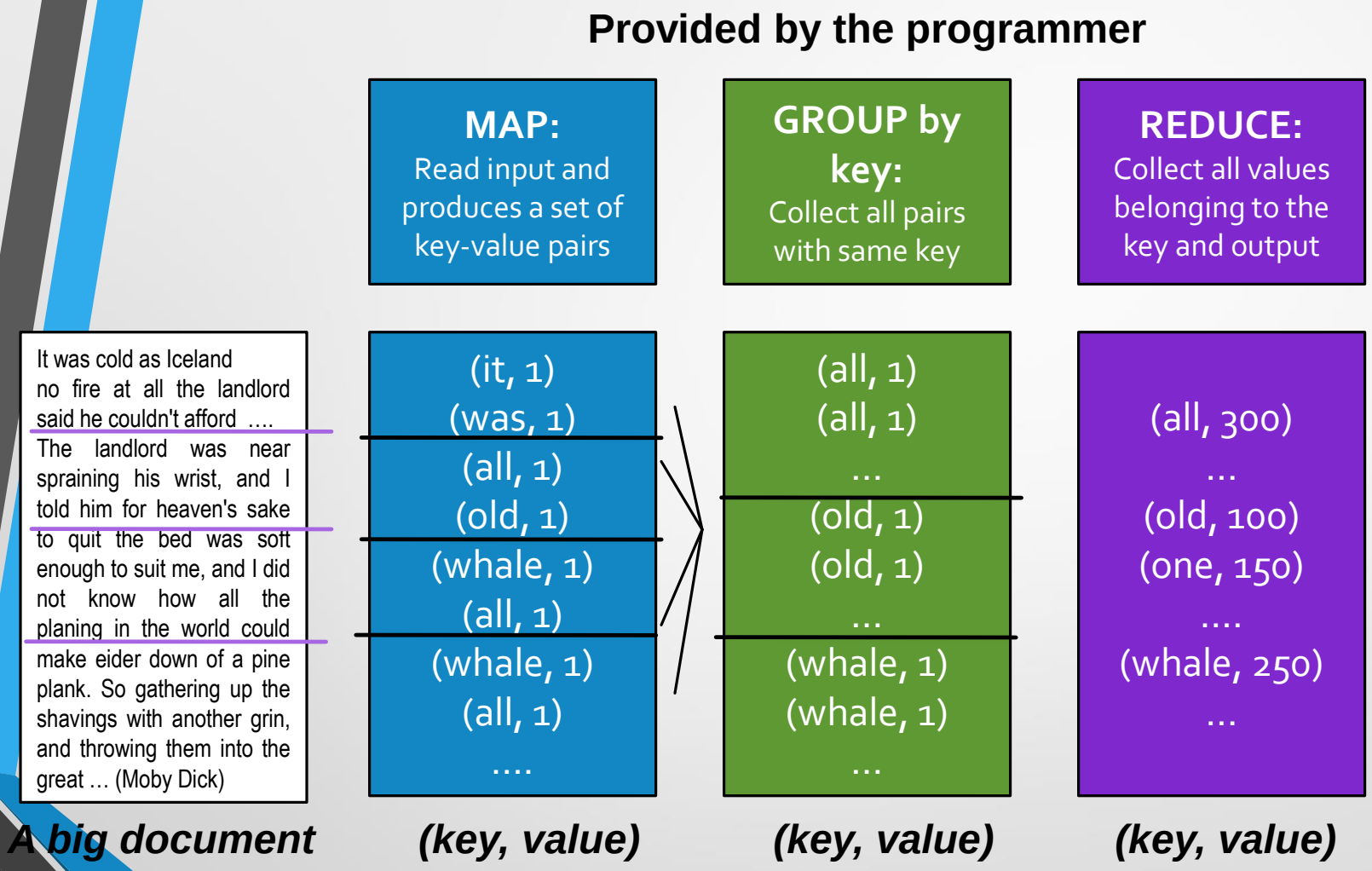
- The model is inspired by the *map* and *reduce* functions commonly used in functional programming.
- The *map* function processes a (key, value) pair and returns a list of intermediate (key, value) pairs:
  - (k1,v1) -> list(k2,v2)
- The *reduce* function merges all intermediate values having the same intermediate key:
  - k2, list(v2) -> list(v3)

MapReduce (2/2)

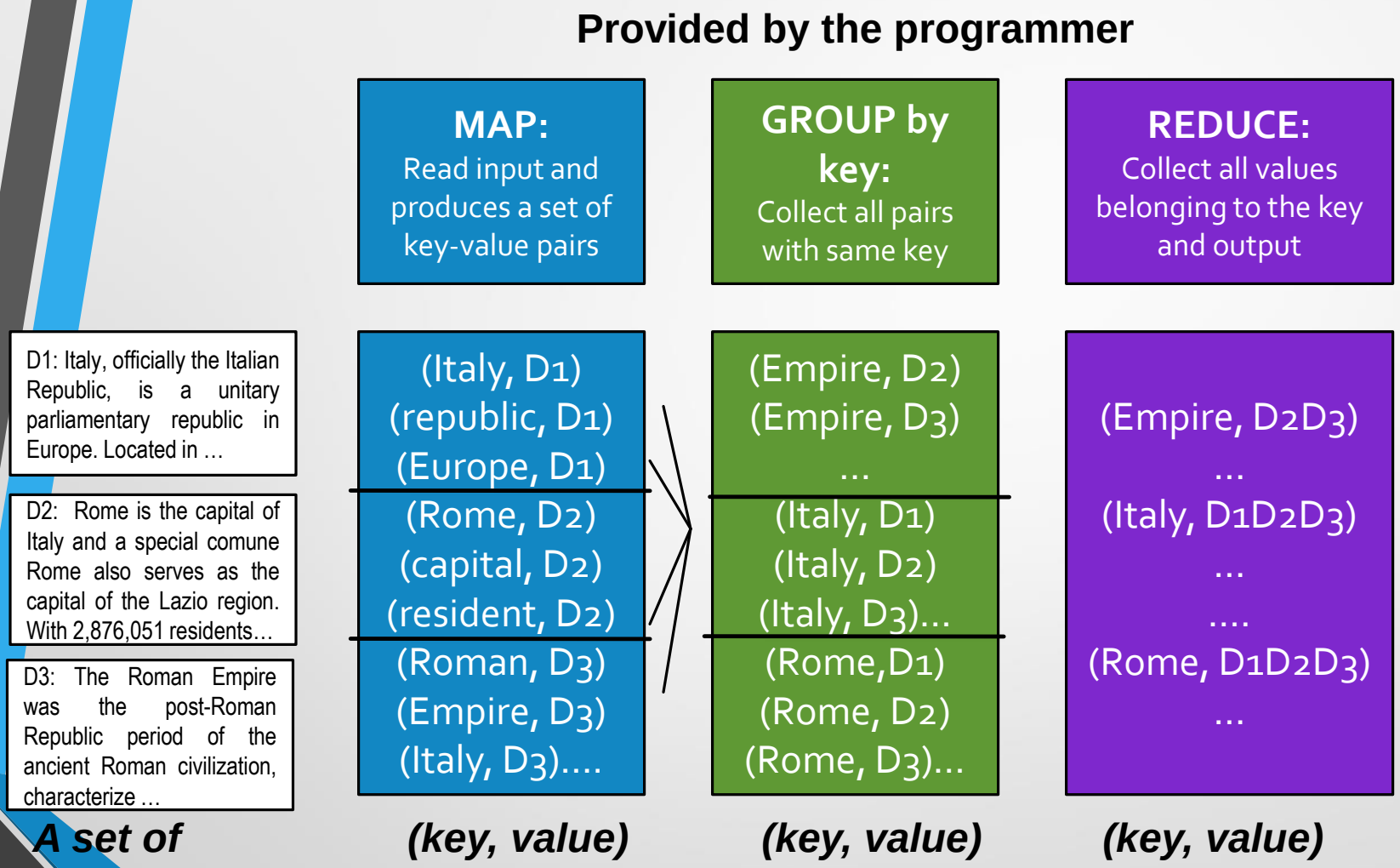
- The MapReduce process can be described as follows:



MapReduce Example: WordCount



MapReduce Example: Inverted index

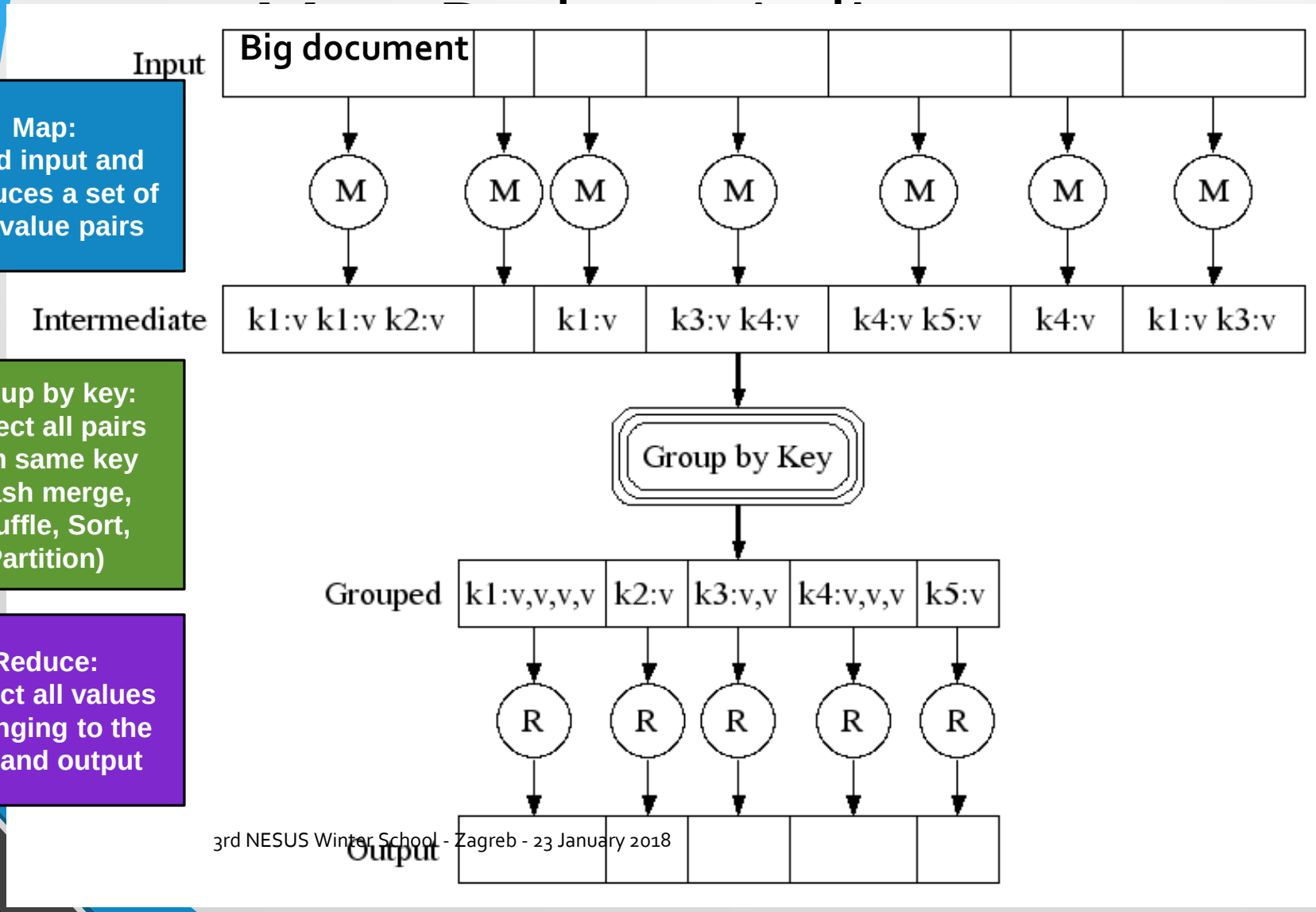


Map-Reduce: A diagram

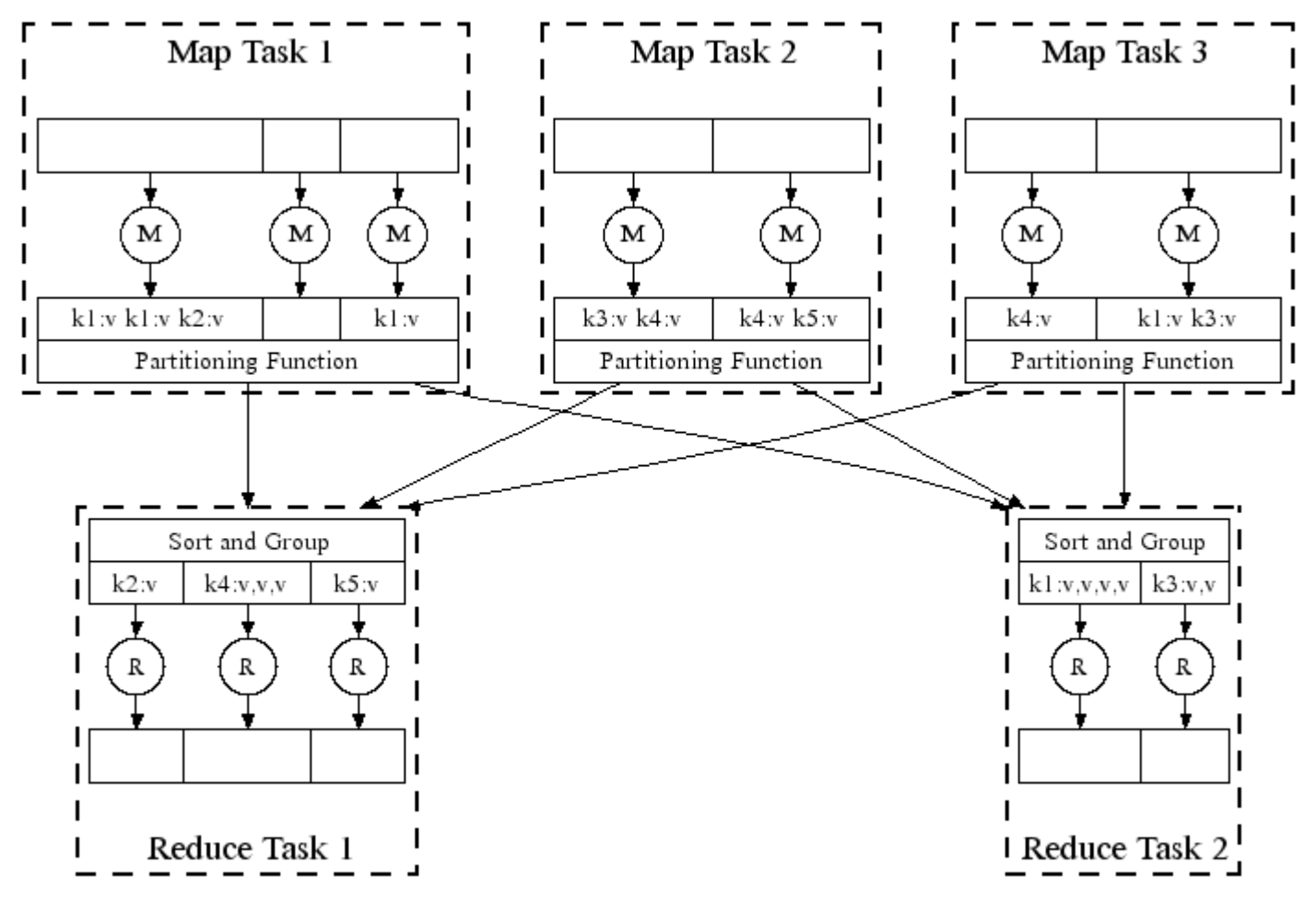
Map:
Read input and produces a set of key-value pairs

Group by key:
Collect all pairs with same key (Hash merge, Shuffle, Sort, Partition)

Reduce:
Collect all values belonging to the key and output



Map-Reduce: In Parallel



Trajectory mining using MapReduce

```
public static void main(String[] args) throws Exception {
    String pathFlickrItems = "FlickrRomeSample.json"; String pathRoIs = "RomeRealShapes.kml"; String outputBasePath = "outputMR/";
    //String pathRoIs = "/home/hadoop/workspace/ParSoDA-COMPSs/RomeRealShapes.kml"; // Use absolute path

    conf.set("fs.defaultFS", "file:///"); //conf.set("fs.defaultFS", "hdfs://10.0.0.100:8020");
    fs = FileSystem.get(conf); job = Job.getInstance(conf);
    job.setJobName("Extracting user trajectories from Rome Flickr dataset");
    job.addCacheFile(new Path(pathRoIs).toUri());

    // Set input and output
    job.setInputFormatClass(TextInputFormat.class); TextInputFormat.setInputPaths(job, new Path(pathFlickrItems));
    job.setOutputFormatClass(TextOutputFormat.class); TextOutputFormat.setOutputPath(job, new Path(outputBasePath+"dataset"));

    // Set Mapper key-value formats
    job.setMapperClass(DataMapper.class); job.setMapOutputKeyClass(TextPair.class); job.setMapOutputValueClass(Text.class);

    // Set Partitioner, Comparator and Sorting
    job.setPartitionerClass(SecondarySort.SSPartitioner.class); job.setGroupingComparatorClass(SecondarySort.SSGroupComparator.class);
    job.setSortComparatorClass(SecondarySort.SSSortComparator.class);

    // Set Reducer
    job.setReducerClass(DataReducer.class); job.setNumReduceTasks(1); job.setOutputKeyClass(NullWritable.class);
    job.setOutputValueClass(Text.class);

    // Delete existing output path
    fs.delete(new Path(outputBasePath), true);

    //Launch MapReduce job
    boolean jobSuccessful = job.waitForCompletion(true);
    if (!jobSuccessful) {
        System.err.println("Error with job " + job.getJobName());
    }

    // Launch MG-FSM Mapreduce job
    String params = "-i " + outputBasePath+"dataset/part-r-00000 -o " + outputBasePath+"mgfsm" + " -m d -g 3 --tempDir /tmp";
    analyzeData(params.split(" "));
    // Post-process results
    String vis_params = outputBasePath + "mgfsm/translatedFS/part-r-00000 " + outputBasePath + "sortedFS";
    VisualizeFSMResult.visualize(vis_params.split(" "));
```

MapReduce - Mapper

```
public void map(LongWritable key, Text geotaggedItemJSON, Context context) throws Interrupted

    item = buildItem(geotaggedItemJSON.toString());
    if (item == null)
        return;

    if(!(isGpsValid(item) && hasLocation(item, rois)))
        return;

    visit = new Visit(item);

    // <GroupKey, SortKey> for SecondarySortMethod
    outputKey.set(visit.getUserId(), visit.getDateTime().toString());
    // Serialize value in JSON format
    outputValue.set(visit.toString());
    context.write(outputKey, outputValue);

}
```

MapReduce - Partitioning and sorting

```
public class SecondarySort {

  private static abstract class TTRawComparator implements RawComparator<TextPair> {
    DataInputBuffer buffer = new DataInputBuffer();
    TextPair a = new TextPair(); TextPair b = new TextPair();

    public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2 ) {
      try {
        buffer.reset(b1, s1, l1); a.readFields(buffer); buffer.reset(b2, s2, l2); b.readFields(buffer); return compare(a,b);
      } catch(Exception ex) { return -1; }
    }
  }

  // Partition only by UID
  public static class SSPartitioner extends Partitioner<TextPair, Object> {
    @Override
    public int getPartition(TextPair k, Object value, int partitions) { return (k.left.hashCode() & Integer.MAX_VALUE) % partitions;  }
  }

  // Group only by UID
  public static class SSGroupComparator extends TTRawComparator {
    public int compare(TextPair first, TextPair second) { return first.left.compareTo(second.left); }
  }

  // But sort by UID and the sortCharacter remember location has a sort character of 'a' and
  // product-id has a sort character of 'b' so the first record will be the location record!
  public static class SSSortComparator extends TTRawComparator {
    public int compare(TextPair first, TextPair second) {
      int lCompare = first.left.compareTo(second.left); if (lCompare == 0) return first.right.compareTo(second.right);
      else return lCompare;
    }
  }
}
```

MapReduce - Reducer

```
public void reduce(TextPair key, Iterable<Text> values, Context context)
        throws java.io.IOException, InterruptedException {

    String oldLocation = null;
    String currentLocation = null;

    StringBuilder sb = new StringBuilder();
    Visit item = null;

    for (Text value : values) {
        item = Visit.loadVisit(value.toString());
        currentLocation = item.getLocation().replace(" ", "_").replace("'", "");
        if (!currentLocation.equals(oldLocation)) {
            sb.append(currentLocation + " ");
            oldLocation = currentLocation;
        }
    }
    outputValue.set(sb.toString().trim());
    context.write(NullWritable.get(), outputValue);
}
```

Some results…

Table 4: Top 5 sequential patterns of length 2, 3 and 4 across places in Rome

Sequential pattern	Support
Colosseum → St. Peter's Basilica	9.07%
St. Peter's Basilica → Colosseum	7.72%
Colosseum → Roman Forum	5.28%
Colosseum → Pantheon	4.44%
Colosseum → Trevi Fountain	4.19%
Colosseum → Roman Forum → St. Peter's Basilica	4.4%
Vatican Museums → St. Peter's Basilica → Colosseum	3.9%
Colosseum → Trevi Fountain → St. Peter's Basilica	3.7%
Colosseum → Roman Forum → Pantheon	3.6%
Colosseum → Pantheon → St. Peter's Basilica	3.6%
Colosseum → Trevi Fountain → Pantheon → St. Peter's Basilica	0.64%
Colosseum → Roman Forum → Trevi Fountain → San St. Peter's Basilica	0.61%
Colosseum → Roman Forum → Piazza Venezia → Piazza di Spagna	0.58%
Colosseum → Roman Forum → Piazza Venezia → St. Peter's Basilica	0.58%
Colosseum → Roman Forum → Pantheon → St. Peter's Basilica	0.58%

References

- L. Belcastro, F. Marozzo, D. Talia, "Programming Models and Systems for Big Data Analysis". *International Journal of Parallel, Emergent and Distributed Systems*, 2018.
- L. Belcastro, F. Marozzo, D. Talia, P. Trunfio, "G-RoI: Automatic Region-of-Interest detection driven by geotagged social media data". *ACM Transactions on Knowledge Discovery from Data*, October 2017.
- E. Cesario, F. Marozzo, D. Talia, P. Trunfio, "SMA4TD: A Social Media Analysis Methodology for Trajectory Discovery in Large-Scale Events". *Online Social Networks and Media*, vol. 3-4, pp. 49-62, October 2017.
- A. Altomare, E. Cesario, C. Comito, F. Marozzo, D. Talia, "Trajectory Pattern Mining for Urban Computing in the Cloud". *Transactions on Parallel and Distributed Systems*, vol. 28, n. 2, pp. 586-599, 2017. *Note: ISSN:1045-9219.*
- D. Talia, P. Trunfio, F. Marozzo, Data Analysis in the Cloud, Elsevier, 2015. *Note: ISBN 978-0-12-802881-0.*

Questions?

Thank you for your attention!