# Unveiling Objects in Big Data

## Distance Matching

Supervisor: Prof. Fabio Porto

Prof. Dennis Shasha, Prof. Patrick Valduriez

By Amir Khatibi

Aug 2015

# Timeline

**2014** Mar | May | Jul | Sep | Nov | 2015 | Mar | May | Jul | Sep | Nov | 2016 **2016**

Today

Mar 2014
**Beginning of my master studies**

Sep 2014
**Presenting “Unveiling objects in Big Data” at 4th HOSCAR international workshop in Gramado, Brazil.**

Oct 2014
**Poster session "Unveiling objects in Big Data - using similarity approach", Khatibi, A., Porto, F. at Many Faces of Distances International workshop in UNICAMP, Campinas, Brazil.**

Nov 2015
**Patent submission “Seismic Pattern Queries” at EMC company.**

Feb 2015
**Working on paper “Unveiling Objects in Big Data- Model based approach”**

Mar 2015
**Patent “Seismic Pattern Queries” approved by United States Patent and Trademark Office**

Apr 2015
**Start implementing the “Unveiling Objects in Big Data- Model based approach” in Hadoop-MapReduce framework using HDFS**

Jun 2015
**Start working on “Unveiling Objects in Big Data- Distance Matching” professors Fabio Porto, Dennis Shasha, Patrick Valduriez**

Jul 2015
**Papers submission "Unveiling objects in Big Data – using model based approach", Khatibi, A., Porto, F., Pires, P., Valduriez, P, Shasha, D., to International Transactions in Operational Research journal (ITOR 2015)**

Aug 2015
**Implementing the 1st and 2nd version of "Distance Matching" using Python**

2016
**Master disertation**

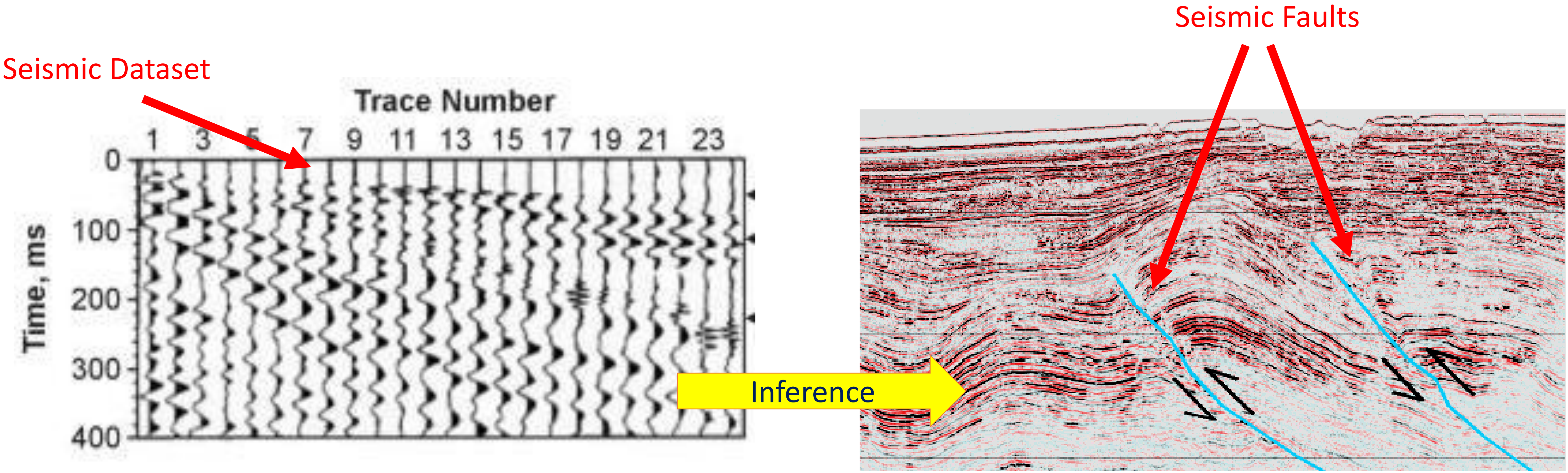
# Unveiling objects

- Data deluge produced by the Big Data phenomenon, blurs high-level objects amongst billions of dataset elements.

- A Sample Query or a Descriptive Language specifies the elements that shall compose higher-level objects, and a composition model.

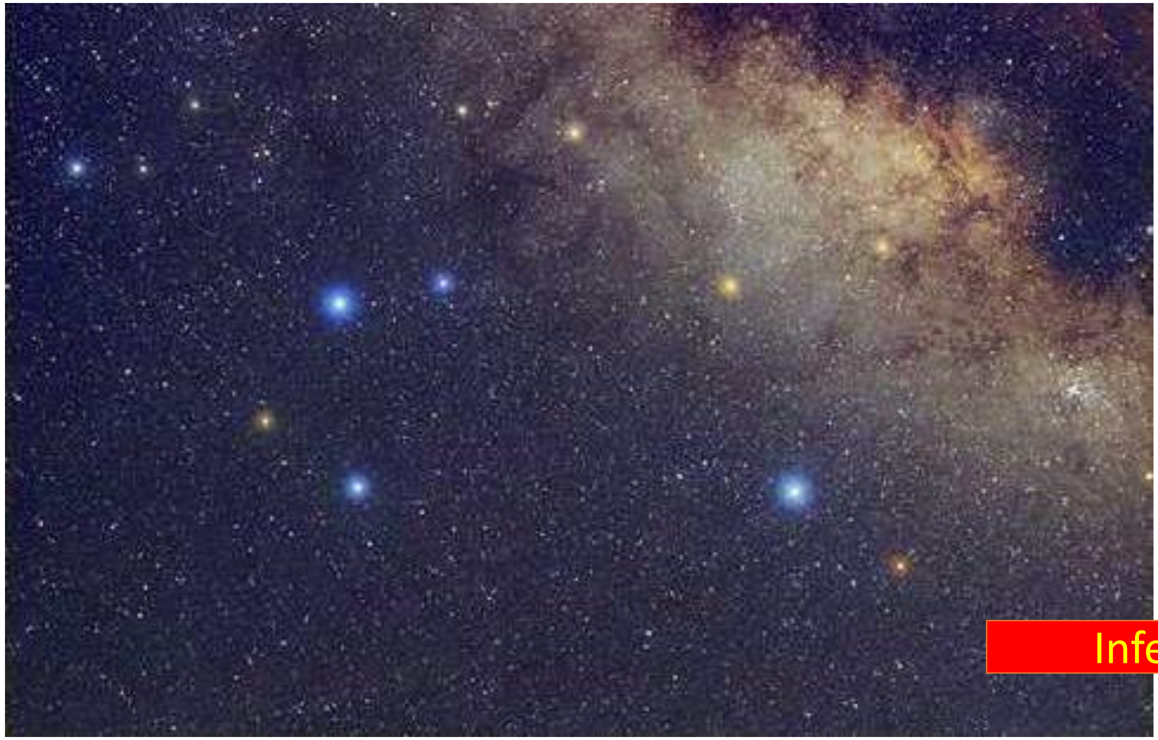
# Unveiling objects – Seismic Data

- The objects of interest are components built from the elements held by the target dataset. Those features maybe obtained from the dataset through a smart combination of low-level elements.



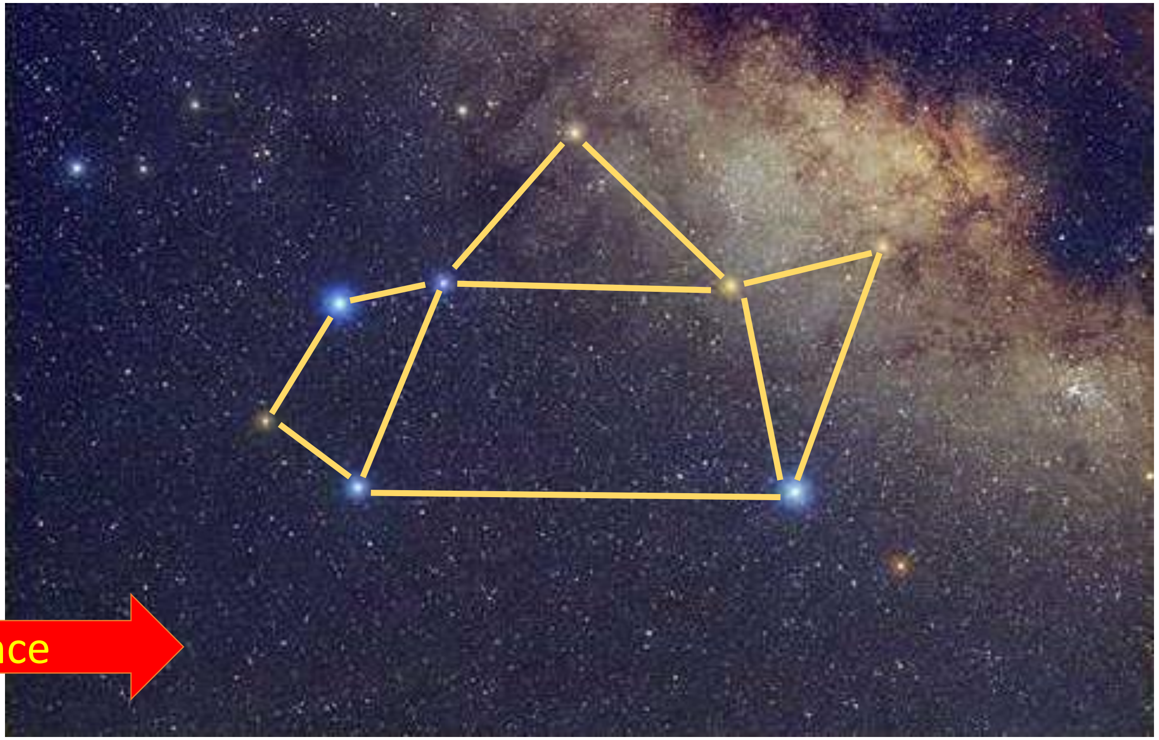
# Unveiling objects – Astronomy

low level objects: stars and their attributes



Inference

Higher level object: constellation



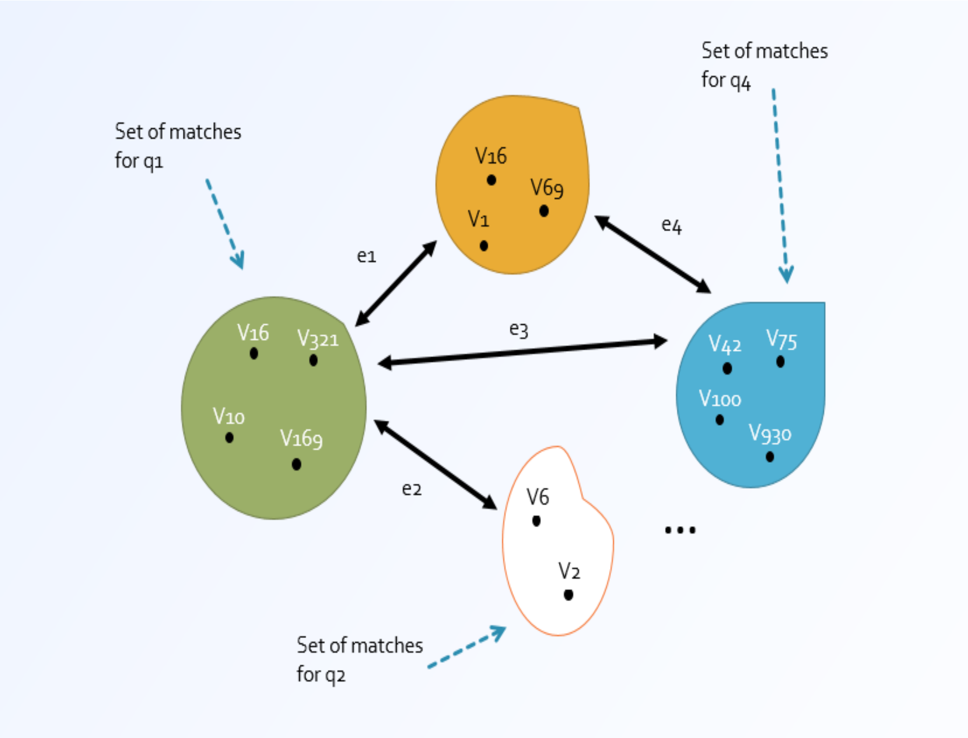
# Unveiling objects

- Distance queries are important in many areas like:
  - Astronomy catalogues
  - Seismic Data
  - Environmental sensor data

  …

- We look for efficient strategies to search for such elements in huge datasets

- In the case of Big data, processing the distributed data store in a parallelized environment is obvious.



# Astronomy application – sample query

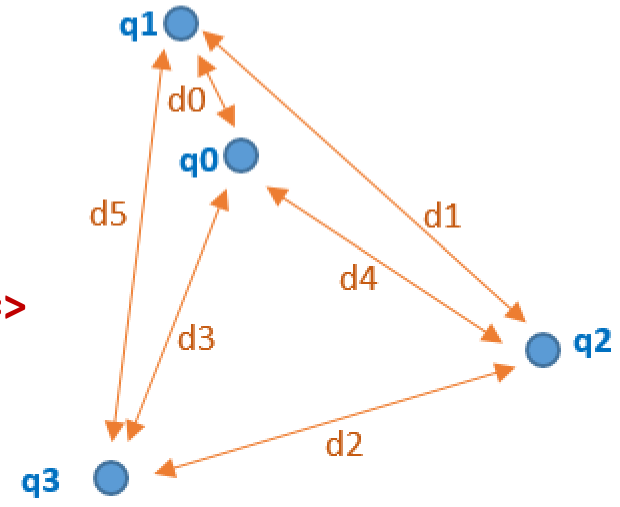
- Astronomy catalogue:

Cat (ra, dec, flux, photo-z, u, g, r, i, z,…)

**magnitudes of light emitted by an sky object**

**position of sky objects specifies by the values of their coordinate in right-ascension (ra) and declination (dec).**

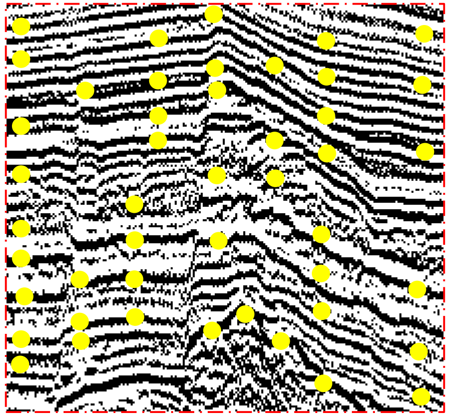
**A sample query =>**



# Pre-computing the Query

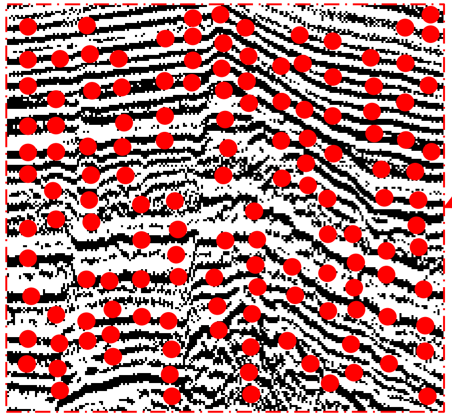
- Find the k most representative points in the query
  - Relieve us from many computations.

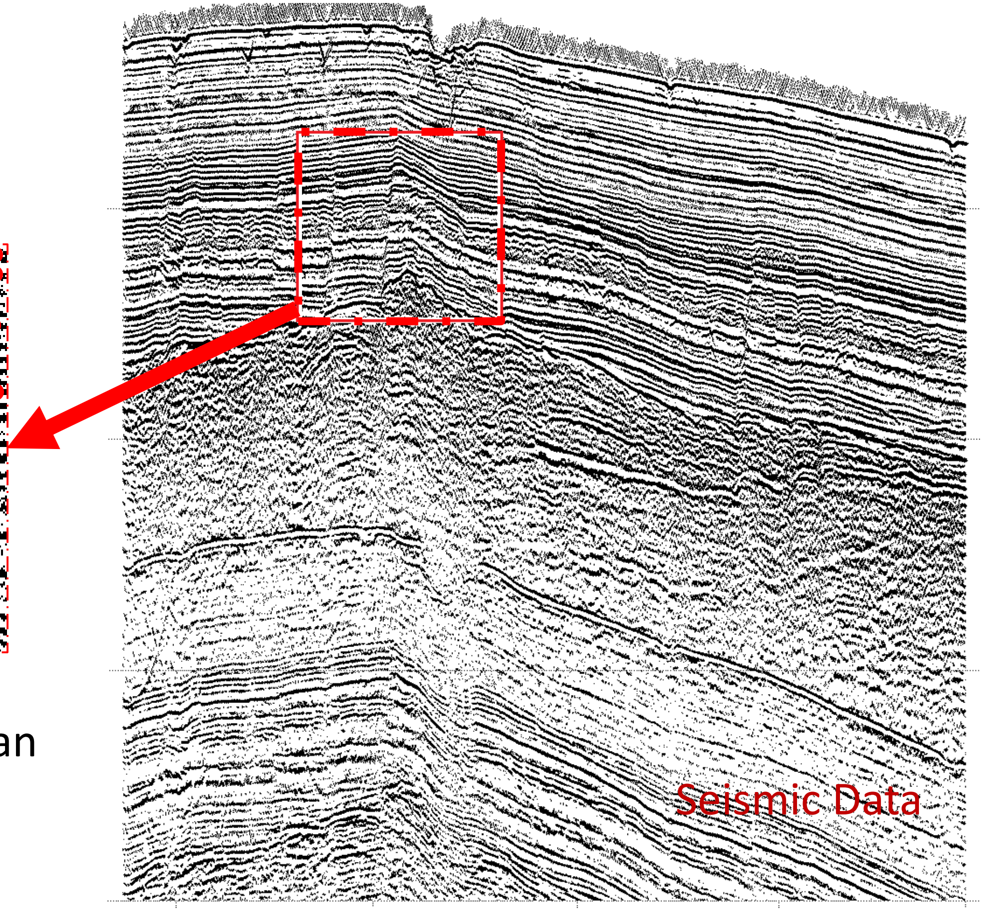
Refined query



Query Pre-computation

Geologist's query

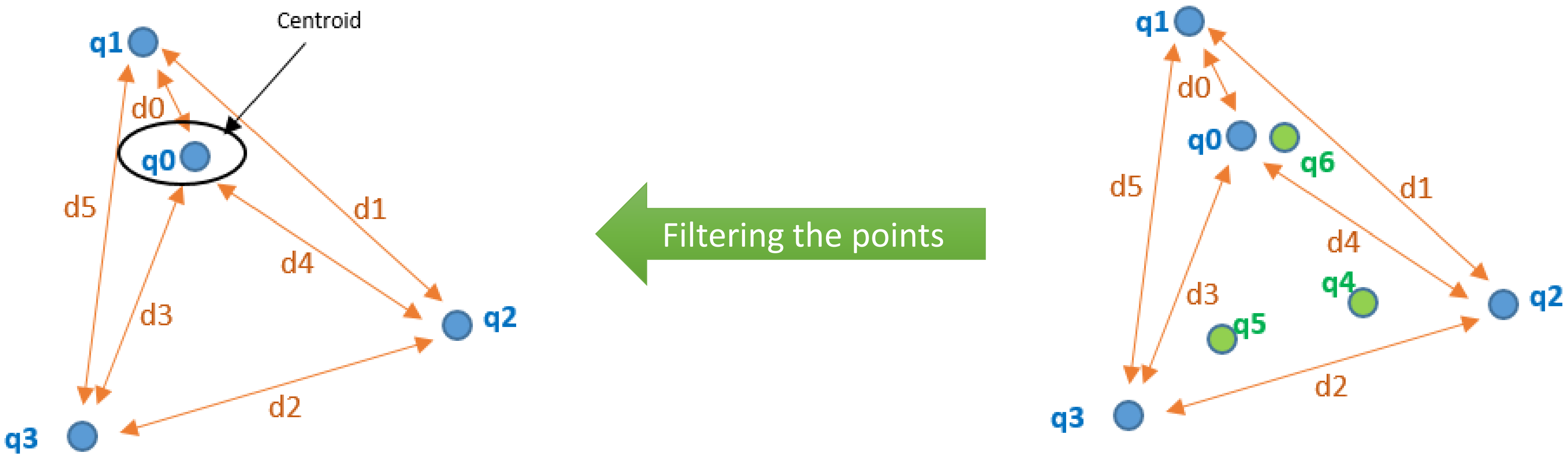





ex: In a query with 50% of points very similar to each other, we can avoid from 50% of the computations in whole.

# Pre-computing the Query

- Filter some small number of points that are far apart from each other.
  - Do the distance match for these points, firstly.
- Find the centroid point in the query
  - The nearest point to the Query's center of mass



# Centroid-based approach

In this approach, each star is a potential centroid of a shape. Initial matching can be applied between the query centroid and the potential centroid of a shape:

- Query_centroid_distances (centroid_id, <<d1,q_1 >, …,<dk,q_k >>)

Centroid's distances from other query elements

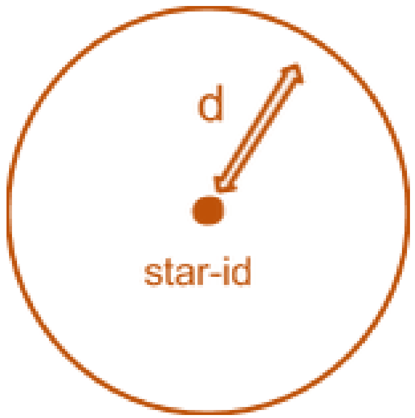
=> **Pre-computing the dataset**

**(centroid_id,<<d1,star-id_1,ra_1,dec_1>,… ,<dn,star-id_n,ra_n,dec_n>>)**

# Pre-Computing the Dataset

- Pre-compute the distances of every sky object from other objects in a specific boundary d-radius of it and store an ordered list of distances and points

- Using a spatial index like **PH-Tree** to quicken the execution time.

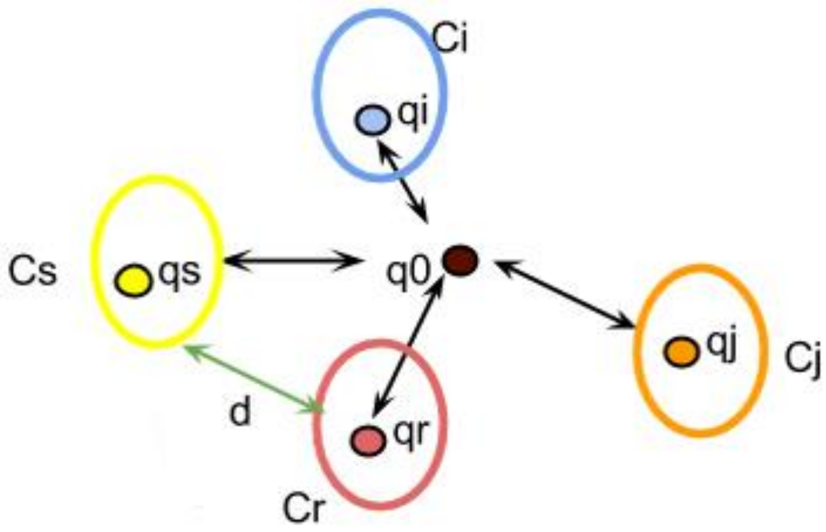


# Initial matching

- We look for candidate matches with respect to the query centroid and the result would be stored in the following relation. Every centroid could have a set of candidates for p-th element in the query.

**Cp (Bucket p)**

**(centroid_star_id, partner_star_id, Ra_partner, Dec_partner)**



# Pairwise distances

The second step is checking the pairwise distances in filtered centroids of a shape with respect to the query pairwise distances:

Element’s distances from other query elements

- Query_pairwise_distances (<<dij,q_i,q_j >>)

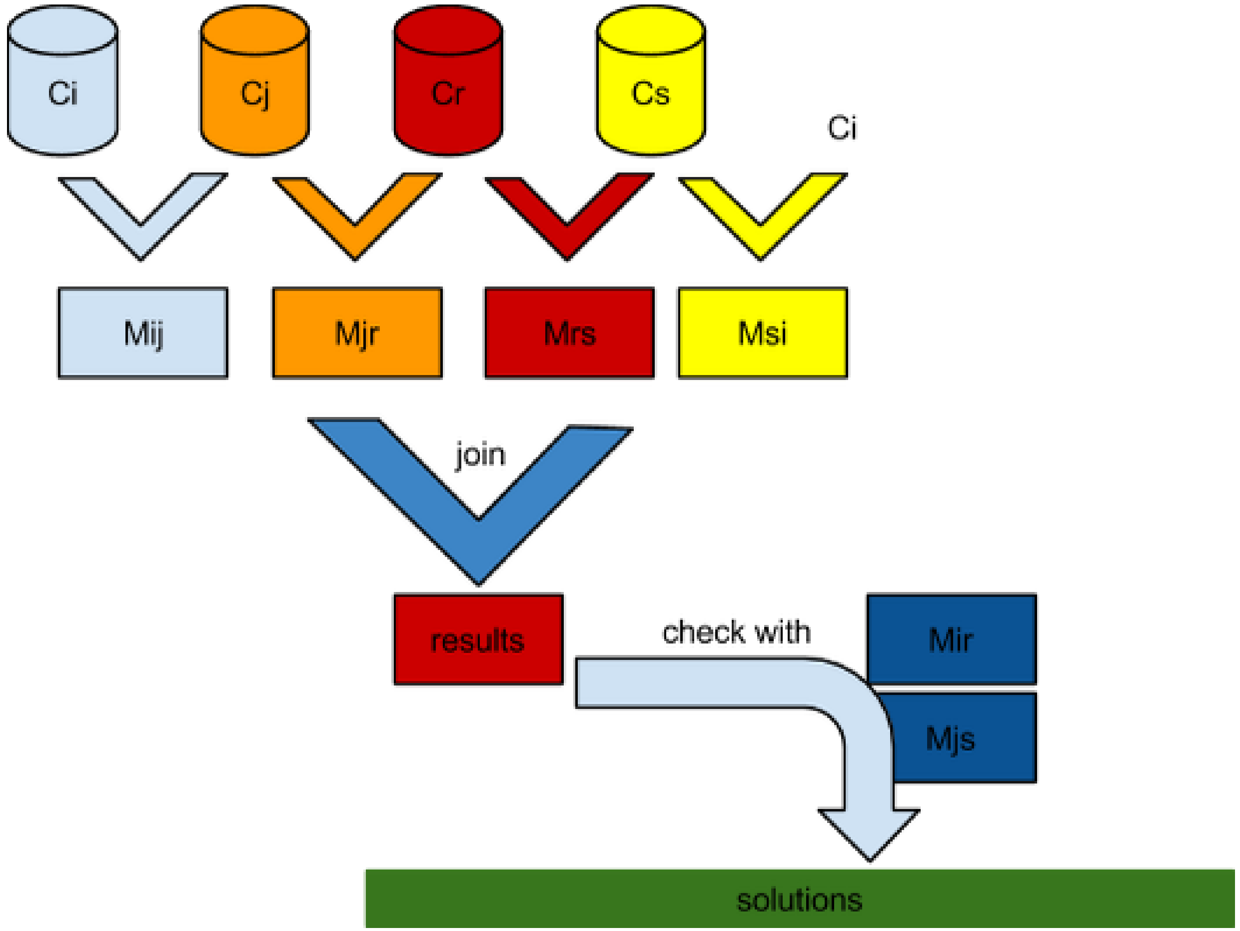
=> **Nested loops or Join Matrices**

# Matrices

- The idea of matrices aims to check the pairwise distances, partially.
- We create a set of matrices Mij

Mij(a,b) = 1 iff:

dist (star a in Ci, star b in Cj) similar to dist (qi, qj)



# Form the shapes

- According the density of intermediate results and the total number of solutions, we can do it differently:

- Nested Loops
  - Using in-memory variables and for loops
- Nested Joins in SQLite
  - Using Matrices and in-memory SQLite to join
- Matrix Multiplication
  - Using binary matrix multiplication to filter the candidate centroids

# Some Results

- Data-set size: **1000 sky objects**
- Pre-computing the dataset: **41 seconds**
- Average number of **neighbours** for each centroid: **170** (between 90 to 240)

(All the following results are only for the first 50 candidate centroids using **Python PL**)

<table>
<tr><th colspan="2" rowspan="2">Method</th><th>avg(|Cp|)= 10<br>Dense matrices<br>(lots of solutions)</th><th>avg(|Cp|)= 10<br>sparse matrices</th><th>avg(|Cp|)= 102<br>Dense matrices<br>(lots of solutions)</th><th>avg(|Cp|)= 102<br>sparse matrices</th></tr>
<tr><th>#solutions:<br>72,187</th><th>#solutions:<br>39</th><th>#solutions:<br>29,790,096</th><th>#solutions:<br>60</th></tr>
<tr><td>Nested loops</td><td>Without matrices</td><td>1.35 s</td><td>65 ms</td><td>550 s</td><td>2.4 s</td></tr>
<tr><td>MM Filtering</td><td>sparse matrix multiplication</td><td>130 ms</td><td>99 ms</td><td>5.3 s</td><td>3.9 s</td></tr>
<tr><td rowspan="2">Matrices and Sqlite</td><td>with indices</td><td>1.11 s</td><td>111 ms</td><td>?more than 20 min</td><td>3.9 s</td></tr>
<tr><td>without indices</td><td>297 ms</td><td>105 ms</td><td>51 s</td><td>3.7 s</td></tr>
<tr><td colspan="2">Prediction</td><td>MM>NL</td><td>MM<NL</td><td>MM>NL</td><td>MM<NL</td></tr>
<tr><td colspan="2">Conclusion</td><td colspan="4">Totally True</td></tr>
</table>

Thank you for your attention