

Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion
Greece
May 29th – June 2nd
2011



[2.1] Introduction to SPARQL

Emanuele Della Valle

emanuele.dellavalle@polimi.it

<http://emanueledellavalle.org>

- This work is licensed under the Creative Commons Attribution 3.0 Unported License.
- **Your are free:**
 -  **to Share** — to copy, distribute and transmit the work
 -  **to Remix** — to adapt the work
- **Under the following conditions**
 -  **Attribution** — You must attribute the work by inserting
 - “© streamreasoning.org” at the end of each reused slide
 - a credits slide stating
 - These slides are partially based on
“Streaming Reasoning for Linked Data 2011”
by Emanuele Della Valle and Jeff Z. Pan
<http://streamreasoning.org/sr4ld2011>
- To view a copy of this license, visit
<http://creativecommons.org/licenses/by/3.0/>

- Introduction
 - What is SPARQL?
 - Why SPARQL
 - Anatomy of a SPARQL query
- Writing simple queries
 - Graph Pattern Syntax
 - Matching RDF literals
 - Value Tests
- More sophisticated Graphs Patterns
 - OPTIONAL
 - UNION
- Results Forms
- Advance Features
 - Aggregates
 - Sub-queries
- Reasoning and Query Answering

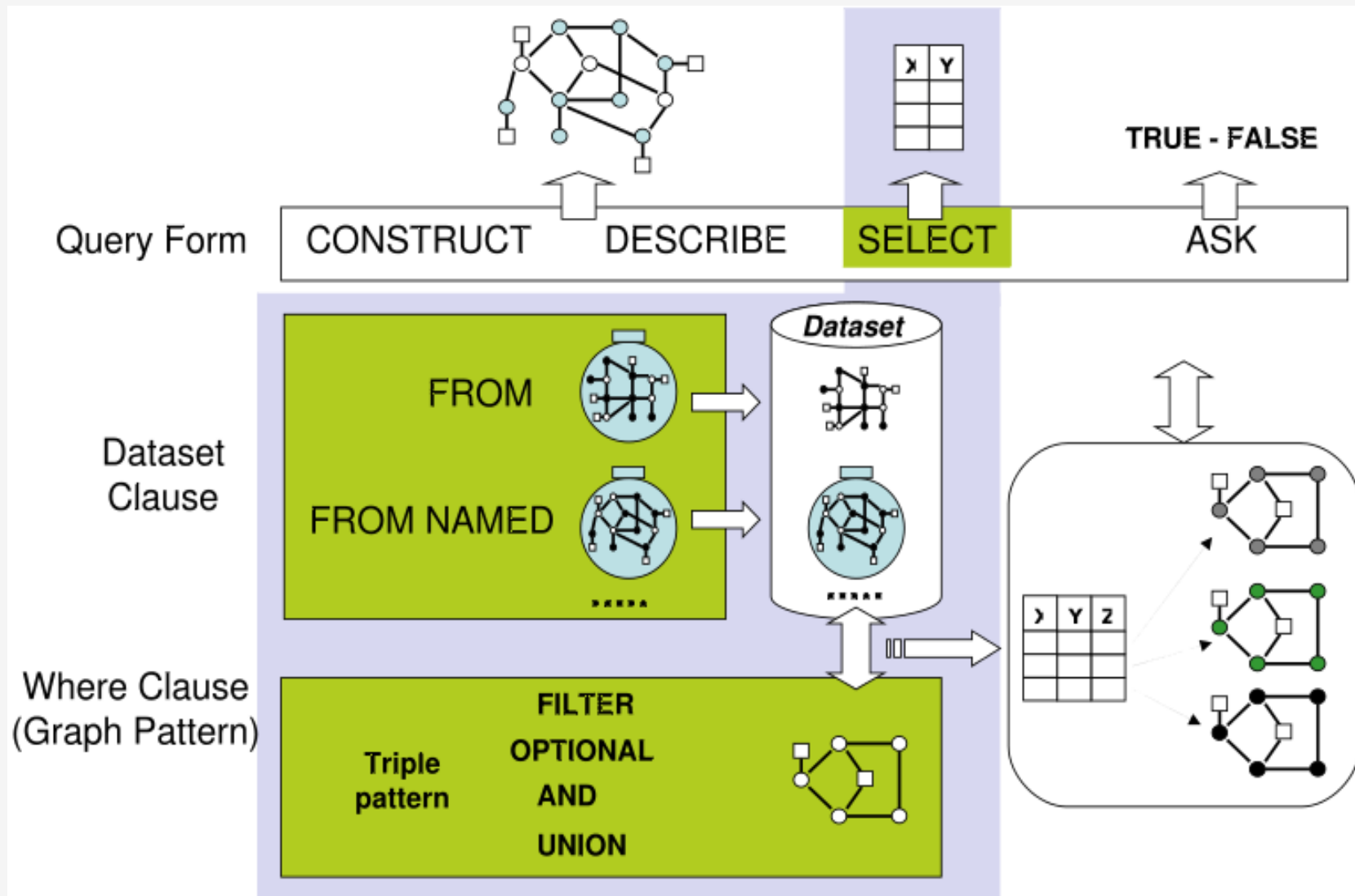
- SPARQL
 - is the query language of the Semantic Web
 - stays for **SPARQL Protocol** and **RDF Query Language**



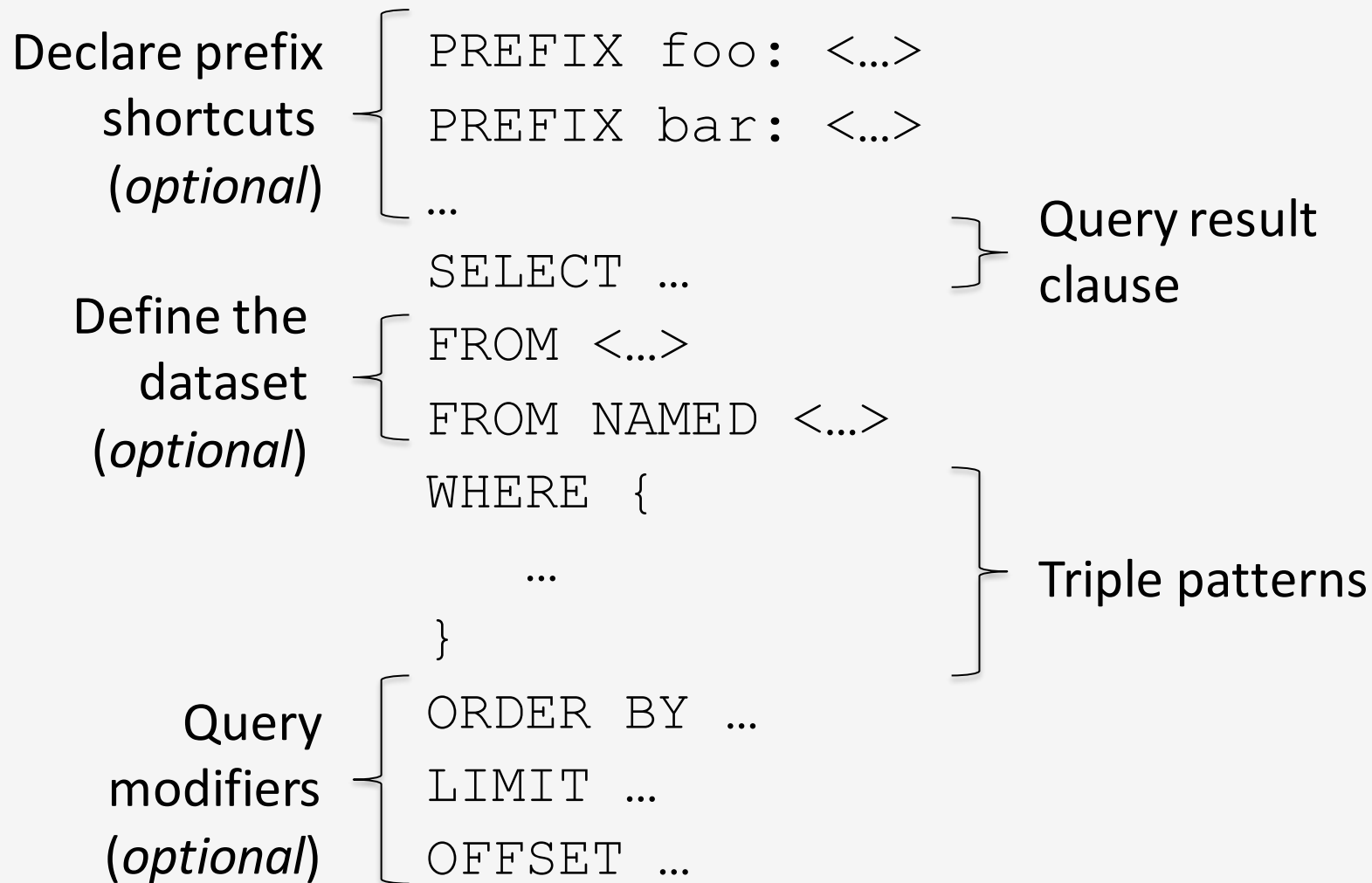
- A Query Language ...:
Find drugs in DBpedia:
- ```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
SELECT ?poi
WHERE { ?poi a sr:NamedPlace . }
```
- ... and a Protocol.  

```
http://lod.openlinksw.com/sparql?&query=PREFIX+sr%3A+%3Chttp%3A%2F%2Fwww.streamreasoning.org%2Fsr4ld2011%2Fonto%2F%3E%0D%0ASELECT+%3Fpoi+WHERE+{+%3Fpoi+a+sr%3ANamedPlace+.+}
```

- SPARQL let us
  - Pull values from structured and semi-structured data represented in RDF
  - Explore RDF data by querying unknown relationships
  - Perform complex joins of disparate RDF repositories in a single query
  - Transform RDF data from one vocabulary to another
  - Develop higher-level cross-platform application



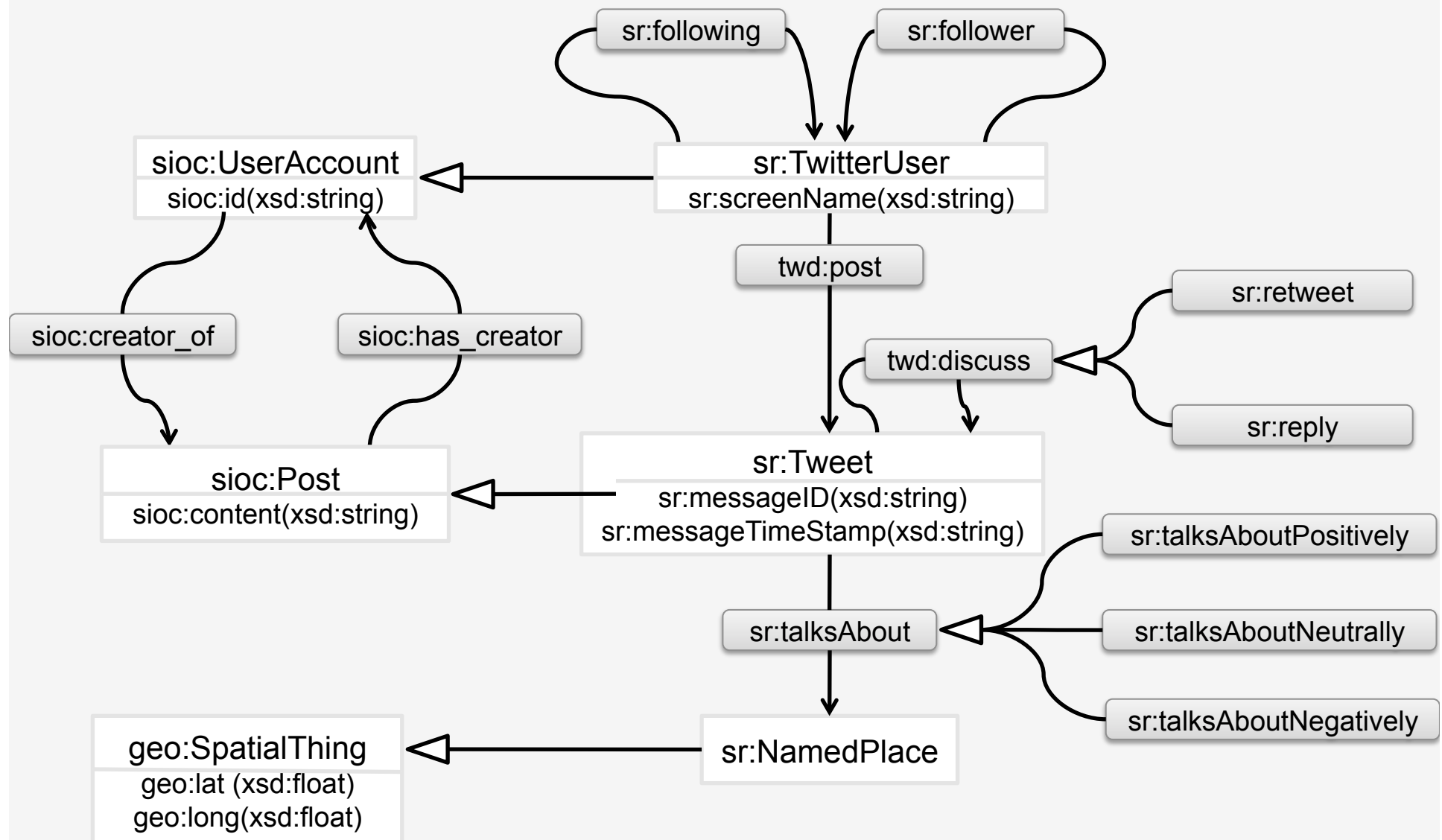
Source: Pérez, Arenas and Gutierrez, Chapter 1: On the Semantics of SPARQL, Semantic Web Information Management: A Model Based Perspective, Springer 2010



- Turtle-like: URIs, QNames, literals, convenience syntax.
- Adds variables to get basic patterns
  - ?var
  - Variable names are a subset of NCNames (no "-" or ".")
- E.g.,
  - simple
    - `?poi a sr:NamedPlace .`
  - complex
    - `?poi a geo:NamedPlace .`
    - `?poi skos:subject ?category .`
- Adds
  - OPTIONAL to cope with semi-structured nature of RDF
  - FILTER to select solution according to some criteria
  - UNION operator to get complex patterns

# Test data: Data Model

Heraklion  
Greece  
May 29<sup>th</sup> – June 2<sup>nd</sup>  
2011



## ■ Data

```
@prefix sr:<http://www.streamreasoning.org/sr4ld2011/onto#>.

sr:LaScala a sr:NamedPlace .
sr:GalleriaVittorioEmanueleII a sr:NamedPlace .
sr:Duomo a sr:NamedPlace .
```

## ■ Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>

SELECT ?poi
WHERE { ?poi a sr:NamedPlace . }
```

## ■ Results

**?poi**<http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII><http://www.streamreasoning.org/sr4ld2011/data#LaScala><http://www.streamreasoning.org/sr4ld2011/data#Duomo>

- *Matches the graph* means find a set of bindings such that the substitution of variables for values **creates a triple** that is in the set of triples making up the graph.
- Solution 1:
  - variable *poi* has value `sr:GalleriaVittorioEmanueleII`
  - Triple `sr:GalleriaVittorioEmanueleII a sr:NamedPlace .` is in the graph.
- Solution 2:
  - variable *poi* has value `sr:LaScala`
  - Triple `sr:LaScala a sr:NamedPlace .` is in the graph.
- Solution 3:
  - variable *poi* has value `sr:Duomo`
  - Triple `sr:Duomo a sr:NamedPlace .` is in the graph.
- No order of solutions in this query.

## ■ Query

```

PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>

SELECT ?poi ?category
WHERE { ?poi a geo:NamedPlace ;
 skos:subject ?category . }

```

## ■ Results

| ?drug                                                                                                                                                           | ?category                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII">http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII</a> | <a href="http://dbpedia.org/resource/Category:Pedestrian_streets_in_Italy">http://dbpedia.org/resource/Category:Pedestrian_streets_in_Italy</a>             |
| <a href="http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII">http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII</a> | <a href="http://dbpedia.org/resource/Category:Buildings_and_structures_in_Milan">http://dbpedia.org/resource/Category:Buildings_and_structures_in_Milan</a> |
| <a href="http://www.streamreasoning.org/sr4ld2011/data#LaScala">http://www.streamreasoning.org/sr4ld2011/data#LaScala</a>                                       | <a href="http://dbpedia.org/resource/Category:Opera_houses_in_Italy">http://dbpedia.org/resource/Category:Opera_houses_in_Italy</a>                         |
| <a href="http://www.streamreasoning.org/sr4ld2011/data#Duomo">http://www.streamreasoning.org/sr4ld2011/data#Duomo</a>                                           | <a href="http://dbpedia.org/class/yago/ChurchesInMilan">http://dbpedia.org/class/yago/ChurchesInMilan</a>                                                   |
| ...                                                                                                                                                             | ...                                                                                                                                                         |

- A basic Patter is a set of triple patterns, all of which **must be matched**.
- In this case *matches the graph* means find a set of bindings such that the substitution of variables for values **creates a subgraph** that is in the set of triples making up the graph.

## ■ Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>

SELECT ?poi
WHERE { ?poi sr:name "Duomo". }
```

## ■ Results

**?poi**

<http://www.streamreasoning.org/sr4ld2011/data#Duomo>

## ■ Alert!

- It may return 0 results if the literal have a language tag
  - E.g., if data contains only the triple  
`sr:Duomo sr:name "Duomo"@it .`
- To obtain results also add the language tag to the triple pattern
  - E.g, `?poi sr:name "Duomo"@it.`

- As in the case of language tags, if the literals are typed (i.e., "3.14"^^xsd:float), they do not match if they are not given explicitly.

- Query

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>

SELECT ?poi
WHERE { ?poi a sr:NamedPlace ;
 geo:lat "45.46416854858398"^^xsd:float ;
 geo:long "9.191389083862305"^^xsd:float . }
```

- Results

**?poi**

<http://www.streamreasoning.org/sr4ld2011/data#Duomo>

- SPARQL allows restricting solutions by applying the FILTER clause.
- An RDF term bound to a variable appears in the results if the FILTER expression, applied to the term, evaluates to TRUE.

- Query

```
PREFIX geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
SELECT ?poi ?lat ?log
WHERE {
 ?poi geo:lat ?lat ; geo:long ?long .
 FILTER (
 ?lat > "45.46"^^xsd:float && ?lat < "45.47"^^xsd:float &&
 ?long > "9.18"^^xsd:float && ?long < "9.20"^^xsd:float)
}
```

- Results

**?poi**<http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII><http://www.streamreasoning.org/sr4ld2011/data#LaScala><http://www.streamreasoning.org/sr4ld2011/data#Duomo>

- SPARQL FILTERs allows also restricting values of strings using the `regex()`

- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?poi ?c
WHERE { ?poi rdfs:comment ?c .
 FILTER(regex(?c, "glass-vaulted arcades", "i"))}
```

- Results

| ?poi                                                                                                                                                            | ?c                                                                                                                                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII">http://www.streamreasoning.org/sr4ld2011/data#GalleriaVittorioEmanueleII</a> | The Galleria Vittorio Emanuele II is a covered double arcade formed of two <b>glass-vaulted arcades</b> at right angles intersecting in an octagon, prominently sited on the northern side of the Piazza del Duomo in Milan, and connects to the Piazza della Scala. |

- Notation for value comparison:  $<$ ,  $>$ ,  $=$ ,  $\leq$ ,  $\geq$  and  $\neq$
- Test functions
  - Check if a variable is bound: BOUND
  - Check the type of resource bound: isIRI, isBLANK, isLITERAL
- Accessing accessories: LANG, DATATYPE
- Logic operators:  $||$  and  $\&\&$
- Comparing strings: REGEX, langMatches
- Constructor functions: bool, dbl, flt, dec, int, dT, str, IRI
- Extensible Value Testing
  - E.g., FILTER ( aGeo:distance(?axLoc, ?ayLoc, ?bxLoc, ?byLoc)  $<$  10 ) .
  - (see <http://www.w3.org/TR/rdf-sparql-query/#extensionFunctions> )

- Find all schools within a 5km radius around a specific location, and for each school find coffeeshops that are closer than 1km.
- ```
PREFIX lgdo: <http://linkedgeodata.org/ontology/>
SELECT ?schoolname ?schoolgeo ?coffeeshopname ?coffeeshopgeo
WHERE {
  ?school      a                lgdo:School .
  ?school      geo:geometry     ?schoolgeo .
  ?school      rdfs:label       ?schoolname .
  ?coffeeshop  a                lgdo:CoffeeShop .
  ?coffeeshop  geo:geometry     ?coffeeshopgeo .
  ?coffeeshop  rdfs:label       ?coffeeshopname .
  FILTER(
    bif:st_intersects(
      ?schoolgeo,bif:st_point (4.892222, 52.373056), 5) &&
    bif:st_intersects(?coffeeshopgeo, ?schoolgeo, 1)
  ) .
}
```
- [Click here for query results](#) on a Virtuoso endpoint used by LinkedGeoData project.

- Signature
 - `st_intersects(g1, g2, prec)`
- Parameters
 - `g1` – The first geometry.
 - `g2` – The second geometry.
 - `prec` – A tolerance for the matching in units of linear distance appropriate to the srid. Default is 0.
- Description
 - Returns intersects between two geometries. If `prec` is supplied, this is a tolerance for the matching in units of linear distance appropriate to the srid. Both geometries should have the same srid. `st_intersects` is true if there is at least one point in common.

- RDF is "semi structured" and has no integrity constraints
- SPARQL addresses this issue with
 - **Group patterns** match if all subpatterns match and all constraints are satisfied
 - In SPARQL syntax, groups are { ... }
 - **OPTIONAL graph patterns** accommodate the need to add information to a result but without the query failing just because some information is missing.
 - In SPARQL syntax, OPTIONAL { ... }
 - **UNION graph patterns** allows to match alternatives
 - In SPARQL syntax, { ... } UNION { ... }

- **OPTIONAL graph patterns** accommodate the need to add information to a result but without the query failing just because some information is missing.

- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX srd: <http://www.streamreasoning.org/sr4ld2011/data#>
SELECT ?t ?p ?poi
WHERE {
    ?t a sr:Tweet .
    ?poi a sr:NamedPlace .
    OPTIONAL { ?t ?p ?poi }
}
```

- Results

?t	?p	?poi
_:post1	sr:talksAbout	srd:GalleriaVittorioEmanueleII
_:post1	sr:talksAboutPositively	srd:LaScala
_:post1	null	srd:Duomo

- **UNION graph patterns** allows to match alternatives
- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX srd: <http://www.streamreasoning.org/sr4ld2011/data#>
SELECT ?t ?poi
WHERE {
  ?t a sr:Tweet .
  ?poi a sr:NamedPlace .
  { ?t sr:talksAbout ?poi }
  UNION
  { ?t sr:talksAboutPositively ?poi }
}
```

- Results

?t	?poi
_:post1	srd:GalleriaVittorioEmanueleII
_:post1	srd:LaScala

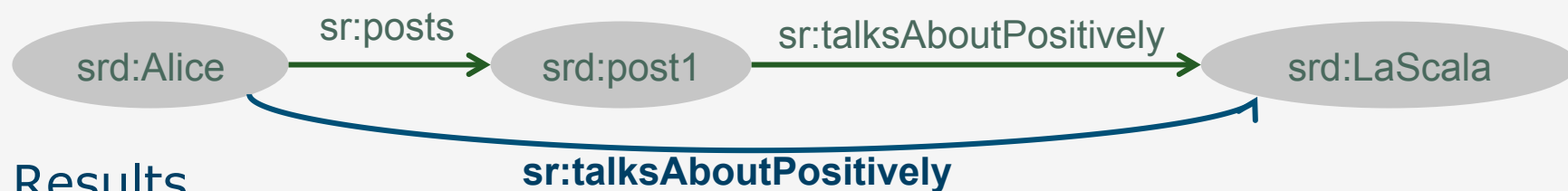
- Besides selecting tables of values, SPARQL allows three other types of queries:
 - ASK - returns a boolean answering, does the query have any results?
 - CONSTRUCT - uses variable bindings to return new RDF triples
 - DESCRIBE - returns server-determined RDF about the queried resources
- SELECT and ASK results can be returned as XML or JSON.
- CONSTRUCT and DESCRIBE results can be returned via any RDF serialization (e.g. RDF/XML or Turtle).

■ Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
CONSTRUCT { ?u sr:talksAboutPositively ?poi }
WHERE {
  ?u a sr:TwitterUser .
  ?u sr:posts ?t .
  ?t sr:talksAboutPositively ?poi .
  ?poi a sr:NamedPlace .
}
```

■ Meaning

- it requires to compute a property chain



■ Results

```
@prefix sr:<http://www.streamreasoning.org/sr4ld2011/onto#> .
srd:Alice sr:talksAboutPositively srd:LaScala .
```

- How many points of interest are in the dataset?
- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT (count(?poi) AS ?numberOfPOI)
WHERE {
    ?poi a sr:NamedPlace .}
```

- How many points of interest are in each category?
- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category (count(?poi) AS ?numberOfPOI)
WHERE {
    ?poi a sr:NamedPlace .
    ?poi skos:subject ?category .
}
GROUP BY ?category
```

- NOTES
 - The available built-ins are SUM, AVG, COUNT, MIN, MAX
 - A DISTINCT clause can be use to avoid processing duplicates

- Which are the categories of points of interest in which the points of interest are positively discussed in more than 100 tweets?
- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category
WHERE {
    ?poi a sr:NamedPlace .
    ?poi skos:subject ?category .
    ?tweet sr:talksAboutPositively ?poi .
}
GROUP BY ?category
HAVING (count(?tweet) > 100)
```

- Which are the categories of points of interest in which the points of interest are positively discussed in more than 100 tweets? How many points of interest does each category contain?

- Query

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category (count(?poi) AS ?numberOfPOI)
WHERE {
    ?poi a sr:NamedPlace .
    ?poi skos:subject ?category .
    ?tweet sr:talksAboutPositively ?poi .
}
GROUP BY ?category
HAVING (count(?tweet) > 100)
```

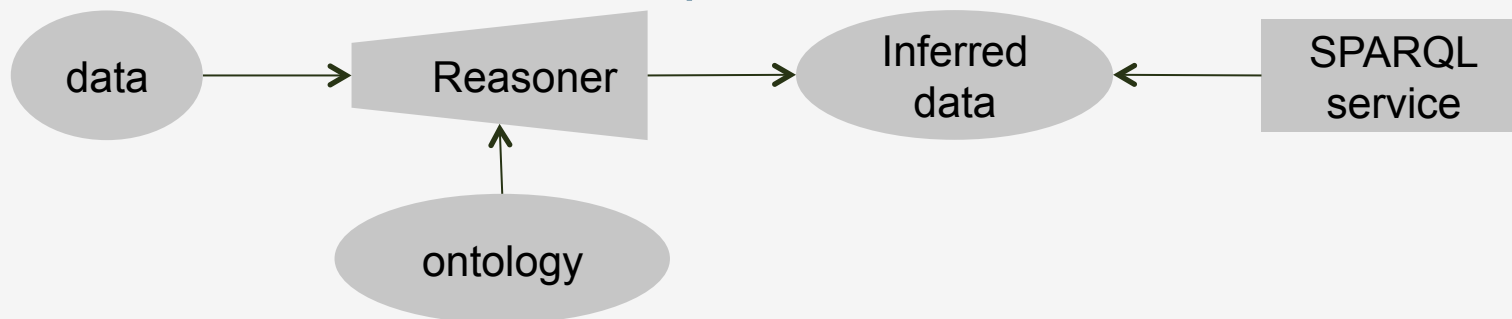
- Any group pattern can be a sub-query
- E.g., which are the categories of points of interest in which **each point of interest is positively discussed** in more than 10 tweets? How many points of interest does each category contain?
- ```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?category (count(?poi) AS ?numberOfPOI)
WHERE {
 ?poi a sr:NamedPlace .
 ?poi skos:subject ?category .
 {
 SELECT ?poi
 WHERE {
 ?poi a sr:NamedPlace .
 ?tweet sr:talksAboutPositively ?poi .
 }
 GROUP BY ?poi
 HAVING (count(?tweet) > 10)
 }
} GROUP BY ?category
```

- When you need to output an RDF graph that contains the results of a query that uses aggregates, you need to use sub-queries.
- Output the points of interest in each category as an RDF graph
- ```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
CONSTRUCT { ?category sr:countedPOIs ?numberOfPOIs }
WHERE {
  {
    SELECT ?category (count(?poi) AS ?numberOfPOIs)
    WHERE {
      ?poi a sr:NamedPlace .
      ?poi skos:subject ?category .
    }
    GROUP BY ?category
  }
}
```

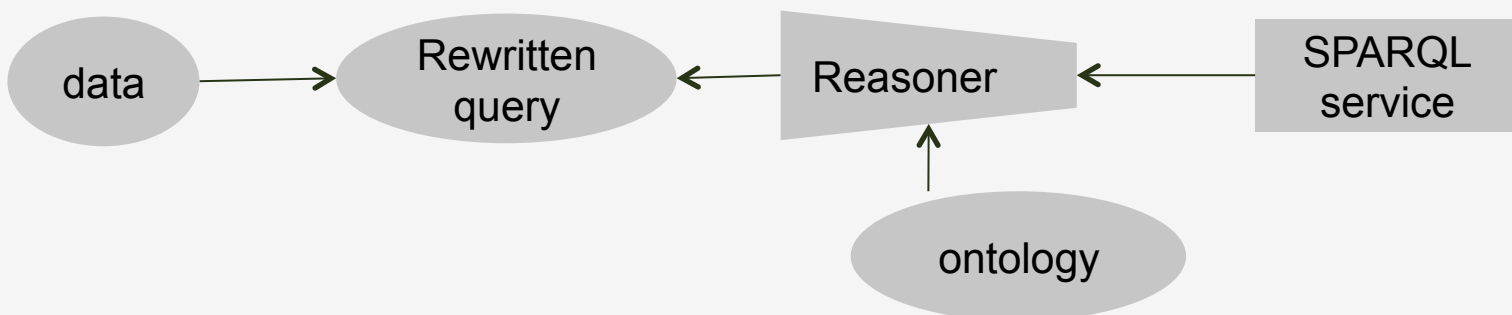
- SPARQL alone cannot answer queries that require reasoning



- but a reasoner can be exposed as a SPARQL service.



- Or a query can be rewritten in order to incorporate the ontology



- SPARQL Frequently Asked Questions
 - <http://thefigtrees.net/lee/sw/sparql-faq>
- SPARQL implementations - community maintained list of open-source and commercial SPARQL engines
 - <http://esw.w3.org/topic/SparqlImplementations>
- Public SPARQL endpoints - community maintained list
 - <http://esw.w3.org/topic/SparqlEndpoints>
- SPARQL extensions - collection of SPARQL extensions implemented in various SPARQL engines
 - <http://esw.w3.org/topic/SPARQL/Extensions>

- When cutting and pasting the SPARQL query from the slides into a query textbox you may get:

```
37000 Error SP030: SPARQL compiler, line 0:  
Invalid character in SPARQL expression at ' '
```

- A workaround is changing all (so called) *white spaces* in " "
 - e.g., in MS Word replace "^w" with " "

Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion
Greece
May 29th – June 2nd
2011



[2.2] Stream Reasoning

Emanuele Della Valle

emanuele.dellavalle@polimi.it

<http://emanueledellavalle.org>

Agenda

Heraklion
Greece
May 29th – June 2nd
2011



- Motivation
- Running Example
- Background
- Concept
- Achievements

- Sensor networks, ...



- traffic engineering, ...



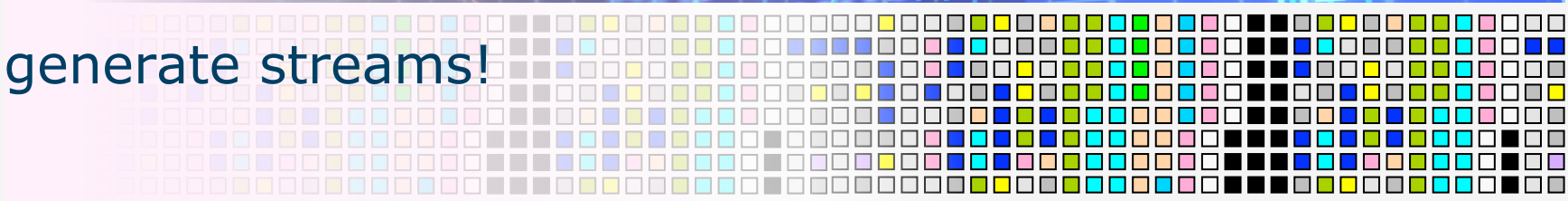
- social networking, ...



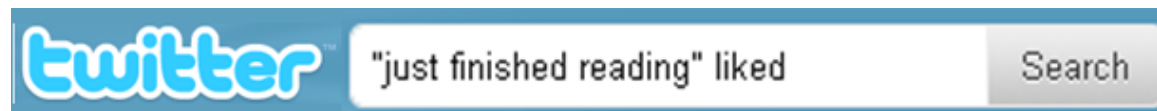
- financial markets, ...



- generate streams!



- Streams are appearing more and more often on the Web in sites that distribute and present information in real-time streams.
- Checkout <http://activitystrea.ms/> for a standard API
- E.g.



saluteDAVE | **just finished reading** "The World Without End" by Ken Follett. I **liked** it albeit his first book "The Pillars of the Earth" was the best.
36 minutes ago from web



jayneb **just finished reading** "the widow's season." **liked** it a lot more after i had time to think about it than while reading it.
about 14 hours ago from web



stina1123 **just finished reading** The Carrie Diaries...very predictable but i **liked** it...makes me want to watch sex and the city now!
about 21 hours ago from mobile web

- Which topics have my close friends discussed **in the last hour**?
- Which book is my friend likely to read **next**?
- What impact have I been creating with my tweets **in the last day**?
- ...
- <query> ... **<period of time>** ?

- Making sense
 - in **real time**
 - of *gigantic* and inevitably *noisy data streams*
 - in order to **support the decision process** of extremely *large numbers of concurrent user*

What are data streams anyway?

- Formally:
 - *Data streams are unbounded sequences of time-varying data elements*

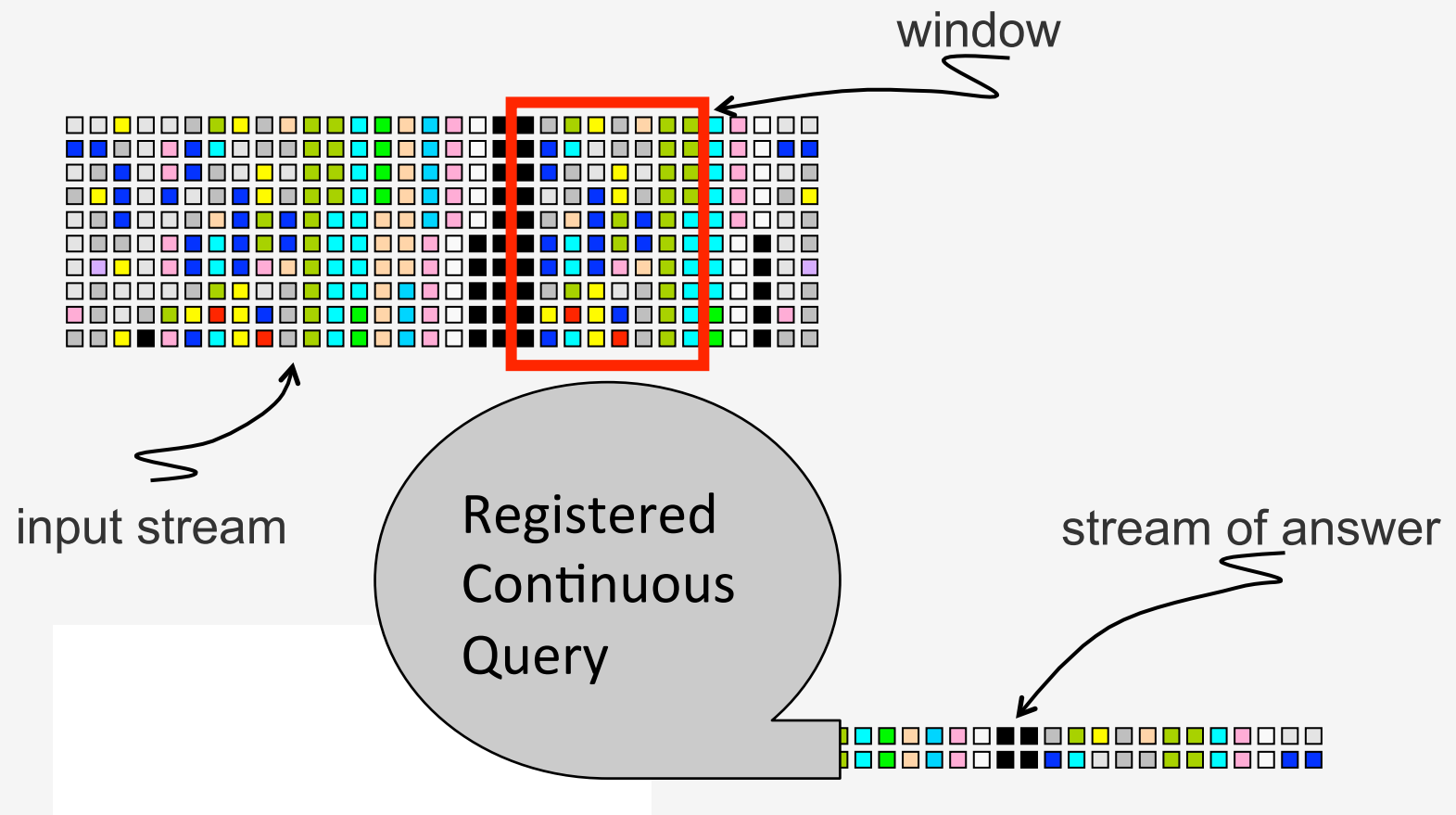


- Less formally:
 - an (almost) “continuous” flow of information
 - with the recent information being more relevant as it describes the current state of a dynamic system

- Processing data streams in the space of **one-time semantics** is difficult because of the very nature of the underlying data
- Innovative* assumption: **continuous semantics!**
 - **streams can be consumed on the fly** rather than being stored forever and
 - **queries are registered** and continuously produce answers

* This innovation arose in DB community in '90s

- **Continuous queries registered over streams that are observed trough windows**



■ Research Prototypes

- Amazon/Cougar (Cornell) – sensors
- Aurora (Brown/MIT) – sensor monitoring, dataflow
- Gigascope: AT&T Labs – Network Monitoring
- Hancock (AT&T) – Telecom streams
- Niagara (OGI/Wisconsin) – Internet DBs & XML
- OpenCQ (Georgia) – triggers, view maintenance
- Stream (Stanford) – general-purpose DSMS
- Stream Mill (UCLA) - power & extensibility
- Tapestry (Xerox) – publish/subscribe filtering
- Telegraph (Berkeley) – adaptive engine for sensors
- Tribeca (Bellcore) – network monitoring

■ High-tech startups

- Streambase, Coral8, Apama, Truviso

■ Major DBMS vendors are all adding stream extensions as well

- Oracle http://www.oracle.com/technology/products/dataint/htdocs/streams_fo.html
- DB2 <http://www.eweek.com/c/a/Database/IBM-DB2-Turns-25-and-Prepares-for-New-Life/>

- The Semantic Web, the Web of Data **is doing fine**
 - RDF, RDF Schema, SPARQL, OWL, RIF
 - well understood theory,
 - rapid increase in scalability
- **BUT it pretends that the world is static**
or at best a low change rate
both in change-volume and change-frequency
 - ontology versioning
 - belief revision
 - time stamps on named graphs
- It sticks to the traditional **one-time semantics**

- Idea origination
 - Can **continuous semantics** be ported to **reasoning**?
 - This is an **unexplored yet high impact** research area!
- Stream Reasoning
 - *Logical reasoning in real time on gigantic and inevitably noisy data streams in order to support the decision process of extremely large numbers of concurrent users.*
 - S. Ceri, E. Della Valle, F. van Harmelen and H. Stuckenschmidt, 2010
- Note: making sense of streams necessarily requires processing them against rich background knowledge

- Relation with data-stream systems
 - Just as RDF relates to data-base systems?
- Query languages for semantic streams
 - Just as SPARQL for RDF but with continuous semantics?
- Reasoning on Streams
 - Formal representations for stream reasoning
 - Notions of soundness and completeness
 - Efficiency
 - Scalability
- Dealing with incomplete & noisy data
 - Even more so than on the current Web of Data
- Distributed and parallel processing
 - Streams are parallel in nature

- We investigated
 - **RDF streams**
 - the natural extension of the RDF data model to the new continuous scenario and
 - Continuous SPARQL (or simply **C-SPARQL**)
 - the extension of SPARQL for querying RDF streams.
 - **Efficient incremental updates of deductive closures**
 - specifically considering the nature of data streams
 - **Effective inductive stream reasoning** (joint work with Siemens - Munich)
 - **Architecture of a Stream Reasoner**
 - See paper in IEEE IS special issue on Social Media Analytics
- Session 2.3
- Session 3.3
- Not covered in this tutorial

[IEEE-IS2009] Emanuele Della Valle, Stefano Ceri, Frank van Harmelen, Dieter Fensel It's a Streaming World! Reasoning upon Rapidly Changing Information. IEEE Intelligent Systems 24(6): 83-89 (2009)

[IEEE-IS2010] Davide Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, Yi Huang, Volker Tresp, Achim Rettinger, Hendrik Wermser, "Deductive and Inductive Stream Reasoning for Semantic Social Media Analytics," IEEE Intelligent Systems, 30 Aug. 2010.

Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion
Greece
May 29th – June 2nd
2011



[2.3] C-SPARQL

Emanuele Della Valle

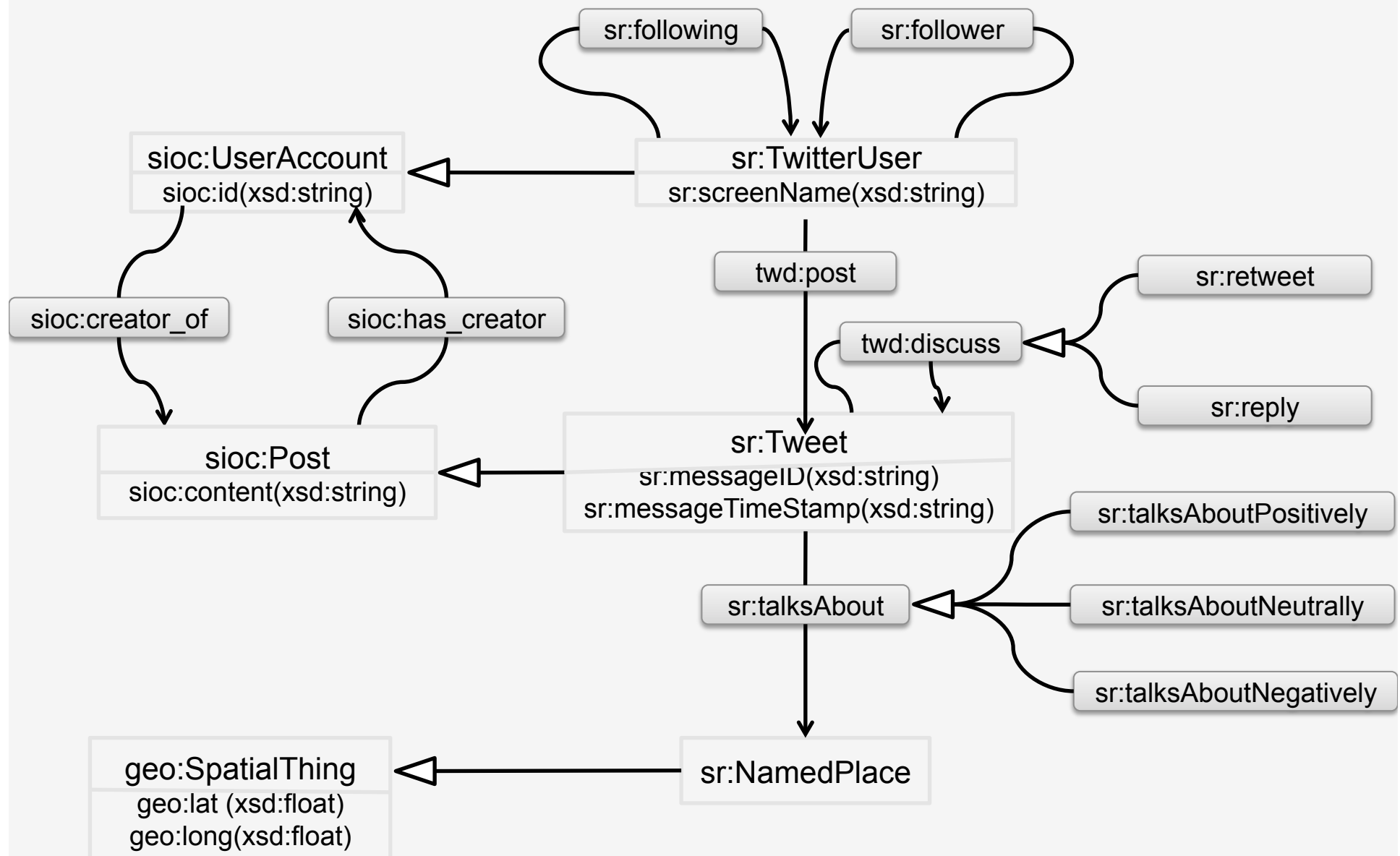
emanuele.dellavalle@polimi.it

<http://emanueledellavalle.org>

- RDF Stream
- C-SPARQL
 - FROM STREAM Clause
 - Query and Stream Registration
 - Query Composability
 - Multiple Streams
 - TimeStamp Function
- An Execution Environment for C-SPARQL
- Applications of C-SPARQL
 - Social Media Analysis
 - Traffic Management
 - Environmental Monitoring
 - Financial Analysis
- C-SPARQL Queries Optimization
- Formal Semantics of C-SPARQL
- Conclusions

MEMO: Data Model Used in the Examples

Heraklion
Greece
May 29th – June 2nd
2011

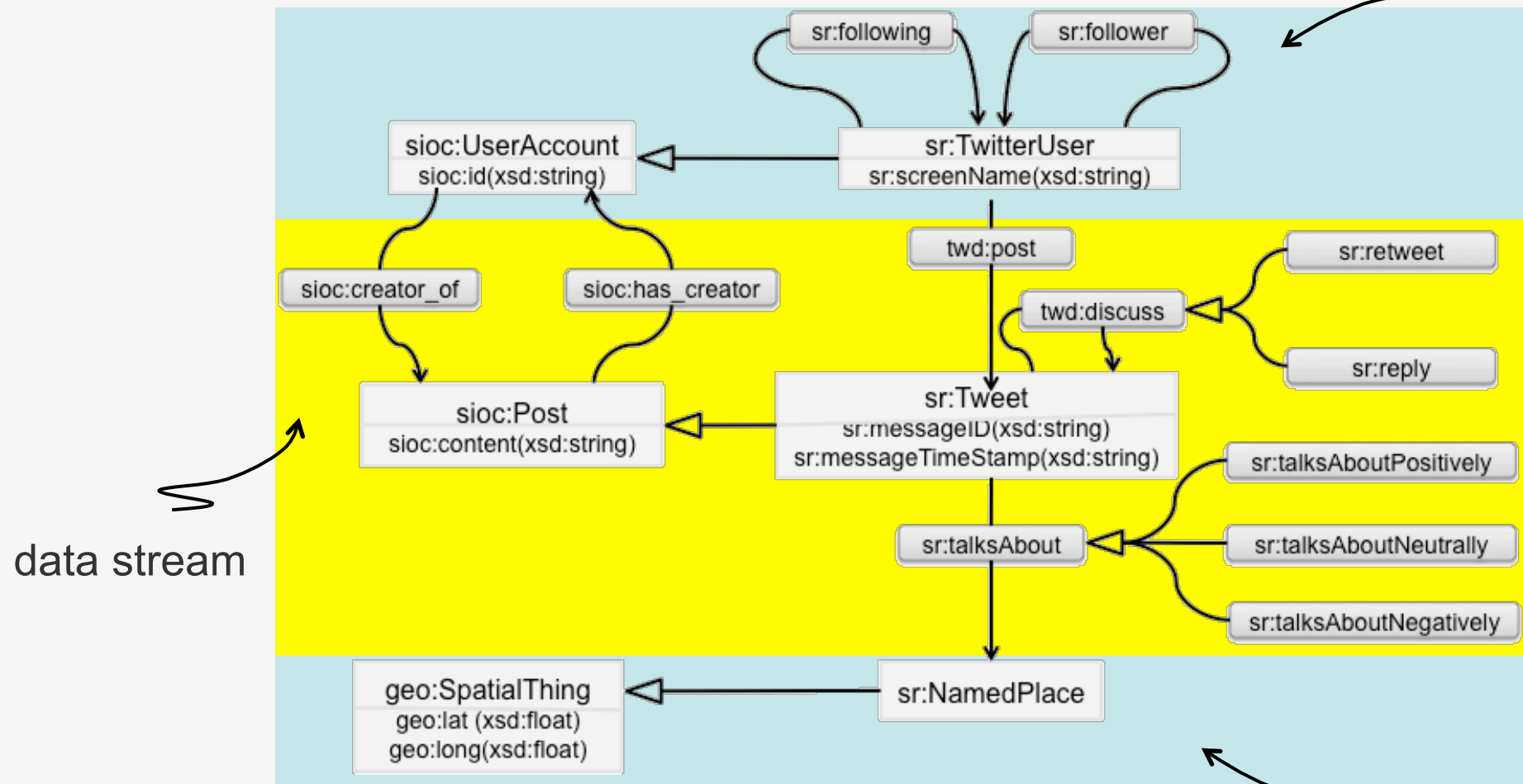


Streaming vs. Background Information

Heraklion
Greece
May 29th – June 2nd
2011



User related background knowledge



Point of Interest related background knowledge

- RDF Stream Data Type

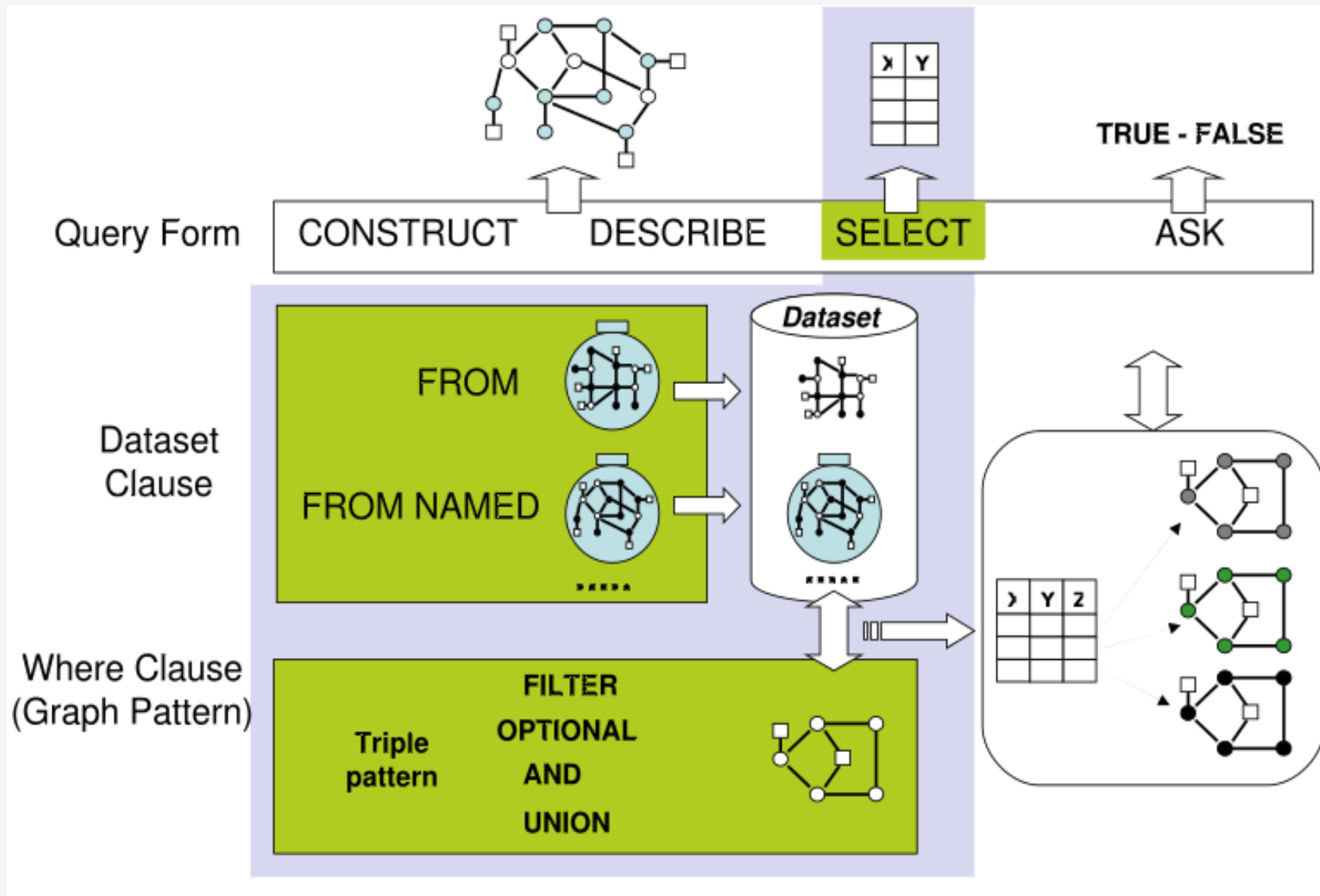
- Ordered sequence of pairs, where each pair is made of an RDF triple and its timestamp

$$\begin{array}{c} \dots \\ (\langle subj_i, pred_i, obj_i \rangle, \tau_i) \\ (\langle subj_{i+1}, pred_{i+1}, obj_{i+1} \rangle, \tau_{i+1}) \\ \dots \end{array}$$

- Timestamps are not required to be unique, they must be non-decreasing

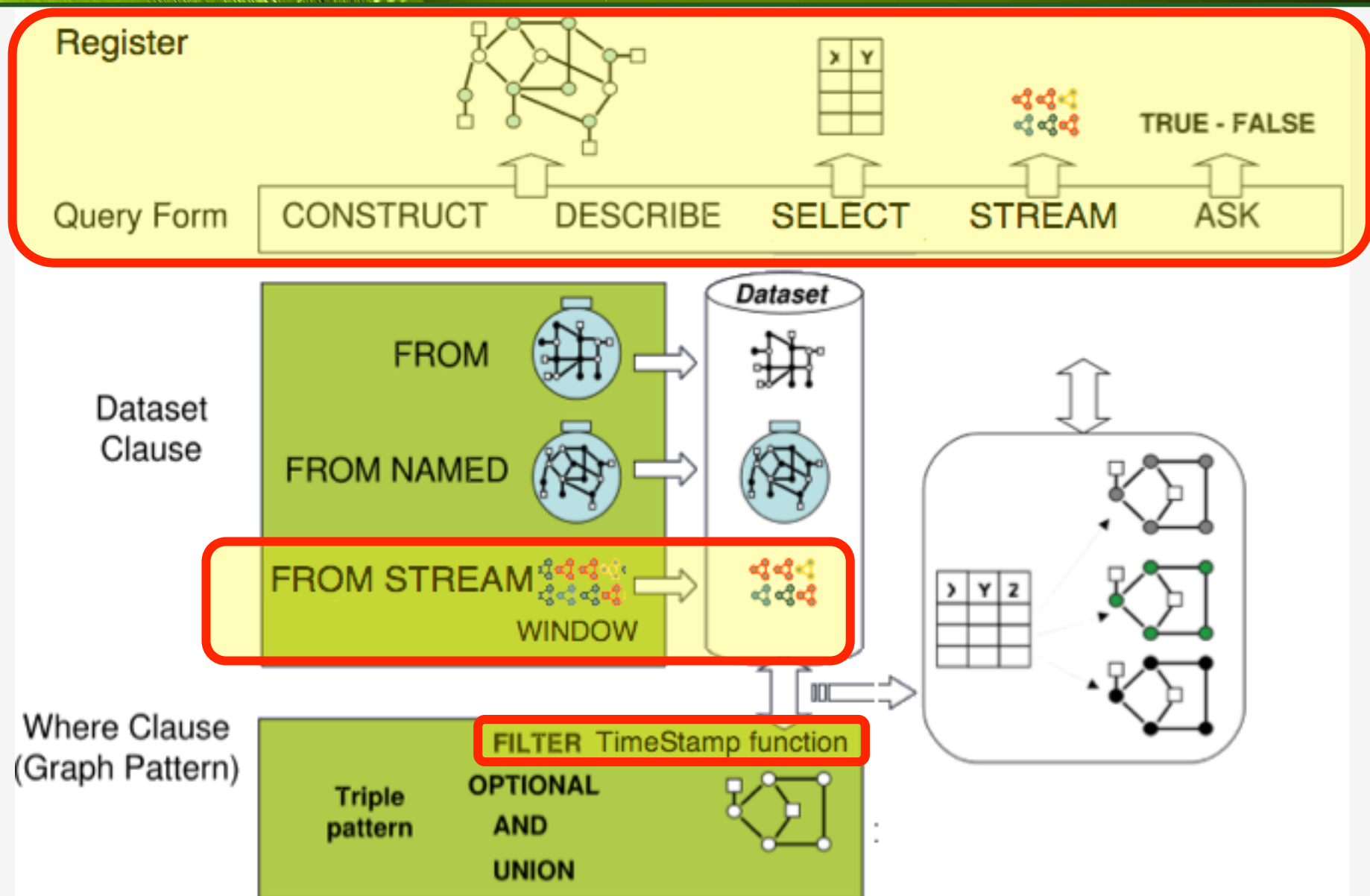
- E.g.,

```
(<:Alice      :posts                :post1 >,    2010-02-12T13:34:41)
(<:post1      :talksAboutPositively :LaScala>,    2010-02-12T13:34:41)
(<:Bob        :posts                :post2 >,    2010-02-12T13:36:28)
(<:post2      :talksAboutNegatively :Duomo>,      2010-02-12T13:36:28)
```



Where C-SPARQL Extends SPARQL

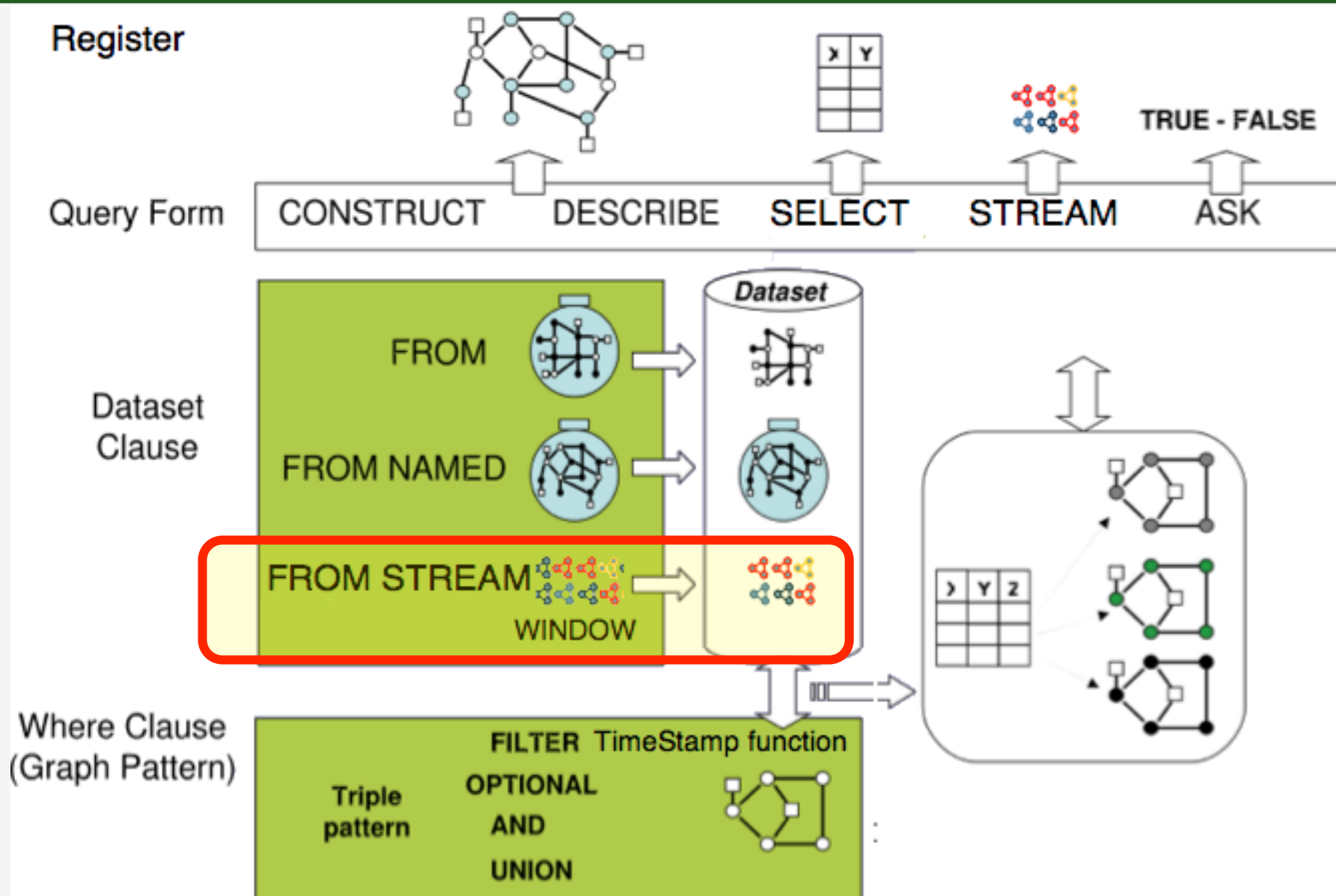
Heraklion
Greece
May 29th – June 2nd
2011



Where C-SPARQL Extends SPARQL

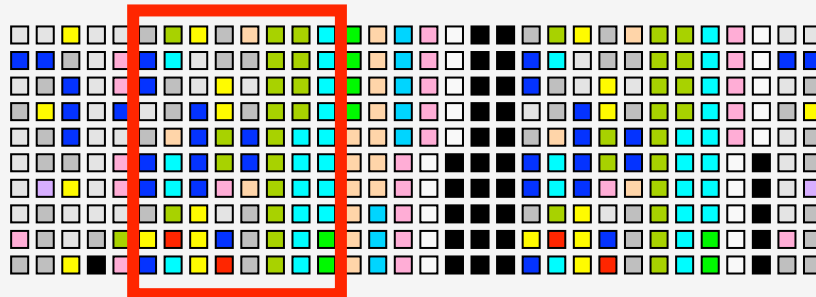
FROM STREAM Clause

Heraklion
Greece
May 29th – June 2nd
2011

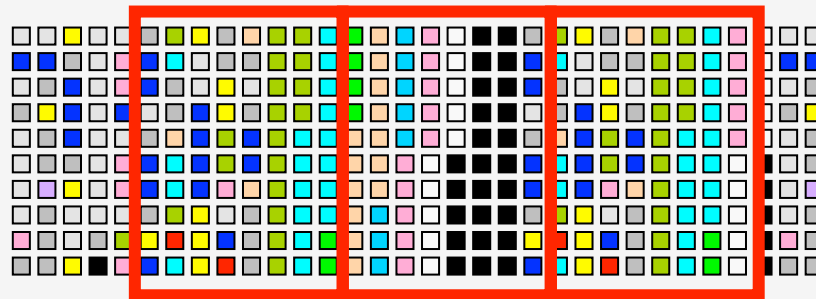


- Processing RDF streams in C-SPARQL requires
 - to **identify** such data sources and
 - to **specify selection criteria** over them.
- For identification,
 - each RDF stream is associated with a distinct **IRI**
- For selection
 - streams are intrinsically infinite,
 - the notion of **windows upon RDF streams** allows for loading in the dataset the most recent triples
 - Windows over streams are specified through the **FROM STREAM** clause

- **physical:** a given number of triples
- **logical:** a variable number of triples which occur during a given time interval (e.g., 1 hour)
 - **Sliding:** they are progressively advanced of a given STEP (e.g., 5 minutes)



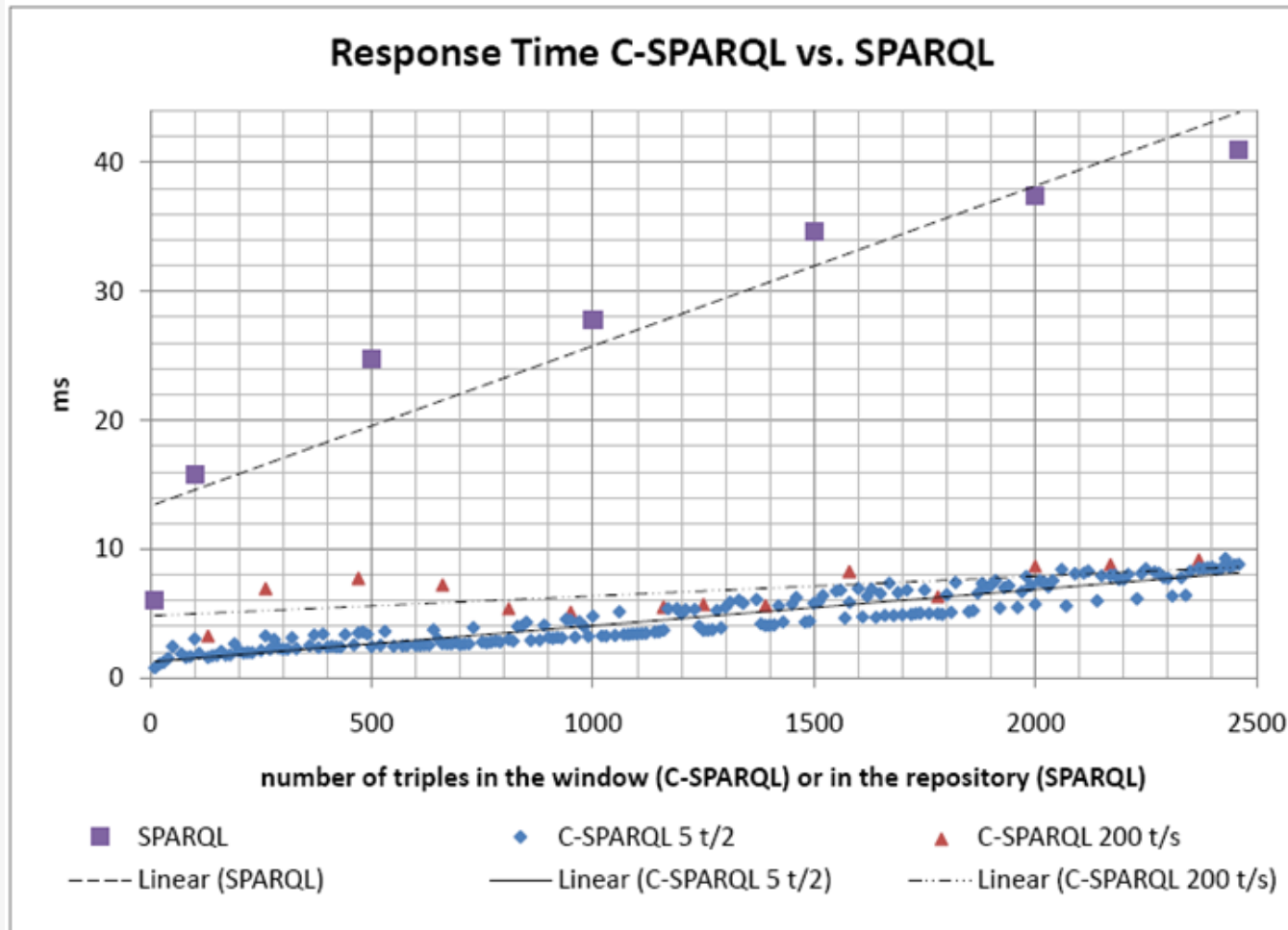
- **Tumbling:** they are advanced of exactly their time interval

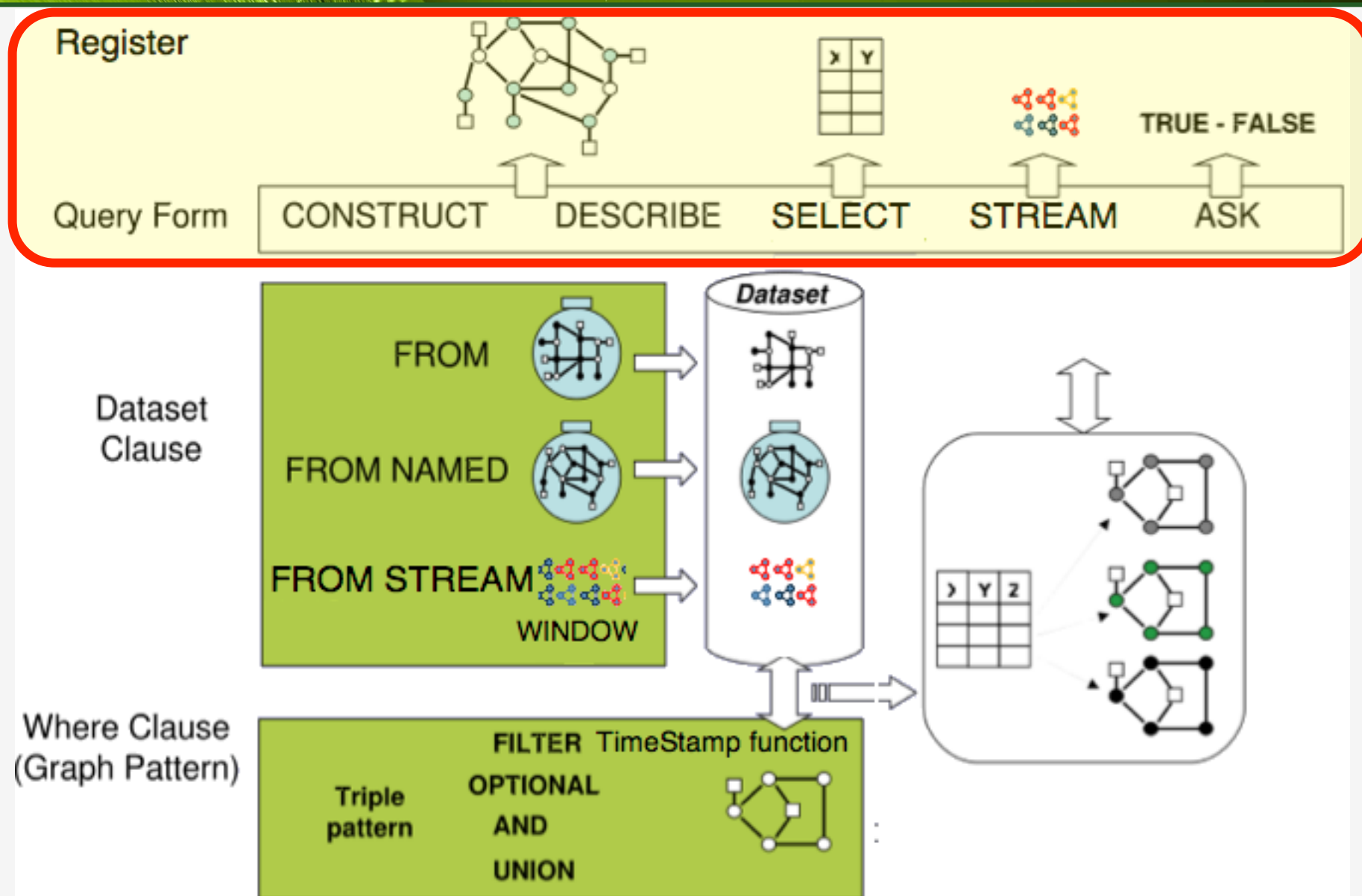


- extract all restaurants (i.e., all POIs whose category is restaurant) the followers of Alice have been talking about positively in the last 15 minutes. Have the window sliding every minute

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
PREFIX srd: <http://www.streamreasoning.org/sr4ld2011/data#>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
PREFIX yago: <http://dbpedia.org/class/yago/>
SELECT ?poi
FROM STREAM <http://www.streamreasoning.org/sr4ld2011/stream#>
[RANGE 15m STEP 1m]
WHERE {
  ?poi a sr:NamedPlace ;
       skos:subject yago:Restaurant .
  ?friend sr:following srd:Alice ;
          sr:posts ?tweet .
  ?tweet sr:talksAboutPositively ?poi .
}
```

- Evaluation of Window-based Selection vs FILTER-base Selection





- $$\textit{Registration} \rightarrow \text{'REGISTER'} \text{'(QUERY'| 'STREAM') } \textit{QueryName}$$
- $$\text{['COMPUTED EVERY' } \textit{Number TimeUnit}] \text{'AS' } \textit{Query}$$

- **Composability:**
 - Query results registered as streams can feed other registered queries just like every other RDF stream

- Create a stream of the restaurants (i.e., all POIs whose category is restaurant) the followers of Alice have been talking about positively in the last 30 minutes. Have the window sliding every 5 minutes.

```
REGISTER STREAM RestaurantsAlicesFollowersLike
```

```
COMPUTED EVERY 5m AS
```

```
PREFIX sr: <http://www.streamreasoning.org/sr4ld2011/onto#>
```

```
PREFIX srd: <http://www.streamreasoning.org/sr4ld2011/data#>
```

```
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
```

```
PREFIX yago: <http://dbpedia.org/class/yago/>
```

```
CONSTRUCT { ?friend sr:talksAboutPositively ?poi }
```

```
FROM STREAM <http://www.streamreasoning.org/sr4ld2011/stream#>  
[RANGE 30m STEP 5m]
```

```
WHERE {
```

```
  ?poi a sr:NamedPlace ;
```

```
        skos:subject yago:Restaurant .
```

```
  ?friend sr:following srd:Alice ;
```

```
        sr:posts ?tweet .
```

```
  ?tweet sr:talksAboutPositively ?poi .
```

```
}
```

- The output is constructed in the format of an RDF stream.
- Every query execution may produce from a minimum of zero triples to a maximum of an entire graph.
- The timestamp is always dependent on the query execution time only, and is not taken from the triples that match the patterns in the WHERE clause.

- For each user, every 5 minutes, count the overall number of tweets posted in the last 30 minutes and the number of distinct point of interest to which the tweets refer.

- **REGISTER QUERY** GlobalCountOfTweets

COMPUTED EVERY 5m AS

PREFIX sr:<<http://www.streamreasoning.org/sr4ld2011/onto#>>

SELECT ?user (COUNT(?tweet) as ?numberOfTweets)

(COUNT(DISTINCT ?poi) as ?numDifferentPois)

FROM STREAM <<http://www.streamreasoning.org/sr4ld2011/stream#>>
[RANGE 30m STEP 5m]

WHERE {

 ?user sr:posts ?tweet .

 ?tweet sr:talksAboutPositively ?poi .

}

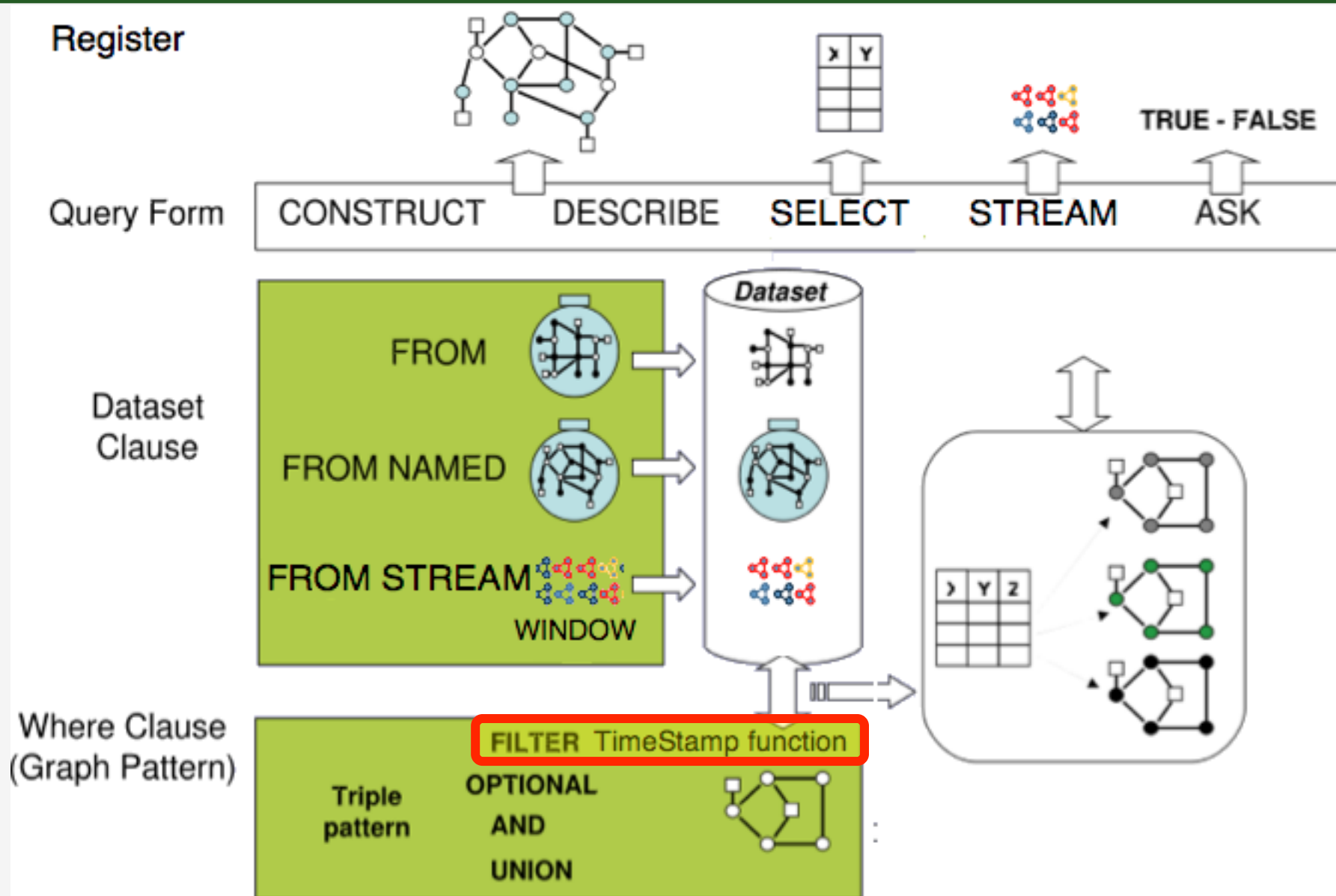
GROUP BY ?user

- Which are the current best 10 restaurants according to Alice's followers?
- The answer can be computed using the stream of results of the `RestaurantsAlicesFollowersLike` C-SPARQL query , i.e., the restaurants Alice's followers have been talking about positively in the last 30 minutes

```
REGISTER QUERY Top10Restaurants4AlicesFollowers
COMPUTED EVERY 5m AS
PREFIX sr:<http://www.streamreasoning.org/sr4ld2011/onto#>
SELECT ?poi (COUNT(DISTINCT ?follower) as ?numOfFollowers)
FROM STREAM <http://www.streamreasoning.org/sr4ld2011/
RestaurantsAlicesFollowersLike/> [RANGE 30m STEP 5m]
WHERE {
    ?follower sr:talksAboutPositively ?poi .
}
GROUP BY ?poi
ORDER BY DESC (?numOfFollowers)
LIMIT 10
```

- C-SPARQL queries can combine triples from multiple RDF stream
- Let's add to our running example a second stream concerning the entrance of people into the restaurants (e.g., obtained by RDF-izing Foursquare check-ins).
- The next query takes as input the stream generated by the query `RestaurantsAlicesFollowersLike` and this second stream, and only consider the advice of followers who, in the last week, actually checked-in the restaurants they positively talked about

```
■ REGISTER QUERY Top10Restaurants4TrustableAlicesFollowers
  COMPUTED EVERY 5m AS
  PREFIX sr:<http://www.streamreasoning.org/sr4ld2011/onto#>
  PREFIX fs:<http://www.streamreasoning.org/foursquare/onto#>
  SELECT ?poi (COUNT(DISTINCT ?follower) as ?numOfFollowers)
  FROM STREAM <http://www.streamreasoning.org/sr4ld2011/
  RestaurantsAlicesFollowersLike/> [RANGE 30m STEP 5m]
  FROM STREAM <http://www.streamreasoning.org/sr4ld2011/
  foursquareCheckIns/> [RANGE 7d]
  WHERE {
    ?follower sr:talksAboutPositively ?poi .
    ?follower fs:checksIn ?poi .
  }
  GROUP BY ?poi
  ORDER BY DESC (?numOfFollowers)
  LIMIT 10
```



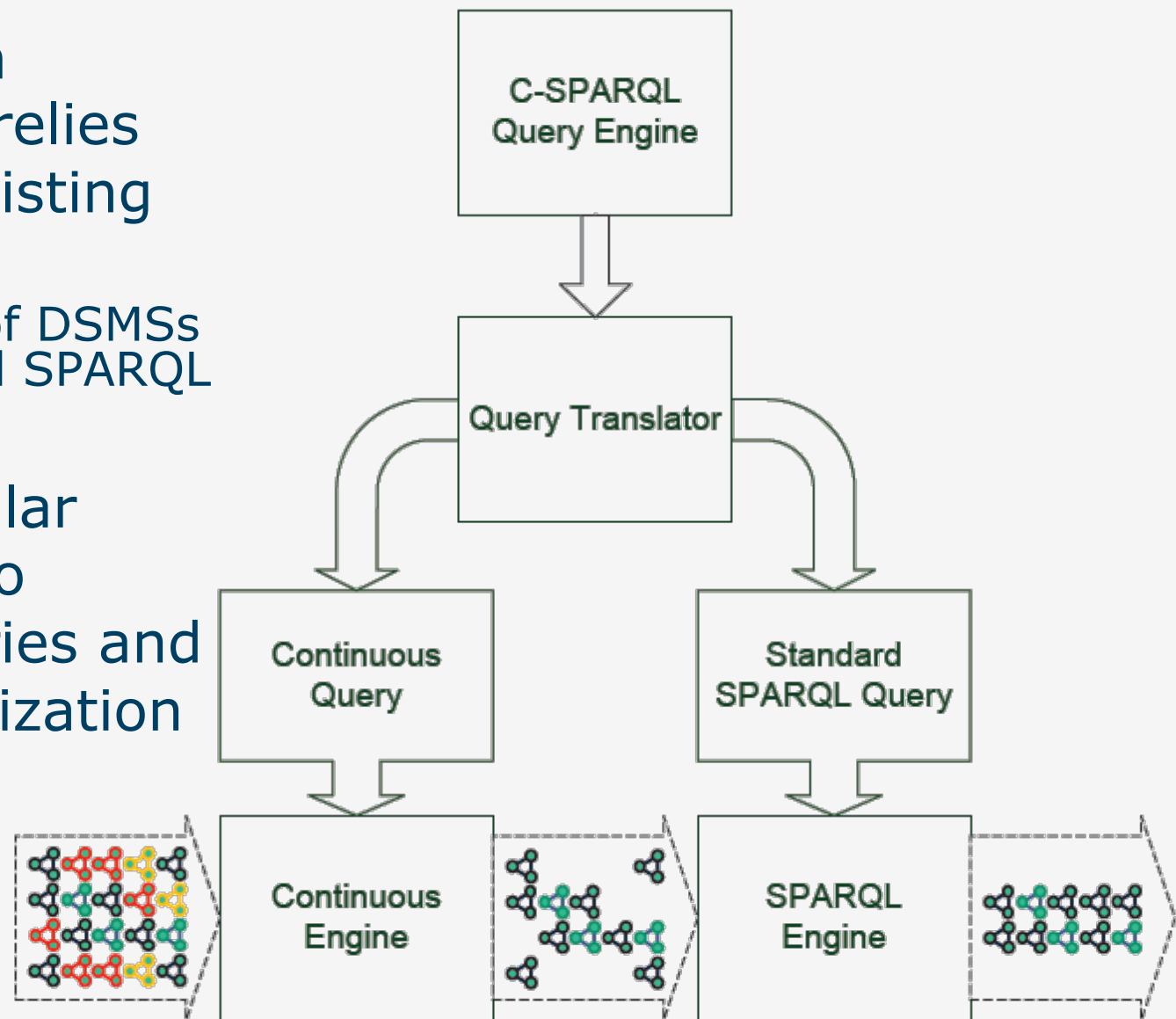
- The timestamp of a triple can be bound to a variable using a `timestamp()` function, which has two arguments:
 - the name of a variable, introduced in the **WHERE** clause
 - The (optional) IRI of the stream, that can be obtained through SPARQL **GRAPH** clause.
- Semantics

?var	Timestamp(?var, [?graph])
Not bound	Type Error
Bound once	Timestamp of ?var in ?graph
Bound more than once	The most recent timestamp of the bindings of ?var • in ?graph, if it is specified • in all the streams referenced in the query otherwise

- Who are the opinion makers? i.e., the users who are likely to influence the behavior of other users who follow them
- ```
REGISTER STREAM OpinionMakers COMPUTED EVERY 5m AS
PREFIX sr:<http://www.streamreasoning.org/sr4ld2011/onto#>
CONSTRUCT { ?opinionMaker sd:about ?resource }
FROM STREAM <http://www.streamreasoning.org/sr4ld2011/stream#>
[RANGE 30m STEP 5m]

WHERE {
 ?opinionMaker ?opinion ?poi.
 ?follower sr:following ?opinionMaker.
 ?follower ?opinion ?poi.
 FILTER (
 sr:timestamp(?follower) > sr:timestamp(?opinionMaker)
 && (?opinion = sr:talksAboutPositively ||
 ?opinion = sr:talksAboutNegatively))
 }
HAVING (COUNT(DISTINCT ?follower) > 3)
```

- The execution environment relies entirely on existing technologies
  - Integration of DSMSs and standard SPARQL engines
- Simple, modular architecture to evaluate queries and our the optimization strategies



- The scenario of social data analysis is just one example of many possible applications of C-SPARQL.
- Other examples include:
  - Traffic Management
  - Environmental Monitoring
  - Financial Analysis

### Detect cars turning from one street into another

**REGISTER STREAM** CarsTurningFromGolgiIntoCeloria  
**COMPUTED EVERY 1 MIN AS**

**PREFIX** c: <http://linkedurbandata.org/city#>

**PREFIX** t: <http://linkedurbandata.org/traffic#>

**SELECT DISTINCT** ?car1

**FROM STREAM** <http://stream.org/milancameras.trdf>

**[RANGE 5 MIN STEP 1 MIN]**

**WHERE** { ?camera1 c:monitors c:MI-via-Celoria .

?camera2 c:monitors c:MI-via-Golgi.

?camera1 c:placedAt ?tr\_light .

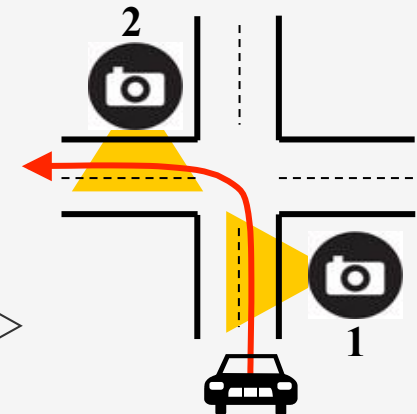
?camera2 c:placedAt ?tr\_light .

?camera1 t:registers ?car1 .

?camera2 t:registers ?car2 .

**FILTER** ( **timestamp(?car1)** < **timestamp(?car2)**

**AND** ?car1 = ?car2 ) }



**timestamp()**  
function allows  
for detecting  
that the car is  
turning in one  
specific  
direction and  
not viceversa

- The following C-SPARQL query detects if a weather station is observing a blizzard, i.e., a severe storm, characterized by low temperatures, strong winds, and heavy snow, lasting for 3 hours or more.
- **REGISTER STREAM** BlizzardDetection **COMPUTED EVERY 10m AS**  
PREFIX w:<http://knoesis.wright.edu/ssw/ont/weather.owl#>  
PREFIX so: :<http://knoesis.wright.edu/ssw/>  
CONSTRUCT {?s so:generatedObservation [a w:blizzard] }  
FROM <http://oilprod.org/weatherStations.rdf>  
**FROM STREAM** <http://oilprod.org/weatherObs.trdf>  
          **[RANGE 3h STEP 10m]**  
WHERE {  
    ?s grs:point "66.348085,10.180662" ;  
    so:generatedObservation [a w:SnowfallObservation] .

```
■ { SELECT ?s WHERE {
 ?s so:generatedObservation ?o1 .
 ?o1 a w:TemperatureObservation ;
 so:observedProperty w:AirTemperature ;
 so:result [so:value ?temperature] .
} GROUP BY ?s
HAVING (AVG(?temperature) < "0.0"^^xsd:float) }
{ SELECT ?s WHERE {
 ?s so:generatedObservation ?o2 .
 ?o2 a w:WindObservation ;
 so:observedProperty w:WindSpeed ;
 so:result [so:value ?speed] .
} GROUP BY ?s
HAVING (MIN(?speed) > "40.0"^^xsd:float) }
}
```

- **NOTE:** read more about Semantic Sensor Web on <http://wiki.knoesis.org/index.php/SSW>

- Get the total amount of money transferred by each Swiss broker in the last 24 hours (ignore small transactions)

**REGISTER QUERY** TotalAmountPerDayAndBroker **AS**

SELECT DISTINCT ?broker **SUM**(?amount)

FROM <http://brokerscentral.org/brokers.rdf>

**FROM STREAM** <http://stockexchange.org/market.trdf>

**[RANGE 24h TUMBLING]**

WHERE {

    ?broker b:is\_from ?country .

    ?broker x:does ?transaction .

    ?transaction x:with ?amount .

    FILTER (?country = 'CH' && ?amount >= 10)

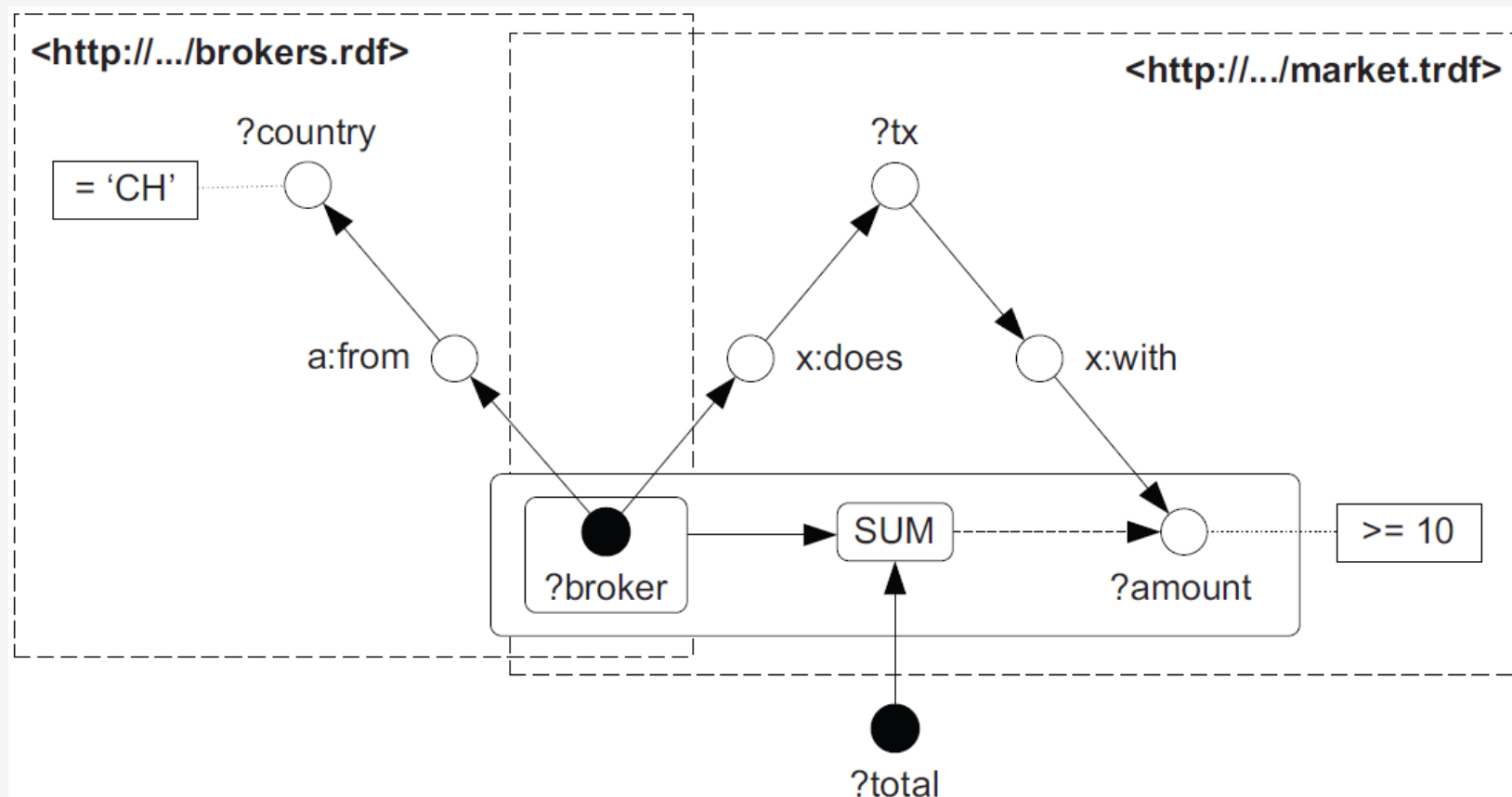
}

**GROUP BY** ?broker

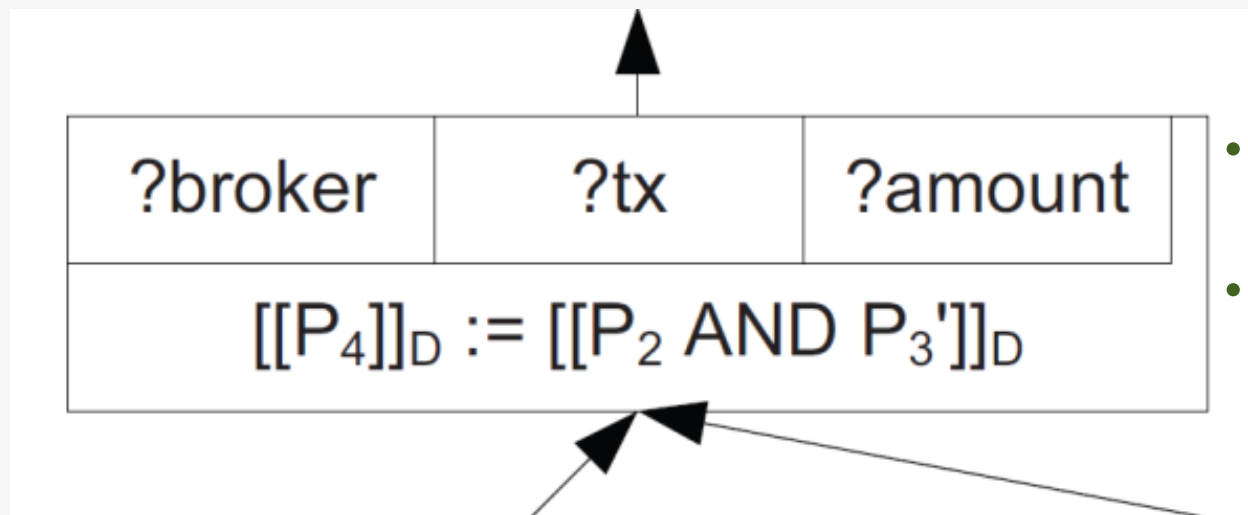
The rationale behind the optimizations:

- Move the computation as near to the data source as possible
- Let the architecture modules do what they do best
- We need to:
  - Represent the query in an algebraic way
  - Recognize where data is coming from
- We have:
  - Implemented the push of filters and projection
  - Studied the problem of pushing aggregates
- We use two graphical formalisms to represent queries
  - The Denotational Graph (**D-Graph**)
  - The Operator Graph (**O-Graph**)

- Represents graph patterns, filters, and aggregations
- Used to represent data provenance
  - Dashed blocks as data sources



- Directed graph where
  - Nodes represent operators
  - Edges represent the flow of data
- Used for both query evaluation and optimization
- Based on the SPARQL Query Graph Model (SQGM)
  - Hartig, Hesse. *The SPARQL Query Graph Model for Query Optimization* (ESWC 2007)



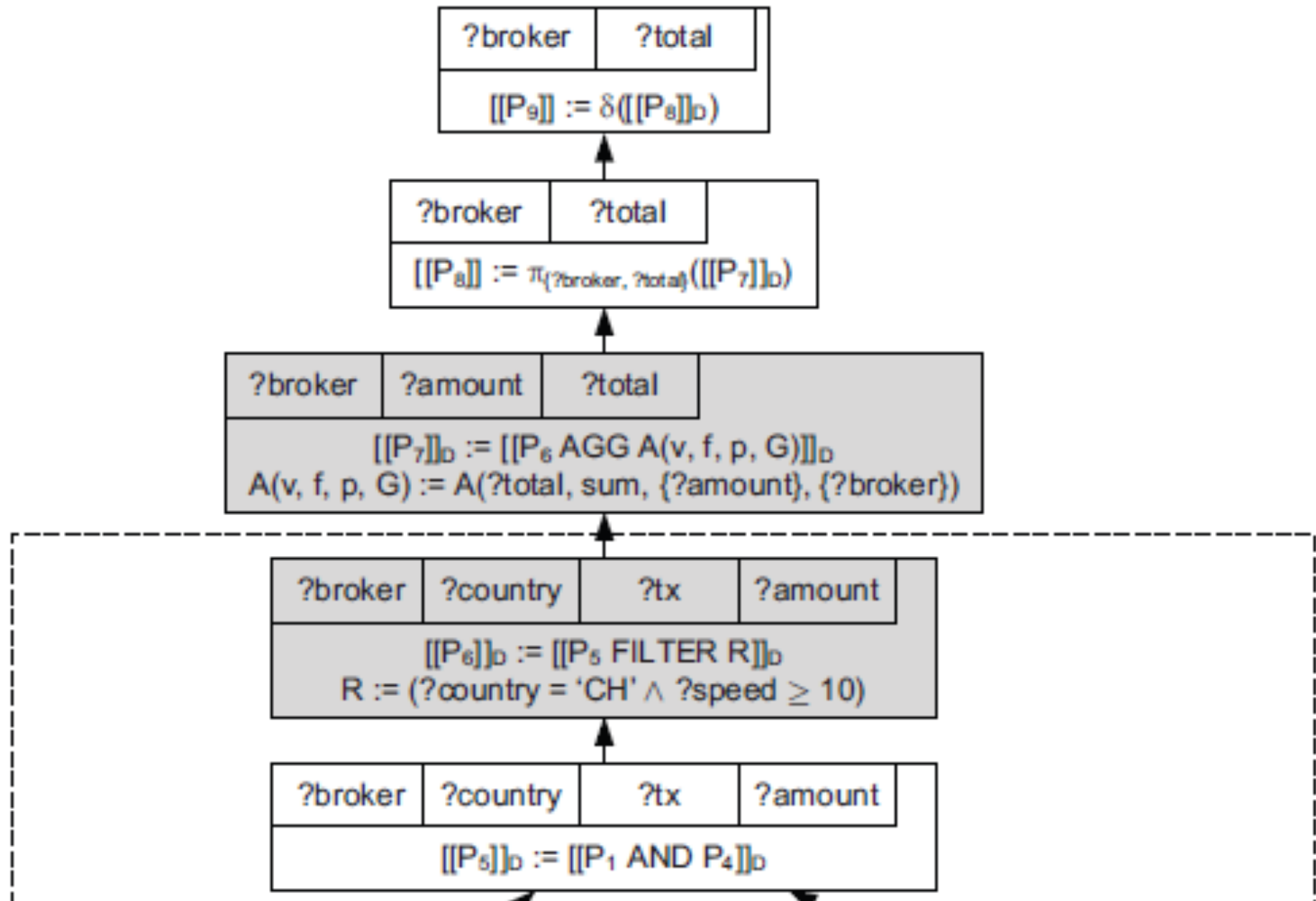
- Output of the operator
- Operator evaluation based on C-SPARQL semantics

C-SPARQL Formal semantics is based on  
Pérez, Arenas, Gutierrez: *Semantics and Complexity of SPARQL*, ISWC 2006

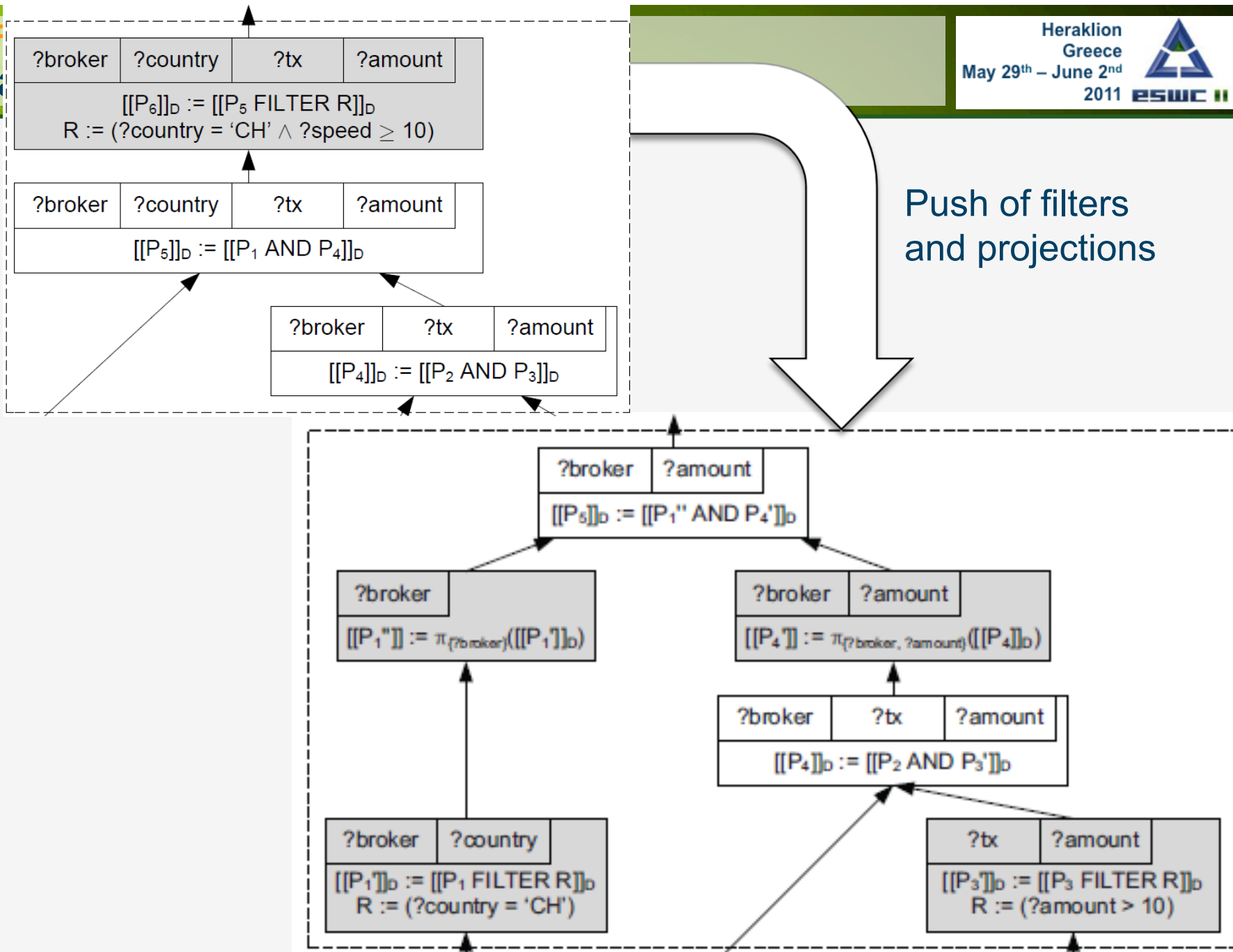
# C-SPARQL Queries Optimization

## Example of Optimization

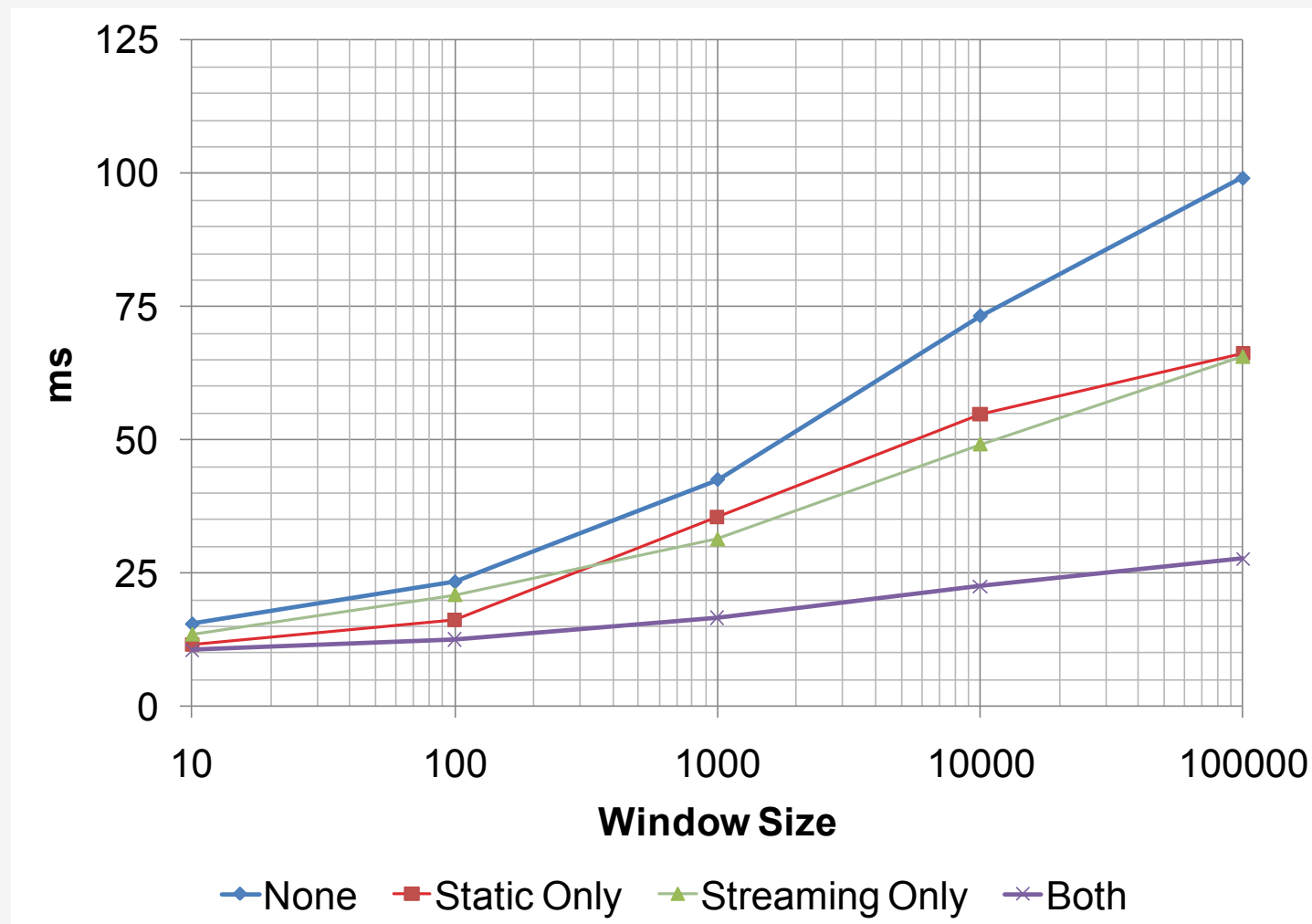
Heraklion  
Greece  
May 29<sup>th</sup> – June 2<sup>nd</sup>  
2011



Push of filters  
and projections



- Push of filters and projections



- As for selection, aggregates can be pushed towards the streaming engine, if some constraints hold
- Aggregate evaluations can then be classified as:
  - **totally distributable**: all the data needed to compute every aggregation belongs to one stream
    - Example: The sum of amounts of transactions, grouped by sellers in each market, when each stream is associated with one market
  - **partially distributable**: there exists a distributed computation of the aggregate function over the streams
    - Example: The sum of amounts of transactions grouped by seller, over all markets, when each stream is still associated with one market
  - **not distributable**
    - Example: The count of distinct brokers who did at least one transaction

- C-SPARQL semantics formalization is based on SPARQL semantics formalization by Perez et al.
  - Pérez, Arenas, Gutierrez: Semantics and Complexity of SPARQL, ISWC 2006
- A mapping  $\mu$  is a partial function  $\mu : V \rightarrow T$  which computes the bindings for all the variables of a query, matching a pattern  $P$  against an RDF dataset ( $D$ )
- Two mappings  $\mu'$  and  $\mu''$  are said to be compatible if
  - $\forall x \in \text{dom}(\mu') \cap \text{dom}(\mu''), \text{ then } \mu'(x) = \mu''(x)$
  - Where  $\text{dom}(\mu)$  is the subset of  $V$  where  $\mu$  is defined

- The evaluation of a graph pattern  $P$  over a dataset  $D$  ( $[[P]]_D$ ) is defined recursively:
  - $[[t]]_D = \{ \mu \mid \text{dom}(\mu) = \text{var}(t) \wedge \mu(t) \in D \}$ , where  $t$  is a triple pattern and  $\text{var}(t)$  is the set of variables occurring in  $t$
  - $[(P1 \text{ AND } P2)]_D = [[P1]]_D \bowtie [[P2]]_D$
  - $[(P1 \text{ OPT } P2)]_D = [[P1]]_D \Join [[P2]]_D$
  - $[(P \text{ AGG } A)]_D = [[P]]_D \Join [[A]]_D$ , where  $A(va, fa, pa, Ga)$  is an aggregation pattern

- Let  $\Omega_1$  and  $\Omega_2$  be sets of mappings.
  - Operators for mappings composition:
    - $\Omega_1 \bowtie \Omega_2 = \{ \mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ are compatible} \}$
    - $\Omega_1 \cup \Omega_2 = \{ \mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2 \}$
    - $\Omega_1 \setminus \Omega_2 = \{ \mu \in \Omega_1 \mid \forall \mu' \in \Omega_2, \mu \text{ and } \mu' \text{ are not compatible} \}$
  - The left outer-join is a derived operator:
    - $\Omega_1 \Join \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$
- The evaluation of a graph pattern  $P$  over a dataset  $D$  ( $[[P]]_D$ ) is defined recursively:
  - $[[t]]_D = \{ \mu \mid \text{dom}(\mu) = \text{var}(t) \wedge \mu(t) \in D \}$ , where  $t$  is a triple pattern and  $\text{var}(t)$  is the set of variables occurring in  $t$ .
  - $[[P_1 \text{ AND } P_2]]_D = [[P_1]]_D \bowtie [[P_2]]_D$
  - $[[P_1 \text{ OPT } P_2]]_D = [[P_1]]_D \Join [[P_2]]_D$
  - $[[P \text{ AGG } A]]_D = [[P]]_D \Join [[A]]_D$ , where  $A(v_a, f_a, p_a, G_a)$  is an aggregation pattern.

- Let  $\Omega_1$  and  $\Omega_2$  be sets of mappings.
  - Operators for mappings composition:
    - $\Omega_1 \bowtie \Omega_2 = \{ \mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ are compatible} \}$
    - $\Omega_1 \cup \Omega_2 = \{ \mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2 \}$
    - $\Omega_1 \setminus \Omega_2 = \{ \mu \in \Omega_1 \mid \forall \mu' \in \Omega_2, \mu \text{ and } \mu' \text{ are not compatible} \}$
  - The left outer-join is a derived operator:
    - $\Omega_1 \Join \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$
- The evaluation of a graph pattern  $P$  over a dataset  $D$  ( $[[P]]_D$ ) is defined recursively:
  - $[[t]]_D = \{ \mu \mid \text{dom}(\mu) = \text{var}(t) \wedge \mu(t) \in D \}$ , where  $t$  is a triple pattern and  $\text{var}(t)$  is the set of variables occurring in  $t$ .
  - $[[P_1 \text{ AND } P_2]]_D = [[P_1]]_D \bowtie [[P_2]]_D$
  - $[[P_1 \text{ OPT } P_2]]_D = [[P_1]]_D \Join [[P_2]]_D$
  - $[[P \text{ AGG } A]]_D = [[P]]_D \Join [[A]]_D$ , where  $A(v_a, f_a, p_a, G_a)$  is an aggregation pattern.

- **REGISTER QUERY** CarsPassingThroughEachStreet  
**COMPUTED EVERY** 5m **AS**  
**SELECT** ?tollgate, (count(?car) AS ?numOfCars)  
**FROM STREAM** <http://streams.org/gates.trdf> [**RANGE** 30m **STEP** 5m]  
**WHERE** {  
    ?tollgate t:registers ?car .  
    ?tollgate c:placedIn ?street . }  
**GROUP BY** ?tollgate
- The query can be expressed as:  $P = t_1 \text{ AND } t_2 \text{ AGG } A_1$ 
  - $t_1 = ?\text{tollgate } t:\text{registers } ?\text{car}$
  - $t_2 = ?\text{tollgate } c:\text{placedIn } ?\text{street}$
  - $A_1 = A(?passages, \text{count}, ?\text{car}, ?\text{street})$
- $D = \omega_1(R; \text{now} - 10; \text{now}) \cup S$ , being  $R$  the stream and  $S$  the static knowledge

- $P$  is evaluated as

$$[[P]]_D = [[t_1]]_D \bowtie [[t_2]]_D \bowtie [[A_1]]_D$$

$$[[P]]_D = [[t_1]]_R \bowtie [[t_2]]_S \bowtie [[A_1]]_D$$

- The evaluation steps are

- $T_1 = \text{Eval}(t_1)_R$
- $T_2 = \text{Eval}(t_2)_S$
- $T_3 = T_1 \text{ AND } T_2$
- $A_1 = \text{Eval}(A(?total, count, ?car, ?street))_D$
- $\text{Eval}(P)_D = T_3 \text{ AGG } A_1$

| Eval( $T_1$ ) |         | Eval( $T_2$ ) |            | Eval( $T_1 \text{ AND } T_2$ ) |         |            | Eval( $A_1$ ) |           |
|---------------|---------|---------------|------------|--------------------------------|---------|------------|---------------|-----------|
| ?tollgate     | ?car    | ?tollgate     | ?street    | ?sensor                        | ?car    | ?street    | ?street       | ?passages |
| T-130         | AB123CD | T-130         | Pine Park  | T-130                          | AB123CD | Pine Park  | Pine Park     | 1         |
| T-142         | CD234EF | T-142         | Oak avenue | T-142                          | CD234EF | Oak avenue | Oak avenue    | 2         |
| T-142         | EF567GH | ...           | ...        | T-142                          | EF567GH | Oak avenue | ...           | ...       |
| ...           | ...     |               |            | ...                            | ...     | ...        |               |           |

| Eval( $P$ ) |         |            |           |
|-------------|---------|------------|-----------|
| ?tollgate   | ?car    | ?street    | ?passages |
| T-130       | AB123CD | Pine Park  | 1         |
| T-142       | CD234EF | Oak Avenue | 2         |
| T-142       | EF567GH | Oak Avenue | 2         |
| ...         | ...     | ...        | ...       |

- We specified of C-SPARQL **syntax**
  - Incrementally, from existing specifications
    - Including windows, grouping, aggregates, timestamping
- We gave the formal **semantics** of C-SPARQL
  - Query registration, handling overloads
  - Order of evaluation, pattern matching over time, ...
- We show **applicability to multiple** industrial **sectors**
- We investigated **efficiency of evaluation**
  - Defining a suitable algebra
  - Applying optimizations

[EDBT2010] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri and Michael Grossniklaus. An Execution Environment for C-SPARQL Queries. EDBT 2010

[WWW2009] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, Michael Grossniklaus: C-SPARQL: SPARQL for continuous querying. WWW 2009: 1061-1062

[IJSC2010] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, Michael Grossniklaus: C-SPARQL: a Continuous Query Language for RDF Data Streams. Int. J. Semantic Computing 4(1): 3-25 (2010)

[IEEE-IS2010] Davide Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, Yi Huang, Volker Tresp, Achim Rettinger, Hendrik Wermser, "Deductive and Inductive Stream Reasoning for Semantic Social Media Analytics," IEEE Intelligent Systems, 30 Aug. 2010.

# Stream Reasoning For Linked Data

E. Della Valle & J.Z. Pan

Heraklion  
Greece  
May 29<sup>th</sup> – June 2<sup>nd</sup>  
2011



## [2.3] C-SPARQL

Emanuele Della Valle

[emanuele.dellavalle@polimi.it](mailto:emanuele.dellavalle@polimi.it)

<http://emanueledellavalle.org>