Stream Reasoning

Processamento de Streams
FCT-NOVA 2018/19

Reasoning over Events

- Suppose we:
  - Have background knowledge, and use it together with the incoming stream of data
  - Combine data coming from various sources in various formats, and have background knowledge that tells what is what (and that may change over time)
  - Deal with incomplete and noisy information

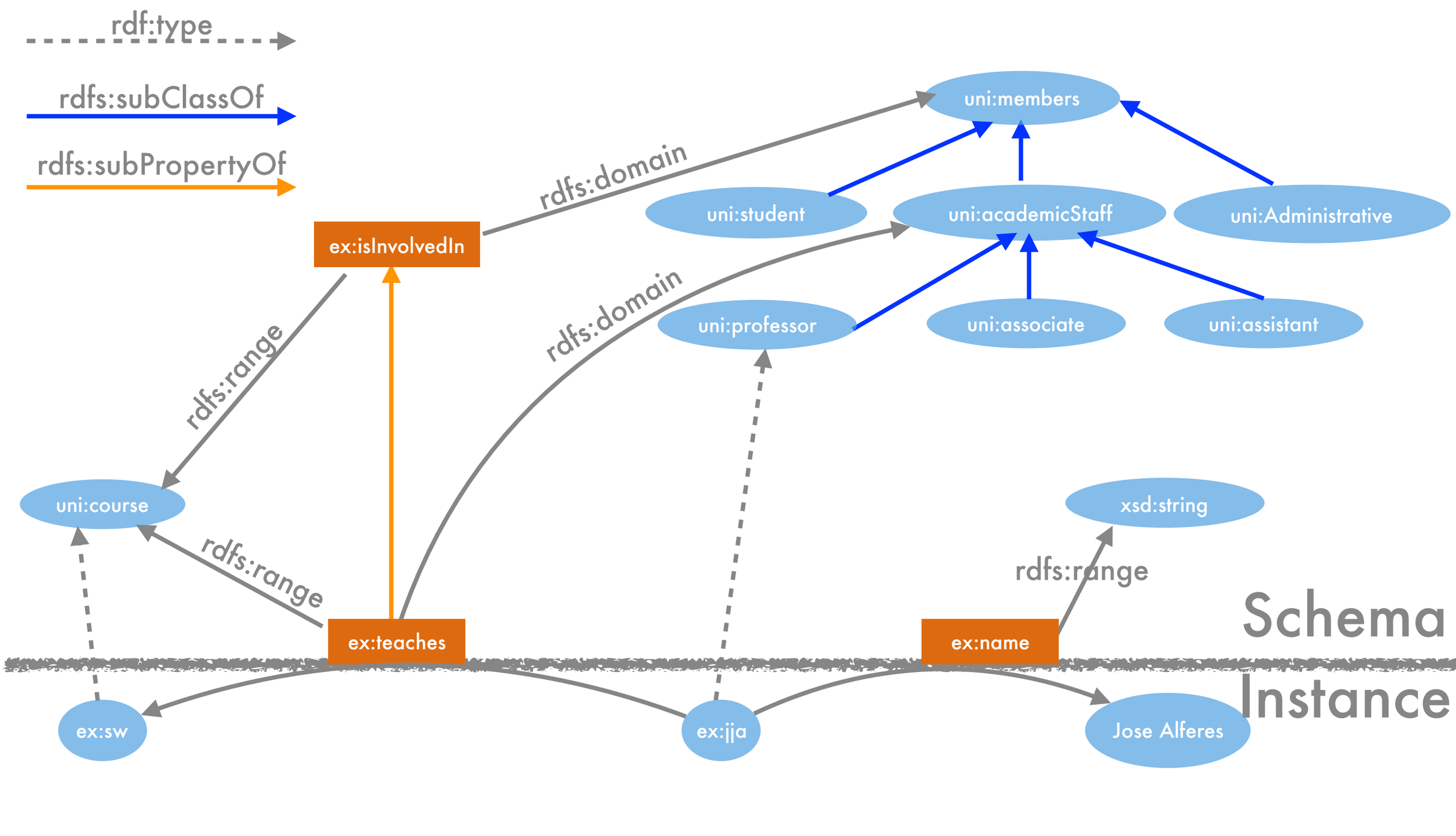
- Stream Reasoning research question
  - Is it possible to make sense, in real time, of multiple, heterogeneous, gigantic, noisy and incomplete data streams, to support decision processes of quite large numbers of simultaneous users?

The Semantic Web

- Provides a common framework that allows data to be shared and reused across applications and community boundaries
  - Data is expressed as a graph of interrelated small pieces (object attribute value/object) - RDF
  - Ontologies describe the meaning of data, directly associated to the graph - RDFS and OWL
  - Languages that query the whole graph (possibly including the data together with ontology) - SPARQL
  - Linked Open Data provides a huge collection of data published in this form

- Goals
  - Semantic Web: Make sense, in real time, of multiple, heterogeneous, gigantic, noisy and incomplete data streams sources

An example schema



An example of integration

- Consider the following data(base) on a bookstore, to be published in a Web page

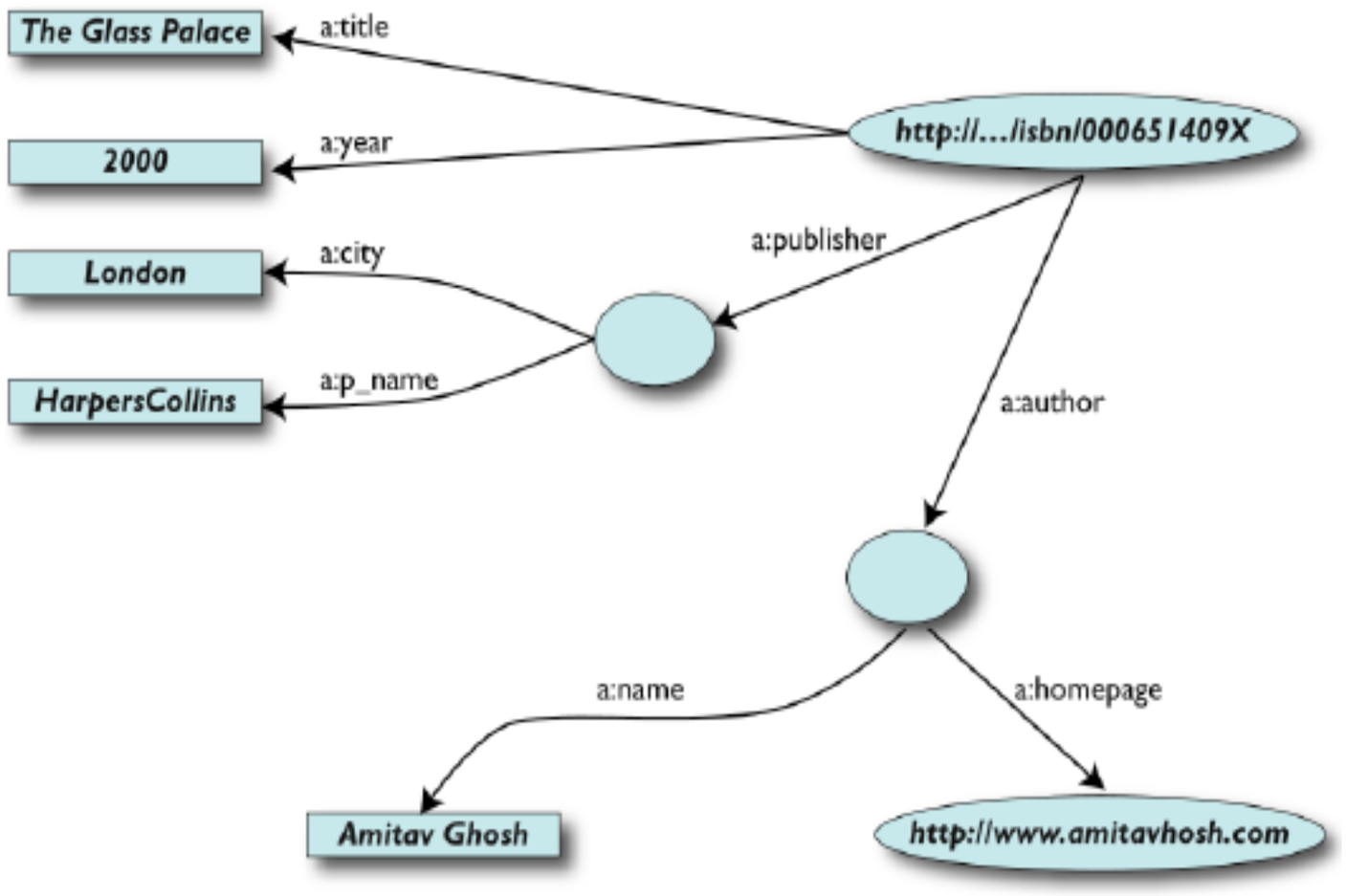
ID	Author	Title	Publisher	Year
ISBN 0-00-651409-X	id_xyz	The Glass Palace	id_qpr	2000

ID	Name	Home page
id_xyz	Amitav Ghosh	http://www.amitavghosh.com/

ID	Publisher Name	City
id_qpr	Harper Collins	London

Data Integration Example

- Besides publishing the data on a nice Web page, also export data as relations
  - Relations form a graph, where nodes refer to either data or identifiers (URI)

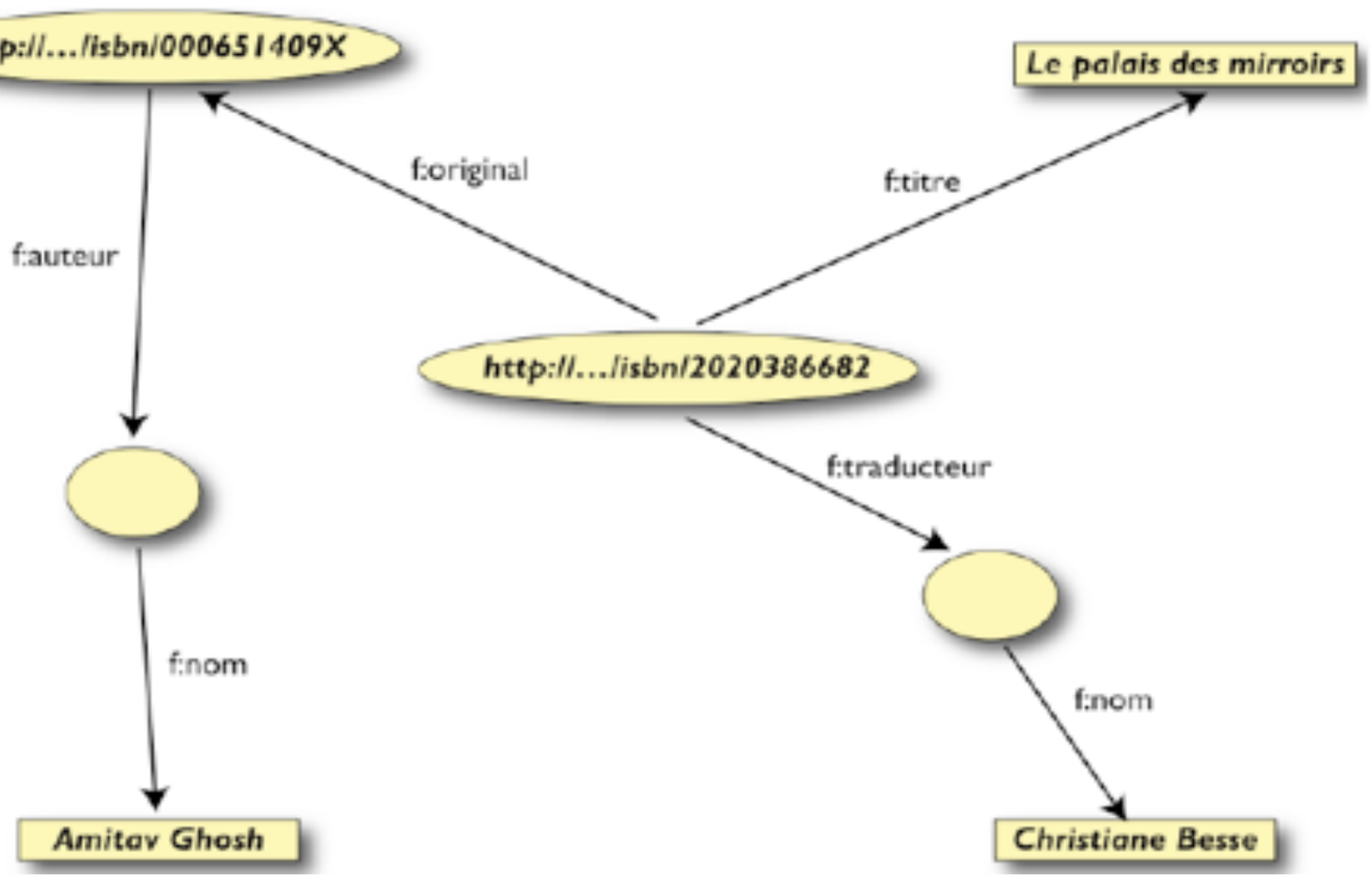


Data Integration Example

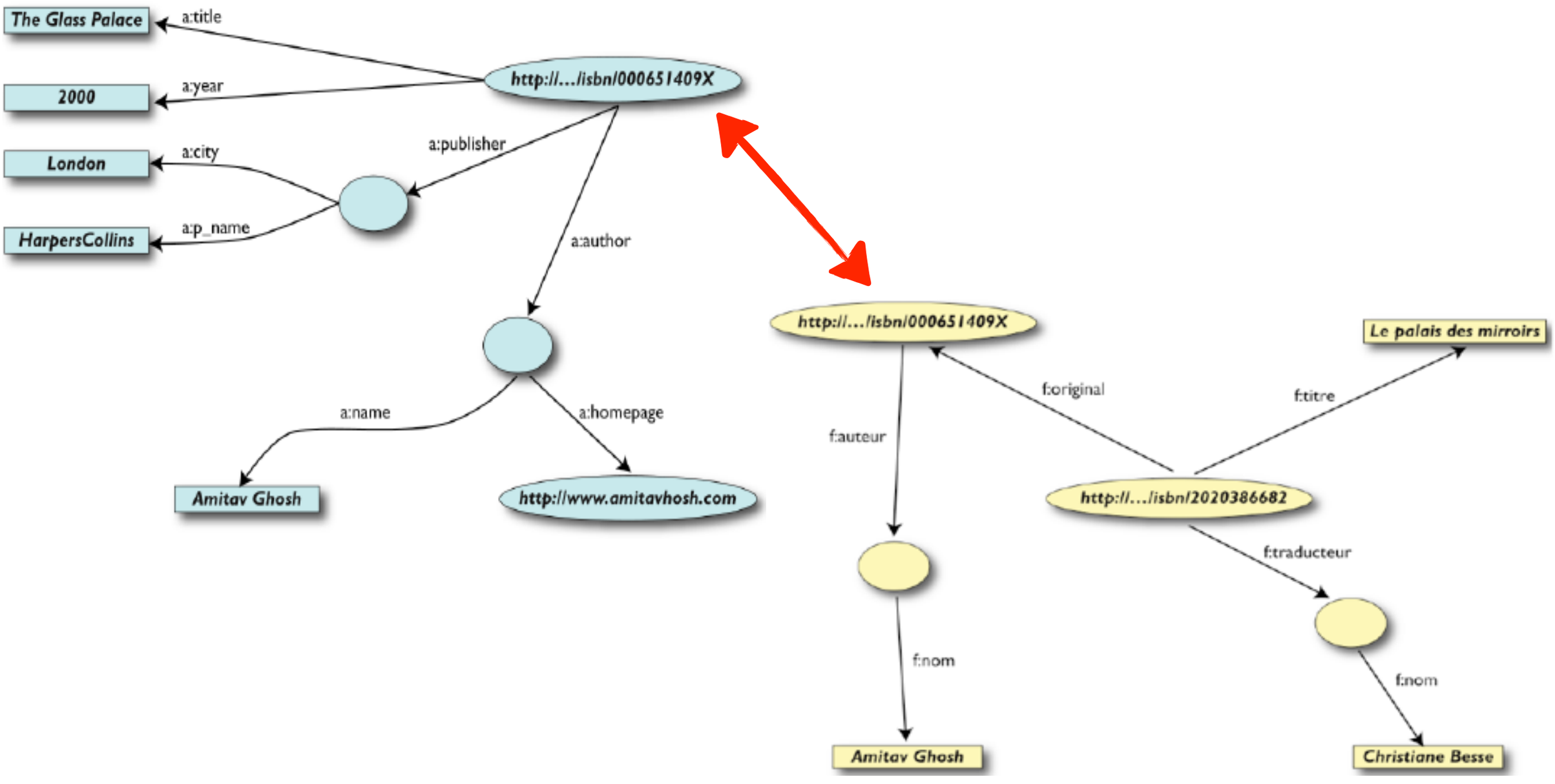
- Data in another bookstore:

ID	Titre	Auteur	Traducteur	Original
ISBN 2020386682	Le Palais des miroirs	i_abc	i_qrs	ISBN 0-00-651409-X

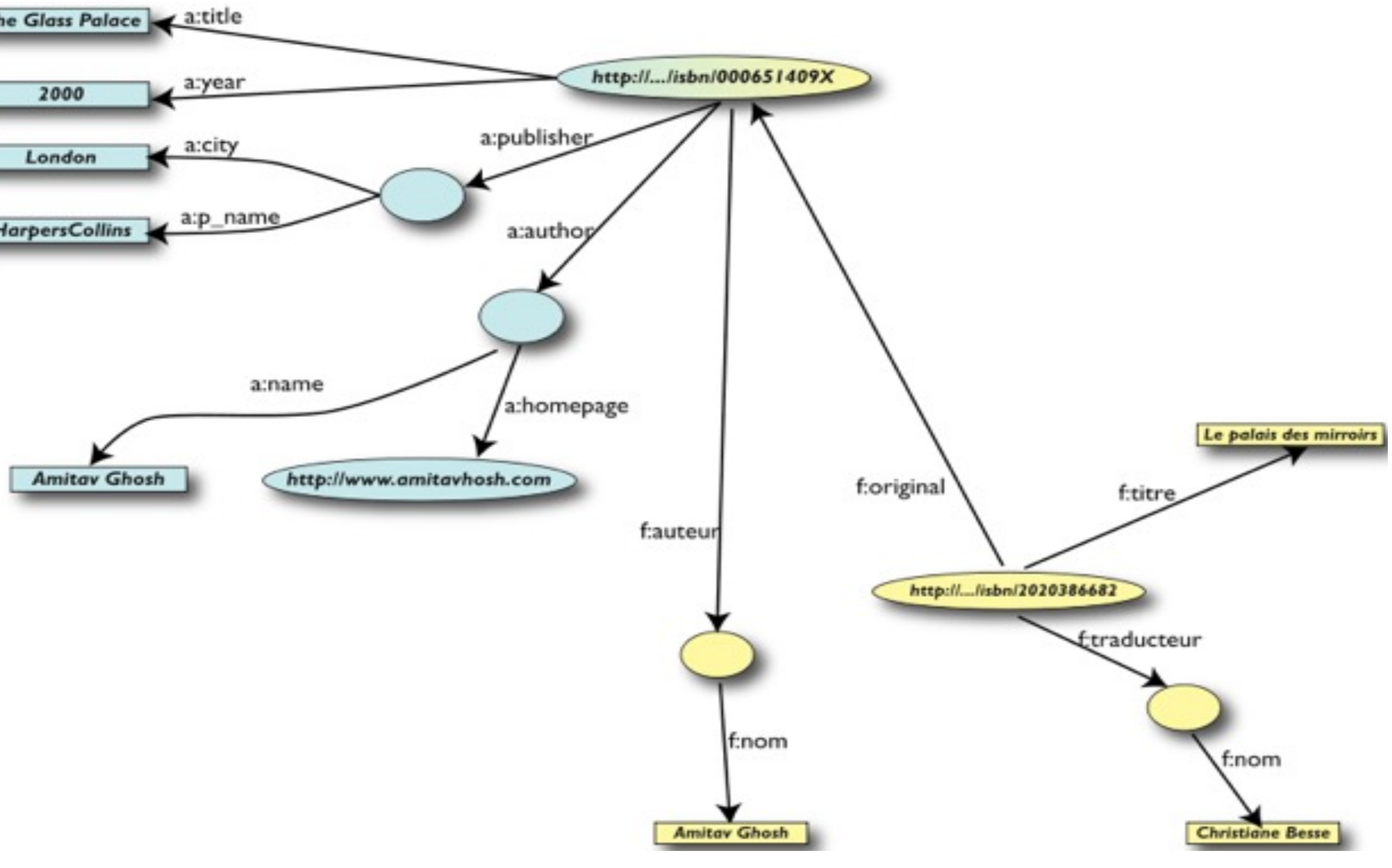
ID	Nom
i_abc	Amitav Ghosh
i_qrs	Christiane Besse



Data Integration Example



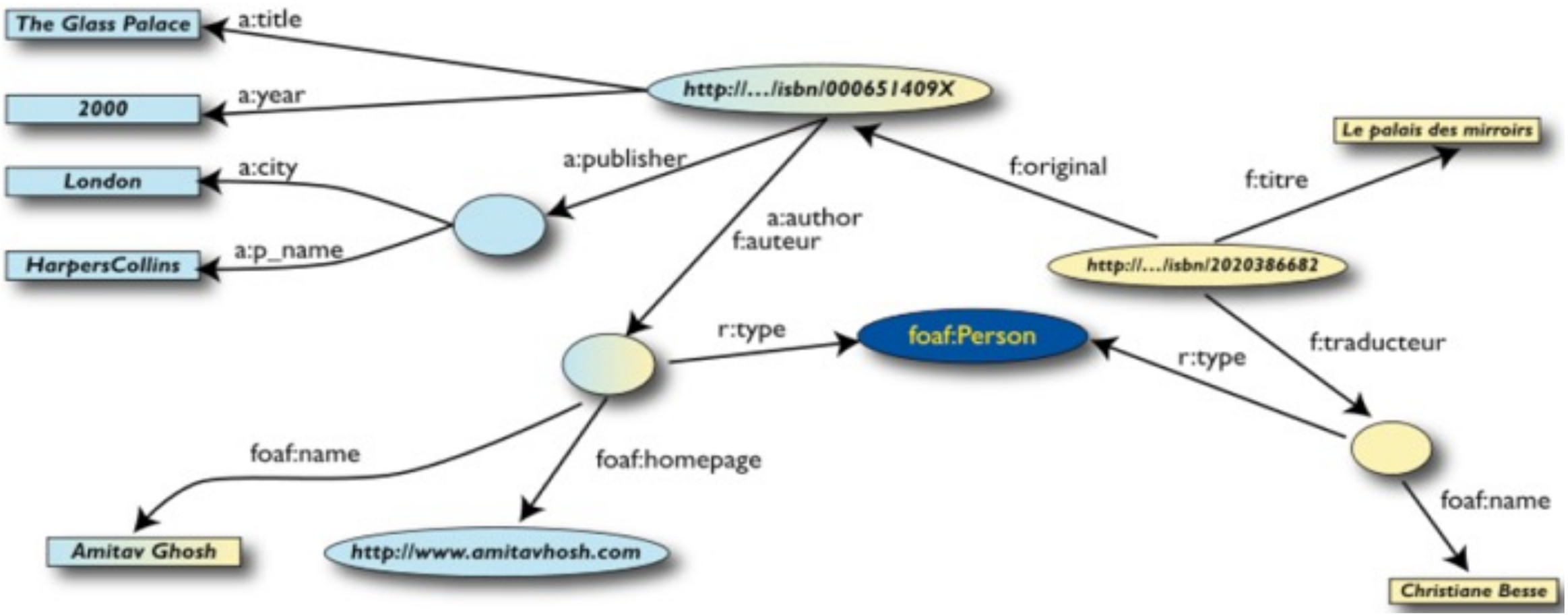
Data Integration Example



Data Integration Example

- What if we had additional knowledge?
- E.g
  - we could know that a:author is the same as f:auteur
  - We could know that in both cases they represent persons
- In other words, we could have
  - ontologies describing the concepts and relations
  - mappings of our relations into the ontologies

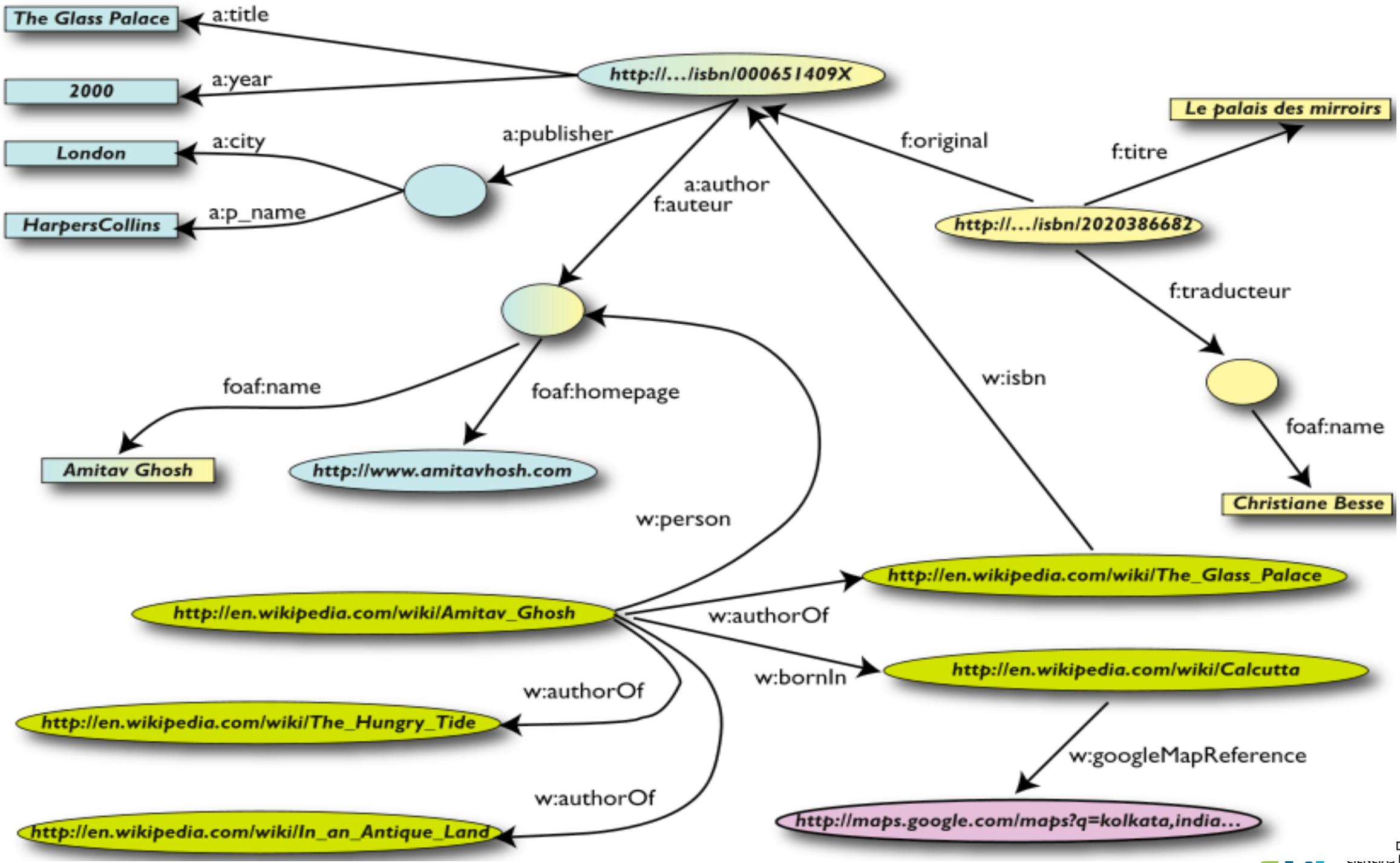
Data Integration Example

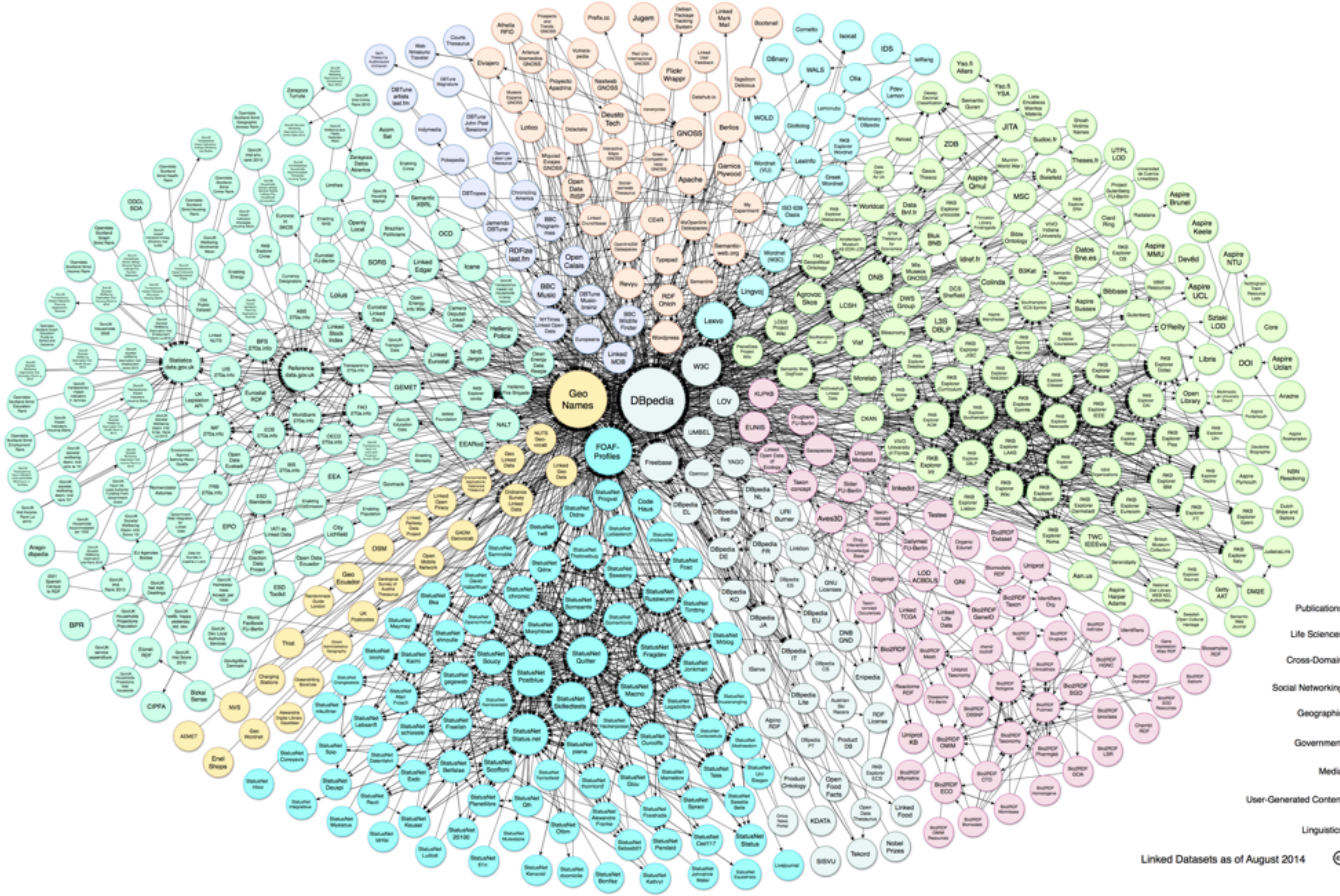


Data Integration Example

- With this merged information, richer queries are possible:
  - The second bookstore can query what is the web page of the author of the book
  - Possibly can ask about other information on the author, or translator via foaf
  - The first bookstore can query information about the translator
  - …

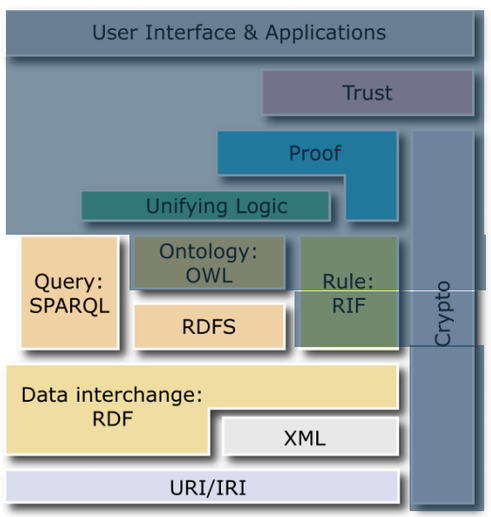
A Web of Data Example




Geo Names
DBpedia
FOAF-Profiles
W3C
LOV
UMBEL
Freebase
Lexvo
YAGO
Lingvoj
GNOSS
Apache
Jugem
Prefix.cc
Bootsnall
DBnary
WALS
Olia
IDS
lexinfo
Lexinfo
JITA
ZDB
MSC
DNB
LCSH
Viaf
Colinda
Libris
DOI
Core
Ariadne
O'Reilly
Open Library
Worldcat
Data Bnf.fr
EEA
EPO
OSM
BPR
CIPFA
AEMET
Thist
GEMET
NALT
OCD
SORS
Lotus
Icane
CKAN
linkedct
Aves3D
Testee
Revyu
Typepad
Wordpress
Reference data.gov.uk
Statistics data.gov.uk
StatusNet Status.net
StatusNet Quitter
StatusNet Postblue
StatusNet Skilledtests
Linked Datasets as of August 2014
Publication
Life Science
Cross-Domai
Social Networkin
Geographi
Governmen
Medi
User-Generated Conten
Linguistic

SPARQL

Querying the Semantic Web



SPARQL Query Language

- SPARQL - Sparql Protocol And Rdf Query Language (read as *sparkle)* provides facilities to:
  - extract information in the form of URIs, blank nodes, plain and typed literals.
  - extract RDF sub-graphs.
  - construct new RDF graphs based on information in the queried graphs.
- RDF graphs can be obtained from several sources, including middleware capable of generating RDF data.
- SPARQL is based on matching of graph patterns, using variables in the patterns
  - Results can come as binding of variables to RDF terms (URIs, literals or blank nodes) in a tabular form

Query forms

- SPARQL has four query forms:
  - **`SELECT`**, returns variable bindings
  - **`ASK`**, returns a boolean indicating whether a query pattern matches or not
  - **`DESCRIBE`**, returns a RDF graph that describes the resources found
  - **`CONSTRUCT`**, returns RDF graphs by substituting variables in a set of triple templates

Simple SELECT query

- The query consists of two parts, the `SELECT` clause and the `WHERE` clause.
  - The `SELECT` clause identifies the variables to appear in the query results.
  - The `WHERE` specifies the graph pattern to match against the RDF data graph.
- The simplest form of graph patterns are triple patterns
  - a triple pattern is an RDF triple with optional query variables in any place of the triple.
- Basic Graph Patterns (BGPs) are sets of triple patterns, in Turtle syntax

Our first SPARQL query

```
PREFIX uni: <http://fct.unl.pt/concepts/>
PREFIX  ex: <http://fct.unl.pt/example/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?x ?name
WHERE {?x ex:teaches ex:sw .
       ?x rdf:type uni:Professor .
       ?x ex:name ?name }
```

- Professor who teach `ex:sw`, and their names
- Syntax:
  - Abbreviated URIs with PREFIX
  - Variable names signalled by ? (or by $)
  - Literals are as in turtle
  - ; and , can be used as in Turtle

Our first SPARQL query

```
…
SELECT ?x ?name
WHERE {?x ex:teaches ex:sw ;
          rdf:type uni:Professor ;
          ex:name ?name }
```

- Professors who teach ex:sw, and their names

- Determines the parts of the graph that match the BGP in the `WHERE` clause
- Returns the bindings of variables in the `SELECT` clause

?x	?name
<http://fct.unl.pt/example/jja>	"Jose Alferes"

Variables everywhere!

- Variables are allowed in subject, object and also in predicate positions of triples
  - E.g. what are the relations between a guy called José Alferes and the object ex:sw?

```
…
SELECT ?relation
WHERE {?x ex:name    “Jose Alferes” .
       ?x ?relation ex:sw }
```

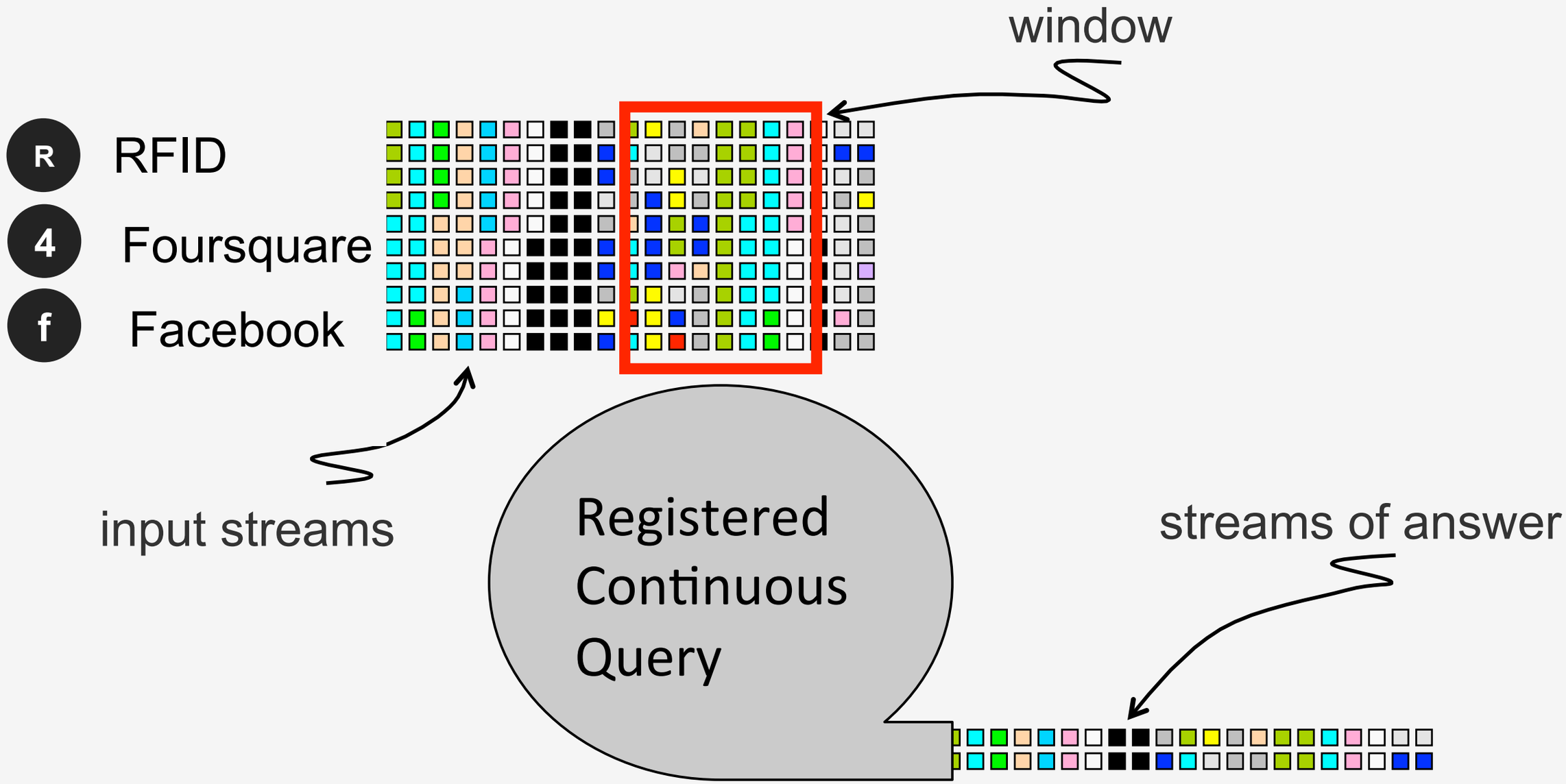
?relation
<http://fct.unl.pt/example/teaches>

Stream Reasoning research questions

- Stream Reasoning research questions
  - make sense of data streams
  - real time continuos processing
  - multiple heterogeneous sources
  - noisy and incomplete data

- Can this be done by extending the Semantic Web to represent data streams, and continuous queries?
  - Several proposal have been made in this line
    - Some are directly made over Semantic Web languages
    - Others are less attached to Semantic Web standards, and are based in different knowledge representation languages
      - But they can be translated to SW like languages as well

Running Example



Example data in the streams

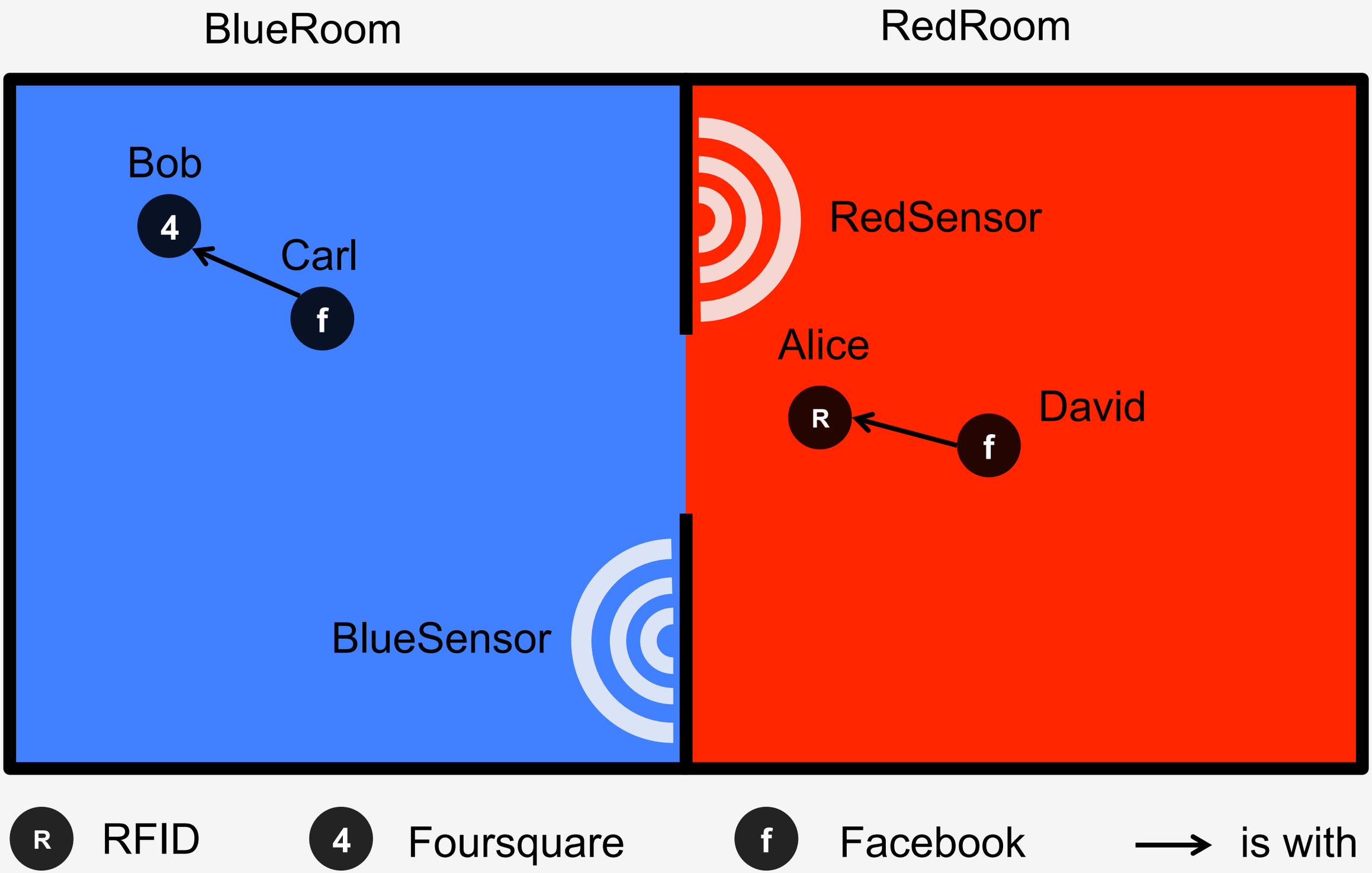
- Three ways to learn who is where

Sensor	Room	Person	Time-stamp
RedSensor	RedRoom	Alice	T_1
…	…	…	…

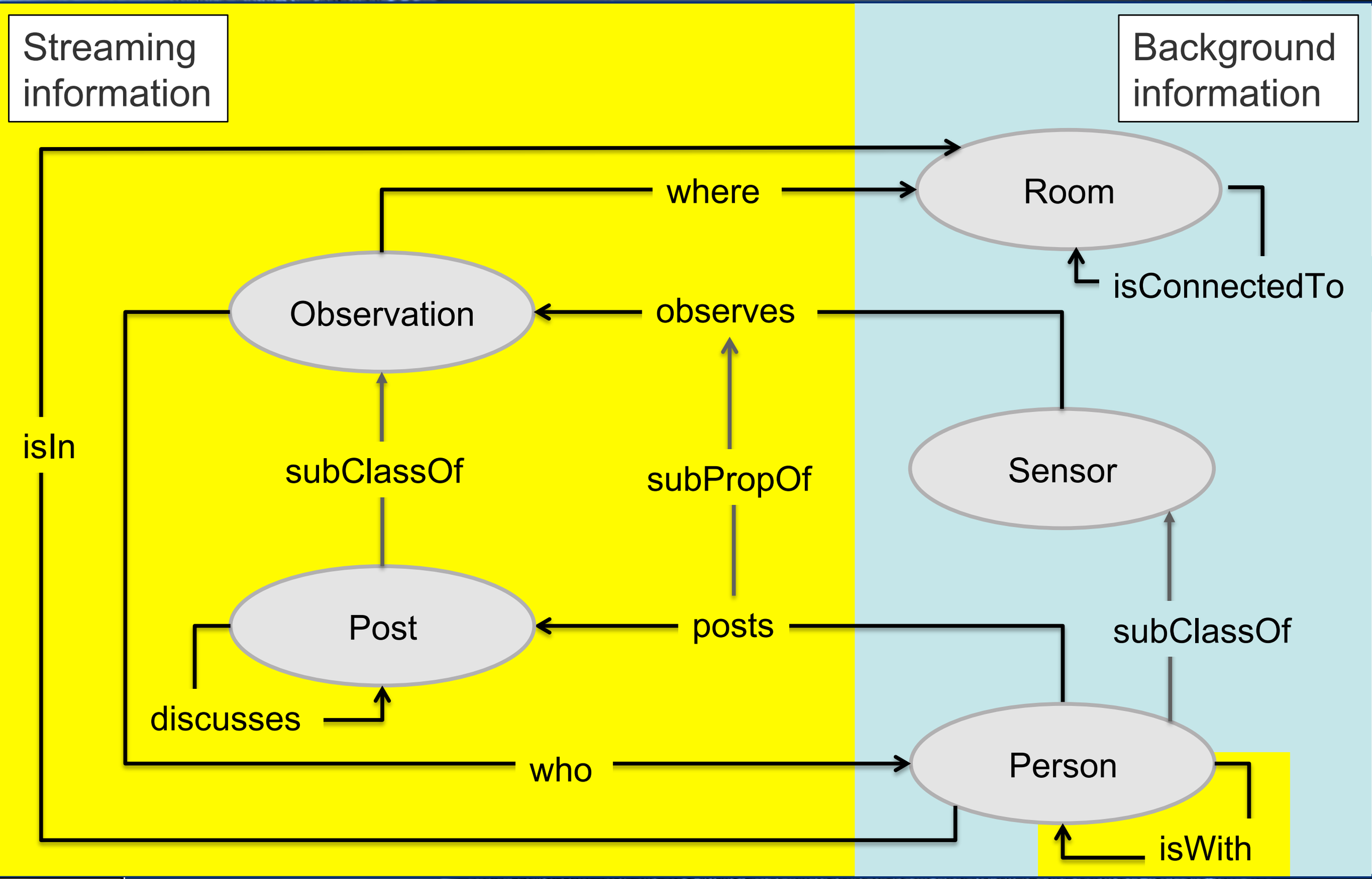
Person	ChecksIn	Time-stamp
Bob	BlueRoom	T_2
…	…	…

Person	IsIn	With	Time-stamp
Carl	null	Bob	T_2
David	RedRoom	Alice	T_3
…	…	…	…

Visualize the running example



Conceptual Integrated Model 1/3



What do we want to achieve?

- **Obtain answers that are not explicit in the data, but that can be inferred** from the data and a conceptual integrated model

Sensor	Room	Person	Time-stamp
RedSensor	RedRoom	Alice	T_1

Person	ChecksIn	Time-stamp
Bob	BlueRoom	T_2

Person	IsIn	With	Time-stamp
Carl	null	Bob	T_2
David	RedRoom	Alice	T_3

Carl is in the BlueRoom at T_2

David is in the RedRoom at T_3

Conceptual Integrated Model 2/3

- Important Axioms
  - *posts* is a subproperty of *observes*
    - It allows
      - to represent posts, checks-in and observations
        - RedSensor observes [who Alice; where RedRoom] .
        - Bob posts [who Bob; where BlueRoom]
      - to homogenous access them
        - s? observes [who ?p; where ?r]

          implied by
        - s? posts [who ?p; where ?r]
  - *isWith* is a composition of *posts* and *who*
    - It allows to conclude that
      - Carl isWith Bob
    - From
      - Bob posts [who Carl]

(see next slide)

Conceptual Integrated Model

(continuous from previous slide)

- *isIn* is a composition of *posts* and *where*
  - It allows to conclude that
    - Carl isIn BlueRoom
  - From
    - Bob posts [where BlueRoom]
- *isIn* is also a composition of *isWith* and *isIn*
  - It allows to conclude that
    - Carl isIn BlueRoom
  - From
    - Bob isWith Carl
    - Carl isIn BlueRoom

Stream Reasoning for Linked Data

M. Balduini, J-P Calbimonte, O. Corcho,
D. Dell'Aglio, E. Della Valle

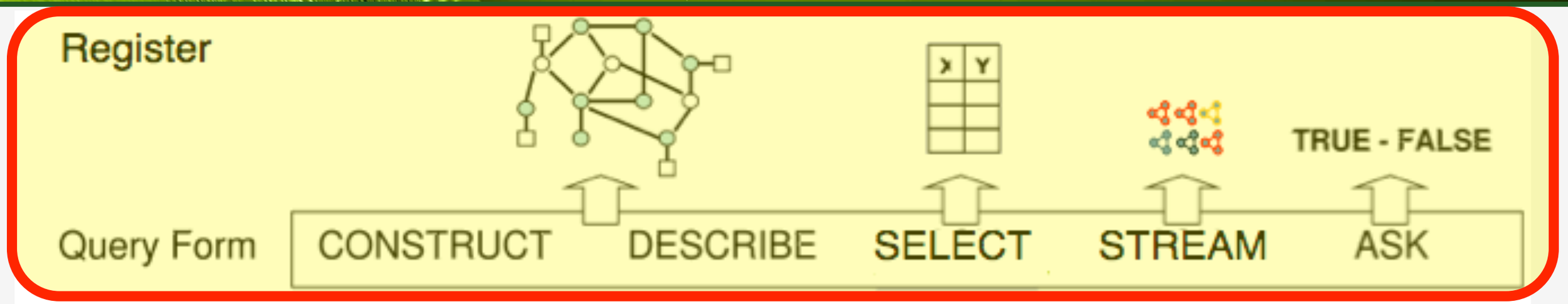
http://streamreasoning.org/events/sr4ld2014

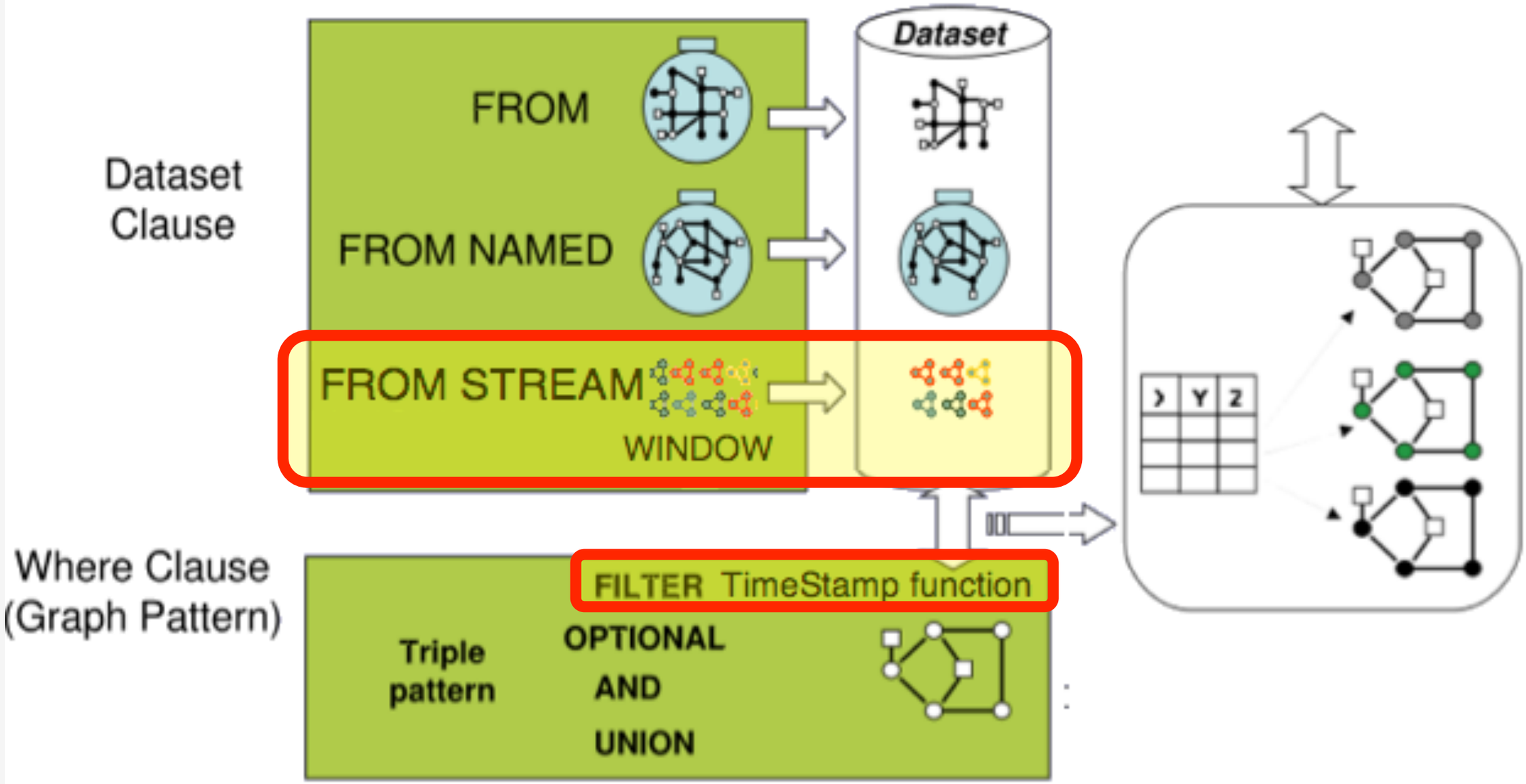
ISWC 2014
Riva del Garda - Trentino, Italy

C-SPARQL: A Continuous Extension of SPARQL

Marco Balduini
marco.balduini@polimi.it

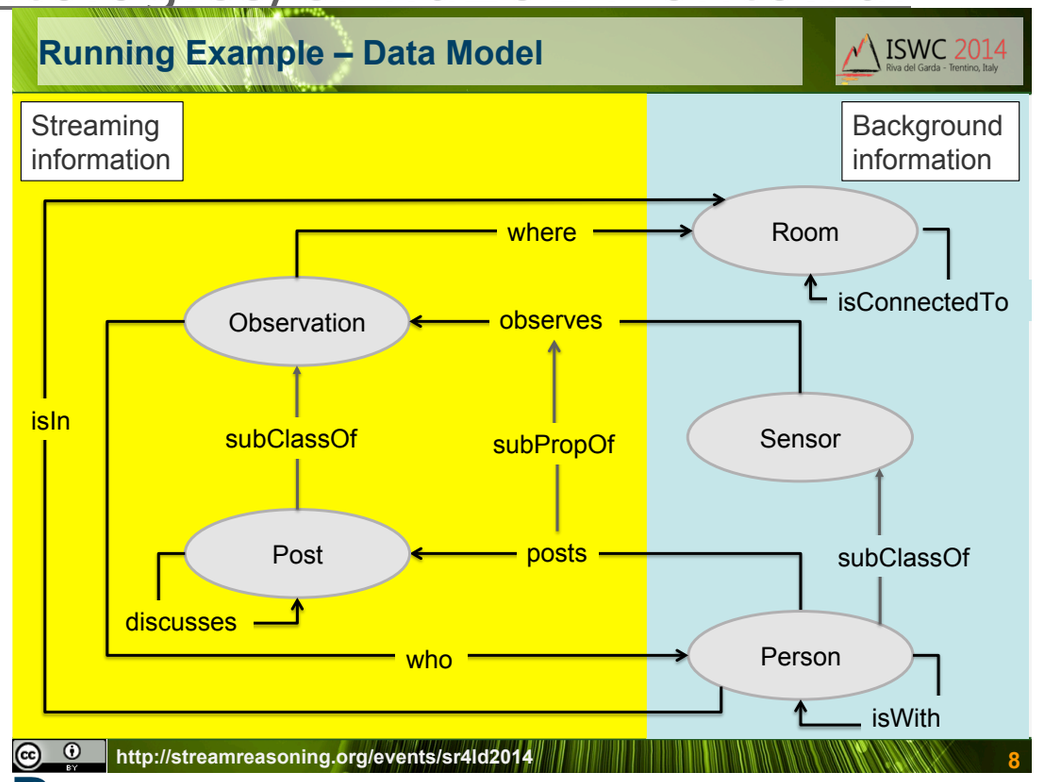
Where C-SPARQL Extends SPARQL





Background Information

- The Ontology
  - http://www.streamreasoning.org/ontologies/sr4ld2014-onto.rdf
- The Static Instances
  - :Alice a :Person .
  - :Bob a :Person .
  - :Carl a :Person .
  - :David a :Person .
  - :Elena a :Person .
  - :RedRoom a :Room .
  - :BlueRoom a :Room .
  - :RedRoom :isConnectedTo :BlueRoom .
  - :RedSensor a :Sensor .
  - :BlueSensor a :Sensor .



Streaming Information

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

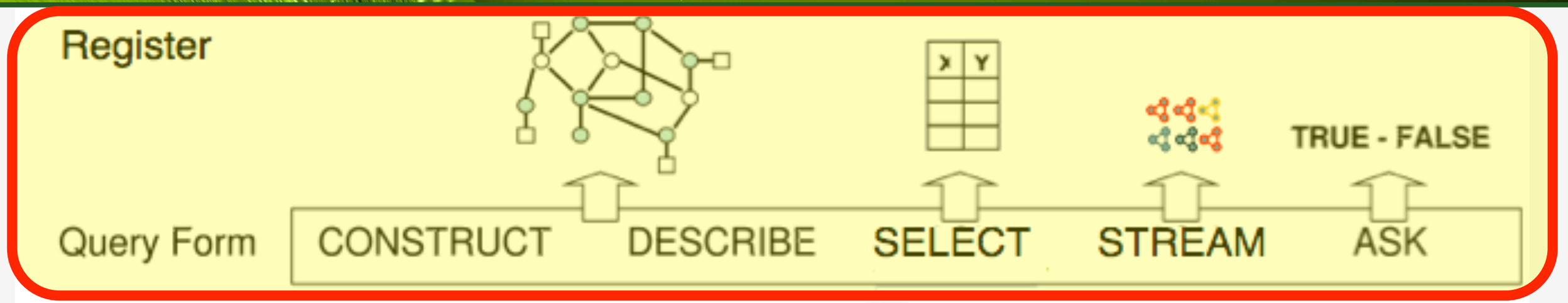
$$\ldots$$
$$(\langle subj_i, pred_i, obj_i \rangle \ , \tau_i)$$
$$(\langle subj_{i+1}, pred_{i+1}, obj_{i+1} \rangle \ , \tau_{i+1})$$
$$\ldots$$

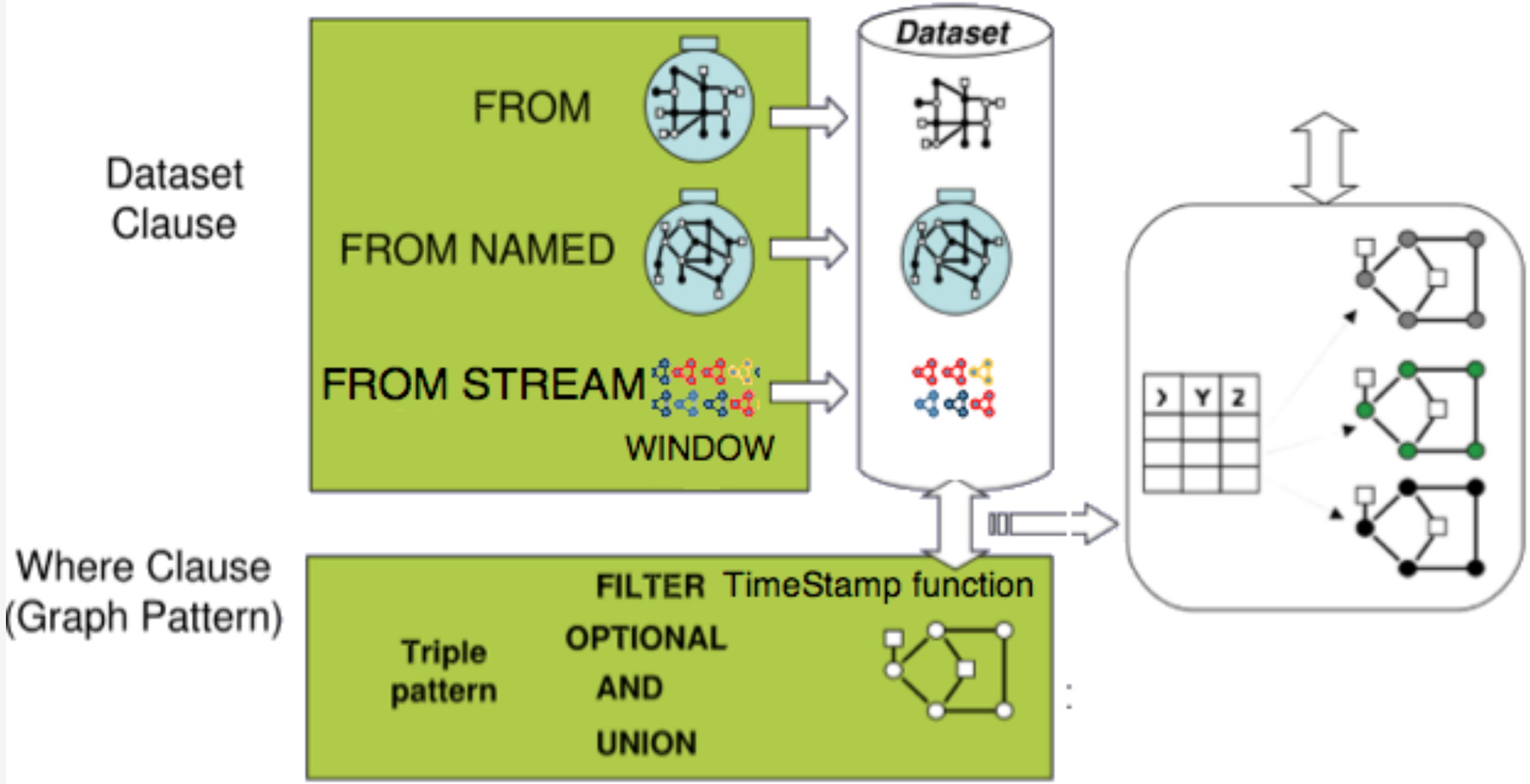
    - Timestamps are not required to be unique, they must be non-decreasing

- E.g.,

Streamed RDF graph	Time-stamp	Stream
:RedSensor :observes [:who :Alice; :where :RedRoom] .	T_1	rfid
:Bob :posts [:who :Bob ; :where :BlueRoom] .	T_2	fs
:Carl :posts [:who :Carl , :Bob] .	T_2	fb
:David :posts [:who :David , :Elena ; :where :RedRoom]	T_3	fb

Query and Stream Registration





Query and Stream Registration

- All C-SPARQL queries over RDF streams are continuous
  - Registered through the REGISTER statement
- The output of queries is in the form of
  - Instantaneous tables of variable bindings
  - Instantaneous RDF graphs
  - RDF stream
- Only queries in the CONSTRUCT form can be registered as generators of RDF streams

$$Registration \rightarrow \text{`REGISTER'}\ (\text{`QUERY'} |\ \text{`STREAM'})\ QueryName\ \text{`AS'}\ Query$$

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

Query registration - Example

- Using the social stream fb, Who is where?

```
REGISTER QUERY QWhoIsWhereOnFb AS
PREFIX : <http://…/sr4ld2014-onto#>
SELECT ?room ?person
FROM STREAM <http://…/fb> [RANGE 1m STEP 10s]
WHERE {
   ?person1 :posts [ :who ?person ; :where ?room ] .
}
```

- The resulting variable bindings has to be interpreted as an instantaneous. It expires as soon as the query is recomputed

Stream registration - Example

- Results of a C-SPARQL query can be stream out for down stream queries

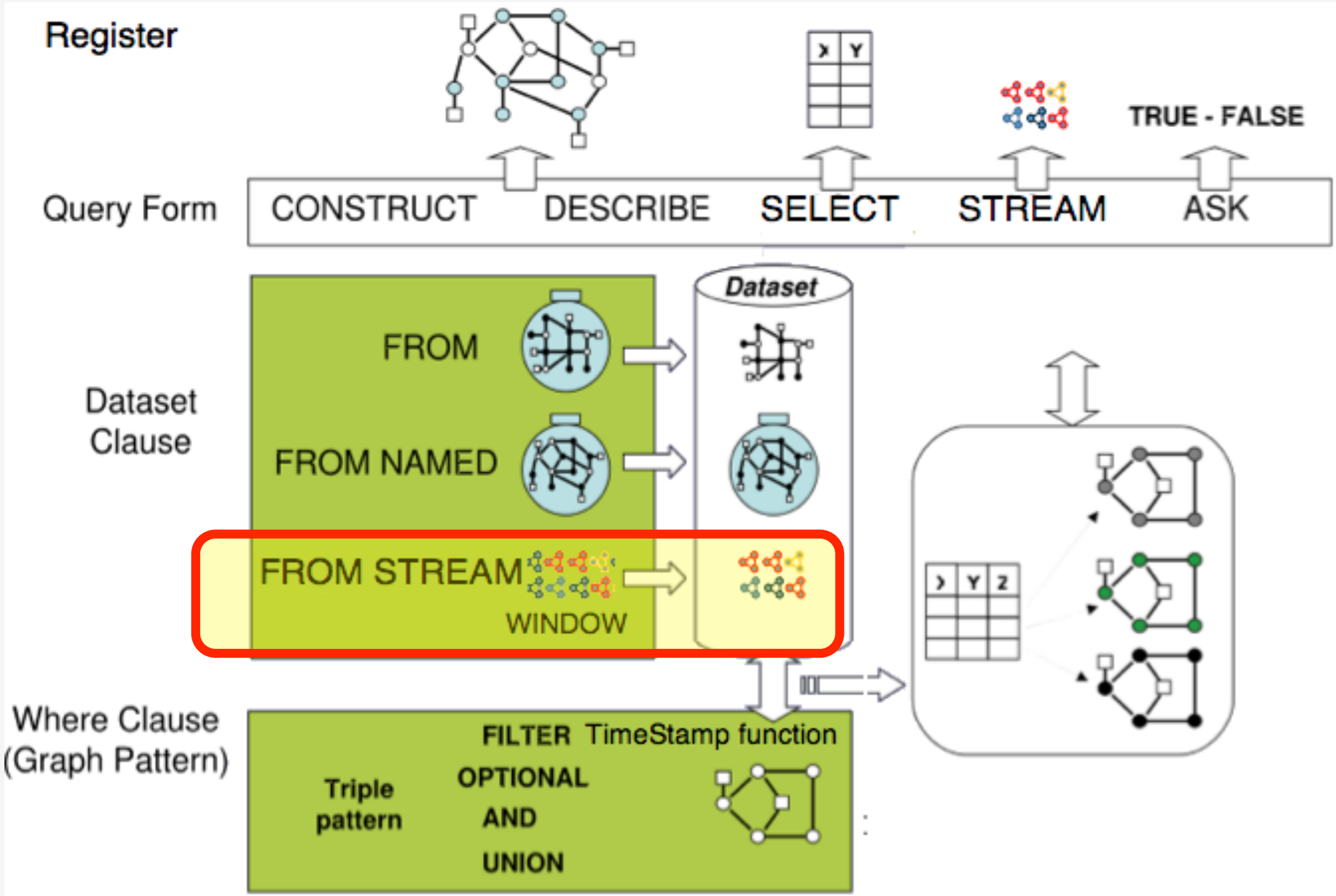
```
REGISTER STREAM SWhoIsWhereOnFb AS
PREFIX : <http://…/sr4ld2014-onto#>
CONSTRUCT { ?person :isIn ?room }
FROM STREAM <http://…/fb> [RANGE 1m STEP 10s]
WHERE {
  ?person1 :posts [ :who ?person ; :where ?room ] .
}
```

- The resulting RDF triples are streamed out on an RDF stream
  - More details in the C-SPARQL Engine hands-on session

Stream Registration - Notes

- 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.

FROM STREAM Clause



FROM STREAM Clause

- FROM STREAM clauses are similar to SPARQL datasets
  - They identify RDF stream data sources
  - They represent windows over a RDF stream
- They define the RDF triples available for querying and filtering.

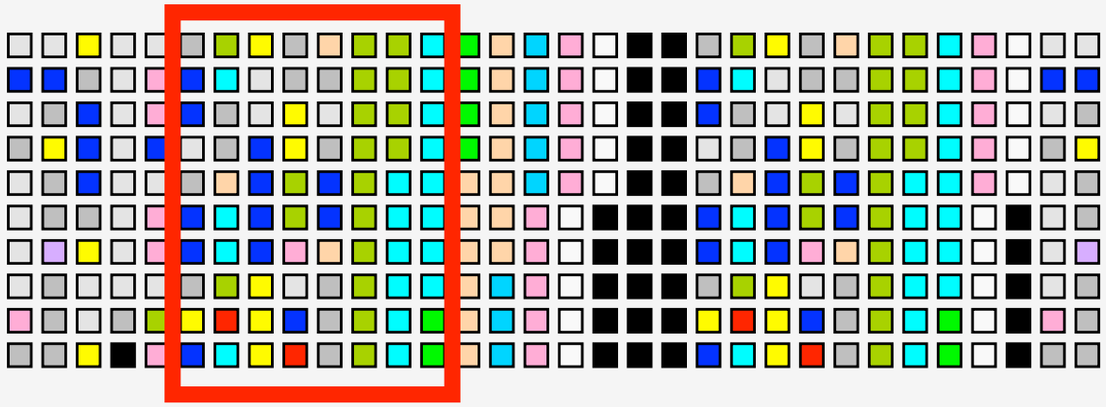
FROM STREAM Clause - Windows

- Grammar of the FROM STREAM clause

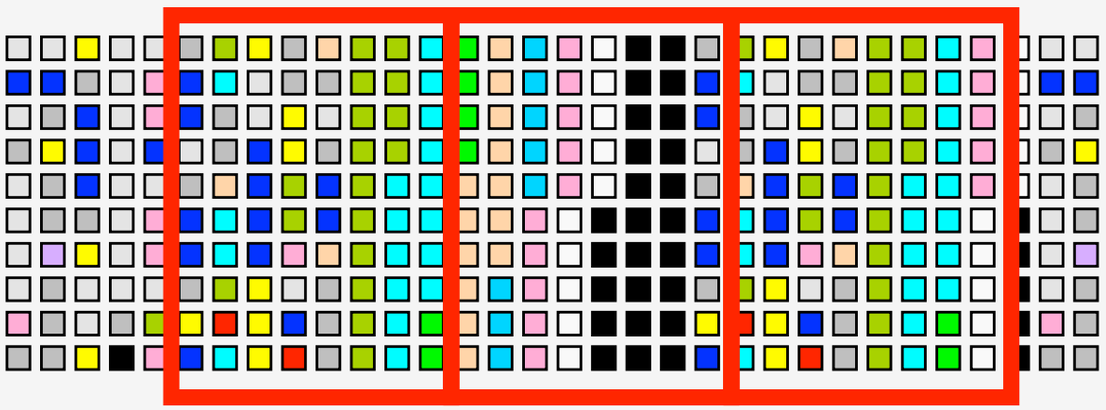
$$
\begin{aligned}
\mathit{FromStrClause} &\longrightarrow \text{`FROM'}\ \text{`STREAM'}\ \mathit{StreamIRI}\ \text{`[ RANGE'}\ \mathit{Window}\ \text{`]'} \\
\mathit{Window} &\longrightarrow \mathit{LogicalWindow} \mid \mathit{PhysicalWindow} \\
\mathit{LogicalWindow} &\longrightarrow \mathit{Number}\ \mathit{TimeUnit}\ \mathit{WindowOverlap} \\
\mathit{TimeUnit} &\longrightarrow \text{`ms'} \mid \text{`s'} \mid \text{`m'} \mid \text{`h'} \mid \text{`d'} \\
\mathit{WindowOverlap} &\longrightarrow \text{`STEP'}\ \mathit{Number}\ \mathit{TimeUnit} \mid \text{`TUMBLING'} \\
\mathit{PhysicalWindow} &\longrightarrow \text{`TRIPLES'}\ \mathit{Number}
\end{aligned}
$$

FROM STREAM Clause - windows

- 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



FROM STREAM Clause - Example

- Using the social stream fb, how many people are in the same room? Count on a window of 1 minute that slides every 10 seconds

```
REGISTER QUERY HowManyPoepleAreInTheSameRoom AS
PREFIX : <http://…/sr4ld2014-onto#>
SELECT ?room (COUNT(DISTINCT ?s) as ?person)
FROM STREAM <http://…/fb> [RANGE 1m STEP 10s]
WHERE {
   ?person1 :posts [ :who ?person ; :where ?room ] .
}
GROUP BY ?room
```

Accessing background Information

- C-SPARQL allows for asking the engine to issue the query also against RDF graphs using the FROM clauses.
- E.g., Where else can Alice go?

```
REGISTER QUERY WhereElseCanAliceGo AS
PREFIX : <http://…/sr4ld2014-onto#>
SELECT ?room
FROM STREAM <http://…/isIn> [RANGE 10m STEP 10m]
FROM <http://…/bgInfo>
WHERE {
   ?:Alice :isIn ?someRoom .
   ?someRoom :isConnectedTo ?room .
}
```

IRI identifying the graph containing the background information

C-SPARQL Engine Architecture

- Simple, modular architecture
- It relies entirely on existing technologies
- Integration of
  - DSMSs (Esper) and
  - SPARQL engines (Jena-ARQ)