RDF datastores

Dimitrios Tsoumakos

Some slides taken from :

- Triple Stores, Dr. Stephan Volmer
- Storing and querying RDF data, Khriyenko Oleksiy
- Distributed Big Graph Management Methods and Systems, N. Papailiou

Part I

INTRO TO SEMANTIC WEB - RDF - SPARQL

Sematic Web

The tale of unstructured data and standardized metadata…

- Unstructured data is becoming more and more common
- How do we best handle unstructured data?

➢ Relational databases are not the answer!

(more on that later)

- Metadata helps describe the content of unstructured data
- Creating a standard will help push forward the semantic web

Semantic Web

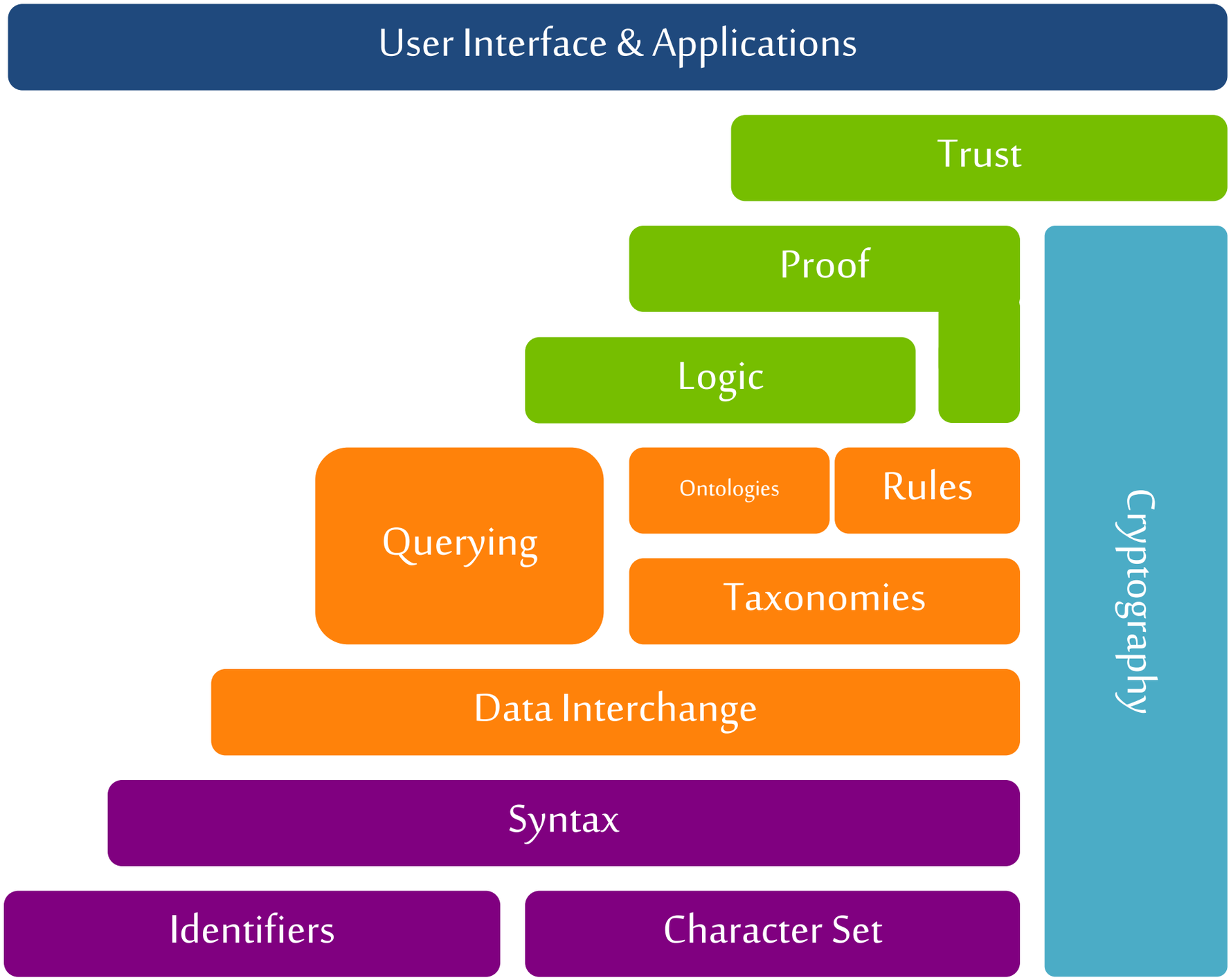
The semantic web is a web of data!

- Collaborative movement led by the international standards body, the World Wide Web Consortium (W3C)

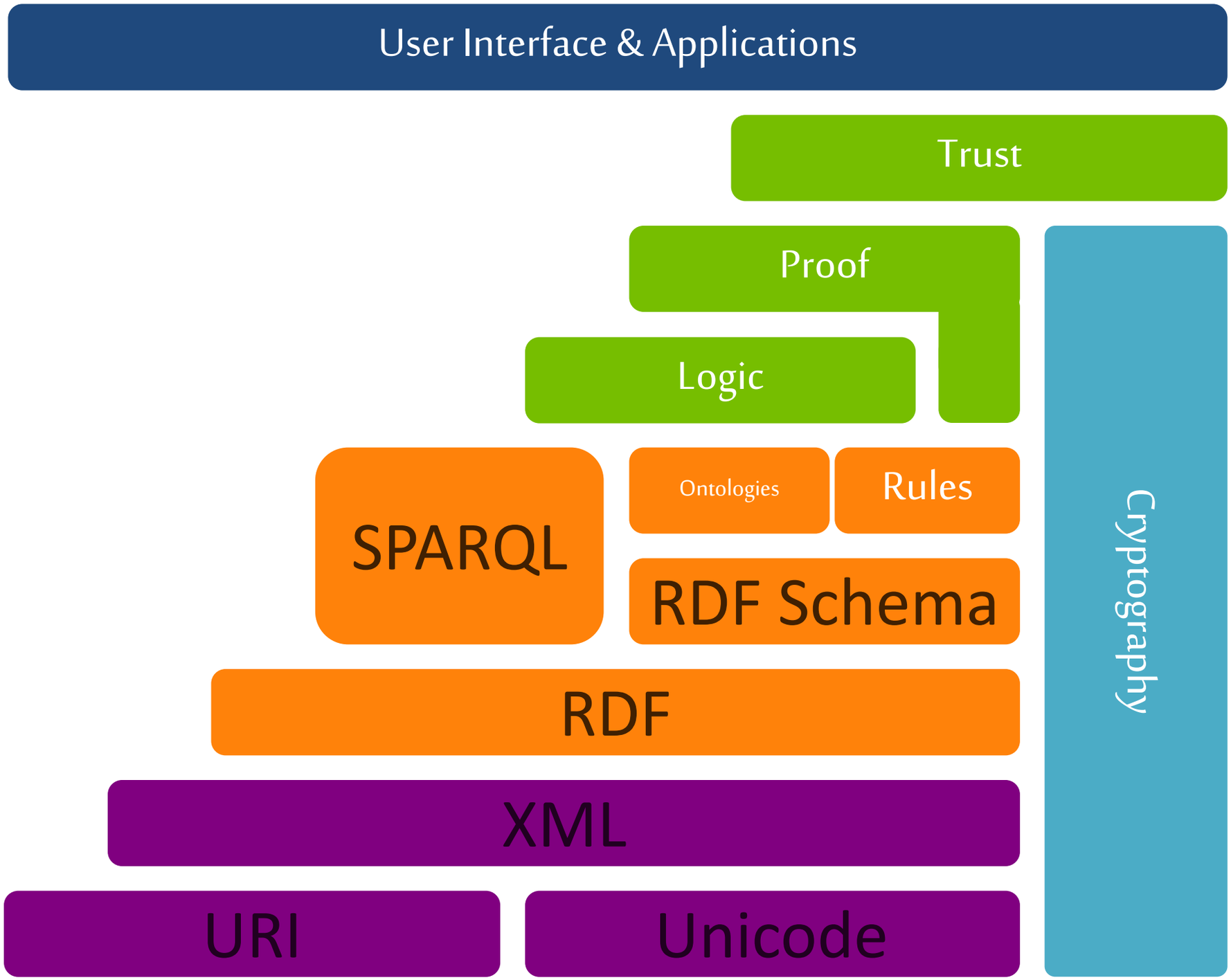
  - Promotes common data formats on the World Wide Web
  - Aims at converting the current web dominated by unstructured and semi-structured documents into a "web of data"

"The Semantic Web provides a common framework that allows data to be shared and reused across applications, enterprises, and community boundaries."

The Semantic Web Stack



The Semantic Web Stack



RDF

Resource Description Framework

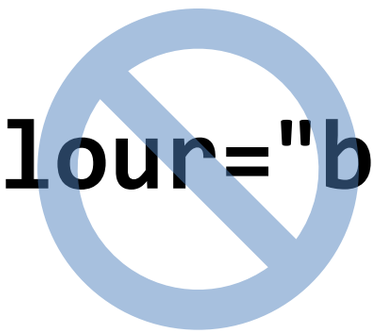
– Family of standards from W3C

`http://www.w3c.org/RDF/`

– Make statements about things

The sky has the colour blue

– Syntax is XML



```
<sky colour="blue"/>
```

RDF

RDF makes statements about things

The sky has the colour blue

thing property value

triple

RDF

- Triple data structure is a simple EAV model
- Any data structure can be represented as triples

<entity>	<attribute-value>	<value>
E	A	V
<subject>	<predicate>	<object>

- RDF uses different terminology

RDF

URIs in RDF

- Provide namespaces to uniquely name the things we want to talk about

- Provide a way to identify the properties and types of things in a way that is sharable and unique

- Anyone can say anything about any things with a shared URI identifier

RDF

- Subject and Predicate are always URIs
- Object is either a literal or another URI

```
<http://www.zuehlke.com/person/stv>          subject
      <http://www.zuehlke.com/properties/name>      predicate
            "Stephan Volmer"                              object

<http://www.zuehlke.com/person/stv>
      <http://www.zuehlke.com/properties/business-unit>
            <http://www.zuehlke.com/business-units/dns>

<http://www.zuehlke.com/person/stv>
      <http://www.zuehlke.com/properties/email>
            "stephan.volmer@zuehlke.com"
```

RDF

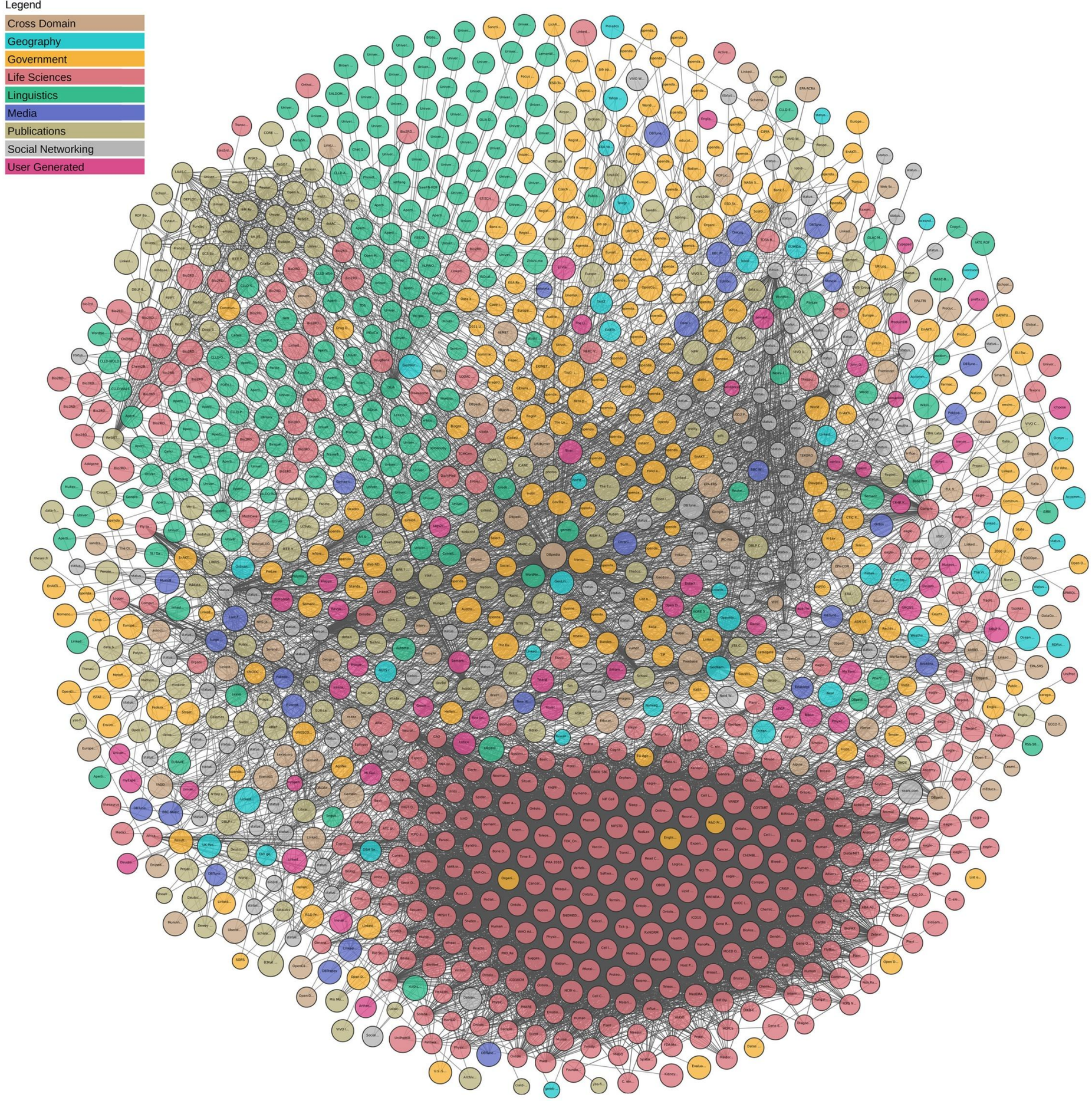
RDF graphs are the glue to make it work.

- Logical collection of triples
- One store may contain many graphs
- Named graphs are RDF graphs with a URI name often called the context
- Graphs can be targeted
  - import data into a graphs
  - export a graph
  - query / update data in graph
  - merging data from different sources
  - controlling access to data

RDF data evolution

2007 2008 2009 2011

30 Apr. 2018
1184 datasets



The Linked Open Data Cloud from lod-cloud.net

Real-life RDF data

RDF-encoded Wikipedia	1.89 billion triples
RDF-encoded biological data	2.7 billion triples
US government data in RDF	5 billion triples
Crawled Web data	2 billion triples
US population statistics	1 billion triples
Yago facts from Wikipedia, Wordnet, Geonames	0.12 billion triples
Linked Open Data cloud	30 billion triples

RDF

Say good-bye to schema-free, say hello to schema-less

- Core RDF is schema-free
- Any shape of data can be poured into a triple store
- Sometimes a common ontology is helpful for sharing and reusing knowledge bases

RDF Schema

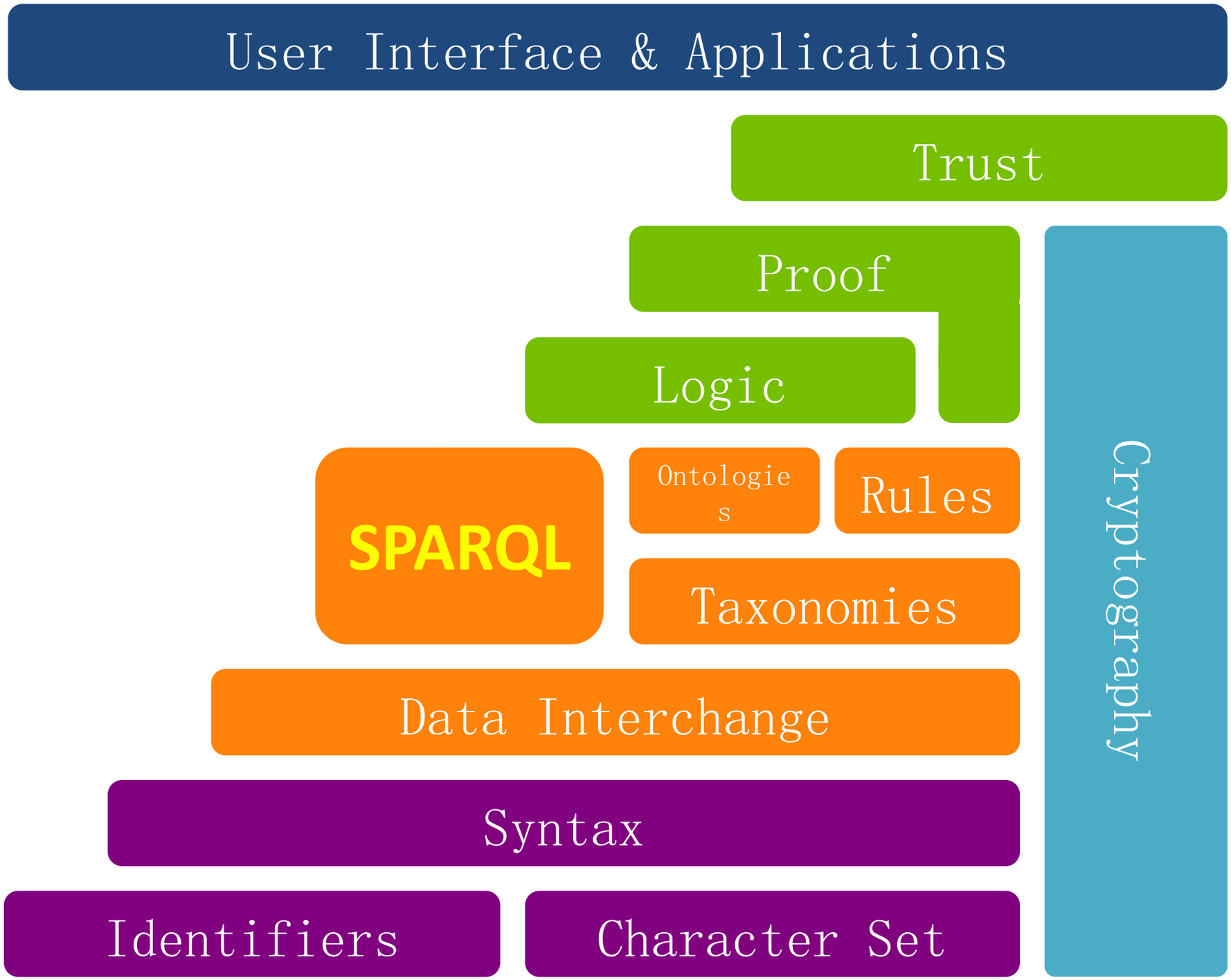
- Set of RDF properties for defining types and their constraints
- RDF schema is expressed as RDF

SPARQL

- Γλώσσα επερώτησης των RDF δεδομένων
- Βασικό στοιχείο τα triple patterns
  - Triples που μπορούν να περιέχουν μεταβλητές
  - π.χ. ?person rdf:type foaf:Person
- SparQL ερωτήματα: Συνδυασμός από BGP
  - Παράδειγμα:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?name ?email
WHERE {
  ?person rdf:type foaf:Person .
  ?person foaf:name ?name .
  ?person foaf:mbox ?email .
}
```

SPARQL


User Interface & Applications
Trust
Proof
Logic
SPARQL
Ontologies
Rules
Taxonomies
Data Interchange
Syntax
Identifiers
Character Set
Cryptography

SPARQL

- SPARQL is pronounced “sparkle”
- SPARQL is a recursive acronym for

  SPARQL **P**rotocol **a**nd **R**DF **Q**uery **L**anguage
- SPARQL became an official W3C recommendation in 2008
- SPARQL allows for a query to consist of
  - triple patterns,
  - conjunctions,
  - disjunctions, and
  - optional patterns

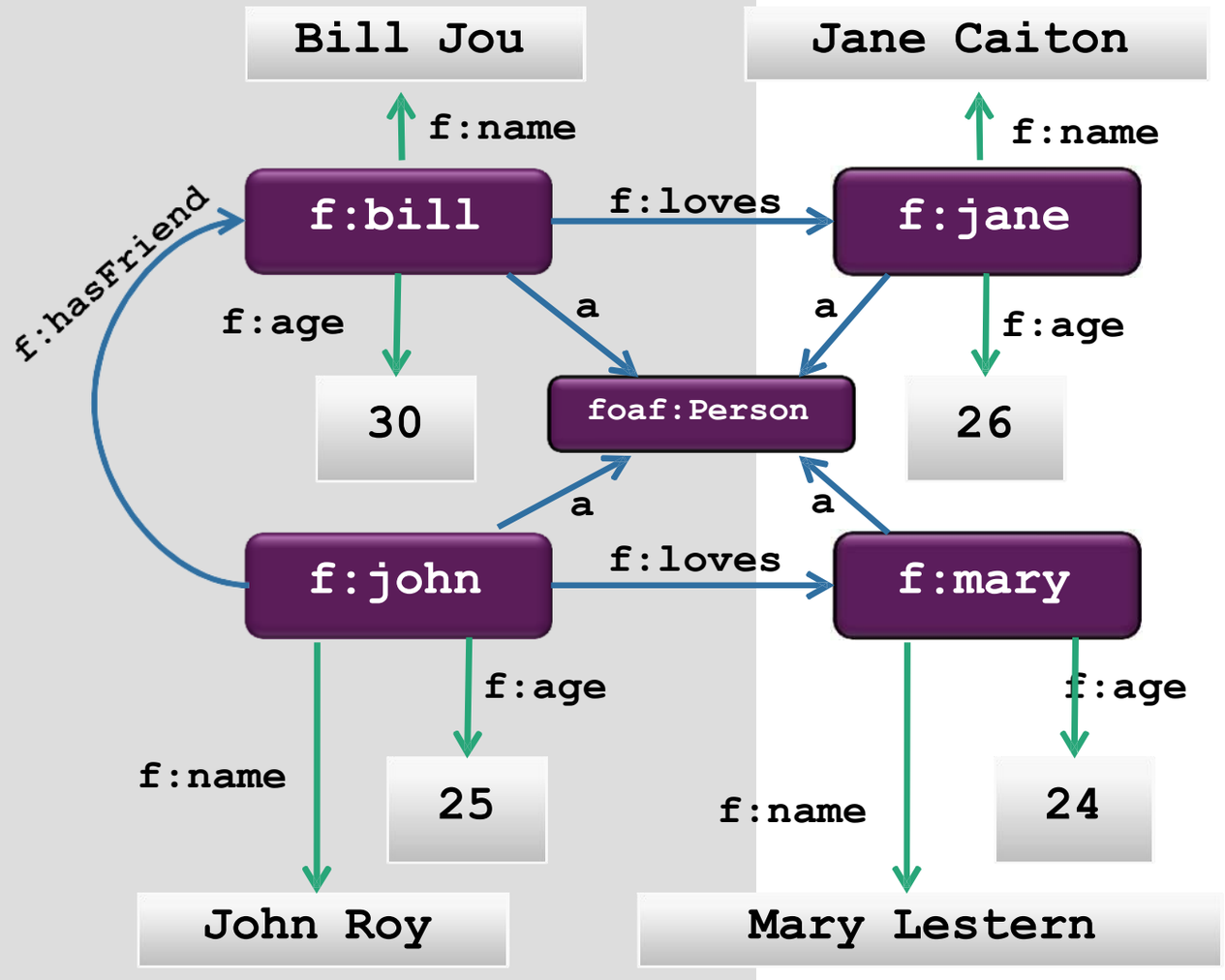
SPARQL: General Form

- SPARQL queries take the following general form

```
PREFIX (Namespace Prefixes)
e.g. PREFIX f: <http://example.org#>

SELECT (Result Set)
e.g. SELECT ?age

FROM (Data Set)
e.g. FROM <http://users.jyu.fi/~olkhriye/itks544/rdf/people.rdf>

WHERE (Query Triple Pattern)
e.g. WHERE { f:mary f:age ?age }

ORDER BY, DISTINCT, etc. (Modifiers)
e.g. ORDER BY ?age
```

Example data set

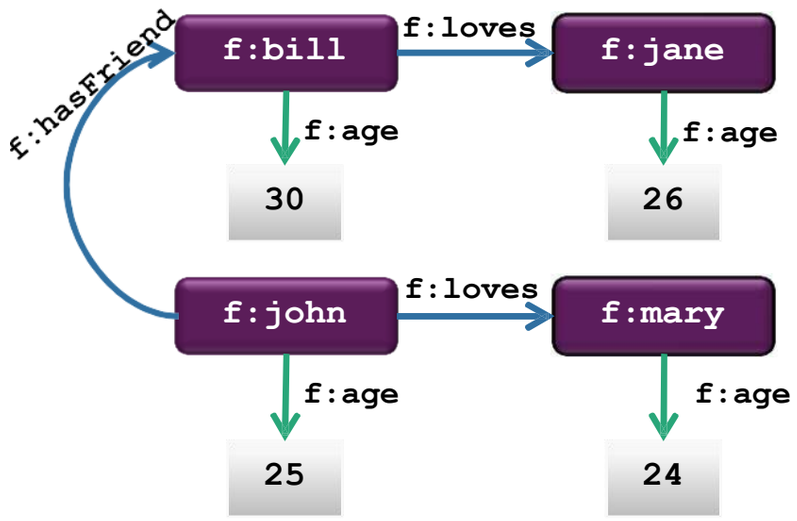
```
@prefix  f: <http://example.org#> .
@prefix  xsd:
@prefix  <http://www.w3.org/2001/XMLSchema#> .
         foaf: <http://xmlns.com/foaf/0.1/>.
f:john a foaf:Person .
f:bill a foaf:Person .
f:mary a foaf:Person .
f:jane a foaf:Person .
f:john f:age "25"^^xsd:int  .
f:bill f:age "30"^^xsd:int  .
f:mary f:age "24"^^xsd:int  .
f:jane f:age "26"^^xsd:int  .
f:john f:loves f:mary .
f:bill f:loves f:jane .
f:john f:hasFriend f:bill.
f:john f:name "John Roy" .
f:bill f:name "Bill Jou" .
f:mary f:name "Mary Lestern" .
f:jane f:name "Jane Caiton" .
f:bill foaf:name "Bill" .
f:john foaf:name "John" .
f:mary foaf:name "Mary" .
f:jane foaf:name "Jane" .
```



Simple SPARQL queries (1)

- Show me the property ***f:age*** of resource ***f:mary***

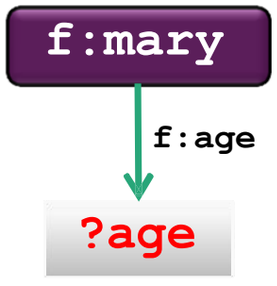
Data



Query

```
SELECT ?age
WHERE { <http://example.org#mary>
<http://example.org#age>   ?age }
```

```
PREFIX f: <http://example.org#>
SELECT ?age
WHERE { f:mary f:age ?age }
```



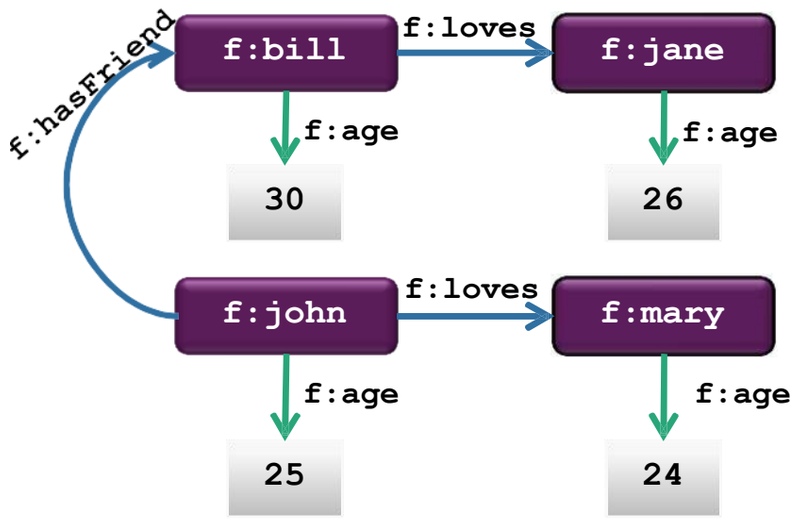
Result

age
24

Simple SPARQL queries (2)

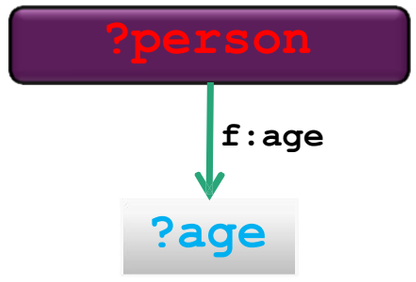
- Show me ***f:age*** of all resources

Data



Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age
WHERE { ?person f:age ?age }
```



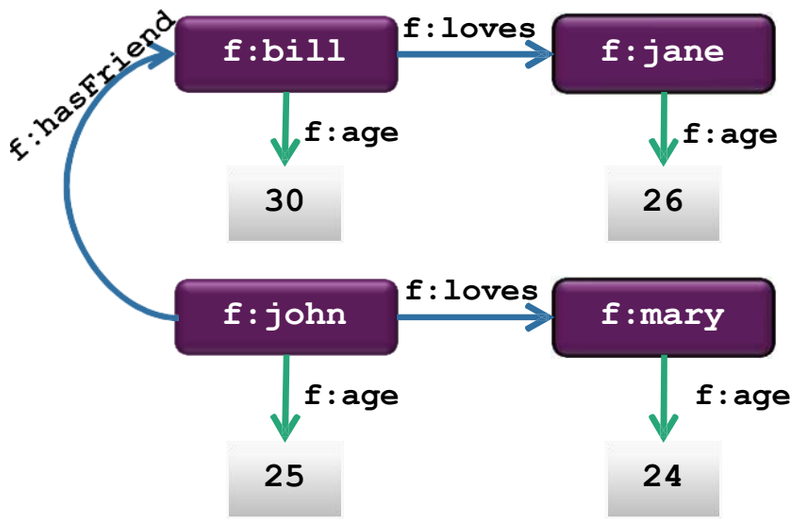
Result

person	age
f:bill	30
f:jane	26
f:john	25
f:mary	24

Simple SPARQL queries (3)

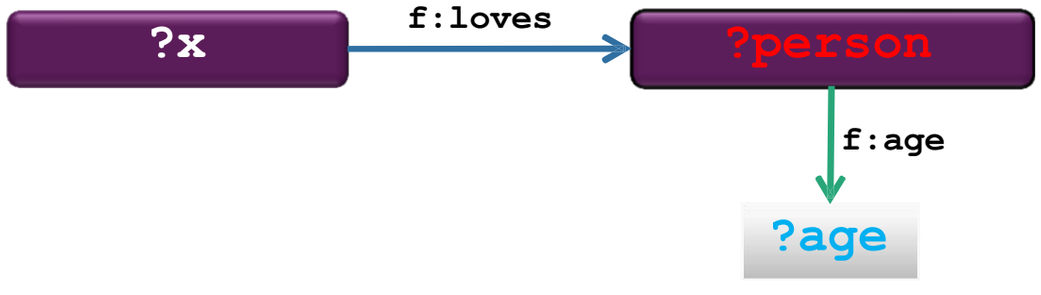
- Show me all things that are loved. Also show me their age (***f:age***)

Data



Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age
WHERE {
?x f:loves ?person .
?person f:age ?age
}
```



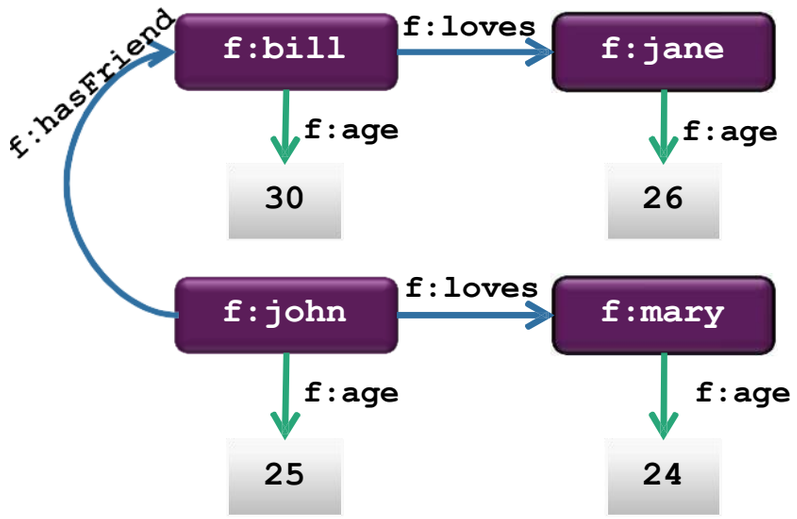
Result

person	age
f:jane	26
f:mary	24

SPARQL: FILTER (testing values)

- Show me people and their age for people older than 25.

Data

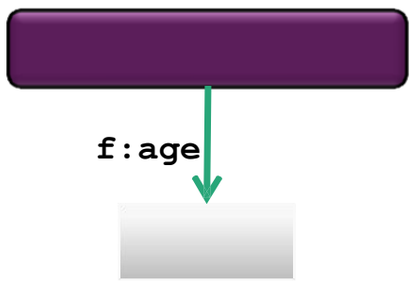


Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age
WHERE {
?person f:age ?age .   FILTER (?age >
25)
}
```

If *?age* is not a number, then it will notwork

Result



?person

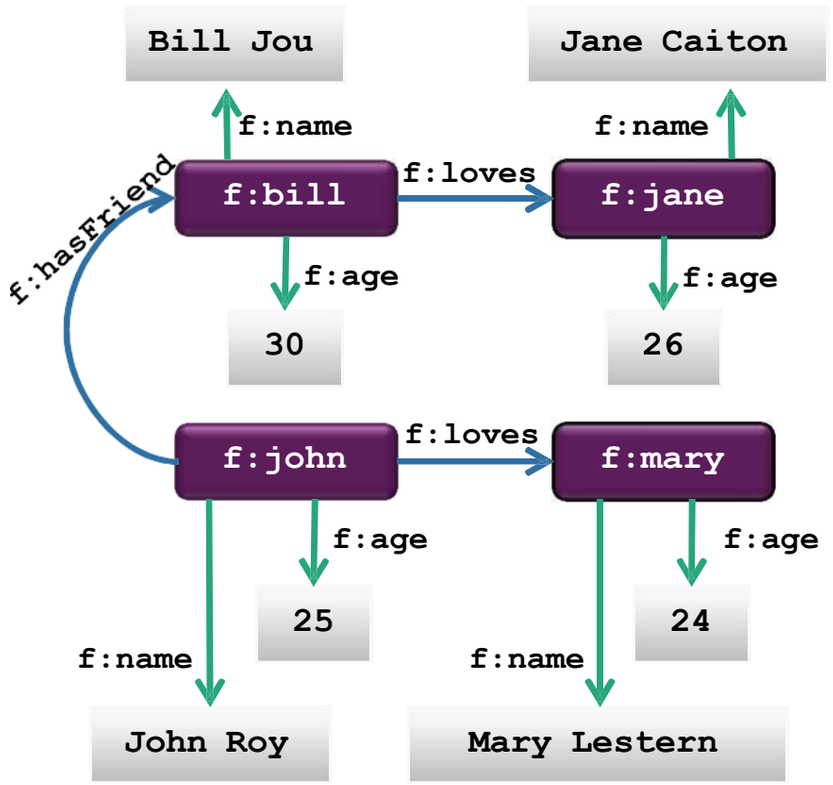
?age

person	age
f:bill	30
f:jane	26

SPARQL: FILTER (string matching)

- Show me people and their name if name has "***r***" or "***R***" in it.

Data



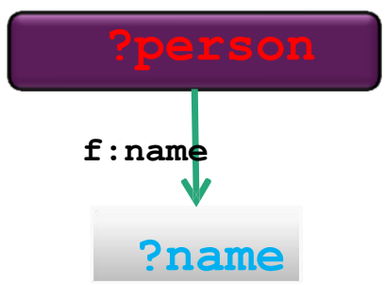
Syntax

```
FILTER regex(?x, "pattern"[, "flags"])
```

Flag " ***i*** *" means a case-insensitive pattern*

Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?name
WHERE {
?person f:name ?name .
FILTER regex(?name, "r", "i" )
}
```



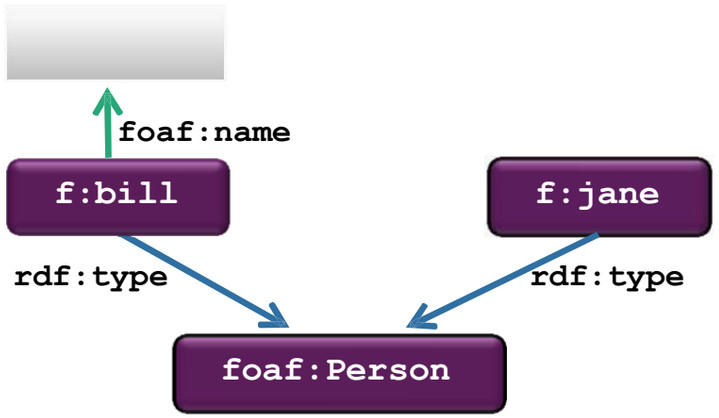
Result

person	name
f:john	John Roy
f:mary	Mary Lestern

SPARQL: FILTER (EXISTS / NOT EXISTS)

- EXISTS expression tests whether the pattern can be found in the data.
- NOT EXISTS expression tests whether the pattern does not match the dataset.

Data



Result

person
f:bill

person
f:jane

Syntax

```
FILTER EXISTS {"pattern"}
```

```
FILTER NOT EXISTS {"pattern"}
```

Query

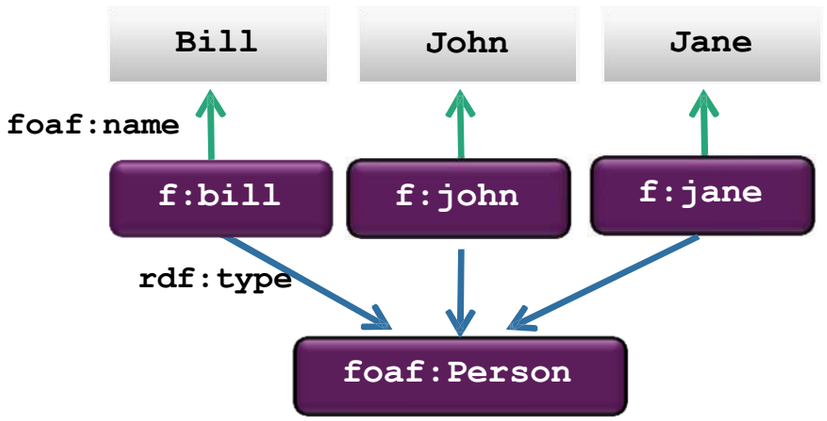
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX f: <http://example.org#>

SELECT ?person  WHERE {
?person rdf:type foaf:Person .
FILTER EXISTS {?person foaf:name ?name}
}
```

```
SELECT ?person  WHERE {
?person rdf:type foaf:Person .
FILTER NOT EXISTS {?person foaf:name ?name}
}
```

SPARQL: FILTER (MINUS)

- MINUS removes matches based on the evaluation of two patterns.

Data



Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX f: <http://example.org#>

SELECT DISTINCT ?s  WHERE { ?s ?p ?o .
MINUS { ?s foaf:name "John" . }
}
```

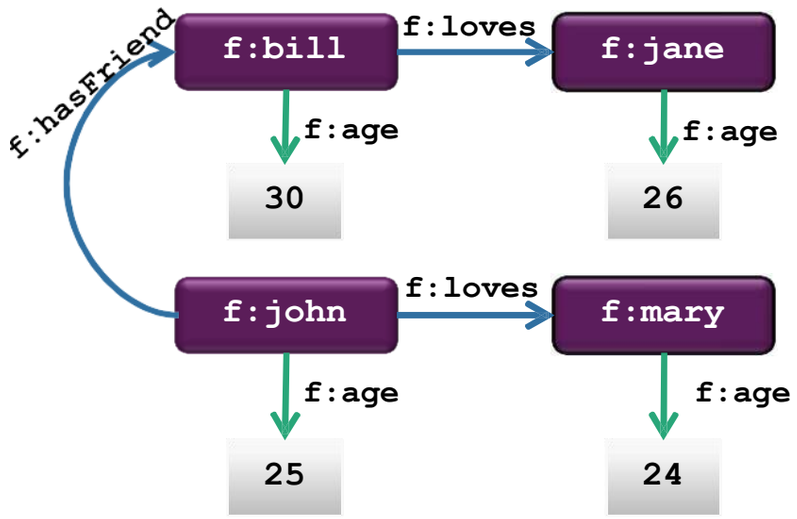
Result

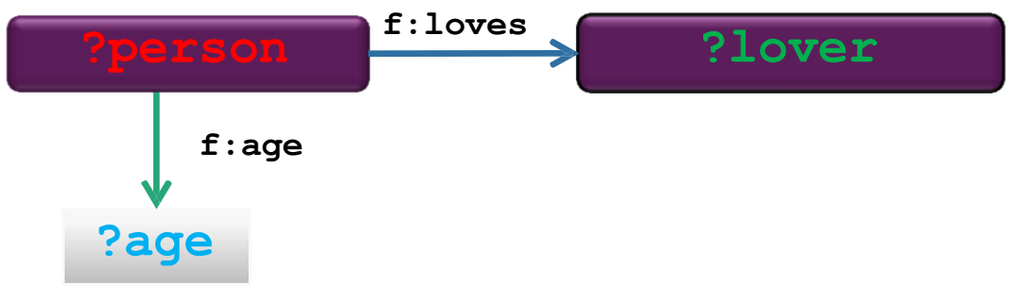
s
f:bill
f:jane

SPARQL: OPTIONAL

- Show me the person and its age (***f:age***). If you have information that person loves somebody, then show it as well.

Data





Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age ?lover
WHERE {
?person f:age ?age .
OPTIONAL {?person f:loves ?lover}
}
```

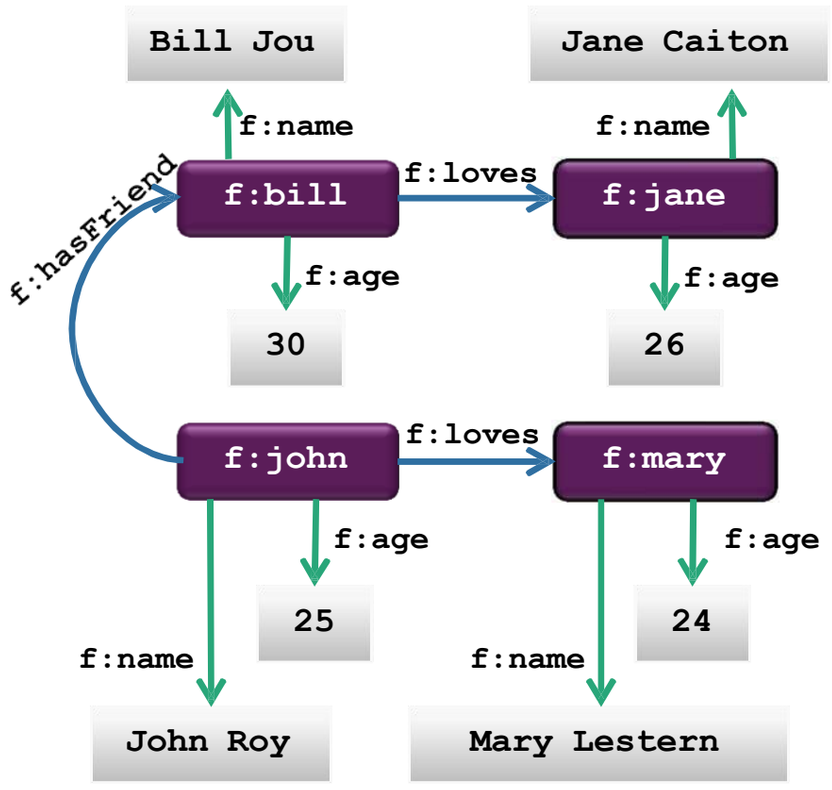
Result

person	age	lover
f:bill	30	f:jane
f:john	25	f:mary
f:mary	24	
f:jane	26	

SPARQL: OPTIONAL with FILTER

- Show me the person and its age (***f:age***). If you have information about that person loving somebody, then show that person if his/her name contains " ***r*** ".

Data



Query

```
PREFIX f: <http://example.org#>
SELECT ?person ?age ?lover
WHERE {
?person f:age ?age .
OPTIONAL {?person f:loves ?lover .
?lover f:name ?loverName .
FILTER regex(?loverName, "r", "i")}
}
```

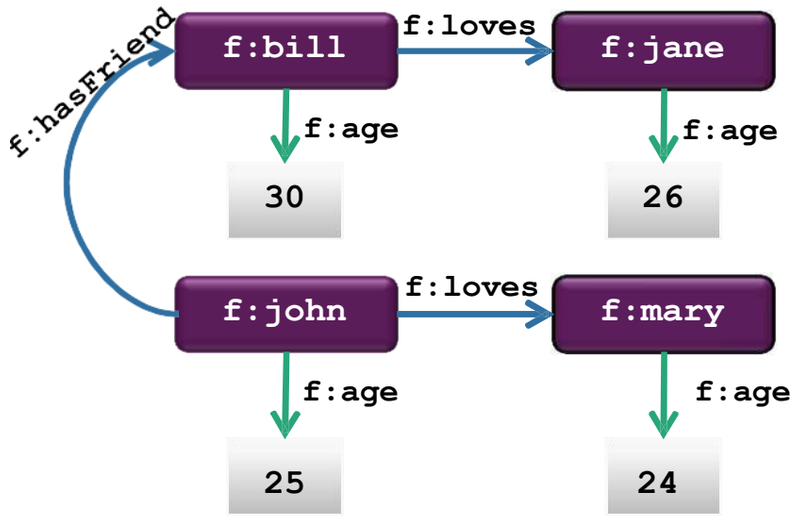
Result

person	age	lover
f:bill	30	
f:john	25	f:mary
f:mary	24	
f:jane	26	

SPARQL: Logical OR (UNION)

- Show me all people who have a friend together with all the people that are younger than 25

Data



Query

```
PREFIX f: <http://example.org#>
SELECT ?person
WHERE {
{?person f:age ?age . FILTER (?age < 25)}
UNION
{?person f:hasFriend ?friend}
}
```

```
PREFIX f: <http://example.org#>  SELECT ?person
WHERE {?person   f:age ?age . FILTER (?age < 25)}
```

+

```
PREFIX f: <http://example.org#>  SELECT ?person
WHERE {?person   f:hasFriend ?friend}
```

Result

person
f:mary
f:john

Triple Stores

- Persistent data store for the RDF model
- Core data structures are triples and graphs
- Triple stores are usually transactional
- Several standards for triple stores are proposed
  - RDF
  - RDFS
  - SPARQL

- Standardization is critical

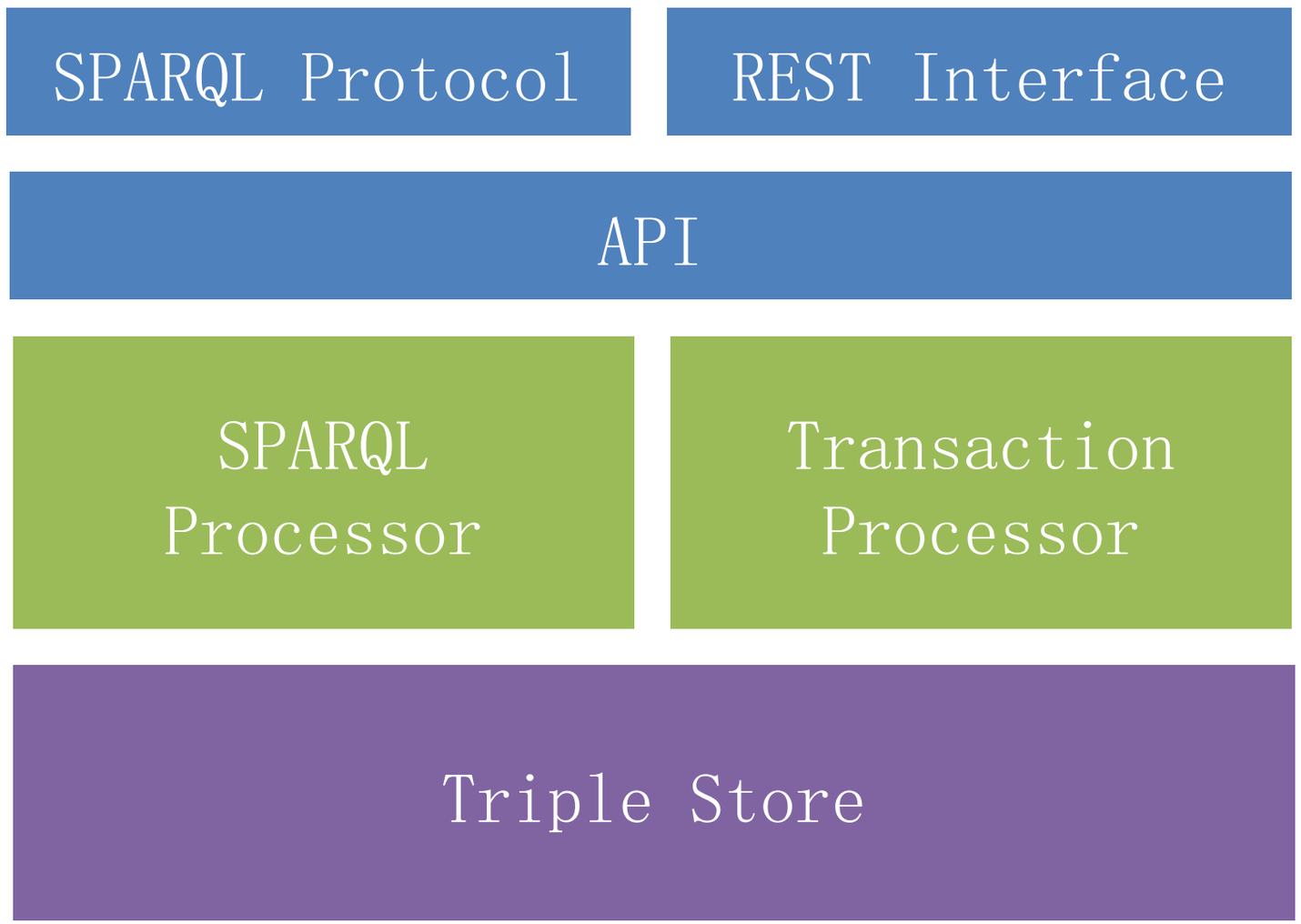
  Standardization is what made SQL popular

Triple Stores

In-Memory Stores	Transient storage of triples in memory
Native Stores	Persistent storage of triples in native stores with their own storage implementation
Non-Native Stores	Persistent storage of triples on top of third-party databases

Triple Stores

- Triple Store Operations
  - Insert/delete
  - Batch insert/delete
  - Export graph
  - SPARQL query
  - SPARQL update



Advantages

Do triple stores offer an advantage over relational databases?

- No need to create schemas
  - Can create new predicates (columns) on the fly
  - No need to link tables because you can have one to many relationships directly
- Data interoperability
- Can run queries off of this data just like in a relational database model

Ok, but...

what are the challenges brought forth?

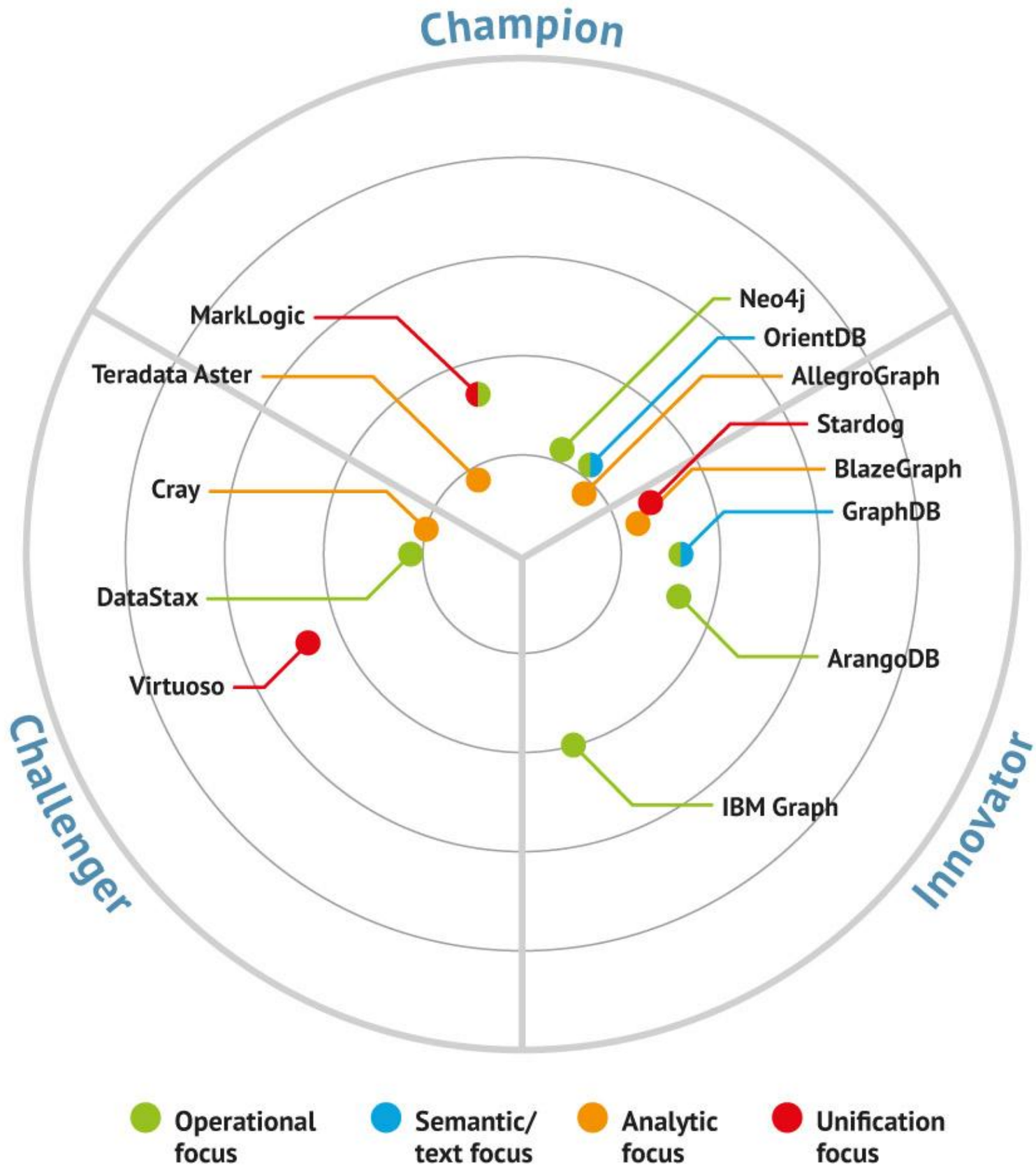
- No schema -> indexing/storage?
- complex queries can be written in SPARQL
  - Simple SQL query may translate to 10s of SPARQL joins!
  - optimization needed! (join orders, cost-based planning, etc)
- Data size + data distribution
  - Big data methods/tools needed
  - Nosql, hadoop, spark, graph databases, …

RDF stores

- RDBMS-based:
  - Αποθηκεύουν τα RDF δεδομένα σύμφωνα με κάποιο σχήμα σε μια σχεσιακή βάση δεδομένων
- Native stores:
  - Χρησιμοποιούν εξειδικευμένο τρόπο αποθήκευσης και επερώτησης των RDF δεδομένων
- Graph indexing:
  - Χρησιμοποιούν αλγορίθμους γράφων για την αποθήκευση και επερώτηση των RDF δεδομένων

Part II

COMMERCIAL AND KNOWN SOLUTIONS



Picture from Graph and RDF databases 2016, by Bloor, 2017

Native RDF Stores

RDF stores that implement their own database engine without reusing the storage and retrieval functionalities of other database management systems:

■***4Store and 5Store*** (under GPL and commercial) are RDF databases developed by Garlik Inc. 4Store is available under GNU General Public License (GPL). Client connectors are available for PHP, Ruby, Python, and Java. 5Store (unlike 4Store) is commercial software and provides similar features as 4Store, but improved efficiency and scalability.

■***AllegroGraph*** (commercial) is a commercial RDF graph database and application framework developed by Franz Inc. There are different editions of AllegroGraph and different clients: the free *RDFStore* server edition is limited to storing less than 50 million triples, a developer edition capable of storing a maximum of 600 million triples, and an enterprise edition with storage capacity only limited by the underlying server infrastructure. Clients connectors are available for Java, Python, Lisp, Clojure, Ruby, Perl, C#, and Scala.

■***Apache Jena TDB*** (open-source) is a component of the Jena Semantic Web framework and available as open-source software released under the BSD license.

■***Mulgara*** (open source, Open Software License) is the community-driven successor of Kowari and is described as a purely Java-based, scalable, and transaction-safe RDF database for the storage and retrieval of RDF-based metadata.

■***GraphDB™*** (formerly ***OWLIM***) – An Enterprise Triplestore with Meaning (GNU LGPL license and commercial). It is a family of commercial RDF storage solutions, provided by Ontotext. There are three different editions: *GraphDB™ Lite*, *GraphDB™ Standard* and *GraphDB™ Enterprise*.

■***And many more***

DBMS-backed Stores

RDF Stores that use the storage and retrieval functionality provided by another database management system:

■***Apache Jena SDB*** (open-source) is another component of the Jena Semantic Web framework and provides storage and query for RDF datasets using conventional relational databases: Microsoft SQL Server, Oracle 10g, IBM DB2, PostgreSQL, MySQL, HSQLDB, H2, and Apache Derby.

■***Oracle Spatial and Graph: RDF Semantic Graph*** (formerly ***Oracle Semantic Technologies***) is a W3C standards-based, full-featured graph store in Oracle Database for Linked Data and Social Networks applications.

■***Semantics Platform*** is a family of products for building medium and large scale semantics-based applications using the Microsoft .NET framework. It provides semantic technology for the storage, services and presentation layers of an application.

■***RDFLib*** is a pure Python package working with RDF that contains most things you need to work with, including: parsers and serializers for RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, RDFa and Microdata; a Graph interface which can be backed by any one of a number of Store implementations; store implementations for in memory storage and persistent storage on top of the Berkeley DB; a SPARQL 1.1 implementation supporting Queries and Update statements.

Hybrid Stores

RDF Stores that supports both architectural styles (native and DBMS-backed):

■***Sesame*** (open-source) is an open source framework for storage, inferencing and querying of RDF data. It is a library that is release under the Aduna BSD-style license and can be integrated in any Java application. Sesame includes RDF parsers and writers (Sesame Rio), a storage and inference layer (SAIL API) that abstracts from storage and inference details, a repository API for handling RDF data, and an HTTP Server for accessing Sesame repositories via HTTP. It operates in any Java-supporting environment and can be used by any Java application. In May 2015, Sesame officially forked into an Eclipse project called RDF4J.

■***OpenLink Virtuoso Universal Server*** is a hybrid storage solution for a range of data models, including relational data, RDF and XML, and free text documents. Through its unified storage it can be also seen as a mapping solution between RDF and other data formats, therefore it can serve as an integration point for data from different, heterogeneous sources. Virtuoso has gained significant interest since it is used to host many important Linked Data sets (e.g., DBpedia), and preconfigured snapshots with several important Linked Data sets are offered. Virtuoso is offered as an open-source version; for commercial purposes several license models exist.

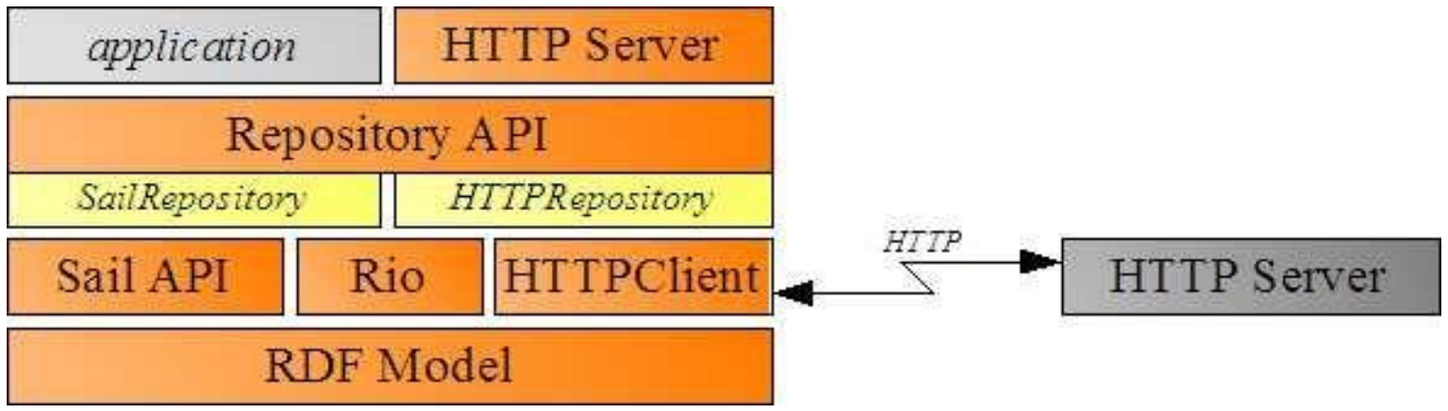
■***Blazegraph*** (open-source and commercial license) is ultra-scalable, high-performance graph database with support for the Blueprints and RDF/SPARQL APIs. Blazegraph is available in a range of versions that provide solutions to the challenge of scaling graphs. Blazegraph solutions range from millions to trillions of edges in the graph.

■***RedStore*** is a lightweight RDF triplestore written in C using the Redland library.

Sesame

- ***Sesame*** is a framework for processing RDF data
- Home: http://rdf4j.org/
- Features:
  - Parsing
    - Supports all major notations
  - Storing
    - In-memory, RDBS-backed, file-based
  - Inferencing
    - Rule-based, Ontology-based
  - Querying
    - SPARQL, SeRQL
- Java-based API + tools
- ***AliBaba*** is an RDF application library for developing complex RDF storage applications. It is a collection of modules that provide simplified RDF store abstractions to accelerate development and facilitate application maintenance.

Sesame architecture



- Rio (RDF I/O)
  - Parsers and writers for various notations
- Sail (Storage And Inference Layer)
  - Low level System API
  - Abstraction for storage and inference
- Repository API
  - Higher level API
  - Developer-oriented methods for handling RDF data
- HTTP Server
  - Accessing Sesame through HTTP

Apache Jena

- ***Apache Jena*** is a Java framework (collection of tools and Java libraries) to simplify the development of Semantic Web and Linked Data applications.
- Home: http://jena.apache.org/index.html
- Includes:
  - RDF API for processing RDF data in various notations
  - Ontology API for OWL and RDFS
  - Rule-based inference engine and Inference API
  - TDB – a native triple store
  - SPARQL query processor (called *ARQ*)
  - Servers for publishing RDF data to other applications

Mulgara

- ***Mulgara*** – is a RDF database (*successor of Kowari RDF database*): http://www.mulgara.org/
- Written entirely in Java
- Querying language – SPARQL and own TQL
- TQL language:
  - Interpreted querying and command language
  - To manage Mulgara storage (upload, etc.)
- SPARQL – query-only language:
- REST interface for TQL and SPARQL

AllegroGraph

- ***AllegroGraph*** is a high-performance persistent graph database
- Editions of AllegroGraph: the free *RDFStore* server edition is limited to storing less than 50 million triples, a developer edition capable of storing a maximum of 600 million triples, and an enterprise edition with storage capacity only limited by the underlying server infrastructure.
- Supports SPARQL, RDFS++, and Prolog reasoning
- Supports REST Protocol clients: Java Sesame, Java Jena, Python, C#, Clojure, Perl, Ruby, Scala and Lisp clients.
- Link: http://allegrograph.com, http://www.franz.com/agraph/allegrograph/
- ***AllegroGraph Web View (AGWebView)*** is a graphical user interface for exploring, querying, and managing AllegroGraph triple stores. It uses HTTP interface to provide the services through a web browser.
- ***Gruff*** - a graph-based triple-store browser for AllegroGraph.

GraphDB™

- ***GraphDB™*** (formerly *OWLIM*) – An Enterprise Triplestore with Meaning.

It is a family of commercial RDF storage solutions, provided by Ontotext. There are three different editions: *GraphDB™ Lite*, *GraphDB™ Standard* and *GraphDB™ Enterprise*.

- Link: http://www.ontotext.com/products/ontotext-graphdb-owlim/

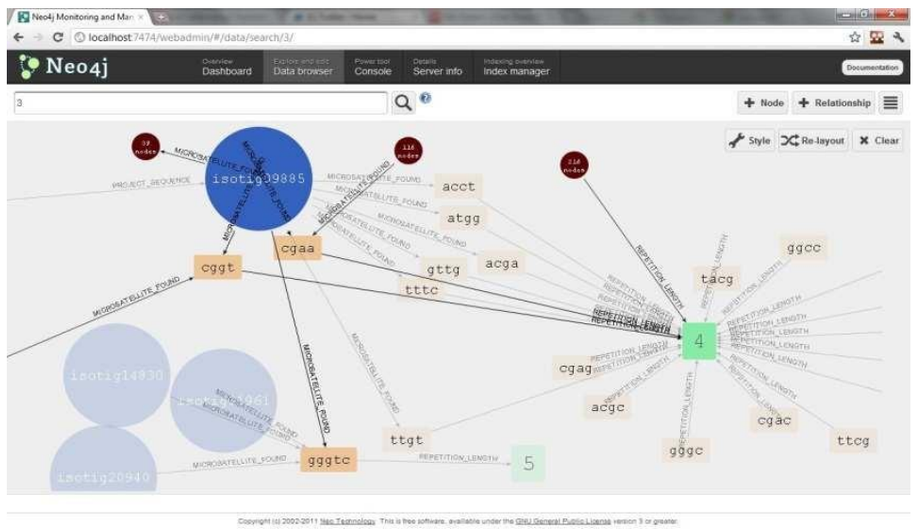
- *GraphDB™ Lite* is a free RDF triplestore that allows you to store up to 100 million triples on a desktop computer, perform SPARQL 1.1 queries in memory (not using files based indices), and supports reasoning operations for inferencing.

- *GraphDB™ Standard* allows organizations to manage tens of billions of semantic triples on one commodity server, load and query RDF statements at scale simultaneously, performs querying and reasoning operations using file based indices, and has optimized performance using “Same As” technology.

- *GraphDB™ Enterprise* has all the features of our Standard Edition with the added advantage that it has been architected to run on enterprise replication clusters with backup, recovery and failover ensuring you are always up. “Enterprise” offers industrial strength resilience and linearly scalable parallel query performance with support for load-balancing across any number of servers.

Neo4j Graph Database

- ***Neo4j*** is a highly scalable, robust (fully ACID) native graph database, used in mission-critical apps by thousands of leading startups, enterprises, and governments around the world.
  - High Performance for highly connected data
  - High Availability clustering
  - Cypher, a graph query language
  - ETL, easy import with Cypher LOAD CSV
  - Hot Backups and Advanced Monitoring



- Link: http://neo4j.com/

Part III

PROMINENT RESEARCH SOLUTIONS

RDF Data Model

Prefix: y= http://en.wikipedia.org/wiki/

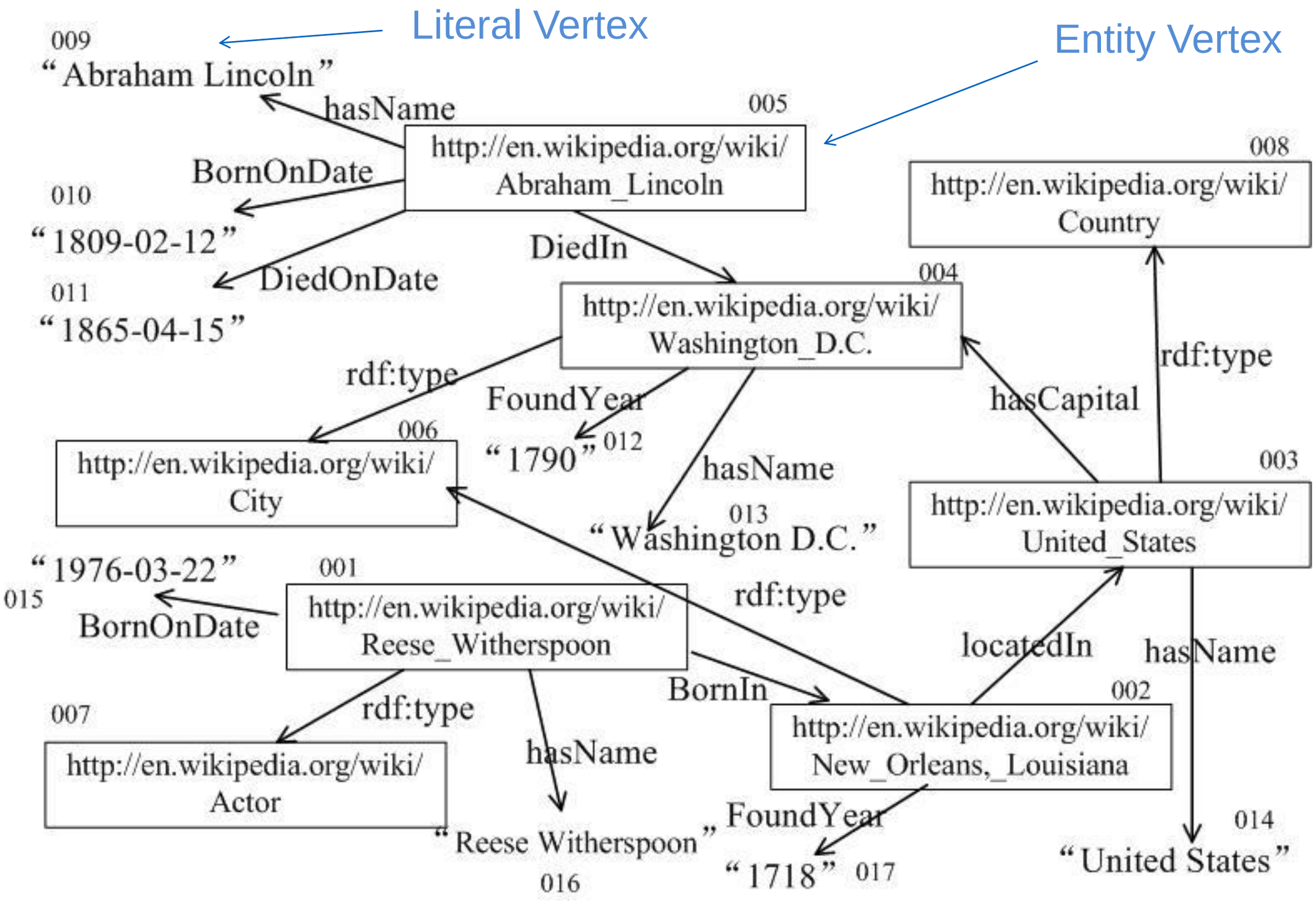
Subject	Predict	Object
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	BornOnDate	"1809-02-12"
y:Abraham_Lincoln	DiedOnDate	1865-04-15
y:Abraham_Lincoln	DiedIn	y:Washington_D.C
y:Washington_D.C	hasName	"Washington D.C."
y:Washington_D.C	FoundYear	1790
y:Washington_D.C	rdf:type	y:city
y:United_States	hasName	"United States"
y:United_States	hasCapital	y:Washington_D.C
y:United_States	rdf:type	Country
y:Reese_Witherspoon	rdf:type	y:Actor
y:Reese_Witherspoon	BornOnDate	"1976-03-22"
y:Reese_Witherspoon	BornIn	y:New_Orleans,_Louisiana
y:Reese_Witherspoon	hasName	"ReeseWitherspoon"
y:New_Orleans,_Louisiana	FoundYear	1718
y:New_Orleans,_Louisiana	rdf:type	y:city
y:New_Orleans,_Louisiana	locatedIn	y:United_States

URI

Literals

URI

RDF Graph

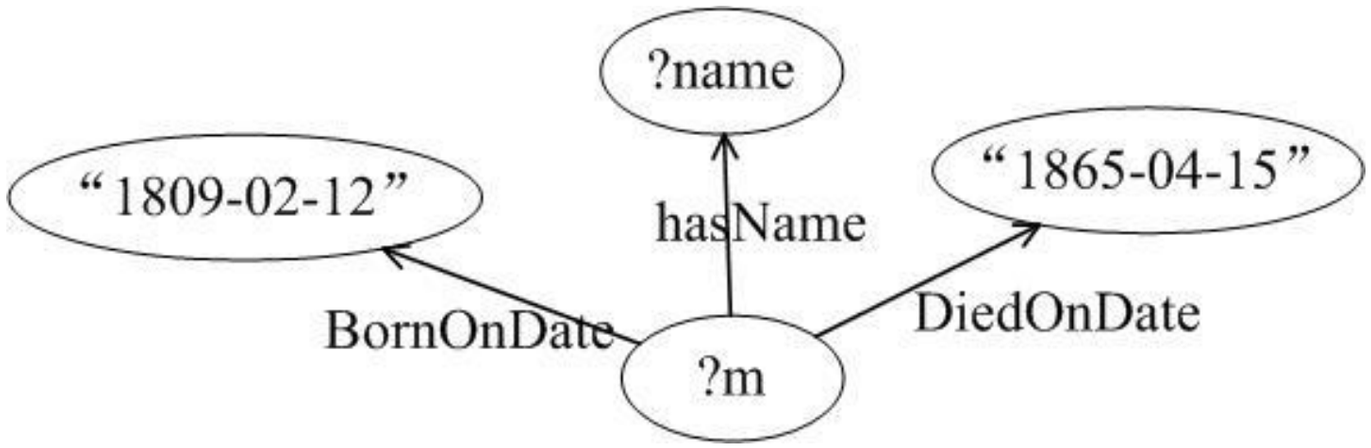

Literal Vertex
Entity Vertex
009
“Abraham Lincoln”
hasName
005
http://en.wikipedia.org/wiki/
Abraham_Lincoln
008
http://en.wikipedia.org/wiki/
Country
BornOnDate
010
“1809-02-12”
DiedIn
DiedOnDate
011
“1865-04-15”
004
http://en.wikipedia.org/wiki/
Washington_D.C.
rdf:type
rdf:type
hasCapital
FoundYear
006
012
“1790”
http://en.wikipedia.org/wiki/
City
hasName
013
“Washington D.C.”
003
http://en.wikipedia.org/wiki/
United_States
“1976-03-22”
015
001
http://en.wikipedia.org/wiki/
Reese_Witherspoon
rdf:type
locatedIn
hasName
BornOnDate
BornIn
002
rdf:type
007
http://en.wikipedia.org/wiki/
Actor
hasName
http://en.wikipedia.org/wiki/
New_Orleans,_Louisiana
“Reese Witherspoon”
016
FoundYear
014
“1718”
017
“United States”

SPARQL Queries

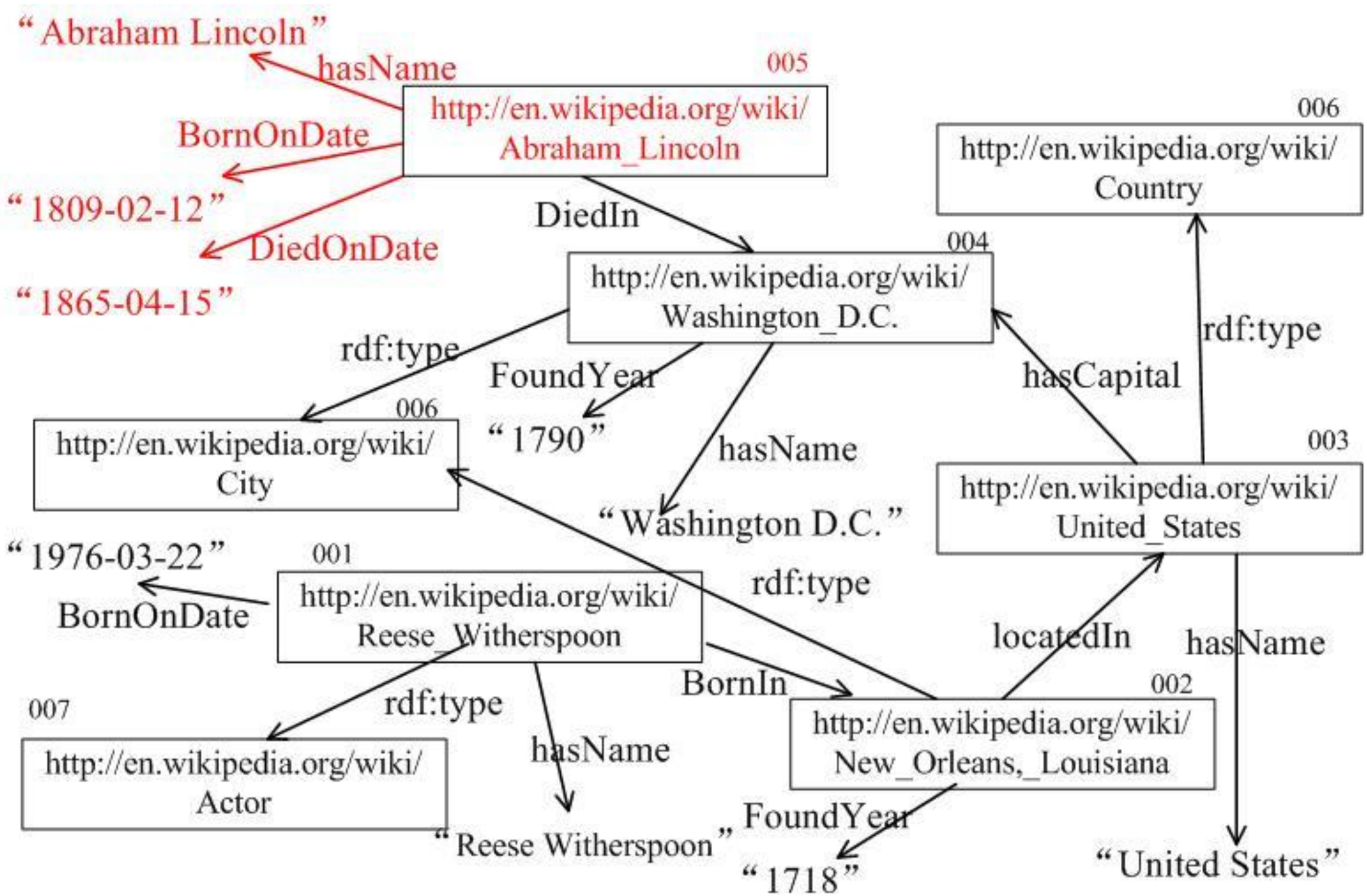
SPARQL Query:

```
Select ?name Where {
?m <hasName> ?name.
?m <BornOnDate> “1809-02-12”.
?m <DiedOnDate> “1865-04-15”. }
```

Query Graph



Subgraph Match vs. SPARQL Queries



Naïve Triple Store

SPARQL Query:

```
Select ?name Where { ?m <hasName> ?name. ?m
<BornOnDate> "1809-02-12". ?m <DiedOnDate> "1865-
04-15". }
```

Too many Self-Joins

Prefix: y= http://en.wikipedia.org/wiki/

Subject	Predict	Object
y:Abraham_Lincoln	hasName	"Abraham Lincoln"
y:Abraham_Lincoln	BornOnDate	"1809-02-12"
y:Abraham_Lincoln	DiedOnDate	1865-04-15
y:Abraham_Lincoln	DiedIn	y:Washington_D.C
y:Washington_D.C	hasName	"Washington D.C."
y:Washington_D.C	FoundYear	1790
y:Washington_D.C	rdf:type	y:city
y:United_States	hasName	"United States"
y:United_States	hasCapital	y:Washington_D.C
y:United_States	rdf:type	Country
y:Reese_Witherspoon	rdf:type	y:Actor
y:Reese_Witherspoon	BornOnDate	"1976-03-22"
y:Reese_Witherspoon	BornIn	y:New_Orleans,_Louisiana
y:Reese_Witherspoon	hasName	"ReeseWitherspoon"
y:New_Orleans,_Louisiana	FoundYear	1718
y:New_Orleans,_Louisiana	rdf:type	y:city
y:New_Orleans,_Louisiana	locatedIn	y:United_States

SQL:

```
Select T3.Subject
From  T as T1, T as T2, T
as T3
Where
T1.Predict="BornOnDate" and
T1.Object="1809-02-12" and
T2.Predict="DiedOnDate" and
T2.Object="1865-04-15" and
T3.  Predict="hasName" and
T1.Subject = T2.Subject
and  T2.  Subject=
T3.subject
```

RDBMS-based engines: Naïve solution

- Naive triple table
  - 3 columns: subject-predicate-object
  - Τα SPARQL ερωτήματα μεταφράζονται σε SQL
  - Δεν έχει καλή απόδοση για μεγάλα ερωτήματα αφού απαιτεί την εκτέλεση πολλών self-join.
  - Η πρόσβαση στον πίνακα δεν μπορεί να γίνει αποδοτικά για όλα τα πιθανά triple queries.

Subject	Predicate	Object
:Student1	:type	:Student
:Student2	:takesCourse	:Course1
:Prof1	:teacherOf	:Course1
:Prof1	:type	:Professor
…	…	…

RDBMS-based engines

- [Carroll et al. 2004]
  - **Δεν υπάρχουν πολλά διαφορετικά predicates**
  - Δημιουργία ενός πίνακα για κάθε predicate, με δυο columns, subject-object
  - Ταιριάζει με column stores γιατί οι πίνακες έχουν μόνο 2 columns (SW-store [Abadi et al. 2009]) και μπορεί να γίνει vertical partitioning
  - Δεν προσφέρουν αποδοτική εύρεση των δεδομένων για κάθε triple query:
    - Πως βρίσκουμε δεδομένα για query με μεταβλητή στο predicate?
  - Δημιουργία πολλών πινάκων.

Vertically Partitioned Solution

Fast Merge Join

Prefix: y= http://en.wikipedia.org/wiki/

hasName

Subject	Object
y:Abraham_Lincoln	"Abraham Lincoln"
y:Washington_D.C	"Washington D.C."
y:Reese_Witherspoon	"ReeseWitherspoon"
y:United_States	"United States"

FoundYear

Subject	Object
y:Washington_D.C	1790
y:New_Orleans,_Louisiana	1718

rdf:type

Subject	Object
y:Washington_D.C	y:city
y:United_States	Country
y:Reese_Witherspoon	y:Actor
y:New_Orleans,_Louisiana	y:city

BornOnDate

Subject	Object
y:Abraham_Lincoln	"1809-02-12"
y:Reese_Witherspoon	"1976-03-22"

DiedOnDate

Subject	Object
y:Abraham_Lincoln	"1865-04-15"

BornIn

Subject	Object
y:Reese_Witherspoon	y:New_Orleans,_Louisiana

DiedIn

Subject	Object
y:Abraham_Lincoln	y:Washington_D.C

LocatedIn

Subject	Object
y:New_Orleans,_Louisiana	y:United_States

Property Table

SPARQL Query:

```
Select ?name Where { ?m <hasName> ?name. ?m
<BornOnDate> "1809-02-12". ?m <DiedOnDate> "1865-
04-15". }
```

Reducing # of join steps

Prefix: y= http://en.wikipedia.org/wiki/

People

Subject	hasName	BornOnDate	DiedOnDate	DiedIn	BornIn	rdf:type
y:Abraham_Lincoln	"Abraham Lincoln"	1809-02-12	1865-04-15	y:Washington_D.C		
y:Reese_Witherspoon	"ReeseWitherspoon"	1976-03-22		y:Washington_D.C	y:New_Orleans,_Louisiana	y:Actor

City

Subject	FoundYear	rdf:type	locatedIn	hasName
y:New_Orleans,_Louisiana	1718	y:city	y:United_States	
y:Washington_D.C	1790	y:city	y:United_States	"Washington D.C."

Country

Subject	hasName	hasCapital	rdf:type
y:United_States	"United States"	y:Washington_D.C	Country

SQL:

```
Select People.hasName from People where
 People.BornOnDate = "1809-02-12" and People.DiedOnDate = "1865-
04-15".
```

RDBMS-based engines

- Type-oriented
  - Ένας πίνακας για κάθε RDF type
    - π.χ. Διαφορετικοί πίνακες για students και universities.
  - Ο κάθε πίνακας έχει διαφορετικά columns ανάλογα με το τι είδους δεδομένα υπάρχουν
  - Πως βρίσκουμε το schema?
    - Με graph coloring πάνω στο γραφο του dataset [Bornea et al. 2013]
    - Με lattice structures [Wang et al. 2010]
    - Με βάση τα characteristic sets (predicates που είναι κοινά για πολλά subjects) [Pham et al. 2015]
  - Online adaptivity στο schema?

Native RDF indexing

- Hexastore [Weiss et al. 2008]
  - Κρατάμε όλα τα sorted permutations των triples
  - SPO, SOP, PSO, POS, OSP, OPS
  - Όλα τα query patterns μπορούν να απαντηθούν με ένα Index scan
  - Όλα τα αρχικά join μπορούν να γίνουν με αποδοτικά merge-joins
  - Πιο ακριβά hash ή sort-merge joins χρειάζονται μόνο όταν κάνουμε join non-ordered intermediate results

Five-fold Increase in Index Space

- Sharing The Same Terminal Lists
  - SPO-PSO, SOP-OSP, POS-OPS

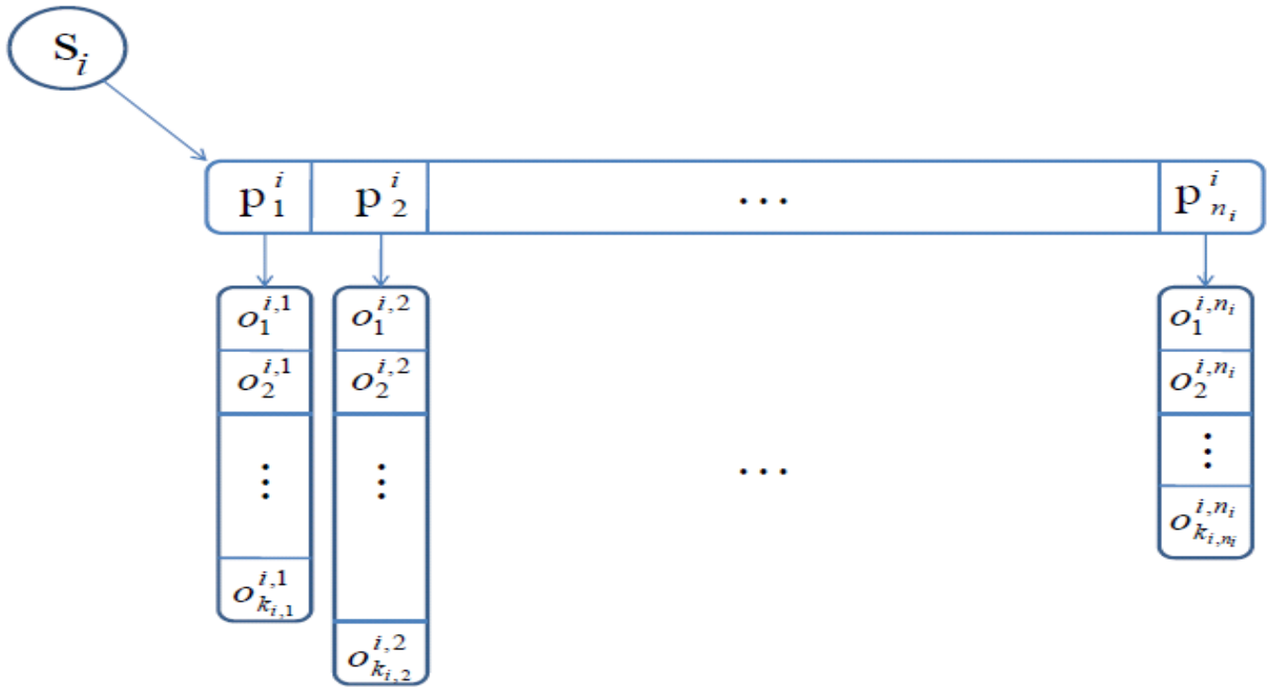


Figure 2: spo **indexing in a Hexastore**

- The key of each of the three resources in a triple appears in two headers and two vectors, but only in one list

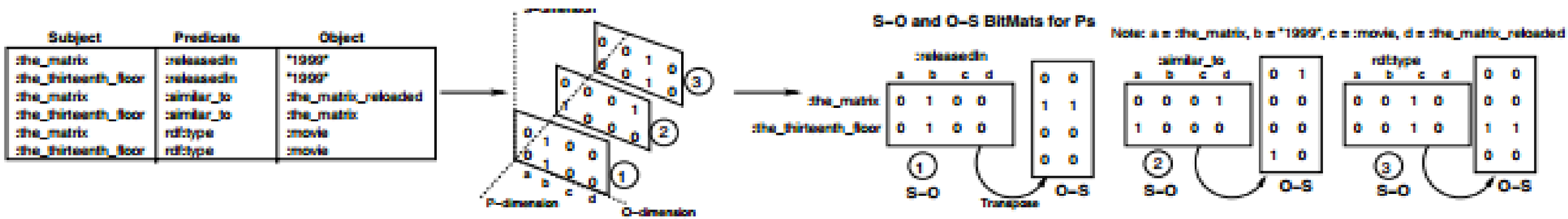
Native RDF indexing

Query pattern			Index
Subject	Predicate	Object	
_	_	_	όλα
?	_	_	pos, ops
_	?	_	osp, sop
_	_	?	spo, pso
?	?	_	osp, ops
_	?	?	spo, sop
?	_	?	pos, pso
?	?	?	όλα

- RDF-3X [Neumann et al. 2008]
  - 6 indexes, aggregated indexes για στατιστικά και εύρεση του βέλτιστου πλάνου εκτέλεσης
  - Aggressive compression

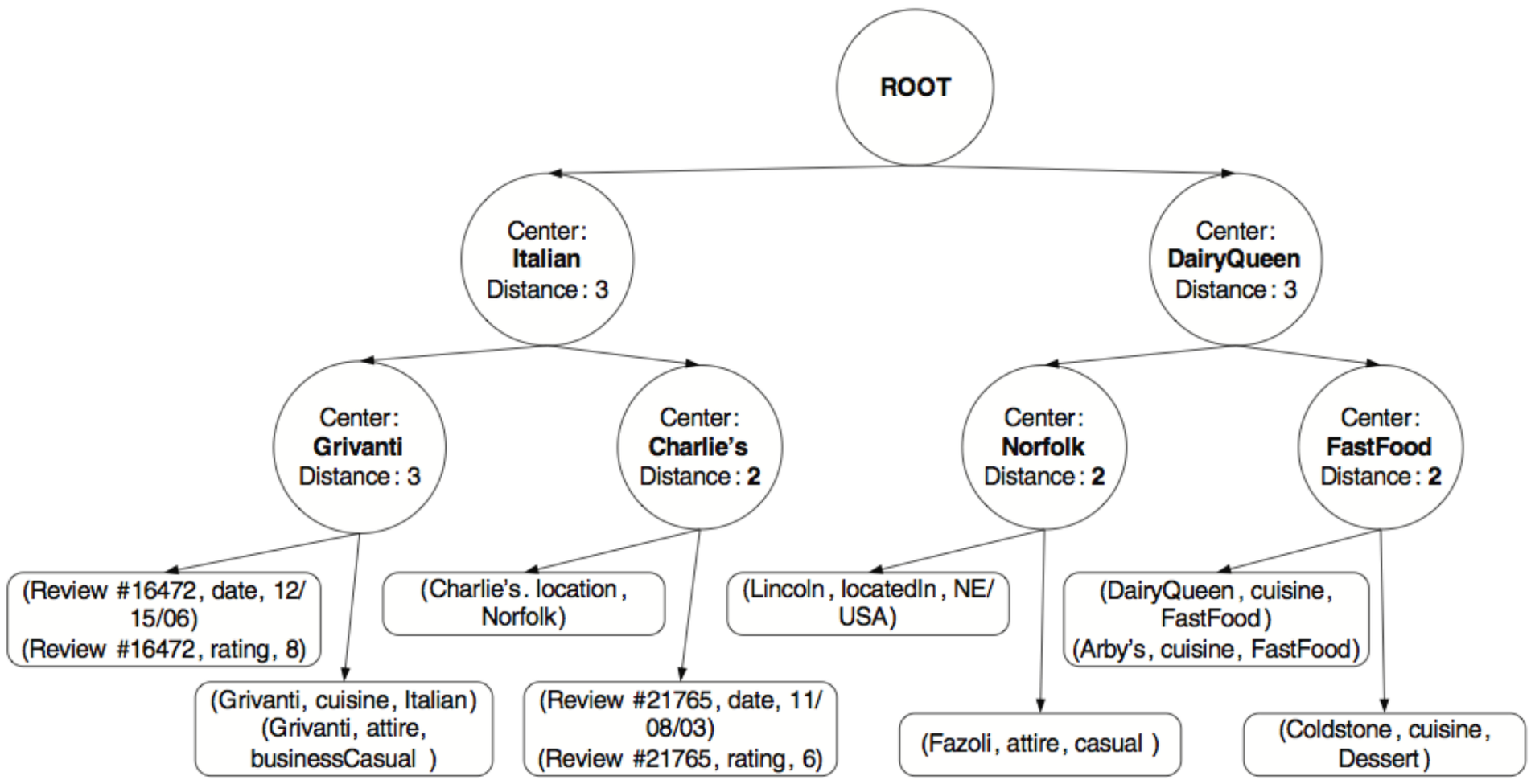
Native RDF indexing

- BitMat [Atre et al. 2008]
  - Βλέπει τα RDF σαν ένα 3D cube
  - Αποθηκεύει compressed bit matrixes
  - Αντί για join εκτελεί ένα semi-join plan που ουσιαστικά είναι σαν μια σειρά από πολλαπλασιασμούς πινάκων
  - Το semi-join processing μπορεί να απαντήσει ακριβώς για ερωτήματα με δενδρική δομή
  - Για ερωτήματα με κύκλους δεν εξασφαλίζει πλήρες reduction των αποτελεσμάτων



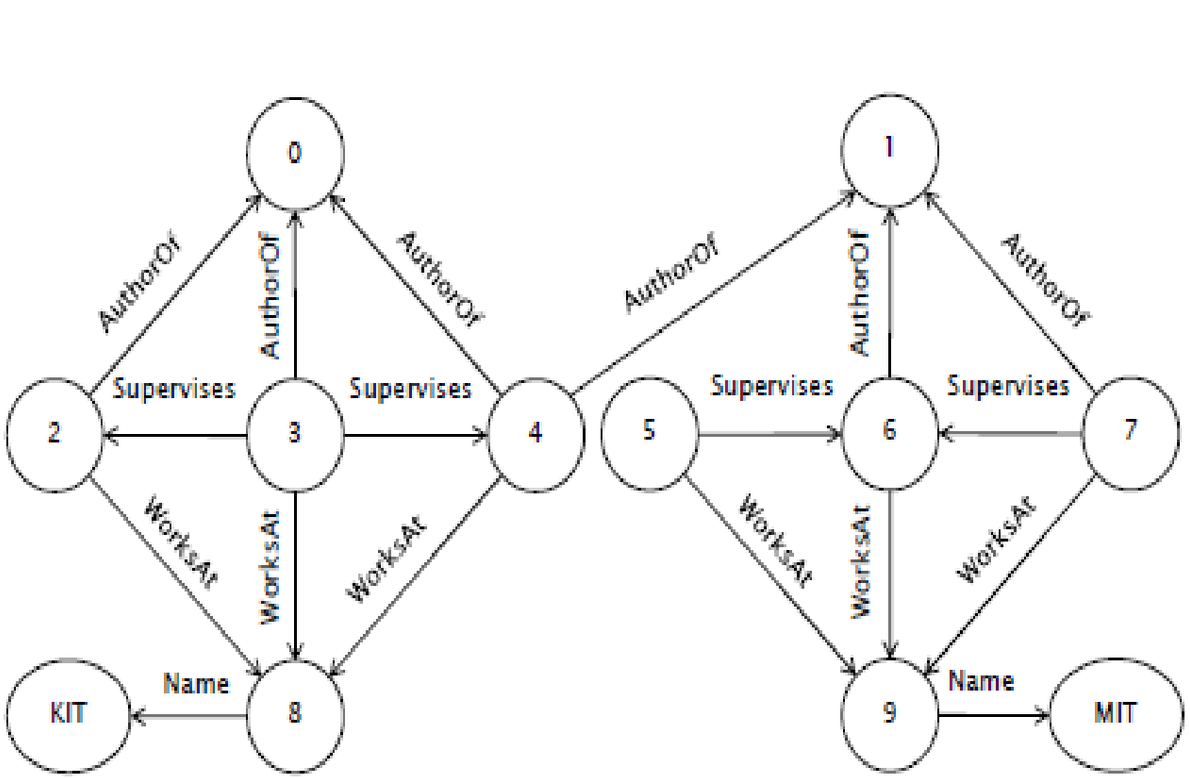
Graph indexing

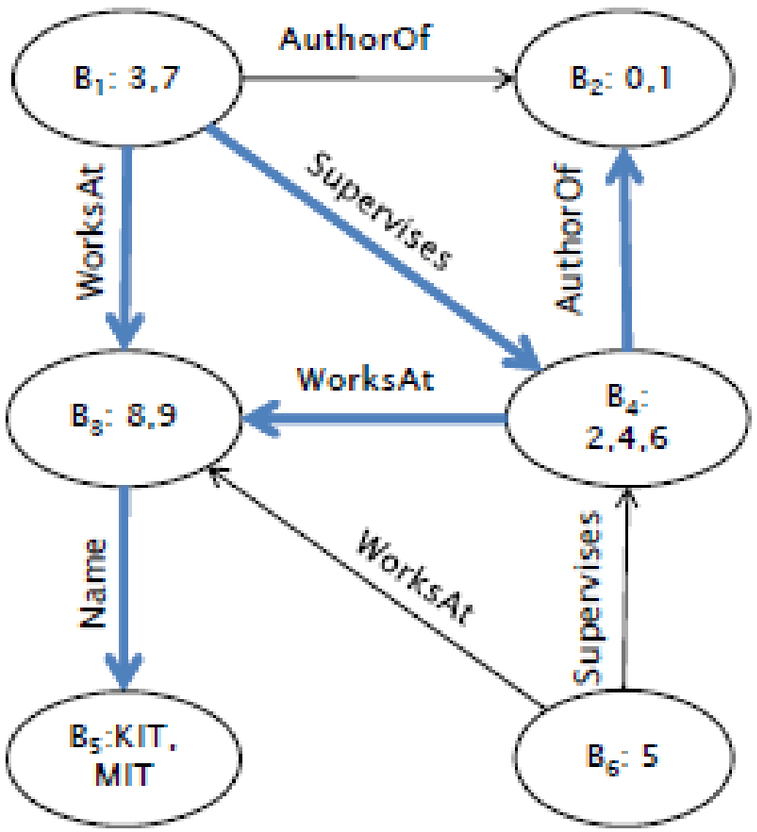
- GRIN index [Udrea et al. 2007]
  - Εύρεση κέντρων του RDF γράφου
  - Ανάθεση των κέντρων σε δενδρική διάταξη
  - Data pruning με βάση τις εκτιμώμενες αποστάσεις



Graph indexing

- Summary based index [Tran and Ladwig 2010]
  - Γενίκευση του RDF γράφου με βάση το forward-backward bisimulation





RDF graph

Summary graph

Graph indexing

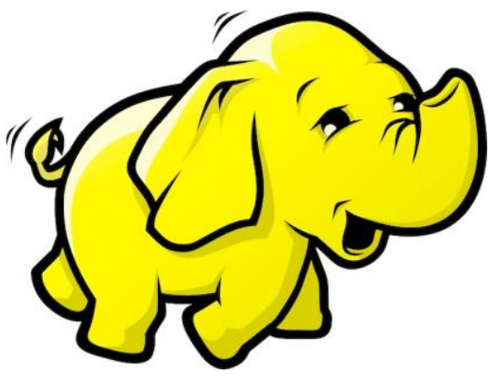
- Summary based index:
- Το SPARQL ερώτημα γίνεται πρώτα match πάνω στον summary graph
  - Αυτό διευκολύνει την μείωση των δεδομένων που χρειάζεται να εξετάσουμε
  - Reduction όχι μόνο με βάση το triple query αλλά και με βάση το neighborhood structure των κόμβων
  - Tradeoff μεταξύ summary size-summary pruning capabilities

Graph indexing

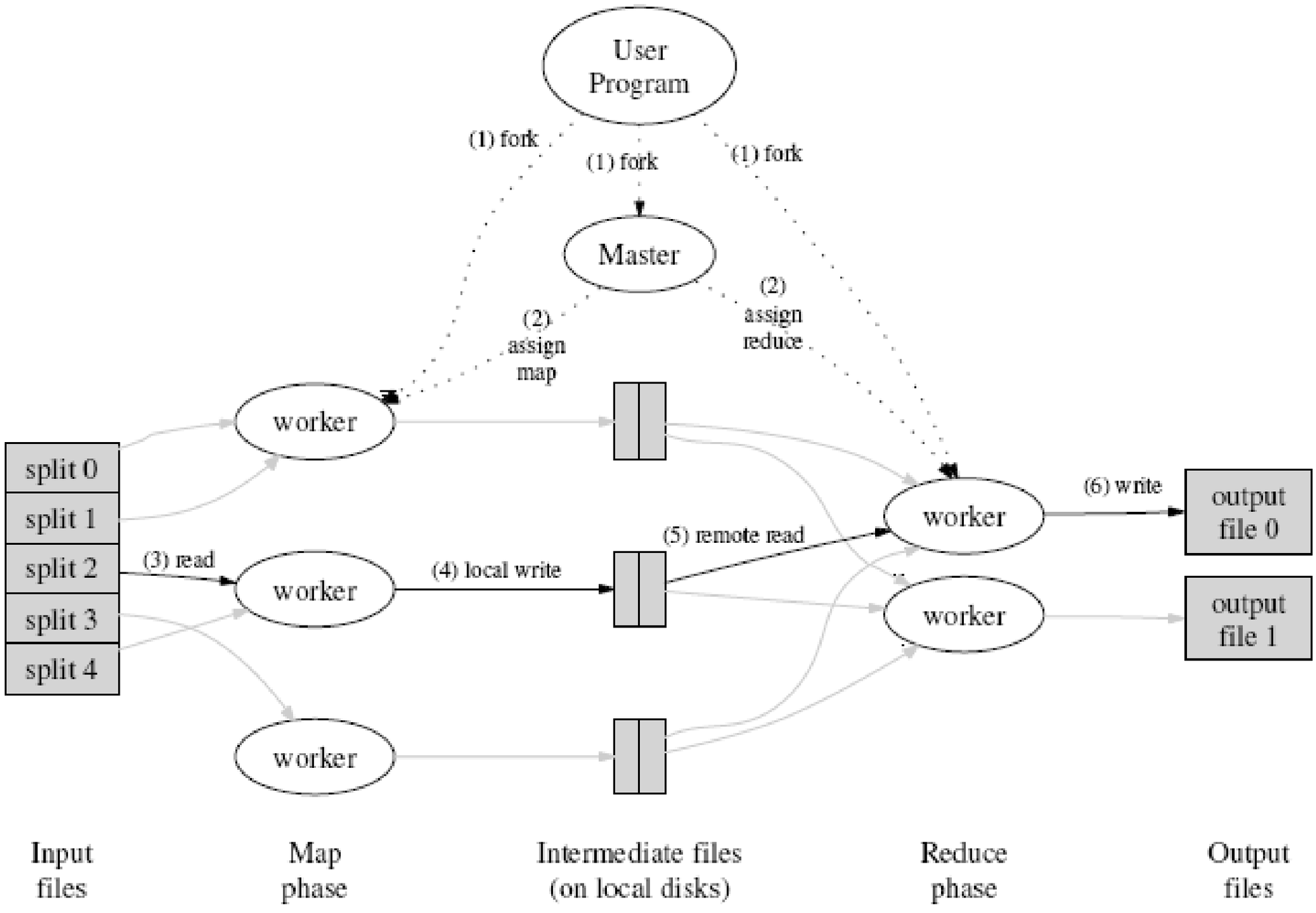
- Χρήση αλγορίθμων subgraph isomorphism [Kim et al. 2015]
  - $\text{Turbo}_{\text{ISO}}$, GADDI, VF2
  - Μετασχηματισμός του SPARQL query προβλήματος σε subgraph isomorphism
  - $\text{Turbo}_{\text{ISO}}$ -> $\text{Turbo}_{\text{HOM}}$++
  - Type-aware transformation για βελτίωση της απόδοσης των ερωτημάτων
  - Numa-aware παραλληλοποίηση

Distributed systems

- Centralized systems:
  - Αποδοτική εκτέλεση μικρών-selective ερωτημάτων
  - Μεγάλη εξάρτηση από το μέγεθος της RAM
  - Περιορισμένη παραλληλοποίηση για δύσκολα, non-selective ερωτήματα
- Distributed systems:
  - Διαχειρίζονται RDF dataset μεγάλου όγκου
  - Εκμεταλλεύονται τις νέες cloud υποδομές
  - Χρησιμοποιούν τις νέες τεχνολογίες των No-SQL βάσεων δεδομένων και του MapReduce



MapReduce

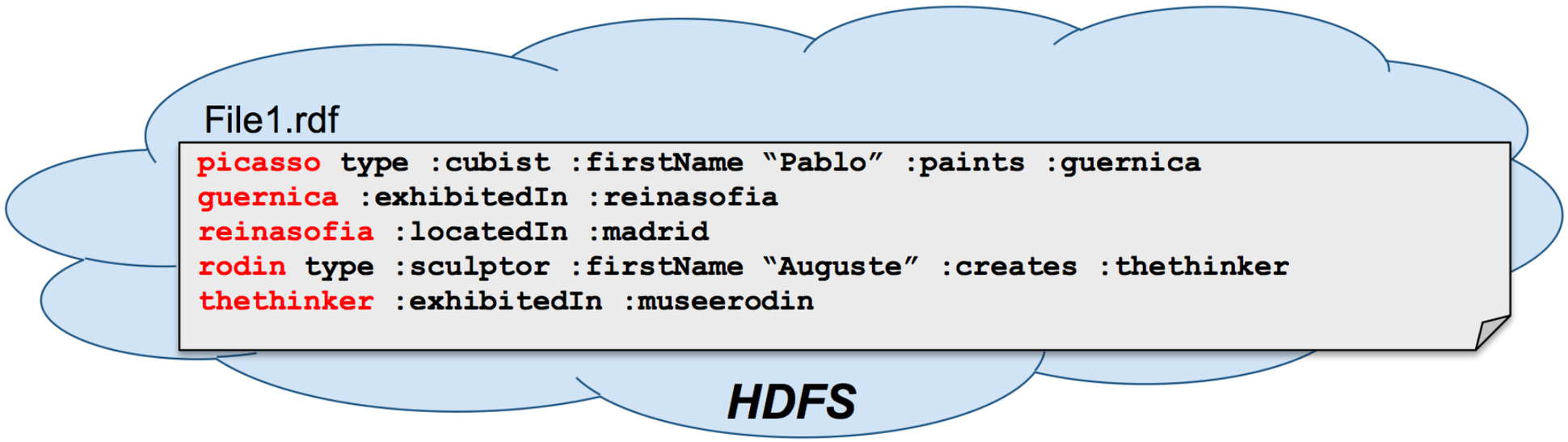

User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

MapReduce-based engines

- Αποθήκευση RDF δεδομένων σε αρχεία HDFS
- Υλοποίηση join με χρήση MapReduce
- Αντιπροσωπευτικά συστήματα
  - SHARD [Rohloff 2010]
  - HadoopRDF [Husain 2011]
  - PigSPARQL [Schätzle 2012]

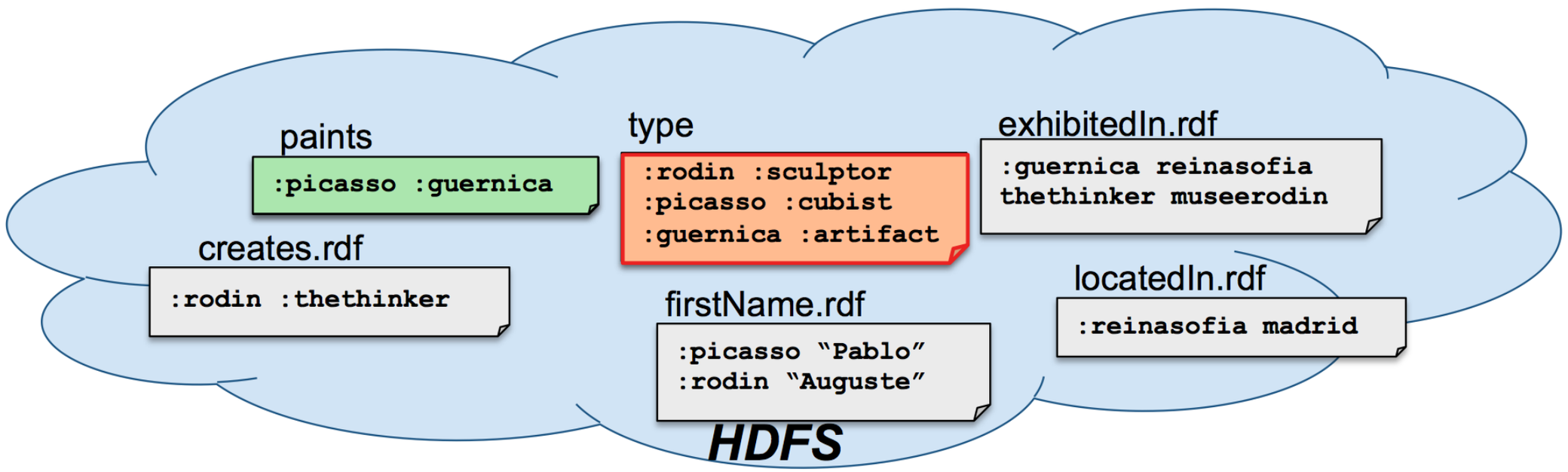
SHARD

- Αρχεία HDFS:
  - Μια γραμμή περιέχει όλες τις τριάδες που έχουν ένα συγκεκριμένο subject
- Query processing
  - Left deep join plans
  - Ένα MapReduce job για κάθε BGP του ερωτήματος
  - Μικρές δυνατότητες φιλτραρίσματος των RDF δεδομένων



HadoopRDF

- Αρχεία HDFS:
  - RDF τριάδες ομαδοποιημένες ανά predicate
  - Εσωτερική ομαδοποίηση των αρχείων με βάση το type του κάθε object
- Query processing
  - Επιλογή αρχείων που ταιριάζουν σε κάθε BGP
  - Πολλαπλά join ανά MapReduce job
  - Heuristic join planner



Pig SPARQL

- Αρχεία HDFS:
  - Ένα ενιαίο αρχείο που περιέχει όλα τα RDF δεδομένα
  - Χωρίς δυνατότητες ευρετηρίασης
- Query processing
  - Μετάφραση SPARQL σε Pig scripts
  - Εκτέλεση των PIG scripts με MapReduce jobs

NoSQL indexing

- Χρήση key-value store για δημιουργία ευρετηρίων
  - Αριθμός ευρετηρίων 1-6
  - Ανάκτηση δεδομένων για BGP:
    - Με lookups ή range scans
- Επεξεργασία ερωτημάτων
  - Με τοπική επεξεργασία
  - Με χρήση MapReduce
- Αντιπροσωπευτικά συστήματα
  - Stratustore [Stein 2010]
  - Rya [Punnoose 2012]
  - H2RDF [Papailiou 2012]
  - H2RDF+ [Papailiou 2013]
  - MAPSIN [Schätzle 2012]

NoSQL indexing

SPO

Key	(attribute, value)
:picasso,:firstName,"Pablo"	-
:picasso,:p…	
:picasso,ty…	
:guernica,:…	
…	

POS

Key	(attribute, value)
:exhibitedIn,:reinasofia,:guernica	-
:firstName,"Pablo",:picasso,	-
:paints,:gu…	
type,:cubis…	
…	

OSP

Key	(attribute, value)
:cubist,:picasso,type,	-
:guernica,:picasso,:paints	-
"Pablo",:picasso,:firstName	-
:reinasofia,:guernica,:exhibitedIn	-
…	-

NoSQL indexing

- Stratustore (Amazon SimpleDB)
  - S|P|O — 1 hash index
- Rya (Apache Accumulo)
  - SPO|–, POS|–, OSP|– — 3 sorted indexes
- H2RDF (Apache HBase)
  - SPO|–, POS|–, OSP|– — 3 sorted indexes
- H2RDF+ (Apache HBase)
  - SPO|–, SOP|–, POS|–, PSO|–, OPS|–, OSP|– — 6 sorted indexes
- MAPSIN (Apache HBase)
  - SPO|, OPS|– — 2 sorted indexes

Rya

- Αποθήκευση δεδομένων
  - Apache Accumulo
  - 3 sorted indexes
  - Ανάκτηση δεδομένων με lookup για όλα τα BGP patterns
- Επεξεργασία ερωτημάτων
  - Index nested loops
  - Αναζήτηση του accumulo index για κάθε κλειδί του join
  - Αποδοτικό για μικρού μεγέθους join
  - Απαιτεί πολλά index lookups για non selective ερωτήματα

H2RDF

- Αποθήκευση δεδομένων
  - Apache HBase
  - 3 sorted indexes
  - Ανάκτηση δεδομένων με lookup για όλα τα BGP patterns
- Επεξεργασία ερωτημάτων
  - Partial Input Hash joins
  - Διαλέγει τον κατάλληλο join αλγόριθμο ανάλογα με το μέγεθος δεδομένων των BGP
    - Αν υπάρχει μικρό pattern μόνο αυτό χρησιμοποιείται ως είσοδος
  - Κατανεμημένη (MapReduce) ή κεντρική εκτέλεση των join

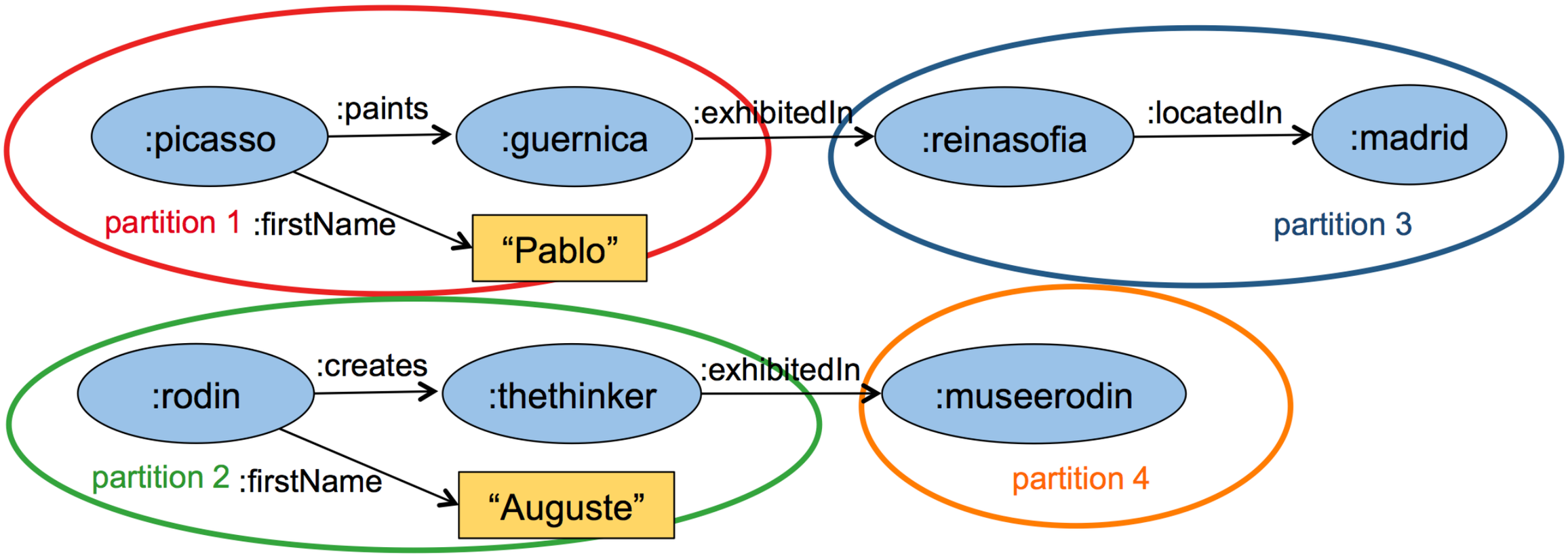
H2RDF+

- Αποθήκευση δεδομένων
  - Apache HBase
  - 6 sorted indexes και aggregated satatistics
  - Συμπιέση των ευρετηρίων
  - Ανάκτηση ταξινομημένων δεδομένων για όλα τα BGP patterns
- Επεξεργασία ερωτημάτων
  - Αξιοποιεί τα 6 index για την εκτέλεση Merge join
  - Multi-way Merge και Sort-Merge joins
  - Κατανεμημένη (MapReduce) ή κεντρική εκτέλεση των join
  - Μοντέλο κόστους για τα joins
  - Ελαστική επιλογή των resources

MAPSIN

- Αποθήκευση δεδομένων
  - Apache HBase
  - 2 sorted indexes
  - Ανάκτηση δεδομένων για τα BGP patterns με γνωστό predicate
- Επεξεργασία ερωτημάτων
  - MapReduce map phase join algorithm
  - Lookup operations για κάθε κλειδί του join

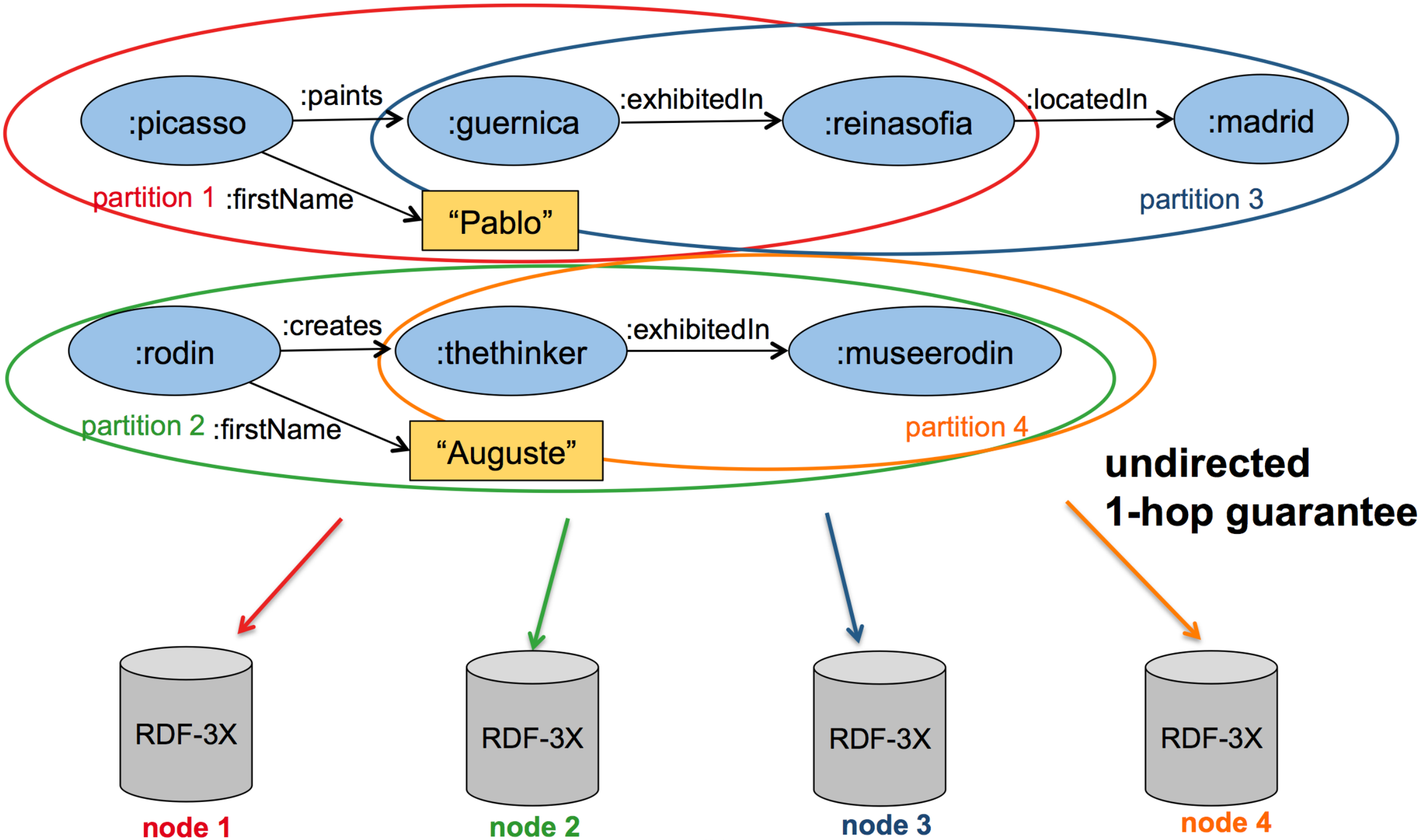
Graph partitioning



Graph partitioning

- Graph partitioning [Huang et al. 11]
  - Graph partitioning για διαμειρασμό των RDF δεδομένων
  - Κάθε κόμβος χρησιμοποιεί ένα RDF-3X για τα δεδομένα του δικού του graph partition
  - n-hop replication scheme
    - Εκτός από τα δεδομένα του partition του ο κάθε κόμβος κάνει replicate και δεδομένα που είναι έως n βήματα μακριά
    - Παράλληλη εκτέλεση για ερωτήματα με διάμετρο μικρότερη του n
    - Τα μεγαλύτερα ερωτήματα χωρίζονται σε μικρότερα, διαμέτρου n
    - Τα αποτελέσματα των υποερωτημάτων συνδυάζονται με χρήση MapReduce

1-hop guarantee

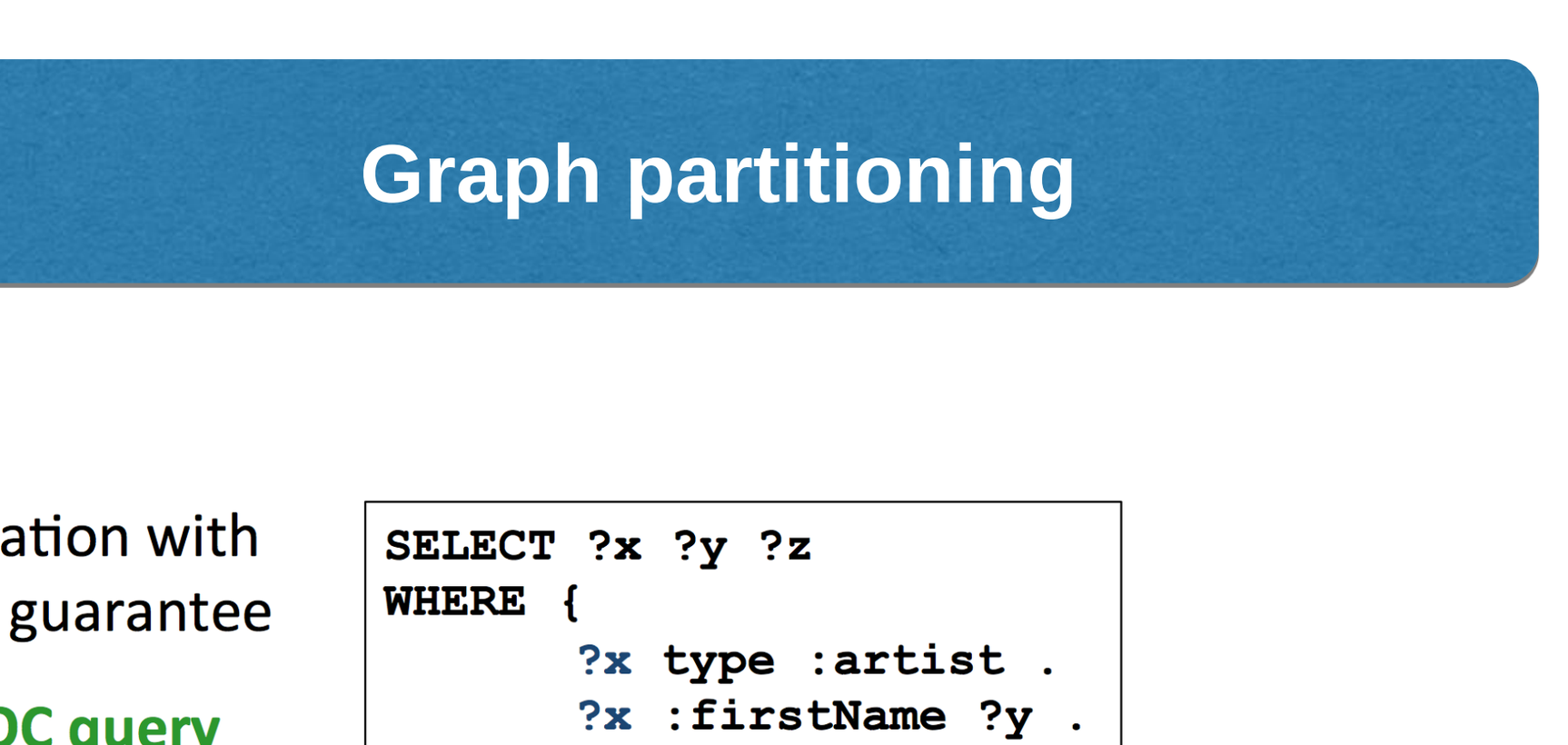


Graph partitioning

replication with
1-hop guarantee

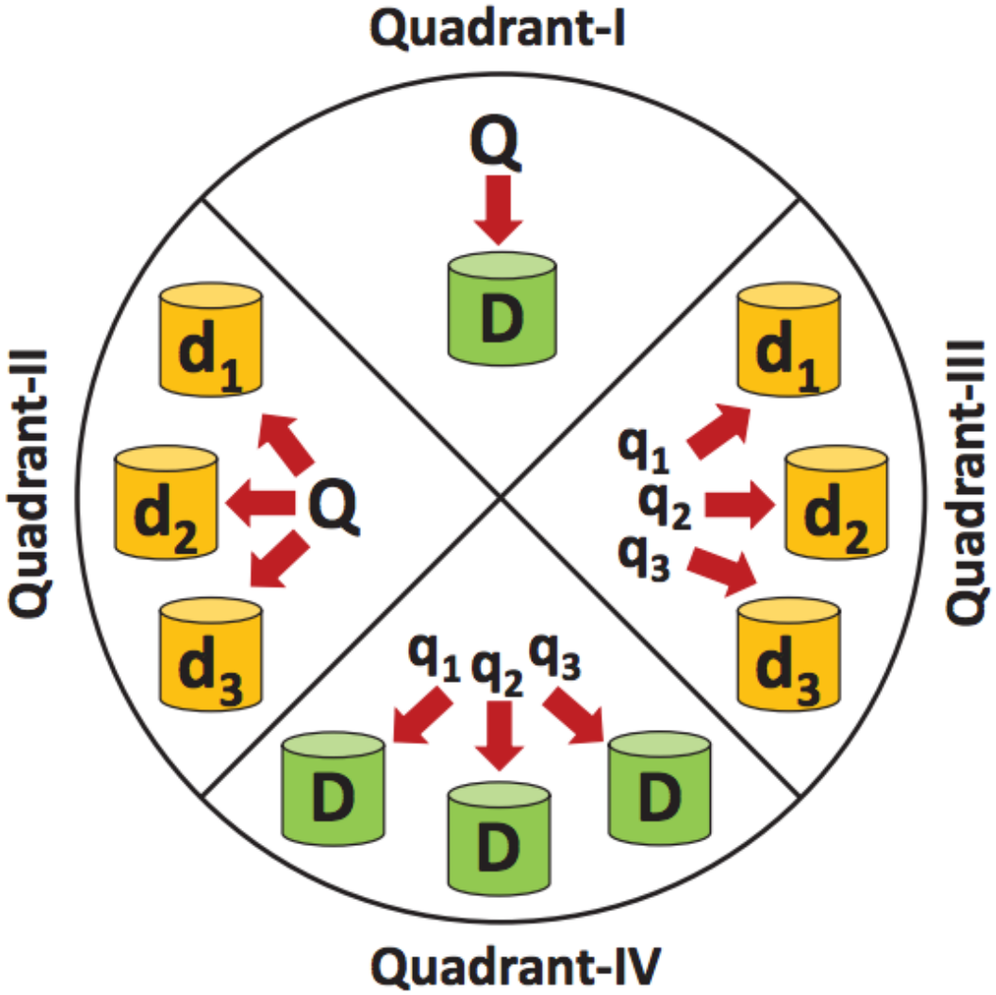
PWOC query

```
SELECT ?x ?y ?z
WHERE {
        ?x type :artist .
        ?x :firstName ?y .
        ?x :creates ?z .}
```



DREAM

- DREAM [Hammoud 2015]
  - Διαχωρισμός συστημάτων ανάλογα με την κατανομή των δεδομένων και της επεξεργασίας
  - Υλοποίηση συστήματος που ανήκει στο Quadrant-IV



Distributed Main Memory

- Memory cloud: TrinityRDF [Zeng et al. 2013]
  - Αποθηκεύει τα RDF δεδομένα στην κύρια μνήμη όλων των κόμβων του cluster
  - Κάθε κόμβος κρατάει ένα memory hash-map των RDF δεδομένων που του αντιστοιχούν
  - Query execution
    - Graph exploration με μηνύματα μεταξύ των κόμβων
    - Ουσιαστικά κάνει ένα semi-join processing για το query
    - Δεν κάνει πλήρες reduction για ερωτήματα με κύκλους
    - Στο τέλος τα reduced αποτελέσματα μαζεύονται σε ένα κεντρικό server που κάνει το τελικό processing

Distributed Main Memory

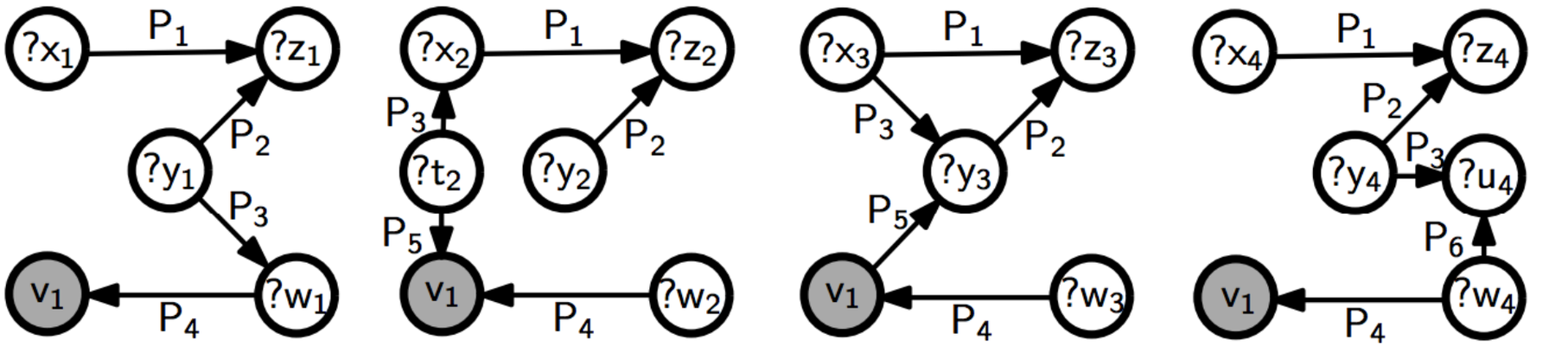
- TriAD [Gurajada 2014]
  - Hash partitioning
  - Αποθηκεύει τα RDF δεδομένα στην κύρια μνήμη όλων των κόμβων
  - 6 indexes κια aggregated statistics
  - MPI-based asynchronous join execution
  - Βασισμένο στο RDF-3X για join planning

Workload-adaptive engines

- Προσαρμογή στο workload
  - Adaptive data partitioning
  - Result caching
  - Multi-query optimization
- Αντιπροσωπευτικά συστήματα
  - SPARQL Multi-query optimization [Le 2012]
  - Partout [Galarraga 2014]
  - Warp [Hose 2014]
  - Chameleon-db [Aluc 2015]
  - H2RDF+ caching [Papailiou 2015]
  - AdPart [Harbi 2016]

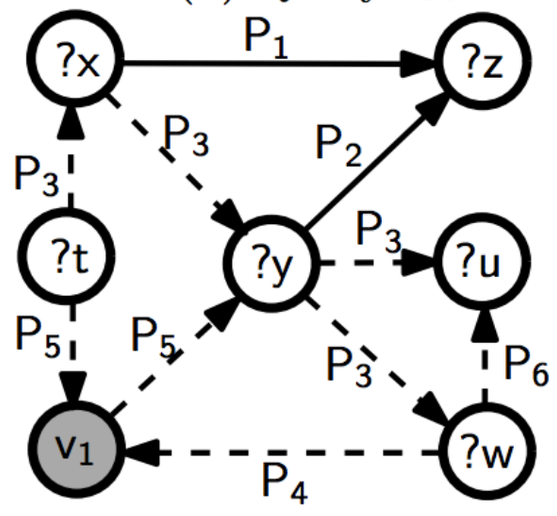
Multi-query optimization

- Συνένωση SPARQL ερωτημάτων που έχουν overlap
  - Χρησιμοποιεί OPTIONAL patterns



(a) Query Q_1 (b) Query Q_2 (c) Query Q_3 (d) Query Q_4

```
SELECT *
WHERE { ?x P1 ?z, ?y P2 ?z,
    OPTIONAL {?y P3 ?w, ?w P4 v1 }
    OPTIONAL {?t P3 ?x, ?t P5 v1, ?w P4 v1 }
    OPTIONAL {?x P3 ?y, v1 P5 ?y, ?w P4 v1 }
    OPTIONAL {?y P3 ?u, ?w P6 ?u, ?w P4 v1 }
}
```



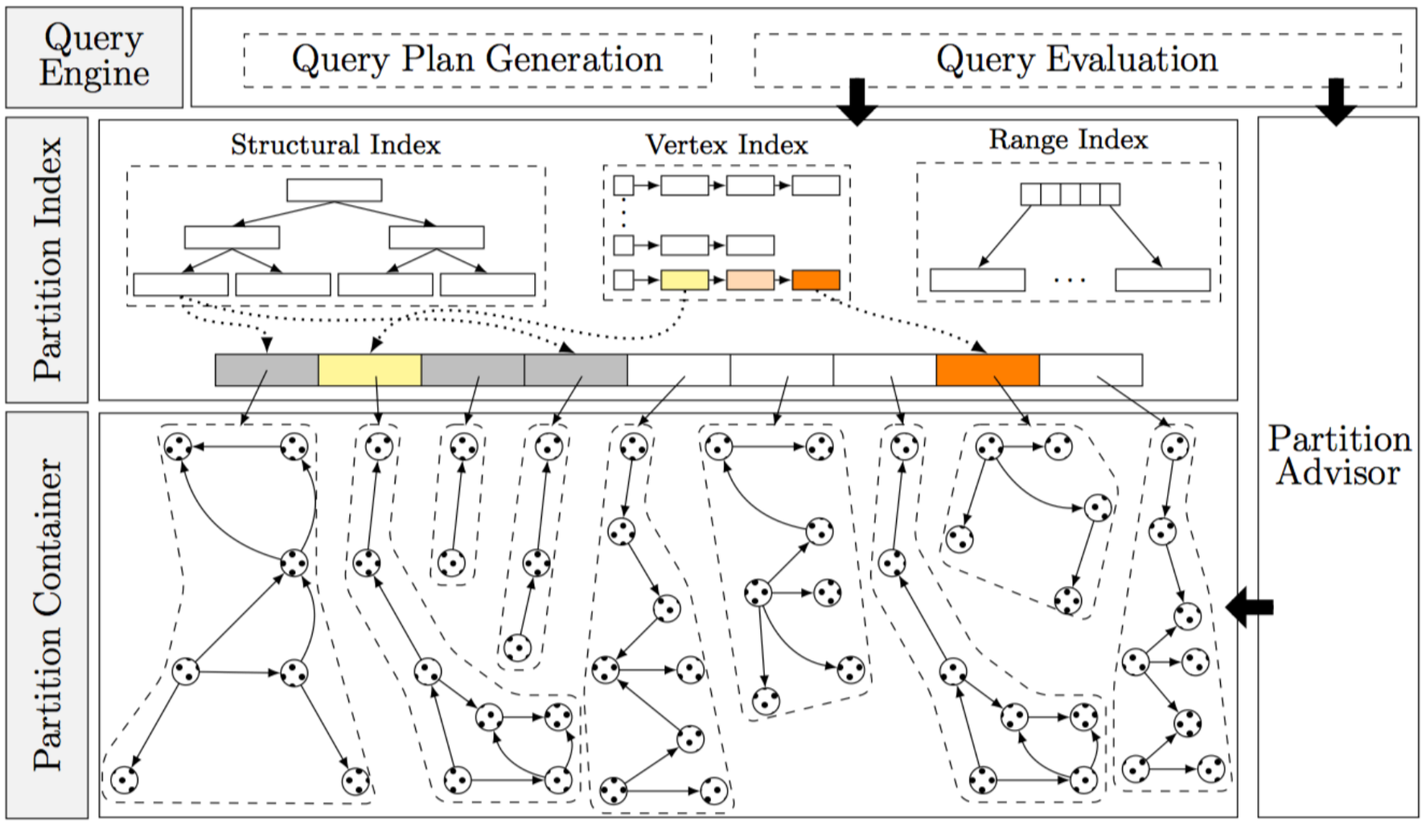
(e) Example query Q_{OPT}

Partout - Warp

- Επεξεργασία του workload των ερωτημάτων
  - Επεκτείνουν graph partitioning τεχνικές
  - Εύρεση ενός καλού graph partitioning με βάση τα ερωτήματα
  - Adaptive replication των RDF τριάδων
  - Ελαχιστοποίηση της επικοινωνίας κατά την εκτέλεση
- Εκτέλεση ερωτημάτων
  - Επικοινωνία μόνο για τα κομμάτια του ερωτήματος που χρειάζονται απομακρυσμένα δεδομένα

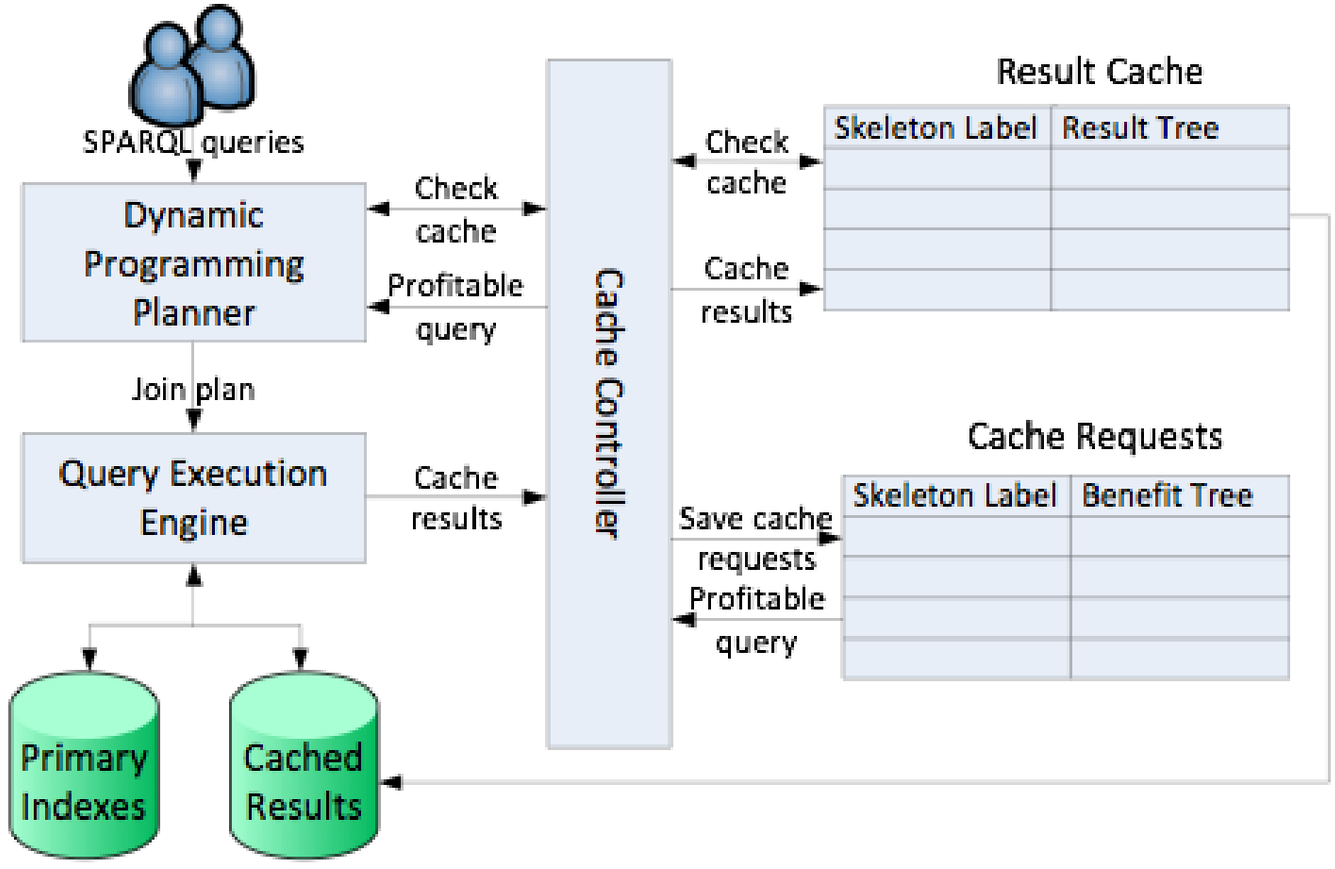
Chameleon-db

- Χρήση ευρετηρίων που περιέχουν αποτελέσματα για graph query patterns
  - Partition με βάση graph patterns
  - Εύρεση του καλύτερου partitioning με βάση τα ερωτήματα του workload



H2RDF+ SPARQL caching

- Canonical labelling για SPARQL ερωτήματα
- Χρήση του label για δημιουργία κρυφής μνήμης
- Επέκτεση ενός DP planner για cache requests
  - Ελέγχει αν η κρυφή μνήμη περιέχει χρήσιμα αποτελέσματα για κάθε subgraph του ερωτήματος
- Cache controller παρακολουθεί τα cache requests
  - Βρίσκει και εκτελεί “profitable” ερωτήματα



AdPart

- Αρχικό Hash based partitioning
  - Ομαδοποίηση των δεδομένων με βάση το subject τους
  - Star ερωτήματα μπορούν να εκτελεστούν με τοπικά δεδομένα
- Παρακολούθηση του workload
  - Εύρεση των hot query pattern
  - Συνδυασμός επιμέρους ερωτημάτων με χρήση OPTIONAL
  - Repartition ή replication αυτών των pattern
- Εκτέλεση ερωτημάτων
  - MPI asynchronous joins
  - Αν τα δεδομένα μπορούν να βρεθούν τοπικά εκτελείται το ερώτημα χωρίς επικοινωνία

Ερωτήσεις