Module 15

RDF, SPARQL and Semantic Repositories

ontotext

GATE

Module 15 Outline

9.45-11.00	• RDF/S and OWL formal semantics and profiles • Querying RDF data with SPARQL
11.00-11.15	Coffee break
11.15-12.30	• Semantic Repositories • OWLIM overview
12.30-14.00	Lunch Break
14.00-16.00	• Benchmarking triplestores • Distributed approaches to RDF materialisation • From RDBMS to RDF • Other RDF tools

ontotext

GATE

About this tutorial

- RDF/S and OWL formal semantics and profiles
- Querying RDF data with SPARQL
- Semantic Repositories
- OWLIM overview
- Benchmarking triplestores
- Distributed approaches to RDF materialisation
- From RDBMS to RDF
- Other RDF tools

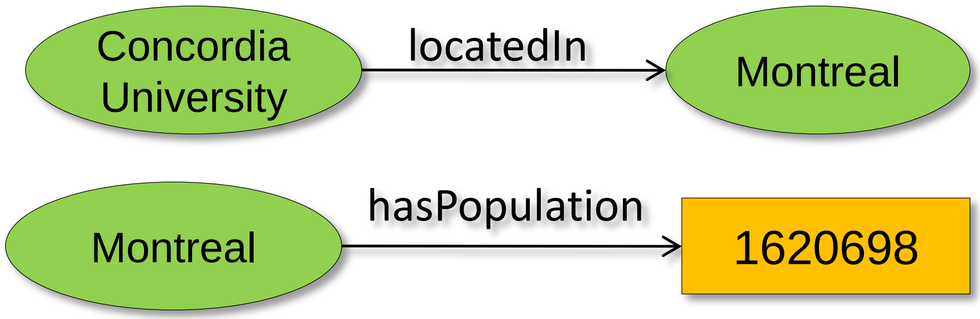
RDF/S and OWL formal semantics, dialects & profiles

Resource Description Framework (RDF)

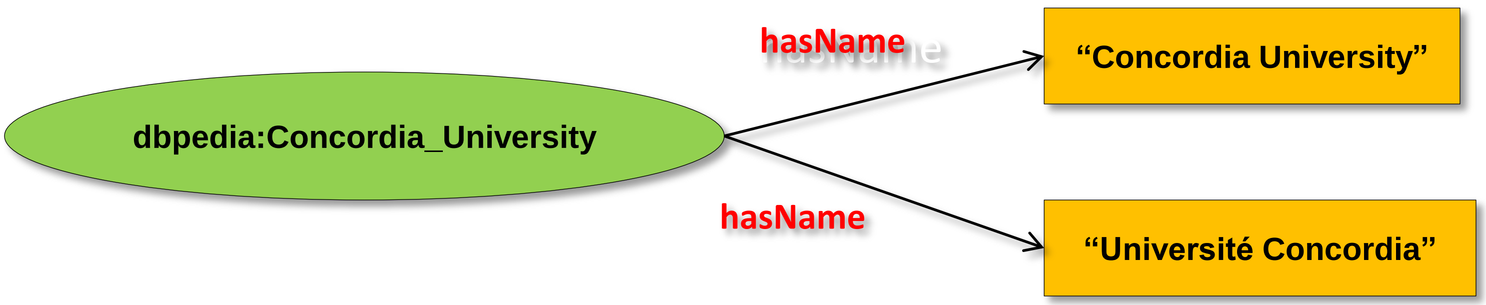
- A simple data model for
  - describing the *semantics* of information in a machine accessible way
  - representing meta-data (data about data)
- A set of representation syntaxes
  - XML (standard) but also JSON, N3, Turtle, …
- Building blocks
  - *Resources* (with unique identifiers)
  - *Literals*
  - Named *relations* between pairs of resources (or a resource and a literal)

RDF (2)

- Everything is a triple (statement)
  - **Subject** (resource), **Predicate** (relation), **Object** (resource or literal)
- The RDF graph is a collection of triples



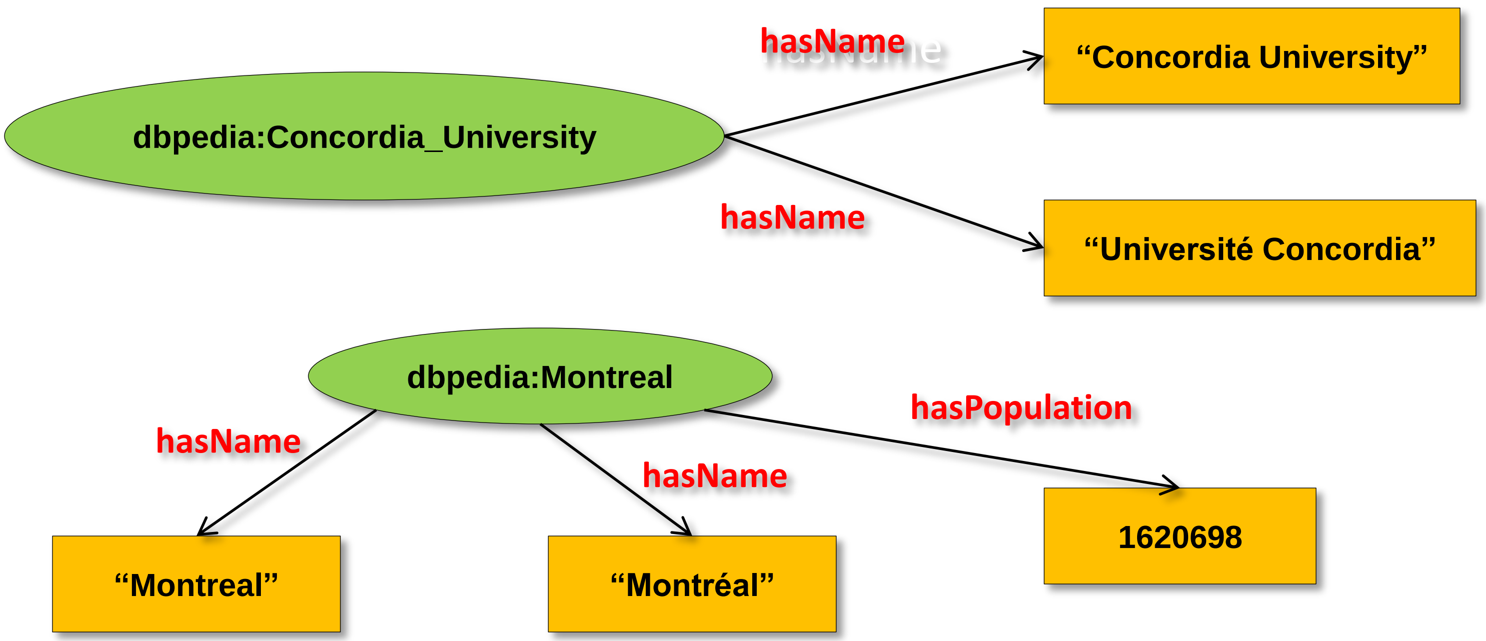
RDF (3)



Subject	Predicate	Object
http://dbpedia.org/resource/Concordia_University	hasName	"Concordia University"
http://dbpedia.org/resource/Concordia_University	hasName	"Université Concordia"

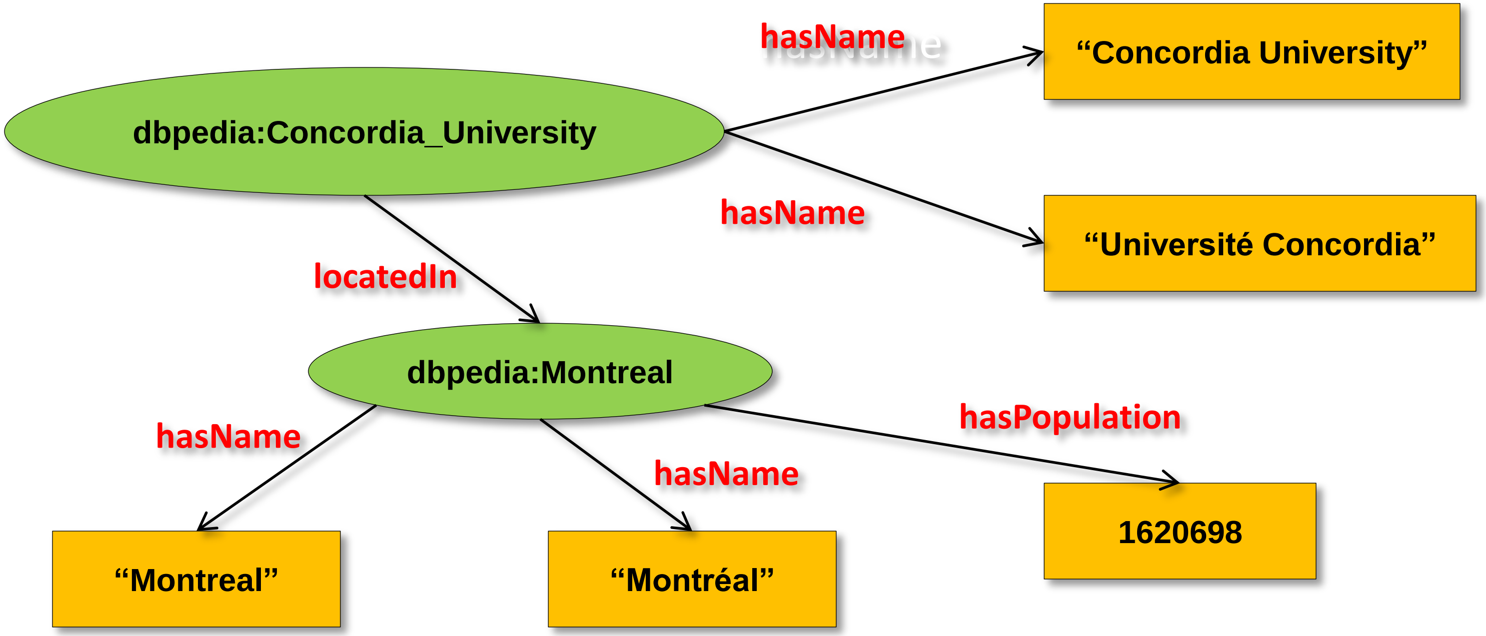
ontotext

RDF (4)



Subject	Predicate	Object
http://dbpedia.org/resource/Montreal	*hasName*	*"Montreal"*
http://dbpedia.org/resource/Montreal	*hasPopulation*	*1620698*
http://dbpedia.org/resource/Montreal	*hasName*	*"Montréal"*
http://dbpedia.org/resource/Concordia_University	**hasName**	**"Concordia University"**
http://dbpedia.org/resource/Concordia_University	**hasName**	**"Université Concordia"**

RDF (5)



Subject	Predicate	Object
http://dbpedia.org/resource/Montreal	**hasName**	**"Montreal"**
http://dbpedia.org/resource/Montreal	**hasPopulation**	**1620698**
http://dbpedia.org/resource/Montreal	**hasName**	**"Montréal"**
http://dbpedia.org/resource/Concordia_University	*locatedIn*	*http://dbpedia.org/resource/Montreal*
http://dbpedia.org/resource/Concordia_University	**hasName**	**"Concordia University"**
http://dbpedia.org/resource/Concordia_University	**hasName**	**"Université Concordia"**

RDF (4)

- RDF advantages
  - Simple but expressive data model
  - Global identifiers of all resources
    - Remove ambiguity
  - Easier & incremental data integration
    - Can handle incomplete information
    - Open world assumption
  - Schema agility
  - Graph structure
    - Suitable for a large class of tasks
    - Data merging is easier

RDF Schema (RDFS)

- RDFS provides means for
  - Defining *Classes* and *Properties*
  - Defining hierarchies (of classes and properties)
- RDFS differs from XML Schema (XSD)
  - Open World Assumption
  - RDFS is about describing resources, not about validation
- Entailment rules (axioms)
  - Infer new triples

RDFS entailment rules

1:	$s\ p\ o$ (if o is a literal)		$\Rightarrow$ _:n rdf:type rdfs:Literal
2:	p rdfs:domain x	& $s\ p\ o$	$\Rightarrow$ s rdf:type x
3:	p rdfs:range x	& $s\ p\ o$	$\Rightarrow$ o rdf:type x
4a:	$s\ p\ o$		$\Rightarrow$ s rdf:type rdfs:Resource
4b:	$s\ p\ o$		$\Rightarrow$ o rdf:type rdfs:Resource
5:	p rdfs:subPropertyOf q	& q rdfs:subPropertyOf r	$\Rightarrow$ p rdfs:subPropertyOf r
6:	p rdf:type rdf:Property		$\Rightarrow$ p rdfs:subPropertyOf p
7:	$s\ p\ o$	& p rdfs:subPropertyOf q	$\Rightarrow$ $s\ q\ o$
8:	s rdf:type rdfs:Class		$\Rightarrow$ s rdfs:subClassOf rdfs:Resource
9:	s rdf:type x	& x rdfs:subClassOf y	$\Rightarrow$ s rdf:type y
10:	s rdf:type rdfs:Class		$\Rightarrow$ s rdfs:subClassOf s
11:	x rdfs:subClassOf y	& y rdfs:subClassof z	$\Rightarrow$ x rdfs:subClassOf z
12:	p rdf:type rdfs:ContainerMembershipProperty		$\Rightarrow$ p rdfs:subPropertyOf rdfs:member
13:	o rdf:type rdfs:Datatype		$\Rightarrow$ o rdfs:subClassOf rdfs:Literal

RDF entailment rules (2)

- Class/Property hierarchies
  - R5, R7, R9, R11

```
:human rdfs:subClassOf :mammal .
:man rdsf:subClassOf :human .
➔ :man rdsf:subClassOf :mammal .
```

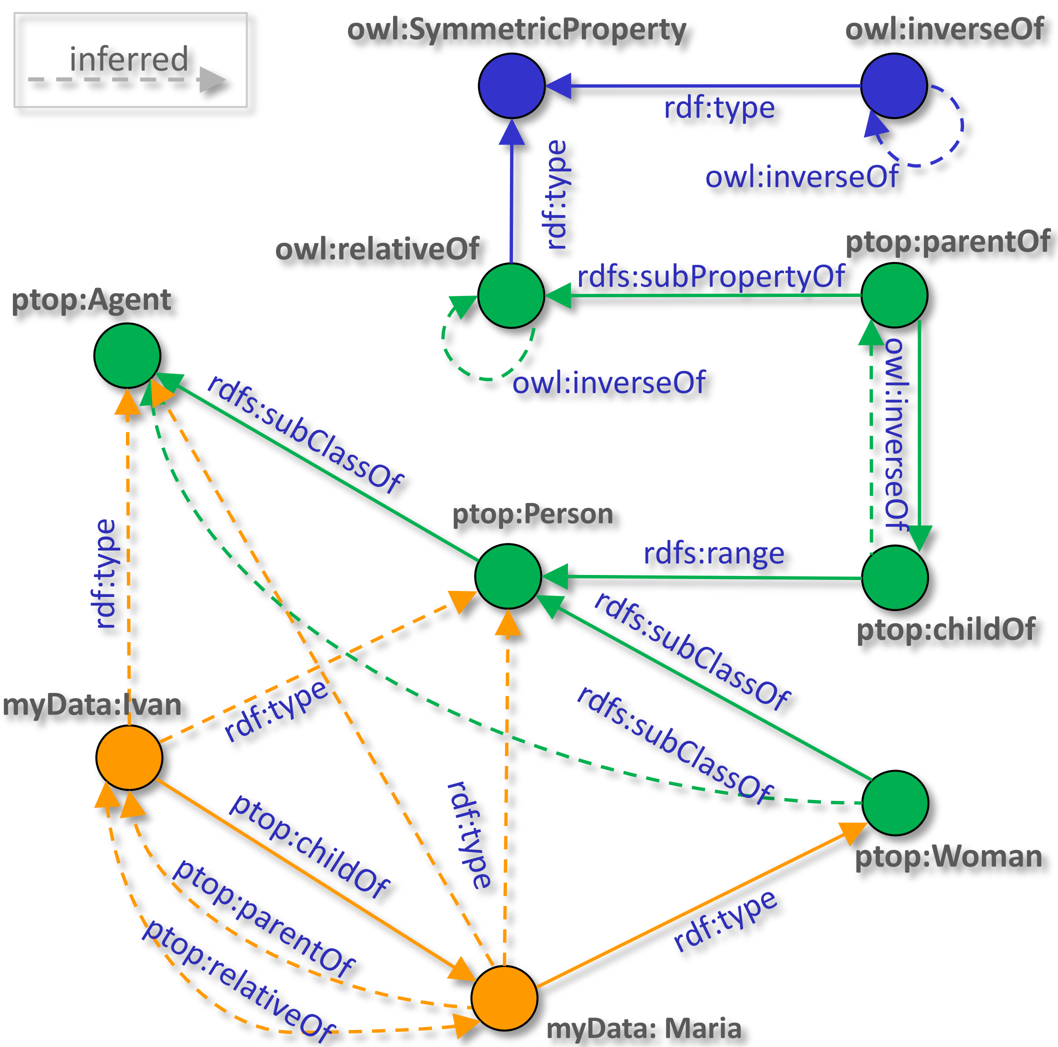
```
:John a :man .
➔ :John a :human .
➔ :John a :mammal .
```

```
:hasSpouse rdfs:subPropertyOf :relatedTo .
:John :hasSpouse :Merry .
➔ :John :relatedTo :Merry .
```

- Inferring types (domain/range restrictions)
  - R2, R3

```
:hasSpouse rdfs:domain :human ;
           rdfs:range :human .
:Adam :hasSpouse :Eve .
➔:Adam a :human .
➔:Eve a :human .
```

RDFS entailment – inferred triples



OWL (2)

- More expressive than RDFS
  - Identity equivalence/difference
    - *sameAs, differentFrom, equivalentClass/Property*
  - More expressive class definitions
    - Class intersection, union, complement, disjointness
    - Cardinality restrictions
  - More expressive property definitions
    - Object/Datatype properties
    - Transitive, functional, symmetric, inverse properties
    - Value restrictions

ontotext

GATE

OWL (3)

- Identity equivalence

```
:Montreal :hasPopulation 1620698 .
:Montreal = :Montréal .
➔ :Montréal :hasPopulation 1620698 .
```

- Transitive properties

```
:locatedIn a owl:TransitiveProperty .
:Montreal :locatedIn :Quebec .
:Quebec :locatedIn :Canada .
➔ :Montreal :locatedIn :Canada.
```

- Symmetric properties

```
:hasSpouse a owl:SymmetricProperty .
:John :hasSpouse :Merry .
➔ :Merry :hasSpouse :John .
```

- Inverse properties

```
:hasParent owl:inverseOf :hasChild .
:John :hasChild :Jane .
➔ :Jane :hasParent :John .
```

- Functional properties

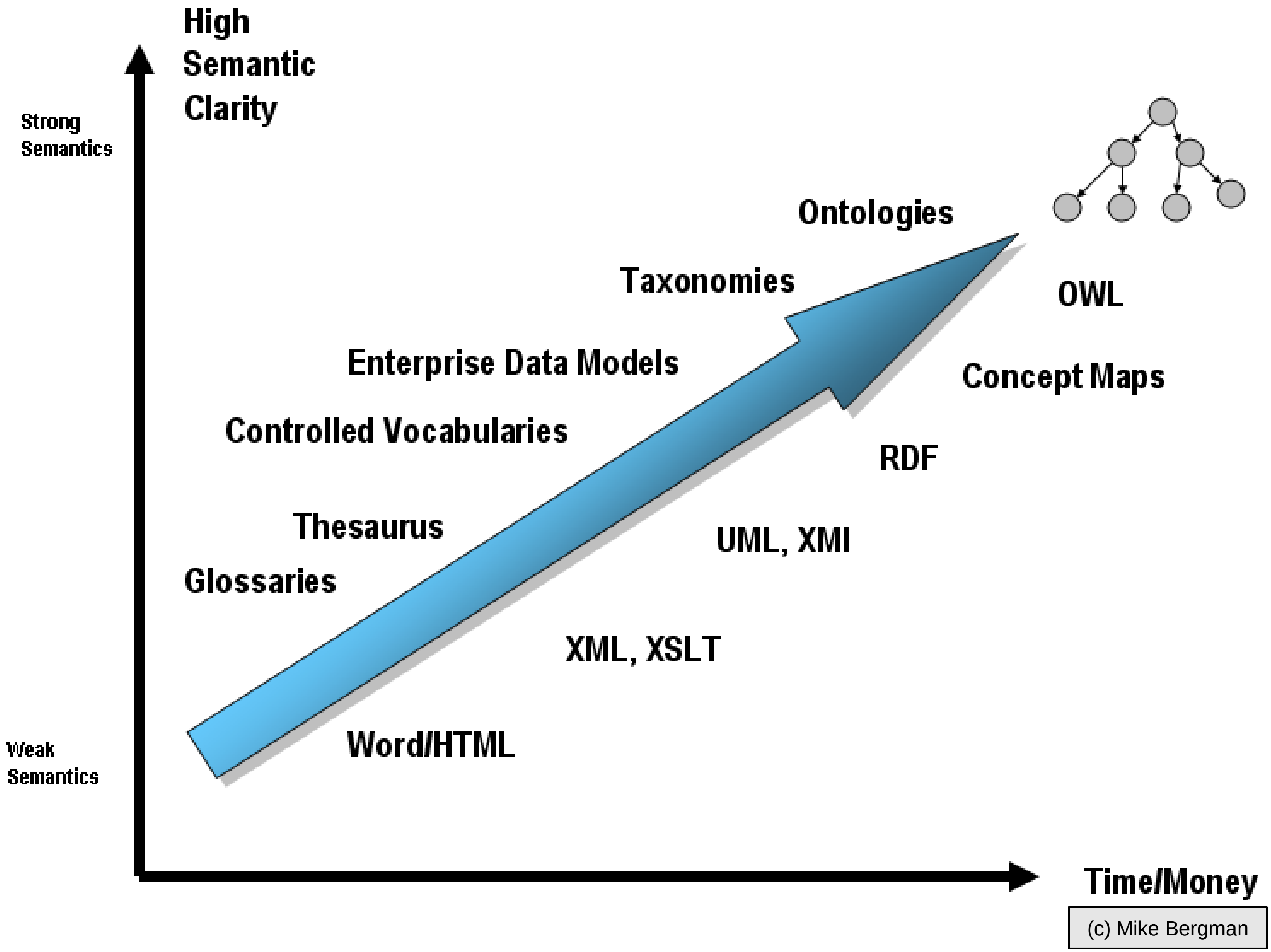
```
:hasSpouse a owl:FunctionalPropety .
:Merry :hasSpouse :John .
:Merry :hasSpouse :JohnSmith .
➔ :JohnSmith = :John .
```

OWL (4)

- Cardinality restrictions

```
:hasSpouse owl:maxCardinality 1 .
:Merry :hasSpouse :John .
:Merry :hasSpouse :JohnSmith .
➔ :JohnSmith = :John .
```

The cost of semantic clarity



OWL sublanguages – OWL Lite

- OWL Lite
  - low expressiveness / low computational complexity
  - All RDFS features
  - *sameAs*/*differentFrom*, equivalent class/property
  - inverse/symmetric/transitive/functional properties
  - property restrictions, cardinality (0 or 1)
  - class intersection

OWL DL & OWL Full

- OWL DL
  - high expressiveness / decidable & complete
  - All OWL Lite features
  - Class disjointness
  - Complex class expressions
  - Class union & complement
- OWL Full
  - max expressiveness / no guarantees
  - Same vocabulary as OWL-DL but less restrictions
    - In OWL DL, a Class <u>cannot</u> also be an Individual or a Property

OWL 2 profiles

- Goals
  - sublanguages that trade expressiveness for efficiency of reasoning
  - Cover important application areas
  - Easier to understand by non-experts
- *OWL 2 EL*
  - Best for large ontologies / small instance data (TBox reasoning)
  - A near maximal fragment of OWL2
  - Computationally optimal
    - Satisfiability checks in PTime

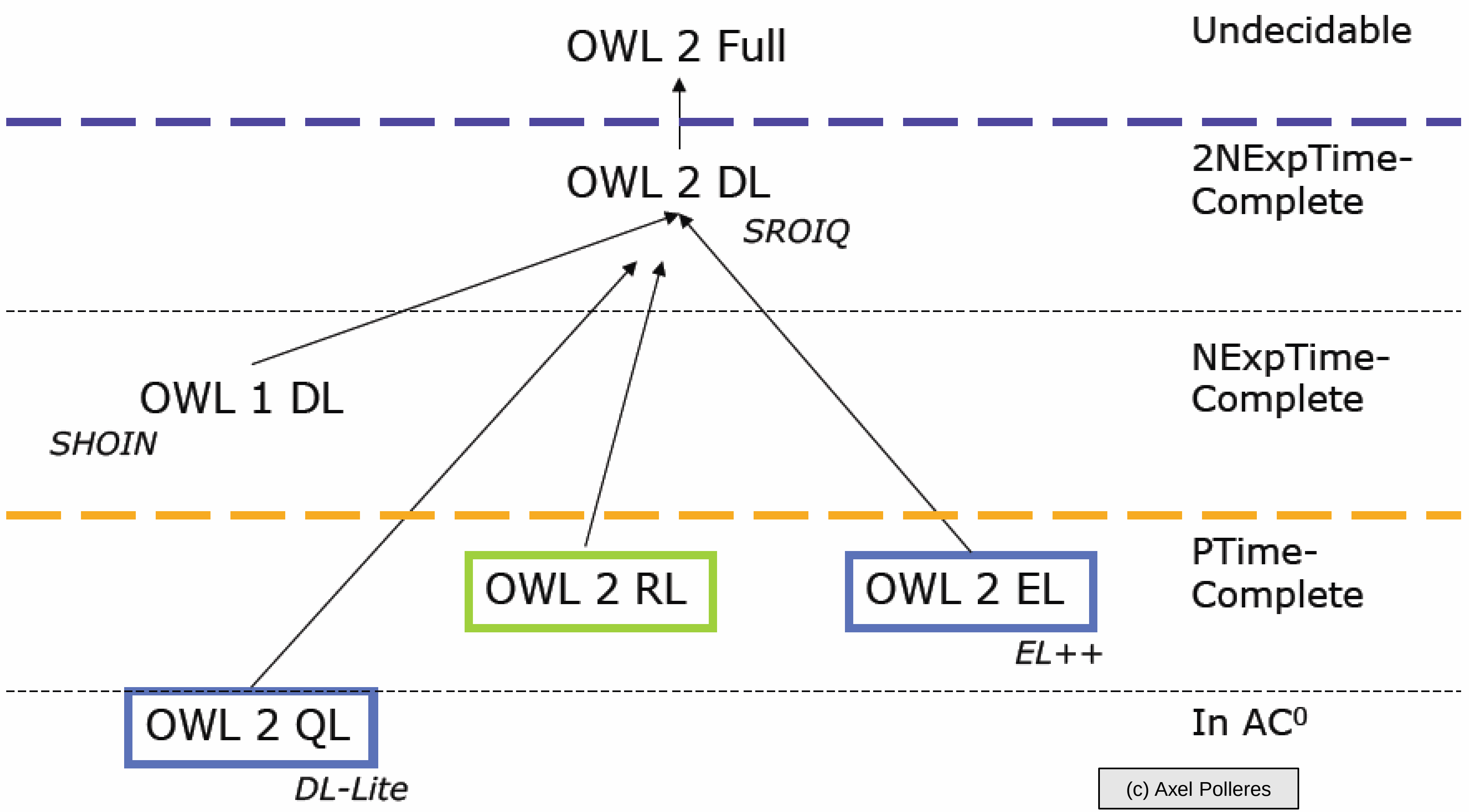
OWL 2 profiles (2)

- *OWL 2 QL*
  - Quite limited expressive power, but very efficient for query answering with large instance data
  - Can exploit query rewriting techniques
    - Data storage & query evaluation can be delegated to a RDBMS
- OWL 2 RL
  - Balance between scalable reasoning and expressive power (ABox reasoning)
  - OWL 2 RL rules can be expressed in RIF BLD

ontotext

OWL 2 profiles (3)

GATE



Querying RDF data with SPARQL

SPARQL Protocol and RDF Query Language (SPARQL)

- SQL-like query language for RDF data
- Simple protocol for querying remote databases over HTTP
- Query types
  - *select* – projections of variables and expressions
  - *construct* – create triples (or graphs) based on query results
  - *ask* – whether a query returns results (result is true/false)
  - *describe* – describe resources in the graph

Describing resources

- Go to www.FactForge.net and execute (in the "SPARQL query" tab):

```
PREFIX dbpedia: <http://dbpedia.org/resource/>

DESCRIBE dbpedia:Montreal
```

SPARQL select

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbp-ont: <http://dbpedia.org/ontology/>

SELECT DISTINCT ?university ?students
WHERE {
   ?university rdf:type dbpedia:University .
   ?university dbp-ont:numberOfStudents ?students .
   ?university dbp-ont:city dbpedia:Montreal .
   FILTER (?students > 5000)
}
ORDER BY DESC (?students)
```

FactForge

SPARQL Query

Results for PREFIX rdf:<http://www.w3... (6) View as Exhibit Downlo

university	students
dbpedia:Con_U	43944
dbpedia:HEC_Montreal	12000
dbpedia:Collège_Ahuntsic	10100
dbpedia:John_Molson_School_of_Business	8026
dbpedia:CEGEP_Vanier	6100
dbpedia:Cégep_du_Vieux_Montréal	6100

ontotext

GATE

Triple patterns

- Whitespace separated list of Subj, Pred, Obj
  - *?x dbp-ont:city dbpedia:Montreal*
  - *dbpedia:Concordia_University db-ont:city ?x*
- Triple patterns with common Subject

```
?uni rdf:type dbpedia:University .
?uni dbp-ont:city dbpedia:Montreal .
```

```
?uni rdf:type dbpedia:University ;
     dbp-ont:city dbpedia:Montreal .
```

  - Triple patterns with common Subject and Predicate

Triple patterns (2)

- Triple patterns with common Subject and Predicate

```
?city rdf:label 'Montreal'@en .
?city rdf:label 'Montréal'@fr .
```

```
?city rdf:label 'Montreal'@en ,'Montréal'@fr .
```

- "a" can be used as an alternative for rdf:type

```
?uni rdf:type dbpedia:University .
```

```
?uni a dbpedia:University .
```

Graph Patterns

- Basic Graph Pattern
  - A conjunction of triple patterns
- Group Graph Pattern
  - A group of 1+ graph patterns
  - Patterns are enclosed in { }
  - FILTERs can constrain the whole group

```
{
   ?uni a dbpedia:University ;
        dbp-ont:city dbpedia:Montreal ;
        dbp-ont:numberOfStudents ?students .
   FILTER (?students > 5000)
}
```

Graph Patterns (2)

- Optional Graph Pattern
  - Optional parts of a graph patterns
  - *pattern OPTIONAL {pattern}*

```
SELECT ?uni ?students
WHERE {
   ?uni a dbpedia:University ;
        dbp-ont:city dbpedia:Montreal .
   OPTIONAL {
      ?uni dbp-ont:numberOfStudents ?students
   }
}
```

Graph Patterns (3)

- Alternative Graph Pattern
  - Combine results of several alternative graph patterns
  - *{pattern} UNION {pattern}*

```
SELECT ?uni
WHERE {
   ?uni a dbpedia:University .
   {
      { ?uni dbp-ont:city dbpedia:Vancouver }
      UNION
      { ?uni dbp-ont:city dbpedia:Montreal }
   }
}
```

Anatomy of a SPARQL query

- List of namespace prefixes
  - PREFIX xyz: <URI>
- List of variables
  - ?x, $y
- Graph patterns + filters
  - Group, alternative, optional
- Modifiers
  - ORDER BY, DISTINCT, OFFSET/LIMIT

ontotext

GATE

Anatomy of a SPARQL query (2)

```
SELECT ?var1 ?var2
WHERE {
   triple-pattern1 .
   triple-pattern2 .
   OPTIONAL {triple-pattern3}
   OPTIONAL {triple-pattern4}
   FILTER (filter-expr)
}
ORDER BY DESC (?var1)
LIMIT 100
```

ontotext

example

GATE

- Go to www.FactForge.net and execute (in the "SPARQL query" tab)
- Find all universities (and their number of students) which are located in Quebec
  - geo-ont:parentFeature, geo-ont:name, geo-ont:alternativeName
- xxx

example (2)

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbp-ont: <http://dbpedia.org/ontology/>
PREFIX geo-ont: <http://www.geonames.org/ontology#>

SELECT DISTINCT ?university ?city ?students
WHERE {
    ?university rdf:type dbpedia:University ;
                     dbp-ont:city ?city .
    ?city geo-ont:parentFeature ?province .
    ?province geo-ont:name 'Québec' .
    OPTIONAL {
       ?university dbp-ont:numberOfStudents ?students
    }
}
ORDER BY ASC (?city)
```

Semantic Repositories

Semantic Repositories

- Semantic repositories combine the features of:
  - Database management systems (DBMS) and
  - Inference engines
- Rapid progress in the last 5 years
  - Every couple of years the scalability increases by an order of magnitude
- “Track-laying machines” for the Semantic Web
  - Extending the reach of the “data railways” and changing the data-economy by allowing more complex data to be managed at lower cost

Semantic Repositories as Track-laying Machines

Property of MSCUA, University of Washington Libraries. Photo Coll 516

Semantic Repositories as Track-laying Machines (2)

ontotext

RDF graph materialisation

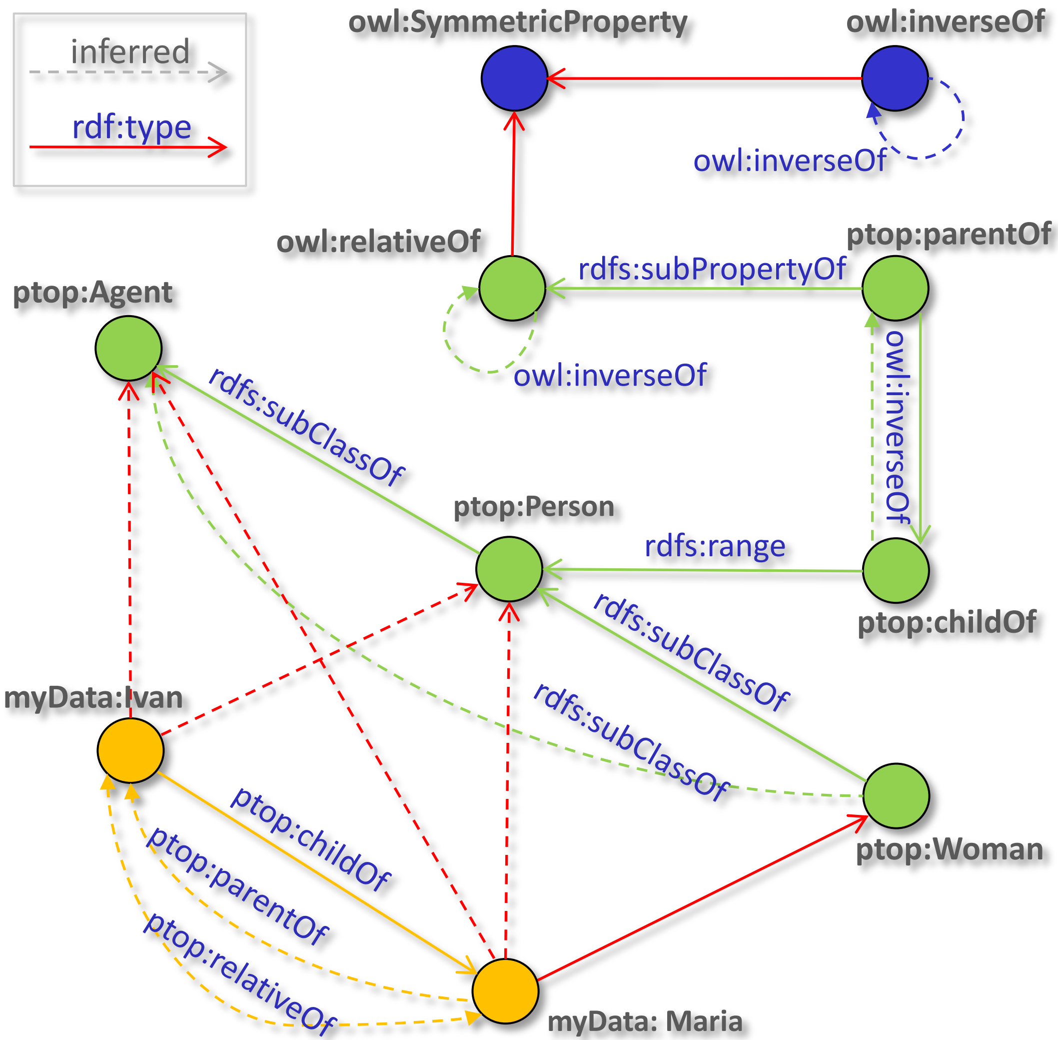
<C1,rdfs:subClassOf,C2>
<C2,rdfs:subClassOf,C3>
⇒ <C1,rdfs:subClassOf,C3>

<I,rdf:type,C1>
<C1,rdfs:subClassOf,C2>
⇒ <I,rdf:type,C2>

<I1,P1,I2>
<P1,rdfs:range,C2>
⇒ <I2,rdf:type,C2>

<P1,owl:inverseOf,P2>
<I1,P1,I2>
⇒ <I2,P2,I1>

<P1,rdf:type,owl:SymmetricProperty>
⇒ <P1,owl:inverseOf,P1>



Semantic Repositories vs. RDBMS

- The major differences with the DBMS are
  - They use **ontologies as semantic schemata**, which allows them to automatically reason about the data
  - They work with a **more generic datamodel**, which allows them more agile to updates and extensions in the schemata (i.e. in the structure of the data)

Physical Data Representation: RDF vs. RDBMS

Person		
ID	Name	Gender
1	Maria P.	F
2	Ivan Jr.	M
3	…	

Parent	
ParID	ChilD
1	2
…	

Spouse			
S1ID	S2ID	From	To
1	3		
…			

Statement		
Subject	Predicate	Object
myo:Person	rdf:type	rdfs:Class
myo:gender	rdfs:type	rdfs:Property
myo:parent	rdfs:range	myo:Person
myo:spouse	rdfs:range	myo:Person
myd:Maria	rdf:type	myo:Person
myd:Maria	rdf:label	"Maria P."
myd:Maria	myo:gender	"F"
myd:Maria	rdf:label	"Ivan Jr."
myd:Ivan	myo:gender	"M"
myd:Maria	myo:parent	Myd:Ivan
myd:Maria	myo:spouse	myd:John
…		

Major characteristics

- *Easy integration of multiple data-sources*
  - Once the schemata of the data-sources is semantically aligned, the inference capabilities of the engine assist the interlinking and combination of facts from different sources
- *Easy querying against rich or diverse data schemata*
  - Inference is applied to match the semantics of the query to the semantics of the data, regardless of the vocabulary and data modeling patterns used for encoding the data

ontotext

Major characteristics (2)

- *Great analytical power*
  - Semantics will be thoroughly applied even when this requires recursive inferences on multiple steps
  - Discover facts, by interlinking long chains of evidence
    - the vast majority of such facts would remain hidden in the DBMS
- *Efficient data interoperability*
  - Importing RDF data from one store to another is straight-forward, based on the usage of globally unique identifiers

Reasoning Strategies

- The strategies for rule-based inference are:
  - **Forward-chaining**: start from the known (explicit) facts and perform inference in an inductive manner until the complete *closure* is inferred
  - **Backward-chaining**: to start from a particular fact and to verify it against the knowledge base using deductive reasoning
    - the reasoner decomposes the query (or the fact) into simpler facts that are available in the KB or can be proven through further recursive decompositions

Reasoning Strategies (2)

- *Inferred closure*
  - the extension of a KB (a graph of RDF triples) with all the implicit facts (triples) that could be inferred from it, based on the pre-defined entailment rules
- *Materialization*
  - Maintaining an up-to-date inferred closure

ontotext

Reasoning Strategies (3)

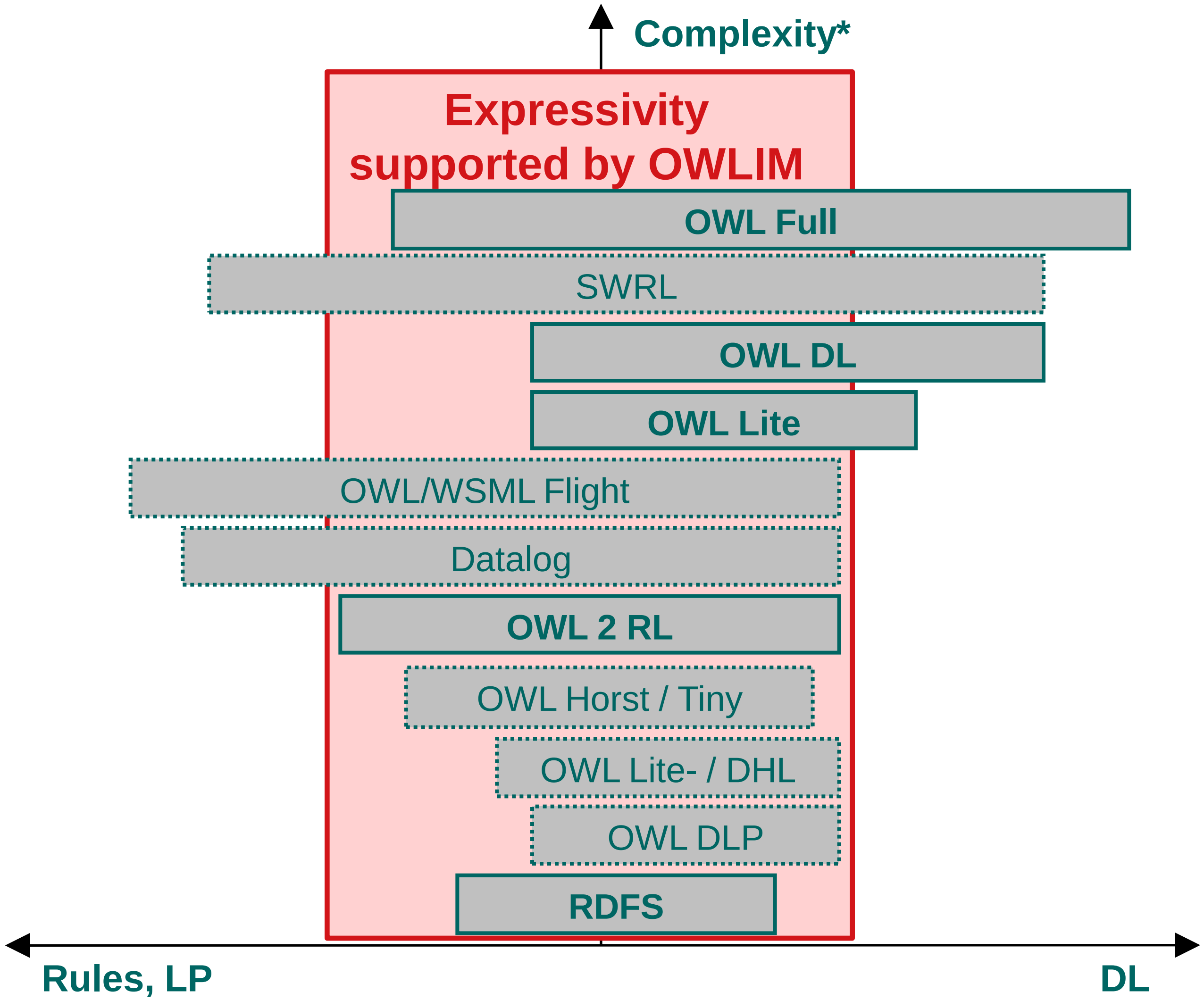
- Pros and cons of forward-chaining based materialization
  - Relatively **slow upload/store/addition** of new facts
    - the repository is extending the inferred closure after each modification (transaction)
  - **Deletion of facts is slow**
    - repository should remove all the facts that are no longer true from the inferred closure
  - The **maintenance of the inferred closure** requires considerable resources
  - **Querying and retrieval are fast**
    - no deduction, satisfiability checking, or other kind of reasoning are required at query time
    - RDBMS-like query evaluation & optimisation techniques are applicable

OWLIM overview

Semantic Repository for RDFS and OWL

- OWLIM is a family of semantic repositories
  - **SwiftOWLIM –** fast in-memory operations, scales to ~100M statements
  - **BigOWLIM –** optimized for data integration, massive query loads and critical applications, scales up to 20B statements
- OWLIM is designed and tuned to provide
  - Efficient management, integration and analysis of heterogeneous data
  - Light-weight, high-performance reasoning

Naïve OWL Fragments Map



BigOWLIM performance

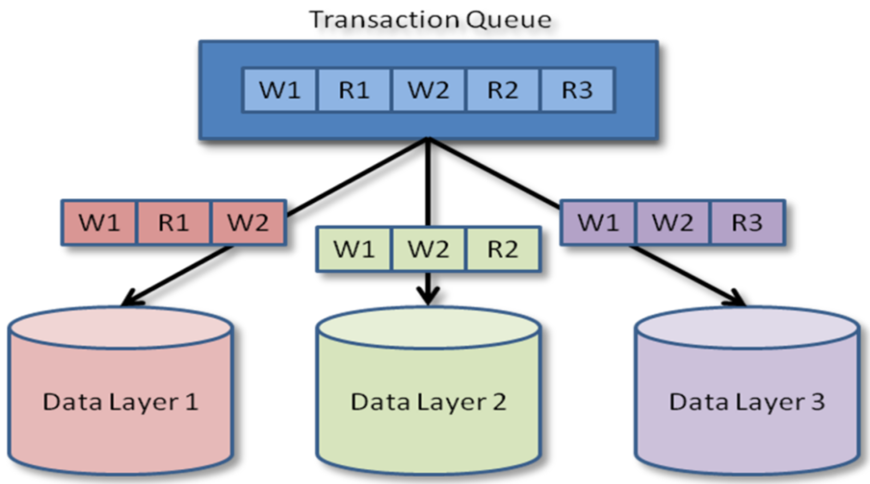
- BigOWLIM is the only engine that can reason with **more than 10B** statements, on a $10,000 server
  - It passes LUBM(90000), indexing **over 20B explicit and implicit statements** and being able to efficiently answer queries
  - It offers the most efficient query evaluation **-** the only RDF database for which full-cycle benchmarking results are published for the LUBM(8000) benchmark or higher

Key Features

- **Clustering support** brings resilience, failover and horizontally scalable parallel query processing
- **Optimized owl:sameAs** handling
- Integrated **full-text search**
- **High performance retraction** of statements and their inferences
- **Consistency checking** mechanisms
- **RDF rank** for ordering query results by relevance
- **Notification mechanism**, to allow clients to react to updates in the data stream

Replication Cluster

- Improve scalability with respect to concurrent user requests
- How does it work?
  - Each data write request is multiplexed to all repository instances
  - Each read request is dispatched to one instance only
  - To ensure load-balancing, read requests are sent to the instance with the shortest execution queue



RDF Rank

- BigOWLIM uses a modification of PageRank over RDF graphs
- The computation of the RDFRank-s for FactForge (couple of billion statements) takes just a few minutes
- Results are available through a system predicate
  - Example**:** get the 100 most “important” nodes in the RDF graph

```
SELECT ?node {?node onto:hasRDFRank ?rank}
ORDER BY DESC(?rank) LIMIT 100
```

ontotext

example

GATE

- Go to www.FactForge.net and execute (in the "SPARQL query" tab)
- Find the 25 "most important" universities located in Quebec
  - geo-ont:parentFeature, geo-ont:name, geo-ont:alternativeName, onto:RR

example (2)

```
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbpedia: <http://dbpedia.org/resource/>
PREFIX dbp-ont: <http://dbpedia.org/ontology/>
PREFIX geo-ont: <http://www.geonames.org/ontology#>
PREFIX om: <http://www.ontotext.com/owlim/>

SELECT DISTINCT ?university ?city ?rank
WHERE {
    ?university rdf:type dbpedia:University ;
                dbp-ont:city ?city ;
                om:hasRDFRank ?rank .
    ?city geo-ont:parentFeature ?province .
    ?province geo-ont:name 'Québec` .
}
ORDER BY DESC (?rank)
LIMIT 25
```

Full-Text Search

- Full-text search is different from SQL-type queries
  - Queries are formulated and evaluated in a different way
  - Different indices are required for efficient handling
  - URIs and literals are retrieved using a set of tokens that should appear in them
  - The matching criteria are determined via system predicates (exact, ignore case, prefix,…)

```
SELECT ?x ?label WHERE {
  ?x rdfs:label ?label .
  <term:> onto:prefixMatchIgnoreCase ?label.
}
```

RDF Search - Advanced FTS in RDF Graphs

- Objective
  - Search in an RDF graph by keywords
  - Get more usable results (standalone literals are not useful in most cases)
- What to index:
  - URIs
  - The *RDF molecule* for an URI
    - the description of the node, including all outgoing statements
- Results returned
  - List of URIs, ranked by FTS + RDF Rank metric

RDF Search – Advanced FTS in RDF Graphs (2)

- The ranking is based on the standard vector-space-model relevance, boosted by RDF Rank

```
PREFIX gossip: <http://www.....gossipdb.owl#>
PREFIX onto: <http://www.ontotext.com/>

SELECT * WHERE {
  ?person gossip:name ?name .
  ?name onto:luceneQuery "American AND life~".
}
```

owl:sameAs Optimisation

- **owl:sameAs** declares that two different URIs denote one and the same resource or object in the world
  - it is used to align different identifiers of the same real-world entity used in different data sources
  - Example, encoding that there are four different URIs for Montreal

```
dbpedia:Montreal owl:sameAs geonames:6077244
dbpedia:Montreal owl:sameAs geonames:6077243
geonames:6077243 owl:sameAs fbase:guid.9202a8c04000641f80000000000028aa7
fbase:guid.9202a8c04000641f80000000000028aa7 owl:sameAs
nytimes:N59179828586486930801
```

owl:sameAs Optimisation (2)

- According to the standard semantics of owl:sameAs:
  - It is a transitive and symmetric relationship
  - Statements, asserted using one of the equivalent URIs should be inferred to appear with all equivalent URIs placed in the same position
  - Thus the 4 statements in the previous example lead to **ten** inferred statements

owl:sameAs Optimisation (3)

```
dbpedia:Montreal owl:sameAs geonames:6077244

dbpedia:Montreal owl:sameAs geonames:6077243

dbpedia:Montreal owl:sameAs
fbase:guid.9202a8c04000641f800000000028aa7

dbpedia:Montreal owl:sameAs nytimes:N59179828586486930801

fbase:guid.9202a8c04000641f800000000028aa7 owl:sameAs
geonames:6077244

fbase:guid.9202a8c04000641f800000000028aa7 owl:sameAs
geonames:6077243

fbase:guid.9202a8c04000641f800000000028aa7 owl:sameAs
nytimes:N59179828586486930801

nytimes:N59179828586486930801 owl:sameAs geonames:6077244

nytimes:N59179828586486930801 owl:sameAs geonames:6077243

geonames:6077244 owl:sameAs geonames:6077243
```

owl:sameAs Optimisation (4)

- BigOWLIM features an optimisation that allows it to use a single master-node in its indices to represent a class of sameAs-equivalent URIs
- Avoids inflating the indices with multiple equivalent statements
- Optionally expands query results
  - The sameAs equivalence leads to multiplication of the bindings of the variables in the process of query evaluation (both forward- and backward-chaining)

Benchmarking triplestores

Tasks to be Benchmarked

- **Data loading**
  - parsing, persistence, and indexing
- **Query evaluation**
  - query preparation and optimization, fetching
- **Data modification**
  - May involve changes to the ontologies and schemata
- *Inference* is not a first-level activity
  - Depending on the implementation, it can affect the performance of the other activities

ontotext

Performance Factors for Loading

- *Materialization*
  - Whether forward-chaining is performed at load time & the complexity of forward-chaining
- *Data model complexity*
  - Support for extended RDF data models (e.g. named graphs), is computationally more expensive
- *Indexing specifics*
  - Repositories can apply different indexing strategies depending on the data loaded, usage patterns, etc.
- *Transaction Isolation*

Performance Factors for Query Evaluation

- *Deduction*
  - Whether and how complex backward-chaining is involved
- *Size of the result-set*
  - Fetching large result-sets can take considerable time
- *Query complexity*
  - Number of constraints (e.g. triple-pattern joins)
  - Semantics of the query (e.g. negation- and disjunction-related clauses)
  - Use of operators that cannot be optimized (e.g. LIKE)
- *Number of concurrent clients*

Performance Dimensions

- *Scale*
  - The size of the repository (number of triples)
- *Schema and data complexity*
  - The complexity of the ontology/logical language
  - The specific ontology (or schema) and dataset
    - E.g. a highly interconnected dataset, with long chains of transitive properties, can appear quite challenging for reasoning
  - Sparse versus dense datasets
  - Presence and size of literals
  - Number of predicates used
  - Use of **owl:sameAs**

Scalability Metrics

- *Number of inserted statements (NIS)*
- *Number of stored statements (NSS)*
  - How many statements have been stored and indexed?
  - Duplicates can make NSS smaller than NIS
  - For engines using forward-chaining and materialization, the volume of data to be indexed includes the inferred triples
- *Number of retrievable statements (NRS)*
  - How many different statements can be retrieved?
    - This number can be different from NSS when the repository supports some sort of backward-chaining

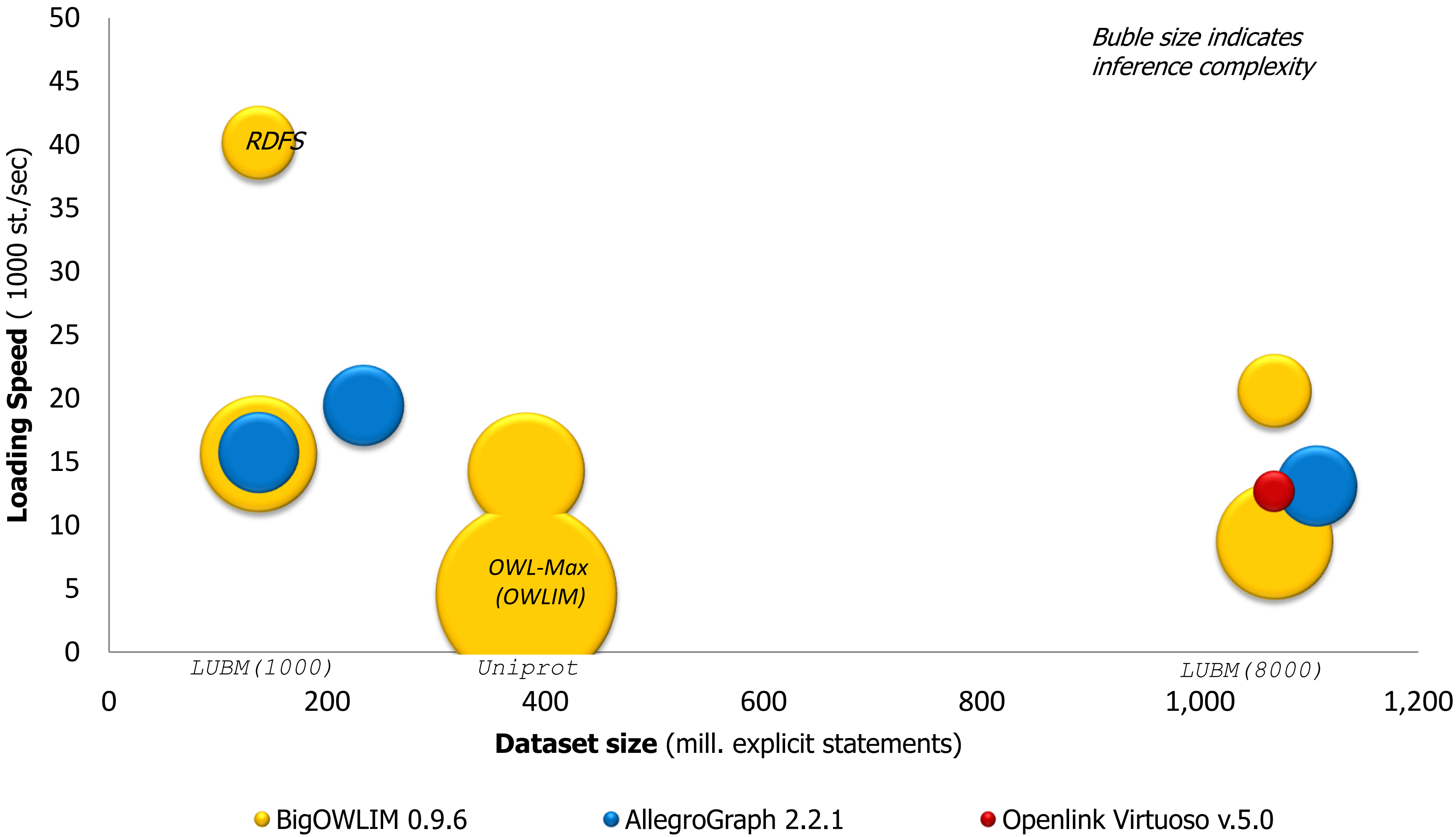
Full-Cycle Benchmarking

- We call *full-cycle benchmarking* any methodology that provides a complete picture of the performance with respect to the full "life cycle" of the data within the engine
  - publication of data for both loading and query evaluation performance in the framework of a single experiment or benchmark run
  - Full-cycle benchmarking requires load performance data to be matched with query evaluation performance
    - "5 billion triples of LUBM were loaded in 30 hours"
    - "… and the evaluation of the 14 queries took 1 hour on a warm database."

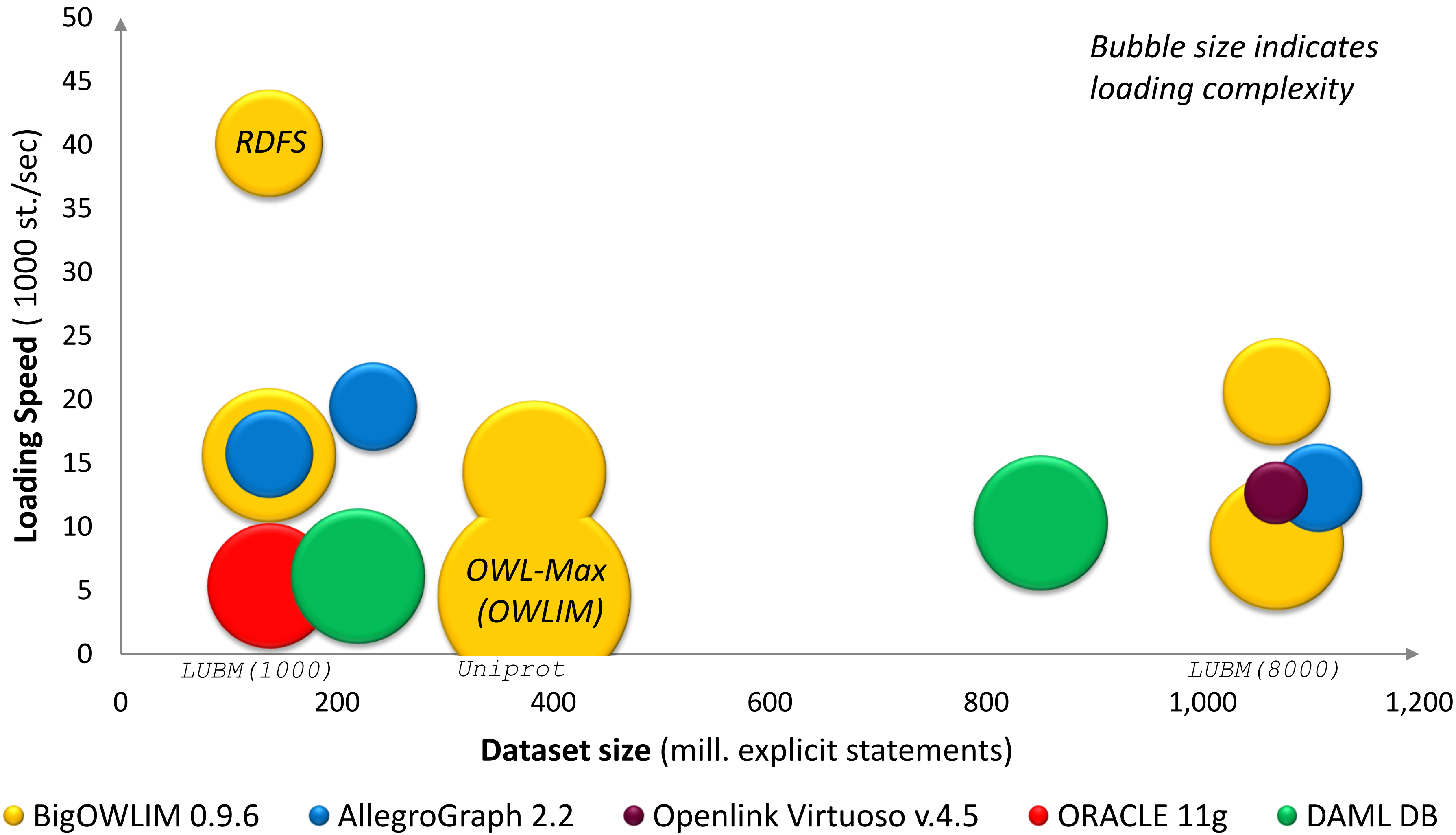
Full-Cycle Benchmarking (2)

- Typical set of activities to be covered:
  1. Loading input RDF files from the storage system
  2. Parsing the RDF files
  3. Indexing and storing the triples
  4. Forward-chaining and materialization (optional)
  5. Query parsing
  6. Query optimization
     - Query re-writing (optional)
  7. Query evaluation, involving
     - Backward-chaining (optional)
     - Fetching the results

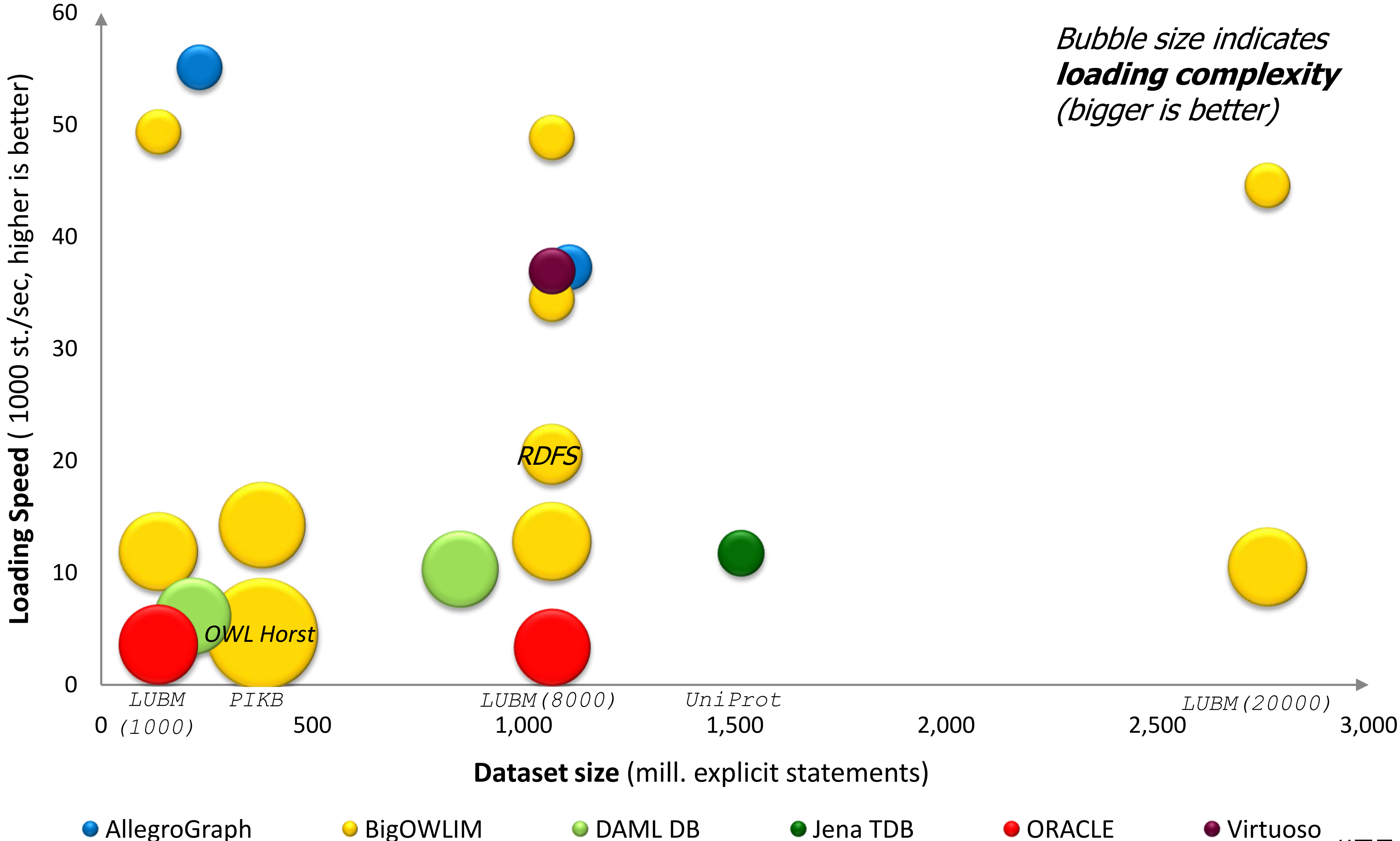
Scalable Inference Map (Sep'07)



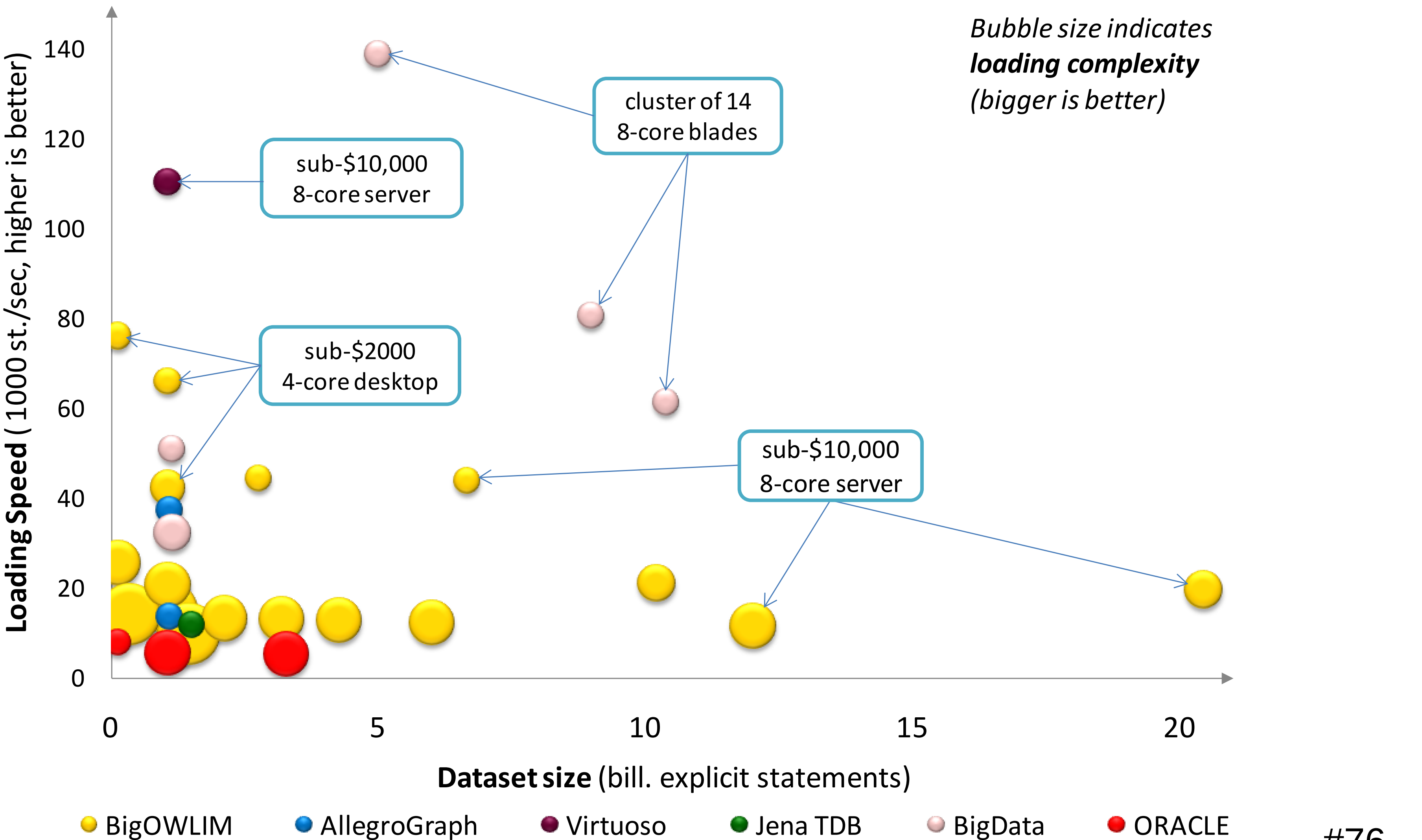
Scalable Inference Map (Nov'07)



Scalable Inference Map (Oct'08)



Scalable Inference Map (Jun'09)

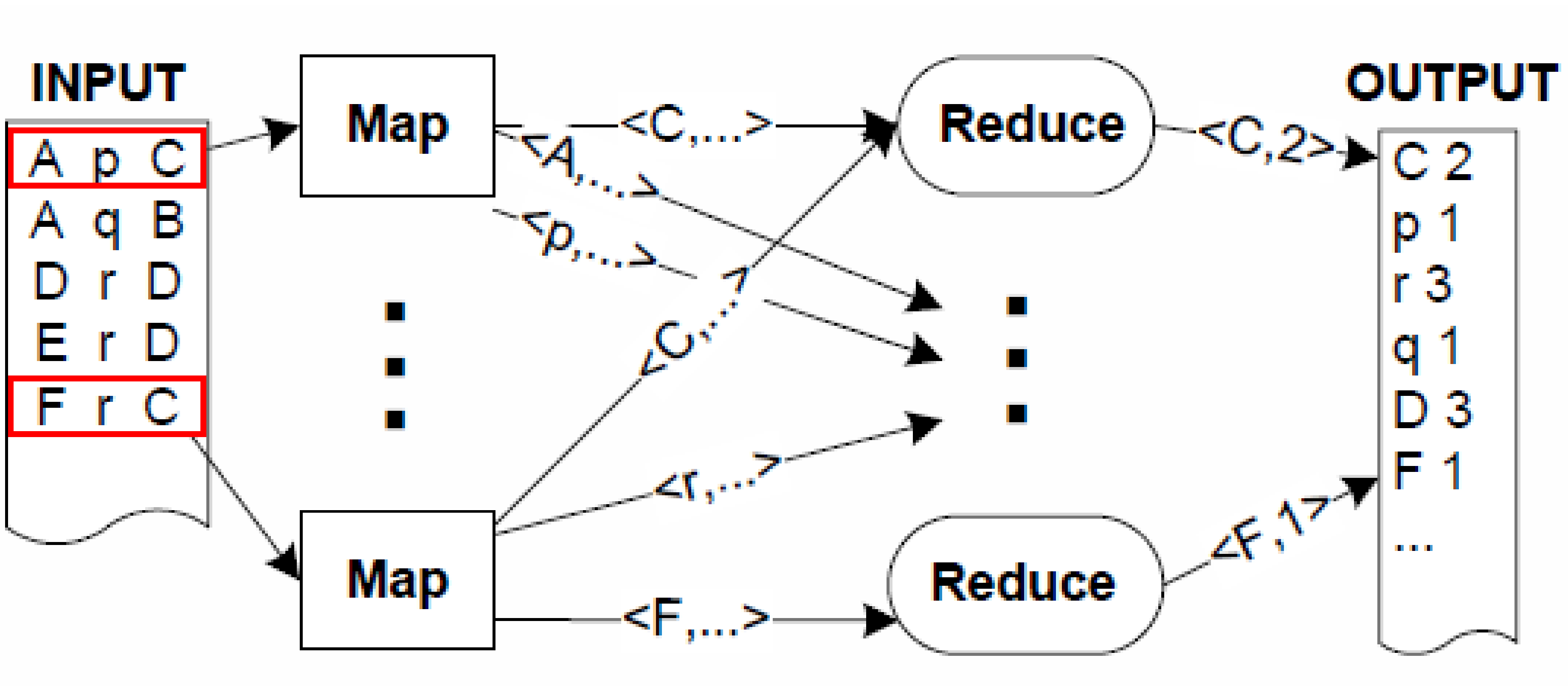


Distributed approaches to RDF materialisation

Distributed RDF materialisation with MapReduce

- Distributed approach by Urbani et al., ISWC'2009
  - "*Scalable Distributed Reasoning using MapReduce*"
- 64 node Hadoop cluster
- MapReduce
  - *Map* phase – partitions the input space by *some* key
  - *Reduce* phase – perform some aggregated processing on a partition (from the *Map* phase)
    - The partition contains all elements for a particular key
    - Skewed key distribution leads to uneven load on *Reduce* nodes
    - Balanced *Reduce* load almost impossible to achieve (major M/R drawback)

Distributed RDF materialisation with MapReduce (2)



(c) Urbani et al.

ontotext

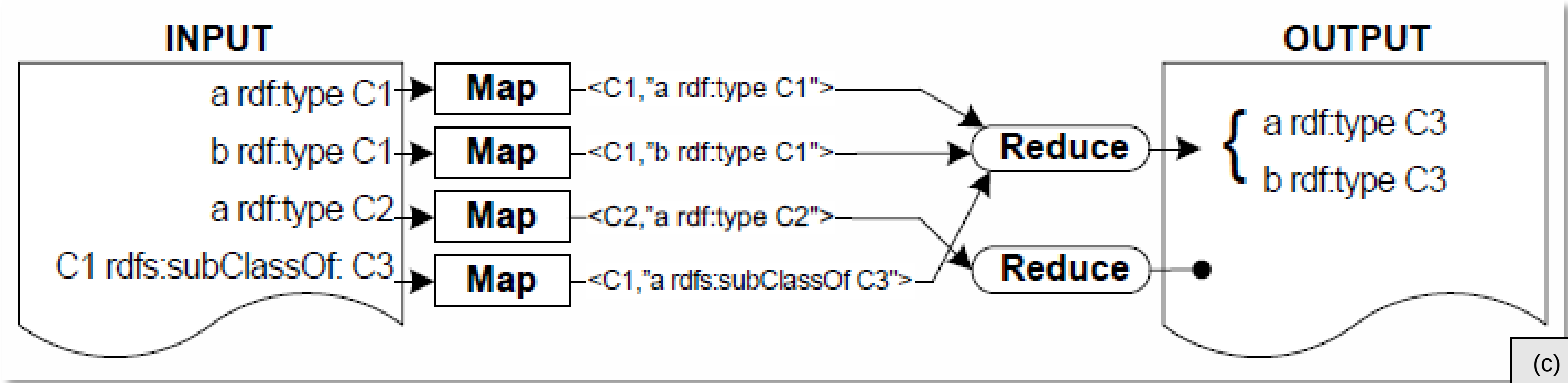
Distributed RDF materialisation with MapReduce (3)

- RDFS entailment rules

1:	$s\ p\ o$ (if o is a literal)		$\Rightarrow$ _:n rdf:type rdfs:Literal
2:	p rdfs:domain x	& $s\ p\ o$	$\Rightarrow$ s rdf:type x
3:	p rdfs:range x	& $s\ p\ o$	$\Rightarrow$ o rdf:type x
4a:	$s\ p\ o$		$\Rightarrow$ s rdf:type rdfs:Resource
4b:	$s\ p\ o$		$\Rightarrow$ o rdf:type rdfs:Resource
5:	p rdfs:subPropertyOf q	& q rdfs:subPropertyOf r	$\Rightarrow$ p rdfs:subPropertyOf r
6:	p rdf:type rdf:Property		$\Rightarrow$ p rdfs:subPropertyOf p
7:	$s\ p\ o$	& p rdfs:subPropertyOf q	$\Rightarrow$ $s\ q\ o$
8:	s rdf:type rdfs:Class		$\Rightarrow$ s rdfs:subClassOf rdfs:Resource
9:	s rdf:type x	& x rdfs:subClassOf y	$\Rightarrow$ s rdf:type y
10:	s rdf:type rdfs:Class		$\Rightarrow$ s rdfs:subClassOf s
11:	x rdfs:subClassOf y	& y rdfs:subClassof z	$\Rightarrow$ x rdfs:subClassOf z
12:	p rdf:type rdfs:ContainerMembershipProperty		$\Rightarrow$ p rdfs:subPropertyOf rdfs:member
13:	o rdf:type rdfs:Datatype		$\Rightarrow$ o rdfs:subClassOf rdfs:Literal

Distributed RDF materialisation with MapReduce (4)

- “naïve” approach
  - applying all RDFS rules iteratively on the input until no new data is derived (fixpoint)
    - rules with one antecedent are easy
    - rules with 2 antecedents require a join
  - Map function
    - Key is *S*, *P* or *O*, value is original triple
  - Reduce function – performs the join



(c) Urbani et al.

Distributed RDF materialisation with MapReduce (4)

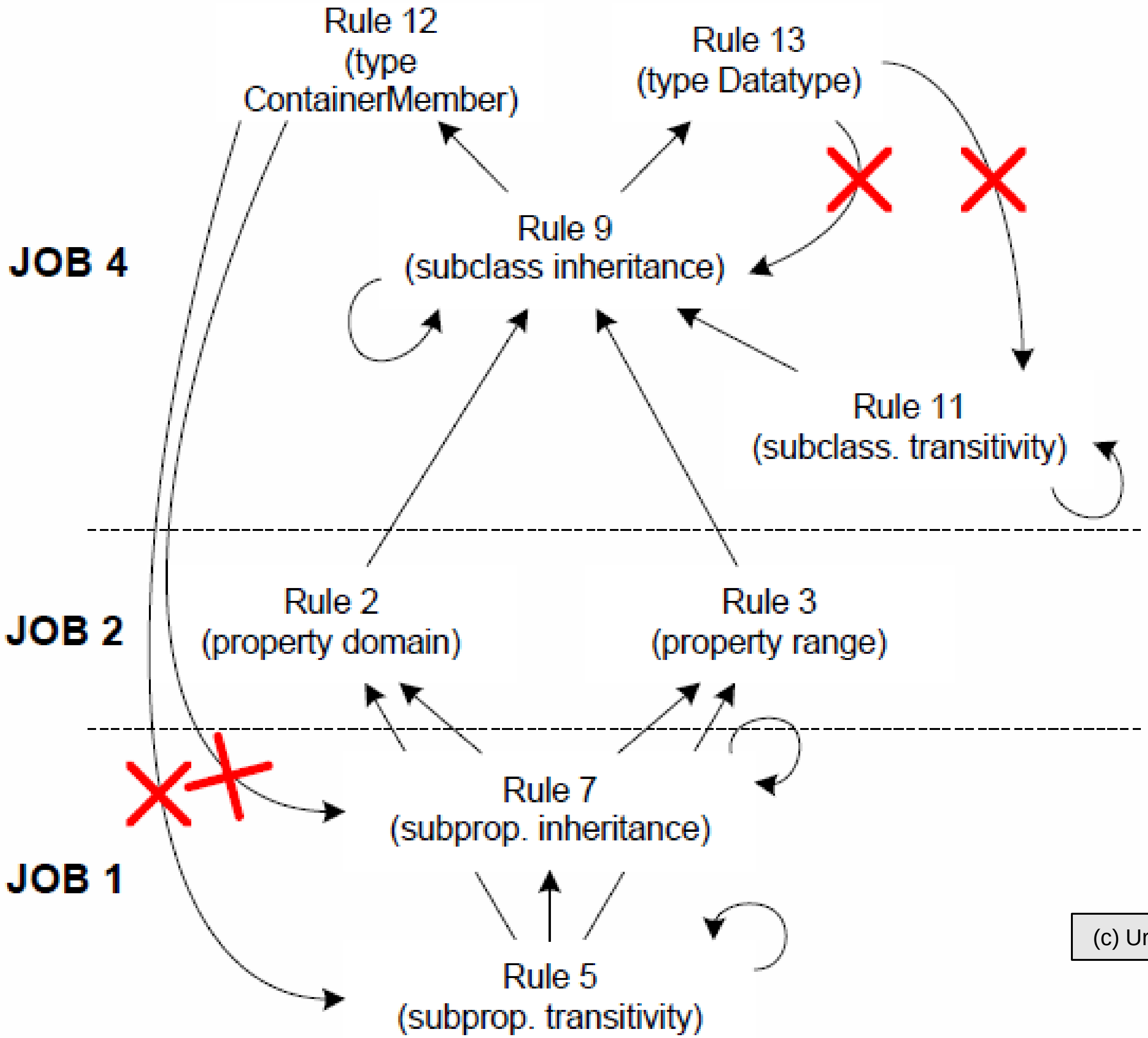
- Problems
  - One iteration is not enough!
  - Too many duplicates generated
    - Ration unique/duplicate triples is more than 1/50
- Optimised approach
  - Load schema triples in memory (0.001-0.01% of triples)
    - On each node joins are made between a very small set of schema triples and a large set of instance triples
    - Only the instance triples are streamed by the MapReduce pipeline

ontotext

Distributed RDF materialisation with MapReduce (5)

- Optimised approach (2)
  - Data Grouping to Avoid Duplicates
    - Map phase: set as key those parts of the input (S/P/O) that are also used in the derived triple. All triples that produce the same triple will be sent to the same Reducer
    - Join schema triples during the *Reduce* phase to reduce duplicates
  - Ordering the Sequence of the Rules
    - Analyse the ruleset and determine which rules may triggered other rules
    - Dependency graph, optimal application of rules from bottom-up

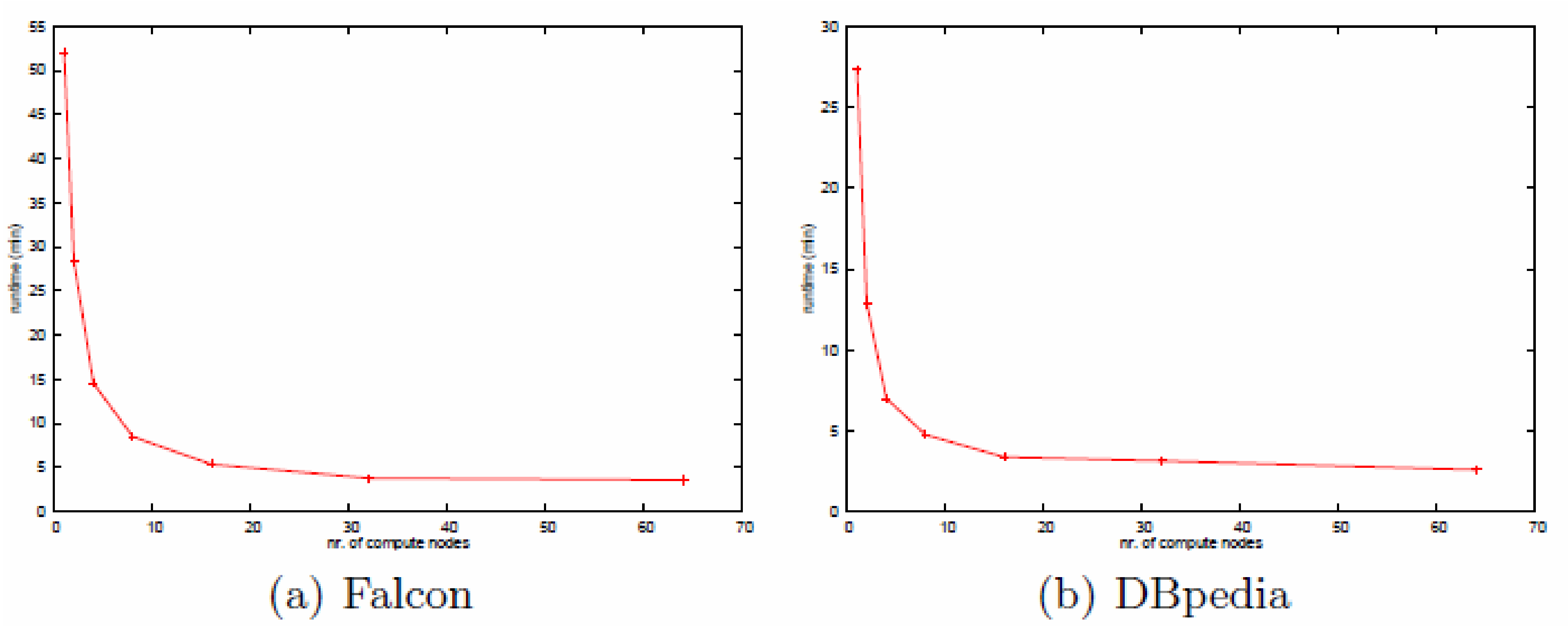
Distributed RDF materialisation with MapReduce (6)



(c) Urbani et al.

Distributed RDF materialisation with MapReduce (7)

- Performance benchmarks
  - 4.3 million triples / sec (30 billion in ~2h)

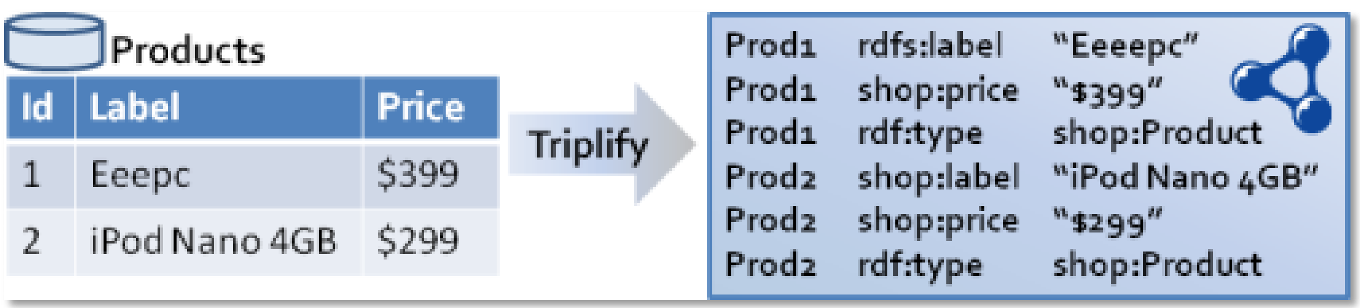


(c) Urbani et al.

From RDBMS to RDF

RDB2RDF

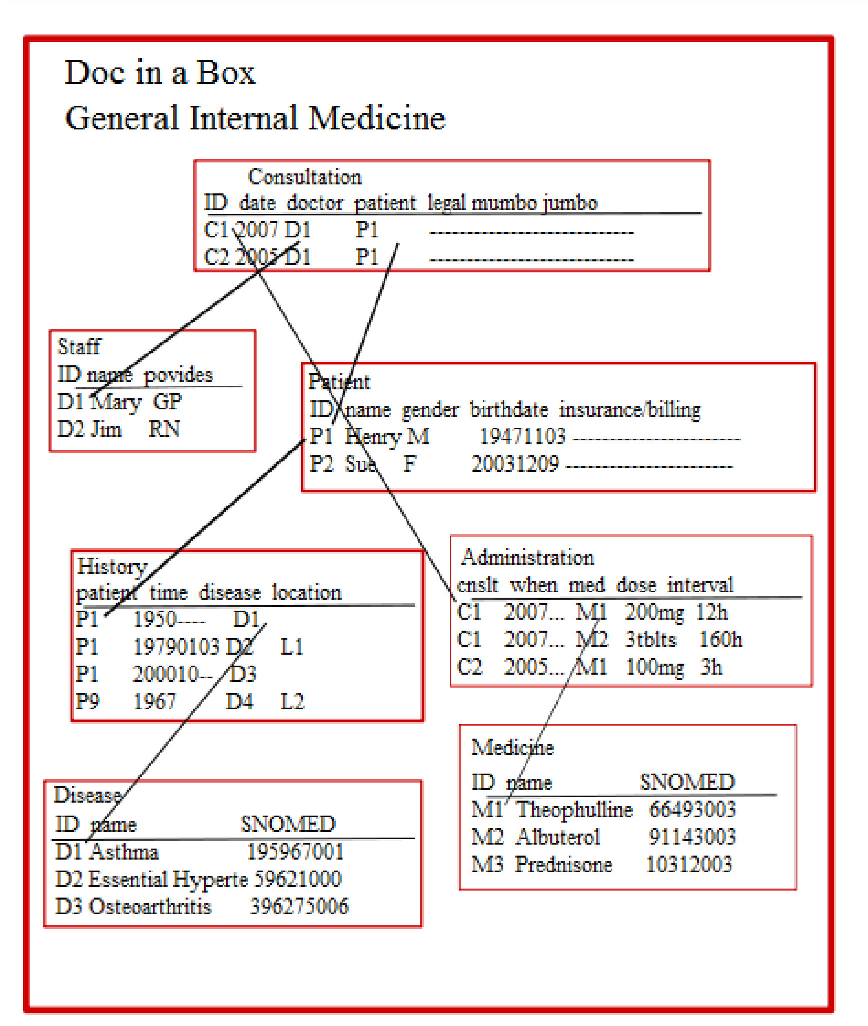
- RDB2RDF working group @ W3C
  - http://www.w3.org/2001/sw/rdb2rdf/
  - standardize a language (R2RML) for mapping relational data / database schemas into RDF / OWL
- Existing RDF-izers
  - Triplify, D2RQ, RDF Views

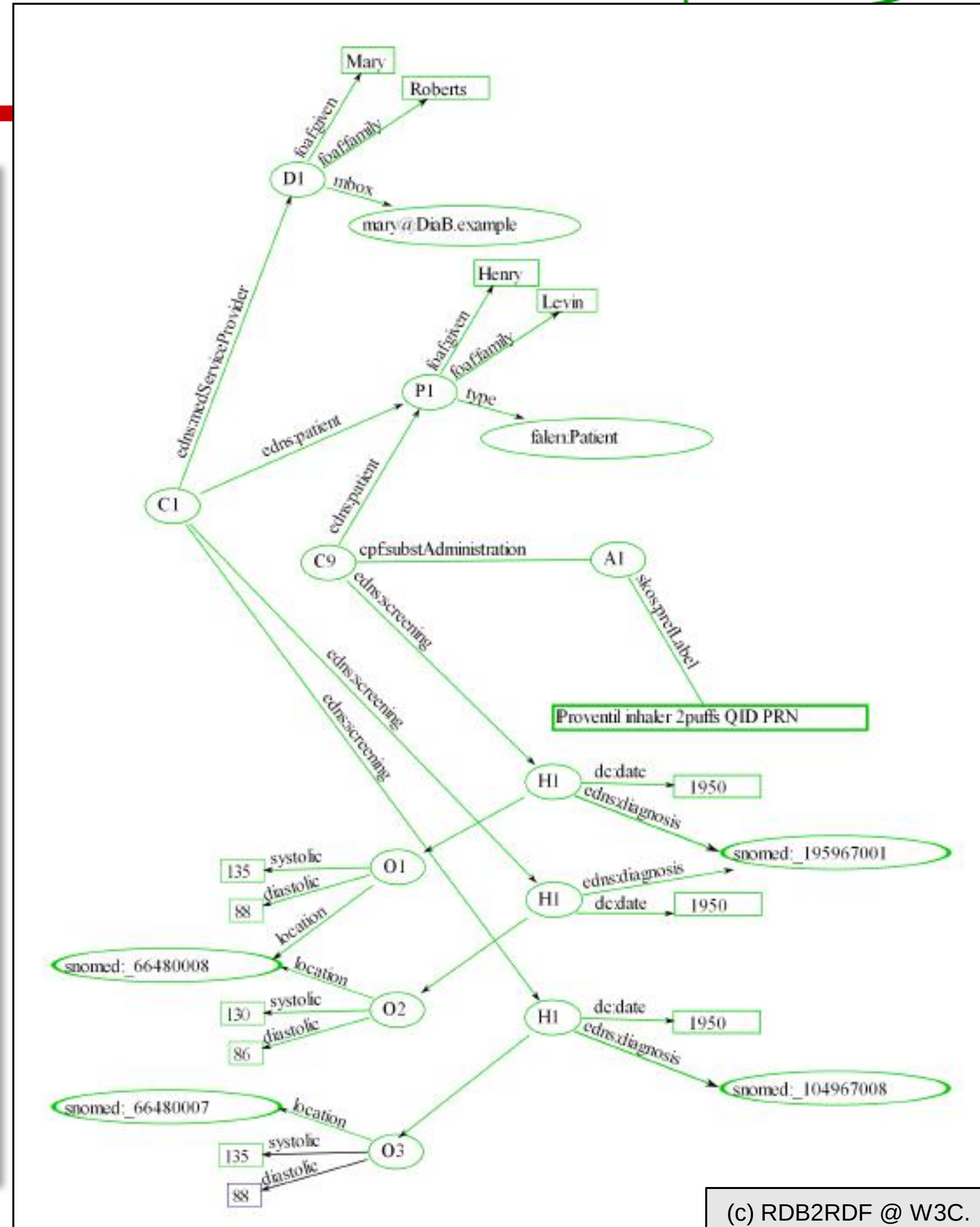


RDB2RDF (2)

- Table-to-class mapping approach
  - Each RDB table is a RDF class
  - Each RDB record is a RDF node
    - Each PK column value is a Subject URI
    - Each non-PK column name in a RDB table is a RDF predicate
    - Each RDB table cell value (non-PK) is a value (Object)

Mapping example





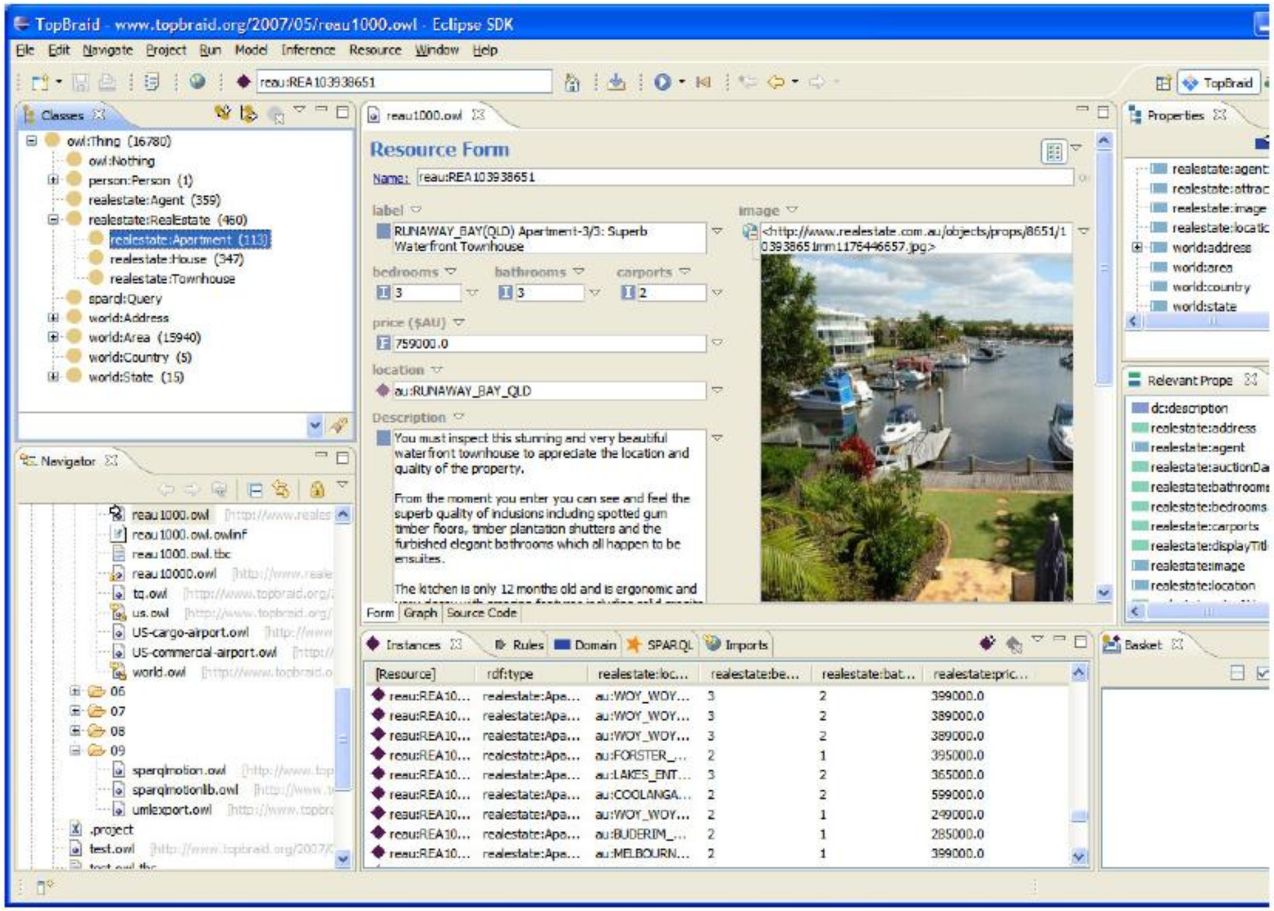
Other RDF tools

ontotext

GATE

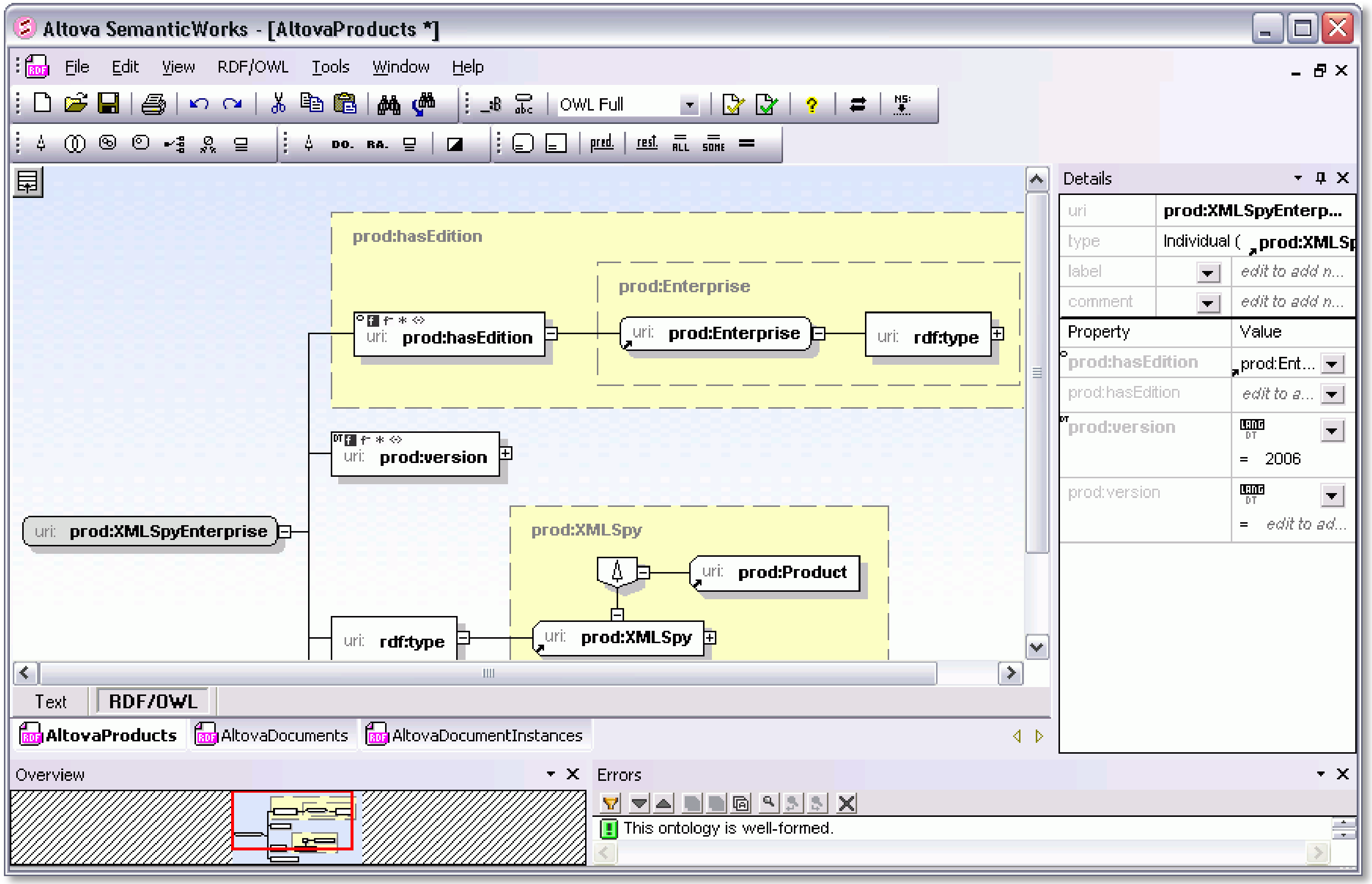
Ontology editors

- TopBraid Composer
  - http://www.topquadrant.com



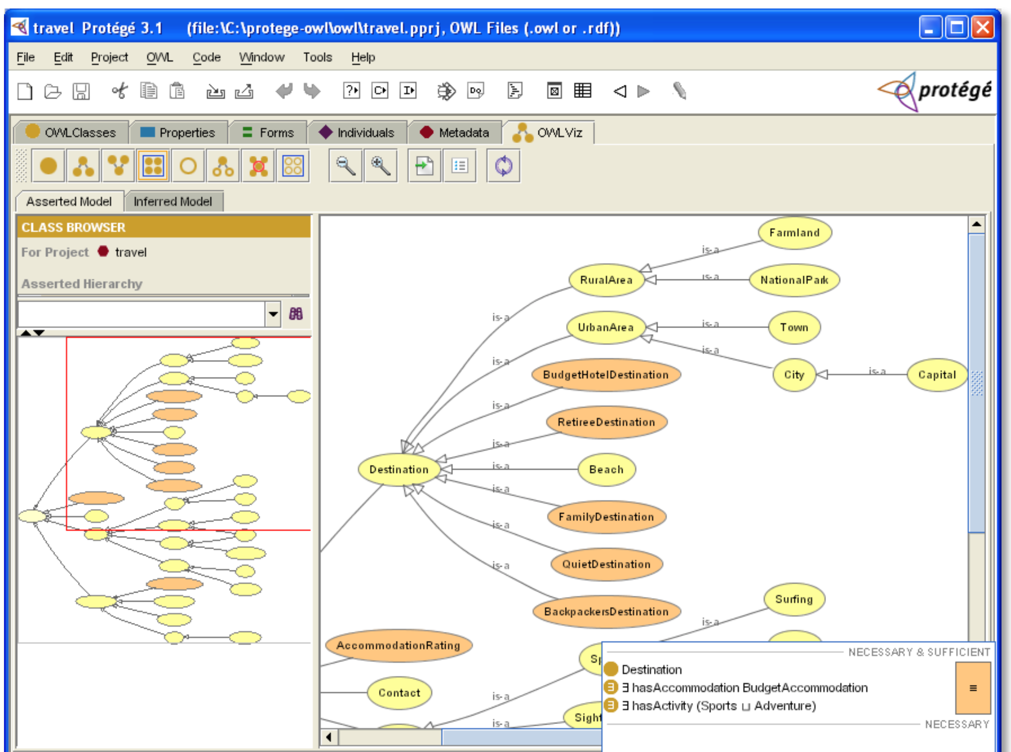
Ontology editors (2)

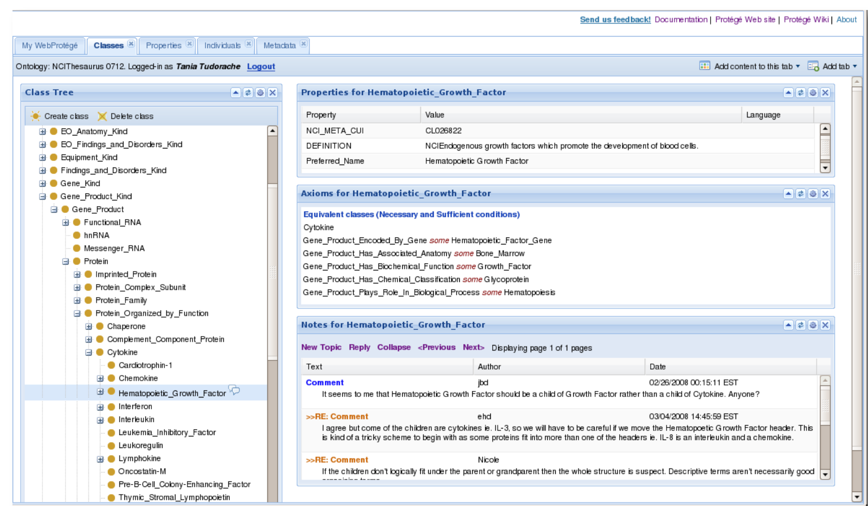
- Altova SemanticWorks
  - http://www.altova.com/semanticworks.html



Ontology editors (3)

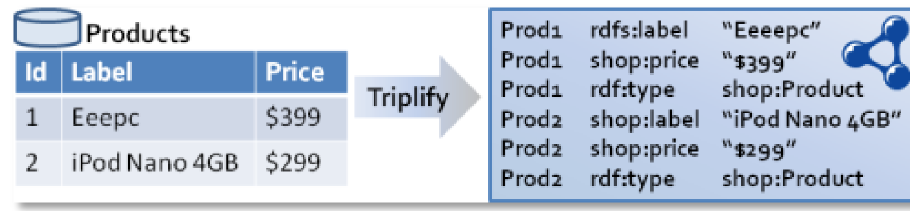
- Protégé
  - http://protege.stanford.edu
  - http://webprotege.stanford.edu



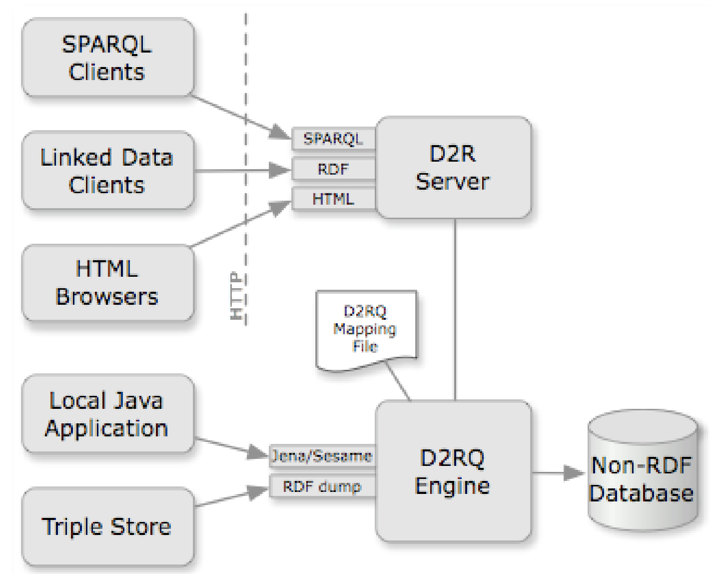


RDF-izers

- Triplify
  - http://triplify.org
  - Transform relational data into RDF / Linked Data



- D2RQ platform
  - http://www4.wiwiss.fu-berlin.de/bizer/d2rq/index.htm
  - D2RQ mapping language
  - D2RQ plugin for Sesame/Jena
  - D2R server
    - Linked Data & SPARQL endpoint



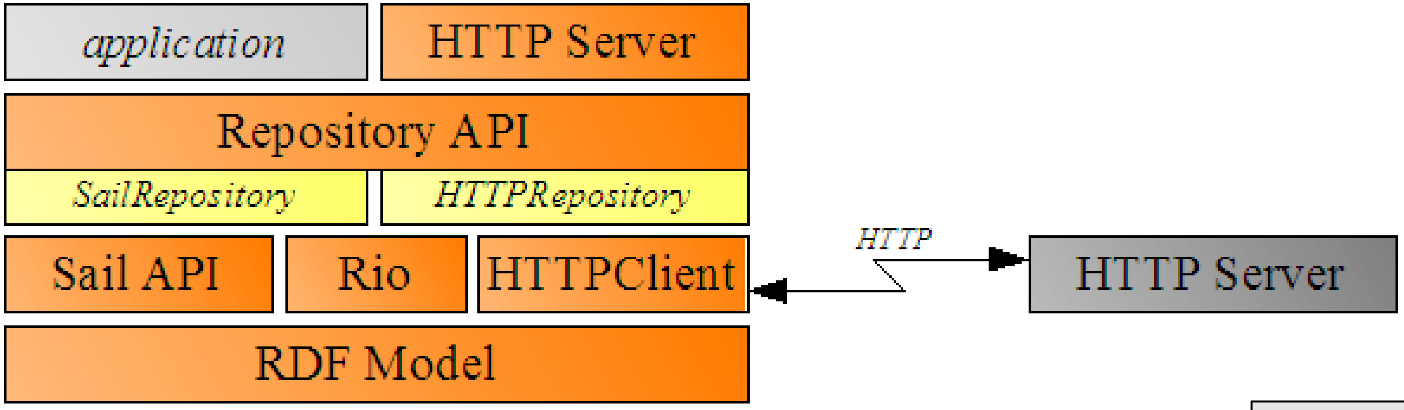
ontotext

RDF APIs

- Jena
  - http://jena.sourceforge.net
  - RDF/OWL API (Java)
  - In-memory or persistent storage
  - SPARQL query engine
- OpenRDF (Sesame)
  - http://www.openrdf.org
  - RDF API (Java), high performance parser
  - Persistent storage
  - SPARQL query engine

Sesame API

- Architecture
  - RDF Model
  - RDF I/O (parsers & serializars)
  - Storage & Inference Layer (for reasoners & databases)
  - High-level Repository API
  - HTTP & REST service frontend



(c) OpenRDF.org

Sesame API (2)

- RDF model
  - Create statements, get S/P/O
- SAIL API
  - Initialise, shut-down repositories
  - Add/remove/iterate statements
  - queries
- Repository API
  - Create a repository (in-memory, filesystem), connect
  - Add, retrieve, remove triples
  - Queries

Sesame API – REST interface

resource	GET	POST	PUT	DELETE
/repositories	List available repositories	-	-	-
/repositories/ID	query evaluation • query text • Bindings • Inference	Query evaluation	-	-
/repositories/ID/statements	Get statements in repository • S, P, O • inference	Adds/updates statements	Modifies existing statements	Deletes statements • S, P, O
/repositories/ID/namesapces	List Namespace definitions	-	-	Deletes all namespace definitions
/repositories/ID/namesapces/PREF	Gets the namespace for a prefix	-	Defines/updates the namespace for a prefix	Removes a namespace declaration

Sesame – OpenRDF Workbench