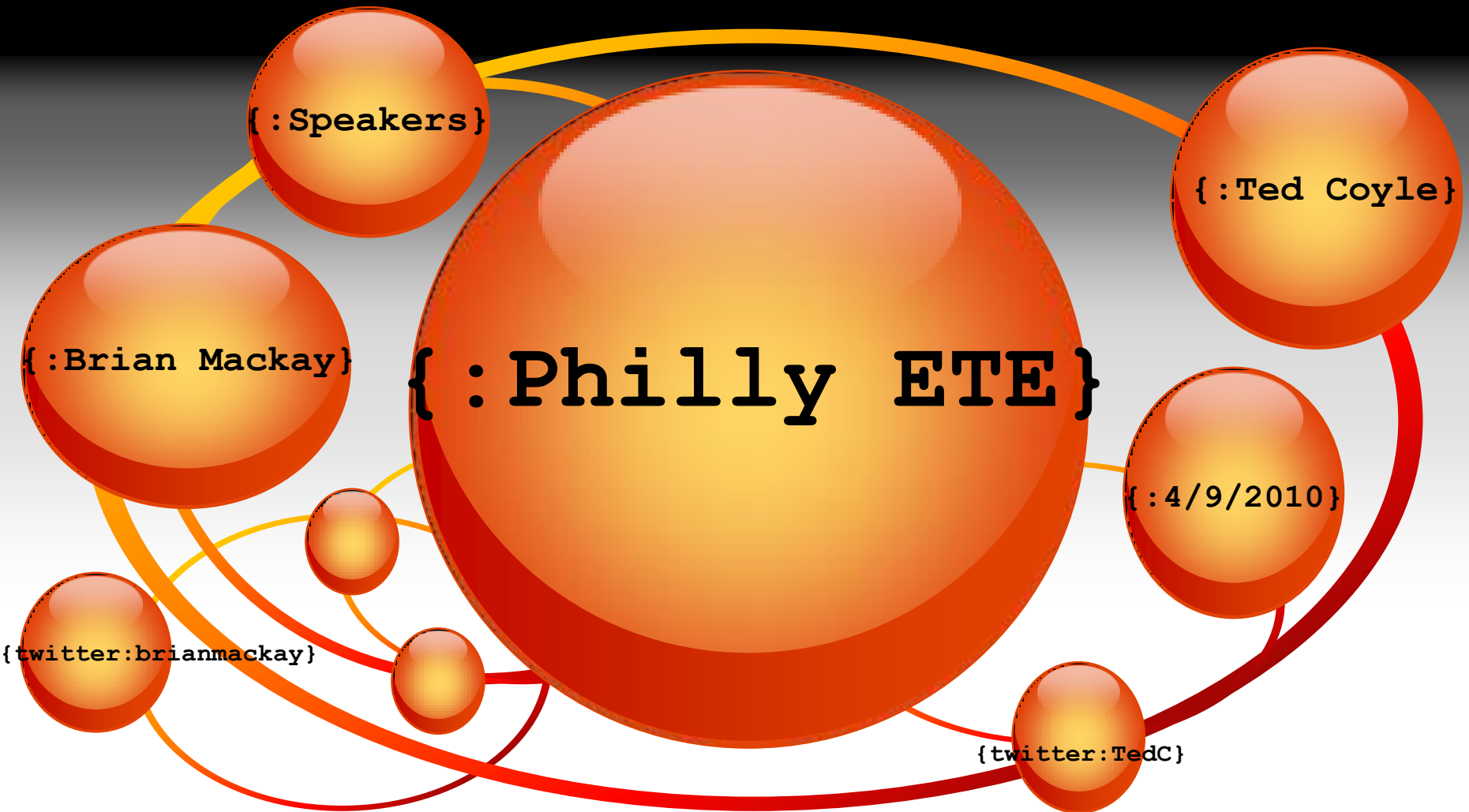


# Semantic Web

## What It Is and Why You Should Care



# Semantic Web

{:Overview:

{:Origins}

{:Reality}

{:Tools}

{:Modeling}

{:Inferencing}

{:Formats}

{:Summary:Q&A}

# Semantic Web

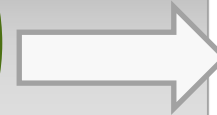
{ :Origins }



{ :1955 }



{ :1989 }



{ :now }

# Semantic Web

{ :Origins :Concepts }

{World:Closed:SQL}  
  {Data:Walled}



{World:Open:SPARQL}  
  {Data:Linked}



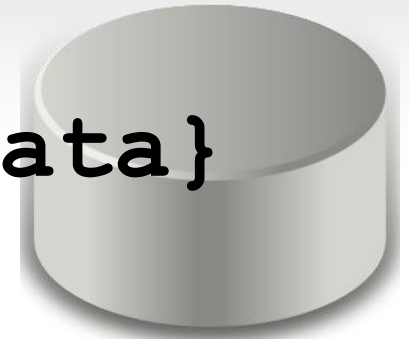
# Semantic Web

{ :Origins:Concepts }

{ :Information:WWW:Documents }



{ Information:SemWeb:Data }



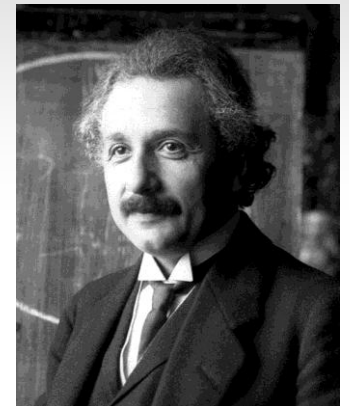
# Semantic Web

{ : Origins : Summary }

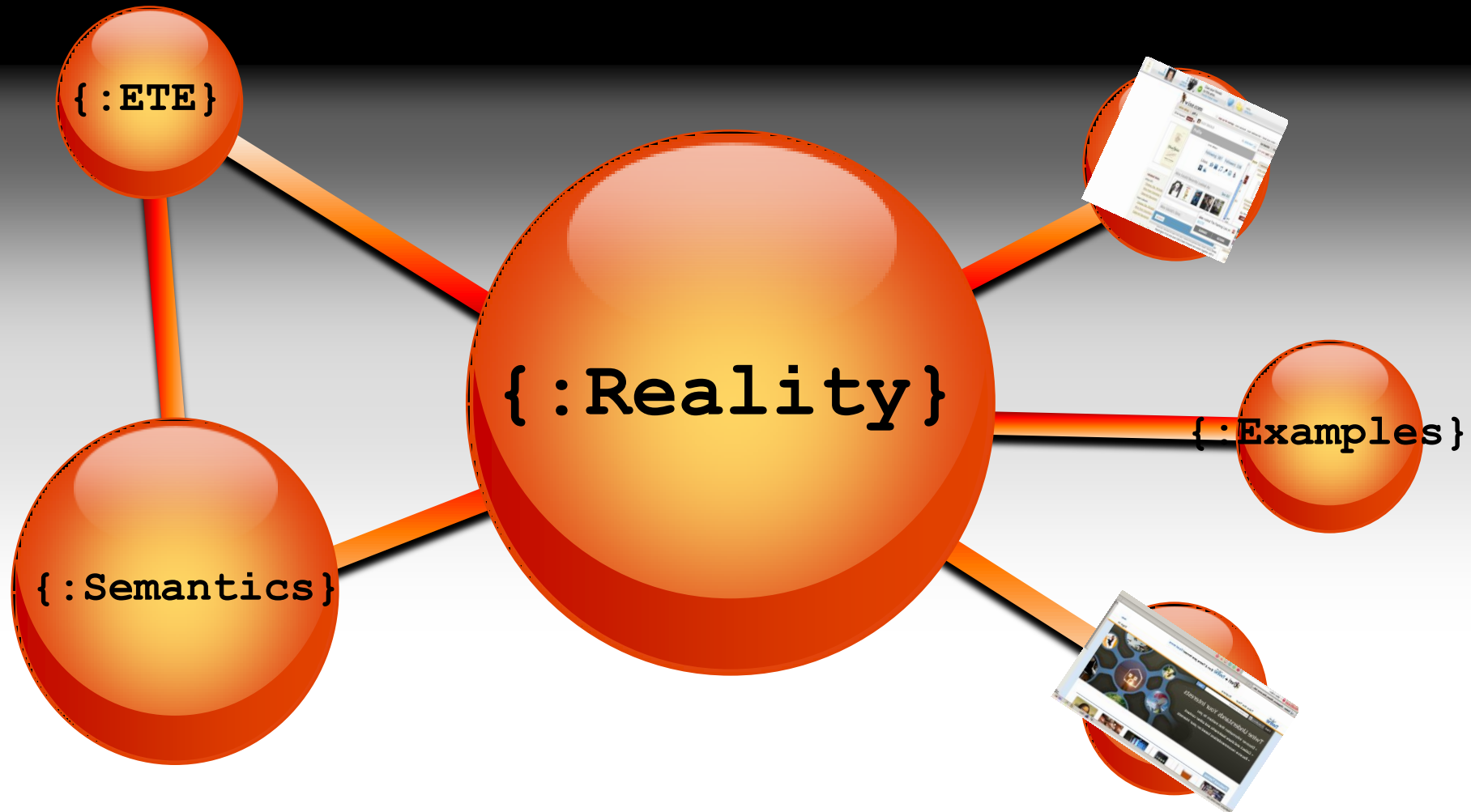
{ : www : dumbWeb : 1994 }



{ : SemWeb : smartWeb : now }



# Semantic Web



# Semantic Web

{ :GetGlue }

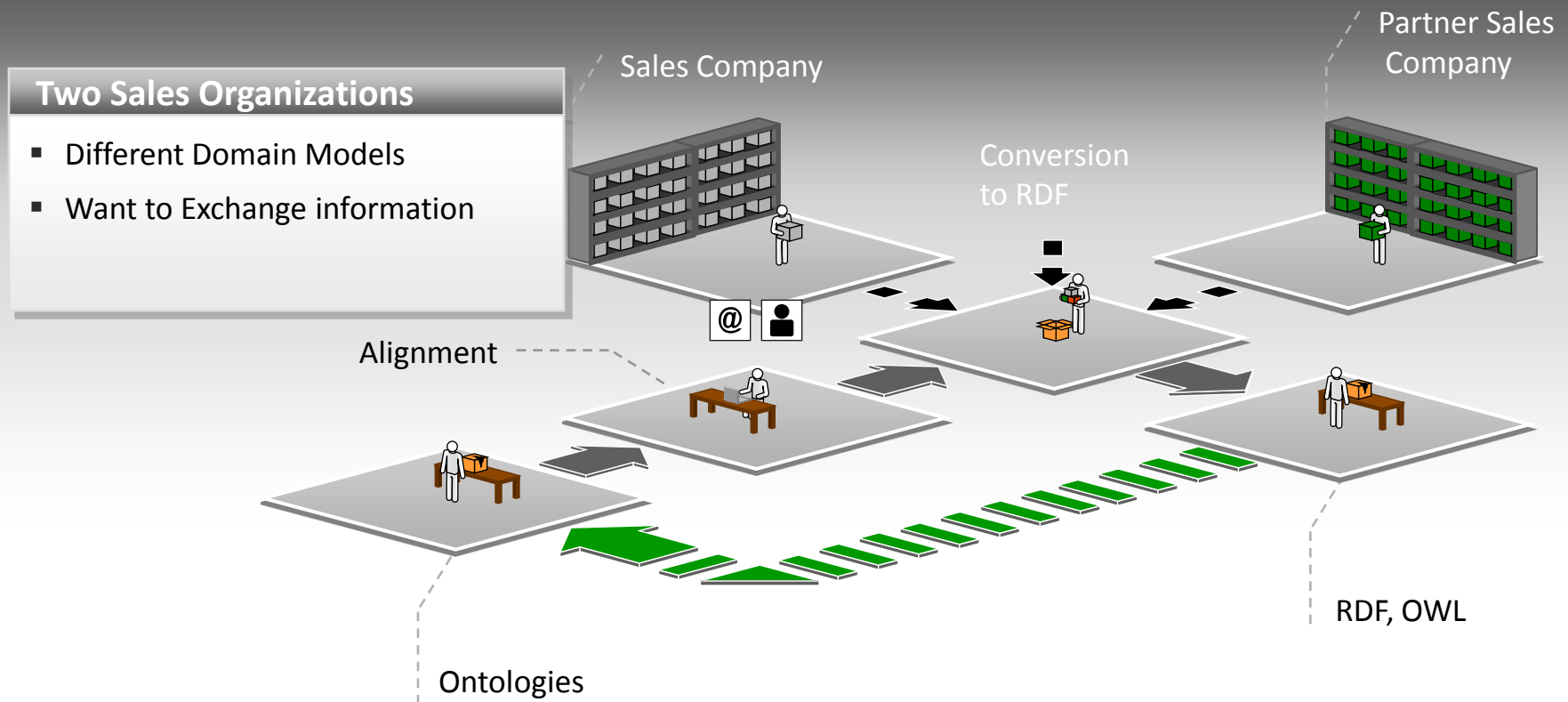


# Semantic Web

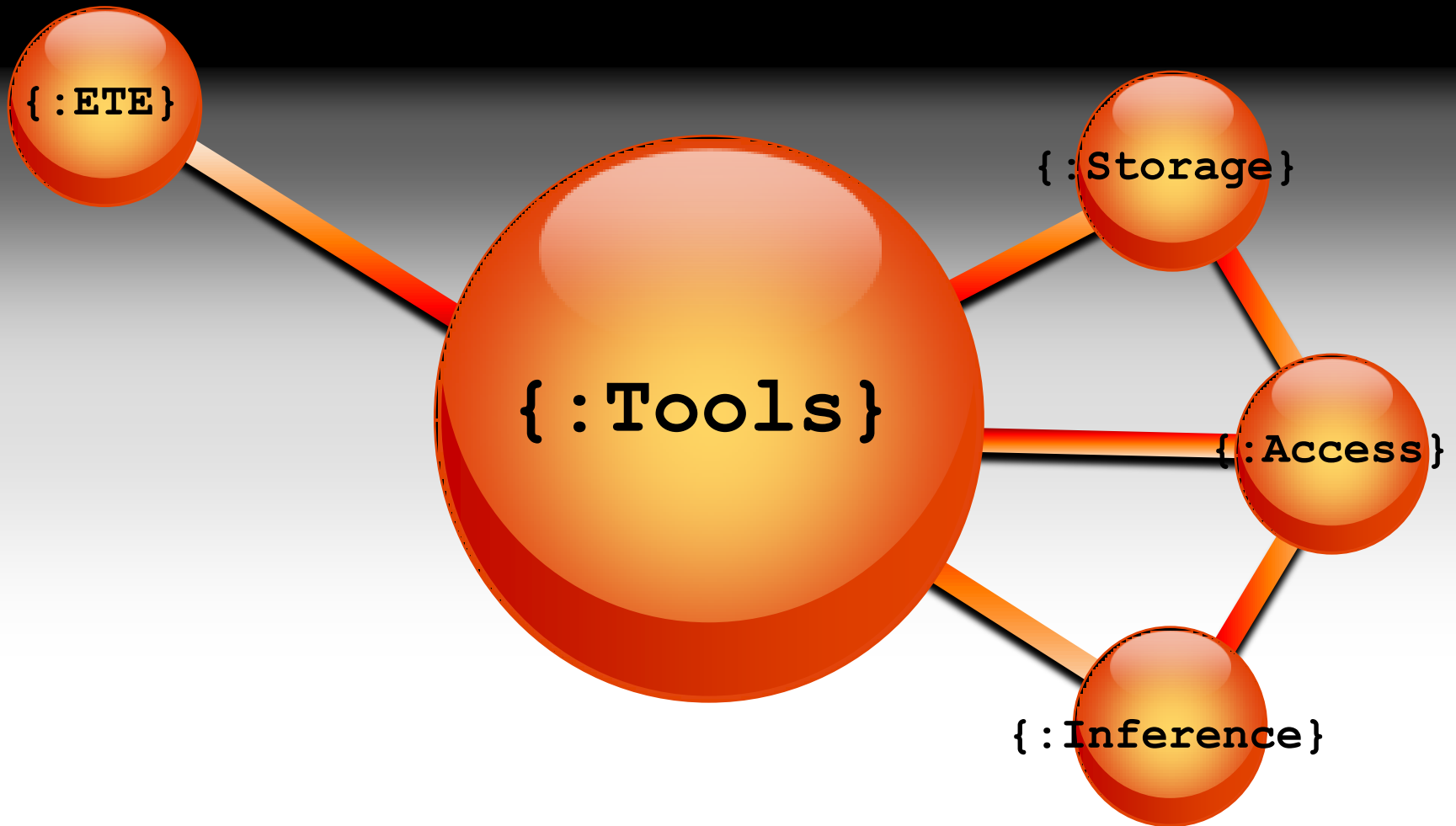
{ :Twine }



# Sample Application Overview



# Semantic Web



# SEMWEB TOOLS OVERVIEW

Examples using these tools will be based on a sales associate sample application.

<http://code.google.com/p/semwebcommons/source/browse/#svn/trunk/demos>

D2RQ

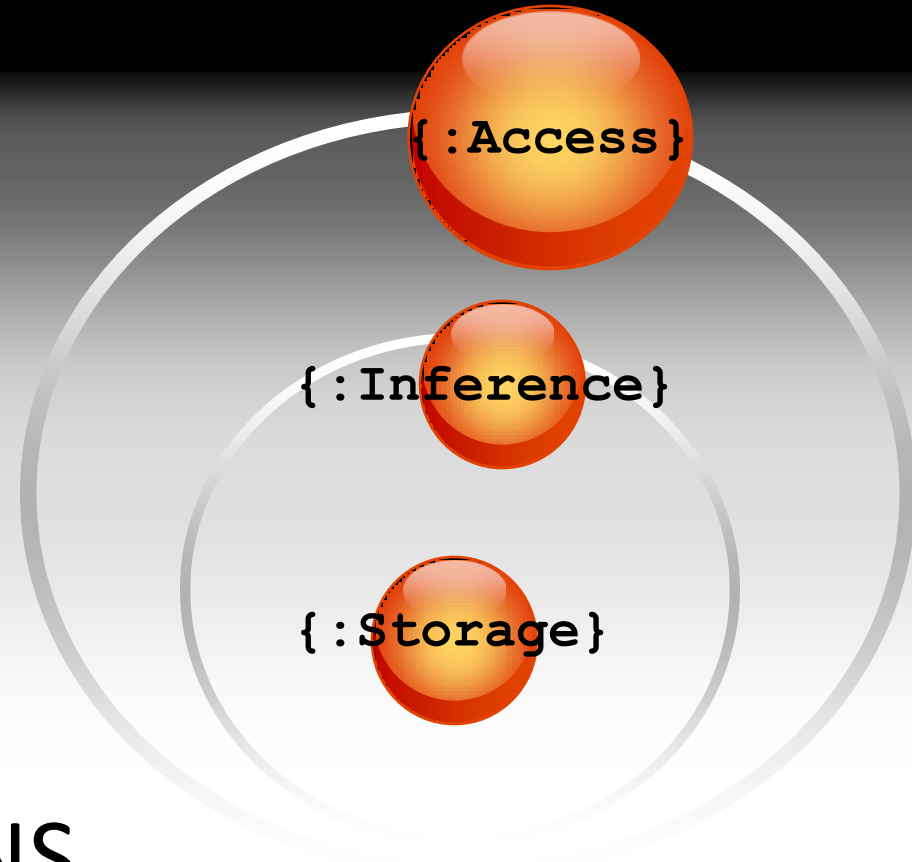
SPARQL

GLOZE

JENA REASONER

SPIN

SEMWEB COMMONS



# D2RQ Platform

Need applications to access databases without having to replicate into RDF ?

The D2RQ mapping enables non-RDF relational databases as virtual RDF graphs.

## **D2RQ Mapping Language**

Relations between an ontology and a relational data model.

## **D2RQ Engine**

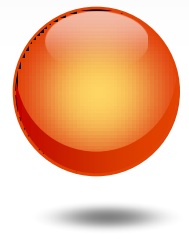
Uses mappings to convert SPARQL to SQL queries, returning RDF.

## **D2R Server**

HTTP server Linked Data view and a SPARQL endpoint.

## **Download**

<http://sourceforge.net/projects/d2rq-map/>



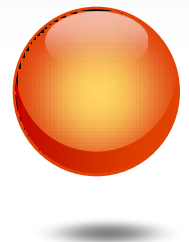
# D2RQ Setup

```
generate-mapping -o SalesDemoMap.n3 -d com.mysql.jdbc.Driver  
jdbc:mysql://127.0.0.1/salesdemo
```

```
dump-rdf -m salesdemo.n3 -f N3 -o salesdemoExample.n3 -b http://salesdemo/
```

**Mapping Spec can be found here**

<http://www4.wiwiiss.fu-berlin.de/bizer/d2rq/spec/index.htm>




# d2rq:ClassMaps & d2rq:PropertyBridge

d2rq:ClassMap defines a SQL table

d2rq:PropertyBridge defines a column in the table



```
[] a  
d2rq:ClassMap;
```



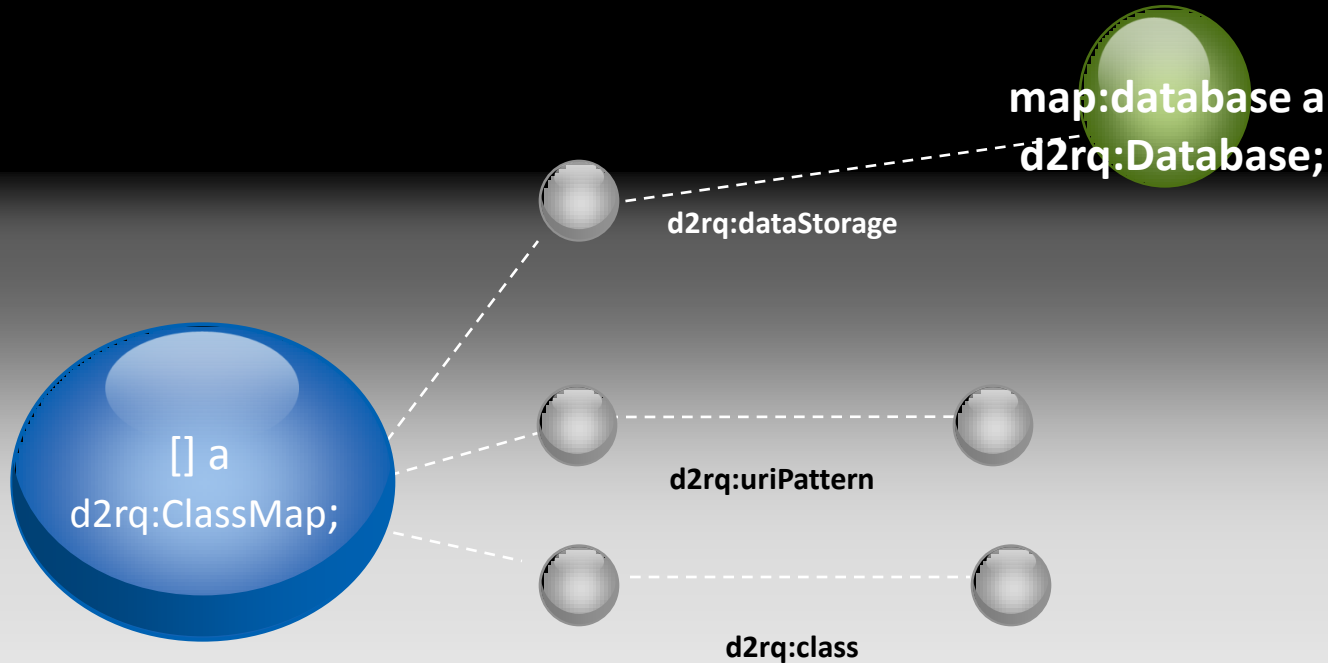
```
map:address_CITY a  
d2rq:PropertyBridge
```

**Mapping Spec can be found here**

<http://www4.wiwiss.fu-berlin.de/bizer/d2rq/spec/index.htm>

# d2rq:ClassMaps

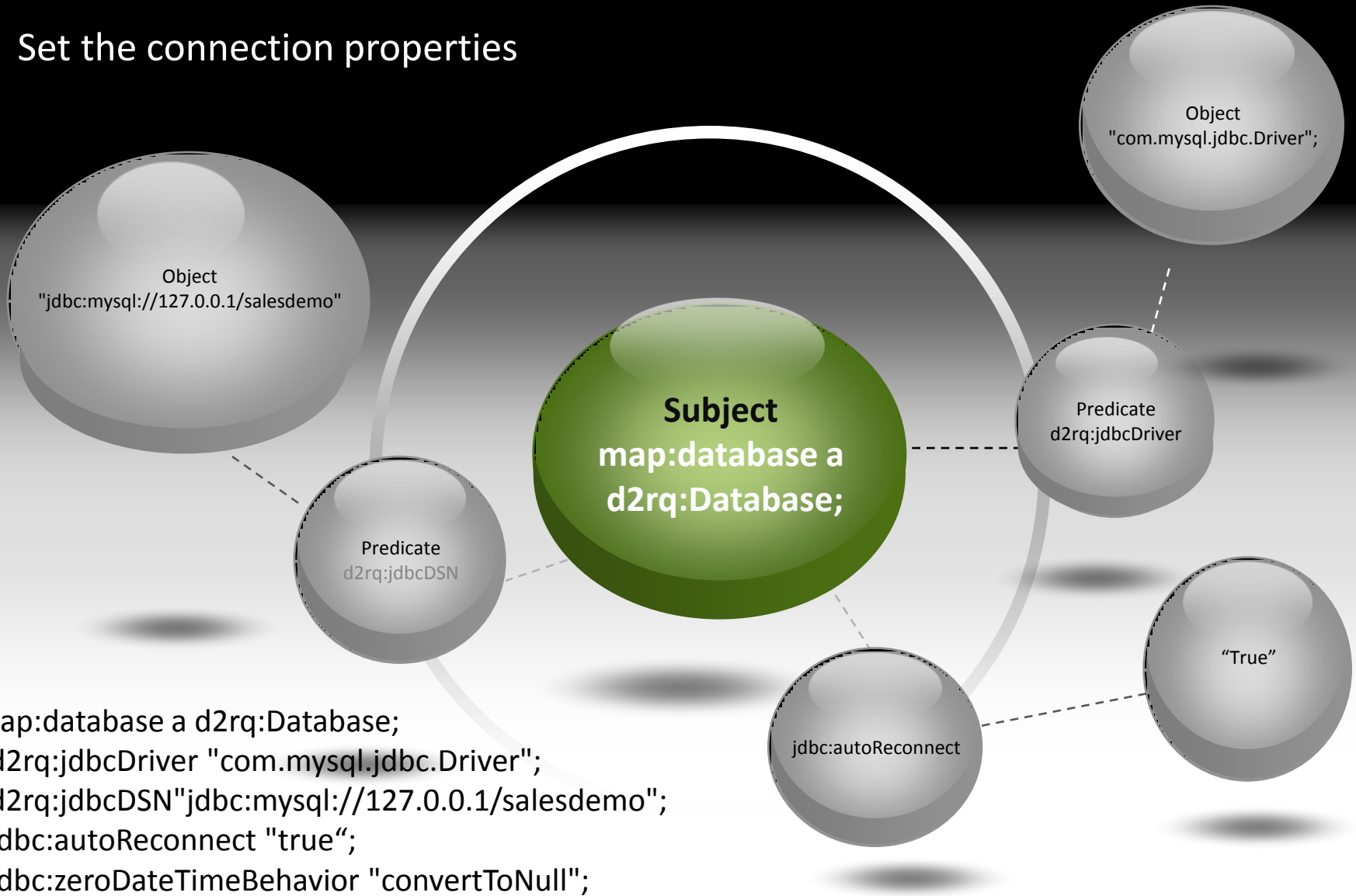
d2rq:ClassMap: group of similar classes of an OWL ontology



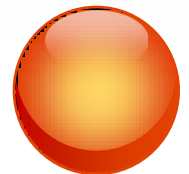
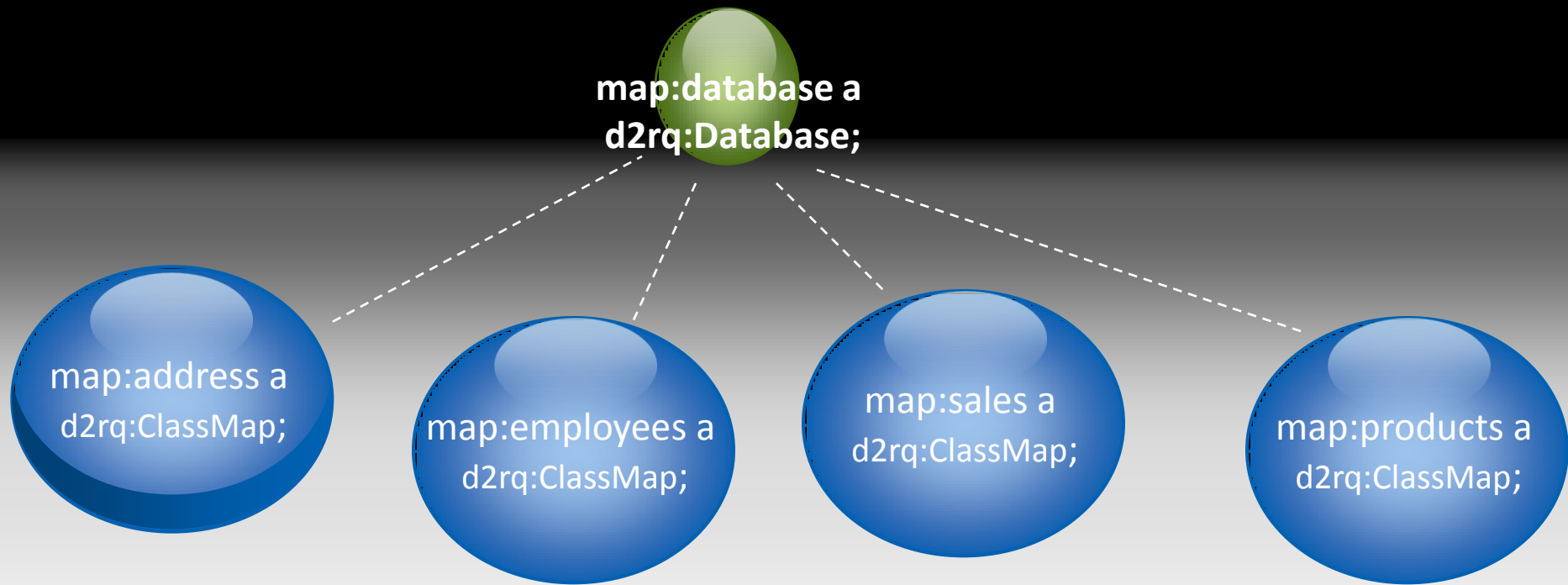
```
map:address a d2rq:ClassMap;  
  d2rq:dataStorage map:database;
```

# D2RQ Database Resource

Set the connection properties

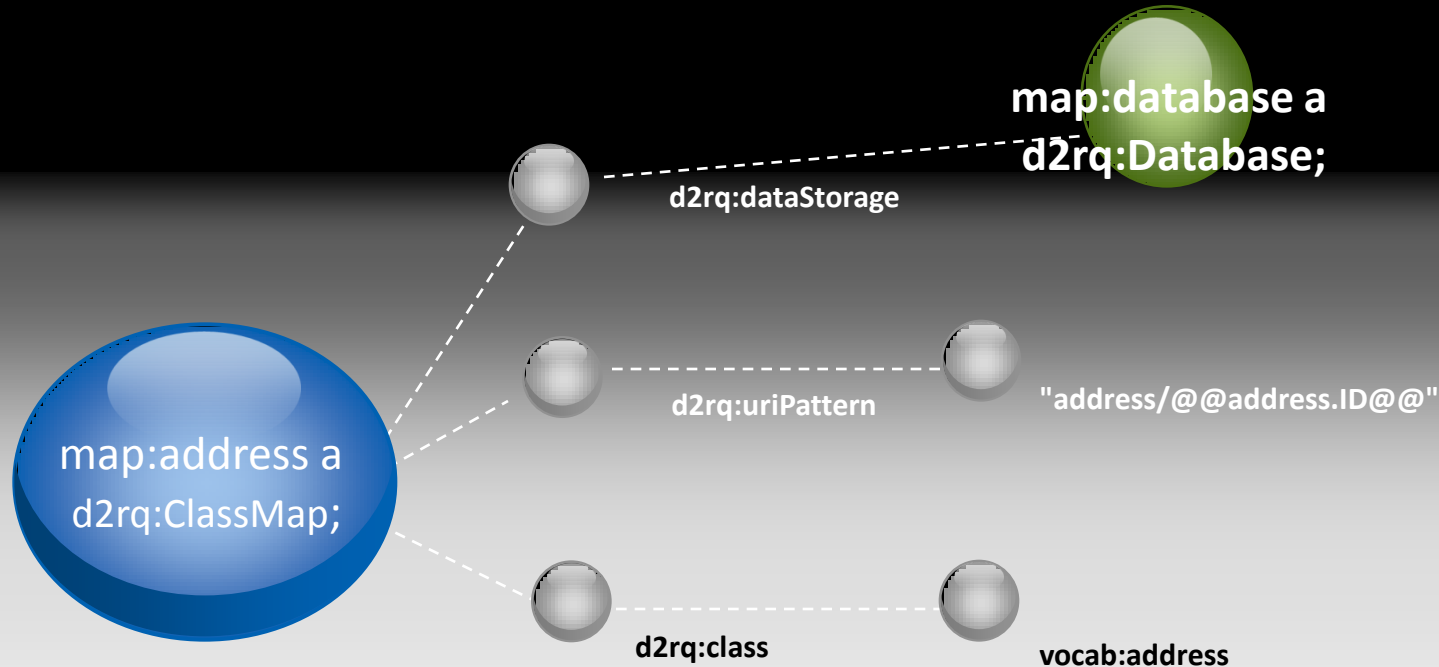


# D2RQ Sales Associates Example

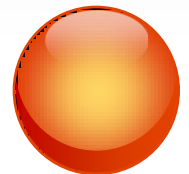


# map:address

d2rq:ClassMap: group of similar classes of an OWL ontology

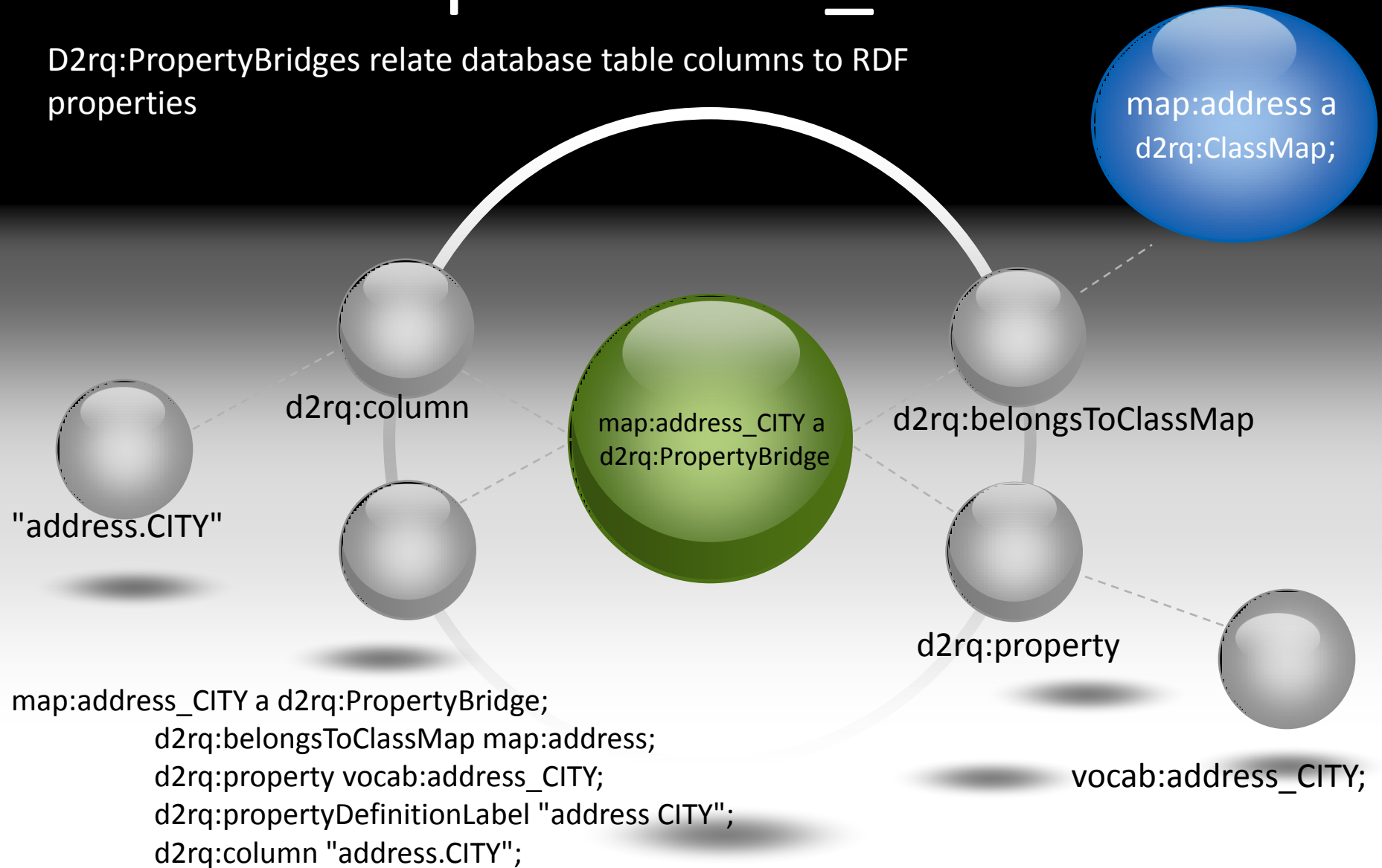


```
map:address a d2rq:ClassMap;  
  d2rq:dataStorage map:database;  
  d2rq:uriPattern  
  "address/@@address.ID@";  
  d2rq:class vocab:address;  
  d2rq:classDefinitionLabel "address";
```



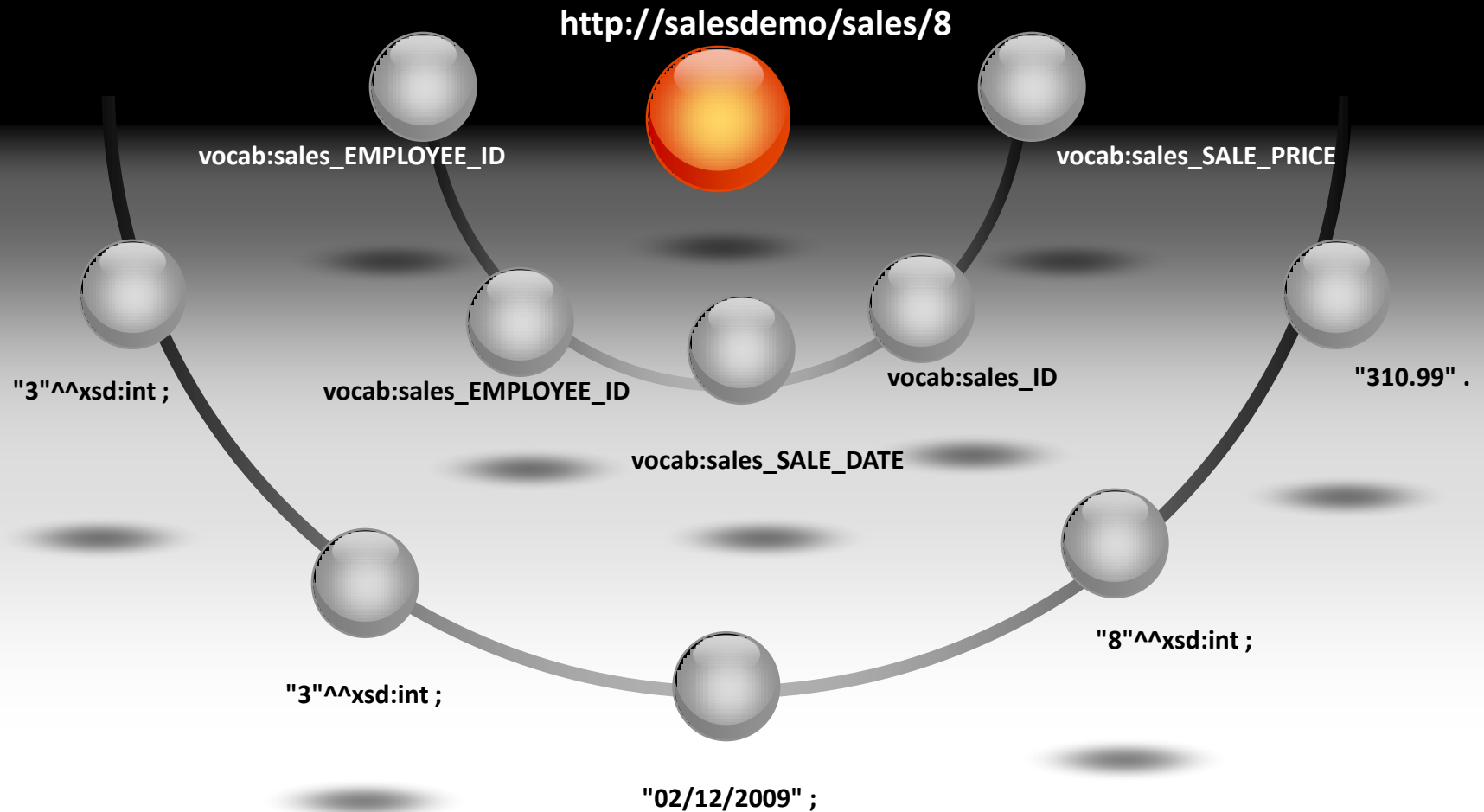
# map:address\_CITY

D2rq:PropertyBridges relate database table columns to RDF properties



# Sales ID #8

Subject linked to sales.id table

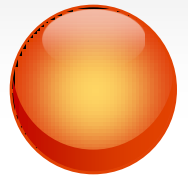


# Sparql to Filter by Address

Not bound by primary keys like SQL, common mistake when starting out.

Any variable can be bound to any other variable.

```
SELECT DISTINCT
  ?id ?streetAddress ?zip ?stateCD
WHERE {
  ?s vocab:address_STATE_CD ?stateCD.
  ?s vocab:address_ID ?id.
  ?s vocab:address_ADDRESS ?streetAddress.
  ?s vocab:address_ZIP ?zip.
  FILTER (?stateCD = str(?code)).
}
```



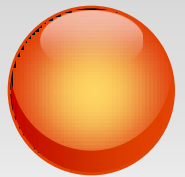
# Using Triples Instead of SPARQL

Sometimes SPARQL is not needed if data is not distributed  
Common pitfall when normal graph query is sufficient

## Example

```
RDFNode node = ModelFactory.createDefaultModel().createResource("PA");  
querySolutionMap.add("code", node);  
result = Factory.createFilteredModel (model, sparqlPath, querySolutionMap);
```

Put parameters in QuerySolutionMap. Can be returned as a Model or XML



## Example

```
Node predicate = Node.createURI("http://localhost:2020/address_STATE_CD");  
Node object = Node.createLiteral("PA");  
Factory.createFilteredModel(m, Triple.createMatch( Node.ANY, pred, object));
```

# GLOZE: XML to RDF and Back Again

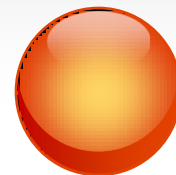
Inspects XSD schema

Lifts XML up to RDF, Drops RDF down to XML

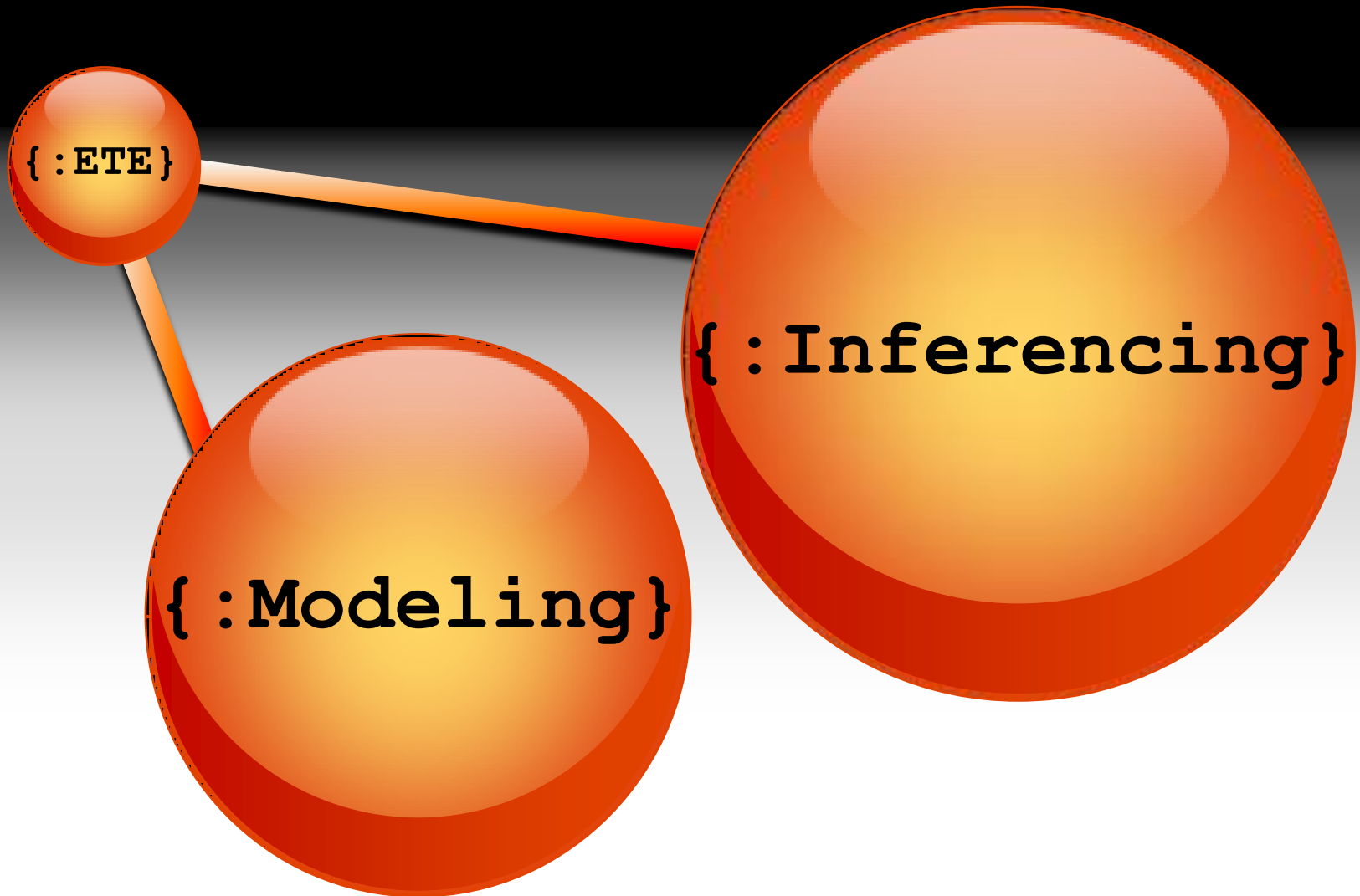
Our Client wants to send us address info in this format, use Gloze to lift

```
<xmlns:address="http://example.com/vocab/#" >  
  <xs:element name="address">  
    <xs:complexType>  
      <xs:sequence>  
        <xs:element name="street"/>  
        <xs:element name="postal"/>  
        <xs:element name="county"/>  
        <xs:element name="state"/>  
        <xs:element name="city"/>  
      </xs:sequence>  
    </xs:complexType>  
  </xs:element>  
</xs:schema>
```

```
address:  
  address:address  
    [ address:city "Malvern" ;  
      address:county "West Chester" ;  
      address:postal "19356" ;  
      address:state "PA" ;  
      address:street "456 Lancaster Drive"  
    ] .
```



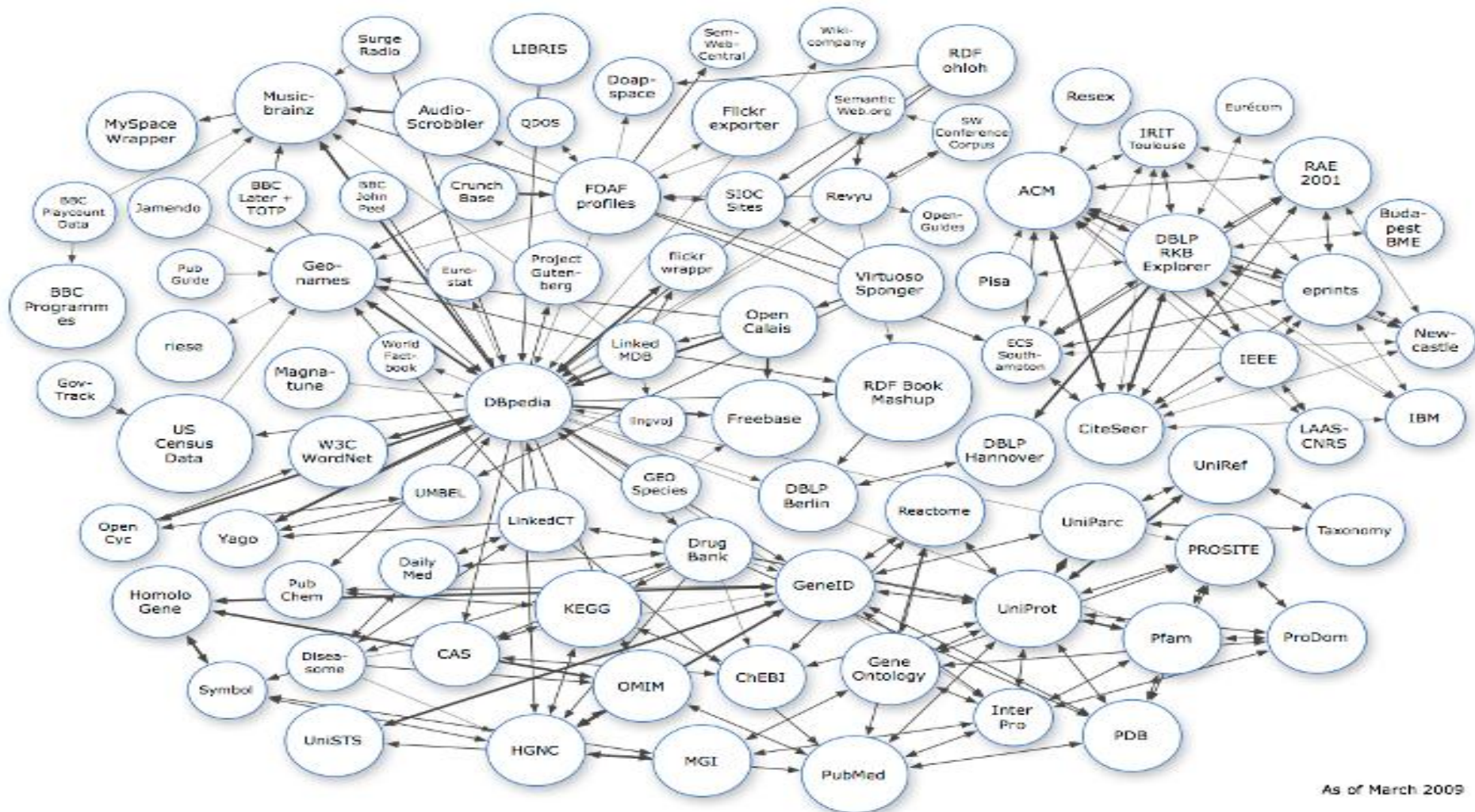
# Semantic Web





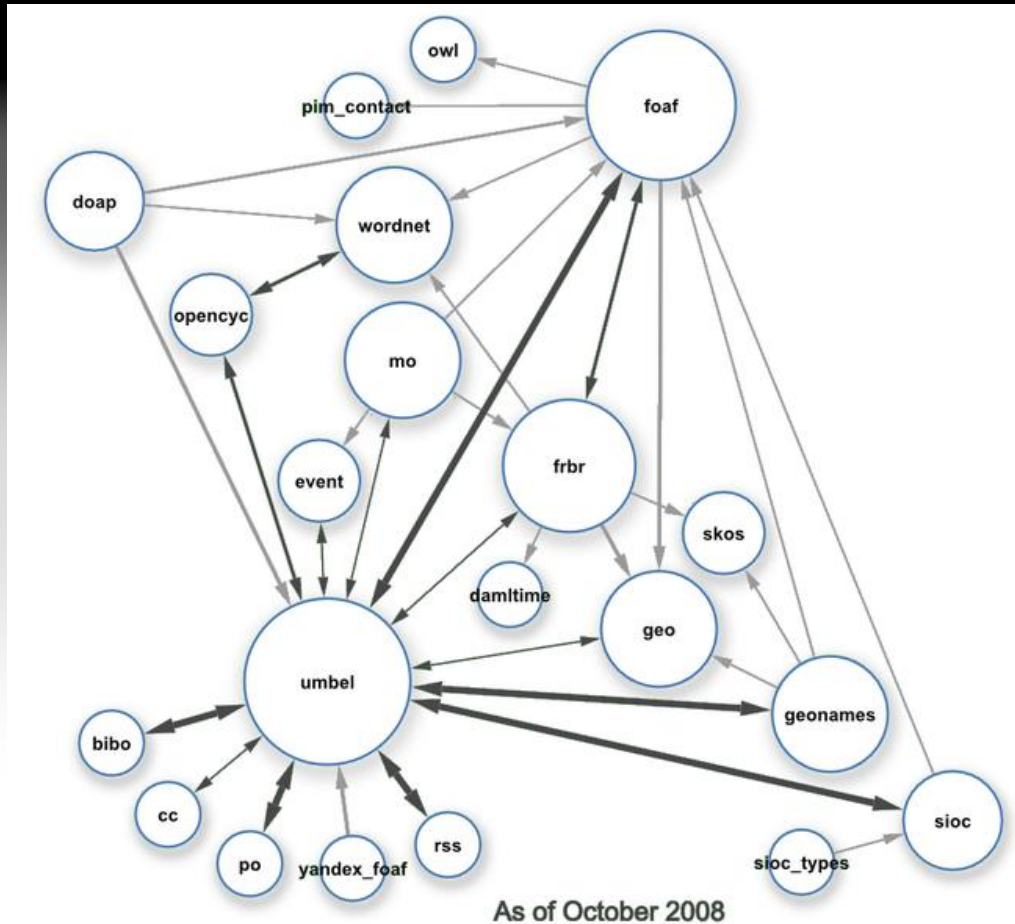
# Semantic Web

{ :LinkedData:Now }



# Semantic Web

{ :LinkingOpenData:DBpedia }



# Semantic Web

```
{ :DataMapping: }  
  { :Ontologies }
```

```
    { :RepresentedKnowledge }  
      { :DomainConcepts }  
        { :Relationships }  
          { :Reason }  
            { :DomainDescriptive }  
              { :SharedVocabulary }  
                { :Class:SubClass }
```

# Semantic Web

{ :GettingAtData: }

{ :inferencing:Backwards }

{ :inferencing:Forwards }

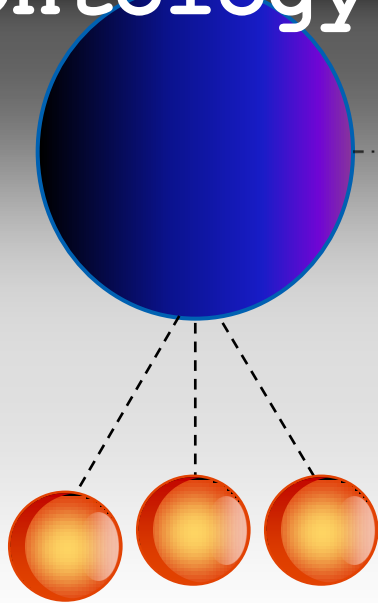
“Inferencing is the glue that holds the Semantic Web together”



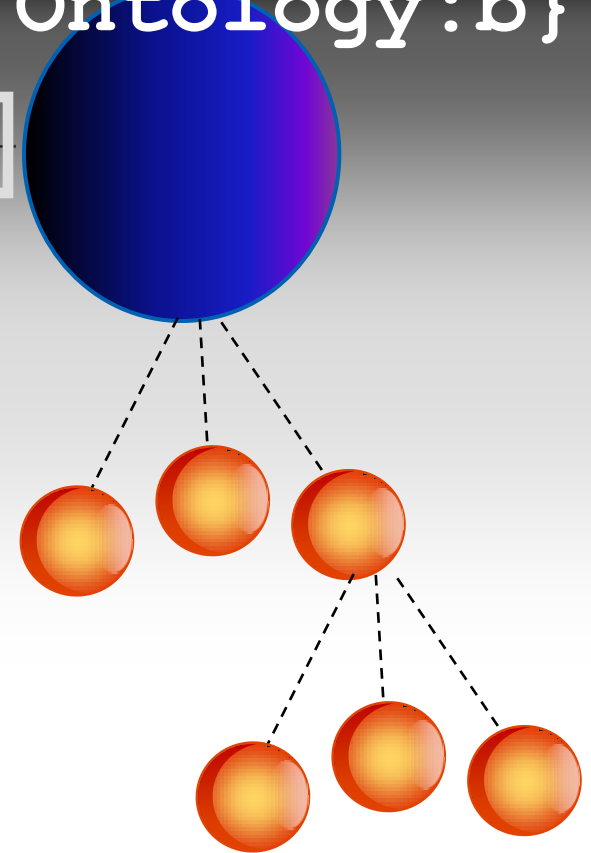
# Semantic Web

{ :Inference:lite }

{ :Ontology:a }



{ :Ontology:b }



{ :inference }

# Sales Reasoning

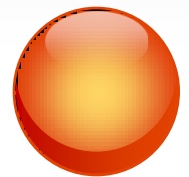
Written in a SPARQL like format

Has built ins to help with most operations

if sales gt 200 lt 400 then a sample:GoodSalesAssociate

```
[hasGoodSales: (?x rdf:type vocab:sales)
  (?x vocab:sales_SALE_PRICE ?p)
  greaterThan(?p,200)
  lessThan(?p,400)
  (?x vocab:sales_EMPLOYEE_ID ?e)
  uriConcat(?x,'/', ?p, '/GoodSales/', ?a1)
  -> (?a1 rdf:type sample:GoodSalesAssociate )
  (?a1, vocab:sales_EMPLOYEE_ID, ?e) ]
```

```
<http://salesdemo/sales/8/310/GoodSales/>
  a <http://localhost:2020/sample/GoodSalesAssociate> ;
  <http://localhost:2020/vocab/resource/sales_EMPLOYEE_ID>
    "3"^^<http://www.w3.org/2001/XMLSchema#int> .
```



# Spin RDF (SPARQL Inferencing Model)

RDF vocabularies enabling the use of SPARQL to define constraints and inference rules on Semantic Web models.

SPIN also provides meta-modeling capabilities that allow users to define their own SPARQL functions and query templates.

```
[[] a sp:Select ;  
  sp:resultVariables ( _:b1 ;  
    sp:where ( [ sp:object ex:Person ;  
      sp:predicate rdf:type ;  
      sp:subject _:b1  
    ] [ sp:object _:b2 ;  
      sp:predicate ex:age ;  
      sp:subject _:b1  
    ] [ a sp:Filter ;  
      sp:expression  
        [ a sp:gt ;  
          sp:arg1 _:b2 ;  
          sp:arg2 18 ] ] ) .
```

\_:b1 sp:varName "person"^^xsd:string .

\_:b2 sp:varName "age"^^ "^^xsd:string .

# Terms You'll See

RDF, RDFS

OWL LITE, OWL DL, OWL FULL, OWL-S,  
OWL-FAST

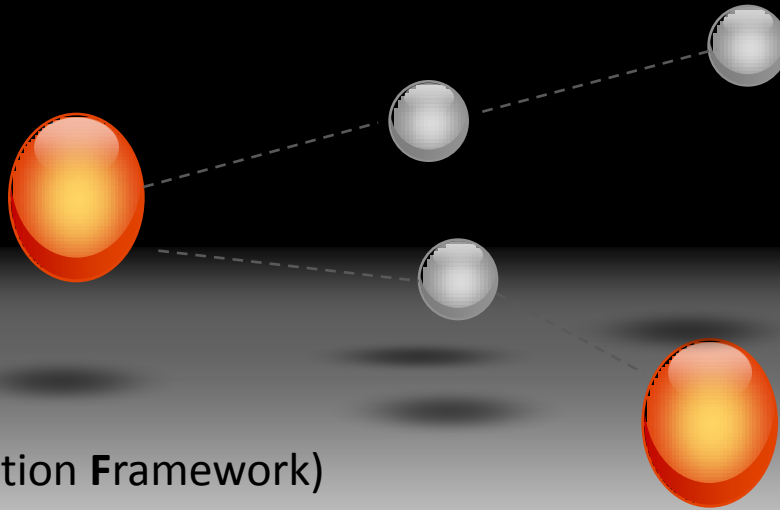
What is the difference? What should I use?

## What You'll Really Do

*OWL Fast: based primarily on RDF, but with some OWL lite*

*A mix of RDF, RDFS, OWL, OWL-S*

# RDF: What is It?



(Resource Description Framework)

Creating meta-data structures that define data.  
Designed to provide a method for classification.

Structured in triple format:

The subject (what the data is about)

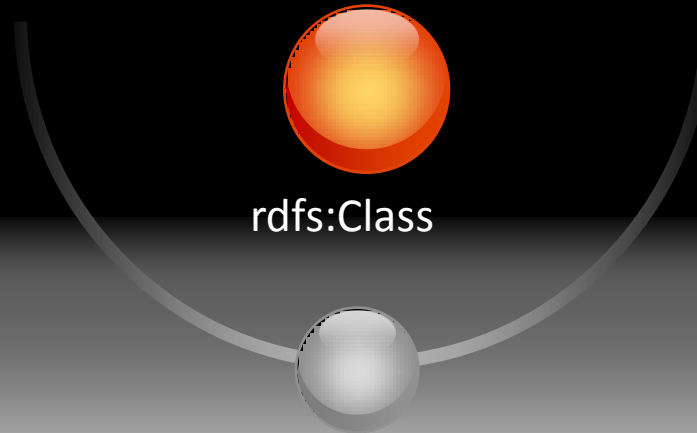
The predicate or property (an attribute of the subject)

The object (the actual value or sometimes another subject)

Each node can be either a URI or Literal. Subject and predicate are usually a URI.

# RDFS, What Do I Need to Know

RDFS (RDF Schema)



**rdfs:Class**

rdfs:subClassOf

linked to the class using the rdf:type predicate

*:Brian rdf:type foaf:Person*

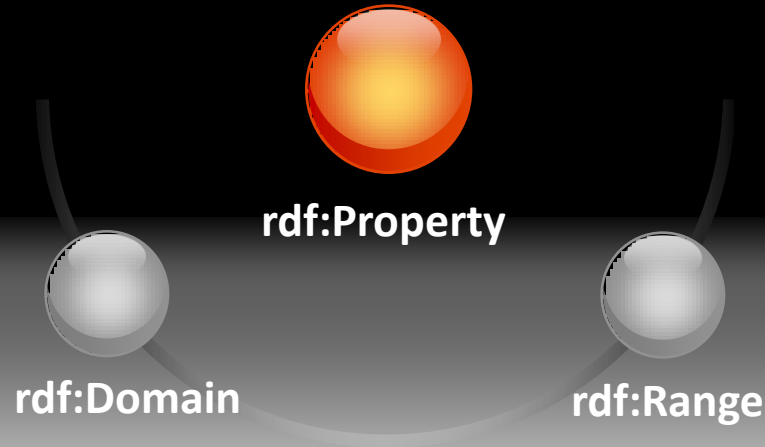
**rdfs:subClassOf**

*foaf:Person rdfs:subClassOf foaf:Agent*

Supports inheritance of a property domain and range

# RDFS, What Do I Need to Know

## RDFS (RDF Schema) Properties



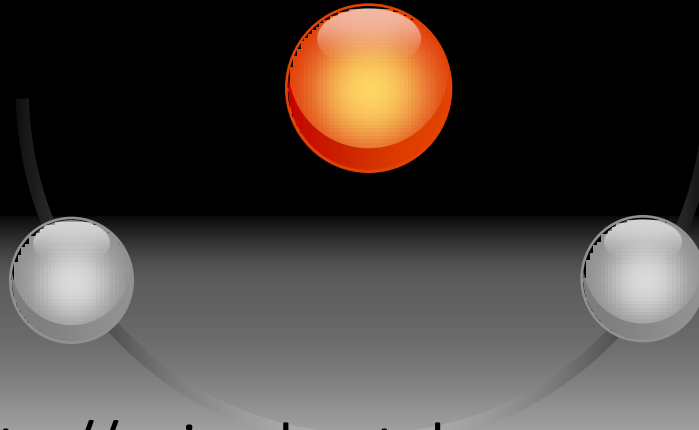
**rdf:Property** *predicate*

**rdfs:domain** declares the class of the *subject*

**rdfs:range** declares the class or datatype of the *object*

# RDFS, What Do I Need to Know

## RDFS (RDF Schema) Example



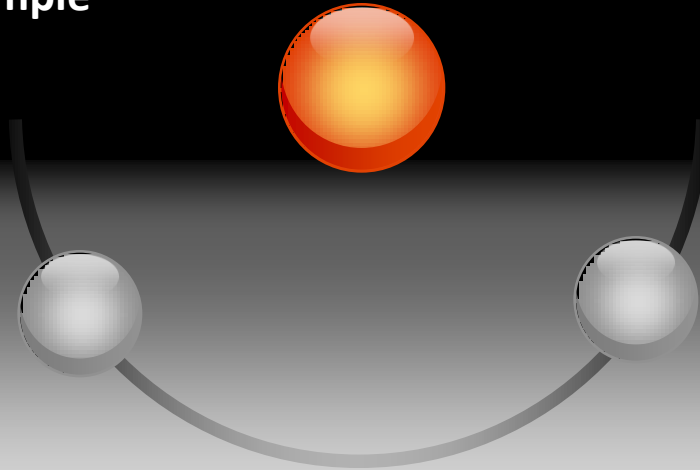
```
@prefix ao: <http://animal.ontology.example.com/#> .
```

```
ao:Duck a rdf:Class;  
  rdfs:subClassOf :Bird;.br/>  rdfs:domain      :animal.
```

```
ao:hasFeathers a rdf:Property;  
  rdfs:range    xsd:boolean.
```

# RDFS, What Do I Need to Know

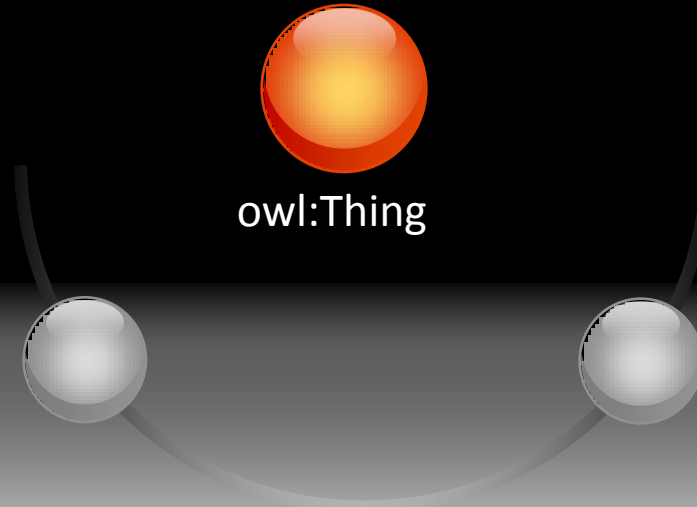
## RDF Animal Ontology Example



```
@prefix ao:    <http://animal.ontology.example.com/#> .
```

```
:Mallard a ao:Duck;  
  :hasFeathers xsd:true.
```

# OWL, What Do I Need to Know



Has three parts,  
OWL individuals, classes, and properties  
Every individual is an owl:Thing

## **Individuals are members of a class**

Duck a owl:Class;  
rdfs:type :Mallard.

rdf:type ties an individual to a class of which it is a member.

# OWL, What Do I Need to Know

## Properties

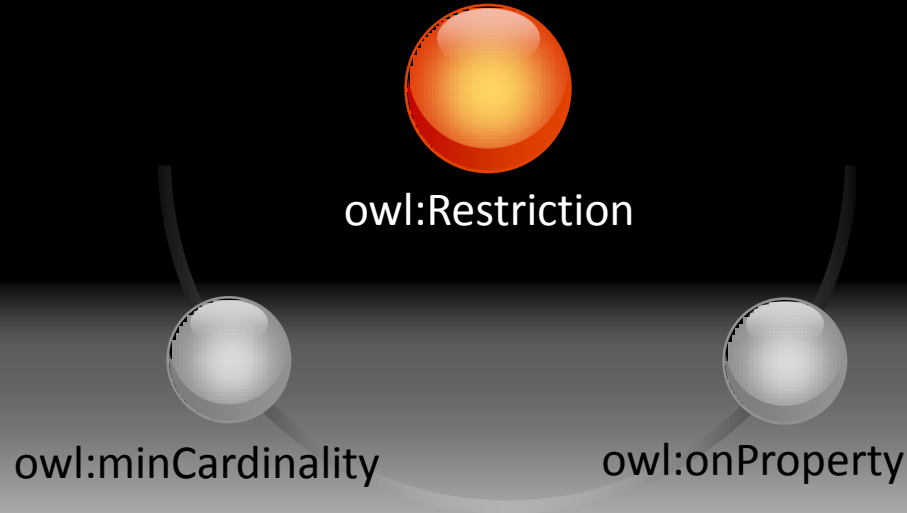


```
[] a owl:ObjectProperty ;  
   rdfs:domain :MyOtherClass.
```

```
[] a owl:DatatypeProperty ;  
   rdfs:domain :MyOtherClass;  
   rdfs:range xsd:string .
```

# OWL, What Do I Need to Know

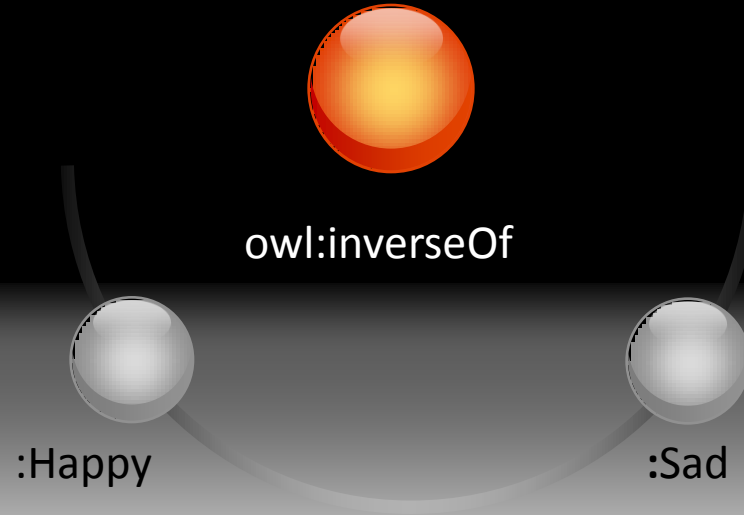
## Restrictions



```
([ a owl:Restriction ;  
    owl:minCardinality "1"^^xsd:nonNegativeInteger ;  
    owl:onProperty :someProperty  
])
```

# OWL, What Do I Need to Know

## Inverse relationships



```
:Happy a owl:Class ;  
  owl:inverseOf  :Sad.
```

```
:Sad a owl:Class ;  
  owl:inverseOf  :Happy.
```

# Semantic Web

{ :NP-complete }

NP-complete problems have no fast solution

“No fast”, meaning calculation time can very quickly grow to trillions of years....

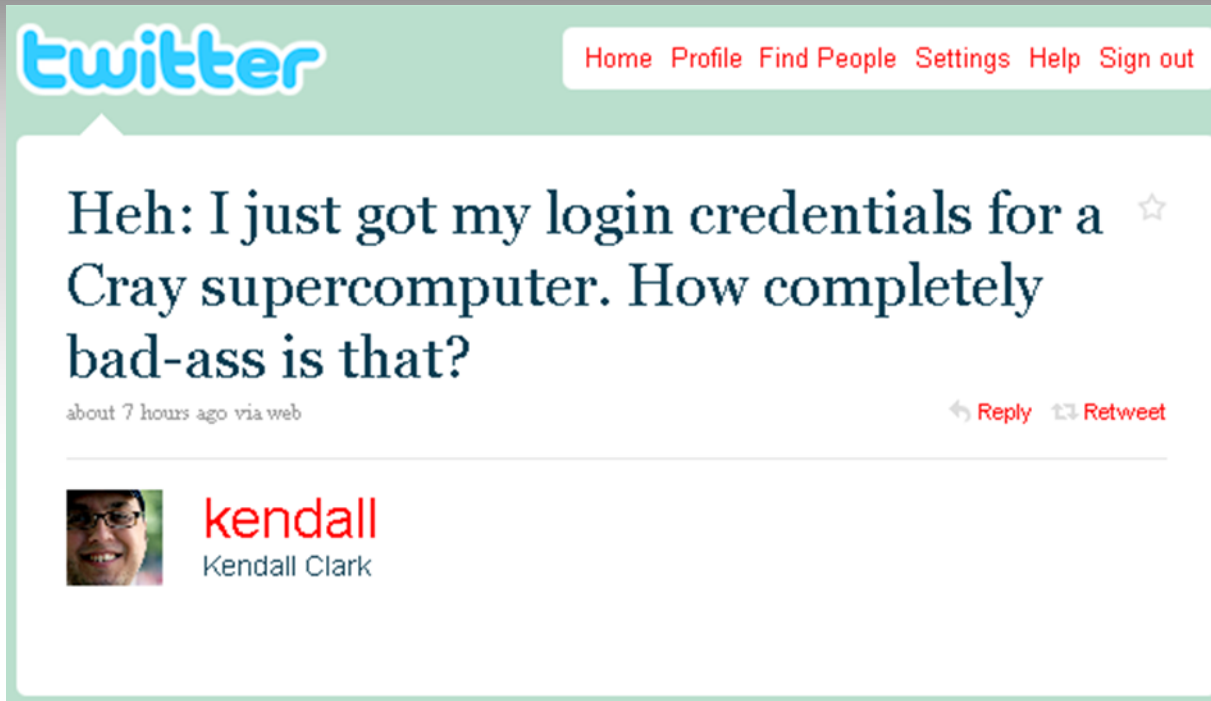
..... But there are options.

# Semantic Web

{ :Inferencing :approximation }

{ :Inference :Infer }

{ :Inference :Assert }



The image shows a screenshot of a Twitter post. At the top left is the Twitter logo. To the right of the logo is a navigation bar with links: Home, Profile, Find People, Settings, Help, and Sign out. The main content is a tweet by Kendall Clark. The text of the tweet is: "Heh: I just got my login credentials for a Cray supercomputer. How completely bad-ass is that?". Below the text is the time "about 7 hours ago via web" and two interaction buttons: "Reply" and "Retweet". At the bottom left of the tweet is a profile picture of Kendall Clark, followed by the name "kendall" in red and "Kendall Clark" in black.


twitter

Home Profile Find People Settings Help Sign out

Heh: I just got my login credentials for a Cray supercomputer. How completely bad-ass is that?

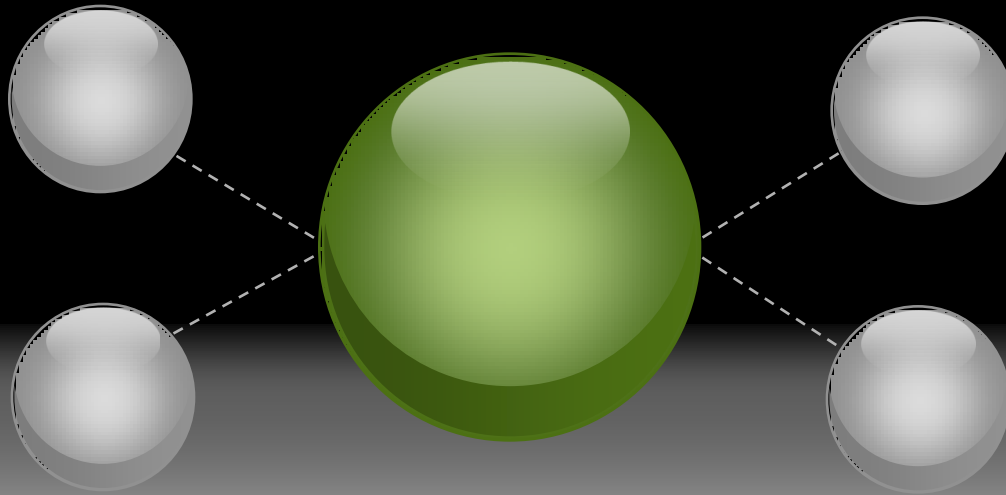
about 7 hours ago via web

Reply Retweet

 **kendall**  
Kendall Clark

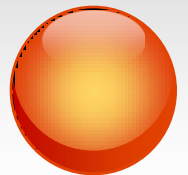
# What is Levenshtein Distance?

## Lower is Better



The Levenshtein distance is a metric for measuring the amount of difference between two sequences (i.e., an edit distance).

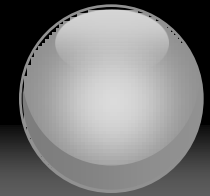
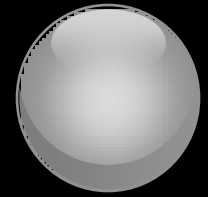
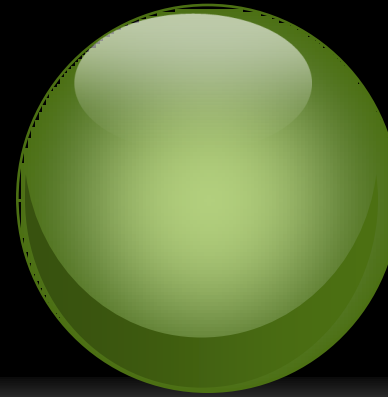
The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other, with the allowable edit operations being insertion or deletion of a single character.



It is named after Vladimir Levenshtein, who considered this distance in 1965.

# RDF Alignment Example Using Levenshtein Scoring

Need to align or replace a triple in two different ontologies?

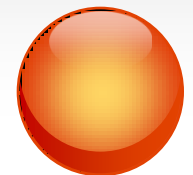


```
LevenshteinScoring levenshteinScoring = new LevenshteinScoring();  
Model m = levenshteinScoring.scoreModel(model, otherModel);
```

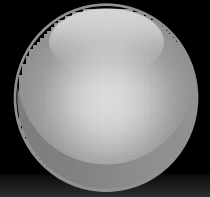
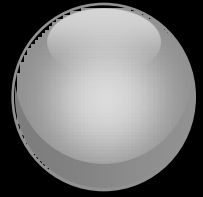
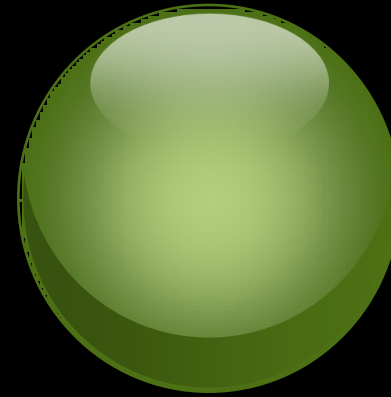
```
int subj = LevenshteinDistance.getDistance( triple.getSubject().getURI().toString(),  
                                             otherTriple.getSubject().getURI().toString());
```

```
int pred = LevenshteinDistance.getDistance( triple.getPredicate().getURI().toString(),  
                                             otherTriple.getPredicate().getURI().toString());
```

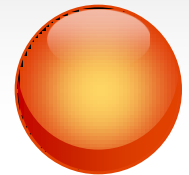
```
int obj = LevenshteinDistance.getDistance( triple.getPredicate(),  
                                           TypeValidation.getStringValue( otherTriple.getPredicate()));
```



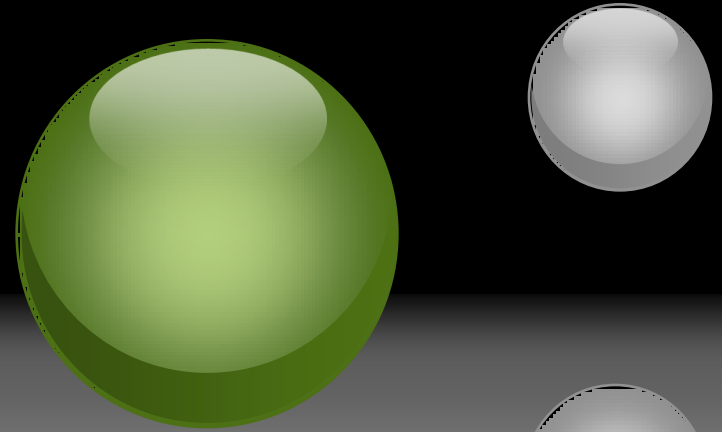
# Example From Sales Demo RDF



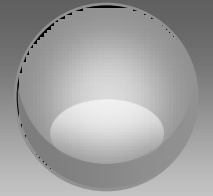
```
<urn:address_STATE_CD/ address_STATE_ABBREV /6>  
  a    <urn:Matching#LevenshteinScoring> ;  
  <urn:matching#otherTriple>  
    <http://localhost:2020/vocab/resource/address_STATE_ABBREV> ;  
  <urn:matching#score>  
    "6" ;  
  <urn:matching#triple>  
    <http://localhost:2020/vocab/resource/address_STATE_CD> .
```



# Need to Change from STATE\_CD to STATE\_ABBREV

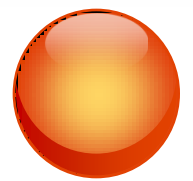


```
SELECT ?stateCD
WHERE {
  ?s vocab:address_STATE_CD ?stateCD.
```



---

```
[] a    sp:Select ;
    sp:resultVariables ( _:b1 _:b2 _:b3 _:b5 ) ;
    sp:where ( [ sp:object _:b5 ;
                sp:predicate <http://localhost:2020/vocab/resource/address_STATE_ABBREV> ;
                sp:subject _:b4
              ] [ sp:object _:b1 ;
```



# SOA Platforms and Architecture

Establish a common, vendor-neutral platform  
Automated discovery, dynamic composition

Outlining the obligations of participants.

Detail how to use a service (sequencing, state management, etc.).

Metadata to accompany a WSDL description.

Replace somewhat imprecise instructions with precise language.

## **OWL-S: Semantic Markup for Web Services**

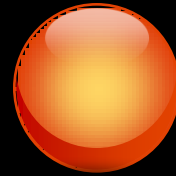
<http://www.daml.org/services/owl-s/1.2/overview>

## **Web Service Choreography (WS-Choreography)**

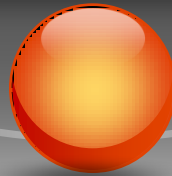
<http://www.w3.org/2005/12/wscwg-charter.html>

# Owl-S Answers These Questions

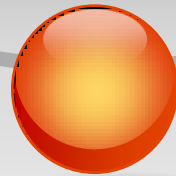
Four main classes



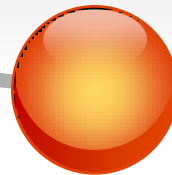
Service



Profile



Process



Grounding

*What does the service provide?*

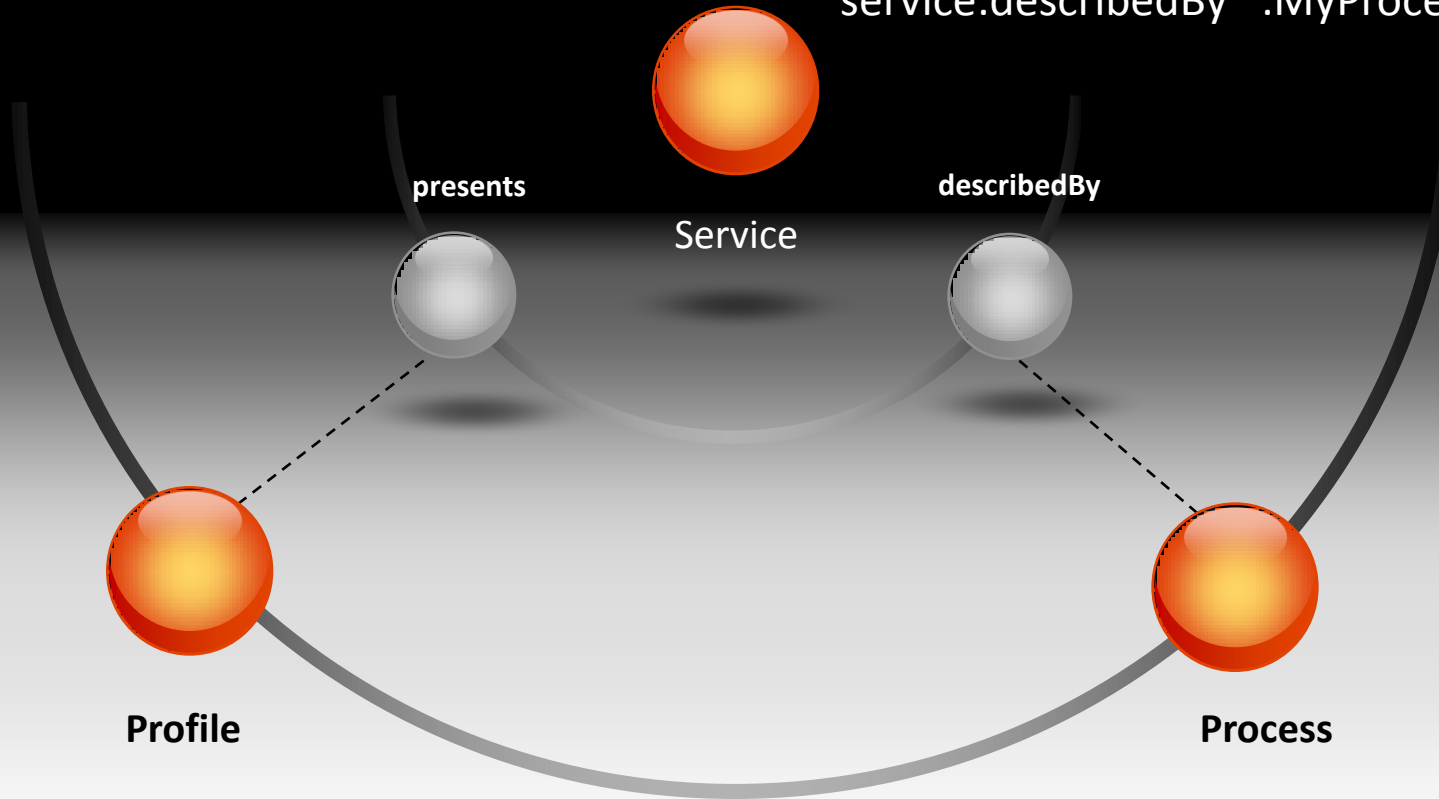
*How is it used?*

*How does one interact with it?*

# owls:service

Answers the “what”

```
MyService a service:Service;  
#Reference to the Profile  
service:presents :MyProfile;  
#Reference to the Process Model  
service:describedBy :MyProcess;
```

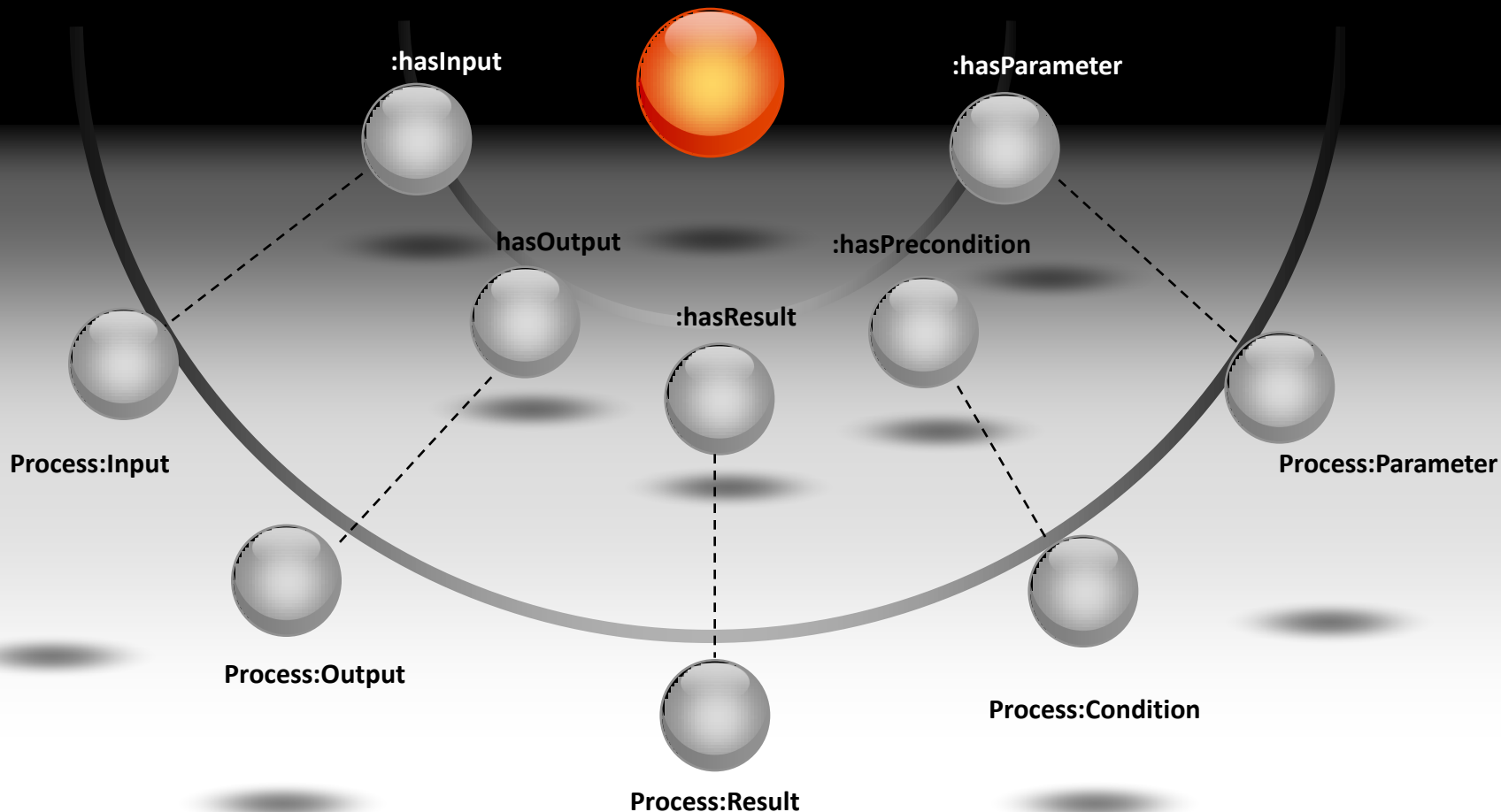


# owls:profile

Inverse of the “how”

```
MyProfile a profile:Profile ;  
profile:contactInformation :MyContactInfo;  
profile:hasInput :input_one , :input_two;  
profile:hasOutput :MyOutput;  
profile:hasPrecondition :MyPrecondition;  
profile:hasResult : MyResult;  
profile:has_process :MyProcess;
```

Profile

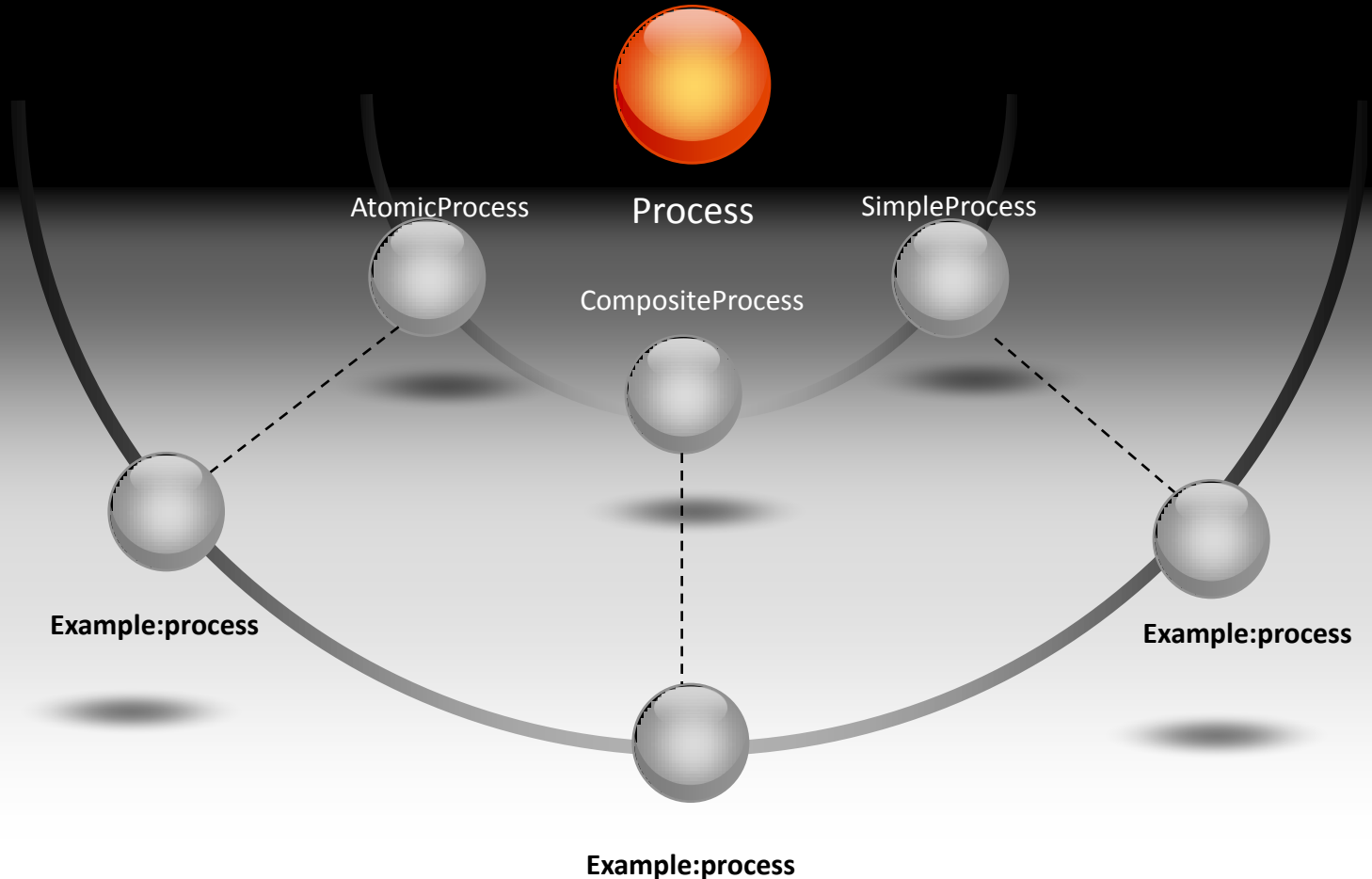


# owls:process

Answers the “how”

```
process:MyProcess
```

```
  a process:AtomicProcess ;  
  process:hasInput :MyInput ;  
  process:hasOutput :process:MyOutput ;  
  process:hasResult process:MyInput .
```

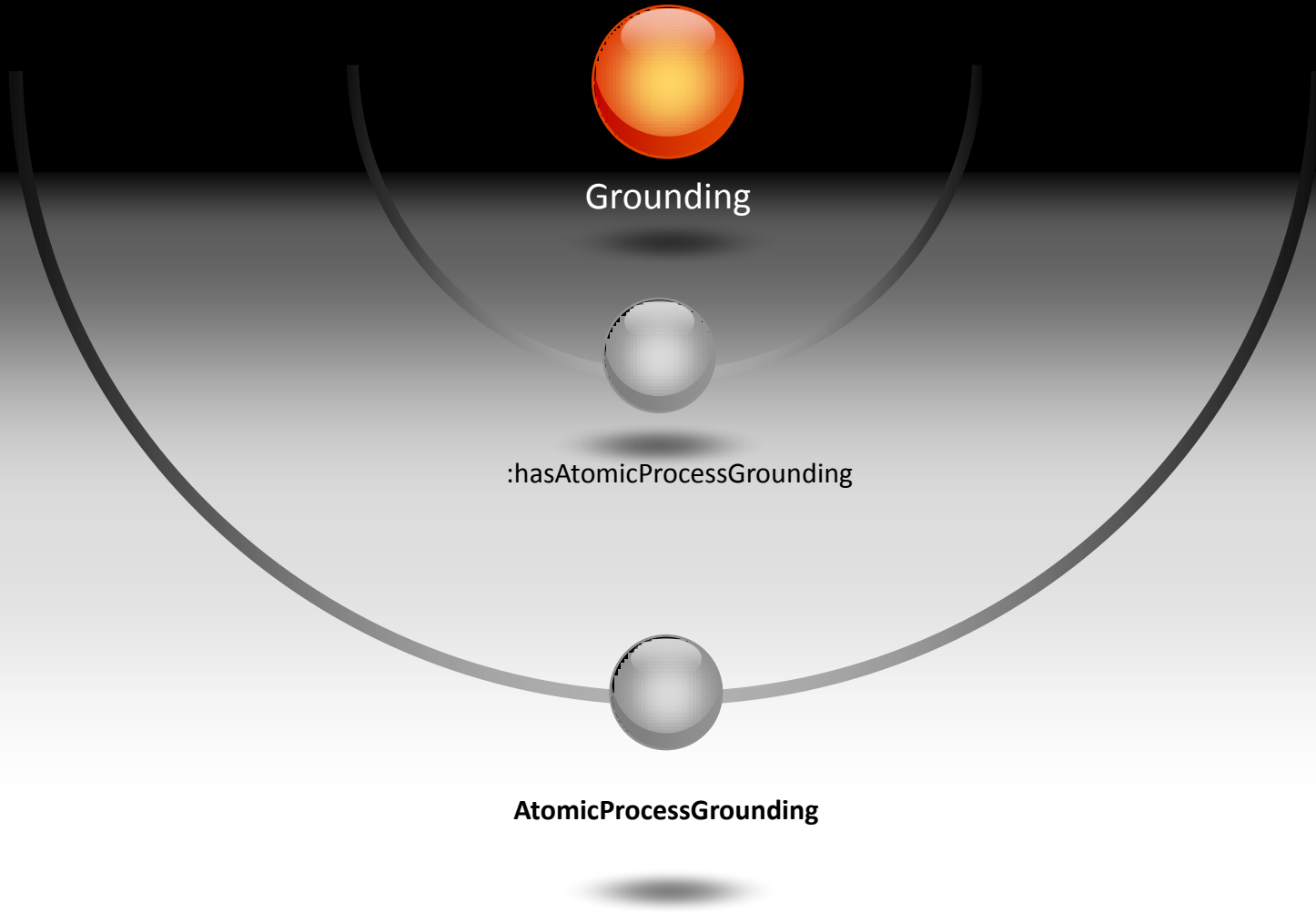


# owls:grounding

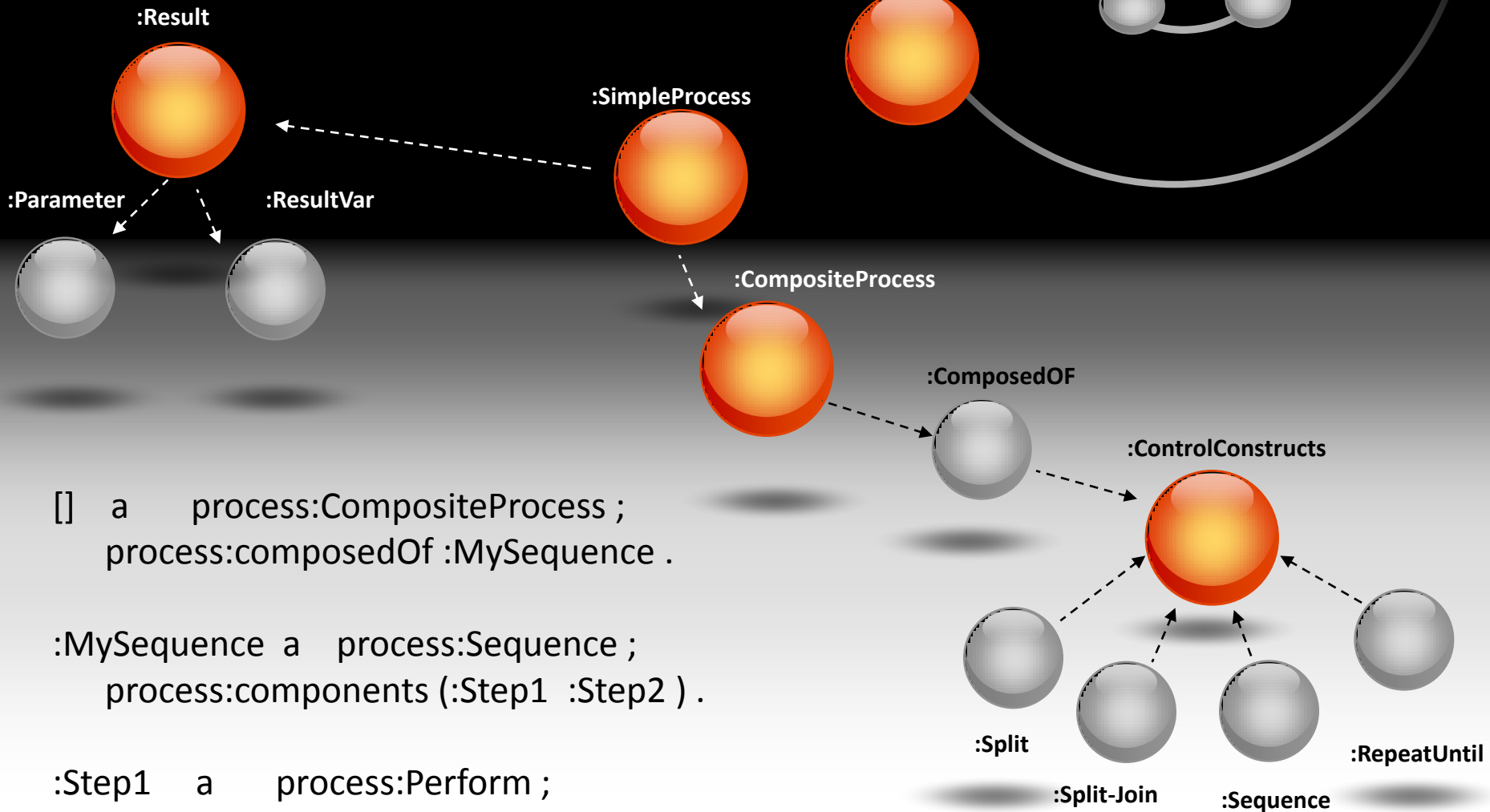
Answers the “where”

```
[] a grounding:WsdIGrounding  
:hasAtomicProcessGrounding :WsdIGrounding.
```

```
:WsdIGrounding  
a grounding:WsdIAtomicProcessGrounding.
```



# Select Classes

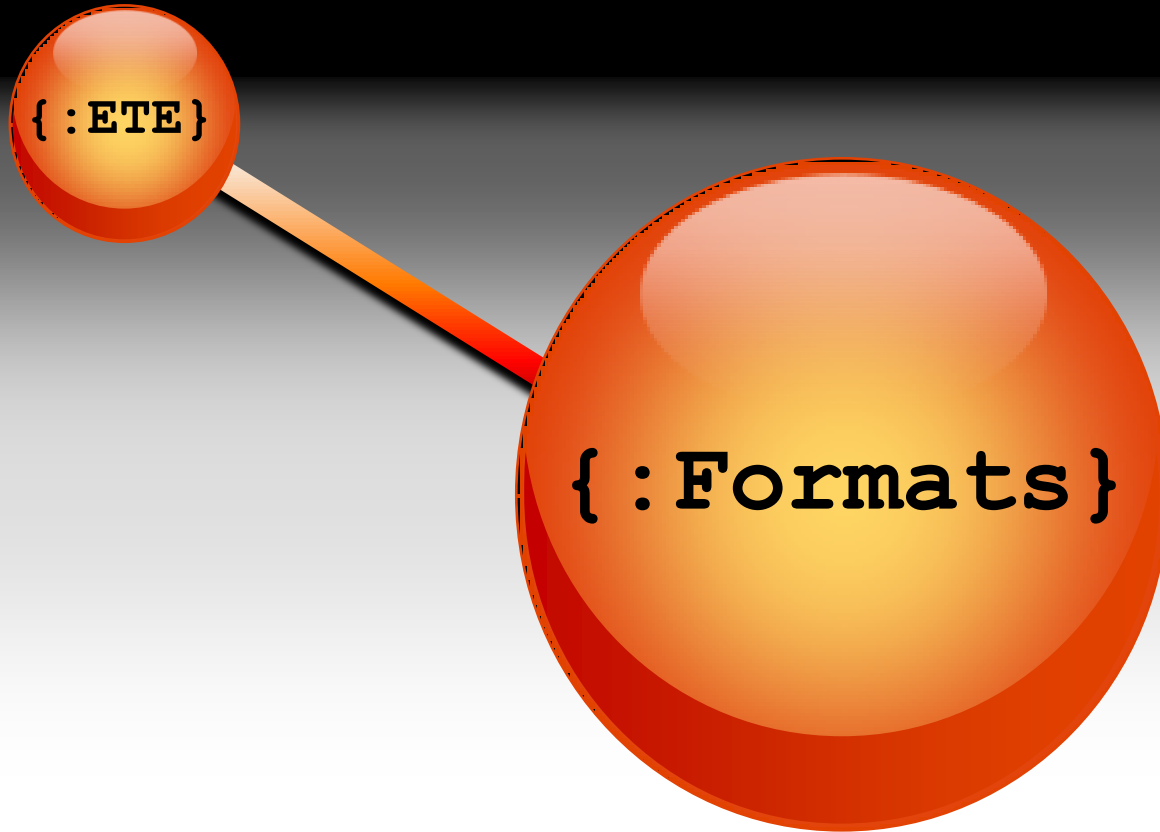


```
[] a process:CompositeProcess ;  
process:composedOf :MySequence .
```

```
:MySequence a process:Sequence ;  
process:components (:Step1 :Step2 ) .
```

```
:Step1 a process:Perform ;  
process:process :MyProcess.
```

# Semantic Web



# RDF: Example

Can be hard to read, too verbose, and large to transfer

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <foaf:Person>
    <foaf:name>Brian Mackay</foaf:name>
    <foaf:mbox rdf:resource="mailto:brian.mackay@brianwmackay@msn.com"/>
  </foaf:Person>
</rdf:RDF>
```

# How to Read and Write RDF in Notation 3

Use prefixes as much as possible, more readable, smaller size

Allows RDF to be shown in a readable, natural, format. Reduces file sizes.

You can learn basic syntax by remembering a few rules.

Basic syntax is enough for most situations.

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

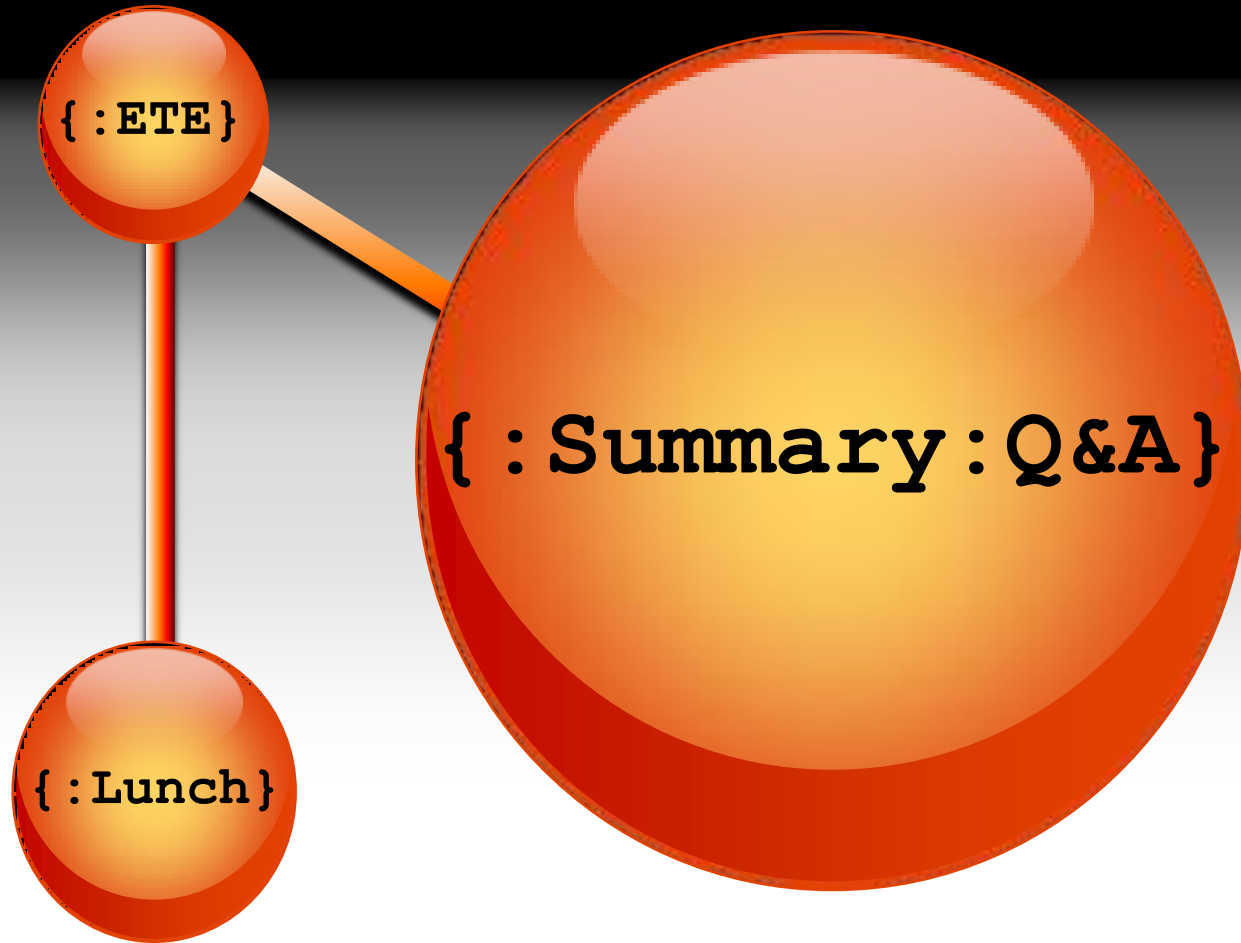
```
@prefix : <http://myurl.com/#> .
```

```
[] a :SomeRDFClass ;  
   :has Type xsd:String ;  
   :hasLiteral "Some Value" .
```

# Review: N3 Rules to Know

- 1). Resources must end with a period.
- 2). Use prefixes as much as possible.
- 3). End each triple with a semi-colon.
- 4). [] is short for bnode or anonymous.
- 5). Use “a” instead of `rdf:type`

# Semantic Web



# Semantic Web

## { :References }

**Semantic Web - History by Charles McCN**

<http://www.w3.org/2001/Talks/0627-semwebh/Overview.html>

**The Semantic Web: The Origins of Artificial Intelligence Redux - Harry Halpin**

<http://www.ibiblio.org/hhalpin/homepage/publications/html/airedux/>

**The D2RQ Plattform - Treating Non-RDF Databases as Virtual RDF Graphs - Prof. Dr. Chris Bizer**

<http://www4.wiwiss.fu-berlin.de/bizer/d2rq/>

**FOAF-a-Matic**

<http://www.ldodds.com/foaf/foaf-a-matic>

**Wikipedia Semantic Web**

[http://en.wikipedia.org/wiki/Semantic\\_Web](http://en.wikipedia.org/wiki/Semantic_Web)

**SIOC Semantically-Interlinked Online Communities**

<http://sioc-project.org/>

**United Kingdom Open Data**

<http://data.gov.uk/>

**United States Open Data**

<http://www.data.gov/>

**TED Talks - Tim Berners-Lee**

[http://www.ted.com/talks/tim\\_berners\\_lee\\_on\\_the\\_next\\_web.html](http://www.ted.com/talks/tim_berners_lee_on_the_next_web.html)

[http://www.ted.com/talks/tim\\_berners\\_lee\\_the\\_year\\_open\\_data\\_went\\_worldwide.html](http://www.ted.com/talks/tim_berners_lee_the_year_open_data_went_worldwide.html)

**TED Talk - 2009 Slides**

<http://www.w3.org/2009/Talks/0204-ted-tbl/#%281%29>

**Protégé Ontology Editor**

<http://protege.stanford.edu/>

**D2Rq**

<http://sourceforge.net/projects/d2rq-map/>

**Semweb Commons**

<http://code.google.com/p/semwebcommons/>

**SpinRDF**

<http://www.spinrdf.org/>

**SuperComputer Fun**

<http://twitter.com/kendall/status/11790574752>

**Hypertext Timeline**

[http://en.wikipedia.org/wiki/Timeline\\_of\\_hypertext\\_technology](http://en.wikipedia.org/wiki/Timeline_of_hypertext_technology)

**Jena**

<http://jena.sourceforge.net/>

<http://sourceforge.net/projects/jena/files/>

<http://openjena.org/index.html>

<http://jena.sourceforge.net/contrib/contributions.html>

**Books**

<http://workingontologist.org/>

<http://semwebprogramming.com/>

**Linked Data**

<http://linkeddata.org/>

# Semantic Web

{ :foaf:PersonalProfileDocument }

