情報・システム研究機構
国立情報学研究所
National Institute of Informatics

GALAXY EXPRESS

OWL-Full Reasoning from Object Oriented Perspective

Seiji Koide[1,2] and Hideaki Takeda[1]

[1]National Institute of Informatics

[2]Galaxy Express Corporation

Agenda

- Semantic Gap between SW and OOP
- Metamodeling in RDF Universe
- How to connect RDF and OWL Universe
- Extended Structural Algorithm for OWL Subsumption
- OWL-Full Programming by Metamodeling

Agenda

- Semantic Gap between SW and OOP
- Metamodeling in RDF Universe
- How to connect RDF and OWL Universe
- Extended Structural Algorithm for OWL Subsumption
- OWL-Full Programming by Metamodeling

The Comparison of OWL/RDF and Object-Oriented Languages (from SETF)

Object-Oriented Languages	OWL and RDF
Domain models consist of classes, properties and instances (individuals). Classes can be arranged in a subclass hierarchy with inheritance. Properties can take objects or primitive values (literals) as values.	
Classes and Instances	
Classes are regarded as types for instances.	Classes are regarded as sets of individuals.
Each instance has one class as its type. Classes cannot share instances.	Each individual can belong to multiple classes.
Instances can not change their type at runtime.	Class membership may change at runtime.
The list of classes is fully known at compile-time and cannot change after that.	Classes can be created and changed at runtime.
Compilers are used at build-time. Compile-time errors indicate problems.	Reasoners can be used for classification and consistency checking at runtime or build-time.

From http://www.w3.org/2001/sw/BestPractices/SE/ODSD/

The Comparison of OWL/RDF and Object-Oriented Languages

CLOS Language	OWL and RDF
Domain models consist of classes, properties and instances (individuals). Classes can be arranged in a subclass hierarchy with inheritance. Properties can take objects or primitive values (literals) as values.	
Classes and Instances	
Classes are regarded as types for instances.	Classes are regarded as sets of individuals.
Each instance has one class as its type. Classes cannot share instances.	Each individual can belong to multiple classes.
change-class is applicable to instances at runtime.	Class membership may change at runtime.
Classes can be created and changed at runtime, because classes are also instances of meta-classes.	Classes can be created and changed at runtime.
Dynamic compiling and linking in runtime is available.	Reasoners can be used for classification and consistency checking at runtime or build-time.

Modified from http://www.w3.org/2001/sw/BestPractices/SE/ODSD/

The Comparison of OWL/RDF and Object-Oriented Languages

CLOS Language	OWL and RDF
Domain models consist of classes, properties and instances (individuals). Classes can be arranged in a subclass hierarchy with inheritance. Properties can take objects or primitive values (literals) as values.	
Classes and Instances	
Classes are regarded as types for instances.	Classes are regarded as sets of individuals.
Each instance has one class as its type. Classes cannot share instances.	Each individual can belong to multiple classes.
change-class is applicable to instances at runtime.	Class membership may change at runtime.
Classes can be created and changed at runtime, because classes are also instances of meta-classes.	Classes can be created and changed at runtime.
Dynamic compiling and linking in runtime is available.	Reasoners can be used for classification and consistency checking at runtime or build-time.

Modified from http://www.w3.org/2001/sw/BestPractices/SE/ODSD/

The Comparison of OWL/RDF and Object-Oriented Languages

SWCLOS Language	OWL and RDF
Domain models consist of classes, properties and instances (individuals). Classes can be arranged in a subclass hierarchy with inheritance. Properties can take objects or primitive values (literals) as values.	
Classes and Instances	
OWL/RDF entailments are implemented.	Classes are regarded as sets of individuals.
Multiple classing is implemented.	Each individual can belong to multiple classes.
change-class is applicable to instance	ay change at runtime.
Classes can be created and changed because classes are also instances of	ted and changed at runtime.
Consistency checking and *proactive entaiments* in runtime are implemented.	Reasoners can be used for classification and consistency checking at runtime or build-time.

perform entailment without demand from users

Modified from http://www.w3.org/2001/sw/BestPractices/SE/ODSD/

The Comparison of OWL/RDF and Object-Oriented Languages

Object-Oriented Languages	OWL and RDF
Properties, Attributes and Values	
Properties are defined locally to a class (and its subclasses through inheritance).	Properties are stand-alone entities that can exist without specific classes.
Instances can have values only for the attached properties. Values must be correctly typed. Range constraints are used for type checking.	Instances can have arbitrary values for any property. Range and domain constraints can be used for type checking and type inference.
Classes encode much of their meaning and behavior through imperative functions and methods.	Classes make their meaning explicit in terms of OWL statements. No imperative code can be attached.
Classes can encapsulate their members to private access.	All parts of an OWL/RDF file are public and can be linked to from anywhere else.
Closed world: If there is not enough information to prove a statement true, then it is assumed to be false.	Open world: If there is not enough information to prove a statement true, then it may be true or false.

From http://www.w3.org/2001/sw/BestPractices/SE/ODSD/

The Comparison of OWL/RDF and Object-Oriented Languages

CLOS Language	OWL and RDF
Properties, Attributes and Values	
CLOS slots are defined locally to a class (and its subclasses through inheritance).	Properties are stand-alone entities that can exist without specific classes.
Instances can have values only for the attached properties. Values must be correctly typed. Range constraints may be used for type checking.	Instances can have arbitrary values for any property. Range and domain constraints can be used for type checking and type inference.
Classes may encode much of their meaning and behavior through imperative functions and methods.	Classes make their meaning explicit in terms of OWL statements. No imperative code can be attached.
Classes can encapsulate their members to private access.	All parts of an OWL/RDF file are public and can be linked to from anywhere else.
Closed world: If there is not enough information to prove a statement true, then it is assumed to be false.	Open world: If there is not enough information to prove a statement true, then it may be true or false.

Modified from http://www.w3.org/2001/sw/BestPractices/SE/ODSD/

The Comparison of OWL/RDF and Object-Oriented Languages

SWCLOS Language	OWL and RDF
Properties, Attributes and Values	
Property definitions are class objects, the extensions of property are object slots.	Properties are stand-alone entities that can exist without specific classes.
Flexible slot attachment is implemented. **Range and domain constraints are also realized**, and the entailments are implemented.	Instances can have arbitrary values for any property. Range and domain constraints can be used for type checking and type inference.
A number of OWL entailments are implemented.	Classes make their meaning explicit in terms of OWL statements. No imperative code can be attached.
Variables may be public or private, depending on programming.	All parts of an OWL/RDF file are public and can be linked to from anywhere else.
t means Boolean true, but nil is regarded unknown.	Open world: If there is not enough information to prove a statement true, then it may be true or false.

Modified from http://www.w3.org/2001/sw/BestPractices/SE/ODSD/

Agenda

- Semantic Gap between SW and OOP
- **Metamodeling in RDF Universe**
- How to connect RDF and OWL Universe
- Extended Structural Algorithm for OWL Subsumption
- OWL-Full Programming by Metamodeling

Straight-forward Mapping

- Name space to lisp package
- QName to lisp symbol
- `rdf:type` to class-instance
- `rdfs:subClassOf` to superclass-subclass
- Property-value to slot-value
- RDF graph to Objects, which is bound to the name symbol, and slots or role-filler pairs of objects

Entailment Rules

http://www.w3.org/TR/rdf-mt/

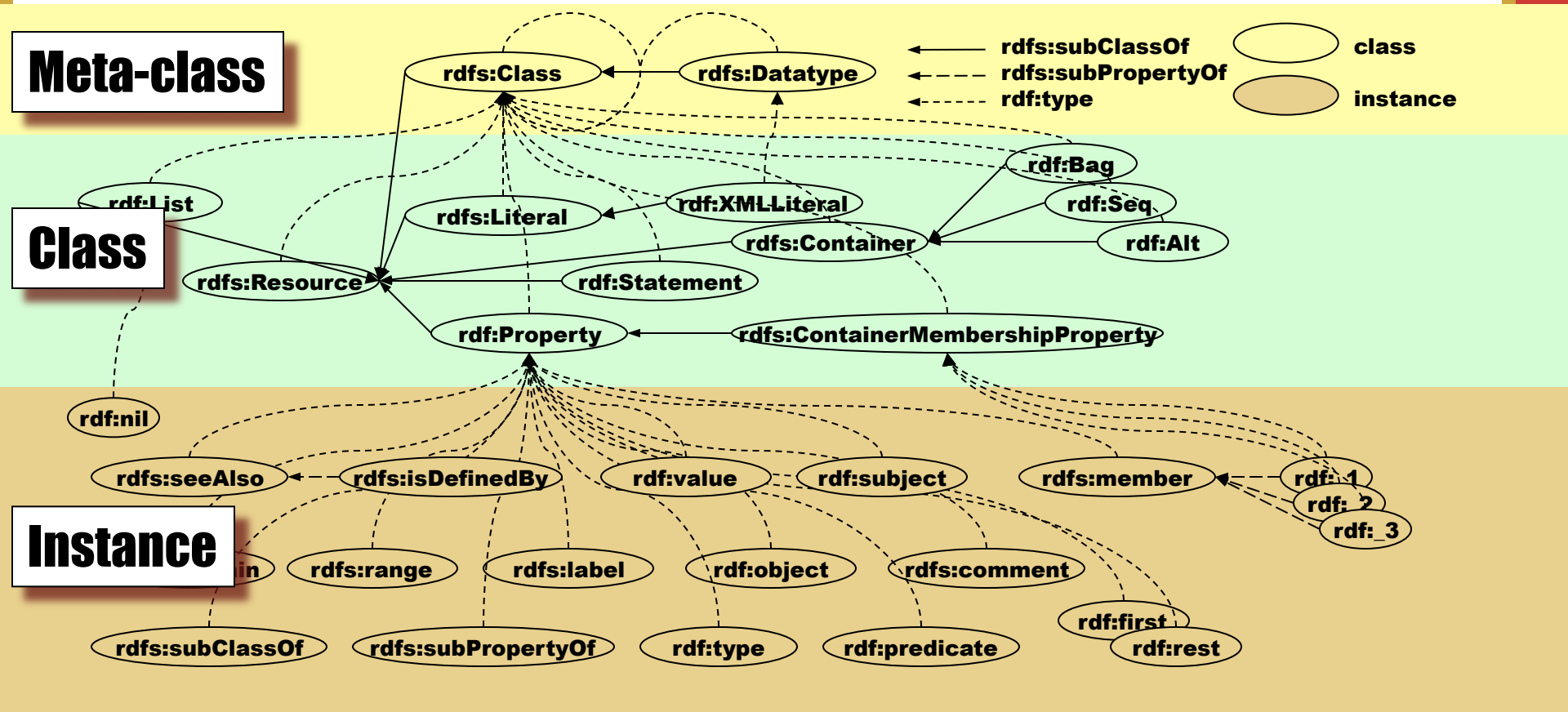
Subsumption Rule

Rule Name	If E contains:	then add:
rdfs9	uuu rdfs:subClassOf xxx . vvv rdf:type uuu .	vvv rdf:type xxx .
rdfs10	uuu rdf:type rdfs:Class .	uuu rdfs:subClassOf uuu .
rdfs11	uuu rdfs:subClassOf vvv . vvv rdfs:subClassOf xxx .	uuu rdfs:subClassOf xxx .
rdfs12	uuu rdf:type rdfs:ContainerMembershipProperty .	uuu rdfs:subPropertyOf rdfs:member
rdfs13	uuu rdf:type rdfs:Datatype .	uuu rdfs:subClassOf rdfs:Literal .

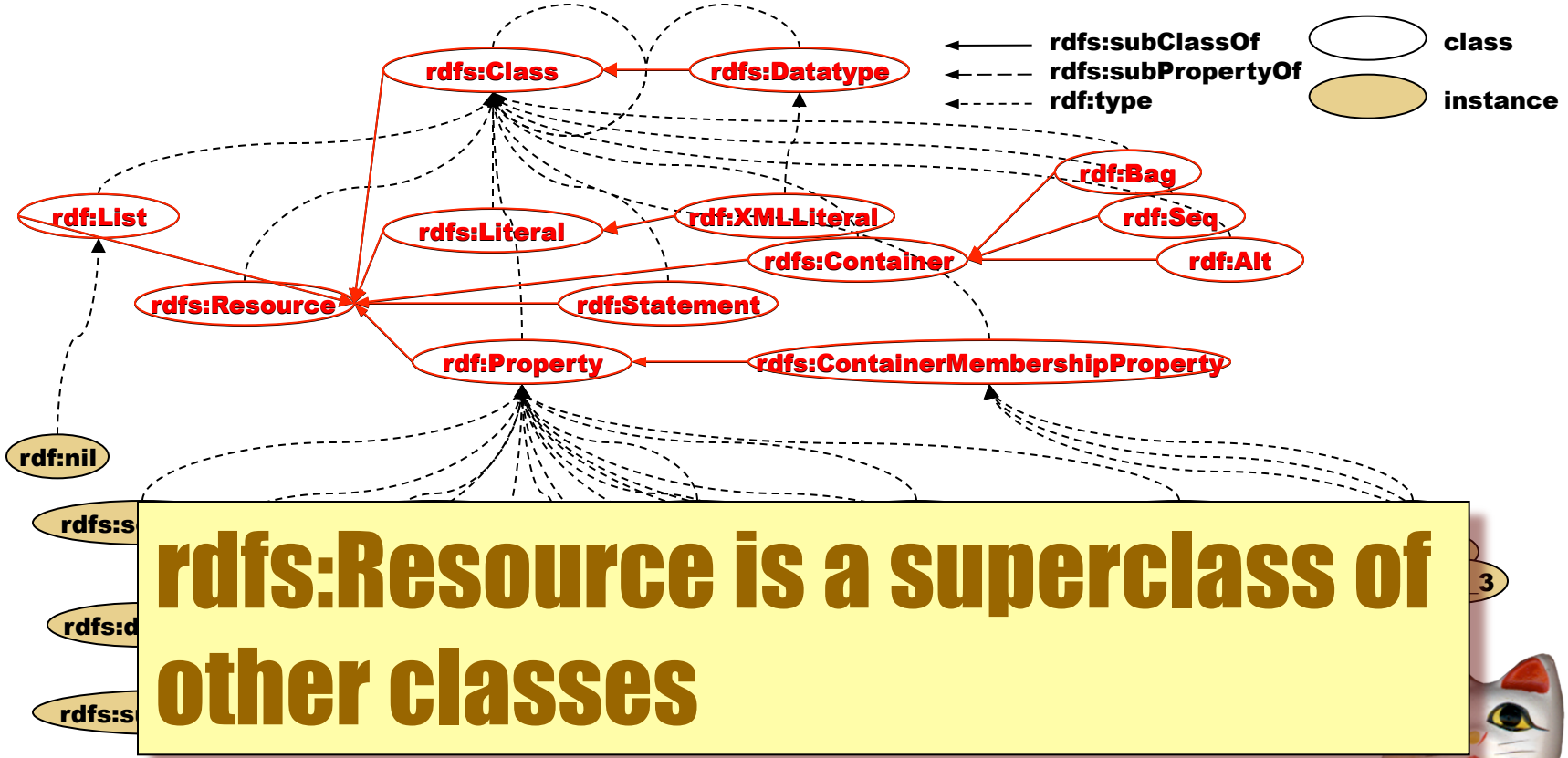
Transitivity Rule

```
(defclass xxx ( ) ( ) )
(setq vvv (make-instance (defclass uuu (xxx) ( ))))
(typep vvv xxx) → true
(defclass xxx ( ) ( ))
(defclass vvv (xxx) ( )) (defclass uuu (vvv) ( ))
(subtypep 'uuu 'xxx) → true
```

Hierarchical Structure of RDFS

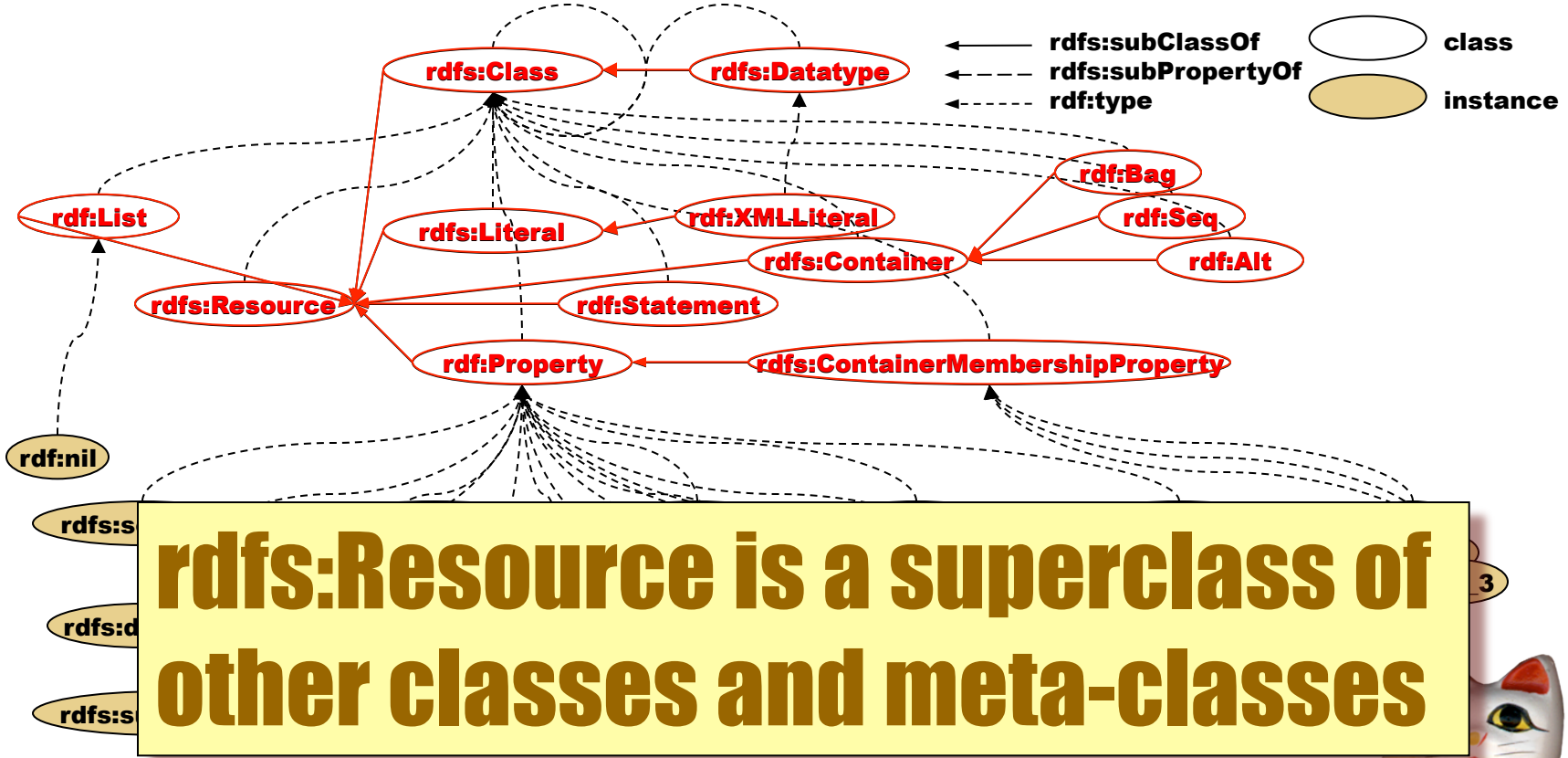


Transitivity Rule implies . . .



rdfs:Resource is a superclass of other classes

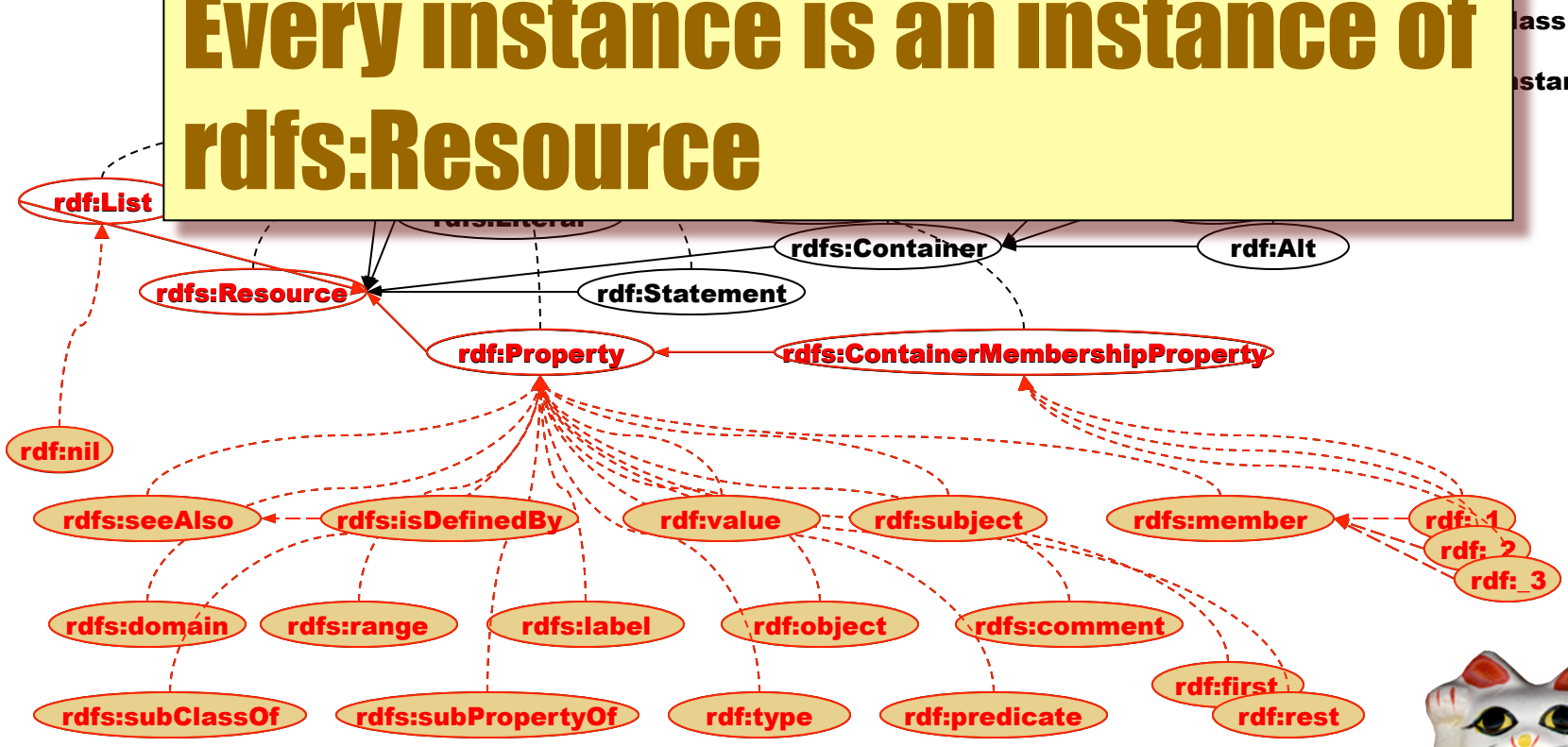
Transitivity Rule implies . . .



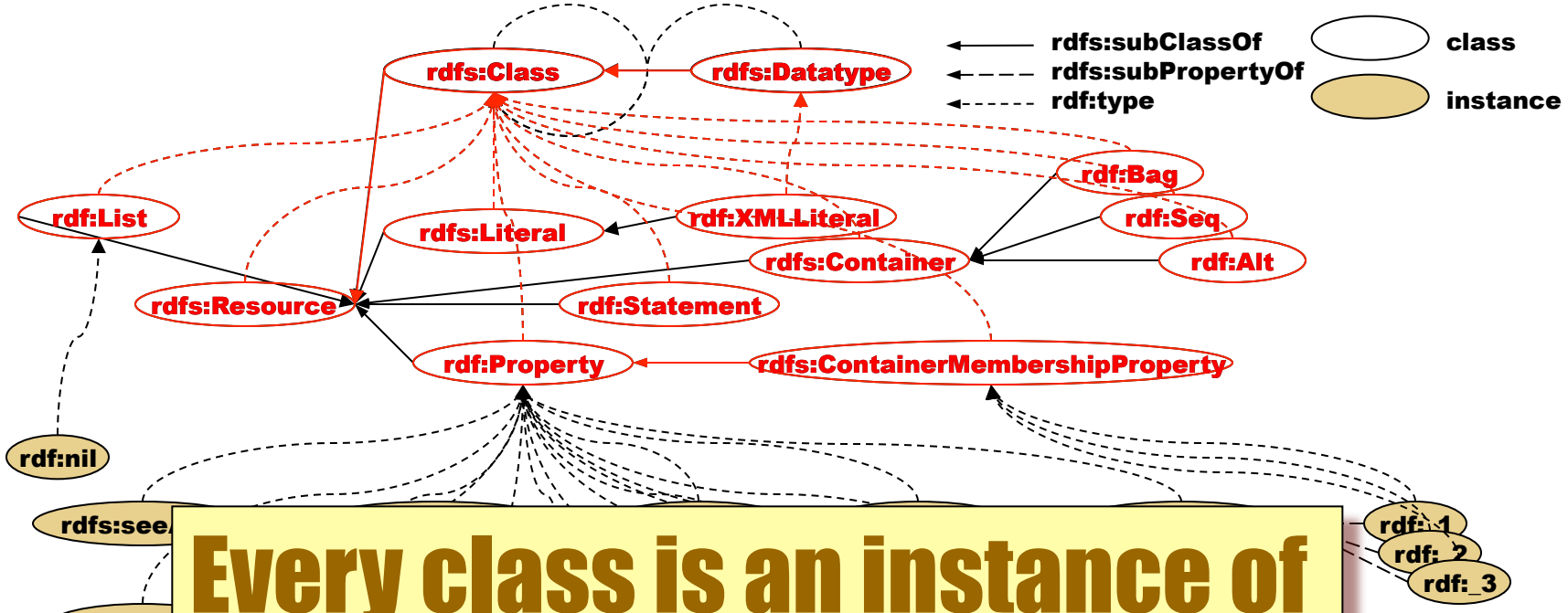
rdfs:Resource is a superclass of other classes and meta-classes

Subsumption Rule implies . . .

Every instance is an instance of rdfs:Resource

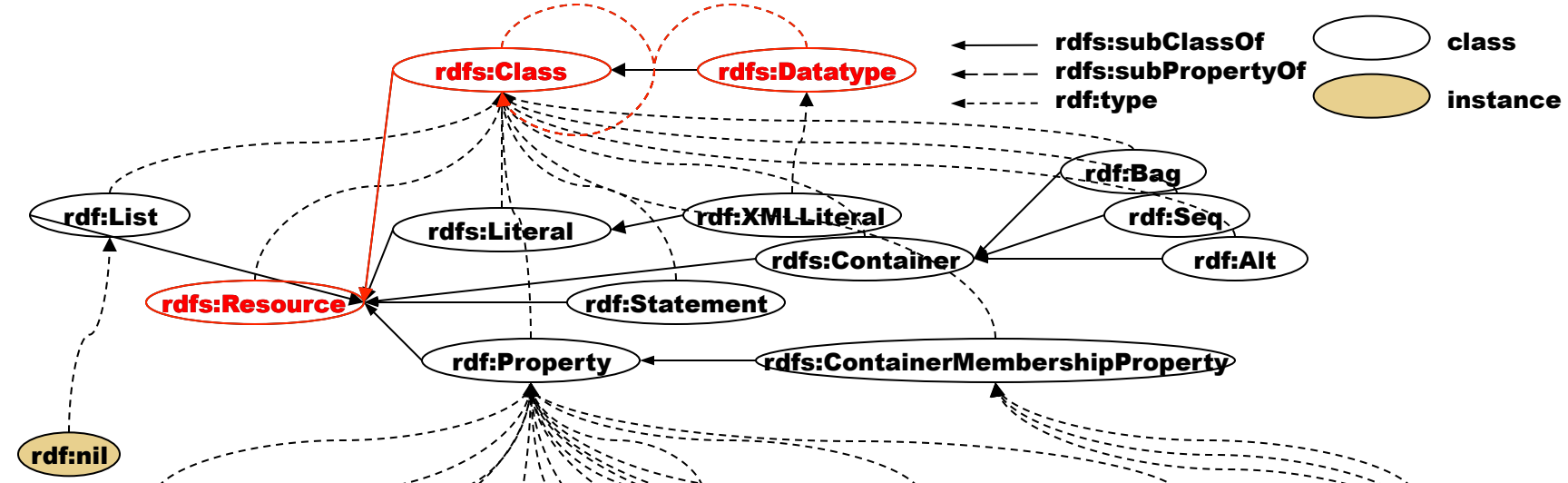


Subsumption Rule implies . . .



Every class is an instance of rdfs:Resource Class

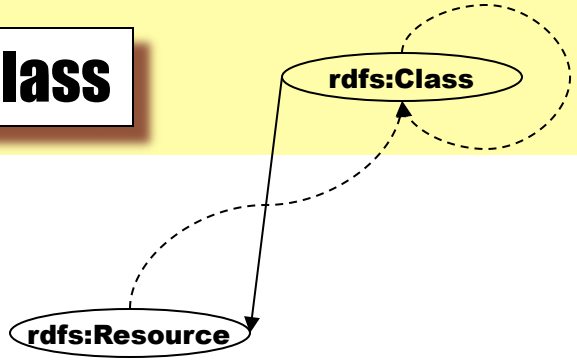
Subsumption Rule implies . . .



rdfs:Datatype and rdfs:Class are also an instance of rdfs:Resource

Membership Loop in RDFS

Meta-class



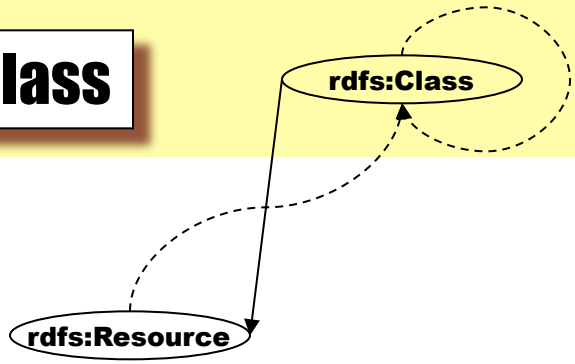
rdfs:subClassOf
rdfs:subPropertyOf
rdf:type

class
instance

Membership Loop in RDFS

rdfs:subClassOf
rdfs:subPropertyOf
rdf:type
class
instance

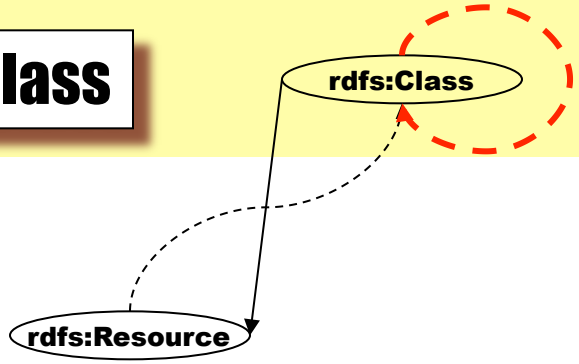
Meta-class



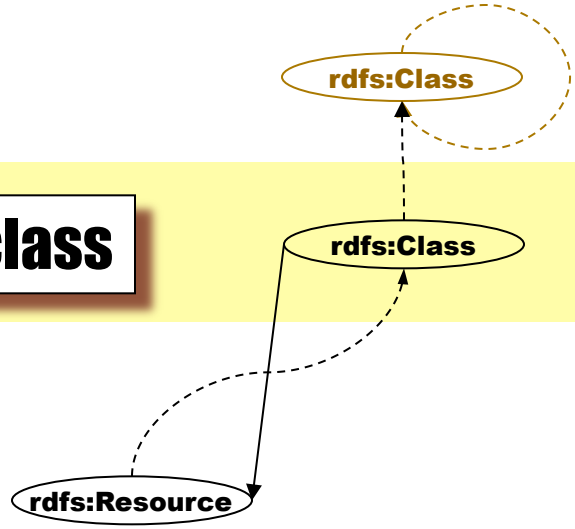
Membership Loop in RDFS

rdfs:subClassOf
rdfs:subPropertyOf
rdf:type
class
instance

Meta-class

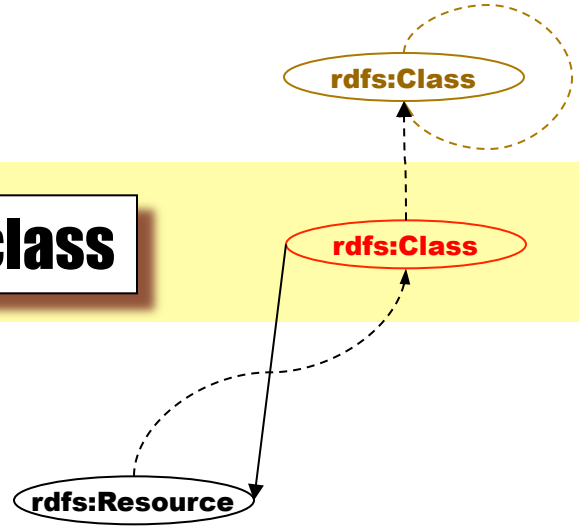


Membership Loop in RDFS



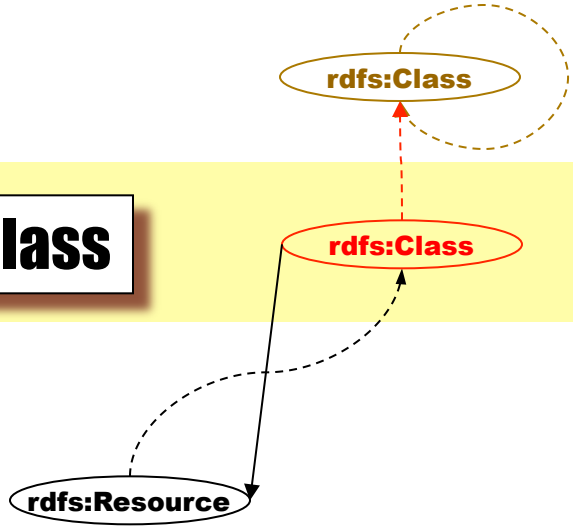
Meta-class

Membership Loop in RDFS



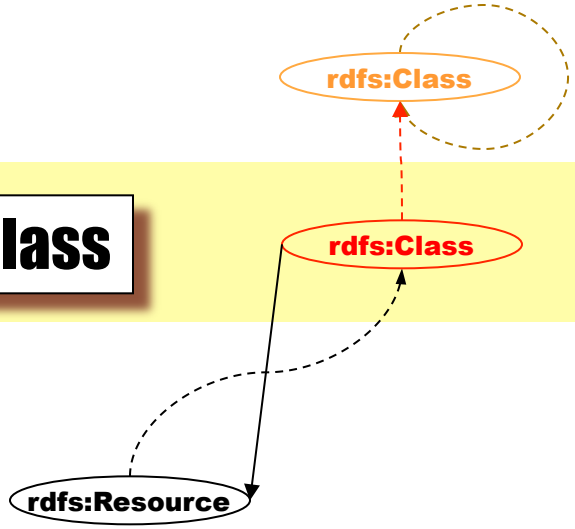
Meta-class

Membership Loop in RDFS



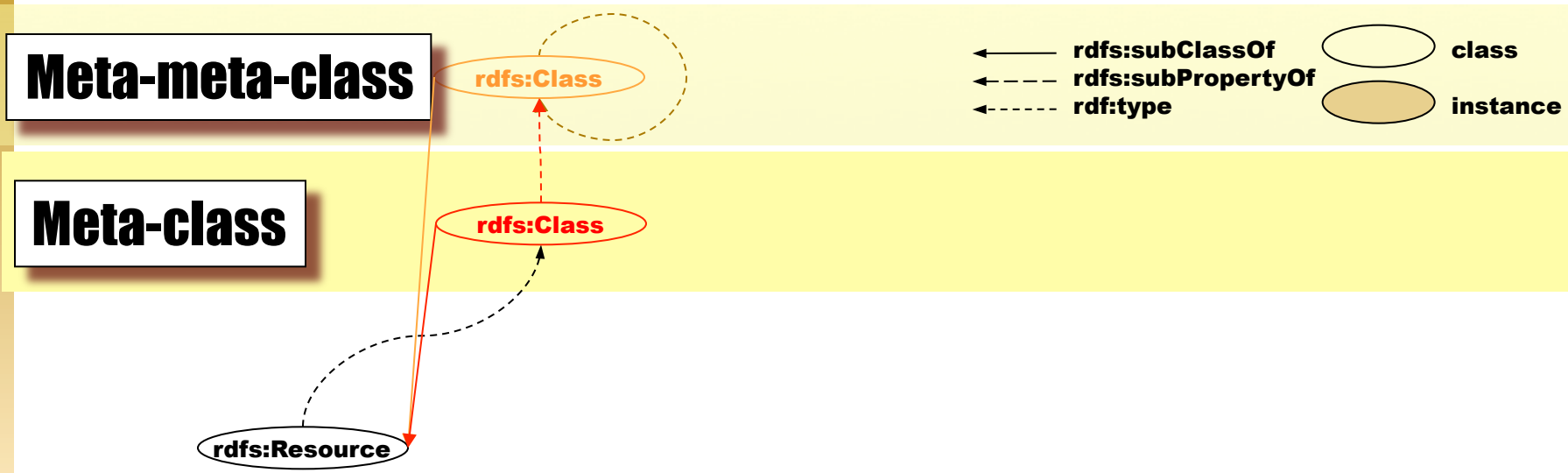
Meta-class

Membership Loop in RDFS

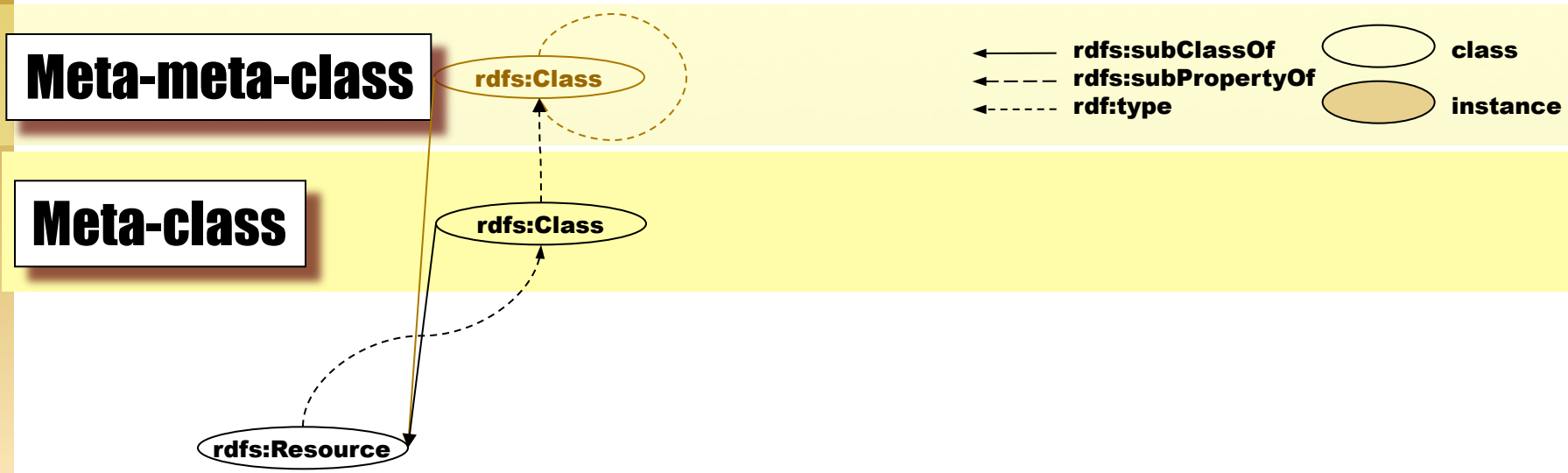


Meta-class

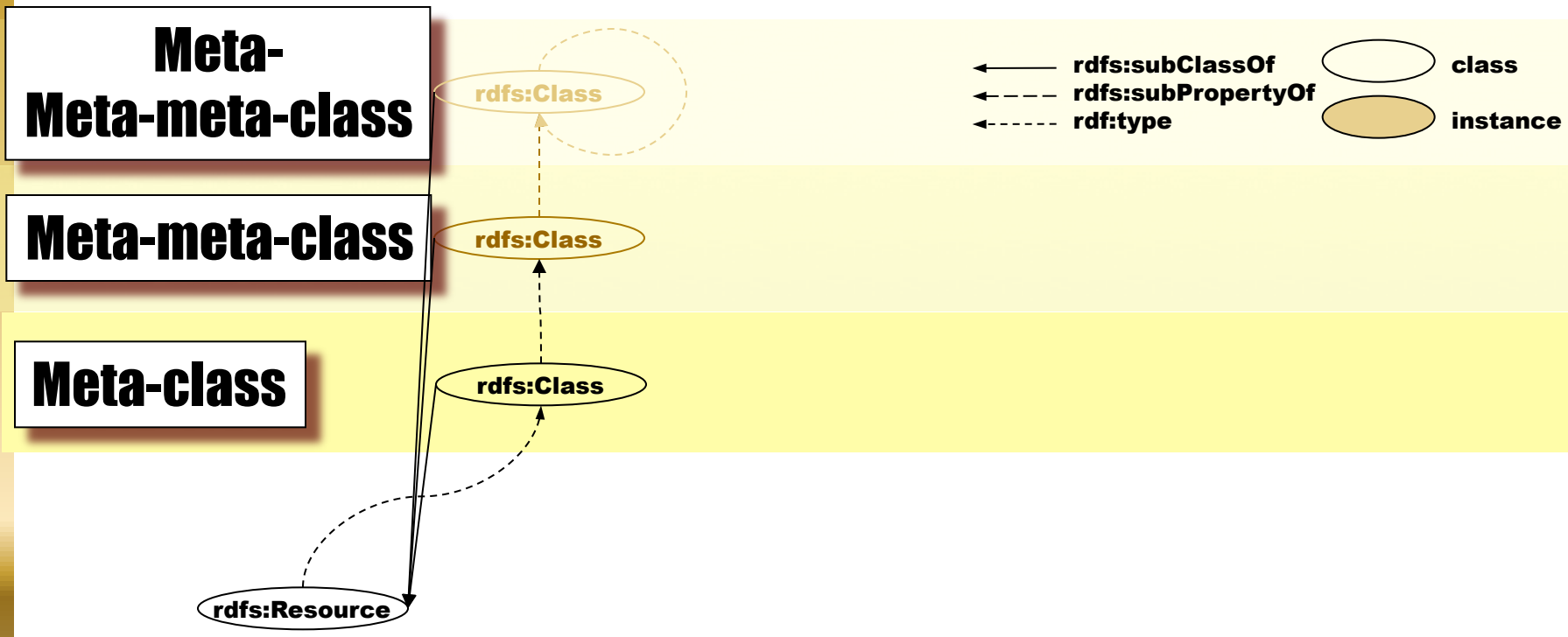
Membership Loop in RDFS



Meta-meta-Class



Meta-meta-meta-Class



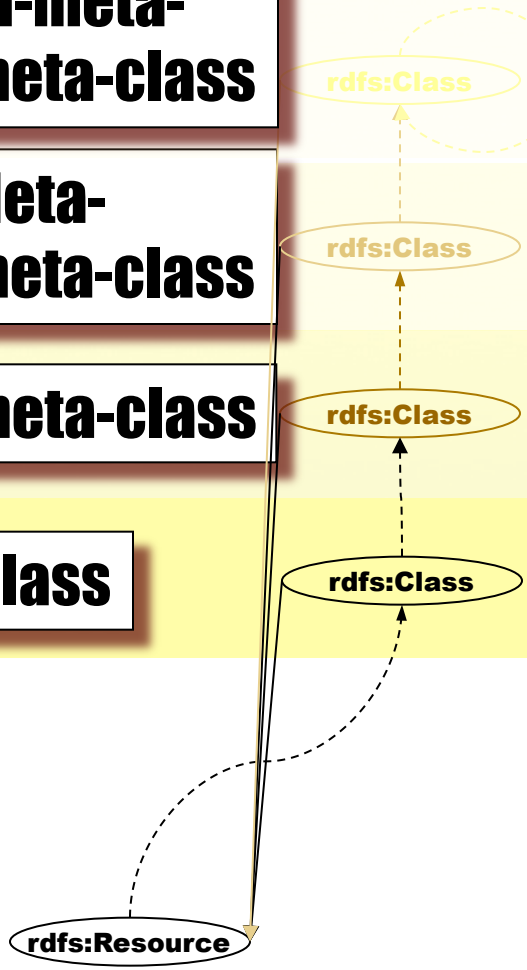
Meta-meta-meta-meta-Class

Meta-meta-meta-meta-class

Meta-meta-meta-class

Meta-meta-class

Meta-class



Meta-meta-meta-meta-meta-Class

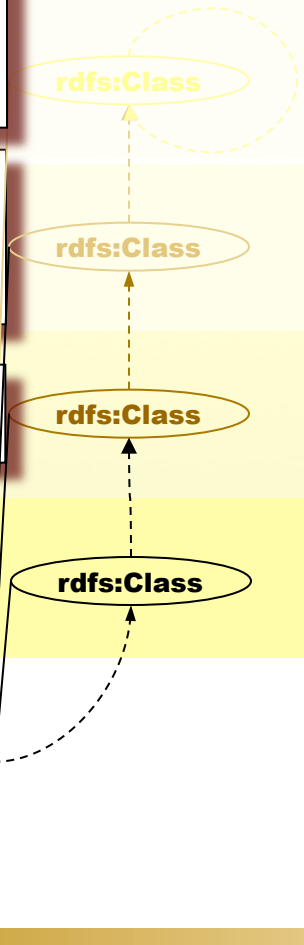
Meta-meta-meta-meta-meta-class

Meta-meta-meta-meta-class

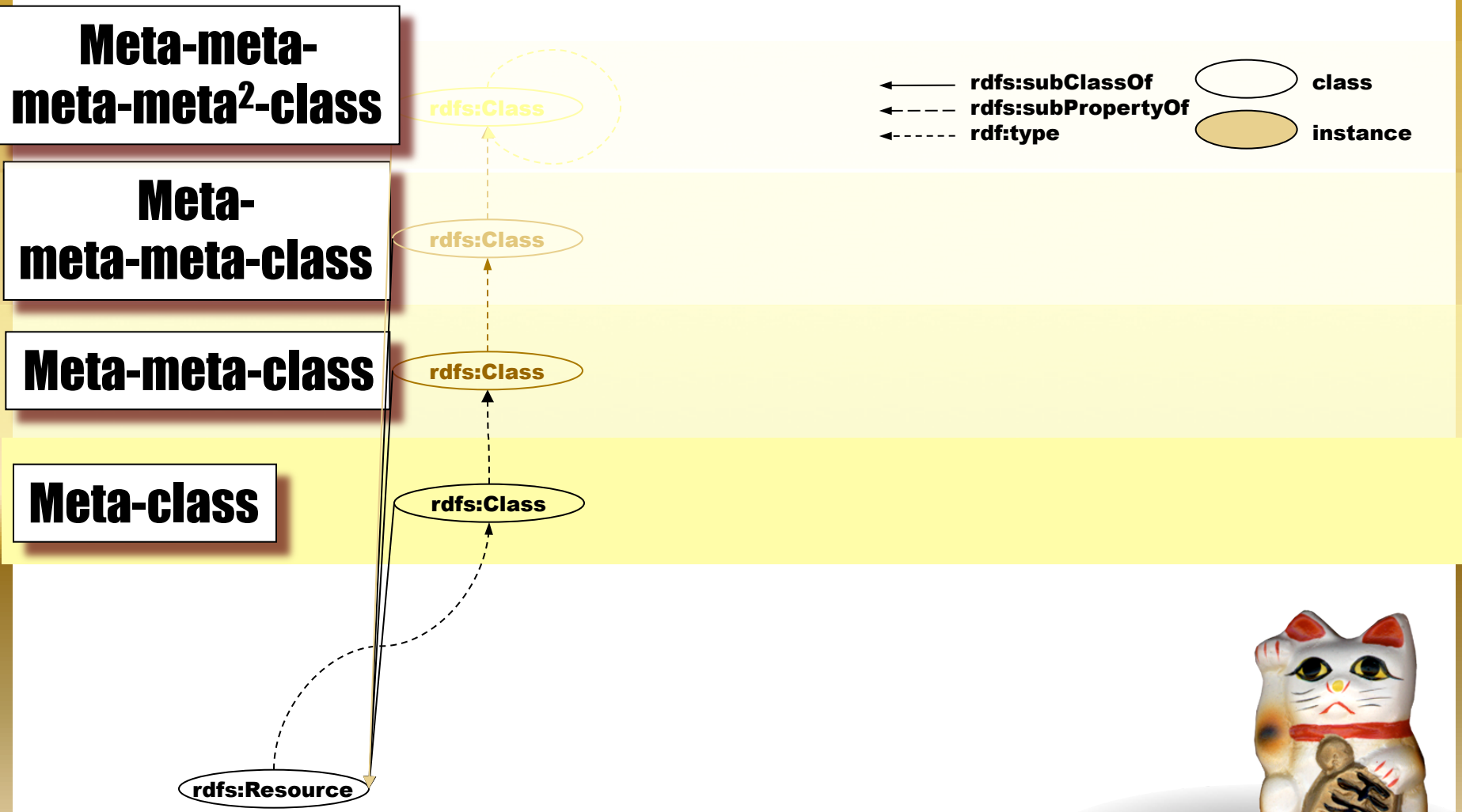
Meta-meta-meta-class

Meta-meta-class

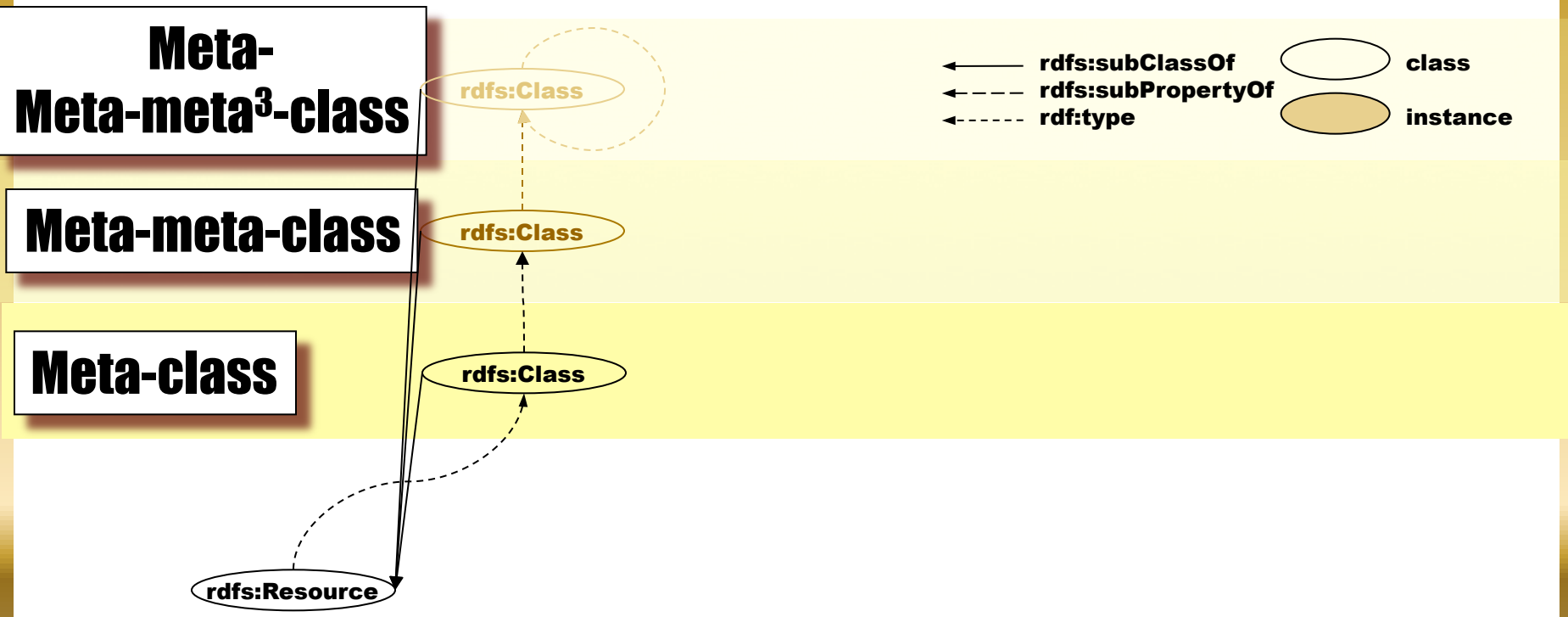
Meta-class



Meta-meta-meta-meta-Class



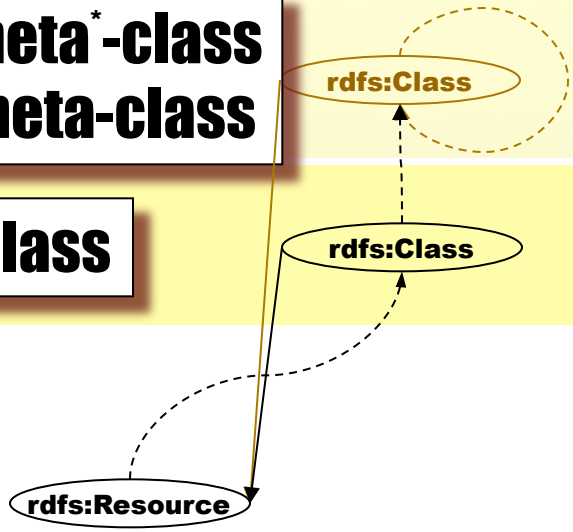
Folded Meta-meta-meta-Class

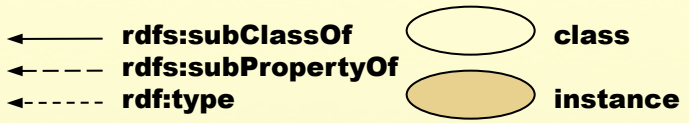


Folded Meta-meta-Class

Meta-meta*-class
Meta-meta-class

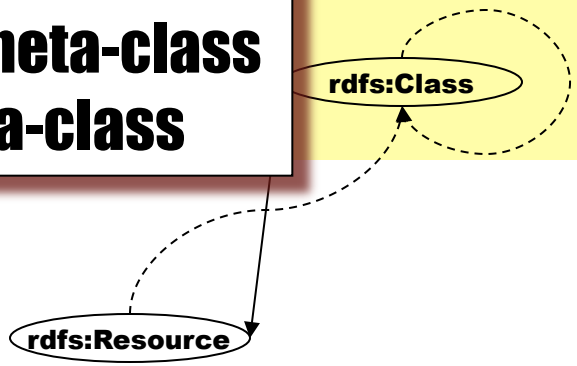
Meta-class



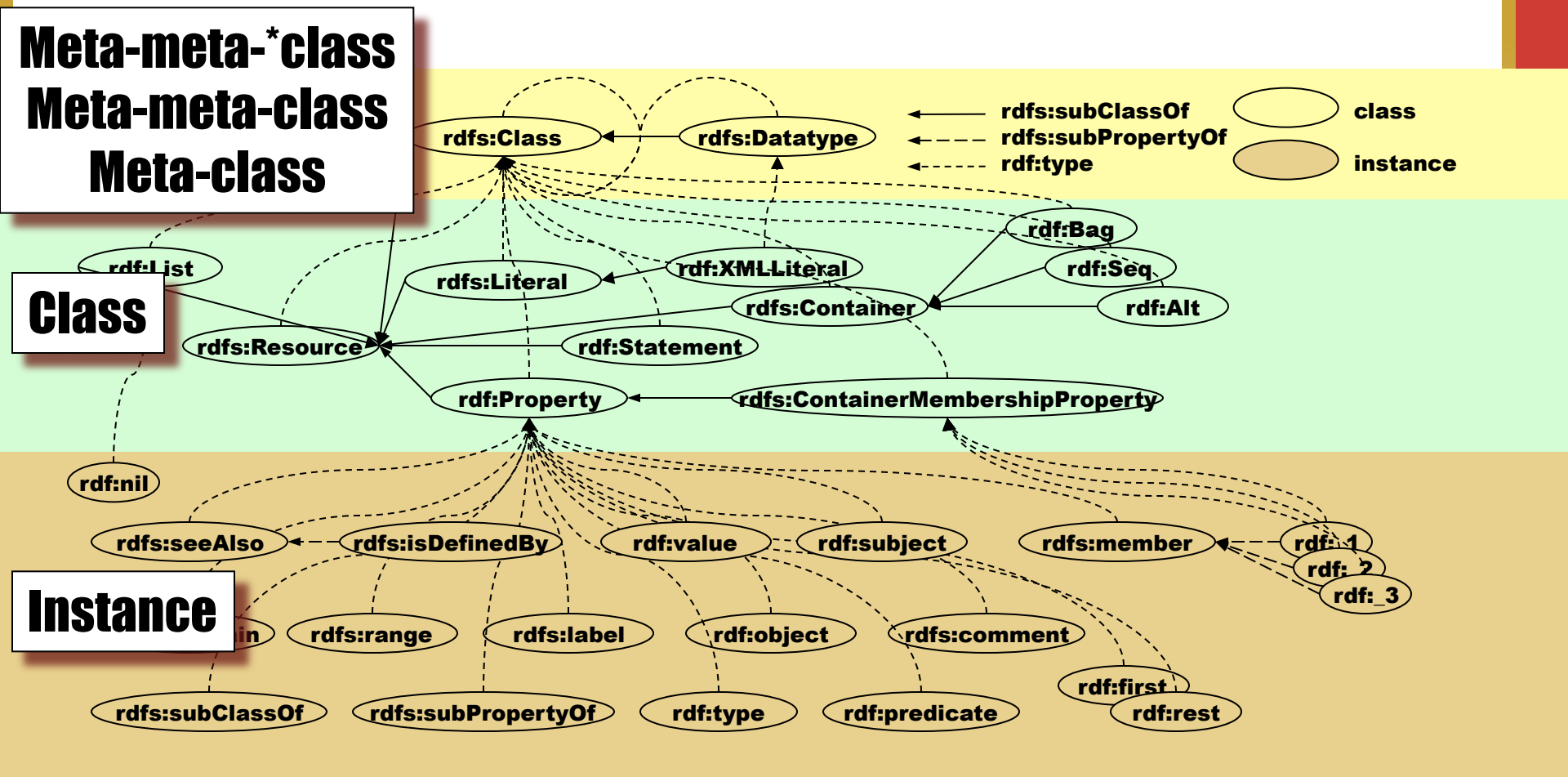


Folded Infinite MetaClass Layers

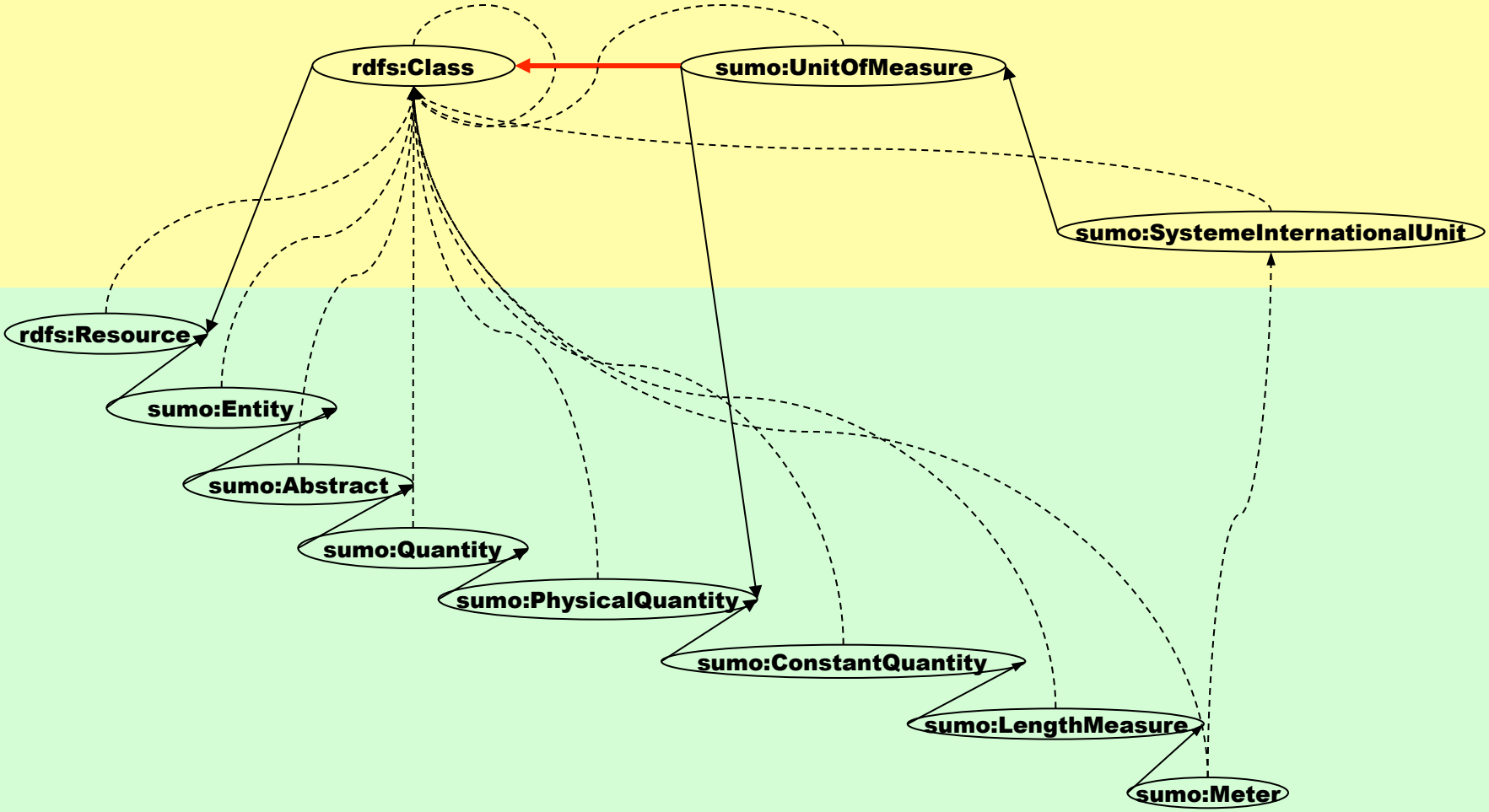
Meta-meta-*class
Meta-meta-class
Meta-class



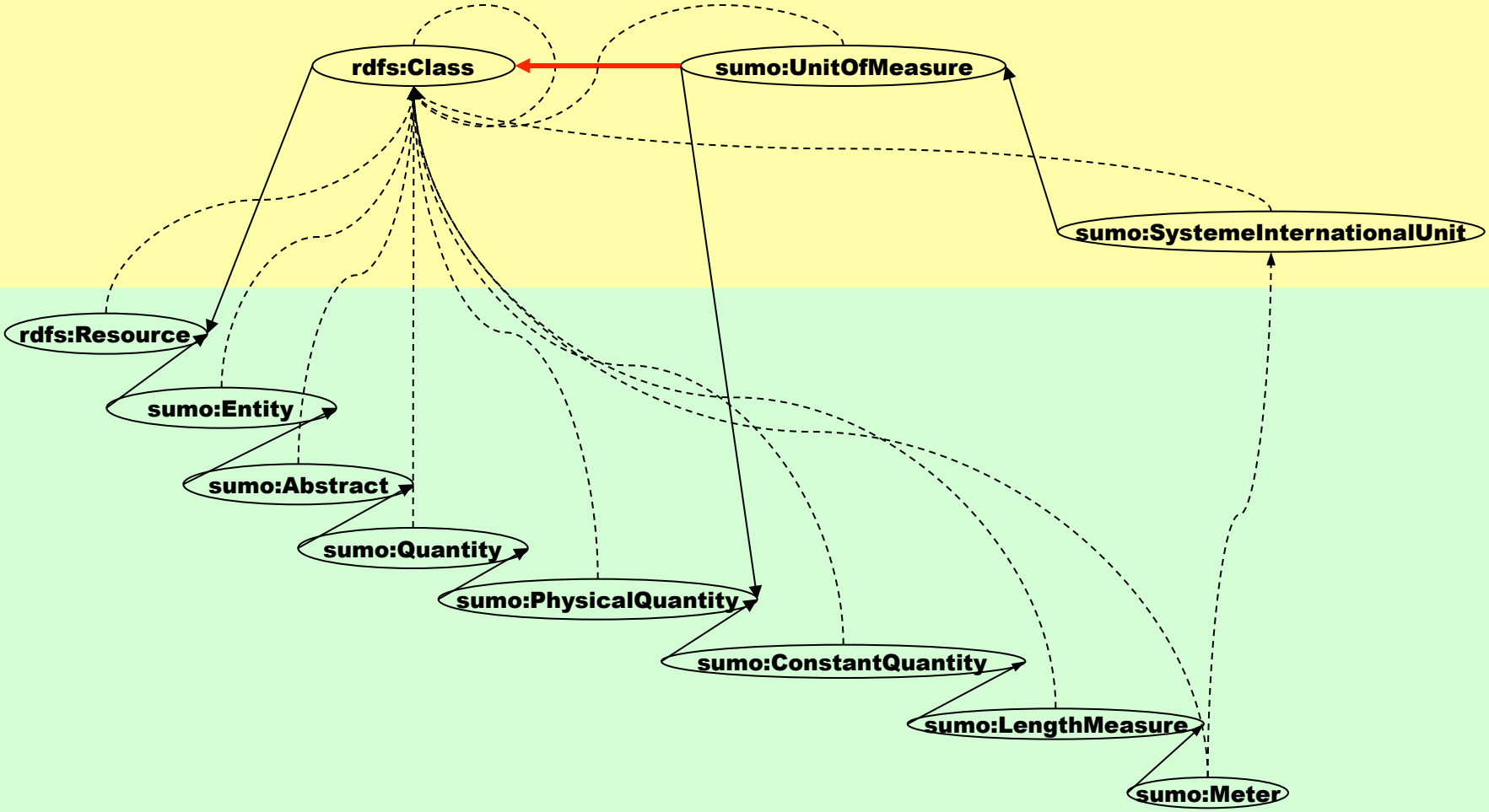
Hierarchical Structure of RDFS



Naive Hierarchical Structure of SUMO



Hierarchical Structure of SUMO



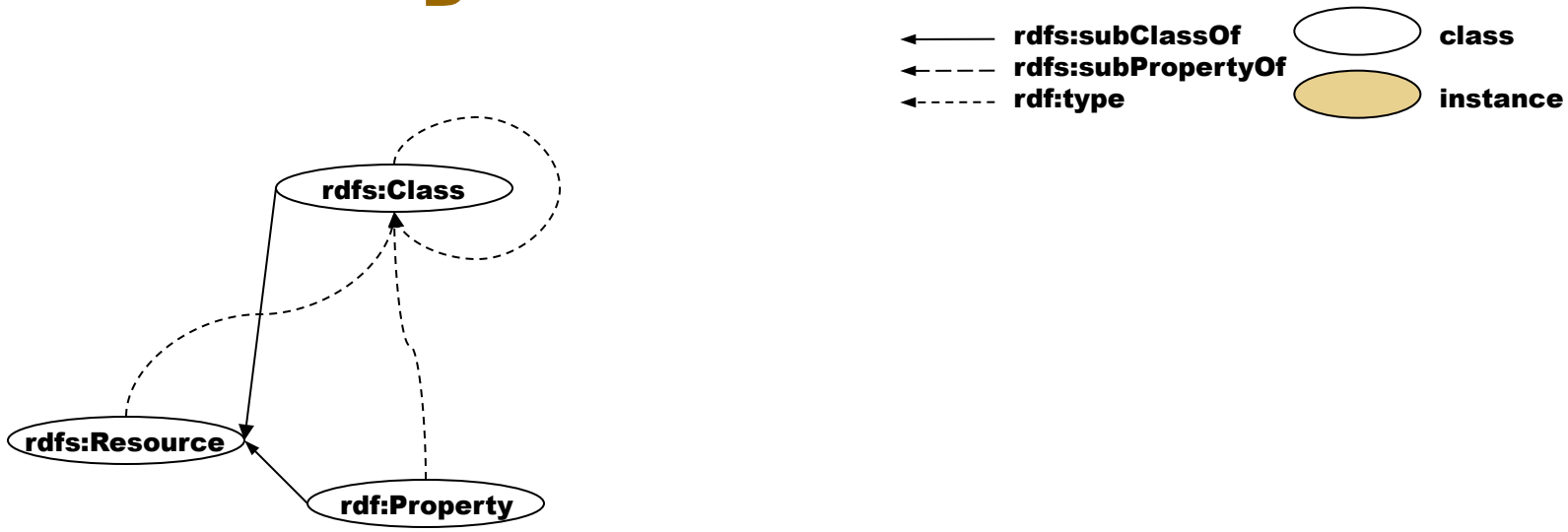
Agenda

- Semantic Gap between SW and OOP
- Metamodeling in RDF Universe
- **How to connect RDF and OWL Universe**
- Extended Structural Algorithm for OWL Subsumption
- OWL-Full Programming by Metamodeling

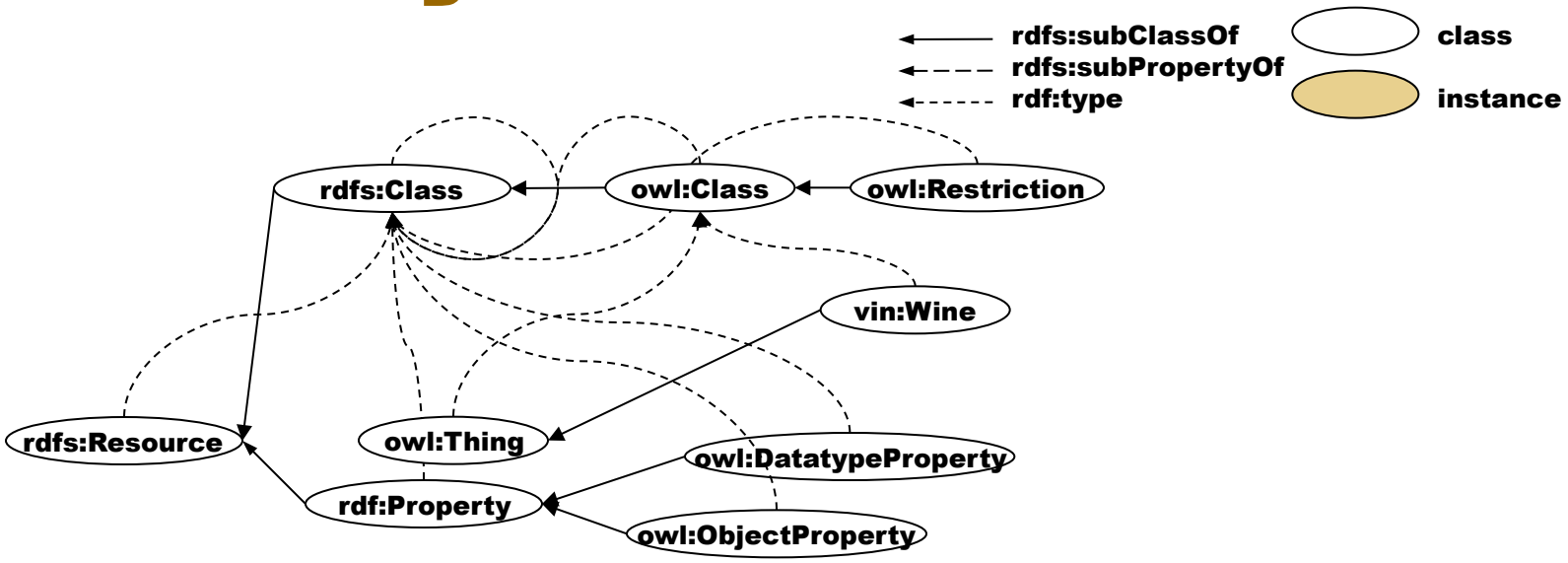
Two Styles of RDFS + OWL

- OWL-Full Style
  - the three parts of the OWL universe are identified with their RDF counterparts, namely the class extensions of rdfs:Resource, rdfs:Class, and rdf:Property.
- OWL-DL Style
  - the three parts are different from their RDF counterparts and, moreover, pairwise disjoint.

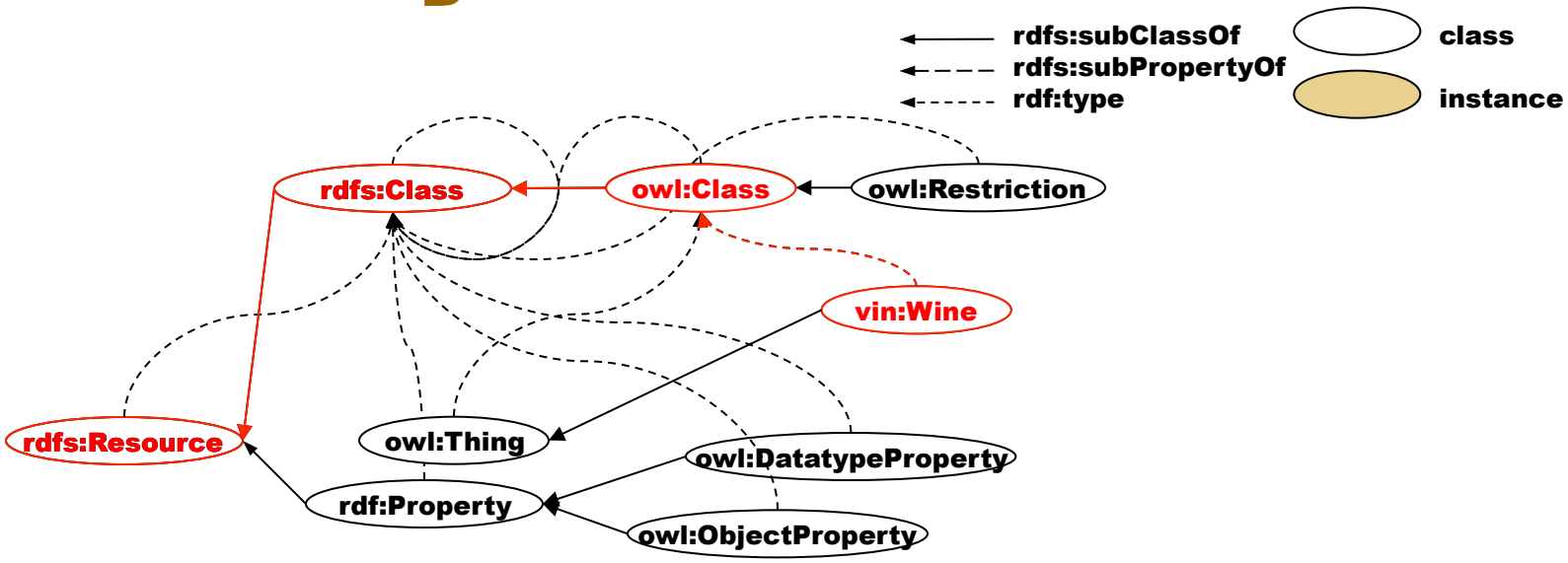
OWL-Full Style Connection



OWL-Full Style Connection

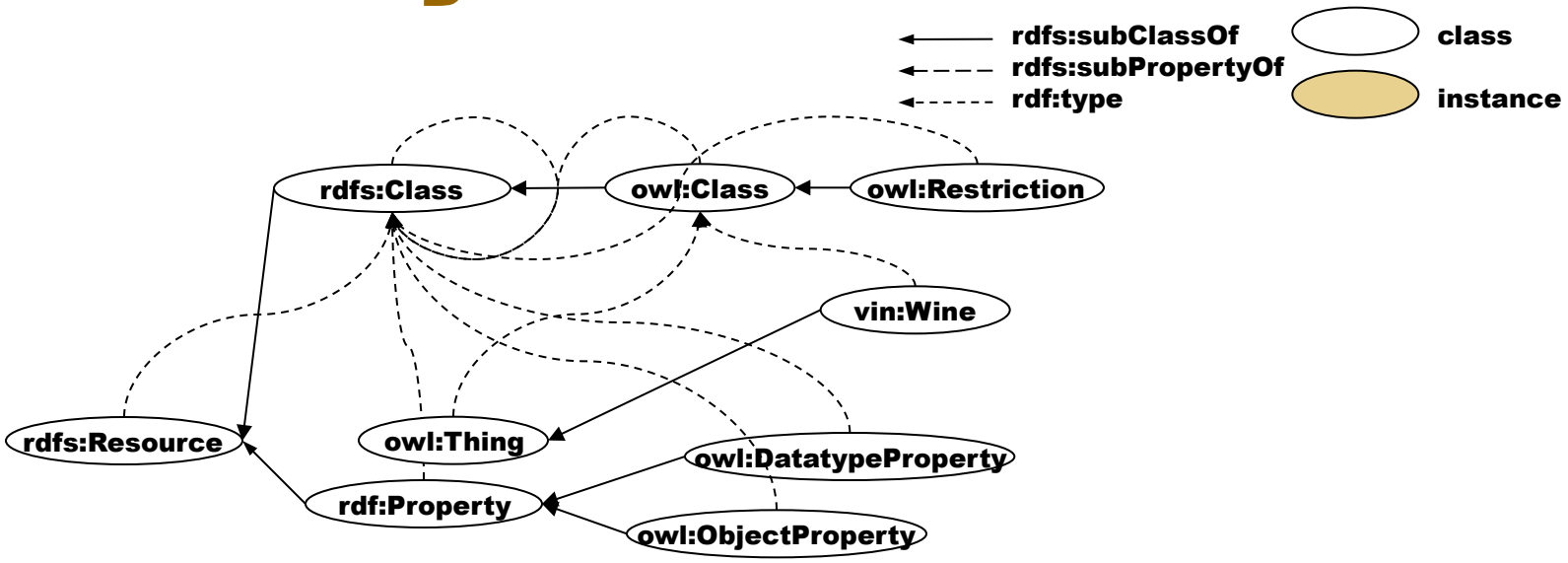


OWL-Full Style Connection

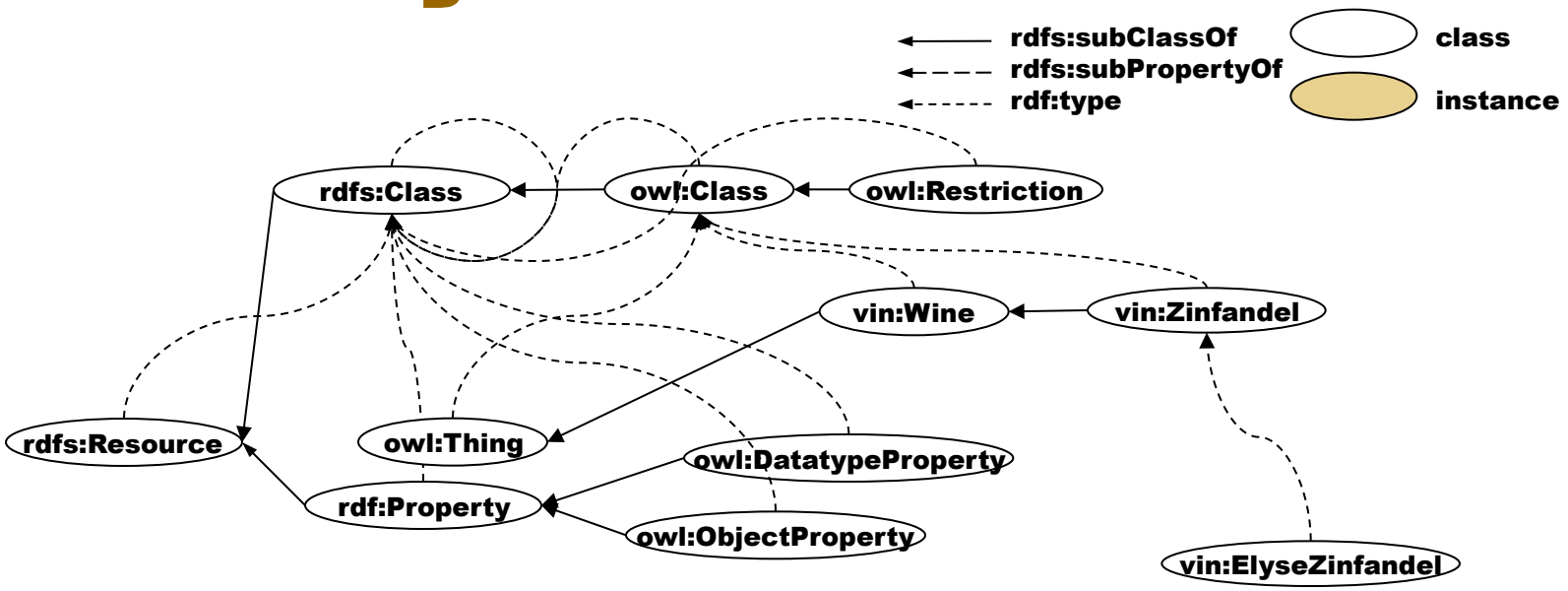


- All classes under owl:Class are in rdfs:Universe

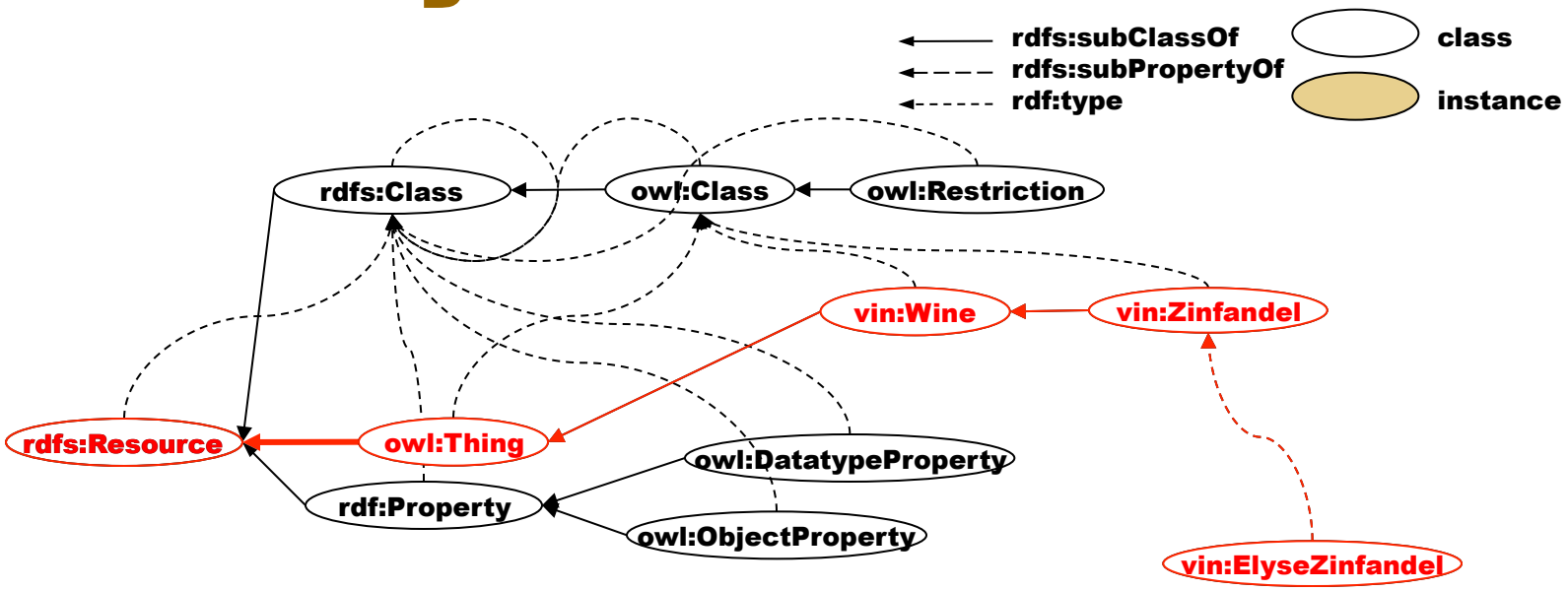
OWL-Full Style Connection



OWL-Full Style Connection

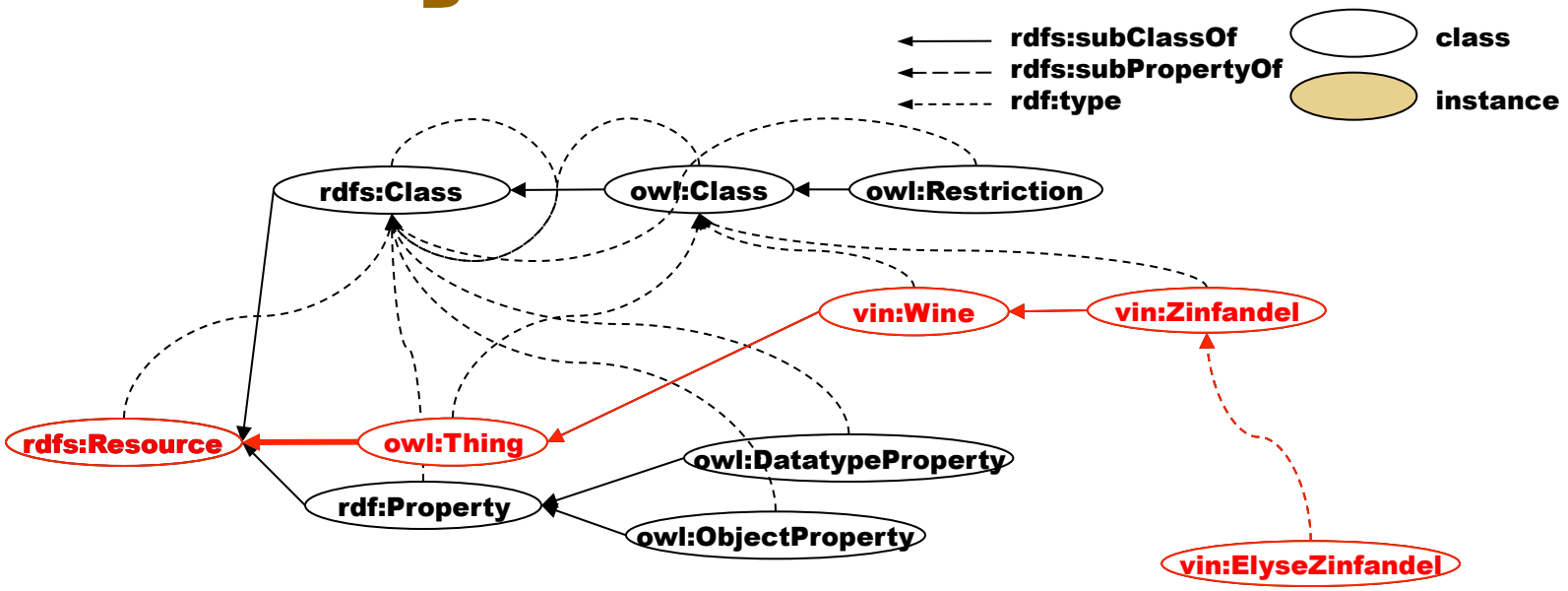


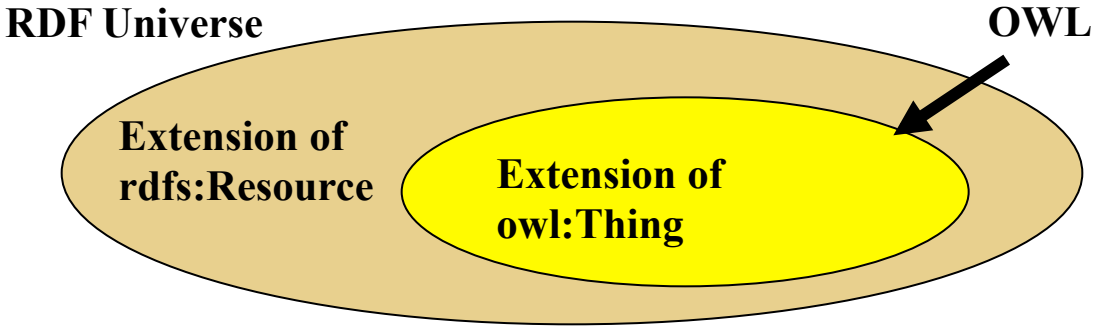
OWL-Full Style Connection



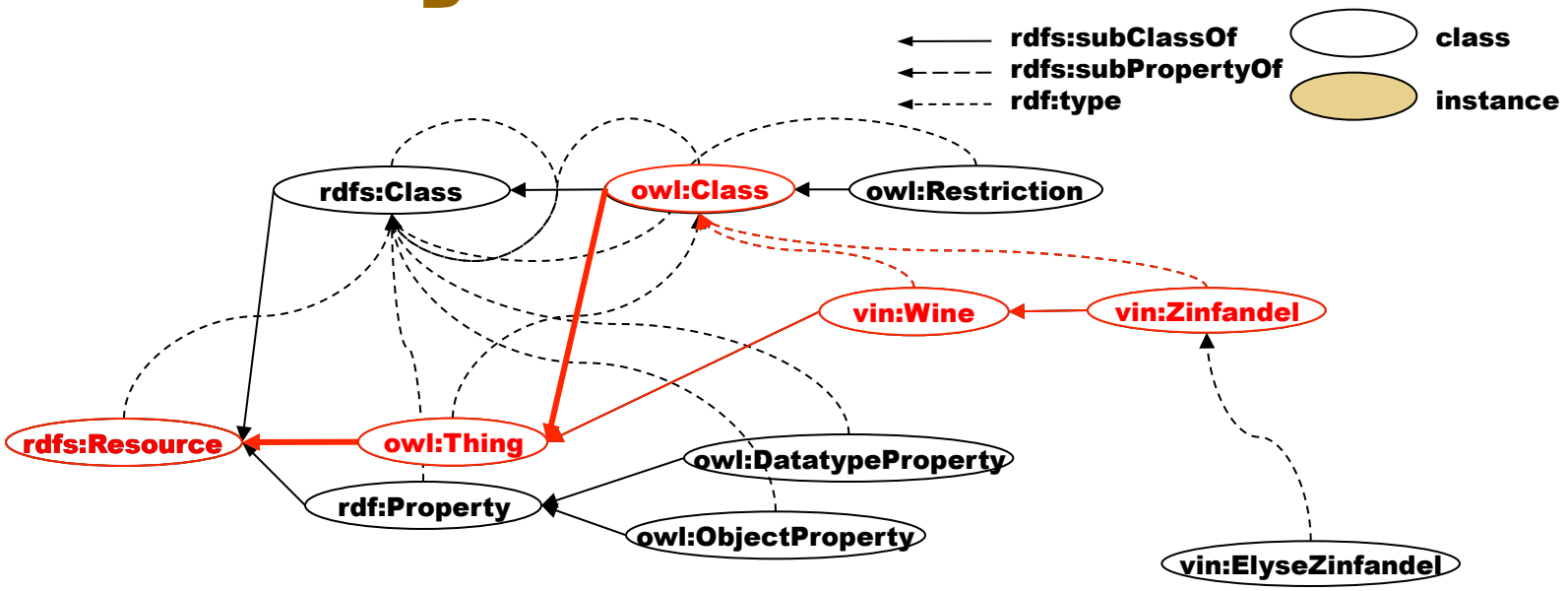
- **We need the new axiom to make OWL individuals be in RDFS universe.**
  - `owl:Thing rdfs:subClassOf rdfs:Resource .`

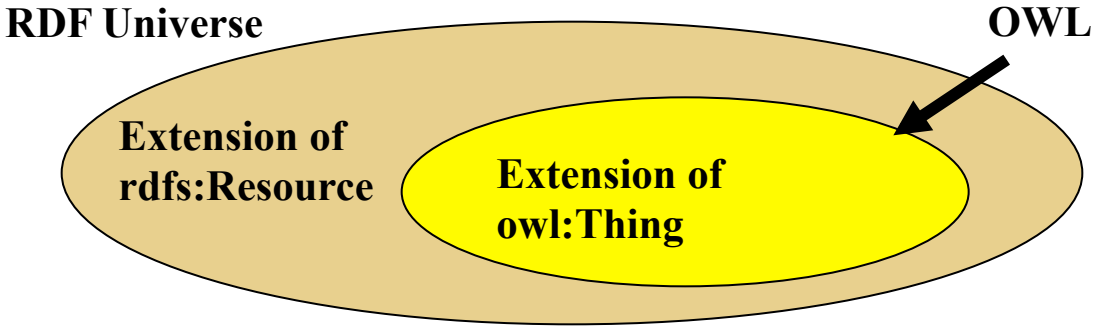
OWL-Full Style Connection



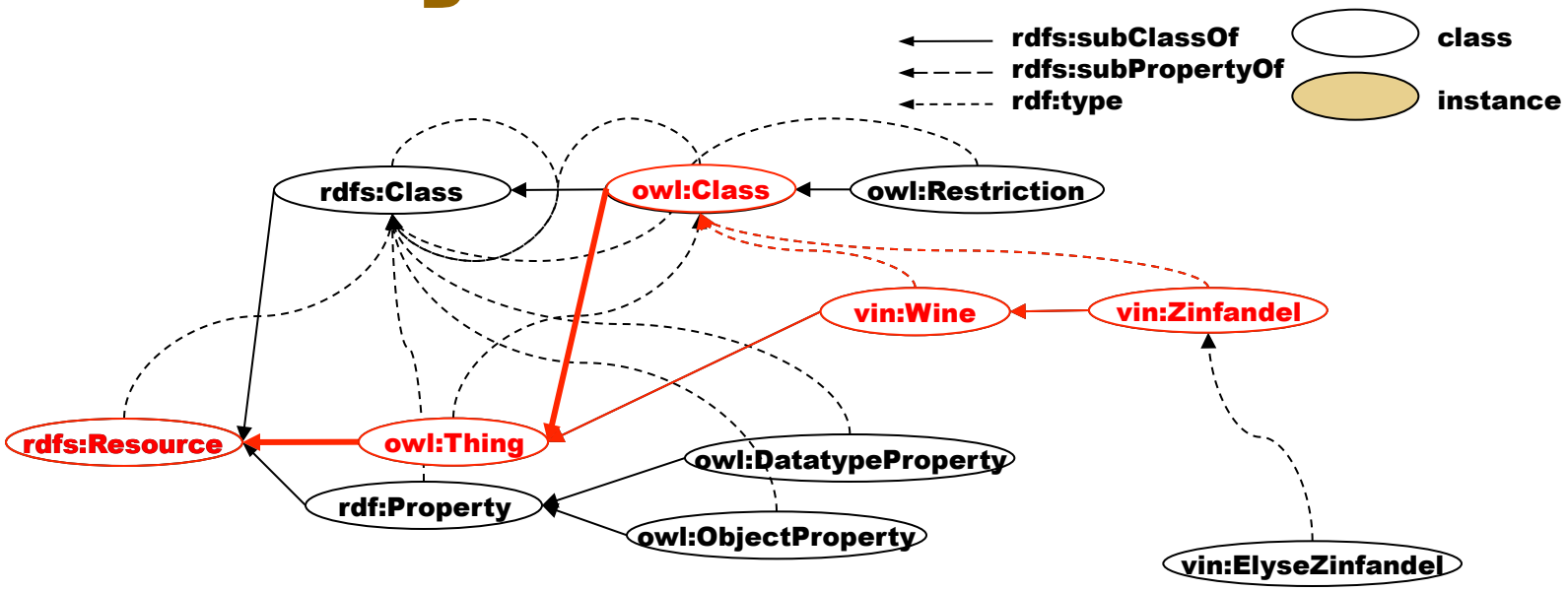


OWL-Full Style Connection



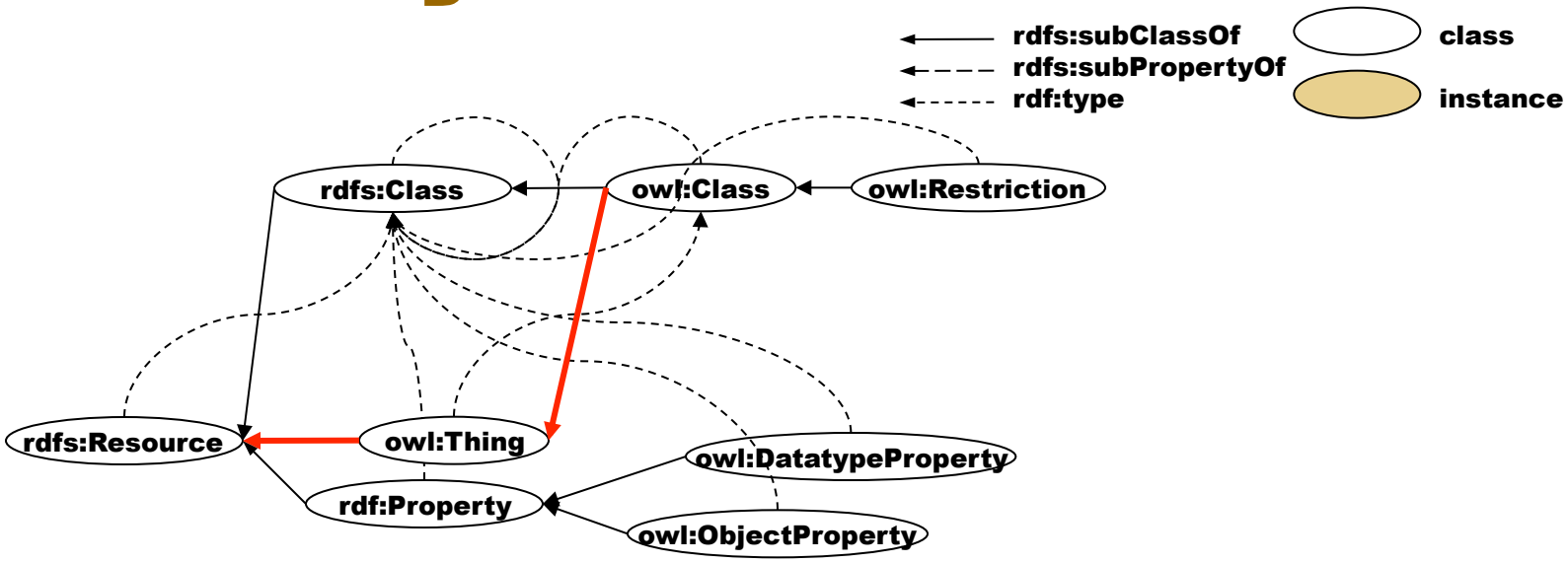


OWL-Full Style Connection



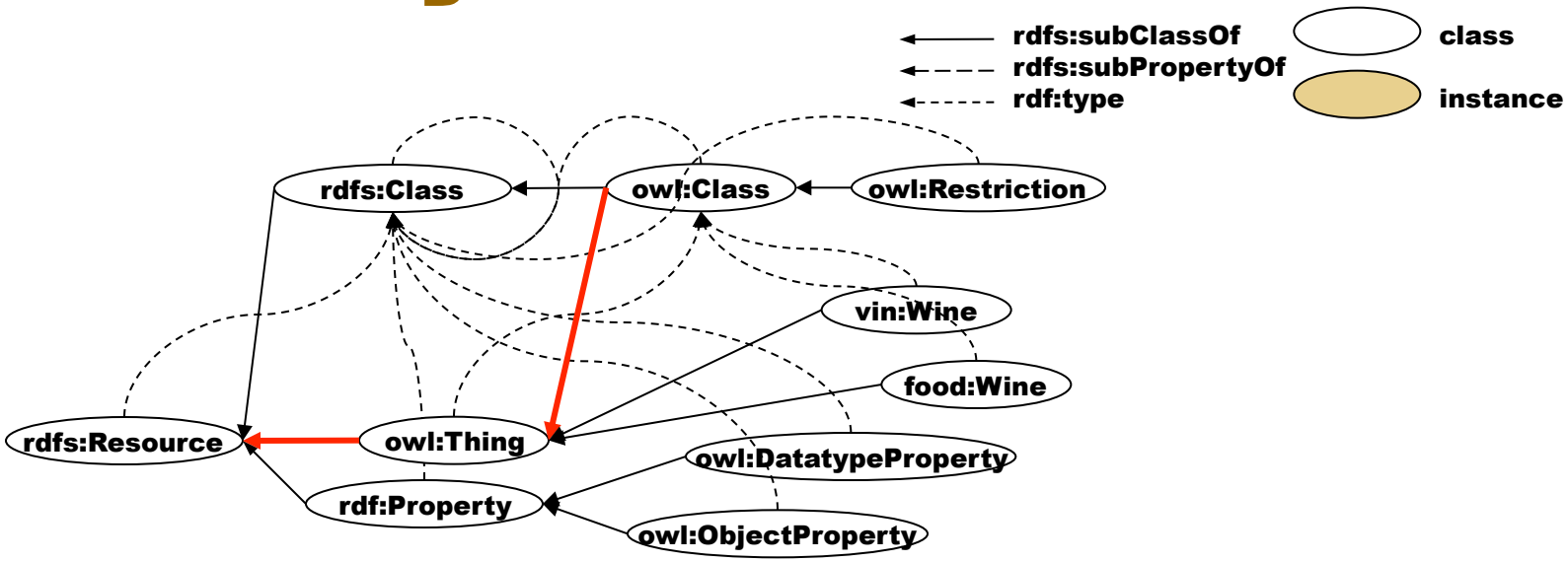
- **We need the new axiom to make OWL classes be in OWL universe.**
  - `owl:Class rdfs:subClassOf owl:Thing .`

OWL-Full Style Connection



- Classes may be treated as individual

OWL-Full Style Connection

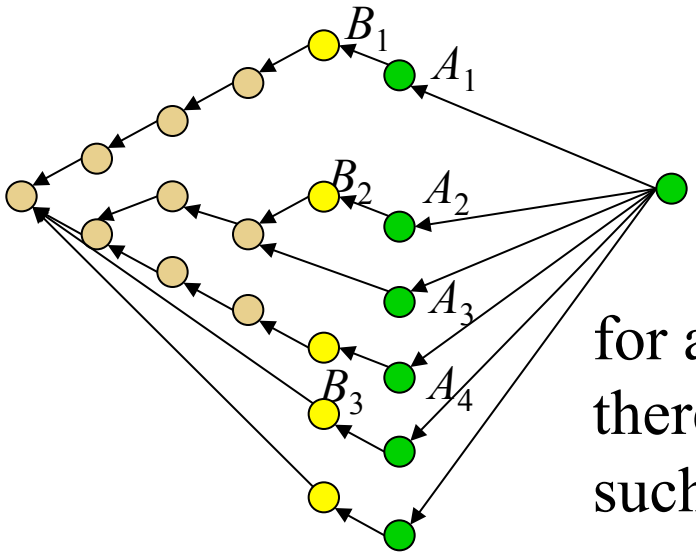


- **Classes may be treated as individual**
  - `vin:Wine owl:sameAs food:Wine .`

Agenda

- Semantic Gap between SW and OOP
- Metamodeling in RDF Universe
- How to connect RDF and OWL Universe
- **Extended Structural Algorithm for OWL Subsumption**
- OWL-Full Programming by Metamodeling

Structural Subsumption Algorithm

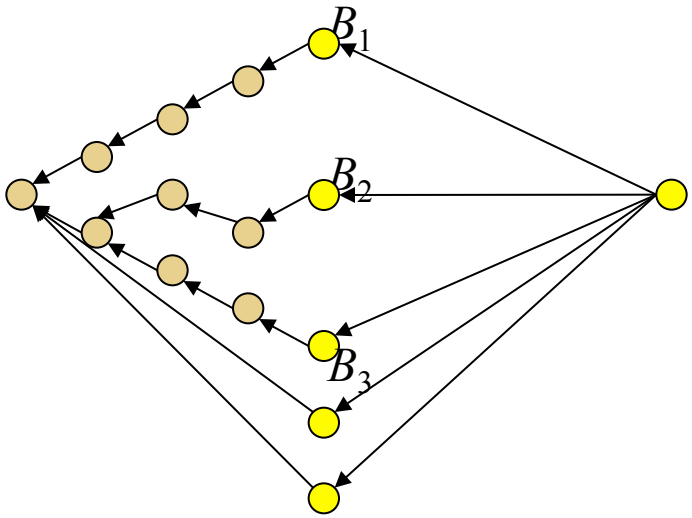


$$C \equiv A_1 \cap A_2 \cap A_3 \cap A_4 \cap \forall R_1.C_1 \cap \cdots \cap \forall R_n.C_n$$

for all B_i ,
there exists A_j
such that $A_j \subseteq B_i$

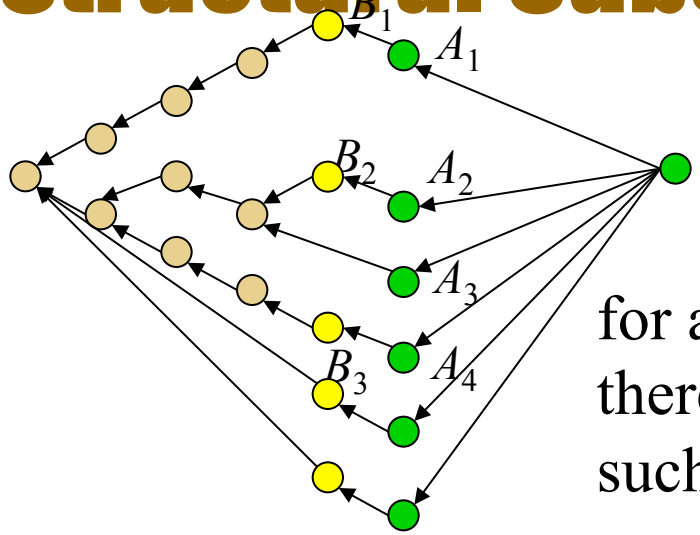
for all D_i ,
there exists C_j
such that $R_j = S_i$, $C_j \subseteq D_i$

then $C \subseteq D$



$$D \equiv B_1 \cap B_2 \cap B_3 \cap \forall S_1.D_1 \cap \cdots \cap \forall S_n.D_n$$

Extended Structural Subsumption Algorithm

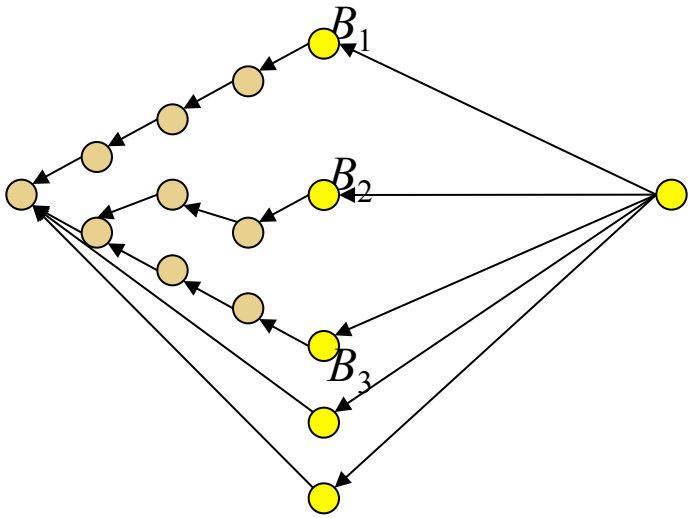


$C \subseteq A_j$, $\forall R_j.C_j$

for all B_i ,
there exists A_j
such that $A_j \subseteq B_i$

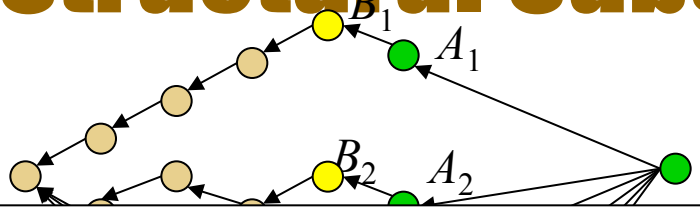
for all D_i ,
there exists C_j
such that $R_j = S_i$, $C_j \subseteq D_i$

then $C \subseteq D$



$$D \equiv B_1 \cap B_2 \cap B_3 \cap \forall S_1.D_1 \cap \cdots \cap \forall S_n.D_n$$

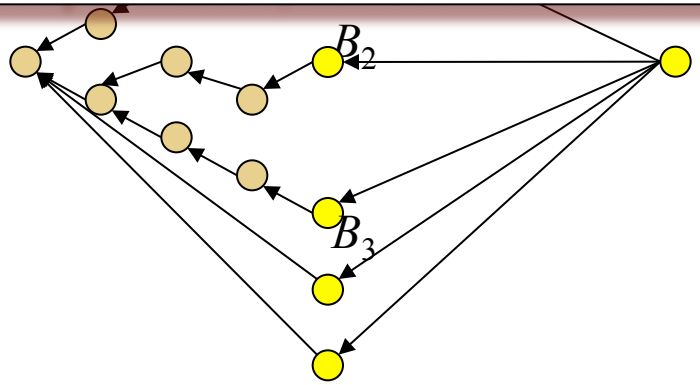
Extended Structural Subsumption Algorithm



$C \subseteq A_j$, $\forall R_j.C_j$

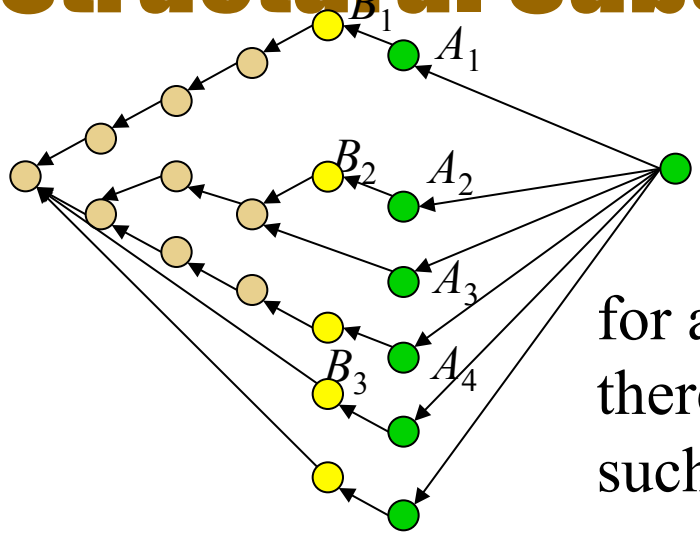
TYPO page 271 item6
Collect all substantially subsuming concepts and restrictions (all of CLOS Superclasses) for each C and each D, instead of C and D, and do the stru-

Correct
Collect all substantially subsuming concepts and restrictions (all of CLOS Superclasses) for each C and each intersections of D, instead of C and D, and do the



$D \equiv B_1 \sqcap B_2 \sqcap B_3 \sqcap$
$\forall S_1.D_1 \sqcap \cdots \sqcap \forall S_n.D_n$

Extended Structural Subsumption Algorithm

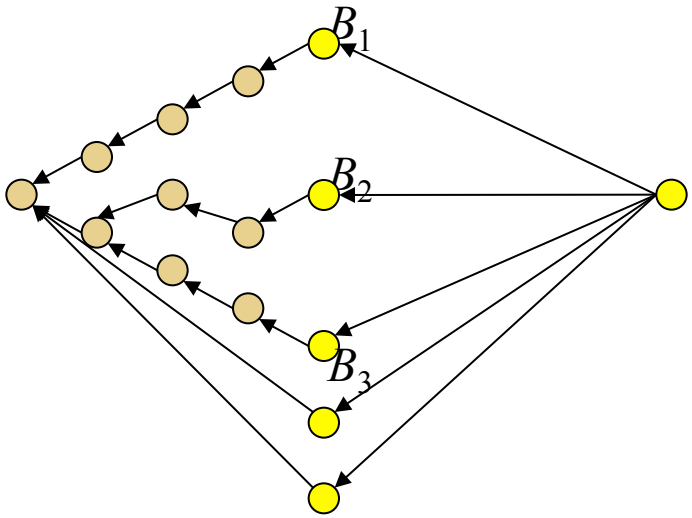


$C \subseteq A_j$, $\forall R_j.C_j$

for all B_i ,
there exists A_j
such that $A_j \subseteq B_i$

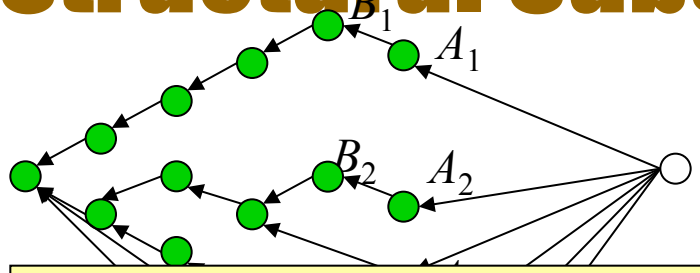
for all D_i ,
there exists C_j
such that $R_j = S_i$, $C_j \subseteq D_i$

then $C \subseteq D$



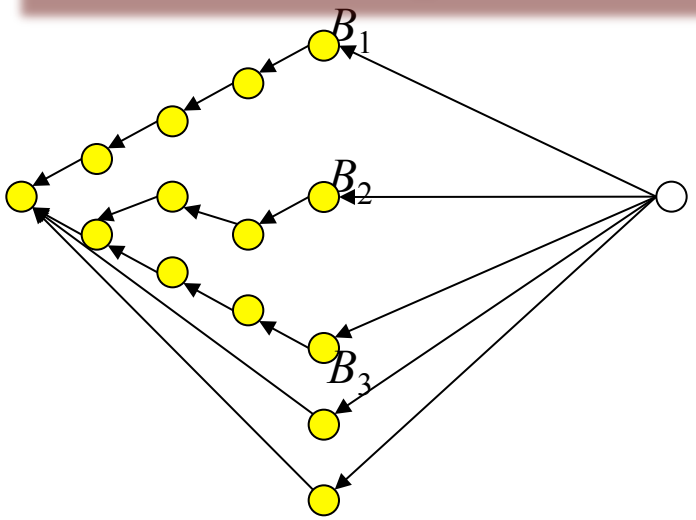
$$D \equiv B_1 \cap B_2 \cap B_3 \cap \forall S_1.D_1 \cap \cdots \cap \forall S_n.D_n$$

Extended Structural Subsumption Algorithm



$C \subseteq A_j$, $\forall R_j.C_j$
pick up all supers of C

For intersection and allValuesFrom it is complete. But, . . .



then $C \subseteq D$

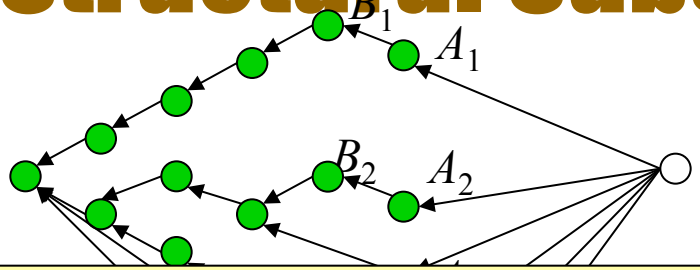
$D \equiv B_1 \cap B_2 \cap B_3 \cap$
$\forall S_1.D_1 \cap \cdots \cap \forall S_n.D_n$
pick up all supers of
Intersections of D

Extended Structural Subsumption Algorithm

- $\forall R_j.C_j$ vs. $\forall S_i.D_i$, if $R_j = S_i, C \subseteq D$ - complete
- $\exists R_j.C_j$ vs. $\exists S_i.D_i$, if $R_j = S_i, C \subseteq D$ - incomplete
- $\forall R_j.C_j$ vs. $\exists S_i.D_i$, if $R_j = S_i, C \subseteq D$ - incomplete
- $\exists R_j.C_j$ vs. $\forall S_i.D_i$, if $R_j = S_i, C \subseteq D$ - incomplete
- $R_j{:}a$ vs. $S_i{:}b$, if $R_j = S_i, a \subseteq b$ - complete
- $R_j{:}a$ vs. $\forall S_i.D_i$, if $R_j = S_i, a \in D$ - complete
- $R_j{:}a$ vs. $\exists S_i.D_i$, if $R_j = S_i, a \in D$ - overestimate

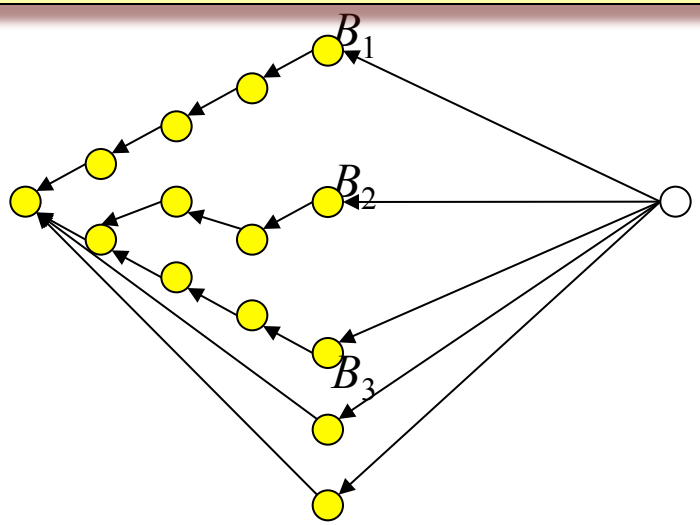
Extended Structural Subsumption Algorithm

to Equivalent Class



$\widetilde{C} \subseteq A_j$, $\forall R_j.C_j$
pick up all supers of $\widetilde{C}$

Instead of C and D,
test all members of equivalent classes



then $C \subseteq D$

$\widetilde{D} \equiv B_1 \cap B_2 \cap B_3 \cap$
$\forall S_1.D_1 \cap \cdots \cap \forall S_n.D_n$
pick up all supers of
Intersections of $\widetilde{D}$

Extended Structural Subsumption Algorithm

to Equivalent Property

- $\forall \overline{R_j}.C_j$ vs. $\forall \overline{S_i}.D_i$, if $\overline{R_j} = \overline{S_i}, C \subseteq D$ - complete

Instead of R and S,
test equivalency of property
with members of the equivalent properties

- $R_j{:}a$ vs. $\forall S_i.D_i$, if $R_j = S_i, a \in D$ - complete
- $\overline{R_j}{:}a$ vs. $\exists \overline{S_i}.D_i$, if $\overline{R_j} = \overline{S_i}, a \in D$ - overestimate

Extended Structural Subsumption Algorithm

to Equivalent Property

- $\forall \overline{R}_j.C_j$ vs. $\forall \overline{S}_i.D_i$, if $\overline{R}_j = \overline{S}_i, C \subseteq D$ - complete
- $\exists \overline{R}_j.C_j$ vs. $\exists \overline{S}_i.D_i$, if $\overline{R}_j = \overline{S}_i, C \subseteq D$ - incomplete
- $\forall \overline{R}_j.C_j$ vs. $\exists \overline{S}_i.D_i$, if $\overline{R}_j = \overline{S}_i, C \subseteq D$ - incomplete
- $\exists \overline{R}_j.C_j$ vs. $\forall \overline{S}_i.D_i$, if $\overline{R}_j = \overline{S}_i, C \subseteq D$ - incomplete
- $\overline{R}_j{:}a$ vs. $\overline{S}_i{:}b$, if $\overline{R}_j = \overline{S}_i, a \subseteq b$ - complete
- $\overline{R}_j{:}a$ vs. $\forall \overline{S}_i.D_i$, if $\overline{R}_j = \overline{S}_i, a \in D$ - complete
- $\overline{R}_j{:}a$ vs. $\exists \overline{S}_i.D_i$, if $\overline{R}_j = \overline{S}_i, a \in D$ - overestimate

Extended Structural Subsumption Algorithm

- Finally, we test subsumption in individual level

sameAs

If $\dot{C} \doteq \dot{D}$, then $C = D$

If $C \sqsubseteq D$, then $C = D$

Transitive-lower for transitive properties

Efficiency of Computation

- Allegro Common Lisp 8.0 on MS-Window 2000
- Pentium 4 (2.6GHz), 1 GB RAM
- Loading time of Food & Wine Ontology = 2 sec
- Lehigh University Benchmark (LUBM)
  - For LUBM(1,0)
    - 8330 persons + 1627 courses + 15 departments + 979 universities
  - Loading time = 4 min 14 sec

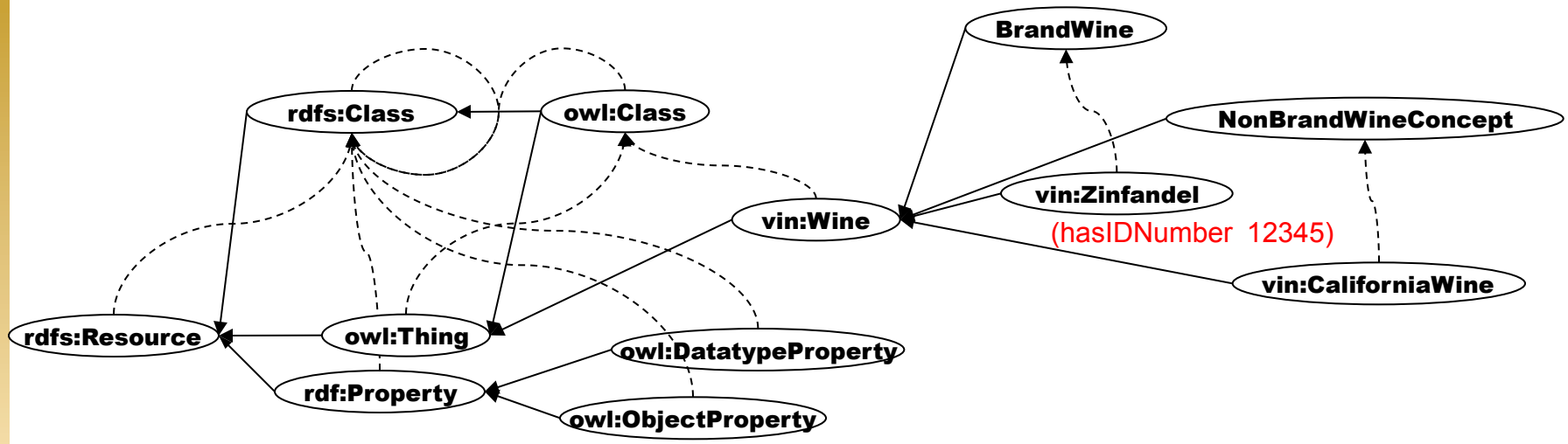
Efficiency of Computation (LUBM(1,0))

Query		DLDB-OWL	Sesame- DB	Sesame- Memory	OWL JessKB-P	OWL JessKB- NP	SWCLOS
1	Time(ms)	59	46	15	9203	200	**63**
2	Time(ms)	181	51878	87	116297	3978	**33640**
3	Time(ms)	218	40	0	13990	164	**125**
4	Time(ms)	506	768	6	211514	8929	**297**
5	Time(ms)	617	2945	17	5929	475	**15140**
6	Time(ms)	481	253	48	1271	112	**328**
	Answers	7790	5916	5916	7790	7790	**7790**
7	Time(ms)	478	603	3	128115	67	**875**
	Answers	67	59	59	67	67	**67**
8	Time(ms)	765	105026	273	164106	4953	**198313**
	Answers	7790	5916	5916	7790	7790	**7790**
9	Time(ms)	634	34034	89	87475	2525	**766**
	Answers	208	103	103	208	208	**208**
10	Time(ms)	98	20	1	141	4	**453**
	Answers	4	0	0	4	4	**4**
11	Time(ms)	48	65	1	1592	45	**62**
	Answers	0	0	0	224	224	**224**
12	Time(ms)	62	4484	12	11266	162	**297**
	Answers	0	0	0	15	15	**15**
13	Time(ms)	200	4	1	90	1	**2500**
	Answers	0	0	0	1	1	**1**
14	Time(ms)	187	218	42	811	20	**16**

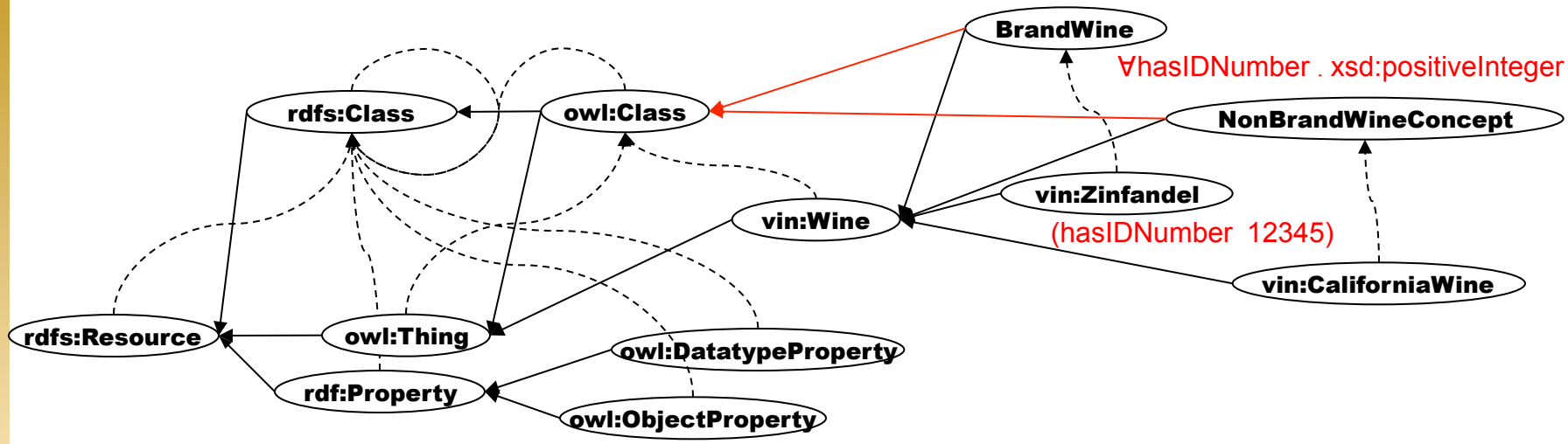
Agenda

- Semantic Gap between SW and OOP
- Metamodeling in RDF Universe
- How to connect RDF and OWL Universe
- Extended Structural Algorithm for OWL Subsumption
- **OWL-Full Programming by Metamodeling**

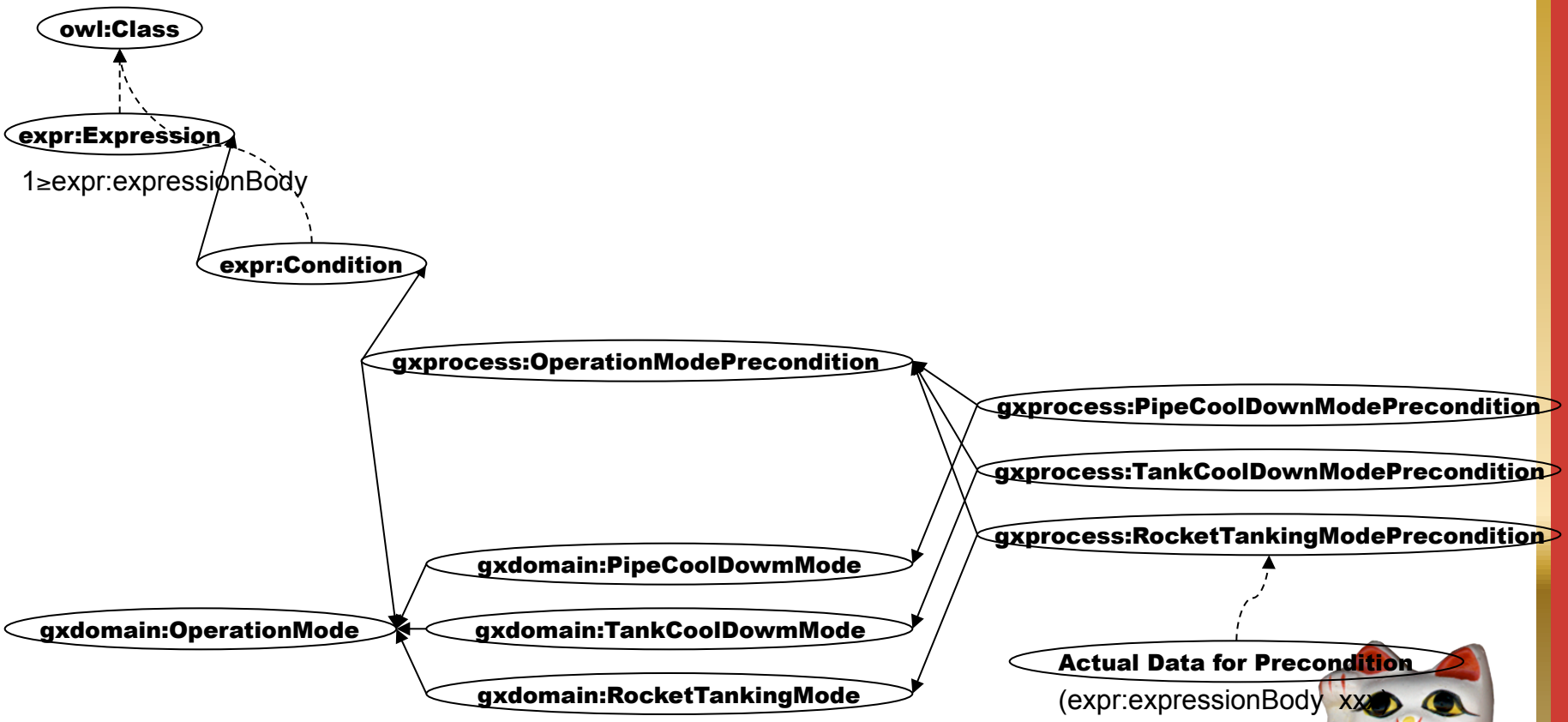
OWL-Full Programming
Meta-class for Role & Filler



OWL-Full Programming
Meta-class for Role & Filler



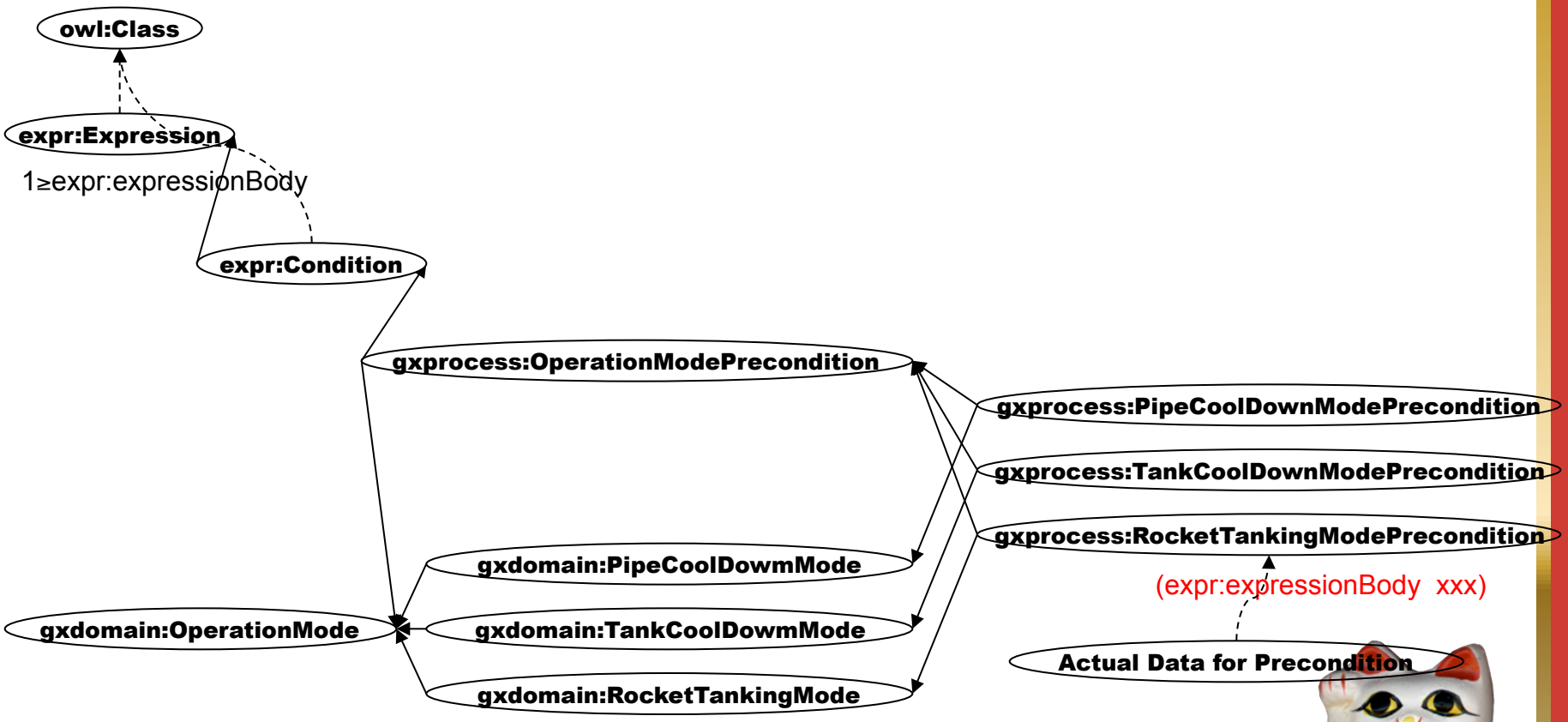
OWL-Full Programming
Treatment of Instance as Class



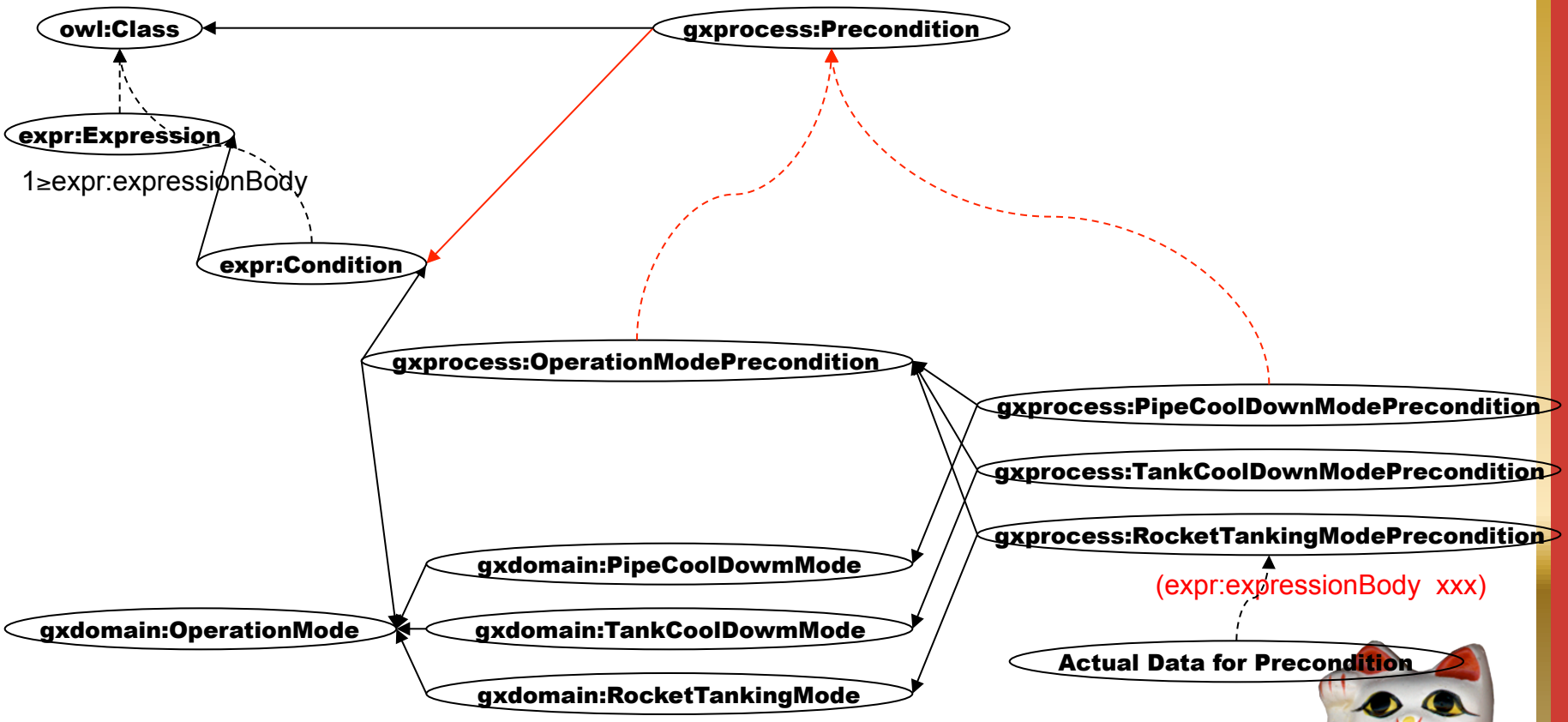
情報・システム研究機構
国立情報学研究所
National Institute of Informatics

GALAXY EXPRESS

OWL-Full Programming
Treatment of Instance as Class



OWL-Full Programming
Treatment of Instance as Class



Conclusions

- OWL Reasoning is implemented on top of dynamic Object Oriented Language, CLOS.
- OWL-Full Performance is obtained by Object-Oriented Metamodeling.
- Extended Structural Subsumtion Algorithm is programmed.
- However, it is still incomplete on someValuesFrom.
- OWL universe is connected to RDF universe in OWL-Full style.