

# Knowledge Representation

Session in the course  
“Programmation Logique et Connaissances”  
at the École nationale supérieure des Télécommunications  
in Paris/France in spring 2011

by Fabian M. Suchanek



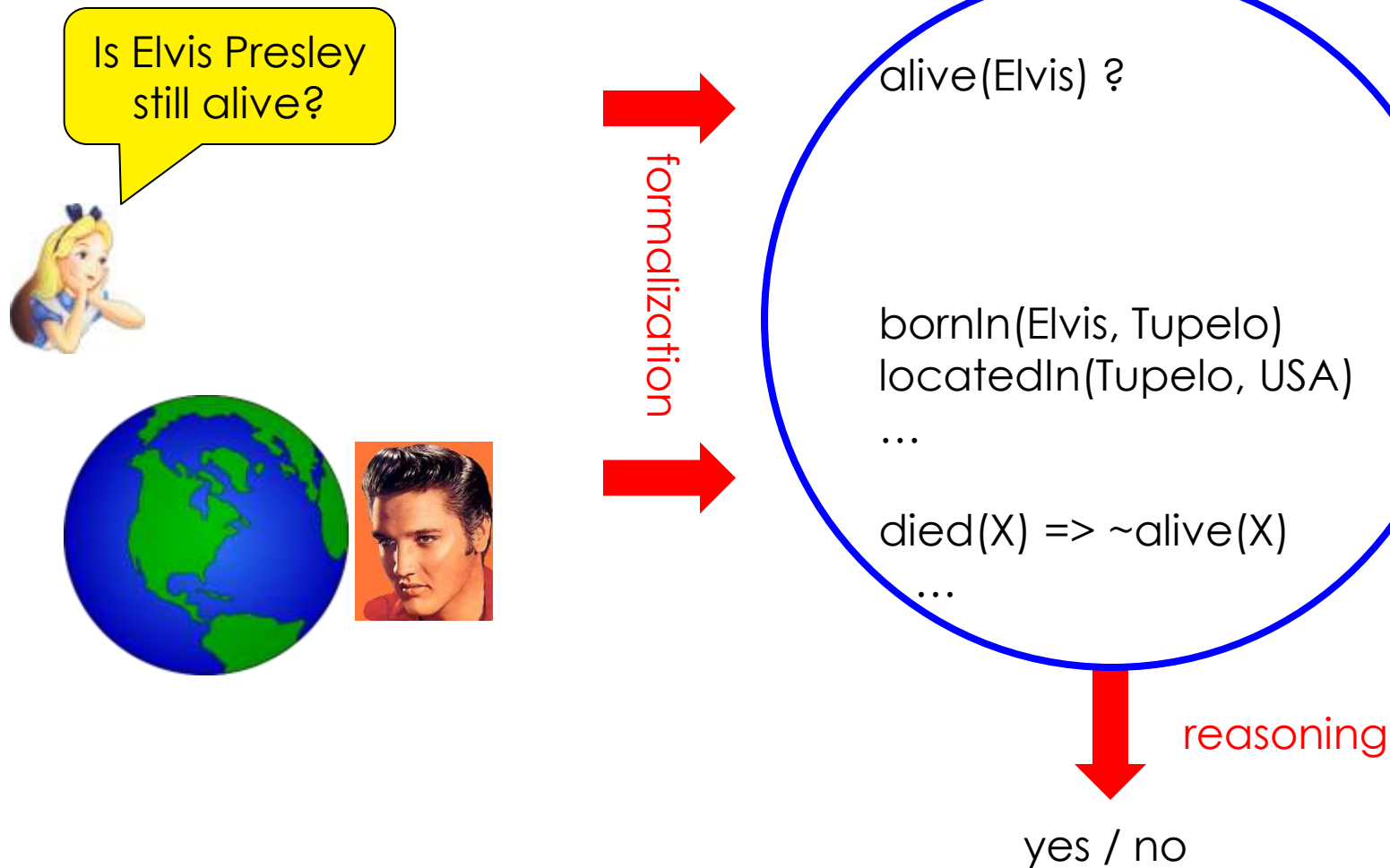
This document is available under a  
[Creative Commons Attribution Non-Commercial License](https://creativecommons.org/licenses/by-nc/4.0/)

# Organisation

- 3h class on Knowledge Representation  
15min break
- Slides are available at the Web-sites  
<http://suchanek.name/> → Teaching  
<http://icc.enst.fr/PLC> → Session of 2011-01-10
- Language of the Slides is English,  
lecturer will do his best to talk in French,  
questions are welcome throughout the session in French or English

# Motivation

**Knowledge representation** (KR) is a research field of artificial intelligence that aims to formalize information so that it can be used for automated reasoning.



# Motivation

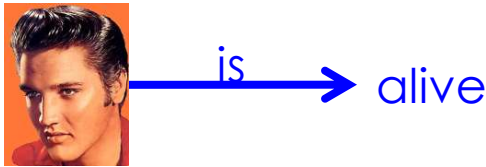
**Knowledge representation** (KR) is a research field of artificial intelligence that aims to formalize information so that it can be used for automated reasoning.

There are different ways to represent knowledge:

- PROLOG-like:

`alive(Elvis), is(Elvis, alive)`

- graphically



- in natural language

“Elvis is alive”

- in propositional logic

`elvis_alive .`

- in first order logic

$\forall x: \text{rocksinger}(x) \Rightarrow \text{alive}(x)$

- in a mathematical notation

`elvis  $\in$  Alive`

- in a programming language

`elvis.alive=true`

- in completely different formalism

`$\wp$  elvis  $\rightarrow$  ☺ ☺ ☺`

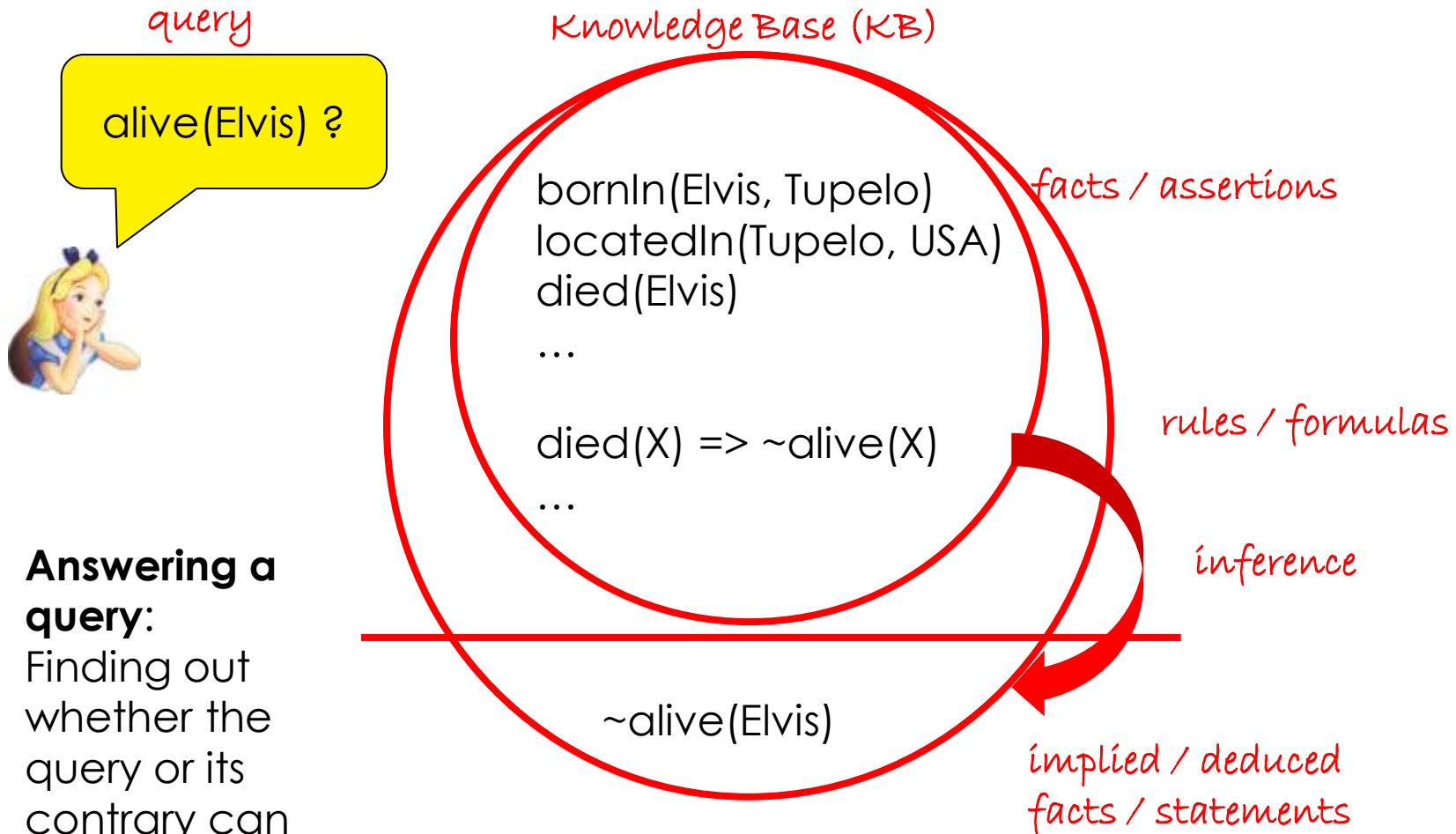
# Overview

- Motivation

- • Knowledge Representation Design

- Knowledge Representation in the Semantic Web  
(RDF, RDFS, OWL, DL)

# KR Design: Big picture



## Answering a query:

Finding out whether the query or its contrary can be deduced from the KB

**KR formalism:** The language of the KB

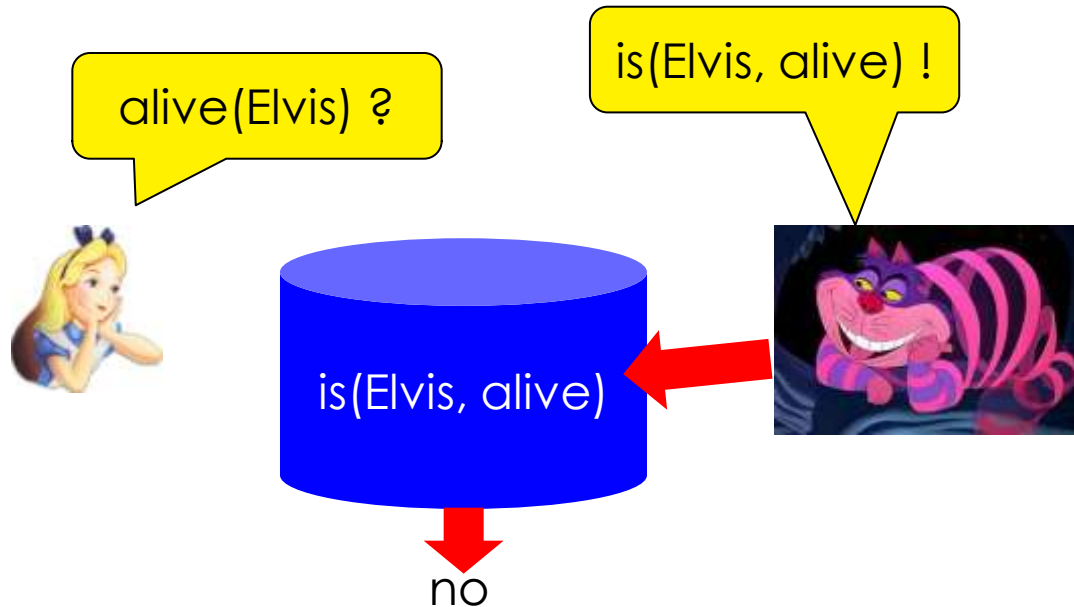
# KR Design: Canonicity

A KR formalism is **canonic** if one piece of knowledge can only be represented in one way

alive(Elvis)  
is(Elvis, alive)  
alive(Elvis, true)  
vivant(Elvis)

} not very canonic

Canonicity is in general desirable because it facilitates co-operation by different people



# KR Design: Canonicity

A KR formalism is **canonic** if one piece of knowledge can only be represented in one way.

A formalism can be made more canonic by

- restricting it

e.g., by allowing only unary predicates:

alive(Elvis)

~~is(Elvis, alive)~~

- providing best practice guidelines

e.g., by prescribing certain conventions:

alive(Elvis)

~~alice(elvis)~~

- providing standard vocabularies

e.g., by listing predicates that should be used in a certain domain:

{alive, dead, young, old, happy, sad}

# KR Design: Expressiveness

A KR formalism is **more expressive** than another one if we can say things in the first formalism that we cannot say in the second.

First order logic

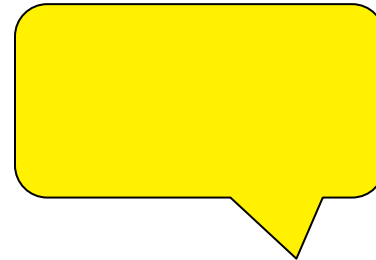
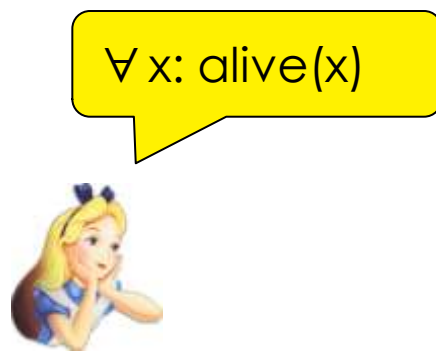
$\forall x: \text{rocksinger}(x) \Rightarrow \text{alive}(x)$



Propositional logic

?

In general, a higher expressiveness is desirable from a modeling point of view



... but it comes at a cost...

# KR Design: Decidability

A KR formalism is **decidable**, if there is an algorithm (computer program) that can answer any query on a knowledge base in that formalism.

(Technically: A logical system is decidable iff there is an effective method for determining whether arbitrary formulas are theorems of the logical system.)

Some formalisms are so expressive that they are undecidable:

- Natural language

Is sentence (\*) true?



(\*) This sentence is false.

- First order logic

First order logic is so powerful that it can express sentences of which it is impossible to determine whether they are true or not.

# KR Design: Decidability

A KR formalism is **decidable**, if there is an algorithm (computer program) that can answer any query on a knowledge base in that formalism.

(Technically: A logical system is decidable iff there is an effective method for determining whether arbitrary formulas are theorems of the logical system.)

In general, decidability is desirable.

The more expressive a formalism is, the more likely it is to be undecidable.

Often, a formalism can be made decidable by restricting it

- Propositional logic is decidable
- First order logic is decidable if all formulae are of the following form:

$$\underbrace{\exists x, y, \dots}_{\text{existential}} \underbrace{\forall z, q, \dots}_{\text{universal}} : \underbrace{p(x, y) \dots \Rightarrow \dots}_{\text{arbitrary formula without quantifiers and function symbols}}$$

existential  
quantifiers

universal  
quantifiers

arbitrary formula  
without quantifiers  
and function symbols

# KR Design: Closed world

A KR formalism follows the **closed world assumption (CWA)**, if any statement that cannot be proven is assumed to be false

PROLOG, e.g., follows the CWA:

```
?- assert(bornIn(Elvis, Tupelo)).
```

```
yes
```

```
?- alive(Elvis).
```

```
no
```

In many contexts, the open world assumption (OWA) is more appropriate. Under the OWA, a statement can be

- provably false
- provable true
- unknown

```
?- alive(Elvis).
```

```
I have no clue
```

# KR Design: Reification

A KR formalism allows **reification**,  
if it can treat statements like entities.

thinks(Fabian, alive(Elvis)).

*reification, a statement  
appears as argument*

---

=> ~~alive(Elvis)~~

**Modal logic** allows talking about necessity and possibility

“It is possible that Elvis is alive”

◇ alive(Elvis)

# KR Design: Unique Name

A KR formalism follows the **unique name assumption (UNA)**, if different names always refer to different objects.



Elvis



The King

The UNA is not useful if different people want to use different identifiers:



- ?- `assert(alive(TheKing)).`  
yes
- ?- `assert(TheKing=Elvis).`  
hmm, OK
- ?- `alive(Elvis).`  
yes

# KR Design: Schema

A KR formalism is **schema-bound**, if one has to decide upfront which entities can have which properties.

PROLOG is schema-free, any entity can have any property:

?- assert(alive(Elvis)).  
yes

?- assert(hasResolution(Elvis,5 megapixel))  
yes



In schema-bound formalisms, one has to decide a priori for classes of things and their properties:

person:

- alive/dead
- birthday
- profession

camera:

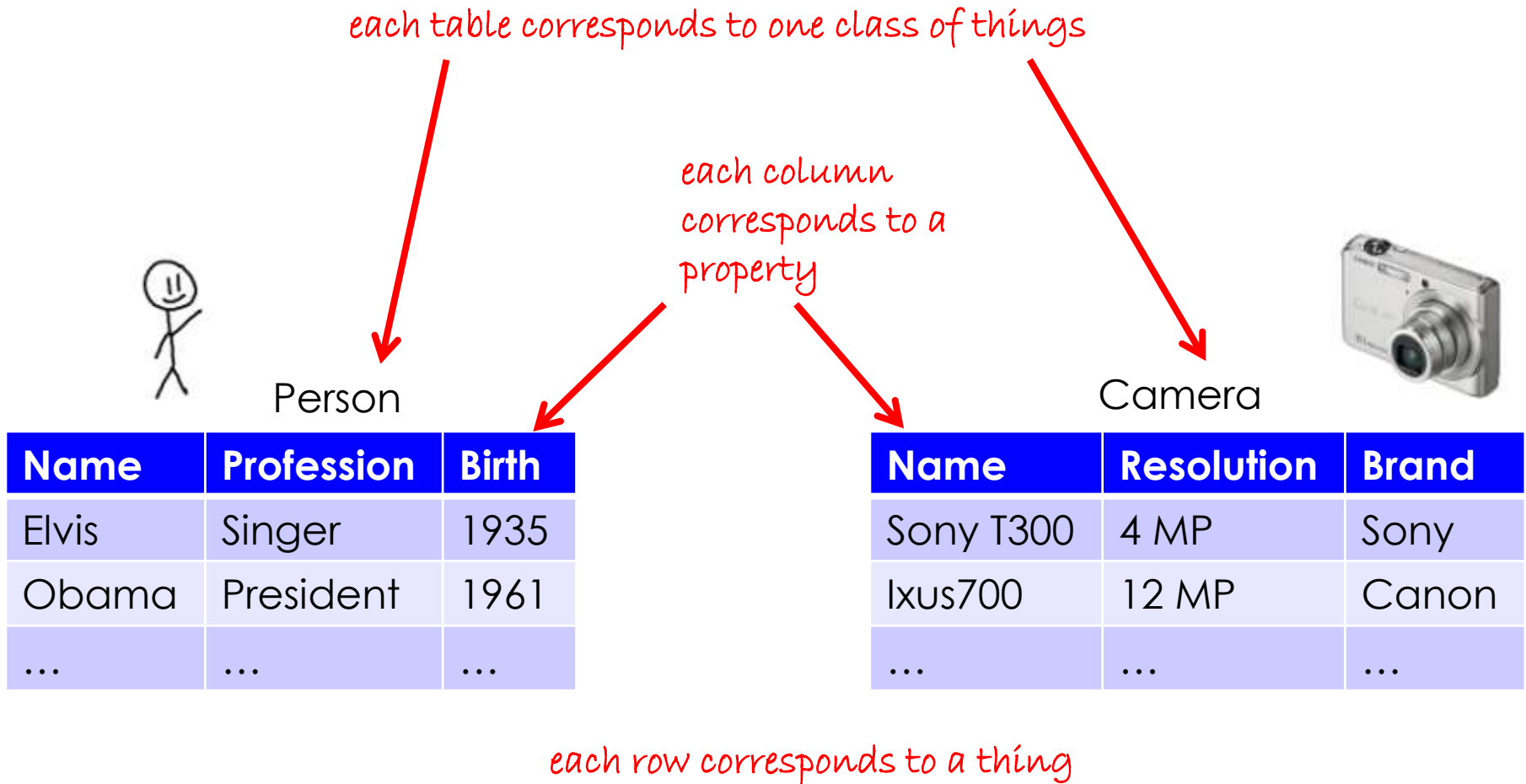
- resolution
- shutter speed
- weight

A schema-bound formalism puts more modeling constraints, but can exclude non-sensible statements.

# KR Design: Schema

**Databases** are a particular schema-bound KR formalism.

A database can be seen as a set of **tables**.



# KR Design: Schema

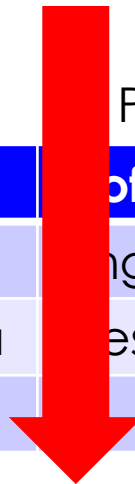
**Databases** are a particular schema-bound KR formalism.

A database can be queried in the Structured Query Language (**SQL**).

```
SELECT name, birth  
FROM person  
WHERE profession='singer'  
AND birth>1930
```

Person

Name	Profession	Birth
Elvis	singer	1935
Obama	President	1961
...		...



Elvis, 1935  
JohnLennon, 1940  
...

Camera

Name	Resolution	Brand
Sony T300	4 MP	Sony
Ixus700	12 MP	Canon
...	...	...

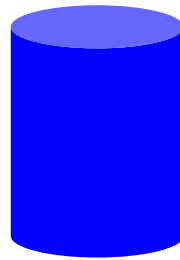
# KR Design: Schema

**Databases** are a particular schema-bound KR formalism.

A database can be queried in the Structured Query Language (**SQL**).

Databases are used in practically every major enterprise, with the main database systems being

- Oracle
- Microsoft SQL Server
- IBM's DB2
- Postgres
- MySQL



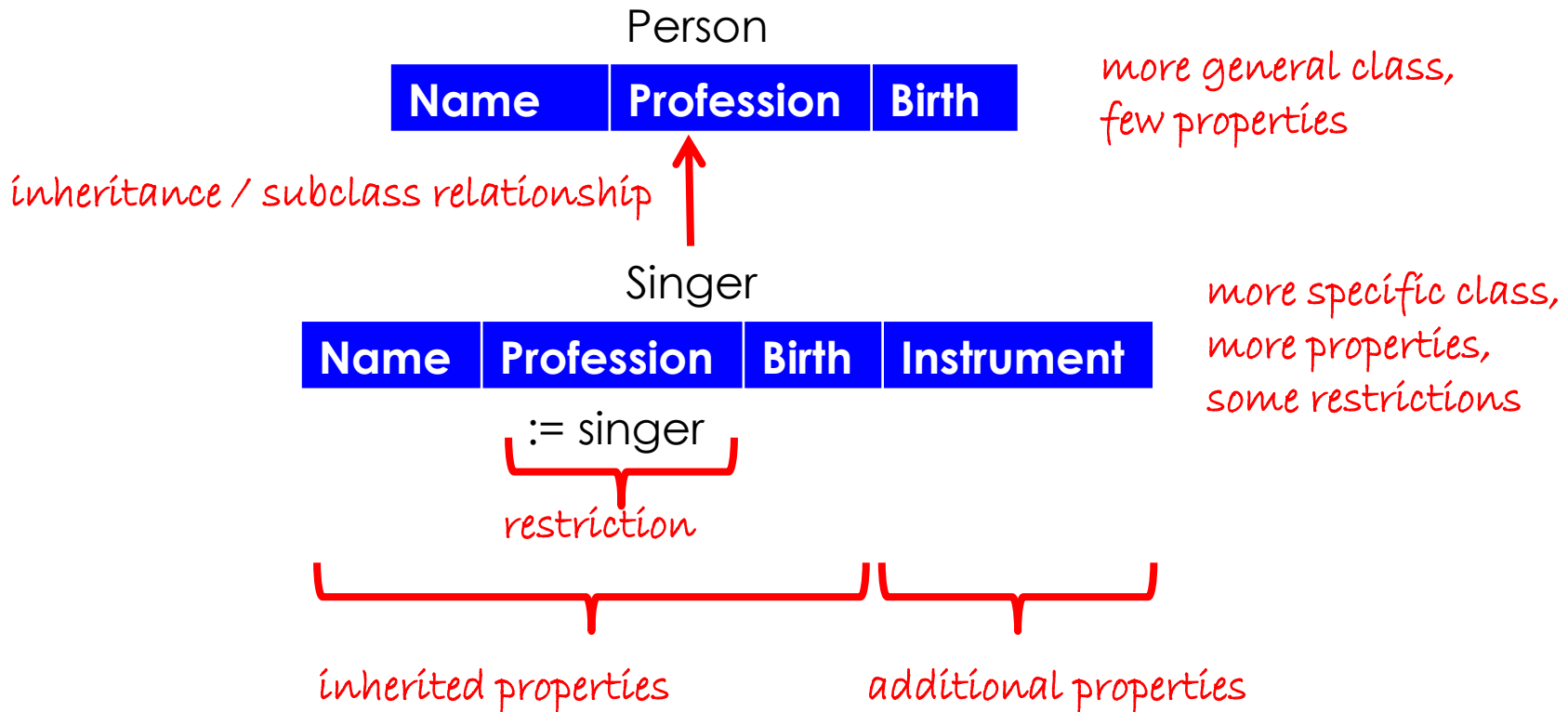
Name	Profession	Birth
Elvis	Singer	1935
Obama	President	1961
...	...	...



*Headquarters of Oracle in Redwood Shores, CA, USA*

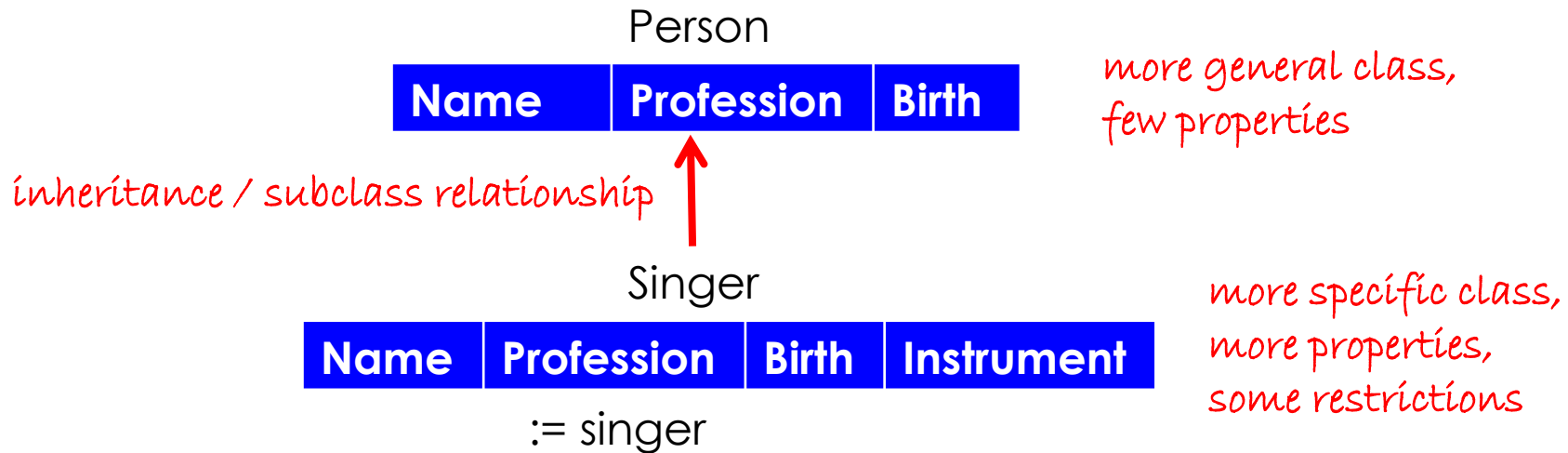
# KR Design: Inheritance

A KR formalism supports **inheritance**, if properties specified for one class of things can be automatically transferred to a more specific class.  
A **class** is a set of entities with the same properties.



# KR Design: Inheritance

A KR formalism supports **inheritance**, if properties specified for one class of things can be automatically transferred to a more specific class.  
A **class** is a set of entities with the same properties.



Inheritance is useful, because

- it avoids duplication of work  
(no need to reinvent the wheel)
- it makes the KR formalism more canonic  
(the subclass automatically has the same properties as the superclass)

# KR Design: Inheritance

**Object-oriented programming languages** (such as e.g., Java) support inheritance.

```
public class Person {  
    String name;  
    String profession;  
}
```

*super-class*  
*\* declares two properties*

```
public class Singer extends Person {  
    String profession="Singer";  
    String instrument;  
}
```

*sub-class*  
*\* overwrites a property*  
*\* adds a property*

```
Singer elvis=new Singer();  
elvis.name="Elvis";  
elvis.instrument="guitar";  
System.out.println(elvis.profession)  
→ Singer
```

*Elvis is a singer*  
*\* but he inherited properties*  
*\* and has predefined values*

# KR Design: Inheritance

**Object-oriented programming languages** (such as e.g., Java) support inheritance.

Most programming languages support object-orientation today, with the most important ones being

- Java
- C++
- C#
- Visual Basic
- Python
- Ruby



```
Singer elvis=new Singer();  
elvis.name="Elvis";  
elvis.instrument="guitar";  
System.out.println(elvis.profession)  
→ Singer
```

# KR Design: Monotonicity

A KR formalism is **monotonous**, if adding new knowledge does not undo deduced facts.

First order logic and propositional logic are monotonus:

elvis\_is\_person  
elvis\_is\_alive  
elvis\_is\_dead => ~ elvis\_is\_alive



elvis\_is\_person  
elvis\_is\_alive  
elvis\_is\_dead => ~ elvis\_is\_alive  
+ elvis\_is\_dead

---

=> elvis\_is\_alive



---

=> elvis\_is\_alive  
=> elvis\_is\_dead  
=> michael\_jackson\_alive

Monotonicity can be very counter-intuitive.

It requires everything to be known upfront.

# KR Design: Monotonicity

A KR formalism is **monotonous**, if adding new knowledge does not undo deduced facts.

**Default logic** is not monotonous:

elvis\_is\_person

prerequisite  $\rightarrow$  elvis\_is\_dead  
conclusion  $\rightarrow$   $\sim$ elvis\_is\_alive

if Elvis is dead  
then he is not alive

justification

elvis\_is\_person: elvis\_is\_alive  
elvis\_is\_alive

if Elvis is a person  
and nothing says he's not alive  
then he is alive

+ elvis\_is\_dead

---

$\Rightarrow$  ~~elvis\_is\_alive~~

# KR Design: Fuzziness

A KR formalism is **fuzzy**, if certain statements can hold to a certain degree. The opposite of fuzzy is **crisp**.



fantastic(Bush)  
0.1



fantastic(Madonna)  
0.8



fantastic(Elvis)  
1.0

First order logic, PROLOG and propositional logic are all crisp.

# KR Design: Fuzziness

A KR formalism is **fuzzy**, if certain statements can hold to a certain degree.  
**Fuzzy logic** is a fuzzy KR formalism.

rainy (0.8)

rainy  $\Rightarrow$  bad\_weather

---

bad\_weather (0.8)

rainy (0.8)

windy(1.0)

rainy  $\vee$  windy  $\Rightarrow$  bad\_weather

---

bad\_weather (??)

1.0

Fuzzy logic defines how to compute fuzzy values for complex combinations of fuzzy predicates, e.g.

- OR is computed as maximum
- AND is computed as minimum
- NOT is computed as  $1-x$

# KR Design: Contradictions

A KR formalism is **tolerant to contradictions** if it allows contradictions in the knowledge base.

First order logic and propositional logic are not tolerant to contradictions:

alive(Elvis).  
~alive(Elvis).

---

=> life\_is\_beautiful

*ex falso quod libet...*

Some domains require handling of contradictions  
(e.g. information extraction from the Web)

Elvisfans.com:  
Elvis is alive

Wikipedia:  
Elvis is dead.

# KR Design: Contradictions

A **Markov Logic Network** is a set of propositional logic formulas with weights:

f1: elvis\_alive [0.8]

f2: elvis\_dead => ~elvis\_alive [0.5]

f3: elvis\_dead [0.6]

A **possible world** is an assignment of truth values to the predicates:

w1:	w2:	w3:	w4:
elvis_alive ✓	elvis_alive ✗	elvis_alive ✗	elvis_alive ✓
elvis_dead ✗	elvis_dead ✗	elvis_dead ✓	elvis_dead ✓
0.3	0.1	0.5	0.9

A Markov Logic Network assigns a probability to each possible world:

$$P(w) \sim \prod_{\text{satisfied formula } f} e^{\text{weight}(f)}$$

(normalized by the sum of the values for all worlds)

$$P(w4) \sim e^{0.8} \cdot \cancel{e^{0.5}} \cdot e^{0.6} = 4.0$$

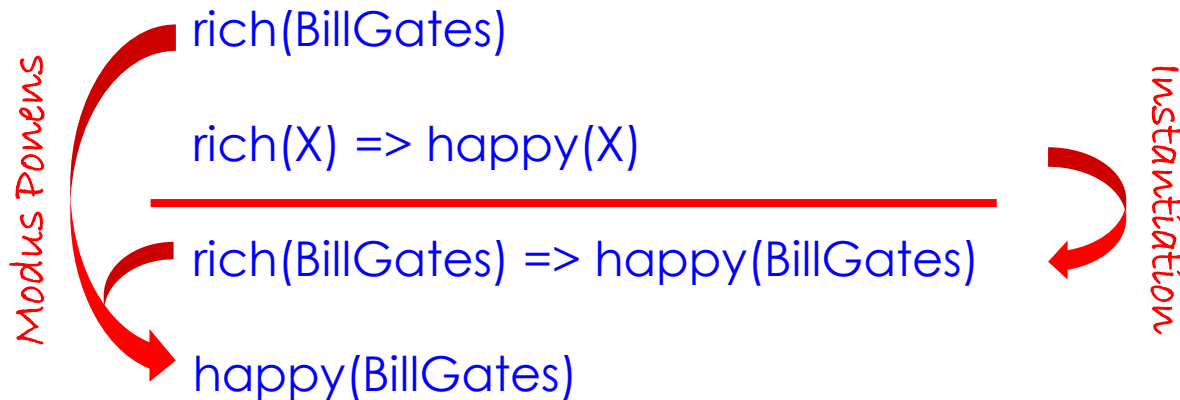
f1      f2      f3

Finding the most probable world boils down to the **Weighted Maximum Satisfiability Problem**.

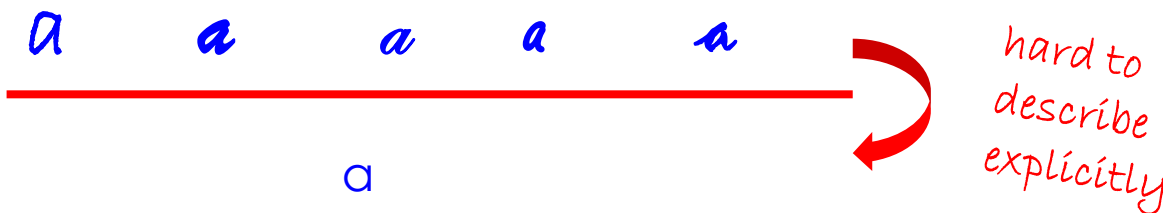
# KR Design: Explicitness

A KR formalism is **explicit**, if it can explain why a certain statement was deduced.

Logic-based formalisms are usually explicit:



Explicitness may not always be useful, consider pattern recognition:



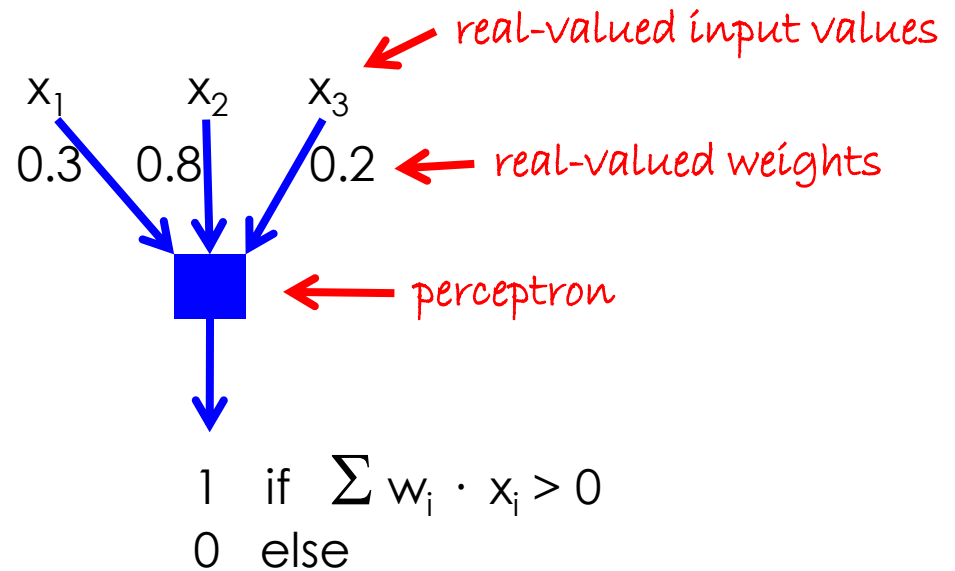
# KR Design: Explicitness

A **perceptron** is a simplified model of a human nerve cell (neuron)



human nerve cell

≈



Examples:

input:  $x_1 = 1$ ,  $x_2 = 2$ ,  $x_3 = 1$

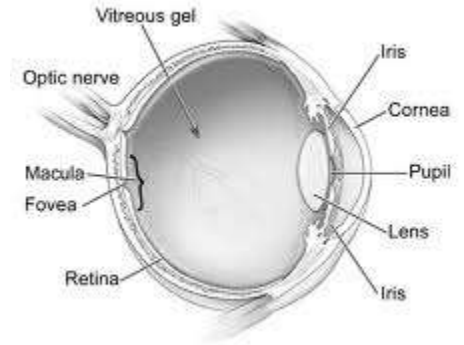
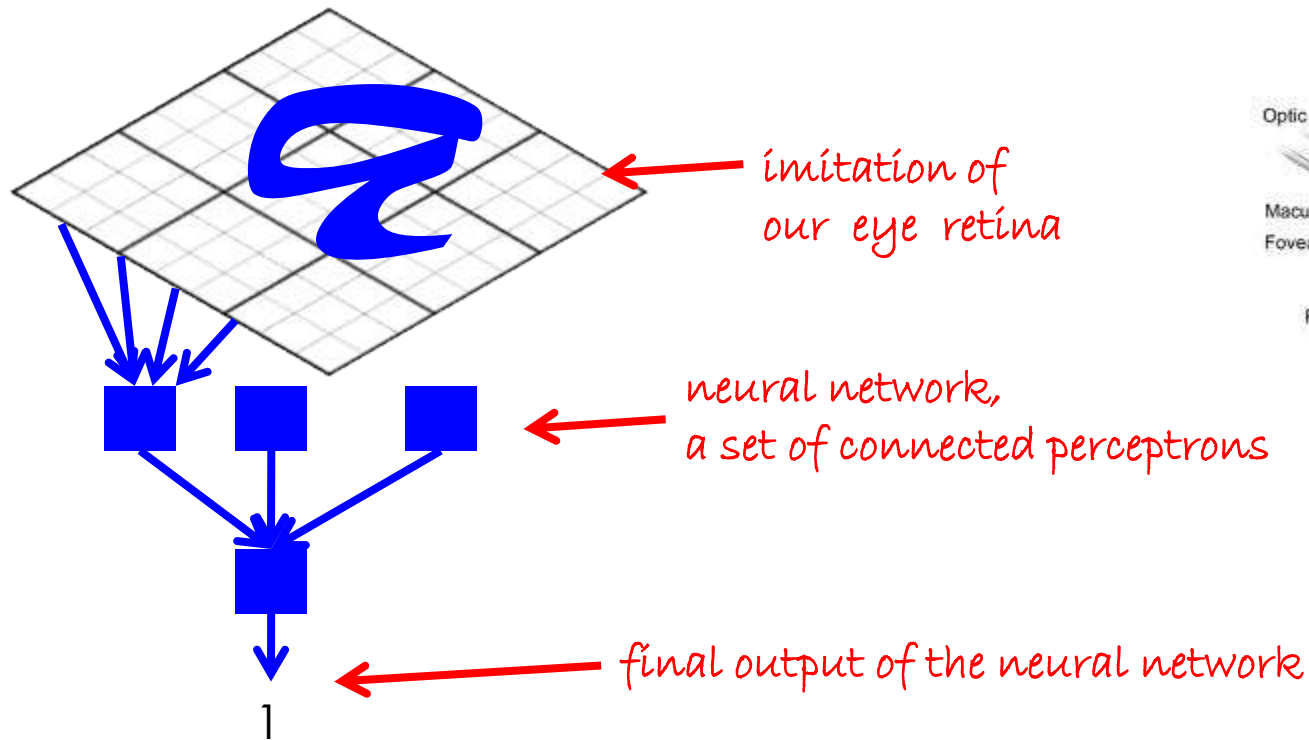
$$\sum w_i \cdot x_i = 0.3 \cdot 1 + 0.8 \cdot 2 + 0.2 \cdot 1 = 2.1 > 0$$

=> output: 1

input:  $x_1 = 0$ ,  $x_2 = -2$ ,  $x_3 = 0$

=> output: 0

# KR Design: Explicitness

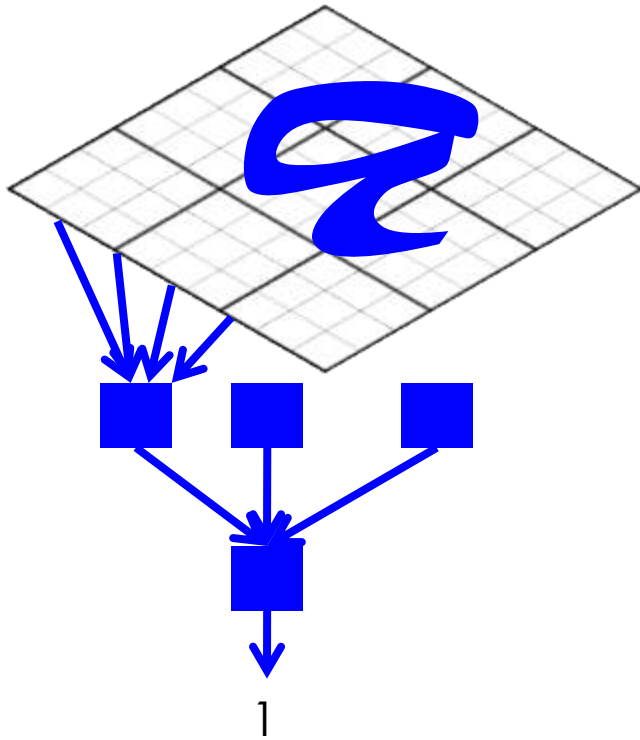


**Training a neural network:** Adjusting the weights so that the network says 1 for all positive examples.



*a* ← net will also recognize this one

# KR Design: Explicitness



A Neural Network is a non-explicit knowledge representation.

Neural Networks are used, e.g,

- to recognize postal codes on letters
- to optimize search engine results

*This network "knows" what the letter "a" is.*

*But this knowledge is not explicit.*

# KR Design: Distributedness

A KR formalism is **distributed**, if it encourages use and co-operation by different people / systems across different places / organizations.



Elvis is alive

~~~~~> We will see a very popular distributed formalism shortly

# KR Design: Summary

In general, a KR formalism serves to

- represent knowledge
- infer facts from that knowledge
- answer queries on the knowledge

There are many KR formalisms with different properties:

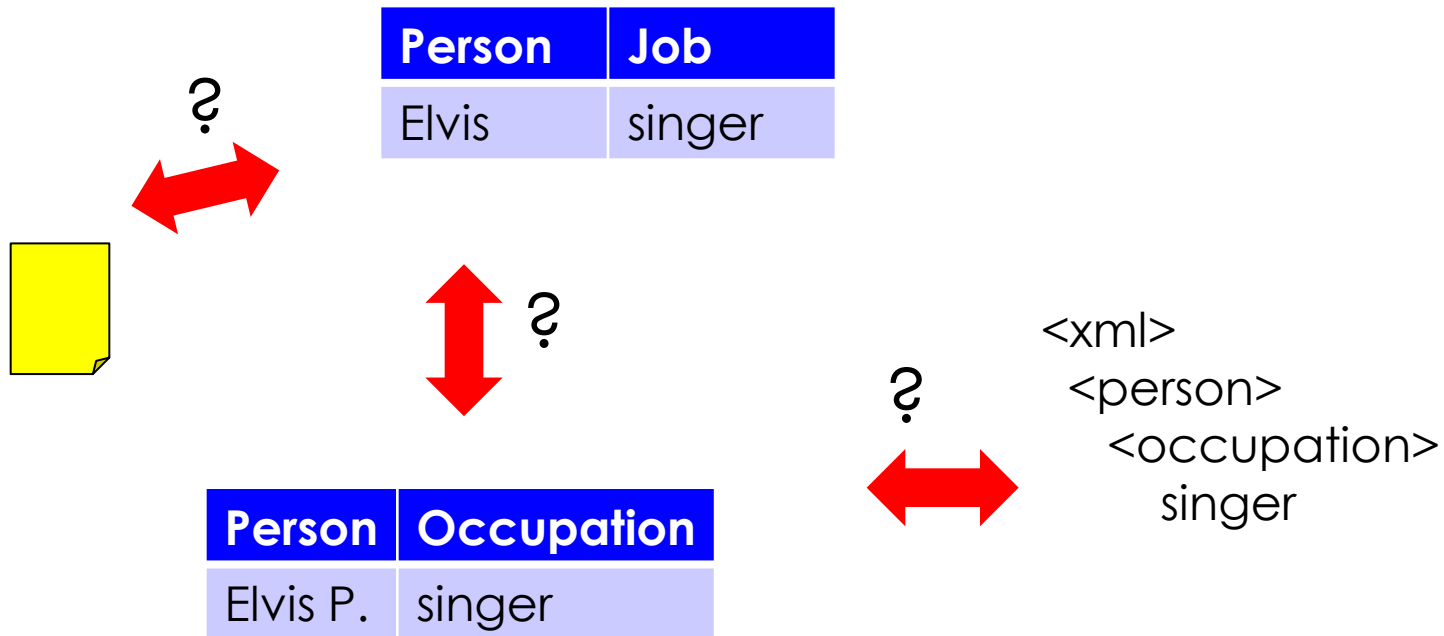
- **Canonicity** (does the formalism allow only one form to represent a statement?)
- **Expressiveness** (how powerful is the formalism)
- **Decidability** (can every query be answered?)
- **Closed World** (does the formalism assume that everything unknown is false?)
- **Unique Name** (do two names always mean two different things?)
- **Schema-bound** (do we have to decide upfront on properties of things?)  
Databases/SQL is a schema-bound KR formalism
- **Inheritance** (can properties be transferred from one class to another?)  
Object-oriented programming languages support inheritance
- **Monotonicity** (will new knowledge never undo existing knowledge?)  
Default logic allows non-monotonicity
- **Fuzziness** (can properties be fulfilled to a certain degree?)  
Fuzzy logic is a fuzzy KR formalism
- **Tolerance to contradictions** (is it OK to have a contradiction in the KB?)  
Markov Logic Networks can deal with contradictions
- **Explicitness** (is knowledge always explainable?)  
Neural Networks store knowledge only implicitly

# Overview

- Motivation
- Knowledge Representation Design
- • Knowledge Representation in the Semantic Web

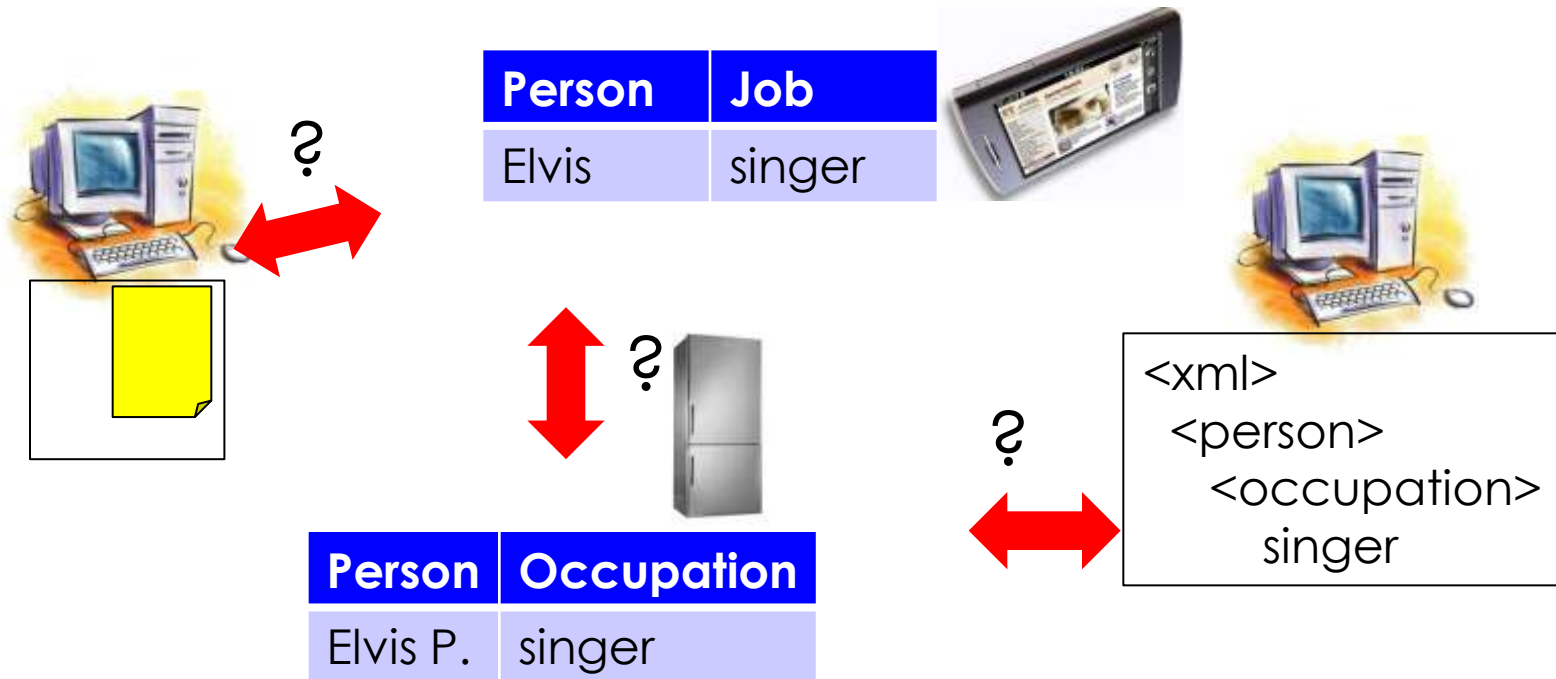
# Motivation

Interaction between data on the Web is difficult, in particular if the data is in different formats



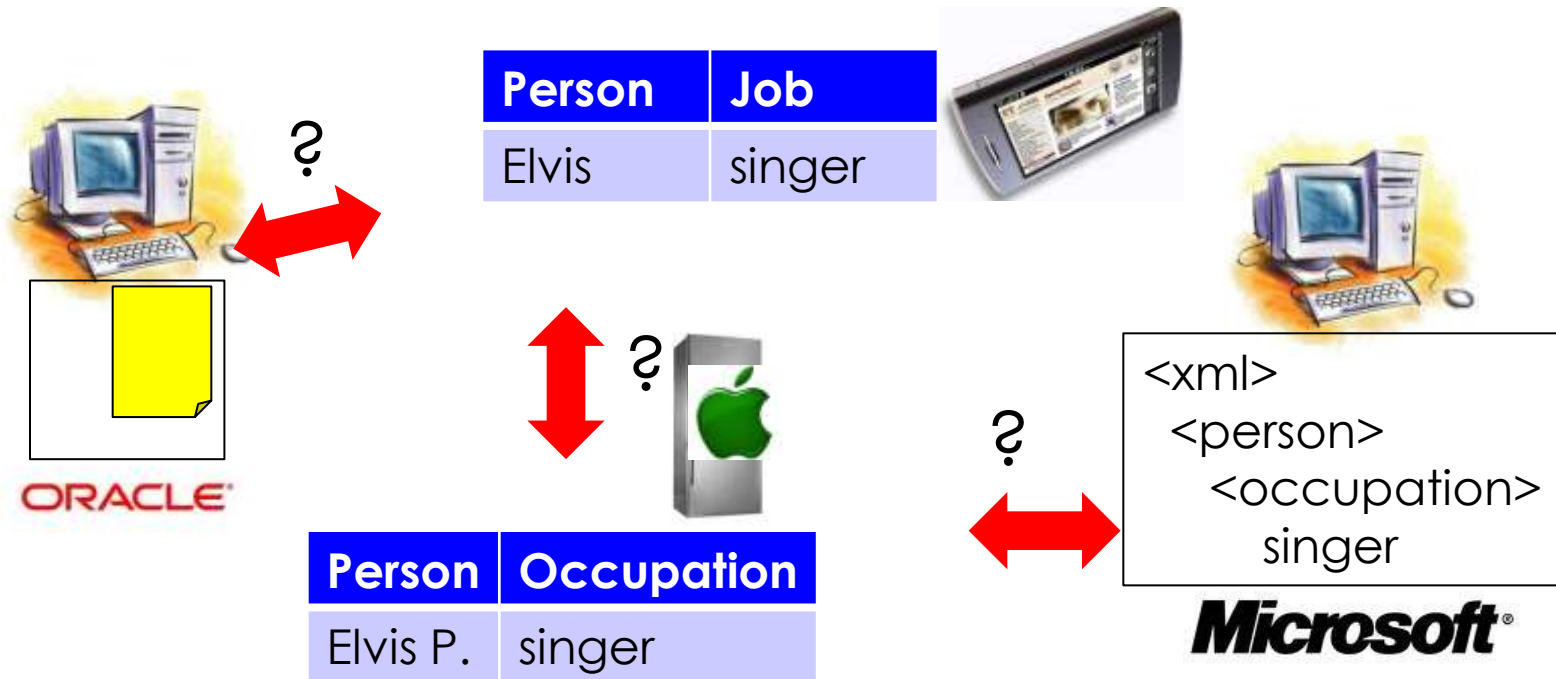
# Motivation

Interaction between data on the Web is difficult, in particular if the data is in different formats, on different machines or devices



# Motivation

Interaction between data on the Web is difficult, in particular if the data is in different formats, on different machines or devices or in different companies



#WCD0911.010

# Motivation: Use cases

## Examples:

- Booking a flight  
Interaction between office computer, flight company, travel agency, shuttle services, hotel, my calendar
- Finding a restaurant  
Interaction between mobile device, map service, recommendation service, restaurant reservation service
- Web search  
Interaction between client, search service, Web page content provider
- Web service composition  
Interaction between client and Web services and Web services themselves
- Intelligent home  
Fridge knows my calendar, orders food if I am planning a dinner
- Intelligent cars  
Car knows my schedule, where and when to get gas, how not to hit other cars, what are the legal regulations

# Motivation: Merging

Examples:

- Adding data to a database  
From XML files, from other databases
- Merging data after company mergers (e.g. Apple buys Microsoft)  
Different terminology has to be bridged, accounts to be merged
- Merging data in research  
e.g. biochemical, genetic , pharmaceutical research data

(Less exciting, but probably more frequent)

# Motivation: Semantic Web

We need a Knowledge Representation that

- allows machines to process data from other machines
- ensures interoperability between different schemas, devices and organizations
- allows data to describe data
- allows machines to reason on the data
- allows machines to answer semantic queries



This is what the Semantic Web aims at

The **Semantic Web** is an evolving extension of the World Wide Web, which promotes a new distributed knowledge representation.

# Semantic Web

The Semantic Web provides KR standards to

- Identify entities (URIs)
- Express facts (RDF)
- Express concepts (RDFS)
- Share vocabularies
- Reason on facts (OWL)

... and it even works.

*These standards are produced  
and endorsed by the **World  
Wide Web Consortium (W3C)***

The **Semantic Web** is an evolving extension of the World Wide Web, which promotes a new distributed knowledge representation.

# URIs

A **Uniform Resource Identifier** (URI) is a string of characters used to identify an entity on the Internet

Knowledge Base 1



<http://imitators.org/Elvis/FG17>

Knowledge Base 2



<http://elvis.org/me>

Knowledge Base 3



Elvis Presley

<http://onto.com/people/singers/EP>

*Entity: anything  
A person, a city, a relationship ("loves"),  
a class of things ("singer")...*

# URIs

A **Uniform Resource Identifier** (URI) is a string of characters used to identify an entity on the Internet

A URI is a generalization of a URL  
(the stuff you type in the address bar in a browser)



<http://elvis.org/me>

Identifies the person,  
not Internet-accessible

Age

75

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

Identifies a file,  
Internet-accessible

5

# URIs

A **Uniform Resource Identifier** (URI) is a string of characters used to identify an entity on the Internet

<http://imitators.org/Elvis/FG17>



World-wide unique  
mapping to domain  
owner

in the responsibility  
of the domain owner

⇒ There should be no  
URI with two meanings

⇒ People can invent all kinds of URIs

- a company can create URIs to identify its products
- an organization can assign sub-domains and each sub-domain can define URIs
- individual people can create URIs from their homepage
- people can create URIs from any URL for which they have exclusive rights to create URIs

# Semantic Web

The Semantic Web provides KR standards to

- Identify entities (URIs) ✓
- Express facts (RDF)
- Express concepts (RDFS)
- Share vocabularies
- Reason on facts (OWL)

... and it even works.

*These standards are produced  
and endorsed by the **World**  
Wide Web Consortium (W3C)*

The **Semantic Web** is an evolving extension of the World Wide Web, which promotes a new distributed knowledge representation.

# RDF: Statements

The **Resource Description Framework** (RDF) is a distributed KR formalism.

Assume we have the following URIs:

A URI for Elvis: <http://elvis.org/himself>

A URI for “winning a prize”: <http://inria.fr/rdf/dta#won>

A URI for the Grammy award: <http://g-a.com/prize>

An **RDF statement** is a triple of 3 URIs: The subject, the predicate and the object.

<http://elvis.org/himself>    <http://inria.fr/rdf/dta#won>    <http://g-a.com/prize>

We can understand an RDF statement as a First Order Logic statement with a binary predicate

`won(Elvis, Grammy award)`

# RDF: Namespaces

A **namespace** is an abbreviation for the prefix of a URI.

@prefix elvis: <http://elvis.org/>

@prefix inria: <http://inria.fr/rdf/dta#>

@prefix grammy: <http://g-a.com/>

An **RDF statement** is a triple of 3 URIs: The subject, the predicate and the object.

<http://elvis.org/himself>      <http://inria.fr/rdf/dta#won>      <http://g-a.com/prize>

... with the above namespaces, this becomes...

elvis:himself

inria:won

grammy:prize

The **default name space** is indicated by ":"

:himself

inria:won

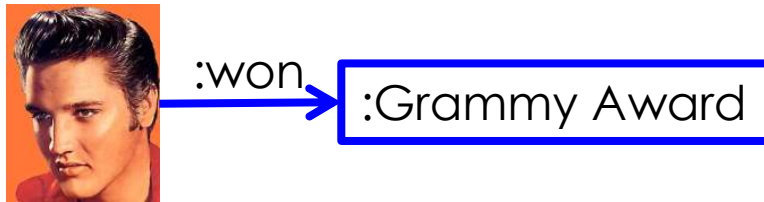
grammy:prize

# RDF: Graphs

A set of triples is **isomorphic to a labeled directed multi-graph**:

The subject and object of a triple correspond to nodes, the predicate corresponds to directed edge from subject to object with a label given by the predicate.

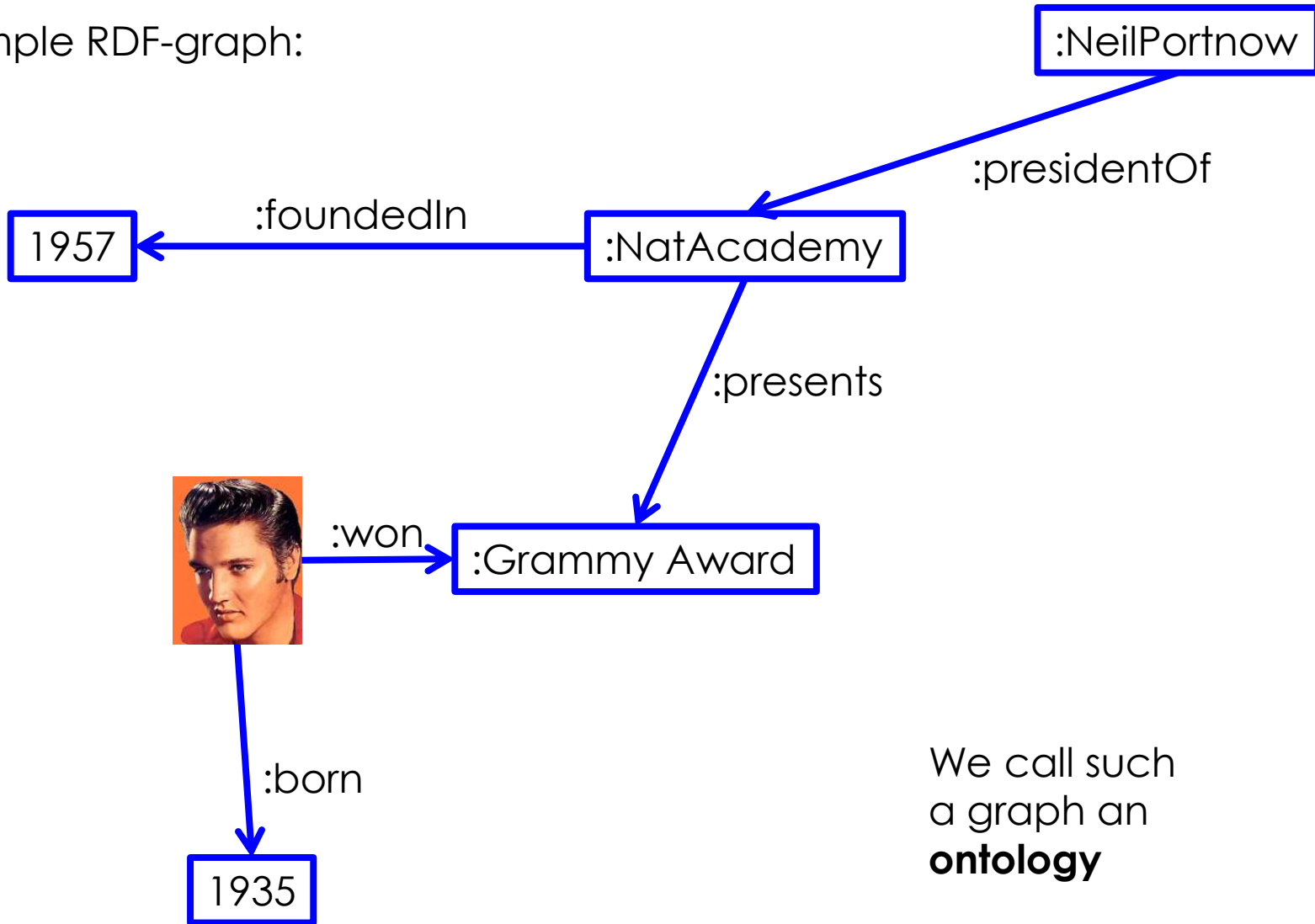
:elvis          :won          :GrammyAward



In the following, we will use this notation, assuming some default name space.

# RDF: Graphs

Example RDF-graph:



We call such a graph an **ontology**

# RDF: Event entities

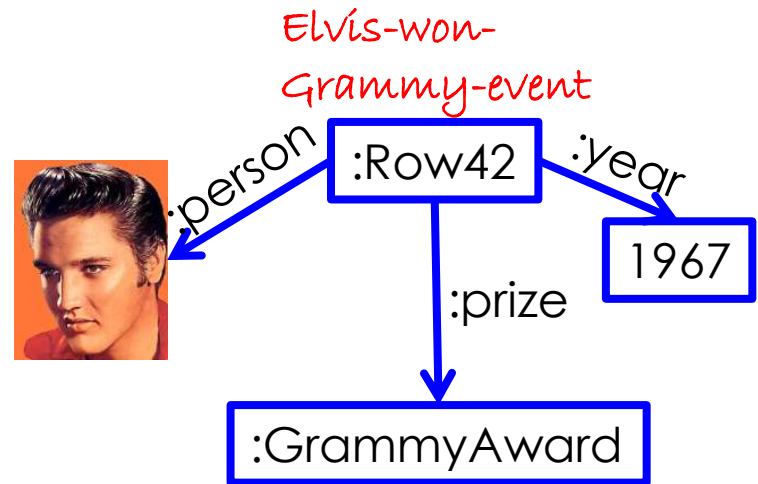
The **Resource Description Framework** (RDF) is a KR formalism that allows only binary predicates.

All tabular data and n-ary relations can be expressed in RDF by **event entities**.

| Person | Prize        | Year |
|--------|--------------|------|
| Elvis  | Grammy Award | 1967 |

42

won(Elvis, GrammyAward, 1967)

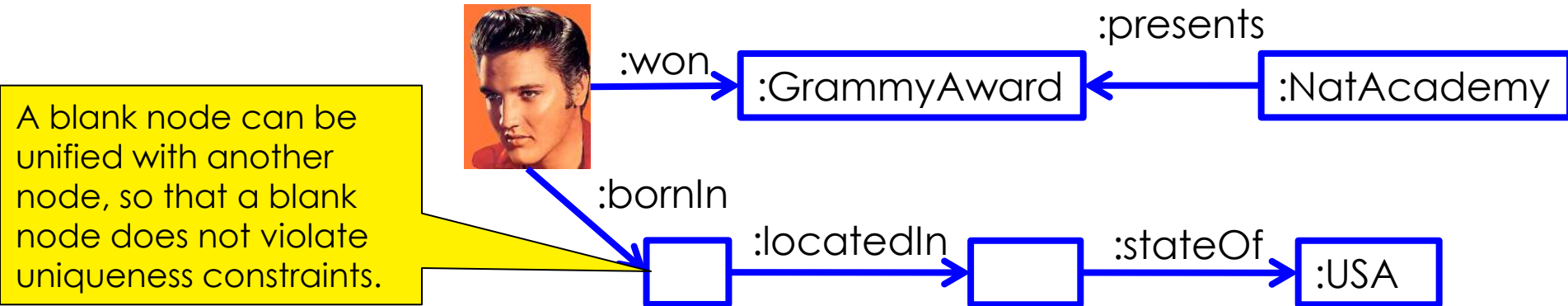


**Event entities** are artificial entities (nodes) that represent a complex constellation

# RDF: Blank nodes

The **Resource Description Framework** (RDF) is a KR formalism that allows only binary predicates.

A **blank node** is an RDF node that has no name.



## Semantics:

A triple  $\langle s, p, o \rangle$  is interpreted as a First Order Logic fact  $p(s, o)$ .

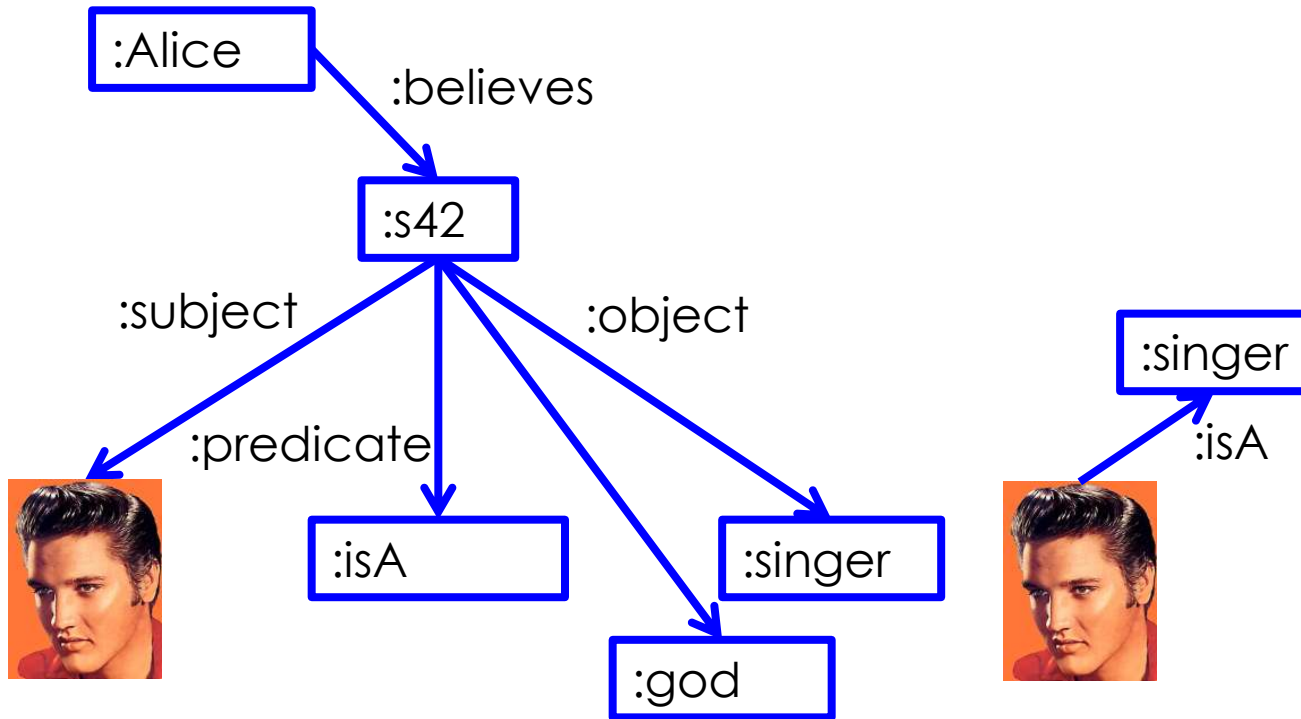
A blank node is interpreted as an existential variable.

$won(Elvis, GrammyAward)$   
 $presents(NatAcademy, GrammyAward)$

$\exists x, y: bornIn(Elvis, x) \wedge locatedIn(x, y) \wedge stateOf(y, USA)$

# RDF: Reification

A **reified statement** is an entity that represents a statement.



The reified statement can be part of the ontology or not, i.e., it does not have to be true.

# Semantic Web

The Semantic Web provides KR standards to

- Identify entities (URIs) ✓
- Express facts (RDF) ✓
- Express concepts (RDFS)
- Share vocabularies
- Reason on facts (OWL)

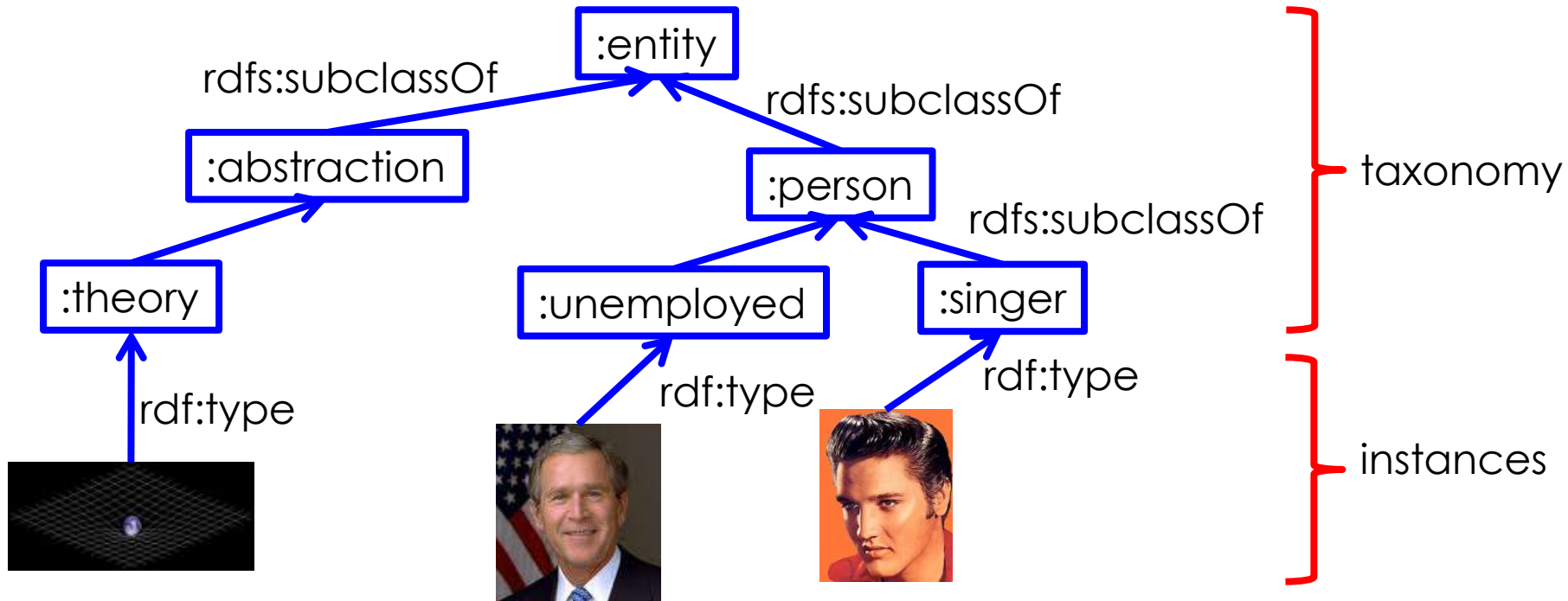
... and it even works.

*These standards are produced  
and endorsed by the **World**  
Wide Web Consortium (W3C)*

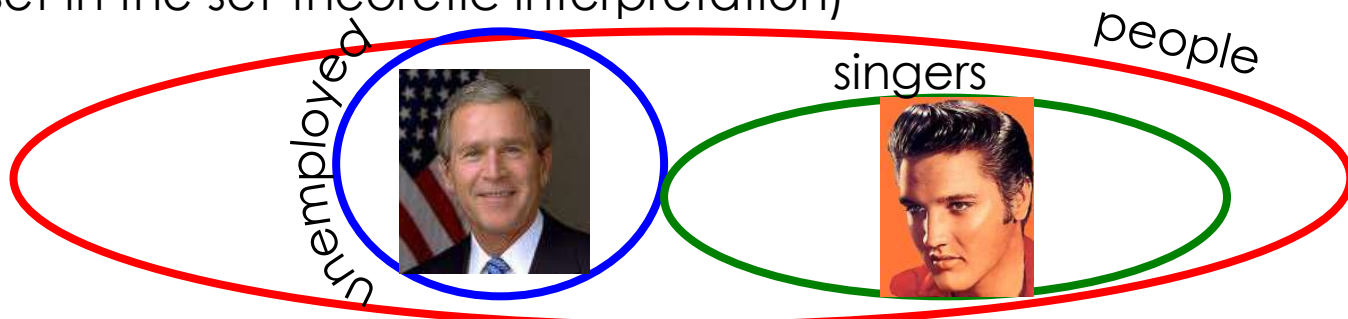
The **Semantic Web** is an evolving extension of the World Wide Web, which promotes a new distributed knowledge representation.

# RDFS: Classes

A **class** (also called concept) can be understood as a set of similar entities.

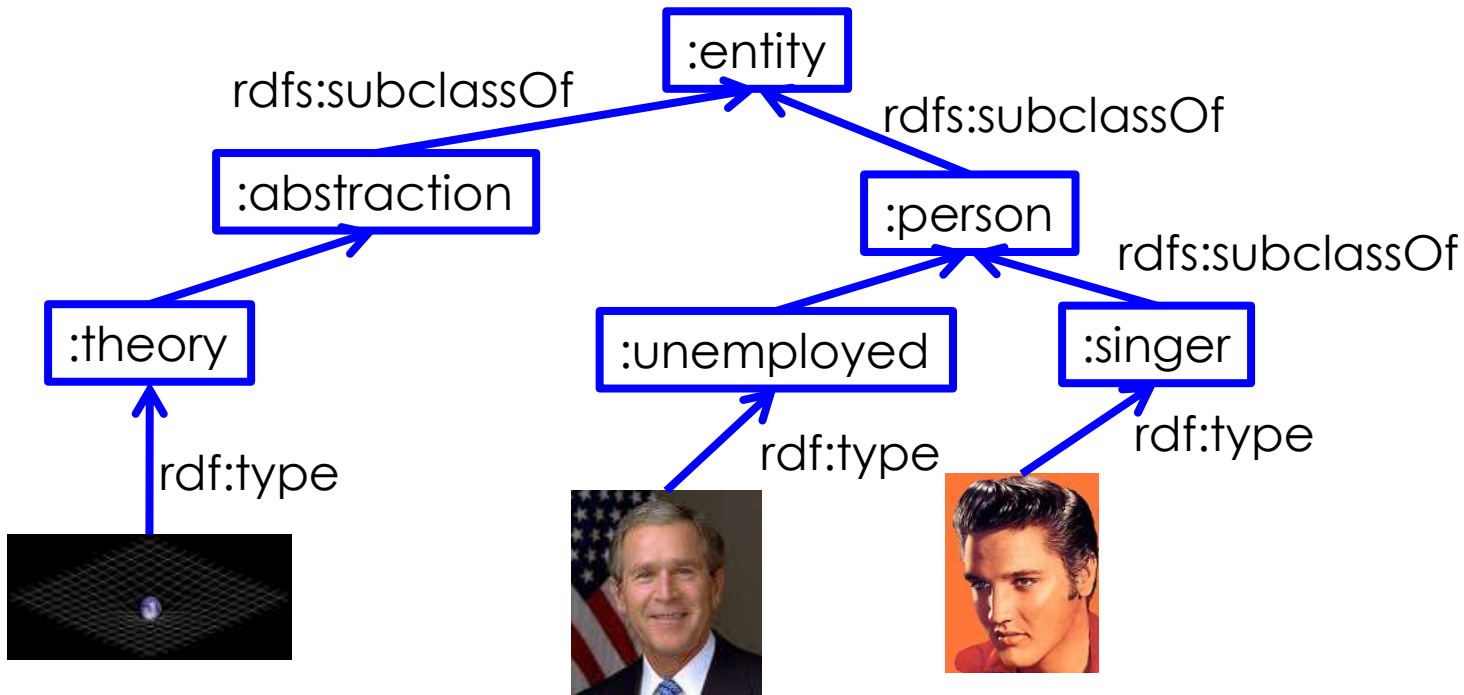


A **super-class** of a class is a class that is more general than the first class (a super-set in the set-theoretic interpretation)



# RDFS: Classes

A **class** (also called concept) can be understood as a set of similar entities.



The fact that an entity belongs to a class is expressed by the **type** predicate from the standard namespace rdf ([http://w3c.org/...](http://w3c.org/)).

The fact that a class is a sub-class of another class is expressed by the **subclassOf** predicate from the standard namespace rdfs ([http://w3c.org/...](http://w3c.org/)).

# RDFS: Entailment

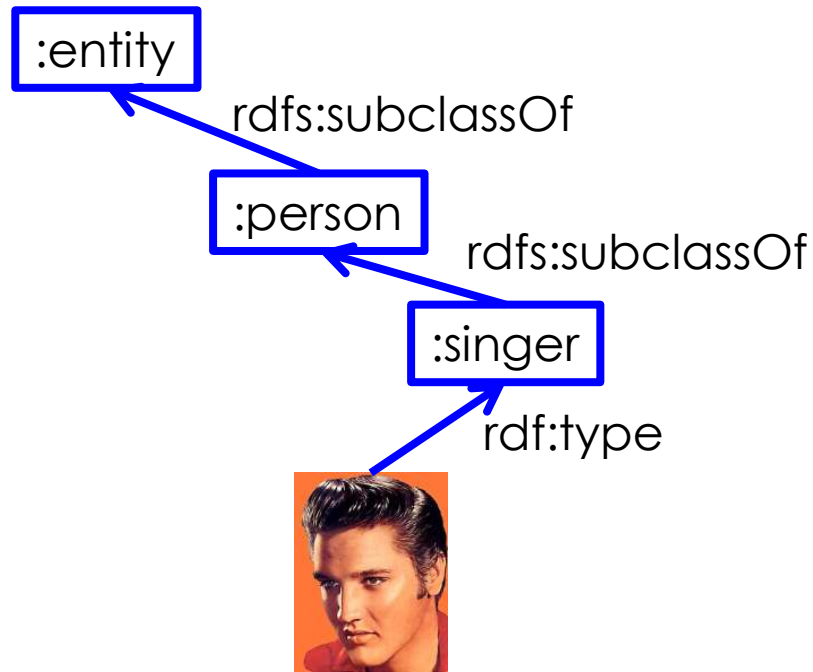
RDFS knows inferences (called **entailment** in RDFS).

Entailment rule:

If the graph contains  
such and such triples

---

then add this triple



# RDFS: Entailment

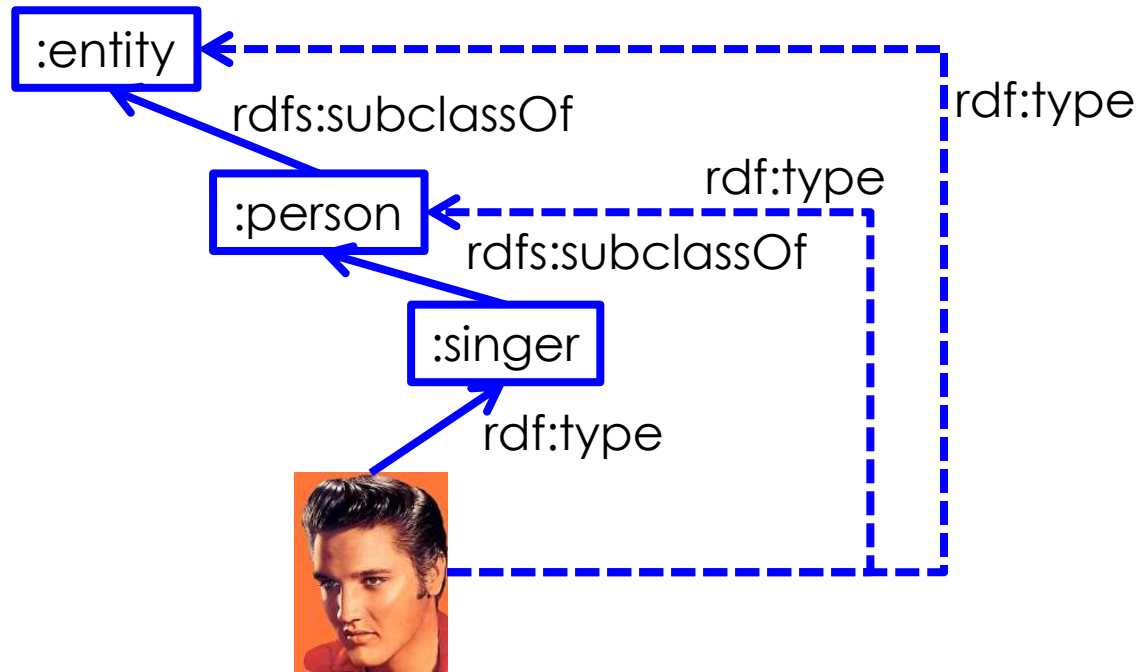
RDFS knows inferences (called **entailment** in RDFS).

Type entailment rule:

$X \text{ rdf:type } Y$   
 $Y \text{ rdfs:subClassOf } Z$

---

$X \text{ rdf:type } Z$



In first order logic:

$\forall x, y, z: \text{type}(x,y) \wedge \text{subClassOf}(y,z) \Rightarrow \text{type}(x,z)$

# RDFS: Entailment

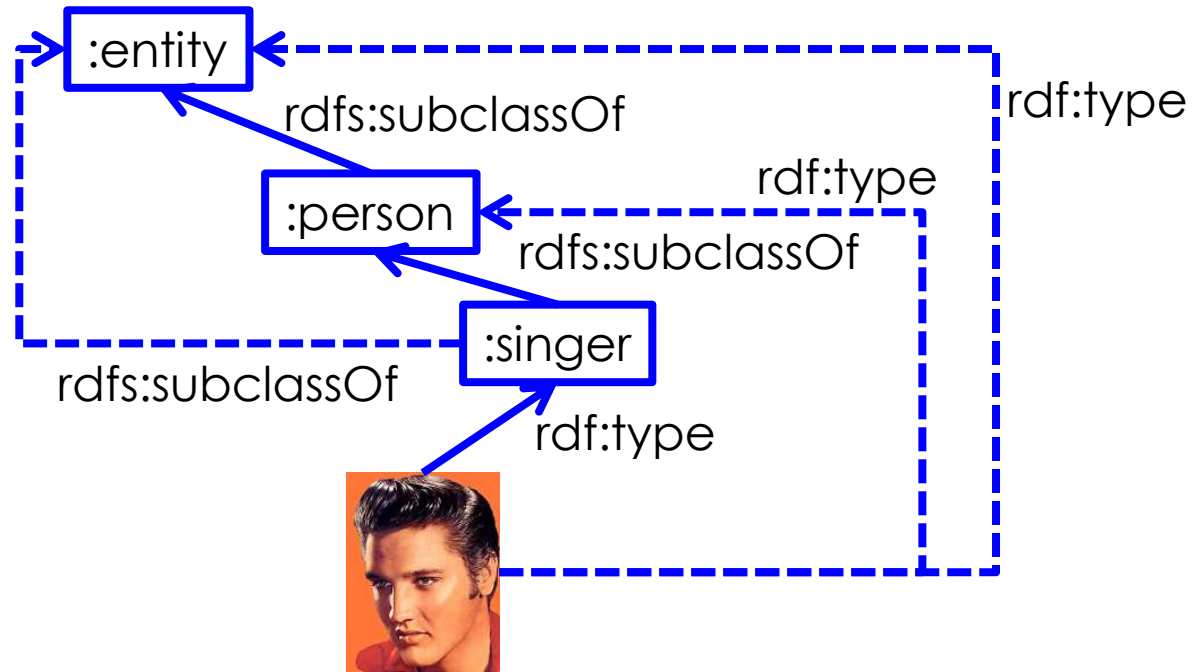
RDFS knows inferences (called **entailment** in RDFS).

Subclass entailment rule:

$X \text{ rdfs:subclassOf } Y$   
 $Y \text{ rdfs:subclassOf } Z$

---

$X \text{ rdfs:subclassOf } Z$



In first order logic:

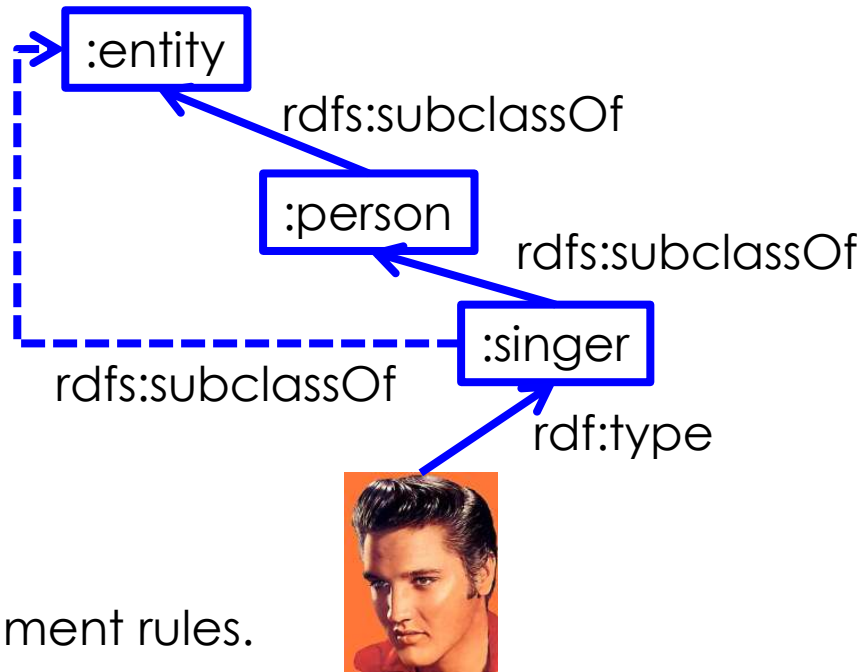
$$\forall x, y, z: \text{subclassOf}(x,y) \wedge \text{subclassOf}(y,z) \Rightarrow \text{subclassOf}(x,z)$$

# RDFS: Entailment

X rdfs:subClassOf Y  
Y rdfs:subClassOf Z

---

X rdfs:subClassOf Z



RDFS defines a set of 44 entailment rules.

The entailment rules are applied recursively until the graph does not change any more.

This can be done in polynomial time. Whether this is done physically or deduced at query time is an implementation issue.

# Semantic Web

The Semantic Web provides KR standards to

- Identify entities (URIs) ✓
- Express facts (RDF) ✓
- Express concepts (RDFS) ✓
- Share vocabularies
- Reason on facts (OWL)

... and it even works.

*These standards are produced  
and endorsed by the **World**  
Wide Web Consortium (W3C)*

The **Semantic Web** is an evolving extension of the World Wide Web, which promotes a new distributed knowledge representation.

# Storing data

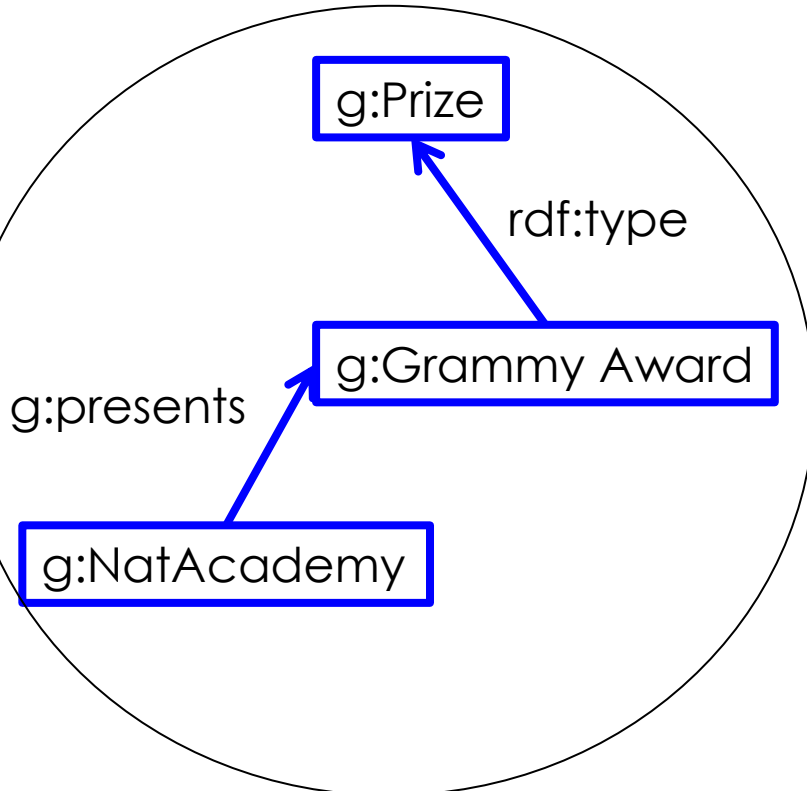
RDF data is usually stored on a server (=internet accessible computer)

Namespace

g = <http://g-a.com>



The server at  
<http://g-a.com>  
stores:



```
@prefix g: http://g-a.com  
@prefix rdf: http://www.w3.org/...
```

```
g:GrammyAward  
  rdf:type    g:Award
```

```
g:NatAcademy  
  g:presents g:GrammyAward
```

*Textual form of the ontology*

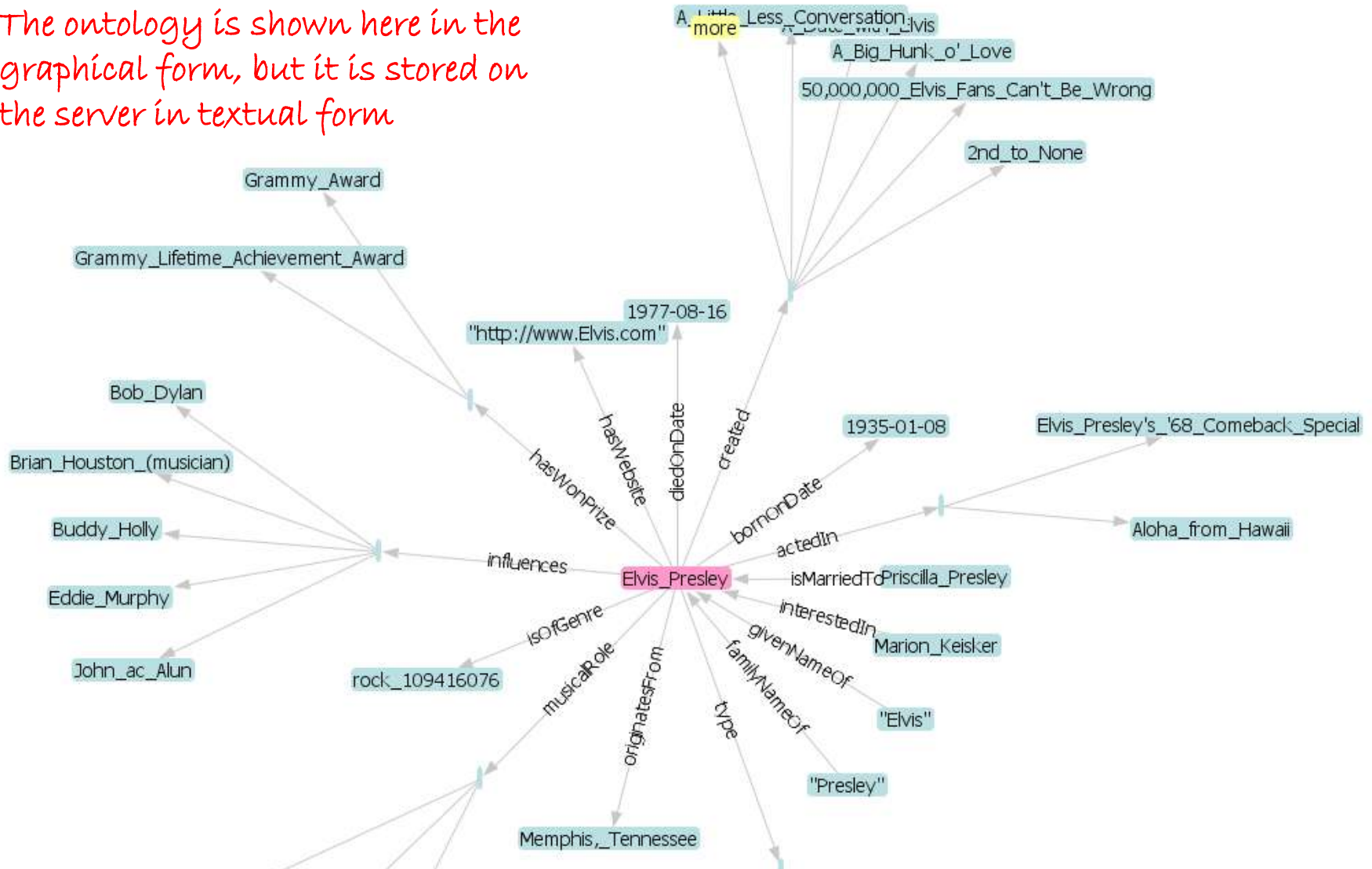
*Graphical form of the ontology*

[Try this](#)

# Storing data

Example: The YAGO ontology is stored at the server at <http://mpii.de/yago>

The ontology is shown here in the graphical form, but it is stored on the server in textual form



# Sharing: Cool URIs

A URI is not necessarily **dereferenceable** (i.e., it cannot be accessed online)

<http://g-a.com/GrammyAward> => HTTP not found error

... but it *can be* dereferenceable. This means that if I access the URL, the server responds with an RDF snippet:

```
@prefix g:      http://g-a.com
@prefix rdf:    http://www.w3.org/1999/02/22-rdf-syntax-ns#

g:GrammyAward      rdf:type      g:Award
http://elvis.com/elvis  g:won      g:GrammyAward
```

Try this out:

rdf:type = <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>

⇒ The RDF graph becomes traversable

# Sharing: Cool URIs

Server at <http://elvis.com>

```
@prefix e:      http://elvis.com
@prefix rdf:    http://www.w3.org/1999/02/22-rdf-syntax-ns#

e:elvis          rdf:type          e:singer
e:elvis          e:born            1935
```

Server at <http://g-a.com>

```
@prefix g:      http://g-a.com
@prefix rdf:    http://www.w3.org/1999/02/22-rdf-syntax-ns#

g:GrammyAward   rdf:type          g:Award
http://elvis.com/elvis  g:won            g:GrammyAward
```

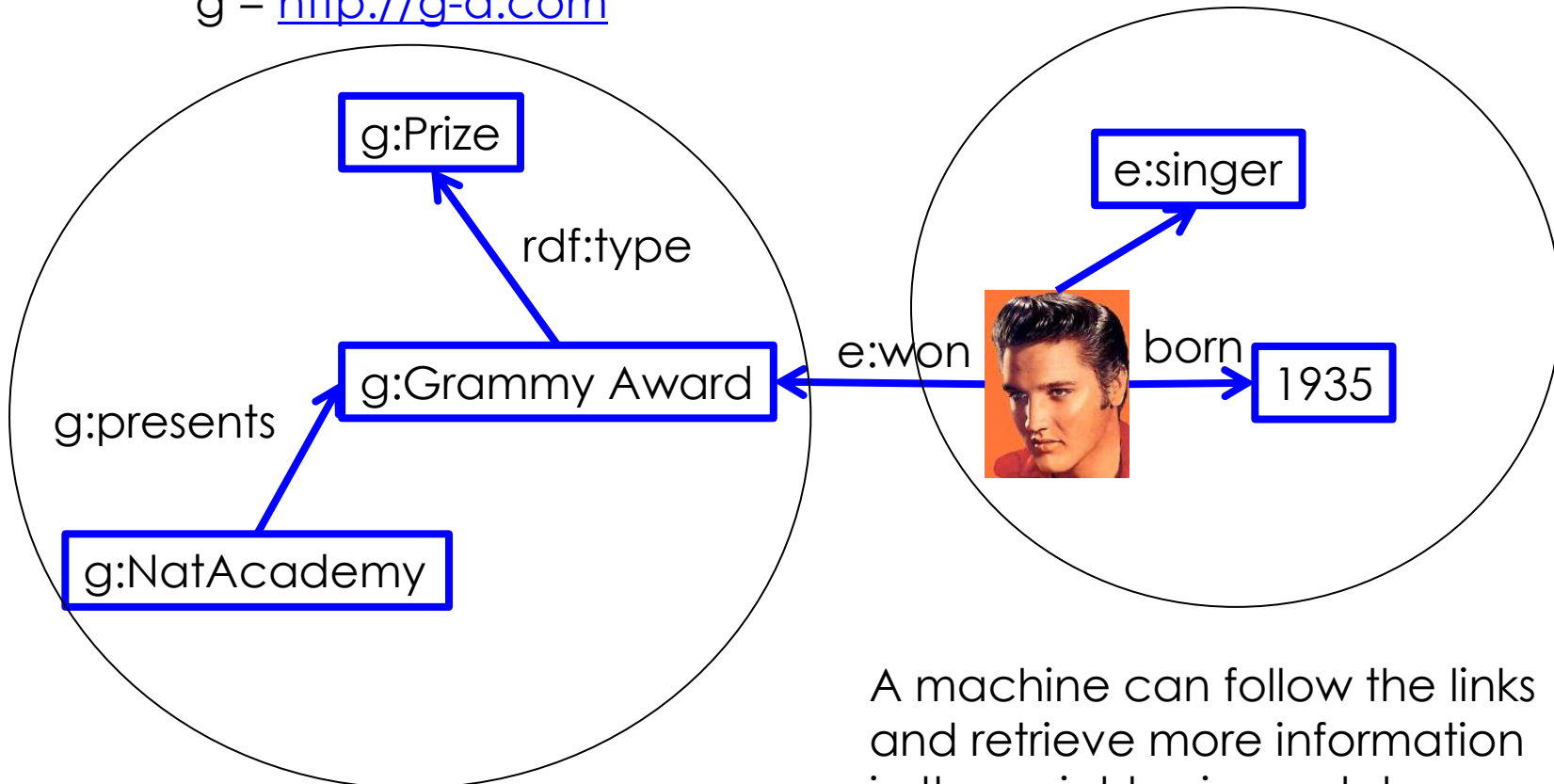
⇒ The RDF graph becomes traversable

# Sharing

If two RDF graphs share one node, they are actually one RDF graph.

Namespace  
g = <http://g-a.com>

Namespace  
e = <http://example.org>



A machine can follow the links and retrieve more information in the neighboring ontology.

# Sharing: Standard Vocabulary

A number of standard vocabularies have evolved

rdf: The basic RDF vocabulary

<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

rdfs: RDF Schema vocabulary

<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

dc: Dublin Core (predicates for describing documents)

<http://purl.org/dc/elements/1.1/>

foaf: Friend Of A Friend (predicates for relationships between people)

<http://xmlns.com/foaf/0.1/>

cc: Creative Commons (types of licences)

<http://creativecommons.org/ns#>

# Sharing: Standard Vocabulary

A number of standard vocabularies have evolved

rdf: The basic RDF vocabulary  
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

rdfs: RDF Schema vocabulary  
<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

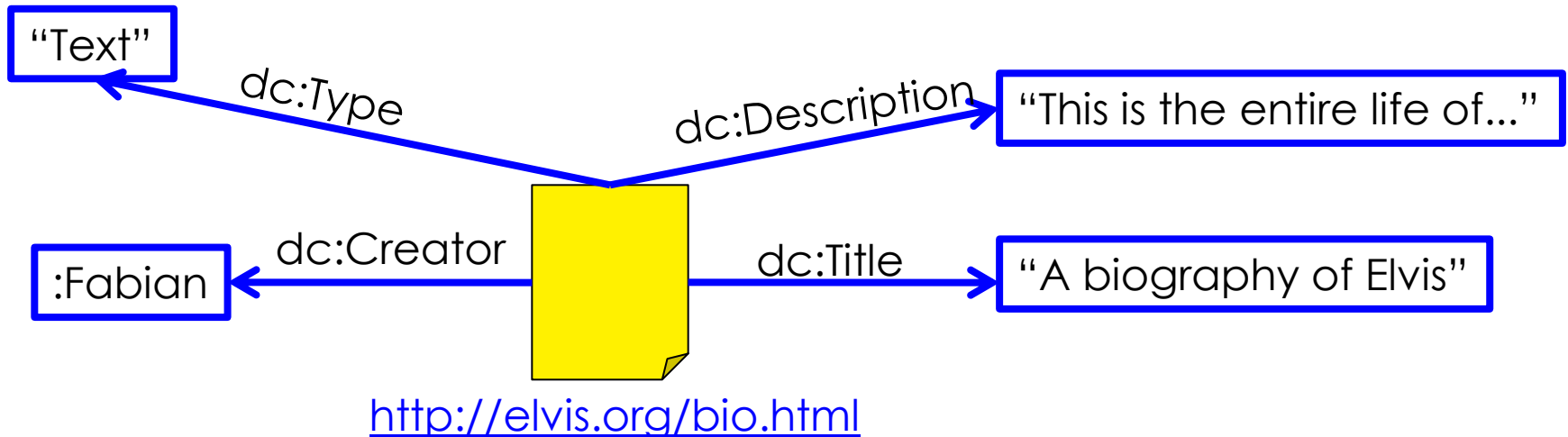
Standard vocabulary provided by the W3C:

- rdf:type
- rdfs:subClassOf
- rdf:List
- around a dozen others

# Sharing: Dublin Core

A number of standard vocabularies have evolved

dc: Dublin Core (predicates for describing documents)  
<http://purl.org/dc/elements/1.1/>

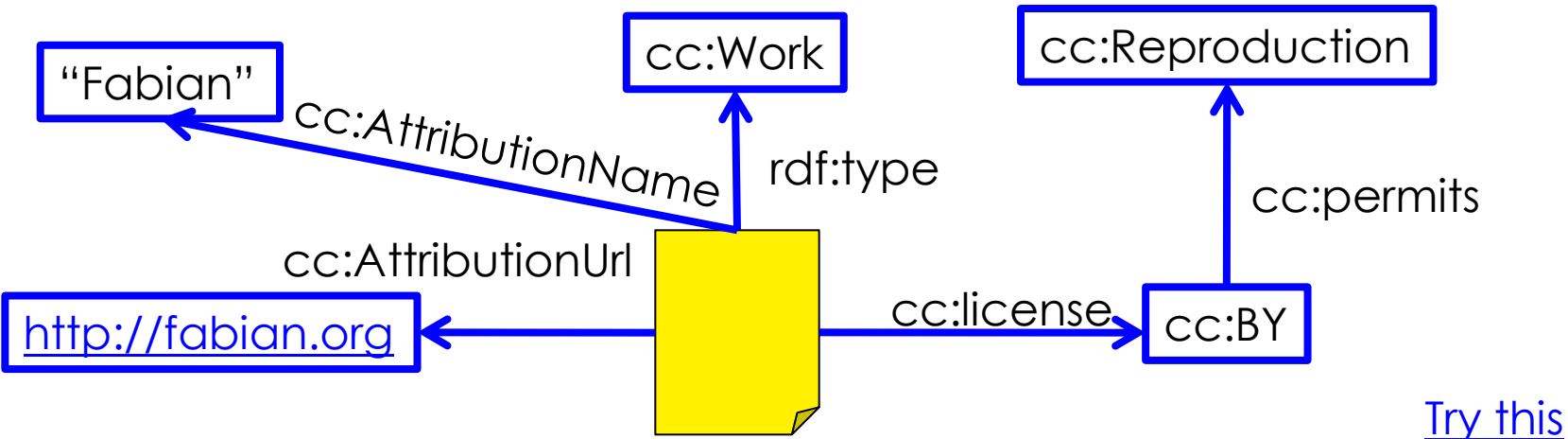


# Sharing: Creative Commons

A number of standard vocabularies have evolved

cc: Creative Commons (types of licences)

<http://creativecommons.org/ns#>



**Creative Commons** is a non-profit organization, which defines very popular licenses, notably

- CC-BY: Free for reuse, just give credit to the author
- CC-BY-NC: Free for reuse, give credit, non-commercial use only
- CC-BY-ND: Free for reuse, give credit, do not create derivative works

# What we have seen so far

RDF is a knowledge representation formalism

- that has a graphical form and a textual form



- that uses URIs to identify entities (and abbreviates them with name spaces)

@prefix yago: <http://mpii.de/yago#>

yago:elvis = <http://mpii.de/yago#elvis>

i.e., two ontologies can talk about exactly the same entity or relation

- that is stored on servers on the Internet and can be accessed by a machine



:elvis rdf:type :singer

# Semantic Web

The Semantic Web provides KR standards to

- Identify entities (URIs) ✓
- Express facts (RDF) ✓
- Express concepts (RDFS) ✓
- Share vocabularies ✓
- Reason on facts (OWL)

... and it even works.

*These standards are produced  
and endorsed by the **World**  
Wide Web Consortium (W3C)*

The **Semantic Web** is an evolving extension of the World Wide Web, which promotes a new distributed knowledge representation.

# OWL: Goal

RDFS just allows us to define classes and subclasses with very limited inference.

Can we go further?

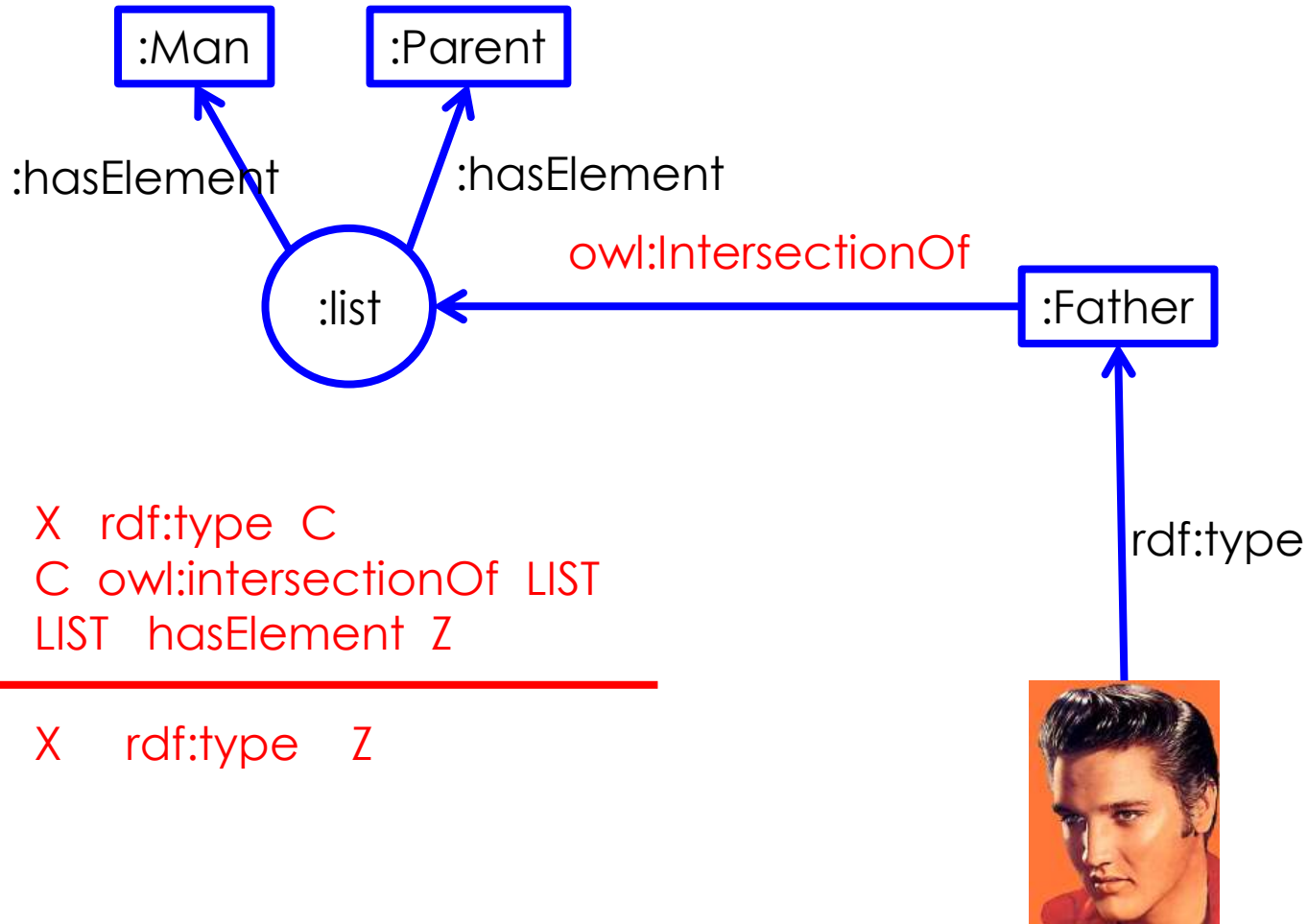
- Reasoning  
If X is left of Y and Y is left of Z  
then X is left of Z
- Class definitions  
The class of husbands is  
the class of married men
- Class properties  
People and tables are two  
disjoint classes



Goal of the  
**Web Ontology Language**  
**(OWL)**

# OWL: Vocabulary

OWL is a namespace that defines predicates with certain semantic rules.



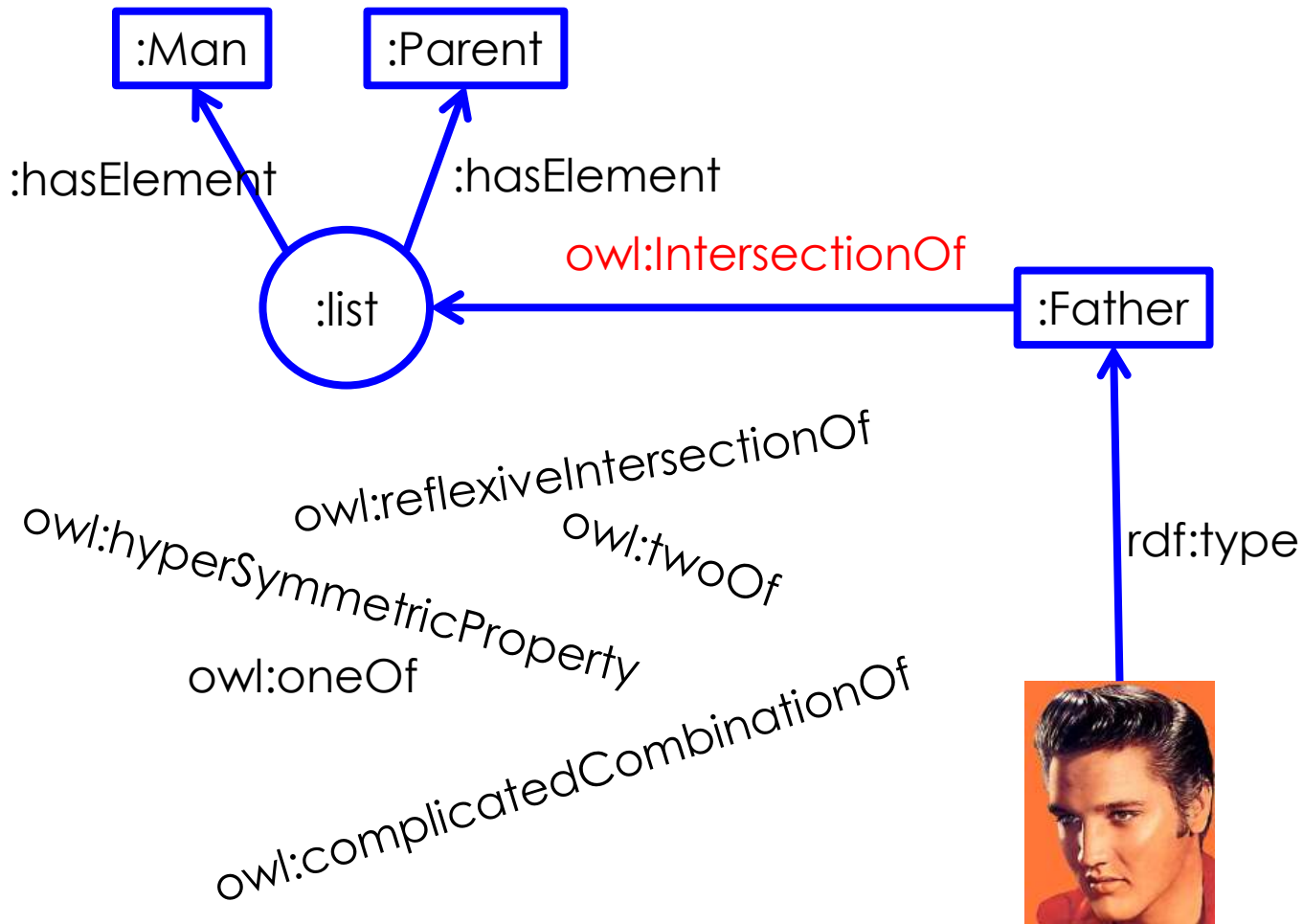
X `rdf:type` C  
C `owl:intersectionOf` LIST  
LIST `hasElement` Z

---

X `rdf:type` Z

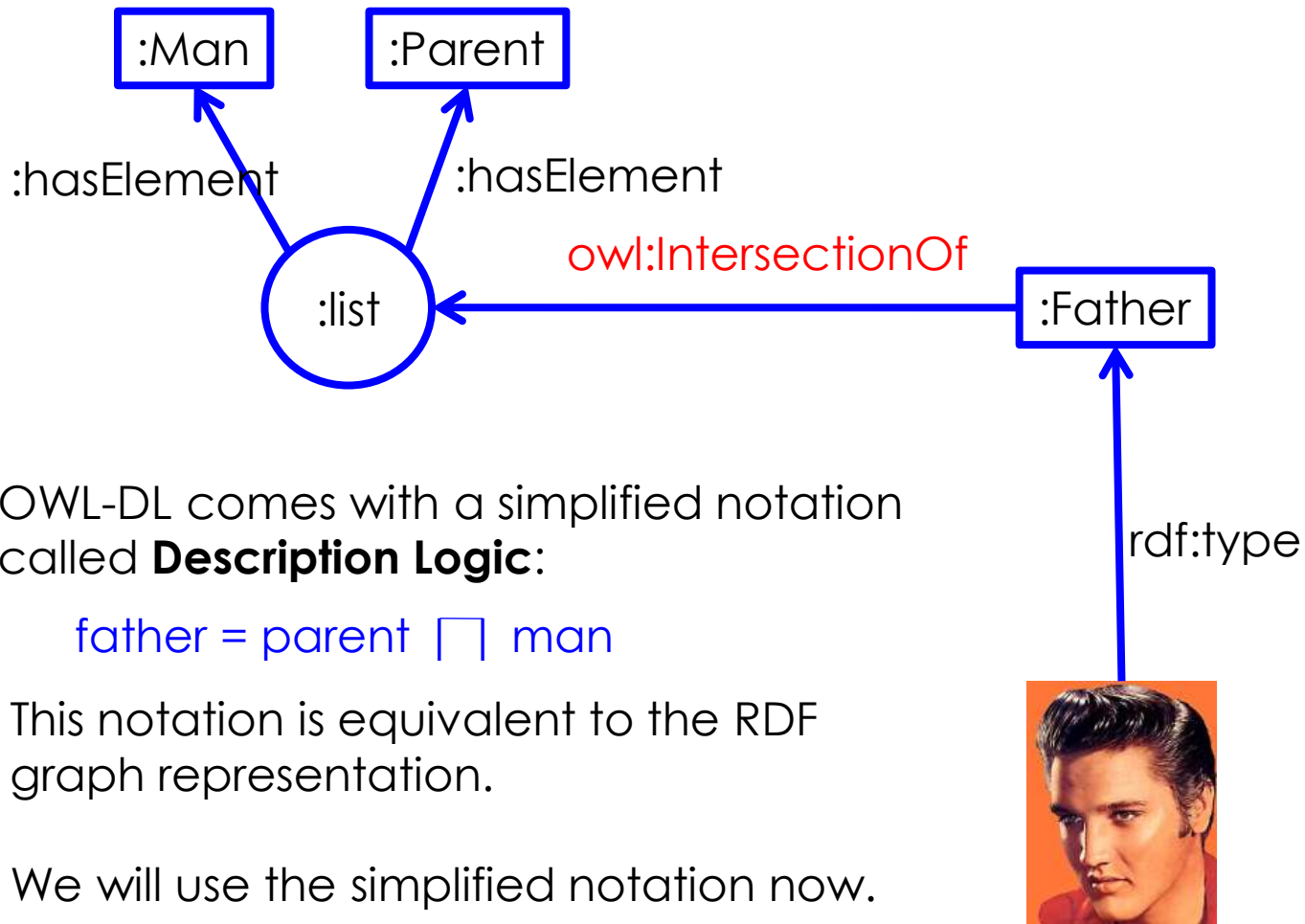
# OWL: Undecidability

OWL is a namespace that defines predicates with certain semantic rules. OWL defines so powerful predicates that it is **undecidable**.



# OWL-DL: Goal

OWL-DL is a subset of OWL that is decidable.



OWL-DL comes with a simplified notation called **Description Logic**:

$\text{father} = \text{parent} \sqcap \text{man}$

This notation is equivalent to the RDF graph representation.

We will use the simplified notation now.

# OWL-DL: Class constructors

Class constructors:

$X \sqcup Y$

The class of things that are in X or in Y

$X \sqcap Y$

The class of things that are in both X and Y

$\sim X$

The class of things that are not in X

Example:

Assume that we have the following classes:

person, parent, hardRockSinger, softRockSinger,  
happyPerson, marriedPerson, malePerson

father = parent  $\sqcap$  malePerson

rockSinger = hardRockSinger  $\sqcup$  softRockSinger

unmarried-rock-singing-father

= (parent  $\sqcap$  man)  $\sqcap$  (hardRockSinger  $\sqcup$  softRockSinger)  $\sqcap$   $\sim$ married

non-rock-singing-person

= person  $\sqcap$   $\sim$ (hardRockSinger  $\sqcup$  softRockSinger)

# OWL: OWL-DL

Class constructors:

$X \sqcap Y$

The class of things that are in both X and Y

$X \sqcup Y$

The class of things that are in X or in Y

$\sim X$

The class of things that are not in X

$\forall R.C$

The class of things where all R-links lead to a C

$\exists R.C$

The class of things where there is a R-link to a C

R: A predicate/role  
C: a class

$\text{has-happy-child} = \exists \text{hasChild.happyPerson}$

This corresponds to the First Order Logic formula:

$\forall x: \text{has-happy-child}(x) \iff \exists y: \text{hasChild}(x,y) \Rightarrow \text{happyPerson}(y)$

# OWL: OWL-DL

Class constructors:

$X \sqcap Y$

The class of things that are in both X and Y

$X \sqcup Y$

The class of things that are in X or in Y

$\sim X$

The class of things that are not in X

$\forall R.C$

The class of things where all R-links lead to a C

$\exists R.C$

The class of things where there is a R-link to a C

R: A predicate/role  
C: a class

`has-only-happy-children =  $\forall$  hasChild.happyPerson`

This corresponds to the First Order Logic formula:

$\forall x: \text{hohc}(x) \iff \forall y: \text{hasChild}(x,y) \Rightarrow \text{happyPerson}(y)$

# OWL: OWL-DL

Class constructors:

$X \sqcap Y$

The class of things that are in both X and Y

$X \sqcup Y$

The class of things that are in X or in Y

$\sim X$

The class of things that are not in X

$\forall R.C$

The class of things where all R-links lead to a C

$\exists R.C$

The class of things where there is a R-link to a C

R: A predicate/role  
C: a class

$\text{person-with-happy-child} = \text{person} \sqcap \exists \text{hasChild.happyPerson}$

$\text{person-with-only-happy-children} = \text{person} \sqcap \forall \text{hasChild.happyPerson}$

# OWL: OWL-DL

Class constructors:

$X \sqcap Y$

The class of things that are in both X and Y

$X \sqcup Y$

The class of things that are in X or in Y

$\sim X$

The class of things that are not in X

$\forall R.C$

The class of things where all R-links lead to a C

$\exists R.C$

The class of things where there is a R-link to a C

Assertions:

$X \sqsubseteq Y$

X is a subclass of Y (everything in X is also in Y)

singer  $\sqsubseteq$  person

This corresponds to the First Order Logic formula:

$\forall x: \text{singer}(x) \Rightarrow \text{person}(x)$

person  $\sqsubseteq$  person  $\sqcap \forall \text{hasChild.happyPerson}$

# OWL: OWL-DL

Class constructors:

$X \sqcap Y$

The class of things that are in both X and Y

$X \sqcup Y$

The class of things that are in X or in Y

$\sim X$

The class of things that are not in X

$\forall R.C$

The class of things where all R-links lead to a C

$\exists R.C$

The class of things where there is a R-link to a C

Assertions:

$X \sqsubseteq Y$

X is a subclass of Y (everything in X is also in Y)

$a:C$

a is a thing in the class C

elvis: singer

This corresponds to the First Order Logic formula:

singer(elvis)

# OWL: OWL-DL

Class constructors:

$X \sqcap Y$

The class of things that are in both X and Y

$X \sqcup Y$

The class of things that are in X or in Y

$\sim X$

The class of things that are not in X

$\forall R.C$

The class of things where all R-links lead to a C

$\exists R.C$

The class of things where there is a R-link to a C

Assertions:

$X \sqsubseteq Y$

X is a subclass of Y (everything in X is also in Y)

$a:C$

a is a thing in the class C

elvis: person  $\sqcap \exists$  hasChild.happyPerson

This corresponds to:

elvis: specialClass

specialClass = person  $\sqcap \exists$  hasChild.happyPerson

# OWL: OWL-DL

Class constructors:

$X \sqcap Y$

The class of things that are in both X and Y

$X \sqcup Y$

The class of things that are in X or in Y

$\sim X$

The class of things that are not in X

$\forall R.C$

The class of things where all R-links lead to a C

$\exists R.C$

The class of things where there is a R-link to a C

Assertions:

$X \sqsubseteq Y$

X is a subclass of Y (everything in X is also in Y)

$a:C$

a is a thing in the class C

$(a,b):R$

a and b stand in the relation R, i.e.,  $R(a,b)$

$(\text{elvis}, \text{lisa}): \text{hasChild}$

This corresponds to:  $\text{hasChild}(\text{elvis}, \text{lisa})$

$(\text{elvis}, \text{priscilla}): \text{marriedTo}$

This corresponds to:  $\text{marriedTo}(\text{elvis}, \text{priscilla})$

# OWL: OWL-DL

Class constructors:

$X \sqcap Y$

The class of things that are in both X and Y

$X \sqcup Y$

The class of things that are in X or in Y

$\sim X$

The class of things that are not in X

$\forall R.C$

The class of things where all R-links lead to a C

$\exists R.C$

The class of things where there is a R-link to a C

Assertions:

$X \sqsubseteq Y$

X is a subclass of Y (everything in X is also in Y)

$a:C$

a is a thing in the class C

$(a,b):R$

a and b stand in the relation R, i.e.,  $R(a,b)$

Examples:

assume the classes: [male](#), [person](#), [happyPerson](#)

and the predicates: [marriedTo](#), [hasChild](#)

- build the class of married people
- build the class of people married to at least one happy person
- build the class of happy male married people
- say that Elvis is such a person

# OWL: OWL-DL

OWL-DL assertions **entail** other assertions:

elvis: singer

singer  $\sqsubseteq$  person

*Elvis is a singer*

*Every singer is a person*

(elvis,lisa): hasChild

lisa: happyPerson

(elvis,priscilla): marriedTo

elvis:  $\forall$  marriedTo.happyPerson

*Elvis has child Lisa*

*Lisa is a happy person*

*Elvis is married to Priscilla*

*All people that Elvis is married to are happy*

---

elvis: person

elvis:  $\exists$  hasChild.happyPerson

priscilla: happyPerson

# OWL: OWL-DL

OWL-DL assertions **entail** other assertions.

Analogue in First Order Logic

elvis: singer  
singer  $\sqsubseteq$  person

singer(elvis)  
 $\forall x: \text{singer}(x) \Rightarrow \text{person}(x)$

(elvis,lisa): hasChild  
lisa: happyPerson

hasChild(elvis,lisa)  
happyPerson(lisa)

---

elvis: person  
elvis:  $\exists$  hasChild.happyPerson

person(elvis)  
 $\exists x: \text{hasChild}(\text{elvis},x) \wedge \text{happyPerson}(x)$

OWL-DL is decidable

Thus, OWL-DL corresponds to a decidable subset of First Order Logic

# OWL: OWL-DL

**OWL-DL** is a decidable KR formalism.

Every OWL-DL statement has an analogue

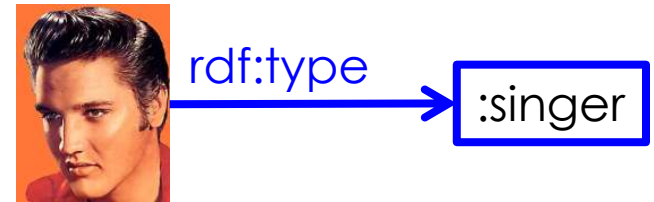
- in first order logic
- and in RDFS

elvis:singer

*Description Logic  
notation*

singer(elvis)

*First Order Logic  
notation*



*RDF graphical notation*

Thus, OWL-DL corresponds to a decidable fragment of First Order Logic and to a decidable fragment of OWL.

There are a number of free OWL DL reasoners available online:

- Pellet
- FaCT++
- Prova

# Semantic Web

The Semantic Web provides KR standards to

- Identify entities (URIs) ✓
- Express facts (RDF) ✓
- Express concepts (RDFS) ✓
- Share vocabularies ✓
- Reason on facts (OWL) ✓

... and it even works.

*These standards are produced  
and endorsed by the **World**  
Wide Web Consortium (W3C)*

The **Semantic Web** is an evolving extension of the World Wide Web, which promotes a new distributed knowledge representation.

# Existing ontologies

There are already hundreds of RDF ontologies on the Web  
( <http://www4.wiwiss.fu-berlin.de/lodcloud/> )

- US census data
- BBC music database
- Gene ontologies
- DBpedia general knowledge (and hub vocabulary), + YAGO, + Cyc etc.
- UK government data
- geographical data in abundance
- national library catalogs (Hungary, USA, Germany etc.)
- publications (DBLP)
- commercial products
- all Pokemons
- ...and many more



# Sigma

**Sigma** is a Semantic Web search engine developed at the DERI Ireland.  
(<http://sig.ma>)



Sigma also allows the user to correct factually wrong information (such as the urban legend that Elvis would be dead).

---

death: 1977-08-16 [7]

---

death date: **hide value** just this value which sources reject sources

---

died: Memphis, Tennessee, [7]

1977-08-16 [7]

United States [7]

---

# Semantic Web

The Semantic Web provides KR standards to

- Identify entities (URIs) ✓
- Express facts (RDF) ✓
- Express concepts (RDFS) ✓
- Share vocabularies ✓
- Reason on facts (OWL) ✓
- ... and it even works. ✓

*These standards are produced  
and endorsed by the **World**  
Wide Web Consortium (W3C)*

The **Semantic Web** is an evolving extension of the World Wide Web, which promotes a new distributed knowledge representation.

# Summary

We have seen RDF and OWL, the knowledge representation formalisms of the Semantic Web

- RDF is a distributed knowledge representation formalism
- OWL-DL is a decidable fragment of First Order Logic
- The Semantic Web already contains a sizeable number of ontologies

