Logic and Knowledge Representation

Knowledge representation, Ontologies, Semantic Web

25 May 2018

Giovanni Sileno gsileno@enst.fr

Télécom ParisTech, Paris-Dauphine University

What is Knowledge?

- **Knowledge** is what we ascribe to an agent to predict his behaviour using principles of rationality.

Newell, A. (1982). The Knowledge Level. Artificial Intelligence, 18(1), 87–127.

What is Knowledge?

- **Knowledge** is what we ascribe to an agent to predict his behaviour using principles of rationality.

  - *example of rationality principle:* If a course of action lead to my goal, I will take that course of action.

Newell, A. (1982). The Knowledge Level. Artificial Intelligence, 18(1), 87–127.

What is Knowledge?

- **Knowledge** is what we ascribe to an agent to predict his behaviour using principles of rationality.

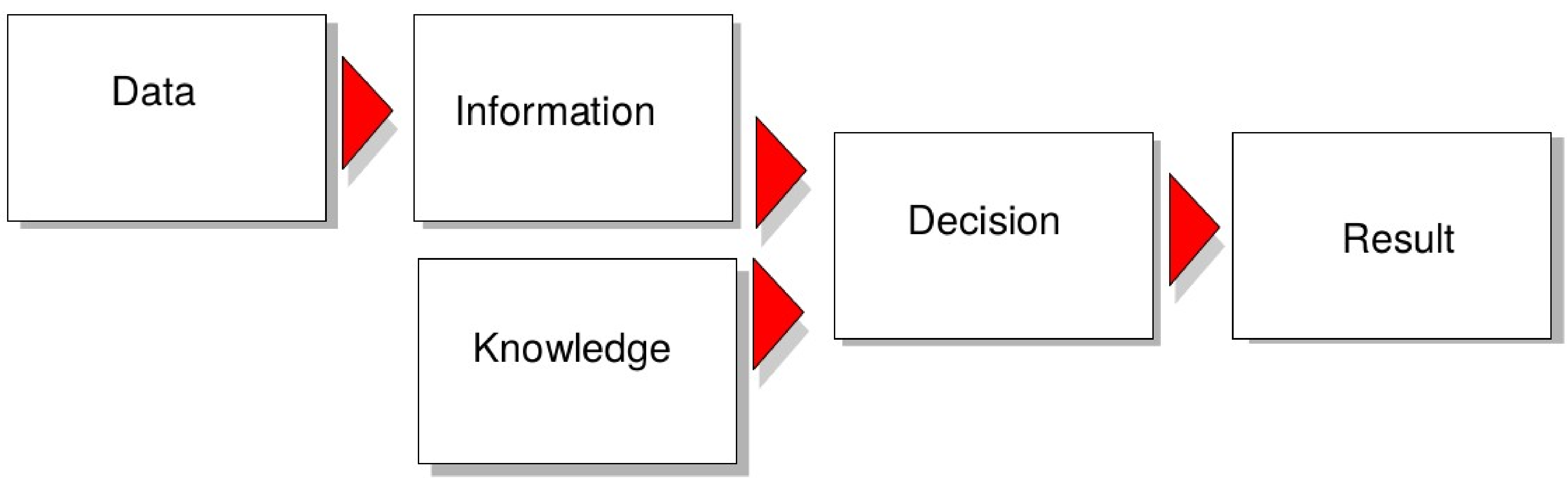
  - *example of rationality principle:* If a course of action lead to my goal, I will take that course of action.

Note: knowledge representation only reproduces that which we ascribe; it is not intended to be accurate, physical model

Newell, A. (1982). The Knowledge Level. Artificial Intelligence, 18(1), 87–127.

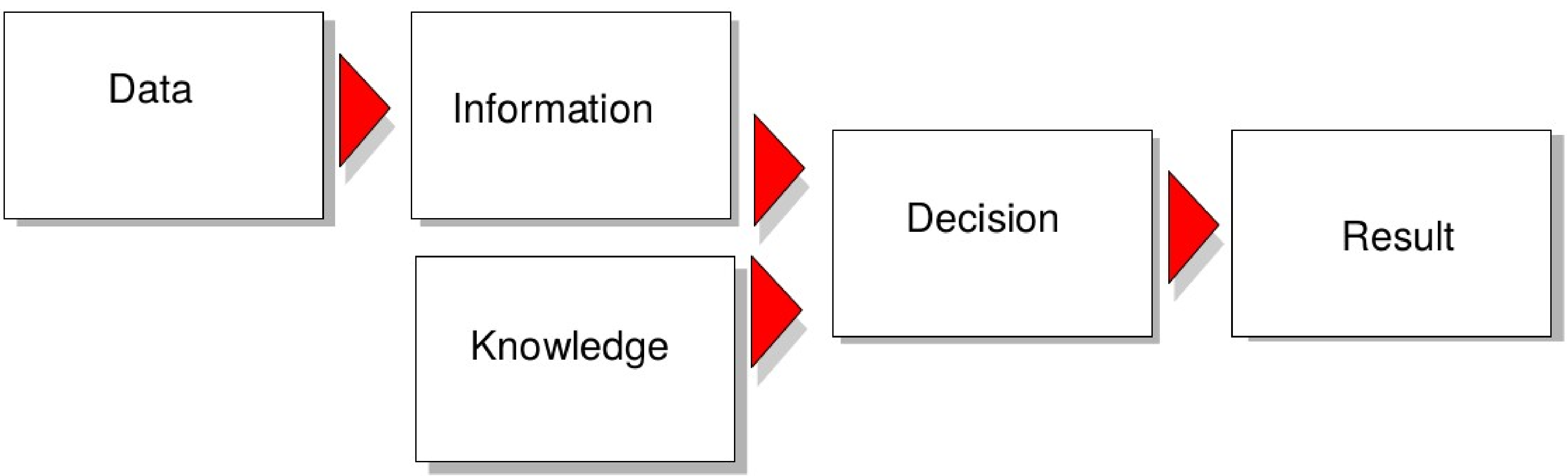
Data, Information, Knowledge

- **Data**: uninterpreted signals or symbols



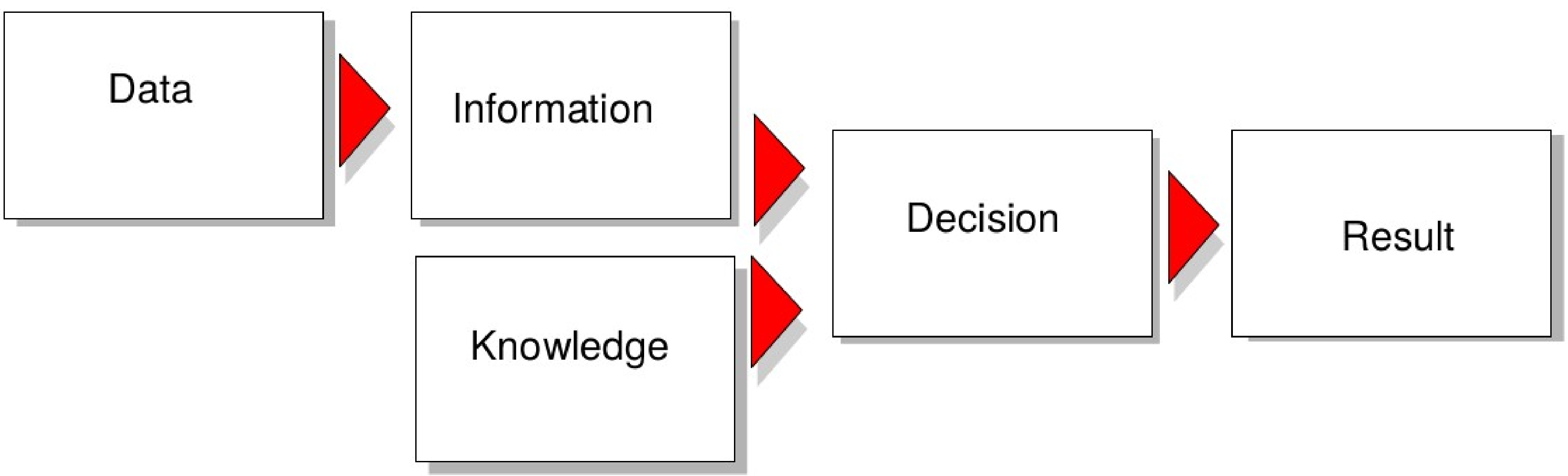
Data, Information, Knowledge

- **Data**: uninterpreted signals or symbols
- **Information**: data with added meaning



Data, Information, Knowledge

- **Data**: uninterpreted signals or symbols
- **Information**: data with added meaning
- **Knowledge:** all data and information that people use to *act, accomplish tasks* and to *create new information* (know-how, -why, -who, -where and -when).



Data, Information, Knowledge

Data: ... - - - ...

Information: it is a message saying S O S

Knowledge: emergency signal, start rescue operation.

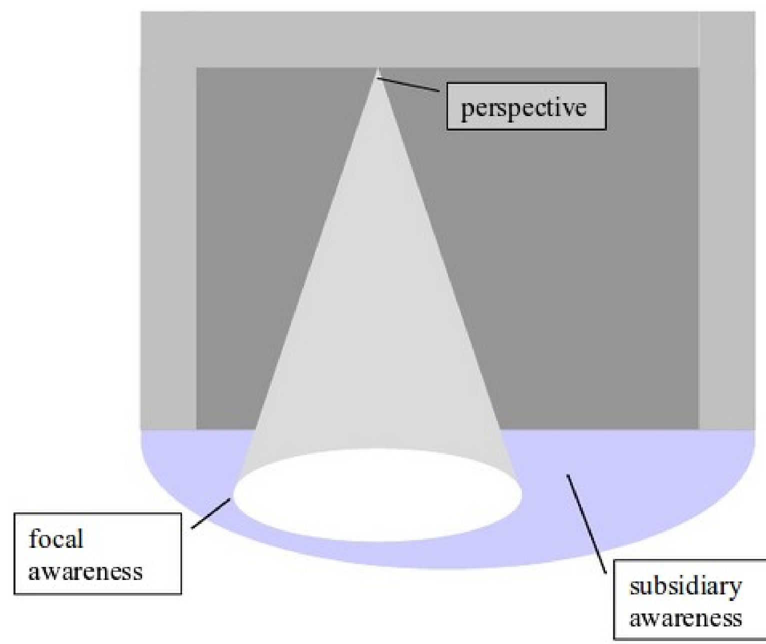
Data: 01 45431200

Information: it is a telephone number of a person

Knowledge: to make an appointment I need to call it

Types of knowledge

- **Explicit,** conscious and external, in focus
- **Implicit,** may be externalized, not in focus
- **Tacit,** often not conscious, internal (Polanyi)



Picture from Brohm, R. (2007). Bringing Polanyi on the Theatre Stage

More types of knowledge

- **Procedural knowledge:** procedures, plans
- **Declarative knowledge:** concepts, objects, facts

More types of knowledge

- **Procedural knowledge:** procedures, plans
- **Declarative knowledge:** concepts, objects, facts
- **Heuristic knowledge:** experience, defaults
- **Knowledge about uncertainty:** probability estimations, defaults, knowing what you know

More types of knowledge

- **Procedural knowledge:** procedures, plans
- **Declarative knowledge:** concepts, objects, facts
- **Heuristic knowledge:** experience, defaults
- **Knowledge about uncertainty:** probability estimations, defaults, knowing what you know
- **Common-sense knowledge:** general concepts and theories, general taxonomies and mereonomies

More types of knowledge

- **Procedural knowledge:** procedures, plans
- **Declarative knowledge:** concepts, objects, facts
- **Heuristic knowledge:** experience, defaults
- **Knowledge about uncertainty:** probability estimations, defaults, knowing what you know
- **Common-sense knowledge:** general concepts and theories, general taxonomies and mereonomies
- **Meta-knowledge:** knowledge about knowledge types and their use

What is Knowledge Representation?

a simplified representation

reifying our attention to the world

and a model of associated reasoning processes

that is accessible to programs

and to people

R. Davis, H. Shrobe, and P. Szolovits. What is a Knowledge Representation? AI Magazine, 14(1):17-33, 1993.

What is Knowledge Representation?

- surrogate

a simplified representation

- expression of ontological commitment

reifying our attention to the world

- theory of intelligent reasoning

and a model of associated reasoning processes

- medium of efficient computation

that is accessible to programs

- medium of human expression

and to people

R. Davis, H. Shrobe, and P. Szolovits. What is a Knowledge Representation? AI Magazine, 14(1):17-33, 1993.

Knowledge systems

Example of expert system

if flower **and** seed **then** phanerogam
if phanerogam **and** bare-seed **then** fir
if phanerogam **and** 1-cotyledon **then** monocotyledonous
if phanerogam **and** 2-cotyledon **then** dicotyledonous
if monocotyledon **and** rhizome **then** thrush
if dicotyledon **then** anemone
if monocotyledon **and** ¬rhizome **then** lilac
if leaf **and** flower **then** cryptogamous
if cryptogamous **and** ¬root **then** foam
if cryptogamous **and** root **then** fern
if ¬leaf **and** plant **then** thallophyte
if thallophyte **and** chlorophyll **then** algae
if thallophyte **and** ¬ chlorophyll **then** fungus
if ¬leaf **and** ¬flower **and** ¬plant **then** colibacille

rhizome + flower + seed + 1-cotyledon ?

From expert systems to KBS

- **Expert systems**

  Separate knowledge (rules) from the reasoning engine

From expert systems to KBS

- **Expert systems**
  Separate knowledge (rules) from the reasoning engine

- **Knowledge-based systems**
  Separate knowledge (concepts) from rules and reasoning
  - example: ***Frames***
    - stereotyped structures of knowledge

From expert systems to KBS

- **Expert systems**
  Separate knowledge (rules) from the reasoning engine

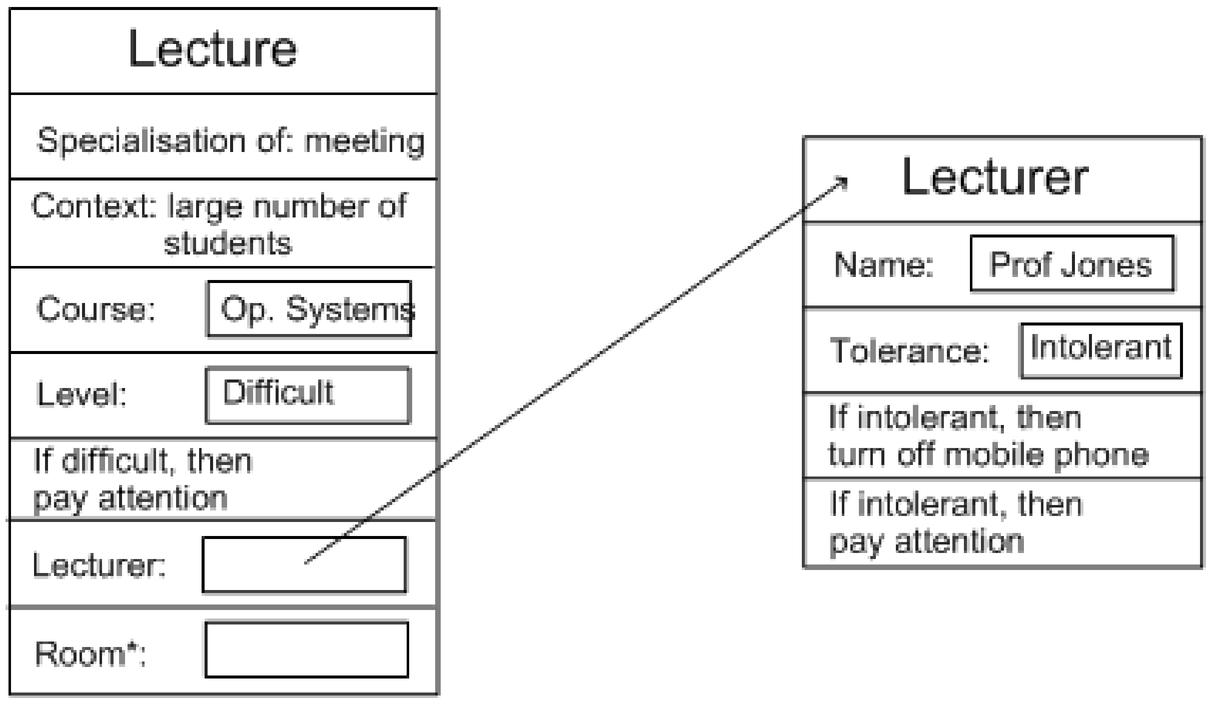
- **Knowledge–based systems**
  Separate knowledge (concepts) from rules and reasoning

  – example: ***Frames***
    - stereotyped structures of knowledge

  – example: ***Semantic networks***
    - representation by graph-based formalism
    - model entities and their relations

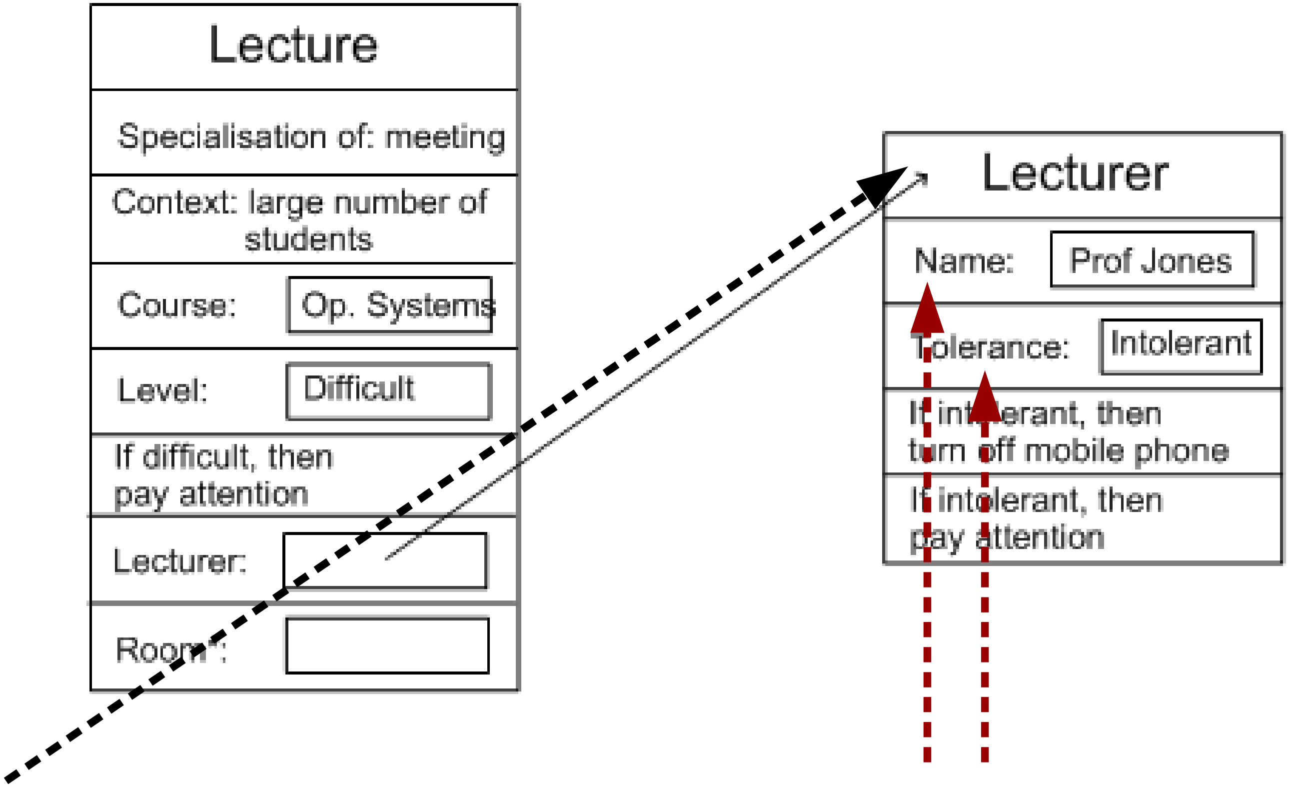
Frames

- Frames are "stereotyped" knowledge units representing situations, objects or events or (classes) sets of such entities.



Frames

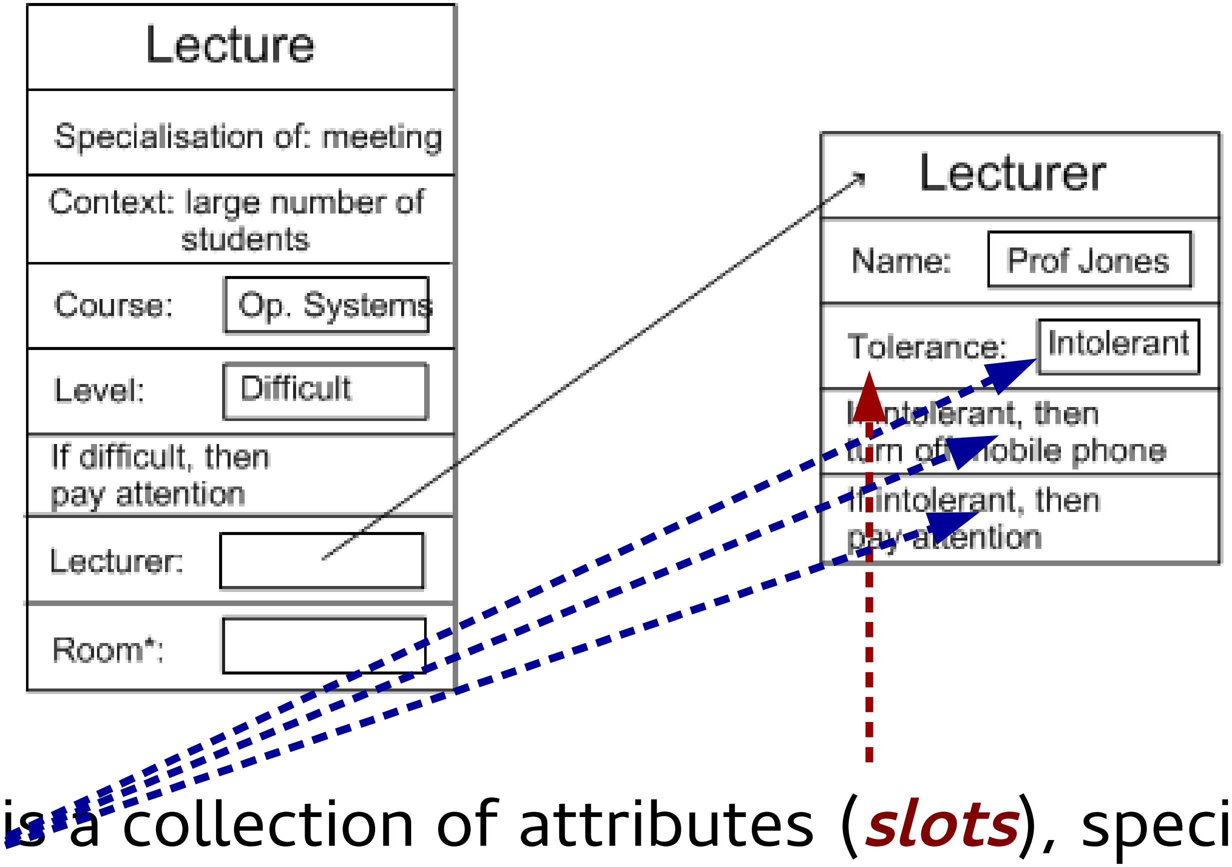
- Frames are "stereotyped" knowledge units representing situations, objects or events or (classes) sets of such entities.



- A frame is a collection of attributes (***slots***), specified by ***facets*** that correspond to the **values** they acquire or **procedures** that launch.

Frames

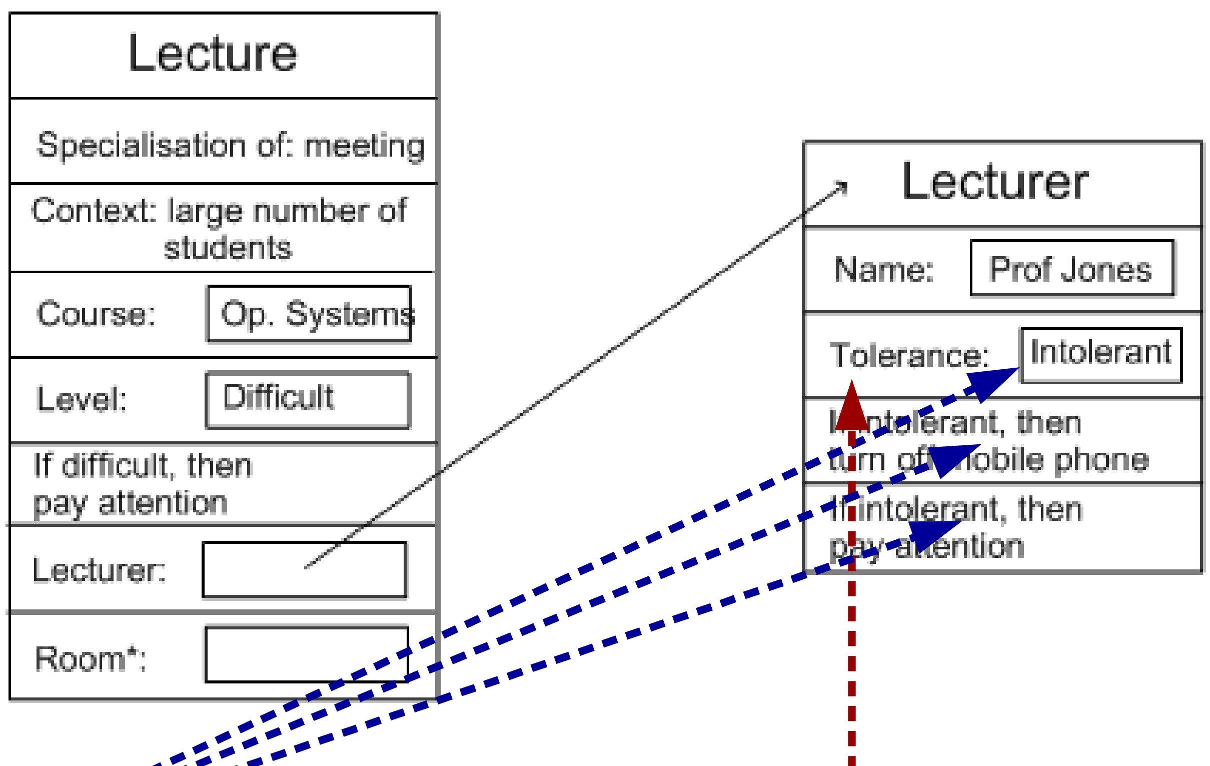
- Frames are "stereotyped" knowledge units representing situations, objects or events or (classes) sets of such entities.



- A frame is a collection of attributes (***slots***), specified by ***facets*** that correspond to the **values** they acquire or **procedures** that launch.

Frames

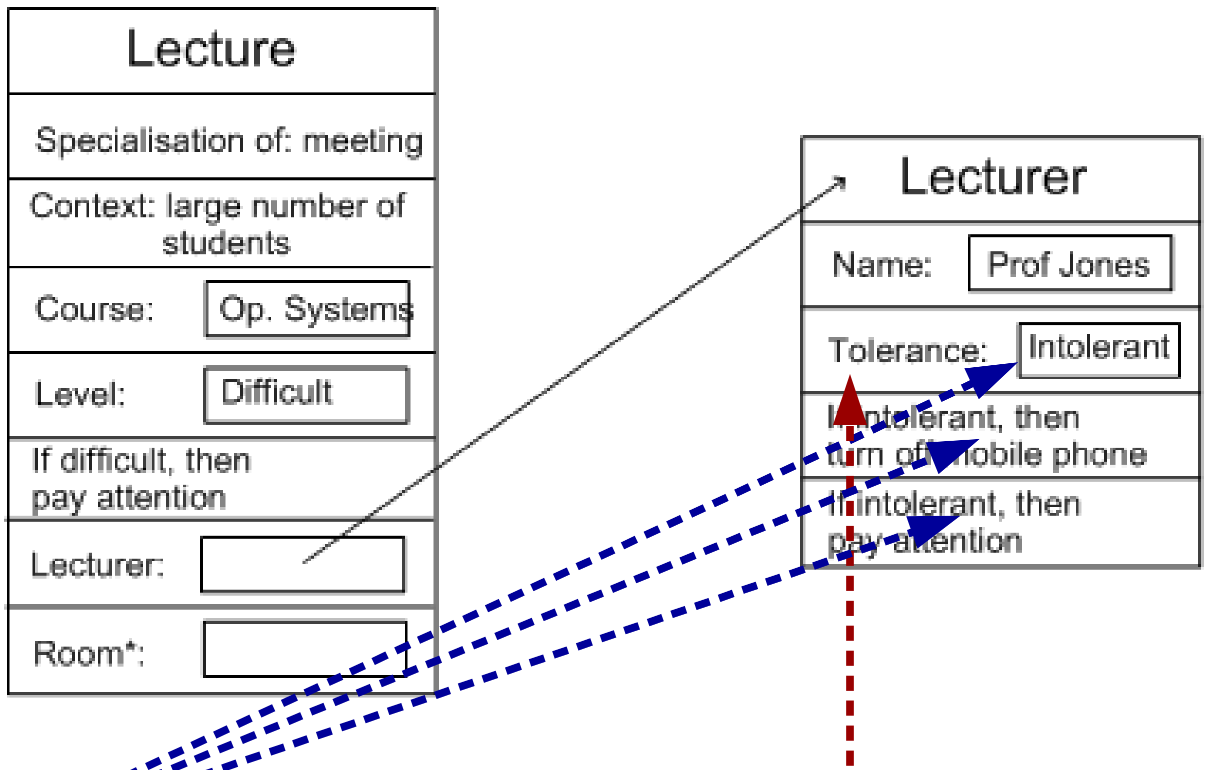
- Frames are "stereotyped" knowledge units representing situations, objects or events or (classes) sets of such entities.



- A frame is a collection of attributes (***slots***), specified by ***facets*** that correspond to the **values** they acquire or **procedures** that launch.

Frames

- Frames are "stereotyped" knowledge units representing situations, objects or events or (classes) sets of such entities.



- A frame is a collection of attributes (***slots***), specified by ***facets*** that correspond to the **values** they acquire or **procedures** that launch.

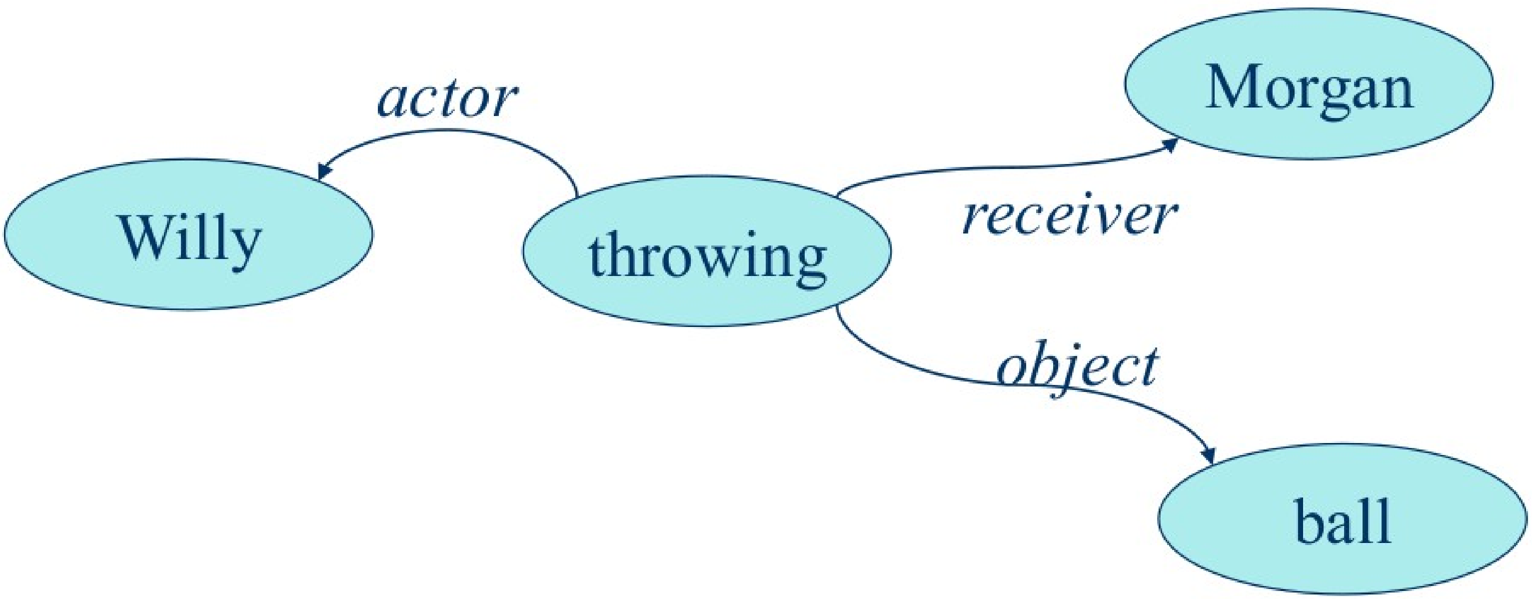
Towards semantic networks

- Rather then focusing on the "object" aspect, we could focus on the predication aspect, just as we do with language.

  ~ objects constituted by what we can say about them

"Willy threw a ball to Morgan."

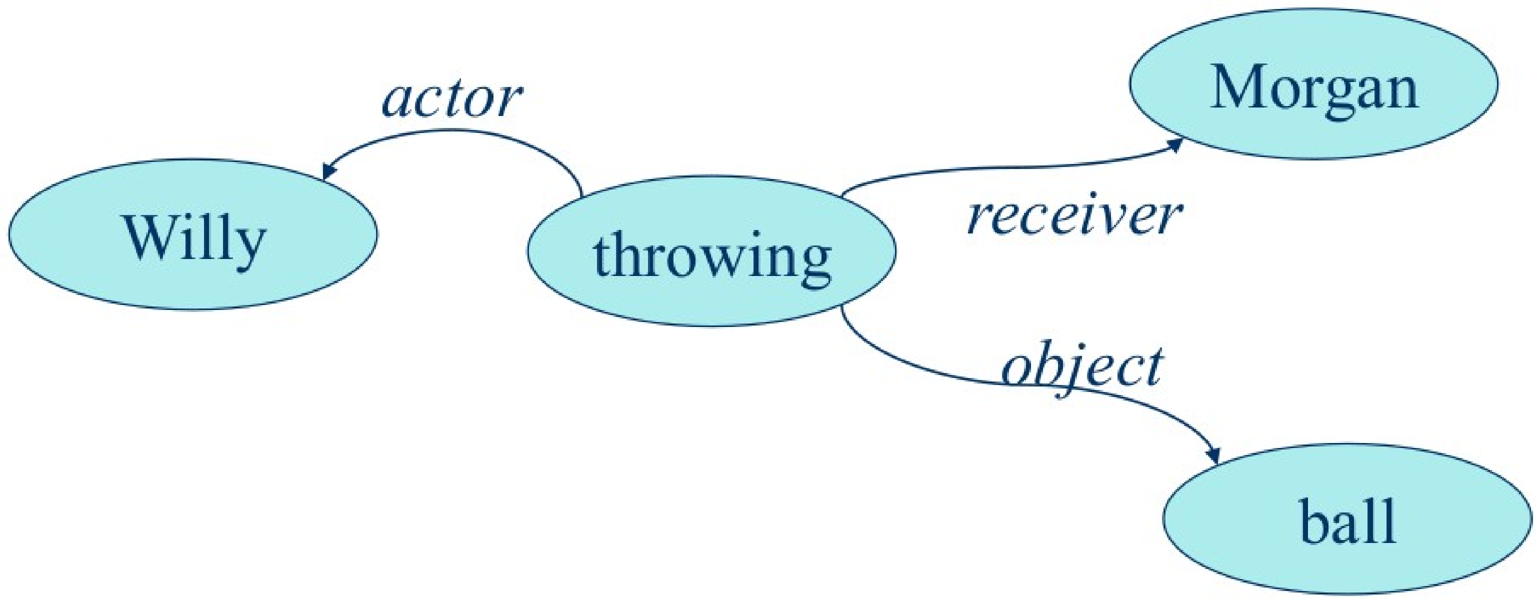
"Willy threw a ball to Morgan."



nodes: **entities**

arcs: **relationships**

"Willy threw a ball to Morgan."



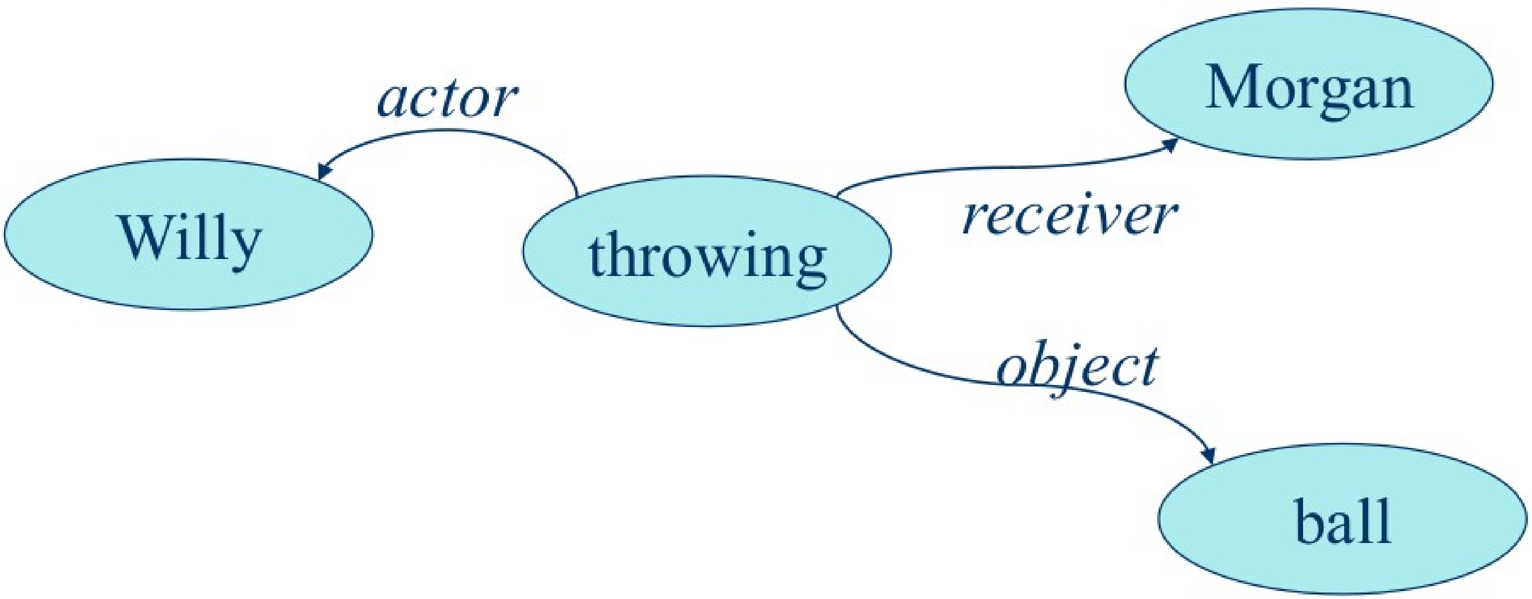
nodes: **entities**

arcs: **relationships**

*labels on nodes: entity **names***

*labels on arcs: relation **names***

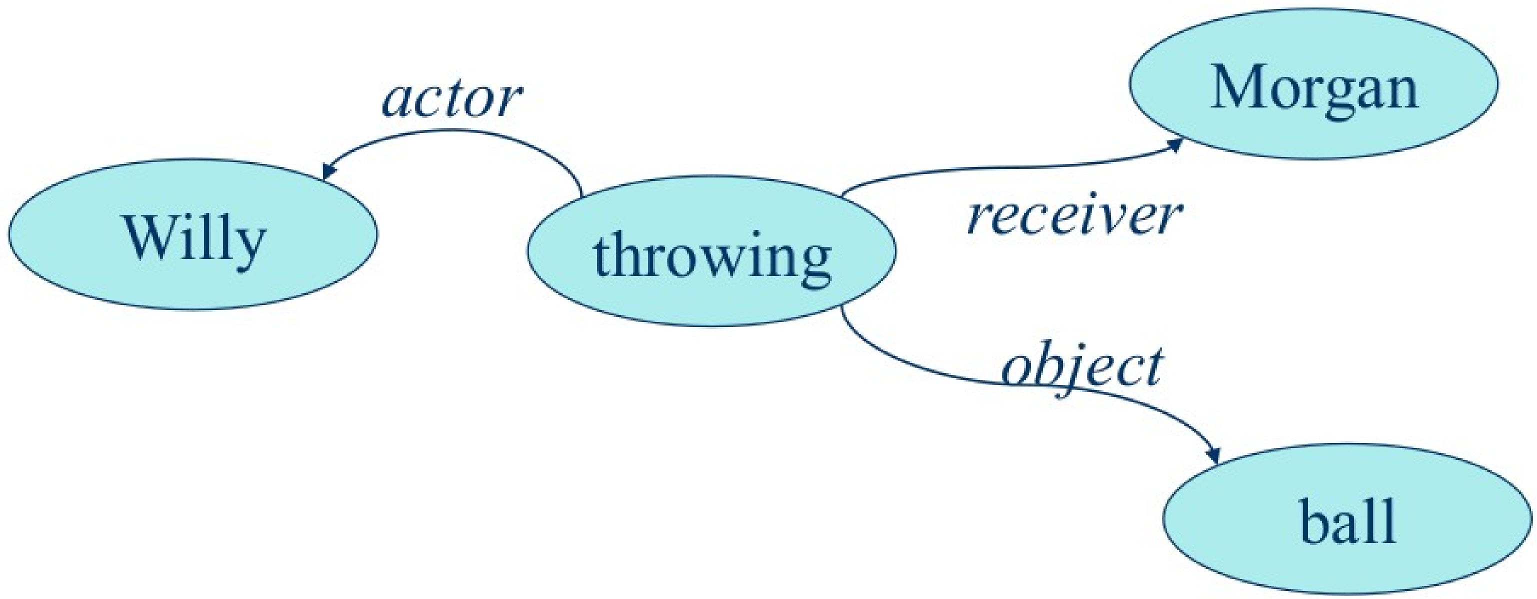
"Willy threw a ball to Morgan."




```
fact( throwing, actor, willy ).       % or: throwing( actor, willy ).
fact( throwing, receiver, morgan ). % or: throwing( receiver, morgan ).
fact( throwing, object, ball ).       % or: throwing( object, ball ).

who_action_what( Who, Act, What) :-
    fact( Act, actor, Who ),
    fact( Act, object, What ).
```

```
?- fact( throwing, X, willy).
?- who_action_what(willy,
throwing, ball).
```

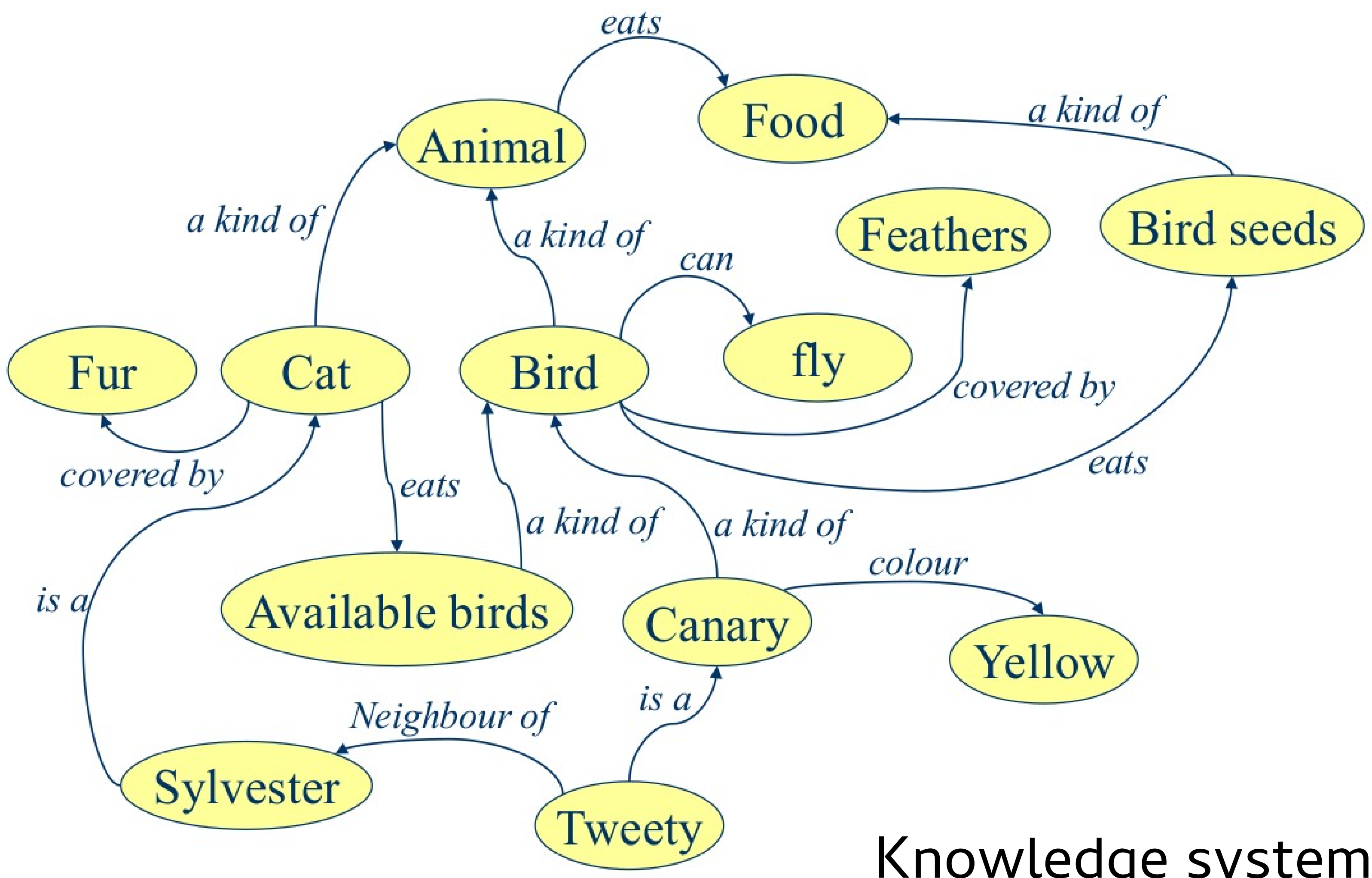
We need more knowledge to infer something more interesting!




```
fact( throwing, actor, willy ).       % or: throwing( actor, willy ).
fact( throwing, receiver, morgan ). % or: throwing( receiver, morgan ).
fact( throwing, object, ball ).       % or: throwing( object, ball ).

who_action_what( Who, Act, What) :-
    fact( Act, actor, Who ),
    fact( Act, object, What ).
```

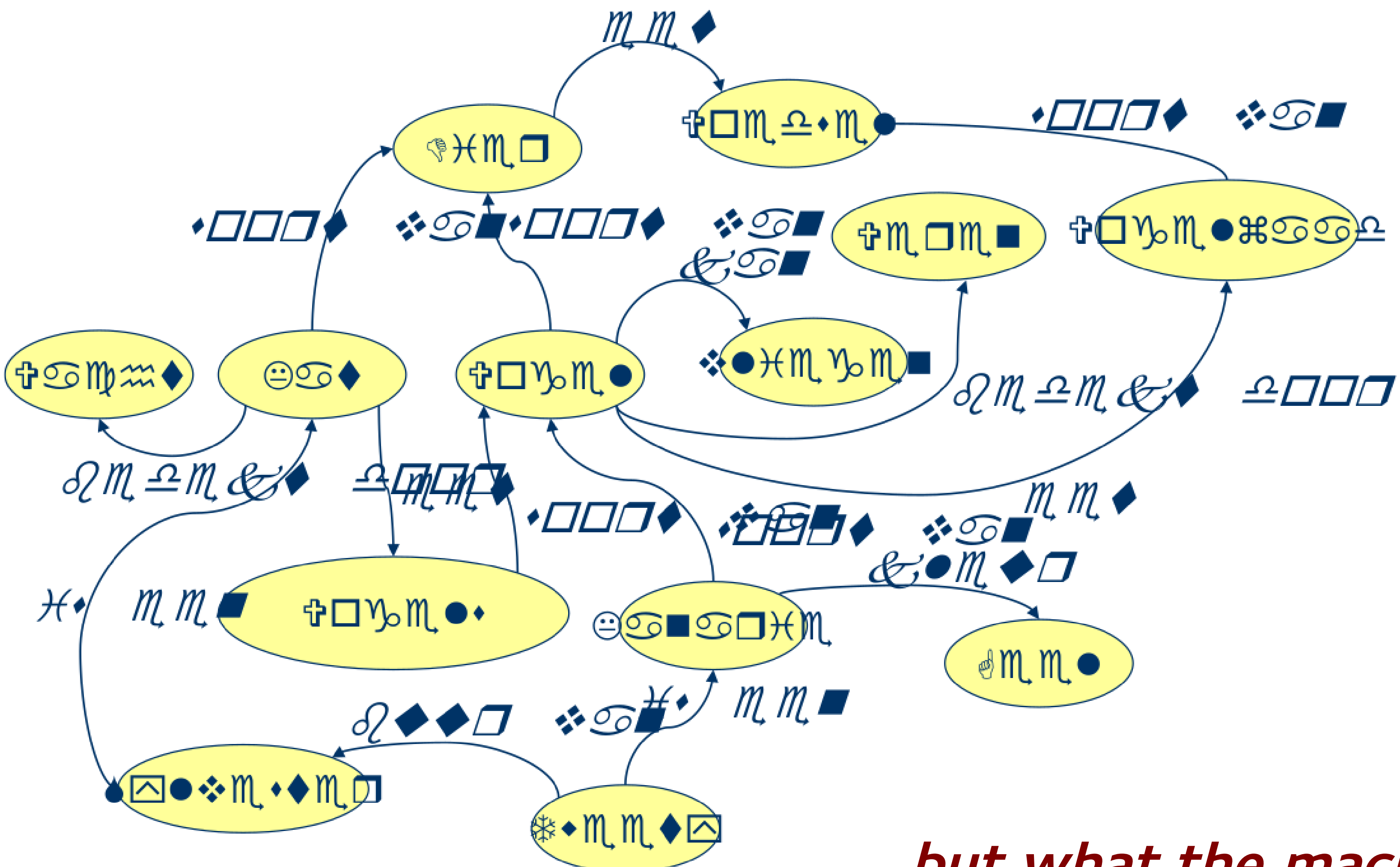
```
?- fact( throwing, X, willy).
?- who_action_what(willy,
throwing, ball).
```

Semantic Networks



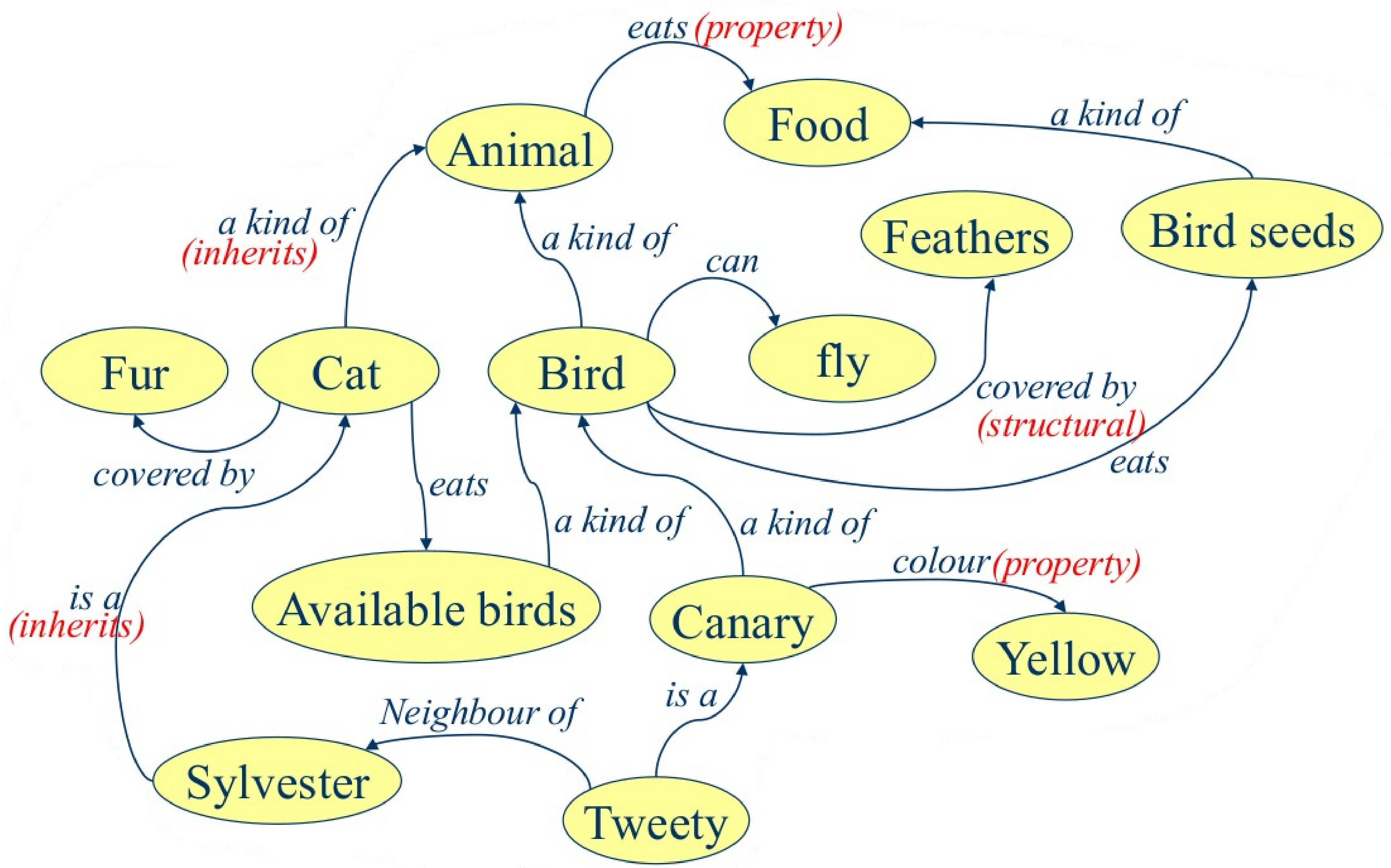
Knowledge systems, inspired by human cognition, aim to be reusable and efficient...

Semantic Networks



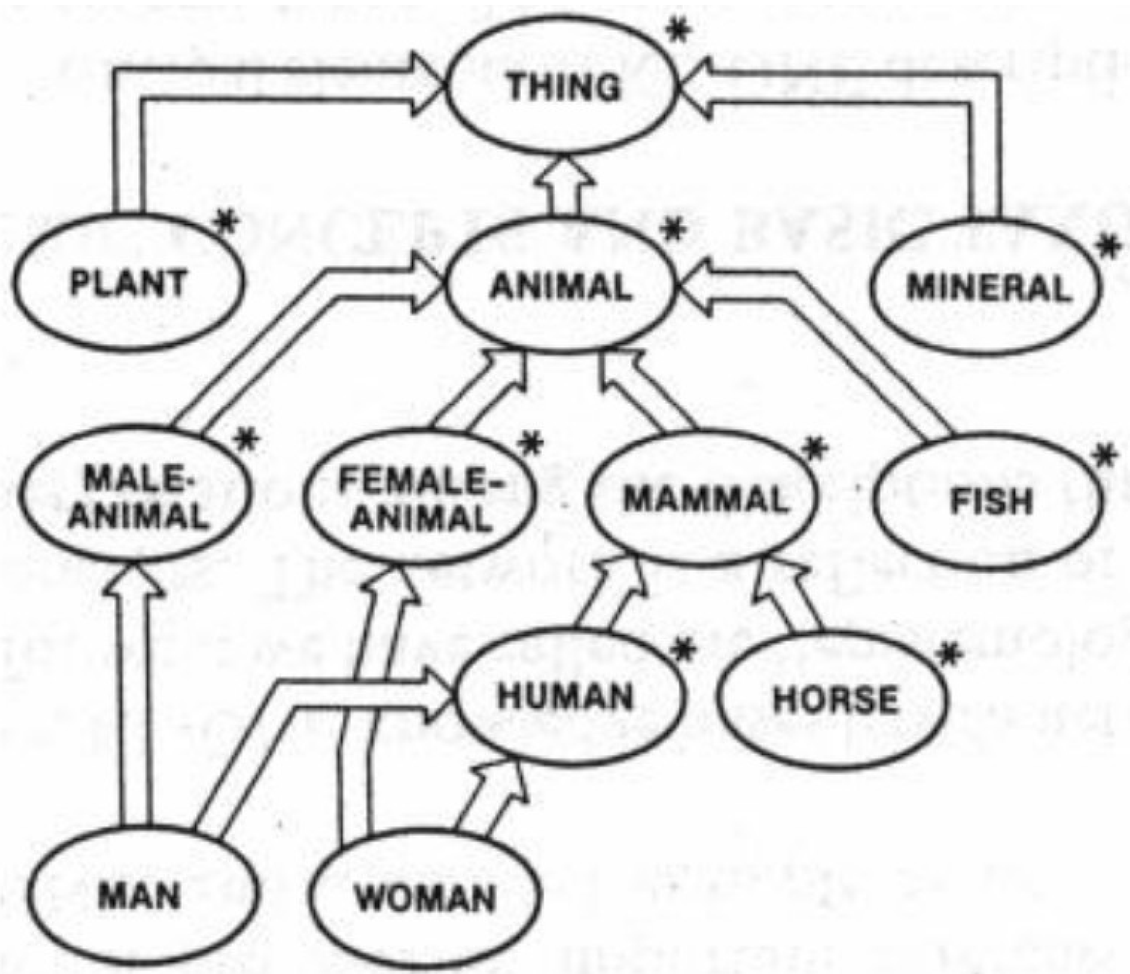
...but what the machine reads is not what we read!

Semantic Networks



Annotation principle: differences in intended processing should be reflected in differences in the symbol structures

First encounter between semantic networks and logic: KL-ONE

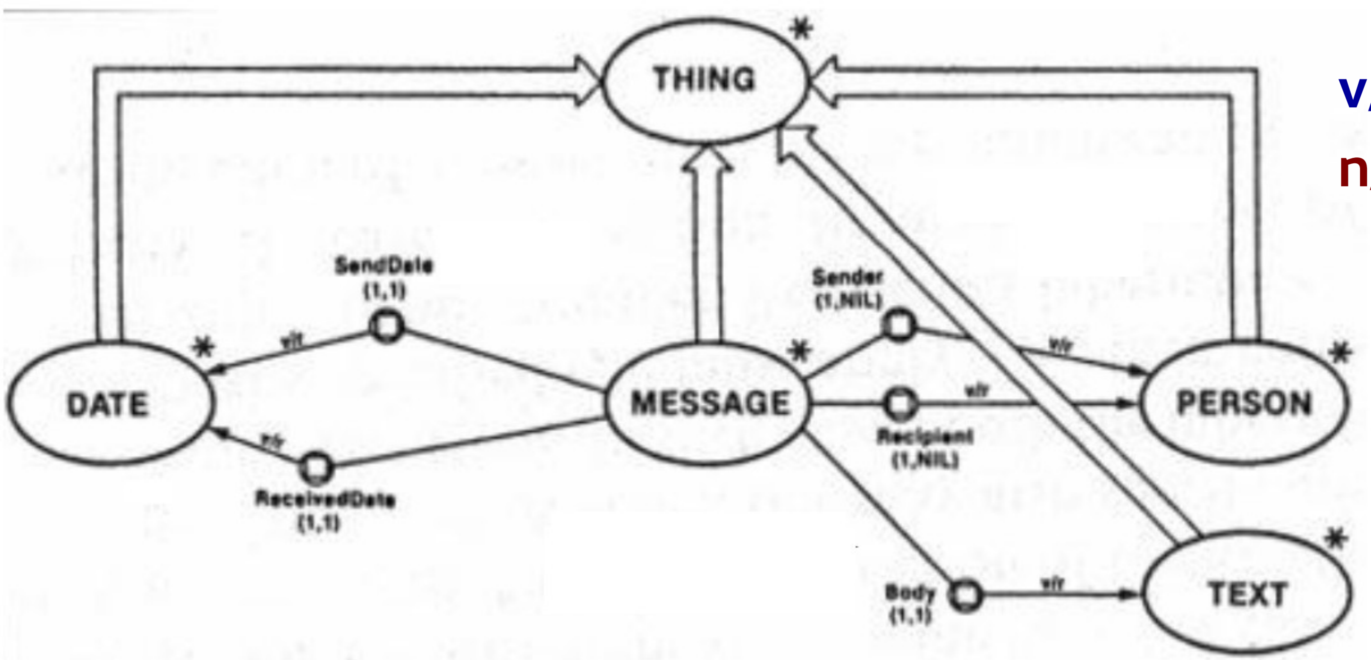


Primitive concepts (*) do not have sufficient conditions, may have necessary

Derived concepts specified by sufficient and necessary conditions.

Brachman, R. J., & Schmolze, J. (1985). An Overview of the KL-ONE Knowledge Representation System. Cognitive Science, 9, 171–216.

First encounter between semantic networks and logic: KL-ONE



v/r: value restriction

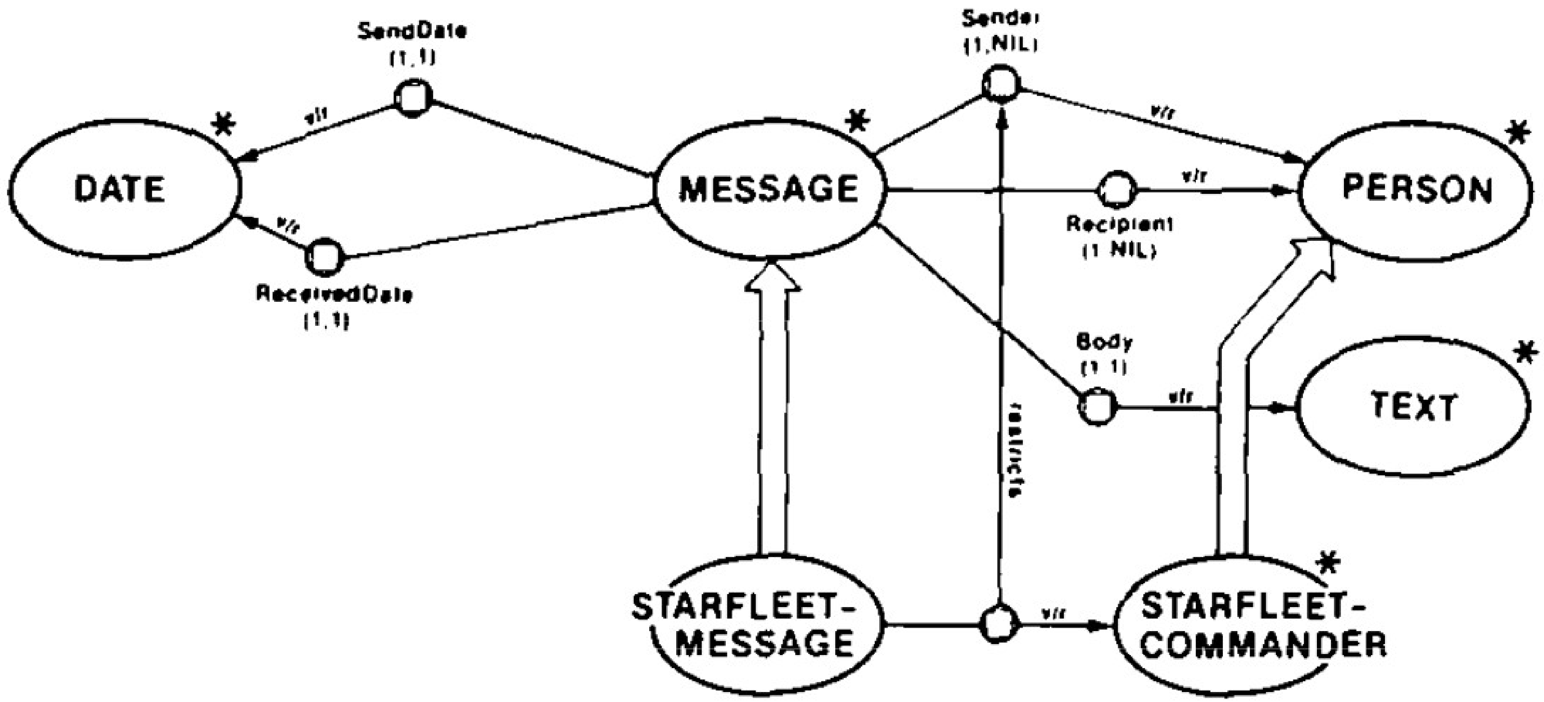
n/r: number restriction

(min, max), NIL = ∞

`A MESSAGE is a THING with at least one Sender, all of which are PERSONs, at least one Recipient, all of which are PERSONs, exactly one Body, which is a TEXT, exactly one SendDate, which is a DATE, and exactly one ReceivedDate, which is a DATE.`

Brachman, R. J., & Schmolze, J. (1985). An Overview of the KL-ONE Knowledge Representation System. Cognitive Science, 9, 171–216.

First encounter between semantic networks and logic: KL-ONE



role restriction through v/r

`A STARFLEET-MESSAGE is a MESSAGE, all of whose Senders are STARFLEET-COMMANDERS.`

Brachman, R. J., & Schmolze, J. (1985). An Overview of the KL-ONE Knowledge Representation System. Cognitive Science, 9, 171–216.

First encounter between semantic networks and logic: KL-ONE

- KL-ONE is an automatic **classifier**
  - takes a new Concept and automatically determines all subsumption relations (is-a) between it and all other Concepts in the network
  - adds new links when new subsumption relations are discovered
  - automates the placement of new Concepts in the taxonomy
  - It is ***sound*** (all found subsumption relations are legitimate) but not ***complete*** (it does not find all subsumption relations)

- basis for OWL (giving semantics to the Semantic Web)

Differences

Prolog

special purpose reasoning engine

- closed world assumption
- negation as failure (NAF)
- only sufficient conditions
- no true existential quantification
- programmer prevents infinite loops

KL-ONE and OWL

general purpose knowledge manipulation

- open world assumption
- no or strong negation
- at least necessary, optionally sufficient conditions
- infinite loops should not be possible

Qualifications of KR

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).
alive(Elvis, true).
vivant(Elvis).
```

- Canonicity is improved by
  - restricting the formalism (e.g. only unary predicates)
  - providing guidelines (e.g. proper name in upper case)
  - using standard vocabularies (e.g. {alive, dead})

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 **>** **Propositional Logic**

$\forall x: man(x) \supset mortal(x)$?

Decidability

- A KR formalism is decidable, if there is an algorithm that can answer any query on a knowledge base in that formalism.

- Typically, the more expressive a formalism, the more likely it is undecidable.

Decidability

- A KR formalism is decidable, if there is an algorithm that can answer any query on a knowledge base in that formalism.

- A formalism can be made decidable by restricting it.
  - propositional logic is decidable
  - FOL is decidable if all formulas are in this form:

$\exists$x, y,....	$\forall$z, q,...	:	p(x,y) ... => ...
existential quantifiers	universal quantifiers		arbitrary formula without quantifiers

Closed and Open World Assumptions

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

Example, UFOs do not exist!

If *it is not the case*

that ufos exist,

then *it is the case*

that ufos do *not* exist.

Closed and Open World Assumptions

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

- Sometimes the open world assumption (OWA) is more appropriate. A statement can then be:
  - ***provable false***
  - ***provable true***
  - ***unknown***

Unique Name Assumption (UNA)

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

Unique Name Assumption (UNA)

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

Canada [edit]

- Paris, Ontario, a community
- Paris, Yukon, a former community

United States [edit]

- Paris, Arkansas, a city
- Paris, Idaho, a city
- Paris, Illinois, a city
- Paris, Indiana, an unincorporated community
- Paris, Iowa, an unincorporated community
- Paris, Kentucky, a city
- Paris, Maine, a town
- Paris, an unincorporated community in Green Charter Township, Michigan
- Paris, Mississippi, an unincorporated community
- Paris, Missouri, a city
- Paris, New Hampshire, an unincorporated community
- Paris, New York, a town

Schemas

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

Schemas

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

- In schema-bound formalisms, one has to decide a priori for classes of things and their properties: a schema-bound formalism puts more modeling constraints, but can exclude non-sensible statements.

- Prolog is schema-free, any entity can have any property.

Schemas

- Databases are a particular schema-bound KR formalism.
- A database can be seen as a set of tables.

each table corresponds to one class of things

each column corresponds to a property

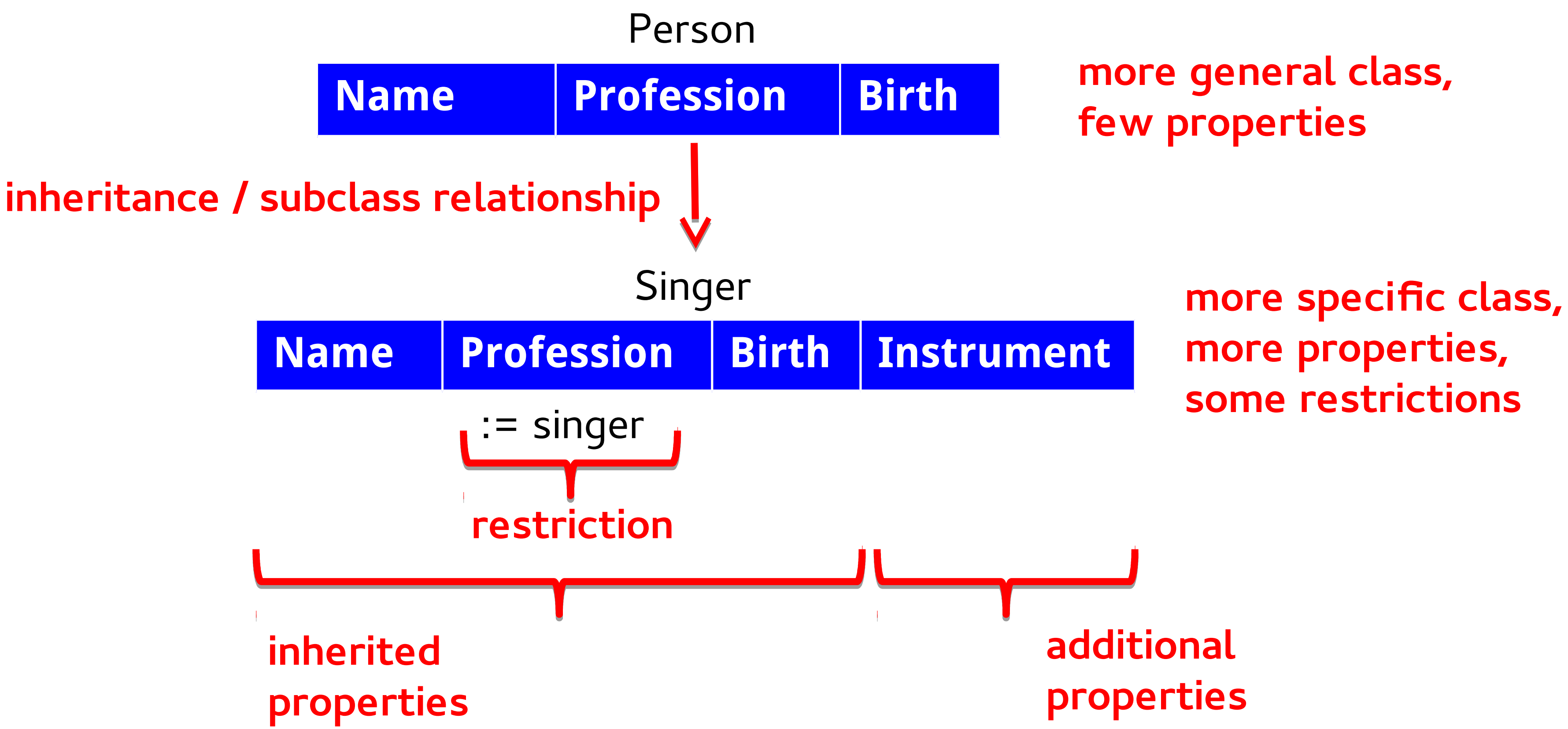
Name	Profession	Birth
Elvis	Singer	1935
Obama	President	1961
...	...	...

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

each row corresponds to a thing

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.



Monotonicity and non-monotonicity

- A KR formalism is monotonic, if adding new knowledge does not undo deduced facts.

Monotonicity and non-monotonicity

- A KR formalism is monotonic, if adding new knowledge does not undo deduced facts.
- **First order logic and propositional logic are monotonic.**

subset

elvis_is_person elvis_is_alive elvis_is_dead => ~ elvis_is_alive	$\subseteq$	elvis_is_person elvis_is_alive elvis_is_dead => ~ elvis_is_alive + elvis_is_dead
=> elvis_is_alive	$\subseteq$	=> elvis_is_alive => elvis_is_dead => michael_jackson_alive

- Monotonicity can be counter-intuitive. It requires to know everything up-front.

Monotonicity and non-monotonicity

- A KR formalism is monotonic, if adding new knowledge does not undo deduced facts.
- **Default logic is not monotonic.**

elvis_is_person

justification

prerequisite
conclusion

$$\frac{\texttt{elvis\_is\_dead}}{\texttt{\~{}elvis\_is\_alive}}$$

if Elvis is dead
then he is not alive

$$\frac{\texttt{elvis\_is\_person: elvis\_is\_alive}}{\texttt{elvis\_is\_alive}}$$

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

\+ elvis_is_dead

=> ~~elvis_is_alive~~

Distributedness

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

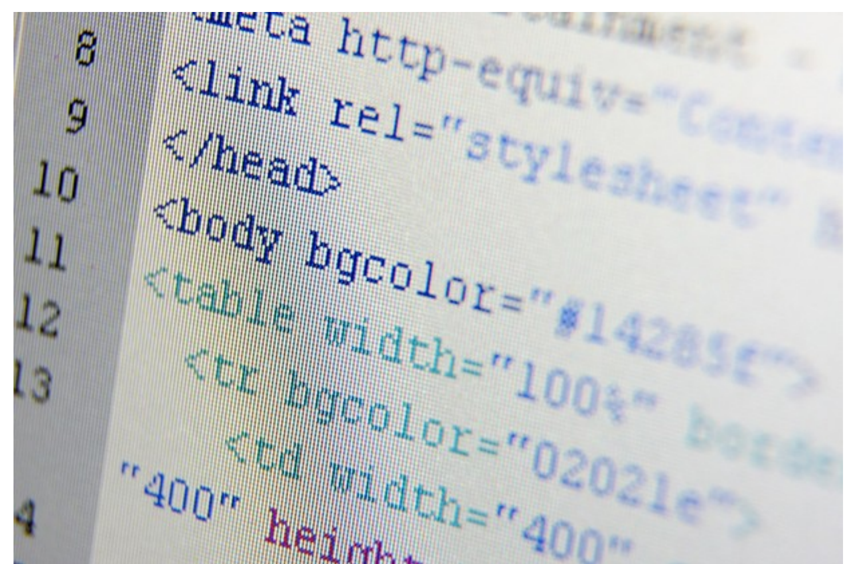
Semantic Web

What's in a web page?

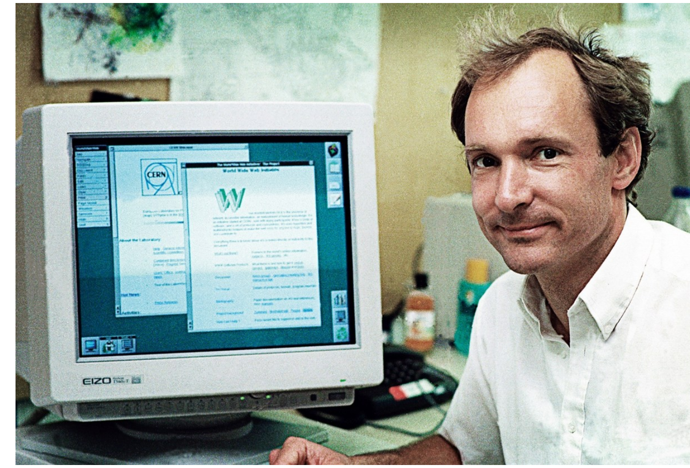
- Textual content, markup and embedded media

- The typical markup consists of:
  - hyper-links to related content,
  - rendering information (pagination, font size and colour, ...)

- The semantic content is accessible to humans but not directly to computers...



The Web was designed as an information space, with the goal that it should be useful not only for *human-human* communication, but also that *machines* would be able to participate and help.



One of the major obstacles to this has been the fact that **most information on the Web is designed for human consumption**, and even if it was derived from a database with well defined meanings (in at least some terms) for its columns, that the structure of the data is not evident to a robot browsing the Web. Leaving aside the artificial intelligence problem of training machines to behave like people, the **Semantic Web** approach instead **develops languages for expressing information in a machine processable form**.

Tim Berners-Lee, The Semantic Web Roadmap.

How to add meaning for machines?

- External agreement on meaning of annotations
  - Problems with this approach
    - Inflexible
    - Limited number of things can be expressed

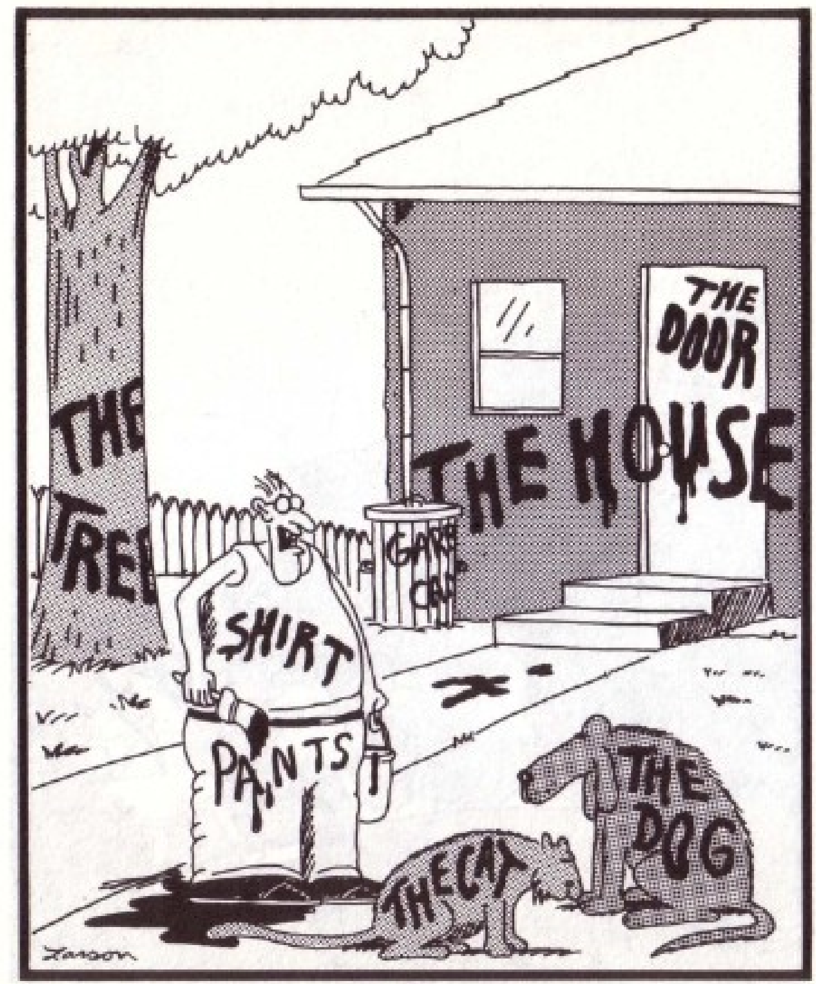
How to add meaning for machines?

- External agreement on meaning of annotations
  - Problems with this approach
    - Inflexible
    - Limited number of things can be expressed

- Use formal vocabularies or *ontologies* to specify meaning of annotations
  - ontologies provide a vocabulary of terms that are machine understandable
  - new terms can be formed by combining existing ones as a kind of "conceptual Lego"
  - meaning (semantics) of such terms is formally specified

Four principles towards a Semantic Web of Data

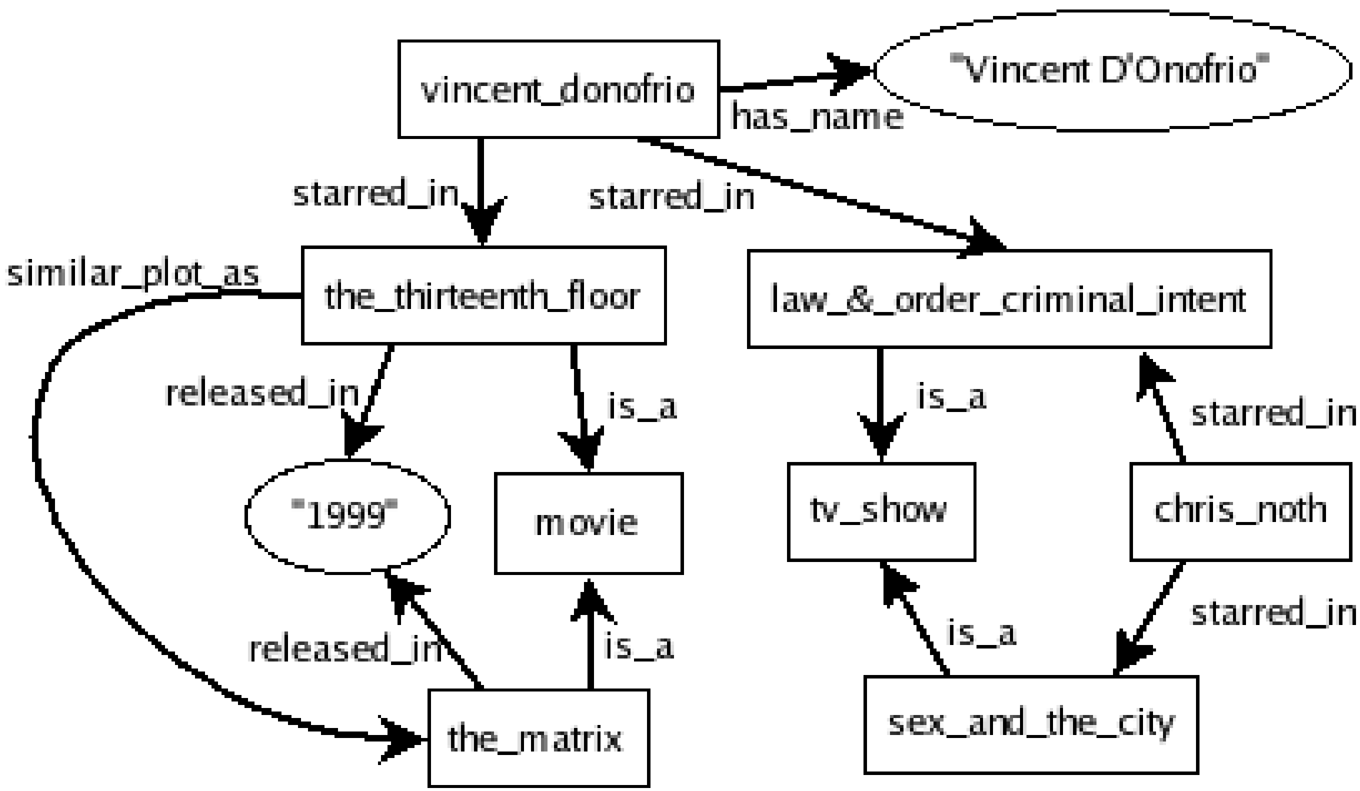
- P1: give all things a name



"Now! ... *That* should clear up a few things around here!"

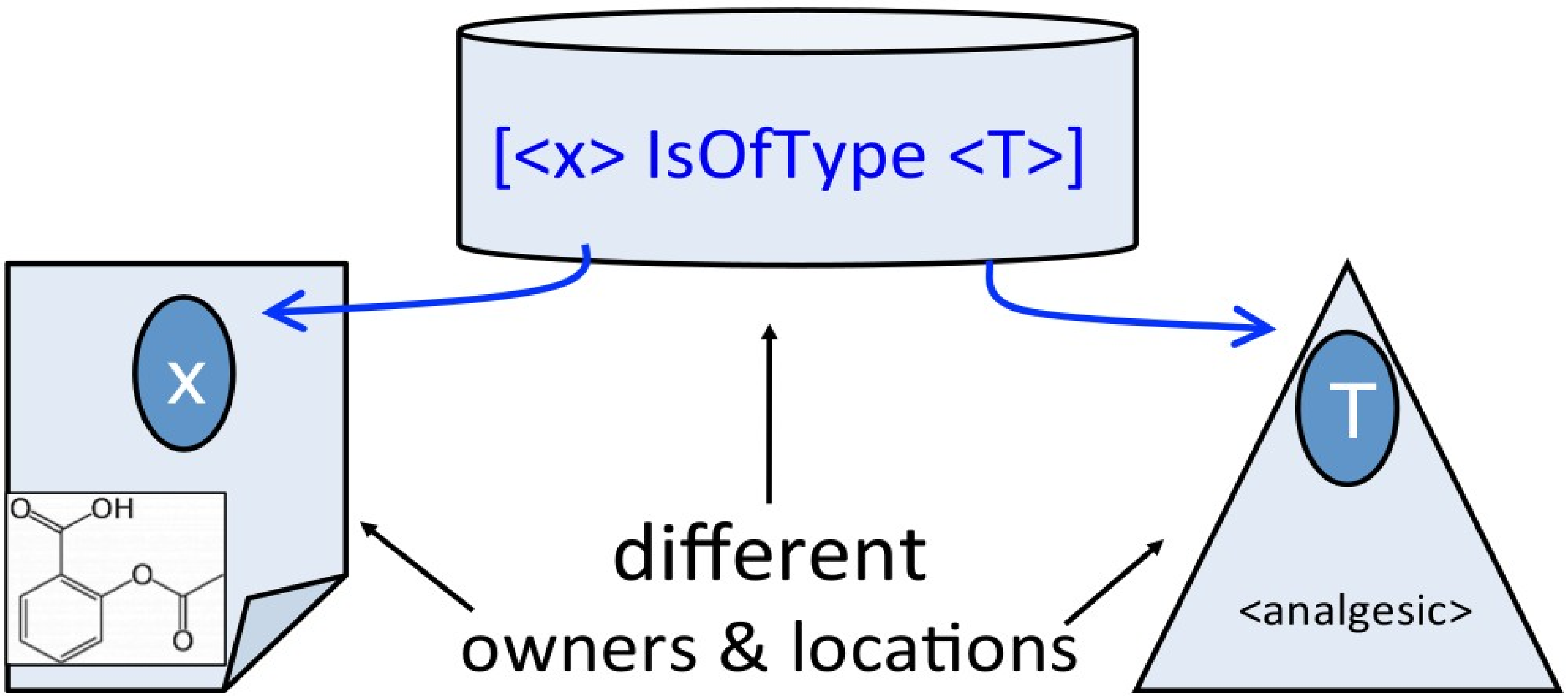
Four principles towards a Semantic Web of Data

- P2: relationships form a graph between things (a *knowledge graph*)

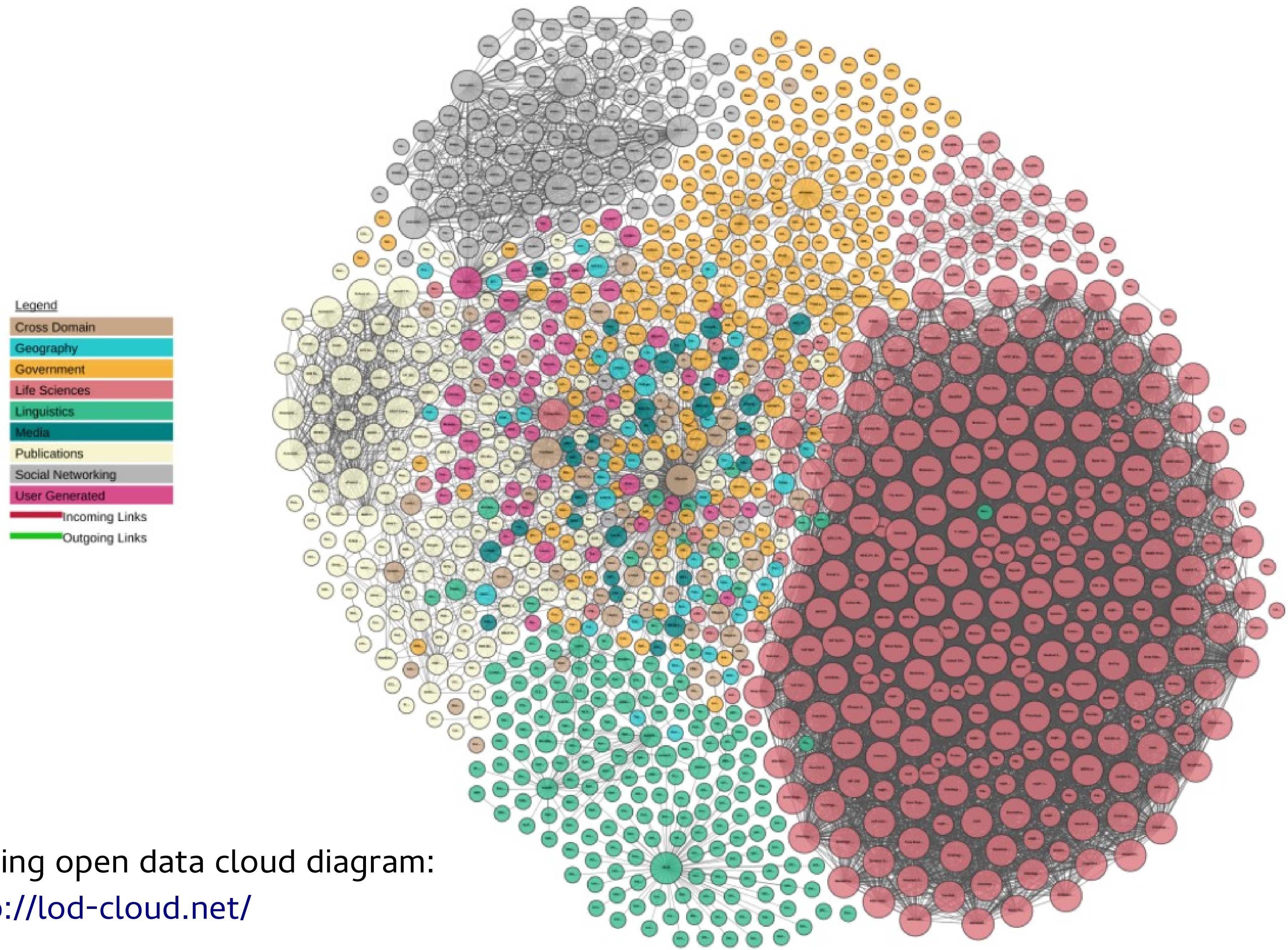


Four principles towards a Semantic Web of Data

- P3: names are addresses on the Web



• P1 + P2 + P3: a *huge* global graph



Linking open data cloud diagram:
http://lod-cloud.net/

Four principles towards a Semantic Web of Data

- P4: give explicit, formal semantics
  - assign types to things
  - assign types to relations
  - organize types in hierarchies
  - specify constraints

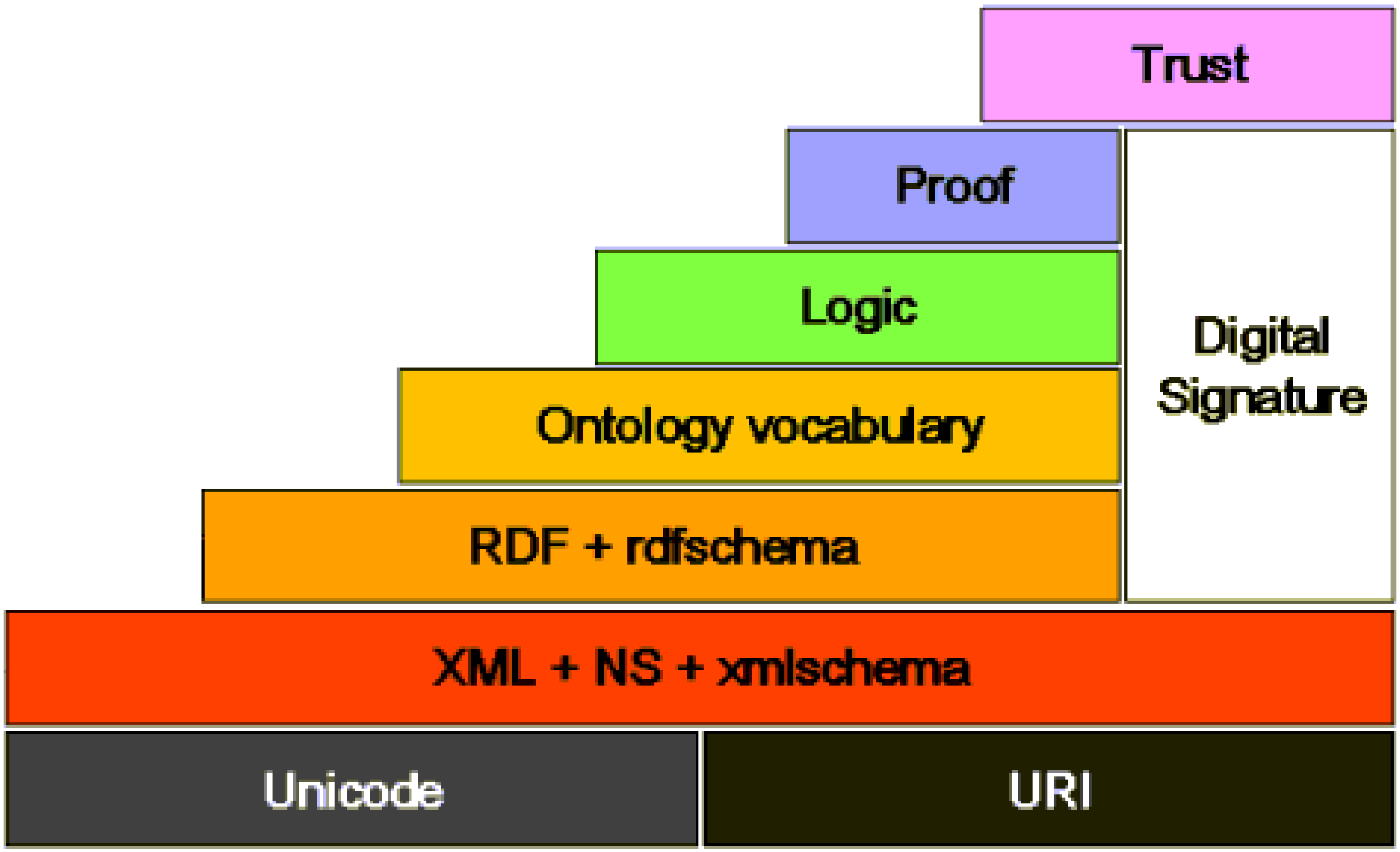
Semantic Web

- The Semantic Web is an evolving extension of the World Wide Web, promoting a *distributed* knowledge representation.

- It provides standards to
  - identify entities (URIs)
  - express facts (RDF)
  - express concepts (RDFS)
  - share vocabularies
  - reason on facts (OWL)

- These standards are produced and endorsed by the Word Wide Web Consortium (W3C)

The Semantic Web Tower



URI and namespaces

- URI = ***uniform resources identifier***
  - *uniform resource names* (URNs): URIs used to name something, even if this is an abstract object that is not available on the Web.
    - for instance, a person might have a URI that is used in ontologies to refer to that person.
  - *uniform resource locators* (URLs): URIs used to specify the location of something. they start with a protocol identifier, with a well-established technical interpretation (e.g. "http").

URI and namespaces

- Namespaces
  - Derived from domain registration (e.g. **epita.fr**)
  - Everything up to # may be *namespace*

- Examples:
  urn:myappname:students#student1234
  http://myserver.com/myapp/students/student1234

URI and namespaces

- Namespaces
  - Derived from domain registration (e.g. **epita.fr**)
  - Everything up to # may be *namespace*

- Examples:
  urn:myappname:students#student1234
  http://myserver.com/myapp/students/student1234

- URI are **dereferenciable** if the resource identified by the URI is retrievable from that URI

RDF (*resource description framework*)

- RDF is a ***data model***:
  - application and domain independent
  - based on simple triple format
  - (labeled and directed) graph

RDF (*resource description framework*)

- RDF is a ***data model***:
  - application and domain independent
  - based on simple triple format
  - (labeled and directed) graph

- RDF "statements" consist of
  - ***resources*** (= nodes)
    - which have ***properties***
      - which have ***values*** (= nodes, strings, ...)

RDF (*resource description framework*)

- RDF is a ***data model***:
  - application and domain independent
  - based on simple triple format
  - (labeled and directed) graph

subject

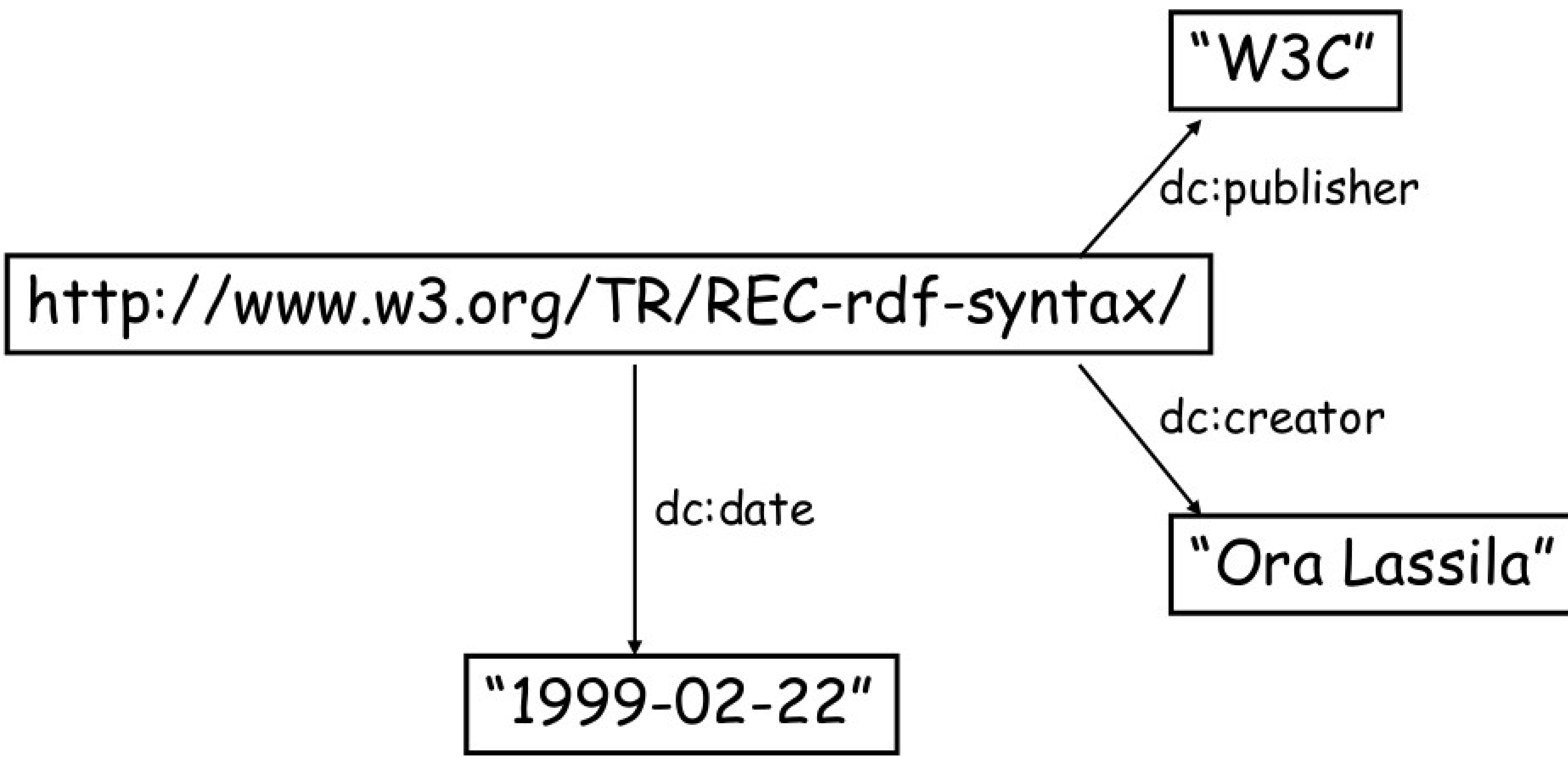
predicate

object

- RDF "statements" consist of
  - ***resources*** (= nodes)
    - which have ***properties***
      - which have ***values*** (= nodes, strings)

predicate(subject, object)

RDF (*resource description framework*)



RDF and XML

- Being so general, syntactic details are relatively insignificant; however XML is an example of commonly used syntactic structure.
  - NOTE: XML is just a way to write and transport RDF, but is not a component of RDF !

    RDF data can also be stored very differently, for example in a relational database.

RDF, a note on resources

- A graph node (corresponding to a resource) can be
  - the value of a property
  - arbitrarily complex tree and graph structures
- Syntactically, values can be
  - embedded (i.e. lexically in-line)
  - or referenced (linked)

RDF Schema

- **Base-level specification of semantics**

RDF Schema

- **Base-level specification of semantics**

- Language constructs include:
  - *class,*
  - *property,*
  - *subclass,*
  - *subproperty*

- Classes and properties are themselves also resources: this enables annotations about annotations
- Vocabulary can be used to define other vocabularies for your application domain

RDF(S) Terminology and Semantics

- Classes and class hierarchy
  - All classes are instances of **rdfs:Class**
  - A class hierarchy is defined by **rdfs:subClassOf**

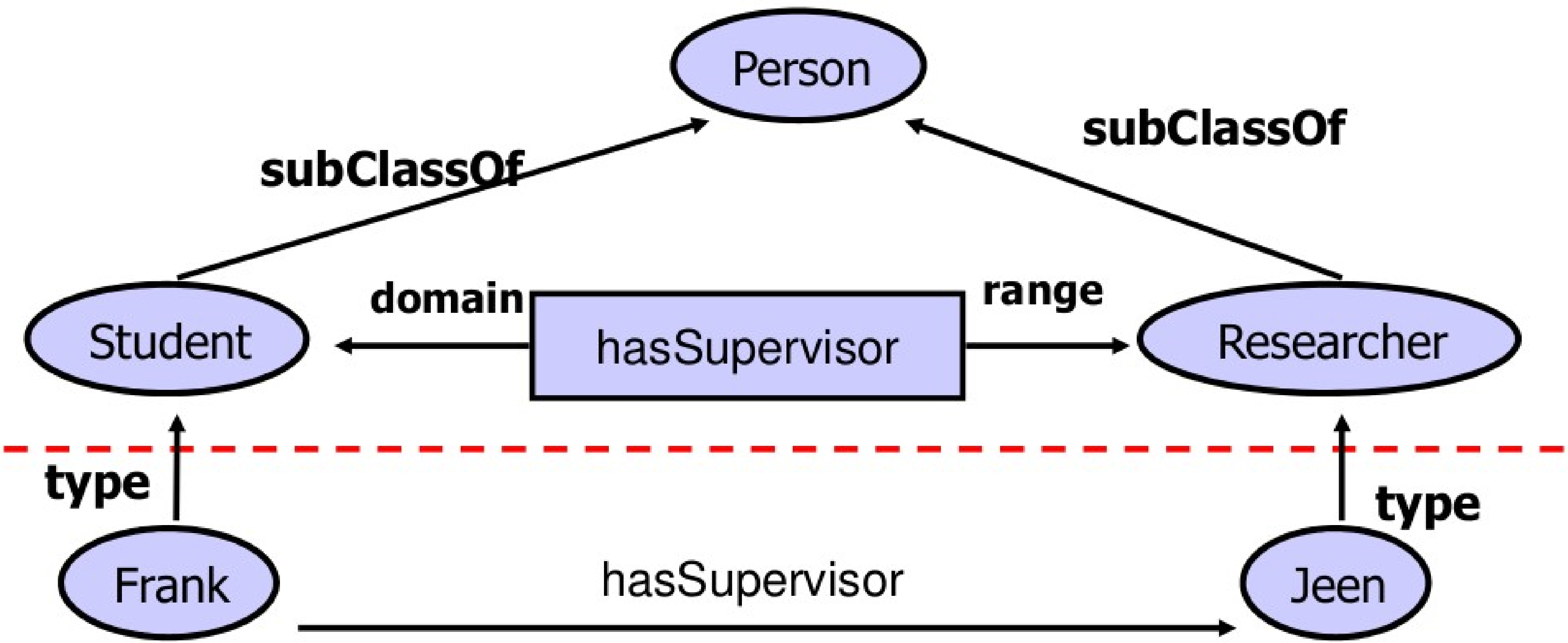
- Instances (individuals) of a class
  - defined by **rdf:type**

RDF(S) Terminology and Semantics

- Properties
  - properties are global: a property name in one place is the same as the property name in another (assuming the same namespace)
  - properties form a hierarchy, **rdfs:subPropertyOf**

- Domain and Range of a property
  - *domain*: the class (or classes) that have the property
  - *range:* the class (or classes) to which property values belong

RDF(S) Terminology and Semantics



RDF(S) Terminology and Semantics

Example:

```
(ex:MotorVehicle, rdf:type, rdfs:Class)
(ex:PassengerVehicle, rdf:type, rdfs:Class)
(ex:Van, rdf:type, rdfs:Class)
(ex:Truck, rdf:type, rdfs:Class)
(ex:MiniVan, rdf:type, rdfs:Class)
(ex:PassengerVehicle, rdfs:subClassOf, ex:MotorVehicle)
(ex:Van, rdfs:subClassOf, ex:MotorVehicle)
(ex:Truck, rdfs:subClassOf, ex:MotorVehicle)
(ex:MiniVan, rdfs:subClassOf, ex:Van)
(ex:MiniVan, rdfs:subClassOf, ex:PassengerVehicle)
```

Querying RDF: SPARQL

- Simple Protocol And RDF Query Language
  - W3C standardisation effort similar to the Xquery query language for XML data
  - suitable for remote use (remote access protocol)

```
PREFIX abc: <http://mynamespace.com/exampleOntology#>
SELECT ?capital ?country
WHERE { ?x abc:cityname ?capital.
        ?y abc:countryname ?country.
        ?x abc:isCapitalOf ?y.
        ?y abc:isInContinent abc:africa. }
```

OWL (*Web Ontology Language*)

- OWL adds expressivity to RDF Schema to enable more powerful semantics:
  - cardinality restrictions,
  - local range constraints,
  - equality of resources,
  - inverse, symmetric and transitive properties,
  - boolean class combinations,
  - disjointness and completeness, ...

OWL (*Web Ontology Language*)

- OWL adds expressivity to RDF Schema to enable more powerful semantics:
  - cardinality restrictions,
  - local range constraints,
  - equality of resources,
  - inverse, symmetric and transitive properties,
  - boolean class combinations,
  - disjointness and completeness, ...

- **OWL Lite**: subset of features that is easy to implement and use
- **OWL DL**: subset of features supporting description-logic reasoning (e.g. useful for ontology construction)

objects hierarchies compositions

Instances, Taxonomies, Mereonomies

Individuals (IS-INSTANCE-OF)

- The concept of class intuitively refers to some entity that belongs to that class.

- This entity or *object* is said to an **instance of** that class.

Individuals (IS-INSTANCE-OF)

```
class Person {
  String name

  void setName(String newName) {
    name = newName
  }
}

p = new Person()
p.setName("Plato")
```

Individuals (IS-INSTANCE-OF)

```
class Person {
  String name

  void setName(String newName) {
    name = newName
  }
}

p = new Person()
p.setName("Plato")
```

describe properties of the system

Individuals (IS-INSTANCE-OF)

```
class Person {
  String name

  void setName(String newName) {
    name = newName
  }
}

p = new Person()
p.setName("Plato")
```

actually allocate (memory) space for the object

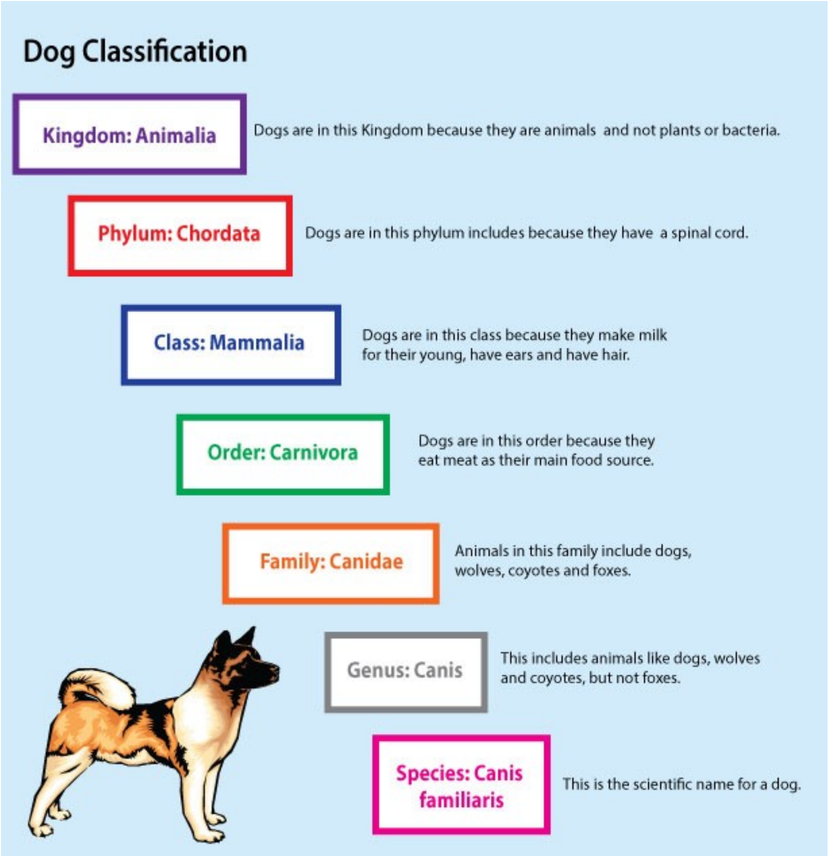
Individuals (IS-INSTANCE-OF)

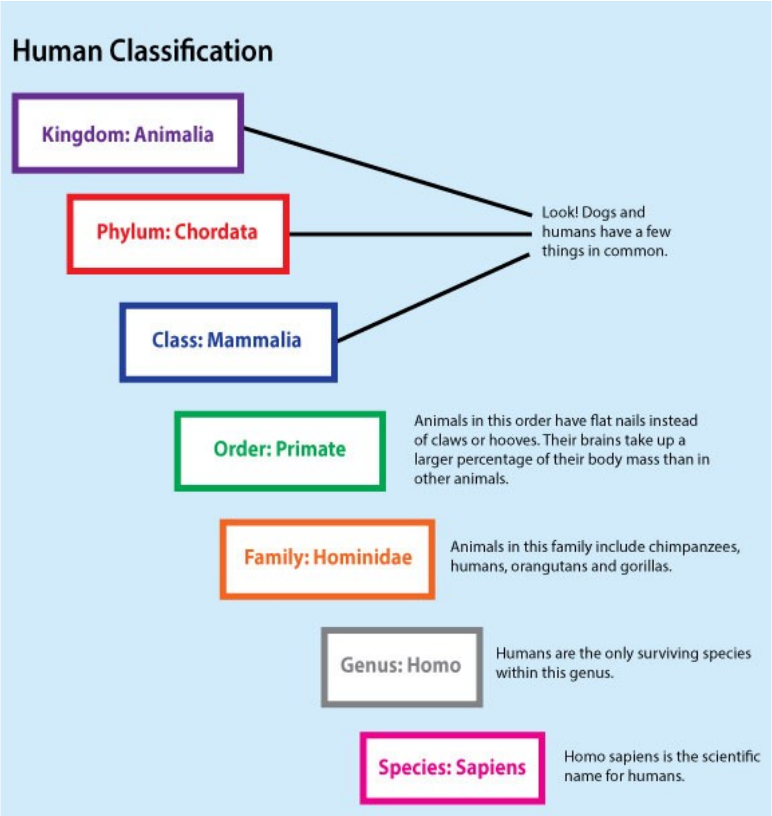
```
class Person {
  String name

  void setName(String newName) {
    name = newName
  }
}

p = new Person()
p.setName("Plato")
```

apply the required method to the object

Hierarchy as Taxonomy (IS-A)

- Things and concepts are usually hierarchically classified both in common and expert knowledge.





Hierarchy as Taxonomy (IS-A)

- Things and concepts are usually hierarchically classified both in common and expert knowledge.
- Given a certan class, a subclass or **derived class** inherits certain properties (as attributes and methods) from the first.
  - From the perspective of the second class, the first is called *superclass*.
  - Usually, derivation might be ***overridden***.

Hierarchy as Taxonomy (IS-A)

```
class A {
  String salutation = "Ciao"
  void show() {
    print(salutation + "! My type is A.")
  }
}

class B extends A {
  @Override
  void show() {
    print(salutation + "! My type is B.")
  }
}

o = new A()
o.show()
o = new B()
o.show()
```

output

Ciao! My type is A.

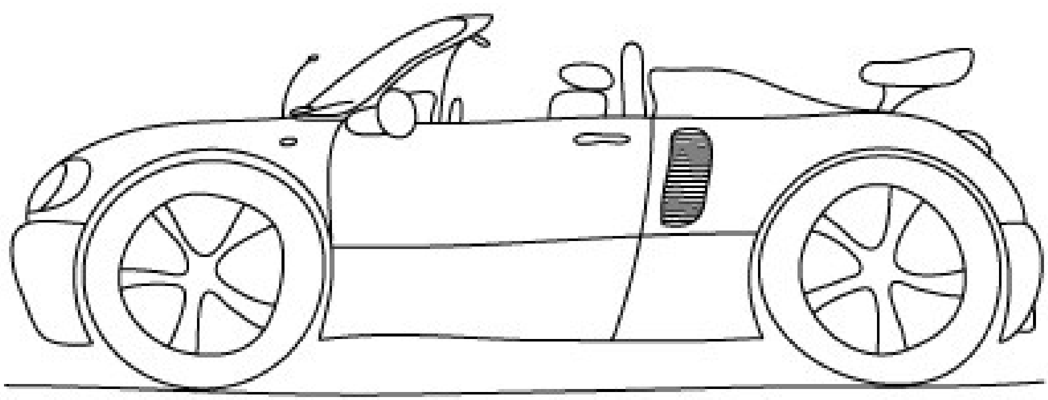
Ciao! My type is B.

Hierarchy as partonomy (HAS-A)

- Given an object of a certain class, if it is composed by other objects, the second ones *belong to* the first.

Hierarchy as partonomy (HAS-A)

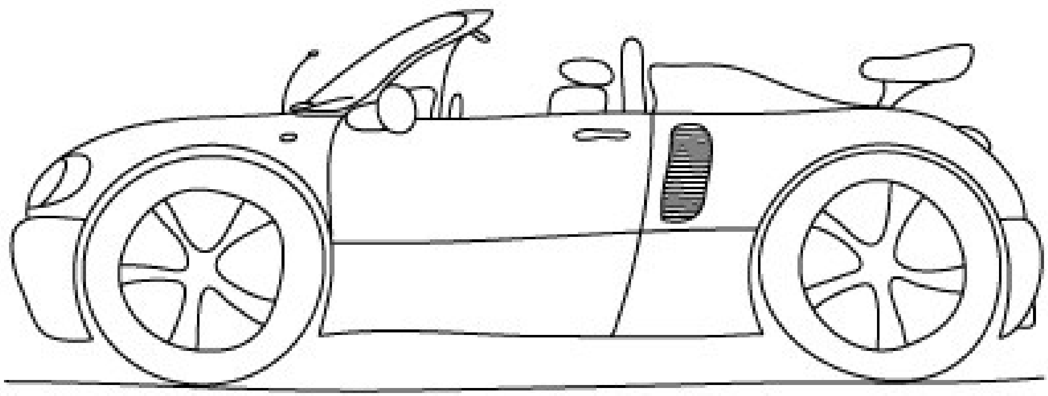
- Given an object of a certain class, if it is composed by other objects, the second ones *belong to* the first.



  - The car *has* four wheels.
  - Those wheels *belongs to* the car.

Hierarchy as partonomy (HAS-A)

- Given an object of a certain class, if it is composed by other objects, the second ones *belong to* the first.



  - The car *has* four wheels.
  - Those wheels *belongs to* the car.

...but things are a bit more complicated!

“Strict” composition

```
class Car {
  Wheel frontLeftWheel
  Wheel frontRightWheel
  Wheel rearLeftWheel
  Wheel rearRightWheel

  Car {
    frontLeftWheel = new Wheel()
    frontRightWheel = new Wheel()
    rearLeftWheel = new Wheel()
    rearRightWheel = new Wheel()
  }
}

car = new Car()
```

The lifetime of the components **depends on** the composed object.

Aggregation or weak composition

```
class Car {
  Wheel frontLeftWheel
  Wheel frontRightWheel
  Wheel rearLeftWheel
  Wheel rearRightWheel

  Car { }

  void mountWheels(fLW, fRW, rLW, rRW) {
    frontLeftWheel = fLW
    frontRightWheel = fRW
    rearLeftWheel = rLW
    rearLeftWheel = rRW
  }
}

car = new Car()
car.mountWheels(...)
```

The lifetime of the components **can differ** of that of the composed object.

Example of ontology in description logic

$\mathtt{Woman} \doteq \mathtt{Person} \sqcap \mathtt{Female}$
$\mathtt{Man} \doteq \mathtt{Person} \sqcap \neg\mathtt{Female}$
$\mathtt{Mother} \doteq \mathtt{Woman} \sqcap \exists\mathtt{hasChild.Person}$
$\mathtt{Father} \doteq \mathtt{Man} \sqcap \exists\mathtt{hasChild.Person}$
$\mathtt{Parent} \doteq \mathtt{Person} \sqcap \exists\mathtt{hasChild.Person}$
$\mathtt{Grandmother} \doteq \mathtt{Mother} \sqcap \exists\mathtt{hasChild.Parent}$
$\mathtt{DaughterlessMother} \doteq \mathtt{Mother} \sqcap \forall\mathtt{hasChild}.\neg\mathtt{Female}$

TBOX

Terminological box

Example of ontology in description logic

$\text{Woman} \doteq \text{Person} \sqcap \text{Female}$
$\text{Man} \doteq \text{Person} \sqcap \neg\text{Female}$
$\text{Mother} \doteq \text{Woman} \sqcap \exists\text{hasChild.Person}$
$\text{Father} \doteq \text{Man} \sqcap \exists\text{hasChild.Person}$
$\text{Parent} \doteq \text{Person} \sqcap \exists\text{hasChild.Person}$
$\text{Grandmother} \doteq \text{Mother} \sqcap \exists\text{hasChild.Parent}$
$\text{DaughterlessMother} \doteq \text{Mother} \sqcap \forall\text{hasChild.}\neg\text{Female}$

TBOX
Terminological box

DaughterlessMother(Paulette)
Child(Paulette, Pierre)
Child(Paulette, Jacques)
Father(Pierre)
Child(Pierre, Marinette)

ABOX
Assertion box