Integrating, querying and reasoning on data and knowledge with Semantic Web technologies

Olivier Dameron
https://orcid.org/0000-0001-8959-7189

Université de Rennes

July 12, 2023

https://gitlab.com/odameron/eins2023

Université de Rennes

UMR IRISA

DyLiSS

Before we begin

- who as an experience of SQL?
- who as an experience of RDF and/or SPARQL?

Outline

- **Life science data:** quantity, complexity and requirements
- **Metadata:** explicit representation of data interpretation
- **Ontologies:** explicit representation of domain knowledge underlying metadata
- **Semantic Web:** crash course on a unified technical framework for integrating all the above, + querying and reasoning about them

By the end of the session you will

- understand the motivations for the Semantic Web
- know how to represent (meta)data in RDF and expose them
- write SPARQL queries for retrieving information from RDF graphs
- write semantically-rich SPARQL queries that perform reasoning based on domain knowledge from ontologies

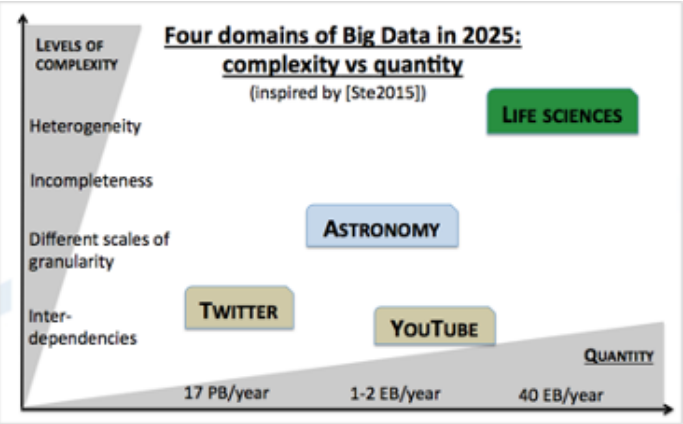
Data, Knowledge and Reasoning in Life Science

Life science from a data science perspective

"Biology has become an information science" [Lenoir1999]

- so have most experimental sciences
- but **life sciences are particular**

[Stephens2015]



Specificities

- data deluge $\rightarrow$ big data [Aldhous1993][Stephens2015]
- data complexity [Stephens2015]
- reasoning complexity
- domain knowledge [Antezana2009]

- Stats and ML succesfully highlight the top-ranking entities in a dataset
- but these are often the consequences

Needs

- **identify the cause**
- **explain the underlying process**

Complexity of life science data and reasoning

Challenge (computational)

How to handle this complexity?

- Experts are very good at doing it on their domain (hint)
  - on their domain
  - on their data
- The difficulty is to do it systematically
- Expertise = ability to **use knowledge for interpreting data**

Remember slide 6: expertise = use knowledge for interpreting data

"Metadata, you see, is really a love note - it might be to yourself, but in fact it's a love note to the person after you, or the machine after you, where you've saved someone that amount of time to find something by telling them what this thing is."
Jason Scott - http://ascii.textfiles.com/archives/3181

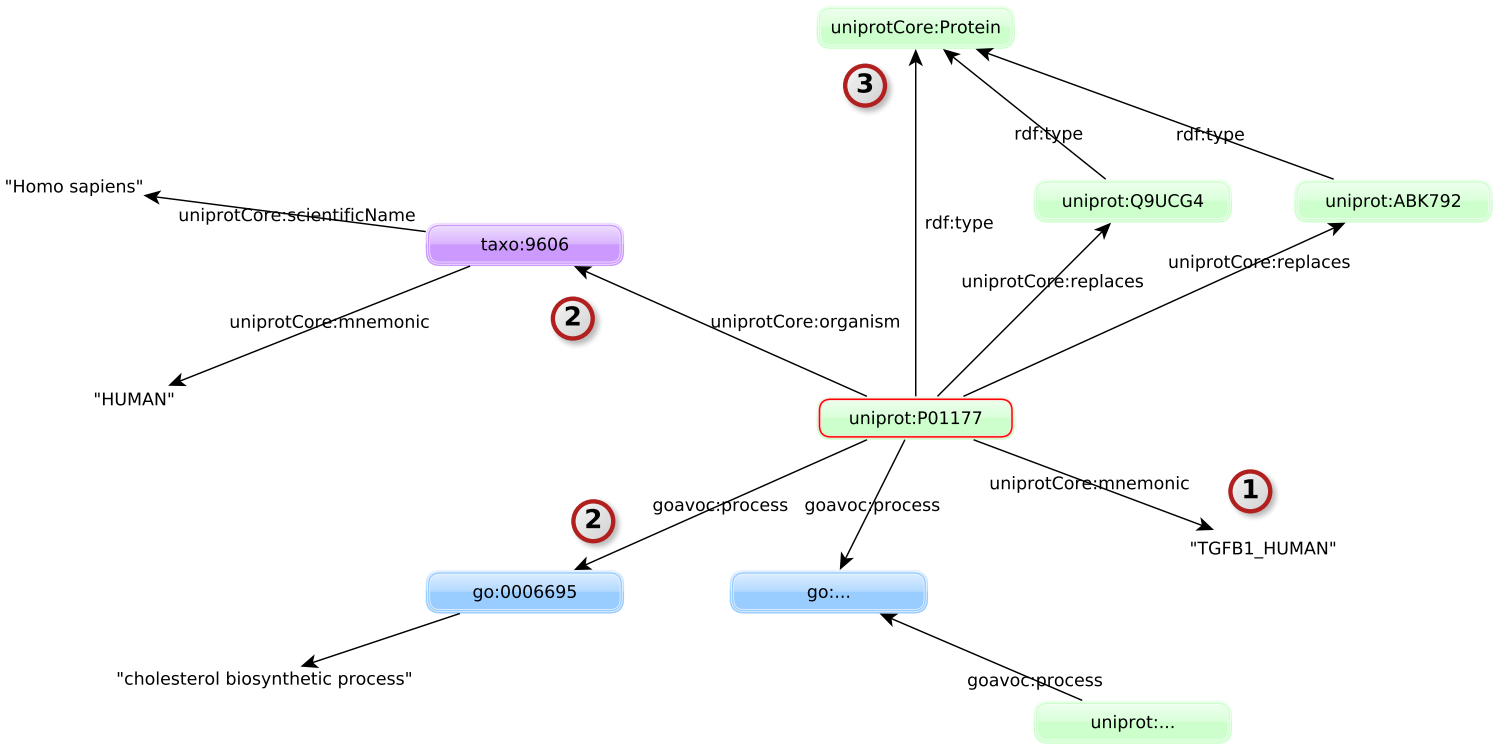
Annotate data = result of some **interpretation process**

- describe **explicitely** the relevant **elements** in your data and the **relations** between these elements
- so that users (incl. you and the non-experts) or programs do not have to go once again through the tedious process of interpreting them
- metadata = capture the result of interpretation

Annotations (aka metadata) describe entities

Annotations are **explicit** descriptions of (**anecdotic**) entities

1. their intrinsic characteristics
2. their relations to other entities
3. the categories they belong to



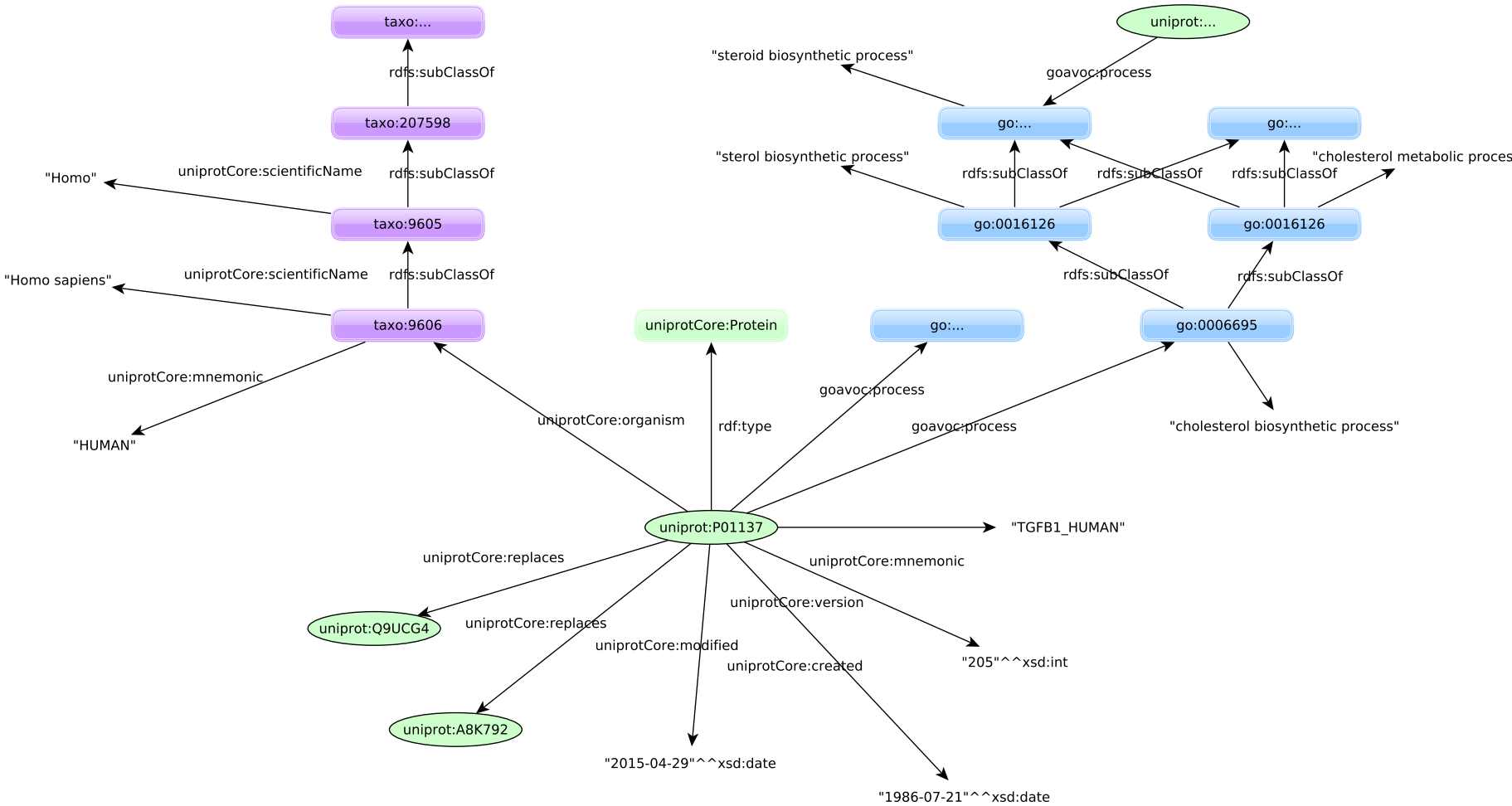
"Ontologies make people happier"
Ph. Rocca-Serra, 2011-10-18

Knowledge underlying annotations remains to be represented

- "Much of biology works by applying prior knowledge [...] to an unknown entity" [Stevens2000]
- "The complex biological data stored in bioinformatics databases often require the addition of knowledge to specify and constrain the values held in that database" [Stevens2000]

Ontologies specify/formalize the meaning of annotations

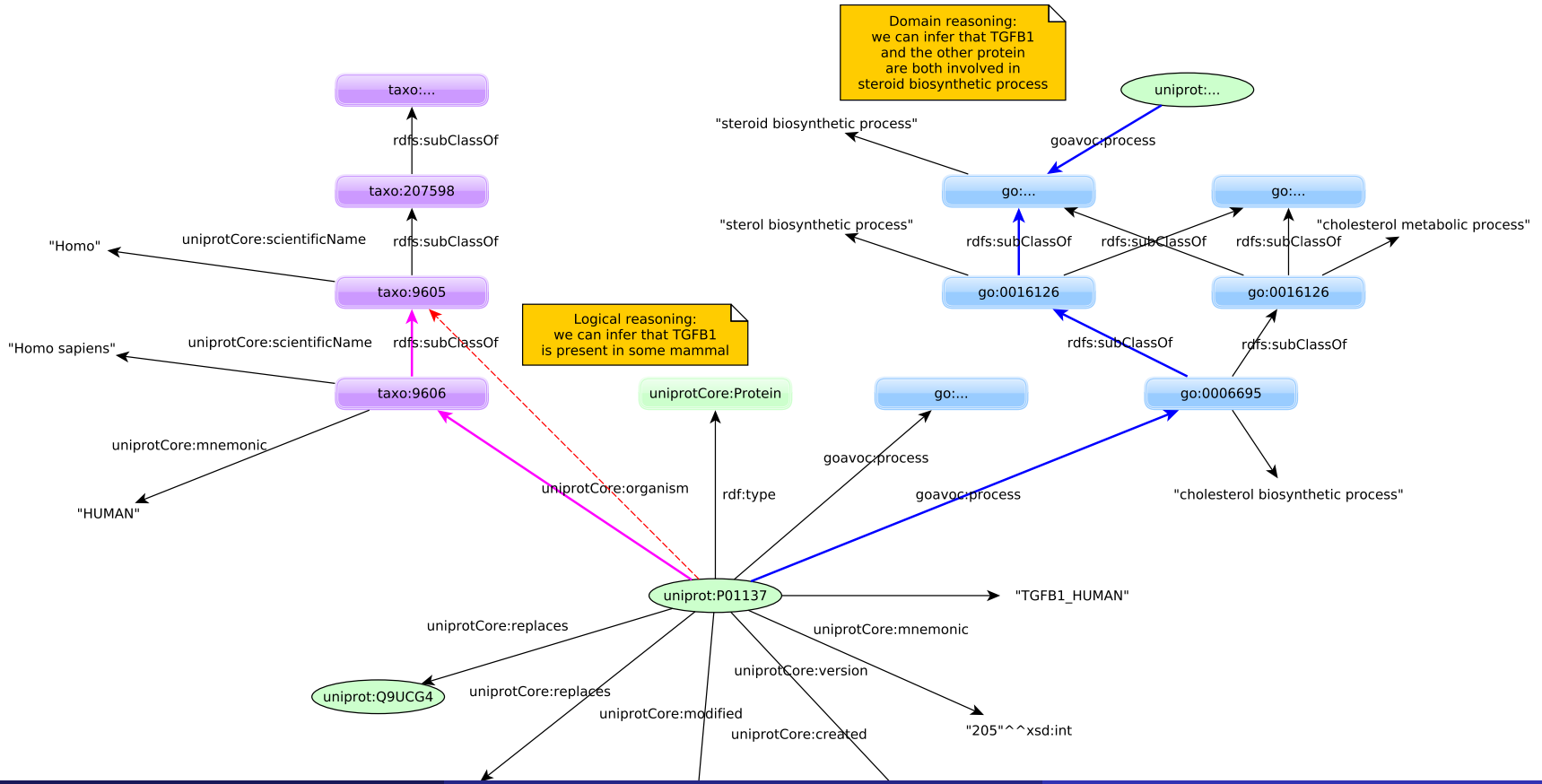
Knowledge is **formalized** as (**generic**) relations between **sets** of entities



Ontologies support reasoning about annotations

Reasoning

Method for **traversing or enriching** the graph of data.
The **rules** may be logical and/or domain-based



Requirements for coping with life science data complexity

- **Requirement 1:** **identify** resources with interoperable identifiers
- **Requirement 2:** **describe** resources
  - their characteristics (e.g. start and end position of a gene,...)
  - their relations to other entities (e.g. the transcripts associated to a gene, the transcription factors that regulate it,...)
  - the categories they belong to
- **Requirement 3:** **combine** descriptions from different origins
- **Requirement 4:** **query** these descriptions
- **Requirement 5:** support **semantically-rich** querying and reasoning (because of the inner complexity) using domain knowledge (this is required for capturing *expertise*)

If only the solutions to all these requirements were compatible!

The Semantic Web: a framework for integrating seamlessly (data,) metadata and knowledge, and querying and reasoning over it

There has to be a better way

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

Semantic Web and Linked (Open) Data

Semantic Web offers a unified framework to Linked Data

- **RDF** for representing and aggregating entities descriptions
- **RDFS+OWL** for representing domain knowledge (and combine it with data descriptions)
- **SPARQL** for querying everything (possibly from multiple repositories)

SPARQL endpoints offer unified query access to RDF repositories
ex: Fuseki, Virtuoso,...

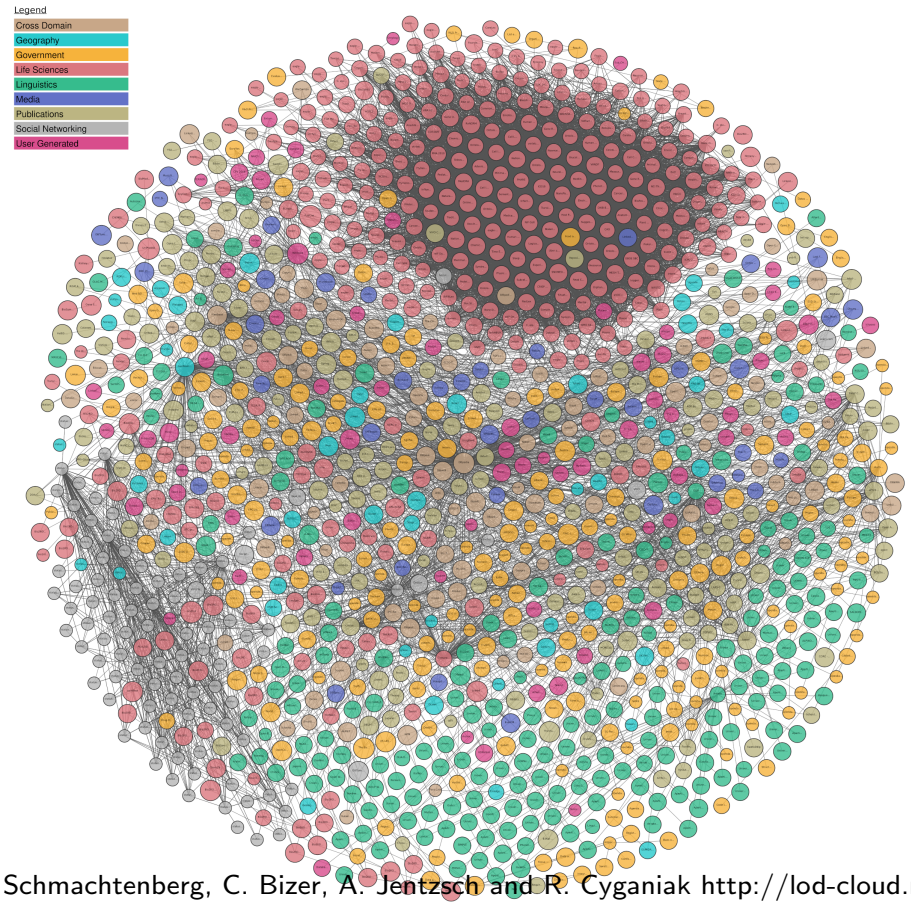
Linked Open Data: a federation of RDF repositories

LODStats (http://lodstats.aksw.org/) [Ermilov2016]

- 9960 datasets; 149.10^9 triples
- general scope; Life sciences = major field (size+density)

Linked open data (in 2021-05-05)

- RDF repositories can be queried in SPARQL via endpoints
- data from one endpoint can make references to data from another endpoint



Linked open data cloud, by M. Schmachtenberg, C. Bizer, A. Jentzsch and R. Cyganiak http://lod-cloud.net/

Semantic Web: RDF for annotating data

RDF is not the problem, it is the solution

People think RDF is a pain because it is complicated. The truth is even worse. RDF is painfully simplistic, but it allows you to work with real-world data and problems that are horribly complicated. While you can avoid RDF, it is harder to avoid complicated data and complicated computer problems. RDF brings together data across application boundaries and imposes no discipline on mandatory or expected structures. This can make working with RDF data frustrating. Its schema and ontology languages can help define the meaning of RDF content but, again, can't express rules about how actual data records should look. The contents of this book are nearly 20 years too late, but better now than never. Recent developments around RDF validation have finally made it easier to record, exchange, and understand rules about validating and otherwise checking RDF data. Who knows what wonders await us in another 20 years.

Dan Brickley, Schema.org and Google
Libby Miller, BBC
July 2017

RDF: Resource Description Framework

RDF is for describing things

- **Things**: called 'resources' (the R in RDF)
  - real things (people, objects, places,...)
  - imaginary things (unicorns,...)
  - ideas, concepts (liberty, algebra, triangle, colors,...)
  - temporal things (instants, periods,...)
  - collections of things (sets, lists,...)
  - ...
- **Describing** (the D in RDF) a thing = representing explicitely
  - its characteristics
  - its relations to other things
  - its categories
- **Annotation** = explicit representation of the descriptions
  - it's not just in your head anymore
  - computers can process them...
  - ... and combine them with other annotations

RDF W3C recommendation

RDF 1.1 is a W3C recommendation (2014-02-25)

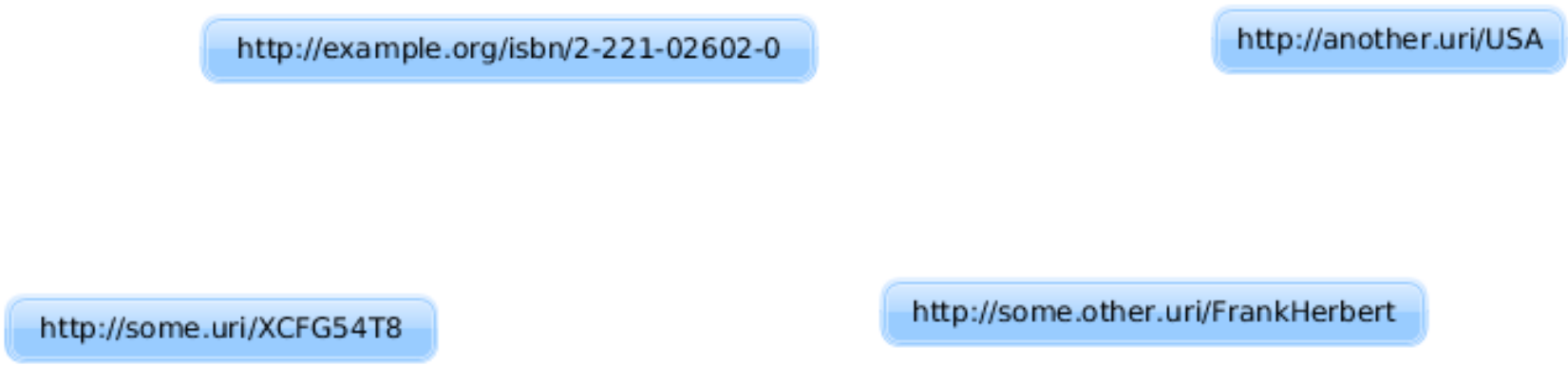
- Concepts and abstract syntax
  https://www.w3.org/TR/rdf11-concepts/
- Semantics
  https://www.w3.org/TR/rdf11-mt/

(RDF 1.0 = recommendation in 1999 + specif. in 2004)
(RDF 1.2 currently in active discussion)

Resources: the things we talk about

Resources: the things we talk about

- Resources are identified by their URI...
- ...but it is cumbersome to read
- You are free to create URIs as you like (don't have to be URL)



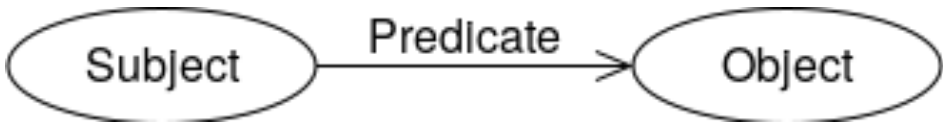
From now on, we hide the URIs but they are still here (box ⇒ URI)

Cool URIs don't change (https://www.w3.org/Provider/Style/URI)

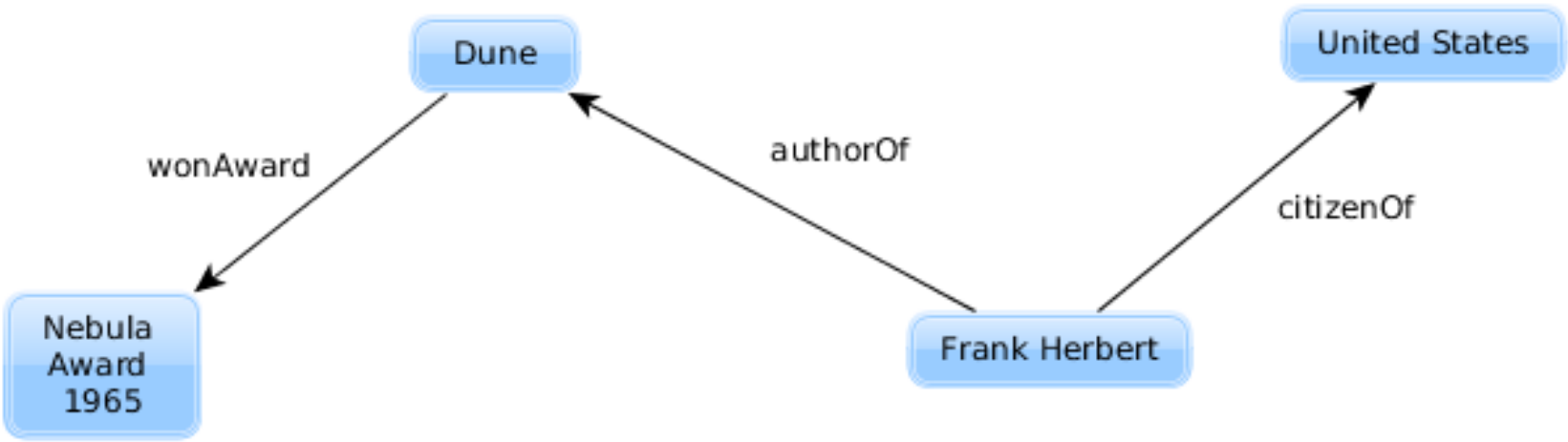
RDF triple

RDF triple = <subject, predicate, object>

- **subject**: the resource being described
- **predicate**: the relation (from the subject to the object)
- **object**: the value of the predicate for the subject
  (one of them if the predicate can have several values for the subject; in this case use as many triples as necessary)

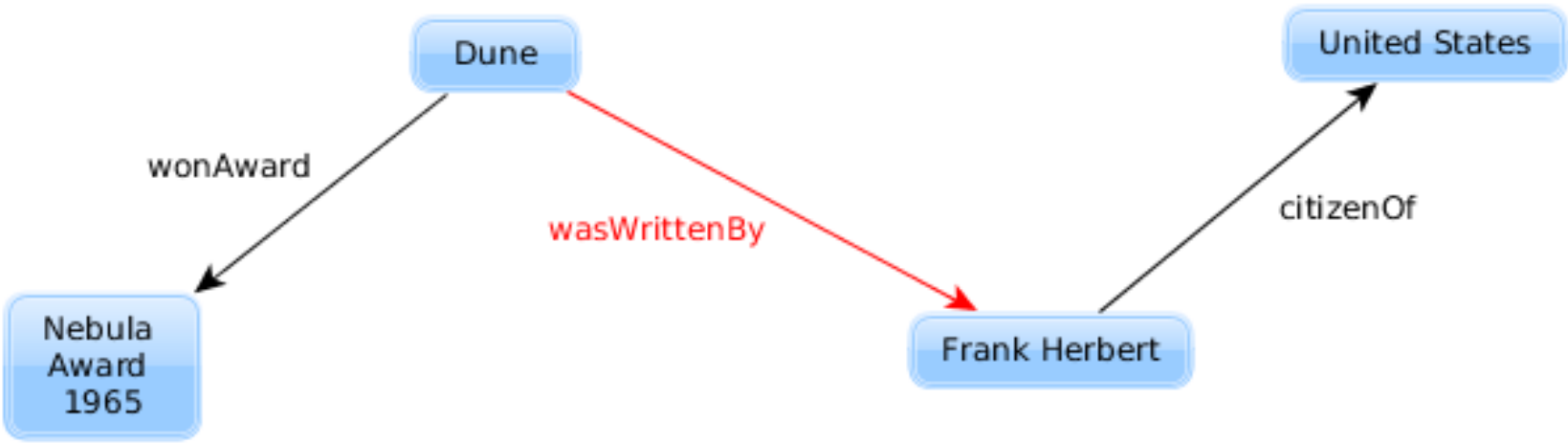


Properties (1/2): relations between resources



- Properties are (also) identified by their URI, and it is still cumbersome to read (so we also hide them)
- You are still free to create URIs as you like (avoid `rdf:type`)

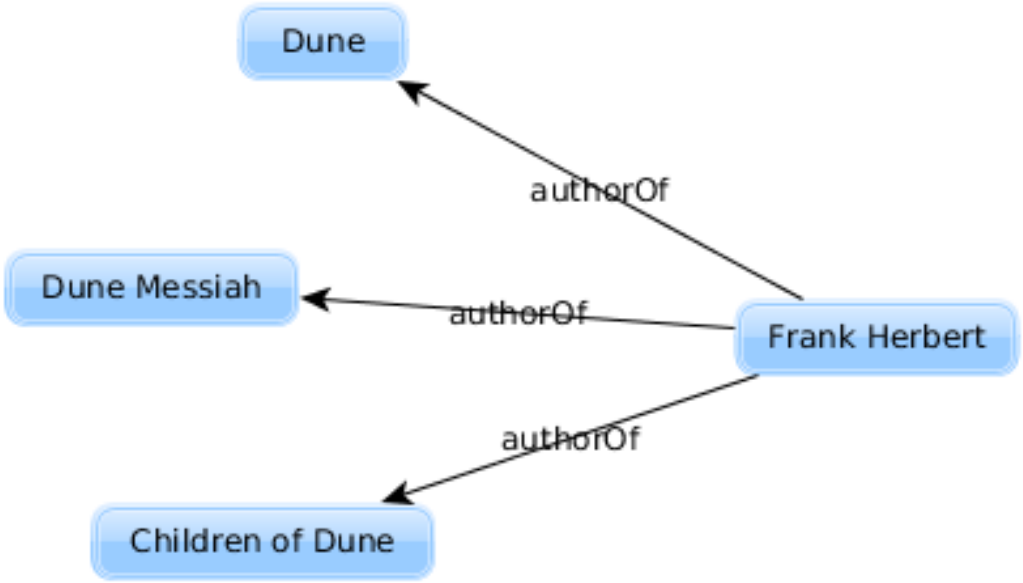
Properties (1/2): relations between resources



is also possible

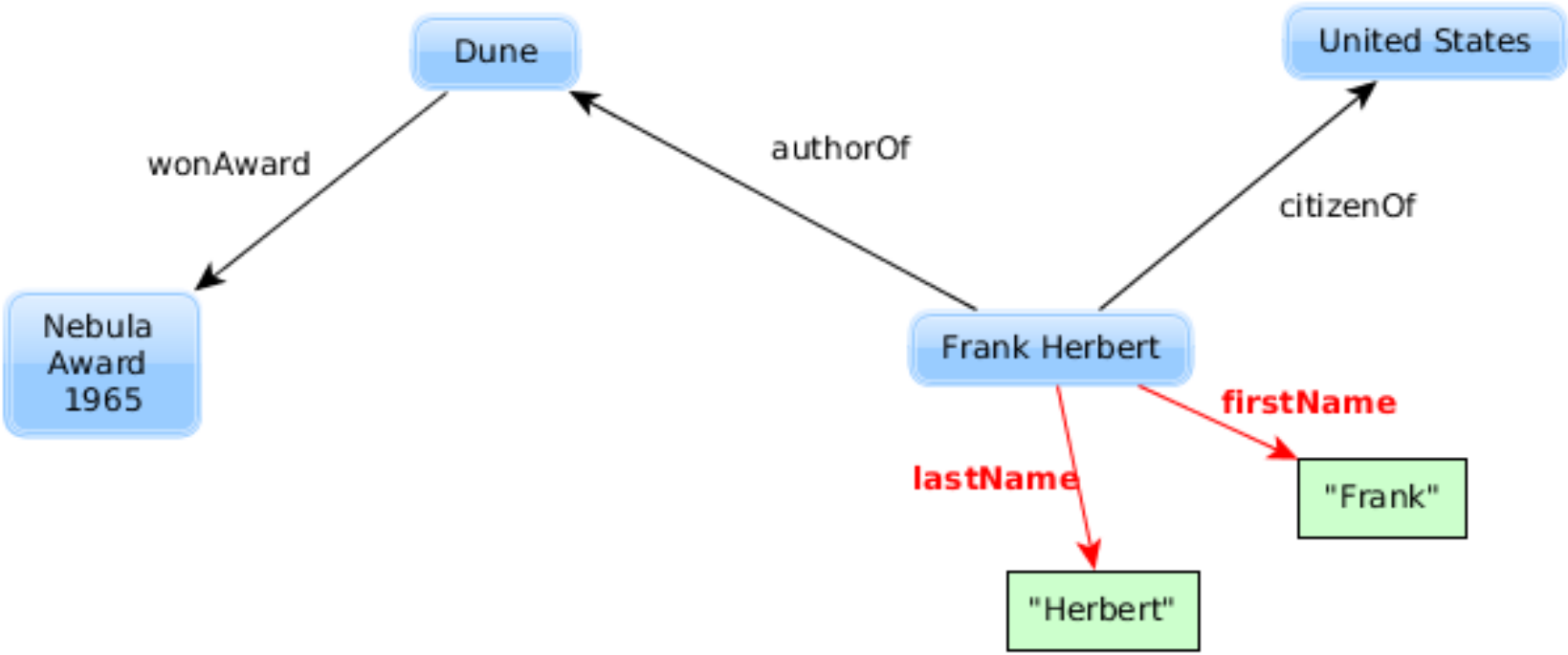
- `authorOf` and `wasWrittenBy` would have a different URIs
- extensions of RDF (such as OWL) even
  - allow to state explicitely that they are the inverse of each-other
  - support the subsequent reasoning

Properties (1/2): relations between resources



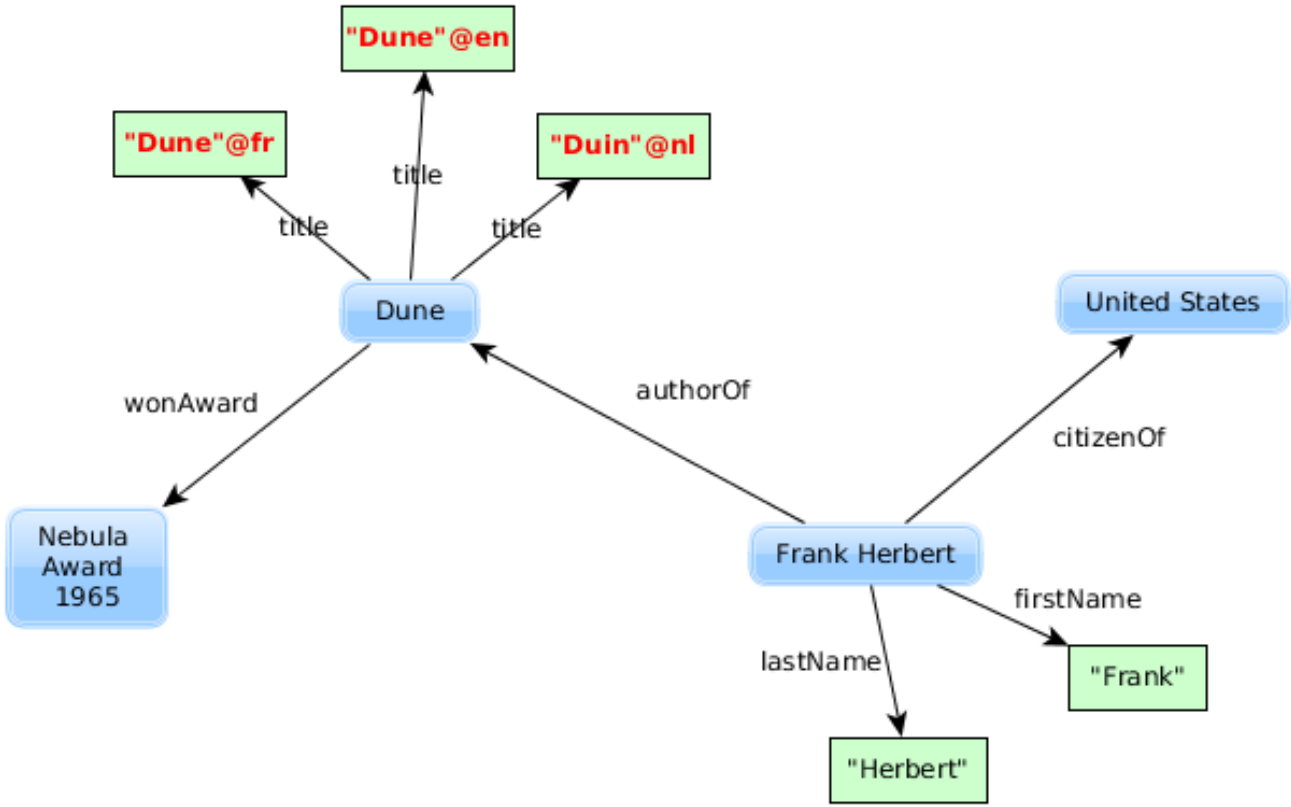
A property can have multiple values

Properties (2/2): relations between a resource and a datatype value



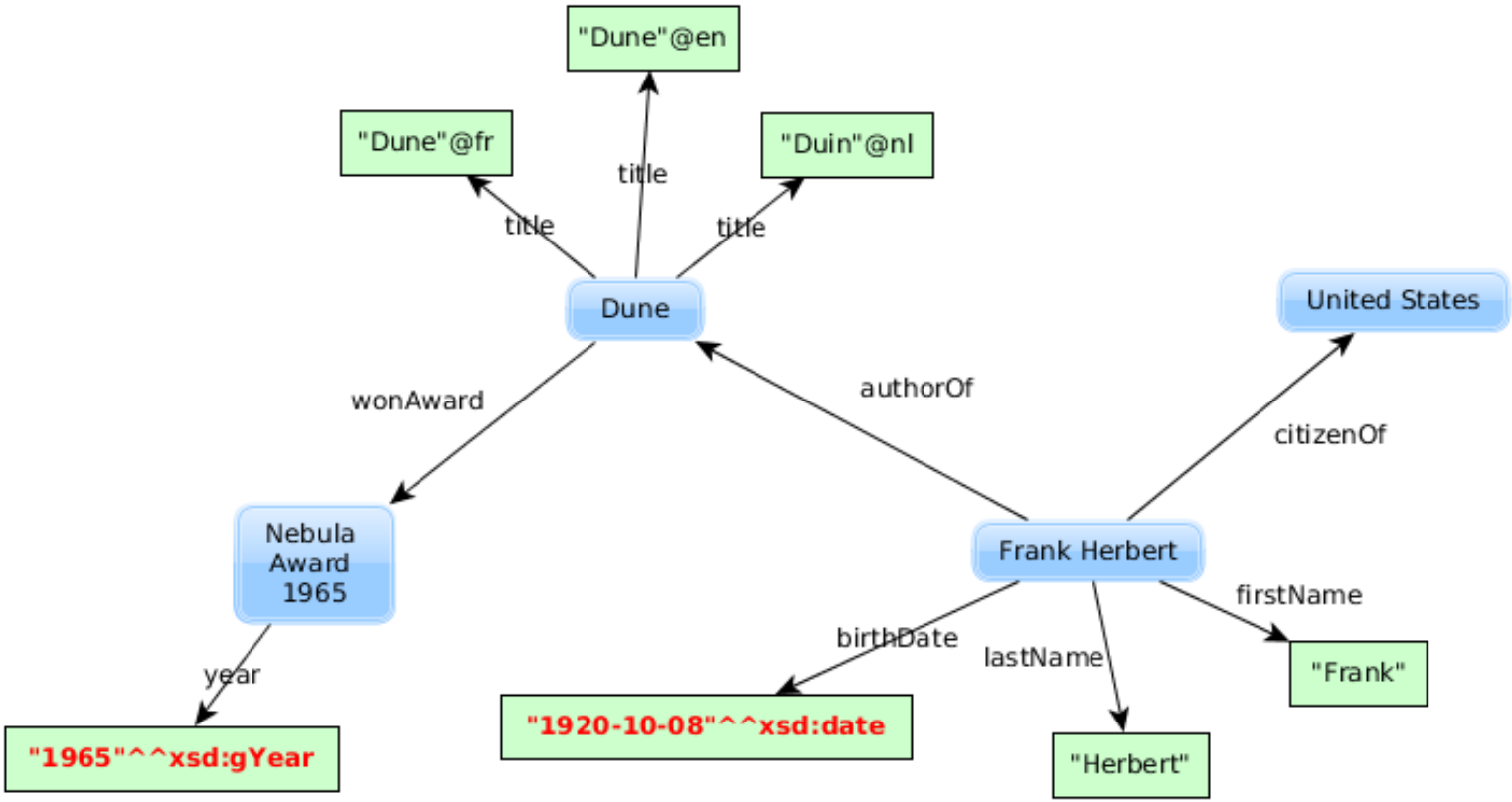
- A property can point to a string, but **never** originate from one
- A string is **not** a resource (it does not have an URI)

Properties (2/2): relations between a resource and a datatype value



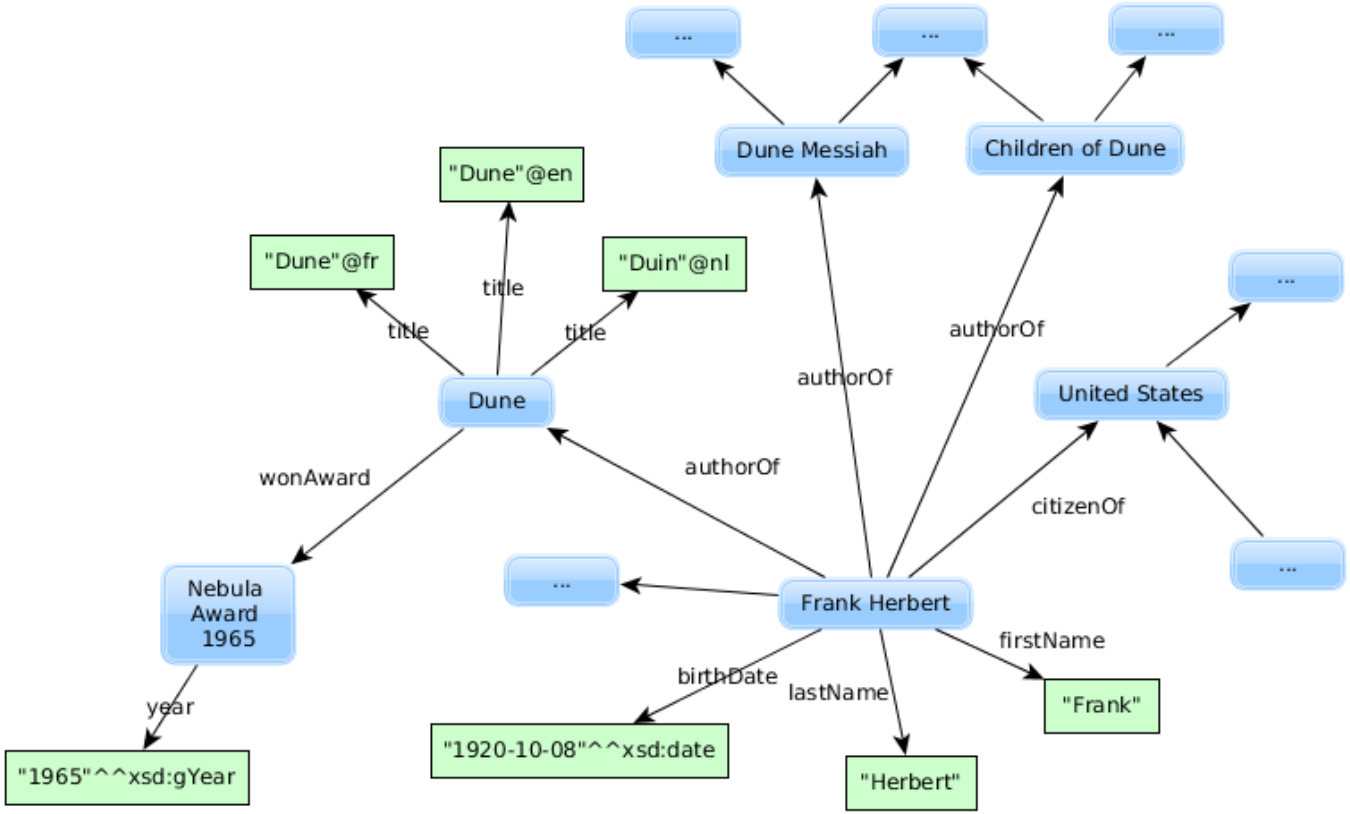
An optional language code can be appended to a string
(because we cannot have a `hasLanguage` property starting from it)

Properties (2/2): relations between a resource and a datatype value

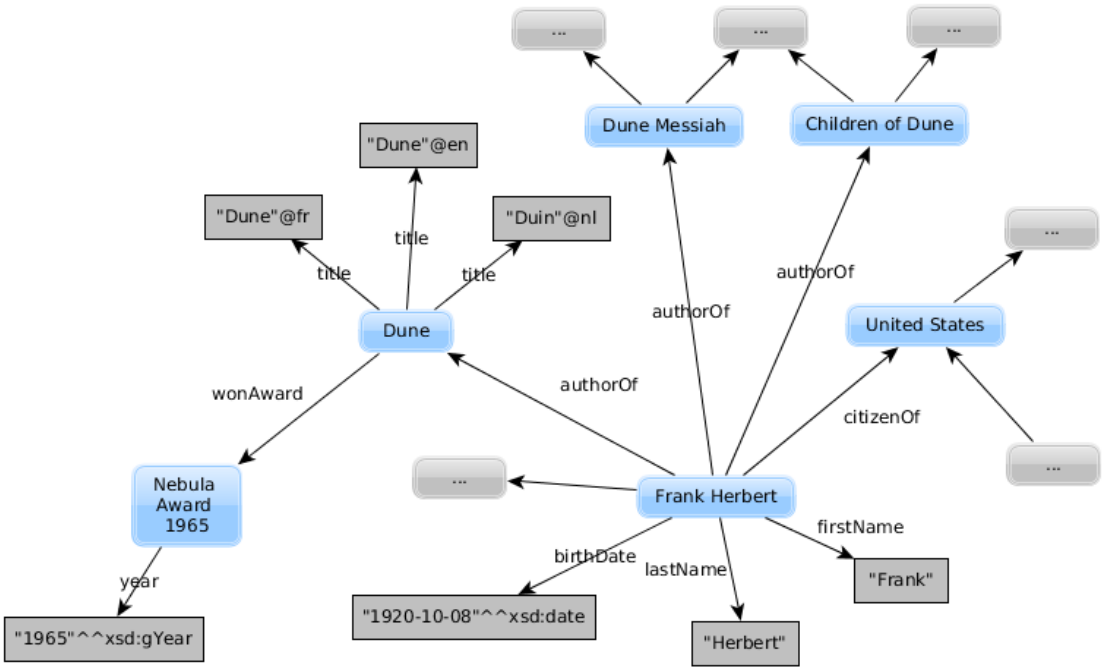


An optional XSD type can be appended to a string
(because we cannot have a `hasType` property starting from it)

RDF triples: a directed labeled graph



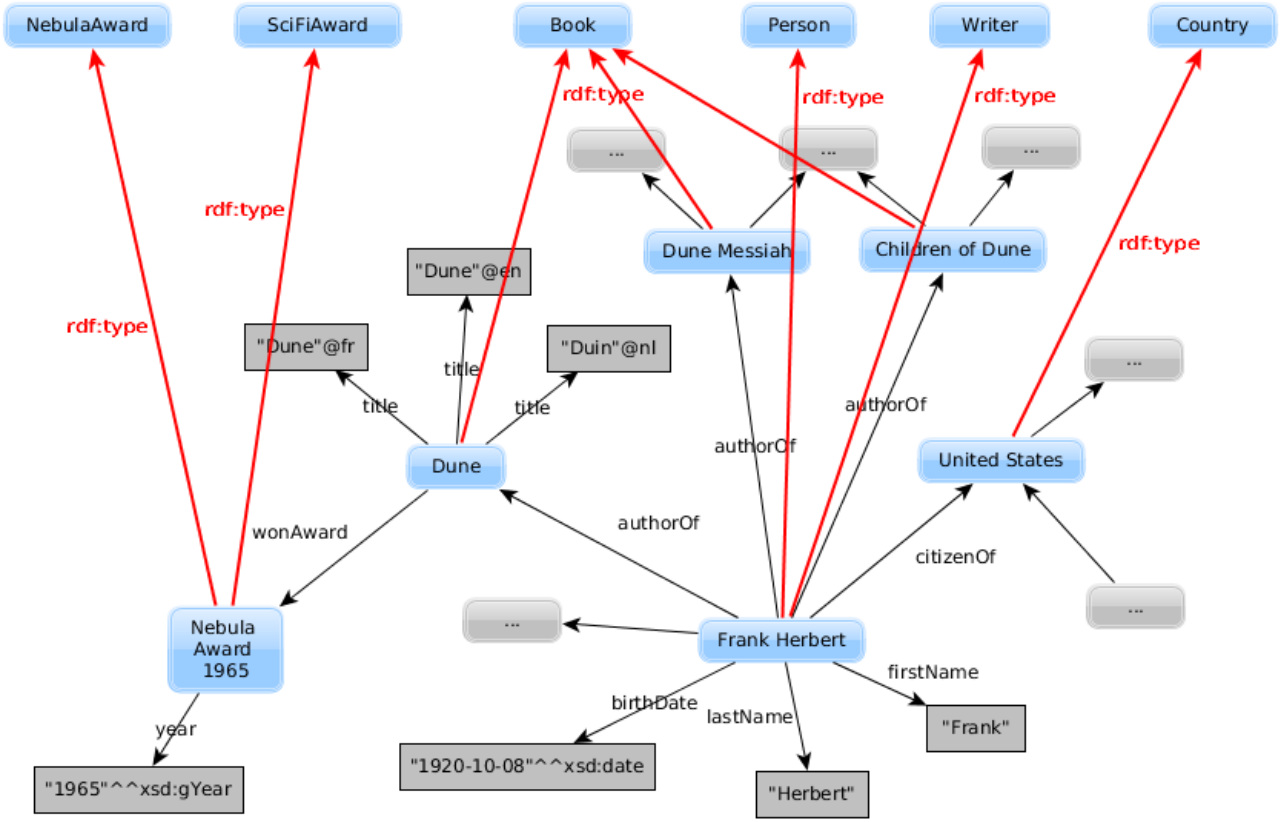
Making entities' categories explicit



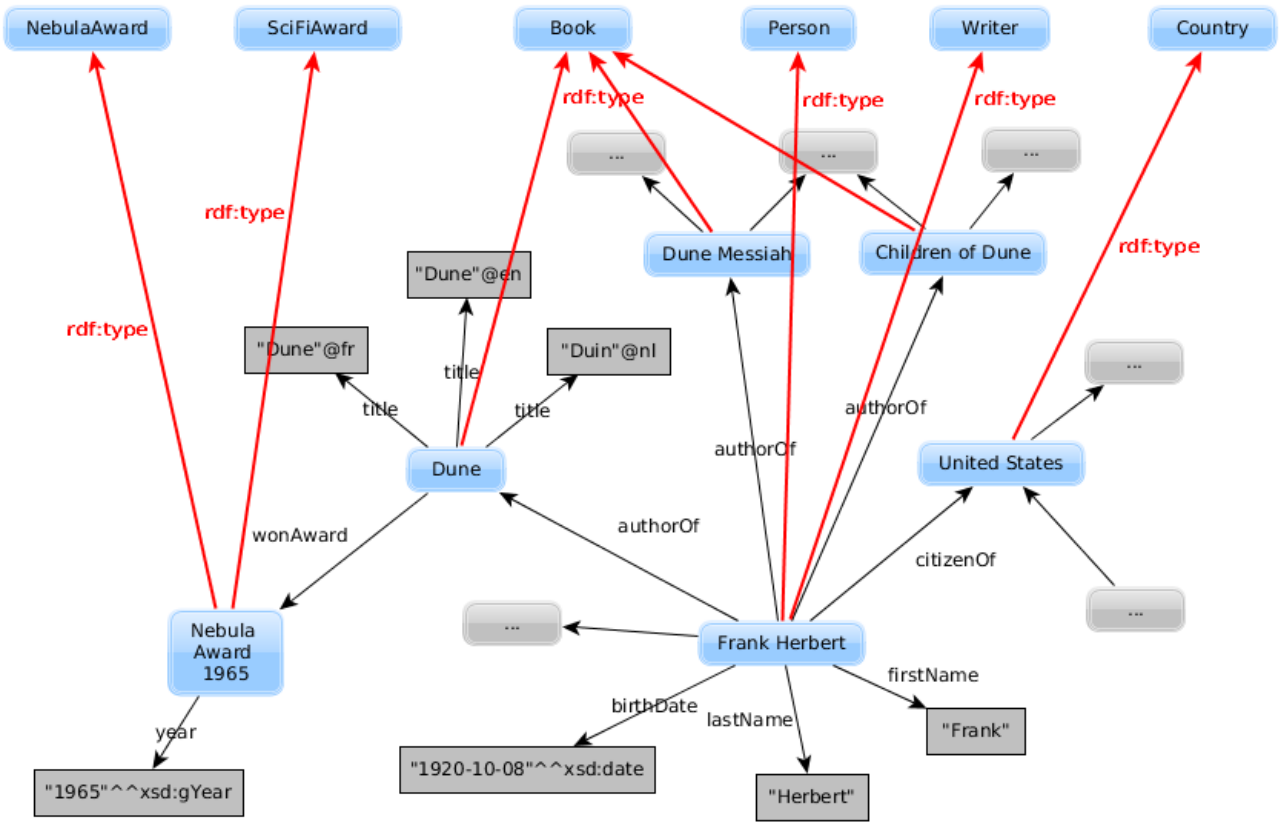
Describing the entities (Dune, Frank Herbert,...) characteristics and the relations between these entities fails to capture the fact that Dune, Dune Messiah and The children of Dune are:

- rather similar
- rather different from the other entities (Frank Herbert, USA,...)

Describe the entities' categories using `rdf:type`



Describe the entities' categories using `rdf:type`



There can also be relations between categories (e.g. between `Writer` and `Person`), but these are typically relations between sets that have a precise semantics (inclusion, disjointness,...). This will be covered when we talk about ontologies, RDFS and OWL.

RDF politeness

- ☑ reuse existing URIs as much as possible
  - for
    - resources (from databases)
    - properties (from ontologies)
    - classes (from ontologies)
  - do not hijack URIs
  - easy (somehow) in life science
- ☑ corollary: reuse classical prefixes too (`http://prefix.cc/`)
- ☑ provide $\geqslant 1$ `rdfs:label` of the resources you create
- ☑ associate your resources to classes with `rdf:type`
- ☑ consider providing an `rdfs:comment`

Exercice 1: understand an RDF graph

See https://gitlab.com/odameron/eins2023/-/blob/main/exercice/eins2023-exercice01.md

RDF in practice

RDF can be stored:

- in files (several formats: n-triples, turtle, n3, xml-rdf,...)
- in triplestores ($\sim$ DB): Virtuoso, fuseki, graphDB, sesame & RDF4J, Oracle, 4store,...
- within HTML: RDFa

RDF is supported by most languages (and most triplestores have bindings)

RDF format: N-triples

- 1 triple / line
  - URI delimited by angle brackets ("<" and ">")
  - subject, predicate and object separated by space
  - line ends with a colon (".")
- no order between triples
- order within triple matters (obviously)

Straightforward, but not easy to read/write (for humans)

RDF format: N-triples

```
1
2  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH>
•  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#firstName> "Frank" .
3  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH>
•  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#lastName> "Herbert" .
4  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH>
•  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#citizenOf>
•  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#USA> .
5  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH>
•  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#authorOf>
•  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#Dune> .
6  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH>
•  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
•  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#Author> .
7  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH>
•  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
•  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#Person> .
8
9  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#Dune>
•  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
•  <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#Book> .
10 <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#Dune>
```

RDF format: Turtle

- N-triples syntax still possible
- comments start with "#" until end of line
- prefix declaration for CURIE
  - `PREFIX prefixName: <URI template>`
  - usually at begining of file
  - you can choose any prefix you like
    - reuse the usual ones
    - `http://prefix.cc` is your friend
- 1 triple / line
  - URIs can be full URIs (<...>) or CURIE (no delimiters)
  - a triple can mix both
  - line ends with ";" ⇒ next line has
    - same subject (no need to repeat)
  - line ends with "," ⇒ next line has
    - same subject (no need to repeat)
    - same property (no need to repeat)

RDF format: Turtle

```
@prefix sf: <http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

#
# FRANK HERBERT
#
sf:FH sf:firstName "Frank" .
sf:FH sf:lastName "Herbert" .
sf:FH sf:citizenOf sf:USA .
sf:FH sf:authorOf sf:Dune .
sf:FH rdf:type sf:Author .
sf:FH rdf:type sf:Person .

sf:Dune rdf:type sf:Book .
sf:Dune rdf:type sf:SciFiBook .
sf:Dune sf:title "Dune" .
sf:Dune sf:wonAward sf:Nebula1965 .
sf:Dune sf:wonAward sf:Hugo1966 .

sf:Nebula1965 rdf:type sf:NebulaAward .
sf:Hugo1966 rdf:type sf:HugoAward .
```

RDF format: N3

Do not confuse with N-Triples!

- N3 = turtle + additional features
  - Named graphs
  - Inference rules
  - ...
- not a W3C recommendation

- every n-Triples file is turtle-compatible
- every turtle file is N3-compatible

RDF format: RDF-XML

- XML serialization of RDF
- a major pain to read or edit these by hand

RDF format: RDF-XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:sf="http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#">
  <rdf:Description rdf:about="http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH">
    <sf:firstName>Frank</sf:firstName>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH">
    <sf:lastName>Herbert</sf:lastName>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH">
    <sf:citizenOf rdf:resource="http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#USA"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH">
    <sf:authorOf rdf:resource="http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#Dune"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH">
    <rdf:type rdf:resource="http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#Author"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#FH">
    <rdf:type rdf:resource="http://www.irisa.fr/dyliss/public/odameron/SciFiAwards#Person"/>
  </rdf:Description>
```

Exercice 2: understand the RDF turtle syntax

See https://gitlab.com/odameron/eins2023/-/blob/main/exercice/eins2023-exercice02.md

Exercice 3: write an RDF graph

See https://gitlab.com/odameron/eins2023/-/blob/main/exercice/eins2023-exercice03.md

RDF synthesis

RDF graphs

- are sets of triples `<subject, predicate, object>`
- the principles are straightforward

How to find the relevant information?

- RDF simplicity does not shield us from the inner complexity of the domain (we had been warned at slide 18)
- just looking at the triples or the visual representation of the graph will seldom yield any insights
- we need a formal representation of patterns allowing to systematically traverse the graph according to some rules

Semantic Web: SPARQL for querying RDF

SPARQL query structure

Prefixes declaration

(optional)

SELECT

list of the **variables** you want to find the values of (separated by spaces)

WHERE {...}

- **constraints** on the variables
  - how they are connected to some resources
  - how they are connected to other variables
  - which value(s) we are interested in
- each constraint is a triple pattern
- together, the triple patterns form a graph pattern

Looks a lot like an SQL query (in a more friendly way)...
this is not a coincidence :-)

Triple patterns extend RDF triples with variables

Variables = the things you look for in the graph of data

- their name begins with a question mark (e.g. `?myVariable`)
- use them in triple patterns to specify where to look

Triple pattern = RDF triple generalized with variable(s)

A **triple pattern** is a `<subject, predicate, object>` triple where:

- the `subject` can be an URI or a variable
- the `predicate` can be an URI or a variable
- the `object` can be an URI, a datatype or a variable

- `ex:P1 rdfs:label "Protein 1" .`
- `?prot rdfs:label "Protein 3" .`
- `ex:P5 rdfs:label ?protLabel .`
- `ex:P3 ?someRelation ex:P2 .`
- `ex:P4 ?someRelation ?someValue .`

Graph patterns

Graph pattern = set of triple patterns

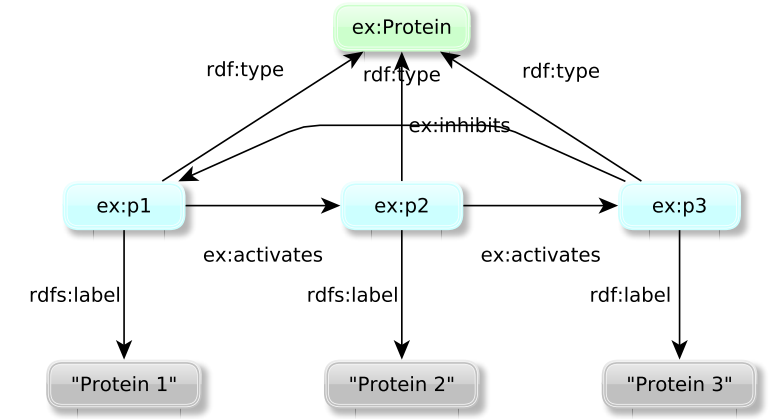
- A **Graph pattern** defines a graph template
- A variable can connect multiple triple patterns (just like an URI)

SPARQL query result on an RDF graph

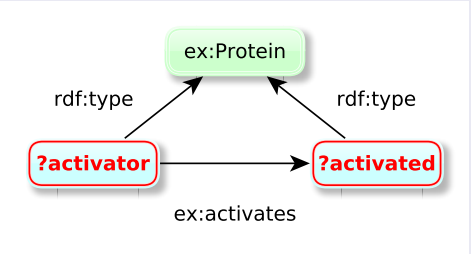
- compute the set of isomorphisms between the graph pattern and the RDF graph
- each isomorphism
  - is a solution (i.e. a row in the table of results)
  - defines a value for each variable

SPARQL query example on an RDF graph

Data



Query




```
1 SELECT DISTINCT ?activator ?activated
2 WHERE {
3   ?activator rdf:type ex:Protein .
4   ?activator ex:activates ?activated .
5   ?activated rdf:type ex:Protein .
6 }
```

Result

?activator	?activated
ex:p1	ex:p2
ex:p2	ex:p3

Exercice 4: your first SPARQL queries

See https://gitlab.com/odameron/eins2023/-/blob/main/exercice/eins2023-exercice04.md

SPARQL: synthesis

SPARQL queries

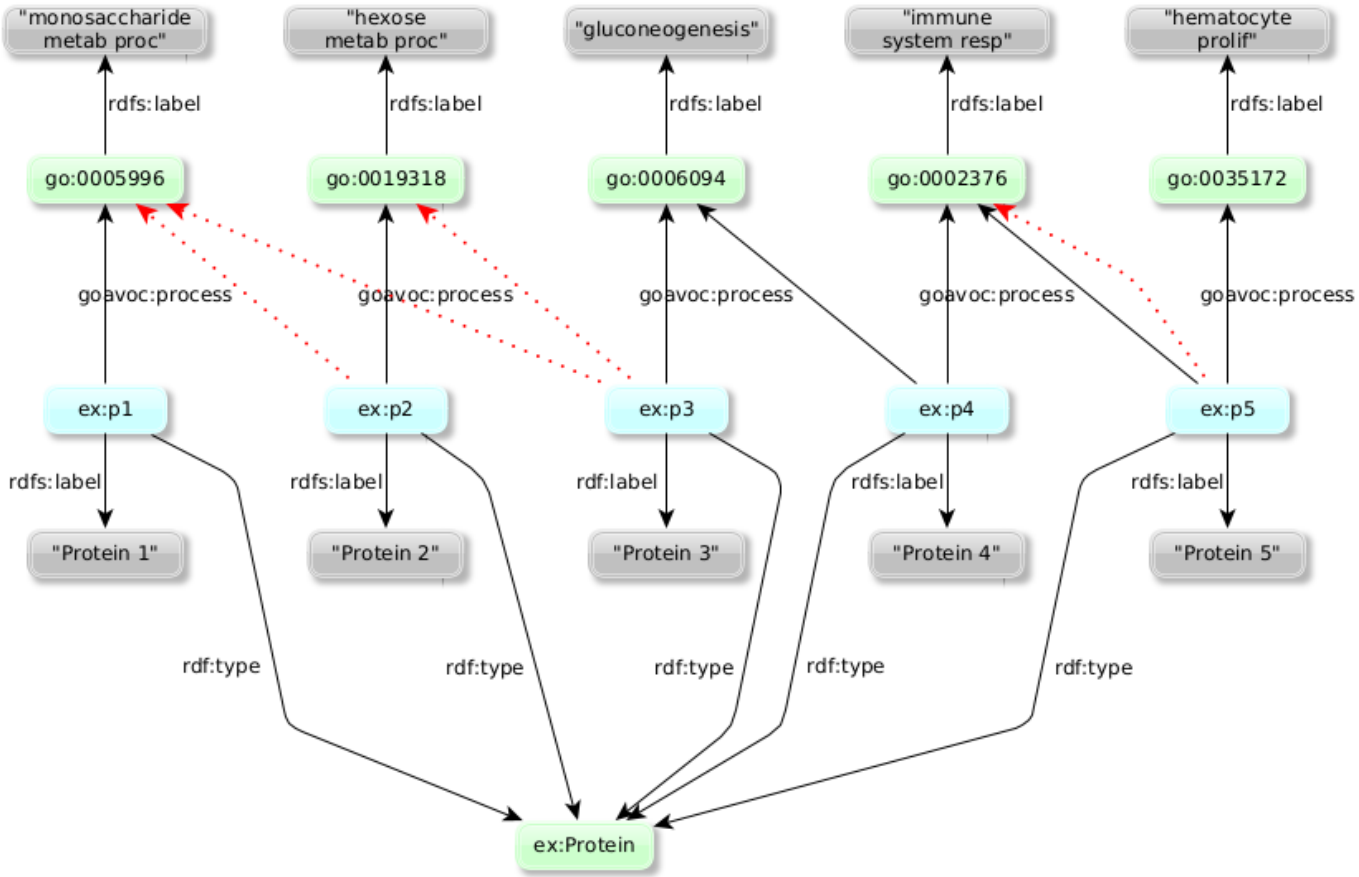
- provide some rules for traversing the dataset graph
- (troll) are more user-friendly than SQL (and more versatile)

there are still some implicit dependencies that are ignored

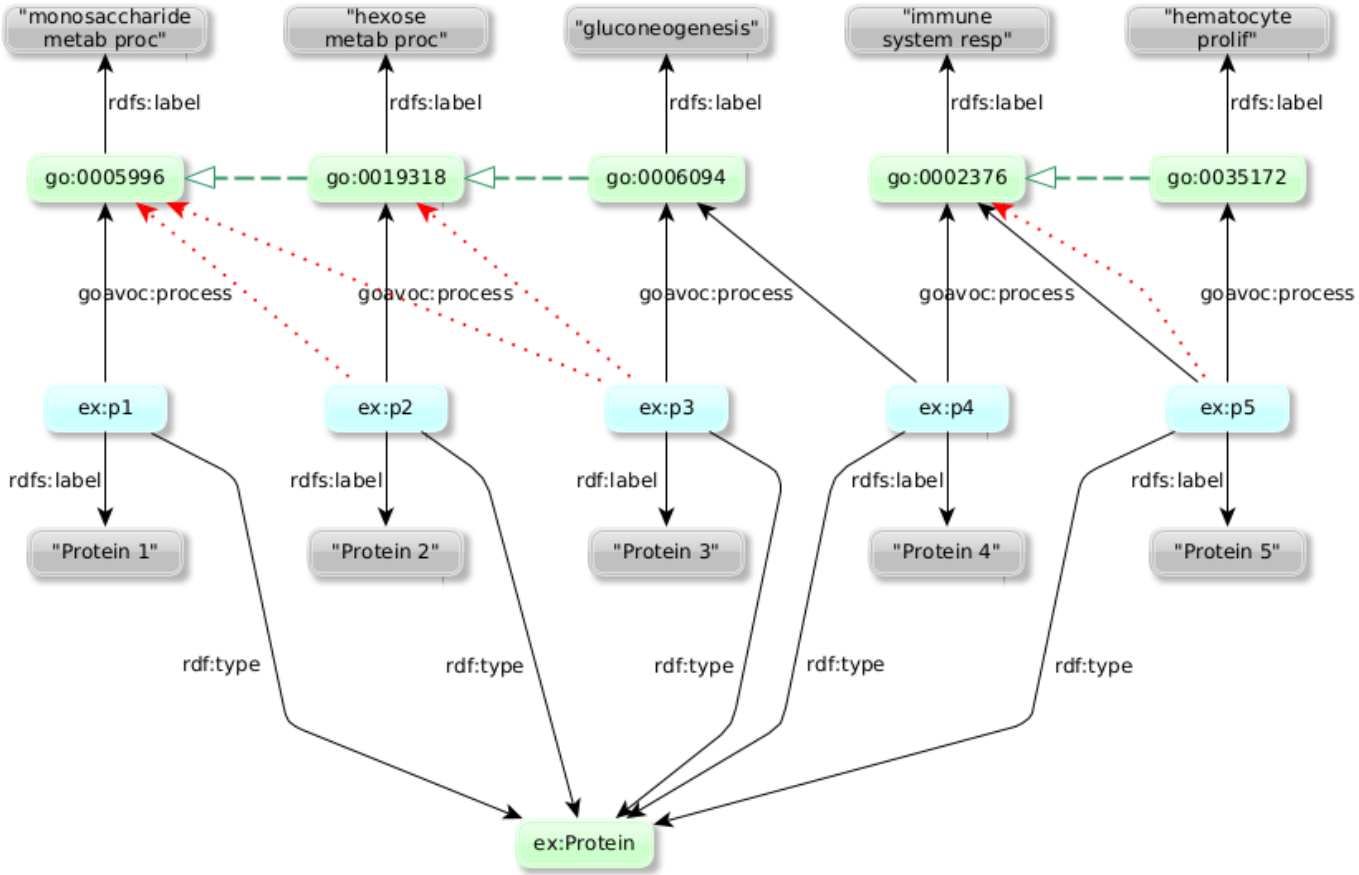
- the proteins involved in hexose metabolic process should also be involved in monosaccharide metabolic process
- the proteins involved in gluconeogenesis should also be involved in hexose metabolic process (and therefore also in monosaccharide metabolic process)
- idem for hematocyte proliferation and immune system response
- well, make them explicit, then (you should be used to it by now)...
- ... just don't do it in your dataset
  - your dataset is fine
  - these dependencies are generic and refer to domain knowledge...
  - ... therefore they fit in some ontologies
  - (and we should adapt our SPARQL queries)

Semantic Web: RDFS and OWL for representing knowledge

Missing annotations due to implicit dependencies



Reasoning with ontologies reveals otherwise implicit dependencies



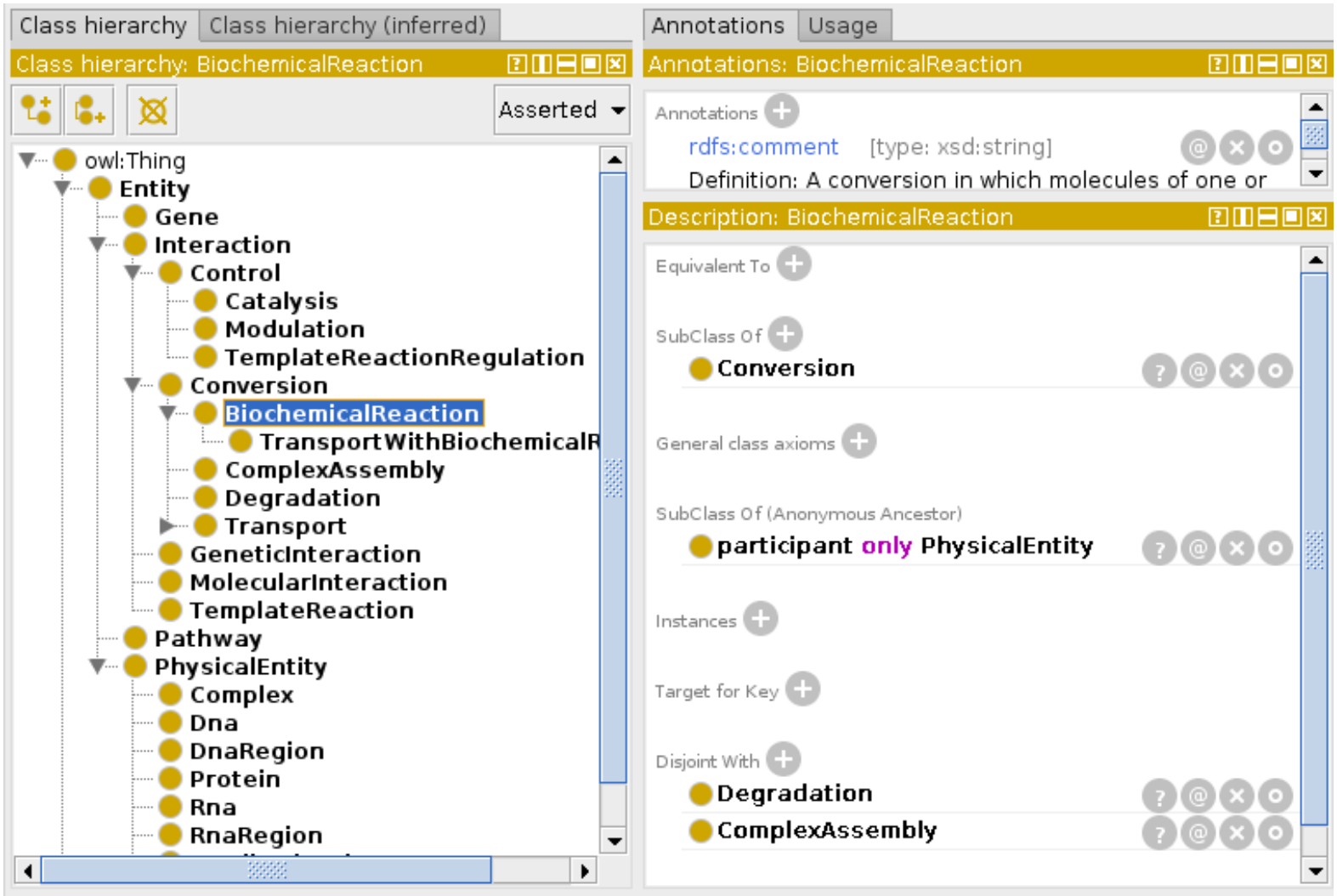
Representing knowledge with RDFS and OWL

- RDFS (RDF Schema)
  - notion of class as a set of elements
    (`ex:Writer rdf:type rdfs:Class`)
  - hierarchies of classes
    (`ex:Writer rdfs:subClassOf foaf:Person`)
  - notion of relation (aka "property")
    (`ex:hasWriten rdf:type rdfs:Property`)
  - hierarchies of relations
    (`ex:hasWriten rdfs:subPropertyOf ex:hasCreated`)
  - relations' domain and range
    (`ex:hasWriten rdfs:domain foaf:Person`)
    (`ex:hasWriten rdfs:range ex:Book`)
- OWL adds richer semantic constraints on top of RDFS (disjointness, necessary and sufficient definitions, inverse properties, transitivity...)

RDFS and OWL

- are defined on top of RDF (special entities and properties)
- have a syntax compatible with RDF

RDFS class hierarchy (left) and OWL logical constraints (right)



Exercice 5: representation of ontologies in RDFS and OWL

See https://gitlab.com/odameron/eins2023/-/blob/main/exercice/eins2023-exercice05.md

Protégé

- is a user interface for modeling knowledge
- that generates the RDFS and OWL code as regular RDF
  (RDFS and OWL are just special RDF resources and properties)

Now that we know how knowledge is represented, we can take it into account in our SPARQL queries

RDFS and OWL synthesis

RDFS and OWL are special classes and relations defined in RDF

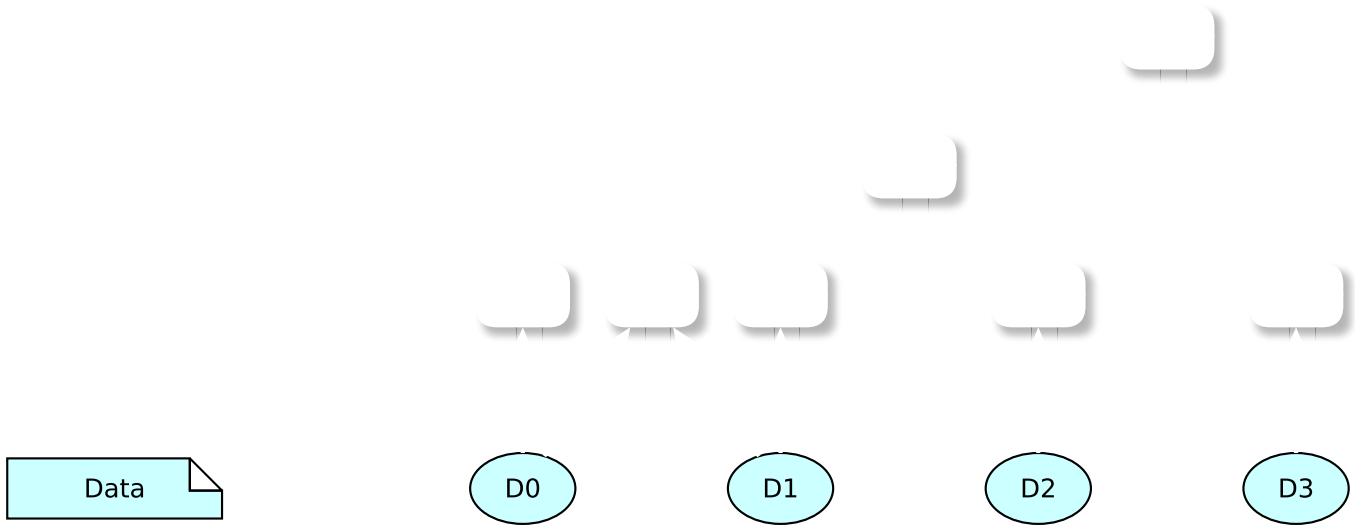
- Principles
  - classes are represented as instances (with the usual ‘rdf:type‘) of ‘owl:Class‘ or ‘rdfs:Class‘ (which are therefore metaclasses)
  - taxonomies are represented with the ‘rdfs:subClassOf‘ relation between classes
  - other RDFS and OWL features are also represented in RDF
    - OWL extends RDFS with richer reasoning capabilities
    - ‘owl:Class‘ is actually an ‘rdfs:subClassOf‘ of ‘rdfs:Class‘ to achieve compatibility
- An unified homogeneous formalism for representing data (in RDF) and knowledge (in RDFS and OWL)...
- ... therefore accessible to SPARQL, as well as reasoners

Exercice 6: semantically-rich SPARQL queries

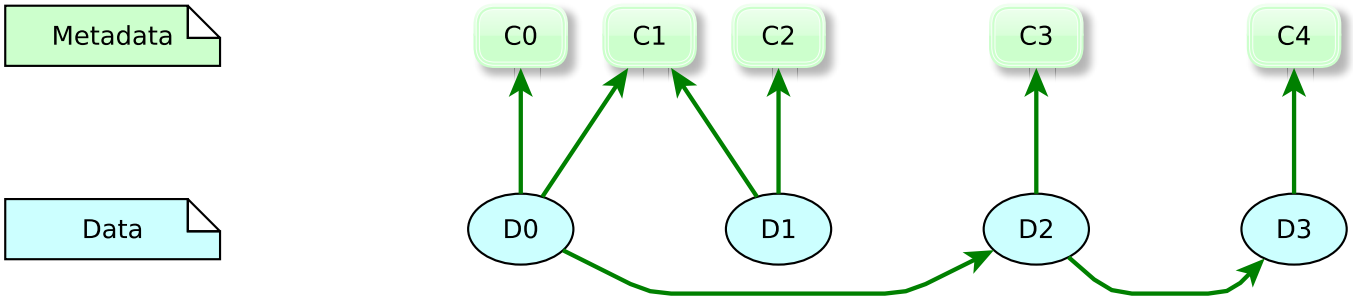
See https://gitlab.com/odameron/eins2023/-/blob/main/exercice/eins2023-exercice06.md

Synthesis

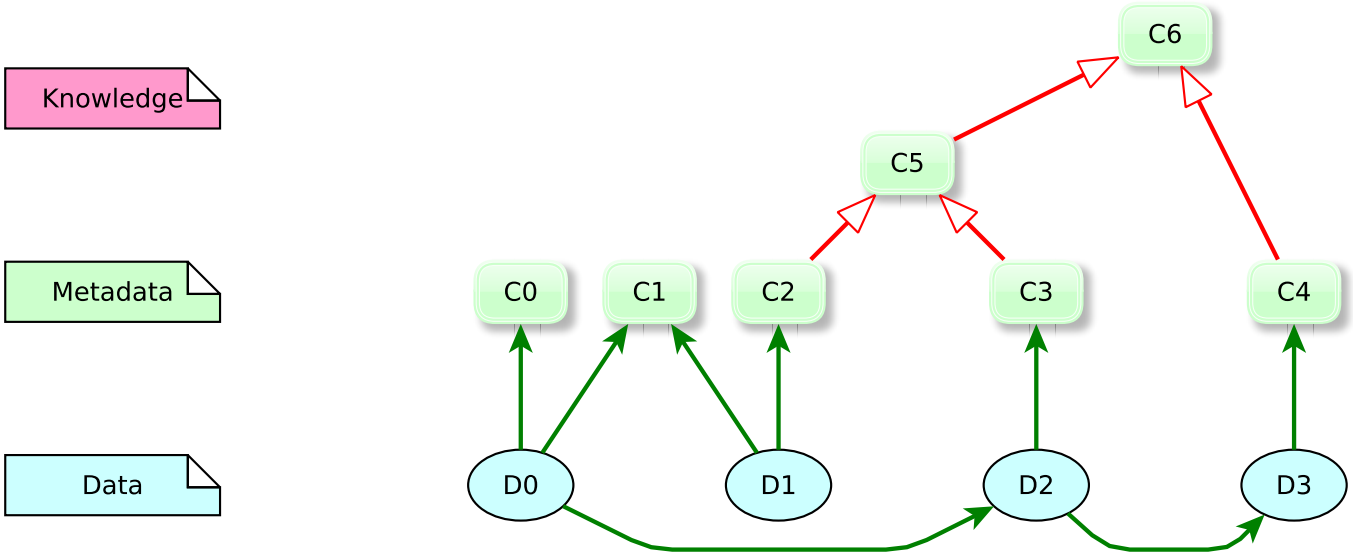
Integrated representation of data with RDF and knowledge with RDFS and OWL



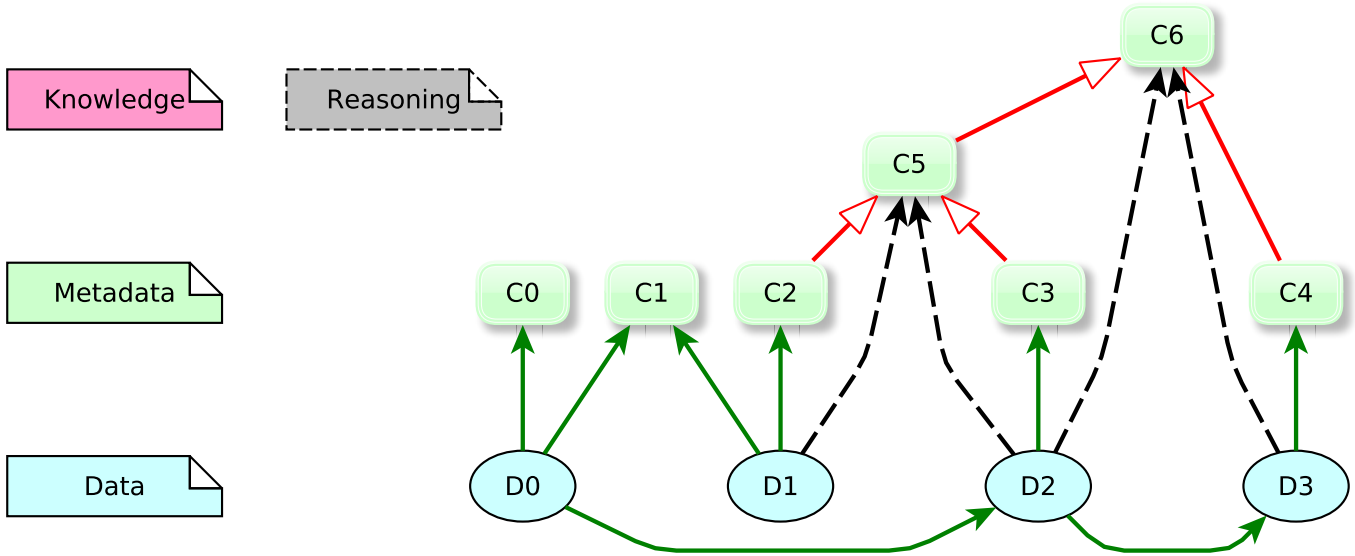
Integrated representation of data with RDF and knowledge with RDFS and OWL



Integrated representation of data with RDF and knowledge with RDFS and OWL



Integrated representation of data with RDF and knowledge with RDFS and OWL



Synthesis

- Life science **data** are distributed and highly complex
- **Annotations** address complexity
- Ontologies formalize the **knowledge** underlying annotations
  - Reused across datasets
  - Shared for interoperability
  - Support reasoning
  - For programs more than humans
- the Semantic Web provides an infrastructure supporting integration and **reasoning**
- Life science annotations and ontologies have matured