Advanced Topics in Artificial Intelligence

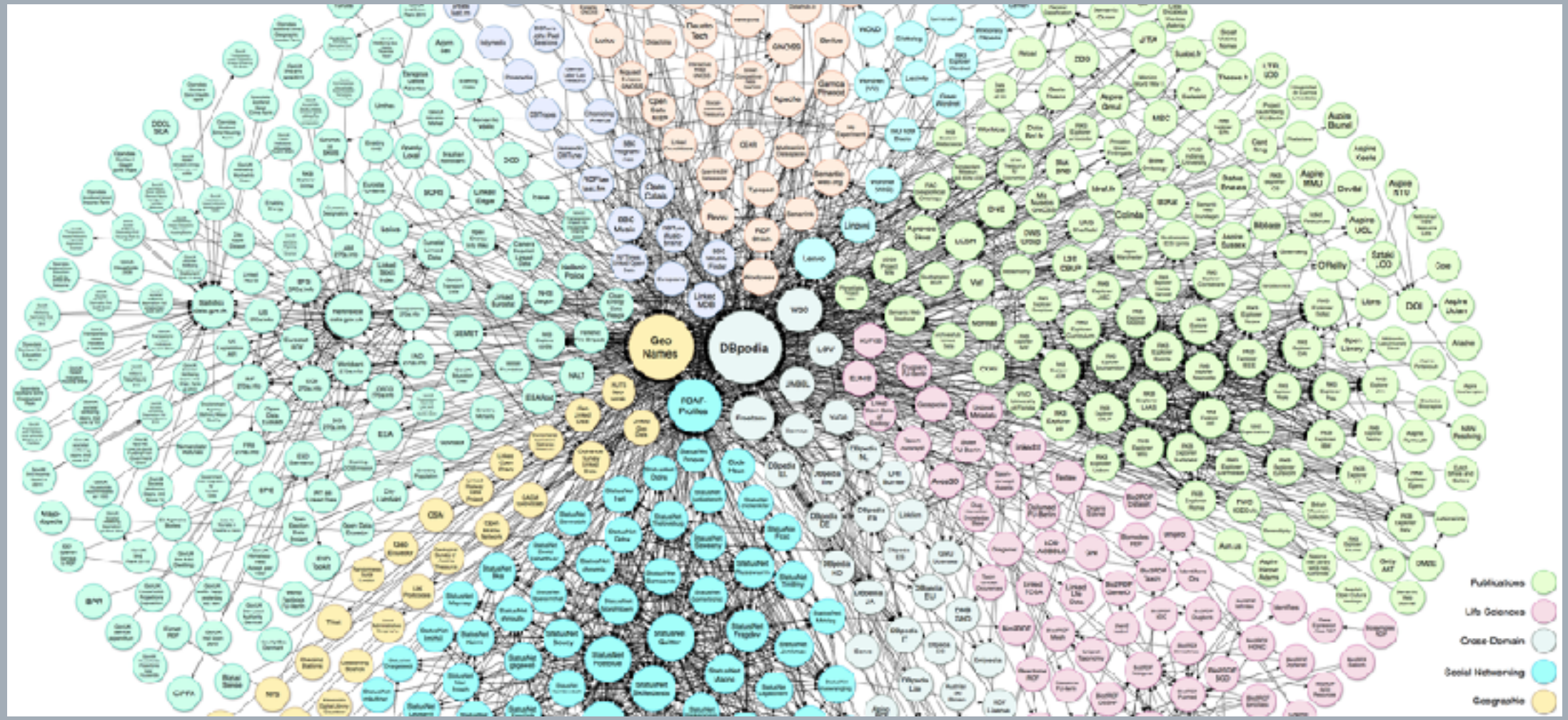
Large-Scale KR: Knowledge Graphs

Abraham Bernstein

Based on:

- Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia D'amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, Axel-Cyrille Ngonga Ngomo, Axel Polleres, Sabbir M. Rashid, Anisa Rula, Lukas Schmelzeisen, Juan Sequeda, Steffen Staab, and Antoine Zimmermann. 2021. **Knowledge Graphs**. *ACM Comput. Surv. 54*, 4, Article 71 (July 2021), 37 pages. DOI: https://doi.org/10.1145/3447772
- Slides by Daniele Dell'Aglio (former DDIS member now Prof in Aalborg)

The Semantic Web



Smart Agents vs. Smart Data

- There are two ways to push «intelligence» in this setting:
  - **Smart agents**: agents use sophisticated algorithms to process data
    - Natural language processing
    - Machine learning
    - Deep learning
  - **Smart data**: the data embeds additional data that describe the content
    - Structured data
- The best option?

What is the Semantic Web?

The Semantic Web is a web of data, in some ways like a global database

Tim Berners-Lee, 09.1998
https://www.w3.org/DesignIssues/Semantic.html

- A web as a database enables new automatic operations
  - **Search**: manage synonyms and homonyms, consider the context of the query
  - **Personalization**: tailor the content of a web site based on agents' descriptions
  - **Linking**: dynamic decisions about the paths to follow to navigate web pages
  - **Integration**: collect information among different web sites and process it without mental copy-paste operations

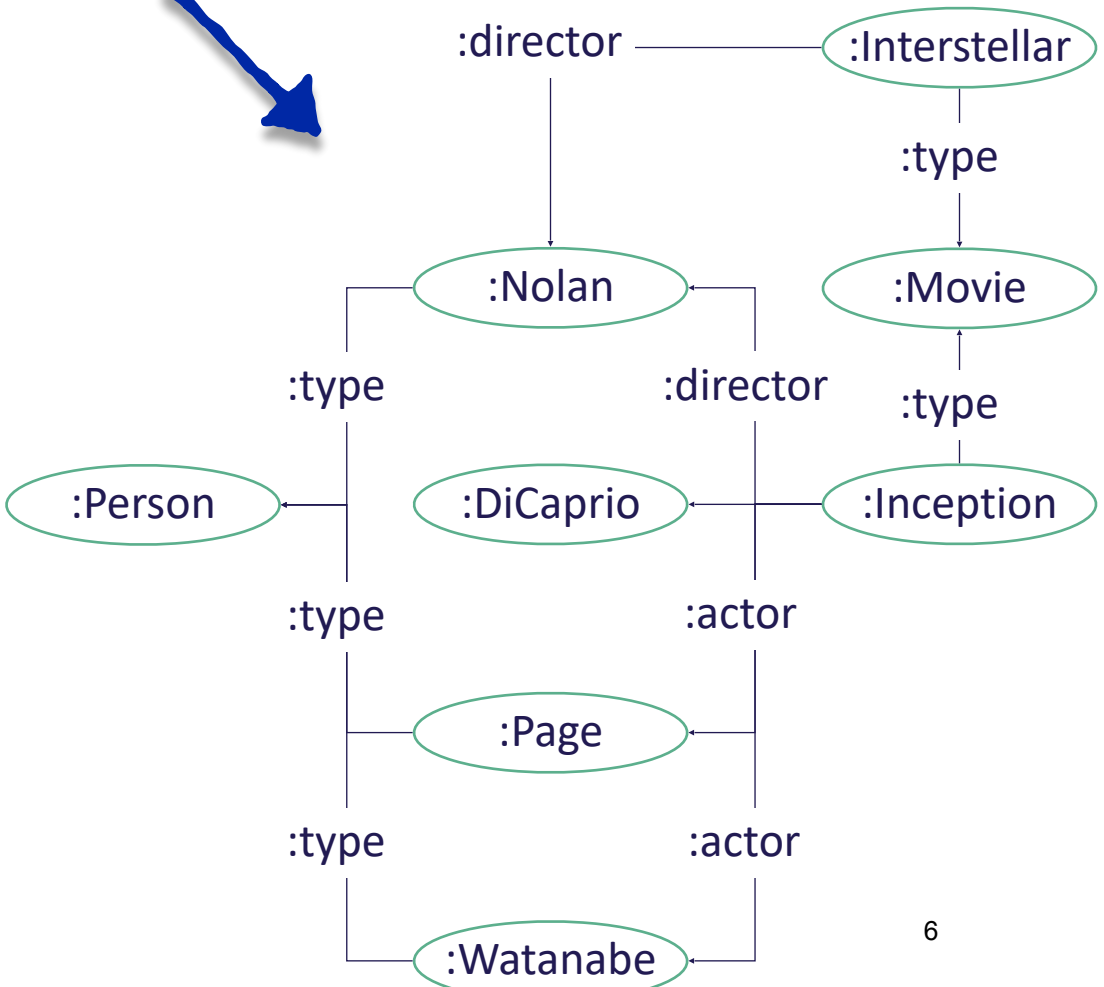
Structured and semi-structured data in the web

- HTML is the base technology to create web documents
  - Hypertext Markup Language
  - Content is organized according to presentation information (titles, tables, lists, highlights)
  - Lack of structured data

- However, structured data exist
  - Web pages are usually generated from databases and Content Management Systems (CMSs)
  - We could link the underlying structured datasets

Publishing data on the web

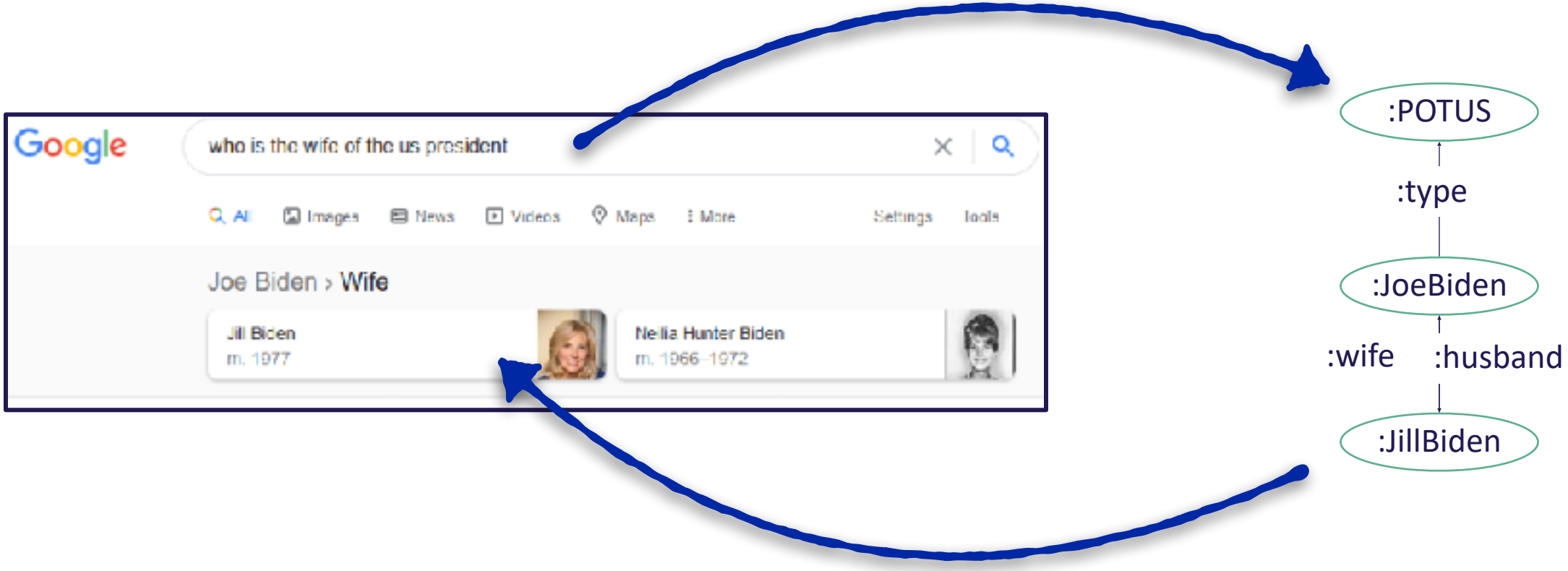
https://www.imdb.com/title/tt1375666



Enterprise knowledge graphs

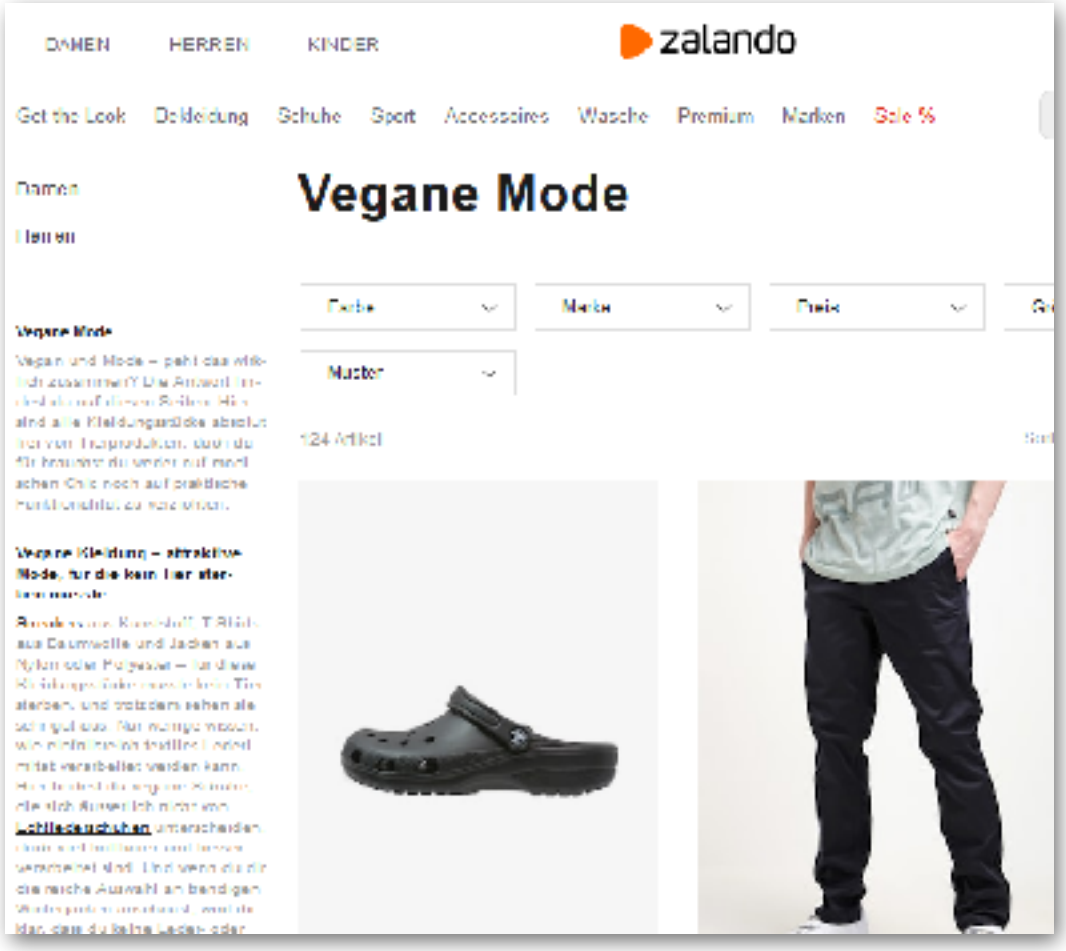
[Frank van Harmelen - https://www.slideshare.net/Frank.van.Harmelen/adoption-of-knowledge-graphs-late-2019]

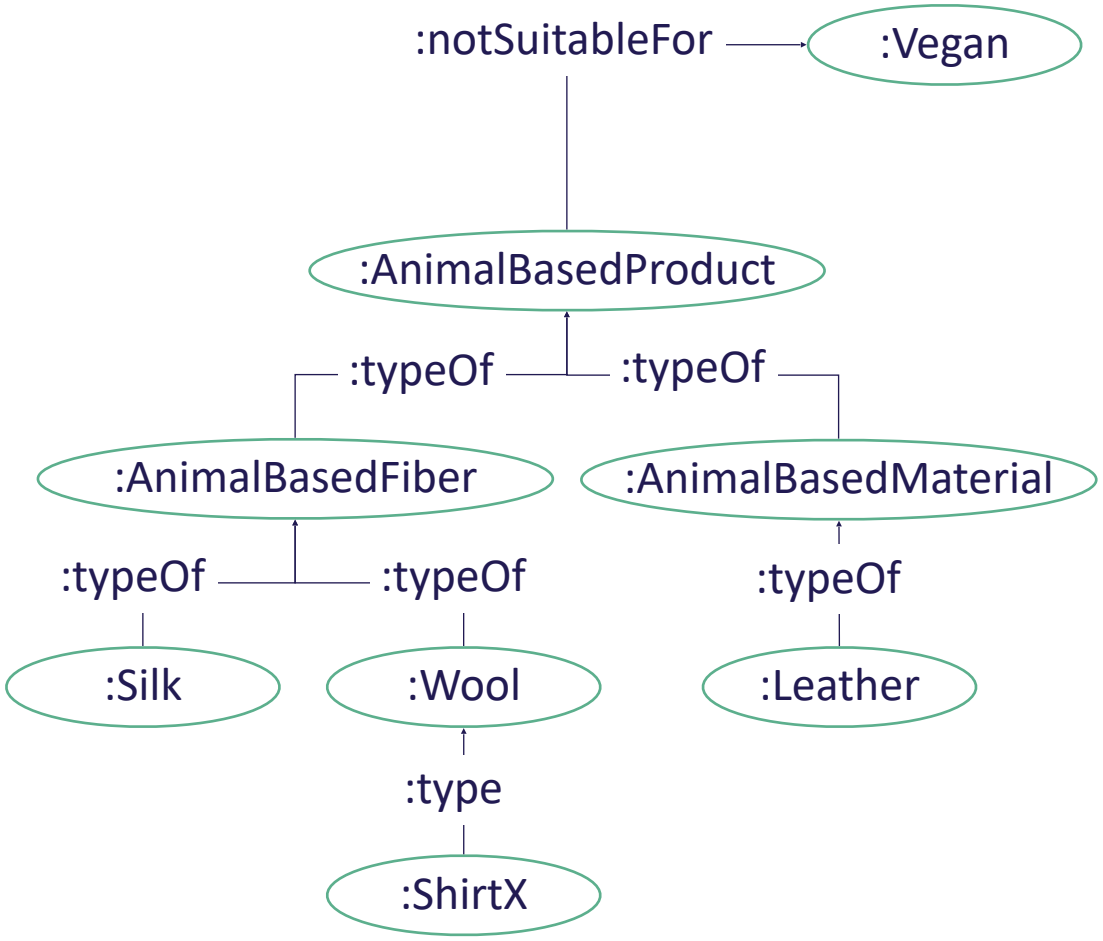
Question answering



[Google - https://www.google.com/search?q=who+is+the+wife+of+the+us+president]

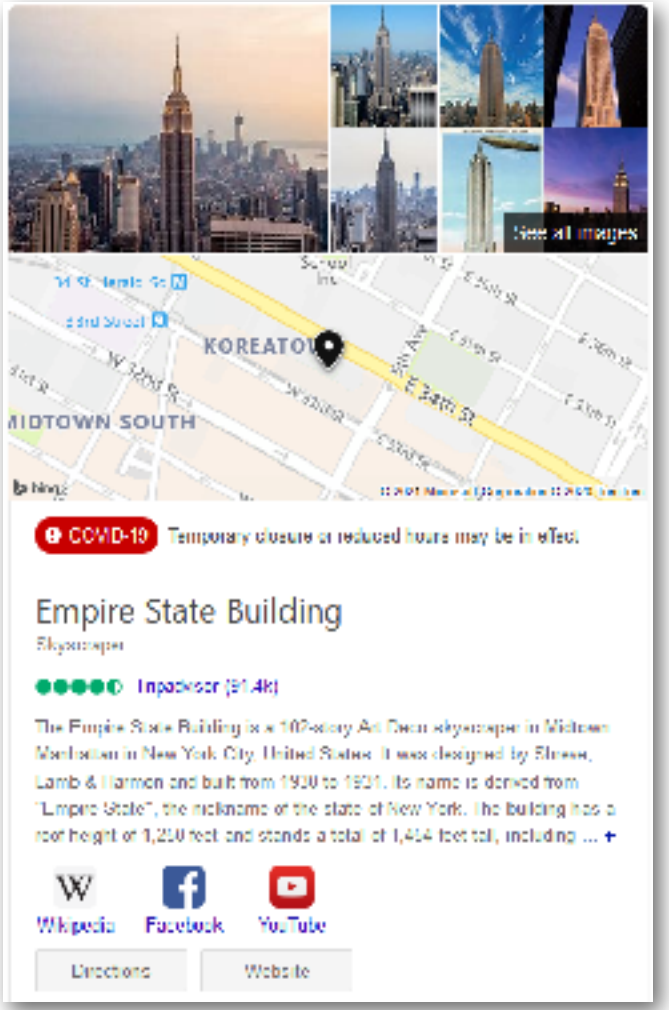
Summaries

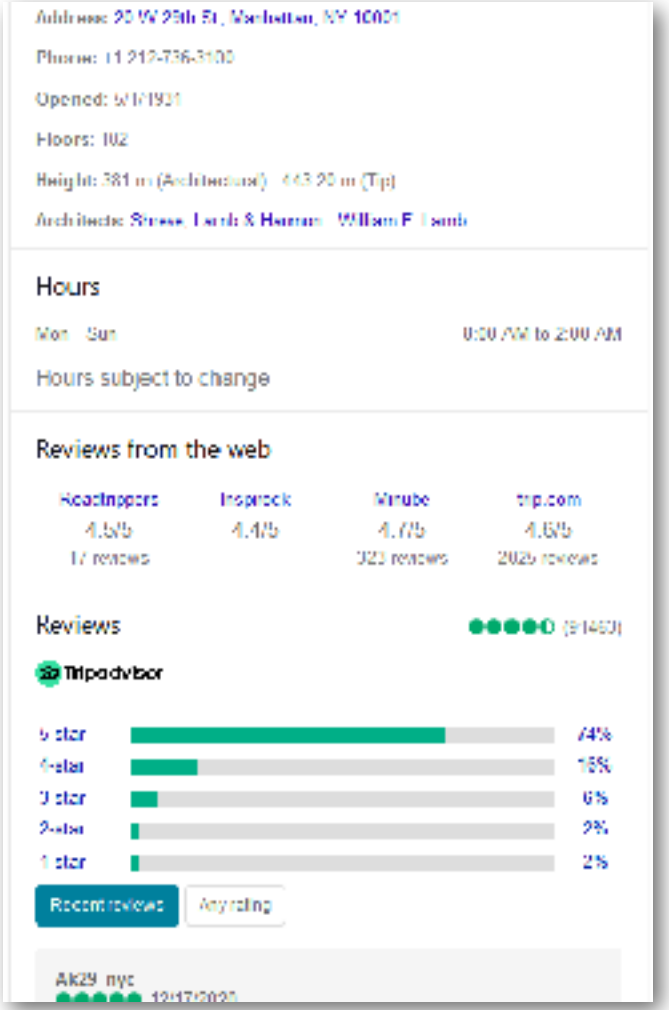


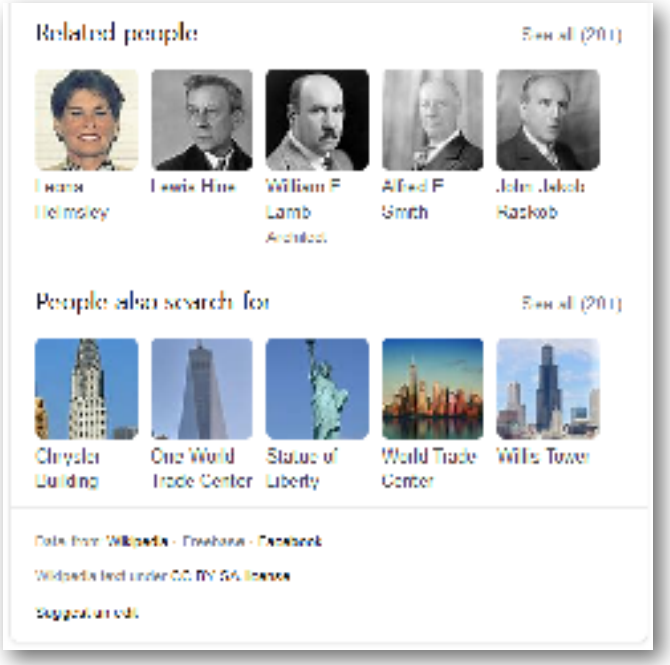


[Zalando - https://tinyurl.com/zalando-sem-web]

Rich snippets

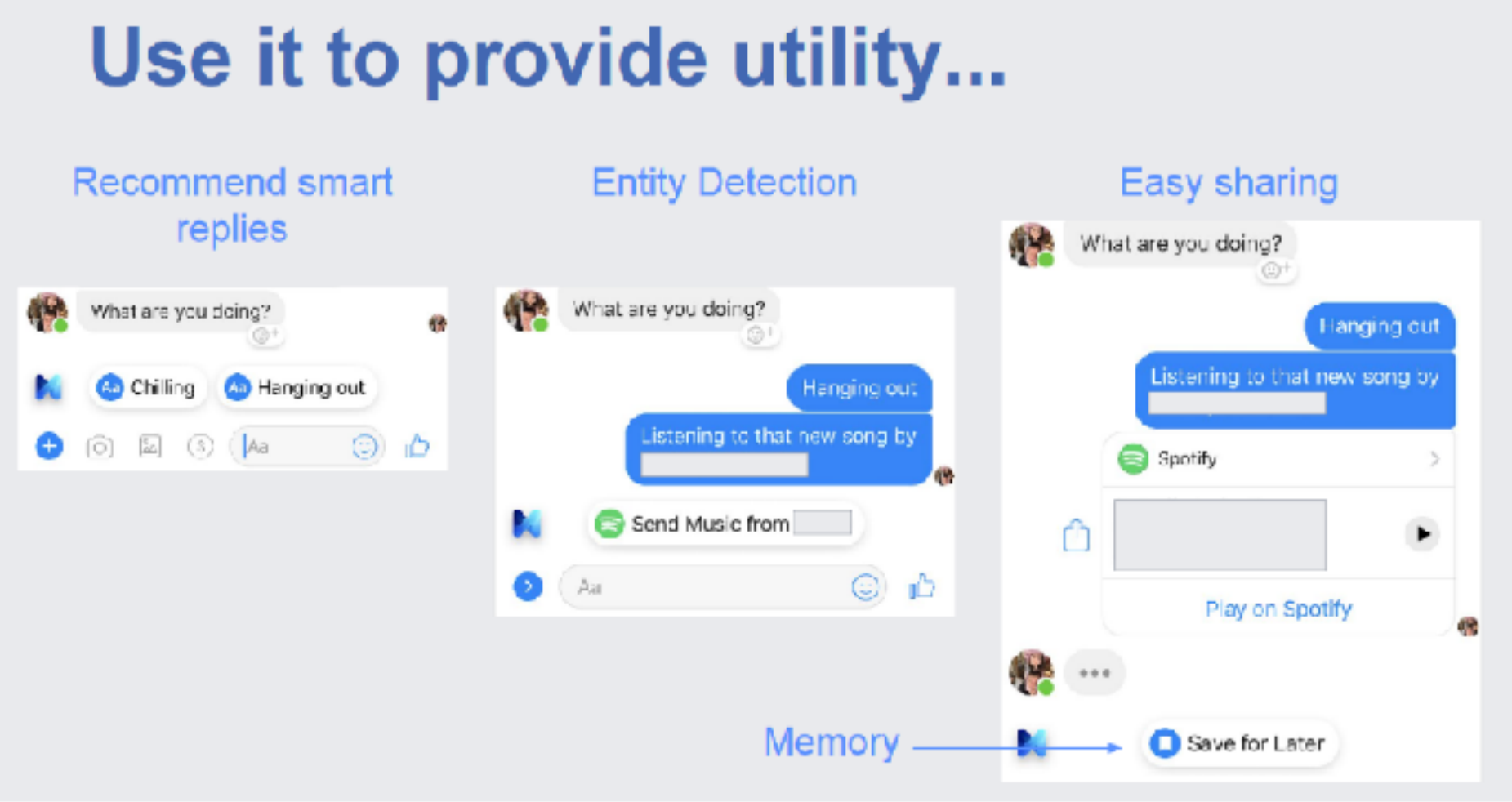






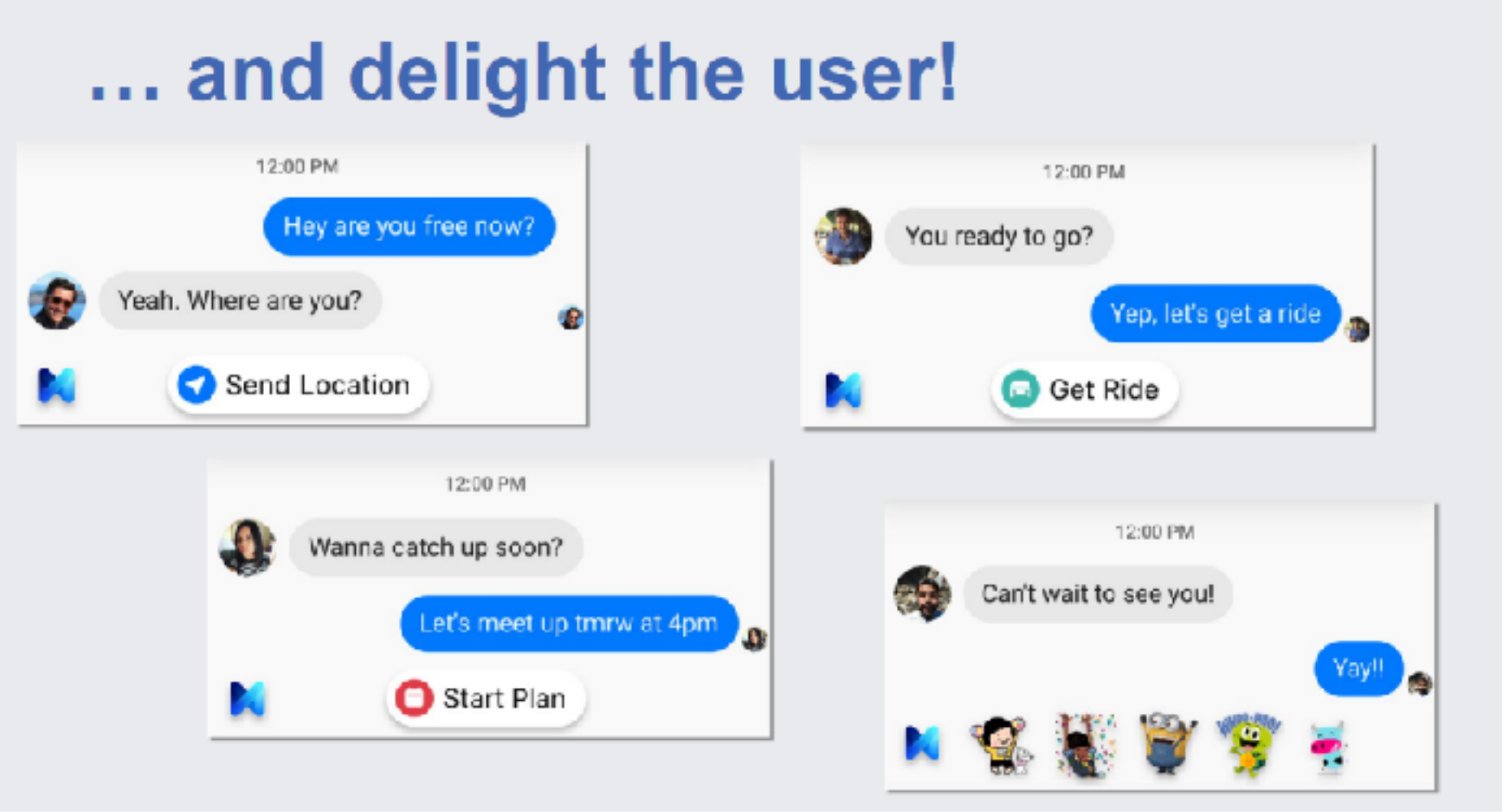
[Microsoft Bing - https://www.bing.com/search?q=empire+state+building]

How Facebook uses its knowledge graph (1/2)



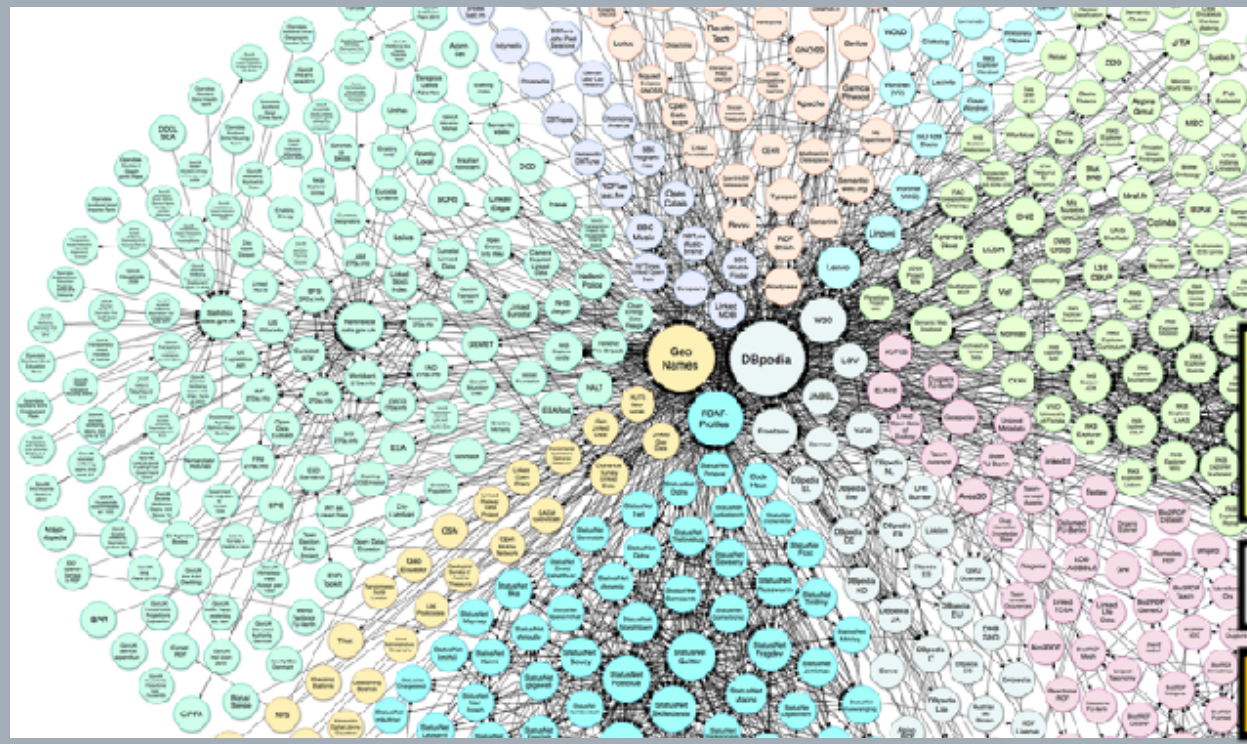
[Facebook - http://iswc2018.semanticweb.org/wp-content/uploads/2018/10/Panel-all.pdf]

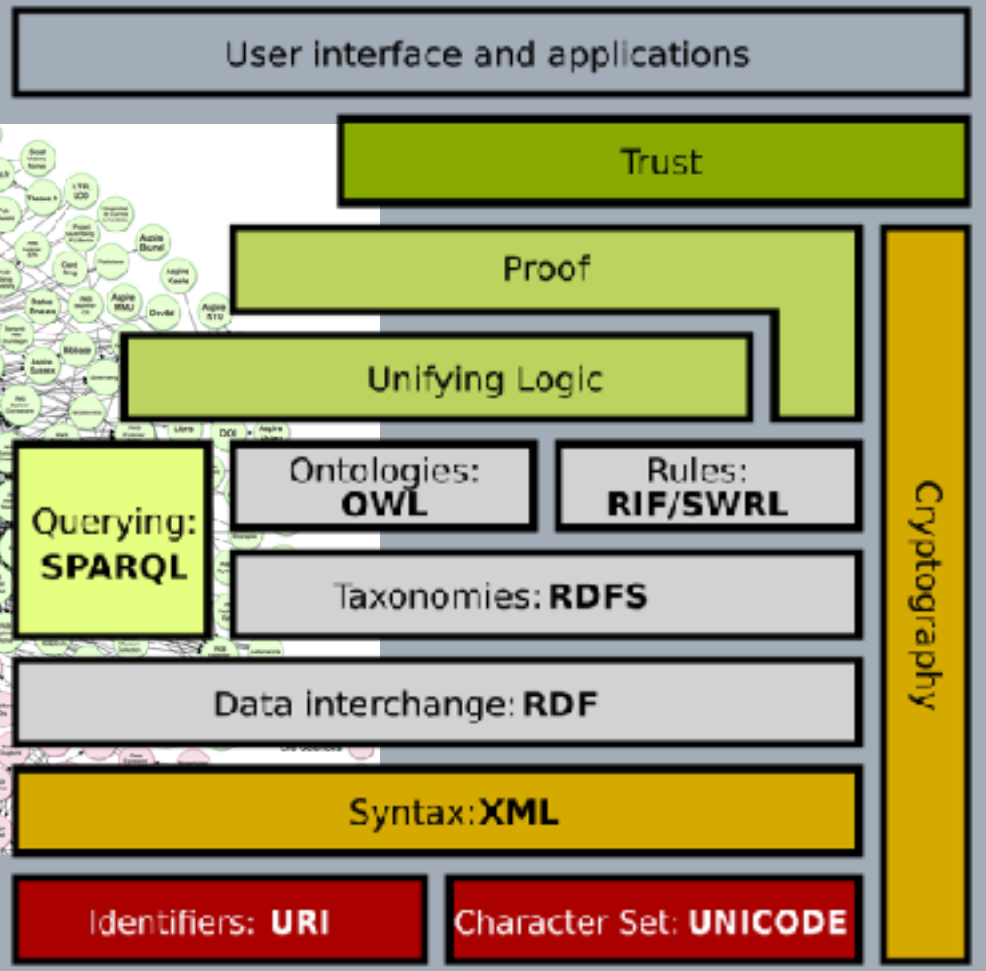
How Facebook uses its knowledge graph (2/2)



[Facebook - http://iswc2018.semanticweb.org/wp-content/uploads/2018/10/Panel-all.pdf]

From The Web to the Semantic Web — A Quick Overview





The Web of Documents — HTML

```
<h1>Battle of Waterloo</h1>
...
<p>The <b>Battle of Waterloo</b> was
fought on Sunday, 18 June 1815, near <a
href="…">Waterloo</a> in <a href="…"
>Belgium<a>, part of the <a href="…"
>United Kingdom of the Netherlands <a
href="…"> at the time. A French army under
the command of <a href="…">Napoleon
Bonaparte</a> was defeated by two of the
armies of the …
```

The Web of Documents — HTML

```
<h1>Battle of Waterloo</h1>
...
<p>The <b>Battle of Waterloo</b> was
fought on Sunday, 18 June 1815, near <a
href="…">Waterloo</a> in <a href="…"
>Belgium<a>, part of the <a href="…"
>United Kingdom of the Netherlands <a
href="…"> at the time. A French army under
the command of <a href="…">Napoleon
Bonaparte</a> was defeated by two of the
armies of the …
```

Place

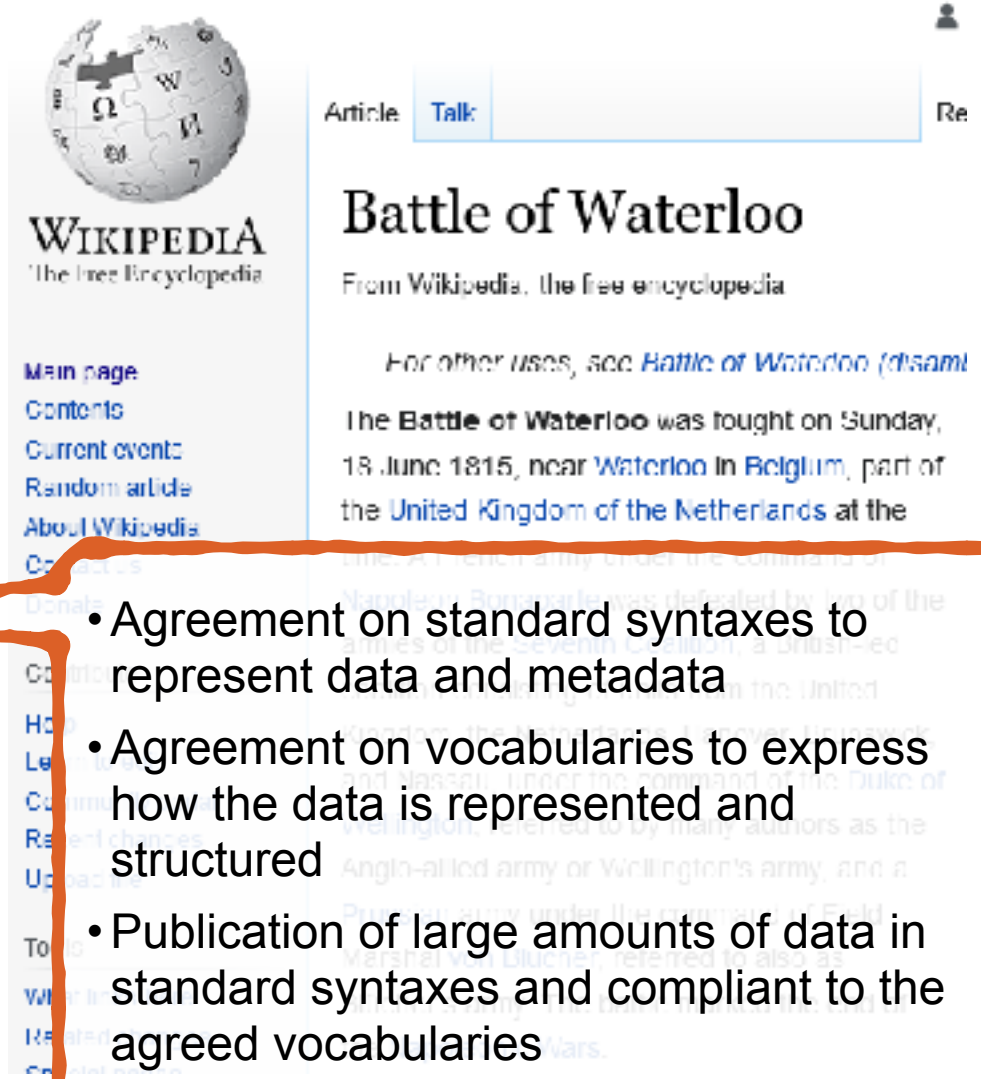
Year

Day in Week

Day

Month

Person

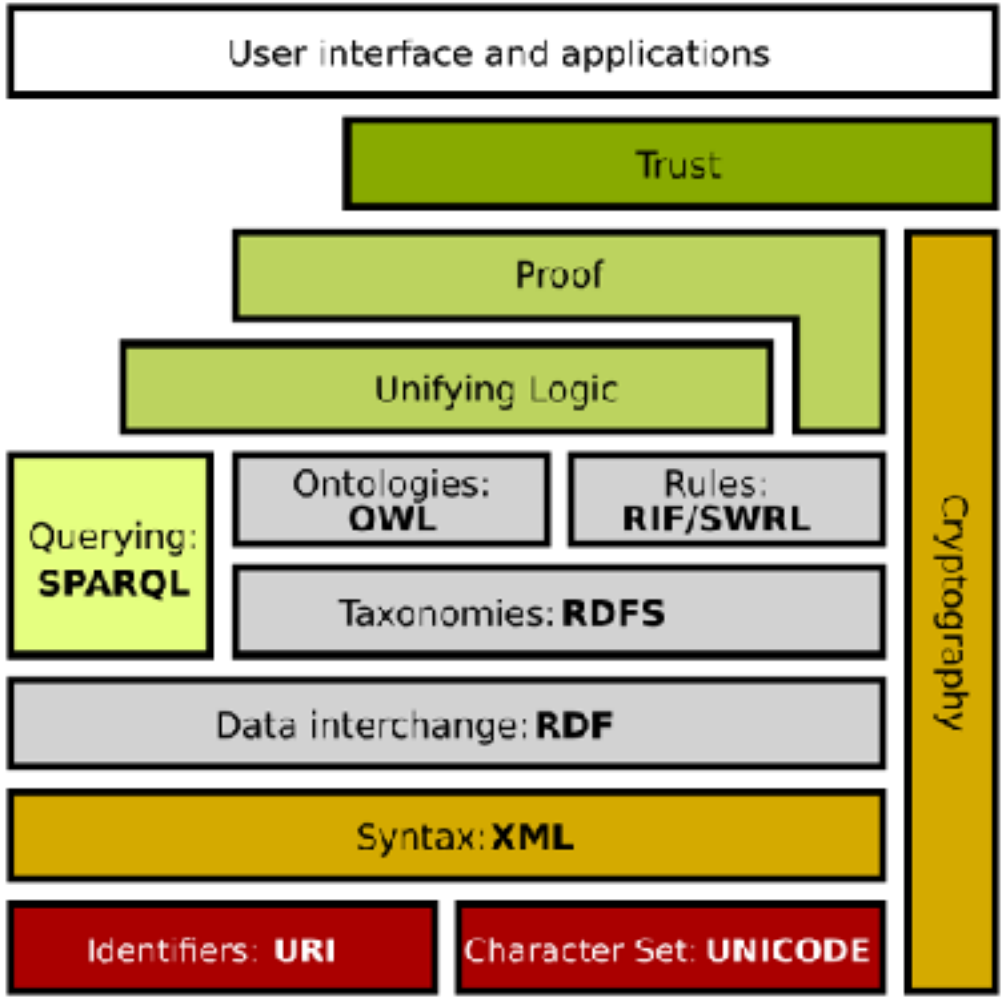


- Agreement on standard syntaxes to represent data and metadata
- Agreement on vocabularies to express how the data is represented and structured
- Publication of large amounts of data in standard syntaxes and compliant to the agreed vocabularies

Architecture of the semantic web

- The web is distributed in:
  - Content
  - Location
  - Ownership
- The semantic web inherits these core pillars

- Approach: Layer Cake of Technologies
  - Consensus on small steps
  - Incremental adoption
  - Downward compatibility: agents are able to interpret the information at lower levels
  - Upward compatibility: agents can (partially) use information at higher levels

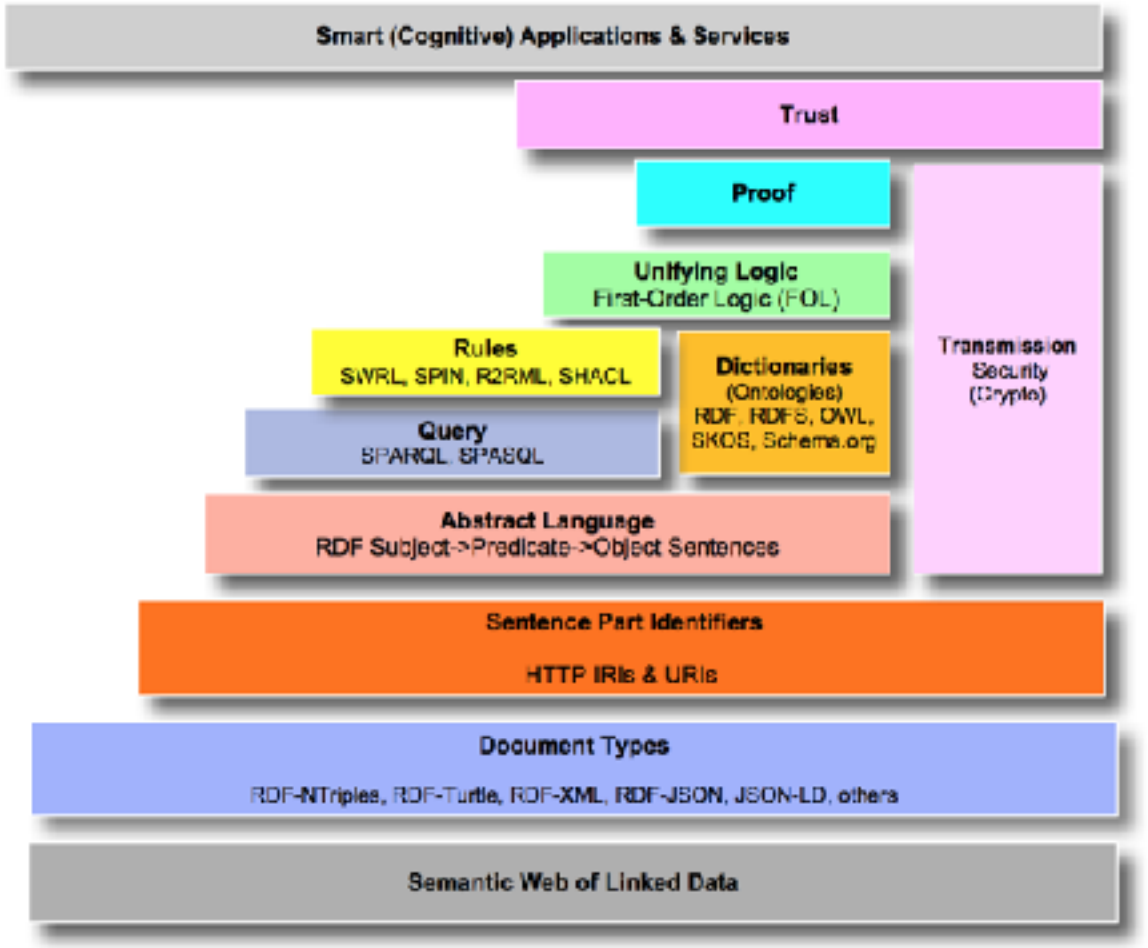


https://en.wikipedia.org/wiki/File:Semantic-web-stack.png

Architecture of the semantic web

- The web is distributed in:
  - Content
  - Location
  - Ownership
- The semantic web inherits these core pillars

- Approach: Layer Cake of Technologies
  - Consensus on small steps
  - Incremental adoption
  - Downward compatibility: agents are able to interpret the information at lower levels
  - Upward compatibility: agents can (partially) use information at higher levels

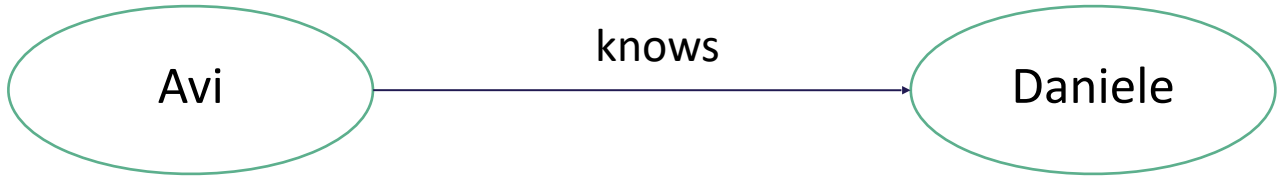


https://medium.com/openlink-software-blog/semantic-web-layer-cake-tweak-explained-6ba5c6ac3fab

Basic technologies: RDF

- Principle 1:
  - **usage of structured and semi-structured data**

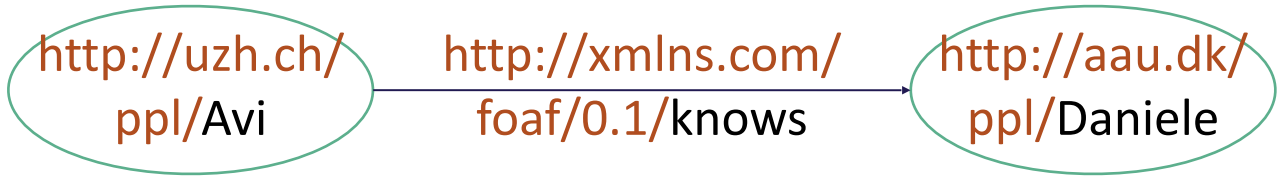
- Resource Description Framework (RDF)
  - labelled graph as data model
  - objects as nodes
  - relations as edges



Basic technologies: URI and IRI

- Principle 2:
  - **data elements as first class citizen in the web**

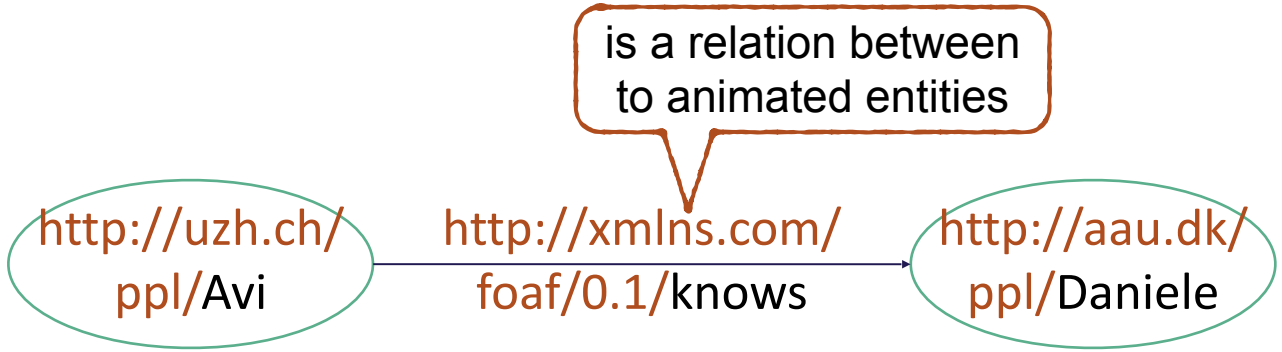
- Uniform Resource Identifiers (URI) and Internationalized Resource Identifiers (IRI)
  - Web identifier
  - RDF relies on URI/IRI
  - Used to identify data elements and relations among them



Basic technologies: RDF Schema and OWL

- Principle 3:
  - **make available the data semantic description**

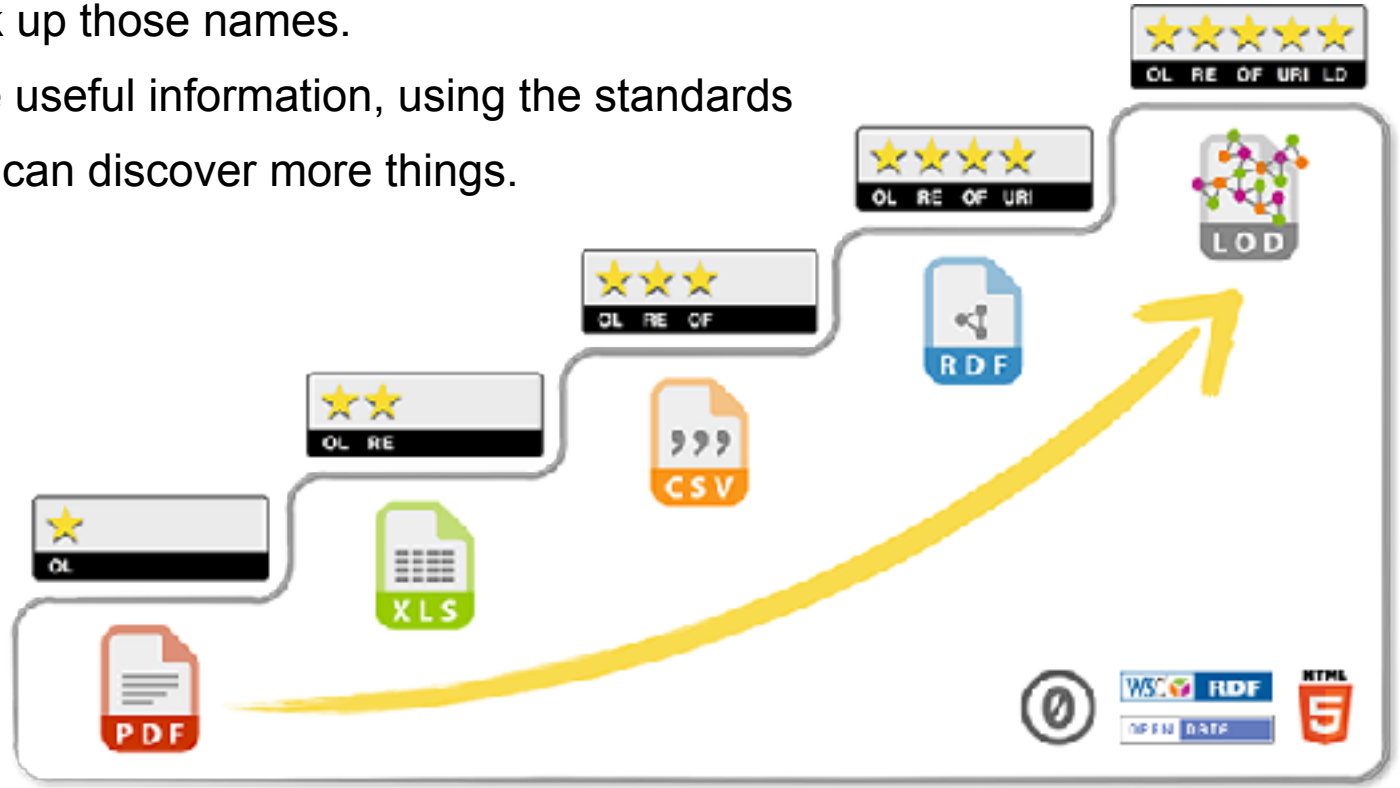
- RDF Schema and Ontology Web Language (OWL)
  - Knowledge representation languages: «logics» to enable inference of implicit information in the data
  - RDF Schema has a low expressivity
  - OWL is richer (depends on the dialect)
  - RDF Schema and OWL can be used to impose upper and lower bounds to the data semantics



Linked Data & Open Data

- **Linked data** are based on four principles:
  - Use URIs as names for things
  - Use HTTP URIs so that people can look up those names.
  - When someone looks up a URI, provide useful information, using the standards
  - Include links to other URIs. so that they can discover more things.

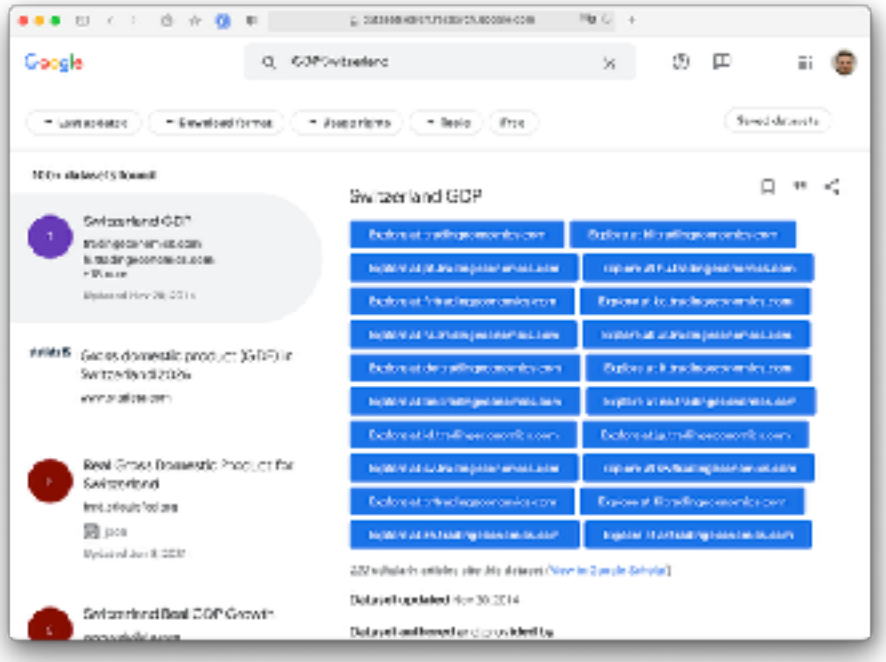
- **Open Data — Five Stars**
  - Publish your data
  - In a structured format
  - in an open format
  - In RDF
  - Linked

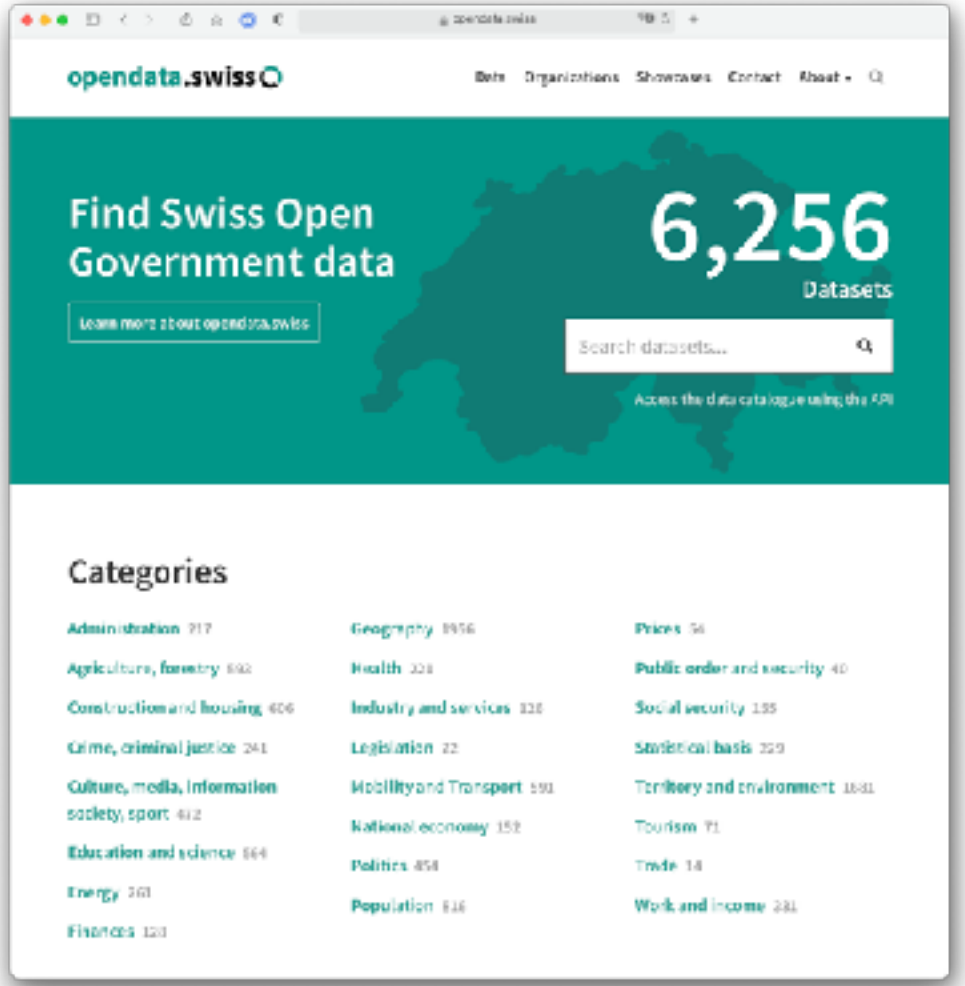


https://5stardata.info/en/

Open Data on the Web

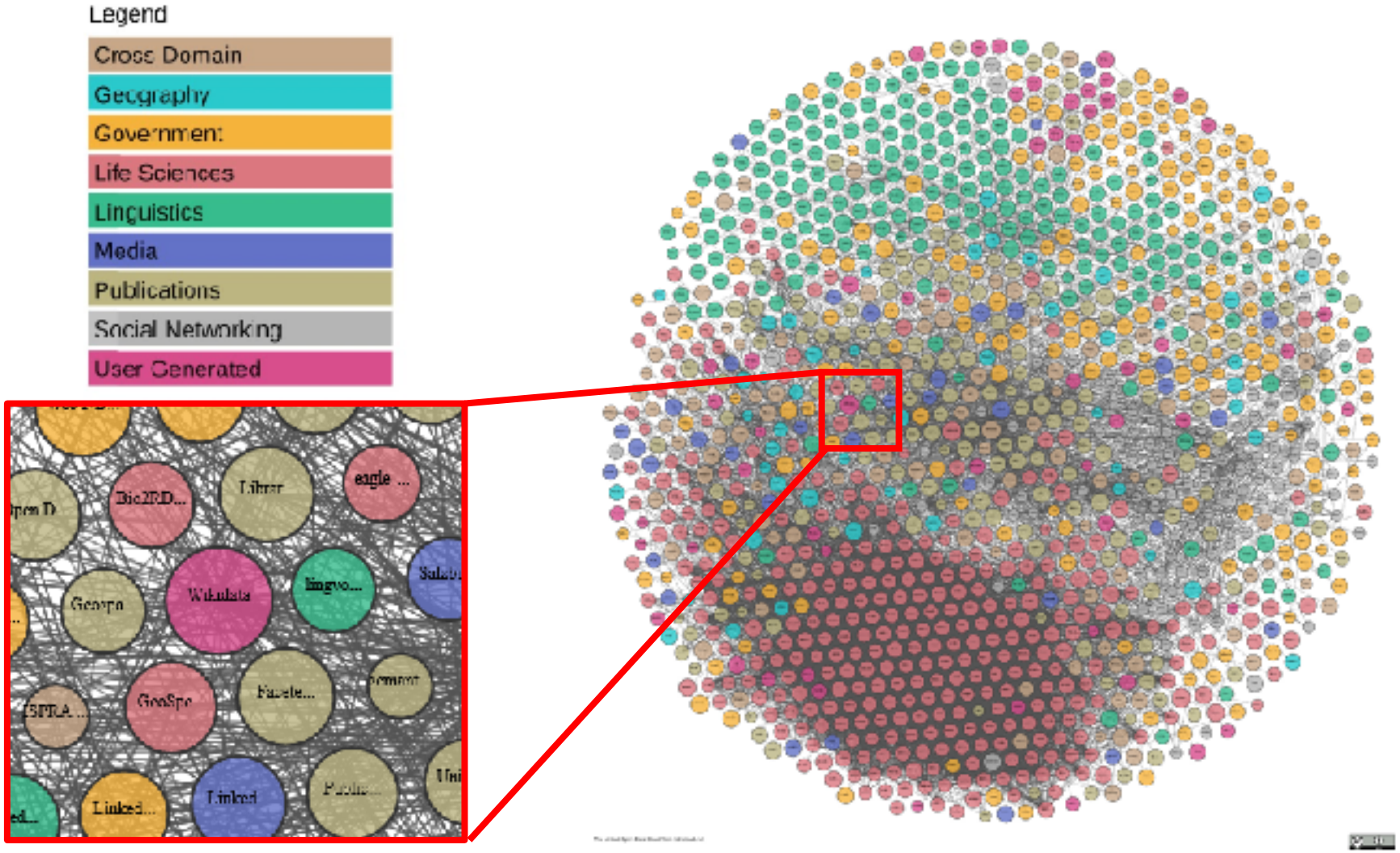
- Open data Switzerland (https://opendata.swiss/en)
- Google Dataset Search (https://datasetsearch.research.google.com)





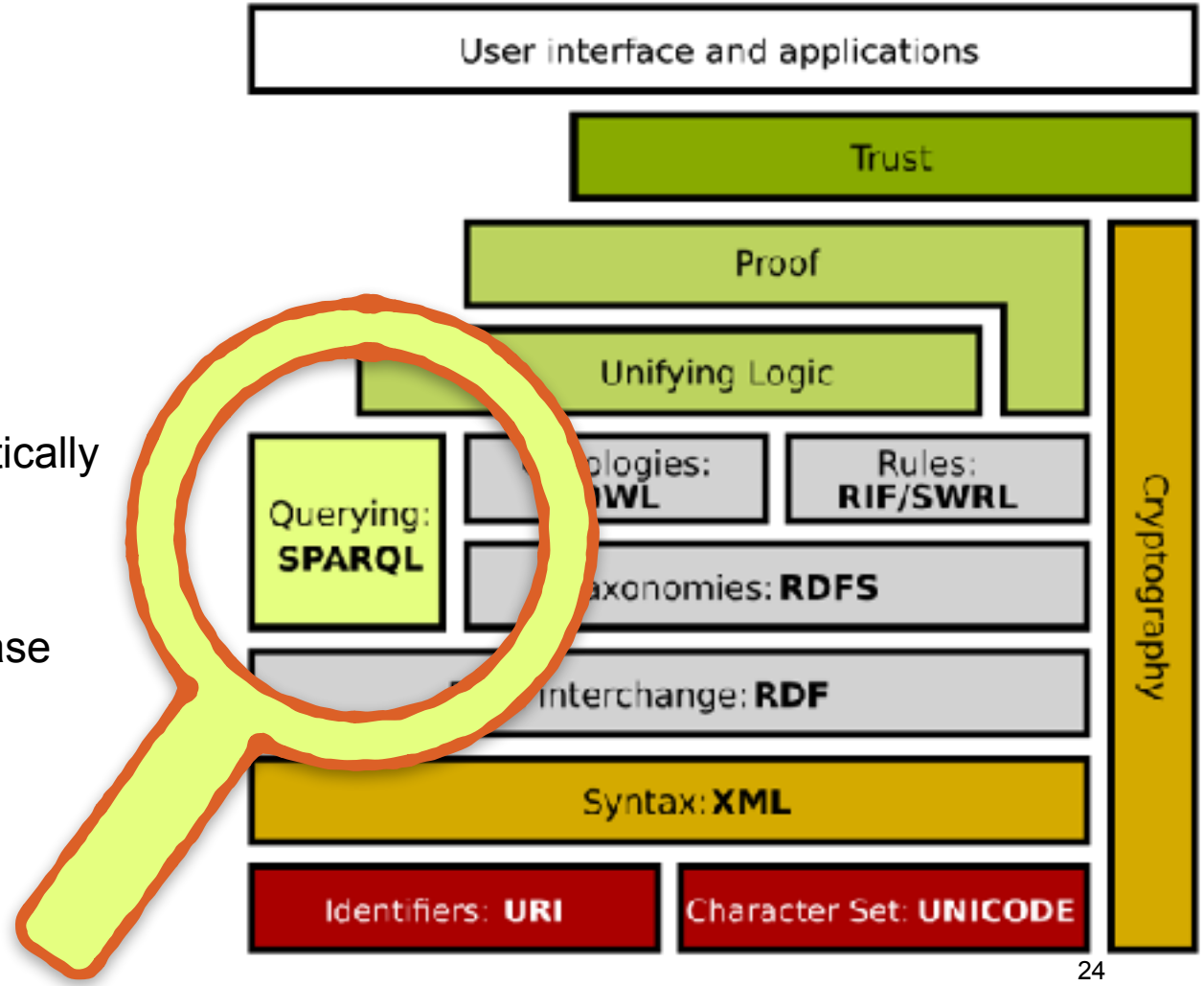
The Linked Open Data Cloud (LODC)

http://lod-cloud.net



Accessing Linked Open Data

- Dataset dump
  - Download and install it in a triple store
- Online navigation
  - Good for human inspection
  - Not so good to access the data programmatically
- Query services
  - Submit a query similar to a relational database



https://en.wikipedia.org/wiki/File:Semantic-web-stack.png

Knowledge Graphs: Basic Definitions

Knowledge Graph Definitions

- KG is a graph, where nodes represent entities, and edges represent relationships between those entities. Often a directed edge-labelled graph is assumed.
  Question: How is a knowledge graph different from a graph (database)? Where does knowledge come into play?
- A KG is a graph-structured knowledge base.
  Question: What, then is a "knowledge base"? → expression from Expert Systems with Logic
- A KG has a set of characteristics, e.g.,
  - *mainly describes real world entities and their interrelations, organized in a graph; defines possible classes and relations of entities in a schema; allows for potentially interrelating arbitrary entities with each other; covers various topical domains (*Paulheim, et al 2017)
    Requires instances, multiple domains,
- List by example (Wikidata, Google's KG, YAGO, DBPedia, etc.)

see KG Survey

Knowledge Graph Definitions — Practical Working Definition

- Noy *et al* 19:
  - a knowledge graph describes ***objects of interest*** and ***connections between them***
  - Knowledge graphs and similar structures usually provide a ***shared substrate of knowledge within an organization***, allowing different products and applications to ***use similar vocabulary and to reuse definitions and descriptions*** that others create. Furthermore, they ***usually provide a compact formal representation*** that developers can use to ***infer new facts and build up the knowledge***

see KG Survey

Knowledge Graph Definitions — Graphs

– A ***directed edge-labelled graph*** is a tuple $G := (V, E, L)$, where $\mathbb{C}$ is a set of constants, $V \subseteq \mathbb{C}$ is a set of nodes, $L \subseteq \mathbb{C}$ is a set of edge labels, and $E \subseteq V \times L \times V$ is a set of edges.

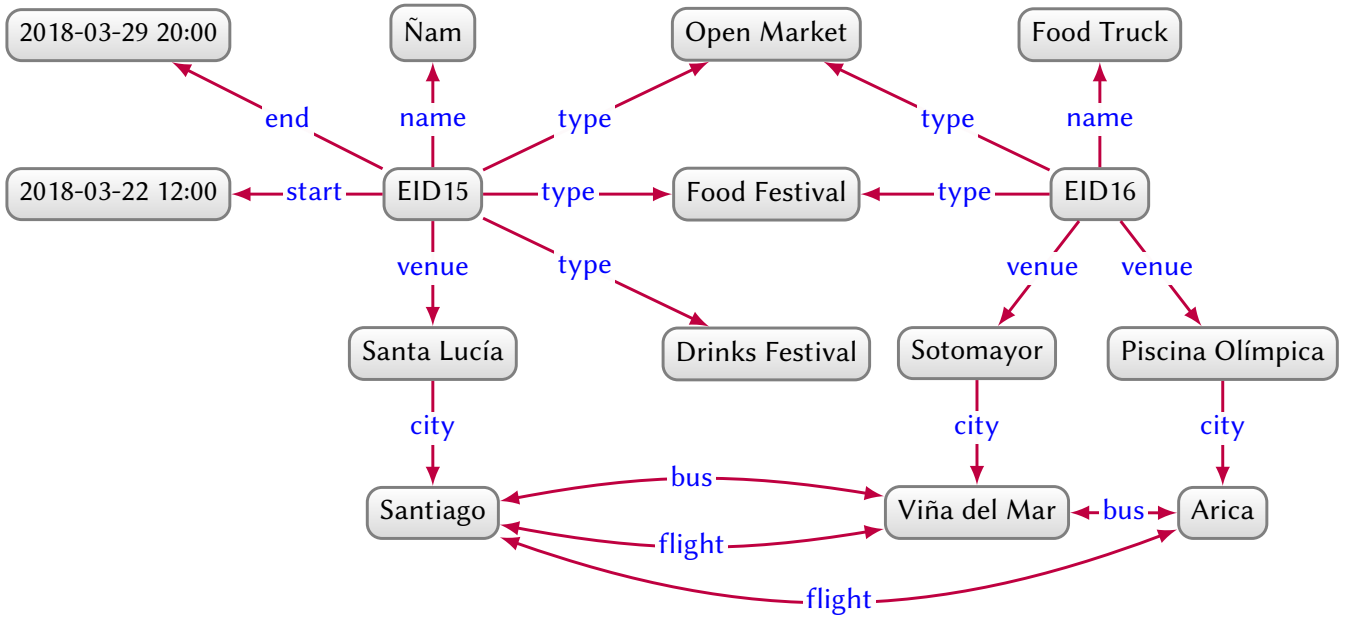
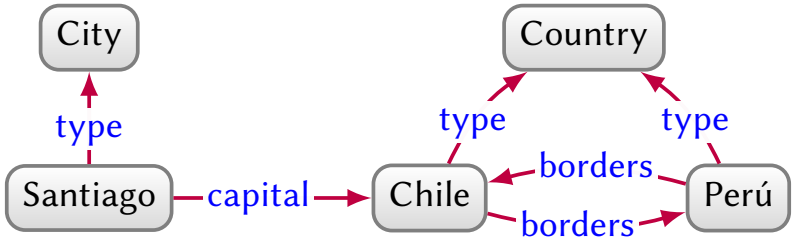


Fig. 1. Directed edge-labelled graph describing events and their venues.

see KG Survey

Knowledge Graph Definitions — Graphs

– A ***heterogeneous graph*** is a tuple $G := (V, E, L, l)$, where $\mathbb{C}$ is a set of constants, $V \subseteq \mathbb{C}$ is a set of nodes, $L \subseteq \mathbb{C}$ is a set of edge labels, $E \subseteq V \times L \times V$ is a set of edges, and $l : V \cup L$ maps each node to a label.



(a) Del graph

(b) Heterogeneous graph

see KG Survey

Knowledge Graph Definitions — Graphs

– A ***property graph*** is a tuple
$G := (V, E, L, P, U, e, l, p)$,
where where $\mathbb{C}$ is a set of constants, $V \subseteq \mathbb{C}$ is a set of node ids, $E \subseteq \mathbb{C}$ is a set of edge ids, $L \subseteq \mathbb{C}$ is a set of labels, $P \subseteq \mathbb{C}$ is a set of properties, $U \subseteq \mathbb{C}$ is a set of values, $e : E \to V \times V$ maps an edge id to a pair of node ids, $l : V \cup E \to 2^L$ maps a node or edge id to a set of labels, and $p : V \cup E \to 2^{P \times U}$ maps a node or edge id to a set of property-value pairs.

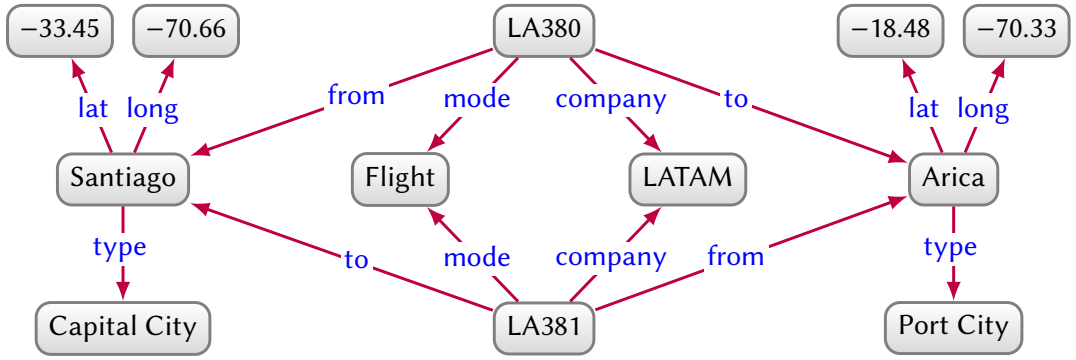


Fig. 3. Directed edge-labelled graph with companies offering flights between Santiago and Arica

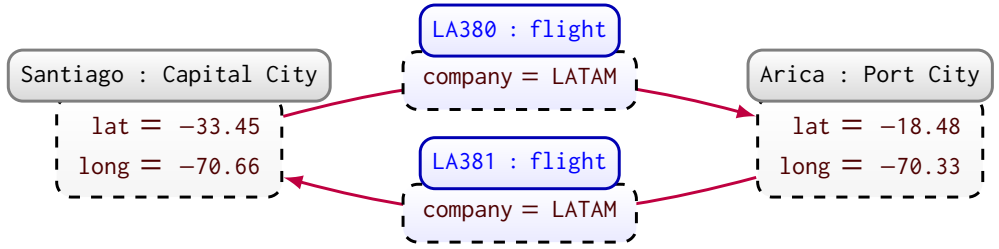


Fig. 4. Property graph with companies offering flights between Santiago and Arica

see KG Survey

Knowledge Graph Definitions — Grap

– A ***property graph*** is a tuple
$G := (V, E, L, P, U, e, l, p)$,
where where $\mathbb{C}$ is a set of constants, $V \subseteq \mathbb{C}$ is a set of node ids, $E \subseteq \mathbb{C}$ is a set of edge ids, $L \subseteq \mathbb{C}$ is a set of labels, $P \subseteq \mathbb{C}$ is a set of properties, $U \subseteq \mathbb{C}$ is a set of values, $e : E \to V \times V$ maps an edge id to a pair of node ids, $l : V \cup E \to 2^L$ maps a node or edge id to a set of labels, and $p : V \cup E \to 2^{P \times U}$ maps a node or edge id to a set of property-value pairs.

V contains `Santiago` and `Arica`
E contains `LA380` and `LA381`
L contains `Capital City`, `Port City`, and `flight`
P contains `lat`, long, and `company`
U contains `-33.45`, `-70.66`, `LATAM`, `-18.48`, and `-70.33`
e gives, for example, e(LA380) = (Santiago, Arica)
l gives, for example, l(LA380) = {flight}and
l(Santiago) = {Capital City}
p gives, for example,
p(Santiago) = {(lat, -33.45), (long, -70.66)}and
p(LA380) = {(company, LATAM)}

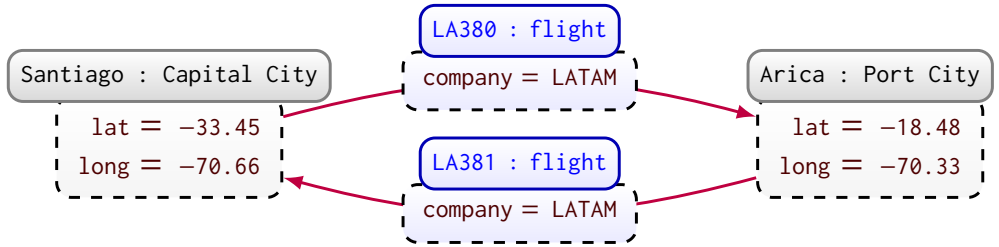


Fig. 4. Property graph with companies offering flights between Santiago and Arica

see KG Survey

Knowledge Graph Definitions

- A **named graph** is a pair (n, G) where G is a graph, and $n \in \mathbb{C}$ is a graph name.
- A **graph dataset** is a pair $D := (G_D, N)$, where G_D is a directed edge-labelled graph called the ***default graph*** and N is either the empty set, or a set of named graphs $\{(n_1, G_1), \ldots (n_k, G_k)\}$ for $k > 0$ such that $n_i = n_j$ if and only if $i = j$ $(1 \leq i \leq k, 1 \leq j \leq k)$.

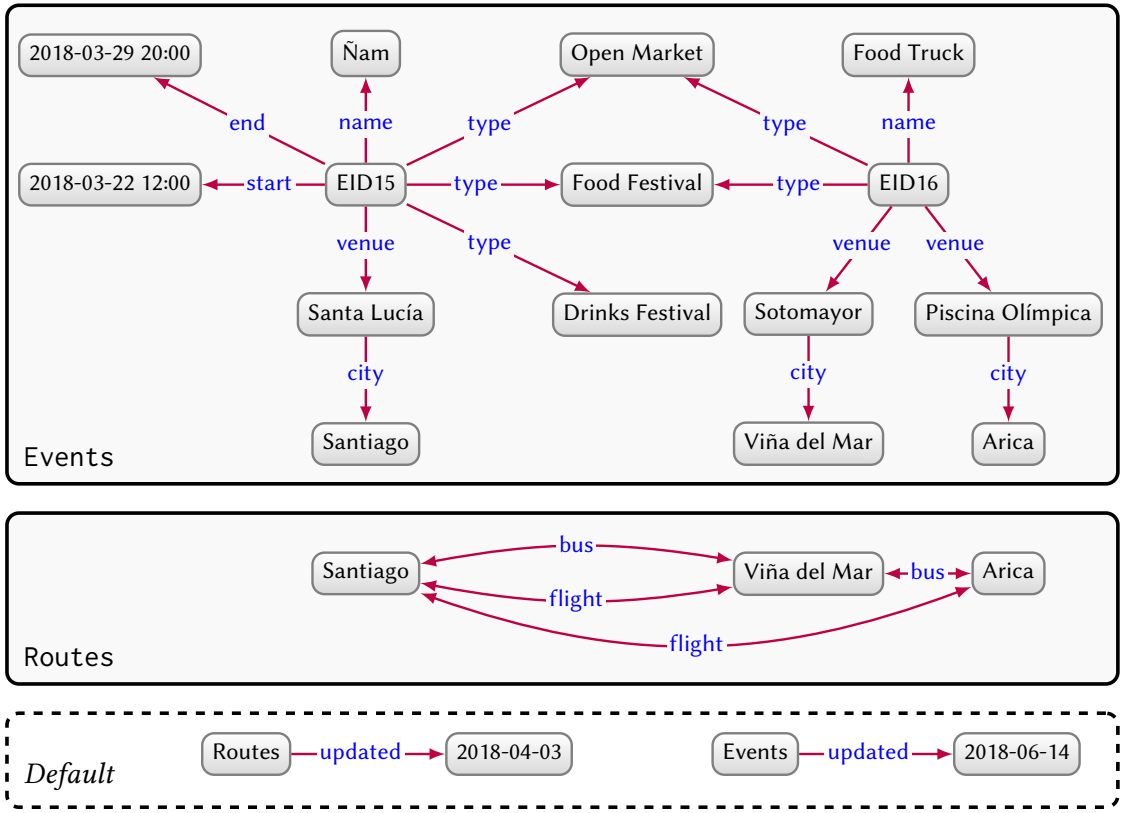
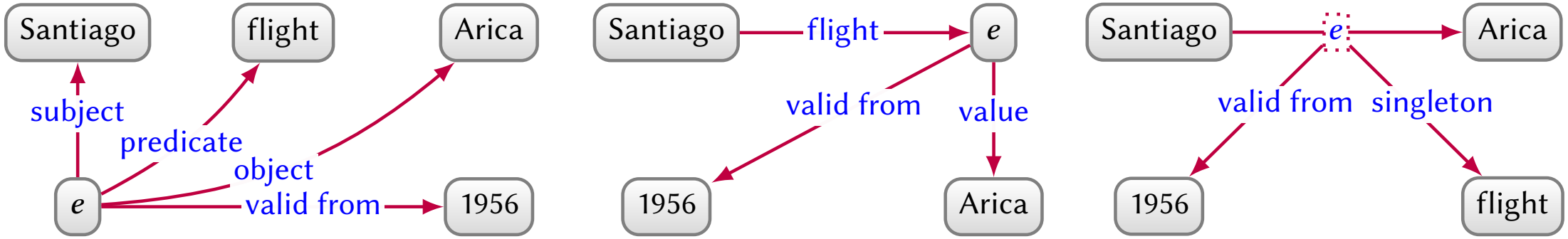


Fig. 5. Graph dataset with two named graphs and a default graph describing events and routes

see KG Survey

Knowledge Graph Definitions — Representational Issues

– Let take the triple (`Santiago, flight, Arica`), what if I want to say more about it?



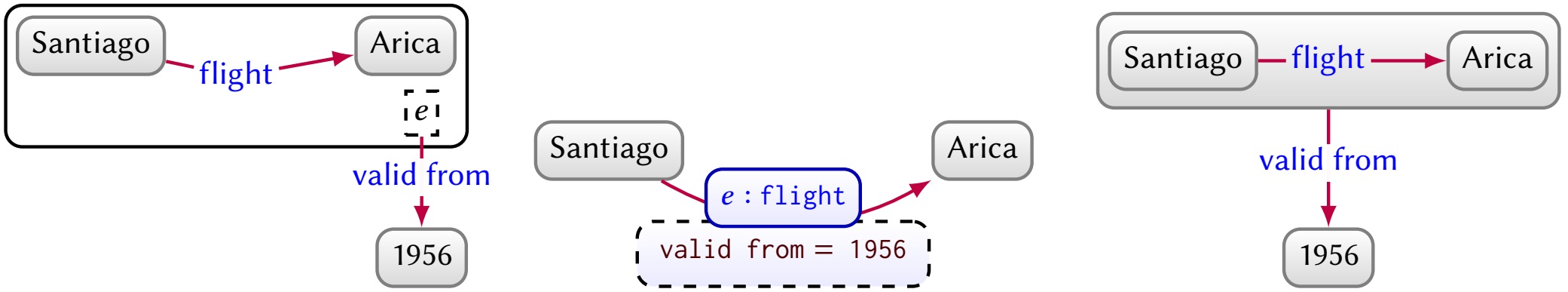
(a) RDF Reification (b) *n*-ary Relations (c) Singleton properties

Fig. 18. Three representations of temporal context on an edge in a directed-edge labelled graph

see KG Survey

Knowledge Graph Definitions — Time

– Let take the triple (`Santiago, flight, Arica`), what if I want to say more about it?



(a) Named graph (b) Property graph (c) RDF*

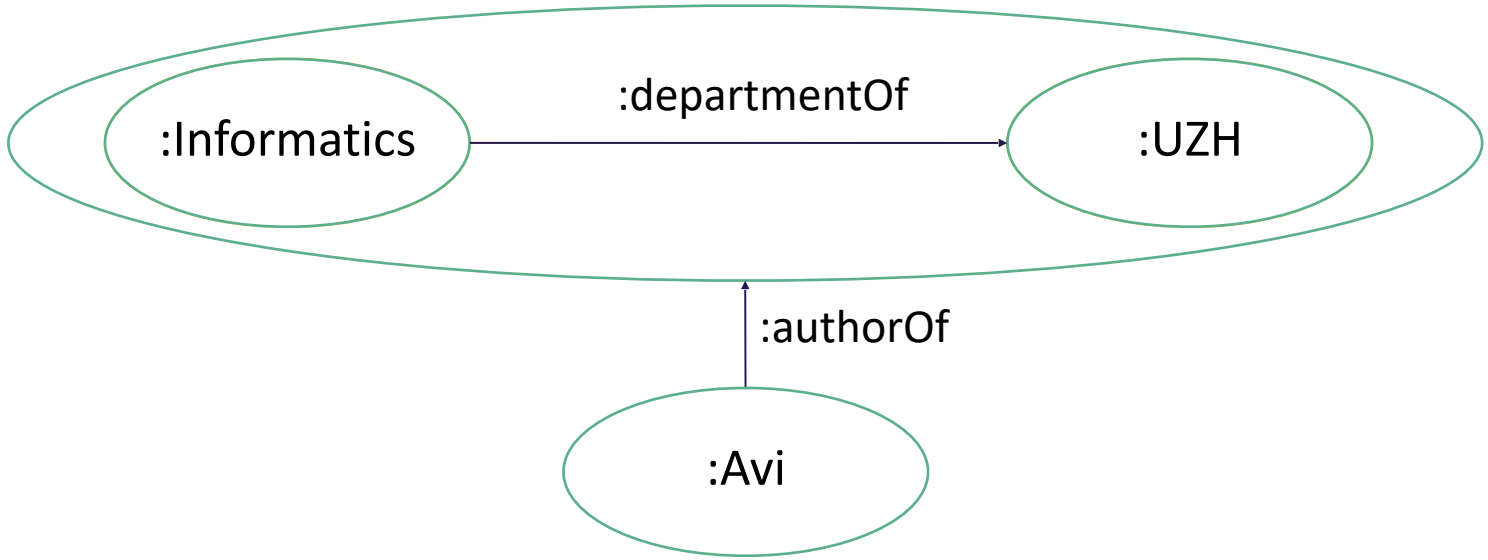
Fig. 19. Three higher-arity representations of temporal context on an edge

see KG Survey

Knowledge Graphs: Basic Definitions — Practical Considerations

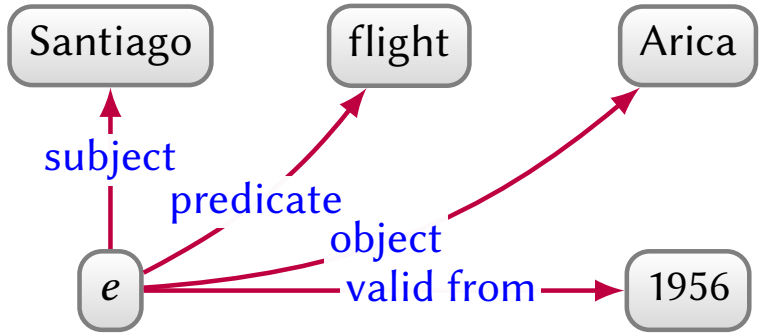
Pointing to Statements

- Two ways to express this in RDF:
  - Reification
  - Named graphs
  - RDF*



Pointing to Statements: Reification

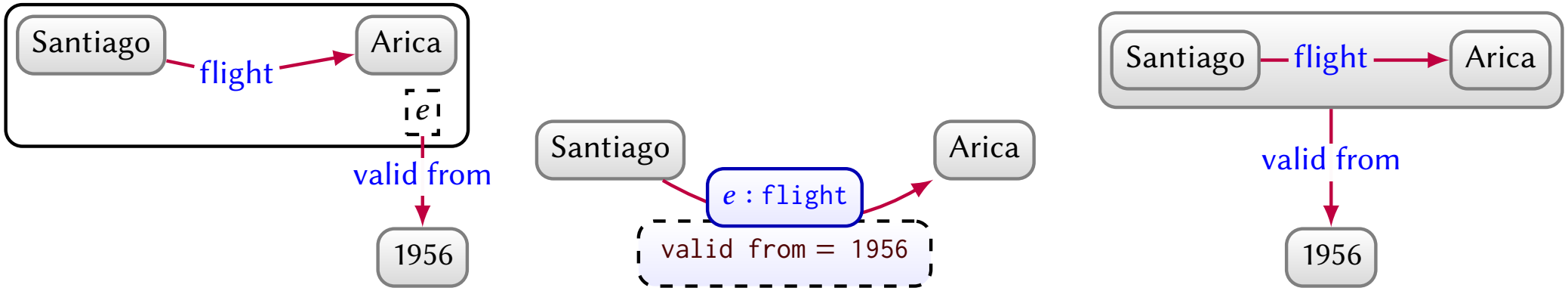
- **Approach**
  - Introduce an auxiliary resource
  - Relate the auxiliary resource to the three elements of the statement
  - It is now possible to predicate on the statement
  - RDF has three special properties to manage reification:
    - http://www.w3.org/1999/02/22-rdf-syntax-ns#subject
    - http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate
    - http://www.w3.org/1999/02/22-rdf-syntax-ns#object
- This technique preserves the RDF model
- Identifying statements would require quadruples rather than triples
- A large overhead is introduced
- One new resource and three new statements for every statement to be reified!



(a) RDF Reification

Pointing to Statements: Named Graph & Non-RDF Approaches

- **Approach**
  - Introduced in the newer versions of RDF
  - An explicit identifier can be assigned to a set of statements
    - The new identifier can be used to create new statements

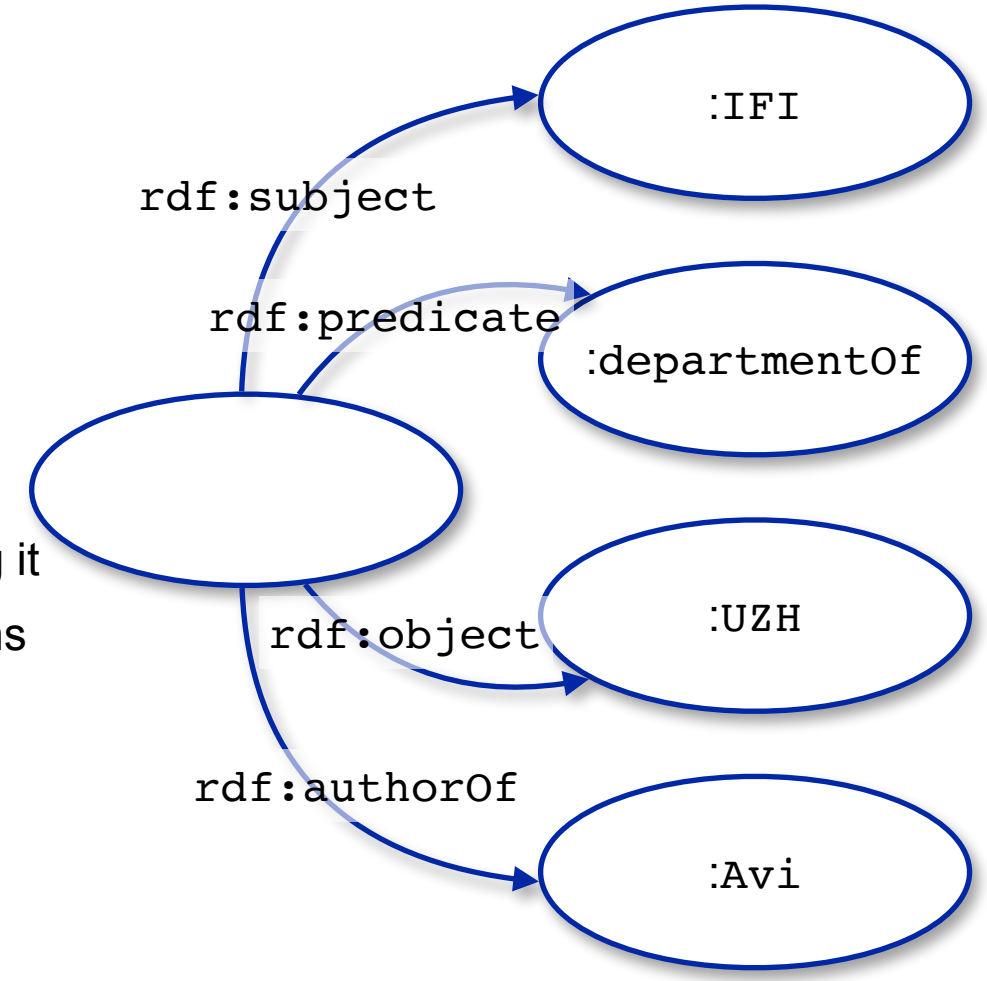


(a) Named graph (b) Property graph (c) RDF*

Fig. 19. Three higher-arity representations of temporal context on an edge

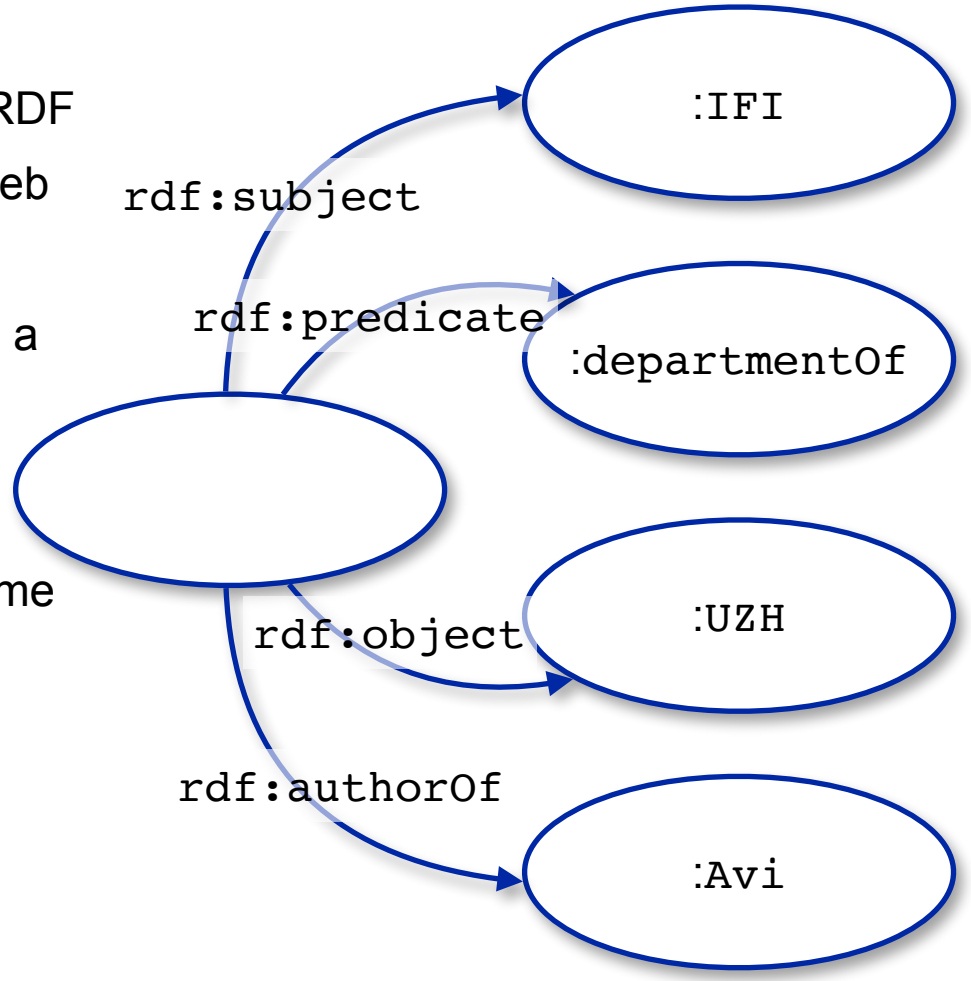
Blank Nodes

- There are cases where the a node of the graph is unknown
- Need of auxiliary nodes
  - Reification
  - Richer predicates
  - Complex attributes
- Lack of information (voluntary or involuntary)
  - Need of assessing the existence of something without naming it
- RDF offers blank nodes (shortly `bnodes`) to tackle these situations
  - Can be only used as subject or object



Blank Nodes II

- Blank nodes are one of the most discussed and debated parts of RDF
- Useful to cope with situations of partial knowledge typical of the Web
- The trickiest part about RDF semantics is related to blank nodes
  - While URIs and literals have a global scope, blank nodes have a scope limited in the context of the graph containing them
  - If two graphs contains a blank node _:a, it does not mean that it represent the same thing!
- It is necessary to determine whenever two blank nodes are the same or not
  - It is a graph isomorphism problem
  - Theoretically a NP problem
  - In practice it can be often treated in the context of RDF



Knowledge Graphs: Basic Definitions — Syntax Issues

Syntax

- RDF comes in many formats (apart from the graphical one)
  - Turtle (and TriG)
  - RDF/XML
  - RDFa
  - JSON-LD

Syntax: Turtle (https://www.w3.org/TR/turtle/)

- Terse RDF Triple Language (Turtle)
  - Text-based syntax for RDF
  - The usual extension of Turtle files is .ttl
- The syntax follows the statement structure:
  - Subject, predicate and object followed by a period:
- URIs are enclosed in < and >
  - <http://uzh.ch/ppl/Avi> <http://xmlns.com/foaf/0.1/knows> <http://uzh.ch/ppl/Elaine> .
  - <http://uzh.ch/ppl/Avi> <http://example.org#worksAt> <http://uzh.ch#lfl> .

Syntax: Turtle — Namespaces & Prefixes

- Turtle allows taking advantage of prefixes
- Use of @prefix to declare the mapping between prefixes and namespaces

<http://uzh.ch/ppl/Avi> <http://xmlns.com/foaf/0.1/knows> <http://uzh.ch/ppl/Elaine> .
<http://uzh.ch/ppl/Avi> <http://example.org#worksAt> <http://uzh.ch#lfl> .

- We can introduce prefixes to move to a more compact representation

@prefix uzh: <http://uzh.ch/ppl/> .
@prefix ex: <http://example.org#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

uzh:Avi foaf: knows uzh: Elaine .
uzh:Avi ex:worksAt <http://uzh.ch#lfl> .

uzh:Avi foaf: knows uzh: Elaine ;
ex:worksAt <http://uzh.ch#lfl> .

uzh:Avi foaf: knows uzh: Elaine .
uzh:Avi foaf: knows uzh: Gerhard .

uzh:Avi foaf: knows uzh: Elaine , uzh: Gerhard .

Syntax: Turtle — Literals and Blank Nodes

- Literals are (usually) represented with double quotes
  - “Friedrich Dürrenmatt”
- When datatypes are involved, they follow the literal value and are separated through ^^
  - “71”^^<http://www.w3.org/2001/XMLSchema#integer>
  - “1.2”^^<http://www.w3.org/2001/XMLSchema#float>
  - “2018-09-28”^^<http://www.w3.org/2001/XMLSchema#date>
- Abbreviations are possible for numbers:
  - 71 (without quotes!) is interpreted as an integer
  - 1.2 is interpreted as a decimal number
- Blank nodes are represented as resources with _ as prefix (e.g. _:a), or as [ ]

RDFa

```
<html xmlns:aau="http://uzh.ch/ppl/" xmlns:ex="http://example.org#" xmlns:foaf="http://xmlns.com/foaf/0.1/">
<body>
<h1>Avi' homepage</h1>
<p about="uzh:Avi">Matteo is a researcher at the <a href="http://ifi.uzh.ch/" rel="ex:worksAt">Department of
Informatics</a> of the University of Zurich.</p>
<h2 about="uzh:Avi">Collaborators</h2>
<ul>
 <li rel="foaf:knows" resource="uzh:Elaine">Elaine</li>
 <li rel="foaf:knows" resource="uzh:Gerhard">Gerhard</li>
</ul>
</body>
</html>
```

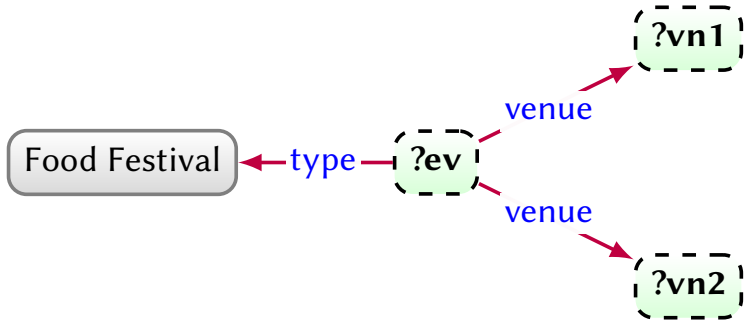
```
@prefix uzh: <http://uzh.ch/ppl/> .
@prefix ex: <http://example.org#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

uzh:Avi foaf:knows aau:Elaine .
uzh:Avi foaf:knows aau:Gerhard .
uzh:Avi ex:worksAt <http://ifi.uzh.ch/> .
```

Querying KGs

Querying KGs — Definitions

- Given *variables* $\mathbf{Var}$, where $\mathbb{C} \cap \mathbf{Var} = \emptyset$ and $\mathbf{Term} = \mathbb{C} \cup \mathbf{Var}$
- A **directed edge-labelled graph pattern** as a tuple $Q = (V', E', L')$, where $V' \subseteq \mathbf{Term}$ is a set of node terms, $L' \subseteq \mathbf{Term}$ is a set of edge terms, and $E' \subseteq V' \times L' \times V'$ is a set of edges (triple patterns)
- Or as algebra operations ($\pi, \sigma, \rho, \bowtie$) assuming a single ternary relation $G(s, p, o)$ representing a graph

$$\pi_{ev,vn1,vn2}(\sigma_{p=\text{type} \wedge o=\text{Food Festival} \wedge p_1=p_2=\text{venue}}(\rho_{s/ev}(G \bowtie \rho_{p/p_1,o/vn1}(G) \bowtie \rho_{p/p_2,o/vn2}(G))))$$



?ev	?vn1	?vn2
EID16	Piscina Olímpica	Sotomayor
EID16	Sotomayor	Piscina Olímpica
EID16	Piscina Olímpica	Piscina Olímpica
EID16	Sotomayor	Sotomayor
EID15	Santa Lucía	Santa Lucía

Fig. 6. Graph pattern (left) with mappings generated over the graph of Figure 1 (right)

see KG Survey

Querying KGs — Definitions

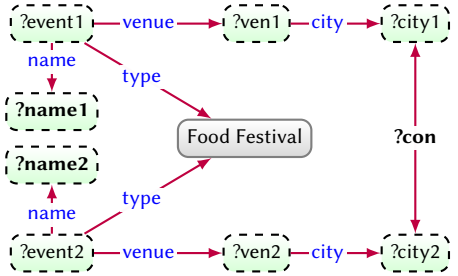
- Given *variables* $\mathbf{Var}$, where $\mathbb{C} \cap \mathbf{Var} = \varnothing$ and $\mathbf{Term} = \mathbb{C} \cup \mathbf{Var}$
- A **property graph pattern** as a tuple $Q = (V', E', L', P', U', e', l', p')$, where $V' \subseteq \mathbf{Term}$ is a set of node id terms, $E' \subseteq \mathbf{Term}$ is a set of edge id terms, $L' \subseteq \mathbf{Term}$ is a set of label terms, $P' \subseteq \mathbf{Term}$ is a set of property terms, $U' \subseteq \mathbf{Term}$is a set of value terms, $e' : E' \rightarrow V' \times V'$ maps an edge id term to a pair of node id terms, $l' : V' \cup E' \rightarrow 2^{L'}$maps a node or edge id term to a set of label terms, and $p' : V' \cup E' \rightarrow 2^{P' \times U'}$ maps a node or edge id term to a set of pairs of property-value terms.

see KG Survey

Querying KGs — Definitions

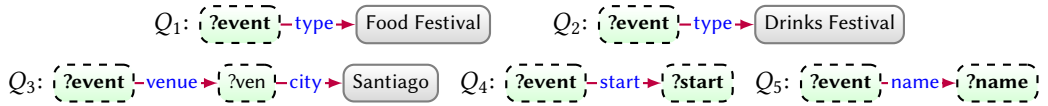
Complex graph patterns are defined recursively:

- If Q is a graph pattern, then Q is a *complex graph pattern*.
- If Q is a complex graph pattern, and $\mathcal{V} \subseteq Var(Q)$, then $\pi_{\mathcal{V}}(Q)$ is a *complex graph pattern*.
- If Q is a complex graph pattern, and $R$is a selection condition with boolean and equality connectives ($\wedge$, $\vee$, $\neg$, $=$) , then $\sigma_R(Q)$ is a *complex graph pattern*.
- If Q_1 and Q_2 are complex graph patterns, then $Q_1 \bowtie Q_2$, $Q_1 \cup Q_2$ and $Q_1 - Q_2$ are also *complex graph pattern*.



?name1	?con	?name2
Food Truck	bus	Food Truck
Food Truck	bus	Food Truck
Food Truck	bus	Ñam
Food Truck	flight	Ñam
Food Truck	flight	Ñam
Ñam	bus	Food Truck
Ñam	flight	Food Truck
Ñam	flight	Food Truck

Fig. 7. Conjunctive query (left) with mappings generated over the graph of Figure 1 (right)



$Q := ((((Q_1 \cup Q_2) \triangleright Q_3) \bowtie Q_4) \bowtie Q_5)$, $Q(G) =$

?event	?start	?name
EID16		Food Truck

Fig. 8. Complex graph pattern (Q) with mappings generated ($Q(G)$) over the graph of Figure 1 (G)

see KG Survey

Querying KGs — Definitions

Complex graph patterns are defined recursively:

Definition B.15 (Complex graph pattern evaluation). Given a complex graph pattern Q, if Q is a graph pattern, then $Q(G)$ is defined per Definition B.12. Otherwise:

$$\pi_{\mathcal{V}}(Q)(G) := \{\mu[\mathcal{V}] \mid \mu \in Q(G)\}$$
$$\sigma_R(Q)(G) := \{\mu \mid \mu \in Q(G) \text{ and } R \models \mu\}$$
$$Q_1 \bowtie Q_2(G) := \{\mu_1 \cup \mu_2 \mid \mu_1 \in Q_2(G) \text{ and } \mu_2 \in Q_1(G) \text{ and } \mu_1 \sim \mu_2\}$$
$$Q_1 \cup Q_2(G) := \{\mu \mid \mu \in Q_1(G) \text{ or } \mu \in Q_2(G)\}$$
$$Q_1 - Q_2(G) := \{\mu \mid \mu \in Q_1(G) \text{ and } \mu \notin Q_2(G)\}$$

Based on these query operators, we can also define some additional syntactic operators, such as the *anti-join* ($\triangleright$, aka *not exists*), or the *left-join* ($\bowtie\!\!\!\!\!\!-$, aka *optional*):

$$Q_1 \triangleright Q_2(G) := Q_1(G) - \pi_{\mathbf{Var}(Q_1)}(Q_1(G) \bowtie Q_2(G))$$
$$Q_1 \bowtie\!\!\!\!\!\!- Q_2(G) := (Q_1(G) \bowtie Q_2(G)) \cup (Q_1(G) \triangleright Q_2(G))$$

see KG Survey

Querying KGs — Path Queries — Definitions

- **Path expression** r:
  - A constant (edge label) c is a *path expression*
  - r is a path expression, then r^- (*inverse*) and r^* (*Kleene star*) are *path expressions*.
  - If r_1 and r_2 are path expressions, then $r_1 \cdot r_2$ (*concatenation*) and $r_1 \mid r_2$ (*disjunction*) are *path expressions*.

Given $G = (V, E, L)$ and path expression r, the ***evaluation of*** r ***over*** G, denoted $r[G]$, is:

- $r[G] := \{(u, v) \mid (u, r, v) \in E\}$ (for $r \in \mathbb{C}$)
- $r^-[G] := \{(u, v) \mid (v, u) \in r[G]\}$
- $r_1 \mid r_2[G] := r_1[G] \cup r_2[G]$
- $r_1 \cdot r_2[G] := \{(u, v) \mid \exists w \in V : (u, w) \in r_1[G] \text{ and } (w, v) \in r_2[G]\}$
- $r^*[G] := V \cup \bigcup_{n \in \mathbb{N}^+} r^n[G]$

where by r^n we denote the n^{th} concatenation of r (e.g., $r^3 = r \cdot r \cdot r$).

see KG Survey

Querying KGs — Path Queries — Definitions

- A **regular path query** is a triple (x, r, y) where $x, y \in \mathbb{C} \cup \mathbf{Var}$ and r is a path expression.

- **Navigational graph pattern** is
  - a graph pattern Q
  - $Q \bowtie (x, r, y)$,
    where Q is a navigational graph pattern and (x, r, y) is a regular path query

Let G denote a directed edge-labelled graph, $c, c_1, c_2 \in \mathbb{C}$ denote constants and $z, z_1, z_2 \in \mathbf{Var}$ denote variables. Then the **evaluation of a regular path query** is defined as follows:

- $(c_1, r, c_2)(G) := \{\mu_\varnothing \mid (c_1, c_2) \in r[G]\}$
- $(c, r, z)(G) := \{\mu \mid dom(\mu) = \{z\}$ and $(c, \mu(z)) \in r[G]\}$
- $(z, r, c)(G) := \{\mu \mid dom(\mu) = \{z\}$ and $(\mu(z), c) \in r[G]\}$
- $(z_1, r, z_2)(G) := \{\mu \mid dom(\mu) = \{z_1, z_2\}$ and $(\mu(z_1), \mu(z_2)) \in r[G]\}$

where $\mu_\varnothing$ denotes the empty mapping such that $dom(\mu) = \varnothing$(the join identity).

see KG Survey

Querying KGs — Path Queries — Definitions

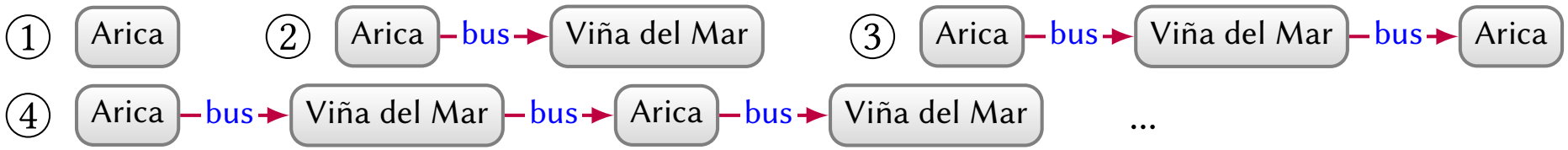
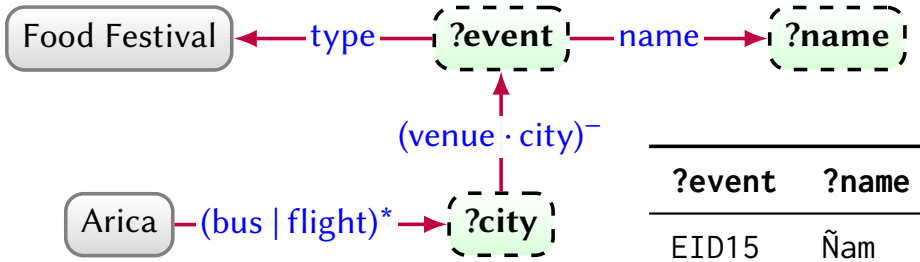
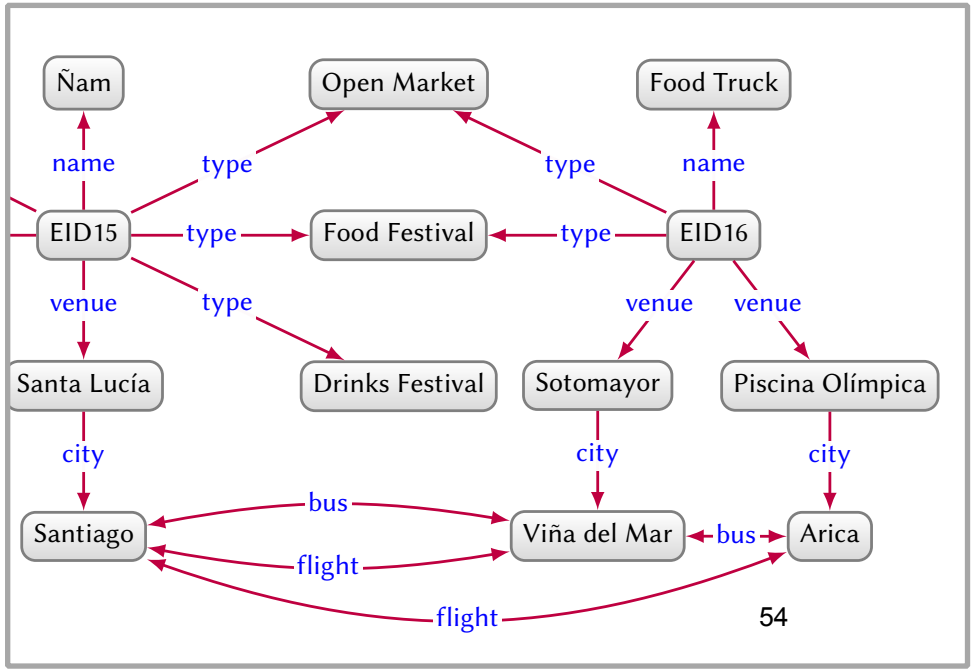


Fig. 9. Some possible paths matching (Arica, bus*, ?city) over the graph of Figure 1



?event	?name	?city
EID15	Ñam	Santiago
EID16	Food Truck	Arica
EID16	Food Truck	Viña del Mar



see KG Survey

Querying KGs — Example for Querying a Temporally Annotated Graph

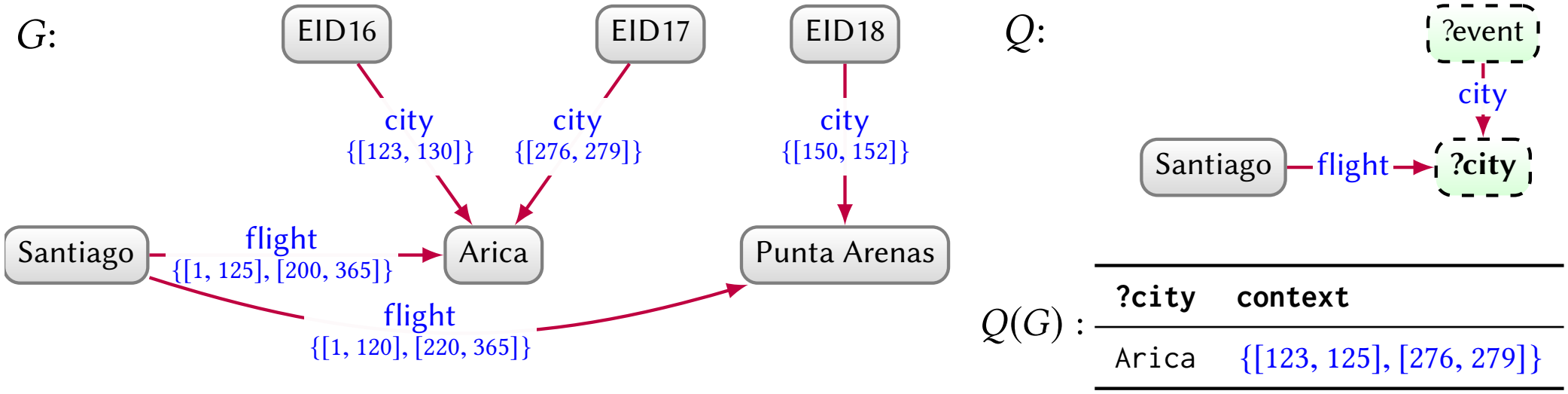


Fig. 20. Example query on a temporally annotated graph

see KG Survey

Querying KGs — Practical Considerations: SPARQL

SPARQL: The Linked Data Query Language
(acronym for: **S**PARQL **P**rotocol **A**nd **R**DF **Q**uery **L**anguage)

- SQL-like language design to query graphs
  - Join is the most important operator
  - Works in distributed environments
- A protocol for submitting queries and retrieving answers through the Web
- SPARQL queries are usually submitted to triple stores (or graph store)
  - A triple store is a database for RDF data that offers data retrieval and data manipulation operations
- Triple stores provide an interface, namely endpoint, where SPARQL queries can be submitted
  - e.g., https://dbpedia.org/sparql?query=select+?x+where+%7B?x+%3Chttp://dbpedia.org/ontology/country%3E+%3Chttp://dbpedia.org/resource/Switzerland%3E%7D+LIMIT+100 is a SPARQL query

SPARQL — Examples

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name (COUNT(?friend) AS ?count)
WHERE  {
       ?person foaf:name ?name .
       ?person foaf:knows ?friend .
} GROUP BY ?person ?name
```

?name	?count
“Alice”	3
“Bob”	1
“Charlie”	1

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://example.org/alice#me>   a          foaf:Person .
<http://example.org/alice#me>   foaf:name  "Alice" .
<http://example.org/alice#me>   foaf:mbox  <mailto:alice@example.org> .
<http://example.org/alice#me>   foaf:knows <http://example.org/bob#me> .
<http://example.org/bob#me>     foaf:knows <http://example.org/alice#me> .
<http://example.org/bob#me>     foaf:name  "Bob" .
<http://example.org/alice#me>   foaf:knows <http://example.org/charlie#me> .
<http://example.org/charlie#me> foaf:knows <http://example.org/alice#me> .
<http://example.org/charlie#me> foaf:name  "Charlie" .
<http://example.org/alice#me>   foaf:knows <http://example.org/snoopy> .
<http://example.org/snoopy>     foaf:name  “Snoopy"@en .
```


https://www.w3.org/TR/sparql11-overview/

SPARQL — Examples

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x dc:title ?title .
          OPTIONAL { ?x ns:price ?price . FILTER (?price < 30) }
        }
```

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

?title	?price
“SPARQL Tutorial”	
“The Semantic Web”	23

https://www.w3.org/TR/sparql11-overview/

SPARQL — Summary of Key Features

- **Query form**: `SELECT`, `ASK`, `CONSTRUCT` and `DESCRIBE`
- **Dataset**: defined through `FROM` and `FROM named`
- **Graph patterns**:
  - Basic graph patterns (BGPs)
  - `UNION` graph patterns
  - `OPTIONAL` graph patterns
  - `GRAPH` graph patterns (specifies a named graph for the BGPs)
  - `SERVICE` graph patterns (specifies *where* a subset of BGPs should be evaluated)
  - Property paths
- **Solution modifiers**: `ORDER BY`, `LIMIT`, `DISTINCT`
- **Aggregations**: `GROUP BY`, aggregate functions

SPARQL — Querying Multiple Graphs

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x dc:title ?title .
          OPTIONAL { ?x ns:price ?price . FILTER (?price < 30) }
        }
```

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

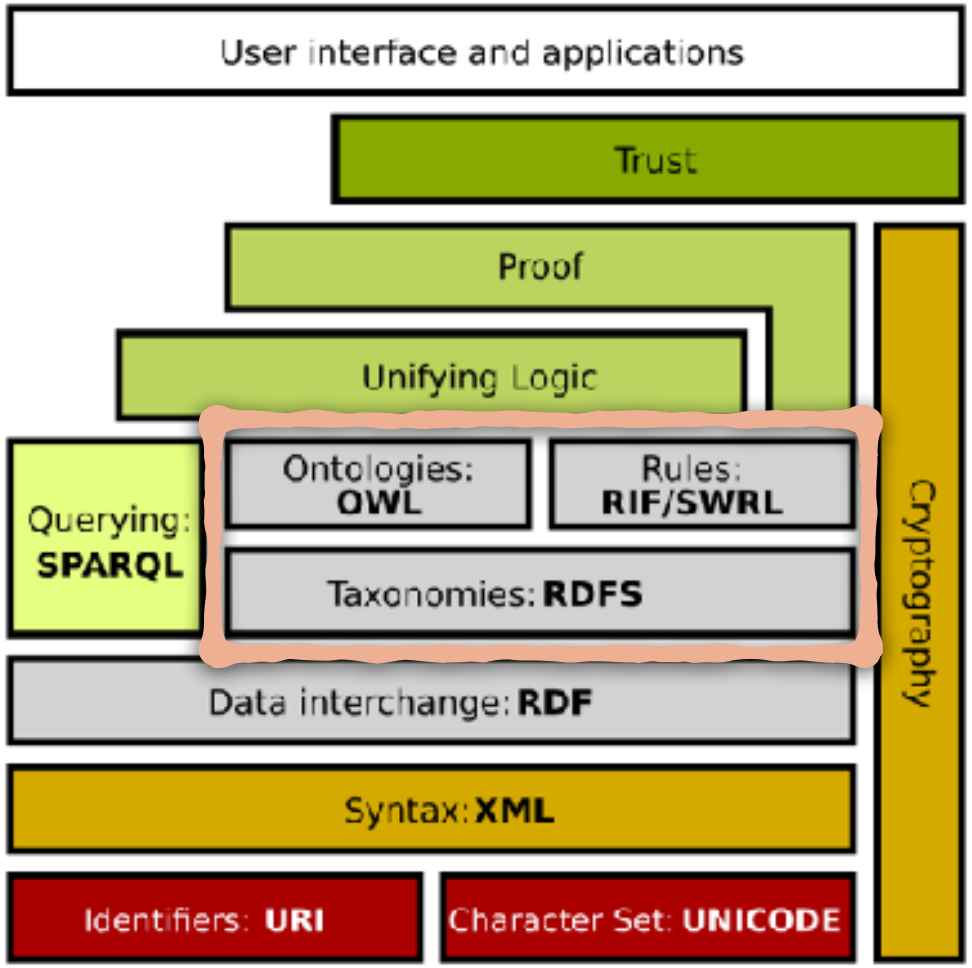
?title	?price
“SPARQL Tutorial”	
“The Semantic Web”	23

https://www.w3.org/TR/sparql11-overview/

Schemas

Why is RDF not enough?

- The RDF model is agnostic to the application domain
- Not limited or restricted to a specific use
- RDF Schema and OWL introduce useful and generic vocabularies to describe domains
- Domain independent structures
- Basic set of relations
- Recommendations at
  - https://www.w3.org/TR/rdf-schema/
  - https://www.w3.org/TR/owl2-overview/

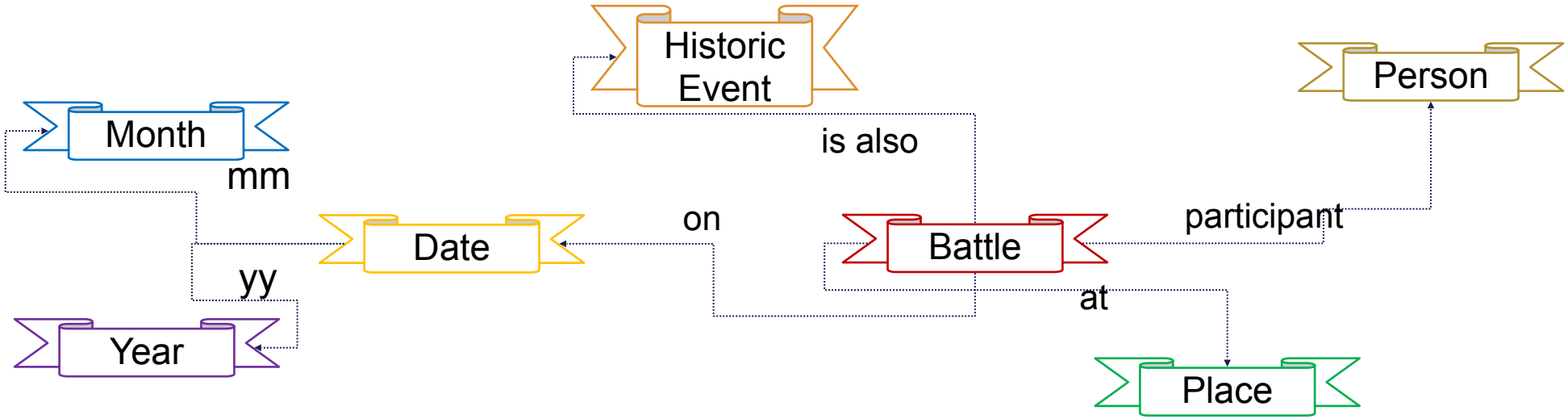


Smarter Data on the Web: Modeling the Domain

...

<p>The Battle of Waterloo was fought on Sunday, 18 June 1815, near Waterloo in <a href="…">Belgium<a>, part of the <a href="…" >United Kingdom of the Netherlands <a href="…"> at the time. A French army under the command of Napoleon Bonaparte was defeated by two of the armies of the .

...



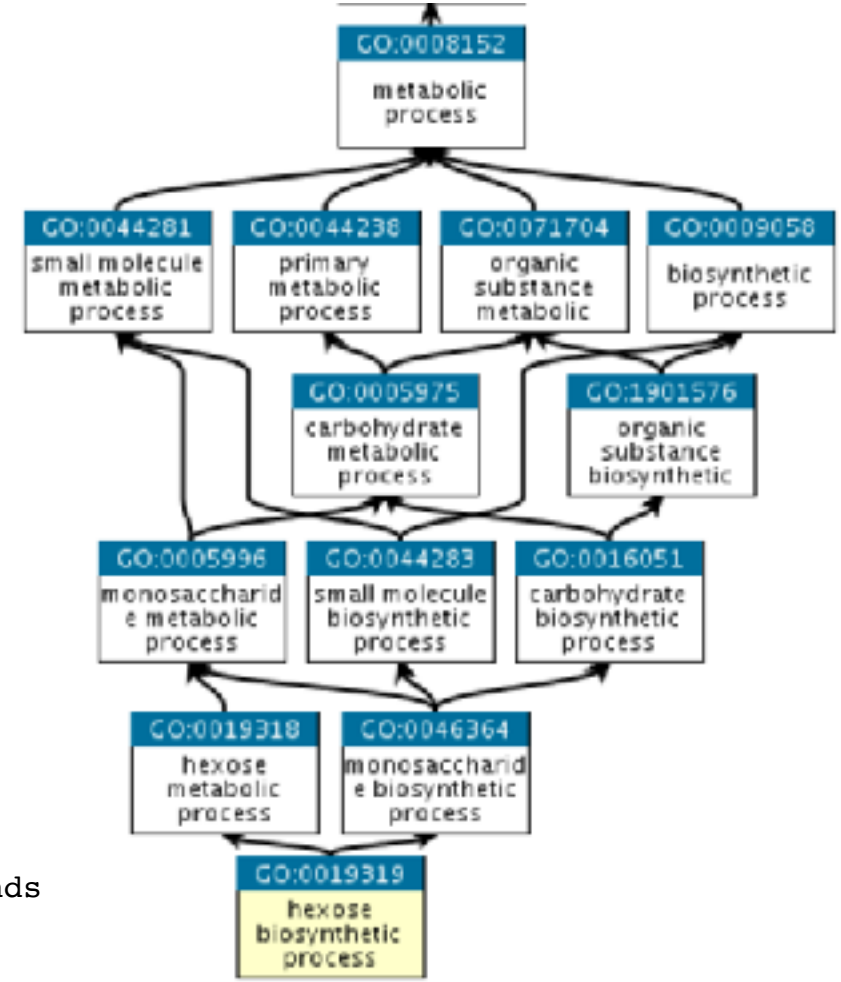
Ontologies

- An **ontology** is an explicit formal specifications of the concepts in a domain
- The ***terminology box*** (TBox) contains the description of the domain model
  - Similar to: classes in OOP, ER model in DBMS
  - A TBox contains *axioms*
  - E.g., `Student is_aPerson`
- The ***assertion box*** (ABox) contains the facts that describe a portion of reality
  - Similar to: instances in OOP, records in DBMS
  - An ABox contains *facts* or *assertions*
  - E.g., `Dale instance Student`

Example: Gene Ontology

- The Gene Ontology is a biomedical dataset
- Unify in a single model all the genes of all species, and the relations among them
- Ontologies to cope with the complex schema and relations
- Example Term:

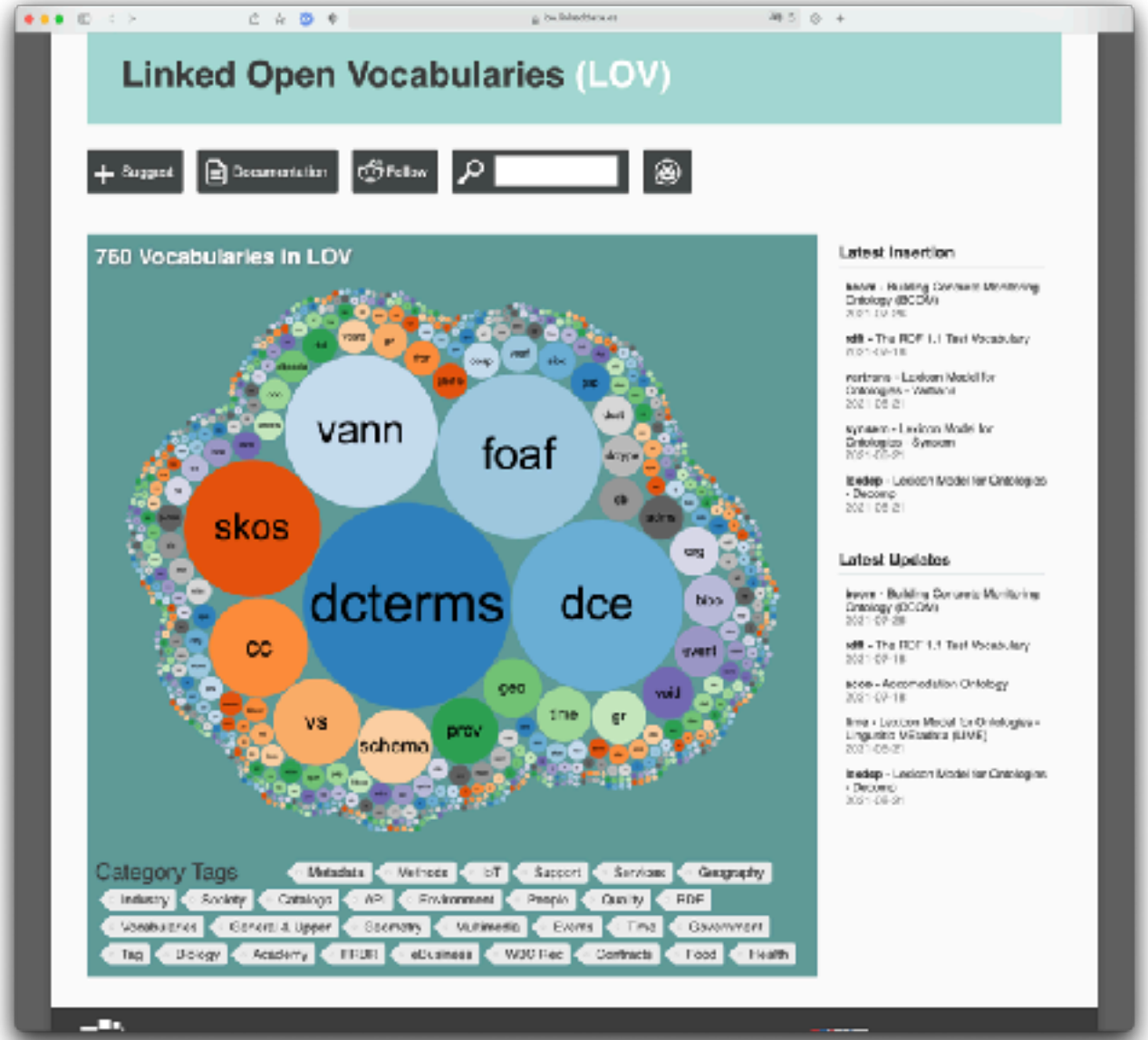
```
d:         GO:0000016
name:      lactase activity
ontology:  molecular_function
def:       "Catalysis of the reaction: lactose + H2O=D-glucose +
            D-galactose." [EC:3.2.1.108]
synonym:   "lactase-phlorizin hydrolase activity" BROAD [EC:3.2.1.108]
synonym:   "lactose galactohydrolase activity" EXACT [EC:3.2.1.108]
xref:      EC:3.2.1.108
xref:      MetaCyc:LACTASE-RXN
xref:      Reactome:20536
is_a:      GO:0004553 ! hydrolase activity, hydrolyzing O-glycosyl compounds
```



http://geneontology.org/docs/ontology-documentation/

Linked Open Vocabularies

- Collection of vocabularies
  - https://lov.linkeddata.es/dataset/lov/
  - More than 750 vocabularies to date
- Some examples:
  - `schema.org`: initiative from Google, Microsoft, Yahoo and Yandex for annotating web pages
  - `foaf`: one of the first ontologies, proposed describe people and their contacts
  - `cc`: a vocabulary to describe copyright licenses



Ontology Languages

- An **ontology language** is a language that allows to express an ontology
- The ontology language we want to define should have
  - A well-defined syntax
  - A formal semantics
  - Sufficient expressive power
  - Convenience of expression
  - Efficient reasoning support

- **RDF Schema (or RDFS)**: https://www.w3.org/TR/rdf-schema/
  - It complements the RDF model with some basic concepts (e.g. class and subclasses)
  - It uses namespace `rdfs`: http://www.w3.org/2000/01/rdf-schema#
- **OWL2**: https://www.w3.org/TR/owl2-overview/
  - A collection of five ontology languages
  - Extends RDF Schema with additional features (e.g. inverse properties)
  - It uses namespace `owl`: http://www.w3.org/2002/07/owl#

Ontology Languages: Well-defined syntaxes

- An **ontology language** is a language that allows to express an ontology
- The ontology language we want to define should have
  - A well-defined syntax
  - A formal semantics
  - Sufficient expressive power
  - Convenience of expression
  - Efficient reasoning support

- A well-defined syntax is necessary to let machine process the information
  - As in programming languages
- Well-defined means that the language can express everything in an unambiguous manner
  - RDF, RDF Schema and OWL have well-defined syntaxes

Ontology Languages: Well-defined syntaxes

- An **ontology language** is a language that allows to express an ontology
- The ontology language we want to define should have
  - A well-defined syntax
  - A formal semantics
  - Sufficient expressive power
  - Convenience of expression
  - Efficient reasoning support

- A well-defined syntax is necessary to let machine process the information
  - As in programming languages
- Well-defined means that the language can express everything in an unambiguous manner
- RDF Schema and OWL are languages to describe the application domain – concepts and relations among them
  - **RDF, RDF Schema and OWL have well-defined syntaxes, which is RDF**
  - OWL has some extensions

Ontology Languages: Formal semantics

- An **ontology language** is a language that allows to express an ontology
- The ontology language we want to define should have
  - A well-defined syntax
  - A formal semantics
  - Sufficient expressive power
  - Convenience of expression
  - Efficient reasoning support

- formal semantics *describes the meaning precisely*
  - It does not rely on subjective intuitions
  - It does not allow different interpretations
- Sentences written with a language with a well-defined syntax and a formal semantics can be interpreted
  - *A program or human can infer the meaning of the sentence*

Knowledge Representation (KR)

- A knowledge representation (KR) is most fundamentally *a surrogate, a substitute for the thing itself, used to enable an entity to determine consequences by thinking rather than acting*, i.e., by reasoning about the world rather than taking action in it.
- It is a set of ontological commitments, i.e., an answer to the question: *In what terms should I think about the world?*
- It is a *fragmentary theory of intelligent reasoning*, expressed in terms of three components: (i) the representation's fundamental conception of intelligent reasoning; (ii) the set of inferences the representation sanctions; and (iii) the set of inferences it recommends.
- It is a medium for pragmatically efficient computation, i.e., the computational environment in which thinking is accomplished…
- It is a medium of human expression, i.e., a language in which we say things about the world.
-
- KR is a part of philosophy and AI
- Logic is the basic concept at the basis of KR
- RDF Schema and OWL are related to modeling and representing knowledge

Davis, R., Shrobe, H., & Szolovits, P. (1993). What is a knowledge representation?. *AI magazine*, *14*(1), 17-17.

Knowledge Graph Schemas

Schema

- One of the benefits of modeling data as graphs — versus, for example, the relational model — is the option to forgo or postpone the definition of a schema.
- However, when modeling data as graphs, schemata *can* be used to prescribe a high-level structure and/or semantics that the graph follows or should follow.
- We discuss three types of graph schemata: *semantic*, *validating*, and *emergent*.

see KG Survey

Semantic Schema — Intuition

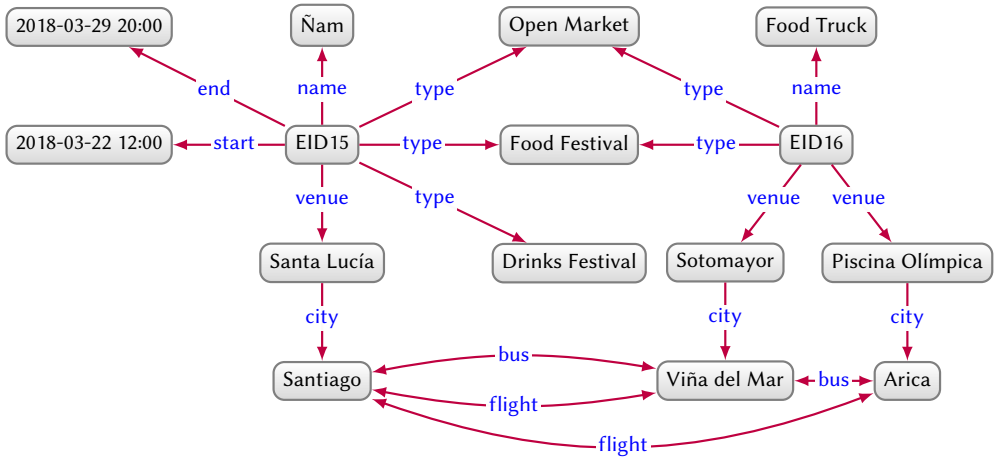


Fig. 1. Directed edge-labelled graph describing events and their venues.

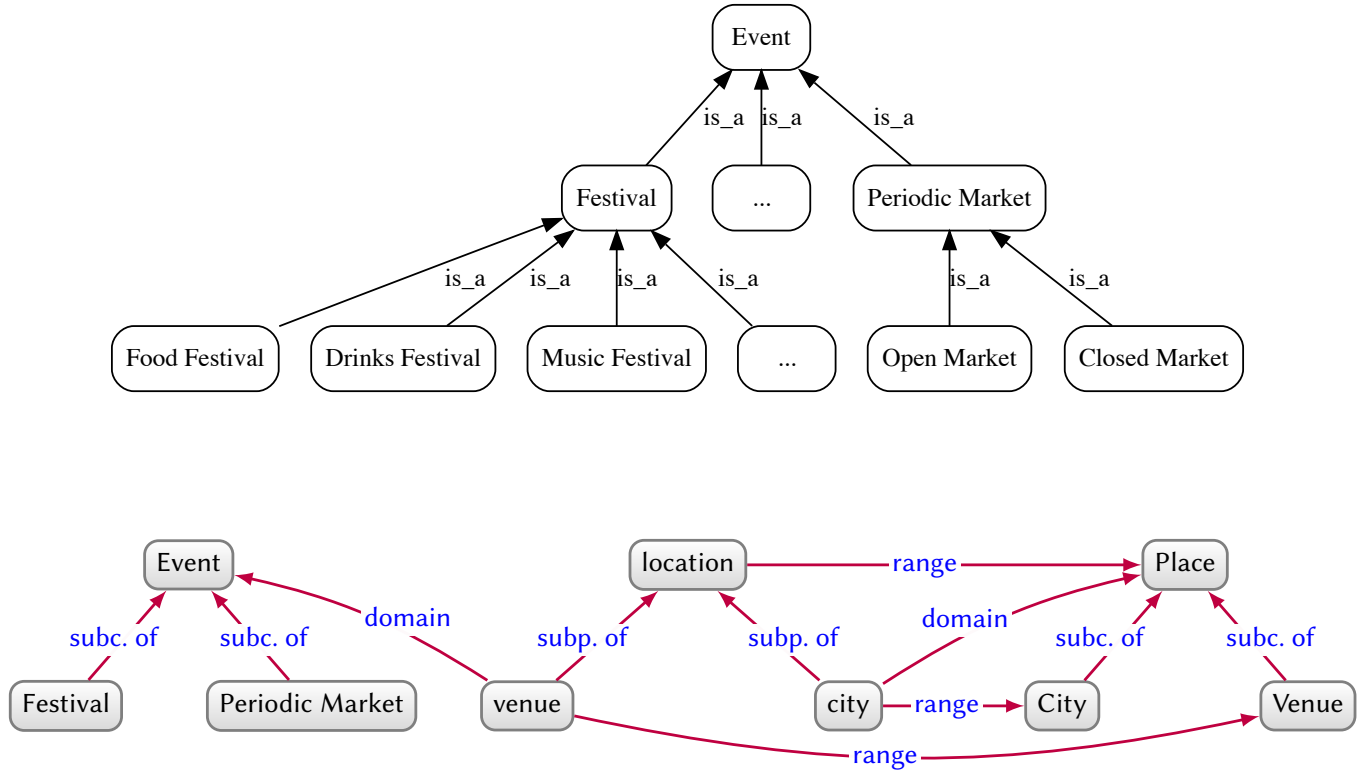


Fig. 12. Example schema graph describing sub-classes, sub-properties, domains, and ranges

see KG Survey

Semantic Schema — Intuition RDFS

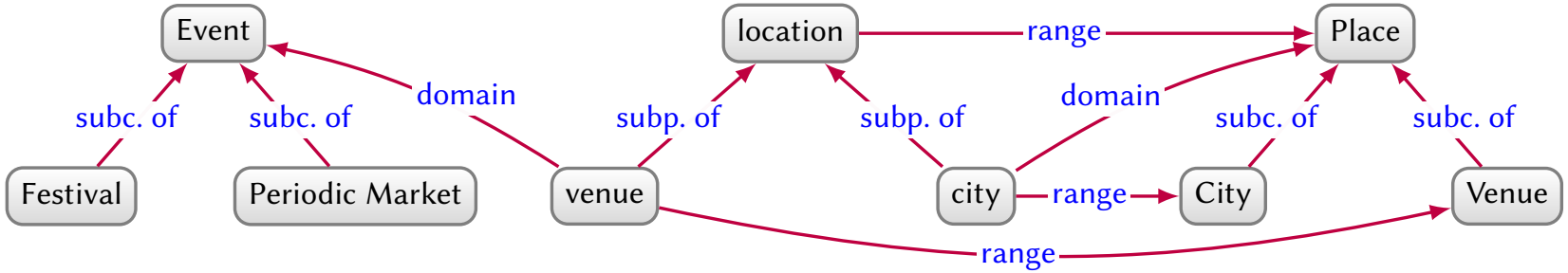


Fig. 12. Example schema graph describing sub-classes, sub-properties, domains, and ranges

Table 2. Definitions for sub-class, sub-property, domain and range features in semantic schemata

Feature	Definition	Condition	Example
Subclass	c —subc. of→ d	x —type→ c implies x —type→ d	City —subc. of→ Place
Subproperty	p —subp. of→ q	x —p→ y implies x —q→ y	venue —subp. of→ location
Domain	p —domain→ c	x —p→ y implies x —type→ c	venue —domain→ Event
Range	p —range→ c	x —p→ y implies y —type→ c	venue —range→ Venue

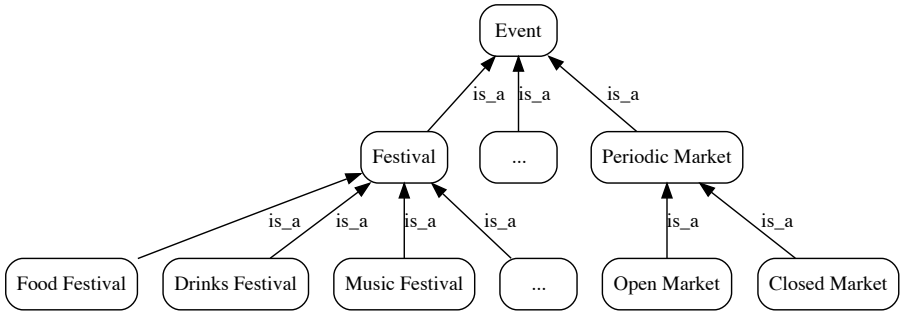
see KG Survey

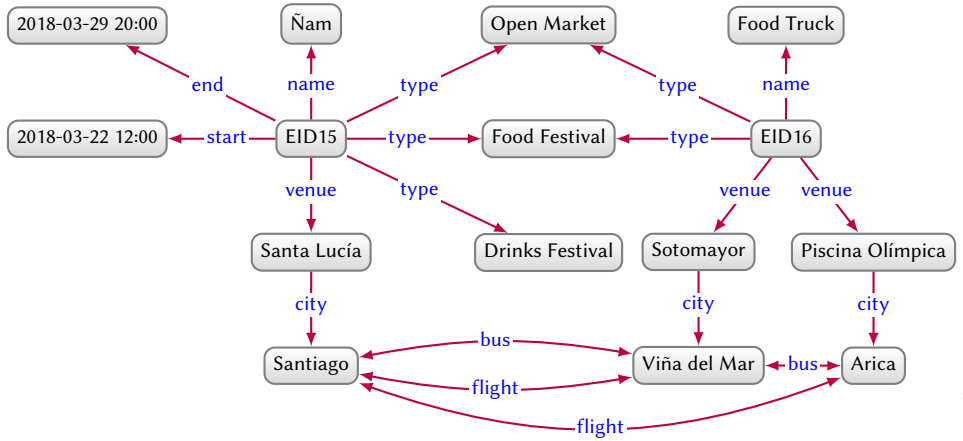
Semantic Schema — Intuition Ontologies

- Figure 1 implies that
  - `EID15` will be in `Santiago` even though 

is not explicitly represented
  - Cities with flights should have airports near by
- But 

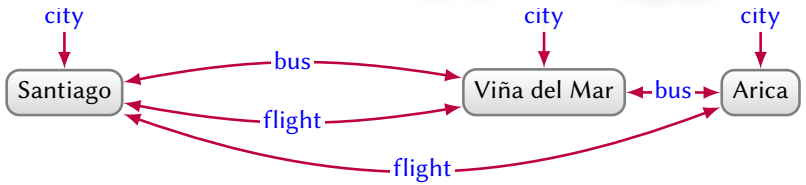
would be empty, as there is no Node named `Festival`
  - Using Class Hierarchy would make clear that `Food Festivals` and `Drinks Festival` are both of type `Festival`





see KG Survey

Semantic Schema — Interesting questions

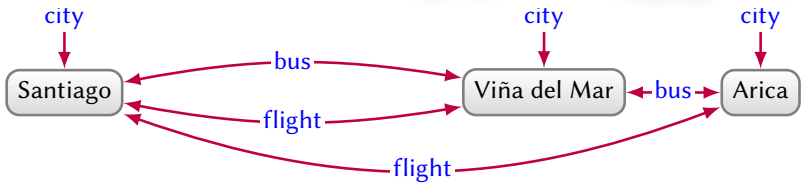


- Is there a `flight` between `Arica` and `Viña del Mar`?
  It depends on your assumptions about knowledge
  - **Closed World Assumption** (CWA: all knowledge is in the DB) → No flight
  - On the Web it might be better to have an **Open World Assumption** (OWA: I may not know everything) → There might or there might be no flight

- How many `flights` are there to `Santiago`? It depends on whether names are unique…
  - Unique Name Assumption (UNA) → *two flights*,
    as `Viña del Mar` and `Arica` are assumed to be two different places
  - Non Unique Name Assumption (NUNA) → *at least one*

see KG Survey

Semantic Schema — Interesting questions



- Is there a `flight` between `Arica` and `Viña del Mar`?
  It depends on your assumptions about knowledge
  - **Closed World Assumption** (CWA: all knowledge is in the DB) → No flight
  - On the Web it might be better to have an **Open World Assumption** (OWA: I may not know everything) → There might or there might be no flight
- How many `flights` are there to `Santiago`? It depends on whether names are unique…
  - Unique Name Assumption (UNA) → *two flights*,
    as `Viña del Mar` and `Arica` sue assumed to be two different places
  - Non Unique Name Assumption (NUNA) → *at least one*

see KG Survey

Semantic Schema — Ontology Features for Individuals

Table 3. Ontology features for individuals

Feature	Axiom	Condition	Example
Assertion	x —y→ z	x —y→ z	Chile —capital→ Santiago
Negation	n —type→ Neg; n —sub→ x; n —pre→ y; n —obj→ z	not x —y→ z	n —type→ Neg; n —sub→ Chile; n —pre→ capital; n —obj→ Arica
Same As	x_1 —same as→ x_2	$x_1 = x_2$	Región V —same as→ Región de Valparaíso
Different From	x_1 —diff. from→ x_2	$x_1 \neq x_2$	Valparaíso —diff. from→ Región de Valparaíso

see KG Survey

Semantic Schema — Ontology Features for Property Axioms

Table 4. Ontology features for property axioms

Feature	Axiom	Condition (for all x_*, y_*, z_*)	Example
Subproperty	p – subp. of → q	x – p → y implies x – q → y	venue – subp. of → location
Domain	p – domain → c	x – p → y implies x – type → c	venue – domain → Event
Range	p – range → c	x – p → y implies y – type → c	venue – range → Venue
Equivalence	p – equiv. p. → q	x – p → y iff x – q → y	start – equiv. p. → begins
Inverse	p – inv. of → q	x – p → y iff y – q → x	venue – inv. of → hosts
Disjoint	p – disj. p. → q	not x (q / p) y	venue – disj. p. → hosts
Transitive	p – type → Transitive	x – p → y – p → z implies x – p → z	part of – type → Transitive
Symmetric	p – type → Symmetric	x – p → y iff y – p → x	nearby – type → Symmetric
Asymmetric	p – type → Asymmetric	not x (p / p) y	capital – type → Asymmetric
see KG Survey	p – type → Reflexive	x – p	part of – type → Reflexive

Semantic Schema — Ontology Features for Property Axions

Asymmetric	p – type → Asymmetric	not $x \overset{p}{\underset{p}{\rightleftarrows}} y$	capital – type → Asymmetric
Reflexive	p – type → Reflexive	$x \circlearrowright p$	part of – type → Reflexive
Irreflexive	p – type → Irreflexive	not $x \circlearrowright p$	flight – type → Irreflexive
Functional	p – type → Functional	$y_1 \leftarrow p - x - p \rightarrow y_2$ implies $y_1 = y_2$	population – type → Functional
Inv. Functional	p – type → Inv. Functional	$x_1 - p \rightarrow y \leftarrow p - x_2$ implies $x_1 = x_2$	capital – type → Inv. Functional
Key	c – key → $p_1 \dots p_n$	c; x_1 – type → y_1 – p_1 → x_2; x_1 – $p_1, \dots, p_n$ → y_n ← $p_1, \dots, p_n$ – x_2; implies $x_1 = x_2$	City – key → lat, long
Chain	p – chain → $q_1 \dots q_n$	$x - q_1 \rightarrow y_1 - \dots \rightarrow y_{n-1} - q_n \rightarrow z$ implies $x - p \rightarrow z$	location – chain → location part of

see KG Survey

Semantic Schema — Ontology Features for Class Axioms and Defns.

Table 5. Ontology features for class axioms and definitions

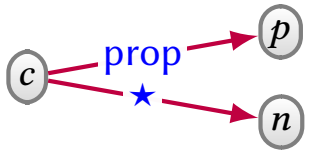
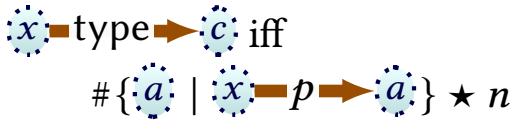
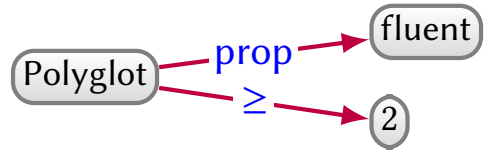
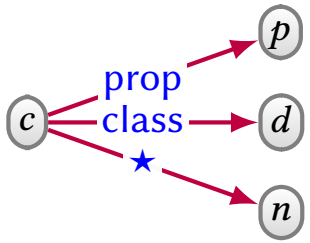
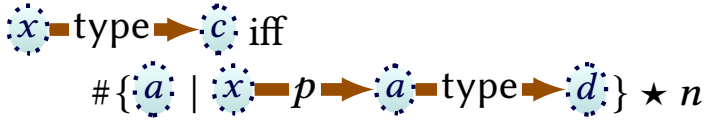
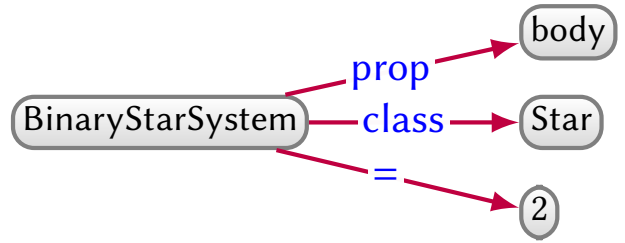
Feature	Axiom	Condition (for all x_*, y_*, z_*)	Example
Subclass	c —subc. of→ d	x —type→ c implies x —type→ d	City —subc. of→ Place
Equivalence	c —equiv. c.→ d	x —type→ c iff x —type→ d	Human —equiv. c.→ Person
Disjoint	c —disj. c.→ d	not c ←type— x —type→ d	City —disj. c.→ Region
Complement	c —comp.→ d	x —type→ c iff not x —type→ d	Dead —comp.→ Alive
Union	c —union→ $d_1 \dots d_n$	x —type→ c iff x —type→ d_1 or … or x —type→ d_n	Flight —union→ DomesticFlight InternationalFlight
Intersection	c —inter.→ $d_1 \dots d_n$	x —type→ c iff x —type→ d_1, …, x —type→ d_n	SelfDrivingTaxi —inter.→ Taxi SelfDriving
see KG Survey	x_1		Austria

Semantic Schema — Ontology Features for Class Axioms and Defns.

Enumeration	c — one of → $x_1, \vdots, x_n$	x — type → c iff $x \in \{x_1, \ldots, x_n\}$	EUState — one of → Austria, ⋮, Sweden
Some Values	c — prop → p; c — some → d	x — type → c iff there exists a such that x — p → a — type → d	EUCitizen — prop → nationality; EUCitizen — some → EUState
All Values	c — prop → p; c — all → d	x — type → c iff for all a with x — p → a it holds that a — type → d	Weightless — prop → has part; Weightless — all → Weightless
Has Value	c — prop → p; c — value → y	x — type → c iff x — p → y	ChileanCitizen — prop → nationality; ChileanCitizen — value → Chile
Has Self	c — prop → p; c — self → true	x — type → c iff x — p → x	SelfDriving — prop → driver; SelfDriving — self → true

see KG Survey

Semantic Schema — Ontology Features for Class Axioms and Defns.

CARDINALITY $\star \in \{=, \leq, \geq\}$			
QUALIFIED CARDINALITY $\star \in \{=, \leq, \geq\}$			

see KG Survey

Semantic Schema — Rule-based Interpretation

Table 6. Example rules for sub-class, sub-property, domain, and range features

Feature	Body	⇒	Head
Subclass (I)	?x —type→ ?c —subc. of→ ?d	⇒	?x —type→ ?d
Subclass (II)	?c —subc. of→ ?d —subc. of→ ?e	⇒	?c —subc. of→ ?e
Subproperty (I)	?x —?p→ ?y ; ?p —subp. of→ ?q	⇒	?x —?q→ ?y
Subproperty (II)	?p —subp. of→ ?q —subp. of→ ?r	⇒	?p —subp. of→ ?r
Domain	?x —?p→ ?y ; ?p —domain→ ?c	⇒	?x —type→ ?c
Range	?x —?p→ ?y ; ?p —range→ ?c	⇒	?y —type→ ?c

see KG Survey

Description Logics (DL)

- **DL knowledge base** K is defined as a tuple (A, T, R), where
  A is the *ABox*: a set of assertional axioms;
  T is the *TBox*: a set of class (aka concept/terminological) axioms; and
  R is the *RBox*: a set of relation (aka property/role) axioms

- **DL interpretation** I is defined as a pair $(\Delta^I, \cdot^I)$, where
  Δ^I is the *interpretation domain*}, and
  $\cdot^I$ is the *interpretation function*.
  The interpretation domain is a set of individuals. The interpretation function accepts a definition of either an individual a, a class C, or a relation R, mapping them, respectively, to an element of the domain ($a^I \in \Delta^I$), a subset of the domain ($C^I \subseteq \Delta^I$), or a set of pairs from the domain ($R^I \subseteq \Delta^I \times \Delta^I$).

see KG Survey

Description Logics (DL) Semantics — Classes

Table 7. Description Logic semantics

Name	Syntax	Semantics $(\cdot^I)$
	Class Definitions	
Atomic Class	A	A^I (a subset of Δ^I)
Top Class	$\top$	Δ^I
Bottom Class	$\bot$	$\emptyset$
Class Negation	$\neg C$	$\Delta^I \setminus C^I$
Class Intersection	$C \sqcap D$	$C^I \cap D^I$
Class Union	$C \sqcup D$	$C^I \cup D^I$
Nominal	$\{a_1, \ldots, a_n\}$	$\{a_1^I, \ldots, a_n^I\}$
Existential Restriction	$\exists R.C$	$\{x \mid \exists y : (x, y) \in R^I \text{ and } y \in C^I\}$
Universal Restriction	$\forall R.C$	$\{x \mid \forall y : (x, y) \in R^I \text{ implies } y \in C^I\}$
Self Restriction	$\exists R.\text{Self}$	$\{x \mid (x, x) \in R^I\}$
Number Restriction	$\star\, n\, R$ (where $\star \in \{\geq, \leq, =\}$)	$\{x \mid \#\{y : (x, y) \in R^I\} \star n\}$
Qualified Number Restriction	$\star\, n\, R.C$ (where $\star \in \{\geq, \leq, =\}$)	$\{x \mid \#\{y : (x, y) \in R^I \text{ and } y \in C^I\} \star n\}$

see KG Survey

Description Logics (DL) Semantics — Classes and Relations

Class Axioms (T-Box)		
Class Inclusion	$C \sqsubseteq D$	$C^I \subseteq D^I$
Relation Definitions		
Relation	R	R^I (a subset of $\Delta^I \times \Delta^I$)
Inverse Relation	R^-	$\{(y, x) \mid (x, y) \in R^I\}$
Universal Relation	U	$\Delta^I \times \Delta^I$
Relation Axioms (R-Box)		
Relation Inclusion	$R \sqsubseteq S$	$R^I \subseteq S^I$
Complex Relation Inclusion	$R_1 \circ \ldots \circ R_n \sqsubseteq S$	$R_1^I \circ \ldots \circ R_n^I \subseteq S^I$
Transitive Relations	$\mathsf{Trans}(R)$	$R^I \circ R^I \subseteq R^I$
Functional Relations	$\mathsf{Func}(R)$	$\{(x, y), (x, z)\} \subseteq R^I$ implies $y = z$
Reflexive Relations	$\mathsf{Ref}(R)$	for all $x \in \Delta^I : (x, x) \in R^I$
Irreflexive Relations	$\mathsf{Irref}(R)$	for all $x \in \Delta^I : (x, x) \notin R^I$
Symmetric Relations	$\mathsf{Sym}(R)$	$R^I = (R^-)^I$
Asymmetric Relations	$\mathsf{Asym}(R)$	$R^I \cap (R^-)^I = \emptyset$
Disjoint Relations	$\mathsf{Disj}(R, S)$	$R^I \cap S^I = \emptyset$

see KG Survey

Description Logics (DL) Semantics — Individuals

Individual	a	a^I (an element of Δ^I)
	Assertional Definitions	
	Assertional Axioms (A-Box)	
Relation Assertion	$R(a, b)$	$(a^I, b^I) \in R^I$
Negative Relation Assertion	$\neg R(a, b)$	$(a^I, b^I) \notin R^I$
Class Assertion	$C(a)$	$a^I \in C^I$
Equality	$a = b$	$a^I = b^I$
Inequality	$a \neq b$	$a^I \neq b^I$

see KG Survey