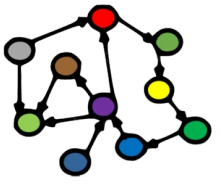
Welcome to INFO216:
Knowledge Graphs
Spring 2023

Andreas L Opdahl
<Andreas.Opdahl@uib.no>

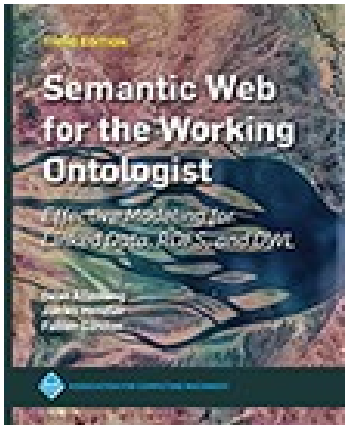
Session 3: Querying and updating KGs (SPARQL)

- Themes:
  - about INFO216!
  - SPARQL queries
    - SPARQL 1.1
  - SPARQL Update
    - SPARQL 1.1 Update Language
  - programming SPARQL queries and updates

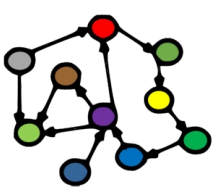


Readings

- Sources:
  - Allemang, Hendler & Gandon (2020): Semantic Web for the Working Ontologist, 3rd edition: chapter 6 (SPARQL, but chapter 5 in the 2nd edition)
  - Blumauer & Nagy (2020): Knowledge Graph Cookbook – Recipes that Work: for example around pages 54-55 and 133 (*supplementary)*
- Materials in the wiki <http://wiki.uib.no/info216>:
  - W3C resources
    - SPARQL 1.1 Cheat Sheet
    - http://www.w3.org/TR/sparql11-query/
    - http://www.w3.org/TR/sparql11-update/
  - Blazegraph wiki
  - RDFLib documentation

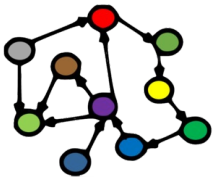


About INFO216



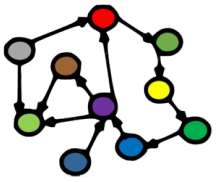
Purpose

- To learn theories, techniques, tools, and best practices for managing knowledge graphs.
- To acquire understanding and skills for programming applications that use and produce such data and metadata.
- To learn about existing sources of and standards for big, open, and semantic data.
- To gain practical experience in developing knowledge graph-based applications using technologies such as RDF, RDFS, OWL, SPARQL, and JSON-LD.



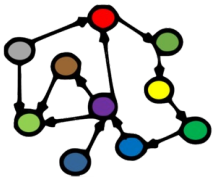
Curriculum

- Course book (*the whole book is mandatory*):
  - Allemang, Hendler & Gandon (2020).
    Semantic Web for the Working Ontologist,
    Effective Modeling for Linked Data, RDFS and OWL (Third Edition)
- Supplementary course book (*supplementary, not mandatory*):
  - Blumauer & Nagy (2020).
    The Knowledge Graph Cookbook - Recipes that Work
- Additional readings (both *mandatory* and *recommended*) will be made available in the course wiki: https://wiki.uib.no/info216
- The lectures and lectures notes are also *mandatory* parts of the curriculum.



Practical

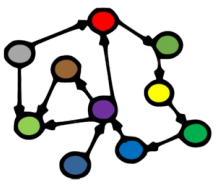
- 14 lectures:
  - Tuesdays 1215-1400
- 14 lab weeks:
  - 2 hours of weekly lab groups
  - starting this week, no labs week 10 and 14 (Easter)
  - seminar/lab leader: Robin Johansen Bøe <Robin.Boe@student.uib.no>
- Evaluation:
  - individual, written 4-hour exam
- Requirements:
  - participation in 75% of labs (11 of 14)
- Course wiki:
  - http://wiki.uib.no/info216



Lecture plan (tentative)

1. Introduction to KGs
2. Representing KGs (RDF)
3. Querying and updating KGs (SPARQL)
4. Open KGs 1
5. Open KGs 2
6. Enterprise KGs
7. Rules (RDFS)
8. Ontologies (OWL)
9. Vocabularies
10. Reasoning about KGs (DL)
11. Formal ontologies (OWL-DL)
12. KG embeddings 1
13. KG embeddings 2
14. Knowledge engineering

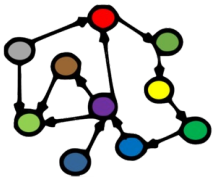
You learn KGs best through practice: do the lab exercises thoroughly!



SPARQL

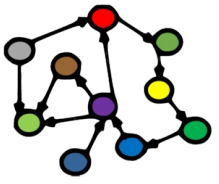
Database languages

- How to manage and use databases (DBs)?
  - everything can be programmed
    - connector APIs
  - native DB languages can be simpler and more effective
    - *data manipulation languages (DMLs)*
      - querying and updating
    - also: data definition (DDL), data control (DCL), transaction control (DCL), ...
    - accessible through GUIs, web GUIs, web APIs
  - supported by database management systems (DBMSs)



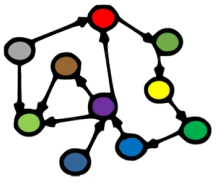
Database languages: SPARQL

- *Simple Protocol and RDF Query Language*
- A data mainpulation language (primarily) for RDF data
- Supported by
  - rdflib and other programming APIs
  - *triple stores* - specialised DBMSs for RDF
    - we will use *Blazegraph*



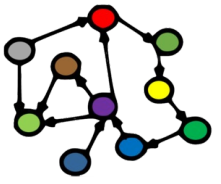
Triple store: Blazegraph

- Native triple store
  - SPARQL queries and updates, including federation
    - also general graph support
  - simple web-based interface
  - multi-tenancy: several *namespaces* in same Blazegraph instance
- Many, many options
  - full-text search, geo-spatial data
  - *plain triples (RDF)*, optional inference
  - “quads” (no inference)
  - “reification done right (RDR)”, optional inference
- *Built to scale for data and processing*



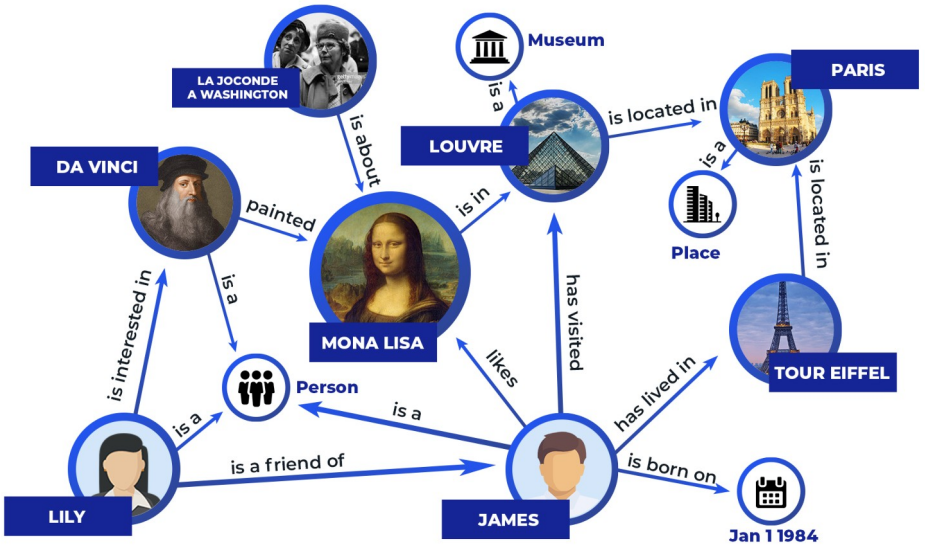
Triple store: Blazegraph

- Dual licensing: both GPLv2 and commercial
- Built around Bigdata, can be run
  - *embedded* in a program
  - single-machine *stand-alone* server (< 50B triples)
  - *replicated* servers (scale-up queries)
  - perhaps also *federated* servers (> 8 machines, scale-out data)
- Used by *Wikidata* and others
  - (most likely) the starting point for *Amazon Neptune*
- Easy to run:
  - from command line: java -server -jar blazegraph.jar
  - then access in a web browser: http://localhost:9999/

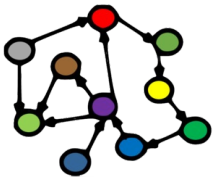


SPARQL queries

```
SELECT ?person ?museum WHERE {
        ?person ex:likes ?painting .
        ?painting ex:is_in ?museum .
}
```

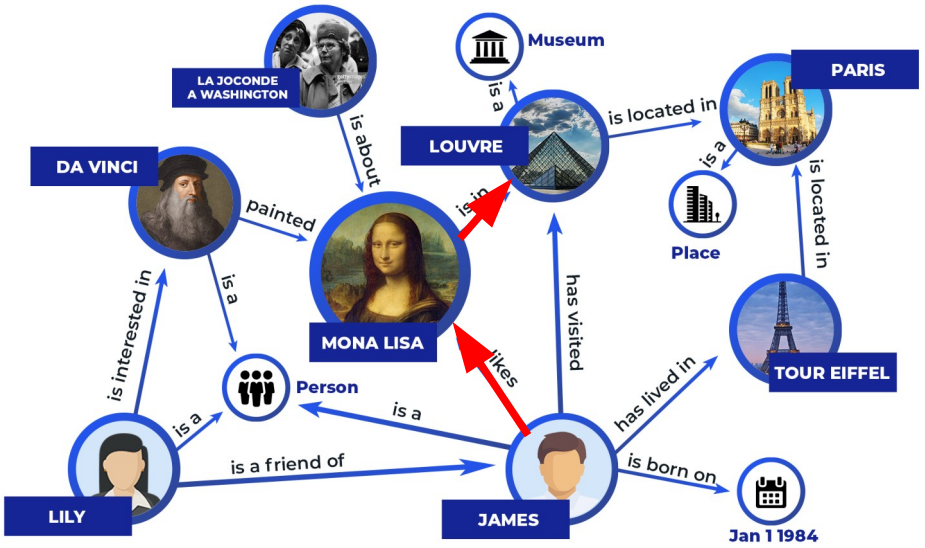


- **Main idea:**
  - give SPARQL an underspecified RDF graph: a **pattern**...
  - some of the nodes are **variables**:
    - other nodes can be **IRIs**, or **literal** values as before
  - SPARQL tries to **match** the pattern to an RDF graph
  - ...and returns each match as a **row** in the **result**

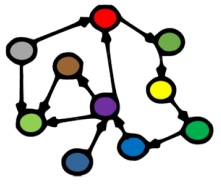


SPARQL queries

```
SELECT ?person ?museum WHERE {
        ?person ex:likes ?painting .
        ?painting ex:is_in ?museum .
}
```

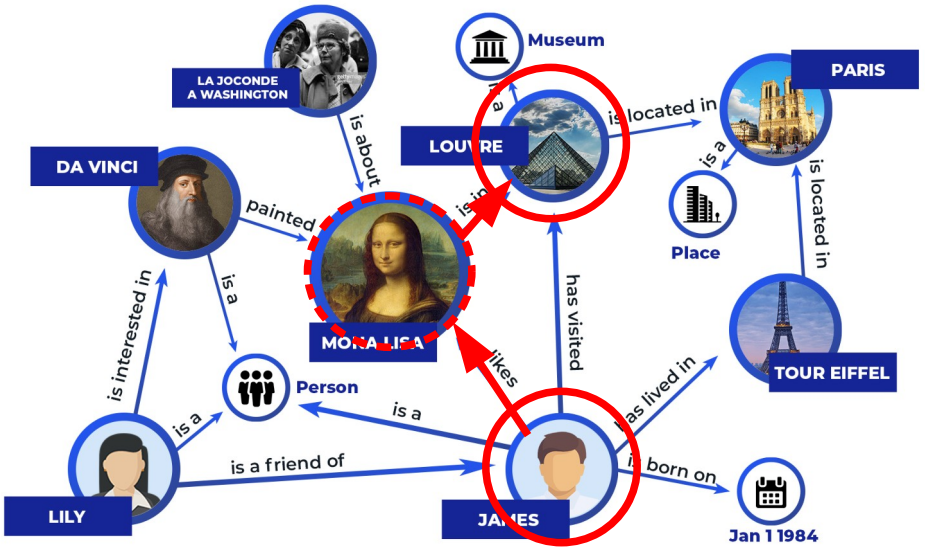


- **Main idea:**
  - give SPARQL an underspecified RDF graph: a **pattern**...
  - some of the nodes are **variables**:
    - other nodes can be **IRIs**, or **literal** values as before
  - SPARQL tries to **match** the pattern to an RDF graph
  - ...and returns each match as a **row** in the **result**

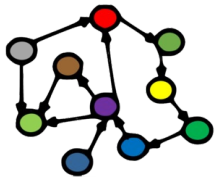


SPARQL queries

```
SELECT ?person ?museum WHERE {
        ?person ex:likes ?painting .
        ?painting ex:is_in ?museum .
}
```

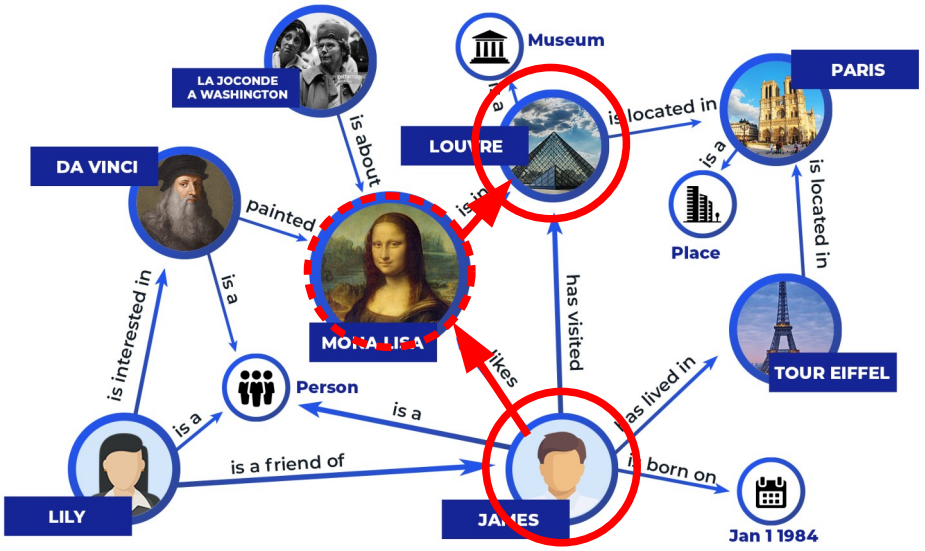


- **Main idea:**
  - give SPARQL an underspecified RDF graph: a **pattern**...
  - some of the nodes are **variables**:
    - other nodes can be **IRIs**, or **literal** values as before
  - SPARQL tries to **match** the pattern to an RDF graph
  - ...and returns each match as a **row** in the **result**

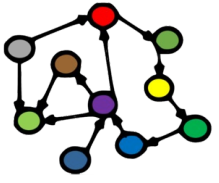


SPARQL queries

```
SELECT ?person ?museum WHERE {
        ?person ex:likes ?painting .
        ?painting ex:is_in ?museum .
        ?museum rdf:type ex:Museum .
}
```

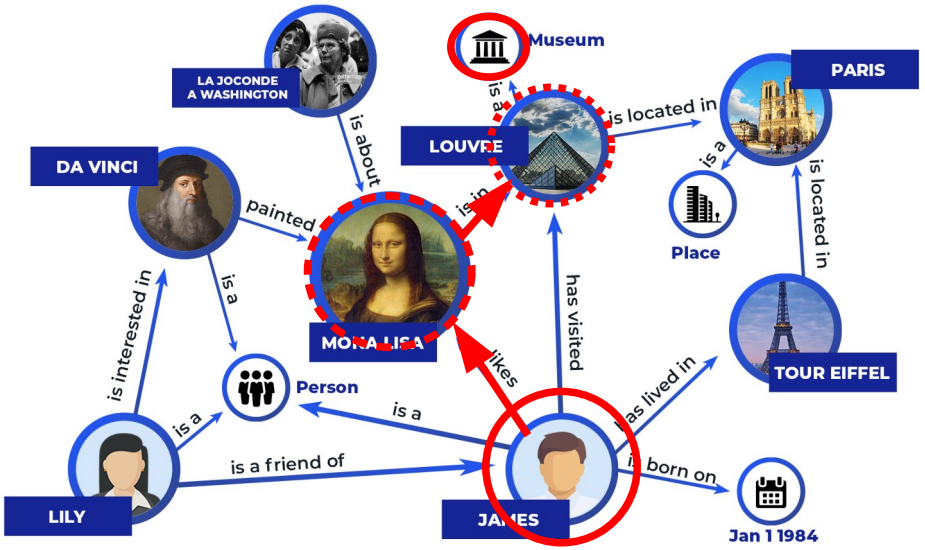


- **Main idea:**
  - give SPARQL an underspecified RDF graph: a **pattern**...
  - some of the nodes are **variables**:
    - other nodes can be **IRIs**, or **literal** values as before
  - SPARQL tries to **match** the pattern to an RDF graph
  - ...and returns each match as a **row** in the **result**

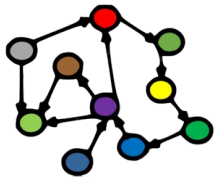


SPARQL queries

```
SELECT ?person ?museum WHERE {
        ?person ex:likes ?painting .
        ?painting ex:is_in ?museum .
        ?museum rdf:type ex:Museum .
}
```

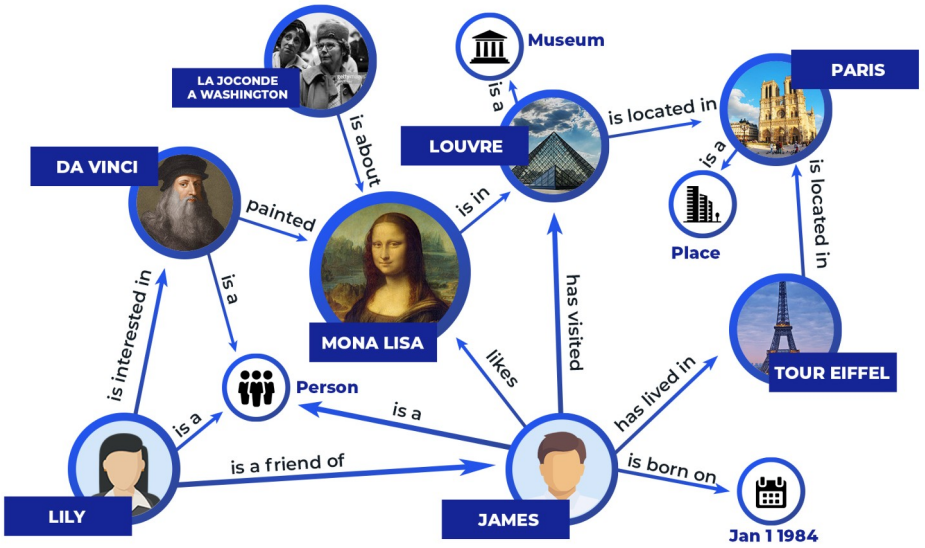


- **Main idea:**
  - give SPARQL an underspecified RDF graph: a **pattern**...
  - some of the nodes are **variables**:
    - other nodes can be **IRIs**, or **literal** values as before
  - SPARQL tries to **match** the pattern to an RDF graph
  - ...and returns each match as a **row** in the **result**

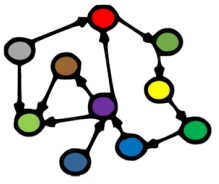


SPARQL queries

```
PREFIX ex: <http://example.org/>
SELECT ?person ?museum WHERE {
        ?person ex:likes ?painting .
        ?painting ex:is_in ?museum .
        ?museum rdf:type ex:Museum .
}
```

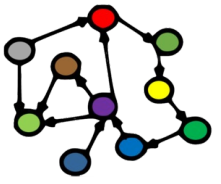


- **Main idea:**
  - give SPARQL an underspecified RDF graph: a **pattern**...
  - some of the nodes are **variables**:
    - other nodes can be **IRIs**, or **literal** values as before
  - SPARQL tries to **match** the pattern to an RDF graph
  - ...and returns each match as a **row** in the **result**

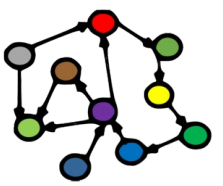


Overview of SPARQL

- SPARQL's DML (RDF data manipulation):
  - four types of *queries*:
    - SELECT: returns table
    - ASK: returns yes/no
    - CONSTRUCT: returns a graph
    - DESCRIBE: returns a graph
  - three types of *updates:*
    - DELETE, INSERT, DELETE/INSERT
- SPARQL's DDL (triple store management):
    - LOAD, CLEAR, ...

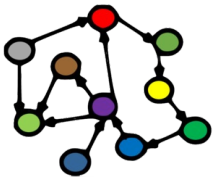


SPARQL queries

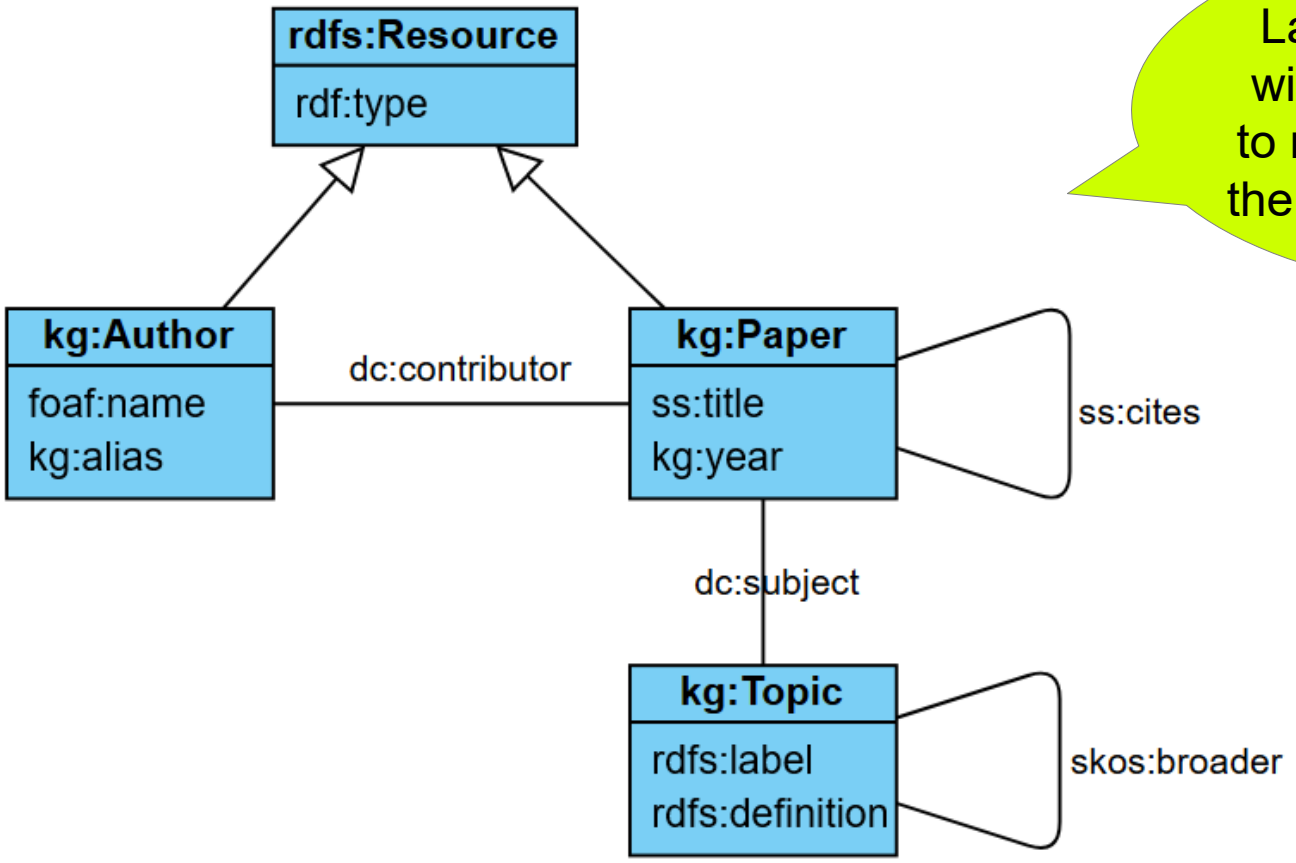


Example KG

- A knowledge graph of research literature related to “Knowledge Graphs for the News”
  - built to support an recent literature study
  - 78 main papers with 291 authors
  - 4086 other papers with 8990 authors
  - 100s of topics and themes, >300k triples
- Accessible at http://bg.newsangler.uib.no/
  - runs on a Blazegraph triple store
  - simple web front end, read only

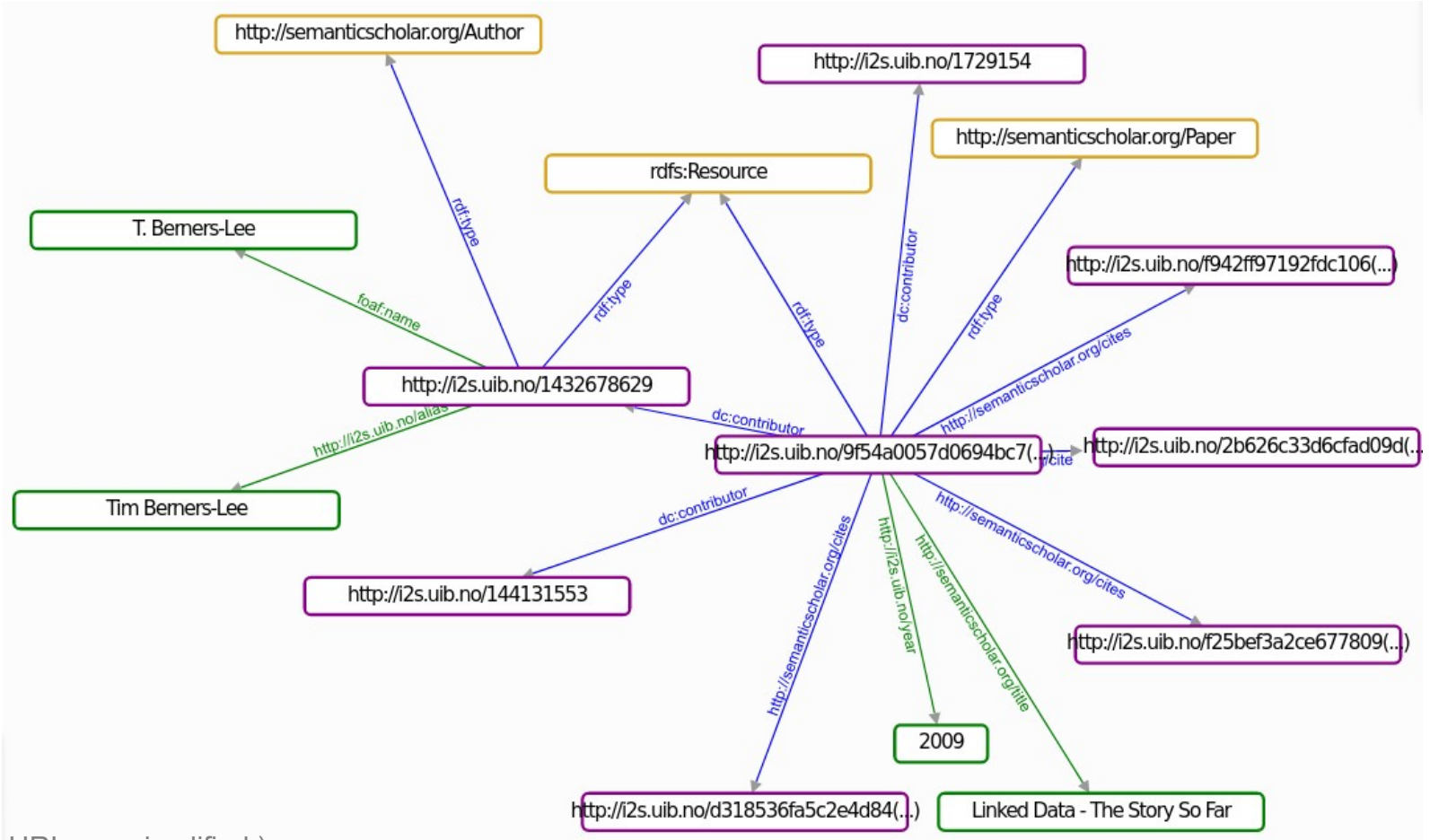


Example KG: graph structure



Later in the course, we will use RDFS and OWL to represent and visualise the structure of ontologies.

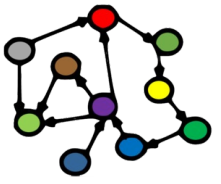
Example KG: resources



(The URIs are simplified.)

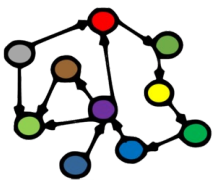
SELECT queries: Basic forms

- Basic form:
  - **SELECT** *projection* **WHERE { ***pattern* **}**
  - the *projection* is a list of *variables*
  - the *pattern* is (essentially) a list of *triples*
  - returns a table with *one row per result* and *one column per projection variable*
- Optional and combinable variations:
  - SELECT ***** WHERE { ... }
  - SELECT **DISTINCT** ... WHERE { ... }
  - SELECT ... WHERE { ... } **LIMIT *n***
  - SELECT ... WHERE { ... } **LIMIT *n* OFFSET *m***
  - SELECT ... WHERE { ... } **ORDER BY** *...variable...*
  - SELECT ... WHERE { ... } **ORDER BY DESC(***...variable...***)**



SELECT queries: Filters

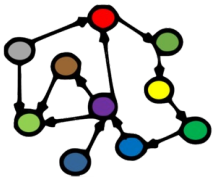
- WHERE-variants with *filter expressions*:
  - ... WHERE { ...*pattern*... **FILTER (...*filter_expression*...)** }
  - ... WHERE { ...*pattern*... **FILTER [NOT] EXISTS { ...*filter_expr*... }** }
- *Filters* are *logical expressions*:
  - standard logic operators: !, &&, ||
  - (in-)equality operators: =, !=, <, <=, >, >=
  - arithmetic: +, -, /. *
  - *built-in, imported (xsd:...) and self-defined functions*



SELECT queries: Built-in functions

- Examples of built-in functions (for example in FILTERs):
  - bound, if
  - exists, not exists
  - in, not in
  - IRI, bnode
  - isIRI, isBlank, isLiteral, isNumeric
  - str, lang, strlang (“for language tagged literals”@en)
  - regex, strlen, contains, substr. ... (from XPath)
  - replace
  - abs, rand, ceil, floor
  - now, year, month, days, hours, minutes, seconds

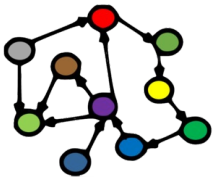
https://en.wikibooks.org/wiki/SPARQL/Expressions_and_Functions



SELECT queries: Grouping

- Grouping of results:
  - SELECT *...grouping or aggregate variables...* WHERE { ... }
    GROUP BY *...grouping variables...*
  - example: counting students in courses

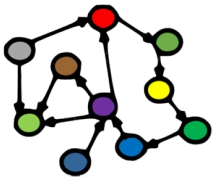
```
SELECT (COUNT(?paper) AS ?count) WHERE {
        ?paper dct:contributor ?author .
}
GROUP BY ?author
LIMIT 10
```



SELECT queries: Grouping

- Grouping of results:
  - SELECT *...grouping or aggregate variables...* WHERE { ... }
    GROUP BY *...grouping variables...*
  - example: counting students in courses

```
SELECT ?author (COUNT(?paper) AS ?count) WHERE {
        ?paper dct:contributor ?author .
}
GROUP BY ?author
LIMIT 10
```

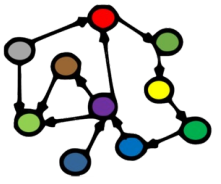


SELECT queries: Grouping

- Grouping of results:
  - SELECT *...grouping or aggregate variables...* WHERE { ... }
    GROUP BY *...grouping variables...* **[HAVING ...]**
  - example: counting students in courses

```
SELECT ?author (COUNT(?paper) AS ?count) WHERE {
        ?paper dct:contributor ?author .
}
GROUP BY ?author
HAVING (?count >= 10)
LIMIT 10
```

Similar to *filter expressions*

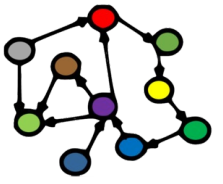


SELECT queries: Grouping

- Grouping of results:
  - SELECT *...grouping or aggregate variables...* WHERE { ... }
    GROUP BY *...grouping variables...* **[HAVING ...] [ORDER BY ...]**
  - example: counting students in courses

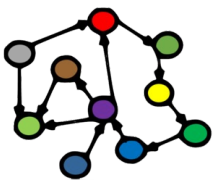
```
SELECT ?author (COUNT(?paper) AS ?count) WHERE {
        ?paper dct:contributor ?author .
}
GROUP BY ?author
HAVING (?count >= 10)
ORDER BY DESC(?count)
LIMIT 10
```

Similar to *filter expressions*



SELECT queries: Grouping

- Grouping of results:
  - SELECT *...grouping or aggregate variables...* WHERE { ... }
    GROUP BY *...grouping variables...*
    [HAVING *...expression...* **]**
    [ORDER BY *...variables... and DESC(...variables...)* **]**
  - *grouping variables:*
    - regular variables
    - used to group the rows in the results
  - *aggregate variables:*
    - the results of aggregate functions, for example:
      SUM(), COUNT(), MIN(), MAX(), AVG(),
      GROUP_CONCAT(), SAMPLE()



SELECT queries: Bindings

- In the list of projection variables:
  - SELECT **(COUNT (...) AS ...)** ... WHERE { ... }
- In the graph pattern:
  - SELECT ... WHERE { ... **BIND(... AS ...)** ... }
- In the GROUP BY variable list
  - SELECT ... WHERE { ... }
    GROUP BY ... **(... AS ...)** ...
- In-line values:

SELECT ... WHERE {

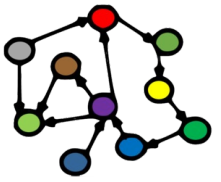
...

VALUES *var* { *value1 value2 ... valuen* }

...

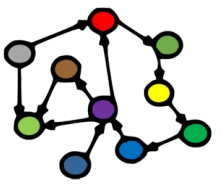
}

var and ***valuen*** can even be tuples ***(v1, v2, … vm)***



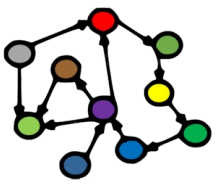
SELECT queries: Composite patterns

- WHERE-variants with multiple *pattern groups*:
  - ... WHERE { ...*pattern*... **OPTIONAL { ...*pattern*... }** }
  - ... WHERE { ...*pattern*... **MINUS { ...*pattern*... }** }
  - ... WHERE { **{ ...*pattern*... } UNION { ...*pattern*... }** }
- ...as well as *filters*:
  - ... WHERE { ...*pattern*... **FILTER (...*filter_expression*...)** }
  - ... WHERE { ...*pattern*... **FILTER [NOT] EXISTS { ...*filter_expr*... }** }



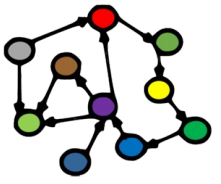
SELECT queries: Federated queries

- Nested queries:
  SELECT ... WHERE { ... **{ SELECT ... WHERE { ... } }** }
- Remote queries:
  SELECT ... WHERE { ... **SERVICE <irl> { ... }** }
  SELECT ... WHERE { ... **SERVICE SILENT <irl> { ... }** }



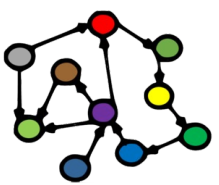
SELECT queries: Federated queries

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>

SELECT DISTINCT ?enname ?person ?wdperson WHERE {

   BIND("T. Berners-Lee"@en AS ?enname)
   SERVICE <https://query.wikidata.org/bigdata/namespace/wdq/sparql> {
      SELECT ?wdperson ?enname WHERE {
         ?wdperson skos:altLabel ?enname .
      }
   }

   BIND(STR(?enname) AS ?name)
   ?person foaf:name ?name .
}
LIMIT 1
```



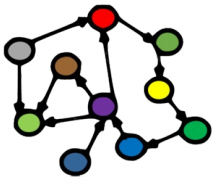
SELECT: Composite properties

- *Property paths (in SPARQL 1.1)*:
  - Concatenation: ***a / b*** (means “first ***a***, then ***a***'s ***b***”)
  - Inversion: ***^a*** (means “***a*** backwards”)
  - Grouping: ***(...)*** (nested composite properties)
  - Repetition: ***a**** (0:n), ***a+*** (1:n), ***a?*** (0:1)
  - Alternative: ***a | b*** (either ***a*** or ***b***)
  - Negation: ***!a*** (any other property than ***a***)



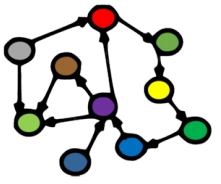
SELECT: Composite properties

- *Property paths (in SPARQL 1.1)*:
- Concatenation: ***a / b*** (means “first ***a***, then ***a***'s ***b***”)
  - example:
    ?paper dct:contributor ?author .
    ?author foaf:name ?name .
  - can be written as:
    ?paper dct:contributor / foaf:name ?name .
- Inversion: ***^a*** (means “***a*** backwards”)
- Grouping: ***(...)*** (nested composite properties)
- Repetition: ***a**** (0:n), ***a+*** (1:n), ***a?*** (0:1)
- Alternative: ***a | b*** (either ***a*** or ***b***)
- Negation: ***!a*** (any other property than ***a***)



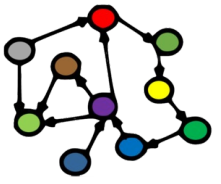
SELECT: Composite properties

- *Property paths (in SPARQL 1.1)*:
- Concatenation: ***a / b*** (first *a*, then *a*'s *b*)
- Inversion: ***^a*** (means ***a*** backwards)
  - example:
    ?paper dct:contributor ?author .
  - can be written as:
    ?author ^dct:contributor ?paper .
- Grouping: ***(...)*** (nested composite properties)
- Repetition: ***a**** (0:n), ***a+*** (1:n), ***a?*** (0:1)
- Alternative: ***a | b*** (either ***a*** or ***b***)
- Negation: ***!a*** (any other property than ***a***)



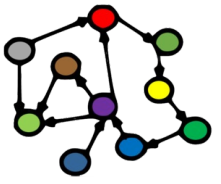
SELECT: Composite properties

- *Property paths (in SPARQL 1.1)*:
- Concatenation: ***a / b*** (first *a*, then *a*'s *b*)
- Inversion: ***^a*** (means ***a*** backwards)
  - example:
    ?author1 ^dct:contributor / dct:contributor ?author2 .
- Grouping: ***(...)*** (nested composite properties)
- Repetition: ***a**** (0:n), ***a+*** (1:n), ***a?*** (0:1)
- Alternative: ***a | b*** (either ***a*** or ***b***)
- Negation: ***!a*** (any other property than ***a***)



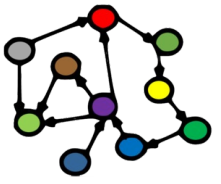
SELECT: Composite properties

- *Property paths (in SPARQL 1.1)*:
- Concatenation: ***a / b*** (first *a*, then *a*'s *b*)
- Inversion: ***^a*** (means ***a*** backwards)
- Grouping: ***(...)*** (nested composite properties)
  - example:
    ?uncle ^(:hasParent / :hasBrother) ?nephew .
- Repetition: ***a**** (0:n), ***a+*** (1:n), ***a?*** (0:1)
- Alternative: ***a | b*** (means ***a*** or ***b***)
- Negation: ***!a*** (any other predicate than a)



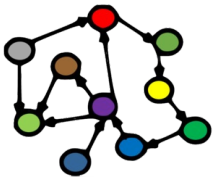
SELECT: Composite properties

- *Property paths (in SPARQL 1.1)*:
- Concatenation: ***a / b*** (first *a*, then *a*'s *b*)
- Inversion: ***^a*** (means ***a*** backwards)
- Grouping: ***(...)*** (nested composite properties)
  - example:
    ?uncle ^(:hasParent / :hasBrother) ?nephew .
    ?uncle (^:hasBrother / ^:hasParent) ?nephew .
- Repetition: ***a**** (0:n), ***a+*** (1:n), ***a?*** (0:1)
- Alternative: ***a | b*** (means ***a*** or ***b***)
- Negation: ***!a*** (any other predicate than a)



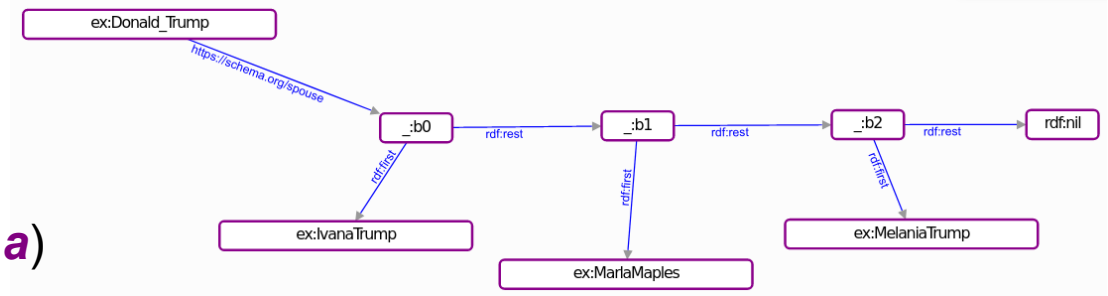
SELECT: Composite properties

- *Property paths (in SPARQL 1.1)*:
- Concatenation: ***a / b*** (means “first ***a***, then ***a***'s ***b***”)
- Inversion: ***^a*** (means “***a*** backwards”)
- Grouping: ***(...)*** (nested composite properties)
- Repetition: ***a**** (0:n), ***a+*** (1:n), ***a?*** (0:1)
  - example:
    ?paper1 ss:cites? ?paper2 .
    ?paper1 ss:cites+ ?paper2 .
    ?paper1 ss:cites* ?paper2 .
- Alternative: ***a | b*** (either ***a*** or ***b***)
- Negation: ***!a*** (any other property than ***a***)



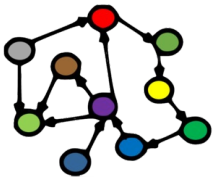
SELECT: Composite properties

- *Property paths (in SPARQL 1.1)*:
- Concatenation: ***a / b*** (means “first ***a***, then ***a***'s ***b***”)
- Inversion: ***^a*** (means “***a*** backwards”)
- Grouping: ***(...)*** (nested composite properties)
- Repetition: ***a**** (0:n), ***a+*** (1:n), ***a?*** (0:1)
  - example:
    ?list rdf:rest* / rdf:first ?member .
- Alternative: ***a | b*** (either ***a*** or ***b***)
- Negation: ***!a*** (any other property than ***a***)



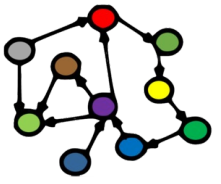
SELECT: Composite properties

- *Property paths (in SPARQL 1.1)*:
- Concatenation: ***a / b*** (first *a*, then *a*'s *b*)
- Inversion: ***^a*** (means ***a*** backwards)
- Grouping: ***(...)*** (nested composite properties)
- Repetition: ***a**** (0:n), ***a+*** (1:n), ***a?*** (0:1)
- Alternative: ***a | b*** (either ***a*** or ***b***)
  - example:
    ?parent ^(:hasFather | :hasMother) ?child .
- Negation: ***!a*** (any other property than ***a***)



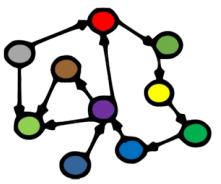
SELECT: Composite properties

- *Property paths (in SPARQL 1.1)*:
- Concatenation: ***a / b*** (first *a*, then *a*'s *b*)
- Inversion: ***^a*** (means ***a*** backwards)
- Grouping: ***(...)*** (nested composite properties)
- Repetition: ***a**** (0:n), ***a+*** (1:n), ***a?*** (0:1)
- Alternative: ***a | b*** (either ***a*** or ***b***)
  - example:
    ?parent ^(:hasFather | :hasMother) ?child .
    ?parent (^:hasFather | ^:hasMother) ?child .
- Negation: ***!a*** (any other property than ***a***)

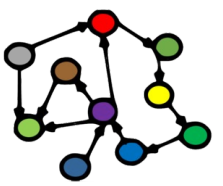


The other query types

- ASK { ... }
  - *can the pattern be matched, yes or no?*
- CONSTRUCT { *?s1 ?p1 ?o1 . ?s2 ?p2 ?o2 . ...* } WHERE { ... }
  - *returns triples*, e.g., for copying a graph:
    CONSTRUCT { ?s ?p ?o }
    WHERE { GRAPH <iri> { ?s ?p ?o } . }
- DESCRIBE ?resource
  - *returns a "relevant excerpt" of the graph for ?resource*
  - not well defined: *all triples where a resource is subject? all triples where a resource is subject or object? concise bounded descriptions (CBDs)? symmetric CBDs?*
- Most variations of SELECT can also be used for ASK/CONSTRUCT/DESCRIBE when they give meaning!



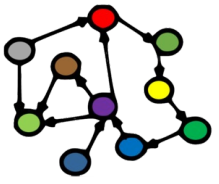
Programming SPARQL queries



SPARQL Query in RDFLib

```
from rdflib import Graph

query_str = """
    PREFIX ex: <http://example.org/>
    PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX schema: <https://schema.org/>

    SELECT ?spouse WHERE {
          ex:Donald_Trump schema:spouse / rdf:rest* / rdf:first ?spouse .
    }"""

g = Graph()
g.parse("spouses.ttl", format='ttl')
result = g.query(query_str)
for row in result:
    print("Donald's spouse: %s" % row)
```

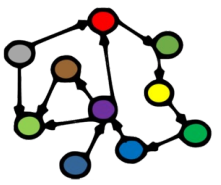
Used to query (and update) rdflib Graphs stored in memory inside a running Python program



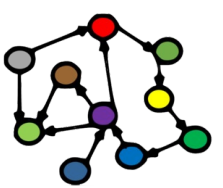
Remote SPARQL Query with SPARQLWrapper

```
from SPARQLWrapper import SPARQLWrapper


query_str = """
...
"""
endpoint = 'http://sandbox.i2s.uib.no/bigdata/namespace/s03/sparql'
client = SPARQLWrapper(endpoint)
client.setReturnFormat('json')
client.setQuery(query_str)
result = client.queryAndConvert()
for result in results["results"]["bindings"]:
    print(result["spouse"]["value"])
```

Used to query (and update) RDF graphs stored in a triple store outside a Python program



SPARQL Update

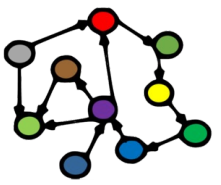


Updates: INSERT DATA

- Add new triples to the graph, e.g.:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX kg: <http://i2s.uib.no/kg4news/>
PREFIX ss: <http://semanticscholar.org/>

INSERT DATA {
    kg:paper_123 rdf:type ss:Paper ;
                ss:title "Semantic Knowledge Graphs for the News: A Review"@en ;
                kg:year 2022 ;
                dct:contributor kg:auth_456, kg:auth_789 .
}
```

- *Suitable for patching and small data sets...*

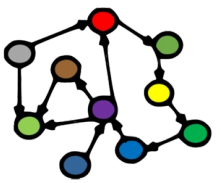


Updates: DELETE DATA

- Remove given triples, e.g.:

```
PREFIX kg: <http://i2s.uib.no/kg4news/>

DELETE DATA
{
   kg:paper_123 kg:year 2022  .
}
```

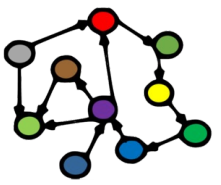
- *Suitable for patching and small data sets...*



Updates: INSERT from pattern

- Add triples according to pattern, e.g.:

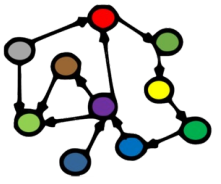
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX ss: <http://semanticscholar.org/>

INSERT {
        ?author rdf:type ss:Author .
        ?paper rdf:type ss:Paper .
} WHERE {
        ?paper dct:contributor ?author .
}
```



Updates: DELETE from pattern

- Remove triples according to pattern, e.g.:

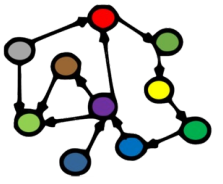
```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX ss: <http://semanticscholar.org/>

DELETE {
        ?author rdf:type ss:Author .
        ?paper rdf:type ss:Paper .
} WHERE {
        ?paper dct:contributor ?author .
}
```



Updates: DELETE/INSERT from pattern

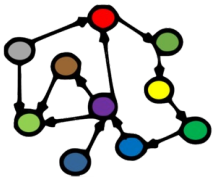
- First remove and then add triples according to patterns, e.g.:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX kg: <http://i2s.uib.no/kg4news/>
PREFIX ss: <http://semanticscholar.org/>

DELETE { ?paper dct:contributor kg:auth_456 }
INSERT { ?paper dct:contributor kg:auth_654 }
WHERE { ?paper dct:contributor kg:auth_456 }
```

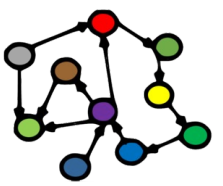


Semantic graphs and data sets

- *Graph*:
  - a collection of *triples/statements* (possibly none)
- *Data set (or "Conjunctive graph")*:
  - a collection of graphs (at least one)
  - one of the graphs is *default/unnamed*
  - the others are *named*
  - from triples/statements:
    - *(subject, predicate, object)*
  - to quadruples *(quads)*:
    - *(graph/"context", subject, predicate, object)*

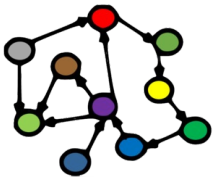


Programming SPARQL Update



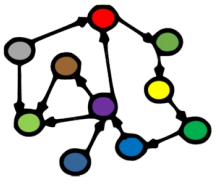
SPARQL Update in RDFLib

```
update_str = """
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dct: <http://purl.org/dc/terms/>
PREFIX kg: <http://i2s.uib.no/kg4news/>
PREFIX ss: <http://semanticscholar.org/>

INSERT DATA {
    kg:paper_123 rdf:type ss:Paper ;
            ss:title "Semantic Knowledge Graphs for the News: A Review"@en ;
          kg:year 2022 ;
          dct:contributor kg:auth_456, kg:auth_789 .
}
"""
```



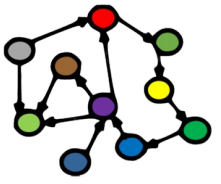
SPARQL Update in RDFLib

```
from rdflib import Graph

g = Graph()
g.update(update_str)
print(g.serialize(format='ttl'))  # format=’turtle’ also works
```



Remote SPARQL Update with SPARQLWrapper

```
from SPARQLWrapper import SPARQLWrapper

endpoint = 'http://sandbox.i2s.uib.no/bigdata/namespace/s03/sparql'
client = SPARQLWrapper(endpoint)
client.setMethod('POST')
client.setQuery(update_str)
res = client.queryAndConvert()
```



Next week:
Open KGs