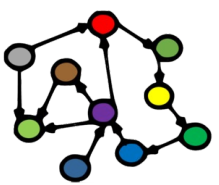
Welcome to INFO216:
Knowledge Graphs
Spring 2022

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

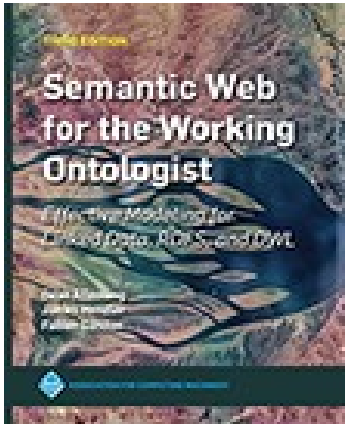
Session 4: Storing and sharing KGs

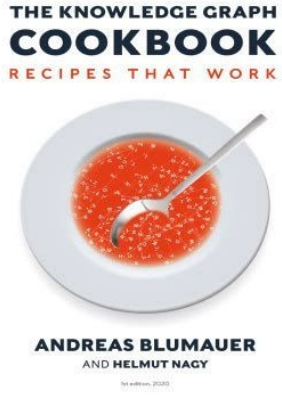
- Themes:
  - triple stores & Blazegraph
  - web APIs & JSON-LD
  - serialisation formats



Readings

- Sources:
  - Allemang, Hendler & Gandon (2020):
    Semantic Web for the Working Ontologist, 3rd edition
    - chapter 4 on application architecture
  - Blumauer & Nagy (2020):
    Knowledge Graph Cookbook – Recipes that Work
    - Part 4 (System Architecture and Technologies)
  - Blazegraph:
    - Introduction - About Blazegraph
    - Getting started
  - Section 2 in W3C's JSON-LD 1.1 Processing Algorithms and API
  - *As always: wiki.uib.no/info216*



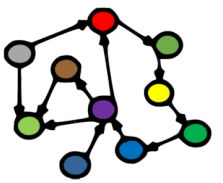


Triple stores and Blazegraph

Triple stores

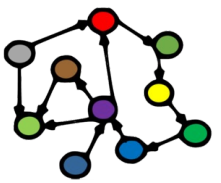
- Basic software for persistent triple stores, or
  - database management systems (DBMSs) for RDF graphs
  - general DBMS properties and behaviours
  - a specialised type of *graph database*
- Examples:
  - *Apache Jena TDB* (simple, file based, RDF-centric)
  - *Eclipse RDF4J, formerly Sesame* (much used, RDF-centric)
  - *OpenLink Virtuoso* (much used, supports multiple data models, large datasets)
  - *Blazegraph* (based on Bigdata and RDF4J)

https://www.w3.org/wiki/LargeTripleStores



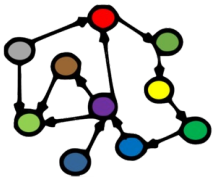
Why different triple stores?

- A few central properties:
  - license, price
  - local server or cloud-hosted
  - scaling, performance, security
  - capacity (trillions of triples (norsk: “billion”, 10^{12}))
  - support different SPARQL versions, APIs, endpoints?
  - functionality: reification, quads, inference, reasoning...
  - data model (RDF only, general graph, multi-DB)
  - technical:
    - in memory / on file / over DB
    - single- / multi-thread and -server



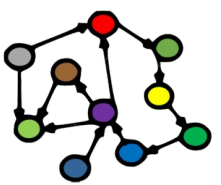
Blazegraph

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



Blazegraph

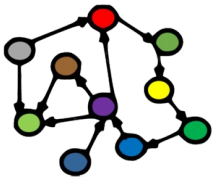
- Built around Bigdata, can be run
  - embedded in a program
  - single-machine stand-alone server (< 50B triples)
  - replicated servers (scale-up queries)
  - federated servers (> 8 machines, scale-out data)
- Dual licensing: GPLv2 and commercial
- Used by *Wikidata* and others
  - also the foundation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



Endpoints

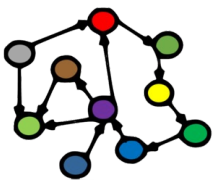
Endpoints

- Providing access to semantic data fragments over the net using standard protocols
  - often reusing simple, working standards and protocols
    - general: URL, HTTP, XML, XSD, JSON, Unicode
    - specific: RDF, JSON-LD, OWL, SPARQL...
- Provide access to
  - “native” RDF resources
    - triple stores, serialised RDF files
  - “wrapped” resources, e.g., relational or graph databases
  - *linked data fragments*



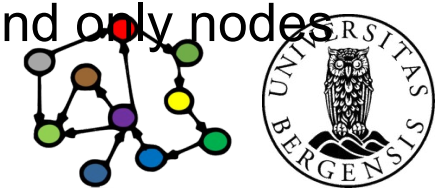
Linked data fragments

- Connected semantic data exchanged over the net using standard protocols
  - fragments of knowledge graphs
  - using standard URIs so they are easy to re-combine
- Provided through:
  - SPARQL endpoints (http://info216.i2s.uib.no)
  - dataset dumps (serialised RDF files)
  - dereferencing URIs (https://www.wikidata.org/wiki/Q13)
  - triple pattern fragments (https://linkeddatafragments.org/)
  - specialised web APIs (XML- or JSON-based)



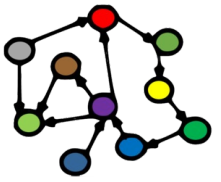
Concise bounded descriptions

- We do not want to exchange graph fragments with blank (anonymous nodes) as “leaf nodes”
- Create a *Concise Bounded Description (CBD)* for a resource R:
  1) include all triples with R as subject
  2) for all objects O in those triples that are blank/anonymous:
     a) include all new triples with O as subject (unless they are already included)
     b) apply 2) recursively to the new triples
- The resulting graph fragment has the resource R in the center and only nodes with URIs or literals as leaf nodes



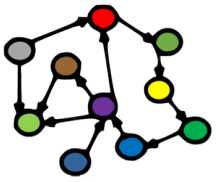
Endpoint tools and techniques

- Triples stores with SPARQL endpoints
  - may provide web interfaces for interactive use, e.g.,
    - the web-interface to *Blazegraph*
    - *SNORQL* (http://dbpedia.org/snorql/)
    - *Wikidata Query Service*
      - lots of features
      - built on top of Blazegraph
- Wrappers around other DBMS types
  - e.g., RDB2RDF



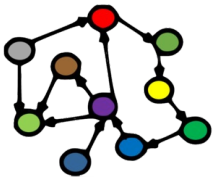
Web APIs

- *What if programs could call other programs over the web (almost) as easy as they can call methods (or sub-routines, procedures, functions)?*
  - *web APIs let us do this!*
- A Web API offers several operations or functions that
  - take inputs in a well-defined format
  - perform a well-defined operation on the inputs
  - return outputs in a well-defined format
  - SPARQL endpoints are (very specialised) examples
- Examples of functions:
  - registering student12345 to course info216 at uib
  - listing all students in the course
  - unregistering the student from the course



Three-level architecture

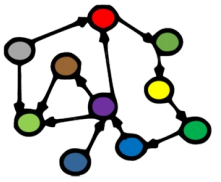
- Raw data sets:
  - available in a standard format
    - perhaps virtually
  - SPARQL end points, RDF dumps
- Abstract data representation (RDF):
  - graph of nodes and arrows
- Queries:
  - standard query languages
  - based on the abstract data representation
- *Enabled by the semantic technologies*



Other serialisation formats

Parsing/serialising

- Reading from (“parsing”) and writing to (“serialising”) standard RDF formats
- Why different formats?
  - compactness, XML-dependency
  - can the same data set be stored in many ways?
  - machine versus human readability, abbreviations
  - CURIEs (“Compact URIs”) with qname/prefix
  - nested resources
  - scope: only basic RDF or also, e.g., quads, rules , OWL...
- Built into all RDF- (and OWL-) programming frameworks
  - e.g. RDFLib, Jena



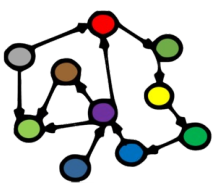
Example: N-TRIPLE

N-TRIPLE:

<http://r.e.x/Harald> <http://r.e.x/ektefelle> <http://r.e.x/Sonja> .

<http://r.e.x/Harald> <http://r.e.x/barn> <http://r.e.x/Haakon_Magnus> .

<http://r.e.x/Harald> <http://r.e.x/barn> <http://r.e.x/Martha_Louise> .



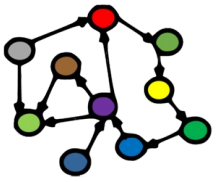
Example: N-TRIPLE

N-TRIPLE:

<http://r.e.x/Harald> <http://r.e.x/ektefelle> <http://r.e.x/Sonja> .
<http://r.e.x/Harald> <http://r.e.x/barn> <http://r.e.x/Haakon_Magnus> .
<http://r.e.x/Harald> <http://r.e.x/barn> <http://r.e.x/Martha_Louise> .

TURTLE:

<http://r.e.x/Harald> <http://r.e.x/ektefelle> <http://r.e.x/Sonja> ;
<http://r.e.x/barn> <http://r.e.x/Haakon_Magnus> ,
<http://r.e.x/Martha_Louise> .



Example: N-TRIPLE

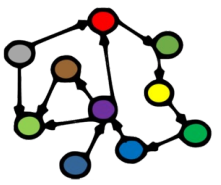
N-TRIPLE:

<http://r.e.x/Harald> <http://r.e.x/ektefelle> <http://r.e.x/Sonja> .
<http://r.e.x/Harald> <http://r.e.x/barn> <http://r.e.x/Haakon_Magnus> .
<http://r.e.x/Harald> <http://r.e.x/barn> <http://r.e.x/Martha_Louise> .

TURTLE:

<http://r.e.x/Harald> <http://r.e.x/ektefelle> <http://r.e.x/Sonja> ;
<http://r.e.x/barn> <http://r.e.x/Haakon_Magnus> ,
<http://r.e.x/Martha_Louise> .

- *semicolon (;) means "new predicate, same subject"*
- *comma (,) means "new object, same subject, predicate"*
- *period (.) means "new subject"*

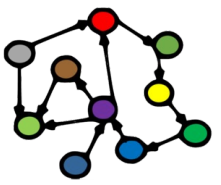


Example: TURTLE

TURTLE:

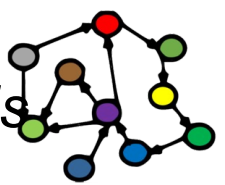
```
@prefix rex: <http://r.e.x/> .
rex:Harald  rex:spouse     rex:Sonja ;
            rex:chld       rex:Haakon_Magnus ,
                           rex:Martha_Louise .
```

- *@prefix allows use of Compact URIs ("Curies")*
- *@base allows use of URI-fragments*
- *we have looked at blank/anonymous nodes already...*



Example: TURTLE

- *"Terse RDF Triple Language"*
  - *extends the N-Triple format, restricts the Notation 3 (N3) -format*
  - *not XML-based (like RDF/XML), but simpler to read*
  - *supports prefixes (and bases)*
  - *writing multiple predicates-objects for the same subject*
  - *writing multiple objects for the same subject-predicate*
  - *flexible notations for blank/anonymous nodes: [], [...]*
  - *SPARQL uses TURTLE-like syntax*
  - *OWL is sometimes written in TURTLE*
    - *but OWL also has its own notations!*
  - *TriG extends TURTLE to support named graphs/quads*



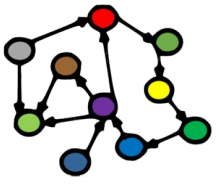
Example: TriG

TriG:

```
@prefix rex: <http://r.e.x/> .
rex:Royal   { rex:Harald   rex:spouse rex:Sonja ;
                           rex:child          rex:Haakon_Magnus ,
                                              rex:Martha_Louise .  }
rex:Mine    { reg:Andreas reg:spouse reg:Monika ;
                            reg:child         reg:Jens_Christian .  }
```

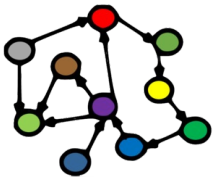
- *extends Turtle with named graphs wrapped in { ... }*
- *Similar to SPARQL, which adds the keyword:*

```
GRAPH rex:Royal   {
        rex:Harald rex:spouse rex:Sonja .
}
```



Even more RDF serialisation formats

- RDF/XML *(the original XML serialisation)*
- TriX *(XML-based, experimental, named graphs)*
- N-TRIPLE *(maximally simple format, has "canonical form")*
- NQ, NQUAD *(extends N-TRIPLE with quads)*
- TURTLE *(builds on N-TRIPLE, human readable, SPARQL ++)*
- TriG *(TURTLE-extension, named graphs)*
- Notation3, N3 *(builds on TURTLE, supports rules, graphs ++)*
- JSON-LD *("JavaScript Object Notation – Linked Data")*
- RDR *("Reification Done Right")*
- Embedded microformats, (eRDF →) RDFa, microdata
- In addition, OWL has its own serialisations...
  - RDF/XML and TURTLE are sometimes used



JSON

JSON

```
{
  "homepage": "http://me.markus-lanthaler.com",
  "name": "Markus Lanthaler",
  "workplaceHomepage": "http://www.tugraz.at/"
}
```

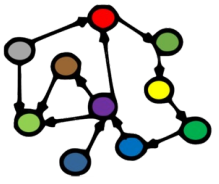
Name-Value pair

Object

Name

Value (String, Number, Boolean, null or Array)

JavaScript Object Notation (JSON)
www.json.org



JSON

```
{
  "homepage": "http://me.markus-lanthaler.com",
  "name": "Markus Lanthaler",
  "workplaceHomepage": "http://www.tugraz.at/"
}
```

Name-Value pair

Object

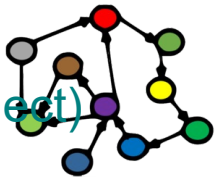
Name

Value (String, Number, Boolean, null or Array)

```
{
  "homepage": "http://me.markus-lanthaler.com",
  "name":
      {
        “firstname” : "Markus",
        “lastname” : “Lanthaler”
      },
  "workplaceHomepages": [ "http://www.tugraz.at/", “http://www.uib.no” ]
}
```

Array

Element (String, Number, Boolean, null or Object)



JSON

```
{
  "homepage": "http://me.markus-lanthaler.com",
  "name": "Markus Lanthaler",
  "workplaceHomepage": "http://www.tugraz.at/"
}
```

Name-Value pair

Object

Name

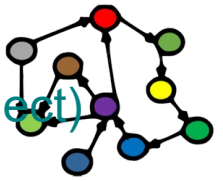
Value (String, Number, Boolean, null or Array)

Almost identical to Python dictionaries and lists!

```
{
  "homepage": "http://me.markus-lanthaler.com",
  "name":
      {
        “firstname” : "Markus",
        “lastname” : “Lanthaler”
      },
  "workplaceHomepages": [ "http://www.tugraz.at/", “http://www.uib.no” ]
}
```

Array

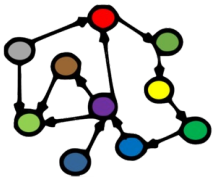
Element (String, Number, Boolean, null or Object)



JSON-LD

JSON

How to represent semantic data in JSON?



JSON-LD

```
{
  "homepage": "http://me.markus-lanthaler.com",
  "name": "Markus Lanthaler",
  "workplaceHomepage": "http://www.tugraz.at/"
}
```

This is the person's id!

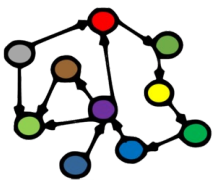
http://xmlns.com/foaf/0.1/name

http://xmlns.com/foaf/0.1/Person

http://xmlns.com/foaf/0.1/workplaceHomepage

JSON Linked Data (JSON-LD)
json-ld.org

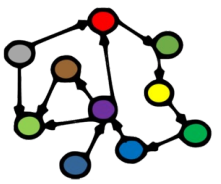
```
{
  "@id": "http://me.markus-lanthaler.com",
  “@type” : “http://xmlns.com/foaf/0.1/Person”,
  "http://xmlns.com/foaf/0.1/name": "Markus Lanthaler",
  "http://xmlns.com/foaf/0.1/workplaceHomepage":
        { “@id” : "http://www.tugraz.at/" }
}
```



JSON-LD to Turtle

JSON Linked Data (JSON-LD)
json-ld.org

```
{
  "@id": "http://me.markus-lanthaler.com",
  “@type” : “http://xmlns.com/foaf/0.1/Person”,
  "http://xmlns.com/foaf/0.1/name": "Markus Lanthaler",
  "http://xmlns.com/foaf/0.1/workplaceHomepage":
          { “@id” : "http://www.tugraz.at/" }
}
```

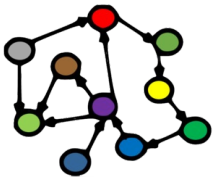


JSON-LD to Turtle

<http://me.markus-lanthaler.com>

JSON Linked Data (JSON-LD)
json-ld.org

```
{
  "@id": "http://me.markus-lanthaler.com",
  “@type” : “http://xmlns.com/foaf/0.1/Person”,
  "http://xmlns.com/foaf/0.1/name": "Markus Lanthaler",
  "http://xmlns.com/foaf/0.1/workplaceHomepage":
          { “@id” : "http://www.tugraz.at/" }
}
```

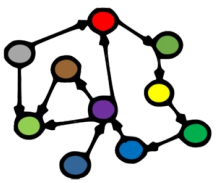


JSON-LD to Turtle

```
<http://me.markus-lanthaler.com>
        a       <http://xmlns.com/foaf/0.1/Person> ;
```

JSON Linked Data (JSON-LD)
json-ld.org

```
{
  "@id": "http://me.markus-lanthaler.com",
  “@type” : “http://xmlns.com/foaf/0.1/Person”,
  "http://xmlns.com/foaf/0.1/name": "Markus Lanthaler",
  "http://xmlns.com/foaf/0.1/workplaceHomepage":
        { “@id” : "http://www.tugraz.at/" }
}
```

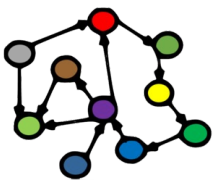


JSON-LD to Turtle

```
<http://me.markus-lanthaler.com>
        a       <http://xmlns.com/foaf/0.1/Person> ;
        <http://xmlns.com/foaf/0.1/name>
                "Markus Lanthaler";
```

JSON Linked Data (JSON-LD)
json-ld.org

```
{
  "@id": "http://me.markus-lanthaler.com",
  “@type” : “http://xmlns.com/foaf/0.1/Person”,
  "http://xmlns.com/foaf/0.1/name": "Markus Lanthaler",
  "http://xmlns.com/foaf/0.1/workplaceHomepage":
        { “@id” : "http://www.tugraz.at/" }
}
```

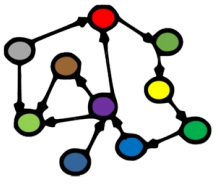


JSON-LD to Turtle

```
<http://me.markus-lanthaler.com>
        a       <http://xmlns.com/foaf/0.1/Person> ;
        <http://xmlns.com/foaf/0.1/name>
                "Markus Lanthaler”;
        <http://xmlns.com/foaf/0.1/workplaceHomepage>
                <http://www.tugraz.at/> .
```

JSON Linked Data (JSON-LD)
json-ld.org

```
{
  "@id": "http://me.markus-lanthaler.com",
  “@type” : “http://xmlns.com/foaf/0.1/Person”,
  "http://xmlns.com/foaf/0.1/name": "Markus Lanthaler",
  "http://xmlns.com/foaf/0.1/workplaceHomepage":
        { “@id” : "http://www.tugraz.at/" }
}
```

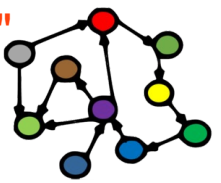


JSON-LD to Turtle

```
<http://me.markus-lanthaler.com>
        a       <http://xmlns.com/foaf/0.1/Person> ;
        <http://xmlns.com/foaf/0.1/name>
                "Markus Lanthaler";
        <http://xmlns.com/foaf/0.1/workplaceHomepage>
                "http://www.tugraz.at/" .
```

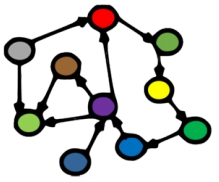
JSON Linked Data (JSON-LD)
json-ld.org

```
{
  "@id": "http://me.markus-lanthaler.com",
  "@type" : "http://xmlns.com/foaf/0.1/Person",
  "http://xmlns.com/foaf/0.1/name": "Markus Lanthaler",
  "http://xmlns.com/foaf/0.1/workplaceHomepage": "http://www.tugraz.at/"
}
```



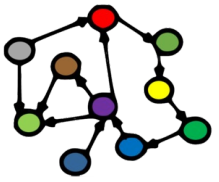
Some reserved keys in JSON-LD

- @id: signifies that the JSON object with the @id key is identified by a particular URI
- @type: signifies that the JSON object with the @type key has a particular RDF type (or several types)
- @value: signifies that a value is a literal
- @context: signifies a JSON object that contains the context (or semantic mapping) for the other objects in the same JSON array
- @base, @graph, @language, @vocab, ...



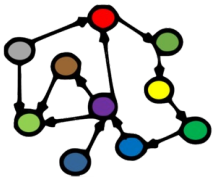
JSON-LD context

```
{
  "@context": {
    "name": "http://xmlns.com/foaf/0.1/name",
    "homepage": {
      "@id": "http://xmlns.com/foaf/0.1/homepage",
      "@type": "@id"
    }
  },
  "@id": "http://me.markus-lanthaler.com/",
  "name": "Markus Lanthaler",
  "homepage": "http://www.markus-lanthaler.com/"
}
```



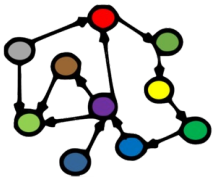
Another JSON-LD context

```
{
  "@context": {
    "website": "http://xmlns.com/foaf/0.1/homepage"
  },
  "@id": "http://me.markus-lanthaler.com/",
  "http://xmlns.com/foaf/0.1/name": "Markus Lanthaler",
  "website": { "@id": "http://www.markus-lanthaler.com/" }
}
```



JSON-LD forms

- The same graph can be expressed in different ways:
  - *expansion* removes context by pushing semantics out into the objects
    - also does regularisation
  - *compaction* simplifies the objects by pulling semantics back into the context
  - *flattening* creates a normalised form for easier parsing by computer
- Regularised and normalised forms are easier to program than “free” JSON-LD because they have a more consistent structure



Next week:
Open KGs