INFO216:
Knowledge Graphs

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

(c) Andreas L. Opdahl, 2021

Session 4: Tools and services

- Themes:
  - tools and services for KGs
    - ...or components and techniques for web-of-data applications
  - triple stores & Blazegraph
  - endpoints & Wikidata Query Service (WDQS)
  - web APIs & JSON-LD
  - serialisation formats

Readings

- Sources:
  - Blumauer & Nagy (2020):
    Knowledge Graph Cookbook – Recipes that Work
    - Part 4 (System Architecture and Technologies)
  - Allemang & Hendler (2011):
    Semantic Web for the Working Ontologist,
    - chapter 4 on application architecture
  - materials at wiki.uib.no/info216

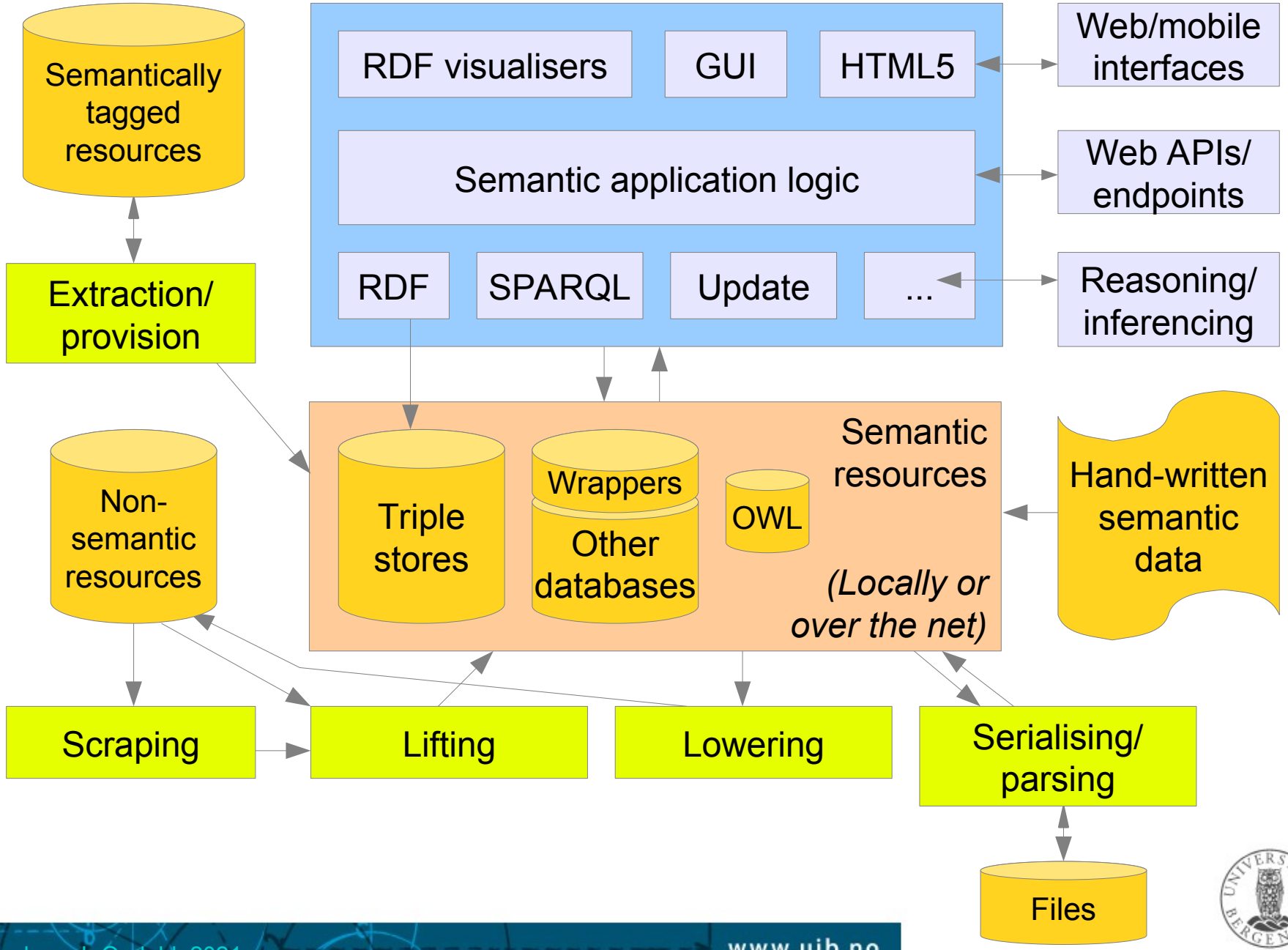
Expectations to the project talks

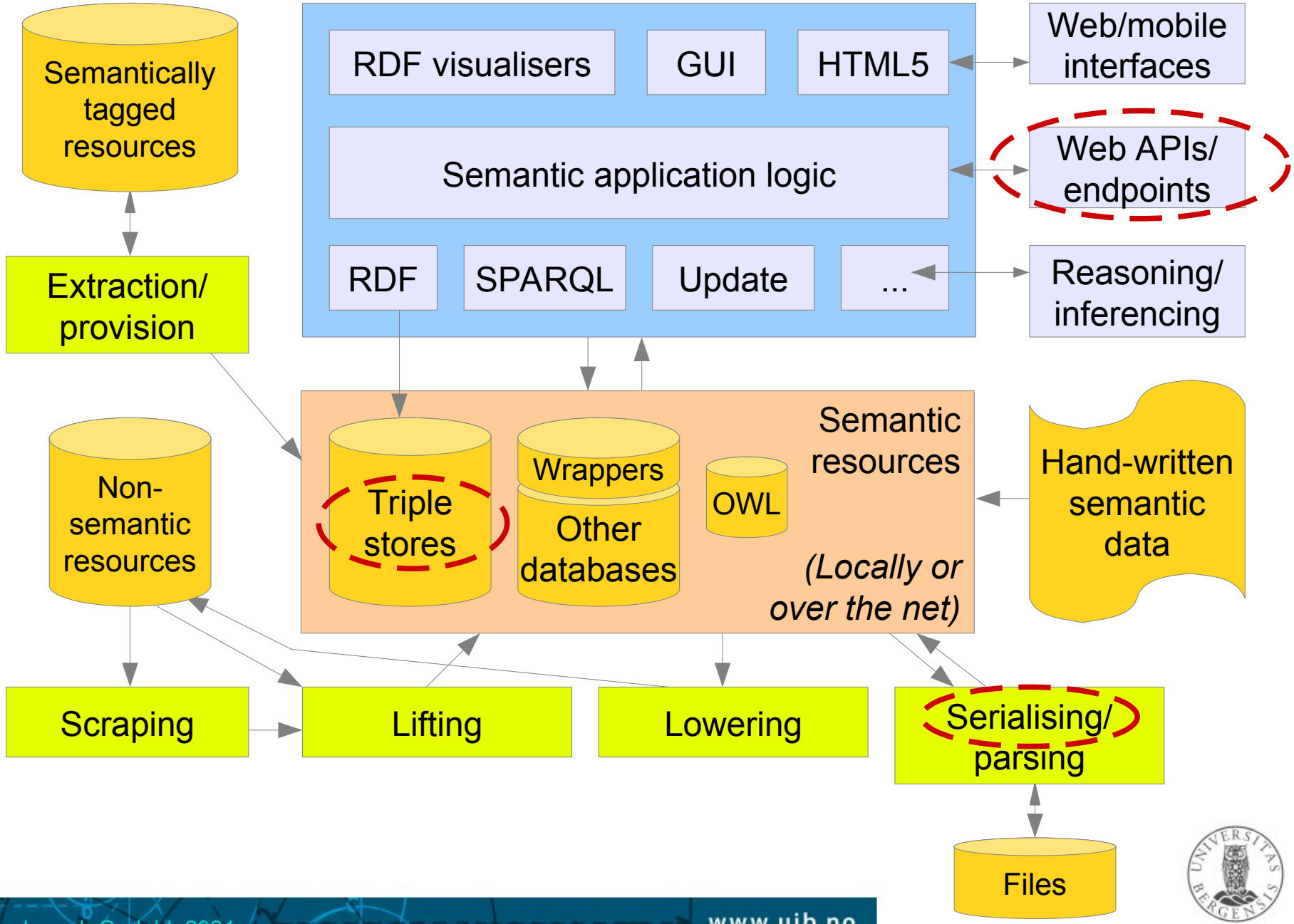
- Week 7, February 15th-19th
- You must have
  - formed a group
  - decided on one or more candidate theme
- You will
  - talk to Markus and Sindre during your regular lab
  - then you will talk with with Andreas
- Which type of data sets do you plan to use?
- Do you know of vocabularies you can use?
- What will you use them for?

Places to start

- Open and semantic:
  - open semantic data sets: *http://lod-cloud.net*
  - vocabularies: *https://lov.linkeddata.es/dataset/lov/*
- Open data in general:
  - internationally: *http://datahub.io* or *http://ckan.org*
  - Norge: *http://data.norge.no*
  - EU: *https://open-data.europa.eu*
  - UK: *http://data.gov.uk*
  - USA: *http://data.gov*

Application architecture for the Web of Data


Semantically tagged resources
RDF visualisers
GUI
HTML5
Web/mobile interfaces
Semantic application logic
Web APIs/ endpoints
RDF
SPARQL
Update
...
Reasoning/ inferencing
Extraction/ provision
Non-semantic resources
Triple stores
Wrappers
Other databases
OWL
Semantic resources
(Locally or over the net)
Hand-written semantic data
Scraping
Lifting
Lowering
Serialising/ parsing
Files


Semantically tagged resources
RDF visualisers
GUI
HTML5
Web/mobile interfaces
Semantic application logic
Web APIs/ endpoints
RDF
SPARQL
Update
...
Reasoning/ inferencing
Extraction/ provision
Non-semantic resources
Triple stores
Wrappers
Other databases
OWL
Semantic resources
(Locally or over the net)
Hand-written semantic data
Scraping
Lifting
Lowering
Serialising/ parsing
Files

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 (Sesame)* (much used, RDF-centric)
  - *Ontotext GraphDB (OWLIM)* (RDF4J compatible)
  - *Stardog* (RDF4J compatible)
  - *OpenLink Virtuoso* (much used, supports multiple data models, large datasets)
  - *Blazegraph* (formerly called Bigdata, also RDF4J based)...

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}))
  - SPARQL version, other 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, incl. 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

Linked Open Data

- 3-4 basic principles (Berners-Lee 2006):

1. URIs (Uniform Resource Identifier) *identify resources*
   - http://dbpedia.org/resource/Bergen
2. URIs *answer to HTTP requests* (*dereferencing*)
   - for example *SPARQL queries, Turtle files, ...*
3. Returns *information about the resource* on standard format, e.g.,
   - *RDF-XML, Turtle, N3, JSON-LD (JSON, XML, CSV, TSV, HTML)*
4. The information contains URI-s that *identify related resources*

The LOD cloud...

- *http://www.lod-cloud.net/*
  - statistics at www.lod-cloud/net/state
  - ca 1250 data sources (LOD-cloud, 2019)
  - based on data from DataHub (+ some crawling)
    - datahub.io or ckan.org
    - an open data portal
    - not necessarily semantic
  - ...also based on LOD crawling

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

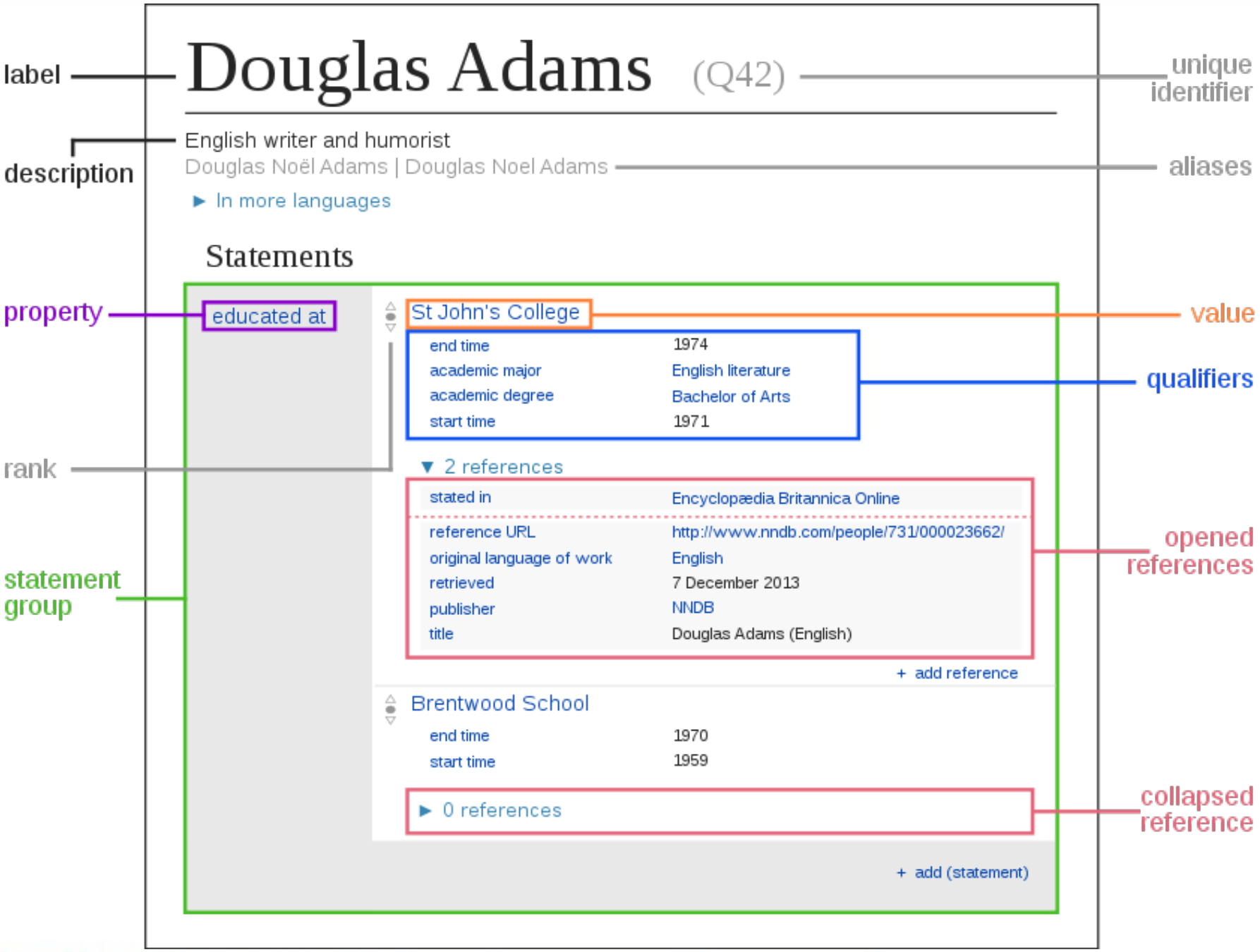
Endpoint tools and techniques

- 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

Wikidata and its Query Service (WDQS)

Wikidata

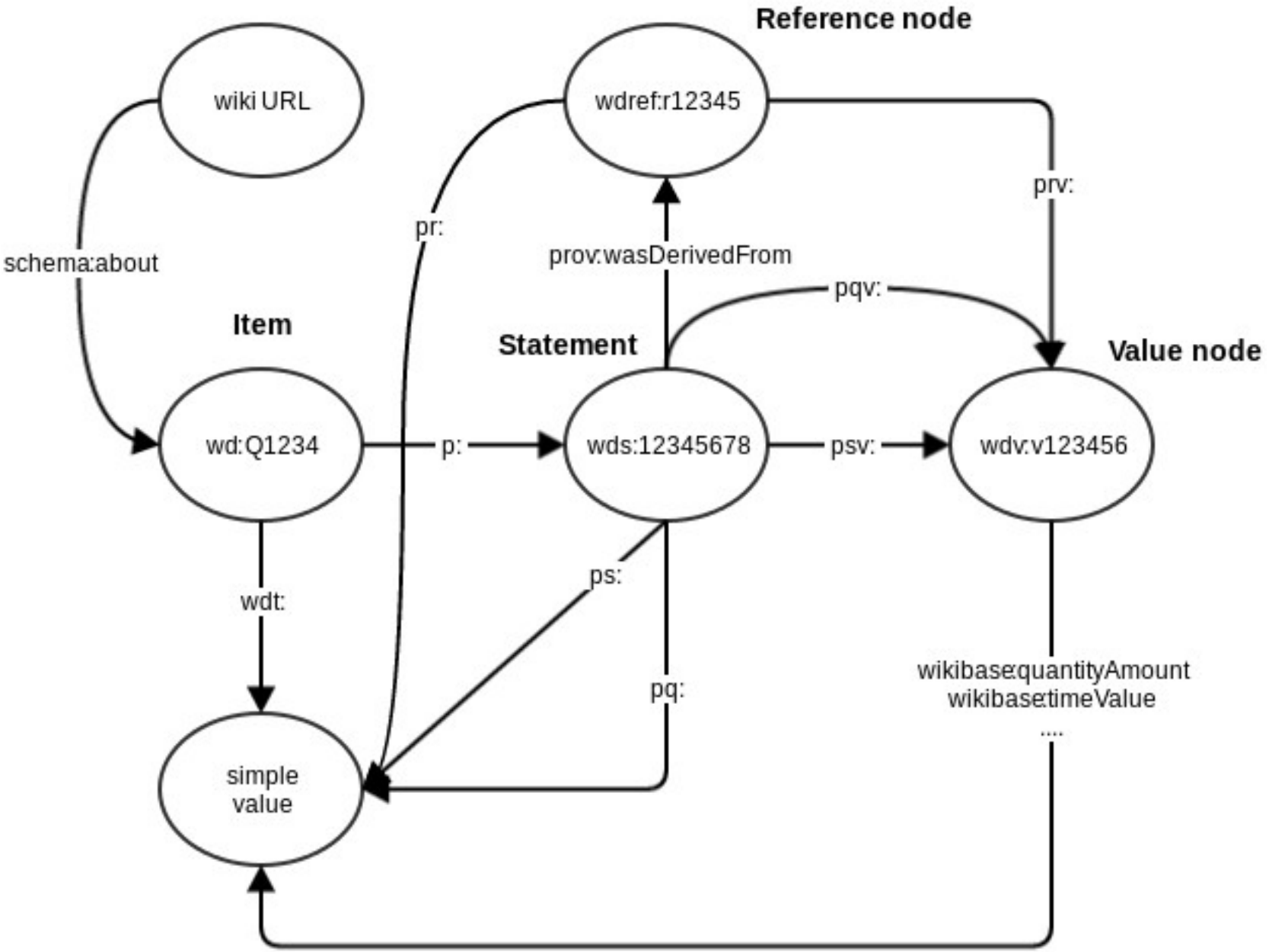
- *A free and open knowledge base that can be read and edited by both humans and machines*
  - a Wikimedia project, crowdsourced, multi-lingual
  - *a Wikipedia for structured, secondary data*
  - verifiability, link to sources, perspectives
  - central storage of the structured data for other Wikimedia projects (Wikipedia and friends)
  - supports many other sites and services
  - free license (CC0), standard formats, interlinked
- Wikidata entities:
  - > 90M items (things)



Wikidata item structure / central concepts

Wikidata item structure

- Items:
  - item identifier (Q*nn*)
  - fingerprint:
    - multilingual label, description, aliases
  - statements, each:
    - claim: a property-value pair
    - qualifiers: additional property-value pairs *about the claim*
  - references (one or more property-value pairs)
  - rank
- Site links
- *Similar structure for properties!*


Reference node
wiki URL
wdref:r12345
prv:
pr:
schema:about
prov:wasDerivedFrom
pqv:
Item
Statement
Value node
wd:Q1234
p:
wds:12345678
psv:
wdv:v123456
wdt:
ps:
pq:
wikibase:quantityAmount
wikibase:timeValue
....
simple
value

```
PREFIX wikibase: <http://wikiba.se/ontology#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>

#defaultView:BubbleChart

SELECT ?cLabel ?p WHERE {
    ?c wdt:P31 wd:Q6256 .
    ?c wdt:P30 wd:Q46 .
    ?c wdt:P1082 ?p .
    SERVICE wikibase:label {
        bd:serviceParam wikibase:language "en" .
    }
}
```

Wikidata access

- Available through
  - the WikiMedia API
  - HTTP: http://www.wikidata.org/entity/Q42
  - RDF: http://www.wikidata.org/entity/Q42.ttl
  - SPARQL endpoint: http://query.wikidata.org
    - Wikidata Query Service (WDQS)
  - for download (JSON, RDF, XML)
  - Linked Data Fragments (https://query.wikidata.org/bigdata/ldf)

Wikidata Query Service (WDQS)

- SPARQL wrapper for Wikidata (http://query.wikidata.org)
  - based on Blazegraph, OpenRDF → RDF4J
  - built-in prefixes
  - generate query URIs
  - various entity/ontology explorers, e.g.,
    - SQID (https://tools.wmflabs.org/sqid/#/)
  - GraphBuilder
  - built-in visualisations
  - built-in SERVICEs (wikibase:label)

WDQS visualisations

- Use a comment: #defaultView:viewName
- Supported viewNames:
  - Table - default view, displays the results as a table
  - Map - displays coordinate points if present
  - ImageGrid - displays result images as a grid
  - BubbleChart - displays numbers as bubble chart
  - TreeMap - displays hierarchical tree map for numbers
  - Timeline - displays timeline for results having dates
  - Dimensions - displays rows as lines between points
  - Graph - displays result as a connected graph
- (More limited) server-side alternative to Sgvizler

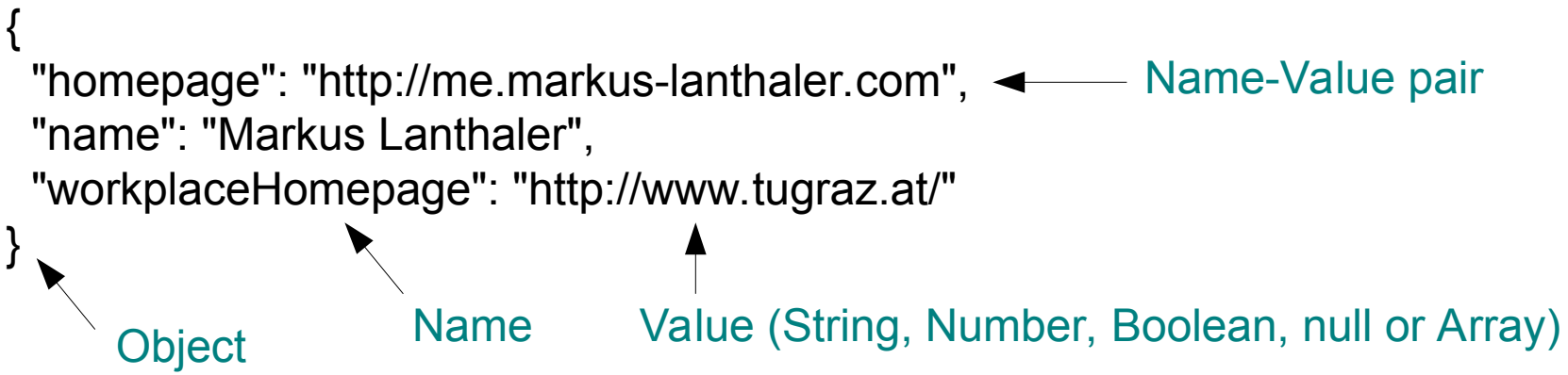
Web APIs
(RESTful web services)

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

JSON

JSON



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)

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

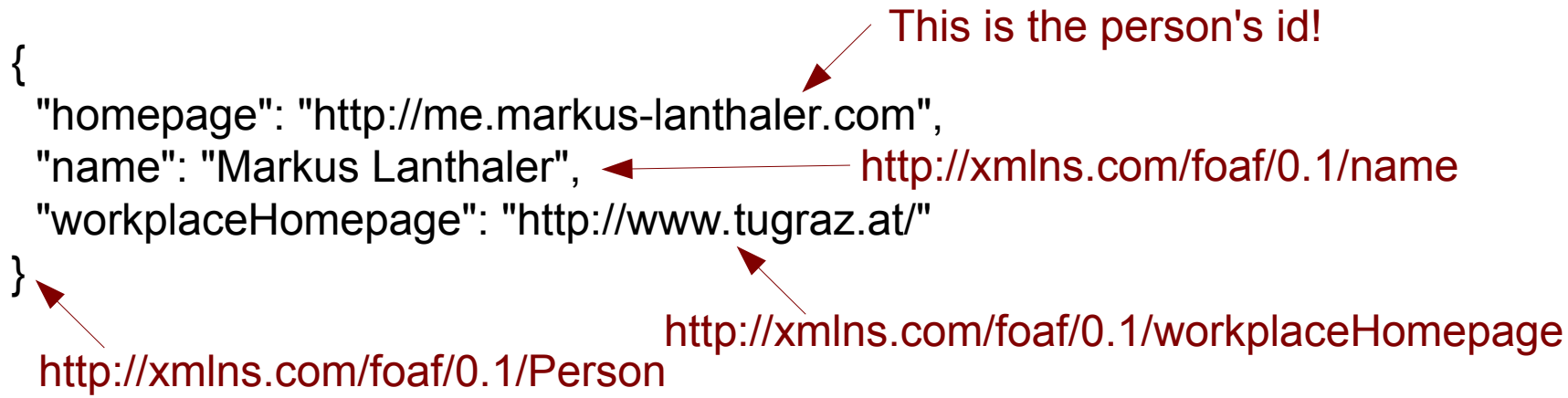
Almost identical to Python dictionaries and lists!

Array

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

JSON-LD

JSON



How to represent semantic data in JSON?

JSON-LD


```
{
  "@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 Linked Data (JSON-LD)
json-ld.org

JSON-LD to Turtle

```
{
  "@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 Linked Data (JSON-LD)
json-ld.org

JSON-LD to Turtle

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

```
{
  "@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 Linked Data (JSON-LD)
json-ld.org

JSON-LD to Turtle

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

```
{
  "@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 Linked Data (JSON-LD)
json-ld.org

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";
```

```
{
  "@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 Linked Data (JSON-LD)
json-ld.org

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/> .
```

```
{
  "@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 Linked Data (JSON-LD)
json-ld.org

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/" .
```

```
{
  "@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/"
}
```

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

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

Programming JSON-LD

- Must first pip install rdflib-jsonld
- Afterwards, JSON-LD becomes a serialisation format:

```
import rdflib
...
g = rdflib.Graph()
g.parse(data=example_string, format='json-ld')
...
g.serialize(destination=file_name_str, format='json-ld')
…
json_str = g.serialize(format='json-ld').decode()
```

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> **.**

- *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 .  }
```

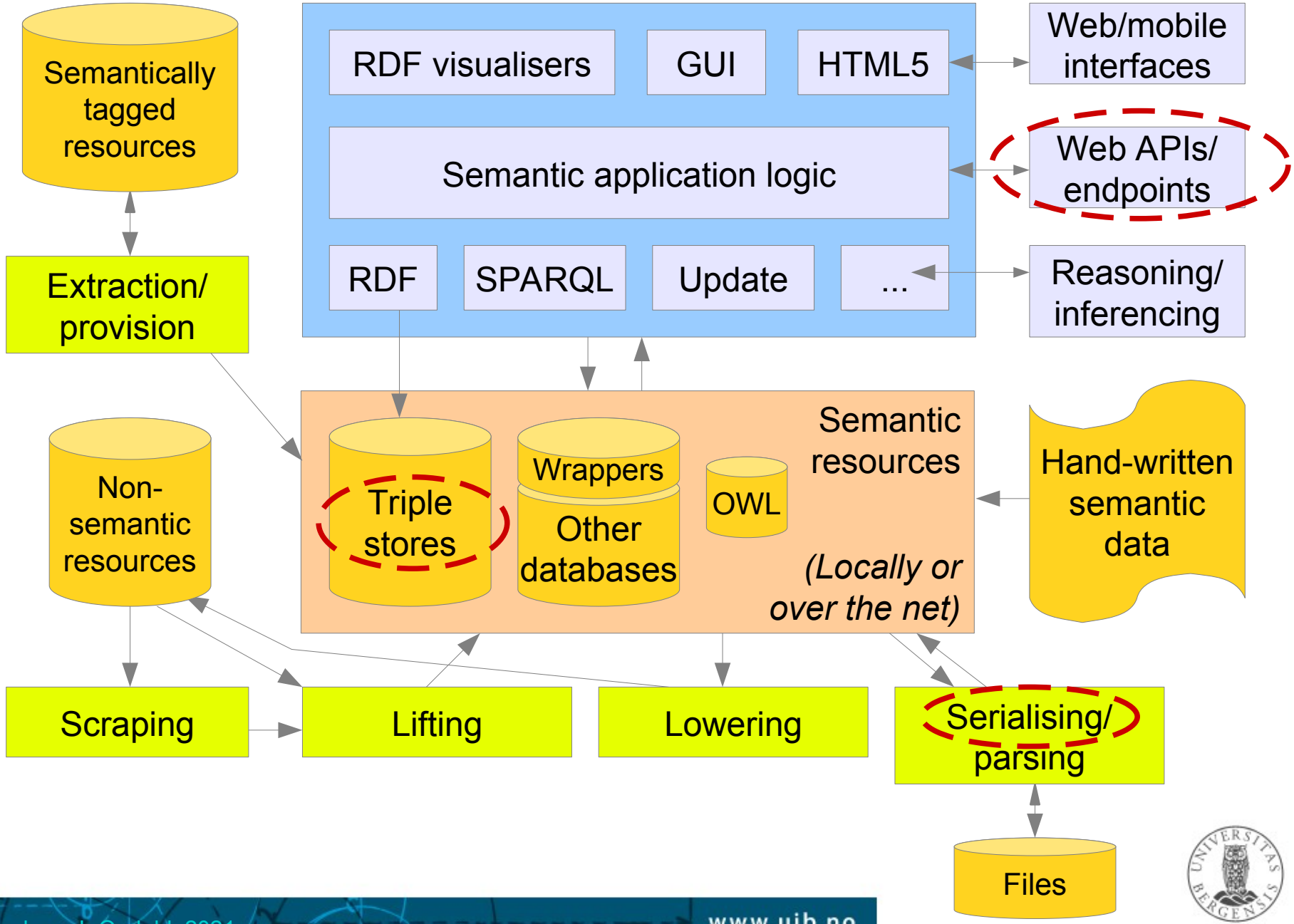
- *extends Turtle with named graphs wrapped in { ... }*
- *Similar to SPARQL, which adds the keyword:*

```
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 ("Terse RDF Triple Language")
  (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

Other components and techniques


Semantically tagged resources
RDF visualisers
GUI
HTML5
Web/mobile interfaces
Semantic application logic
Web APIs/ endpoints
RDF
SPARQL
Update
...
Reasoning/ inferencing
Extraction/ provision
Non-semantic resources
Triple stores
Wrappers
Other databases
OWL
Semantic resources
(Locally or over the net)
Hand-written semantic data
Scraping
Lifting
Lowering
Serialising/ parsing
Files

Extraction

- Retrieving RDF triples from (semantically) tagged resources
  - e.g., microformats, (eRDF ->) RDFa, Microdata
  - RDFLib parses RDFa and Microdata
- Replaces scraping + lifting
  - but is much simpler
    - the tags already do much of the job
  - open-source code is often available

Scraping

- Making less structured data locally available in a well-structured format
- Typically used on internet data:
  - from less to more explicitly structured formats
  - HTML, PDF, DOCX, TXT, tagged file formats
- Storing the result in, e.g., CSV, XML or JSON
- A useful technical skill
  - but not our focus
  - many APIs available
    - BeautifulSoup, Selenium
  - scripting, regular expressions
  - *think continuous process – not once-off conversion!*

Semantic lifting

- Making structured data semantic
  - ...important for us
- Often the next step after scraping
  - ...or in parallel with scraping
  - storing the result in, e.g., RDF, RDFS, OWL...
- Tasks:
  1. creating triples *(make everything (s, p, o)-triples)*
  2. creating graphs *(one or several?)*
  3. selecting URIs *(standard URIs as identifiers)*
  4. selecting vocabularies *(standard URIs as predicates)*
  5. selecting types *(standard URIs as resource types)*
  6. external linking *(owl:sameAs)*

Visualisation

- APIs:
  - general GUI APIs
  - graph drawing/editing APIs
- Cloud based:
  - graph and general visualisers
  - e.g., embedded in web pages
  - often SPARQL-based
    - a SPARQL query extracts the dataset
    - the SELECTed variables are used to draw
      - graphs, bar charts, pie charts...
    - e.g., *http://mgskjaeveland.github.io/sgvizler/*

Wrappers

- Wrapping existing structured data resources to present them as semantic resources
  - often relational data
    - but also, e.g., spreadsheets, XML, JSON
  - on-demand (live) semantic lifting
    - attributes/columns are mapped to predicates
  - read-only or read+update?
  - handwritten or wrapper software
    - e.g., RDB2RDF (https://rdb2rdf.org/)
  - wrapped resources can be used locally
    - or made accessible through an endpoint

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*