Lecture 2: RDF data Serialization and Storing

TIES4520 Semantic Technologies for Developers
Autumn 2023

University of Jyväskylä

Khriyenko Oleksiy

Part 1

RDF data Serialization

Serialization

- The way of representing the graph in textual form
- Serializations (notations):
  - ***RDF/XML***
  - ***TriX***
  - ***N-triples***
  - ***Turtle*** *(Terse RDF Triple Language)*
  - ***Notation 3***
  - ***…***

RDF/XML

- Suitable for machines
- Many XML parsers exist
- Difficult for humans to see *subject*-*predicate*-*object* triples

```
<rdf:RDF
  xmlns="http://data.gov/ontology/edu#"
  xmlns:log="http://www.w3.org/2000/10/swap/log#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description rdf:about="http://www.jyu.fi/courses/TIES4520">
        <credits>5</credits>
    </rdf:Description>
    <rdf:Description rdf:about="http://www.jyu.fi/people/Mary">
        <studies rdf:resource="http://www.jyu.fi/courses/TIES4520"/>
        <livesIn xmlns="http://data.gov/ontology/urban#"
                 rdf:resource="http://www.geo.com/city/Turku"/>
    </rdf:Description>
</rdf:RDF>
```

TriX notation (1)

- Another way of writing RDF in XML
- Contains named graphs
  - Named graphs contain triples
- Syntactically extendable
- More info: (*http://www.hpl.hp.com/techreports/2004/HPL-2004-56.html*)

- Why is RDF/XML not good enough?
  - RDF embedded in XHTML and other XML documents is hard to validate + other validation problems
  - Some unresolved syntactic issues (blank nodes as predicates, reification, literals as subjects, etc.)

TriX notation (2)

```
<TriX xmlns="http://www.w3.org/2004/03/trix/trix-1/">
  <graph>
    <uri>http://example.org/graph1</uri>
    <triple>
      <uri>http://example.org/John</uri>
      <uri>http://example.org/loves</uri>
      <uri>http://example.org/Mary</uri>
    </triple>
    <triple>
      <uri>http://example.org/John</uri>
      <uri>http://example.org/hasName</uri>
      <plainLiteral>John</plainLiteral>
    </triple>
    <triple>
      <uri>http://example.org/John</uri>
      <uri>http://example.org/hasAge</uri>
      <typedLiteral datatype="http://www.w3.org/2001/XMLSchema#integer">
        32
      </typedLiteral>
    </triple>
  </graph>
</TriX>
```

Graph URI optional

S

P

O

N-triples

- Simple textual serialization of RDF statements
- Each statement consists of *subject*, *predicate* and *object* separated by a white space
- Statements are separated by dots (.)
- Resources are referred to with full URIs in **<>** brackets
- Literals are wrapped into double quotes (“ ”)

```
<http://www.jyu.fi/people/Mary> <http://data.gov/ontology/urban#livesIn>
   <http://www.geo.com/city/Turku> .
<http://www.jyu.fi/people/Mary> <http://data.gov/ontology/edu#studies>
   <http://www.jyu.fi/courses/TIES4520> .
<http://www.jyu.fi/courses/TIES4520> <http://data.gov/ontology/edu#credits>
   "5" .
```

Turtle (1)

- ***Turtle*** is similar to N-triples, but even more compact
- Uses ***@prefix*** to define the prefix to further use just qualified names (e.g. *fam:John*)

```
@prefix   p: <http://www.jyu.fi/people/> .
@prefix   u: <http://data.gov/ontology/urban#> .

p:Mary u:hasAge "25" .
```

- Abbreviated form of triples with ***semantic sugar***:
  - Semicolon (**;**) to separate statements about the same subject

```
x:Mary x:hasAge "25"; x:gender x:female; x:likes x:chocolate .
```

  - Comma (**,**) to separate statements about the same subject with the same predicate

```
x:Mary x:likes x:chocolate, x:cheese, x:bread .
```

- And more features: *http://www.w3.org/TeamSubmission/turtle/*
  http://www.w3.org/TR/turtle/

Turtle (2)

- Same example as before:


```
<http://www.jyu.fi/people/Mary> <http://data.gov/ontology/urban#livesIn>
   <http://www.geo.com/city/Turku> .
<http://www.jyu.fi/people/Mary> <http://data.gov/ontology/edu#studies>
   <http://www.jyu.fi/courses/TIES4520> .
<http://www.jyu.fi/courses/TIES4520> <http://data.gov/ontology/edu#credits>
   "5" .
```

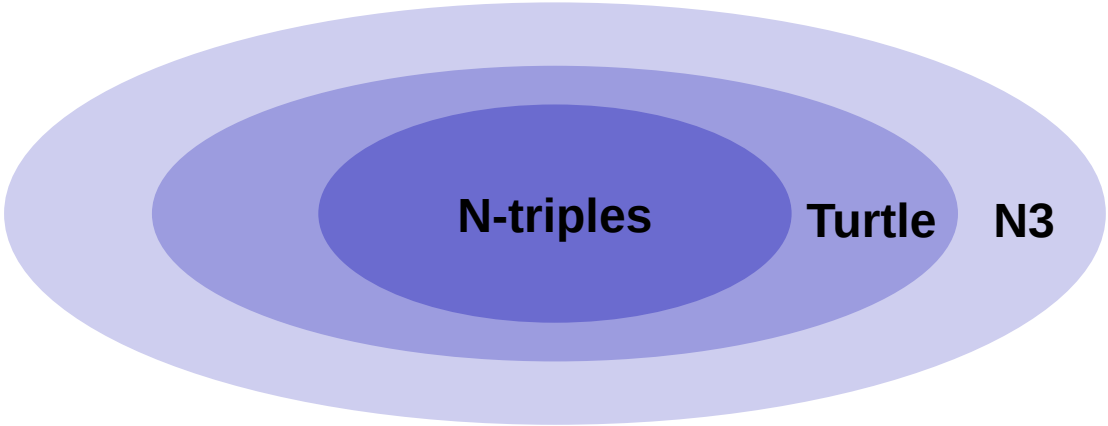

```
@prefix   p: <http://www.jyu.fi/people/> .
@prefix   u: <http://data.gov/ontology/urban#> .
@prefix edu: <http://data.gov/ontology/edu#> .
@prefix  co: <http://www.jyu.fi/courses/> .
@prefix  ci: <http://www.geo.com/city/> .

p:Mary u:livesIn ci:Turku .
p:Mary edu:studies co:TIES4520 .
co:TIES4520 edu:credits "5" .
```

```
p:Mary u:livesIn ci:Turku ; edu:studies co:TIES4520 .
co:TIES4520 edu:credits "5" .
```

Notation 3

- ***Notation 3*** (superset of Turtle) is the most important RDF notation, most clearly captures the abstract graph.
- Introduces formula *{ … }*
  - E.g. to make statements about statements
- New logic-related predicates (e.g. *=>*, *@forSome*, etc.)
- Variables allowed (*?varName*)
- Suffix: **.n3*
- Relationship to other notations:



Three rules of RDF

1. Every fact is expressed as a triple (subject, predicate, object)
2. Subjects, predicates and objects are names for entities represented as URIs
3. Objects can be literal values as well (***subjects and predicates are not!!!***)

```
<urn:isbn:1-12-456789-0>  <http://www.jyu.fi/library#hasAuthor>  "John" .
```

Subject *(URI)*

Predicate *(URI)*

Object *(URI or Literal value)*

```
c:TIES4520 co:hasLecture lec:Lecture1 .
c:TIES4520 co:hasCode "TIES4520" .
p:per673 fam:hasSurname "Doe" .
"John" fam:hasAge "25" .
```

Literals: Language Tags

Literal values are raw text that can be used instead of objects in RDF triples.

- *Plain literal* represents string:
  - *language tag*, to specify what language the raw text is written in;

```
:jone :name   "John" .
```
*name is a language-less, datatype-less **raw text value**.*

```
:jone :name   "John"@en .
```
*name **in English**, is John.*

```
:jone :name   "Jacque"@fr .
```
*name **in French**, is Jacque.*

```
:a :b "The first line\nThe second line\n more" .
:a :b """The first line
      The second line
      more""" .
```
”simple literal”.

“””long literal”””.

Literals: Datatype

Literal values are raw text that can be used instead of objects in RDF triples.

- ***Datatype literal*** represents non-string values (e.g. boolean, numbers, date or time, etc.) with *datatype* that indicates how to interpret the raw text.
  - Lexical form of the literal + URI of the datatype (datatypes defined in *XML Schema* are used by convention: *xsd:integer, xsd:decimal, xsd:float, xsd:double, xsd:string, xsd:boolean, xsd:dateTime*, etc.).

	Datatype	Value space (informative)
Core types	xsd:string	Character strings
	xsd:boolean	true, false
	xsd:decimal	Arbitrary-precision decimal numbers
	xsd:integer	Arbitrary-size integer numbers
IEEE floating-point numbers	xsd:double	64-bit floating point numbers incl. ±Inf, ±0, NaN
	xsd:float	32-bit floating point numbers incl. ±Inf, ±0, NaN
Time and date	xsd:date	Dates (yyyy-mm-dd) with or without timezone
	xsd:time	Times (hh:mm:ss.sss…) with or without timezone
	xsd:dateTime	Date and time with or without timezone
	xsd:dateTimeStamp	Date and time with required timezone
Recurring and partial dates	xsd:gYear	Gregorian calendar year
	xsd:gMonth	Gregorian calendar month
	xsd:gDay	Gregorian calendar day of the month
	xsd:gYearMonth	Gregorian calendar year and month
	xsd:gMonthDay	Gregorian calendar month and day
	xsd:duration	Duration of time
	xsd:yearMonthDuration	Duration of time (months and years only)
	xsd:dayTimeDuration	Duration of time (days, hours, minutes, seconds only)
Limited-range integer numbers	xsd:byte	-128…+127 (8 bit)
	xsd:short	-32768…+32767 (16 bit)
	xsd:int	-2147483648…+2147483647 (32 bit)
	xsd:long	-9223372036854775808…+9223372036854775807 (64 bit)
	xsd:unsignedByte	0…255 (8 bit)
	xsd:unsignedShort	0…65535 (16 bit)
	xsd:unsignedInt	0…4294967295 (32 bit)
	xsd:unsignedLong	0…18446744073709551615 (64 bit)
	xsd:positiveInteger	Integer numbers >0
	xsd:nonNegativeInteger	Integer numbers ≥0
	xsd:negativeInteger	Integer numbers <0
	xsd:nonPositiveInteger	Integer numbers ≤0
Encoded binary data	xsd:hexBinary	Hex-encoded binary data
	xsd:base64Binary	Base64-encoded binary data
Miscellaneous XSD types	xsd:anyURI	Absolute or relative URIs and IRIs
	xsd:language	Language tags per [BCP47]
	xsd:normalizedString	Whitespace-normalized strings
	xsd:token	Tokenized strings
	xsd:NMTOKEN	XML NMTOKENs
	xsd:Name	XML Names
	xsd:NCName	XML NCNames

```
"1234" .
```
An untyped literal value. No datatype.

```
"1234"^^xsd:integer .
```
A typed literal value using a namespace.

```
"1234"^^<http://www.w3.org/2001/XMLSchema#integer> .
```
The same with the full datatype URI.

Important:

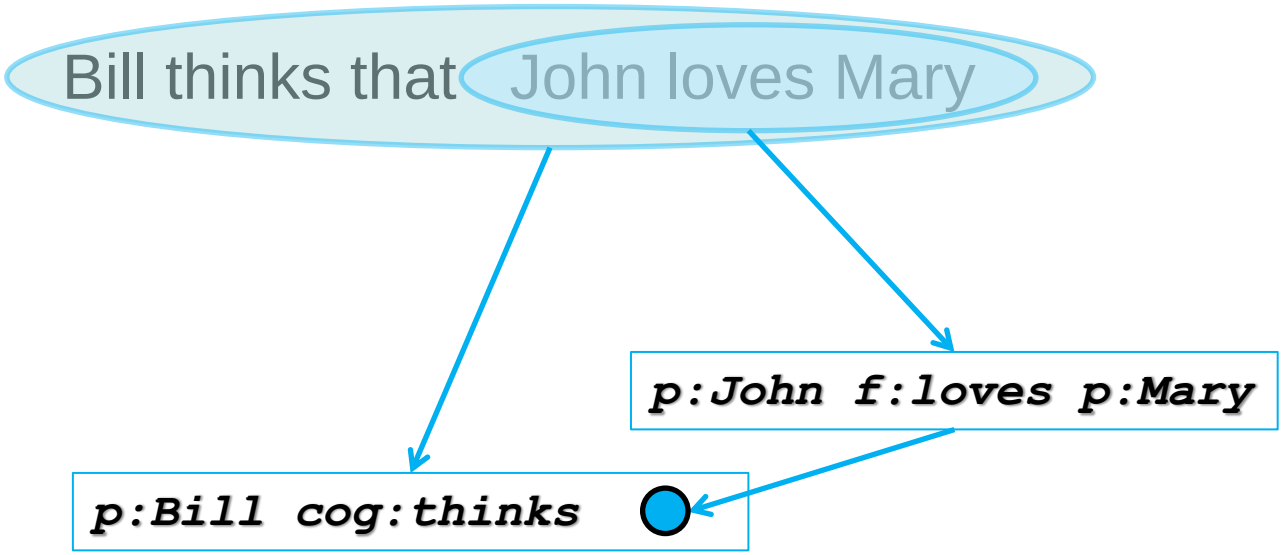
```
:jone :age "10"^^xsd:float .
```
is the ***same*** *as*
```
:jone :age "10.000"^^xsd:float .
```

```
:jone :age "10" .
```
is ***not*** *the* ***same*** *as*
```
:jone :age "10.000"^^xsd:float .
```

```
:jone :taxesPayed "true" .
```
is ***not*** *the* ***same*** *as*
```
:jone :taxesPayed "true"^^xsd:boolean .
```

Reification problem

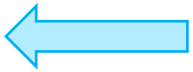
- Making statement about a statement or more statements
- Used to represent the context
- *Example:*



Solutions to reification

- Referring to the statement as a resource:

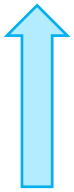
```
p:Bill cog:thinks s:st1 .
s:st1 rdf:type rdf:Statement.
s:st1 rdf:subject p:John .
s:st1 rdf:predicate f:loves .
s:st1 rdf:object p:Mary .
```



valid in Turtle
valid in Notation3

- Using special context brackets:

```
p:Bill cog:thinks {p:John f:loves p:Mary} .
```



***not** valid in Turtle*
valid in Notation3

Blank nodes

- When we want to annotate a resource, whose identity is unknown we use ***blank node***.
- Example: Larry has a fast red car.

```
p:Larry p:owns ? .
?   rdf:type v:Car; v:color c:red; v:speed s:fast .
```

Use abbreviation (***[ ]***) for anonymous blank node, if there is no need to reuse the blank node:

```
p:Larry p:owns [
   rdf:type v:Car
   ; v:color c:red
   ; v:speed s:fast .] .
```

Use a blank node identifier (***_:name***), that allows the same blank node to be referred to in more then one place:

```
p:Larry p:owns _:xe46na54an .
_:xe46na54an rdf:type v:Car
   ; v:color c:red
   ; v:speed s:fast .
```

Turtle abbreviations

- *@prefix* – abbreviation for prefix definition
- *a* – abbreviation for the property<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>

```
@prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

p:Larry <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> p:Human .
p:Larry rdf:type p:Human .
p:Larry a p:Human .
```

- *,* – is used to repeat the *subject* and the *predicate* of triples that only differ in the *object*

```
p:Larry p:owns v:Car , v:bike.
```

- *;* – is used to repeat the *subject* of triples that vary in the *predicate* and the *object*

```
p:Larry a p:Human ; p:owns v:Car .
```

- *[ ]* – abbreviation for the blank node

```
p:Larry p:owns [rdf:type v:Car ; v:color c:red ; v:speed s:fast .].
```

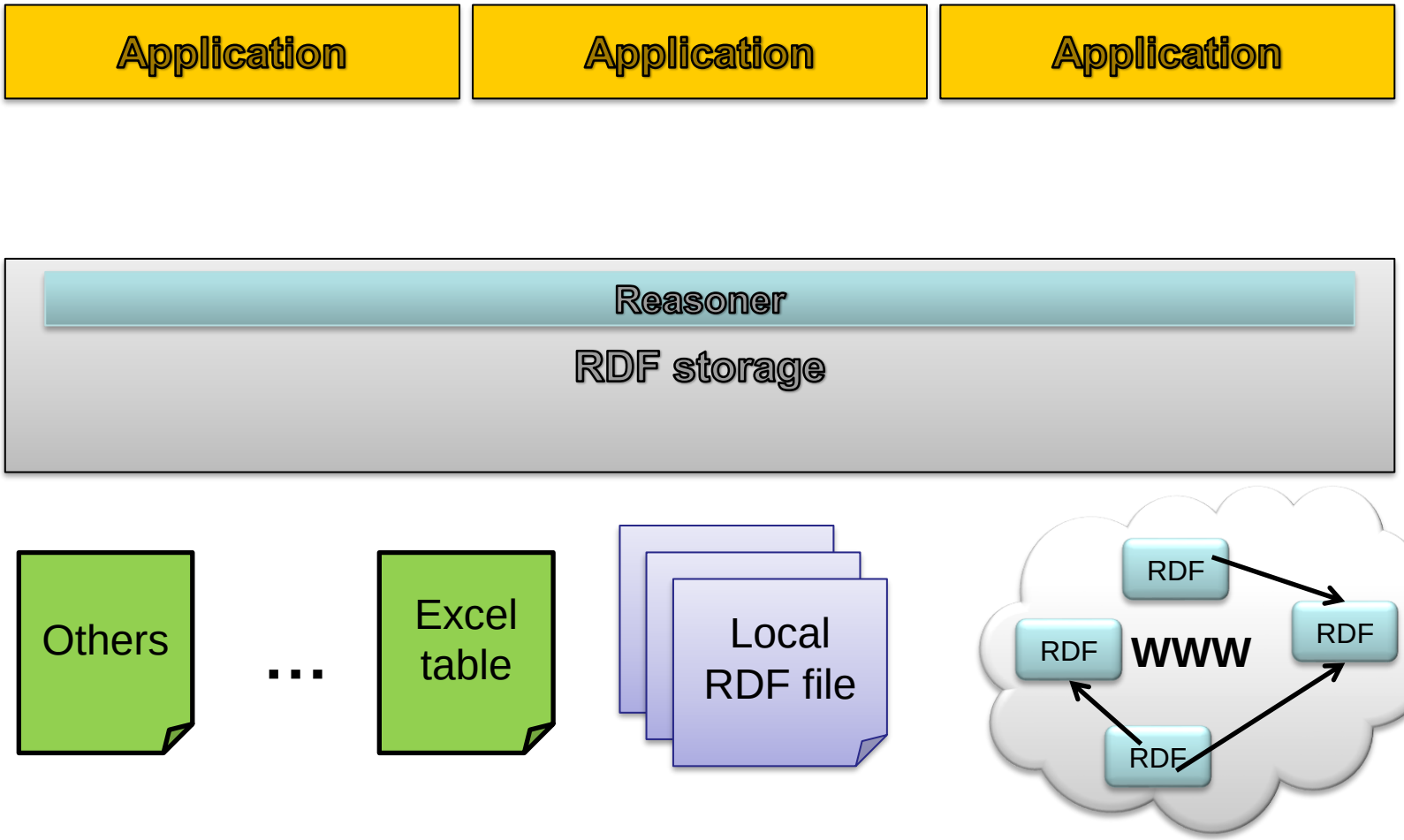
- *( )* – abbreviation for RDF Collection (structure for lists of RDF nodes)

```
cd:Beatles cd:artist ( b:George b:John b:Paul b:Ringo ).
```

Manipulation of RDF data

- Storing
  - RDF file on the web
  - Specialized RDF storage (“RDF database”)
  - Other form (*.xls, DB, …) exposed as RDF
- Querying
  - Like in relational DB there is a query language (SPARQL)
  - Can query from several sources (web sources, local RDF storages, etc.)
- Reasoning
  - Can derive new facts from already existing facts
  - Can check consistency of the model
  - Does not exist in relational DB !!!

RDF for a machine



Simple Task

- Solve small tasks about RDF and its serializations
  - Convert it from N-triple to Turtle (try to save space by using abbreviated syntax)
  - Convert it to a graph (e.g. draw on a piece of paper)

```
<http://transport.data.gov.uk/id/stop-point/9100BAYFORD>
<http://transport.data.gov.uk/def/naptan/station>
<http://transport.data.gov.uk/id/station/BAY>.
<http://transport.data.gov.uk/id/stop-point/9100BAYFORD>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://transport.data.gov.uk/def/naptan/StopPoint>.
<http://transport.data.gov.uk/id/stop-point/9100BAYFORD>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://transport.data.gov.uk/def/naptan/RailAccessArea>.
<http://transport.data.gov.uk/id/station/BAY>
<http://transport.data.gov.uk/def/naptan/crsRef> "BAY" .
<http://transport.data.gov.uk/id/station/BAY>
<http://transport.data.gov.uk/def/naptan/tiplocRef> "BAYFORD" .
<http://transport.data.gov.uk/id/station/BAY>
<http://www.w3.org/2004/02/skos/core#prefLabel> "Bayford Rail Station" .
```

Part 2

RDF data Storing

Storing of RDF

- Small datasets (few triples)
  - RDF file published on the web or stored locally
  - Examples: `*.rdf`, `*.nt`, `*.ttl`, `*.n3`, etc.
- Large datasets (thousands to millions of triples)
  - Database-bases solution better
  - Usually in form of RDF storage
- Legacy data
  - Keep in original form
  - Provide mapping to RDF
  - Expose as RDF to the outer world

Triplestore: *http://en.wikipedia.org/wiki/Triplestore*
Graph databases: *https://en.wikipedia.org/wiki/Graph_database*
List of triplestores: *http://www.w3.org/wiki/LargeTripleStores*

Native RDF Stores

RDF stores that implement their own database engine without reusing the storage and retrieval functionalities of other database management systems:

- ***AllegroGraph*** (commercial) is a commercial RDF graph database and application framework developed by Franz Inc. There are different editions of AllegroGraph and different clients: the free *RDFStore* server edition is limited to storing less than 50 million triples, a developer edition capable of storing a maximum of 600 million triples, and an enterprise edition with storage capacity only limited by the underlying server infrastructure (1+Trillion). Clients connectors are available for Java, Python, Lisp, Clojure, Ruby, Perl, C#, and Scala. Link: *https://allegrograph.com/allegrograph/*, *http://franz.com/agraph/allegrograph/*
- ***GraphDB™*** (formerly ***OWLIM***) – An Enterprise Triplestore with Meaning (GNU LGPL license and commercial) It is a family of commercial RDF storage solutions, provided by Ontotext. There are three different editions: *GraphDB™ Lite*, *GraphDB™ Standard* and *GraphDB™ Enterprise*. Link: *http://ontotext.com/products/graphdb/*
- ***Stardog*** is an enterprise data unification platform built on smart graph technology: query, search, inference, and data virtualization. Stardog is a pure Java RDF database which supports all of the OWL2 profiles using a dynamic (backward-chaining) approach. It also includes unique features such as Integrity Constraint Validation and explanations, i.e. proofs, for inferences and integrity constraint violations. It also integrates a full-text search index based on Lucene. Link: *http://stardog.com/*
- ***Apache Jena TDB*** (open-source) is a component of the Jena Semantic Web framework and available as open-source software released under the BSD license. Link: *http://jena.apache.org/*
- ***Mulgara*** (open source, Open Software License) is the community-driven successor of Kowari and is described as a purely Java-based, scalable, and transaction-safe RDF database for the storage and retrieval of RDF-based metadata. Link: *http://www.mulgara.org/*
- ***RDFox*** is a highly scalable in-memory RDF triple store that supports shared memory parallel datalog reasoning. It is a cross-platform software written in C++ that comes with a Java wrapper allowing for an easy integration with any Java-based solution. Link: *http://www.cs.ox.ac.uk/isg/tools/RDFox/*

DBMS-backed Stores

RDF Stores that use the storage and retrieval functionality provided by another database management system:

- **Apache Jena SDB** (open-source) is another component of the Jena Semantic Web framework and provides storage and query for RDF datasets using conventional relational databases: Microsoft SQL Server, Oracle 10g, IBM DB2, PostgreSQL, MySQL, HSQLDB, H2, and Apache Derby. Link: *http://jena.apache.org/*
- **Oracle Spatial and Graph: RDF Semantic Graph** (formerly **Oracle Semantic Technologies**) is a W3C standards-based, full-featured graph store in Oracle Database for Linked Data and Social Networks applications.
  Link: *https://www.oracle.com/database/technologies/spatialandgraph/rdf-graph-features.html*
- **Semantics Platform** is a family of products for building medium and large scale semantics-based applications using the Microsoft .NET framework. It provides semantic technology for the storage, services and presentation layers of an application. Link: *http://www.intellidimension.com/products/semantics-platform/*
- **ARC2** is a free, open-source semantic web framework for PHP applications released under the W3C Software License and the GNU GPL. It is designed as a PHP library and includes RDF parsers (RDF/XML, N-Triples, Turtle, SPARQL + SPOG, Legacy XML, HTML tag soup, RSS 2.0, Google Social Graph API JSON…) and serializers (N-Triples, RDF/JSON, RDF/XML, Turtle, SPOG dumps…), an RDBMS-backed (MySQL) RDF storage, and implements the SPARQL query and update specifications. Active code development has been discontinued due to lack of funds and the inability to efficiently implement the ever-growing stack of RDF specifications. Link: *https://github.com/semsol/arc2/wiki*
  More reading: *http://tinman.cs.gsu.edu/~raj/8711/sp11/presentations/ARC_RDF_Store.pdf*

  EASYRDF A PHP library designed to make it easy to consume and produce RDF. Designed for use in mixed teams of experienced and inexperienced RDF developers. Written in PSR-2 compliant PHP and tested extensively using PHPUnit. Link: *http://www.easyrdf.org/*
- **RDFLib** is a pure Python package working with RDF that contains most things you need to work with, including: parsers and serializers for RDF/XML, N3, NTriples, N-Quads, Turtle, TriX, RDFa and Microdata; a Graph interface which can be backed by any one of a number of Store implementations; store implementations for in memory storage and persistent storage on top of the Berkeley DB; a SPARQL 1.1 implementation supporting Queries and Update statements. Link: *https://rdflib.readthedocs.org/en/latest/*

Non-RDF DB support

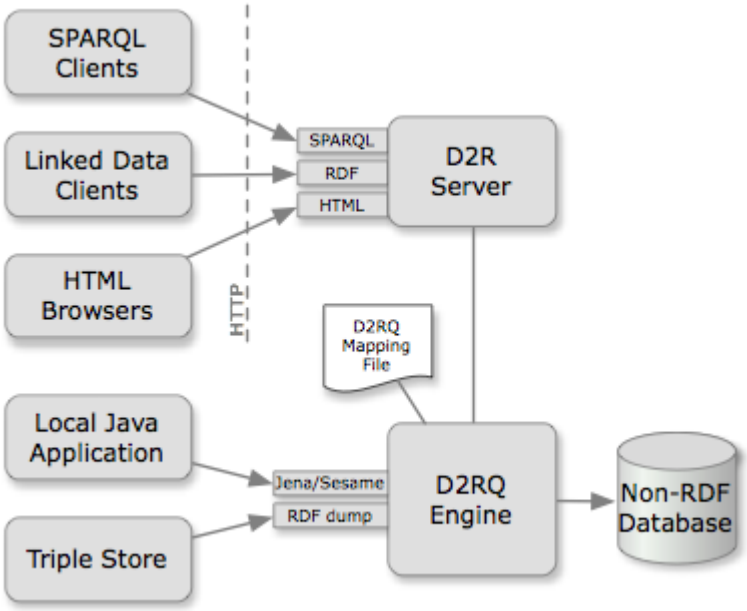
- ***D2RQ Platform*** is a system for accessing relational databases as virtual, read-only RDF graphs. It offers RDF-based access to the content of relational databases without having to replicate it into an RDF store. Using D2RQ you can: query a non-RDF database using SPARQL, access the content of the database as Linked Data over the Web, create custom dumps of the database in RDF formats for loading into an RDF store, access information in a non-RDF database using the Apache Jena API.

  Link: *http://d2rq.org/*

The D2RQ Platform consists of:

- **D2RQ Mapping Language**, a declarative mapping language for describing the relation between an ontology and an relational data model.
- **D2RQ Engine**, a plug-in for the Jena Semantic Web toolkit, which uses the mappings to rewrite Jena API calls to SQL queries against the database and passes query results up to the higher layers of the frameworks.
- **D2R Server**, an HTTP server that provides a Linked Data view, a HTML view for debugging and a SPARQL Protocol endpoint over the database.

Supported databases: Oracle, MySQL, PostgreSQL, SQL Server, HSQLDB, Interbase/Firebird



Hybrid Stores

RDF Stores that supports both architectural styles (native and DBMS-backed):

- **RDF4J (Sesame)** (open-source) is an open source framework for storage, inferencing and querying of RDF data. It is a library that is release under the Aduna BSD-style license and can be integrated in any Java application. Sesame includes RDF parsers and writers (Sesame Rio), a storage and inference layer (SAIL API) that abstracts from storage and inference details, a repository API for handling RDF data, and an HTTP Server for accessing Sesame repositories via HTTP. It operates in any Java-supporting environment and can be used by any Java application. Link: *http://rdf4j.org/*
- **OpenLink Virtuoso Universal Server** is a hybrid storage solution for a range of data models, including relational data, RDF and XML, and free text documents. Through it unified storage it can be also seen as a mapping solution between RDF and other data formats, therefore it can serve as an integration point for data from different, heterogeneous sources. Virtuoso has gained significant interest since it is used to host many important Linked Data sets (e.g., DBpedia), and preconfigured snapshots with several important Linked Data sets are offered. Virtuoso is offered as an open-source version; for commercial purposes several license models exist. Link: *http://virtuoso.openlinksw.com/*
- **Blazegraph** (former Bigdata)(open-source and commercial license) is ultra-scalable, high-performance graph database with support for the Blueprints and RDF/SPARQL APIs. Blazegraph is available in a range of versions that provide solutions to the challenge of scaling graphs. Blazegraph solutions range from millions to trillions of edges in the graph. Link: *https://www.blazegraph.com/product/*
- **CumulusRDF** is an RDF store on a cloud-based architecture. CumulusRDF provides a REST-based API with CRUD operations to manage RDF data. The current version uses Apache Cassandra as storage backend. Link: *https://github.com/cumulusrdf/cumulusrdf*
- **RedStore** is a lightweight RDF triplestore written in C using the Redland library. Link: *http://www.aelius.com/njh/redstore/*

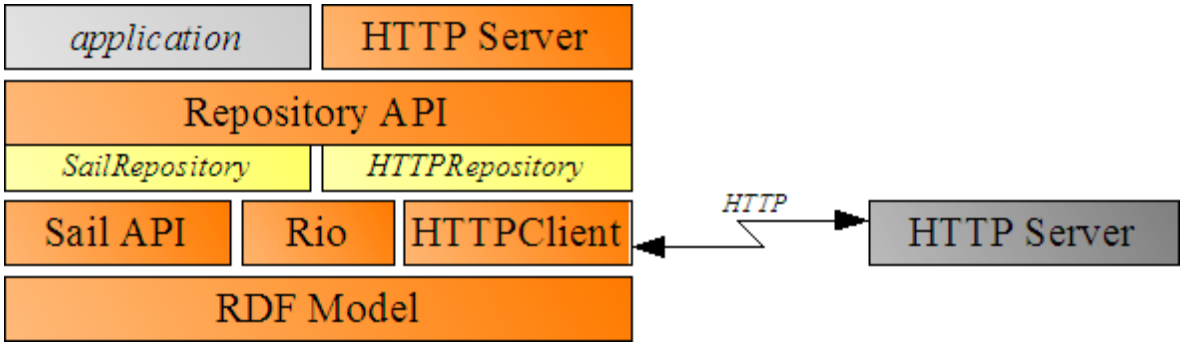
RDF4J

DRF4J (former **Sesame**) an open-source Java framework for processing RDF data. Link: *http://rdf4j.org/*

- Features:
  - Parsing: Supports all major notations
  - Storing: In-memory, native store, file-based, and supports many third-party storage solutions (*http://rdf4j.org/about/rdf4j-databases/*)
  - Inferencing: Ontology-based, custom Rule-based
  - Querying: SPARQL, SeRQL
- easy-to-use API that can be connected to all leading RDF storage solutions.

AliBaba is an RDF application library for developing complex RDF storage applications. It is a collection of modules that provide simplified RDF store abstractions to accelerate development and facilitate application maintenance. Link: *https://bitbucket.org/openrdf/alibaba*

RDF4J (Sesame) architecture



- Rio (RDF I/O)
  - Parsers and writers for various notations
- Sail (Storage And Inference Layer)
  - Low level System API
  - Abstraction for storage and inference
- Repository API
  - Higher level API
  - Developer-oriented methods for handling RDF data
- HTTP Server
  - Accessing Sesame through HTTP

Environmental variables

practical

- OS stores simple variables in its memory
- Applications can ask OS for the value
- Each variable has a name and a simple textual value
- To see variables and their values run (Win):
  - See all: `set`
  - See a concrete variable: `set VARNAME`
- Scope:
  - Global (valid for the whole OS)
  - Local (valid in the current command line window)
- To change or set new local variable run:

```
set VARNAME=some text here
```

Installing RDF4J workbench

- Simple web interface for storing and querying RDF data
- Install steps (no admin rights needed): **practical**
  1. Download and unzip newest RDF4J and Tomcat
  2. Copy all **`*.war`** files from RDF4Js **`war`** folder to Tomcat's **`webapps`** folder
  3. Start Tomcat
     - From bin folder by running **`startup.sh`** (UNIX) or **`startup.bat`** (Win)
     - You may need to set **`JAVA_HOME`** variable (it should point to JDK or JRE main folder)
  4. Go to *http://localhost:8080/rdf4j-workbench/*

More information: *http://docs.rdf4j.org/server-workbench-console/*

GraphDB

Otnotext GraphDB (formerly OWLIM) is a leading RDF Triplestore built on OWL (Ontology Web Language) standards. GraphDB handles massive loads, queries and OWL inferencing in real time. There are three different editions: *GraphDB Free*, *GraphDB Standard* and *GraphDB Enterprise*. Link: *http://ontotext.com/products/graphdb/*, *http://graphdb.ontotext.com/*

- Features:
  - Parsing: Supports all major notations
  - Storing: file-based native store
  - Inferencing: at load time, Ontology-based, custom Rule-based
  - Querying: SPARQL, supports HTTP based querying
- *GraphDB Server*: a set of programming interfaces that exposes all database functionality as a REST API.
- *GraphDB Workbench*: management interfaces implemented on top of the GraphDB Server REST API.
- Fully compliant with the RDF4J (Sesame) service API
- GraphDB on the Cloud: database as a service solution for small or medium database size and query load. (*https://cloud.ontotext.com/#/home*)
- Tools: *LoadRDF* tool: an OFFLINE tool, designed for fast loading of large data sets. (*http://graphdb.ontotext.com/documentation/free/loadrdf-tool.html*)

  Storage tool: an application for scanning and repairing a GraphDB repository. (*http://graphdb.ontotext.com/documentation/free/storage-tool.html*)

Installing GraphDB

- Installation options:
  - Run GraphDB as a desktop installation (only for GraphDB Free users)
  - Run GraphDB as a stand-alone server
  - Run GraphDB as a docker container

- *GraphDBFree* installation: **practical**
  1. Download installation file and run it (*https://ontotext.com/products/graphdb/*)
  2. Access workbench via *http://localhost:7200/*

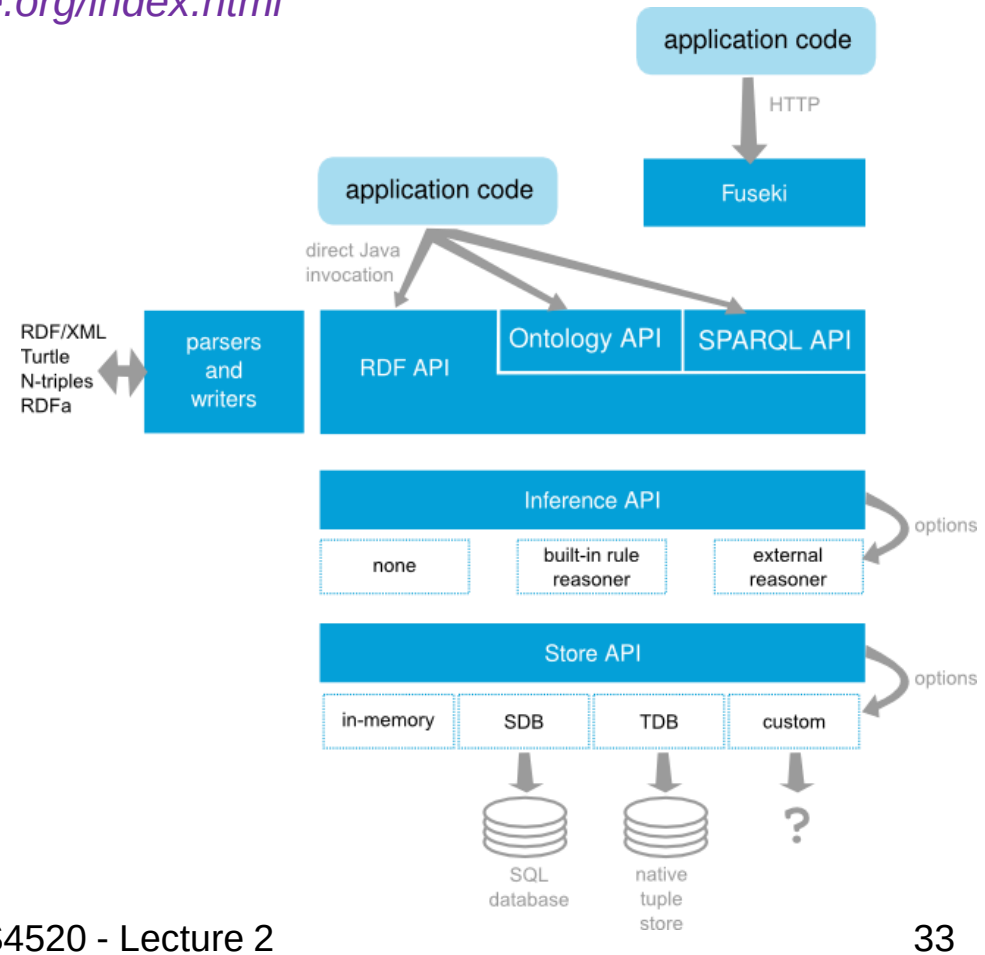
More information: *http://graphdb.ontotext.com/documentation/free/running-graphdb.html*

Apache Jena

Apache Jena is a Java framework (collection of tools and Java libraries) to simplify the development of Semantic Web and Linked Data applications. Link: *http://jena.apache.org/index.html*

Includes:

- RDF API for processing RDF data in various notations
- Ontology API for OWL and RDFS
- Rule-based inference engine and Inference API
- TDB – a native triple store
- SPARQL query processor (called *ARQ*). Supports *Basic Federated SPARQL Query*.
- Servers for publishing RDF data to other applications
  - *Fuseki* – a SPARQL end-point accessible over HTTP



ARQ

practical

- Installation:
  - Download and unzip Jena
  - Set **`JENAROOT`** environmental variable to folder where you unzipped Jena
  - In **`bat`** (Win) or **`bin`** (UNIX) you find **`arq`** executable
- Usage:
  - Prepare a SPARQL query and save it into a file (here: **`query.sparql`**)
  - Prepare some data file (if needed) – e.g. **`data.rdf`**(.**`ttl`**)
  - Execute the query on top of the data by running:

```
arq --query=query.sparql --data=data.rdf
```

- In case of problems use **`arq --help`**

Apache Jena Fuseki

- ***Fuseki*** provides REST-style interaction with RDF data. It is a SPARQL server that provides REST-style SPARQL HTTP Update, SPARQL Query, and SPARQL Update using the SPARQL protocol over HTTP.
- Installation: **practical**

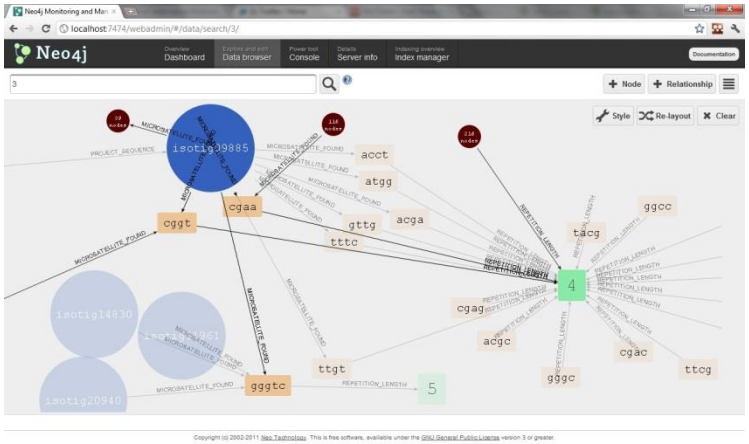
1. Download (*http://jena.apache.org/download/* or from course webpage) and unzip Fuseki
2. Check Fuseki, Java, Tomcat versions compatibility. *(e.g. F3.13.0, J13, T9)*
3. Set **`FUSEKI_HOME`** and **`FUSEKI_BASE`** environmental variables to folder where you unzipped Fuseki
4. Start server from Fuseki folder by running:
   ```
   fuseki-server --update --mem /ds
   ```
   Open *http://localhost:3030/*
   ```
   fuseki-server --update --loc=c:/dir… /ds
   ```
5. Fuseki can run from a WAR file (fuseki.war). Put the file to the <webapp> folder of Tomcat server and run it. Access Fuseki by: *http://localhost:8080/fuseki/*
6. Read about other run options from official documentation …
7. Documentation: *http://jena.apache.org/documentation/fuseki2/*
   http://jena.apache.org/documentation/serving_data

AllegroGraph

- ***AllegroGraph*** is a high-performance persistent graph database
- Editions of AllegroGraph: the free *RDFStore* server edition is limited to storing less than 50 million triples, a developer edition capable of storing a maximum of 600 million triples, and an enterprise edition with storage capacity only limited by the underlying server infrastructure.
- Supports SPARQL, RDFS++, and Prolog reasoning
- Supports REST Protocol clients: Java RDF4J (Sesame), Java Jena, Python, C#, Clojure, Perl, Ruby, Scala and Lisp clients.
- Link: *https://allegrograph.com/products/allegrograph/*, *http://www.franz.com/agraph/allegrograph/*
- ***AllegroGraph Web View (AGWebView)*** is a graphical user interface for exploring, querying, and managing AllegroGraph triple stores. It uses HTTP interface to provide the services through a web browser. It is included with the *AllegroGraph*.
- ***Gruff*** - a graph-based triple-store browser for AllegroGraph.
  - Download and unzip the package: *https://allegrograph.com/products/gruff/* , *https://allegrograph.com/gruff-learning-center/*, *http://www.franz.com/agraph/gruff/*
  - Run the browser, create new triple store

Neo4j Graph Database

- ***Neo4j*** is a highly scalable, robust (fully ACID) native graph database, used in mission-critical apps by thousands of leading startups, enterprises, and governments around the world.
  - High Performance for highly connected data
  - High Availability clustering
  - Cypher, a graph query language
  - ETL, easy import with Cypher LOAD CSV
  - Hot Backups and Advanced Monitoring



- Link: *http://neo4j.com/*
  - *http://en.wikipedia.org/wiki/Neo4j*
  - *https://www.youtube.com/channel/UCvze3hU6OZBkB1vkhH2lH9Q*
- *Neo4j Manual* - *http://neo4j.com/docs/stable/index.html*

Further reading

- N-triples: *http://www.w3.org/2001/sw/RDFCore/ntriples/*
- Turtle: *http://www.w3.org/TeamSubmission/turtle/* ; *http://www.w3.org/TR/turtle/*
- Notation3: *http://www.w3.org/DesignIssues/Notation3.html*
- TriX: *http://www.hpl.hp.com/techreports/2004/HPL-2004-56.html*
- Datatypes: *https://www.w3.org/TR/2013/WD-rdf11-concepts-20130115/#xsd-datatypes*

RDF validators:

- *https://www.w3.org/RDF/Validator*
- *https://rdf-translator.appspot.com*
- *https://www.ldf.fi/service/rdf-grapher*
- *http://rdfvalidator.mybluemix.net*

Task 1