

Crafting **RDF**

Generating Knowledge Graphs from Legacy Data

Dr. Ora Lassila

Founder, **So Many Aircraft** + Principal Technologist, Amazon Neptune

Connected Data London, November 2025



Speaker introduction: Ora Lassila

Founder, So Many Aircraft
Principal Technologist, Amazon Neptune

Past:

Head of Ontology Engineering, State Street

Research Fellow, Nokia Research

W3C Fellow, MIT/LCS

Project Manager, CMU Robotics Institute

...

Education:

Ph.D (D.Sc) AI+CS, Helsinki University of Technology

Co-author of the seminal paper on the Semantic Web

Co-editor of the original RDF specification

Co-chair, W3C RDF 1.2 WG

Designer of the KR substrate for NASA's Deep Space 1

Grand Prize Winner, USENIX Obfuscated C Code Contest

Some background

My “day job” is to help **Amazon Neptune** customers to

Understand the Semantic Web technologies

Model their data, to define or adopt ontologies

Learn how to create data for their knowledge graphs

Make sense of the choice between RDF and LPGs

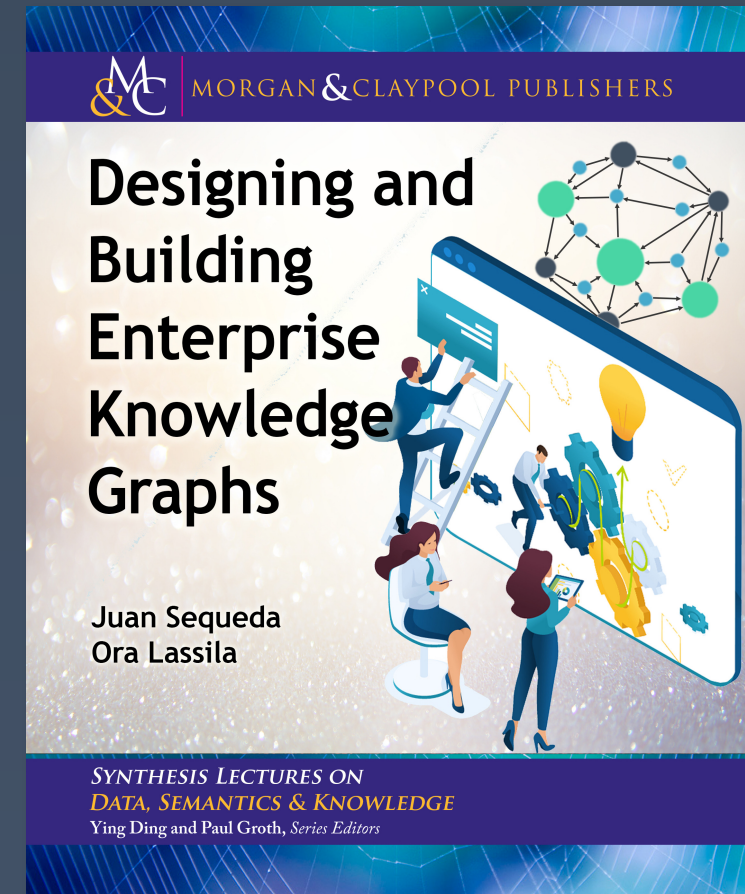
etc.

I am co-chairing the W₃C RDF 1.2 (“RDF-star”) working group

I see the “pain points” in RDF and Semantic Web adoption

(NB: Raw scalability is not a problem, Neptune and most other triple stores scale quite well)

This talk was inspired by my experience at the OMG Semantic Augmentation Challenge



Some more background

So Many Aircraft is a publisher focusing on aircraft, aviation history, and scale models

We have published two books (so far)

We use a knowledge graph for

Our reference library (5,000+ publications)

Photo collection (60,000+ original photos)

Aviation-related SKOS taxonomy (5,000+ concepts)

Our internal software uses our own OSS libraries

rdhelpers – help for building RDF “pipelines” (on top of RDFLib)

tinyrml – a “pythonic” take on RML (with extensions)

xmptools – support for XMP metadata



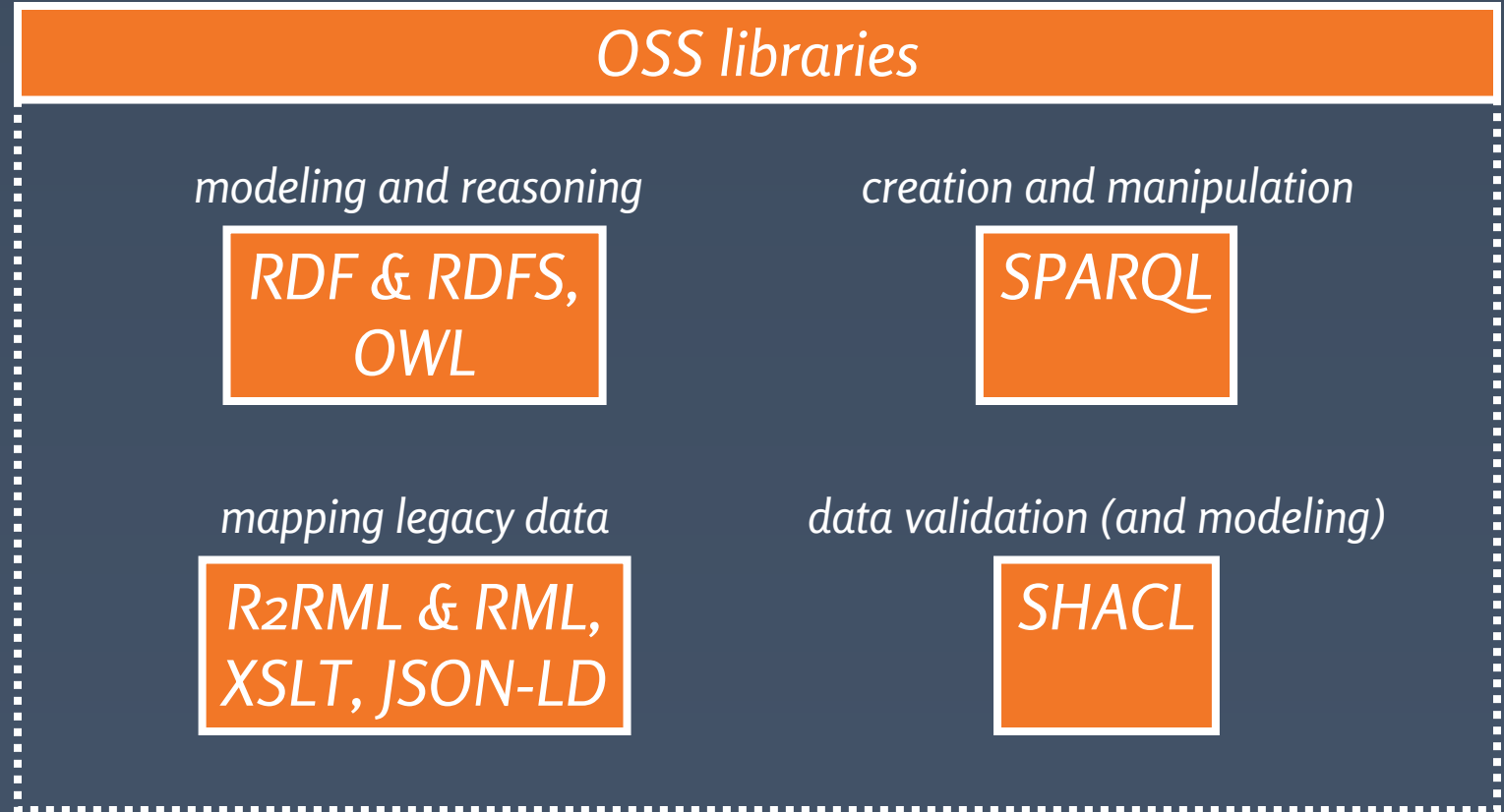
“Legacy data” = not RDF

Premise: Tools for RDF exist (and have for many years)

The Semantic Web stack is a collection of easy-to-use standards and technologies.

They can be “orchestrated” with simple scripting and some open-source libraries.

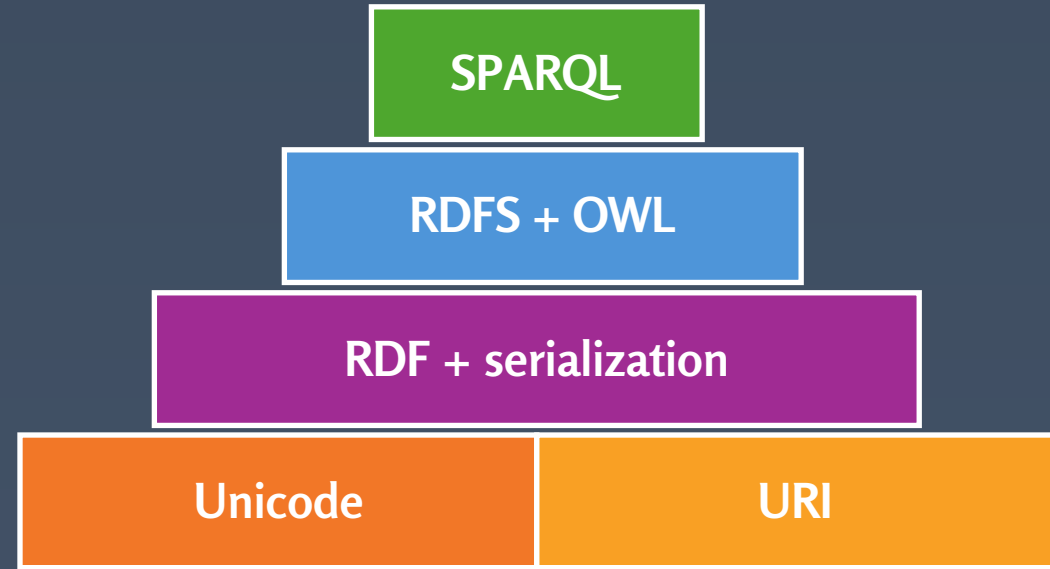
All of these have been around for years already.



This is not rocket science!

Semantic Web stack: Layer cake or pipeline?

Forget the layer cake, it is not useful



Semantic Web stack: Layer cake or pipeline?

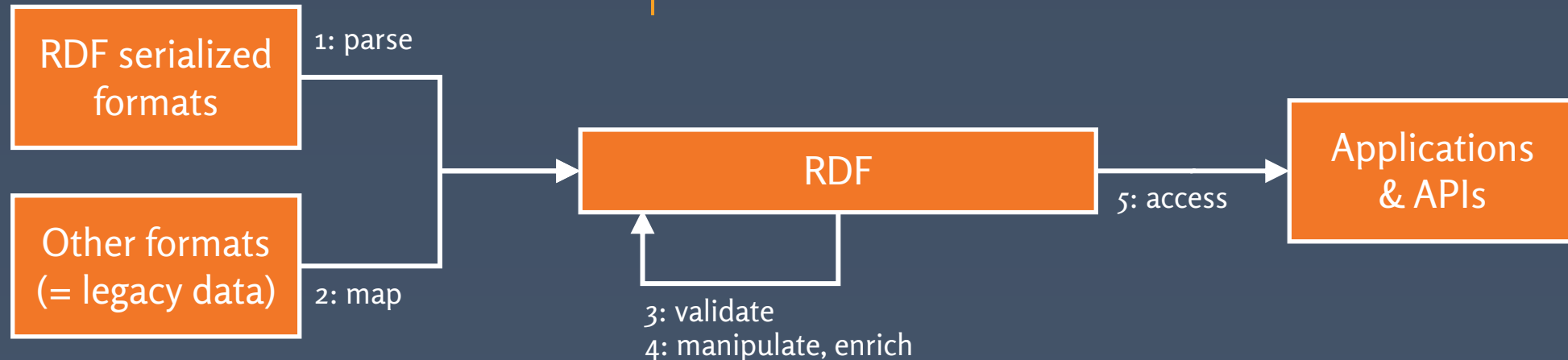
Forget the layer cake, it is not useful

Rather, think of this as a pipeline:

1. Create and ingest
2. Store and enrich
3. Access and consume

Use standard technologies:

1. **parse**: standard RDF syntaxes
2. **map**: R2RML, RML, XSLT, JSON-LD
3. **validate**: SHACL, SPARQL
4. **manipulate, enrich**: SPARQL, RDFS/OWL reasoning
5. **access**: SPARQL



Semantic Web stack: Layer cake or pipeline?

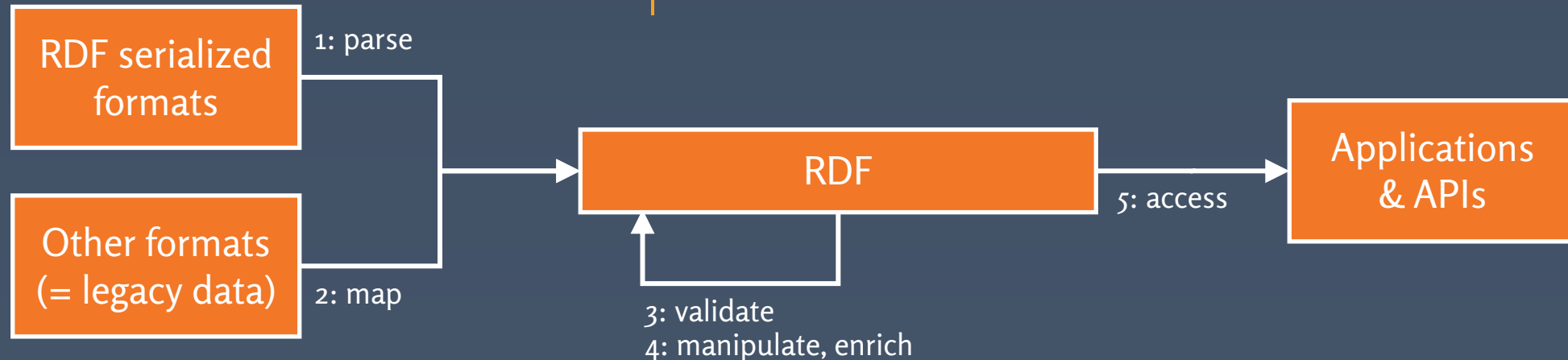
Forget the layer cake, it is not useful

Rather, think of this as a pipeline:

1. Create and ingest
2. Store and enrich
3. Access and consume

Use standard technologies:

1. **parse**: standard RDF syntaxes
2. **map**: R2RML, RML, XSLT, JSON-LD
3. **validate**: SHACL, SPARQL
4. **manipulate, enrich**: SPARQL, RDFS/OWL reasoning
5. **access**: SPARQL



Semantic Web stack: Layer cake or pipeline?

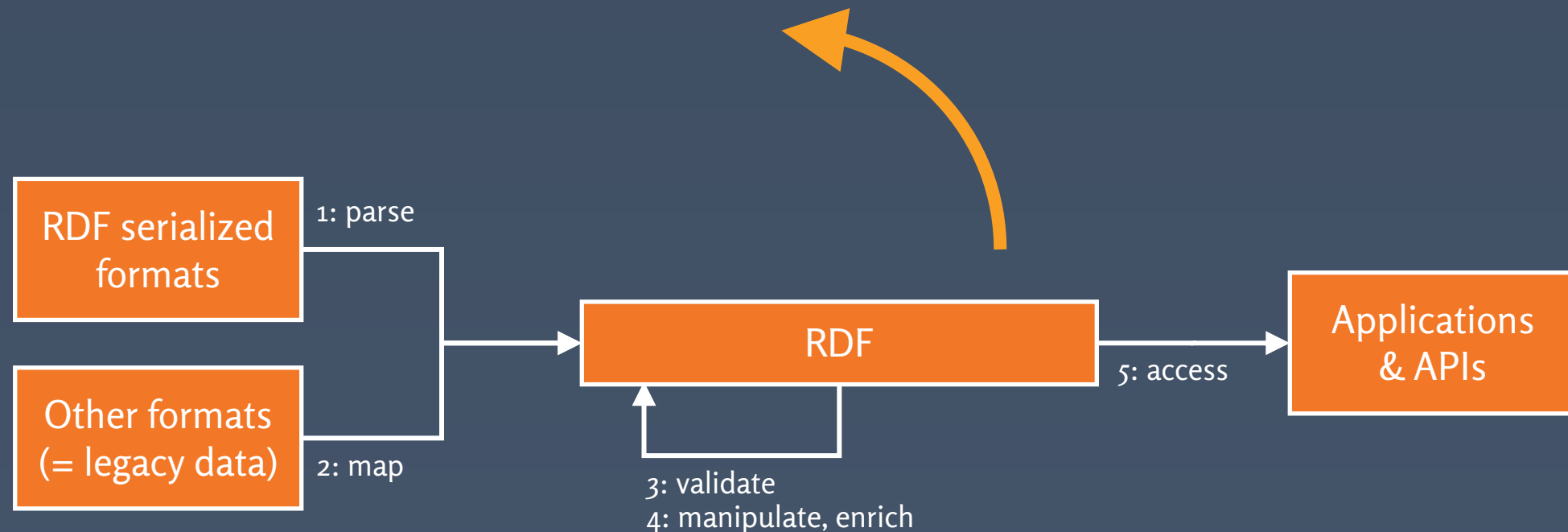
Universal pattern for KG systems

1. Create and ingest
2. Store and enrich
3. Access and consume

Semantic layer...?

A semantic layer can benefit all your applications, not just BI

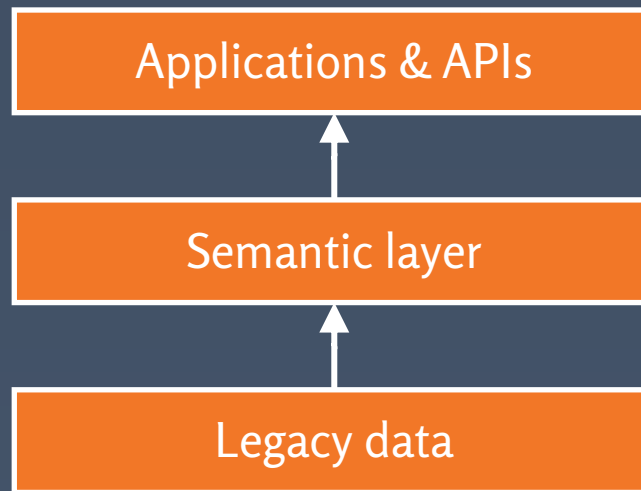
Turn the previous diagram 90 degrees counterclockwise ;-)



Semantic layer...?

A semantic layer can benefit all your applications, not just BI

Turn the previous diagram 90 degrees counterclockwise ;-)



Please ignore the people who talk about separate “ontology” and “semantic” layers

R2RML: RDB-to-RDF Mapping Language

Going from tables to graphs is easy; default mapping:

1. Every table becomes a **class**
2. Every row becomes a **node**
3. Primary keys inform you how to mint node **URIs**
4. Foreign keys are **edges** to other nodes (owl:ObjectProperty)
5. Other columns are **properties** with scalar values (owl:DatatypeProperty)

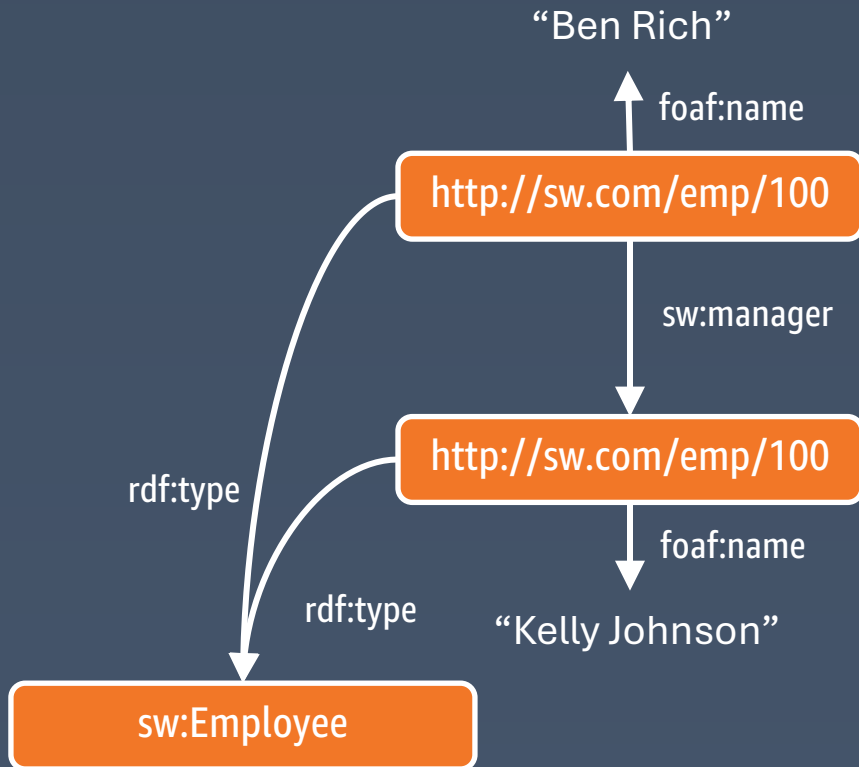
R2RML is a language for writing **custom** mappings

Can also be used to establish “virtual” graphs

R2RML example

Employee

id	mgr_id	name
100	101	Ben Rich
101	NULL	Kelly Johnson



```
:EmployeeMap
  rr:logicalTable [
    rr:tableName "Employee"
  ];
rr:subjectMap [
  rr:template "http://sw.com/emp/{id}" ;
  rr:class sw:Employee
];
rr:predicateObjectMap [
  rr:predicate sw:manager ;
  rr:objectMap [ rr:template "http://sw.com/emp/{mgr_id}" ]
];
rr:predicateObjectMap [
  rr:predicate foaf:name ;
  rr:objectMap [ rr:column "name" ]
].
```

RML: RDF Mapping Language

An extension of R2RML

Can map tabular as well as **semi-structured** formats

Not a W3C standard, but well accepted and implemented

Relies on XPath and JSONPath

XSLT: Extensible Stylesheet Language Transformations

A language for transforming XML (to other XML)

RDF has an XML-based syntax (among many), and therefore...

You can use XSLT to transform **legacy XML to RDF**

XSLT example (input)

```
<Employees>
```

```
  <Employee>
```

```
    <id>100</id>
```

```
    <mgr-id>101</mgr-id>
```

```
    <name>Ben Rich</name>
```

```
  </Employee>
```

```
  <Employee>
```

```
    <id>101</id>
```

```
    <name>Kelly Johnson</name>
```

```
  </Employee>
```

```
</Employees>
```

```
<xsl:template match="mgr-id">
```

```
  <xsl:variable name="mgr-id" select="."/>
```

```
  <sw:manager rdf:resource="http://sw.com/emp/{$mgr-id}"/>
```

```
</xsl:template>
```

```
<xsl:template match="/">
```

```
  <rdf:RDF>
```

```
    <xsl:for-each select="/Employees/Employee">
```

```
      <sw:Employee rdf:about="http://sw.com/emp/{id}">
```

```
        <sw:name>
```

```
          <xsl:value-of select="name"/>
```

```
        </sw:name>
```

```
        <xsl:apply-templates select="mgr-id"/>
```

```
      </sw:Employee>
```

```
    </xsl:for-each>
```

```
  </rdf:RDF>
```

```
</xsl:template>
```

XSLT example (result)

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:sw="http://sw.com/schema/emp#"
  xmlns:foaf="http://xmlns.com/foaf/0.1/">
```

```
<sw:Employee rdf:about="http://sw.com/emp/100">
  <foaf:name>Ben Rich</sw:name>
  <sw:manager rdf:resource="http://sw.com/emp/101"/>
</sw:Employee>
```

```
<sw:Employee rdf:about="http://sw.com/emp/101">
  <foaf:name>Kelly Johnson</sw:name>
</sw:Employee>
```

```
</rdf:RDF>
```

```
@prefix sw: <http://sw.com/schema/emp#> .
@prefix employee: <http://sw.com/emp/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

```
employee:100 a sw:Employee ;
  sw:manager employee:101 ;
  foaf:name "Ben Rich" .
```

```
employee:101 a sw:Employee ;
  foaf:name "Kelly Johnson" .
```

JSON-LD: JSON for Linked Data

Opinionated JSON convention for representing RDF

JSON-LD processors transform JSON to RDF using a “context”

- Maps JSON keys to RDF URIs

- Can be embedded or be a separate document

“Plain” JSON can be transformed to RDF by introducing a suitable external context

- Shortcoming: JSON-LD has no string templating mechanism for minting URIs

- Resulting RDF may have to be “post-processed” (say, using SPARQL updates)

SPARQL: RDF → RDF transformations

Sometimes this is the “method of last resort”

Typical uses:

Converting containers/collections to repeated properties

Switching namespaces (e.g., dc:subject → dcterms:subject)

Cleaning up, removing unneeded data

Adding annotations: owl:Ontology, PROV-O, etc.

SPARQL can also be used for QA in lieu of SHACL

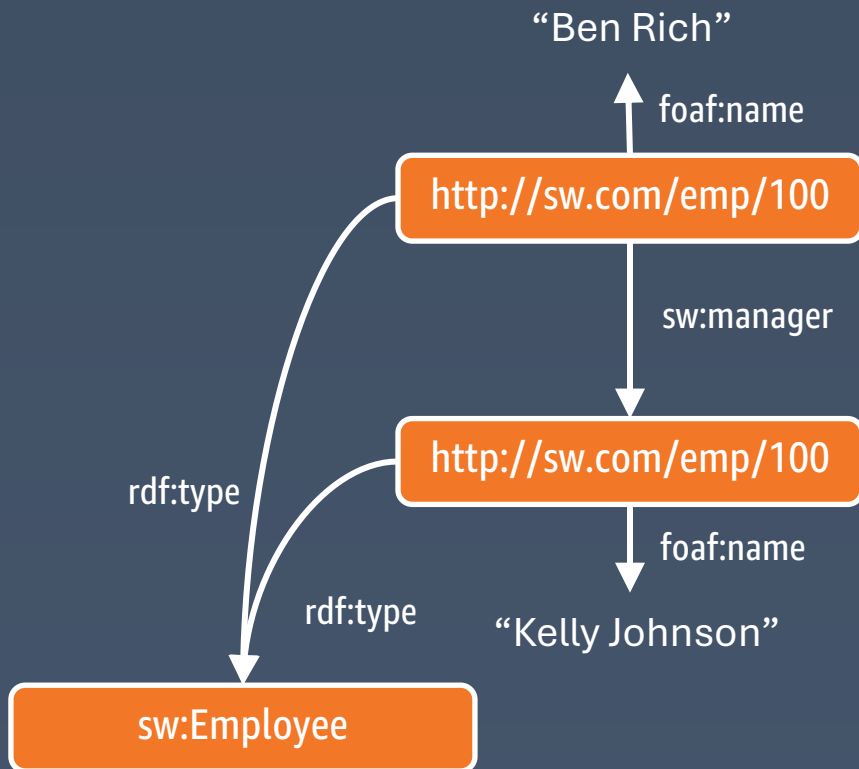
```
DELETE {  
    # old stuff  
}  
INSERT {  
    # new stuff  
}  
WHERE {  
    # matching conditions  
}
```



SPARQL: RDF → RDF transformations (example)

Add an organization

Link newly created employees



```
INSERT {
```

```
  <http://sw.com> a foaf:Group ;  
    foaf:name "SkunkWorks" ;  
    foaf:member ?emp
```

```
}
```

```
WHERE {
```

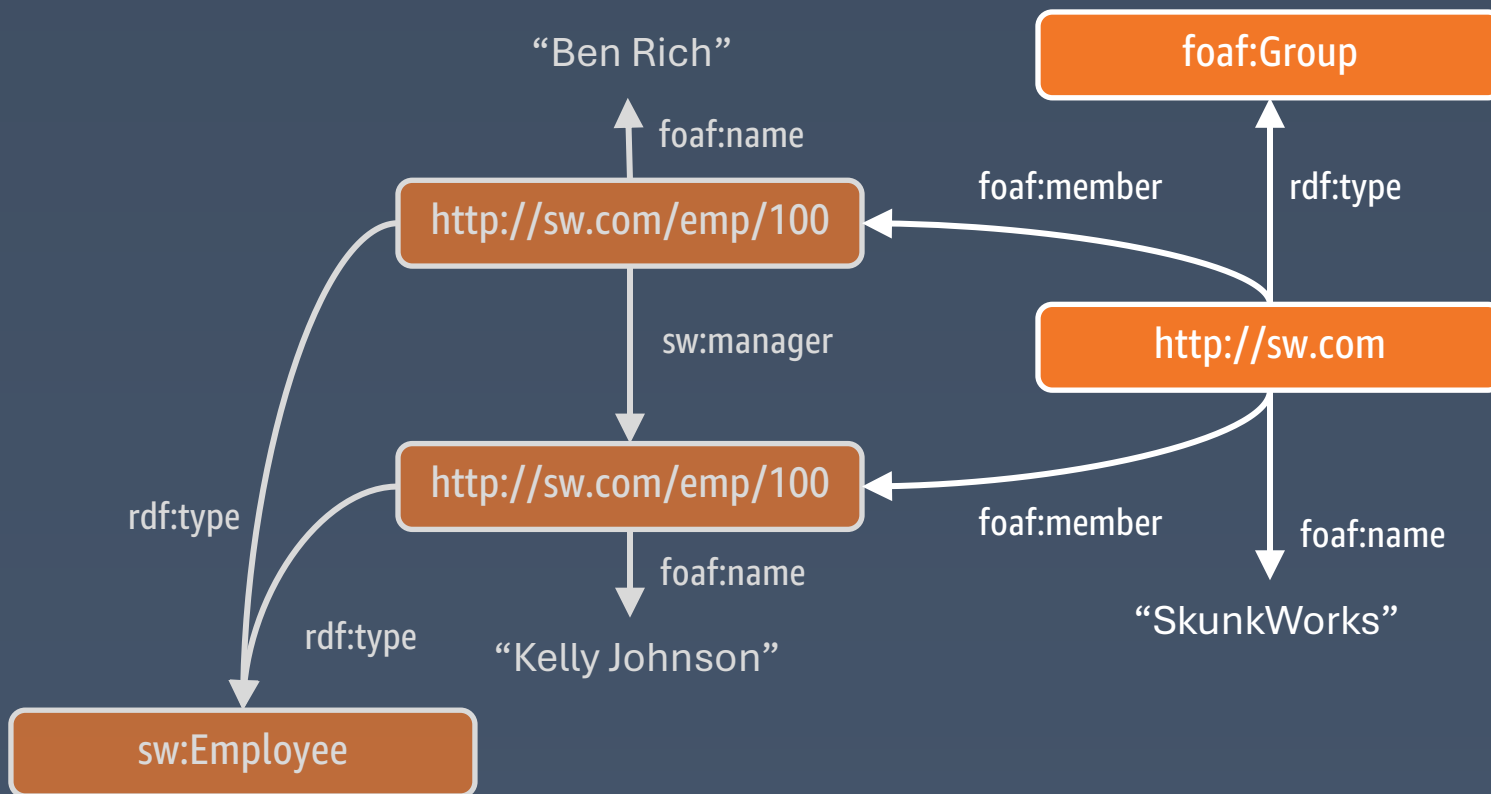
```
  ?emp a sw:Employee
```

```
}
```

SPARQL: RDF → RDF transformations (example)

Add an organization

Link newly created employees



```
INSERT {
```

```
<http://sw.com> a foaf:Group ;  
  foaf:name "SkunkWorks" ;  
  foaf:member ?emp
```

```
}
```

```
WHERE {
```

```
?emp a sw:Employee
```

```
}
```

Legacy constructs → RDF graphs

File system → a graph

Web site → a graph

Relational tables → a graph

CSV files → a graph

Semi-structured data → a graph

Metadata from media → a graph

API results → (yep, you guessed it...) a graph

Summary

R2RML, RML, XSLT, and JSON-LD can be used to map legacy data to RDF

SPARQL can be used to transform existing RDF

Many things can have a useful RDF representation

Standards and software are **mature and easy to use**

Think in terms of **pipelines**, not layer cakes

Questions?

ora@somanyaircraft.com

ora@amazon.com

smasw.gitlab.io

www.somanyaircraft.com

aws.amazon.com/neptune/