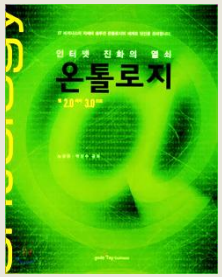


인 터 넷 진 화 의 열 쇠

온 톨 로 지

웹 2.0에서 3.0으로

노상규·박진수 공저

god's toy business, 2007

SNU IDB laboratory

Ontology Contents

MODULE 1 온톨로지의 개념 및 응용

MODULE 2 온톨로지 언어와 구축도구

Chapter 6 RDF(S): RDF와 RDF Schema

- RDF(Resource Description Framework)
  - 단순한 Triple 형태로 웹 자원을 기술
  - W3C에 의해 개발, 1999년 W3C 권고안 설정

- RDF Schema
  - RDF를 프레임 지식표현 패러다임으로 확장한 언어
  - 객체지향 모델링과 비슷한 도메인 구성에 대한 표현력 제공
  - W3C에 의해 개발, 2004년 W3C 권고안 설정

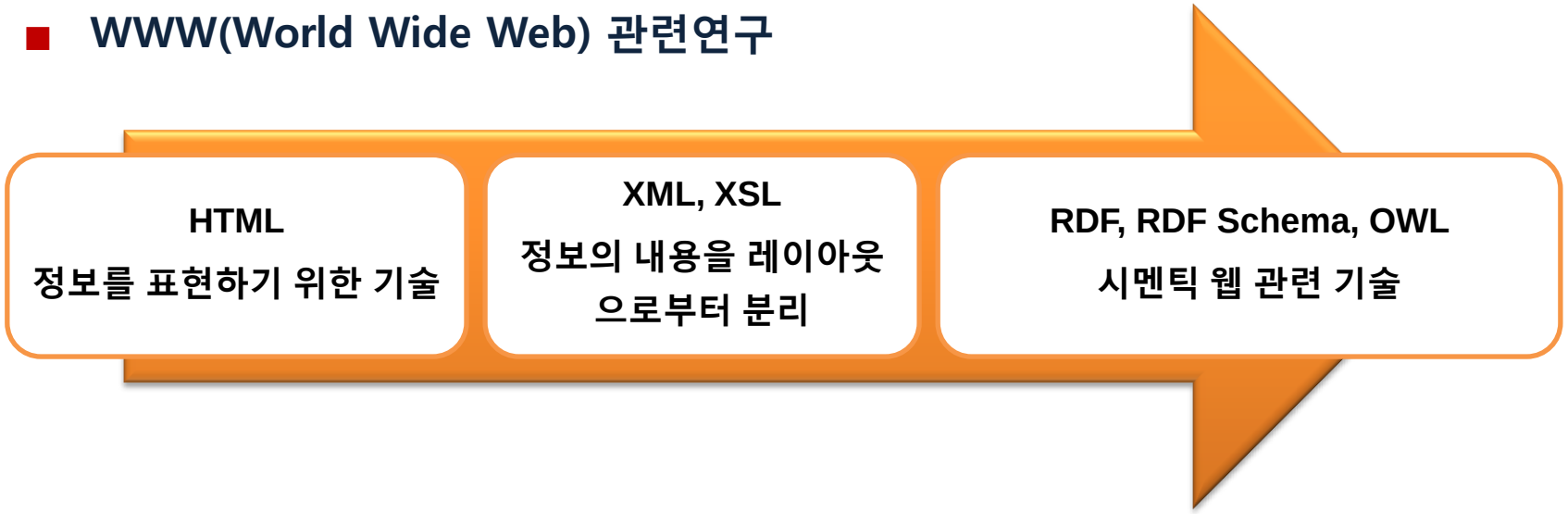
- RDF(S) = RDF + RDF Schema
- RDF(S) + 기술논리 지식표현 패러다임 → OIL, DAML+OIL, OWL

Chapter 6 RDF(S): RDF와 RDF Schema

- 1. XML과 RDF
- 2. RDF
  - 2.1 RDF 데이터 모델
  - 2.2 RDF 그래프와 코딩 예
- 3. RDF Schema
  - 3.1 RDF와 RDF Schema
  - 3.2 기본 요소: 클래스와 속성
  - 3.3 RDF Schema 코딩 예
- 4. RDF(S)의 한계점

1. XML과 RDF

- WWW(World Wide Web) 관련연구



- XML: W3C의 후원으로 결성된 XML Working Group에 의해 1996년 제안
  - XML 태그의 역할
    - 정보검색 및 정렬을 위해 정보를 구분하는 이름을 사용자가 임의로 작성
  - 텍스트 파일로 저장
  - 여러 기종과 OS에 대해 이식성 높음
  - 문서의 구조를 정의하는 스키마 사용, 우수한 호환성
  - 내용과 디자인을 완전히 분리

1. XML과 RDF: XML의 Limitation

- 정보의 의미를 명확히 전달해주는 매커니즘을 제공하지 않음
  - <전화번호> = <전화> = <Telephone> = <TelephoneNumber>..
- 태그 해석규칙이 없어 태그 간의 의미의 연관성을 컴퓨터가 추론하기 어려움
- HTML보다 유연하나 Semantic Web을 위한 표준 언어로는 부족함

예) "김동건이 '생각하는 컴퓨터'를 썼다." 의 여러가지 XML문장들

```
<책 이름="생각하는 컴퓨터">
    <저자>김동건</저자>
</책>
```

```
<저자 이름="김동건">
    <썼다>생각하는 컴퓨터</썼다>
</저자>
```

```
<책>
    <저자>김동건</저자>
    <제목>생각하는 컴퓨터</제목>
</책>
```

1. XML과 RDF: RDF의 등장

- XML의 문제 해결을 위해 제시된 RDF
  - '자원(주어부)-속성(서술부)-속성값(목적부)'을 하나의 기본 단위로 취급
  - 예)"김동건이 '생각하는 컴퓨터'를 썼다." → Triple을 기본 단위로 연결해 표현 가능

- RDF/XML
  - 위와 같은 RDF 데이터 모델을 컴퓨터가 이해할 수 있는 기계적 언어로 표현
  - 일반적으로 RDF 데이터 모델은 XML Syntax를 사용
    - XML에 대한 오랜 연구 축적, 분산환경에 적합한 특성 높은 호환성 등에 기인
  - RDF데이터 모델의 표현을 위한 XML Syntax: RDF/XML

    (혹은 XML Serialization of RDF)

1. XML과 RDF: 데이터 모델 관점에서의 비교

XML	RDF
순서가 중요한 트리구조	트리플구조(Subject/Predicate/Object)
검색하기 복잡	검색이 용이(독립적 트리플 집합)
Tag 배치 순서 다르면 다른 문서로 인식	각각 트리플이 독립적 집합 - 순서 상관 없음
메타데이터 표현의 유연성 낮음 노드(Tag)가 문서에 포함되어 인덱싱	메타데이터 표현의 유연성 높음 노드가 URIref 를 갖는 자원
XML Schema: 문서의 구문적 해석이 주요 기능	RDF Schema: 주로 의미적 해석에 사용

1. XML과 RDF: URI(Uniform Resource Identifier)

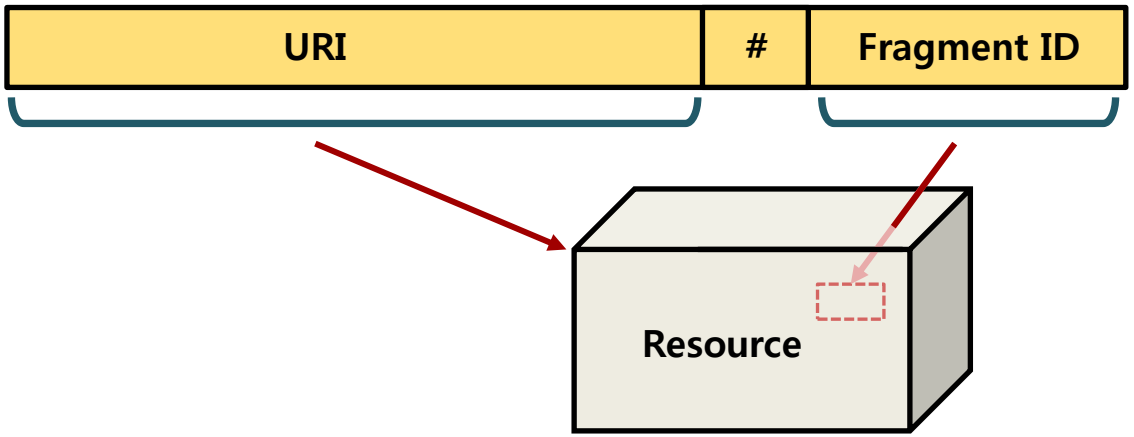
- URI: 웹 상에 존재하는 자원을 지칭하는 스트링 표준형식
  - URL (Uniform Resource Location)
  - URN (Uniform Resource Name)
- URL: 웹 페이지 같은 자원에 접근할 때 사용되는 실제 네트워크 경로
  - '프로토콜://파일이 저장된 서버의 DNS이름/디렉터리/파일 이름'으로 구성
  - 예) http:/www.snu.ac.kr/index.html
- URN: 임의의 자원을 가리키는 영속적이고 고유한 이름
  - 자원이 저장된 위치와 무관
  - 'urn: NID(Namespace ID) : NSS(Namespace Specific String)'으로 구성
  - NSS는 NID안에서 유일해야 함
    - 예) ISBN 번호 3960152782를 가진 책 ➔ urn: isbn: 3960152782
    - 예) 대한민국 국민, NSS로 주민등록번호 사용➔ urn: korean:901010-1234567

1. XML과 RDF: URIref (URI Reference)

- RDF는 자원에 대한 식별자로 URIref 사용

1. Absolute URIref: Context Independent

- URI + '#' + Fragment Identifier(단편식별자) 표시
  - 단편식별자 : 네임 스페이스에 해당하는 앞의 URI 안에서 효력이 있는 자원의 식별자

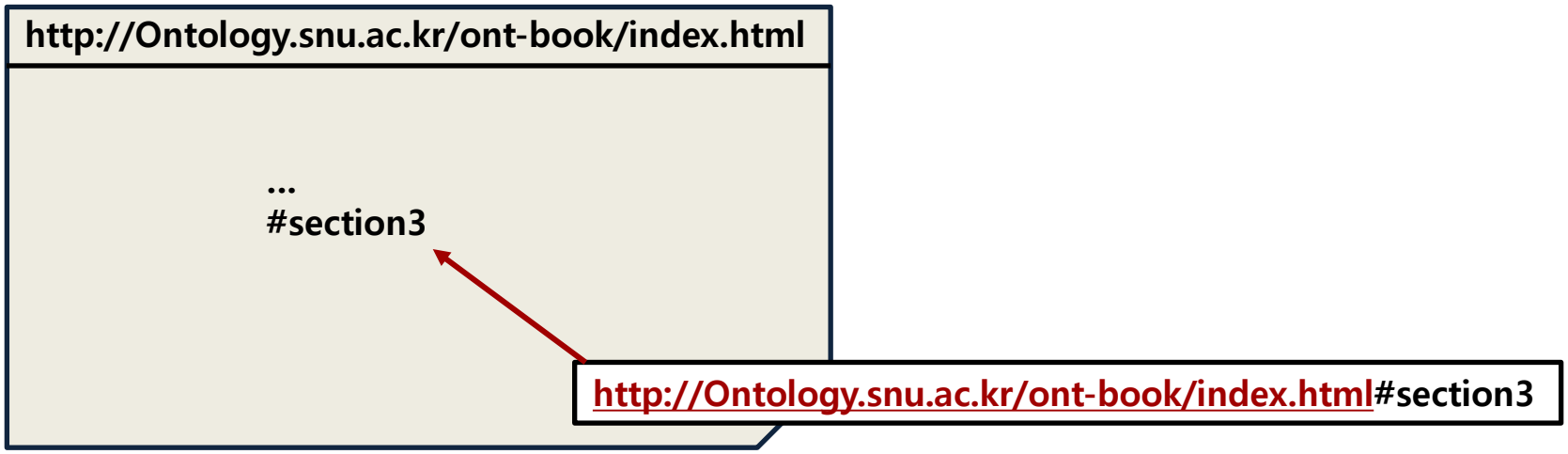


- 예) http://Ontology.snu.ac.kr/ont-book/index.html#section3
  - RDF에서 기술하는 자원이 URL로 표현되는 전자문서일 때
  - 이 문서의 full path가 URIref가 되는 것

1. XML과 RDF: URIref(URI Reference)

2. Relative URIref (shortcut): Context Dependent

- **Absolute URIref의 축약형, URIref의 URI 부분 생략**
- **예) @prefix obook 'http://Ontology.snu.ac.kr/ont-book/index.html'**
  - **'obook:#section3'라는 Relative URIref가 있으면,**
  - **Absolute URIref 'http://Ontology.snu.ac.kr/ont-book/index.html#section3'로 해석**



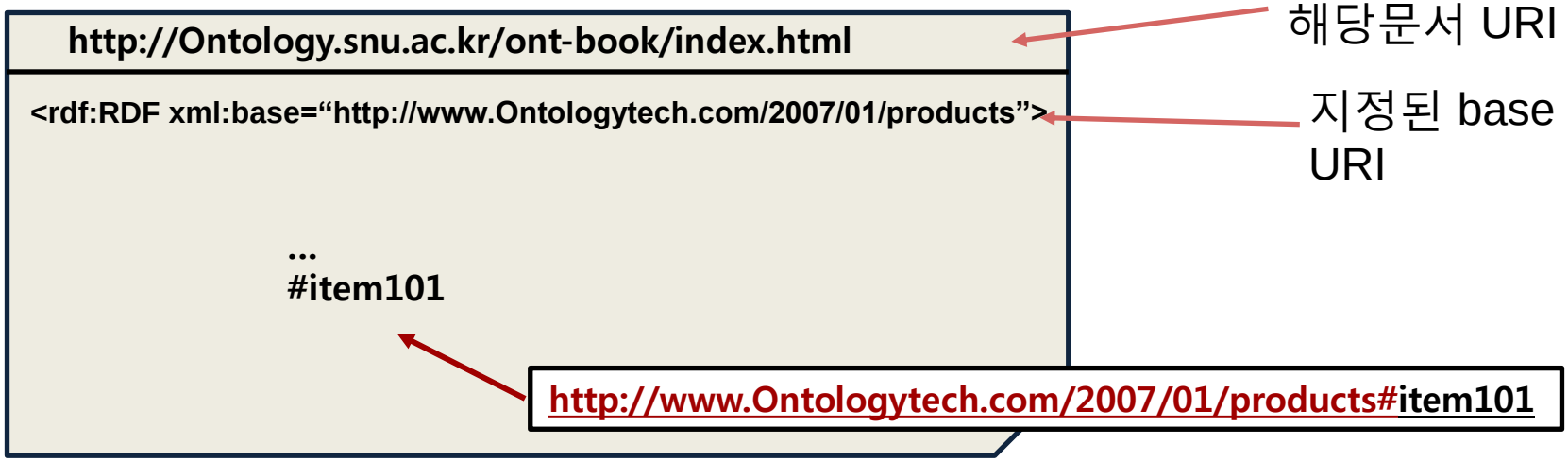
What if several section3?
Physical location meaningful?

1. XML과 RDF: xml:base

- 해당 문서의 Base URI가 무엇인지 찾는 방법
- Base URI를 지정 안하면 – 해당문서의 URI가 Base URI
- Base URI를 지정 하면 – Attribute 'xml:base' 사용하여 지정
  - 예) <rdf:RDF xml:base="http://www.Ontologytech.com/2007/01/products">

    이 문서에 Relative URIref '#item101'가 있으면, 문서의 URI와 상관없이

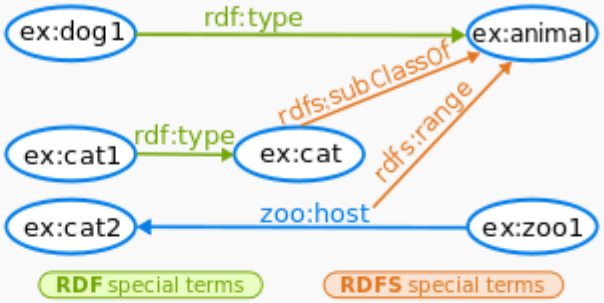
    'http://www.Ontologytech.com/2007/01/products#item101'로 해석



base URI와 @prefix는 같은것?

Chapter 6 RDF(S): RDF와 RDF Schema

- 1. XML과 RDF
- 2. RDF
  - 2.1 RDF 데이터 모델
  - 2.2 RDF 그래프와 코딩 예
- 3. RDF Schema
  - 3.1 RDF와 RDF Schema
  - 3.2 기본 요소: 클래스와 속성
  - 3.3 RDF Schema 코딩 예
- 4. RDF(S)의 한계점

In English	The graph
• Dog1 is an animal • Cat1 is a cat • Cats are animals • Zoos host animals • Zoo1 hosts the Cat2	

RDF/turtle

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ex:   <http://example.org/> .
@prefix zoo:   <http://example.org/zoo/> .
ex:dog1    rdf:type         ex:animal .
ex:cat1    rdf:type         ex:cat .
ex:cat     rdfs:subClassOf  ex:animal .
zoo:host   rdfs:range       ex:animal .
ex:zoo1    zoo:host         ex:cat2 .
```

If your triplestore (or RDF database) implements the regime entailment of RDF and RDFS, th

```
PREFIX  ex: <http://example.org/>
SELECT ?animal
WHERE
  { ?animal a ex:animal . }
```

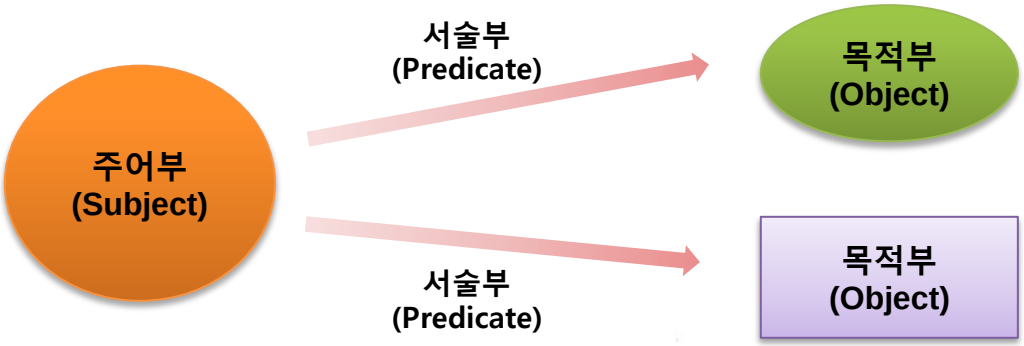
Gives the following result with *cat1* in it because the Cat's type inherits of Animal's type:

animal
<http://example.org/dog1>
<http://example.org/cat1>
<http://example.org/cat2>

2.1 RDF Data Model

- RDF
  - 모든 사물/개념을 Resource로 본다
  - ID로 URIref를 사용한다
    - “http://Ontology.snu.ac.kr/ontbook/plant/fruits#apple”
    - “http://dictionary.snu.ac.kr/language/words#apology”
  - Resource의 속성, Resource간의 관계를 기술하는 데이터 모델

- 기본단위: Triple 구조 (자원(주어부)-속성(서술부)-속성값(목적부)로 이루어진 서술문)



2.1 RDF Data Model: Triple

- **Subject (주어부)**
  - 문장의 주어 역할
  - Subject로 기술되는 Resource를 URIref로 나타냄
- **Predicate or Property (서술부)**
  - Subject의 Resource를 설명하는 속성이나 Resource간의 관계 표현
  - Predicate도 특수한 형태의 Resource이므로 URIref로 명시
- **Object (목적부)**
  - 문장의 목적어 역할
  - Resource로 표현되어 URIref를 명시하거나
  - 혹은 Literal Data 가능(strings, integer..)

2.1 RDF Data Model: Triple

- ‘동건의 나이는 34세이다.’

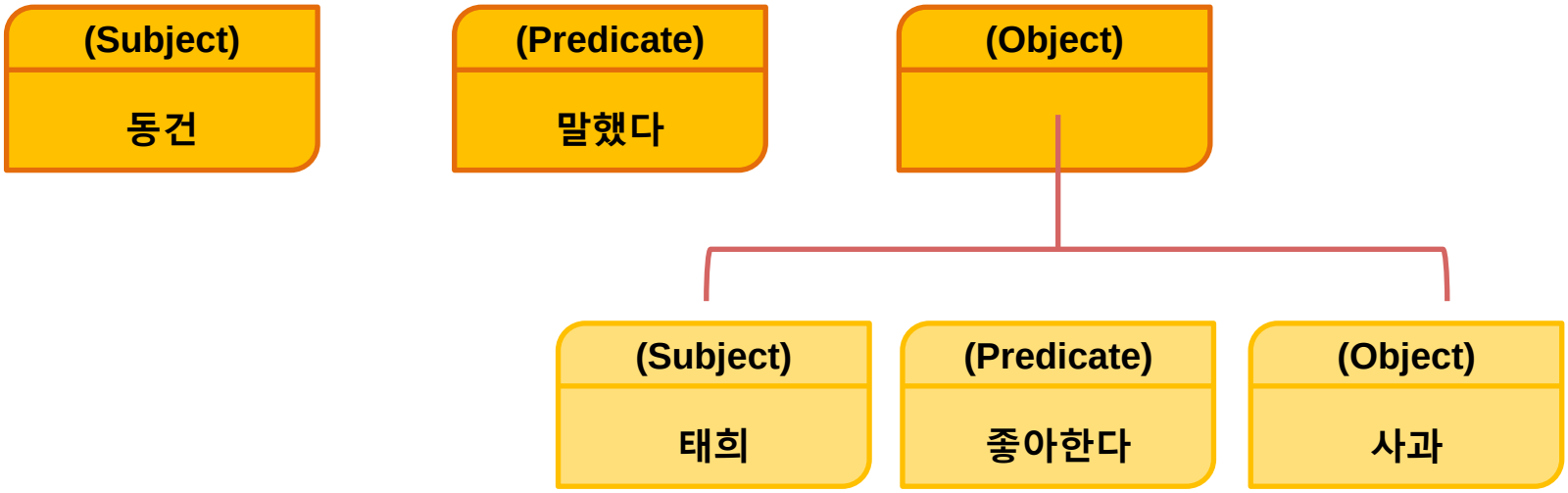
자원 (Subject)	속성 (Predicate)	속성값 (Object)
동건	나이	34

- ‘동건은 경영학을 전공했다.’

자원 (Subject)	관계 (Predicate)	자원 (Object)
동건	전공했다	경영학

2.1 RDF Data Model: Triple

- **Reification (구체화):** 하나의 서술문에 대한 서술문을 추가하여 모델링
- 동건은 태희가 사과를 좋아한다고 말했다.
  - Subject: 동건, Predicate: 말했다, Object: 태희가 사과를 좋아한다



2.1 RDF Data Model: Limitation

- RDF 속성은 양쪽에 인수가 두개인 이진 속성
  - 인수가 두개보다 많은 상황에는 한 번에 표현하기 어려움
- Reification 매커니즘: powerful expression, but complicated
- RDF의 XML 기반 Syntax: computer-friendly, not human-friendly

- But ! RDF는 시맨틱 웹의 기본 계층으로서의 충분한 표현력을 제공
- 다양한 온톨로지 툴들을 이용하면 RDF Syntax를 정확히 몰라도 RDF 편집 가능

2.2 RDF 그래프의 표현방법 (인코딩)

- **Encoding RDF Graphs**
  - **RDF/XML (XML Serialization)**
    - **XML syntax for RDF**
    - **Namespaces in XML, the XML Information Set and XML Base.**
  - ***Notation 3* (also known as *N3*)**
    - **an assertion and logic language which is a superset of RDF**
    - **adding formulae (literals which are graphs themselves), variables, logical implication, and functional predicates**
  - **N-Triple (Turtle Representation)**
    - **line-based, plain text format**
    - **Subset of N3**

2.2 RDF 그래프 코딩 예: RDF/XML

ENTITY (XML의 Entity 선언, ShortCut)

- Entity 정의는 DOCTYPE(Document Type Declaration) 안에 포함
- 긴문장을 지정된 짧은 문장으로 대체
- 사용법: &shortname;

```
<?xml version="1.1">
<!DOCTYPE rdf:RDF [
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#"> ]>

<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" ...
          xmlns:processor="http://www.intel.com/core2duo#">

  <rdf:Description rdf:about="MacBookAir">
    <computer:processor> IntelCore2Duo </computer:processor>
    <computer:weight rdf:datatype="&xsd;decimal"> 3.0 </computer:weight>
  </rdf:Description>

</rdf:RDF>
```

< computer:weight rdf:datatype= http://www.w3.org/2001/XMLSchema#decimal > 3.0</computer:weight>

2.2 RDF 그래프 코딩 예: RDF/XML

■ **Attribute의 value를 위한 shortcut은 Entity 방식**

```
<rdf:Description rdf:about = "http://Ontology.snu.ac.kr/ont-book#Kimdk">
```

⇩

```
<!DOCTYPE rdf : RDF  [
  <!ENTITY ontbook  "http://Ontology.snu.ac.kr/ont-book#"> ]>

<rdf:Description rdf:about="&ontbook;Kimdk">
```

'rdf:about'라는 Attribute에 대한 값을 'ontbook'이란 Entity를 사용해 표현

■ **Tag name 혹은 Attibute name에서의 shortcut은 Qname(Qualified Name) 방식**

- Syntax: [Namespace name]+ [':'] + [Local Name]
-
```
<?xml version='1.0'?>
  <doc xmlns: ontbook = "http://Ontology.snu.ac.kr/ont-book#">
  < ontbook:owns />
 </doc>
```

"ontbook:owns" → http://Ontology.snu.ac.kr/ont-book#owns

2.2 RDF 그래프 코딩 예: RDF/XML

- 예문
  - "김동건은 온톨로지텍이란 회사를 소유하고 있다."
  - "온톨로지텍의 홈페이지 주소는 http://www.ontologytech.com/~ont 이다"
  - "김동건은 경영학을 전공했다."
  - "김동건의 나이는 34세이다."
- RDF Graph

2.2 RDF 그래프 코딩 예: RDF/XML

```
<!DOCTYPE rdf : RDF  [
   <!ENTITY ontbook  "http://Ontology.snu.ac.kr/ont-book#"> ]>
```

```
1. <rdf:RDF
2.    xmlns : rdf = "http://www.x3.org/1999/02/22-rdf-syntax-ns#"
3.    xmlns : ontbook = "http://Ontology.snu.ac.kr/ont-book#">
4.  <rdf:Description rdf:about = "&ontbook; Kimdk">
5.      <ontbook:majorsIn rdf : resource = "&ontbook; Mangement" />
6.      <ontbook:owns>
7.          <rdf:Description rdf:about = "&ontbook ; OntologyTech">
8.              <ontbook : hasHomepage rdf : resource=http://www.Ontologytech.com/~ont/>
9.          </rdf:Description>
10.     </ outbook:owns>
11.    <ontbook:age  rdf : datatype="&xsd; integer" > 34 </ ontbook:age>
12.  </rdf: Description>
13. </rdf: RDF>
```

Attribute value의 축약을 위해 ENTITY선언

Tag name와 attribute name의 축약을 위해 XML namespace선언

정리!!: Short Cut Mechanism in RDF

- **Shortcut of URI: xml:base or ENTITY 선언** — Value Side
  - <rdf:RDF xml:base="http://www.Ontologytech.com/2007/01/products">
    - #piano → http://www.Ontologytech.com/2007/01/products#piano로 인식
  - <!ENTITY ontbook "http://Ontology.snu.ac.kr/ont-book#">
    - &ontbook; Kimdk → http://Ontology.snu.ac.kr/ont-book#Kimdk로 인식

- **Shortcut of Tag name and Attribute name: Name space 선언** — Variable Side
  - xmlns : ontbook = http://Ontology.snu.ac.kr/ont-book#
    - <ontbook:owns> → http://Ontology.snu.ac.kr/ont-book#owns로 인식

- **@prefix obook 'http://Ontology.snu.ac.kr/ont-book/index.html/'** — Turtle만의 기법?
  - obook:swc → http://Ontology.snu.ac.kr/ont-book/index/swc로 인식
  - Prefix문장은 / 혹은 /xyz# 로 끝남

2.2 RDF 그래프 코딩 예: RDF/XML

- 대부분의 RDF문서: 하나의 rdf:RDF와 하나 이상의 rdf:Description들로 구성
  - <rdf:RDF> 안에는 필요한 Name Space(Resource를 정의한 문서)를 선언

```
<rdf:RDF
        xmlns : rdf =“http://www.x3.org/1999/02/22-rdf-syntax-ns#”
        xmlns : ontbook =“http://Http://ontology.sun.ac.kr/ont-book#”>
```

```
<!ENTITY ontbook   “http://Ontology.snu.ac.kr/ont-book#”> ]>
```

  - <rdf:Description> 안에는 하나이상의 서술문이 있고, 중첩가능

```
<rdf:Description  rdf:about = “&ontbook;Kimdk ”>
    <ontbook:majorsIn  rdf:resource = “&ontbook;Mangement” />
     …
    <ontbook:age   rdf:datatype=“&xsd;integer” > 34 </ontbook:age>
</rdf: Description>
```

Note semicolon(;) in “&ontbook;Kimdk

2.2 RDF 그래프 코딩 예: RDF/XML

- <rdf:Description> 는 자원을 지칭하는 attribute인 <rdf:about> 포함
  - 동일한 자원의 정의를 여러 번 할 수는 없고 한 곳에서 자원을 <rdf:ID>로 정의하고 다른 곳에서는 추가적인 속성 기술 가능

- Property Element: <rdf:Description> 의 Child Element
  - 해당 자원이 가지고 있는 속성을 나타냄
  - 이러한 속성에 대한 값은 요소의 내용으로 기록

```
<rdf:Description rdf:about = "&ontbook;Kimdk" >
    <ontbook:majorsIn  rdf:resource = "&ontbook;Mangement"/>
     …
    <ontbook:age  rdf:datatype = "&xsd; integer" > 34 </ontbook:age>
</rdf:Description>
```

2.2 RDF 그래프 코딩 예: RDF/XML

- <rdf:resource> attribute
  - <rdf:about>나 <rdf:ID>처럼 resource를 분명하게 지칭하기 위해 사용

```
<rdf:Description rdf:about = "#000001" >
        <ontbook:name> Kimdk </ontbook:name>
        <ontbook:age rdf:datatype = "&xsd;integer">  34  </ontbook:age>
</rdf:Description>
```

```
<rdf:Description rdf:about = "#COM345" >
        <ontbook:name> OntologyTech </ontbook:name>
        <ontbook:isOwnedBy> Kimdk </ontbook:isOwnedBy>
</rdf : Description>
```

'#000001'의 '김동건'과 '#COM345'의 '김동건'이 동명이인일 가능성이 있다

```
<rdf:Description rdf : about = "#COM345" >
        <ontbook:name> OntologyTech </ontbook:name>
        <ontbook:isOwnedBy  rdf:resource = "#000001" />
</rdf:Description>
```

<rdf:resource>를 사용해 '김동건'이 두개의 description에서 같은 사람이라는 것을 명시적으로 표현가능

2.2 RDF 그래프 코딩 예: RDF/XML

- <rdf:resource>도 URIref를 값으로 취하며 Entity를 활용해 나타낼 수 있다
- <rdf:datatype> attribute: 데이터 타입의 속성 값 범위 지정

```
<rdf:Description rdf:about = “&ontbook;Kimdk”>
    <ontbook:majorsIn rdf:resource = “&ontbook;Mangement” />
     …
    <ontbook:age  rdf:datatype = “&xsd; integer” > 34 </ontbook:age>
</rdf:Description>
```

데이터 타입 age가 integer임을 나타냄

Resource
Description
Framework

Harald Sack **Subject**

has phone number **Property**

++49 (331) 5509-927 **Object**

Resource Description Framework

- **Resources:**

  Objects that can be addressed via **URI**

- **Properties:**

  Attributes for the description of resources

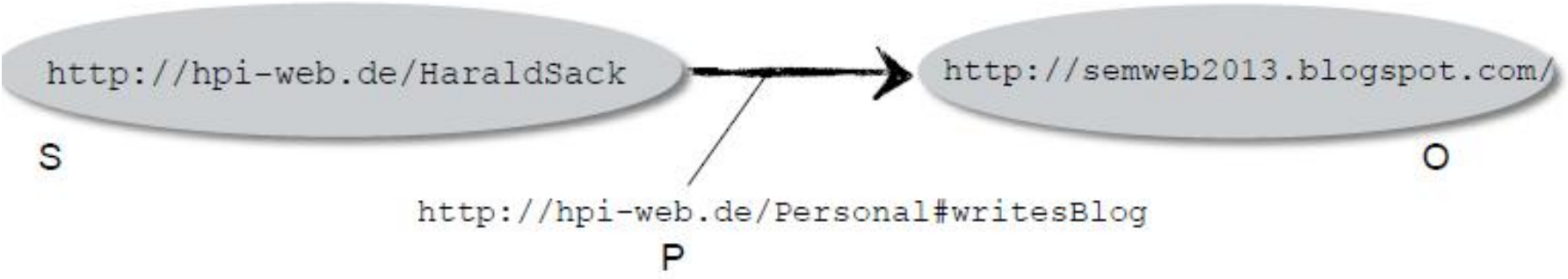
- **Statements (RDF-Triple):**

Resource	+	Property	+	Object / Value
URI		**URI**		**URI / Literal**

Resource Description Framework

Constituents of the RDF-Graph

- **URI**:
  - to reference resources uniquely



Resource Description Framework

Constituents of the RDF-Graphen

- **Literals**:
  - typed literals can be expressed via XML Schema datatypes
  - Namespace for typed literals: `http://www.w3.org/2001/XMLSchema#`
    - Example: `"Semantics"^^<http://www.w3.org/2001/XMLSchema#string>`
  - Language Tags denote the (natural) language of the text:
    - Example: `"Semantik"@de , "Semantics"@en`

Resource Description Framework

RDF Representations

- **Node-Edge-Node Triple**

Resource Description Framework

RDF Representations

- **N3 Notation**
  - Simple listing of triples
    - { http://hpi-web.de/HaraldSack,
      http://hpi-web.de/Personal#writesBlog,
      http://semweb2013.blogspot.com/ }

Resource Description Framework

RDF Representations

- **Turtle (Terse RDF Tripel Language)**
  - Extension of N3
  - URIs in angle brackets
  - Literals in quotation marks
  - Triple ends with a period
  - Whitespaces will be ignored

```
<Subject> <Property> <Object> .
<Subject> <Property> "Object" .
```

- **Tripel: 독어로 삼중의 (= triple)**

Resource Description Framework

RDF Representations

- **Turtle (Terse RDF Tripel Language)**

```
<http://hpi-web.de/HaraldSack>
<http://hpi-web.de/Personal#writesBlog>
<http://semweb2013.blogspot.com/>.

<http://hpi-web.de/HaraldSack>
<http://hpi-web.de/Personal#hasPhoneNumber>
"++49-331-5509-927".
```

Resource Description Framework

RDF Representations

- **Turtle (Terse RDF Tripel Language)**

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .

<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:title "RDF/XML Syntax Specification (Revised)"@en ;
  ex:editor [
    ex:fullname "Dave Beckett"^^xsd:string ;
    ex:homePage <http://purl.org/net/dajobe/>
  ] .
```

Semantic Web Technologies | Dr. Harald Sack, Hasso-Plattner-Institut, Universität Potsdam

- **RDF/XML Syntax Specification (Revised) is edited by Dave Beckett whose homepage is <http://purl.org/net/dajobe/>**

Resource Description Framework

7

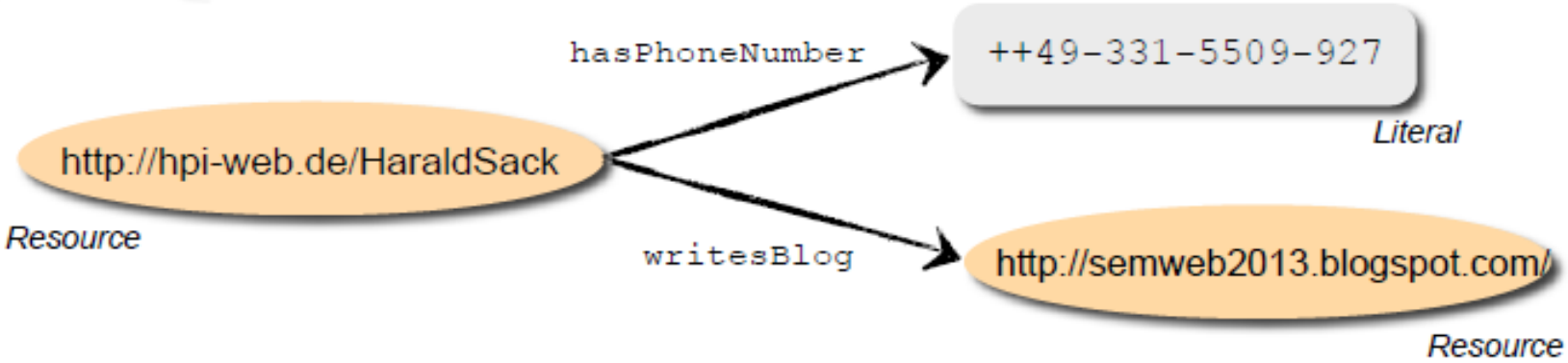
RDF Representations

- **RDF XML-Serialization**

```
<xml version="1.0" encoding="utf-8">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:pers="http://hpi-web.de/Personal#">

 <rdf:Description  rdf:about="http://hpi-web.de/HaraldSack">
  <pers:hasPhoneNr>++49-331-5509-927</pers:hasPhoneNumber>
 </rdf:Description>

 <rdf:Description  rdf:about="http://hpi-web.de/HaraldSack">
  <pers:writesBlog>
   <rdf:Description rdf:about="http://semweb2013.blogspot.com/"></rdf:Description>
  </pers:writesBlog>
 </rdf:Description>
</rdf:RDF>
```

- XML serialization is good for embedding inside HTML web page
- <rdf:Description rdf:about ….> is for URI

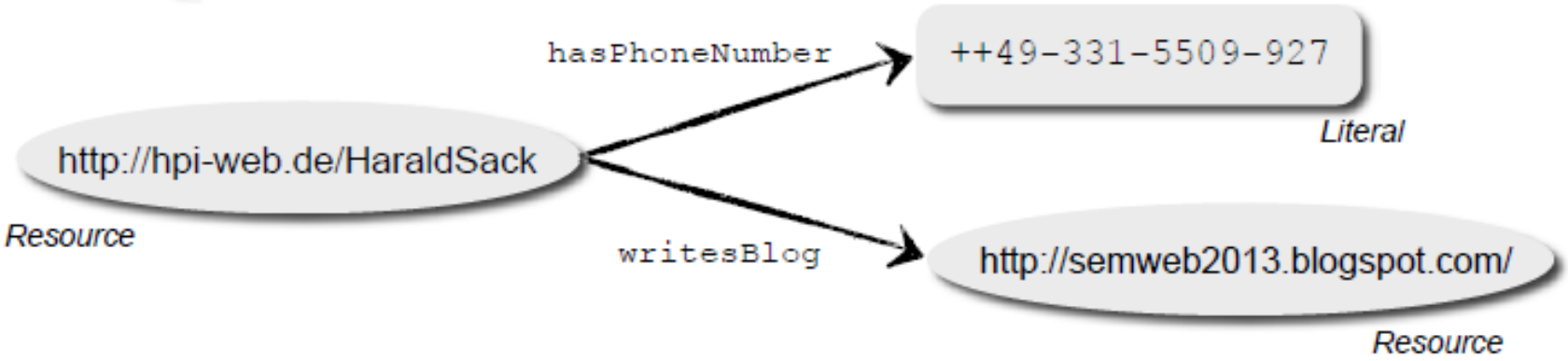
Resource Description Framework




```
<xml version="1.0" encoding="utf-8">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:pers="http://hpi-web.de/Personal#">

 <rdf:Description rdf:about="http://hpi-web.de/HaraldSack">
  <pers:hasPhoneNumber>++49-331-5509-927</pers:hasPhoneNumber>
 <rdf:Description>

 <rdf:Description rdf:about="http://hpi-web.de/HaraldSack">
  <pers:writesBlog rdf:resource="http://semweb2013.blogspot.com/" />
 </rdf:Description>
</rdf:RDF>
```

- **Object's URI <Rdf:description ref:about …> can be replaced with rdf:resource attribute in a property in XML serialization**

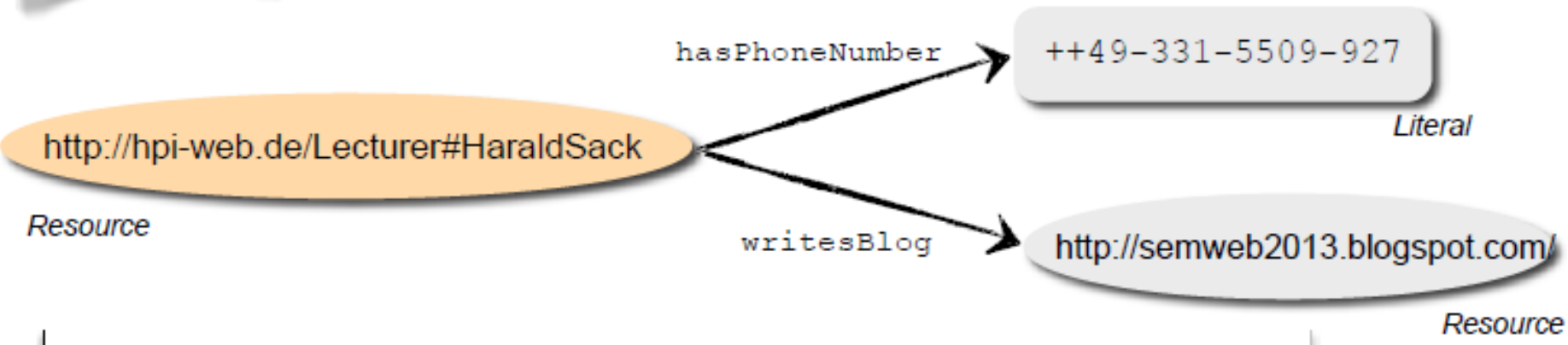
Resource Description Framework




```
<xml version="1.0" encoding="utf-8">
<rdf:RDF   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:pers="http://hpi-web.de/Personal#">

 <rdf:Description  rdf:about="http://hpi-web.de/HaraldSack"
                   pers:hasPhoneNumber="++49-331-5509-927">
  <pers:writesBlog rdf:resource="http://semweb2013.blogspot.com/"/>
 </rdf:Description>
</rdf:RDF>
```

- **Making a phone number (O) as an attribute of S and a blog (O) as an attribute of P respectively in XML serialization**

Resource Description Framework

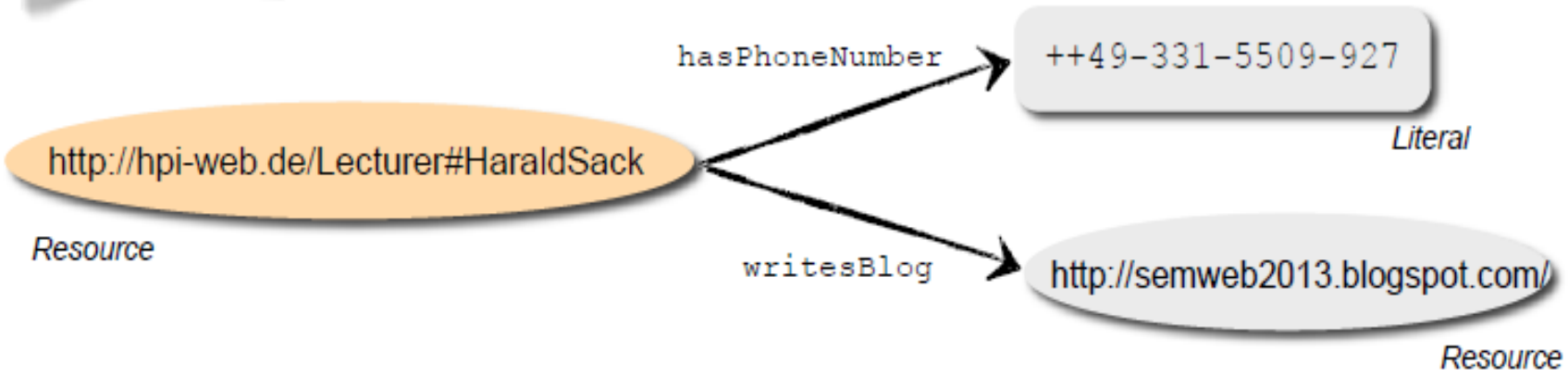



```
<xml version="1.0" encoding="utf-8">
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:pers="http://hpi-web.de/Personal#"
         xml:base="http://hpi-web.de/Lecturer">

  <rdf:Description  rdf:about="#HaraldSack"
                    pers:hasPhoneNumber="++49-331-5509-927">
    <pers:writesBlog rdf:resource="http://semweb2013.blogspot.com/"/>
  </rdf:Description>
</rdf:RDF>
```

- **Making a base prefix in XML serialization**

Resource Description Framework

9

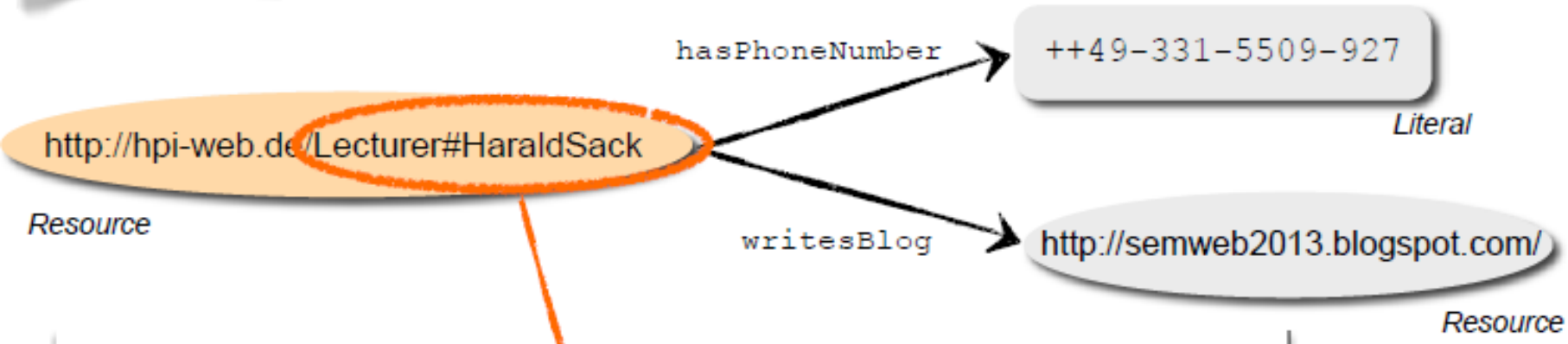



```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix pers: <http://hpi-web.de/Personal#> .
@base <http://hpi-web.de/Lecturer> .

:HaraldSack pers:hasPhoneNumber "++49-331-5509-927" ;
            pers:writesBlog <http://semweb2013.blogspot.com/> .
```

Short Cut

- Making prefixes in Turtle representation
- The base prefix can be omitted as in ":HaraldSack"

Resource Description Framework

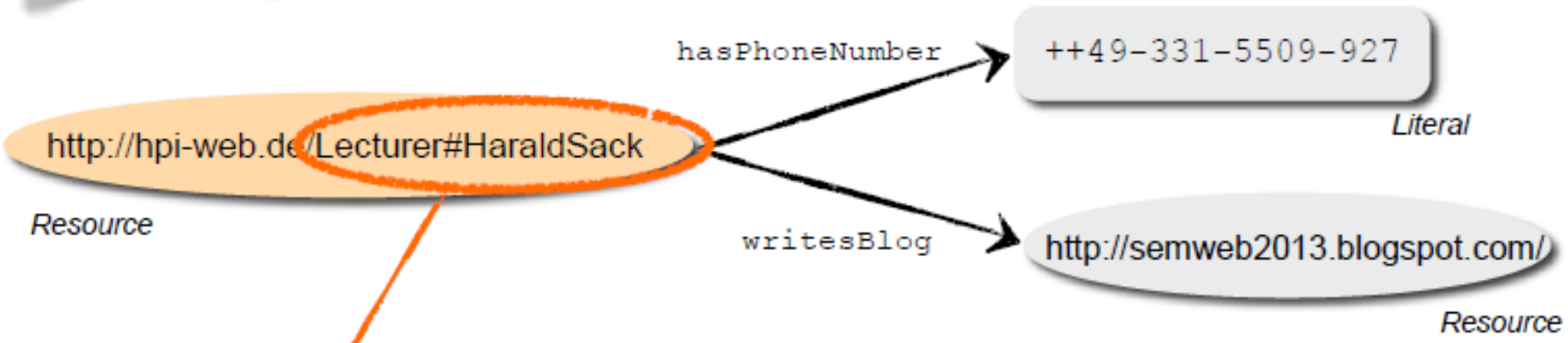



```
<xml version="1.0" encoding="utf-8">
  <rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:pers="http://hpi-web.de/Personal#"
            xml:base="http://hpi-web.de/Lecturer"

  <rdf:Description rdf:ID="HaraldSack"
                   pers:hasPhoneNumber="++49-331-5509-927">
    <pers:writesBlog rdf:resource="http://semweb2013.blogspot.com/"/>
  </rdf:Description>
</rdf:RDF>
```

Datei http://hpi-web.de/Dozenten.rdf

- rdf:ID="HaraldSack" is always extended with the base prefix in XML serialization

Resource Description Framework



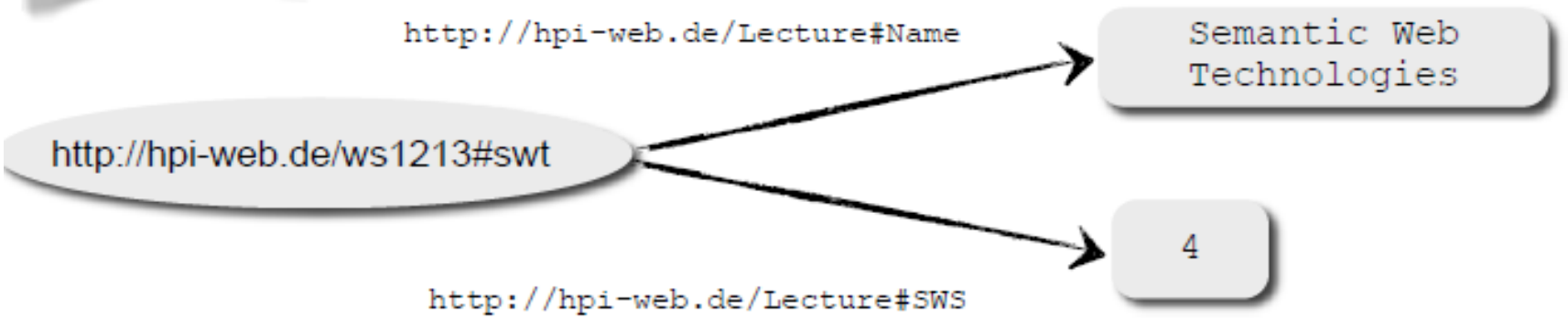

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix pers: <http://hpi-web.de/Personal#> .

:HaraldSack pers:hasPhoneNumber "++49-31-5509-927" ;
            pers:writesBlog <http://semweb2013.blogspot.com/> .
```

Datei http://hpi-web.de/Lecturer.rdf

- In Turtle representation, even a base prefix can be omitted

```
@base <http://hpi-web.de/Lecturer> .
```

Resource Description Framework




```
<xml version="1.0" encoding="utf-8">
<rdf:RDF   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns:lv="http://hpi-web.de/Lecture#" >

 <rdf:Description  rdf:about="http://hpi-web.de/ws1213#swt">
  <lv:Name rdf:datatype="http://www.w3c.org/2001/XMLSchema#string">
    Semantic Web Technologies
  </lv:Name>
  <lv:SWS rdf:datatype= "http://www.w3c.org/2001/XMLSchema#integer"> 4 </lv:SWS>
 </rdf:Description>
</rdf:RDF>
```

- **XML schema meta-data can be included using rdf:datatype**

Resource Description Framework

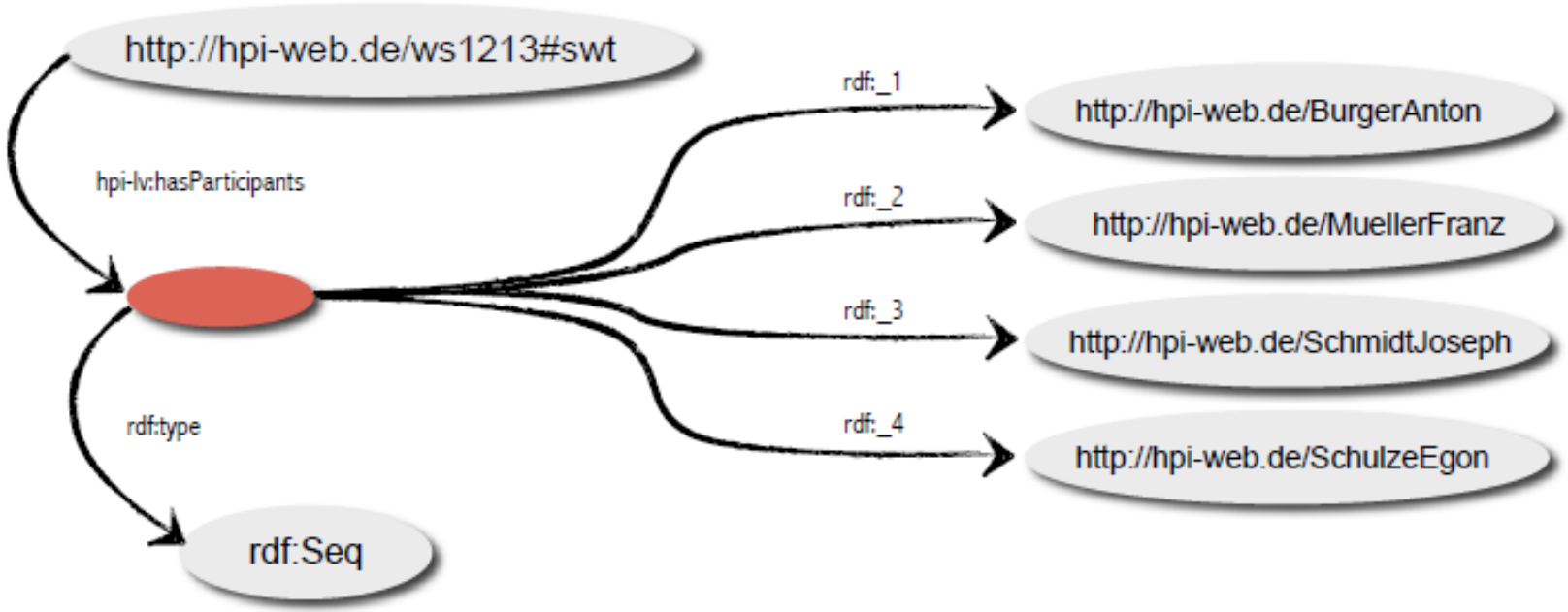
Lists

- General Data structure to enumerate any resoures or literals
- Only shortcuts, no additional semantic expressivity

- Distinguish between
  - **Container**
    open list, i.e. extension (new entries) possible
  - **Collections**
    closed list, i.e. no extension possible

Resource Description Framework

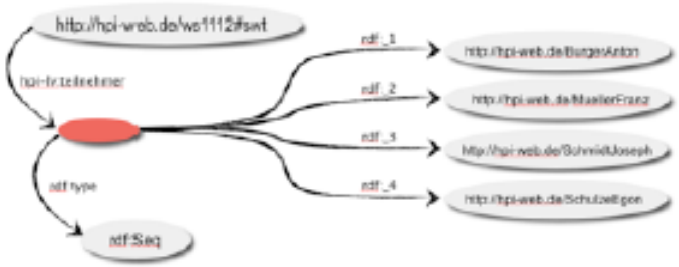
RDF Container



@prefix hpi-lv: <http://hpi-web.de/Lecture#> .
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

Resource Description Framework

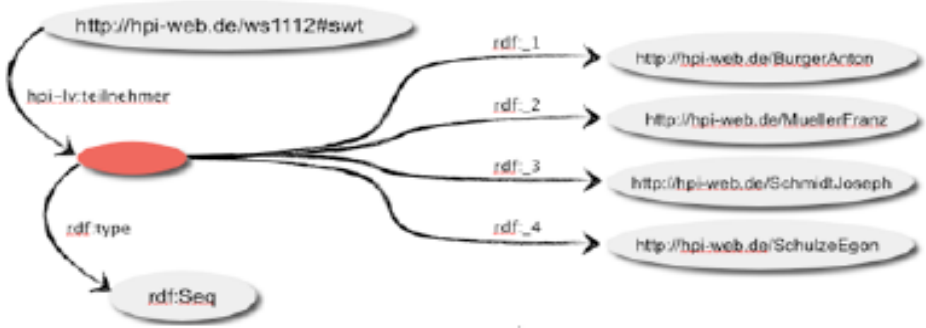
RDF-Container



- the root node of the container is assigned a container-typ via rdf:type
  - **rdf:Bag**
    unordered set of elements,
    there is no given order of elements
  - **rdf:Seq**
    ordered set of elements,
  - **rdf:Alt**
    defines alternatives of elements
    only one element of the given alternatives is relevant for the application

Resource Description Framework

RDF Container



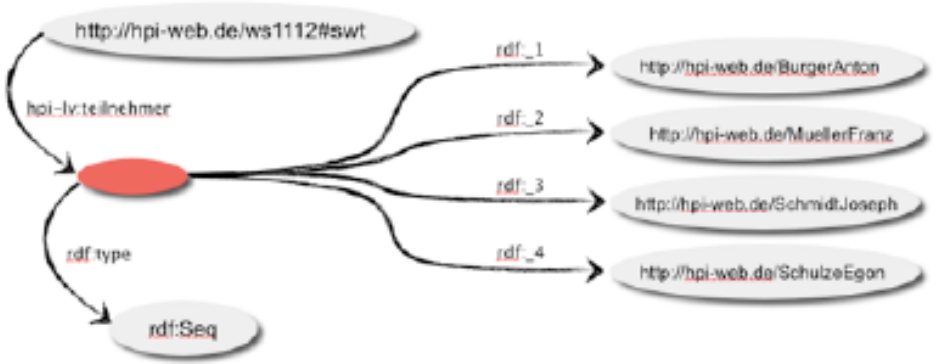

```
<rdf:Description  rdf:about="http://hpi-web.de/ws1213#swt">
   <lv:hasParticipants>
    <rdf:Seq>
     <rdf:li rdf:resource="http://hpi-web.de/BurgerAnton" />
     <rdf:li rdf:resource="http://hpi-web.de/MuellerFranz" />
     <rdf:li rdf:resource="http://hpi-web.de/SchmidtJoseph" />
     <rdf:li rdf:resource="http://hpi-web.de/SchulzeEgon" />
    </rdf:Seq>
   </lv:hasParticipants>
 </rdf:Description>
```

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

- **XML Serialization**

Resource Description Framework

RDF Container



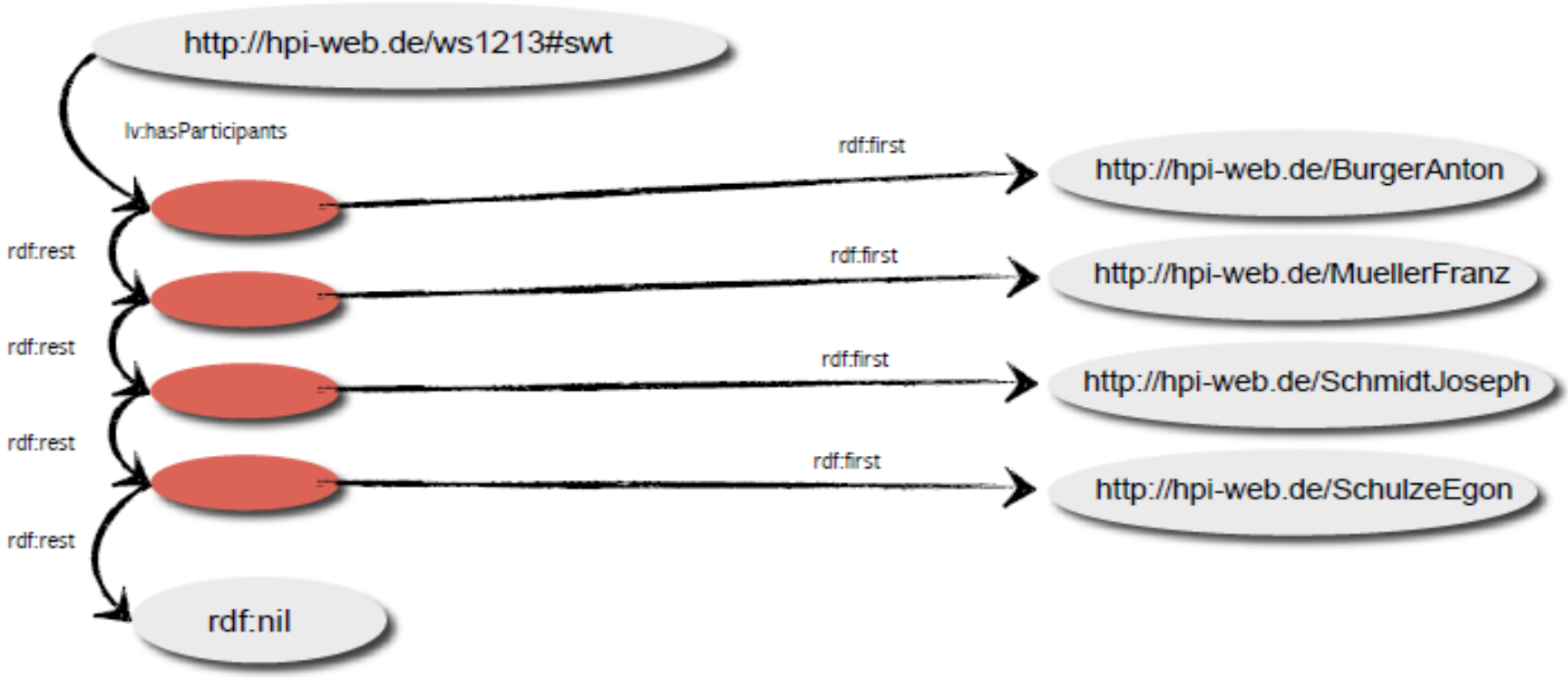

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix lv: <http://hpi-web.de/Lecture#> .
@base <http://hpi-web.de/>

:swt lv:hasParticipants [
  a rdf:Seq;
  rdf:_1 <BurgerAnton>;
  rdf:_2 <MuellerFranz>;
  rdf:_3 <SchmidtJoseph>;
  rdf:_4 <SchulzeEgon> .
] .
```

- **Turtle Representation**

Resource Description Framework

RDF-Collection

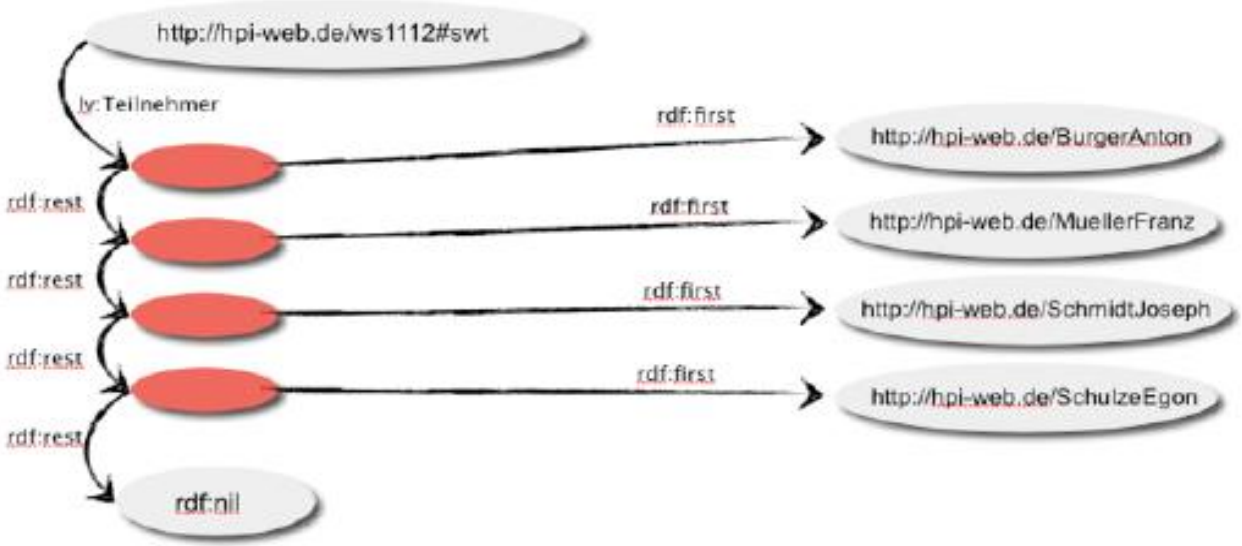
```
@prefix hpi-lv: <http://hpi-web.de/Lecture#> .
@prefix rdf: http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```



List is splitted recursively in Head (first) and Tail (rest).

Resource Description Framework

RDF-Collection

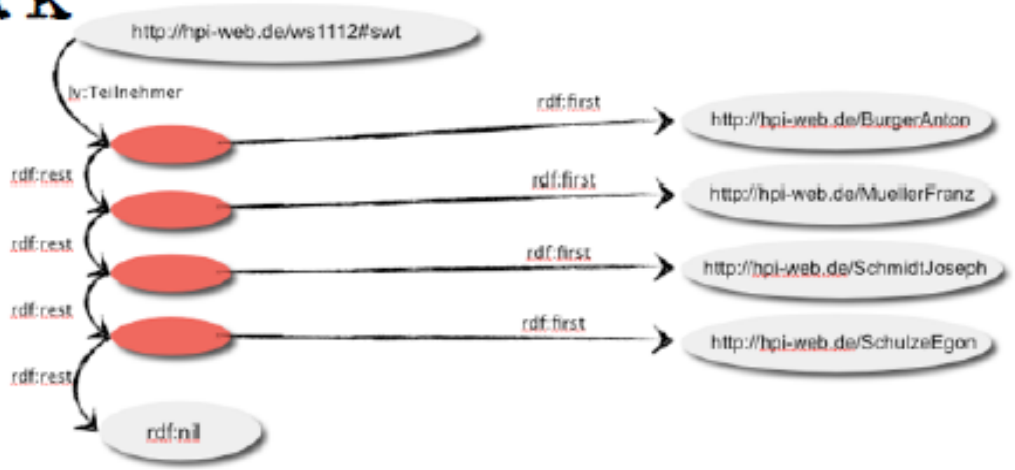



```
<rdf:Description  rdf:about="http://hpi-web.de/ws1213#swt">
    <lv:hasParticipant rdf:parseType="Collection">
      <rdf:Description rdf:about="http://hpi-web.de/BurgerAnton" />
      <rdf:Description rdf:about="http://hpi-web.de/MuellerFranz" />
      <rdf:Description rdf:about="http://hpi-web.de/SchmidtJoseph" />
      <rdf:Description rdf:rabout="http://hpi-web.de/SchulzeEgon" />
    </lv:hasParticipant>
 </rdf:Description>
```

- **XML Serialization**

Resource Description Framework

RDF-Collection

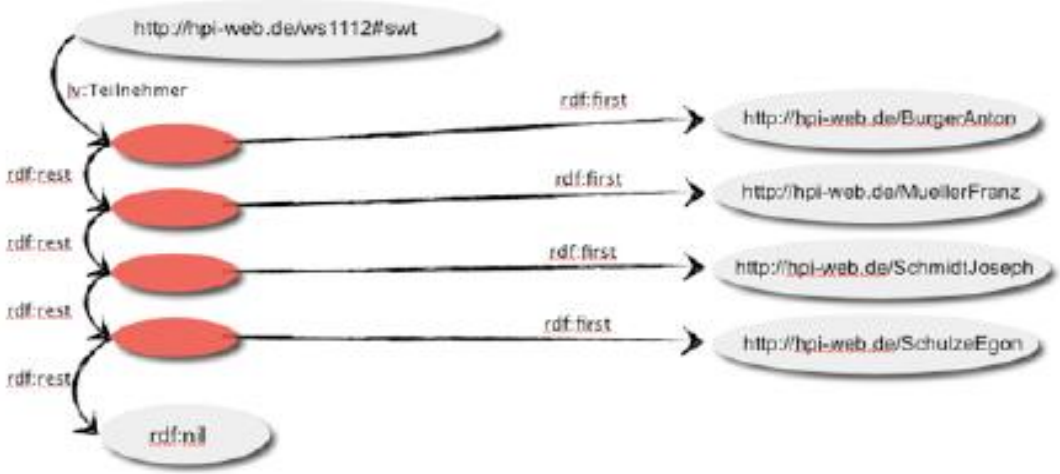



```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix lv: <http://hpi-web.de/Lecture#> .
@base <http://hpi-web.de/>

:swt lv:hasParticipant [
  rdf:first <BurgerAnton>; rdf:rest [
  rdf:first <MuellerFranz>; rdf:rest [
  rdf:first <SchmidtJoseph>; rdf:rest [
  rdf:first <SchulzeEgon>;
  rdf:rest  rdf:nil .
] ] ] ] .
```

- **Turtle representation**

Resource Description Framework

RDF-Collection




```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix lv: <http://hpi-web.de/Lecture#> .
@base <http://hpi-web.de/>

:swt lv:hasParticipant
( <BurgerAnton> <MuellerFranz> <SchmidtJoseph> <SchulzeEgon>) .
```

- **Turtle representation with shortcut**

Resource Description Framework

추가자료

RDF-Reification

- RDF permits interleaving of statements, i.e. to make statements about statements
- Example:
  - **Sherlock Holmes supposes that the Gardener has killed the Butler**
    - Part 1: The Gardener has killed the Butler

```
exv:Gardener  exv:hasKilled  exv:Butler .
```

    - Part 2: Sherlock Holmes supposes

```
exv:SherlockHolmes  exv:supposes  ???? .
```

- **Reification = 구체화**

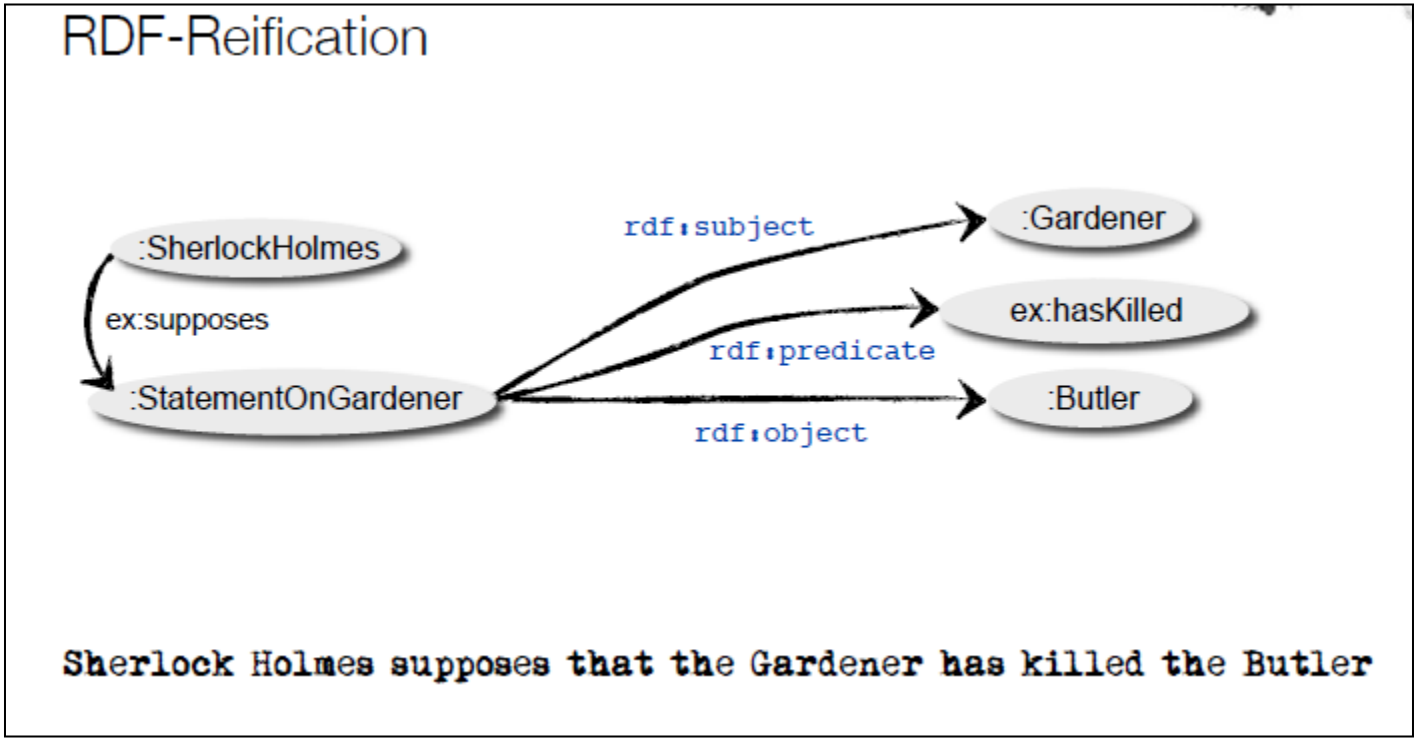
RDF-Reification

추가자료

- **rdf:Statement**

  defines an RDF Statement, consisting of Subject, Predicate and Object

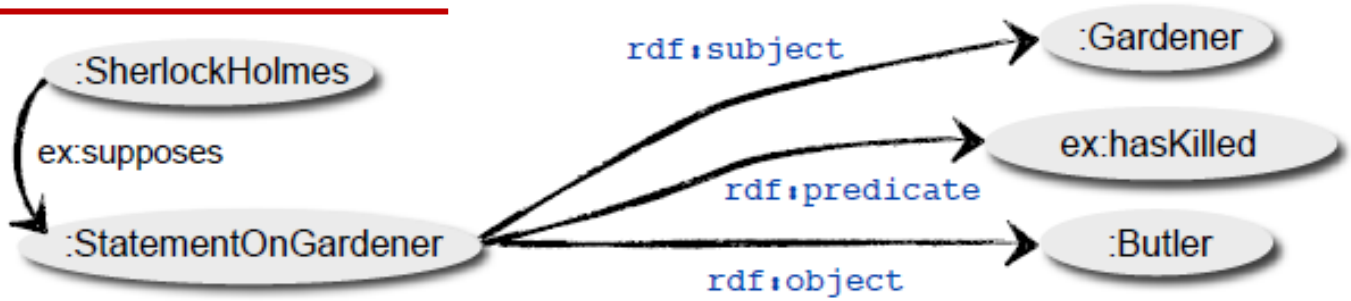
  - **rdf:subject** - the described resource
  - **rdf:predicate** - the original property
  - **rdf:object** - the value of the property



Resource Description Framework

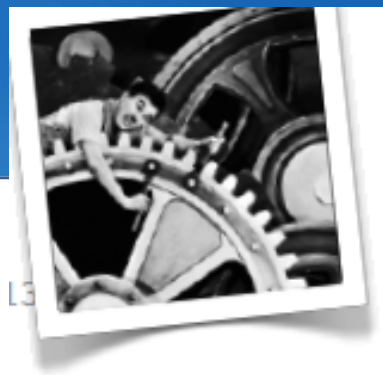
추가자료

RDF-Reification




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

:SherlockHolmes ex:supposes :StatementOnGardener .
:StatementOnGardener a rdf:Statement ;
    rdf:subject :Gardener ;
    rdf:predicate ex:hasKilled ;
    rdf:object :Butler .
```

- **Turtle representation**



RDF and Data Integration

추가자료

- Simple Example: Bibliography Database

Books

ID	Author	Title	Publisher	Year
ISBN 0-00-651409-X	HS-123	WWW	S-001	2004

Authors

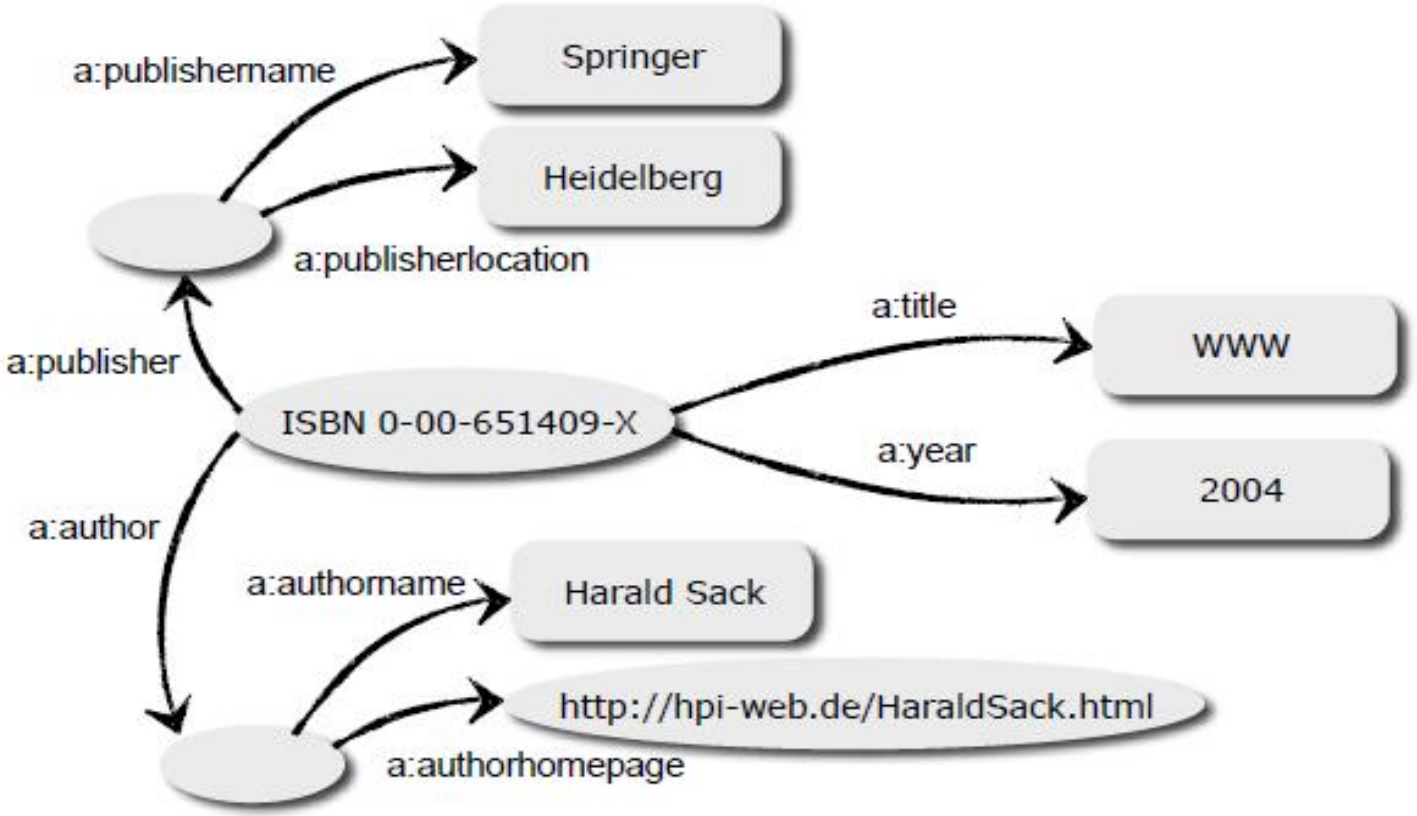
ID	Name	Homepage
HS-123	Harald Sack	http://hpi-web.de/HaraldSack.html

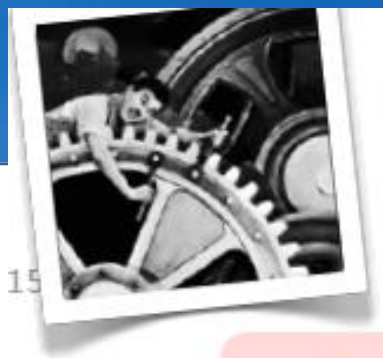
Publishers

ID	Publisher	Location
S-001	Springer	Heidelberg

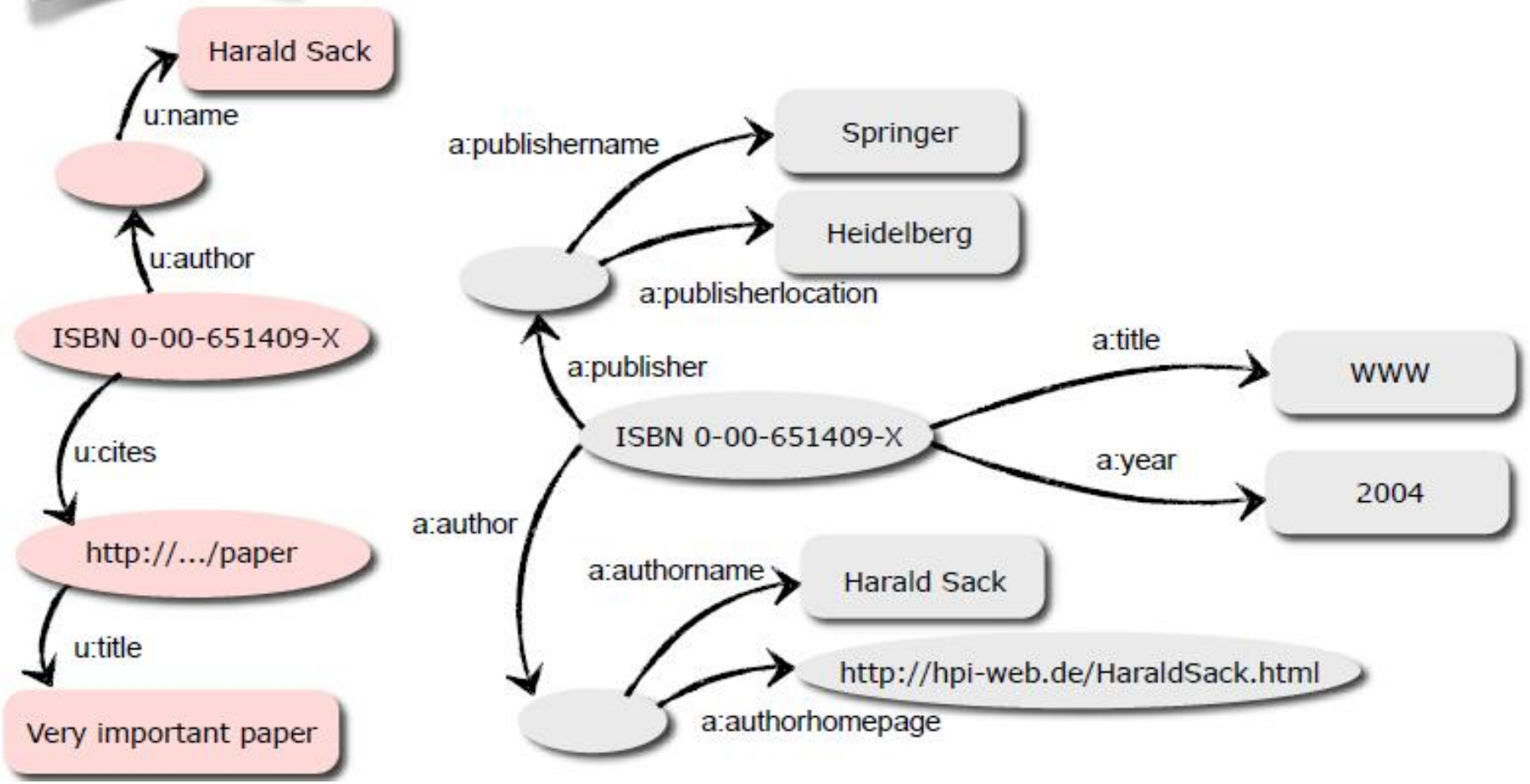
RDF and Data Integration

- Database export into a set of relations





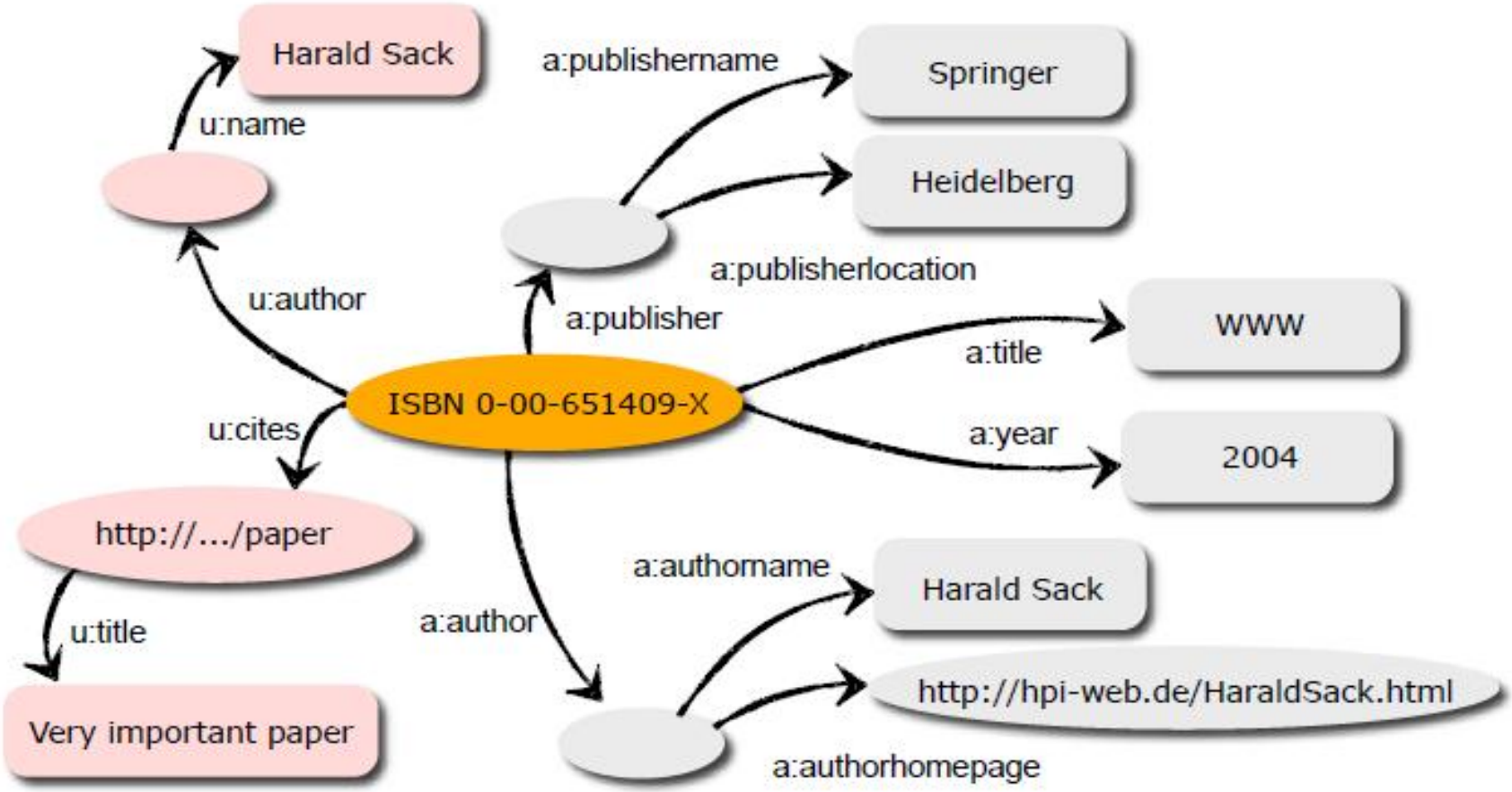
RDF and Data Integration



RDF and Data Integration

RDF and Data Integration

추가자료



Chapter 6 RDF(S): RDF와 RDF Schema

- 1. XML과 RDF
- 2. RDF
  - 2.1 RDF 데이터 모델
  - 2.2 RDF 그래프와 코딩 예
- 3. RDF Schema
  - 3.1 RDF와 RDF Schema
  - 3.2 기본 요소: 클래스와 속성
  - 3.3 RDF Schema 코딩 예
- 4. RDF(S)의 한계점

3.1 RDF와 RDF Schema

- RDF
  - 속성(attribute)의 domain을 제한하지 못함
  - 비슷한 자원을 한 class로 묶어 표현하는 기능 없음
  - 트리플 구조의 실질적 의미를 컴퓨터가 정확하게 이해 할 수 없다
    - “김동건은 학생이다”, “학생은 사람이다”
    - 컴퓨터는 김동건이 사람이라는 사실을 Reasoning(추론)할 수 없다.

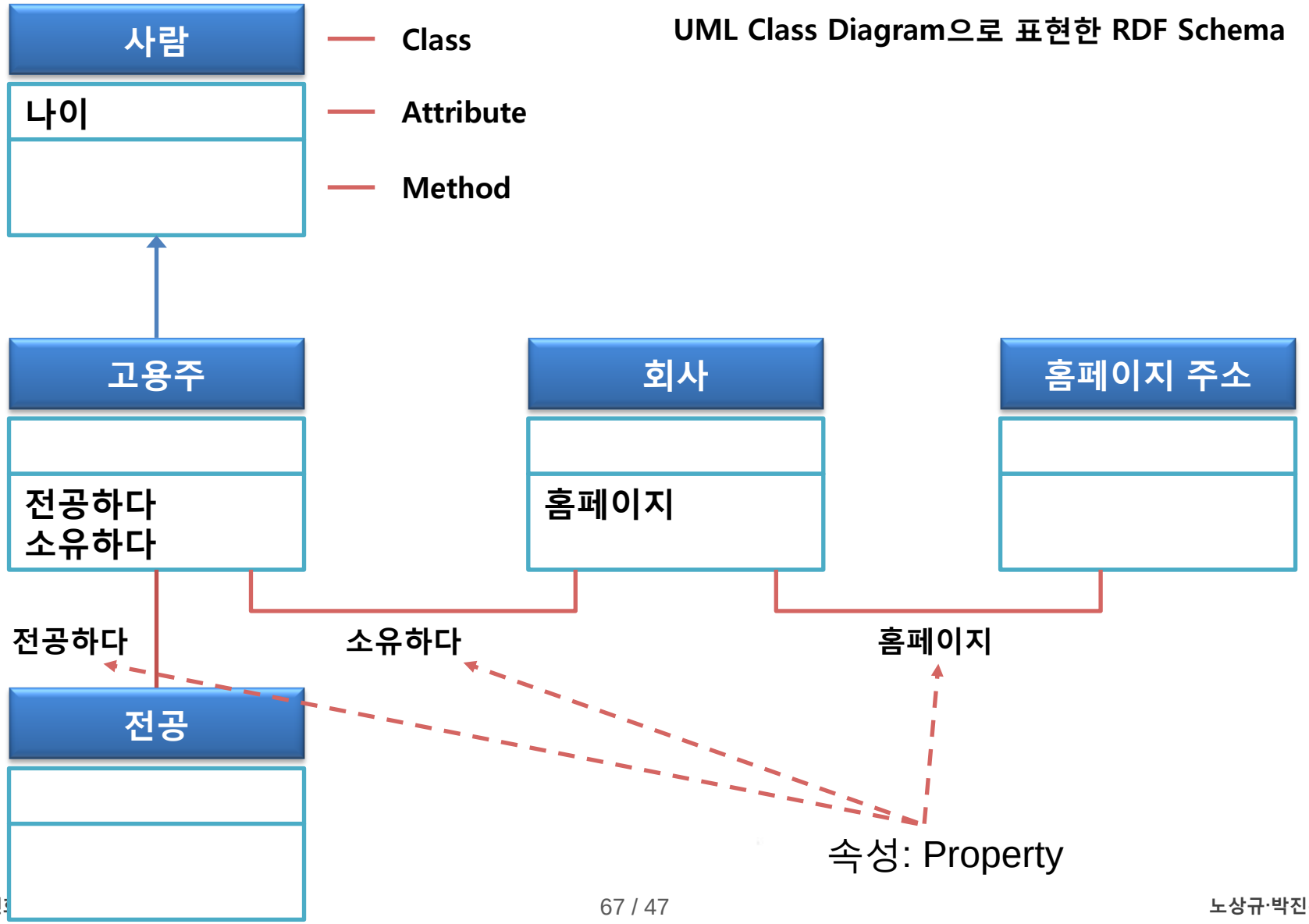
- RDF Schema
  - RDF를 프레임기반으로 확장, 2004년 2월 W3C 권고안으로 발표
  - 도메인에 필요한 어휘와 기본 가정들 정의 가능 (Brickly and Guha, 2004)
  - OO Programming Language의 데이터 모델과 유사
    - 클래스 상속 개념 지원

3.1 RDF와 RDF Schema

- **RDF Schema와 객체지향 모델링의 차이점 : 속성을 다루는 방법**

RDF Schema	Object Oriented Modeling
Property를 Class와 독립적으로 정의 Property가 온톨로지 전 범위에 걸쳐 유효	클래스 정의 안에 속성 포함
Class 정의 수정없이 New Property 적용가능	New Property 추가하려면 Class 정의까지 수정해야 함
Property에 대해, Subject가 될 수 있는 클래스(domain)와 Object로 올 수 있는 클래스(range)를 명시	

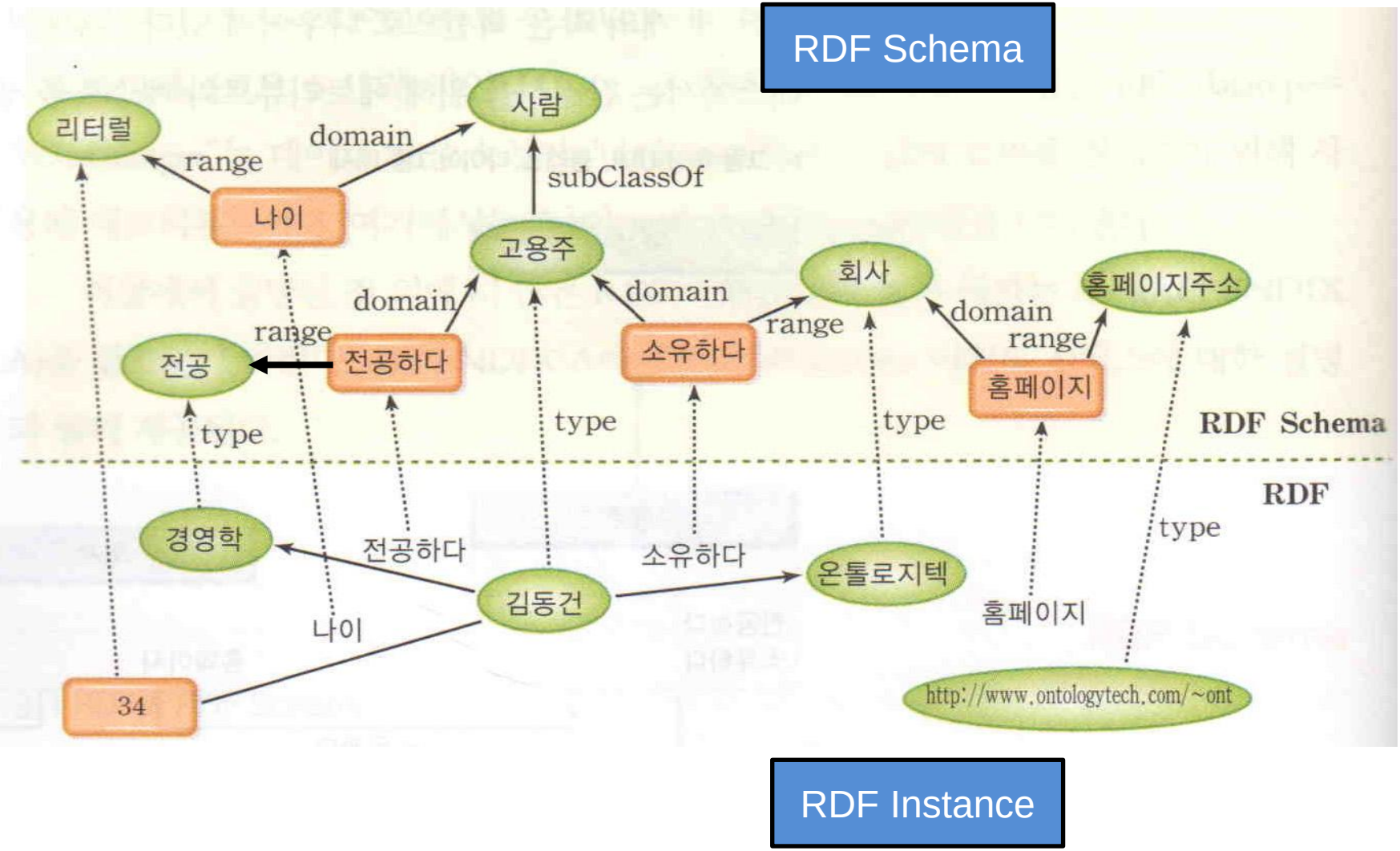
3.1 RDF Schema by UML Class Diagram



3.1 RDF Instance와 RDF Schema

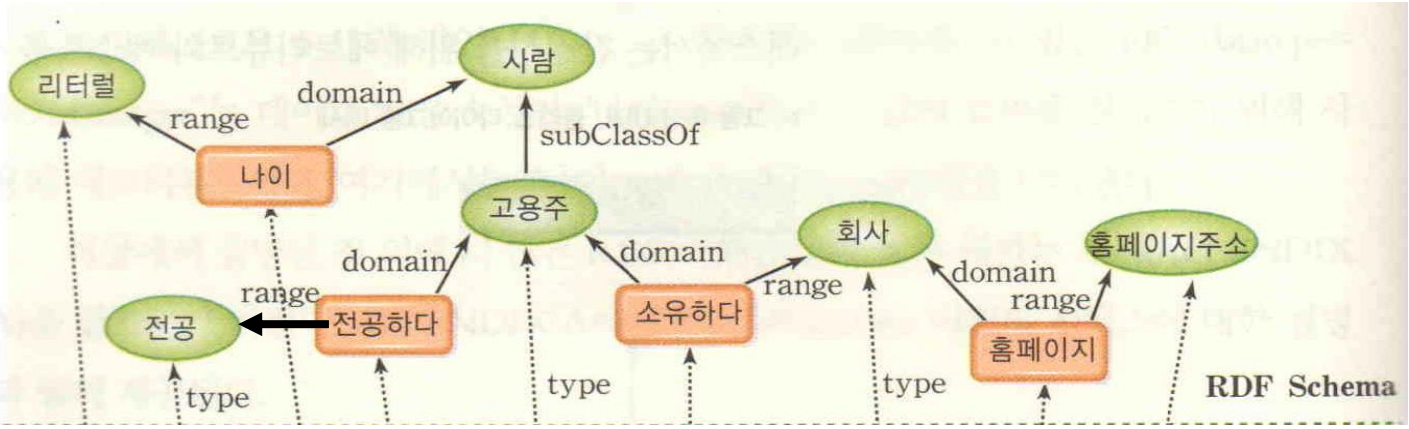
** 아래와 같은 RDF Graph는 RDF Schema에 대한 Instance 정보라고 할 수 있다

3.1 RDF Instance와 RDF Schema



3.1 RDF와 RDF Schema

- RDF Schema의 역할
  - 해당 도메인을 기술하기 위해 필요한 어휘를 정의
  - Property의 Doman과 Range를 정의
  - Resource 및 Property 간의 계층구조를 포함하는 다양한 관계를 정의
- 예: 만약 "서태희는 의류디자인학을 소유하고 있다"라는 RDF/XML이 있다면
  - RDF Schema의 "소유하다"에 대한 Constraint로 이 문장의 잘못을 탐지가능
    - Property "소유하다": Domain은 '고용주', Range는 '회사'로 규정
    - 의류디자인학'은 회사가 아니므로 '소유하다'의 Range가 될 수 없음

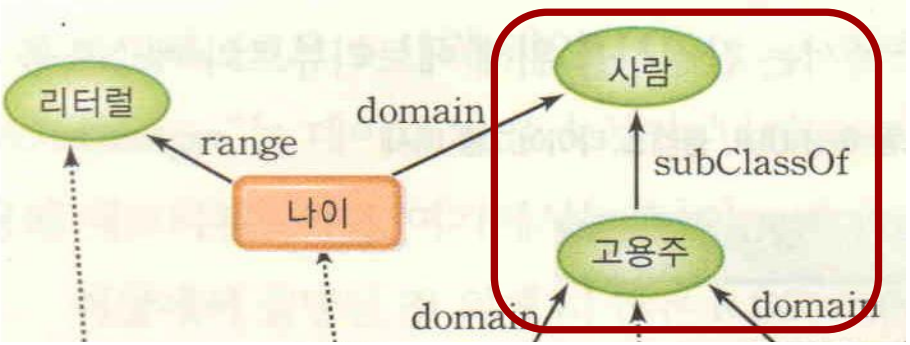


3.1 RDF와 RDF Schema

- XML에서 아래 표현에 대해서 모든 '사람'을 검색하면 두사람만 검색

```
<사람> 서태희 </사람>
<사람> 오혜수 </사람>
<고용주 이름="김동건">
        <소유하다> 온톨로지텍</소유하다>
</고용주>
```

- RDF Schema가 있으면 세사람이 검색



- '고용주'는 '사람'의 subClassOf의 Semantics에 의해
  - "모든 고용주 Instance는 사람의 Instance"

3.2 기본 요소: Class and Property

- 3. RDF Schema
  - 3.1 RDF와 RDF Schema
  - 3.2 기본 요소: 클래스와 속성
    - 3.2.1 클래스 (Class)
    - 3.2.2 속성 (Property)
  - 3.3 RDF Schema 코딩 예

3.2.1 Class and Subclass in RDF Schema

- 동일한 Property들을 가지는 개체들의 그룹
  - 예) A,B,C가 서울대 경영학과에서 MIS전공 대학원생이라는 공통의 특징 → '서울대 경영학과 MIS 전공 대학원생' Class 생성

- <rdfs:subClassOf> tag: 하위 클래스를 상위클래스와 연결
  - 하위 클래스에 속한 Instance는 자동으로 상위 클래스에 속함
  - 하나의 하위 클래스는 여러 개의 상위 클래스를 가질 수 있음
  - <rdfs : Class>와 </rdfs : Class> 사이에서만 쓰임
  - 예) 'LaserPrinter'가 'Printer'의 하위 클래스일때,

```
<rdfs:Class  rdf:ID = "LaserPrinter">
   <rdfs:subClassOf    rdf:resource = "#Printer" />
</rdfs:Class>
```

3.2.2 Property in RDF Schema

- Class 와 Class 간의 관계 (↔ RDF의 Property는 Resource와 Resource 간의 관계)
- Class 개념을 통해 모든 개체를 효율적으로 묶어 표현 가능
- Class는 다른 Class와 관계를 형성, 풍부한 지식표현 가능

- 예) 교수 Class – '김동건', '서태희', '오혜수'
  강좌 Class – '경영학개론', '온톨로지개론', '시맨틱웹'

  '가르친다' Property를 Domain '교수' 와 Range '강좌'로 정의하면
  ➔ '오혜수 교수가 시맨틱웹 강좌를 가르친다' 등의 새로운 의미 형성 가능

3.2.2 Property: 정의역(Domain), 공역(Range)

- Property의 Subject와 Object에 올 수 있는 Class의 범위를 지정
  - Subject : Property가 값을 취하는 domain class <rdfs : domain>
  - Object : 그 propert가 값을 취하는 range class <rdfs : range>

```
<rdf:Property  rdf:ID  = "제조되다">
        <rdfs:domain rdf:resource = "#프린터"  />
        <rdfs:range  rdf:resource = "#제조회사" />
</rdf:Property>
```

- 특정 Class를 Domain이나 Range로 지정하지 않으면 모든 Class를 범위로 인식

```
<rdf : Property rdf : ID = "제조되다" />
```

- 하나의 Property에 대해,
  - Domain과 Range는 하나씩만 정의 가능
  - 한 번 지정된 Domain/Range를 번복안됨
  - 다른 Domain/Range 추가할 수 없음

3.2.2 Property and Subproperty: Hierarchy

- <rdfs : subPropertyOf> : Property간 계층관계 표현
- Subproperty는 상위 Property의 Domain/Range를 자신의 Domain/Range로 인식

```
<rdf:Property rdf:ID = "isAdaughterOf">
        <rdfs : subPropertyOf  rdf:resource = "#isAChildOf">
</rdf:Property>
```

'~의 딸이다'(isAdaughterOf)라는 속성은

'~의 자식이다'(isAChildOf)라는 속성의 SubProperty

3.3 RDF Schema 코딩 예

- 3. RDF Schema
  - 3.1 RDF와 RDF Schema
  - 3.2 기본 요소: 클래스와 속성
    - 3.2.1 클래스 (Class)
    - 3.2.2 속성 (Property)
  - 3.3 RDF Schema 코딩 예

3.3 RDF Schema 코딩 예

■ RDF Schema 주요 어휘 (http://www.w3.org/2000/01/rdf-schema)

어휘	설명
rdfs : Class	클래스를 정의하는 Element Property들을 클래스에 할당하기 위해 rdf : Property, rdfs : range, rdfs : domain을 함께 사용 클래스 Identifier로서 attribute <rdf:about> 에 URIref를 써 줌
rdfs : label	클래스에 사람이 이해할 수 있는 라벨을 붙여주는 attribute
rdfs :subclassOf	한 클래스가 다른 기존 클래스의 하위 클래스임을 명시하는 요소
rdf : Property	클래스의 Property를 정의하는 Element rdfs : range, rdfs : domain과 함께 사용됨
rdfs : domain	미리 정의된 클래스를 값으로 가짐, 한 속성이 클래스에 속하는지를 정의하는 attribute(주어부, 정의역)
rdfs : range	미리 정의된 클래스를 값으로 가짐, 한 속성이 취할 수 있는 값의 범위를 정의하는 attribute(목적부, 공역)
rdfs : Literal	문자열이나 정수 같은 상수 값들로 이루어진 클래스

3.3 RDF Schema 코딩 예: Class 부분

```
<?xml version="1.1“ encoding=“euc-kr” ?>
<!DOCTYPE rdf:RDF [
        <!ENTITY rdfs “http://www.w3.org/2000/01/rdf-Schema#” >
        <!ENTITY rdf “http://www.w3.org/1999/02/22-rdf-syntax-ns#” >
]>

<rdf : RDF xmlns : rdf   = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
           xmlns : rdfs = “http://www.w3.org/2000/01/rdf-schema#”
           xml : base    = “http://Ontology.snu.ac.kr/ont-book” >


        <rdfs : Class rdf : ID = “Person“>
                <rdfs : label=“사람” />
                <rdfs : subClassOf  rdf : resource=“&rdfs ; Resource” />
        </rdfs : Class>


        <rdfs : Class rdf : ID = “Employer”>
                <rdfs : label>고용주</rdfs : label>
                <rdfs : subClassOf  rdf : resource=“#Person” />
        </rdfs : Class>


         <rdfs : Class rdf : ID = “Company”>
                <rdfs : label=“회사” />
                <rdfs : subClassOf  rdf : resource=“&rdfs ; Resource” />
        </rdfs : Class>

         …
```

Entity 선언

Name space 선언

Base URI 선언

Base URI인 "http://Ontology.snu.ac.kr/ont-book" 과 "#"을 붙인
Absolute URIref "http://Ontology.snu.ac.kr/ont-book#Person"을 의미

3.3 RDF Schema 코딩 예: Property 부분

```
<?xml version="1.1“ encoding = “euc-kr” ?>
…

        <rdfs : Class rdf : ID = “Person“>
                …
        </rdfs : Class>

        …


        <rdf : Property rdf : ID = “owns">
                <rdfs : label=“소유하다” />
                <rdfs : domain  rdf : resource="#Employer" />
                <rdfs : range    rdf : resource="#Company" />
        </rdf:Property>

        <rdf : Property rdf : ID = “age">
                <rdfs : label=“나이” />
                <rdfs : domain  rdf : resource="#Person" />
                <rdfs : range    rdf : resource=“&rdfs ; Literal" />
        </rdf:Property>

         …

</rdf : RDF>
```

UML에서 Attribute 였던 “Age”/ Method 였던 “소유하다”가
RDF Schema에서는 모두 Property로 표현

4. RDF(S): RDF & RDF Schema

- RDF와 RDFS는 현실 세계의 여러 개념을 표현할 수 있는 방법 제공
- RDF
  - Resource – Predicate(Property) – Resource의 간단한 Triple 형태
    - Resource를 연결한것에 불과해 Resource 간의 의미적관계의 표현에는 한계
      - (“김동건은 학생이다”, “학생은 사람이다” ➔ 김동건이 사람이란 추론 불가)
- RDF Schema (RDF의 한계를 극복!)
  - 객체들를 class로 정의하고 class간의 Property를 기술하고 계층구조도 설정
  - Class간의 Property가 Subject와 Object에 어떤 값을 취하는지 Domain/Range
  - 메타데이터의 속성과 클래스 간의 관계 표현이 가능

- 그러나 아직도 정보의 의미를 표현하는 데에 부족함이 많음!

4. RDF(S): Limitation [Antonious and Harmelen, 2004]

- Class간의 AND, OR, NOT 혹은 Union, Interesection, Disjointness 개념 부재
- Class와 Property에서 Equality과 Inequality의 표현부재
  - 각기 다른 온톨로지를 병합하거나 재사용 할 때,
    - 같은 의미를 지닌 다른 이름의 클래스들과 속성들의 Equality 표현 못함
    - 다른 의미를 지닌 같은 이름의 클래스들과 속성들의 Inequality을 표현 못함
- Property에 대한 semantics표현은 소극적
  - 예, Property에 대한 cardinality 표현 부제

➔ 이와 같은 표현상의 한계 때문에 모델링 요소들을 확장하고 언어의 표현력을 강화한 OWL과 같은 온톨로지 언어가 등장

Why RDFSchema...?

- An application program can define and use RDF data
- ….if the application program knows, which terms and classes to use, as e.g.
  - name, title, year, …
  - name, blog, phone number…
  - author, cites, …
- But.....
  - Are all terms known?
  - Are all terms correct?
  - Are there (logical) relations among the terms?
- We need a language for data definition: **RDF Schema**
- officially called "**RDF Vocabulary Description Language**"

RDFSchema

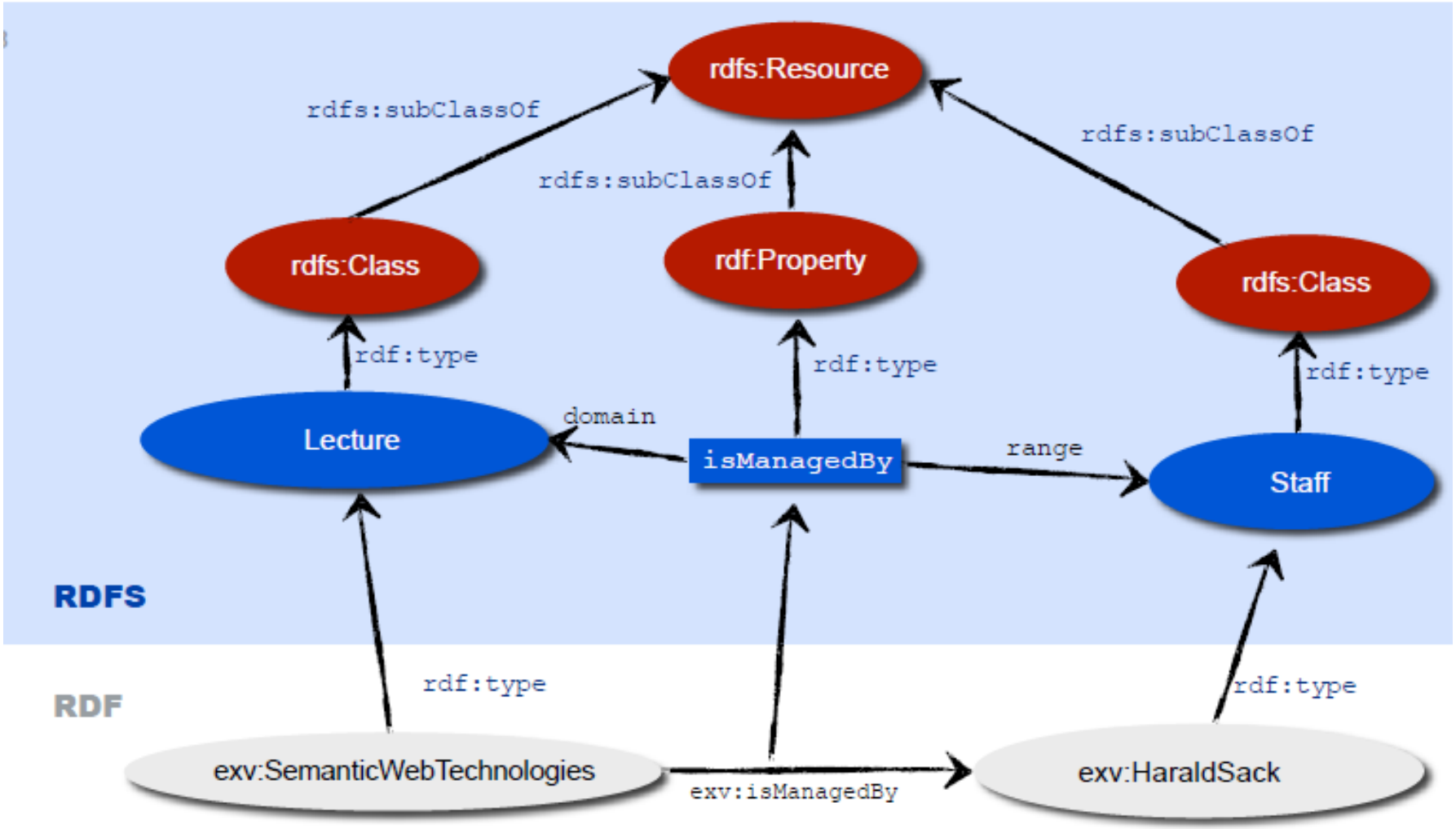
- First W3C draft in April 1998, W3C Recommendation Feb. 2004.
- RDF Schema defines a **data model** for the creation of RDF statements
- RDFSchema allows:
  - Definition of **classes**
    - **Class instantiation** in RDF via `<rdf:type>`
  - Definition of **properties** and **restrictions**
  - Definition of **hierarchies**
    - Subclasses and superclasses
    - Subproperties and superproperties

추가자료

- Classes
  - `rdfs:Class`
    Concept of a class, defines an abstract object and is applied (with rdf:type) to create instances
  - `rdf:Property`
    Base class for properties
  - `rdfs:Literal`
    Class for literals
  - `rdfs:Resource`
    every entity of an RDF model is instance of this class
  - and additionally
    `rdfs:Datatype, rdf:XMLLiteral, rdfs:Container, rdfs:ContainerMembershipProperty`

W3C RDFS Example

추가자료

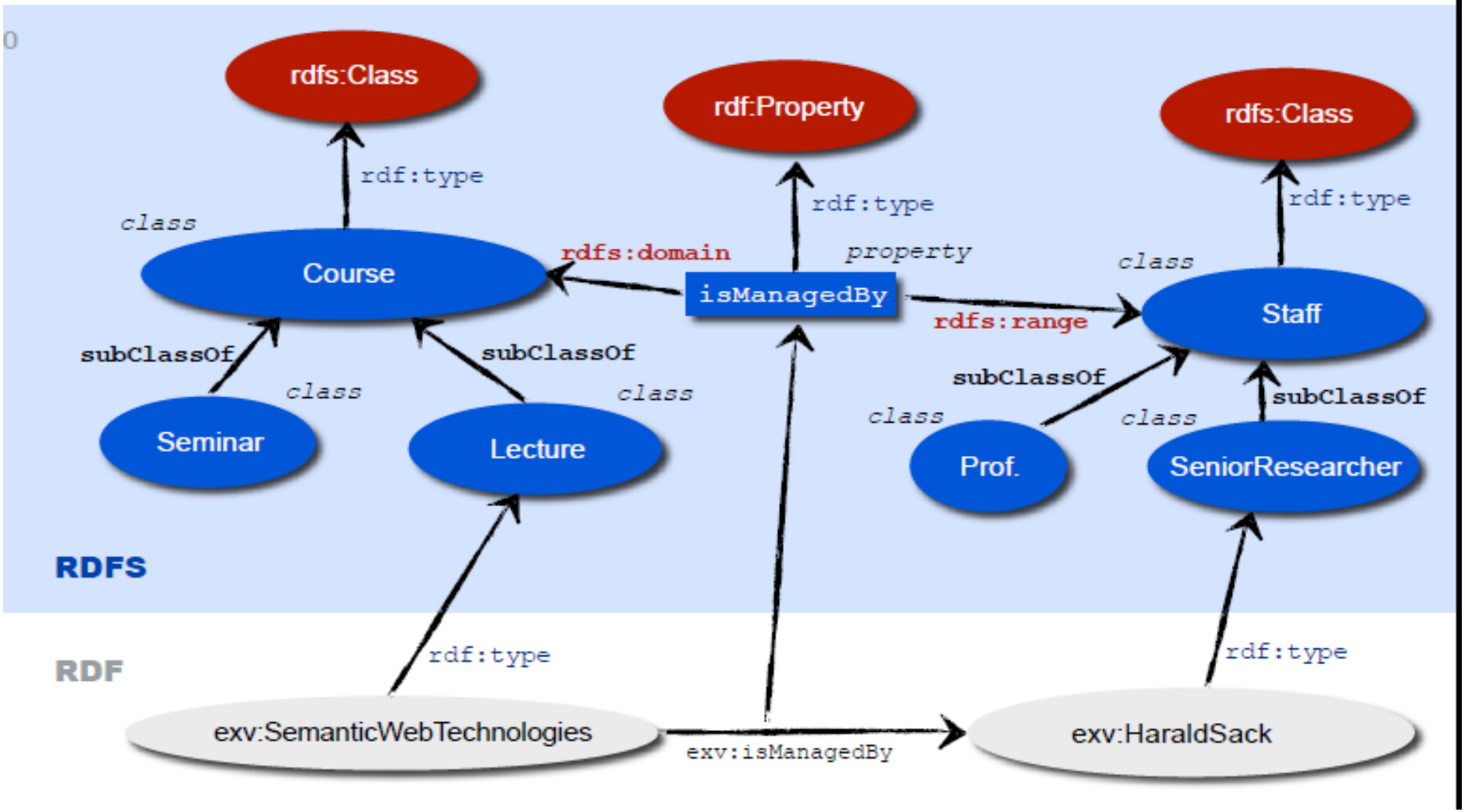


RDFS Vocabulary

- Properties
  - `rdfs:subClassOf`
    transitive property to define inheritance hierarchies for classes
  - `rdfs:subPropertyOf`
    transitive property to define inheritance hierarchies for properties
  - `rdfs:domain`
    defines the domain of a property concerning a class
  - `rdfs:range`
    defines range of a property concerning a class

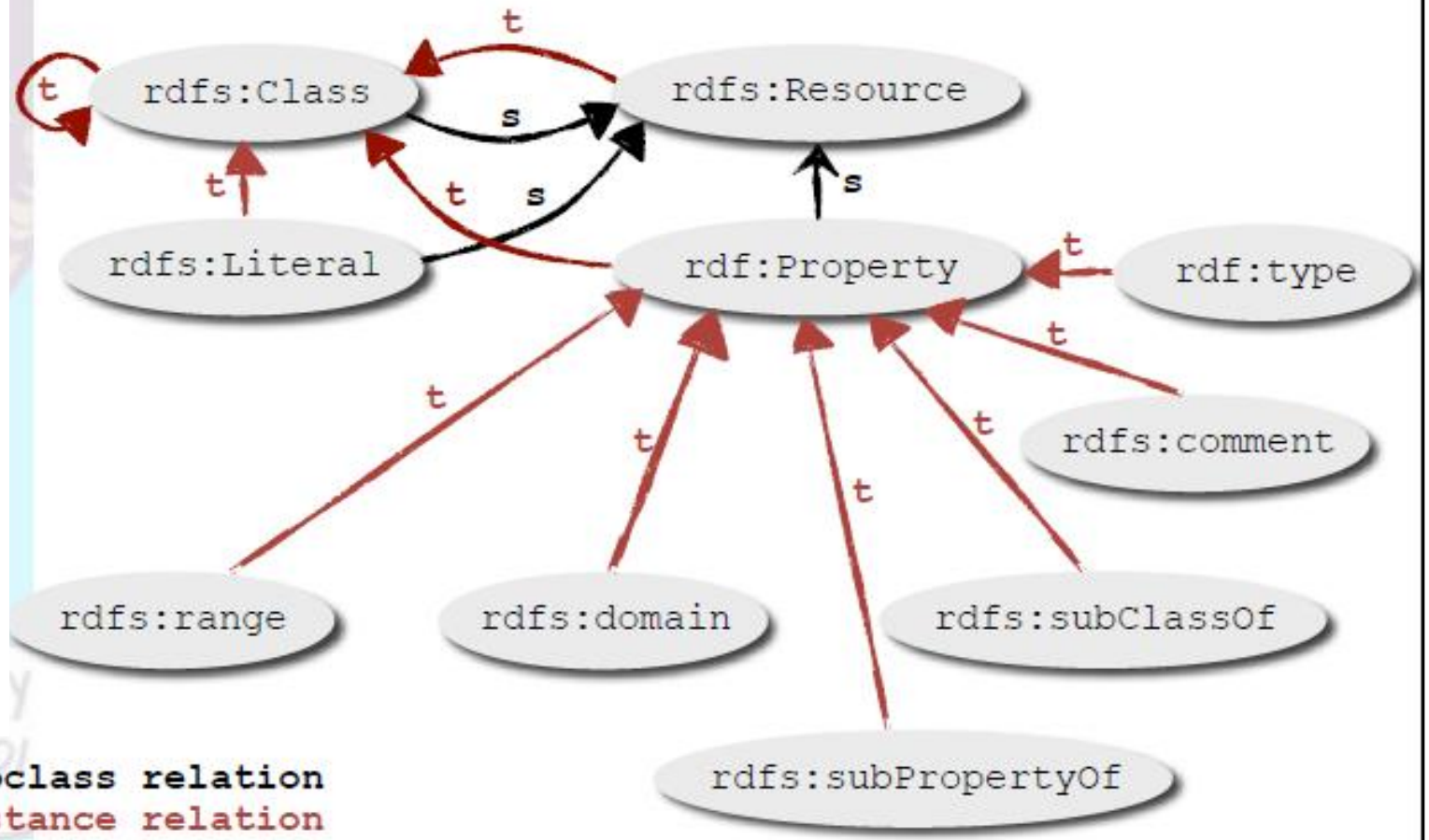
RDFS Example

추가자료



RDFS Language Model

추가자료



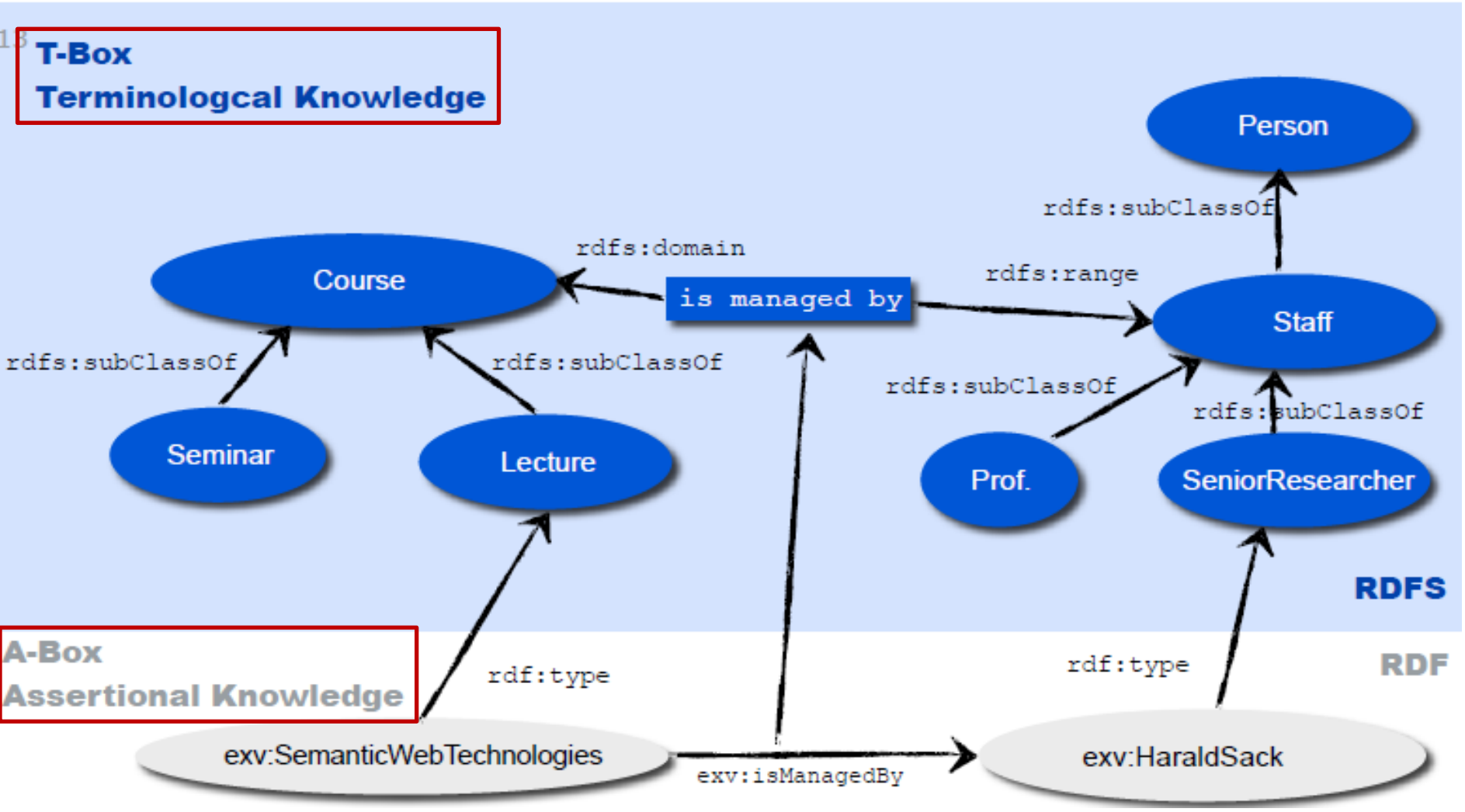
Vocabulary

추가자료

- Further Properties
  - `rdfs:seeAlso`
    defines a relation of a resource to another, which explains it
  - `rdfs:isDefinedBy`
    subproperty of `rdf:seeAlso`, defines the relation of a resource to its definition
  - `rdfs:comment`
    comment, usually as text
  - `rdfs:label`
    „readable“ name of a resource (contrary to ID)

RDF(S) Knowledge Base

추가자료



W3C RDFS Example

Turtle

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

:Course a rdfs:Class .
:Lecture a rdfs:Class;
    rdfs:subClassOf :Course.
:Seminar a rdfs:Class ;
    rdfs:subClassOf :Course.

:Person a rdfs:Class .
:Staff a rdfs:Class ;
    rdfs:subClassOf :Person .
:SeniorResearcher a rdfs:Class ;
    rdfs:subClassOf :Staff .
:Professor a rdfs:Class ;
    rdfs:subClassOf :Staff .

:isManagedBy a rdf:Property;
    rdfs:domain :Course ;
    rdfs:range :Staff .

:SemanticWebTechnologies a :Lecture .
:HaraldSack a :SeniorResearcher .
:SemanticWebTechnologies :isManagedBy :HaraldSack .
```

- **Turtle representation**

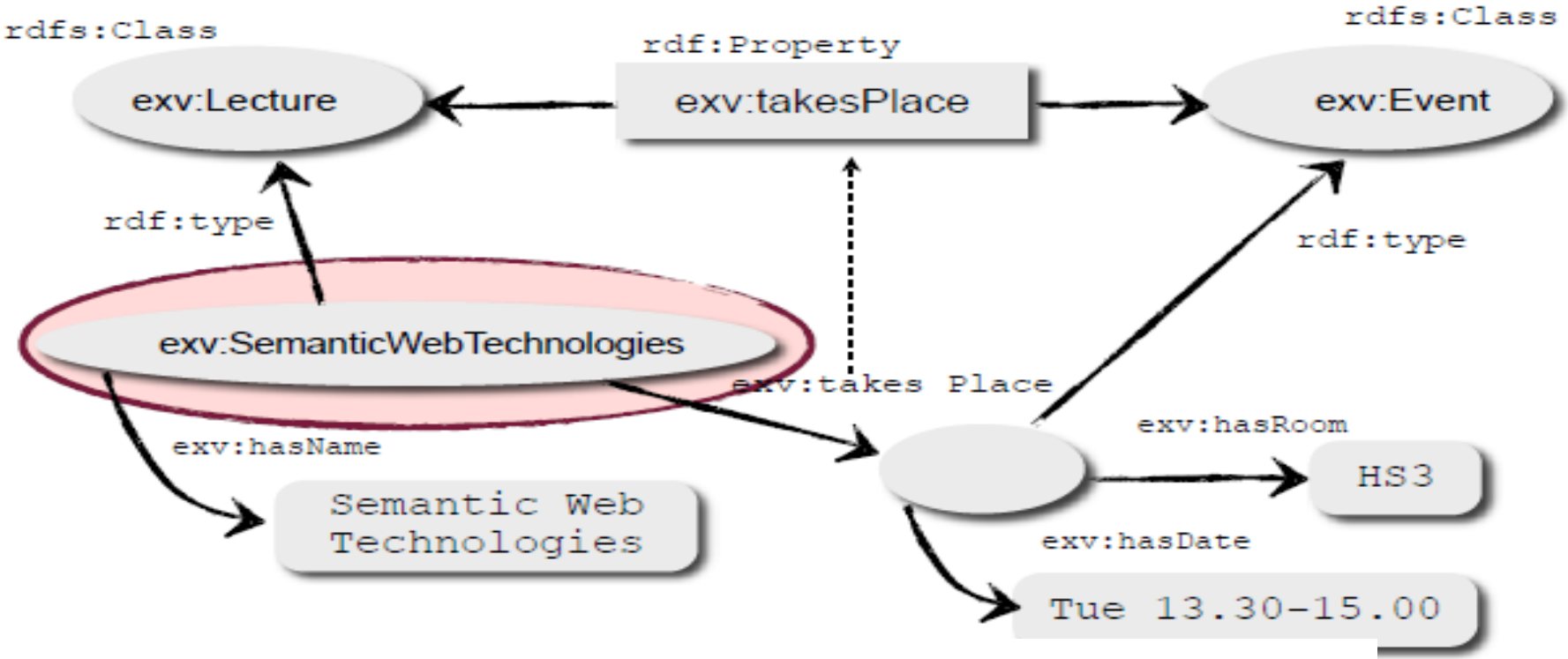
RDFS Summary

- RDFSchema defines a data model for the definition of simple ontologies (knowledge representations).
- Via RDFSchema ontologies RDF statements (facts) can be expressed

- More than XML:
  - (small) ontological agreement about modelling primitives
  - Possibility to define own vocabularies

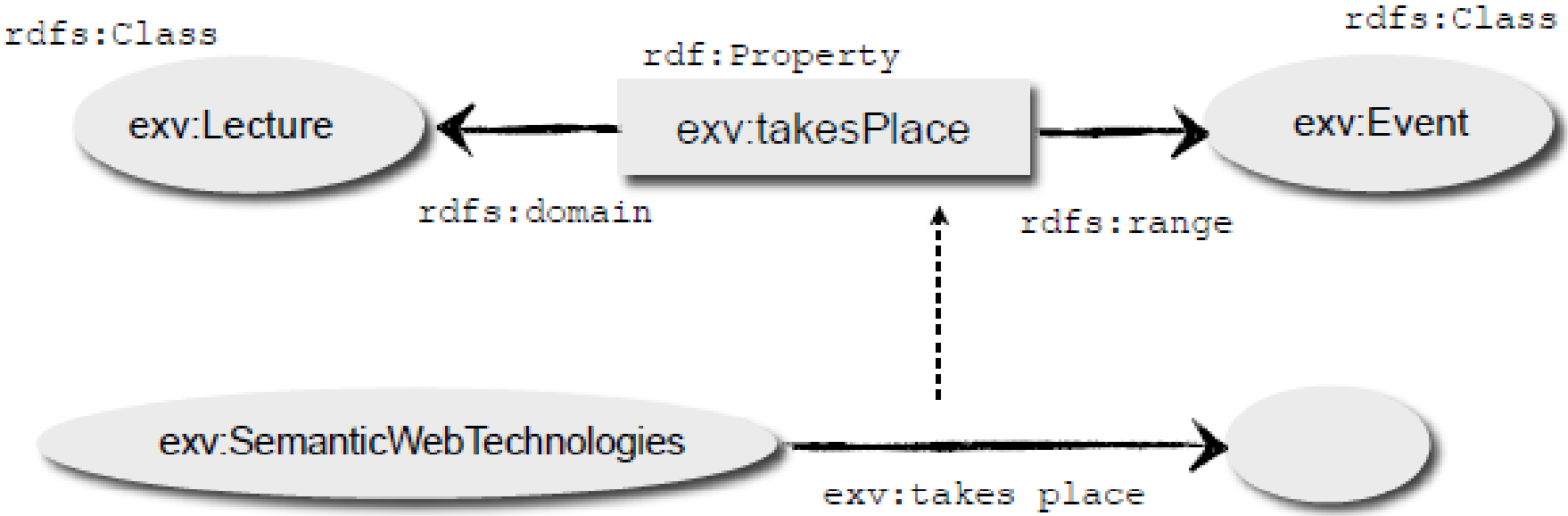
추가자료

How much knowledge (semantics) has RDF(S)



- The semantics of a term from an RDF(S) ontology is given in terms of its properties and its values (instances)

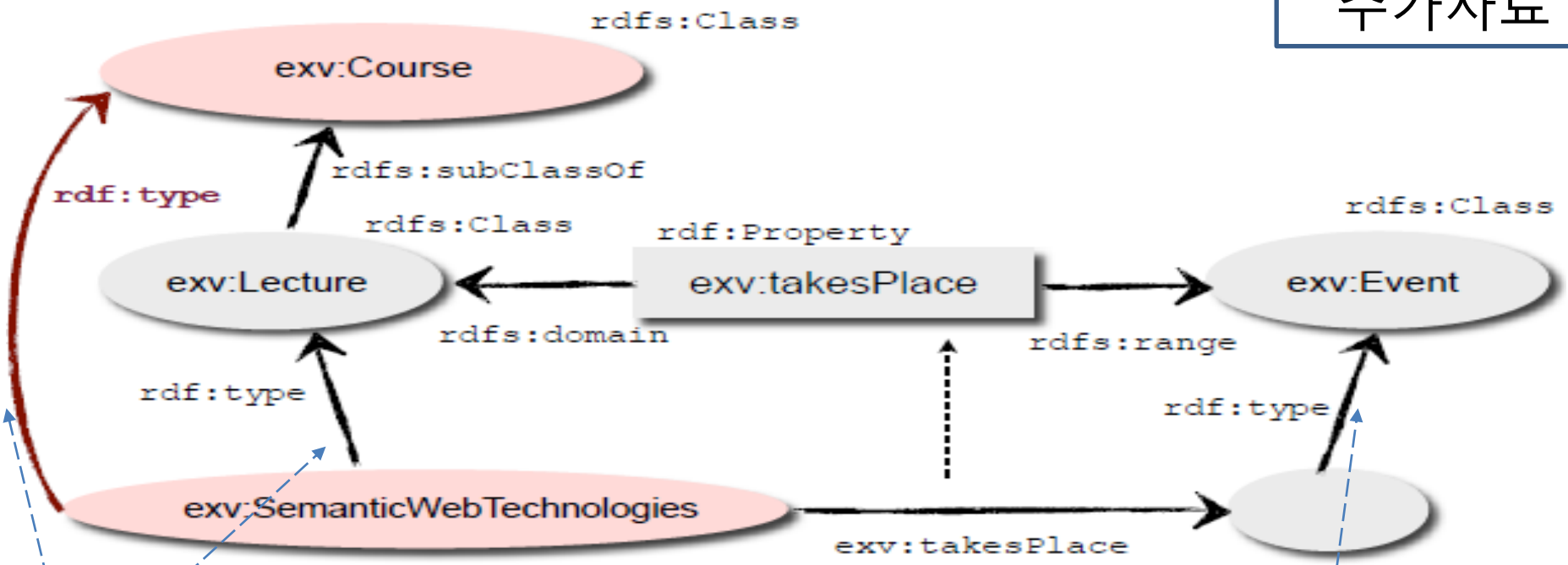
What conclusions can we deduce with RDF(S)?



추가자료

What conclusions can we deduce with RDF(S)?

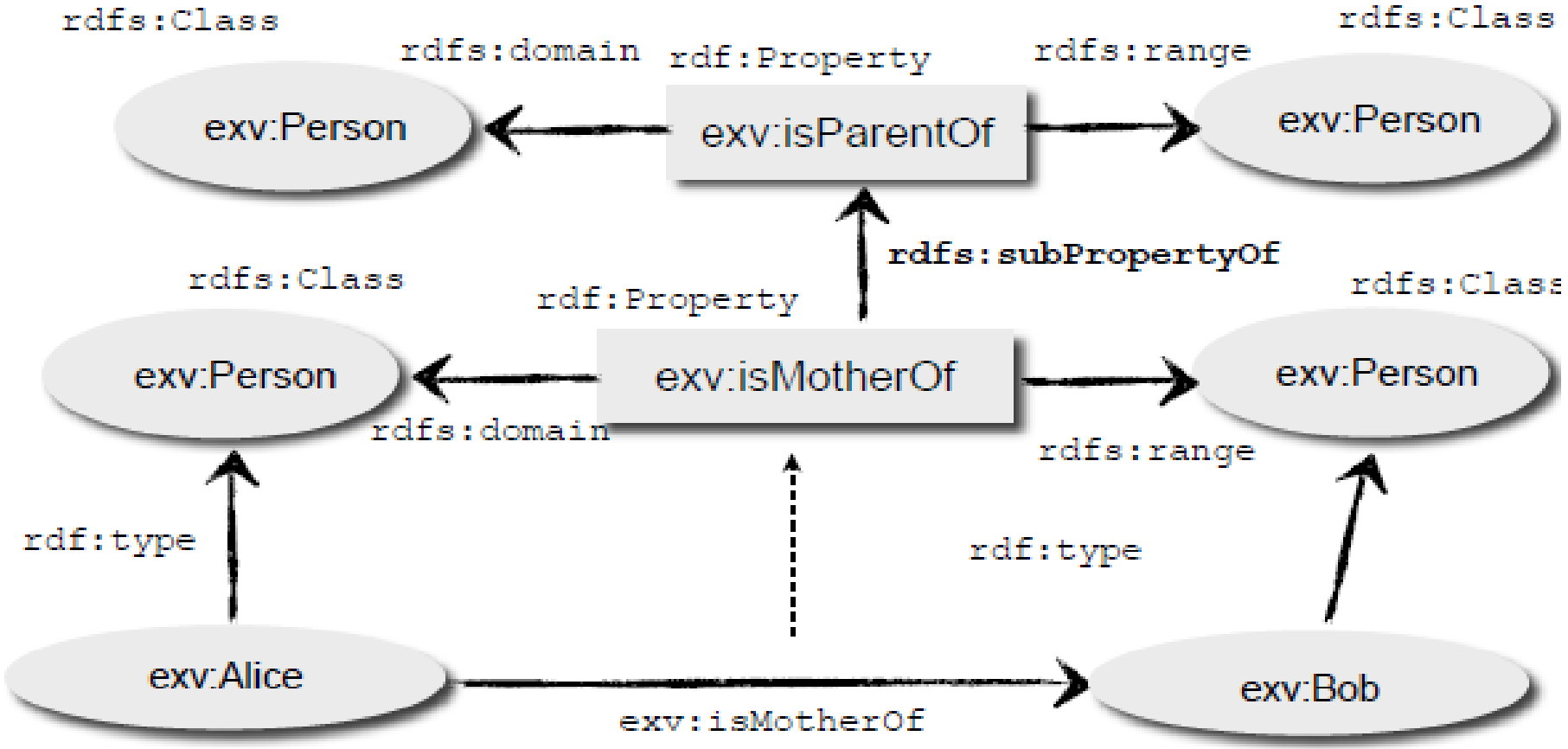
추가자료



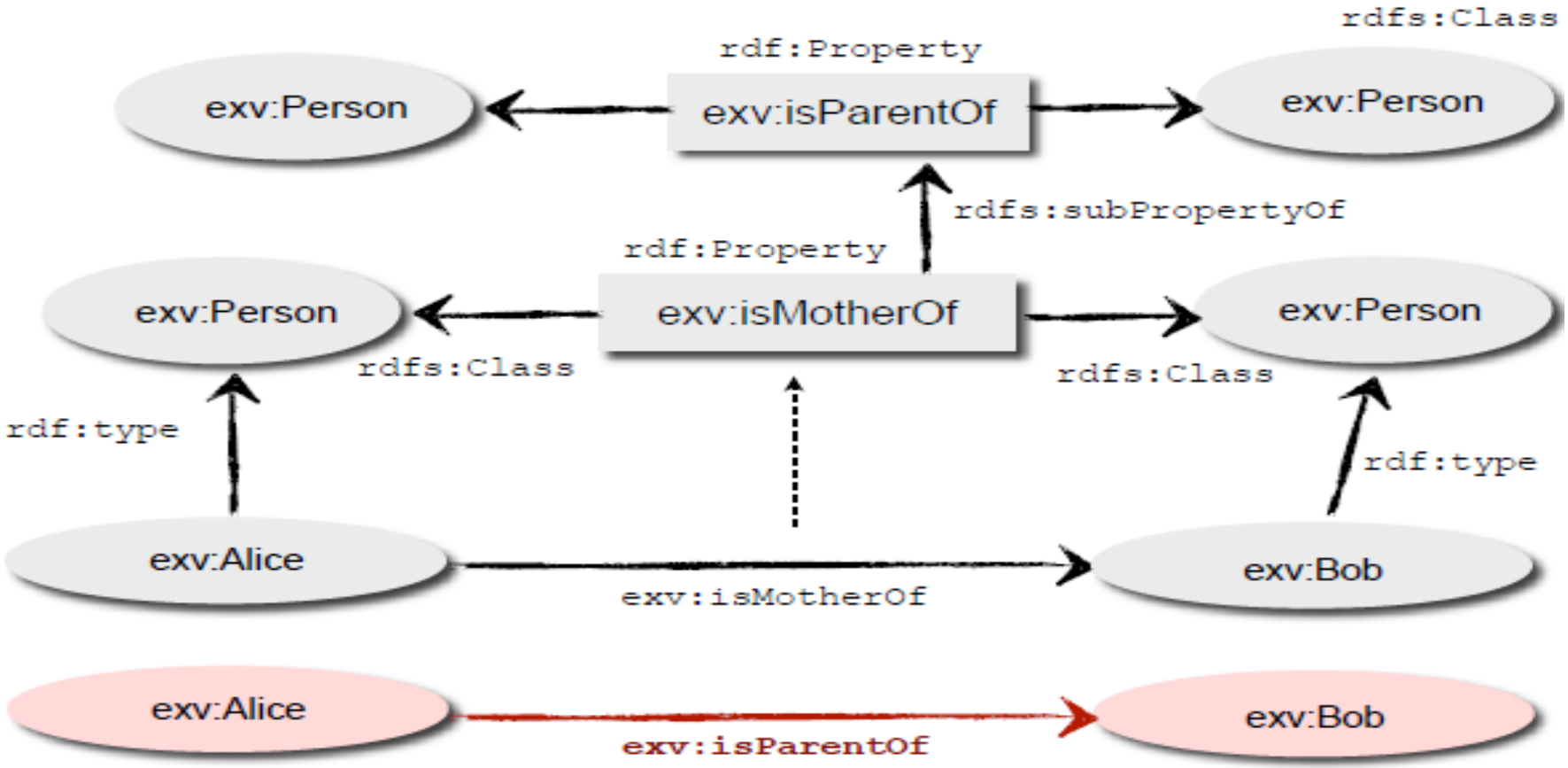
- Deduction of entity superclass membership from a class hierarchy.
- Deduction of entity class membership from the range of one of its properties

What conclusions can we deduce with RDF(S)?

추가자료



What conclusions can we deduce with RDF(S)?



- Deduction of new facts from subproperty relationships

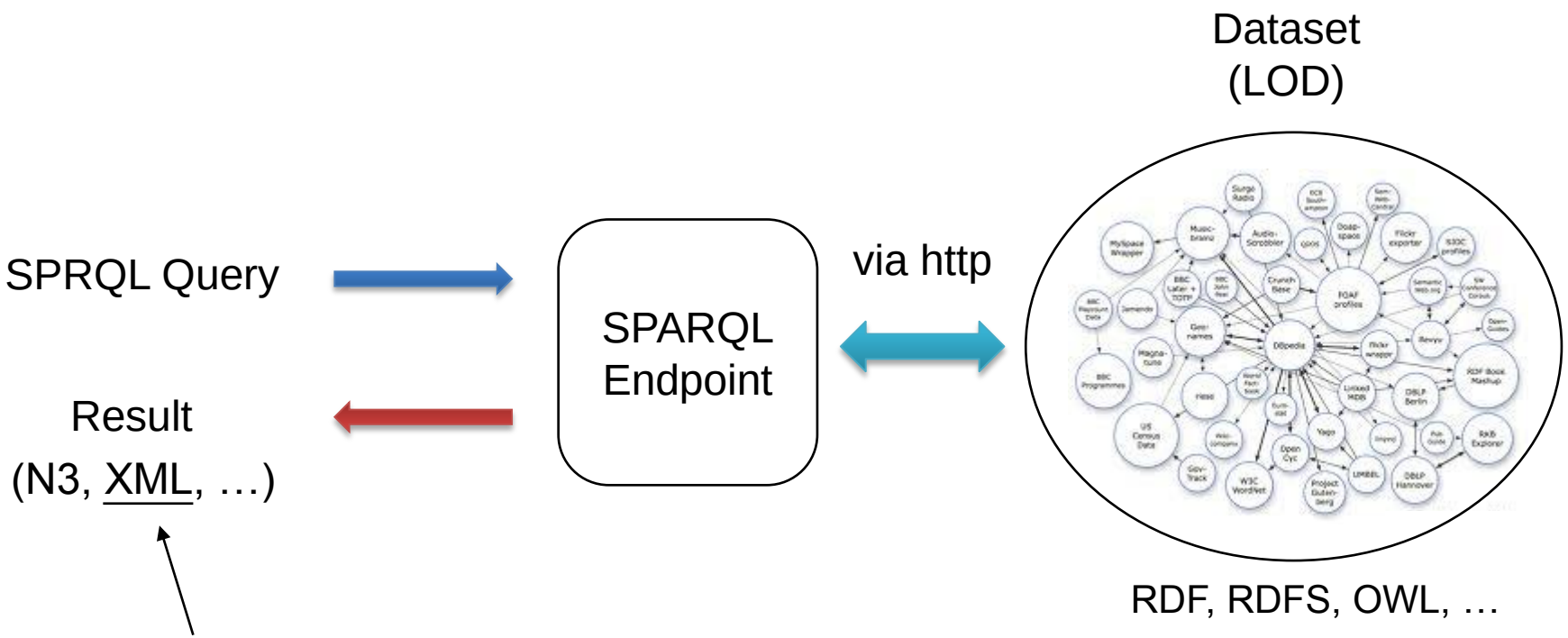
SPARQL Query Language

SQL vs. SPARQL

- SQL (Structured Query Language)
  - Relational data 관리용 프로그래밍 언어
  - Set theory
    - Relational algebra

- SPARQL (Simple Protocol And RDF Query Language)
  - RDF data 관리용 프로그래밍 언어
  - RDF graph pattern
    - Extract RDF subgraphs

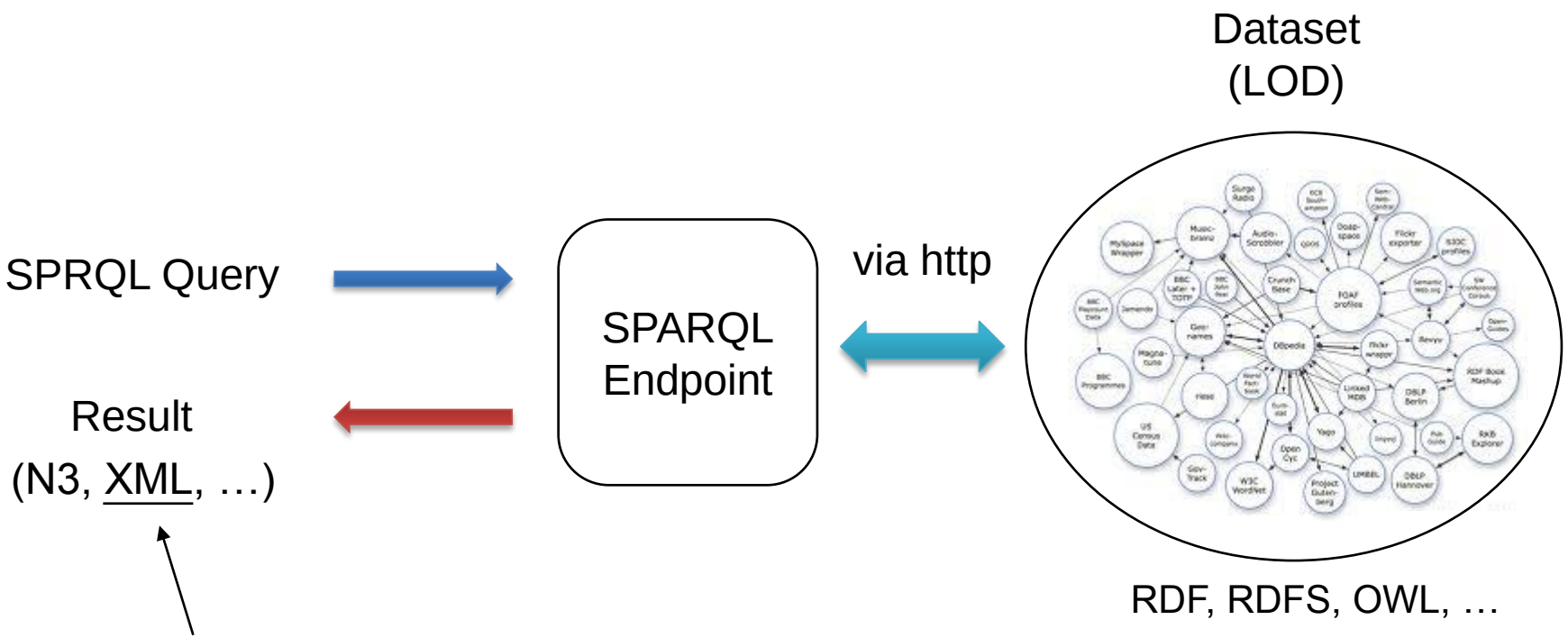
SQL vs. SPARQL



follows a specific schema
<sparql xmlns="http://www.w3.org/2005/sparql-results#">

(similar to RDF/XML:
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" ..>)

SPARQL: Simple Protocol And RDF Query Language



follows a specific schema
<sparql xmlns="http://www.w3.org/2005/sparql-results#">

(similar to RDF/XML:
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" ..>)

SPARQL Query Language

- How do we access the knowledge from RDF/RDFS Knowledge Bases?
  - Manual Parsing/Reading and Combination of RDF-Triple is complex
- Can you imagine a relational database without SQL?

- **SPRQL** (**S**imple **P**rotocol **A**nd **R**DF **Q**uery **L**anguage)
  - a **Query Language** for RDF Graph Traversal
    (*SPARQL Query Language Specification*)
  - a **Protocol Layer**, to use SPARQL via http
    (*SPARQL Protocol for RDF Specification*)
  - an **XML Output Format Specification** for SPARQL Queries
    (*SPARQL Query XML Results Format*)
  - W3C Standard (SPARQL 1.0 since Januar 2008)
  - inspired by SQL

http://www.w3.org/TR/2013/REC-rdf-sparql-XMLres-20130321/

- **SPARQL 1.0** allows
  - **Extraction** of data as
    - URIs, Blank Nodes, typed and untyped Literals.
    - RDF Subgraphs
  - **Exploration** of Data via Query for unknown relations
  - Execution of **complex Join Operations** on heterogeneous databases in a single query
  - **Transformation** of RDF Data from one vocabulary into another
  - **Construction** of new RDF Graphs based on RDF Query Graphs

- **SPARQL 1.1** (in progress) allows
  - **additional query features**
    - aggregate functions, subqueries, negations, project expressions, property paths
  - enables logical **Entailment** for
    - RDF, RDFS, OWL Direct and RDF-Based Semantics entailment, and RIF Core entailment.
  - enables **Update of RDF Graphs** as a full data manipulation language
  - enables the **Discovery of information** about the SPARQL service
  - enables **Federated Queries** distributed over different SPARQL endpoints

SPARQL Example

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>

SELECT  ?country_name  ?population
WHERE {
        ?country  a                       type:LandlockedCountries ;
                  rdfs:label              ?country_name            ;
                  prop:populationEstimate ?population              .
        FILTER (  ?population > 15000000  ) .
}
LIMIT 50
```

SPARQL Structure

SPRQL Language Format

1. PREFIX

xsd, dc, foaf, 등 사용할 URI의 shortcut

2. SELECT

보고 싶은 결과 항목 설정 (이름, 나이, …)

3. FROM

검색할 대상 Data (<http://ex.com/student.rdf>)

4. WHERE

Filter the results and apply conditions
(남학생 중 23세 이상, …)

5. GROUP BY/HAVING/
ORDER BY / LIMIT / OFFSET ..

Query modifiers
(결과 중 상위 10개만 내림차순으로, …)

- SPARQL **Variables** are bound to RDF terms
  - e.g. **?journal, ?disease, ?price**
- In the same way as in SQL a **Query** for variables is performed via SELECT statement
  - e.g. **SELECT ?title ?author ?published**
- A SELECT statement returns **Query Results** as a table

?title	?author	?published
1984	George Orwell	1948
Brave New World	Aldous Huxley	1932
Fahrenheid 451	Ray Bradbury	1953

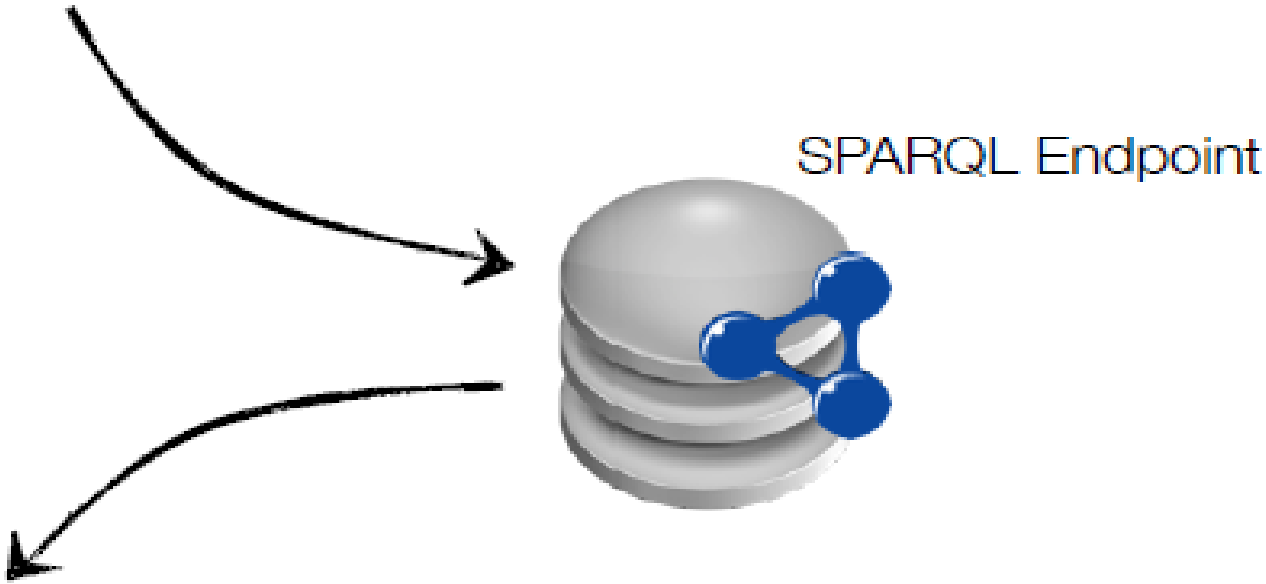
SPARQL Endpoint

- A SPARQL endpoint enables users (human or other) to query a knowledge base via the SPARQL language.
  - Results are typically returned in one or more machine-processable formats.
  - A conformant SPARQL protocol service as defined in the SPROT specification.

- Endpoint (= Port)
  - Endpoint is an association between a fully-specified InterfaceBinding and a network address, specified by a URI for communicating with an instance of a Web Service.
  - An EndPoint indicates a specific location for accessing a Web Service using a specific protocol and data format.

SPARQL Query

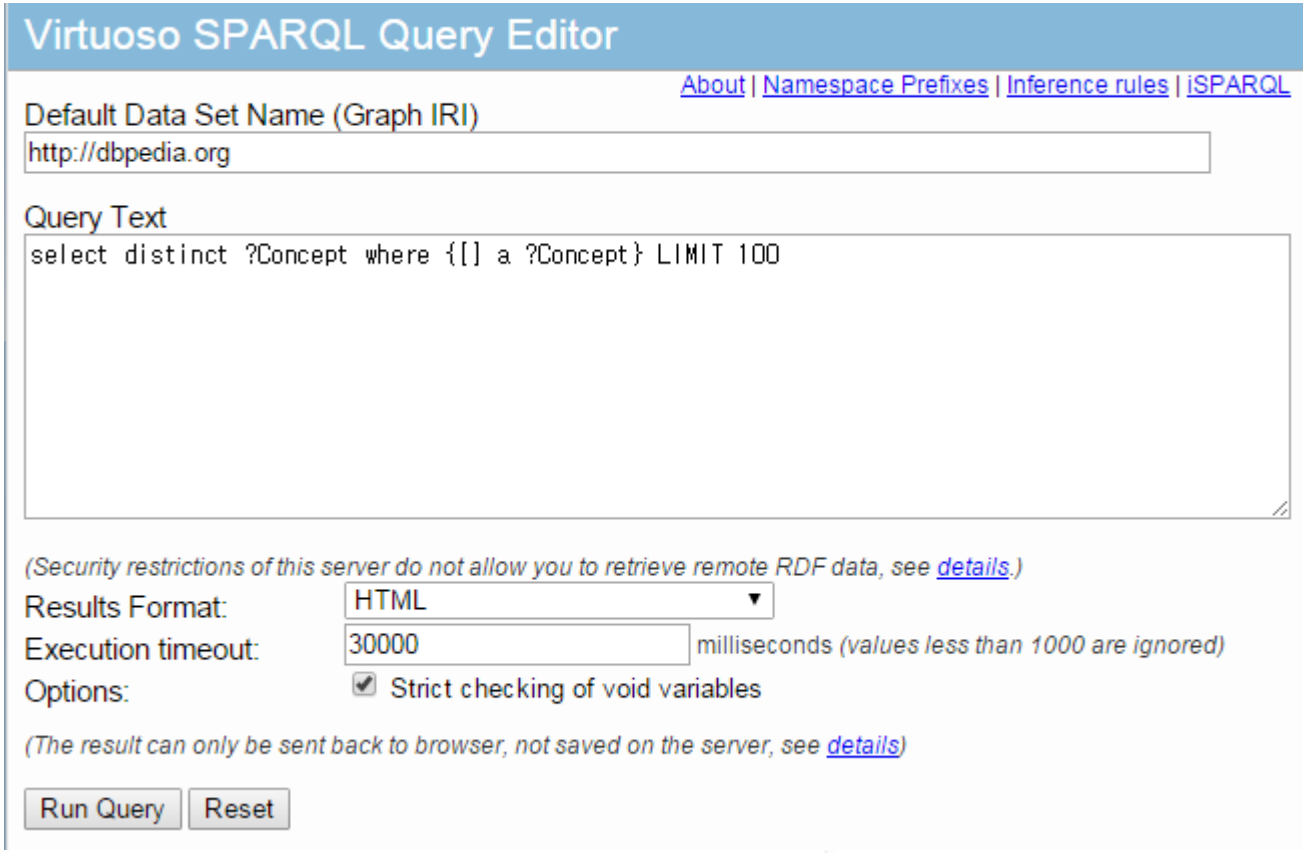
```
SELECT ?title ?author ?published
```



?title	?author	?published
1984	George Orwell	1948
Brave New World	Aldous Huxley	1932
Fahrenheid 451	Ray Bradbury	1953

SPARQL Endpoint: DBpedia Endpoint

- **http://dbpedia.org/sparql**
- **http://ko.dbpedia.org/sparql**



SPARQL Endpoint: 학술연구정보서비스 (RISS)

- **http://data.riss.kr/sparqlEndpoint.do**

RISS Linked Open Data 발행서비스

소개 및 구축방법 | MARC to RDF매칭 규칙 | 온톨로지모델 | 데이터네비게이션 | SPARQL Endpoint

Query

--TEMPLATE--

```
SELECT ?uri ?제목 WHERE {
?s a <http://xmlns.com/foaf/0.1/Agent>.
?s <http://www.w3.org/2004/02/skos/core#prefLabel> "이청준"^^xsd:string.
?s <http://data.riss.kr/ontology/createBook> ?uri.
?uri <http://data.riss.kr/ontology/heldByUniv> ?k.
?k <http://www.w3.org/2004/02/skos/core#prefLabel> "경희대학교"^^xsd:string.
?uri <http://purl.org/dc/elements/1.1/title> ?제목.
}
```

HTML | Result | Download | DBpedia

'DBpedia'를 직접 Query 하실 경우 DBpedia 데이터의 상태에 따라 시간이 지연될 수 있습니다.

Query 결과

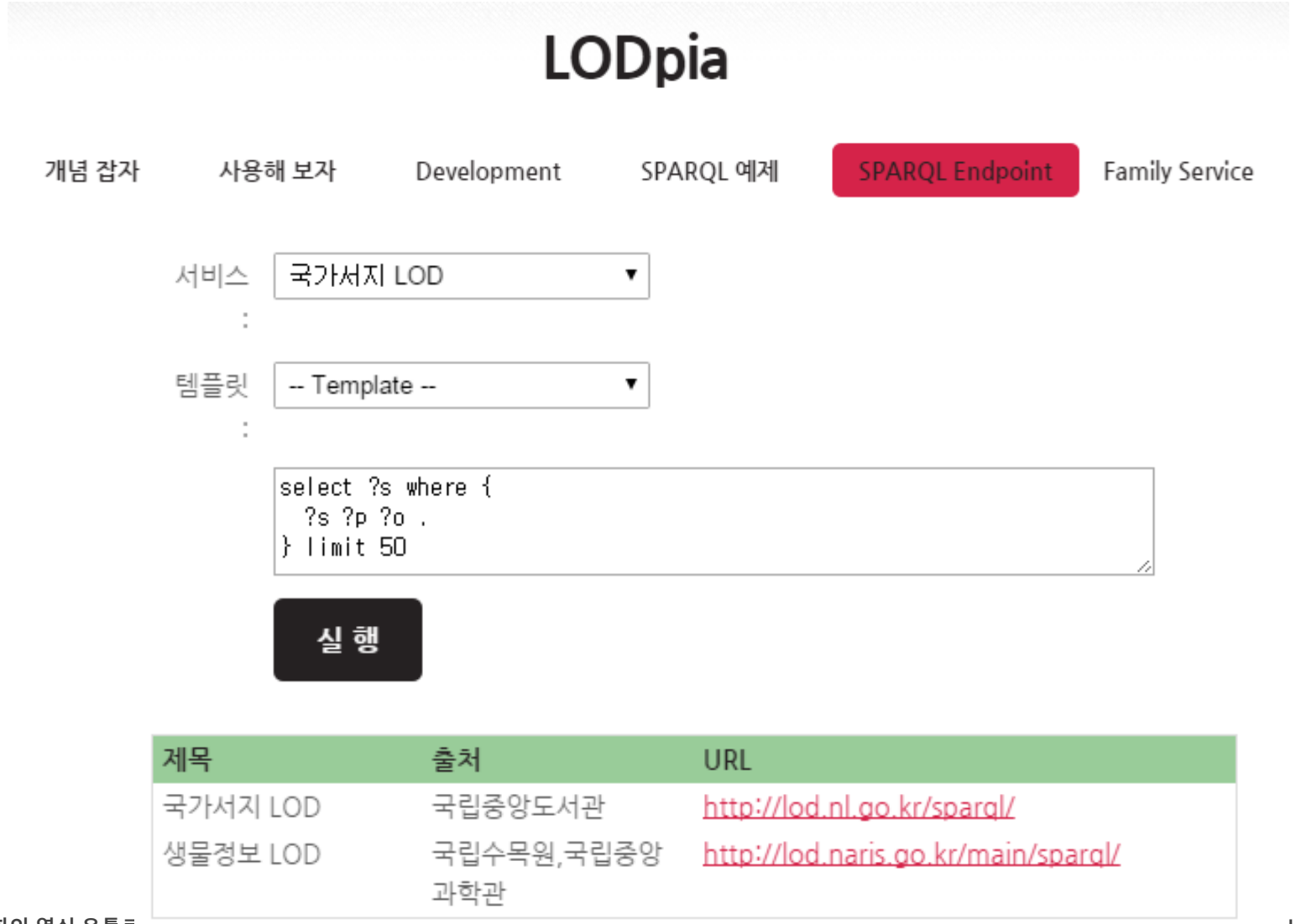
URI	제목
http://data.riss.kr/resource/Book/000000034773	조율사
http://data.riss.kr/resource/Book/000000925710	씌어지지 않은 자서전:이청준 장편소설
http://data.riss.kr/resource/Book/000007209829	제3의 현장
http://data.riss.kr/resource/Book/000007720284	당신들의 천국

SPARQL Endpoint: 서울 LOD 서비스

- **http://lod.seoul.go.kr/linkdata_beta/endpoint/ep_contents.jsp?vParm=0**
- **화면에서 결과 출력 + 결과 XML File로 제공**

SPARQL Endpoint: LODpia

- **http://lodpia.com/sparqlendpoint.jsp?menu=5**



Graph Pattern

- SPARQL is based on **RDF Turtle serialization** and **basic graph pattern matching**.
- A Graph Pattern (Triple Pattern) is a RDF Triple that contains variables at any arbitrary place (Subject, Predicate, Object).
- **(Graph) Triple Pattern = Turtle + Variables**
- **Example:**
  - *Look for countries and their capitals:*

```
?country geo:capital ?capital .
```

Triple Pattern

```
?country geo:capital ?capital .
```

RDF Graph

```
dbpedia:Venezuela rdf:type dbpedia-owl:Country .
dbpedia:Venezuela geo:capital "Caracas" .
dbpedia:Venezuela dbprop:language "Spanish" .
dbpedia:Germany rdf:type dbpedia-owl:Country .
dbpedia:Germany geo:capital "Berlin" .
dbpedia:Germany dbprop:language "German" .
...
```

Graph Pattern

- **More Examples:**
  - *Given a FOAF URI, find the name of a person:*

```
<http://hpi-web.de/id#haraldsack> foaf:name ?surname .
```

  - *Which persons have the family name „Schmidt“?*

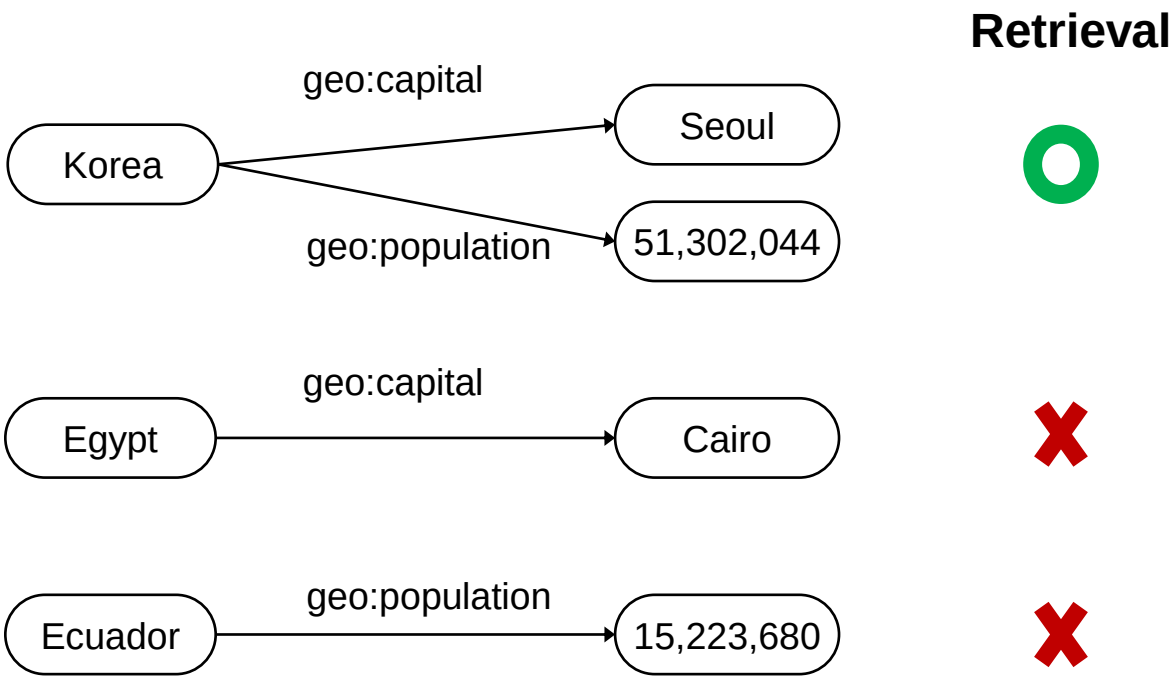
```
?person pers:familyName “Schmidt“ .
```

Complex Query Pattern

- Graph Pattern can be combined to form **complex (conjunctive) queries** for RDF graph traversal.

- *Find countries, their capitals and their population:*

```
?country geo:capital ?capital .
?country geo:population ?population .
```



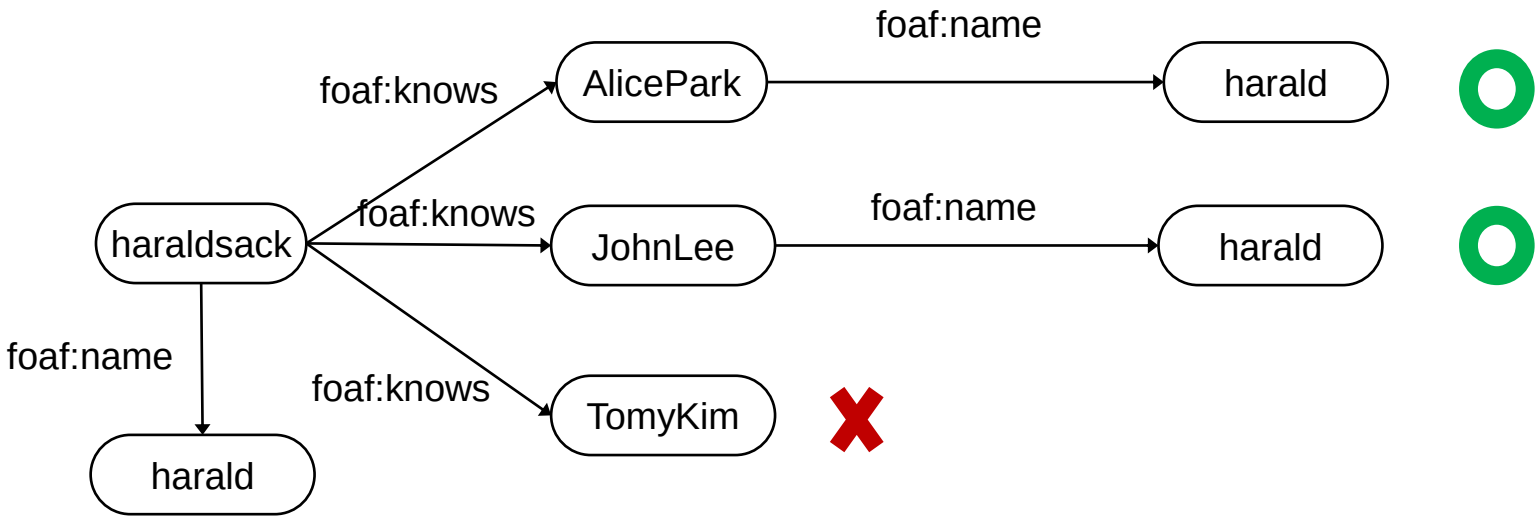
Complex Query Pattern

- Graph Pattern can be combined to form **complex (conjunctive) queries** for RDF graph traversal.

- *Given a FOAF URI, find the name of a person and her friends:*

```
<http://hpi-web.de/id#haraldsack> foaf:name ?surname ;
                                  foaf:knows ?friend .
?friend foaf:name ?friend_surname .
```

Retrieval

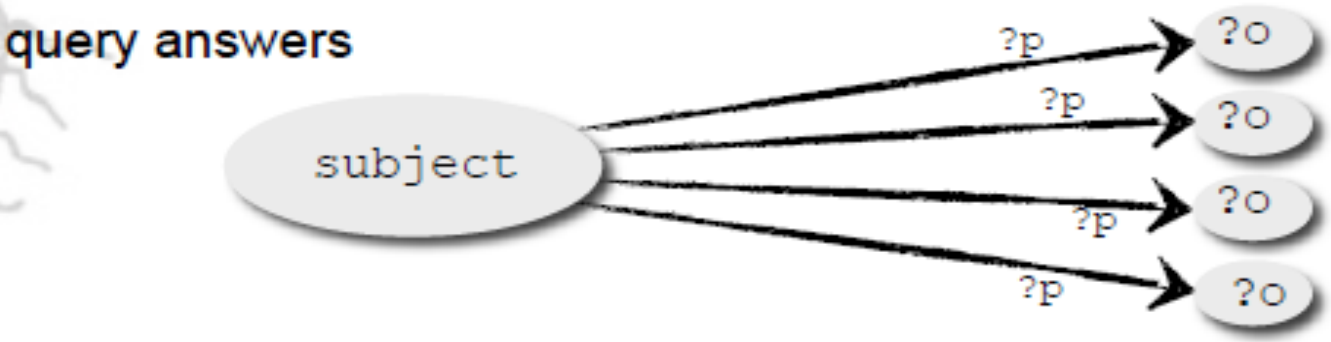


SPARQL Query Format

- inspired by SQL

```
SELECT ?p ?o
WHERE { <subject> ?p ?o. }
```

- Triple in „WHERE“ part defines graph query with variables ?p and ?o
- Query returns table with matching ?p, ?o pairs



SPARQL Query Format

- *Search all lectures and their managers:*

```
PREFIX hpi: <http://hpi-web.de/swt1213#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?lecture ?manager
FROM <...>
WHERE {
        ?x rdf:type hpi:Lecture .
        ?x rdfs:label ?lecture .
        ?y rdf:type hpi:Staff .
        ?y rdfs:label ?manager
        ?x hpi:isManagedBy ?y .
}
```

- `WHERE` - specifies graph pattern to be matched
- `PREFIX` - specifies one or more namespaces for using compact URIs (CURIEs)
- `FROM` - specifies one or more RDF source graphs
- `BASE` - defines a base URI

SPARQL Query Format

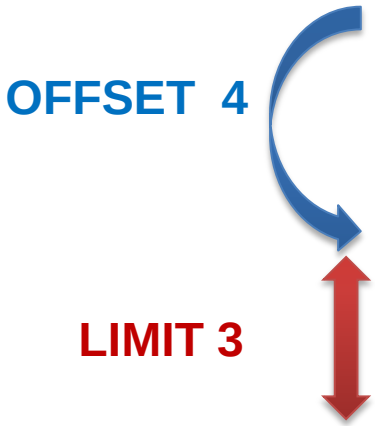
- Search all lectures and their managers **ordered by** managers in descending order and **limit** the results to the first 10 **starting** the list at position 10:

```
PREFIX hpi: <http://hpi-web.de/swt1213#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?lecture ?manager
FROM <...>
WHERE {
       ?x rdf:type hpi:Lecture ;
            rdfs:label ?lecture .
       ?y rdf:type hpi:Staff ;
            rdfs:label ?manager
       ?x hpi:isManagedBy ?y .
}
ORDER BY DESC (?manager)
LIMIT 10
OFFSET 10
```

ORDER BY, LIMIT, OFFSET

- ORDER BY ASC: 오름차순
- ORDER BY DESC: 내림차순
- LIMIT (NUMBER) 결과 중 명시한 개수만 출력
- OFFSET (NUMBER) 결과 중 명시한 위치부터 출력

```
SELECT …
FROM …
…

ORDER BY ASC
OFFSET  4
LIMIT     3
```



AAA		
BBB		
CCC		
DDD		
EEE		
FFF		
GGG		
HHH		

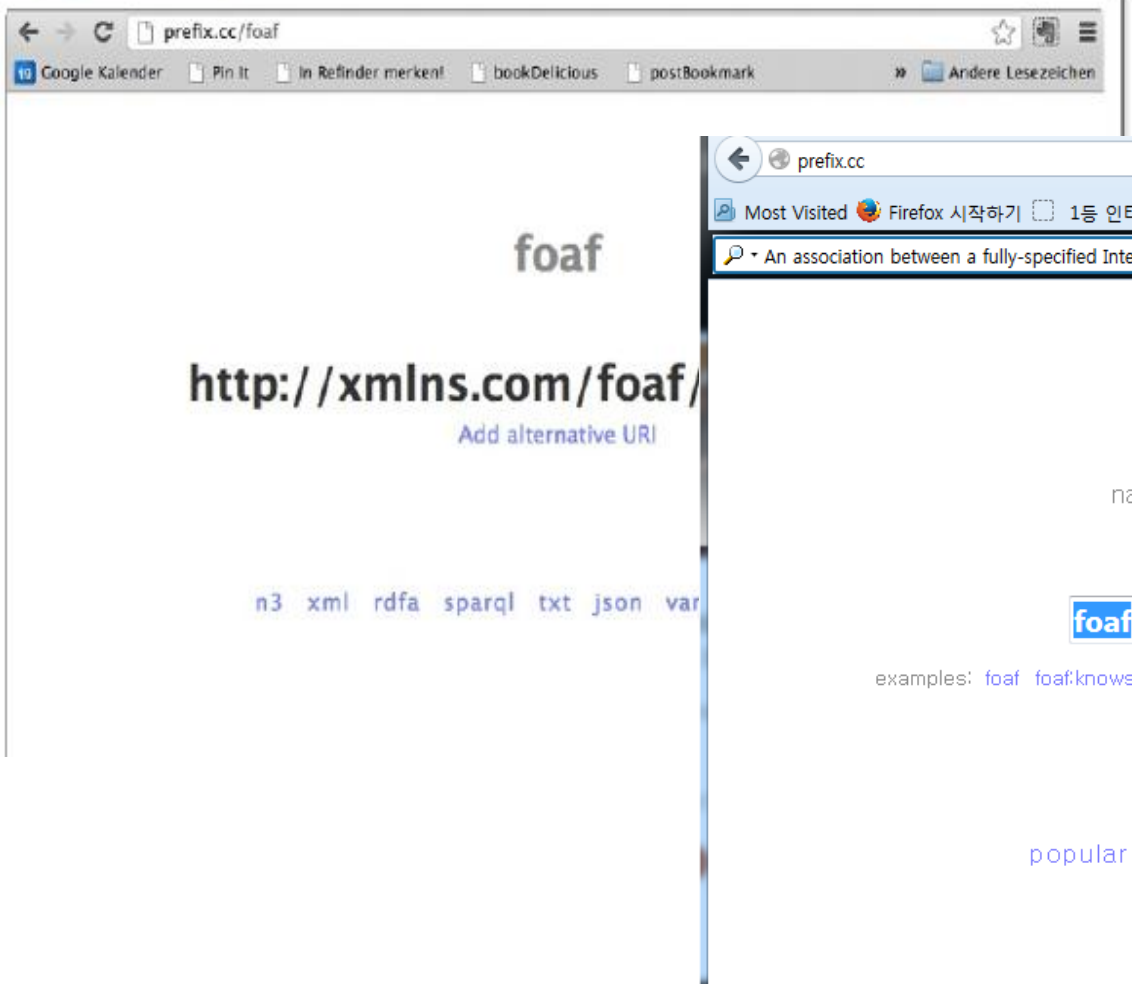
Blank Nodes in SPARQL Queries

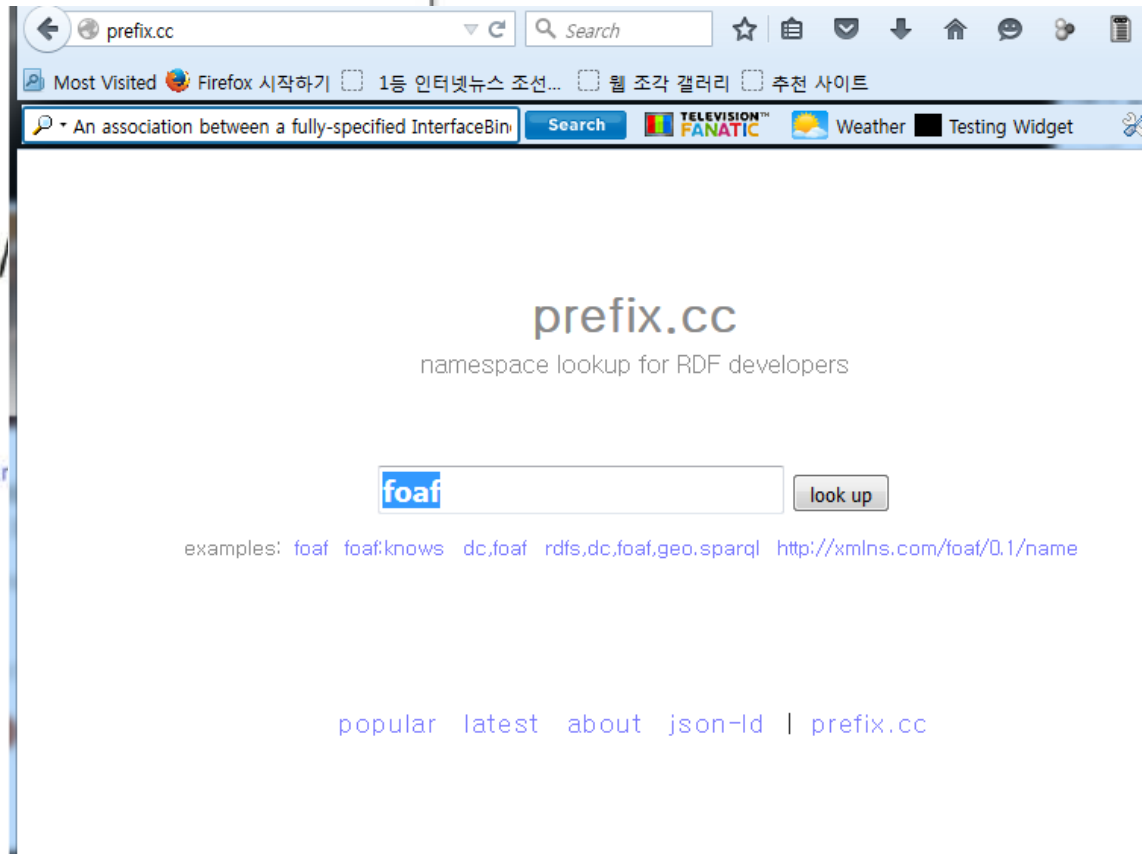
- as subject or object of a triple pattern
- „non selectable“ variables

```
PREFIX hpi: <http://hpi-web.de/swt1213#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?name
WHERE {
      _:x rdf:type hpi:Lecture ;
          hpi:isManagedBy [rdfs:label ?name] .
}
```

- blank node identifier might also occur in the results

How should I know all those prefixes?





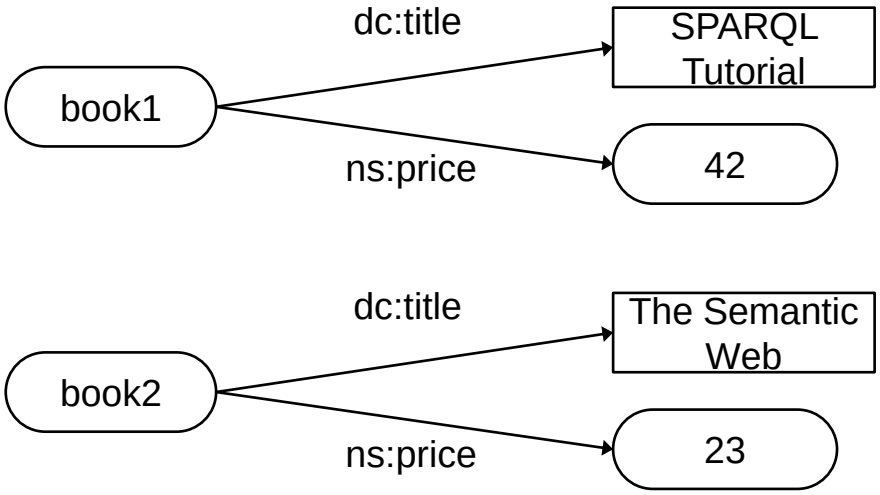
- **prefix.cc**

SPARQL Query Format

- The keyword **FILTER** specifies constraints for the results

```
# Default Graph (stored at http://example.org/book)
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

Anticipated result



SPARQL Query Format

- The keyword **FILTER** specifies constraints for the results

```
# Default Graph (stored at http://example.org/book)
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

- FILTER expressions contain operators and functions
- FILTER can **NOT** assign/create new values

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
FROM     <http://example.org/book>
WHERE  {
         ?x ns:price ?price .
                  FILTER (?price < 30.5)
         ?x dc:title ?title .
}
```

- http://example.org/book.rdf

SPARQL Unary Operators in Constraints

Operator	Type(A)	Result Type
`!A`	xsd:boolean	xsd:boolean
`+A`	numeric	numeric
`-a`	numeric	numeric
`BOUND(A)`	variable	xsd:boolean
`isURI(A)`	RDF term	xsd:boolean
`isBLANK(A)`	RDF term	xsd:boolean
`isLITERAL(A)`	RDF term	xsd:boolean
`STR(A)`	literal/URI	simple literal
`LANG(A)`	literal	simple literal
`DATATYPE(A)`	literal	URI

More SPARQL Operators

- Logical connectives **&&** and **||** for xsd:boolean
- Comparison operators **=**, **!=**, **<**, **>**, **<=**, and **>=** for numeric datatypes, xsd:dateTime, xsd:string, and xsd:boolean
- Comparison operators **=** and **!=** for other datatypes
- Arithmetic operators **+**, **-**, *****, and **/** for numeric datatypes

- and in addition:
  - **REGEX**(String,Pattern) or **REGEX**(String,Pattern,Flags)
  - **sameTERM**(A,B)
  - **langMATCHES**(A,B)

- **regular expression, compare terms,**

Filter Constraints are Evaluated in 3-valued Logic

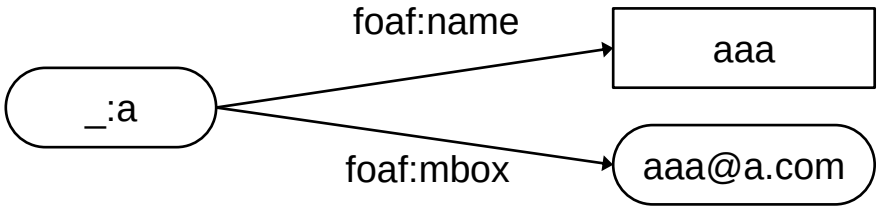
- true, false, and **error**

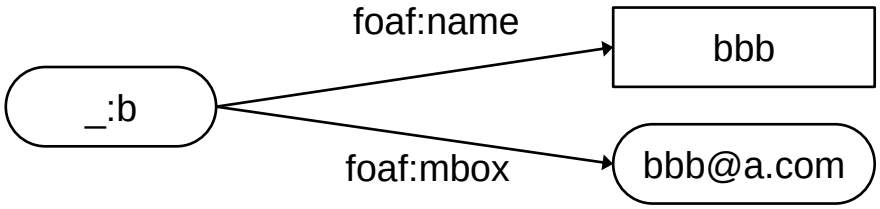
A	B	A || B	A && B
T	T	T	T
T	F	T	F
F	T	T	F
F	F	F	F
T	E	T	E
E	T	T	E
F	E	E	F
E	F	E	F
E	E	E	E

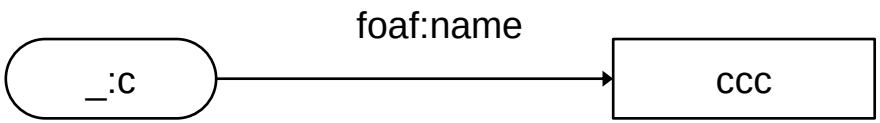
A	!A
T	F
F	T
E	E

SPARQL Query Format

- The keyword **OPTIONAL** selects optional elements from the RDF graph
- complies to a Left Outer Join





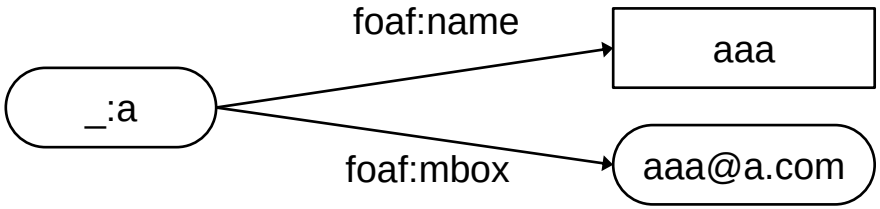


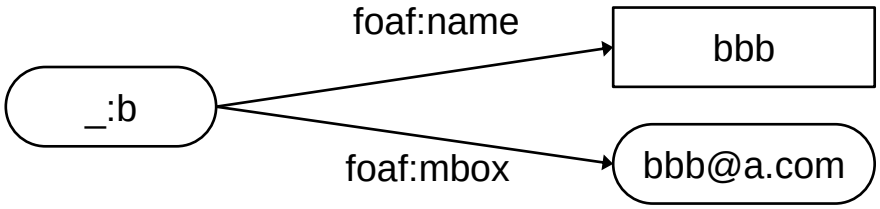
```
SELECT ?name
WHERE {
   ?x foaf:name ?name .
}
```

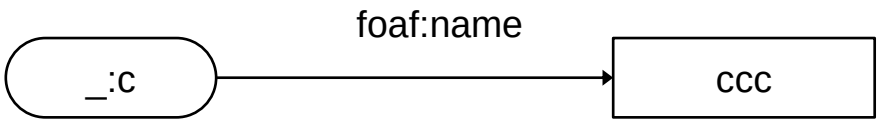
?name
aaa
bbb
ccc

SPARQL Query Format

- The keyword **OPTIONAL** selects optional elements from the RDF graph
- complies to a Left Outer Join





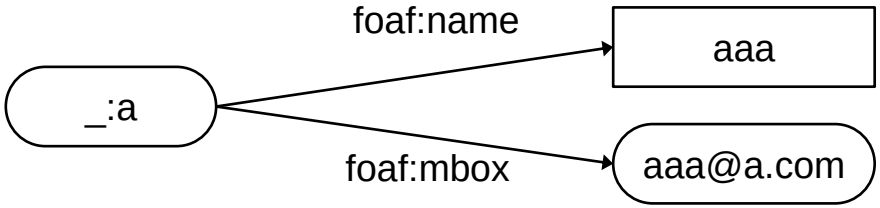


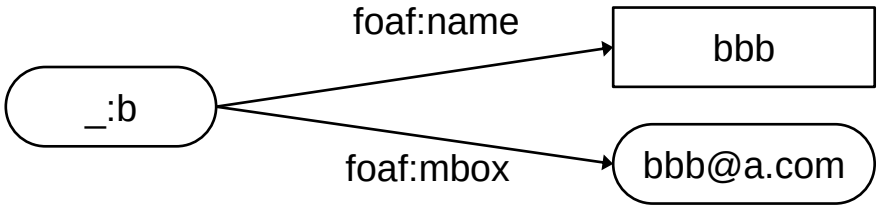

```
SELECT ?name  ?mail
WHERE {
   ?x foaf:name ?name .
   ?x foaf:mbox ?mail .
}
```

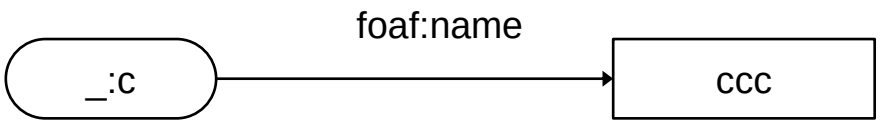
?name	?mail
aaa	aaa@a.com
bbb	bbb@b.com

SPARQL Query Format

- The keyword **OPTIONAL** selects optional elements from the RDF graph
- complies to a Left Outer Join








```
SELECT ?name  ?mail
WHERE {
  ?x foaf:name ?name .
  OPTIONAL {
     ?x foaf:mbox ?mail .
  }
}
```

?name	?mail
aaa	aaa@a.com
bbb	bbb@b.com
ccc	

SPARQL Query Format

- The keyword **OPTIONAL** selects optional elements from the RDF graph
- complies to a Left Outer Join

```
# Default Graph (stored at http://example.org/addresses)
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

_:a  rdf:type   foaf:Person .
_:a  foaf:name  "Alice" .
_:a  foaf:mbox  <mailto:alice@example.com> .
_:a  foaf:mbox  <mailto:alice@work.example> .

_:b  rdf:type   foaf:Person .
_:b  foaf:name  "Bob" .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
FROM <http://example.org/addresses>
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x  foaf:mbox ?mbox }
}
```

Result:

Alice	mailto:alice@example.com
Alice	mailto:alice@work.example
Bob	

다중 OPTIONAL 조건

만족하는 조건만 적용해서 결과 반환

```
@prefix foaf:        <http://xmlns.com/foaf/0.1/> .
@prefix dc:          <http://purl.org/dc/elements/1.1/> .
@prefix xsd:         <http://www.w3.org/2001/XMLSchema#> .

_:a  foaf:givenName  "Alice".
_:a   foaf:birthday  “2003-03-03T03:03:03Z”^^xsd:dateTime .
_:b  foaf:givenName  "Bob" .
_:b  dc:date         "2005-04-04T04:04:04Z"^^xsd:dateTime .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
SELECT ?name ?date
 WHERE {
         ?x foaf:givenName  ?name .
         OPTIONAL { ?x dc:date ?date } .
         OPTIONAL { ?x foaf:birthday ?date } .
}
```

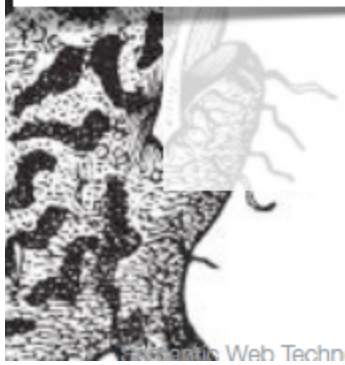
Result:

“Alice”	“2003-03-03T03:03:03Z”^^xsd:dateTime
“Bob”	“2005-04-04T04:04:04Z"^^xsd:dateTime

SPARQL Query Format

- The keyword **UNION** allows for alternatives (logical disjunction)

```
# Default Graph (stored at http://example.org/book)
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .

_:a  dc10:title     "SPARQL Query Language Tutorial" .
_:a  dc10:creator   "Alice" .

_:b  dc11:title     "SPARQL Protocol Tutorial" .
_:b  dc11:creator   "Bob" .

_:c  dc10:title     "SPARQL" .
_:c  dc11:title     "SPARQL (updated)" .
```



```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>
PREFIX dc11:  <http://purl.org/dc/elements/1.1/>

SELECT ?title
FROM <http://example.org/books>
WHERE  { { ?book dc10:title  ?title } UNION {
           ?book dc11:title  ?title }  }
```

Result: "SPARQL Query Language Tutorial"
"SPARQL Protocol Tutorial"
"SPARQL"

SPARQL Query Format

- **Negation** in SPARQL
- (complies to ‚NOT EXISTS' in SQL)

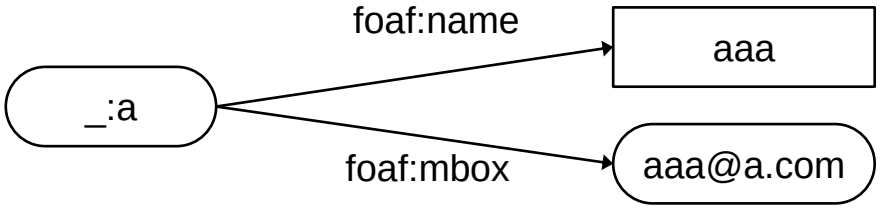
```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
@prefix dc:         <http://purl.org/dc/elements/1.1/> .
@prefix xsd:        <http://www.w3.org/2001/XMLSchema#> .

_:a  foaf:givenName  "Alice".

_:b  foaf:givenName  "Bob" .
_:b  dc:date         "2005-04-04T04:04:04Z"^^xsd:dateTime .
```

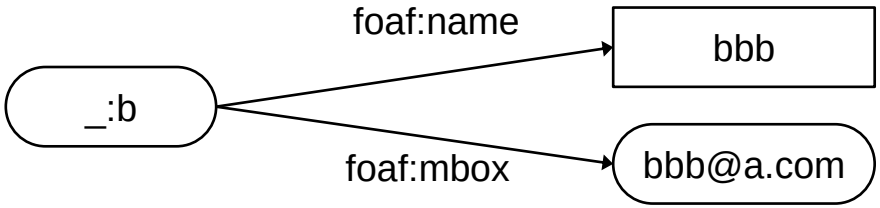
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
SELECT ?name
 WHERE {
         ?x foaf:givenName  ?name .
         OPTIONAL { ?x dc:date ?date } .
         FILTER (!bound(?date))
}
```

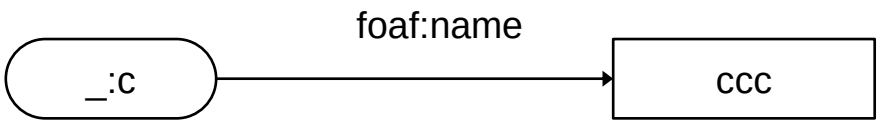
Result: "Alice"

SPARQL Query Format

- The keyword **OPTIONAL** selects optional elements from the RDF graph
- complies to a Left Outer Join





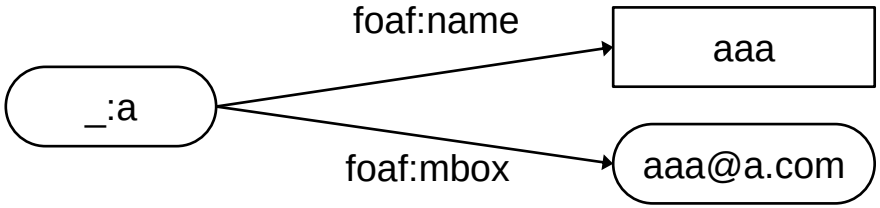


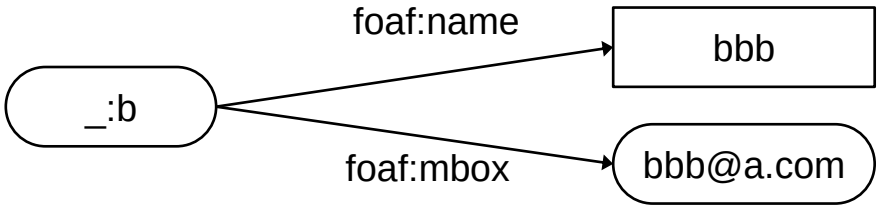

```
SELECT ?name  ?mail
WHERE {
  ?x foaf:name ?name .
  OPTIONAL {
     ?x foaf:mbox ?mail .
  }
  FILTER( bound(?mail) )
}
```

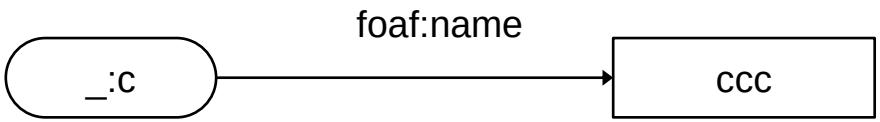
?name	?mail
aaa	aaa@a.com
bbb	bbb@b.com

SPARQL Query Format

- The keyword **OPTIONAL** selects optional elements from the RDF graph
- complies to a Left Outer Join








```
SELECT ?name  ?mail
WHERE {
  ?x foaf:name ?name .
  OPTIONAL {
     ?x foaf:mbox ?mail .
  }
  FILTER( !bound(?mail) )
}
```

?name	?mail
ccc	

SPARQL Query Format

- **Negation** in SPARQL
- (complies to ‚NOT EXISTS' in SQL)

```
@prefix foaf:        <http://xmlns.com/foaf/0.1/> .
@prefix dc:          <http://purl.org/dc/elements/1.1/> .
@prefix xsd:         <http://www.w3.org/2001/XMLSchema#> .

_:a  foaf:givenName  "Alice".

_:b  foaf:givenName  "Bob" .
_:b  dc:date         "2005-04-04T04:04:04Z"^^xsd:dateTime .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>

SELECT ?name ?date
 WHERE {
         ?x foaf:givenName  ?name .
         OPTIONAL { ?x dc:date ?date } .
         FILTER ( !bound(?date) || ?date > "2003-03-03T00:00:00Z"^^xsd:dateTime ) .
}
```

Filtering of Record

Check when the data has ‘date’ value

Result:

“Alice”	
“Bob”	“2005-04-04T04:04:04Z"^^xsd:dateTime

SPARQL Query Format

- **Negation** in SPARQL
- (complies to ‚NOT EXISTS' in SQL)

```
@prefix foaf:        <http://xmlns.com/foaf/0.1/> .
@prefix dc:          <http://purl.org/dc/elements/1.1/> .
@prefix xsd:         <http://www.w3.org/2001/XMLSchema#> .

_:a  foaf:givenName  "Alice".

_:b  foaf:givenName  "Bob" .
_:b  dc:date         "2005-04-04T04:04:04Z"^^xsd:dateTime .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
SELECT ?name ?date
 WHERE {
         ?x foaf:givenName  ?name .
          OPTIONAL { ?x   ?dc:date   ?date .
                     FILTER ( ?date > " 2003-03-03T00:00:00Z"^^xsd:dateTime ) .
                   }
 }
```

FILTER in OPTIONAL Clause

Check if the variable meets the condition

Result:

“Alice”	
“Bob”	“2005-04-04T04:04:04Z"^^xsd:dateTime

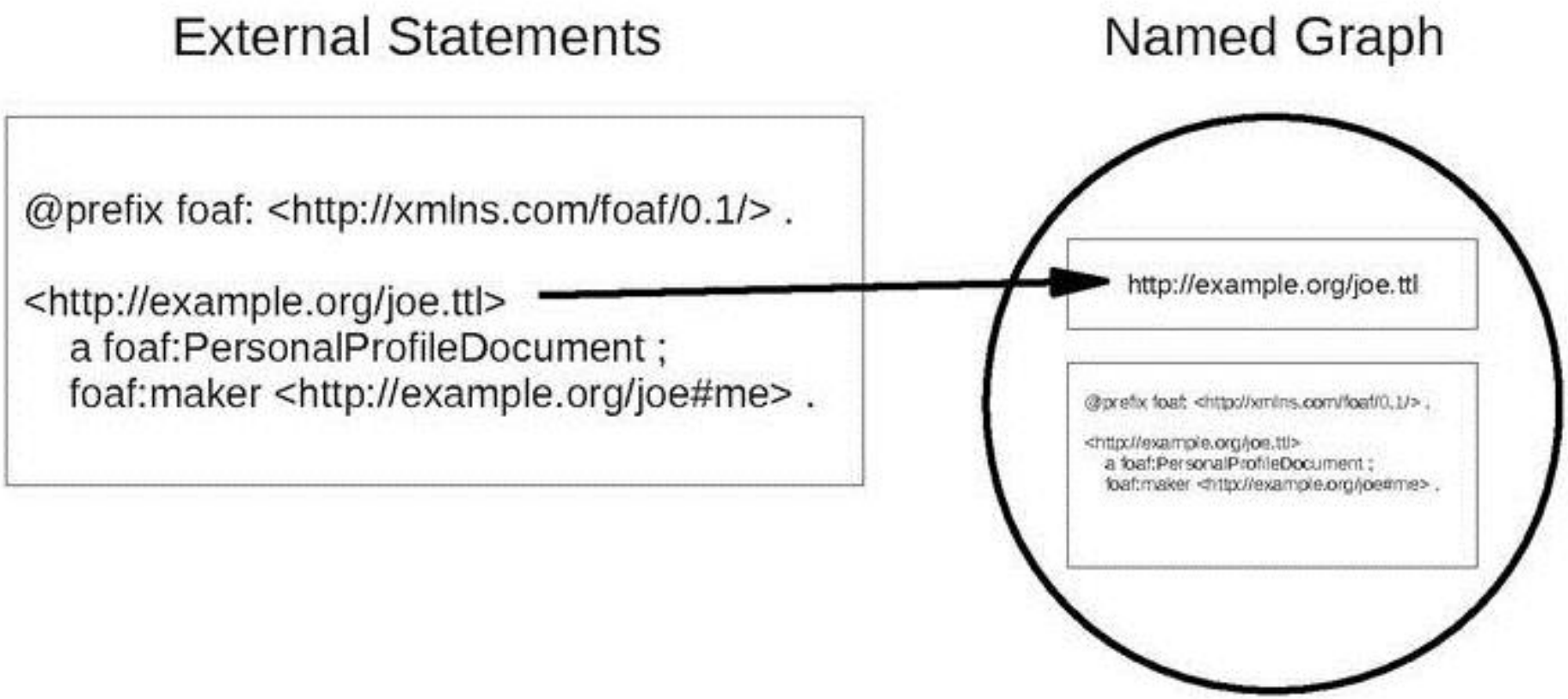
SPARQL Query Format

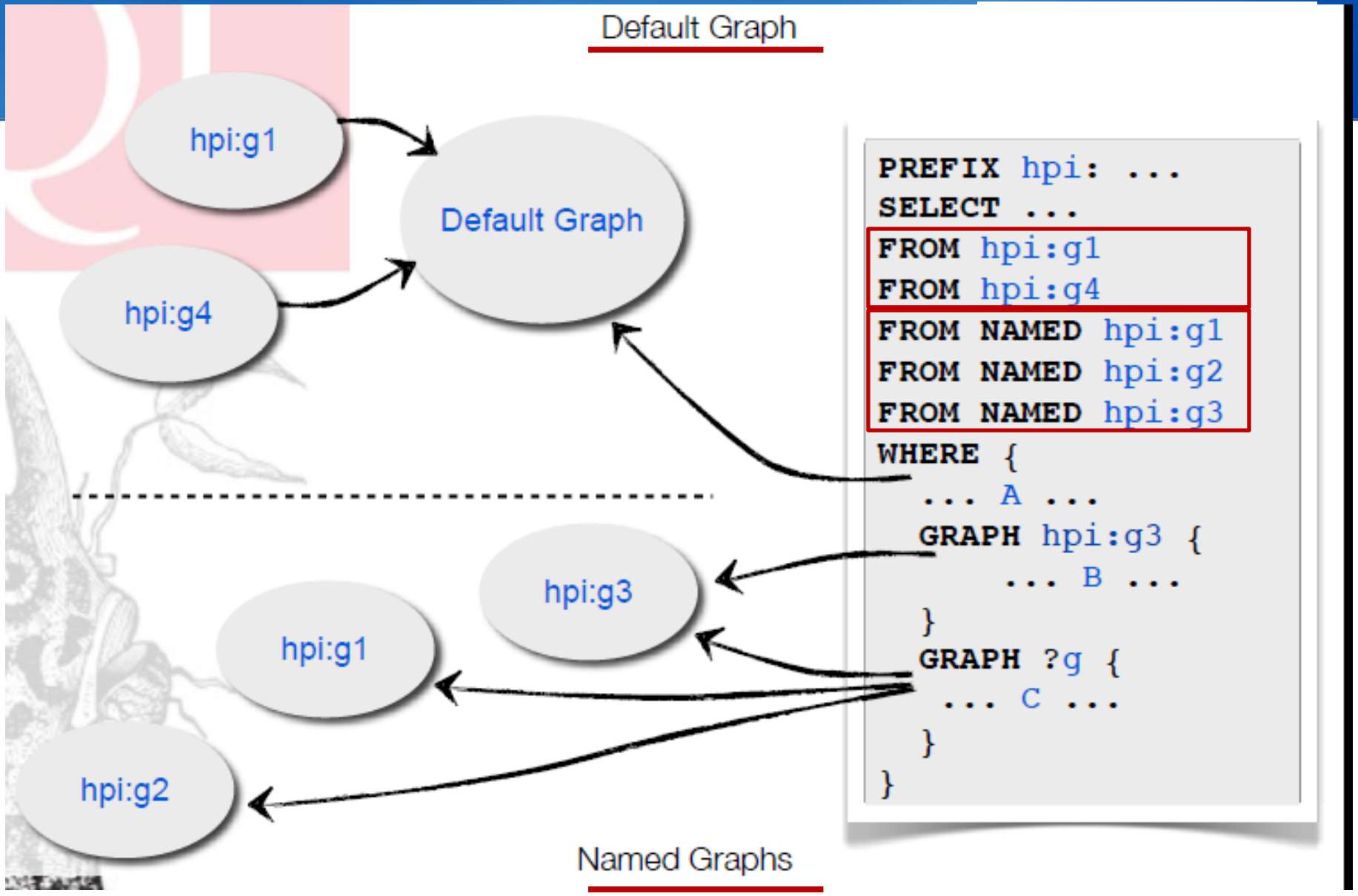
- SPARQL queries are executed over an RDF dataset
  - one (or more) default RDF graph
  - zero or more named RDF graphs
- **Named Graphs** can explicitely addressed via keyword **GRAPH** and the URI of the named graph

```
GRAPH <http://example.org/graph1.rdf> {
        ?x foaf:mbox ?mbox
}
```

Named Graph

- **Formalization of the intuitive idea**
  - 웹 상의 RDF 문서 (그래프) 내용을 Named Graph라고 한다
  - 그 문서의 URI로 문서를 지칭/명명할 수 있기 때문에 Named Graph




Default Graph
hpi:g1
hpi:g4
Default Graph
PREFIX hpi: ...
SELECT ...
FROM hpi:g1
FROM hpi:g4
FROM NAMED hpi:g1
FROM NAMED hpi:g2
FROM NAMED hpi:g3
WHERE {
... A ...
GRAPH hpi:g3 {
... B ...
}
GRAPH ?g {
... C ...
}
}
hpi:g3
hpi:g1
hpi:g2
Named Graphs

Example

```
PREFIX ex: <http://example.org/>

SELECT ?s ?g ?x
FROM ex:graph1
FROM NAMED ex:graph2
WHERE {
  ?s ?p ?o .
  GRAPH ?g { ?x ?y ?z . }
}
```

```
 s                    | g           | x
======================|=============|=====================
 <graph1:subject>     | ex:graph2   | <graph2:subject>
```

SPARQL Query Format

- Example for Named Graphs

Default Graph
(stored at http://example.org/dft.ttl)

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .

<http://example.org/bob>    dc:publisher  "Bob Hacker" .
<http://example.org/alice>  dc:publisher  "Alice Hacker" .
```

Named Graph: http://example.org/bob

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Bob" .
_:a foaf:mbox <mailto:bob@oldcorp.example.org> .
```

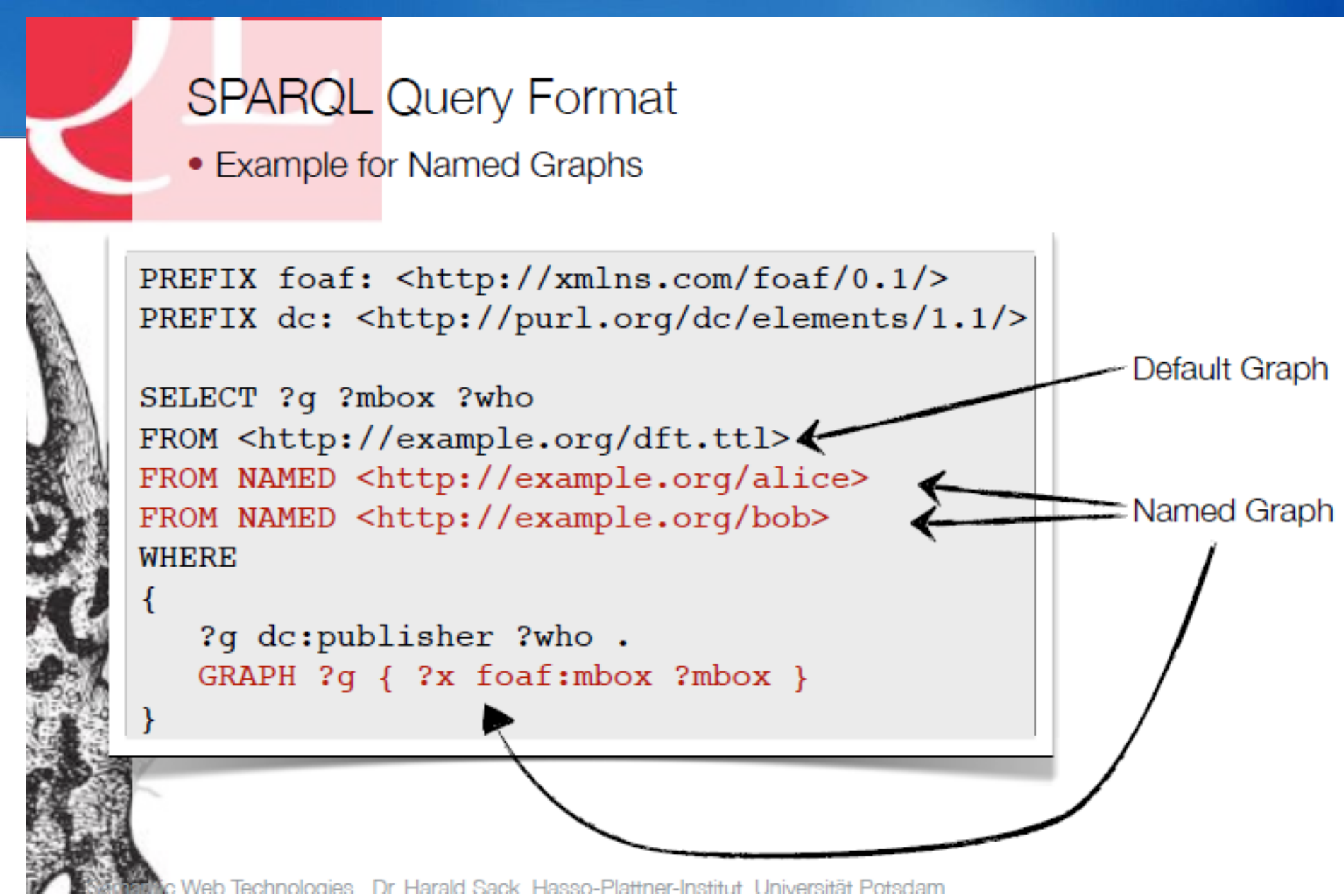
Named Graph: http://example.org/alice

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example.org> .
```

SPARQL Query Format

- Example for Named Graphs

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?g ?mbox ?who
FROM <http://example.org/dft.ttl>
FROM NAMED <http://example.org/alice>
FROM NAMED <http://example.org/bob>
WHERE
{
   ?g dc:publisher ?who .
   GRAPH ?g { ?x foaf:mbox ?mbox }
}
```



SPARQL is not only a query language

- **SPARQL** Protocol **a**nd **R**DF **Q**uery **L**anguage ist
  - a **Query Language** for RDF Graph Traversal
    (*SPARQL Query Language Specification*)
  - a **Protocol Layer**, to use SPARQL via http
    (*SPARQL Protocol for RDF Specification*)
  - an **XML Output Format Specification** for SPARQL Queries
    (*SPARQL Query XML Results Format*)

SPARQL Result Format

- SPARQL results are given as well formed and valid XML documents

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  ...
</sparql>
```

- In a **<head>** element all variables of the SPARQL query are listed

```
<head>
    <variable name="x"/>
    <variable name="hpage"/>
    <variable name="name"/>
    <variable name="mbox"/>
    <variable name="blurb"/>
</head>
```

SPARQL Result Format

- For each SPARQL Query result exists a **<result>** element

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="x"/>
    ...
  </head>
  <results>
    <result>
      <binding name="x"> ... </binding>
      <binding name="hpage"> ... </binding>
    </result>

    <result> ... </result>
    ...
  </results>
</sparql>
```

SPARQL Result Format

- Within a **<binding>** element a <head> variable is bound to a result

```
<result>
      <binding name="x">
        <bnode>r2</bnode>
      </binding>
      <binding name="hpage">
        <uri>http://work.example.org/bob/</uri>
      </binding>
      <binding name="name">
        <literal xml:lang="en">Bob</literal>
      </binding>
      <binding name="age">
        <literal datatype="http://www.w3.org/2001/XMLSchema#integer">
          30
        </literal>
      </binding>
      <binding name="mbox">
        <uri>mailto:bob@work.example.org</uri>
      </binding>
    </result>
```

SPARQL Query Format (continued)

- In addition to SELECT queries SPARQL allows:
  - **ASK**
    - Check whether there is at least one result
    - Result: true or false
    - Result is delivered as XML or JSON
  - **CONSTRUCT**
    - Result: an RDF graph constructed from a template
    - Template: graph pattern with variables from the query pattern
    - Result is RDF/XML or Turtle
  - **DESCRIBE**
    - Result: an RDF graph with data about resources
    - Result is RDF/XML or Turtle

SPARQL Construction of RDF Graphs

- CONSTRUCT defines a template for the construction of new RDF Graphs

data

```
@prefix org:    <http://example.com/ns#> .

_:a  org:employeeName    "Alice" .
_:a  org:employeeId      12345 .

_:b  org:employeeName    "Bob" .
_:b  org:employeeId      67890 .
```

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
PREFIX org:    <http://example.com/ns#>

CONSTRUCT { ?x foaf:name ?name }
WHERE  { ?x org:employeeName ?name }
```

- Result of a CONSTRUCT query as serialized RDF/XML

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <rdf:Description>
    <foaf:name>Alice</foaf:name>
  </rdf:Description>
  <rdf:Description>
    <foaf:name>Bob</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

SPARQL Protocol

- Method to query/respond of SPARQL queries via http
- A SPARQL URI consists of 3 parts:

1. URL of a SPARQL endpoint
   (e.g. http://example.org/sparql)
2. RDF Graph(s) to be queried
   (optional, part of the query string,
   z.B. named-graph-uri=http://example.org/testrdf.rdf)
3. Query string
   (part of the query string, e.g. query=SELECT...)

```
http://example.org/sparql?named-graph-uri=http%3A%2F%2Fexample.org%2Ftestrdf&
query=SELECT+%3Freview_graph+WHERE+%7B%0D%0A++GRAPH+%3Frev
iew_graph+%7B%0D%0A+++++%3Freview+rev%3Arating+10+.%0D%0A++
%7D%0D%0A%7D
```

SPARQL Protocol -- Example

- Simple SPARQL Query

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?book ?who
WHERE { ?book dc:creator ?who }
```

- HTTP Trace of the SPARQL Query

```
GET /sparql/?query=EncodedQuery&default-graph-uri=http://
www.other.example/books HTTP/1.1
Host: www.other.example
User-agent: my-sparql-client/0.1
```

HTTP Trace of the SPARQL Response

```
HTTP/1.1 200 OK
Date: Fri, 06 May 2008 20:55:12 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.4 DAV/1.0.3
Connection: close
Content-Type: application/sparql-results+xml
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
 <head>
   <variable name="book"/>
   <variable name="who"/>
 </head>
 <results ordered="false" distinct="false">
   <result>
     <binding name="who">
       <literal>Bob Hacker</literal>
     </binding>
     <binding name="book">
       <literal>The Art of Hacking</literal>
     </binding> ...
</sparql>
```

- some popular SPARQL (1.0) Endpoints
  - http://sparql.org/sparql.html --
    General Purpose SPARQL Endpoint
  - http://dbpedia.org/sparql --
    DBpedia SPARQL Endpoint
  - http://www4.wiwiss.fu-berlin.de/dblp/sparql --
    DBLP (Computer Science Bibliographies) SPARQL Endpoint
  - http://linkedmdb.org/sparql --
    Linked Movie Database SPARQL Endpoint
  - http://www4.wiwiss.fu-berlin.de/factbook/sparql --
    CIA World Factbook SPARQL Endpoint
  - ...

SPARQL 1.1 -- New Features

- SPARQL 1.1 Query:
  - Assignments (e.g. BIND, SELECT expressions)
  - Aggregate functions (e.g. COUNT, SUM, AVG)
  - Subqueries
  - Negation (EXISTS, NOT EXISTS, MINUS)
  - Property paths
  - Basic query federation (SERVICE, BINDINGS)
- SPARQL 1.1 Update:
  - Graph update (INSERT DATA, DELETE DATA, INSERT, DELETE, DELETE WHERE, LOAD, CLEAR)
  - Graph management (CREATE, DROP, COPY, MOVE, ADD)
- SPARQL 1.1 Entailment for RDF, RDFS, OWL, RIF
- SPARQL 1.1 Service Descriptions

SPARQL 1.1 -- Assignments

- SPARQL 1.1 allows the creation of new values in a query

```
PREFIX ex: <http://example.org/>
SELECT ?Item (?Pr*1.1 AS ?NewP )
WHERE { ?Item ex:price ?Pr }
```

Data

```
@prefix ex: <http://example.org/> .

ex:lemonade1      ex:price 3 .
ex:beer1          ex:price 3.
ex:wine1          ex:price 3.50 .
ex:liqueur1       ex:price "n/a".
```

Result

?Item	?NewP
lemonade1	3.3
beer1	3.3
wine1	3.85
liqueur1	

SPARQL 1.1 -- Aggregate Functions

- SPARQL 1.1 allows the use of aggregate functions in queries

```
PREFIX ex: <http://example.org/>
SELECT (Count(?Item) AS ?C)
WHERE { ?Item ex:price ?Pr }
```

Data

```
@prefix ex: <http://example.org/> .

ex:lemonade1    ex:price 3 ;
                rdf:type ex:Softdrink.
ex:beer1        ex:price 3;
                rdf:type ex:Beer.
ex:wine1        ex:price 3.50 ;
                rdf:type ex:Wine.
ex:wine2        ex:price 4 .
                rdf:type ex:Wine.
ex:wine3        ex:price "n/a";
                rdf:type ex:Wine.
```

Result

?C
5

SPARQL 1.1 -- Aggregate Functions

- SPARQL 1.1 allows the use of aggregate functions in queries

```
PREFIX ex: <http://example.org/>
SELECT (Count(DISTINCT ?T) AS ?C)
WHERE { ?Item rdf:type ?T }
```

Data

```
@prefix ex: <http://example.org/> .

ex:lemonade1    ex:price 3 ;
                rdf:type ex:Softdrink.
ex:beer1        ex:price 3;
                rdf:type ex:Beer.
ex:wine1        ex:price 3.50 ;
                rdf:type ex:Wine.
ex:wine2        ex:price 4 .
                rdf:type ex:Wine.
ex:wine3        ex:price "n/a";
                rdf:type ex:Wine.
```

Result

?C
3

SPARQL 1.1 -- Aggregate Functions

- SPARQL 1.1 allows the use of aggregate functions in queries

```
PREFIX ex: <http://example.org/>
SELECT ?T (Count(?Item) AS ?C)
WHERE { ?Item rdf:type ?T }
GROUP BY ?T
```

Data

```
@prefix ex: <http://example.org/> .

ex:lemonade1    ex:price 3 ;
                rdf:type ex:Softdrink.
ex:beer1        ex:price 3;
                rdf:type ex:Beer.
ex:wine1        ex:price 3.50 ;
                rdf:type ex:Wine.
ex:wine2        ex:price 4 .
                rdf:type ex:Wine.
ex:wine3        ex:price "n/a";
                rdf:type ex:Wine.
```

Result

?T	?C
Softdrink	1
Beer	1
Wine	3

SPARQL 1.1 -- Aggregate Functions

- SPARQL 1.1 allows the use of aggregate functions in queries

```
PREFIX ex: <http://example.org/>
SELECT ?T (Count(?Item) AS ?C)
WHERE { ?Item rdf:type ?T }
GROUP BY ?T
HAVING Count(?Item) > 1
```

Data

```
@prefix ex: <http://example.org/> .

ex:lemonade1    ex:price 3 ;
                rdf:type ex:Softdrink.
ex:beer1        ex:price 3;
                rdf:type ex:Beer.
ex:wine1        ex:price 3.50 ;
                rdf:type ex:Wine.
ex:wine2        ex:price 4 .
                rdf:type ex:Wine.
ex:wine3        ex:price "n/a";
                rdf:type ex:Wine.
```

Result

?T	?C
Wine	3

SPARQL 1.1 -- Aggregate Functions

- more aggregate functions
  - **SUM**
  - **AVG**
  - **MIN**
  - **MAX**
  - **SAMPLE** -- „pick“ one non-deterministically
  - **GROUP_CONCAT** -- concatenate values with a designated string separator

SPARQL 1.1 -- Aggregate Functions

- Example for SAMPLE, GROUP_CONCAT

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ( SAMPLE(?N) as ?Name)
       ( GROUP_CONCAT(?M; SEPARATOR = ", ") AS ?Nicknames )
WHERE { ?P a foaf:Person ;
           foaf:name ?N ;
           foaf:nick ?M . }
GROUP BY ?P
```

Data

```
@prefix ex: <http://example.org/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

ex:alice a foaf:Person; foaf:name "Alice Wonderland";
          foaf:nick "Alice", "The real Alice".

ex:bob a foaf:Person;
       foaf:name "Robert Doe", "Robert Charles Doe",
                 "Robert C. Doe";
       foaf:nick "Bob","Bobby","RobC","BobDoe".

ex:charles a foaf:Person;
       foaf:name "Charles Charles";
       foaf:nick "Charlie" .
```

Result

?Name	?Nicknames
Alice Wonderland	The real Alice, Alice
Charles Charles	Charlie
Robert C. Doe	Bob, BobDoe, RobC, Bobby

SPARQL 1.1 -- Subqueries

- Subqueries are a way to embed SPARQL queries within other queries

```
SELECT ?T
WHERE {
 ?D foaf:maker ?P ; rdfs:label ?T .
 {
 SELECT DISTINCT ?P
 WHERE { ?D foaf:maker <http://dblp.l3s.de/…/authors/Harald_Sack>, ?P .
         FILTER ( ?P != <http://dblp.l3s.de/…/authors/Harald_Sack> )
       }
 LIMIT 10
 }
}
```

- result is achieved by first evaluating the inner query

SPARQL 1.1 -- Negation

- Filtering of query solutions is done within a FILTER expression using NOT EXISTS and EXISTS.

```
PREFIX  rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?person
WHERE
{
    ?person rdf:type  foaf:Person .
    FILTER NOT EXISTS { ?person foaf:name ?name }
}
```

Data

```
@prefix  :       <http://example/> .
@prefix  rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix  foaf:   <http://xmlns.com/foaf/0.1/> .

:alice  rdf:type   foaf:Person .
:alice  foaf:name  "Alice" .
:bob    rdf:type   foaf:Person .
```

Result

?person
:bob

SPARQL 1.1 -- Negation

- Filtering of query solutions be removing possible solutions with MINUS.

```
PREFIX :       <http://example/>
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?s
WHERE {
   ?s ?p ?o .
   MINUS {
      ?s foaf:givenName "Bob" .
   }
}
```

Data

```
@prefix :       <http://example/> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .

:alice  foaf:givenName "Alice" ;
        foaf:familyName "Smith" .

:bob    foaf:givenName "Bob" ;
        foaf:familyName "Jones" .

:carol  foaf:givenName "Carol" ;
        foaf:familyName "Smith" .
```

Result

?s
:alice
:carol

RDF in RDBMS

- RDF Graph can be represented by a set of Triples (s,p,o)
- Triples can simply be stored in a **relational database management system** (RDBMS)

- Motivation: Use a specific relational schema for RDF data and benefit from 40 years of research in the DB community

- 3 steps for SPARQL query processing:
  (1) Convert SPARQL query to SQL query (w.r.t. the schema)
  (2) Use RDBMS to answer SQL query
  (3) Generate SPARQL query result from SQL query result

Problem

- How to store RDF triples to carry out efficient SPARQL queries?

- 4 alternatives:
  1. Giant Triple Storage
  2. Property Tables
  3. Vertically Partitioned Tables (Binary Tables)
  4. Hexastore

Giant Triple Storage

- Basic idea:
  - Store all RDF triples in a single table
  - performance depends on efficient indexing

Subj	Property	Obj
ID1	type	FullProfessor
ID1	teacherOf	'AI'
ID1	bachelorFrom	'MIT'
ID1	mastersFrom	'Cambridge'
ID1	phdFrom	'Yale'
ID2	type	AssocProfessor
ID2	worksFor	'MIT'
ID2	teacherOf	'DataBases'
ID2	bachelorsFrom	'Yale'
ID2	phdFrom	'Stanford'
ID3	type	GradStudent
ID3	advisor	ID2
ID3	teachingAssist	'AI'
ID3	bachelorsFrom	'Stanford'
ID3	mastersFrom	'Princeton'
ID4	type	GradStudent
ID4	advisor	ID1
ID4	takesCourse	'DataBases'
ID4	bachelorsFrom	'Columbia'

Pros:
- easy to implement
- works for huge numbers of properties, if indexes are chosen with care

Cons:
- many self joins

Giant Triple Storage

- Basic idea:
  - Store all RDF triples in a single table
  - performance depends on efficient indexing

Subj	Property	Obj
ID1	type	FullProfessor
ID1	teacherOf	'AI'
ID1	bachelorFrom	'MIT'
ID1	mastersFrom	'Cambridge'
ID1	phdFrom	'Yale'
ID2	type	AssocProfessor
ID2	worksFor	'MIT'
ID2	teacherOf	'DataBases'
ID2	bachelorsFrom	'Yale'
ID2	phdFrom	'Stanford'
ID3	type	GradStudent
ID3	advisor	ID2
ID3	teachingAssist	'AI'
ID3	bachelorsFrom	'Stanford'
ID3	mastersFrom	'Princeton'
ID4	type	GradStudent
ID4	advisor	ID1
ID4	takesCourse	'DataBases'
ID4	bachelorsFrom	'Columbia'

SPARQL

```
SELECT ?university WHERE {
   ?v rdf:type :GradStudent;
        :bachelorsFrom ?university . }
```

SQL

```
SELECT t2.o AS university
FROM triples AS t1, triples AS t2
WHERE t1.p='type' AND
   t1.o='GradStudent' AND
   t2.p='bachelorsFrom' AND
   t1.s=t2.s
```

ID Based Triple Storage

- Use numerical identifier for each RDF term in the dataset
- saves space and enhances efficiency

RDF Term	ID
:ID1	1
rdf:type	2
:FullProfessor	3
:teacherOf	4
"AI"	5
:bachelorsFrom	6
"MIT"	7
:mastersFrom	8
"Cambridge	9

s	p	o
1	2	3
1	4	5
1	6	7
1	8	9
1	10	11
12	13	14
12	15	7
12	4	16

Quad Tables

- Storing **multiple** RDF graphs
- used for provenance, versioning, contexts, etc.

RDF Term	ID
:ID1	1
rdf:type	2
:FullProfessor	3
:teacherOf	4
"AI"	5
:bachelorsFrom	6
"MIT"	7
:mastersFrom	8
"Cambridge	9

g	s	p	o
100	1	2	3
100	1	4	5
101	12	16	7
100	1	8	9
102	23	21	11
100	1	6	7
102	23	22	8
101	12	4	16