# Grundlagen Semantic Web

Lehrveranstaltung im WS11/12
Seminar für Computerlinguistik
Universität Heidelberg

PD Dr. Sebastian Rudolph
Institut AIFB
Karlsruher Institut für Technologie

AIFB

# Agenda



- Vorstellung des Dozenten
- Organisatorisches zur Vorlesung
- Was ist das "Semantic Web"?

# Agenda



- Vorstellung des Dozenten
- Organisatorisches zur Vorlesung
- Was ist das "Semantic Web"?

# Vorstellung des Dozenten

AIFB

Dr. Sebastian Rudolph

- 1995 – 2000 Studium Lehramt Mathematik/Physik/Informatik, TU Dresden
- 2000 – 2003 Stipendiat im Graduiertenkolleg 334, TU Dresden
- 2003 – 2005 wissenschaftlicher Mitarbeiter an der Professur für die Psychologie des Lehrens und Lernens, TU Dresden
- 2006 Promotion in Mathematik, TU Dresden
- seit 2006 als Postdoc, seit 2008 als Projektleiter am AIFB
- 2011 Habilitation in Informatik, Karlsruhe Institute of Technology

Themen:

formale Aspekte der Wissensverarbeitung
Logik
Komplexitätstheorie
Formale Begriffsanalyse
NLP
E-Learning

# Agenda



- Vorstellung des Dozenten
- Organisatorisches zur Vorlesung
- Was ist das "Semantic Web"?

# Organisatorisches: Zeit und Ort

- 4.-7. Oktober 2011, täglich
  - vormittags, 9:15 - 12:45
  - nachmittags, 14:15 - 17:00
  - Vorlesung & Übung flexibel im Wechsel
- Ort: Im Neuenheimer Feld 325, SR 24
- Webseite:
  http://semantic-web-grundlagen.de/wiki/GSW_WS11/12

# Organisatorisches: Inhalt

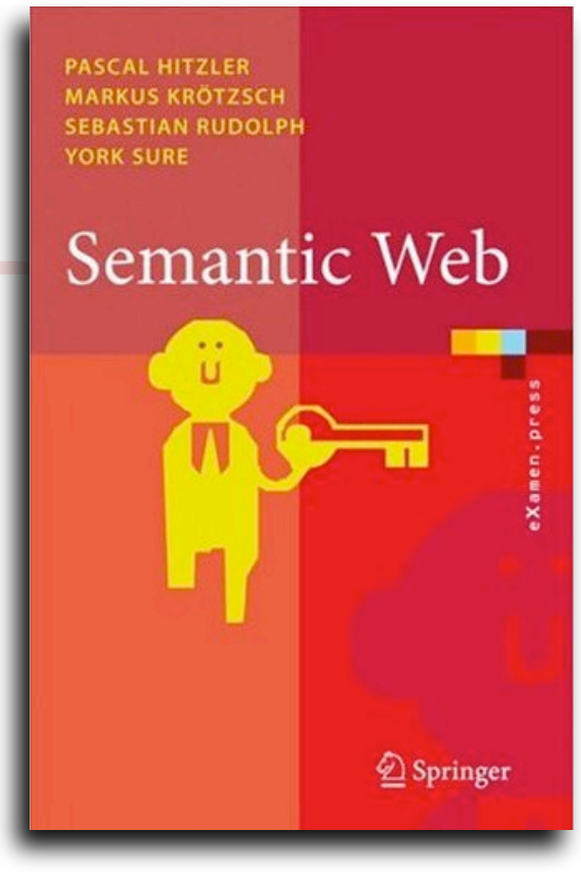
AIFB

| |
|---|
| Einleitung und Ausblick |
| XML und URIs |
| Einführung in RDF |
| RDF Schema |
| Logik - Grundlagen |
| Semantik von RDF(S) |
| OWL - Syntax und Intuition |
| OWL - Semantik und Reasoning |
| OWL 2 |
| SPARQL - Syntax und Intuition |
| Semantik von SPARQL |

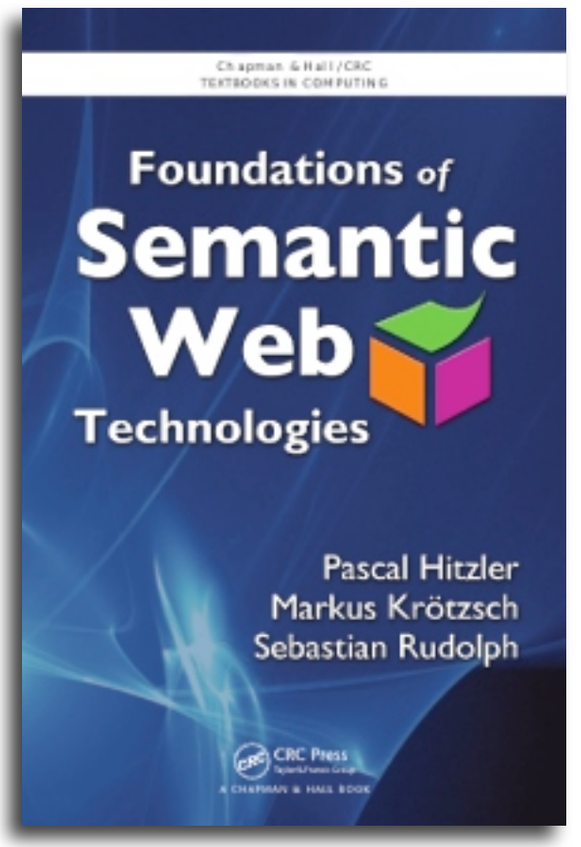
# Literatur

AIFB

- Hitzler, Krötzsch, Rudolph, Sure "Sematic Web. Grundlagen." Springer-Verlag




- Hitzler, Krötzsch, Rudolph "Foundations of Semantic Web Technologies" CRC Press



# Agenda



- Vorstellung des Dozenten
- Organisatorisches zur Vorlesung
- **Was ist das "Semantic Web"?**

# Das Web

AIFB

Das Web flankiert den Übergang von der Industrie- zur Informationsgesellschaft und bietet die Infrastruktur für eine neue Qualität des Umgangs mit Information hinsichtlich Beschaffung wie auch Bereitstellung.

- hohe Verfügbarkeit
- hohe Aktualität
- geringe Kosten

# Das Web

AIFB

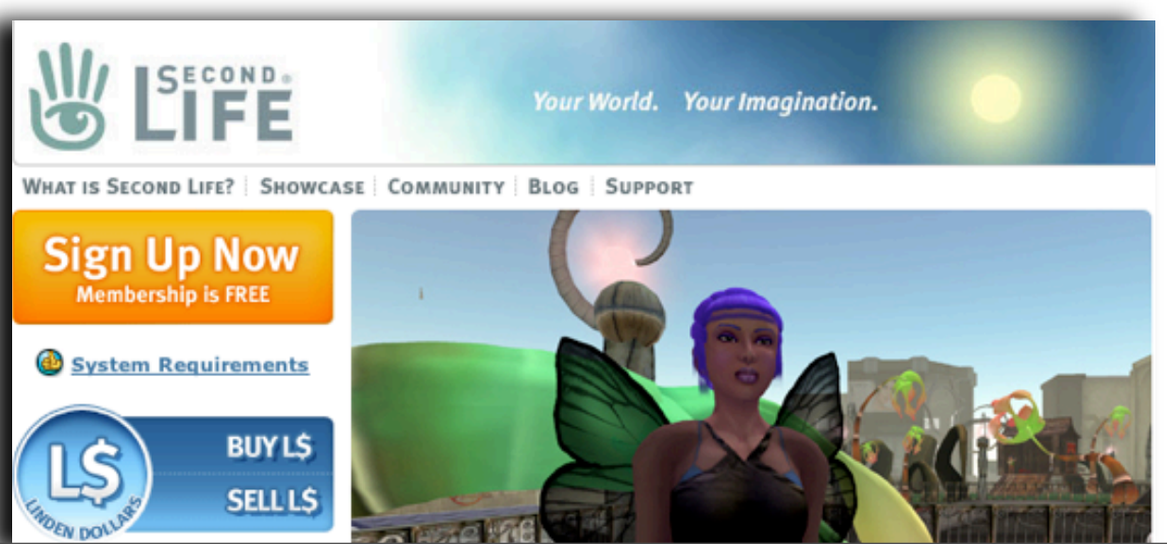
## Kommerzialisierung in allen Größenordnungen

# Das Web



weitere Lebensbereiche werden "webisiert":

- Behörden, Verwaltung (eGovernment)
- Ausbildung (eLearning, eEducation)
- Sozialkontakte (Social-Networking-Plattformen, Partnerbörsen)
- Alltag?



# Warum Semantic Web?

AIFB

## Exkurs: Syntax vs. Semantik

- **Syntax**
  (von grch. συνταξις – *Zusammenstellung, Satzbau*) steht für die (normative) Struktur von Daten, d.h. sie charakterisiert, was "wohlgeformte" Daten sind.

- **Semantik**
  (grch. σημαντικος – *zum Zeichen gehörend*) steht für die Bedeutung von Daten, d.h. sie charakterisiert beispielsweise, welche inhaltliche Schlussfolgerungen sich ziehen lassen.

| *4+)=(* | *3+4=12* | *3+4=7* |
|---|---|---|
| syntaktisch falsch | syntaktisch richtig | syntaktisch richtig |
| -- | semantisch falsch | semantisch richtig |

# Probleme des Web

AIFB

- Fülle an Informationen
- ausgerichtet auf Menschen als Endnutzer
  - Erfassen der Bedeutung einer Webseite
  - Unabhängig von konkreter Repräsentation
  - Bilden von Zusammenhängen

# Probleme des Web



- **Lokalisierung** von Information problematisch
- heutige Suchmaschinen gut, aber stichwortbasiert
- wünschenswert: inhaltliche, *semantische Suche*

# Probleme des Web



- **Heterogenität** der vorhandenen Information auf verschiedensten Ebenen:
  - Zeichenkodierung (z.B. ASCII vs. Unicode)
  - verwendete natürliche Sprachen
  - Anordnung von Information auf Webseiten
- *Informationsintegration*


# Probleme des Web



- **implizites Wissen**, d.h. Informationen, sind nicht explizit spezifiziert, folgen aber aus der Kombination gegebener Daten
- formallogische Methoden erforderlich
- *automatisches Schlussfolgern*

# Probleme des Web



Lösungsansätze:

I. Ad hoc: Verwendung von KI-Methoden zur Auswertung bestehender unstrukturierter Informationen im Web

II. A priori: Strukturierung der Web-Informationen zur Erleichterung der automatisierten Auswertung:
➜ **Semantic Web**

# Semantic Web - Zutaten



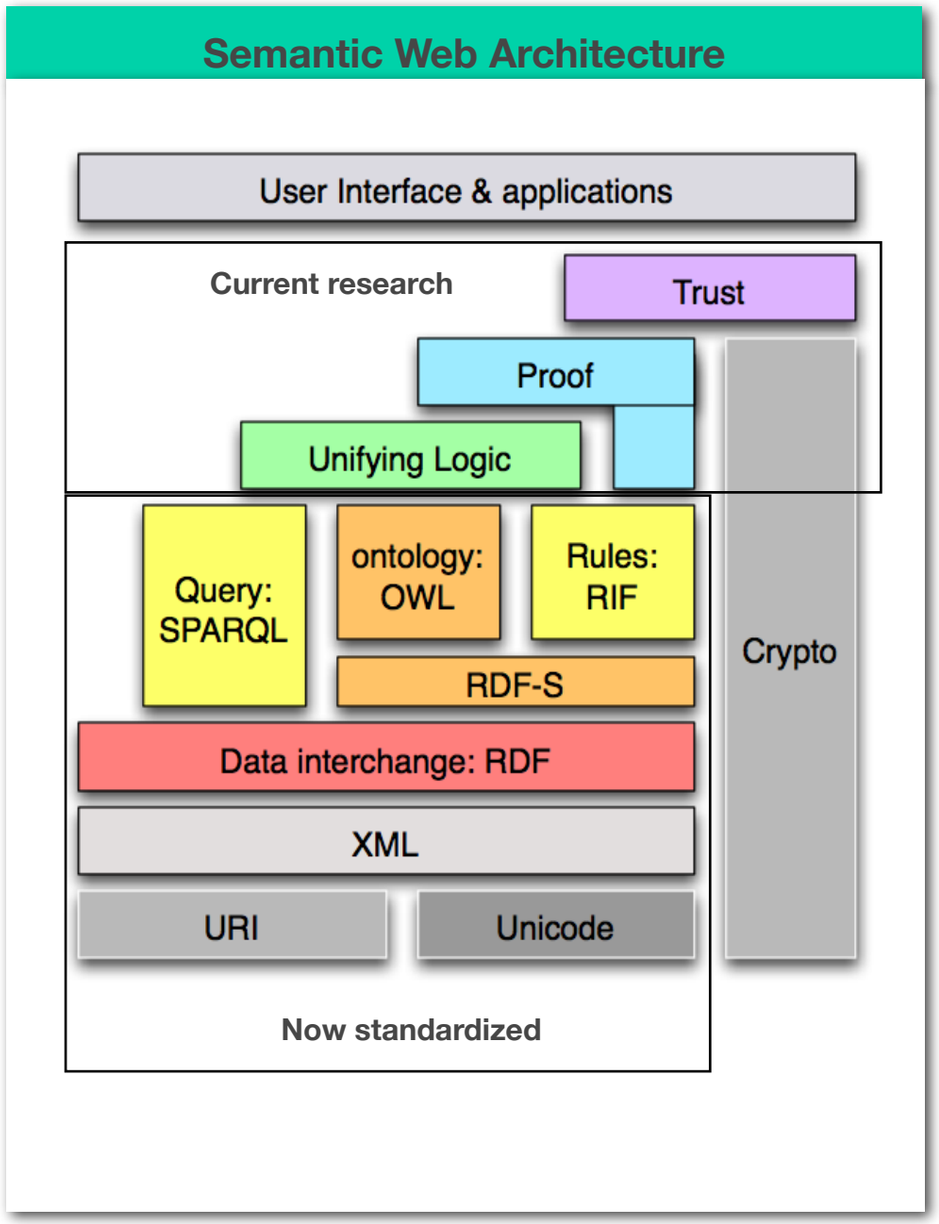
Zwei essentielle Voraussetzungen zur Realisierung:

1. offene Standards zur Beschreibung von Informationen
   - klar definiert
   - flexibel
   - erweiterbar
2. Methoden zur Gewinnung von Informationen aus derlei Beschreibungen

# Semantic Web – Standards

AIFB

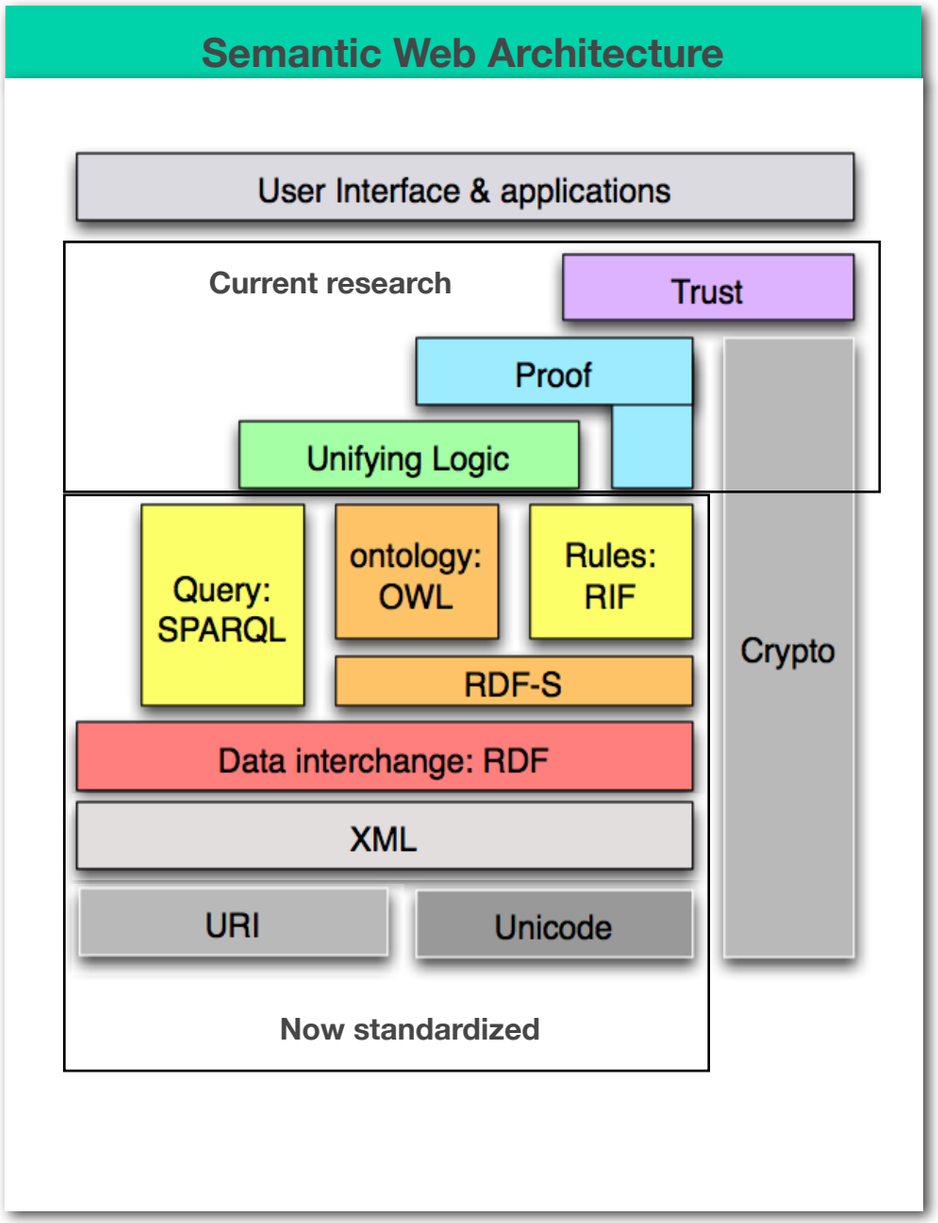
W3C Semantic Web



# XML und URIs

Dr. Sebastian Rudolph





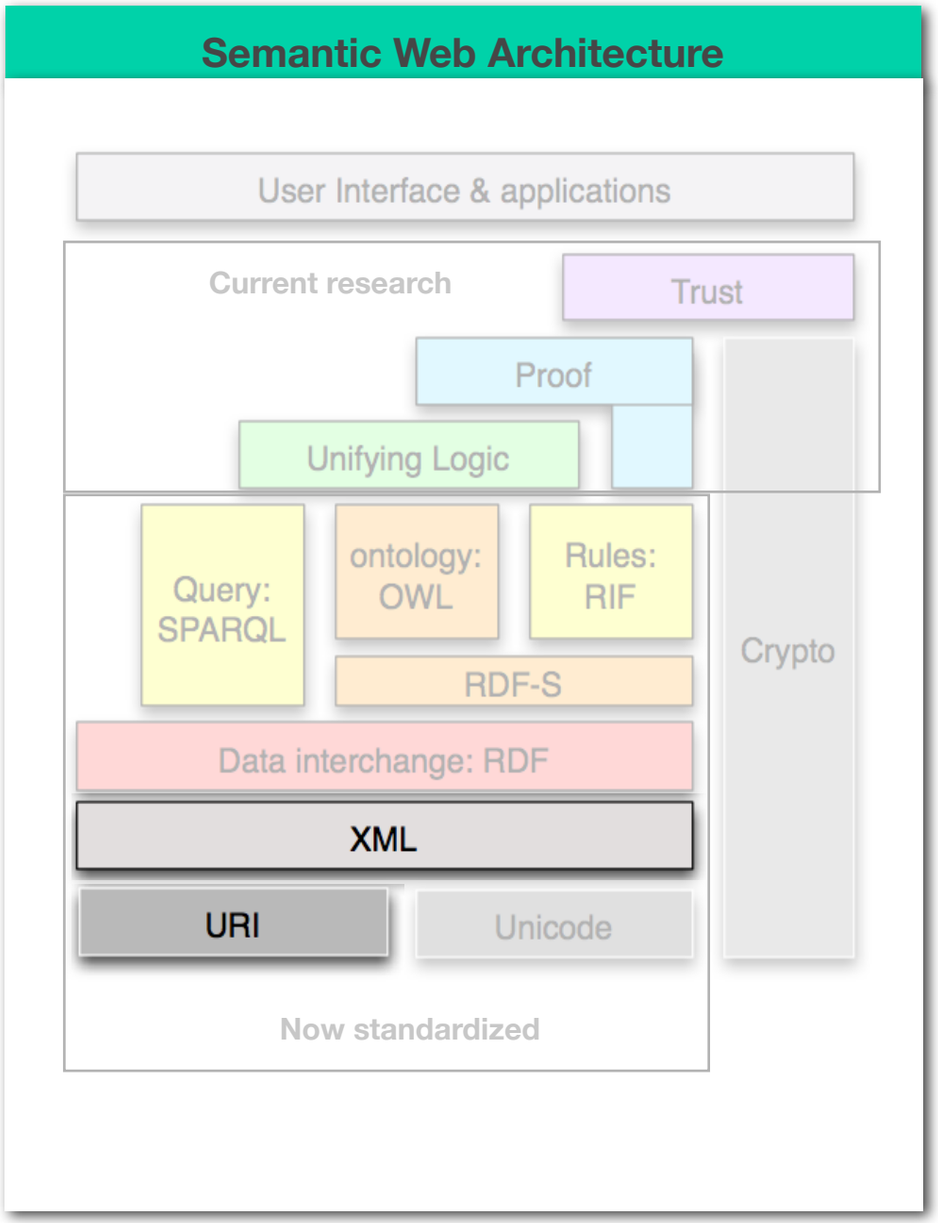



AIFB

# XML und URIs

Dr. Sebastian Rudolph








AIFB

# Agenda



- XML - Motivation/Idee
- XML - Syntax
- URIs
- Namensräume

# Agenda



- XML - Motivation/Idee
- XML - Syntax
- URIs
- Namensräume

# Annotation mit Markupsprachen



- Grundidee des Markup: versehen von (unstrukturiertem) Text mit zusätzlicher Information (bzw. Struktur)

- synonym: *auszeichnen*, auch: *annotieren* von Text

- Text = Daten
  Zusatzinformation = *Metadaten*

# Annotation mit Markupsprachen



- häufige Markup-Strategie: Einschließen des zu annotierenden Textes in sogenannte *tags* (engl.: Etikett, Schild):

<Tag-Bezeichner>... Text ...</Tag-Bezeichner>

*öffnendes Tag* *schließendes Tag*

- Zusatzinformation wird von verarbeitenden Programmen gelesen und interpretiert

# Annotation mit Markupsprachen



- prominentestes Beispiel: HTML tags kodieren Darstellungsinformationen:
  Dieses Buch hat den Titel Semantic Web Grundlagen.
- Darstellung im Browser:
  *Dieses Buch* hat den Titel **Semantic Web Grundlagen**.
- Strategie auch geeignet zur inhaltlichen Annotation, z.B.:
  <Buch>Dieses Buch</Buch> hat den Titel <Titel>Semantic Web Grundlagen</Titel>.

# Annotation mit Markupsprachen



- Verschachtelung von Tags erlaubt:

```
<Vorlesung>
    <Titel>
      XML und URIs
    </Titel>
    <Dozent>
      <Titel>
        Dr.
      </Titel>
      <Vorname>
        Sebastian
      </Vorname>
      <Nachname>
        Rudolph
      </Nachname>
    </Dozent>
</Vorlesung>
```

# Annotation mit Markupsprachen



- Verschachtelung von Tags erlaubt:

```
<Vorlesung>
                                    <Titel>
                                      XML und URIs
                                    </Titel>
                                    <Dozent>
                                                <Titel>
                                                  Dr.
                                                </Titel>
                                                <Vorname>
                                                  Sebastian
                                                </Vorname>
                                                <Nachname>
                                                  Rudolph
                                                </Nachname>
                                    </Dozent>
</Vorlesung>
```

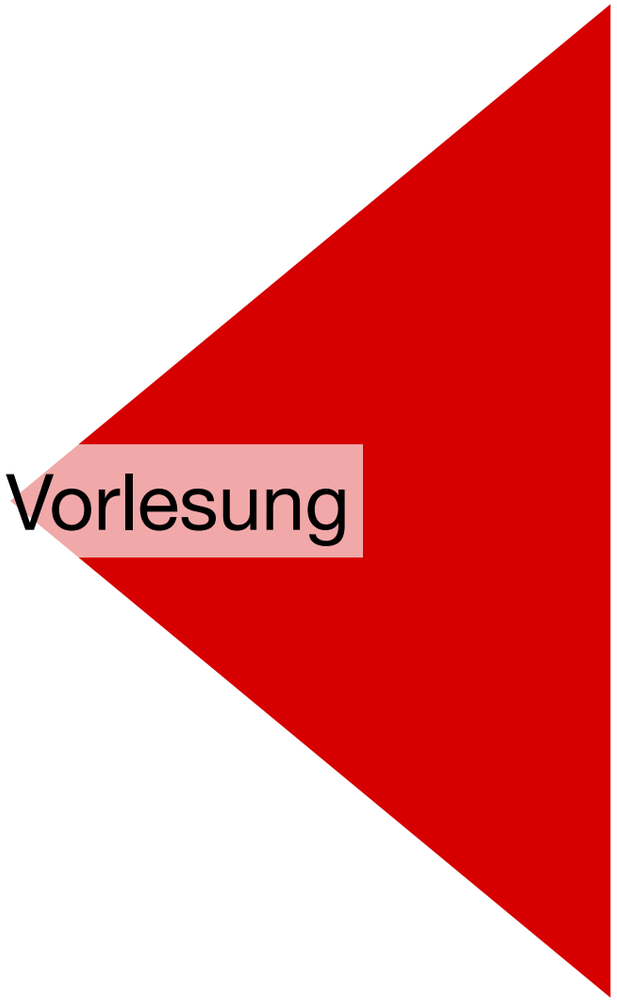
# Annotation mit Markupsprachen



```
<Vorlesung>
                                    <Titel>
                                      XML und URIs
                                    </Titel>
                                    <Dozent>
                                                <Titel>
                                                 Dr.
                                                </Titel>
                                                <Vorname>
                                                 Sebastian
                                                </Vorname>
                                                <Nachname>
                                                 Rudolph
                                                </Nachname>
                                    </Dozent>
</Vorlesung>
```

# Annotation mit Markupsprachen

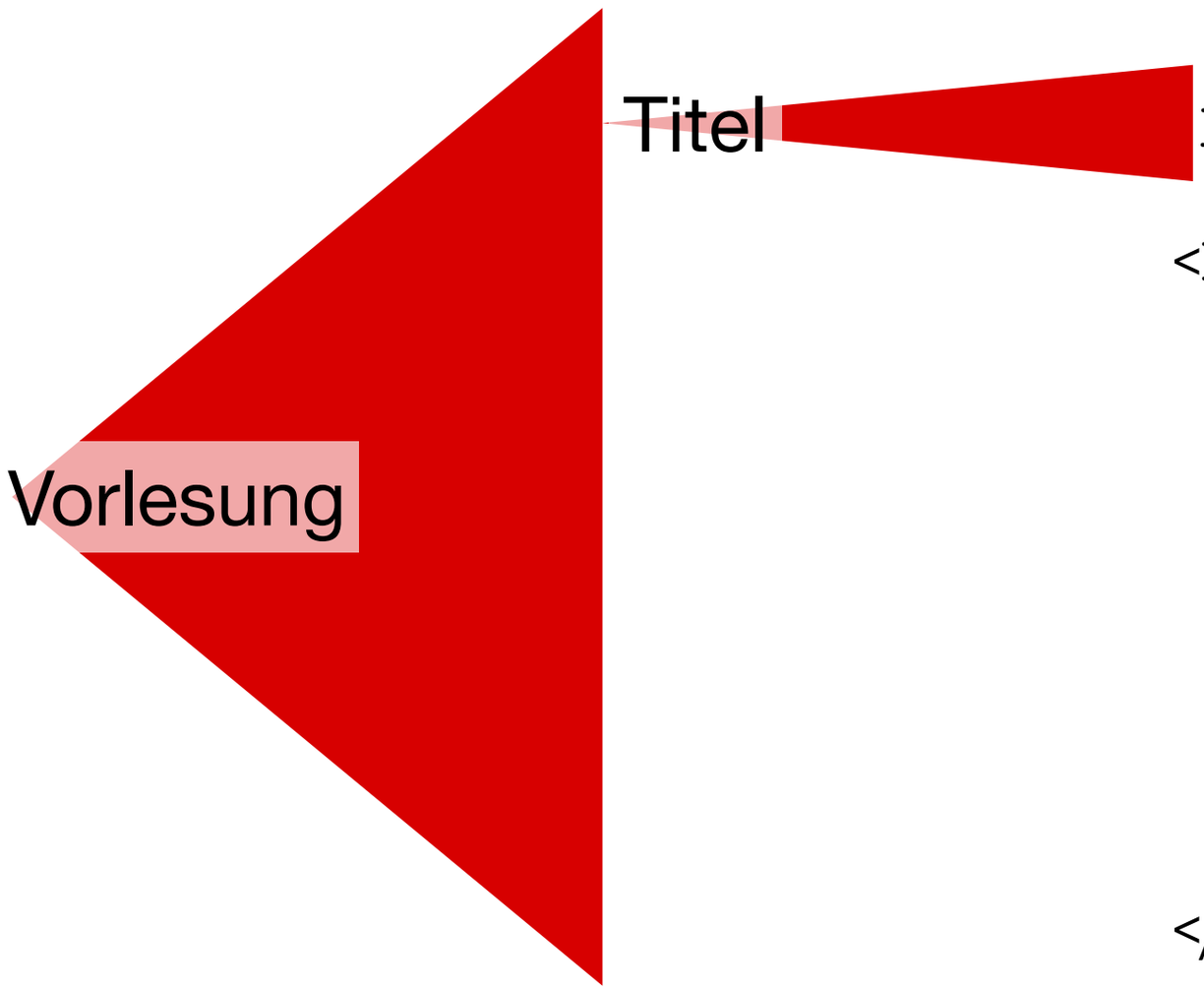
AIFB




```
<Titel>
  XML und URIs
</Titel>
<Dozent>
                    <Titel>
                     Dr.
                    </Titel>
                    <Vorname>
                     Sebastian
                    </Vorname>
                    <Nachname>
                     Rudolph
                    </Nachname>
</Dozent>
```

# Annotation mit Markupsprachen

AIFB

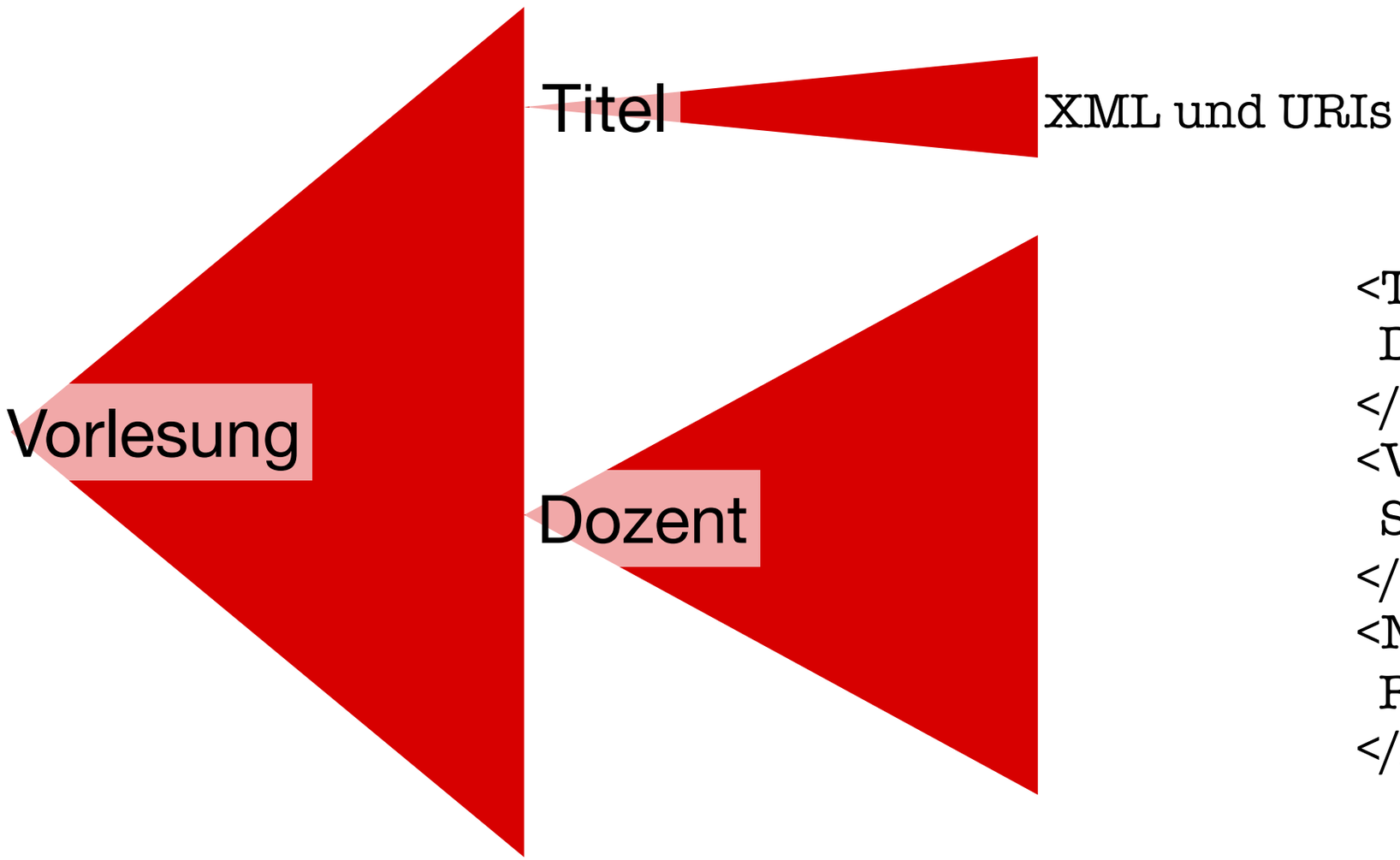



XML und URIs

```
<Dozent>
        <Titel>
          Dr.
        </Titel>
        <Vorname>
          Sebastian
        </Vorname>
        <Nachname>
          Rudolph
        </Nachname>
</Dozent>
```

# Annotation mit Markupsprachen

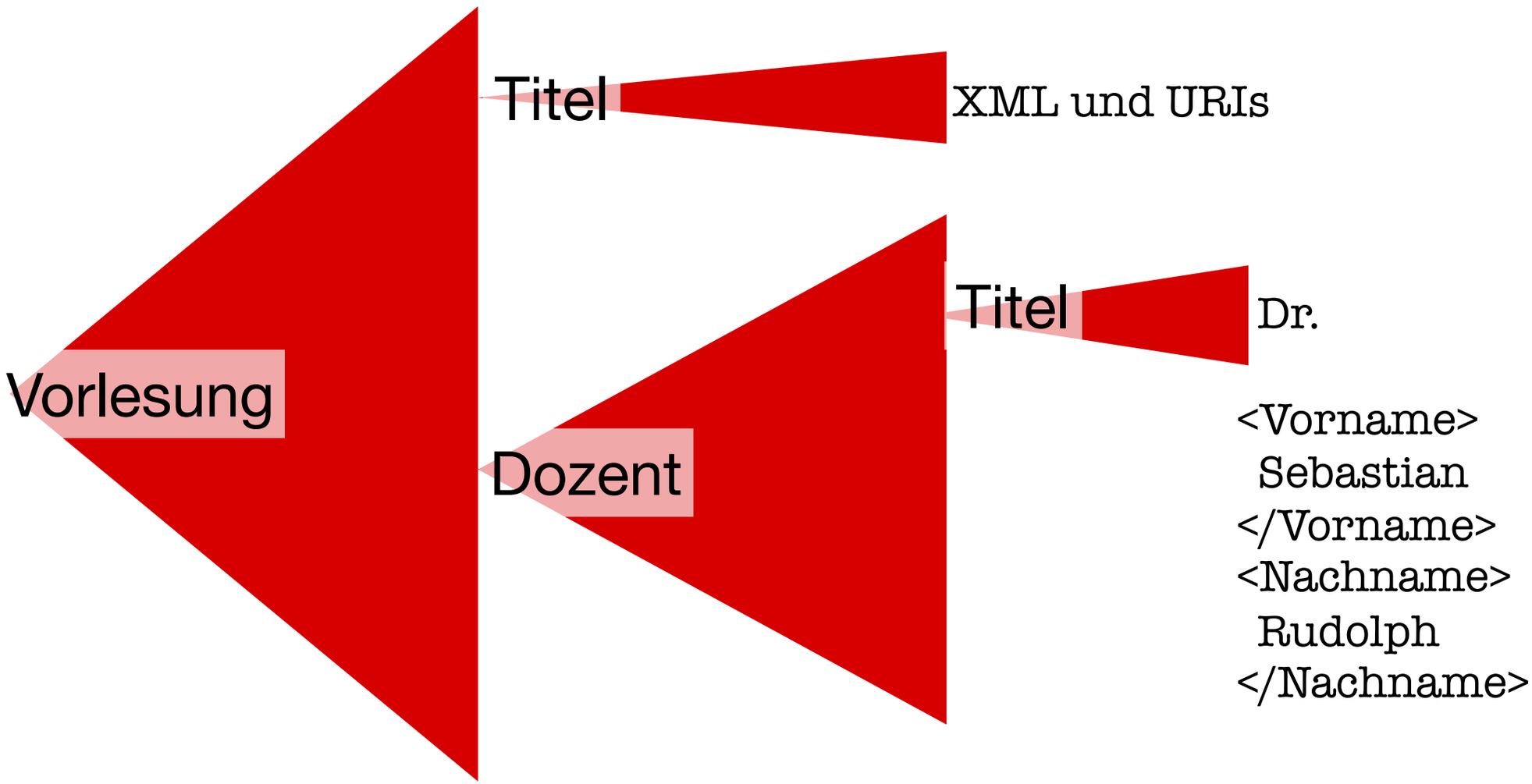
AIFB




```
<Titel>
 Dr.
</Titel>
<Vorname>
 Sebastian
</Vorname>
<Nachname>
 Rudolph
</Nachname>
```

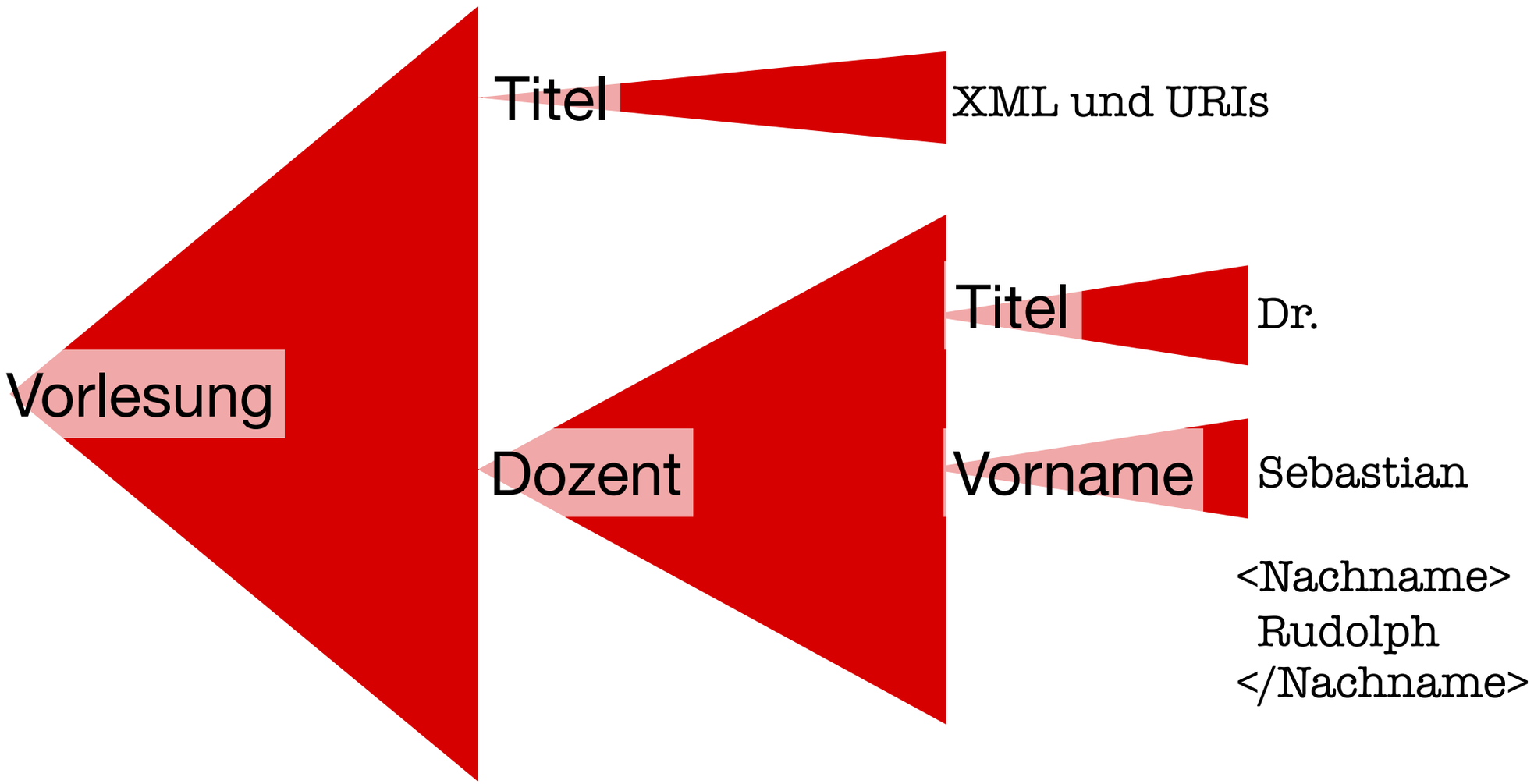
# Annotation mit Markupsprachen

AIFB



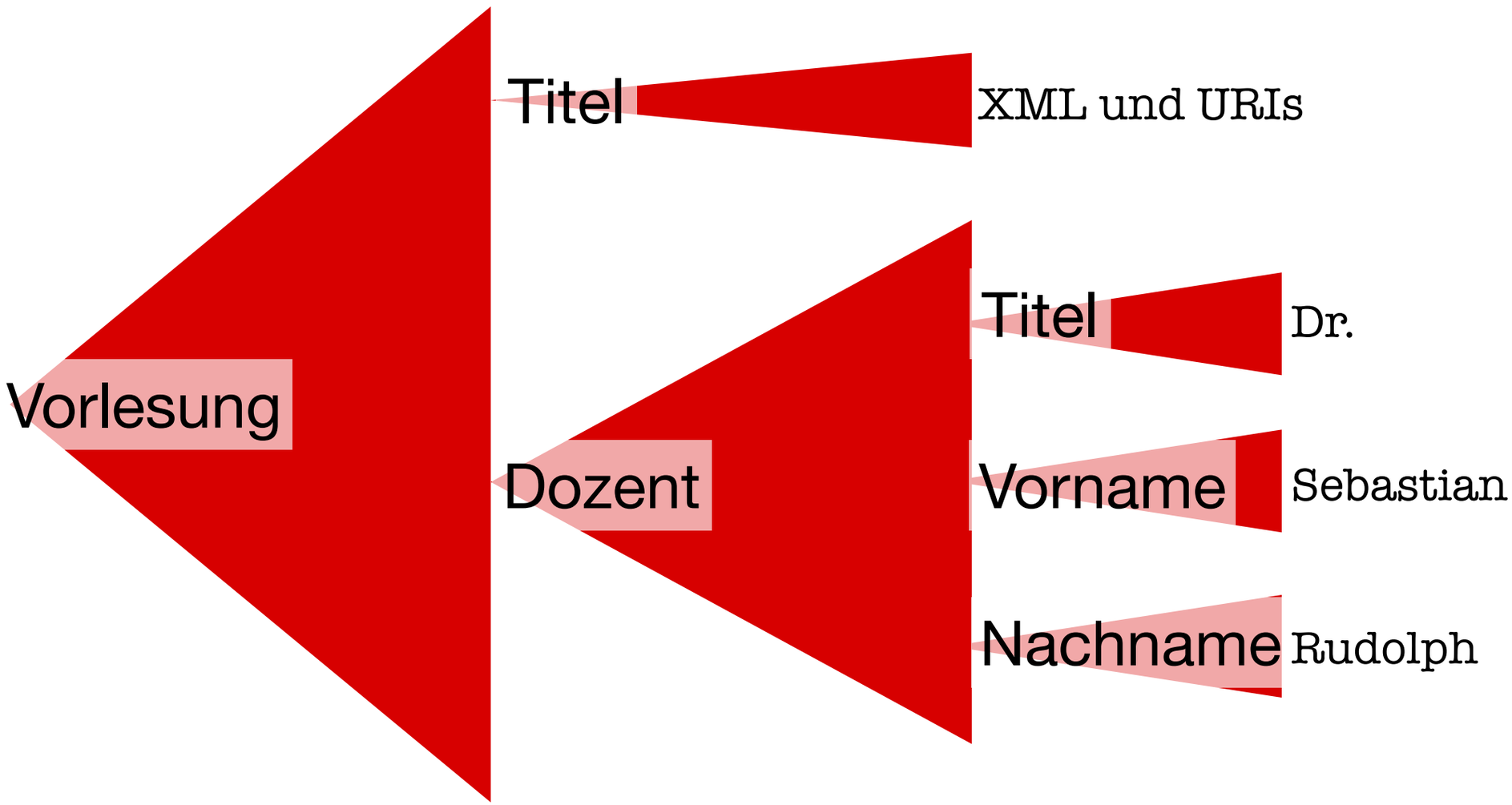
# Annotation mit Markupsprachen

AIFB



# Annotation mit Markupsprachen

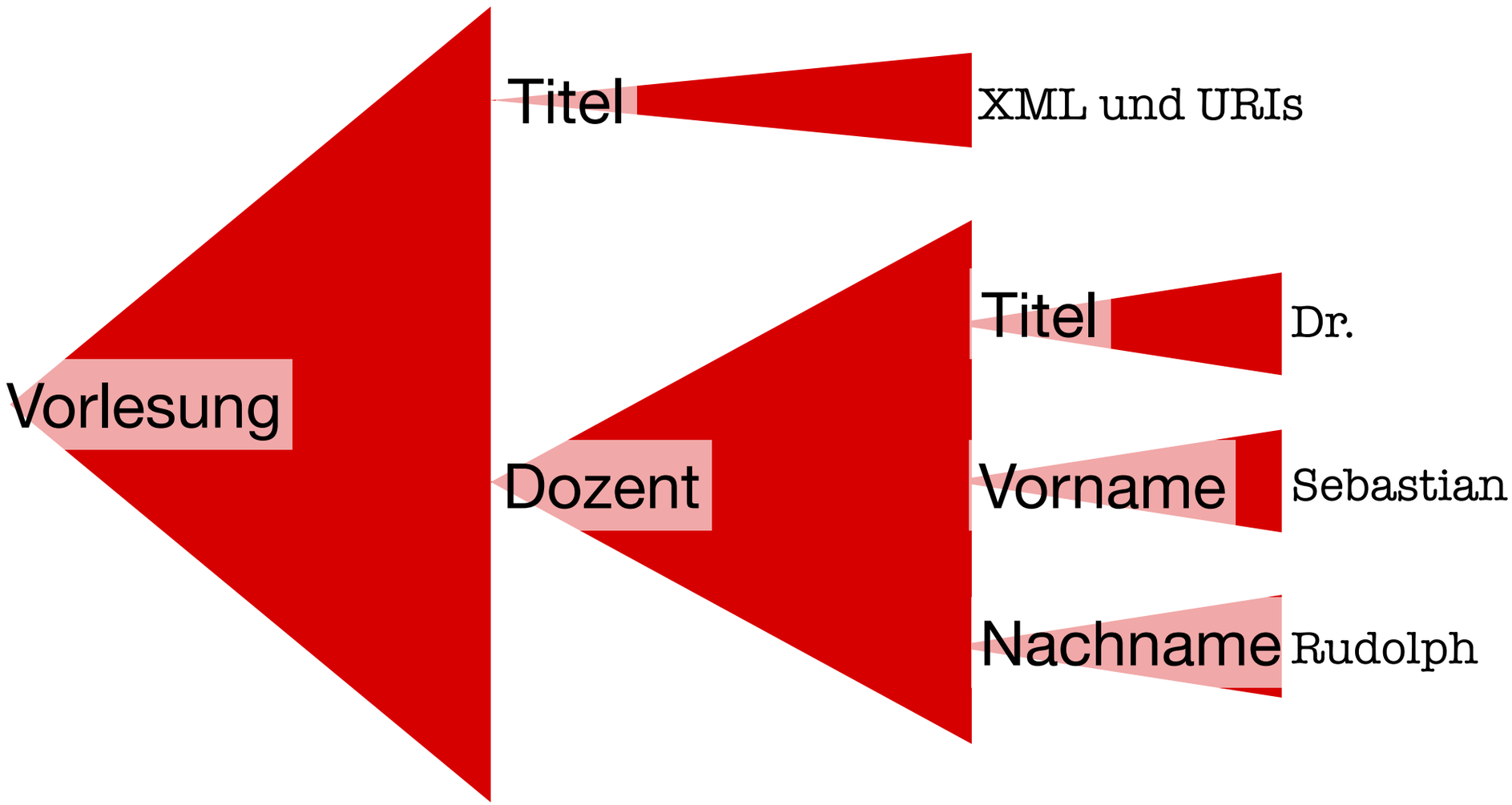
AIFB



# Annotation mit Markupsprachen

AIFB

## Baumstruktur



# Agenda



- XML - Motivation/Idee
- XML - Syntax
- URIs
- Namensräume

# XML



- eXtensible Markup Language
- Ursprung: strukturierter Text (HTML4.0 ∈ XML ⊂ SGML)
- Web-Standard (W3C) zum Datenaustausch:
  - Ein- und Ausgabedaten von Anwendungen können mittels XML beschrieben werden
  - Industrie muss sich nur noch auf standardisierte Beschreibung (= Vokabular) einigen
- Komplementärsprache zu HTML:
  - HTML beschreibt die Präsentation
  - XML beschreibt den Inhalt

# XML-Syntax (1) Präambel



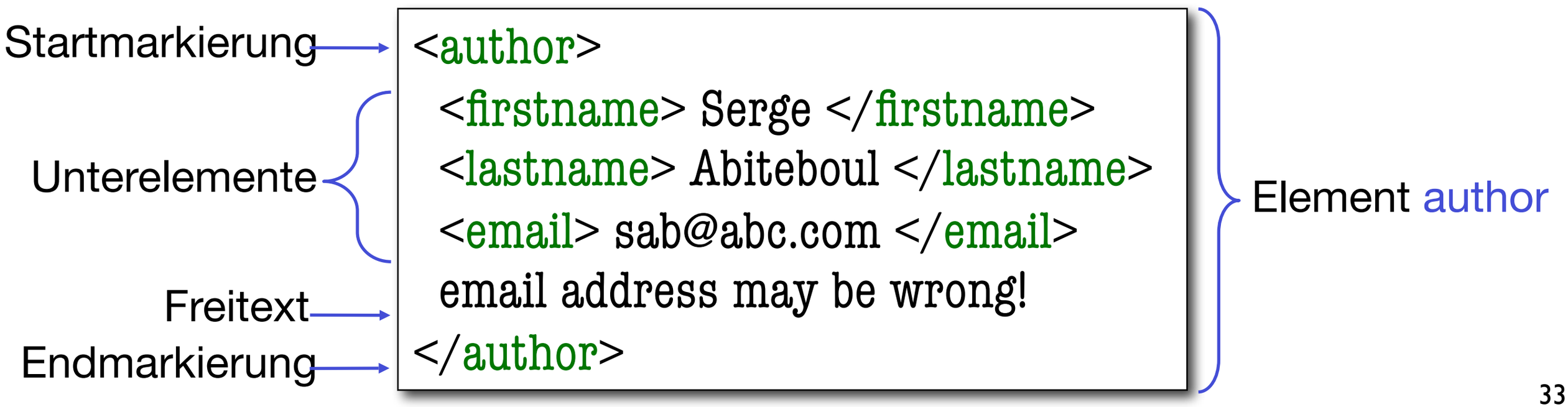
- XML-Dokument ist Textdokument
- beginnt mit Deklaration, die Versionsnummer des verwendeten Standards und optional die Zeichenkodierung enthält, z.B.:

```
<?xml version="1.0" encoding="utf-8"?>
```

# XML-Syntax (2) – XML-Element



- XML-Element (engl. element):
  - Beschreibung eines Objekts, die durch passende Markierungen (tags) geklammert ist
  - Inhalt eines Elements: Text und/oder weitere Elemente (beliebige Schachtelung möglich)
  - Leere Elemente: <year></year> kurz: <year/>
  - "äußerstes" Element heißt Wurzelelement (nur eines pro XML-Dokument)



# XML-Syntax (3) – XML-Attribut

AIFB

- XML-Attribut (engl. attribute):
  - Name-Zeichenkettenwert-Paar in Start- oder selbstschließendem Tag
  - Assoziiert mit einem Element
  - Alternative Möglichkeit, Daten zu beschreiben

Attribut email

```
<author email="sab@abc.com">
  <firstname> Serge </firstname>
  <lastname> Abiteboul </lastname>
</author>
```

# XML-Syntax (3) – XML-Attribut

AIFB

- XML-Attribut (engl. attribute):
  - Name-Zeichenkettenwert-Paar in Start- oder selbstschließendem Tag
  - Assoziiert mit einem Element
  - Alternative Möglichkeit, Daten zu beschreiben

Attribut email

```
<author email="sab@abc.com">
  <firstname> Serge </firstname>
  <lastname> Abiteboul </lastname>
</author>
```

Weitere denkbare Beschreibung derselben Daten:

```
<author firstname="Serge" lastname="Abiteboul" email="sab@abc.com"/>
```

# XML vs. HTML

AIFB

- HTML: festes Vokabular (Menge von tags) und Semantik (die Darstellung von Text)
- XML: freie Bezeichner zur Beschreibung von anwendungsspezifischer Syntax und Semantik
- XML⊂SGML

```
<h1> Bib </h1>
  <p>
    <i> Foundations of Databases </i>
     Serge Abiteboul
    < br > Addison Wesley, 1997
  <p>
    ...
```

**HTML**

```
<Bib id="o1">
  <paper id="o12">
    <title> Foundations of Databases </title>
    <author>
      <firstname> Serge </firstname>
      <lastname> Abiteboul </lastname>
    </author>
    <year> 1997 </year>
    <publisher> Addison Wesley </publisher>
  </paper>
  ...
</Bib>
```

**XML**

# Agenda



- XML - Motivation/Idee
- XML - Syntax
- URIs
- Namensräume

# URIs - Idee



- URI = Uniform Resource Identifier
- dienen zur weltweit eindeutigen Bezeichnung von Ressourcen
- Ressource kann jedes Objekt sein, was (im Kontext der gegebenen Anwendung) eine klare Identität besitzt (z.B. Bücher, Orte, Menschen, Verlage, Beziehungen zwischen diesen Dingen, abstrakte Konzepte usw.)
- in bestimmten Domänen ähnliches bereits realisiert: ISBN für Bücher

# URIs - Syntax



- Erweiterung des URL-Konzeptes; nicht jede URI bezeichnet aber ein Webdokument (umgekehrt wird als URI für Webdokumente häufig deren URL verwendet)
- Beginnt mit dem sogenannten URI-Schema das durch ":" vom nachfolgenden Teil getrennt ist (z.B.: `http`, `ftp`, `mailto`)
- häufig hierarchisch aufgebaut

# Selbstdefinierte URIs



- nötig, wenn für eine Ressource (noch) keine URI existiert (bzw. bekannt ist)
- Strategie zur Vermeidung von (ungewollten) Überschneidungen: Nutzung von `http`-URIs einer eigenen Webseite
- ermöglicht auch Ablegen einer Dokumentation zur URI an dieser Stelle

# Beschreibendes vs. Beschriebenes



- Trennung von URI für Ressource und deren Dokumentation durch URI-Referenzen (durch "#" angehängte Fragmente) oder content negotiation
- z.B.: als URI für Shakespeares "Othello"
  http://de.wikipedia.org/wiki/Othello
  nicht geeignet, besser
  http://de.wikipedia.org/wiki/Othello#URI

# Agenda



- XML - Motivation/Idee
- XML - Syntax
- URIs
- Namensräume

# XML-Namensräume: Motivation



- XML-Dokumente besitzen Element- und Attributnamen (“Markup Vocabulary”) mit allgemeiner Gültigkeit
- Eine XML-Anwendung basiert auf allgemeiner Interpretation dieser Namen
- Ein XML-Dokument soll Markup-Vokabular aus mehreren 'Dictionaries’ enthalten können. (Erinnerung: XML-Dokument muss keine DTD haben.)
- Namespaces zur Vermeidung von Namenskonflikten.

# XML-Namensräume



- XML Namespaces sind ähnlich zu Modul-Konzepten in Programmiersprachen
- Disambiguierung von Tag-Namen durch Verwendung unterschiedlicher “Prefixe”
- Ein Prefix wird vom lokalen Namen separiert durch ein “:”, so entstehen prefix:name Tags
- Namespace-Bindungen werden von manchen Werkzeugen ignoriert, sog. “flache Namespaces”

# Namensraum-Bindungen



- Prefixe werden belegt mit Namespace URIs, indem ein Attribut xmlns:prefix bei dem relevanten Element oder einem seiner Vorgängerelemente eingefügt wird: prefix:name1, …, prefix:namen
- Der Wert des xmlns:prefix-Attributes ist eine URI, welche (für XML Schemata) auf eine Beschreibung der Namespace Syntax verweisen kann aber nicht muss
- Ein Element kann Bindings nutzen für mehrere (unterschiedliche) Namespaces durch Verwendung separater Attribute xmlns:prefix1, …, xmlns:prefixm

# Beispiel: Ohne Namensräume



```
<Vorlesung>
    <Titel>
      XML und URIs
    </Titel>
    <Dozent>
      <Titel>
        Dr.
      </Titel>
      <Vorname>
        Sebastian
      </Vorname>
      <Nachname>
        Rudolph
      </Nachname>
    </Dozent>
</Vorlesung>
```

Titel ist mehrdeutig verwendeter Tagname

# Zwei verschiedene Namensräume

AIFB

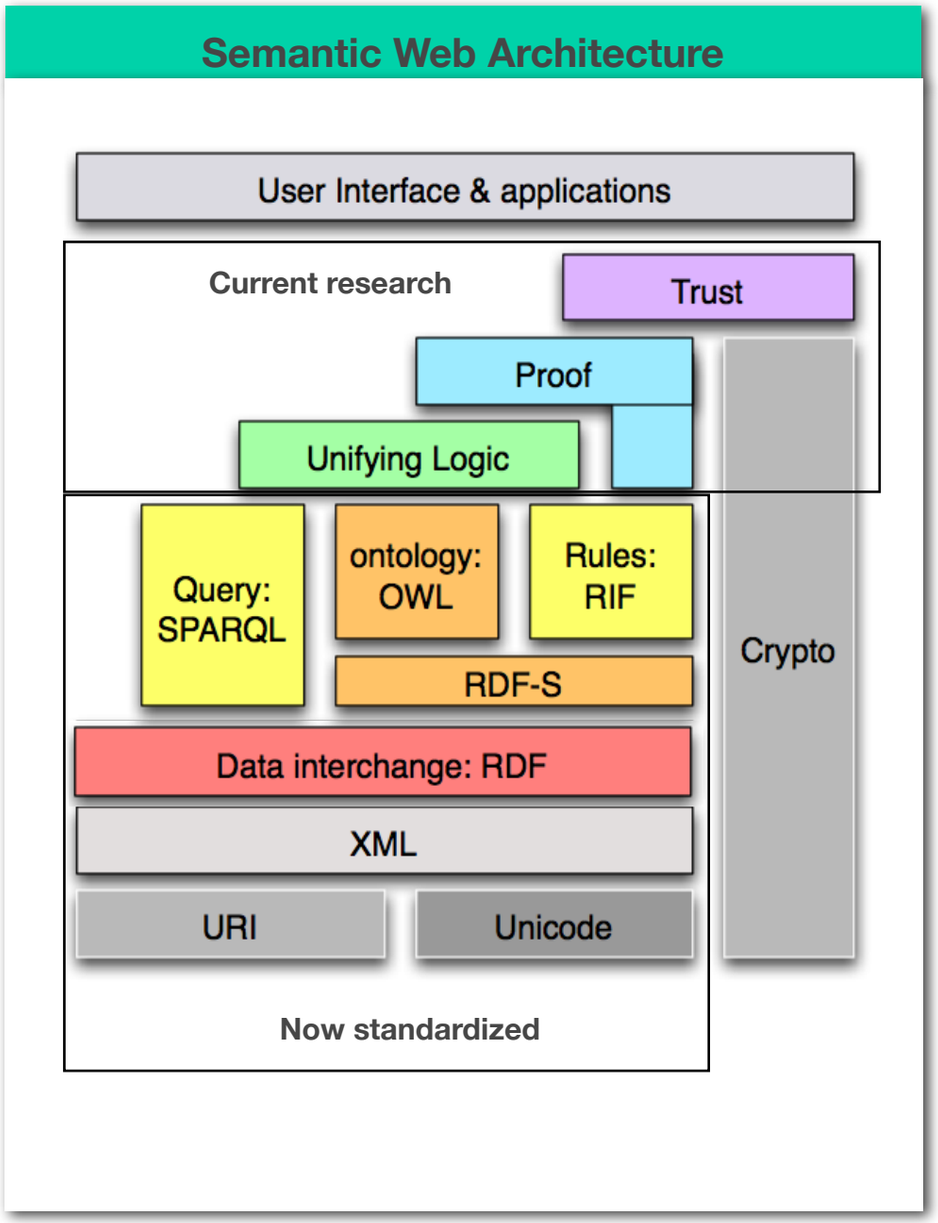
```
<Vorlesung xmlns:lv="http://www.semantic-web-Grundlagen/Lehrveranstaltungen"
           xmlns:person:="http://www.semantic-web-Grundlagen/Person" >
  < lv:Titel>
    XML und URIs
  </lv:Titel>
  < lv:Dozent>
    < person:Titel>
      Dr.
    </person:Titel>
    < person:Vorname>
      Sebastian
    </person:Vorname>
    < person:Nachname>
      Rudolph
    </person:Nachname>
  </lv:Dozent>
</lv:Vorlesung>
```

Titel wurde disambiguiert durch Verwendung der Prefixe lv und person

# Einführung in RDF

Dr. Sebastian Rudolph

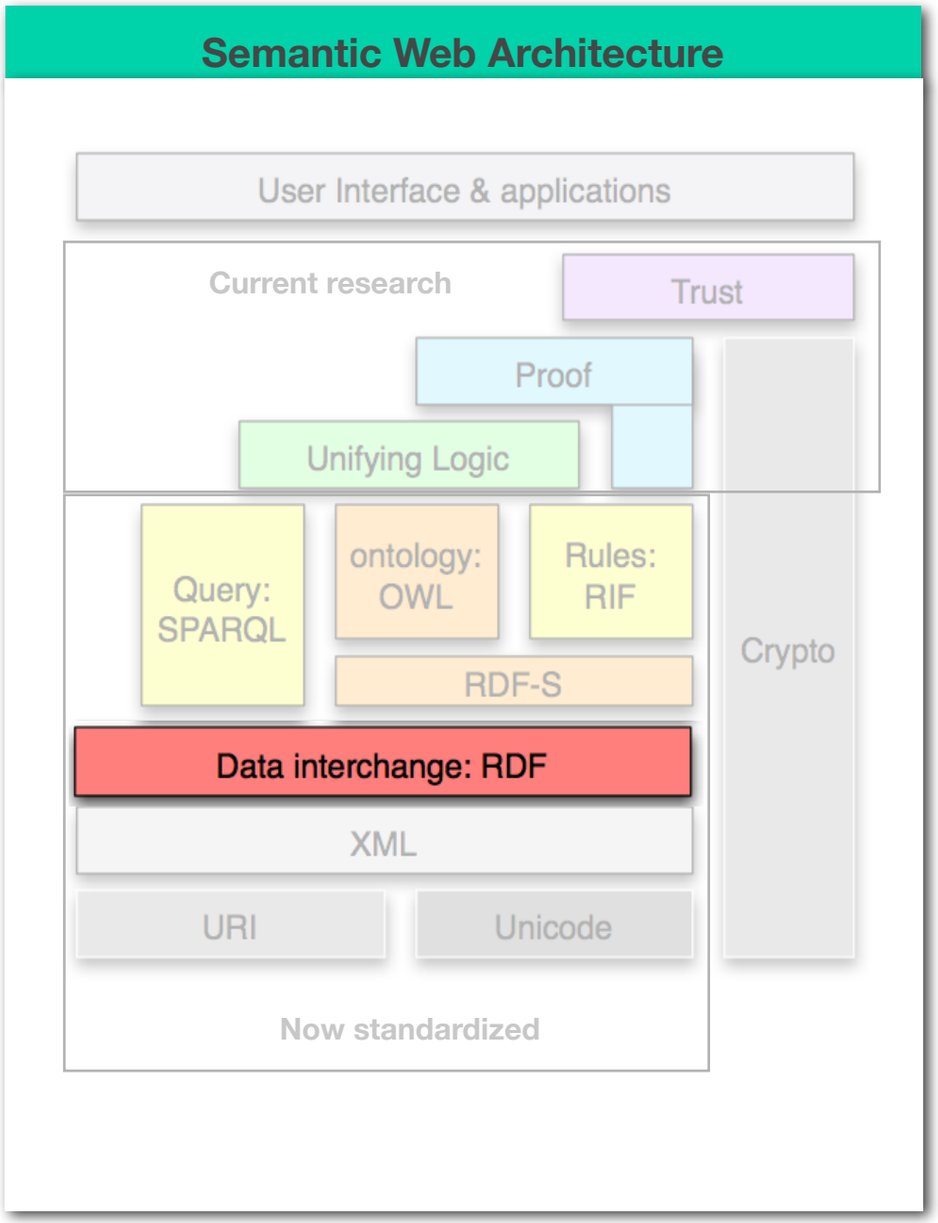




# Einführung in RDF

Dr. Sebastian Rudolph






entspricht dem ersten Teil von Kapitel 3 des Buches „Semantic Web - Grundlagen“ - siehe auch http://www.semantic-web-grundlagen.de/index.php/Kapitel_3

# Agenda



- Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML
- Datentypen
- mehrwertige Beziehungen
- leere Knoten
- Listen

# Agenda



- Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML
- Datentypen
- mehrwertige Beziehungen
- leere Knoten
- Listen

# Unzulänglichkeiten von XML



- Tag-Namen ambig (durch Namespaces und URIs behebbar)
- Baumstruktur nicht optimal für
  - intuitive Beschreibung der Daten
  - Informationsintegration
- Beispiel: wie kodiert man in einem Baum den Fakt:
  "Das Buch 'Semantic Web - Grundlagen' wird beim Springer-Verlag verlegt"?

# Modellierungsprobleme in XML

AIFB

- "Das Buch 'Semantic Web - Grundlagen' wird beim Springer-Verlag verlegt"

```
<Verlegt>
<Verlag>Springer-Verlag</Verlag>
<Buch>Semantic Web - Grundlagen</Buch>
</Verlegt>
```

```
<Verlag Name="Springer-Verlag">
<Verlegt Buch="Semantic Web - Grundlagen/>
</Verlag>
```

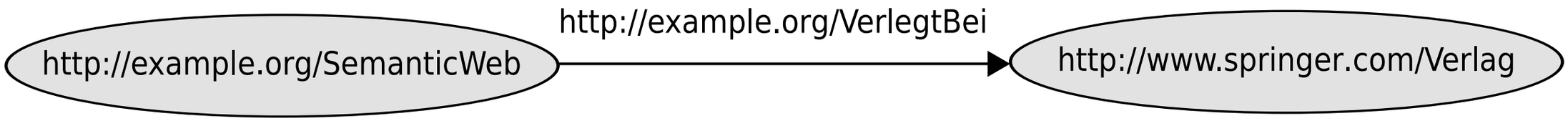
```
<Buch Name="Semantic Web - Grundlagen">
<Verleger Verlag="Springer-Verlag">
</Buch>
```

etc. ....

# RDF: Graphen statt Bäume



- Lösung: Darstellung durch (gerichtete Graphen)



# Agenda



- Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML

- Datentypen
- mehrwertige Beziehungen
- leere Knoten
- Listen

# Allgemeines zu RDF



- "Resource Description Framework"
- W3C Recommendation (http://www.w3.org/RDF)
- RDF ist ein Datenmodell
  - ursprünglich: zur Angabe von Metadaten für Web-Ressourcen, später allgemeiner
  - kodiert strukturierte Informationen
  - universelles, maschinenlesbares Austauschformat

# Bestandteile von RDF-Graphen

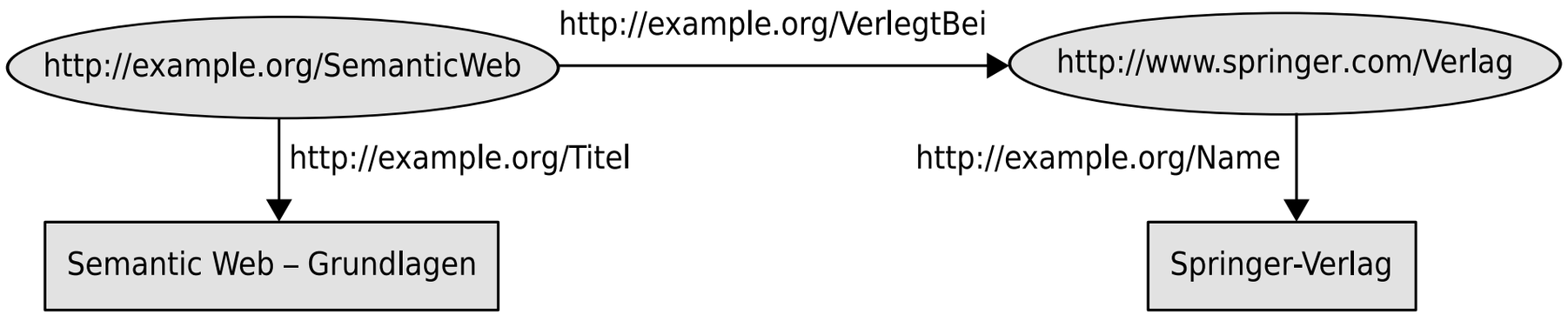


- URIs
  - zur eindeutigen Referenzierung von Ressourcen
  - bereits im Rahmen von XML behandelt
- Literale
  - beschreiben Datenwerte, denen keine separate Existenz zukommt
- leere Knoten
  - erlauben Existenzaussagen über ein Individuum mit gewissen Eigenschaften, ohne es zu benennen

# Literale



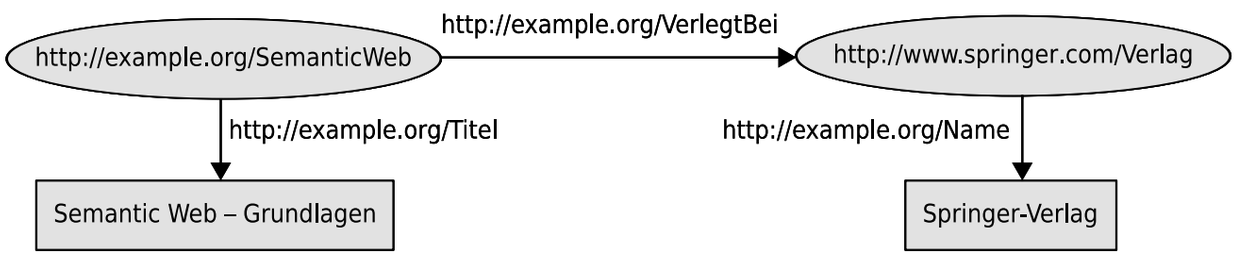
- zur Repräsentation von Datenwerten
- Darstellung als Zeichenketten
- Interpretation erfolgt durch *Datentyp*
- Literale ohne Datentyp werden wie Zeichenketten behandelt



# Graph als Menge von Tripeln



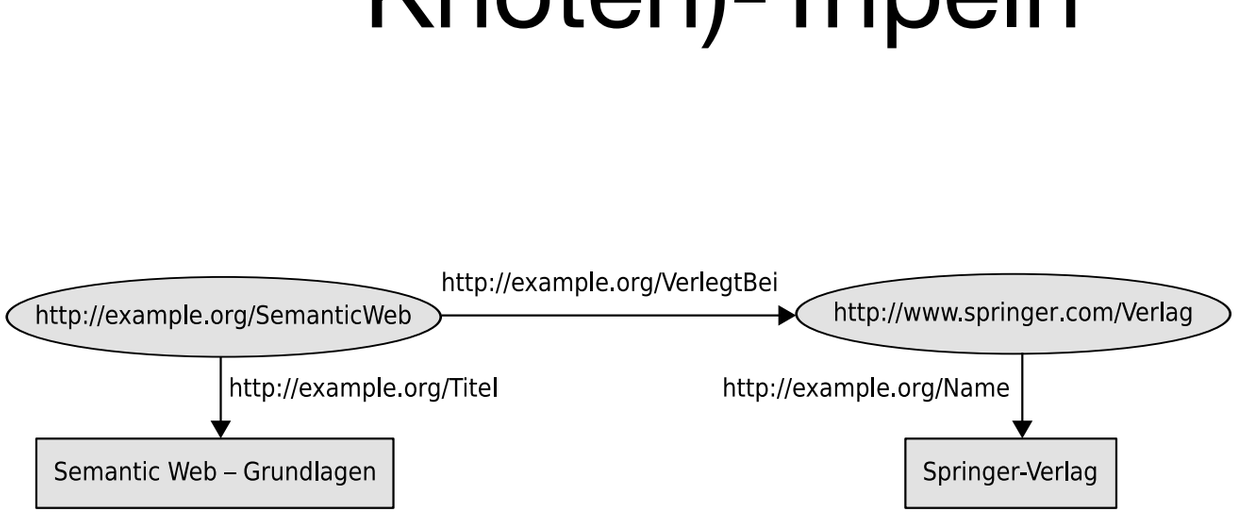
- verschiedene Darstellungsmöglichkeiten für Graphen
- hier verwendet: Liste von (Knoten-Kante-Knoten)-Tripeln



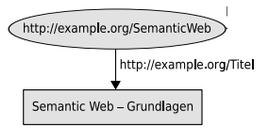
# Graph als Menge von Tripeln

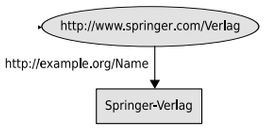


- verschiedene Darstellungsmöglichkeiten für Graphen
- hier verwendet: Liste von (Knoten-Kante-Knoten)-Tripeln







# RDF-Tripel

AIFB

- Bestandteile eines RDF-Tripels


- angelehnt an linguistische Kategorien, aber nicht immer stimmig

- erlaubte Belegungen:
  Subjekt : URI oder leerer Knoten
  Prädikat: URI (auch Propertys genannt)
  Objekt : URI oder leerer Knoten oder Literal

- Knoten- und Kantenbezeichner eindeutig, daher ursprünglicher Graph aus Tripel-Liste rekonstruierbar

# Agenda



- Motivation
- RDF-Datenmodell
- **Syntax für RDF: Turtle und XML**
- Datentypen
- mehrwertige Beziehungen
- leere Knoten
- Listen

# Einfache Syntax für RDF



- direkte Auflistung der Tripel:
  - N3: "Notation 3" - umfangreicher Formalismus
  - N-Triples: Teil von N3
  - Turtle: Erweiterung von N-Triples (Abkürzungen)

- Syntax in Turtle:
  - URIs in spitzen Klammern
  - Literale in Anführungszeichen
  - Tripel durch Punkt abgeschlossen
  - Leerzeichen und Zeilenumbrüche außerhalb von Bezeichenern werden ignoriert

# Turtle Syntax: Abkürzungen



- Beispiel

```
<http://example.org/SemanticWeb>
    <http://example.org/VerlegtBei>  <http://springer.com/Verlag> .
<http://example.org/SemanticWeb>
    <http://example.org/Titel>       "Semantic Web - Grundlagen" .
<http://springer.com/Verlag>
    <http://example.org/Name>        "Springer-Verlag" .
```

- auch in Turtle können Abkürzungen für Präfixe festgelegt werden:

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

ex:SemanticWeb   ex:VerlegtBei  springer:Verlag .
ex:SemanticWeb   ex:Titel       "Semantic Web - Grundlagen" .
springer:Verlag  ex:Name        "Springer-Verlag" .
```

# Turtle Syntax: Abkürzungen



- mehrere Tripel mit gleichem Subjekt kann man zusammenfassen:

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

ex:SemanticWeb   ex:VerlegtBei  springer:Verlag ;
                 ex:Titel       "Semantic Web - Grundlagen" .
springer:Verlag  ex:Name        "Springer-Verlag", "Springer" .
```

- ebenso Tripel mit gleichem Subjekt und Prädikat:

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

ex:SemanticWeb ex:Autor ex:Hitzler, ex:Krötzsch, ex:Rudolph, ex:Sure ;
               ex:Titel "Semantic Web - Grundlagen" .
```

# XML-Syntax von RDF



- Turtle intuitiv gut lesbar und maschinenverarbeitbar
- aber: bessere Tool-Unterstützung und Programmbibliotheken für XML
- daher: XML-Syntax am verbreitetsten

# XML-Syntax von RDF



- wie in XML werden Namensräume eingesetzt, um Tagnamen zu disambiguieren
- RDF-eigene tags haben einen festgelegten Namensraum, der Bezeichner ist standardmäßig ‘rdf’

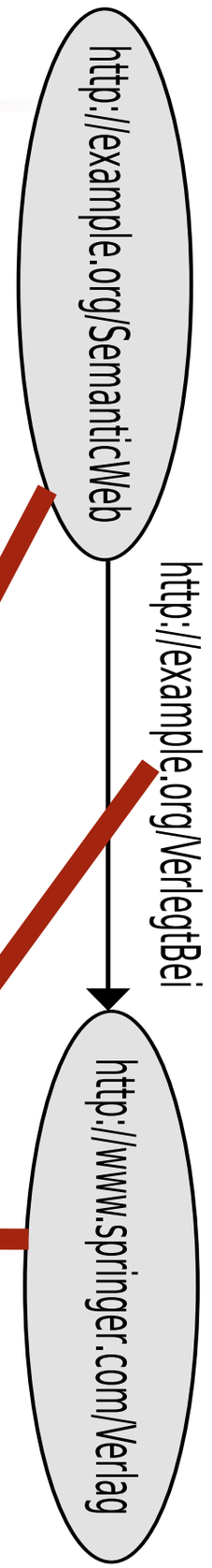
```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:ex ="http://example.org/">

  <rdf:Description rdf:about="http://example.org/SemanticWeb">
    <ex:VerlegtBei>
      <rdf:Description rdf:about="http://springer.com/Verlag">
      </rdf:Description>
    </ex:VerlegtBei>
  </rdf:Description>

</rdf:RDF>
```

# XML-Syntax von RDF

AIFB

- Das rdf:Description-Element kodiert das Subjekt (dessen URI wird als Wert des zugehörigen rdf:about-Attributs angegeben).
- Jedes geschachtelt im rdf:Description-Element enthaltene Element steht für ein Prädikat (dessen URI ist der Elementname), das wiederum das Tripel-Objekt als rdf:Description-Element enthält.

```
<rdf:Description rdf:about="http://example.org/SemanticWeb">
  <ex:VerlegtBei>
    <rdf:Description rdf:about="http://springer.com/Verlag">
    </rdf:Description>
  </ex:VerlegtBei>
</rdf:Description>
```

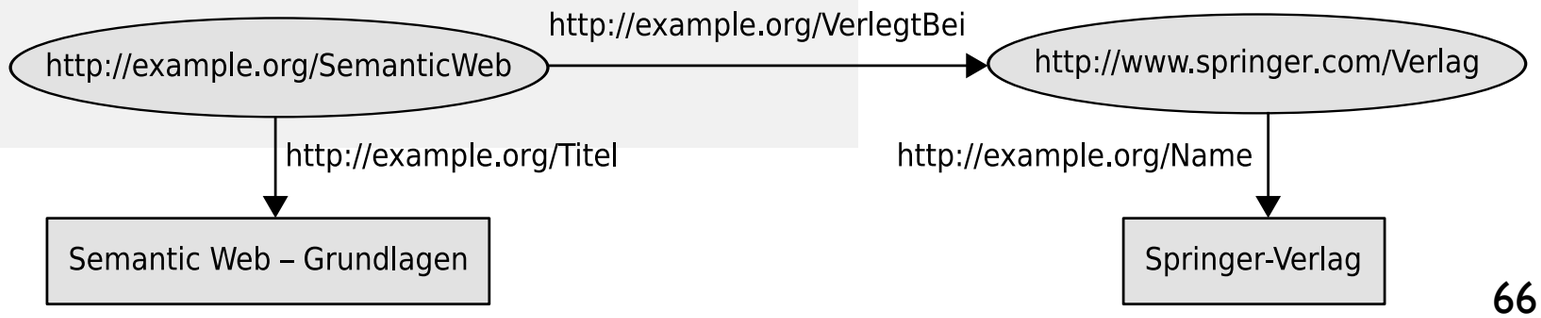


# XML-Syntax von RDF



- ungetypte Literale können als Freitext in das Prädikatelement eingeschlossen werden
- Verkürzte Darstellung erlaubt:
  - ein Subjekt enthält mehrere Property-Elemente
  - eine Objekt-Description dient als Subjekt für ein weiteres Tripel

```
<rdf:Description rdf:about="http://example.org/SemanticWeb">
  <ex:Titel>Semantic Web - Grundlagen</ex:Titel>
  <ex:VerlegtBei>
    <rdf:Description rdf:about="http://springer.com/Verlag">
      <ex:Name>Springer-Verlag</ex:Name>
    </rdf:Description>
  </ex:VerlegtBei>
</rdf:Description>
```

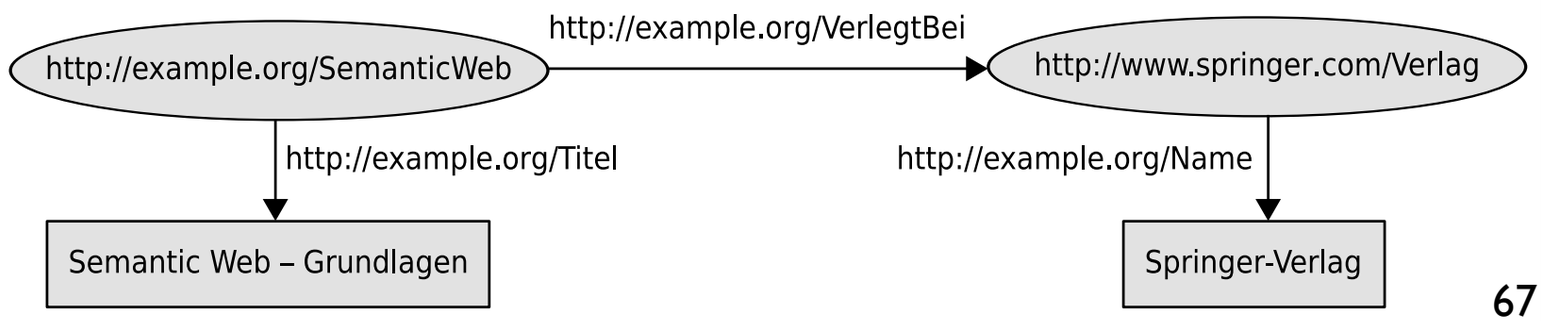


# XML-Syntax von RDF



- Alternative (aber semantisch gleichwertige) Darstellung für Literale als XML-Attribute
- Attributnamen sind dann die Property-URIs
- Angabe von Objekt-URIs als Wert des rdf:resource-Attributs innerhalb eines Property-Tags

```
<rdf:Description rdf:about="http://example.org/SemanticWeb"
                 ex:Titel= "Semantic Web - Grundlagen">
  <ex:VerlegtBei rdf:resource="http://springer.com/Verlag" />
</rdf:Description>
<rdf:Description rdf:about="http://springer.com/Verlag"
                 ex:Name="Springer-Verlag"                  />
```



# RDF/XML-Syntax: Komplikationen



- Namensräume sind essentiell (nicht nur Abkürzung), da in XML-Elementen und -Attributen keine Doppelpunkte zulässig, die keine Namensräume kodieren
- Problem: in XML keine Namensräume in Attributwerten möglich (würde im Sinne eines URI-Schemas interpretiert), also z.B. verboten:

```
rdf:about="ex:SemanticWeb"
```

- "Workaround" via XML-Entitäten:
  Deklaration:

```
<!ENTITY ex 'http://example.org/'>
```

  Verwendung:

```
rdf:resource="&ex;SemanticWeb"
```

# RDF/XML-Syntax: Basis-URIs

AIFB

- Arbeit mit Basis-URIs:

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xml:base="http://www.example.org/" >

  <rdf:Description rdf:about="SemanticWeb">
    <ex:VerlegtBei rdf:resource="http://springer.com/Verlag" />
  </rdf:Description>

</rdf:RDF>
```

- Erkennung relativer URIs an Abwesenheit eines Schemateils

# Agenda



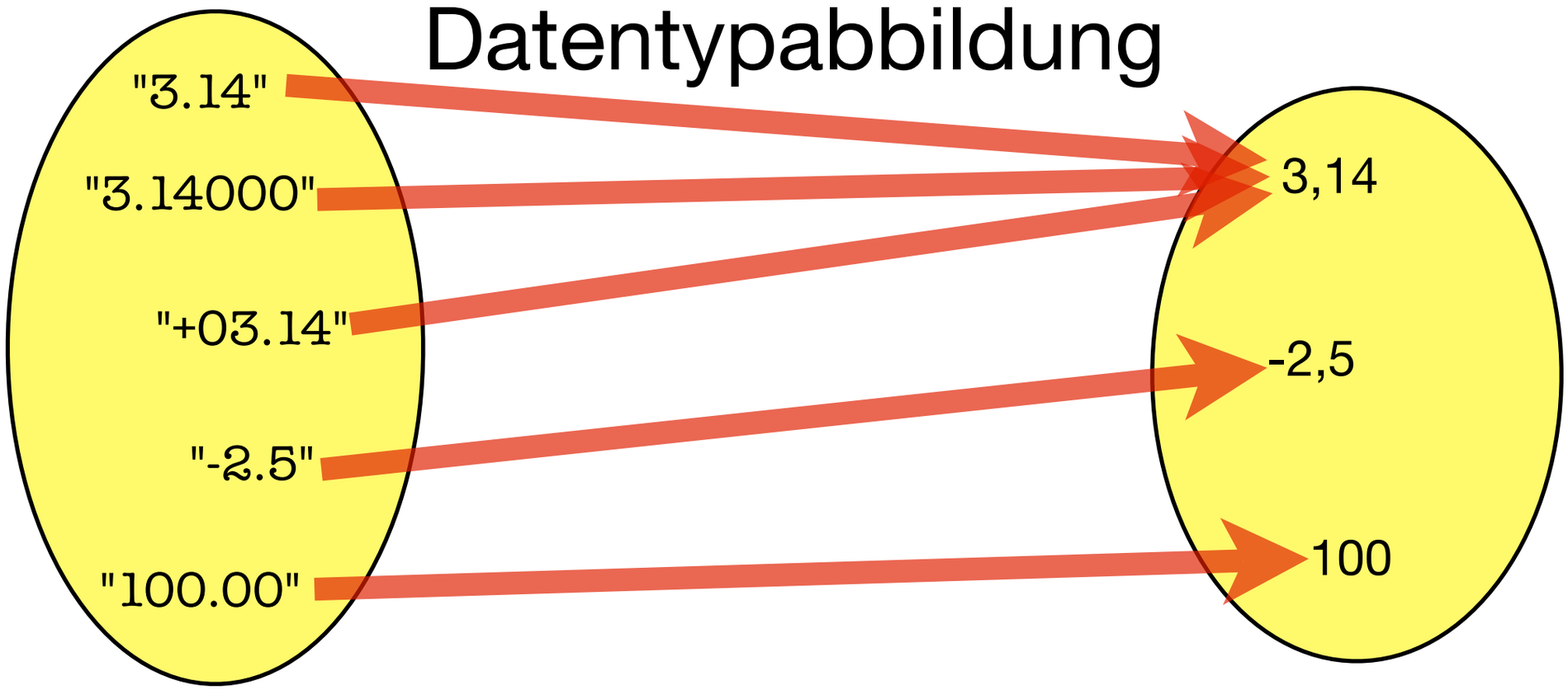
- Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML
- Datentypen
- mehrwertige Beziehungen
- leere Knoten
- Listen

# Datentypen – Abstrakt

AIFB

- Beispiel: xsd:decimal

lexikalischer Bereich Wertebereich




- bzgl. xsd:decimal gilt "3.14"="+03.14"
  bzgl. xsd:string nicht!

# Datentypen in RDF

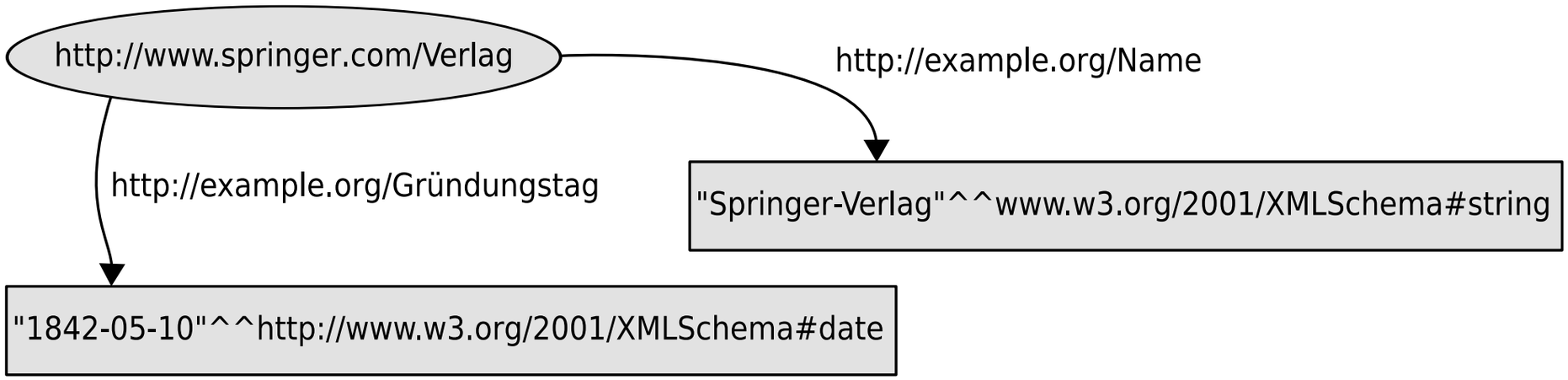


- Bisher: Literale ungetypt, wie Zeichenketten behandelt (also z.B.: "02"<"100"<"11"<"2")
- Typung erlaubt besseren (semantischen = bedeutungsgemäßen) Umgang mit Werten
- Datentypen werden durch URIs identifiziert und sind im Prinzip frei wählbar
- häufig: Verwendung von xsd-Datentypen
- Syntax:
  "*Datenwert*"^^*Datentyp-URI*

# Datentypen in RDF – Beispiel

AIFB

- Graph:




- Turtle:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
<http://springer.com/Verlag>
        <http://example.org/Name>  "Springer-Verlag"^^xsd:string ;
        <http://example.org/Gründungstag> "1842-05-10"^^xsd:date .
```

- XML:

```
<rdf:Description rdf:about="http://springer.com/Verlag">
  <ex:Name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
    Springer-Verlag
  </ex:Name>
  <ex:Gründungstag
          rdf:datatype="http://www.w3.org/2001/XMLSchema#date">
    1842-05-10
  </ex:Gründungstag>
</rdf:Description>
```

# Der vordefinierte Datentyp

- rdf:XMLLiteral ist einziger vordefinierter Datentyp in RDF
- bezeichnet beliebige (balancierte) XML-Fragmente
- in RDF/XML besondere Syntax zur eindeutigen Darstellung:

```
<rdf:Description rdf:about="http://example.org/SemanticWeb">
   <ex:Titel rdf:parseType="Literal">
      <b>Semantic Web</b>
      <br />
      Grundlagen
   </ex:Titel>
</rdf:Description>
```

# Sprachangaben und Datentypen



- Sprachinformationen beeinflussen nur ungetypte Literale
- Beispiel:
  - XML

```
<rdf:Description rdf:about="http://springer.com/Verlag">
   <ex:Name xml:lang="de">Springer-Verlag</ex:Name>
   <ex:Name xml:lang="en">Springer Science+Business Media</ex:Name>
</rdf:Description>
```

  - Turtle

```
<http://springer.com/Verlag> <http://example.org/Name>
   "Springer-Verlag"@de, "Springer Science+Business Media"@en .
```

# Sprachangaben und Datentypen



- nach RDF-Spezifikation sind demnach die folgenden Literale unterschiedlich:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
<http://springer.com/Verlag>     <http://example.org/Name>
                        "Springer-Verlag",
                        "Springer-Verlag"@de,
                        "Springer-Verlag"^^xsd:string .
```

- ...werden aber häufig (intuitionsgemäß) als gleich implementiert.

# Agenda



- Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML
- Datentypen
- mehrwertige Beziehungen
- leere Knoten
- Listen

# Mehrwertige Beziehungen



- Kochen mit RDF:
  "*Für die Zubereitung von Chutney benötigt man 450g grüne Mango, einen Teelöffel Cayennepfeffer, ...*"

- erster Modellierungsversuch:

```
@prefix ex: <http://example.org/> .
ex:Chutney  ex:hatZutat   "450g grüne Mango", "1TL Cayennepfeffer"
```

- nicht zufriedenstellend: Zutaten samt Menge als Zeichenkette. Suche nach Rezepten, die grüne Mango beinhalten, so nicht möglich.

# Mehrwertige Beziehungen



- Kochen mit RDF:
  "*Für die Zubereitung von Chutney benötigt man 450g grüne Mango, einen Teelöffel Cayennepfeffer, ...*"

- zweiter Modellierungsversuch:

```
@prefix ex: <http://example.org/> .
ex:Chutney  ex:Zutat   ex:grüneMango;      ex:Menge   "450g" ;
            ex:Zutat   ex:Cayennepfeffer;  ex:Menge   "1TL" .
```

- überhaupt nicht zufriedenstellend: keine eindeutige Zuordnung von konkreter Zutat und Menge mehr möglich.

# Mehrwertige Beziehungen



- Problem: es handelt sich um eine echte dreiwertige (auch: ternäre) Beziehung (s. z.B. Datenbanken)

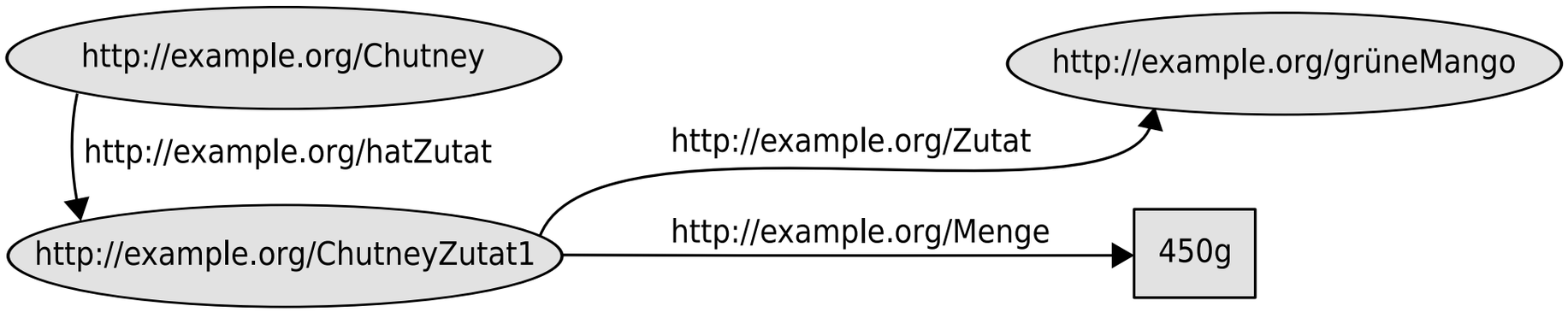
| Gericht | Zutat | Menge |
|---|---|---|
| Chutney | grüne Mango | 450g |
| Chutney | Cayennepfeffer | 1 TL |

- direkte Darstellung in RDF nicht möglich
- Lösung: Einführung von Hilfsknoten

# Mehrwertige Beziehungen

AIFB

- Hilfsknoten in RDF:
  - als Graph




  - Turtle-Syntax (mit Verwendung von rdf:value)

```
@prefix ex:  <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
ex:Chutney          ex:hatZutat   ex:ChutneyZutat1 .
ex:ChutneyZutat1    rdf:value     ex:grüneMango;
                    ex:Menge      "450g" .
```

# Agenda

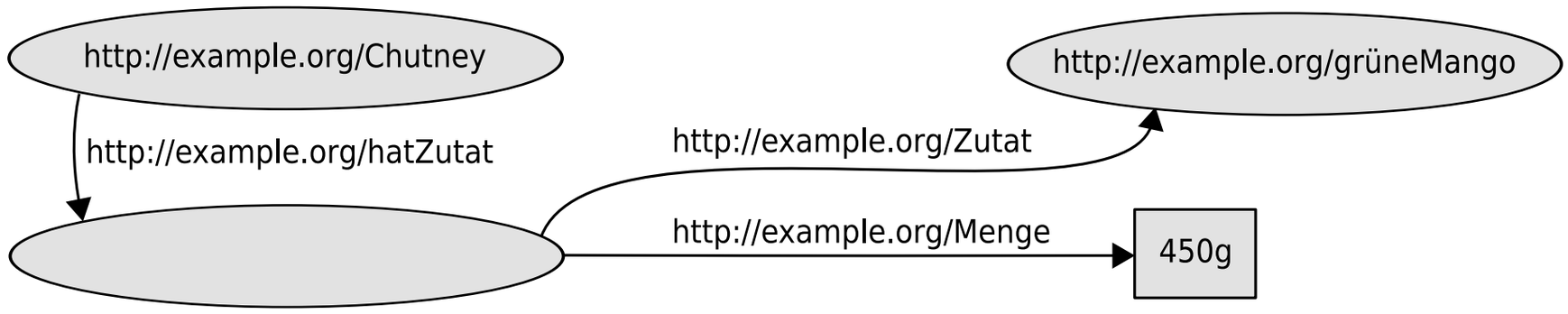


- Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML
- Datentypen
- mehrwertige Beziehungen
- leere Knoten
- Listen

# Leere Knoten



- leere Knoten (blank nodes, bnodes) können für Ressourcen verwendet werden, die nicht benannt werden müssen (z.B. Hilfsknoten)
- können als Existenzaussagen gelesen werden
- Syntax:
  - als Graph



# Leere Knoten



- Syntax:
  - RDF/XML-Syntax

```
<rdf:Description rdf:about="http://example.org/Chutney">
   <ex:hatZutat rdf:nodeID="id1" />
</rdf:Description>
<rdf:Description rdf:nodeID="id1">
   <ex:Zutat rdf:resource="http://example.org/grüneMango" />
   <ex:Menge>450g</ex:Menge>
</rdf:Description>
```

  - verkürzt

```
<rdf:Description rdf:about="http://example.org/Chutney">
   <ex:hatZutat rdf:parseType="Resource">
     <ex:Zutat rdf:resource="http://example.org/grüneMango" />
     <ex:Menge>450g</ex:Menge>
   </ex:hatZutat>
</rdf:Description>
```

# Leere Knoten



- Syntax:
  - Turtle

```
@prefix ex:   <http://example.org/> .
ex:Chutney     ex:hatZutat   _:id1 .
_:id1          ex:Zutat      ex:grüneMango;   ex:Menge    "450g" .
```

  - verkürzt

```
@prefix ex:   <http://example.org/> .
ex:Chutney     ex:hatZutat
              [ ex:Zutat      ex:grüneMango;   ex:Menge   "450g" ] .
```

# Agenda



- Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML
- Datentypen
- mehrwertige Beziehungen
- leere Knoten
- Listen

# Listen

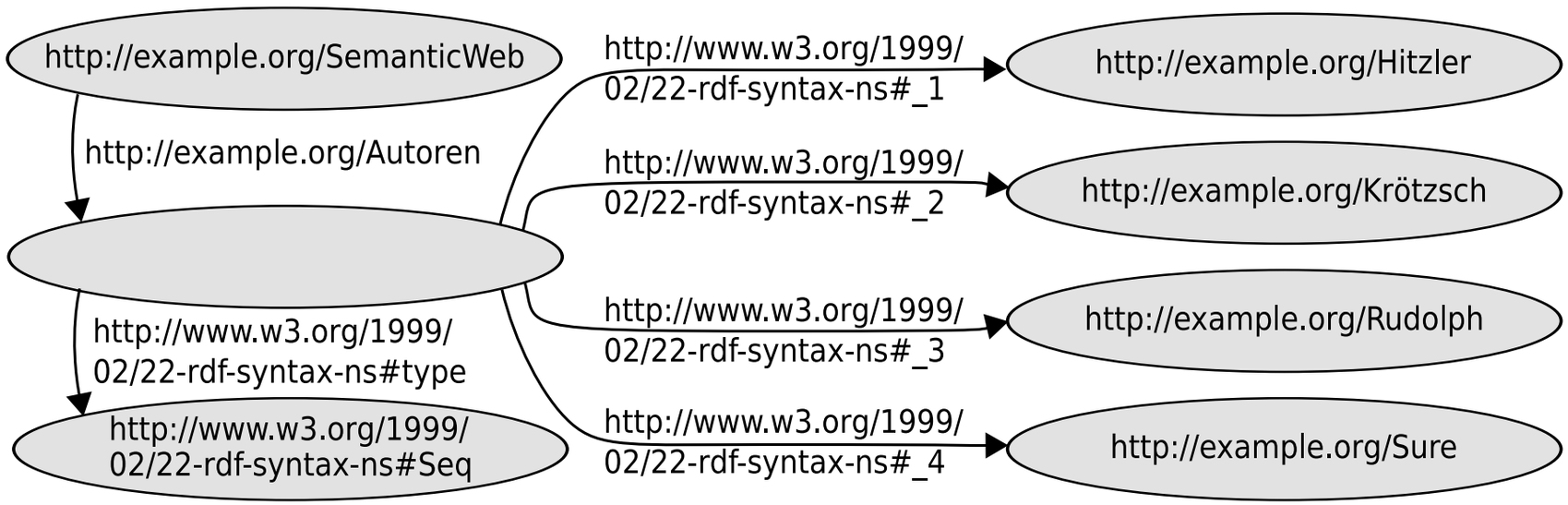


- allgemeine Datenstrukturen zur Aufzählung von beliebig vielen Ressourcen (Reihenfolge relevant), z.B. Autoren eines Buches
- Unterscheidung zwischen
  - offenen Listen (Container)
    Hinzufügen von neuen Einträgen möglich
  - geschlossenen Listen (Collections)
    Hinzufügen von  neuen Einträgen nicht möglich
- Können auch mit bereits vorgestellten Ausdrucksmitteln modelliert werden, also keine zusätzliche Ausdrucksstärke!

# Offene Listen (Container)



- Graph:




- verkürzt in RDF/XML:

```
<rdf:Description rdf:about="http://example.org/SemanticWeb">
   <ex:Autoren>
      <rdf:Seq>
         <rdf:li rdf:resource="http://www.example.org/Hitzler" />
         <rdf:li rdf:resource="http://www.example.org/Krötzsch" />
         <rdf:li rdf:resource="http://www.example.org/Rudolph" />
         <rdf:li rdf:resource="http://www.example.org/Sure" />
      </rdf:Seq>
   </ex:Autoren>
</rdf:Description>
```

# Typen offener Listen

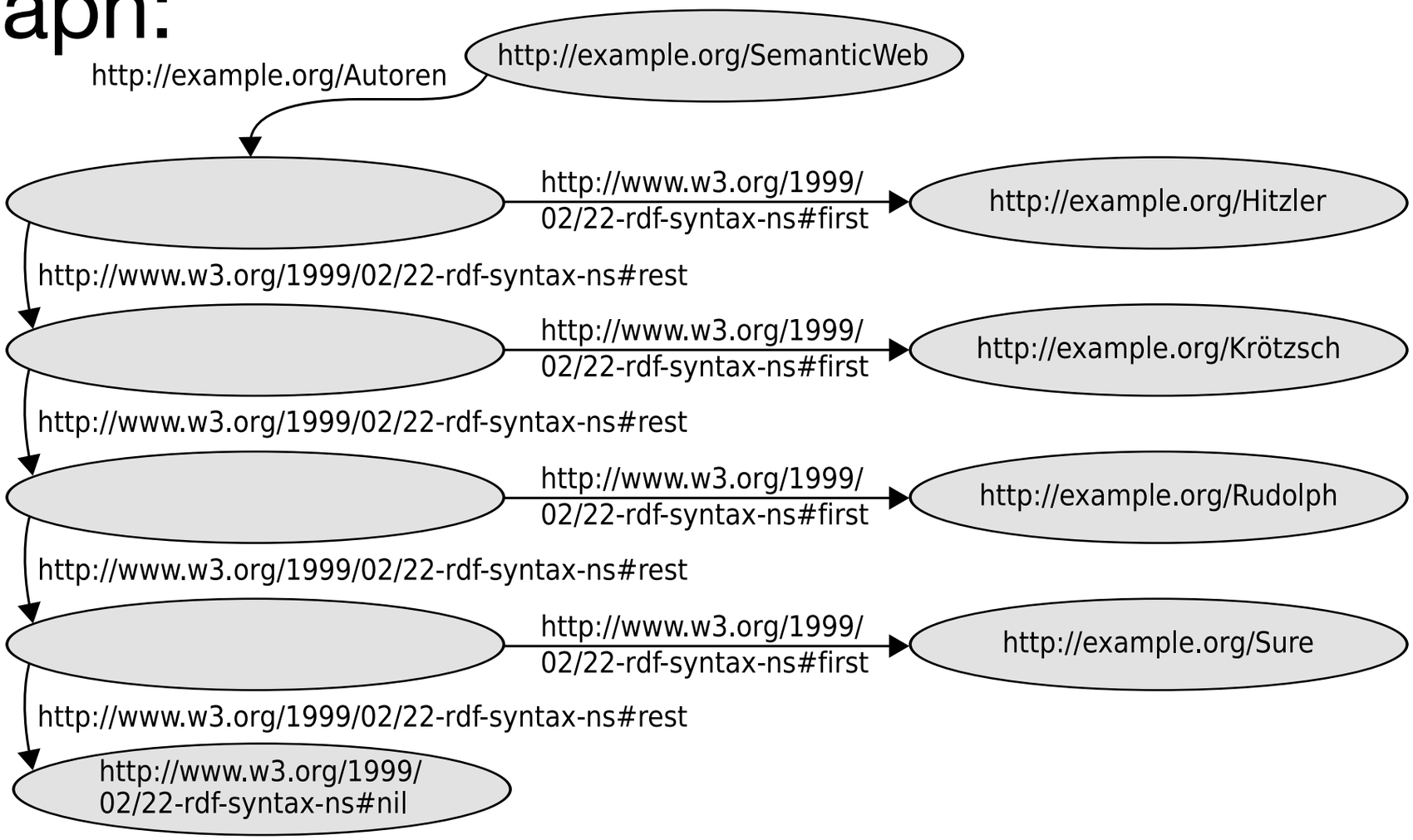


- via rdf:type wird dem Listen-Wurzelknoten ein Listentyp zugewiesen:
  - rdf:Seq
    Interpretation als geordnete Liste (Sequenz)
  - rdf:Bag
    Interpretation als ungeordnete Menge
    in RDF kodierte Reihenfolge nicht von Belang
  - rdf:Alt
    Menge alternativer Möglichkeiten
    im Regelfall immer nur ein Listeneintrag relevant

# Geschlossene Listen (Collections)



- Graph:




- Idee: rekursive Zerlegung der Liste in Kopfelement und (möglicherweise leere) Restliste.

# Geschlossene Listen (Collections)



- RDF/XML-Syntax

```
<rdf:Description rdf:about="http://example.org/SemanticWeb">
   <ex:Autoren rdf:parseType="Collection">
      <rdf:Description rdf:about="http://www.example.org/Hitzler">
      <rdf:Description rdf:about="http://www.example.org/Krötzsch" />
      <rdf:Description rdf:about="http://www.example.org/Rudolph" />
      <rdf:Description rdf:about="http://www.example.org/Sure" />
   </ex:Autoren>
</rdf:Description>
```

- Turtle

```
@prefix ex:  <http://example.org/> .
ex:SemanticWeb   ex:Autor
                ( ex:Hitzler   ex:Krötzsch   ex:Rudolph   ex:Sure ) .
```

# Verbreitungsgrad von RDF



- heute existiert Vielzahl von RDF-Tools
- Programmier-Bibliotheken für praktisch jede Programmiersprache
- frei verfügbare Systeme zum Umgang mit großen RDF-Datenmengen (sogenannte RDF Stores oder Triple Stores)
- auch kommerzielle Anbieter (z.B. Oracle) unterstützen zunehmend RDF
- Grundlage für Datenformate: RSS 1.0, XMP (Adobe), SVG (Vektorgrafikformat)

# Bewertung von RDF



- weitläufig unterstützter Standard für Speicherung und Austausch von Daten
- ermöglicht weitgehend syntaxunabhängige Darstellung verteilter Informationen in graphbasiertem Datenmodell
- reines RDF sehr "individuenorientiert"
- kaum Möglichkeiten zur Kodierung von Schemawissen

  ➜ RDF Schema (nächste Vorlesung)

LINKINGOPENDATA
W3C SWEO Community Project

AIFB

- Community-Projekt mit Unterstützung des W3C, begonnen 2007
- Idee: Veröffentlichung vorhandener offener Datensätze im Web im RDF-Format und wechselseitige Verknüpfung

LINKINGOPENDATA
W3C SWEO Community Project

AIFB

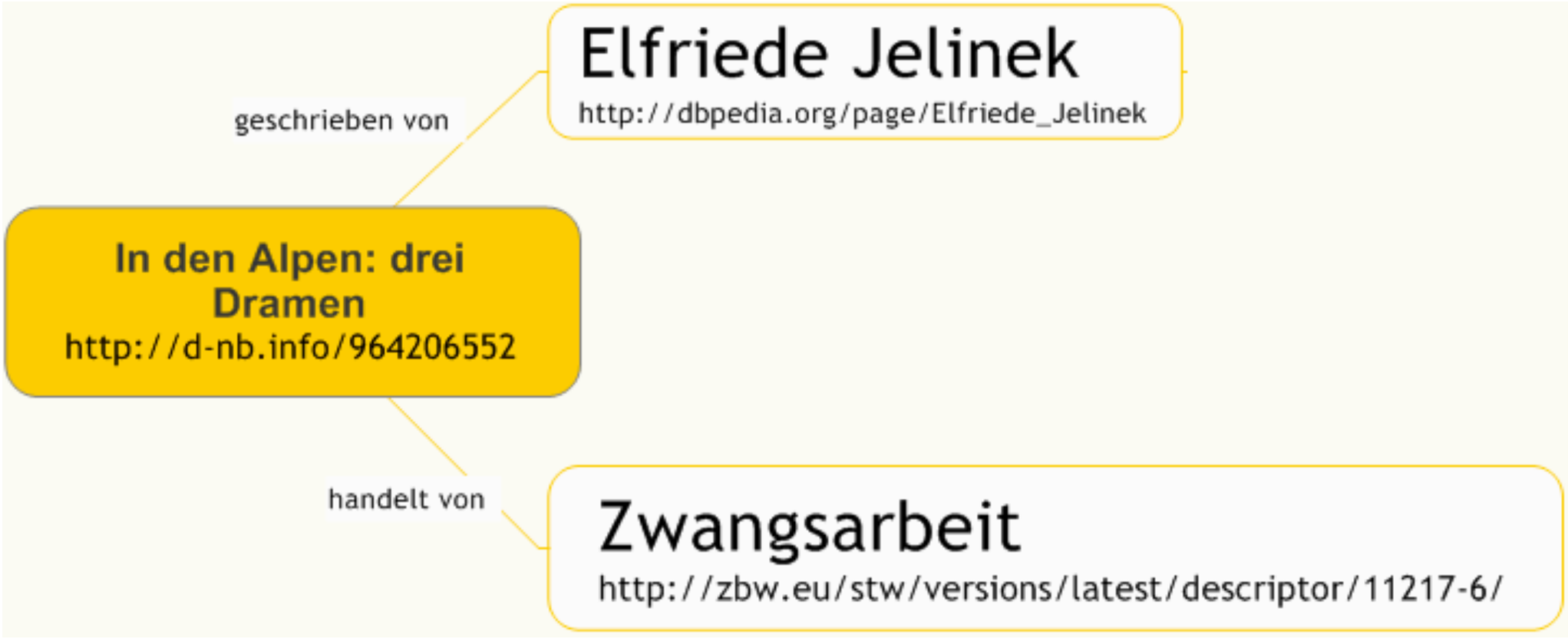
# Grundprinzipien

- konsequente Verwendung von URIs als Namen für Dinge
  - nicht nur Dokumente/Informationsquellen sondern alle Entitäten (Personen, Orte, ...)
- Verwendung von HTTP-konformen URIs
  - global eindeutige Bezeichner, verteilte Kontrolle
  - ermöglicht „Nachschlagen“ der Bezeichner im Web
- Bereitstellung von Informationen in RDF
  - wenn eine URI nachgeschlagen wird
- Verlinkung zu URIs anderer Datenquellen
  - ermöglicht Zugriff auf relevante zusätzliche Information

LINKINGOPENDATA
W3C SWEO Community Project

AIFB

1. konsequente Verwendung von URIs als Namen für Dinge




2. Verwendung von HTTP-konformen URIs

LINKINGOPENDATA
W3C SWEO Community Project

AIFB

# 2. Verwendung von HTTP-konformen URIs

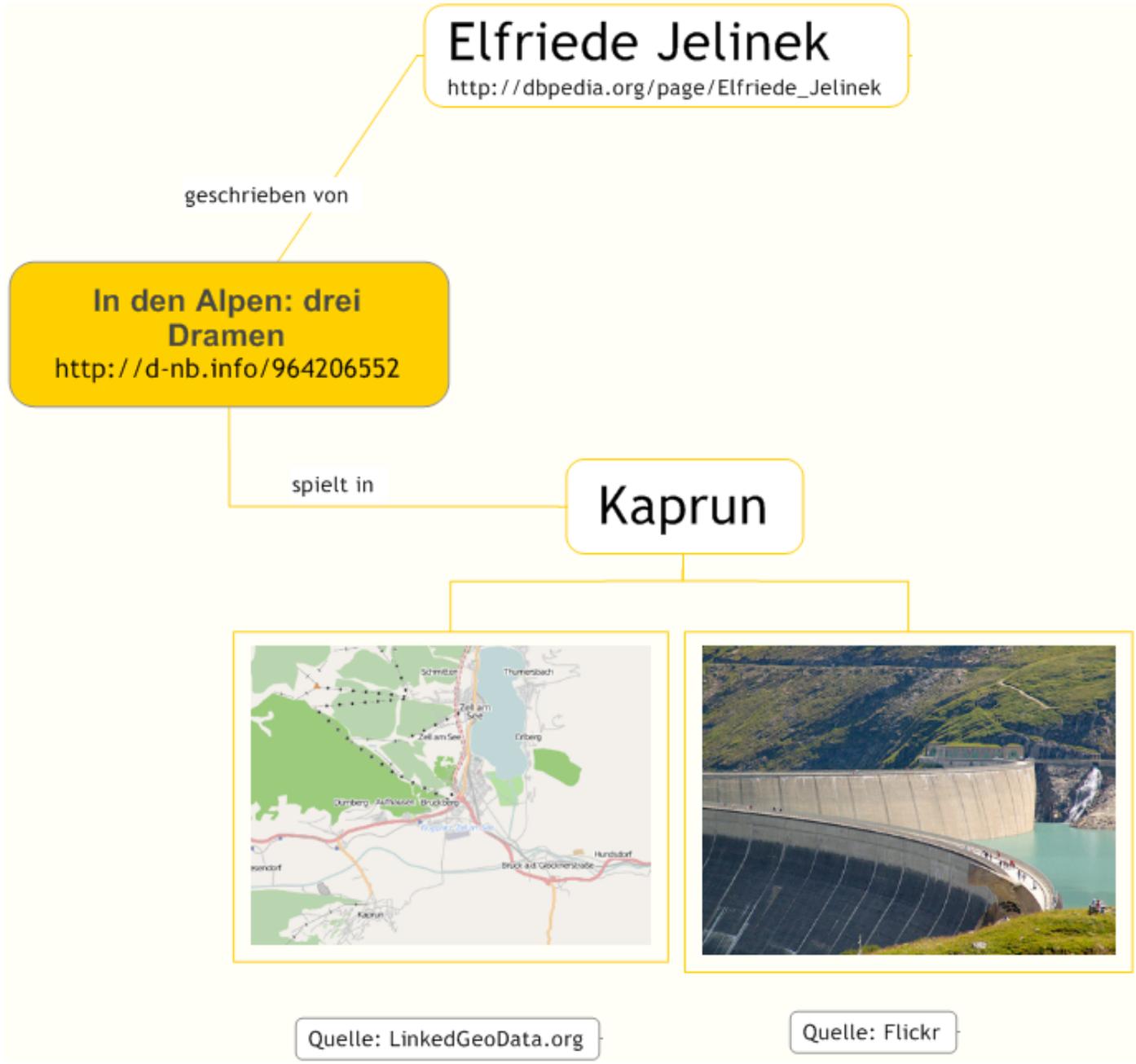

# 3. Bereitstellung von Informationen in RDF

LINKINGOPENDATA
W3C SWEO Community Project

AIFB

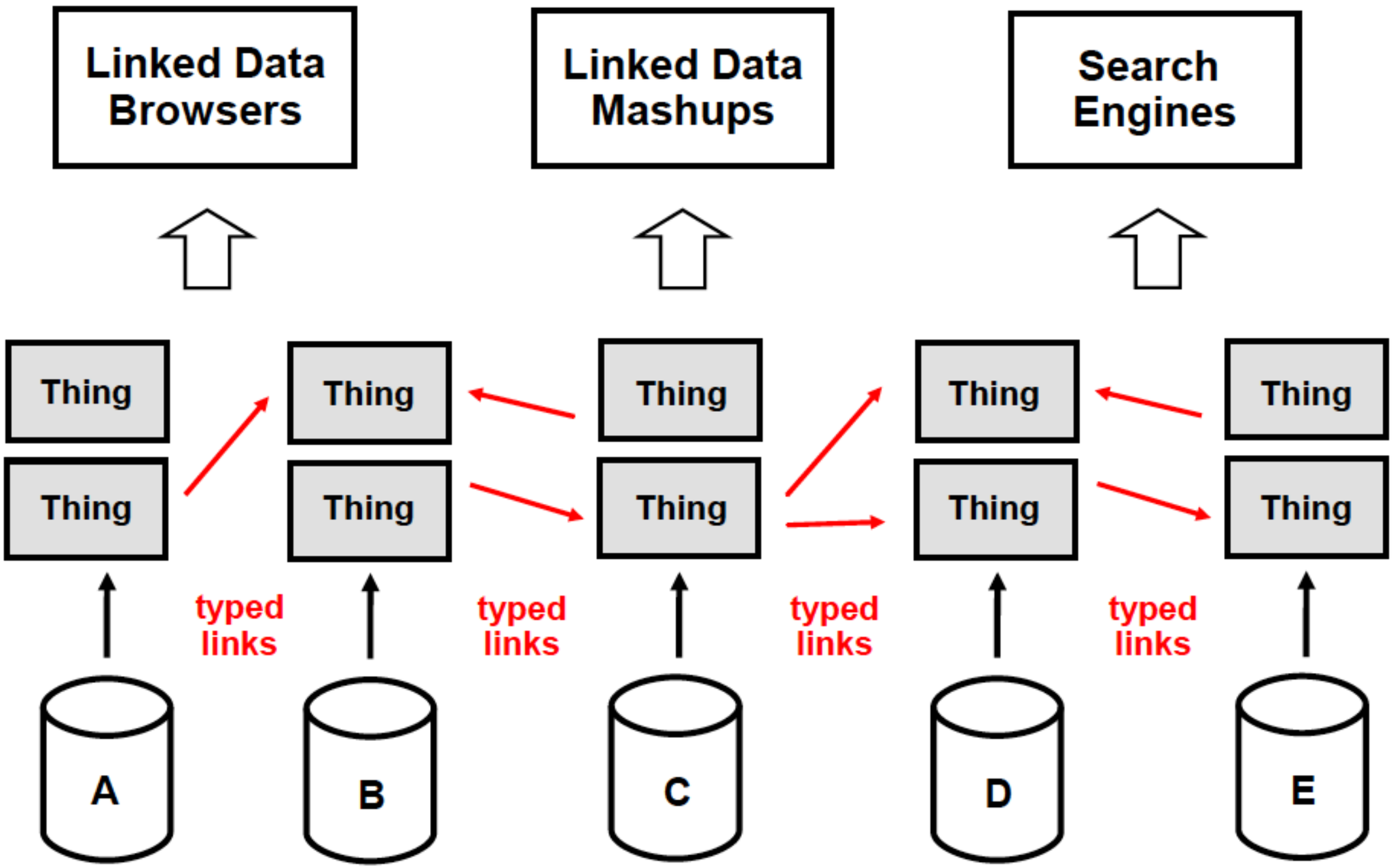
# 3. Bereitstellung von Informationen in RDF

# 4. Verlinkung zu URIs anderer Datenquellen



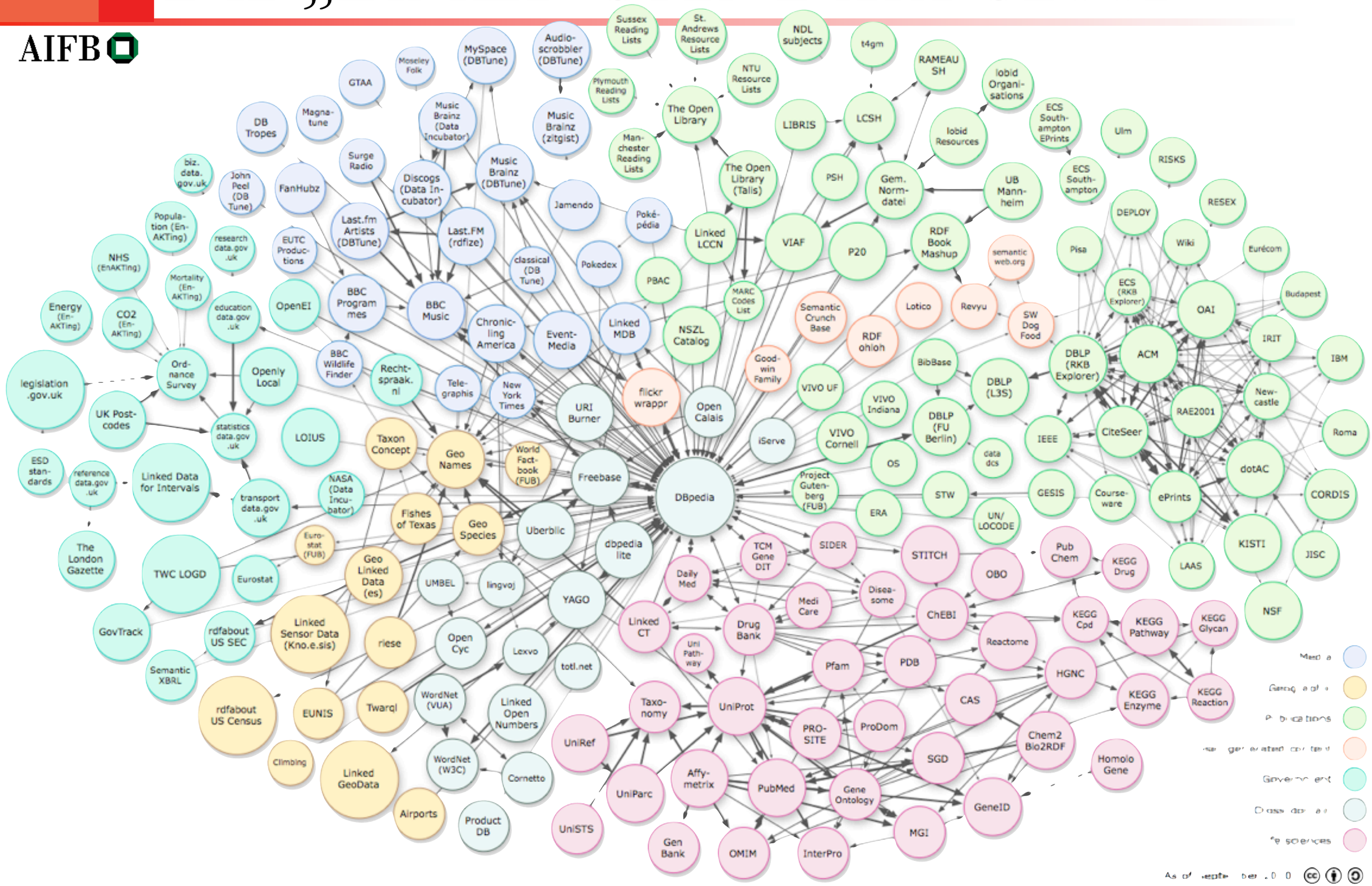
LINKINGOPENDATA
W3C SWEO Community Project

AIFB



# Die „Linked Open Data Cloud“

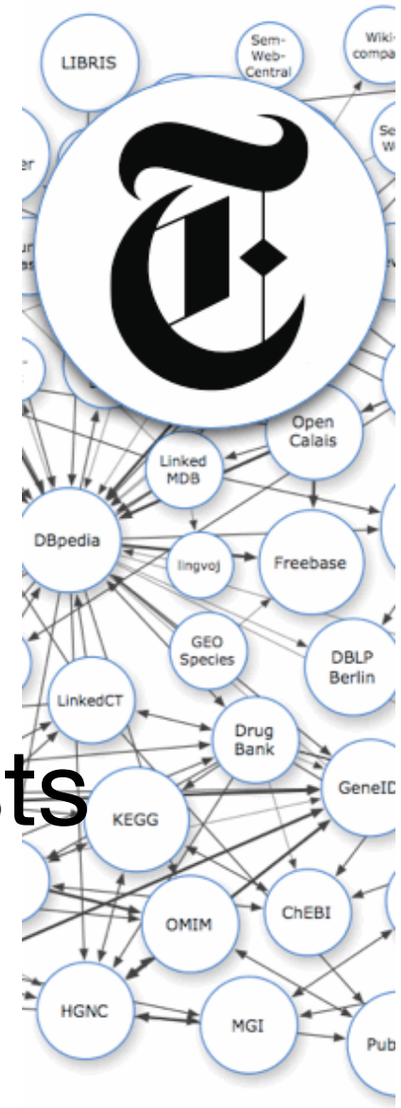
AIFB



# Showcases

AIFB

- OpenCalais www.opencalais.com
- New York Times data.nytimes.com
- BBC Music www.bbc.co.uk/music/artists




CALAIS
Powered by Thomson Reuters

BBC Music BETA

# Allgemeine Vorteile durch Semantic Web / Linked Data



- **Discovery** – das Finden/Entdecken relevanter Information durch intelligente Suche und Navigation
- **Consumption** – Erleichterung der Weiterverarbeitung/-verwendung von Information durch standardkonforme Datenhaltung
- **Quality** – Verringerung der Redundanz, effizientere Pflege der Datenbestände durch Spezialisierung
- **Meshup** – erhöhter Nutzen durch automatische Kombination von Information aus unterschiedlichen Quellen
- **Distributed Query** – Stellen von datenquellenübergreifenden Anfragen
- **Serendipity** – unvorhergesehenen Nutzungsszenarien durch Wechselwirkungen von Daten und Dienstleistungen

# Grundlagen Semantic Web

Lehrveranstaltung im WS11/12
Seminar für Computerlinguistik
Universität Heidelberg

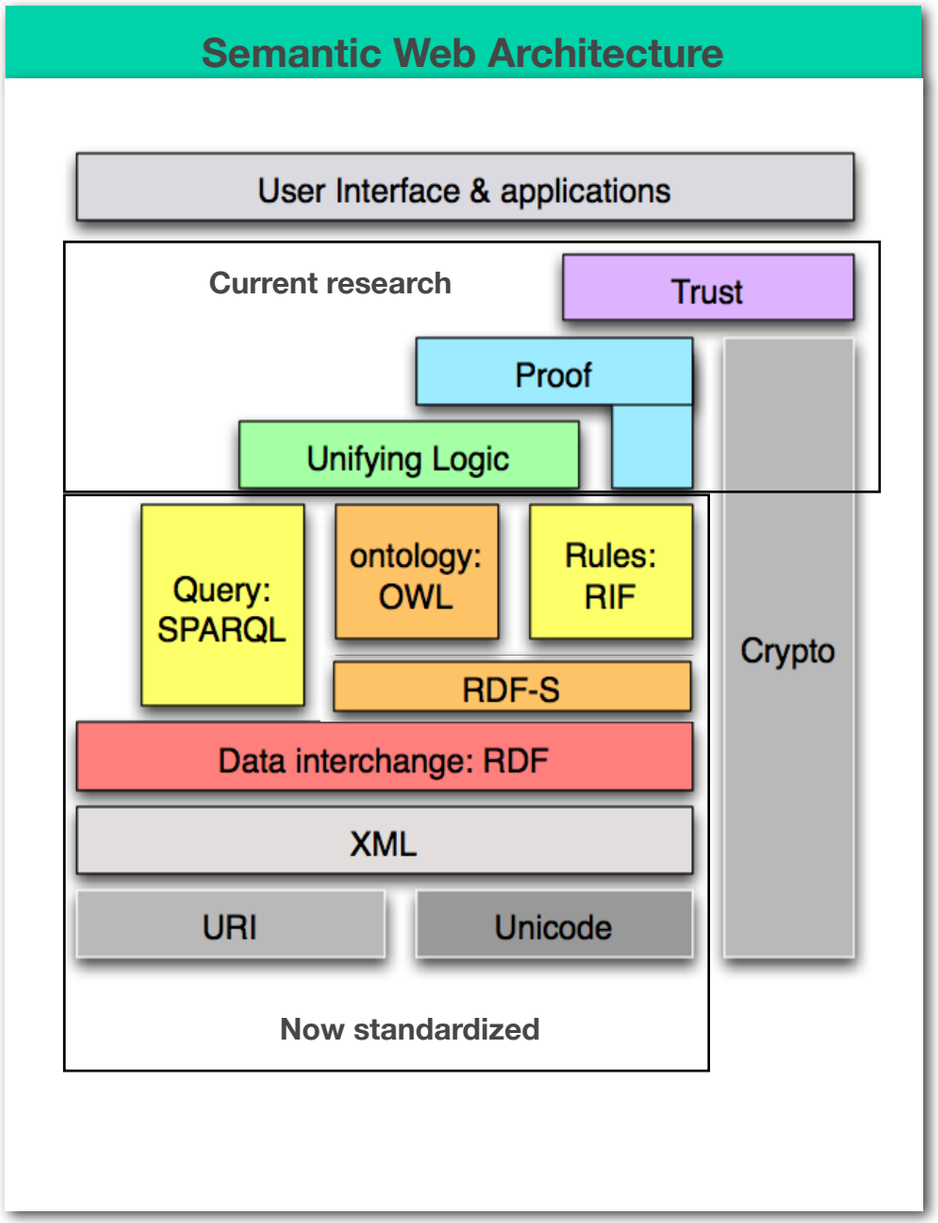
PD Dr. Sebastian Rudolph
Institut AIFB
Karlsruher Institut für Technologie

AIFB

# RDF Schema

Dr. Sebastian Rudolph

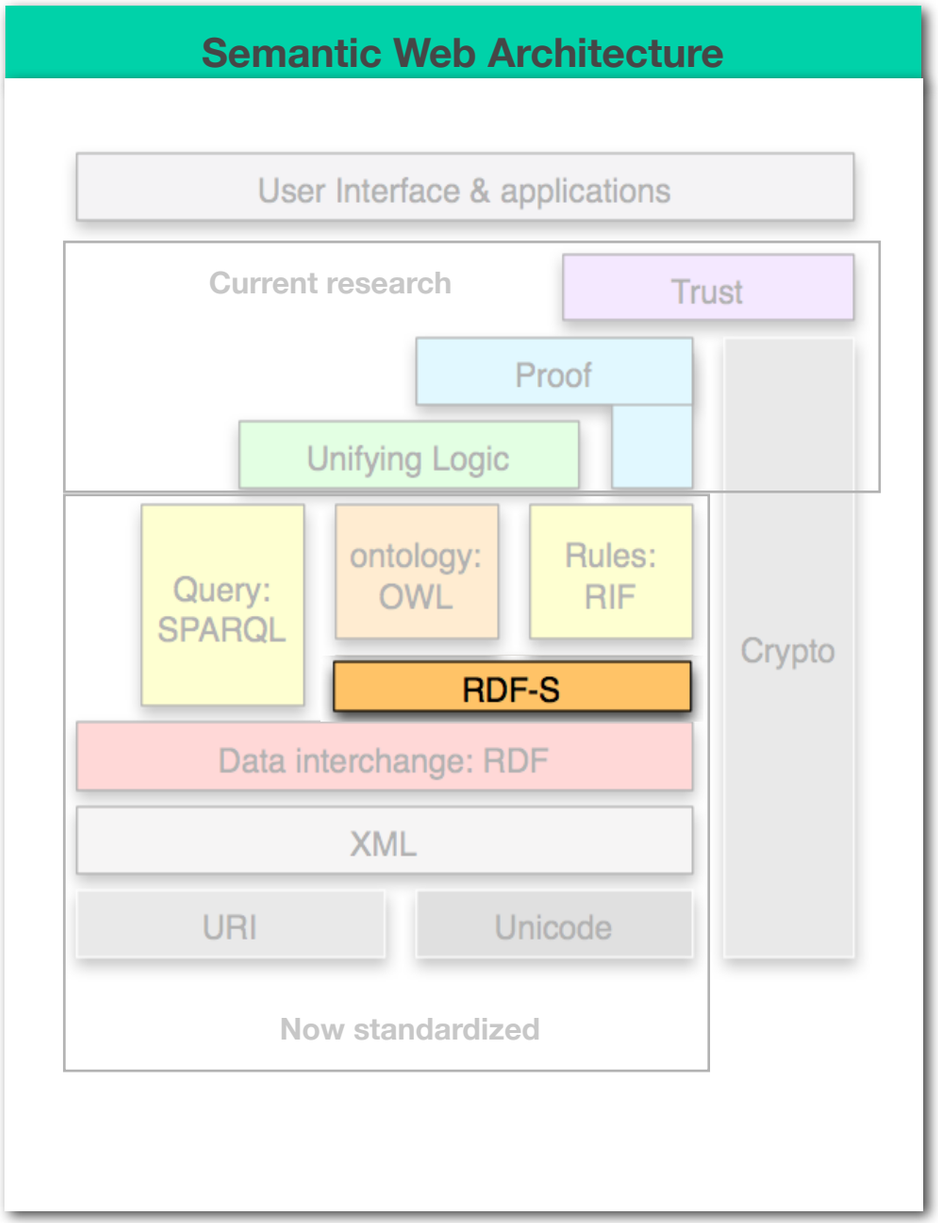




# RDF Schema

Dr. Sebastian Rudolph





# Agenda



- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# Agenda

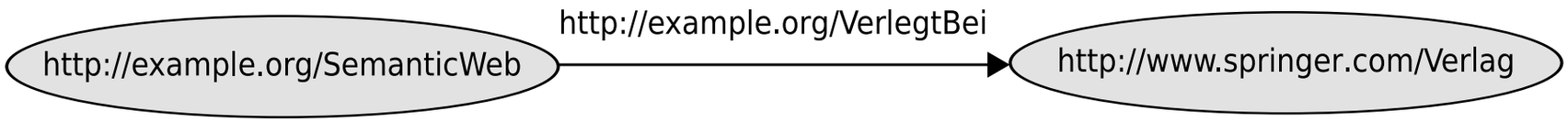


- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# Schemawissen mit RDFS

AIFB

- RDF bietet universelle Möglichkeit zur Kodierung von faktischen Daten im Web:




- = Aussagen über einzelne Ressourcen (Individuen) und deren Beziehungen
- wünschenswert: Aussagen über generische Mengen von Individuen (Klassen), z.B. Verlage, Organisationen, Personen etc.

# Schemawissen mit RDFS



- weiterhin wünschenswert: Spezifikation der logischen Zusammenhänge zwischen Individuen, Klassen und Beziehungen, um möglichst viel Semantik des Gegenstandsbereiches einzufangen, z.B.:
  "*Verlage sind Organisationen.*"
  "*Nur Personen schreiben Bücher.*"
- in Datenbanksprache: Schemawissen

# Schemawissen mit RDFS



- RDF Schema (RDFS):
  - Teil der W3C Recommendation zu RDF
  - ermöglicht Spezifikation von *schematischem* (auch: *terminologischem*) *Wissen*
  - spezielles RDF-Vokabular (also: jedes RDFS-Dokument ist ein RDF-Dokument
  - Namensraum (i.d.R. abgekürzt mit `rdfs:`) : `http://www.w3.org/2000/01/rdf-schema#`

# Schemawissen mit RDFS



- RDF Schema (RDFS):
  - jedoch: Vokabular nicht themengebunden (wie z.B. bei FOAF), sondern generisch
  - erlaubt die Spezifikation (von Teilen) der Semantik beliebiger RDF-Vokabulare (ist also eine Art „Metavokabular“)
  - Vorteil: jede Software mit RDFS-Unterstützung interpretiert jedes vermittels RDFS definierte Vokabular korrekt
  - Funktionalität macht RDFS zu einer Ontologiesprache (für leichtgewichtige - engl.: lightweight - Ontologien)
  - „A little semantics goes a long way.“

# Agenda

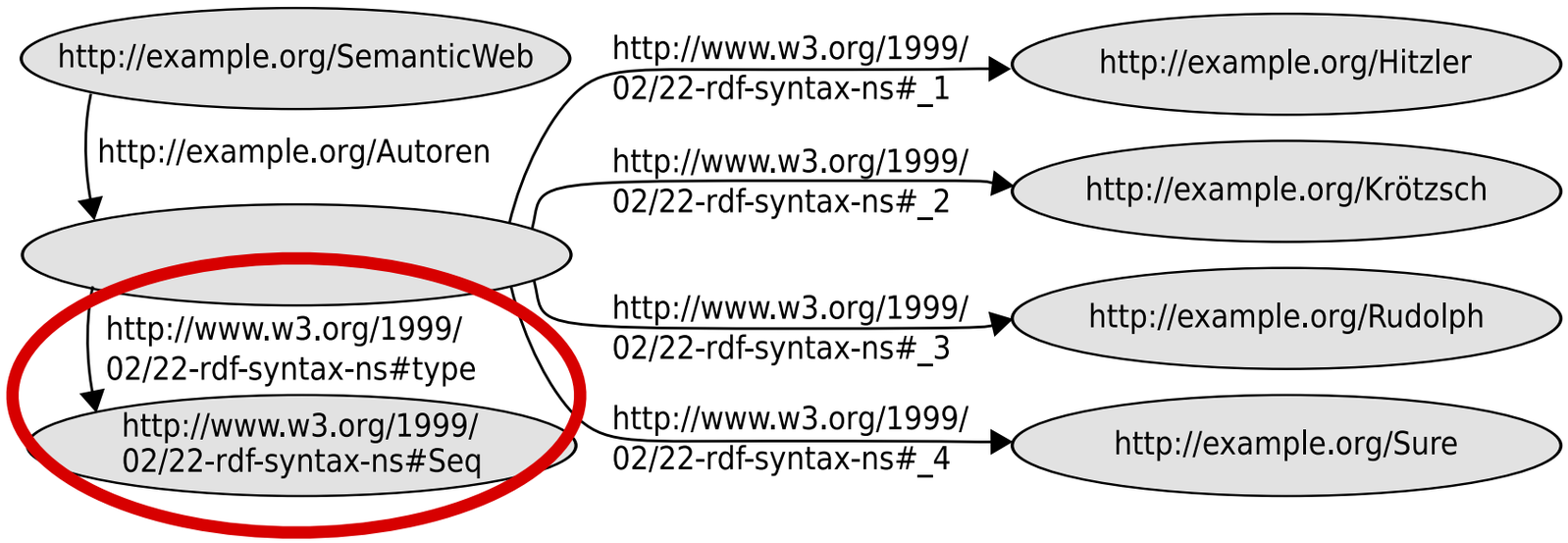


- Motivation
- **Klassen und Klassenhierarchien**
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# Klassen und Instanzen



- Typisierung von Ressourcen bereits in RDF zur Kennzeichnung von Listen:




- Prädikat `rdf:type` weist dem Subjekt das Objekt als Typ zu
- Objekt aufgefasst als Bezeichner für *Klasse*, der die durch das Subjekt bezeichnete Ressource (als sog. *Instanz*) angehört

# Klassen und Instanzen



ex:SemanticWeb rdf:type ex:Lehrbuch .

- charakterisiert „Semantic Web - Grundlagen“ als Instanz der (neu definierten) Klasse „Lehrbuch“
- Klassenzugehörigkeit ist nicht exklusiv, z.B. mit o.g. Tripel gleichzeitig möglich:
  ex:SemanticWeb rdf:type ex:Unterhaltsam .
- allgemein: a priori syntaktisch keine eindeutige Unterscheidung zwischen Individuen- und Klassenbezeichnern möglich
- auch in der Realität Charakterisierung manchmal schwierig, beispielsweise für http://www.un.org/#URI

# Die Klasse aller Klassen



- jedoch manchmal eindeutige Kennzeichnung einer URI als Klassenbezeichner wünschenswert
- möglich durch Typung der betreffenden URI als `rdfs:Class`

```
es:Lehrbuch  rdf:type  rdfs:Class  .
```

- `rdfs:Class` ist also die „Klasse aller Klassen“ und enthält sich damit auch selbst, d.h. das folgende Tripel ist immer wahr:

```
rdfs:Class  rdf:type  rdfs:Class  .
```

# Unterklassen - Motivation

- gegeben Tripel
  `ex:SemanticWeb rdf:type ex:Lehrbuch .`
- Problem: Suche nach Instanzen der Klasse `ex:Buch` liefert kein Resultat
- Möglichkeit: Hinzufügen von Tripel
  `ex:SemanticWeb rdf:type ex:Buch .`
- löst das Problem aber nur für die eine Ressource `ex:SemanticWeb`
- automatisches Hinzufügen für alle Instanzen führt zu unnötig großen RDF-Dokumenten

# Unterklassen



- Sinnvoller: einmalige Aussage, dass jedes Lehrbuch auch ein Buch ist, d.h. jede Instanz der Klasse `ex:Lehrbuch` ist automatisch auch eine Instanz der Klasse `ex:Buch`

- realisiert durch die `rdfs:subClassOf`-Property:

  ```
  ex:Lehrbuch  rdfs:subClassOf  ex:Buch .
  ```

  „Die Klasse der Lehrbücher ist eine *Unterklasse* der Klasse der Bücher.“

# Unterklassen



- rdfs:subClassOf-Property ist reflextiv, d.h. jede Klasse ist Unterklasse von sich selbst, so dass z.B. gilt:

```
ex:Lehrbuch  rdfs:subClassOf  ex:Lehrbuch .
```

- umgekehrt: Festlegung der Gleichheit zweier Klassen durch gegenseitige Unterklassenbeziehung, etwa:

```
ex:Hospital  rdfs:subClassOf  ex:Krankenhaus .
ex:Krankenhaus  rdfs:subClassOf  ex:Hospital .
```

# Klassenhierarchien



- Üblich: nicht nur einzelne Unterklassenbeziehungen sondern ganze *Klassenhierarchien* (auch: *Taxonomien*)
  z.B.:

```
ex:Lehrbuch   rdfs:subClassOf  ex:Buch .
ex:Buch       rdfs:subClassOf  ex:Printmedium .
ex:Zeitschrift rdfs:subClassOf ex:Printmedium .
```

- in RDFS-Semantik verankert: Transitivität der rdfs:subClassOf-Property, d.h. es folgt automatisch

```
ex:Lehrbuch   rdfs:subClassOf  ex:Printmedium .
```

# KLASSENHIERARCHIEN

AIFB

- Klassenhierarchien besonders ausgeprägt etwa in Biologie (z.B. *Klass*ifikation von Lebewesen)
- z.B. zoologische Einordnung des modernen Menschen

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:ex="http://www.semantic-web-grundlagen.de/
Beispiele#">
<rdfs:Class rdf:about="&ex;Animalia">
<rdfs:label xml:lang="de">Tiere</rdfs:label>
</rdfs:Class>
<rdfs:Class rdf:about="&ex;Chordata">
<rdfs:label xml:lang="de">Chordatiere</rdfs:label>
<rdfs:subClassOf rdfs:resource="&ex;Animalia"/>
</rdfs:Class>
<rdfs:Class rdf:about="&ex;Mammalia">
<rdfs:label xml:lang="de">S¨augetiere</rdfs:label>
<rdfs:subClassOf rdfs:resource="&ex;Chordata"/>
</rdfs:Class>
<rdfs:Class rdf:about="&ex;Primates">
<rdfs:label xml:lang="de">Primaten</rdfs:label>
<rdfs:subClassOf rdfs:resource="&ex;Mammalia"/>
</rdfs:Class>
<rdfs:Class rdf:about="&ex;Hominidae">
<rdfs:label xml:lang="de">Menschenaffen</rdfs:label>
<rdfs:subClassOf rdfs:resource="&ex;Primates"/>
</rdfs:Class>
<rdfs:Class rdf:about="&ex;Homo">
<rdfs:label xml:lang="de">Mensch</rdfs:label>
<rdfs:subClassOf rdfs:resource="&ex;Hominidae"/>
</rdfs:Class>
<rdfs:Class rdf:about="&ex;HomoSapiens">
<rdfs:label xml:lang="de">Moderner Mensch</rdfs:label>
<rdfs:subClassOf rdfs:resource="&ex;Homo"/>
</rdfs:Class>
<ex:HomoSapiens rdf:about="&ex;SebastianRudolph"/>
</rdf:RDF>
```

# Klassen



- intuitive Parallele zur Mengenlehre:

| | | |
|---|---|---|
| rdf:type | entspricht | $\in$ |
| rdfs:subClassOf | entspricht | $\subseteq$ |

- rechtfertigt beispielsweise auch die Reflexivität und Transitivität von rdfs:subClassOf

# Klassen in RDF/XML-Syntax



- verkürzte Darstellungen bei Angabe von Klasseninstanzen möglich:

```
<ex:HomoSapiens rdf:about="&ex;SebastianRudolph"/>
```

an Stelle von

```
<rdf:Description rdf:about= "&ex;SebastianRudolph">
<rdf:type rdf:resource= "&ex;HomoSapiens">
</rdf:Description>
```

- dementsprechend auch

```
<rdfs:Class rdf:about="&ex;HomoSapiens"/>
```

# Vordefinierte Klassenbezeichner



- rdfs:Resource
  Klasse aller Ressourcen (also sämtliche Elemente des Gegenstandsbereiches)

- rdf:Property
  Klasse aller Beziehungen
  (= die Ressourcen, die durch Prädikats-URIs referenziert werden)

- rdf:List, rdf:Seq, rdf:Bag, rdf:Alt, rdfs:Container
  Klassen verschiedener Arten von Listen

- rdfs:ContainerMembershipProperty
  Klasse aller Beziehungen, die eine Enthaltenseinsbeziehung darstellen

# Vordefinierte Klassenbezeichner

- rdf:XMLLiteral
  Klasse aller Werte des vordefinierten Datentyps XMLLiteral

- rdfs:Literal
  Klasse aller Literalwerte (enthält also alle Datentypen als Unterklassen)

- rdfs:Datatype
  Klasse aller Datentypen (ist also wie rdfs:Class eine Klasse von Klassen)

- rdf:Statement
  Klasse aller reifizierten Aussagen (s. dort)

# Agenda



- Motivation
- Klassen und Klassenhierarchien
- **Propertys und Propertyhierarchien**
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# Propertys



- andere Bezeichnungen: Relationen, Beziehungen
- Achtung: Propertys sind in RDF(S) nicht (wie in OOP) speziellen Klassen zugeodnet
- Property-Bezeichner in Tripeln üblicherweise an Prädikatsstelle
- charakterisieren, auf welche Art zwei Ressourcen zueinander in Beziehung stehen
- mathematisch oft dargestellt als Menge von Paaren: verheiratet_mit = {(Adam,Eva),(Brad,Angelina),...}
- URI wird als Property-Bezeichner gekennzeichnet durch entsprechende Typung:
  `ex:verlegtBei rdf:type rdf:Property .`

# Unterpropertys



- ähnlich zu Unter-/Oberklassen auch Unter-/Oberpropertys denkbar und sinnvoll
- Darstellung in RDFS mittels `rdfs:subPropertyOf` z.B.:

```
ex:glücklichVerheiratetMit  rdf:subPropertyOf  rdf:verheiratetMit  .
```

- erlaubt, aus dem Tripel

  ex:Markus ex:glücklichVerheiratetMit ex:Anja .

  zu schlussfolgern, dass

  ex:Markus ex:verheiratetMit ex:Anja .

# Agenda



- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# Einschränkung von Propertys



- häufig: Property kann sinnvoll nur ganz bestimmte Ressourcen verbinden, z.B. verbindet ex:verlegtBei nur Publikationen mit Verlagen

- d.h. für alle URIs a, b folgt aus dem Tripel

  a ex:verlegtBei b .

  dass auch gilt:

  a rdf:type ex:Publikation .
  b rdf:type ex:Verlag .

- kann in RDFS direkt kodiert werden:

  ex:verlegtBei rdfs:domain ex:Publikation .
  ex:verlegtBei rdfs:range ex:Verlag .

- auch zur Angabe von Datentypen für Literale:

  ex:hatAlter rdfs:range xsd:nonNegativeInteger .

# Einschränkung von Propertys



- Propertyeinschränkungen bieten die einzige Möglichkeit, semantische Zusammenhänge zwischen Propertys und Klassen zu spezifizieren
- Achtung: Propertyeinschränkungen wirken global und konjunktiv, z.B.

```
ex:autorVon  rdfs:range  ex:Kochbuch  .
ex:autorVon  rdfs:range  ex:Märchenbuch  .
```

bedeutet: jede Entität, von der jemand Autor ist, ist **gleichzeitig** Kochbuch und Märchenbuch

- daher: als domain/range immer allgemeinste mögliche Klasse verwenden

# Agenda

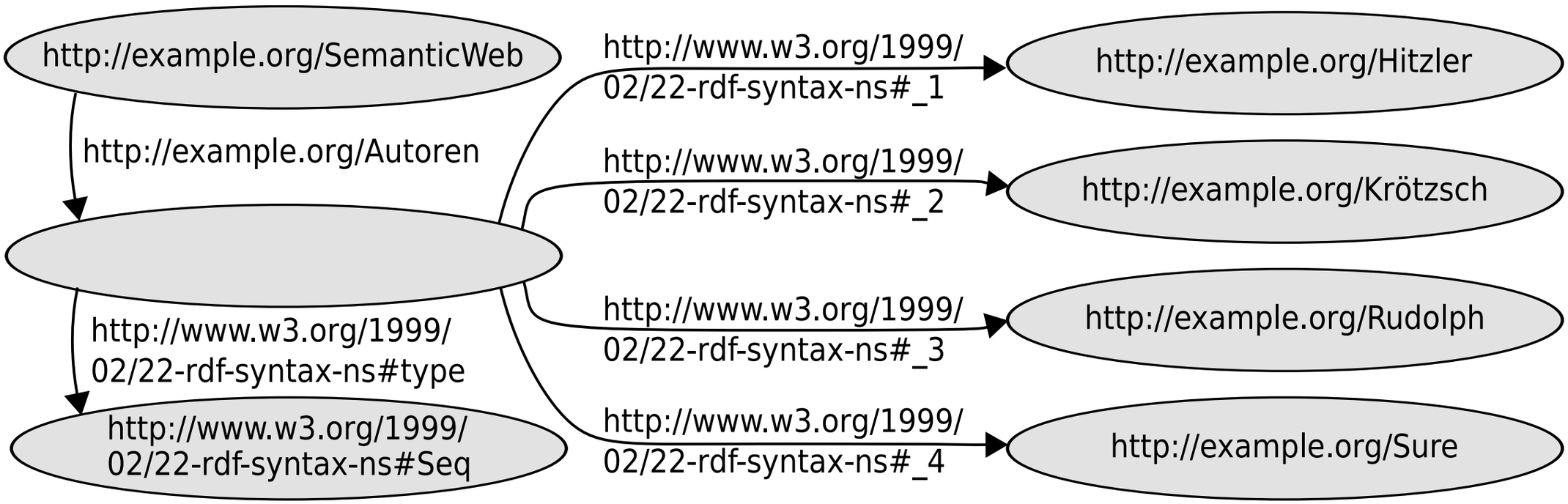


- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# Arbeit mit offenen Listen



- zur Erinnerung: offene Listen in RDF:



# Arbeit mit offenen Listen



- neue Klasse: rdfs:Container als Oberklasse von rdf:Seq, rdf:Bag, rdf:Alt
- neue Klasse: rdfs:ContainerMembershipProperty
  Elemente sind keine Individuen i.e.S. sondern selbst Propertys
- intendierte Semantik: jede Property, die aussagt, dass das Subjekt im Objekt enthalten ist, ist Instanz von rdfs:ContainerMembershipProperty
- Es gilt also insbesondere
  rdf:_1 rdf:type rdfs:ContainerMembershipProperty .
  rdf:_2 rdf:type rdfs:ContainerMembershipProperty .
  etc.

# Arbeit mit offenen Listen



- neue Property: rdfs:member
  Oberproperty aller in rdfs:ContainerMembershipProperty enthaltenen Propertys, also die „universelle Enthaltenseinsrelation“

- damit in RDFS-Semantik verankert: wann immer für eine Property p das Tripel

  p rdf:type rdfs:ContainerMembershipProperty .

  gilt, folgt aus dem Tripel

  a p b .

  sofort das Tripel

  a rdfs:member b .

# Agenda



- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- **Reifikation**
- zusätzliche Informationen in RDFS
- einfache Ontologien

# Reifikation



- Problematisch in RDF(S): Modellierung von Aussagen über Aussagen (häufig zu erkennen am Wort„dass“), z.B.: *„Der Detektiv vermutet, **dass** der Butler den Gärtner ermordet hat.“*

- erster Modellierungsversuch:
  `ex:detektiv ex:vermutet "Der Butler hat den Gärtner ermordet." .`
  - ungünstig: auf Literal-Objekt kann schlecht in anderen Aussagen Bezug genommen werden (keine URI)

- zweiter Modellierungsversuch:
  `ex:detektiv ex:vermutet ex:derButlerHatDenGärtnerErmordet .`
  - ungünstig: innere Struktur der dass-Aussage geht verloren

# Reifikation



- Problematisch in RDF(S): Modellierung von Aussagen über Aussagen (häufig zu erkennen am Wort„dass“), z.B.: *„Der Detektiv vermutet, **dass** der Butler den Gärtner ermordet hat.“*
- einzelne dass-Aussage leicht in RDF modellierbar:

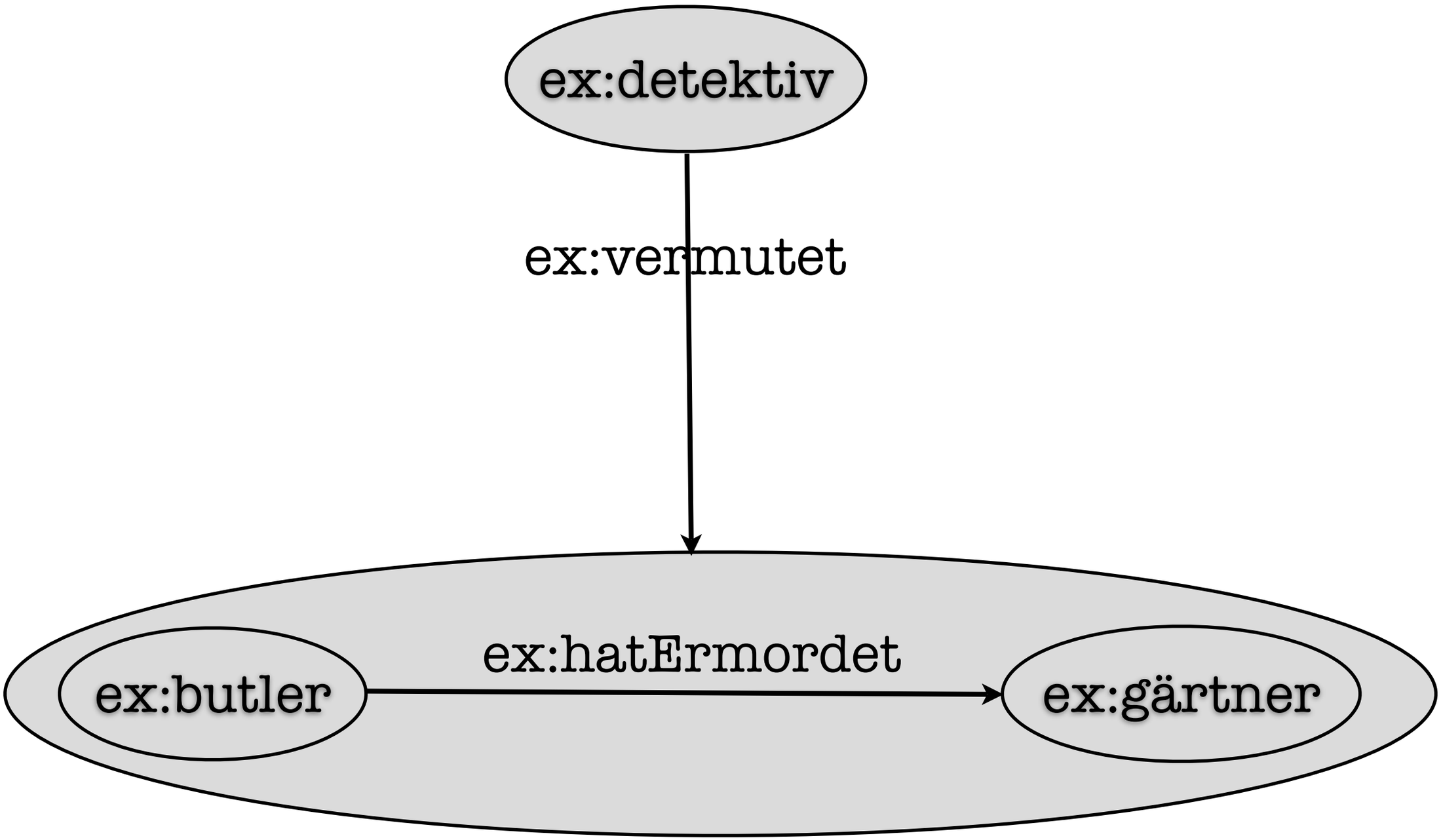
```
ex:butler  ex:hatErmordet  ex:gärtner  .
```

- wünschenswert: ganzes RDF-Tripel als Objekt eines anderen Tripels; ist aber kein gültiges RDF

# Reifikation



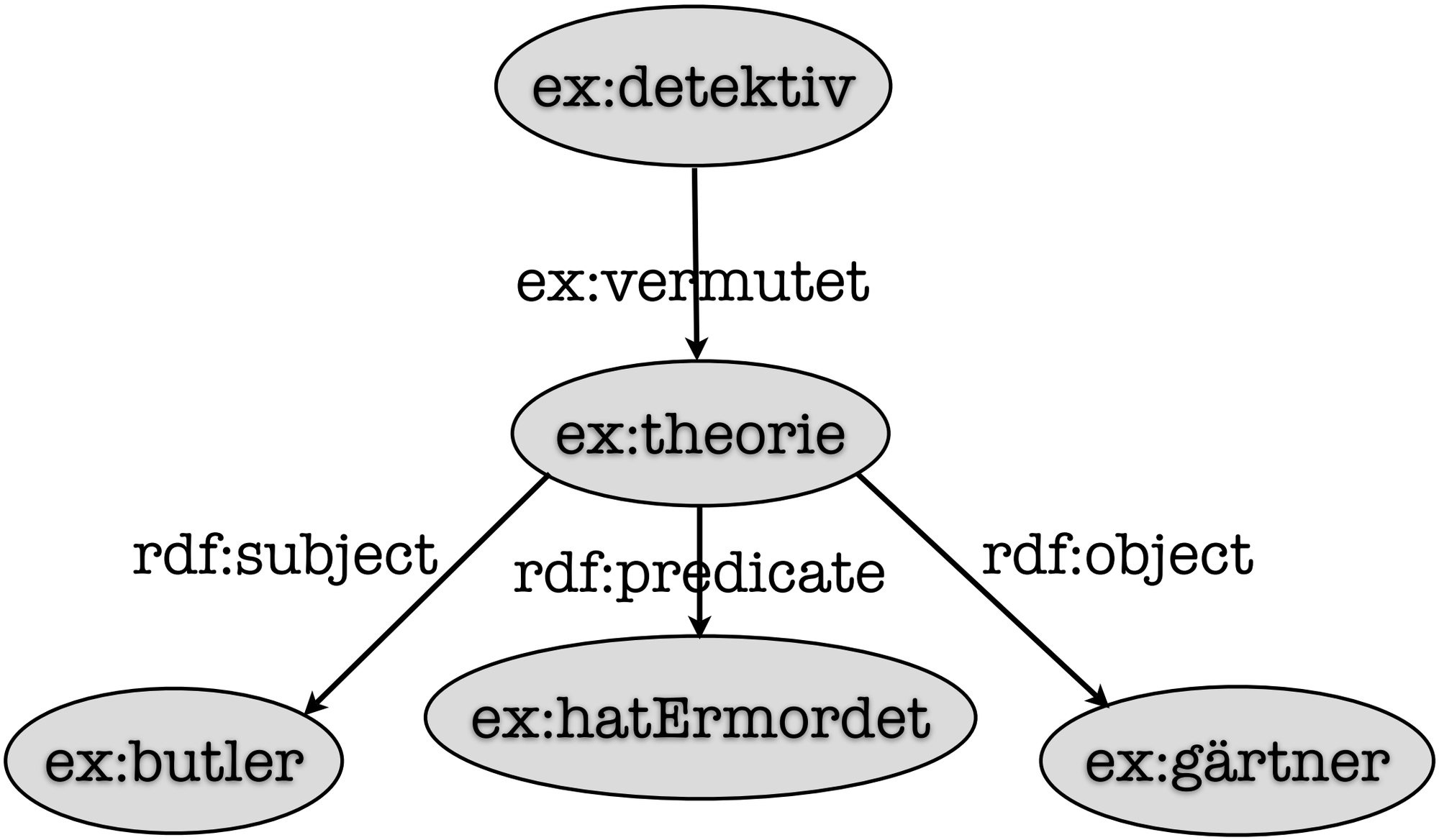
- Lösung (ähnlich wie bei mehrwertigen Beziehungen): Hilfsknoten für die geschachtelte Aussage:



# Reifikation



- Lösung (ähnlich wie bei mehrwertigen Beziehungen): Hilfsknoten für die geschachtelte Aussage:



# Reifikation

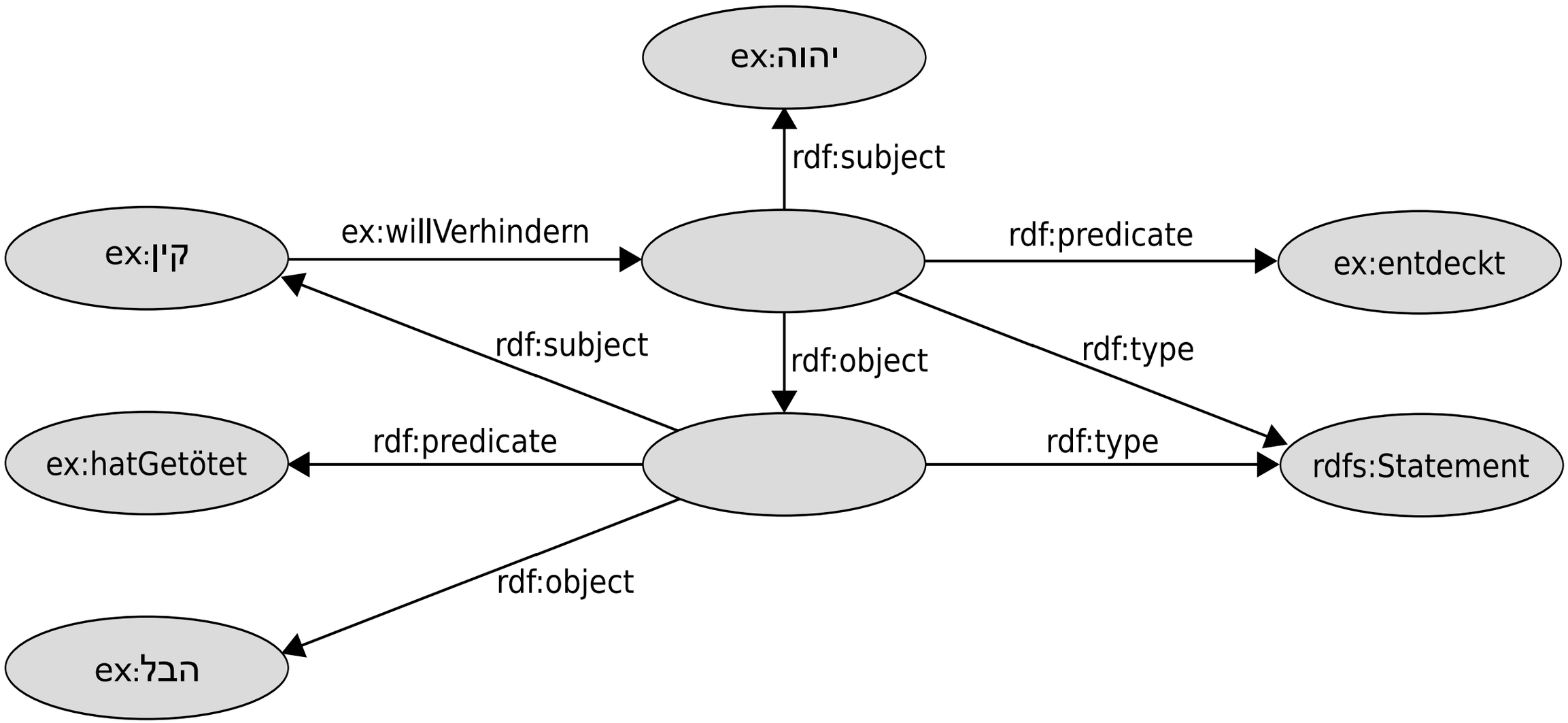


- Achtung: reifiziertes Tripel muss nicht unbedingt gelten (wäre auch nicht immer sinnvoll, z.B. bei Aussagen wie: „Der Detektiv bezweifelt, dass der Butler den Gärtner ermordet hat.“)
- falls dies gewünscht ist, muss das originale (unreifizierte) Tripel dem RDF-Dokument nochmals hinzugefügt werden
- der Klassenbezeichner `rdf:Statement` dient zur Kennzeichnung aller solcher Aussagen-Hilfsknoten
- falls auf eine Aussage nicht (extern) Bezug genommen wird, kann der entsprechende Hilfsknoten ein bnode sein

# Reifikation

AIFB

- Übungsaufgabe: noch eine Kriminalgeschichte...



# Agenda



- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- **zusätzliche Informationen in RDFS**
- einfache Ontologien

# Zusatzinformationen



- wie bei Programmiersprachen manchmal Hinzufügen von Kommentaren (ohne Auswirkung auf Semantik) wünschenswert
- Zweck: Erhöhung der Verständlichkeit für menschlichen Nutzer
- es empfiehlt sich (z.B. aus Tool-Kompatibilitätsgründen) auch dieses Wissen als Graph zu repräsentieren
- also: Satz von Propertys, die diesem Zweck dienen

# Zusatzinformationen



- rdfs:label
  - Property, die einer (beliebigen) Ressource einen alternativen Namen zuweist (Literal)
  - oftmals sind URIs schwer lesbar; zumindest „unhandlich“
  - durch rdfs:label zugewiesener Name wird z.B. häufig von Tools bei der graphischen Darstellung verwendet
  - Beispiel (incl. Sprachinformation):

```
<rdfs:Class rdf:about="&ex;Hominidae">
<rdfs:label xml:lang="de">Menschenaffen</rdfs:label>
</rdfs:Class>
```

# Zusatzinformationen



- rdfs:comment
  - Property, die einer (beliebigen) Ressource einen umfangreichen Kommentar zuweist (Literal)
  - beinhaltet z.B. natürlichsprachliche Definition einer neu eingeführten Klasse - erleichtert spätere intentionsgemäße Wiederverwendung
- rdfs:seeAlso, rdfs:definedBy
  - Propertys, die Ressourcen (URIs!) angeben, die weitere Informationen bzw. eine Definition der Subjekt-Ressource bereitstellen

# Zusatzinformationen



- Verwendungsbeispiel

```
 :
 xmlns:wikipedia="http://de.wikipedia.org/wiki/"
 :
<rdfs:Class rdf:about="&ex;Primates">
    <rdfs:label xml:lang="de">Primaten</rdfs:label>
    <rdfs:comment>
      Eine Säugetierordnung. Primaten zeichnen sich durch ein
      hochentwickeltes Gehirn aus. Sie besiedeln hauptsächlich
      die wärmeren Erdregionen.
      Die Bezeichnung Primates (lat. "Herrentiere") stammt von
      Carl von Linné.
    </rdfs:comment>
    <rdfs:seeAlso rdf:resource="&wikipedia;Primaten"/>
    <rdfs:subClassOf rdfs:resource="&ex;Mammalia"/>
</rdfs:Class>
```

# Agenda



- Motivation
- Klassen und Klassenhierarchien
- Propertys und Propertyhierarchien
- Einschränkungen auf Propertys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- **einfache Ontologien**

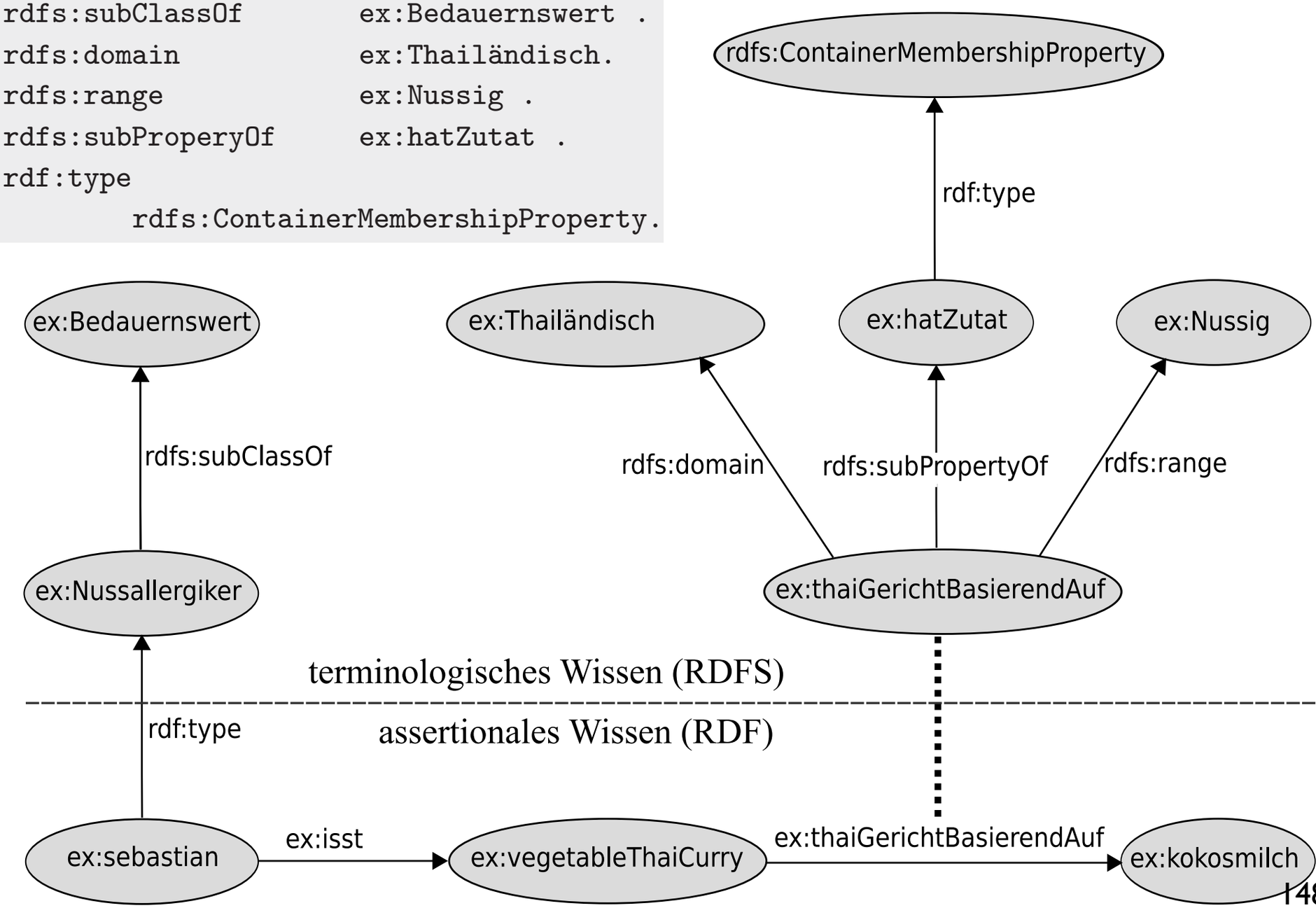
# Einfache Ontologien



- mit den durch RDFS bereitgestellten Sprachmitteln können bestimmte Gegenstandsbereiche bereits in wichtigen Aspekten semantisch erfasst werden
- auf der Basis der speziellen Semantik von RDFS kann schon ein gewisses Maß impliziten Wissens geschlussfolgert werden
- mithin stellt RDFS eine (wenn auch noch vergleichsweise wenig ausdrucksstarke) Ontologiesprache dar

# Einfache Ontologien – Beispiel

```
ex:VegetableThaiCurry   ex:ThaigerichtBasierendAuf   ex:Kokosmilch .
ex:Sebastian            rdf:type                     ex:Nussallergiker .
ex:Sebastian            ex:isst                   ex:VegetableTaiCurry .

ex:Nussallergiker          rdfs:subClassOf       ex:Bedauernswert .
ex:ThaigerichtBasierendAuf rdfs:domain           ex:Thailändisch.
ex:ThaigerichtBasierendAuf rdfs:range            ex:Nussig .
ex:ThaigerichtBasierendAuf rdfs:subProperyOf     ex:hatZutat .
ex:hatZutat                rdf:type
                                  rdfs:ContainerMembershipProperty.
```

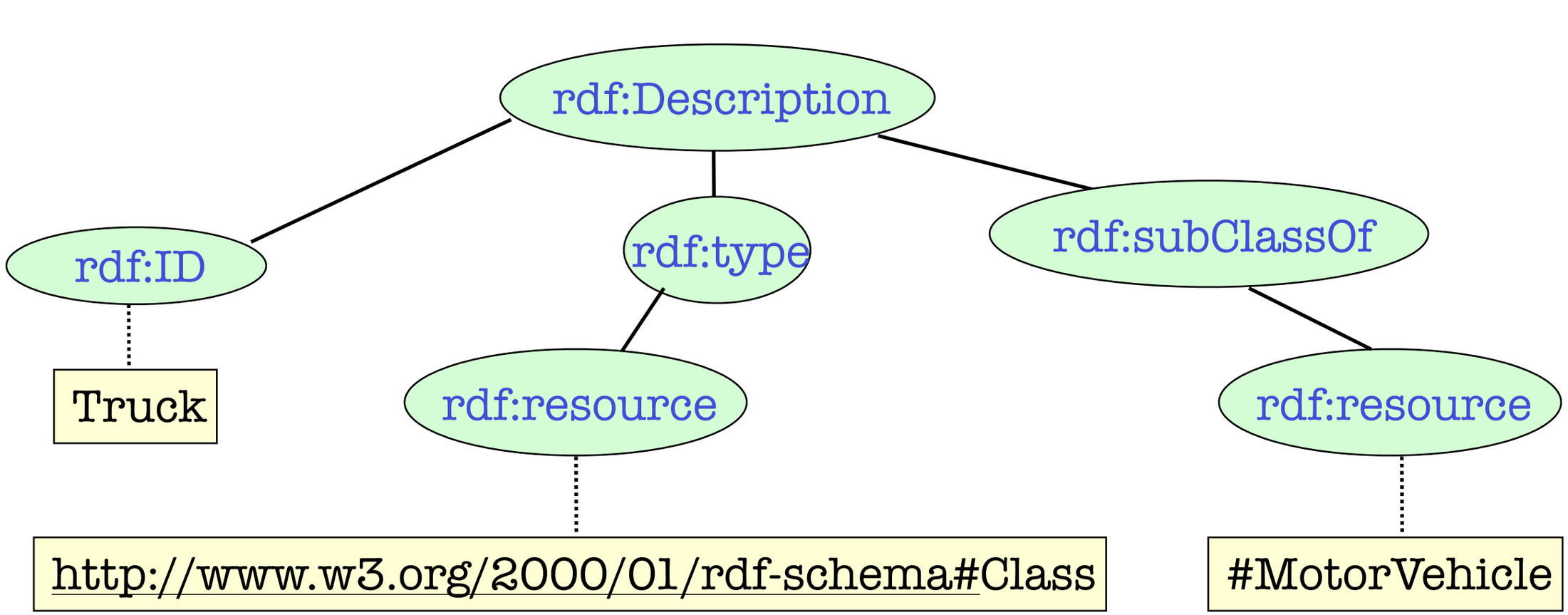


# 1 Dokument – 3 Interpretationen

AIFB

```
<rdf:Description rdf:ID="Truck">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
```

- Interpretation als XML:



# 1 Dokument – 3 Interpretationen

AIFB

```
<rdf:Description rdf:ID="Truck">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
```

- Interpretation als RDF:
  - Anderes Datenmodell
  - rdf:Description, rdf:ID und rdf:resource haben eine festgelegte Bedeutung

| | subject | predicate | object |
|---|---|---|---|
| 1. | #Truck | rdf:type | rdfs:Class |
| 2. | #Truck | rdfs:subClassOf | #MotorVehicle |

# 1 Dokument – 3 Interpretationen

AIFB

```
<rdf:Description rdf:ID="Truck">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>
```

- Interpretation als RDF Schema
  - Wieder anderes Datenmodell
  - `rdf:type` und `rdfs:subClassOf` werden speziell interpretiert