

[www.semantic-web-grundlagen.de](http://www.semantic-web-grundlagen.de)



# RDF Schema

Dr. Sebastian Rudolph

Einleitung und XML

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

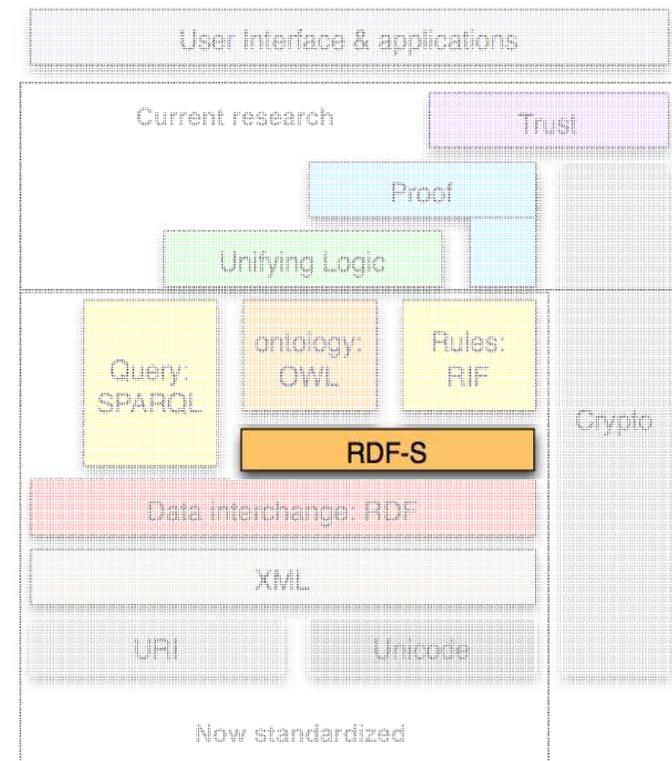
Konjunktive Anfragen / Einführung Regelsprachen

Regeln für OWL

Ontology Engineering

Semantic Web - Anwendungen

## Semantic Web Architecture



# 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

- 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

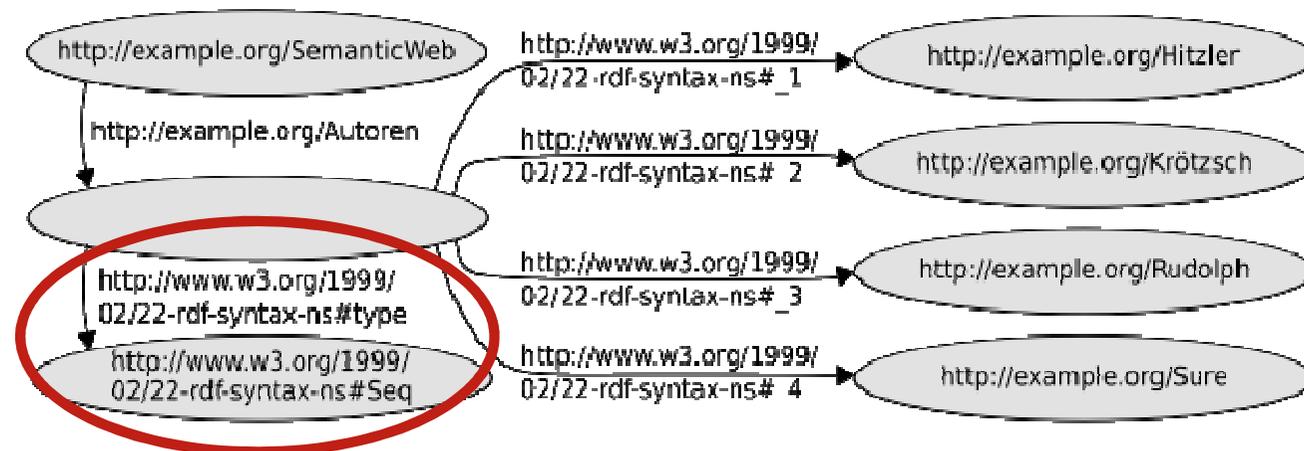
- RDF Schema (RDFS):
  - jedoch: Vokabular nicht themengebunden (wie z.B. bei FOAF), sondern generisch
  - erlaubt die Spezifikation (von Teilen) der Semantik beliebiger RDF-Vokabulare (ist also eine Art „Metavokabular“)
  - Vorteil: jede Software mit RDFS-Unterstützung interpretiert jedes mittels RDFS 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 reflexiv, 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

- Klassenhierarchien besonders ausgeprägt etwa in Biologie (z.B. *Klassifikation 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">Säugetiere</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
- **Propertyys und Propertyhierarchien**
- Einschränkungen auf Propertyys
- offene Listen
- Reifikation
- zusätzliche Informationen in RDFS
- einfache Ontologien

# Property

- andere Bezeichnungen: Relationen, Beziehungen
- Achtung: Property sind in RDF(S) nicht (wie in OOP) speziellen Klassen zugeordnet
- 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 .

# Unterproperty

- ähnlich zu Unter-/Oberklassen auch Unter-/Oberproperty denkbar und sinnvoll

- Darstellung in RDFS mittels `rdfs:subPropertyOf` z.B.:

```
ex:glücklichVerheiratetMit rdfs: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

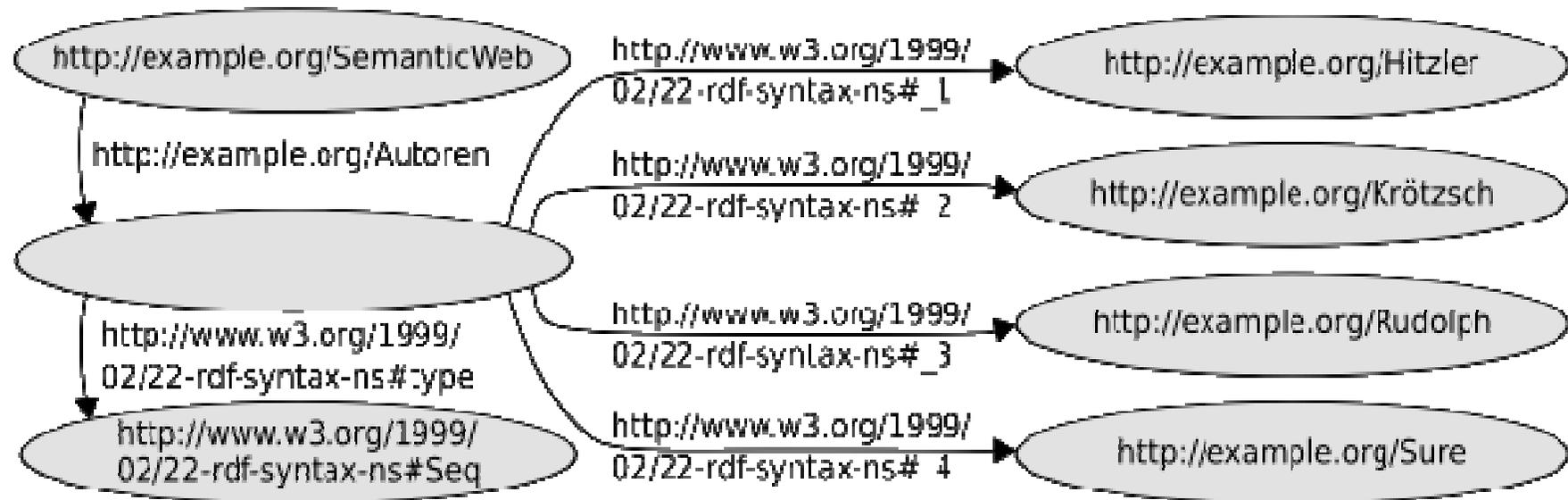
- daher: als domain/range immer allgemeinste mögliche Klasse verwenden

# 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 Property
- 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 Property, 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

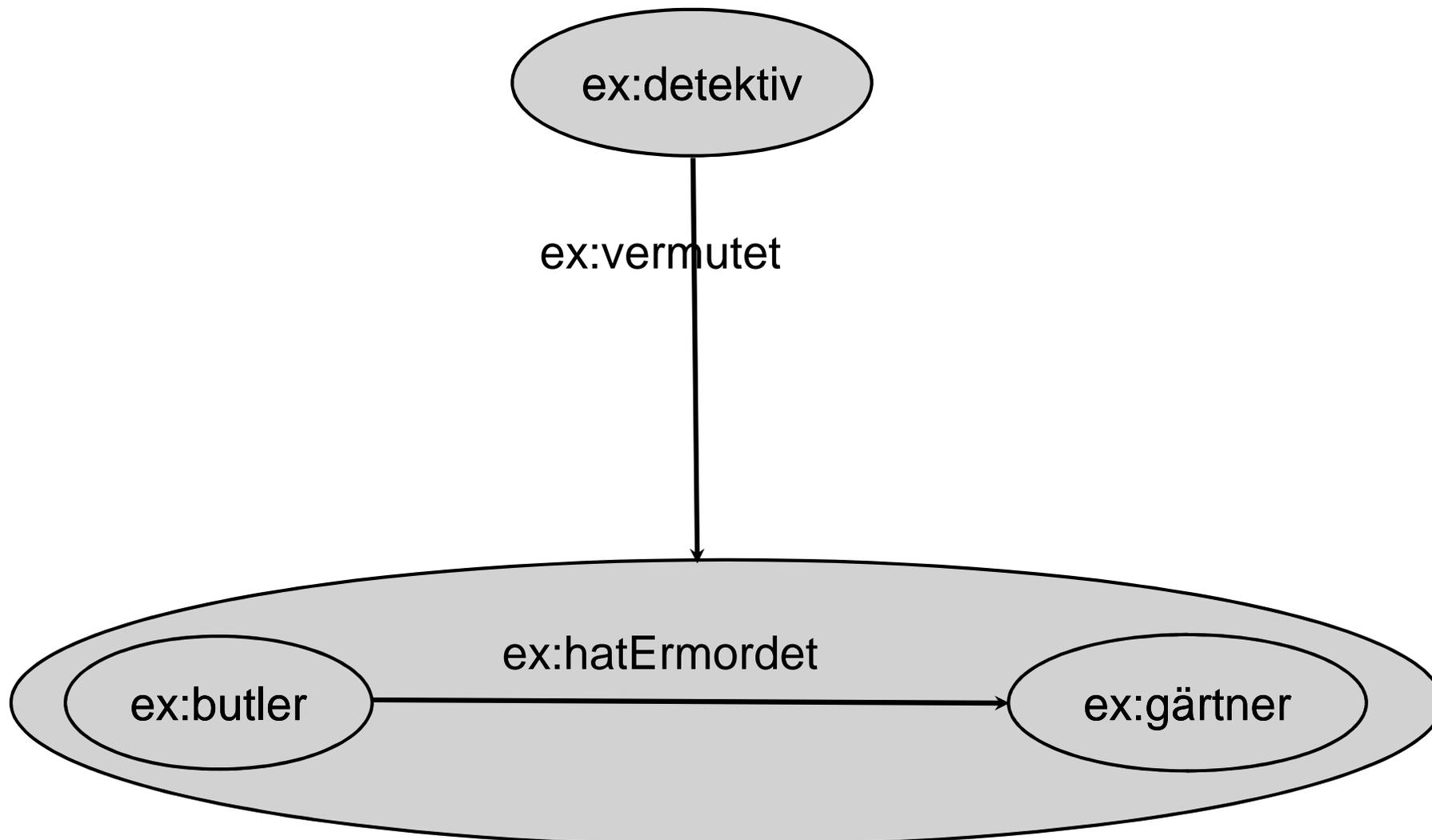
- Problematisch in RDF(S): Modellierung von Aussagen über Aussagen (häufig zu erkennen am Wort „dass“), z.B.:  
*„Der Detektiv vermutet, **dass** der Butler den Gärtner ermordet hat.“*
- erster Modellierungsversuch:  
ex:detektiv ex:vermutet "Der Butler hat den Gärtner ermordet." .
  - ungünstig: auf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

# 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

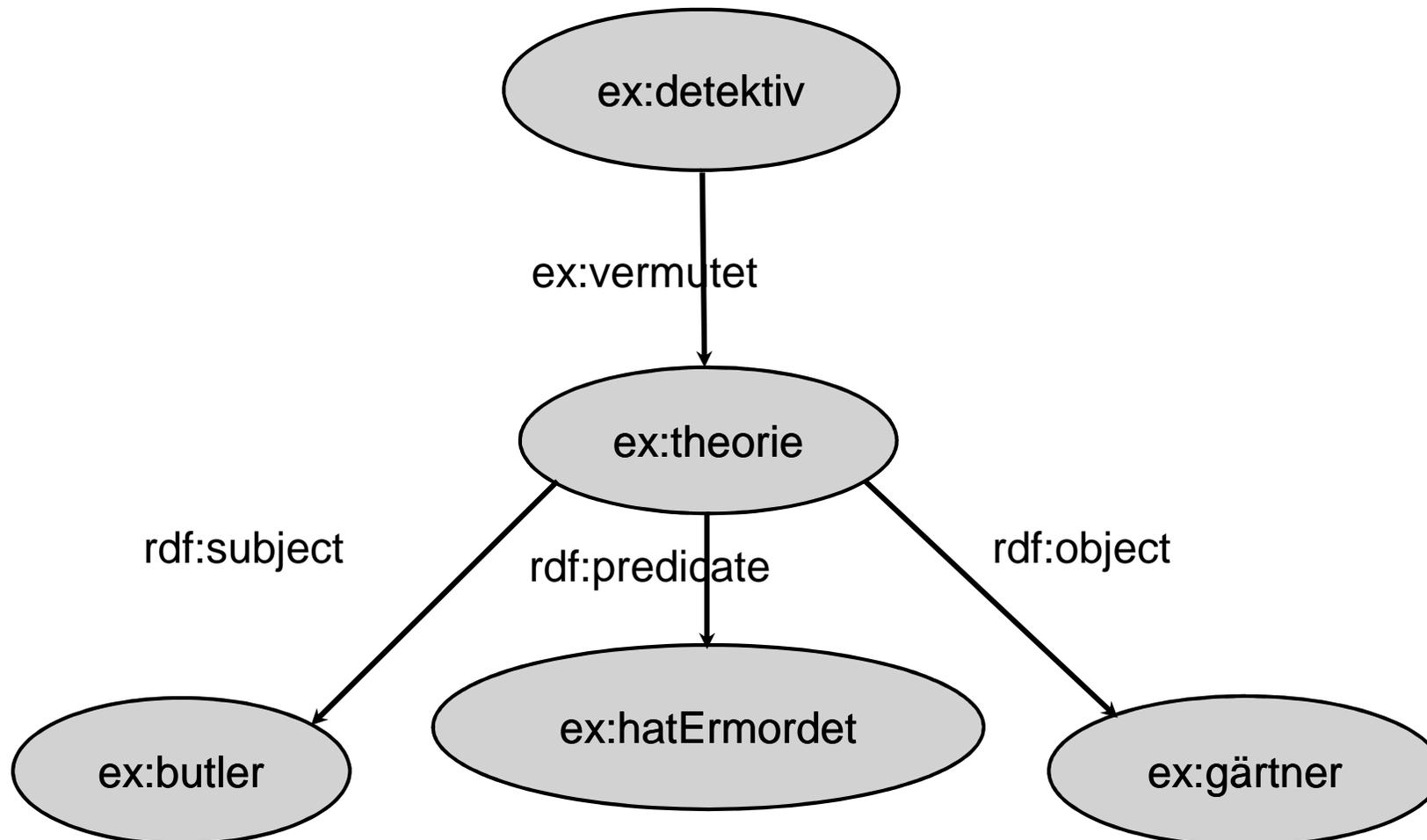
# Reifikation

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



# Reifikation

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

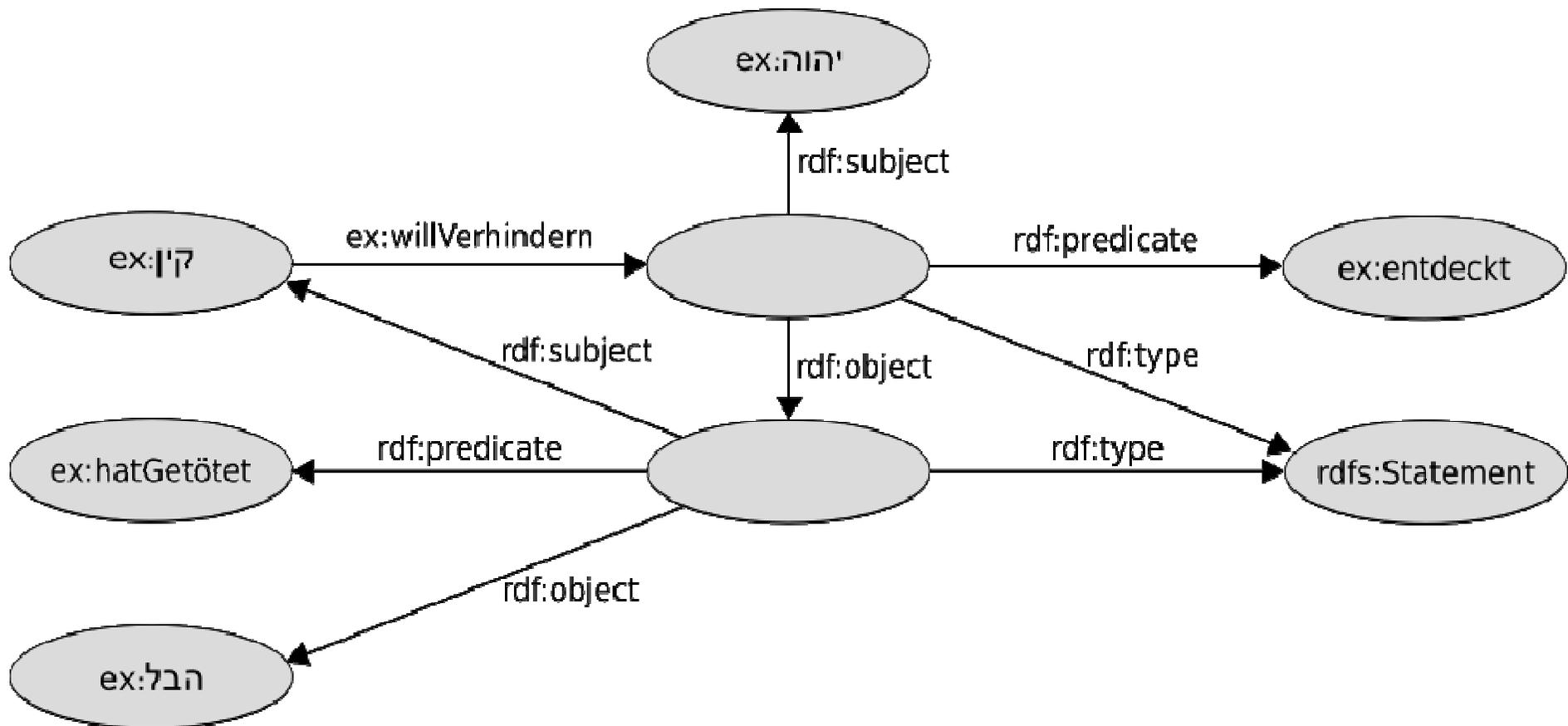


# Reifikation

- 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

- 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**
  - Properties, 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 rdf: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

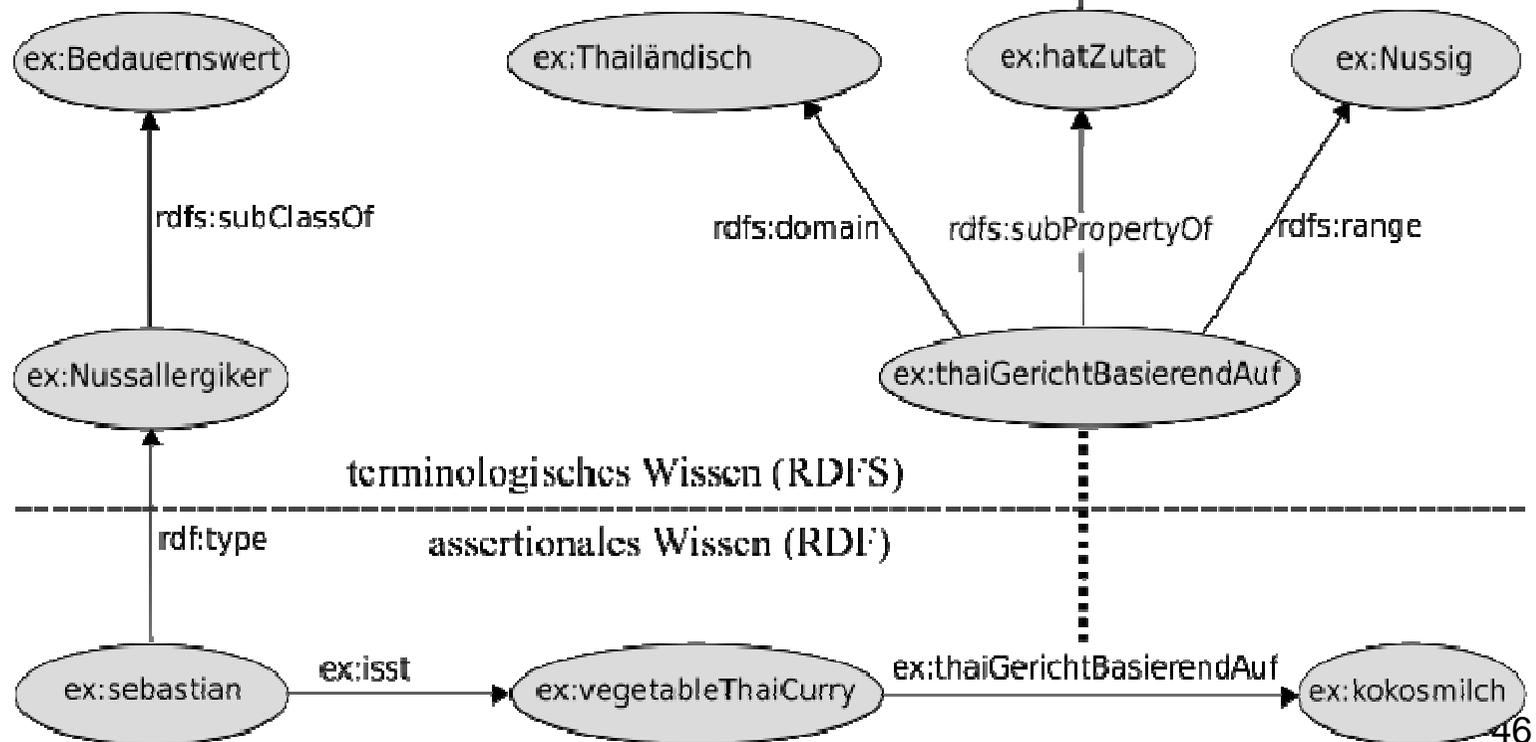
- mit den durch RDFS bereitgestellten Sprachmitteln können bestimmte Gegenstandsbereiche bereits in wichtigen Aspekten semantisch erfasst werden
- auf der Basis der speziellen Semantik von RDFS kann schon ein gewisses Maß impliziten Wissens geschlussfolgert werden
- mithin stellt RDFS eine (wenn auch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

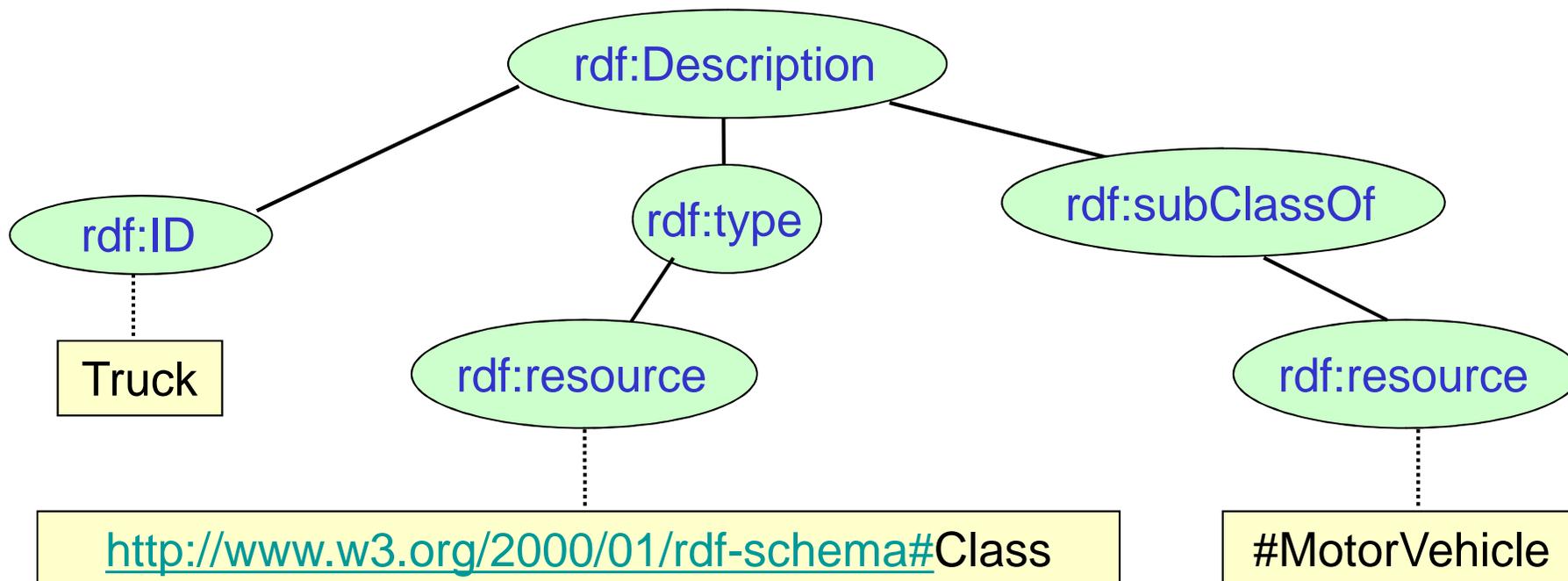
ex:Nussallergiker         rdfs:subClassOf              ex:Bedauernswert .
ex:ThaigerichtBasierendAuf rdfs:domain                  ex:Thailändisch.
ex:ThaigerichtBasierendAuf rdfs:range                   ex:Nussig .
ex:ThaigerichtBasierendAuf rdfs:subPropertyOf          ex:hatZutat .
ex:hatZutat     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



# 1 Dokument - 3 Interpretationen

```
<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

```
<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

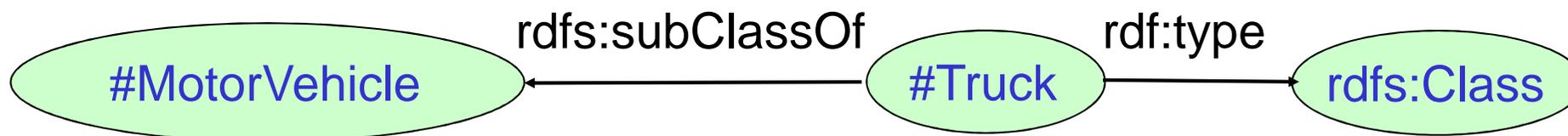
`rdf:type`

`rdfs:Class`

2. #Truck

`rdfs:subClassOf`

`#MotorVehicle`



# 1 Dokument - 3 Interpretationen

```
<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

