ulm university universität ulm

# Semantic Web Grundlagen

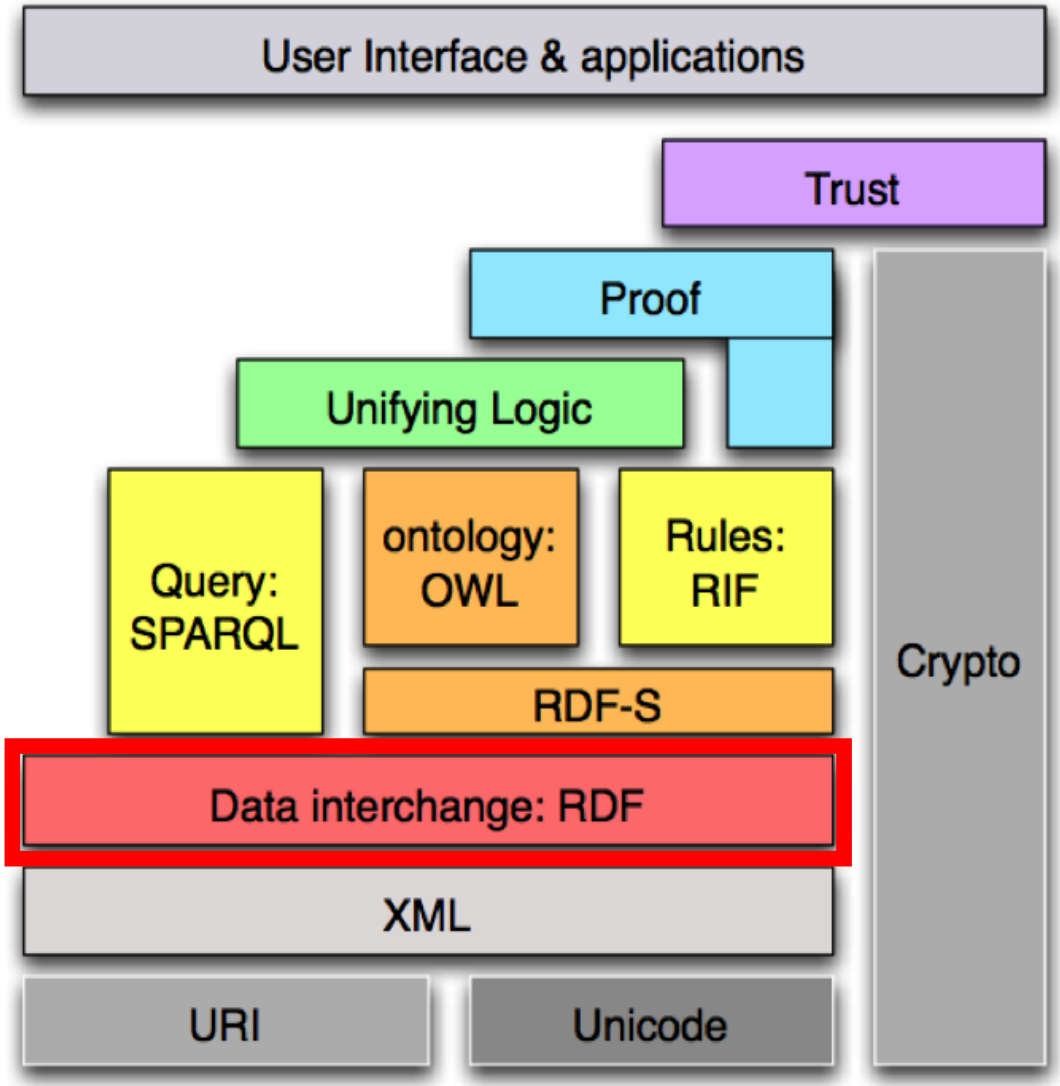
## Einführung in RDF

Birte Glimm
Institut für Künstliche Intelligenz | 20. Okt 2011

# Organisatorisches: Inhalt

| | | | |
|---|---|---|---|
| Einleitung und XML | 17. Okt | SPARQL Syntax & Intuition | 12. Dez |
| Einführung in RDF | 20. Okt | Übung 4 | 15. Dez |
| RDF Schema | 24. Okt | SPARQL Semantik | 19. Dez |
| fällt aus | 27. Okt | SPARQL 1.1 | 22. Dez |
| Logik – Grundlagen | 31. Okt | Übung 5 | 9. Jan |
| Übung 1 | 3. Nov | SPARQL Entailment | 12. Jan |
| Semantik von RDF(S) | 7. Nov | SPARQL Implemetierung | 16. Jan |
| RDF(S) & Datalog Regeln | 10. Nov | Abfragen & RIF | 19. Jan |
| OWL Syntax & Intuition | 14. Nov | Übung 6 | 23. Jan |
| Übung 2 | 17. Nov | Ontology Editing | 26. Jan |
| OWL & BLs | 21. Nov | Ontology Engineering | 30. Jan |
| OWL 2 | 24. Nov | Linked Data | 2. Feb |
| Tableau | 28. Nov | Übung 7 | 6. Feb |
| Übung 3 | 1. Dez | SemWeb Anwendungen | 9. Feb |
| Blocking & Unravelling | 5. Dez | Wiederholung | 13. Feb |
| Hypertableau | 8. Dez | Übung 8 | 16. Feb |

# Einführung in RDF



# Agenda

- XML – 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

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

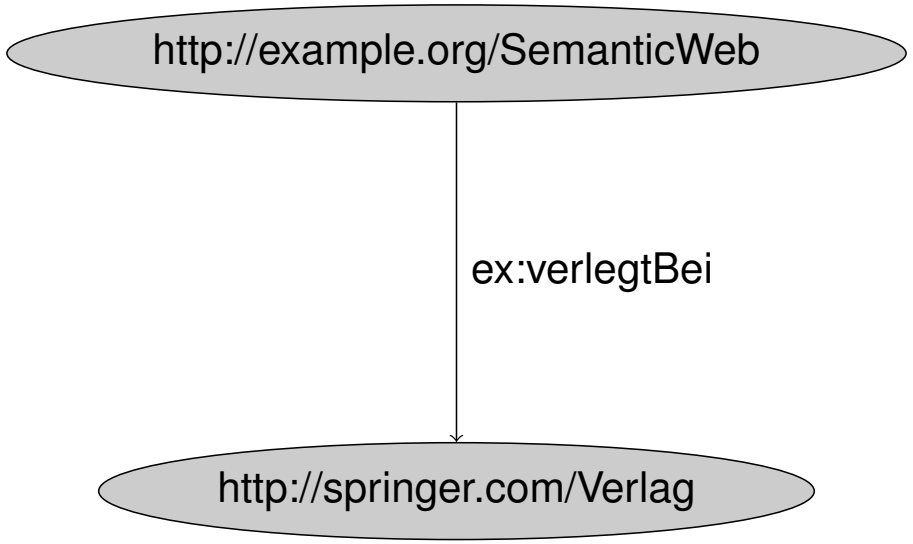
```
<Verlegt>
  <Verlag>Springer-Verlag</Verlag>
  <Buch>Semantic Web -- Grundlagen</Buch>
</Verlegt>

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

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

# RDF: Graphen statt Bäume

Lösung: Darstellung durch (gerichtete Graphen)



# Agenda

- XML – Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML
- Datentypen
- Mehrwertige Beziehungen
- Leere Knoten
- Listen

# Allgemeines zu RDF

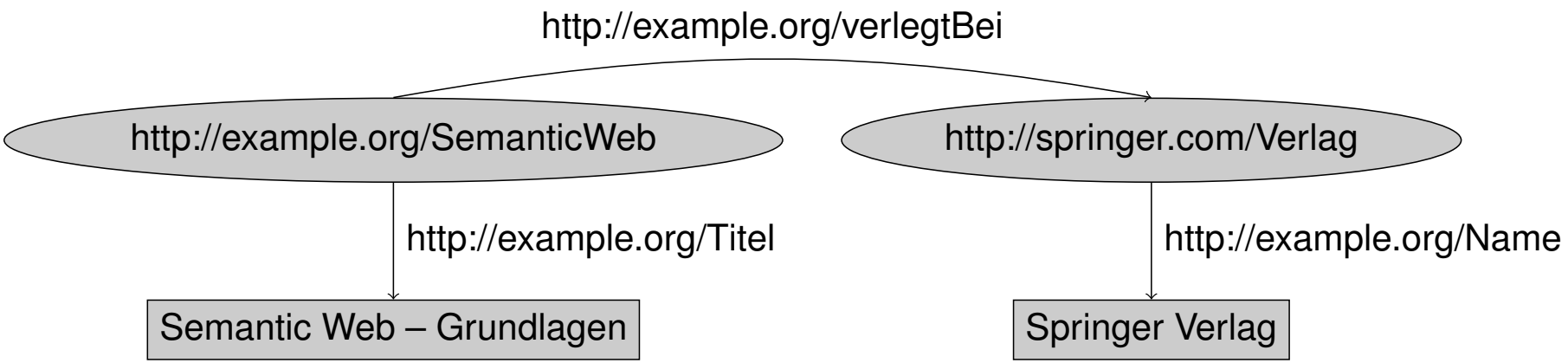
- “Resource Description Framework”
- W3C Recommendation (http://www.w3.org/RDF)
- Zur Zeit in der Überarbeitung
- RDF ist ein Datenmodell
  - ursprünglich: zur Angabe von Metadaten für Web-Ressourcen, später allgemeiner
  - kodiert strukturierte Informationen
  - universelles, maschinenlesbares Austauschformat

# 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

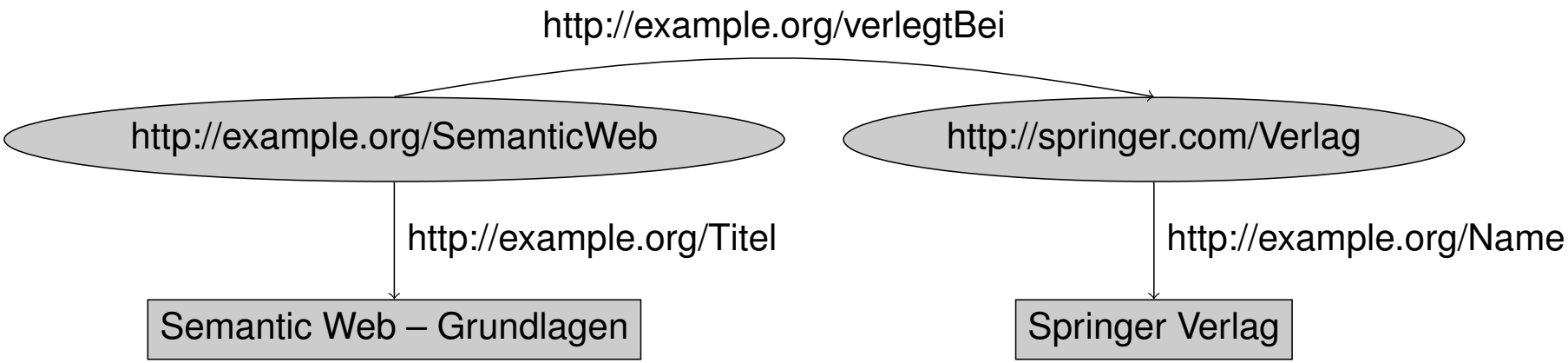
# Literale

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



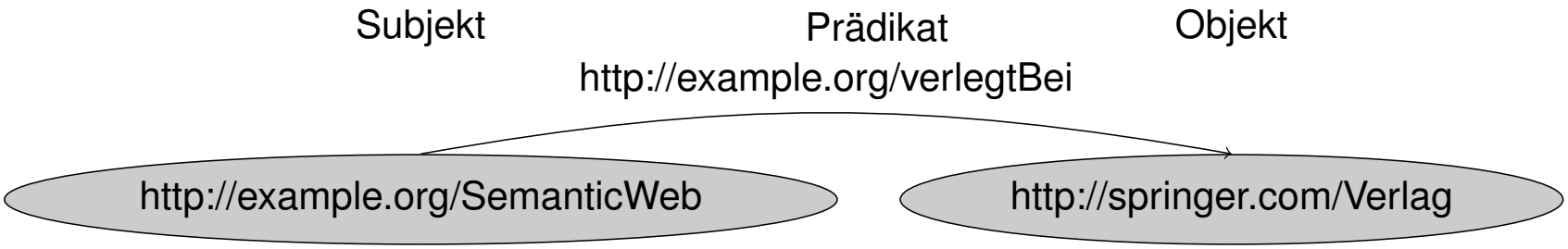
# Graph als Menge von Tripeln

- Verschiedene Darstellungsmöglichkeiten für Graphen
- Hier verwendet: Liste von (Knoten-Kante-Knoten)-Tripeln



# RDF-Tripel

## 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

- XML – 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 Bezeichnern werden ignoriert

# Turtle Syntax: Abkürzungen

## Beispiel

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

## Auch in Turtle können Abkürzungen für Präfixe festgelegt werden:

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

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

Ebenso Tripel mit gleichem Subjekt und Prädikat:

```
@prefix ex: <http://ex.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"/>
  </ex:verlegtBei>
 </rdf:Description>

</rdf:RDF>
```

# XML-Syntax von RDF

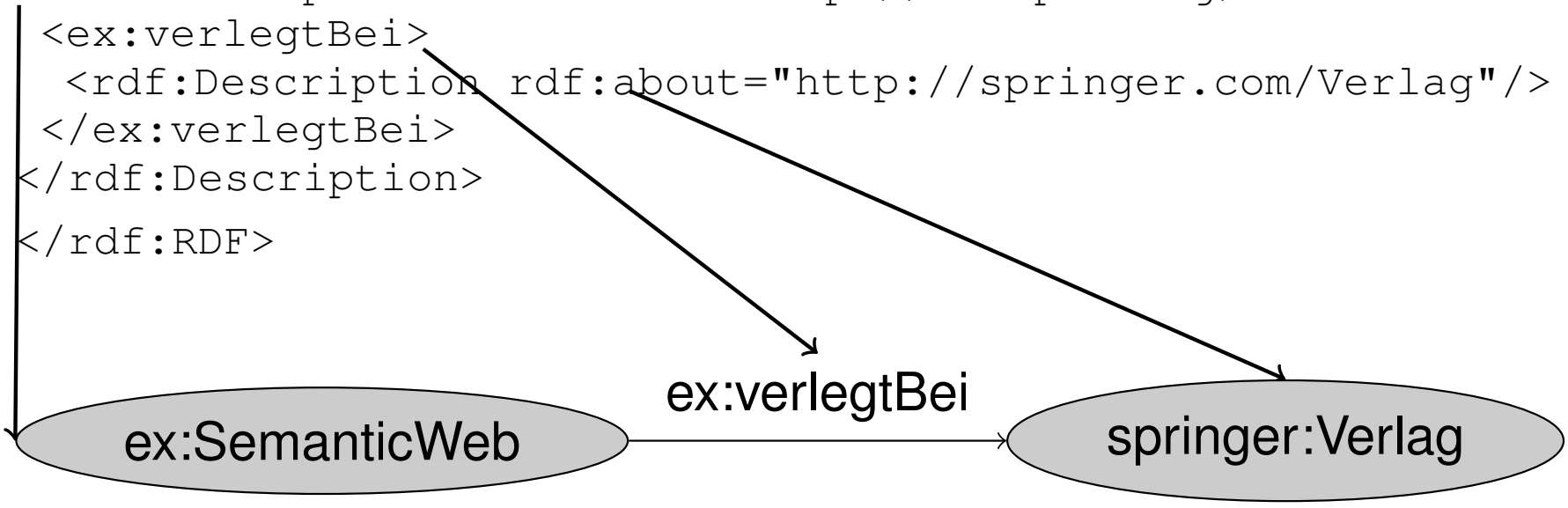
- Das rdf:Description-Element kodiert das Subjekt (dessen URI wird als Wert des zugehörigen rdf:about-Attributs angegeben).
- Jedes geschachtelt im rdf:Description-Element enthaltene Element steht für ein Prädikat (dessen URI ist der Elementname), das wiederum das Tripel-Objekt als rdf:Description-Element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

```
<?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"/>
  </ex:verlegtBei>
 </rdf:Description>
```

# XML-Syntax von RDF

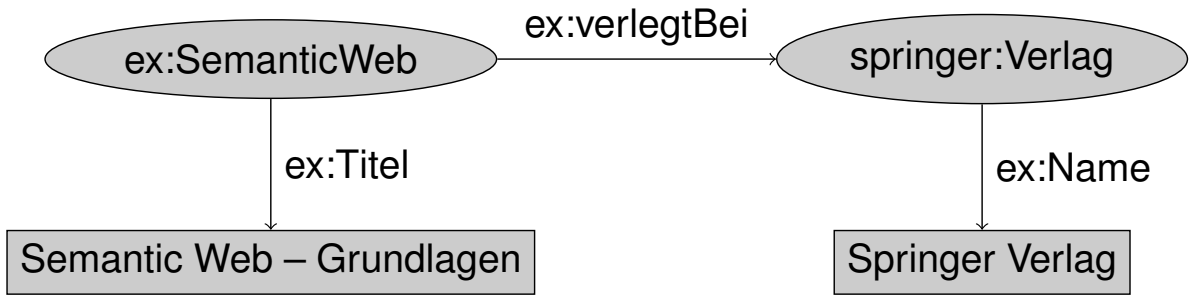
```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:ex="http://example.org/">
<rdf:Description rdf:about="http://example.org/SemanticWeb">
 <ex:verlegtBei>
  <rdf:Description rdf:about="http://springer.com/Verlag"/>
 </ex:verlegtBei>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



# 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

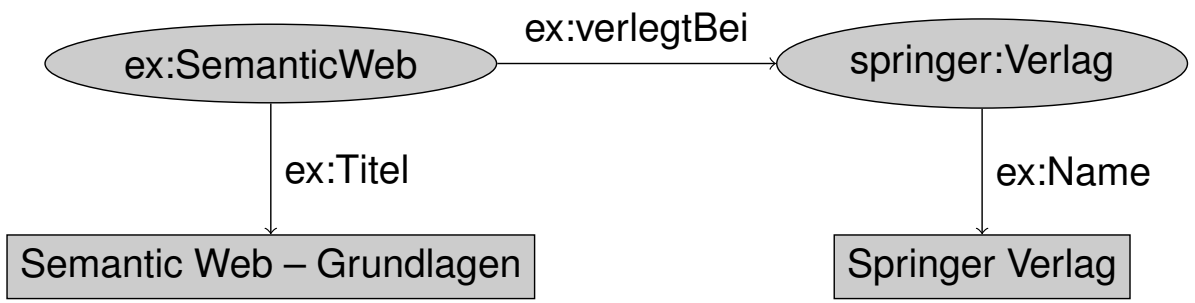
```
<rdf:Description rdf:about="http://example.org/SemanticWeb">
 <ex:Titel>Semantic Web -- Grundlagen</ex:Titel>
 <ex:verlegtBei>
  <rdf:Description rdf:about="http://springer.com/Verlag"/>
   <ex:Name>Springer Verlag</ex>Name>
  </rdf:Description>
 </ex:verlegtBei>
</rdf:Description>
```



# 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

- Arbeit mit Basis-URIs:

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

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

</rdf:RDF>
```

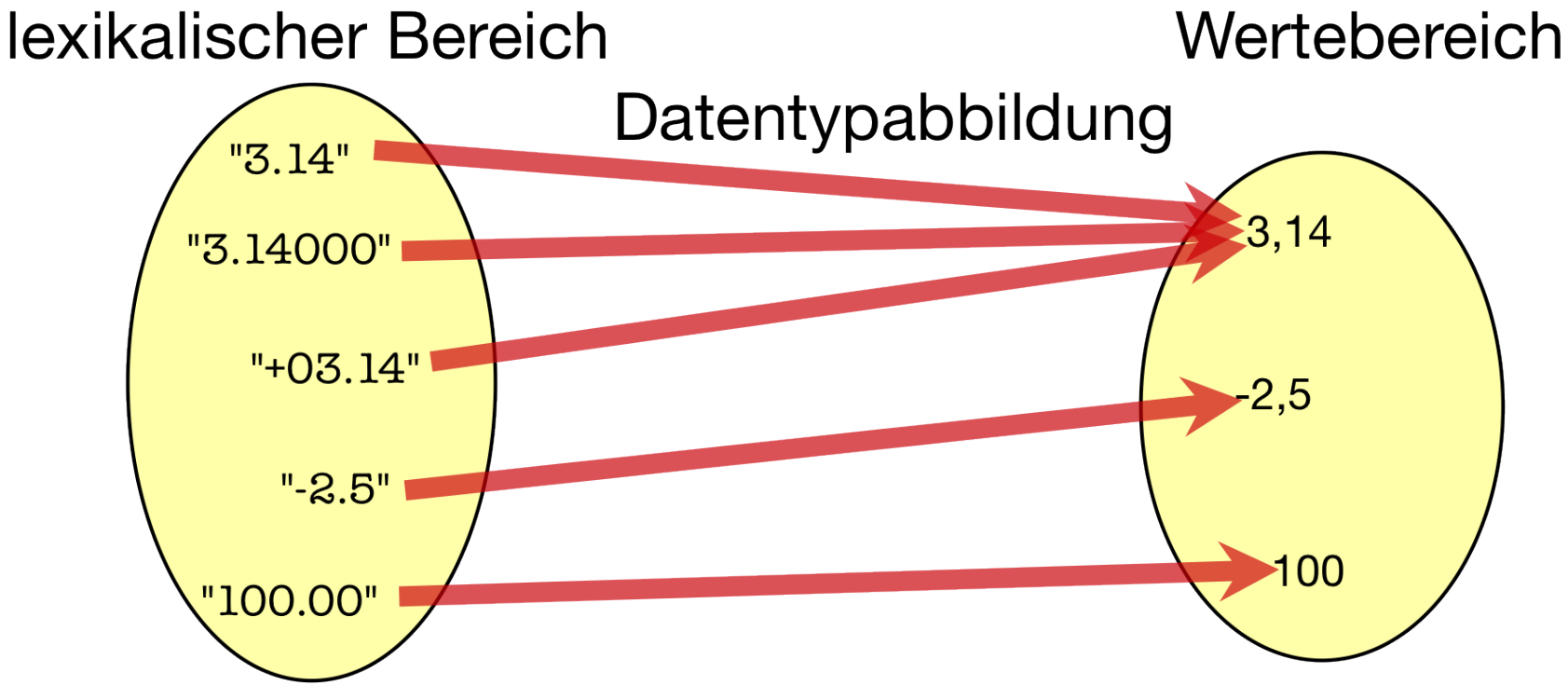
- Erkennung relativer URIs an Abwesenheit eines Schemateils

# Agenda

- XML – Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML
- Datentypen
- Mehrwertige Beziehungen
- Leere Knoten
- Listen

# Datentypen – Abstrakt

Beispiel: `xsd:decimal`



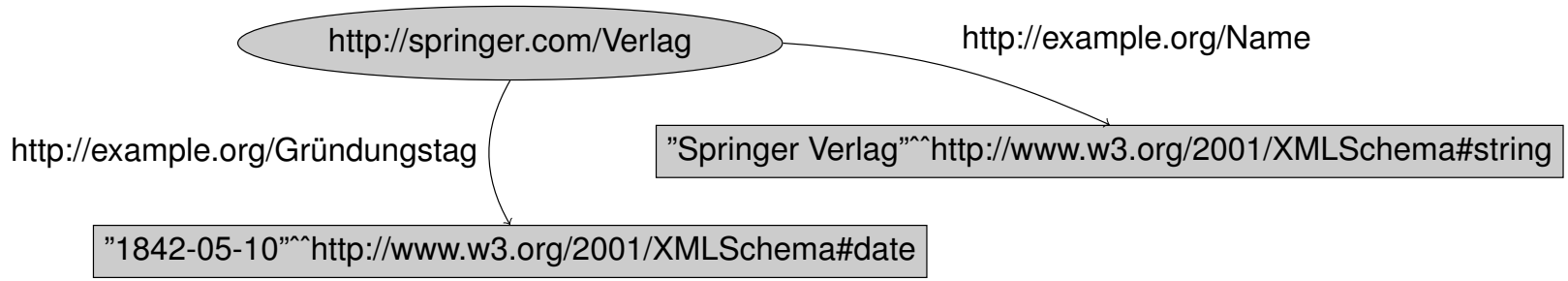

Bzgl. xsd:decimal gilt ”3.14”=”+03.14”
Bzgl. xsd:string nicht!

# 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

Graph:



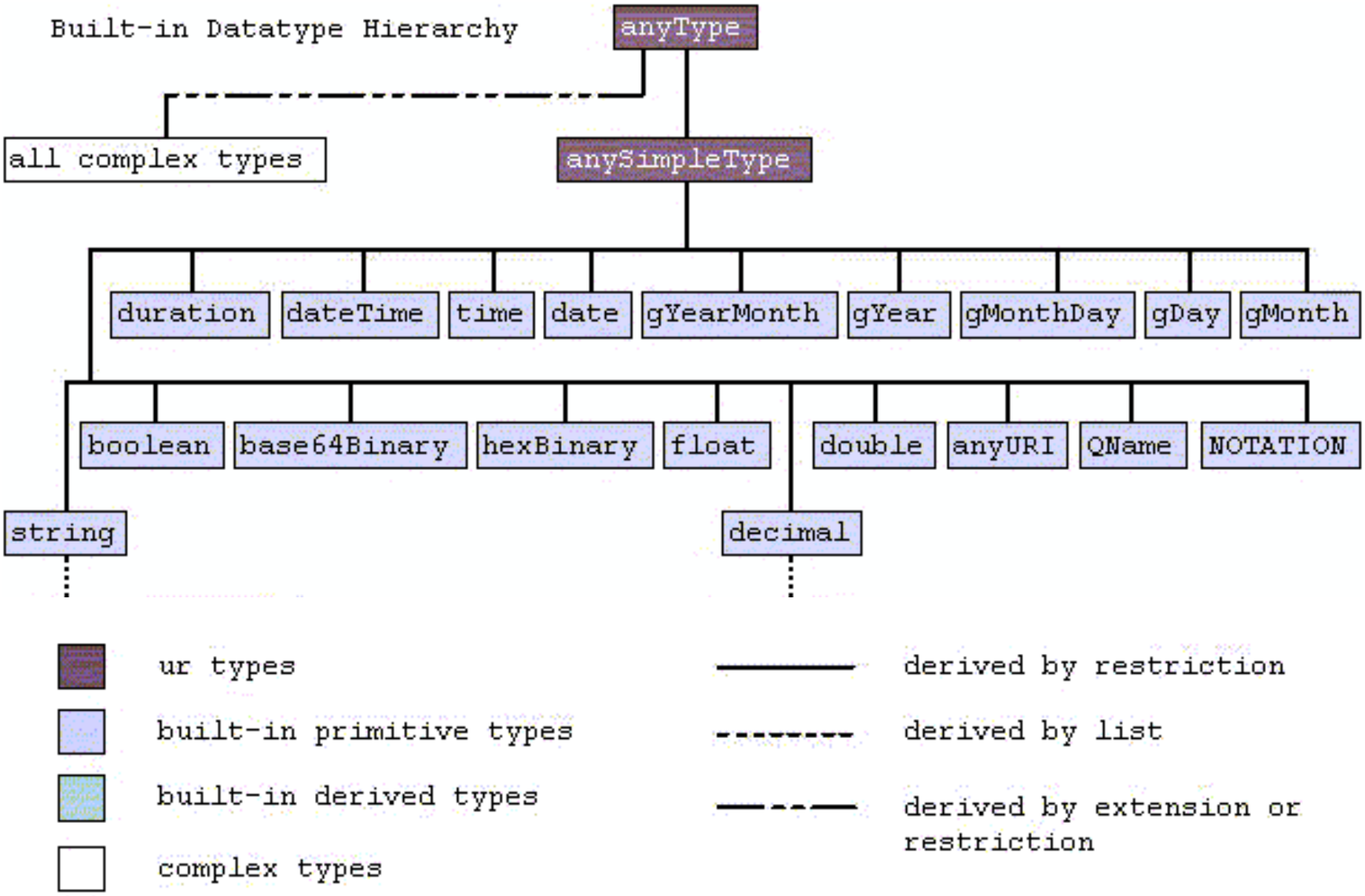

Turtle:

```
@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>
```

# XML Schema Datentypen



# XML Schema – Facets

- Facets (Deutsch: Aspekte, Facetten) sind definierende Eigenschaften eines Wertebereichs
- Grundlege Facette:
  - Abstrakte Eigenschaft zur semantischen Charakterisierung der Werte eines Wertebereichs
  - Definition von Gleichheit, Art der Ordnung (total, partiell), Grenzwerte, Kardinalität, numerischer/nicht-numerischer
- Einschränkenden Facette:
  - Optionale Eigenschaft um den Wertebereich (und damit den lexikalischen Bereich) einzuschränken
  - length (z.B. für Strings), minLength, maxLength, pattern (regulärer Ausdruck), enumeration (Beschränkung auf aufgelistete Werte), whiteSpace (mögliche Werte: preserve, replace (z.B. Tab durch Leerzeichen), collapse (erweitert replace), maxInclusive, maxExclusive, minExclusive, minInclusive, totalDigits, fractionDigits

# XML Schema – duration

- `duration` repräsentiert eine Zeitdauer
- sechs-dimensionaler Raum in dem Koordinaten das Gregorianische Jahr, Monat, Tag, Stunde, Minute, und Sekunde angeben wie definiert in ISO 8601 §5.5.3.2
- Lex. Form: PnYnMnDTnHnMnS
- Bsp.: P1Y2M3DT10H30M: Dauer 1 Jahr, 2 Monate, 3 Tage, 10 Stunden, und 30 Minuten)
- Facets: pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive

# XML Schema – dateTime

- `dateTime`: Objekte mit Jahr, Monat, Tag, Stunde und Minute als Integerwert, Sekunde als Decimalwert, optionale Zeitzonenangabe
- Korrespondierender Dezimalwert timeOnTimeline
- Lex. Form: ’-’? yyyy ’-’ mm ’-’ dd ’T’ hh ’:’ mm ’:’ ss (’.’ s+)? (((’+’ — ’-’) hh ’:’ mm) — ’Z’)?
- Z steht für UTC (Coordinated Universal Time = Greenwich Mean Time)
- Bsp.: 2002-10-10T12:00:00-05:00: Mittag des 10. Oktober 2002, Central Daylight Savings Time/Eastern Standard Time in der USA, entspricht 2002-10-10T17:00:00Z
- Facets: pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive

# XML Schema – time

- `time`: Bestimmter Zeitpunkt, sich täglich wiederholend
- Wie `dateTime` beschränkt auf die Zeitangabe
- Lex. Form: hh ':' mm ':' ss ('.' s+)? ((('+' — '-') hh ':' mm) — 'Z')?
- Bsp.: 12:00:00-05:00: 12:00 Central Daylight Savings Time/Eastern Standard Time in der USA, entspricht 17:00:00 UTC
- Facets: pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive

# XML Schema – date

- `date`: Ein Tag (nach oben hin offenes Intervall)
- Wie `dateTime` beschränkt auf die Datumsangabe (plus optionale Zeitzonenangabe)
- Lex. Form: ’-’? yyyy ’-’ mm ’-’ dd (((’+’ — ’-’) hh ’:’ mm) — ’Z’)?
- Bsp.: 2002-10-10-05:00: 10. Oktober 2002, Intervall startet mit -5 Stunden verglichen mit UTC

# XML Schema – gXXX

- `gYearMonth`: Ein bestimmter Gregorianischer Monat in einem bestimmten Gregorianischen Jahr
- `gYear`: Ein bestimmtes Gregorianischer Jahr
- `gMonthDay`: Ein (wiederkehrender) Tag eines Gregorianischen Jahres (dritter Tag des Aprils)
- `gDay`: Ein (wiederkehrender) Tag eines Monats im Gregorianischen Kalender (dritter Tag des Monats)
- `gMonth`: Ein (wiederkehrender) Monat im Gregorianischen Kalender (erster Monat/Januar)

# XML Schema – boolean, base64 und hexBinary

- `boolean`: Werte der Boolschen Logik
  - Lex. Form: { true, false, 1, 0 }
  - Facets: pattern, whiteSpace
- `base64`: Binäre Daten mit base64-encoding mit Alphabet: a-z, A-Z, 0-9, +, /, = und whitespace
  - Facets: length, minLength, maxLength, pattern, enumeration, whiteSpace
- `hexBinary`: Binäre Daten mit hex-encoding: ”0FB7” ist hex encoding für 16-bit Integer 4023 (binäre Darstellung 0000.1111.1011.0111)
  - Facets: length, minLength, maxLength, pattern, enumeration, whiteSpace

# XML Schema – float und double

- `float`: wie IEEE single-precision 32-bit floating point type, Werte $m \times 2^e$ mit $m$, $e$ Integer, $|m| < 2^{24}$, $-149 \leq e \leq 104$ plus positiv Infinity (`INF`) und negativ Infinity (`-INF`) und not-a-number (`NaN`)
- `double`: wie IEEE double-precision 64-bit floating point type, Werte $m \times 2^e$ mit $m$, $e$ Integer, $|m| < 2^{53}$, $-1075 \leq e \leq 970$ plus positiv Infinity (`INF`) und negativ Infinity (`-INF`) und not-a-number

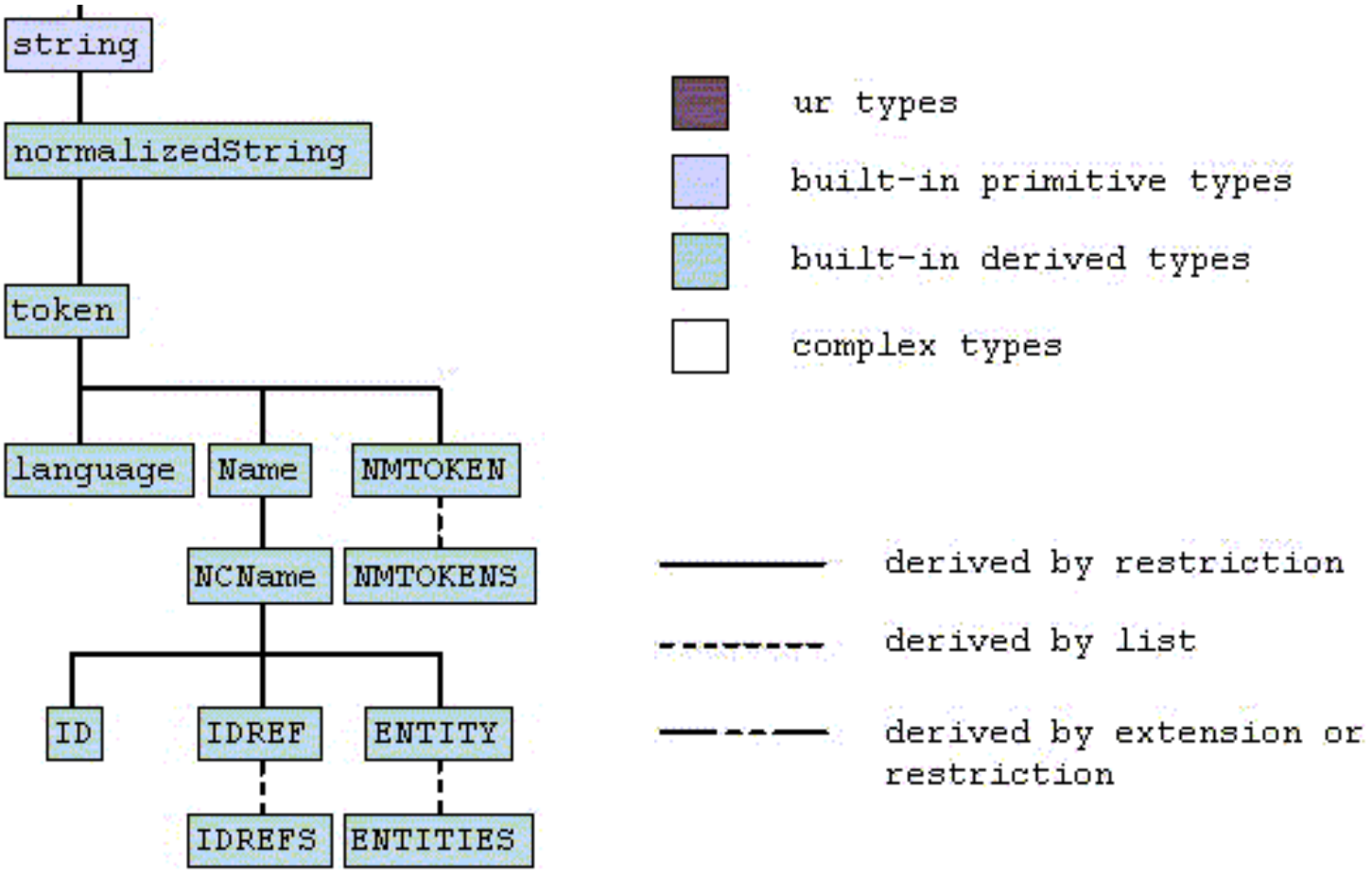
- Nicht alle Dezimalzahlen innerhalb der minimalen und maximalen Werte sind darstellbar
- Bsp.: -1E4, 1267.43233E12, 12.78e-2, 12 , -0, 0, INF
- Facets: pattern, enumeration, whiteSpace, maxInclusive, maxExclusive, minInclusive, minExclusive

# XML Schema – anyURI, QName, NOTATION

- `anyURI`: Ein Uniform Resource Identifier, absolut oder relativ, mit oder ohne Fragmentidentifizierer
- `QName`: ein qualifizierter XML Name (Namensraum plus lokaler Teil mit Namensraum `anyURI` und lokaler Teil `NCName`
- `NOTATION`: wie NOTATION Attributtyp in XML 1.0, nicht direkt verwendbar (nur abgeleitete Typen)

- Facets: length, minLength, maxLength, pattern, enumeration, whiteSpace

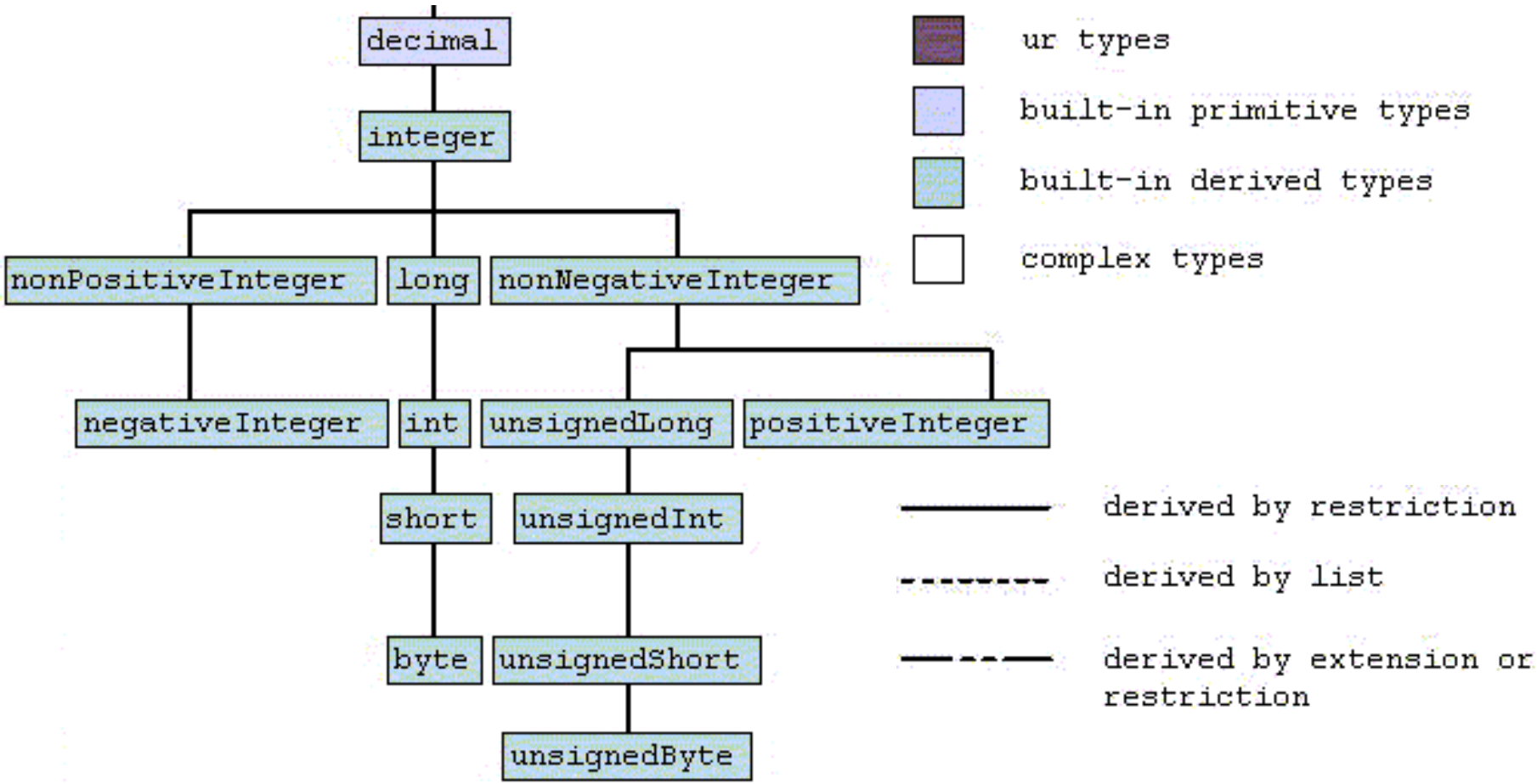
# XML Schema Datentypen



# XML Schema – string

- `string`: Sequenz von characters, character ist atomare Einheit mit entsprechendem code point (Integer) im Universal Character Set
- Facets: length, minLength, maxLength, pattern, enumeration, whiteSpace
- Weitere Spezialisierungen: siehe Spezifikation

# XML Schema Datentypen



# XML Schema – decimal und integer

- `decimal`: Teilmenge der reellen Zahlen, reprs̈entierbar als Dezimalzahlen. Werteraum: Zahlen ausdrückbar als $i \times 10^{-n}$ mit $i, n$ Integers und $n \geq 0$, Präzision unerheblich: 2.0 gleich zu 2.00
- Bsp.: -1.23, 12678967.543233, +100000.00, 210
- facets: totalDigits, fractionDigits, pattern, whiteSpace, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive
- `integer` Einschränkung von `decimal`: `fractionDigits`=0, kein Dezimalpunkt

# XML Schema – Abgeleitete Typen von integer

- `long` Einschränkung von `integer`: `maxInclusive`=9223372036854775807, `minInclusive`=-9223372036854775808
- `int` Einschränkung von `long`: `maxInclusive`=2147483647, `minInclusive`=-2147483648
- `short` Einschränkung von `int`: `maxInclusive`=32767, `minInclusive`=-32768
- `byte` Einschränkung von `short`: `maxInclusive`=127, `minInclusive`=-128

# XML Schema – Abgeleitete Typen von integer

- `nonNegativeInteger` Einschränkung von `integer`: `minInclusive`=0
- `positiveInteger` Einschränkung von `nonNegativeInteger`: `minInclusive`=1
- `unsignedLong` Einschränkung von `nonNegativeInteger`: `maxInclusive`=18446744073709551615
- `unsignedInt` Einschränkung von `unsignedLong`: `maxInclusive`=4294967295
- `unsignedShort` Einschränkung von `unsignedInt`: `maxInclusive`=65535
- `unsignedByte` Einschränkung von `unsignedShort`: `maxInclusive`=255

# XML Schema – Abgeleitete Typen von integer

- `nonPositiveInteger` Einschränkung von `integer`: `maxInclusive`=0
- `negativeInteger` Einschränkung von `nonPositiveInteger`: `minInclusive`=-1

# XML Schema – Kanonische Werte

- Es kann mehrere lexikalische Formen für einen Wert geben
- Kanonische Form legt davon eine *kanonische* Form fest
- Nützlich um verschiedene Schreibweisen gleicher Werte zu identifizieren
- Die folgenden lexikalischen Formen für den `decimal` Datentyp repr
  äsentieren den gleichen Wert: 100.5, +100.5, 0100.5, 100.50, 100.500, 100.5000, kanonisch ist: 100.5

# 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

- XML – 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",
                       "1 TL 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:hatZutat ex:grüneMango;
           ex:Menge "450g";
           ex:hatZutat ex:Cayennepfeffer;
           ex:Menge "1 TL";
           ...
```

- 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)

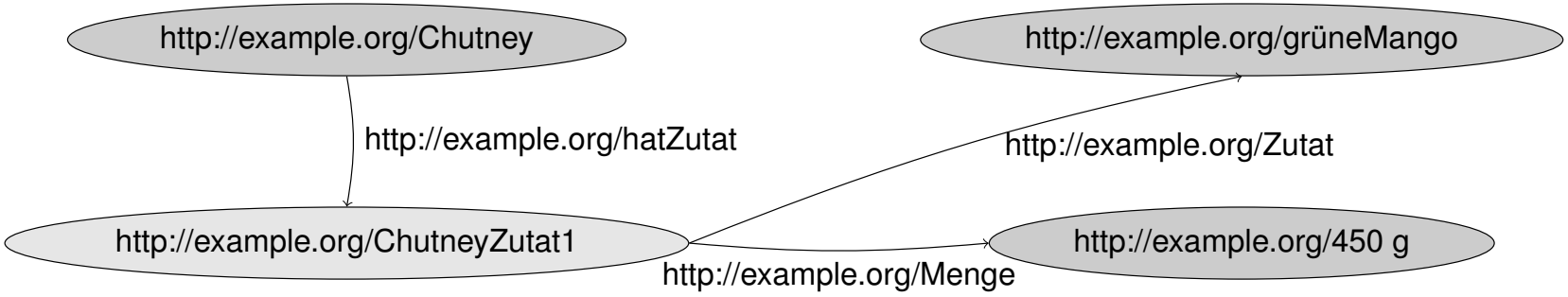
| Gericht | Zutat | Menge |
|---|---|---|
| Chutney | grüne Mango | 450g |
| Chutney | Cayennepfeffer | 1 TL |

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

# Mehrwertige Beziehungen

## 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" .
...
```

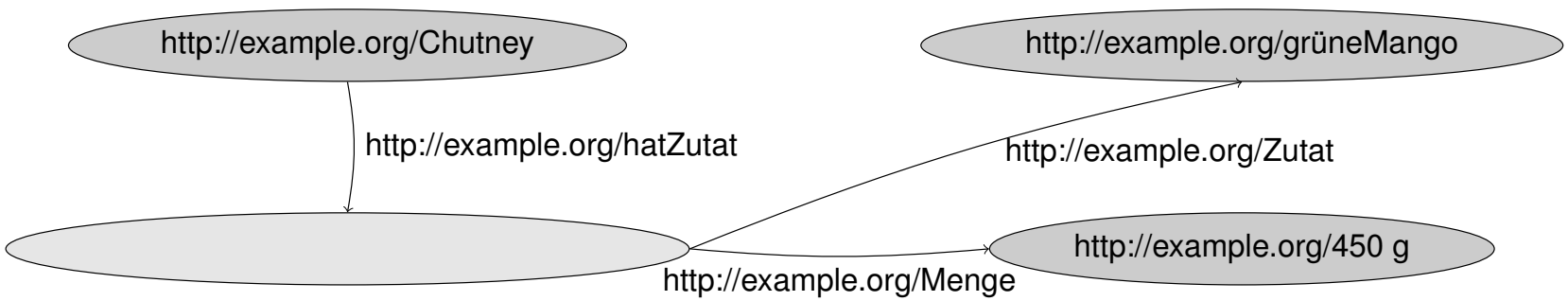
# Agenda

- XML – Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML
- Datentypen
- Mehrwertige Beziehungen
- Leere Knoten
- Listen

# Leere Knoten

Hilfsknoten in RDF:

- 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

## 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

## Turtle-Syntax:

```
@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

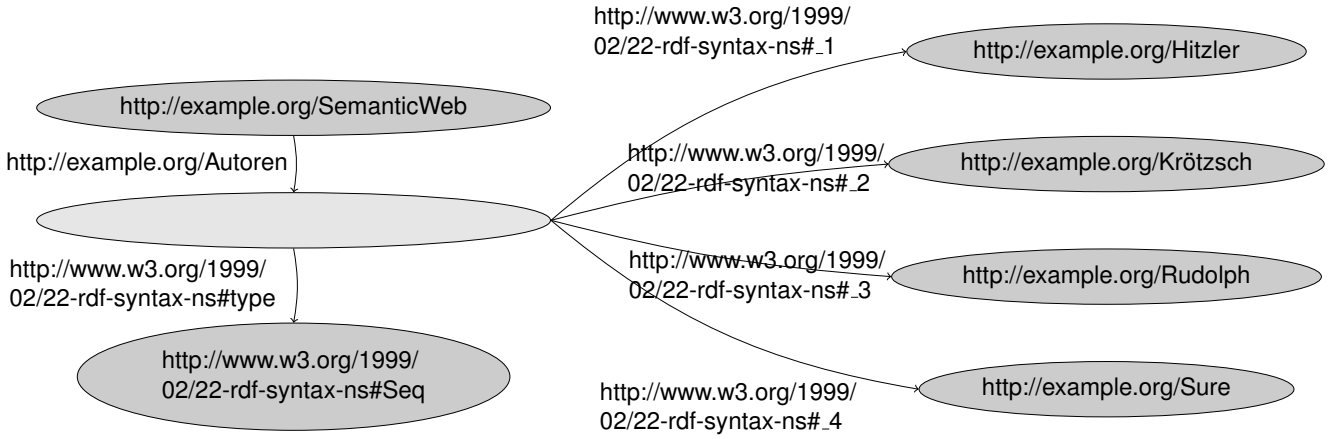
- XML – Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML
- Datentypen
- Mehrwertige Beziehungen
- Leere Knoten
- Listen

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

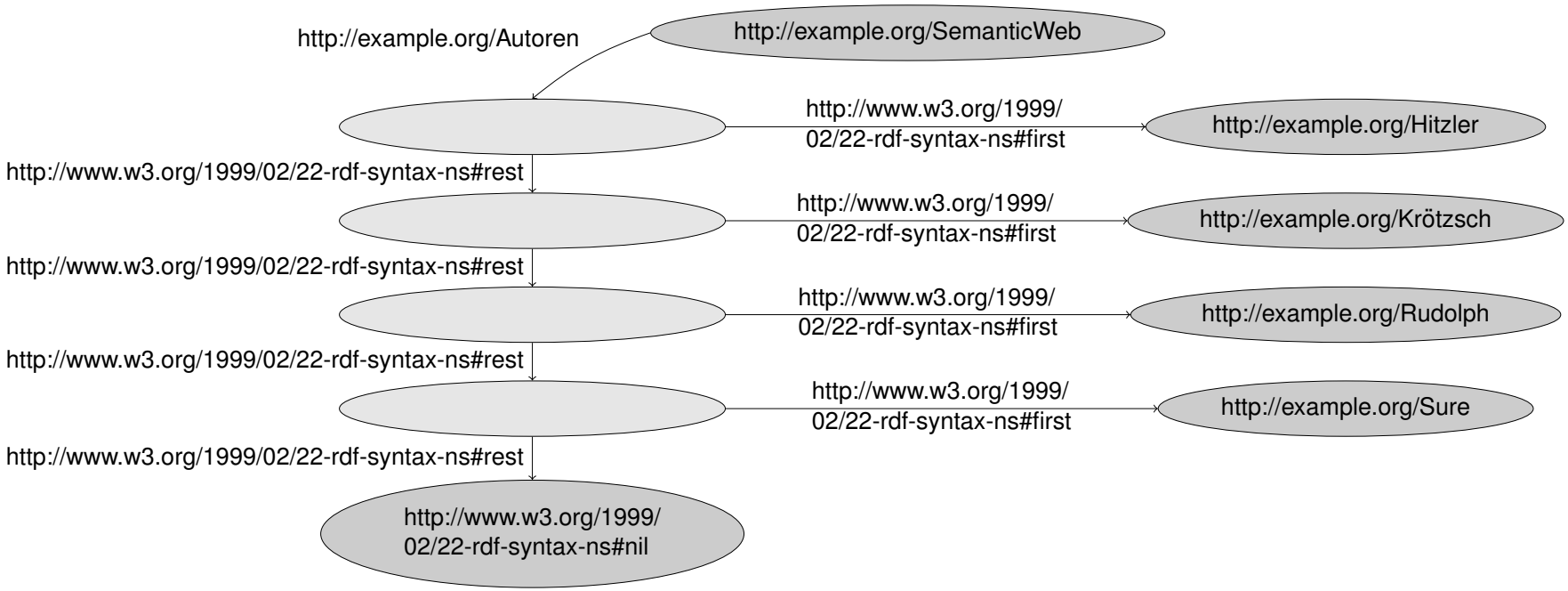
# Typen offener Listen

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

## Turtle

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

# Graph Definitionen

- Ein *RDF triple* besteht aus drei Komponenten:
  1. dem Subjekt, welches eine URI oder ein leerer Knoten ist,
  2. dem Prädikat, welches eine URI ist, und
  3. dem Object, welches eine URI, ein leerer Knoten oder ein Literal ist.
- Das Prädikat heisst auch Property.
- Ein R*RDF Graph* (oder einfach Graph) ist eine Menge von RDF Tripeln. Die Knoten des Graphen sind die Subjekte und Objekte des Tripel in dem Graph.
- Ein Teilgraph eines RDF Graphens ist eine Teilmenge der Tripel in dem Graph. Ein echter Teilgraph ist eine echte Teilmenge der Tripel des Graphen.
- Ein *ground* Graph ist ein RDF Graph ohne leere Knoten.

# Graph Definitionen

- Ein *Name* ist eine URI Referenz oder ein Literal
- Ein typisiertes Literal umfasst zwei Namen: das Literal selbst und seine Typenreferenz (URI)
- Eine Menge von Namen bezeichnet man als *Vokabular*
- Das *Vokabular eines Graphen* besteht aus der Menge der Namen die als Subjekt, Prädikat oder Objekt in einem der Tripel des Graphen auftauchen.
- Bemerkung: Die URI Referenzen die nun innerhalb der typisierten Literale auftauchen gehören nicht zum Vokabular des Graphen

# Graph Definitionen

- Sei $M$ ein Mapping von leeren Knoten zu einer Menge von Literalen, leeren Knoten und URI Referenzen. Wir bezeichen $M$ als *Instanz Mapping*.
- Jeder Graph $G'$ der durch Ersetzen von (einigen oder allen) leeren Knoten $\ell$ in $G$ durch $M(\ell)$ entstanden ist, ist eine *Instanz* von $G$.
- Eine Instanz bzgl. eines Vokabulars $V$ ist eine Instanz in der alle Namen die leere Knoten ersetzt haben aus dem Vokabular $V$ stammen.
- Eine *echte Instanz* eines Graphen ist eine Instanz in der ein leerer Knoten durch einen Namen ersetzt wurde oder in der zwei leere Knoten zu einem Knoten gemapped wurden.
- Graphen die sich nur in der Benennung ihrer leeren Knoten unterscheiden werden als äquivalent angesehen

# Graph Definitionen

- Ein RDF Graph ist lean (Deutsch: mager, geringhaltig) wenn der Graph keine Instanz hat, die ein echter Teilgraph des Graphen ist

$\rightsquigarrow$ Graphen, die nicht lean sind, haben interne Redundanz

```
{ex:a ex:p _:x . _:y ex:p _:x .}      (1)
{ex:a ex:p _:x . _:x ex:p _:x .}      (2)
```

- (1) ist nicht lean, aber (2) ist lean

# Graph Definitionen

Die Verschmelzung (merge) zweier RDF Graphen $G_1$ und $G_2$ ist definiert wie folgt:

- Wenn $G_1$ und $G_2$ keine leeren Knoten gemeinsam haben, ist die Verschmelzung $G_1 \cup G_2$
- Andernfalls ist die Verschmelzung von $G_1$ und $G_2$ die Vereinigung von $G_1'$ und $G_2'$, wobei $G_1'$ und $G_2'$ äquivalent zu $G_1$ und $G_2$ sind, aber keine leeren Knoten gemeinsam haben
- Zu dieser Variablenumbenennung sagt man auf Engl. “blank nodes have been are standardized apart”

# 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)

# Agenda

- XML – Motivation
- RDF-Datenmodell
- Syntax für RDF: Turtle und XML
- Datentypen
- Mehrwertige Beziehungen
- Leere Knoten
- Listen