ulm university universität ulm

# Linked Data & SPARQL

## Eine Einführung

Birte Glimm
Institut für Künstliche Intelligenz | 12. Mai 2012

# Agenda

## Resource Description Framework

## Linked Data

## SPARQL

- Komplexe Graph-Muster in SPARQL
- Filter in SPARQL
- Ausgabeformate in SPARQL
- Modifikatoren in SPARQL

# Agenda

## Resource Description Framework

## Linked Data

## SPARQL

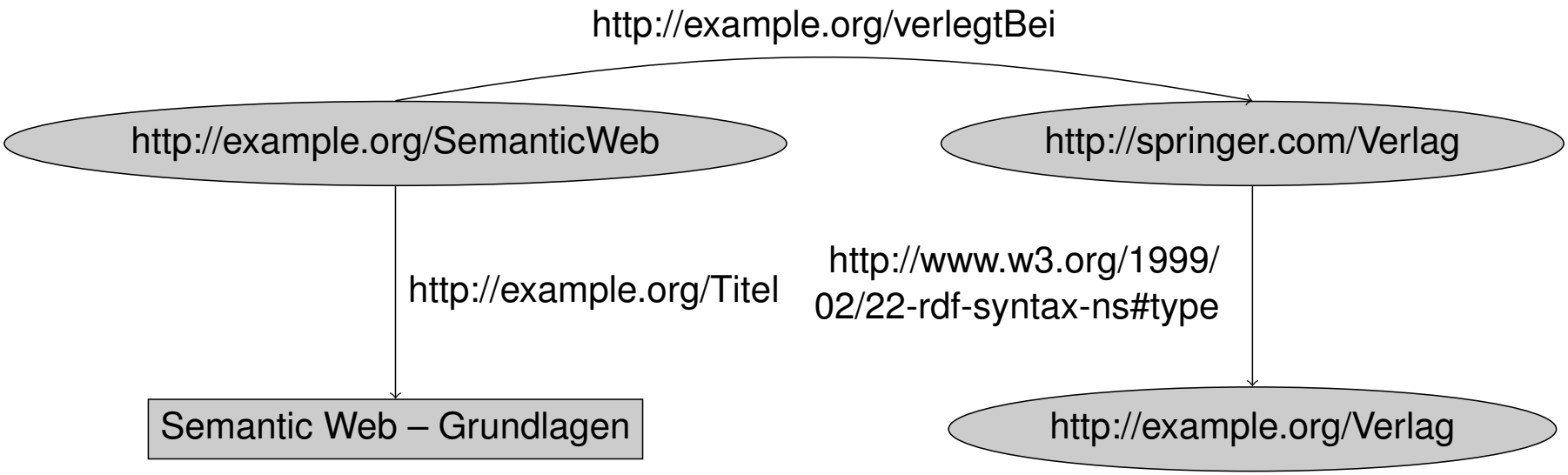
- Komplexe Graph-Muster in SPARQL
- Filter in SPARQL
- Ausgabeformate in SPARQL
- Modifikatoren in SPARQL

# 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

# Daten als Menge von Tripeln $\rightsquigarrow$ Graphdarstellung

- Verschiedene syntaktische Darstellungsformen



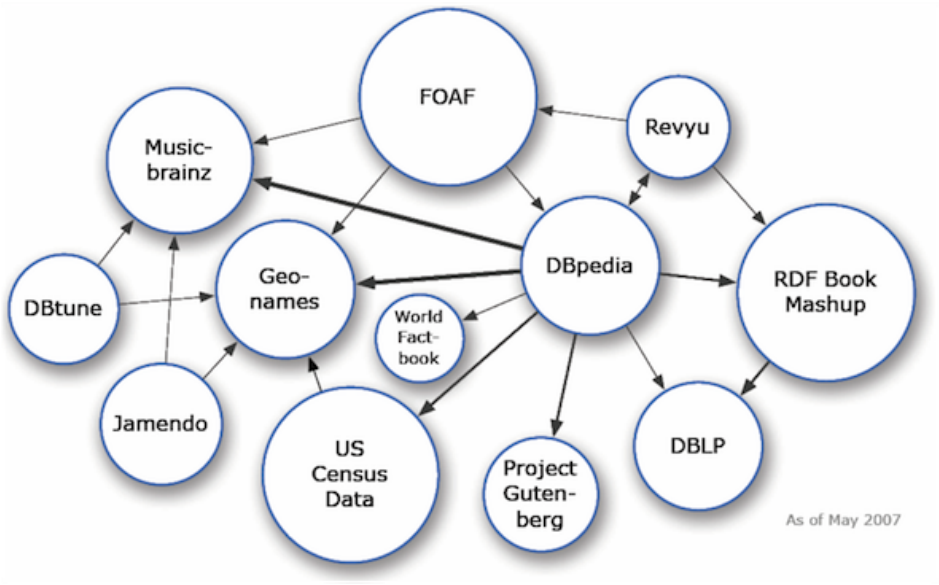
# Agenda

# Daten im Web

- Immer mehr Anbieter stellen nicht nur Webseiten (HTML) im Web zur Verfügung sondern (auch) Daten
- Dabei werden Semantic Web Standards verwendet (siehe Linking Open Data (LOD) Initiative)
  `http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData`
- Verwendet werden APIs, z.B. via JSON/REST, für den programmatischen Zugriff

- Semantic Web Technologien vereinfachen die Integration von Daten aus verschiedenen Quellen
- Die Kombination von Daten erlaubt tiefere Einblicke

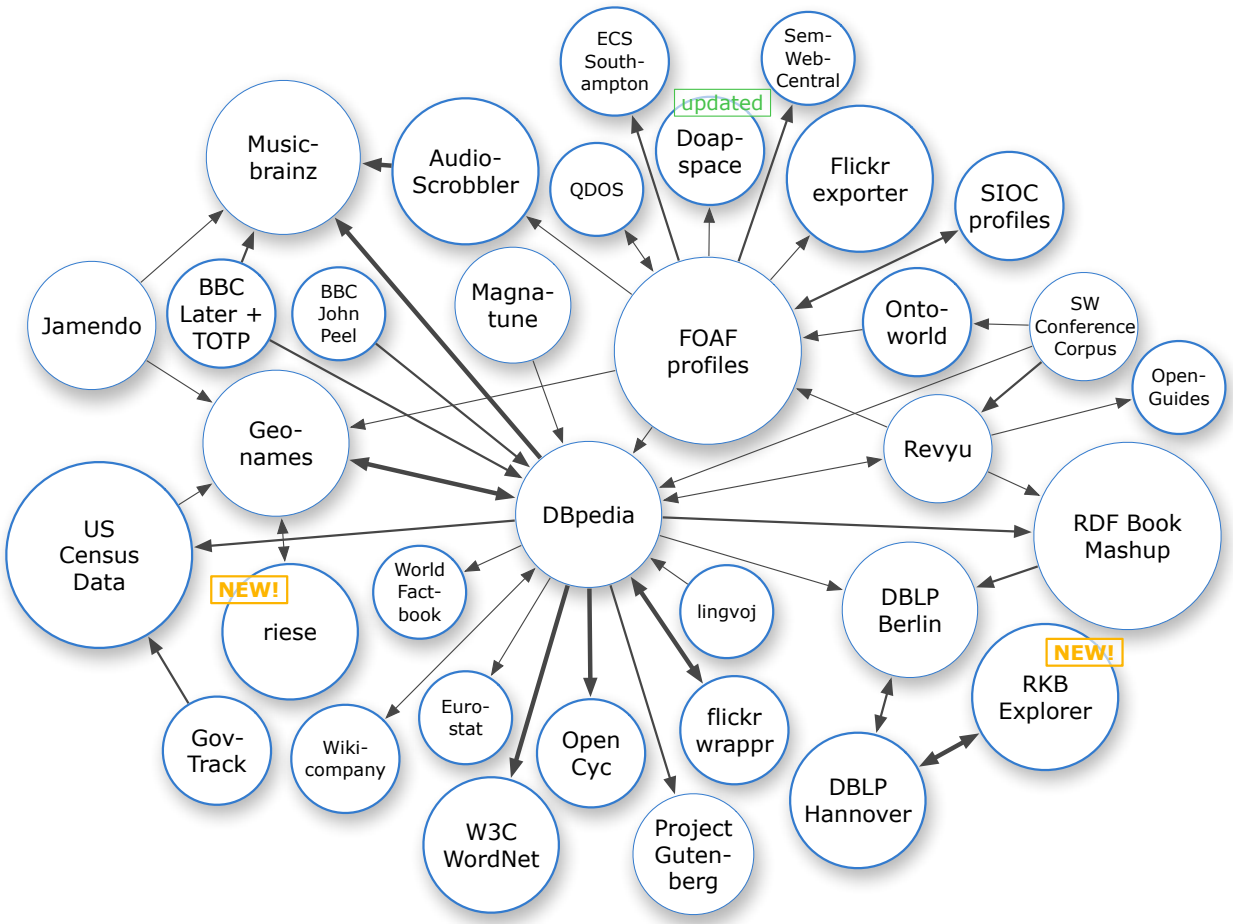
# Linked Data im Web 01.05.2007




Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.

`http://lod-cloud.net/`

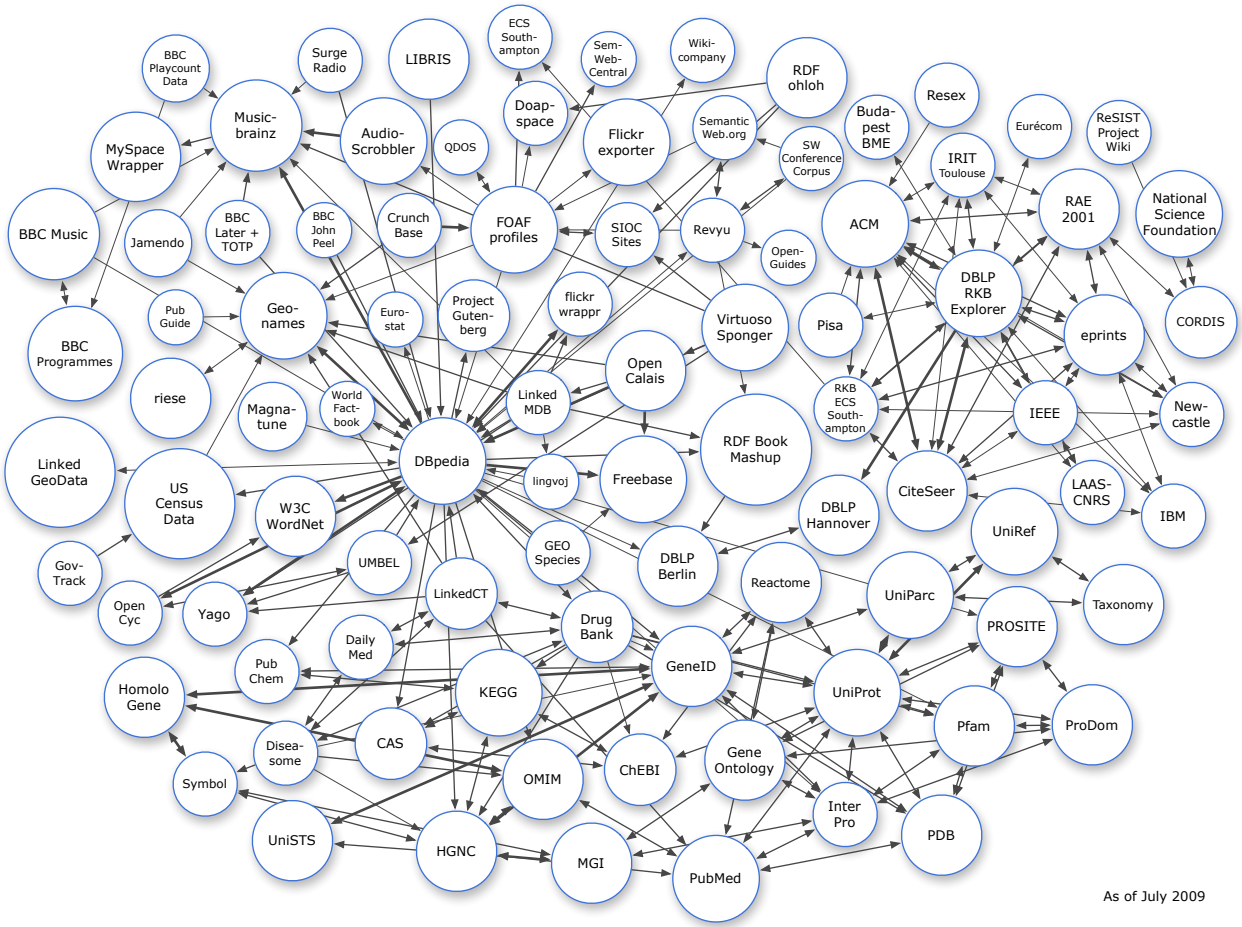
# Linked Data im Web 31.03.2008




Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.

`http://lod-cloud.net/`

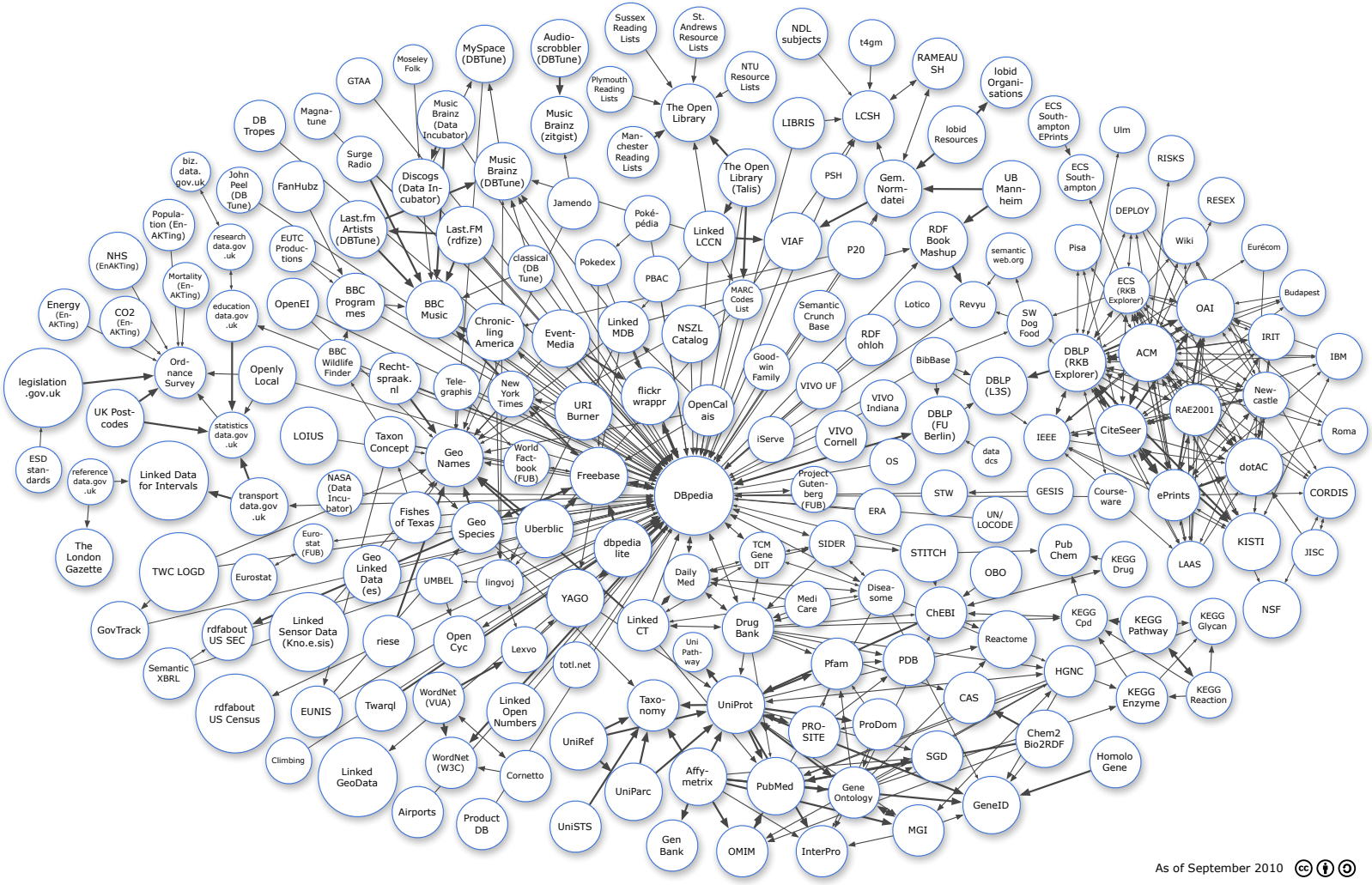
# Linked Data im Web 14.07.2009




Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.

`http://lod-cloud.net/`

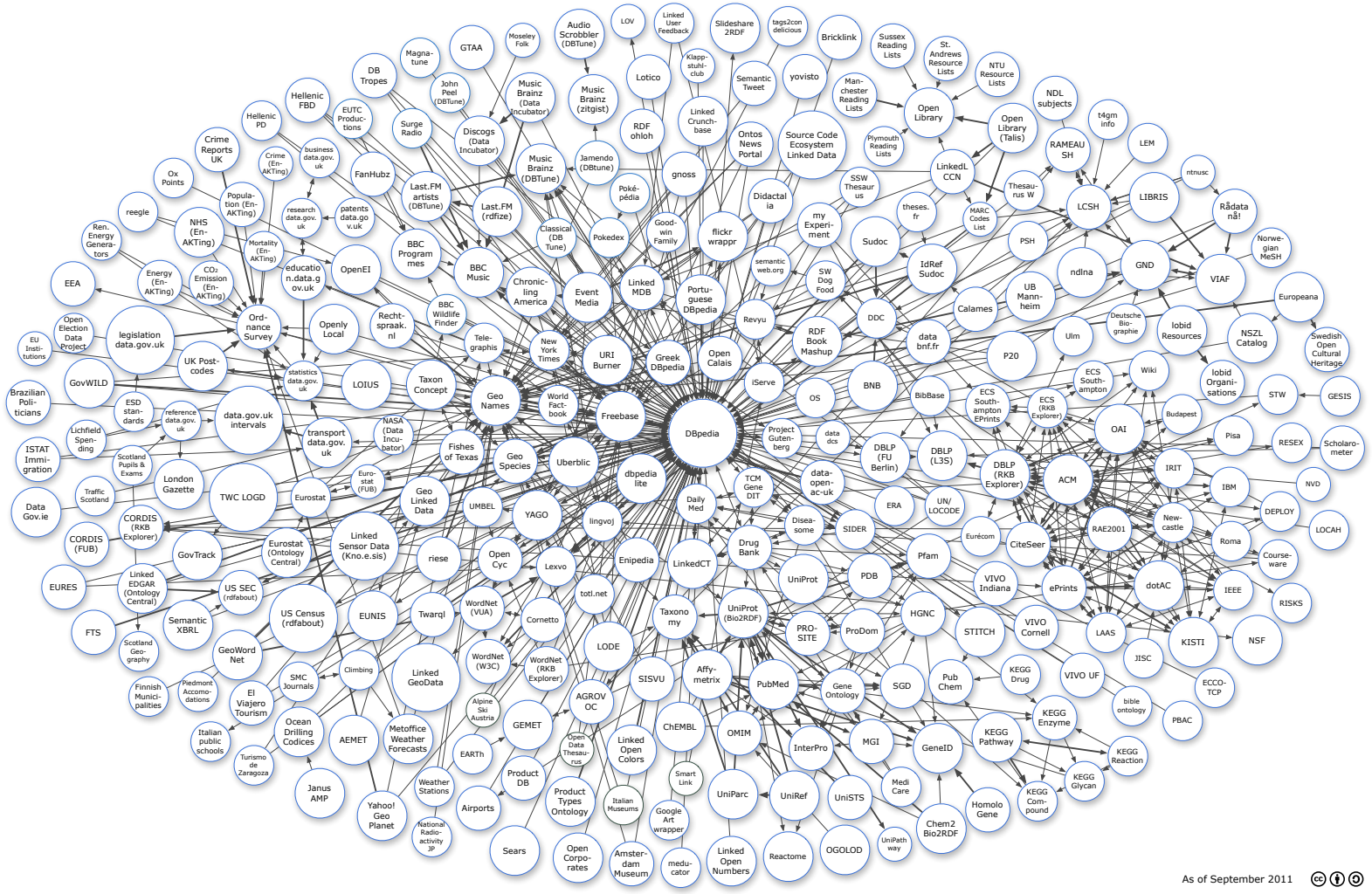
# Linked Data im Web 22.09.2010




Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.

`http://lod-cloud.net/`

# Linked Data im Web 19.09.2011




Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch.

`http://lod-cloud.net/`

# Semantic Web Technologien

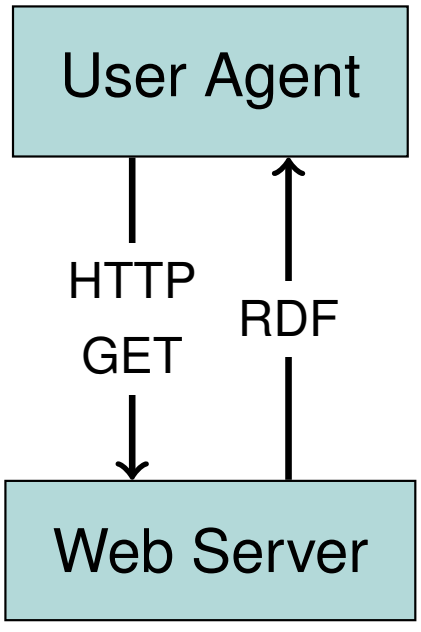
- Nützlich zum Publizieren, zum Austausch und zur Integration von Daten
- Semantic Web Technologien sind mittlerweile recht ausgereift
  - IRIs (IETF RFC 3987, 2005)
  - HTTP (IETF RFC 2616, 1999)
  - RDF (W3C Recommendation, 1999, Update in 2004)
  - RDFS (W3C Recommendation, 2004)
  - SPARQL (W3C Recommendation, 2008, Update im Moment)
  - OWL (W3C Recommendation, 2004, Update in 2009)
- Linked Data besteht aus einigen Prinzipien zum Publizieren von Daten im Web

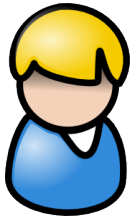
# Linked Data Principles*

1. Use URIs to name things; not only documents, but also people, locations, concepts, etc.
2. To enable agents (human users and machine agents alike) to look up those names, use HTTP URIs
3. When someone looks up a URI we provide useful information; with 'useful' in the strict sense we usually mean structured data in RDF.
4. Include links to other URIs allowing agents (machines and humans) to discover more things

*http://www.w3.org/DesignIssues/LinkedData.html

# Zusammenhang zwischen URI einer Sache und URI einer Quelle

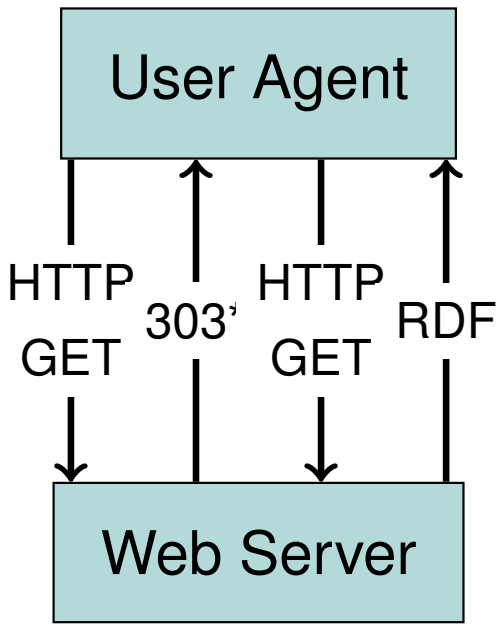


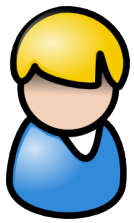



http://www.polleres.net/foaf.rdf#me


http://www.polleres.net/foaf.rdf

# Zusammenhang zwischen URI einer Sache und URI einer Quelle



http://dbpedia.org/resource/Gordon_Brown

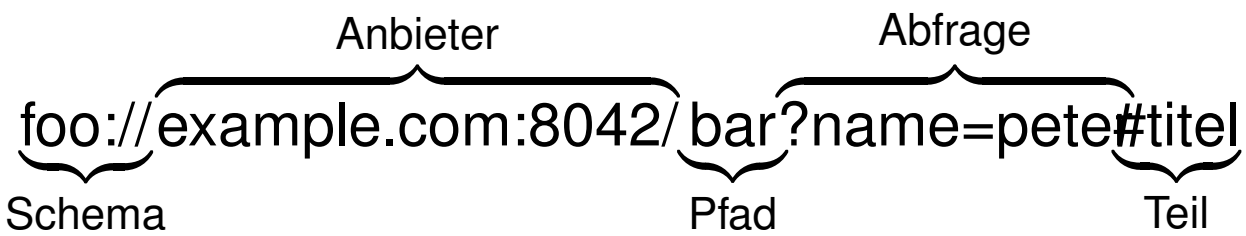

http://dbpedia.org/data/Gordon_Brown

http://dbpedia.org/page/Gordon_Brown

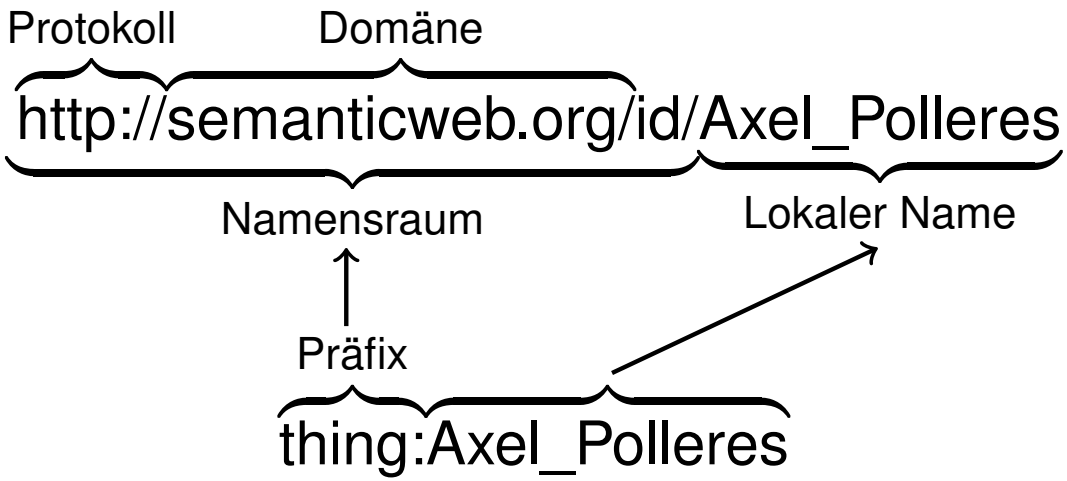
*HTTP Response Code 303: See Other

# Hintergrund: Uniform Resource Identifiers

- Ein Uniform Resource Identifier ist eine kompakte Sequenz von Charakteren, die eine abstrakte oder physikalische Ressource identifizieren [RFC3986]
- Syntax
  URI = Schema ":" [ Anbieter ] Pfad [ "?" Abfrage ] [ "#" Teil ]
- Beispiel



# URIs/IRIs




- URIs sind “Uniform Resource Identifiers”
  - IRIs sind Unicode-basierte “Internationalized Resource Identifiers”
- Jede URI identifiziert eine Entität
- Semantic Web URIs nutzen üblicherweise HTTP
  - HyperText Transfer Protocol
  - Können idealerweise aufgelöst werden, um weitere Daten zu erhalten
  - Linked Data

# HTTP Übersicht

- HTTP Nachrichten bestehen aus der Anfrage eines Clients an einen Server und die Antworten des Servers zum Client
- Bestimmte Methoden sind vordefiniert (z.B. GET, POST, etc.), aber weitere können definiert werden
- Eine Menge von Statuscodes ist definiert:
  - Informational 1xx, provisional response, (100 Continue)
  - Successful 2xx, request successfully received, understood, and accepted (201 Created)
  - Redirection 3xx, further action needs to be taken by user agent to fulfill the request (301 Moved Permanently)
  - Client Error 4xx, client erred (405 Method Not Allowed)
  - Server Error 5xx, server encountered an unexpected condition (501 Not Implemented)

# HTTP Content Negotiation

- Content Negotiation (CN, conneg) ist der Prozess der Selektion der besten Repräsentation für eine Anfrage wenn mehrere Repräsentationen verfügbar sind
- Drei Arten: server-driven, agent-driven, transparent

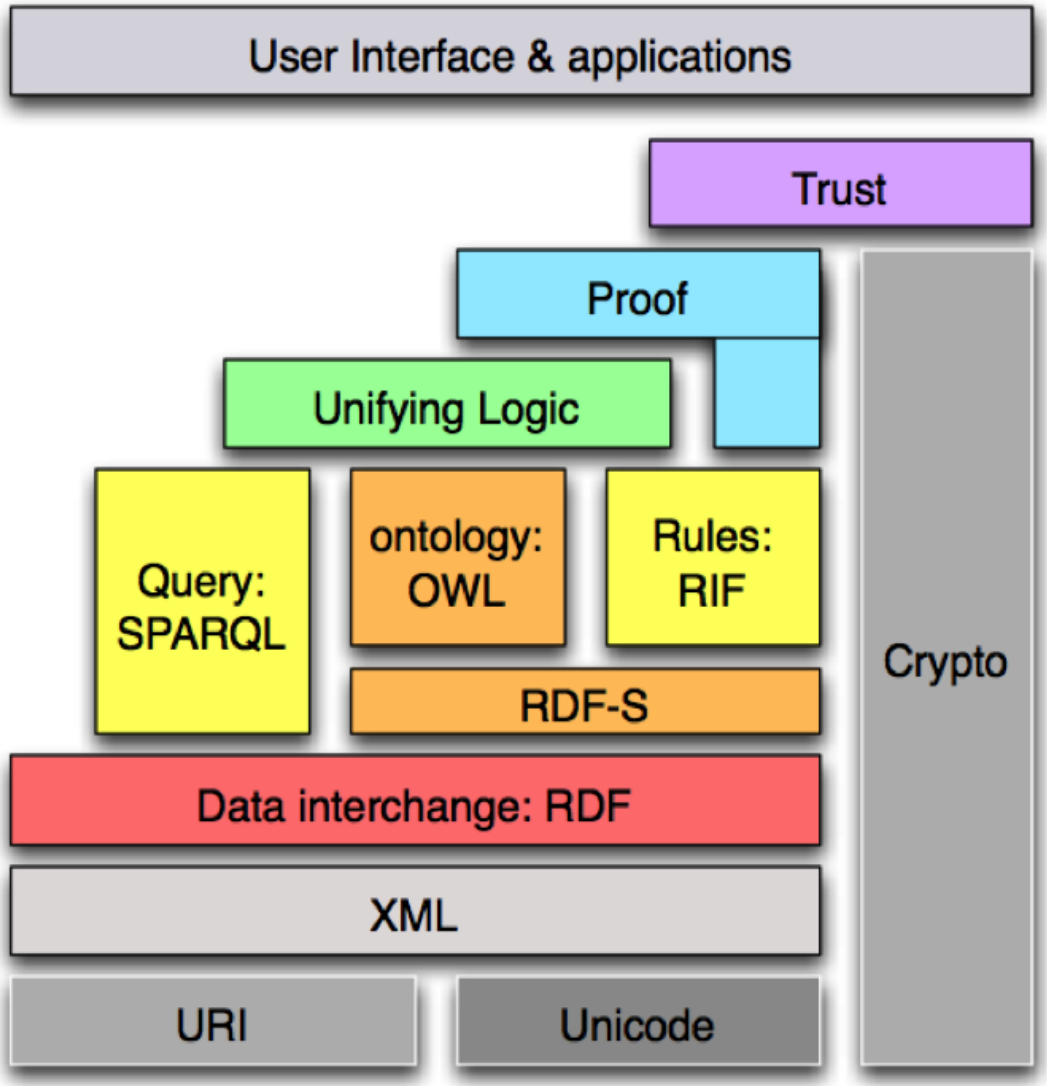
```
$ curl -H "Accept: application/rdf+xml"
    http://dbpedia.org/resource/Galway

HTTP/1.1 303 See Other
Content-Type: application/rdf+xml
Location: http://dbpedia.org/data/Galway.rdf
$
```

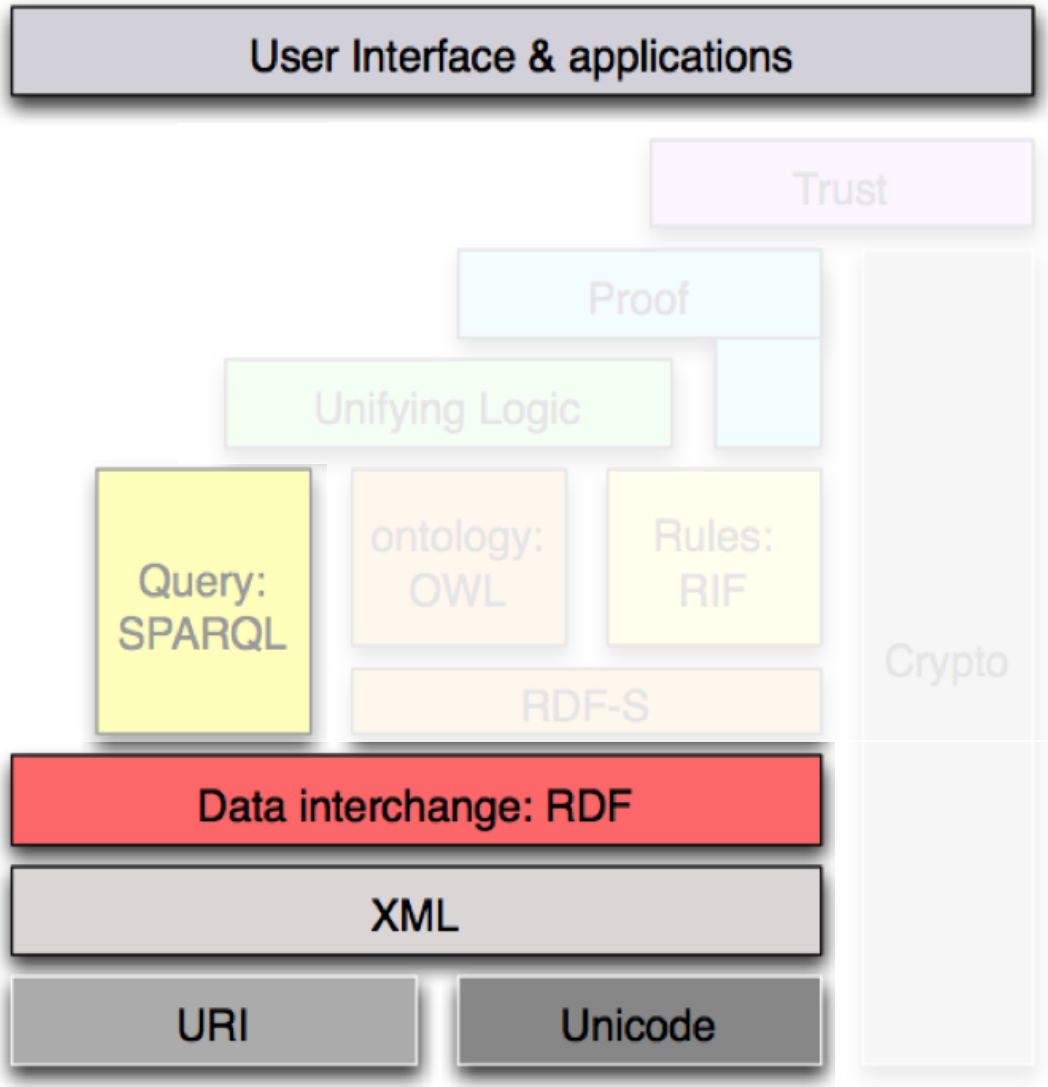
`curl` – Tool um Daten zu einem Server zu schicken oder von einem Server zu empfangen

`-H` bedeutet nur HTTP/HTTPS

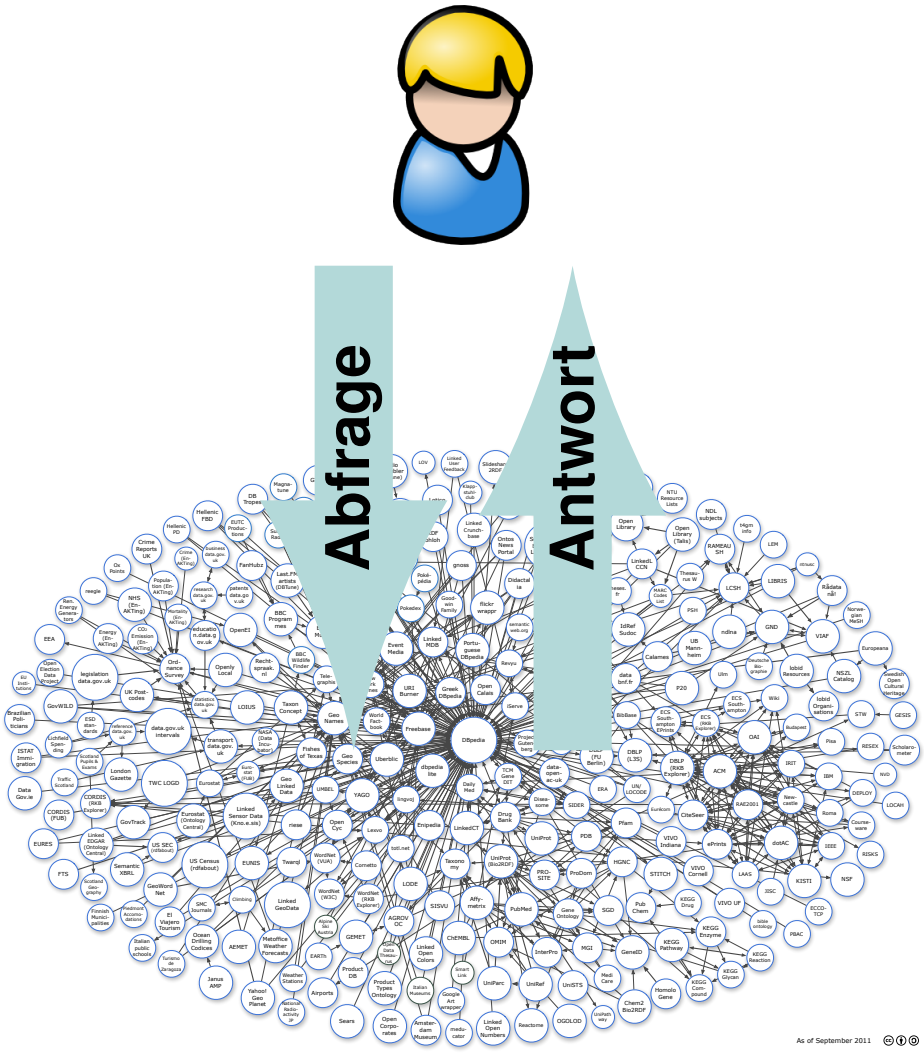
# Semantic Web Application Architecture



# Semantic Web Application Architecture

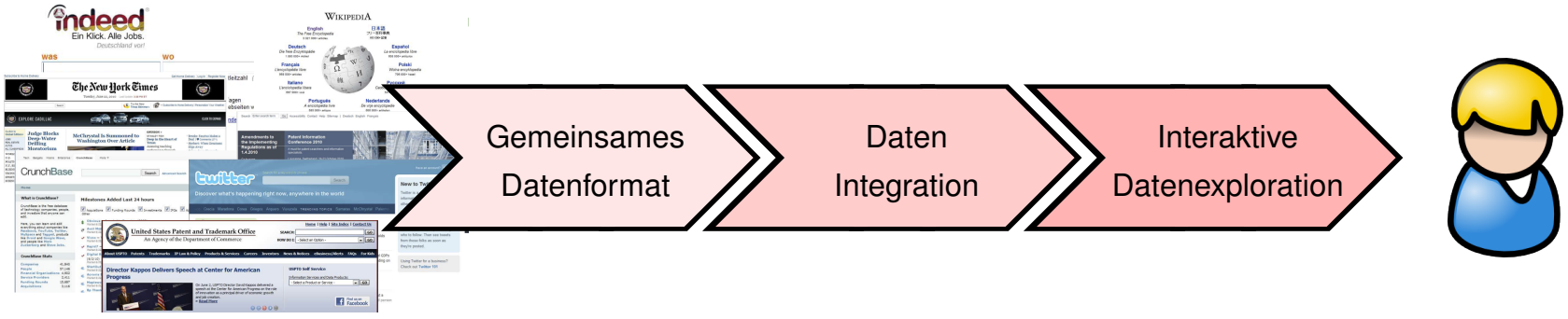


# Linked Data Anwendungen: Minimale Architektur



# Szenario

- Typisches Datenintegrationsszenario




- Anfrage: Welche Jobangebote gibt es von Konkurrenten von Facebook?

# Agenda

Resource Description Framework

Linked Data

SPARQL

- Komplexe Graph-Muster in SPARQL
- Filter in SPARQL
- Ausgabeformate in SPARQL
- Modifikatoren in SPARQL

# SPARQL

SPARQL (sprich engl. *sparkle*) steht für
**SPARQL Protocol And RDF Query Language**

- W3C Spezifikation seit 2008
- Zur Zeit Erweiterung auf SPARQL 1.1
- Anfragsprache zur *Abfrage von Instanzen aus RDF-Dokumenten*
- Große praktische Bedeutung

## Teile der SPARQL 1.0 Spezifikation

- Anfragesprache: Thema dieser Vorlesung
- Ergebnisformat: Darstellung von Ergebnissen in XML
- Anfrageprotokoll: Übermittlung von Anfragen und Ergebnissen

# Teile der SPARQL 1.1 Spezifikation

- Query: Erweitert die Sprachkonstrukte für SPARQL Abfragen
- Update: zur Modifikation von RDF Graphen (Hinzufügen, Löschen)
- Graph Store HTTP Protocol: HTTP Operationen um eine Menge von Graphen zu verwalten
- Entailment Regimes: Abfragen auch von impliziten Konsequenzen
- Service Description: Methode zum Beschreiben von SPARQL Endpunkten
- Federation Extensions: Ausführung von verteilten Abfragen
- Query Results JSON Format: Abfrageergebnisse in JSON
- Query Results CSV, TSV Format: Komma und Tab separierte Abfrageergebnisse

# Einfache Anfragen

Eine einfache Beispielanfrage:

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

# Einfache Anfragen

Eine einfache Beispielanfrage:

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

- Die Bedingung der WHERE Klausel heisst *query pattern/Abfragemuster*

# Einfache Anfragen

Eine einfache Beispielanfrage:

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

- Die Bedingung der WHERE Klausel heisst *query pattern/Abfragemuster*
- Tripel mit Variablen heissen basic graph pattern (BGP)
  - ⇝ BGPs verwenden Turtle Syntax für RDF
  - ⇝ BGPs können Variablen (`?variable` oder `$variable`) enthalten

# Einfache Anfragen

Eine einfache Beispielanfrage:

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

- Die Bedingung der WHERE Klausel heisst *query pattern/Abfragemuster*
- Tripel mit Variablen heissen basic graph pattern (BGP)
  - ⇝ BGPs verwenden Turtle Syntax für RDF
  - ⇝ BGPs können Variablen (`?variable` oder `$variable`) enthalten
- Abgekürzte IRIs sind möglich (`PREFIX`)

# Einfache Anfragen

Eine einfache Beispielanfrage:

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

- Die Bedingung der WHERE Klausel heisst *query pattern/Abfragemuster*
- Tripel mit Variablen heissen basic graph pattern (BGP)
  - ⇝ BGPs verwenden Turtle Syntax für RDF
  - ⇝ BGPs können Variablen (`?variable` oder `$variable`) enthalten
- Abgekürzte IRIs sind möglich (`PREFIX`)
- Abfrageergebnis für die selektierten Variablen (`SELECT`)

# Beispielergebnis

BGP: `{?x foaf:name ?name. ?x foaf:mbox ?mbox}`

```
@prefix foaf: http://xmlns.com/foaf/0.1/ .
_:a foaf:name "Birte Glimm" ;
    foaf:mbox "b.glimm@googlemail.com" ;
    foaf:icqChatID "b.glimm" ;
    foaf:aimChatID "b.glimm" .
_:b foaf:name "Sebastian Rudolph" ;
    foaf:mbox <mailto:rudolph@kit.edu> .
_:c foaf:name "Pascal Hitzler" ;
    foaf:aimChatID "phi" .
foaf:icqChatID rdfs:subPropertyOf foaf:nick .
foaf:name rdfs:domain foaf:Person .
```

BGP Matching Ergebnis (Tabelle mit einer Zeile je Ergebnis):

| x | name | mbox |
|---|---|---|
| _:a | "Birte Glimm" | "b.glimm@googlemail.com" |
| _:b | "Sebastian Rudolph" | <mailto:rudolph@kit.edu> |

# Beispielergebnis

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

## BGP Matching Ergebnis:

| x | name | mbox |
|---|---|---|
| _:a | "Birte Glimm" | "b.glimm@googlemail.com" |
| _:b | "Sebastian Rudolph" | <mailto:rudolph@kit.edu> |

## Abfrageergebnis:

| name | mbox |
|---|---|
| "Birte Glimm" | "b.glimm@googlemail.com" |
| "Sebastian Rudolph" | <mailto:rudolph@kit.edu> |

# Einfache Graph-Muster

Die grundlegenden Anfragemuster sind einfache Graph-Muster oder basic graph patterns (BGPs)

- Menge von RDF-Tripeln in Turtle-Syntax
- Turtle-Abkürzungen (mittels , und ;) zulässig
- Variablen werden durch ? oder $ gekennzeichnet (`?variable` hat gleiche Bedeutung wie `$variable`)
- Variablen zulässig als Subjekt, Prädikat oder Objekt

# Einfache Graph-Muster

Die grundlegenden Anfragemuster sind einfache Graph-Muster oder basic graph patterns (BGPs)

- Menge von RDF-Tripeln in Turtle-Syntax
- Turtle-Abkürzungen (mittels , und ;) zulässig
- Variablen werden durch ? oder $ gekennzeichnet (`?variable` hat gleiche Bedeutung wie `$variable`)
- Variablen zulässig als Subjekt, Prädikat oder Objekt

Zulässig $\neq$ lesbar:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?rf456df ?ac66sB
WHERE { ?h4dF8Q foaf:name ?rf456df .
        ?h4dF8Q foaf:mbox ?ac66sB }
```

(semantisch äquivalent zur vorherigen Anfrage)

# Was bedeuten leere Knoten in SPARQL?

Leere Knoten in Anfragemustern:

- Zulässig als Subjekt oder Objekt
- ID beliebig, aber niemals gleiche ID mehrfach pro Anfrage
- Verhalten sich wie Variablen, die nicht ausgewählt werden können

# Was bedeuten leere Knoten in SPARQL?

Leere Knoten in Anfragemustern:

- Zulässig als Subjekt oder Objekt
- ID beliebig, aber niemals gleiche ID mehrfach pro Anfrage
- Verhalten sich wie Variablen, die nicht ausgewählt werden können

Leere Knoten in Ergebnissen:

- Platzhalter für unbekannte Elemente
- IDs beliebig, aber eventuell an andere Ergebnisteile gebunden:

| subj | Wert |
|---|---|
| _:a | "zum" |
| _:b | "Beispiel" |

| subj | Wert |
|---|---|
| _:y | "zum" |
| _:g | "Beispiel" |

| subj | Wert |
|---|---|
| _:z | "zum" |
| _:z | "Beispiel" |

# Datasets und FROM (NAMED)

- Keine FROM Klausel notwendug
- Jeder SPARQL Service spezifiziert ein Dataset bestehend aus einem Default Graphen und keinem oder mehr benannten Graphen (named graphs)

Keine FROM Klausel

⇝ Auswertung bzgl. des Default Graphen

FROM NAMED in Kombination mit dem GRAPH Schlüsselwort

⇝ Auswertung über den benannten Graphen

FROM Klausel

⇝ Erzeugung eines neuen Default Graphen für die Abfrage

```
SELECT ?name ?mbox
FROM NAMED <http://ex.org/a> <http://ex.org/b>
WHERE { GRAPH ?g
          { ?x foaf:name ?name. ?x foaf:mbox ?mbox }
        }
```

# Datentypen

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex:  <http://example.org/> .
ex:ex1  ex:p  "test" .
ex:ex2  ex:p  "test"^^xsd:string .
ex:ex3  ex:p  "test"@en .
ex:ex4  ex:p  "42"^^xsd:integer .
```

## Was liefert eine Anfrage mit folgendem Muster?

```
{ ?subject <http://example.org/p> "test" . }
```

# Datentypen

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex:  <http://example.org/> .
ex:ex1  ex:p  "test" .
ex:ex2  ex:p  "test"^^xsd:string .
ex:ex3  ex:p  "test"@en .
ex:ex4  ex:p  "42"^^xsd:integer .
```

## Was liefert eine Anfrage mit folgendem Muster?

```
{ ?subject <http://example.org/p> "test" . }
```

⇝ `ex:ex1` als einziges Ergebnis
⇝ genaue Übereinstimmung der Datentypen gefordert

## Aber: Abkürzung für Zahlenwerte möglich

```
{ ?subject <http://example.org/p> 42 . }
```

⇝ Datentyp wird aus syntaktischer Form bestimmt:
`xsd:integer` (42), `xsd:decimal` (42.2), `xsd:double` (1.0e6)

# Gruppierende Graph-Muster

Einfache Graph-Muster können durch {...} gruppiert werden.

Beispiel:

```
PREFIX ex:  <http://example.org/>
SELECT ?titel ?author
WHERE
{ { ?buch ex:publishedBy <http://springer.com/Verlag>.
    ?buch ex:Titel ?titel . }
  { }
  ?buch ex:Author ?author .
}
```

$\rightsquigarrow$ Sinnvoll erst bei Verwendung zusätzlicher Konstruktoren

# Optionale Muster

Das Schlüsselwort `OPTIONAL` erlaubt die Angabe optionaler Teile eines Musters.

Beispiel:

```
{ ?book ex:publishedBy <http://springer.com/Verlag> .
   OPTIONAL { ?book ex:Titel ?titel . }
   OPTIONAL { ?book ex:Author ?author . }
}
```

# Optionale Muster

Das Schlüsselwort `OPTIONAL` erlaubt die Angabe optionaler Teile eines Musters.

Beispiel:

```
{ ?book ex:publishedBy <http://springer.com/Verlag> .
   OPTIONAL { ?book ex:Titel ?titel . }
   OPTIONAL { ?book ex:Author ?author . }
}
```

$\rightsquigarrow$ Teile eines Anfrageergebnisses können ungebunden sein:

| book | titel | author |
|---|---|---|
| <http://ex.org/book1> | "Titel1" | <http://ex.org/author1> |
| <http://ex.org/book2> | "Titel2" | |
| <http://ex.org/book3> | "Titel3" | _:a |
| <http://ex.org/book4> | | _:a |
| <http://ex.org/book5> | | |

# Alternative Muster

Das Schlüsselwort `UNION` erlaubt die Angabe alternativer Teile eines Musters.

Example:

```
{ ?book ex:publishedBy <http://springer.com/Verlag> .
  { ?book ex:Author ?author . } UNION
  { ?book ex:Editor ?author . }
}
```

$\rightsquigarrow$ Ergebnis entspricht Vereinigung der Ergebnisse mit einer der beiden Bedingungen

Anm.: Gleiche Variablennamen in beiden Teilen von `UNION` beeinflussen sich nicht

# Kombination von Optionen und Alternativen (1)

Wie sind Kombinationen von `OPTIONAL` und `UNION` zu verstehen?

```
{ ?book ex:publishedBy  <http://springer.com/Verlag> .
  { ?book ex:Author ?author . } UNION
  { ?book ex:Editor ?author . } OPTIONAL
  { ?author ex:Surname ?name . } }
```

- Vereinigung zweier Muster mit angefügtem optionalem Muster oder
- Vereinigung zweier Muster, wobei das zweite einen optionalen Teil hat?

# Kombination von Optionen und Alternativen (1)

Wie sind Kombinationen von `OPTIONAL` und `UNION` zu verstehen?

```
{ ?book ex:publishedBy  <http://springer.com/Verlag> .
  { ?book ex:Author ?author . } UNION
  { ?book ex:Editor ?author . } OPTIONAL
  { ?author ex:Surname ?name . } }
```

- Vereinigung zweier Muster mit angefügtem optionalem Muster oder
- Vereinigung zweier Muster, wobei das zweite einen optionalen Teil hat?

$\rightsquigarrow$ Erste Interpretation korrekt:

```
{ ?book ex:publishedBy <http://springer.com/Verlag> .
  { { ?book ex:Author ?author . } UNION
    { ?book ex:Editor   ?author . }
  } OPTIONAL { ?author ex:Surname ?name . } }
```

# Kombination von Optionen und Alternativen (2)

## Allgemeine Regeln

- `OPTIONAL` bezieht sich immer auf genau ein gruppierendes Muster rechts davon.
- `OPTIONAL` und `UNION` sind gleichwertig und beziehen sich auf jeweils alle links davon stehenden Ausdrücke (*linksassoziativ*)

# Wozu Filter?

Viele Anfragen sind auch mit komplexen Graph-Mustern nicht möglich:

- „Welche Personen sind zwischen 18 und 23 Jahre alt?“
- „Der Nachname welcher Personen enthält einen Bindestrich?“
- „Welche Texte in deutscher Sprache sind in der Ontologie angegeben?“

⇝ Filter als allgemeiner Mechanismus für solche Ausdrucksmittel

# Filter in SPARQL

## Beispiel:

```
PREFIX ex: <http://example.org/>
SELECT ?book WHERE
  { ?book ex:publishedBy <http://springer.com/Verlag> .
    ?book ex:Price ?price
    FILTER (?price < 35)
  }
```

- Schlüsselwort `FILTER`, gefolgt von Filterausdruck in Klammern
- Filterbedingungen liefern Wahrheitswerte (und ev. auch Fehler)
- Viele Filterfunktionen nicht durch RDF spezifiziert
  $\rightsquigarrow$ Funktionen teils aus XQuery/XPath-Standard für XML übernommen

# Filterfunktionen: Vergleiche und Arithmetik

Vergleichoperatoren: `<`, `=`, `>`, `<=`, `>=`, `!=`

- Vergleich von Datenliteralen gemäß der jeweils *natürlichen* Reihenfolge
- Unterstützung für numerische Datentypen, `xsd:dateTime`, `xsd:string` (alphabetische Ordnung), `xsd:Boolean` (1 > 0)
- für andere Typen und sonstige RDF-Elemente nur `=` und `!=` verfügbar
- kein Vergleich von Literalen inkompatibler Typen (z.B. `xsd:string` und `xsd:integer`)

# Filterfunktionen: Vergleiche und Arithmetik

Vergleichoperatoren: `<`, `=`, `>`, `<=`, `>=`, `!=`

- Vergleich von Datenliteralen gemäß der jeweils *natürlichen* Reihenfolge
- Unterstützung für numerische Datentypen, `xsd:dateTime`, `xsd:string` (alphabetische Ordnung), `xsd:Boolean` (1 > 0)
- für andere Typen und sonstige RDF-Elemente nur `=` und `!=` verfügbar
- kein Vergleich von Literalen inkompatibler Typen (z.B. `xsd:string` und `xsd:integer`)

Arithmetische Operatoren: `+`, `-`, `*`, `/`

- Unterstützung für numerische Datentypen
- Verwendung zur Kombination von Werten in Filterbedingungen

Bsp.: `FILTER( ?gewicht/(?groesse * ?groesse) >= 25 )`

# Filterfunktionen: Spezialfunktionen für RDF

SPARQL unterstützt auch RDF-spezifische Filterfunktionen:

| | |
|---|---|
| `BOUND(A)` | `true` falls `A` eine gebundene Variable ist |
| `isURI(A)` | `true` falls `A` eine URI ist |
| `isBLANK(A)` | `true` falls `A` ein leerer Knoten ist |
| `isLITERAL(A)` | `true` falls `A` ein RDF-Literal ist |
| `STR(A)` | lexikalische Darstellung (`xsd:string`) von RDF-Literalen oder URIs |
| `LANG(A)` | Sprachcode eines RDF-Literals (`xsd:string`) oder leerer String falls kein Sprachcode |
| `DATATYPE(A)` | Datentyp-URI eines RDF-Literals (`xsd:string` bei ungetypten Literalen ohne Sprachangabe) |

# Ausgabeformatierung mit SELECT

Bisher waren alle Ergebnisse Tabellen: Ausgabeformat `SELECT`

Syntax: `SELECT <Variablenliste>` oder `SELECT *`

## Vorteil

Einfache sequentielle Abarbeitung von Ergebnissen

## Nachteil

Struktur/Beziehungen der Objekte im Ergebnis nicht offensichtlich

# Ausgabeformatierung mit CONSTRUCT

Kodierung von Ergebnissen in RDF-Graphen: Ausgabeformat `CONSTRUCT`

Syntax: `CONSTRUCT <RDF-Schablone in Turtle>`

```
PREFIX ex: <http://example.org/>
CONSTRUCT { ?person  ex:mailbox  ?email .
            ?person  ex:telefon  ?telefon . }
WHERE { ?person  ex:email   ?email .
        ?person  ex:tel     ?telefon . }
```

# Ausgabeformatierung mit CONSTRUCT

Kodierung von Ergebnissen in RDF-Graphen: Ausgabeformat `CONSTRUCT`

Syntax: `CONSTRUCT <RDF-Schablone in Turtle>`

```
PREFIX ex: <http://example.org/>
CONSTRUCT { ?person  ex:mailbox  ?email .
            ?person  ex:telefon  ?telefon . }
WHERE { ?person  ex:email   ?email .
        ?person  ex:tel     ?telefon . }
```

## Vorteil

Stukturiertes Ergebnis mit Beziehungen zwischen Elementen

## Nachteile

- Sequentielle Abarbeitung von Ergebnissen erschwert
- Keine Behandlung von ungebundenen Variablen

# Weitere Formate: ASK und DESCRIBE

SPARQL unterstützt zwei weitere Ausgabeformate:

- `ASK` prüft nur, ob es Ergebnisse gibt, keine Parameter
- `DESCRIBE` (informativ) liefert zu jeder gefundenen URI eine RDF-Beschreibung (anwendungsabhängig)

```
DESCRIBE ?x WHERE { ?x <http://ex.org/employeeId> "1234" }
```

Kann als Ergebnis z.B. die folgende Ausgabe haben (ohne Prefix Deklarationen):

```
_:a       exOrg:employeeId    "1234" ;
          foaf:mbox_sha1sum   "ABCD1234" ;
          vcard:N
          [ vcard:Family       "Smith" ;
            vcard:Given        "John"  ] .
foaf:mbox_sha1sum  rdf:type  owl:InverseFunctionalProperty .
```

# Wozu Modifikatoren?

Bisher nur grundsätzliche Formatierungseinstellungen für Ergebnisse:

- Wie kann man definierte Teile der Ergebnismenge abfragen?
- Wie werden Ergebnisse geordnet?
- Können wiederholte Ergebniszeilen sofort entfernt werden?

$\rightsquigarrow$ Modifikatoren der Lösungssequenz (*solution sequence modifiers*)

# Ergebnisse Sortieren

## Sortierung von Ergebnissen mit Schlüsselwort `ORDER BY`

```
SELECT ?book ?price
WHERE { ?book <http://example.org/Price> ?price . }
ORDER BY ?price
```

# Ergebnisse Sortieren

## Sortierung von Ergebnissen mit Schlüsselwort `ORDER BY`

```
SELECT ?book ?price
WHERE { ?book <http://example.org/Price> ?price . }
ORDER BY ?price
```

- Sortierung wie bei Filter-Vergleichoperatoren,
- Sortierung von URIs alphabetisch als Zeichenketten
- Reihenfolge zwischen unterschiedlichen Arten von Elementen:
  Ungebundene Variable < leere Knoten < URIs < RDF-Literale
- Nicht jede Möglichkeit durch Spezifikation definiert

# Ergebnisse sortieren

Weitere mögliche Angaben:

- `ORDER BY DESC(?price)`: absteigend
- `ORDER BY ASC(?price)`: aufsteigend, Voreinstellung
- `ORDER BY DESC(?price), ?titel`: hierarchische Ordnungskriterien

# LIMIT, OFFSET und DISTINCT

Einschränkung der Ergebnismenge:

- `LIMIT`: maximale Anzahl von Ergebnissen (Tabellenzeilen)
- `OFFSET`: Position des ersten gelieferten Ergebnisses
- `SELECT DISTINCT`: Entfernung von doppelten Tabellenzeilen

```
SELECT DISTINCT ?book ?price
WHERE { ?book <http://example.org/Price> ?price . }
ORDER BY ?price  LIMIT 5  OFFSET 25
```

$\rightsquigarrow$ `LIMIT` und `OFFSET` nur mit `ORDER BY` sinnvoll!

# SPARQL 1.1 Erweiterungen

SPARQL 1.1 vorerst ausgelassen

- Aggregate
- Subqueries
- Negation
- Ausdrücke in der SELECT Klausel
- Property Paths
- Assignment/Zuweisung
- CONSTRUCT Kurzform
- Weitere Funktionen und Operatoren

# Zusammenfassung

- Die Menge an verfügbaren strukturierten Daten im Web wächst ständig
- Semantik wird gebraucht, um Daten aus verschiedenen Quellen zu integrieren
- Abfrage und Visualisierung von Daten in Kombination möglich
- SPARQL zur Abfrage der Daten kennengelernt
  - Grundlegende Strukturen (Prefixe, Muster)
  - Ausgabeformate
  - Einfache und komplexe Muster (Alternativen, Optionale Teile, Gruppen)
  - Filter
  - Modifikatoren
- Semantik definiert durch Übersetzung in SPARQL Algebra