

Big Data

- **Top-k / Ranking / Skyline**
- **Semantic Web: RDF**
- **Information Retrieval**
 - PageRank / HITS
- **Map Reduce: Massiv parallele Verarbeitung**
- **Datenströme**
- **Peer to Peer Informationssysteme**
 - No SQL Systeme
- **Multi-Tenancy/Cloud-Datenbanken**

21 Big Data	711
21.1 Datenbanken für das Semantic Web	711
21.1.1 RDF: Resource Description Framework	711
21.1.2 SPARQL: Die RDF Anfragesprache	714
21.1.3 Implementierung einer RDF-Datenbank	716
21.2 Datenströme	719
21.3 Information Retrieval und Suchmaschinen	726
21.3.1 TF-IDF: Dokument-Ranking basierend auf Begriffs-Häufigkeit	727
21.3.2 Invertierte Indexierung	728
21.3.3 Page Rank	729
21.3.4 Der HITS Algorithmus	732
21.4 MapReduce: Massiv parallele Datenverarbeitung	735

21.5 Peer-to-Peer-Informationssysteme	740
21.5.1 P2P-Systeme für den Datenaustausch (File-Sharing)	741
21.5.2 Verteilte Hashtabellen (Distributed Hash Tables DHTs)	742
21.5.3 Mehrdimensionaler P2P-Datenraum	746
21.6 No-SQL- und Key/Value-Datenbanksysteme	748
21.7 Multi-Tenancy, Cloud Computing und Software as a Service	750
21.8 Übungen	756
21.9 Literatur	758

Weitere Decision-Support Anfrage-Typen

- Top N-Anfragen
 - Ich will nur die N besten Treffer erhalten und nicht alle 5 Millionen
 - Muss bei der Anfrageoptimierung berücksichtigt werden
- Online Aggregation
 - Man berechnet das Ergebnis approximativ
 - Je länger die Anfrage läuft, desto genauer wird das Ergebnis

Top N-Anfragen

Select A.*

From Angestellte A, Abteilungen abt

Where A.Abteilung = abt.AbteilungsNr and abt.Ort = Passau

Order by A.Gehalt

Stop after 20

Top N-Anfragen auf Verteilten Datenquellen

Mietspiegel	
Ort	Miete
Garching	800
Ismaning	900
Unterföhring	1000
Nymphenburg	1500
Bogenhausen	1600
Grünwald	1700

Kindergarten	
Ort	Beitrag
Grünwald	-100
Unterföhring	0
Bogenhausen	100
Ismaning	200
Garching	250
Nymphenburg	300

WohnLage	
Ort	Lage
Grünwald	München-Süd
Unterföhring	München-Nord
Ismaning	München-Nord
Garching	München-Nord
Bogenhausen	München-City
Nymphenburg	München-City

```
select m.Ort, m.Miete + k.Beitrag as Kosten
from Mietspiegel m, Kindergarten k
where m.Ort = k.Ort
order by Kosten
fetch first 1 rows only
```

Ranking in DB2

```
with KostenVergleich as (select m.Ort, m.Miete, k.Beitrag
                        from Mietspiegel m, Kindergarten k
                        where m.Ort=k.Ort)
```

```
select r.Ort, r.Rang
from (select v.Ort,
            RANK() over (order by v.Miete + 7 * v.Beitrag) as Rang
     from KostenVergleich v) as r
where r.Rang <= 3
```

Wir erhalten folgendes Ergebnis:

Top-3-Ergebnis	
Ort	Rang
Grünwald	1
Unterföhring	1
Bogenhausen	3
Ismaning	3

Hier klicken **Blick ins Buch!**



Ranking innerhalb von Untergruppen

```
with KostenLageVergleich as (  
  select m.Ort, m.Miete, k.Beitrag, l.Lage  
  from Mietspiegel m, Kindergarten k, WohnLage l  
  where m.Ort = k.Ort and k.Ort = l.Ort
```

```
  select k.Ort, k.Lage, k.Miete + 3 * k.Beitrag as Kosten,  
         rank() over (partition by k.Lage  
                     order by k.Miete + 3 * k.Beitrag asc) as LageRang  
  from KostenLageVergleich k  
  order by k.Lage, LageRang
```

Die Partitionierung der (virtuellen) Relation *KostenLageVergleich* erfolgt gemäß des Attributs *Lage*. Als Ergebnis der Anfrage erhalten wir folgende Relation:

Ergebnis			
Ort	Lage	Kosten	LageRang
Bogenhausen	München-City	1900	1
Nymphenburg	München-City	2400	2
Unterföhring	München-Nord	1000	1
Ismaning	München-Nord	1500	2
Garching	München-Nord	1550	3
Grünwald	München-Süd	1400	1

Threshold-Algorithmus zur Auswertung von Top-n-Anfragen (3)

Mietspiegel		Kindergarten	
Ort	Miete	Ort	Beitrag
Garching	800	Grünwald	-100
Ismaning	900	Unterföhring	0
Unterföhring	1000	Bogenhausen	100
Nymphenburg	1500	Ismaning	200
Bogenhausen	1600	Garching	250
Grünwald	1700	Nymphenburg	300

Zwischenergebnis: Phase 1	
Ort	m.Miete + k.Beitrag
Threshold	700
Garching	1050
Grünwald	1600

Mietspiegel		Kindergarten	
Ort	Miete	Ort	Beitrag
Garching	800	Grünwald	-100
Ismaning	900	Unterföhring	0

Threshold-Algorithmus zur Auswertung von Top-n-Anfragen (3)

Mietspiegel			Kindergarten	
Ort	Miete		Ort	Beitrag
Garching	800		Grünwald	-100
Ismaning	900	↗ ↘	Unterföhring	0
Unterföhring	1000		Bogenhausen	100
Nymphenburg	1500		Ismaning	200
Bogenhausen	1600		Garching	250
Grünwald	1700		Nymphenburg	300

Zwischenergebnis: Phase 2	
Ort	m.Miete + k.Beitrag
Threshold	900
Unterföhring	1000
Garching	1050
Ismaning	1100
Grünwald	1600

Threshold-Algorithmus zur Auswertung von Top_n-Anfragen

Mietspiegel	
Ort	Miete
Garching	800
Ismaning	900
Unterföhring	1000
Nymphenburg	1500
Bogenhausen	1600
Grünwald	1700

Kindergarten	
Ort	Beitrag
Grünwald	-100
Unterföhring	0
Bogenhausen	100
Ismaning	200
Garching	250
Nymphenburg	300



Zwischenergebnis: Phase 3	
Ort	m.Miete + k.Beitrag
Unterföhring	1000
Garching	1050
Ismaning	1100
Threshold	1100
Grünwald	1600
Bogenhausen	1700

No-Random-Access-Algorithmus

Mietspiegel	
Ort	Miete
Garching	800
Ismaning	900
Unterföhring	1000
Nymphenburg	1500
Bogenhausen	1600
Grünwald	1700

Kindergarten	
Ort	Beitrag
Grünwald	-100
Unterföhring	0
Bogenhausen	100
Ismaning	200
Garching	250
Nymphenburg	300

Zwischenergebnis	
Ort	Kosten
Garching	700 ↗
Grünwald	700 ↗

Mietspiegel	
Ort	Miete
Garching	800
Ismaning	900
Unterföhring	1000
Nymphenburg	1500
Bogenhausen	1600
Grünwald	1700

Kindergarten	
Ort	Beitrag
Grünwald	-100
Unterföhring	0
Bogenhausen	100
Ismaning	200
Garching	250
Nymphenburg	300

Zwischenergebnis	
Ort	Kosten
Garching	800 ↗
Grünwald	800 ↗
Unterföhring	900 ↗
Ismaning	900 ↗

Mietspiegel	
Ort	Miete
Garching	800
Ismaning	900
Unterföhring	1000
Nymphenburg	1500

Kindergarten	
Ort	Beitrag
Grünwald	-100
Unterföhring	0
Bogenhausen	100
Ismaning	200

Zwischenergebnis	
Ort	Kosten
Garching	900 ↗
Grünwald	900 ↗
Unterföhring	1000 ✓
Ismaning	1000 ↗

No-Random-Access-Algorithmus

Mietspiegel	
Ort	Miete
Garching	800
Ismaning	900
Unterföhring	1000
Nymphenburg	1500
Bogenhausen	1600
Grünwald	1700

Kindergarten	
Ort	Beitrag
Grünwald	-100
Unterföhring	0
Bogenhausen	100
Ismaning	200
Garching	250
Nymphenburg	300

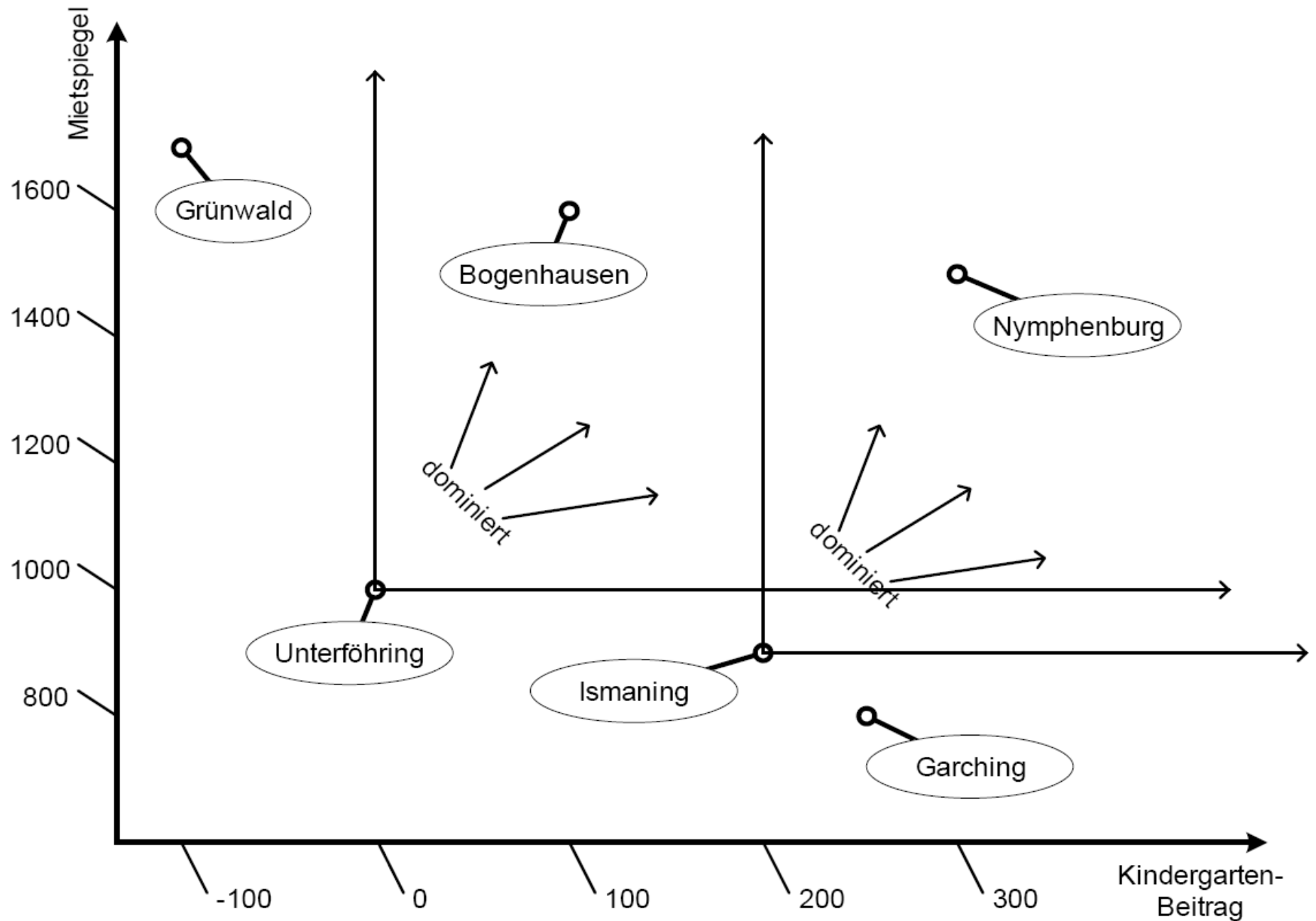
Zwischenergebnis	
Ort	Kosten
Garching	900 ↗
Grünwald	900 ↗
Unterföhring	1000 ✓
Ismaning	1000 ↗
Bogenhausen	1100 ↗

Mietspiegel	
Ort	Miete
Garching	800
Ismaning	900
Unterföhring	1000
Nymphenburg	1500
Bogenhausen	1600
Grünwald	1700

Kindergarten	
Ort	Beitrag
Grünwald	-100
Unterföhring	0
Bogenhausen	100
Ismaning	200
Garching	250
Nymphenburg	300

Zwischenergebnis	
Ort	Kosten
Unterföhring	1000 ✓
Garching	1000 ↗
Ismaning	1100 ✓
Grünwald	1400 ↗
Bogenhausen	1600 ↗
Nymphenburg	1700 ↗

Skyline / Pareto-Optimum



Skyline in SQL

```
with KostenVergleich as (select m.Ort, m.Miete, k.Beitrag
                        from Mietspiegel m, Kindergarten k
                        where m.Ort=k.Ort)
```

```
select k.Ort
from KostenVergleich k
skyline of k.Miete min, k.Beitrag min
```

Skyline in Standard-SQL

```
select k.Ort
from KostenVergleich k
where not exists
    (select * from KostenVergleich dom
     where dom.Miete <= k.Miete and dom.Beitrag <= k.Beitrag and
          (dom.Miete < k.Miete or dom.Beitrag < k.Beitrag))
```

Online-Aggregation

Select abt.Ort, avg(A.Gehalt)

From Angestellte A, Abteilungen abt

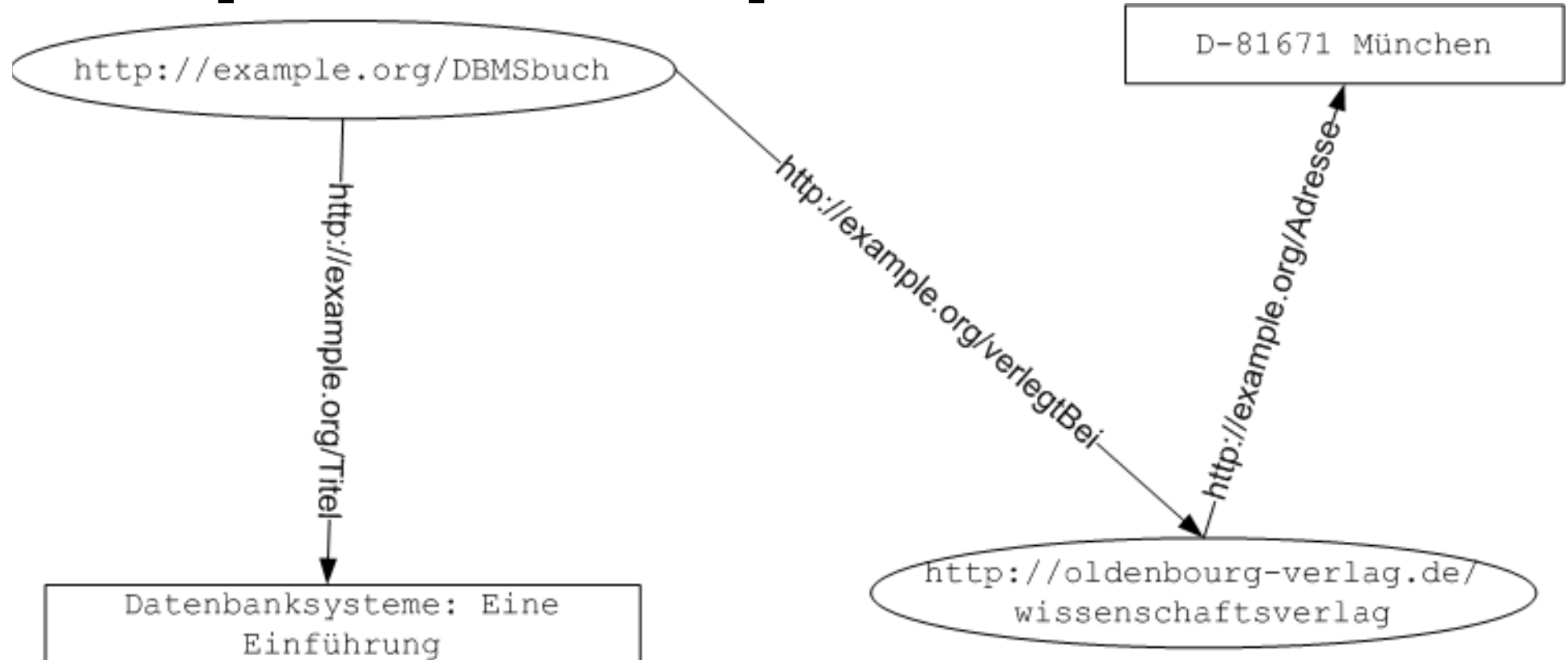
Where A.Abteilung = abt.AbteilungsNr

Group by abt.Ort

Semantic Web: Resource Description Framework (RDF)

- Triple-Datenmodell
 - (Subjekt, Prädikat, Objekt)
- Meist graphische Visualisierung
 - Subjekte und Objekte sind Knoten
 - Prädikate sind gerichtete Kanten
 - Von Subjekt-Knoten nach Objekt Knoten

Beispiel-RDF-Graph



Textuelle Darstellung des Graphen

- Das Buch mit der Kennung `<http://example.org/DBMSbuch>` wird `<http://example.org/verlegtBei>` dem Verlag `<http://oldenbourg-verlag.de/wissenschaftsverlag>`.
- Der `<http://oldenbourg-verlag.de/wissenschaftsverlag>` hat als `<http://example.org/Adresse>` das Literal "D-81671 München".
- Das `<http://example.org/DBMSbuch>` hat als `<http://example.org/Titel>` das Literal "Datenbanksysteme: Eine Einführung".

Somit wird klar, dass das erste Element des Tripels das Subjekt einer Aussage, das zweite das Prädikat und das dritte Element das Objekt darstellt.

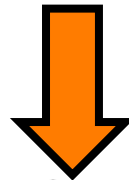
Es gibt mehrere leicht unterschiedliche, textuelle RDF-Tripeldarstellungen, die man unter den Namen *N3*, *N-Triples* oder *Turtle* findet. In unserem Fall sieht die Beispieldatenbank in der *Turtle*-Notation wie folgt aus:

```
<http://example.org/DBMSbuch> <http://example.org/verlegtBei>
    <http://oldenbourg-verlag.de/wissenschaftsverlag>.
<http://oldenbourg-verlag.de/wissenschaftsverlag>
    <http://example.org/Adresse> "D-81671 München".
<http://example.org/DBMSbuch> <http://example.org/Titel>
    "Datenbanksysteme: Eine Einführung".
```


Kurzform

```
@prefix ex: <http://example.org>.  
@prefix ol: <http://oldenbourg-verlag.de>.
```

```
ex:DBMSbuch ex:verlegtBei ol:wissenschaftsverlag.  
ol:wissenschaftsverlag ex:Adresse "D-81671 München".  
ex:DBMSbuch ex:Titel "Datenbanksysteme: Eine Einführung".
```



```
ex:DBMSbuch ex:verlegtBei ol:wissenschaftsverlag;  
    ex:Titel "Datenbanksysteme: Eine Einführung".  
ol:wissenschaftsverlag ex:Adresse "D-81671 München".
```

Man beachte, dass man jetzt ein Semikolon verwendet, um anzugeben, dass das vorherige Subjekt übernommen wird. Man kann mengenwertige Beziehungen auch noch weiter kompaktifizieren, wenn man Cluster gleicher Subjekte und Prädikate bildet. Beispielsweise:

```
ex:DBMSbuch ex:AutorNachName "Kemper" ,  
    "Eickler".
```

Bei diesen Clustern verwendet man Kommata als Abgrenzung der unterschiedlichen Objekte, die zu demselben Subjekt und Prädikat gehören.

Namenlose Knoten

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

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
```

```
ex:DBMSbuch ex:Autor _:k.
```

```
_:k ex:NachName "Kemper"^^xsd:string.
```

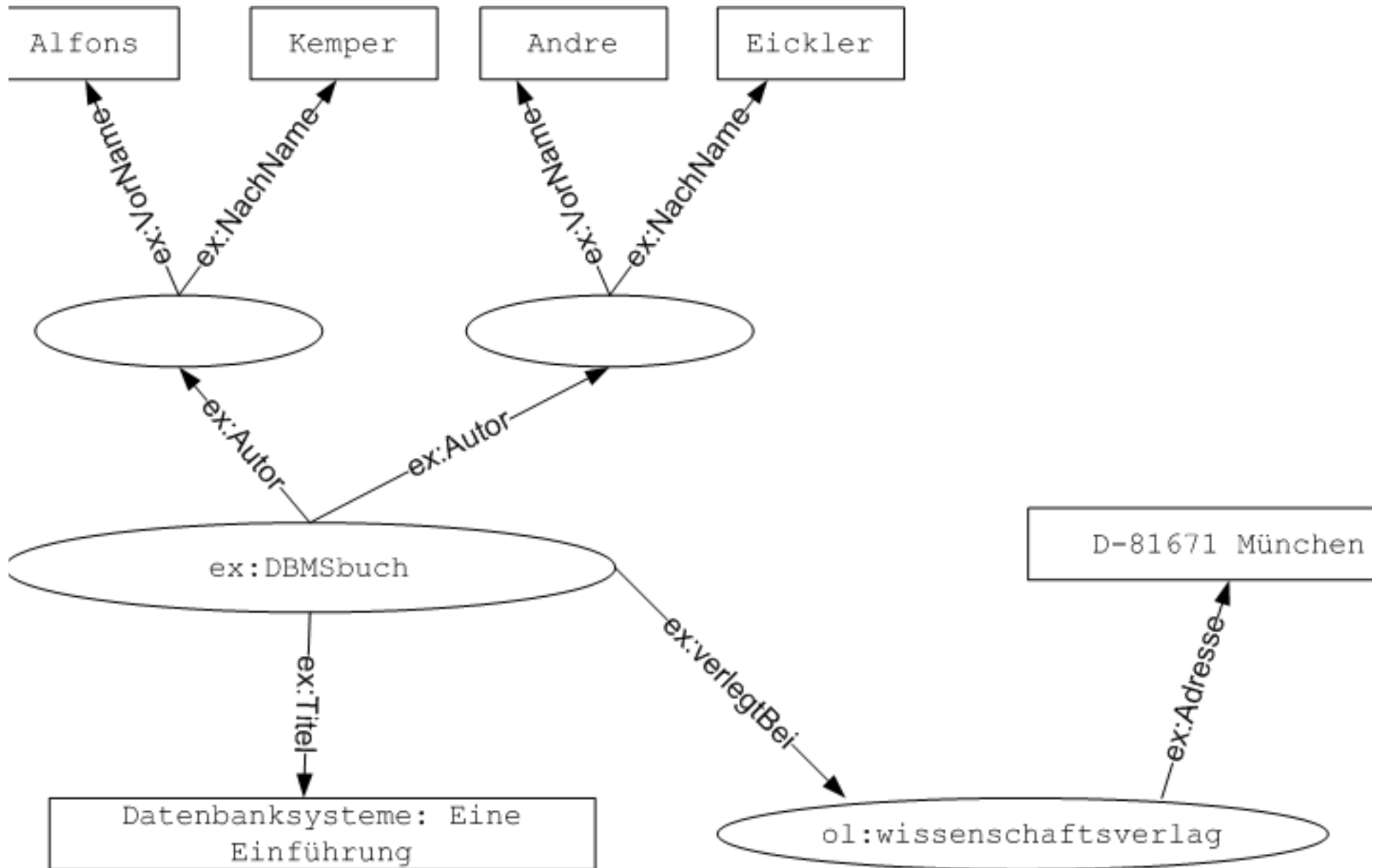
```
_:k ex:VorName "Alfons"^^xsd:string.
```

```
ex:DBMSbuch ex:Autor _:e.
```

```
_:e ex:NachName "Eickler"^^xsd:string.
```

```
_:e ex:VorName "Andre"^^xsd:string.
```

Graph mit unbenannten Knoten



SPARQL: Die RDF Anfragesprache

```
SELECT ?Var1 ?Var2 ... $VarN  
WHERE {Muster1. Muster2. ... MusterM. }
```

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

```
SELECT ?AutorenDesOldenbourgVerlags WHERE  
{ ?buch ex:Autor ?a.  
  ?a ex:NachName ?AutorenDesOldenbourgVerlags.  
  ?buch ex:verlegtBei <http://oldenbourg-verlag.de/  
  wissenschaftsverlag>.  
}
```

SPARQL: Die RDF Anfragesprache

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

SELECT ?KempersBuecherTitel WHERE

{ ?KempersBuecher ex:Autor ?k.

?k ex:NachName "Kemper".

?KempersBuecher ex:Titel ?KempersBuecherTitel.

}

SPARQL: Die RDF Anfragesprache

Union

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

```
SELECT ?KempersOderEicklersBuecherTitel WHERE
{ { ?KempersBuecher ex:Autor ?k.
  ?k ex:NachName "Kemper".
  ?KempersBuecher ex:Titel ?KempersOderEicklersBuecherTitel.
} UNION
{ ?EicklersBuecher ex:Autor ?k.
  ?k ex:NachName "Eickler".
  ?EicklersBuecher ex:Titel ?KempersOderEicklersBuecherTitel.
} }
```

SPARQL: Die RDF Anfragesprache

optional

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

```
SELECT ?KempersBuecherTitel ?KempersBuecherISBN WHERE
{
  ?KempersBuecher ex:Autor ?k.
  ?k ex:NachName "Kemper".
  ?KempersBuecher ex:Titel ?KempersBuecherTitel.
  OPTIONAL { ?KempersBuecher ex:hatISBN ?
KempersBuecherISBN }
}
```

SPARQL: Die RDF Anfragesprache

filter

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

```
SELECT ?KempersBuecherTitel ?auflagenNr WHERE  
{  
  ?KempersBuecher ex:Autor ?k.  
  ?k ex:NachName "Kemper".  
  ?KempersBuecher ex:Titel ?KempersBuecherTitel.  
  ?KempersBuecher ex:Auflage ?auflagenNr.  
  FILTER ( ?auflagenNr > 7 )  
}
```


SPARQL: Die RDF Anfragesprache

count-Aggregation

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

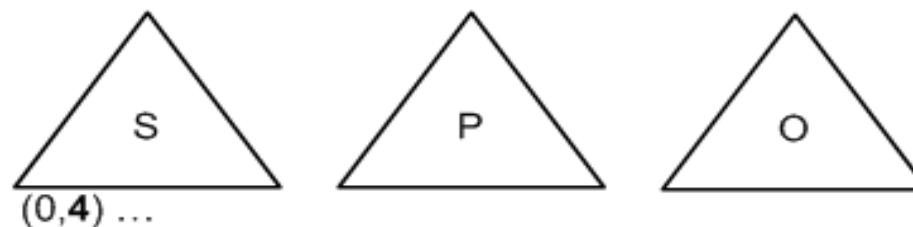
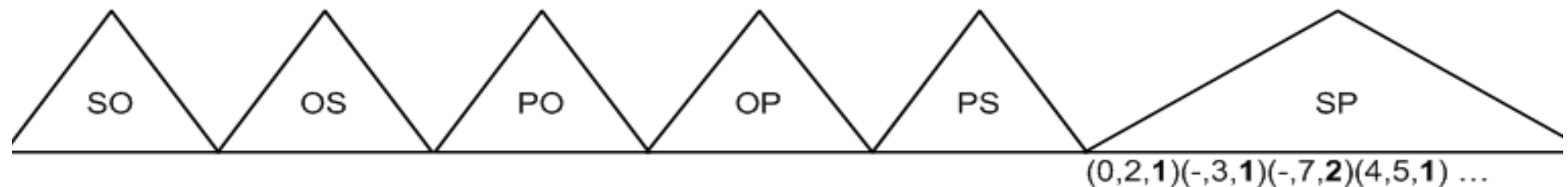
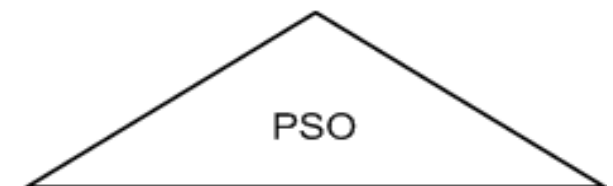
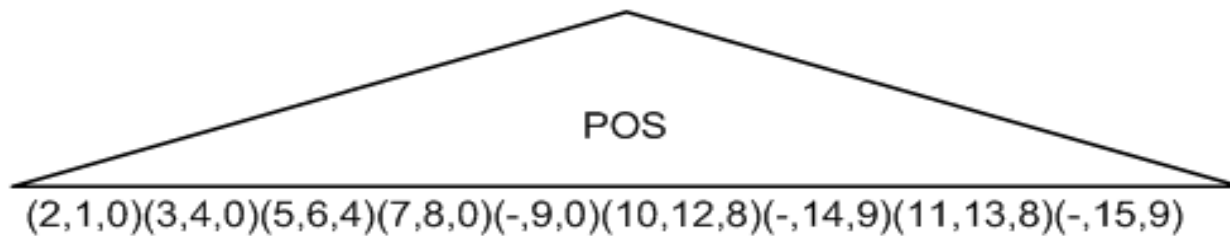
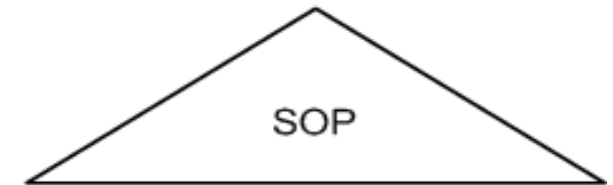
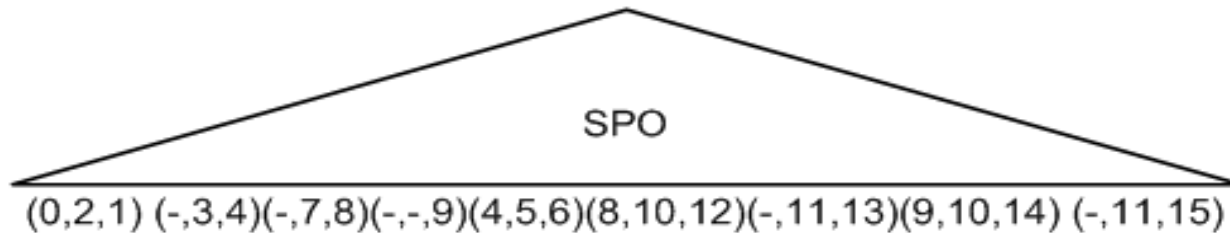
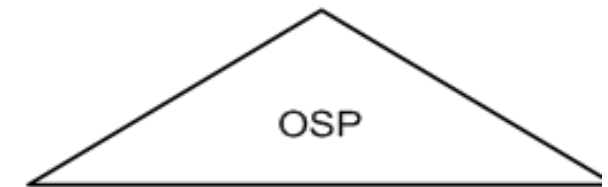
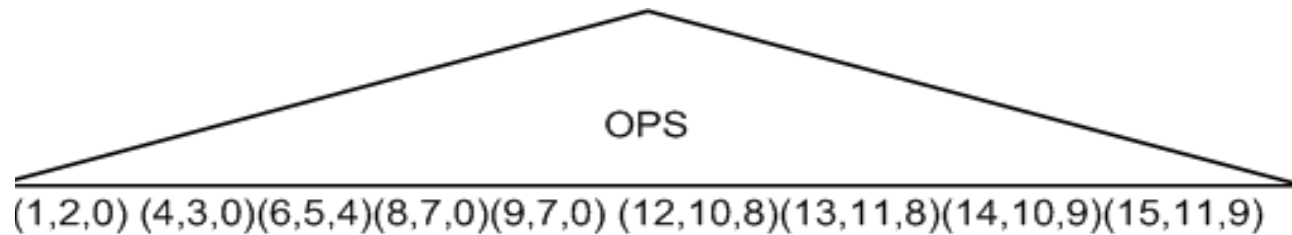
```
SELECT COUNT ?verlag WHERE
{
    ?buch ex:verlegtBei ?verlag.
}
```

Für SQL-affine Leser ist diese Anfrage etwas gewöhnungsbedürftig, da man ja in der Tat die Bücher zählen will. Die SPARQL-Formulierung zielt aber darauf ab, die Anzahl der Vorkommnisse des Musters „**?buch ex:verlegtBei ?verlag**“ für jeden **?verlag** zu zählen.

Implementierung einer RDF-Datenbank: RDF-3X

Dictionary	
Literal/URI	Code
<http://www.example.org/DBMSbuch>	0
Datenbanksysteme: Eine Einführung	1
<http://www.example.org/Titel>	2
<http://www.example.org/verlegtBei>	3
<http://oldenbourg-verlag.de/wissenschaftsverlag>	4
<http://www.example.org/Adresse>	5
D-81671 München	6
<http://www.example.org/Autor>	7
<dummy1>	8
<dummy2>	9
<http://www.example.org/VorName>	10
<http://www.example.org/NachName>	11
Alfons	12
Kemper	13
Andre	14
Eickler	15

B-Bäume ... so viele wie möglich



Kompressionstechnik: Dictionary und Präfix

- Jedes Tripel (s,p,o) wird also genau 6 mal repliziert abgelegt -- allerdings in permutierter Subjekt/Prädikat/Objekt-Reihenfolge, nämlich
 - (p,s,o) , (s,p,o) , (p,o,s) , (o,s,p) , (s,o,p) und (o,p,s) .
- Zusätzlich gibt es noch die sogenannten aggregierten Indexe, die die Anzahl der Vorkommen des jeweiligen Musters repräsentieren. Zum Beispiel bedeutet der Eintrag $(s,o,7)$, dass das Subjekt s siebenmal mit dem Objekt o in einer Beziehung steht -- mit beliebigem Prädikat.
- Das Speichervolumen wird dadurch (dramatisch) reduziert, dass man in den Blättern der Bäume eine Präfix-Komprimierung durchführt. Z.B. wird in dem zweiten Eintrag des SPO-Baums das Subjekt o weggelassen, da es identisch zum ersten Eintrag ist. In dem vierten Eintrag kann sogar die Subjekt- und die Prädikat-Kennung weggelassen werden, da beide identisch zum dritten Eintrag sind

Kompressionstechnik: Dictionary und Präfix

- Es werden aber nicht nur gleiche Präfixe weggelassen; zusätzlich wird auch anstatt des jeweiligen Codes nur die Differenz zum Code des Vorgänger-Tripels gespeichert. Der letzte Eintrag im SPO-Baum würde demnach als $(-,1,1)$ gespeichert, da er in der ersten Komponente identisch zum Vorgänger-Tripel ist, in der zweiten und dritten Komponente ist die Differenz zum Vorgänger-Tripel jeweils 1.
- Diese Kompression ist sehr effektiv, da die Tripel in den Blattknoten ja fortlaufend sortiert sind und sich deshalb immer nur geringfügig vom Vorgänger-Tripel unterscheiden.
- Als Anker für diese Differenz-Kompression wird auf jeder Blatt-Seite immer nur ein vollständiges Tripel, nämlich das Erste, gespeichert.

Anfrageauswertung

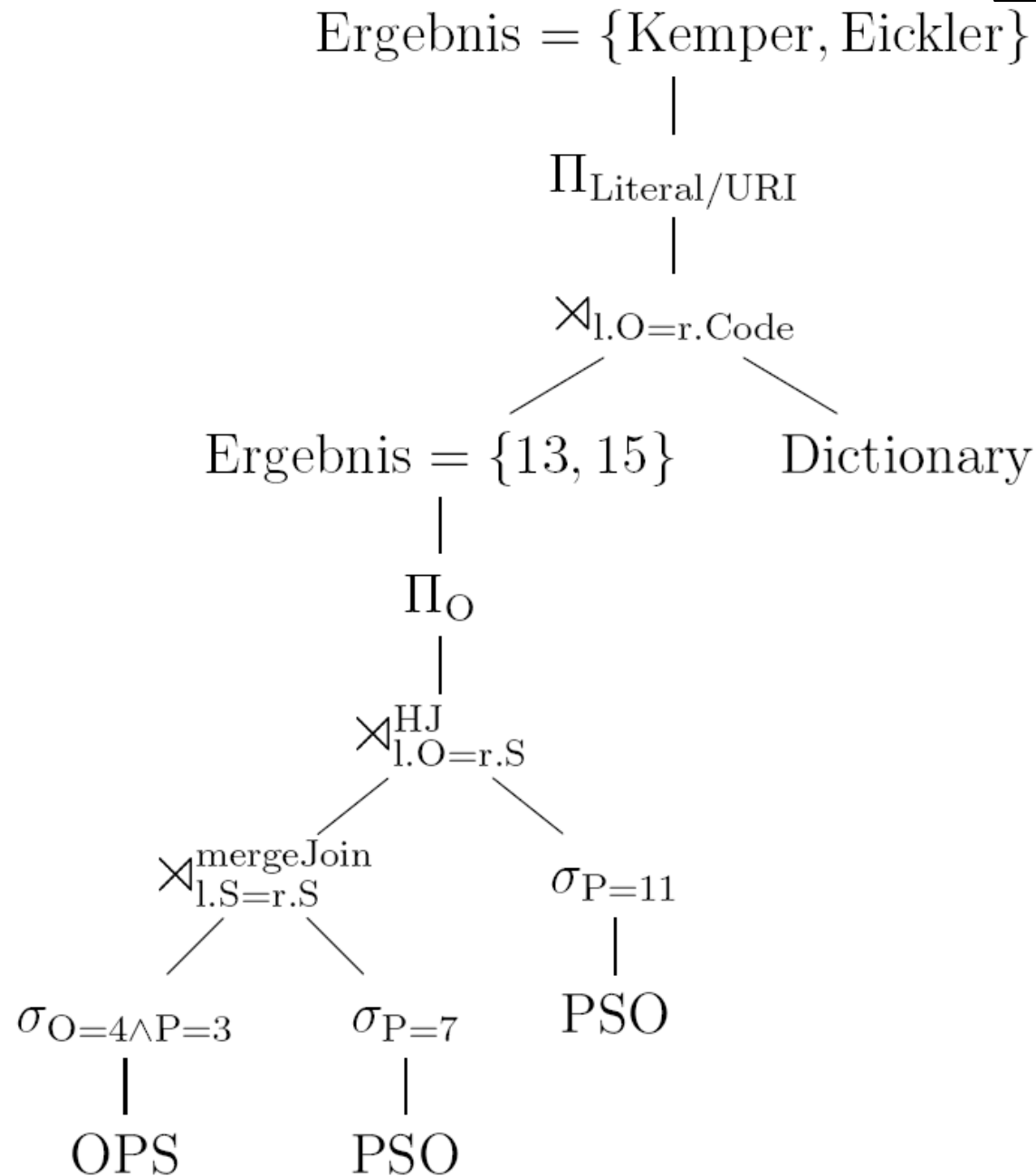
PREFIX ex: <http://www.example.org>

```
SELECT REDUCED ?AutorenDesOldenbourgVerlags WHERE  
{ ?buch ex:Autor ?a.  
  ?a ex:NachName ?AutorenDesOldenbourgVerlags.  
  ?buch ex:verlegtBei <http://oldenbourg-verlag.de/  
    wissenschaftsverlag>.  
}
```



```
SELECT REDUCED ?AutorenDesOldenbourgVerlags WHERE  
{ ?buch 7 ?a.  
  ?a 11 ?AutorenDesOldenbourgVerlags.  
  ?buch 3 4.  
}
```

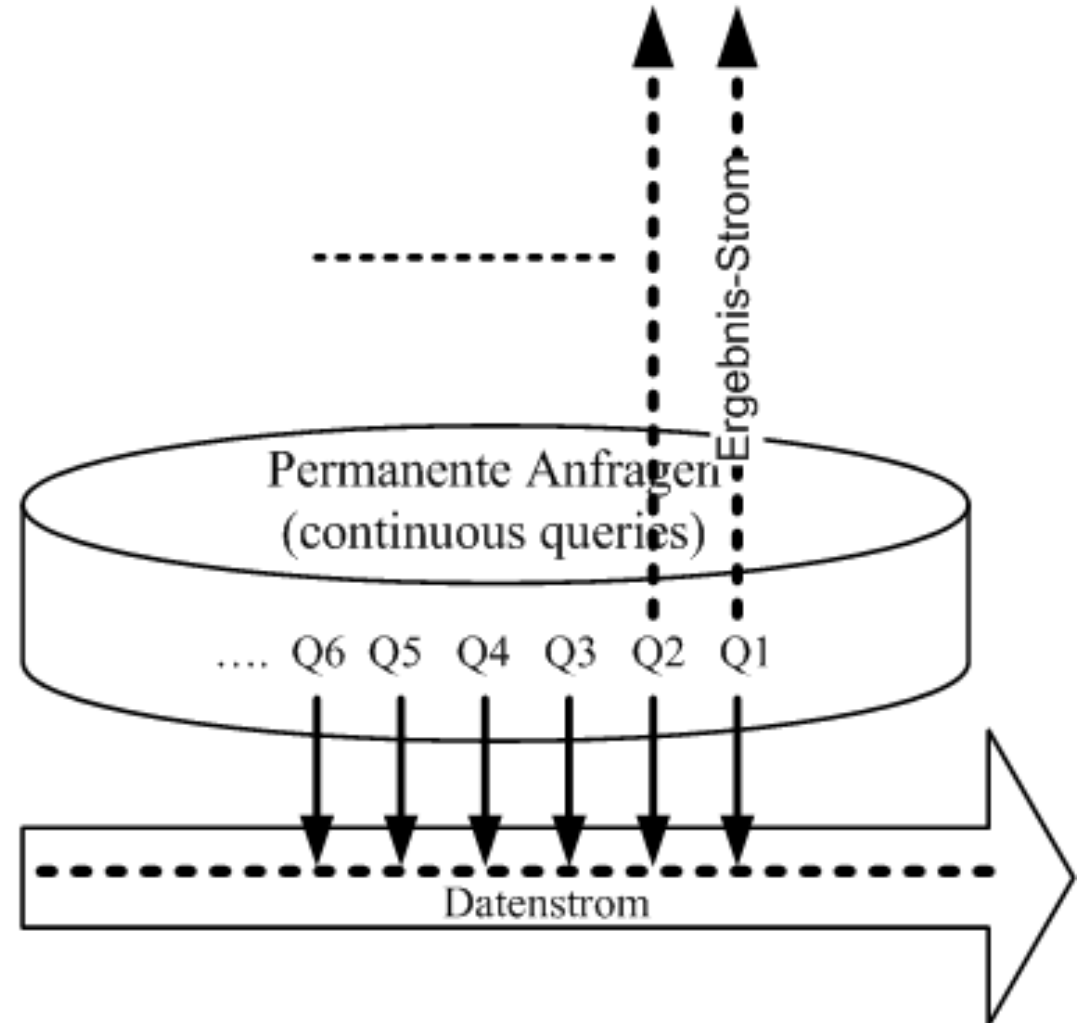
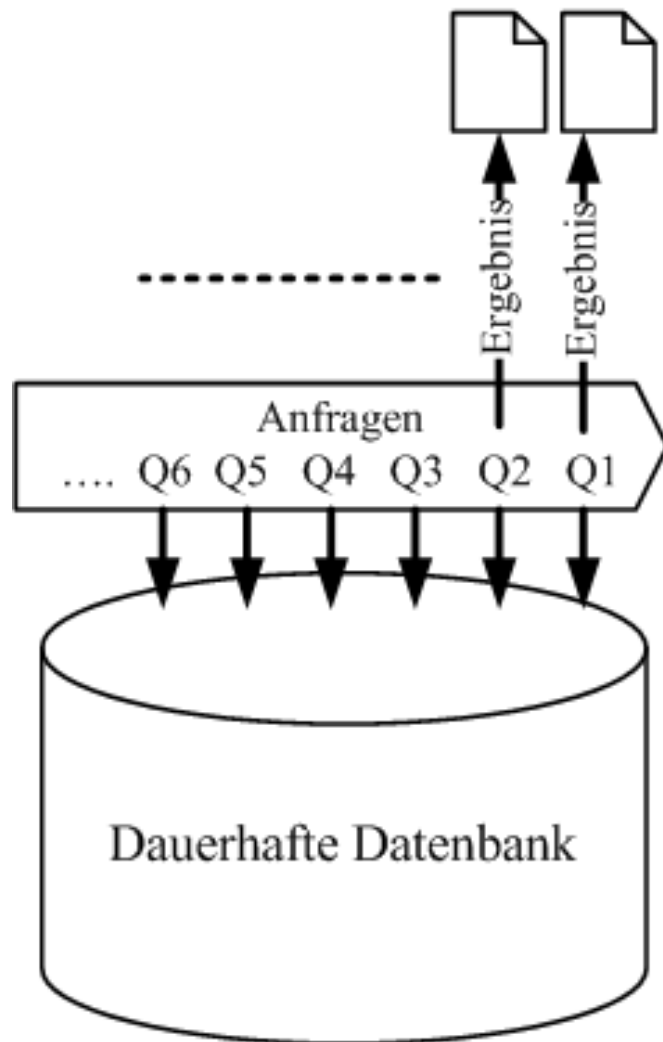
Merge-Joins ... so weit das Auge reicht ...



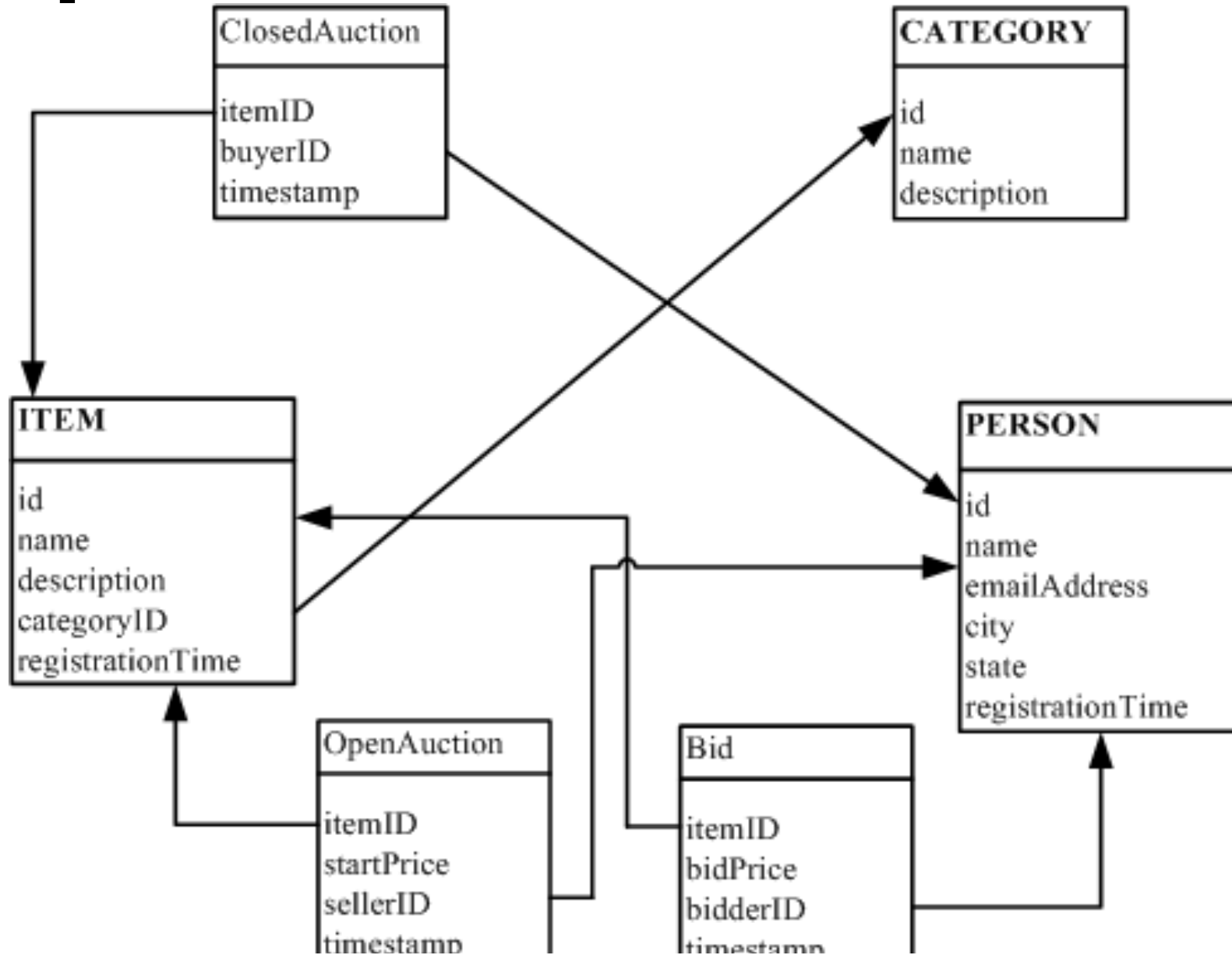
Datenströme

- RFID-Leser, die in der Nähe befindliche („vorbeikommende“) RFID-Tags (Radio Frequency IDentifiers) lesen. Diese Sensoren emittieren Tripel der Art (ReaderID, EPC-Code, TimeStamp), wobei der EPC-Code (electronic product code) eine eindeutige Warenkennzeichnung darstellt.
- Umweltsensoren, um Klimadaten zu ermitteln.
- Börsenticker emittieren die jüngsten Aktienkurse basierend auf den letzten Handelstransaktionen.
- Kameras und andere Sensoren liefern kontinuierliche Informationen über bewegliche Objekte wie z.B. Autos, Menschen, etc.
- Fast alle Menschen liefern über ihr Handy kontinuierlich Daten über ihr Bewegungsmuster. Manche Personen machen es den interessierten Unternehmen noch einfacher, ein präzises Bewegungsmuster via GPS-Daten zu erstellen.

Datenbank versus Datenstrom



Beispiel-Datenstrom



Datenstrom-Definition und einfache Anfrage

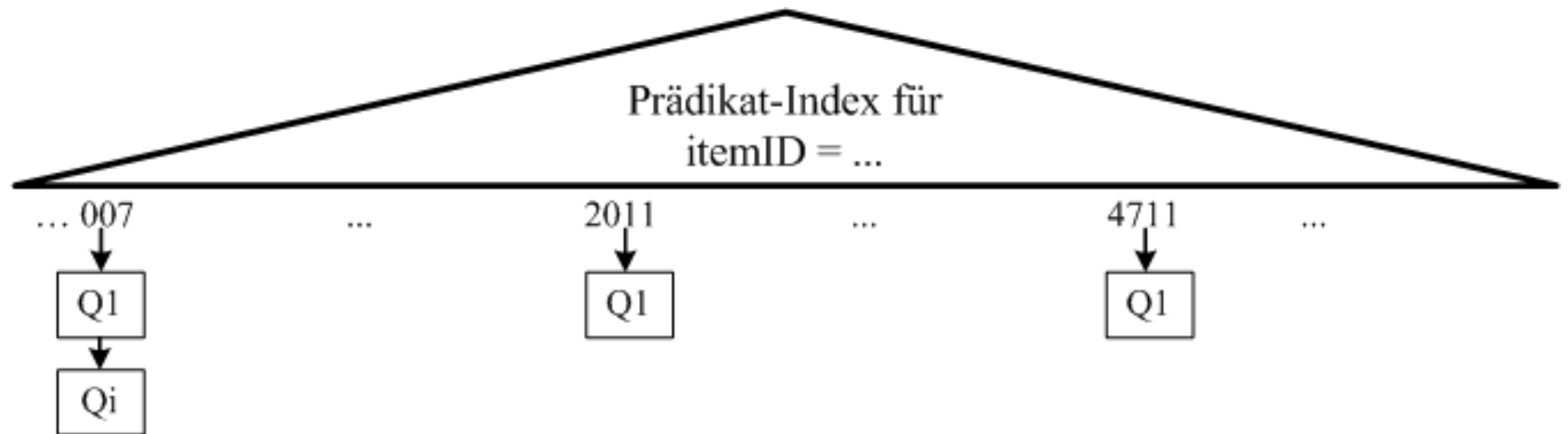
```
create stream OpenAuction(itemID int, sellerID int,  
                           startPrice real, time timestamp)  
source establishConnection('port4711' , 'converter')  
ordered by time;
```

```
select itemID, DollarToEuro(bidPrice), bidderID  
from Bid
```

Subskriptions-Anfrage

```
Q1: select *  
      from Bid  
      where itemID = 4711 or itemID = 007 or itemID = 2011 or ...
```

Prädikat-Index

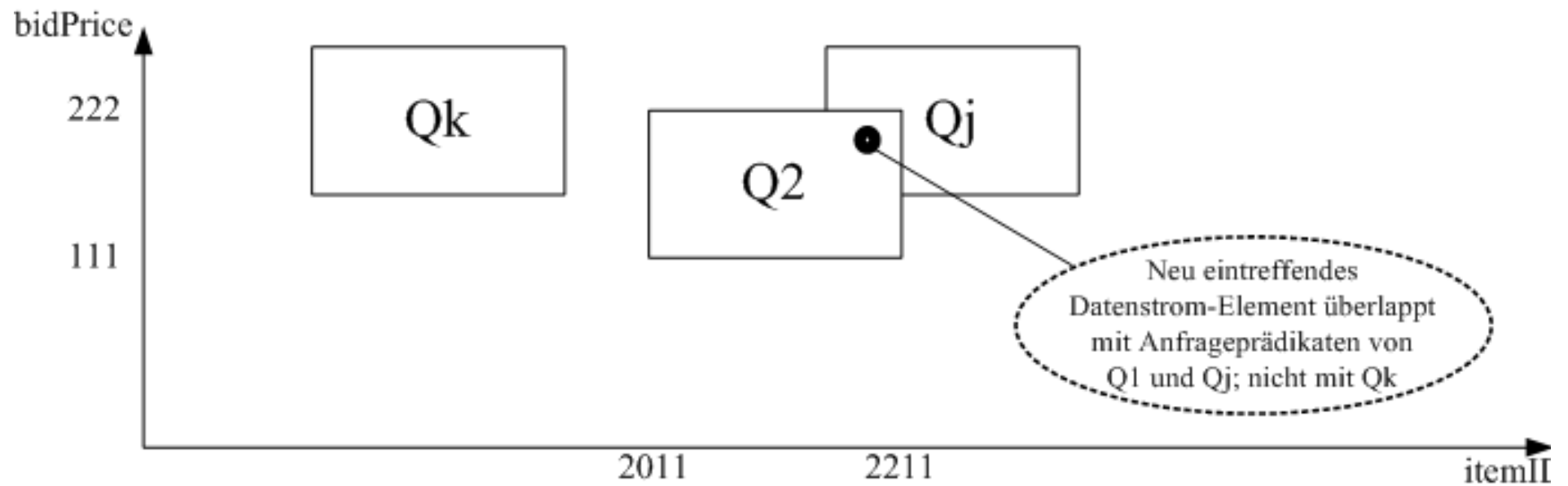


Bereichsanfrage

— 0

```
Q2: select *  
      from Bid  
      where itemID between 2011 and 2211 and  
             bidPrice between 111 and 222
```

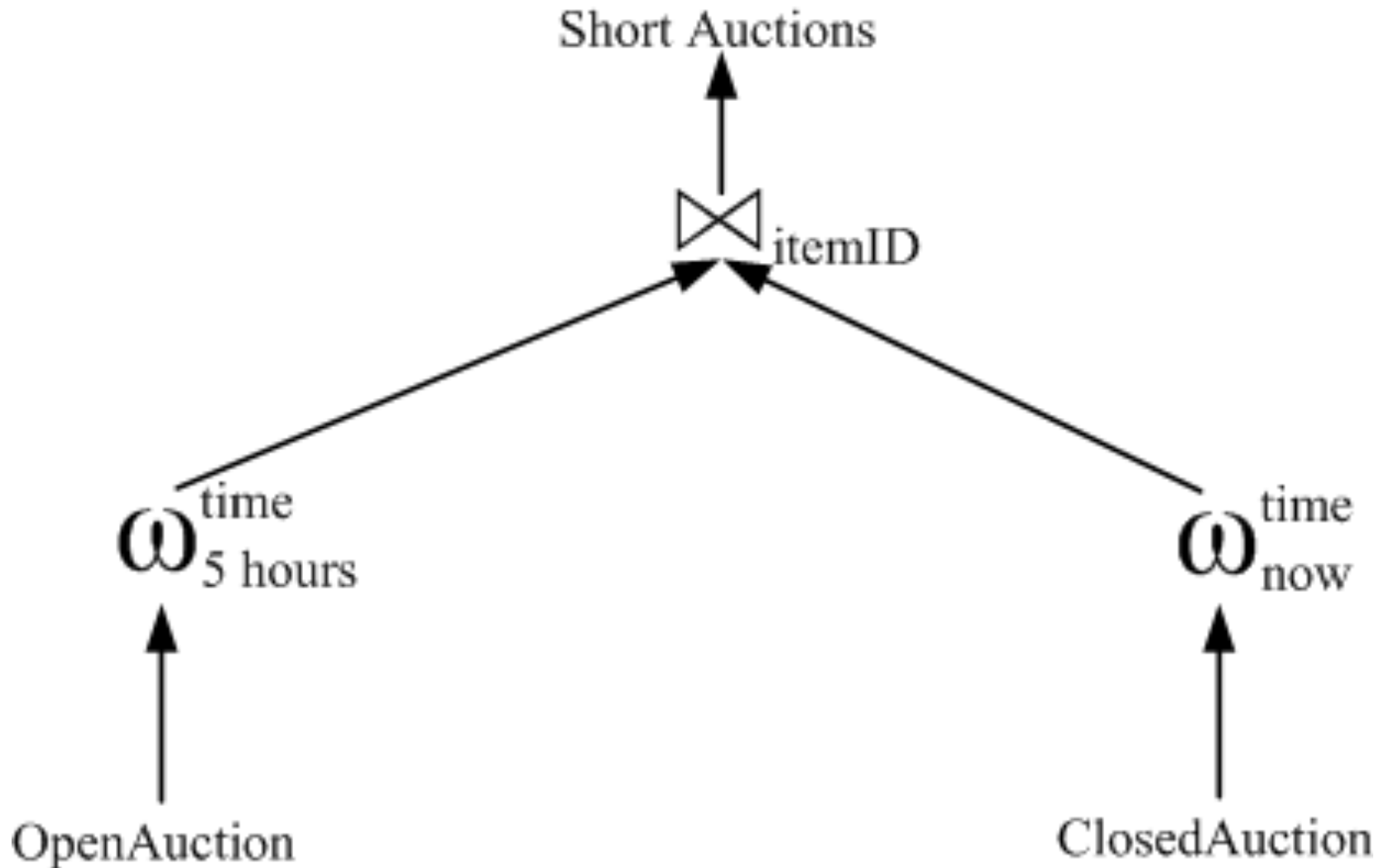
R-Baum-Index



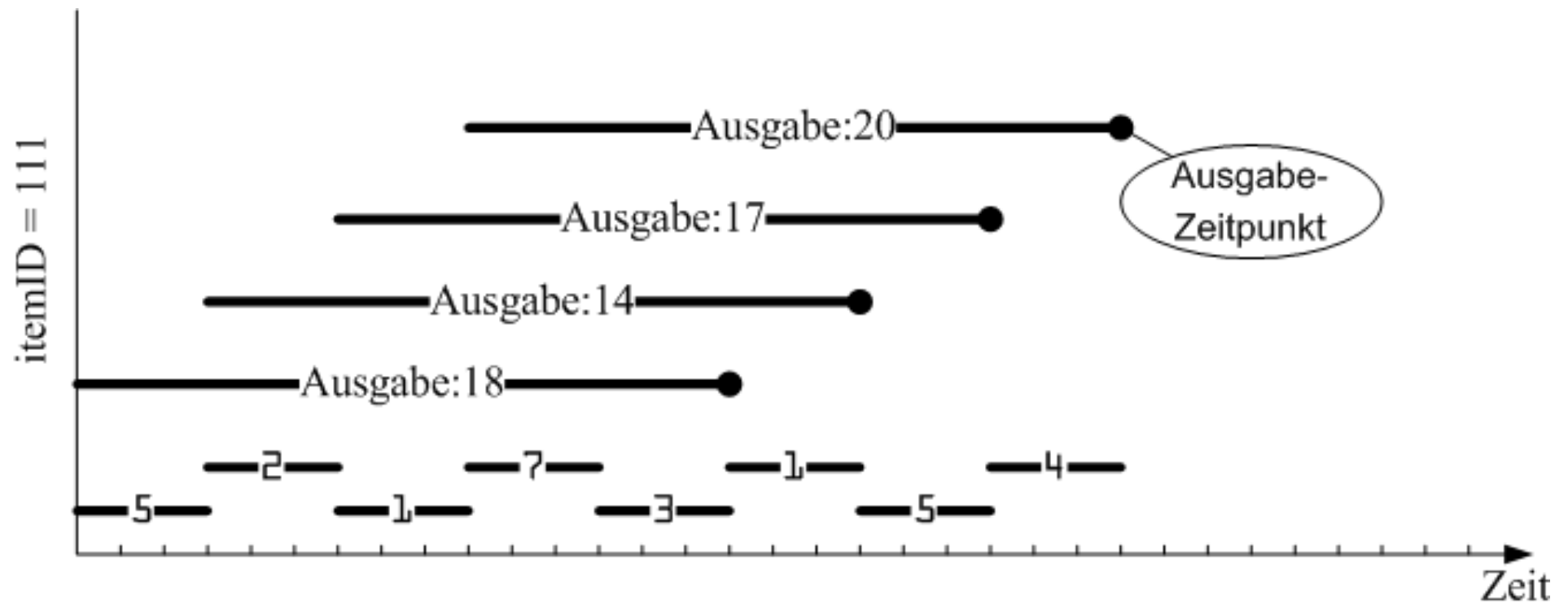
Fenster-Anfrage

```
select O.*  
from OpenAuction O window(range 5 hours),  
     ClosedAuction c  
where o.itemID = c.itemID
```

Auswertung: Short Auctions



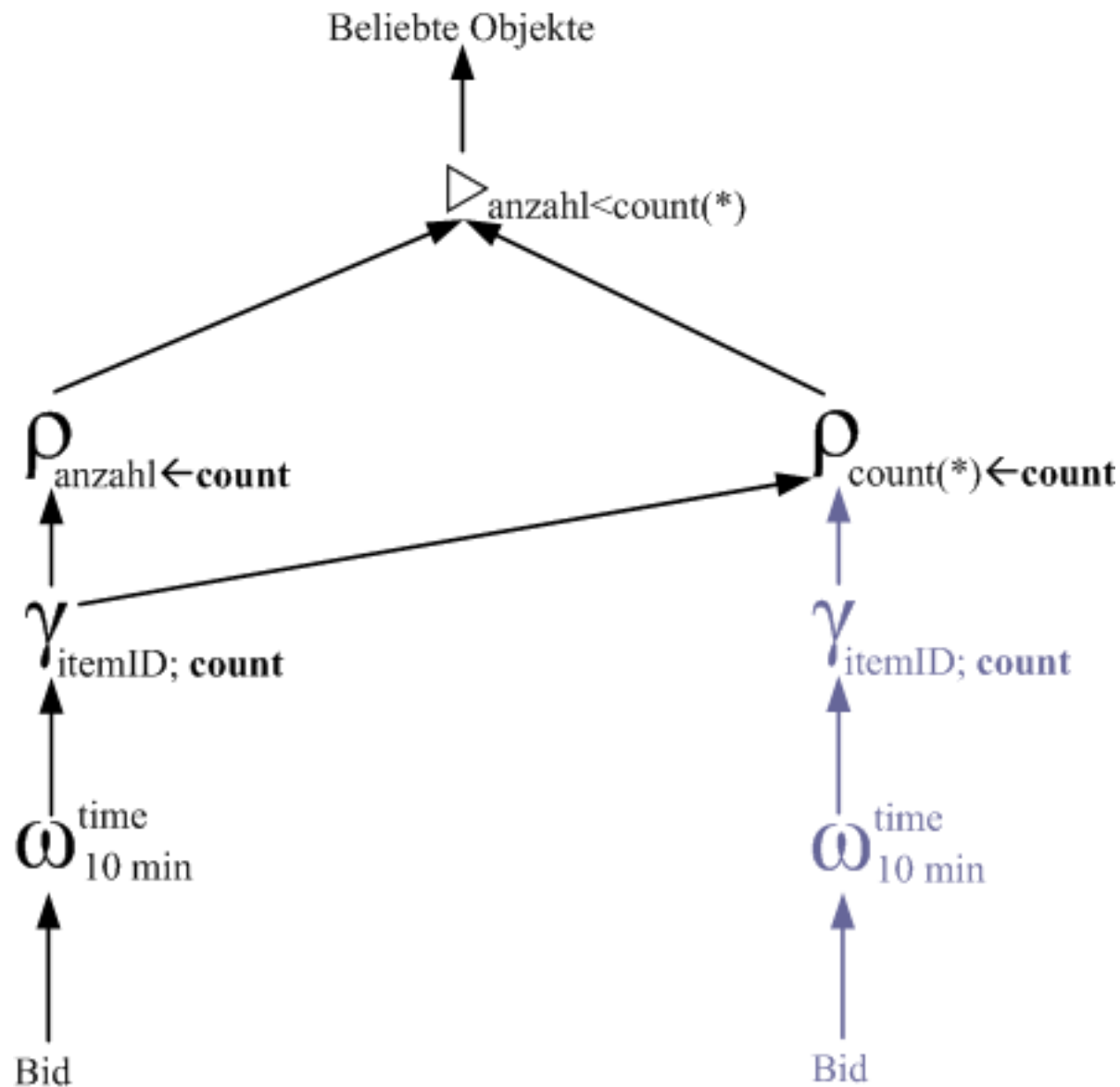
Überlappende Fenster



Hot Items-Anfrage

```
select itemID
from (select b1.itemID as itemID, count(*) as anzahl
      from Bid b1 window(range 10 minutes)
      group by b1.itemID)
where anzahl >= all (select count(*)
                    from Bid b2 window(range 10 minutes)
                    group by b2.itemID)
```

Auswertung: Hot Item



Information Retrieval

- Informationsexplosion im Internet
- Ranking von Dokumenten um relevante Information zu finden
- Ähnlichkeit von Dokumenten (Dissertationen) zu erkennen

TF-IDF: Term Frequency – Inverse Document Frequency

$$TF_{ij} = f_{ij} / \sum_{i=1 \dots |V|} f_{ij}$$

$$IDF_i = \log(N/n_i)$$

$$rel(D_j, Q) = \sum_{i \in Q} TF_{ij} * IDF_i$$

Relevanz-Ranking am Beispiel

D_1	D_2	D_3
Nach dem Spiel ist vor dem Spiel.	Was wir ersinnen ist des Zufalls Spiel.	Der Ball ist rund und ein Spiel dauert neunzig Minuten.

TF_{ij}

Wort i	D_1	D_2	D_3
1: Ball	0	0	1/3
2: Minute	0	0	1/3
3: Spiel	2/2	1/2	1/3
4: Zufall	0	1/2	0

IDF_i

Wort i	N	n_i	N/n_i	$\log(N/n_i)$
1: Ball	3	1	3	0,477121255
2: Minute	3	1	3	0,477121255
3: Spiel	3	3	1	0
4: Zufall	3	1	3	0,477121255

Anfrage $Q \equiv \text{Ball} \wedge \text{Spiel}$ ermittelt man für das Dokument D_3

$$rel(D_3, Q) = 1/3 * 0,477121255 + 1/3 * 0 = 0,1590404182$$

Relevanz-Ranking am Beispiel

D_1	D_2	D_3
Nach dem Spiel ist vor dem Spiel.	Was wir ersinnen ist des Zufalls Spiel.	Der Ball ist rund und ein Spiel dauert neunzig Minuten.

TF_{ij}

Wort i	D_1	D_2	D_3
1: Ball	0	0	1/3
2: Minute	0	0	1/3
3: Spiel	2/2	1/2	1/3
4: Zufall	0	1/2	0

IDF_i

Wort i	N	n_i	N/n_i	$\log(N/n_i)$
1: Ball	3	1	3	0,477121255
2: Minute	3	1	3	0,477121255
3: Spiel	3	3	1	0
4: Zufall	3	1	3	0,477121255

Für die Anfrage $Q' \equiv \text{Ball} \wedge \text{Spiel} \wedge \text{Zufall}$ hat D_2 den höchsten Relevanzwert

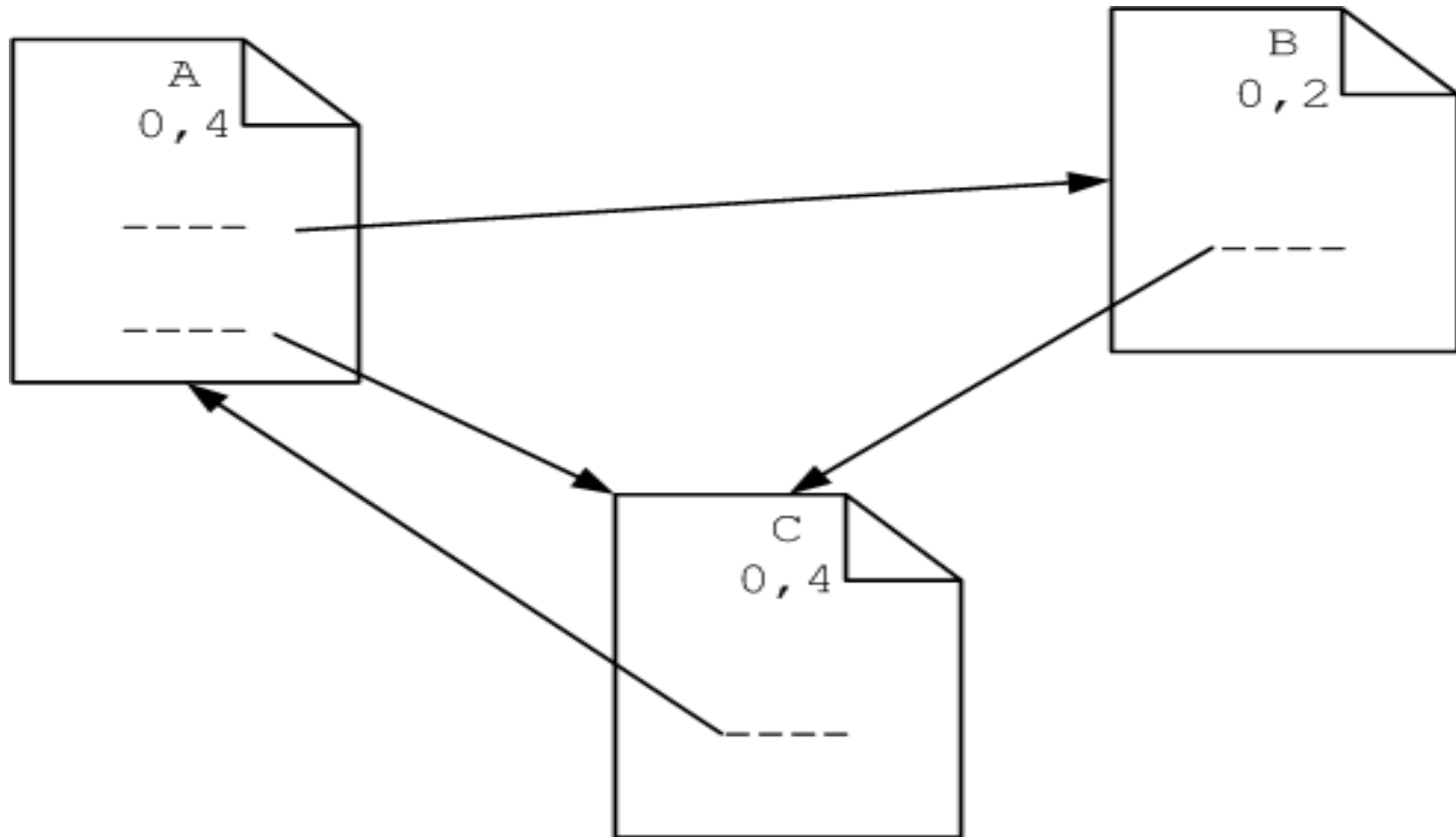
$$\text{rel}(D_2, Q') = 0 * 0,477121255 + 1/2 * 0 + 1/2 * 0,477121255 = 0,2385606275$$

Invertierte Indexierung

Begriff	→	Dokumente
Ball	→	$(D_3, 1)$
Minute	→	$(D_3, 1)$
Spiel	→	$(D_1, 2)$ $(D_2, 1)$ $(D_3, 1)$
Zufall	→	$(D_2, 1)$

Page Rank: Grundidee

$$r(A) = \frac{\alpha}{N} + (1 - \alpha) \left(\frac{r(B_1)}{|B_1|} + \dots + \frac{r(B_n)}{|B_n|} \right)$$

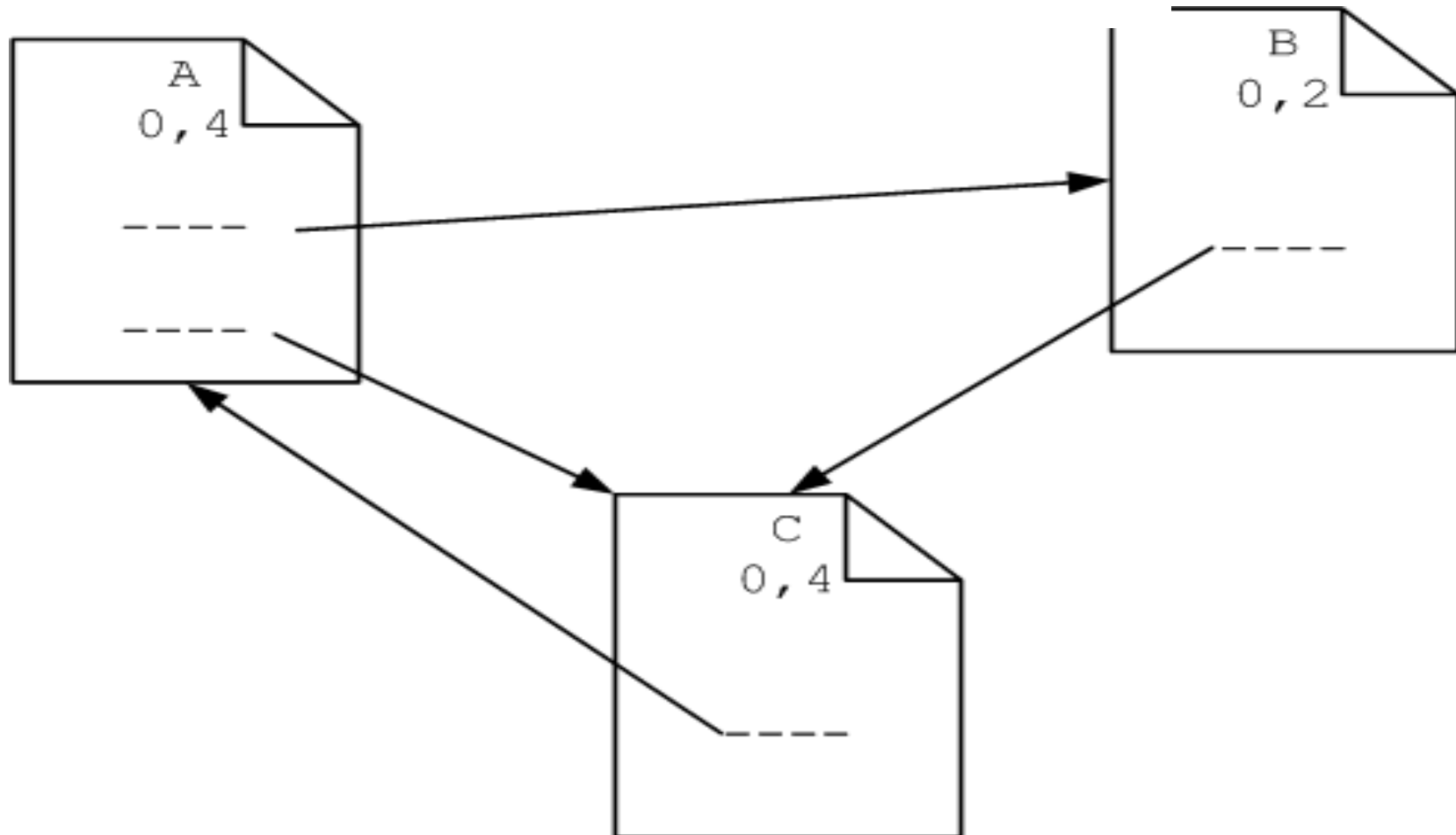


Page Rank: Grundidee

$$r(A) = r(C)/1$$

$$r(B) = r(A)/2$$

$$r(C) = r(A)/2 + r(B)$$



Mathematisches Modell des PageRank

$$M_{ij} = \begin{cases} 1/|P_j| & \text{falls } P_j \text{ auf } P_i \text{ verweist} \\ 0 & \text{sonst} \end{cases}$$

$$M = \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \end{pmatrix} \quad p_0 = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix}$$

Man berechnet dann iterativ die Vektoren

$$p_1 = M * p_0, \quad p_2 = M * p_1 = M * (M * p_0) = M^2 * p_0, \dots, p_i = M^i * p_0$$

Mathematisches Modell des PageRank: unser Beispiel

$$M = \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \end{pmatrix} \quad p_0 = \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix}$$

$$p_1 = Mp_0 = \begin{pmatrix} 0 & 0 & 1 \\ 1/2 & 0 & 0 \\ 1/2 & 1 & 0 \end{pmatrix} * \begin{pmatrix} 1/3 \\ 1/3 \\ 1/3 \end{pmatrix} = \begin{pmatrix} 1/3 \\ 1/6 \\ 1/2 \end{pmatrix}$$

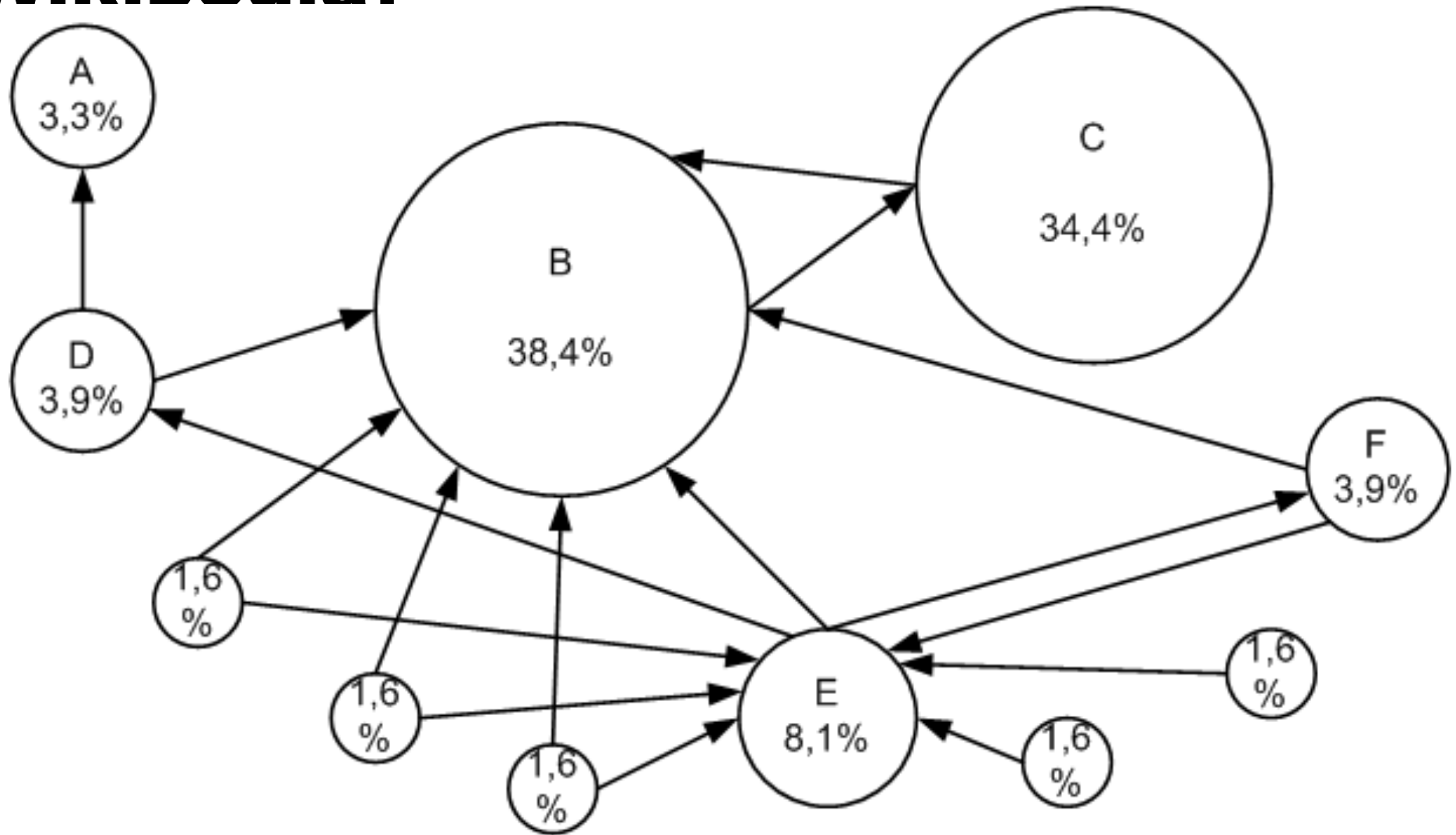
$$p_2 = M * Mp_0 = M * p_1 = \begin{pmatrix} 1/2 \\ 1/6 \\ 1/3 \end{pmatrix}$$
$$p_7 = M^7 * p_0 = \begin{pmatrix} 20/48 \\ 9/48 \\ 19/48 \end{pmatrix}$$

Konvergenz und Dämpfung

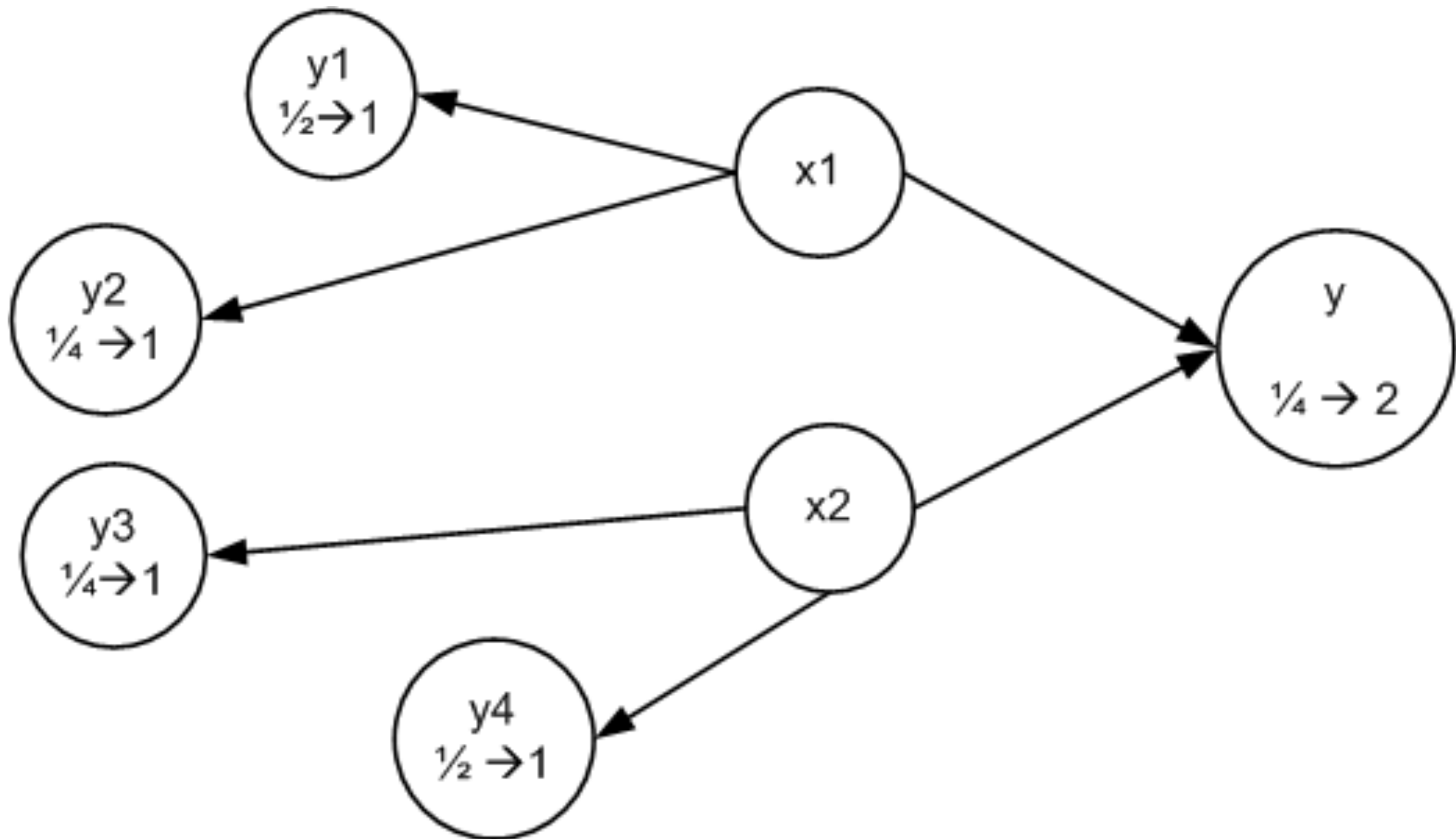
$$p_{\infty} = \lim_{n \rightarrow \infty} M^n p_0$$

$$p_i = (((1 - \alpha) * M) * p_{i-1}) + \begin{pmatrix} \frac{\alpha}{N} \\ \vdots \\ \frac{\alpha}{N} \end{pmatrix}$$

PageRank für größeren Graph [aus Wikipedia]



HITS-Algorithmus: Hubs und Autoritäten



HITS-Algorithmus: Hubs und Autoritäten

Dann wird der Hub-Wert einer Seite i wie folgt definiert:

$$h_i = \delta \sum_{j=1 \dots N} A_{ij} a_j$$

Analog wird die Gewichtung der Autorität einer Seite i wie folgt errechnet:

$$a_i = \lambda \sum_{k=1 \dots N} A_{ik}^T h_k$$

Eigenvektoren der Matrizen AA^T bzw. $A^T A$

$$h = \delta \lambda AA^T h$$

$$a = \delta \lambda A^T A a$$

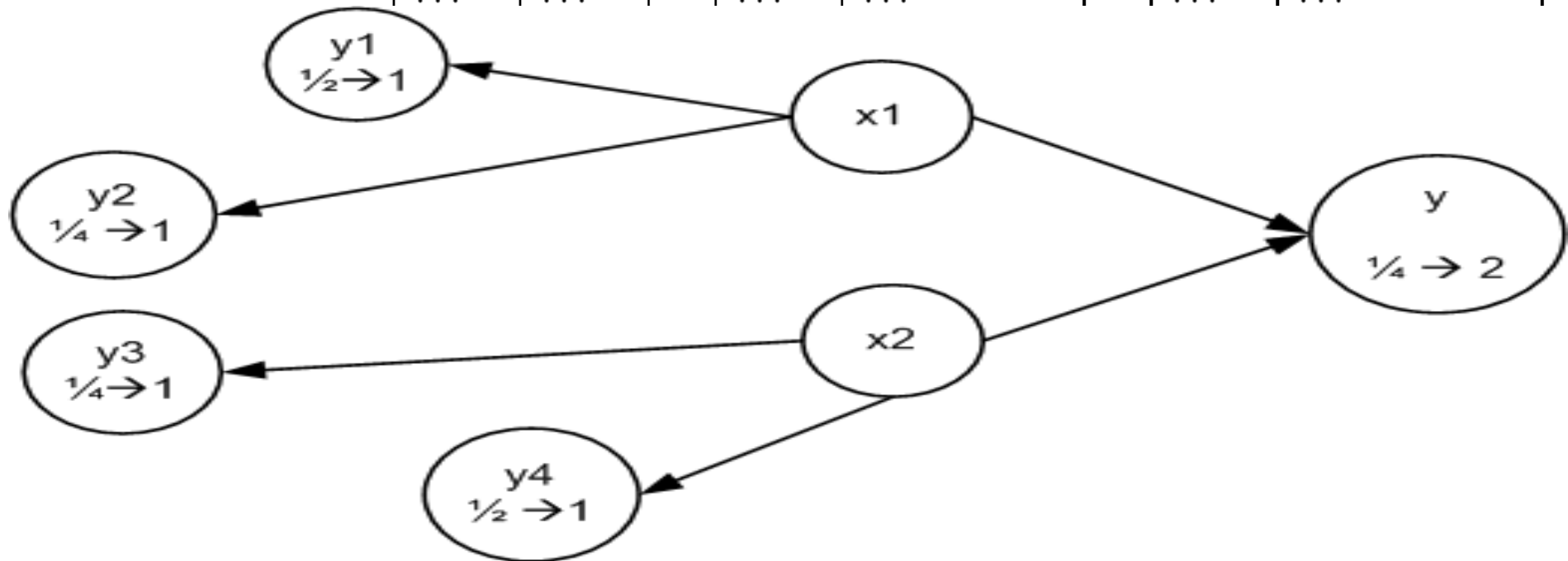
Relationale HITS-Modellierung

A	
von	nach
x1	y1
x1	y2
x2	y3
x2	y4
x1	y
x2	y
...	...

Aut	
Seite	Wert
x1	...
x2	...
y1	1/2
y2	1/4
y3	1/4
y4	1/2
y	1/4
...	...

Aut2 (nach insert)	
Seite	Wert
x1	...
x2	...
y1	1
y2	1
y3	1
y4	1
y	2
...	...

Aut2 (nach update)	
Seite	Wert
x1	...
x2	...
y1	1/2
y2	1/2
y3	1/2
y4	1/2
y	1
...	...



Algorithmus

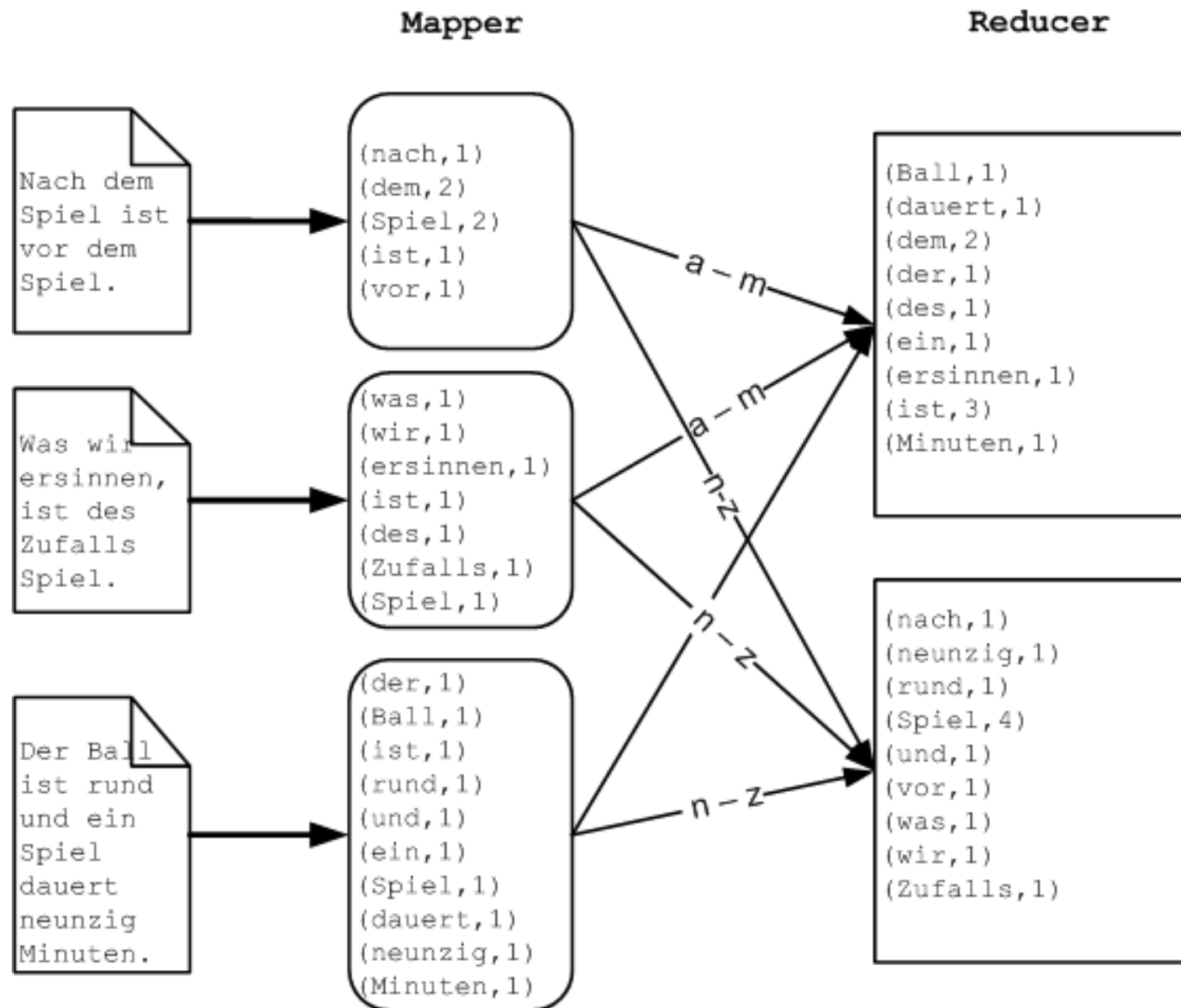
1. Berechne die Hub-Werte jeder Seite q indem man die Summe der Autoritätswerte aller Seiten r ermittelt, auf die q verweist.
2. Berechne die Autorität der Seite p durch Summierung der Hub-Werte der Seiten q , die auf p verweisen
3. Normalisiere die so erhaltenen Autoritätswerte indem man sie mit $\lambda = 1/\max$ multipliziert, wobei \max den Maximalwert aller gerade neu berechneten Autoritätswerte darstellt. Diese Normalisierung ist nötig, um die Werte nicht ins Unermessliche steigen zu lassen.

Algorithmus ... in SQL

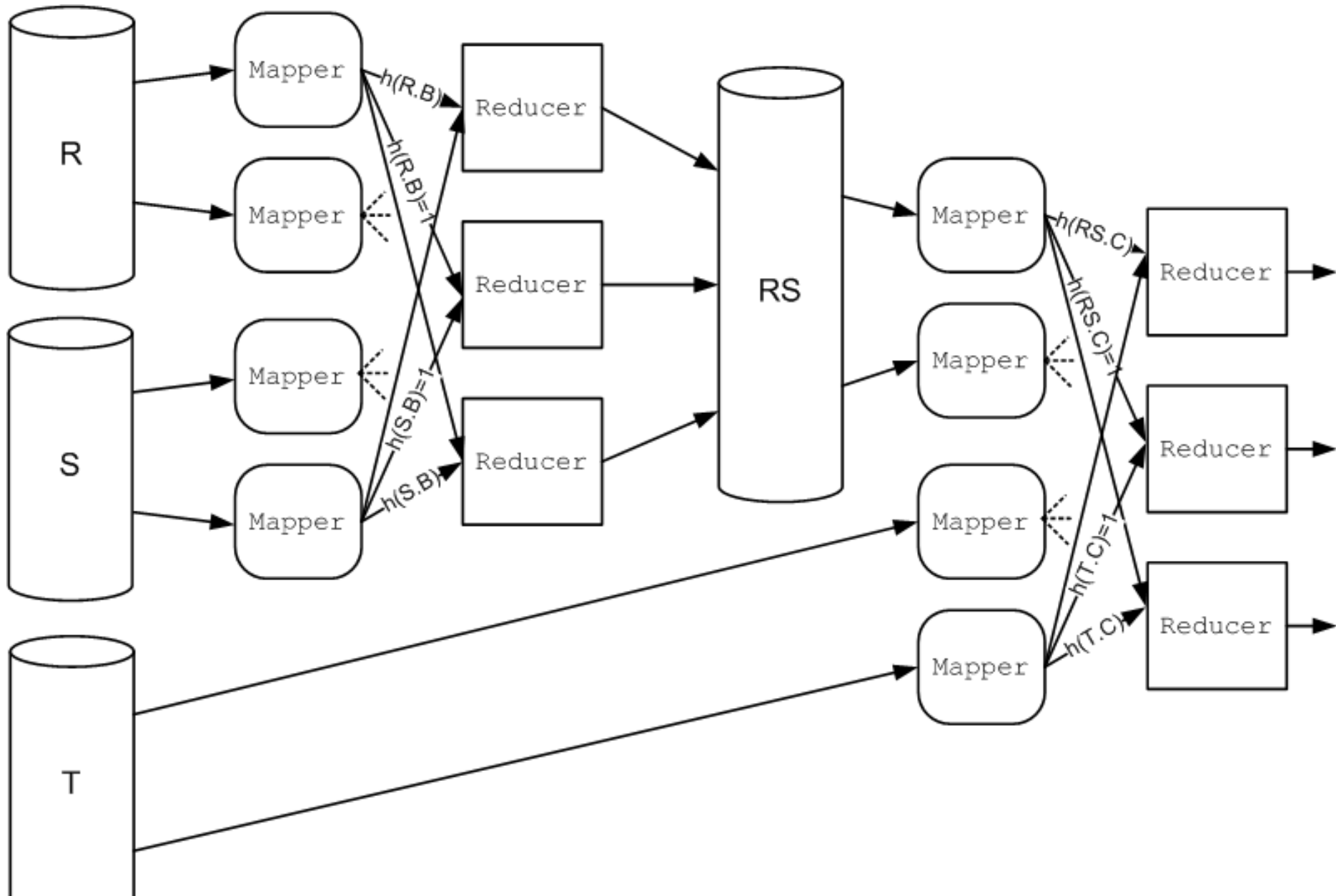
```
insert into Aut2 (  
    select a1.nach, sum(Aut.Wert)  
    from Aut, A a1, A a2  
    where Aut.Seite = a2.nach and a1.von = a2.von  
    group by a1.nach )
```

```
update Aut2  
    set Wert = Wert / (select max(Wert) from Aut2)
```

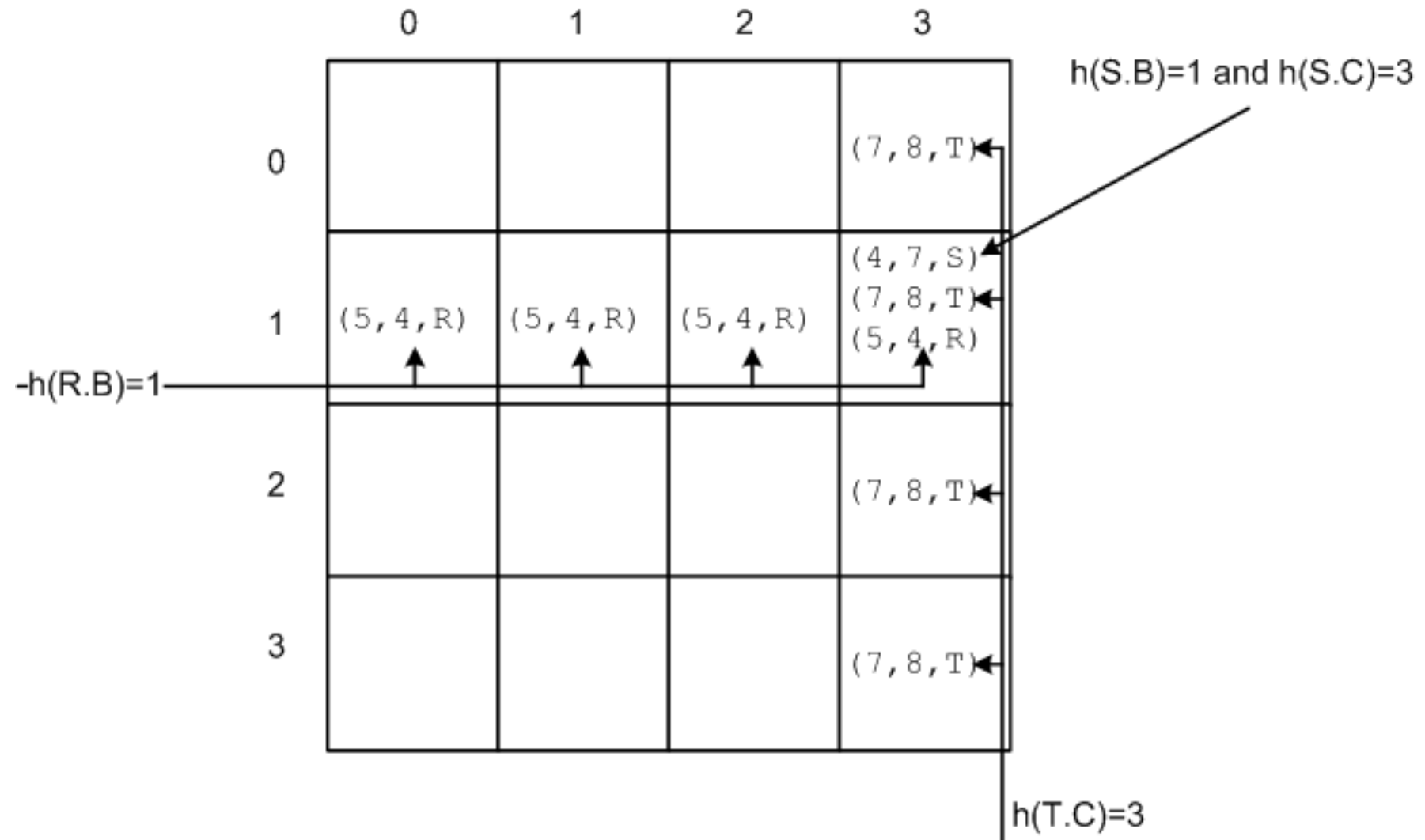
Map Reduce



Join mit Map Reduce



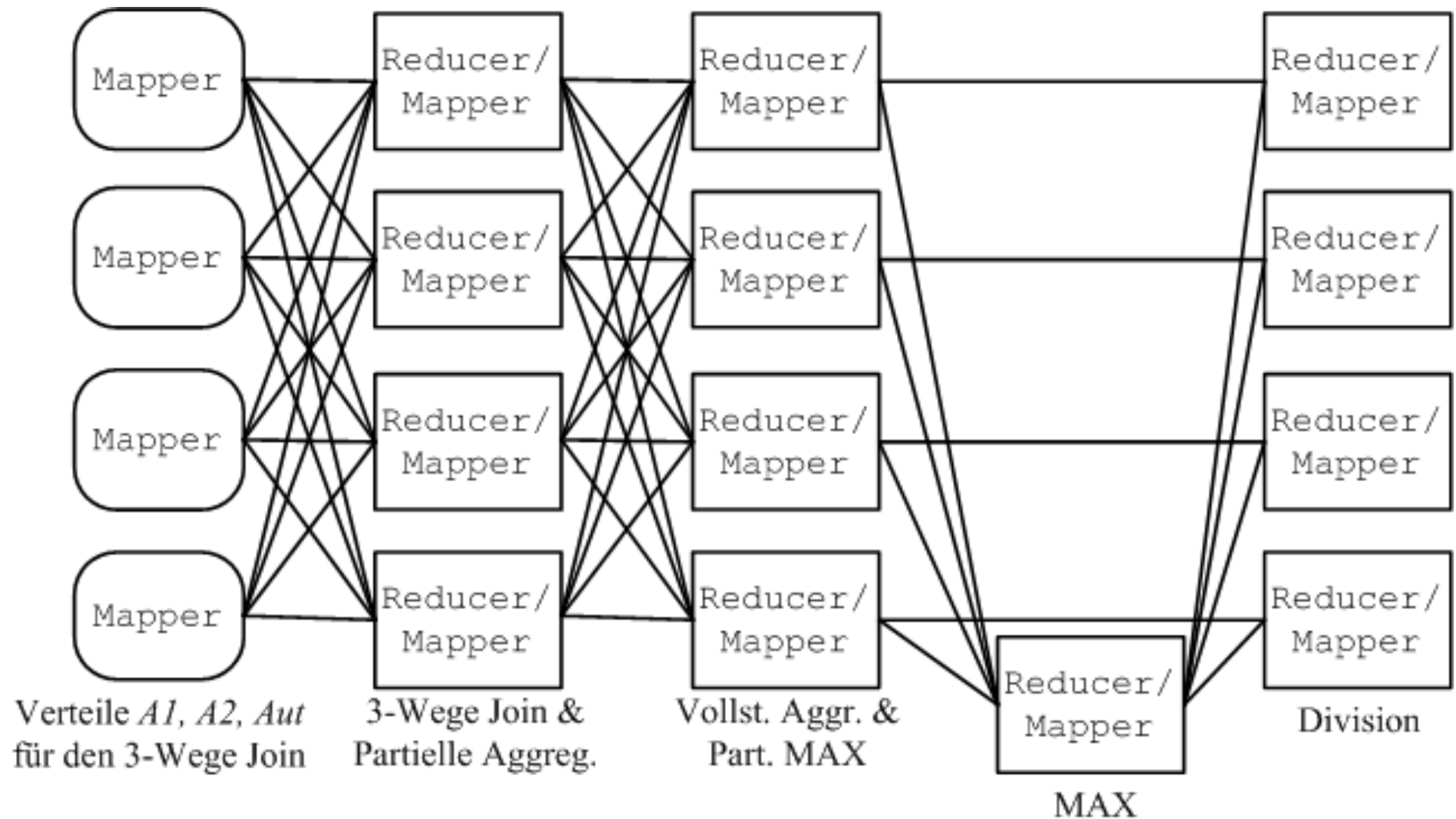
Verbesserung nach Ullman



Map Reduce Skriptsprache: PigLatin

```
a1 = LOAD 'A' AS (a1von, a1nach);
a2 = LOAD 'A' AS (a2von, a2nach);
aut = LOAD 'Aut' AS (seite, wert);
j1 = JOIN a2 BY a2nach, aut BY seite;
j2 = JOIN a1 BY a1von, j1 BY a2von;
g1 = GROUP j2 BY a1nach;
aut2 = FOREACH g1 GENERATE group AS seite, SUM(j2.wert) AS wert;
g2 = GROUP aut2 ALL;
max = FOREACH g2 GENERATE MAX(aut2.wert) AS max;
c = CROSS aut2, max;
aut2Up = FOREACH c GENERATE seite, wert / max;
STORE aut2Up INTO 'Aut2';
```

Auswertung des HITS Algorithmus

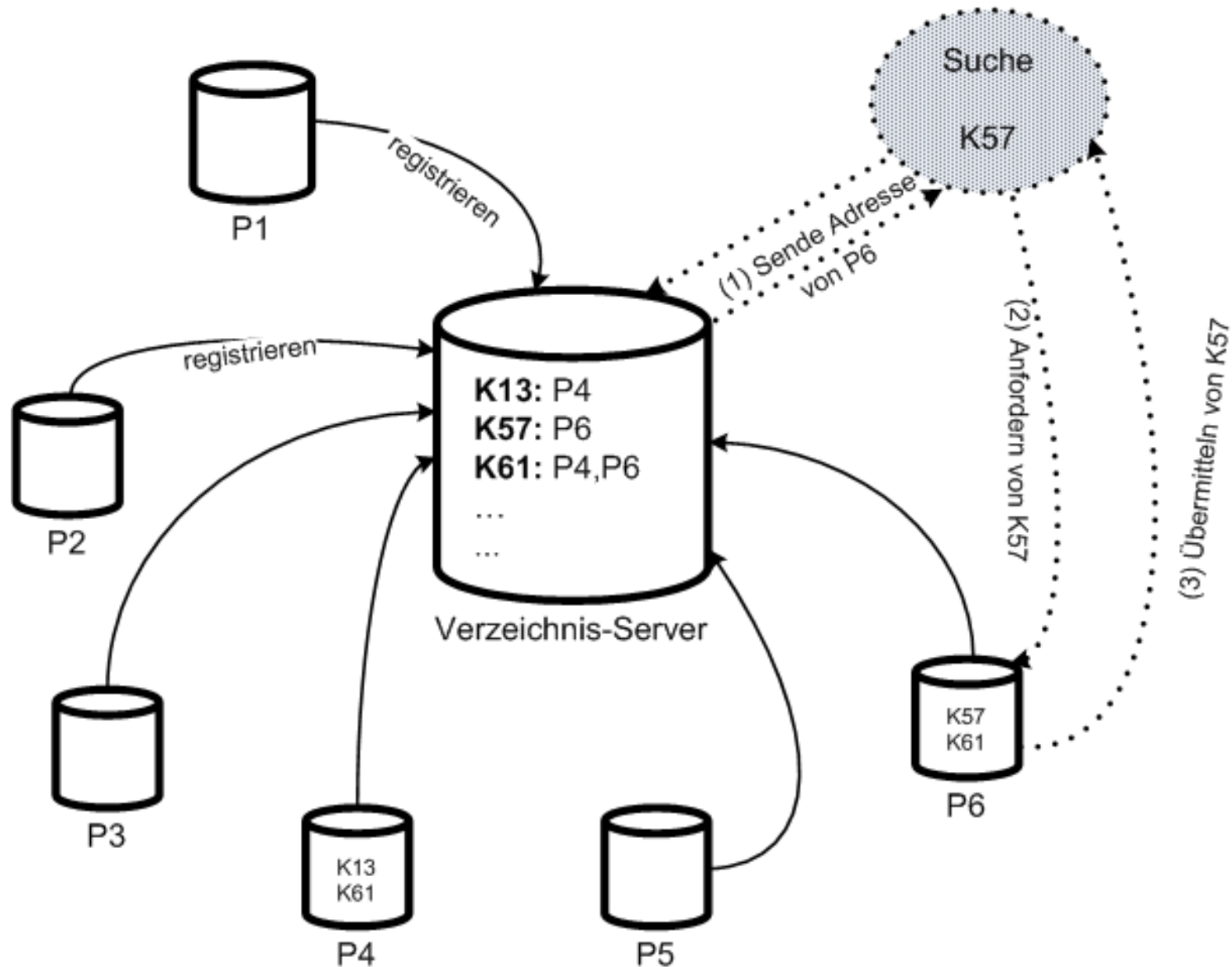


Peer to Peer-Informationssysteme

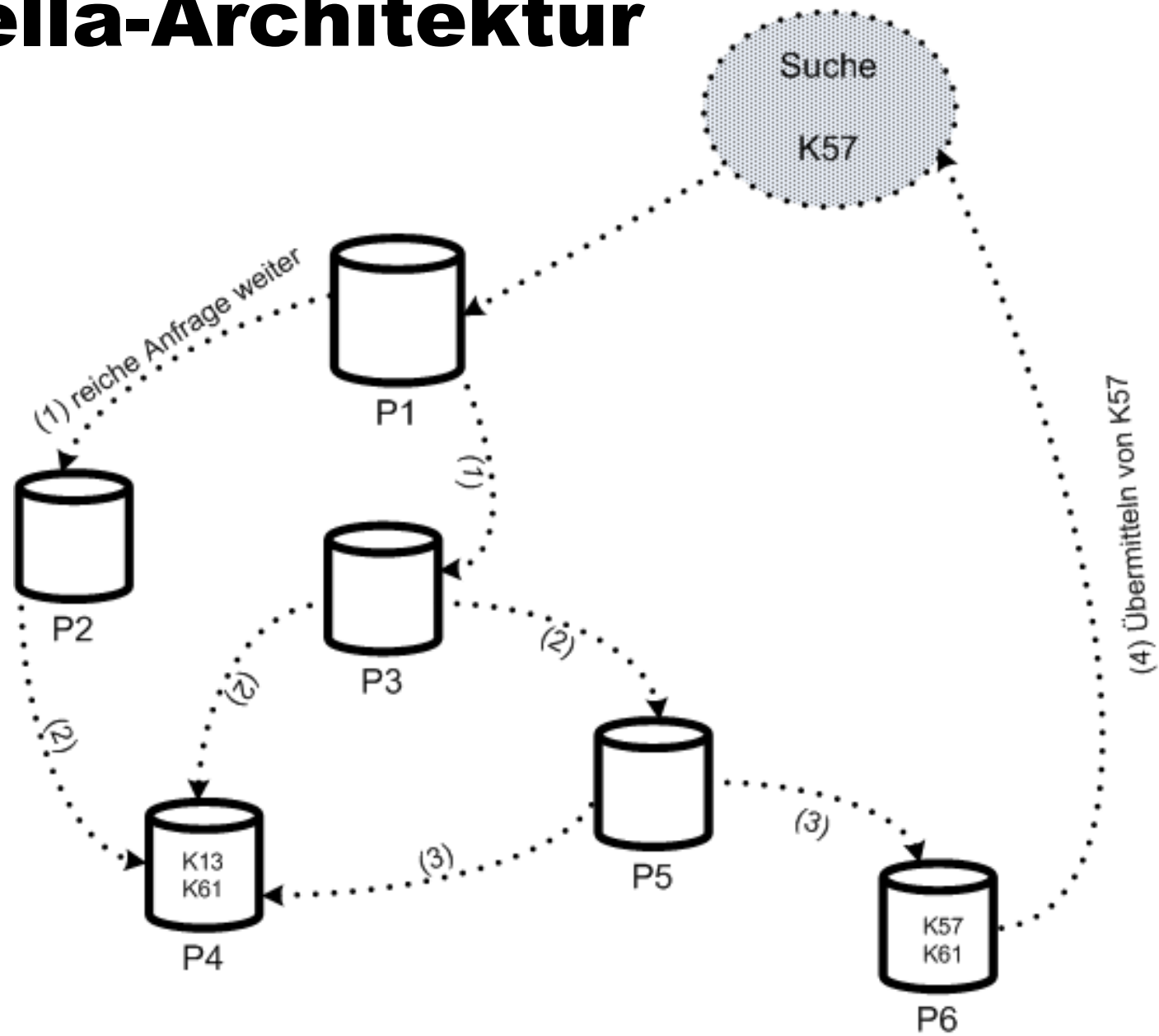
- Seti@Home
 - P2P number crunching

- Napster
 - P2P file sharing / Informationsmanagement

Napster-Architektur



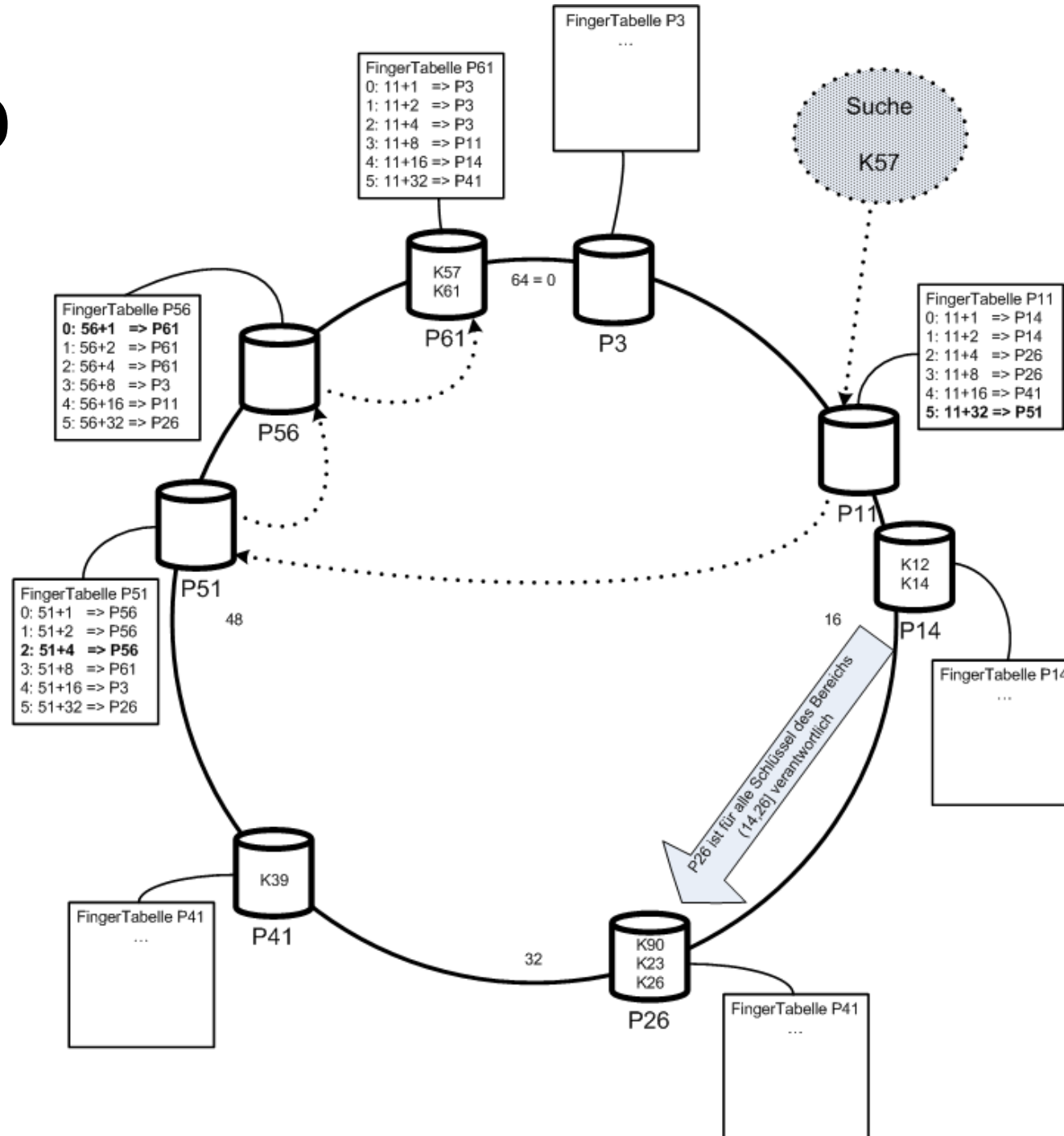
Gnutella-Architektur



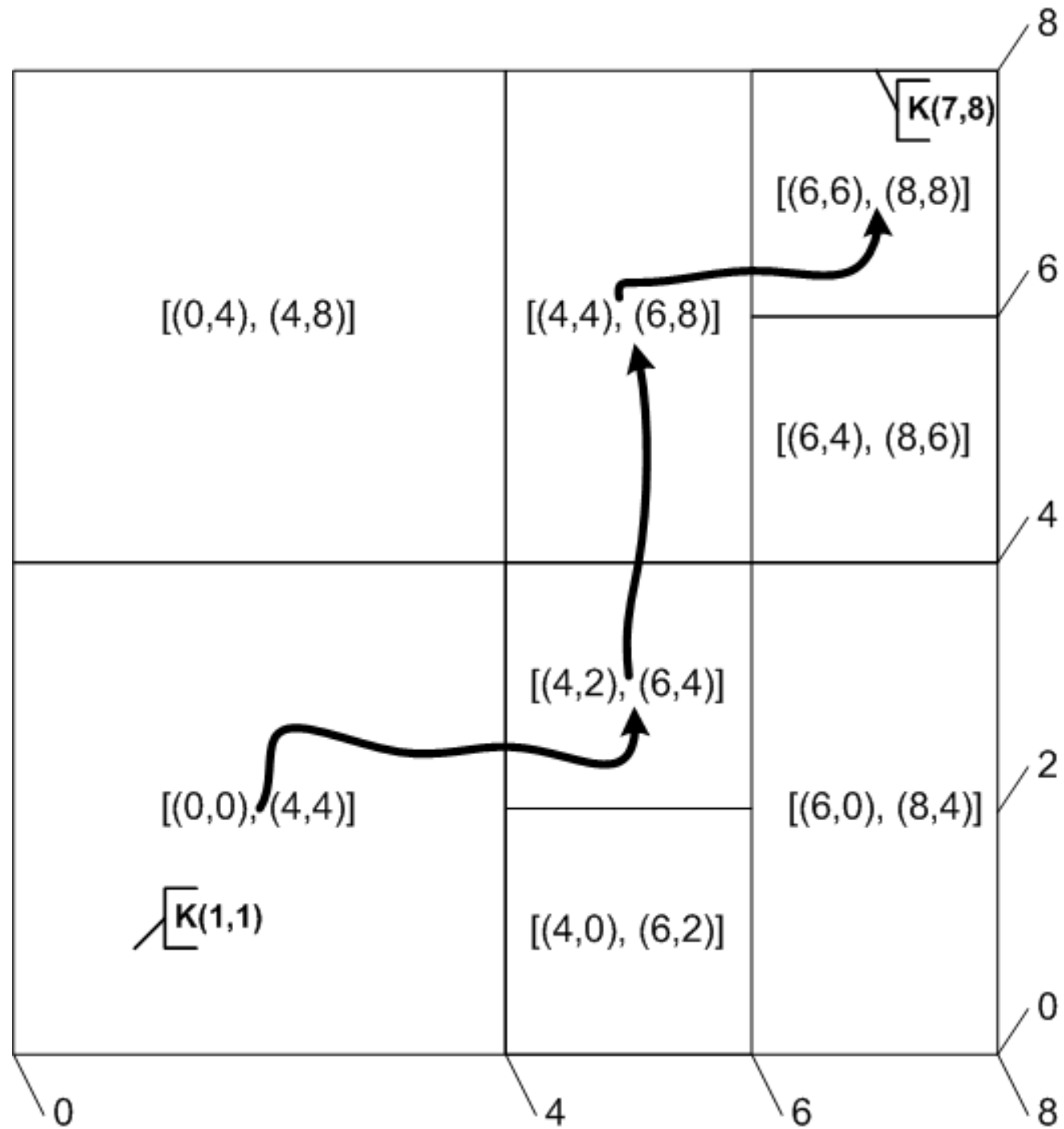
DHT: Distributed Hash Table

- Basieren auf „consistent hashing“
- Vollständige Dezentralisierung der Kontrolle
- Dennoch zielgerichtete Suche

CHORD



CAN



No-SQL Datenbanken

- Internet-scale Skalierbarkeit
- CAP-Theorem: nur 2 von 3 Wünschen erfüllbar
 - Konsistenz (Consistency)
 - Zuverlässigkeit/Verfügbarkeit (Availability)
 - Partitionierungs-Toleranz
- No-SQL Datenbanksysteme verteilen die Last innerhalb eines Clusters/Netzwerks
 - Dabei kommen oft DHT-Techniken zum Einsatz

Schnittstelle der No-SQL Datenbanken

- Insert(k,v)
- Lookup(k)
- Delete(k)

- Extrem einfach → effizient
- Aber: wer macht denn die Joins/Selektionen/...
 - → das Anwendungsprogramm

Konsistenzmodell: ~~C~~AP

- Relaxiertes Konsistenzmodell
 - Replizierte Daten haben nicht alle den neuesten Zustand
 - Vermeidung des (teuren) Zwei-Phasen-Commit-Protokolls
 - Transaktionen könnten veraltete Daten zu lesen bekommen
 - Eventual Consistency
 - Würde man das System anhalten, würden alle Kopien irgendwann (also eventually) in denselben Zustand übergehen
 - Read your Writes-Garantie
 - Tx leist auf jeden Fall ihre eigenen Änderungen
 - Monotonic Read-Garantie
 - Tx würde beim wiederholten Lesen keinen älteren Zustand als den vorher mal sichtbaren lesen

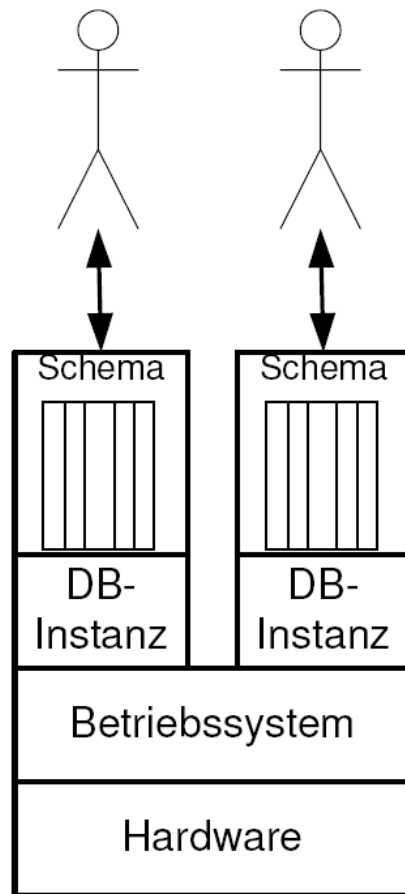
Systeme

- MongoDB
- Cassandra
- Dynamo
- BigTable
- Hstore
- SimpleDB
- S3

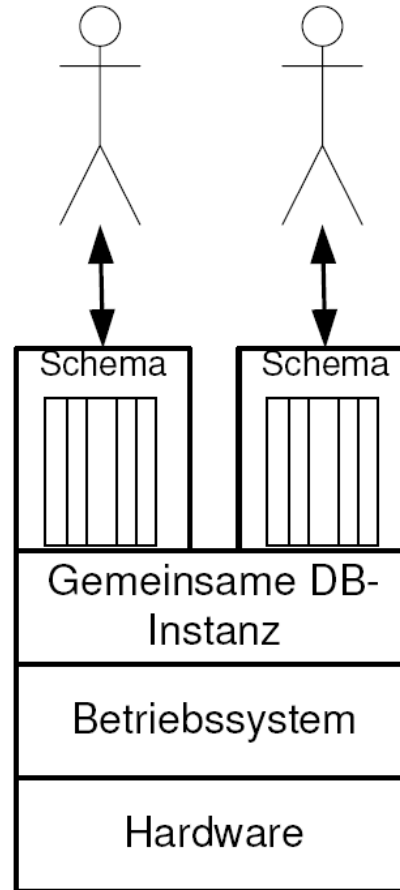
Multi-Tenancy / Cloud-Datenbanken

- **Infrastructure as a Service (IaaS):** Hierbei werden den Kunden de facto virtuelle Maschinen zur Verfügung gestellt, auf denen dann beliebige Software installiert werden könnte. Kunden könnten also auch existierende Anwendungen auf eine derartige virtuelle Maschine portieren. Insbesondere kann man natürlich „normale“ Datenbanksysteme installieren. Amazon Web Services ist ein typisches Beispiel einer IaaS, die aber zusätzlich auch nicht-relationale (also No-SQL) Datenbankfunktionalität durch die Systeme SimpleDB und S3 anbietet.
- **Platform as a Service (PaaS):** Hierzu zählen die Systeme Google AppEngine und Microsoft Azure, die reichhaltige Schnittstellen für die Neuentwicklung von Web-Applikationen bereitstellen. Hierzu dienen insbesondere die Datenspeicher, also Google's App Engine Datastore oder Microsoft's Azure Table Storage.
- **Software as a Service (SaaS):** Diese Systeme stellen komplexe, anwendungsspezifische Funktionalität zur Verfügung. Bekannteste Beispiele im betrieblichen Umfeld sind das Customer-Relationship-Management-System von Salesforce oder das umfassende betriebliche Anwendungssystem Business-By-Design von SAP. Diese Software wird nicht mehr bei den Nutzern installiert sondern von den Betreibern als „Hosting-Modell“ angeboten und von den Nutzern via Web-Schnittstellen zugegriffen.

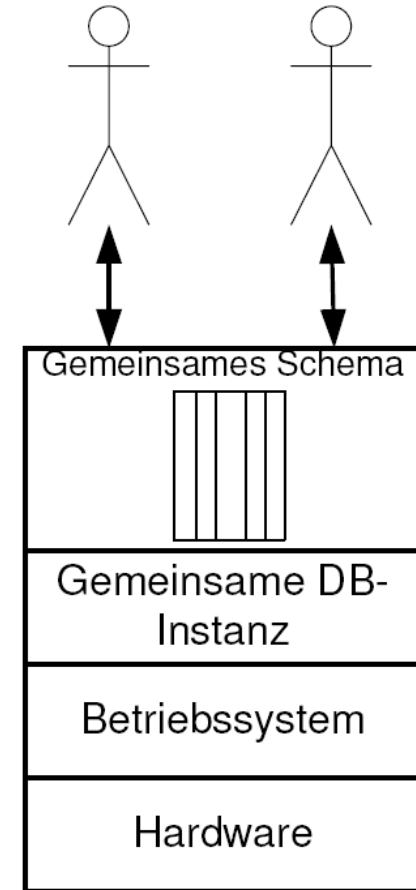
Multi-Tenancy Datenbankarchitekturen



(a)
gemeinsame
Maschine
(shared machine)



(b)
gemeinsames
Datenbanksystem
(shared process)



(c)
Gemeinsame
Relationen
(shared tables)

Shared Tables

Account			
Tenant	Aid	Name	...
17	1	Acme	...
17	2	Gump	...
35	1	Ball	...
42	1	Big	...

Die Applikationssoftware muss dann sicherstellen, dass jede Anfrage entsprechend umgeformt wird, je nachdem von welchem Nutzer sie empfangen wurde:

Tenant 35 \longrightarrow **select *
from Account
where Name like 'B%'** \longrightarrow **select *
from Account
where Name like 'B%'
and Tenant = 35**

Private Relationen

Account17			
Aid	Name	Hospital	Beds
1	Acme	St. Mary	135
2	Gump	State	1042

Account35	
Aid	Name
1	Ball

Account42		
Aid	Name	Dealers
1	Big	65

Erweiterungs-Relationen

Account_Ext			
Tenant	Row	Aid	Name
17	0	1	<i>Acme</i>
17	1	2	<i>Gump</i>
35	0	1	<i>Ball</i>
42	0	1	<i>Big</i>

Healthcare_Account			
Tenant	Row	Hospital	Beds
17	0	<i>St. Mary</i>	135
17	1	<i>State</i>	1042

Automotive_Account		
Tenant	Row	Dealers
42	0	65

Universal Relation

Universal							
Tenant	Table	Col1	Col2	Col3	Col4	Col5	eCol6
17	0	1	<i>Acme</i>	<i>St. Mary</i>	135	-	-
17	0	2	<i>Gump</i>	<i>State</i>	1042	-	-
35	1	1	<i>Ball</i>	-	-	-	-
42	2	1	<i>Big</i>	65	-	-	-

Zerlegung: Pivot-Relationen

Pivot_int				
Tenant	Table	Col	Row	Int
17	0	0	0	1
17	0	3	0	135
17	0	0	1	2
17	0	3	1	1042
35	1	0	0	1
42	2	0	0	1
42	2	2	0	65

Pivot_str				
Tenant	Table	Col	Row	Str
17	0	1	0	<i>Acme</i>
17	0	2	0	<i>St. Mary</i>
17	0	1	1	<i>Gump</i>
17	0	2	1	<i>State</i>
35	1	1	0	<i>Ball</i>
42	2	1	0	<i>Big</i>

Ballung logisch verwandter Werte: Chunk Tables

Account_Row			
Tenant	Row	Aid	Name
17	0	1	<i>Acme</i>
17	1	2	<i>Gump</i>
35	0	1	<i>Ball</i>
42	0	1	<i>Big</i>

Chunk_Row					
Tenant	Table	Chunk	Row	Int1	Str1
17	0	0	0	135	<i>St. Mary</i>
17	0	0	1	1042	<i>State</i>
42	2	0	0	65	-

Key/Value-Store

HBase Key/Value-Store		
Row Key	Account	Contact
17Act1	[name: <i>Acme</i> , hospital: <i>St. Mary</i> , beds: <i>135</i>]	
17Act2	[name: <i>Gump</i> , hospital: <i>State</i> , beds: <i>1042</i>]	
17Ctc1		[...]
17Ctc2		[...]
35Act1	[name: <i>Ball</i>]	
35Ctc1		[...]
42Act1	[name: <i>Big</i> , dealers: <i>65</i>]	

XML-basiertes Schema

Account			
Tenant	Aid	Name	Ext_XML
17	1	<i>Acme</i>	<pre><ext> <hospital>St. Mary</hospital> <beds>135</beds> </ext></pre>
17	2	<i>Gump</i>	<pre><ext> <hospital>State</hospital> <beds>1042</beds> </ext></pre>
35	1	<i>Ball</i>	
42	1	<i>Big</i>	<pre><ext> <dealers>65</dealers> </ext></pre>