

---

# **Non-Standard-Datenbanken**

Semistrukturierte Datenbanken und  
Volltextindizierung

Prof. Dr. Ralf Möller

**Universität zu Lübeck**

**Institut für Informationssysteme**



# Beispiel für ein Dokument

```
<bib>
  <book> <publisher> Addison-Wesley </publisher>
    <author> Serge Abiteboul </author>
    <author> <first-name> Rick </first-name>
      <last-name> Hull </last-name>
    </author>
    <author> Victor Vianu </author>
    <title> Foundations of Databases </title>
    <year> 1995 </year>
  </book>
  <book price="55">
    <publisher> Freeman </publisher>
    <author> Jeffrey D. Ullman </author>
    <title> Principles of Database and Knowledge Base Systems </title>
    <year> 1998 </year>
  </book>
</bib>
```

# Semistrukturierte Datenbanken: Überblick



- **Datenspezifikationsprache XML**

- Gedacht zur Datenkommunikation und –integration
- Begriff des XML-“Dokuments“
- Strukturprüfung möglich (DTD, XML-Schema)
- Repräsentation von XML-Daten in SQL möglich?



- **Anfragesprache XPath**

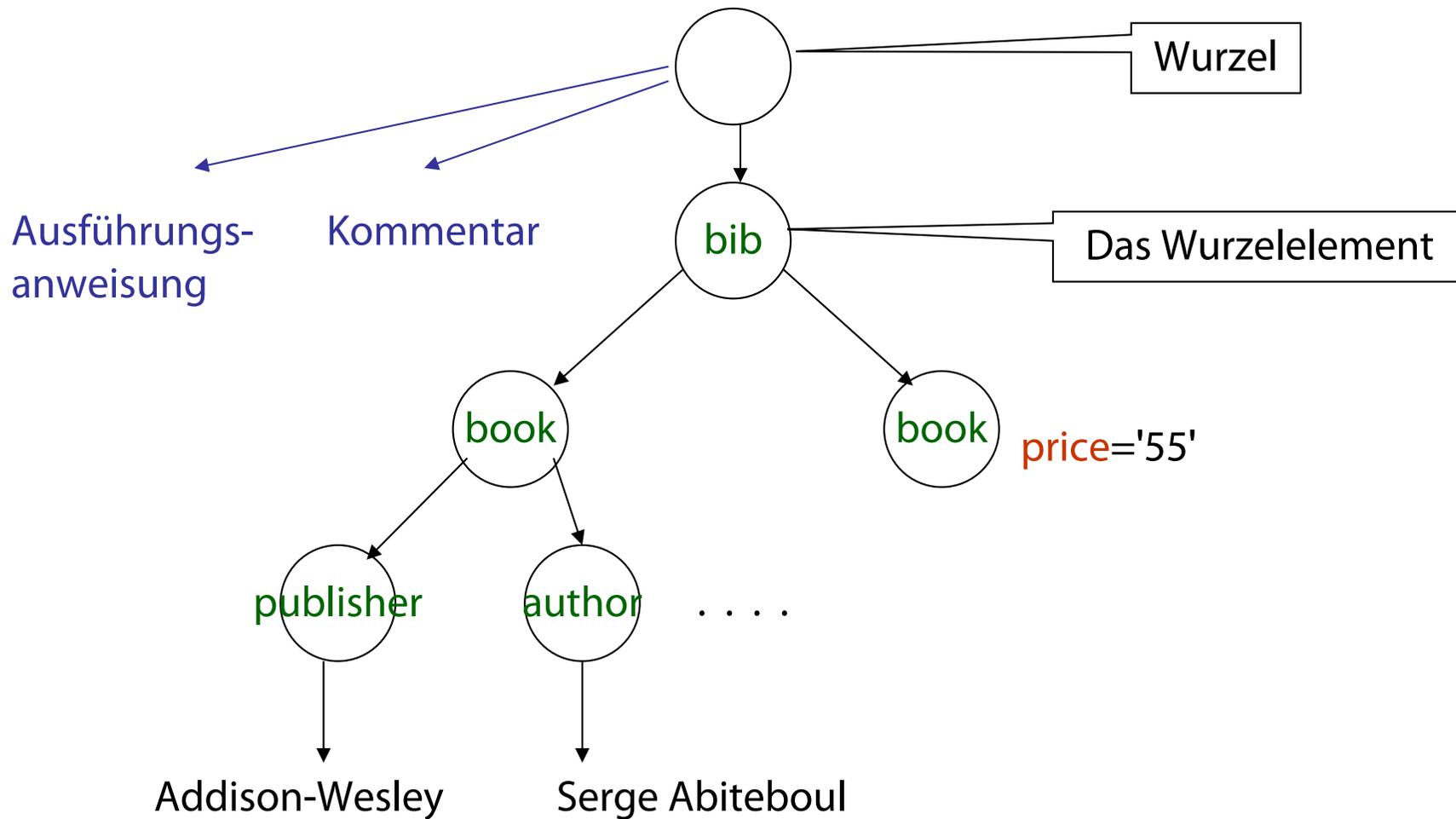
- W3C-Standard: <http://www.w3.org/TR/xpath> (11/99)
- Übersetzung nach SQL möglich?
- Einige Aspekte der optimierten Anfragebeantwortung



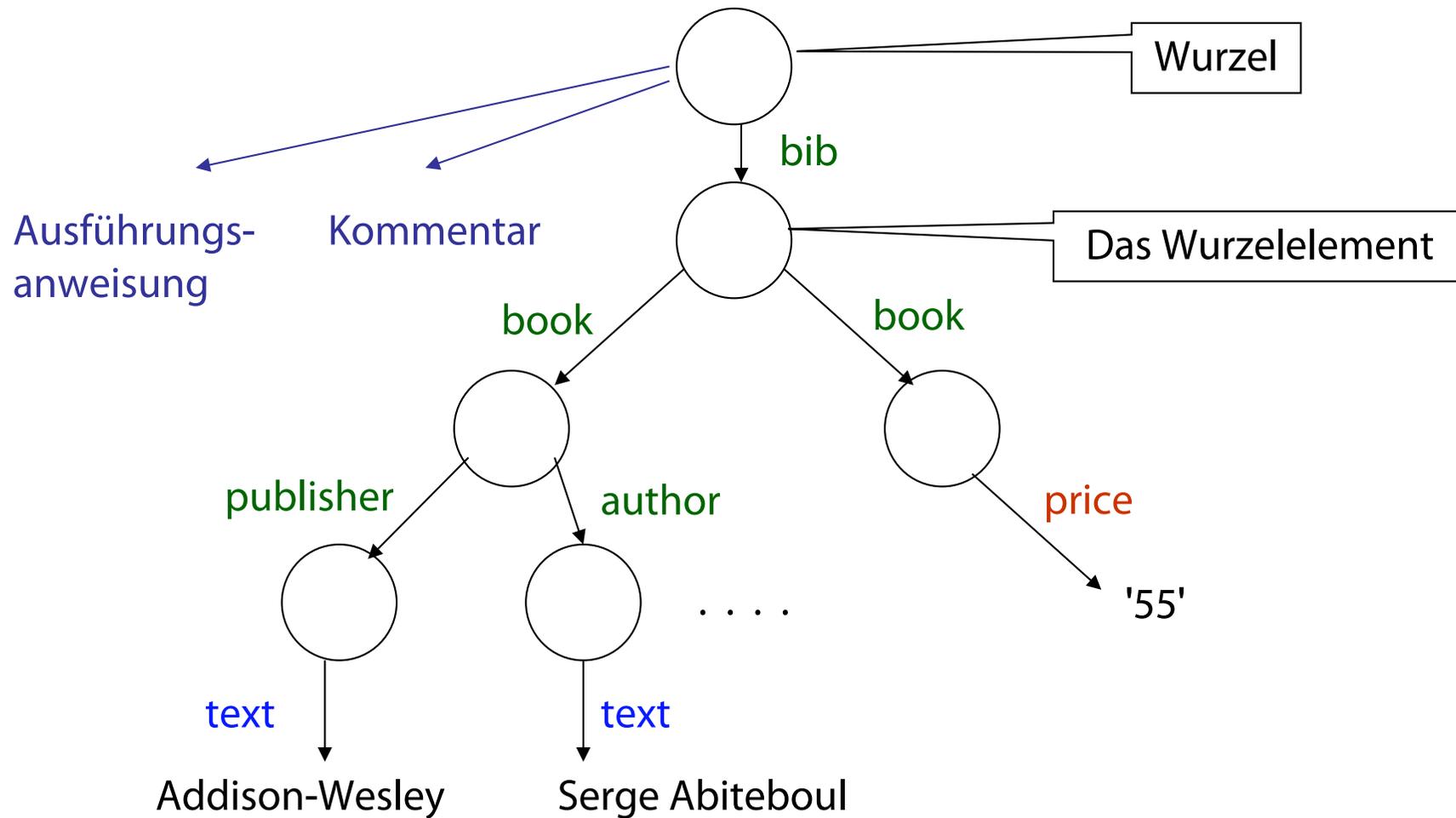
Acknowledgment: XPath-Präsentationen  
basieren auf Darstellungen von Dan Suciu,  
Univ. Washington

[http://www.cs.washington.edu/homes/suciu/  
COURSES/590DS](http://www.cs.washington.edu/homes/suciu/COURSES/590DS)

# Datenmodell für XPath



# Datenmodell für XPath: Duale Sicht



# Dokumenttypdefinitionen (DTDs)

---

- Beschreibung von akzeptierten Strukturen (optional)
- Kontextfreie Grammatik

```
<!ELEMENT paper (title, author*, year?)>  
<!ELEMENT author (firstName, lastName)>
```

- Siehe auch XML-Schema  
(ausdrucksmächtiger als DTDs,  
z.B. Minimal- und Maximalangaben von Nachfolgern)

# XPath: Einfache Ausdrücke

---

/bib/book/year

Ergebnis: <year> 1995 </year>  
<year> 1998 </year>

/bib/paper/year

Ergebnis: empty (keine Papiere in der bib)

# XPath: Hülloperator //

---

//author

Ergebnis:<author> Serge Abiteboul </author>  
<author> <first-name> Rick </first-name>  
          <last-name> Hull </last-name>  
</author>  
<author> Victor Vianu </author>  
<author> Jeffrey D. Ullman </author>

/bib//first-name

Ergebnis: <first-name> Rick </first-name>

# XPath: Platzhalter

---

`//author/*`

Ergebnis: `<first-name> Rick </first-name>`  
`<last-name> Hull </last-name>`

\* passt auf jedes Element

# XPath: Attributknoten

---

/bib/book/@price

Ergebnis: '55'

@price steht für eine Referenz auf den Attributwert

# XPath: Qualifizierung

---

`/bib/book/author[first-name]`

Ergebnis: `<author>` `<first-name>` Rick `</first-name>`  
`<last-name>` Hull `</last-name>`  
`</author>`

# XPath: Funktionen

---

`/bib/book/author/text()`

Ergebnis: Serge Abiteboul  
Victor Vianu  
Jeffrey D. Ullman

Rick Hull tritt nicht auf, da `firstname`, `lastname` vorhanden

`/bib/book/*[name() != author]`

Zugriff auf Namen eines Knotens mittels Funktion `name()`

# XPath: Weitere Qualifizierungen

---

`/bib/book/author[firstname][address[//zip][city]]/lastname`

Ergebnis: `<lastname> ... </lastname>`  
`<lastname> ... </lastname>`

# XPath: Weitere Qualifizierungen

---

`/bib/book[@price < '60']`

`/bib/book[author/@age < '25']`

`/bib/book[author/text()]`

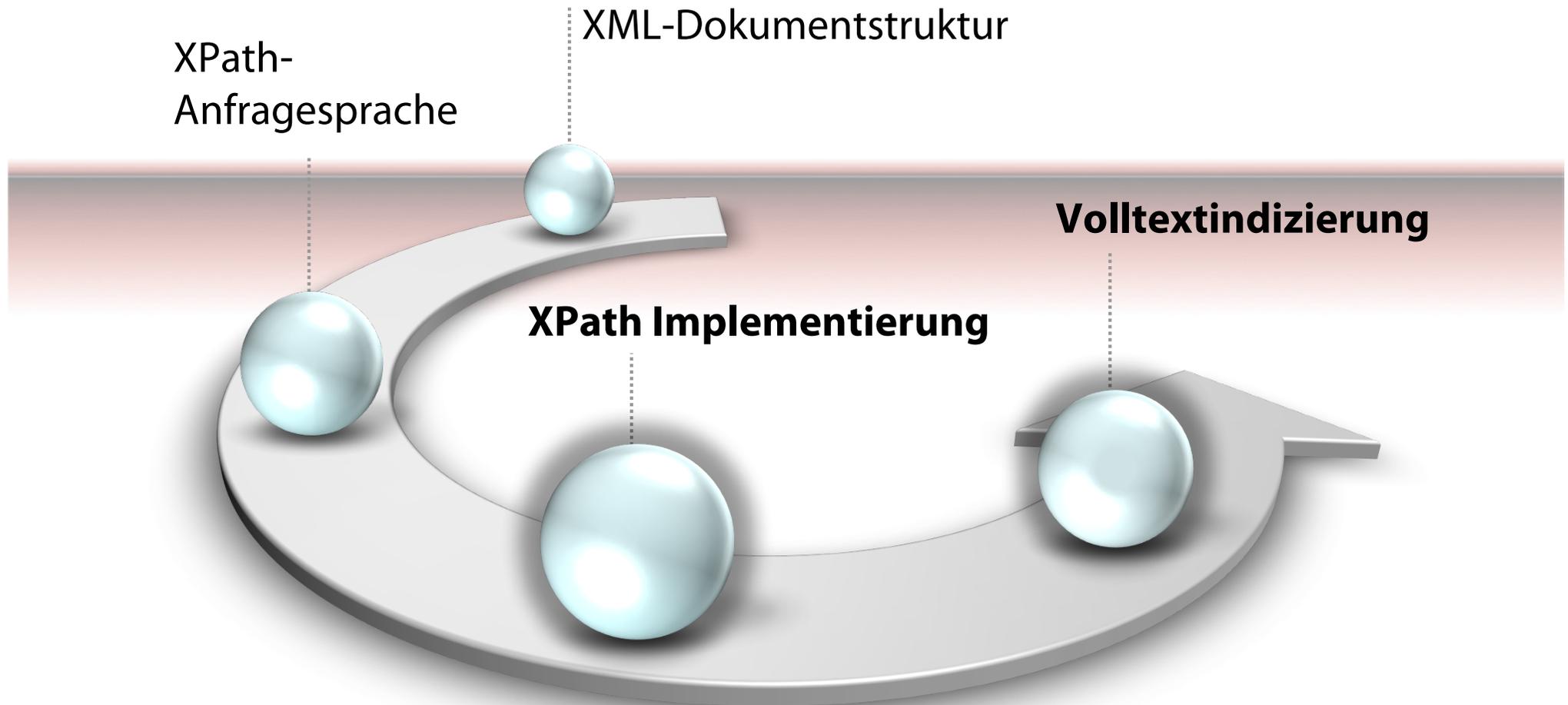
# XPath: Zusammenfassung

---

<code>bib</code>	passt auf ein <code>bib</code> Element
<code>*</code>	passt auf beliebiges Element
<code>/</code>	passt auf das <code>root</code> Element
<code>/bib</code>	passt auf <code>bib</code> Element unter <code>root</code>
<code>bib/paper</code>	passt auf <code>paper</code> in <code>bib</code>
<code>bib//paper</code>	passt auf <code>paper</code> in <code>bib</code> , in jeder Tiefe
<code>//paper</code>	passt auf <code>paper</code> in jeder Tiefe
<code>paper book</code>	passt auf <code>paper</code> oder <code>book</code>
<code>@price</code>	passt auf <code>price</code> Attribut
<code>bib/book/@price</code>	passt auf <code>price</code> Attribute in <code>book</code> , in <code>bib</code>
<code>bib/book/[@price&lt;'55']/author/lastname</code>	passt auf ...

# Semistrukturierte Datenbanken

---



Im nächsten Teil werden wir uns die letzten 2 Themen erarbeiten

# Speicher- und Zugriffstechniken für XML

---

## 1. Verwendung bestehender Techniken

- Abbildung auf relationale Datenbanken
  - Verwendung des physischen Datenmodells
  - Verwendung der Zugriffsoperatoren und deren Optimierungstechniken
- Abbildung des XML-Datenmodells auf relationale Schemata notwendig

## 2. Entwicklung neuer Techniken

- Neues physisches Datenmodell

# XML-Daten in Relationalen Datenbanken

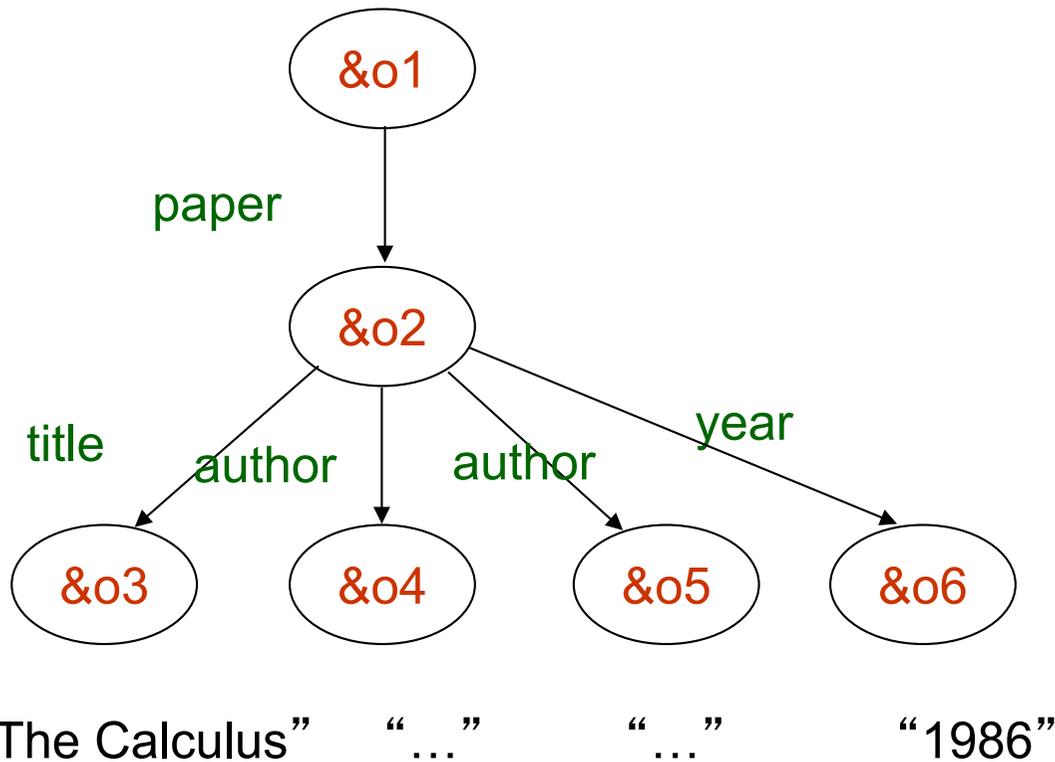
---

- Verwendung eines generischen Schemas  
D. Florescu, D. Kossmann, Storing and Querying XML Data using an RDBMS, Bulletin of the Technical Committee on Data Engineering, 22(3):27-34, September **1999**.
- Verwendung von DTDs zur Herleitung eines Schemas  
J. Shanmugasundaram, K. Tufte, C. Zhang, H. Gang, DJ. DeWitt, and JF. Naughton  
Relational databases for querying xml documents: Limitations and opportunities. Proceedings of the 25th International Conference on Very Large Data Bases, **1999**.
- Herleitung eines Schemas aus gegebenen Daten  
A. Deutsch, M. Fernandez, D. Suciu, Storing semistructured data with STORED, ACM SIGMOD Record 28 (2), 431-442, **1999**.
- Verwendung einer sog. Pfad-Relation  
M. Yoshikawa, T. Amagasa, S. Takeyuki, S. Uemura,  
XRel: A Path-Based Approach to Storage and Retrieval of XML Documents using Relational Databases, ACM TOIT Volume 1 (1), 110-141, **2001**.

# Generisches Schema: Ternäre Relation

**Ref**( Source, Label, Dest )

**Val**( Node, Value )



**Ref**

Source	Label	Dest
& o 1	paper	& o 2
& o 2	title	& o 3
& o 2	author	& o 4
& o 2	author	& o 5
& o 2	year	& o 6

**Val**

Node	Value
& o 3	The Calculus
& o 4	...
& o 5	...
& o 6	1986

# XML in ternären Relationen: Aufgabe

- Schema für SQL:

**Ref**( Source, Label, Dest )

**Val**( Node, Value )

- XPath: `/paper[year='1986']/author`

- SQL:

# Generisches Schema: Ternäre Relation?

---

- In der Praxis werden mehrere Tabellen benötigt:
  - Sonst Tabellen zu groß
  - Sonst Datentypen nicht unterstützt

**RefTag1**( Source, Dest )

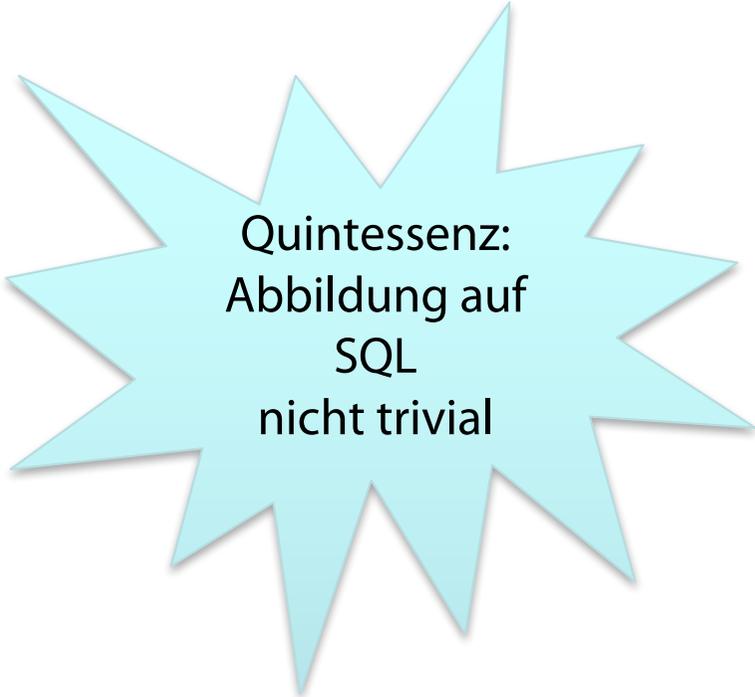
**RefTag2**( Source, Dest )

...

**IntVal**( Node, IntVal )

**RealVal**( Node, RealVal )

...



Quintessenz:  
Abbildung auf  
SQL  
nicht trivial

# DTDs zur Herleitung eines Schemas

---

- DTD (Kontextfreie Grammatik)

```
<!ELEMENT paper (title, author*, year?)>  
<!ELEMENT author (firstName, lastName)>
```

- Relationales Schema:

```
Paper( PID, Title, Year )  
PaperAuthor( PID, AID)  
Author( AID, FirstName, LastName )
```

[Shanmugasundaram et al. 1999 siehe oben]

# Aus DTD hergeleitetes Schema: Aufgabe

- Schema für SQL:

**Paper( PID, Title, Year )**

**PaperAuthor( PID, AID )**

**Author( AID, FirstName, LastName )**

- XPath:

`/paper[year='1986']/author`

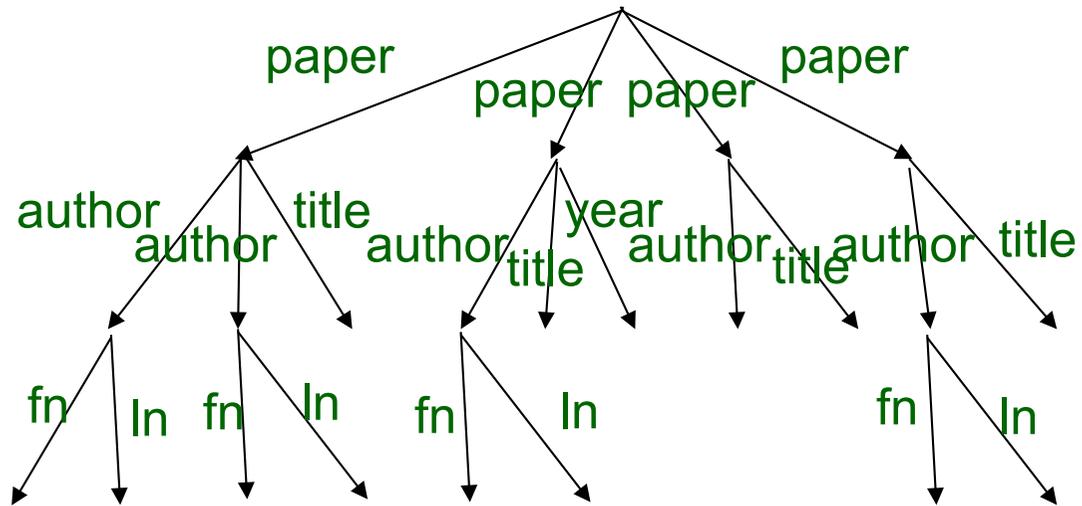
- SQL:

# Aus Daten hergeleitetes Schema

---

- (Große) XML Dokumente
- Kein Schema bzw. DTD
- Problem: Finde ein “gutes” relationales Schema
- NB: Selbst wenn DTD gegeben ist, kann die Ausdrucksstärke zu gering sein:
  - Z.B. wenn eine Person 1-3 Telefonnummer hat, steht trotzdem:  
`phone*`

# Aus Daten hergeleitetes Schema



## Paper1

auth	fn1	ln1	fn2	ln2	title	year
or						
X	X	X	X	X	X	.
X	X	X	.	.	X	X
X	X	X	.	.	X	.

## Paper2

author	title
X	X

# Aus Daten hergeleitetes Schema: Aufgabe

- Schema für SQL:

**Paper1**( fn1, ln1, fn2, ln2, title, year )  
**Paper2**( author, title )

- XPath:

/paper[year='1986']/author

- SQL:

# Pfad-Relations-Methode

---

- Speicherung von Pfaden als Zeichenketten
- Xpath-Ausdrücke werden durch SQL **like** umgesetzt (vgl. auch **contains**)
- Das Prozentzeichen '%' steht für eine beliebige Zeichenkette mit 0 oder mehr Zeichen

```
SELECT * FROM Versicherungsnehmer  
WHERE Ort LIKE '%alt%';
```

- Der Unterstrich '\_' steht für ein beliebiges einzelnes Zeichen, das an der betreffenden Stelle vorkommen soll.

# Pfad-Relations-Methode

Path

pathID	Pathexpr
1	/bib
2	/bib/paper
3	/bib/paper/author
4	/bib/paper/title
5	/bib/paper/year
6	/bib/book/author
7	/bib/book/title
8	/bib/book/publisher

Ein Eintrag für jeden vorkommenden Pfad

Annahme: Nicht zu viele verschiedene Pfadbezeichner notwendig

# Pfad-Relations-Methode

---

## Element

NodeID	pathID	ParentID
1	1	-
2	2	1
3	3	2
4	3	2
5	3	2
6	3	2
7	4	2
8	2	1
...		

Eine Eintrag für jeden Knoten in der Datenbasis  
Recht große Tabelle (Baum der Höhe  $h$  hat max.  $2^h$  Blätter)

# Pfad-Relations-Methode

---

Val  
text()

NodeID	Val
3	Smith
4	Vance
5	Tim
6	Wallace
7	The Best Cooking Book Ever
6	3
7	4
8	2
...	

Ein Eintrag für jedes Blatt in der Datenbasis  
und jedes Attribut  
Recht große Tabelle

# Pfad-Relations-Methode: Aufgabe

- Schema wie oben vereinbart

- XPath: `/bib/paper[year='1986']//figure`

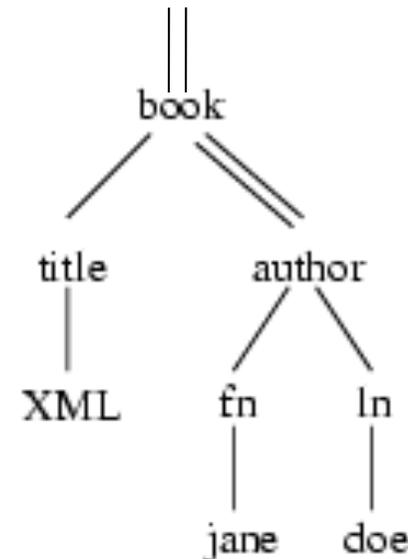
- SQL:

# XPath vs. XQuery

---

- 13 Navigationsmöglichkeiten in XPath:  
nicht nur Nachfolger
- XQuery: Nicht nur Pfad-,  
sondern **Zweig-basierte Anfragen**:

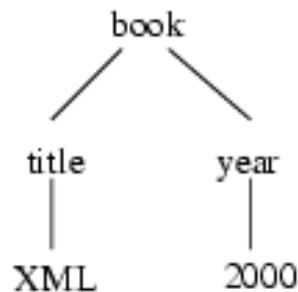
Finde das Publikationsjahr  
aller Bücher  
über "XML" geschrieben  
von "Jane Doe".



# Frühere Ansätze

- Basierend auf **binären Joins** [Zhang 01, Al-Khalifa 02].
- Umfassende SQL-Optimierung nötig
  - Zwischenresultate können groß sein
  - SQL-Maschinen überfordert

Join mit  
Values-  
Tabelle



- ((book ⋈ title) ⋈ XML) ⋈ (year ⋈ 2000)
  - (((book ⋈ year) ⋈ 2000) ⋈ title) ⋈ XML
- Viele andere Möglichkeiten...

[Zhang 01] C. Zhang, J. Naughton, D. DeWitt, Q. Luo, G. Lohman, On Supporting Containment Queries in Relational Database Management Systems, In SIGMOD 2001, pp. 425-436, **2001**

[Al-Khalifa 02] Shurug Al-khalifa, Jignesh M. Patel, H. V. Jagadish, Divesh Srivastava, Nick Koudas & Yuqing Wu. Structural joins: A Primitive for Efficient XML Query Pattern Matching. In Proc. ICDE, pages 141–152, **2002**



# BLAS: An Efficient XPath Processing System

---

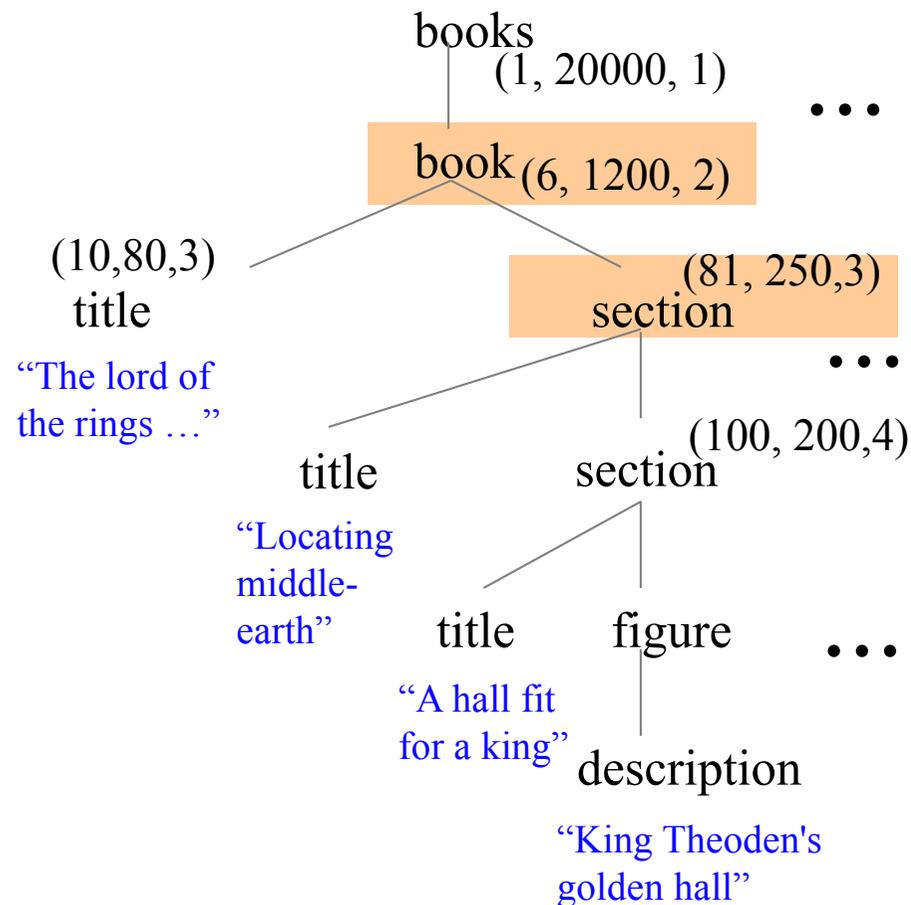
- **Bi-LA**beling based XPath processing **S**ystem
  - D-Beschriftung
    - **<Start, End, Ebene>** (minimale Angaben)
    - Aufbau eines B-Baums zur Unterstützung von Vorgänger/Nachfolger-Anfragen
  - P-Beschriftung
    - basiert auf XPRESS → eine XML Datenkompressionstechnik, die eine arithmetische Kodierung für Pfadbezeichner verwendet
  - Anfragebeantwortung über komprimierten Dokumenten unter Verwendung der D- und P-Beschriftungen

J.-K. Min, M.-J. Park, and C.-W. Chung. XPRESS: A queryable compression for XML data. In Proceedings of SIGMOD, **2003**.

BLAS: An efficient XPath processing system, Y. Chen, S.B. Davidson, Y. Zheng, In Proceedings SIGMOD '04 Proceedings of the 2004 ACM SIGMOD International Conference on Management of data, 47-58, **2004**



# D-Beschriftung: Dynamische Intervallkodierung



- Beschriftung  $\langle \text{Start, Ende, Ebene} \rangle$  kann verwendet werden, um Vorgänger-Nachfolger-Beziehungen in einem Baum zu entdecken

D. DeHaan, D. Toman, M. Consens, and M.T. Ozsü. A comprehensive XQuery to SQL translation using dynamic interval encoding. In Proceedings of SIGMOD, 2001

J. Celko. Trees, Databases and SQL. DBMS, 7(10):48–57, 1994

# D-Beschriftungen

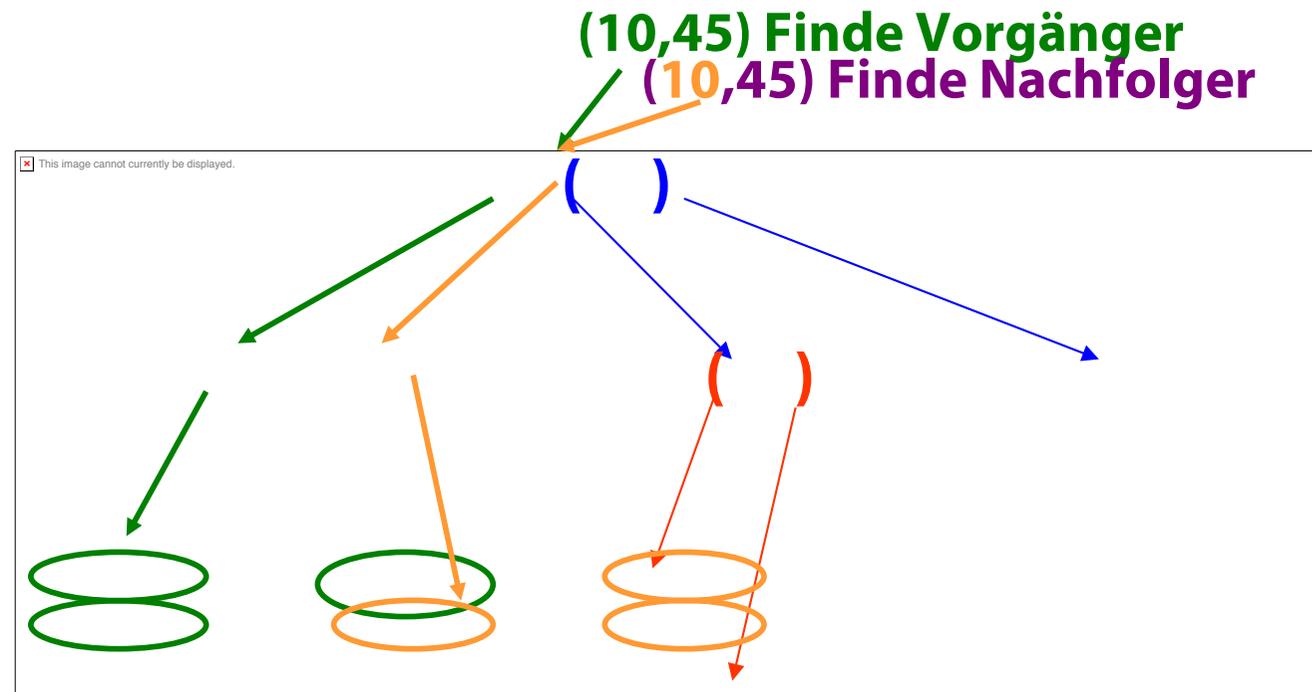
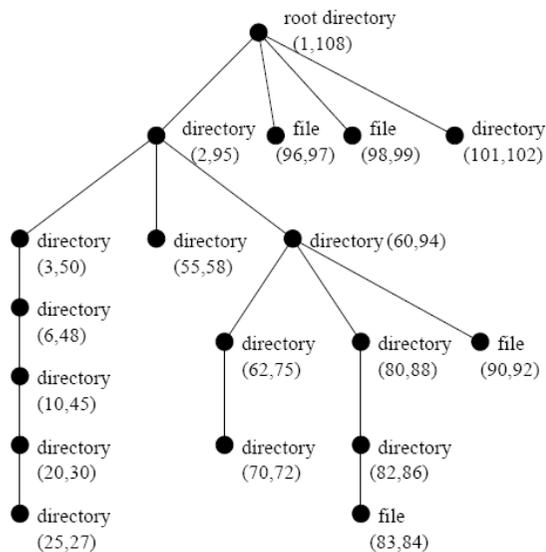
- Intervalle zur Nachfolgeranzeige:  $\langle \text{Start, Ende, Ebene} \rangle$
- Tripel  $\langle d_1, d_2, d_3 \rangle$  wird für jeden XML-Knoten  $n$  vergeben ( $n.d_1 \leq n.d_2$ )
  - $m$  ist Nachfolger von  $n$  gdw.  $n.d_1 < m.d_1$  und  $m.d_2 < n.d_2$
  - $m$  ist Kind von  $n$  gdw.  
 $m$  ist Nachfolger von  $n$  und  $n.d_3 + 1 = m.d_3$

- Verwendung mit Tag-basierter SQL-Tabellenrepräsentation

author	→	(1,1:150,1) (1,180:200,1) ...
book	→	(1,6:20,3) (1,22:40,3) ...
jane	→	(1,8:8,5) (1,43:43,5) ...
...		...
title	→	(1,65:67,3) ...
XML	→	(1,66:66,4) (2,140:140,6) ...
year	→	(1,61:63,2) (1,88:90,2) ...

# XB-Tree: Nachfolger und Vorgängernavigation

- Verwendung von Indexen zur Effizienzsteigerung
- XB-Bäume sind wie B-Bäume
  - Interne Knoten haben die Form [L:R], sortiert nach L
  - Eltern-Intervall enthält Kinder-Intervalle



# Zugriff über B-Baum: Clustered Index

---

- Blattknoten sind normalerweise nicht in sequentieller Reihenfolge auf der Festplatte gespeichert
- Dieses muss explizit angefordert werden (→ clustered index)

Ning Zhang, Varun Kacholia & M. Tamer Özsu. A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML. In Proc. ICDE, pages 54–63, **2004**

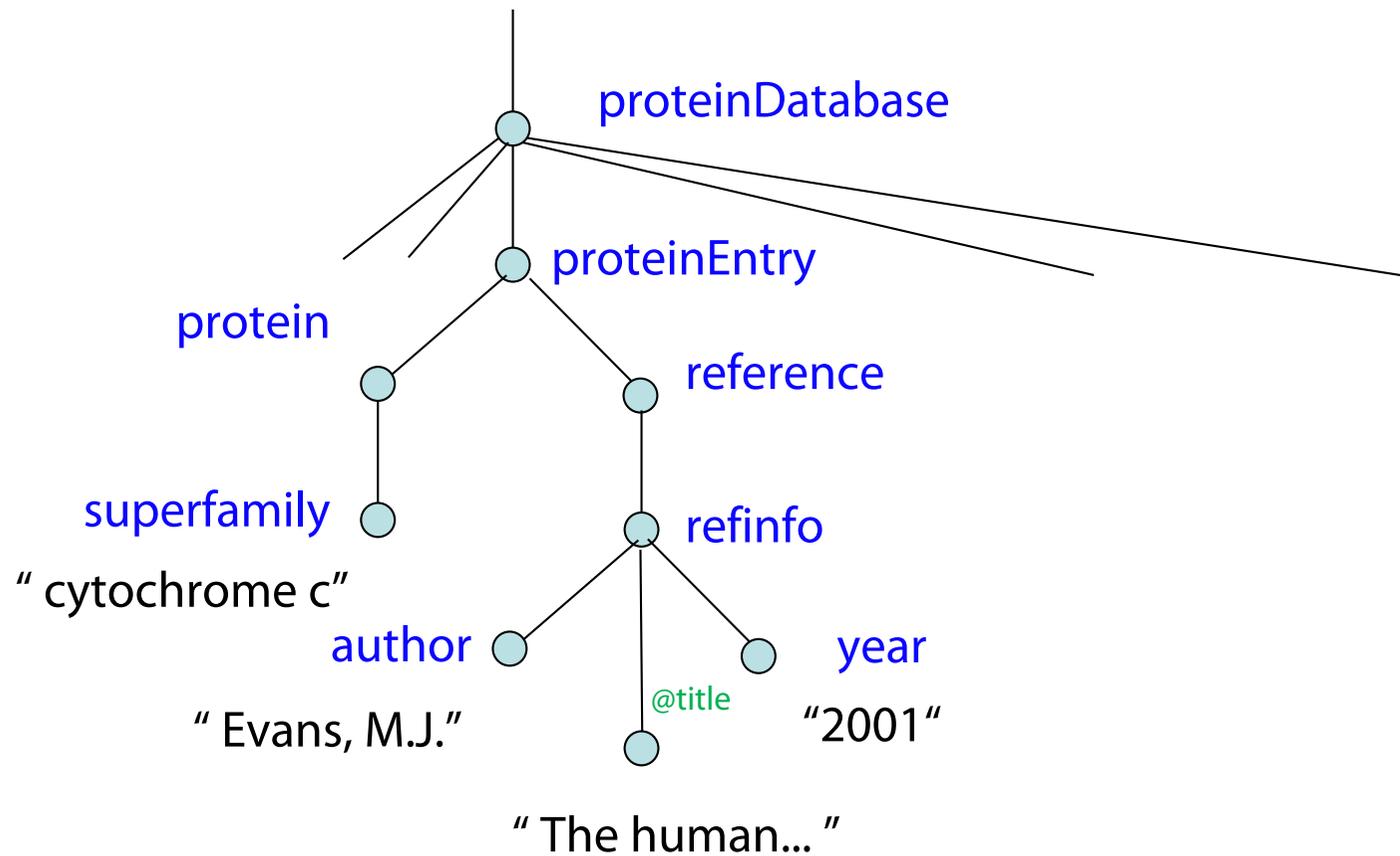
R. Bayer and E. M. McCreight, Organization and Maintenance of Large Ordered Indexes, Acta Informatica, vol. 1, no. 3, **1972**

# Ein Beispieldokument

---

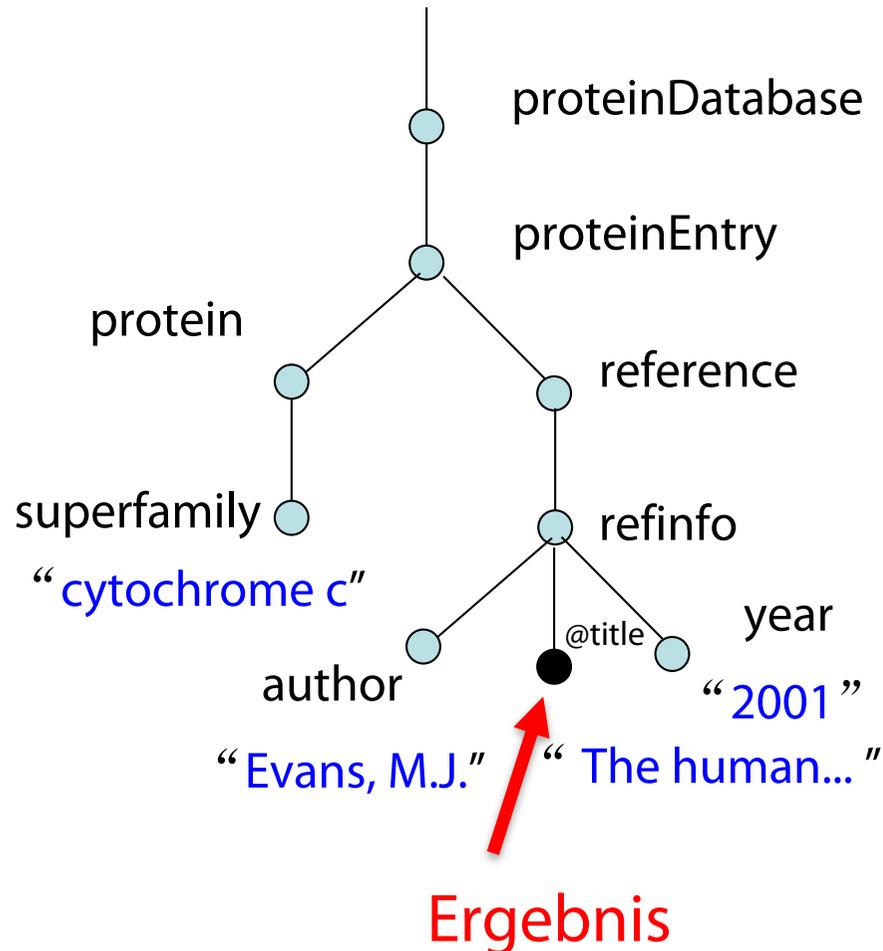
```
<proteinDatabase>
  <proteinEntry>
    <protein>
      <Name> cytochrome c</name>
      <classification>
        <superfamily>cytochrome c</superfamily>
      </classification>...
    </protein>
    <reference>
      <refinfo title="The human somatic cytochrome c gene">
        <authors>
          <author>Evans, M.J.</author>...
        </authors>
        <year>2001</year>
        ...
      </refinfo>
      ...
    </reference>
    ...
  </proteinEntry>
  ...
</proteinDatabase>
```

# Graphische Darstellung eines kleinen Teils



# Beispiel

XML-Dokument



Anfrage: `//proteinDatabase//refinfo/@title`

Suche alle Knoten  
proteinDatabase und refinfo

Seien proteinDatabase und refinfo zwei  
Relationen, die die jeweiligen Knoten  
repräsentieren, dann D-join ausführen

```
select refinfo.title
from proteinDatabase pDB, refinfo
where pDB.start < refinfo.start and
refinfo.end < pDB.end
```

# P-Beschriftungen

---

- Kind-Navigation sollte ebenfalls effizient implementiert werden
  - Beispiel  
`/proteinDatabase/proteinEntry/protein/name`
- Bisher: **Extrem viele Joins bei naiver SQL-Codierung**
- Aufgabe: Verbesserung der “/”-Auswertung für Pfade
- Fokus auf **Suffix-Pfadanfragen**:
  - Beispiel: `//protein/name`

# Definitionen / Notation

---

- Die Auswertung eines Pfadausdrucks  $P$  (geschrieben  $[P]$ ) gibt eine Sequenz (Menge) von Knoten in einem XML-Baum  $T$  zurück, die über  $P$  von der Wurzel des Baums  $T$  erreichbar sind.
- **Pfadausdruck**, **Pfad** und **Anfrage** werden synonym verwendet
- $P \sqsubseteq Q$  gdw.  $[P] \subseteq [Q]$  (Enthaltensein, Containment)
- $\text{Disjoint}(P, Q)$  gdw.  $[P] \cap [Q] = \emptyset$  (Disjunktheit)

# Definitionen (Forts.)

---

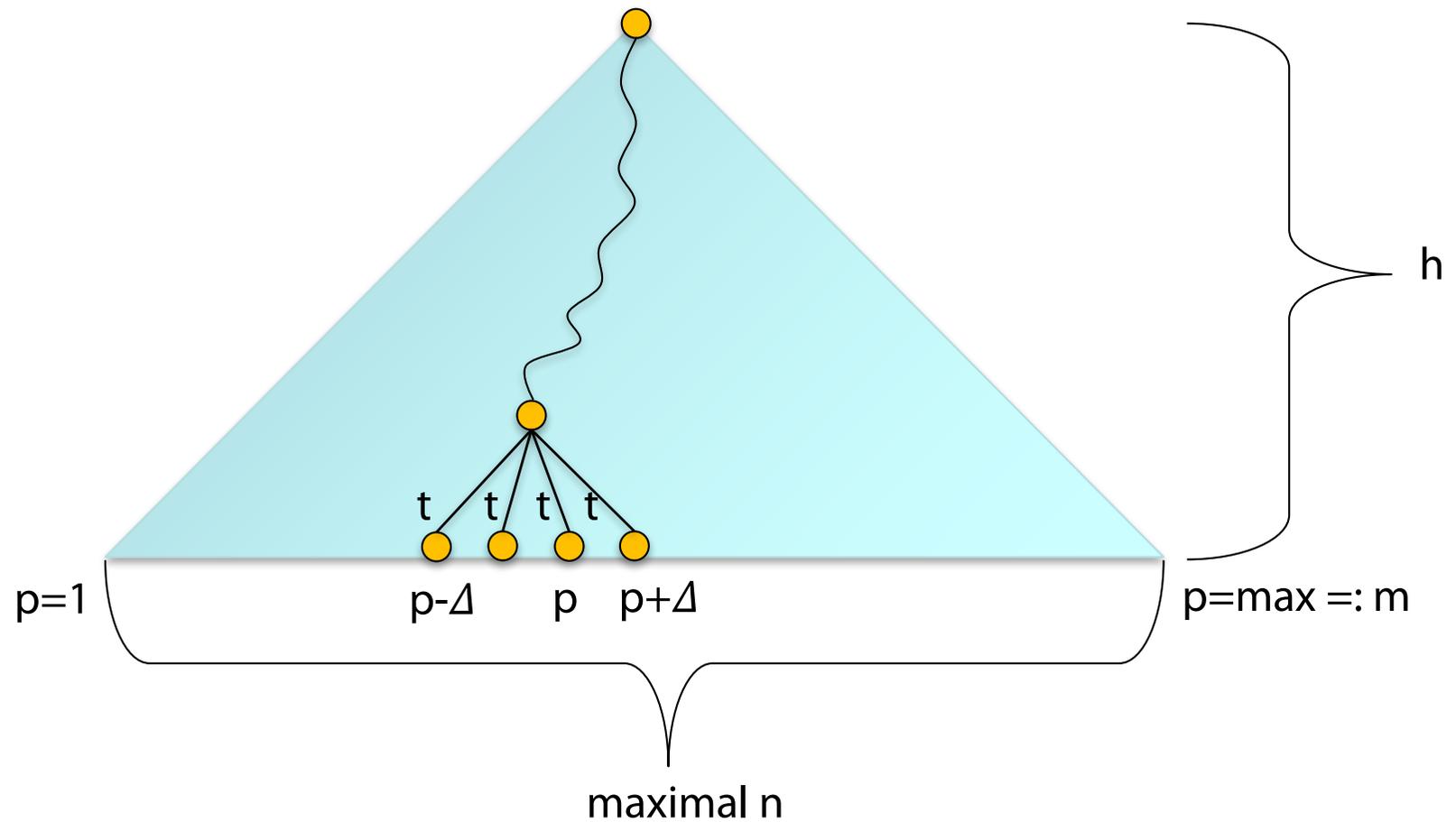
- **Suffix-Pfadausdruck**: Ein Pfadausdruck  $P$  mit einer Folge von Kind-Schritten  $/$ , ggf. mit einem Nachfolger-Schritt  $//$  am Anfang
- Beispiele:
  - $//\text{protein/name}$
  - $/\text{proteinDatabase/proteinEntry/protein/name}$
- $SP(n)$  : der eindeutige einfache Pfad  $P$  von der Wurzel bis zum Knoten  $n$  (SP = simple path)
- Die Auswertung eines Suffix-Pfadausdrucks  $Q$  ist die Bestimmung aller Knoten  $n$ , so dass  $SP(n) \sqsubseteq Q$

# P-Beschriftungen

---

- Weise jedem Knoten  $n$  eine Zahl  $p$  zu und jedem Suffix-Pfad ein Intervall  $[p_1, p_2]$ , so dass für Suffix-Pfade  $Q_1$  and  $Q_2$  gilt:
  - $Q_1 \sqsubseteq Q_2$  ( $Q_1$  in  $Q_2$  enthalten)  
falls  $Q_2.p_1 \leq Q_1.p_1$  und  $Q_1.p_2 \leq Q_2.p_2$
- Ein Knoten  $n$  ist in einem Suffix-Pfad  $Q$  enthalten falls  $Q.p_1 \leq SP(n).p_1 \leq Q.p_2$
- Sei  $Q$  ein Suffix-Pfad, dann gilt  $[Q] = \{n \mid Q.p_1 \leq n.p \leq Q.p_2\}$  wobei  $n.p = SP(n).p_1$

# P-Beschriftungen



# P-Beschriftung Beispiel

- Annahme: Längster Pfad:  $h=6$
- Sei die Maximalanzahl der Auszeichner (Tags) auf 99 festgelegt
- Wähle maximalen p-Wert  $m = 100^6 = 10^{12}$   
(wir wollen, dass  $m \geq (n+1)^h$ )
- Jedem Auszeichner (Tag) wird ein Bereich  $r$  zugewiesen:  
 $r_i = 0.01$  (bei 99 Auszeichnern also ausreichend viele)
- P-Beschriftung für jeden Suffix-Pfad bestimmen
- Beispiel  $P = /ProteinDatabase/ProteinEntry/protein/name$

Path expression	P-label
//name	$\langle 4 \times 10^{10}, 5 \times 10^{10} - 1 \rangle$
//protein/name	$\langle 4.03 \times 10^{10}, 4.04 \times 10^{10} - 1 \rangle$
//ProteinEntry/protein/name	$\langle 4.0302 \times 10^{10}, 4.0303 \times 10^{10} - 1 \rangle$
//ProteinDatabase/ProteinEntry/protein/name	$\langle 4.030201 \times 10^{10}, 4.030202 \times 10^{10} - 1 \rangle$
/ProteinDatabase/ProteinEntry/protein/name	$\langle 4.030201 \times 10^{10}, 4.03020101 \times 10^{10} - 1 \rangle$

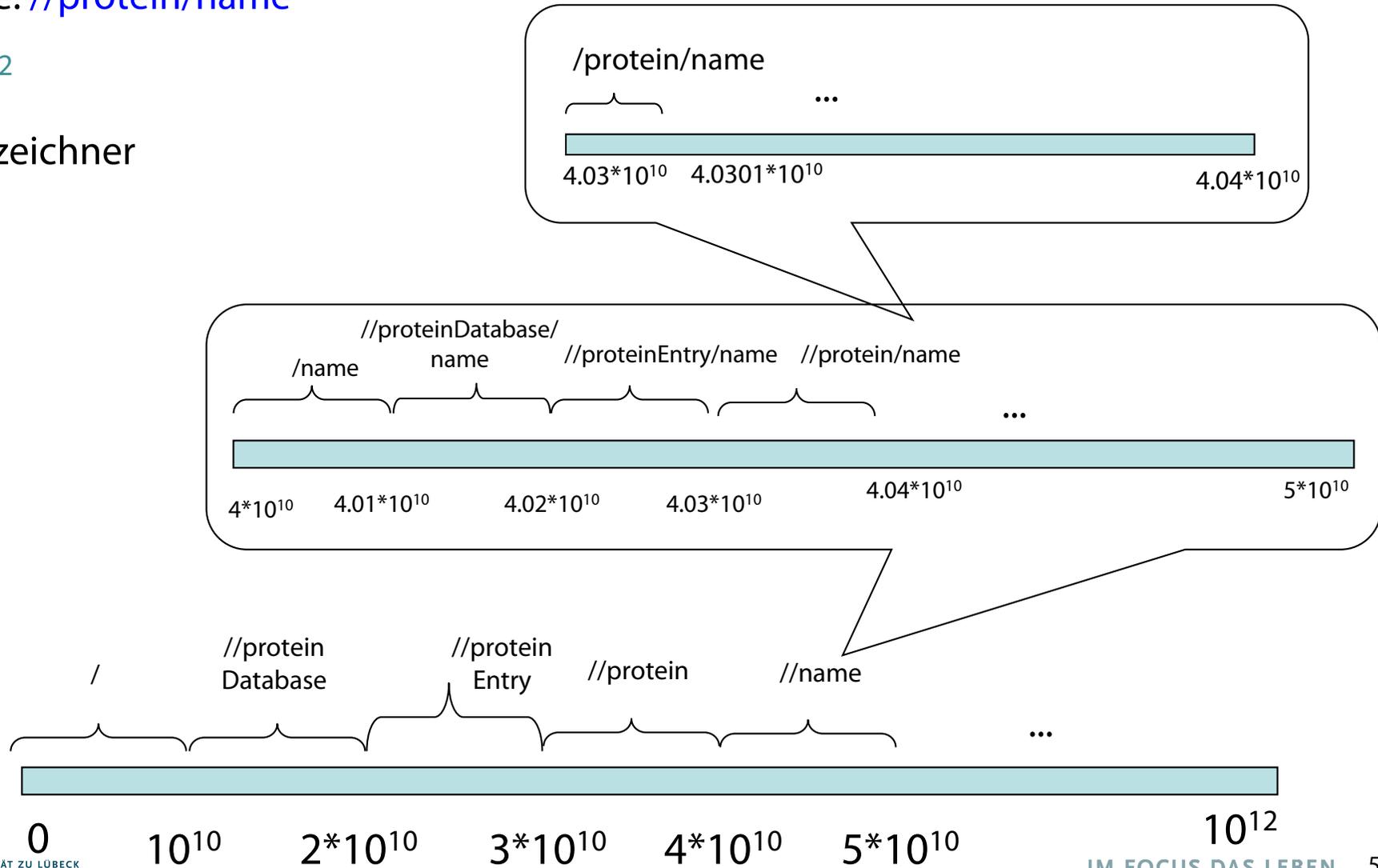
# P-Beschriftungen (Beispiel)

Anfrage: `//protein/name`

$m=10^{12}$

99 Auszeichner

$r_i = 0.01$



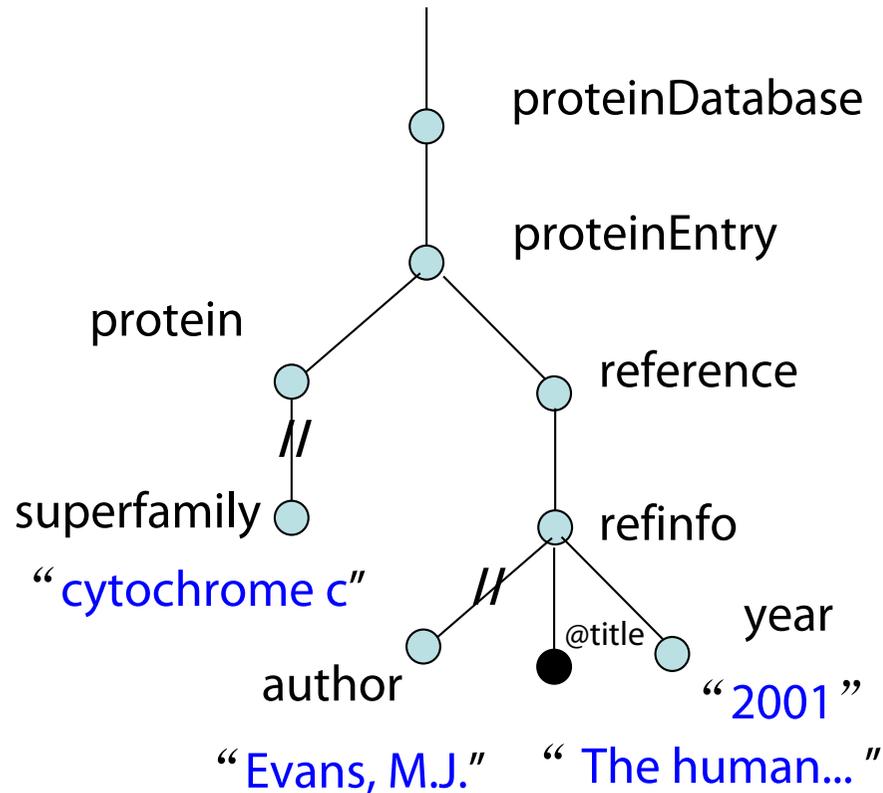
# Zum Nachvollziehen zuhause:

---

- Seien  $n$  Auszeichner gegeben  $(t_1, t_2, \dots, t_n)$ .
- Weise  $"/"$  einen Wert  $r_0$  und jedem Auszeichner  $t_i$  einen Wert  $r_i$  zu, so dass  $r_0 + r_1 + r_2 + \dots + r_n = 1$ .
- Setze  $r_i = 1/(n+1)$ .
- Definiere den Wertebereich der Zahlen in P-Beschriftungen als Integer in  $[0, m-1]$ ,  $m$  wird gewählt, so dass  $m \geq (n+1)^h$ , wobei  $h$  der längste Pfad im XML-Baum ist
- P-Beschriftung wie folgt:
  - Pfad  $//$  ist ein Intervall (P-label) von  $\langle 0, m-1 \rangle$  zugeordnet
  - Partitionierung des Intervalls  $\langle 0, m-1 \rangle$  in der Ordnung der Auszeichnungen proportional zum Abschnitt  $r_i$  von  $t_i$ , für jeden Pfad  $//t_i$  und jeden Abschnitt des Kind-Nachfolgers  $r_0$ .
  - Wir allozieren das Intervall  $\langle 0, m \cdot r_0 - 1 \rangle$  für  $"/"$  and  $\langle p_i, p_{i+1} \rangle$  für jedes  $t_i$ , so dass  $(p_{i+1} - p_i)/m = r_i$  und  $p_1/m = r_0$

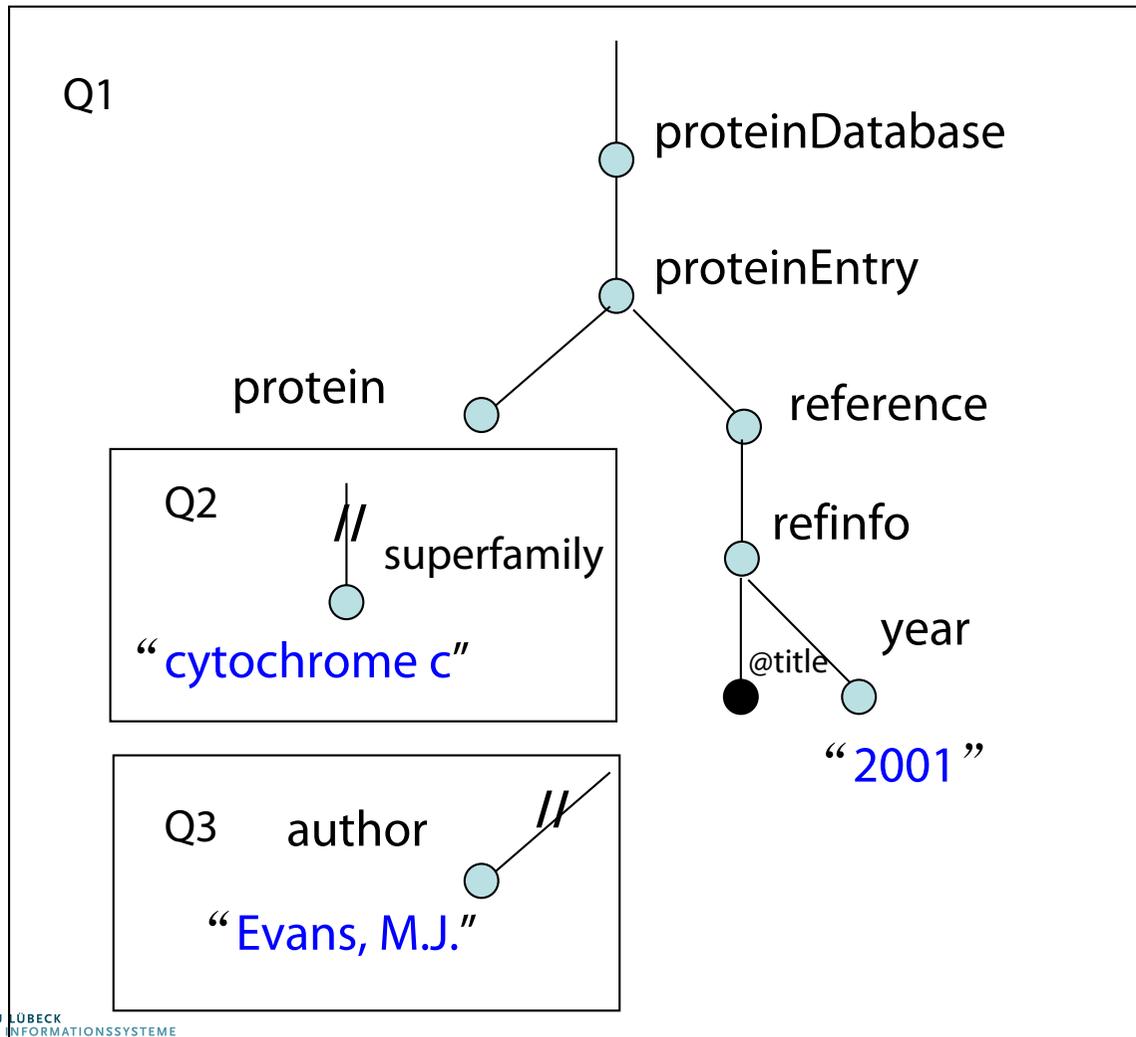
# Zweiganfragen (TWIG-Patterns)

Beispielanfrage (ähnlich zu vorigen Daten, beachte aber //)



# Ein Beispiel

- Aufspaltung:
  - Verzweigungselimination (B-Eliminierung)



$p//q \rightarrow p \text{ and } //q$

Tiefensuche

Spalte  $p//q$  in  $p \text{ and } //q$

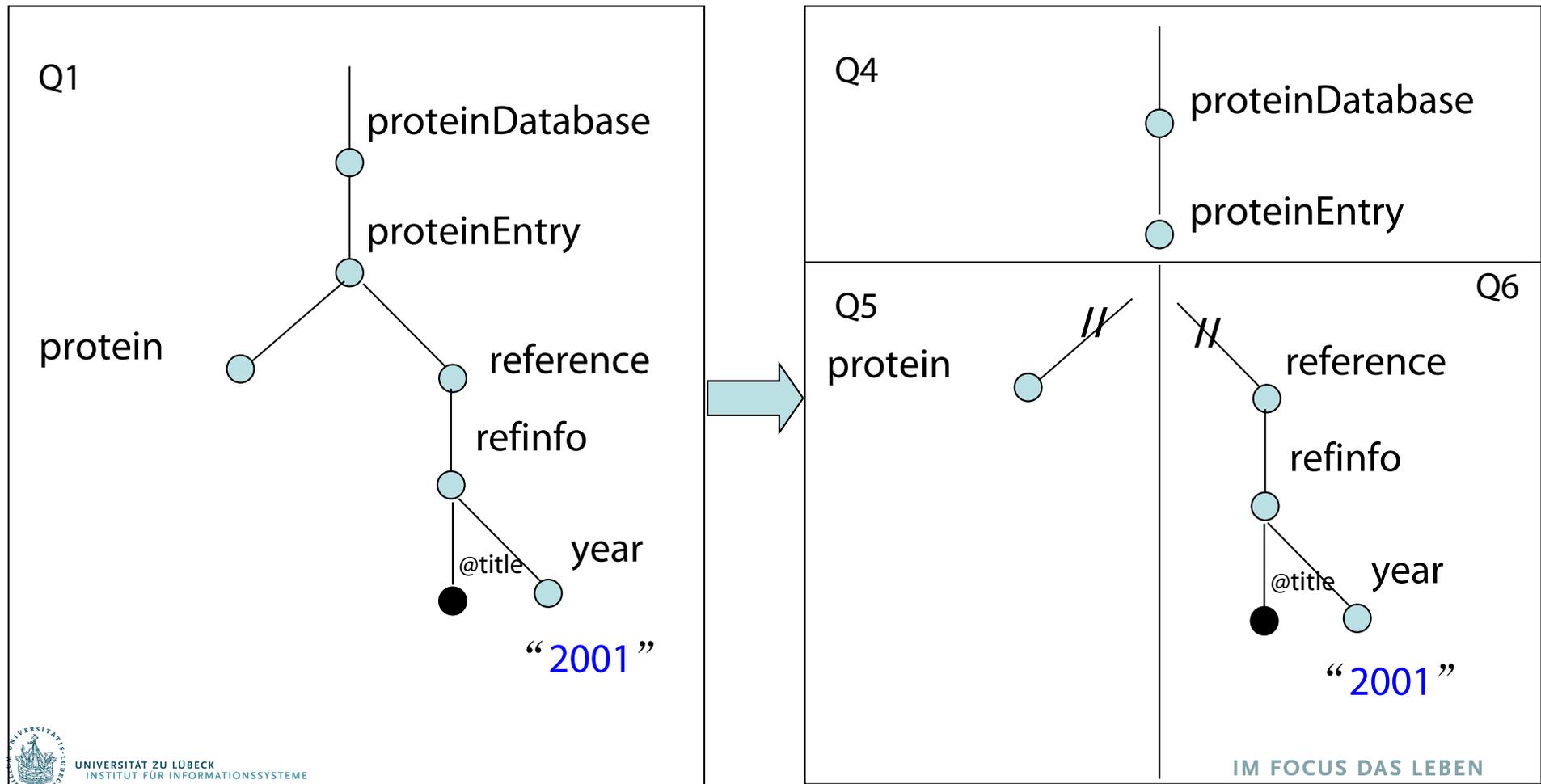
Eliminiere Verzweigung falls vorhanden, sonst evaluiere Q mit P-Beschriftungen

Join für Zwischenresultate unter Verwendung der D-Beschriftungen

# Ein Beispiel

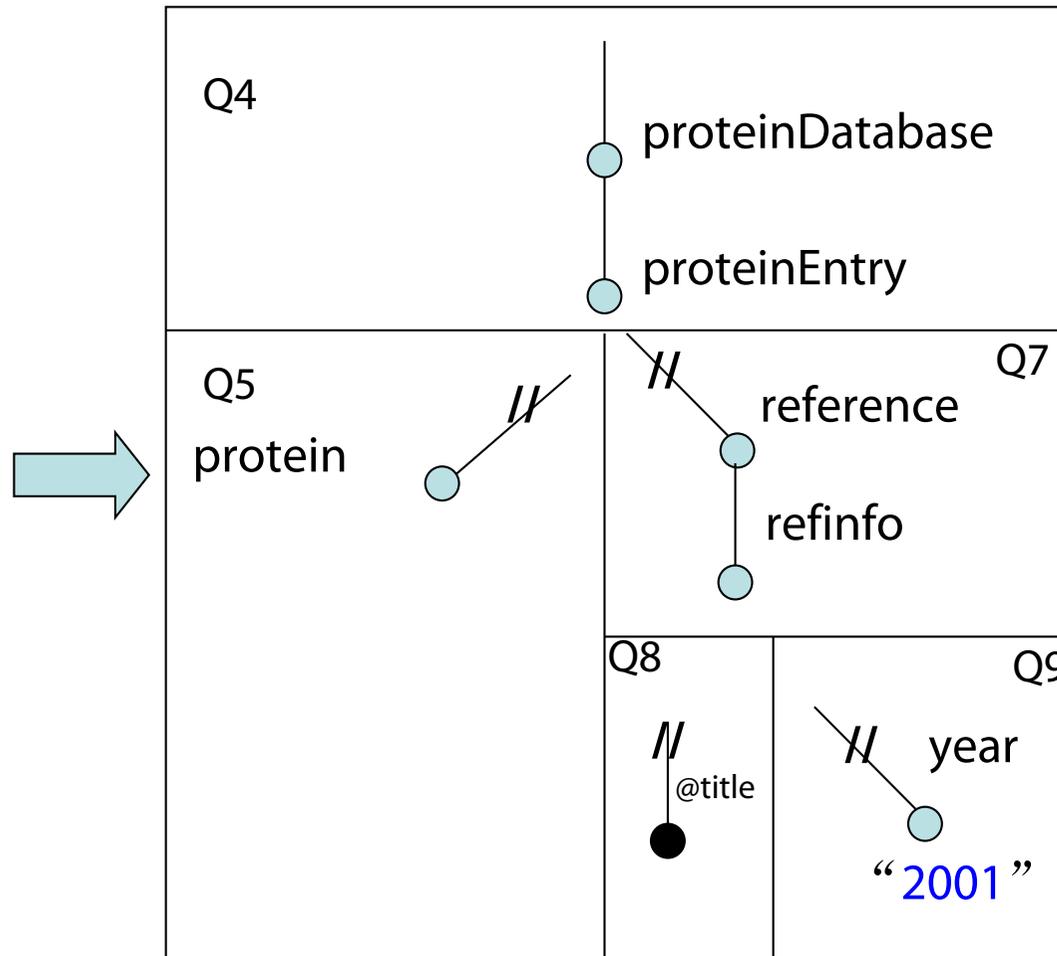
- Verzweigungselimination

$P[q_1, q_2, \dots, q_i] / r \rightarrow p, //q_1, //q_2, \dots, //q_i, //r$

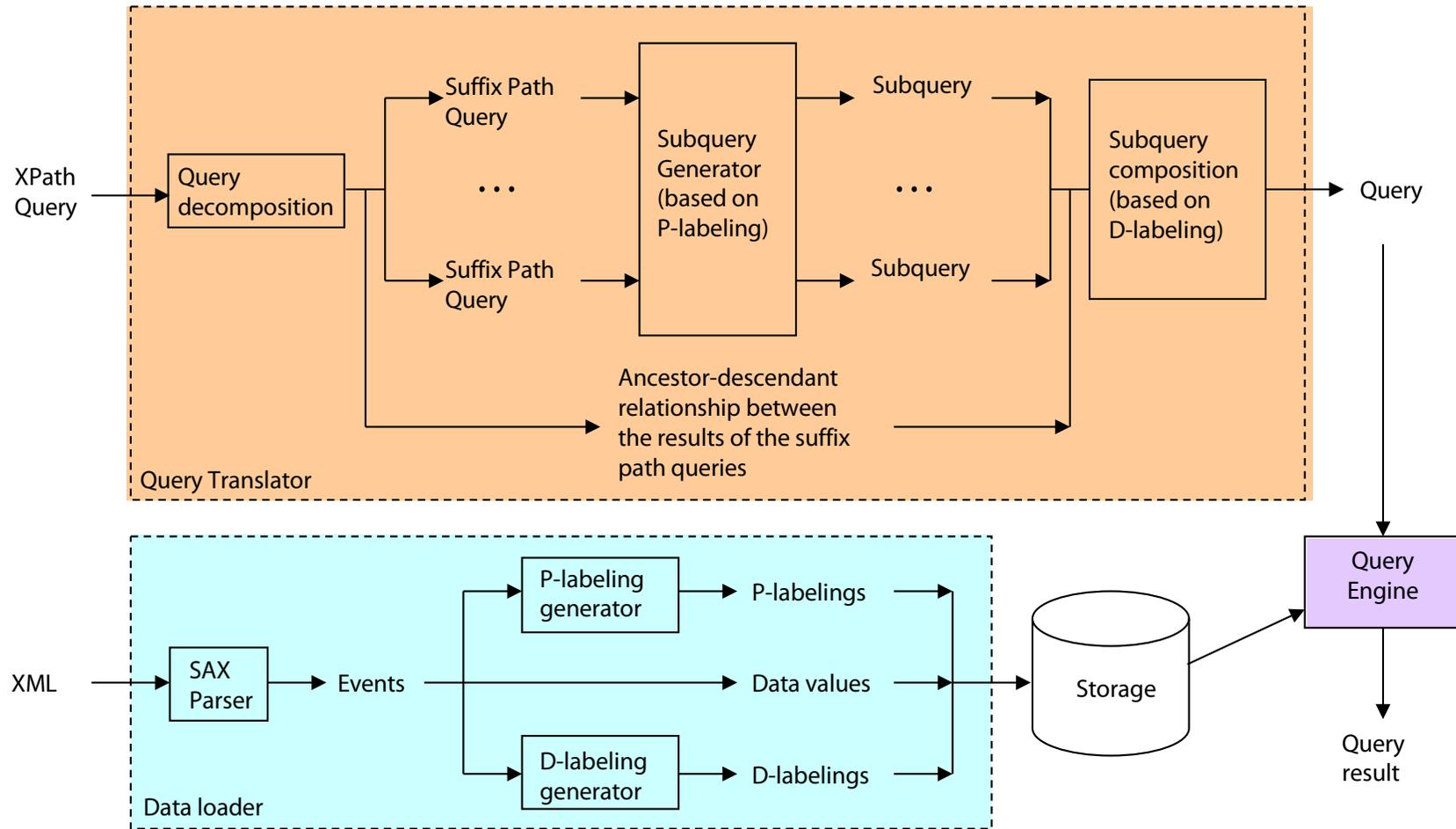


# Ein Beispiel

## Verzweigungselimination



# BLAS Architektur



# Aufspaltung von Anfragen

---

- Übersetzung einer Anfrage nach SQL
  - **Anfragedekomposition**
    - Aufspaltung einer Anfrage in eine Menge von Suffix-Pfad-Anfragen unter Speicherung von Vorgänger-Nachfolger-Beziehungen
  - **SQL-Generierung**
    - Bestimmung der P-Beschriftung der Anfrage zur Verwendung in einer SQL-Unteranfrage (Bereichsanfrage)
  - **SQL-Komposition**
    - Komposition der Teilanfragen auf Basis der D-Beschriftungen und der Vorgänger-Nachfolger-Beziehung

# Vertiefung in der Übung

---



# Weitere Arbeiten: Anfrageoptimierung

---

- Automatische Selektion von Indizierungstechniken

Beda Christoph Hammerschmidt, KeyX: Selective Key-Oriented Indexing in Native XML-Databases, Dissertation, **IFIS Universität Lübeck**, Akademische Verlagsgesellschaft Aka GmbH, DISDBIS 93, **2005**

Christian Mathis, Storing, Indexing, and Querying XML Documents in Native XML Database Management Systems, Dissertation, Universität Kaiserslautern, **2009**

- Erfüllbarkeits- und Enthaltenseins-Tests unter Berücksichtigung von XML-Schema-Spezifikationen

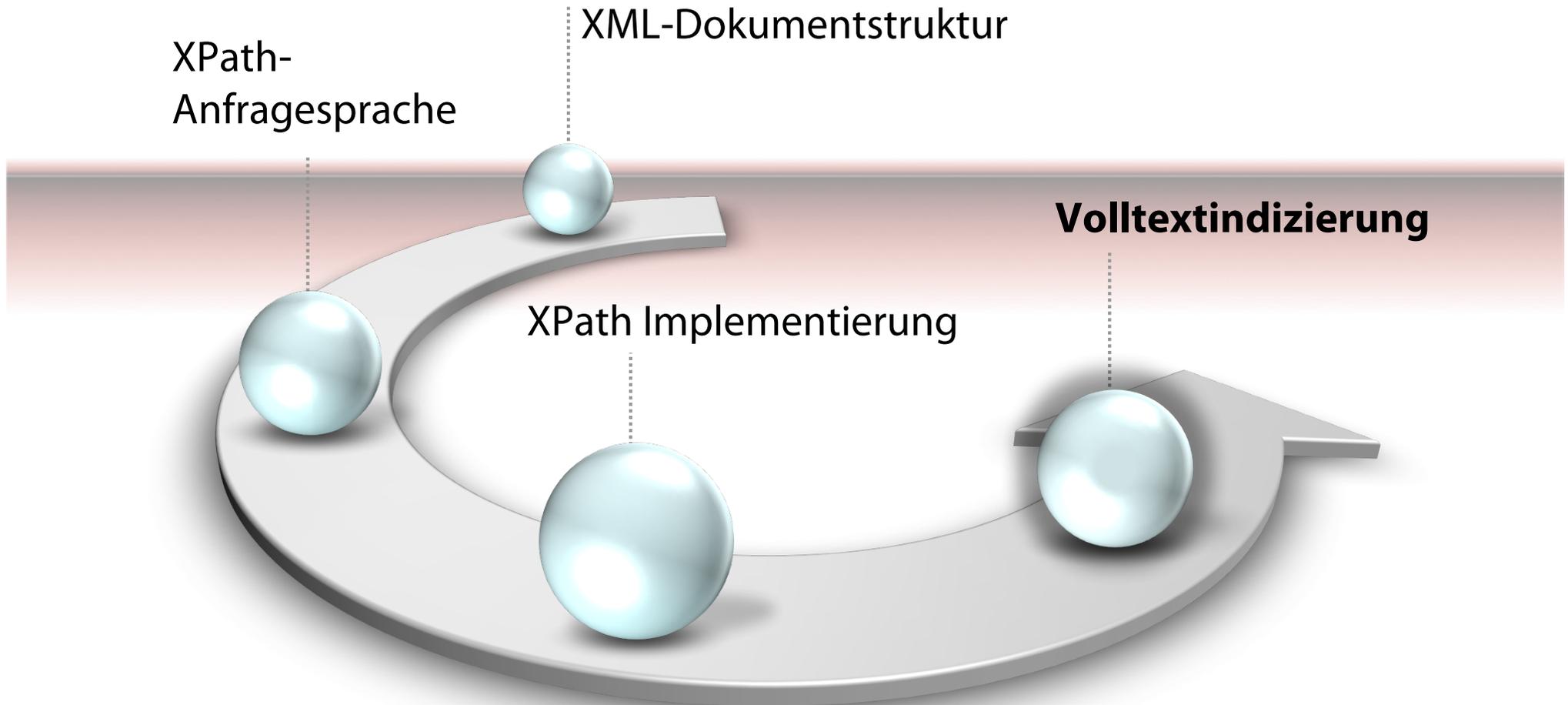
Jinghua Groppe, Speeding up XML Querying: Satisfiability Test & Containment Test of XPath Queries in the Presence of XML Schema Definitions, Dissertation, **IFIS Universität Lübeck**, dissertation.de: Verlag im Internet GmbH, **2008**

- Neue Optimierungstechniken für XML-Anfragen bei Datenverteilung in großen Sensornetzwerken (Kommunikation energieaufwändig)
- Caches in der Nähe von Datenquellen zur Vermeidung von Kommunikation und Verifikation der Cache-Kohärenz bei nicht-zuverlässigen Kommunikationskanälen

Nils Höller, Efficient XML Data Management and Query Evaluation in Wireless Sensor Networks, Dissertation **IFIS Universität Lübeck**, **2010**  
(vgl. auch das IFIS/ITM-Projekt AESOP's TALE)

# Non-Standard-Datenbanken

## Semistrukturierte Datenbanken



# Text als unstrukturierte Daten

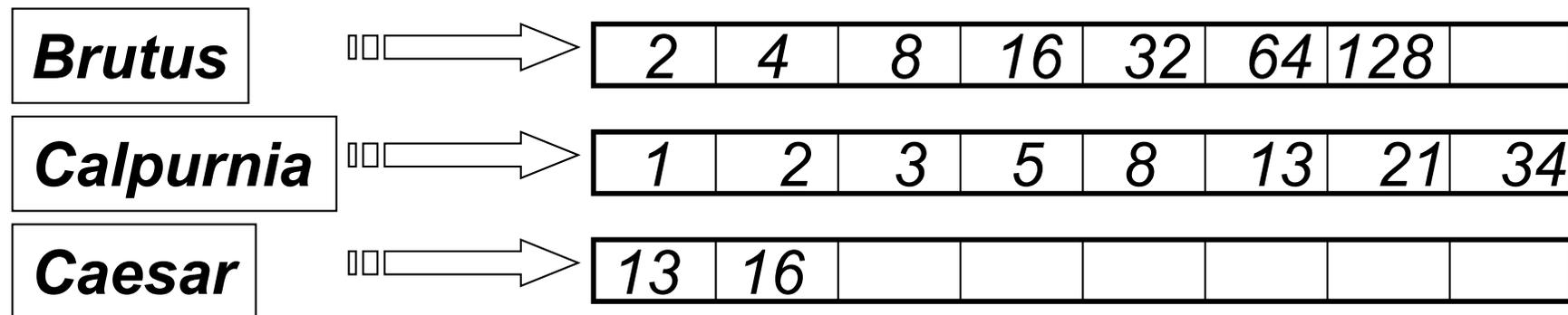
---

Welche Stücke `//drama[author='shakespeare']` enthalten die Worte **Brutus** UND **Caesar** aber NICHT **Calpurnia**?

- `//drama[author='shakespeare' and matches(content,'Brutus') & matches(content 'Caesar') and not(matches(content, 'Calpurnia'))]`
- Verwendung z.B. von KMP um alle Stücke von Shakespeare mit **Brutus** und **Caesar** zu finden, um dann solche mit **Calpurnia** zu entfernen?
  - Langsam (für große Korpora)
  - NICHT **Calpurnia** ist keinesfalls einfacher zu behandeln
  - Andere Operationen (z.B. Finde das Wort **romans** in der Nähe von **countrymen**) sind nicht einfach umsetzbar
  - Ggf. Bewertung von Ergebnissen gewünscht

# Invertierter Index

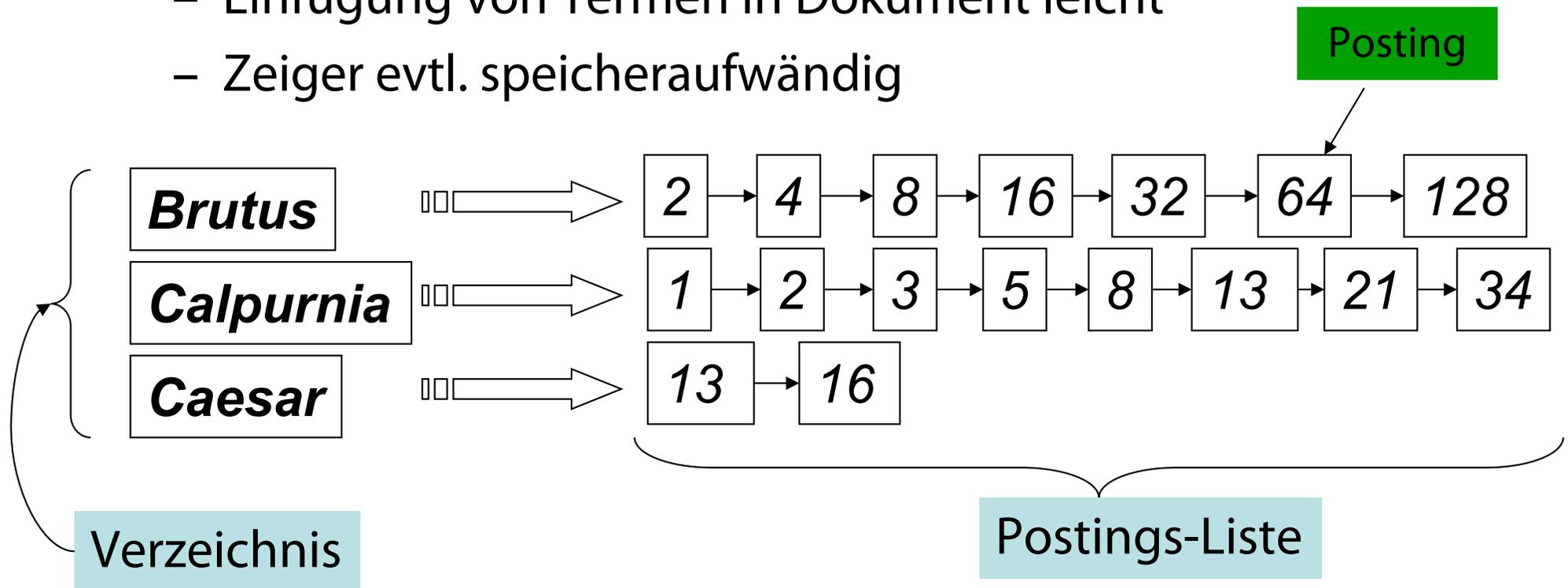
- Für jeden Term **T**, müssen wir eine Liste aller Dokumente, die **T** enthalten, speichern
- Sollen wir ein Feld oder eine Liste verwenden?



*Was machen wir, wenn das Wort **Caesar** zu Dokument 14 hinzugefügt wird?*

# Invertierter Index

- Verkettete Listen i.a. bevorzugt
  - Dynamische Speicherallokation
  - Einfügung von Termen in Dokument leicht
  - Zeiger evtl. speicheraufwändig



Sortiert nach docID (später mehr dazu)

# Antworten im Kontext

---

## Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,  
When Antony found Julius **Caesar** dead,  
He cried almost to roaring; and he wept  
When at Philippi he found **Brutus** slain.

Lineare Suche des Teilausschnitts  
könnte schon zu langsam sein  
→ Positionen auch abspeichern

## Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius **Caesar** I was killed i' the  
Capitol; **Brutus** killed me.

# Indexierungsschritte

- Geg.: Sequenz von Paaren (Token, DocID).

Doc 1

I did enact Julius  
Caesar I was killed  
i' the Capitol;  
Brutus killed me.

Doc 2

So let it be with  
Caesar. The noble  
Brutus hath told you  
Caesar was ambitious



Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

# Indizierungsschritte

## Sortierung nach Term

Hoher Aufwand

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

# Indizierungsschritte

- Mehrfacheinträge für Terme aus einem Dokument werden verschmolzen
- Anzahlinformation wird hinzugefügt

Warum Anzahl?  
Bewertung von Antworten.  
Wird später behandelt.

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



Term	Doc #	Term freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

# Umsetzung in SQL?

• SQL:

## TermDoc

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2



Term	Doc #	Term freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1

# Das Ergebnis wird in eine Verzeichnis- und eine Postings-Tabelle unterteilt

Term	Doc #	Freq
ambitious	2	1
be	2	1
brutus	1	1
brutus	2	1
capitol	1	1
caesar	1	1
caesar	2	2
did	1	1
enact	1	1
hath	2	1
I	1	2
i'	1	1
it	2	1
julius	1	1
killed	1	2
let	2	1
me	1	1
noble	2	1
so	2	1
the	1	1
the	2	1
told	2	1
you	2	1
was	1	1
was	2	1
with	2	1



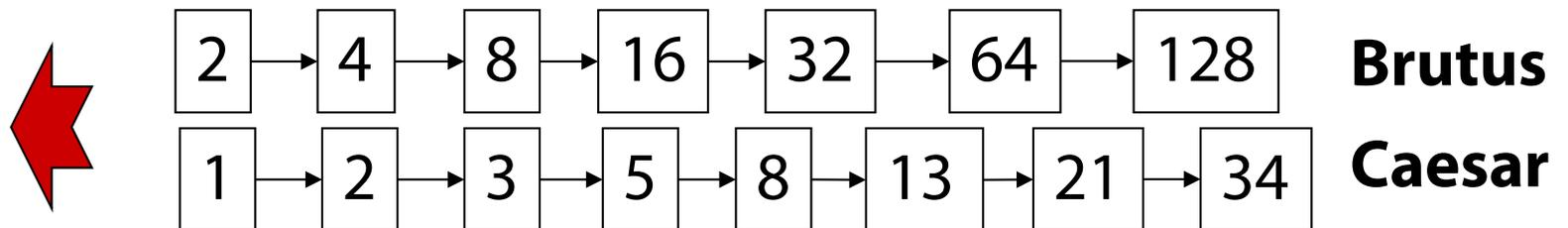
Term	N docs	Coll freq	Zeiger
ambitious	1	1	→
be	1	1	→
brutus	2	2	→
capitol	1	1	→
caesar	2	3	→
did	1	1	→
enact	1	1	→
hath	1	1	→
I	1	2	→
i'	1	1	→
it	1	1	→
julius	1	1	→
killed	1	2	→
let	1	1	→
me	1	1	→
noble	1	1	→
so	1	1	→
the	2	2	→
told	1	1	→
you	1	1	→
was	2	2	→
with	1	1	→

Doc #	Freq
2	1
2	1
1	1
2	1
1	1
1	1
2	2
1	1
1	1
2	1
1	2
1	1
2	1
1	1
1	2
2	1
1	1
2	1
2	1
1	1
2	1
2	1
1	1
2	1
2	1

Warum N docs und Coll freq?  
Bewertung von Antworten.  
Wird später behandelt.

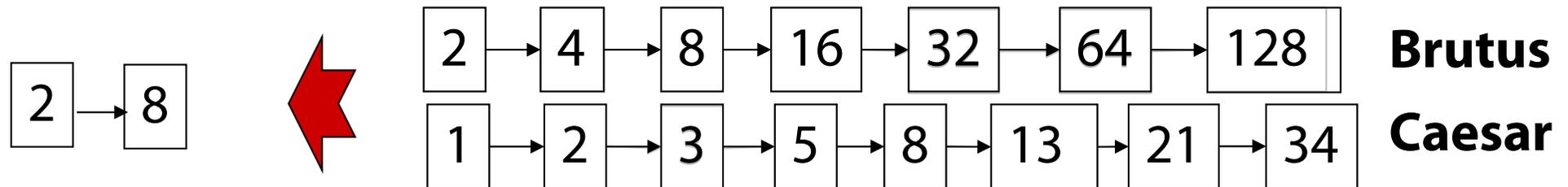
# Anfrageverarbeitung: UND

- Betrachten wir folgende Anfrage:  
**Brutus** UND **Caesar**
  - Suche **Brutus** im Verzeichnis
    - Hole die zugeordnete Postings-Liste
  - Suche **Caesar** im Verzeichnis
    - Hole die Postings-Liste
  - “Verschmelze” die Listen



# Die Verschmelzung

- Gehe synchron durch die Postings-Listen



Falls die Listen die Länge  $x$  und  $y$  haben, dauert die Verschmelzung  $O(x+y)$  Schritte.

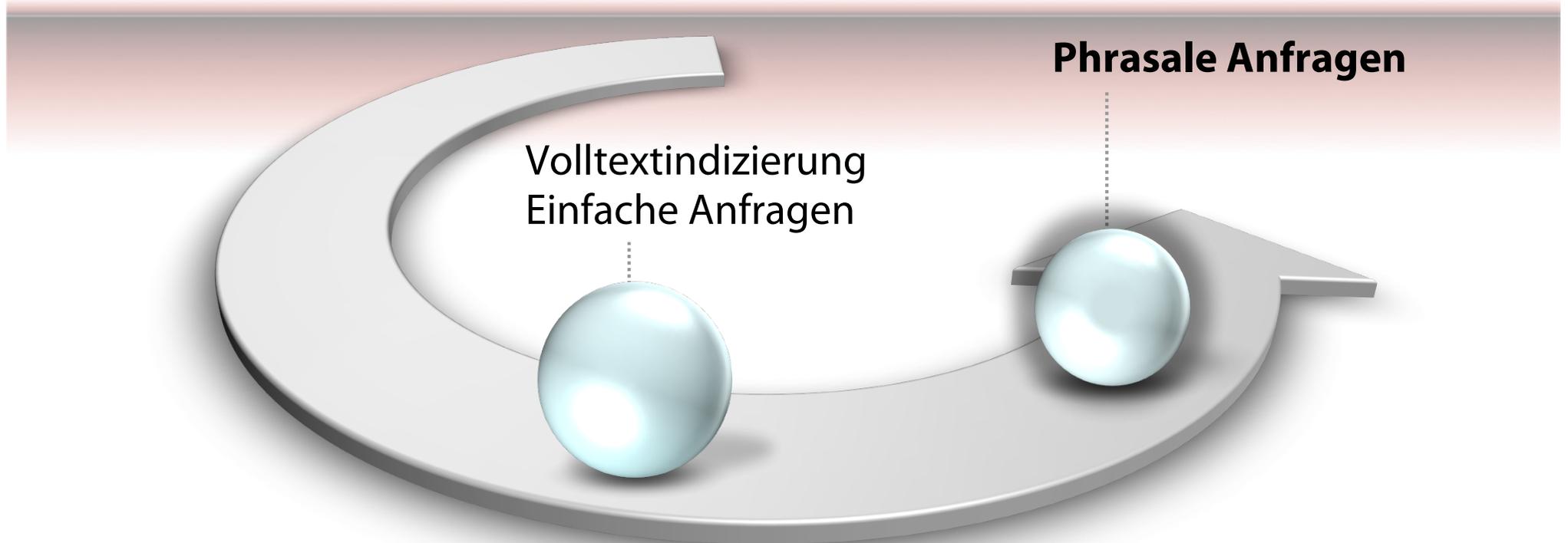
Notwendig: Postings nach docID sortiert

Knuth, D. E. Kap. 6.5. "Retrieval on Secondary Keys". The Art of Computer Programming, Reading, Massachusetts: Addison-Wesley, **1973**

Andere Methode: Signature Files z.B. mit Bloom Filter (hier nicht vertieft)  
Bloom, Burton H., Space/Time Trade-offs in Hash Coding with Allowable Errors, Communications of the ACM 13 (7), 422–426, **1970**

# Non-Standard-Datenbanken

Von semistrukturierten Datenbanken zur Volltextsuche



# Phrasale Anfragen

---

- Gesucht sind Antworten auf **“stanford university”** – als eine Phrase
- Der Satz “I went to university at Stanford” stellt keinen Treffer dar
  - Das Konzept der Phrasenanfrage wird von Nutzern gut verstanden und einigermaßen häufig verwendet (10% der Anfragen sind phrasal)
- Es reicht nicht, nur <Term : docID>-Einträge zu speichern

# Erster Versuch: Zweiwort-Indexe

---

- Indexiere jedes aufeinanderfolgende Paar von Termen im Text als Phrase
- Beispieltext: "Friends, Romans, Countrymen"
- Zweiworte:
  - **friends romans**
  - **romans countrymen**
- Jedes dieser Zweiworte wird nun ein Eintrag im Verzeichnis
- Zweiwort Phrasenanfragen können nun einfach behandelt werden

# Längere Phrasenanfragen

---

- Beispiel:  
**stanford university palo alto**
- Behandlung als boolesche Kombination von  
Zweiwortanfragen:

**stanford university** UND **university palo**  
UND **palo alto**



Falsch-positive Lösungen möglich!

- Nachbetrachtung der gefundenen Dokumente  
notwendig

# Positionsbezogene Indexe

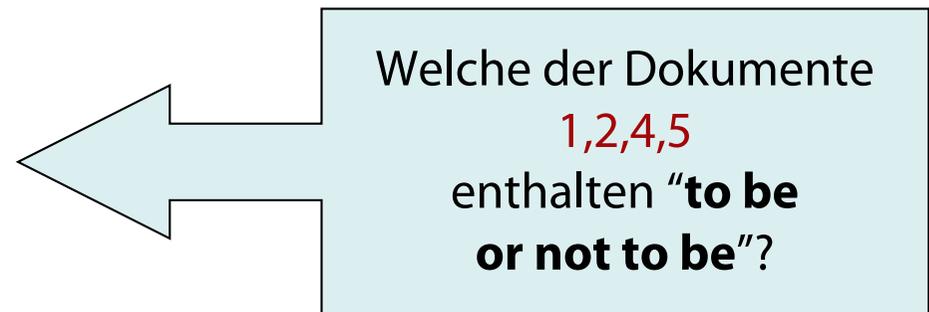
---

- Speichere für jeden Term folgende Einträge: <Anzahl der Dokumente, die Term enthalten;  
Doc<sub>1</sub>: Position<sub>1</sub>, Position<sub>2</sub> ... ;  
Doc<sub>2</sub>: Position<sub>1</sub>, Position<sub>2</sub> ... ;  
...>
- Position<sub>i</sub> kann Offset sein oder Wortnummer

# Beispiel: Positionsbezogener Index

---

<**be**: 993427;  
1: 7, 18, 33, 72, 86, 231;  
2: 3, 149;  
4: 17, 191, 291, 430, 434;  
5: 363, 367, ...>



- Datenkomprimierung möglich
- Allerdings erhöht sich der Speicherverbrauch substantiell

# Verarbeitung einer Phrasenanfrage

---

- Extrahiere die Einträge im invertierten Index für **to, be, or, not**.
- Verschmelze die Dok-Positions-Listen um alle Positionen zu finden mit **“to be or not to be”**.
  - **to:**
    - 2:1,17,74,222,551; 4:8,16,190,429,433; 7:13,23,191; ...
  - **be:**
    - 1:17,19; 4:17,191,291,430,434; 5:14,19,101; ...
- Auch anwendbar für Suchen mit “In der Nähe von”-Operatoren

Denkaufgabe:

Finde alle Dokumente, die ein Wort enthalten, das mit "mon" beginnt.

Wie können wir das realisieren?

# Platzhalter-Anfragen: \*

---

- **mon\***: Finde alle Dokumente, die ein Wort enthalten, das mit "mon" beginnt
- Verwendung eines B-Baums mit Eintragungen in lexikographischer Ordnung
- Finde alle Worte **w**, so dass **mon**  $\leq$  **w**  $<$  **moo**

Denkaufgabe:

Finde alle Dokumente, die ein Wort enthalten, das mit "mon" endet.

Wie können wir das realisieren?

# Platzhalter-Anfragen

---

- **\*mon:** Finde Worte, die mit mon enden
- Mögliche Lösung:
  - Erstelle B-Baum für Terme rückwärts geschrieben
  - Realisierung der Anfrage als Bereichsanfrage: **nom**  $\leq$  **w** < **non**.

Denkaufgabe:

Wie können wir alle Terme (und damit Dokumente) finden, die auf **pro\*cent** passen?

Was machen wir bei **se\*ate AND fil\*er**?

# B-Bäume für \*'ne am Ende der Anfrage

---

- Bei Platzhaltern in der Mitte viele Konjunkte in der resultierenden Anfrage
- Was machen wir bei multiplen Platzhaltern?
- Neuer Ansatz:  
Transformiere Anfrage, so dass Platzhalter am Ende der Anfrage auftreten  
→ "Permuterm"-Index

# Permuterm-Index

---

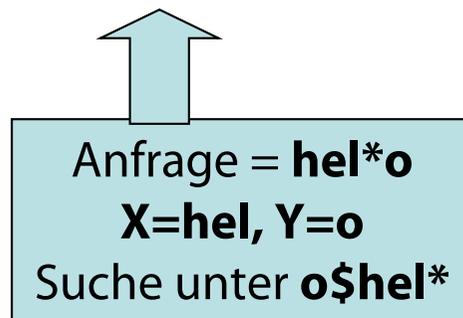
- Für Term **hello** erstelle Indexeinträge:
  - **hello\$, ello\$h, llo\$he, lo\$hel, o\$hell, \$hello**Wobei \$ ein spezielles Symbol ist

- Behandlung von Anfragen:

- **X** suche unter **X\$**
- **\*X** suche unter **X\$\***
- **X\*Y** suche unter **Y\$X\***

**X\*** suche unter **\$X\***

**\*X\*** suche unter **X\***



Denkaufgabe:

Wie behandeln wir  $X*Y*Z$ ?

# Permuterm-Anfrageverarbeitung

---

- Rotiere Anfrageplatzhalter nach rechts
- Verwende B-Baum-Zugriff wie bekannt
- Permuterm-Problem:  $\approx$  Vervierfachung der zu speichernden Daten im "Lexikon"



Empirisches Resultat für das Englische

# Bigramm-Indexe

---

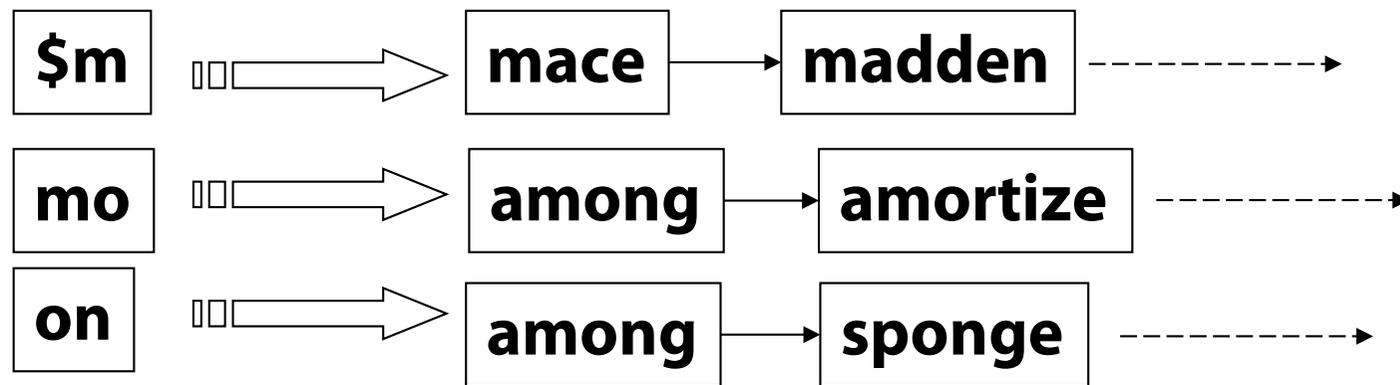
- Bestimme alle n-Gramme (Sequenz von n Zeichen), die in den Termen vorkommen
- Beispieltext "**April is the cruelest month**"  
Wir erhalten folgende 2-Gramme (Bigramme)

\$a,ap,pr,ri,il,l\$, \$i,is,s\$, \$t,th,he,e\$, \$c,cr,ru,  
ue,el,le,es,st,t\$, \$m,mo,on,nt,h\$

- \$ ist ein besonderes Abgrenzungssymbol
- Aufbau eines invertierten Index für Bigramme auf Verzeichnisterme, in denen der Index vorkommt

# Beispiel: Bigramm-Index

---



# Verarbeitung von n-Gramm-Platzhaltern

---

- Anfrage **mon**\* kann als
  - **\$m AND mo AND on** formuliert werden
- Schnell und speichereffizient
- Hole Terme und gleiche die UND-Version der Platzhalteranfrage ab
- Leider kriegen wir z.B. auch **moon** zu fassen
- Nachverarbeitung notwendig
- Überlebende Terme führen dann über den Term-Dokument-Index zum Dokument (und ggf. zur Position darin zur Hervorhebung)
  
- Suche nach ungefähr passenden Texten? Plagiate?

# BLAST: Basic Local Alignment Search Tool

- **Annahme:** Falls zwei Sequenzen eine gute lokale Übereinstimmung haben, dann ist mit großer Wahrscheinlichkeit eine kleine **Teilzeichenkette** sogar identisch (**Seed**)

TTGACTCGATTATAGTCGCGGATATACTATCG  
CCTATCACAAAGAATATAGTCCCTGATCCAGC

TTGACTC GATTATAGTCGCGGAT AACTATCG  
CCTATCAA GAATATAGTCCCTGAT CCAGC

TTGACTC GATTATAGTCGCGGAT AACTATCG  
CCTATCAA GAATATAGTCCCTGAT CCAGC

# BLAST-Algorithmus

---

Gegeben sei eine Anfragesequenz  $s$  und eine Sequenzdatenbank  $D=\{d_i\}$

1. Berechne Teilsequenzen  $s_i$  von  $s$  der Länge  $q$ 
  - Auch  $q$ -Gramme genannt (wieviele?)
2. Find all approximate occurrences of all  $s_i$  in all  $d_j$ 
  - Gap-free alignment with matrix; score must be above threshold  $t$  – Hits are called seeds – approx. occurrences of some  $s_i$  in some  $d_j$
3. Extend seeds to left and right in  $s_i$  and  $d_j$  until – [Constantly update the similarity score]
  - ... the score drops sharply
  - ... $s_i$  or  $s_j$  ends
  - ... the score gets too bad compared to other hits found earlier

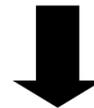
# Beispiel

$q=5, t=3$ , Matrix:  $M=+1, R=-1$   
 $s=ACGTGATA$   
 $d=GATTGACGTGACTGCTAGTGATACTATAT$



$s_1=ACGTG$   
 $s_2=CGTGA$   
 $s_3=GTGAT$   
 $s_4=TGATA$

GATTG**ACGT**GACTGCTAGTGATACTATAT  
GATTGAC**GTGA**CTGCTAGTGATACTATAT  
GATTGACGTGACTGCTAG**TGATA**CTATAT  
GATTGACGTGACT**TC**TAGTGATACTATAT



GATTGAC**GTGA**CTGCAAGTGATACTATAT  
AC**GTGATA** 5  
AC**GTGATA** 5+1=6  
AC**GTGATA** 6-1=5  
... ..

# Eigenschaften

---

- Finding seeds efficiently requires more work
  - Pre-compute all q-grams of all  $d_i$
  - Group by q-gram
  - Called a hash-index (should be kept in main memory)
  - Lookup: Given  $s$ , find all matching q-grams (as seeds)
- Exclusion method
  - Vast majority of all sequences in DB are never looked at because they do not contain a seed
  - The “seed” idea is the basis of nearly all fast alignment methods
- Where it fails
  - Sensitive to  $t$ : Too high – missing hits; too low – slow
  - Does not consider gaps

# Zusammenfassung

---

