



**Hasso  
Plattner  
Institut**

IT Systems Engineering | Universität Potsdam

Datenbanksysteme I  
Anfragebearbeitung und -optimierung

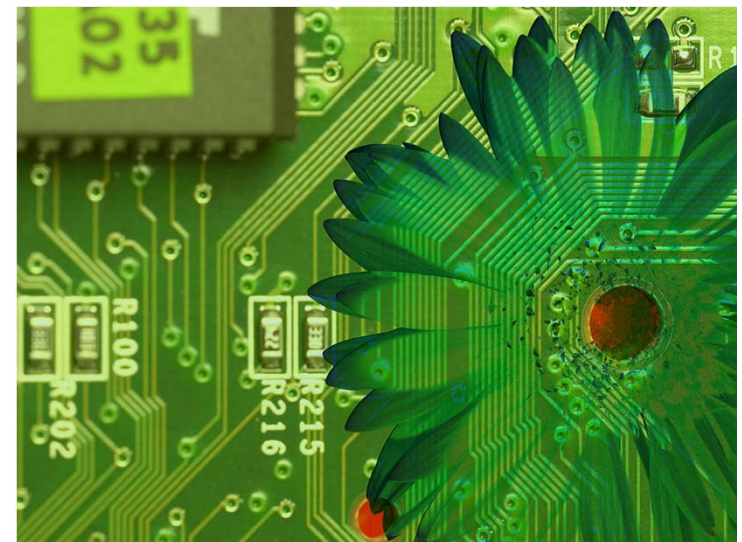
27.6.2011

Felix Naumann

# Anfragebearbeitung – Grundproblem

2

- Anfragen sind deklarativ.
  - SQL, Relationale Algebra
- Anfragen müssen in ausführbare (prozedurale) Form transformiert werden.
- Ziele
  - „QEP“ – prozeduraler Query Execution Plan
  - Effizienz
    - ◇ Schnell
    - ◇ Wenig Ressourcenverbrauch (CPU, I/O, RAM, Bandbreite)
    - ◇ Energie



# Ablauf der Anfragebearbeitung

3

## 1. Parsing

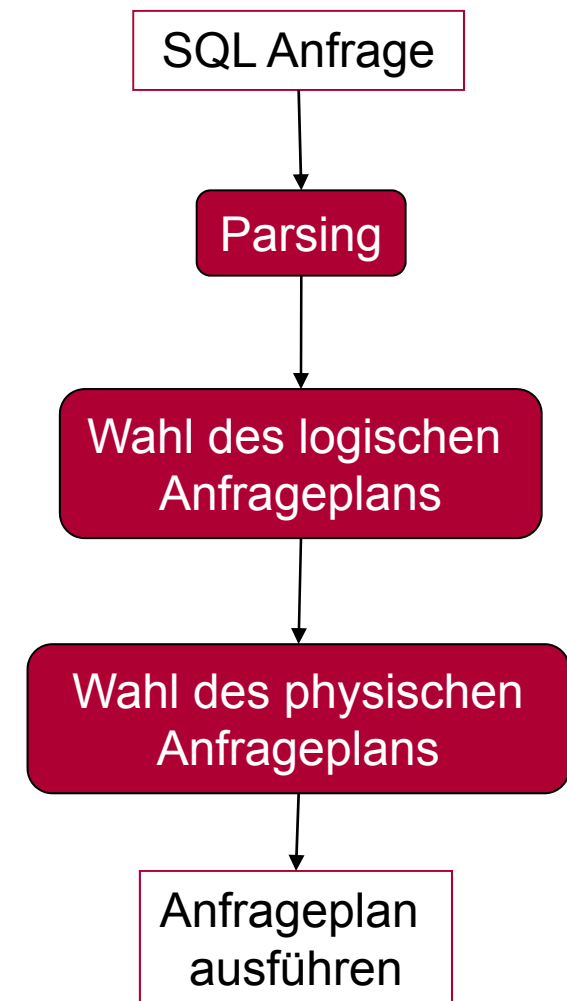
- Parsen der Anfrage (Syntax)
- Überprüfen der Elemente („Semantik“)
- Parsebaum

## 2. Wahl des logischen Anfrageplans

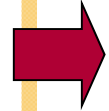
- Baum mit logischen Operatoren
- Potentiell exponentiell viele
- Wahl des optimalen Plans
  - ◇ Logische Optimierung
  - ◇ Regelbasierter Optimierer
  - ◇ Kostenbasierter Optimierer

## 3. Wahl des physischen Anfrageplans

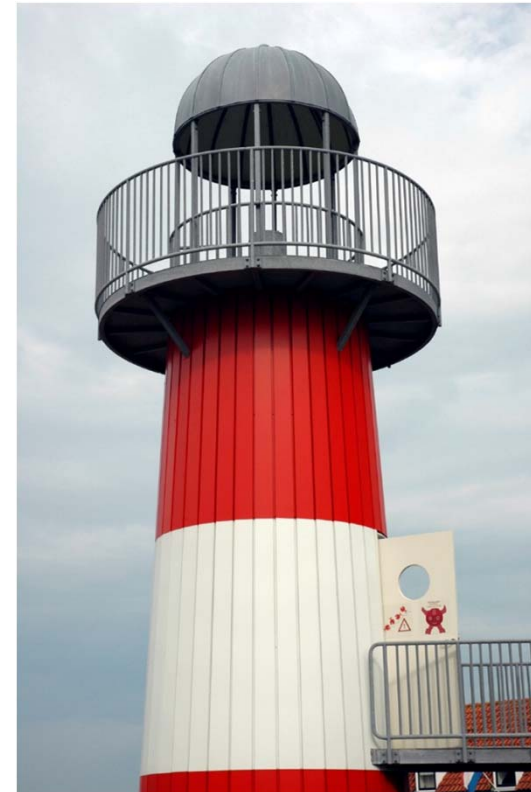
- Ausführbar
- Programm mit physischen Operatoren
  - ◇ Algorithmen
  - ◇ Scan Operatoren
- Wahl des optimalen Plans
  - ◇ physische Optimierung



4



- Parsen der Anfrage
- Transformationsregeln der RA
- Optimierung
- Kostenmodelle



*Hinweis: Ausflug ins Kapitel 16 des Lehrbuchs*

# Syntaxanalyse

5

Aufgabe: Umwandlung einer SQL Anfrage in einen Parsebaum.

- Atome (Blätter)
  - Schlüsselworte
  - Konstanten
  - Namen (Relationen und Attribute)
  - Syntaxzeichen
  - Operatoren
- Syntaktische Kategorien
  - Namen für Teilausdrücke einer Anfrage

# Eine Grammatik für einen Teil von SQL

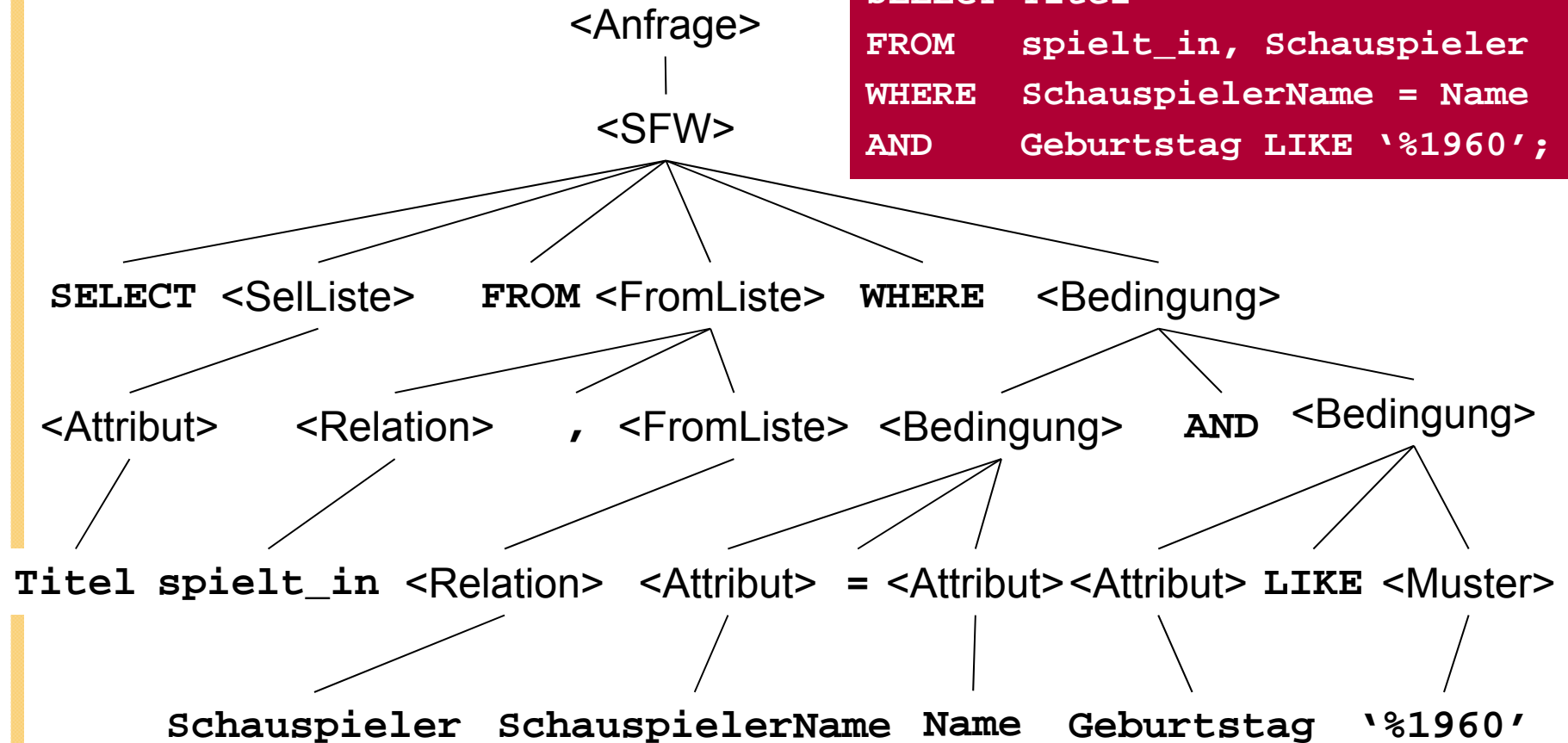
6

- Anfragen
  - $\langle \text{Anfrage} \rangle ::= \langle \text{SFW} \rangle$
  - $\langle \text{Anfrage} \rangle ::= ( \langle \text{SFW} \rangle )$
  - Mengenoperatoren fehlen
- SFWs
  - $\langle \text{SFW} \rangle ::= \text{SELECT } \langle \text{SelListe} \rangle \text{ FROM } \langle \text{FromListe} \rangle \text{ WHERE } \langle \text{Bedingung} \rangle$
  - Gruppierung, Sortierung etc. fehlen
- Listen
  - $\langle \text{SelListe} \rangle ::= \langle \text{Attribut} \rangle, \langle \text{SelListe} \rangle$
  - $\langle \text{SelListe} \rangle ::= \langle \text{Attribut} \rangle$
  - $\langle \text{FromListe} \rangle ::= \langle \text{Relation} \rangle, \langle \text{FromListe} \rangle$
  - $\langle \text{FromListe} \rangle ::= \langle \text{Relation} \rangle$
- Bedingungen (Beispiele)
  - $\langle \text{Bedingung} \rangle ::= \langle \text{Bedingung} \rangle \text{ AND } \langle \text{Bedingung} \rangle$
  - $\langle \text{Bedingung} \rangle ::= \langle \text{Tupel} \rangle \text{ IN } \langle \text{Anfrage} \rangle$
  - $\langle \text{Bedingung} \rangle ::= \langle \text{Attribut} \rangle = \langle \text{Attribut} \rangle$
  - $\langle \text{Bedingung} \rangle ::= \langle \text{Attribut} \rangle \text{ LIKE } \langle \text{Muster} \rangle$
- $\langle \text{Tupel} \rangle, \langle \text{Attribut} \rangle, \langle \text{Relation} \rangle, \langle \text{Muster} \rangle$  nicht durch grammatische Regel definiert
- Vollständig z.B. hier: <http://docs.openlinksw.com/virtuoso/GRAMMAR.html>

# Parse-Baum

7

```
SELECT Titel
FROM spielt_in, Schauspieler
WHERE SchauspielerName = Name
AND Geburtstag LIKE '%1960';
```



# Prüfung der Semantik

8

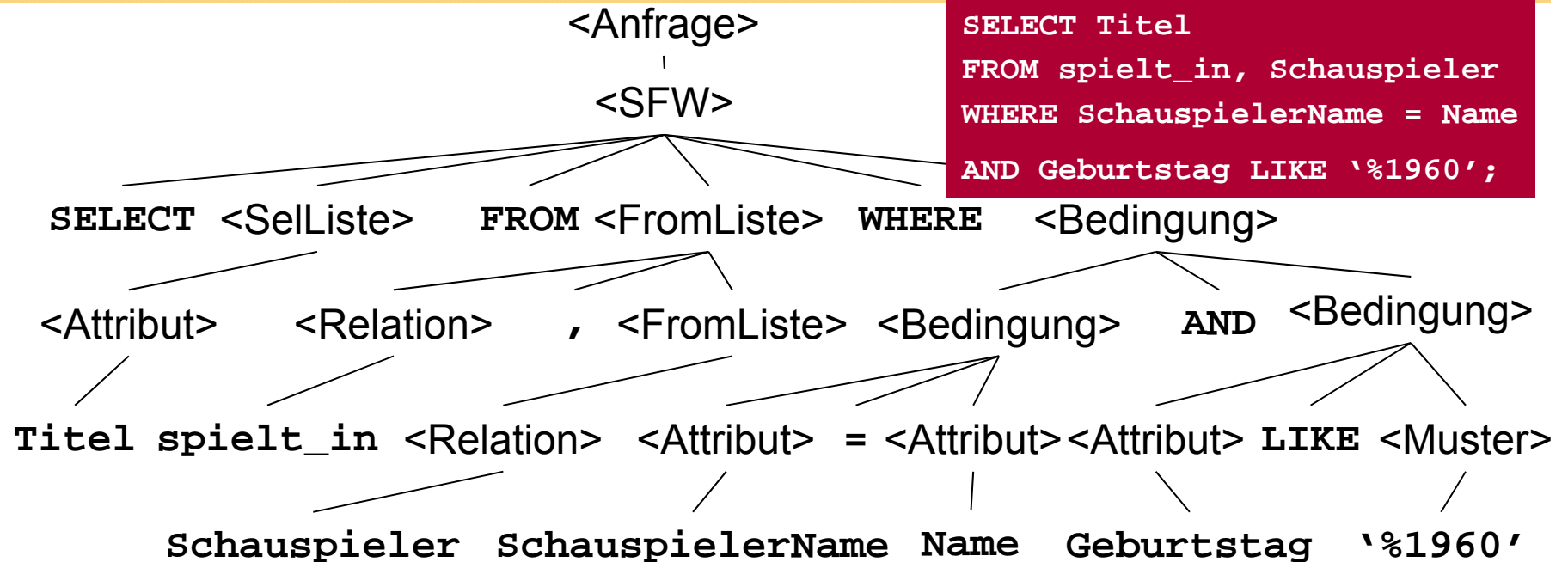
Während der Übersetzung semantische Korrektheit prüfen

- Existieren die Relationen und Sichten der FROM Klausel?
- Existieren die Attribute in den genannten Relationen?
  - Sind sie eindeutig?
- Korrekte Typen für Vergleiche?
- Aggregation korrekt?
- ...



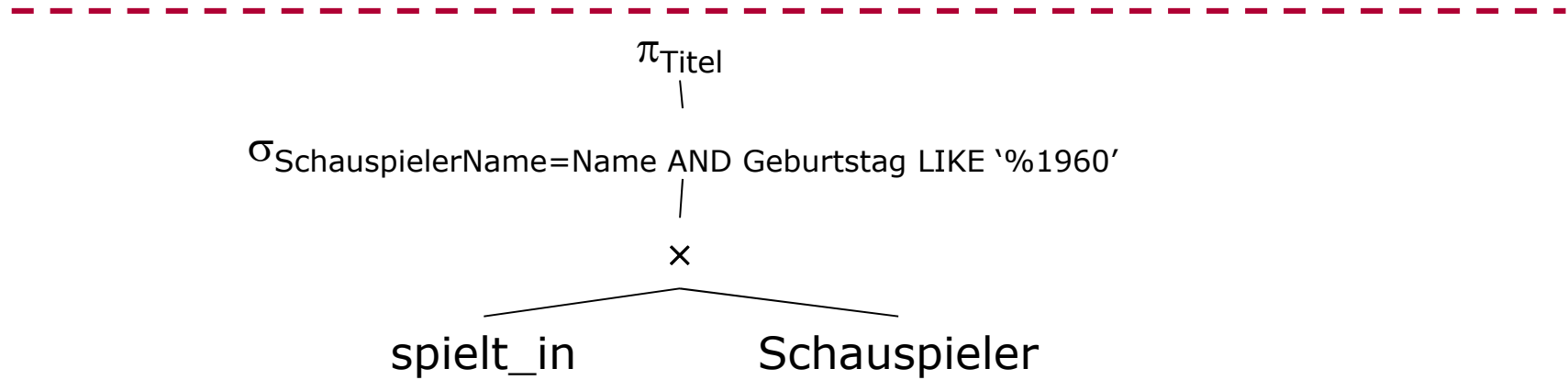
# Vom Parse-Baum zum Operatorbaum

9



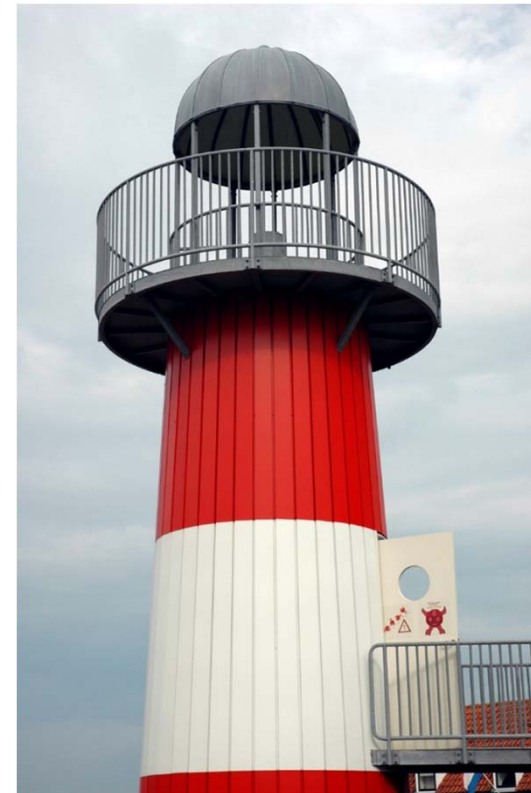
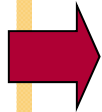
```

SELECT Titel
FROM spielt_in, Schauspieler
WHERE SchauspielName = Name
AND Geburtstag LIKE '%1960';
  
```

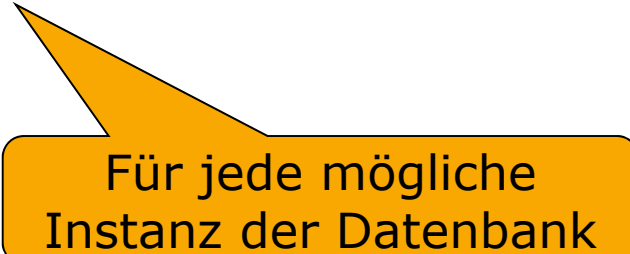


10

- Parsen der Anfrage
- Transformationsregeln der Relationalen Algebra
- Optimierung
- Kostenmodelle



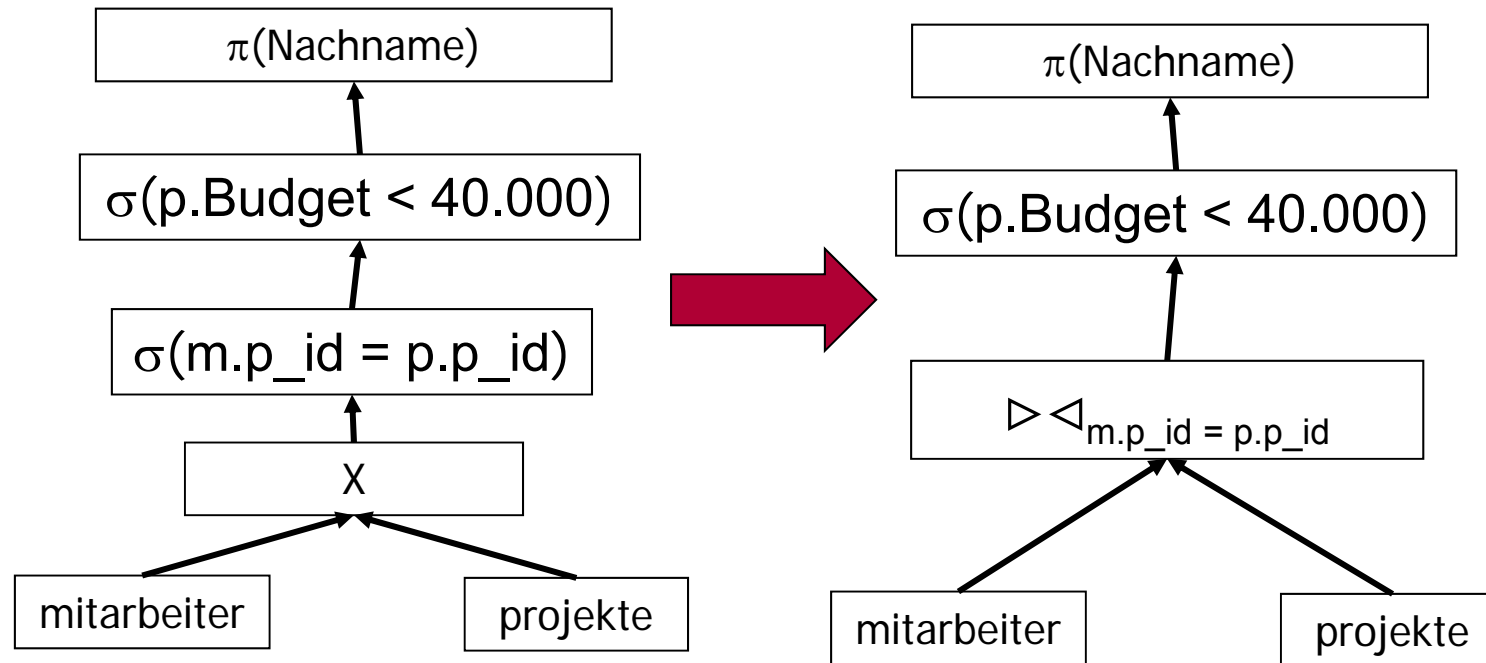
- Transformation der internen Darstellung
  - Ohne Semantik zu verändern
  - Zur effizienteren Ausführung
    - ◇ Insbesondere: Kleine Zwischenergebnisse
- Äquivalente Ausdrücke
  - Zwei Ausdrücke der relationalen Algebra heißen äquivalent, falls
    - ◇ Gleiche Operanden (= Relationen)
    - ◇ Stets gleiche Antwortrelation
      - Stets?



Für jede mögliche  
Instanz der Datenbank

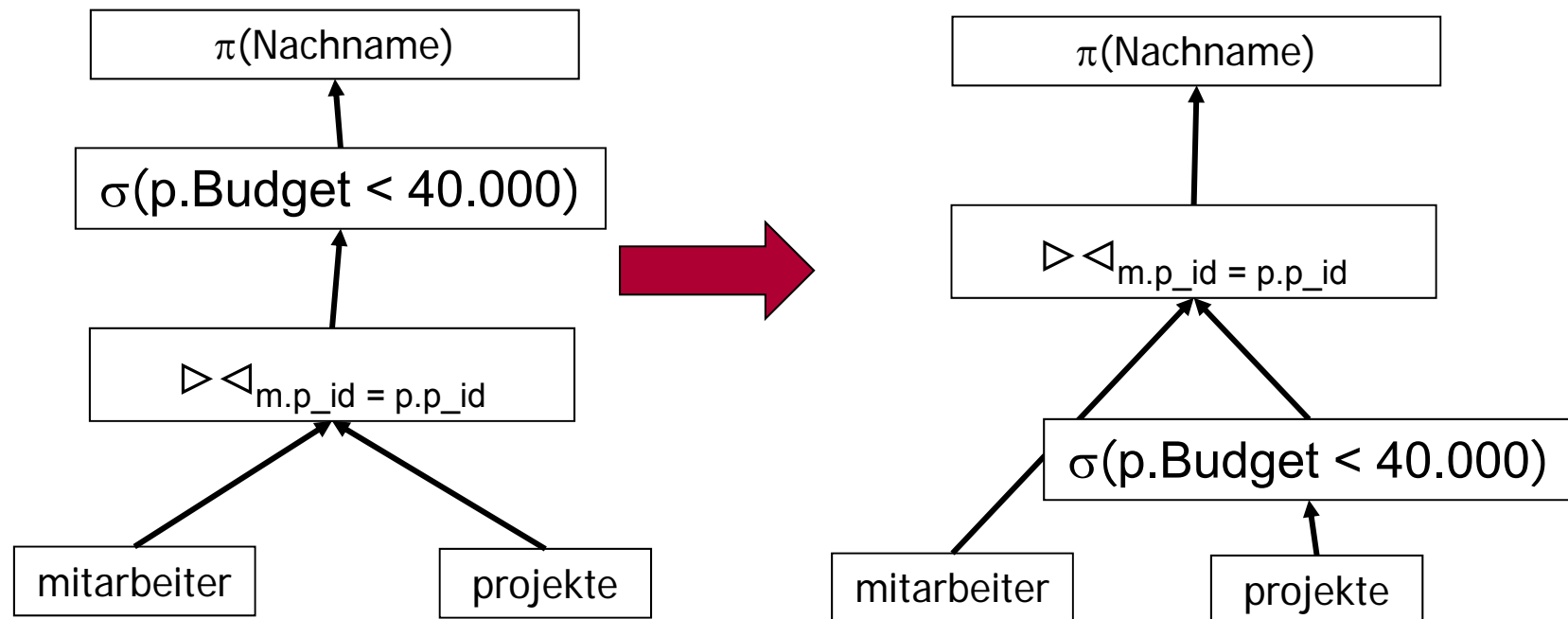
# Anfragebearbeitung – Beispiel

12



# Anfragebearbeitung – Beispiel

13



# Kommutativität und Assoziativität

14

- $\times$  ist kommutativ und assoziativ
  - $R \times S = S \times R$
  - $(R \times S) \times T = R \times (S \times T)$
- $\cup$  ist kommutativ und assoziativ
  - $R \cup S = S \cup R$
  - $(R \cup S) \cup T = R \cup (S \cup T)$
- $\cap$  ist kommutativ und assoziativ
  - $R \cap S = S \cap R$
  - $(R \cap S) \cap T = R \cap (S \cap T)$
- $\bowtie$  ist kommutativ und assoziativ
  - $R \bowtie S = S \bowtie R$
  - $(R \bowtie S) \bowtie T = R \bowtie (S \bowtie T)$

Gilt jeweils für Mengen  
und Multimengen

Ausdrücke können in beide  
Richtungen verwendet werden.

# Weitere Regeln

15

## Selektion

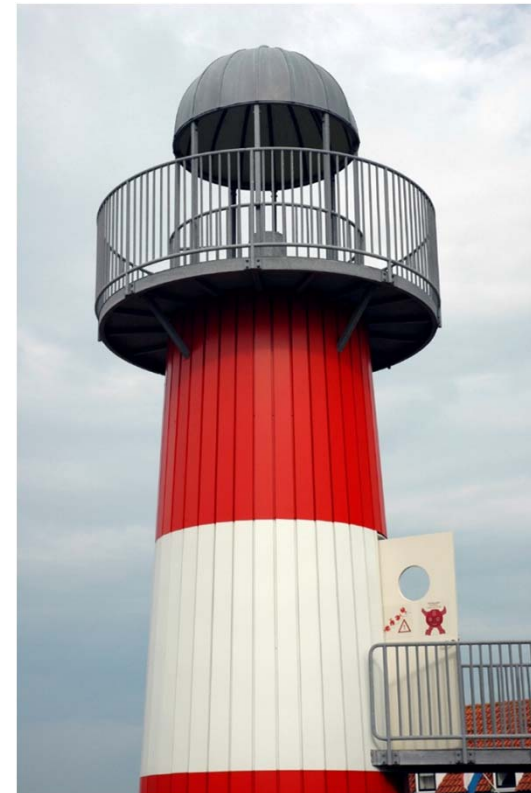
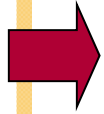
- $\sigma_{c_1 \text{ AND } c_2}(R) = \sigma_{c_1}(\sigma_{c_2}(R))$
- $\sigma_{c_1 \text{ OR } c_2}(R) = \sigma_{c_1}(R) \cup \sigma_{c_2}(R)$ 
  - Nicht bei Multimengen
- $\sigma_{c_1}(\sigma_{c_2}(R)) = \sigma_{c_2}(\sigma_{c_1}(R))$
- $\sigma_c(R \Phi S) \equiv (\sigma_c(R)) \Phi (\sigma_c(S))$ 
  - $\Phi \in \{\cup, \cap, -, \bowtie\}$
- $\sigma_c(R \Phi S) \equiv (\sigma_c(R)) \Phi S$ 
  - $\Phi \in \{\cup, \cap, -, \bowtie\}$
  - Falls sich c nur auf Attribute in R bezieht.

## Projektion

- $\pi_L(R \bowtie S) = \pi_L(\pi_M(R) \bowtie \pi_N(S))$
- $\pi_L(R \bowtie_C S) = \pi_L(\pi_M(R) \bowtie_C \pi_N(S))$
- $\pi_L(R \times S) = \pi_L(\pi_M(R) \times \pi_N(S))$
- $\pi_L(\sigma_C(R)) = \pi_L(\sigma_C(\pi_M(R)))$

16

- Parsen der Anfrage
- Transformationsregeln der RA
- Optimierung
- Kostenmodelle





# Grundsätze der Anfrageoptimierung

17

- High-level SQL (deklarativ nicht prozedural)
  - „was“, nicht „wie“.
- Das „wie“ bestimmt sich aus der Abbildung der mengenorientierten Operatoren auf die Schnittstellen-Operatoren der internen Ebene.
  - Zugriff auf Datensätze in Dateien
  - Einfügen/Entfernen interner Datensätze
  - Modifizieren interner Datensätze
- Zu einem „was“ kann es zahlreiche „wie`s“ geben.
  - Äquivalenzerhaltende Transformationen
- Im Allgemeinen wird nicht die optimale Auswertungsstrategie gesucht, sondern eine einigermaßen effiziente Variante.
  - Ziel: *Avoid the worst case.*

Folie: Prof. Alfons Kemper, TU München

# Anfragebearbeitung - Optimierung

18

## Regelbasierte Optimierung

- Fester Regelsatz schreibt Transformationen gemäß der genannten Regeln vor.
- Prioritäten unter den Regeln
  - Heuristik

## Kostenbasierte Optimierung

- Kostenmodell
- Transformationen um Kosten zu verringern
- Bestimmung des optimalen Plans
  - Bestimmung der optimalen Joinreihenfolge

# Logische und physische Optimierung

19

- Logische Optimierung
  - Jeder Ausdruck kann in viele verschiedene, semantisch äquivalente Ausdrücke umgeschrieben werden.
  - Wähle den (hoffentlich) besten Ausdruck (=Plan, =QEP)
- Physische Optimierung
  - Für jede relationale Operation gibt es viele verschiedene Implementierungen.
  - Zugriff auf Tabellen
    - ◇ Scan, verschiedene Indizes, sortierter Zugriff, ...
  - Joins
    - ◇ Nested loop, sort-merge, hash, ...
  - Wähle für jede Operation die (hoffentlich) beste Implementierung
- Abhängigkeit beider Probleme!

# Logische Optimierung

20

- Grundsätze der logischen Optimierung
  - Selektionen so weit wie möglich im Baum nach unten schieben.
  - Selektionen mit **AND** können aufgeteilt und separat verschoben werden.
  - Projektionen so weit wie möglich im Baum nach unten schieben,
    - ◇ bzw. neue Projektionen können eingefügt werden.
  - Duplikateliminierung kann manchmal entfernt werden oder verschoben werden.
  - Kreuzprodukte mit geeigneten Selektionen zu einem Join zusammenfassen.
  
- Noch nicht hier: Suche nach der optimalen Joinreihenfolge

*Folie: Prof. Alfons Kemper, TU München*

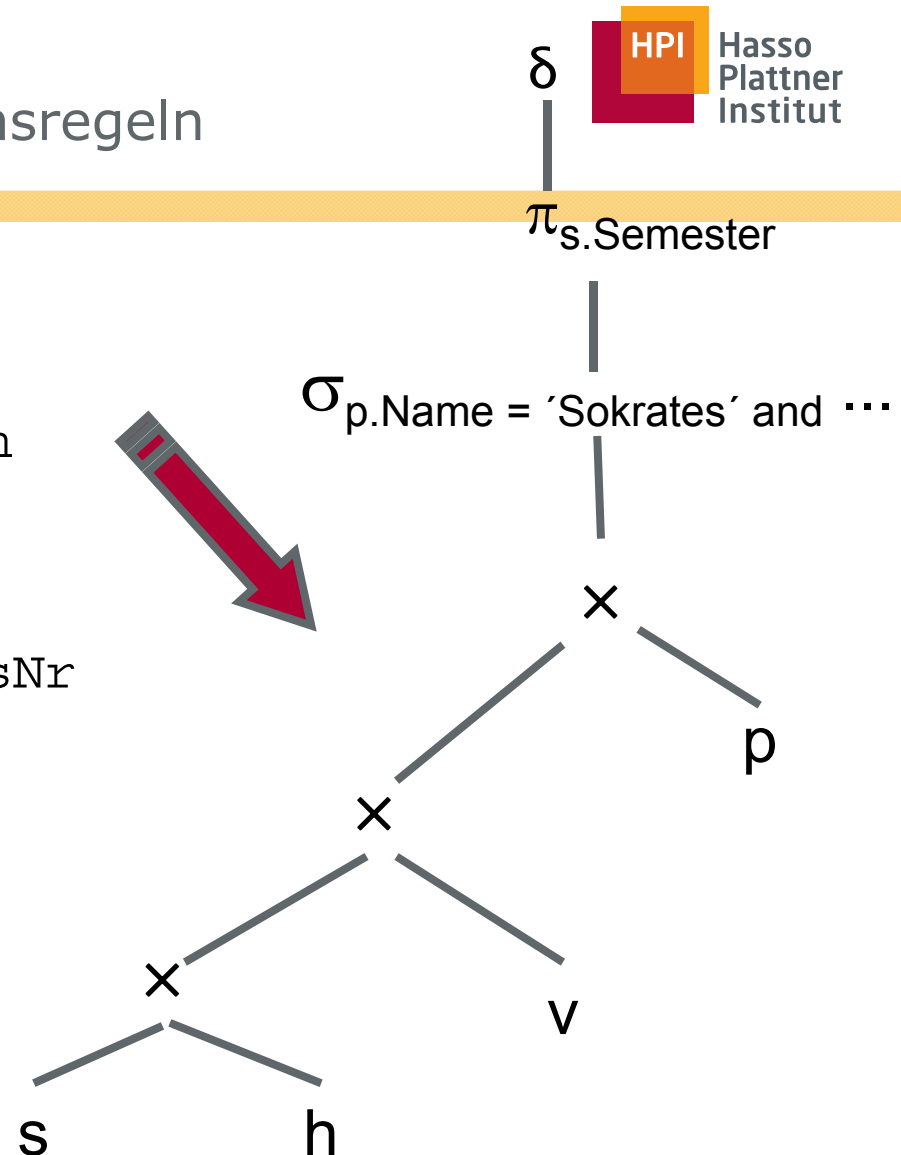
# Anwendung der Transformationsregeln

21

```

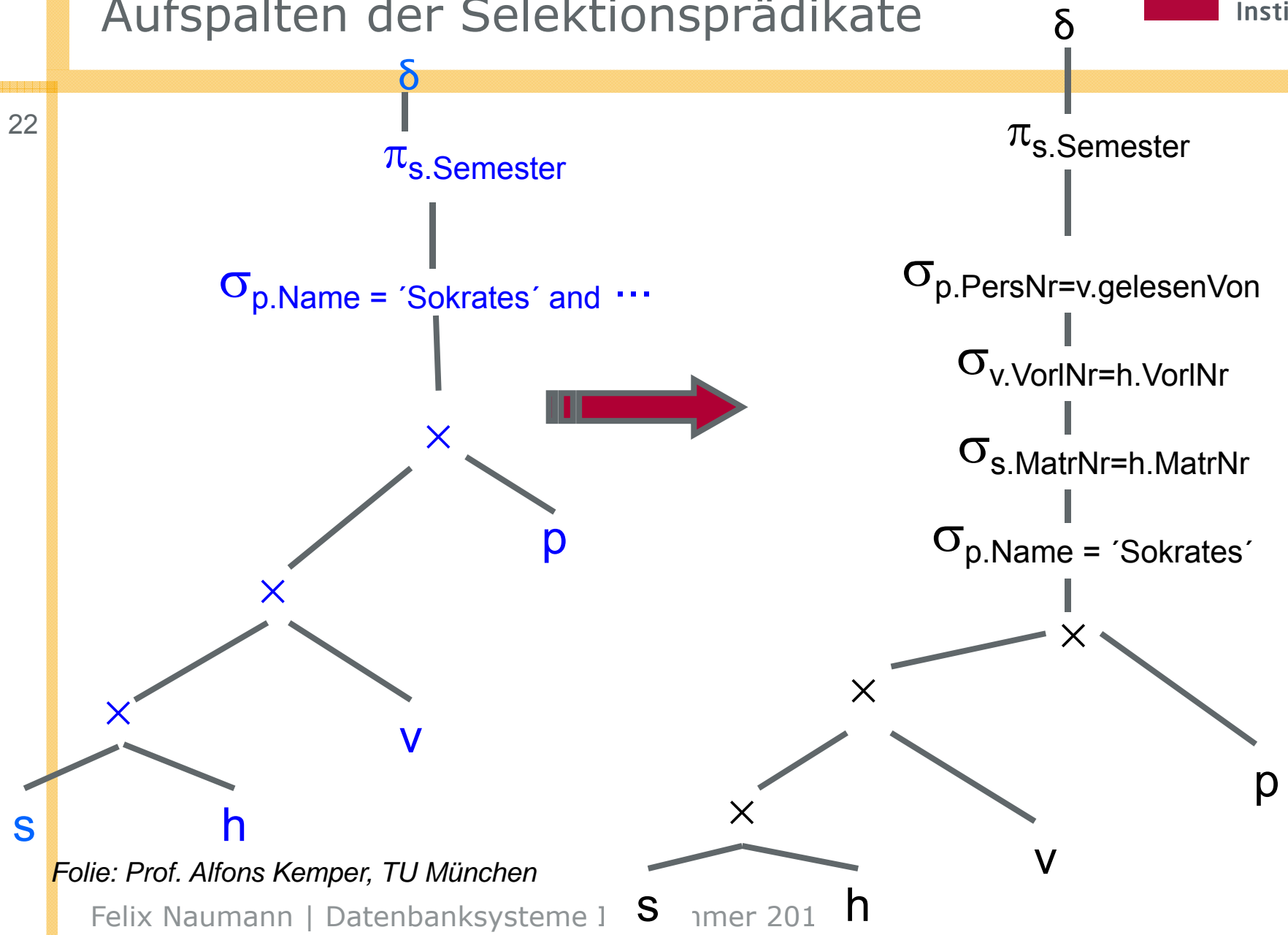
select distinct s.Semester
from Studenten s, hören h
      Vorlesungen v,
      Professoren p
where p.Name = 'Sokrates'
and v.gelesenVon = p.PersNr
and v.VorlNr = h.VorlNr
and h.MatrNr = s.MatrNr
  
```

*In welchen Semestern sind die Studenten, die VLen bei Sokrates hören?*



# Aufspalten der Selektionsprädikate

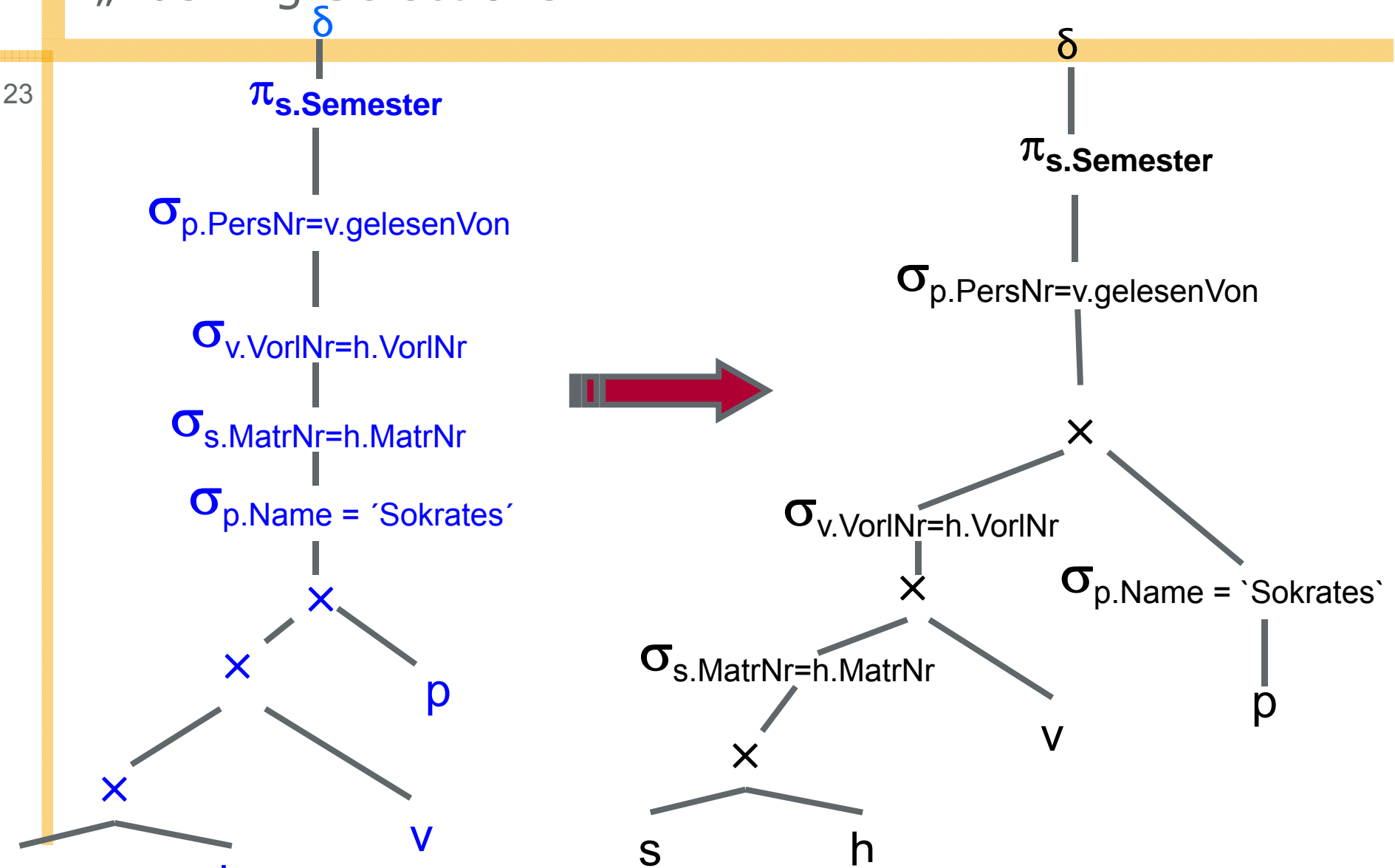
22



Folie: Prof. Alfons Kemper, TU München

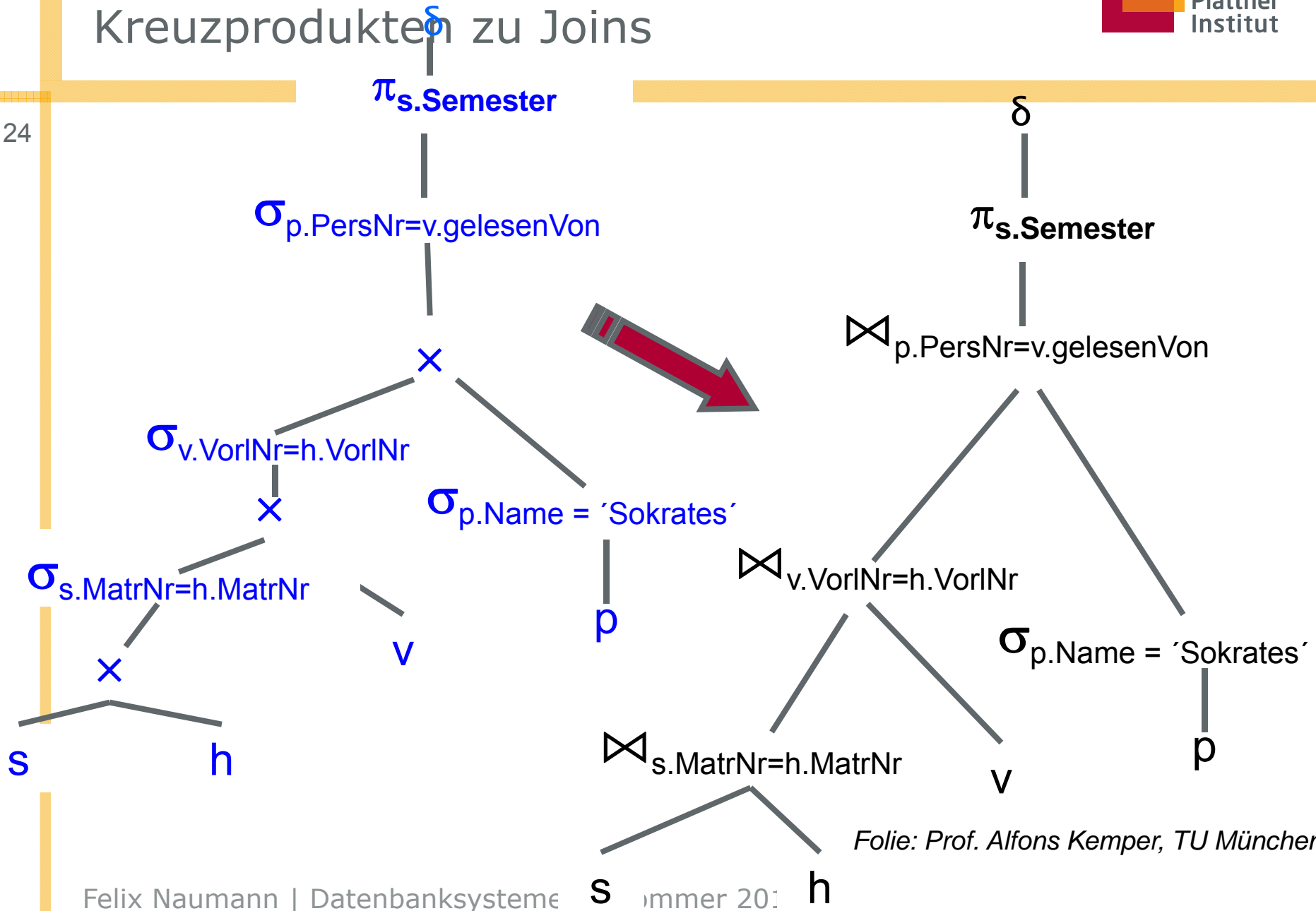
# Verschieben der Selektionsprädikate „Pushing Selections“

23



# Zusammenfassung von Selektionen und Kreuzprodukten zu Joins

24

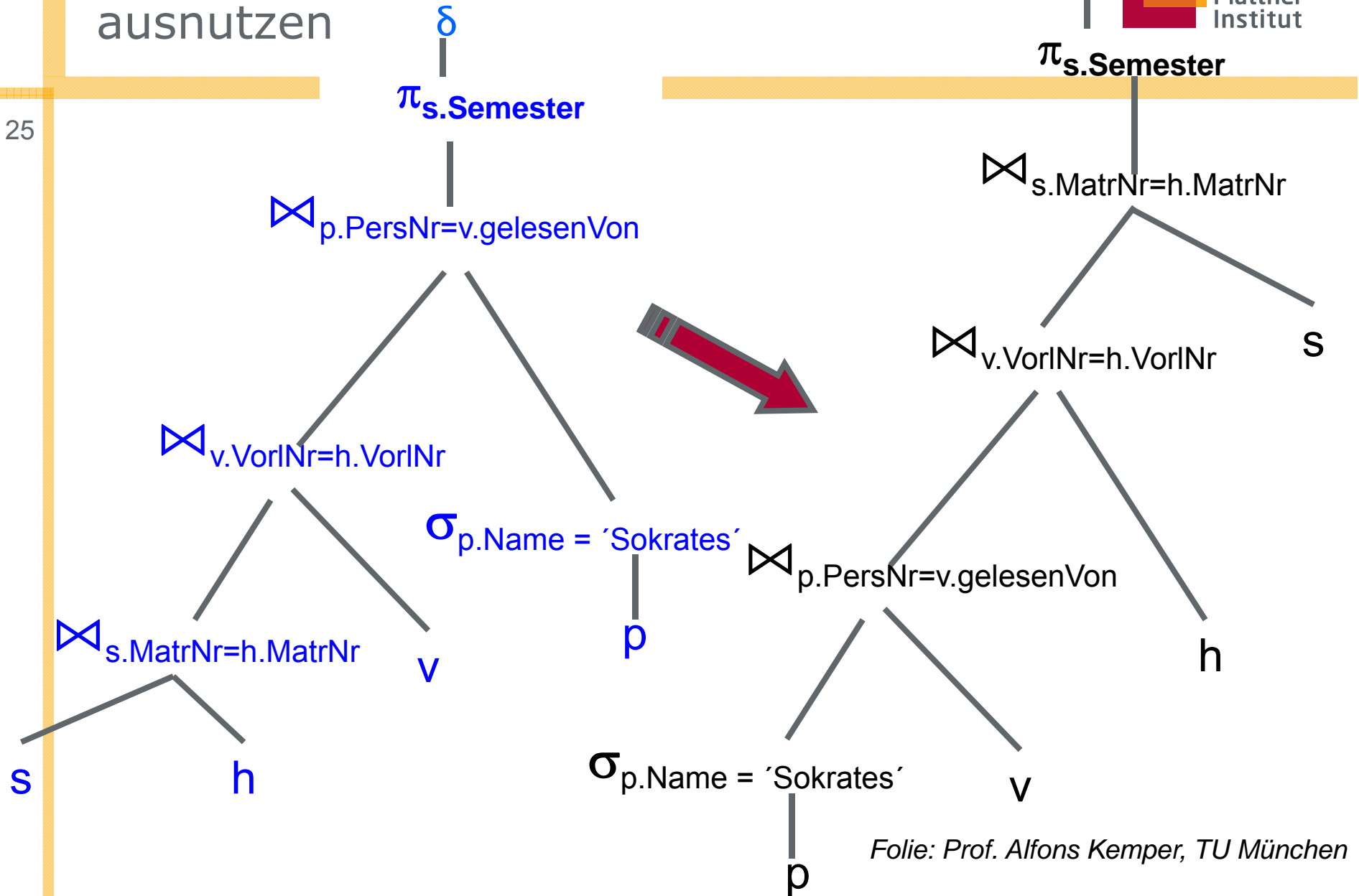


Folie: Prof. Alfons Kemper, TU München



# Optimierung der Joinreihenfolge: Kommutativität und Assoziativität ausnutzen

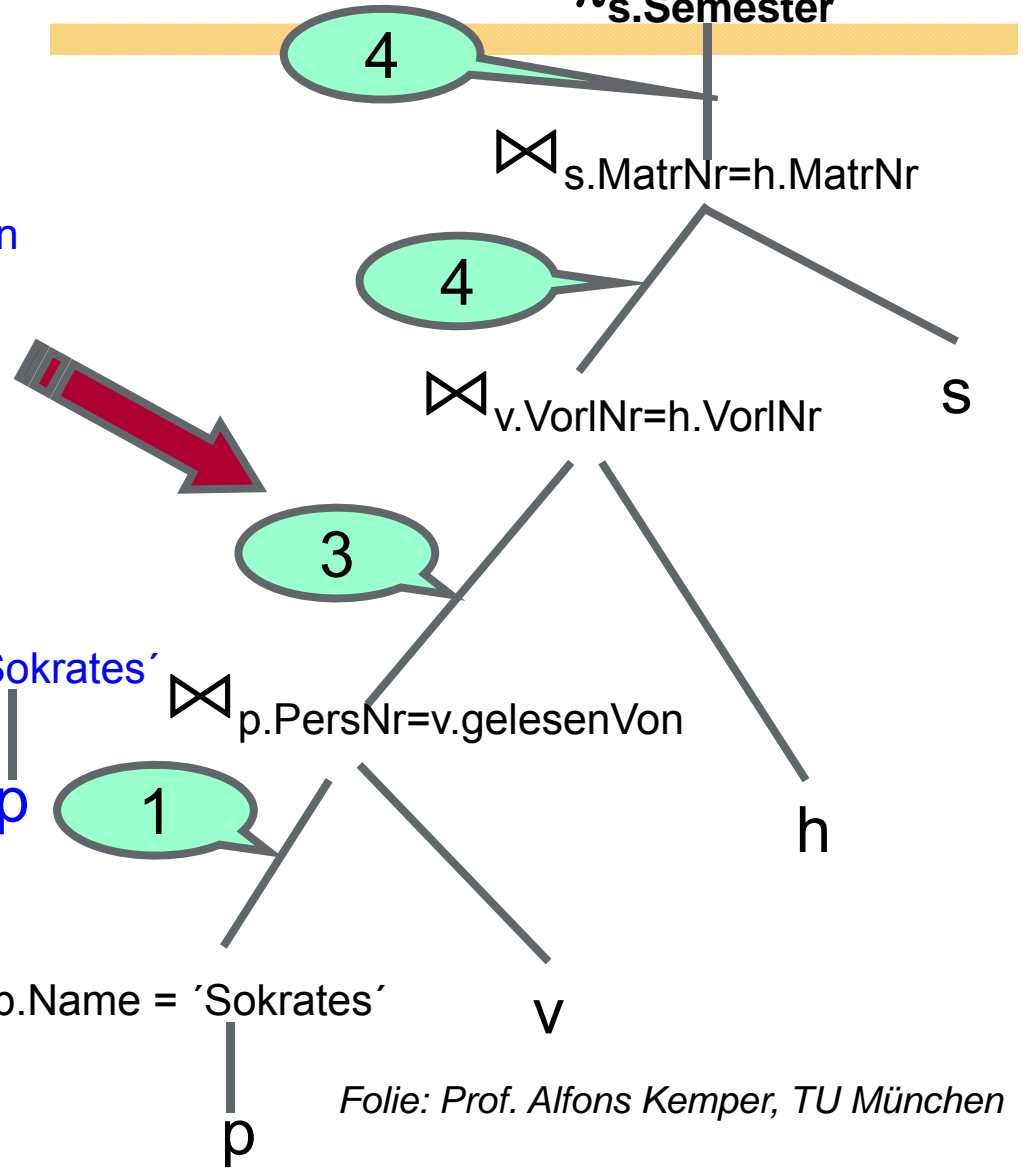
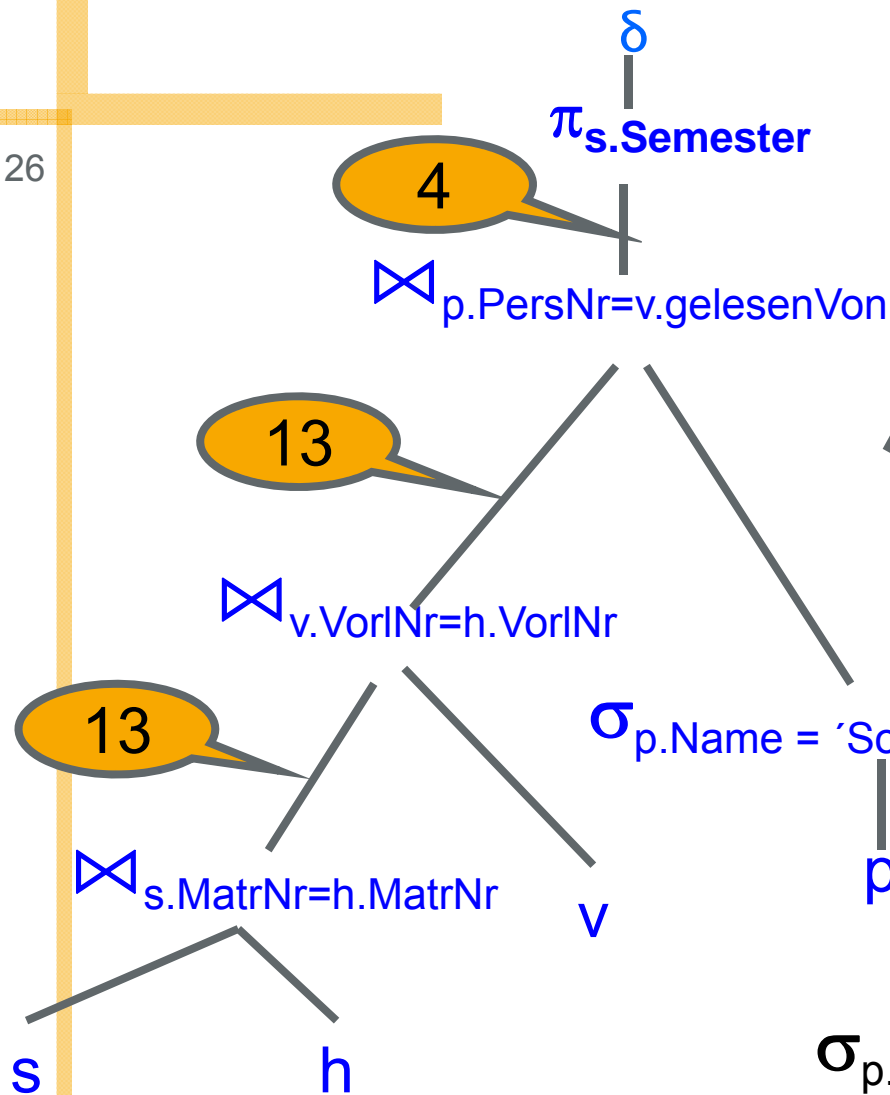
25



Folie: Prof. Alfons Kemper, TU München

# Was hat 's gebracht?

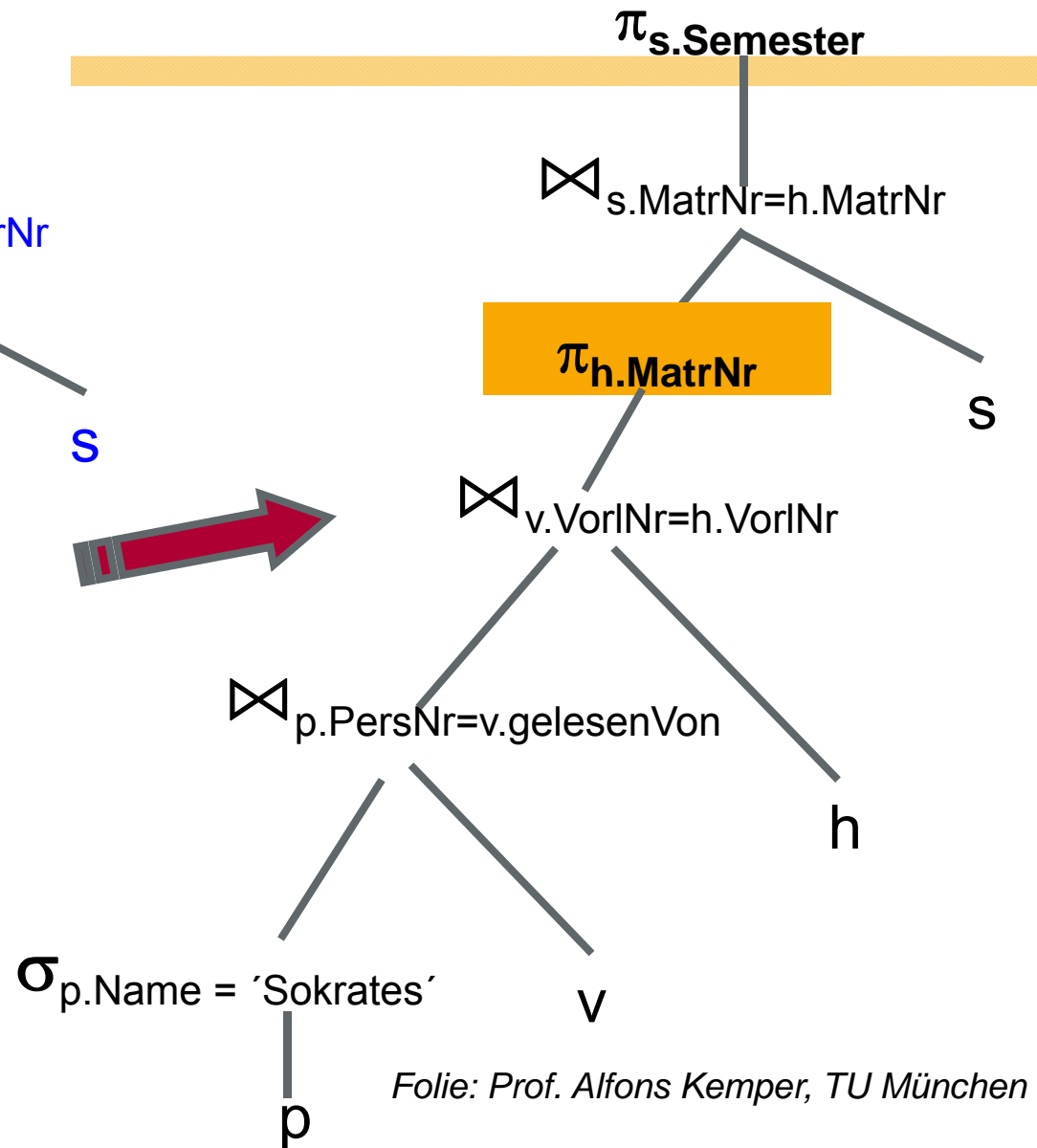
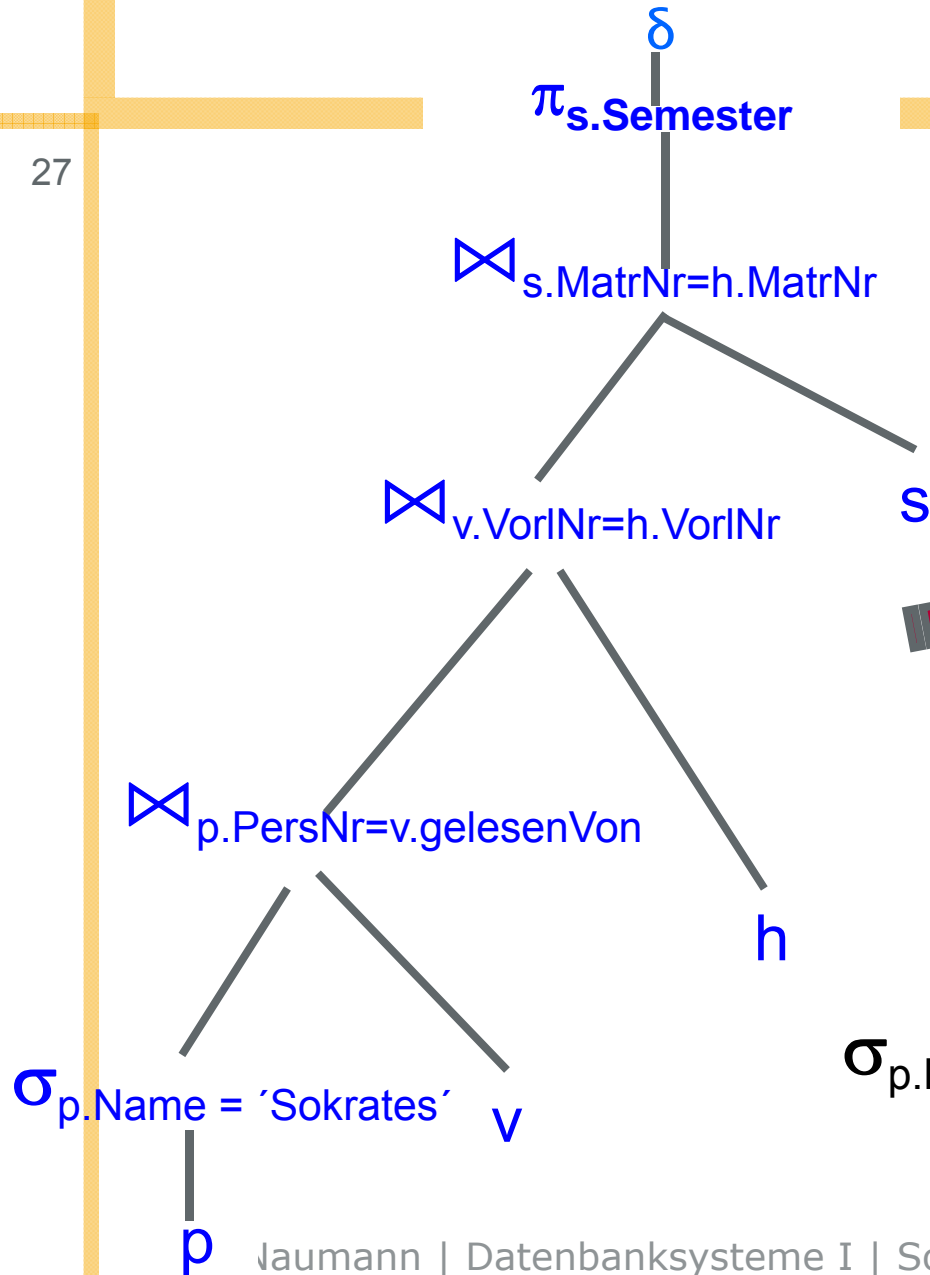
26



Folie: Prof. Alfons Kemper, TU München

# Einfügen von Projektionen

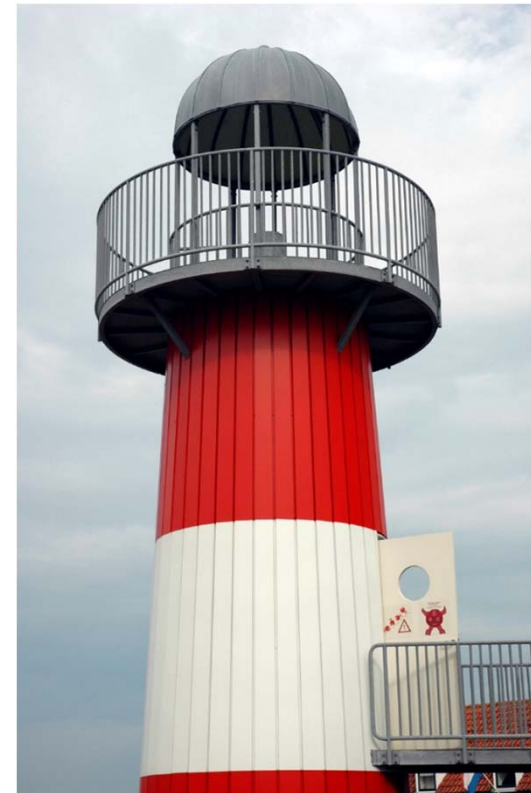
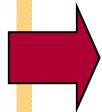
27



Folie: Prof. Alfons Kemper, TU München

28

- Parsen der Anfrage
- Transformationsregeln der RA
- Optimierung
- Kostenmodelle



# Kostenbasierte Optimierung

29

- Konzeptionell: Generiere alle denkbaren Anfrageausführungspläne.
- Bewerte deren Kosten anhand eines Kostenmodells
  - Statistiken und Histogramme
  - Kalibrierung gemäß verwendeter Rechner
  - Abhängig vom verfügbaren Speicher
  - Aufwands-Kostenmodell
    - ◇ Durchsatz-maximierend
    - ◇ Nicht Antwortzeit-minimierend
- Führe billigsten Plan aus

Achtung: Nicht zu lange optimieren!

# Problemgröße (Suchraum)

30

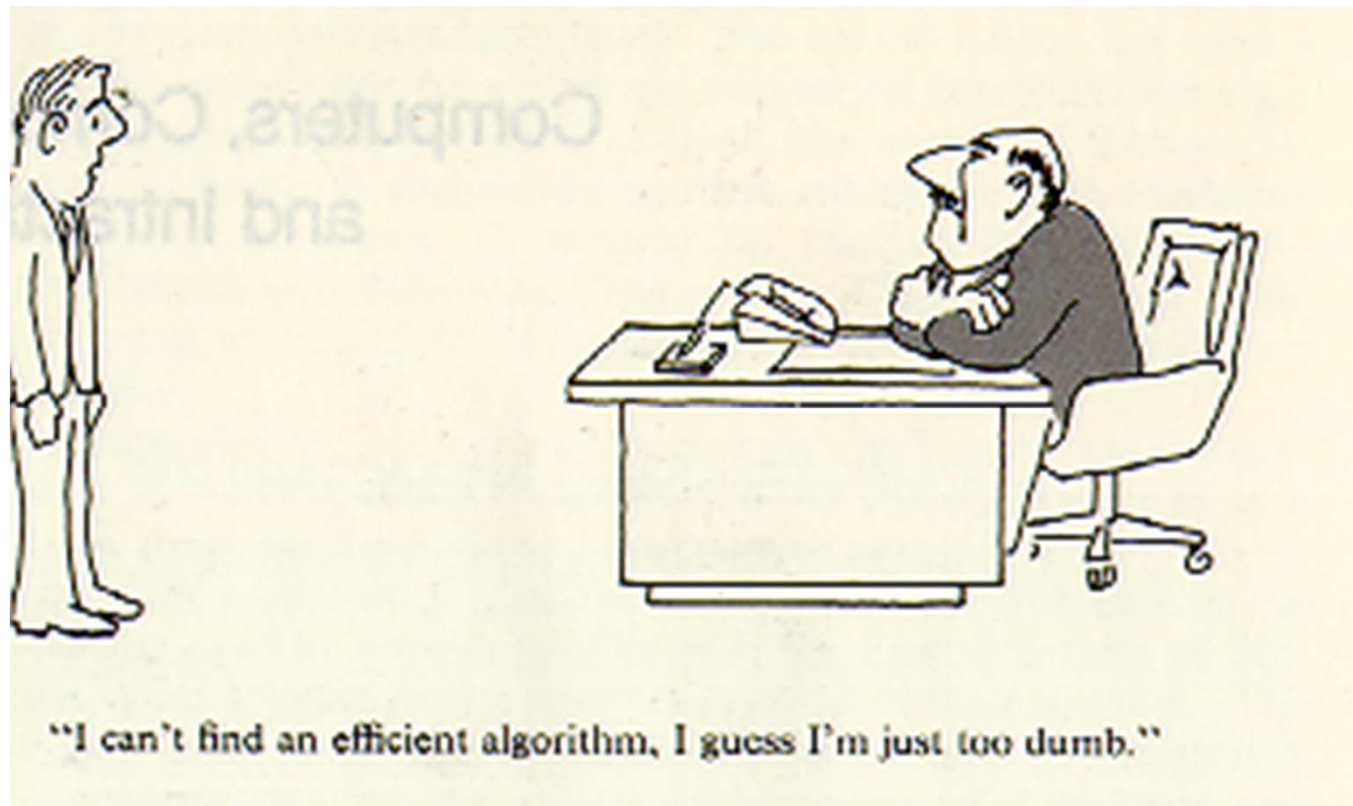
- Konzeptionell: Generiere alle denkbaren Anfrageausführungspläne
- Anzahl Bushy-Pläne mit n Tabellen

$$\frac{(2(n-1))!}{(n-1)!}$$

n	2 <sup>n</sup>	(2(n-1))!/(n-1)!
2	4	2
5	32	1.680
10	1.024	1,76*10 <sup>10</sup>
20	1.048.576	4,3*10 <sup>27</sup>

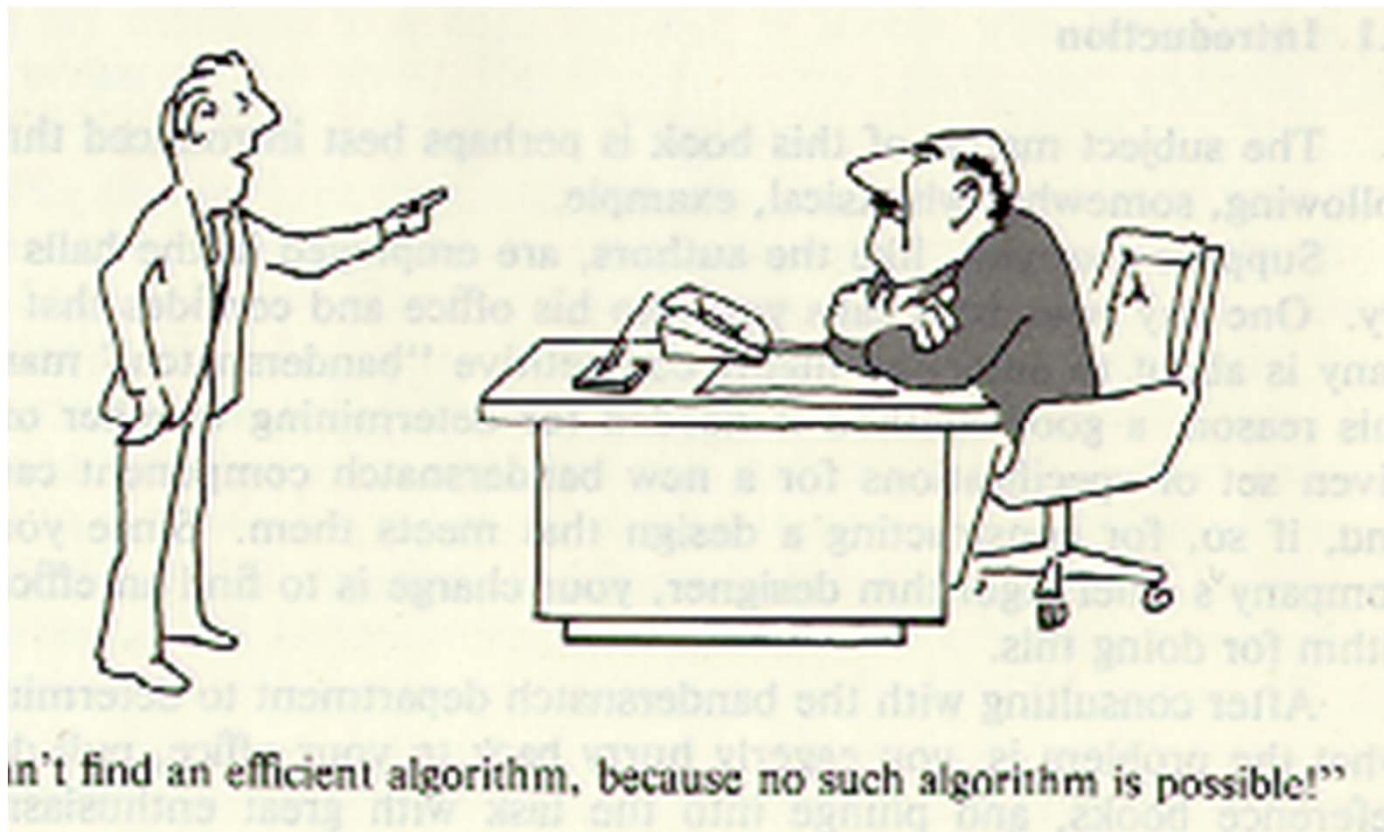
- Plankosten unterscheiden sich um viele Größenordnungen.
- Optimierungsproblem ist NP-hart
- Jetzt: Kleiner Ausflug in die NP-Vollständigkeit

*Folie: Prof. Alfons Kemper, TU München*

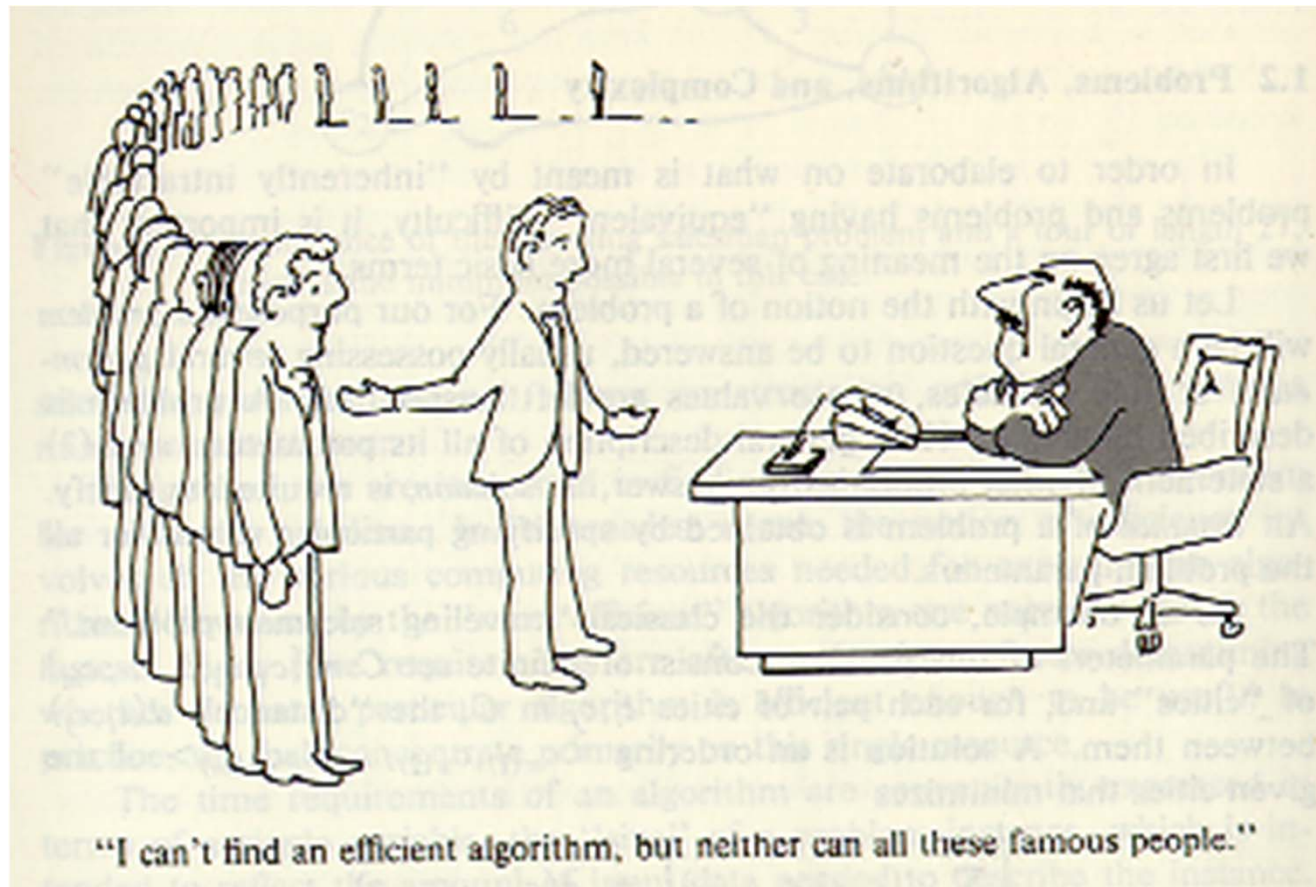


# Besser, aber schwierig

32







## NP-vollständig heißt NICHT:

35

- Nicht-Polynomial
- Exponentieller Suchraum
  - Gegenbeispiel: One-Prozessor Scheduling
- Exponentielle Laufzeit
  - vielleicht nur schlechter Algorithmus
- unlösbar bzw. eine gute Entschuldigung

# $M$ - Prozessor Scheduling

36

MS: Instanz: Jobs  $j \in J$ , Joblängen  $l(j)$ ,  $m$  Prozessoren, deadline  $D$   
Frage: Gibt es Schedule der Länge  $D$ ?

MS: NP-vollständig

MS + gleiche  $l(j)$ : polynomial

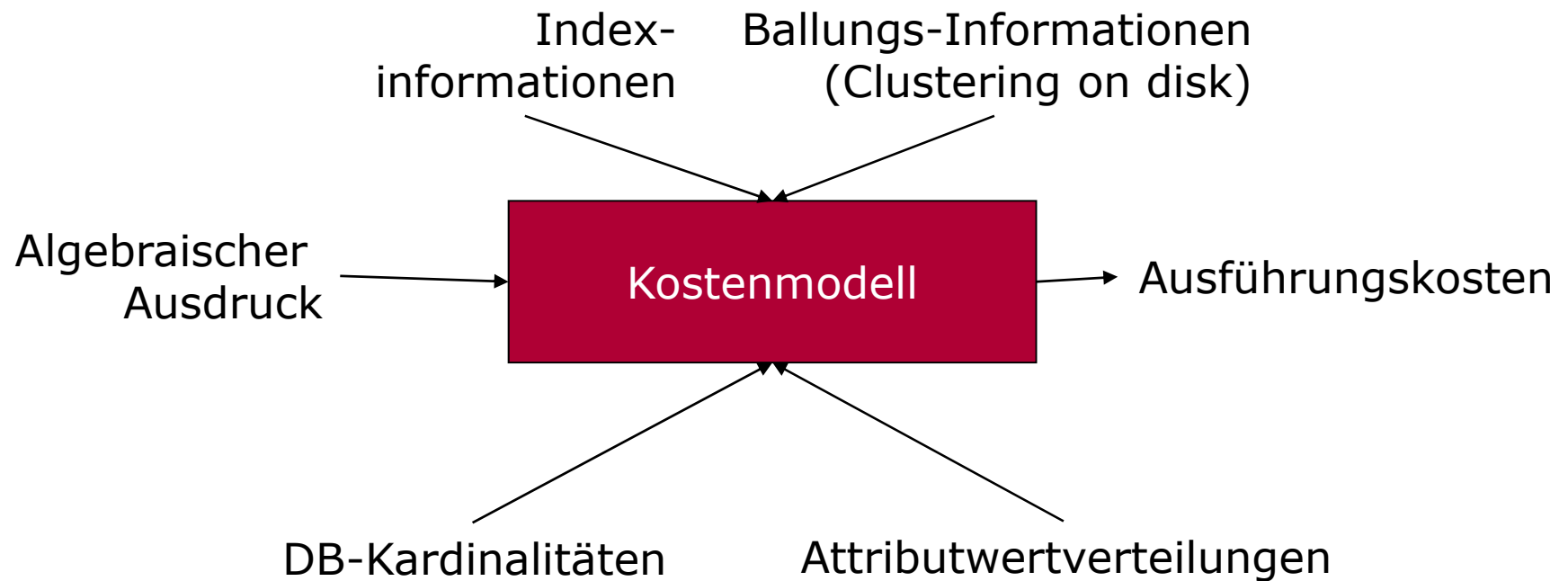
MS + gleiche  $l(j)$ + Halbordnung: NP-vollständig

MS + gleiche  $l(j)$ + Halbordnung + Wald: polynomial

MS + gleiche  $l(j)$ + Halbordnung + Wald + Prozessorzuordnung: NP-vollständig

MS + gleiche  $l(j)$ + Halbordnung + Wald + Prozessorzuordnung + festes  $m$  :  
polynomial

Verkleinerung des Suchraums bedeutet NICHT Vereinfachung des Problems!



*Folie nach Prof. Alfons Kemper, TU München*

# Statistiken

38

- Zu jeder Basisrelation
  - Anzahl der Tupel (Kardinalität)
  - Tupelgröße
- Zu (jedem) Attribut
  - Min / Max
  - Werteverteilung (Histogramm)
  - Anzahl der distinct Werte
- Zum System
  - Speichergröße
  - Bandbreite
  - I/O Zeiten
  - CPU Zeiten
- Problem: Erstellung und Update der Statistiken
  - Deshalb meist nur explizit/manuell zu initiieren
    - ◇ `runstats()`

# Kosten von Operationen

39

- Projektion:
  - Keine Kosten falls mit anderem Operator kombiniert
  
- Selektion
  - Ohne Index: Gesamte Relation von Festplatte lesen
  - Mit Baum-Index: Teil des Index von Platte lesen (Baumtiefe) und gesuchte Seite von Platte lesen
  - Bei Pipelining: (Fast) keine Kosten
  
- Join
  - Je nach Joinalgorithmus
  - Nested Loops, Hash-Join, Sort-Merge Join
  
- Sortierung: Nicht hier

# Kosten von Operationen

40

- Wesentliches Kostenmerkmal: Anzahl der Tupel im Input
  - Insbesondere: Passt die Relation in den Hauptspeicher?
  - Selektion, Projektion, Sortierung, Join
  
- Output ist Input des nächsten Operators.
  
- Deshalb: Ein Kostenmodell schätzt u.a. für jede Operation die Anzahl der Ausgabebetupel.
  - „Selektivität“ in Bezug auf Inputgröße
  - $\# \text{Ausgabebetupel} = \# \text{Eingabetupel} \times \text{Selektivität}$
  - Auch „Selektivitätsfaktor“ (*selectivity factor, sf*)

# Selektivität

41

- Selektivität schätzt Anzahl der qualifizierenden Tupel relativ zur Gesamtanzahl der Tupel in der Relation.
  
- Projektion:
  - $sf = |R|/|R| = 1$
  
- Selektion:
  - $sf = |\sigma_C(R)| / |R|$
  
- Join:
  - $sf = |R \bowtie S| / |R \times S| = |R \bowtie S| / (|R| \cdot |S|)$



# Selektivität schätzen

42

- Selektion:
  - Selektion auf einen Schlüssel:
    - ◇  $sf = 1 / |R|$
  - Selektion auf einen Attribut A mit m verschiedenen Werten:
    - ◇  $sf = (|R| / m) / |R| = 1/m$
    - ◇ Dies ist nur geschätzt!
- Join
  - Equijoin zwischen R und S über Fremdschlüssel in S
    - ◇  $sf = 1/ |R|$
    - ◇ „Beweis“:  $sf = |R \bowtie S| / (|R| \times |S|) = |S| / (|R| \cdot |S|)$

TABLE 1 SELECTIVITY FACTORS

column = value  
 $F = 1 / \text{ICARD}(\text{column index})$  if there is an index on column  
 This assumes an even distribution of tuples among the index key values.  
 $F = 1/10$  otherwise

column1 = column2  
 $F = 1/\text{MAX}(\text{ICARD}(\text{column1 index}), \text{ICARD}(\text{column2 index}))$   
 if there are indexes on both column1 and column2  
 This assumes that each key value in the index with the smaller cardinality has a matching value in the other index.  
 $F = 1/\text{ICARD}(\text{column-i index})$  if there is only an index on column-i  
 $F = 1/10$  otherwise

column > value (or any other open-ended comparison)  
 $F = (\text{high key value} - \text{value}) / (\text{high key value} - \text{low key value})$   
 Linear interpolation of the value within the range of key values yields F if the column is an arithmetic type and value is known at access path selection time.  
 $F = 1/3$  otherwise (i.e. column not arithmetic)  
 There is no significance to this number, other than the fact that it is less selective than the guesses for equal predicates for which there are no indexes, and that it is less than 1/2. We hypothesize that few queries use predicates that are satisfied by more than half the tuples.

column BETWEEN value1 AND value2  
 $F = (\text{value2} - \text{value1}) / (\text{high key value} - \text{low key value})$   
 A ratio of the BETWEEN value range to the entire key value range is used as the selectivity factor if column is arithmetic and both value1 and value2 are known at access path selection.  
 $F = 1/4$  otherwise  
 Again there is no significance to this choice except that it is between the default selectivity factors for an equal predicate and a range predicate.

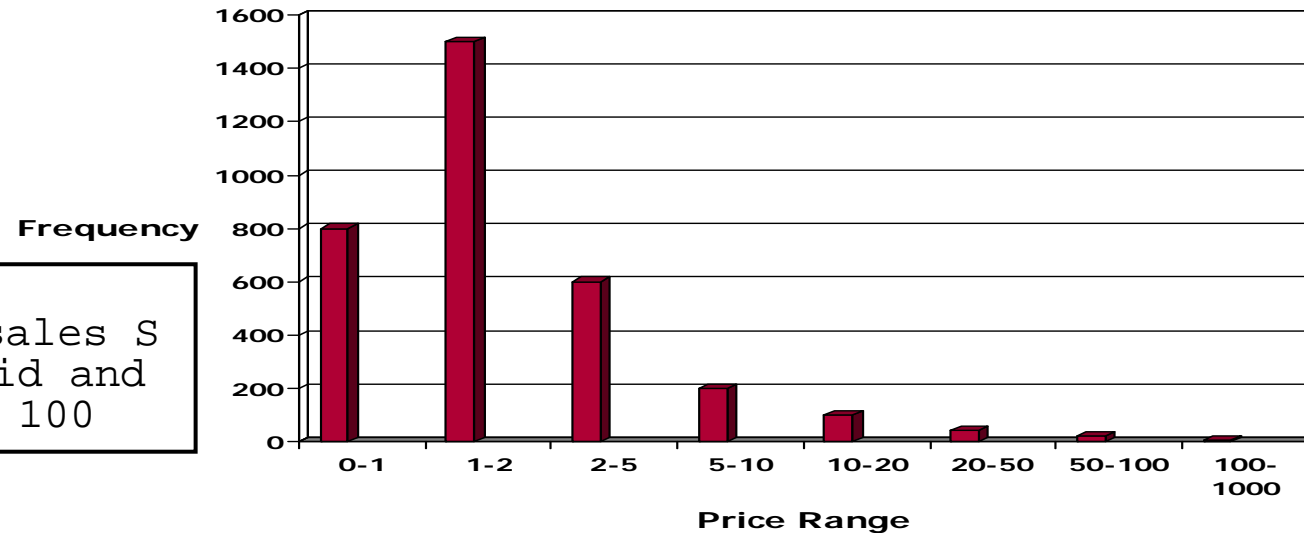
43

- Gleichverteilung der Werte
  - Platzsparend (count, max, min), einfach
  - Schlechte Abschätzung bei "skew" (ungleiche Verteilung)
- Histogramme (Beispiel gleich)
  - Parametrisierte Größe, einfach
  - Güte der Abschätzung hängt von Histogrammtyp und -größe ab.
  - Außerdem: Aktualität
- Sampling
  - Repräsentative Teilmenge der Relation
  - Parametrisierte Größe, schwierig zu finden
  - Güte hängt von Samplingmethode und Samplegröße ab
  - Außerdem: Aktualität

# Beispiel zu Histogrammen

45

```
SELECT *
FROM product p, sales S
WHERE p.id=s.p_id and
      p.price > 100
```



- Gegeben 3300 products, 1M sales
- Gleichverteilung
  - Preisspanne ist 0-1000 => Selektivität der Bedingung ist 9/10
    - ◇ Erwartet:  $9/10 * 3300 \approx 3000$  Produkte
- Histogramm-basiert
  - Angenommen 10 equi-width buckets
  - Selektivität der Bedingung ist  $5/3300 \approx 0,0015$  also 5 Produkte

# Kosten – Weitere Komplikationen

46

- Parallelität / Pipelining
  - Kosten aller Operatoren können nicht addiert werden.
  
- Hauptspeichergröße
  - Pufferung und Caching
  
- I/O Kosten (Lesen einer Seite) vs. CPU Kosten
  
- Multiuser: Durchsatz statt Antwortzeit
  
- => Kostenmodelle sind hochkomplex

# Ausblick auf DBS II

47

- Diverse Algorithmen für einzelne Operatoren
  - Insbesondere Join und Sortierung
- Kostenmodelle/Kostenschätzung genauer
- Optimale Joinreihenfolge: Dynamische Programmierung
- Physische Anfragepläne / Pipelining

