

UNIT 0x0c

TRANSAKTIONEN

ANFRAGEOPTIMIERUNG

Motivation

Transaktionen

- Situation: Mehrere Benutzer greifen auf das DBMS 'gleichzeitig' zu.

Q&A

- Was kann schon schiefgehen... ?
- Welche Aktionen sind betroffen?

Anfrageoptimierung

- Situation: Abfragen dauern sehr lange.

Q&A

- Welche Engpässe gibt es?
- Wie könnte man Anfragen im DBMS optimieren?

Transaktionen – Motivation

Standardbeispiel

- Bei einer Überweisung wird Geld von einem Konto auf ein anderes transferiert.
- Nur beide Aktionen zusammen sind zulässig, also Abbuchen und Einzahlen.
- Schlägt nur eine der Aktionen fehl, so darf der Datenbestand nicht verändert sein bzw. muss eine vollständige Rückabwicklung gewährleistet werden.

Konzept

- Aus logischer Sicht ist eine Transaktion *ein* (1) Arbeitspaket. So klein wie möglich aber so gross wie nötig, um alle Integritätsbedingungen einhalten zu können.
- Aus technischer Sicht ist eine Transaktion eine Folge von Datenbankoperationen mit einem definierten Beginn und einem definierten Abschluss.
- Die Transaktionsverwaltung ist eine nicht triviale Kernaufgaben eines Datenbanksystems mit Mehrbenutzerverwaltung.

Transaktionen im DBMS

Commit / Rollback(Abort) - Grundidee

- Es wird, explizit oder implizit, eine Transaktion gestartet (`START TRANSACTION`).
- Alle Befehle, die während einer laufenden Transaktion ausgeführt werden, verändern die lokale Sicht auf die Daten. Andere Benutzer oder andere Sessions sehen diese Änderungen erst einmal nicht.
- Läuft alles gut, werden alle Änderungen bestätigt (`COMMIT`). Tritt ein Fehler auf, wird die Transaktion abgebrochen und der Zustand vor Beginn der Transaktion wieder hergestellt (`ROLLBACK`).
- Je nach DBMS ist in der Standardeinstellung ein sog. `autocommit` aktiv, d.h. jeder Befehl wird implizit mit einem `COMMIT` oder `ROLLBACK` beendet, so dass alle Nutzer und Sessions alle erfolgreichen Änderungen automatisch sofort sehen oder aber im Fehlerfall nichts verändert wurde.
- Details dazu im SQL-Praktikum.

ACID

Anforderungen an ein DBMS - ACID-Prinzip

- Atomicity (Atomarität)
- Consistency (Konsistenz)
- Isolation (Nebenläufigkeit)
- Durability (Dauerhaftigkeit)

Ein Verständnis der Ideen hinter den Begriffen ist wichtig.
Wir stellen erst die Begriffe vor und danach die technische Umsetzung der Transaktionen im DBMS/in SQL und die Locking-Mechanismen.



Anmerkungen

- 'ACID' wird häufig zur Charakterisierung eines DBMS genannt, insbesondere auch für Nicht-Relationale DBMS. Dort gibt es aber auch 'analoge' Konzepte wie etwa 'BASE' (Basically Available, Soft state, Eventual Consistency)...
- 'Locking' und 'Transaktionen' sind wesentlich für Parallelität der Anwendung.
- Konzept der 'Transaktionen' ist nicht auf DBMS beschränkt, z.B. Daten im Speicher.

ACID

Anforderungen Atomicity / Atomarität

- Es gibt Transaktionen und eine Transaktion ist atomar in dem Sinne, dass entweder alle durch die Operationen der Transaktion bedingten Änderungen in der Datenbank umgesetzt werden oder gar keine.
→ "alles-oder-nichts"-Prinzip.
- Als Konsequenz benötigen wir einen Mechanismus in dem DBMS, um Transaktionen zu definieren bzw. zu starten, zu beenden oder abzubrechen.
- Beobachtung bisher: Alle SQL-Befehle wurden direkt ausgeführt und die Folgen waren sofort sichtbar...

ACID

Anforderungen Consistency / Konsistenz

- Ausgangspunkt ist eine konsistente Datenbank vor dem Beginn einer Transaktion.
- Am Ende einer Transaktion ist eine Datenbank in einem konsistentem Zustand, d.h.
 - entweder jede Integritätsregel ist erfüllt, etwa referentielle Integrität (Fremdschlüssel), oder
 - die Transaktion wird komplett zurückgesetzt und die Datenbank befindet sich wieder in dem (konsistenten) Zustand wie zu Beginn.
- Achtung: Während der Abarbeitung einer Transaktion können bestimmte Inkonsistenzen erlaubt sein, da muss man im konkreten DBMS nachlesen, ob dem so ist.

ACID

Anforderungen Isolation / Nebenläufigkeit

- Nebenläufige, also parallele bzw. gleichzeitig ausgeführte Transaktionen (generell Operationen) beeinflussen sich nicht.
- Logisch betrachtet: jede Transaktion hat exklusiven Zugriff auf die Datenbank und erhält 'Ihre' Version während der Abarbeitung der Transaktion.
- Achtung: In so einem Szenario ist es sehr leicht, 'falsche' Daten zu erzeugen. Es werden Konzepte benötigt, z.B. Sperren, um Effekte wie
 - **'lost updates'**
 - **'dirty reads'**zu verhindern. Diese Situationen werden im Anschluss vorgestellt.

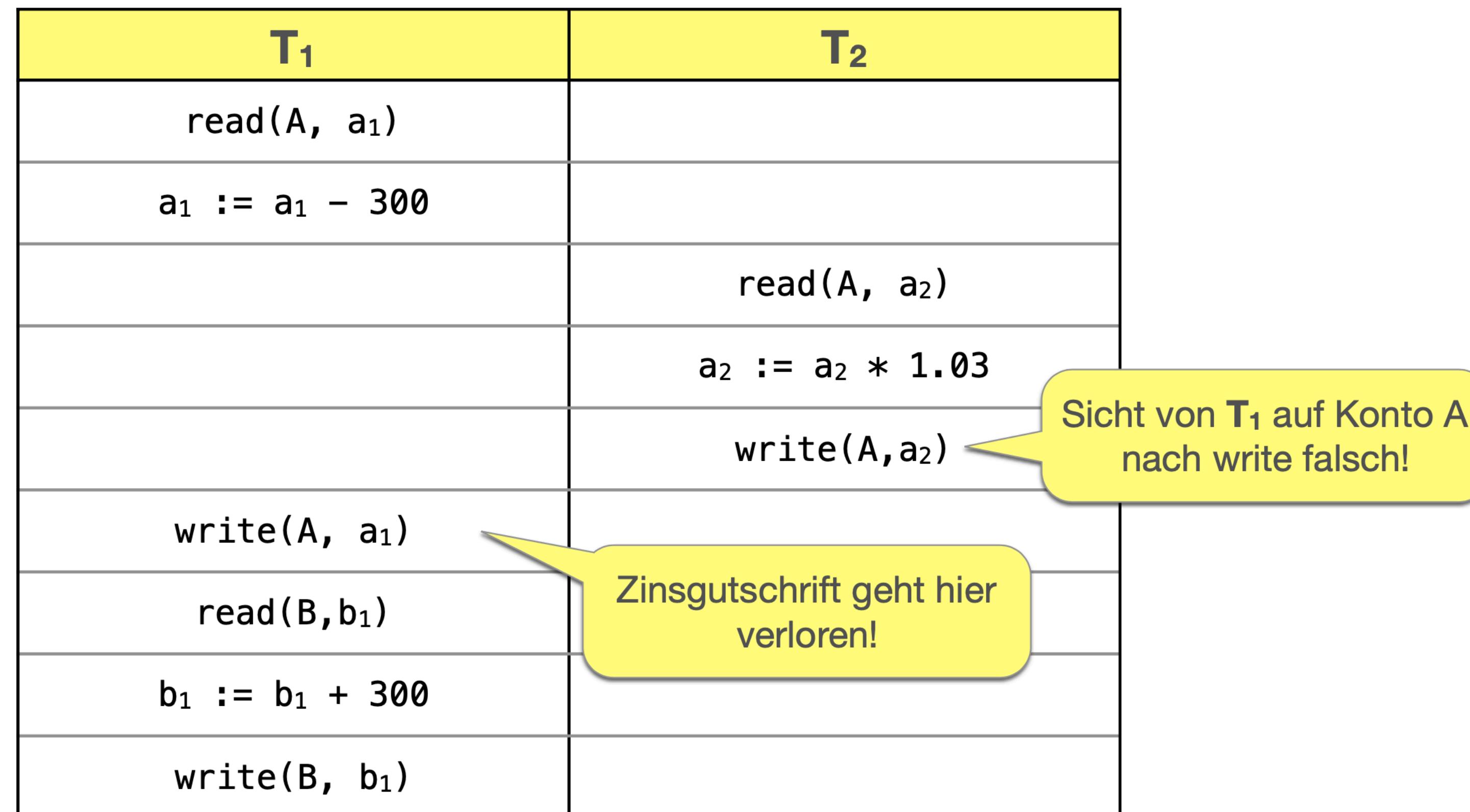
ACID

Anforderungen Durability / Dauerhaftigkeit

- Nach Abschluss einer Transaktion haben die von ihr ausgeführten Änderungen dauerhaft bestand. Das bedeutet, auch Prozessabstürze oder Plattenfehler dürfen nicht zu Datenverlust führen.
- Diese Anforderung bedeutet üblicherweise, dass Daten auf nicht-flüchtigen Speichermedien gespeichert (persistiert) werden.

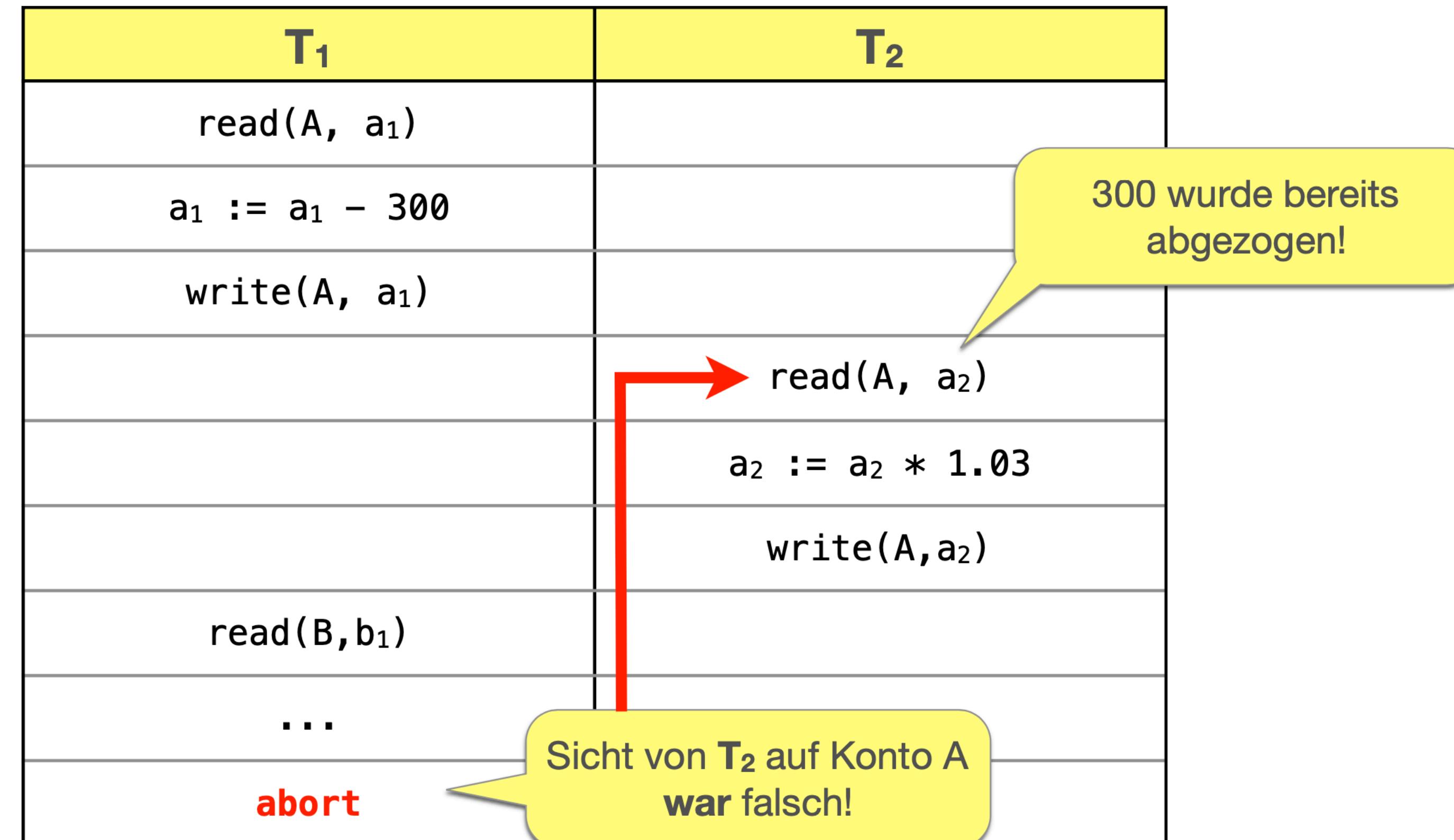
Nebenläufigkeit - Lost Update

- **Betrachte folgende, nebenläufige Transaktionen**
 - **T₁**: Buche Betrag 300 von Konto A auf Konto B
 - **T₂**: Schreibe Konto A 3% Zinsen zu



Nebenläufigkeit - Dirty Read

- **Betrachte folgende, nebenläufige Transaktionen**
 - T_1 : Buche Betrag 300 von Konto A auf Konto B
 - T_2 : Schreibe Konto A 3% Zinsen zu



Nebenläufigkeit - Locking

Je nach Operation (read / write) zwei Sperrmodi

- **S** (shared lock, Lesesperre):
 - S-Sperre auf Datum A ermöglicht Lesen mittels read. Mehrere S-Sperren auf ein Datum sind möglich
- **X** (exclusion lock, Schreibsperre):
 - X-Sperre auf Datum A ermöglicht Schreiben mittels write. Nur eine X-Sperre auf einem Datum ist möglich.
 - X-Sperre auf Datum A setzt voraus, dass keine S-Sperre auf A gesetzt ist!

		Aktuelle Locks auf Datum		
		kein Lock	S	X
Anforderung	S	✓	✓	✗
	X	✓	✗	✗

Verträglichkeitsmatrix

Nebenläufigkeit – Locking

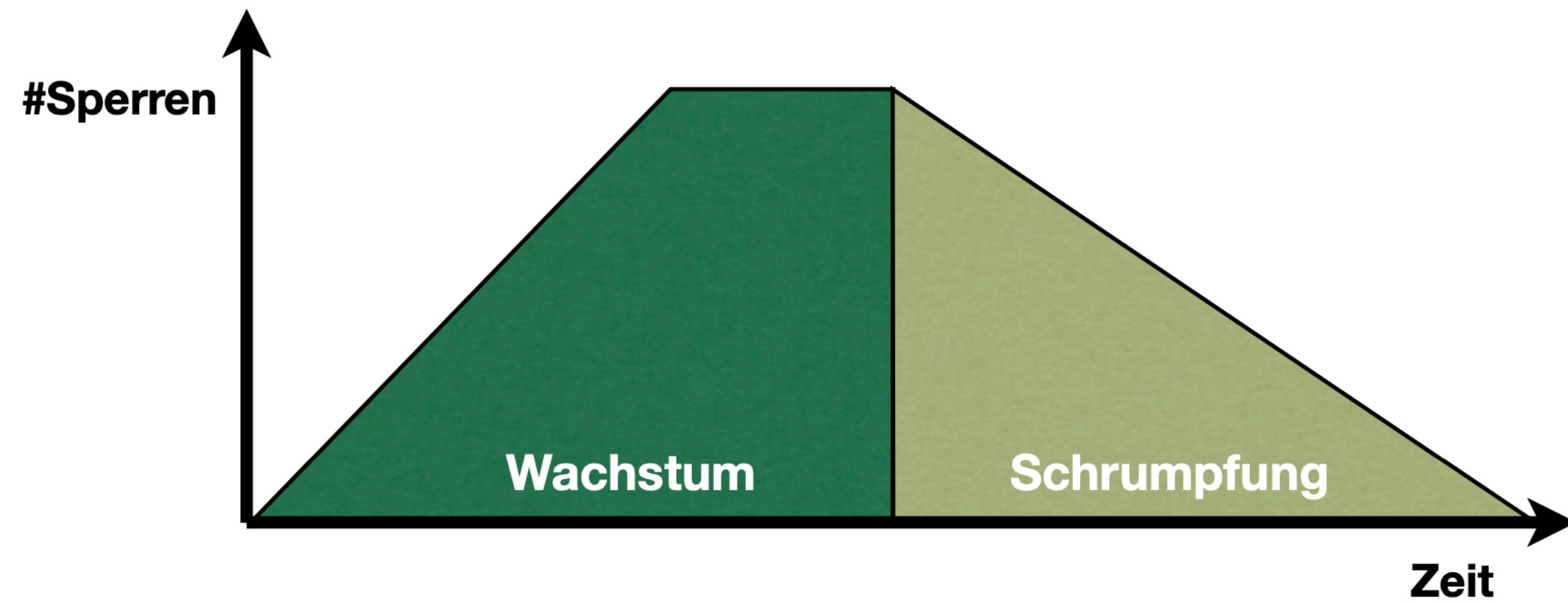
Folgende Regeln müssen eingehalten werden:

- Jedes Objekt muss vor der Benutzung gesperrt werden.
- Eine Transaktion fordert eine Sperre, die sie schon besitzt, nicht erneut an.
- Eine Transaktion respektiert vorhandene Sperren gemäß der Verträglichkeitsmatrix und wird ggf. in eine Warteschlange eingereiht.
- Jede Transaktion durchläuft zwei Phasen:
 - → Wachstumsphase (nur Sperren anfordern)
 - → Freigabephase (nur Sperren freigeben)
- Spätestens bei Transaktionsende muss eine Transaktion all ihre Sperren zurückgeben

Nebenläufigkeit - Locking

T ₁	T ₂	Bemerkung
BOT		
lockX(A)		
read(A)		
write(A)		
	BOT	
	lockS(A)	T ₂ muss warten – T ₁ hat X-Lock auf A
lockX(B)		
read(B)		
unlockX(A)		T ₂ wecken
	read(A)	
	lockS(B)	T ₂ muss warten – T ₁ hat X-Lock auf B
write(B)		
unlockX(B)		T ₂ wecken
	read(B)	
commit		
	unlockS(A)	
	unlockS(B)	
	commit	

Nebenläufigkeit - Locking



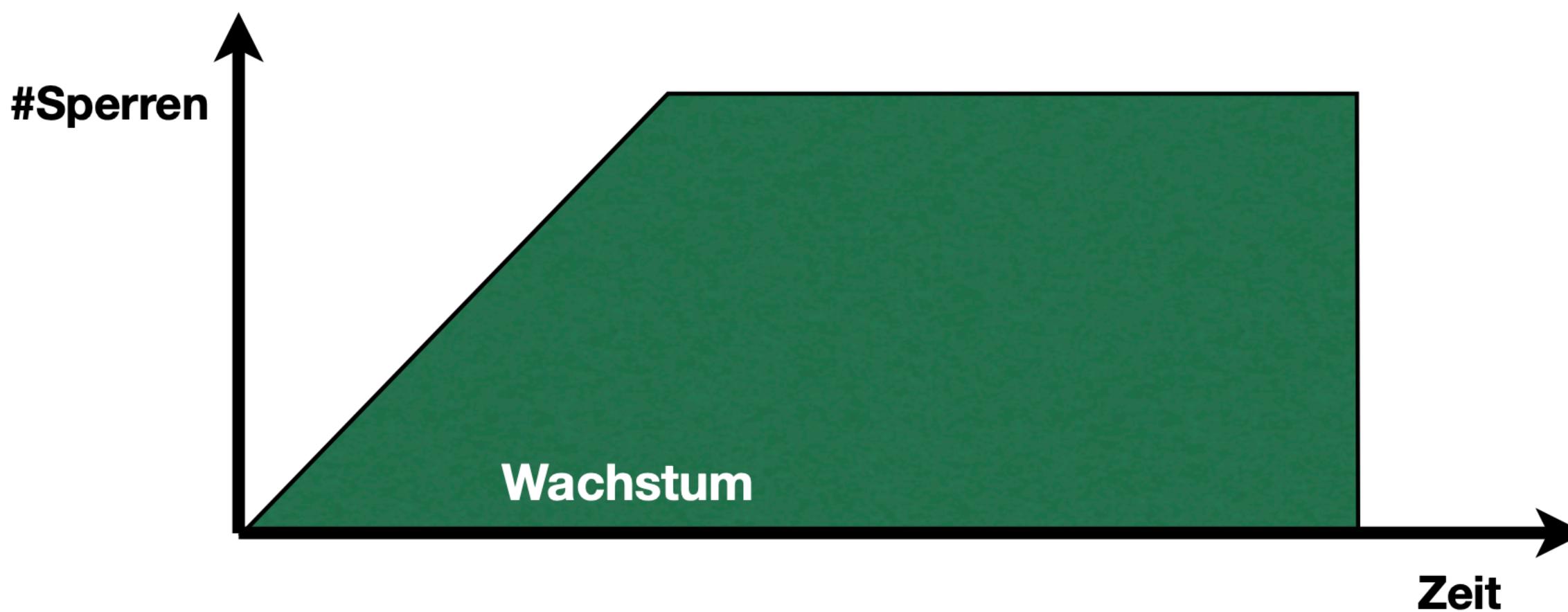
- **Rollback:** Rücknahme der Operationen einer abgebrochenen Transaktion
- Problem mit 2PL: Kaskadierendes Rollback

Nebenläufigkeit - Locking

T ₁	T ₂	Bemerkung
BOT lockX(A) read(A) write(A)	Rollback Alle Änderungen zurücknehmen	
lockX(B) read(B) unlockX(A)	BOT lockS(A)	T ₂ muss warten – T ₁ hat X-Lock auf A
write(B) unlockX(B)	read(A) lockS(B)	T ₂ wecken
abort Dirty Reads!	read(B)	T ₂ muss warten – T ₁ hat X-Lock auf A
	unlockS(A) unlockS(B)	T ₂ wecken
	commit	Kaskadiertes Rollback Alle Änderungen zurücknehmen

Nebenläufigkeit - Locking

- **Vermeidet kaskadierende Rollbacks**
- **Wie 2-Phasen-Sperrprotokoll, aber**
 - Keine Schrumpfungsphase
 - Alle Sperren werden erst zum Transaktionsende freigegeben



Nebenläufigkeit - Locking

T₁	T₂	Bemerkung
BOT		
lockX(A)		
read(A)		
write(A)		
	BOT	
	lockS(A)	T ₂ muss warten – T ₁ hat X-Lock auf A
lockX(B)		
read(B)		
write(B)		
unlockX(A)		
unlockX(B)		
abort		T ₂ wecken
 Unproblematisch Keine Dirty Reads!		
	read(A)	
	lockS(B)	
	read(B)	
	unlockS(A)	
	unlockS(B)	
	commit	

Nebenläufigkeit - Locking

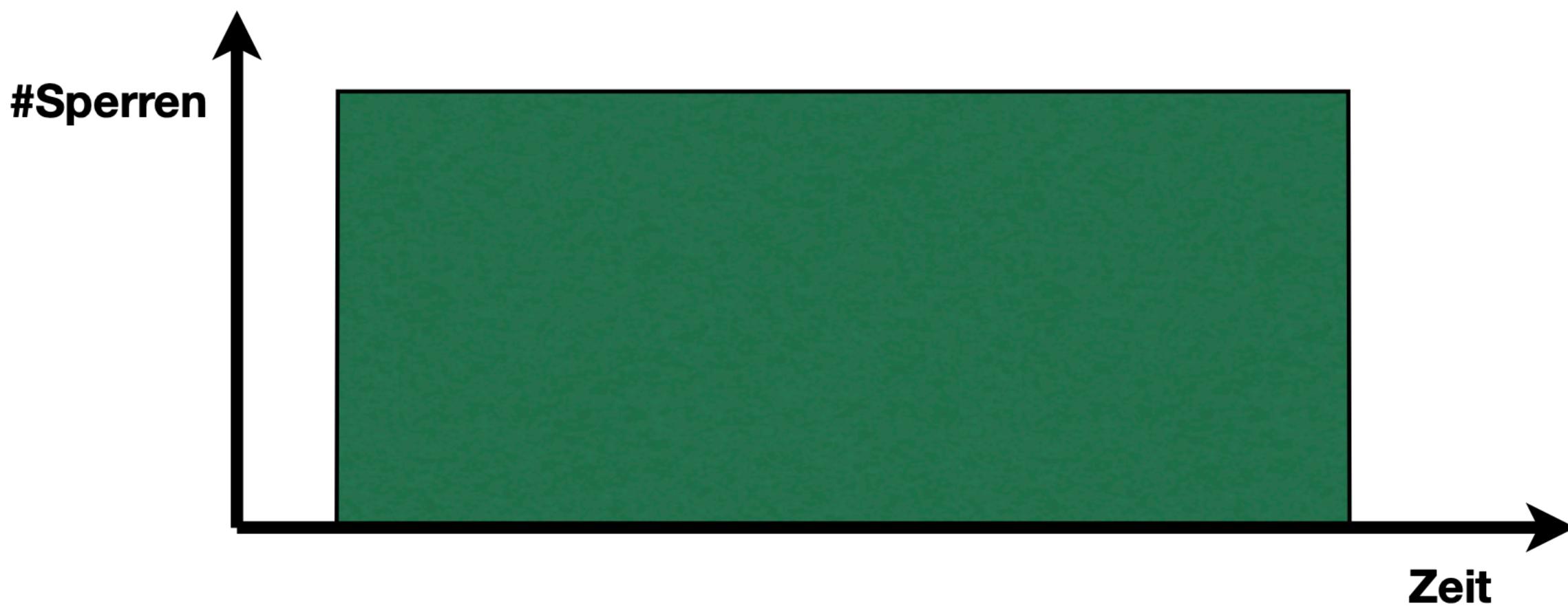
Inherentes Problem der auf Sperren basierte Synchronisation

- Verklemmungen (Deadlocks)

T ₁	T ₂	Bemerkung
B0T		
lockX(A)		
	B0T	
	lockS(B)	
	read(B)	
read(A)		
write(A)		
lockX(B)		T ₁ muss warten – T ₂ hat S-Lock auf B
	lockS(A)	T ₂ muss warten – T ₁ hat X-Lock auf A
Deadlock T ₁ und T ₂ blockieren sich gegenseitig		

Nebenläufigkeit - Locking

- **Idee: Notwendige Sperren mit BOT anfordern**
 - Konsequenz: keine ausgewiesene Wachstumsphase mehr



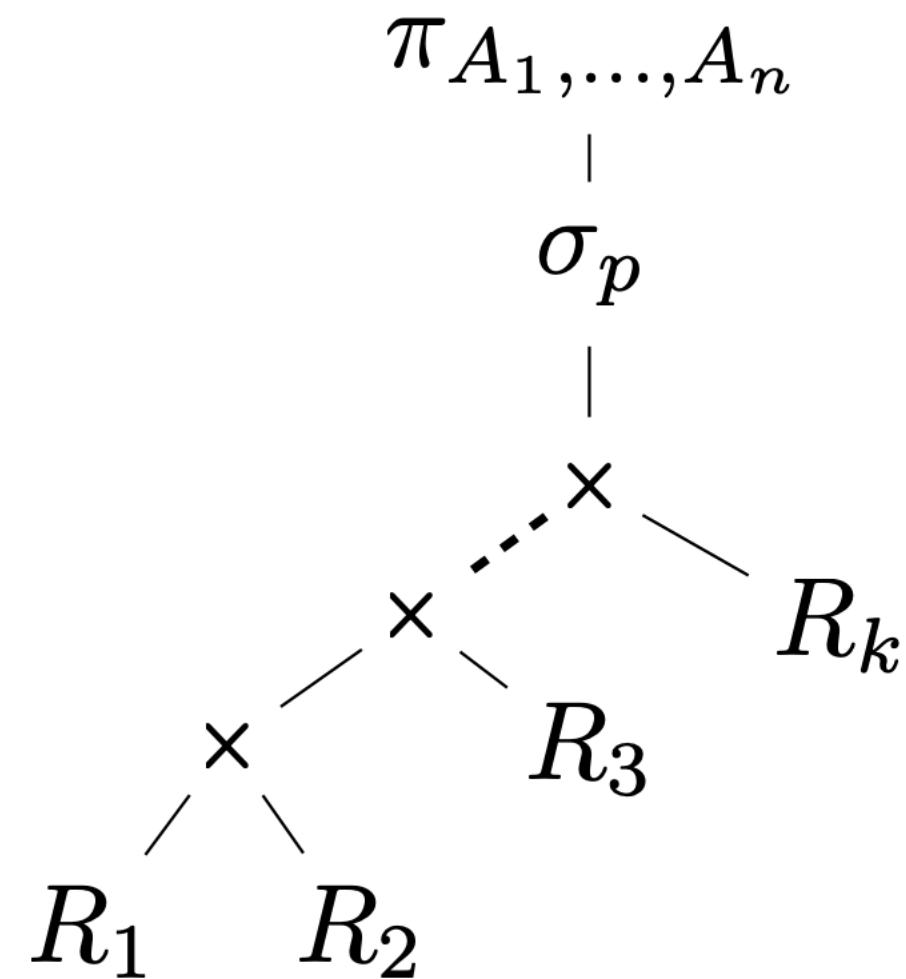
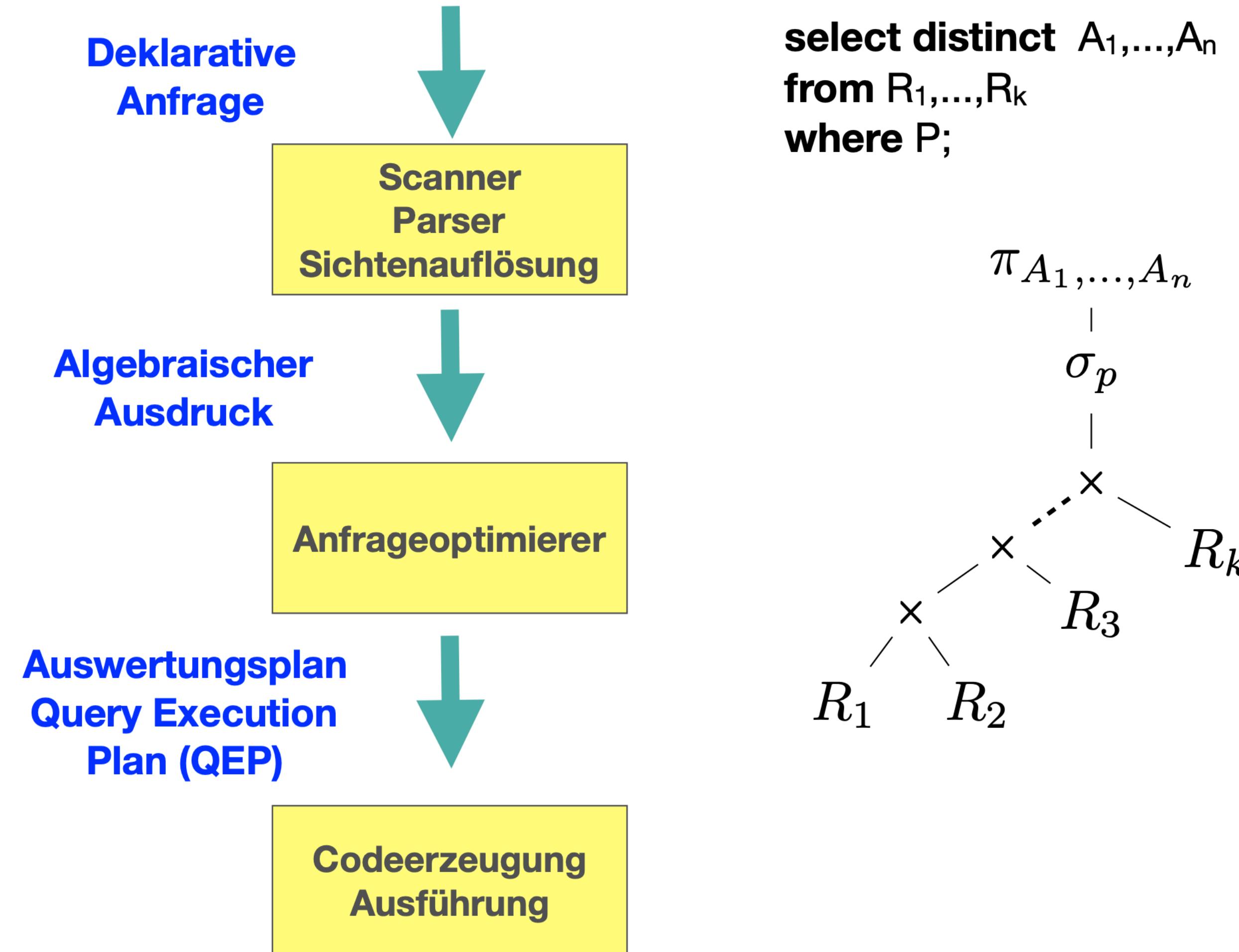
- **Problem:**
 - Notwendige Sperren nicht vorhersehbar (z.B. if .. then .. else ..)
 - **Daher:** Obermenge tatsächlich benötigter Sperren anfordern
 - das geht aber zu Lasten der Parallelität!

Nebenläufigkeit - Locking

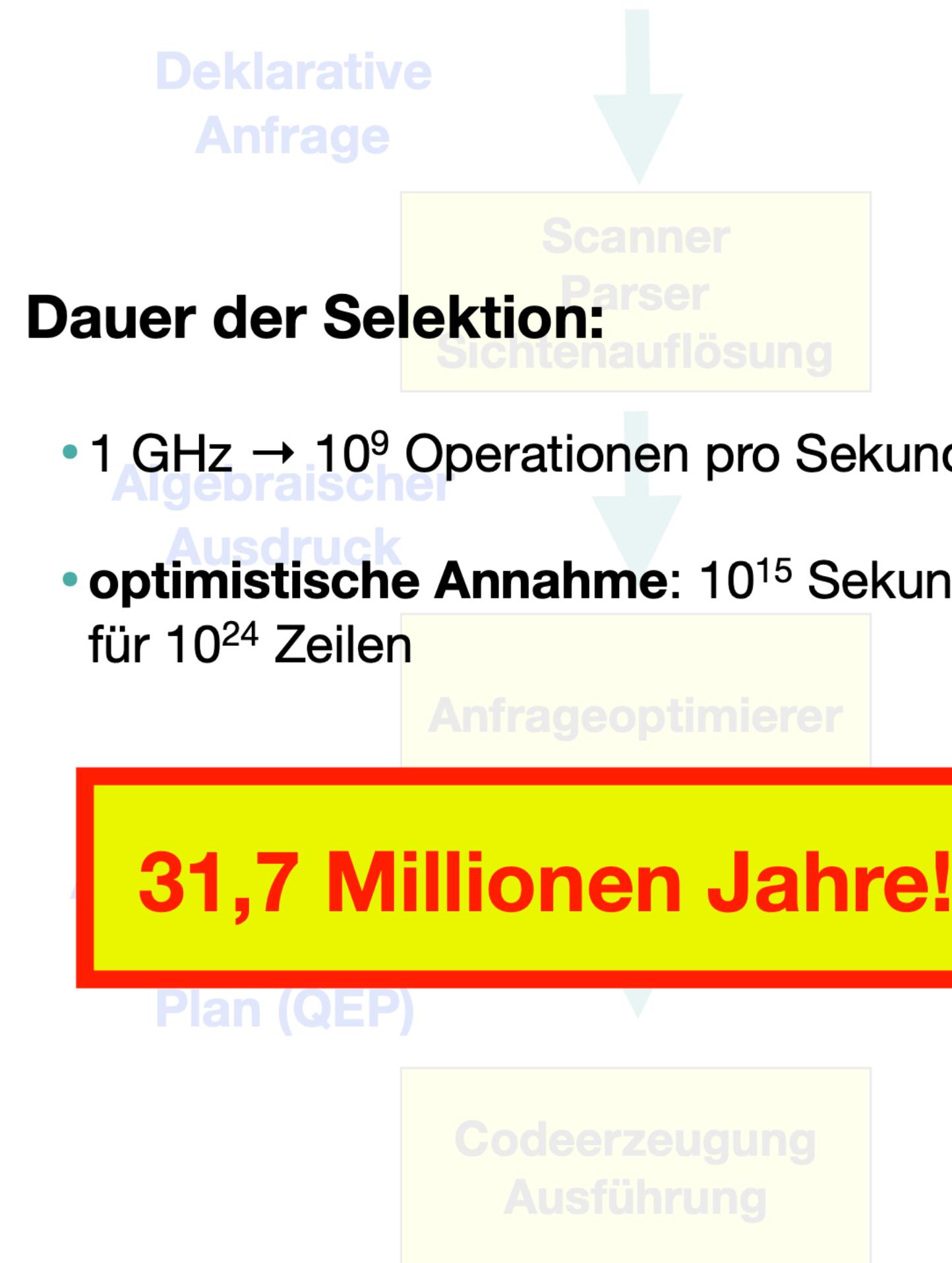
Anmerkungen

- Umgang mit Parallelität nicht DBMS-spezifisch.
- Beste Optimierung: 'Nachdenken' und z.B. Realisierung über möglichst unabhängige Aufteilung von Daten und/oder Aufgaben.
- Parallelität kostet!
- Zu restriktives Locking sealisiert de-fakto.

Ablauf der Anfrageoptimierung

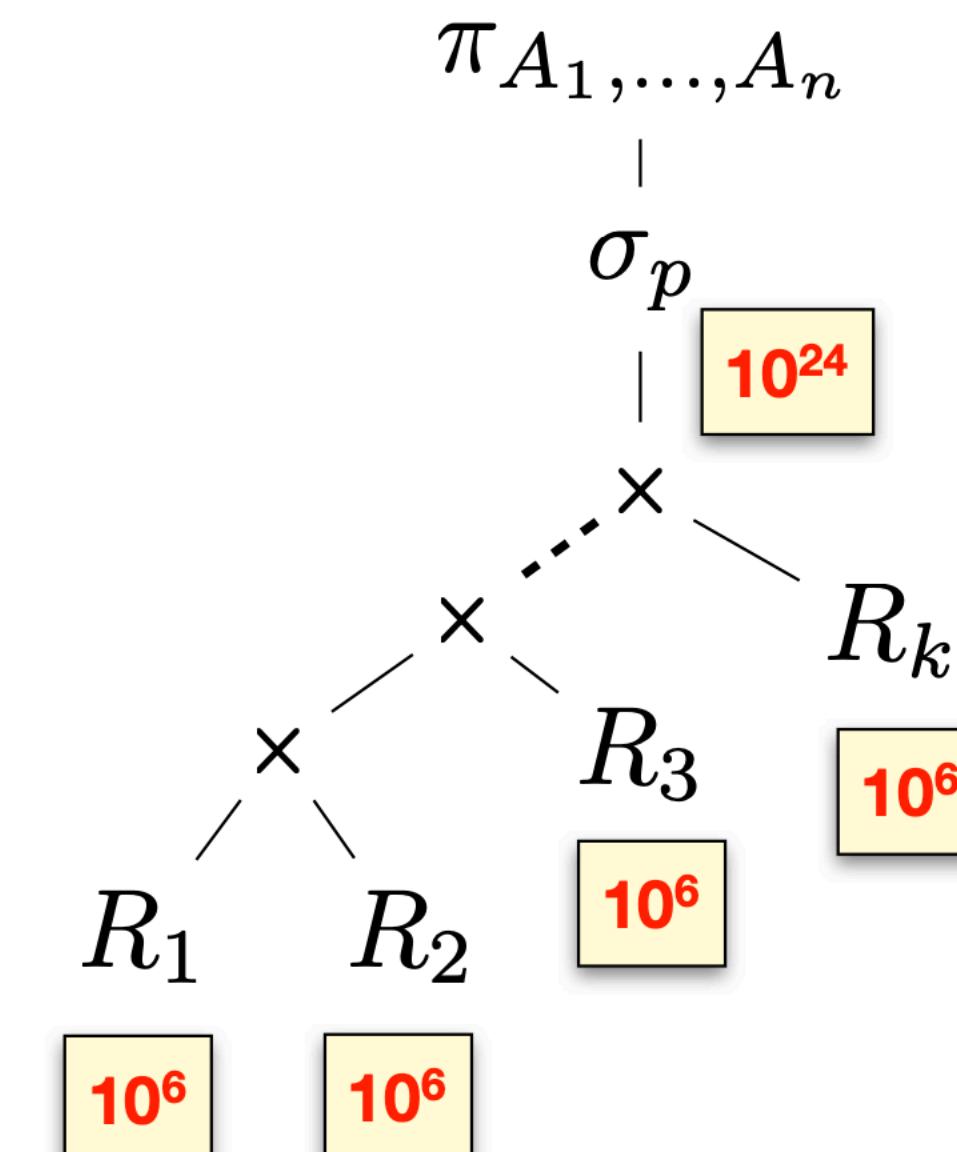


Ablauf der Anfrageoptimierung



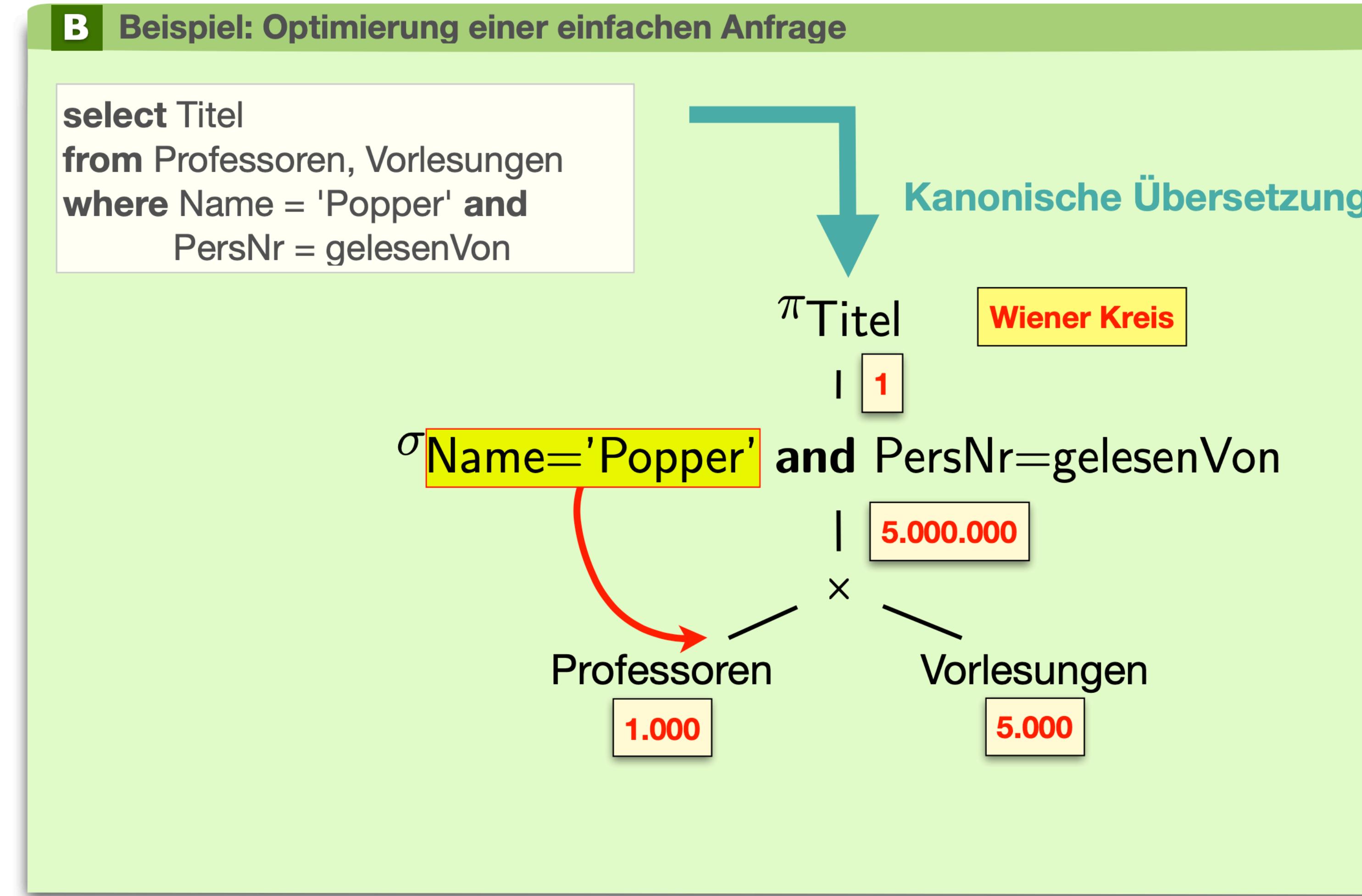
```

select distinct A1,...,An
from R1,...,Rk
where P;
  
```

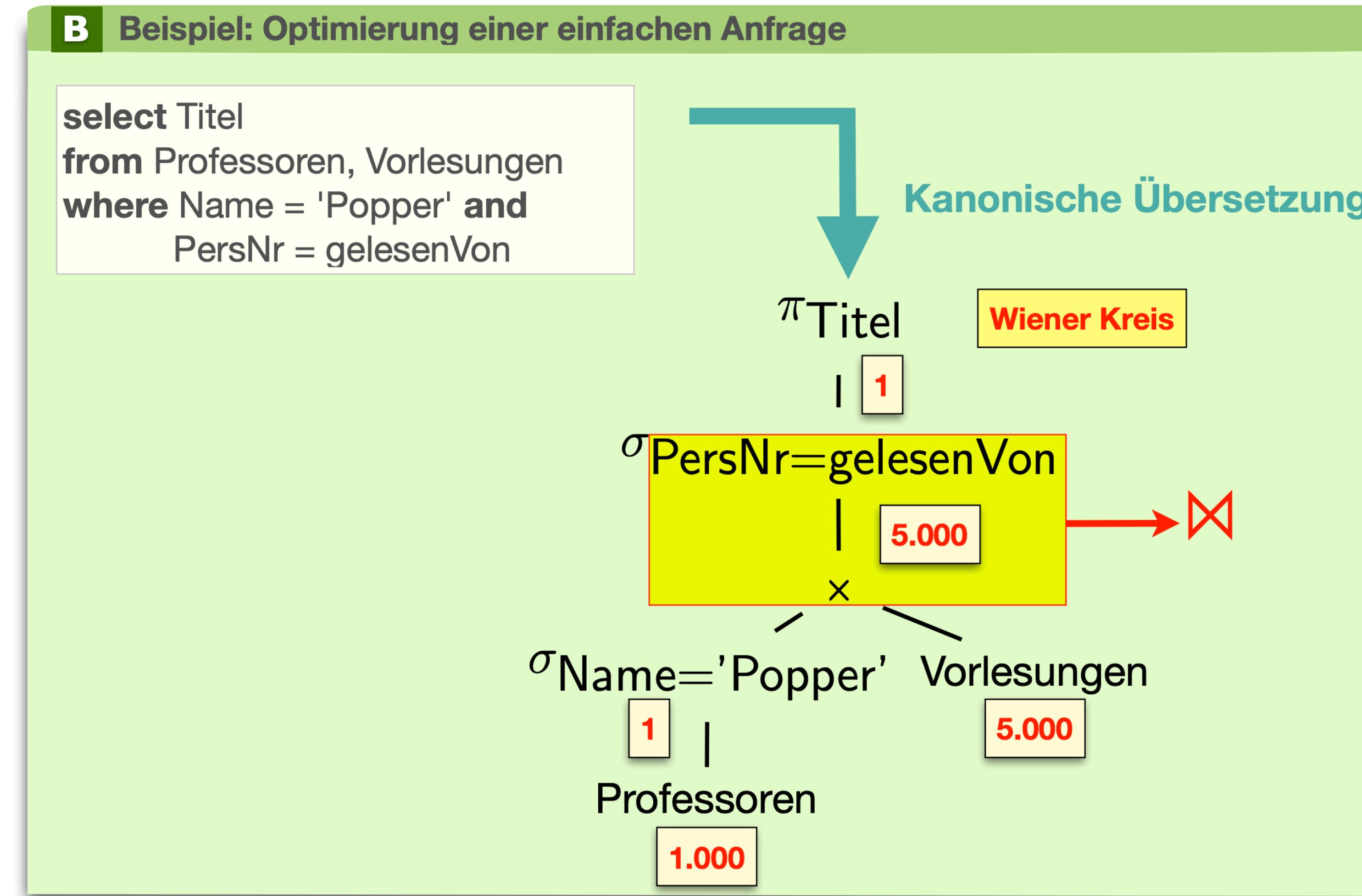


Kreuzprodukt von 4 Tabellen mit jeweils 1 Millionen Zeilen liefert Tabelle mit **1 Quadrillionen Spalten!**

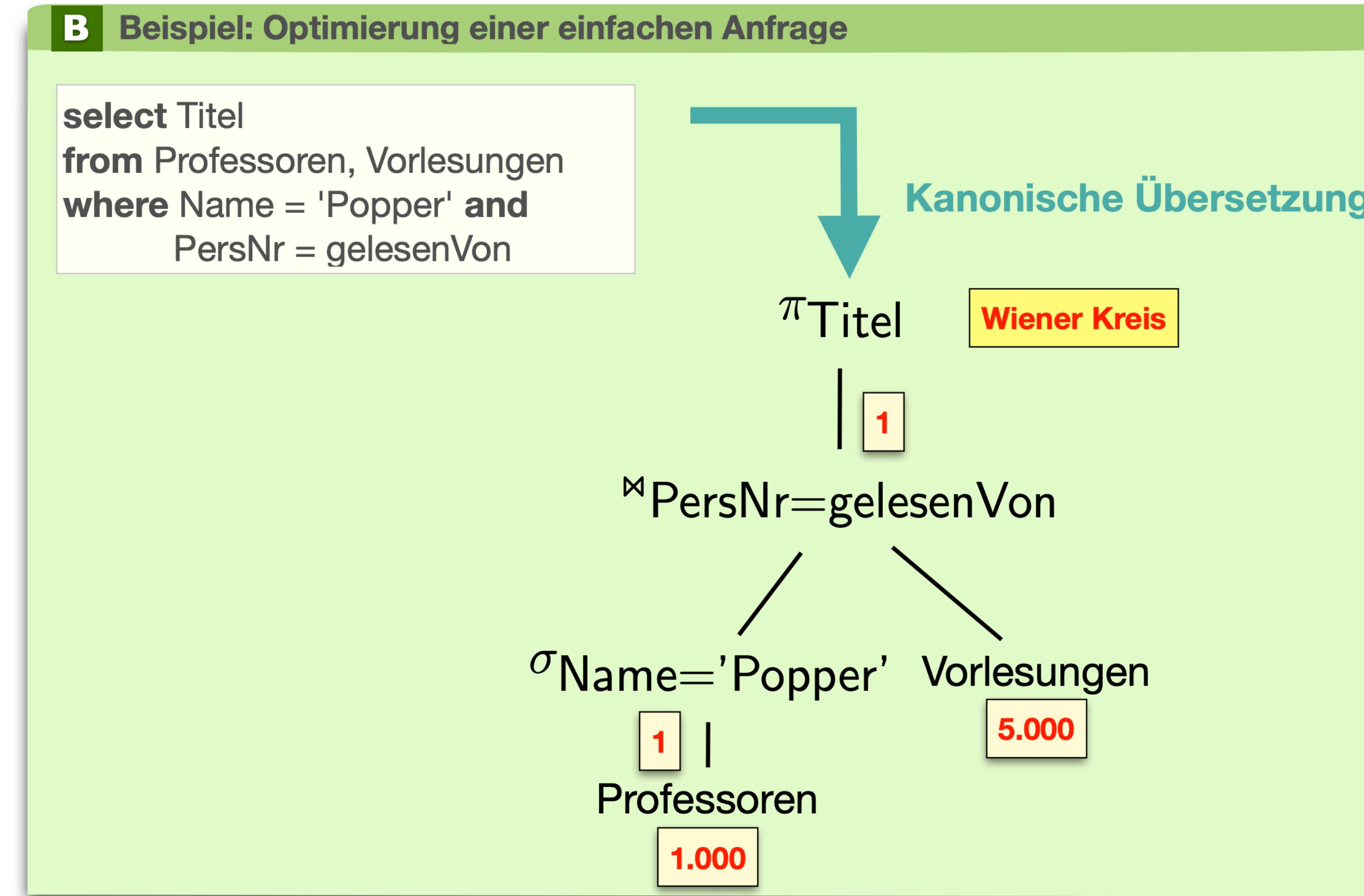
Beispiel: Einfache Anfrageoptimierung



Beispiel: Einfache Anfrageoptimierung



Beispiel: Einfache Anfrageoptimierung



Äquivalenzerhaltende Transformationen

1 Aufbrechen von Konjunktionen im Selektionsprädikat

$$\sigma_{C_1 \wedge C_2 \wedge \dots \wedge C_n} \equiv \sigma_{C_1}(\sigma_{C_2}(\dots(\sigma_{C_n}(R))\dots))$$

2 σ ist kommutativ

$$\sigma_{C_1}(\sigma_{C_2}(R)) \equiv \sigma_{C_2}(\sigma_{C_1}(R))$$

3 π -Kaskaden

- falls $L_1 \subseteq L_2 \subseteq \dots \subseteq L_n$

$$\pi_{L_1}(\pi_{L_2}(\dots(\pi_{L_n}(R))\dots)) \equiv \pi_{L_1}(R)$$

4 Vertauschen von σ und π

- Falls sich Selektionsprädikat C nur auf Attribute A_1, \dots, A_n der Projektionsliste bezieht:

$$\pi_{A_1, \dots, A_n}(\sigma_C(R)) \equiv \sigma_C(\pi_{A_1, \dots, A_n}(R))$$

Äquivalenzerhaltende Transformationen

5 \times , \cup , \cap und \bowtie sind kommutativ

- sei $\Theta = \{\times, \cup, \cap, \bowtie\}$

$$R\Theta S \equiv S\Theta R$$

6 Pushing Selections

- **Idee:** Selektiere so früh wie möglich
- Falls Selektionsprädikat C sich **nur** auf Attribute von R bezieht:

$$\sigma_C(R \bowtie_{C_J} S) \equiv \sigma_C(R) \bowtie_{C_J} S$$

- **Anmerkung:** Falls Selektionsprädikat C eine Konjunktion der Form $C_1 \wedge C_2$ ist und
 - C_1 sich nur auf Attribute von R bezieht und
 - C_2 sich nur auf Attribute von S bezieht

$$\sigma_{C_1 \wedge C_2}(R \bowtie_{C_J} S) \equiv \sigma_{C_1}(R) \bowtie_{C_J} \sigma_{C_2}(S)$$

Äquivalenzerhaltende Transformationen

7

Vertauschen von π mit \bowtie

- **Idee:** Projeziere so früh wie möglich!
- Sei die Projektionsliste $L = \{A_1, \dots, A_n, B_1, \dots, B_m\}$, wobei $\{A_1, \dots, A_n\} \subseteq R$ und $\{B_1, \dots, B_m\} \subseteq S$ und beziehe sich das Joinprädikat C_J **ausschließlich** auf Attribute aus L , dann

$$\pi_L(R \bowtie_{C_J} S) \equiv \pi_{A_1, \dots, A_n}(R) \bowtie_{C_J} \pi_{B_1, \dots, B_m}(S)$$

- Bezieht sich das Joinprädikat C_J auf weitere Attribute $\{A'_1, \dots, A'_k\} \subseteq R$ und $\{B'_1, \dots, B'_l\} \subseteq S$, so müssen diese für den Join erhalten bleiben und können erst danach herausprojiziert werden:

$$\pi_L(R \bowtie_{C_J} S) \equiv \pi_L(\pi_{A_1, \dots, A_n, A'_1, \dots, A'_k}(R) \bowtie_{C_J} \pi_{B_1, \dots, B_m, B'_1, \dots, B'_l}(S))$$

Äquivalenzerhaltende Transformationen

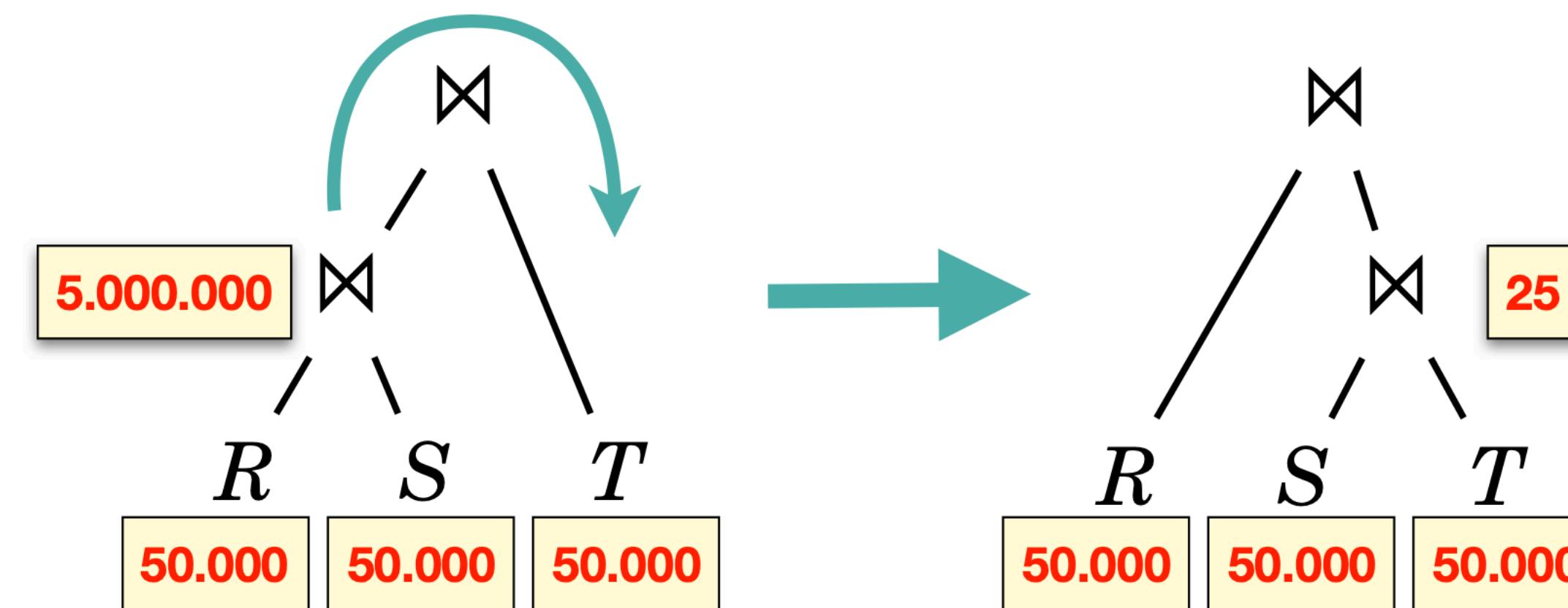
8

Die Operationen $\bowtie, \times, \cup, \cap$ sind jeweils assoziativ

- seit $\Theta = \{\times, \cup, \cap, \bowtie\}$

$$(R \Theta S) \Theta T \equiv R \Theta (S \Theta T)$$

- Anmerkung: Regel hat in Zusammenhang mit Join viel Potential!
 - Merke: DBMS führt intern Statistiken z.B. über Tabellengrößen



Äquivalenzerhaltende Transformationen

9 σ ist distributiv mit \cup, \cap, \neg

- se sei $\Theta = \{\cup, \cap, \neg\}$

$$\sigma_C(R \Theta S) \equiv \sigma_C(R) \Theta \sigma_C(S)$$

10 π ist distributiv mit \cup

$$\pi_C(R \cup S) \equiv \pi_C(R) \cup \pi_C(S)$$

11 De Morgans Regeln für Join- und Selektionsprädikate

- $\neg(a \wedge b) \equiv \neg a \vee \neg b$
- $\neg(a \vee b) \equiv \neg a \wedge \neg b$

Vorsicht bei NULL-Werten!

Äquivalenzerhaltende Transformationen

12

Zusammenfassung von σ und \times zu \bowtie

- Kartesisches Produkt $R \times S$, das von Selektionsoperation mit Prädikat C gefolgt wird und bei dem C nur Attribute aus $R \cup S$ enthält, kann in Join überführt werden:

$$\sigma_C(R \times S) \equiv R \bowtie_C S$$

Heuristik: Optimieren

Ziel: worst case vermeiden, nicht die optimale Anfrage!

- Mittels Regel 1 werden konjunktive Selektionsprädikate in Kaskaden von σ -Operationen zerlegt.
- Mittels Regeln 2, 4, 6, und 9 werden Selektionsoperationen soweit „nach unten“ propagiert wie möglich.
- Mittels Regel 8 (Assoziativgesetz) werden die Blattknoten so vertauscht, dass derjenige, der das kleinste Zwischenergebnis liefert, zuerst ausgewertet wird.
- Eine \times -Operation, die von einer σ -Operation gefolgt wird, wird eine \bowtie -Operation.
- Mittels Regeln 3, 4, 7, und 10 werden Projektionen soweit wie möglich nach unten propagiert.
- Versuche Operationen zusammenzufassen, wenn sie in einem "Durchlauf" ausführbar sind (z.B. Anwendung von Regel 1, Regel 3, aber auch Zusammenfassung aufeinanderfolgender Selektionen und Projektionen zu einer „Filter“-Operation).

Beispiel: Anfrageoptimierung

B Beispiel: Anwendung der Heuristik auf einfache Anfrage

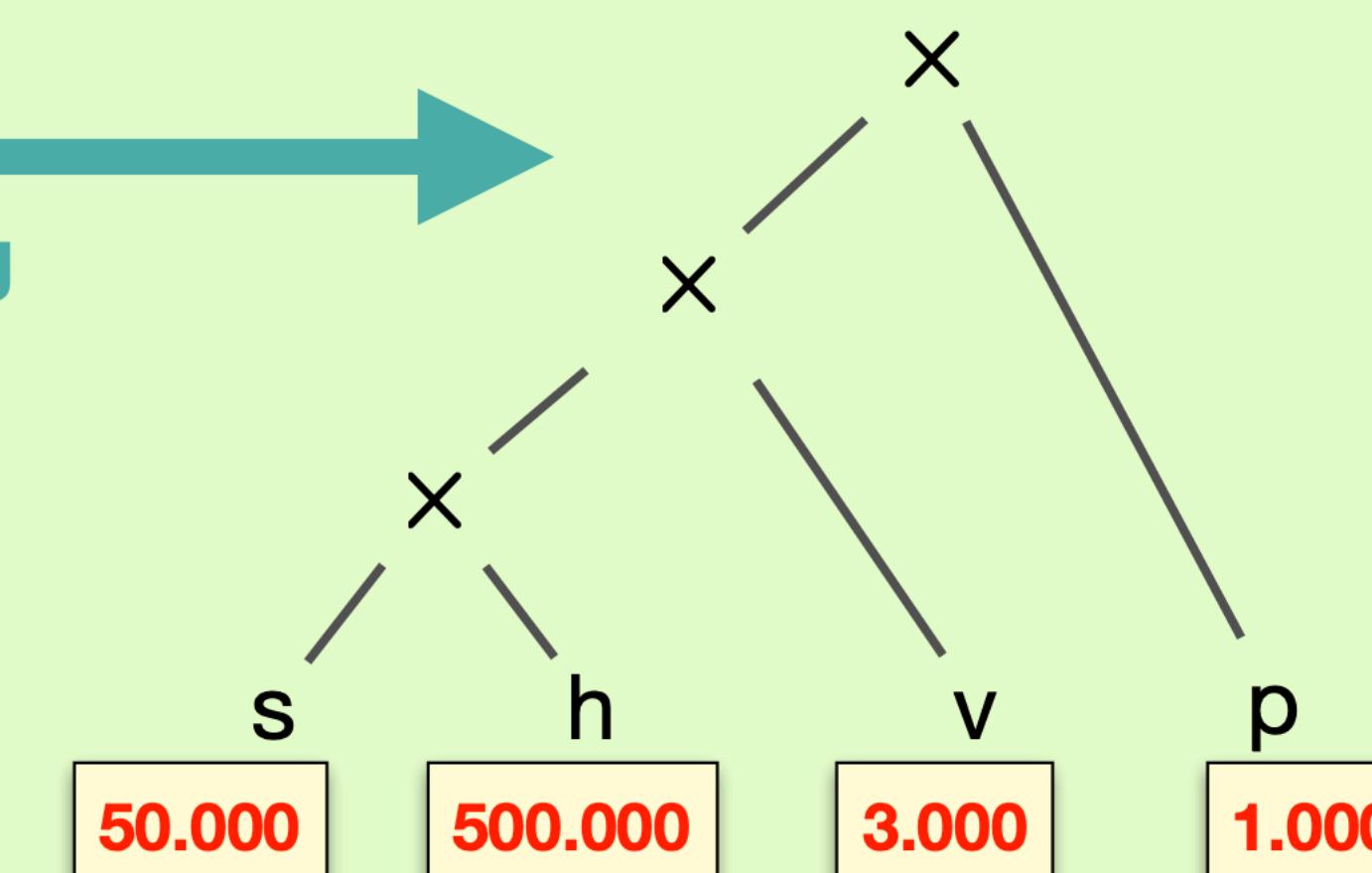
```
// In welchen Semestern befinden sich Studenten,
// die eine Vorlesung von Sokrates besuchen
select distinct s.Semester
from Studenten s, Professoren p,
      Vorlesungen v, hören h
where p.Name = 'Sokrates' and
      v.gelesenVon = p.PersNr and
      v.VorlNr = h.VorlNr and
      h.MatrNr = s.MatrNr
```

Kanonische Übersetzung

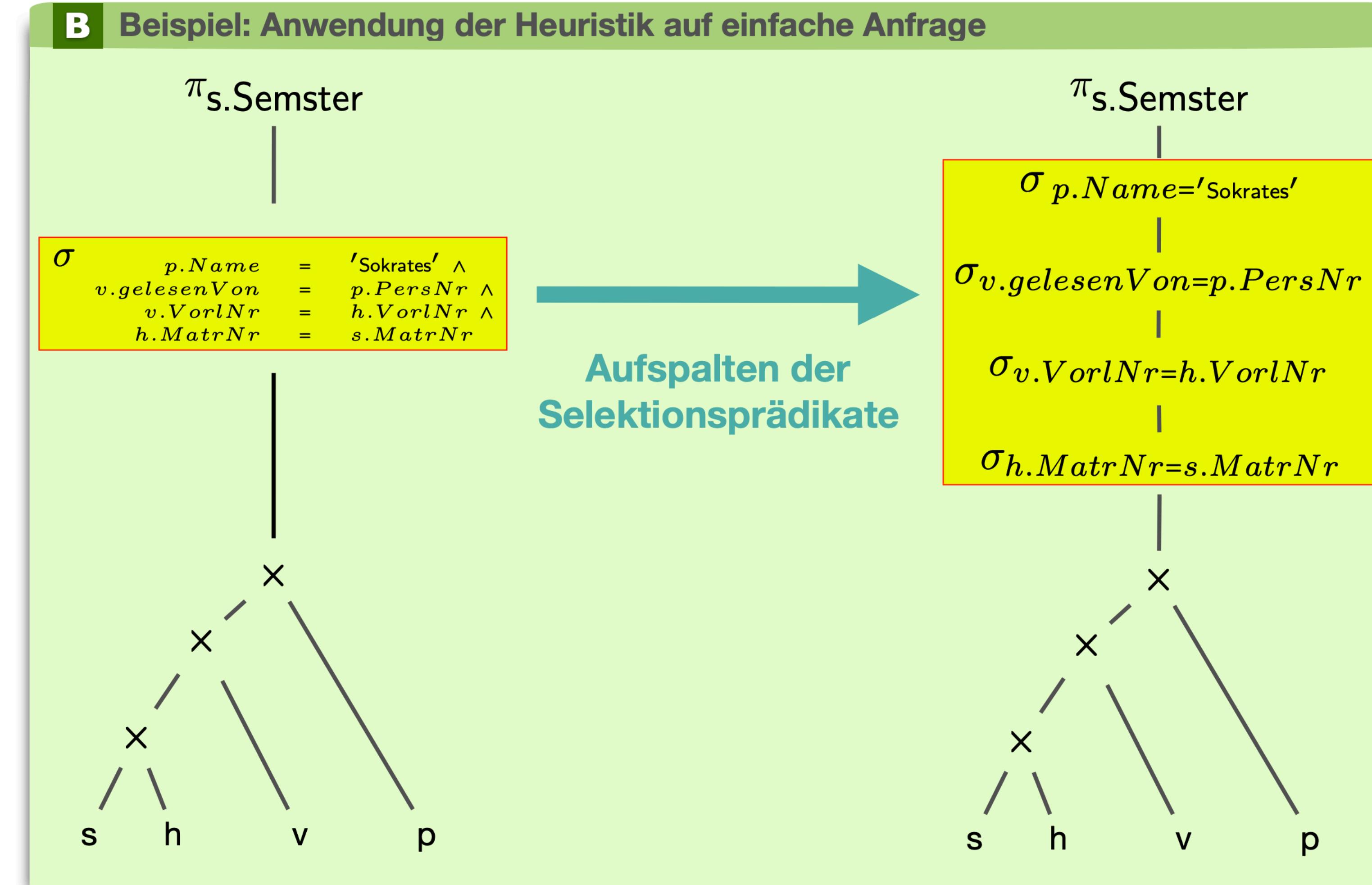
$$\sigma_{\begin{array}{ll} p.Name = 'Sokrates' \wedge \\ v.gelesenVon = p.PersNr \wedge \\ v.VorlNr = h.VorlNr \wedge \\ h.MatrNr = s.MatrNr \end{array}}$$

$\pi_{s.Semster}$

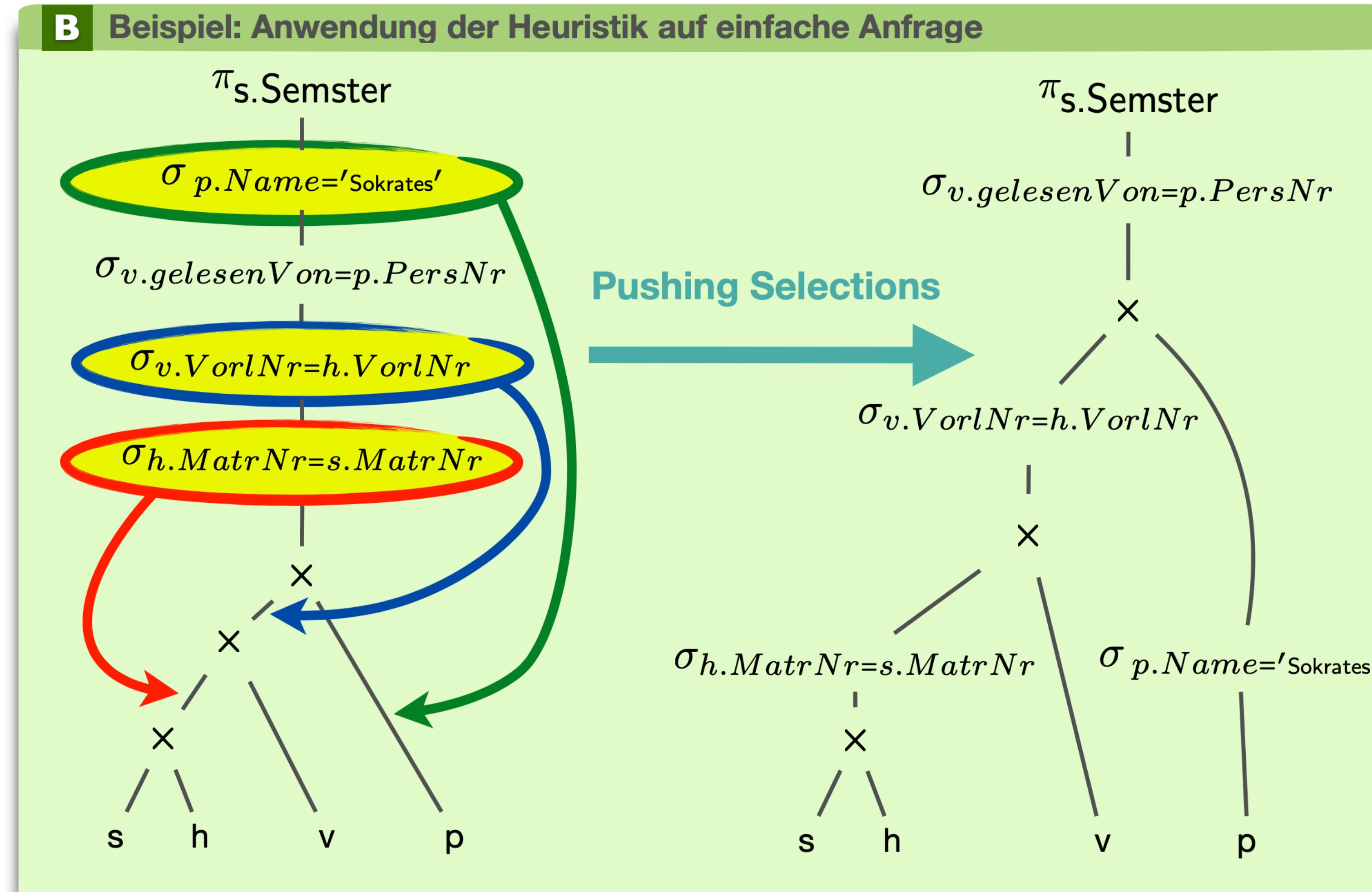
$750 * 10^{15}$



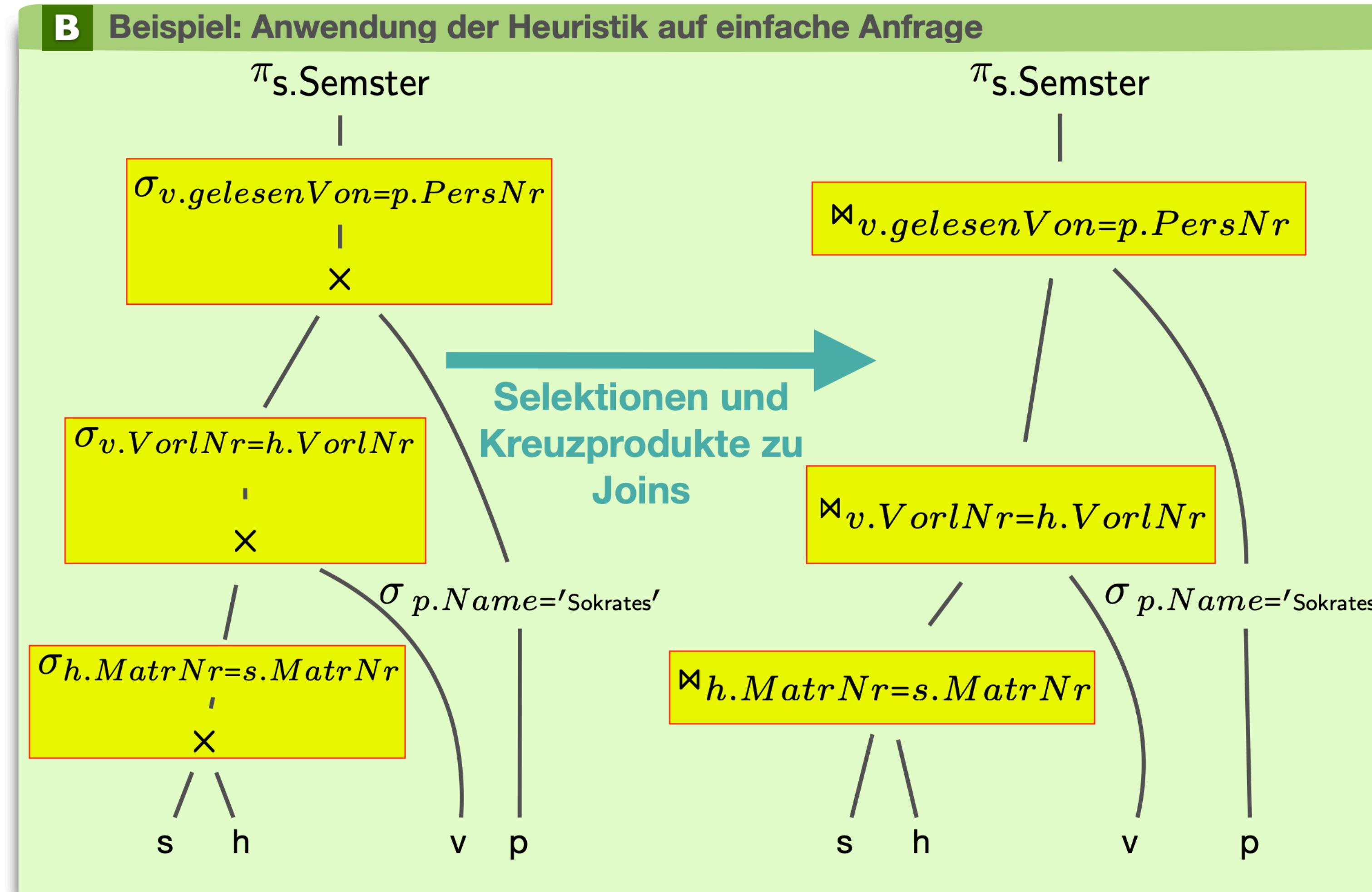
Beispiel: Anfrageoptimierung



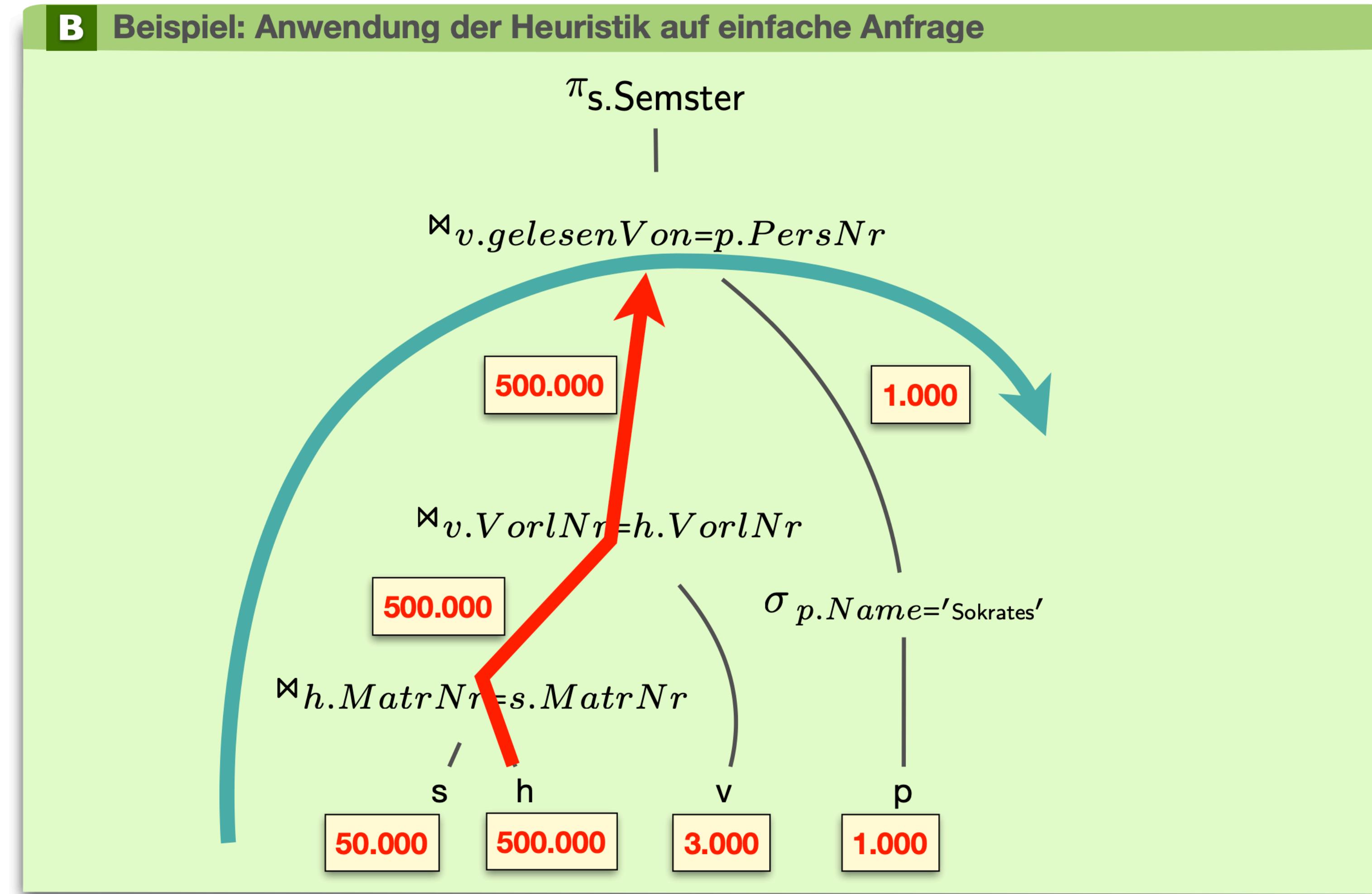
Beispiel: Anfrageoptimierung



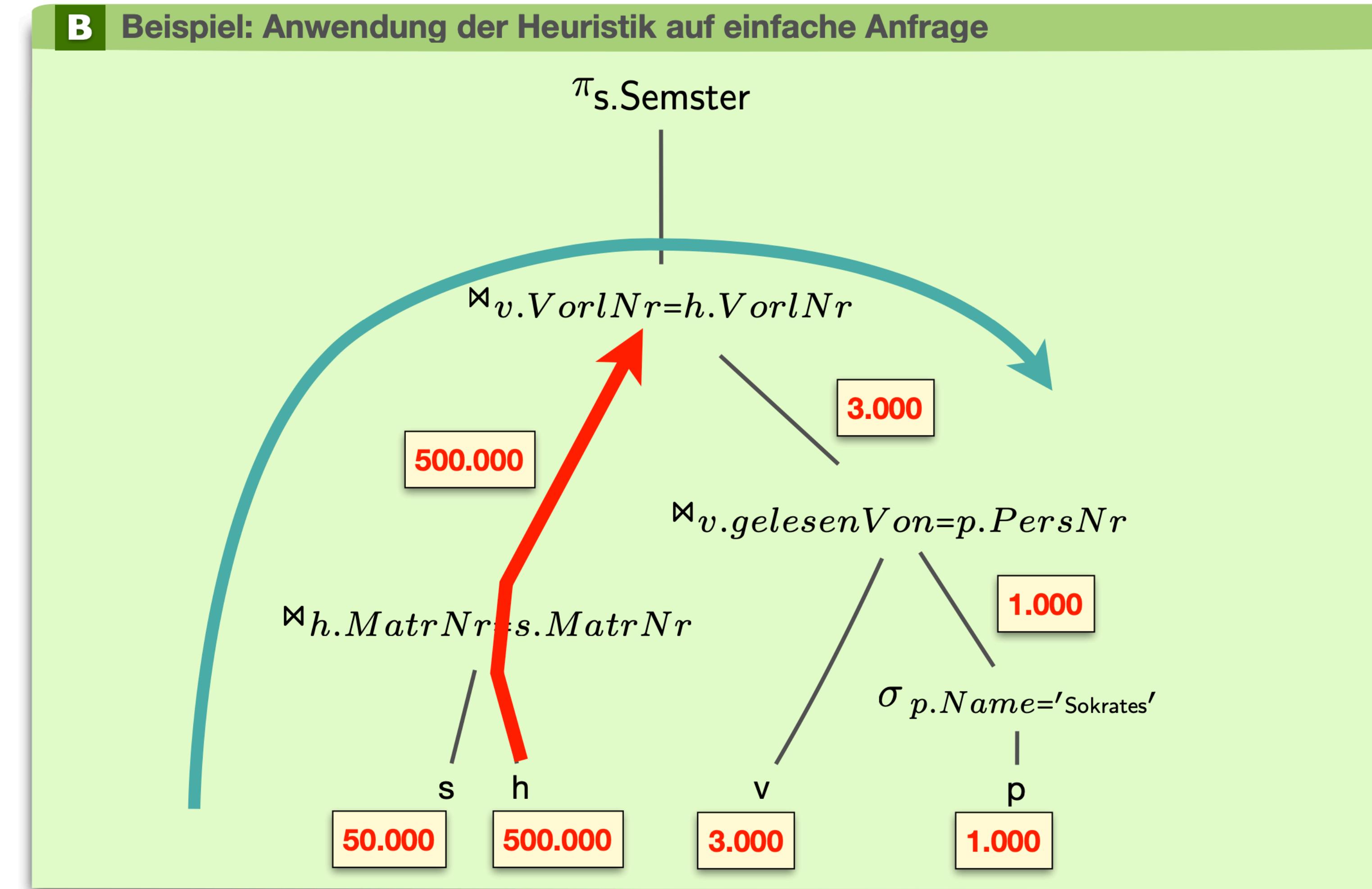
Beispiel: Anfrageoptimierung



Beispiel: Anfrageoptimierung

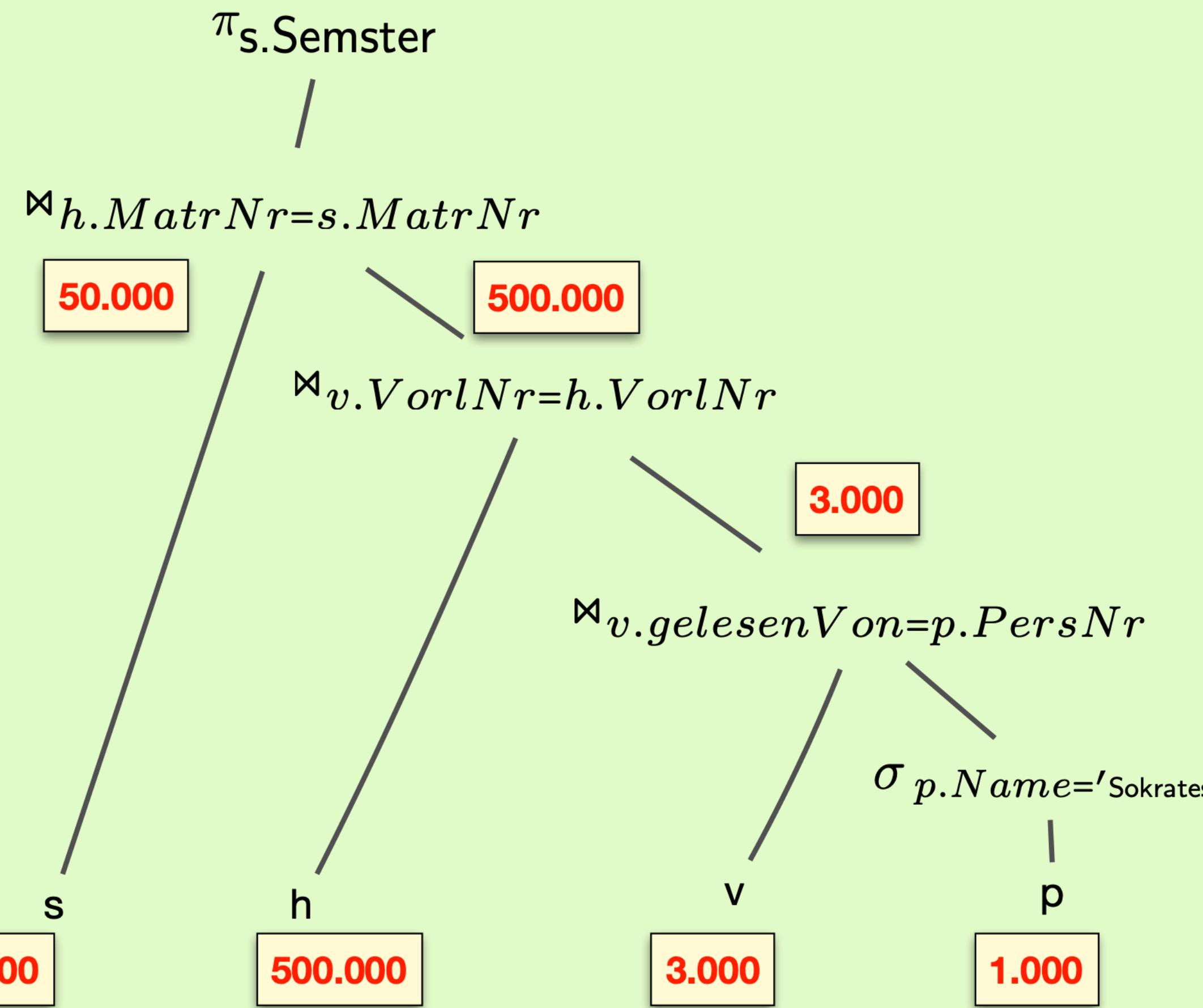


Beispiel: Anfrageoptimierung



Beispiel: Anfrageoptimierung

B Beispiel: Anwendung der Heuristik auf einfache Anfrage



Anfrageoptimierung

Anmerkungen

- Optimierung ist Anwendungsfall-spezifisch.
- Optimierung nicht per Gefühl.
- Tools nutzen.
- Indizes nutzen.
- Normalformen dürfen in Frage gestellt werden.