

Informationsintegration
Verteilte Anfragebearbeitung

Felix Naumann

Überblick

2

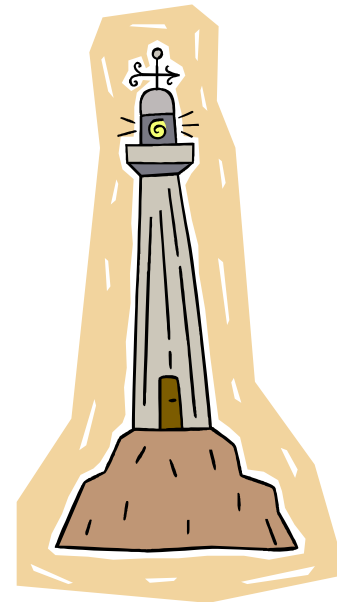
Anfragebearbeitung im Überblick

Techniken der Verteilten Anfragebearbeitung

- Row Blocking
- Multicasts
- Multithreading
- Partitionierung

Joinbearbeitung

- Semi-Join
- Reduzierung
- Semi-Join mit Filter



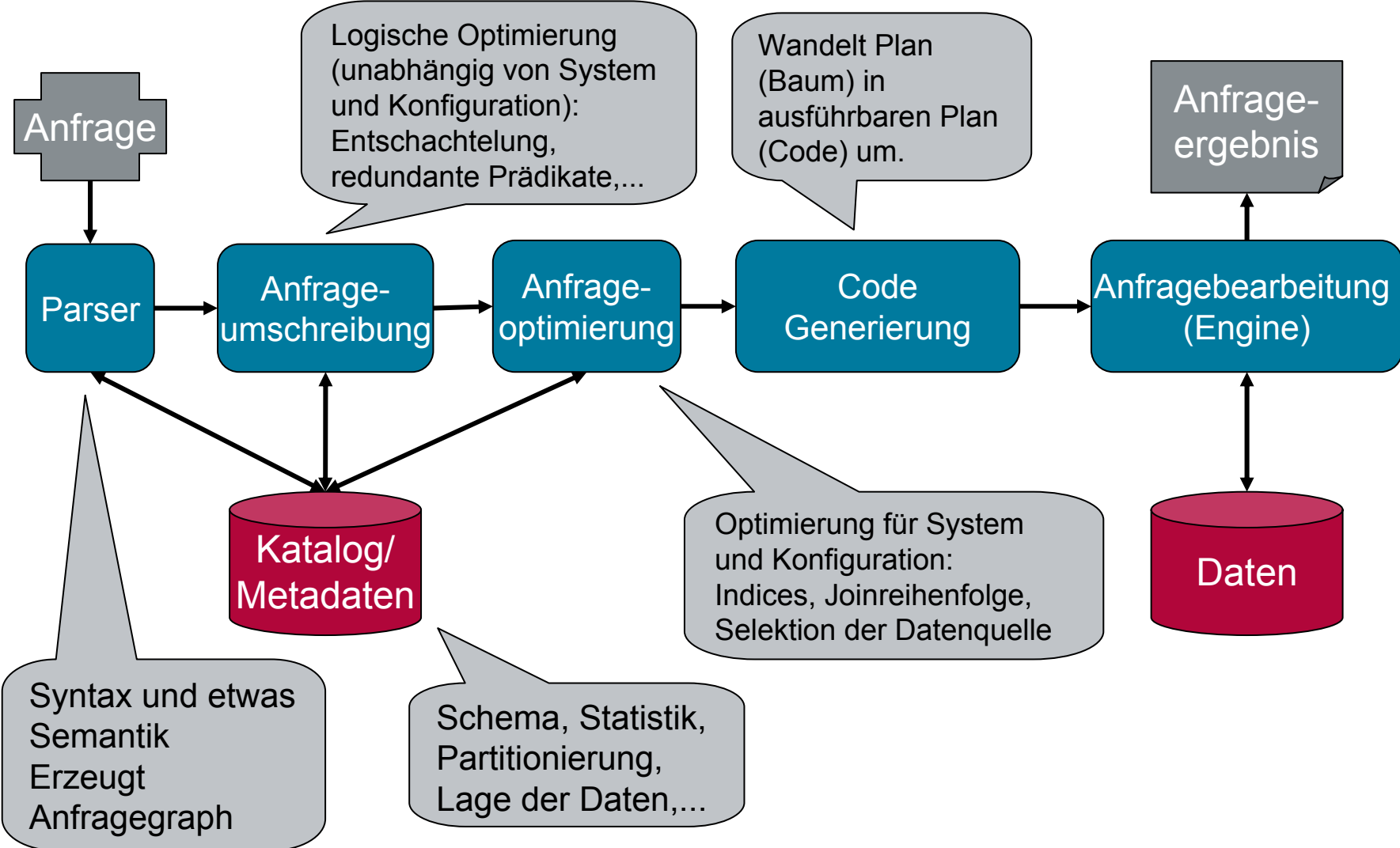
Parallelität vs. Verteilung

3

- Parallele DBMS
 - Shared memory
 - Shared disk
 - Shared nothing
 - Fokus auf Transaktionen und Anfragebearbeitung
- Verteilte DBMS
 - Shared nothing
 - Fokus auf Heterogenität und Anfragebearbeitung
 - Unser Fokus!

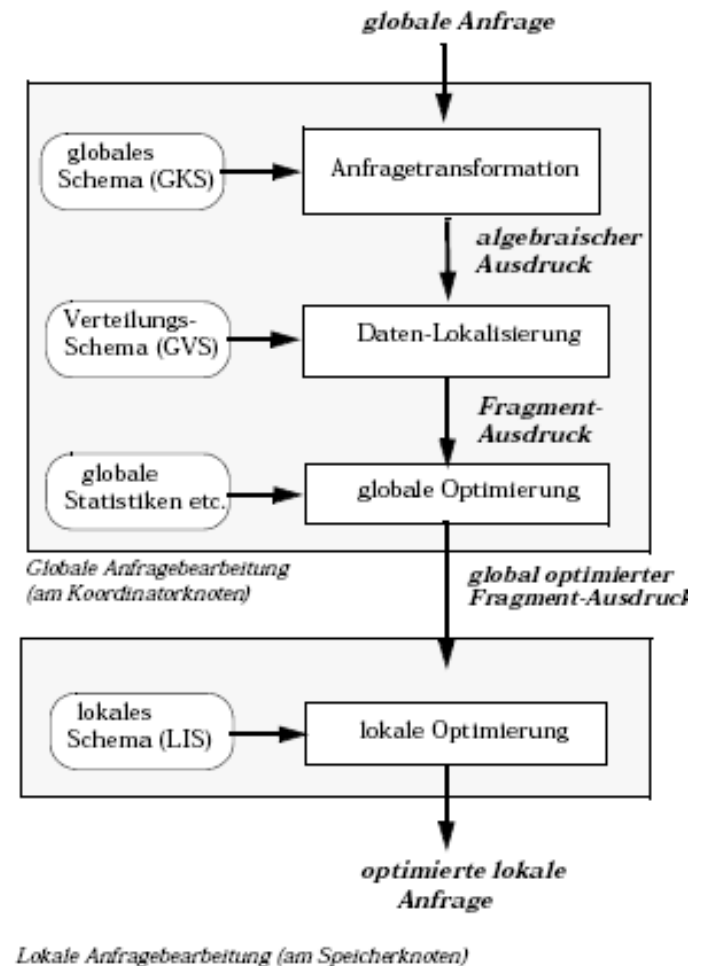
Architektur zur zentralen Anfragebearbeitung

4



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

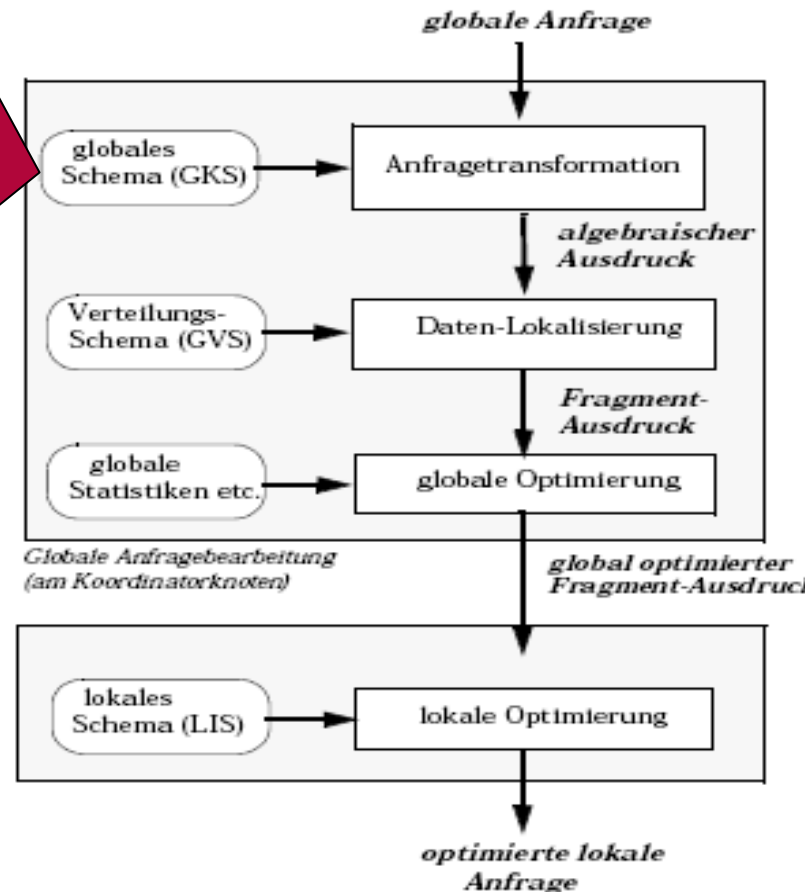
Phasen der verteilten Anfragebearbeitung



Schritt 1: Anfragetransformation

6

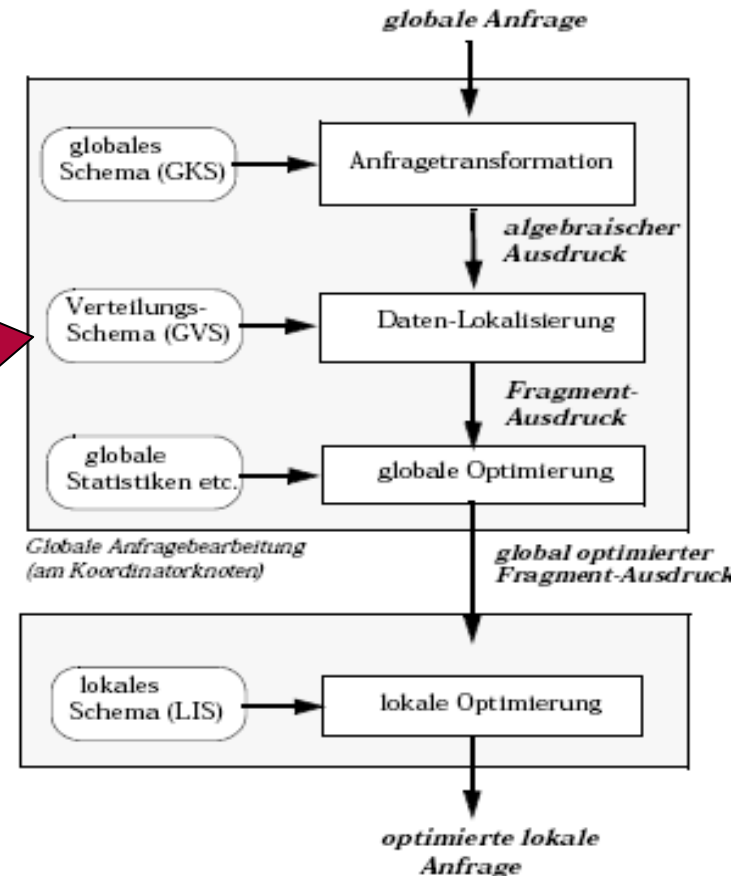
1. Parsen der Anfrage (Syntax)
2. Namensauflösung
3. Überprüfen der Elemente (Semantik)
4. Normalisierung
 - Konjunktive Normalform in WHERE Klausel
5. Algebraische Vereinfachung
 - Eliminierung redundanter Teilausdrücke
6. Transformation zum Operatorbaum



Schritt 2: Daten-Lokalisierung

7

1. Globale Relationen werden abgebildet auf lokale Relationen
 - GaV / LaV
2. Algebraische Vereinfachungen



Schritt 3: Globale Optimierung

8

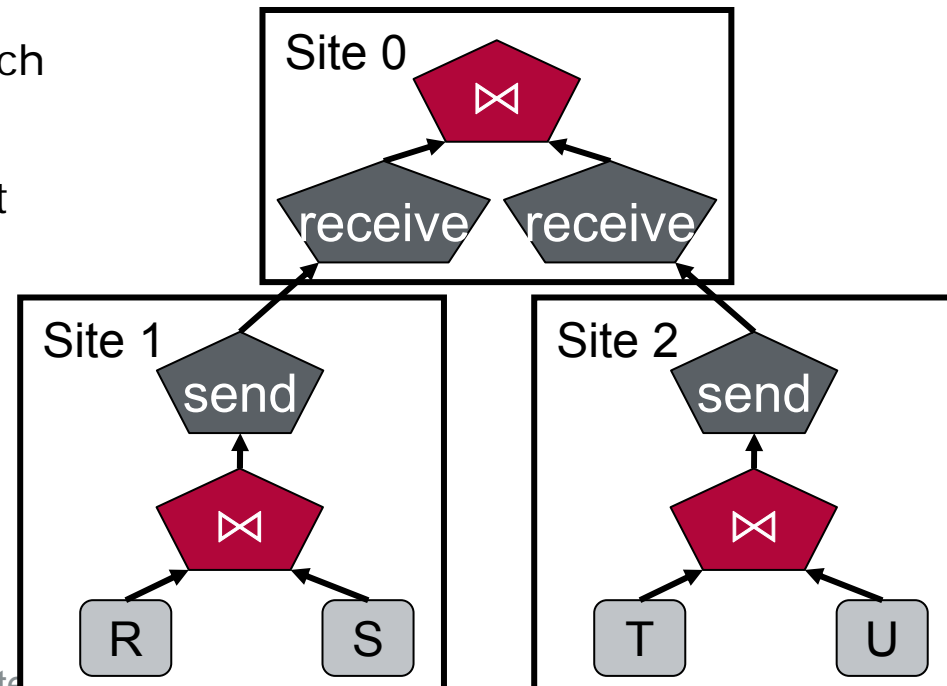
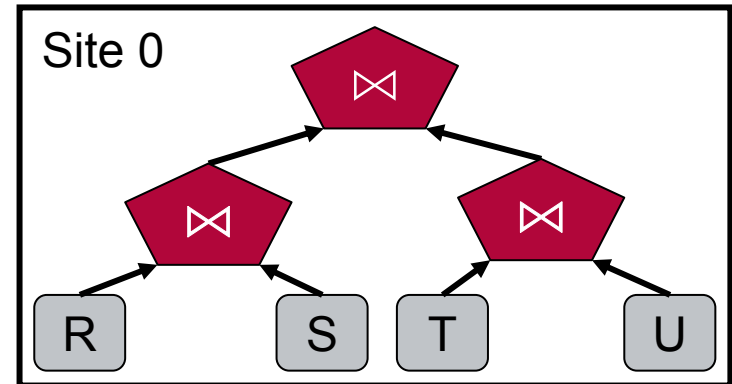
- Bestimmung eines Ausführungsplanes mit minimalen globalen Kosten
 - Bestimmung der Ausführungsknoten
 - Festlegung der Ausführungsreihenfolge (sequentiell, parallel)
 - Alternative Strategien zur Join-Berechnung bewerten (z.B. mit Semi-Join)
 - Aber: Trennung zwischen globaler und lokaler Optimierung kann zur Auswahl suboptimaler Pläne führen

- Kostenmodell
 - $| \text{Instruktionen} | + | \text{Diskzugriffe} | + | \text{Nachrichten} | + | \text{übertragene Byte} |$
 - $\text{Kosten} = (T_{\text{CPU}} * \# \text{insts}) + (T_{\text{I/O}} * \# \text{I/Os}) + (T_{\text{MSG}} * \# \text{msgs}) + (T_{\text{TR}} * \# \text{byte})$

Neue Ziele der Anfragebearbeitung

9

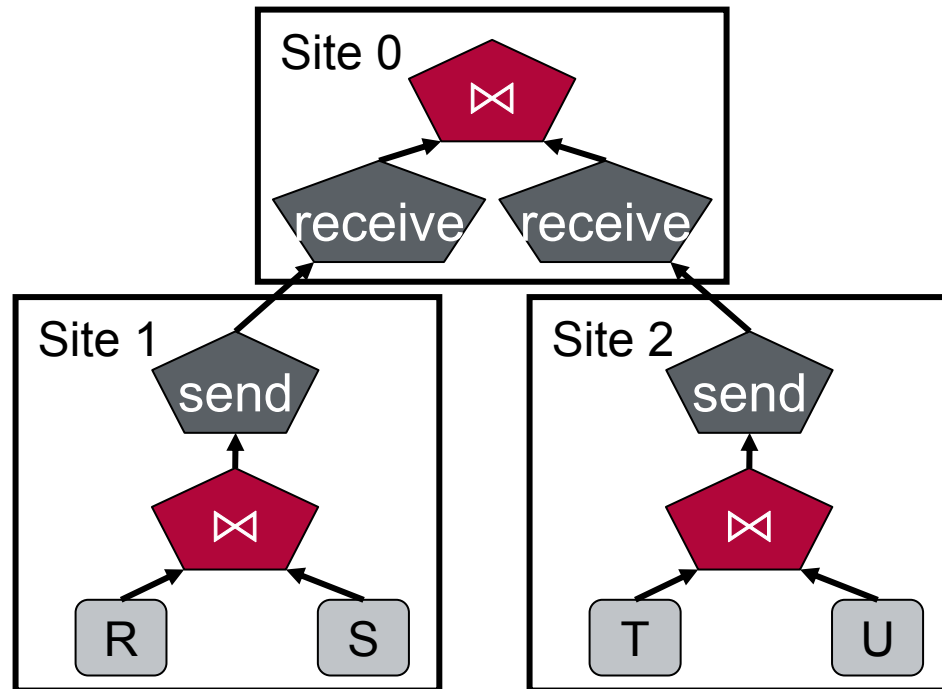
- In zentralisierten (mehr-Benutzer) DBMS
 - Durchsatz (*throughput*) = Anzahl der verarbeiteten Tupel
 - Minimierung des Ressourcenverbrauchs
- In verteilten DBMS
 - Gesamt-Ressourcenverbrauch erhöhen um schnellere Antworten zu erhalten
 - Minimierung der Antwortzeit



Dimensionen der Anfragebearbeitung

10

- Festlegung des Ausführungsknotens
- Festlegung der Auswertungsstrategie
 - *Ship whole*
 - Vollständige Relationen
 - Wenig Nachrichten
 - Viele Byte
 - *Fetch rows as needed*
 - Bindings
 - Viele Nachrichten
 - Nur relevante Byte
 - Semi Join
 - *Fetch columns as needed*
 - Semi-Join

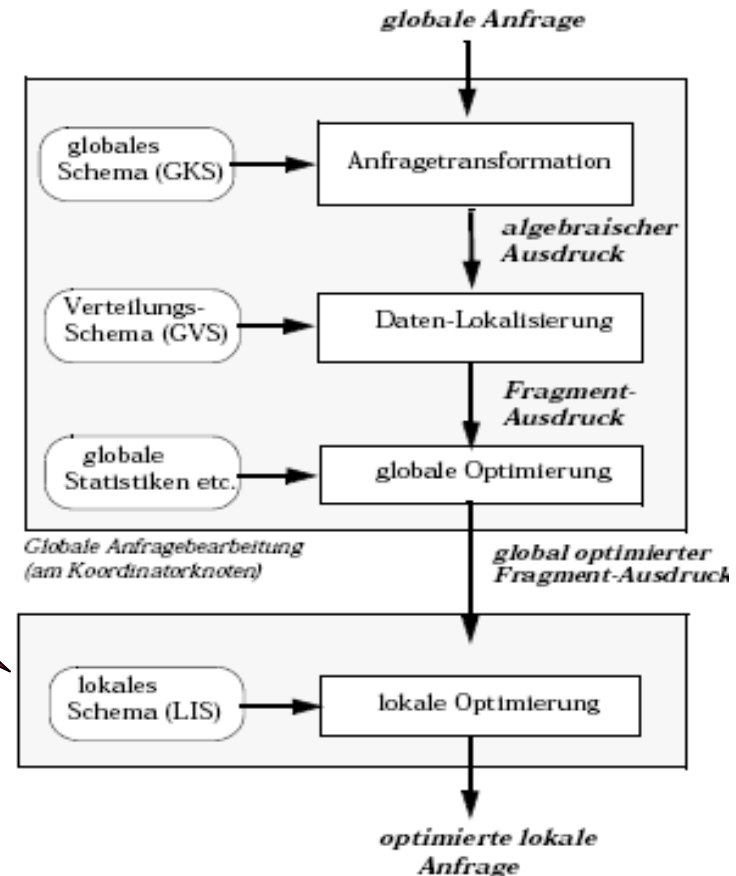


Schritt 4: Lokale Optimierung

11

Entsprechend des jeweiligen Systems

- Lokale Katalogdaten
- Lokale Parameter
 - T_{CPU} und $T_{I/O}$
- Etc.



Überblick

12

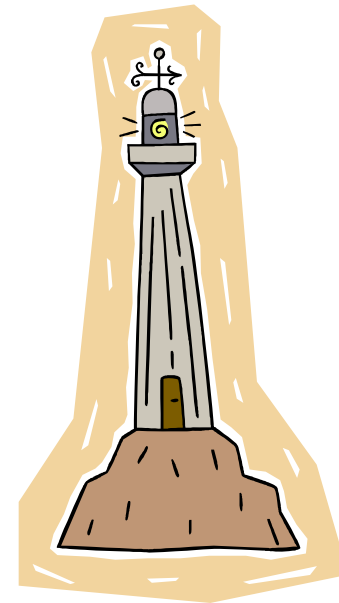
Anfragebearbeitung im Überblick

Techniken der Verteilten Anfragebearbeitung

- Row Blocking
- Multicasts
- Multithreading
- Partitionierung

Joinbearbeitung

- Semi-Join
- Reduzierung
- Semi-Join mit Filter



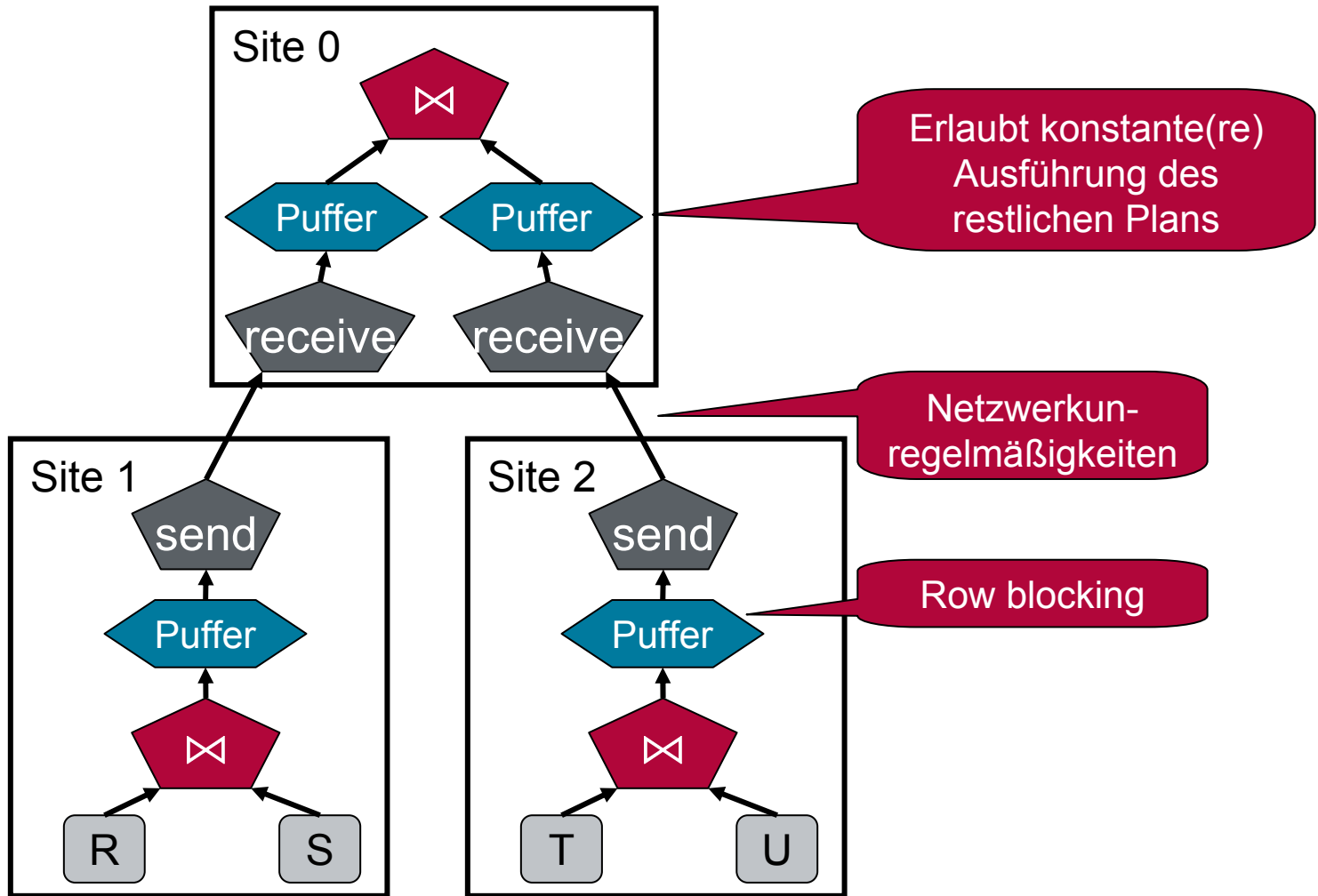
Row Blocking

13

- Anfragebearbeitung in verteilten DBMS mittels *send* und *receive* Operatoren
- Naiv: Für jedes Tupel eine *send* und eine *receive* Operation
- Besser: *Row blocking*
 - Tupel werden gesammelt und en block versendet.
 - Blockgröße abhängig von datagram-Größe im Netzwerk (z.B. 64kB)
 - Queues puffern unregelmäßige Pipelines
 - Wichtig in Netzwerken; Stillstand wird vermieden.

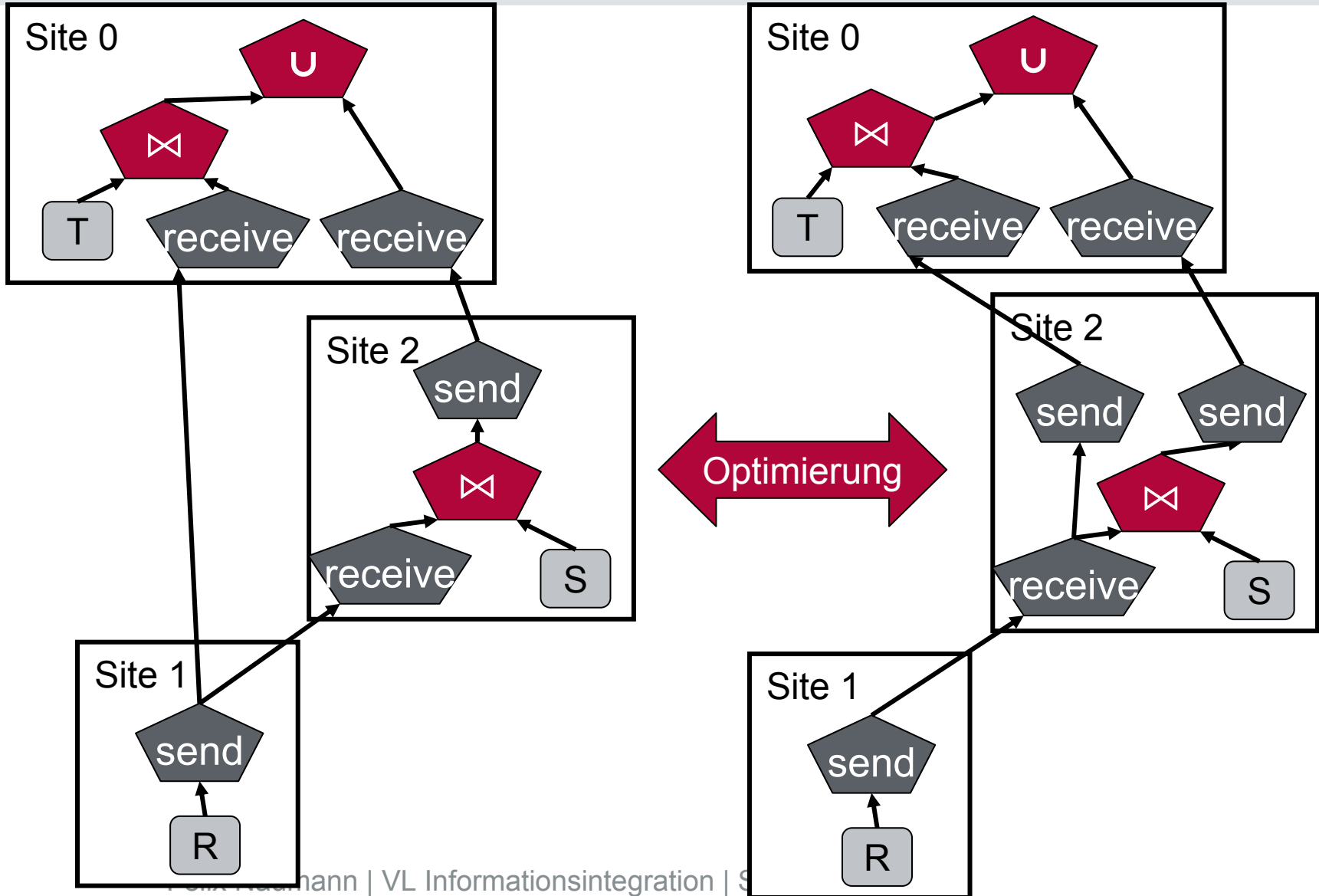
Row Blocking

14



Multicasts

15



Multicasts

16

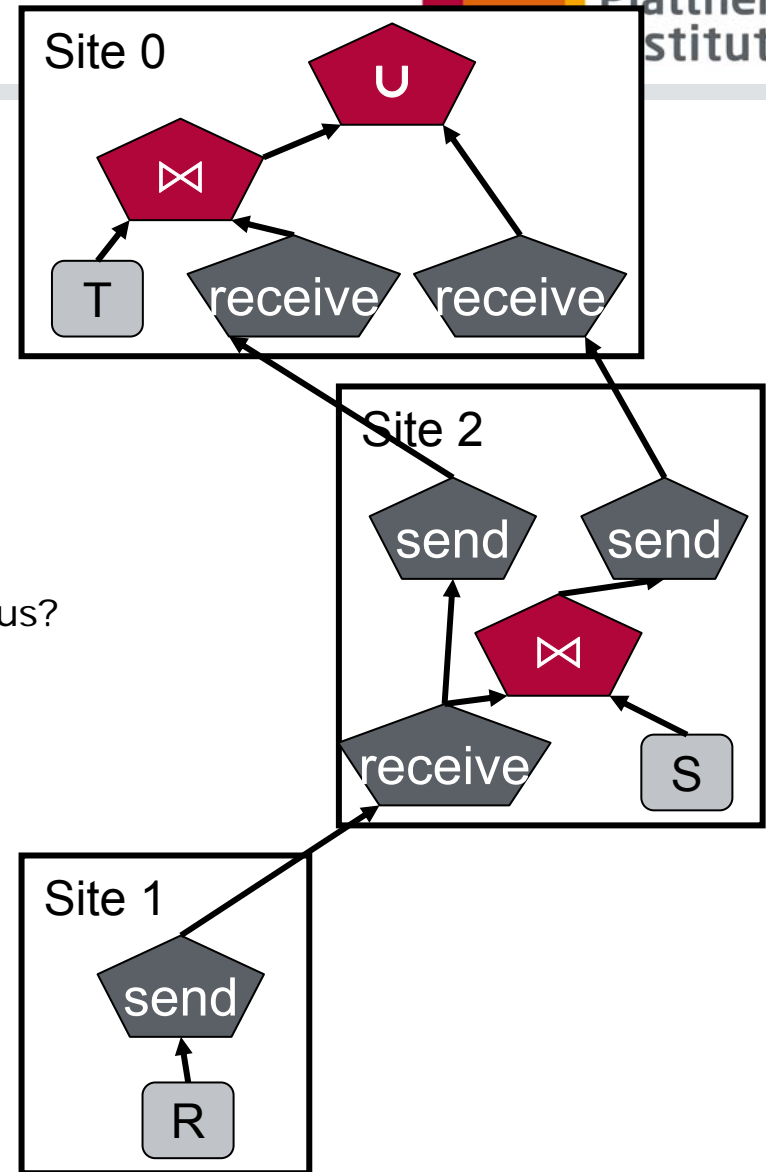
Einsparungen

- Netzverkehr
- CPU (packing und unpacking)

Optimierung

- Viele Alternativen
 - Physisch:
 - Welche Daten schicke ich wohin?
 - Welchen Operator führe ich wo aus?
 - Logisch:
 - Outer joins statt inner joins
 - Späte/Frühe Projektion
 - Usw.

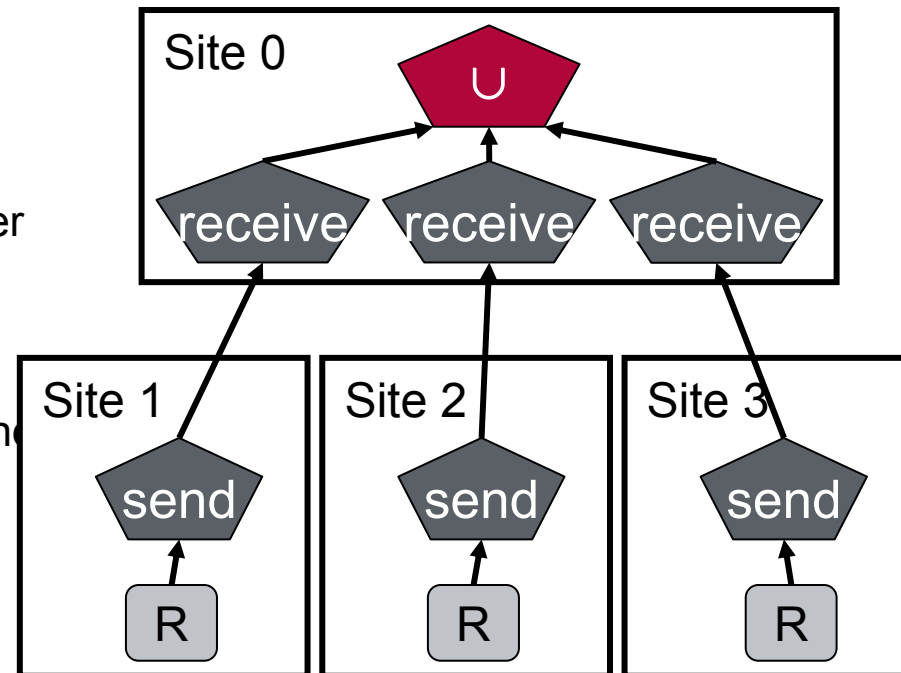
- Kostenmodell
- Dynamische Entscheidungen
 - noch während der Ausführung!



Multi-Threading

17

- Single-threaded UNION:
 - Sukzessive ein Datagramm von einer Site
 - „Round-robin“-Verfahren
- Multi-threaded UNION:
 - Parallele Verarbeitung der send und receive Operationen
- Vorteile
 - Netzwerkverkehr parallel
 - Send Operationen parallel
- Nachteile
 - Zusätzliche Kosten durch Synchronisation (shared memory)
 - Zusätzliche Kosten durch Konkurrenz um Ressourcen
 - Optimierer muss entscheiden welche Operatoren in wie vielen Threads ausgeführt wird.

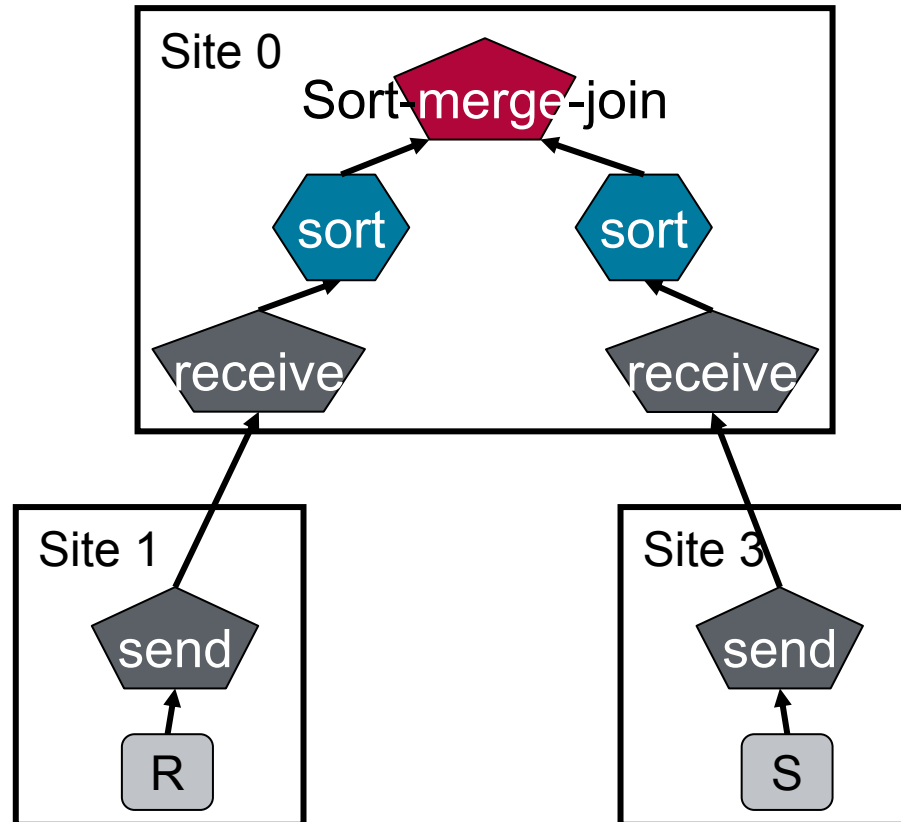


Multi-Threading

18

Multi-threading nicht immer vorteilhaft

- z.B. Speicherplatz



Partitionierung

19

- Grundlagen von DBMS
- Vertikal
 - Verschiedene Projektionen einer Relation
 - Jedes Attribut in mindestens einer Partition
 - Erhalt eines Schlüssels!
 - Auch: Normalisierung
- Horizontal
 - Verschiedene (disjunkte) Selektionen einer Relation
 - Beispiel
 - Partition A: `SELECT * FROM R WHERE area='Nord'`
 - Partition B: `SELECT * FROM R WHERE area='Süd' (bzw. ≠ 'Nord')`
 - Jedes Tupel in mindestens einer Partition
- Optimierung
 - Wie partitionieren?
 - Wie die Partitionen verteilen?
 - Abhängig von Anwendungen und *query workload*

Joins über horizontale Partitionen

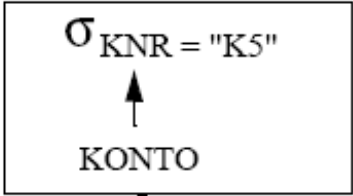
20

- Sei R horizontal partitioniert
 - $R = R_1 \cup R_2$
- Alternativen
 - $R \bowtie S = (R_1 \cup R_2) \bowtie S$
 - $R \bowtie S = (R_1 \bowtie S) \cup (R_2 \bowtie S)$
- Komplikationen
 - R noch weiter partitioniert
 - S ebenfalls partitioniert
 - Unterschiedliche Kosten
 - Wer führt was wo aus?

Selektion über horizontale Partitionierung

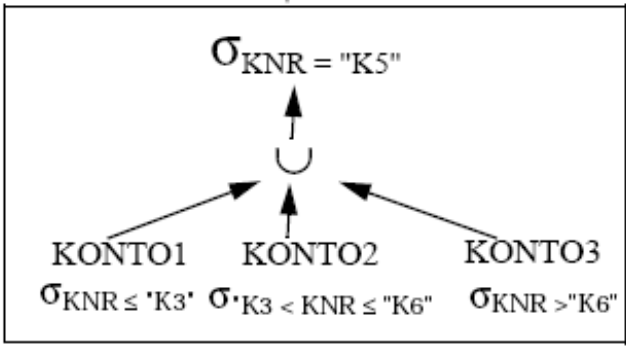
21

KONTO sei horizontal in drei Fragmente zerlegt (Fragmentierungsattribut KNR)
Bestimmung aller Konten von Kunde K5



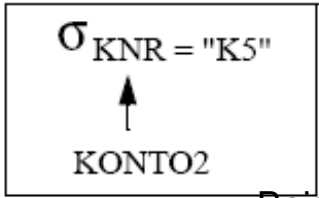
Rekonstruktion

Initialer Fragment-Ausdruck



Algebraische Optimierung

Reduzierter Fragment-Ausdruck

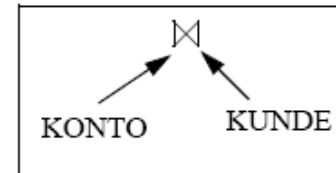


Beispiel: Mitschang, VL „Verteilte DBMS“

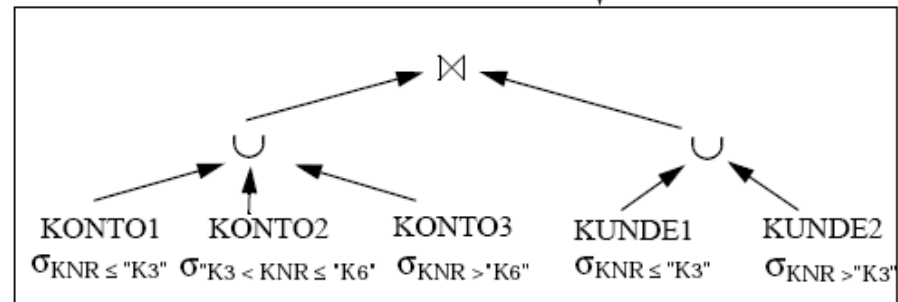
Join über horizontale Partitionierung

22

KONTO und KUNDE seien horizontal über KNR zerlegt (unterschiedl. Bereiche)

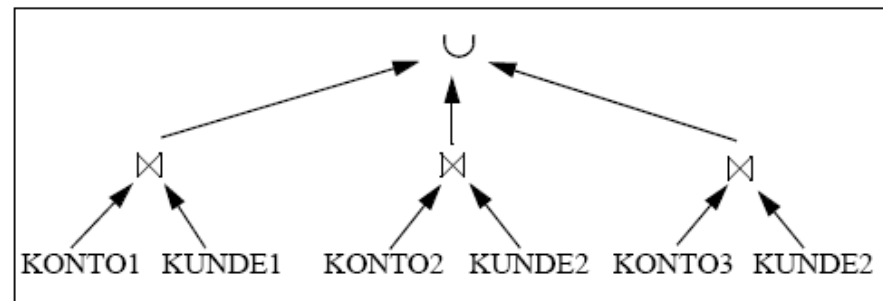


Rekonstruktion



Initialer Fragment-Ausdruck

Algebraische Optimierung



Reduzierter Fragment-Ausdruck

Beispiel: Mitschang, VL „Verteilte DBMS“

Joins über horizontale Partitionen

23

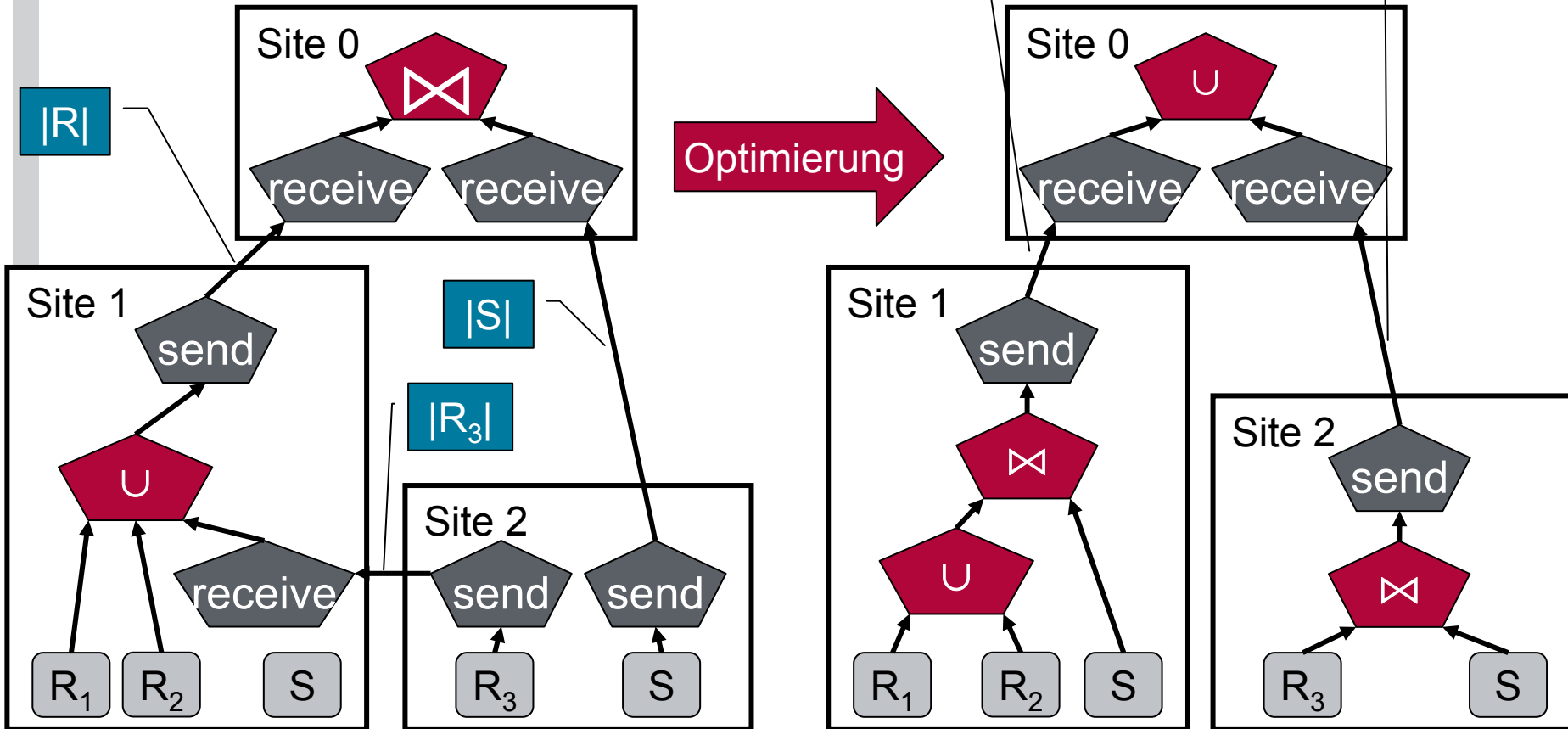
Frage: Wieviel wird jeweils übertragen?

$$|(R_1 \cup R_2) \bowtie S|$$

$$|R_3 \bowtie S|$$

$$|R|$$

Optimierung



Joins über horizontale Partitionen

24

- Optimierung
 - Lage der Partitionen
 - Kostenvergleiche
 - Leere Teilergebnisse ($R_i \bowtie S_j = \emptyset$)
 - Vorhersagen & Vermeiden
 - Häufig: Zwei Relationen nach gleichem Prädikat partitioniert
 - „Abteilungen“ nach Standort partitioniert (Nord, Süd)
 - „Mitarbeiter“ nach Abteilung (also auch nach Standort) partitioniert
 - Dann: $R \bowtie S$
 - $= (R_1 \bowtie S_1) \sqcup (R_1 \bowtie S_2) \sqcup (R_2 \bowtie S_1) \sqcup (R_2 \bowtie S_2)$
 - $= (R_1 \bowtie S_1) \sqcup (R_2 \bowtie S_2)$
 - D. h. Joins ($R_i \bowtie S_j$) nur ausführen falls $i = j$

Überblick

25

Anfragebearbeitung im Überblick

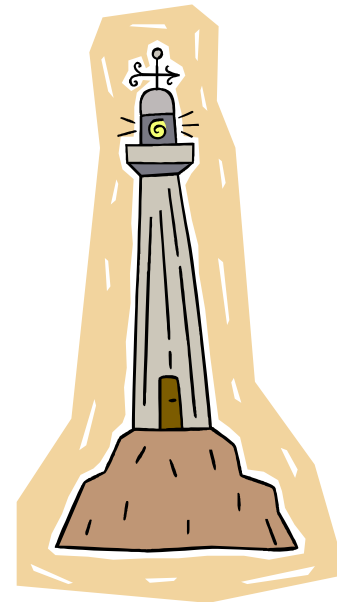
Techniken der Verteilten Anfragebearbeitung

- Row Blocking
- Multicasts
- Multithreading
- Partitionierung



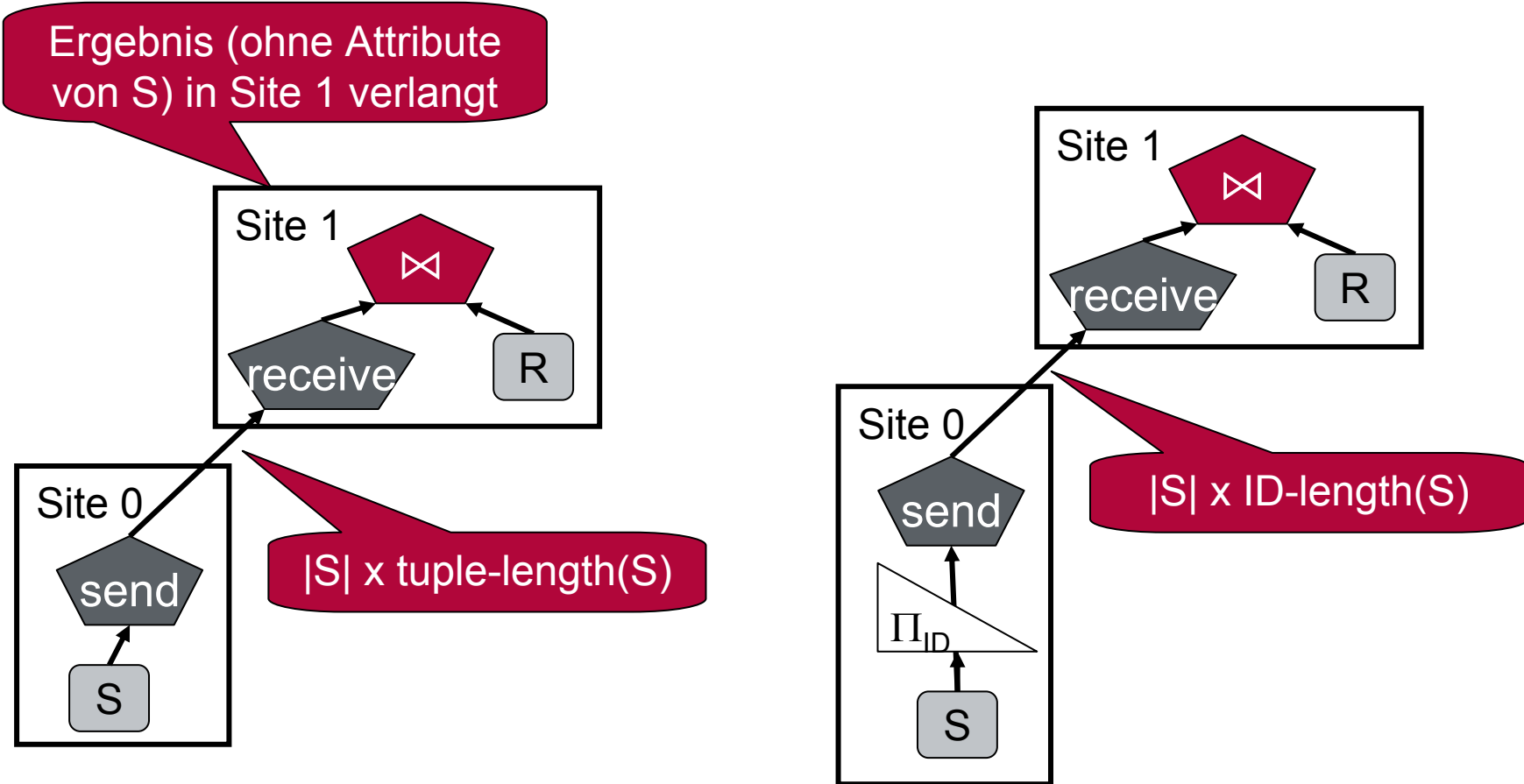
Joinbearbeitung

- Semi-Join
- Reduzierung
- Semi-Join mit Filter



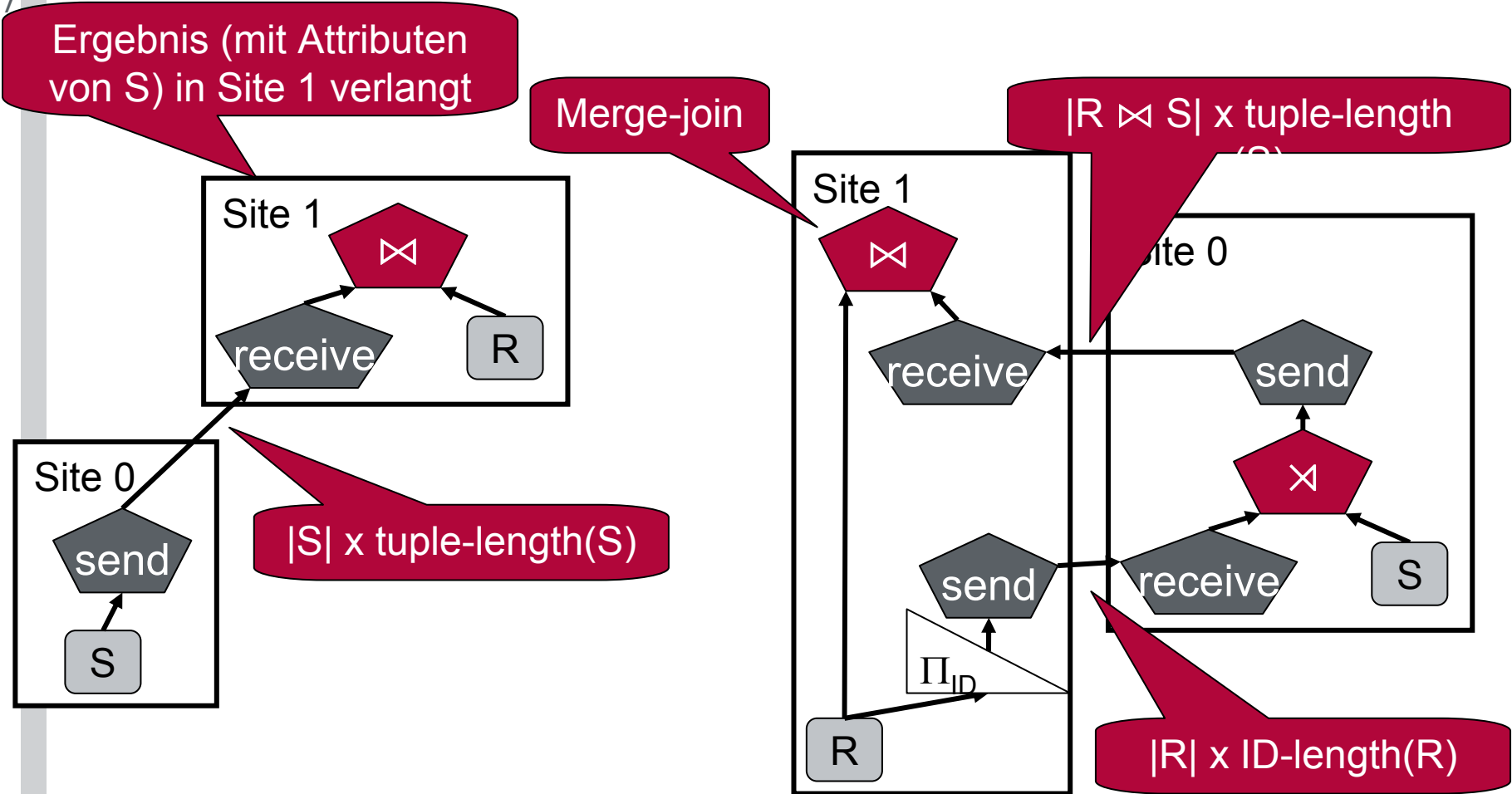
Semi-Join

26



Semi-Join

27



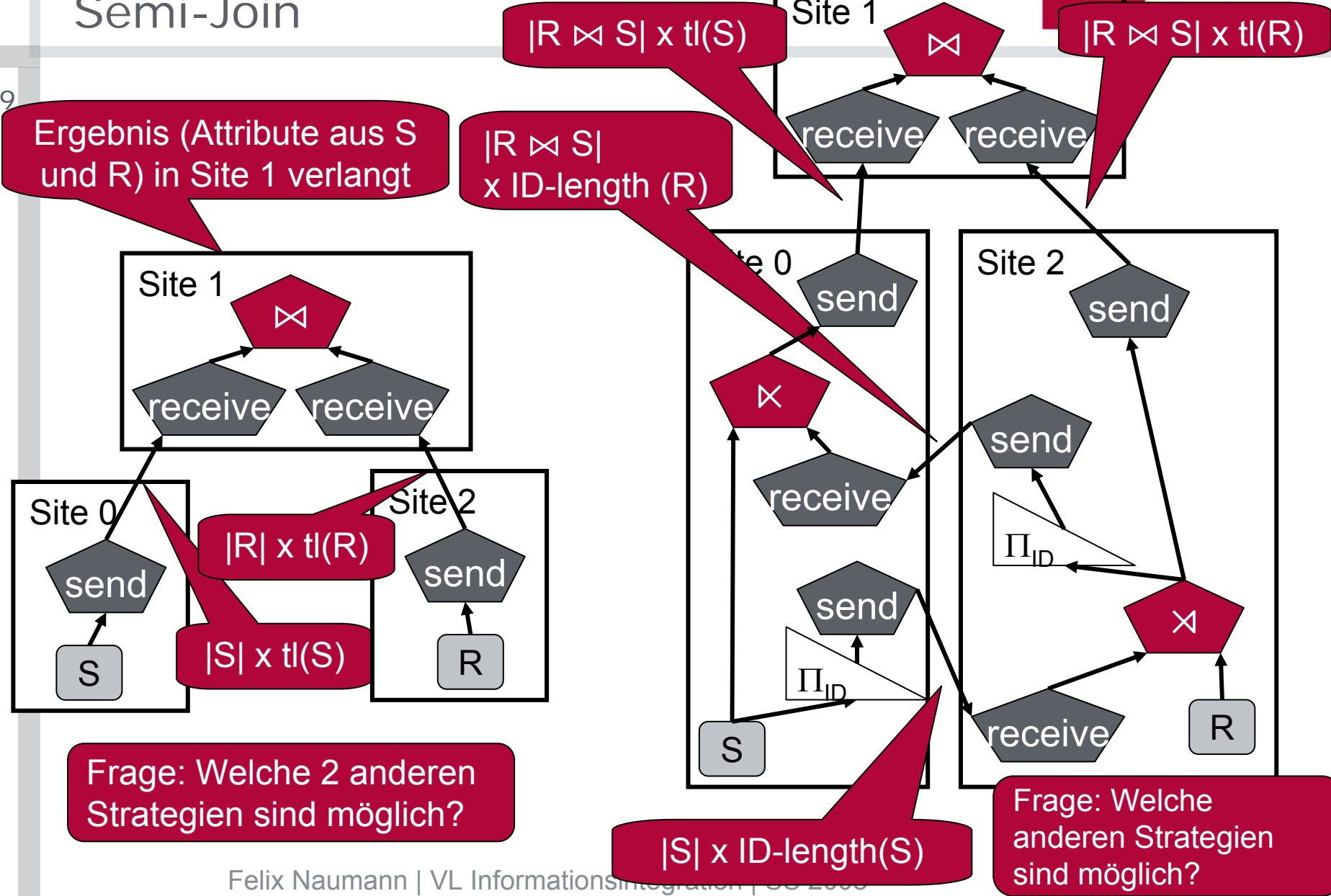
Semi-Join

28

- Formal
 - $R(A), S(B)$
 - $R \bowtie S := \Pi_A(R \bowtie_F S)$
 - $= \Pi_A(R) \bowtie_F \Pi_{A \cap B}(S)$
 - $= R \bowtie_F \Pi_{A \cap B}(S)$
 - i.d.R. $= R \bowtie_F \Pi_F(S)$
- Nicht symmetrisch!
- Literatur:
 - [BC81]
 - in jedem DB Lehrbuch

Semi-Join

29



Optimierung mit Semi-Join

30

- Transformationsregeln für Joins
- $R \bowtie_F S =$
 - $(R \times_F S) \bowtie_F S$
 - R verkleinern, dann Join mit S
 - $R \bowtie_F (S \times_F R)$
 - S verkleinern, dann Join mit R
 - $(R \times_F S) \bowtie_F (S \times_F R)$
 - R und S verkleinern dann Join.
- Problem: Wann welche Variante einsetzen?

Frage: Welche beiden Varianten standen auf der vorigen Folie?

Optimierung mit Semi-Join

31

- Kostenmodell

- $|Instruktionen| + |Diskzugriffe| + |Nachrichten| + |übertragene Byte|$

- $Kosten = T_{CPU} * \#insts$
 $+ T_{I/O} * \#I/Os$
 $+ T_{MSG} * \#msgs$
 $+ T_{TR} * \#byte$

- Hier nur: $|übertragene Byte|$

- Kostenvergleich (Annahme: $|R| < |S|$)

- $K(R \bowtie_F S) = byte(R)$

- $K((R \times_F S) \bowtie_F S) = byte(\Pi_F(S)) + byte((R \times_F S))$

Überblick

32

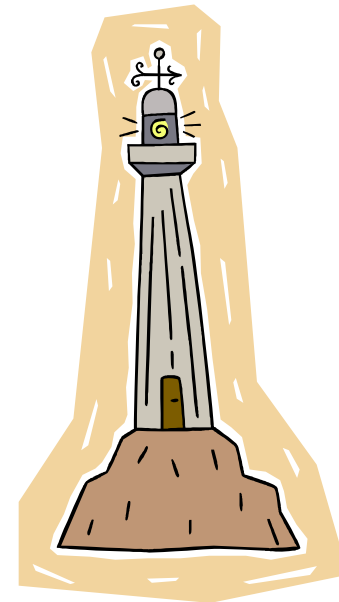
Anfragebearbeitung im Überblick

Techniken der Verteilten Anfragebearbeitung

- Row Blocking
- Multicasts
- Multithreading
- Partitionierung

Joinbearbeitung

- Semi-Join
- Reduzierung
- Semi-Join mit Filter



Optimierung mit Semi-Join

33

Komplexeres Beispiel

- $R \bowtie_F S \bowtie_G T =$
- $(R \bowtie_F S) \bowtie_F (S \bowtie_G T) \bowtie_G T =$
- $(R \bowtie_F (S \bowtie_G T)) \bowtie_F (S \bowtie_G T) \bowtie_G T$

Allgemein: Man suche für jede Relation das beste *Semi-Join Programm*, den vollständigen Reduzierer („*full reducer*“).

- Eine fully reduced Relation enthält keine Tupel, die nicht zur Anfragebearbeitung benötigt werden.

Für jede Relation in einer Anfrage existieren exponentiell viele Semi-Join Programme.

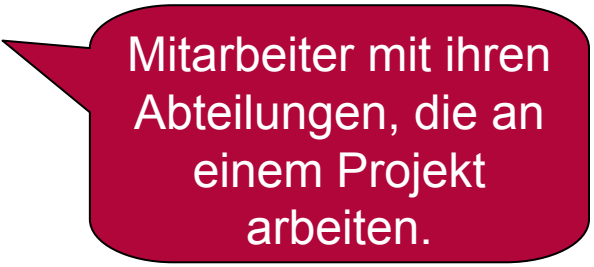
- Zyklische Anfragen: i.A. existiert kein full reducer.
- Baum-Anfragen: Finden des full reducer NP-schwer.
- Ketten-Anfragen: Finden des full reducer polynomial.

Optimierung mit Semi-Join

34

Ketten-Anfragen

```
SELECT EMP.name, DEPT.name  
FROM EMP, DEPT, PROJ  
WHERE EMP.d_ID = DEPT.ID  
AND EMP.p_ID = PROJ.ID
```



Mitarbeiter mit ihren
Abteilungen, die an
einem Projekt
arbeiten.



Ermittlung der full reducer für jede Relation in 2 Phasen

1. Vorwärts
2. Rückwärts

Fully Reduce

35

Ermittlung der full reducer für jede Relation in 2 Phasen

1. Vorwärts
2. Rückwärts

Allgemein für Kettenanfragen

- $R1 \bowtie_A R2 \bowtie_B \dots \bowtie_Y R(n-1) \bowtie_Z Rn$

- Vorwärts

- $R2' = R2 \bowtie R1$
- $R3' = R3 \bowtie R2' = R3 \bowtie (R2 \bowtie R1)$
- ...
- $Rn' = Rn \bowtie R(n-1)' = \dots$

- Rückwärts

- $R(n-1)'' = R(n-1)' \bowtie Rn'$
- $R(n-2)'' = R(n-2)' \bowtie R(n-1)''$
- ...
- $R1'' = R1 \bowtie R2''$

Full reducer für Rn

Full reducer für $R(n-1)$

Full reducer für $R1$

Fully Reduce

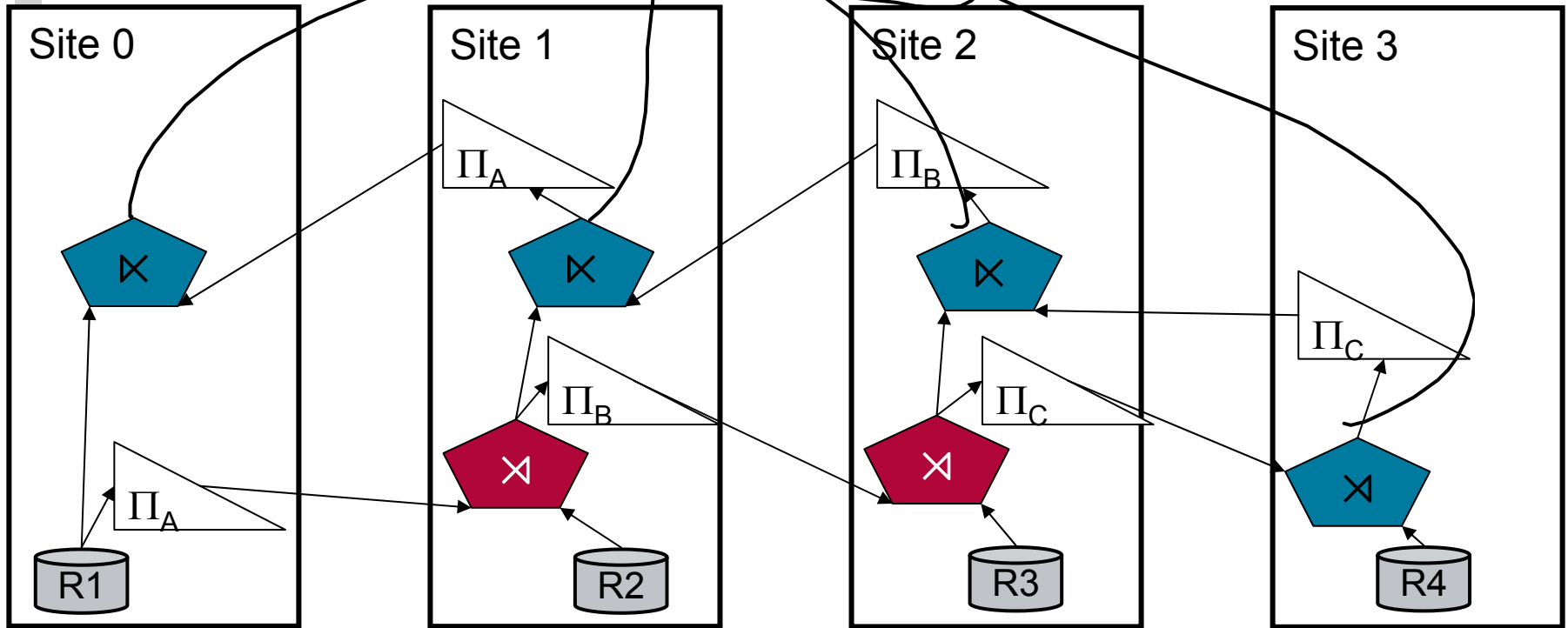
36

Beispiel für Kettenanfragen

- $R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$
- Vorwärts
 - $R2' = R2 \bowtie R1$
 - $R3' = R3 \bowtie R2' [= R3 \bowtie (R2 \bowtie R1)]$
 - $R4' = R4 \bowtie R3' [= R4 \bowtie (R3 \bowtie (R2 \bowtie R1))]$
- Rückwärts
 - $R3'' = R3' \bowtie R4'$
 - $R2'' = R2' \bowtie R3''$
 - $R1'' = R1 \bowtie R2''$

Fully Reduce – Beispiel

37



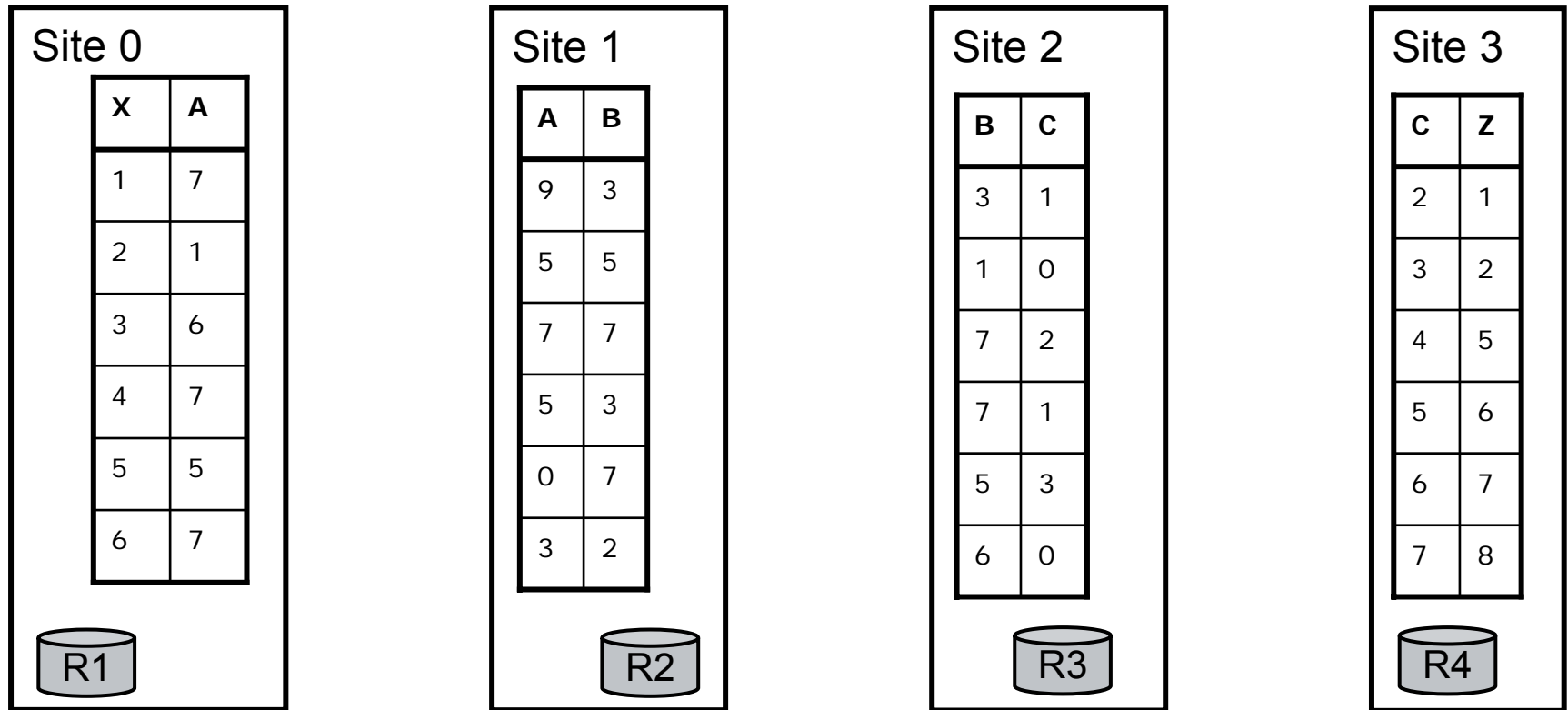
Vorwärts
 $R2' = R2 \times R1$
 $R3' = R3 \times R2'$
 $R4' = R4 \times R3'$

Rückwärts
 $R3'' = R3' \times R4'$
 $R2'' = R2' \times R3''$
 $R1'' = R1 \times R2''$



Fully Reduce – Beispiel

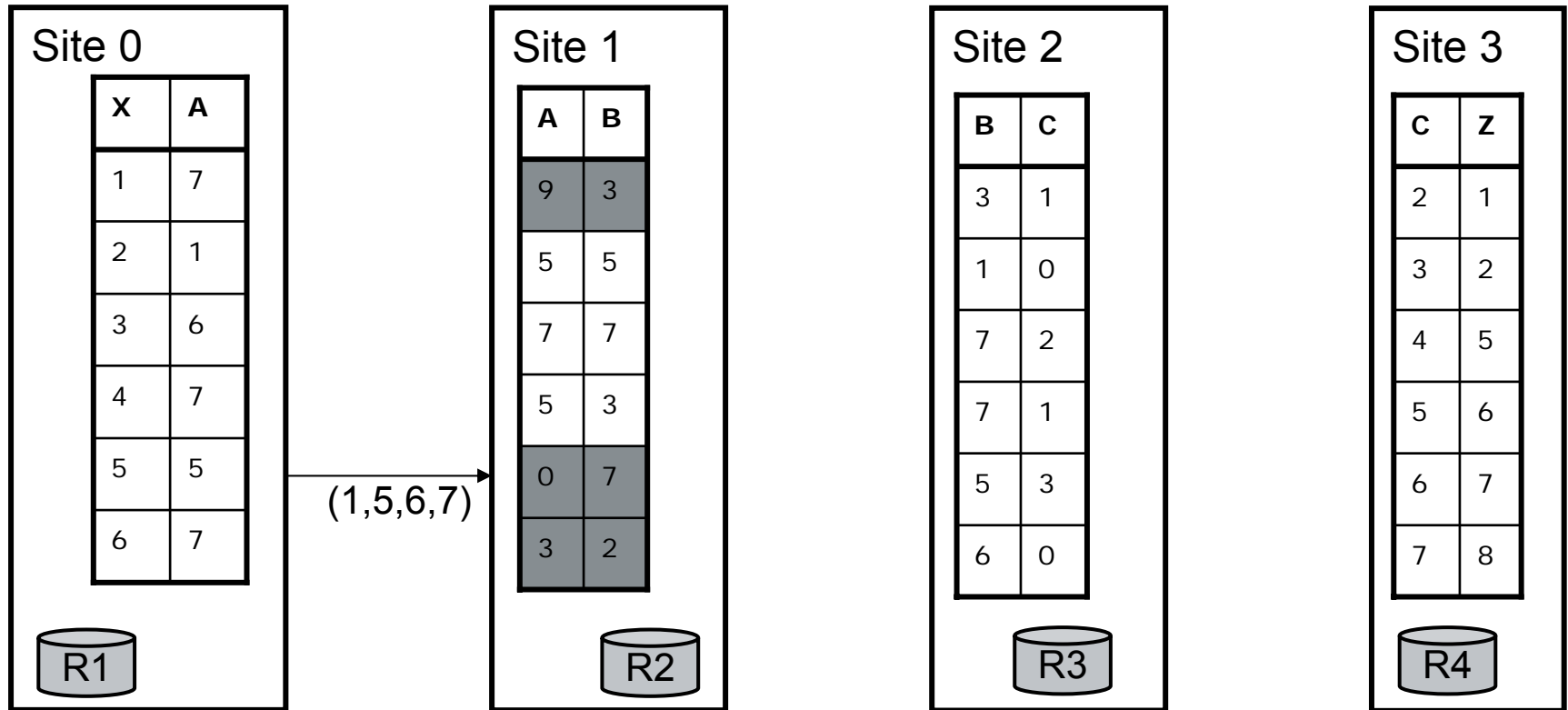
38



$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce – Beispiel

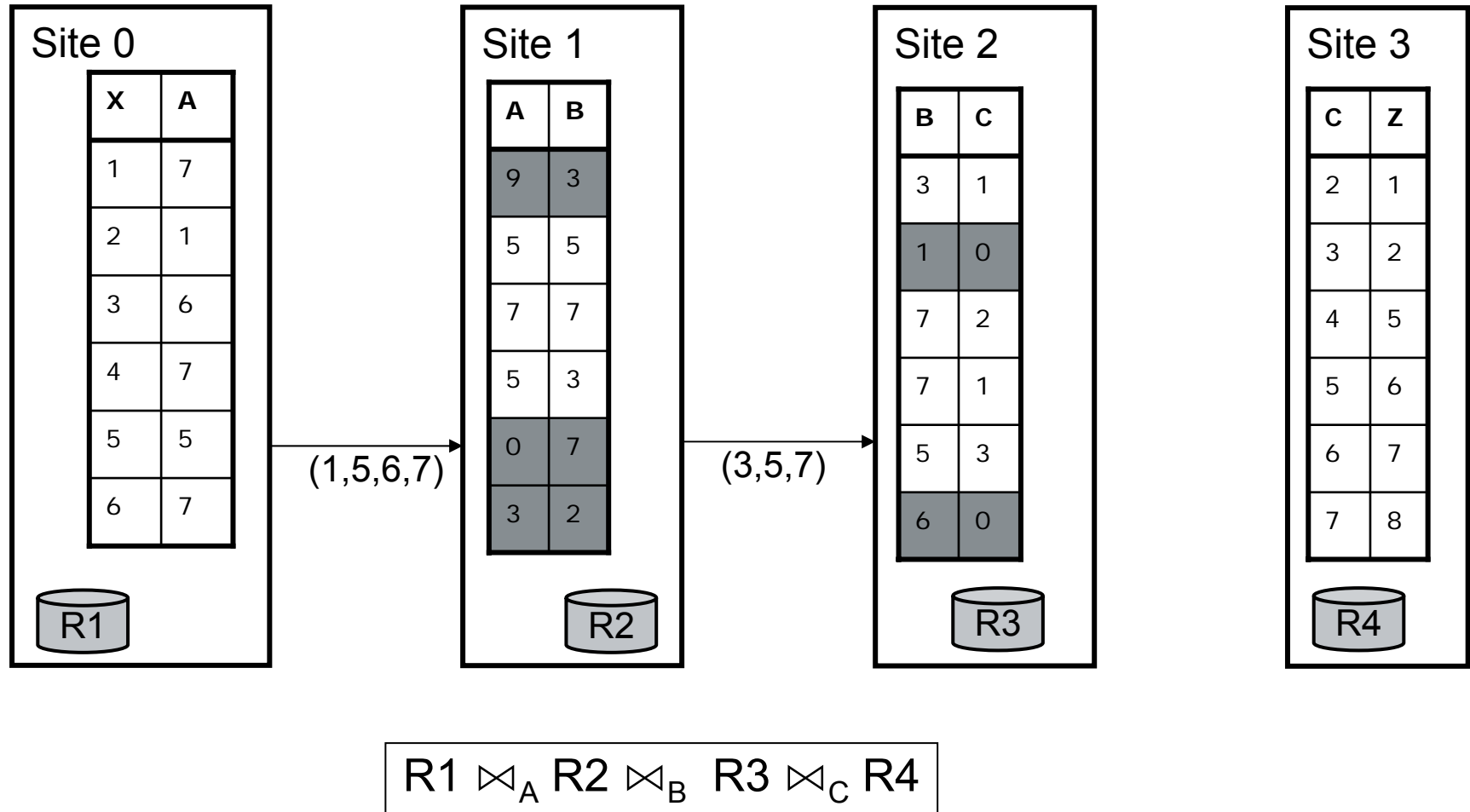
39



$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

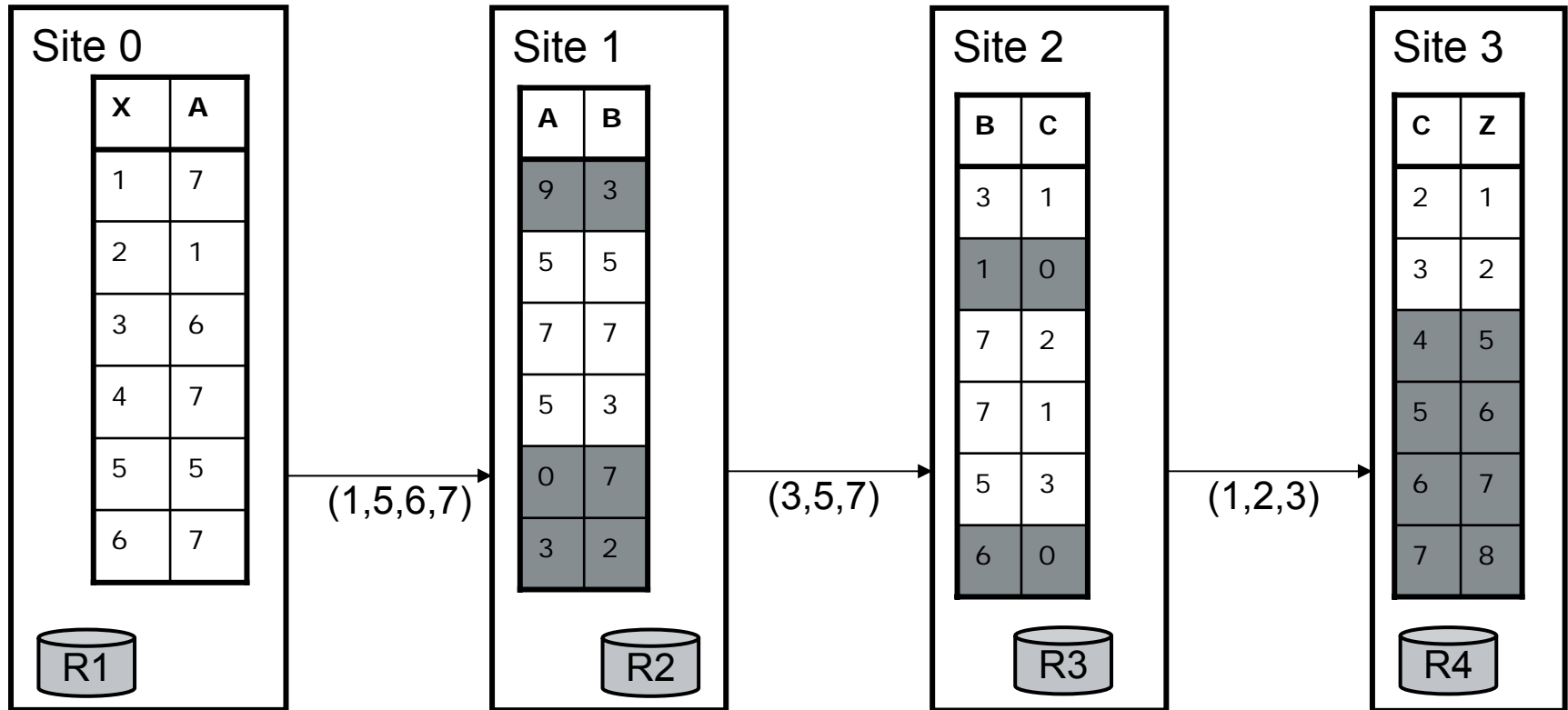
Fully Reduce – Beispiel

40



Fully Reduce – Beispiel

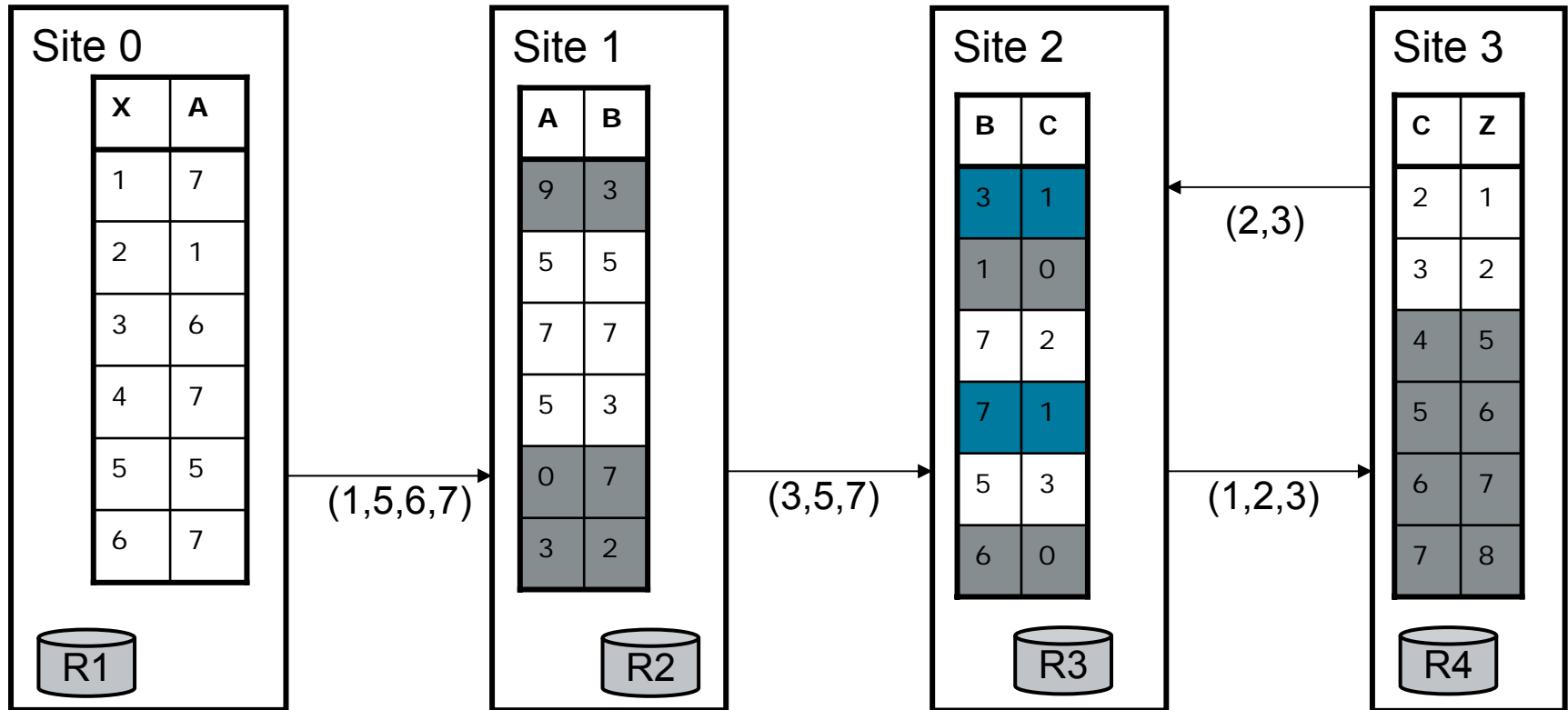
41



$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce – Beispiel

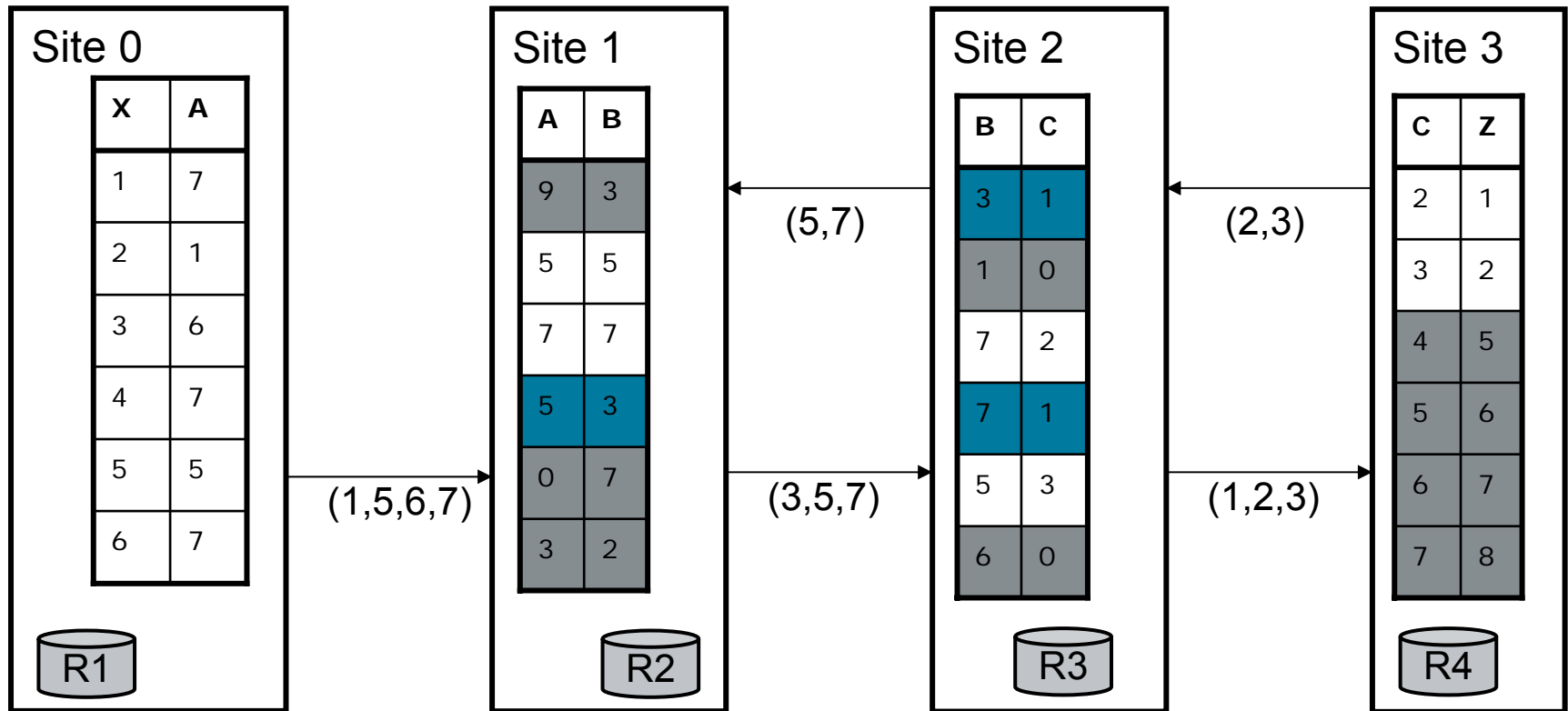
42



$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce – Beispiel

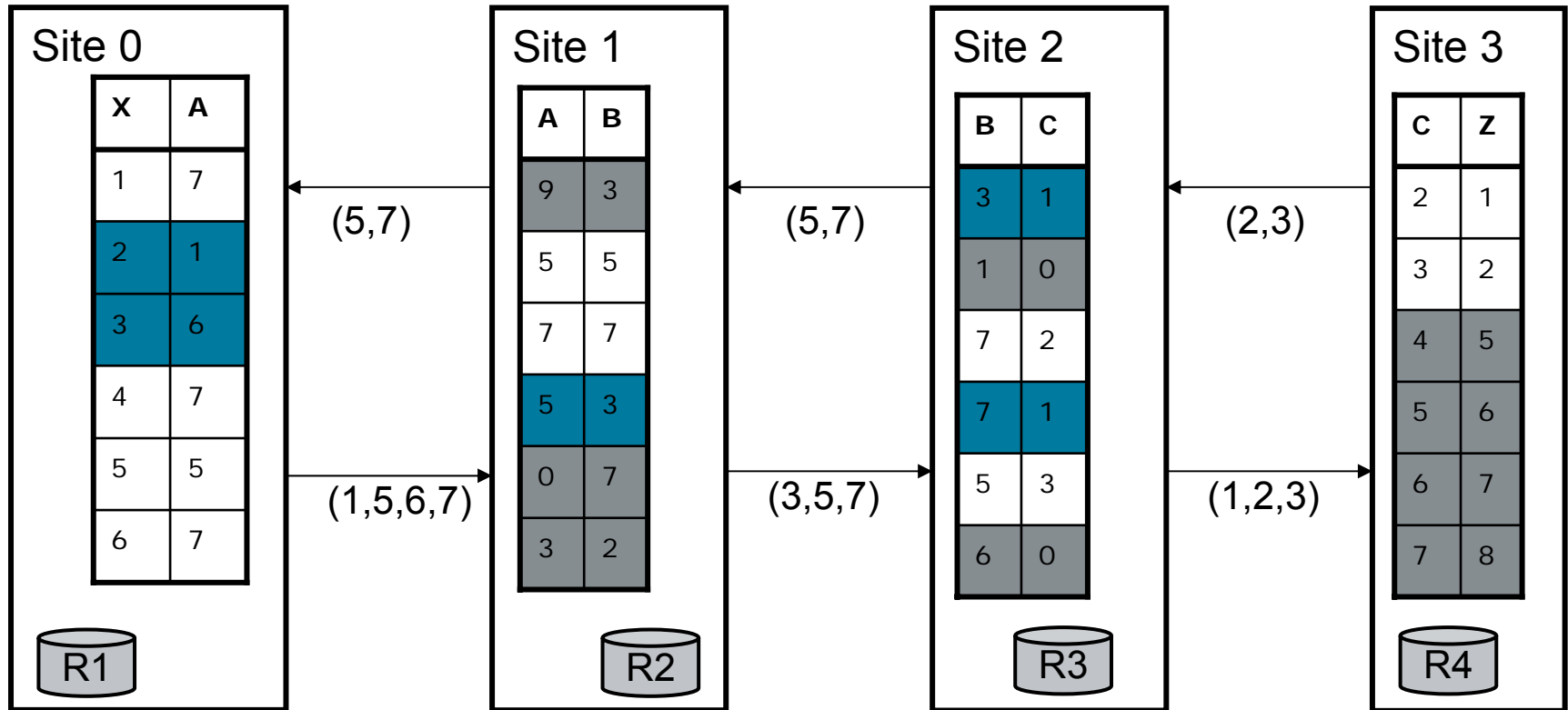
43



$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce – Beispiel

44



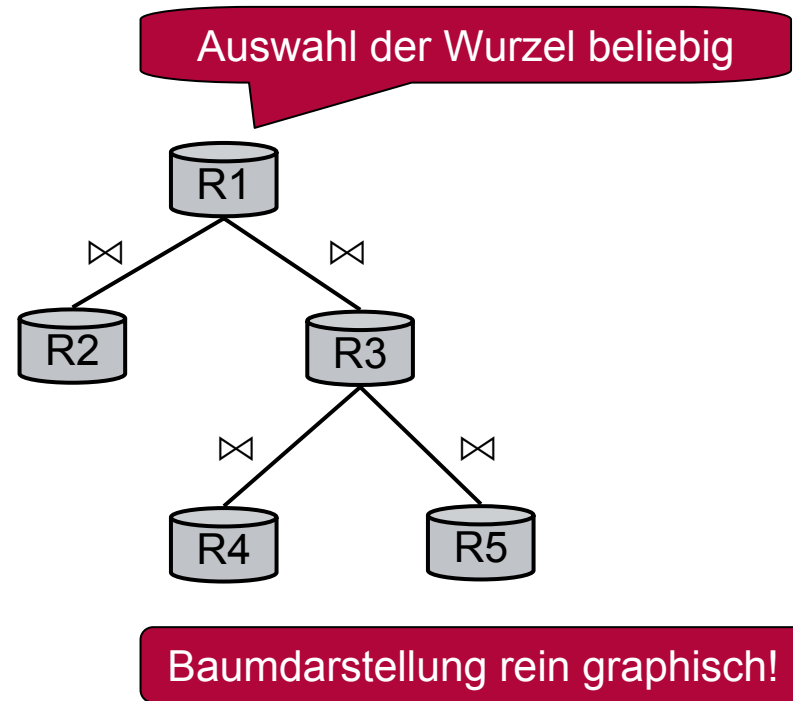
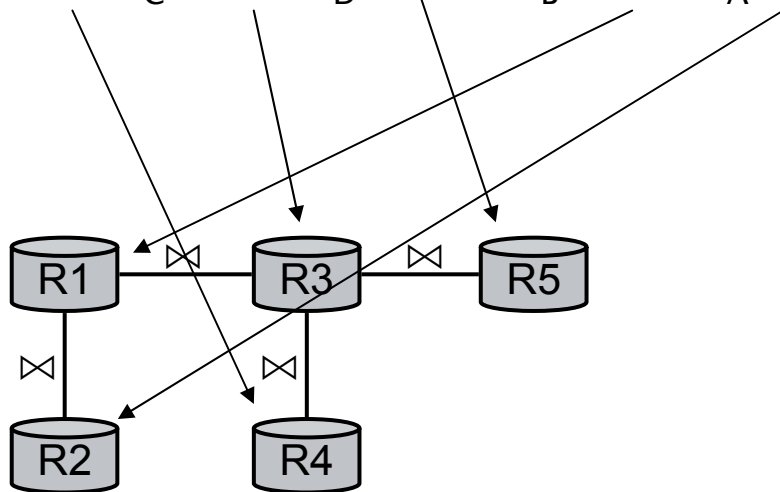
$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce

45

Gegeben Baum-Anfrage Q

$$\blacksquare (R4 \bowtie_C (R3 \bowtie_D R5)) \bowtie_B (R1 \bowtie_A R2)$$



Fully Reduce

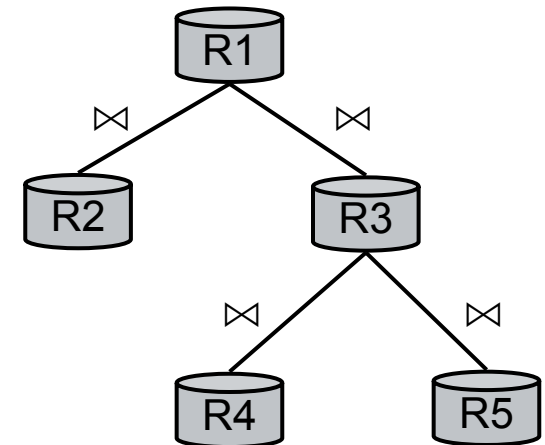
46

Phase 1

- Von unten nach oben
- Ziel: *Full reducer* für Wurzel
- Einführung von Semi-Joins von Knoten zu ihren Eltern

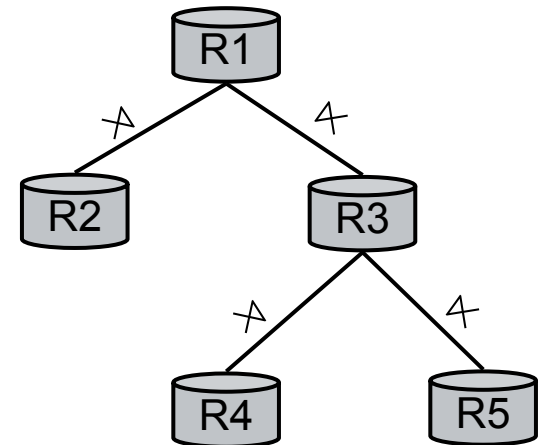
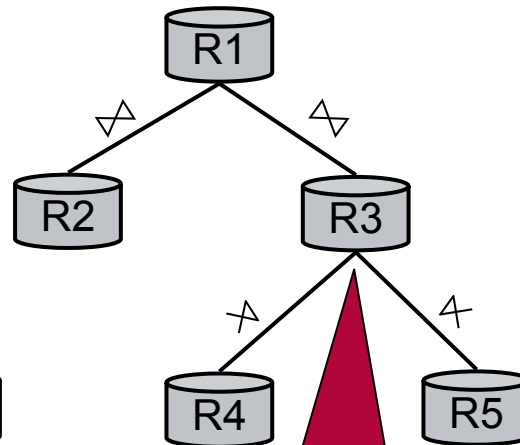
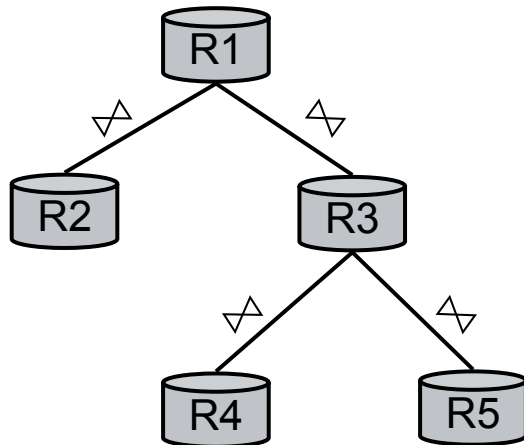
Phase 2

- Von oben nach unten
- Ziel: *Full reducer* für alle anderen Knoten
- Einführung von Semi-Joins von Knoten zu ihren Kindern



Fully Reduce – Phase 1

47



Erfüllte Bedingungen:

- R3.C = R4.C
- R3.D = R5.D
- R1.A = R2.A
- R1.B = R3.B

Erfüllte Bedingungen:

- R3.C = R4.C
- R3.D = R5.D

Ergebnis in R1 erfüllt alle Bedingungen, ist also fully reduced.

Optimierung mit Semi-Join

48

Zyklische Anfragen

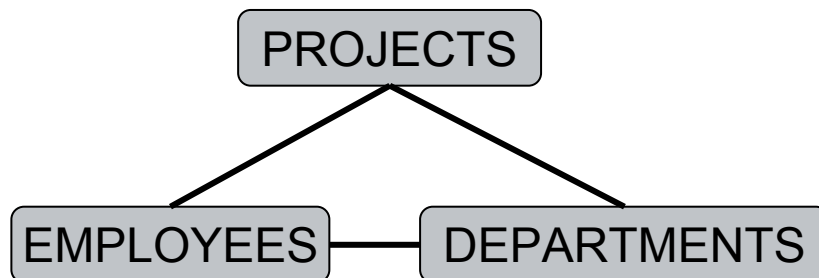
```
SELECT EMP.name, DEPT.name
FROM EMP, DEPT, PROJ
WHERE EMP.d_ID = DEPT.ID
AND EMP.p_ID = PROJ.ID
AND PROJ.d_ID = DEPT.ID
```

Mitarbeiter, die an Projekten der eigenen Abteilung arbeiten

DEPT	
DName	ID
a	1
b	2

PROJ		
d_ID	ID	PName
1	6	Clio
2	7	HumMer

EMP		
EName	d_ID	p_ID
x	1	7
y	2	6



Problem: Semi-Join betrachtet immer nur eine Kante im Anfragebaum und blickt nicht voraus. Ergebnis ist nie die leere Menge.

Überblick

49

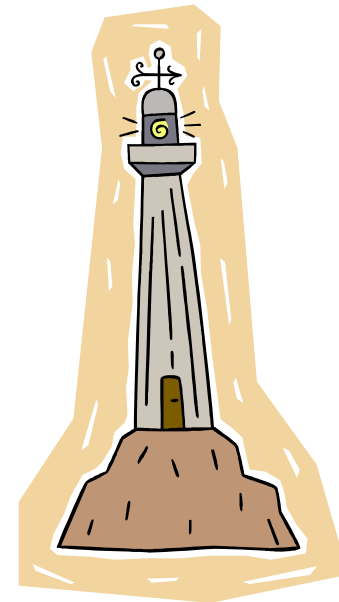
Anfragebearbeitung im Überblick

Techniken der Verteilten Anfragebearbeitung

- Row Blocking
- Multicasts
- Multithreading
- Partitionierung

Joinbearbeitung

- Semi-Join
- Reduzierung
- Semi-Join mit Filter



Optimierung mit Semi-Join

50

Verbesserung durch Hashfilter

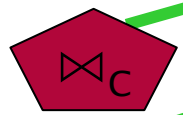
- Idee: Statt $\Pi_A(R)$ sende man nur eine „Signatur“ (Bloom-Filter).
- Weniger Netzwerkverkehr.

Join mit Hashfilter (Bloom-Filter)

51

$R \bowtie S$				
A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_3	b_3	c_1	d_1	e_1
a_5	b_5	c_3	d_2	e_2

er
jt



15 Werte

12 Werte (4 Tupel, inkl. 2 False Drops)

R		
A	B	C
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_1
a_4	b_4	c_2
a_5	b_5	c_3
a_6	b_6	c_2
a_7	b_7	c_6



6 Bit



S		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2
e_4	d_3	e_3
c_5	d_4	e_4
c_7	d_5	e_5
c_8	d_6	e_6
e_5	d_7	e_7

False drops

Rückblick

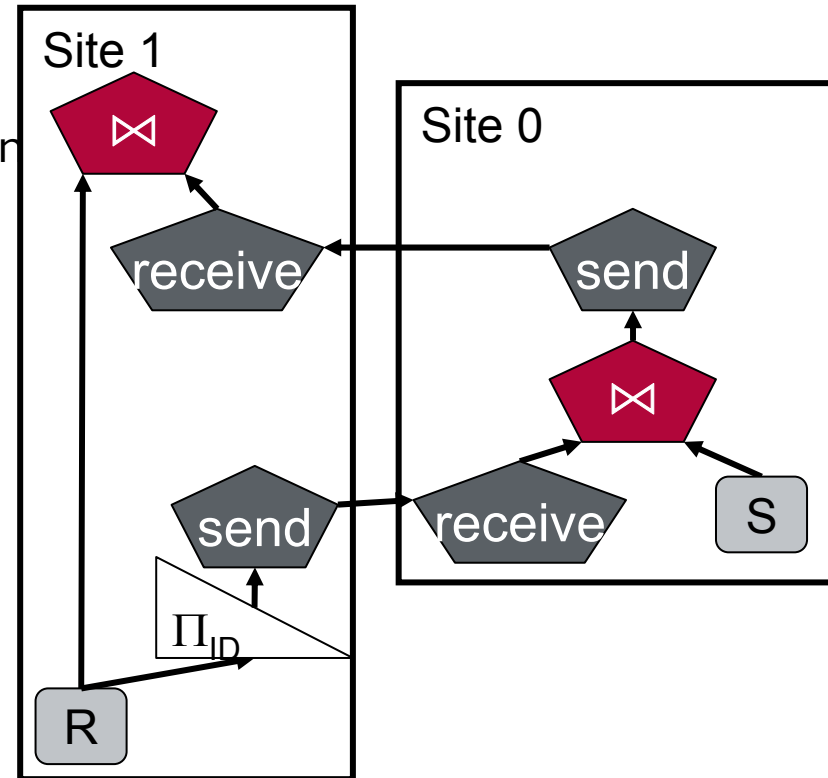
52

Techniken der verteilten Anfragebearbeitung

- Row Blocking
- Multicasts
- Multithreading
- Partitionierung

Semi-Join

- Basics
- Reduzierung
- Mit Filter



Überblick

- Fast jedes deutsche DBMS Lehrbuch
- Englisch: [GMUW00] Hector Garcia-Molina, Jeffrey D. Ullman, Jennifer Widom: Database System Implementation Prentice-Hall 2000

Spezielles

- [Ko00] The State of the Art in Distributed Query Processing, Donald Kossmann, ACM Computing Surveys 32(4), pages 422-469. (Link auf WWW)
- [Graefe93] Goetz Graefe: Query Evaluation Techniques for Large Databases. ACM Comput. Surv. 25(2): 73-170 (1993)

Semijoin

- [BC81] Philip A. Bernstein, Dah-Ming W. Chiu: Using Semi-Joins to Solve Relational Queries. Journal of the ACM 28(1): 25-40 (1981)