Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Informationsintegration
Verteilte Anfragebearbeitung

Felix Naumann

Überblick

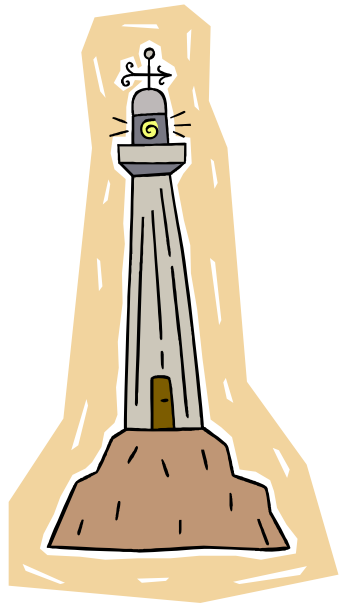
Anfragebearbeitung im Überblick

Techniken der Verteilten Anfragebearbeitung

- Row Blocking
- Multicasts
- Multithreading
- Partitionierung

Joinbearbeitung

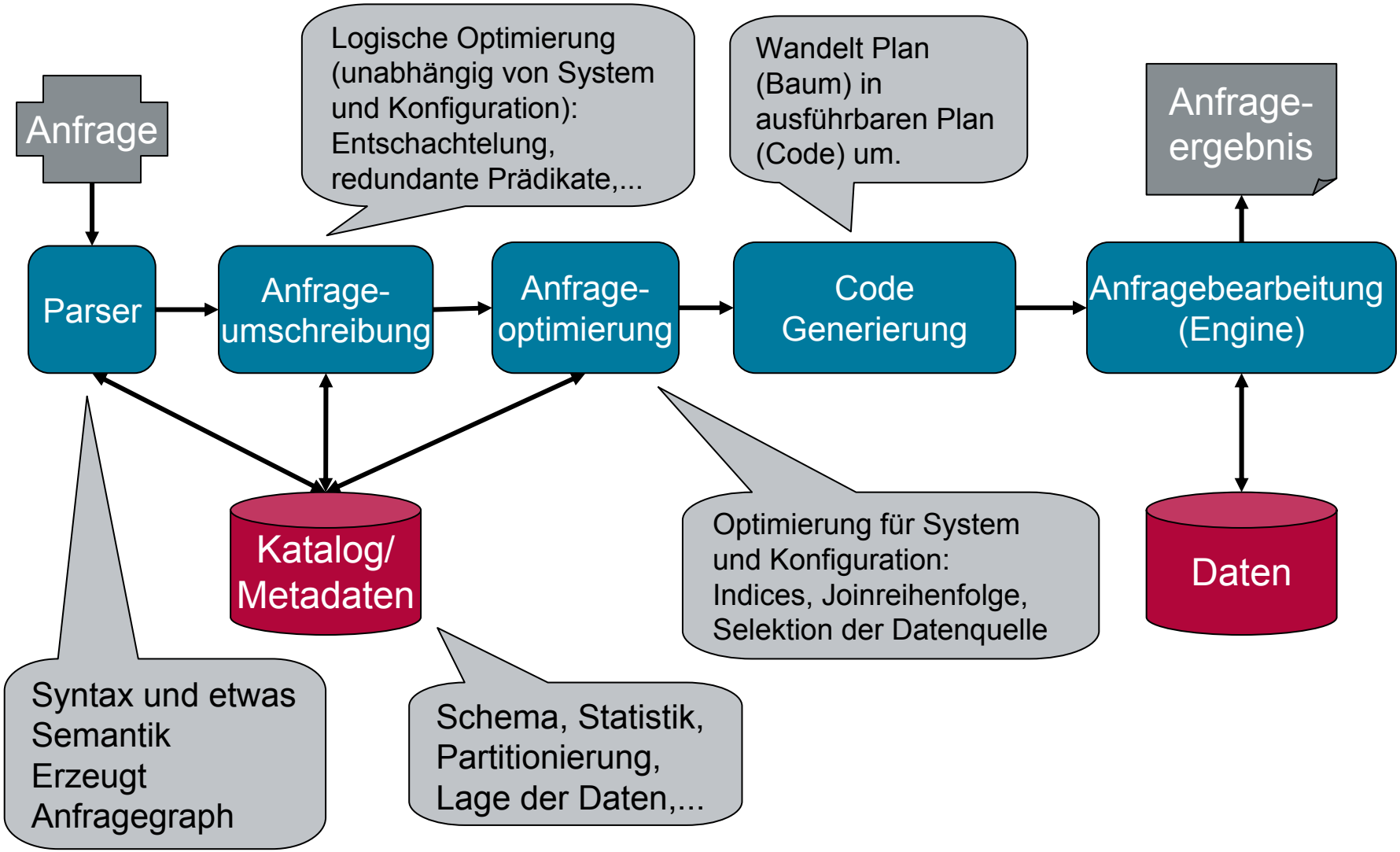
- Semi-Join
- Reduzierung
- Semi-Join mit Filter



Parallelität vs. Verteilung

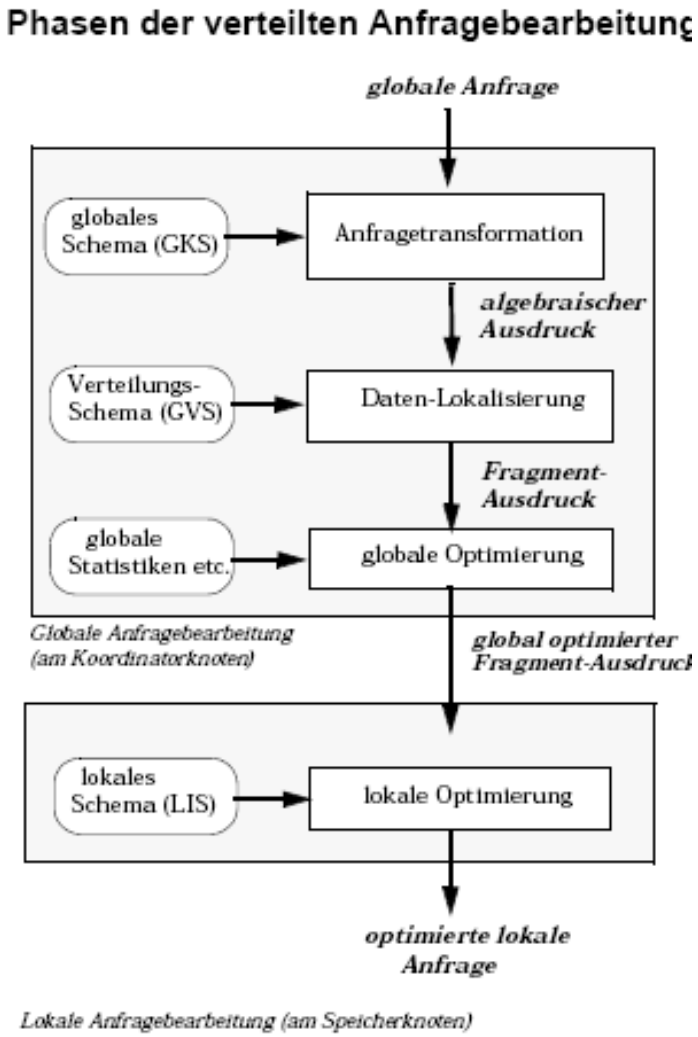
- Parallele DBMS
  - Shared memory
  - Shared disk
  - Shared nothing
  - Fokus auf Transaktionen und Anfragebearbeitung
- Verteilte DBMS
  - Shared nothing
  - Fokus auf Heterogenität und Anfragebearbeitung
  - Unser Fokus!

Architektur zur zentralen Anfragebearbeitung



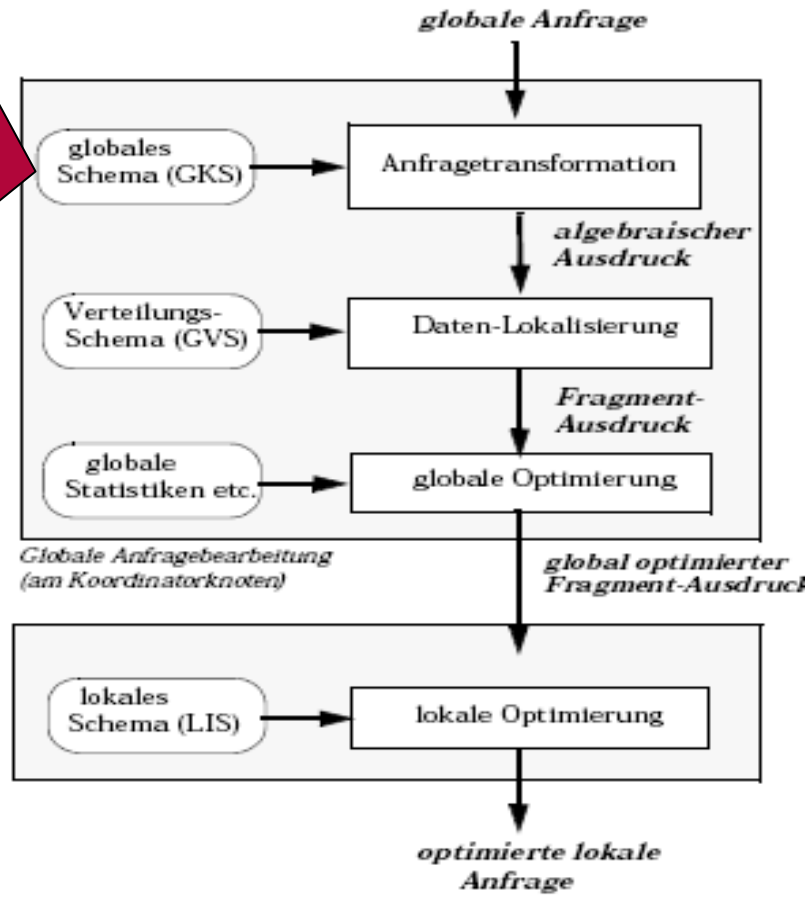
Anfragebearbeitung in verteilten Systemen

- 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)



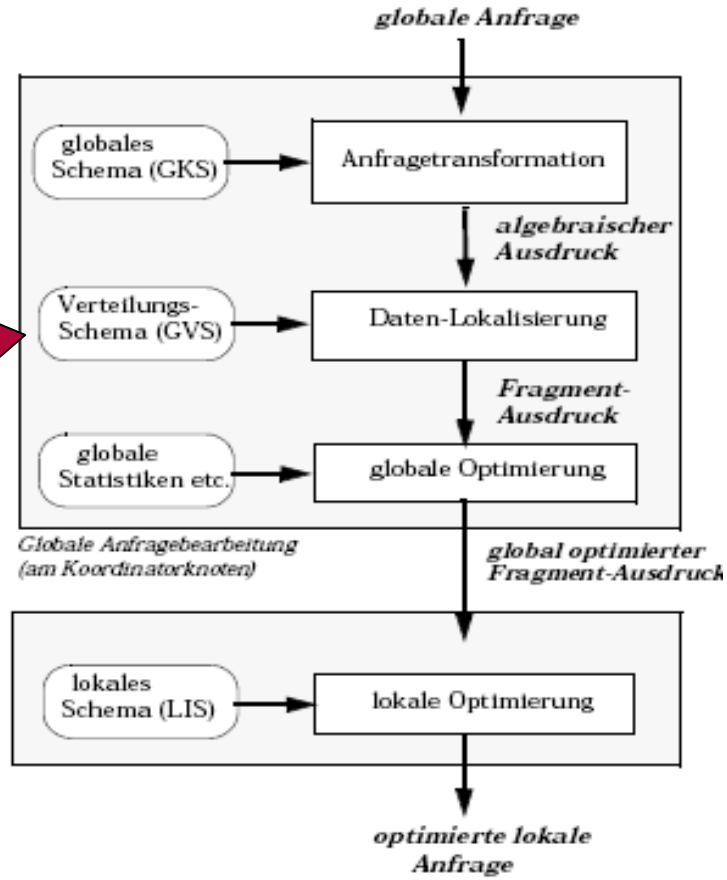
Schritt 1: Anfragetransformation

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

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



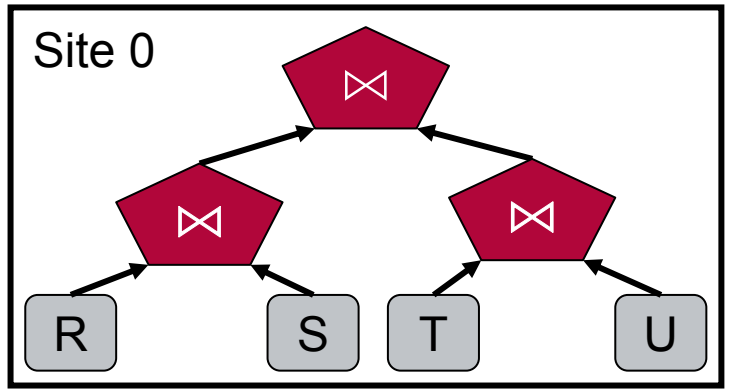
Schritt 3: Globale Optimierung

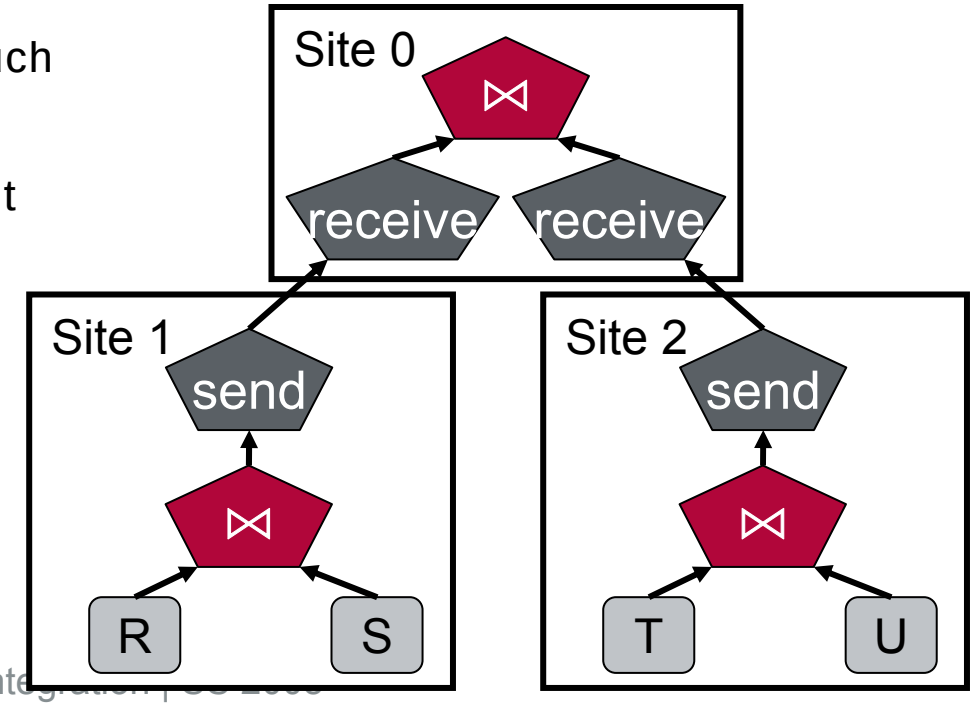
- Bestimmung eines Ausführungsplanes mit minimalen globalen Kosten
  - Bestimmung der Ausführungsknoten
  - Festlegung der Ausführungsreihenfolge (sequentiell, parallel)
  - Alternative Strategien zur Join-Berechnung bewerten (z.B. mit Semi-Join)
  - Aber: Trennung zwischen globaler und lokaler Optimierung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

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

Neue Ziele der Anfragebearbeitung

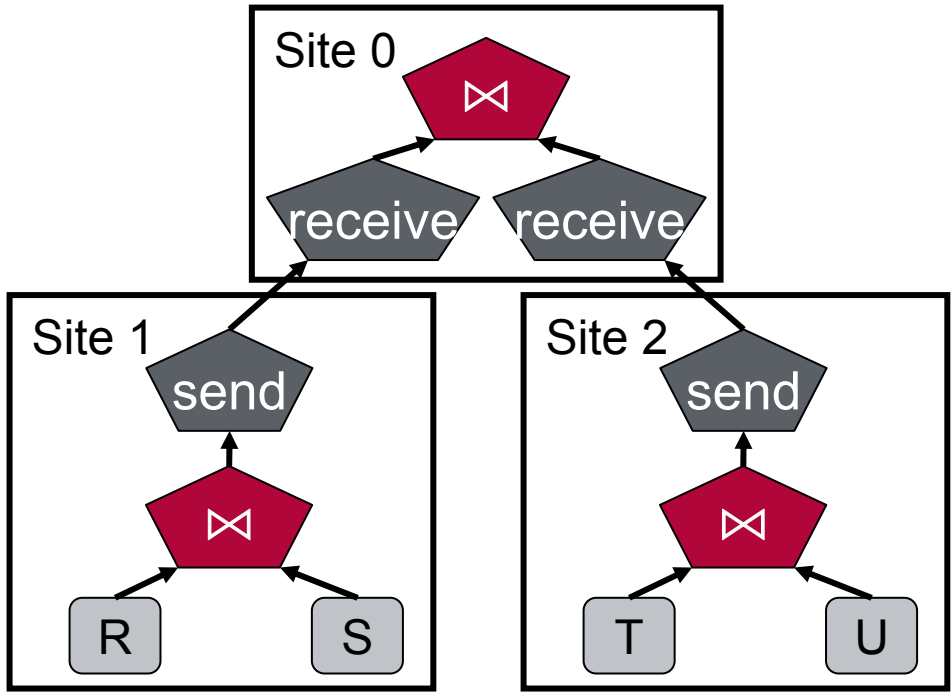
- In zentralisierten (mehr-Benutzer) DBMS
  - Durchsatz (*throughput*) = Anzahl der verarbeiteten Tupel
  - Minimierung des Ressourcenverbrauchs
- In verteilten DBMS
  - Gesamt-Ressourcenverbrauch erhöhen um schnellere Antworten zu erhalten
  - Minimierung der Antwortzeit





Dimensionen der Anfragebearbeitung

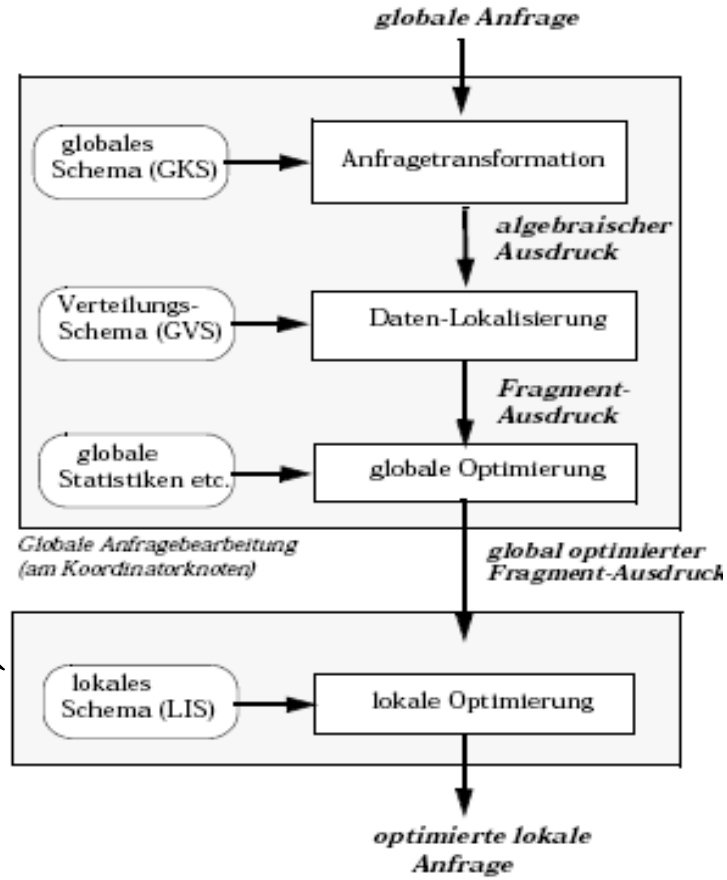
- Festlegung des Ausführungsknotens
- Festlegung der Auswertungsstrategie
  - *Ship whole*
    - Vollständige Relationen
    - Wenig Nachrichten
    - Viele Byte
  - *Fetch rows as needed*
    - Bindings
    - Viele Nachrichten
    - Nur relevante Byte
    - Semi Join
  - *Fetch columns as needed*
 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



Schritt 4: Lokale Optimierung

Entsprechend des jeweiligen Systems

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



Überblick

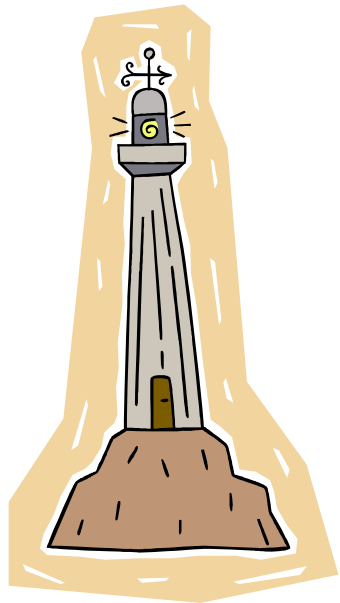
Anfragebearbeitung im Überblick

Techniken der Verteilten Anfragebearbeitung

- Row Blocking
- Multicasts
- Multithreading
- Partitionierung

Joinbearbeitung

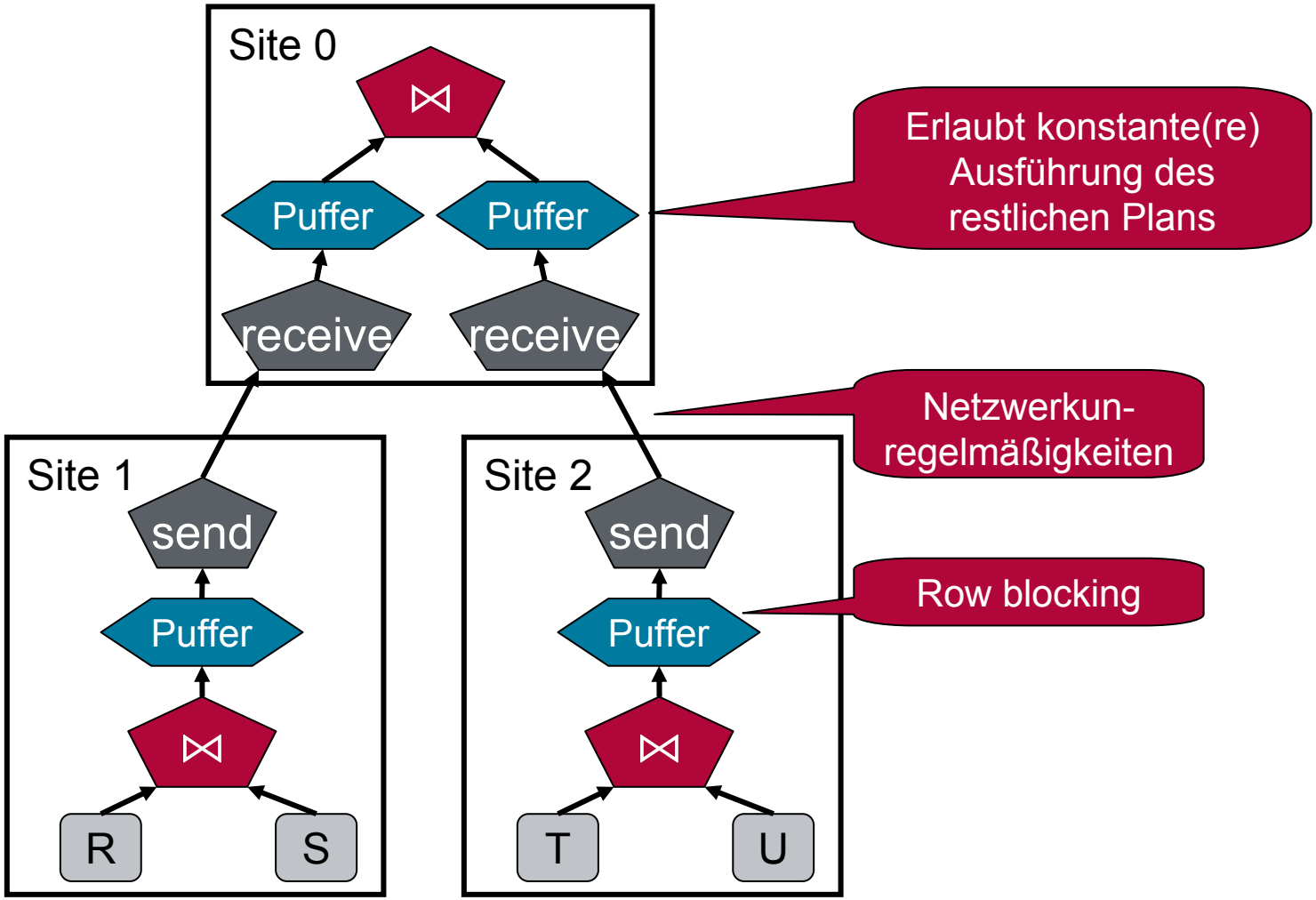
- Semi-Join
- Reduzierung
- Semi-Join mit Filter



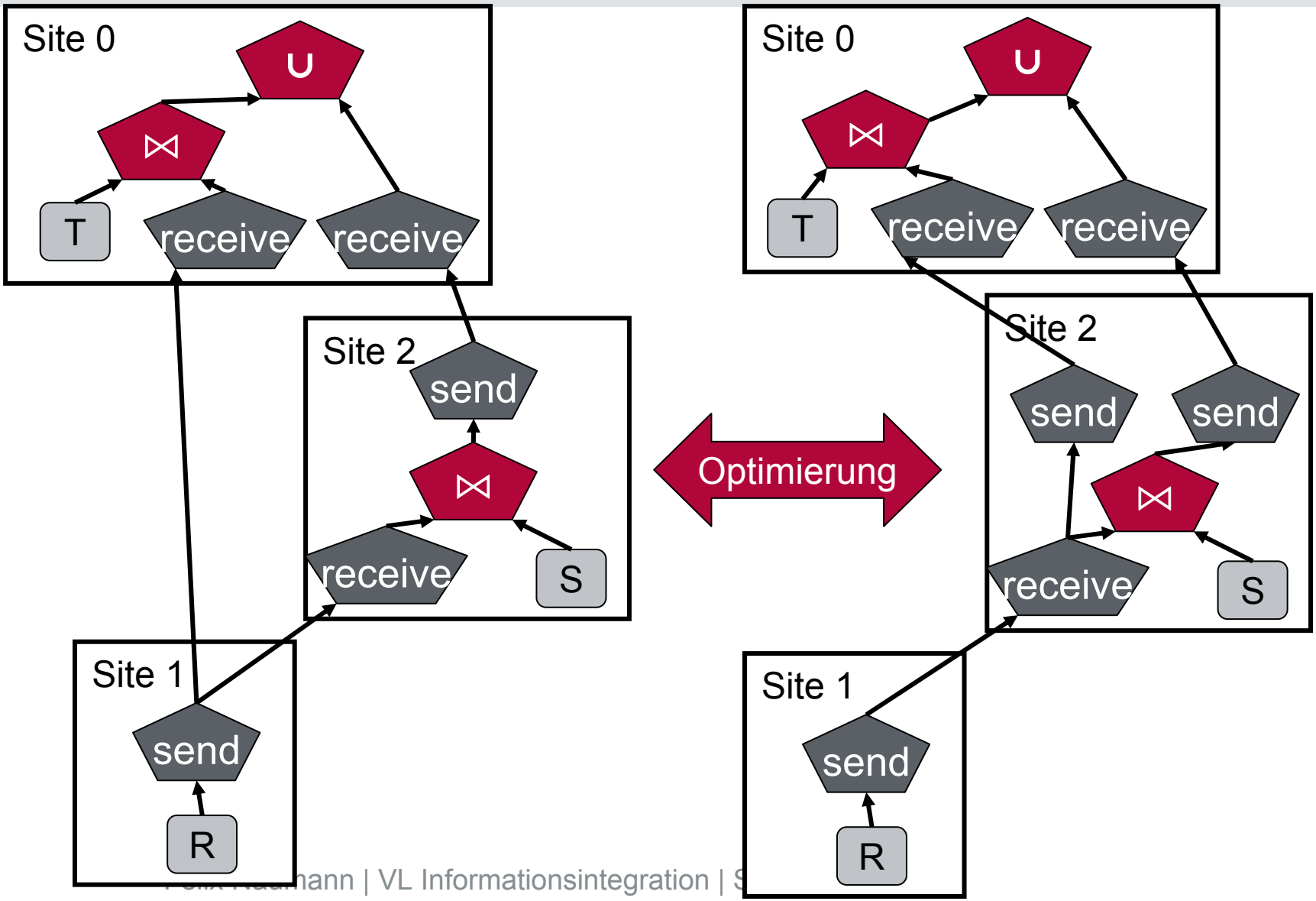
Row Blocking

- 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



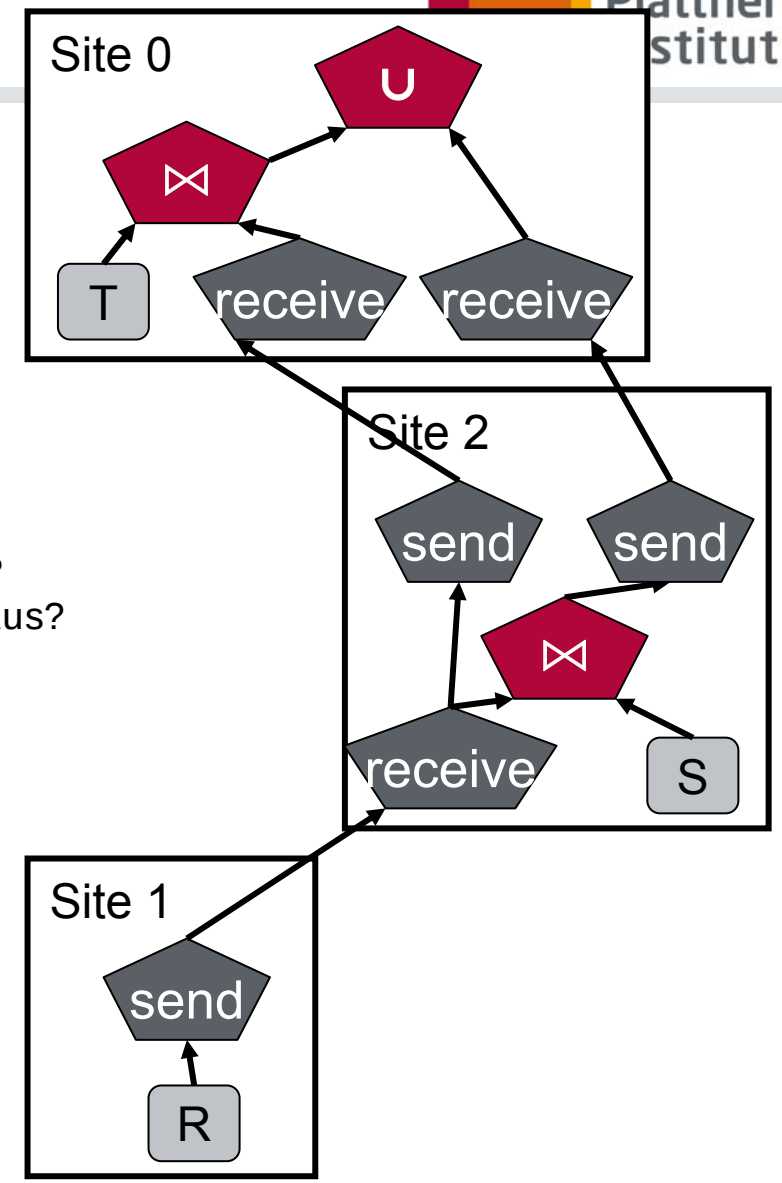
Multicasts


Site 0
U
⋈
T
receive
receive
Site 2
send
⋈
receive
S
Site 1
send
R
Optimierung
Site 0
U
⋈
T
receive
receive
Site 2
send
send
⋈
receive
S
Site 1
send
R

Multicasts

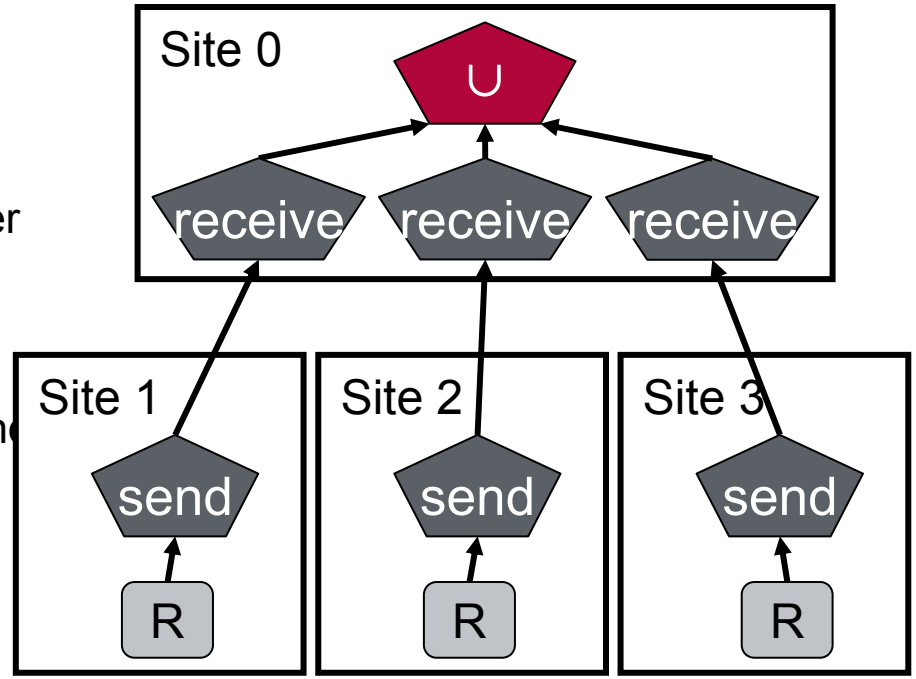
Einsparungen
- Netzverkehr
- CPU (packing und unpacking)

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

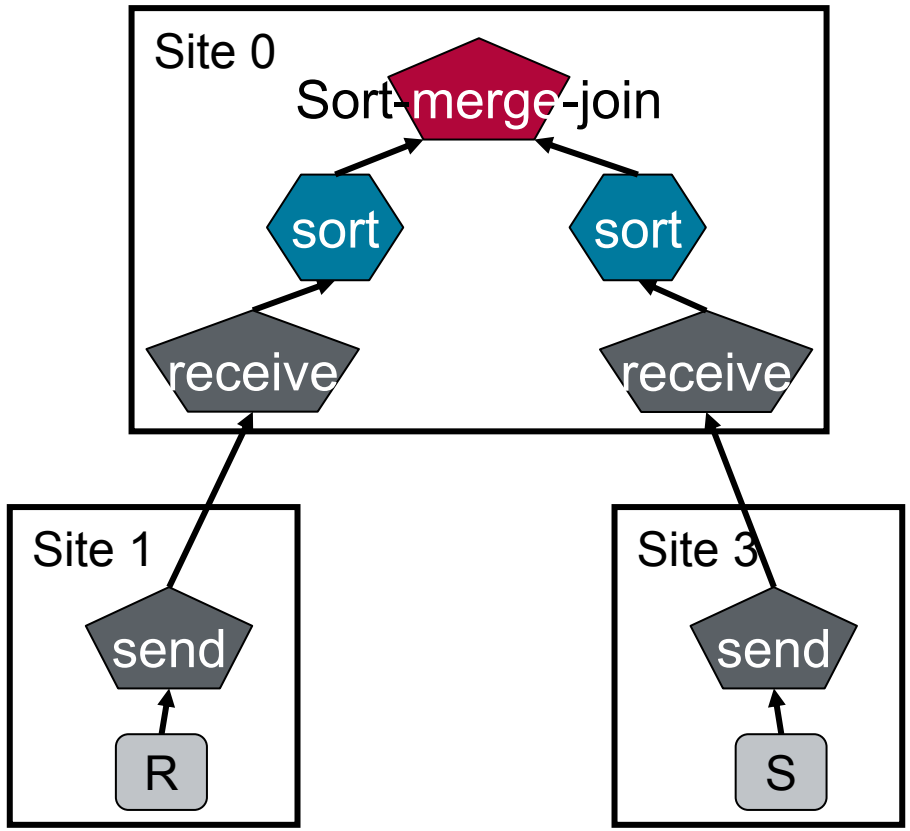
- Single-threaded UNION:
  - Sukzessive ein Datagram von einer Site
  - „Round-robin“-Verfahren
- Multi-threaded UNION:
  - Parallele Verarbeitung der send und receive Operationen
- Vorteile
  - Netzwerkverkehr parallel
  - Send Operationen parallel
- Nachteile
  - Zusätzliche Kosten durch Synchronisation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



Multi-Threading

Multi-threading nicht immer vorteilhaft

- z.B. Speicherplatz



Partitionierung

- 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*

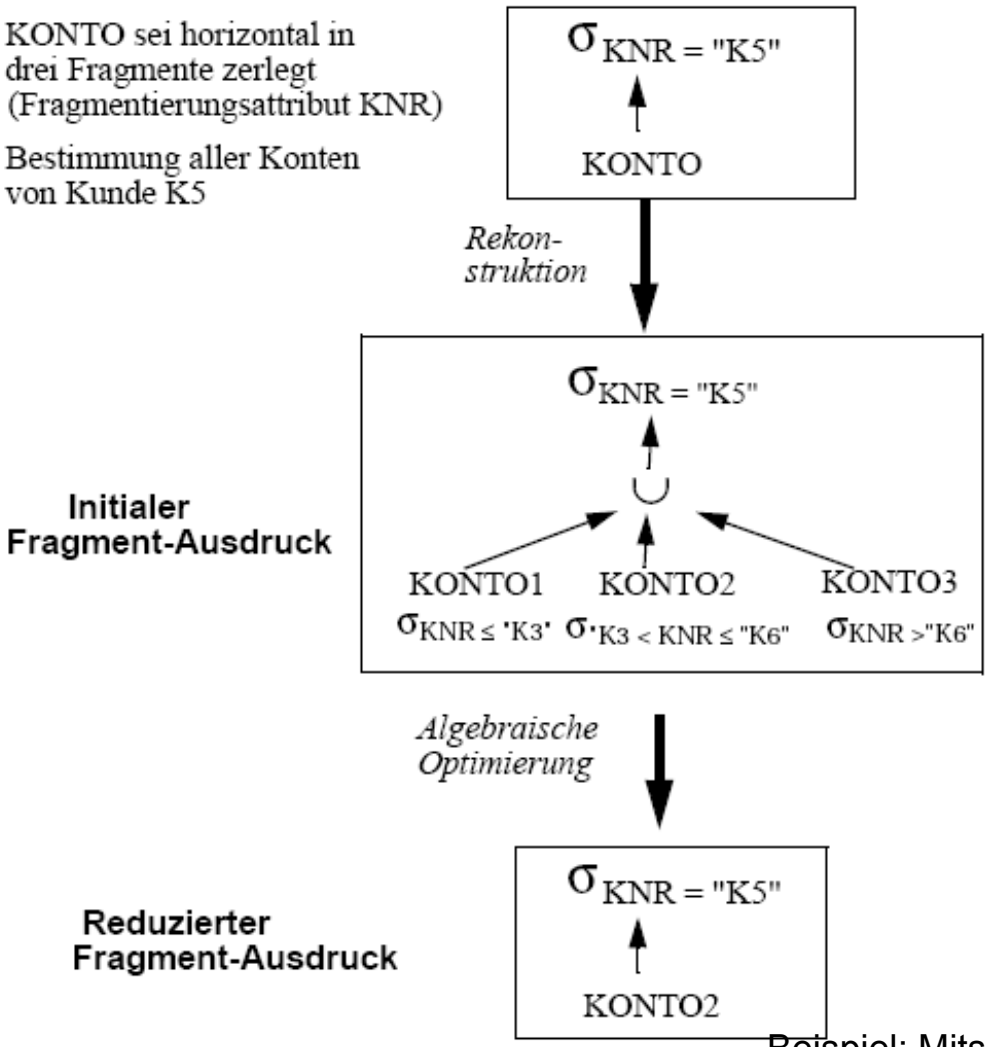
Joins über horizontale Partitionen

- 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

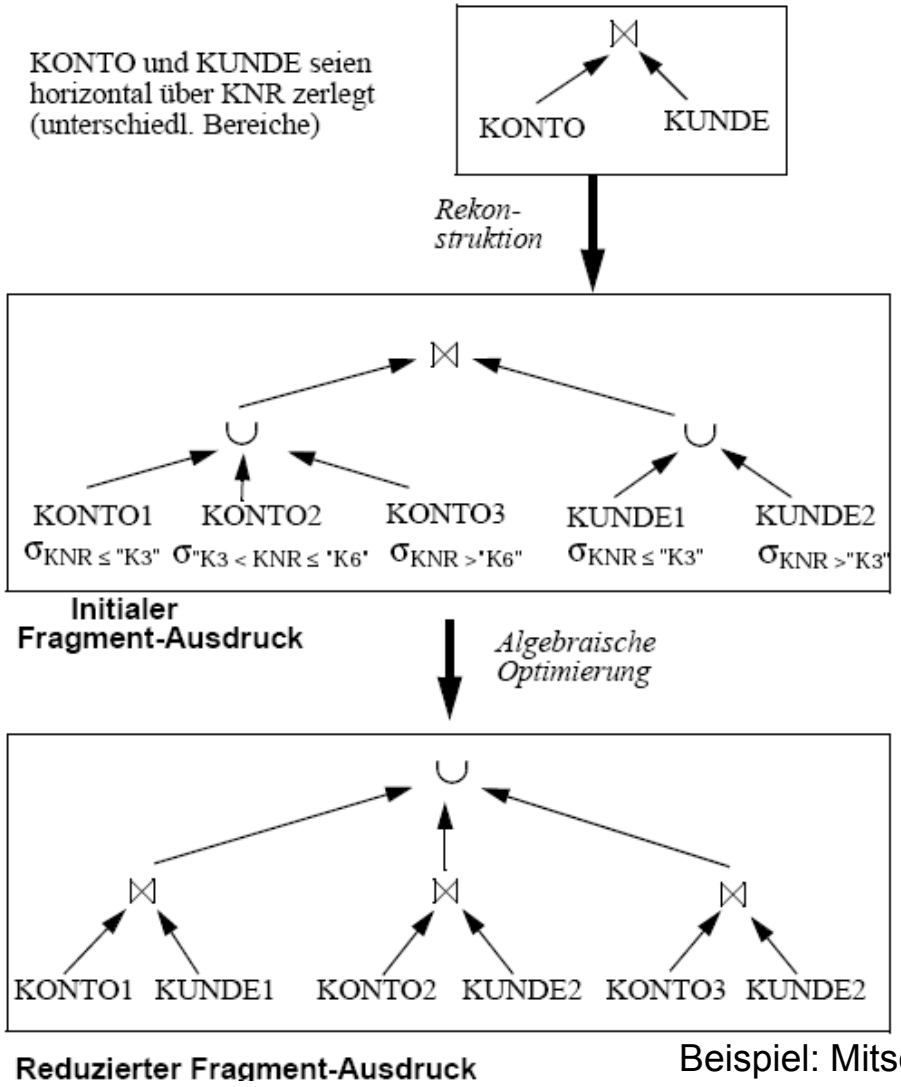
KONTO sei horizontal in drei Fragmente zerlegt (Fragmentierungsattribut KNR)

Bestimmung aller Konten von Kunde K5



Beispiel: Mitschang, VL „Verteilte DBMS“

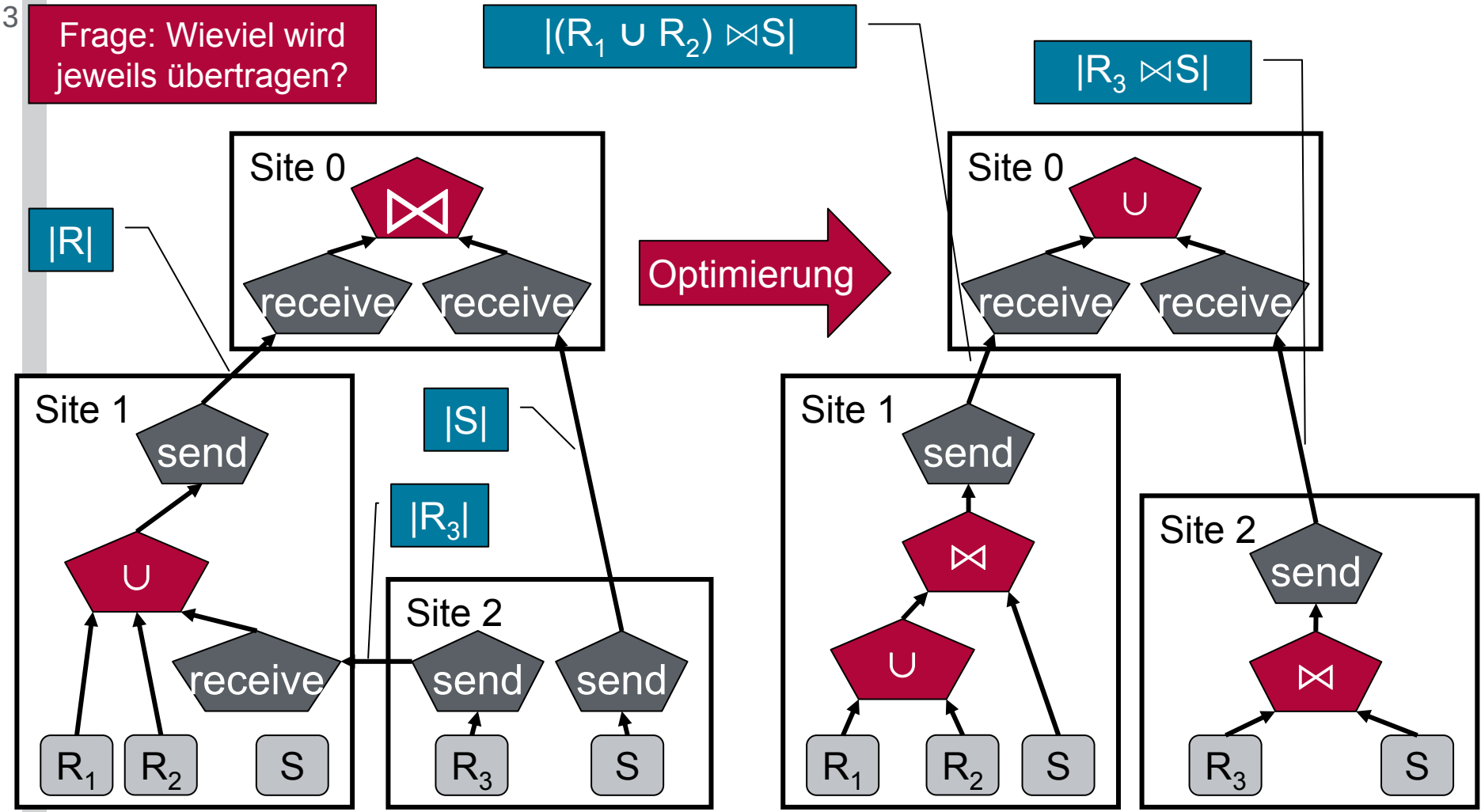
Join über horizontale Partitionierung



Beispiel: Mitschang, VL „Verteilte DBMS“

Joins über horizontale Partitionen

Frage: Wieviel wird jeweils übertragen?



Joins über horizontale Partitionen

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

Überblick

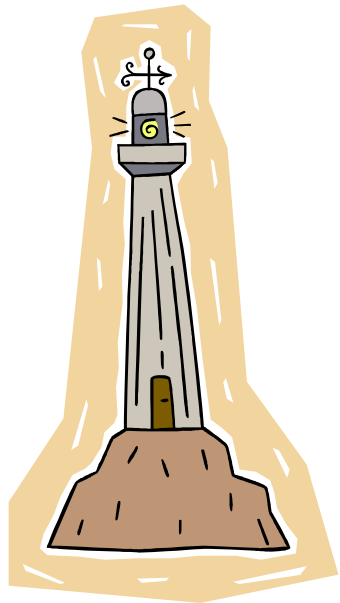
Anfragebearbeitung im Überblick

Techniken der Verteilten Anfragebearbeitung

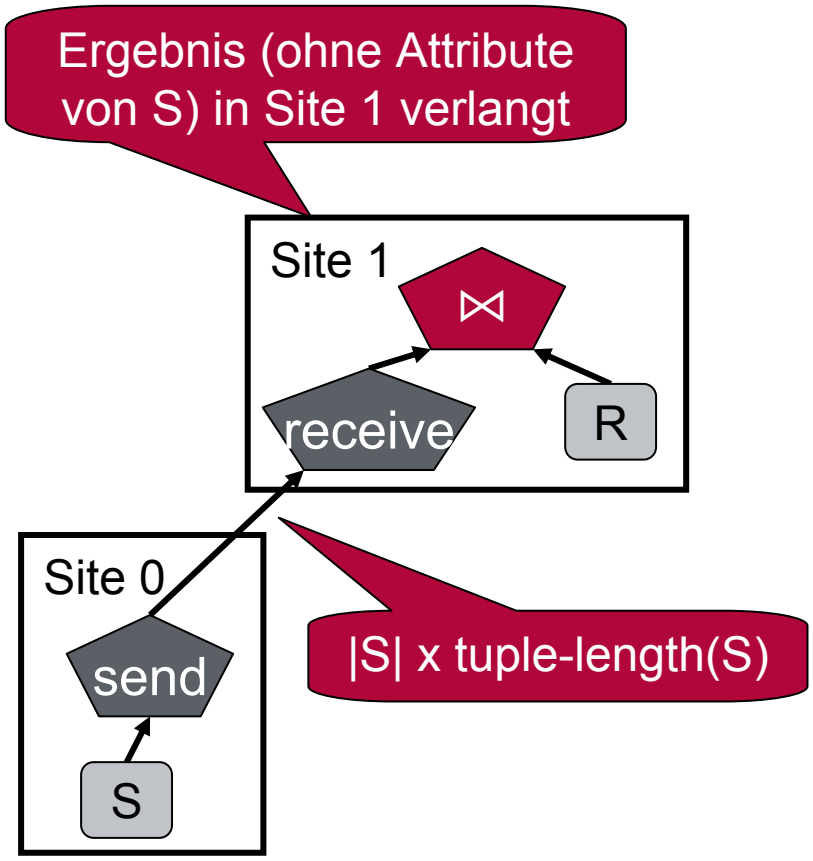
- Row Blocking
- Multicasts
- Multithreading
- Partitionierung

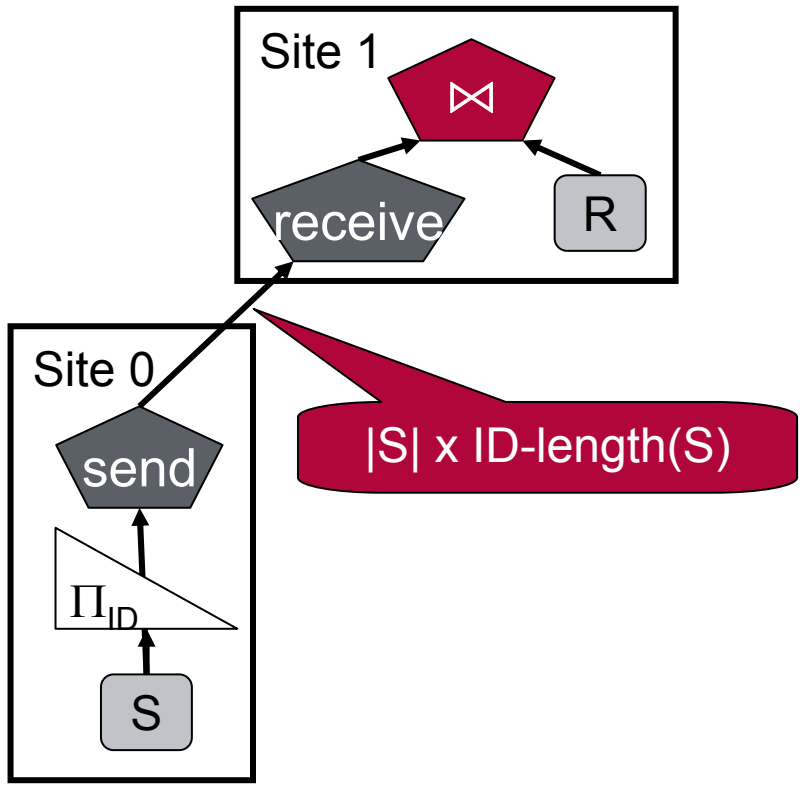
Joinbearbeitung

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

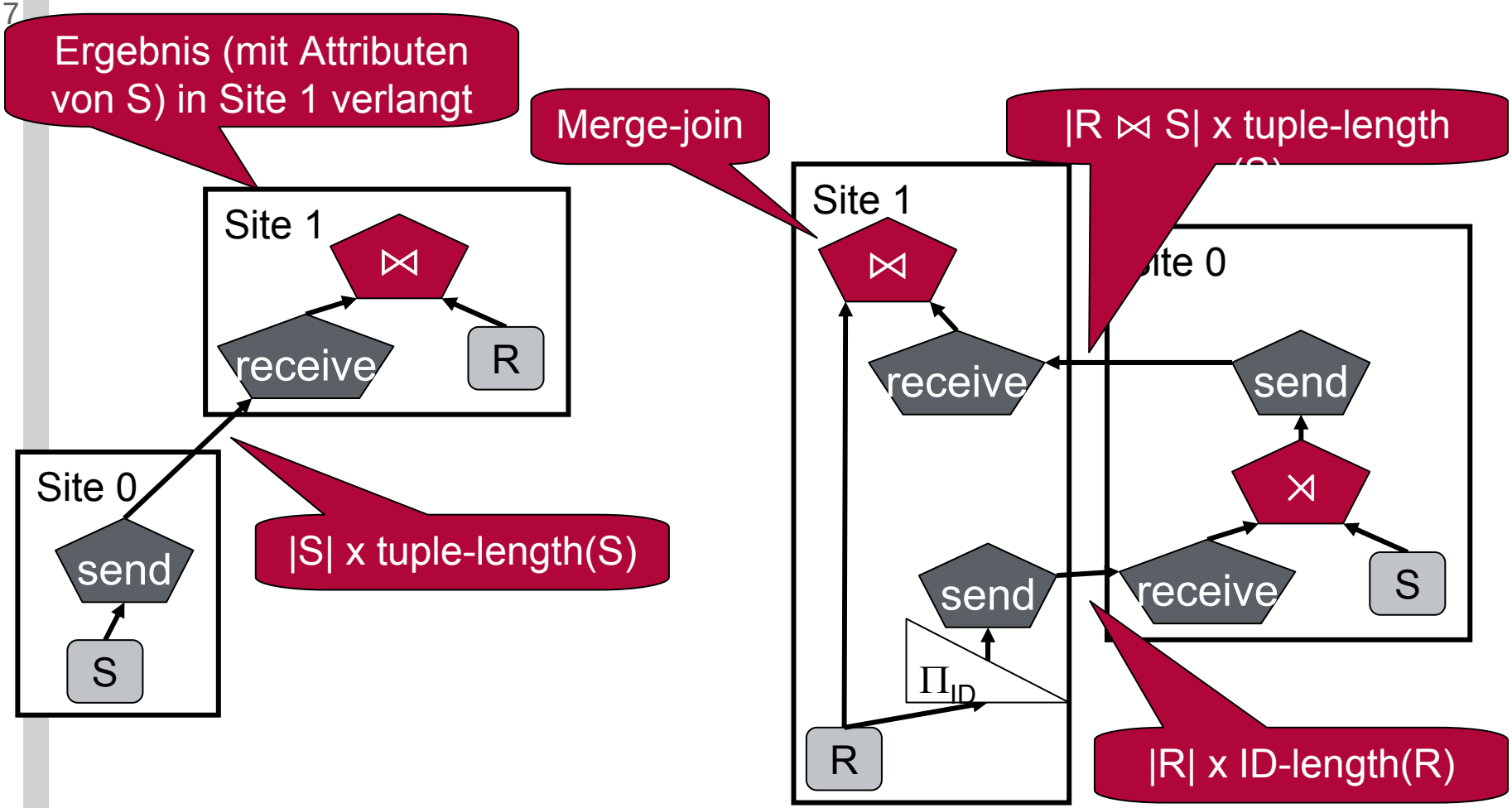


Semi-Join


Ergebnis (ohne Attribute von S) in Site 1 verlangt
Site 1
⋈
receive
R
Site 0
send
S
|S| x tuple-length(S)


Site 1
⋈
receive
R
Site 0
send
Π_{ID}
S
|S| x ID-length(S)

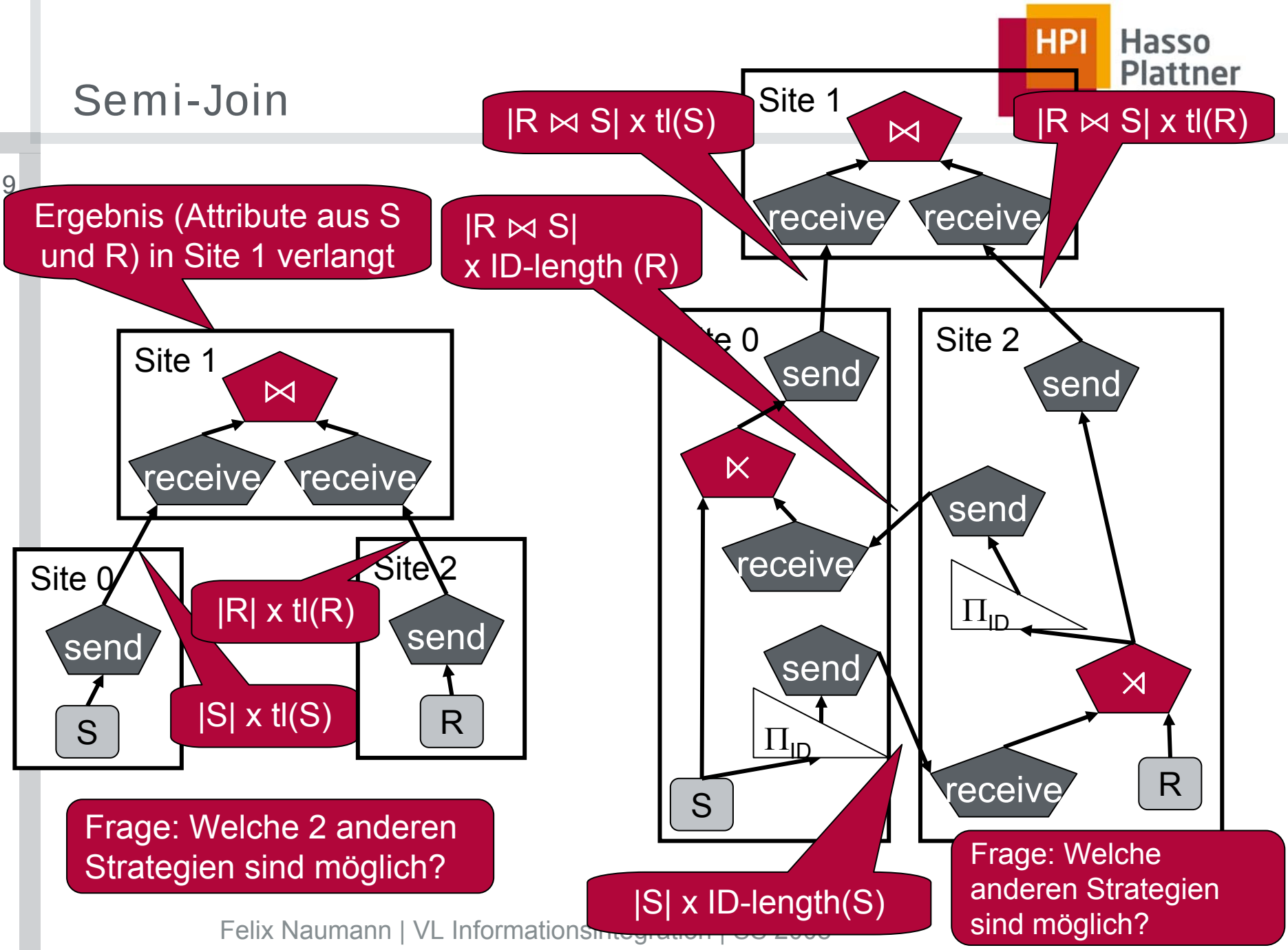
Semi-Join


Ergebnis (mit Attributen von S) in Site 1 verlangt
Merge-join
|R ⋈ S| x tuple-length
Site 1
⋈
receive
R
Site 0
send
S
|S| x tuple-length(S)
Site 1
⋈
receive
Site 0
send
⋈
send
receive
S
Π_{ID}
R
|R| x ID-length(R)

Semi-Join

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

Semi-Join


Ergebnis (Attribute aus S und R) in Site 1 verlangt
|R ⋈ S| x tl(S)
|R ⋈ S| x tl(R)
|R ⋈ S| x ID-length (R)
Site 1
⋈
receive
receive
Site 0
send
S
Site 2
send
R
|R| x tl(R)
|S| x tl(S)
⋉
send
receive
send
Π_{ID}
send
Π_{ID}
receive
⋊
R
S
|S| x ID-length(S)
Frage: Welche 2 anderen Strategien sind möglich?
Frage: Welche anderen Strategien sind möglich?

Optimierung mit Semi-Join

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

Frage: Welche beiden Varianten standen auf der vorigen Folie?

Optimierung mit Semi-Join

- 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)
  - $K((R \ltimes_F S) \bowtie_F S)$ = byte(Π_F(S)) + byte((R $\bowtie_F$ S))

Überblick

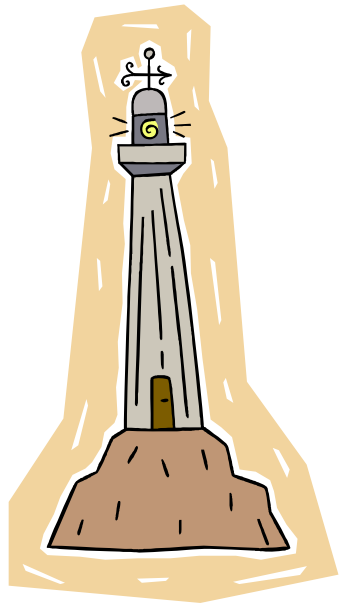
Anfragebearbeitung im Überblick

Techniken der Verteilten Anfragebearbeitung

- Row Blocking
- Multicasts
- Multithreading
- Partitionierung

Joinbearbeitung

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



Optimierung mit Semi-Join

Komplexeres Beispiel

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

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

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

Ermittlung der full reducer für jede Relation in 2 Phasen

1. Vorwärts
2. Rückwärts

Allgemein für Kettenanfragen

- $R1 \bowtie_A R2 \bowtie_B \ldots \bowtie_Y R(n-1) \bowtie_Z Rn$
- Vorwärts
  - $R2' = R2 \ltimes R1$
  - $R3' = R3 \ltimes R2' = R3 \ltimes (R2 \ltimes R1)$
  - ...
  - $Rn' = Rn \ltimes R(n-1)' = \ldots$
- Rückwärts
  - $R(n-1)'' = R(n-1)' \ltimes Rn'$
  - $R(n-2)'' = R(n-2)' \ltimes R(n-1)'$
  - ...
  - $R1'' = R1 \ltimes R2''$

Full reducer für Rn

Full reducer für R(n-1)

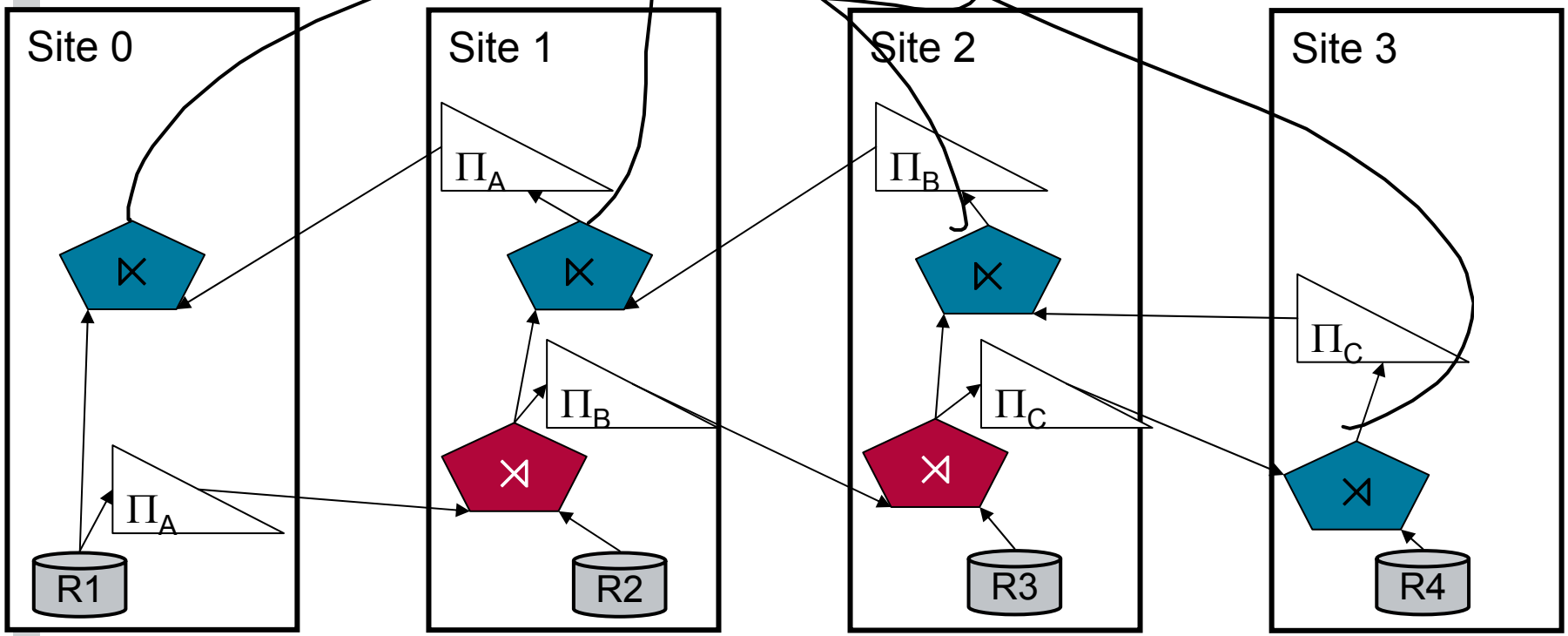
Full reducer für R1

Fully Reduce

Beispiel für Kettenanfragen

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

Fully Reduce – Beispiel



Vorwärts
R2‘ = R2 ⋉ R1
R3‘ = R3 ⋉ R2‘
R4‘ = R4 ⋉ R3‘

Rückwärts
R3‘‘ = R3‘ ⋉ R4‘
R2‘‘ = R2‘ ⋉ R3‘‘
R1‘‘ = R1 ⋉ R2‘‘

Full reducer

Fully Reduce – Beispiel

Site 0

X	A
1	7
2	1
3	6
4	7
5	5
6	7

R1

Site 1

A	B
9	3
5	5
7	7
5	3
0	7
3	2

R2

Site 2

B	C
3	1
1	0
7	2
7	1
5	3
6	0

R3

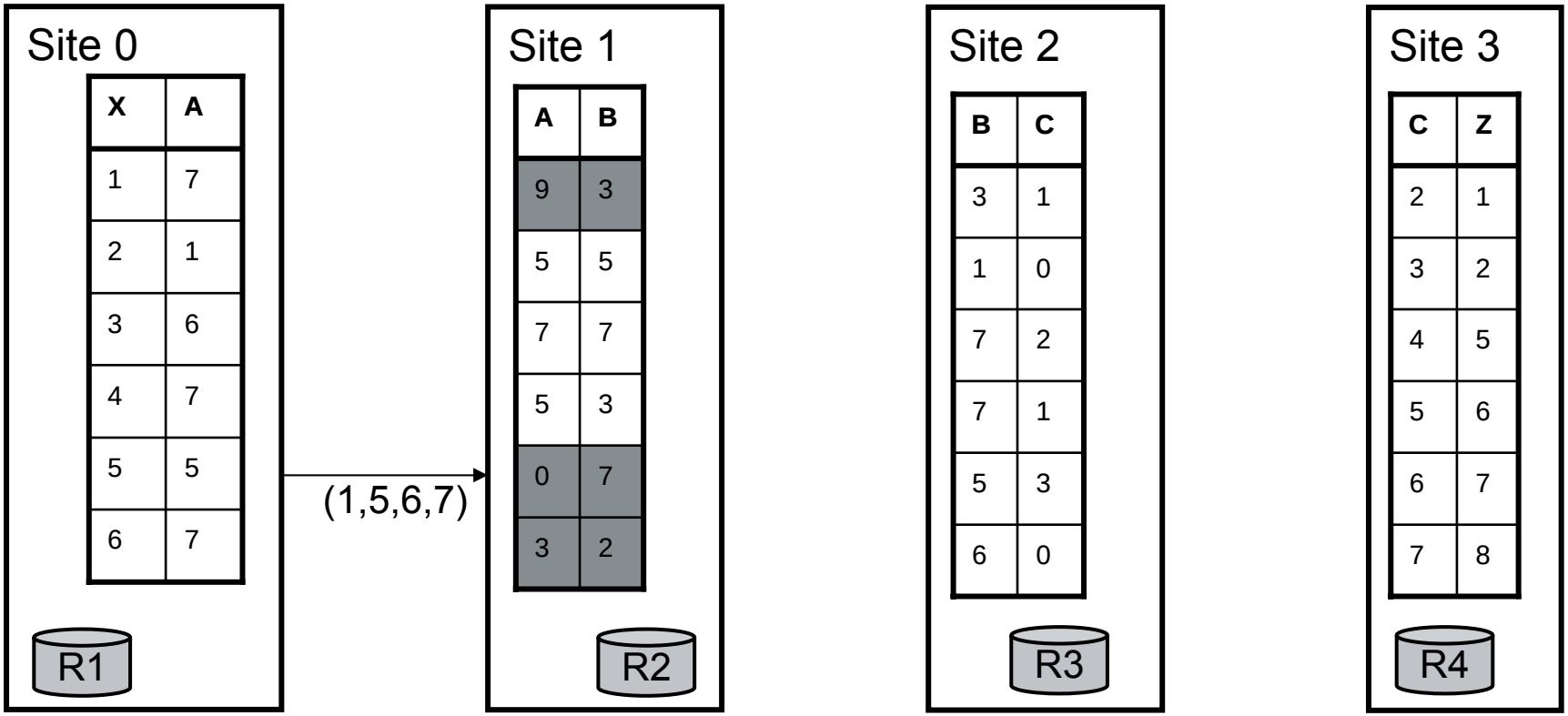
Site 3

C	Z
2	1
3	2
4	5
5	6
6	7
7	8

R4

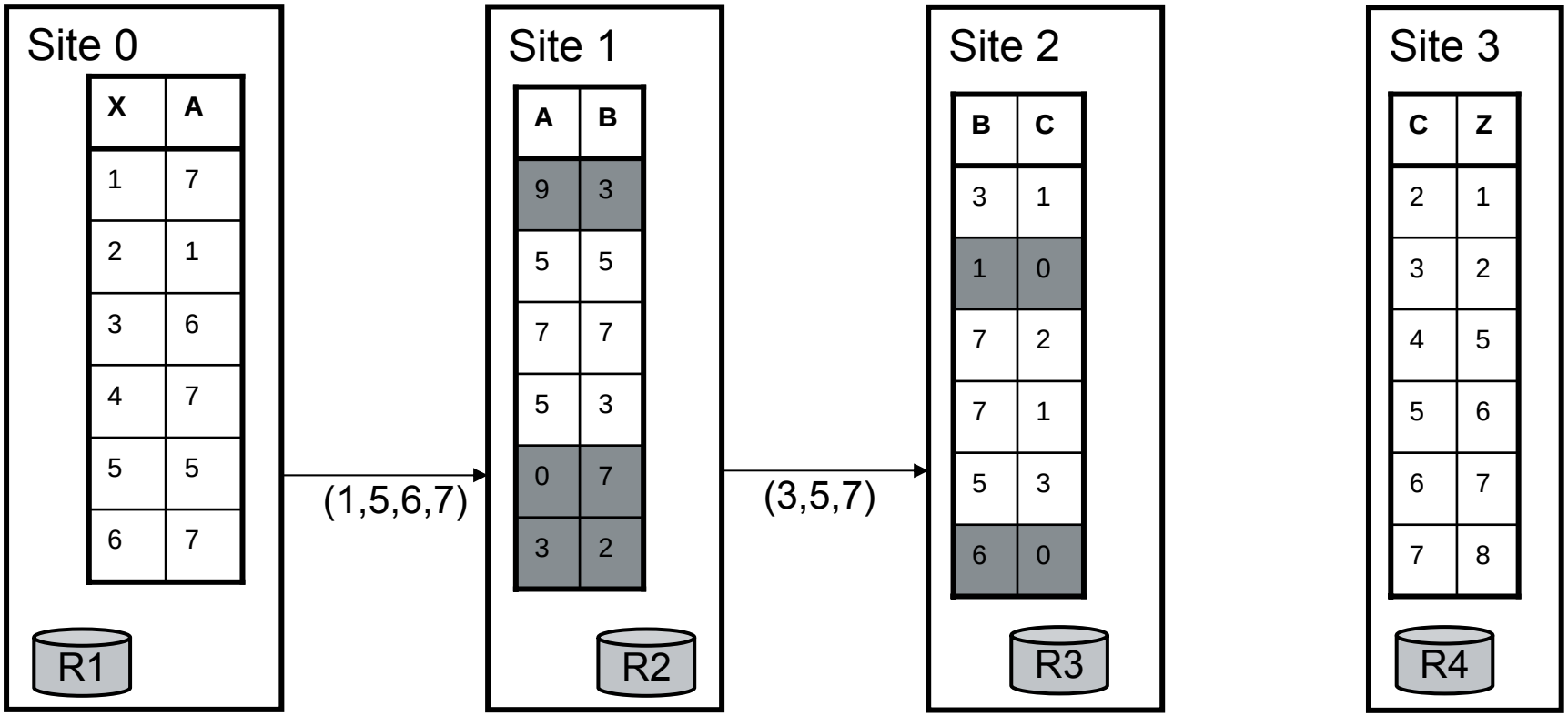
$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce – Beispiel



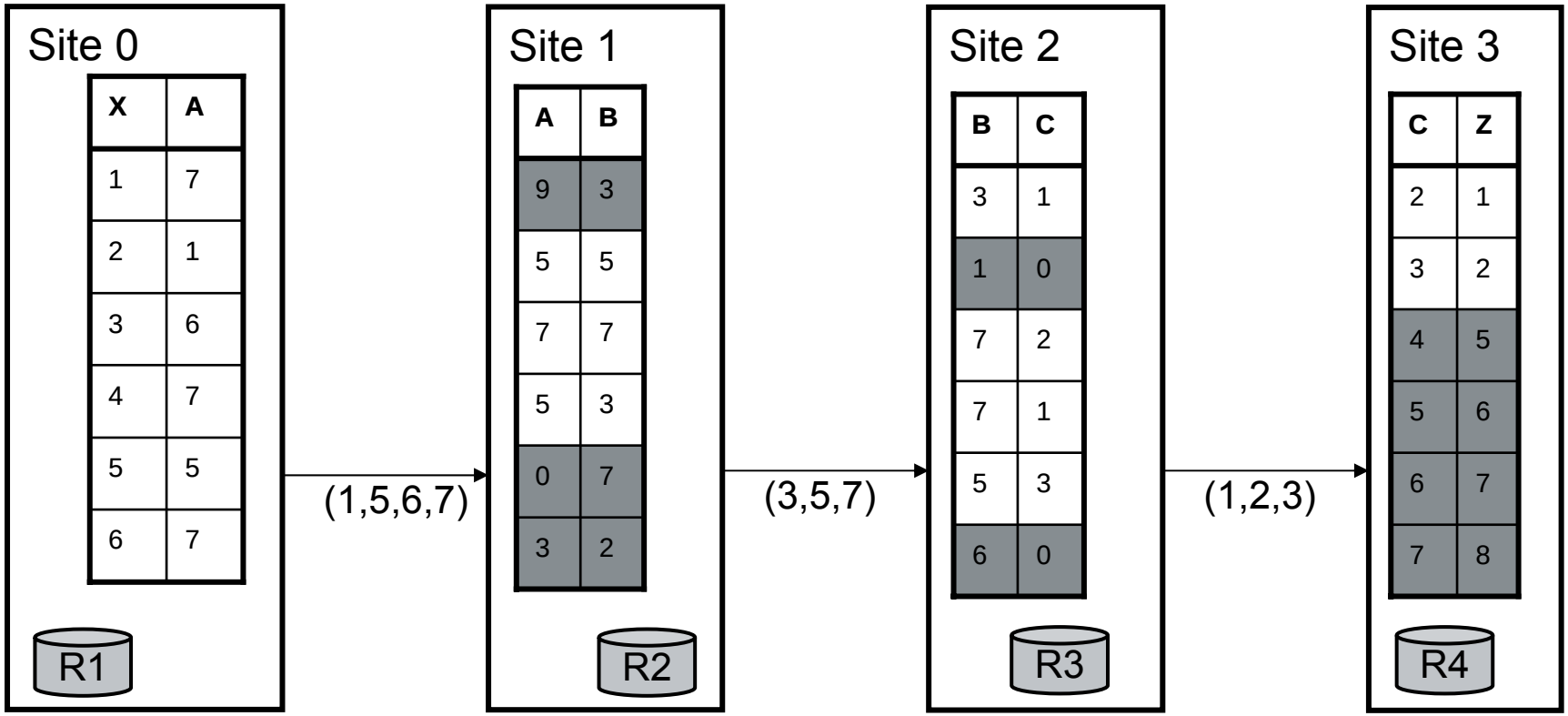
$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce – Beispiel



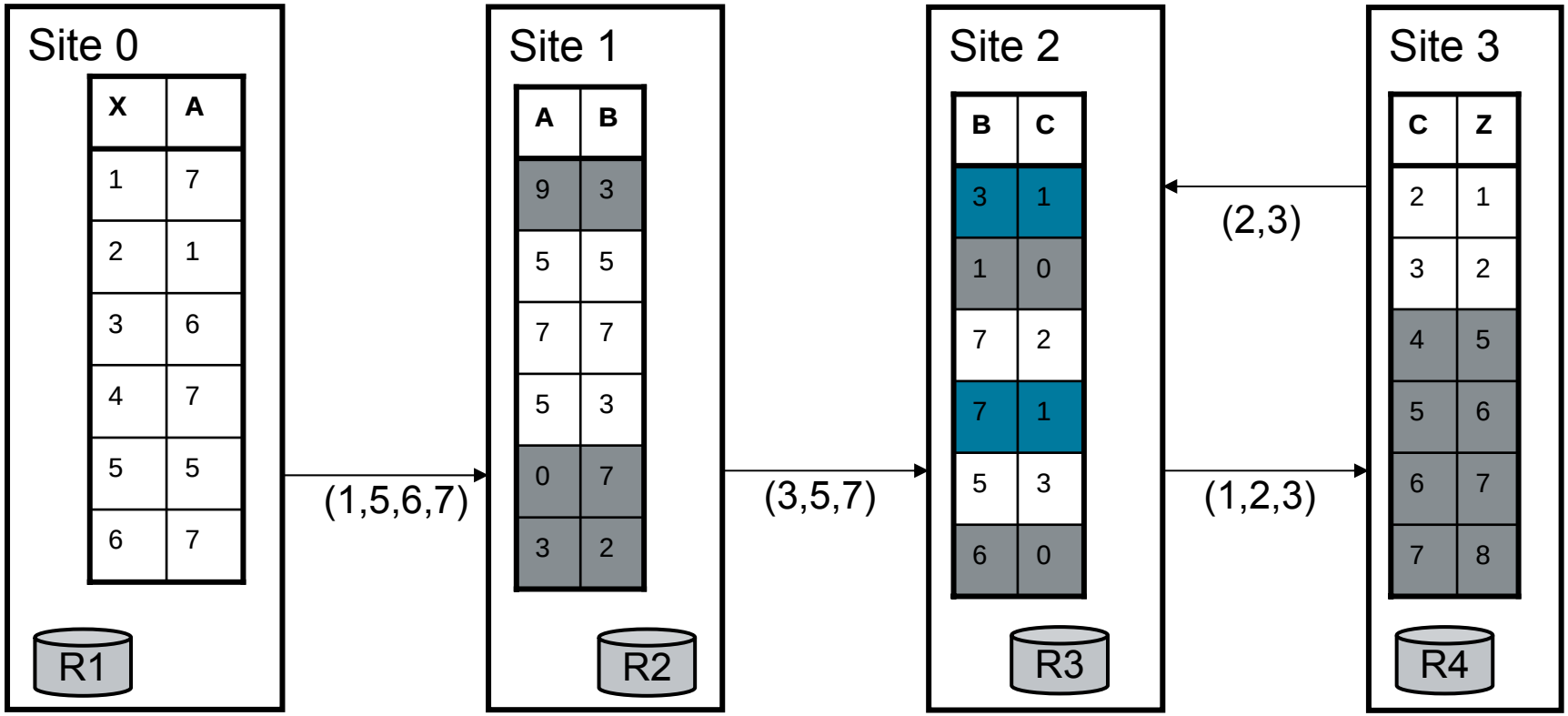
$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce – Beispiel



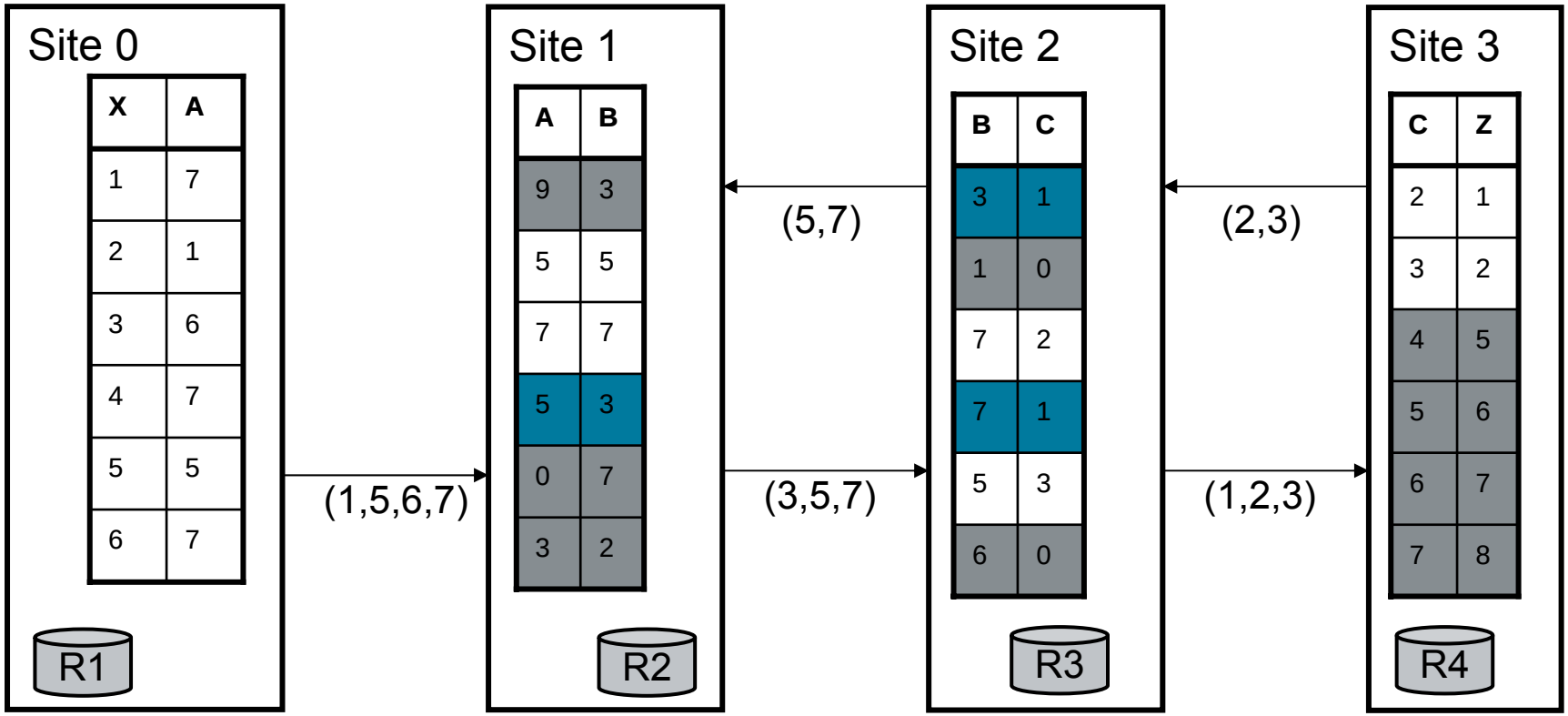
$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce – Beispiel



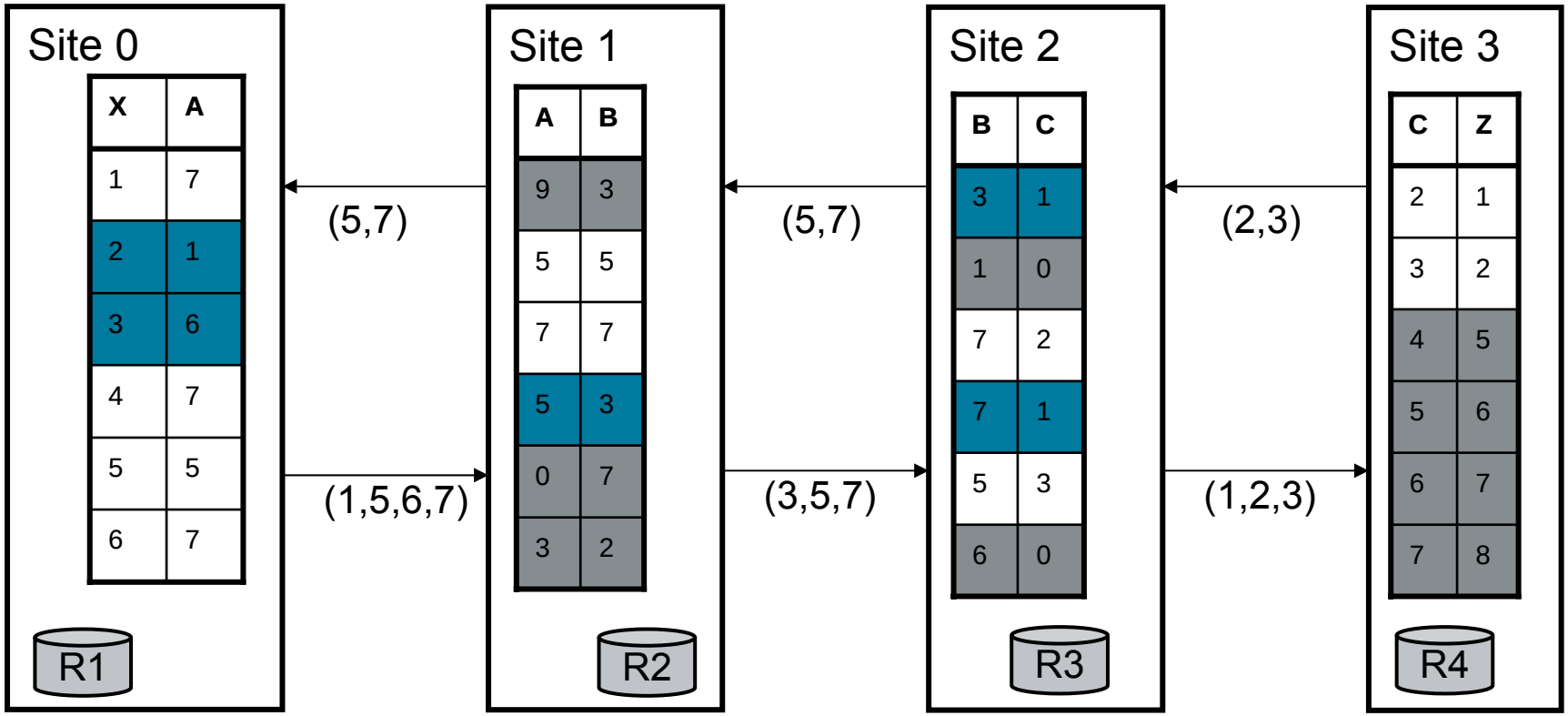
$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce – Beispiel



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

Fully Reduce – Beispiel

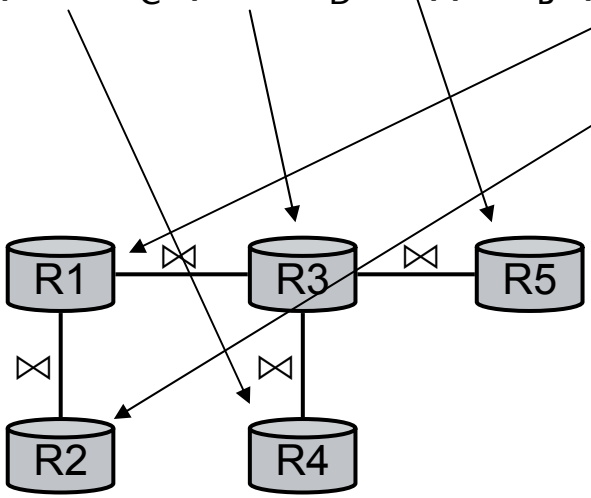


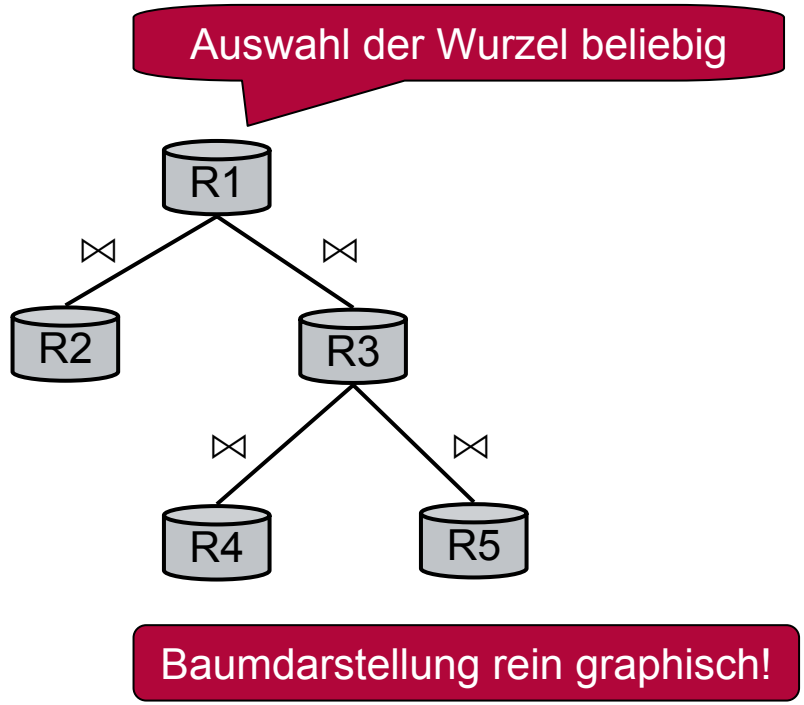
$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce

Gegeben Baum-Anfrage Q

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





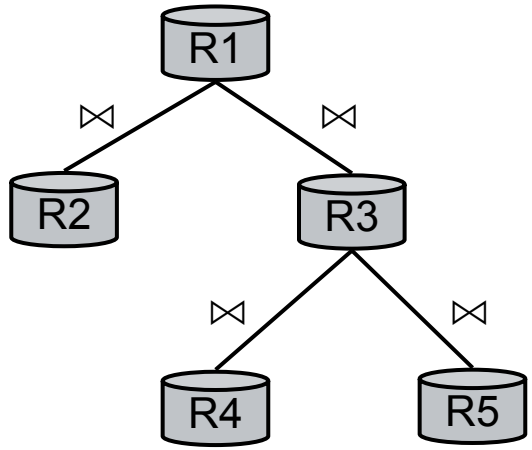
Fully Reduce

Phase 1

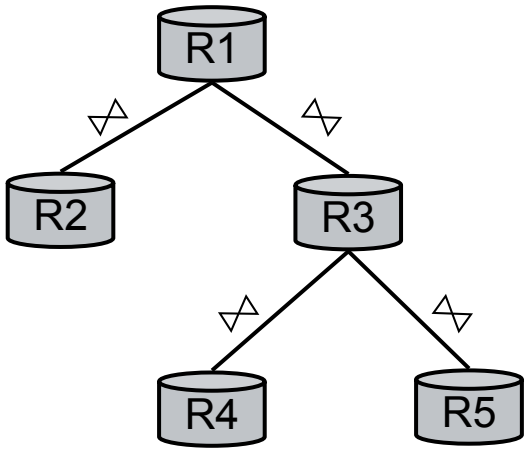
- Von unten nach oben
- Ziel: *Full reducer* für Wurzel
- Einführung von Semi-Joins von Knoten zu ihren Eltern

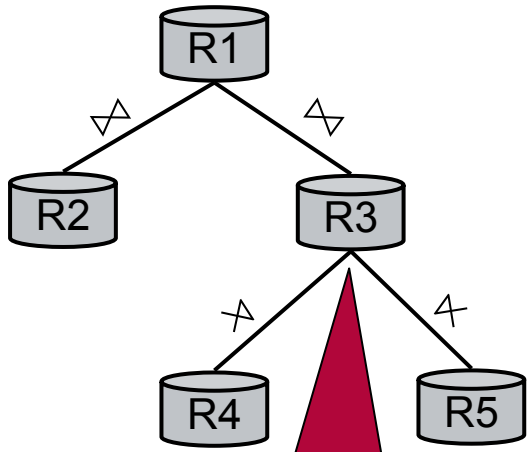
Phase 2

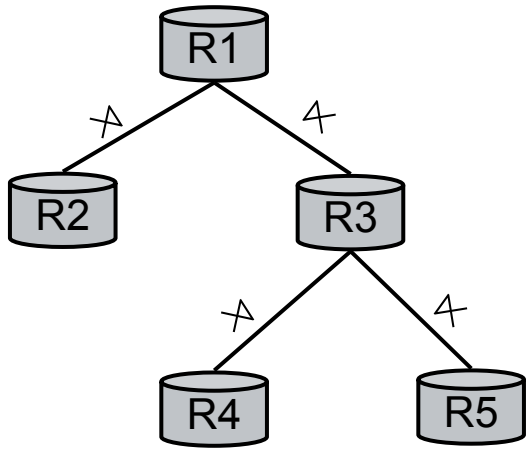
- Von oben nach unten
- Ziel: *Full reducer* für alle anderen Knoten
- Einführung von Semi-Joins von Knoten zu ihren Kindern



Fully Reduce – Phase 1







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

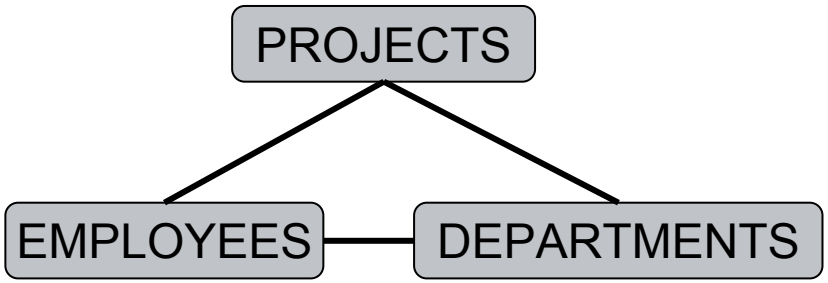
Ergebnis in R1 erfüllt alle Bedingungen, ist also fully reduced.

Optimierung mit Semi-Join

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

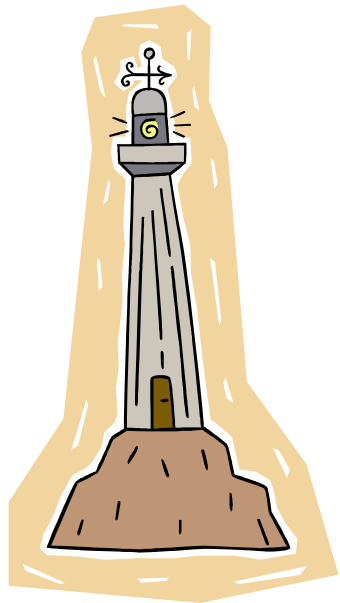
Anfragebearbeitung im Überblick

Techniken der Verteilten Anfragebearbeitung

- Row Blocking
- Multicasts
- Multithreading
- Partitionierung

Joinbearbeitung

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

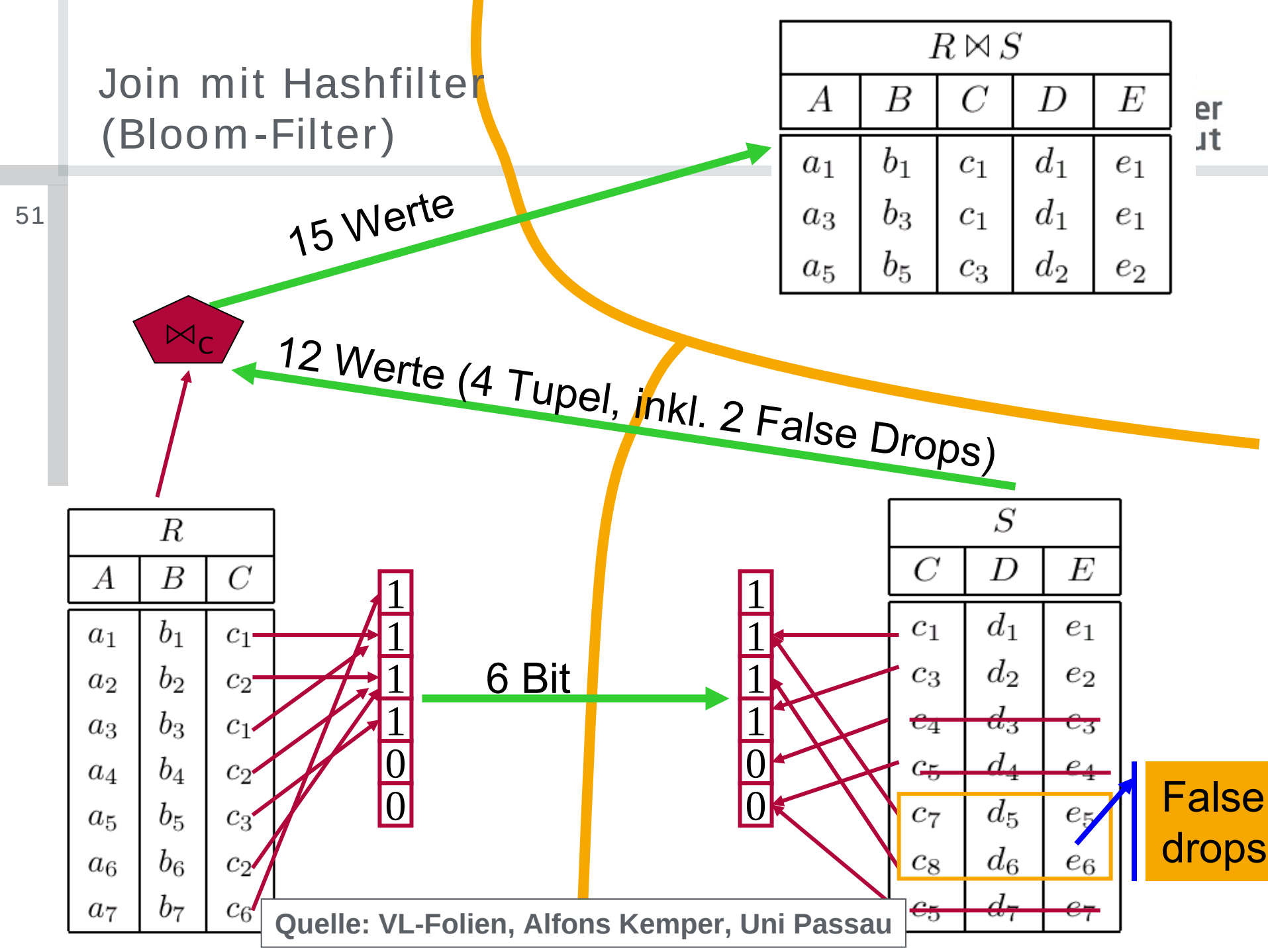


Optimierung mit Semi-Join

Verbesserung durch Hashfilter

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

Join mit Hashfilter (Bloom-Filter)



$R \bowtie S$

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

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

1 1 1 1 0 0

1 1 1 1 0 0

S

C	D	E
c_1	d_1	e_1
c_3	d_2	e_2
~~c_4~~	~~d_3~~	~~e_3~~
~~c_5~~	~~d_4~~	~~e_4~~
c_7	d_5	e_5
c_8	d_6	e_6
~~c_5~~	~~d_7~~	~~e_7~~

Quelle: VL-Folien, Alfons Kemper, Uni Passau

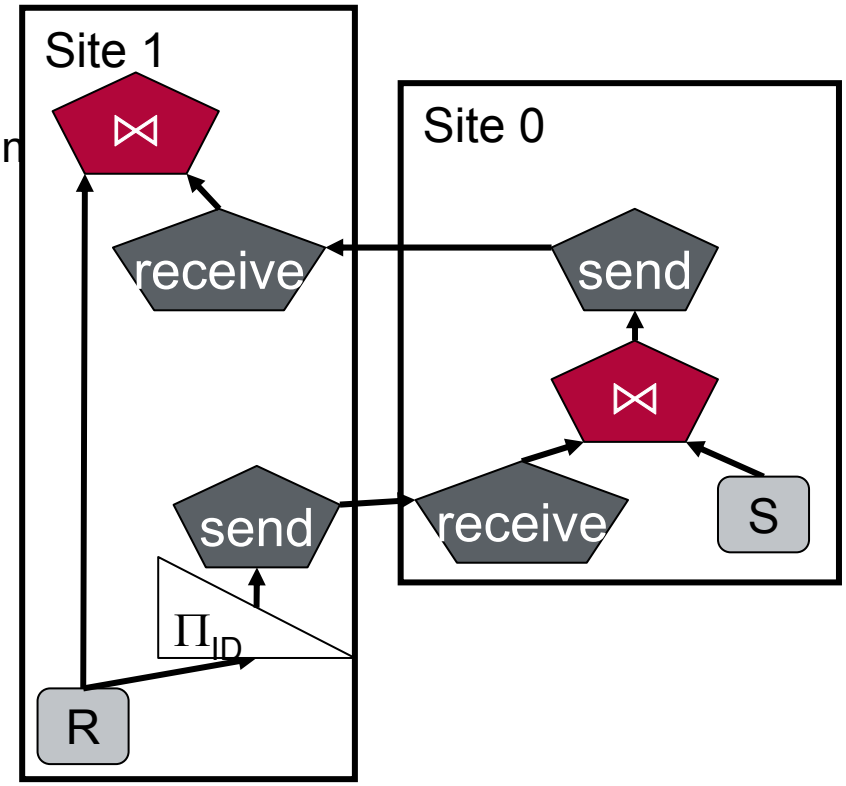
Rückblick

Techniken der verteilten Anfragebearbeitung

- Row Blocking
- Multicasts
- Multithreading
- Partitionierung

Semi-Join

- Basics
- Reduzierung
- Mit Filter



Literatur

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)