



Informationsintegration

Optimierung mit Semi-Joins

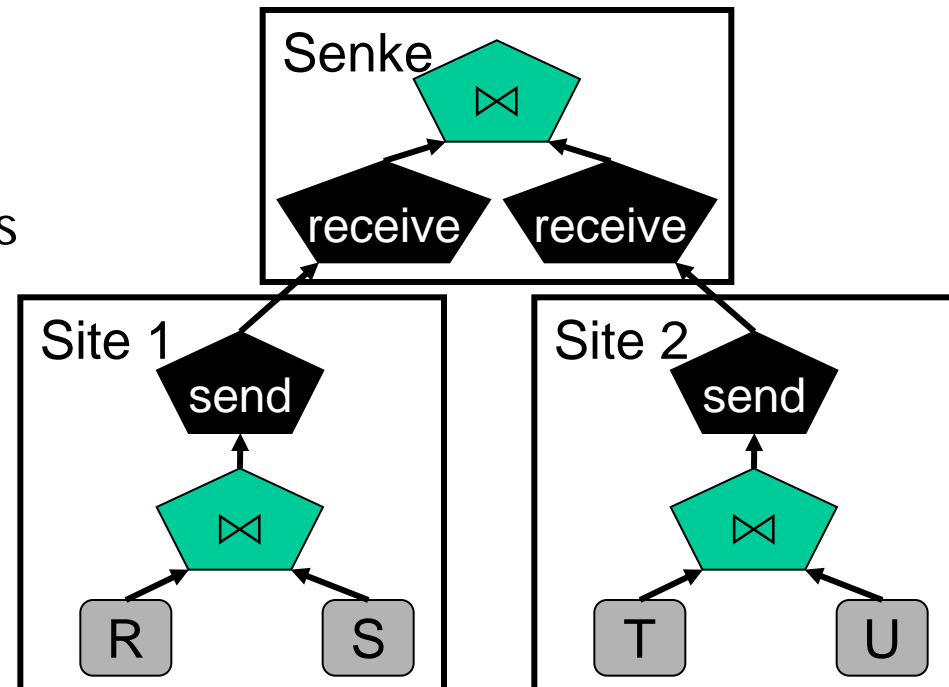
Ulf Leser

Inhalt dieser Vorlesung

- Grundidee des Semi-Joins
- Bloomfilter: Semi-Join Optimierung
- Semi-Joins mit mehreren Relationen: Full Reducer

Erinnerung: Auswertungsstrategien

- Ship whole
 - Vollständige Relationen
 - Wenig Nachrichten, viele Bytes
- Fetch rows as needed
 - Pushen von Selektionen/Joins
 - Bindings für Variable
 - Viele Nachrichten, wenig Bytes
- Fetch columns as needed
 - Semi-Join

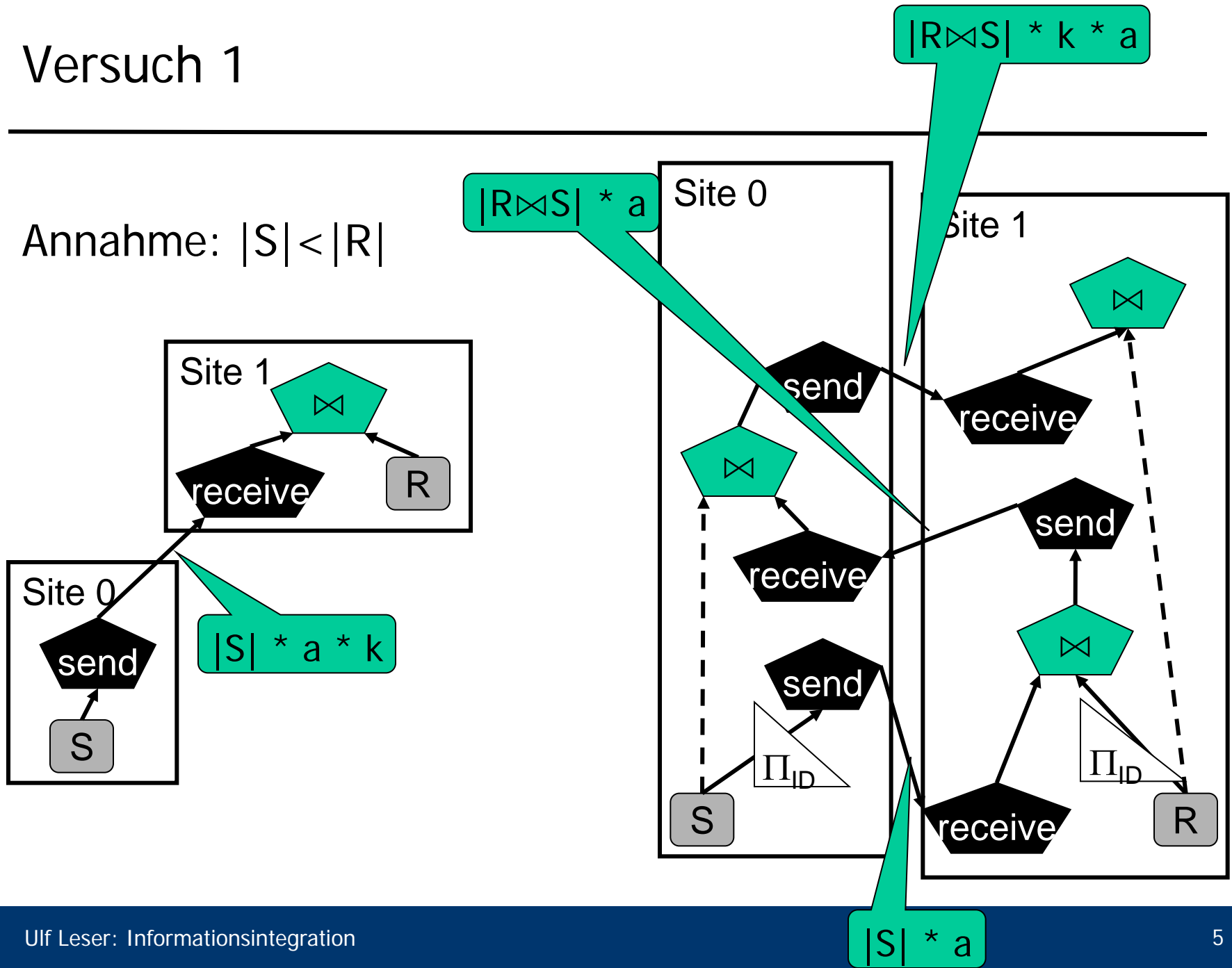


Ausgangslage

- Wir betrachten einen einzelnen Join $R \bowtie S$
 - Jedes Attribut hat Größe a , R/S mit k Attributen, keine Projektionen
- Zunächst **drei mögliche Strategien**
 - Daten von S nach R ; Join in R ausführen
 - Daten von R nach S ; Join in S ausführen
 - Daten von R und S zu drittem Knoten bewegen, Join dort ausrechnen
- Welche Daten bewegen wir?
 - Join kennt zwei Arten von Daten: Joinattribute, andere Attribute
- Semi-Join: Konzentriert sich **erst mal auf die Join(attribute)**

Versuch 1

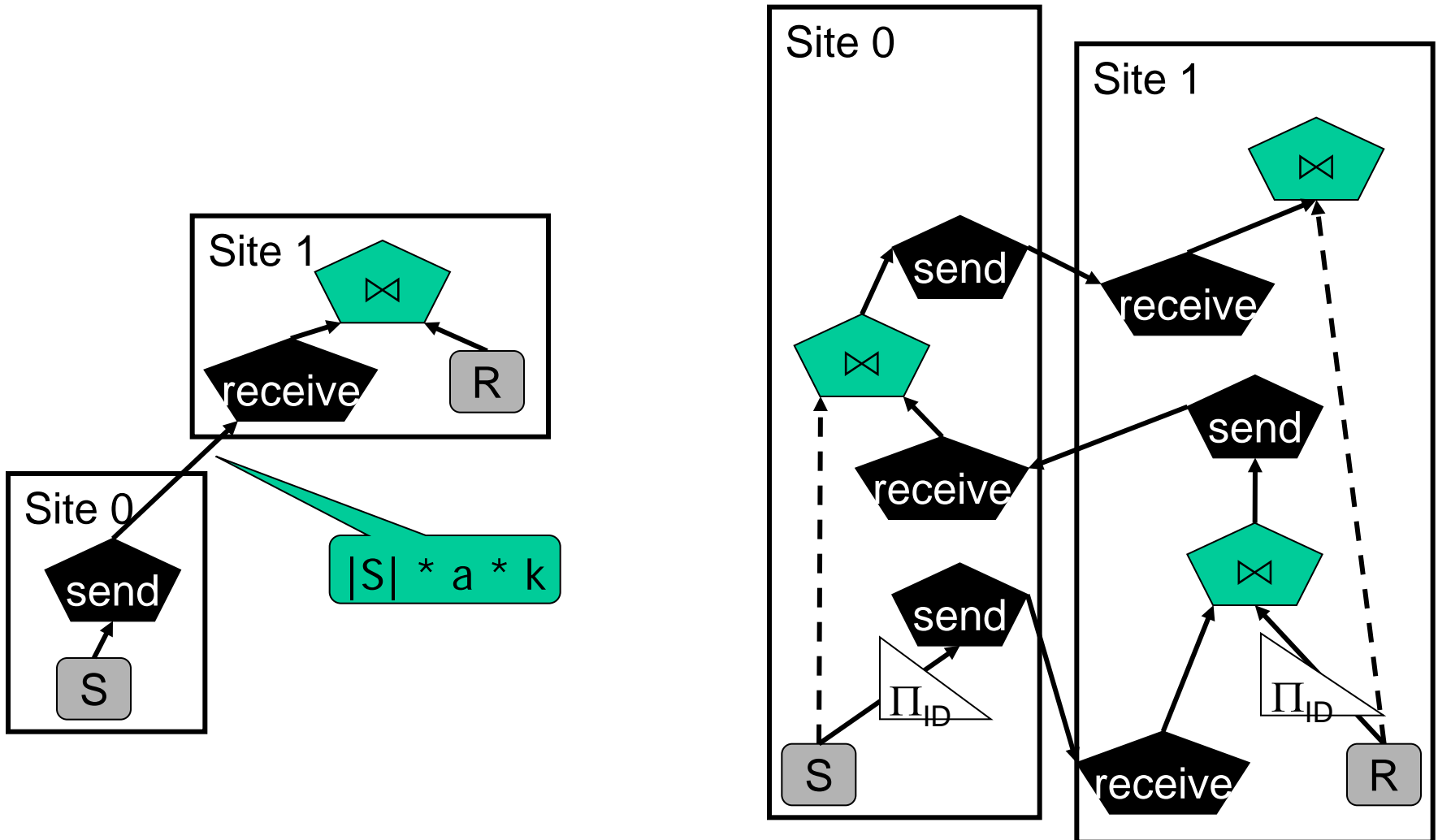
Annahme: $|S| < |R|$



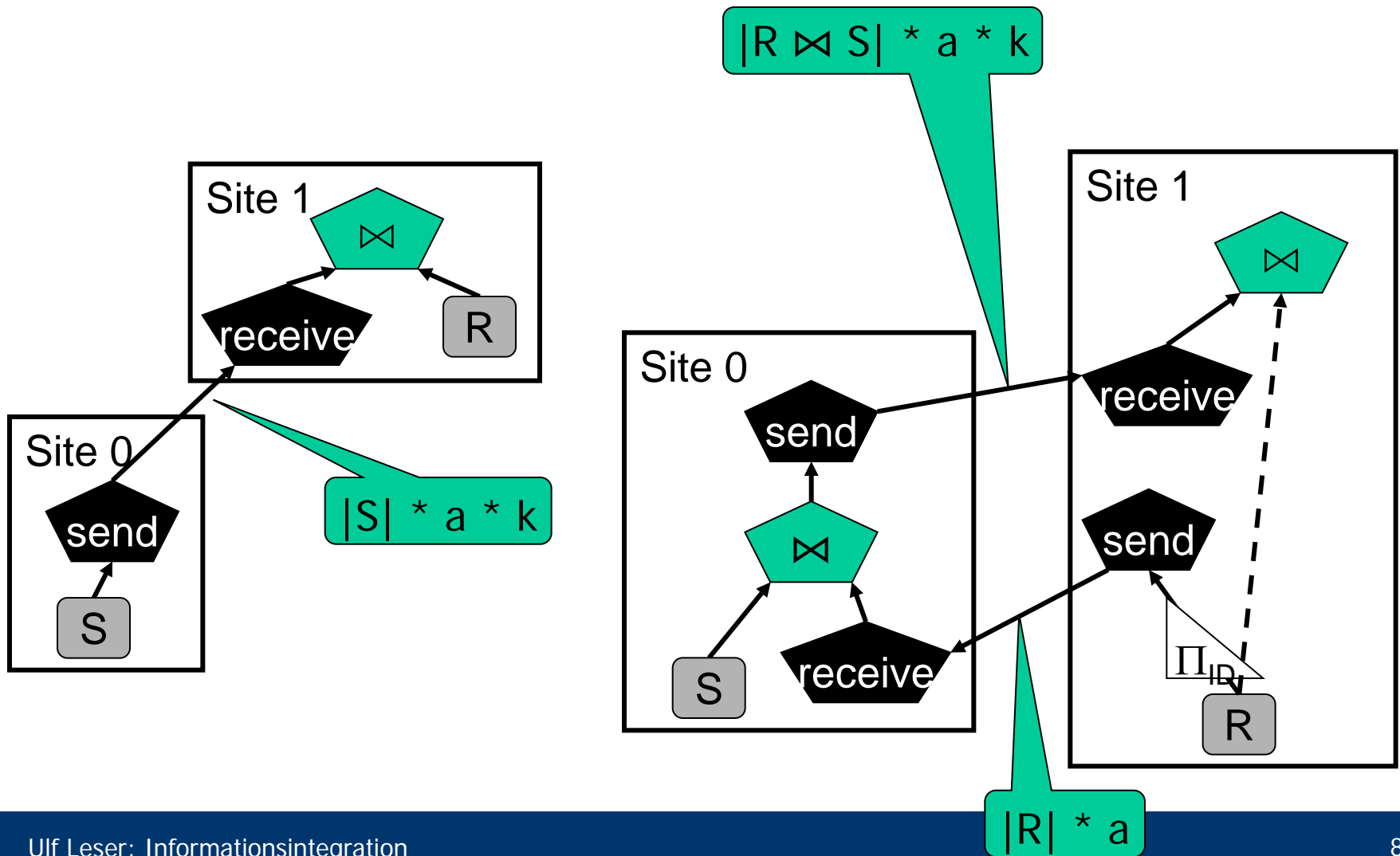
Lohnt sich wann?

- Kosten Plan 1: $|S|^*a^*k$
- Kosten Plan 2: $|S|^*a + |R \bowtie S|^*a + |R \bowtie S|^*a^*k =$
 $|S|^*a + |R \bowtie S|^*a^*(k+1)$
- Wann ist $|S|^*a^*k > |S|^*a + |R \bowtie S|^*a^*(k+1)$
 - Unabhängig von Attributgröße: $|S|^*k > |S| + |R \bowtie S|^*(k+1)$
 - Wenn $|S| \gg |R \bowtie S|$ ist
 - Wenn $|R \bowtie S|$ klein ist
 - Wenn k groß ist (viele Projektionen machen Semi-Join unattraktiv)
- Wenn der **Join hochselektiv** ist und viele nicht-Joinattribute transportiert werden müssen

Geht das nicht (vielleicht) besser?



Semi-Join

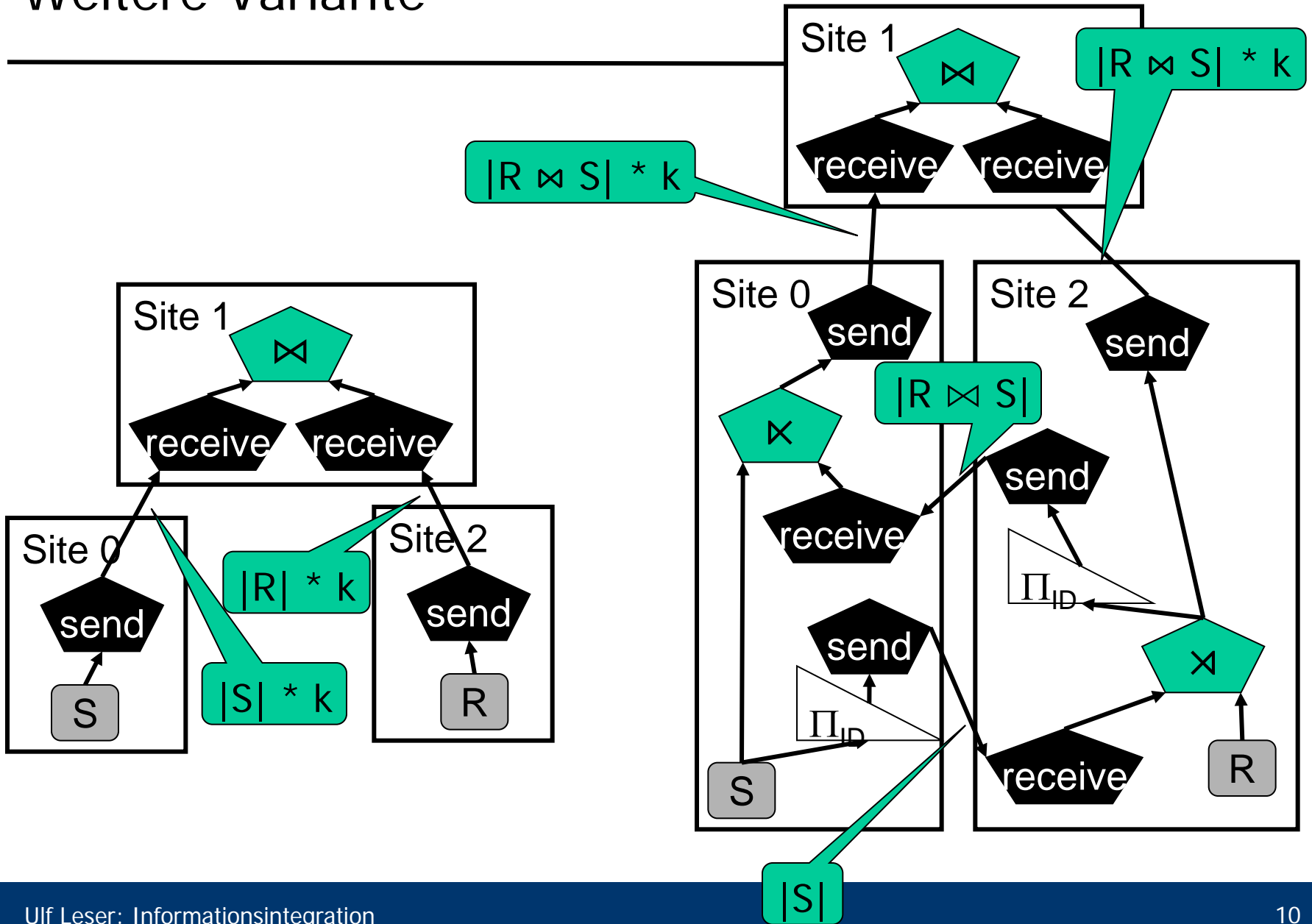


Lohnt sich wann?

- Kosten Plan 1: $|S|^k$
- Kosten Plan 2: $|S| + |R \bowtie S|^k (k+1)$
- Kosten Plan 3: $|R| + |R \bowtie S|^k$

- 2 und 3 besser als 1, wenn **kleines Joinergebnis** und k groß
- 3 besser als 2, wenn $|R| < |S|$
 - Die **kleinere Quelle initiiert** und berechnet den Join
 - $|R \bowtie S|$ ist immer gleich groß, egal welche Quelle kleiner ist

Weitere Variante



Definition Semi-Join

- Definition

*Gegeben Relationen R mit Attributmengemenge A und S mit Attributmengemenge B . Der **Semi-Join** $R \bowtie S$ ist definiert als*

$$\begin{aligned} R \bowtie S &:= \Pi_A(R \bowtie_{A \cap B} S) \\ &= \Pi_A(R) \bowtie_{A \cap B} \Pi_{A \cap B}(S) \\ &= R \bowtie_{A \cap B} \Pi_{A \cap B}(S) \end{aligned}$$

- Bemerkungen

- Bei Join zwischen $R.X$ und $S.Y$ gilt: $R \bowtie S := R \bowtie_{X=Y} \Pi_Y(S)$
- S wirkt als **Filter auf den Tupeln** von R
- Semi-Join ist **asymmetrisch**

Transformationsregeln

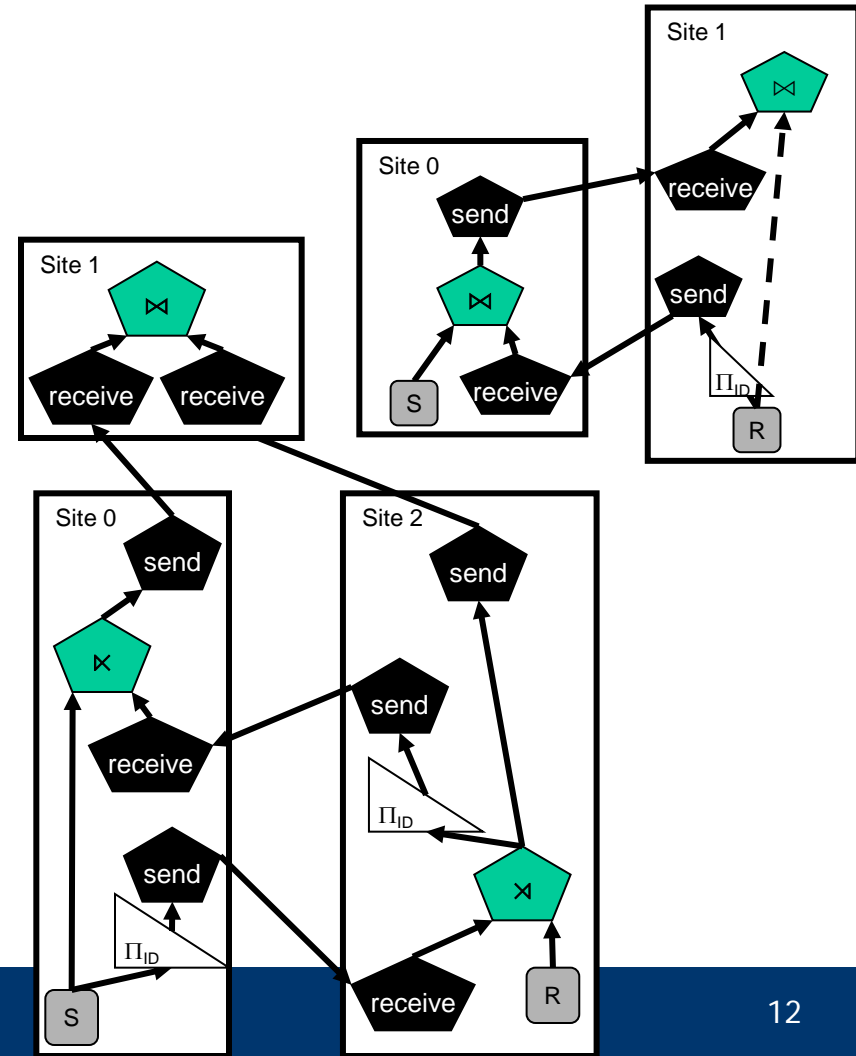
- Semi-Joins können auf verschiedene Arten zur Optimierung von Joins eingesetzt werden

- Äquivalenzumformungen

$$R \bowtie_F S =$$

- $(R \bowtie_F S) \bowtie_F S$
 - R verkleinern, dann Join mit S
- $R \bowtie_F (S \bowtie_F R)$
 - S verkleinern, dann Join mit R
- $(R \bowtie_F S) \bowtie_F (S \bowtie_F R)$
 - R und S verkleinern, dann Join

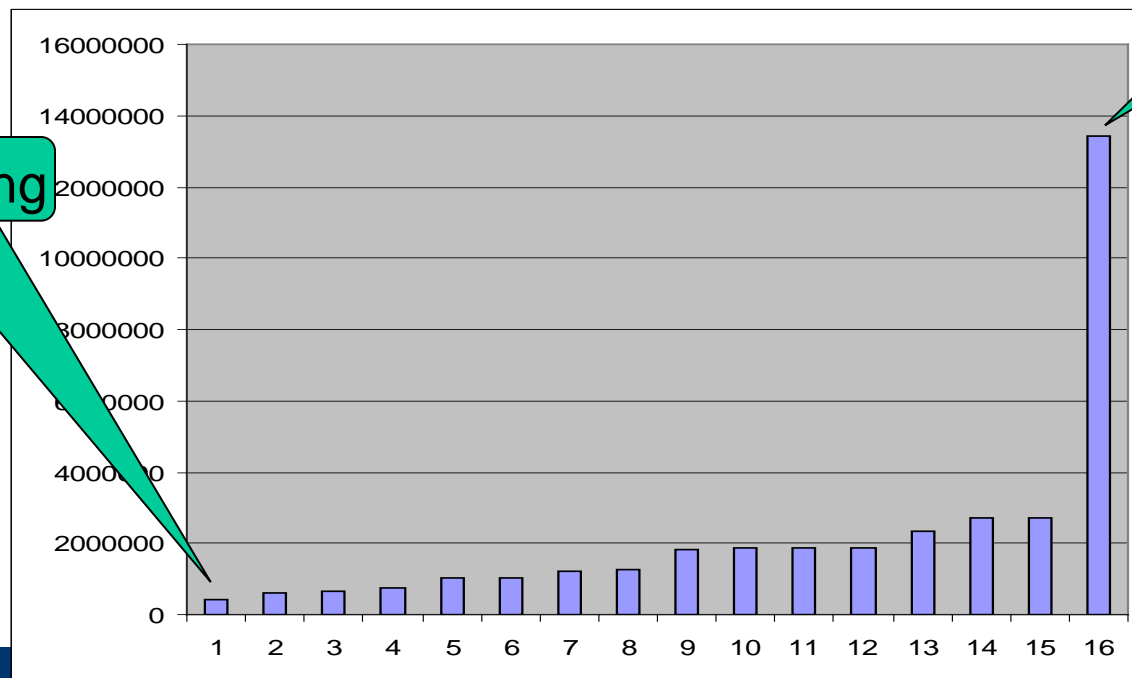
- Entspricht welchen verteilten Plänen?



Aus einer Übung

- Titel und Regisseur aller Filme, die jünger als 1980 sind

```
SELECT      F1.Titel, F2.Regie
FROM        Movie1.Film1 F1, Movie2.Filme2 F2
WHERE       F1.Titel = F2.Titel
            AND F1.Jahr > 1980
```
- Anzahl übertragener Bytes



Tricks

- Nur notwendige Bytes übertragen: Rtrim()
- Richtige Reihenfolge: Filme2 ist kleiner
- Projektionen wo immer möglich
- Kompression
- **Duplikate** müssen nicht übertragen werden
 - Semi-Join gemischt mit DISTINCT
- **Bloomfilter**

Inhalt dieser Vorlesung

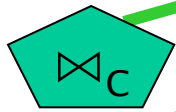
- Grundidee des Semi-Joins
- Bloomfilter: Semi-Join Optimierung
- Semi-Joins mit mehreren Relationen: Full Reducer

Bloomfilter

- Effiziente **Implementierung** von (Semi-)Joins
- Beobachtung
 - Semi-Joins lohnen sich, wenn man die **Join-Selektivität** als sehr hoch einschätzt
 - Also werden nur sehr wenige Werte einen Joinpartner finden
 - Warum dann alle übertragen?
- Idee von Bloomfiltern (Hashfilter) für $R \bowtie_F S$
 - **Hashe** alle Werte $R.F$ mit Hashfunktion h in Hasharray H
 - Übertrage H nach S
 - $\forall f \in S.F$ mit $H(h(f))=0$ gilt: f hat keinen Join-Partner in R
 - $\forall f \in S.F$ mit $H(h(f))=1$ gilt: f hat **vielleicht** einen Join-Partner in R

Beispiel

$R \bowtie S$				
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



15 Werte

12 Werte (4 Tupel, davon 2 falsch positiv)

Hashfunktion
n: mod 6

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

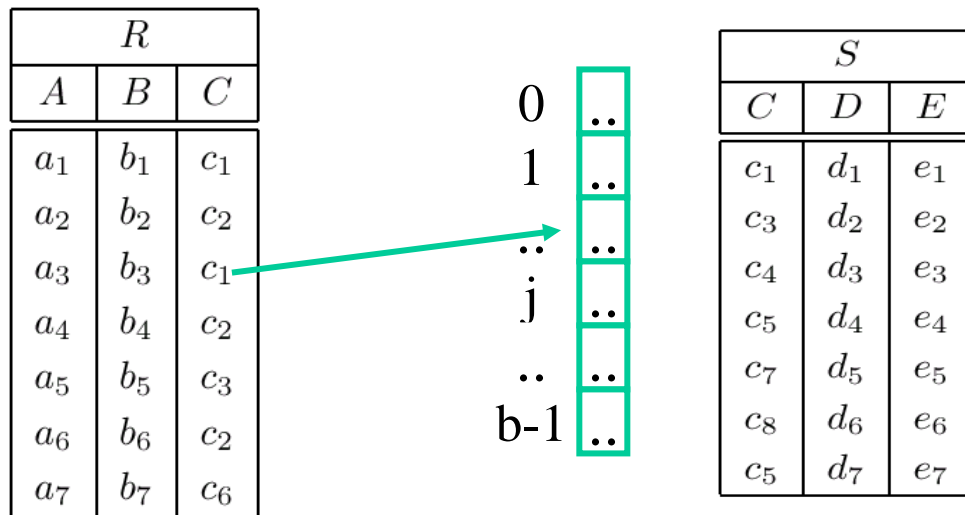


S		
C	D	E
c_1	d_1	e_1
c_3	d_2	e_2
c_4	d_3	e_3
c_5	d_4	e_4
c_7	d_5	e_5
c_8	d_6	e_6
c_5	d_7	e_7

Falsch
Positiv

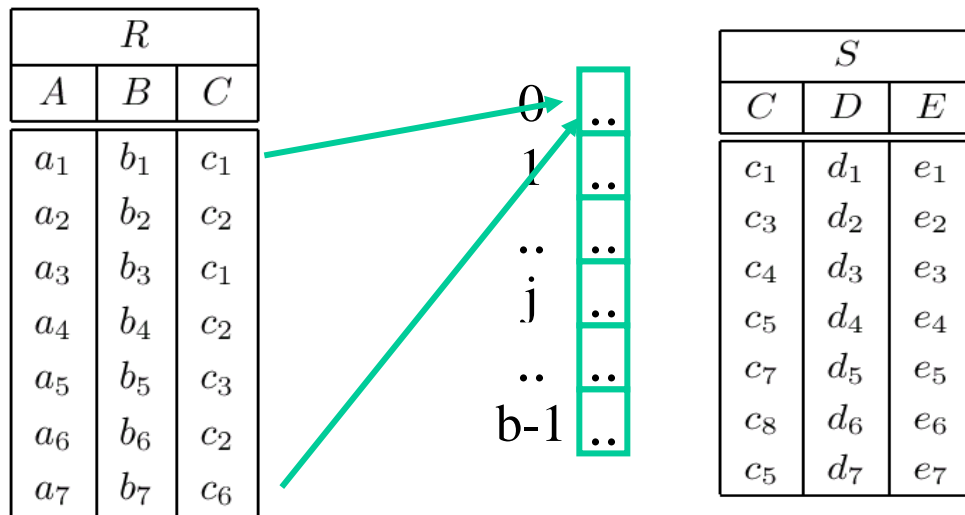
Wsk für eine 1

- Wsk, dass ein bestimmtes Bit gesetzt ist
 - Sei $b = |H|$; wir nehmen **Gleichverteilung** an
 - Wsk, dass ein bestimmtes $f \in R$ das Bit setzt: $1/b$
 - Wsk, dass kein $f \in R$ das Bit setzt: $(1 - 1/b)^{|R|}$
 - Wsk, dass **irgendein $f \in R$ ein bestimmtes Bit** setzt: $1 - (1 - 1/b)^{|R|}$



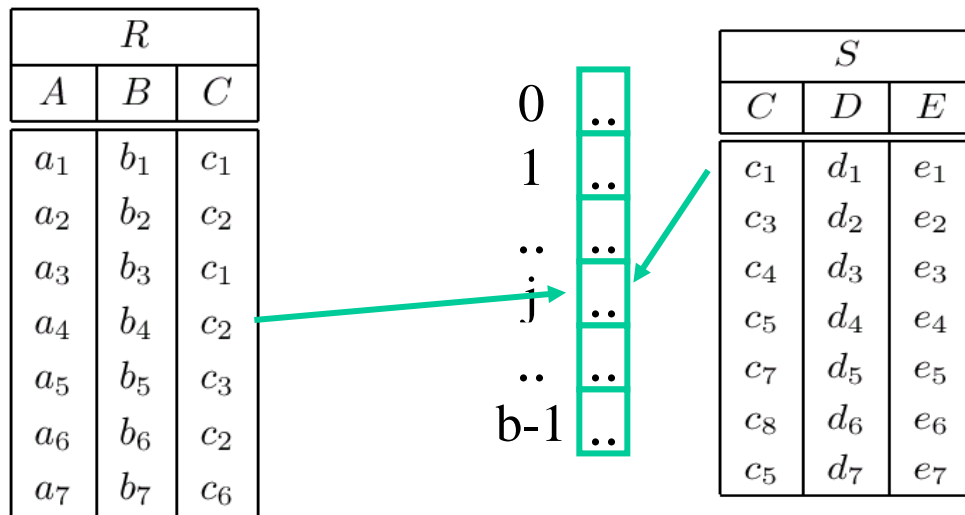
Anzahl von 1'ern

- Wie viele Bits erwarten wir als gesetzt? $b \cdot [1 - (1 - 1/b)^{|R|}]$
- Vorsicht: Mehrere $f \in R$ können dasselbe Bit setzen
- Approximation: Wir nehmen (konservativ) an, dass **alle $f \in R$ unterschiedliche Bits** setzen
 - Nur, wenn $b \gg |R|$
 - Damit: Wsk, dass ein bestimmtes Bit j gesetzt ist: $|R|/b$



Auswirkung auf S

- Wsk, dass von einer Position j im Hasharray ein **bestimmtes $f \in S$ ausgewählt** wird: $1 - (1 - 1/b)^{|R|}$
- Wie viele $f \in S$ werden ausgewählt? $|S| * [1 - (1 - 1/b)^{|R|}]$
- Approximation (alle f setzen unterschiedliche Bits)
 - Wsk, dass ein bestimmtes Bit j gesetzt ist: $|R|/b$
 - $|S| * (|R|/b)$ Elemente aus S werden ausgewählt



Beispiel

- $|R| = |S| = 1000$, $b = 4000$, $a = 15$
 - Übertragung aller Schlüssel: 15.000 Byte
 - Übertragung von H: $4000/8 = 500$ Byte
- Erwartete **Anzahl ausgewählter Werte** in S
 - $|S| * [1 - (1 - 1/b)^{|R|}] = 1000 * [1 - (1 - 1/4000)^{1000}] \sim 221$
 - Approximation: $|S| * (|R|/b) = 250$
- Nur in ca. 250 Bits erwartet man in beiden Arrays eine „1“
 - Wenn Werte durch Hashfunktion gleichverteilt werden und R und S unabhängig voneinander sind
- Nur wenige der „1“ sind falsch Positive
 - Kann man genau ausrechnen

Trade-Off

- Je größer b
 - Desto breiter wird R über S gestreut
 - Desto **mehr Byte müssen im Filterschritt** übertragen werden
 - Desto weniger Tupel aus S finden eine 1 in H
 - Desto weniger Tupel aus S finden fälschlicherweise eine 1 in H
 - Desto **weniger Tupel müssen an R zurückgeschickt** werden

Weitere Einsatzmöglichkeiten

- Signatur-Files (~ Indexe für die Filterung von Daten)
- Beim „normalen“ Hash-Join
- Für Star-Joins in Data Warehouses
- Bloomfilter: Immer, wenn
 - ... Mengen verglichen werden und
 - ... man erwartet, dass nur wenige Elemente Treffer sind und
 - ... Datenübertragung teuer ist

Inhalt dieser Vorlesung

- Grundidee des Semi-Joins
- Bloomfilter: Semi-Join Optimierung
- Semi-Joins mit mehreren Relationen: Full Reducer

Semi-Joins mit mehr als einem Join

$$\begin{array}{lll} R & \bowtie_F S & \bowtie_G T = \\ (R \bowtie_F S) & \bowtie_F (S \bowtie_G T) & \bowtie_G T = \\ (R \bowtie_F (S \bowtie_G T)) & \bowtie_F (S \bowtie_G T) & \bowtie_G T = \\ & \dots & \end{array}$$

- Jeder Semi-Join reduziert die Zahl von Tupeln einer Relation, die übertragen werden müssen
 - Man nennt **Semi-Joins** daher auch „**Reducer**“
- Eine Relation heisst „**reduced**“, wenn sie keine Tupel mehr enthält, die nicht im **Gesamtergebnis** gebraucht werden
 - Globale Eigenschaft – auch weit entfernte Joins beeinflussen die „notwendigen“ Tupel einer Relation

Formaler

- Definition

*Seien R_1, \dots, R_n Relationen. Ein **Semi-Join Programm** ist eine Folge von Semi-Joins der Art*

$$R_i := R_i \bowtie R_j$$

- Bemerkung

- Über welche Attribute gejoined wird, interessiert uns nicht mehr
- Die Wirkung jedes Semi-Joins ist eine Reduzierung der Tupel in R_i
- Gemeint ist nur eine **temporäre** Änderung von R_i
 - Es wird also eigentlich ein R_i' produziert

Full Reducer

- Definition

Sei $Q=R_1 \bowtie \dots \bowtie R_n$ eine relationale Anfrage:

- Ein *Reducer* für eine Relation R_i in Q ist ein Semi-Join Programm, das aus R_i alle Tupel entfernt, die nicht zur Berechnung von $result(Q)$ benötigt werden
- Ein *Full Reducer* für Q ist ein Semi-Join Programm, das ein Reducer für alle R_i in Q ist

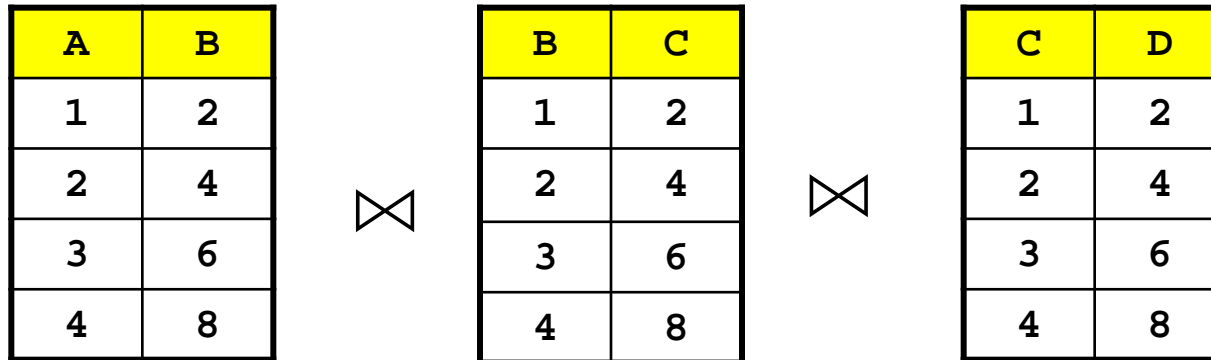
- Bemerkung

- Die R_i müssen nicht unterschiedlich sein
- Reducer für eine Relation – *Full Reducer für eine Query*

Full Reducer und verteilte Anfragen

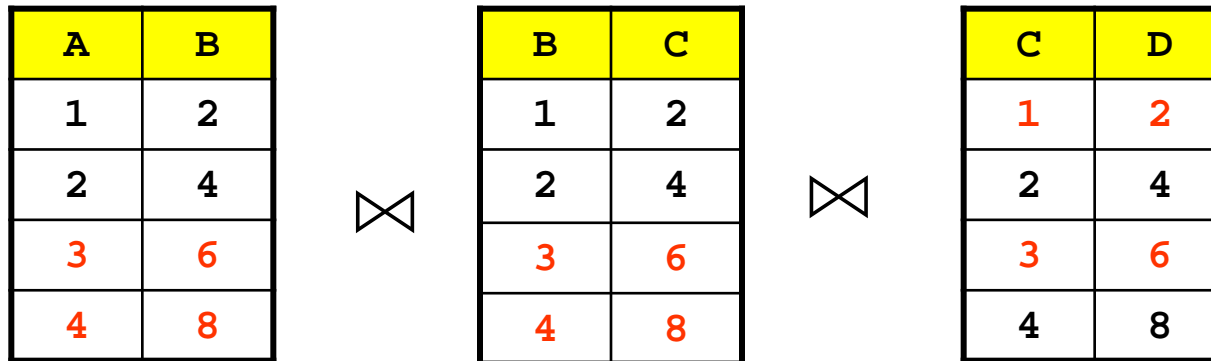
- Ein Full Reducer entspricht einem Plan zur Abarbeitung einer verteilten Anfrage
- Der überträgt wenig (am wenigsten) Tupel für **Joins, die „hoch“ im Ausführungsplan** sind (also spät berechnet werden)
- Dafür werden Joins öfters ausgeführt – viel größerer Suchraum
- Ist **nicht optimal** im Sinne der insgesamt kleinsten übertragenen Datenmenge
 - Denn zur Reduktion müssen ja Tupel übertragen werden
- Wie schwer ist es, einen Full Reducer zu finden?

Beispiel



- Viele Tupel sind „dangling“
 - Z.B. 2-4, 4-8, 8-? oder ?-?, ?-1, 1-2
- Beispiel für ein Semi-Join Programm:
 - $AB := AB \bowtie BC$ entfernt aus AB: (3,6) und (4,8)
 - $BC := BC \bowtie CD$ entfernt aus BC: (3,6) und (4,8)
 - $CD := CD \bowtie BC$ entfernt aus CD: (1,2) und (3,6)

Beispiel



- Ist das ein Full Reducer?

- Nein: in AB ist (2,4) überflüssig; ebenso BC: (1,2) und CD: (2,4)
- Um das zu sehen, muss man reduzierte Relationen erneut benutzen und erneut reduzieren

- Full Reducer?

- $BC := BC \bowtie AB$ entfernt aus BC: (1,2) und (3,6)
- $CD := CD \bowtie BC$ entfernt aus CD: (1,2) und (2,4) und (3,6)
- $BC := BC \bowtie CD$ entfernt aus BC: (4,8)
- $AB := AB \bowtie BC$ entfernt aus AB: (2,4) und (3,6) und (4,8)

Etwas Theorie

- Die Schwere des Problems „Finde einen Full Reducer für eine gegebene Query Q “ hängt von der Art von Q ab
- Definition
*Sei $Q=R_1 \bowtie \dots \bowtie R_n$. Der **Hypergraph** von Q wird wie folgt konstruiert*
 - *Jedes Attribut in Q wird ein Knoten*
 - *Verschmelze (transitiv) alle Knoten, die über einen Equi-Join verbunden sind*
 - *Für jede R_i füge eine Hyperkante ein, die alle Attribute von R_i verbindet*
- Bemerkung
 - Wir zeigen **Hyperkanten als Mengen**

Beispiel

- Schema

- `Books(title, author, publisher, ISBN)`
- `Publisher(publisher, paddr, pcity)`
- `Borrower(name, baddr, bcity, ID)`
- `Loan(ID, ISBN, date)`

- Query (berechnet was?)

```
- SELECT *  
FROM books, publisher, borrower, loan  
WHERE books.publisher = publisher.publisher AND  
books.ISBN = loan.ISBN AND  
borrower.ID = loan.ID AND  
borrower.bcity = publisher.pcity
```

- Hypergraph ...

- Verschmelzung von Joinattributen ist durch die Namensgleichheit der Attribute schon fast erledigt

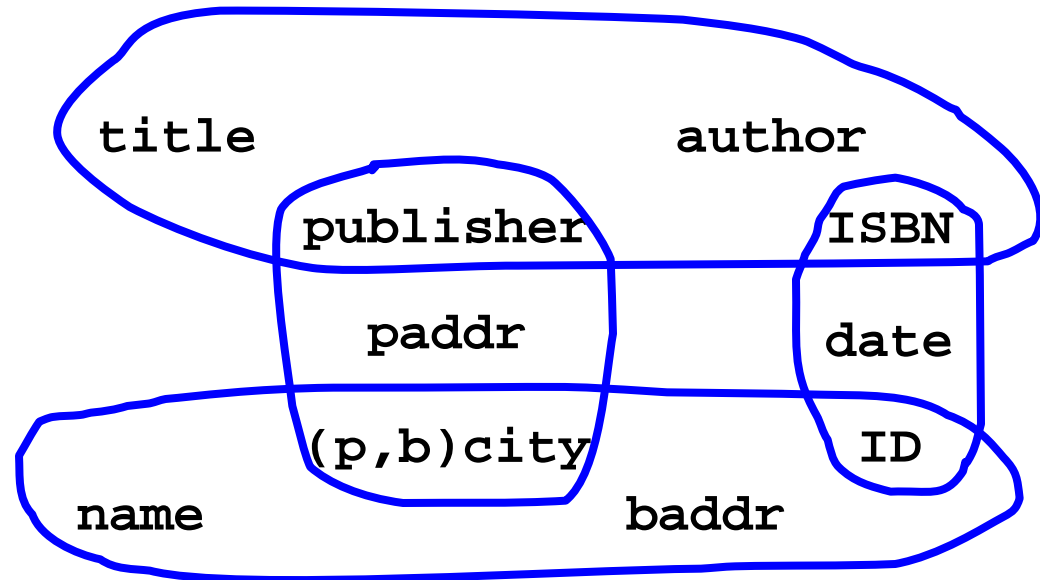
Hypergraph

Title, publisher, author, ISBN

Publisher, paddr, pcity

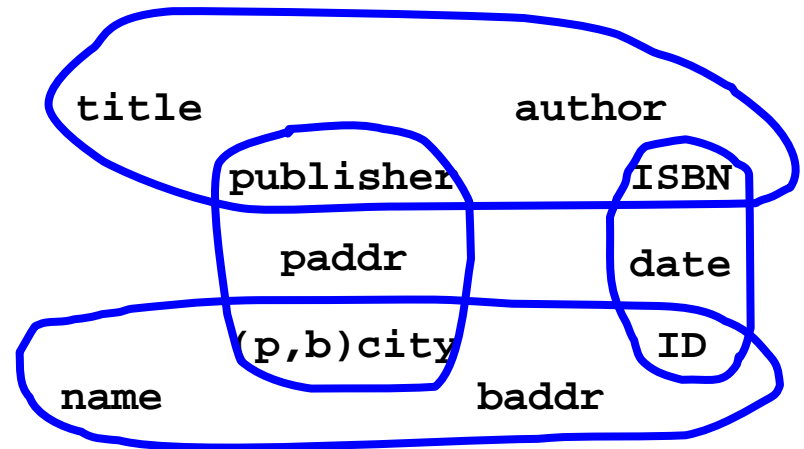
Name, baddr, bcity, ID

ID, ISBN, city



GYO-Reduktion (Graham, Yu and Ozsoyoglu)

- **GYO Reduktion** eines Hypergraphen H
 - Sei E eine Hyperkante aus H . Wir **nennen E ein Ohr**, wenn
 - es kein Attribut mit einer anderen Hyperkante gemeinsam hat oder
 - es eine Hyperkante F gibt, so dass die Attribute in $E-F$ in keiner anderen Hyperkante als E enthalten sind
- Ohren kann man abschneiden
 - Denn sie werden nur an einer Stelle gehalten
- Abschneiden heißt
 - Entfernen von E
 - Entfernen aller Knoten aus $E-F$
- Keine Ohren:



Azyklische Hypergraphen

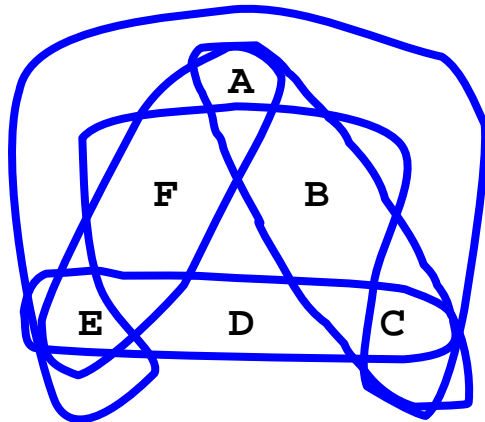
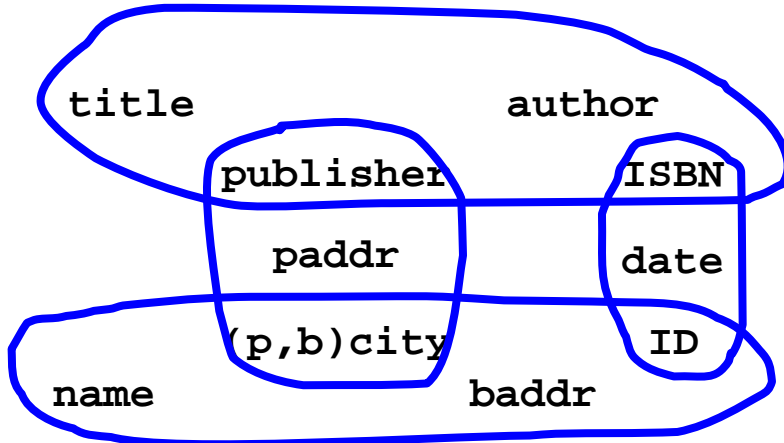
- Definition

*Ein Hypergraph H ist **azyklisch**, wenn ein wiederholtes Entfernen aller Ohren so lange, bis es kein Ohr mehr gibt, den **leeren Graphen** erzeugt.*

- Bemerkung

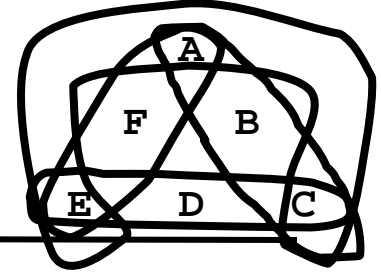
- Wir können uns beim Entfernen von Ohren nicht verlaufen – die **Reduktion ist eindeutig**

Beispiel

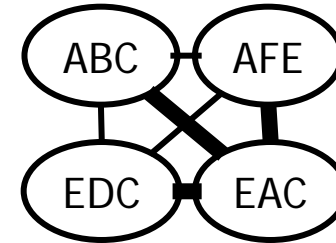


- Ist zyklisch
- Ist?
 - $(ABC)-(ACE)=(B)$ und B ist nur in (ABC) – also (ABC) entfernen
 - $(AEF)-(ACE)=(F)$... also (AEF) entfernen
 - $(EDC)-(ACE)=(D)$... also (EDC) entfernen
 - Nun auch (ACE) entfernen

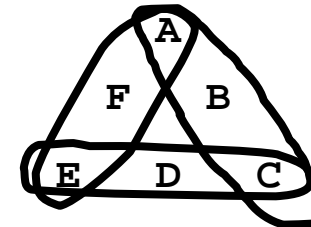
Join-Graphen



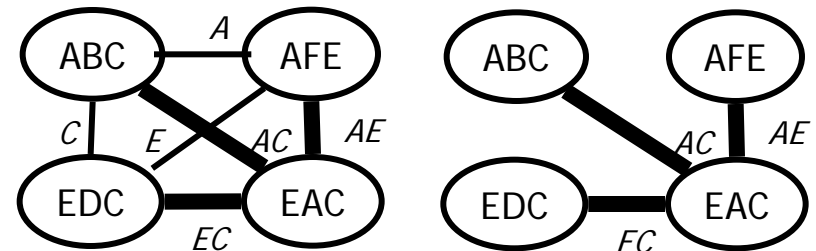
- Die Definition von azyklisch ist intuitiv nicht leicht zu erfassen
 - Der **Join-Graph** des letzten Beispiels ist zyklisch



- Der azyklische Hypergraph hat einen **zyklischen Subgraphen**



- Die Query kann so (**äquivalent umgeformt**) werden, dass der Join-Graph azyklisch ist



- Theorem
*Eine Query Q ist azyklisch, wenn und nur wenn es eine äquivalente Umformung von Q gibt, deren **Join-Graph azyklisch** ist*

Komplexität

- Theoreme

- *Eine Query hat einen Full Reducer gdw. ihr **Hypergraph** azyklisch ist*
- *Sei Q eine azyklische Anfrage und E eine Hyperkante ihres Hypergraphen. Dann gibt es eine Sequenz von Ohr-Entfernungen, die E als **letztes Element** entfernt*
 - *Wird noch wichtig – später*
- *Für eine **lineare Query** ist das Finden eines **Full Reducers** linear*
 - *Eine Query ist linear, wenn man ihre Relationen so anordnen kann, dass jede Relation nur einen Join mit ihrem Vorgänger und einen Join mit ihrem Nachfolger hat*

- Beweise: Literatur

Full Reducer für lineare Anfragen

- Wir benötigen zwei Phasen
 - Vorwärts
 - Rückwärts
- Ablauf
 - $R1 \bowtie_A R2 \bowtie_B \dots \bowtie_Y R(n-1) \bowtie_Z Rn$
 - Vorwärts
 - $R2' = R2 \bowtie R1$
 - $R3' = R3 \bowtie R2' = R3 \bowtie (R2 \bowtie R1)$
 - ...
 - $Rn' = Rn \bowtie R(n-1)'$
 - Rückwärts
 - $R(n-1)'' = R(n-1)' \bowtie Rn'$
 - $R(n-2)'' = R(n-2)' \bowtie R(n-1)''$
 - ...
 - $R1'' = R1 \bowtie R2''$

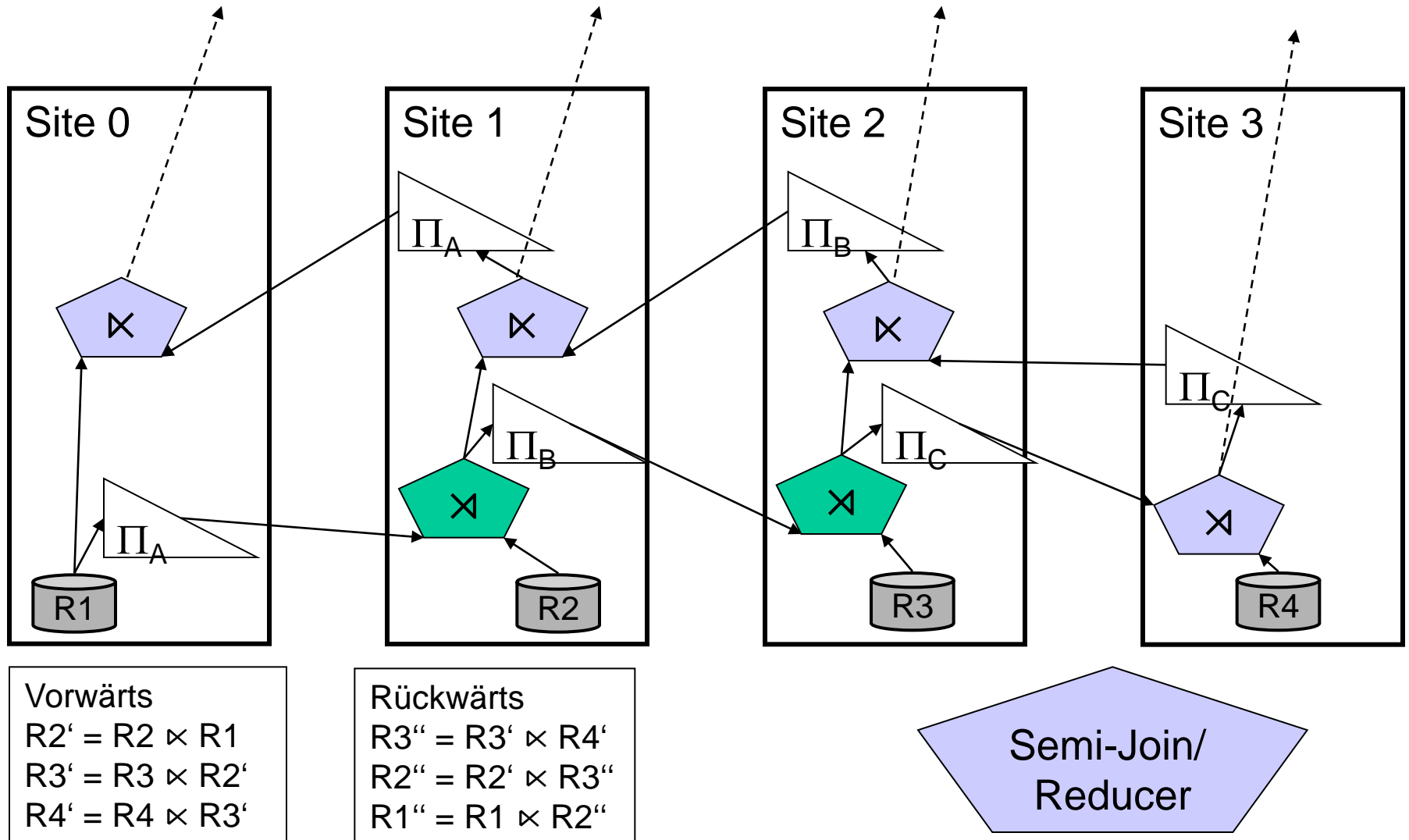
Reducer für Rn

Reducer für $R(n-1)$

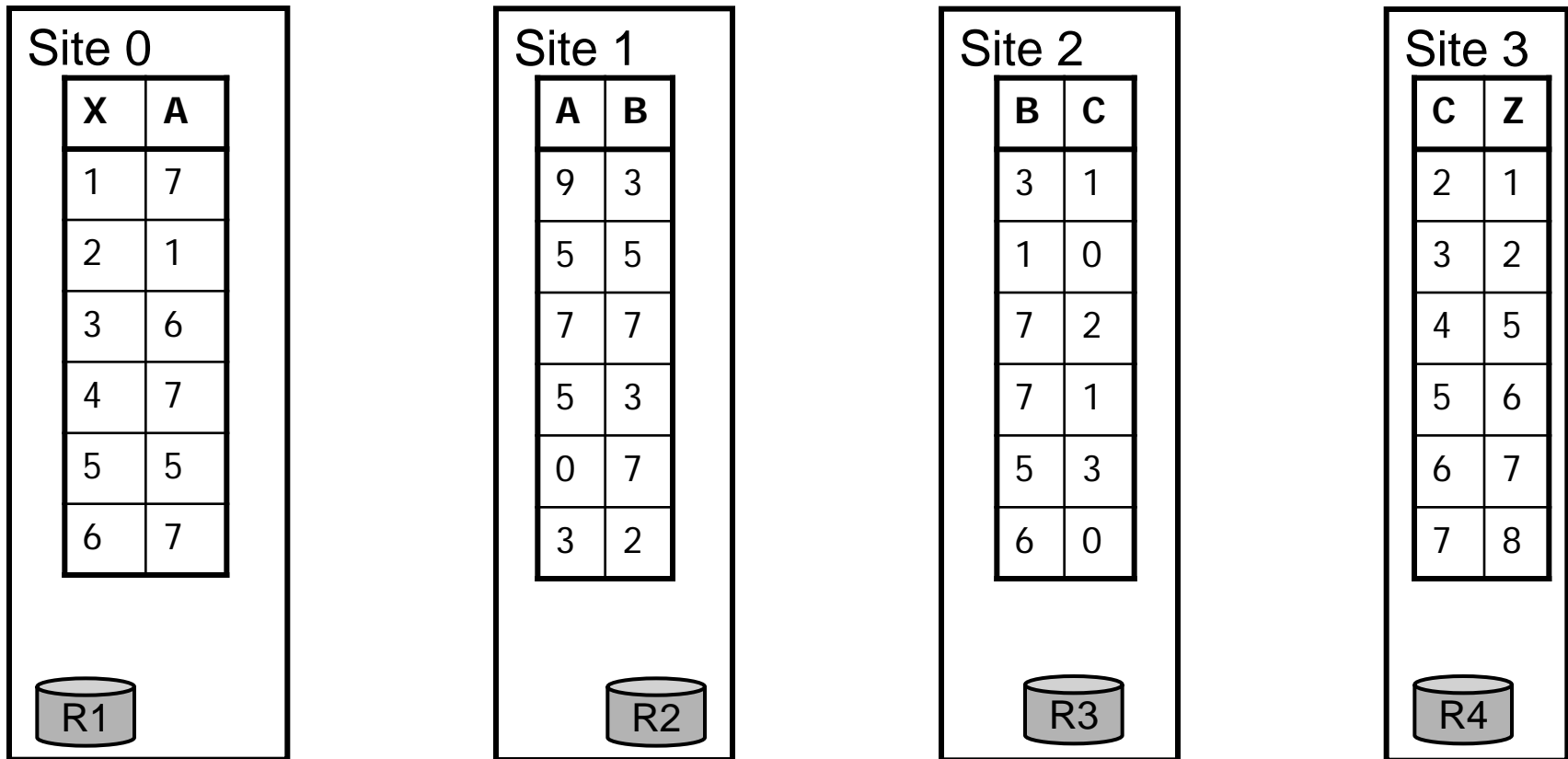
Reducer für $R1$

Full Reducer für Q

Als Ausführungsplan

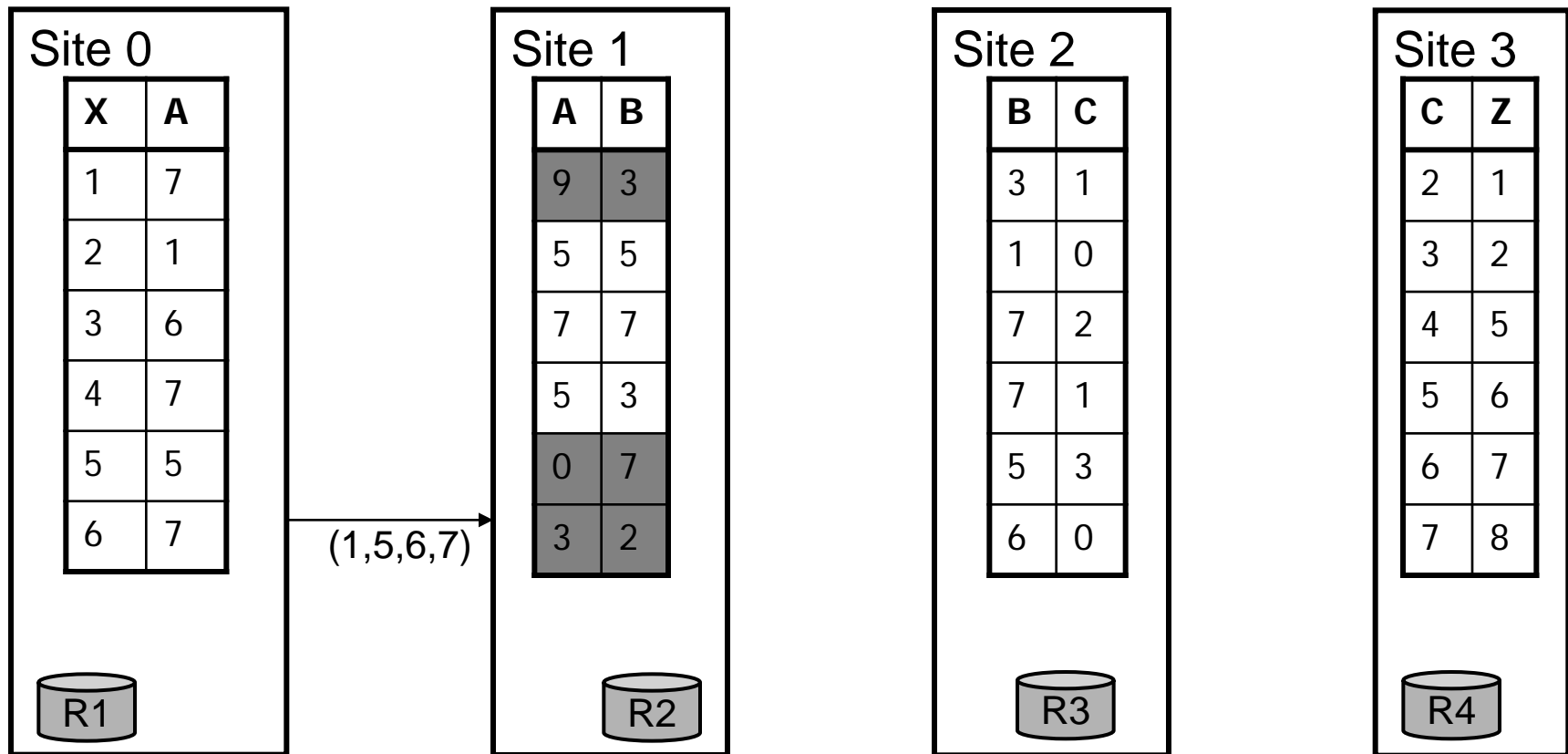


Beispiel



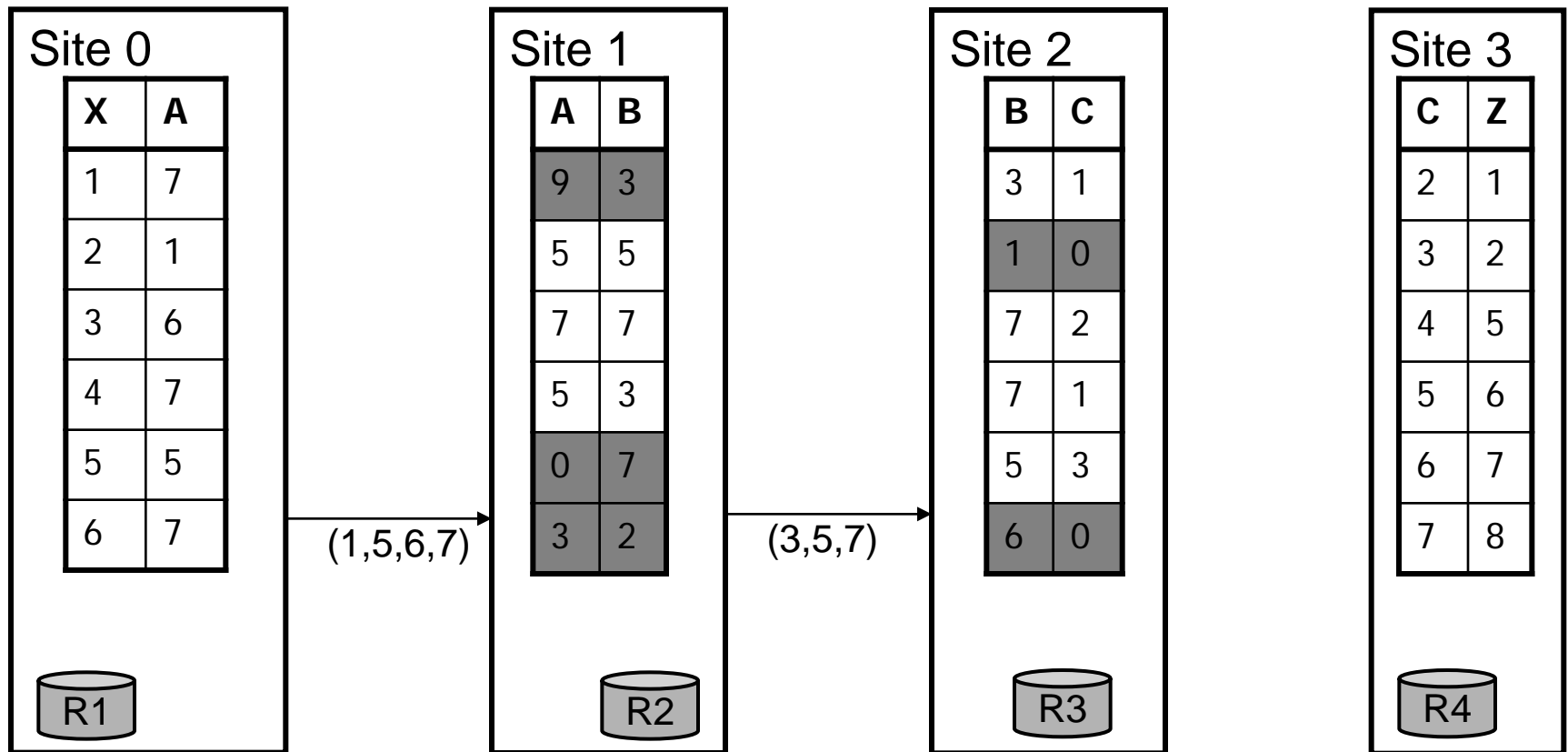
$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$

Fully Reduce – Beispiel



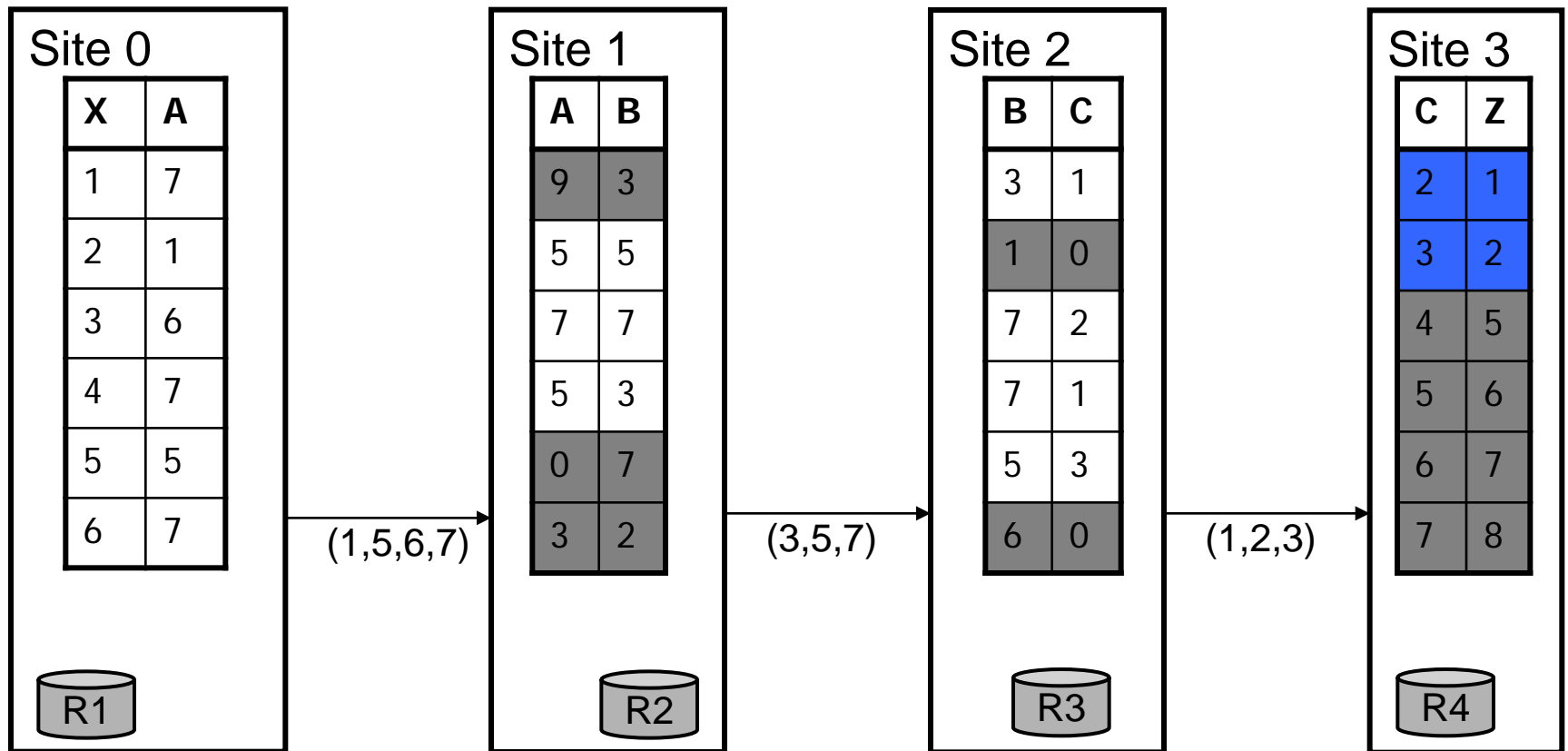
$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$

Fully Reduce – Beispiel



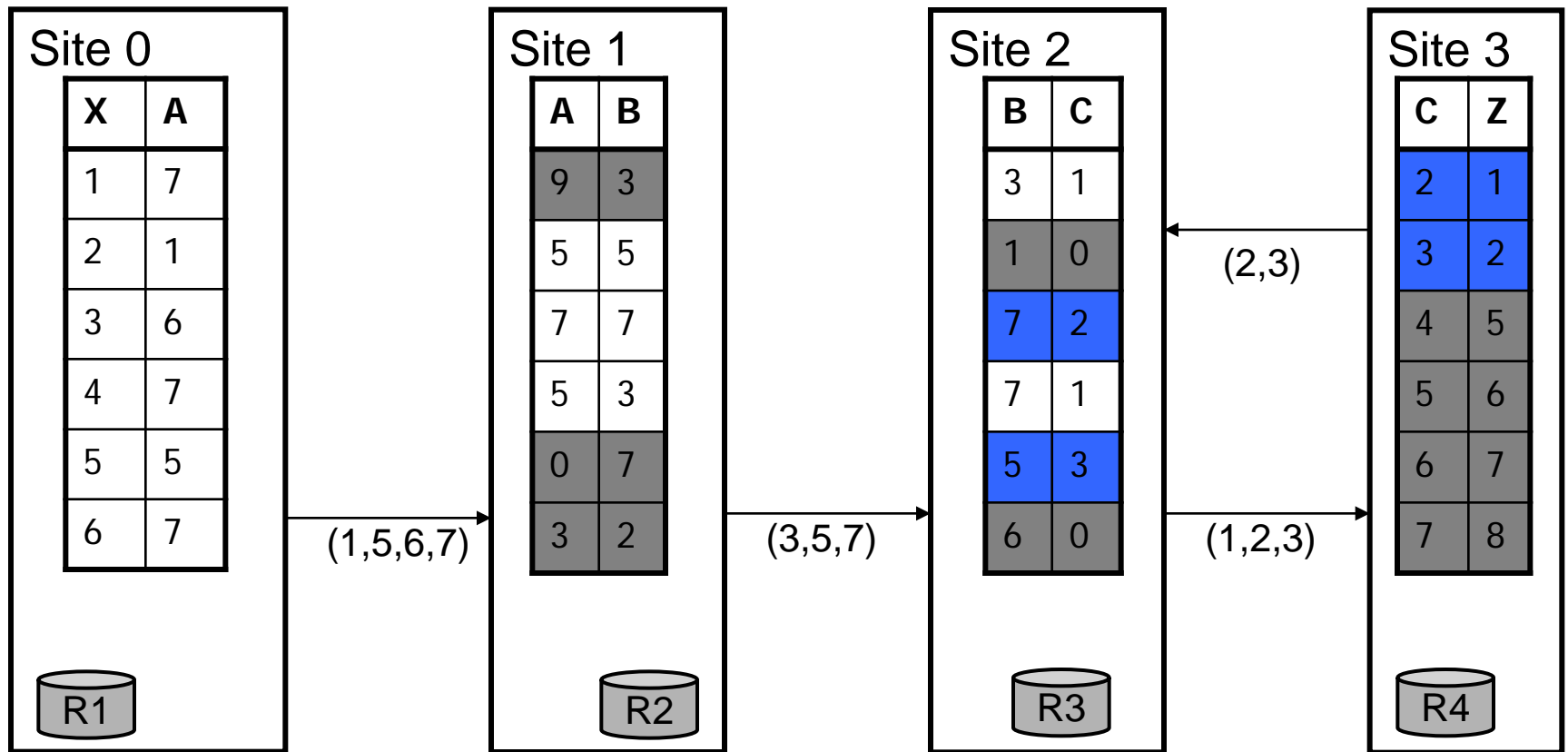
$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$

Fully Reduce – Beispiel



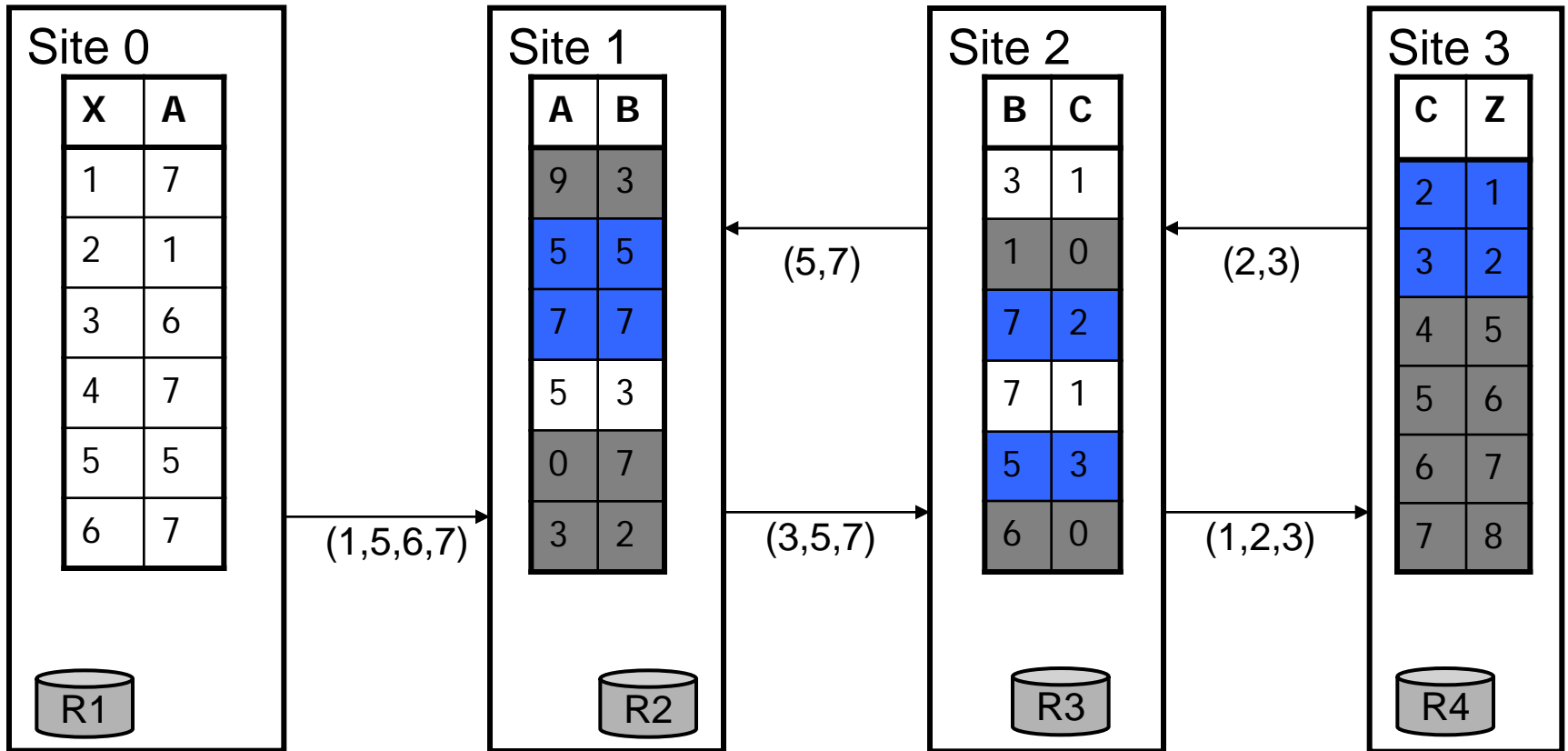
$$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$$

Fully Reduce – Beispiel



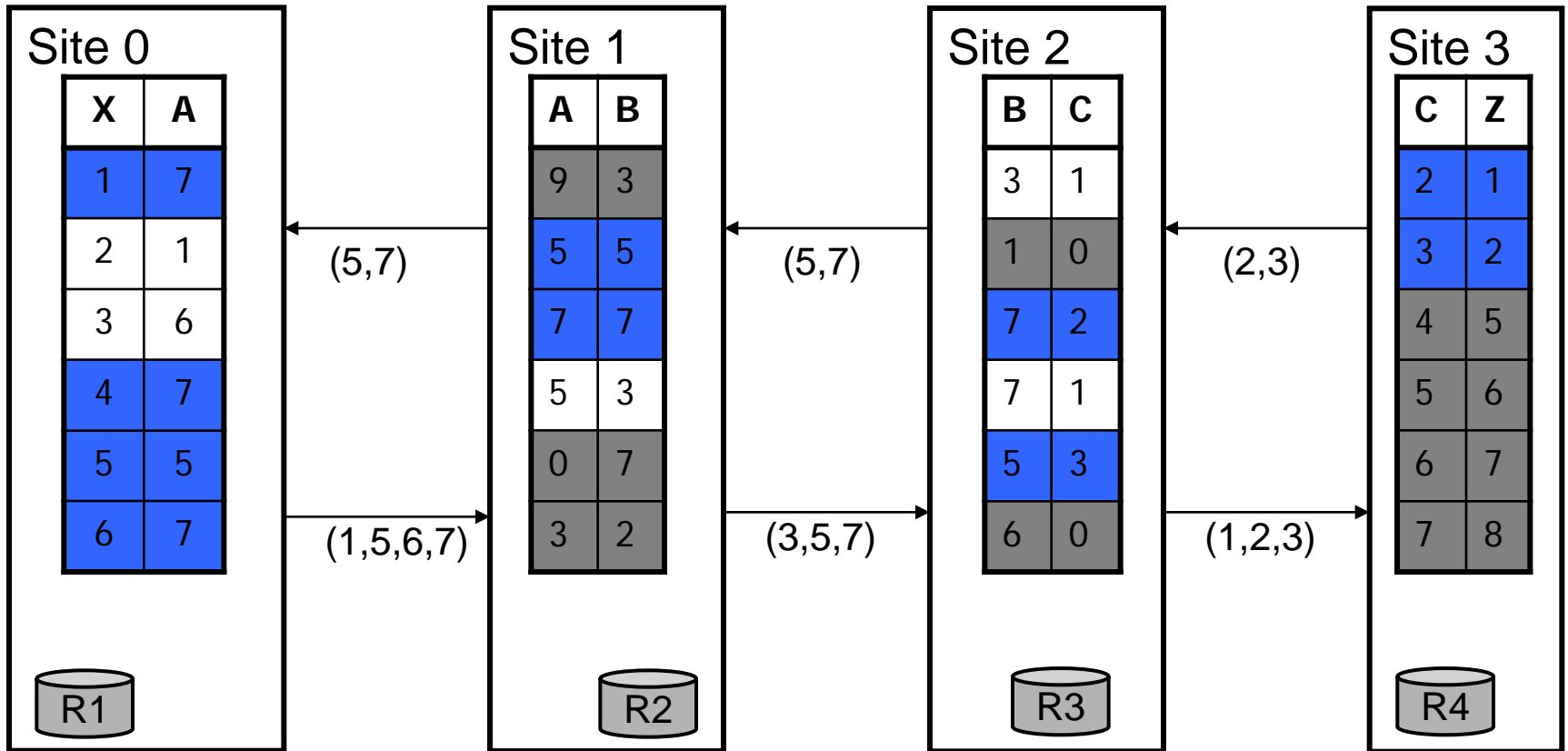
$R1 \bowtie_A R2 \bowtie_B R3 \bowtie_C R4$

Fully Reduce – Beispiel



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

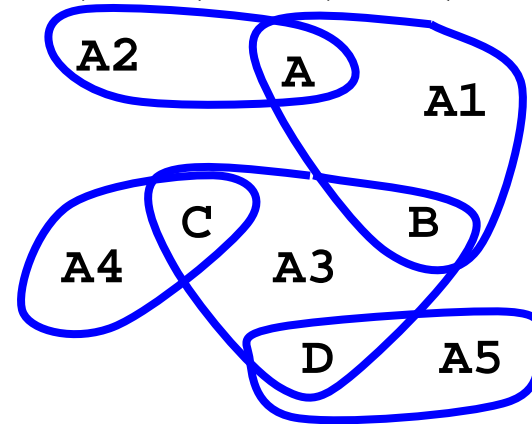
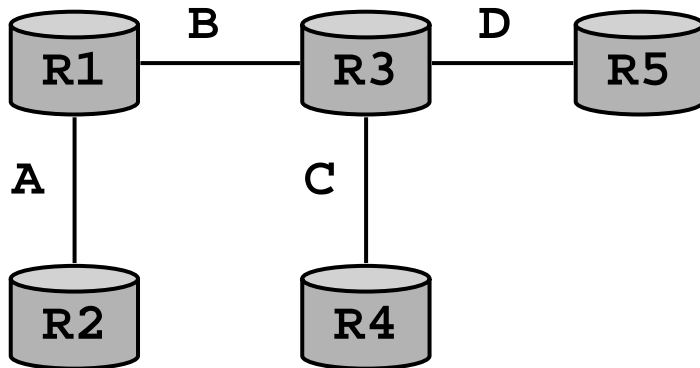
Fully Reduce – Beispiel



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

Azyklische Anfragen

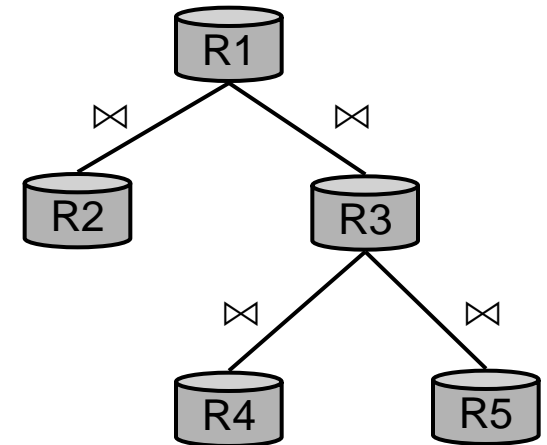
- Für azyklische Anfragen können wir in linearer Zeit einen **Reducer für eine beliebig gewählte Relation** finden
- Gegeben eine azyklische, nicht-lineare Anfrage Q
 - $R1(A1,A,B)$, $R2(A2,A)$, $R3(A3,B,C,D)$, $R4(A4,C)$, $R5(A5,D)$



- Offensichtlich azyklisch
- Wähle eine Relation (R1) als letztes Element

Reducer für azyklische Anfragen

- Setze die gewählte Relation als **Wurzel eines Baumes**
- Baue den Baum von den Blätter her auf
 - Füge sukzessive abgeschnittene Ohren O zu dem Baum
 - Kinder sind die Ohren, die vorher abgeschnitten werden mussten, um O „abschneidbar“ zu machen
 - Wurzel muss bleiben
- Reducer für die Wurzel
 - Von unten nach oben
 - Einführung von Semi-Joins von Knoten zu ihren Eltern

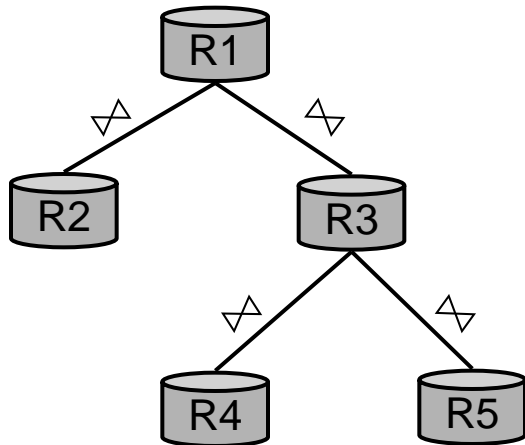


Reducer für R1

R1(A1,A,B), R2(A2,A), R3(A3,B,C,D), R4(A4,C), R5(A5,D)

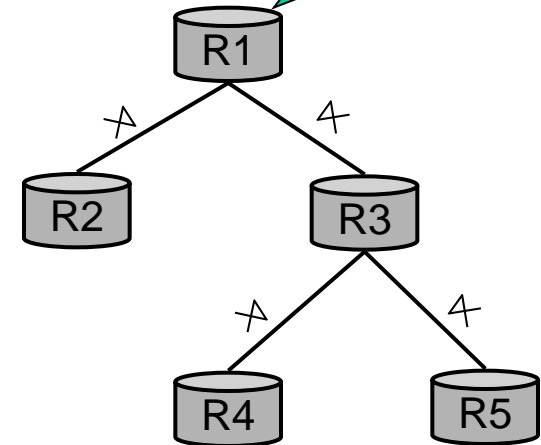
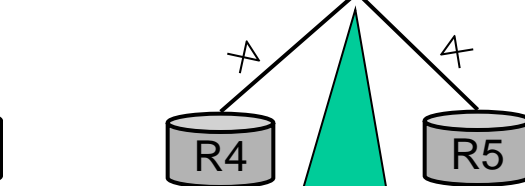
Erfüllte Bedingungen:

- 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 reduced.

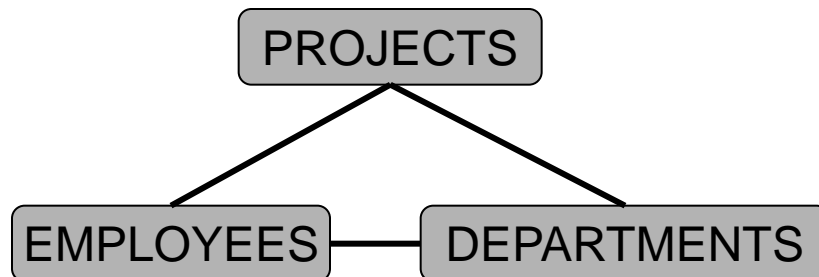
Zyklische Anfragen

- Zyklische Anfragen
 - Alle Mitarbeiter, die an Projekten der eigenen Abteilung arbeiten

```
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
```

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

Was passiert?

Zyklische Anfragen

- Zyklische Anfragen
 - Alle Mitarbeiter, die an Projekten der eigenen Abteilung arbeiten

```
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
```

DEPT	
DName	ID
A	1
b	2

PROJ		
d_ID	ID	PName
1	6	Clio
2	7	HumMer

- Semi-Join betrachtet nur zwei Relationen
- Kein paarweiser Join ist leer
- Es gibt keinen Full Reducer
- Aber das Gesamtergebnis ist leer

EMP		
EName	d_ID	p_ID
X	1	7
Y	2	6

Warnung

- Ein Full Reducer entfernt alle überflüssigen Tupel
- Um einen Full Reducer auszurechnen, müssen aber schon Tupel bewegt werden
- Wann die minimale Menge von Zwischenergebnissen bewegt wird, haben wir noch nicht betrachtet
 - **Freiheitsgrade**, z.B. Welche Relation wähle ich als Wurzel?
- Ein optimaler Plan unter Verwendung eines Full-Reducers muss auch andere Optimierungstechniken benutzen
 - Welches ist der beste Full Reducer?
 - Wo und in welcher Reihenfolge werden welche Semi-Joins ausgeführt?
 - **Minimaler Gesamttransport von Daten**

Literatur

- Philip A. Bernstein, Dah-Ming W. Chiu: Using Semi-Joins to Solve Relational Queries. *Journal of the ACM* 28(1): 25-40 (1981)
- W. Meng and C. Yu, "Query Processing in Multidatabase Systems," in *Modern Database Systems*, W. Kim, Ed. New York: ACM Press, Addison-Wesley, 1995, pp. 551-572.
- J. D. Ullman, „Principles of Database Systems and Knowledge-Based Systems. Volume II: The New Technologies“. Computer Science Press, Rockville, 1989.