5. Verteilte und parallele Anfragebearbeitung

- **Einführung**
- **Verteilte Anfragebearbeitung: Teilschritte**
  - Anfragetransformation
  - Daten-Lokalisierung
  - Globale Optimierung
- **Verteilte / parallele Verarbeitung von Selektion, Projektion, Aggregationen**
- **Verteilte Join-Verarbeitung**
  - Einfache Strategien (Ship Whole, Fetch as Needed)
  - Semi-Join und Bitvektor-Join
- **Parallele Join-Verarbeitung**
  - Dynamische Replikation und Partionierung der Eingaberelationen
  - Paralleler Hash-Join
- **Mehr-Wege-Joins**
- **Parallele Sortierung**

Problemstellung

- **Ziel:**

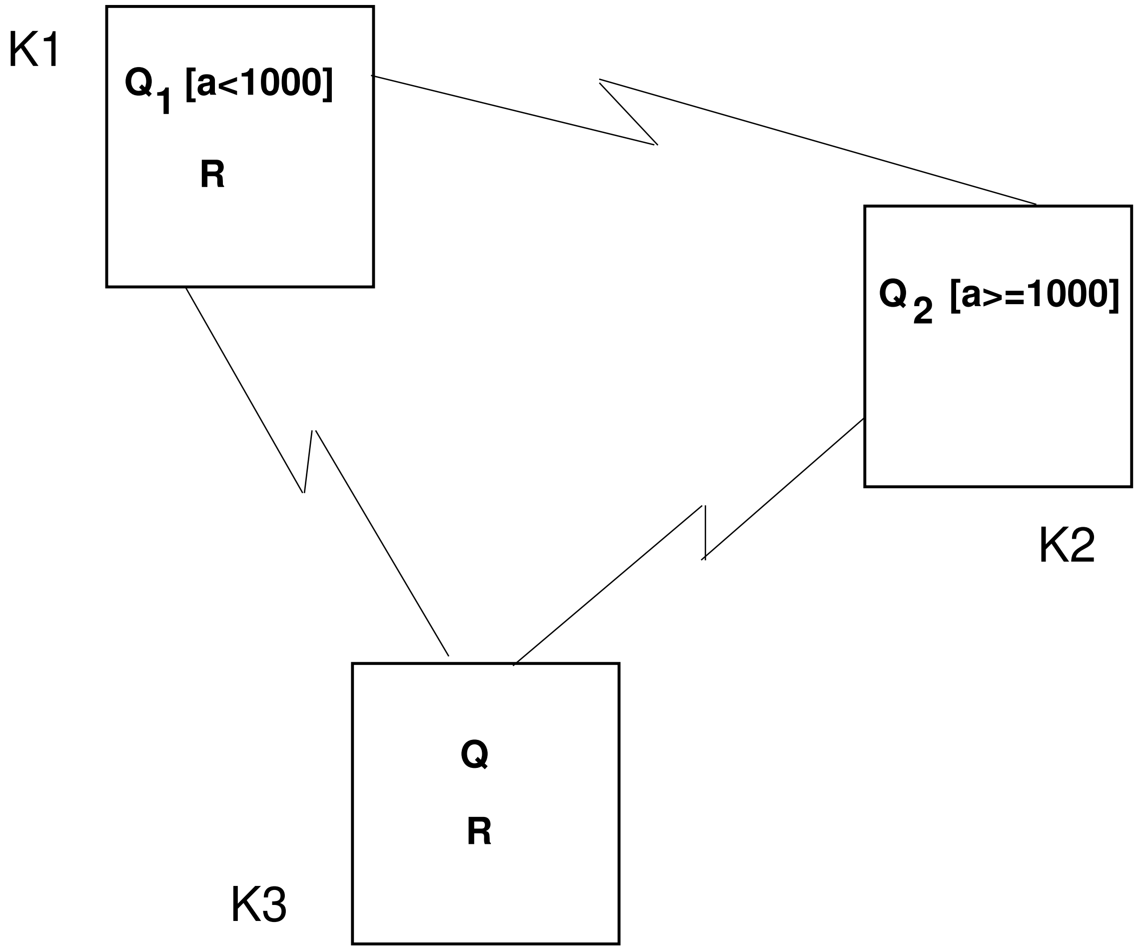
  Bestimmung eines Ausführungsplanes einer verteilten Anfrage in Abhängigkeit zur Datenverteilung, so daß eine Zielfunktion optimiert wird (z. B. Antwortzeit)

- **Kostenfaktoren:**
  - Nachrichtenanzahl
  - Nachrichtengröße
  - E/A
  - CPU-Bedarf
  - Hauptspeicherbedarf

- **Optimierungsentscheidungen:**
  - Query-Zerlegung in lokal ausführbare Teilanfragen
  - Ausführungsreihenfolge für Restriktion, Projektion und Join
  - Nutzung von Indexstrukturen
  - Parallelisierung von Teilanfragen
  - Auswahl der globalen und lokalen Join-Strategie (Nested-Loop, Sort-Merge, Hash-Join)
  - Rechnerauswahl, z.B. zur Join-Berechnung
  - Auswahl der Replikate

- **Berücksichtigung des aktuellen Systemzustandes zur Laufzeit wünschenswert**

Anfrageoptimierung: Problemstellung



■ **Anfrage in K3:**

SELECT * FROM Q WHERE a IS IN (482, 517, 763);

- lokale Ausführung in K3 oder
- Ausführung in K1 mit (kleinerem) Fragment Q1

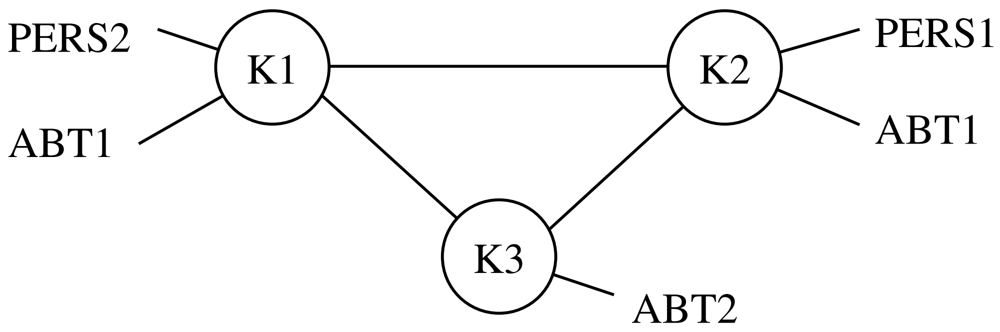
■ **Anfrage in K2:**

SELECT x, y, z FROM Q, R WHERE Q.j = R.k;

- Sende Anfrage zur Ausführung nach K3 oder
- Sende Fragment Q2 zur Join-Berechnung nach K1

Anfrage-Optimierung - Beispiel

- ABT (ANR, AORT,...)
  PERS (PNR, PNAME, ANR, ...)



➥ Fragmentierung durch Prädikate
ABT1: AORT = 'Stuttgart' OR AORT = 'München'
ABT2: AORT = 'Frankfurt'
PERS1: ANR $\leq$ 'K50'
PERS2: ANR > 'K50'

- **Anfrage in K3:**
  Finde die Namen aller Angestellten,
  die in Stuttgart Abteilungen arbeiten

- **Operatorbaum (Relationenalgebra)**

Projektion π PNAME

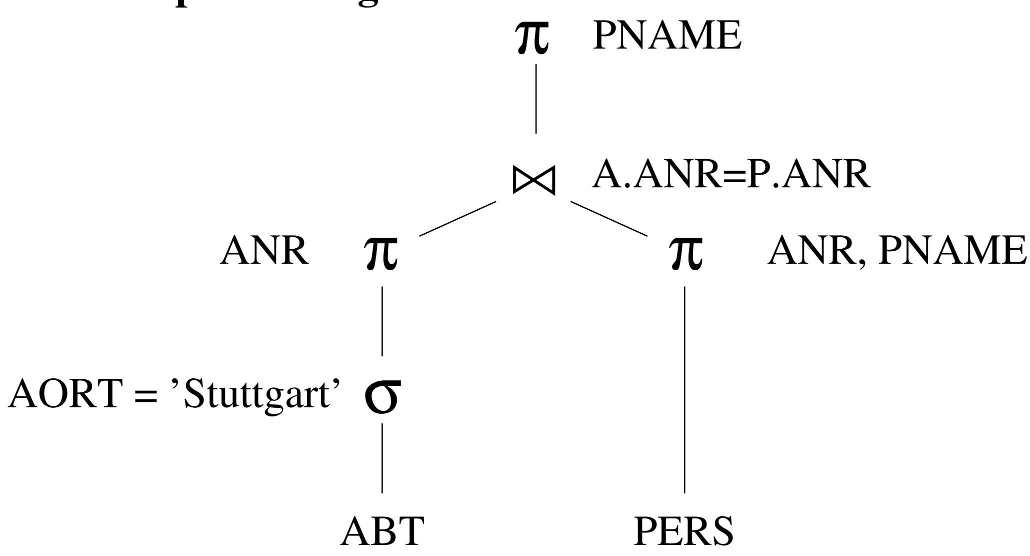
Selektion σ AORT = "Stuttgart"

Verbund $\bowtie$ A.ANR = P. ANR

ABT PERS

Anfrage-Optimierung - Beispiel (2)

■ **Algebraische Optimierung**



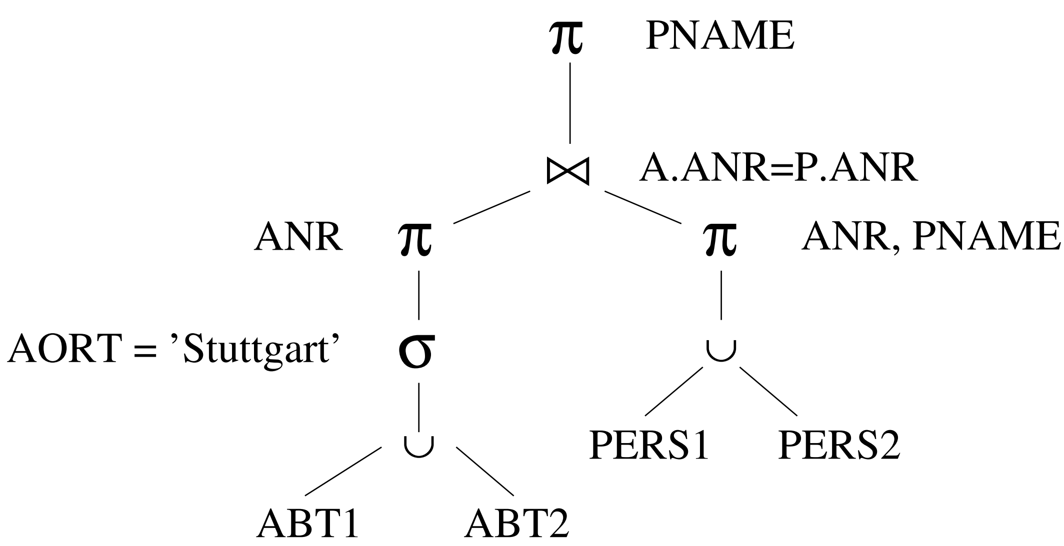
➥ Optimaler Operatorbaum für ein zentralisiertes DBS

■ **Im zentralisierten wie im verteilten Fall:** Fatale Annahmen

- Gleichverteilung aller Attributwerte eines Attributes
- Unabhängigkeit der Attributwerte untereinander
- weitere Annahme: gleichmäßige Belastung des Netzes

➥ Berechnung der erwarteten Anzahl der Nachrichten von obigen Annahmen abhängig

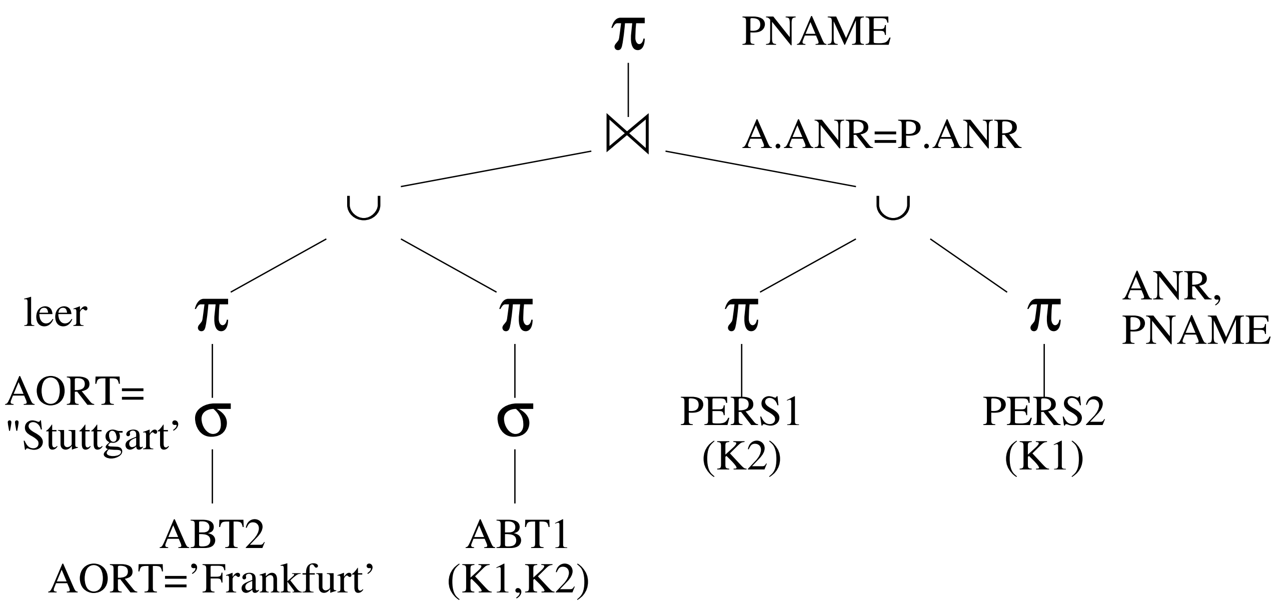
■ **Fragmentierungstransformation**



ABT und PERS sind mit Hilfe von Prädikaten fragmentiert

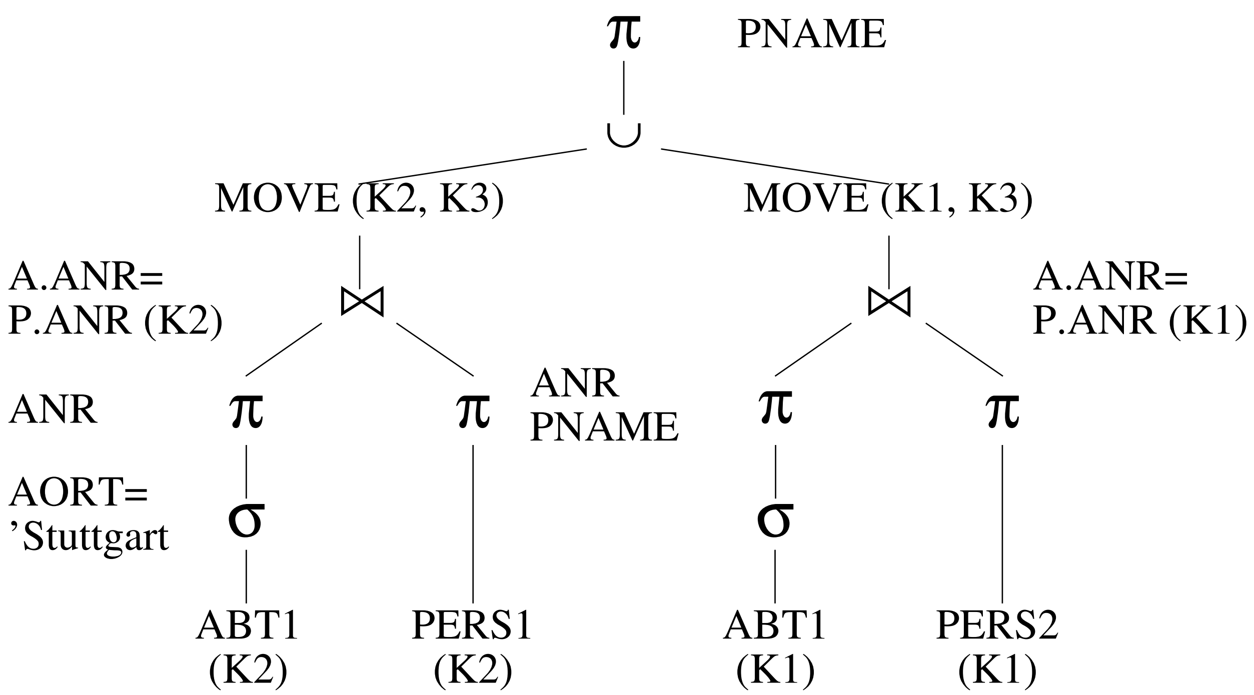
Anfrage-Optimierung - Beispiel (3)

- **Optimierung und Verteilungstransformation**



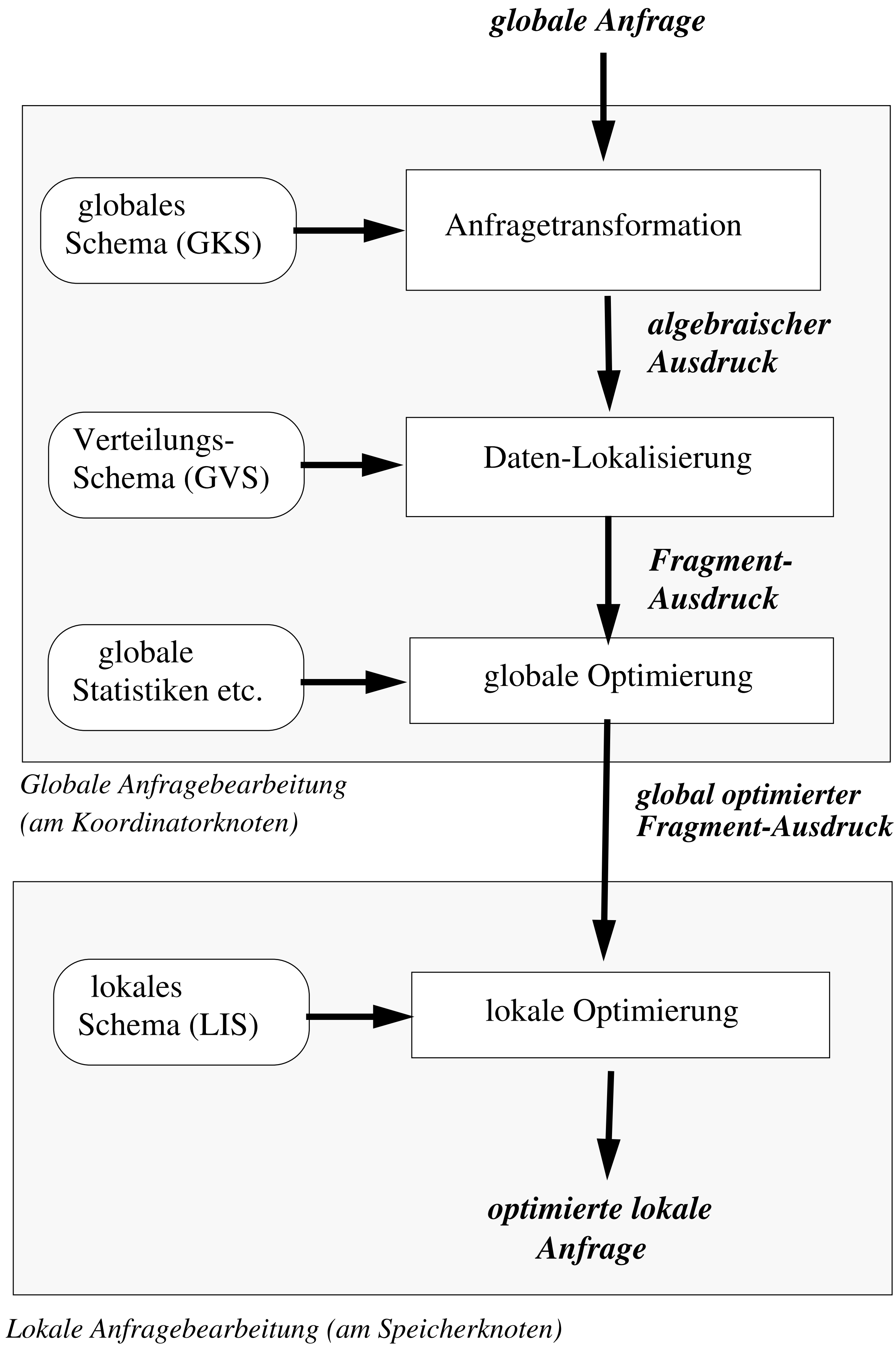
➥ PERS1 und ABT1 in K2
PERS2 und ABT1 in K1 } **erst** $\bowtie$, **dann** $\cup$

- **Verteilungsoptimierung**



➥ Minimierung des Kommunikationsaufwandes

Phasen der verteilten Anfragebearbeitung



Anfragetransformation

■ **Erzeugung einer Interndarstellung für die Anfrage:**
Anfragegraph (AG) oder Operatorgraph
(z.B. in Relationenalgebra)

■ **Einzelschritte**

- Lexikalische uns syntaktische Analyse (Parsing)
- Semantische Analyse und Namensauflösung
- Zugriffs- und Integritätskontrolle
- Standardisierung und algebraische Vereinfachung
- Restrukturierung und Transformation

■ **Algebraische Vereinfachung**
(z.B. Eliminierung redundanter Teilausdrücke)

■ **Anwendung von Idempotenz-/Äquivalenzregeln für logische Operationen**

1. $p \wedge p \Leftrightarrow p$
2. $p \vee p \Leftrightarrow p$
3. $p \wedge \text{TRUE} \Leftrightarrow p$
4. $p \vee \text{FALSE} \Leftrightarrow p$
5. $p \wedge \text{FALSE} \Leftrightarrow \text{FALSE}$
6. $p \vee \text{TRUE} \Leftrightarrow \text{TRUE}$
7. $p \wedge \neg p \Leftrightarrow \text{FALSE}$
8. $p \vee \neg p \Leftrightarrow \text{TRUE}$
9. $p1 \wedge (p1 \vee p2) \Leftrightarrow p1$
10. $p1 \vee (p1 \wedge p2) \Leftrightarrow p1$

Anfragetransformation (2)

- **Überführung der Anfrage in Operatorbaum**

- Umstrukturierungen zur effizienteren Auswertbarkeit durch Nutzung von Äquivalenzbeziehungen für relationale Operatoren

$\sigma_{P1}(\sigma_{P2}(R)) \Leftrightarrow \sigma_{P1 \wedge P2}(R)$

$\pi_{A}(\pi_{A,B}(R)) \Leftrightarrow \pi_{A}(R)$

$\sigma_{P}(\pi_{A}(R)) \Leftrightarrow \pi_{A}(\sigma_{P}(R))$ *falls P nur Attribute aus A umfaßt*

$\sigma_{P}(\pi_{A}(R)) \Leftrightarrow \pi_{A}(\sigma_{P}(\pi_{A,B}(R)))$
falls P auch Attribute aus B umfaßt

$\sigma_{P(R1)}(R1 \bowtie R2) \Leftrightarrow \sigma_{P(R1)}(R1) \bowtie R2$ *(P(R1) sei Prädikat auf R1)*

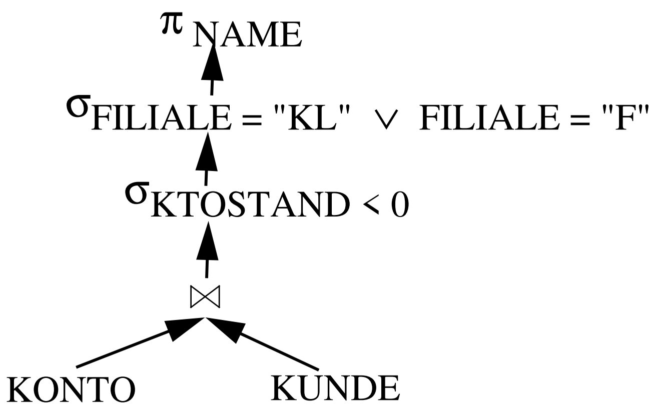
$\pi_{A,B}(R1 \bowtie R2) \Leftrightarrow \pi_{A}(R1) \bowtie \pi_{B}(R2)$ *(A /B seien Attributmengen aus R1 / R2 inklusive Join-Attribut)*

$\sigma_{P}(R1 \cup R2) \Leftrightarrow \sigma_{P}(R1) \cup \sigma_{P}(R2)$

$\pi_{A}(R1 \cup R2) \Leftrightarrow \pi_{A}(R1) \cup \pi_{A}(R2)$

- **Heuristiken**
  - frühzeitige Ausführung von Selektionsoperationen
  - frühzeitige Durchführung von Projektionen (ohne Duplikateliminierung)
  - Zusammenfassung mehrerer Selektionen und Projektionen auf demselben Objekt
  - Bestimmung gemeinsamer Teilausdrücke

- **Beispiel**



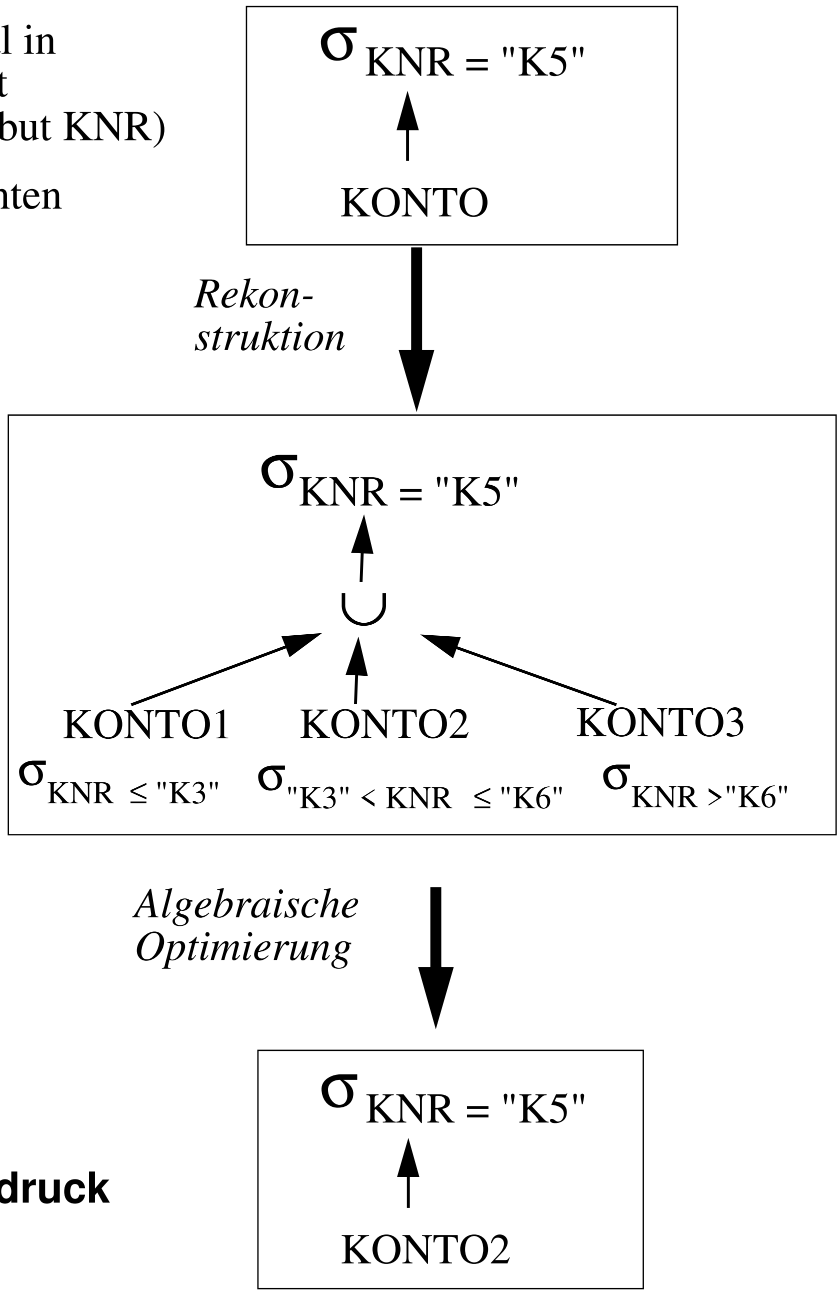
Erzeugung von Fragment-Anfragen
(Daten-Lokalisierung)

- **Relationen im Operatorbaum müssen auf Fragmente abgebildet werden**
  - Ersetze Relationen durch Rekonstruktionsanweisungen auf Fragmenten
  - Führe algebraische Vereinfachungen durch

- **Fall 1: Horizontale Fragmentierung**

KONTO sei horizontal in drei Fragmente zerlegt (Fragmentierungsattribut KNR)

Bestimmung aller Konten von Kunde K5



Initialer Fragment-Ausdruck

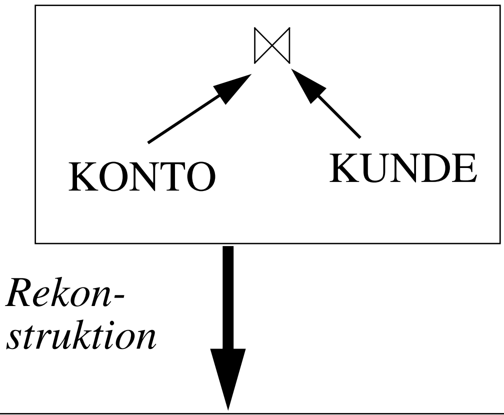
Reduzierter Fragment-Ausdruck

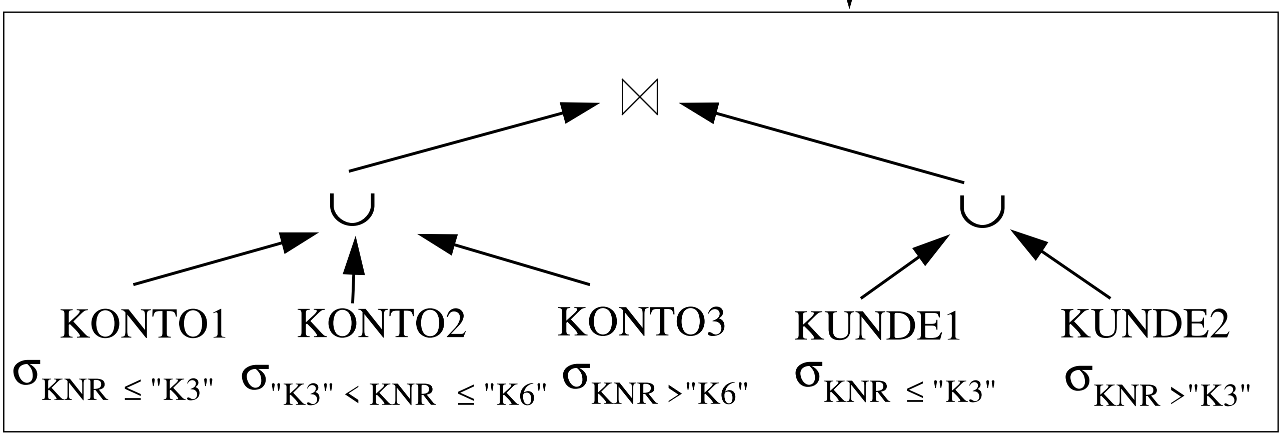
Erzeugung von Fragment-Anfragen (2)

■ **Join-Berechnung bei horizontaler Fragmentierung**

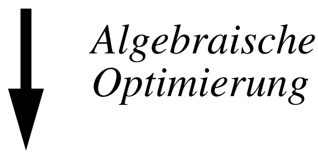
- reduzierter Kommunikations- und Verarbeitungsumfang für Joins auf Fragmentierungsattribut
- Unterstützung paralleler Join-Berechnung

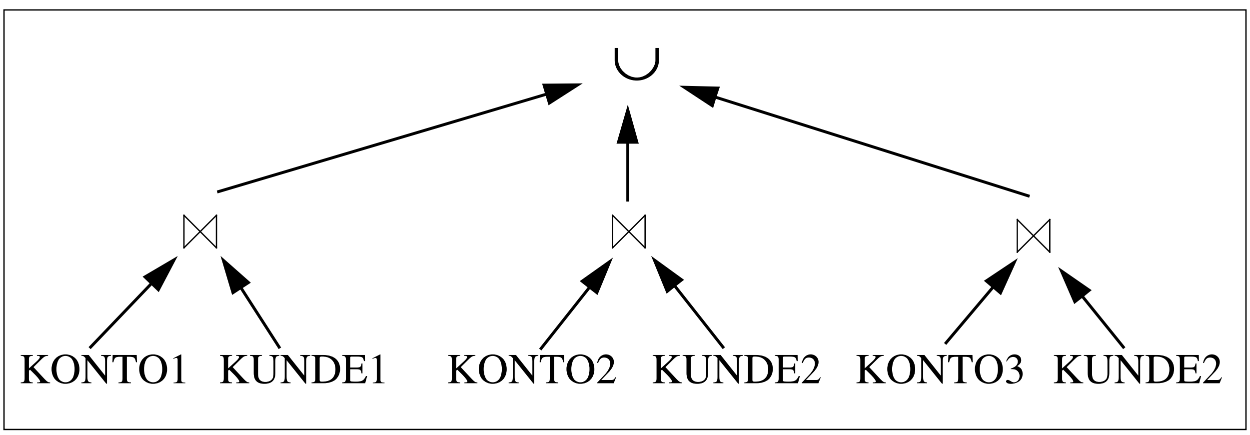
KONTO und KUNDE seien horizontal über KNR zerlegt (unterschiedl. Bereiche)





Initialer Fragment-Ausdruck





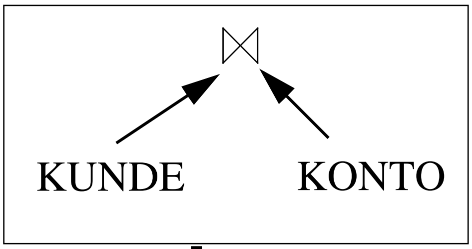
Reduzierter Fragment-Ausdruck

Erzeugung von Fragment-Anfragen (3)

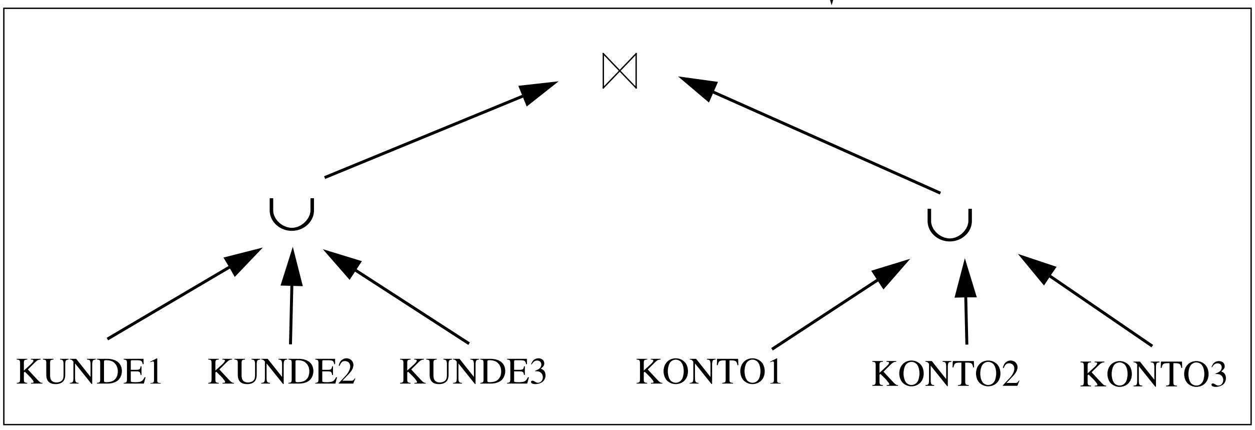
■ **Fall 2: Abhängige horizontale Fragmentierung**

Horizontale Fragmentierung von KUNDE über Filiale,

abhängige Fragmentierung von KONTO

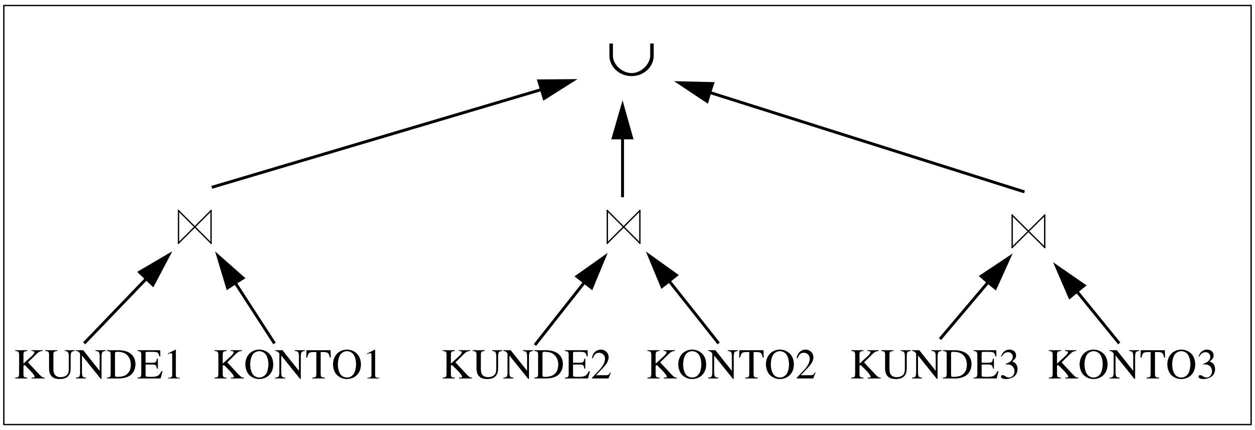


*Rekon-
struktion*



Initialer Fragment-Ausdruck

Algebraische Optimierung



Reduzierter Fragment-Ausdruck

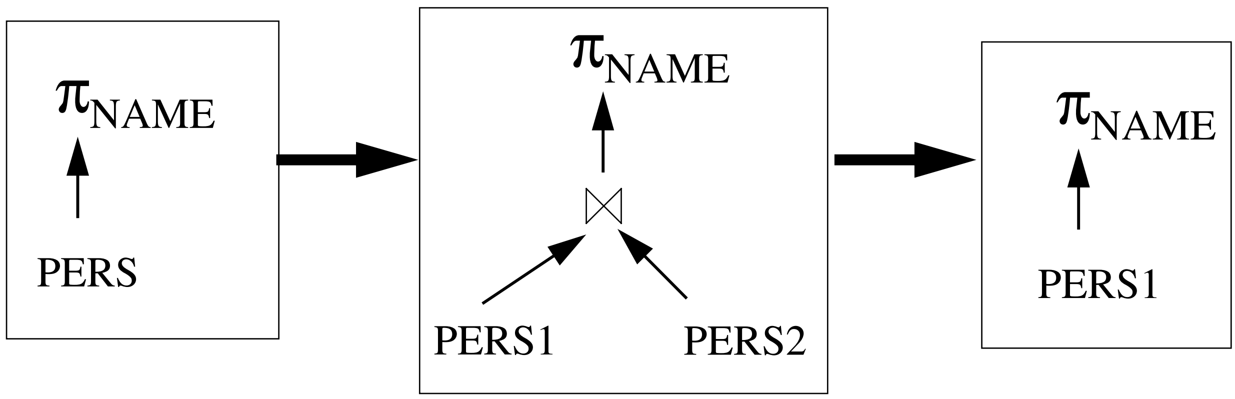
➥ Vollkommen lokale und parallele Join-Berechnung möglich

Erzeugung von Fragment-Anfragen (4)

■ Fall 3: Vertikale Fragmentierung

PERS1 (PNR, NAME); PERS2 (PNR, ANR)

Bestimme alle Angestelltennamen



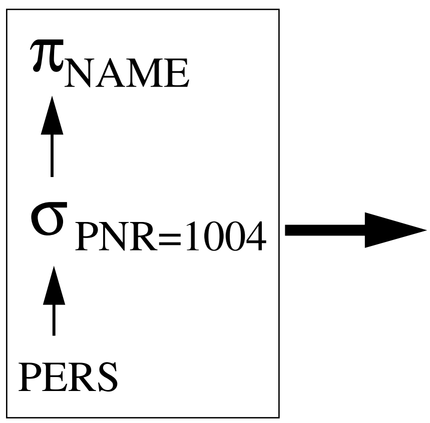
■ Fall 4: Hybride Fragmentierung

PERS1 = $\sigma_{PNR<1003}(\pi_{PNR,\ NAME}\ PERS)$

PERS2 = $\sigma_{PNR>=1003}(\pi_{PNR,\ NAME}\ PERS)$

PERS3 = $\pi_{PNR,\ ANR}\ PERS$

Bestimme den Namen von Angestelltem mit PNR 1004



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)

- **Trennung zwischen globaler und lokaler Optimierung kann zur Auswahl suboptimaler Pläne führen**

- **Kostenmodell**

  Berücksichtigung von CPU-, E/A- und Kommunikationskosten

$$
\begin{aligned}
\text{Gesamtkosten} = \; & W_{CPU} * \#\text{Instruktionen} + \\
& W_{I/O} * \#\text{E/A} + \\
& W_{Msg} * \#\text{Nachrichten} + \\
& W_{Byt} * \#\text{Bytes}
\end{aligned}
$$

Globale Optimierung (2)

■ **Benötigte Statistiken**

- Kardinalitäten von Relationen und Fragmenten
- Tupel- und Attributgrößen
- Häufigkeitsverteilung von Attributwerten, • • •

■ **Abschätzung von *Selektivitätsfaktoren SF***
(Größe von Zwischenergebnissen)

$Card(\sigma_p(R)) = SF(p) \cdot Card(R)$

- $SF(A = k) = 1 / Card(\pi_A(R)$
- SF (A > k) = Max (A) - k / Max (A) - Min (A)
- SF (A < k) = k - Min (A) / Max (A) - Min (A)
- $SF(p(A) \wedge p(B)) = SF(p(A)) \cdot SF(p(B))$
- $SF(p(A) \vee p(B)) = SF(p(A)) + SF(p(B)) - SF(p(A)) \cdot SF(p(B))$
- $SF(\neg p(A)) = 1 - SF(p(A))$

■ **Join-Selektivitätsfaktor (JSF)**

- Card ($R \bowtie S$) = JSF $*$ Card(R) $*$ Card(S)
- bei (N:1)-Joins (verlustfrei): $Card(R \bowtie S) = Max(Card(R), Card(S))$

Parallele Berechnung von Selektion und Projektion

- **Datenlokalisierung führt zu Selektions- und Projektionsoperationen auf Fragmenten**

- **Effektive Parallelisierung bei horizontaler Fragmentierung**

  $R = \cup\ (R1, R2, \ldots, Rn)$

  Selektion: $\sigma(R) \Rightarrow \cup\ (\sigma(R1), \sigma(R2), \ldots, \sigma(Rn))$

  Projektion: $\pi(R) \Rightarrow \cup\ (\pi(R1), \pi(R2), \ldots, \pi(Rn))$

  - Datenverteilung gestattet parallele Berechnung der lokalen Teilselektionen/-projektionen
  - Mischen der Teilergebnisse (Vereinigung)
  - Projektion: ggf. Duplikateliminierung (z.B. durch Sortierung)

- **Shared Nothing**
  - Ausführung auf den Datenknoten
  - Rechner und Parallelitätsgrad (n) durch Datenverteilung bestimmt (Ausnahme: bestimmte Anfragen auf dem Verteilattribut)
  - auch Index-Scans i. allg. auf allen n Rechnern auszuführen

- **Shared Disk / Shared Everything**
  - Datenverteilung auf Platte bestimmt nur maximalen Parallelitätsgrad
  - selektive Anfragen lassen sich auf einen Prozessor beschränken (➥ min. Kommunikationsaufwand)
  - Relationen-Scans können von n Prozessoren bearbeitet werden
  - Auswahl der Rechner kann zur Laufzeit erfolgen (➥ dynamische Lastbalancierung)
  - Parallelitätsgrad kann nicht nur vom Anfragetyp, sondern auch von der aktuellen Auslastung abhängig gemacht werden

Parallele Berechnung von Aggregatfunktionen

Sei Q(R) ein Attribut von R, auf das Built-in-Funktionen angewendet werden sollen

- **Parallele Berechnung für MIN, MAX immer möglich:**

MIN (Q(R)) ➥ MIN (MIN (Q(R1)), ... , MIN (Q(Rn))

MAX (Q(R)) ➥ MAX (MAX (Q(R1)), ... , MAX (Q(Rn))

- parallele Berechnung der lokalen Minima/Maxima
- Bestimmung des globalen Extremwerts

- **Für SUM, COUNT, AVG parallele Berechnung nur anwendbar, falls keine *Duplikateliminierung* erforderlich ist**

SUM (Q(R)) ➥ Σ SUM (Q(R_i))

COUNT (Q(R)) ➥ Σ COUNT (Q(R_i))

AVG (Q(R)) ➥ SUM (Q(R)) / COUNT (Q(R))

Join-Berechnung

■ Join (Verbund)

- satztypübergreifende Operation: gewöhnlich sehr teuer
- häufige Nutzung: wichtiger Optimierungskandidat
- typische Anwendung: Gleichverbund; allgemeiner Θ-Verbund selten

■ Implementierungsalternativen in zentralisierten DBS

- Nested Loop:
  - jeder Satz der ersten wird mit jedem Satz der zweiten Relation abgeglichen
  - für alle Join-Arten nutzbar
- Sort-Merge:
  - Eingaberelationen sind auf Join-Attribut sortiert (bzw. haben Index auf Join-Attribut)
  - Gleichverbund über schritthaltenden Durchgang und Abgleich der Relationen
- Hash-Join:
  - kleinere Relation wird in Hauptspeicher-Hash-Tabelle gebracht (Hash-Funktion auf Join-Attribut)
  - Prüfung für jeden Satz der zweiten Relation, ob Wert des Join-Attributs in Hash-Tabelle (Relation 1)
  - nur für Gleichverbund

■ Optimierungsziele in VDBS/PDBS:

- Reduzierung der Kommunikationskosten sowie Nutzung von Parallelverarbeitung
- Mehr-Wege-Joins: Wahl der Ausführungsreihenfolge; Einsatz von Inter-Operator-Parallelität

Join-Berechnung in Verteilten DBS

- **Anfrage in Knoten K, die Join zwischen**
  - (Teil-)Relation R an Knoten K_R und
  - (Teil-)Relation S an K_S erfordert

- **Festlegung des Ausführungsknotens:** K, K_R oder K_S

- **Bestimmung der Auswertungsstrategie**

  a) Sende beteiligte Relationen vollständig an einen Knoten und führe lokale Join-Berechnung durch (*"Ship Whole"*)
  - minimale Nachrichtenanzahl
  - sehr hohes Übertragungsvolumen

  b) Fordere für jeden Verbundwert der ersten Relation zugehörige Tupel der zweiten an (*"Fetch as Needed"*)
  - hohe Nachrichtenanzahl
  - nur relevante Tupel werden berücksichtigt

  c) Kompromißlösung:
  Semi-Join bzw. Erweiterungen (*Hash-Filter-Join*)

- **Mehr-Wege-Joins: Wahl der Ausführungsreihenfolge !**

Kommunikationskosten der Join-Berechnung

■ **Hier: Verwendung von zwei Kostenfaktoren**

- K_N = Kommunikationskosten pro Nachricht
- K_A = Kommunikationskosten für Übertragung eines Attributwerts

■ **Beispiel:**

R

A	B
1	7
2	2
3	7
4	8
5	6
6	3
7	9

S

B	C	D
1	6	4
2	5	1
1	4	2
5	3	3
5	2	5
6	1	8

$R \bowtie S$

A	B	C	D
2	2	5	1
5	6	1	8

■ **Strategie 1:**

- Übertragung von R und S nach Knoten K
- Join-Berechnung an Knoten K

■ **Strategie 2:**

- Übertragung von R nach Knoten K_S
- Join-Berechnung an Knoten K_S

■ **Strategie 3:**

- Join-Berechnung an Knoten K_S
- pro S-Tupel sukzessive Anforderung von zugehörigen R-Tupeln

Semi-Join (1)

■ **Reduzierung des Kommunikationsaufwandes**

- alle Verbundpartner können auf einmal angefordert werden
- Wegfiltern von Attributen, die für Join nicht benötigt werden

■ **Entspricht Anwendung des Semi-Joins:**

Semi-Join von R auf S entspricht dem Verbund zwischen R und S, wobei Attribute von S im Ergebnis nicht enthalten sind

$$R \ltimes S \;=\; \pi_{Attr(R)}\, R \bowtie S \;=\; R \bowtie \pi_v\, S$$

(v = Verbundattribut)

$$R \bowtie S \;\Leftrightarrow\; R \bowtie (S \ltimes R) \;\Leftrightarrow\; R \bowtie (S \bowtie \pi_v\,(R))$$

■ **Nutzung zur Join-Berechnung an Knoten K_R:**

1. Bestimme $R' = \pi_v\, R$ in K_R und sende R' an Knoten K_S
2. In K_S bestimme $S' = (S \bowtie R') = S \ltimes R$ und sende S' an K_R
3. K_R berechnet den Join zwischen R und S' (entspricht $R \bowtie S$)

Beispiel:

1) Bestimme $R' = \pi_B\, R$

2) Bestimme $S' = S \bowtie R'$

3) Bestimme $R \bowtie S'$

Kosten:

Semi-Join

Frankfurt

ABTEILUNGEN

ABTNR	ORT	MGR
47		
39		
64		

Join

ABTNR	NAME
47	Hans
47	Anna

Verschicke die ganze VA-Spalte

Verschicke projizierte Join-Partner zurück

München

ABTNR
47
39
64

Finde Join-Partner

ANGESTELLTE

ABTNR	NAME	ADRESSE	TELEFON
69			
28			
75			
47	Hans		
47	Anna		
44			

Semi-Join (2)

■ **Nutzung zur Join-Berechnung an drittem Knoten K:**

1. Bestimme R' = π_V R in K_R und sende R' an Knoten K_S
2. In K_S bestimme Semi-Join S' = S $\ltimes$ R' und sende S' an K
3. In K_S ermittle außerdem S'' = π_V S' und sende S'' an K_R
4. In K_R berechne Semi-Join R'' = R $\ltimes$ S'' und schicke R'' an K.
5. K berechnet das Ergebnis durch R'' $\bowtie$ S'

Beispiel:

1) Bestimme R' = π_B R

2) Bestimme S' = S $\ltimes$ R'

3) Bestimme S'' = π_B S'

4) Bestimme R'' = R $\ltimes$ S''

5) Bestimme R'' $\bowtie$ S'

■ **Semi-Join besonders wirksam,** wenn sich wenige Tupel qualifizieren (sonst schlechtere Ergebnisse als mit einfacher Join-Auswertung möglich)

Bitvektor-Join (Hash-Filter-Join)

■ **weitergehende Reduzierung des Übertragungsvolumens** als bei Semi-Join durch gröberen Filter

■ **Abbildung von Werten des Join-Attributes auf Bitvektor mittels Hash-Funktion**
Übertragung des Bitvektors anstatt von Attributwerten

■ **Nutzung zur Join-Berechnung an Knoten K_S:**

1. In K_R setze für jeden Wert in π_V R zugehöriges Bit im Bitvektor und sende diesen an Knoten K_S
2. In K_S bestimme S' = $\{s \in S \mid \text{Bit für s.v im Bitvektor gesetzt}\}$ und sende S' an K_R
3. K_R berechnet den Join zwischen R und S'

■ Teilrelation R' in Schritt 2 i. allg. größer als bei Semi-Join, dafür Bit-Vektor kompakter als Menge der Attributwerte

Hash-Filter (Bit-Vektor)-Join

Frankfurt

ABTEILUNGEN

ABTNR	ORT	MGR
47		
39		
64		

Test + Join

ABTNR	NAME
28	Fritz
47	Hans
47	Anna

Erzeuge Bit-Vektor durch Hashing

1	0	0	1	1	0	0	0

Schicke potentielle Join-Partner zurück

Verschicke Bit-Vektor

ANGESTELLTE

ABTNR	NAME	ADRESSE	TELEFON
69			
28	Fritz		
75			
47	Hans		
47	Anna		
44			

1	0	0	1	1	0	0	0

München

Hashing der ABTNR-Werte, um potentielle Join-Kandidaten zu finden

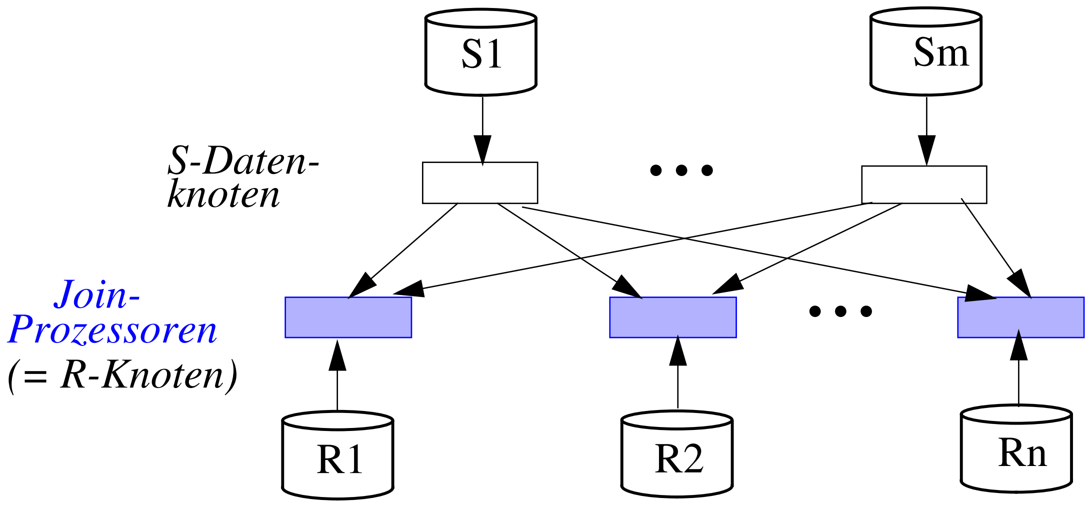
Parallele Join-Berechnung durch dynamische Replikation der kleineren Relation

- **Join zwischen R und S:**

  R $= \cup$ (R1, R2, ... , Rn) und S $= \cup$ (S1, S2, ... , Sm), S sei kleiner als R

- **Algorithmus**

  1. *Koordinator: initiiere Join auf allen R_i (i=1 ... n) und allen S_j (j =1 .. m)*
  2. *Scan-Phase: in jedem S-Knoten führe parallel durch:*
     lies lokale Partition S_j und sende sie an jeden Knoten R_i (i= 1.. n)
  3. *Join-Phase: in jedem R-Knoten mit Partition R_i führe parallel durch:*
     - $S := \cup S_j$ (j=1..m)
     - berechne $T_i := R_i \bowtie S$ (impliziert Lesen von R_i)
     - schicke T_i an Koordinator
  4. *Koordinator: empfange und mische alle T_i*

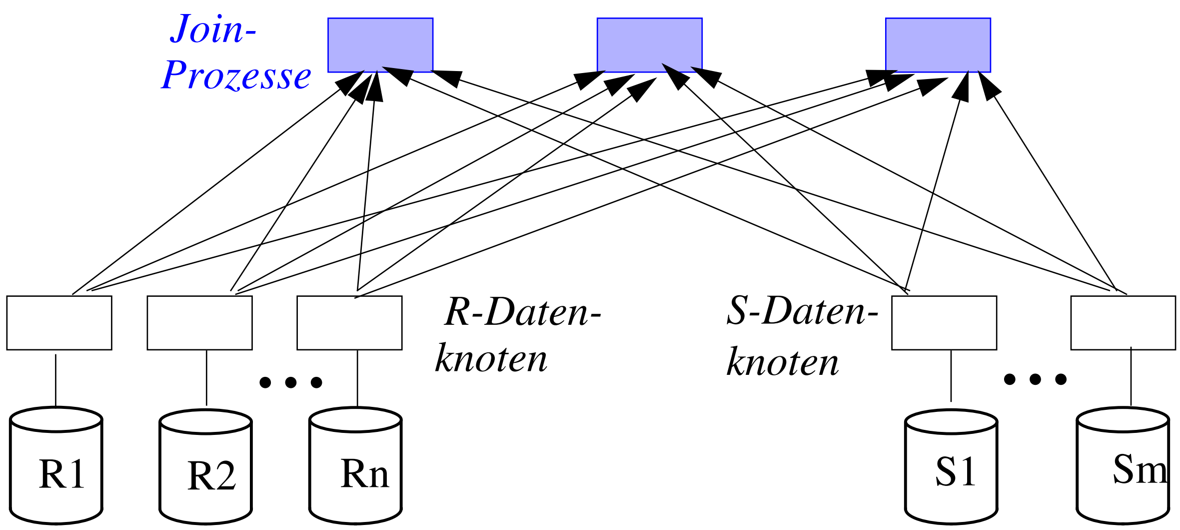


Eigenschaften:

- für alle Join-Prädikate einsetzbar
- lokale Join-Berechnung mit beliebiger Strategie möglich

Parallele Join-Berechnung durch dynamische Partitionierung

- **Voraussetzung: Gleichverbund !**
- Allgemeiner Fall:
  - Umverteilung beider Relationen unter p Join-Prozessoren
  - Verteilungsfunktion (Hash- oder Bereichspartitionierung) auf dem Join-Attribut



- **Bewertung:**
  - jeder **lokale** Join-Algorithmus anwendbar
  - reduzierter Join-Aufwand gegenüber dynamischer Replikation
  - hohe Flexibilität zur dynamischen Lastbalancierung (Wahl des Parallelitätsgrades p sowie der Join-Prozessoren)
  - hoher Kommunikationsaufwand

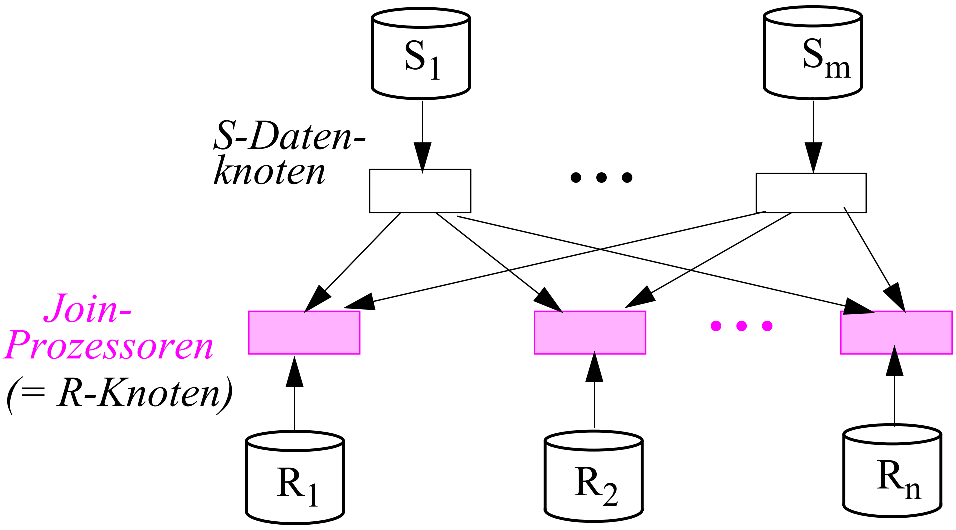
Dynamische Partitionierung (2)

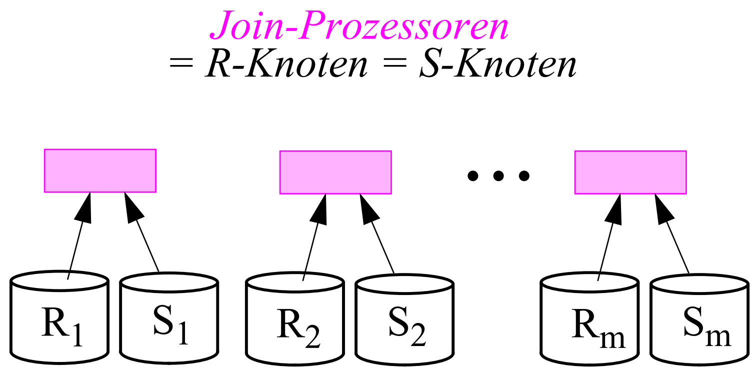
■ **Spezialfall 1:**
Verteilungsattribut = Join-Attribut für eine Relation

- nur eine Relation braucht umverteilt zu werden
- Potential zur dynamischen Lastbalancierung entfällt

■ **Spezialfall 2:**
Beide Relationen besitzen Join-Attribut als Verteilattribut und identische Verteilfunktion (m=n, R und S an denselben Rechnern)

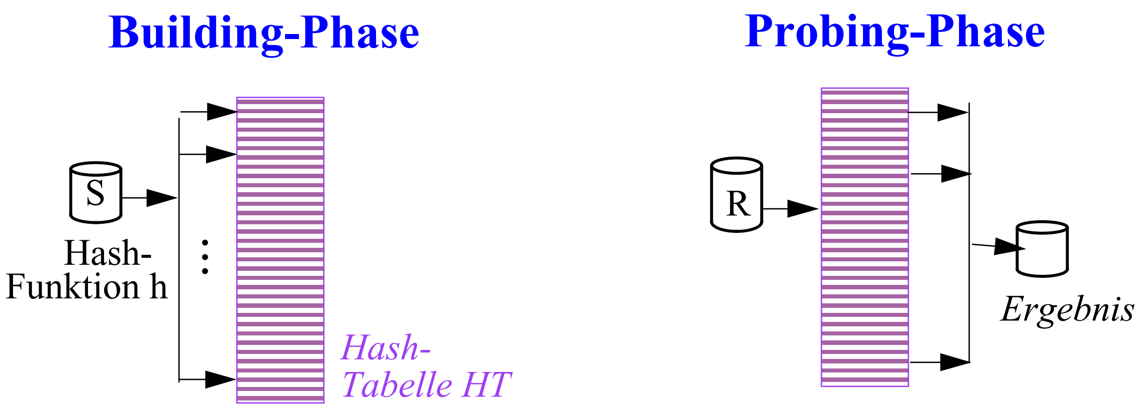
- entspricht abhängiger horizontaler Fragmentierung
- keinerlei Umverteilung erforderlich !





Hash-Join (zentralisierter Fall)

- **nur für Gleichverbund**
- **Idealfall: kleinere (innere) Relation S paßt vollständig in Hauptspeicher**
  - Building-Phase:
    Einlesen von S und Speicherung in HT im Hauptspeicher unter Anwendung einer Hash-Funktion h auf dem Join-Attribut
  - Probing-Phase:
    Einlesen von R und Überprüfung für jeden Join-Attributwert, ob zugehörige S-Tupel vorliegen (wenn ja, erfolgt Übernahme ins Join-Ergebnis)



- **Vorteile**
  - lineare Kosten O (N)
  - Hashing reduziert Suche nach Verbundpartnern auf Sätze einer Hash-Klasse (Partitionierung des Suchraumes)
  - Nutzung großer Hauptspeicher
  - auch für Joins auf Zwischenergebnissen gut nutzbar

Hash-Join (2)

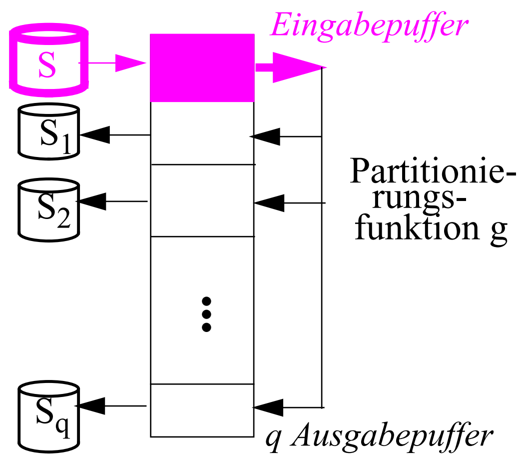
■ **Allgemeiner Fall:**
kleinere Relation paßt nicht vollständig in Hauptspeicher
➥ Überlaufbehandlung erforderlich

■ **Lösung: Partionierung der Eingaberelationen**

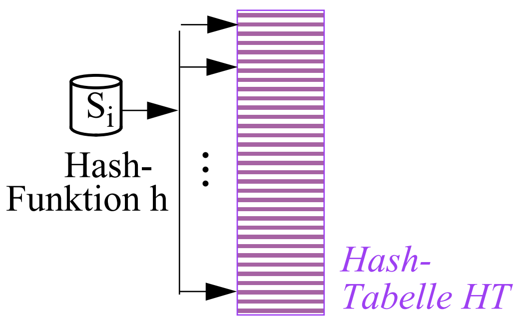
- Partitionierung von S und R in q Partitionen über (Hash-)Funktion g auf dem Join-Attribut, so daß jede S-Partition in die HT paßt
- q-fache Anwendung des Basisalgorithmus’ auf je zwei zusammengehörigen Partitionen

Partitionierungs-Phase

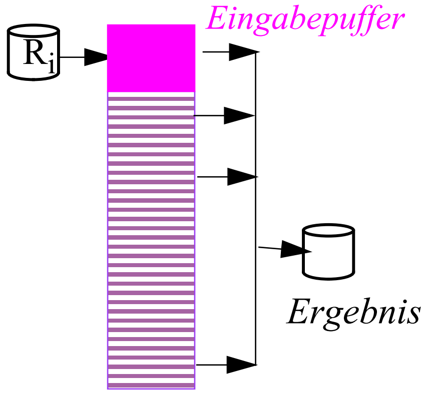
(für R analog)



Building-Phase



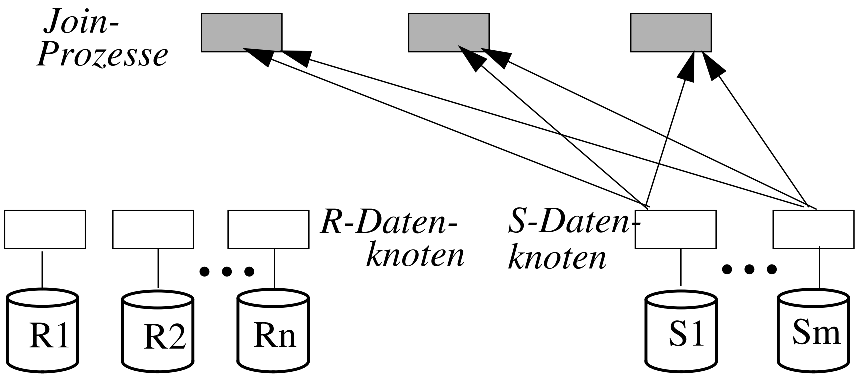
Probing-Phase



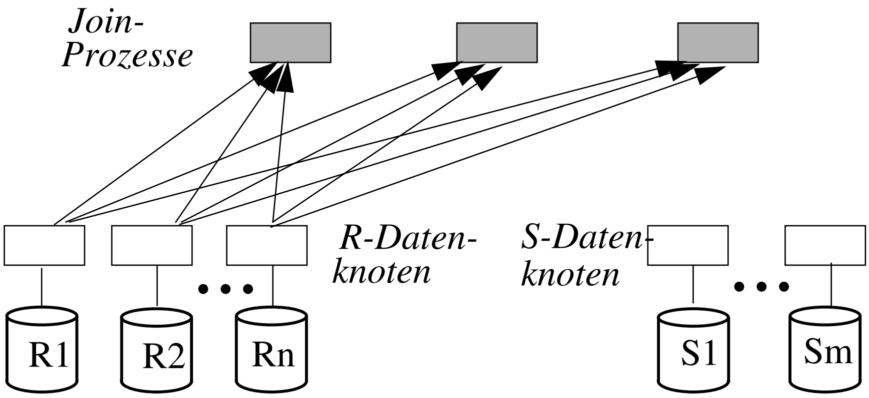
■ rund 3-facher E/A-Aufwand gegenüber Basisverfahren ohne Überlauf

Paralleler Hash-Join

Building-Phase:



Probing-Phase:



- **Prinzip:**
  - Partitionierung und Umverteilung der kleineren Relation S über Hash-Funktion h_P auf dem Join-Attribut
  - In Join-Prozessoren kommen eingehende Tupel in HT (Hash-Fkt. h_{HT})
  - Umverteilung der zweiten Relation R auf die Join-Prozessoren unter Anwendung von h_P
  - Probing: für eingehende Tupel werden Verbundpartner in HT ermittelt

- **Merkmale:**
  - Sequentialisierung der Scan-Phasen !
  - Vorteil: Reduzierung des Umverteilungsaufwandes für R durch Anwendung von Bitvektor-Filterung möglich
  - Pipeline-Parallelität in Building- und Probing-Phase möglich
  - Überlaufbehandlung erforderlich, falls S-Partitionen nicht vollständig in HT (Hauptspeicher) Platz finden (➥ 3-stufige Partitionierung)

Dynamische Bestimmung der Join-Parallelität*

- Dynamische Umverteilung ermöglicht hohe Flexibilität zur Lastbalancierung: **Wahl des Parallelitätsgrades p sowie der Join-Prozessoren**
  - Kostenmodell zur Bestimmung eines optimalen Parallelitätsgrades $p_{su\text{-}opt}$ im Einbenutzerbetrieb
  - Zuweisung zu den $p_{su\text{-}opt}$ Prozessoren mit der geringsten CPU-Auslastung
- **Paralleler Hash-Join: Berücksichtigung der Hauptspeicherverfügbarkeit**
  - dynamische Bestimmung der minimalen Rechneranzahl, welche temporäre E/A vermeidet (falls nicht möglich, Bestimmung der Konfiguration, die E/A minimiert)
  - erfordert Information zur aktuellen Hauptspeicherbelegung *AVAIL-MEMORY [1..#PE] of (node-ID, free)*; sortiert gemäß verfügbarem Hauptspeicher (free)
  - p_{mu} = MIN (k | AVAIL-MEMORY [k].free *k > b); b = Größe der kleineren Relation
  - Zuweisung zu den p Rechnern mit der maximalen Hauptspeicherverfügbarkeit
  - falls mehrere Konfigurationen temporäre E/A ausschließen sind oft höhere Parallelitätsgrade sinnvoll (z.B. Anlehnung an Einbenutzer-Optimum)

1) keine Überlauf-E/A

k	1	2	3	4
nodeID	P3	P2	P4	P1
free (MB)	45	40	35	30

2) Überlauf-E/A erforderlich

k	1	2	3	4
nodeID	P2	P1	P4	P3
free (MB)	80	10	0	0

b = 100 MB

* Rahm, E.; Marek, R.: Dynamic Multi-resource Load Balancing in Parallel Database Systems. Proc. 21st VLDB Conf., 1995

Mehr-Wege-Joins

- **Mehr-Wege-Joins realisierbar als Folge von Zwei-Wege-Joins**

- **Bestimmung der Join-Reihenfolge ist ein zentrales Problem**

- **Minimierung der Zwischenergebnisse liefert auch bei "Ship-Whole" nicht unbedingt die geringsten Kommunikationskosten**

- **Beispiel**

  Zu berechnen: $R \bowtie S \bowtie T$

  Card (R) = 10000, *Card (S)* = 1000, *Card (T)* = 10000,

  Joinselektivitäten JSF ($R \bowtie S$) = 0.01, *JSF* ($S \bowtie T$) = 0.1

  jede der drei Relationen habe 10 Attribute

  1. $R \to K_S$; $Z := R \bowtie S$; $Z \to K_T$. Berechnung des Ergebnisses $Z \bowtie T$

  2. $S \to K_R$; $Z := R \bowtie S$; $Z \to K_T$. Berechnung des Ergebnisses $Z \bowtie T$

  3. $S \to K_T$; $Z := S \bowtie T$; $Z \to K_R$. Berechnung des Ergebnisses $Z \bowtie R$

  4. $T \to K_S$; $Z := S \bowtie T$; $Z \to K_R$. Berechnung des Ergebnisses $Z \bowtie R$

  5. $R \to K_S$; $T \to K_S$. Berechnung des Ergebnisses $R \bowtie S \bowtie T$

  6. $S \to K_R$; $T \to K_R$. Berechnung des Ergebnisses $R \bowtie S \bowtie T$

Mehr-Wege-Joins (2)

■ **Reduzierung des Kommunikationsaufwandes durch Semi-Joins:**

- Tupel qualifiziert sich nur, wenn es Verbundpartner in allen beteiligten Relationen findet
- Reduzierungspotential steigt mit N (= Anzahl der Relationen)

■ ***Vollständige Reduzierung (full reducer):***
Folge von Semi-Joins, welche jede der beteiligten Relationen auf diejenigen Tupel reduzieren kann, die im Endergebnis enthalten sind

R

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

$\bowtie$

S

B	C
9	8
1	5
9	4
4	3
4	2
5	7

$\bowtie$

T

C	D
1	8
6	1
4	2
7	3
2	6
3	8

R $\bowtie$ **S** $\bowtie$ **T**

A	B	C	D
4	5	7	3

Mögliche Folge von Semi-Joins:

R' := R $\ltimes$ S

S' := S $\ltimes$ T

T' := T $\ltimes$ S

Ergebnis: R' $\bowtie$ S' $\bowtie$ T'

➥ *keine vollständige Reduzierung*

Mehr-Wege-Joins (3)

- **Vollständige Reduzierungen** bestehen i. allg. nur für azyklische Join-Anfragen, insbesondere für *gekettete Joins* (chained queries)

  $$R_1\ (A, A_1) \bowtie R_2\ (A_1, A_2) \bowtie\ \ldots\ \bowtie\ R_N\ (A_{N-1}, B)$$

  vollständige Ordnung über Join-Attributen A_i gegeben

- **In diesem Fall besteht die vollständige Reduzierung aus 2N-2 Semi-Joins:**
  - Vorwärtsreduzierung von R_1 bis R_N
    $R_2' := R_2 \ltimes R_1$
    $R_3' := R_3 \ltimes R_2'$
    • • •
    $R_N' := R_N \ltimes R_{N-1}'$
  - Rückwärtsreduzierung von R_N' bis R_1
    $R_{N-1}'' := R_{N-1} \ltimes R_N'$
    $R_{N-2}'' := R_{N-2} \ltimes R_2''$
    • • •
    $R_1' := R_1 \ltimes R_2''$

- **Beispiel**

R

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

S

B	C
9	8
1	5
9	4
4	3
4	2
5	7

T

C	D
1	8
6	1
4	2
7	3
2	6
3	8

R $\xrightarrow{7, 1, 6, 5, 2}$ S $\xrightarrow{5, 7}$ T

R $\xleftarrow{5}$ S $\xleftarrow{7}$ T

Vollständige Reduzierung:

$S' := S \ltimes R$

$T' := T \ltimes S'$

$S'' := S' \ltimes T'$

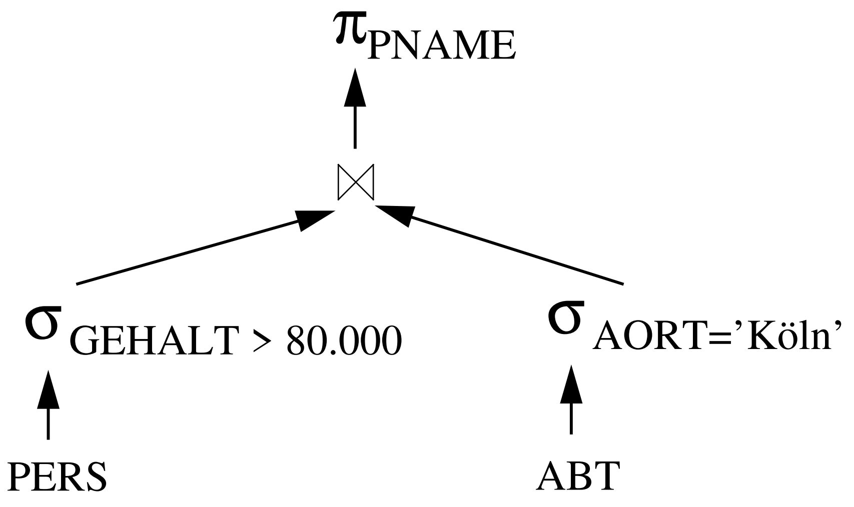
$R' := R \ltimes S''$

R ⋈ S ⋈ T

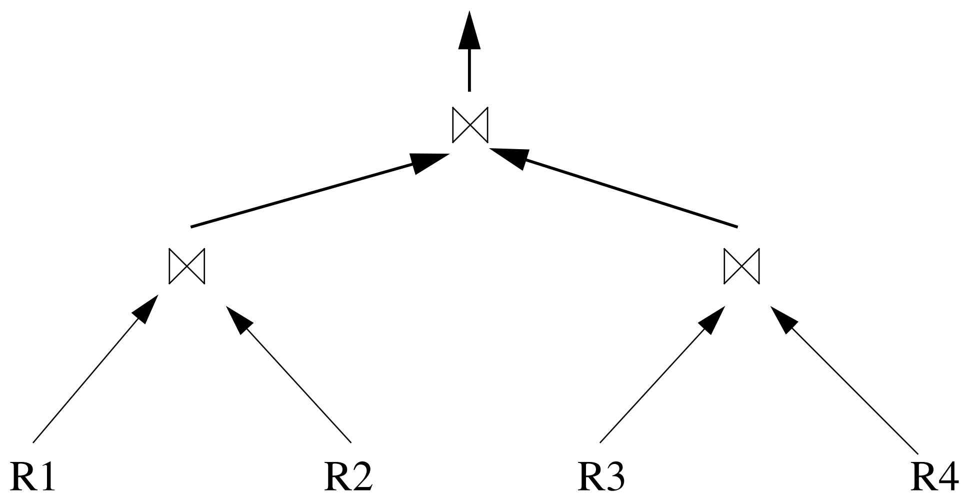
A	B	C	D
4	5	7	3

Inter-Operator-Parallelität

- **Analyse des Operatorbaumes zur parallelen Berechnung unabhängiger Knoten (Operatoren)**
- **Beispiele:**



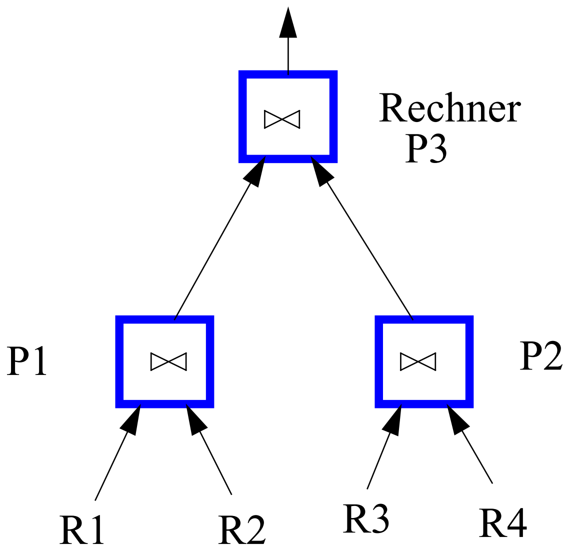
Parallele Berechnung verschiedener Selektionen



Mehr-Wege-Join
(R1 ⋈ R2 ⋈ R3 ⋈ R4)

Mehr-Wege-Joins:
Nutzung von Pipeline-Parallelität

- **Beispiel:**
  Pipeline-Join in Rechner P3



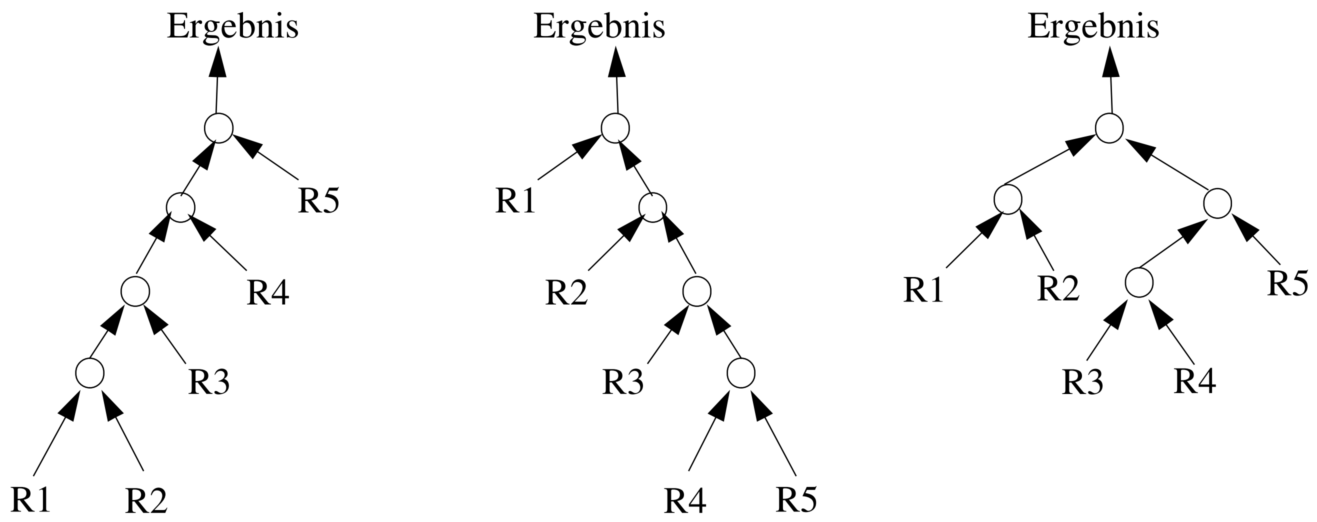
- **Annahmen:**
  - P3 hat Eingangswarteschlange Q für Tupel aus P1 und P2
  - spezielle Nachrichten ENDP1, ENDP2 zeigen an, daß keine weiteren Tupel mehr folgen

- **Algorithmus:**

```
Fertig1, Fertig2 := FALSE;
Von1, Von2, Ergebnis := { }
WHILE NOT (Fertig1 AND Fertig2) DO
BEGIN
    IF Q leer THEN warte bis neue Tupel eintreffen;
    t := erstes Tupel in Q;
    IF t = ENDP1 THEN Fertig1 := TRUE
    ELSE IF t=ENDP2 THEN Fertig2 := TRUE
        ELSE IF t von P1 THEN BEGIN
                Von1 := Von1 ∪ {t};
                Ergebnis := Ergebnis ∪ ( {t} ⋈ Von2 ) END
        ELSE BEGIN (* t von P2 *)
                Von2 := Von2 ∪ {t};
                Ergebnis := Ergebnis ∪ ( {t} ⋈ Von1 ) END
END; (* WHILE *)
```

Mehr-Wege-Joins (2)

- **Beschränkte Berechnungsreihenfolgen** für Mehrwege-Joins erleichtern Optimierungsproblem



links-tiefer Operatorbaum *rechts-tiefer Operatorbaum* *unbeschränkt (bushy tree)*

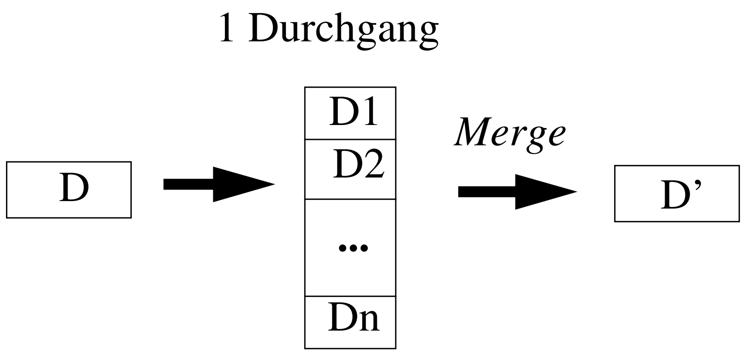
- **Rechts-tiefe Bäume** erlauben bei Verwendung von Hash-Joins einen hohen Grad an Parallelität,
  jedoch mit hohem Ressourcenbedarf
  - Join-Berechnung in 2 Schritten
  - Building Phase: paralleles Einlesen von N-1 Relationen in eine Hauptspeicher-Hashtabelle
  - Probing-Phase beginnt mit der N-ten Relation an einem der Rechner, danach wird sie mit den jeweiligen Join-Resultaten sukzessive an den anderen Knoten fortgeführt

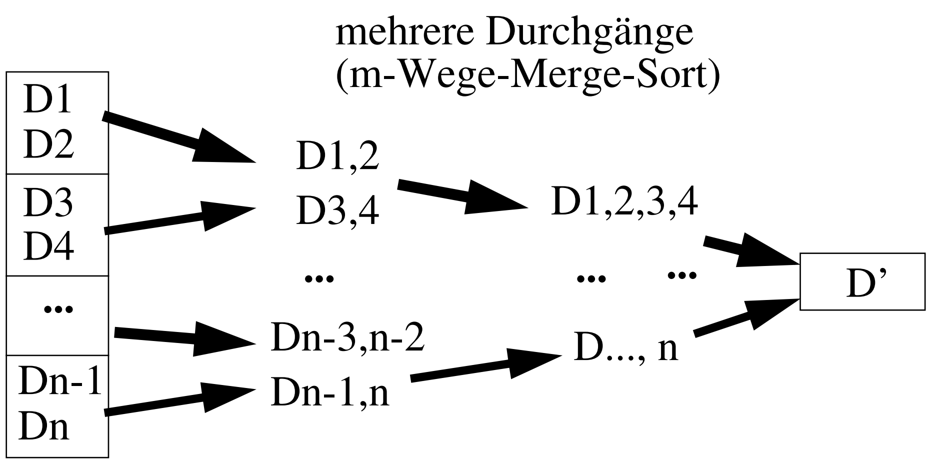
- **Links-tiefe Bäume**
  - Join-Berechnung in N Schritten
  - pro Schritt werden nur zwei Relationen im Hauptspeicher gehalten

Parallele Sortierung

■ **DBS: Externes Sortieren**

- Zerlegung der Eingabe in mehrere Läufe (runs)
- Sortieren und Zwischenspeichern der sortierten Läufe
- Sukzessives Mischen bis ein sortierter Lauf entsteht





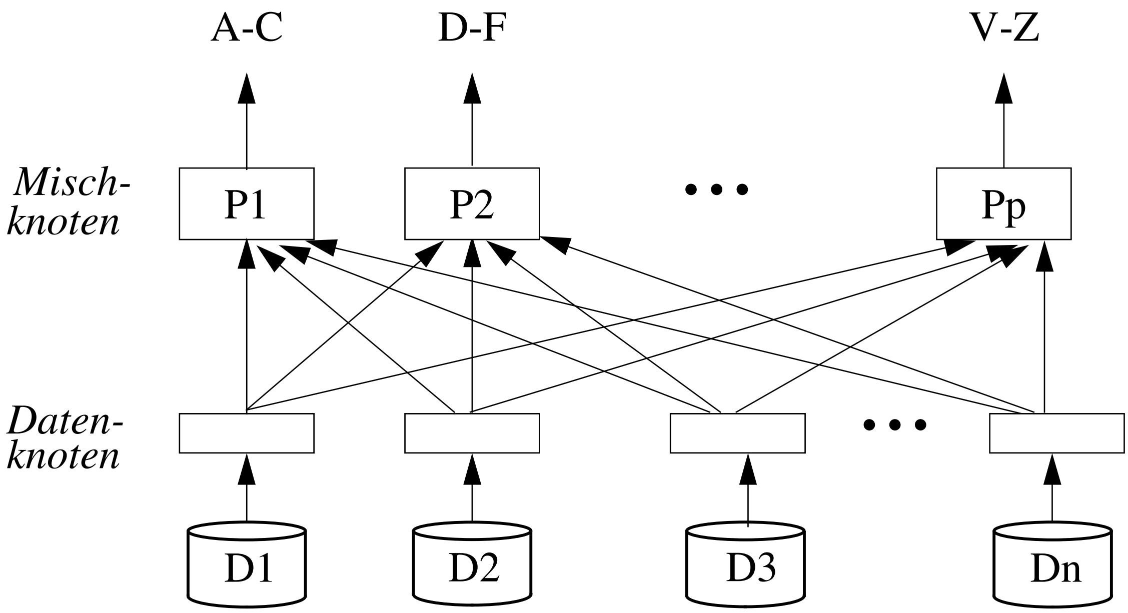
■ **Anforderungen an parallele Sortierung**

- parallele Eingabe (multiple input)
- parallele Sortierphasen
- paralleles Mischen
- Partitionierung der sortierten Ausgabe (multiple output)

Parallele Sortierung (2)

■ **Ansatz**

- lokale Sortierung der Partitionen an den Datenknoten
- dynamische Umverteilung der sortierten Läufe unter p Mischknoten
- Umverteilung wird über eine dynamische Bereichsfragmentierung auf dem Sortierattribut gesteuert
- paralleles Mischen in den p Mischknoten
- partitionierte Ausgabe



Zusammenfassung

- **Prinzipielle Vorgehensweise**
  - wie in zentralisierten DBS
  - jedoch zusätzliche Optimierungsschritte zur Übersetzungs- und Laufzeit
- **Weit höhere Komplexität der verteilten Anfrageverarbeitung** als im zentralisierten Fall
  - Berücksichtigung der Fragmentierung
  - Nutzung von Replikaten
  - Optimierung der Objektverteilung und der Verarbeitungsorte (Minimierung des Kommunikationsaufwandes)

  ➥ Zusätzliche Kosten: #Nachrichten + zu übertragende Daten
- **Arten der Optimierung**
  - Behandlung von Fragmentanfragen
  - Join-Berechnung
    "Ship Whole" oft dem "Fetch as Needed" vorzuziehen
  - Bestimmung der Join-Reihenfolge ist ein zentrales Problem
- **Anfrageoptimierung und -bearbeitung**
  - global am Koordinatorknoten und
  - lokal an den Speicherknoten

  ➥ Optimierungsannahmen noch problematischer als im zentralisierten Fall

Zusammenfassung (2)

- **Verteilte und parallele Anfragebearbeitung**
  - Anfragetransformation, algebraische Optimierung
  - Daten-Lokalisierung: Abbildung von Operatoren auf Fragmente
  - Globale Optimierung: Kostenbewertung unter Berücksichtigung von Kommunikationsaufwand und Parallelsierungsalternativen
- **Selektion, Projektion, Aggregationen**
  - Datenallokation bestimmt Ausführungsort (Shared Nothing)
  - Parallelisierbarkeit durch horizontale Fragmentierung
- **Verteilte Join-Verarbeitung**
  - Alternativen bezüglich Wahl des Join-Knotens und Übertragung der Daten
  - Semi-Join und Bitvektor-Join effektiv nutzbar
- **Parallele Join-Verarbeitung**
  - hohes Lastbalancierungsaufwand bei dynamischer Umverteilung der Relationen
  - Kommunikationseinsparungen falls Join-Attribut = Verteilattribut (insbesondere bei abhängiger horizontaler Fragmentierung)
  - paralleler Hash-Join: Optimierung der Hauptspeichernutzung
- **Mehr-Wege-Joins**
  - Kommunikationsreduzierung durch mehrfache Semi-Joins
  - Nutzung von Inter-Operator-Parallelität, insbesondere Pipeline-Parallelität