4. Datenallokation in verteilten und parallelen DBS

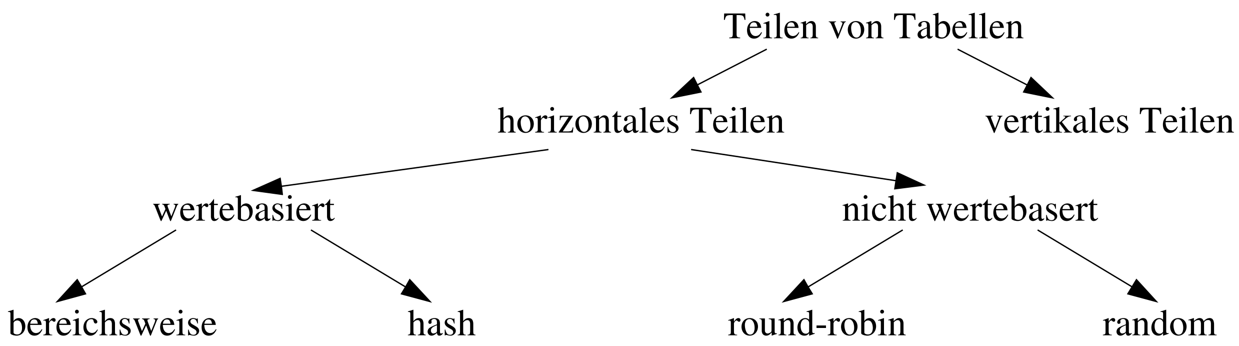
- **Fragmentierung und Allokation**
- **Anforderungen**
  - *Vollständigkeit*: jedes Datenelement muß in wenigstens einem Fragment enthalten sein
  - *Rekonstruierbarkeit*: Verlustfreiheit der Zerlegung
  - (weitestgehende) *Disjunktheit*
- **Fragmentierungsvarianten**
  - horizontale Fragmentierung
  - vertikale Fragmentierung
  - hybride Fragmentierung
- **Fragmentierungstransparenz**
- **Bestimmung einer Datenallokation für VDBS**
- **Datenallokation für Parallele DBS**
  - Verteilgrad
  - Varianten der horizontalen Fragmentierung (Round Robin, Hash, Bereichsfragmentierung: einfach, verfeinert, mehrdimensional)
  - Allokation der Fragmente
  - Datenallokation bei Shared Disk / Shared Everything

Fragmentierung

■ **Gründe für eine Fragmentierung**

- Lastbalancierung
- Nutzung von Lokalität
- Reduzierung des Verarbeitungsumfangs (Anfrageoptimierung!)
- Unterstützung von Parallelverarbeitung

■ **Klassifikation der Fragmentierung**



■ **Eigenschaften wertebasierter Verfahren**

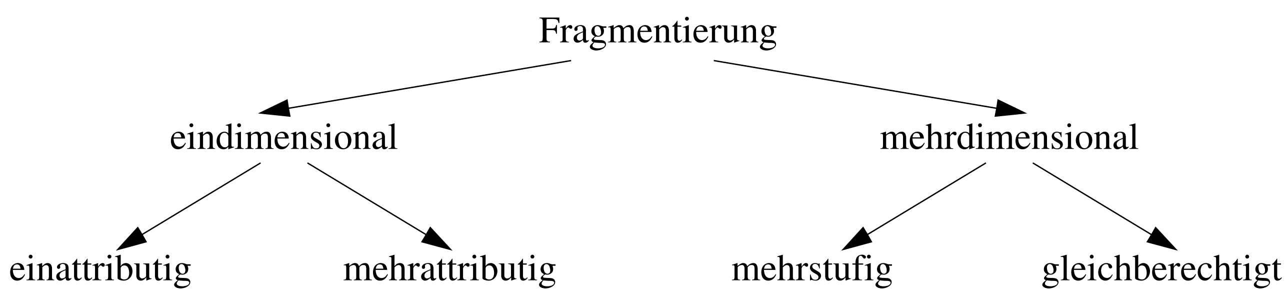
- Zuordnung ist auch im Nachhinein (durch DBS) nachvollziehbar und
- kann bei Anfrageoptimierung berücksichtigt werden
- Verteilung erfolgt über die Werte eines oder mehrerer Attribute
- Hash erlaubt (angenähert) Bestimmung der Fragmentgröße und unterstützt Exact-Match-Anfragen
- Bereichsfragmentierung hat die größte Bedeutung und unterstützt Bereichsanfragen (z.B. beim Data Warehousing)

■ **Eigenschaften nicht wertebasierter Verfahren**

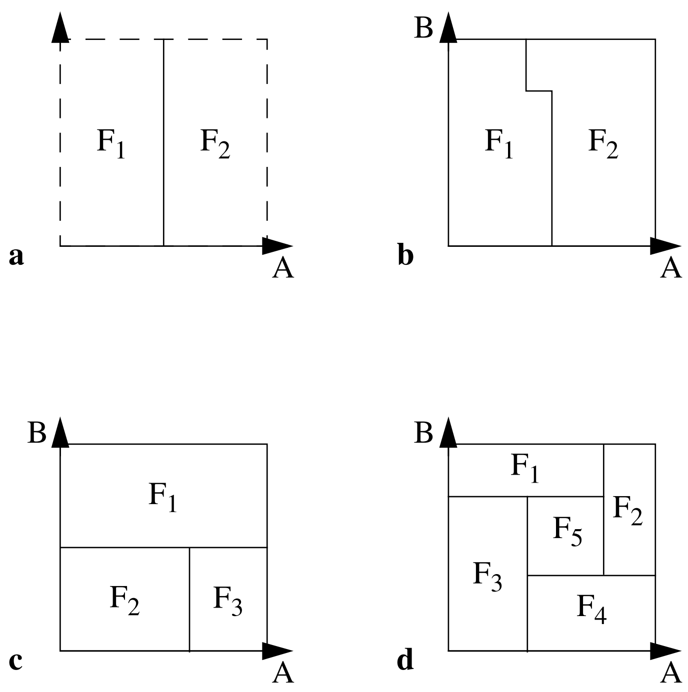
- Zuordnung eines Tupels zum Fragment ist in der Regel weder wiederholbar noch im Nachhinein nachvollziehbar
- Round-Robin verteilt Tupel in Einfügereihenfolge auf Fragemente, während Random dies zufällig tut
- Beide erzielen eine Gleichverteilung und damit eine gute Lastbalancierung
  ➥ ***Verfahren zielen auf parallele Anfrageverarbeitung ab***

Fragmentierung (2)

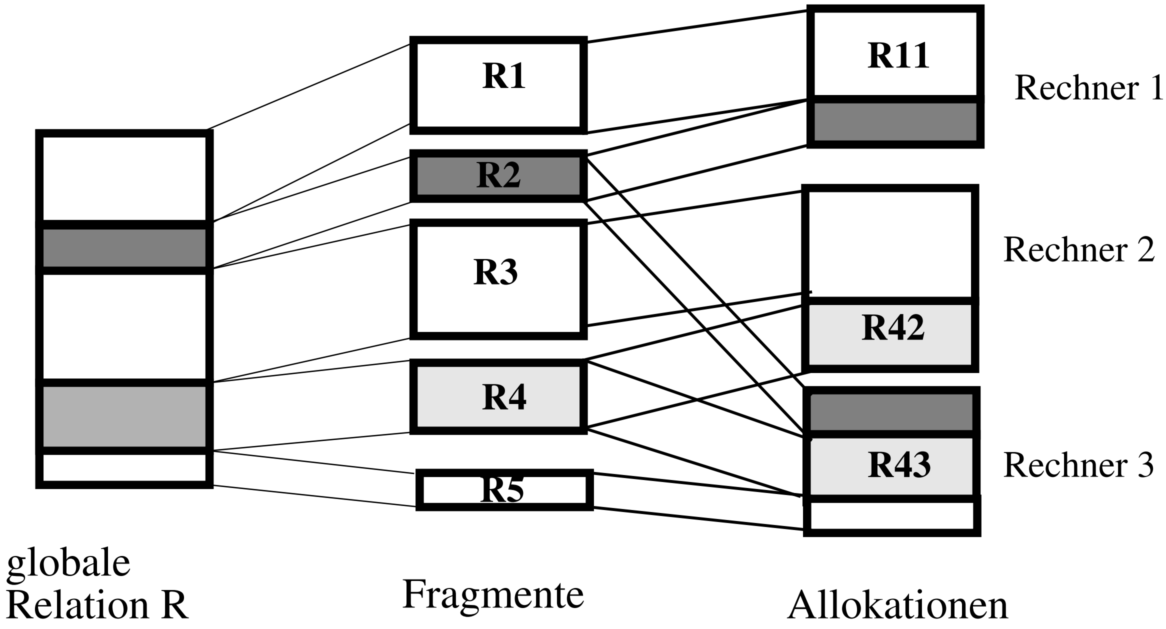
■ Strategien der Bereichsfragmentierung



- eindimensional: Es werden nur Anfragen auf dem (zusammengesetzten) Fragmentierungsattribut unterstützt
- mehrdimensional: Anfragen, die wenigstens eines der Fragmentierungs-attribute besitzen, können unterstützt werden
  - mehrstufig: die Reihenfolge des Anwenders der Fragmentierungsfunktion auf die Fragmentierungsattribute ist entscheidend
  - gleichberechtigt: Fragmentierung erfolgt vollkommen wahlfrei



Bestimmung der Datenverteilung



■ **Fragmentierung:**

- ***Fragmente:*** Einheiten der Datenverteilung
- oft: ganze Relationen bzw. sogar Datenbanken
- wünschenswert: Teile von Relationen
  (horizontale und vertikale Fragmentierung)

■ **Allokation von Fragmenten:**

- bestimmt weitgehend Ausführungsort von DB-Operationen
- Erstellung für "durchschnittliches" Lastprofil mittels Heuristiken
- aufwendige Umverteilungen
- widersprechende Teilziele:
  Minimierung der Kommunikationskosten vs. Lastbalancierung

■ **Replizierte Speicherung von Fragmenten**

- Partitionierung, partielle oder volle Replikation
- erhöhte Freiheitsgrade bei Anfrageoptimierung
- hoher Änderungsaufwand

Horizontale Fragmentierung

■ **Zeilenweise Aufteilung von Relationen**

KNR	*NAME*	*GEBDAT*	*Filiale*
K2	Schulz	2.11.1976	F
K4	Meier	23.8.1972	KL
K3	Müller	4.7.1987	KL
K1	Scholz	24.4.1959	F
K5	Weber	17.3.1942	L

Globale Relation KUNDE

KNR	*NAME*	*GEBDAT*	*Filiale*
K5	Weber	17.3.1942	L

KUNDE1 = $\sigma_{\textit{Filiale = "L"}}$ ***(KUNDE)***

KNR	*NAME*	*GEBDAT*	*Filiale*
K2	Schulz	2.11.1976	F
K1	Scholz	24.4.1959	F

KUNDE2 = $\sigma_{\textit{Filiale = "F"}}$ ***(KUNDE)***

KNR	*NAME*	*GEBDAT*	*Filiale*
K4	Meier	23.8.1972	B
K3	Müller	4.7.1987	B

KUNDE3 = $\sigma_{\textit{Filiale = "KL"}}$ ***(KUNDE)***

■ Definition der Fragmentierung durch Selektionsprädikate Pi auf der Relation:

$$\mathbf{Ri := \sigma_{Pi}\ (R) \qquad (1 \le i \le n)}$$

■ **Vollständigkeit:** Jedes Tupel ist einem Fragment eindeutig zugeordnet

■ Fragmente sind **disjunkt**: $\mathbf{Ri \cap Rj = \{\}\quad (i \neq j))}$

■ **Verlustfreiheit:** Relation ist Vereinigung aller Fragmente

$$\mathbf{R = \cup\, Ri \qquad (1 \le i \le n)}$$

Abgeleitete horizontale Fragmentierung

- Fremdschlüssel-Primärschlüssel-Beziehungen definieren Abhängigkeiten zwischen Relationen

- Abgeleitete (abhängige) Fragmentierung: Partitionierung der abhängigen Relation wird auf Vater-Relation abgestimmt

KTONR	*KNR*	*KTOSTAND*
1234	K2	122,34
2345	K4	-12,43
3231	K1	1222,22
7654	K5	63,79
9876	K2	55,77
5498	K4	-4506,77

Globale Relation KONTO

KTONR	*KNR*	*KTOSTAND*
7654	K5	63,79

KONTO1 = KONTO $\ltimes$ ***KUNDE1***

KTONR	*KNR*	*KTOSTAND*
1234	K2	122,34
3231	K1	1222,22
9876	K2	55,77

KONTO2 = KONTO $\ltimes$ ***KUNDE2***

KTONR	*KNR*	*KTOSTAND*
2345	K4	-12,43
5498	K4	-4506,77

KONTO3 = KONTO $\ltimes$ ***KUNDE3***

KONTO von KUNDE abhängig (KNR)

KUNDE-Partitionierung (über FILIALE) bestimmt KONTO-Partitionierung

- **Fragmentdefinition** über Semi-Join zwischen abhängiger Relation S und Vater-Relation R:

$$\mathbf{Si = S \ltimes Ri = S \ltimes \sigma_{Pi}(R)}$$

$$\mathbf{= \pi_{S\text{-}Attribute}(S \bowtie \sigma_{Pi}(R)) \qquad (1 \le i \le n)}$$

- **Joins zwischen Vater- und Sohn-Relationen** können lokal berechnet werden

Vertikale Fragmentierung

- **Spaltenweise Aufteilung von Relationen**

Globale Relation KUNDE

KNR	*NAME*	*GEBDAT*	*Filiale*
K2	Schulz	2.11.1976	F
K4	Meier	23.8.1972	KL
K3	Müller	4.7.1987	KL
K1	Scholz	24.4.1959	F
K5	Weber	17.3.1942	L

KNR	*NAME*	*Filiale*
K2	Schulz	F
K4	Meier	KL
K3	Müller	KL
K1	Scholz	F
K5	Weber	L

KUNDE1 =
$\pi_{KNR,\ NAME,\ Filiale}$ ***(KUNDE)***

KNR	*GEBDAT*
K2	2.11.1976
K4	23.8.1972
K3	4.7.1987
K1	24.4.1959
K5	17.3.1942

KUNDE2 =
$\pi_{KNR,\ GEBDAT}$ ***(KUNDE)***

- Definition der Fragmentierung durch Projektion

- **Vollständigkeit:** jedes Attribut in wenigstens 1 Fragment enthalten

- **Verlustfreie Zerlegung:**
  - Primärschlüssel i. allg. in jedem Fragment enthalten
  - JOIN-Operation zur Rekonstruktion des gesamten Tupels

Hybride Fragmentierung

■ **Kombination von horizontaler und vertikaler Fragmentierung**

Globale Relation KUNDE

KNR	*NAME*	*GEBDAT*	*Filiale*
K2	Schulz	2.11.1976	F
K4	Meier	23.8.1972	KL
K3	Müller	4.7.1987	KL
K1	Scholz	24.4.1959	F
K5	Weber	17.3.1942	L

KUNDE1 = $\pi_{\text{KNR, NAME, Filiale}}$ ***(KUNDE)***

KNR	*NAME*	*Filiale*
K2	Schulz	F
K4	Meier	KL
K3	Müller	KL
K1	Scholz	F
K5	Weber	L

KUNDE2 = $\pi_{\text{KNR, GEBDAT}}$ ***(KUNDE)***

KNR	*GEBDAT*
K2	2.11.1976
K4	23.8.1972
K3	4.7.1987
K1	24.4.1959
K5	17.3.1942

KUNDE11 = $\sigma_{\textit{Filiale = "L"}}$ ***(KUNDE1)***

KNR	*NAME*	*Filiale*
K5	Weber	L

KUNDE12 = $\sigma_{\textit{Filiale = "F"}}$ ***(KUNDE1)***

KNR	*NAME*	*Filiale*
K2	Schulz	F
K1	Scholz	F

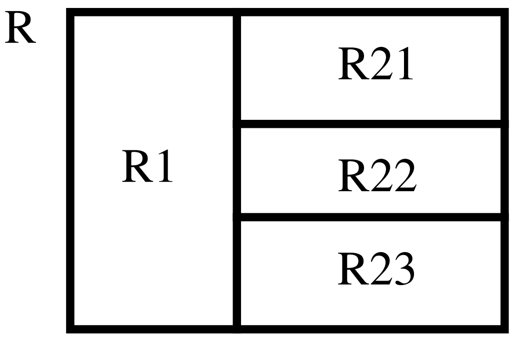
KUNDE13 = $\sigma_{\textit{Filiale = "KL"}}$ ***(KUNDE1)***

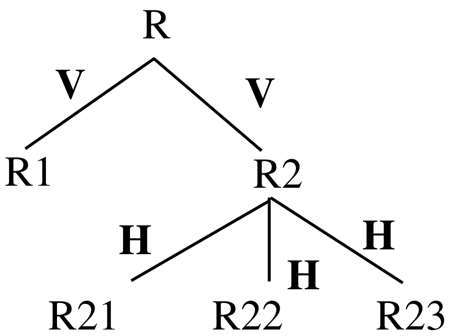
KNR	*NAME*	*Filiale*
K4	Meier	KL
K3	Müller	KL

KUNDE = (KUNDE11 $\cup$ KUNDE12 $\cup$ KUNDE13) $\bowtie$ KUNDE2

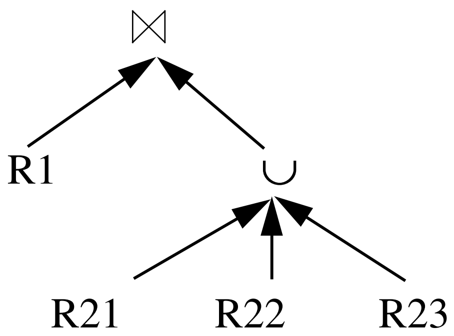
Hybride Fragmentierung (2)

a) vertikale gefolgt von horizontaler Partitionierung



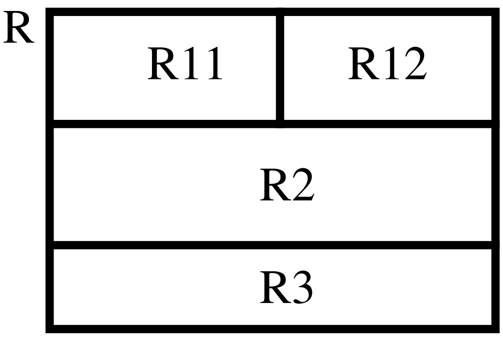


Fragmentierungsbaum



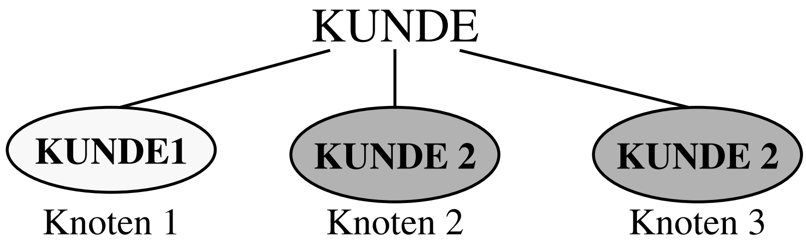
Operatorbaum (Rekonstruktion)

b) horizontale gefolgt von vertikaler Partitionierung



Fragmentierungstransparenz

■ **Beispiel 1:**



Die Relation KUNDE wurde horizontal in zwei Fragmente zerlegt,
wobei Fragment KUNDE2 an zwei Knoten repliziert gespeichert ist

■ **Keine Transparenz:**

```
SELECT      NAME INTO :NAME
FROM        KUNDE1@Knoten1
WHERE       KNR = :KNO
IF NOT FOUND THEN
            SELECT      NAME INTO :NAME
            FROM        KUNDE2@Knoten3
            WHERE       KNR = :KNO
            . . .
```

■ **Orts- und Replikationstransparenz:**

Weglassen der Ortsbezeichnungen (@Knoten)

■ **Orts-, und Replikations- und Fragmentierungstransparenz:**

```
SELECT      NAME INTO :NAME
FROM        KUNDE
WHERE       KNR = :KNO
```

Fragmentierungstransparenz (2)

■ **Beispiel 2:**

	KUNDE1	KUNDE2	FILIALE = "KL"
KUNDE	KUNDE3	KUNDE4	FILIALE $\neq$ "KL"

KUNDE1 (KNR, NAME)
KUNDE3 (KNR, NAME, FILIALE)
KUNDE2 (KNR, GEBDAT, FILIALE)
KUNDE4 (KNR, GEBDAT)

KUNDEi sei an Knoten #i gespeichert
KUNDE1 zusätzlich an Knoten 5 und KUNDE4 an Knoten 6

Orts-, Replikations- und Fragmentierungstransparenz:

```
UPDATE  KUNDE
   SET FILIALE = "KL"
   WHERE  KNR = "K3";   (*Wechsel  nach KL *)
```

Orts- und Replikationstransparenz:

```
SELECT  NAME  INTO :Name    FROM KUNDE3    WHERE  KNR = "K3";
SELECT  GEBDAT  INTO :Geb    FROM KUNDE4    WHERE  KNR = "K3";

INSERT INTO KUNDE1 (KNR, NAME)  VALUES ("K3", :Name)
INSERT INTO KUNDE2 (KNR, GEBDAT, FILIALE)  VALUES ("K3", :Geb, "KL")

DELETE  KUNDE3  WHERE KNR = "K3";
DELETE  KUNDE4  WHERE KNR = "K3";
```

Keine Transparenz:

```
SELECT  NAME  INTO :Name    FROM KUNDE3@Knoten3  WHERE  KNR = "K3";
SELECT  GEBDAT  INTO :Geb  FROM KUNDE4@Knoten4 WHERE  KNR = "K3";

INSERT INTO KUNDE1@Knoten1 (KNR, NAME)    VALUES ("K3", :Name)
INSERT INTO KUNDE1@Knoten5 (KNR, NAME)    VALUES ("K3", :Name)
INSERT INTO KUNDE2@Knoten2 (KNR, GEBDAT, FILIALE)
                                              VALUES ("K3", :Geb, "KL")

DELETE  KUNDE3@Knoten3 WHERE KNR = "K3";
DELETE  KUNDE4@Knoten4  WHERE KNR = "K3";
DELETE  KUNDE4@Knoten6 WHERE KNR = "K3";
```

Bestimmung der Datenallokation

■ **Widersprüchliche Ziele** (vor allem bei Replikation)**:**

- Lese- vs. Änderungsoperationen
- Lokalität vs. Parallelität
- Lokalität vs. Lastbalancierung

■ **Optimierungsziele:**

- Unterstützung kurzer Antwortzeiten bzw. eines hohen Durchsatzes
- Minimierung des Kommunikationsbedarfs
- Lastbalancierung usw.

■ **Mathematisches Modell:**

- Minimierung einer Kostenfunktion unter
- Einhaltung von Randbedingungen

■ **Einfaches Modell:**

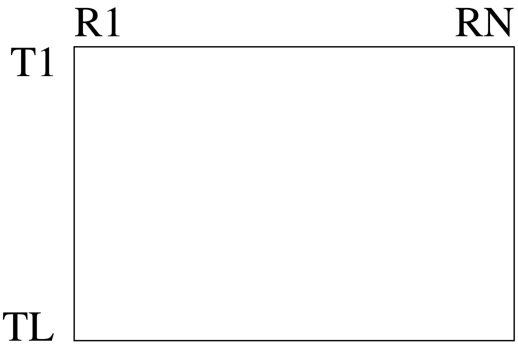
- **Hauptziel:** Minimierung von Kommunikation, d. h. Maximierung der lokalen Verarbeitung
- **Nebenbedingung:** Auslastung einzelner Rechner soll Grenzwert nicht überschreiten
- **Prinzipielle Vorgehensweise:** partitionierte Allokation von Fragmenten (hier: keine Replikation)

Bestimmung der Datenallokation (2)

- **Parameter**
  - *L*: Anzahl der Transaktionstypen
  - *N*: Anzahl der Rechner (R), *M*: Anzahl der Fragmente (F)
  - *I-ref*: #Instruktionen pro (lokaler) Referenz
  - *I-komm*: #Instruktionen zur Kommunikation (pro externer Referenz)
  - *C (n):* CPU-Kapazität von Rechner n   ($1 \leq n \leq N$, in Mips)
  - *u-max:* maximale CPU-Auslastung   ($0 < \text{u-max} < 1$)

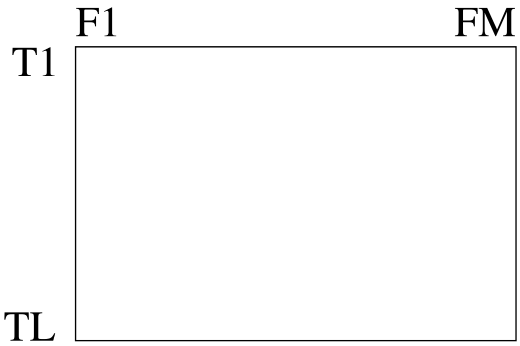
- **Eingabe**

Lastverteilungsmatrix W



TA pro Sek.

Referenzmatrix R



Zugriffe pro Sek.

- **Ausgabe:**

Allokationsmatrix A

A (n,m) = 1 (0), falls Fragment m Knoten n (nicht) zugeordnet

Bestimmung der Datenallokation (2)

- **Zugriffsfrequenz auf Fragment m**

$$ZF(\mathbf{m}) = \sum_{n=1}^{N} \sum_{l=1}^{L} W(n, l) \times R(l, \mathbf{m})$$

- **Häufigkeit lokaler Zugriffe auf Fragment m**

$$ZFL(\mathbf{m}) = \sum_{n=1}^{N} \sum_{l=1}^{L} W(n, l) \times R(l, \mathbf{m}) \times A(n, \mathbf{m})$$

- **Optimierungsziel:** $\sum_{m=1}^{N} ZFL(m) = Max\,!$

- **Nebenbedingung 1:** Jedes Fragment soll genau einem Rechner zugeordnet werden (keine Replikation)

$$\sum_{n=1}^{N} A(n, m) = 1 \qquad (\forall\, m \mid 1 \leq m \leq M)$$

- **Nebenbedingung 2:** Lastbalancierung

Bestimmung der Datenallokation (3)

- **Lastanteil L1:** Fragmentzugriffe an Rechner n

$$L1\,(\mathbf{n}) = \text{l-ref} \times \sum_{m=1}^{M} ZF\,(m) \times A\,(\mathbf{n}, m)$$

- **Lastanteil L2:** Kommunikationskosten auf lokale Datenfragmente von Rechner n durch externe Transaktionen

$$L2\,(\mathbf{n}) = \text{l-komm} \times \sum_{m=1}^{M} (ZF\,(m) - ZFL(m)) \times A\,(\mathbf{n}, m)$$

- **Lastanteil L3:** Kommunikationskosten lokaler Transaktionen von Rechner n für externe Datenzugriffe

$$L3\,(\mathbf{n}) = \text{l-komm} \times \sum_{m=1}^{M} \sum_{l=1}^{L} W\,(\mathbf{n},l) \times R(l,m) \times (1-A\,(\mathbf{n}, m))$$

- **Nebenbedingung 2:** Lastbalancierung

$$L1\,(n) + L2\,(n) + L3\,(n) < \text{u-max} \times C\,(n) \qquad (\forall\, n \mid 1 \leq n \leq N)$$

Bestimmung der Datenallokation (4)

■ **Heuristische Berechnung der Allokationsmatrix A**

(Hilfsvariable CU_n: CPU-Auslastung von Rechner n aufgrund bereits vorgenommener Zuordnungen)

1. Berechne aus der Lastverteilung W und der Referenzverteilung R zunächst für jedes Fragment m die Zugriffshäufigkeit *ZF (m)*.

2. Bestimme das nächste, noch nicht allokierte Fragment m mit der höchsten Zugriffsfrequenz *ZF (m).*

3. Berechne für jeden Rechner die lokale Zugriffsfrequenz *ZFL (m)*, die sich durch Allokation von Fragment m an diesen Rechner ergibt, ebenso die sich daraus ergebende Erhöhung der CPU-Auslastung.

4. Ordne das Fragment demjenigen Rechner zu, für den sich der größte Anteil lokaler Zugriffe ergibt, ohne daß sich ein Überschreiten der maximalen CPU-Auslastung ergibt. Ergänze die Allokationsmatrix *A* und die *CU*-Werte für die getroffene Allokation.

5. Falls noch weitere Fragmente zuzuordnen sind, gehe zu Schritt 2. Ansonsten ist die Berechnung der Allokation beendet.

Bestimmung der Datenallokation - Beispiel

W	R1	R2	R3
T1		16	8
T2	15	6	
T3	7		10

Lastverteilung W
(Aufrufhäufigkeiten pro Sekunde)

R	F1	F2	F3	F4
T1	70	30		10
T2	5		50	
T3	5			80

Referenzverteilung R

■ Bestimmung der Zugriffshäufigkeiten ZF (Schritt 1)

$$ZF(\boldsymbol{m}) = \sum_{n=1}^{N} \sum_{l=1}^{L} W(n, l) \times R(l, \boldsymbol{m})$$

m = 1: $0 \cdot 70 + 15 \cdot 5 + 7 \cdot 5$ (= 110 von R1)
$+ \; 16 \cdot 70 + 6 \cdot 5 + 0 \cdot 5$ (= 1150 von R2)
$+ \; 8 \cdot 70 + 0 \cdot 5 + 10 \cdot 5$ (= 610 von R3)
= ZF(m = 1) = <u>1870</u>

m = 2: $0 \cdot 30 + 15 \cdot 0 + 7 \cdot 0$ (= 0 von R1)
$+ \; 16 \cdot 30 + 6 \cdot 0 + 0 \cdot 0$ (= 480 von R2)
$+ \; 8 \cdot 30 + 0 \cdot 0 + 10 \cdot 0$ (= 240 von R3)
= ZF(m = 2) = <u>720</u>

m = 3: . . . (= 750 von R1)
(= 300 von R2)
(= 0 von R3)
= ZF(m = 3) = <u>1050</u>

m = 4: . . . (= 560 von R1)
(= 160 von R2)
(= 880 von R3)
= ZF(m = 4) = <u>1600</u>

Bestimmung der Datenallokation - Beispiel (2)

I-ref =100000, *I-komm* =25000, *C(n)* =300 Mips, *u-max* = 0.8

■ Zuordnung von Fragment F1 durch ZFL(1)

von R1 : 110
von R2 : 1150 ⇒ Zuordnung nach R2
von R3 : 610

L1(2) = $10^5 \cdot 1870$ = 187 Mips

L2(2) = $(1870 - 1150) \cdot 2{,}5 \cdot 10^4$ = 18 Mips

L3(1) = $110 \cdot 2{,}5 \cdot 10^4$ = 2,75 Mips von R1 → F1

L3(3) = $610 \cdot 2{,}5 \cdot 10^4$ = 15,75 Mips von R3 → F1

<u>Momentane Last:</u>

$$\left.\begin{array}{lcl} CU1 & = & 2{,}75 \text{ Mips} \\ CU2 & = & 205 \text{ Mips} \\ CU3 & = & 15{,}25 \text{ Mips} \end{array}\right\} \leq 240 \text{ Mips} = 0{,}8 \cdot 300 \text{ Mips}$$

■ Zuordnung von Fragment F4 durch ZFL(4)

von R1 : 560
von R2 : 160
von R3 : 880 ⇒ Zuordnung nach R3

<u>Momentane Last:</u>

$$\left.\begin{array}{lcl} CU1 & = & 16{,}75 \text{ Mips} \\ CU2 & = & 209 \text{ Mips} \\ CU3 & = & 193{,}25 \text{ Mips} \end{array}\right\} \leq 240 \text{ Mips}$$

Bestimmung der Datenallokation - Beispiel (3)

■ Zuordnung von Fragment F3 durch ZFL(3)

von R1 : 750 ⇒ Zuordnung nach R1
von R2 : 300
von R3 : 0

Momentane Last:

$$\left.\begin{array}{lcl} CU1 & = & 129{,}25 \text{ Mips} \\ CU2 & = & 216{,}50 \text{ Mips} \\ CU3 & = & 193{,}25 \text{ Mips} \end{array}\right\} \leq 240 \text{ Mips}$$

■ Zuordnung von Fragment F2 durch ZFL(2)

von R1 : 0 ⇒ Zuordnung nach R1
von R2 : 480 ⇒ Überlast
von R3 : 240 ⇒ Überlast

Last:

$$\left.\begin{array}{lcl} CU1 & = & 219{,}25 \text{ Mips} \\ CU2 & = & 228{,}50 \text{ Mips} \\ CU3 & = & 199{,}25 \text{ Mips} \end{array}\right\} \leq 240 \text{ Mips}$$

■ Allokationsmatrix A

	F1	F2	F3	F4
R1	0	I	I	0
R2	I	0	0	0
R3	0	0	0	I

Datenallokation in PDBS

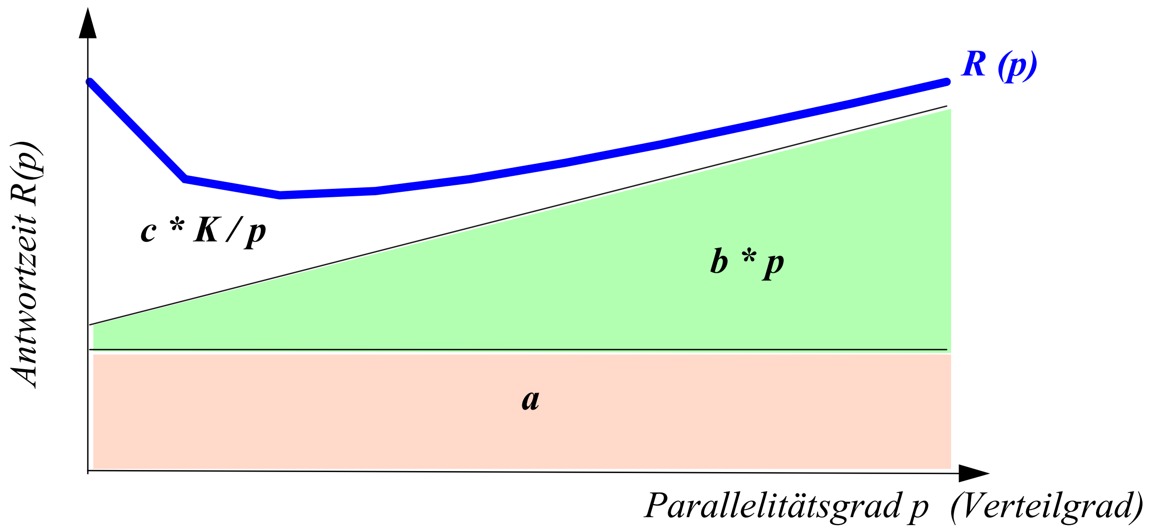
- **zunächst: Shared Nothing**
- **Horizontale Verteilung von Relationen über mehrere Rechner**
  - Nutzung von Datenparallelität
  - Verbesserung von Bandbreite und E/A-Rate (E/A-Parallelität)
  - Lastbalancierung
- **Teilaufgaben zur Bestimmung der Datenverteilung**
  - Festlegung des Verteilgrades einer Relation
    Der Verteilgrad (degree of declustering) D einer Relation legt fest, über wieviele Rechner (Platten) diese verteilt wird: Er bestimmt v. a. die Leistungsfähigkeit von Selektionsoperationen
  - Fragmentierung
  - Allokation
- **Bestimmung des "optimalen" Verteilgrades schwierig**
  - Kommunikations-Overhead steigt mit Rechneranzahl
  - Parallelisierungsgewinn sinkt mit wachsender Rechneranzahl, v.a. für kleinere Relationen
  - "Full Declustering" oft nicht sinnvoll
  - einfache Anfragen (kleine Relationen) sollen zur Reduzierung des Kommunikationsaufwandes auf möglichst wenige Partitionen beschränkt werden
  - datenintensive Anfragen (große Relationen) sollen zur Nutzung von Parallelität/Lastbalancierung auf größere Anzahl von Partitionen verteilt werden

Bestimmung des Verteilgrades einer Relation

- **Mathematisches Modell (K = Kardinalität der Relation):**

  a = konstanter Anteil (Initialisierungskosten u. ä.)
  b = Kosten zum Starten und Beenden einer Teiloperation
  c = Einheitskosten (Nutzarbeit c*K)
  (c*K)/p = Kosten pro Verarbeitungsort

Antwortzeit $R(p) = a + b \times p + \frac{c \times K}{p}$

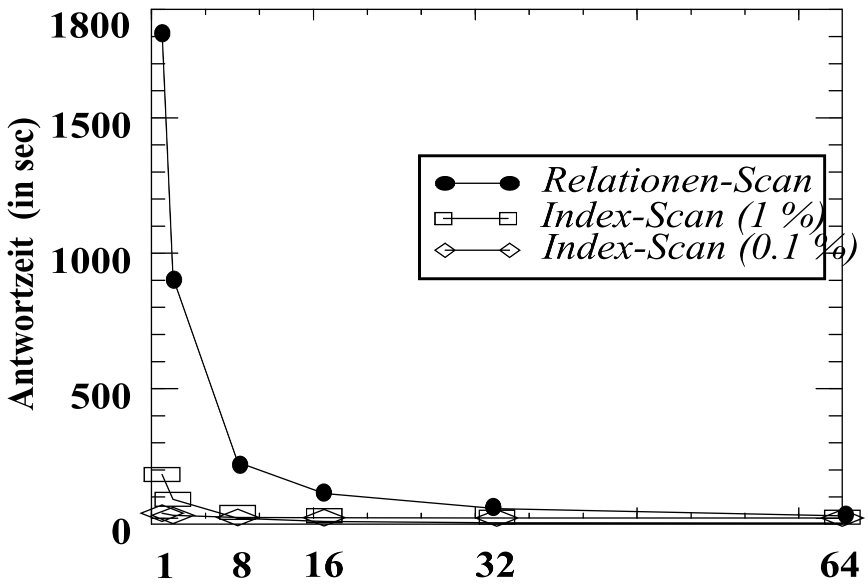


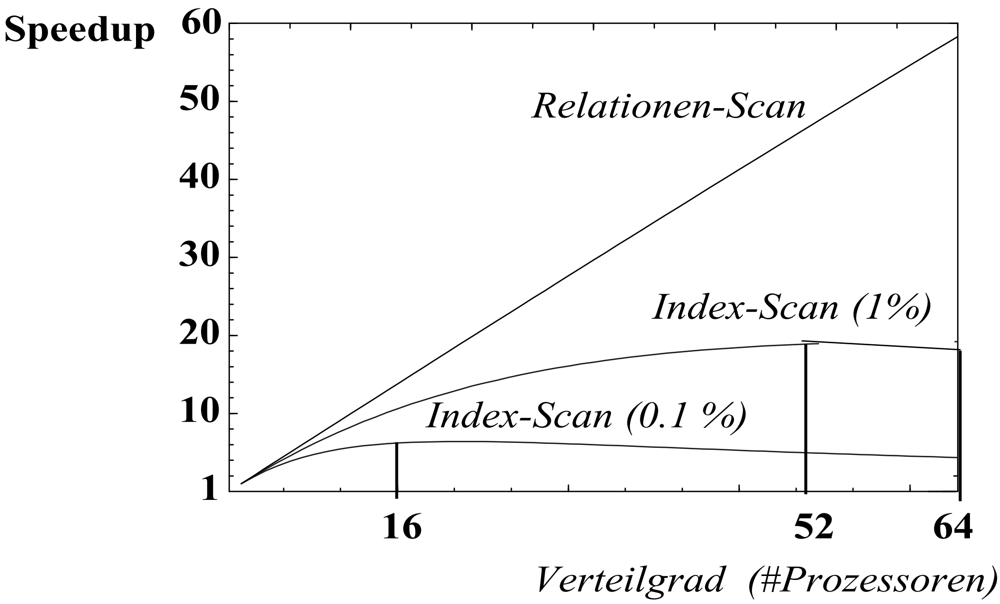
- **optimaler Parallelitäts-/Verteilgrad** $p_{opt} = \sqrt{\frac{c \times K}{b}}$

Bestimmung des Verteilgrades (2)

■ **Bestimmung des Verteilgrades D**

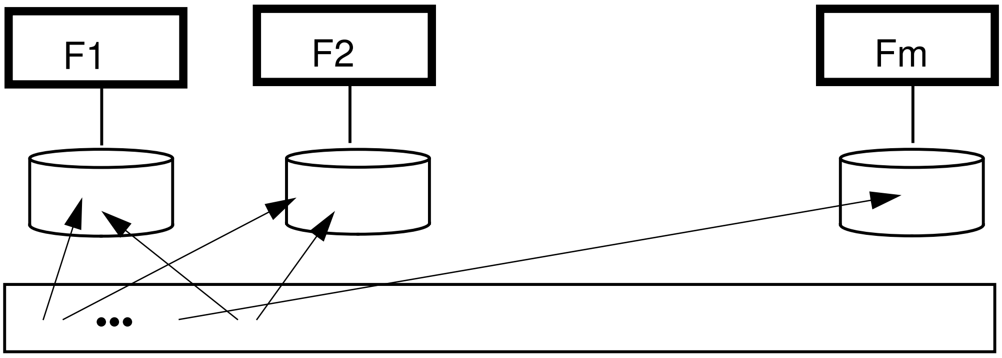
- für jeden Anfragetyp p_{opt} bestimmen
- Verteilgrad D ergibt sich aus gewichtetem Mittel (➥ Kompromißlösung für erwartetes Lastprofil)





- Relationen-Scan:
- Index-Scan (1%):
- Index-Scan (0.1%):

Fragmentierung: Round-Robin



Satz i -> Rechner (i mod m) + 1

- **Vorteile:**
  - Einfachheit
  - gleichmäßige Fragmentgrößen
  - günstige Lastbalancierung (geringer "Skew" für Relationen-Scans)
  - gleiche Tupelanzahl pro Rechner garantiert jedoch nicht immer gleichmäßige Zugriffshäufigkeit

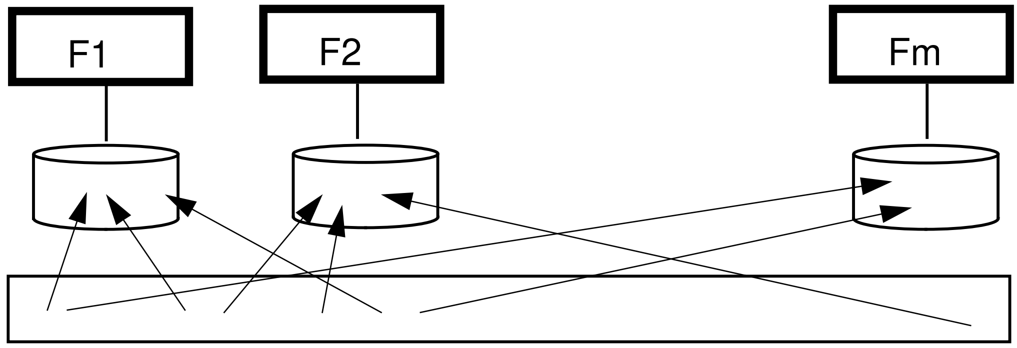
- **Nachteil:**

  Verteilung von Attributwerten unbekannt

➥ ***Sämtliche Anfragen müssen an allen Rechnern bearbeitet werden***

Das ist besonders ineffizient für Einzelsatzzugriffe sowie sonstige exakte Anfragen

Hash-Fragmentierung



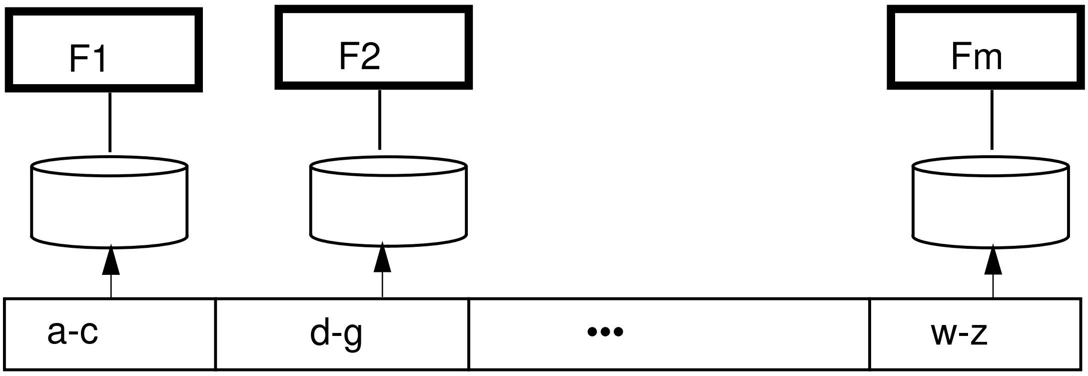
Satz i -> Rechner h (VA_i)

- **Verteilung der Tupel über Hash-Funktion h**
  auf Fragmentierungs- bzw. Verteilattribut VA
  (z. B. Primärschlüssel)

- **Vorteile:**
  - Einfachheit
  - Exakte Anfragen auf Verteilattribut sind auf ein Fragment (Rechner) eingrenzbar
  - Equi-Joins über das Verteilattribut werden unterstützt

- **Nachteile:**
  - keine Unterstützung für Bereichsanfragen (range queries)
  - Gefahr ungleichmäßiger Partitionsgrößen (-> Skew)
    bei "schlechter" Hash-Funktion und ungünstiger Werteverteilung
    (v. a. bei nicht-eindeutigen Verteilattributen)

Bereichsfragmentierung



- **Festlegung der Fragmentierung durch Wertebereiche auf Verteilattribut**
  - Spezifizierung durch Prädikate
  - Fragmentierungsansatz verteilter DBS
- **Vorteile:**
  - Exakte Anfragen sowie Bereichsanfragen auf Verteilattribut auf relevante Fragmente (Rechner) eingrenzbar
  - Unterstützung für Equi-Joins über das Verteilattribut
  - stabiler als Hash-Fragmentierung gegenüber ungleichmäßiger Werteverteilung
- **Nachteile:**
  - erhöhte Gefahr ungünstiger Lastbalancierung (zu geringer Parallelisierung)
  - Bestimmung der einzelnen Wertebereiche relativ aufwendig

Verfeinerte Bereichsfragmentierung

■ **Parallelisierung wird bei Bereichsanfragen durch die Bereichsfragmentierung eingeschränkt**

Einfache Bereichsfragmentierung beschränkt Anfragen auf Verteilattribut auf minimale Anzahl von Rechnern, falls m=D (ein Fragment pro Rechner)

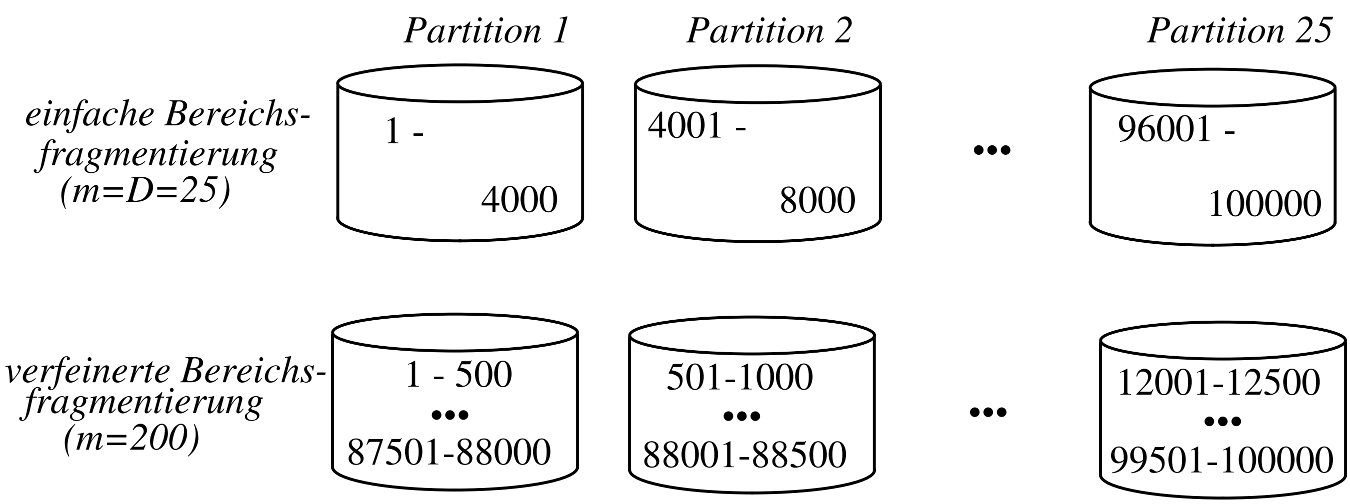
■ **Ziel:**
Verbesserung der Lastbalancierung durch größere Anzahl von Fragmenten (m > D)

■ **Ansatz:**
Erhöhe Fragmentanzahl, so daß relevante Fragmente im Mittel p_{opt} Rechnern zugeordnet werden können:

$m = p_{opt} / s$ (s = mittlere Selektivität der Bereichsanfragen, $0 \leq s \leq 1$)

■ **Beispiel:**

- CARD (KONTO) = 100.000
- Verteilattribut KTONR
- s=0.05, p_{opt} = 10, D=25



Mehrdimensionale Bereichsfragmentierung

■ **Ziel:** Unterstützung von **exakten Anfragen und Bereichsanfragen** auf mehreren Attributen

■ **Beispiel:**

- Zwei Anfragetypen:

```
SELECT * FROM PERS           und        SELECT * FROM PERS
WHERE NAME = :Z                         WHERE GEHALT BETWEEN [:X,:Y]
```

- Hash- und Bereichsfragmentierung können nur **einen** Anfragetyp unterstützen (für anderen sind alle Rechner involviert)
- Mehrdimensionale Bereichsfragmentierung erlaubt, beide Anfragetypen auf Teilmenge der Fragmente zu beschränken

		Gehalt					
		< 20K	< 50K	< 70K	< 90K	< 120K	≥ 120 K
	A-D	1	1	4	4	7	7
	E-H	1	1	4	4	7	7
Name	I-L	2	2	5	5	8	8
	M-P	2	2	5	5	8	8
	Q-S	3	3	6	6	9	9
	T-Z	3	3	6	6	9	9

9 Rechner,
36 Fragmente

■ **nur empfehlenswert,**
wenn auf mehrere Attribute etwa gleiche Zugriffshäufigkeit (ansonsten eindimensionale Fragmentierung ggf. besser)

Allokation von Fragmenten

- **Allokation: Partitionenbildung durch Rechnerzuordnung der Fragmente (SN)**
  - Festlegung des Verteilgrades D
  - "gleichmäßige" Aufteilung der m Fragmente unter D Rechnern: (statische) Lastbalancierung
  - analoge Partitionierung von Indexstrukturen

- **Balancierung der Zugriffshäufigkeit von Partitionen**
  - Round Robin-Zuordnung der Fragmente
  - (gierige Heuristik) bei stark unterschiedlichen Fragment-Zugriffshäufigkeiten:
    - ordne Fragmente gemäß Zugriffshäufigkeiten
    - solange noch Fragmente aufzuteilen, wähle das nächste Fragment mit der höchsten Zugriffsfrequenz aus
    - ordne es dem bis dahin am geringsten ausgelasteten Knoten (Partition) zu

Beispiel:

Fragment	F1	F2	F3	F4	F5	F6	F7	F8
Zugriffshäufigkeit	100	600	400	150	200	500	400	50

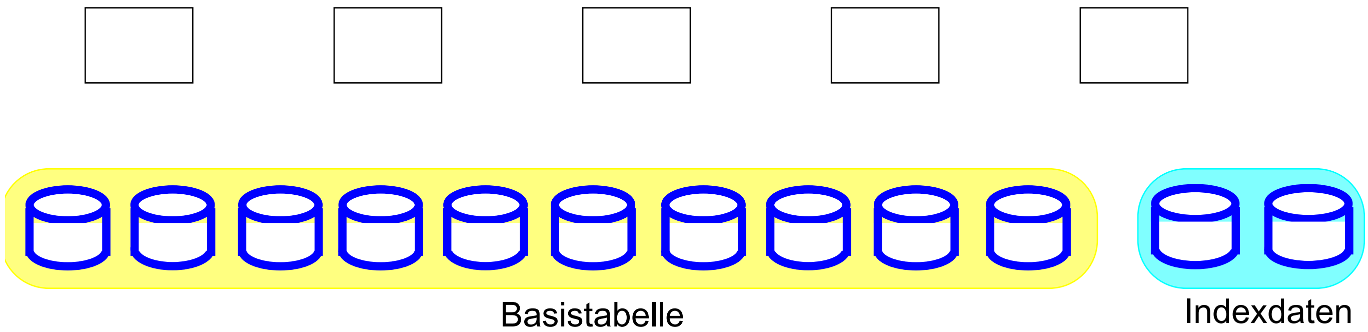
D=4: Round Robin:

Greedy:

Datenverteilung bei SD und SE

■ **wesentliche Unterschiede gegenüber SN**

- Festlegung der Datenverteilung bezieht sich nur auf Platten, nicht auf Prozessoren
- Datenallokation hat keinen Einfluß auf Kommunikationshäufigkeit
- größere Freiheitsgrade für Verarbeitungsparallelität
- Indexallokation kann unabhängig von Tabellenallokation gewählt werden
- physische und logische Fragmentierung möglich
- Transfer von Zwischenergebnisse über Platte kann Kommunikationskosten einsparen



■ **Datenverteilung sollte**
E/A-Parallelität sowie flexible Verarbeitungsparallelität ermöglichen und Plattenengpässe möglichst umgehen

■ **großes Lastbalancierungspotential**
(Parallelitätsgrad, Ausführungsort)

SE/SD-Datenallokation (2)

■ **Ansatz zur Bestimmung des Verteilgrades**

- breites Declustering zur optimalen Abdeckung von Relationen-Scans
- selektivere Anfragen bzw. Anfragen im Mehrbenutzerbetrieb können dennoch mit geringerer Parallelität bearbeitet werden

■ **Physisches Declustering, z.B. auf Blockebene**

- transparent für DBS realisierbar, z.B. durch Disk-Array
- zur Nutzung sequentieller E/A sollte Fragmentierungsgranulat mehrere Blöcke umfassen
- Plattenbehinderungen selbst innerhalb einer Anfrage möglich

■ **Logische Fragmentierung** (analog SN, z.B. über Hash- oder Bereichspartitionierung)

- DBMS kann Datenallokation gezielt nutzen, u.a. zur Bestimmung des Parallelitätsgrades
- reduzierter Datenraum für Anfragen auf Verteilattribut

  - Beispiel:

    Bereichspartitionierung von Relation R auf Attribut A (D_R=20)

    A (1 - 10.000; 10.001 - 20.000; 20.001 - 30.000; ... 190.001 - 200.000)

  - Anfrage *Select MAX (B) FROM R*
    WHERE A > 70.000 AND A <= 110.000

  - Parallele Ausführung mit 4 (2, 1) Teilanfragen ohne Plattenengpässe

Zusammenfassung

- **Datenpartitionierung:**
  Fragmentierung + Allokation der Fragmente

- **Hauptziele der Fragmentierung**
  - Reduzierung des Verarbeitungsumfangs
  - Reduzierung des Kommunikationsaufwandes / Unterstützung von Lokalität (v.a. wichtig in VDBS)
  - Unterstützung von Parallelität (v.a. wichtig für PDBS)

- Parallele DBS basieren auf horizontaler Fragmentierung

- hohe Flexibilität durch Bereichsfragmentierung und Varianten

- mehrdimensionale Fragmentierung: Eingrenzung des Verarbeitungsumfangs hinsichtlich mehrerer Attribute

- **Datenallokation: Zuordnung der Fragmente zu Knoten**
  - Bestimmung von Verteilgrad und Auswahl der Knoten
  - wesentlich für Lastbalancierung
  - ggf. replizierte Speicherung von Fragmenten

- **SE/SD:**
  Datenallokation bezüglich Externspeicher (Platten)
  mit erhöhten Freiheitsgraden