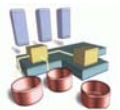
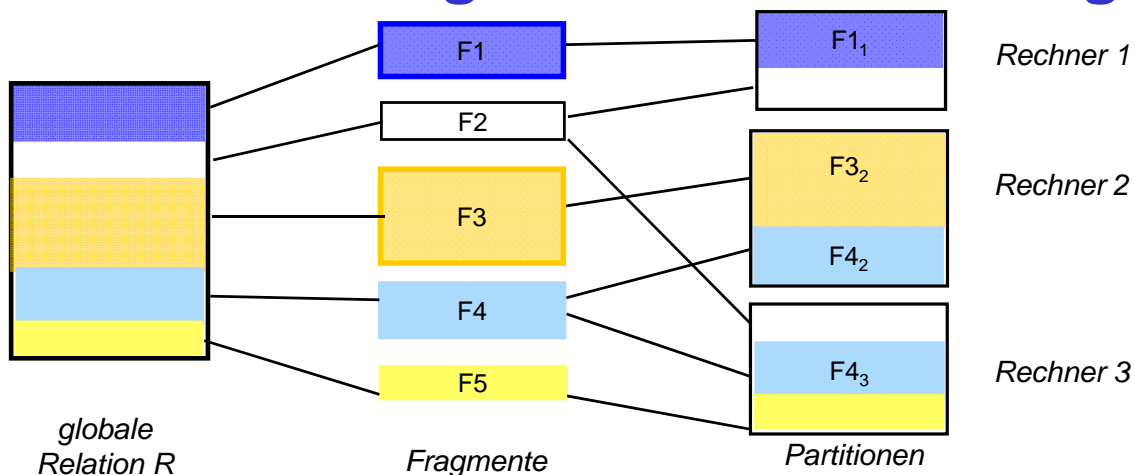


# 4. Datenallokation in VDBS und PDBS

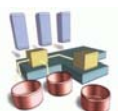
- Fragmentierung und Allokation
- Fragmentierungsvarianten
  - horizontale Fragmentierungen
  - vertikale Fragmentierung
  - hybride Fragmentierung
- Verteilungstransparenz
- Datenallokation für Parallele DBS
  - Verteilgrad
  - Varianten der horizontalen Fragmentierung
    - Round Robin
    - Hash-Fragmentierung
    - Bereichsfragmentierung: einfach, verfeinert, mehrdimensional
  - Allokation der Fragmente
  - Datenallokation für Shared Disk / Shared Everything
- Datenallokation in kommerziellen DBS (Oracle, DB2)



## Bestimmung der Datenverteilung

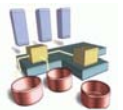


- Fragmentierung
  - Fragmente: Einheiten der Datenverteilung
  - wünschenswert: (horizontale / vertikale) Teile von Relationen
- Allokation von Fragmenten
  - bestimmt weitgehend Ausführungsort von DB-Operationen
  - Zielkonflikt: Minimierung der Kommunikationskosten vs. Lastbalancierung
  - ggf. replizierte Speicherung von Fragmenten



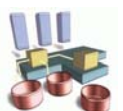
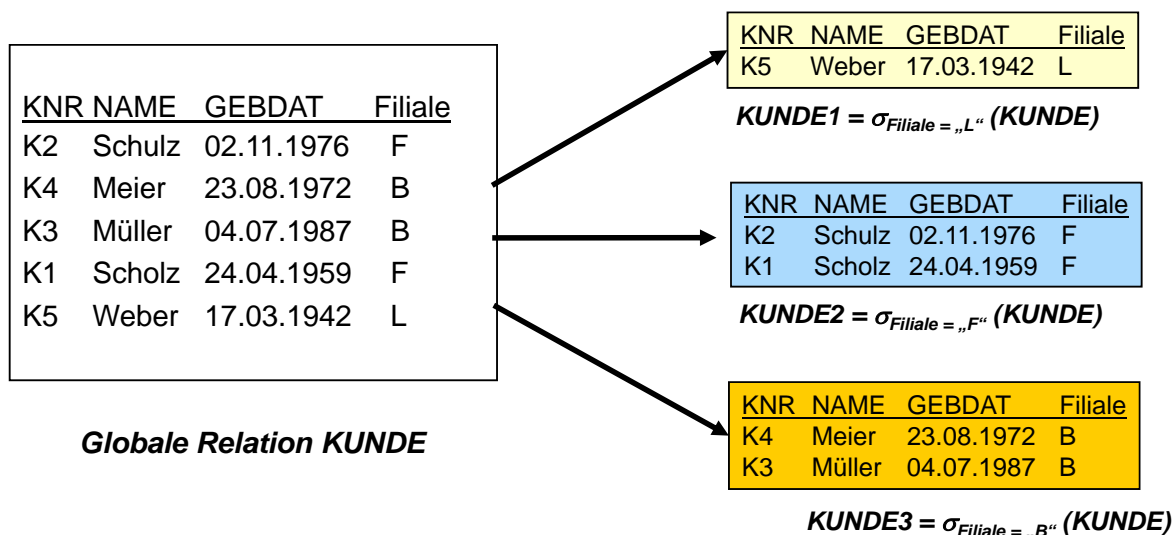
# Fragmentierung

- Gründe für (horizontale bzw. vertikale) Fragmentierung
  - Lastbalancierung
  - Nutzung von Lokalität
  - Reduzierung des Verarbeitungsumfangs für Anfragen
  - Unterstützung von Parallelverarbeitung
  - Bessere Administrierbarkeit sehr großer Tabellen (Sicherung, Reorganisation)
- Anforderungen
  - *Vollständigkeit*: jedes Datenelement muss in wenigstens einem Fragment enthalten sein
  - *Rekonstruierbarkeit*: Verlustfreiheit der Zerlegung
  - (weitestgehende) *Disjunktheit*



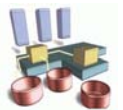
## Horizontale Fragmentierung

- Zeilenweise Aufteilung von Relationen
- Definition der Fragmentierung durch Selektionsprädikate  $P_i$  auf der Relation:  $R_i := \sigma_{P_i}(R)$  ( $1 \leq i \leq n$ )



## Horizontale Fragmentierung (2)

- Fragmentierungsprädikate sind so zu wählen, damit gilt
  - Vollständigkeit: jedes Tupel ist einem Fragment eindeutig zugeordnet
  - Fragmente sind disjunkt:  $R_i \cap R_j = \{\}$  ( $i \neq j$ )
  - Verlustfreiheit (Relation ist Vereinigung aller Fragmente):  
 $R = \cup R_i$  ( $1 \leq i \leq n$ )
- Vorteile
  - Anfragen können ggf. auf Teilmenge der Fragmente begrenzt werden
  - optimale Parallelisierbarkeit von Anfragen auf großen Tabellen



## Abgeleitete horizontale Fragmentierung

- Horizontale Fragmentierung einer Tabelle S wird über Prädikate einer anderen Relation R definiert, zu der S abhängig ist
- i.a. funktionale Abhängigkeit (n:1) über Fremdschlüssel-Primärschlüssel-Beziehungen
- Beispiel: Fragmentierung von Relation KONTO analog zu KUNDE

### Globale Relation KONTO

KUNDE-Partitionierung (über FILIALE)  
bestimmt KONTO-Partitionierung

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

KTONR	KNR	KTOSTAND
7654	K5	63,79

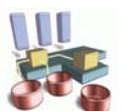
KONTO1: Konten zu KUNDE1 (Filiale „L“)

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

KONTO2: Konten zu KUNDE2 (Filiale „F“)

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

KONTO3: Konten zu KUNDE3 (Filiale „B“)



## Abgeleitete horizontale Fragmentierung (2)

- Semi-Join zwischen Relationen S und R

$$S \bowtie R = \pi_{S\text{-Attribute}} (S \bowtie R)$$

– asymmetrischer Operator

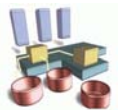
- Seien  $R_1, R_2, \dots, R_n$  Fragmente einer horizontalen Fragmentierung von R. Eine von R abgeleitete horizontale Fragmentierung einer Tabelle S hat die Fragmente

$$S_i = S \bowtie R_i = S \bowtie \sigma_{P_i} (R) \quad (1 \leq i \leq n)$$

- effiziente Join-Verarbeitung zwischen R und S

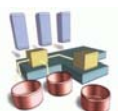
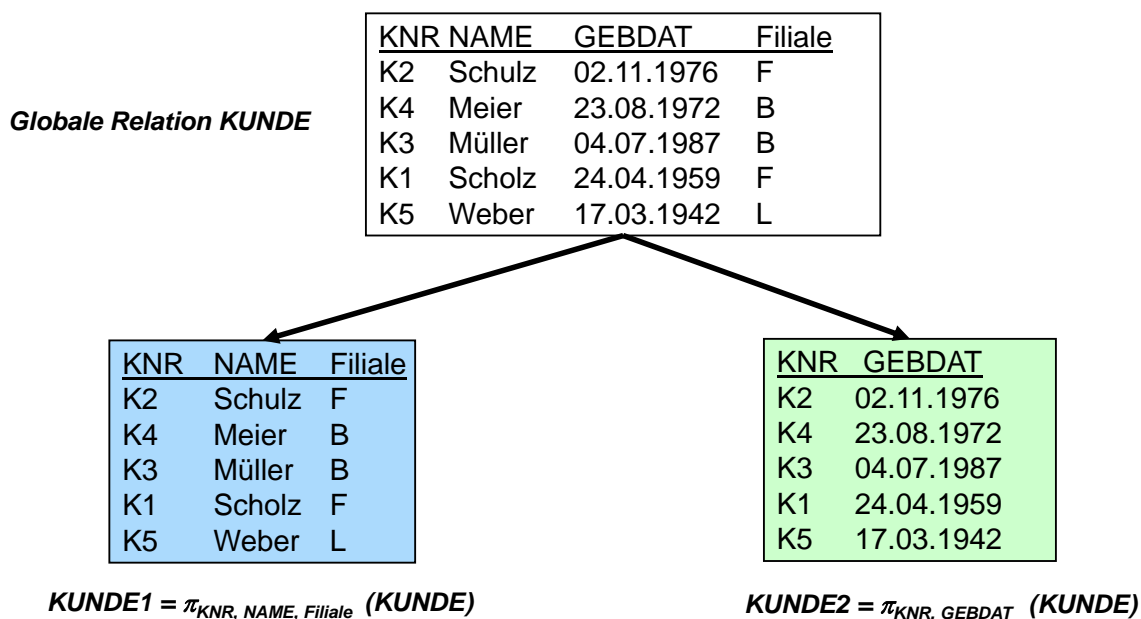
– lokale Join-Durchführung falls Fragmente  $R_i$  und  $S_i$  am selben Rechner  
 – reduzierte Datenmengen für Suche nach Verbundpartnern

- Vollständigkeit?



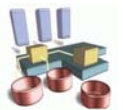
## Vertikale Fragmentierung

- Spaltenweise Aufteilung von Relationen
- Definition der Fragmentierung durch Projektion



## Vertikale Fragmentierung (2)

- Vollständigkeit:
  - jedes Attribut in wenigstens 1 Fragment enthalten
- Verlustfreie Zerlegung:
  - Primärschlüssel i.a. in jedem Fragment enthalten
  - JOIN-Operation zur Rekonstruktion des gesamten Tupels
- Vorteile?

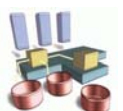
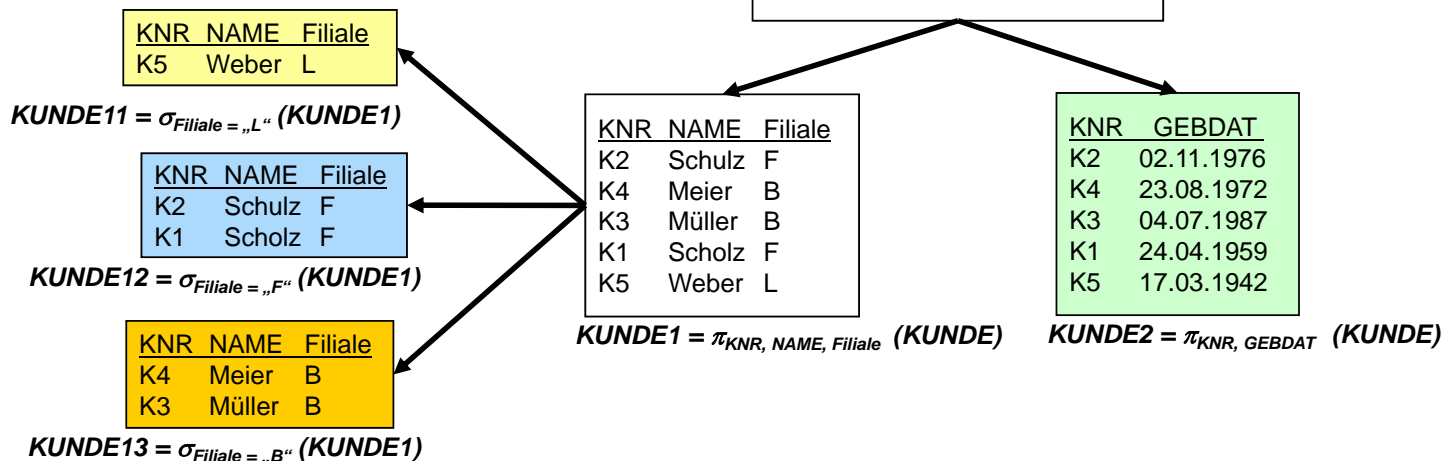


## Hybride Fragmentierung

- Kombination von horizontaler und vertikaler Fragmentierung

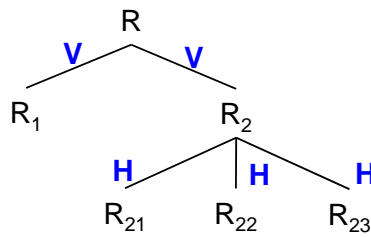
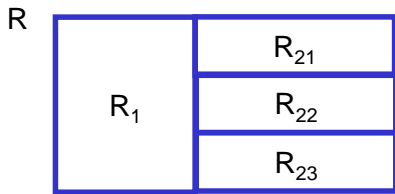
Globale Relation KUNDE

KNR	NAME	GEBDAT	Filiale
K2	Schulz	02.11.1976	F
K4	Meier	23.08.1972	B
K3	Müller	04.07.1987	B
K1	Scholz	24.04.1959	F
K5	Weber	17.03.1942	L

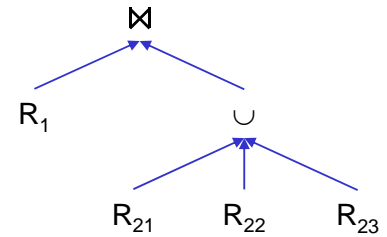


# Hybride Fragmentierung (2)

a) vertikale gefolgt von horizontaler Partitionierung

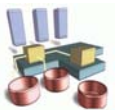
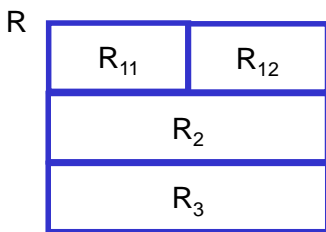


Fragmentierungsbaum



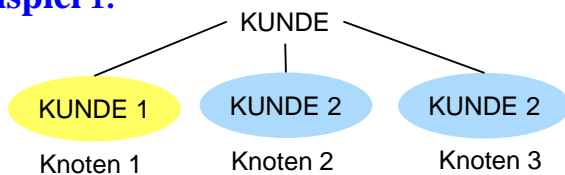
Operatorbaum (Rekonstruktion)

b) horizontale gefolgt von vertikaler Partitionierung



# Verteilungstransparenz

Beispiel 1:



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

## Keine Transparenz:

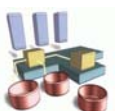
```
SELECT     NAME INTO $NAME FROM      Knoten1@KUNDE1
           WHERE KNR = $KNO
IF NOT FOUND THEN
  SELECT   NAME INTO $NAME FROM      Knoten3@KUNDE2
           WHERE KNR = $KNO
```

## Orts- und Replikationstransparenz:

– Weglassen der Ortsbezeichnungen (Knoten@)

## Orts-, und Replikations- und Fragmentierungstransparenz:

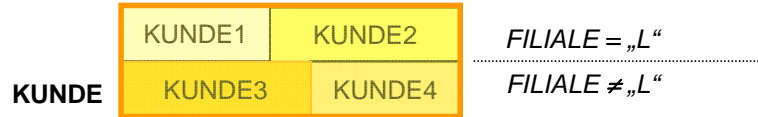
```
SELECT     NAME INTO $NAME FROM      KUNDE
           WHERE KNR = $KNO
```



# Verteilungstransparenz (2)

## Beispiel 2:

### Hybride Fragmentierung



KUNDE1 (KNR, NAME)

KUNDE3 (KNR, NAME, FILIALE)

KUNDE2 (KNR, GEBDAT, FILIALE)

KUNDE4 (KNR, GEBDAT)

KUNDE<sub>i</sub> sei an Knoten<sub>i</sub> gespeichert;

KUNDE1 zusätzlich an Knoten 5 und KUNDE4 an Knoten 6

### ■ Orts-, Replikations- und Fragmentierungstransparenz

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

### ■ Orts- und Replikations-, keine Fragmentierungstransparenz

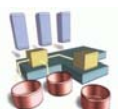
```
SELECT NAME INTO $Name FROM KUNDE3 WHERE KNR = "K3";  
SELECT GEBDAT INTO $Geb FROM KUNDE4 WHERE KNR = "K3";  
INSERT INTO KUNDE1 (KNR, NAME) VALUES ("K3", $Name)  
INSERT INTO KUNDE2 (KNR, GEBDAT, FILIALE) VALUES ("K3", $Geb, "L")  
DELETE KUNDE3 WHERE KNR = "K3";  
DELETE KUNDE4 WHERE KNR = "K3";
```



# Verteilungstransparenz (3)

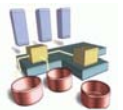
## ■ Beispiel 2: Keine Transparenz

```
SELECT NAME INTO $Name FROM Knoten3@KUNDE3 WHERE KNR = "K3";  
SELECT GEBDAT INTO $Geb FROM Knoten4@KUNDE4 WHERE KNR = "K3";  
INSERT INTO Knoten1@KUNDE1 (KNR, NAME) VALUES ("K3", $Name)  
INSERT INTO Knoten5@KUNDE1 (KNR, NAME) VALUES ("K3", $Name)  
INSERT INTO Knoten2@KUNDE2 (KNR, GEBDAT, FILIALE)  
VALUES ("K3", $Geb, "L")  
DELETE Knoten3@KUNDE3 WHERE KNR = "K3";  
DELETE Knoten4@KUNDE4 WHERE KNR = "K3";  
DELETE Knoten6@KUNDE4 WHERE KNR = "K3";
```



# Datenallokation in PDBS

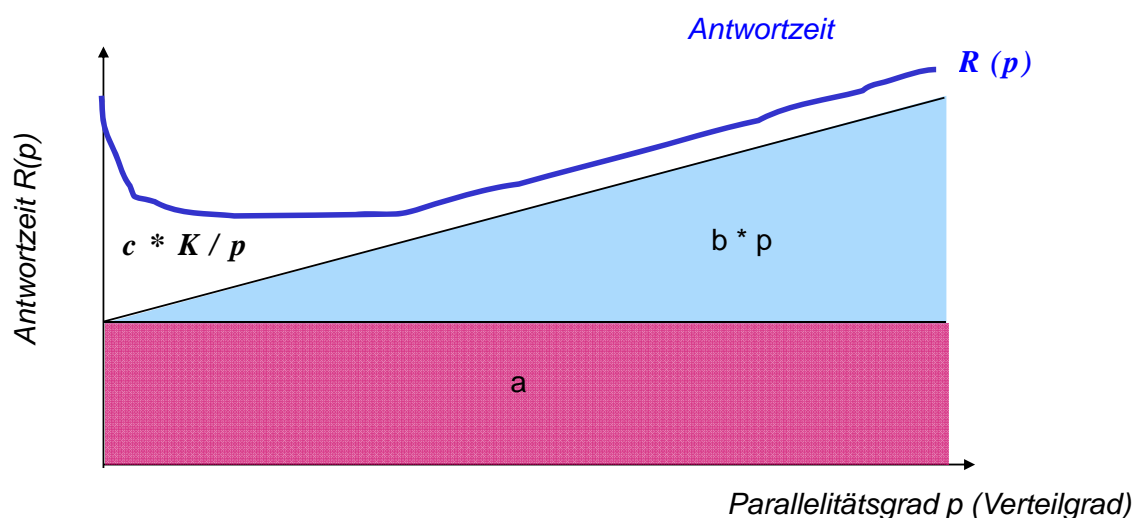
- zunächst: Shared Nothing
- Horizontale Verteilung von Relationen über mehrere Rechner
  - Nutzung von Datenparallelität
  - Lastbalancierung
- Teilaufgaben zur Bestimmung der Datenverteilung
  - Festlegung des Verteilgrades einer Relation
  - Fragmentierung
  - Allokation
- Bestimmung des "optimalen" Verteilgrades schwierig
  - wachsende Rechneranzahl erhöht Kommunikations-Overhead und reduziert Parallelisierungsgewinn, v.a. für kleinere Relationen
  - "Full Declustering" oft nicht sinnvoll
  - einfache Anfragen (kleine Relationen) -> wenige Partitionen
  - datenintensive Anfragen (große Relationen) -> viele Partitionen



## Bestimmung des Verteilgrades einer Relation

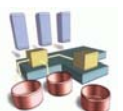
- Einfaches Antwortzeitmodell ( $K$  = Kardinalität der Relation):

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



- Optimaler Parallelitäts-/Verteilgrad

$$p_{\text{opt}} = \sqrt{\frac{c \times K}{b}}$$

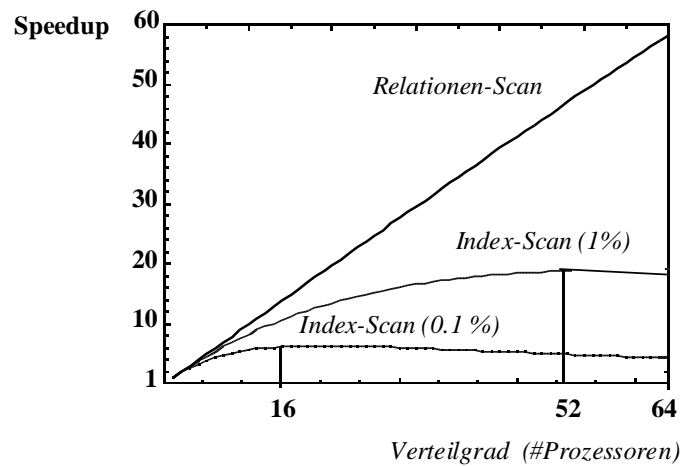
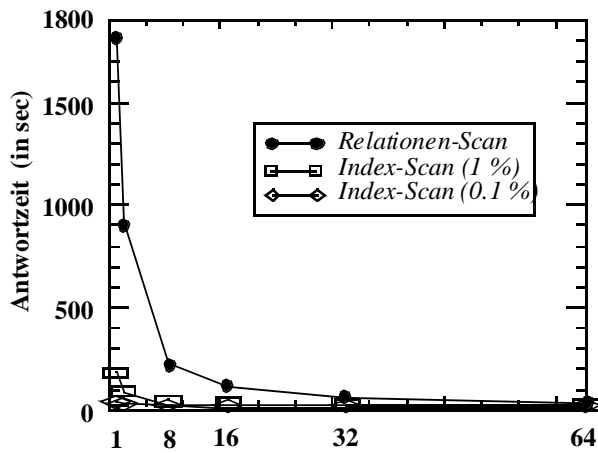




# Bestimmung des Verteilgrades

## ■ Bestimmung des Verteilgrades D

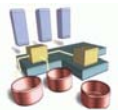
- für jeden Anfragetyp  $p_{opt}$  bestimmen
- Verteilgrad D ergibt sich aus gewichtetem Mittel ( $\Rightarrow$  Kompromisslösung für erwartetes Lastprofil)



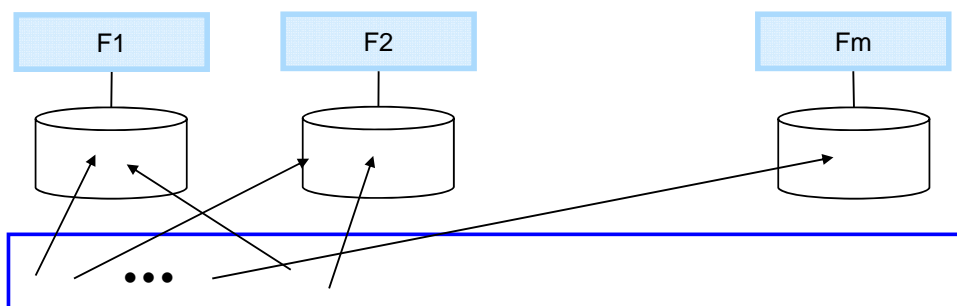
Relationen-Scan:

Index-Scan (1 %):

Index-Scan (0.1 %):



# Fragmentierung: Round-Robin



Satz  $i \rightarrow$  Fragment  $(i \bmod m) + 1$

## ■ Vorteile

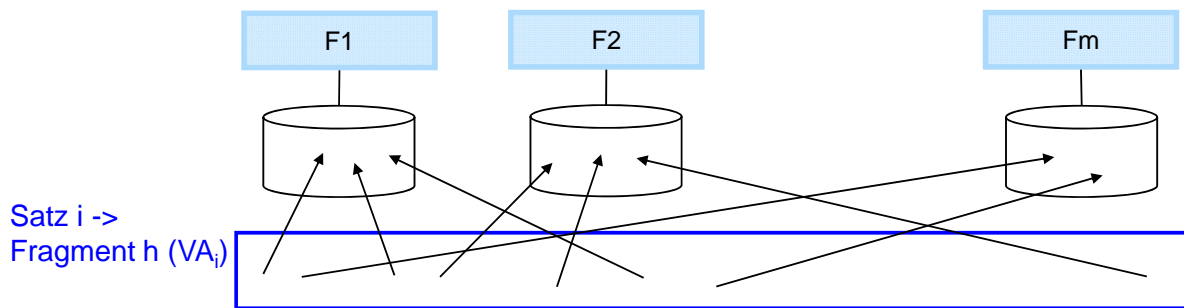
- gleichmäßige Fragmentgrößen
- günstige Lastbalancierung bei gleichmäßiger Zugriffsverteilung (z.B. geringer "Skew" für Relationen-Scans)

## ■ Nachteil: Verteilung von Attributwerten unbekannt

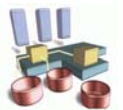
- sämtliche Anfragen an allen Rechnern zu bearbeiten
- besonders ineffizient für Exact-Match-Anfragen und Einzelsatzzugriffe



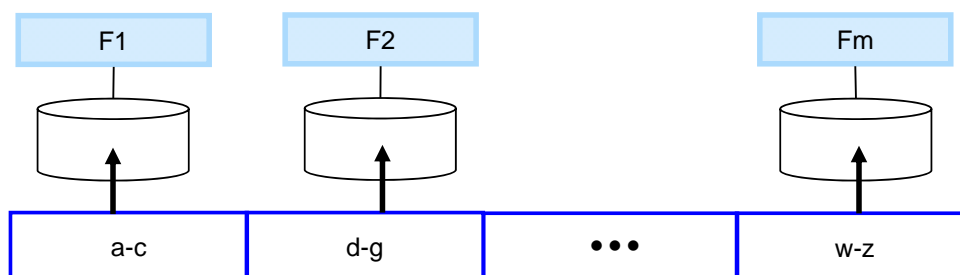
# Hash-Fragmentierung



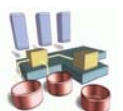
- Hash-Funktion  $h$  auf Fragmentierungs- bzw. Verteilattribut (z. B. Primärschlüssel)
- Vorteile:
  - Einfachheit
  - Exact-Match-Queries auf Verteilattribut auf 1 Fragment eingrenzbar
  - Equi-Joins über das Verteilattribut werden unterstützt
- Nachteile:
  - keine Unterstützung für Bereichsanfragen (range queries)
  - Gefahr ungleichmäßiger Partitionsgrößen ( $\rightarrow$  Skew)



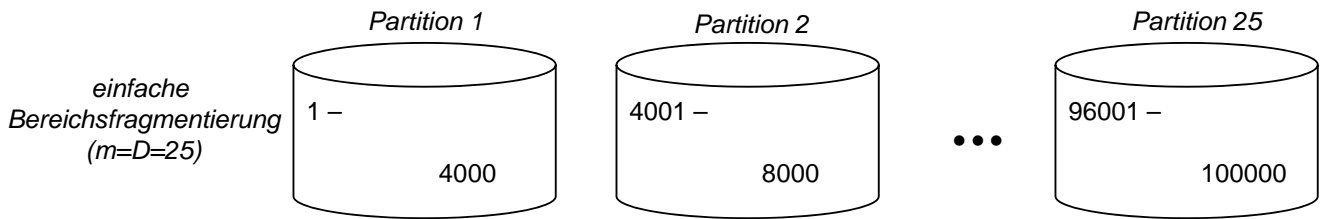
# Bereichsfragmentierung



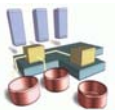
- Fragmentierung über Wertebereiche auf Verteilattribut (VA)
  - Spezifizierung durch Bereichsprädikate
  - Fragmentierungsansatz verteilter DBS
- Vorteile:
  - Exact-Match- sowie Range-Queries auf VA auf relevante Fragmente eingrenzbar
  - Unterstützung für Equi-Joins über das Verteilattribut
  - stabiler als Hash-Fragmentierung gegenüber ungleichmäßiger Werteverteilung
- Nachteile:
  - Gefahr eingeschränkter Parallelisierung
  - Bestimmung der Wertebereiche



# Bereichsfragmentierung: Beispiel

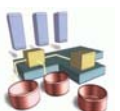
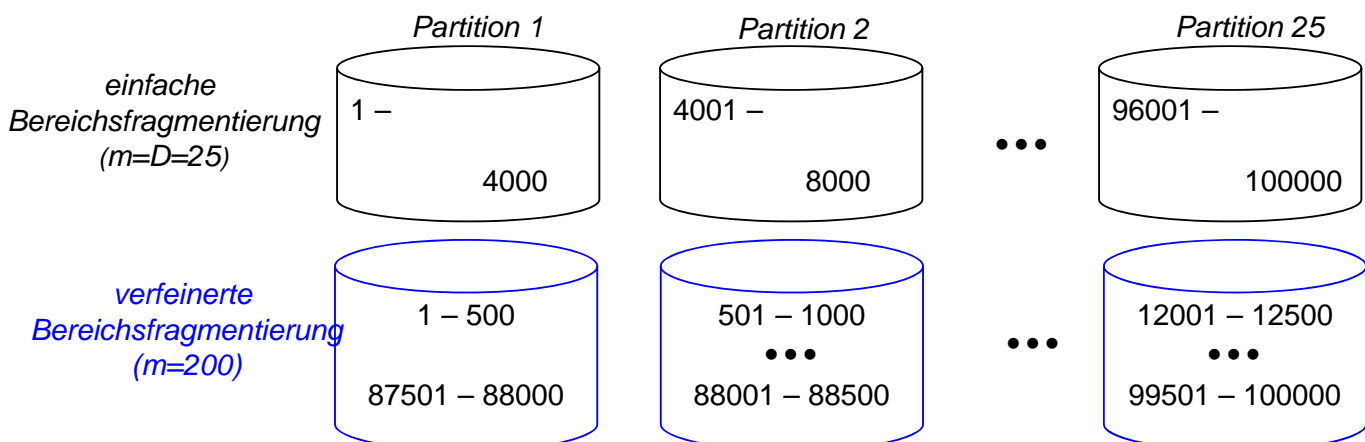


- Beispiel: Kontoaufteilung über VA Kontonummer
  - Exact-Match-Anfragen gehen auf 1 Rechner
  - Bereichsanfragen auf VA werden auf minimale Rechnerzahl begrenzt
  - Relationen-Scan gehen auf alle Rechner
- Aber: 1 Fragment pro Rechner ( $m=D$ ) beschränkt auch Parallelisierung / Lastbalancierung für Bereichsanfragen
- Verbesserung durch größere Anzahl von Fragmenten ( $m > D$ )
  - > verfeinerte Bereichsfragmentierung



## Verfeinerte Bereichsfragmentierung

- Ansatz: erhöhe Fragmentanzahl, so dass relevanten Fragmente im Mittel  $p_{opt}$  Rechnern zugeordnet werden können
  - $K$  = Kardinalität der Tabelle
  - $s$  = mittlere Selektivität der Bereichsanfragen,  $0 \leq s \leq 1$
  - $p_{opt}$  : optimaler Parallelitätsgrad für durchschnittliche Bereichsanfrage
  - Wähle:  $m = p_{opt} / s$  (Fragmentgröße:  $K/m = K * s / p_{opt}$ )
- Beispiel: CARD (KONTO) = 100.000, Verteilattribut KTONR,  $s=0.05$ ,  $p_{opt} = 10$ ,  $D=25$



# Mehrdimensionale Bereichsfragmentierung

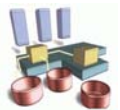
- Ziel: Unterstützung von Exact-Match- und Bereichsanfragen auf mehreren Attributen
- Beispiel: Zwei Anfragetypen

**A:** SELECT \* FROM PERS WHERE **NAME** = :Z

**B:** SELECT \* FROM PERS WHERE **GEHALT** BETWEEN [:X,:Y]

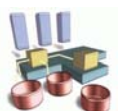
- Hash- und Bereichsfragmentierung können nur 1 Anfragetyp unterstützen (für andere sind alle Rechner involviert)
- Mehrdimensionale Bereichsfragmentierung erlaubt, beide Anfragetypen auf Teilmenge der Fragmente zu beschränken

		< 20K	< 50K	< 70K	< 90 K	< 120K	≥ 120K	<i>Gehalt</i>
9 Rechner 36 Fragmente	<i>Name</i> A-D	1	1	4	4	7	7	
	E-H	1	1	4	4	7	7	
	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	



## 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



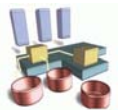
# Allokation: Beispiel

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

D=4:

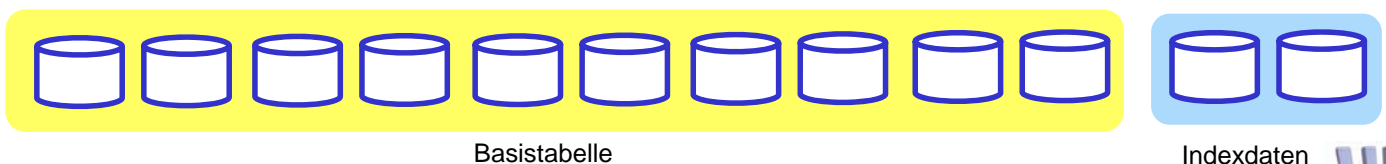
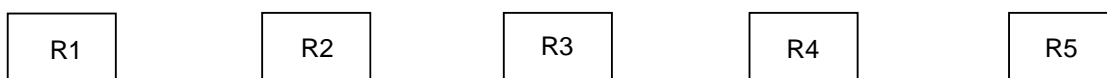
Round Robin:

Greedy:



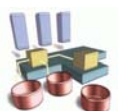
## Datenverteilung bei SD und SE

- 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
- großes Lastbalancierungspotential (Parallelitätsgrad, Ausführungsort)



Basistabelle

Indexdaten



# 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

## ■ Fragmentierung analog SN möglich, z. B. über Hash- oder Bereichspartitionierung

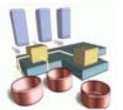
- reduzierter Datenraum für Anfragen auf Verteilattribut

- Beispiel: Bereichspartitionierung von Relation R auf Attribut A ( $D_R=20$ )

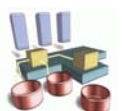
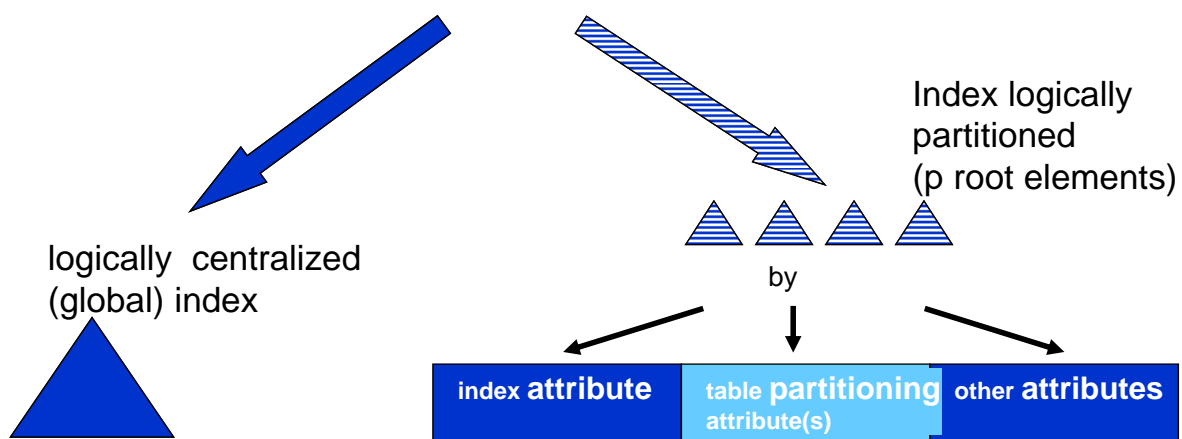
A: (1 - 10.000; 10.001 - 20.000; 20.001 - 30.000; ... 190.001 - 200.000)

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

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

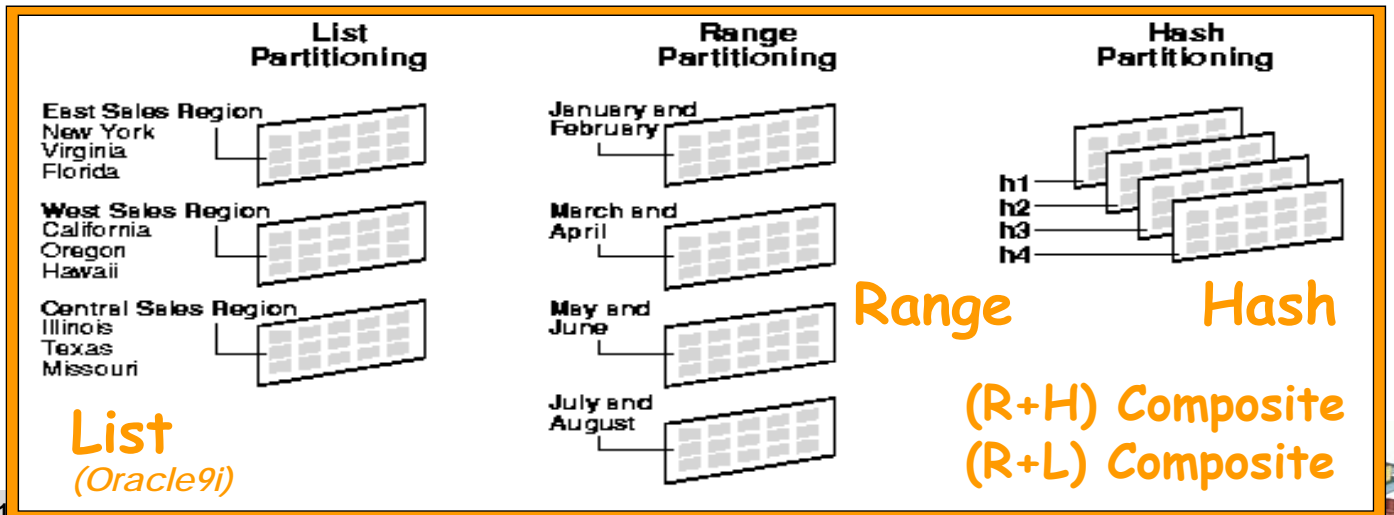


# Index-Partitionierung in PDBS



# Partitionierung in Oracle (SE, SD)

- **Range** - partition by predefined ranges of continuous values
- **Hash** - partition according to hashing algorithm applied by Oracle
- **Composite** - e.g. range-partition by key1, hash sub-partition by key2
- **List** - partition by lists of predefined discrete values

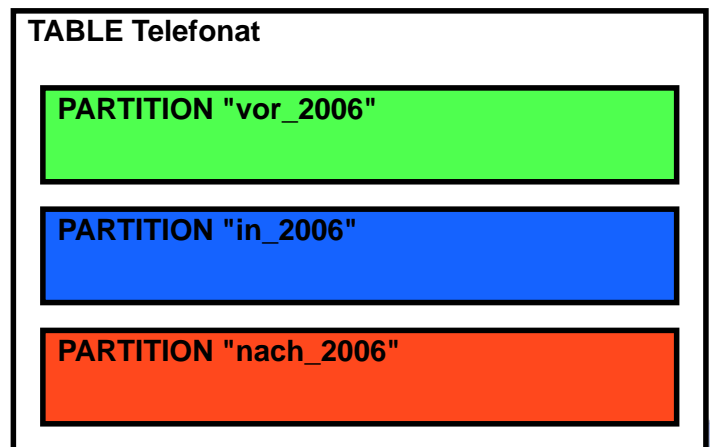


WS11/12

## Beispiel: Oracle Range-Partitionierung

```
CREATE TABLE Telefonat (Datum DATE, KdNr NUMBER, Dauer NUMBER)
PARTITION BY RANGE(Datum)
(PARTITION vor_2006 VALUES
LESS THAN TO_DATE('01-JAN-2006', 'DD-MON-YYYY'),
PARTITION in_2006 VALUES
LESS THAN TO_DATE('01-JAN-2007', 'DD-MON-YYYY'),
PARTITION nach_2006 VALUES LESS THAN MAXVALUE) ;
```

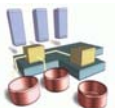
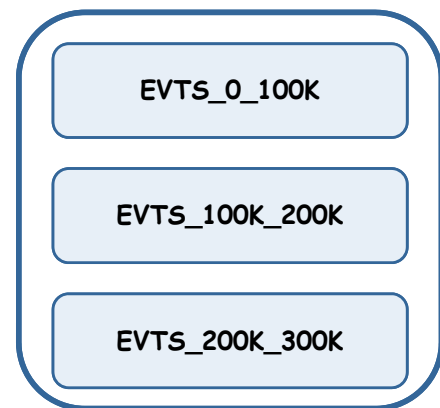
Datum	KdNr	Dauer
2005-05-15	1	45
2005-07-21	2	215
2006-01-16	1	200
2006-10-16	3	20
2007-03-07	2	50



# Beispiel: Oracle Range-Partitionierung

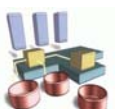
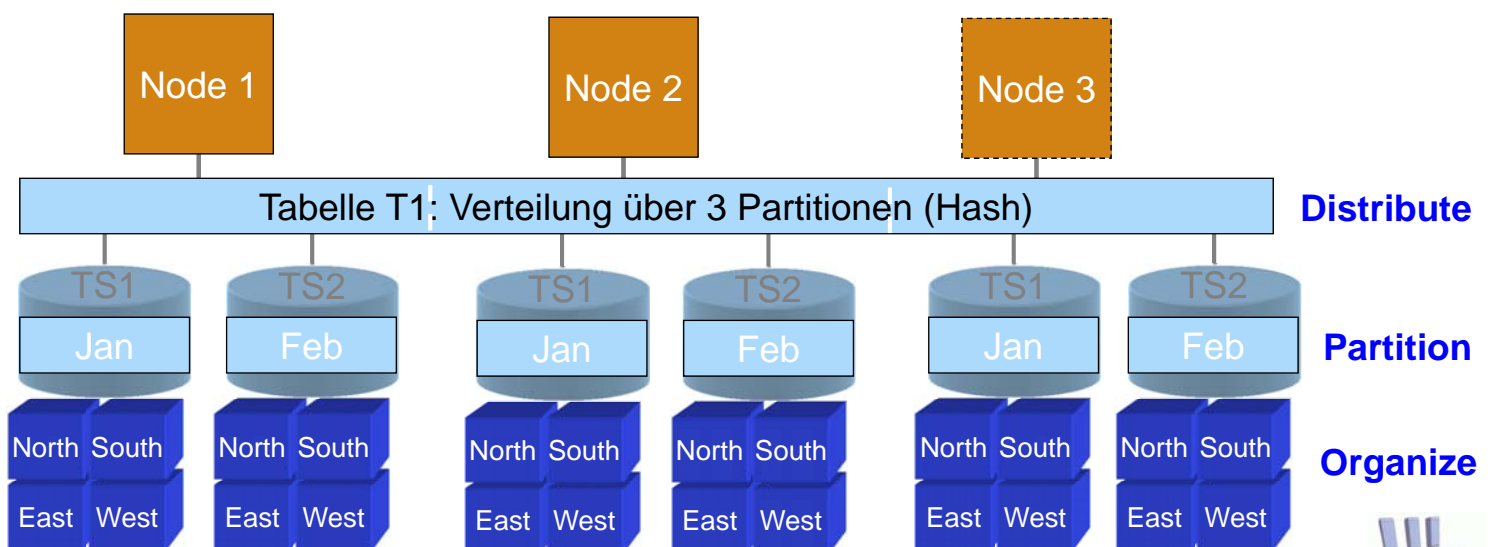
```
CREATE TABLE events
(event_id NUMBER(10),
event_data BLOB)
PARTITION BY RANGE(event_id) (
PARTITION evts_0_100k
VALUES LESS THAN (100000)
TABLESPACE tsa,
PARTITION evts_100k_200k
VALUES LESS THAN (200000)
TABLESPACE tsb,
PARTITION evts_200k_300k
VALUES LESS THAN (300000)
TABLESPACE tsc
);
```

Partitionen können unterschiedlichen Table Spaces zugeordnet werden (Parallele I/O; unabhängiges Backup; etc.)



## DB2 für LUW: Datenallokation (SN)

- 3-stufige Vorgehensweise
  - DISTRIBUTE BY HASH - Tabellen-Partitionierung zwischen Knoten
  - PARTITION BY RANGE – partitionsinterne Fragmentierung von Tabellen
  - ORGANIZE BY DIMENSIONS – Clustering innerhalb von Fragmenten (mehrdimensionales Clustering, MDC)

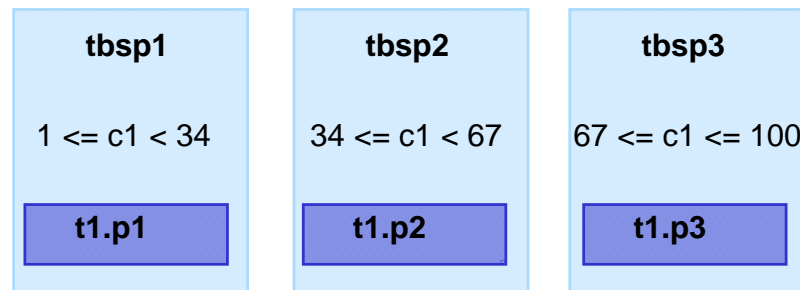




# DB2: Definition Range-Partitioned Table

- Kurze und lange Syntax
- Spezialwerte MINVALUE, MAXVALUE zur Definition offener Wertebereiche, z.B.

CREATE TABLE t1 ... (STARTING (MINVALUE) ENDING (MAXVALUE) ...

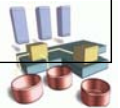


## Kurzform

```
CREATE TABLE t1(c1 INT)
  IN tbsp1, tbsp2, tbsp3
  PARTITION BY RANGE(c1)
  (STARTING FROM (1) ENDING( 100)
   EVERY (33))
```

## Langform

```
CREATE TABLE t1(c1 INT)
  PARTITION BY RANGE(c1)
  (STARTING FROM (1) ENDING(34)IN tbsp1,
   ENDING(67) IN tbsp2,
   ENDING(100) IN tbsp3)
```



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

