

# 7. Hash-basierte Zugriffspfade

- **Ziele**

- Nutzung der Schlüsseltransformation als Entwurfsprinzip für Zugriffspfade auf die Sätze einer Tabelle bei denen ein Suchkriterium unterstützt wird
- Einschränkung auf Schlüsselzugriff, keine Bereichssuche usw.

- **Schnellerer Schlüsselzugriff erfordert Hash-Verfahren**

- Hash-Verfahren auf Externspeichern
  - Statische Verfahren
  - Dynamisches Hashing
- (nur) direkter Zugriff
- idealerweise 1 Seitenzugriff

- **Erweiterbares Hashing**

- Kombination von Konzepten der Digitalbäume und B-Bäume
- Erweiterbares Hashing unterstützt stark wachsende Datenbestände ( $\leq 2$  Seitenzugriffe)

- **Externes Hashing mit Separatoren**

- **Lineares Hashing**

- **Wichtige Kenngrößen:**

- $n$  = #Sätze eines Satztyps
- $b$  = #Sätze/Bucket (Kapazität)
- $N$  = #Buckets
- $\beta$  = Belegungsfaktor

# Gestreute Speicherungsstrukturen (Hash-Verfahren)

- **Direkte Berechnung** der Satzadresse über Schlüssel (Schlüsseltransformation)

- **Hash-Funktion**

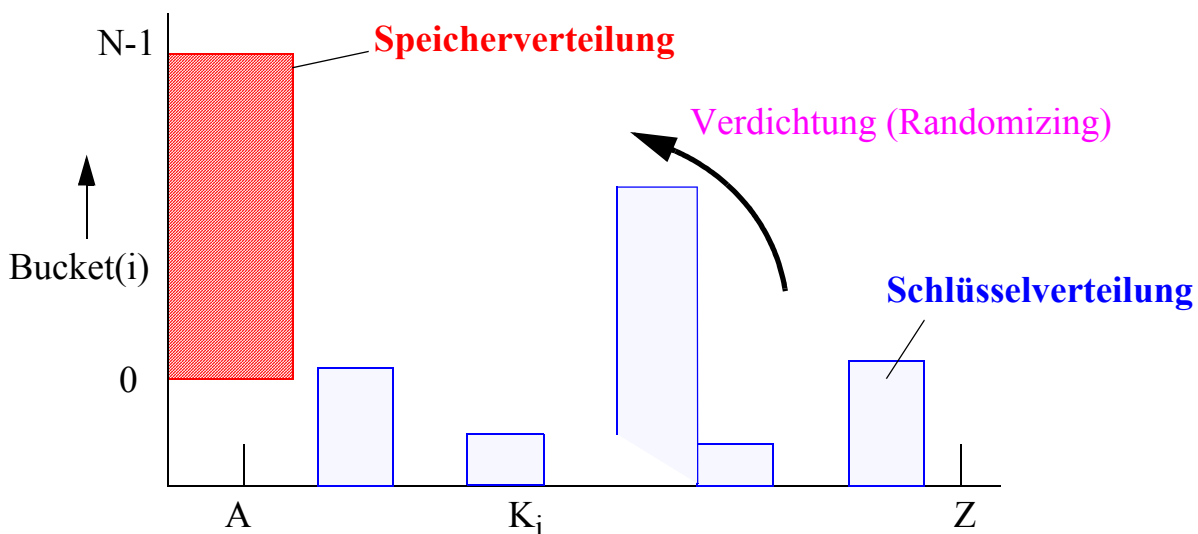
$h: S \rightarrow \{0, 1, \dots, N-1\}$      $S =$  Schlüsselraum  
 $N =$  Größe des statischen Hash-Bereiches  
in Seiten (*Buckets*)

- **Idealfall:  $h$  ist injektiv (keine Kollisionen)**

- nur in Ausnahmefällen möglich ('dichte' Schlüsselmenge)
- jeder Satz kann mit einem Seitenzugriff gefunden werden

- **Statische Hash-Bereiche mit Kollisionsbehandlung**

- vorhandene Schlüsselmenge  $K$  ( $K \subseteq S$ ) soll möglichst gleichmäßig auf die  $N$  Buckets verteilt werden



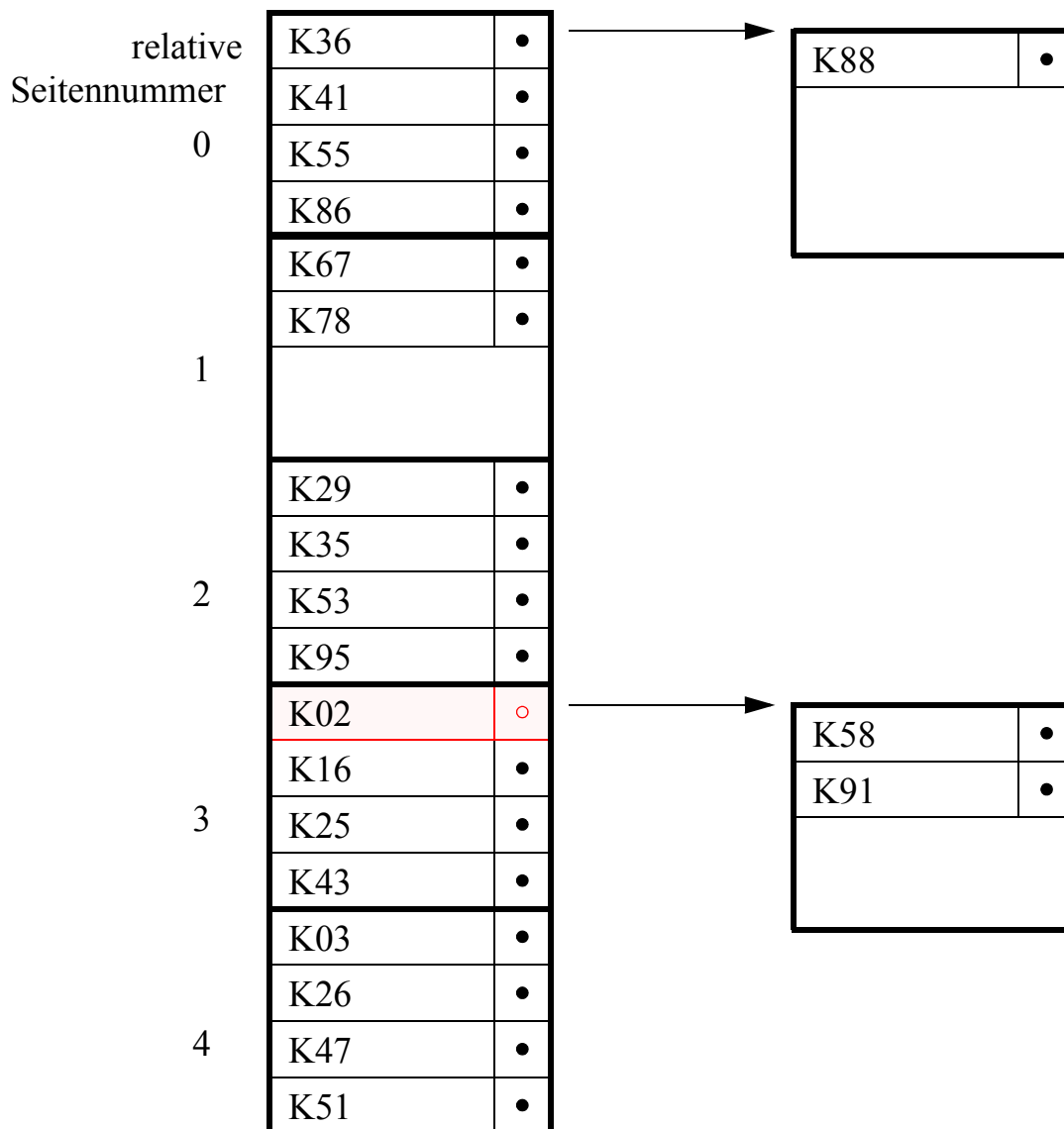
- Behandlung von Synonymen
  - Aufnahme im selben Bucket, wenn möglich
  - ggf. Anlegen und Verketteten von Überlaufseiten
- typischer Zugriffsfaktor: 1.1 bis 1.4
- **Vielzahl von Hash-Funktionen anwendbar**  
z. B. Divisionsrestverfahren, Faltung, Codierungsmethode, ...

## Statisches Hash-Verfahren mit Überlaufbereichen: Beispiel

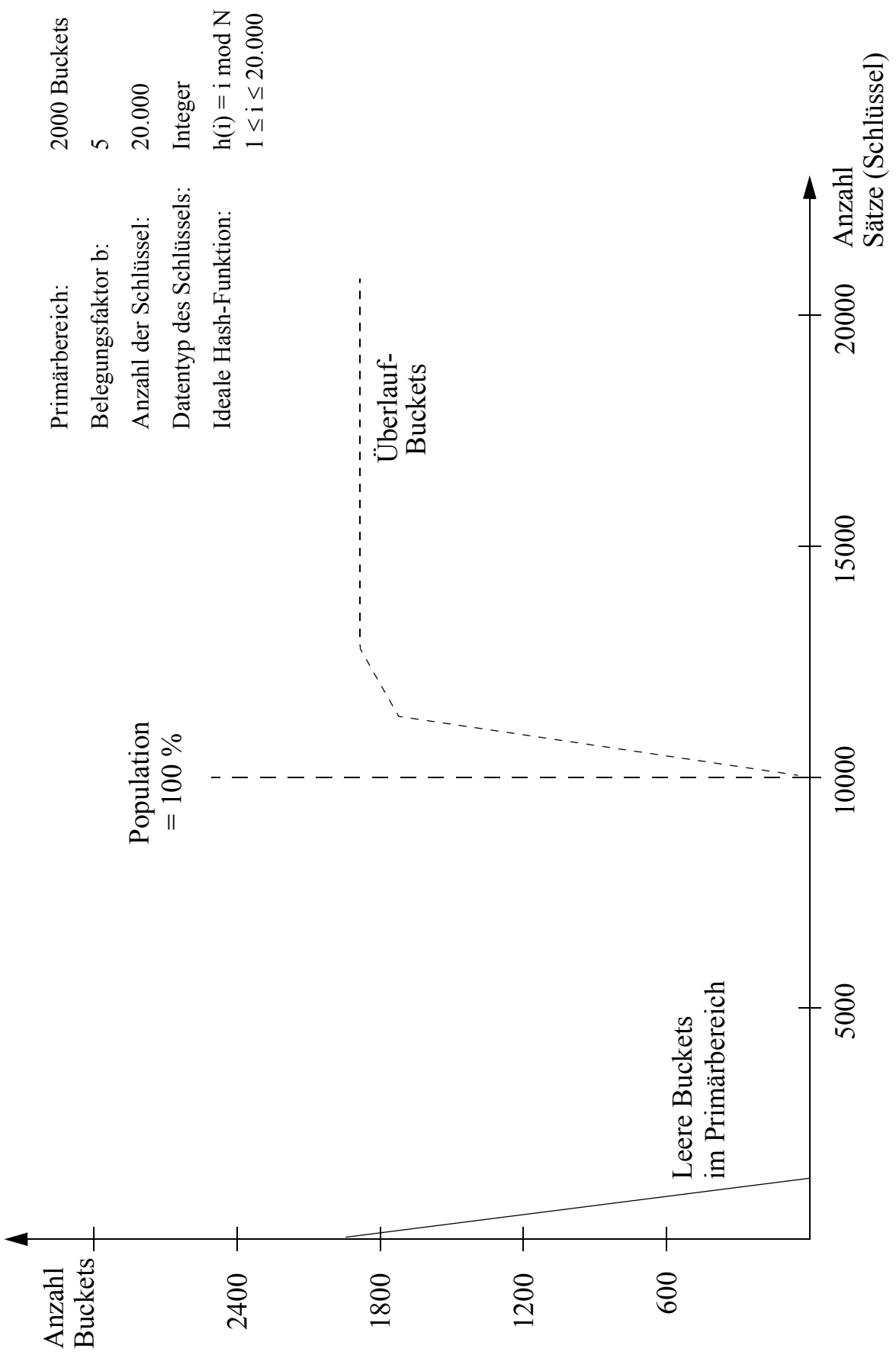
- **Adreßberechnung für Schlüssel K02:**

$$\begin{array}{r}
 1101\ 0010 \\
 \oplus 1111\ 0000 \\
 \oplus 1111\ 0010 \\
 \hline
 1101\ 0000 = 208_{10}
 \end{array}$$

$$208 \bmod 5 = 3$$

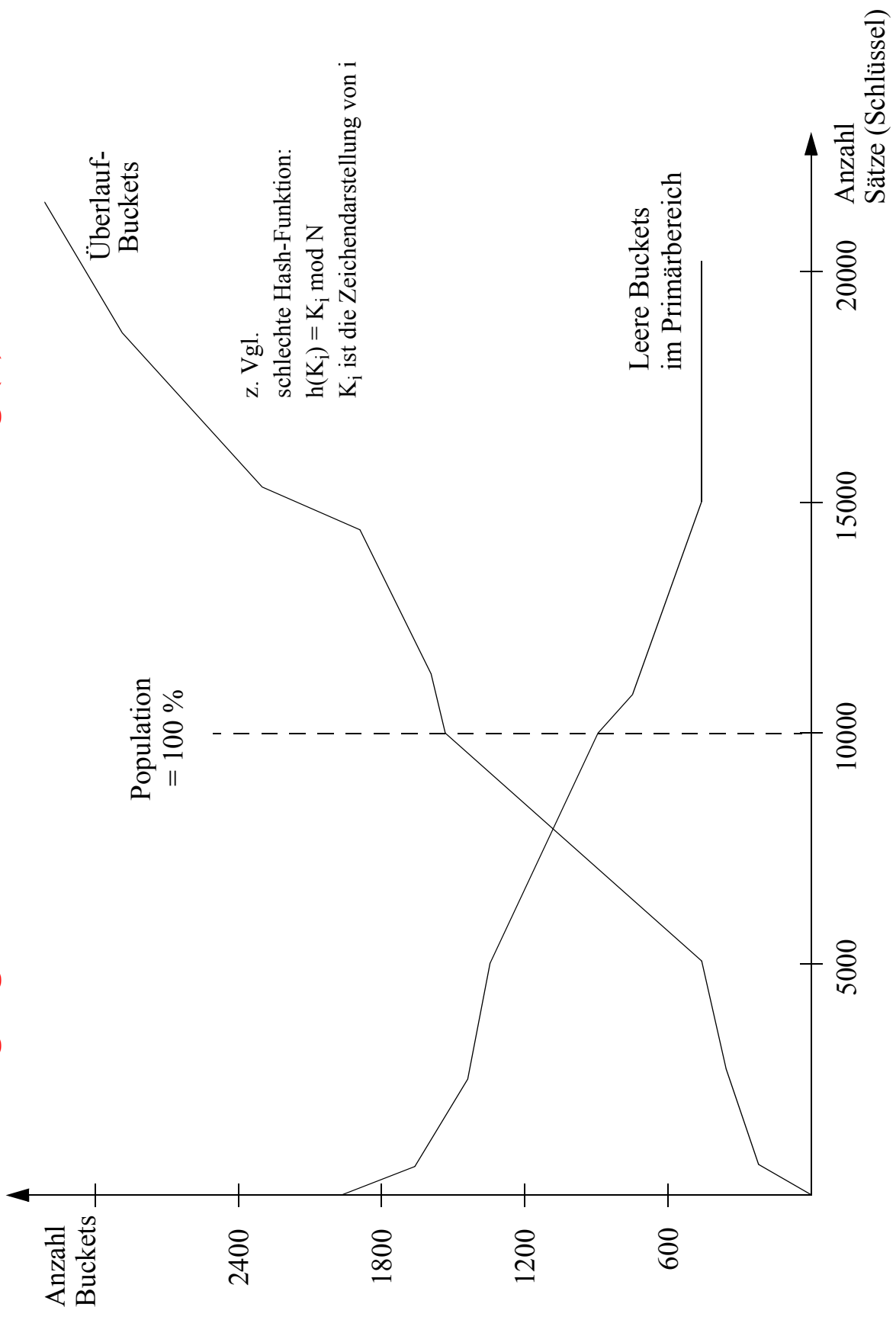


# Belegung von Hash-Bereichen – Messung (1)



Primärbereich: 2000 Buckets  
 Belegungsfaktor b: 5  
 Anzahl der Schlüssel: 20.000  
 Datentyp des Schlüssels: Integer  
 Ideale Hash-Funktion:  $h(i) = i \bmod N$   
 $1 \leq i \leq 20.000$

# Belegung von Hash-Bereichen – Messung (2)

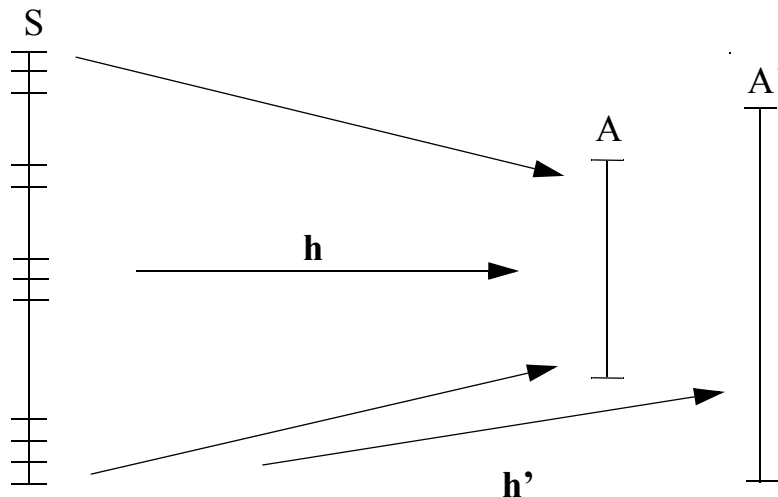


# Dynamische Hash-Verfahren

- **Wachstumsproblem bei statischen Verfahren**

- Statische Allokation von Speicherbereichen: Speicherausnutzung?
- Bei Erweiterung des Adreßraums: Rehashing

➔ **Kosten, Verfügbarkeit, Adressierbarkeit**



➔ **Alle Sätze erhalten eine neue Adresse**

- **Entwurfsziele**

- Dynamische Struktur erlaubt Wachstum und Schrumpfung des Hash-Bereichs (Datei)
- Keine Überlauftechniken
- Zugriffsfaktor  $\leq 2$  für die direkte Suche

- **Viele konkurrierende Ansätze**

- Extensible Hashing (Fagin et al., 1978)
- Virtual Hashing und Linear Hashing (Litwin, 1978, 1980)
- Dynamic Hashing (Larson, 1978)

➔ **Lösungsvorschläge mit und ohne Index (Hilfsdaten)**

„Any hashing which may dynamically change its hashing function“.

# Erweiterbares Hashing

- **Lösungsidee: Verknüpfung der**

- von den B-Bäumen bekannten Split- und Mischtechniken von Seiten zur Konstruktion eines dynamischen Hash-Bereichs mit der
- von den Digitalbäumen her bekannten Adressierungstechnik zum Aufsuchen eines Speicherplatzes

- **Prinzipielle Vorgehensweise**

- Die einzelnen Bits eines Schlüssels steuern den Weg durch den zur Adressierung benutzten Digitalbaum.
- $K_i = (b_0, b_1, b_2, \dots)$ . Es ist prinzipiell möglich, die Bitfolge von  $K_i$  direkt für die Adressierung heranzuziehen. Bei Ungleichverteilung der Schlüssel ist dann ein unausgewogener Digitalbaum zu erwarten.
- Da Digitalbäume keinen Balancierungsmechanismus für ihre Höhe besitzen, muß die Ausgewogenheit "von außen" aufgezwungen werden.
- $h(K_i) = (b_0, b_1, b_2, \dots)$ . Die Verwendung von  $h(K_i)$  als sog. Pseudoschlüssel (PS) soll bessere Gleichverteilung gewährleisten.

- **Gleichverteilung der Pseudoschlüssel PS**

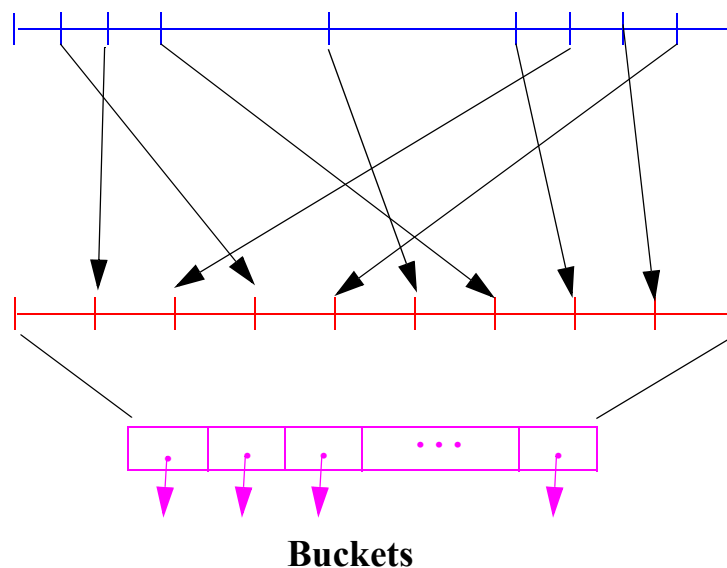
impliziert minimale Höhe des Digitalbaumes

Ungleichverteilung  
der Schlüssel K

$h(K_i) \rightarrow$  PS

Gleichverteilung  
der PS

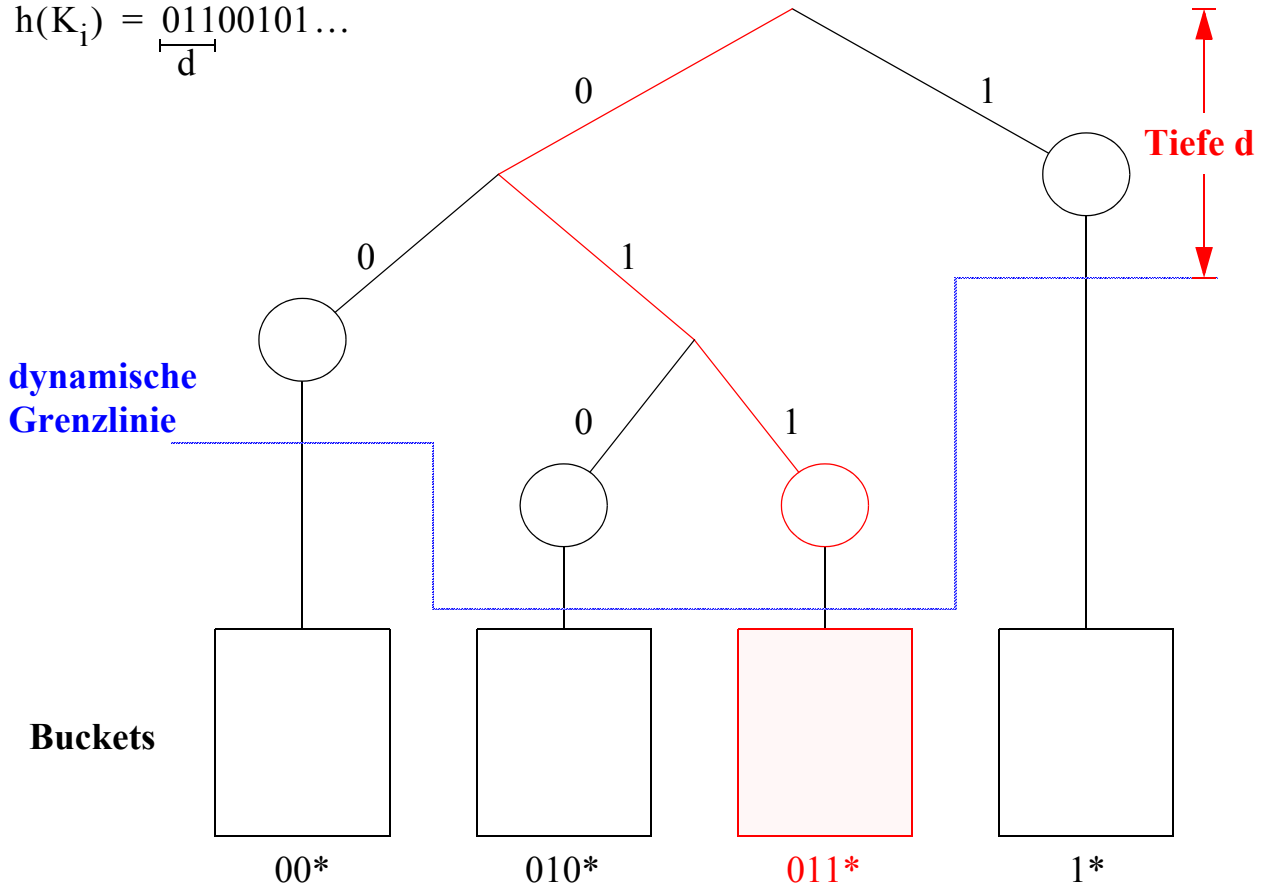
PS werden abgebildet  
auf Directory



# Erweiterbares Hashing<sup>1</sup> (2)

- Prinzipielle Abbildung der Pseudoschlüssel

$$h(K_i) = \underbrace{01100101\dots}_d$$



- Zur Adressierung eines Buckets sind  $d$  Bits erforderlich, wobei sich dafür i. allg. eine dynamische Grenzlinie variierender Tiefe ergibt.
- Die Digitalbaum-Adressierung bricht ab, sobald ein Bucket den ganzen Teilbaum aufnehmen kann.

➔ ausgeglichener Digitalbaum garantiert minimales  $d_{\max}$

- Dynamisches Wachsen und Schrumpfen des Hash-Bereiches

- Buckets werden erst bei Bedarf bereitgestellt
- Knoten unterschiedlicher Tiefe verweisen auf ein Bucket

➔ hohe Speicherplatzbelegung möglich

1. Fagin, R., et. al: Extendible hashing— a fast access method for dynamic files. ACM Trans. Database Syst. 4:3. 1979. 315-344



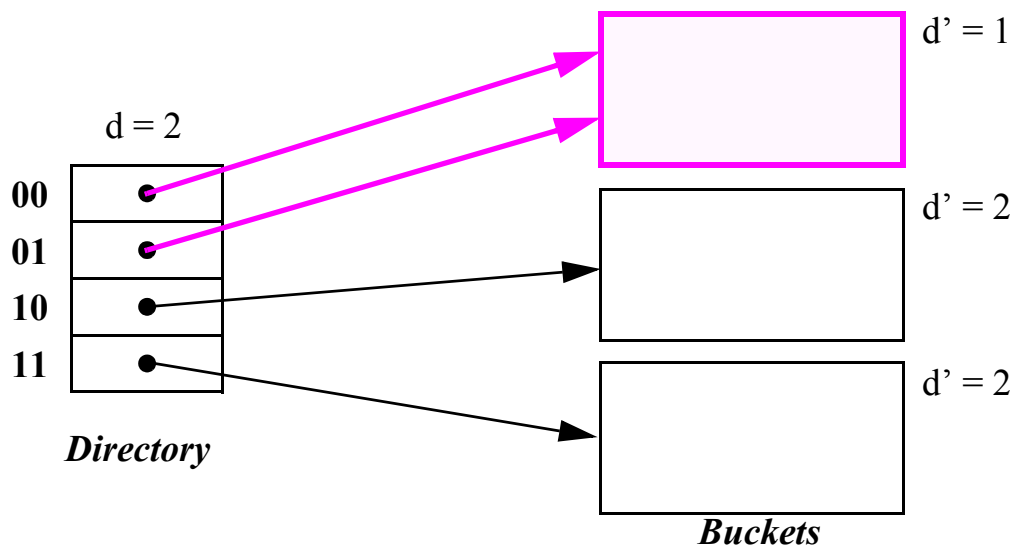
## Erweiterbares Hashing (3)

- Verfahren benötigt keine Überlaufbereiche, jedoch erfolgt Zugriff über *Directory* (Index)

- Binärer Digitalbaum der Höhe  $d$  wird durch einen  $(2^d)$ -Digitalbaum der Höhe 1 implementiert (Trie der Höhe 1 mit  $2^d$  Einträgen)
- $d$  wird festgelegt durch den längsten Pfad im binären Digitalbaum
- In einem Bucket werden nur Sätze gespeichert, deren PS in den ersten  $d'$  Bits übereinstimmen ( $d'$  = lokale Tiefe)
- $d = \text{MAX}(d')$ :  $d$  Bits des PS werden zur Adressierung verwendet ( $d$  = globale Tiefe)
- Directory enthält  $2^d$  Einträge

- **Speicherungsstruktur**

Der Trie läßt sich als Directory oder Adreßverzeichnis auffassen. Die  $d$  Bits von  $h(K_i)$  zeigen im Directory auf einen Eintrag mit der Adresse des Buckets, das den Schlüssel  $K_i$  enthält. Wenn  $d' < d$ , können (benachbarte) Einträge auf dasselbe Bucket verweisen.



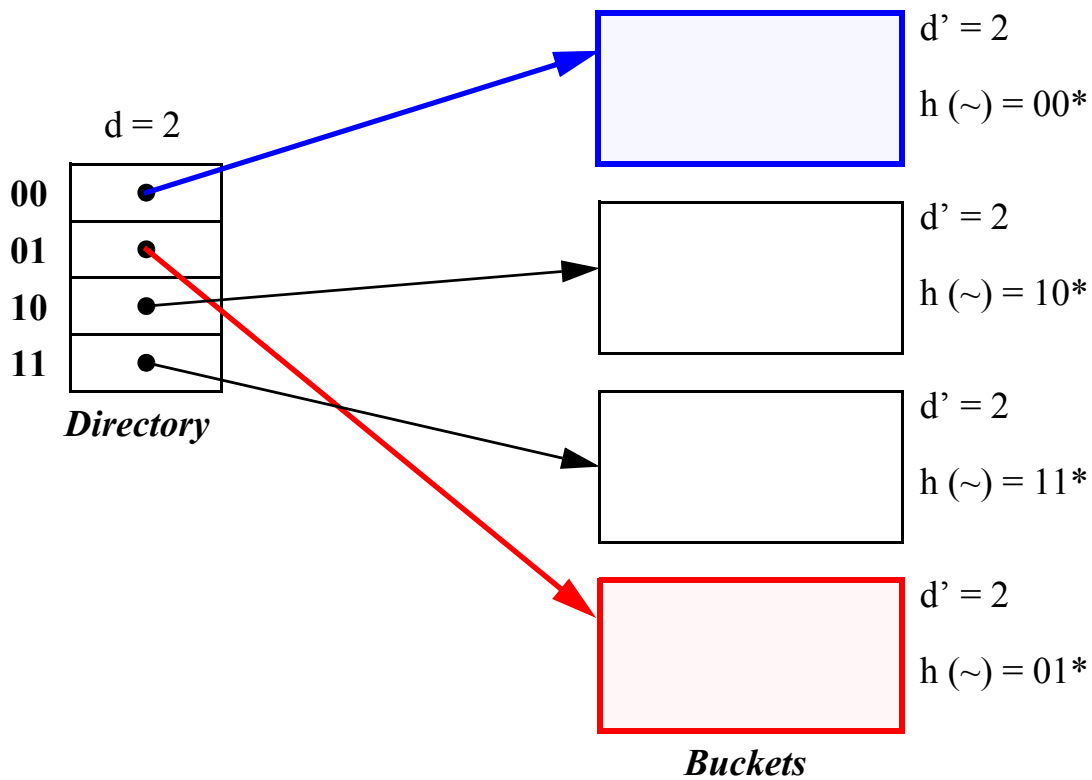
➔ Kosten der direkten Suche: **max. 2 Seitenzugriffe**

## Erweiterbares Hashing: Splitting von Buckets (1)

- **Fall 1: Überlauf eines Buckets, dessen lokale Tiefe kleiner als die globale Tiefe  $d$  ist**

➔ Anlegen eines neuen Buckets (Split) mit

- lokaler Neuverteilung der Daten
- Erhöhung der lokalen Tiefe
- lokaler Korrektur der Verweise im Directory

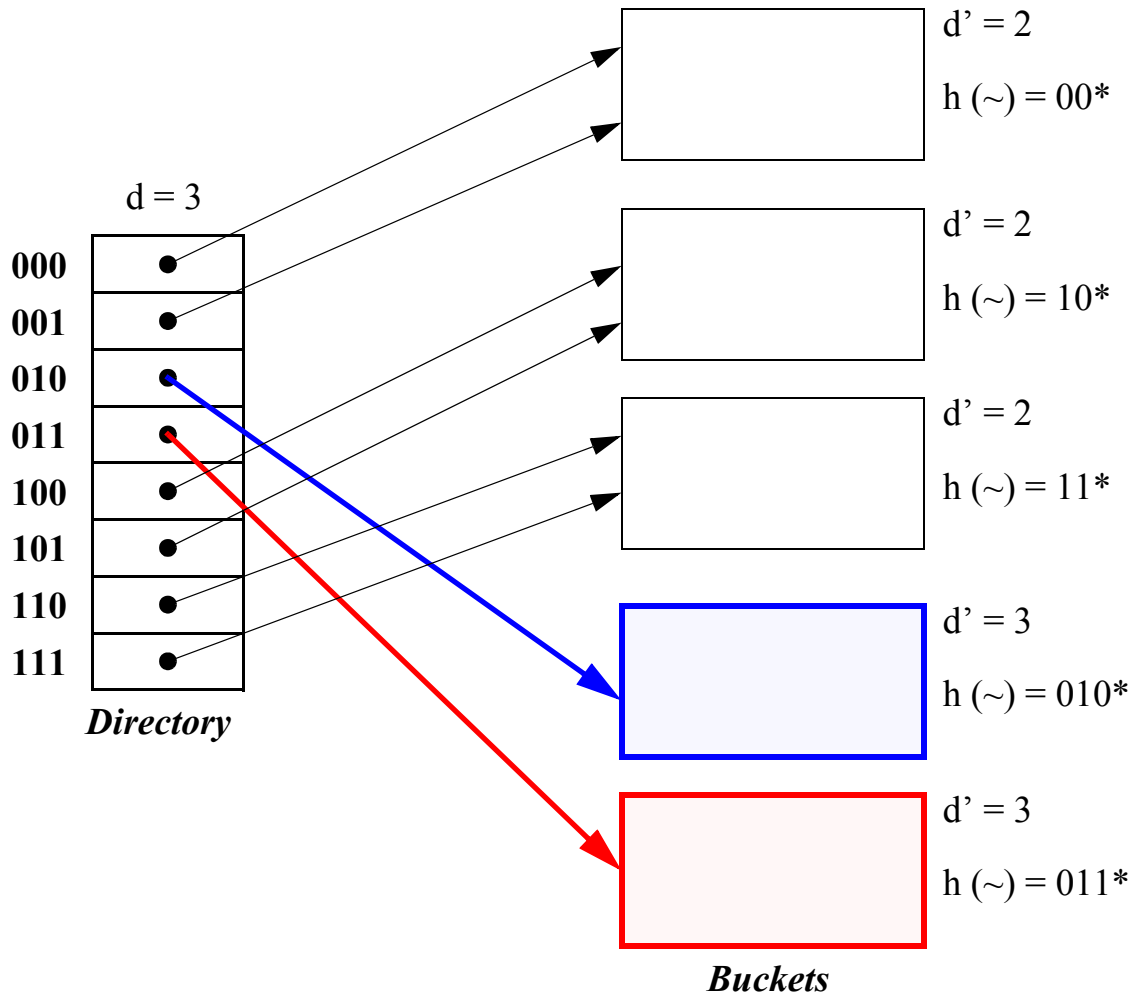


## Erweiterbares Hashing: Splitting von Buckets (2)

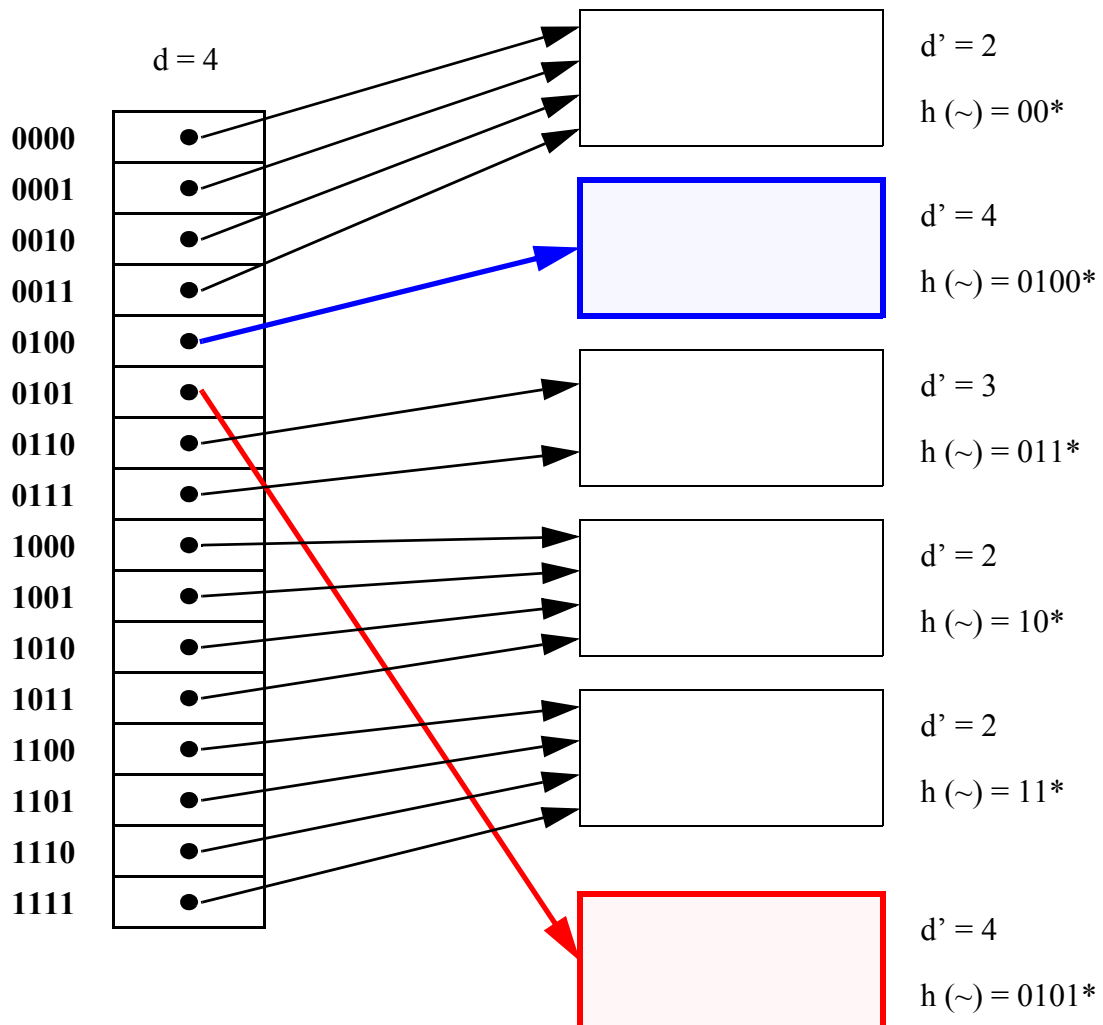
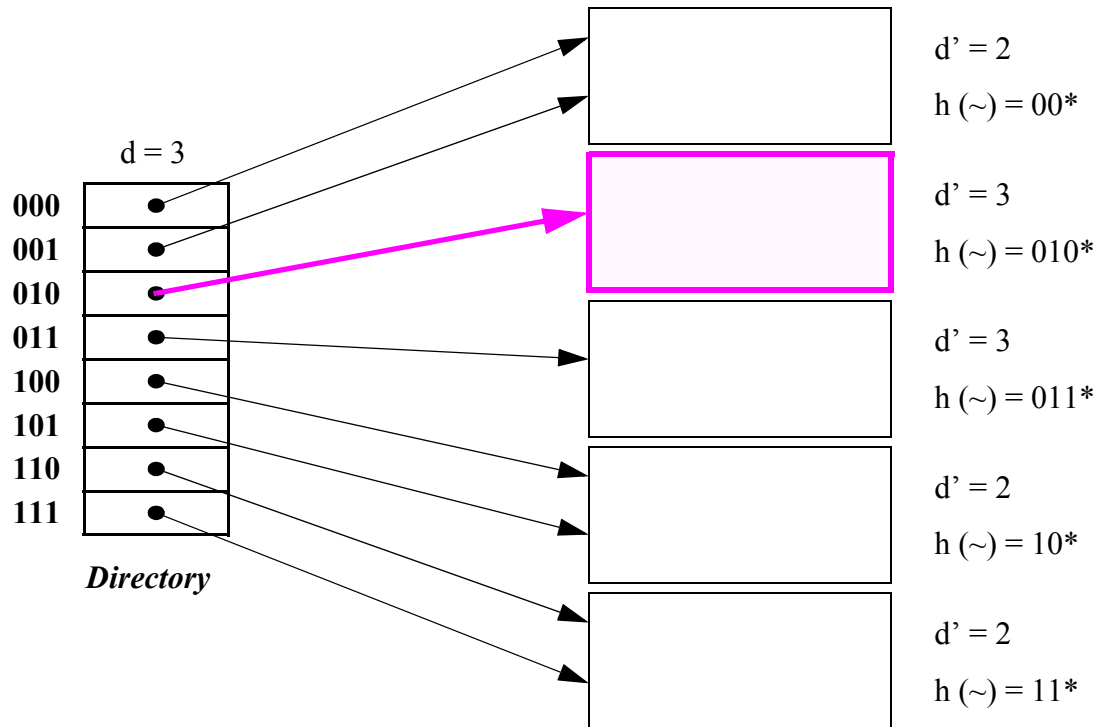
- **Fall 2: Überlauf eines Buckets, dessen lokale Tiefe gleich der globalen Tiefe ist**

➔ Anlegen eines neuen Buckets (Split) mit

- lokaler Neuverteilung der Daten (Erhöhung der lokalen Tiefe)
- Verdopplung des Directories (Erhöhung der globalen Tiefe)
- globaler Korrektur/Neuverteilung der Verweise im Directory



## Erweiterbares Hashing: Splitting von Buckets (3)



# Externes Hashing ohne Überlaufbereiche

- **Ziel**

- Jeder Satz kann mit genau einem E/A-Zugriff gefunden werden

→ Gekettete Überlaufbereiche können nicht benutzt werden

- **Statisches Hashing**

- n Sätze, N Buckets mit Kapazität b

- Belegungsfaktor  $\beta = \frac{n}{N \cdot b}$

- **Überlaufbehandlung**

- Open Adressing (ohne Kette oder Zeiger)

- Bekannteste Schemata: Lineares Sondieren und Double Hashing

- Sondierungsfolge für einen Satz mit Schlüssel k:

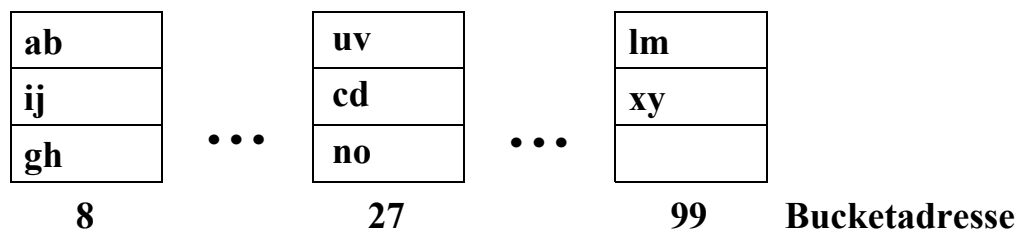
- $H(k) = (h_1(k), h_2(k), \dots, h_N(k))$

- bestimmt Überprüfungsreihenfolge der Buckets (Seiten) beim Einfügen und Suchen

- wird durch k festgelegt und ist eine Permutation der Menge der Bucketadressen  $\{0, 1, \dots, N - 1\}$

- **Erster Versuch**

- Aufsuchen oder Einfügen von  $k = xy$



- Sondierungsfolge sei  $H(xy) = (8, 27, 99, \dots)$

- Viele E/A-Zugriffe

- Wie geschieht das Einfügen?

# Externes Hashing mit Separatoren<sup>2</sup>

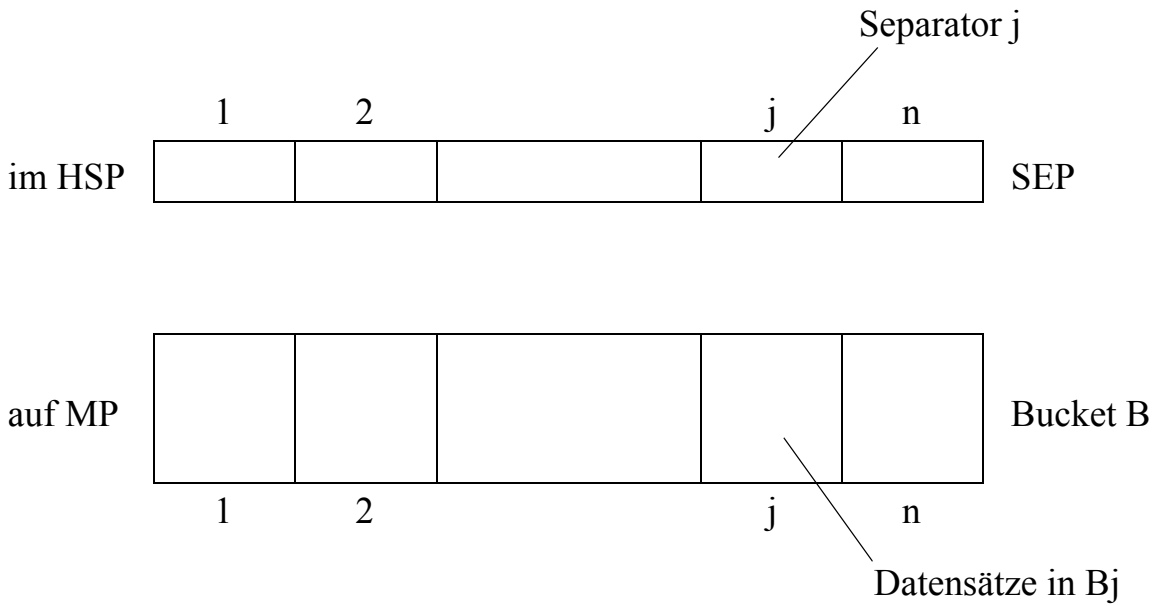
- **Zugriffspfad für Primärschlüssel**
- **Einsatz von Signaturen**
  - Jede Signatur  $s_i(k)$  ist ein t-Bit Integer
  - Für jeden Satz mit Schlüssel k wird eine Signaturfolge benötigt:  
$$S(k) = (s_1(k), s_2(k), \dots, s_N(k))$$
  - Die Signaturfolge wird eindeutig durch k bestimmt
  - Die Berechnung von  $S(k)$  kann durch einen Pseudozufallszahlen-Generator mit k als Saat erfolgen  
(Gleichverteilung der t Bits wichtig)
- **Nutzung der Signaturfolge zusammen mit der Sondierungsfolge**
  - Bei Sondierung  $h_i(k)$  wird  $s_i(k)$  benutzt,  $i = 1, 2, \dots, N$
  - Für jede Sondierung wird eine neue Signatur berechnet!
- **Einsatz von Separatoren**
  - Ein Separator besteht aus t Bits
  - Separator  $j$ ,  $j = 0, 1, 2, \dots, N-1$ , gehört zu Bucket j
  - Eine Separatortabelle SEP enthält die N Separatoren und wird im Hauptspeicher gehalten.

---

2. Larson, P.-A. and Kajla, A.: File organization: implementation of a method guaranteeing retrieval in one access, in: Comm. of the ACM 27,7 (1984), 670-677.

# Externes Hashing mit Separatoren – Beispiel

- Speicherungsmodell



- Prinzip des Hashing

### 1. Sondierungsfolge

$$H(k) = (h_1(k), h_2(k), \dots, h_n(k))$$

$$H(xz) = (8, 27, 99, \dots, 1)$$

Bucketadr.  $\longleftarrow$

### 2. Signaturfolge

$$S(xz) = (0110, 1100, \dots, 1100)$$

$$S(k) = (s_1(k), s_2(k), \dots, s_n(k))$$

Signatur  $\longleftarrow$

## Externes Hashing mit Separatoren (2)

### • Nutzung der Separatoren

- Wenn Bucket  $B_i$   $r$ -mal ( $r > b$ ) sondiert wurde, müssen mindestens  $(r - b)$  Sätze abgewiesen werden; sie müssen das nächste Bucket in ihrer Sondierungsfolge aufsuchen.
- Für die Entscheidung, welche Sätze im Bucket gespeichert werden, sind die  $r$  Sätze nach ihren momentanen Signaturen zu sortieren.
- Sätze mit niedrigen Signaturen werden in  $B_i$  gespeichert, die mit hohen Signaturen müssen weitersuchen.
- Eine Signatur, die die Gruppe der niedrigen eindeutig von der der höheren Signaturen trennt, wird als Separator  $j$  für  $B_j$  in SEP aufgenommen. Separator  $j$  enthält den niedrigsten Signaturwert der Sätze, die weitersuchen müssen.
- Ein Separator partitioniert also die  $r$  Sätze von  $B_j$ . Wenn die ideale Partitionierung  $(b, r - b)$  nicht gewählt werden kann, wird eine der folgenden versucht:

$(b - 1, r - b + 1), (b - 2, r - b + 2), \dots, (0, r)$

➔ Ein Bucket mit Überlaufsätzen kann weniger als  $b$  Sätze gespeichert haben.

### • Beispiel: $r = 5, t = 4$

- Signaturen 

0001	}	für Bucket $B_i$
0011		
0100		
0100		
1000		

- $b = 4$ : Separator = 1000, Aufteilung (4, 1)

➔ SEP  $[j] = 1000$

- $b = 3$ : Separator = 0100, Aufteilung (2, 3)

➔ SEP  $[j] = 0100$



## Externes Hashing mit Separatoren (3)

### • **Aufsuchen**

- In der Sondierungsfolge  $S(k)$  werden die  $s_i(k)$  mit  $SEP[h_i(k)]$ ,  $i = 1, 2, \dots$ , im Hauptspeicher verglichen.
- Sobald ein  $SEP[h_i(k)] > s_i(k)$  gefunden wird, ist die richtige Bucketadresse  $h_i(k)$  lokalisiert.
- Das Bucket wird eingelesen und durchsucht. Wenn der Satz nicht gefunden wird, existiert er nicht.

➔ Es ist genau ein E/A-Zugriff erforderlich

### • **Einfügen**

- kann Verschieben von Sätzen und Ändern von Separatoren erfordern.
- Wenn für einen Satz  $s_i(k) < SEP[j]$  mit  $j = h_i(k)$  gilt, muß er in Bucket  $B_j$  eingefügt werden.
- Falls  $B_j$  schon voll ist, müssen ein oder mehrere Sätze verschoben und  $SEP[j]$  entsprechend aktualisiert werden.
- Alle verschobenen Sätze müssen dann in Buckets ihrer Sondierungsfolgen wieder eingefügt werden

➔ Dieser Prozeß kann kaskadieren

➔  $\beta$  nahe bei 1 ist unsinnig, da die Einfügekosten explodieren;

Empfehlung:  $\beta < 0,8$

## Externes Hashing mit Separatoren (4)

- **Initialisierung der Separatoren mit  $2^t - 1$**

- Separator eines Buckets, das noch nicht übergelaufen ist, soll größer als alle Signaturen sein

→ einfachere Algorithmen

- daher Bereich der Signaturen:  $0, 1, \dots, 2^t - 2$  .

- **Beispiel: Startsituation**

1111	1111	1111	1111	Separator																																		
<table style="width: 100%; border-collapse: collapse;"> <thead> <tr><th style="text-align: left;">Key</th><th style="text-align: left;">Sign.</th></tr> </thead> <tbody> <tr><td>ab</td><td>0100</td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>	Key	Sign.	ab	0100					<table style="width: 100%; border-collapse: collapse;"> <thead> <tr><th style="text-align: left;">Key</th><th style="text-align: left;">Sign.</th></tr> </thead> <tbody> <tr><td>ef</td><td>0010</td></tr> <tr><td>cd</td><td>0101</td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>	Key	Sign.	ef	0010	cd	0101			<table style="width: 100%; border-collapse: collapse;"> <thead> <tr><th style="text-align: left;">Key</th><th style="text-align: left;">Sign.</th></tr> </thead> <tbody> <tr><td>uv</td><td>0101</td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>	Key	Sign.	uv	0101					<table style="width: 100%; border-collapse: collapse;"> <thead> <tr><th style="text-align: left;">Key</th><th style="text-align: left;">Sign.</th></tr> </thead> <tbody> <tr><td>lm</td><td>0100</td></tr> <tr><td>xy</td><td>0110</td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>	Key	Sign.	lm	0100	xy	0110					
Key	Sign.																																					
ab	0100																																					
Key	Sign.																																					
ef	0010																																					
cd	0101																																					
Key	Sign.																																					
uv	0101																																					
Key	Sign.																																					
lm	0100																																					
xy	0110																																					
8	18	27	99	Bucketadr.																																		

Einfügen von  $k = gh$  mit  $h_1(gh) = 18$  ,  $s_1(gh) = 1110$

Einfügen von  $k = ij$  mit  $h_1(ij) = 18$  ,  $s_1(ij) = 0101$

- **Erster Bucketüberlauf**

$k = gh$  muß weiter sondieren: z.B.:  $h_2(gh) = 99$  ,  $s_2(gh) = 1010$

1111	1110	1111	1111	Separator																																				
<table style="width: 100%; border-collapse: collapse;"> <thead> <tr><th style="text-align: left;">Key</th><th style="text-align: left;">Sign.</th></tr> </thead> <tbody> <tr><td>ab</td><td>0100</td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>	Key	Sign.	ab	0100					<table style="width: 100%; border-collapse: collapse;"> <thead> <tr><th style="text-align: left;">Key</th><th style="text-align: left;">Sign.</th></tr> </thead> <tbody> <tr><td>ef</td><td>0010</td></tr> <tr><td>cd</td><td>0101</td></tr> <tr><td>ij</td><td>0101</td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>	Key	Sign.	ef	0010	cd	0101	ij	0101			<table style="width: 100%; border-collapse: collapse;"> <thead> <tr><th style="text-align: left;">Key</th><th style="text-align: left;">Sign.</th></tr> </thead> <tbody> <tr><td>uv</td><td>0101</td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>	Key	Sign.	uv	0101					<table style="width: 100%; border-collapse: collapse;"> <thead> <tr><th style="text-align: left;">Key</th><th style="text-align: left;">Sign.</th></tr> </thead> <tbody> <tr><td>lm</td><td>0100</td></tr> <tr><td>xy</td><td>0110</td></tr> <tr><td> </td><td> </td></tr> <tr><td> </td><td> </td></tr> </tbody> </table>	Key	Sign.	lm	0100	xy	0110					
Key	Sign.																																							
ab	0100																																							
Key	Sign.																																							
ef	0010																																							
cd	0101																																							
ij	0101																																							
Key	Sign.																																							
uv	0101																																							
Key	Sign.																																							
lm	0100																																							
xy	0110																																							
8	18	27	99	Bucketadr.																																				

## Externes Hashing mit Separatoren (5)

- Situation nach weiteren Einfügungen und Löschungen

1000	1110	1111	1000	Separator																																
<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Key</th> <th style="text-align: left;">Sign.</th> </tr> </thead> <tbody> <tr> <td>ab</td> <td>0100</td> </tr> <tr> <td> </td> <td> </td> </tr> <tr> <td> </td> <td> </td> </tr> </tbody> </table>	Key	Sign.	ab	0100					<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Key</th> <th style="text-align: left;">Sign.</th> </tr> </thead> <tbody> <tr> <td>ef</td> <td>0010</td> </tr> <tr> <td>cd</td> <td>0101</td> </tr> <tr> <td>ij</td> <td>0101</td> </tr> </tbody> </table>	Key	Sign.	ef	0010	cd	0101	ij	0101	<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Key</th> <th style="text-align: left;">Sign.</th> </tr> </thead> <tbody> <tr> <td>uv</td> <td>0101</td> </tr> <tr> <td>mn</td> <td>1001</td> </tr> <tr> <td> </td> <td> </td> </tr> </tbody> </table>	Key	Sign.	uv	0101	mn	1001			<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Key</th> <th style="text-align: left;">Sign.</th> </tr> </thead> <tbody> <tr> <td>lm</td> <td>0010</td> </tr> <tr> <td>xy</td> <td>0110</td> </tr> <tr> <td> </td> <td> </td> </tr> </tbody> </table>	Key	Sign.	lm	0010	xy	0110			Bucket
Key	Sign.																																			
ab	0100																																			
Key	Sign.																																			
ef	0010																																			
cd	0101																																			
ij	0101																																			
Key	Sign.																																			
uv	0101																																			
mn	1001																																			
Key	Sign.																																			
lm	0010																																			
xy	0110																																			
<b>8</b>	<b>18</b>	<b>27</b>	<b>99</b>	Bucketadr.																																

- Einfügung von  $H(qr) = (8, 18, \dots)$  und  $S(qr) = (1011, 0011, \dots)$

1000	0101	1111	1000	Separator																																
<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Key</th> <th style="text-align: left;">Sign.</th> </tr> </thead> <tbody> <tr> <td>ab</td> <td>0100</td> </tr> <tr> <td>ij</td> <td>0110</td> </tr> <tr> <td> </td> <td> </td> </tr> </tbody> </table>	Key	Sign.	ab	0100	ij	0110			<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Key</th> <th style="text-align: left;">Sign.</th> </tr> </thead> <tbody> <tr> <td>ef</td> <td>0010</td> </tr> <tr> <td>qr</td> <td>0011</td> </tr> <tr> <td> </td> <td> </td> </tr> </tbody> </table>	Key	Sign.	ef	0010	qr	0011			<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Key</th> <th style="text-align: left;">Sign.</th> </tr> </thead> <tbody> <tr> <td>uv</td> <td>0101</td> </tr> <tr> <td>mn</td> <td>1001</td> </tr> <tr> <td>cd</td> <td>1011</td> </tr> </tbody> </table>	Key	Sign.	uv	0101	mn	1001	cd	1011	<table style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="text-align: left;">Key</th> <th style="text-align: left;">Sign.</th> </tr> </thead> <tbody> <tr> <td>lm</td> <td>0010</td> </tr> <tr> <td>xy</td> <td>0110</td> </tr> <tr> <td> </td> <td> </td> </tr> </tbody> </table>	Key	Sign.	lm	0010	xy	0110			Bucket
Key	Sign.																																			
ab	0100																																			
ij	0110																																			
Key	Sign.																																			
ef	0010																																			
qr	0011																																			
Key	Sign.																																			
uv	0101																																			
mn	1001																																			
cd	1011																																			
Key	Sign.																																			
lm	0010																																			
xy	0110																																			
<b>8</b>	<b>18</b>	<b>27</b>	<b>99</b>	Bucketadr.																																

Sondierungs- und Signaturfolgen von cd und ij seien

$H(cd) = (18, 27, \dots)$  und  $S(cd) = (0101, 1011, \dots)$

$H(ij) = (18, 99, 8, \dots)$  und  $S(ij) = (0101, 1110, 0110, \dots)$

# Lineares Hashing<sup>3</sup>

- **Dynamisches Wachsen und Schrumpfen des (primären) Hash-Bereichs (Datei)**

- minimale Verwaltungsdaten
- keine großen Directories für die Hash-Datei

- **Aber: es gibt keine Möglichkeit, Überlaufsätze vollständig zu vermeiden!**

- eine hohe Rate von Überlaufsätzen wird als Indikator dafür genommen, daß die Datei eine zu hohe Belegung aufweist und deshalb erweitert werden muß
- Buckets werden in einer fest vorgegebenen Reihenfolge gesplittet

↳ **einzigste Information: nächstes zu splittendes Bucket**

- **Prinzipieller Ansatz**

- N: Größe der Ausgangsdatei in Buckets
- Folge von Hash-Funktionen  $h_0, h_1, \dots$

- wobei  $h_0(k) \in \{0, 1, \dots, N-1\}$

- und  $h_{j+1}(k) = h_j(k)$  oder

$$h_{j+1}(k) = h_j(k) + N \cdot 2^j$$

für alle  $j \geq 0$  und alle Schlüssel  $k$  gilt

- gleiche Wahrscheinlichkeit für beide Fälle von  $h_{j+1}$  erwünscht

- **Beispiel**

- $h_j(k) = k(\text{mod } N \cdot 2^j)$  ,  $j = 0, 1, \dots$

---

3. Litwin, W.: Linear hashing: a new tool for files and tables implementation. Proc. 6th Int. Conf.. VLDB. Montreal. 1980. 212-223.

# Lineares Hashing – Beispiel

- Prinzip: LH**

$$h_0(k_i) \in \{0, \dots, N - 1\} = \{0, 1\} \text{ für } N = 2$$

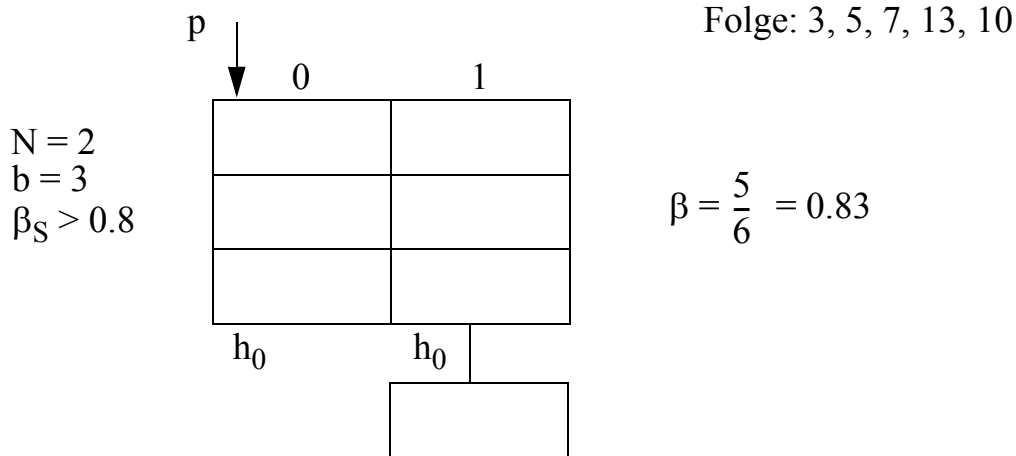
$$\left. \begin{aligned} h_1(k_i) &= h_0(k_i) \text{ oder} \\ h_1(k_i) &= h_0(k_i) + N \cdot 2^0 \end{aligned} \right\} j = 0$$

allgem.:

$$h_{j+1}(k_i) = h_j(k_i) \text{ oder}$$

$$h_{j+1}(k_i) = h_j(k_i) + \underbrace{N \cdot 2^j}_{\text{um } N \cdot 2^j \text{ versetzt}}$$

- $h_0(k_i) = k_i \bmod (2^0 * N)$**

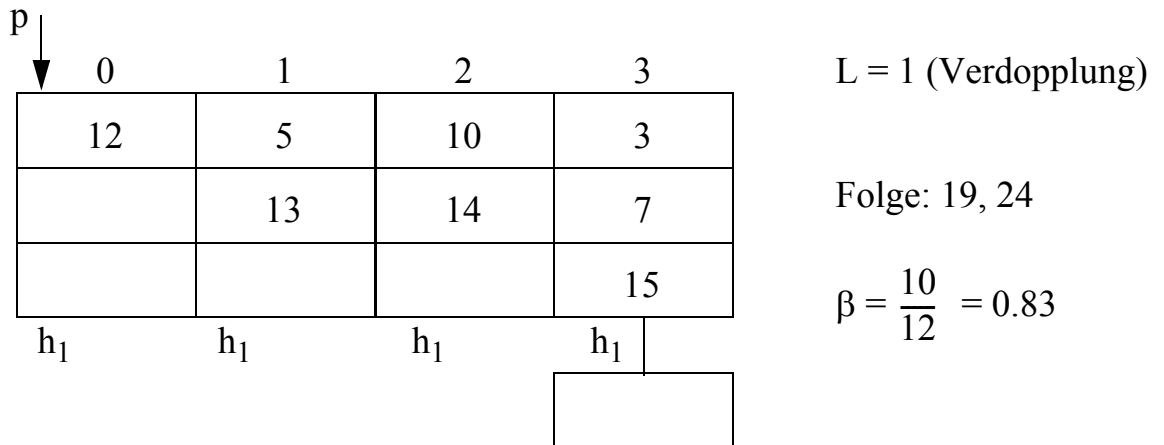
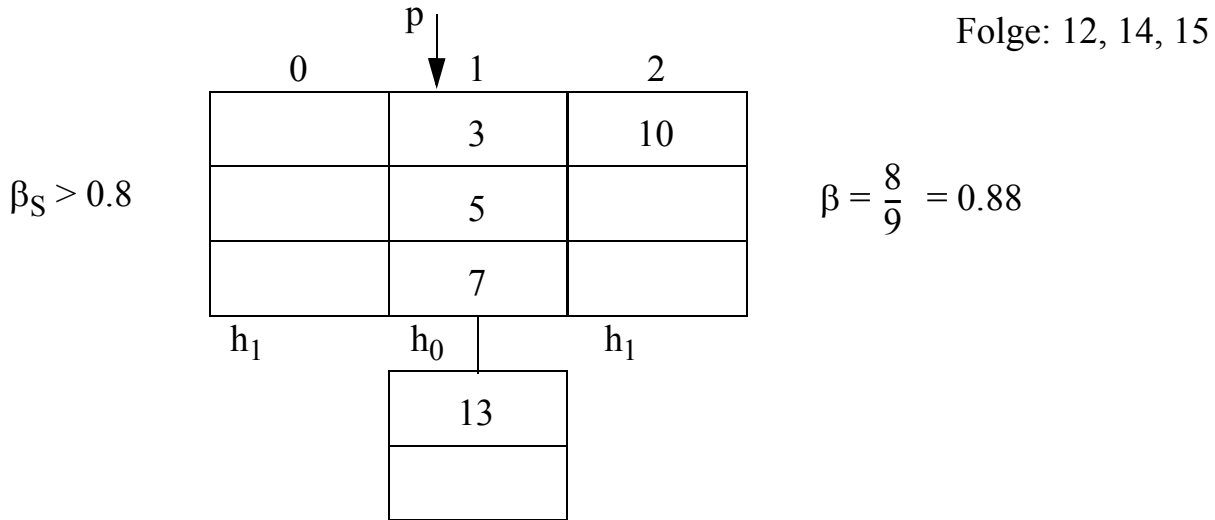


**Erweiterung:**

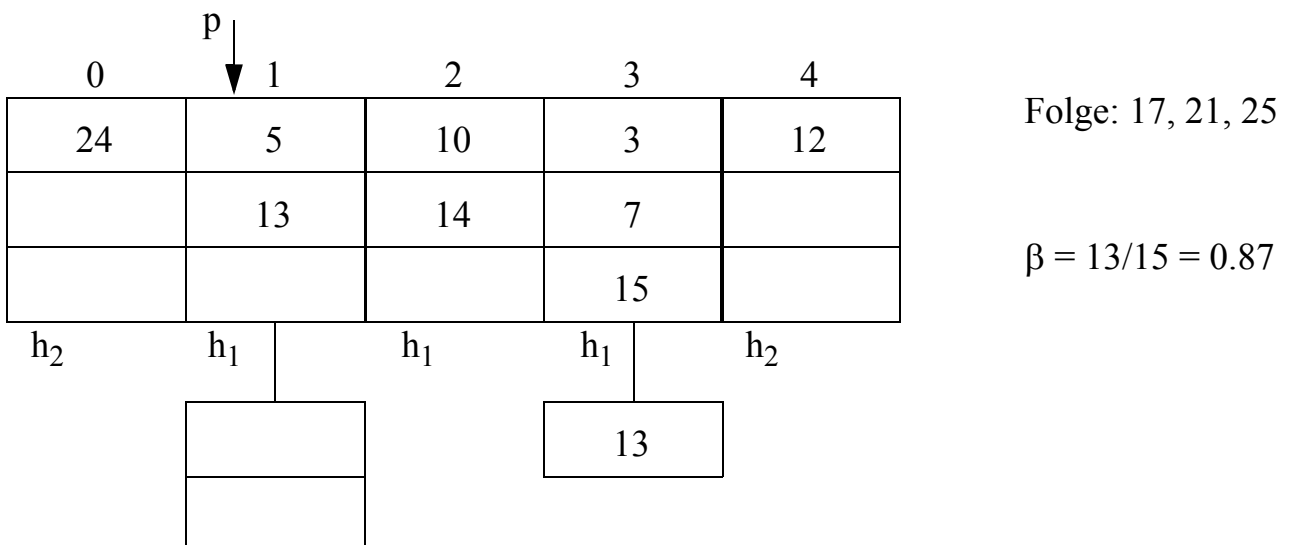
- Neuaufteilung von Bucket  $p$ :       $p := p+1$
- Adressierung:       $h := h_L(k)$       ( $L = 0$ )
- if  $h < p$  then       $h := h_{L+1}(k)$

## Lineares Hashing – Beispiel 2

- $h_1(k_i) = k_i \bmod (2^1 * N)$



- $h_2(k_i) = k_i \bmod (2^2 * N)$



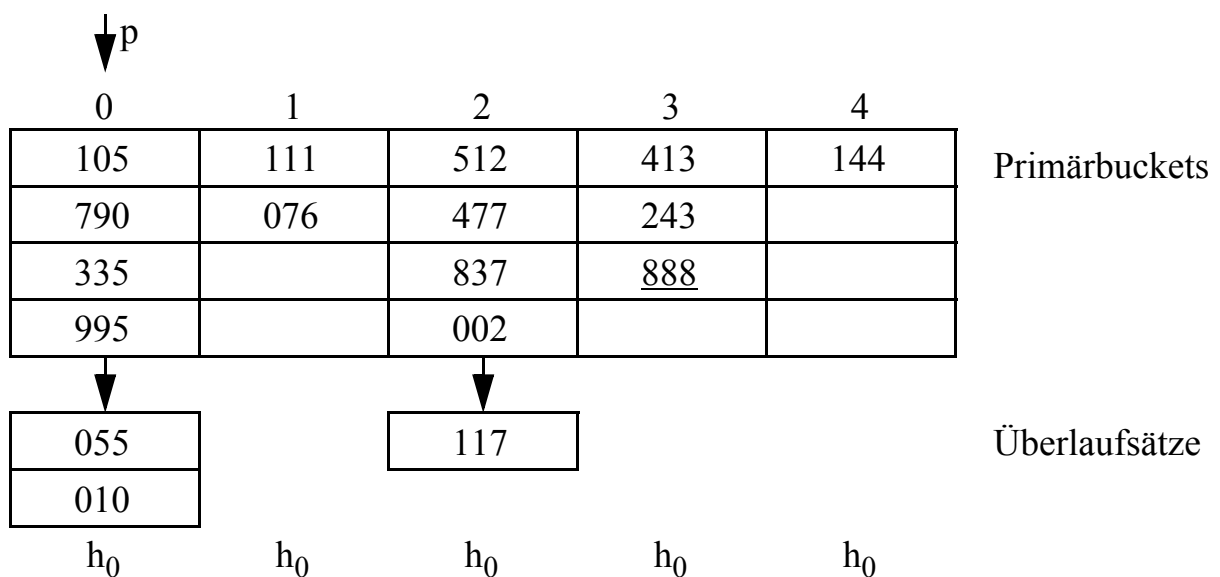
## Lineares Hashing (2)

- **Beschreibung des Dateizustandes**

- L: Anzahl der bereits ausgeführten Verdopplungen
- p: zeigt auf nächstes zu splittendes Bucket ( $0 \leq p < N \cdot 2^L$ )
- $\beta$ : Belegungsfaktor =  $\frac{n}{(N \cdot 2^L + p) \cdot b}$
- n: Anzahl der gespeicherten Sätze
- b: Kapazität eines Buckets

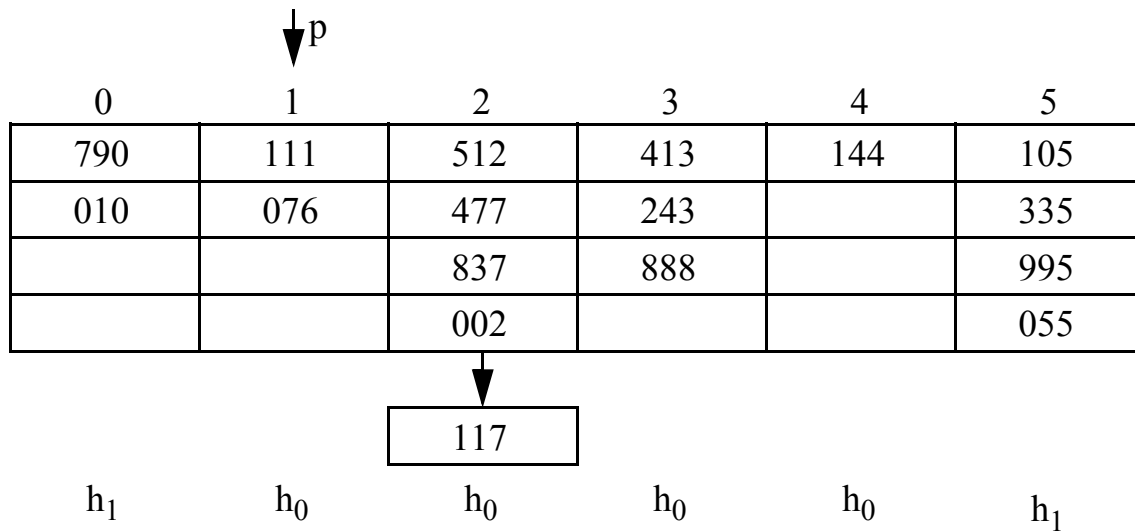
- **Beispiel: Prinzip des linearen Hashing**

- $h_0(k) = k \bmod 5$
- $h_1(k) = k \bmod 10, \dots$
- $b = 4, L = 0, N = 5$
- Splitting, sobald  $\beta > \beta_s = 0.8$

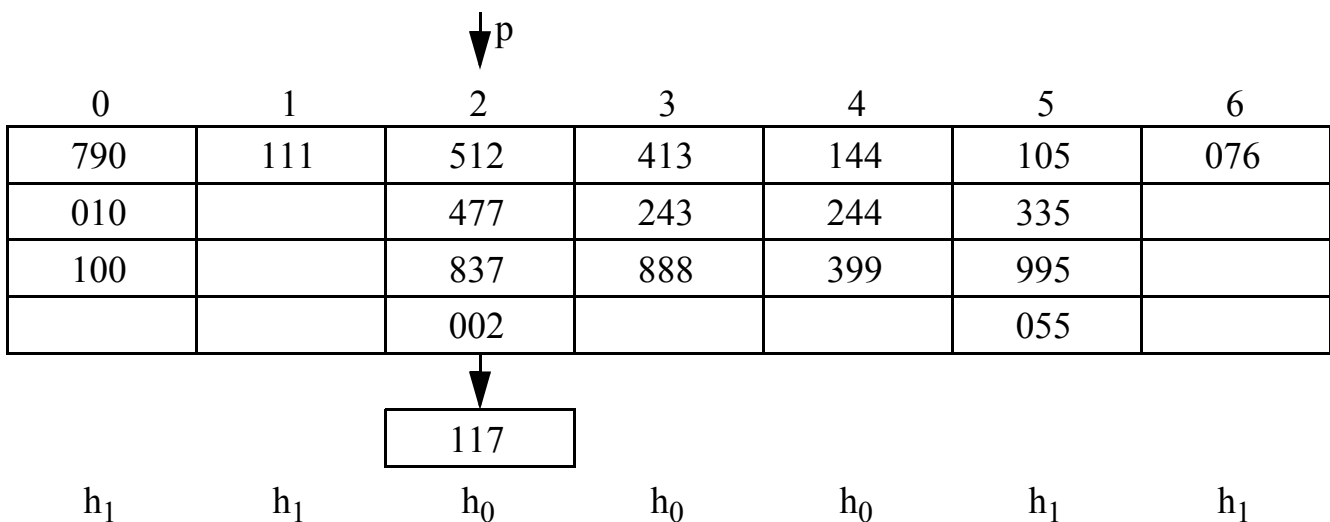


## Lineares Hashing (3)

- Einfügen von 888 erhöht Belegung auf  $\beta = 17/20 = 0.85$  und löst Splitting aus.



- Einfügen von 244, 399 und 100. Die letzte Einfügung erhöht die Belegung auf  $\beta = 20/24 = 0.83$  und löst Splitting aus:





## Lineares Hashing (4)

### • Splitting

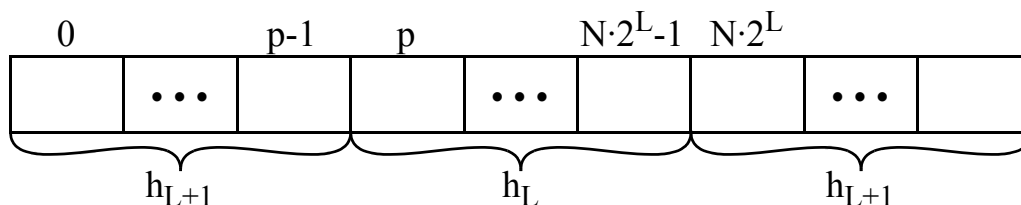
- Auslöser:  $\beta$
- Position:  $p$
- Datei wird um 1 vergrößert
- $p$  wird um 1 erhöht:  $p := (p + 1) \bmod (N \cdot 2^L)$
- Wenn  $p$  wieder auf Null gesetzt wird (Verdopplung der Datei beendet), wird  $L$  um 1 erhöht

### • Adreßberechnung

- Wenn  $h_0(k) \geq p$ , dann ist  $h_0$  die gewünschte Adresse
- Wenn  $h_0(k) < p$ , dann war das Bucket bereits gesplittet.  $h_1(k)$  liefert die gewünschte Adresse
- allgemein:

$$h := h_L(k);$$

$$\text{if } h < p \text{ then } h := h_{L+1}(k);$$



### • Split-Strategien

#### - Unkontrolliertes Splitting

- Splitting, sobald ein Satz in den Überlaufbereich kommt
- $\beta \sim 0.6$ , schnelleres Aufsuchen

#### - Kontrolliertes Splitting

- Splitting, wenn ein Satz in den Überlaufbereich kommt und  $\beta > \beta_s$
- $\beta \sim \beta_s$ , längere Überlaufketten möglich

## Vergleich der wichtigsten Zugriffsverfahren

Zugriffsverfahren	Speicherungsstruktur	Direkter Zugriff	Sequentielle Verarbeitung	Änderungsdienst (Ändern ohne Aufsuchen)
Fortlaufender Schlüsselvergleich	Sequentielle Liste	$O(n) \approx 10^4$	$O(n) \approx 2 \cdot 10^4$	$O(1) \leq 2$
	Gekettete Liste	$O(n) \approx 5 \cdot 10^5$	$O(n) \approx 10^6$	$O(1) \leq 3$
Baumstrukturierter Schlüsselvergleich	Balancierte Binärbäume	$O(\log_2 n) \approx 20$	$O(n) \approx 10^6$	$O(1) = 2$
	Mehrwegebäume	$O(\log_k n) \approx 3 - 4$	$O(n) \approx 10^{6*}$	$O(1) = 2$
Konstante Schlüsseltransformationsverfahren	Externes Hashing mit separatem Überlaufbereich	$O(1) \approx 1.1 - 1.4$	$O(n \log_2 n)^{**}$	$O(1) \approx 1.1$
	Externes Hashing mit Separatoren	$O(1) = 1$	$O(n \log_2 n)^{**}$	$O(1) = 1 (+D)$
Variable Schlüsseltransformationsverfahren	Erweiterbares Hashing	$O(1) = 2$	$O(n \log_2 n)^{**}$	$O(1) \approx 1.1 (+R)$
	Lineares Hashing	$O(1) = 1$	$O(n \log_2 n)^{**}$	$O(1) < 2$

\* Bei Clusterbildung bis zu Faktor 50 geringer

\*\* Physisch sequentielles Lesen, Sortieren und sequentielles Verarbeiten der gesamten Sätze  
Beispielangaben für  $n = 10^6$