# Externe Hashverfahren



- Zur Organisation von **sehr großen Datenmengen** reicht der Hauptspeicher nicht mehr aus
  → Organisation der Datensätze auf dem Sekundärspeicher
  → externe Verfahren notwendig.
- Wesentliche Eigenschaft: Bucketgröße $b > 1$

**Ziel:**

- Anzahl der Sekundärspeicherzugriffe (Zugriffe auf Buckets) gering halten → gute Hashfunktion sehr wichtig!

Unterscheidung in Anhängigkeit von der Kollisionsbehandlung:

- Verfahren mit Directory (= geschlossene Hashverfahren) und
- Verfahren ohne Directory (= offene Hashverfahren)

(hier nur Beispielverfahren ohne Directory!)

# Externe Hashverfahren: Lineares Hashing I



## Das Lineare Hashing (Litwin, 1980)

- **Grundlegende Eigenschaften:**
  - Größe der Datei (Hashtabelle) ist abhängig von der Anzahl der Schlüssel (**dynamisch**)
  - Überlaufsätze einzelner Buckets
    → getrennte Verkettung in einem Überlaufbereich
- **Überlaufstrategie der getrennten Verkettung**:
  → zwei verschiedene Seitentypen:
  - **Primärseiten → Primärdatei**
    → Datenseiten (Buckets), deren Adressen von der Hashfunktion berechnet werden
  - **Überlaufseiten → Überlaufdatei (Sekundärdatei)**
    → speichern die Überlaufsätze der Primärdatei

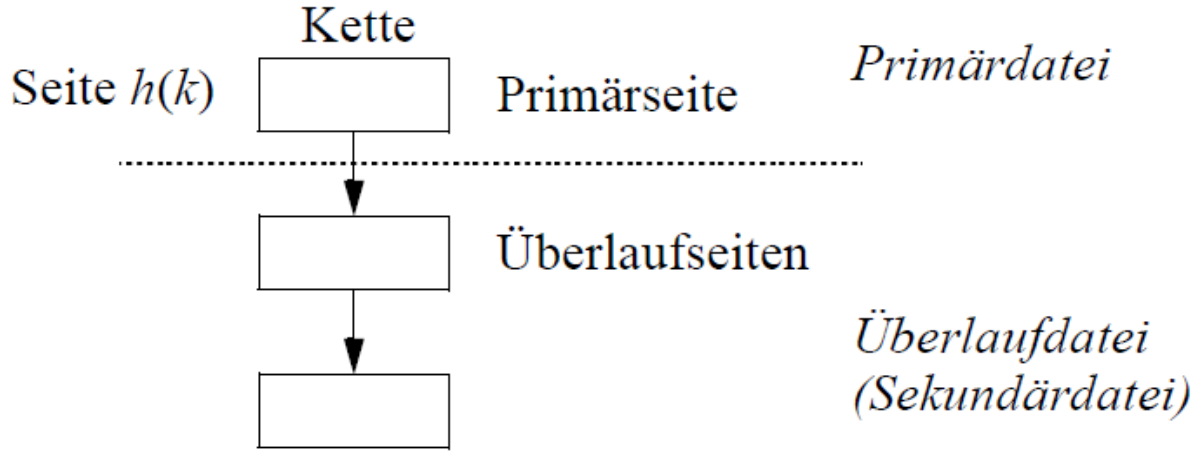
# Externe Hashverfahren: Lineares Hashing II

Es gilt:

(i) Überlaufseiten und Primärseiten sind verschieden

(ii) Jede Überlaufseite ist genau einer Hashadresse (einem Bucket) zugeordnet

(iii) Überlaufseiten derselben Primärseite werden verkettet




→ Suche nach einem Überlaufsatz benötigt mind. 2 Seitenzugriffe (Primärseite und 1 Überlaufseite)
Lange Ketten von Überlaufseiten → Suchzeiten degenerieren

# Externe Hashverfahren: Lineares Hashing III

**Bis ca. 1978:**

- Größe der Primärdatei (Hashtabelle) ist statisch vorgegeben
- Überläufe können nur durch Einführung von Überlaufsätzen gelöst werden

→ schnelle Degenerierung der Suchzeiten, wenn die Primärseiten voll werden

**Jetzt:** Dynamische Erweiterung der Primärdatei:

- Wenn Primärseiten zu voll werden → Erweiterung der Datei
- Neue Seiten werden über eine neue Hashfunktion adressiert

→ Man benötigt eine Folge von Hashfunktionen $h_0, h_1, h_2, \ldots$

# Externe Hashverfahren: Lineares Hashing IV



## Das Verfahren:

- Zu Beginn → kleine Primärdatei (z.B. $m = 5$); Hashfunktion: $h_0: K \rightarrow \{0,1,2,3,4\}$.
  - Die ersten $b$ Schlüssel → sicher in Primärseiten.
  - Zunehmende Schlüsselanzahl → Wahrscheinlichkeit für Überlaufsätze steigt
- **Idee:** Vergrößerung (**Expansion**) der Primärdatei um jeweils ein Bucket, falls eine vorgegebene Auslastung der Primärseiten erreicht ist (**Kontrollfunktion**)
- Definition: **Belegungsfaktor**:

$$bf = \frac{Anzahl\ der\ Schlüssel}{\begin{matrix}Anzahl\ Schlüssel, die\ in \\ Primärseiten\ passen\end{matrix}} = \frac{Anzahl\ der\ Schlüssel}{b \cdot Anzahl\ der\ Primärseiten}$$

Es wird vorgeschlagen: Expansion der Datei, falls $bf > 0{,}8$.

# Externe Hashverfahren: Lineares Hashing V



## Beispiel:

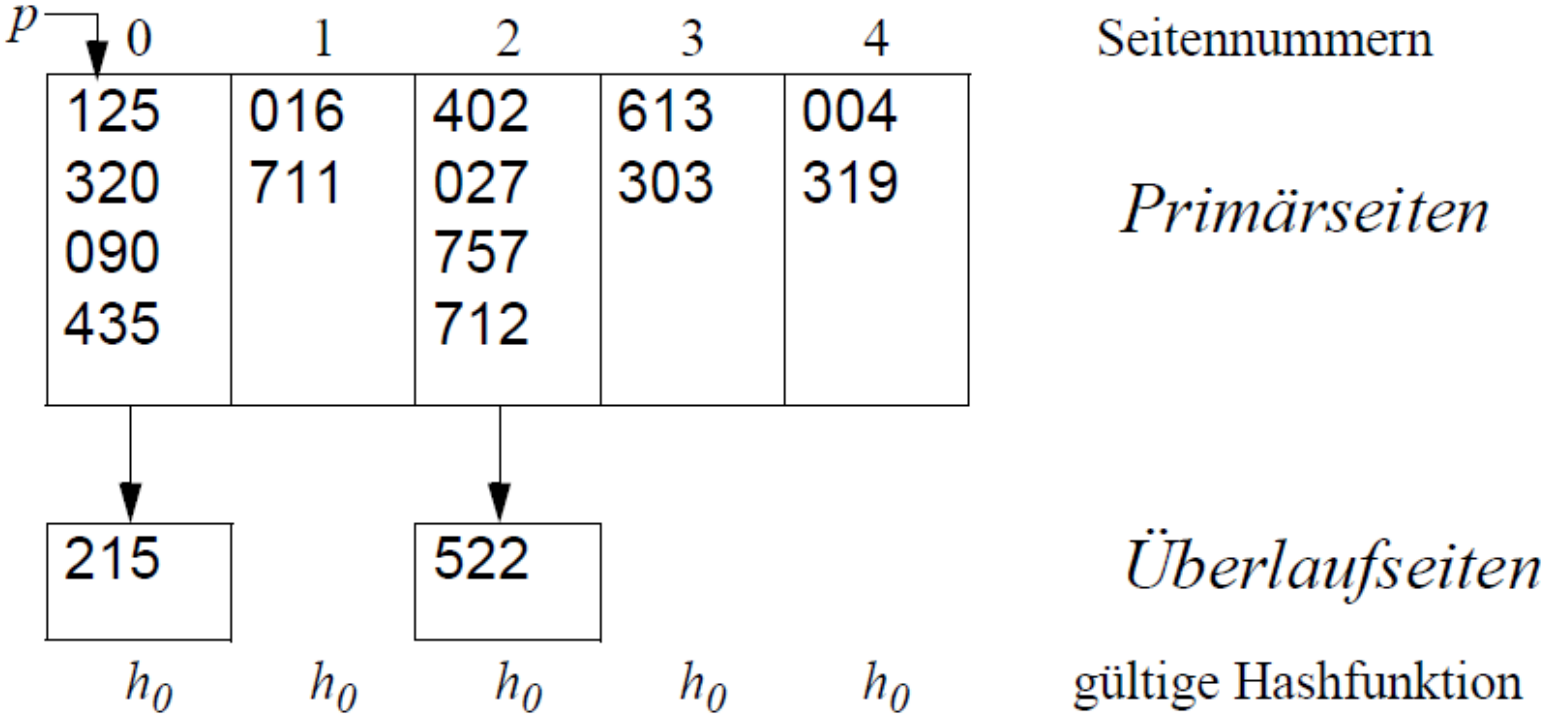
## Ausgangssituation:

- Datei mit 16 Datensätzen auf 5 Primärseiten (Buckets) der Seitengröße $b = 4$.
- Folge von Hashfunktionen:
  $h_0(k) = k\ MOD\ 5, \qquad \mathrm{h}_1(k) = k\ MOD\ 10, \qquad \ldots$
- Aktuelle Hashfunktion: $h_0$.
- Belegungsfaktor $bf = \frac{16}{20} = 0{,}8$
- Schwellenwert für den Belegungsfaktor $= 0{,}8$
- Expansionszeiger $p$ zeigt auf Seite $0$.

# Externe Hashverfahren: Lineares Hashing VI

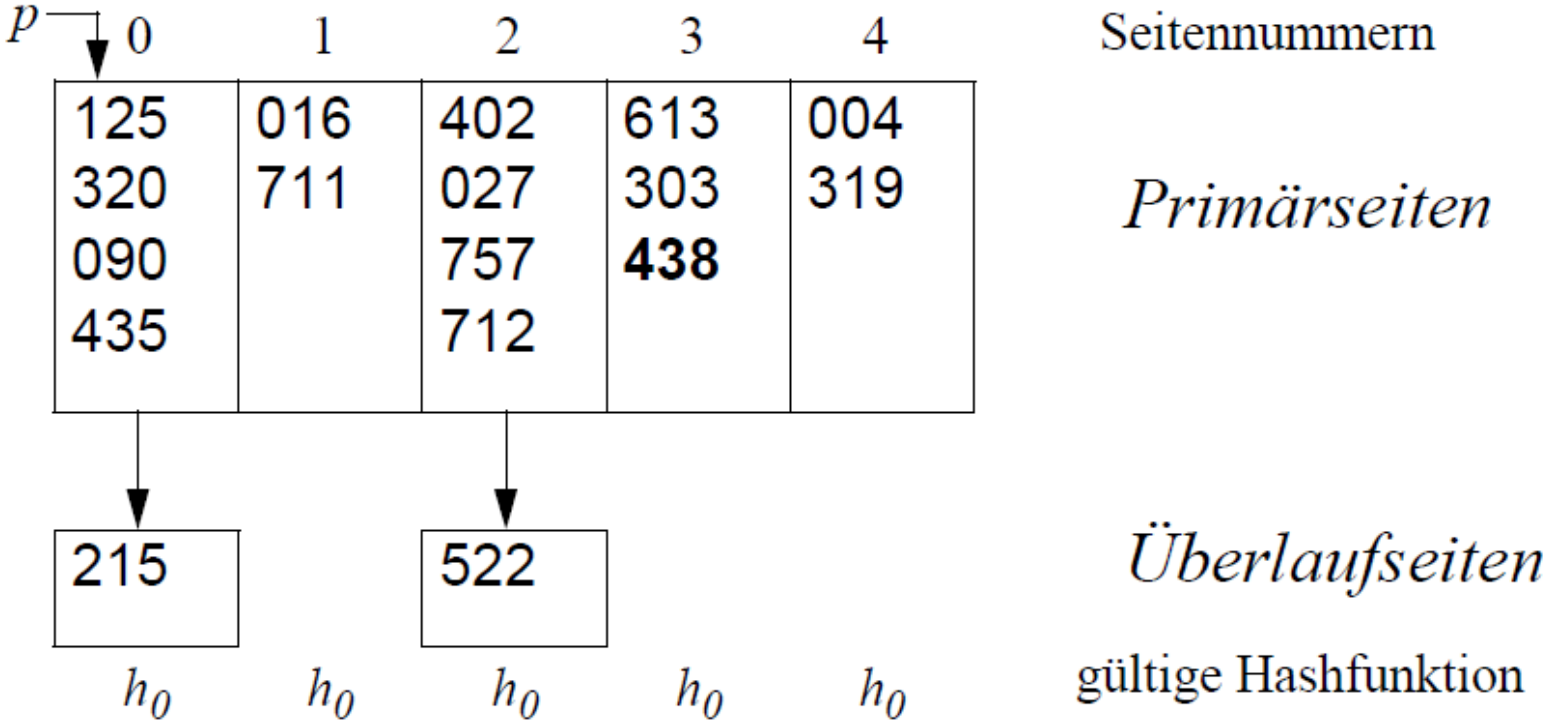


## Ausgangssituation:



# Externe Hashverfahren: Lineares Hashing VII

Einfügen eines neuen Datensatzes mit Schlüssel 438:

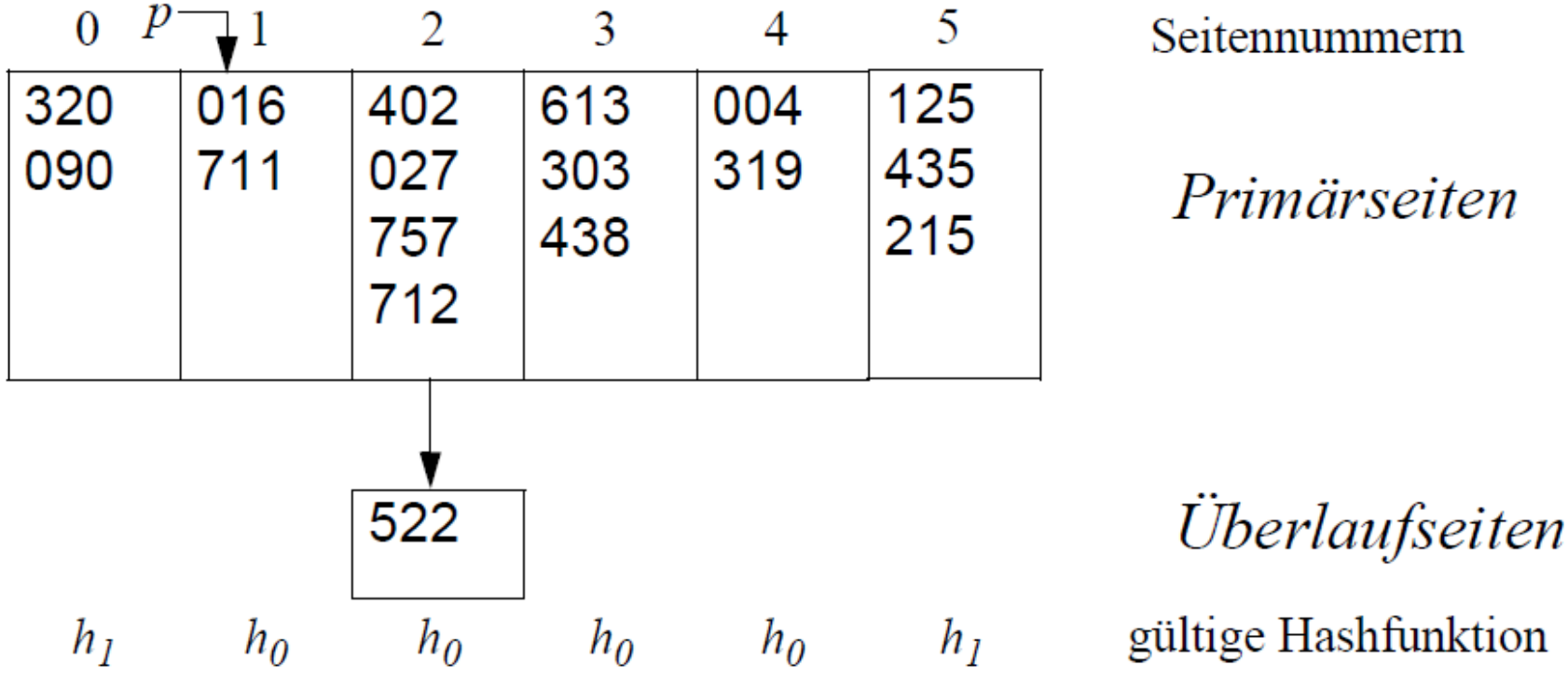


# Externe Hashverfahren: Lineares Hashing VIII

Der Belegungsfaktor übersteigt den Schwellenwert:

$$bf = \frac{17}{20} = 0{,}86 > 0{,}8$$

→ Expansion durch Split der Seite 0 auf die Seite 0 und 5.



# Externe Hashverfahren: Lineares Hashing IX



Welche Datensätze von Seite 0 nach Seite 5 umgespeichert werden, bestimmt die Hashfunktion $h_1$:

→ Sätze mit $h_1(k) = 5$ werden umgespeichert
→ Sätze mit $h_1(k) = 0$ bleiben

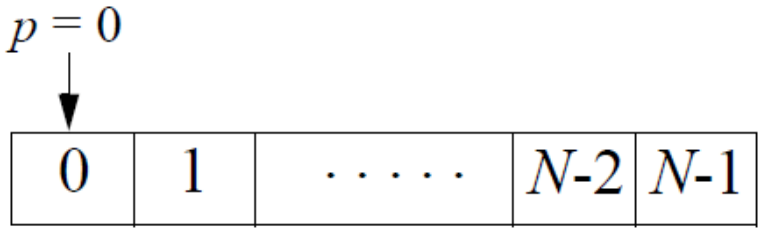
Anschließend wird der Expansionszeiger $p$ um 1 heraufgesetzt.

# Externe Hashverfahren: Lineares Hashing X

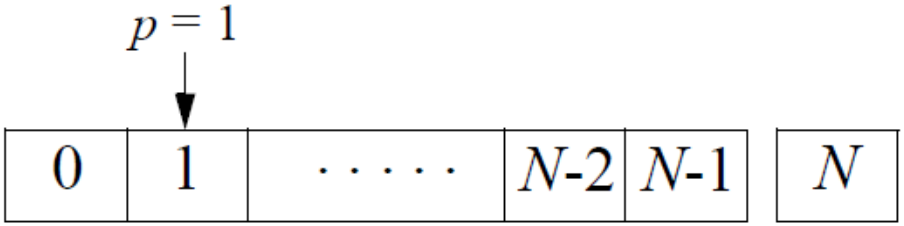


## Prinzip der Expansion durch Splits:

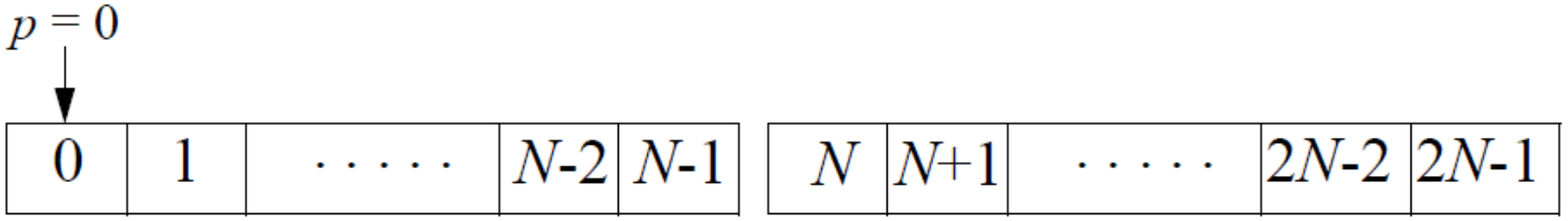
Ausgangssituation:




Nach dem ersten Split:




Nach der Verdoppelung der Datei:



# Externe Hashverfahren: Lineares Hashing XI



## Anforderungen an die Hashfunktionen:

- Die Folge von Hashfunktionen $\{h_i\}, i \geq 0$, muss die folgenden Bedingungen erfüllen:

**Bereichsbedingung:**

$$h_L : K \rightarrow \{0,1, \ldots, N \cdot 2^L - 1\}, L \geq 0$$

**Splitbedingung:**

$$h_{L+1}(k) = h_L(k) \quad \text{oder} \quad h_{L+1}(k) = h_L(k) + N \cdot 2^L, \; L \geq 0$$

- Der **Level $L$** gibt dabei an, wie oft sich die Datei bereits vollständig verdoppelt hat.

# Externe Hashverfahren: Lineares Hashing XII



**Beispiel** für eine mögliche Folge von Hashfunktionen:

$$h_L(k) = k\ MOD\ (N \cdot 2^L)$$

(erfüllt die Bereichs- und die Splitbedingung)

**Zusätzlich erwünschte Eigenschaft:**

Für $p = 0$ (nach einer Verdoppelung) besitzen alle Buckets die gleiche Wahrscheinlichkeit, einen neu eingefügten Datensatz aufzunehmen
(**Ideale Hashfunktion**)

# Externe Hashverfahren: Lineares Hashing XIII



## Wichtige Eigenschaften des linearen Hashing:

- Splitreihenfolge der Seiten ist fest vorgeschrieben
  → Überlaufende Seiten werden erst mit einer gewissen Verzögerung behandelt.
- Der Adressraum wächst **linear** an und ist gerade so groß wie notwendig
- Unter der Voraussetzung einer guten Hashfunktion ist der Prozentsatz der Überlaufsätze gering
- Gute Suchzeiten für gleichverteilte Schlüssel

# Externe Hashverfahren: Partielle Expansionen I



## Lineares Hashing mit partieller Erweiterung (Larson, 1980)

**Beobachtung:**

Hashverfahren sind am effizientesten, wenn die Schlüssel möglichst gleichmäßig auf die Seiten der Datei verteilt sind.

**Problem:**

Während jeder Verdoppelung der Datei gilt:
Der durchschnittliche Belegungsfaktor bereits gesplitteter Seiten ist nur halb so hoch wie der durchschnittliche Belegungsfaktor noch nicht gesplitteter Seiten.

**Verbesserung:** Einführung **partieller Expansionen**:

→ Verdoppelung der Datei erfolgt in mehreren Durchläufen

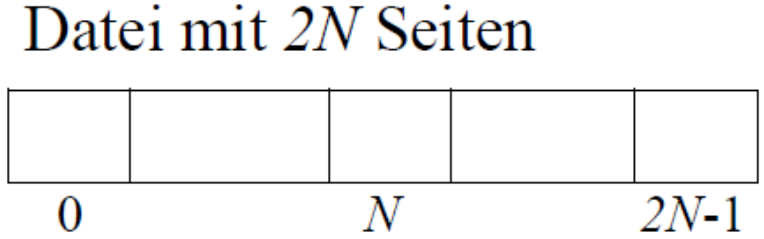
→ Serie von $n_0 > 2$ partiellen Expansionen

# Externe Hashverfahren: Partielle Expansionen II



## Vorgehensweise des linearen Hashing mit partiellen Erweiterungen (für $n_0 = 2$)

### Ausgangssituation:

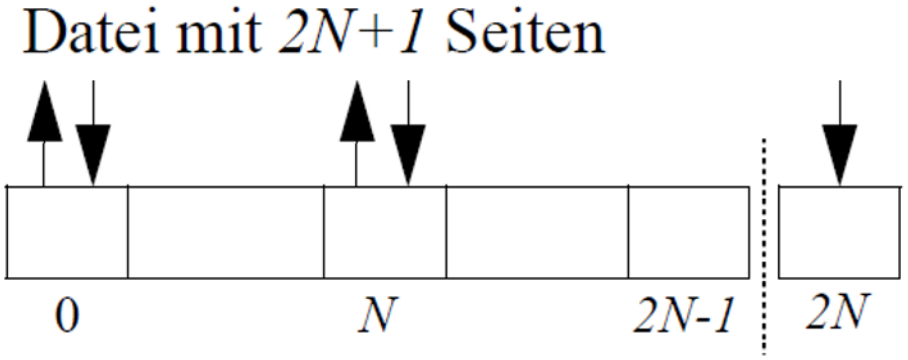



Logisch unterteilt in $N$ Paare $(j, j+N)$ für $j = 0,1,\ldots,N-1$

# Externe Hashverfahren: Partielle Expansionen III

## 1. Partielle Expansion:

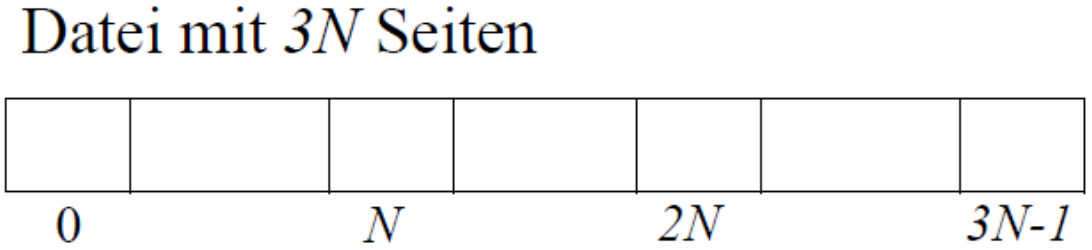
- Nach Einfügen von Schlüsseln wird aufgrund der Kontrollfunktion eine Expansion nötig → Expansion der Datei um die Seite $2N$.
- Der Inhalt der 2 (alten) Seiten wird auf die nun 3 Seiten verteilt.
- Nach der Umverteilung des Inhalts von den Seiten 0 und $N$ ist die Seite $2N$ zu etwa $\frac{2}{3}$ gefüllt.




Logisch unterteilt in die verbleibenden $N-1$ Paare $(j, j+N)$ für $j = 1, \ldots, N-1$ sowie das Tripel $(0, N, 2N)$.

# Externe Hashverfahren: Partielle Expansionen IV

- Bei weiteren Einfügungen wird, wenn es die Kontrollfunktion verlangt, für $j = 1,2, \ldots, N-1$ jeweils das Paar $(j, j+N)$ um die Seite $j+2N$ expandiert.
  → nach $N$ Schritten: die Datei ist von $2N$ auf $3N$ Seiten (auf das 1,5-fache) angewachsen:



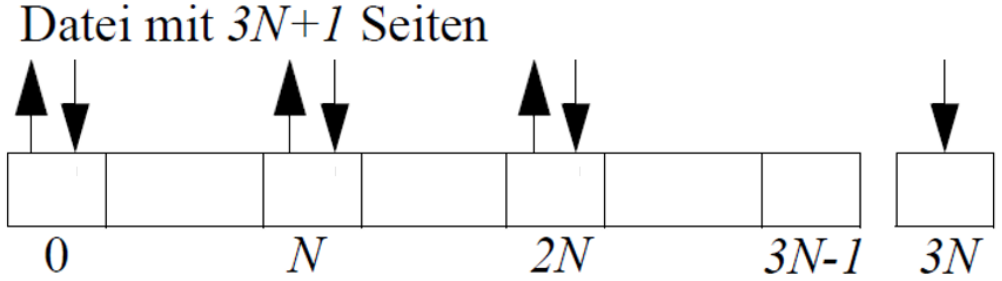

Logisch unterteilt in $N$ Tripel $(j, j+N, j+2N)$ für $j = 0,1, \ldots, N-1$.

# Externe Hashverfahren: Partielle Expansionen V



## 2. Partielle Expansion:

- Zunächst Expansion der Datei um die Seite $3N$.
- Der Inhalt der 3 (alten) Seiten wird auf die nun 4 Seiten verteilt.
- Nach der Umverteilung des Inhalts von den Seiten 0 und $N$ ist die Seite $3N$ zu etwa $\frac{3}{4}$ gefüllt.




Logisch unterteilt $N-1$ Tripel $(j, j+N, j+2N)$ für $j = 1,2,\ldots,N-1$ und ein Quadrupel $(0, N, 2N, 3N)$.

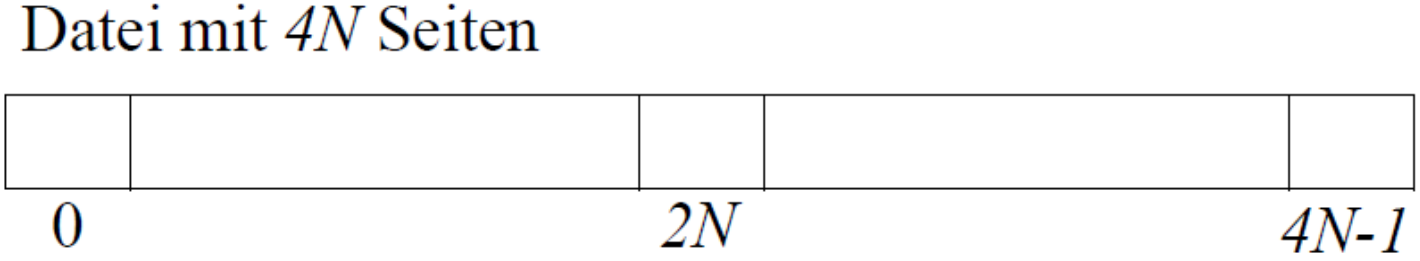
- Nachfolgend werden für $j = 1,2,\ldots,N-1$ die Tripel $(j, j+N, j+2N)$ jeweils um die Seite $j+3N$ expandiert, wobei ca. $\frac{1}{4}$ der Sätze aus den Seiten $j, j+N, j+2N$ umgespeichert werden.

# Externe Hashverfahren: Partielle Expansionen VI



**Resultat:**

- Die Datei hat sich auf $4N$ Seiten verdoppelt:




→ dies ist die neue Ausgangssituation für die nächste Verdoppelung der Datei.

# Externe Hashverfahren: Partielle Expansionen VII



## Analyse der partiellen Expansionen:

Es gilt während der 1. bzw. 2. partiellen Expansion:
Der durchschnittliche Belegungsfaktor einer bereits gesplitteten Seite beträgt $\frac{2}{3}$ bzw. $\frac{3}{4}$ des durchschnittlichen Belegungsfaktors einer noch nicht gesplitteten Seite.

*(verglichen mit dem Faktor $\frac{1}{2}$ ohne partielle Expansionen ($n_0 = 1$))*

## Kontrollfunktion:

Larson schlägt vor, die Speicherplatzausnutzung zu wählen:

$$Speicherplatzausnutzung = \frac{Anzahl\ der\ Schlüssel}{Gesamtkapazität\ von\ Primär - und\ Überlaufseiten}$$

# Externe Hashverfahren: Partielle Expansionen VIII



**Expansionsregel:**

- Bei jeder Einfügung wird die Speicherplatzausnutzung überprüft
- Falls ihr Wert größer als ein vorgegebener Schwellenwert $\alpha, 0 < \alpha < 1$ ist, wird die Datei um eine weitere Seite expandiert.

→ Die Speicherplatzausnutzung liegt annähernd konstant bei $\alpha$, falls nur Einfügungen auftreten.

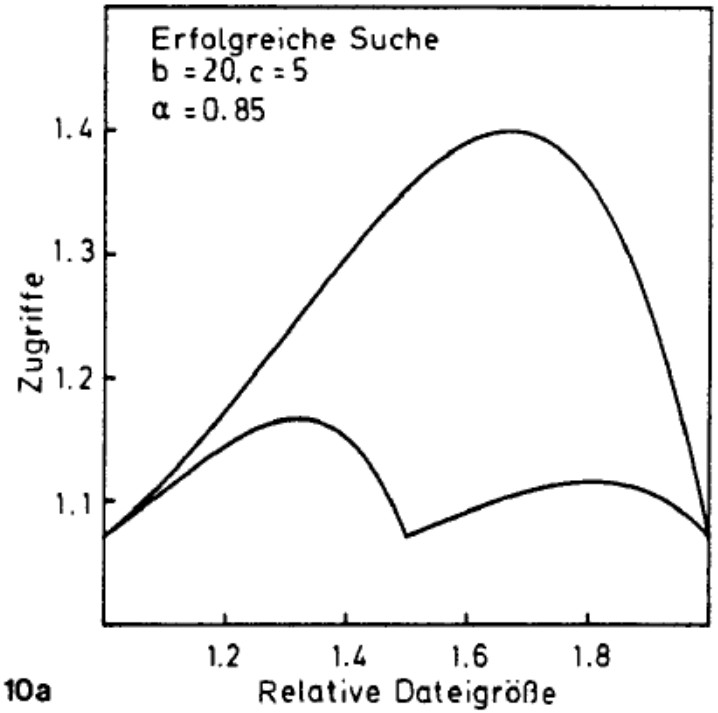
# Externe Hashverfahren: Partielle Expansionen IX



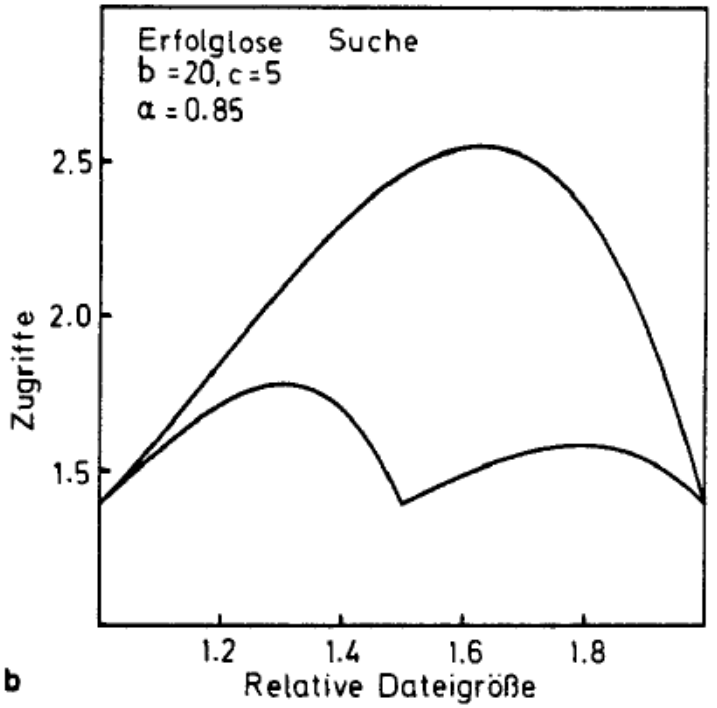
## Leistungsverhalten:

$b$ = Kapazität einer Primärseite, $c$ = Kapazität einer Überlaufseite.

Anzahl benötigter Seitenzugriffe bei erfolgreicher, bzw. bei erfolgloser Suche für $n_0 = 1$ und $n_0 = 2$:




10a




b

# Externe Hashverfahren: Partielle Expansionen X



## Beobachtungen:

- Effizienz ist dann am größten, wenn die Belegung aller Ketten nahezu gleich ist
- Zyklisches Verhalten → Länge eines Zyklus = 2 · Länge des vorangegangenen Zyklus

## Vergleich:

Durchschnittliche Anzahl benötigter Zugriffe mit den zugrundeliegenden Parametern $b = 20$, $c = 5$ und $\alpha = 0{,}85$:

| | $n_0 = 1$ | $n_0 = 2$ | $n_0 = 3$ |
|---|---|---|---|
| Erfolgreiche Suche | 1,27 | 1,12 | 1,09 |
| Erfolglose Suche | 2,12 | 1,58 | 1,48 |
| Einfügen | 3,57 | 3,21 | 3,31 |
| Entfernen | 4,04 | 3,53 | 3,56 |

# Externe Hashverfahren: Partielle Expansionen XI



**Folgerung:**

$n_0 = 2$ partielle Expansionen stellen den besten Kompromiss dar zwischen

- Effizienz des Suchens,
- Effizienz der Update-Operationen (Einfügen und Entfernen)
- Komplexität des Programmcodes