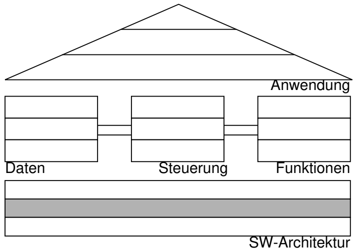
2. E/A-Architektur und Zugriff

- **GBIS-Rahmen: Einordnung**



- **E/A-Architektur von Informationssystemen**
- **Einsatz einer Speicherhierarchie**
  - Aufbauprinzip
  - Verarbeitungseigenschaften
  - Prognosen
- **Datenstrukturen auf Externspeichern**
  - sequentielle und gekettete Listen
  - Mehrwegbäume
- **B-Bäume und B*-Bäume**
  - Definitionen, Grundoperationen, Kosten
  - Überlaufbehandlung
  - Schlüsselkomprimierung
- **Informationssuche bei**
  - strukturierten und unstrukturierten Daten
  - semi-strukturierten Daten
- **Woher kommt die schlechte Suchqualität?**
  - Probleme der Indexierung
  - Anfrageauswertung: Precision und Recall
- **Dokumentenzugriff im Web**
  - Proxy-Cache-Server, Prinzip der Dokumentenzugriffs
  - Web-Caching und Web-Replikation

E/A-Architektur von Informationssystemen

- **Bisherige Annahmen**
  - Datenstrukturen wie Felder, Listen, Bäume, Graphen, ...
    - lokale Speicherung und direkter Zugriff auf alle Elemente
    - hohe Zugriffsgeschwindigkeit
  - ausschließlich im **Hauptspeicher** verfügbar,
    d. h. Bestand nur für die Dauer einer Programmausführung
    („transiente“ Daten)

- **hier nun neue Aspekte:**
  Nutzung von Externspeichern und neuen Operationen
  - Datenstrukturen werden gespeichert und verwaltet
    - verteilt und dezentral
    - auf verschiedenen Speichertypen
      (Magnetplatte, Magnetband, optische Speicher, ...)
    - in verschiedenen Rechensystemen
    - im Web
  - andere Arten des Zugriffs: Lese- und Schreiboperationen,
    in vorgegebenen Einheiten von Blöcken und Sätzen

    ➥ Strukturen und zugehörige Algorithmen (Suchen, Sortieren),
    die im Hauptspeicher optimal sind, sind es auf Sekundärspeicher
    nicht unbedingt!
  - gezielter, wertabhängiger Zugriff auch auf sehr große Datenmengen
  - umfangreiche Attributwerte (z. B. Bilder, Texte)
  - **Persistenz:**
    Werte bleiben über Programmende, Sitzungsende, Aus- und Einschalten
    des Rechners, . . . hinaus erhalten

E/A-Architektur von Informationssystemen (2)

- **Problemstellung**
  - langfristige Speicherung und Organisation der Daten im Hauptspeicher?
    - Speicher ist (noch) flüchtig! (--> Persistenz ?)
    - Speicher ist (noch) zu teuer und zu klein (--> Kosteneffektivität)
  - mindestens **zweistufige Organisation** der Daten erforderlich
    - langfristige Speicherung auf großen, billigen und nicht-flüchtigen Speichern (Externspeicher)
    - Verarbeitung erfordert Transport zum/vom Hauptspeicher
    - vorgegebenes Transportgranulat: Seite

  ➥ Wie sieht die bestmögliche Lösung aus?

- **Idealer Speicher besitzt**
  - nahezu unbegrenzte Speicherkapazität
  - kurze Zugriffszeit bei wahlfreiem Zugriff
  - hohe Zugriffsraten
  - geringe Speicherkosten
  - Nichtflüchtigkeit
  - Fähigkeit zu logischen, arithmetischen u. ä. Verknüpfungen?

  ➥ Was würde das für einen Externspeicher bedeuten?

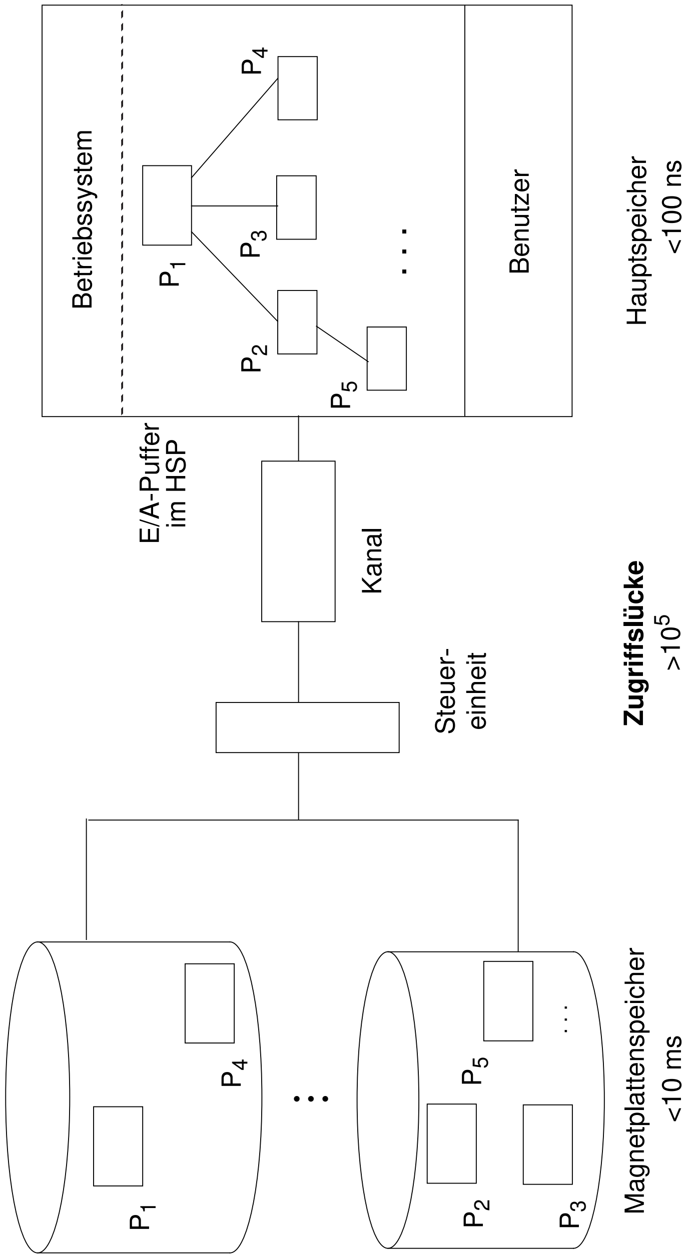
- **Realer Speicher sollte diese Eigenschaften näherungsweise bieten!**
  - Einsatz verschiedener Speichertypen (mit großen Unterschieden bei Kapazität, Zugriffszeit, Bandbreite, Preis, Flüchtigkeit, ...)
  - Organisation als Speicherhierarchie

  ➥ E/A-Architektur zielt auf kosteneffektive Lösung ab

Zugriffswege bei einer zweistufigen Speicherhierarchie

Vereinfachte E/A-Architektur :

- Modell für Externspeicheranbindung
- vor Bearbeitung sind die Seiten in den HSP-Puffer zu kopieren
- geänderte Seiten müssen zurückgeschrieben werden



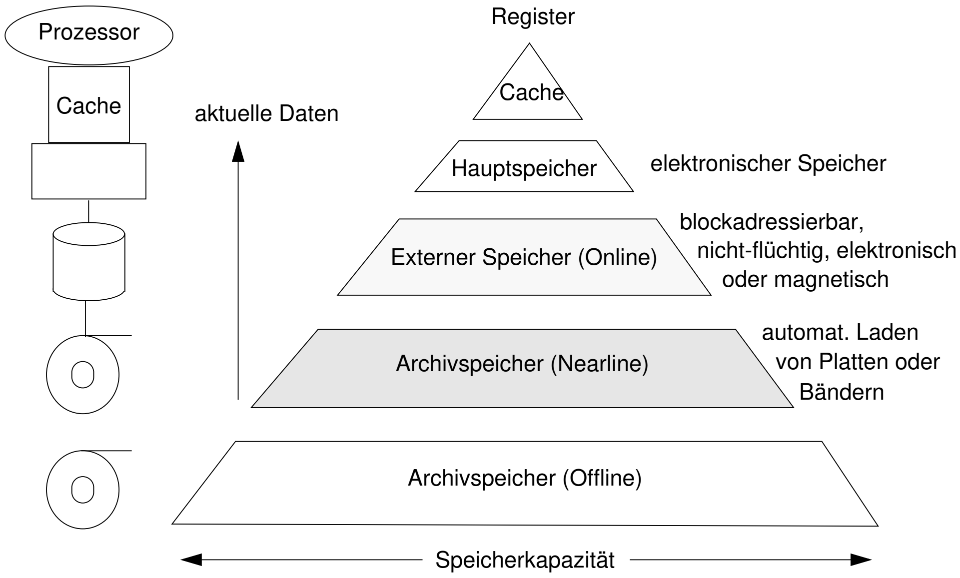
Speicherhierarchie

- **Wie kann die Zugriffslücke überbrückt werden?**
  - Neue Speichertypen, die in die Lücke passen?
    - Blasenspeicher (*magnetic bubbles*), CCD usw. sind untauglich (~1980)
    - teilweise: erweiterte HSP, SSD (*solid state disks*) und Platten-Caches
  - Nutzung von Lokalitätseigenschaften
    - Allokation von Daten mit hoher Zugriffswahrscheinlichkeit in schnelle (relativ kleine) Speicher
    - Mehrheit der Daten verbleibt auf langsameren, kostengünstigeren Speichern

➥ Speicherhierarchie versucht Annäherung an idealen Speicher durch reale Speichermedien zu erreichen

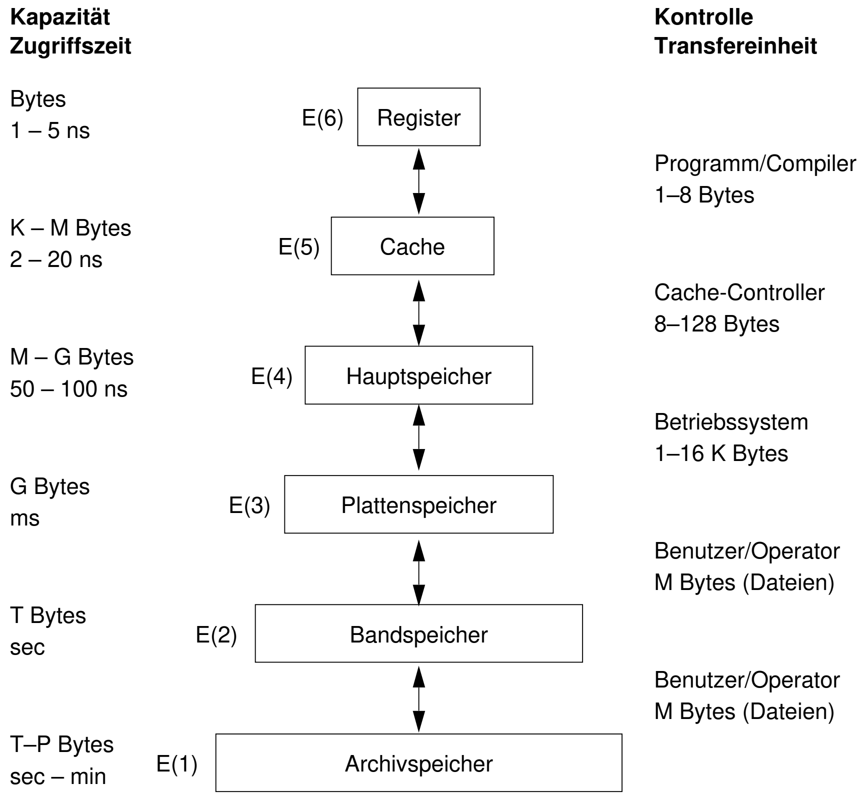
- **Rekursives Prinzip**

Kleinere, schnellere und teurere Cache-Speicher werden benutzt, um Daten zwischenzuspeichern, die sich in größeren, langsameren und billigeren Speichern befinden



Speicherhierarchie (2)

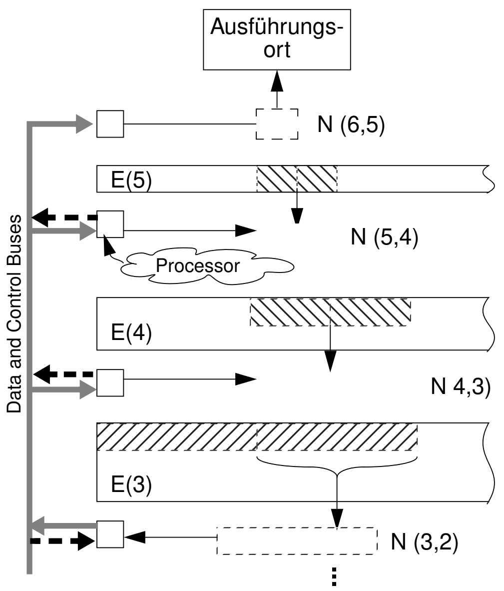
- **Charakteristische Merkmale einer Speicherhierarchie**

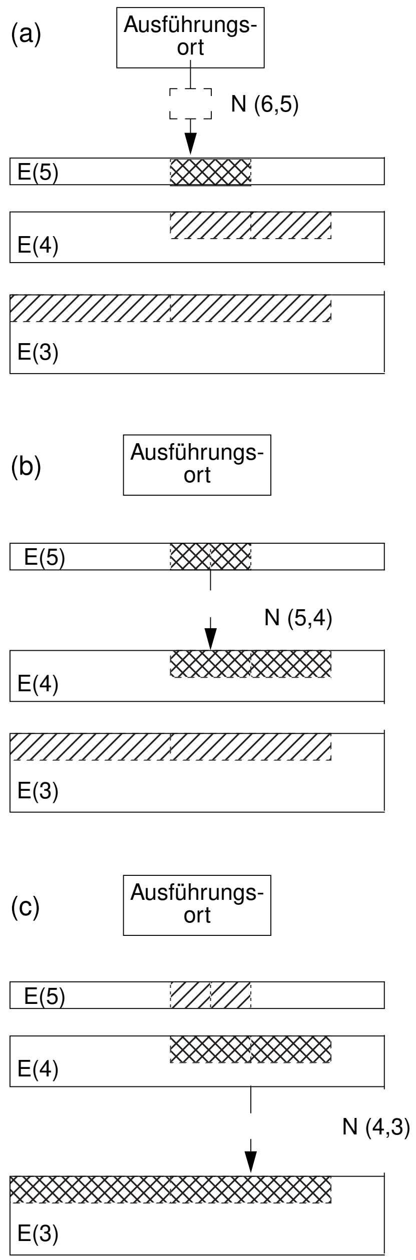


In naher Zukunft sind an verschiedenen Stellen der Speicherhierarchie wesentliche Verbesserungen bei Zugriffszeit und Transfergröße zu erwarten, welche ihre Arbeitsweise zwischen E(3) und E(6) stark beeinflussen könnte. IRAM (*Intelligent Random Access Memory*) zielt darauf ab, Mikroprozessor und DRAM-Speicher (*Dynamic Random Access Memory*) auf einem Chip anzusiedeln, um so weniger Controller zu benötigen und dichtere Integration mit geringerer Latenzzeit (Faktor 5–10) und größerer Bandbreite (Faktor 50–100) zu erreichen. IDISK *(Intelligent Disk*) erweitert Magnetplatten mit Verarbeitungslogik und Speicher in Form von IRAM, um bestimmte Zugriffs- und Sortieroperationen zu beschleunigen.

Speicherhierarchie (3)

- **Inklusionseigenschaft beim Lesen und Schreiben**





Read-through:
Auf Anforderung werden die Daten im entsprechenden Granulat stufenweise in der Hierarchie „nach oben“ bewegt

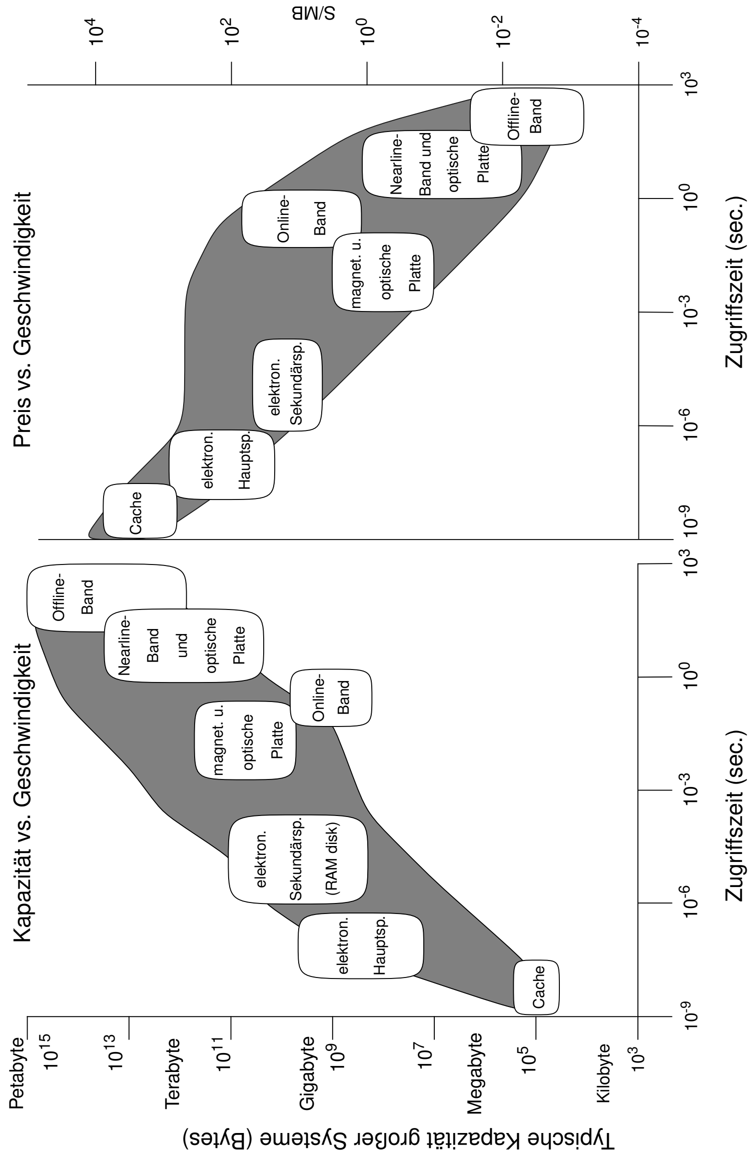
Write-through:
Sofort nach Änderung werden die Daten auch auf der nächsttieferen Stufe geändert (FORCE)

Write-back:
Bei Ersetzung werden die geänderten Daten stufenweise „nach unten“ geschoben (NOFORCE)

Speicherhierarchie (4)

- **Leistungscharakteristika und Kosten heutiger Speicher**

Schneller Speicher ist teuer und deshalb klein; Speicher hoher Kapazität ist typischerweise langsamer



Relative Zugriffszeiten in einer Speicherhierarchie

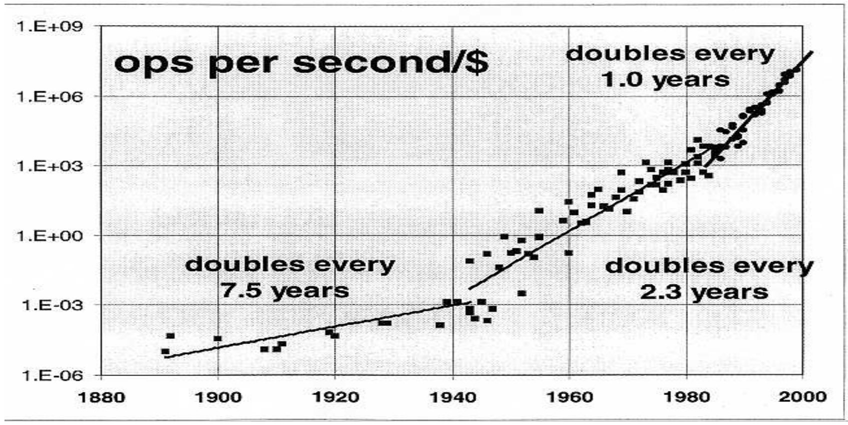
- **Wie weit sind die Daten entfernt?**

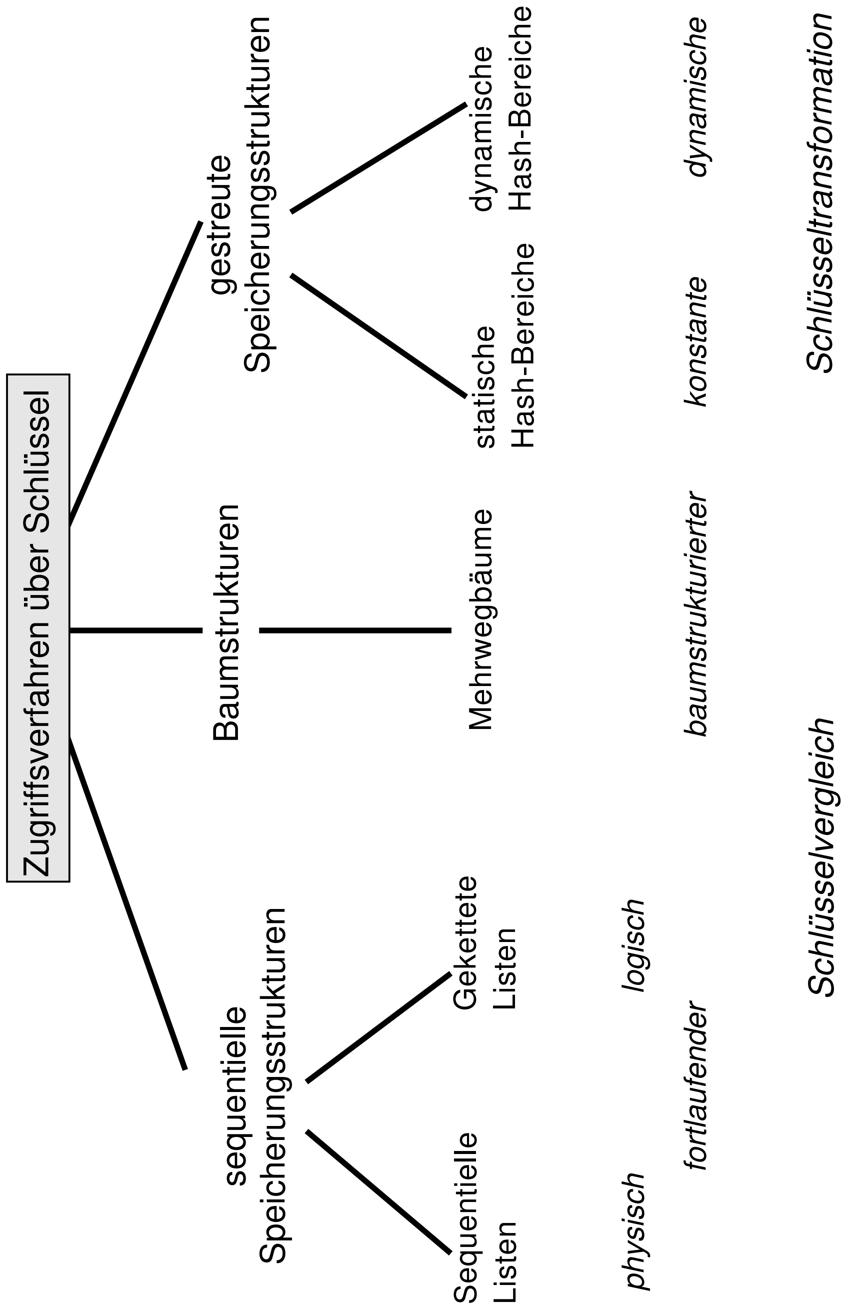
Speicherhierarchie		Was bedeutet das in „unseren Dimensionen“?	
Speichertyp	Rel. Zugriffszeit	Zugriffsort	Zugriffszeit
Register	1	mein Kopf	
Cache (on chip)	2	dieser Raum	
Cache (on board)	10	dieses Gebäude	
Hauptspeicher	100	Frankfurt (mit Auto)	
Magnetplatte	10^6–10^7	Pluto ($5910 * 10^6$ km)	
Magnetband/Opt. Speicher (automat. Laden)	10^9–10^{10}	Andromeda	

- **Welche Konsequenzen ergeben sich daraus?**
  - Caching
  - Replikation
  - Prefetching

Prognosen

- **Moore’s Gesetz** (interpretiert nach J. Gray)
  - Verhältnis „Leistung/Preis“ verdoppelt sich alle 18 Monate!
  - Faktor 100 pro Dekade
  - **Exponentielles Wachstum:**
    Fortschritt in den nächsten 18 Monaten = Gesamter Fortschritt bis heute
    - neuer Speicher = Summe des gesamten „alten“ Speichers
    - neue Verarbeitungsleistung = Summe der bisherigen Verarbeitungsleistung
  - Bakterium E. coli verdoppelt sich alle 20 Minuten!
- **Was kann das für den Einsatz von Speichern bedeuten?**
  - in 10 Jahren zum Preis von heute
    - Hauptspeicher
    - Externspeicher
  - **„The price of storage is the cost of managing the storage!”**
- **Drei Wachstumsphasen bei „ops per second/$”**
  - 1890-1945: mechanische Relais, 7 Jahre Verdopplungszeit
  - 1945-1985: Röhre, Transistor, 2.3 Jahre Verdopplungszeit
  - 1985-2010: Mikroprozessor,1.0 Jahre Verdopplungszeit?





Sequentielle Listen auf Externspeichern

- **Prinzip: Zusammenhang der Satzmenge wird durch physische Nachbarschaft hergestellt**
  - Reihenfolge der Sätze: ungeordnet (Einfügereihenfolge) oder sortiert nach einem oder mehreren Attributen (sog. Schlüssel)
  - **wichtige Eigenschaft:** Cluster-Bildung, d. h., physisch benachbarte Speicherung von logisch zusammengehörigen Sätzen
  - Ist physische Nachbarschaft aller Seiten der Liste sinnvoll?

  ➥ Sequentielle Listen garantieren Cluster-Bildung in Seiten für die Schlüssel-/Speicherungsreihenfolge

  ➥ pro Satztyp kann Cluster-Bildung nur bezüglich eines Kriteriums erfolgen, falls keine Redundanz eingeführt werden soll

- **Zugriffskosten** (N_S = Anzahl physischer Seitenzugriffe):
  - Sequentieller Zugriff auf alle Datensätze eines Typs: bei n Sätzen und mittlerem Blockungsfaktor b

    N_S =

  - **Sortierte Listen:**
    - Zugriff über eindeutigen Schlüssel kostet im Mittel

      N_S =

    - Binärsuche in Spezialfällen möglich (welche ?)
  - **Ungeordnete Listen:**
    Wieviele Seitenzugriffe benötigt der direkte Zugriff ?

    N_S =

Sequentielle Listen auf Externspeichern (2)

- **Änderungen sind sehr teuer:**
  Einfügen von K7 ?

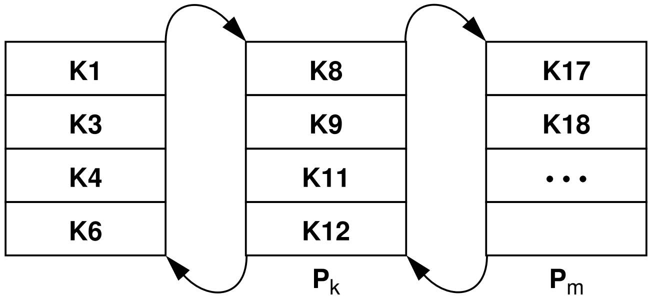
K1	**K8**	**K17**
K3	**K9**	**K18**
K4	**K11**	**• • •**
K6	**K12**	

Dominoeffekt !?

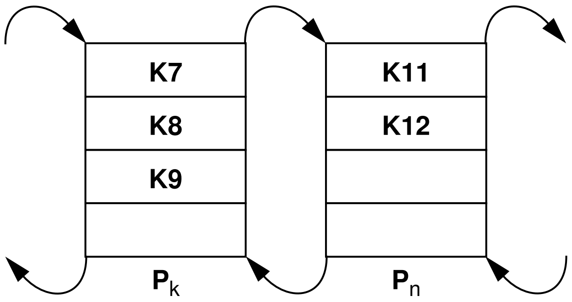
Welche Kosten verursacht der Änderungsdienst?

Verschiebekosten im Mittel: N_S =

- **Deshalb Einsatz einer Splitting-Technik:**



Welche Eigenschaft einer sequentiellen Liste geht verloren?

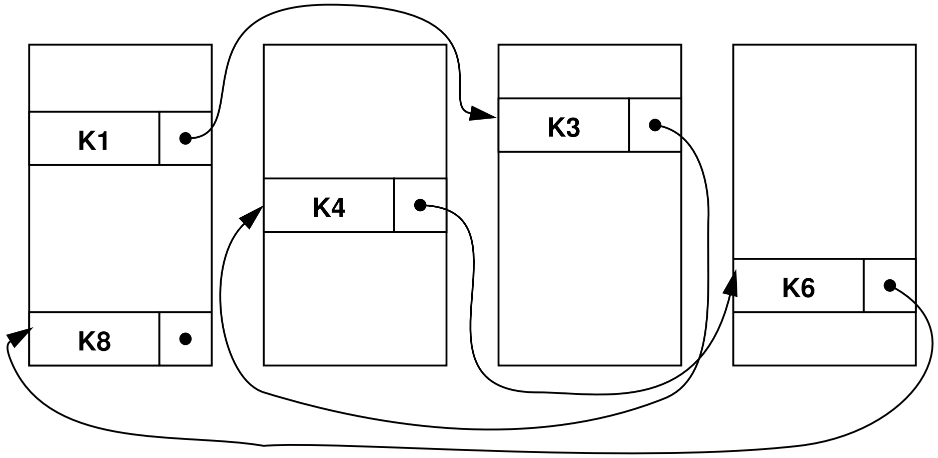


Speicherplatz-
belegung ?

➥ Änderungen auf max. 3 Seiten beschränkt, aber Suche der Position der durchzuführenden Änderung

Gekettete Listen auf Externspeichern

- **Prinzip: Verkettung erfolgt zwischen Datensätzen**
  - Speicherung der Sätze i. allg. in verschiedenen Seiten
  - Seiten können beliebig zugeordnet werden



➥ im allgemeinen keine Cluster-Bildung

- **Zugriffskosten**
  - sequentieller Zugriff auf alle Sätze:

    N_S =

  - direkte Suche (= fortlaufende Suche) bei Verkettung in Sortierreihenfolge:

    N_S =

  - Einfügen relativ billig, wenn Einfügeposition bekannt
    (≤ 2 Seitenänderungen)

- **Mehrfachverkettung nach verschiedenen Kriterien (Schlüsseln) möglich**

Mehrwegbäume

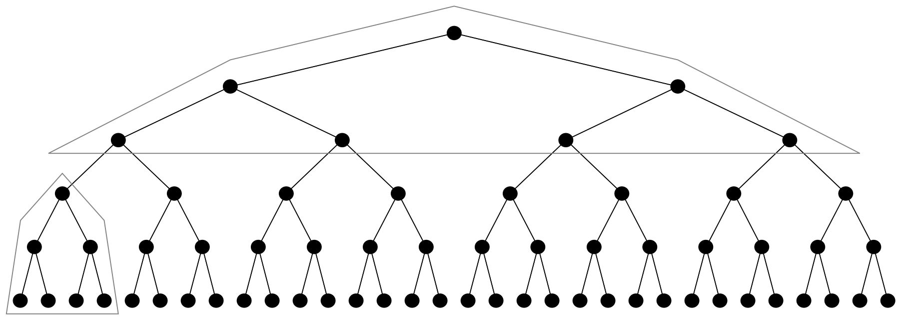
- **Binäre Suchbäume (SB) sind bekannt**
  - natürlicher binärer SB
  - AVL-Baum (höhenbalanciert)
  - gewichtsbalancierter SB
  - ➥ Bei Abbildung auf Externspeicher ist die Höhe des SB entscheidend!
  - Höhe von binären SB

- **Ziel:** Aufbau sehr breiter Bäume von geringer Höhe
  - in Bezug auf Knotenstruktur vollständig ausgeglichen
  - effiziente Durchführung der Grundoperationen
  - Zugriffsverhalten ist weitgehend unabhängig von Anzahl der Sätze

  ➥ Einsatz als Zugriffs- oder Indexstruktur für 10 als auch für 10^8 Sätze

- **Bezugsgröße bei Mehrwegbäumen:**
  Seite = Transporteinheit zum Externspeicher

  Unterteilung eines großen binären Suchbaumes in Seiten:



  ➥ Beachte: Seiten werden immer größer !

Mehrwegbäume (2)

- **Vorfahr (1965): ISAM**
  (statisch, periodische Reorganisation)

- **Weiterentwicklung: B- und B*-Baum**
  - B-Baum: 1970 von R. Bayer und E. McCreight entwickelt
  - treffende Charakterisierung (D. Comer): “The Ubiquitous B-Tree”
  - ➥ dynamische Reorganisation durch Splitten und Mischen von Seiten

- **Grundoperationen:**
  - Einfügen eines Satzes
  - Löschen eines Satzes
  - direkter Schlüsselzugriff auf einen Satz
  - sortiert sequentieller Zugriff
    auf alle Sätze oder auf Satzbereiche

- **Höhe von Mehrwegbäumen**
  $h_B = O(\log_{k+1} n)$

- **Balancierte Struktur:**
  - unabhängig von Schlüsselmenge
  - unabhängig von Einfügereihenfolge

- **Breites Spektrum von Anwendungen**
  - Dateiorganisation („logische Zugriffsmethode“, VSAM)
  - Datenbanksysteme
    (Varianten des B*-Baumes sind in allen DBS zu finden!)
  - Text- und Dokumentenorganisation
  - Suchmaschinen im Web
  - . . .

B-Bäume

- **Def.:** Seien k, h ganze Zahlen, $h \geq 0$, $k > 0$. Ein B-Baum B der Klasse $\tau(k, h)$ ist entweder ein leerer Baum oder ein geordneter Suchbaum mit folgenden Eigenschaften:

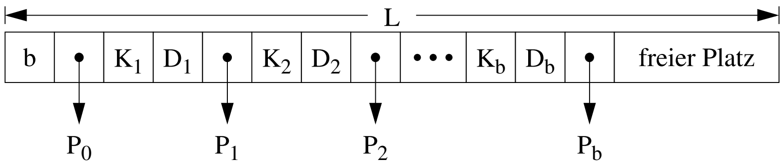
  (1) Jeder Pfad von der Wurzel zu einem Blatt hat die gleiche Länge h-1.

  (2) Jeder Knoten außer der Wurzel und den Blättern hat mindestens k+1 Söhne.
  Die Wurzel ist ein Blatt oder hat mindestens 2 Söhne.

  (3) Jeder Knoten hat höchstens 2k+1 Söhne.

  (4) Jedes Blatt mit der Ausnahme der Wurzel als Blatt hat mindestens k und höchstens 2k Einträge.

  Für einen B-Baum ergibt sich folgendes Knotenformat:



- **Einträge**
  - Die Einträge für Schlüssel, Daten und Zeiger haben die festen Längen l_b, l_K, l_D und l_p.
  - Die Knoten- oder Seitengröße sei L.
  - Maximale Anzahl von Einträgen pro Knoten $\quad b_{max} = \left\lfloor \dfrac{L - l_b - l_p}{l_K + l_D + l_p} \right\rfloor = 2k$

- **Reformulierung der Def.:**

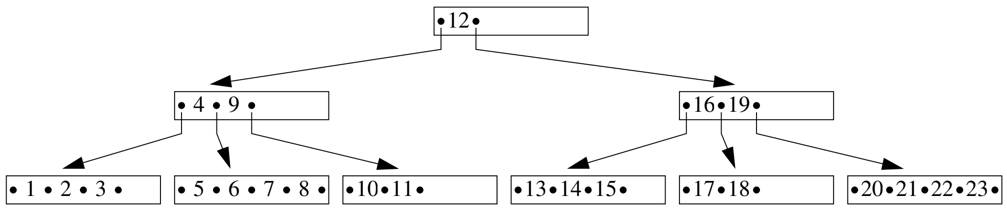
  (4) und (3). Eine Seite darf höchstens voll sein.

  (4) und (2). Jede Seite (außer der Wurzel) muß mindestens halb voll sein.
  Die Wurzel enthält mindestens einen Schlüssel.

  (1) Der Baum ist, was die Knotenstruktur angeht, vollständig ausgeglichen.

B-Bäume (2)

- **Beispiel:** B-Baum der Klasse $\tau(2, 3)$



  - In jedem Knoten stehen die Schlüssel in aufsteigender Ordnung mit $K_1 < K_2 < \ldots < K_b$.
  - Jeder Schlüssel hat eine Doppelrolle als Identifikator eines Datensatzes und als Wegweiser im Baum
  - Die Klassen $\tau(k, h)$ sind nicht alle disjunkt. Beispielsweise ist ein Baum aus $\tau(2, 3)$ mit maximaler Belegung ebenso in $\tau(3, 3)$ und $\tau(4, 3)$ ist.

- **Höhe h**: Bei einem Baum der Klasse $\tau(k, h)$ mit n Schlüsseln gilt für seine Höhe:

$$\log_{2k+1}(n+1) \leq h \leq \log_{k+1}((n+1)/2) + 1 \qquad \text{für } n \geq 1$$

$$\text{und} \quad h = 0 \qquad \text{für } n = 0$$

- **Balancierte Struktur:**
  - unabhängig von Schlüsselmenge
  - unabhängig von ihrer Einfügereihenfolge

➥Wie wird das erreicht ?

B-Bäume - Beispiel

- **Was sind typische Größen in B-Bäumen?**

$$b_{max} = \left\lfloor \frac{L - l_b - l_p}{l_K + l_D + l_p} \right\rfloor = 2k$$

Zahlenwerte in Byte: $L = 4096$, $l_b = 2$, $l_P = 4$, $l_K = 8$-?, $l_D = 88$-?

B-Bäume (3)

- **Einfügen in B-Bäumen**

Wurzel läuft über:

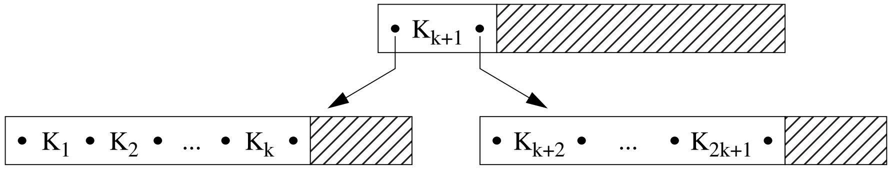
$\bullet\ K_1\ \bullet\ K_2\ \bullet\ \ldots\ \bullet\ K_{2k}\ \bullet \qquad K_{2k+1}$

Fundamentale Operation: Split-Vorgang
1. Anforderung einer neuen Seite und
2. Aufteilung der Schlüsselmenge nach folgendem Prinzip

$$\begin{array}{ccc} & \Uparrow & \\ K_1\ K_2\ \ldots\ K_k & K_{k+1} & K_{k+2}\ \ldots\ K_{2k+1} \end{array}$$

Der mittlere Schlüssel (Median) wird zum Vaterknoten gereicht. Ggf. muß ein Vaterknoten angelegt werden (Anforderung einer neuen Seite).

Hier wird eine neue Wurzel angelegt.

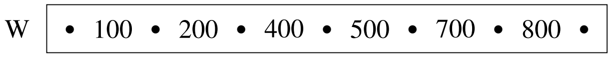


Sobald ein Blatt überläuft, wird ein Split-Vorgang erzwungen, was eine Einfügung in den Vaterknoten (hier: Wurzel) impliziert. Wenn dieser überläuft, muß ein Split-Vorgang ausgeführt werden. Betrifft der Split-Vorgang die Wurzel, so wird das Anlegen einer neuen Wurzel erzwungen. Dadurch vergrößert sich die Höhe des Baumes um 1.

➥Bei B-Bäumen ist das Wachstum von den Blättern zur Wurzel hin gerichtet

B-Bäume - Einfügebeispiel

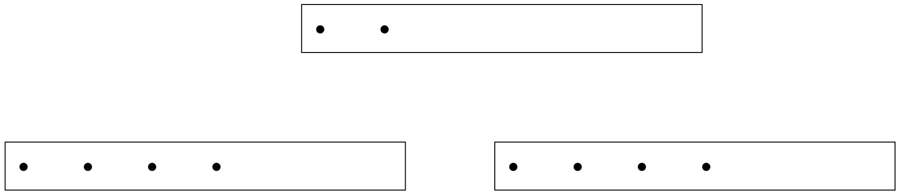
- **Einfügen in Wurzel (=Blatt)**



➥ Einfügen von 600 erzeugt Überlauf

- **Neuaufteilung von W**

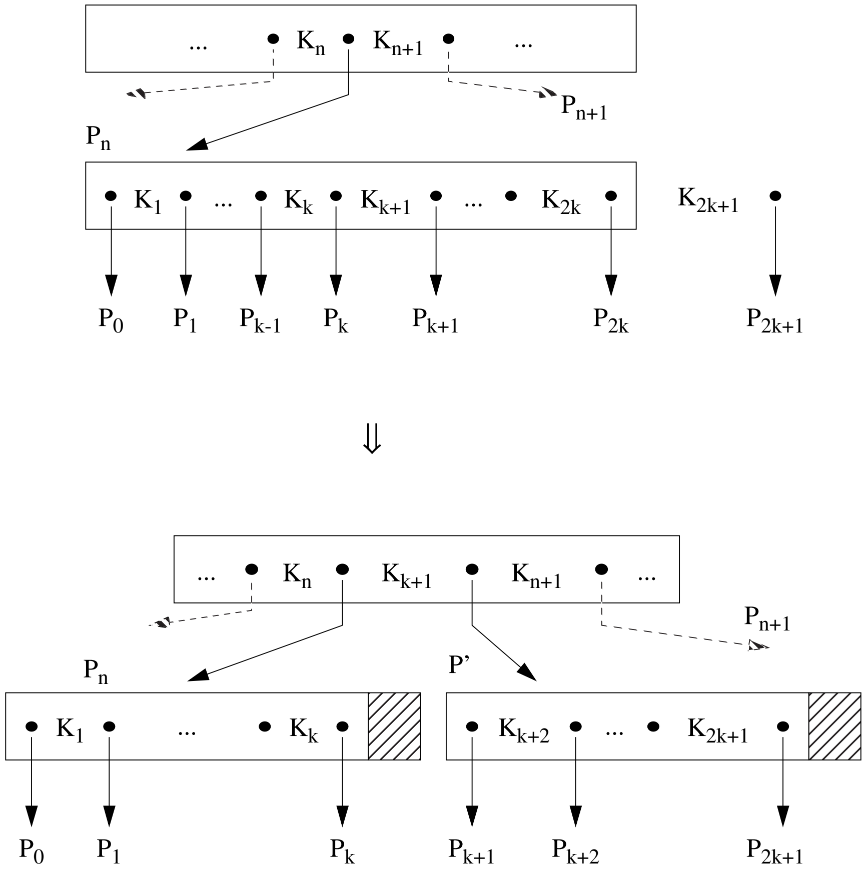
- **B-Baum nach Split-Vorgang**



B-Bäume (4)

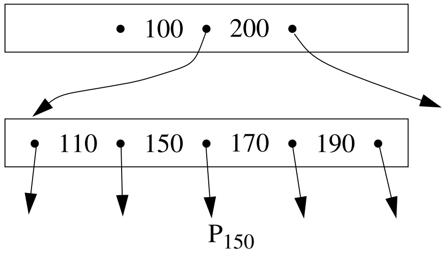
- **Einfügealgorithmus (ggf. rekursiv)**
  - Suche Einfügeposition
  - Wenn Platz vorhanden ist, speichere Element,
    sonst schaffe Platz durch Split-Vorgang und füge ein.

- **Split-Vorgang als allgemeines Wartungsprinzip**



B-Bäume - Split-Beispiel

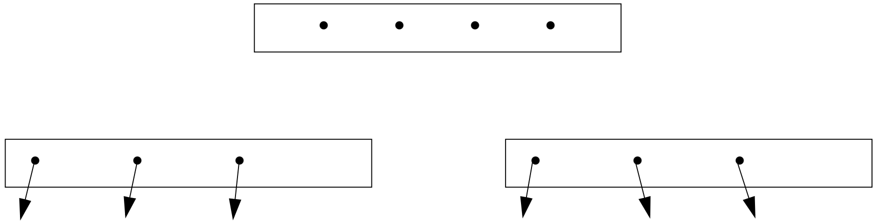
- **Split-Vorgang - rekursives Schema**



➥ Einfügen von 130 mit P_{130}

- **Neuaufteilung des kritischen Knotens**

- **Baumausschnitt nach Split-Vorgang**

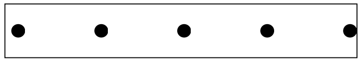


B-Bäume (5)

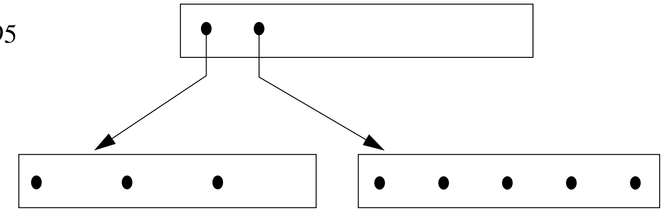
- **Aufbau eines B-Baumes der Klasse τ(2, h)**

Einfügereihenfolge:

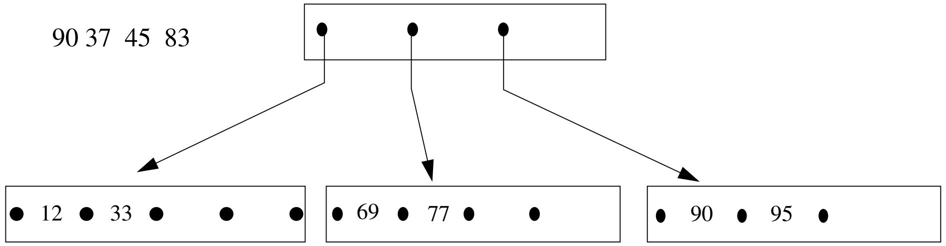
77 12 48 69



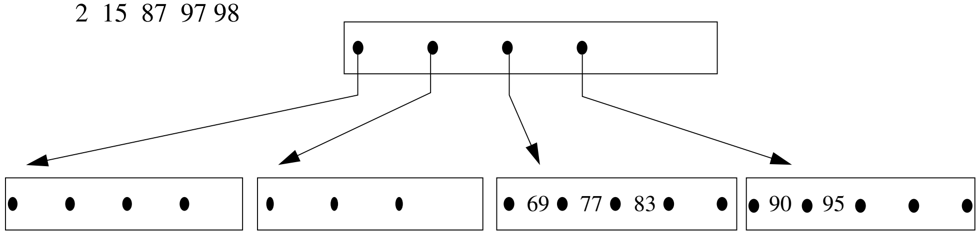
33 89 95



90 37 45 83

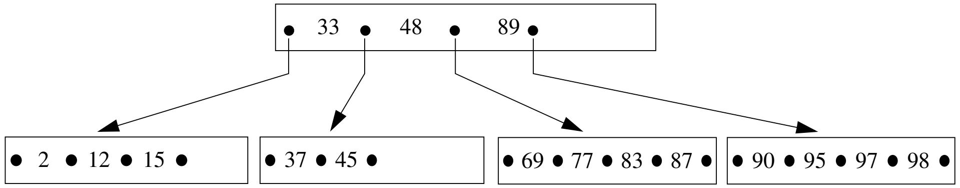


2 15 87 97 98

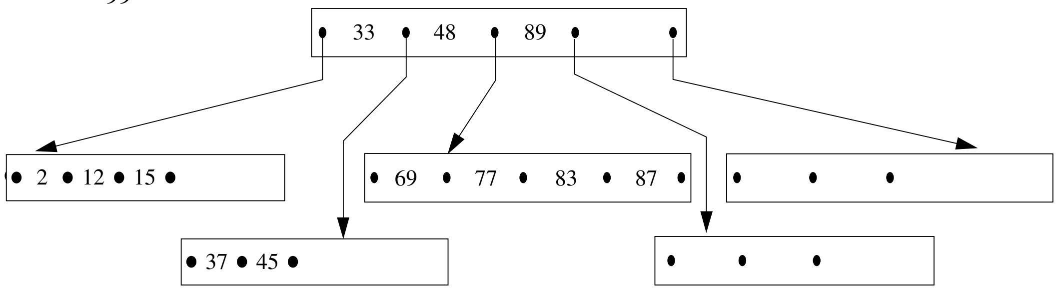


- **Aufbau eines B-Baumes der Klasse $\tau(2, h)$**

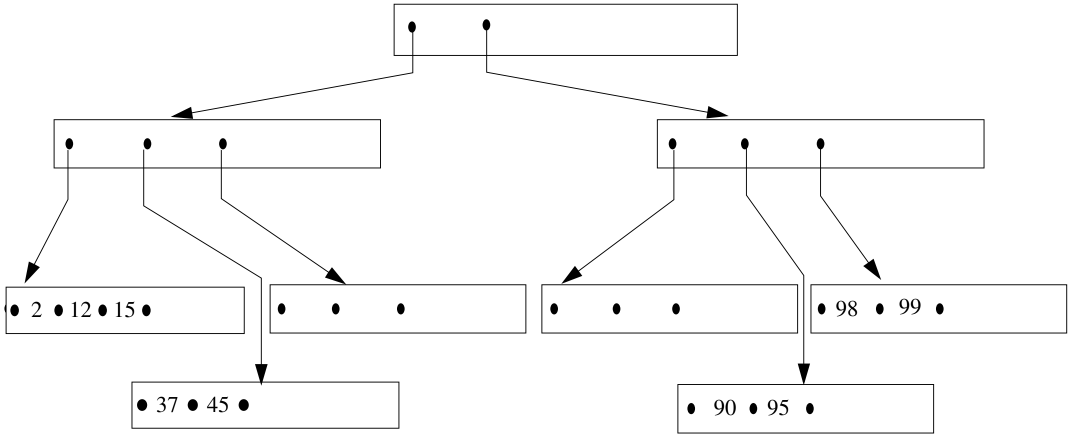
Einfügereihenfolge:



99



50



- **Kostenanalyse für Suchen**
  - Anzahl der zu holenden Seiten : f (fetch)
  - Anzahl der zu schreibenden Seiten : w (write)

- **Direkte Suche**
  - $f_{min} = 1$: der Schlüssel befindet sich in der Wurzel
  - $f_{max} = h$: der Schlüssel ist in einem Blatt

- **mittlere Zugriffskosten**
  (Näherung wegen $(k \approx 100 - 200)$ möglich)

$$f_{avg} = h \qquad \text{für } h = 1$$

$$\text{und} \quad h - \frac{1}{k} \le f_{avg} \le h - \frac{1}{2k} \qquad \text{für } h > 1.$$

- Beim B-Baum sind die maximalen Zugriffskosten h eine gute Abschätzung der mittleren Zugriffskosten.

  ➥Bei $h = 3$ und einem $k = 100$ ergibt sich $2.99 \le f_{avg} \le 2.995$

- **Sequentielle Suche**
  - Durchlauf in symmetrischer Ordnung : $f_{seq} = N$
  - Pufferung der Zwischenknoten im HSP wichtig!

- **Kostenanalyse für Einfügen**
  - günstigster Fall - kein Split-Vorgang: $f_{min} = h$; $w_{min} = 1$;
  - ungünstigster Fall: $f_{max} = h$; $w_{max} = 2h + 1$;
  - durchschnittlicher Fall: $f_{avg} = h; w_{avg} < 1 + \frac{2}{k}$;

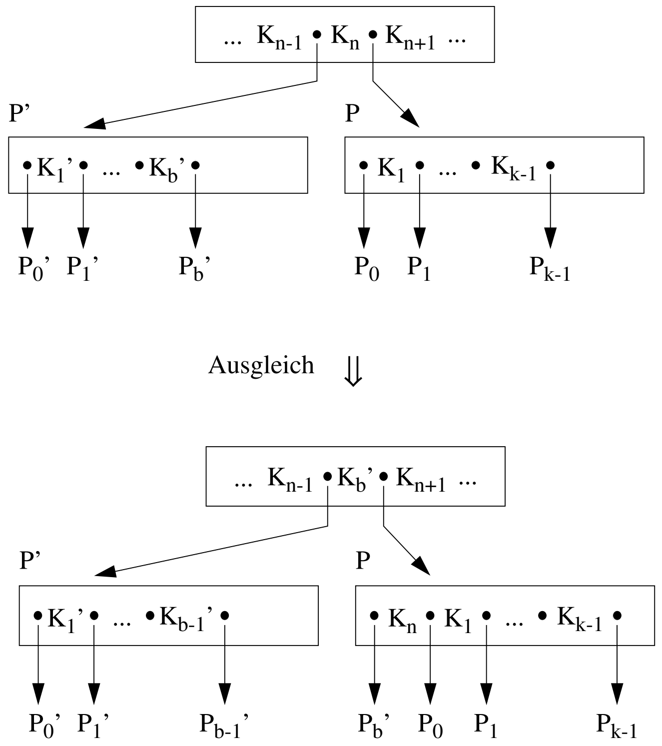
➥ Eine Einfügung kostet bei k=100 im Mittel $w_{avg} < 1 + 2/100$ Schreibvorgänge, d.h., es entsteht eine Belastung von 2% für den Split-Vorgang.

- **Löschen in B-Bäumen**

Die B-Baum-Eigenschaft muß wiederhergestellt werden, wenn die Anzahl der Elemente in einem Knoten kleiner als k wird.

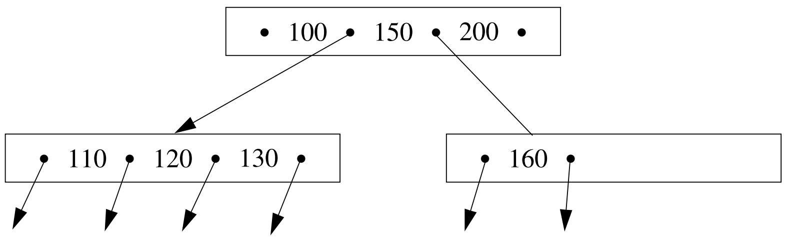
Durch **Ausgleich** mit Elementen aus einer Nachbarseite oder durch **Mischen** (Konkatenation) mit einer Nachbarseite wird dieses Problem gelöst.
Beim Ausgleichsvorgang sind in der Seite P k-1 Elemente und in P' mehr als k Elemente.

(1) Maßnahme: Ausgleich durch Verschieben von Schlüsseln



B-Bäume - Löschbeispiele

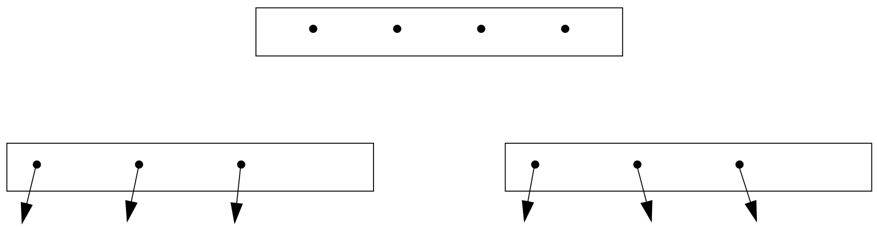
- **Löschprinzip: Ausgleich**



➥ $b_1 > k$ und $b_2 = k - 1$

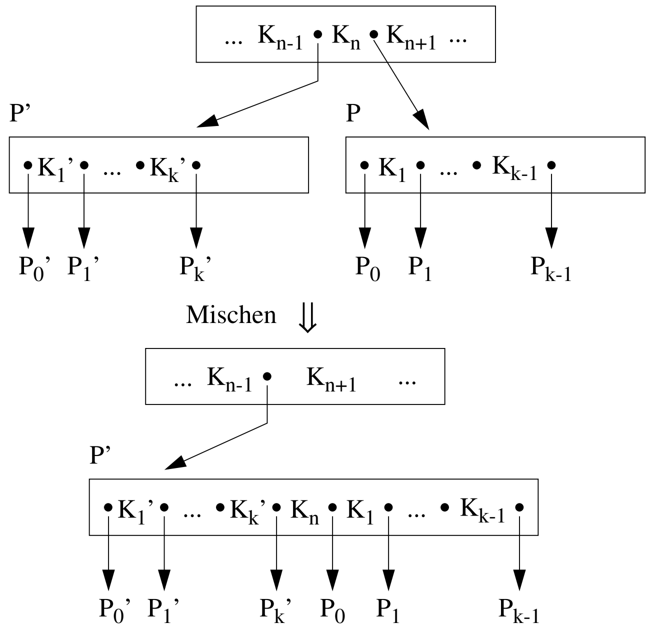
- **Neuaufteilung durch Ausgleich**

- **Baumausschnitt nach Ausgleich**



B-Bäume (10)

(2) **Maßnahme: Mischen von Seiten**



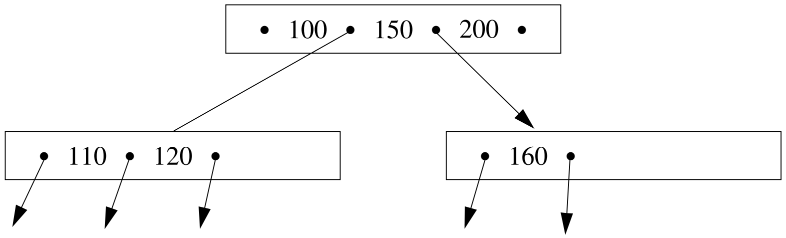
- **Löschalgorithmus**

(1) Löschen in Blattseite
- Suche x in Seite P
- Entferne x in P und wenn
  - a) $b \geq k$ in P: tue nichts
  - b) $b = k-1$ in P und $b > k$ in P’: gleiche Unterlauf über P’ aus
  - c) $b = k-1$ in P und $b = k$ in P’: mische P und P’.

(2) Löschen in innerer Seite
- Suche x
- Ersetze $x = K_i$ durch kleinsten Schlüssel y in $B(P_i)$ oder größten Schlüssel y in $B(P_{i-1})$ (nächstgrößerer oder nächstkleinerer Schlüssel im Baum)
- Entferne y im Blatt P
- Behandle P wie unter (1)

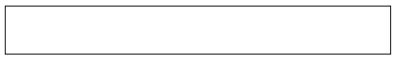
B-Bäume - Löschbeispiele

- **Löschprinzip: Mischen**

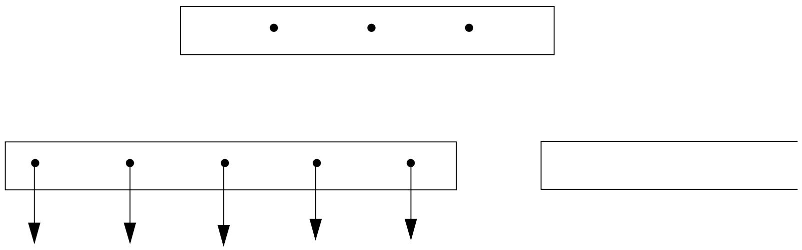


➥ $b_1 = k$ und $b_2 = k - 1$

- **Neuaufteilung durch Mischen**

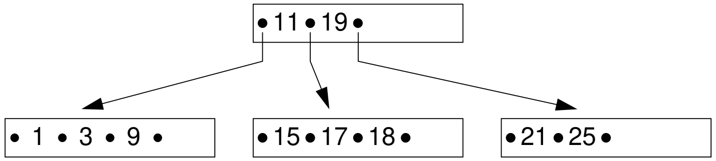


- **Baumausschnitt nach Mischen**

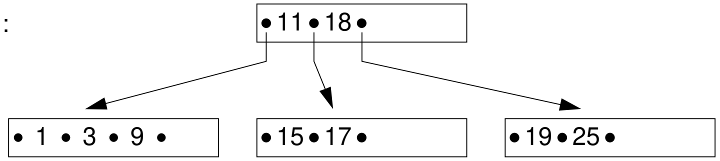


B-Bäume (11)

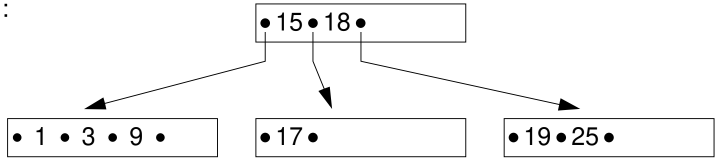
Beispiel: Löschen



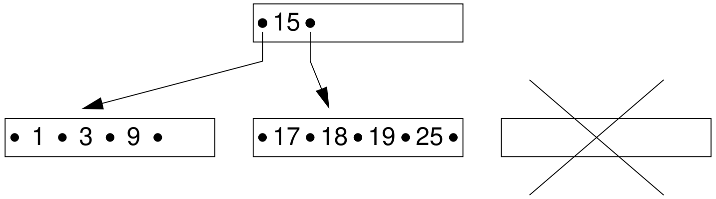
Lösche 21:



Lösche 11:



Mische:



B-Bäume (12)

- **Kostenanalyse für das Löschen**
  - günstigster Fall: $f_{min} = h$; $w_{min} = 1$;
  - ungünstigster Fall (pathologisch): $f_{max} = 2h - 1$; $w_{max} = h + 1$;
  - obere Schranke für durchschnittliche Löschkosten
    (drei Anteile: 1. Löschen, 2. Ausgleich, 3. anteilige Mischkosten):

$$f_{avg} \le f_1 + f_2 + f_3 < h + 1 + \frac{1}{k}$$

$$w_{avg} \le w_1 + w_2 + w_3 < 2 + 2 + \frac{1}{k} = 4 + \frac{1}{k}$$

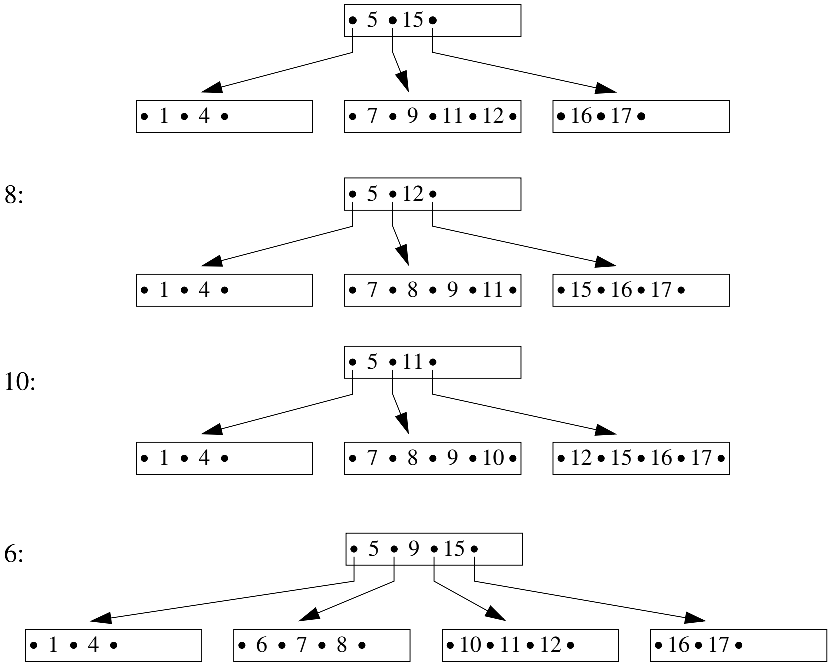
B-Bäume - Überlaufbehandlung

- Welcher Belegungsgrad β_{avg} des B-Baums wird erzielt
  - bei einfacher Überlaufbehandlung (Split-Faktor m = 1) ?
  - beim Einfügen einer sortierten Schlüsselfolge ?
- Einfügen in den B-Baum **bei doppeltem Überlauf** ($\tau(2,2)$)

Einfüge 8:

Einfüge 10:

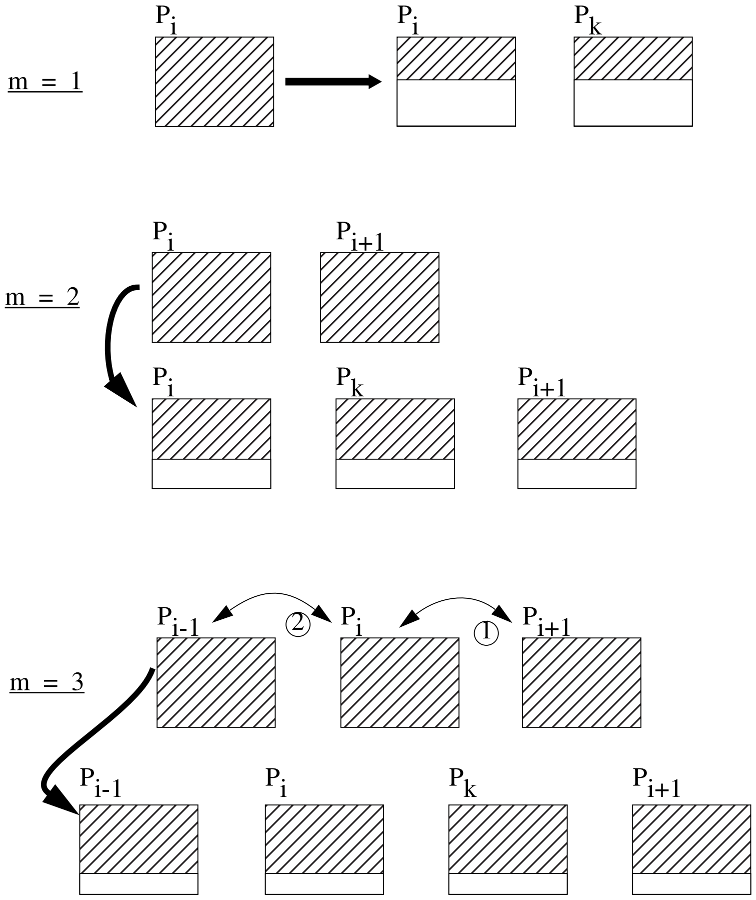
Einfüge 6:



➥Split-Faktor **m = 2**

- **Einfügekosten bei m = 2**
  - $f_{min} = h$; $w_{min} = 1$
  - $f_{max} = 2h - 1$; $w_{max} = 3h$
  - $f_{avg} \le h + 1 + \frac{2}{k}$; $w_{avg} \le 1 + 2 + \frac{3}{k} = 3 + \frac{3}{k}$

- **Verbesserung des Belegungsgrades**
  ➥ verallgemeinerte Überlaufbehandlung



- **Speicherplatzbelegung** als Funktion des Split-Faktors

		Belegung	
Split-Faktor	β_{min}	β_{avg}	β_{max}
1	1/2 = 50%	**ln 2 ≈ 69%**	1
2	2/3 = 66%	2 · ln (3/2) ≈ 81%	1
3	3/4 = 75%	3 · ln (4/3) ≈ 86%	1
m	$\frac{m}{m+1}$	$m \cdot \ln\left(\frac{m+1}{m}\right)$	1

B*-Bäume

- **Unterscheidungsmerkmale:**

  In **B-Bäumen** spielen die Einträge (K_i, D_i, P_i) in den inneren Knoten zwei ganz verschiedene Rollen:

  - **Die zum Schlüssel K_i gehörenden Daten D_i werden gespeichert.**
  - **Der Schlüssel K_i dient als Wegweiser im Baum.**

  Für diese zweite Rolle ist D_i vollkommen bedeutungslos. In B*-Bäumen wird in inneren Knoten nur die Wegweiser-Funktion ausgenutzt, d.h., es sind nur (K_i, P_i) als Einträge zu führen.

  - Die Information (K_i, D_i) wird in den Blattknoten abgelegt.
  - Für einige K_i ergibt sich eine redundante Speicherung. Die inneren Knoten bilden also einen Index (index part), der einen schnellen direkten Zugriff zu den Schlüsseln gestattet.
  - Der Verzweigungsgrad erhöht sich beträchtlich, was wiederum die Höhe des Baumes reduziert.
  - Die Blätter enthalten alle Schlüssel mit ihren zugehörigen Daten in Sortierreihenfolge. Durch Verkettung aller Blattknoten (sequence set) läßt sich eine effiziente sequentielle Verarbeitung erreichen, die beim B-Baum einen umständlichen Durchlauf in symmetrischer Ordnung erforderte.

  ➥ **Die für den praktischen Einsatz wichtigste Variante des B-Baums ist der B*-Baum.**

- **Def.:** Seien k, k* und h* ganze Zahlen, $h^* \geq 0$, k, $k^* > 0$.
  Ein <u>B*-Baum</u> B der Klasse τ (k, k*, h*) ist entweder ein leerer Baum oder ein geordneter Suchbaum, für den gilt:

  (1) Jeder Pfad von der Wurzel zu einem Blatt besitzt die gleiche Länge h*-1.

  (2) Jeder Knoten außer der Wurzel und den Blättern hat mindestens k+1 Söhne, die Wurzel mindestens 2 Söhne, außer wenn sie ein Blatt ist.

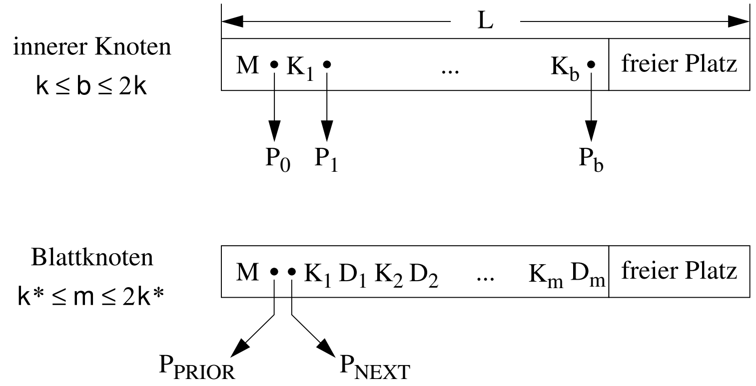
  (3) Jeder innere Knoten hat höchstens 2k+1 Söhne.

  (4) Jeder Blattknoten mit Ausnahme der Wurzel als Blatt hat mindestens k* und höchstens 2k* Einträge.

  Bem.: Der so definierte Baum heißt in der Literatur gelegentlich auch B^+-Baum.

B*-Bäume (2)

- **Unterscheidung von zwei Knotenformaten:**

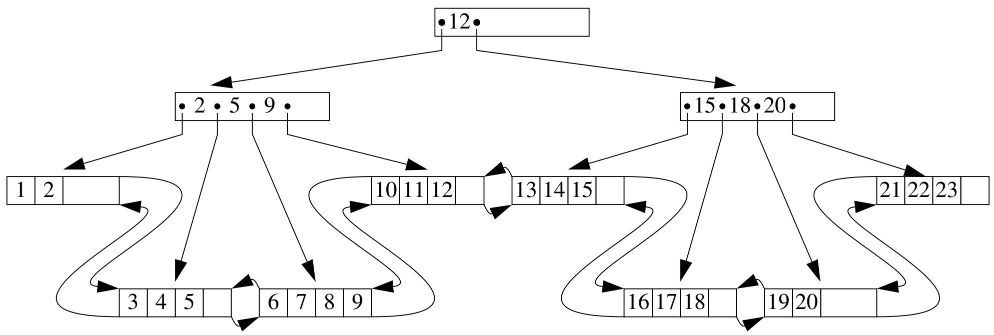


Das Feld M enthalte eine Kennung des Seitentyps sowie die Zahl der aktuellen Einträge. Da die Seiten eine feste Länge L besitzen, läßt sich aufgrund der obigen Formate k und k* bestimmen:

$$L = l_M + l_P + 2 \cdot k(l_K + l_P)\ ; \qquad k = \left\lfloor \frac{L - l_M - l_P}{2 \cdot (l_K + l_P)} \right\rfloor$$

$$L = l_M + 2 \cdot l_P + 2 \cdot k^*(l_K + l_D)\ ; \qquad k^* = \left\lfloor \frac{L - l_M - 2l_P}{2 \cdot (l_K + l_D)} \right\rfloor$$

- **B*-Baum der Klasse τ(3, 2, 3)**



B*-Bäume - Beispiel

- **Was sind typische Größen in B*-Bäumen?**

Zahlenwerte in Byte: $L = 4096$, $l_M = 4$, $l_P = 4$, $l_K = 8$-?, $l_D = 92$ -?

- **Innerer Knoten**

$$k = \left\lfloor \frac{L - l_M - l_P}{2 \cdot (l_K + l_P)} \right\rfloor$$

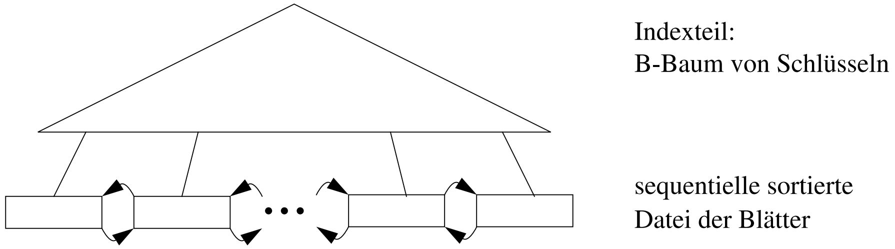
- **Blatt**

$$k^* = \left\lfloor \frac{L - l_M - 2l_P}{2 \cdot (l_K + l_D)} \right\rfloor$$

B*-Bäume (3)

- **Erklärungsmodell für den B*-Baum**

Der B*-Baum läßt sich auffassen als eine gekettete sequentielle Datei von Blättern, die einen Indexteil besitzt, der selbst ein B-Baum ist. Im Indexteil werden insbesondere beim Split-Vorgang die Operationen des B-Baums eingesetzt.



- **Grundoperationen beim B*-Baum**

(1) Direkte Suche:
Da alle Schlüssel in den Blättern sind, kostet jede direkte Suche h* Zugriffe. h* ist jedoch im Mittel kleiner als h in B-Bäumen. Da f_{avg} beim B-Baum in guter Näherung mit h abgeschätzt werden kann, erhält man also durch den B*-Baum eine effizientere Unterstützung der direkten Suche.

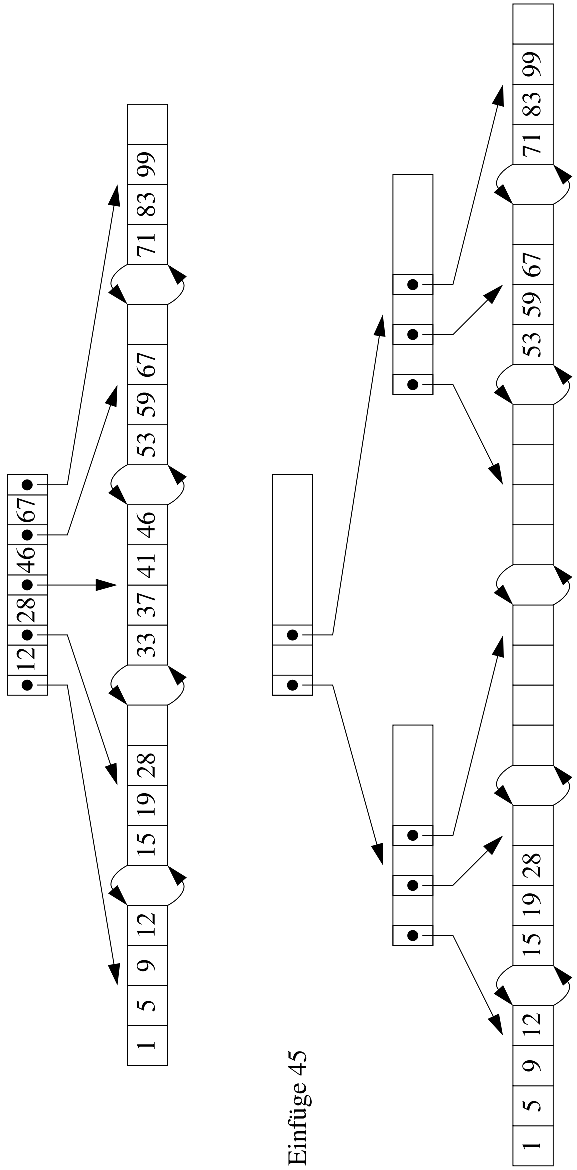
(2) Sequentielle Suche:
Sie erfolgt nach Aufsuchen des Linksaußen der Struktur unter Ausnutzung der Verkettung der Blattseiten. Es sind zwar ggf. mehr Blätter als beim B-Baum zu verarbeiten, doch da nur h*-1 innere Knoten aufzusuchen sind, wird die sequentielle Suche ebenfalls effizienter ablaufen.

(3) Einfügen:
Es ist von der Durchführung und vom Leistungsverhalten her dem Einfügen in einen B-Baum sehr ähnlich. Bei inneren Knoten wird die Spaltung analog zum B-Baum durchgeführt. Beim Split-Vorgang einer Blattseite muß gewährleistet sein, daß jeweils die höchsten Schlüssel einer Seite als Wegweiser in den Vaterknoten kopiert werden. Die Verallgemeinerung des Split-Vorgangs ist analog zum B-Baum.

(4) Löschen:
Datenelemente werden immer von einem Blatt entfernt (keine komplexe Fallunterscheidung wie beim B-Baum). Weiterhin muß beim Löschen eines Schlüssels aus einem Blatt dieser Schlüssel nicht aus dem Indexteil entfernt werden; er behält seine Funktion als Wegweiser.

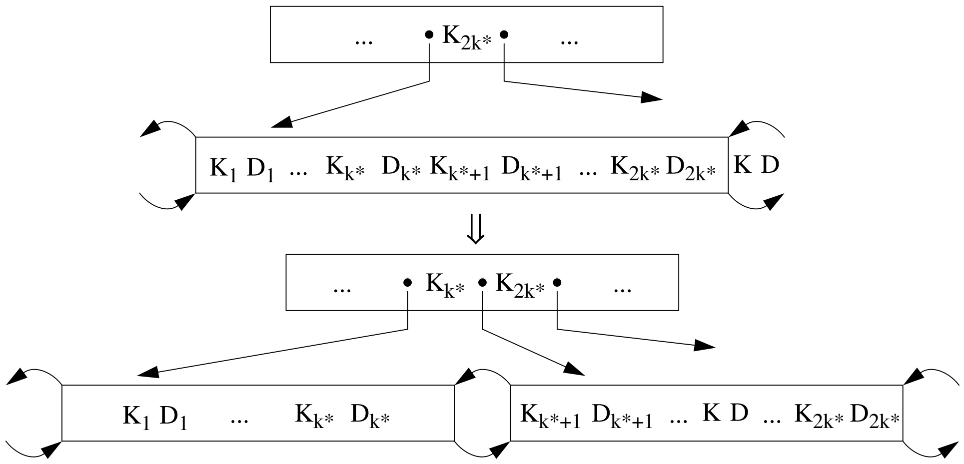
B*-Bäume (4)

Einfügen im B*-Baum

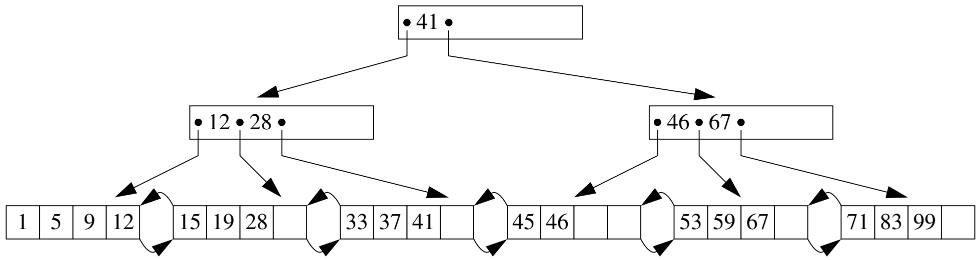


B*-Bäume (5)

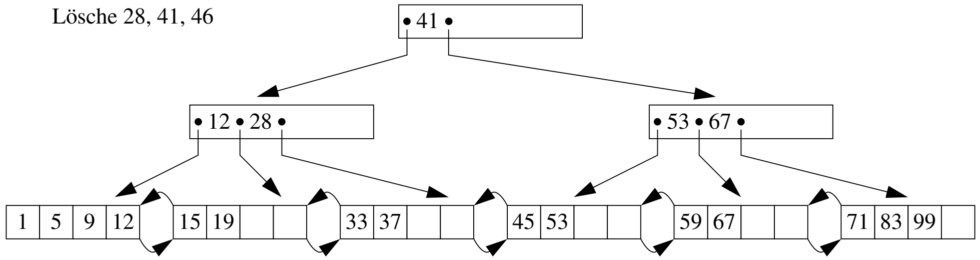
- **Schema für Split-Vorgang:**



- **Löschen im B*-Baum: $\tau(2, 2, h^*)$**



Lösche 28, 41, 46



B*-Bäume (6)

- **Verbesserung der Baumbreite (fan-out)**
  - Erhöhung der Anzahl der Zeiger in den inneren Knoten
  - Schlüsselkomprimierung und Nutzung von „Wegweisern"
- **Schlüsselkomprimierung**
  - Zeichenkomprimierung hat nur begrenztes Optimierungspotential
  - **Präfix-Suffix-Komprimierung** ermöglicht beim B*-Baum weit höhere Anzahl von Einträgen pro Seite (Front and Rear Compression)
    - u.a. in VSAM eingesetzt
    - gespeichert werden nur solche Zeichen eines Schlüssels, die sich vom Vorgänger und Nachfolger unterscheiden
- ***Verfahrensparameter*:**

V = Position im Schlüssel, in der sich der zu komprimierende Schlüssel vom *Vorgänger* unterscheidet

N = Position im Schlüssel, in der sich der zu komprimierende Schlüssel vom *Nachfolger* unterscheidet

F = V - 1 (Anzahl der Zeichen des komprimierten Schlüssels, die mit dem Vorgänger übereinstimmen)

L = MAX (N-F, 0) Länge des komprimierten Schlüssels

Schlüssel	V	N	F	L	kompr. Schlüssel
HARALD	1	4	0	4	HARA
HARTMUT	4	2	3	0	-
HEIN	2	5	1	4	EIN.
HEINRICH	5	5	4	1	R
HEINZ	5	3	4	0	-
HELMUT	3	2	2	0	-
HOLGER	2	1	1	0	-

➥Durchschnittl. komprimierte Schlüssellänge ca. 1.3 - 1.8

B*-Bäume - Schlüsselkomprimierung

- **Präfix-Komprimierung**

- **Präfix-Suffix-Komprimierung**

- **Anwendungsbeispiel für die Präfix-Suffix-Komprimierung**

Schlüssel (unkomprimiert)		V	N	F	L	Wert
CITY_OF_NEW_ORLEANS	... GUTHERIE, ARLO	1	6	0	6	CITY_O
CITY_TO_CITY	... RAFFERTTY, GERRY	6	2	5	0	
CLOSET_CHRONICLES	... KANSAS	2	2	1	1	L
COCAINE	... CALE, J.J	2	3	1	2	OC
COLD_AS_ICE	... FOREIGNER	3	6	2	4	LD_A
COLD_WIND_TO_WALHALLA	... JETHRO_TULL	6	4	5	0	
COLORADO	... STILLS, STEPHEN	4	5	3	2	OR
COLOURS	... DONOVAN	5	3	4	0	
COME_INSIDE	... COMMODORES	3	13	2	11	ME_INSIDE__
COME_INSIDE_OF_MY_GUITAR	... BELLAMY_BROTHERS	13	6	12	0	
COME_ON_OVER	... BEE_GEES	6	6	5	1	O
COME_TOGETHER	... BEATLES	6	4	5	0	
COMING_INTO_LOS_ANGELES	... GUTHERIE, ARLO	4	4	3	1	I
COMMOTION	... CCR	4	4	3	1	M
COMPARED_TO_WHAT?	... FLACK, ROBERTA	4	3	3	0	
CONCLUSION	... ELP	3	4	2	2	NC
CONFUSION	... PROCOL_HARUM	4	1	3	0	

➥Welche Platzeinsparung läßt sich erzielen ?

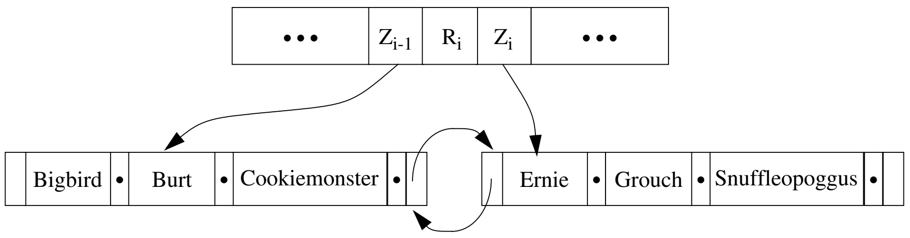
- **Präfix-B-Bäume**

Wegen der ausschließlichen Wegweiser-Funktion der Referenzschlüssel R_i ist es nicht nötig, die Optimierungsmaßnahmen auf die Schlüssel zu stützen, die in den Blattknoten tatsächlich vorkommen.

➥Einsatz von minimalen Separatoren als Wegweiser in den inneren Knoten

- Beispiel: Konstruktion irgendeines Separators S mit der Eigenschaft

Cookiemonster ≤ S < Ernie



Schlüsselbelegung in einem Präfix-B-Baum

- **Höhe des B*-Baumes**

Die Anzahl der Blattknoten bei minimaler Belegung eines B*-Baumes ergibt sich zu

$$B_{min}(k, h^*) = \begin{cases} 1 & \text{für } h^* = 1 \\ 2(k+1)^{h^*-2} & \text{für } h^* \geq 2 \end{cases}$$

Für die Anzahl von Elementen erhalten wir

$$n_{min}(k, k^*, h^*) = 1 \qquad \text{für } h^* = 1 \text{ und}$$

$$n_{min}(k, k^*, h^*) = 2k^* \cdot (k+1)^{h^*-2} \qquad \text{für } h^* \geq 2 \tag{0.1}$$

Bei maximaler Belegung gilt für die Anzahl der Blattknoten

$$B_{max}(k, h^*) = (2k+1)^{h^*-1} \qquad \text{für } h^* \geq 1$$

und für die Anzahl der gespeicherten Elemente

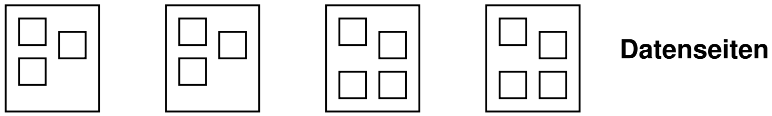
$$n_{max}(k, k^*, h^*) = 2k^* \cdot (2k+1)^{h^*-1} \qquad \text{für } h^* \geq 1 \tag{0.2}$$

Mit Hilfe von (0.1) und (0.2) läßt sich leicht zeigen, daß die Höhe h* eines B*-Baumes mit n Datenelementen begrenzt ist durch

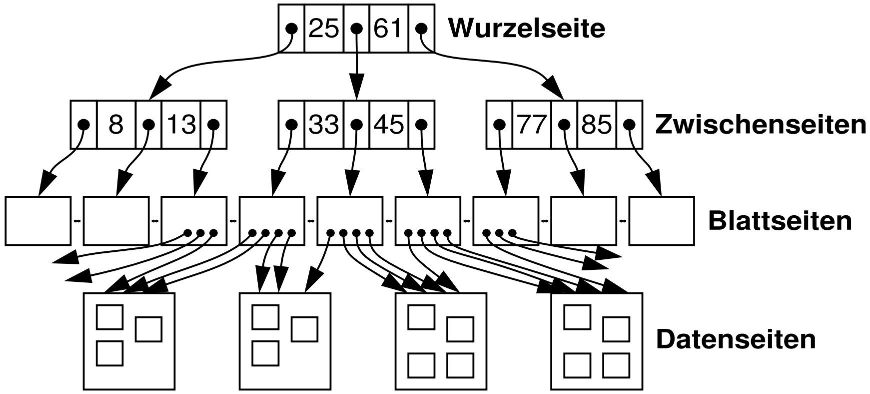
$$1 + \log_{2k+1}\frac{n}{2k^*} \leq h^* \leq 2 + \log_{k+1}\frac{n}{2k^*} \qquad \text{für } h^* \geq 2.$$

Zugriff auf eine Satzmenge

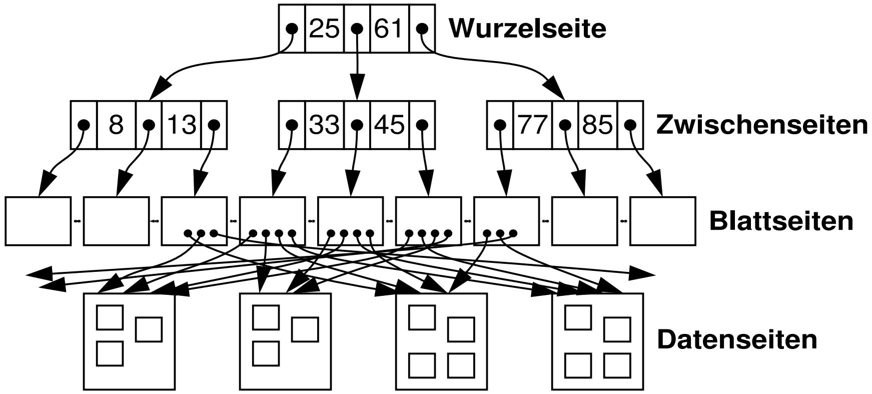
- **Einfaches Erklärungs- und Kostenmodell**
  - m = #Seiten im Segment, N_Z = #Seiten einer Zeigerliste
  - b = mittlere #Sätze /Seite, q = #Treffer der Anfrage
- **Datei- oder Segment-Scan**



- **Index-Scan mit Cluster-Bildung**



- **Index-Scan ohne Cluster-Bildung**



Informationssuche – strukturierte Daten

- **DB-Beispiel**

Schema

ANGESTELLTER

Satztyp (Relation)

Ausprägungen

PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	PEINL	PFOERTNER	2100	63
497	KINZINGER	KOPIST	2800	25
498	MEYWEG	KALLIGRAPH	4500	56

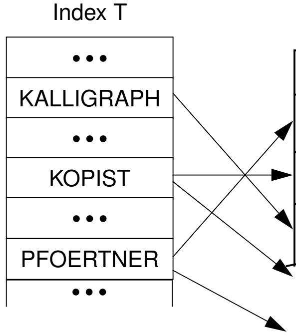
- **Suche in DBS bei strukturierten Daten**
  - Zeichen-/Wertvergleich:
    (TAETIGKEIT = ‘PFOERTNER’) AND (ALTER > 60)
  - <u>**exakte**</u> Fragebeantwortung:
    alle Sätze mit spezifizierter Eigenschaft werden gefunden
    (und nur solche)
  - Suche nach syntaktischer Ähnlichkeit: (TAETIGKEIT LIKE ‘%PF%RTNER’)
    LIKE-Prädikat entspricht der Maskensuche
- **Annahmen**
  - N_S = Anzahl physischer Seitenzugriffe
  - ANGESTELLTER hat n Sätze, in Datei/Segment mit m Seiten
  - Speicherung:
    - als sequentielle Liste mit mittlerem Blockungsfaktor b
    - verstreut über alle m Seiten
- **Sequentielle Suche** (z. B. nach TAETIGKEIT = ’KALLIGRAPH’)
  - Datei- oder Segment-Scan
  - N_S =
  - im Mehrbenutzerbetrieb? Jeder sucht nach anderen Informationen!

Informationssuche (2) – strukturierte Daten

- **Anfrage**

  Select *
  From ANGESTELLTER
  Where TAETIGKEIT = ’KALLIGRAPH’

- **Indexierung bei strukturierten Daten**
  - Invertierung von Attributen erlaubt „direkten Zugriff“ über die einzelnen Werte
  - Beispiel: TAETIGKEIT = {PFOERTNER, KOPIST, KALLIGRAPH, . . .}



PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	PEINL	PFOERTNER	2100	63
497	KINZINGER	KOPIST	2800	25
498	MEYWEG	KALLIGRAPH	4500	56

- **Annahmen**
  - Gleichverteilung der Attributwerte; Unabhängigkeit der Attribute
  - j_i = Anzahl der Attributwerte von Attribut A_i
  - Liste von Zeigern (TIDs) verweist vom Attributwert zu den Sätzen
  - Index als B*-Baum mit Höhe h_B realisiert

- **Index-Suche**
  - Suche im Index und anschließend gezielter Zugriff auf Sätze (Treffer)
  - $h_T = 2$, $j_T = 1000$, $n = 10^6$, N_{ZT} =
  - N_S =

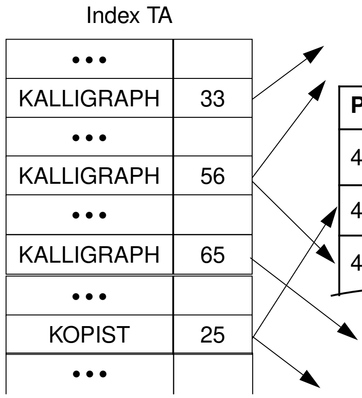
➥ Invertierung eines Attributs wird bestimmt durch den erwarteten Leistungsgewinn bei der Anfrageauswertung

Informationssuche (3) – strukturierte Daten

- **Anfrage**

  Select *
  From ANGESTELLTER
  Where TAETIGKEIT = ’KALLIGRAPH’ AND ALTER = 50

- **Mehrattribut-Indexierung**
  - Anlegen eines Index (TAETIGKEIT | ALTER)



PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	PEINL	PFOERTNER	2100	63
497	KINZINGER	KOPIST	2800	25
498	MEYWEG	KALLIGRAPH	4500	56

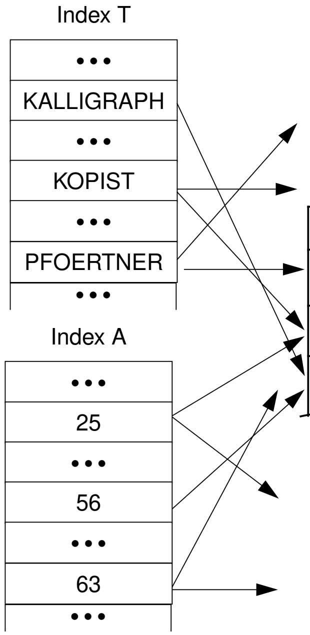
  - $h_{TA} = 3$, $j_T = 1000$, $j_A = 50$, $j_{TA} = j_T \cdot j_A$, $n = 10^6$, N_{ZTA} =
  - N_S =
  - erhöhter Spezialisierungsgrad bei der Indexnutzung
  - Nutzung von Index TA nur für TAETIGKEIT oder nur für ALTER?

Informationssuche (4)
– strukturierte Daten

- **Anfrage**

  Select *
  From ANGESTELLTER
  Where TAETIGKEIT = 'KALLIGRAPH' AND ALTER = 50

- **Kombination von zwei separaten Indexen: TAETIGKEIT und ALTER**



PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	PEINL	PFOERTNER	2100	63
497	KINZINGER	KOPIST	2800	25
498	MEYWEG	KALLIGRAPH	4500	56

- $h_T = h_A = 2$, $j_T = 1000$, $j_A = 50$, $n = 10^6$, $N_{ZT} =$, $N_{ZA} =$

- $N_S =$

- besser: $N_S =$

Informationssuche (5)
– unstrukturierte Daten

- **Daten in IRS**
  - Objekte sind nur durch Dokumenttyp und Wert (D_i/W_k) beschrieben
  - Die Bedeutung der Objekte ist „inhärent"
  - Sie sind durch „lange" Werte repräsentiert und in speziellen Containern (Dateien) gespeichert

- **Einsatzmöglichkeiten**
  - Bibliotheken, Literaturrecherche, z. B. im Internet (WWW)
  - Informationsdienste, Informations-Broker:
    Chemie, Recht, Patente, . . .

- **Beispiel**

  Call for Papers (Dokumenttyp)

  Die rasante Entwicklung des Web hat gerade im Datenbankbereich einerseits eine Vielzahl neuer Anwendungsfelder eröffnet, andererseits aber auch neue Herausforderungen geschaffen. Vor diesem Hintergrund soll der Workshop eine Plattform für Forscher und Praktiker bilden, um Ergebnisse der Datenbanktechnologie und -theorie mit Anforderungen und Erfahrungen aus Internet-Anwendungen abzustimmen. Im einzelnen sollen folgende Themenstellungen sowie damit verbundene Bereiche behandelt werden: . . .

- **Was wird gesucht?**

  Select *
  From ???
  Where ???

- **Wie wird inhaltsbasiert (nach W_k) gesucht?**

  (Beispiel: Informatik-Bibliothek)
  - sequentiell: Volltextsuche
  - Indexnutzung

Informationssuche (6)
– unstrukturierte Daten

- **Indexierung bei unstrukturierten Daten**
  - Invertierung der gesamten Texte der Dokumente vom Typ D_i durch Bibliothekar oder durch spezielle Programme
  - Wertevorrat für Deskriptoren (Schlüsselwörter) ist nicht begrenzt!

  ➥ typischerweise: Anlegen eines Index pro Typ D_i als B*-Baum

- **Was sind geeignete Deskriptoren für „Call for Papers”-Index**

- **Art der Suche**
  - Anfragen relativ **unscharf**
  - **Suchbegriffe** des Benutzers müssen **„irgendwie”** mit den Deskriptoren im Index korrespondieren
  - **Mehrdeutigkeit:** bei Wörtern in Dokumenten und Anfragen (Synonym-, Homonymproblem usw.)
  - Ähnlichkeitssuche (*nearest neighbor, best match, pattern matching* usw.)
  - Ergebnisbewertung: Relevanzproblem (*Precision*, *Recall*)

- **Verbesserung**
  - Synonymsuche, unscharfe Suche usw. sollen unterstützt werden
  - Einsatz eines Thesaurus oder einer Ontologie[1] (Begriffssystematik): Festlegung von Begriffen und ihren Beziehungen zueinander

  ➥ „Verstehen natürlicher Sprache”, Erkennen von Mehrdeutigkeiten sind **„harte Probleme“**
  Bsp.: “Time flies like an arrow – fruit flies like a banana” (Groucho Marx)

1. Ontologie ist ursprünglich eine philosophische Disziplin, neuerdings ein Modebegriff der Informatik. Nach Meyers Enzyklopädischem Lexikon ist Ontologie die „Lehre von dem Wesen und den Eigenschaften des Seienden”. In der Informatik ist sie die „formale Spezifikation eines bestimmten Gegenstandsbereichs in Form eines Begriffssystems”.

Informationssuche (7)
– semi-strukturierte Daten

- **Beispiel: wohl-geformtes XML-Dokument**

```
<book ISBN=“1575213346“>
      <title>Presenting XML</title>
      <author><firstname>Richard</firstname><lastname>Light</lastname></author>
      <author><firstname>Tim</firstname><lastname>Bray</lastname></author>
      <date>Sept.1997</date>
      <price currency=“USD“>19.99</price>
      <comment rating=“4“>
            <writtenby>
                  <firstname>Harald</firstname><lastname>Schöning</lastname>
            </writtenby>
            <text>A quite useful book for <userclass>beginners</userclass>.</text>
      </comment>
      <comment>
            <writtenby>
                  <firstname>A</firstname><lastname>Reader</lastname>
            </writtenby>
            <text> I did not like the cover</text>
      </comment>
</book>
```

➥ Die **Semantik** der Tags muß speziell festgelegt werden

- **Verbesserte Suchunterstützung**
  - Dokumentenstruktur kann bei Suche ausgenutzt werden
  - **Indexierung** läßt sich verbessern, da mit XML Inhalte genauer spezifiziert werden können
  - aber: XML-Elemente können wieder „lange” Werte sein!

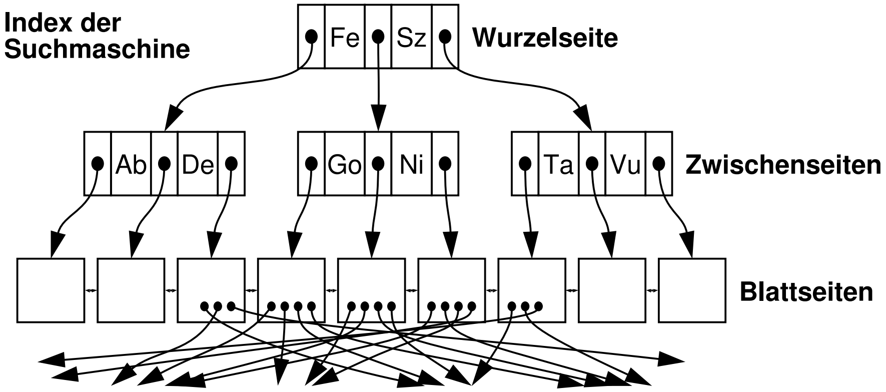
➥ **Indexierung und Suche** erfordern DBS- und IRS-Techniken

Indexierung bei
– un- und semi-strukturierten Dokumenten

- **Indexaufbau**
  - typischerweise B*-Baum
  - Programm sucht/bestimmt die Deskriptoren für den Index!

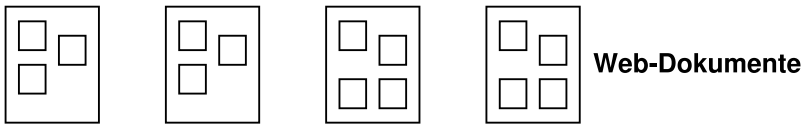
  ➥ Wie kommen schlechte Suchergebnisse zustande?

- **Vereinfachtes Beispiel zur Suche im WWW**
  - Einstieg über eine bekannte Adresse (URL) und Verfolgen von HyperLinks (Navigation)
  - Nutzung von Suchmaschinen
    - Zugriff über HTML-Formular
    - Angabe von Suchkriterien (Suche von „außen“)
    - Ergebnis: Liste von Adressen der WWW-Dokumente (manchmal nach „Relevanz“ bewertet und geordnet)



Ergebnis:
Liste von URLs (http://...), ggf. aufbereitet durch Bewertungsfunktionen

Benutzerzugriff aufs WWW:



Indexierung – Probleme (1)

- **Trennung, Indexierung, Klassifizierung erfordern Stammbildung, Silbenbestimmung etc.**

- **Automatische Indexierung**
  - erfordert „**Verstehen natürlicher Sprache**“
  - erzeugt manchmal „köstlichste und kindlichste“ Fehler
  - bringt zuweilen „herrlichen Sprachquatsch mit Tiefsinn“ hervor

- **Beispiele:**

Winterspor - torte	Beat - mung
Gra - binschrift	Fre - aks
Leich - tathleten	So - und
Gebirg - sorte	Seinein - sel
Bet - trost	bein - halten
Schu - labschluß	Mattsch - eibe

- **Poetische Varianten:**

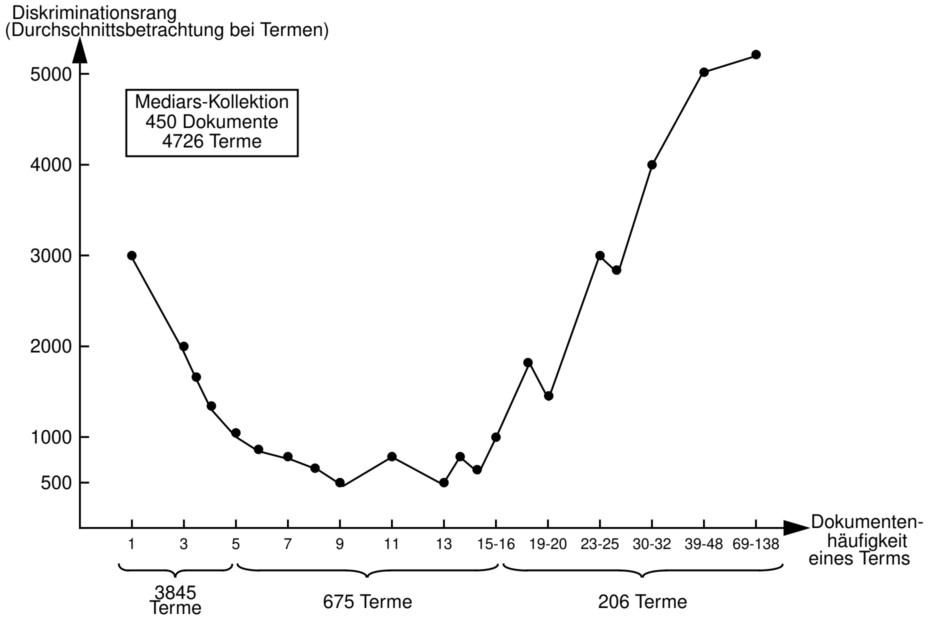
  Wel - traum
  Ka - minnische
  Auslöseele - ment
  Autoren - nen
  Seele - opard

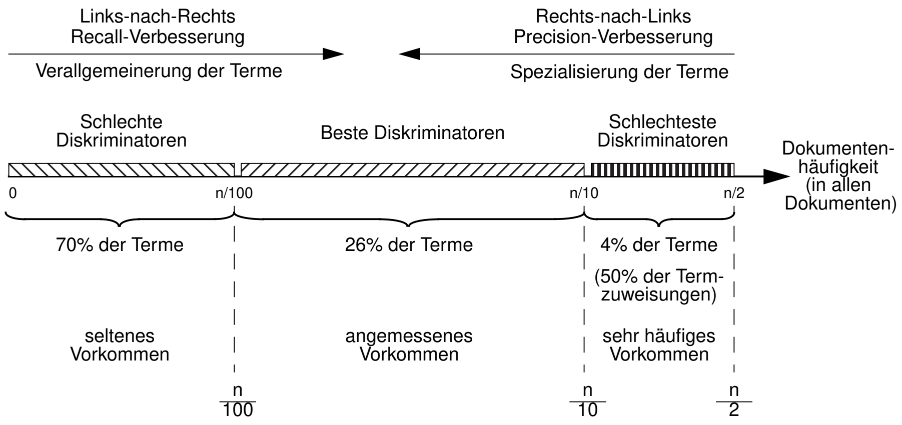
- **Kuriose Trennungen:**

  Gassi - cherung
  Nachteil - zug
  Galauni - form
  Talent - wässerung
  Spargel - der
  Gehörner - ven
  Urin - stinkt

Indexierung – Probleme (2)

- **Welche Terme sollen als Deskriptoren für den Index ausgewählt werden?**
- **Ergebnis typischer Indexierungsversuche**





➥ **Zu spezielle und zu allgemeine Deskriptoren sind ungeeignet!**
Warum?

Ergebnis der Anfrageauswertung

- **Anfragen auf Tabellen (in relationalen DBS)**

Tabelle: PROFESSOREN

PNR	NAME	FB	FG
1234	HÄRDER	INFORMATIK	DBS
5678	STREICH	R&U	GIS
6780	MITSCHANG	INFORMATIK	DBS

Tabelle: MITARBEITER

MANR	NAME	FB	LEHRE
123 766	ZHANG	INFORMATIK	NEIN
130 680	RITTER	R&U	JA
196 481	ZIMMER	W-ING.	JA
225 332	STEIERT	R&U	NEIN
330 129	JAEDICKE	INFORMATIK	JA
291 758	LOESER	INFORMATIK	NEIN

ANFRAGE:

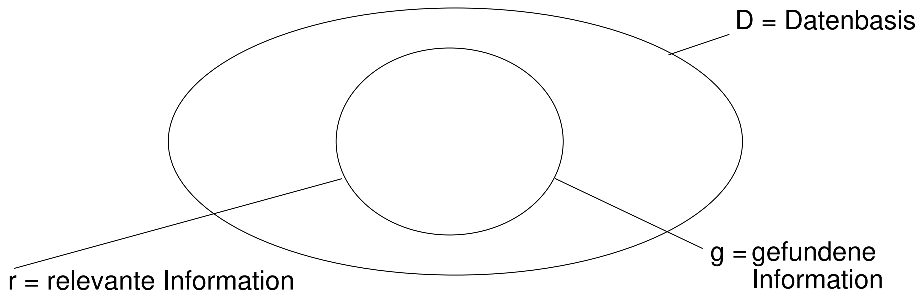
```
SELECT   NAME, FB
FROM     MITARBEITER
WHERE    LEHRE = 'JA'
```

ERGEBNIS:

NAME	FB
RITTER	R&U
ZIMMER	W-ING.
JAEDICKE	INFORMATIK

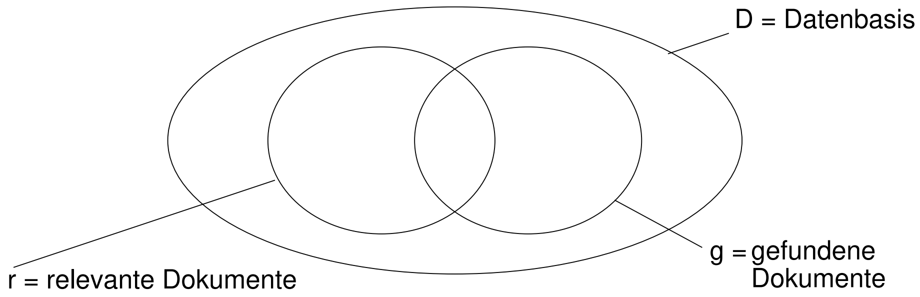
- genau festgelegter Bereich der Anfrage (Tabellen der FROM-Klausel)
- Verknüpfung verschiedener Tabellen ausschließlich über Werte
- Auswahl von Sätzen mit Hilfe von Prädikaten (WHERE-Klausel)
  - ➥ Ergebnis liefert genau alle das Suchprädikat erfüllenden Sätze

- **Auswertung von Anfragen in DBS**



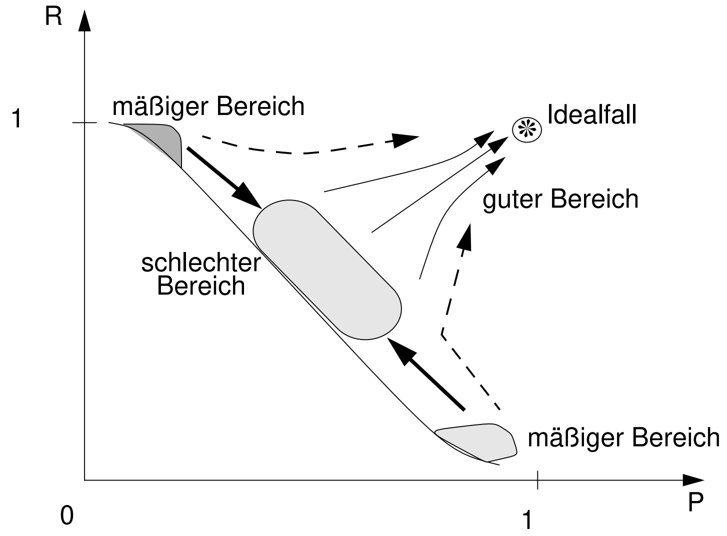
Ergebnis der Anfrageauswertung (2)

- **Auswertung von Anfragen in IRS (und analog im Web)**
  Recall R bezieht sich auf die relevanten Informationen,
  Precision P bezieht sich auf die gefundenen Informationen



$$R = \frac{|r \cap g|}{|r|} \qquad P = \frac{|r \cap g|}{|g|}$$

- **Messung der Systemgüte**
  empirische Ermittlung des R/P-Verhältnisses über Mengen von IRS-Anfragen



➥ **Recall und Precision sind wesentliche IRS-Qualitätsmaße!**
Bei Suchmaschinen im Web zusätzliche Kriterien: Überdeckungsgrad, ...

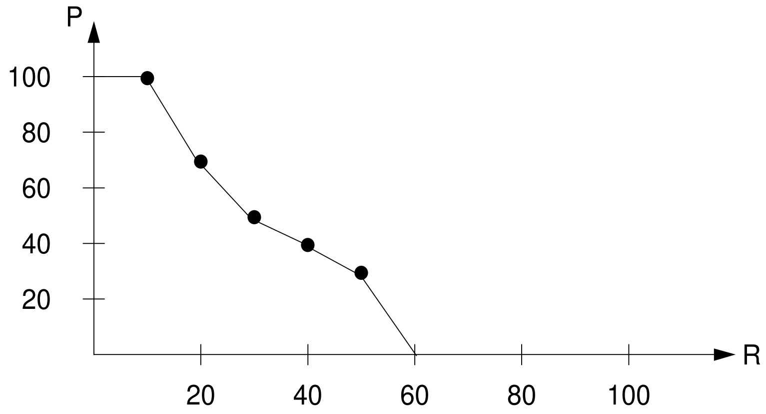
Retrieval-Bewertung - Recall und Precision

- **Variation von R und P**
  - R und P unterstellen, daß alle Dokumente in der Antwortmenge (g) überprüft werden
  - Typischerweise sind die Dokumente nach ihrem Relevanzgrad sortiert (Ranking); der Benutzer inspiziert die Liste „von oben nach unten"
  - R und P variieren mit dem Listendurchlauf
  - genauere Bewertung durch 11 Standard-Recall-Ebenen (P bei 0%, 10%, ..., 100% R)

- **Beispiel für eine Anfrage**
  - Menge der relevanten Dokumente in D für die Anfrage Q sei r_q mit $|r_q| = 10$
    $r_q = \{d_4, d_5, d_8, d_{25}, d_{49}, d_{64}, d_{70}, d_{75}, d_{101}, d_{199}\}$
  - Retrieval-Algorithmus liefere die Menge g_q mit folgendem Ranking:

1. d_{199} •	6. d_8 •	11. d_{47}
2. d_{95}	7. d_{929}	12. d_{62}
3. d_{49} •	8. d_{747}	13. d_{471}
4. d_{12}	9. d_{111}	14. d_{123}
5. d_9	10. d_{25} •	15. d_4 •

- **Precision P bei 11 Standard-Recall-Ebenen**



Retrieval-Bewertung (2)

- **Durchschnittsbildung über mehrere Anfragen**

  R_i = Recall-Ebene i, i = 0, ..., 10 (R_5 bezeichnet Recall-Ebene 50%)

  $P(R_i)$ = Precision bei Recall-Ebene i

  n_q = Anzahl der Anfragen

  $$\overline{P}(R_i) = \sum_{j=1}^{n_q} \frac{P_j(R_i)}{n_q}$$

- **Interpolation bei Recall-Ebenen**

  $P(R_i) = \max_{R_i \le R \le R_{i+1}} P(R)$

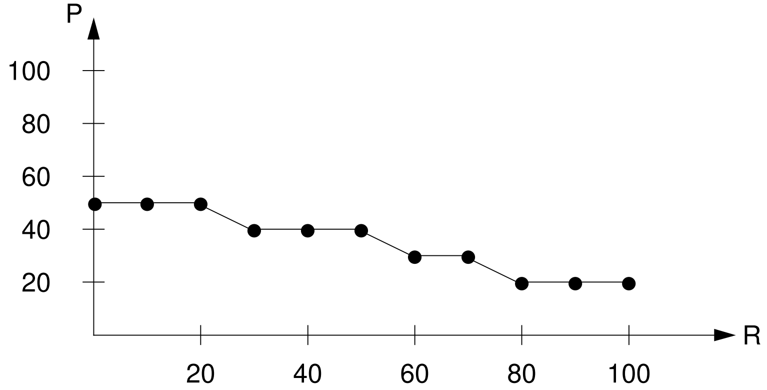
  - Interpolationsbeispiel

  r'_q = $\{d_4, d_9, d_{95}, d_{747}\}$

  - Ranking der gefundenen Menge g_q

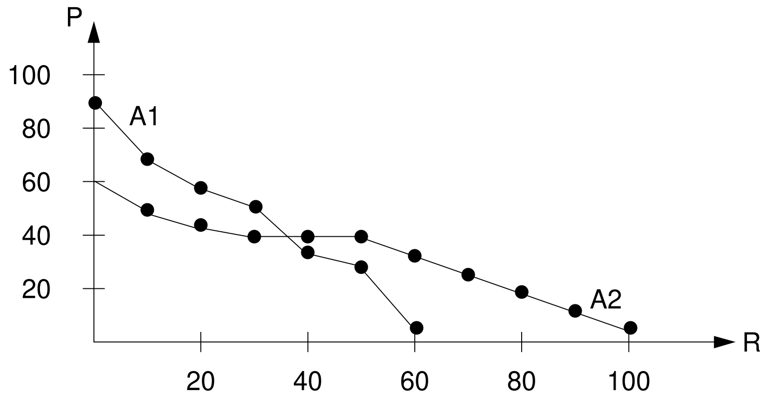
1. d_{199}	6. d_8	11. d_{47}
2. d_{95} •	7. d_{929}	12. d_{62}
3. d_{49}	8. d_{747} •	13. d_{471}
4. d_{12}	9. d_{111}	14. d_{123}
5. d_9 •	10. d_{25}	15. d_4 •

  - d_{95} hat Recall-Level 25% und Precision 50%



Retrieval-Bewertung (3)

- **Vergleich zweier (fiktiver) Retrieval-Algorithmen**



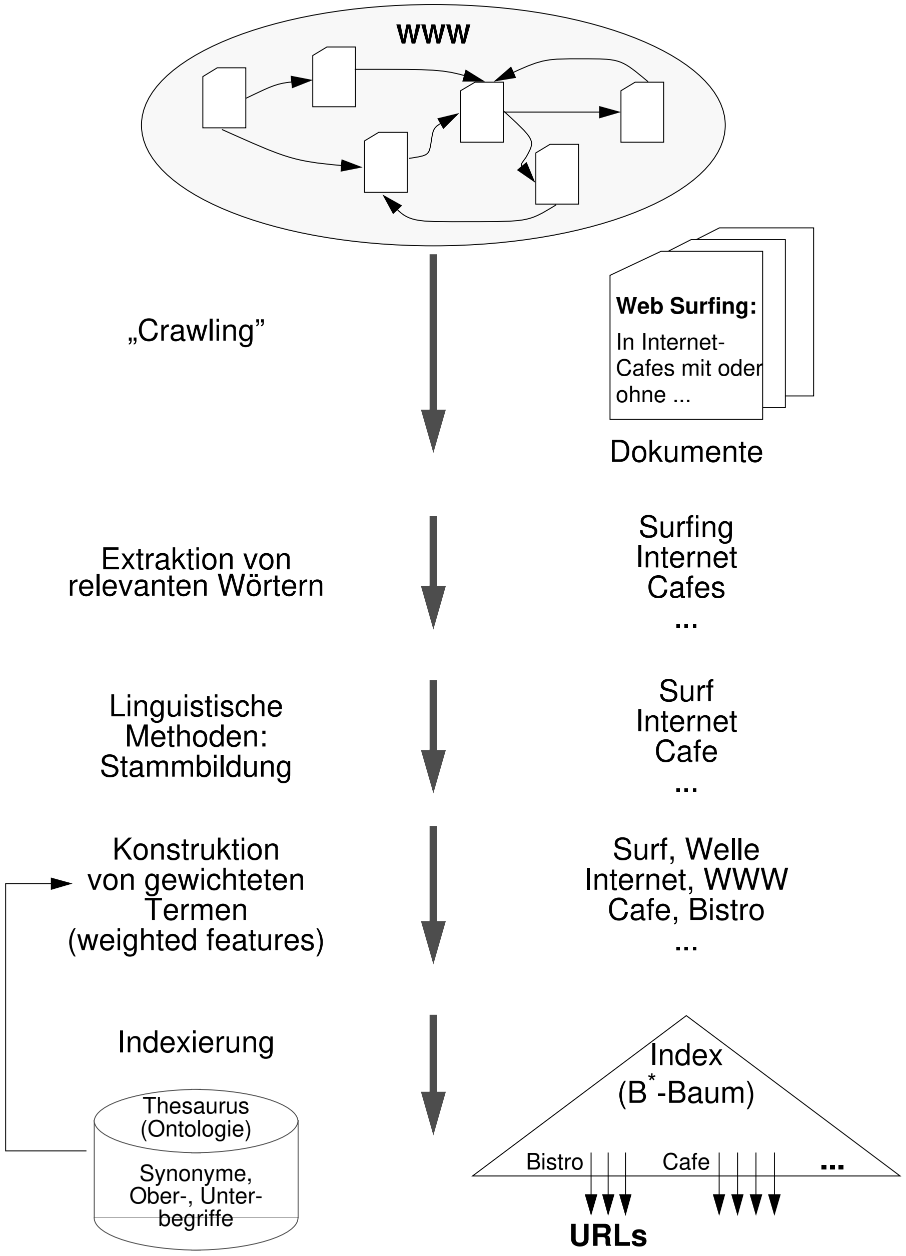
➥ Welche Retrieval-Performance besitzen A1 und A2?

- **Weitere R/P-Variationen**
  - Zusammenfassung zu einem „**goldenen**” Wert
  - **Average Precision at Seen Relevant Documents**
    - Durchschnittsbildung mit den ersten n relevanten Dokumenten
    - Beispiel: P-Werte seien 1, 0.66, 0.5, 0.4, 0.3

      P_{avg} = (1 + 0.66 + 0.5 + 0.4 + 0.3) / 5 = 0.57
  - **R-Precision**
    - n_R sei $|r_q|$;
      R-Precision wird an der Stelle n_R der Ergebnisliste bestimmt
    - Beispiel:
      n_R = 10; mit 4 relevanten Dokumenten bis zu dieser Position:
      R-Precision = 0.4

Suchen im Web

- **Erfassung, Analyse und Indexierung von Web-Dokumenten**



Suchen im Web (2)

- **Suche der Dokumente**
  - Welche Dokumente qualifizieren sich für eine Anfrage?
  - Wie geschieht das Ranking nach **Relevanz** (Ähnlichkeit)?
  - Einsatz des **Vektorraum-Modells** zur Bestimmung der Inhaltsrelevanz
    - Dokumente lassen sich als Term-Vektoren auffassen: $d_i \in [0,1]^{|T|}$
    - Anfrage wird abgebildet auf Anfrage-Vektor: $q \in [0,1]^{|T|}$

Benutzer schickt Anfrage an Suchmaschine SM ↓ "Navigation im Internet"

SM bestimmt mit Index Anfrage-Vektor und relevante Term-Vektoren ↓ $q \in [0,1]^{|T|}$, $d_i \in [0,1]^{|T|}$

SM berechnet Relevanz ↓ Ähnlichkeitsmetrik:

$$\mathrm{sim}(d_i,q) := \frac{\sum_{j=1}^{|T|} d_{ij}\ q_j}{\sum_{j=1}^{|T|} d_{ij}^2 \ \sum_{j=1}^{|T|} q_j^2}$$

(mit komplexer Formel für d_{ij} (Gewichtung))

SM gibt Ergebnisliste aus ↓ Ranking nach absteigender Relevanz

Benutzer bewertet Ergebnis und gibt Hinweise an SM zur Verbesserung ↓ Relevanz-Feedback

Verbesserung der Web-Suche

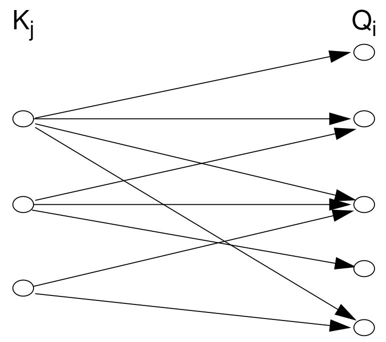
- **Schwierigkeiten bei der Suche**
  - „Harvard" kommt in > Mio. Webseiten vor, Begriff auf Homepage von Harvard ist für Suchmaschinen „uninteressant"
  - Auf der Homepage von „IBM" kommt der Begriff Computer nicht vor
  - ➥ Bei manchen Suchmaschinen werden solche Probleme manuell gelöst: Die „Harvard"-Seite kommt an die Spitze der Liste für „Harvard".
    Aber: Handverlesene Seiten bei unbegrenzter Anzahl von Fragen?

- **Nutzung des Wissens, das in den Hyperlinks steckt**
  - Annahme: Hyperlink ist eine Empfehlung der Seite, auf die er zeigt (Ausnahmen: strukturelle Links (zurück zur Homepage) oder Reklame)
  - Programm Clever von IBM
    Unterscheidung von Webseiten als Quellen und als Knotenpunkte
  - Quellen sind Seiten über ein spezielles Thema, auf die zahlreiche andere Webseiten verweisen (und ihnen damit eine gewisse Glaubwürdigkeit zusprechen); beispielsweise die Homepage von Amnesty International zum Thema Menschenrechte
  - Knotenpunkte sind Seiten, die eine große Anzahl von Quellen auflisten, wie z. B. die Rubrik „Meine Lieblingslinks" einer persönlichen Homepage oder professionelle Portale

- **Quellen und Knotenpunkte**
  als lose und nicht zentral geplante Ordnungsstruktur des Web
  - Eine renommierte Quelle Q_i ist eine Seite, auf die von einer großen Anzahl von Knotenpunkten verwiesen wird
  - Ein Knotenpunkt K_j ist gut, wenn er auf eine große Anzahl von beachtenswerten Quellen verweist
  - ➥ Es wird versucht, die Klassifikationsarbeit der Web-Benutzer auszunutzen

Verbesserung der Web-Suche (2)

- **Suchalgorithmus mit Bewertung von Quellen und Knotenpunkten**
  1. Finden einer Liste von Kandidatenseiten mit herkömmlichen Verfahren
  2. Anfängliche Grobbewertung von Q_i und K_j



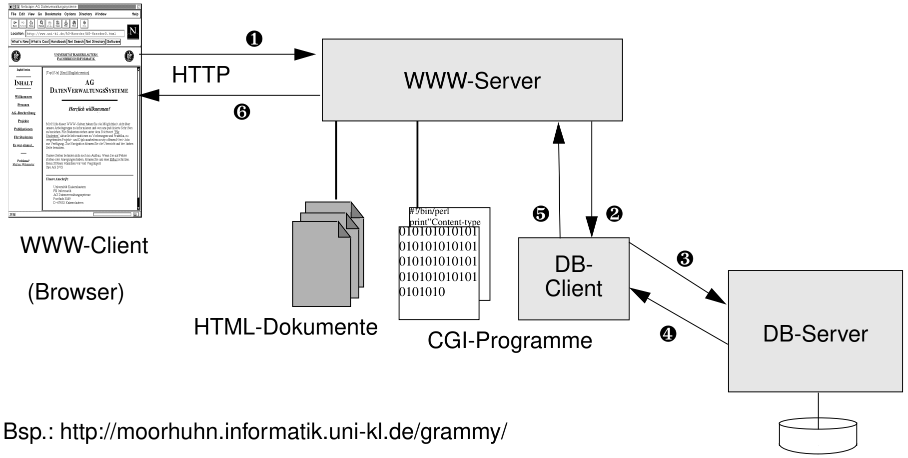
  3. Neubewertung der Q_i: eine Quelle gilt als „gut", wenn viele „gute" Knotenpunkte auf sie verweisen
  4. Neubewertung der K_j: die neue Note von K_j ist umso besser, je besser die bisherigen Noten der Q_i sind, auf die er verweist
  - schnelle Konvergenz: bei 3000 Primärseiten 5 Iterationen, bis sich die Bewertungen kaum noch ändern
  - Endergebnisse sind im wesentlichen unabhängig von den anfänglichen Schätzwerten. Man könnte sogar alle Anfangswerte auf 1 setzen

- **Leistungszahlen von Google (2002)**
  - Suchmaschine besteht aus Cluster von > 10.000 PCs
  - Suchvorgänge
    - durchschnittlich 150 Mio/Tag
    - Spitzenlast > 2000/sec
  - Index über
    - > 2 Mrd Dokumente
    - > 300 Mio Bilder
    - > 700 Mio Usenet-Nachrichten

Dokumentenzugriff im Web

- **Zugriffe im Web**
  - Zustandslose Verbindung zwischen Browser und WWW-Server über HTTP (*HyperText Transfer Protocol*)
  - orts- und plattformunabhängiger Zugriff durch HTTP-Protokoll
  - DB-Zugriff durch externes CGI-Programm (*Common Gateway Interface*) bzw. Server-Erweiterungsmodul (*WWW Server API*) als DB-Client
  - HTML (*HyperText Markup Language*) dient zur Ergebnispräsentation

- **Beispiel mit Anbindung eines DB-Servers**



Bsp.: http://moorhuhn.informatik.uni-kl.de/grammy/

- **Vorgehensweise:**

  ❶ Abschicken des Formulars mit aktuellen Parameterwerten

  ❷ Starten des entsprechenden CGI-Programms (DB-Client)

  ❸ Anfragen des CGI-Programms an den DB-Server

  ❹ Übertragung der Ergebnismenge zum DB-Client

  ❺ Rückgabe der erstellten HTML-Seite an den WWW-Server

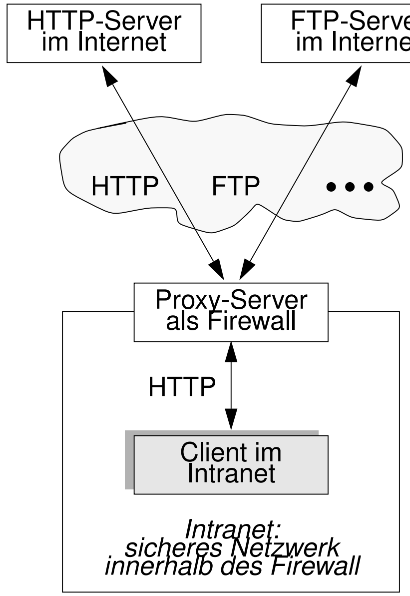
  ❻ Übertragung der HTML-Seite zum Browser

Proxy-Server

- **Bandbreite-Problem im Web**
  - Der Datendurchsatz (Bandbreite) bestimmt z. B. die Geschwindigkeit, mit der eine Datei übertragen wird, oder die Bildrate bei Videokonferenzen
  - Bandbreite wird ständig durch den Einsatz von Kupfer- und Glasfaserkabeln und durch zusätzliche Telekommunikationssatelliten erhöht
  - Trotzdem bleiben gravierende Engpässe in den nächsten Jahren

    ➥ **Abhilfe:** Mehrfachübertragung von Web-Dokumenten gerade über lange Strecken vermeiden

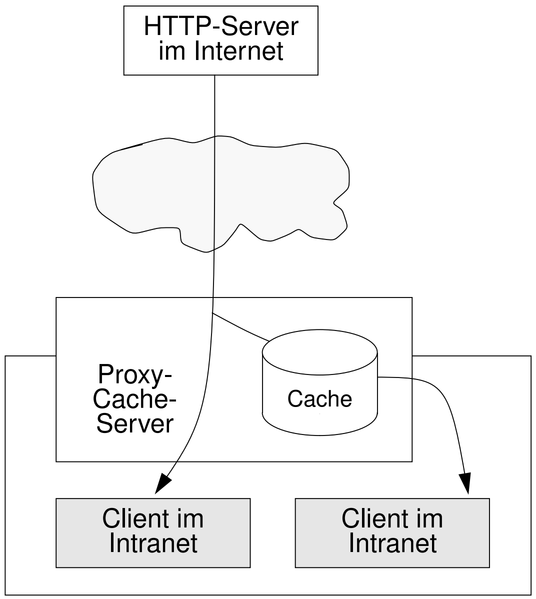
- **Architektur eines Proxy-Servers**



  - Ein „Proxy“ (Bevollmächtigter, Stellvertreter) übernimmt bestimmte Aufgaben im Namen seiner Clients
  - Er kontrolliert in Firewall-Architekturen die Verbindung eines Intranets zum I nternet. Der Zugriff von außen ist durch Firewall eingeschränkt, von innen dagegen ist er üblicherweise uneingeschränkt
  - Anfragen werden von den Clients an den Proxy gerichtet, der diese ggf. weiterleitet; Antworten können z. B. nach Viren durchsucht werden, bevor sie dem Client zugestellt werden

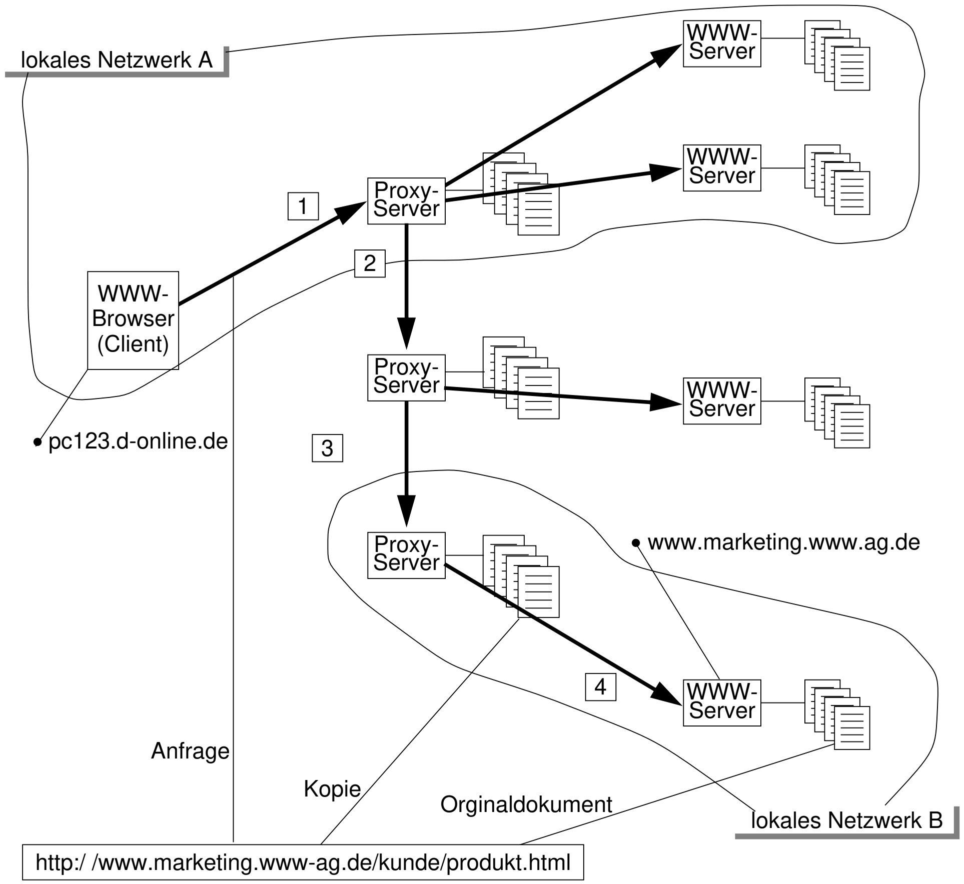
Proxy-Cache-Server

- **Cache** (deutsch: Lager)
  - „kleiner“, schneller Speicher, der den wiederholten Zugriff auf Daten optimieren soll
  - Idee: Kombination eines Proxy-Servers mit einem Cache
- **Architektur eines Proxy-Cache-Servers**



  - spezielle LRU-Ersetzungsverfahren in den Caches (Variationen von *Least Recently Used*)
  - gemeinsame Nutzung von allen Clients (*shared cache*): Cache-Trefferquote in großen Intranets oft 30% - 50%
  - **im allgemeinen Hierarchie von Caches**
    - Browser-Cache oder Client-Cache (privat)
    - Proxy-Cache
    - nationaler Cache
    - internationaler Cache

Prinzip des Dokumentenzugriffs



1. Anforderung wird zunächst an lokalen Proxy-Server (Stellvertreter) weitergeleitet
2. Wenn Suche im Proxy-Cache erfolglos, leitet Proxy-Server die Anforderung zu den Proxy-Servern in der Aufrufhierarchie (der Anfrage) weiter
3. Falls das Dokument in den Proxy-Servern der Aufrufhierarchie nicht gefunden wird, beschafft der letzte Proxy-Server das Dokument beim spezifizierten WWW-Server (oder bei einem Server, der entspr. Replikate verwaltet)
4. Von der Fundstelle aus wird das Dokument entlang der Aufrufhierarchie zum WWW-Browser weitergeleitet. Dabei wird es in den Caches der beteiligten Proxy-Server abgelegt

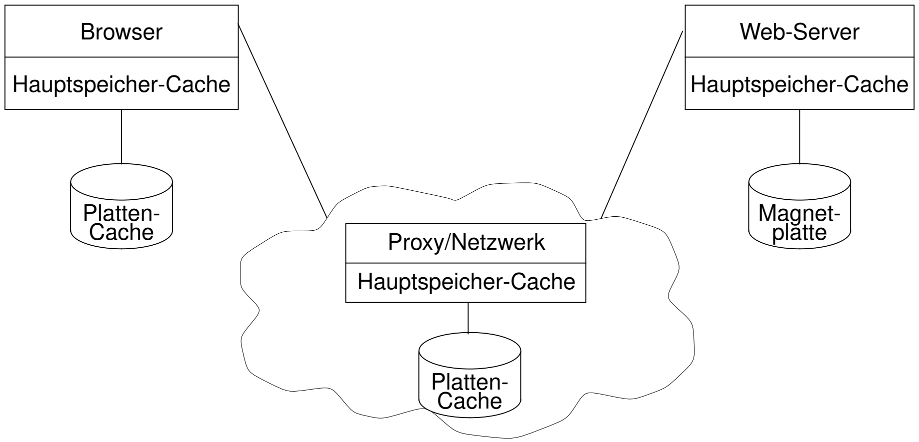
Web-Caching vs. Web-Replikation – eine unscharfe Abgrenzung

- **Caching**
  - Speicherung eines Objektes an einem Ort, der **auf dem Aufsuchweg** liegt
  - Beispiele:
    Browser-Cache, Proxy-Cache, Hauptspeicher-Cache des Servers
  - Cache sieht bei den Zugriffen Treffer und Fehlversuche (hits and misses)

- **Replikation („Push Caching")**
  - Speicherung eines Objektes an einem Ort, an dem **sonst nicht** nach dem Objekt gesucht wird
  - Beispiel: Spiegelung (mirrors)
  - Replikat sieht nur „Treffer"

- **Prefetching („Pull Caching")**
  - Ablegen eines Objektes im Cache **vor** Anforderungen („spekulatives Holen")

Orte für das Caching



Zusammenfassung

- **Ziel einer Speicherhierarchie**

  Geschwindigkeitsanpassung des jeweils größeren und langsameren Speichers an den schnelleren zu vertretbaren Kosten (Kosteneffektivität)

- **Listen auf Externspeichern**
  - unterstützen vorwiegend fortlaufende Verarbeitung von Satzmengen
  - sequentielle Listen gewährleisten Cluster-Bildung

- **Konzept des Mehrwegbaumes**
  - Aufbau sehr breiter Bäume von geringer Höhe (h ~ 3)
  - Bezugsgröße: Seite als Transporteinheit zum Externspeicher
  - Seiten werden künftig immer größer, d. h., das Fan-out wächst weiter

- **B- und B*-Baum gewährleisten eine balancierte Struktur**
  - unabhängig von Schlüsselmenge
  - unabhängig von Einfügereihenfolge

- **Informationssuche bei strukturierten Dokumenten**
  - effektive Indexnutzung
  - genaue Anfrageergebnisse

- **Informationssuche bei un-/semi-strukturierten Dokumenten**
  - Probleme bei der Indexierung
  - Precision/Recall-Metrik
  - ungenaue Ergebnisse: Bewertung erforderlich

- **Zugriff auf Dokumente im Web**
  - Vor- und Nachteile von Proxy-Cache-Servern
  - Prinzip des Dokumentenzugriffs: Einfluß von Caching und Replikation