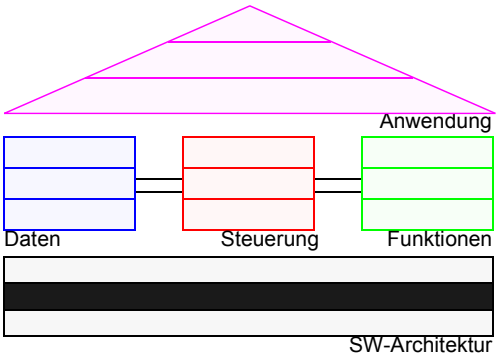
2. E/A-Architektur und Zugriff

- **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 Daten**
  - Zugriffskosten

- **Aufbau des DB-Servers**
  - Aufgaben und Anforderungen
  - Modellierung/Realisierung durch Schichtenmodelle

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** (HSP) 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 (MP), Magnetband, DVD, 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?

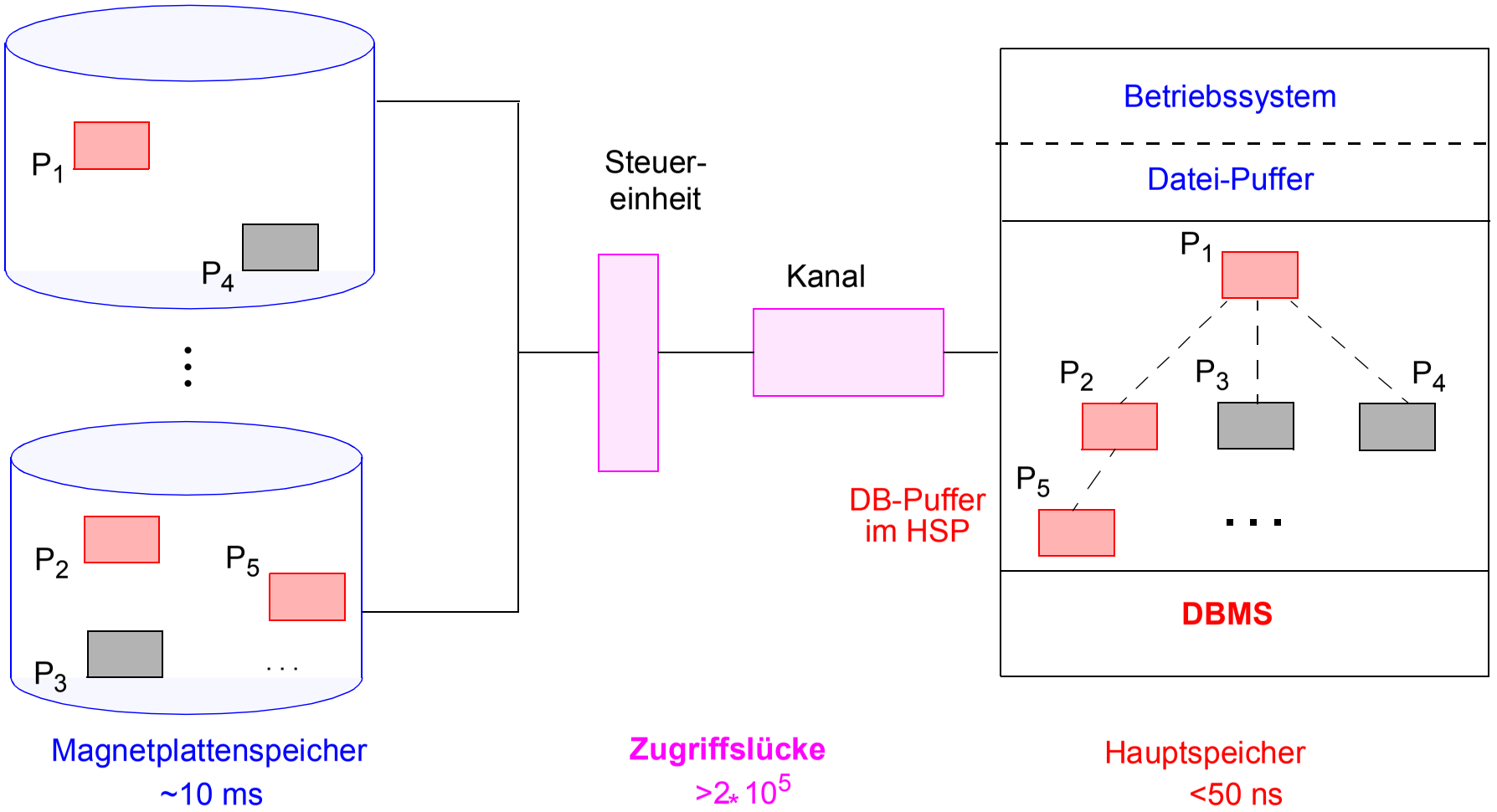
- **Idealer Speicher besitzt**
  - nahezu unbegrenzte Speicherkapazität
  - kurze Zugriffszeit bei wahlfreiem Zugriff
  - hohe Zugriffsraten
  - geringe Speicherkosten
  - Nichtflüchtigkeit
  - Fähigkeit zu logischen, arithmetischen u. ä. Verknüpfungen?

  ➥ Was würde das für einen Externspeicher bedeuten?

Zugriffswege bei einer zweistufigen Speicherhierarchie

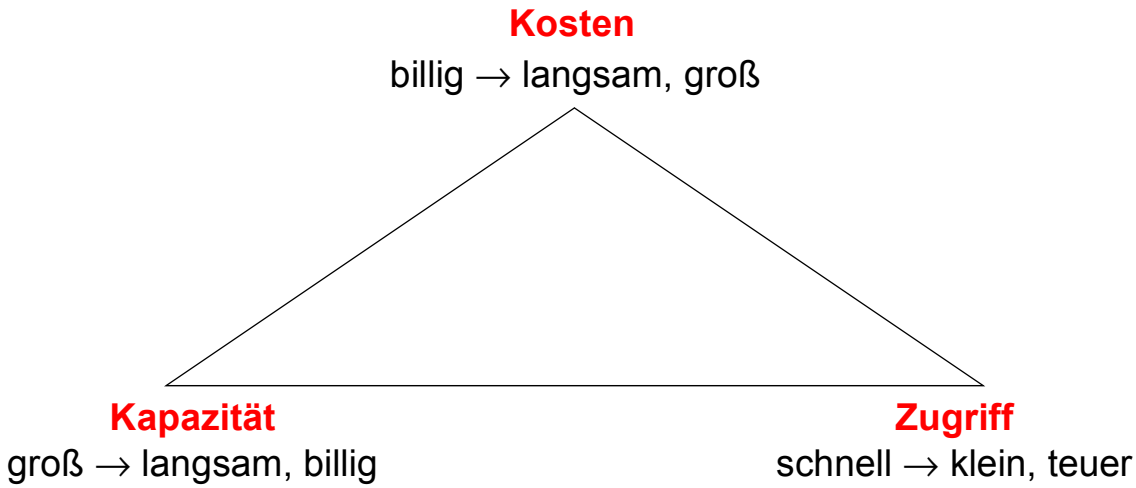
Vereinfachte E/A-Architektur :

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



E/A-Architektur von Informationssystemen (2)

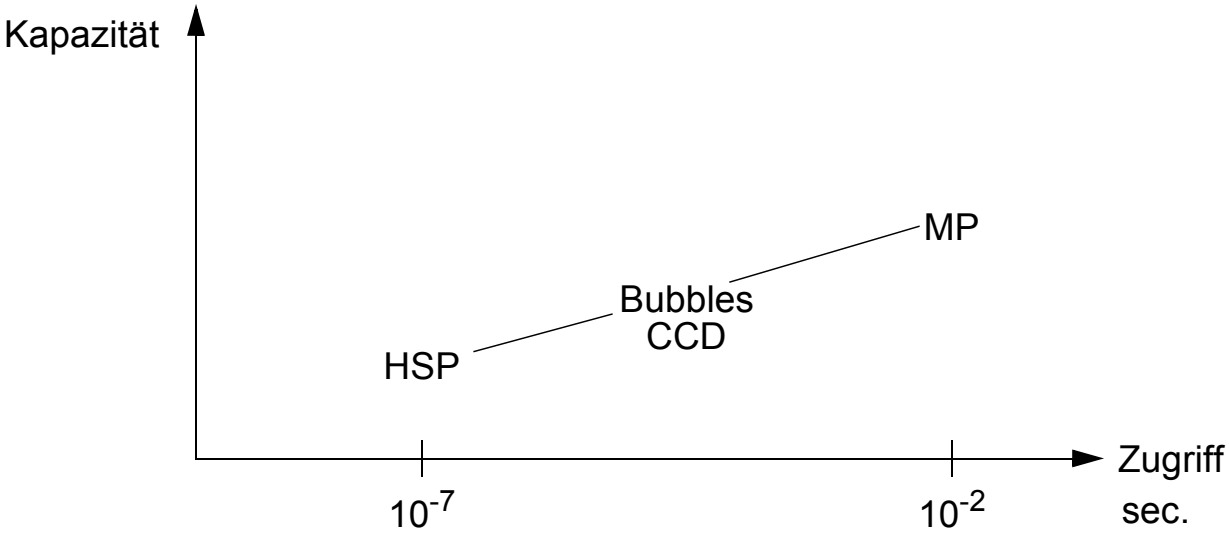
- **Ziel: groß – billig – schnell:**



- **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

- **Lückenfüller-Technologien**
  - Neue Speichertypen, die in die Lücke passen?
  - Blasenspeicher (*magnetic bubbles*), CCD usw. sind untauglich (~1980)



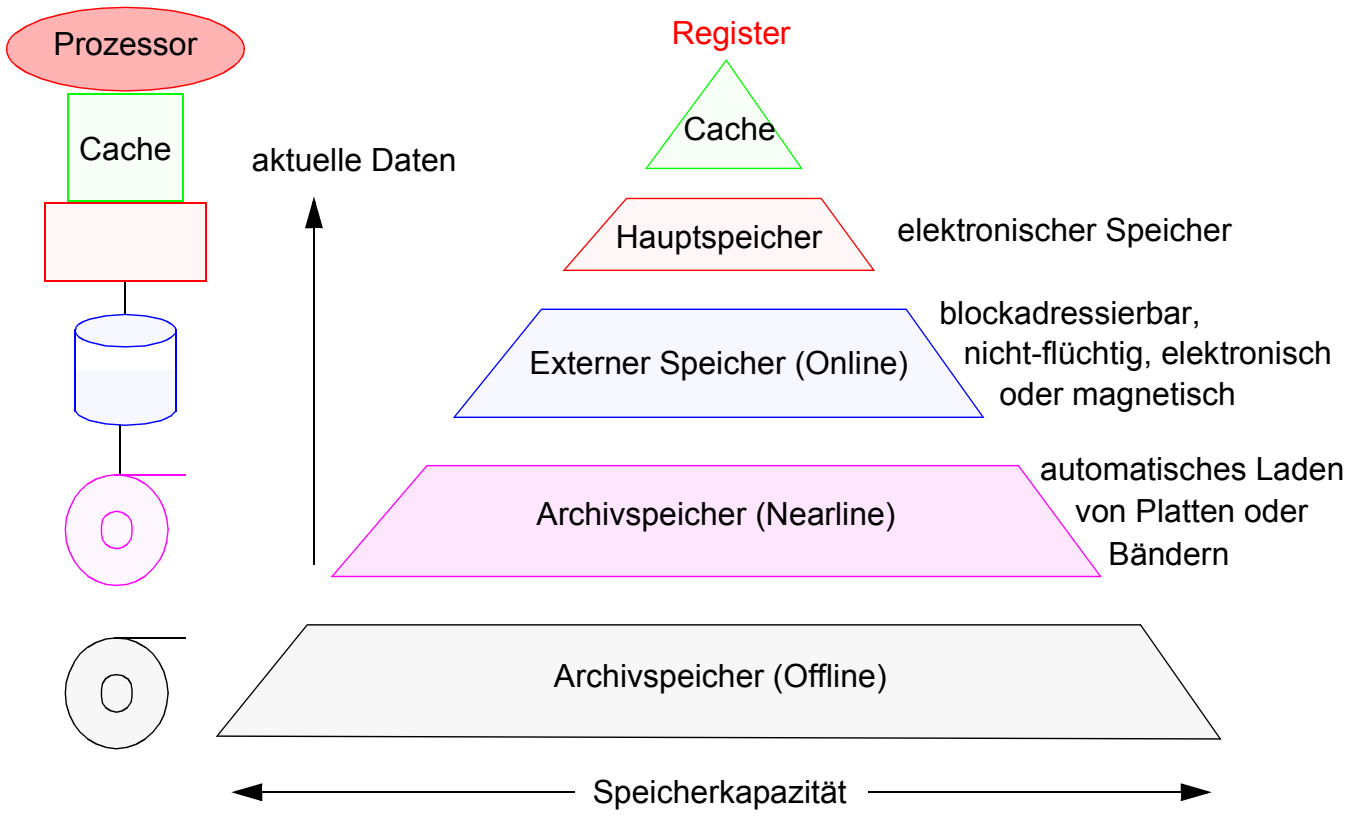
Speicherhierarchie

- **Wie kann die Zugriffslücke überbrückt werden?**
  - teilweise durch erweiterte HSP, SSD (*solid state disks*) und MP-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!

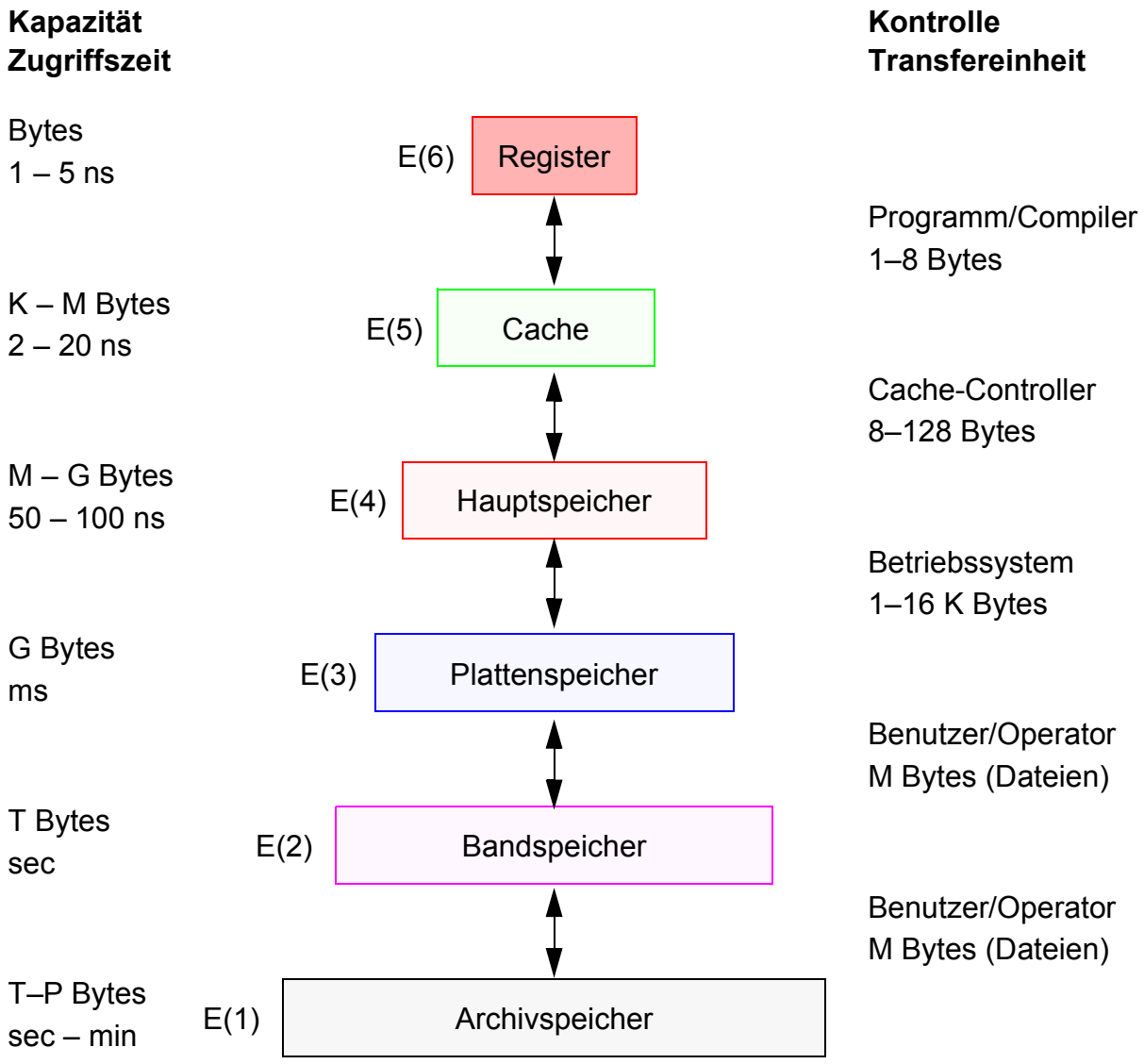
- **Rekursives Prinzip**

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



Speicherhierarchie (2)

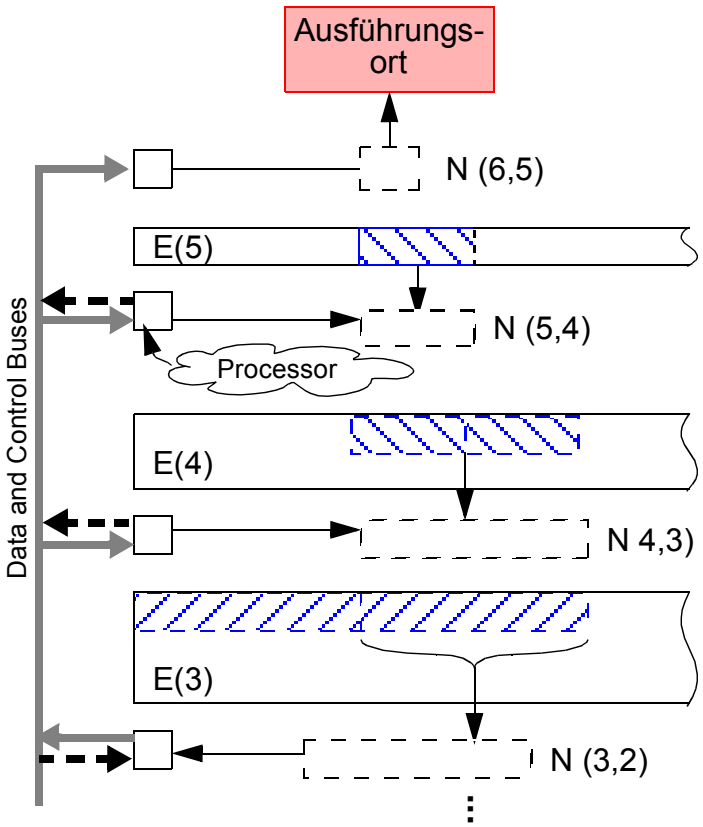
- **Charakteristische Merkmale einer Speicherhierarchie**

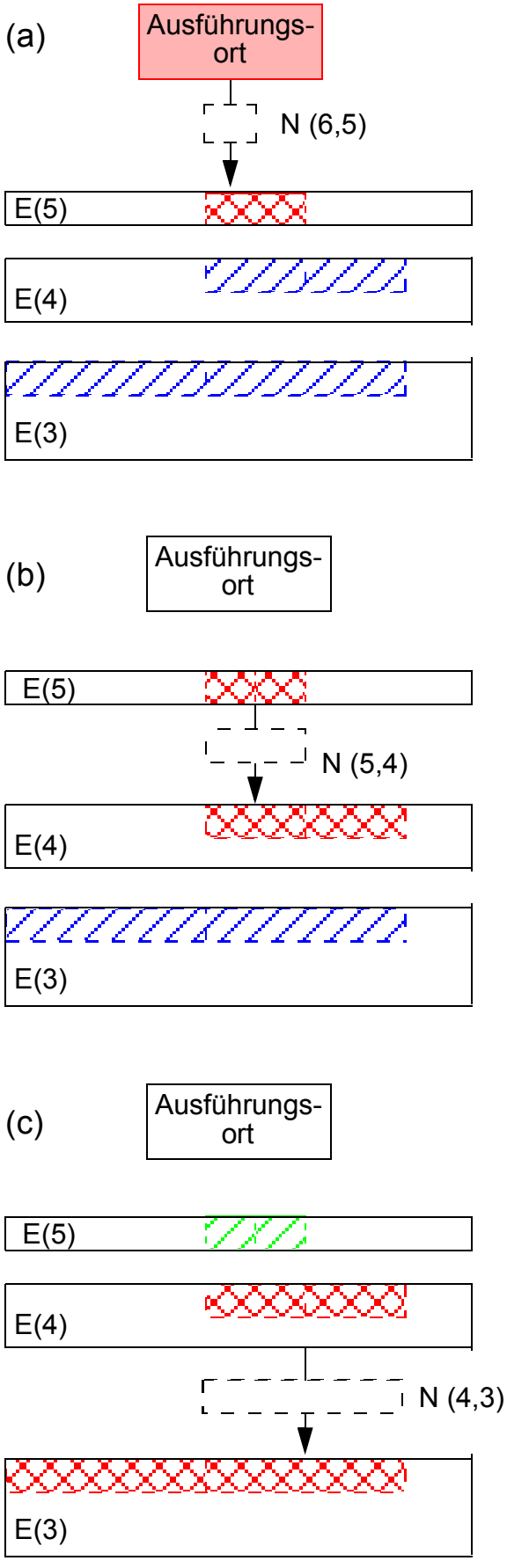


In naher Zukunft sind an verschiedenen Stellen der Speicherhierarchie wesentliche Verbesserungen bei Zugriffszeit und Transfergröße zu erwarten, welche ihre Arbeitsweise zwischen E(3) und E(6) stark beeinflussen könnte. IRAM (*Intelligent Random Access Memory*) zielt darauf ab, Mikroprozessor und DRAM-Speicher (*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:
Bedarfsgetrieben werden die Daten im schichtenspezifischen Granulat schichtweise in der Hierarchie „nach oben“ bewegt

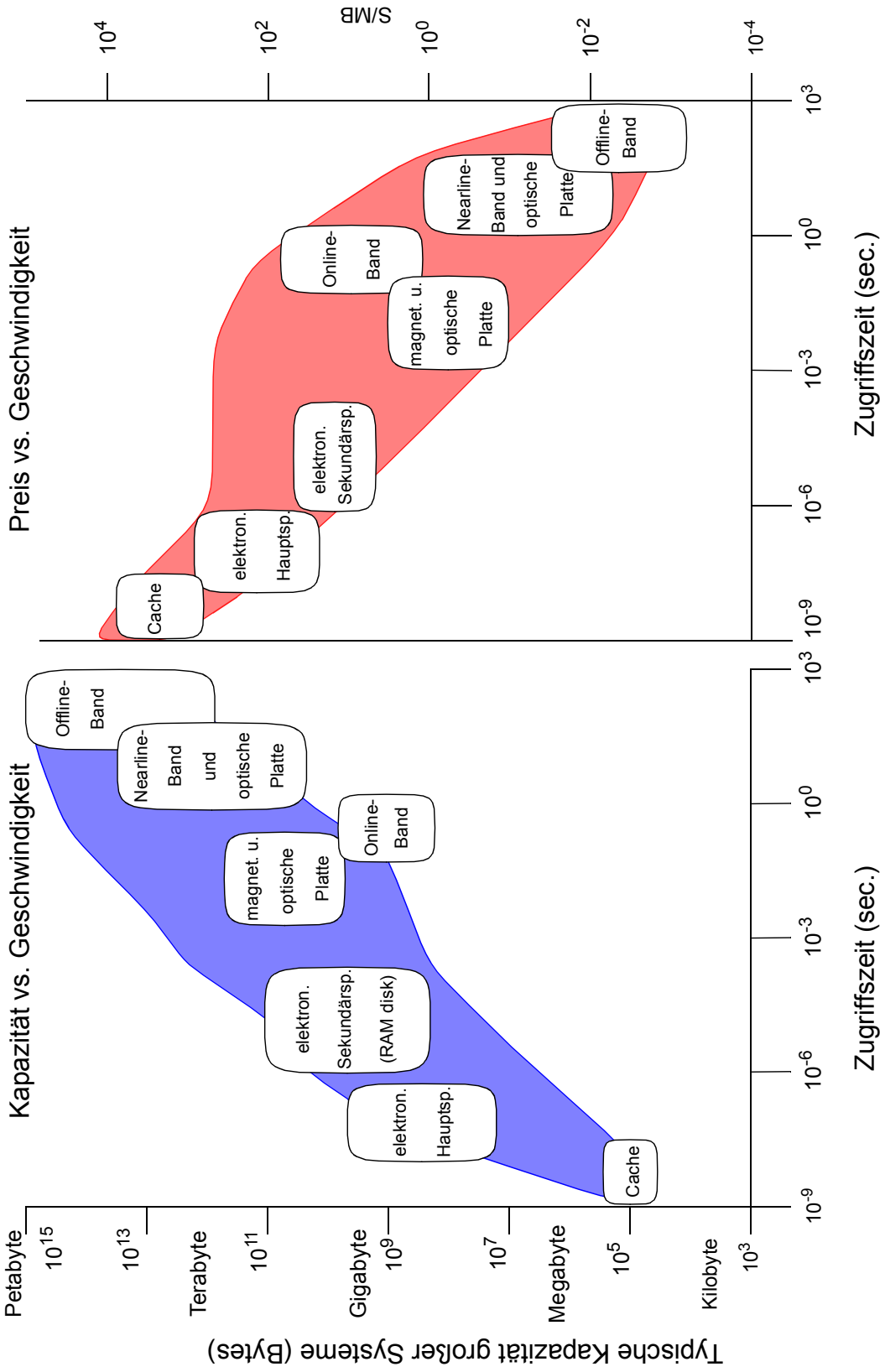
Write-through:
Sofort nach Änderung werden die Daten auch in der nächsttieferen Schicht geändert (FORCE)

Write-back:
Bei Ersetzung werden die geänderten Daten schichtweise „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	

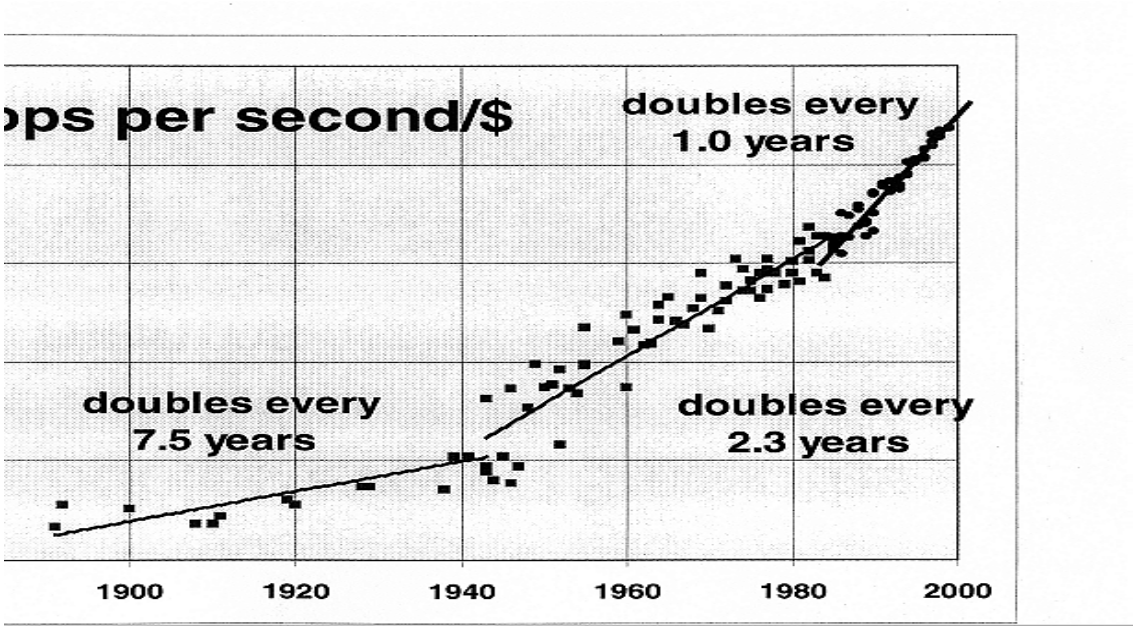
- **Welche Konsequenzen ergeben sich daraus?**
  - Caching
  - Replikation
  - Prefetching

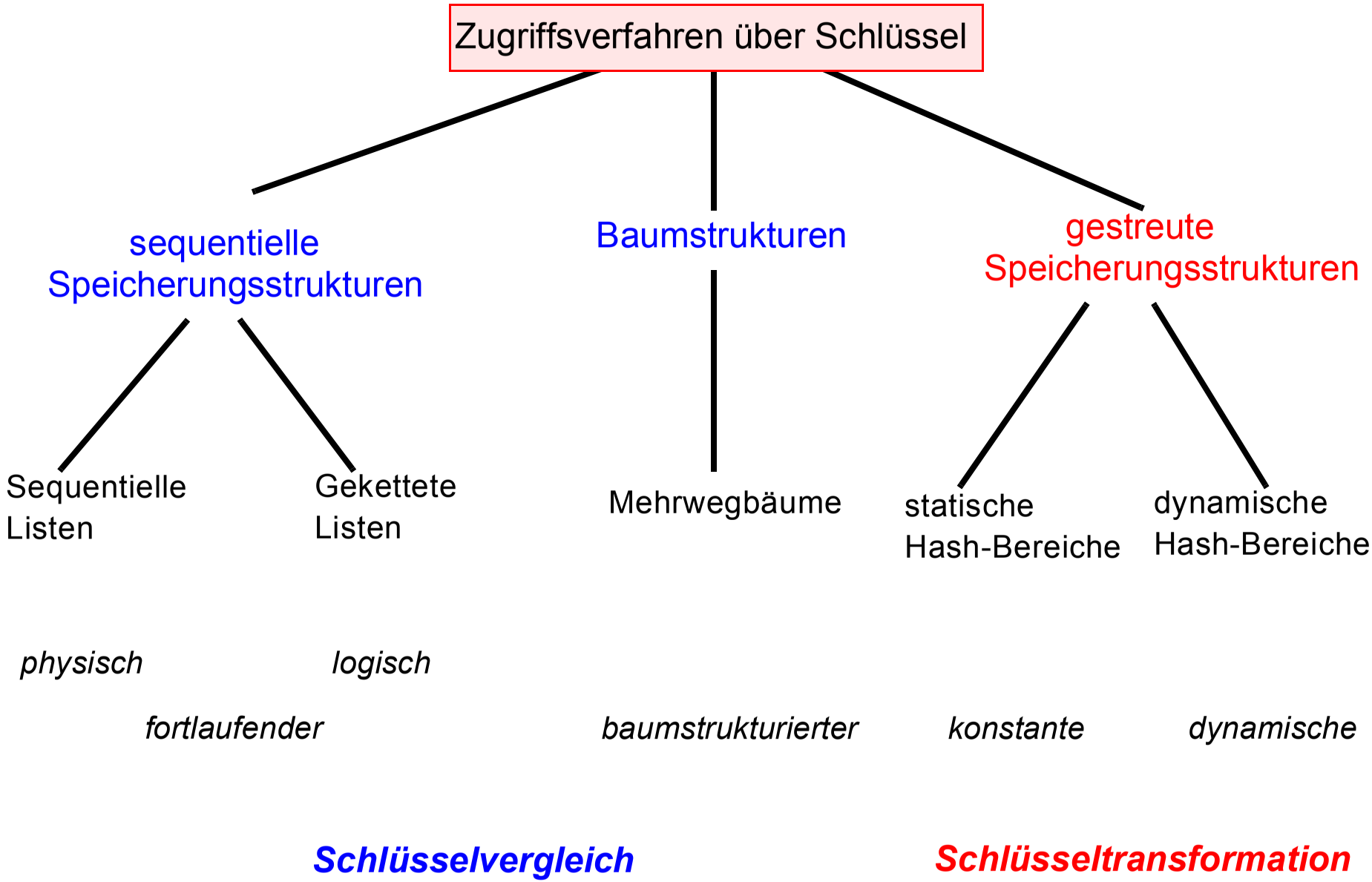
Prognosen

- **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?




Zugriffsverfahren über Schlüssel
sequentielle Speicherungsstrukturen
Baumstrukturen
gestreute Speicherungsstrukturen
Sequentielle Listen
Gekettete Listen
Mehrwegbäume
statische Hash-Bereiche
dynamische Hash-Bereiche
physisch
logisch
fortlaufender
baumstrukturierter
konstante
dynamische
Schlüsselvergleich
Schlüsseltransformation

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

  ➥ Cluster-Bildung kann pro Satztyp nur bezüglich eines Kriteriums erfolgen; sonst wird Redundanz eingeführt!

- **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 (welche?) wesentlich billiger:

      N_S =

  - **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	K97
K3	K9		K98
K4	K11	• • •	• • •
K6	K12		

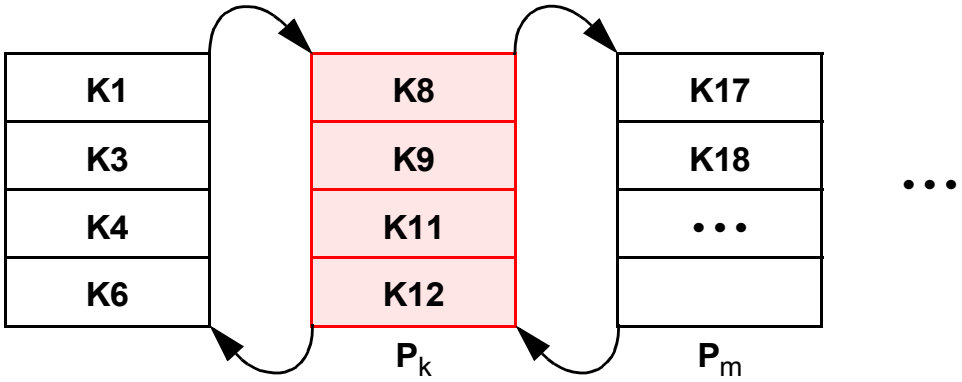
Dominoeffekt !?

Welche Kosten verursacht der Änderungsdienst?

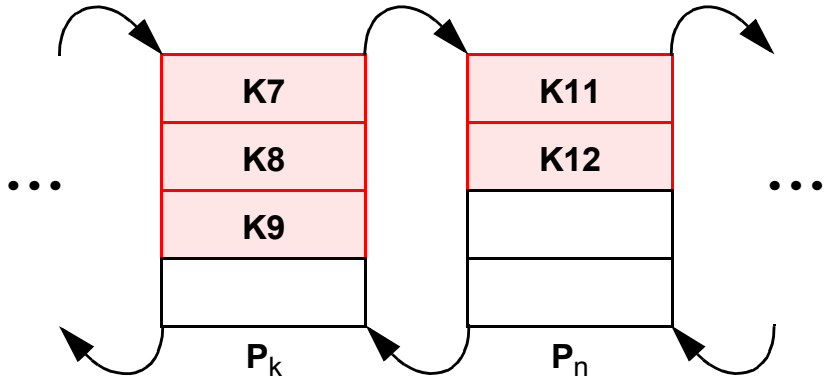
Verschiebekosten im Mittel:

N_S =

- **Deshalb Einsatz einer Split-Technik (Splitting):**



Welche Eigenschaft einer sequentiellen Liste geht verloren?



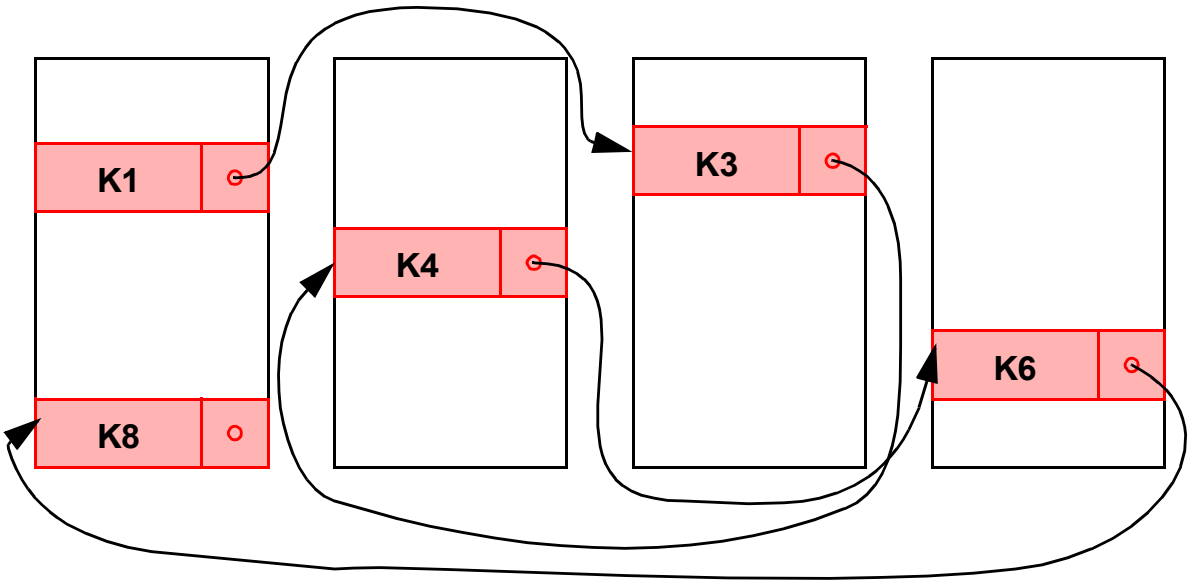
Speicherplatz-belegung ?

$N_S \leq$

➥ 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

    $N_S \leq$

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

Mehrwegbäume[1]

- **Binäre Suchbäume (SB) sind bekannt**
  - natürlicher binärer SB
  - AVL-Baum (höhenbalanciert)
  - gewichtsbalancierter SB

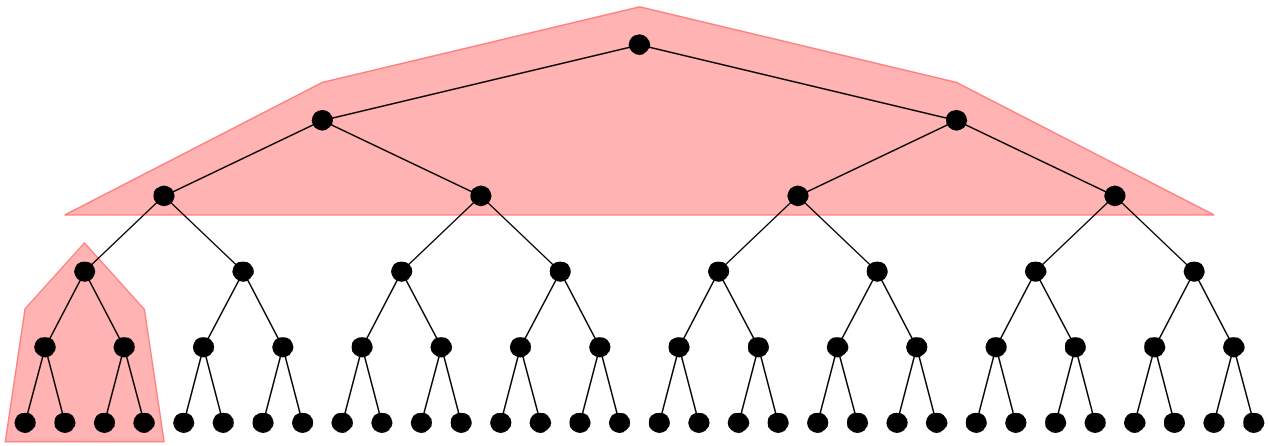
  ➥ Bei Abbildung auf Externspeicher ist die Höhe des SB entscheidend!
  - Höhe von binären SB

- **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!

1. R. Bayer, E. McCreight: Organization and Maintenance of Large Ordered Indices. Acta Inf. 1: 173-189 (1972)

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: “The Ubiquitous B-Tree”[2]

  ➥ dynamische Reorganisation durch Splitting 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, ...

---

2. Douglas Comer: The Ubiquitous B-Tree. ACM Comput. Surv. 11(2): 121-137 (1979)

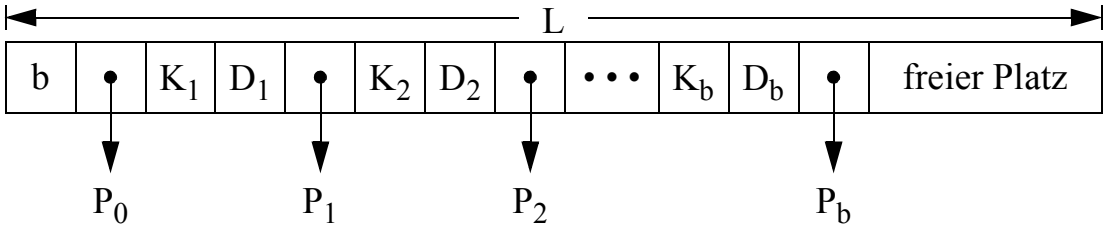
B-Bäume

- **Definition:**

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

- **Reformulierung der Definition:**

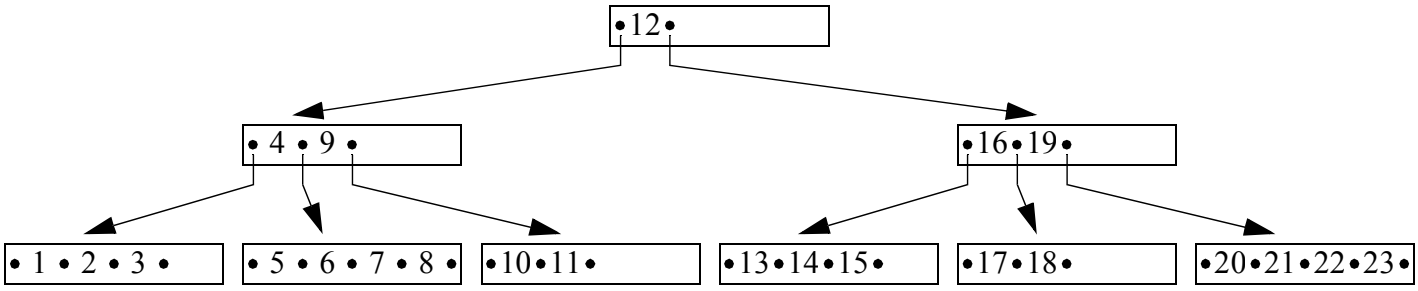
  (4) und (3). Eine Seite darf höchstens voll sein.

  (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 $\tau(4, 3)$.

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

$$\log_{2k+1}(n+1) \le h \le \log_{k+1}((n+1)/2) + 1 \quad \text{für } n \ge 1$$

und $h = 0$ 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 – ?

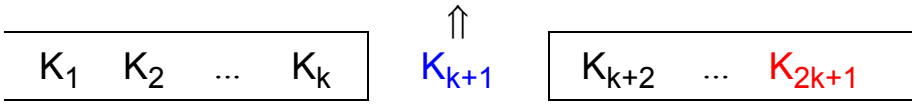
B-Bäume (3)

- **Einfügen in B-Bäumen**

  Wurzel läuft über:

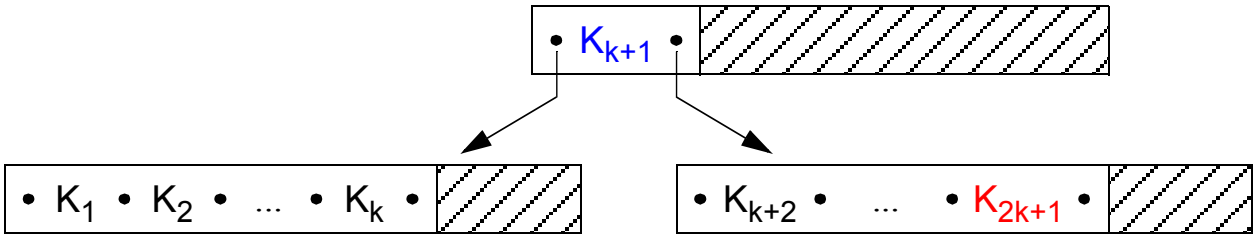
  Fundamentale Operation: Split-Vorgang

  1. Anforderung einer neuen Seite und
  2. Aufteilung der Schlüsselmenge nach folgendem Prinzip



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

  Hier wird eine neue Wurzel angelegt.

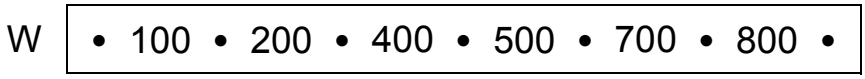


  Sobald ein Blatt überläuft, wird ein Split-Vorgang erzwungen, was eine Einfügung in den Vaterknoten (hier: Wurzel) impliziert. Wenn dieser überläuft, muss ein Split-Vorgang ausgeführt werden. Betrifft der Split-Vorgang die Wurzel, so wird das Anlegen einer neuen Wurzel erzwungen. Dadurch vergrößert sich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

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

B-Bäume – Einfügebeispiel

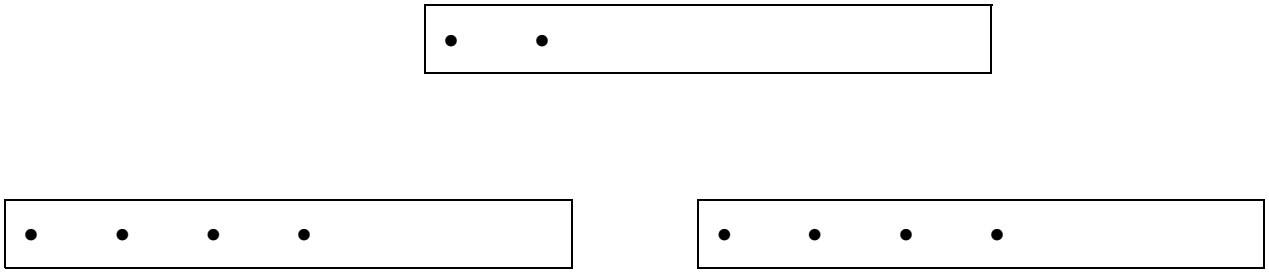
- **Einfügen in Wurzel (= Blatt)**



➥ Einfügen von 600 erzeugt Überlauf

- **Neuaufteilung von W**

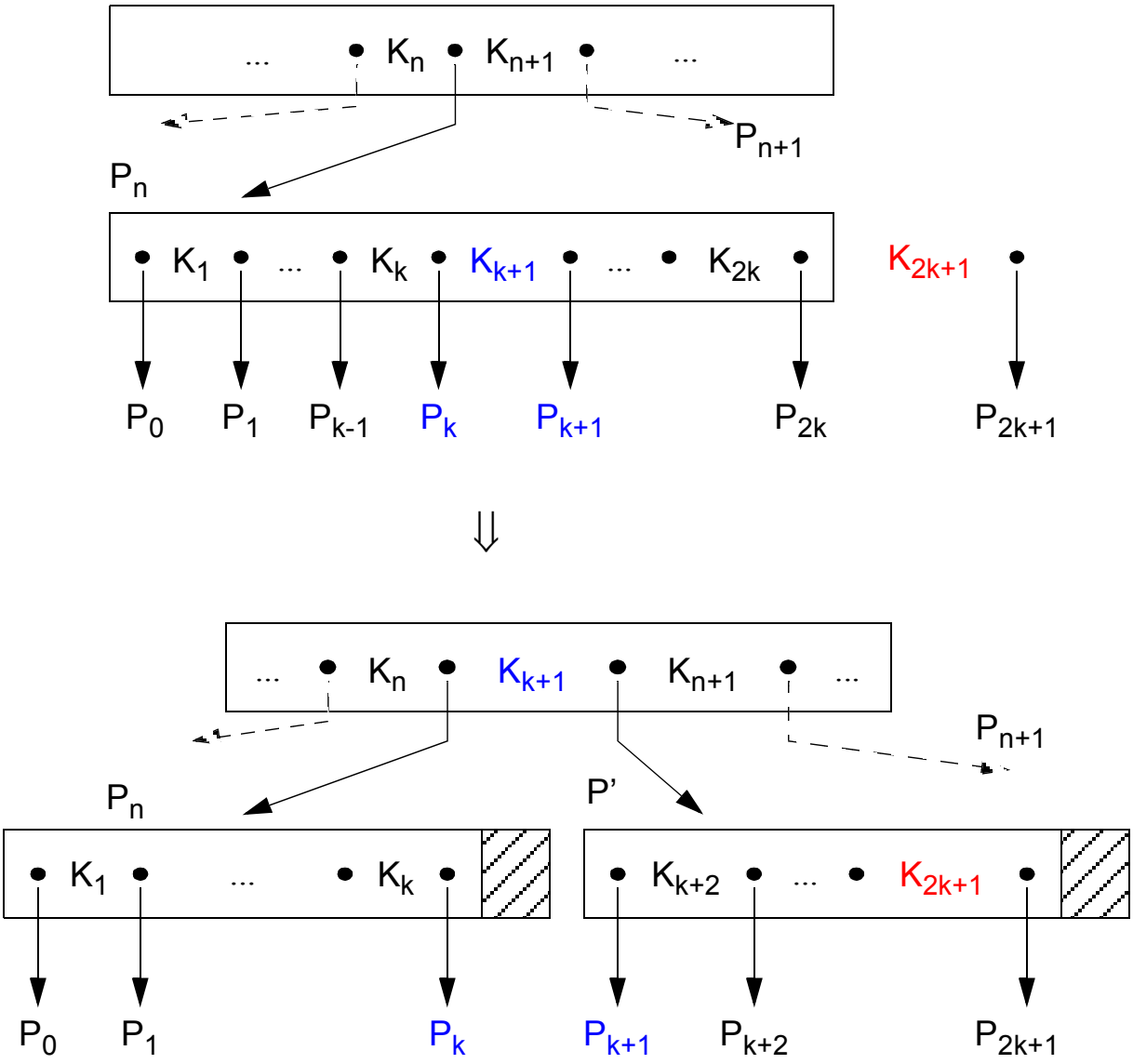
- **B-Baum nach Split-Vorgang**



B-Bäume (4)

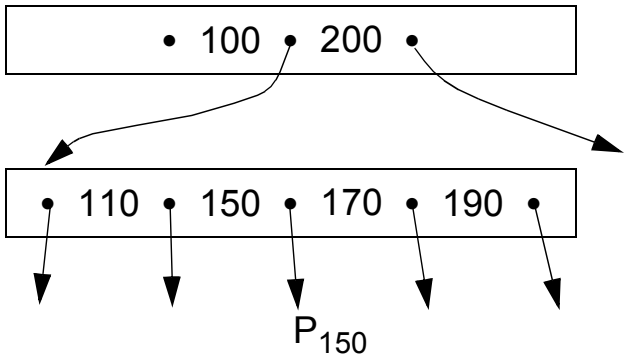
- **Einfügealgorithmus (ggf. rekursiv)**
  - Suche Einfügeposition
  - Wenn Platz vorhanden ist, speichere Element, sonst schaffe Platz durch Split-Vorgang und füge ein.

- **Split-Vorgang als allgemeines Wartungsprinzip**



B-Bäume – Split-Beispiel

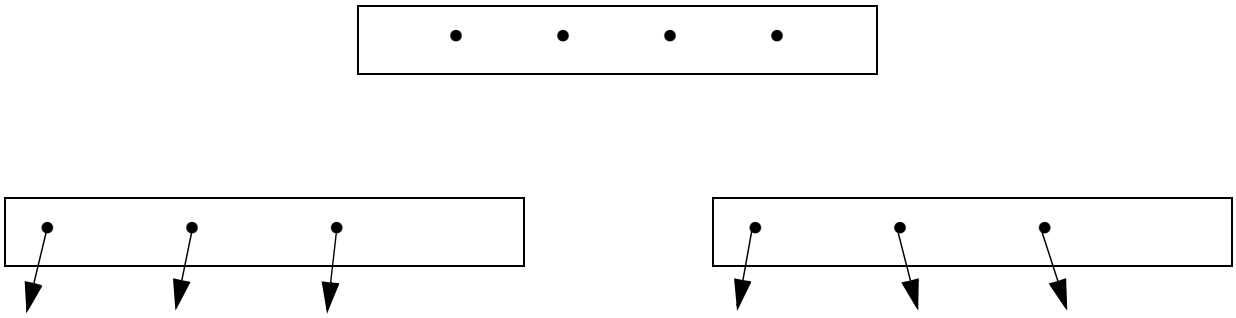
- **Split-Vorgang – rekursives Schema**



➥ Einfügen von 130 mit P_{130}

- **Neuaufteilung des kritischen Knotens**

- **Baumausschnitt nach Split-Vorgang**

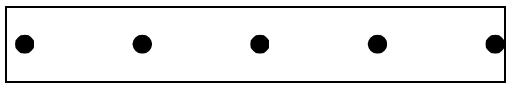


B-Bäume (5)

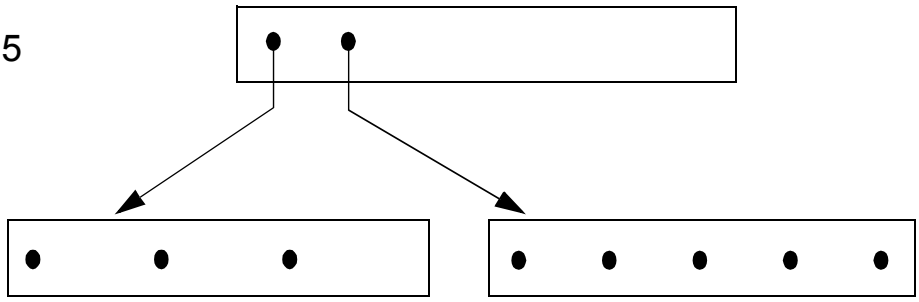
- **Aufbau eines B-Baumes der Klasse τ(2, h)**

  Einfügereihenfolge:

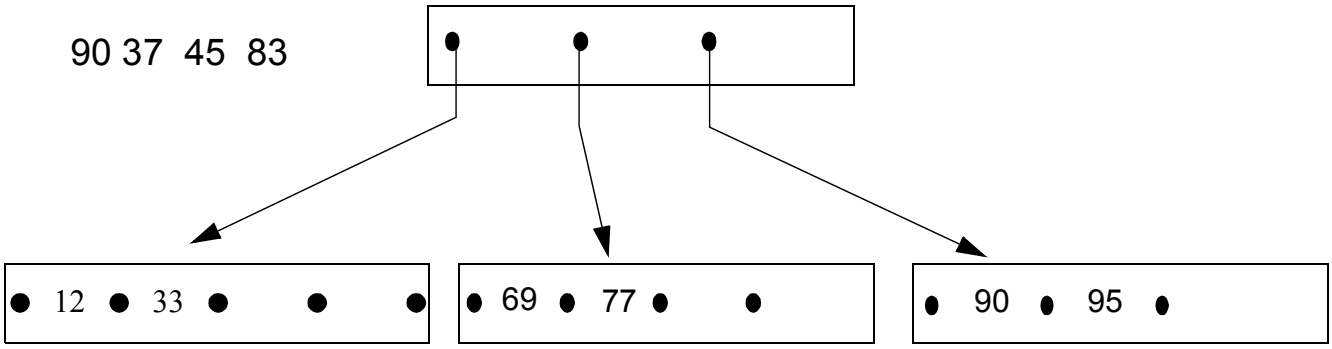
77 12 48 69



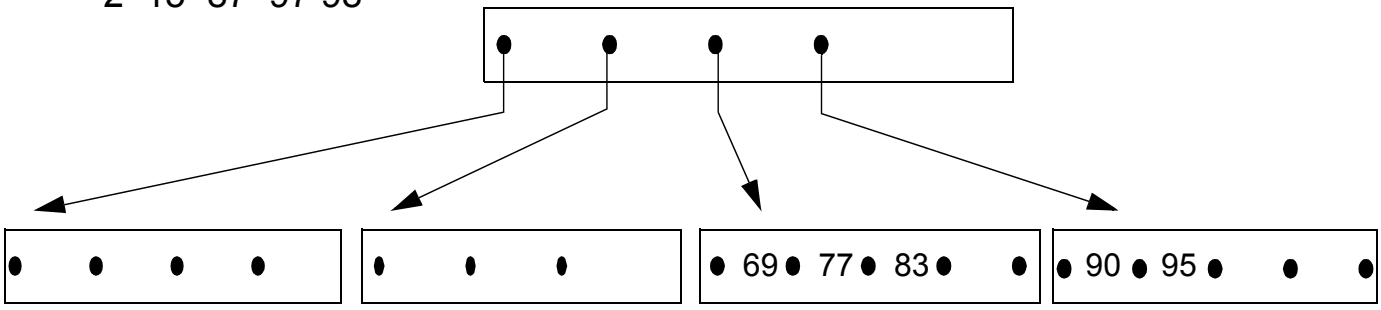
33 89 95



90 37 45 83



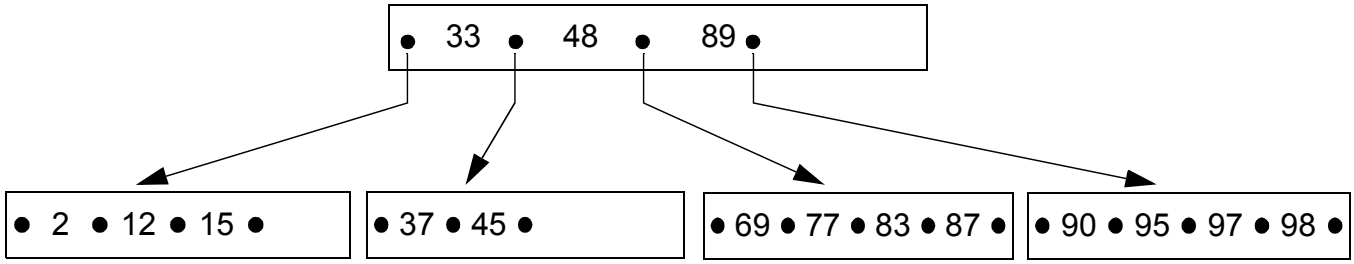
2 15 87 97 98



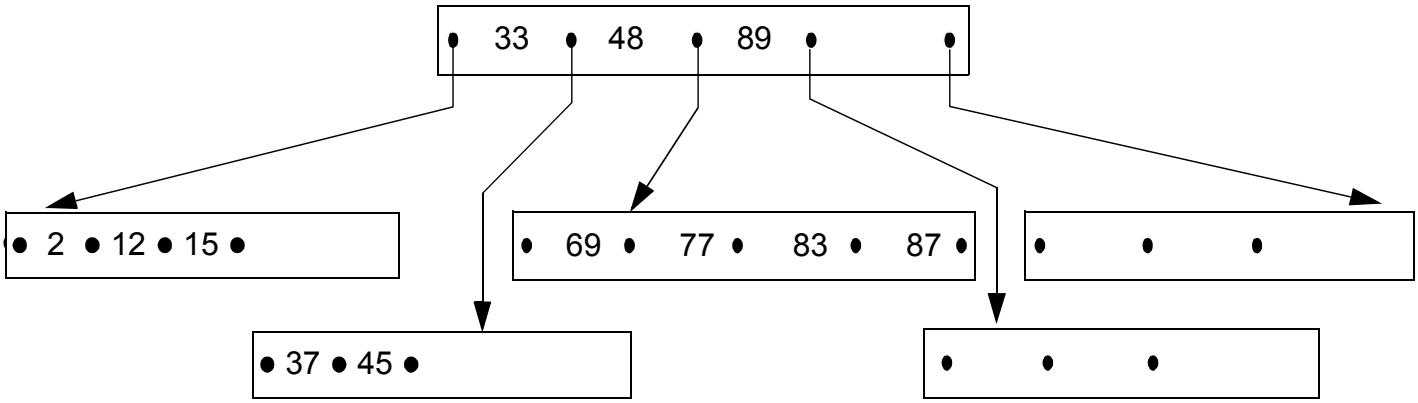
B-Bäume (6)

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

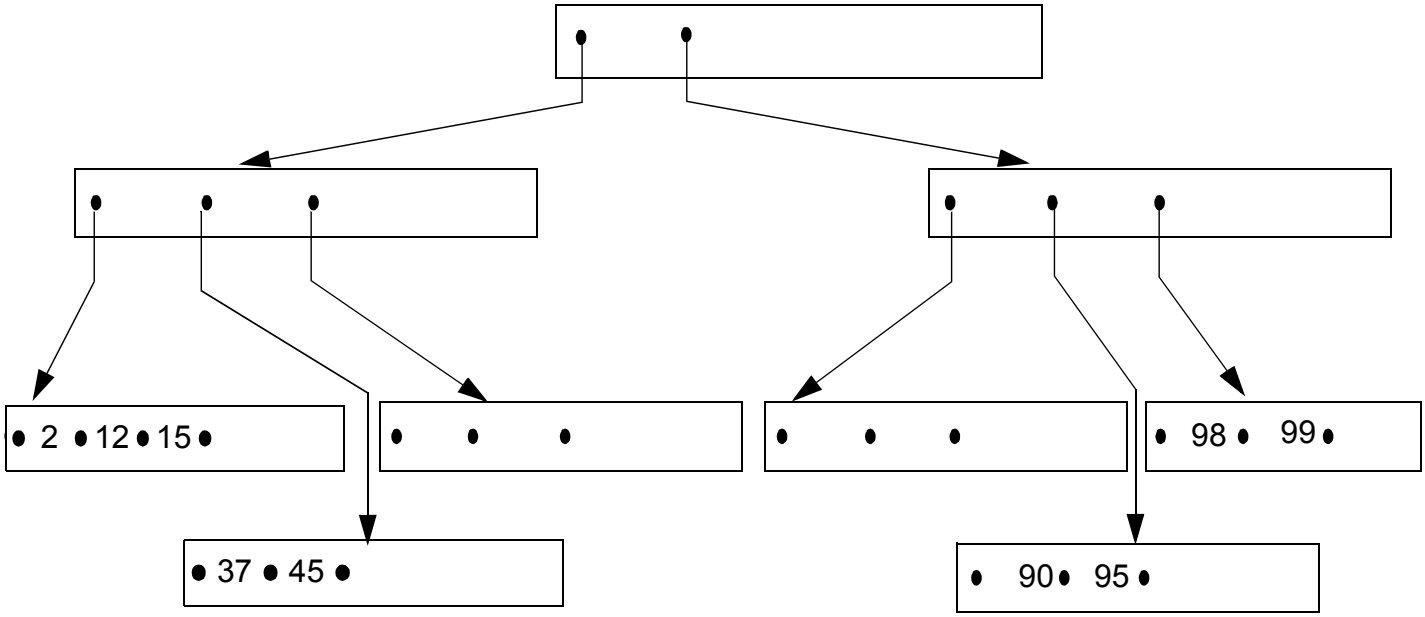
  Einfügereihenfolge:



99



50



B-Bäume (7)

- **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)

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

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

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

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

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

B-Bäume (8)

- **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.

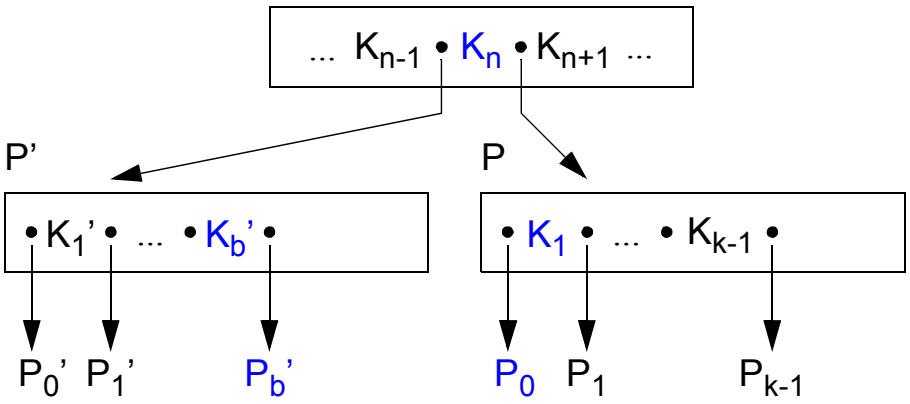
B-Bäume (9)

- **Löschen in B-Bäumen**

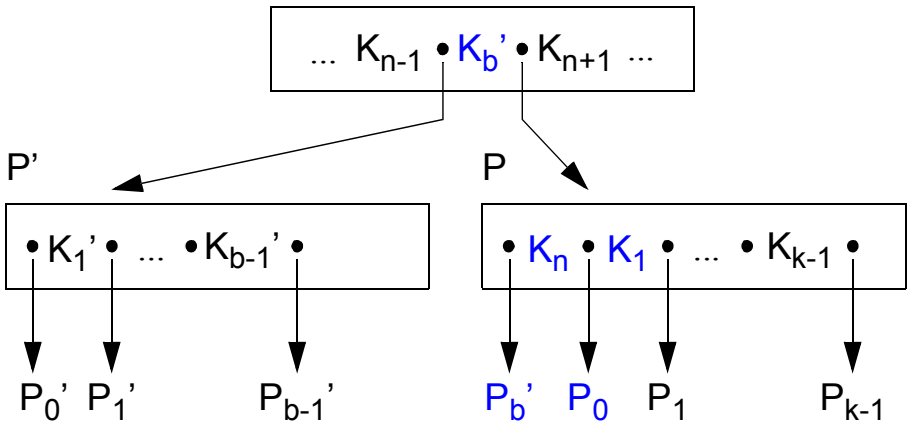
  Die B-Baum-Eigenschaft muss wiederhergestellt werden, wenn die Anzahl der Elemente in einem Knoten kleiner als k wird.

  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

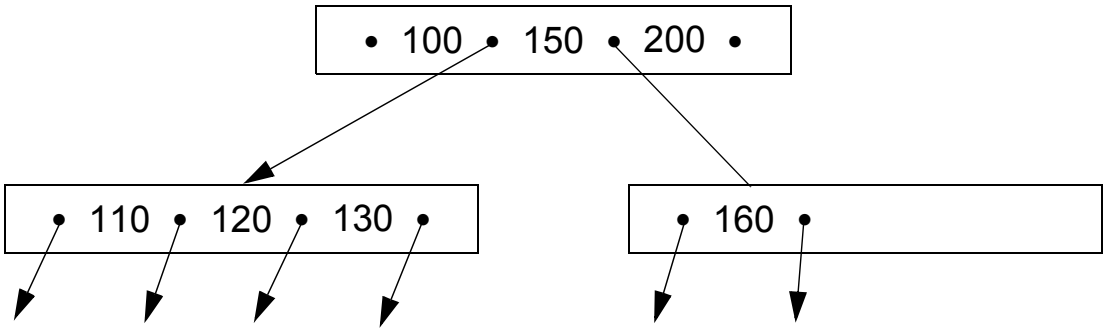


Ausgleich ⇓



B-Bäume – Löschbeispiele

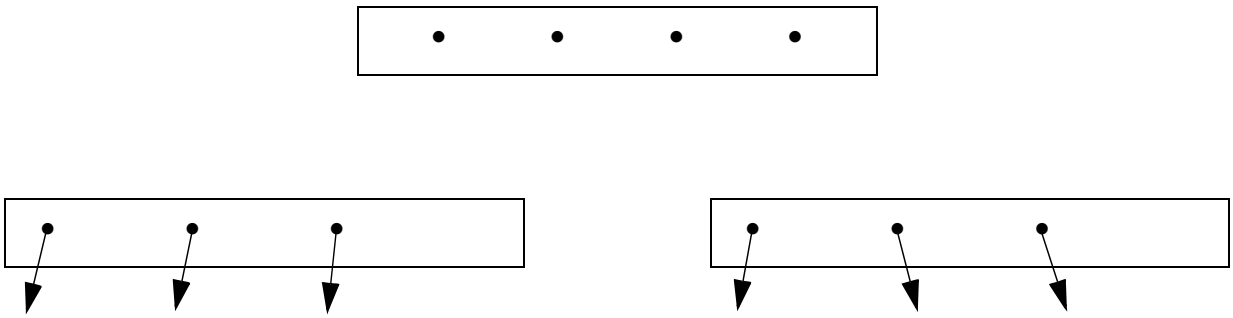
- **Löschprinzip: Ausgleich**



➥ $b_1 > k$ und $b_2 = k - 1$

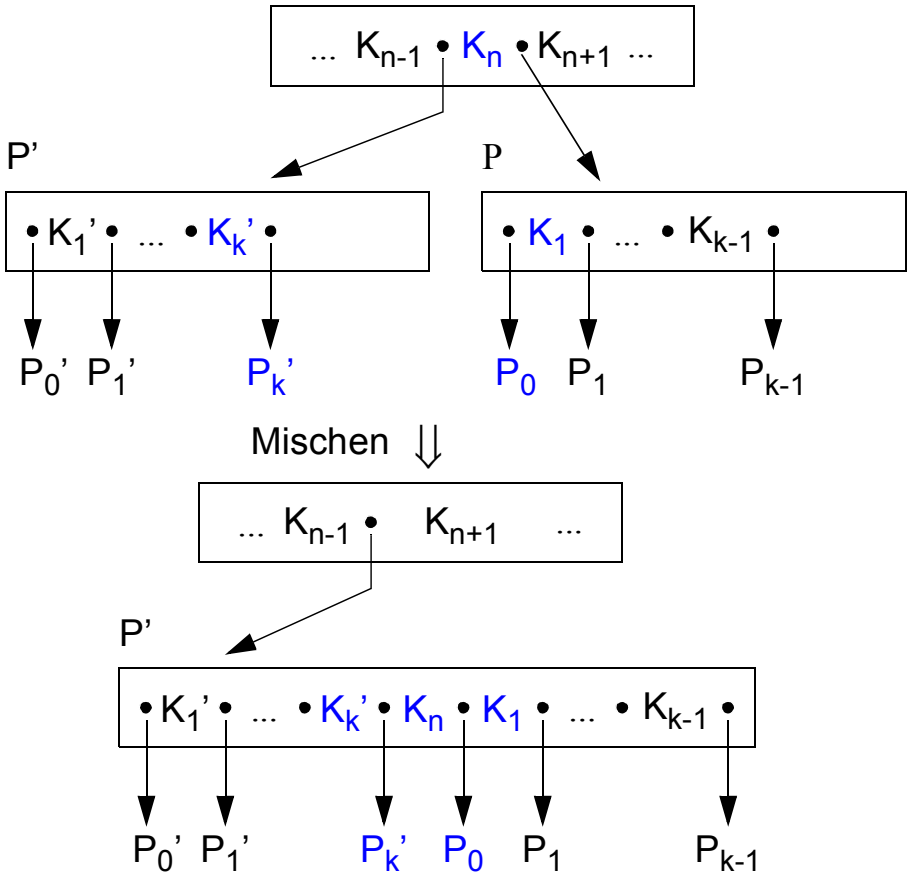
- **Neuaufteilung durch Ausgleich**

- **Baumausschnitt nach Ausgleich**



B-Bäume (10)

(2) Maßnahme: Mischen von Seiten



- **Löschalgorithmus**

(1) Löschen in Blattseite

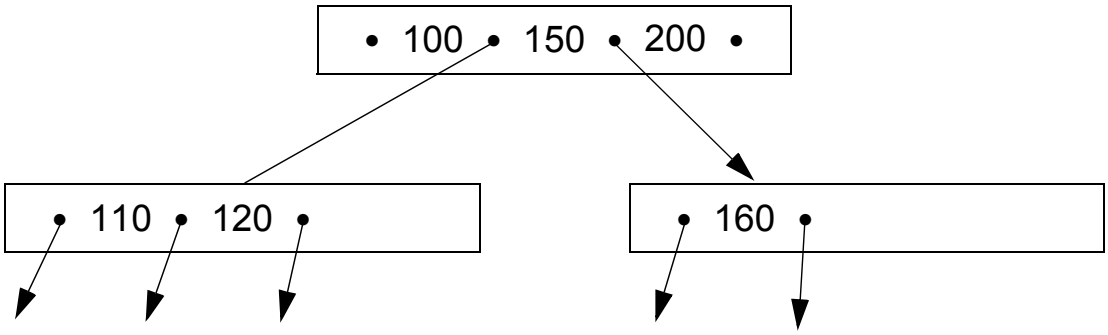
- Suche x in Seite P
- Entferne x in P und wenn
  a) b ≥ k in P: tue nichts
  b) b = k–1 in P und b > k in P': gleiche Unterlauf über P' aus
  c) b = k–1 in P und b = k in P': mische P und P'.

(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)

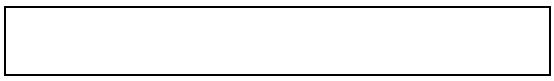
B-Bäume – Löschbeispiele

- **Löschprinzip: Mischen**

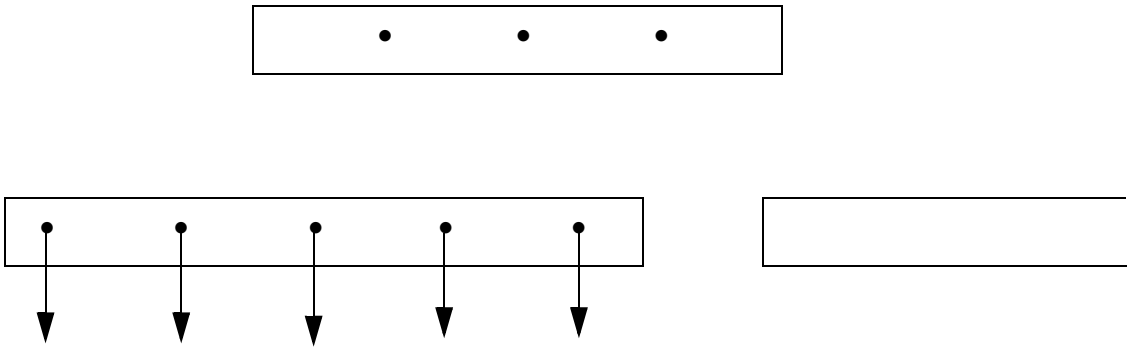


➥ $b_1 = k$ und $b_2 = k - 1$

- **Neuaufteilung durch Mischen**

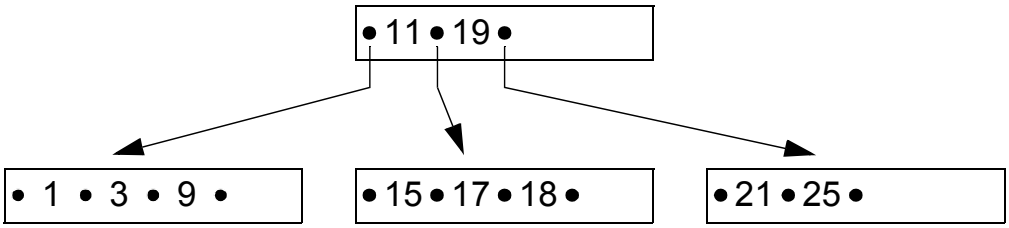


- **Baumausschnitt nach Mischen**

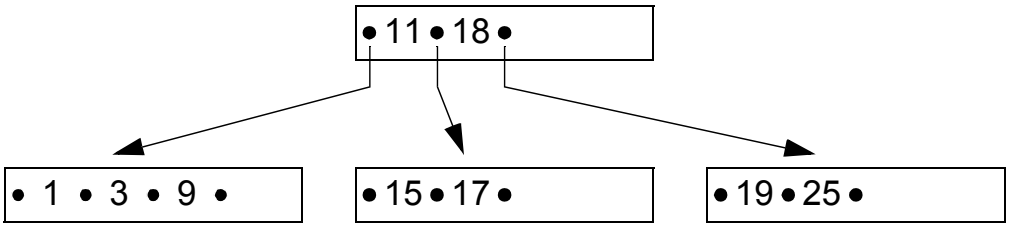


B-Bäume (11)

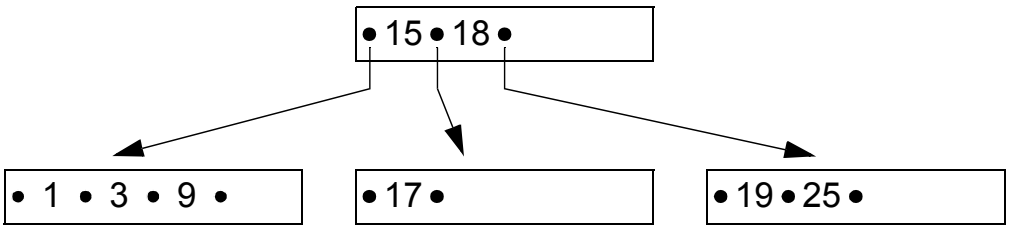
- **Beispiel: Löschen**



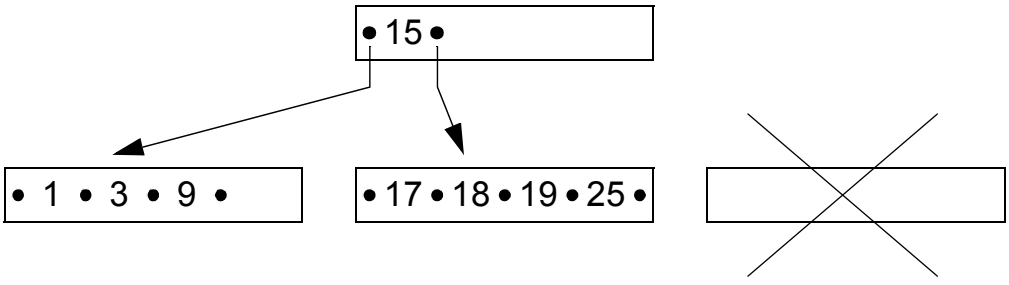
Lösche 21:



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):

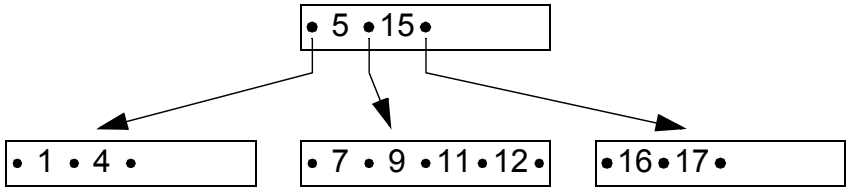
$$f_{avg} \leq f_1 + f_2 + f_3 < h + 1 + \frac{1}{k}$$

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

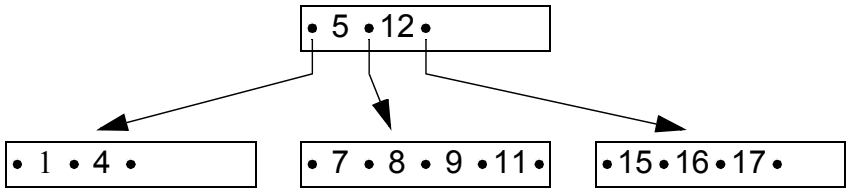
B-Bäume – Überlaufbehandlung

- **Welcher Belegungsgrad β_{avg} des B-Baums wird erzielt**
  - bei einfacher Überlaufbehandlung (Split-Faktor m = 1)?
  - beim Einfügen einer sortierten Schlüsselfolge?

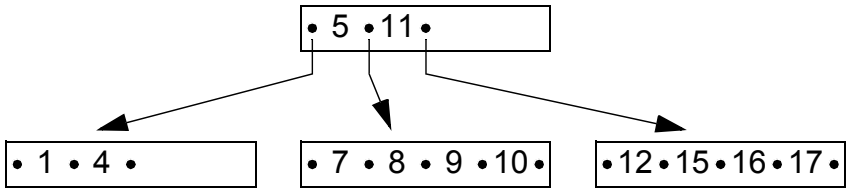
- **Einfügen in den B-Baum bei doppeltem Überlauf ($\tau(2,2)$)**



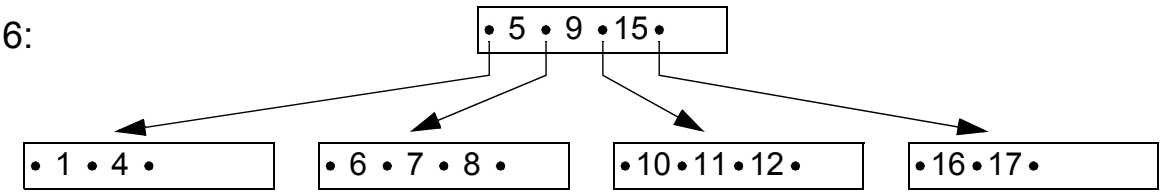
Einfüge 8:



Einfüge 10:



Einfüge 6:



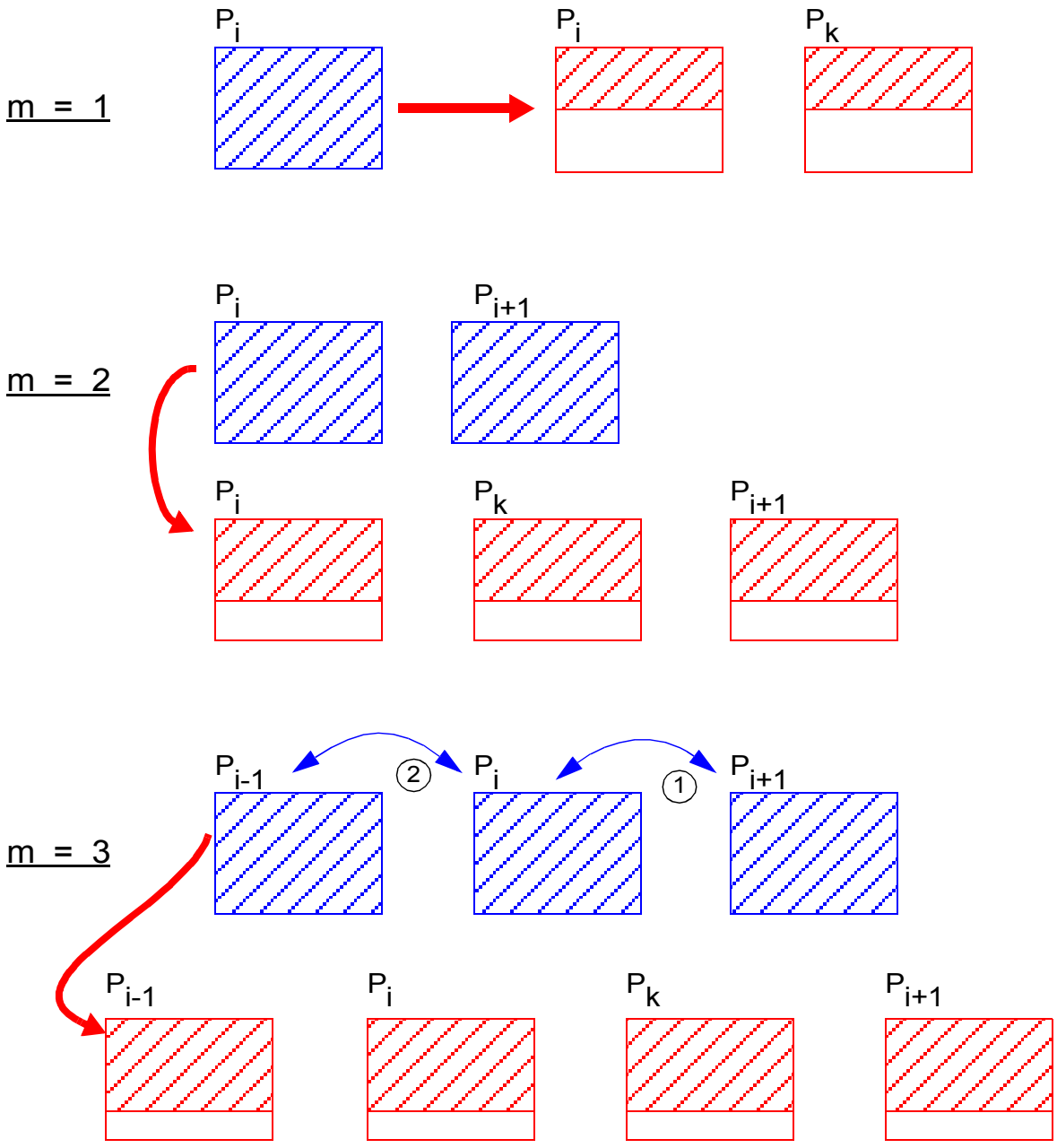
➥ Split-Faktor **m = 2**

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

B-Bäume – Überlaufbehandlung (2)

- **Verbesserung des Belegungsgrades**

  ➥ verallgemeinerte Überlaufbehandlung



  ➥ m > 1: verbesserte Speicherplatzbelegung, dafür höhere Einfügekosten

B-Bäume – Überlaufbehandlung (3)

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

Split-Faktor	Belegung β_{min}	Belegung β_{avg}	Belegung β_{max}
1	1/2 = 50%	**$\ln 2 \approx 69\%$**	1
2	2/3 = 66%	$2 \cdot \ln (3/2) \approx 81\%$	1
3	3/4 = 75%	$3 \cdot \ln (4/3) \approx 86\%$	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.

- **Definition[3]:**
  Seien k, k* und h* ganze Zahlen, $h^* \geq 0$, $k, k^* > 0$. Ein B*-Baum 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.

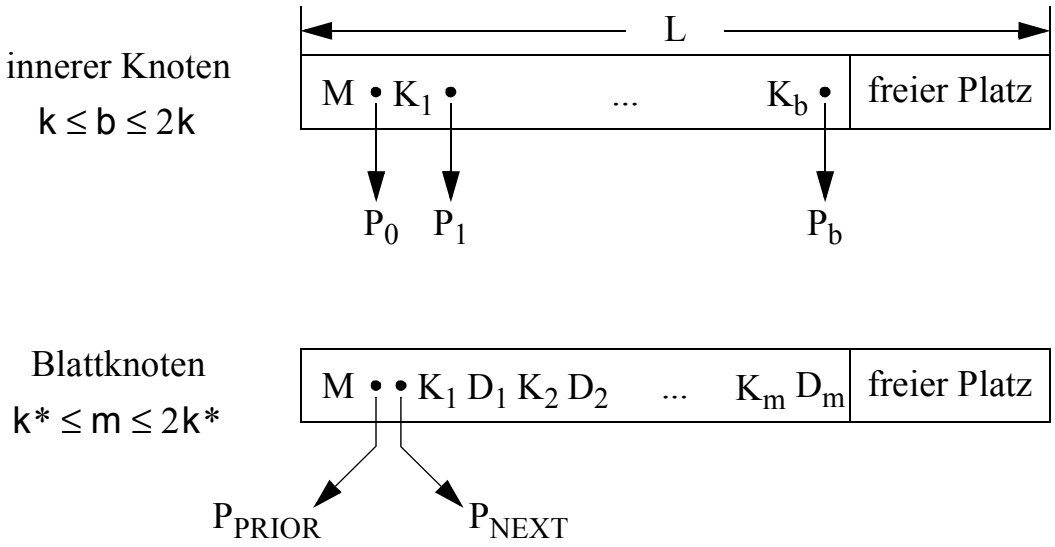
  (4) Jeder Blattknoten mit Ausnahme der Wurzel als Blatt hat mindestens k* und höchstens 2k* Einträge.

---

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

B*-Bäume (2)

- **Unterscheidung von zwei Knotenformaten:**

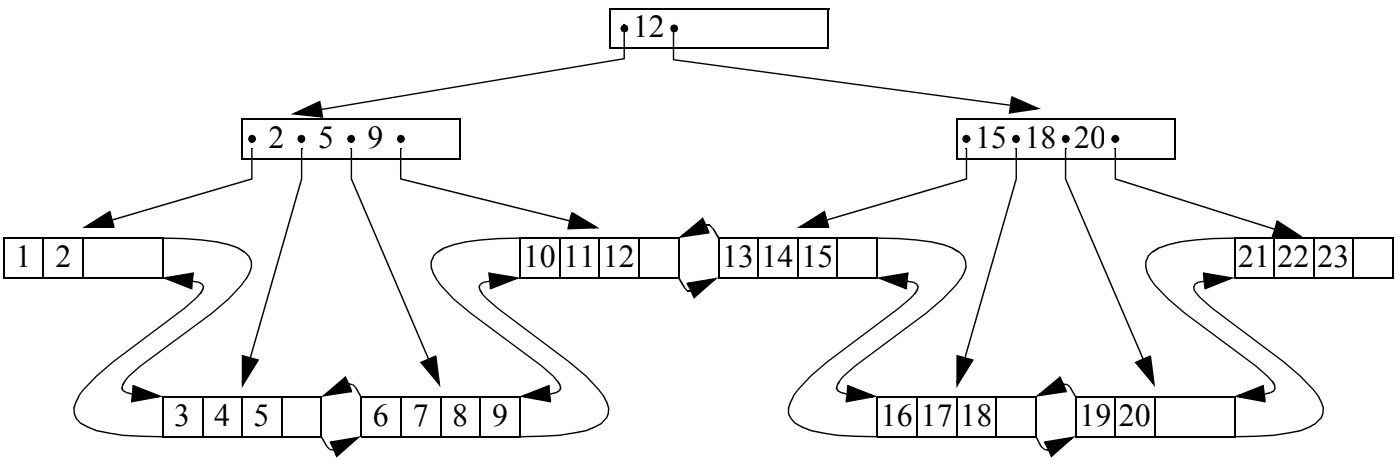


Das Feld M enthalte eine Kennung des Seitentyps sowie die Zahl der aktuellen Einträge. Da die Seiten eine feste Länge L besitzen, lässt sich aufgrund der obigen Formate k und k* bestimmen:

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

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

- **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**

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

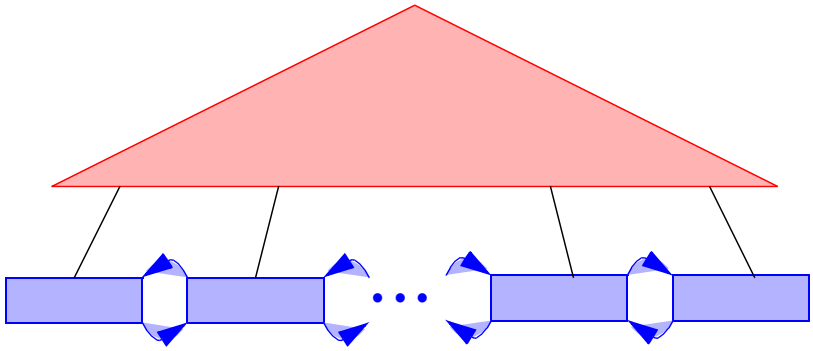
- **Blatt**

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

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.



Indexteil:
B-Baum von Schlüsseln

sequentielle sortierte
Datei der Blätter

- **Grundoperationen beim B*-Baum**

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

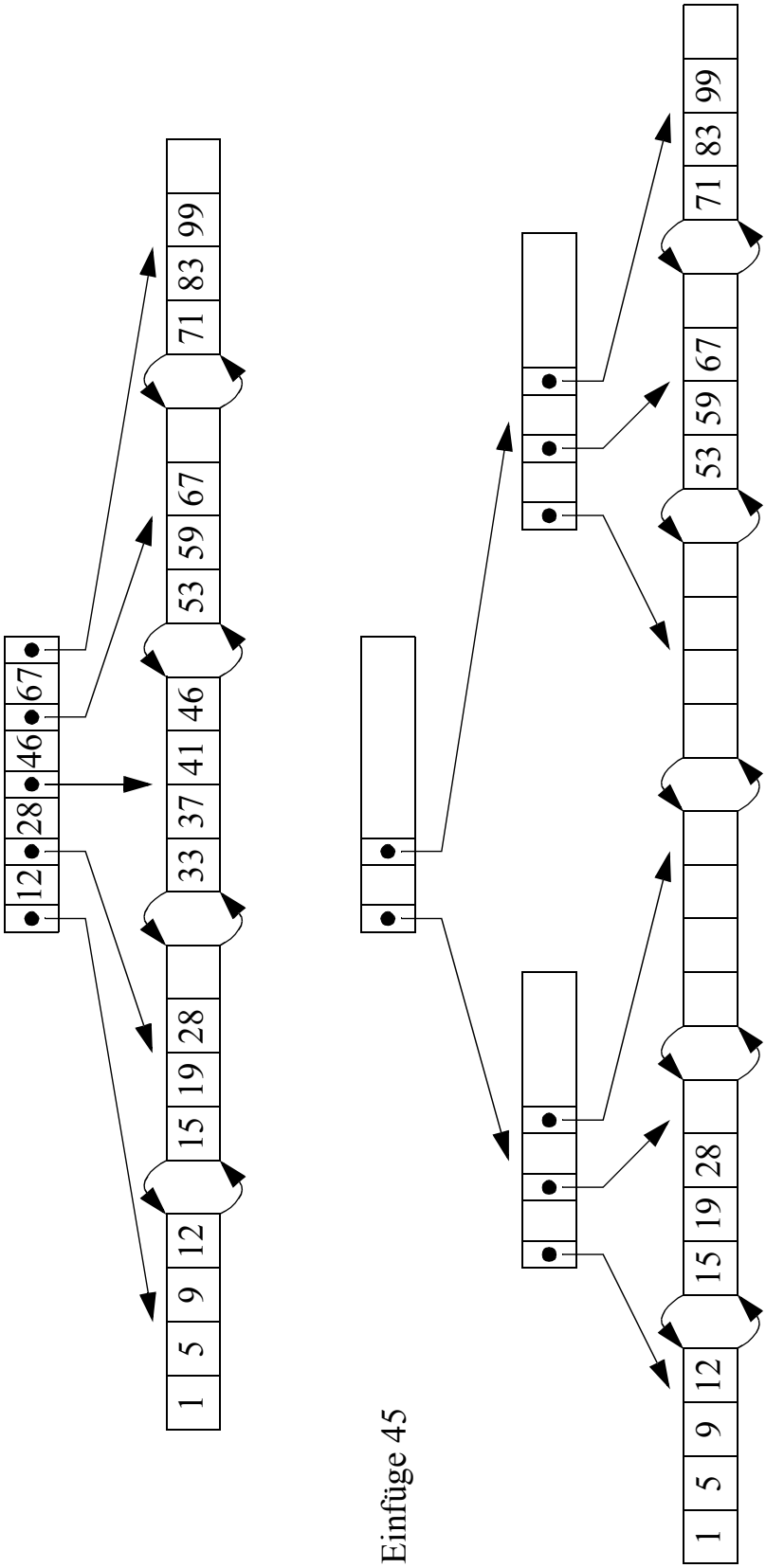
  (2) Sequentielle Suche:
  Sie erfolgt nach Aufsuchen des Linksaußen der Struktur unter Ausnutzung der Verkettung der Blattseiten. Es sind zwar ggf. mehr Blätter als beim B-Baum zu verarbeiten, doch da nur h*–1 innere Knoten aufzusuchen sind, wird die sequentielle Suche ebenfalls effizienter ablaufen.

  (3) Einfügen:
  Es ist von der Durchführung und vom Leistungsverhalten her dem Einfügen in einen B-Baum sehr ähnlich. Bei inneren Knoten wird das Splitting 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.

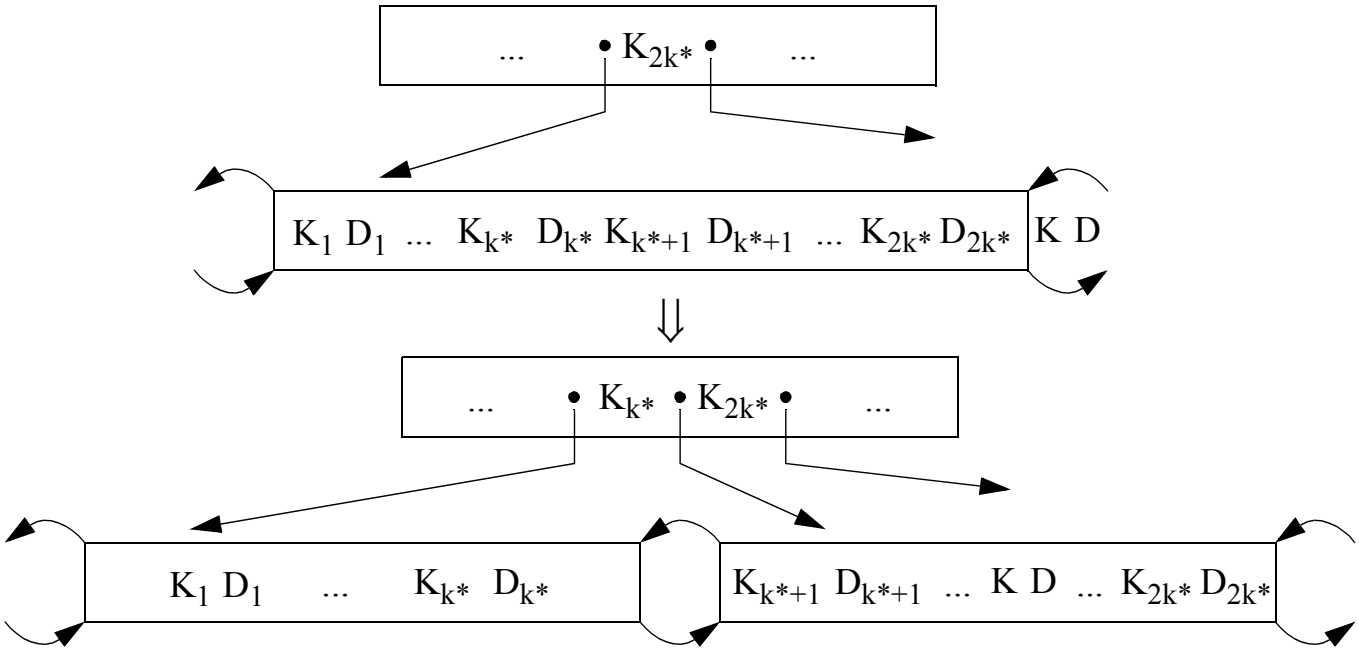
B*-Bäume (4)

Einfügen im B*-Baum

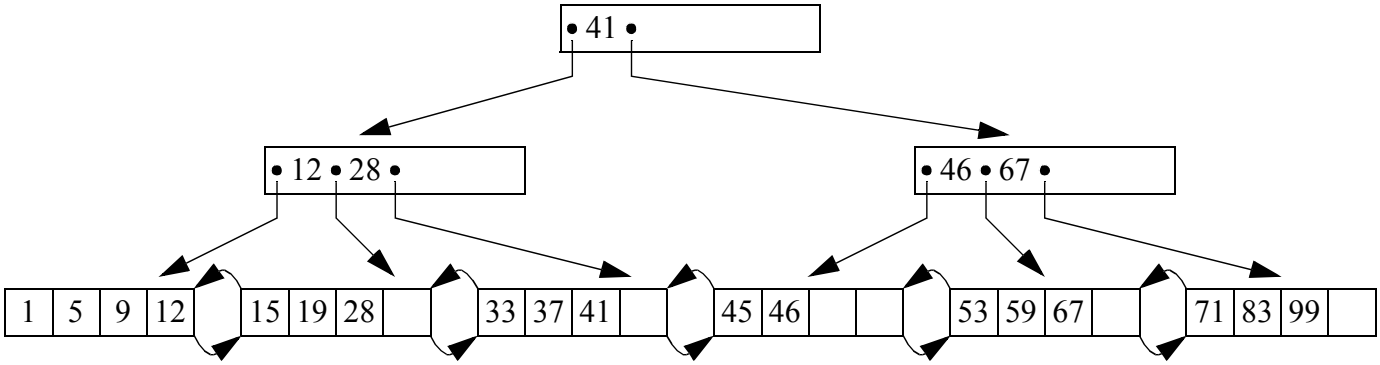


B*-Bäume (5)

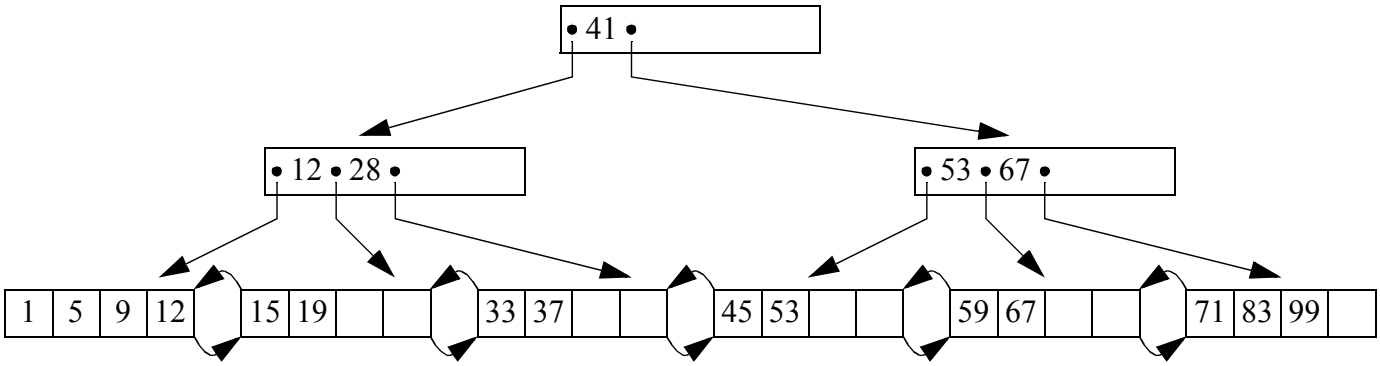
- **Schema für Split-Vorgang:**



- **Löschen im B*-Baum: τ(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	komprimierter 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	-

➥ Durchschnittliche komprimierte Schlüssellänge ca. 1.3 – 1.8

B*-Bäume – Schlüsselkomprimierung

- **Präfix-Komprimierung**

- **Präfix-Suffix-Komprimierung**

B*-Bäume (7)

- **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?

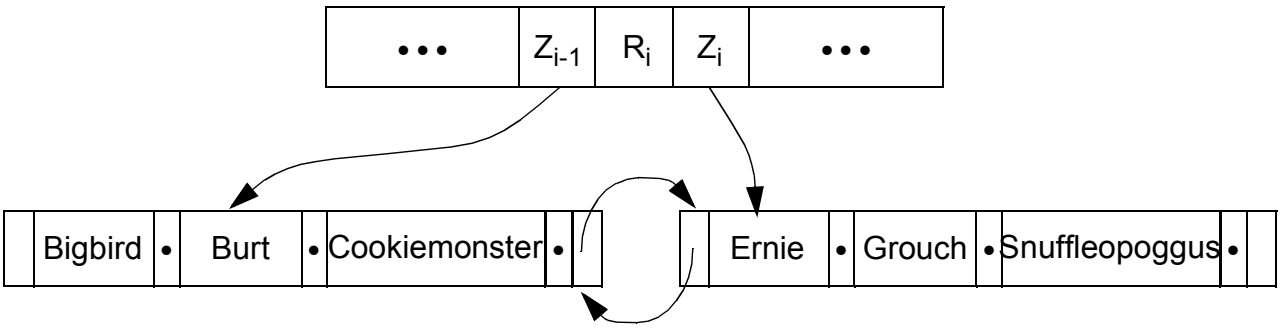
- **Präfix-B-Bäume**

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

B*-Bäume (8)

- **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

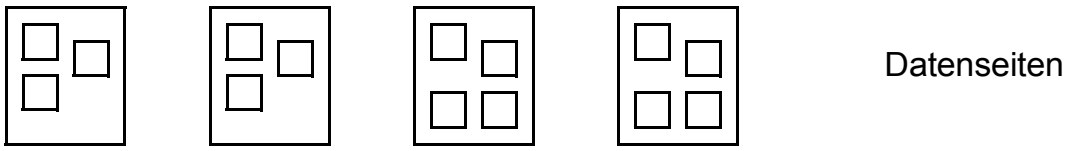
$$n_{max}(k, k^*, h^*) = 2k^* \cdot (2k+1)^{h^*-1} \qquad \text{für } h^* \geq 1 \tag{0.2}$$

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

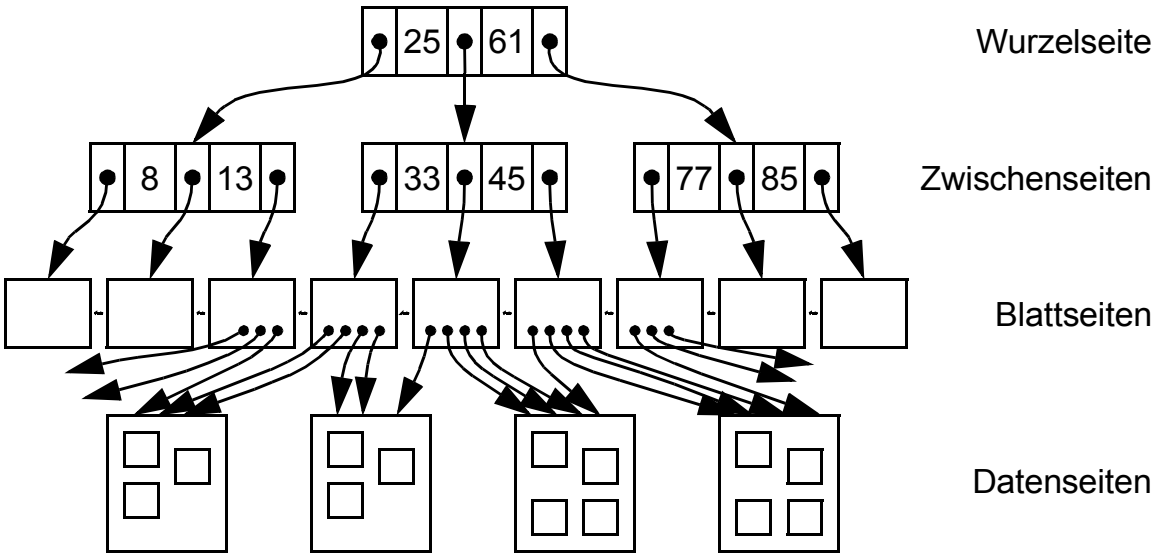
$$1 + \log_{2k+1}\frac{n}{2k^*} \leq h^* \leq 2 + \log_{k+1}\frac{n}{2k^*} \qquad \text{für } h^* \geq 2.$$

Zugriff auf eine Satzmenge

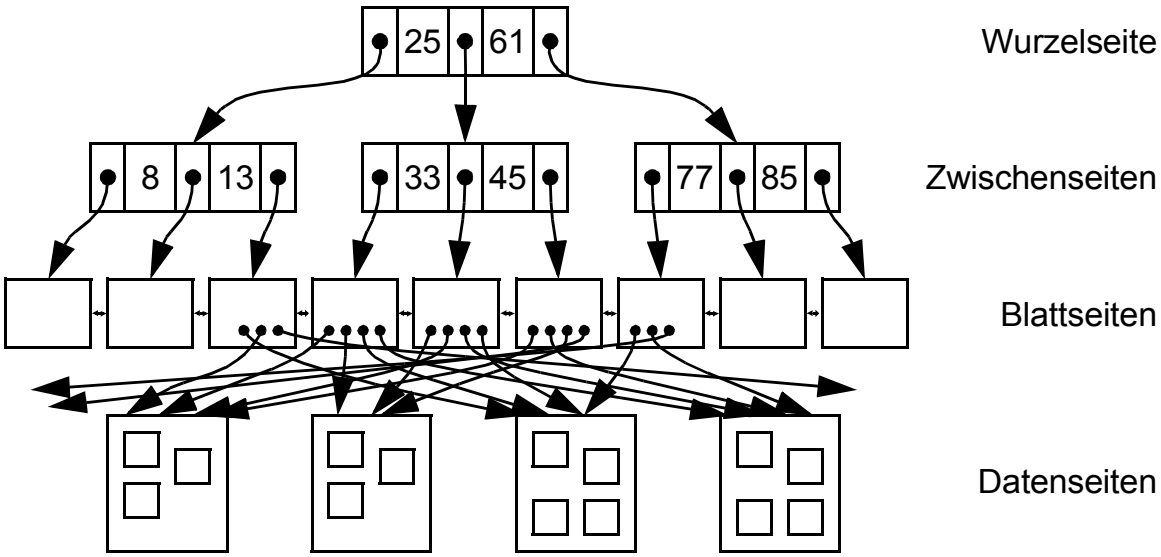
- **Einfaches Erklärungs- und Kostenmodell**
  - m = #Seiten im Segment, N_Z = #Seiten einer Zeigerliste
  - b = mittlere #Sätze /Seite, q = #Treffer der Anfrage
- **Datei- oder Segment-Scan**



- **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)
  - **exakte** 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!

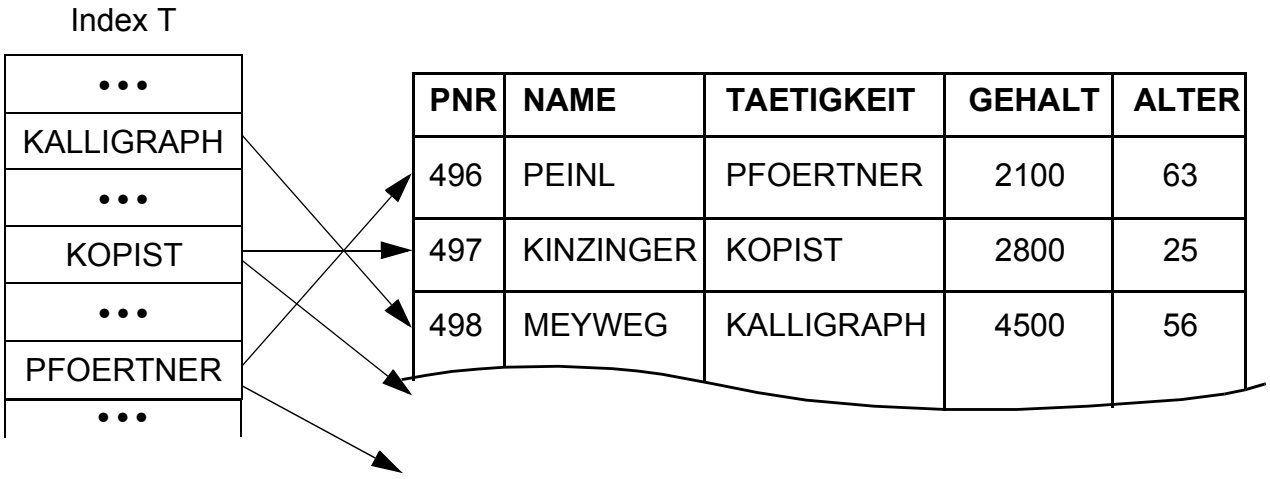
Informationssuche (2)
– strukturierte Daten

- **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, ...}



- **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

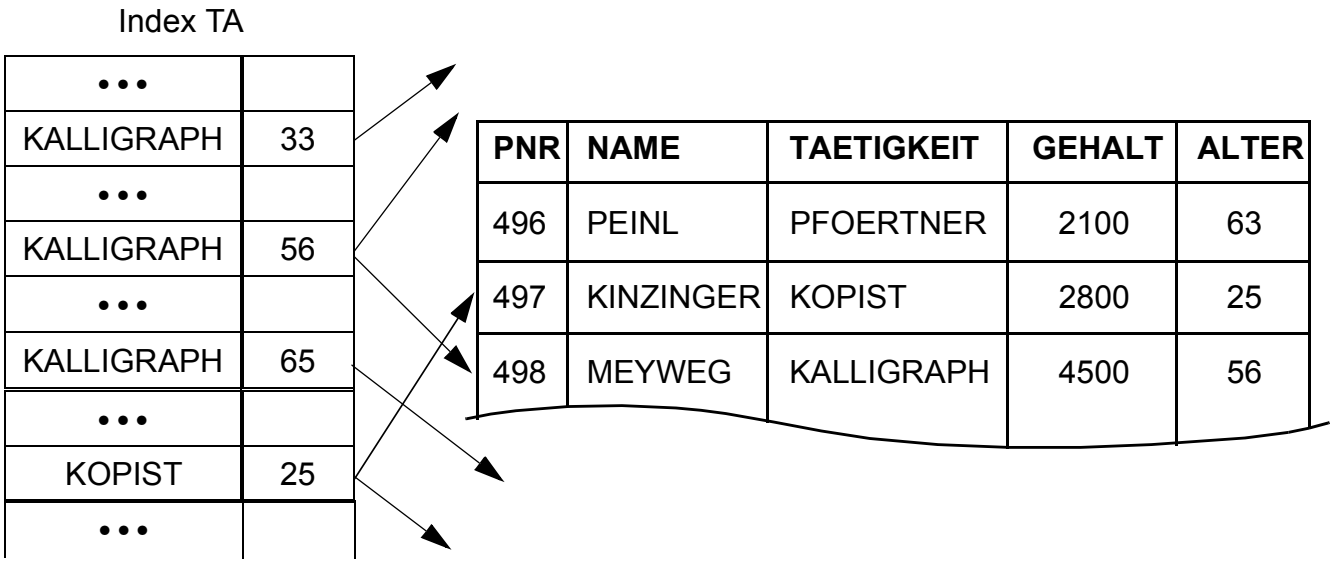
Informationssuche (3)
– strukturierte Daten

- **Anfrage**

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

- **Mehrattribut-Indexierung**

  - Anlegen eines Index (TAETIGKEIT | ALTER)



  - $h_{TA} = 3$, $j_T = 1000$, $j_A = 50$, $j_{TA} = j_T \cdot j_A$, $n = 10^6$, N_{ZTA} =

  - N_S =

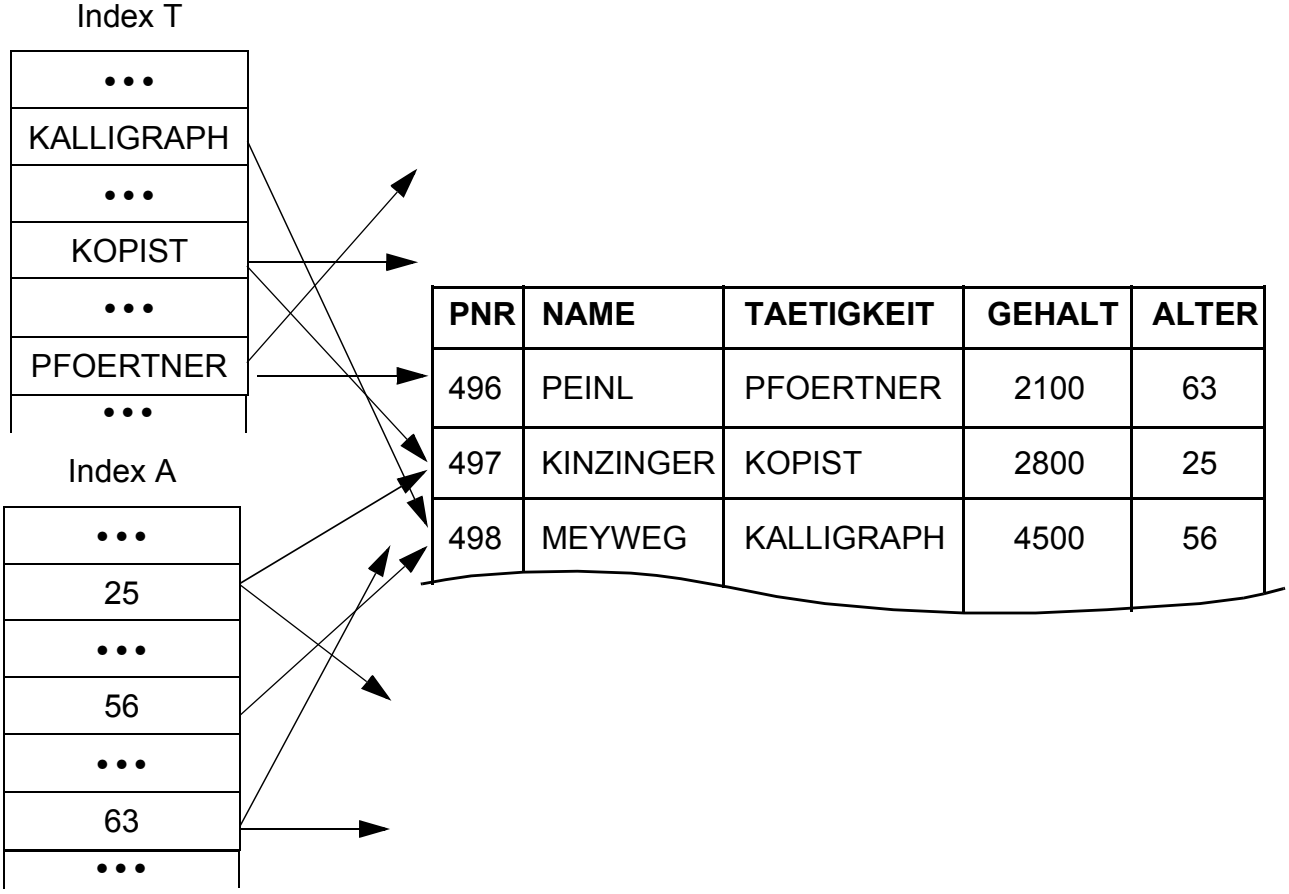
  - erhöhter Spezialisierungsgrad bei der Indexnutzung
  - Nutzung von Index TA nur für TAETIGKEIT oder nur für ALTER?

Informationssuche (4)
– strukturierte Daten

- **Anfrage**

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

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



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

- N_S =

- besser: N_S =

Aufbau des DB-Servers

- **Allgemeine Aufgaben/Eigenschaften von DBS**
  - Verwaltung von persistenten Daten (lange Lebensdauer)
  - effizienter Zugriff (Suche und Aktualisierung) auf große Mengen von Daten (GBytes – PBytes)
  - flexibler Mehrbenutzerbetrieb
  - Verknüpfung / Verwaltung von Objekten verschiedenen Typs
    (➥ typübergreifende Operationen)
  - Kapselung der Daten und ihre Isolation von den Anwendungen
    (➥ möglichst hoher Grad an Datenunabhängigkeit)
  - . . .

- **Datenmodell / DBS-Schnittstelle (SQL-API)**

Schema

ANGESTELLTER

Satztyp (Relation)

Ausprägungen

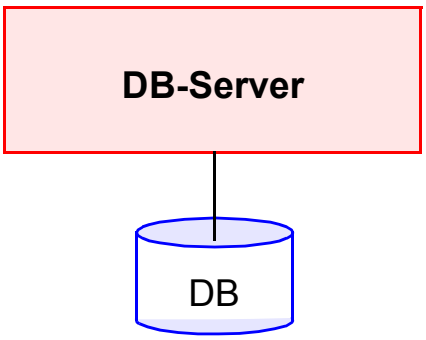
PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	PEINL	PFOERTNER	2100	63
497	KINZINGER	KOPIST	2800	25
498	MEYWEG	KALLIGRAPH	4500	56

  - Operationen zur Definition von Objekttypen (Beschreibung der Objekte)
    ➥ DB-Schema: Welche Objekte sollen in der DB gespeichert werden?
  - Operationen zum Aufsuchen und Verändern von Daten
    ➥ AW-Schnittstelle: Wie erzeugt, aktualisiert und findet man DB-Objekte?
  - Definition von Integritätsbedingungen (*Constraints*)
    ➥ Sicherung der Qualität: Was ist ein akzeptabler DB-Zustand?
  - Definition von Zugriffskontrollbedingungen
    ➥ Maßnahmen zum Datenschutz: Wer darf was?

Aufbau des DB-Servers (2)

- **Wie werden diese System- und Schnittstelleneigenschaften technisch umgesetzt?**
  - Komplexität des DBS
  - Modularität, Portabilität, Erweiterbarkeit, Zuverlässigkeit, . . .
  - Laufzeiteigenschaften (Leistung[4], Betriebsverhalten, Fehlertoleranz, . . .)
- **Ist ein monolithischer Ansatz sinnvoll?**
  - Beliebige mengenorientierte Operationen auf abstrakten Objekten (Tabellen) müssen „auf einmal“ umgesetzt und abgewickelt werden!
  - DBS-Schnittstelle: Beispiel

```
Select *
From    ANGESTELLTER P, ABTEILUNG A, . . .
Where   P.GEHALT > 8000 AND ALTER < 25 AND P.ANR = A.ANR . . .
```



  - Schnittstelle zum Externspeicher: Lesen und Schreiben von Seiten (DB ist ein sehr langen Bitstring!)
- **Außerdem: Alles ändert sich ständig!**
  - DBS-Software hat eine Lebenszeit von > 20 Jahren
  - permanente Evolution des Systems
    - wachsender Informationsbedarf: Objekttypen, Integritätsbedingungen, ...
    - neue Speicherungsstrukturen und Zugriffsverfahren, ...
    - schnelle Änderungen der eingesetzten Technologien: Speicher, ...

4. Denn für DBS und ihre Anwendungen gilt folgender populäre Spruch in besonderer Weise: „Leistung ist nicht alles, aber ohne Leistung ist alles nichts“.

Schichtenmodelle für DBS

- **Deshalb als wichtigstes Entwurfsziel:**
  Architektur eines datenunabhängigen DBS

- **Systementwurf**
  - Was sind die geeigneten Beschreibungs- und Kommunikationstechniken?
    Sie sind notwendigerweise informal.
  - Was ist auf welcher Beschreibungsebene sichtbar?
    Es ist angemessene Abstraktion erforderlich![5]
  - Wie kann eine Evolution des Systems erfolgen?
    Es muß eine Kontrolle der Abhängigkeiten erfolgen!

- **Aufbau in Schichten:**
  - „günstige Zerlegung“ des DBS in „nicht beliebig viele“ Schichten
  - optimale Bedienung der Aufgaben der darüberliegenden Schicht
  - implementierungsunabhängige Beschreibung der Schnittstellen

  ➥ Es gibt keine Architekturlehre für den Aufbau großer SW-Systeme

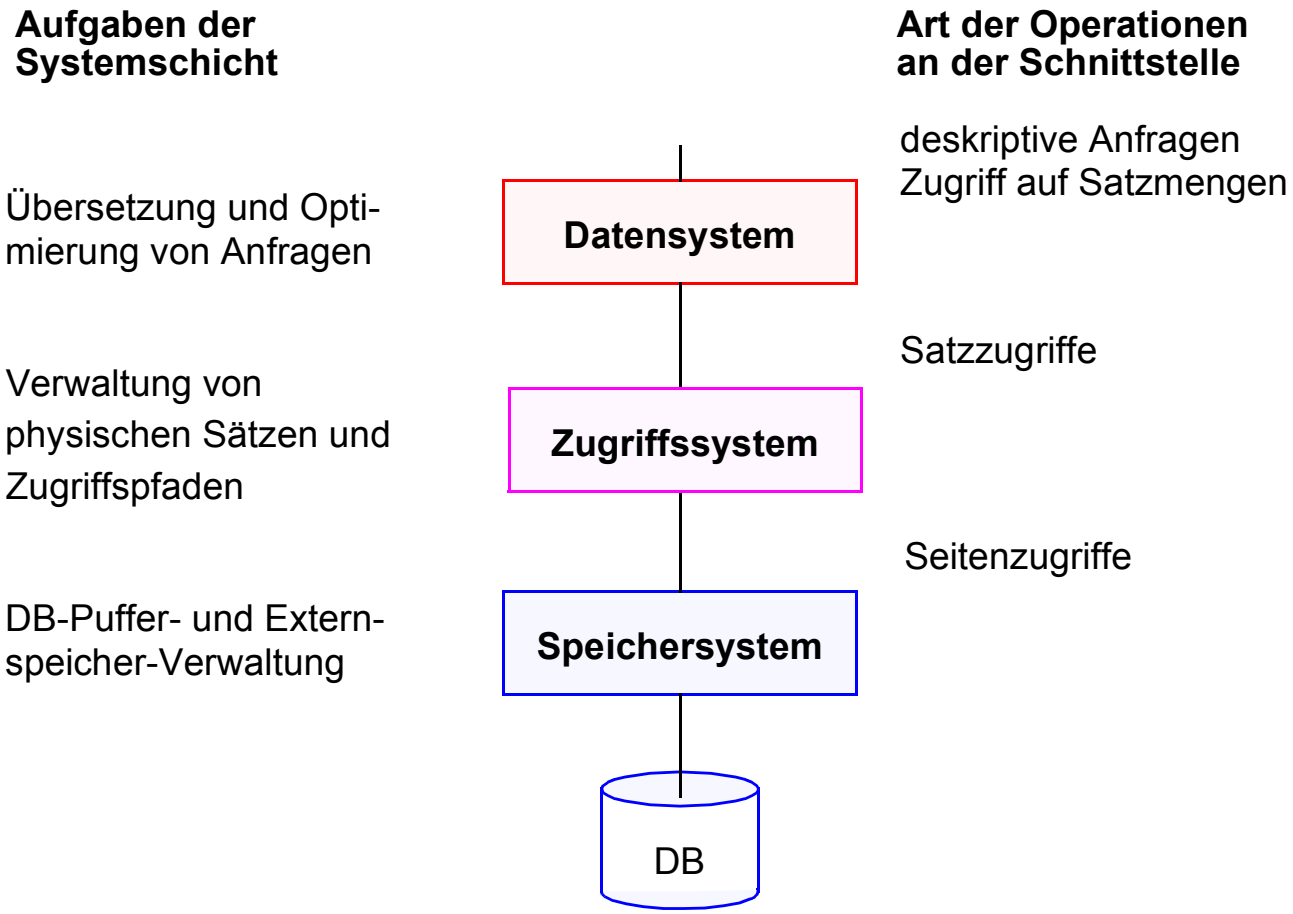
- **Empfohlene Konzepte:**
  - Geheimnisprinzip (*Information Hiding*)
  - hierarchische Strukturierung
  - generische Auslegung der Schnittstellen:
    - Nur bestimmte Objekttypen mit charakteristischen Operationen sind vorgegeben (z. B. Tabellen mit Such- und Änderungsoperationen)
    - Ihre anwendungsbezogene Spezifikation und Semantik wird im DB-Schema festgelegt
      (z. B. Tabelle ANGESTELLTER mit Integritätsbedingungen)

---

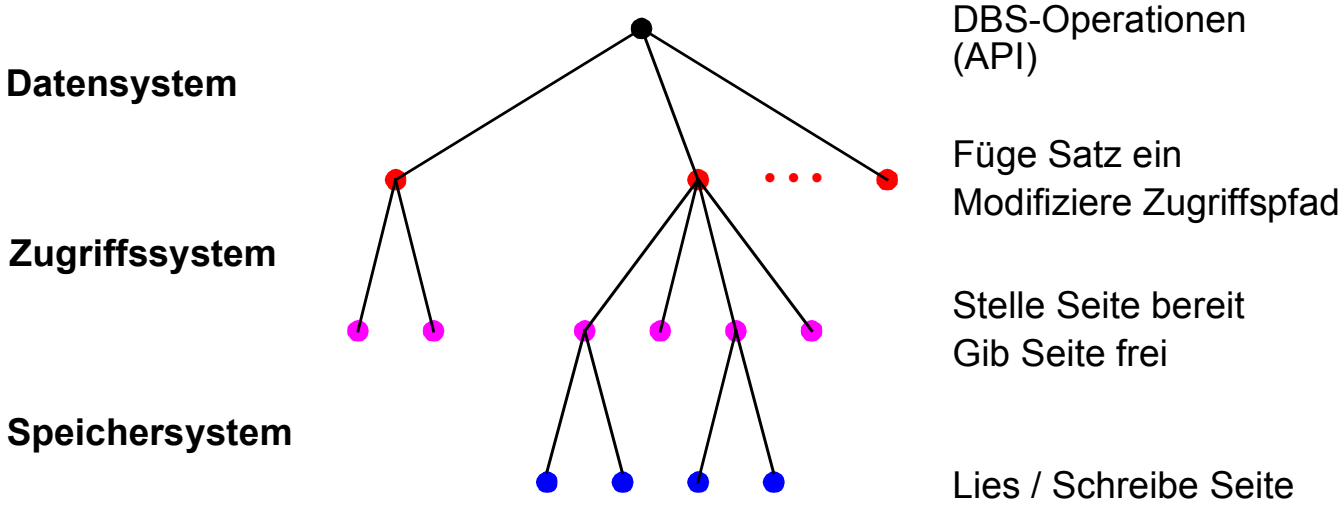
5. „Die durch Abstraktion entstandenen Konstrukte der Informatik als Bedingungen möglicher Information sind zugleich die Bedingungen der möglichen Gegenstände der Information in den Anwendungen“ (H. Wedekind in Anlehnung an eine Aussage Kants aus der „Kritik der reinen Vernunft“)
Vereinfacht ausgedrückt: Informatiker erfinden (konstruieren) abstrakte Konzepte; diese ermöglichen (oder begrenzen) wiederum die spezifischen Anwendungen.

Schichtenmodelle für DBS (2)

- **Vereinfachtes Schichtenmodell**

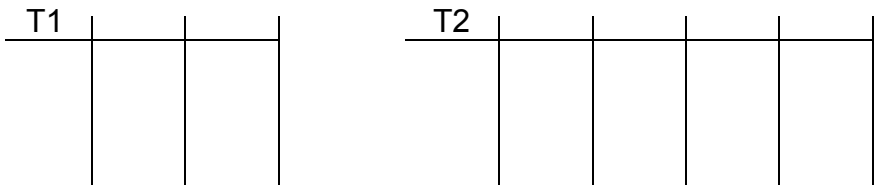


- **Dynamischer Kontrollfluß einer Operation an das DBS**



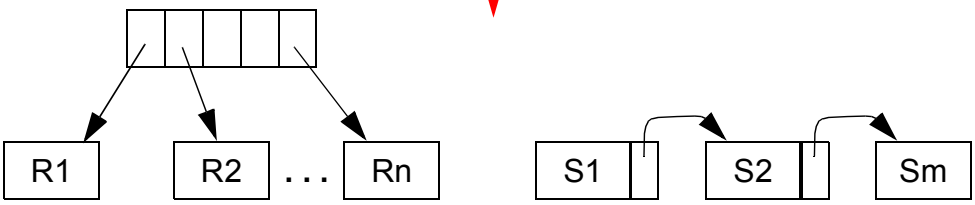
Drei-Schichten-Modell – Abbildungen

Relationen/Sichten
mit mengenorientierten
Operationen



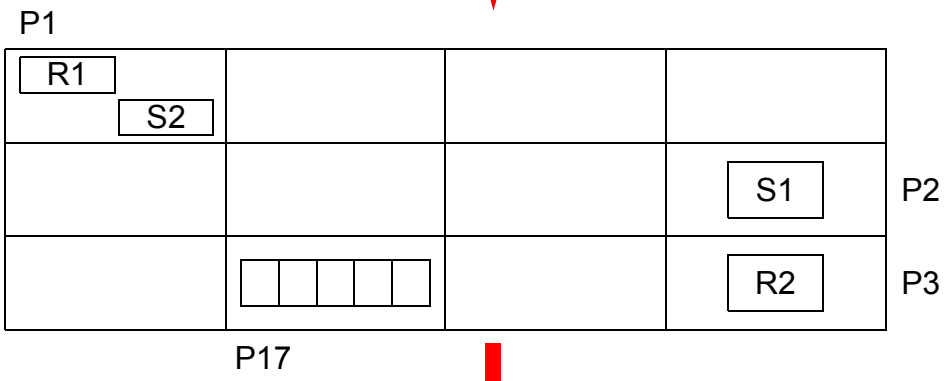
Datensystem

Vielfalt an Satztypen
und Zugriffspfaden
mit satzorientierten
Operationen



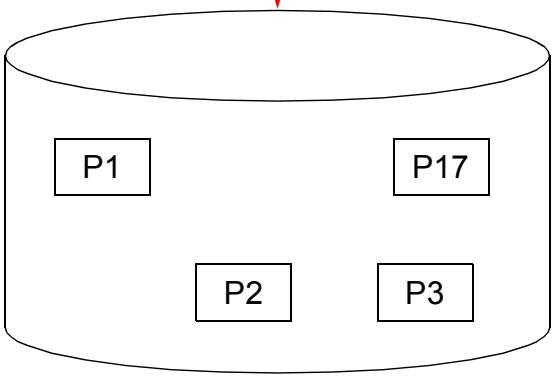
Zugriffssystem

DB-Puffer im HSP
mit Seitenzugriff



Speichersystem

Externspeicher
mit Dateien
verschiedenen Typs



Architektur eines DBS – weitere Komponenten

- **Entwurfsziel:**

  DBS müssen von ihrem Aufbau und ihrer Einsatzorientierung her in hohem Maße generische Systeme sein. Sie sind so zu entwerfen, daß sie flexibel durch Parameterwahl und ggf. durch Einbindung spezieller Komponenten für eine vorgegebene Anwendungsumgebung zu konfigurieren sind

- **Rolle der Metadaten**

  - Metadaten enthalten Informationen über die zu verwaltenden Daten
  - Sie beschreiben also diese Daten (Benutzerdaten) näher hinsichtlich Inhalt, Bedeutung, Nutzung, Integritätsbedingungen, Zugriffskontrolle usw.
  - Die Metadaten lassen sich unabhängig vom DBS beschreiben (für alle Schichten: Daten-, Zugriffs- und Speichersystem)

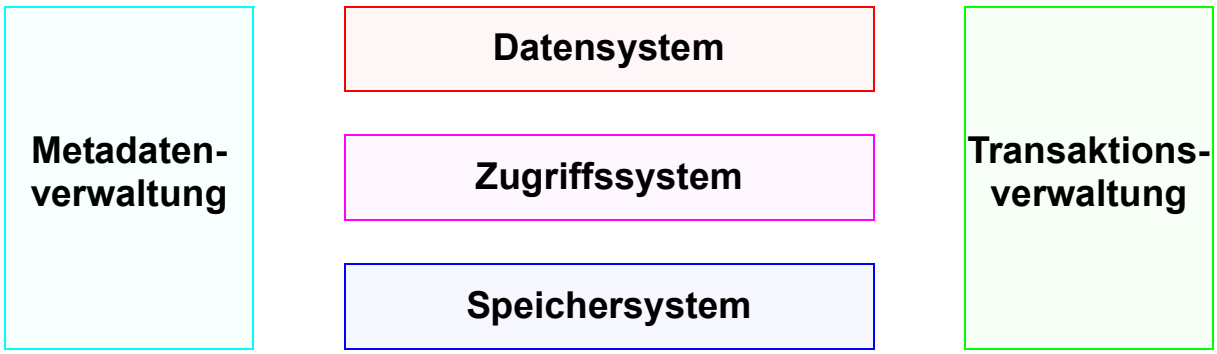
  ➥ Dadurch erfolgt das „Zuschneidern eines DBS“ auf eine konkrete Einsatzumgebung. Die separate Spezifikation, Verwaltung und Nutzung von Metadaten bildet die Grundlage dafür, daß DBS hochgradig „generische“ Systeme sind

- **Verwaltung der Daten, die Daten beschreiben:**

  - Metadaten fallen in allen DBS-Schichten an
  - Metadatenverwaltung, DB-Katalog, Data-Dictionary-System, DD-System, ...

- **Transaktionsverwaltung**

  - **Realisierung der ACID-Eigenschaften**
    (Synchronisation, Logging/Recovery, Integritätssicherung)



Zusammenfassung

- **(Momentan) exponentielle Wachstumsgesetze**
  - Moore's Law:
    Verdopplung der der Anzahl der Bauelemente pro Chip
    bei Halbierung der Preise alle18 Monate
  - Gilder's Law:
    Verdreifachung der Kommunikationsbandbreite alle 18 Monate

- **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

- **Aufbau des DB-Servers**
  - DBS sind hochgradig generische Systeme,
    die permanente Evolution zulassen müssen
  - Hierarchische Schichtenmodelle sind unerlässlich!

Entwicklungstrends

- **Die Beobachtung von Gordon Moore**

  Zunächst sollte G. Moore, Director, Research and Dev. Lab., Fairchild Semiconductor eine Vorhersage über die Entwicklung der Halbleiterindustrie für die nächsten 10 Jahre machen, die er in Form einer logarithmisch-linearen Beziehung zwischen Gerätekomplexität (höhere Packungsdichte bei reduzierten Kosten) und Zeit beschrieb[6]. 1975 revidierte er seine ursprüngliche Vorhersage über die Kapazität/Packungsdichte zu

  CircuitsPerChip (year) = $2^{(year - 1975)/1.5} * K$. (Dies wurde als Moore's Law bekannt).

- **Prozessoren**

  Die Mips-Raten von Prozessoren stiegen zwischen 1965 und 1990 von etwa 0.6 Mips auf 40 Mips (Skalarleistung), was einem Faktor von 70 entspricht. Die Leistungsfähigkeit von Mikroprozessoren (Ein-Chip-Prozessoren) erhöhte sich seit 1980 wesentlich stärker; sie verdoppelte sich ungefähr alle 18 Monate.

  Joy's Law:

  SunMips (year) = $2^{year - 1984}$ Mips for year in [1984 . . . 2000]

- **Elektronische Speicher**

  Die Speicherkapazität pro Chip erhöhte sich um den Faktor 4 alle 3 Jahre.

  Moore's Law (Roadmap für el. Speicher):

  MemoryChipCapacity (year) = $4^{\frac{(year - 1970)}{3}}$ Kb/chip for year in [1970 ...2000]

- **Magnetische Speicher**

  Die Lese- und Schreibdichte bei magnetischen Speichern stieg jeweils um den Faktor 10 innerhalb von einer Dekade.

  Hoaglands's Law:

  MagneticAreaDensity (year) = $10^{\frac{(year - 1970}{10}}$ Mb/inch^2 for year in [1970 ... 2000]

---

6. Moore, G.: Cramming more Components onto Inregrated Circuits, in: Electronics, April 1965. „The complexity for minimum component costs has increased at a rate of roughly a factor of 2 per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will remain nearly constant for at least 10 years".