2. E/A-Architektur und Zugriff

Stefan Deßloch

Technische Universität
Kaiserslautern

Gliederung

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

E/A-Architektur von IS

- Bisherige Annahmen
  - Datenstrukturen wie Felder, Listen, Bäume, Graphen, ...
    - lokale Speicherung und direkter Zugriff auf alle Elemente
    - hohe Zugriffsgeschwindigkeit
  - ausschließlich im Hauptspeicher (HSP als DRAM) verfügbar, d. h. Bestand nur für die Dauer einer Programmausführung („transiente" Daten – nur für ein Programm)
- Hier nun neue Aspekte
  - Nutzung von Externspeichern und neuen Operationen
  - Datenstrukturen werden gespeichert und verwaltet
    - verteilt und dezentral
    - auf verschiedenen Speichertypen (Magnetplatte (MP), Flash-Speicher (Solid State Disk, SSD), Magnetband, DVD, optische Speicher, ...)
    - in verschiedenen Rechensystemen
    - im Web

E/A-Architektur von IS (2)

- Neue Aspekte (Forts.)
  - andere Arten des Zugriffs: Lese- und Schreiboperationen, in vorgegebenen Einheiten von Blöcken und Sätzen
    - ➪ Strukturen und zugehörige Algorithmen (Suchen, Sortieren), die im HSP 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
  - Energieeffizienz (Green Computing in DBS): Energieverbrauch für Lesen und Schreiben, für Idle-Zustand im Betrieb

Problemstellung

- Langfristige Speicherung und Organisation der Daten im Hauptspeicher?
  - Speicher ist (noch) flüchtig! (➪ Persistenz)
  - Speicher ist (noch) zu teuer und zu klein
  - Kostenverhältnis MP : DRAM (➪ 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?

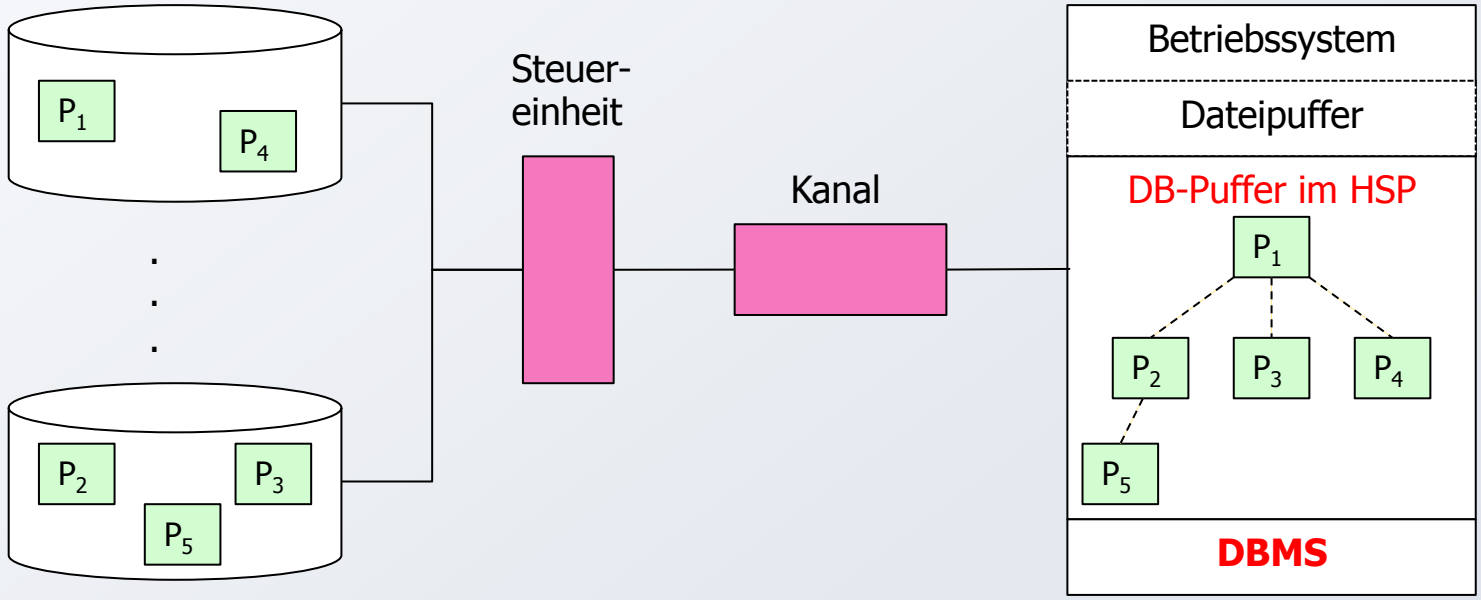
„Idealer Speicher"

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

  ➪ 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



Magnetplattenspeicher
ca. 10 ms

Zugriffslücke
$>2*10^5$

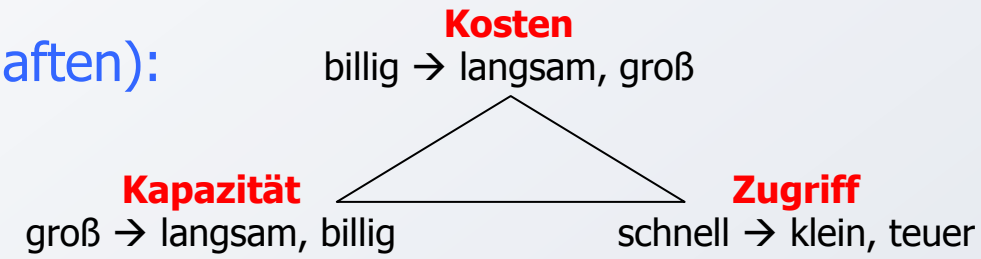
Hauptspeicher
< 50 ns

Faktor 100 - 1000 ←

→ *Faktor 100*

Annäherung an idealen Speicher

- Ziel (ideale Eigenschaften):
  - 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

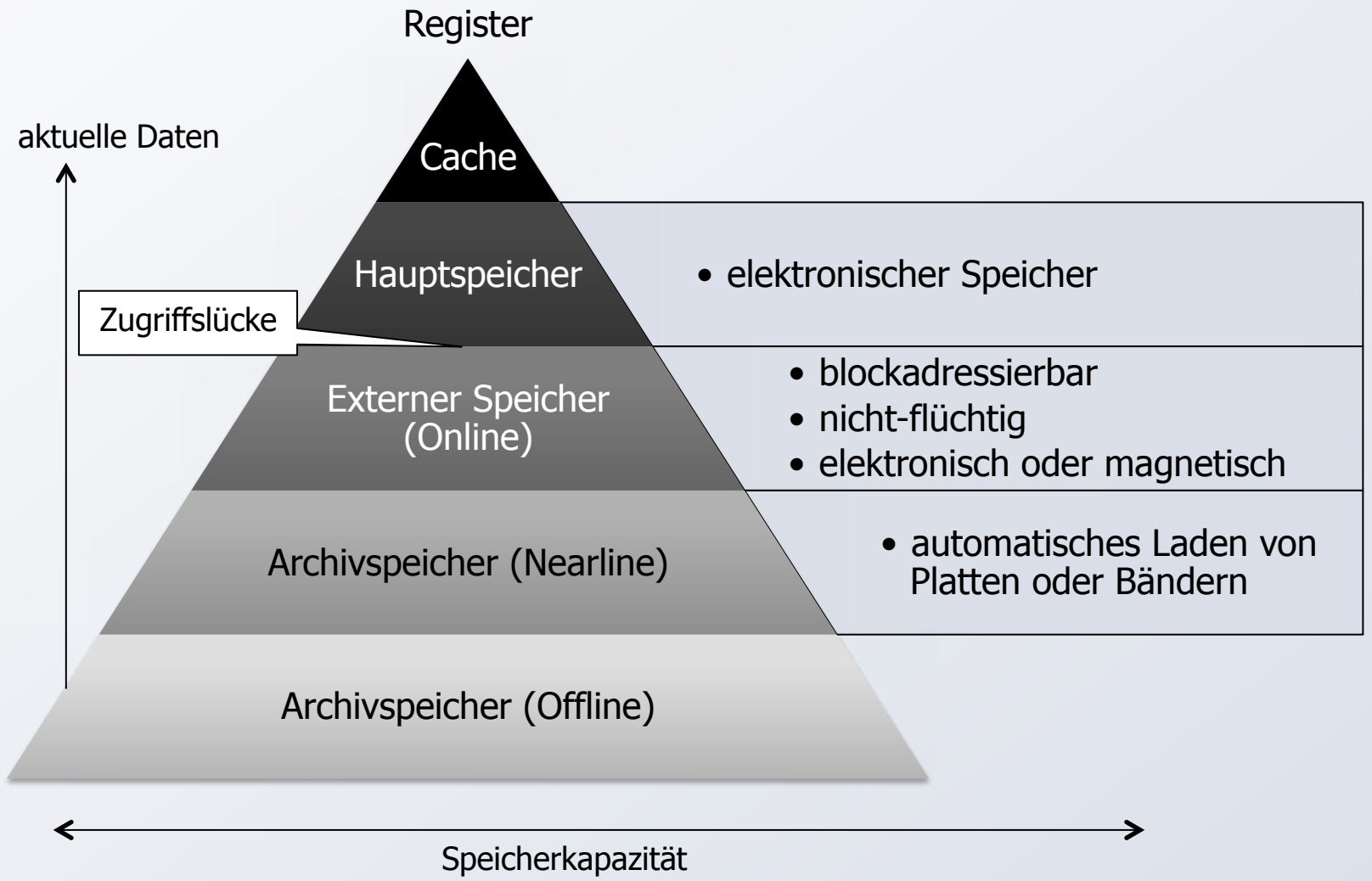
  ➪ E/A-Architektur zielt auf kosteneffektive Lösung ab
- Lückenfüller-Technologien
  - Neue Speichertypen, die in die Lücke passen?
  - Blasenspeicher (magnetic bubbles), CCD usw. sind untauglich (~1980)
  - Flash-Speicher (SSDs)?!

Speicherhierarchie

- Wie kann die Zugriffslücke überbrückt werden?
  - teilweise durch erweiterte HSP, SSDs und MP-Caches
  - Nutzung von Lokalitätseigenschaften
    - Allokation von Daten mit hoher Zugriffswahrscheinlichkeit in schnelle (relativ kleine) Speicher
    - Größter Teil 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)

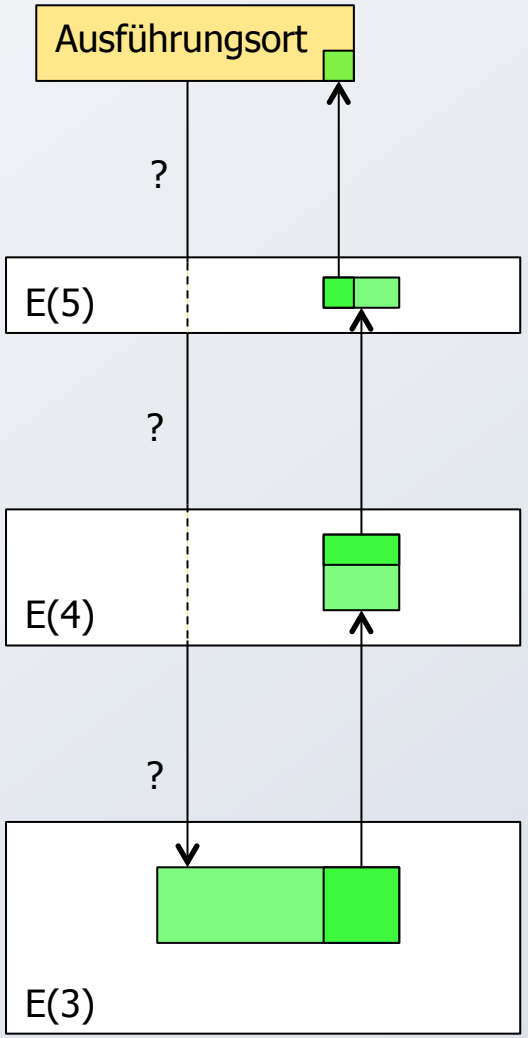


Charakteristische Merkmale

	Ebene	Kapazität	Zugriffs-zeit	Transfer: Kontrolle	Transfer: Einheit
6	Register	Bytes	1-5 ns	Programm/ Compiler	1-8 Bytes
5	Cache	K - M Bytes	2-20 ns	Cache-Controller	8-128 Bytes
4	Hauptspeicher	M- G Bytes	50-100 ns	Betriebs-system	1-16 KBytes
3	Plattenspeicher	G – T Bytes	ms	Benutzer/ Operator	MBytes (Dateien)
2	Bandspeicher	T Bytes	sec	Benutzer/ Operator	MBytes (Dateien)
1	Archivspeicher	T – P Bytes	sec-min		

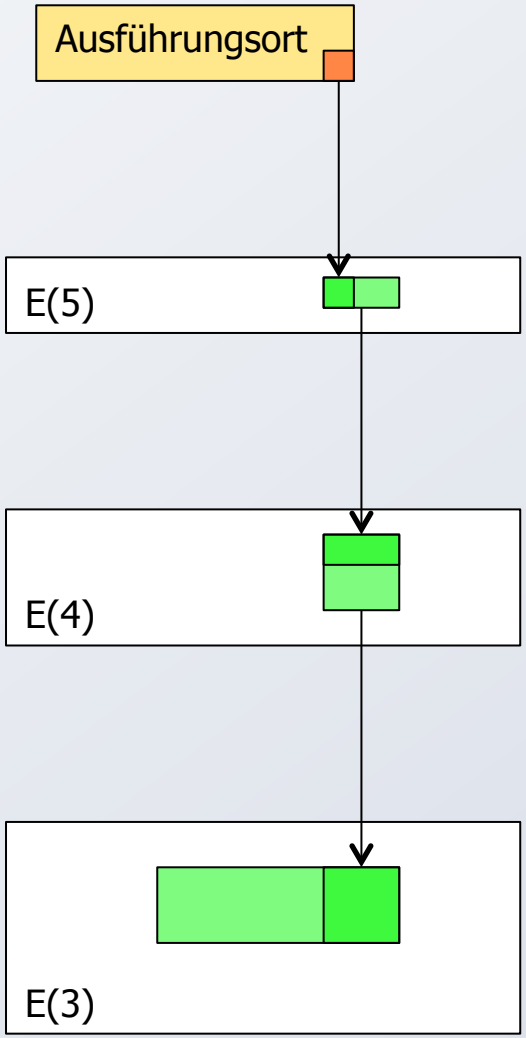
Inklusionseigenschaften

- Aufgaben auf jeder Ebene
  - Lokalisieren von Datenelementen
  - Allokation von Speicherplatz
  - Ersetzung von Objekten
  - Realisierung der Schreibstrategie
- Inklusionseigenschaft
  - Für Datum auf Ebene Ei ist auch immer ein (mglw. veraltete) Kopie auf allen darunter liegenden Ebenen E_j mit j<i vorhanden
  - Vorteil: Rückschreiben wird einfach, betrifft immer nur E_{i-1}
- Read-through:
  - Daten werden bedarfsgetrieben im schichtenspezifischen Granulat schichtweise „nach oben“ bewegt



Inklusion beim Schreiben

- Write-through:
  - Sofort nach der Änderung wird das Datum auch in der nächsttieferen Schicht geändert (FORCE)
- Write-back:
  - Erst bei einer Ersetzung werden die geänderten Daten schichtweise „nach unten" geschoben (NOFORCE)
- Wahl des Verfahrens kann schichtenspezifisch sein
- Durch Inklusionseigenschaft:
  - (Altes) Datum auf der nächsttieferen Schicht vorhanden, muss nicht angefordert werden, dadurch keine weiteren Verdrängungen



Relative Zugriffszeiten in einer Speicherhierarchie

- Wie weit sind die Daten entfernt?
- Welche Konsequenzen ergeben sich daraus?
  - Caching
  - Replikation
  - Prefetching

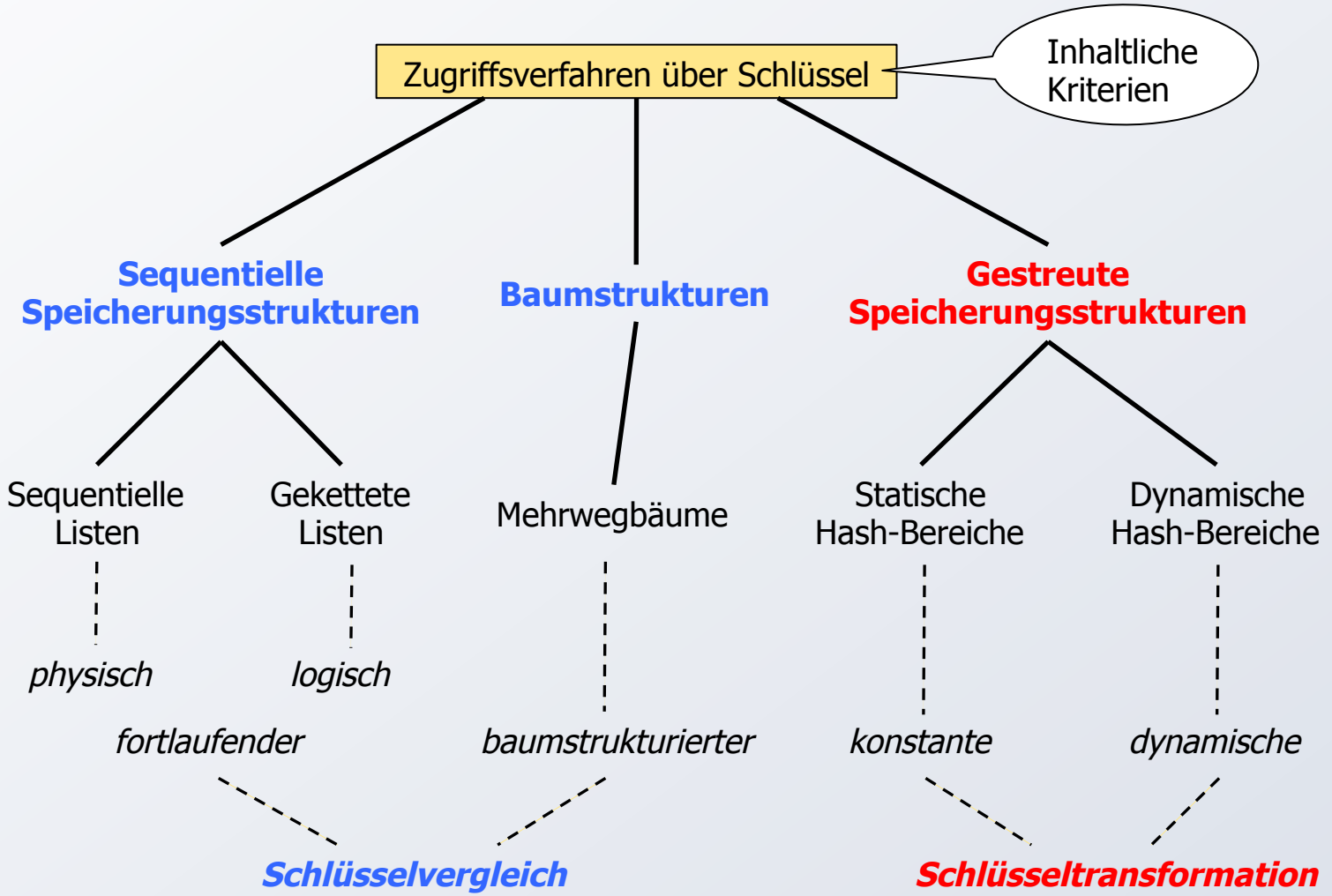
Speicherhierarchie		**Was bedeutet das in „unseren Dimensionen"?**	
Speichertyp	Rel. Zugriffszeit	Zugriffsort	Zugriffszeit
Register	1	mein Kopf	1 min
Cache (on chip)	2	dieser Raum	2 min
Cache (on board)	10	dieses Gebäude	10 min
Hauptspeicher	100	Frankfurt (mit Auto)	100 min
Magnetplatte	10^6–10^7	Pluto ($5910 * 10^6$ km)	> 2 Jahre
Magnetband/ Opt. Speicher (automat. Laden)	10^9–10^{10}	Andromeda	> 2000 Jahre

Prognosen

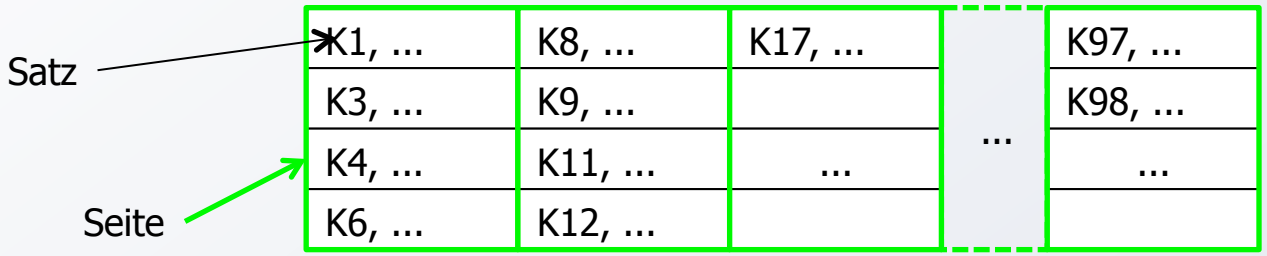
- Moore's Gesetz (interpretiert nach J. Gray)
  - Verhältnis „Leistung/Preis" verdoppelt sich alle 18 Monate!
  - Faktor 100 pro Dekade
  - Exponentielles Wachstum: Fortschritt in den nächsten 18 Monaten = Gesamter Fortschritt bis heute
    - neuer Speicher = Summe des gesamten „alten" Speichers
    - neue Verarbeitungsleistung = Summe der bisherigen Verarbeitungsleistung
- 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!"

Zugriffsverfahren



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

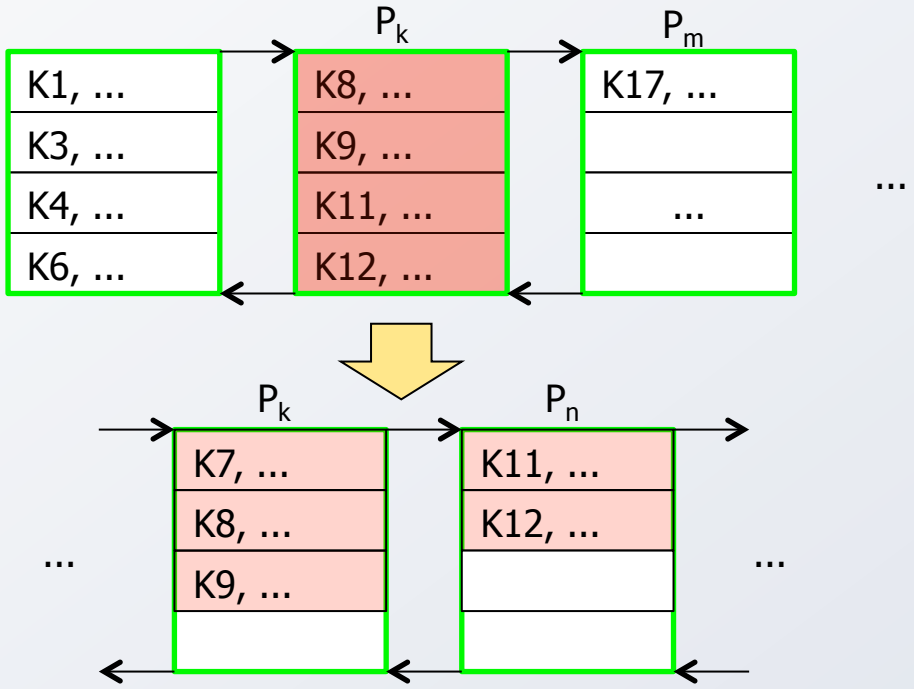
  ➪ Cluster-Bildung kann pro Satztyp nur bezüglich eines Kriteriums erfolgen; sonst wird Redundanz eingeführt!

Sequentielle Listen - Kosten

- Zugriffskosten (NS = Anzahl physischer Seitenzugriffe)
  - Sequentieller Zugriff auf alle Datensätze eines Typs: bei n Sätzen und mittlerem Blockungsfaktor b (#Sätze pro Seite)
    - $N_S = n/b$
- Sortierte Listen:
  - Zugriff über eindeutigen Schlüssel kostet im Mittel
    - $N_S = n/2b$
  - Bei fortlaufender Seitenzuordnung Binärsuche anwendbar:
    - $N_S = \log_2(n/b)$
- Ungeordnete Listen:
  - Wieviele Seitenzugriffe benötigt der direkte Zugriff?
    - $N_S = n/2b$
- Änderungen in sortierten Listen sind sehr teuer:
  - Einfügen von K7?
  - Änderungsdienst: Verschiebekosten im Mittel:
    - $N_s = n/2b * 2$
    - (Faktor 2: Lesen + Rückschreiben)

Sequentielle Listen - Splitting

- Deshalb Einsatz einer Split-Technik (Splitting):



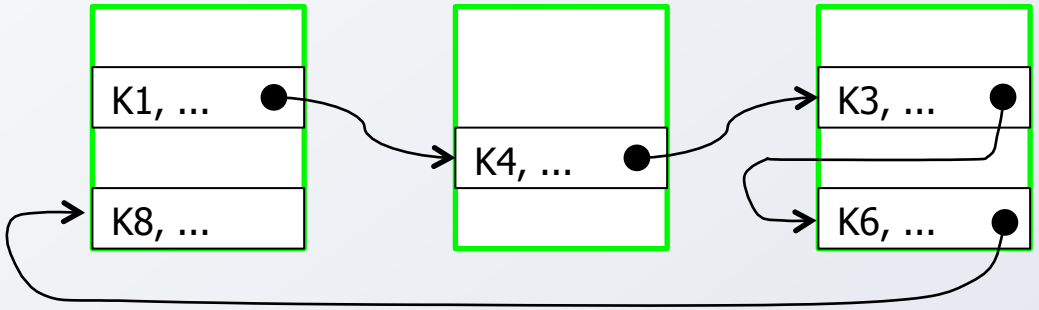
$N_S = 3 * 2$ (Änderung auf max. 3 Seiten beschränkt!)

➪ aber Suche der Position der durchzuführenden Änderung

- Welche Eigenschaften der seq. Liste gehen verloren?
  - physische Seitennachbarschaft (→ erhöhter Suchaufwand)
  - max. Belegungsgrad

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

Gekettete Listen - Kosten

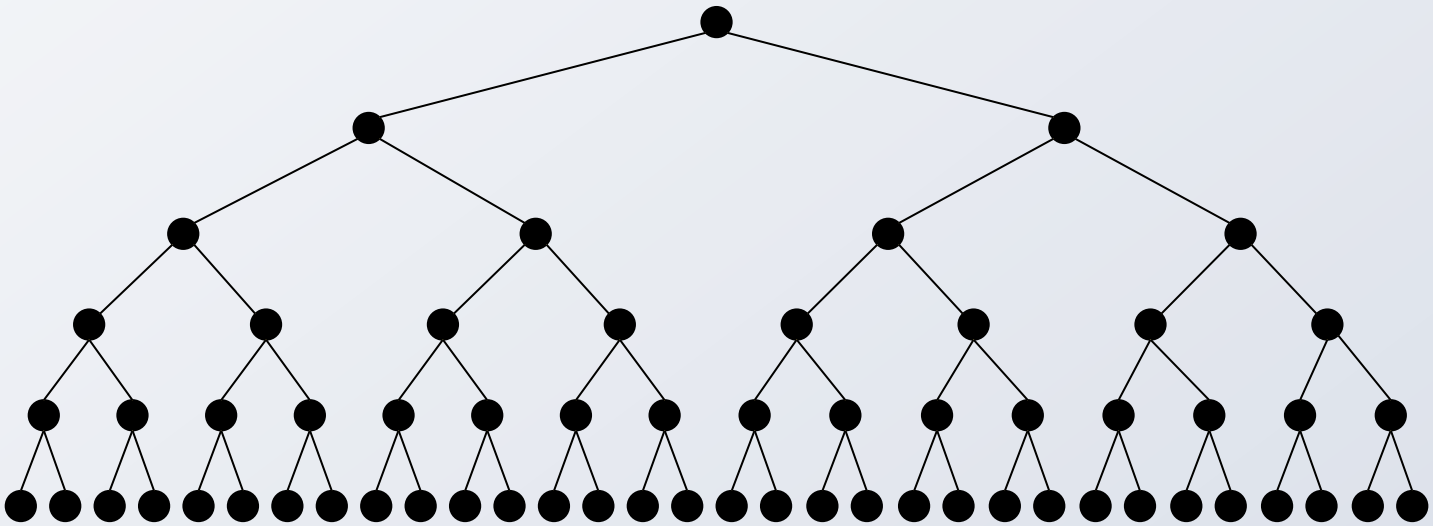
- Zugriffskosten
  - sequentieller Zugriff auf alle Sätze:
    - $N_S = n$
  - direkte Suche (= fortlaufende Suche) bei Verkettung in Sortierreihenfolge:
    - $N_S = n/2$
  - Einfügen relativ billig, wenn Einfügeposition bekannt
    - $N_S \leq 2 * 2$
- Mehrfachverkettung nach verschiedenen Kriterien (Schlüsseln) möglich

Mehrwegbäume - Motivation

- Binäre Suchbäume (SB) sind bekannt
  - natürlicher binärer SB
  - AVL-Baum (höhenbalanciert)
  - gewichtsbalancierter SB
  - ➪ Bei Abbildung auf Externspeicher ist die Höhe des SB entscheidend!
  - Höhe h_b von binären SB mit n Knoten

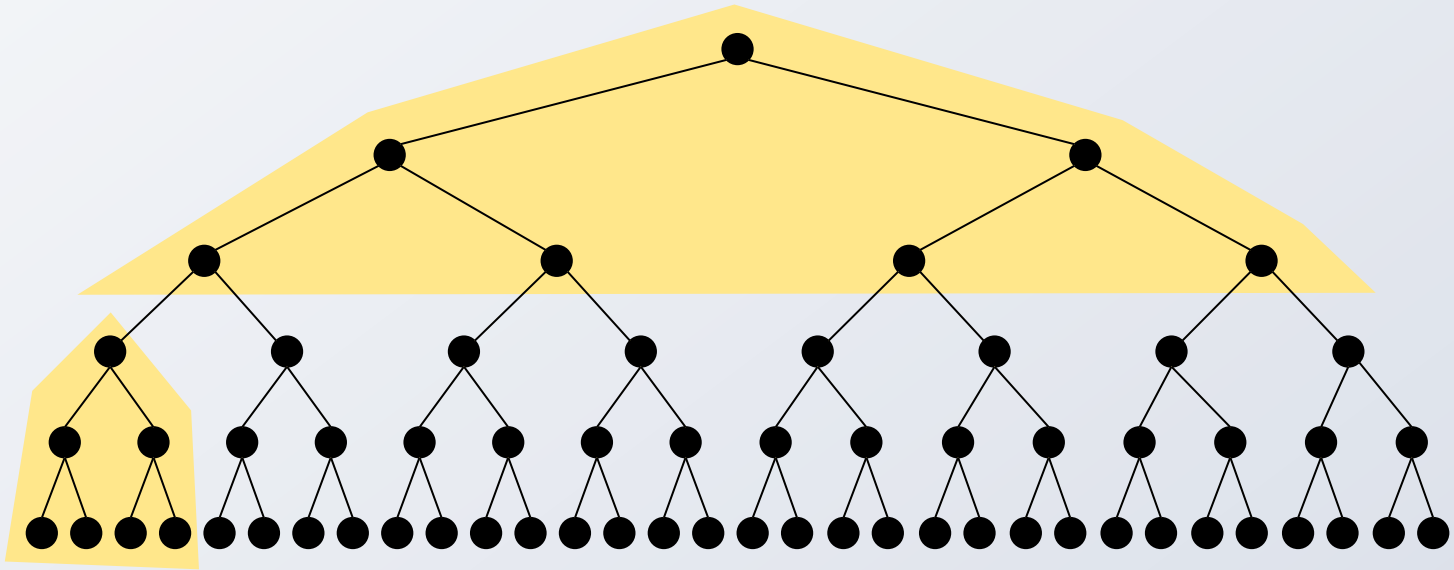
$$\lceil \log_2(n+1) \rceil \leq h_b \leq n$$

    - Beispiel: n = 10^6 → $h_b \approx 20$; worst case: 200ms Zugriffszeit



Mehrwegbäume*

- 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!



(*) R. Bayer, E. McCreight: *Organization and Maintenance of Large Ordered Indices*. (1972)

Mehrwegbäume (3)

- 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" *
  - ⇨ 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

(*) D. Comer: *The Ubiquitous B-Tree*. ACM Computing Surveys 11(2), 1979

Mehrwegbäume (4)

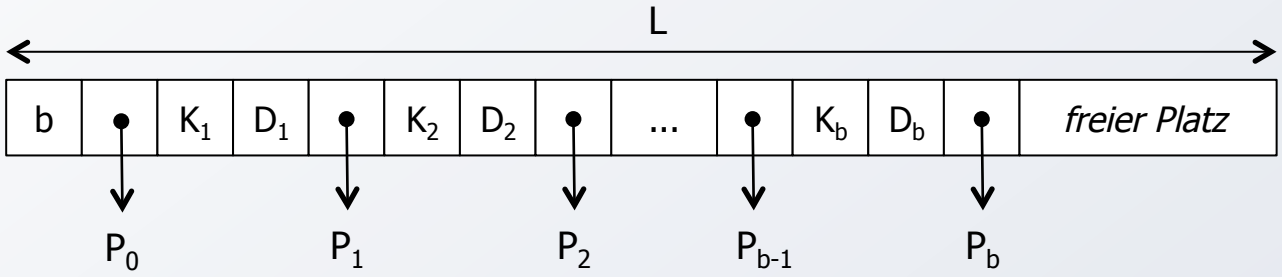
- Höhe von Mehrwegbäumen: $h_B = O(\log_{k+1} n)$
- Balancierte Struktur:
  - unabhängig von Schlüsselmenge
  - unabhängig von Einfügereihenfolge
- Breites Spektrum von Anwendungen
  - Dateiorganisation („logische Zugriffsmethode", VSAM)
  - Datenbanksysteme (Varianten des B*-Baumes sind in allen DBS zu finden!)
  - Text- und Dokumentenorganisation
  - Suchmaschinen im Web, ...

B-Bäume

- Definition:
  - Seien k, h ganze Zahlen, $h \geq 0$, $k > 0$. Ein **B-Baum** B der Klasse τ (k, h) ist entweder ein leerer Baum oder ein geordneter Suchbaum mit folgenden Eigenschaften:
  - Jeder Pfad von der Wurzel zu einem Blatt hat die gleiche Länge h–1.
  - Jeder Knoten außer der Wurzel und den Blättern hat mindestens k+1 Söhne. Die Wurzel ist ein Blatt oder hat mindestens 2 Söhne.
  - Jeder Knoten hat höchstens 2k+1 Söhne.
  - Jedes Blatt mit der Ausnahme der Wurzel als Blatt hat mindestens k und höchstens 2k Einträge.

B-Bäume - Knotenformat

- Für einen B-Baum ergibt sich folgendes Knotenformat:



- Einträge
  - Die Felder für #Einträge (b), Schlüssel (K_i), Daten (D_i) 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

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

B-Bäume – Reformulierung der Definition

Definition:

Seien k, h ganze Zahlen, $h \geq 0$, $k > 0$. Ein **B-Baum B** der Klasse τ (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.

- Reformulierung der Definition:

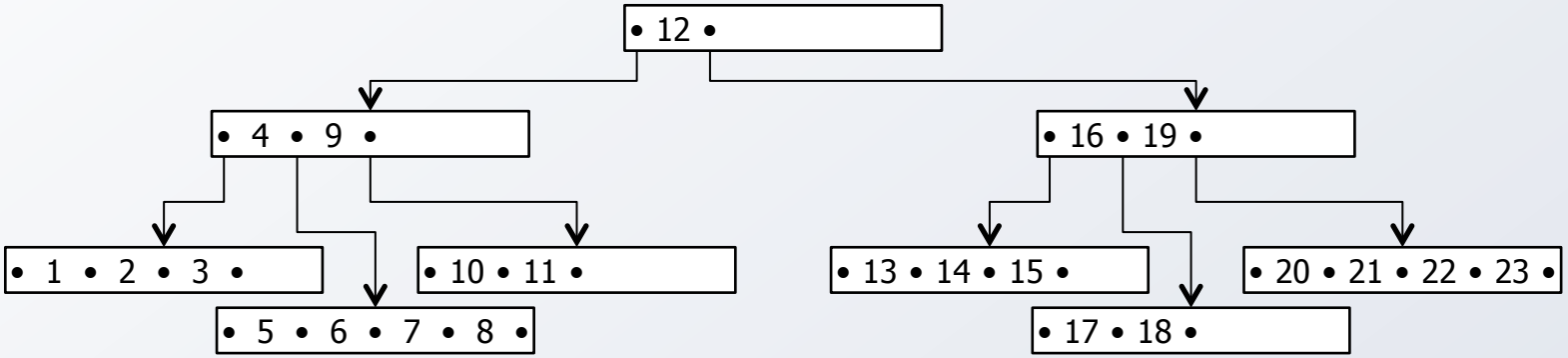
  (4) und (3): Eine Seite darf höchstens voll sein.

  (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 – Beispiel und Eigenschaften

- B-Baum der Klasse τ (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
  - p_i verweist auf Teilbaum mit Schlüsseln > K_i und < K_{i+1}
  - in den Blattknoten sind die Verweise undefiniert
- Alle Schlüssel/Sätze können also sortiert aufgesucht werden
- Die Klassen τ (k, h) sind nicht alle disjunkt. Beispielsweise ist ein Baum aus τ (2, 3) mit maximaler Belegung ebenso in τ (3, 3) und τ (4, 3).

B-Bäume – Beispiel (2)

- Was sind typische Größen in B-Bäumen?
  - (in Byte:) L = 8192, lb = 2, lP = 4, lK = 8, lD = 88

$$k = \frac{b_{\max}}{2} = \left\lfloor \frac{L - l_b - l_p}{2(l_K + l_D + l_p)} \right\rfloor = \left\lfloor \frac{8192 - 2 - 4}{2 \cdot (8 + 88 + 4)} \right\rfloor = \left\lfloor \frac{8186}{200} \right\rfloor = 40$$

- Bei ausgelagerten Daten?
  - lD = 4 (Verweis auf Datenseite)

$$k = \left\lfloor \frac{8192 - 2 - 4}{2 \cdot (8 + 4 + 4)} \right\rfloor = \left\lfloor \frac{8186}{64} \right\rfloor = 127$$

B-Bäume - Höhe

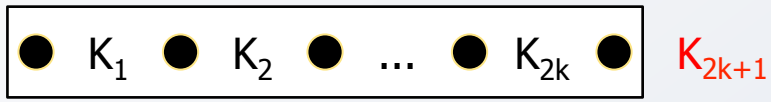
- Höhe h: Bei einem Baum der Klasse τ (k, h) mit n Schlüsseln gilt für seine Höhe:
  - $\log_{2k+1}(n+1) \leq h \leq \log_{k+1}((n+1)/2) + 1$ für $n \geq 1$

  und
  - $h = 0$ für $n = 0$
- Beispiel:
  - mit $2k+1 = 100$, $n = 10^6$: Höhe $h \geq 3$
- Balancierte Struktur:
  - unabhängig von Schlüsselmenge
  - unabhängig von ihrer Einfügereihenfolge

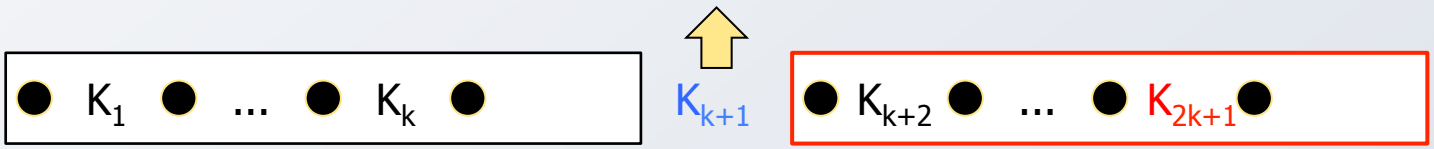
  ⇨ Wie wird das erreicht?

B-Bäume - Einfügen

- Einfügen in B-Bäumen: Wurzel läuft über



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

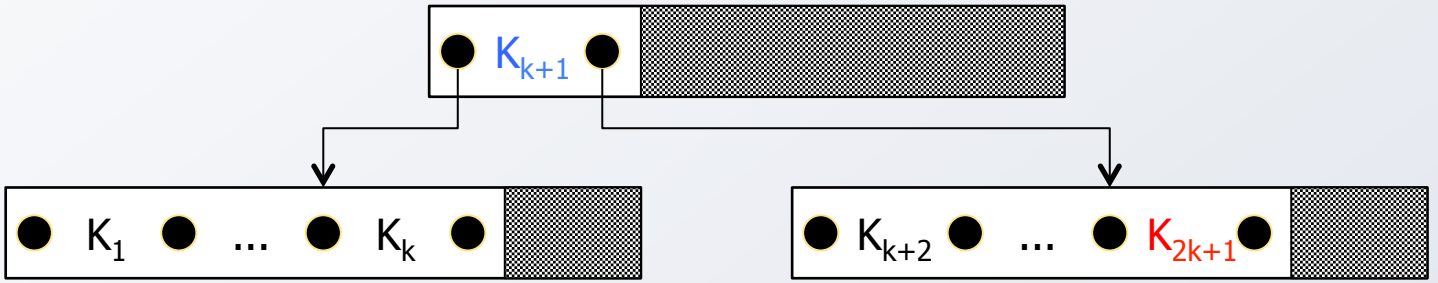


  - Der mittlere Schlüssel (Median) und der Verweis auf die neue Seite werden zum Vaterknoten gereicht.
  - Ggf. muss ein Vaterknoten angelegt werden (Anforderung einer neuen Seite).

B-Bäume – Einfügen (2)

- Einfügen in B-Bäumen (Forts.)

  Hier wird eine neue Wurzel angelegt.



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

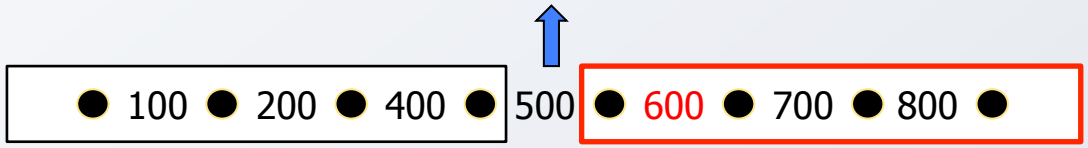
  ➪ Bei B-Bäumen ist das Wachstum von den Blättern zur Wurzel hin gerichtet

B-Bäume – Einfügebeispiel

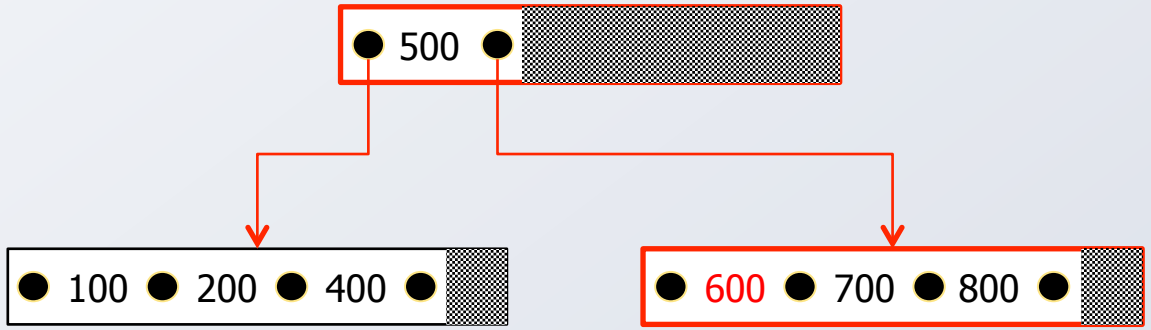
- Wurzel (= Blatt)

⇨ Einfügen von 600 erzeugt Überlauf

- Neuaufteilung von W



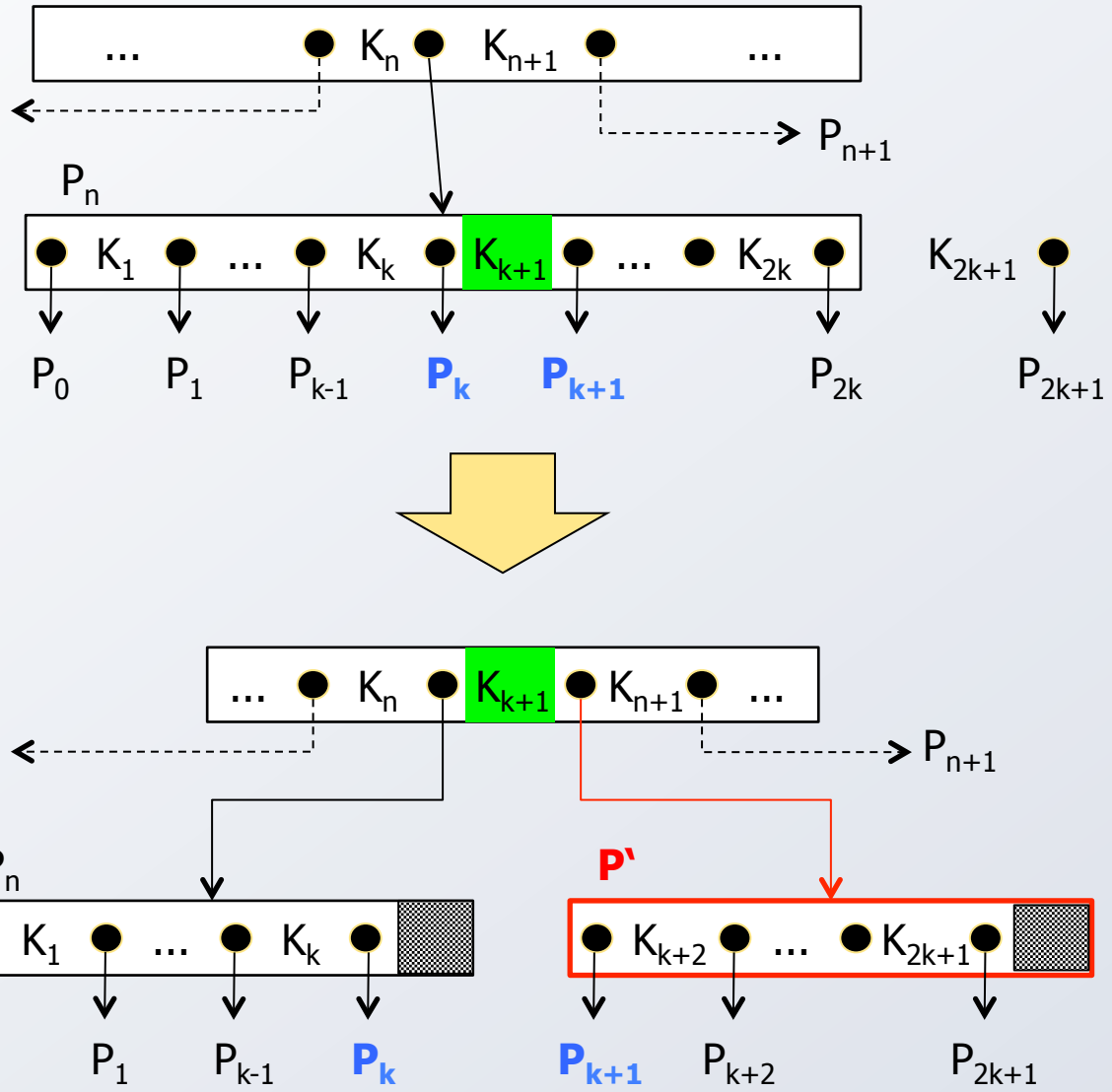
- B-Baum nach Split-Vorgang



B-Bäume - Einfügealgorithmus

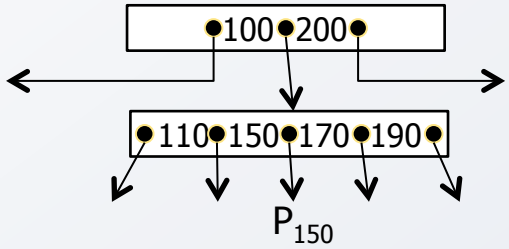
- Ermittle Blattknoten
  - Suche nach dem Knoten P_n, in dem das neue Element gespeichert werden soll (immer ein Blatt)
  - Füge neues Element K_j in den ermittelten Knoten P_n ein
    - ElementEinfügen(P_n, K_j, ∅)
- ElementEinfügen(P, K, V)
  - Suche Einfügeposition für K in P
  - Wenn Platz vorhanden ist, speichere Element K und V, sonst schaffe Platz durch Split-Vorgang und füge ein.
    - Fordere neue Seite (P‘) an
    - Ermittle Median K_{k+1} für 2k+1 Schlüsseleinträge (inkl. K_j)
    - Speichere Elemente mit $K < K_{k+1}$ in P, $K > K_{k+1}$ in P‘
    - Füge K_{k+1} und Verweis auf P‘ in die Vaterseite ein, falls dies vorhanden ist: ElementEinfügen(Vater(P), K_{k+1}, V(P‘))
    - Ansonsten: neue Wurzelseite mit V(P), K_{k+1}, V(P‘)
- ⇨ Split-Vorgang als allgemeines Wartungsprinzip

B-Bäume – Allgemeiner Splitvorgang



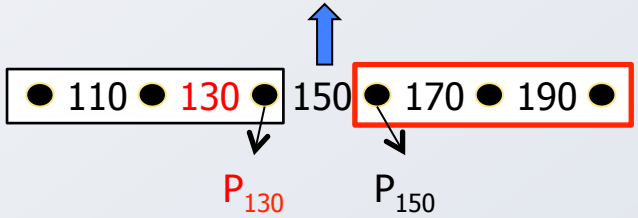
B-Bäume – Split-Beispiel

- Split-Vorgang – rekursives Schema

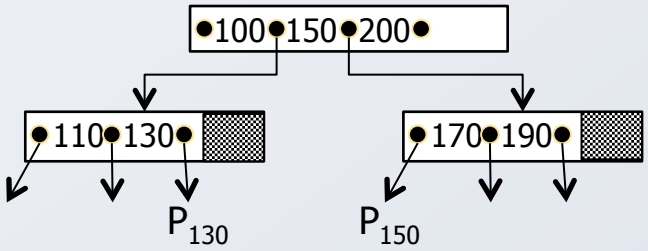


  - ⇨ Einfügen von 130 mit P130

- Neuaufteilung
  des kritischen Knotens



- Baumausschnitt nach Split-Vorgang



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

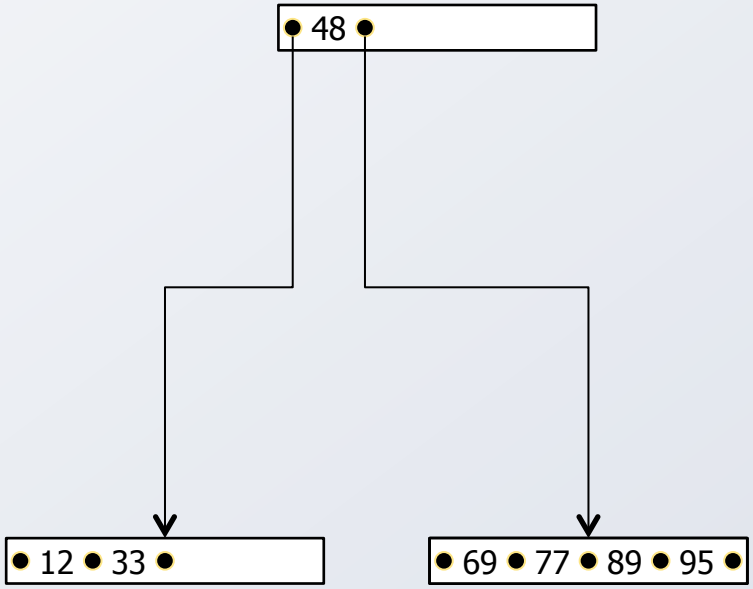
Einfügereihenfolge: **77 12 48 69 33 89 95 90 37 45 83 2 15 87 97 98 99 50**

Aufbau eines B-Baumes der Klasse $\tau(2, h)$ (1)

Einfügereihenfolge: 77 12 48 69 **33 89 95 90 37 45 83 2 15 87 97 98 99 50**

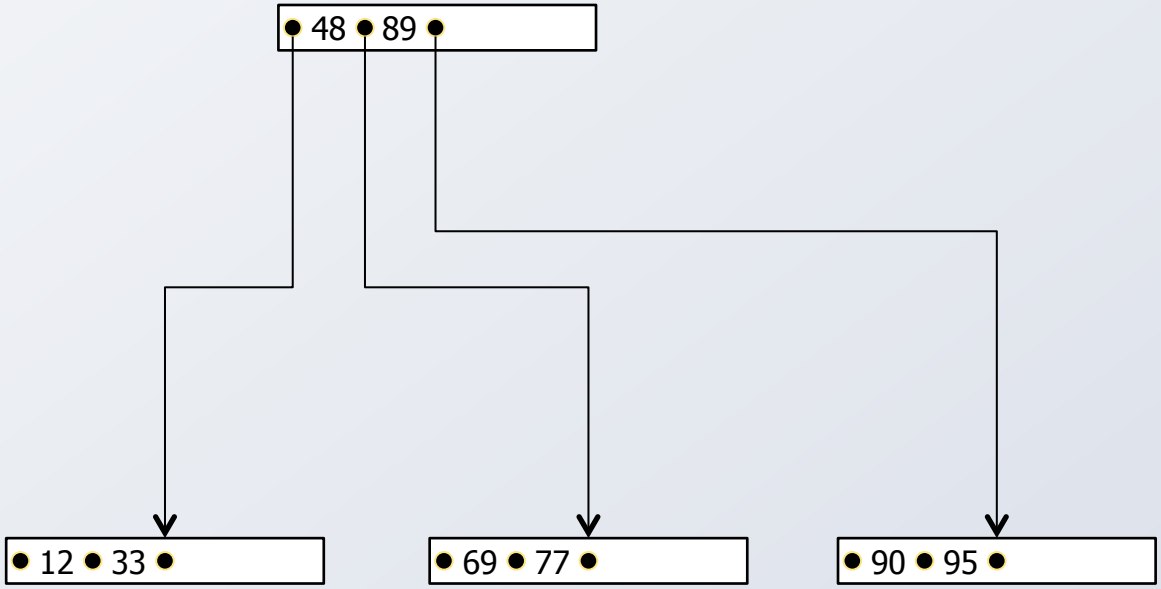
Aufbau eines B-Baumes der Klasse $\tau(2, h)$ (2)

Einfügereihenfolge: 77 12 48 69 33 89 95 **90 37 45 83 2 15 87 97 98 99 50**



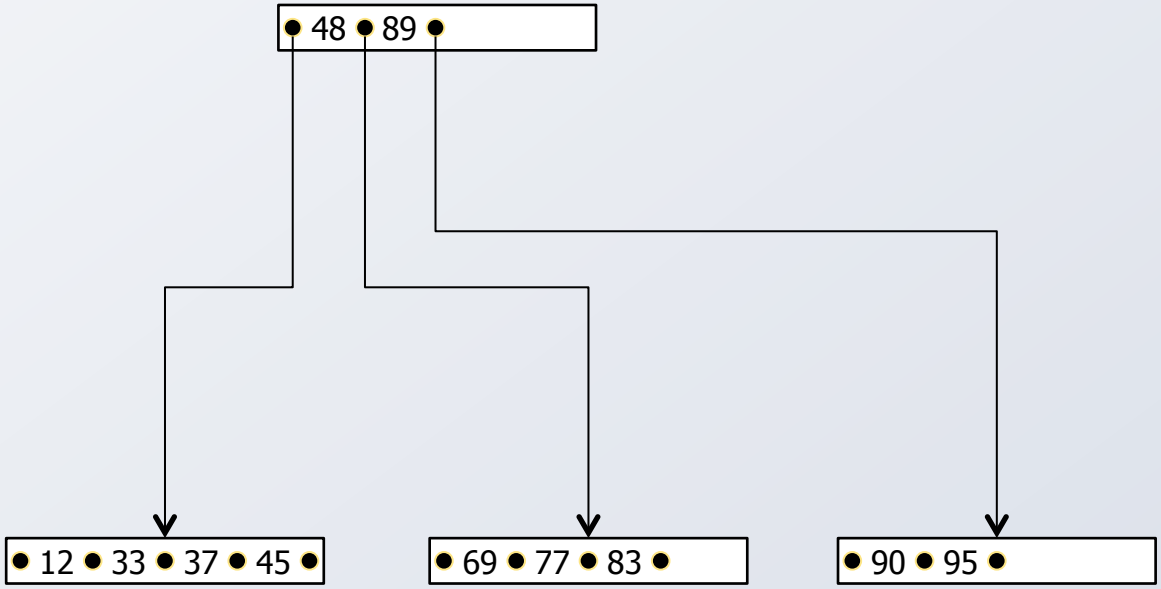
Aufbau eines B-Baumes der Klasse $\tau(2, h)$ (3)

Einfügereihenfolge: 77 12 48 69 33 89 95 90 **37 45 83 2 15 87 97 98 99 50**



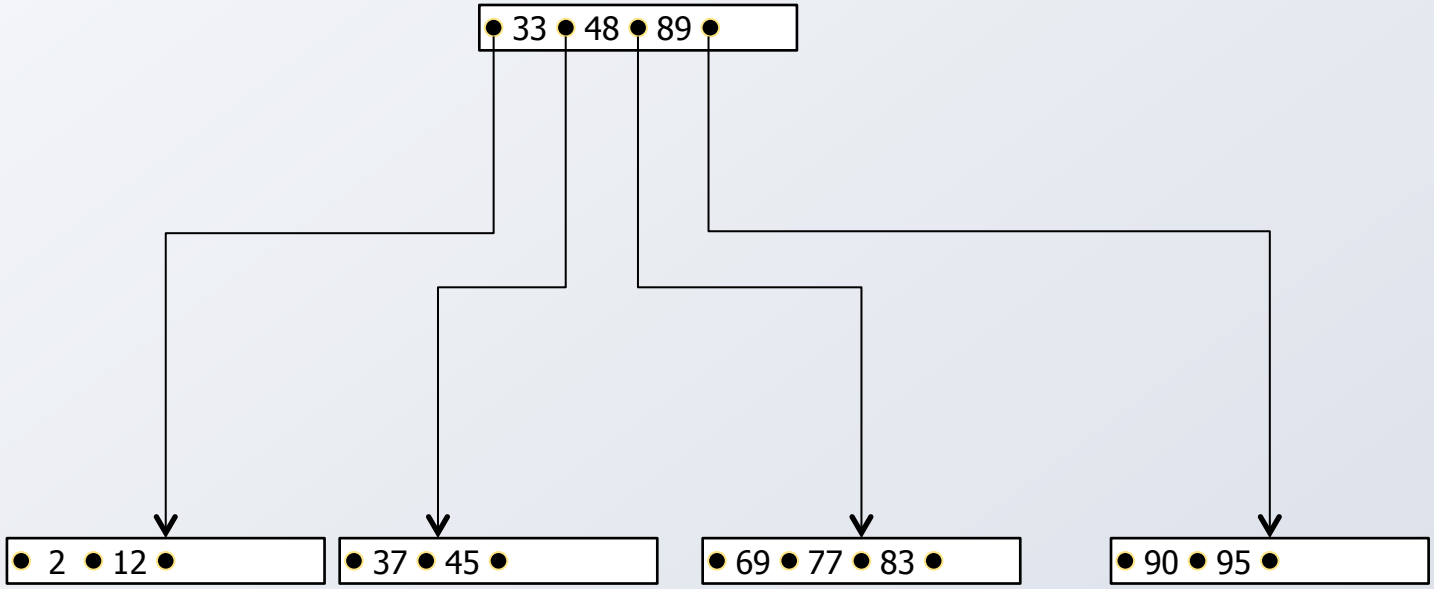
Aufbau eines B-Baumes der Klasse $\tau(2, h)$ (4)

Einfügereihenfolge: 77 12 48 69 33 89 95 90 37 45 83 **2 15 87 97 98 99 50**



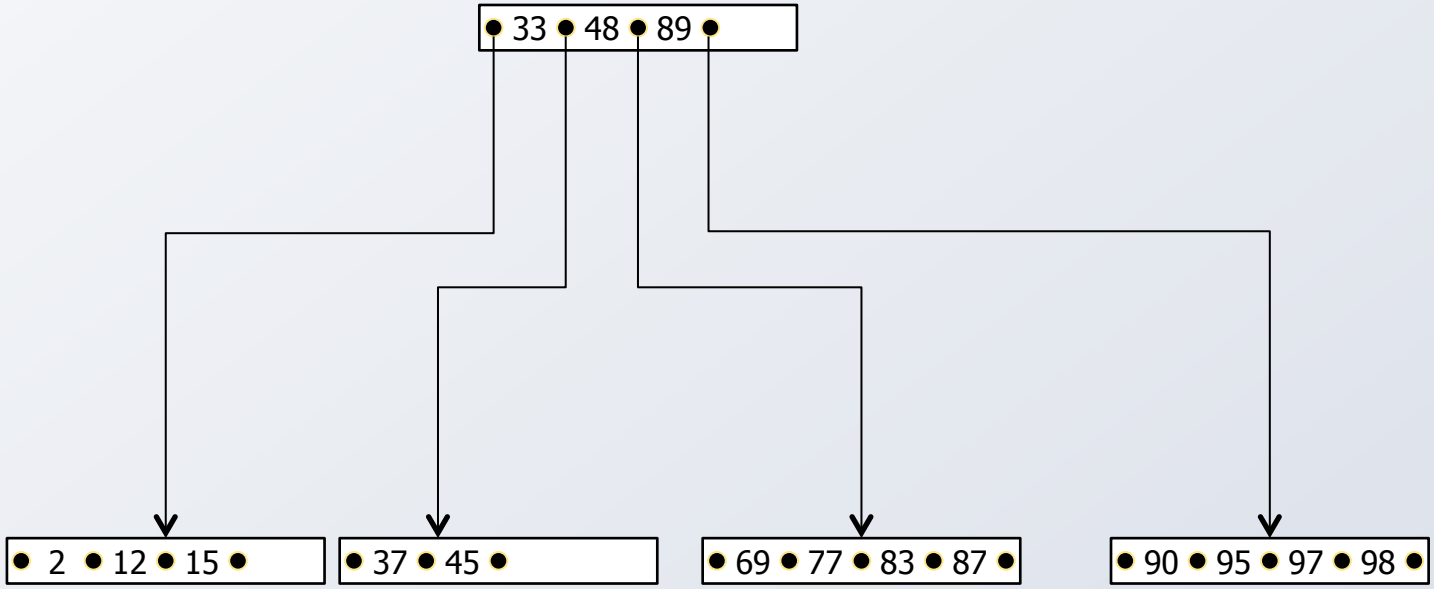
Aufbau eines B-Baumes der Klasse $\tau(2, h)$ (5)

Einfügereihenfolge: 77 12 48 69 33 89 95 90 37 45 83 2 **15 87 97 98 99 50**



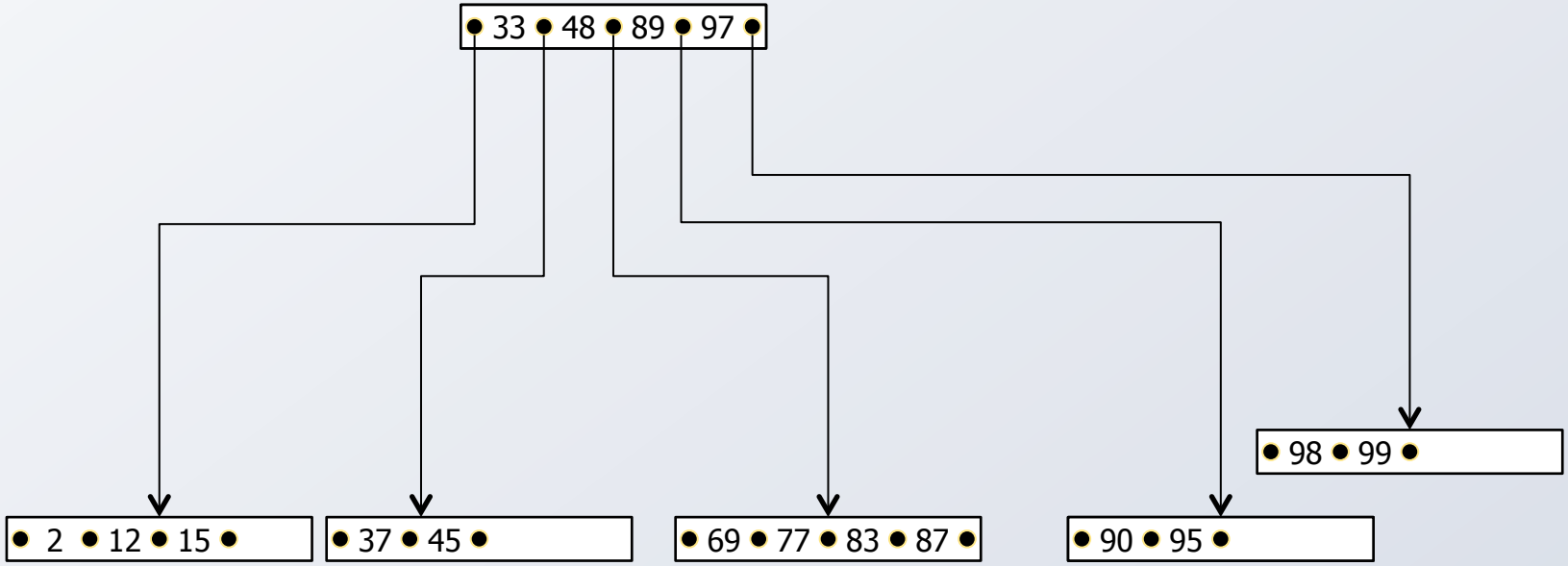
Aufbau eines B-Baumes der Klasse $\tau(2, h)$ (6)

Einfügereihenfolge: 77 12 48 69 33 89 95 90 37 45 83 2 15 87 97 98 **99** **50**



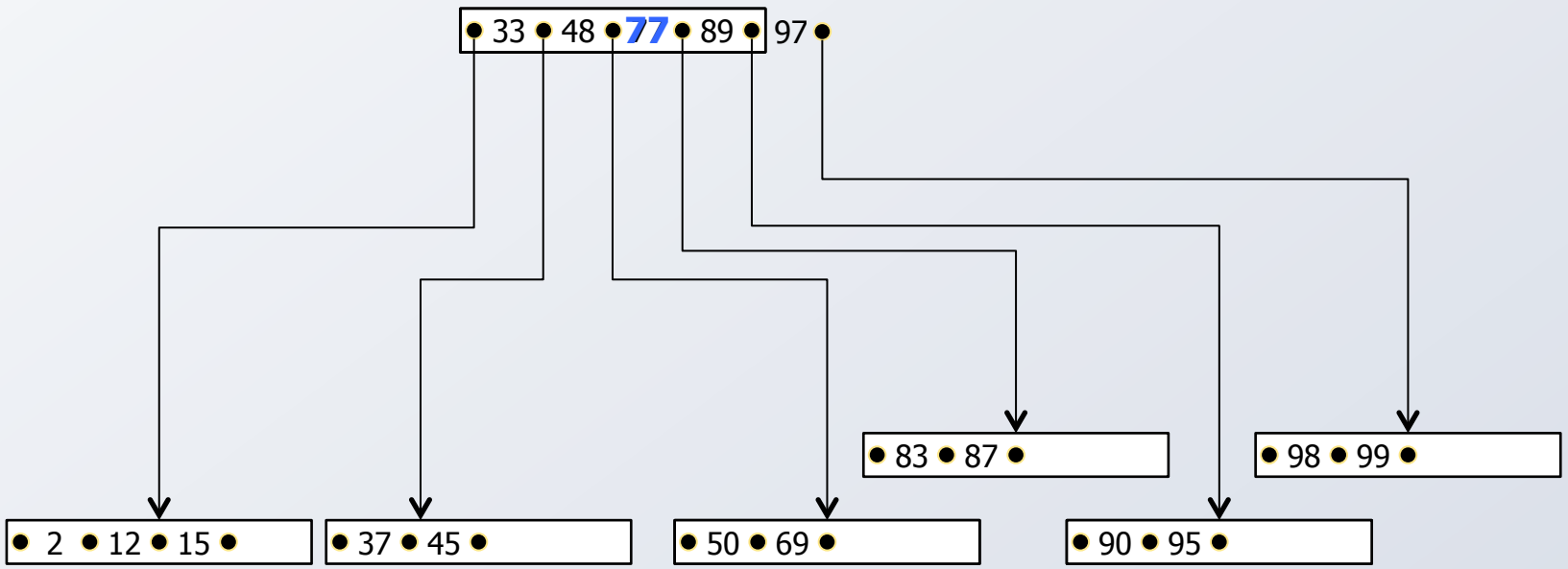
Aufbau eines B-Baumes der Klasse $\tau(2, h)$ (7)

Einfügereihenfolge: 77 12 48 69 33 89 95 90 37 45 83 2 15 87 97 98 99 **50**



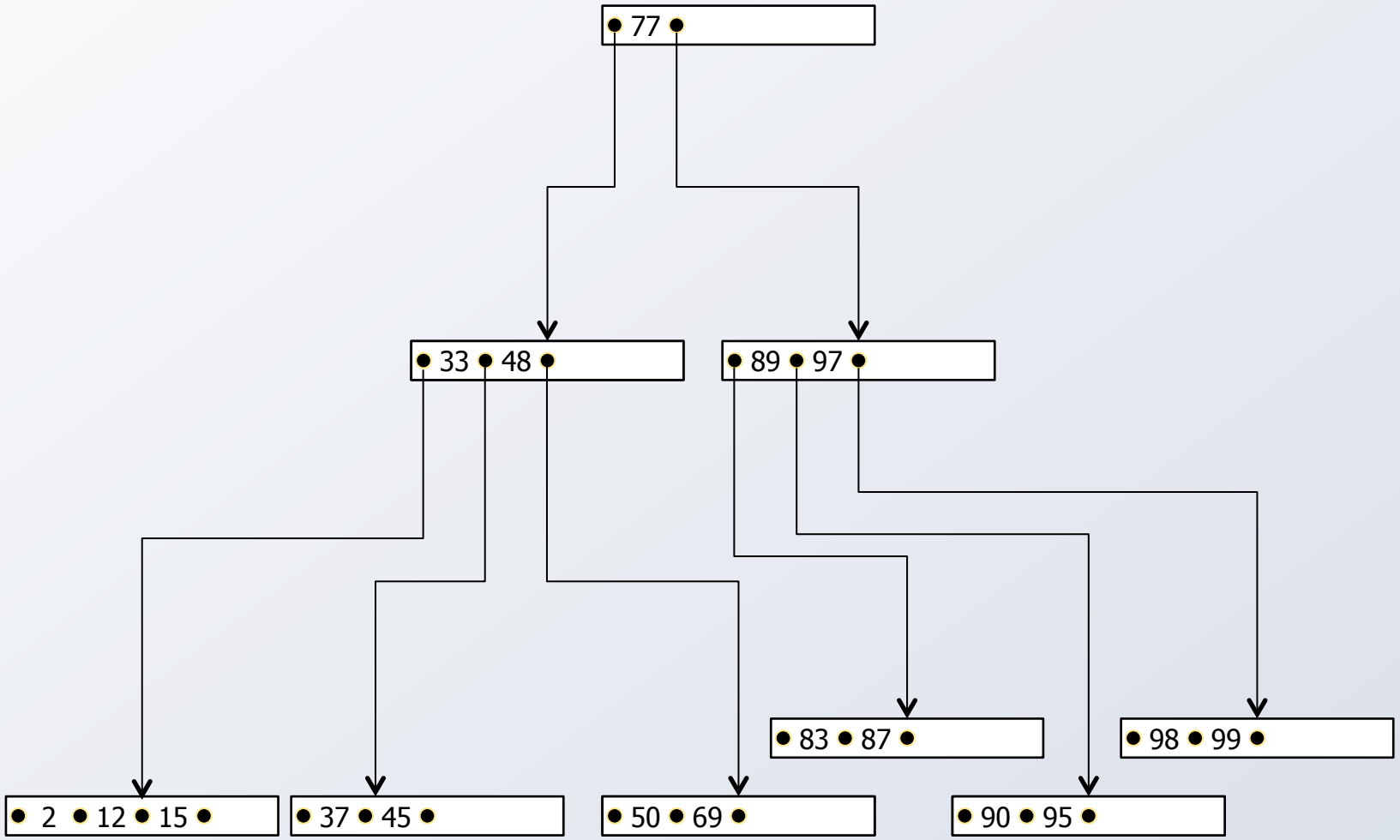
Aufbau eines B-Baumes der Klasse $\tau(2, h)$ (8)

Einfügereihenfolge: 77 12 48 69 33 89 95 90 37 45 83 2 15 87 97 98 99 **50**



Aufbau eines B-Baumes der Klasse $\tau(2, h)$ (9)

Einfügereihenfolge: 77 12 48 69 33 89 95 90 37 45 83 2 15 87 97 98 99 50



B-Bäume - Kosten für das Suchen

- 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 ≈ 100 – 200) möglich)

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

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

- 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!

B-Bäume - Kosten für das Einfügen

- Kostenanalyse für Einfügen
  - günstigster Fall – kein Split-Vorgang: $f_{min}= h$; $w_{min}= 1$;
  - ungünstigster Fall: $f_{max}= h$; $w_{max}= 2h + 1$;
    - Änderung der Höhe, d.h., Split für alle Seiten im Pfad (2h) plus neue Wurzel
  - durchschnittlicher Fall: $f_{avg}= h$; $_{wavg}= 1 + 2/k$;
    - Split erfolgt mit Wahrscheinlichkeit 1/k, dann 2 zusätzliche Seiten zu schreiben (neue Seite und Vaterknoten)

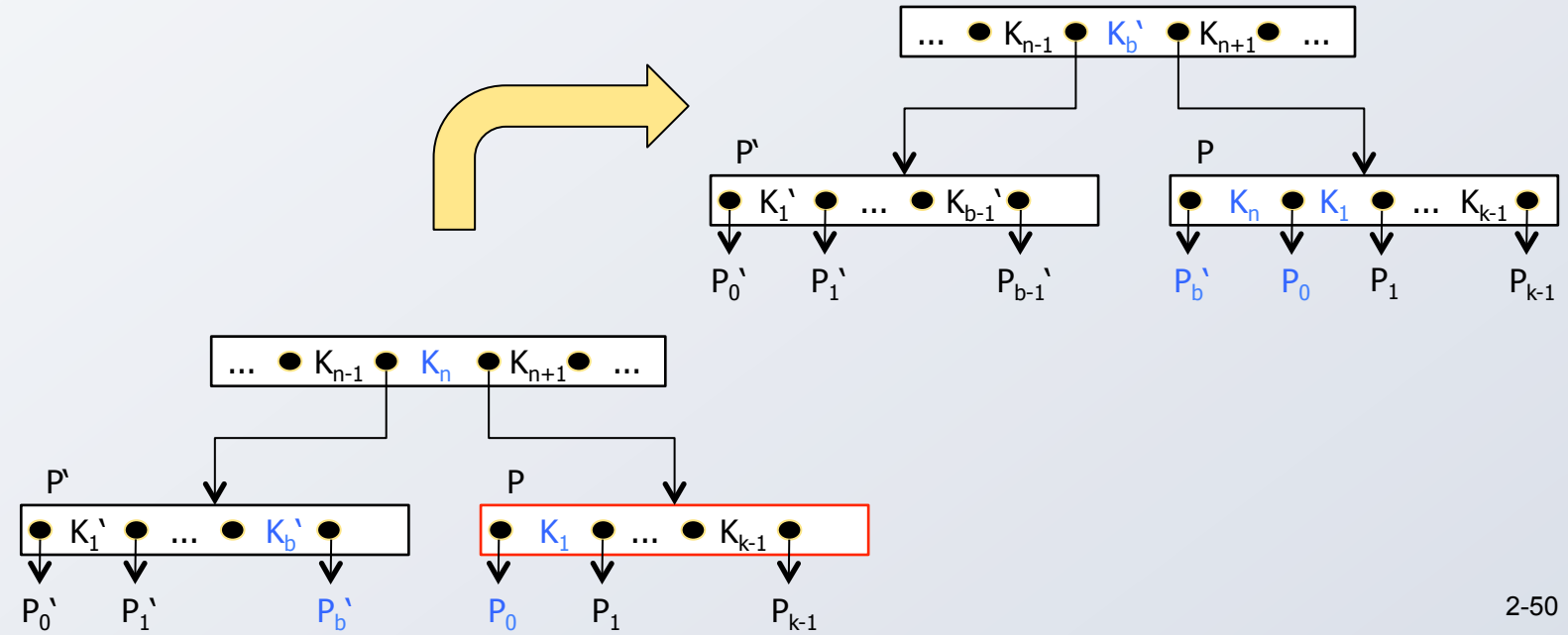
➪ Eine Einfügung kostet bei k=100 im Mittel $w_{avg} < 1 + 2/100$ Schreibvorgänge, d.h., es entsteht eine Belastung von 2% für den Split-Vorgang.

B-Bäume - Löschen

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.

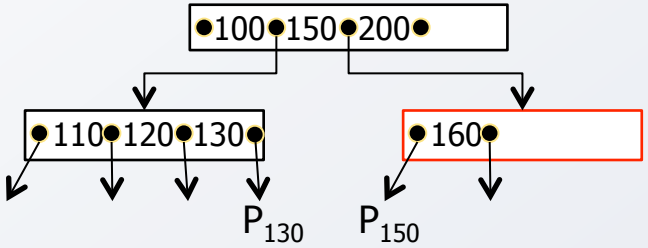
(1) Maßnahme: Ausgleich durch Verschieben von Schlüsseln

Beim Ausgleichsvorgang sind in der Seite P k–1 Elemente und in P' mehr als k Elemente.



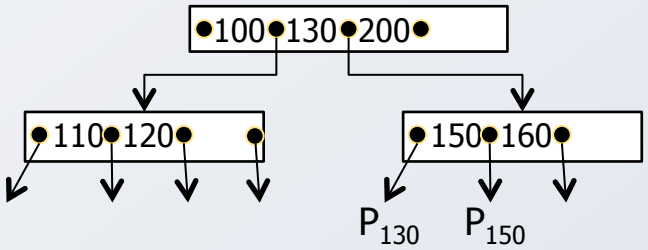
B-Bäume – Löschbeispiele

- Löschprinzip: Ausgleich



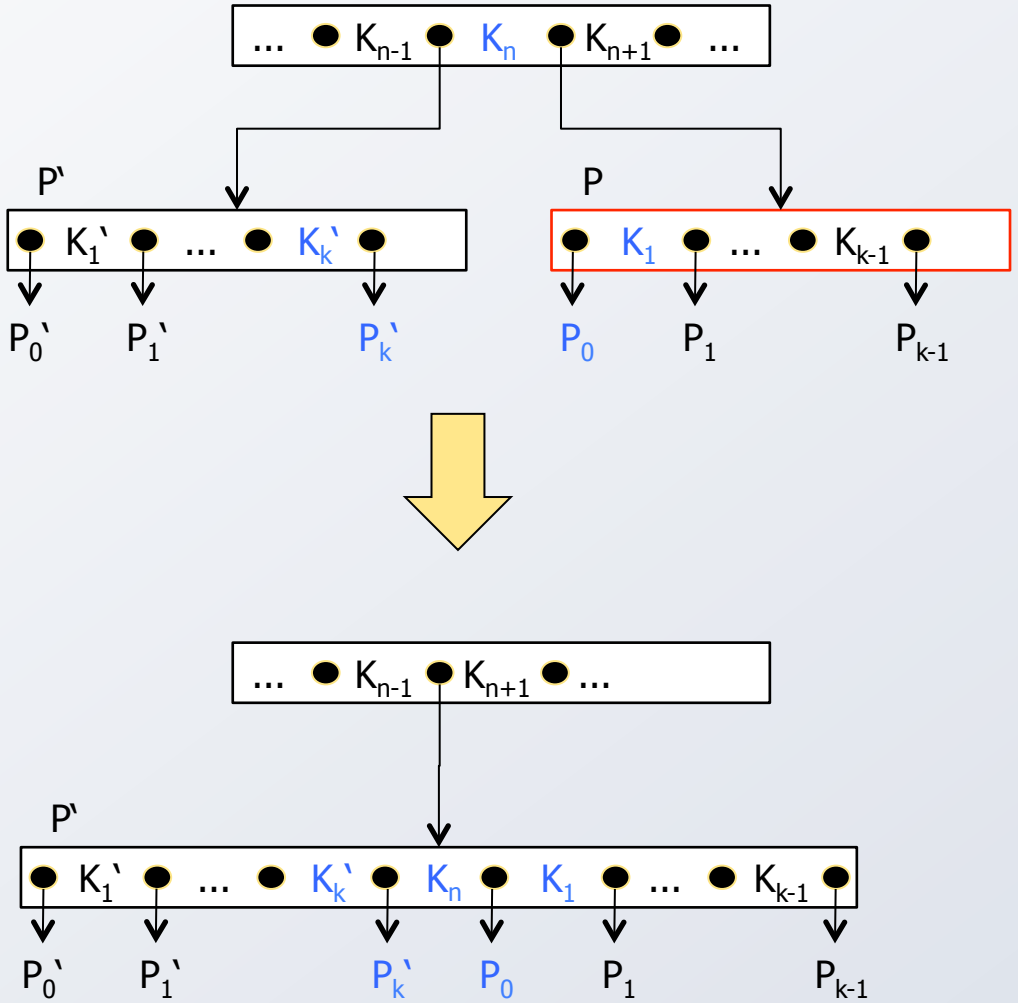
⇨ $b_1 > k$ und $b_2 = k - 1$

- Baumausschnitt nach Ausgleich



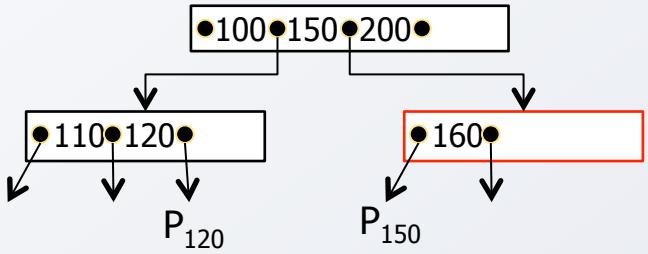
B-Bäume – Löschen (2)

(2) Maßnahme: Mischen von Seiten

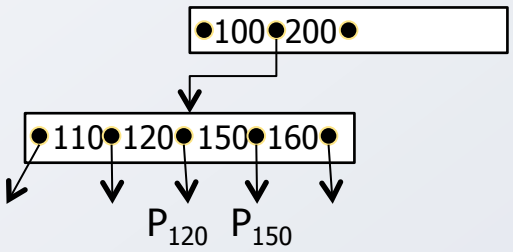


B-Bäume – Löschbeispiele (2)

- Löschprinzip: Mischen



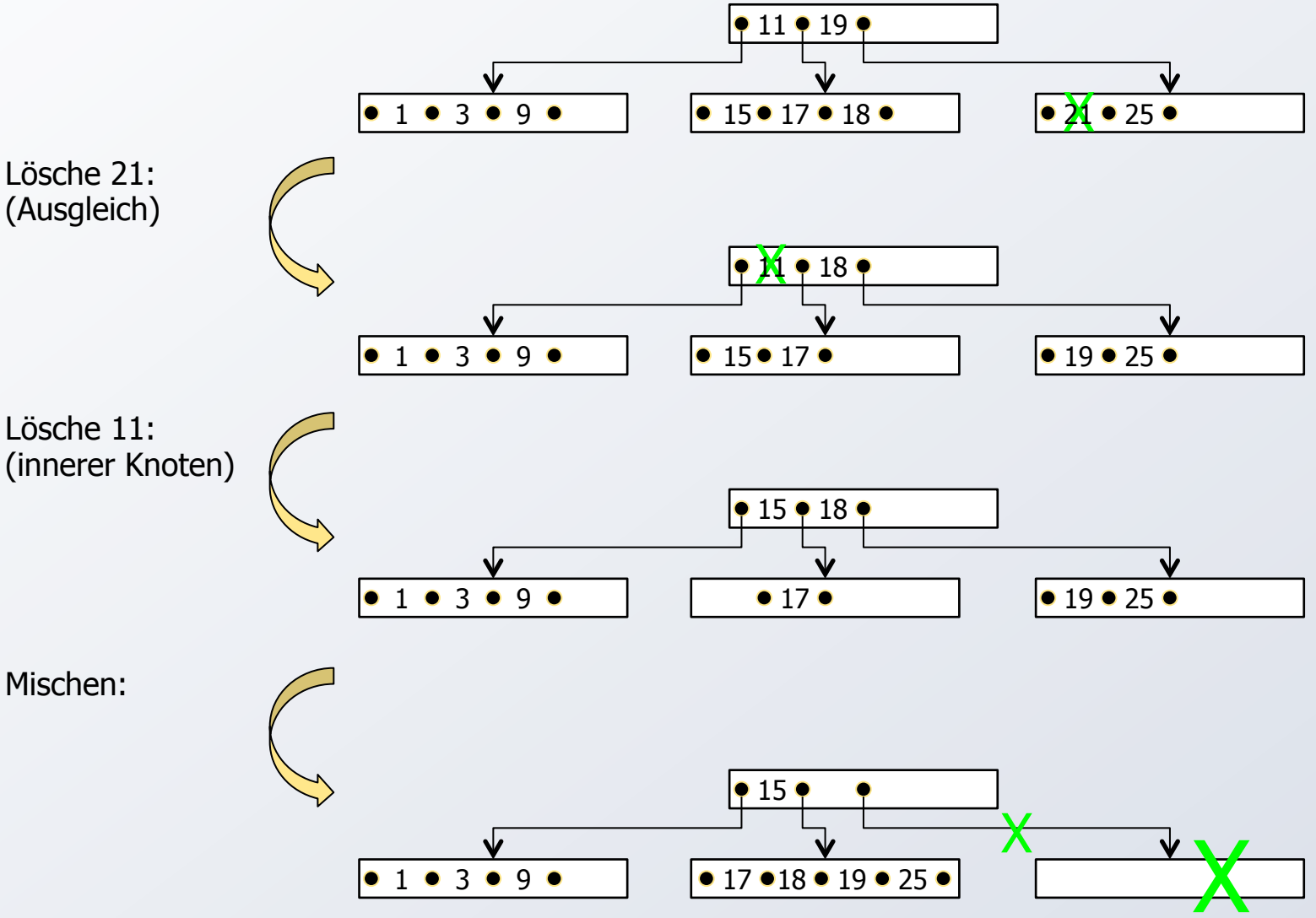
- Baumausschnitt nach Mischen



B-Bäume - 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 (3)



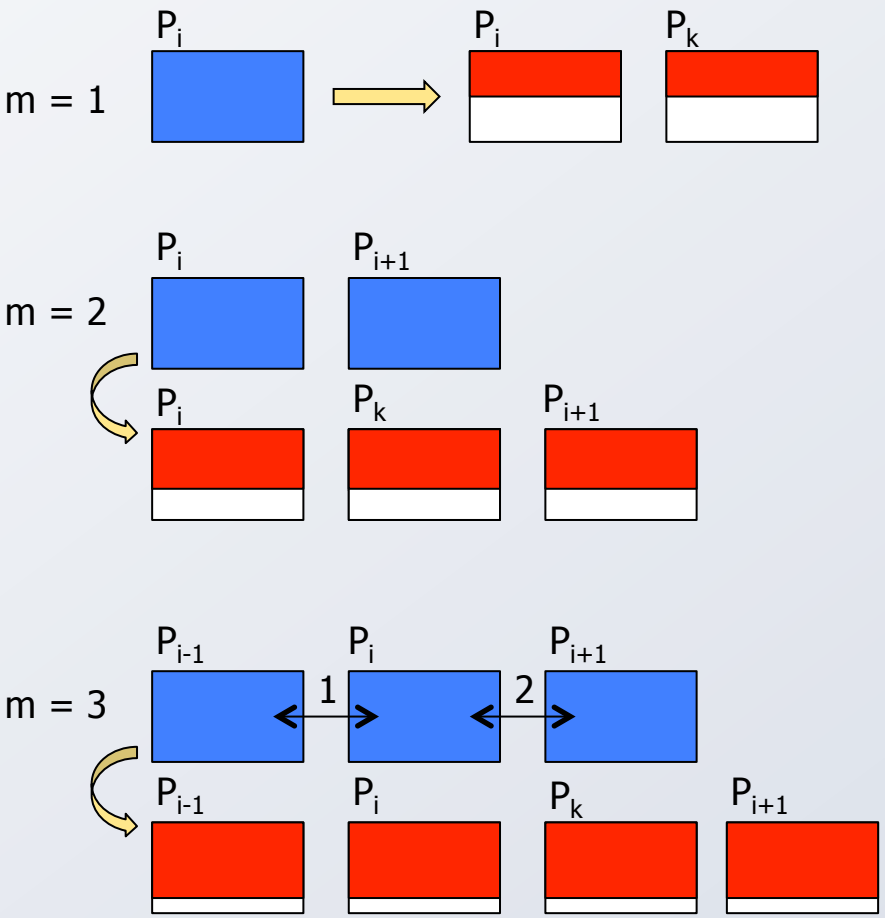
B-Bäume – Kosten für das Löschen

- Kostenanalyse
  - günstigster Fall: $f_{min} = h$; $w_{min} = 1$
    Löschen im Blatt ohne Unterlauf
  - ungünstigster Fall (pathologisch): $f_{max} = 2h - 1$; $w_{max} = h + 1$
    Auf allen Ebenen Mischen, bei der Wurzel Ausgleichen
  - obere Schranke für durchschnittliche Löschkosten (drei Anteile: 1. Löschen, 2. Ausgleich, 3. anteilige Mischkosten):
    - $f_{avg} \leq f_1 + f_2 + f_3 < h + 1 + 1/k$
    - $w_{avg} \leq w_1 + w_2 + w_3 < 2 + 2 + 1/k = 4 + 1/k$

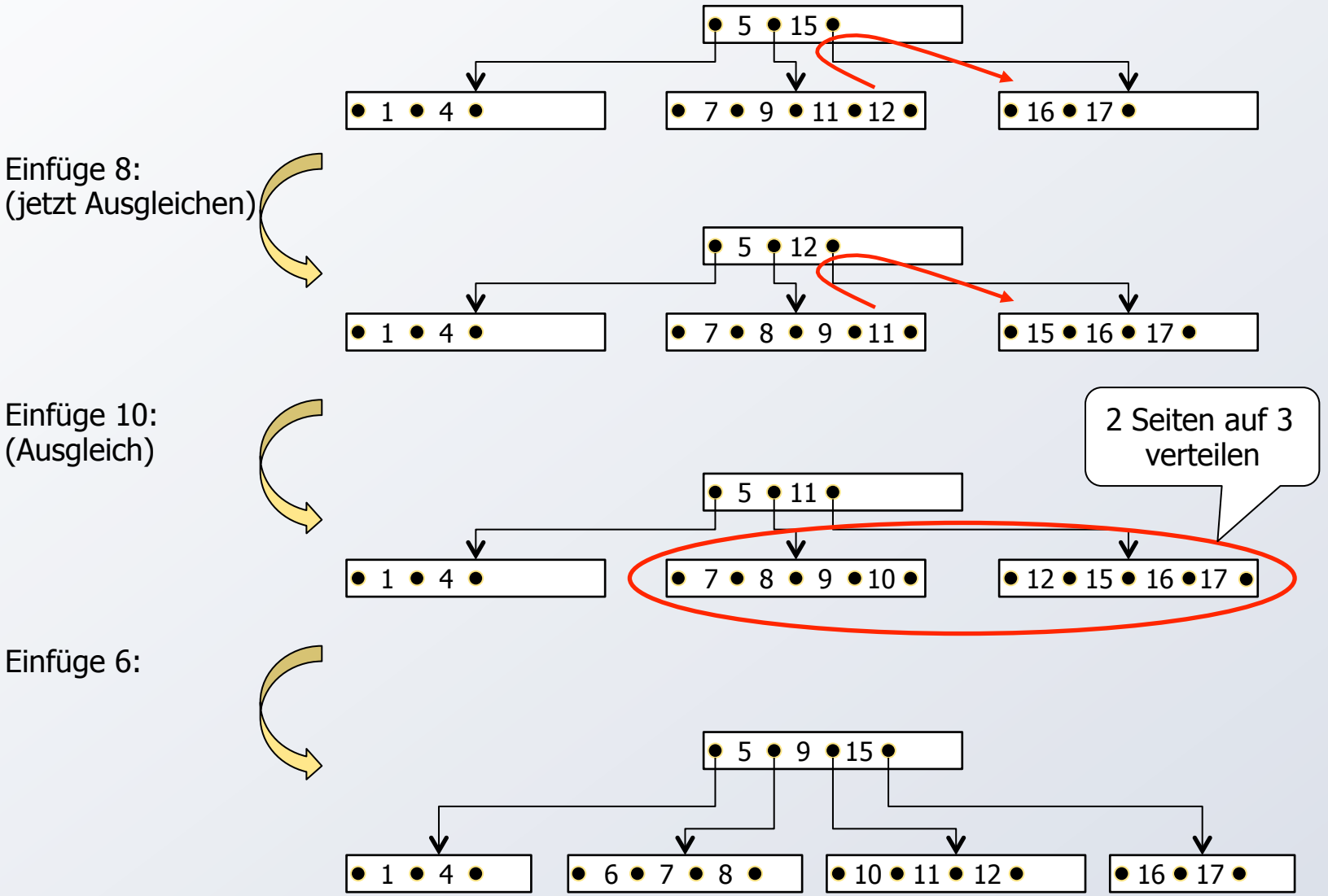
B-Bäume – Überlaufbehandlung

- Verbesserung des Belegungsgrades
  - verallgemeinerte Überlaufbehandlung
  - Split-Faktor m > 1

⇨ Verbesserte Speicherplatzbelegung, dafür höhere Einfügekosten



B-Bäume – Beispiel mit m = 2



B-Bäume – Kosten bei m=2

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

Ausgleich: min + 1 Lesen + 2 Schreiben

Splitten: min + 2 Lesen + 3 Schreiben

B-Bäume – Belegungsgrad

- Welcher Belegungsgrad β des B-Baums wird erzielt
  - bei einfacher Überlaufbehandlung (Split-Faktor m = 1)?
  - beim Einfügen einer sortierten Schlüsselfolge?
- Speicherplatzbelegung als Funktion des Split-Faktors

Split-Faktor	Belegung β_{min}	β_{avg}	β_{max}
1	1/2 = 50%	**ln 2 ≈ 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.

B*-Bäume - Definition

- Definition*:

  Seien k, k* und h* ganze Zahlen, h* ≥ 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:

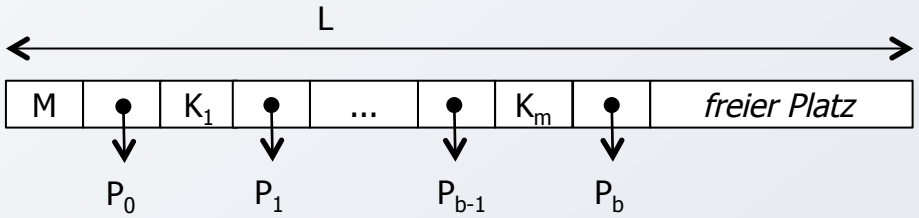
  1. Jeder Pfad von der Wurzel zu einem Blatt besitzt die gleiche Länge h*–1.
  2. Jeder Knoten außer der Wurzel und den Blättern hat mindestens k+1 Söhne, die Wurzel mindestens 2 Söhne, außer wenn sie ein Blatt ist.
  3. Jeder innere Knoten hat höchstens 2k+1 Söhne.
  4. Jeder Blattknoten mit Ausnahme der Wurzel als Blatt hat mindestens k* und höchstens 2k* Einträge.

(*) Der so definierte Baum heißt in der Literatur gelegentlich auch B^+-Baum.

B*-Bäume - Knotenformate

- Unterscheidung von zwei Knotenformaten:

Innere Knoten
$k \le b \le 2k$



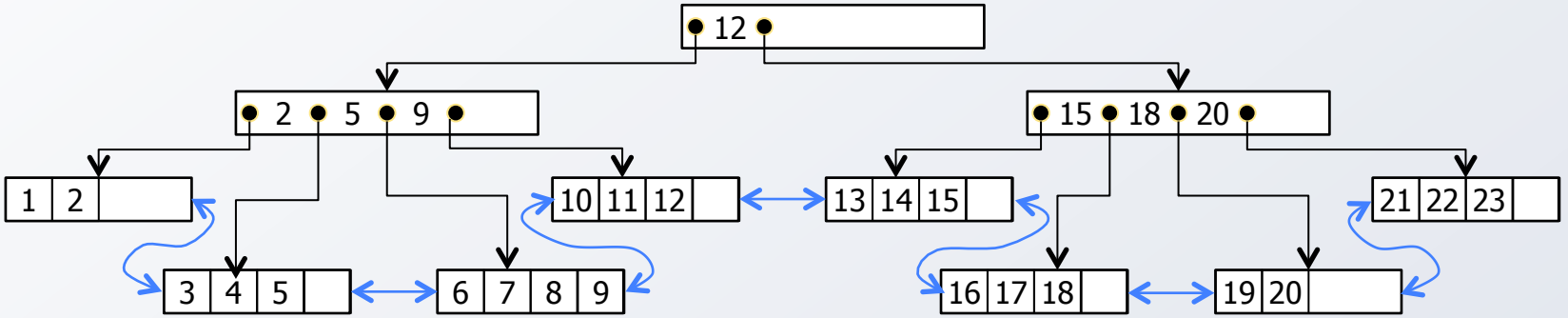
Blattknoten
$K^* \le m \le 2k^*$

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:

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

B*-Bäume – Beispiel

- B*-Baum der Klasse τ (3, 2, 3)



- Was sind typische Größen in B*-Bäumen?

(in Byte:) L = 8192, l_M= 4, l_P = 4, l_K = 8, l_D = 92

Innerer Knoten

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

$$= \left\lfloor \frac{4096 - 4 - 4}{2 * (8 + 4)} \right\rfloor = \left\lfloor \frac{4084}{24} \right\rfloor = 170$$

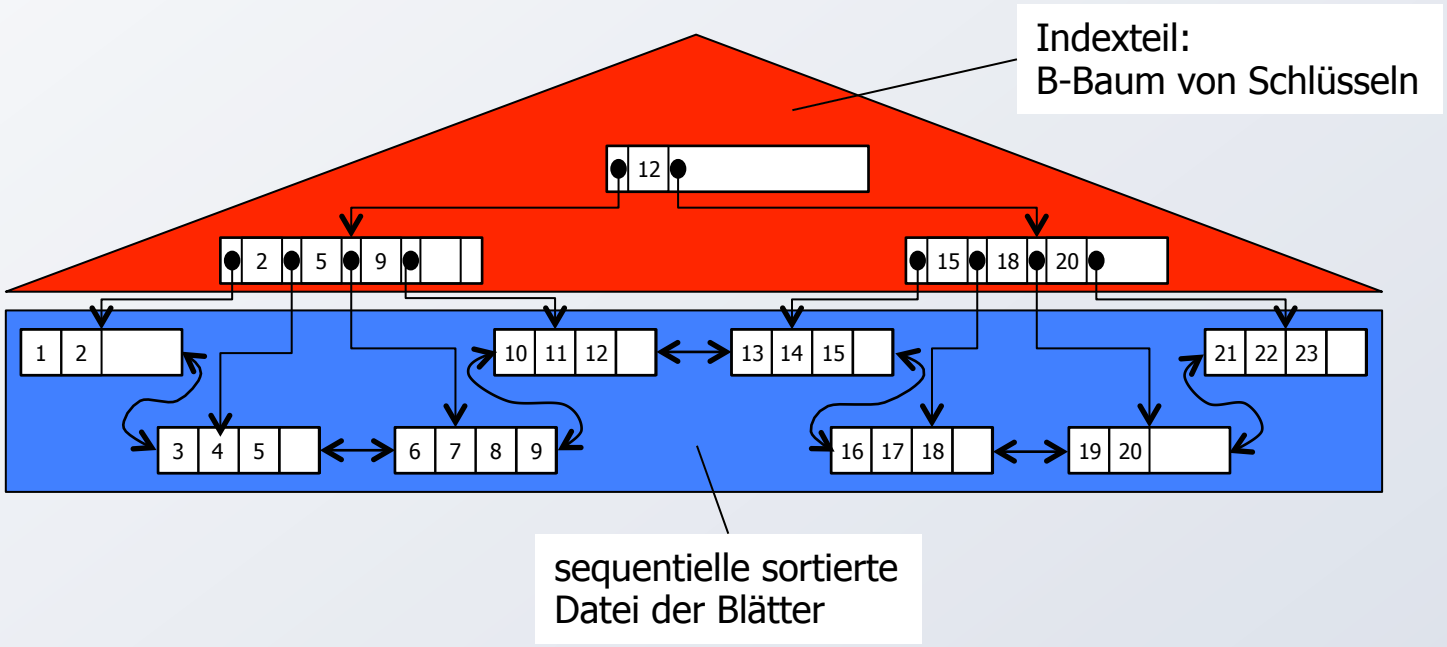
Blatt

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

$$= \left\lfloor \frac{4096 - 4 - 8}{2 * (8 + 92)} \right\rfloor = \left\lfloor \frac{4084}{200} \right\rfloor = 20$$

B*-Bäume - Erklärungsmodell

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.



B*-Bäume - Grundoperationen

(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.

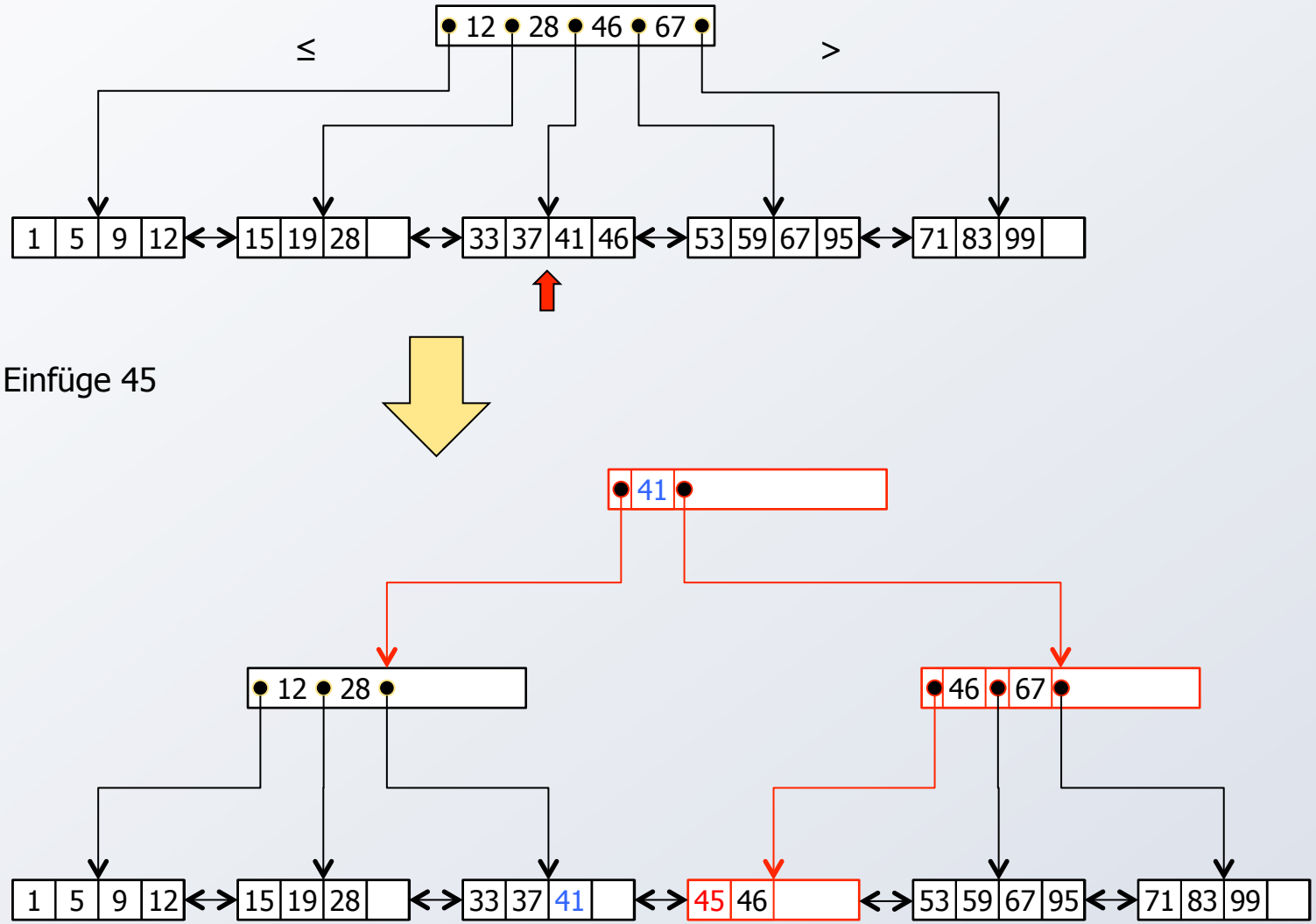
(3) Einfügen:

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.

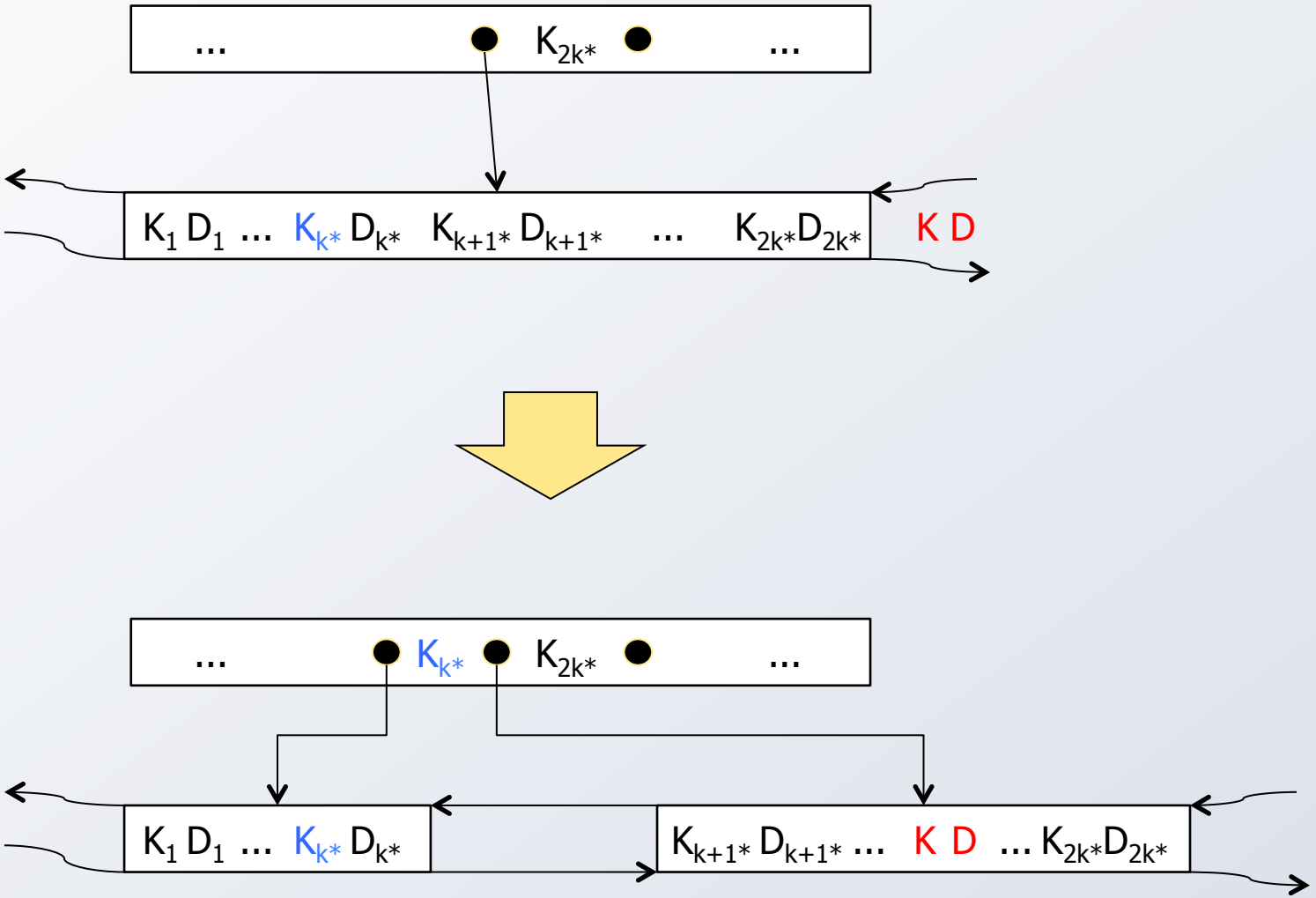
(4) Löschen:

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.

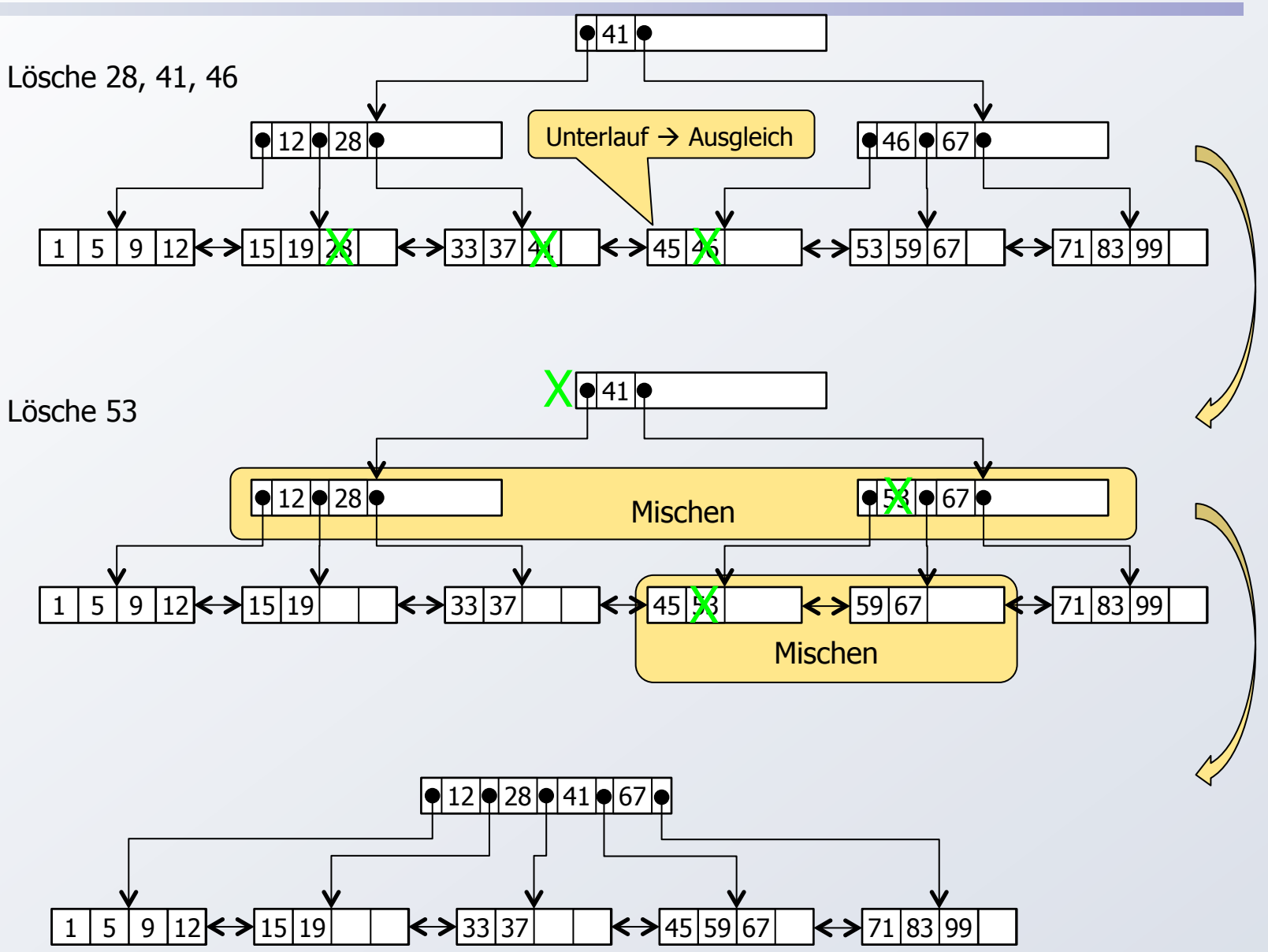
B*-Bäume - Einfügebeispiel



B*-Bäume – Allgemeiner Splitvorgang



B*-Bäume - Löschen



B*-Bäume - Schlüsselkomprimierung

- 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
  - Variabel lange Einträge erfordern Kontrolle des Unterlaufs über die tatsächliche Speicherplatzbelegung

B*-Bäume – Schlüsselkomprimierung (2)

- 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	-

tatsächlich gespeichert

➪ Durchschnittliche komprimierte Schlüssellänge ca. 1.3 – 1.8

B*-Bäume – Komprimierungsbeispiel

- Präfix-Komprimierung

  F, L, **komp**. Schlüssel:
  0, 10, **Hildebrand**
  0, 4, **Kehl**
  1, 3, K**los**
  4, 1, Klos**e**
  0, 4, **Lahm**
  1, 6, L**ehmann**
  0, 5, **Neuer**

- Präfix-Suffix-Komprim.

  F, L, **komp**. Schlüssel:
  0, 1, **H**ildebrand
  0, 2, **Ke**hl
  1, 4, K**los_**
  4, 0, Klose
  0, 2, **La**hm
  1, 0, Lehmann
  0, 1, **N**euer

- Suche „Lahm“

  Im Index (expandiert):
  1. Lahm > Hildebrand
  2. Lahm > Kehl
  3. Lahm > Klos
  4. Lahm > Klose
  5. Lahm ≤ Lahm

- Suche „Lahm“

  Im Index (expandiert):
  1. L > H
  2. La > Ke
  3. Lahm > Klos_
  4. Lahm > Klos
  5. La ≤ La

B*-Bäume – Komprimierungsbeispiel (2)

- 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_T O_CITY	... RAFFER TTY , GERR Y	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_T O_WALHALLA	... JETHRO_TULL	6	4	5	0	
COLORADO	... STILLS, STEPHEN	4	5	3	2	OR
COLOURS	... DONOV AN	5	3	4	0	
COME_INSIDE	... COMMODORES	3	13	2	11	ME_INSIDE__
COME_INSIDE_OF_MY_GUIT AR	... BELLAMY_BROTHERS	13	6	12	0	
COME_ON_OVER	... BEE_GEES	6	6	5	1	O
COME_T OGETHER	... BEA TLES	6	4	5	0	
COMING_INT O_LOS_ANGELES	... GUTHERIE, ARLO	4	4	3	1	I
COMMOTION	... CCR	4	4	3	1	M
COMP ARED_T O_WHA T?	... FLACK, ROBER TA	4	3	3	0	
CONCLUSION	... ELP	3	4	2	2	NC
CONFUSION	... PROCOL_HARUM	4	1	3	0	

➪ Welche Platzeinsparung lässt sich erzielen?
> 95%!

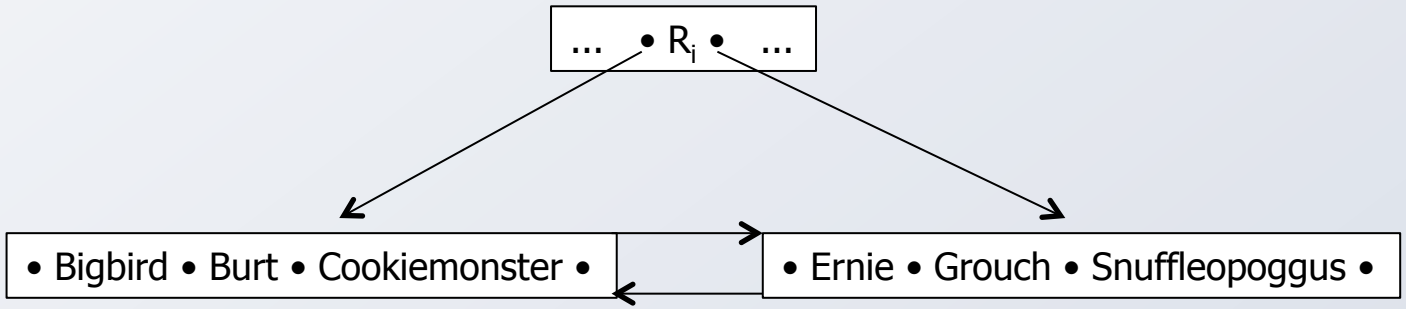
B*-Bäume – Präfix-B-Bäume

Wegen der ausschließlichen Wegweiser-Funktion der Referenzschlüssel R_i ist es nicht nötig, die Optimierungs-maßnahmen 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



Höhe des B*-Baumes

1. Bei minimaler Belegung
   - Die Anzahl der Blattknoten 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 Anzahl von Elementen erhalten wir

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

2. Bei maximaler Belegung
   - Anzahl der Blattknoten

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

   - Anzahl der gespeicherten Elemente

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

- Deshalb ist die Höhe h* eines B*-Baumes mit n Datenelementen begrenzt ist durch

$$\underbrace{1+\log_{2k+1}\frac{n}{2k^*}}_{2} \leq h^* \leq \underbrace{2+\log_{k+1}\frac{n}{2k^*}}_{1} \quad \text{für } h^* \geq 2$$

Zugriff auf eine Satzmenge

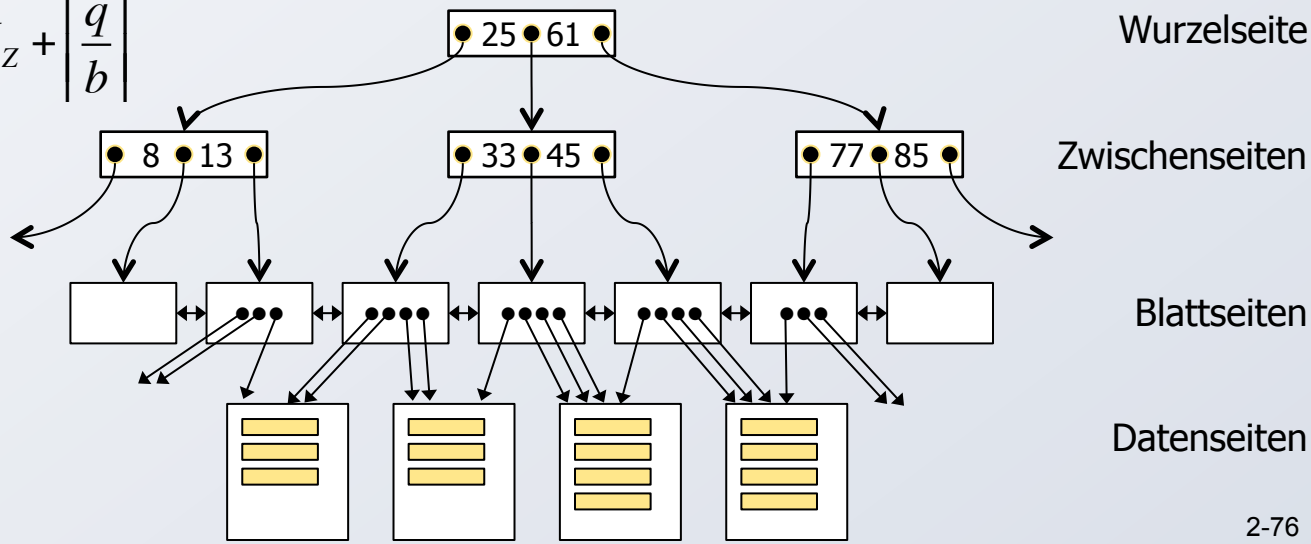
- Einfaches Erklärungs- und Kostenmodell
  - m = #Seiten im Segment, N_Z = #Seiten einer Zeigerliste
  - b = mittlere #Sätze/Seite, q = #Treffer der Anfrage
- Datei- oder Segment-Scan

$$N_S = m$$

Datenseiten

- Index-Scan mit Cluster-Bildung

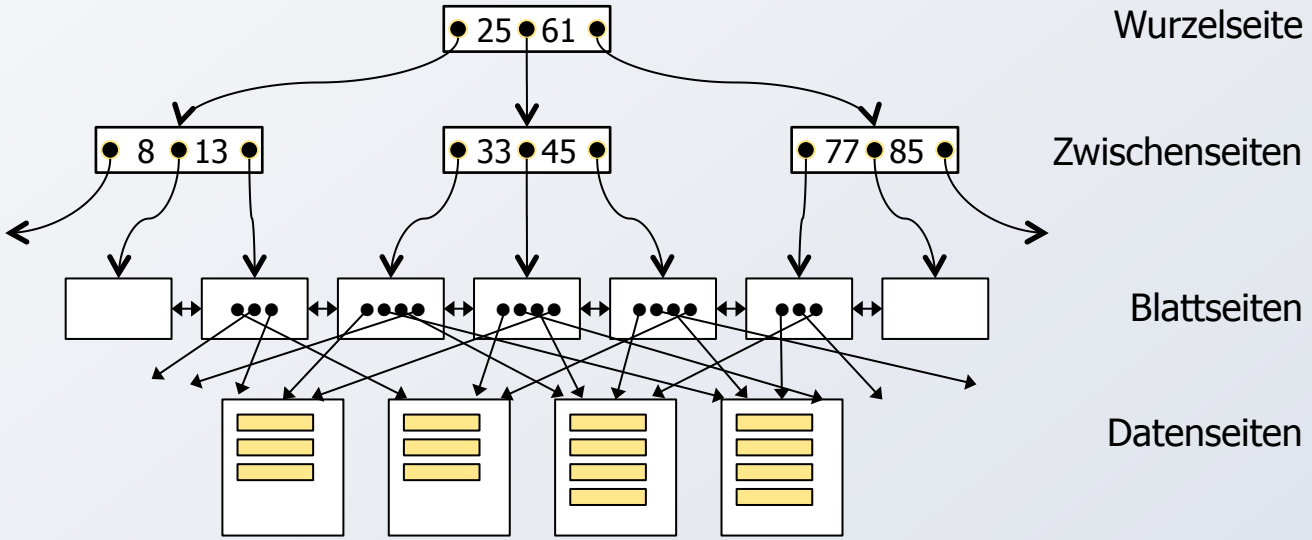
$$N_S \approx h + N_Z + \left\lceil \frac{q}{b} \right\rceil$$



Zugriff auf eine Satzmenge (2)

- Index-Scan ohne Cluster-Bildung

$$N_S \approx h + N_Z + q$$



Informationssuche – strukturierte Daten

- DB-Beispiel

ANGESTELLTER

PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	Peinl	Pförtner	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

Informationssuche – sequentielle Suche

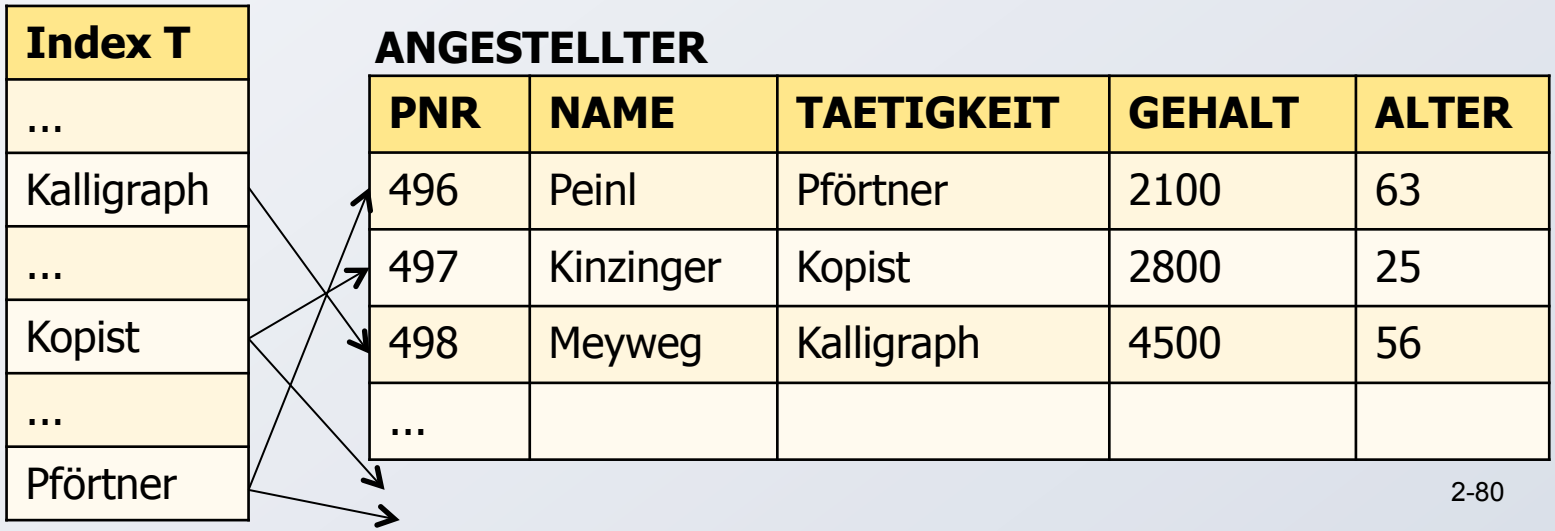
- Annahmen
  - N_S = Anzahl physischer Seitenzugriffe
  - ANGESTELLTER hat n Sätze, in Datei/Segment mit m Seiten (z. B. $n = 10^6$, $m = 10^5$)
  - Speicherung:
    - als sequentielle Liste mit mittlerem Blockungsfaktor b
    - verstreut über alle m Seiten
- Sequentielle Suche
  (z. B. nach TAETIGKEIT = 'KALLIGRAPH')
  - Datei- oder Segment-Scan
  - $N_S = n/b$ oder $N_s = m$

Informationssuche mit Index

- 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, ...}



Informationssuche mit Index - Kosten

- 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)
  - $h_T = 2$, $j_T = 1000$, $n = 10^6$, $N_{ZT} = 1$
  - $N_S = h_T + N_{ZT} + n/j_T = 2 + 1 + 10^6/10^3 = 1003$
  - Cluster-Bildung bei T, b=100: $N_S = 2 + 1 + 1000/100 = 13$

  ➪ Indexierung eines Attributs wird bestimmt durch den erwarteten Leistungsgewinn bei der Anfrageauswertung

Informationssuche - Mehrattributindex

- Anfrage

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

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

Index T	
...	
Kalligraph	33
...	
Kalligraph	56
...	
Kopist	25

ANGESTELLTER

PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	Peinl	Pförtner	2100	63
497	Kinzinger	Kopist	2800	25
498	Meyweg	Kalligraph	4500	56
...				

Informationssuche - Kosten

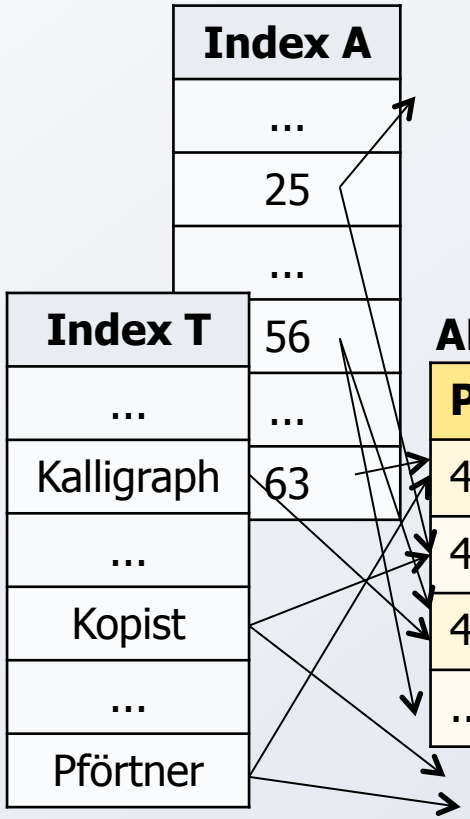
- Mehrattribut-Indexierung (Forts.)
  - $h_{TA} = 3$, $j_T = 1000$, $j_A = 50$, $j_{TA} = j_T \cdot j_A$, $n = 10^6$, $N_{ZTA} = 1$
  - $N_S = h_{TA} + N_{ZTA} + n/(j_T \cdot j_A)$
    $= 3 + 1 + 10^6/(5 \cdot 10^4) = 24$
  - N_S (Cluster-Bildung bei TA) $= 4 + 10^6/(5 \cdot 10^4 \cdot b) = 5$
  - erhöhter Spezialisierungsgrad bei der Indexnutzung
  - Nutzung von Index TA nur für TAETIGKEIT oder nur für ALTER?

Informationssuche – separate Indexe

- Anfrage

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

- Kombination von zwei Indexen: TAETIGKEIT und ALTER



ANGESTELLTER

PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	Peinl	Pförtner	2100	63
497	Kinzinger	Kopist	2800	25
498	Meyweg	Kalligraph	4500	56
...				

Informationssuche - Kosten

- Kombination von zwei separaten Indexen: TAETIGKEIT und ALTER (Forts.)
  - $h_T = h_A = 2$, $j_T = 1000$, $j_A = 50$, $n = 10^6$, $N_{ZT} = 1$, $N_{ZA} = 20$
    - 1000 Tupel mit T=„Kalligraph"
    - 20000 Tupel mit A = 50
  - Ansatz 1: Mischen der Sätze

    $N_S = h_T + N_{ZT} + n/j_T + h_A + N_{ZA} + n/j_A$

    $= 2 + 1 + 10^3 + 2 + 20 + 2 \cdot 10^4 = 21025!$
  - Ansatz 2: Ansatz 1 mit Clusterbildung bei A:

    $N_S = 2 + 1 + 10^3 + 2 + 20 + 2 \cdot 10^2 = 1225$
  - Ansatz 3: Mischen der Zeigerlisten im HSP:

    $N_S = h_T + N_{ZT} + h_A + N_{ZA} + n/(j_A \cdot j_T)$

    $= 2 + 1 + 2 + 20 + 10^6/5 \cdot 10^4 = 3 + 22 + 20 = 45!$
  - Cluster-Bildung bei A für Verbesserung von (3) sinnvoll?

DB-Server – Allgemeine Aufgaben

- 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)
  - . . .

DB-Server - Schnittstellen

Schema

ANGESTELLTER

Ausprägungen

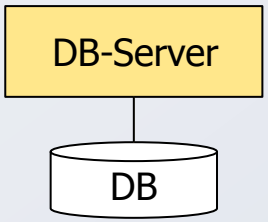
PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	Peinl	Pförtner	2100	63
497	Kinzinger	Kopist	2800	25
498	Meyweg	Kalligraph	4500	56
...				

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

DB-Servers – technische Umsetzung

- Wie werden diese System- und Schnittstelleneigenschaften technisch umgesetzt?
  - Komplexität des DBS
  - Modularität, Portabilität, Erweiterbarkeit, Zuverlässigkeit, . . .
  - Laufzeiteigenschaften (Leistung, 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 > 30 Jahren
  - permanente Evolution des Systems
  - wachsender Informationsbedarf: Objekttypen, Integritätsbedingungen, ...
  - neue Speicherungsstrukturen und Zugriffsverfahren, ...
  - schnelle Änderungen der eingesetzten Technologien: Speicher, ...

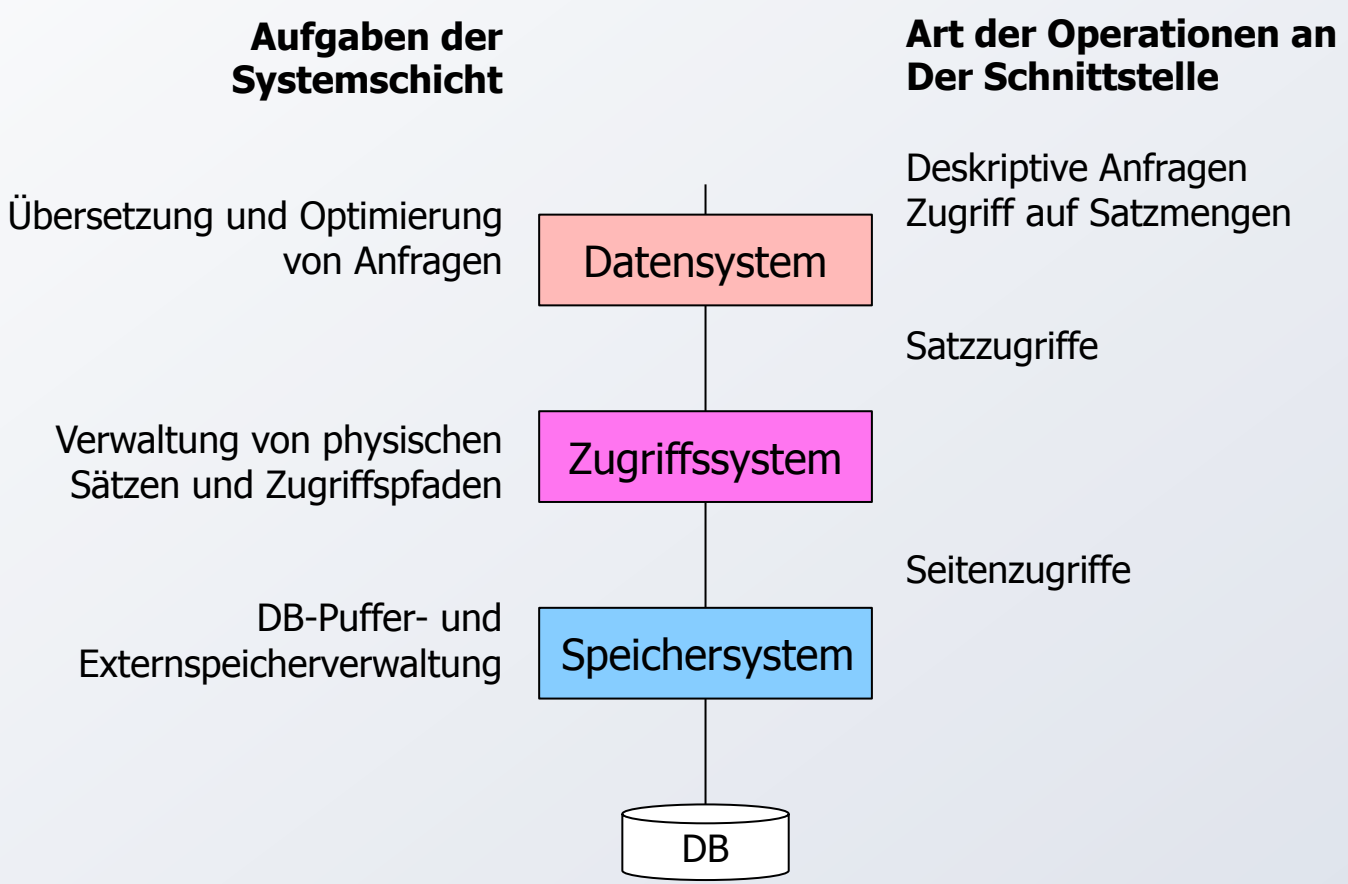
Architektur für DBS

- 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!
  - 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

Architektur für DBS

- 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)

Vereinfachtes Schichtenmodell

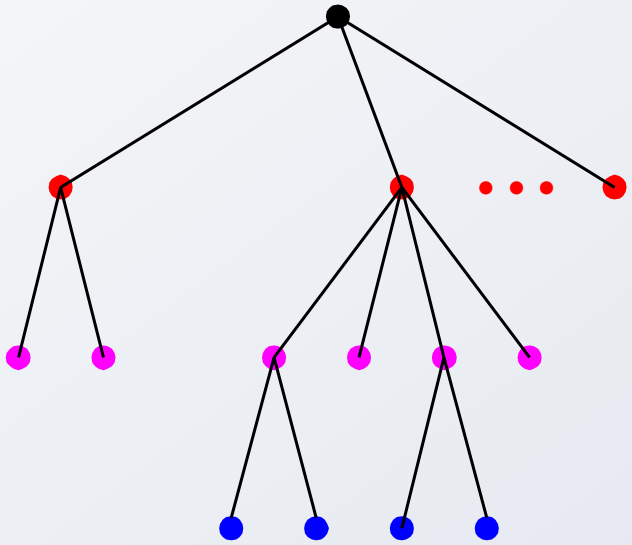


Dynamischer Kontrollfluß einer Operation an das DBS

Datensystem

Zugriffssystem

Speichersystem



DBS-Operationen
(API)

Füge Satz ein
Modifiziere Zugriffspfad

Stelle Seite bereit
Gib Seite frei

Lies / Schreibe Seite

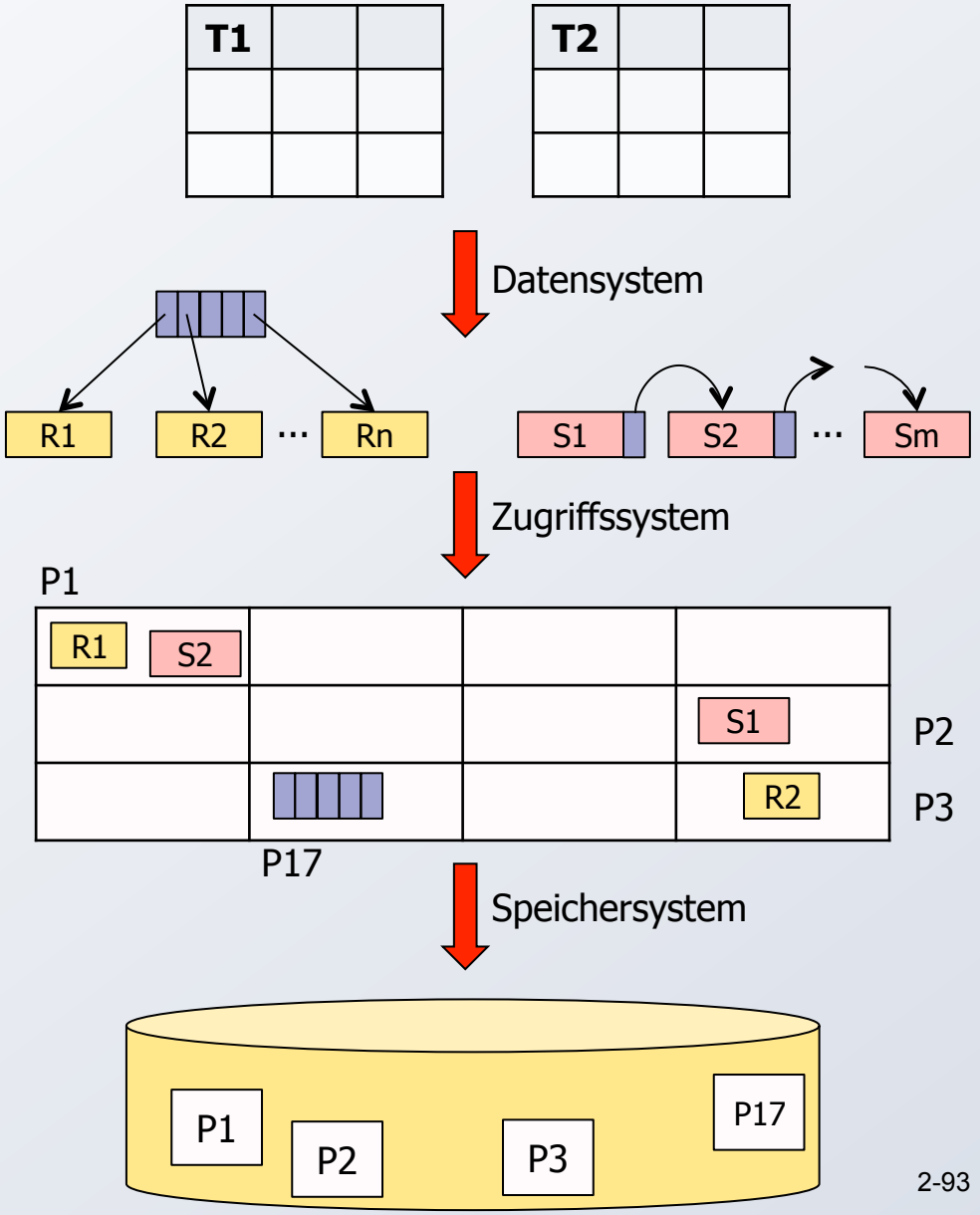
Drei-Schichten-Modell – Abbildungen

Relationen/Sichten
mit mengenorientierten
Operationen

Vielfalt an Satztypen
und Zugriffspfaden mit
satzorientierten
Operationen

DB-Puffer im HSP mit
Seitenzugriff

Externspeicher mit
Dateien verschiedenen
Typs



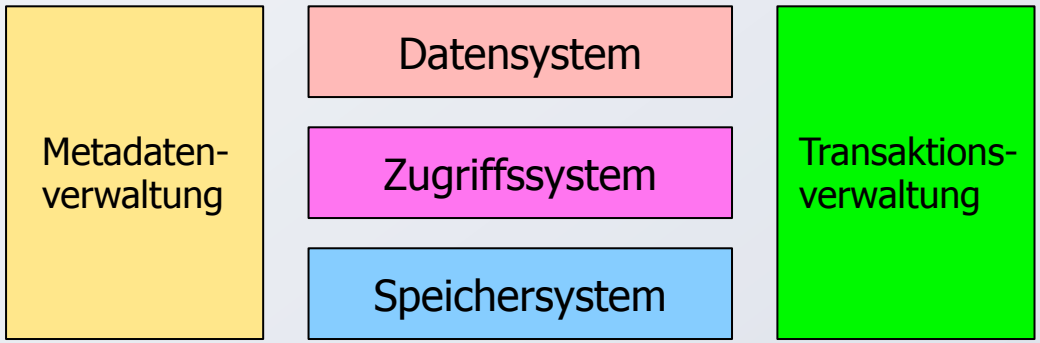
DBS-Architektur – weitere Komponenten

- 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.

DBS-Architektur – weitere Komponenten

- 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 alle 18 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

Zusammenfassung (2)

- 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!