Teil IV

Indexierung von Daten

Überblick

1. Klassifikation der Speichertechniken

Überblick

1. Klassifikation der Speichertechniken

2. Statische Verfahren

Überblick

1. Klassifikation der Speichertechniken

2. Statische Verfahren

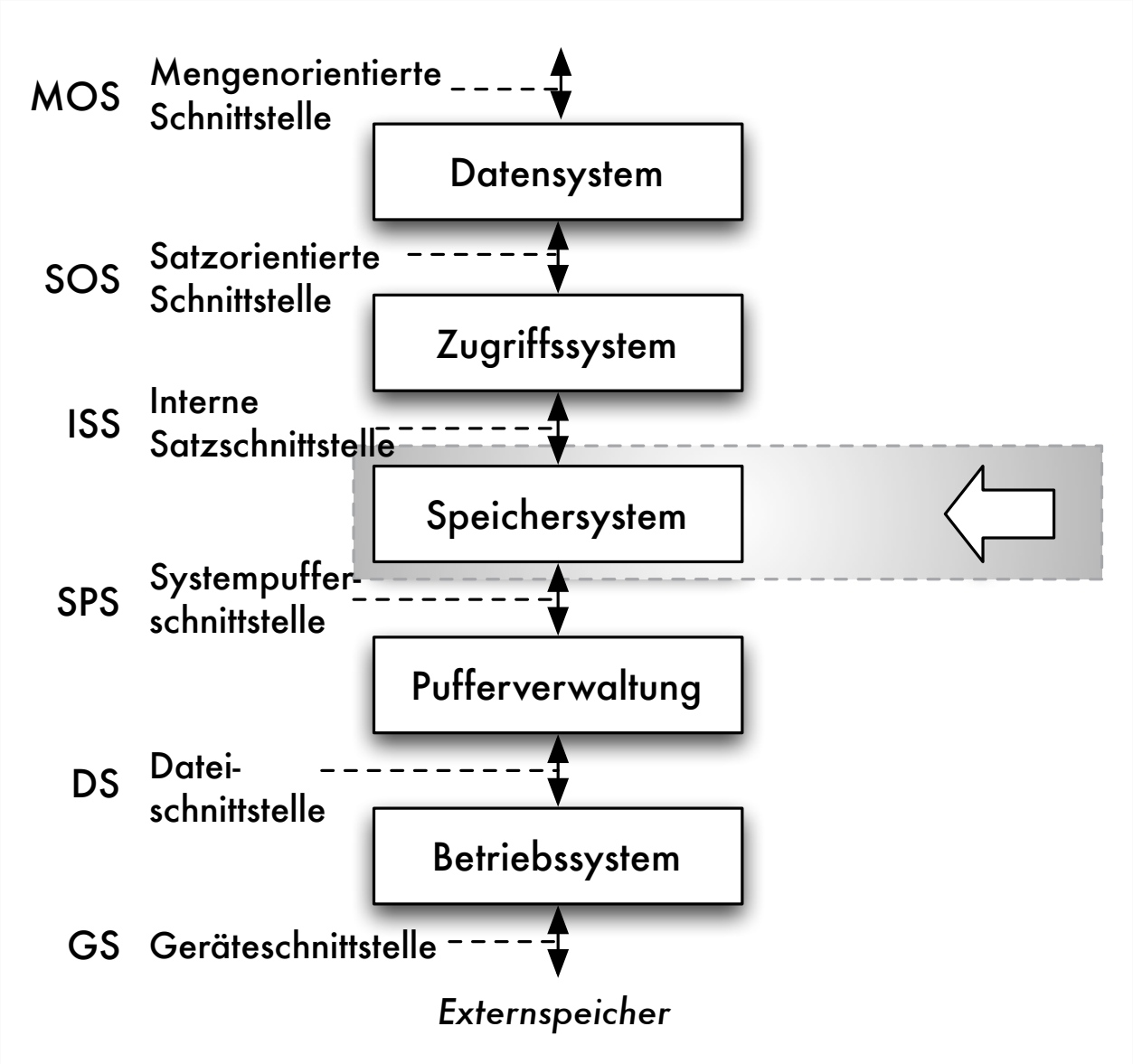
3. Indexsequenzielle Dateiorganisation

Klassifikation der Speichertechniken

Einordnung in 5-Schichten-Architektur

- **Speichersystem** fordert über Systempufferschnittstelle Seiten an
- interpretiert diese als **interne Datensätze**
- interne Realisierung der logischen Datensätze mit Hilfe von Zeigern, speziellen Indexeinträgen und weiteren Hilfsstrukturen
- **Zugriffssystem** abstrahiert von der konkreten Realisierung

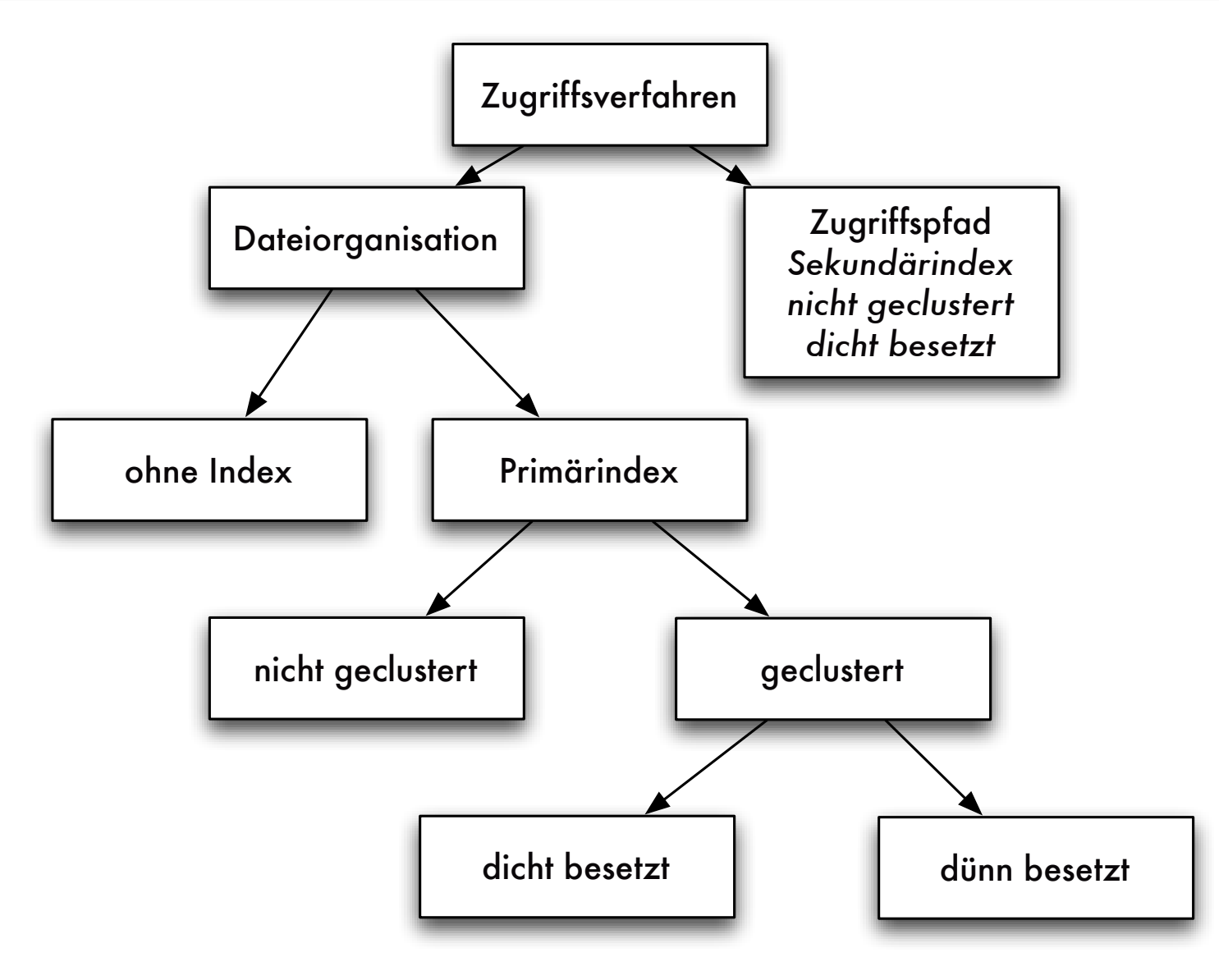
Einordnung /2



Klassifikation der Speichertechniken

- Kriterien für **Zugriffsstrukturen** oder **Zugriffsverfahren**:
  - organisiert interne Relation selbst (**Dateiorganisationsform**) oder zusätzliche Zugriffsmöglichkeit auf bestehende interne Relation (**Zugriffspfad**)
  - Art der Zuordnung von gegebenen Attributwerten zu Datensatz-Adressen: **Schlüsselvergleich** = Zuordnung von Schlüsselwert zu Adresse über Hilfsstruktur; **Schlüsseltransformation** = Berechnung der Adresse aus Schlüsselwert (z.B. über Hashfunktion)
  - Arten von Anfragen, die durch Dateiorganisationsformen und Zugriffspfade effizient unterstützt werden können

Klassifikation der Speichertechniken /2



Klassifikation /3

Dünn- vs. dichtbesetzter Index

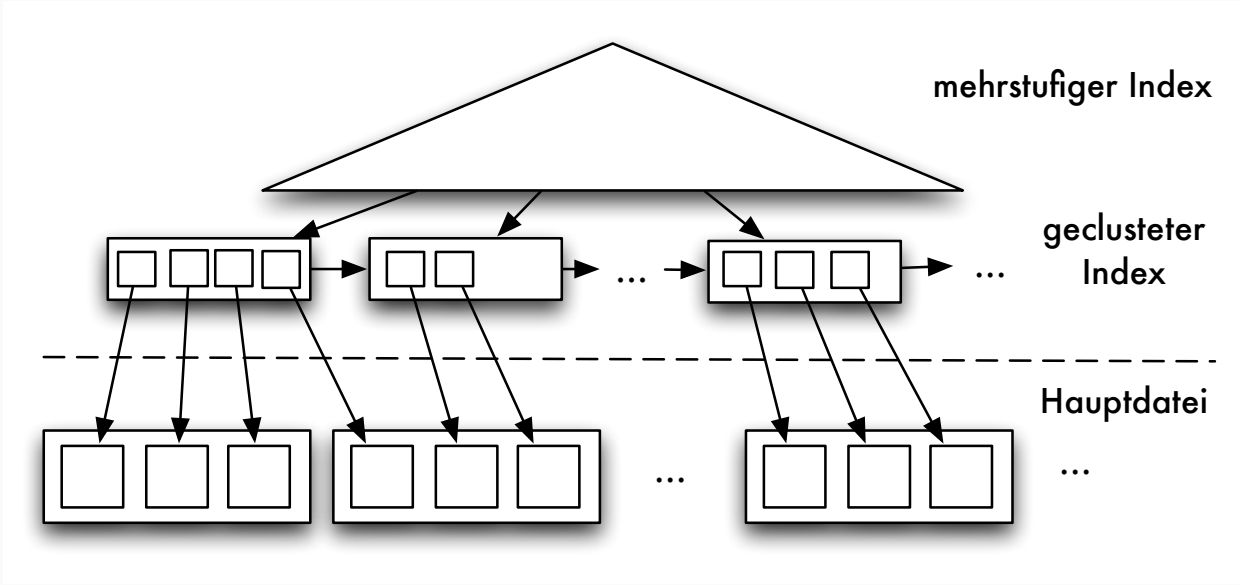
- **dünnbesetzter Index**: nicht für jeden Zugriffsattributwert K ein Eintrag in Indexdatei sondern z.B. nur für *Seitenanführer* einer sortierten Relation
- **dichtbesetzter Index**: für jeden Datensatz der internen Relation ein Eintrag in Indexdatei

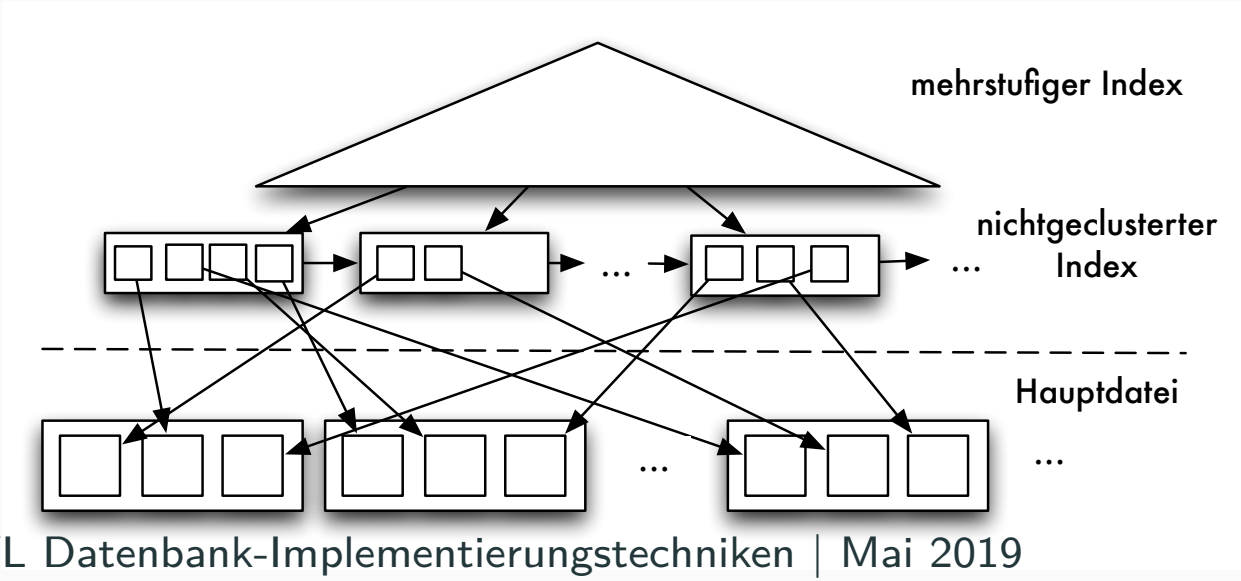
Klassifikation /4

Geclusterter vs. nicht-geclusterter Index

- **geclusterter Index**: in der gleichen Form sortiert wie interne Relation
- **nicht-geclusterter Index**: anders organisiert als interne Relation
- Primärindex oft dünnbesetzt und geclustert
- jeder dünnbesetzte Index ist auch geclusterter Index, aber nicht umgekehrt
- Sekundärindex kann nur dichtbesetzter, nicht-geclusterter Index sein (auch: invertierte Datei)

Geclusterter vs. nicht-geclusterter Index

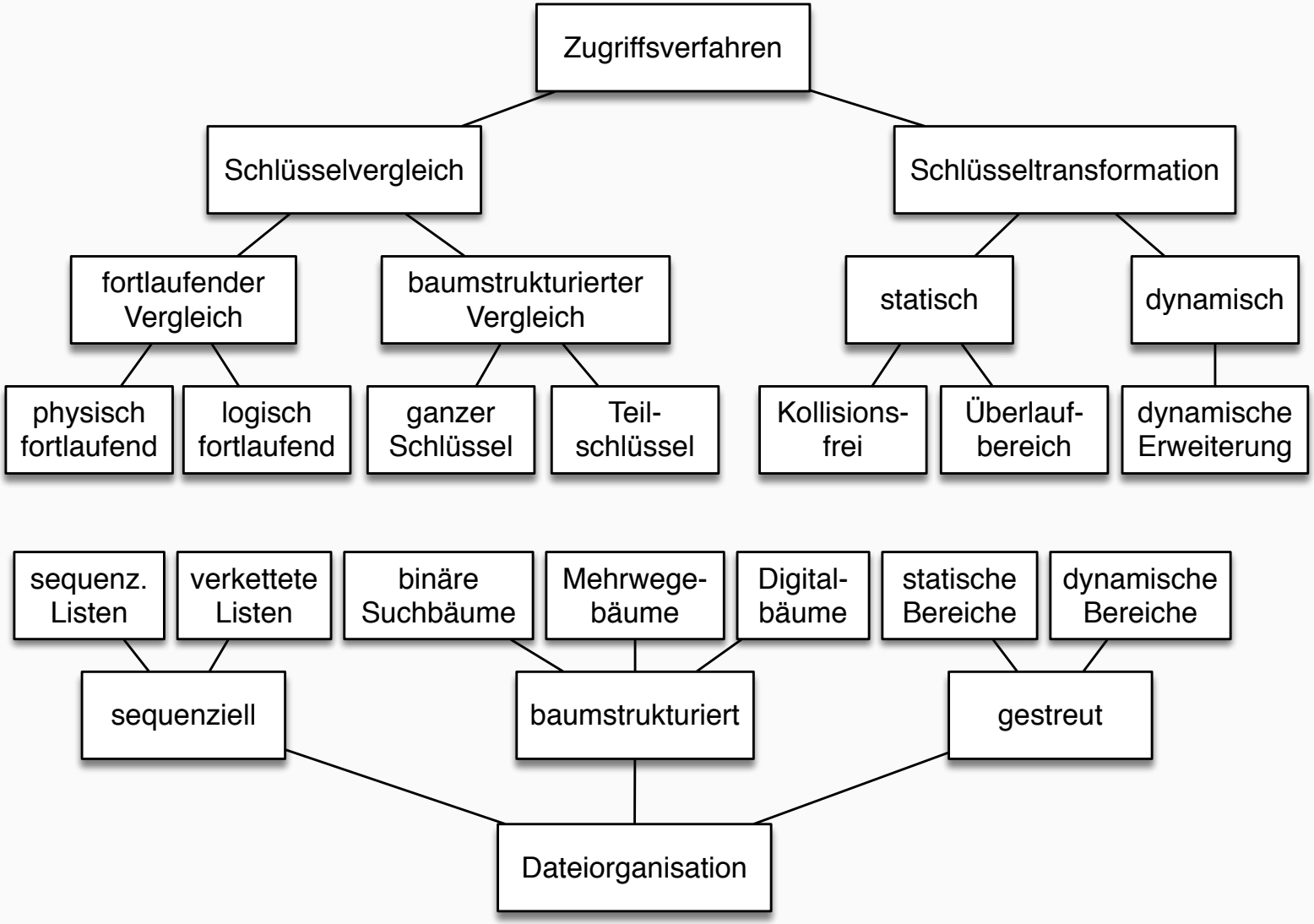




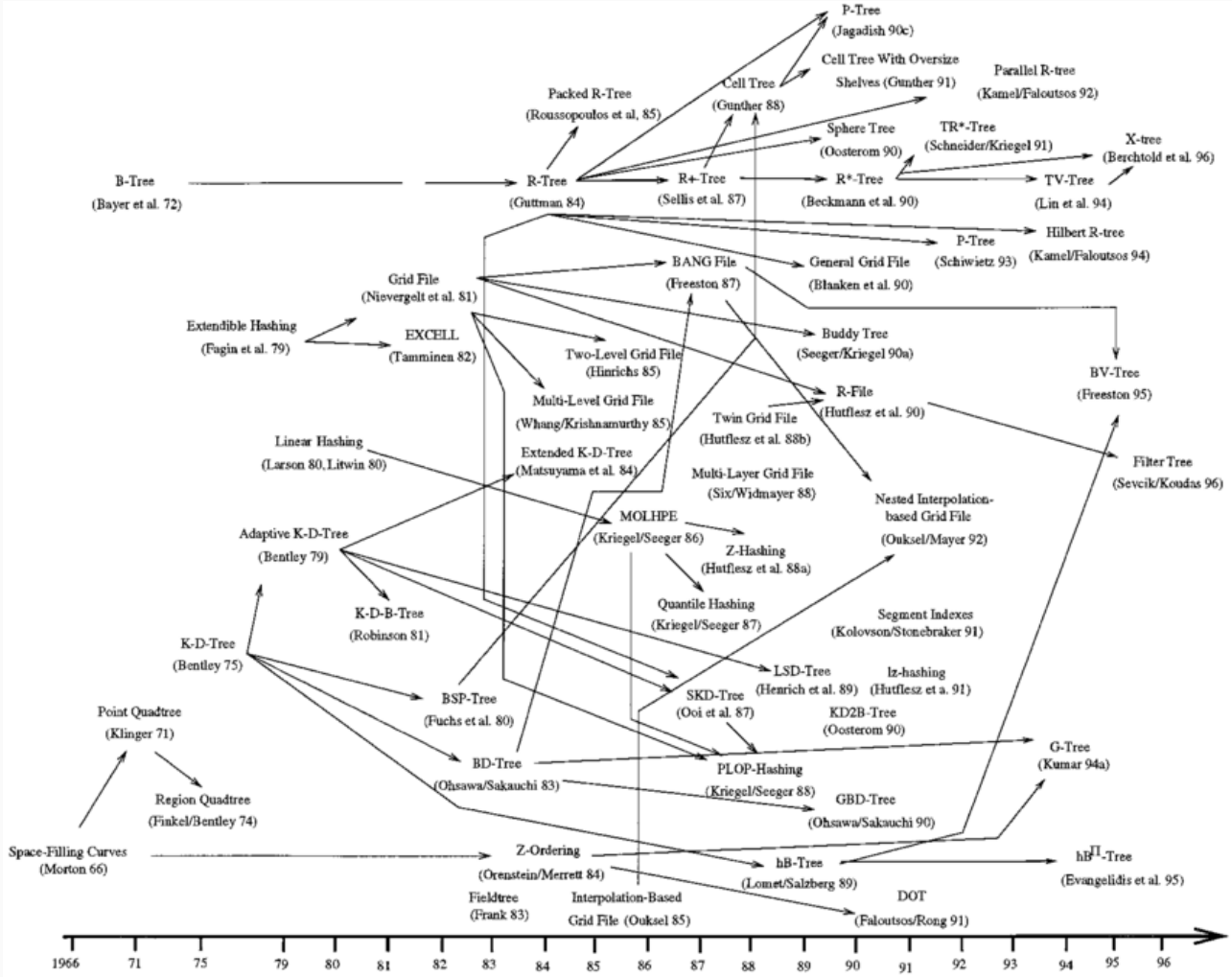
Statische vs. dynamische Struktur

- **statische Zugriffsstruktur**: optimal nur bei bestimmter (fester) Anzahl von verwaltenden Datensätzen
- **dynamische Zugriffsstruktur**: unabhängig von der Anzahl der Datensätze optimal
  - dynamische Adresstransformationsverfahren verändern dynamisch Bildbereich der Transformation
  - dynamische Indexverfahren verändern dynamisch Anzahl der Indexstufen $\Rightarrow$ in DBS üblich

Klassifikation



Genealogie von Indexstrukturen [Gaede/Günther, 1999]

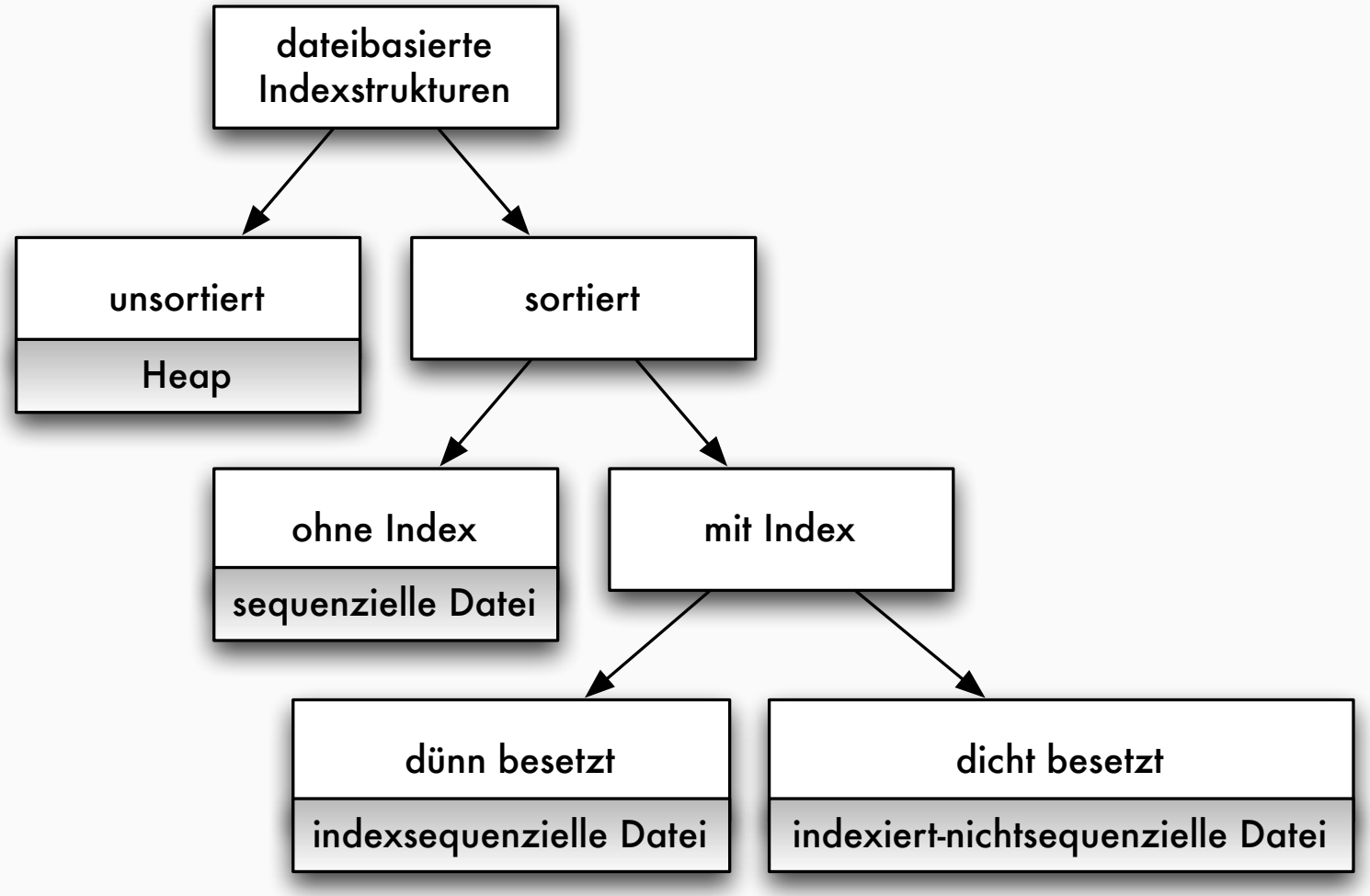


Statische Verfahren

Statische Verfahren

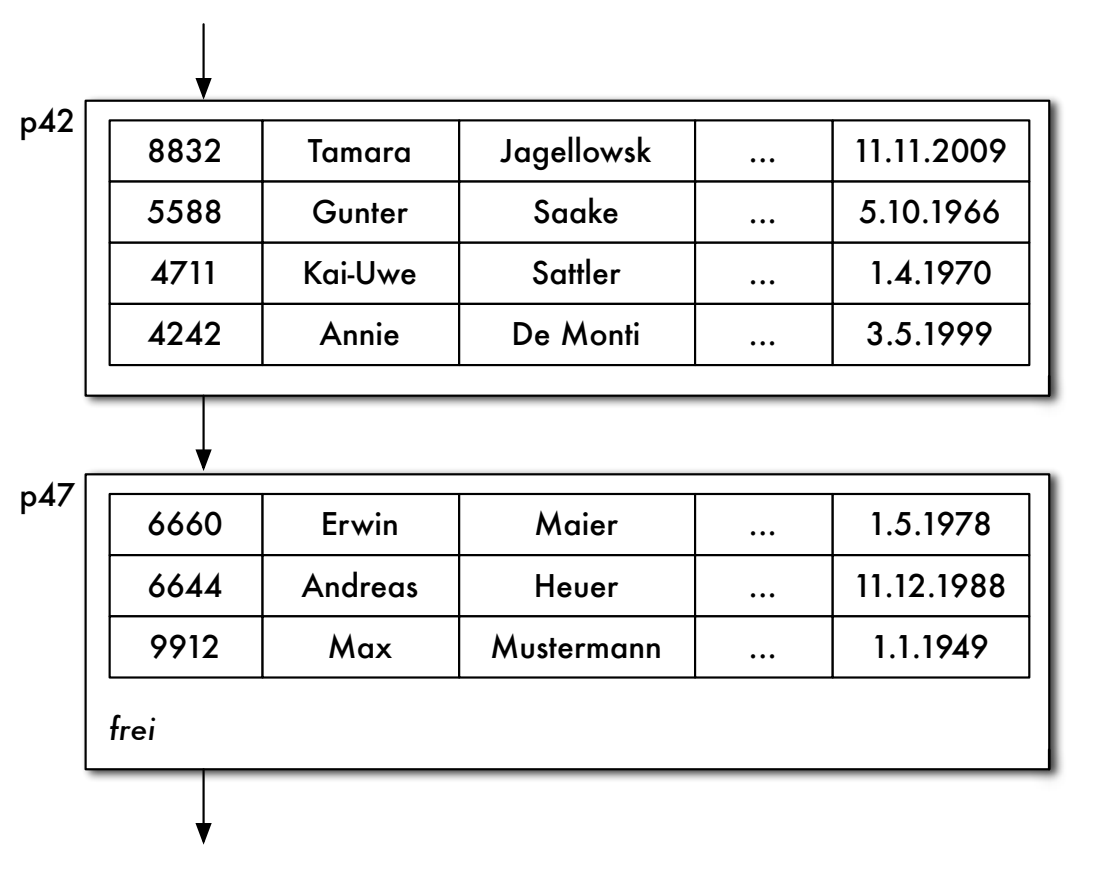
- Heap, indexsequenziell, indiziert-nichtsequenziell
- oft grundlegende Speichertechnik in RDBS
- direkte Organisationsformen: keine Hilfsstruktur, keine Adressberechnung (Heap, sequenziell)
- statische Indexverfahren für Primärindex und Sekundärindex

Statische Verfahren: Überblick



Heap-Organisation

- völlig unsortiert speichern
- physische Reihenfolge der Datensätze ist zeitliche Reihenfolge der Aufnahme von Datensätzen

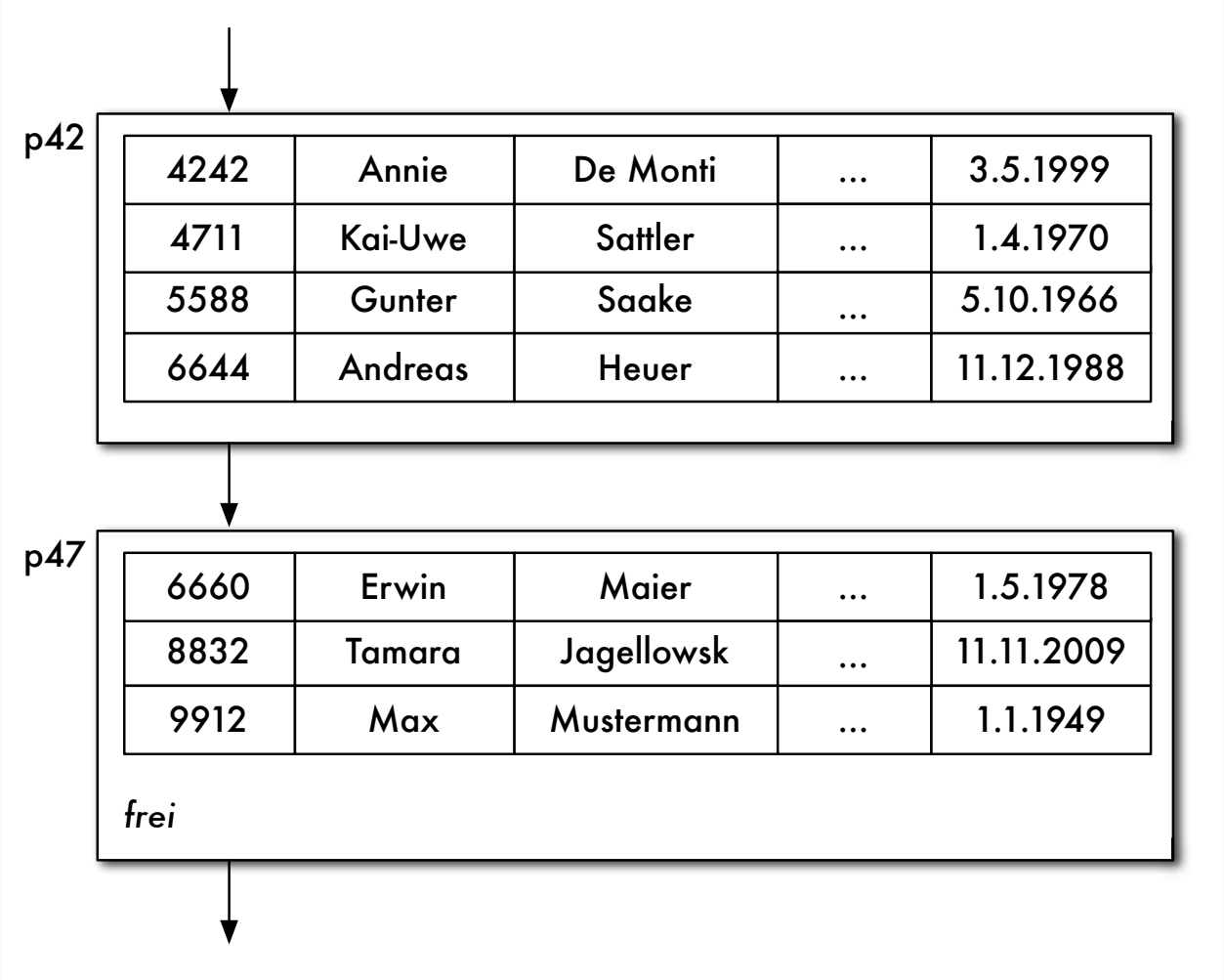


Heap: Operationen

- `insert`: Zugriff auf letzte Seite der Datei. Genügend freier Platz $\Rightarrow$ Satz anhängen. Sonst nächste freie Seite holen
- `delete`: `lookup`, dann Löschbit auf 0 gesetzt
- `lookup`: sequenzielles Durchsuchen der Gesamtdatei, maximaler Aufwand (Heap-Datei meist zusammen mit Sekundärindex eingesetzt; oder für sehr kleine Relationen)
- Komplexitäten:
  - Neuaufnahme von Daten $O(1)$
  - Suchen $O(n)$

Sequenzielle Speicherung

- sortiertes Speichern der Datensätze



Sequenzielle Datei: Operationen

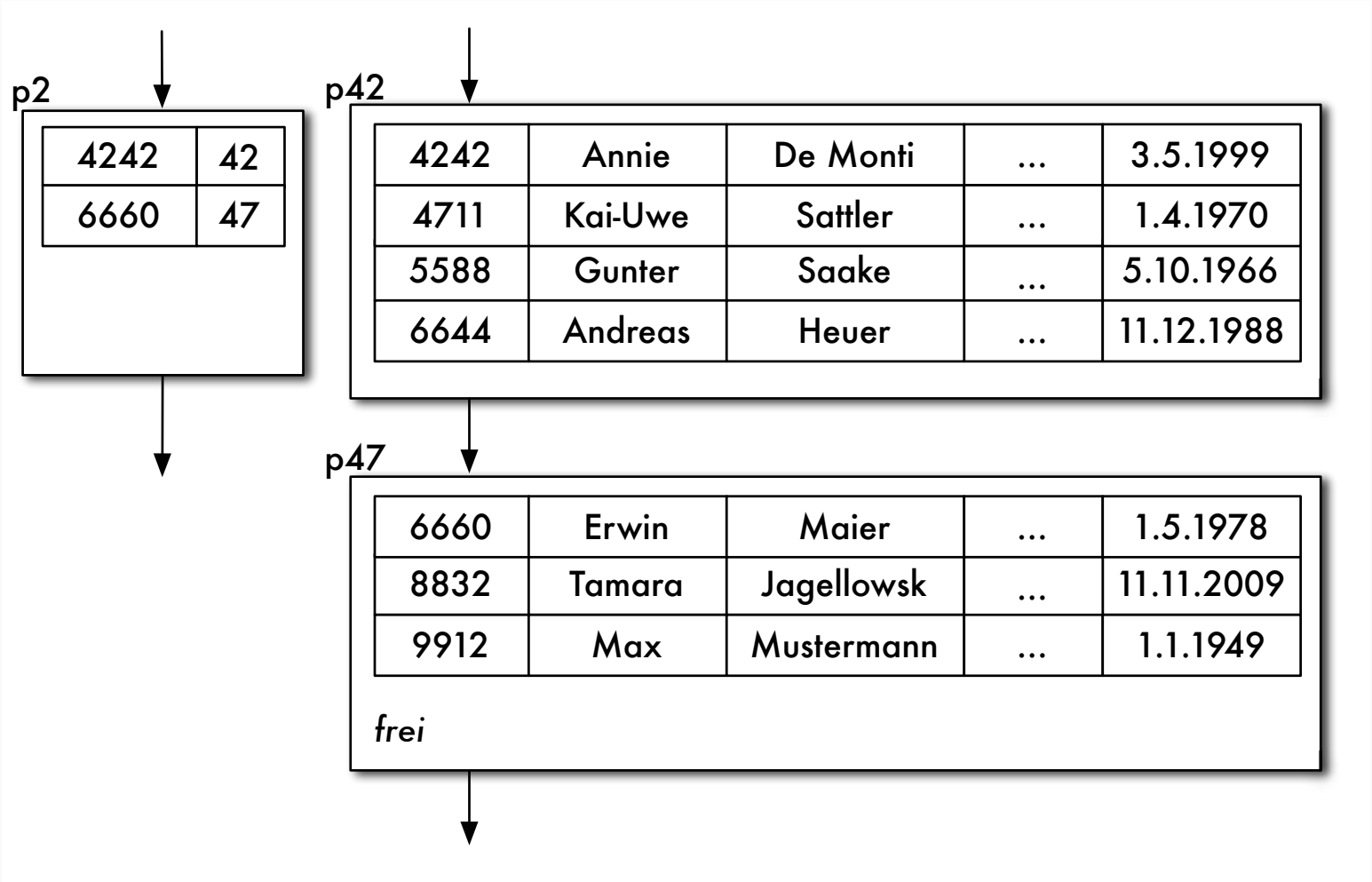
- `insert`: Seite suchen, Datensatz einsortieren $\Rightarrow$ beim Anlegen oder sequenziellen Füllen einer Datei jede Seite nur bis zu gewissem Grad (etwa 66%) füllen
- `delete`: Aufwand bleibt
- Folgende Dateiorganisationsformen:
  - schnelleres `lookup`
  - mehr Platzbedarf (durch Hilfsstrukturen wie Indexdateien)
  - mehr Zeitbedarf bei `insert` und `delete`
- klassische Indexform: indexsequenzielle Dateiorganisation

Indexsequenzielle Dateiorganisation

Indexsequenzielle Dateiorganisation

- Kombination von sequenzieller Hauptdatei und Indexdatei: **indexsequenzielle Dateiorganisationsform**
- Indexdatei kann geclusterter, dünnbesetzter Index sein
- mindestens zweistufiger Baum
  - Blattebene ist **Hauptdatei** (Datensätze)
  - jede andere Stufe ist **Indexdatei**

Indexsequenzielle Dateiorganisation /2



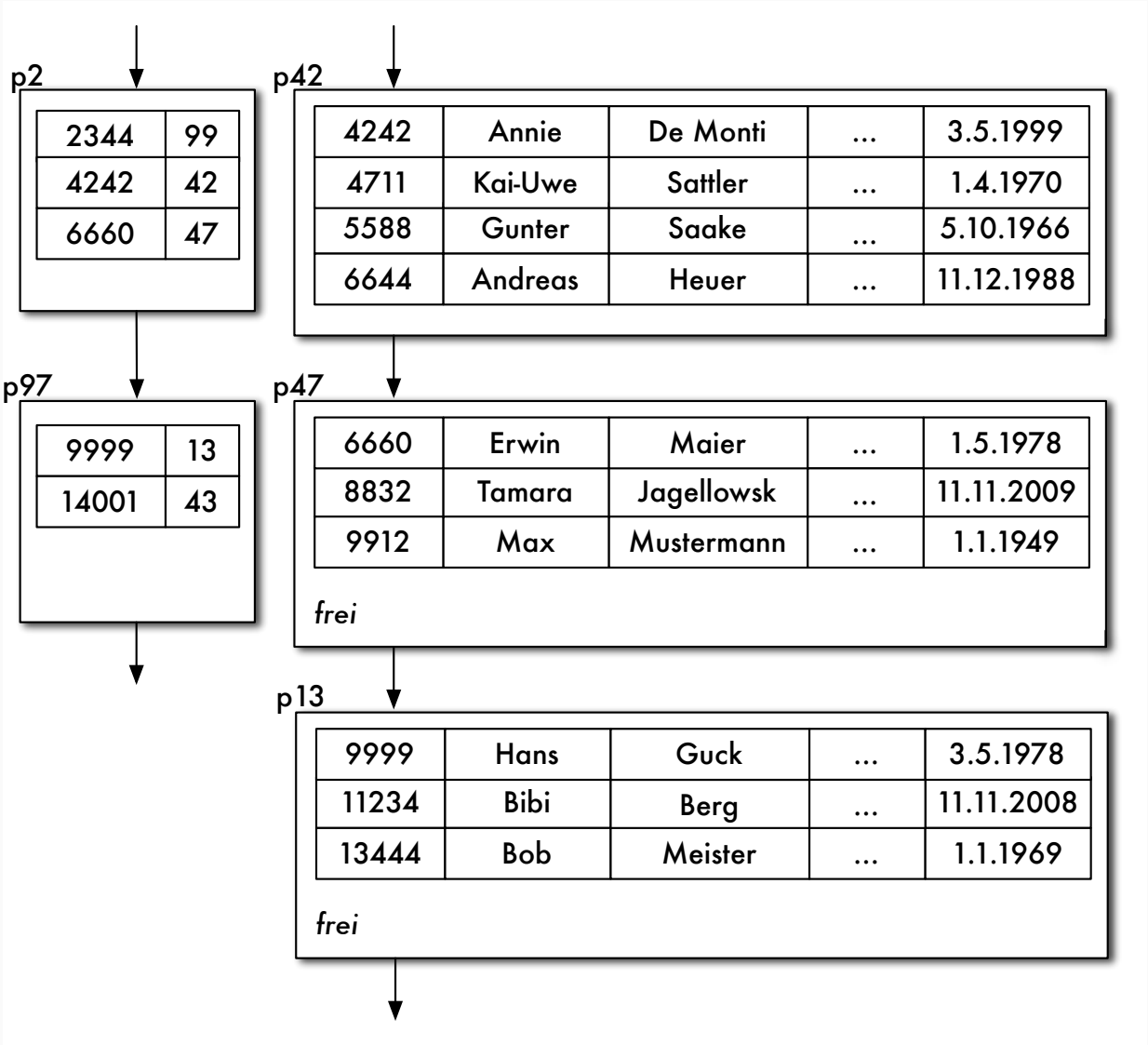
Aufbau der Indexdatei

- Datensätze in Indexdatei:

  (Primärschlüsselwert, Seitennummer)

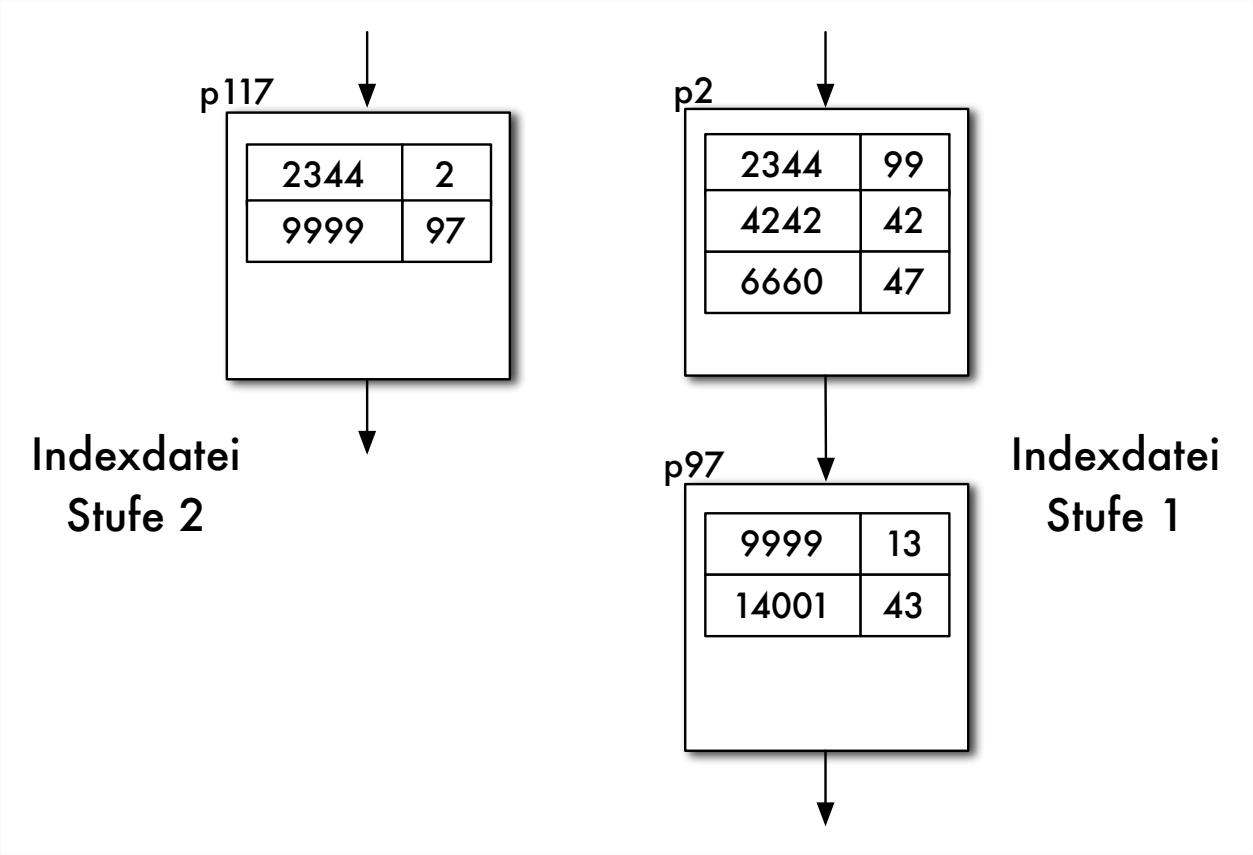
  zu jeder Seite der Hauptdatei genau ein Index-Datensatz in Indexdatei
- Problem: „Wurzel“ des Baumes bei einem einstufigen Index nicht nur eine Seite

Aufbau der Indexdatei /2



Mehrstufiger Index

- Optional: Indexdatei wieder indexsequenziell verwalten
- Idealerweise: Index höchster Stufe nur noch eine Seite



`lookup` bei indexsequenziellen Dateien

- `lookup`-Operation sucht Datensatz zum Zugriffsattributwert w
- Indexdatei sequenziell durchlaufen, dabei (v_1, s) im Index gesucht mit $v_1 \leq w$:
  - (v_1, s) ist letzter Satz der Indexdatei, dann kann Datensatz zu w höchstens auf dieser Seite gespeichert sein (wenn er existiert)
  - nächster Satz (v_2, s') im Index hat $v_2 > w$, also muß Datensatz zu w, wenn vorhanden, auf Seite s gespeichert sein
- (v_1, s) **überdeckt** Zugriffsattributwert w

`insert` bei indexsequenziellen Dateien

- **`insert`**: zunächst mit **`lookup`** Seite finden
- Falls Platz, Satz sortiert in gefundener Seite speichern; Index anpassen, falls neuer Satz der erste Satz in der Seite
- Falls kein Platz, neue Seite von Freispeicherverwaltung holen; Sätze der „zu vollen“ Seite gleichmäßig auf alte und neue Seite verteilen; für neue Seite Indexeintrag anlegen
- Alternativ neuen Datensatz auf Überlaufseite zur gefundenen Seite

`delete` bei indexsequenziellen Dateien

- **`delete`**: zunächst mit **`lookup`** Seite finden
- Satz auf Seite löschen (Löschbit auf 0)
- erster Satz auf Seite: Index anpassen
- Falls Seite nach Löschen leer: Index anpassen, Seite an Freispeicherverwaltung zurück

Probleme indexsequenzieller Dateien

- **stark wachsende Dateien**: Zahl der linear verketteten Indexseiten wächst; automatische Anpassung der Stufenanzahl nicht vorgesehen
- **stark schrumpfende Dateien**: nur zögernde Verringerung der Index- und Hauptdatei-Seiten
- **unausgeglichene Seiten** in der Hauptdatei (unnötig hoher Speicherplatzbedarf, zu lange Zugriffszeit)

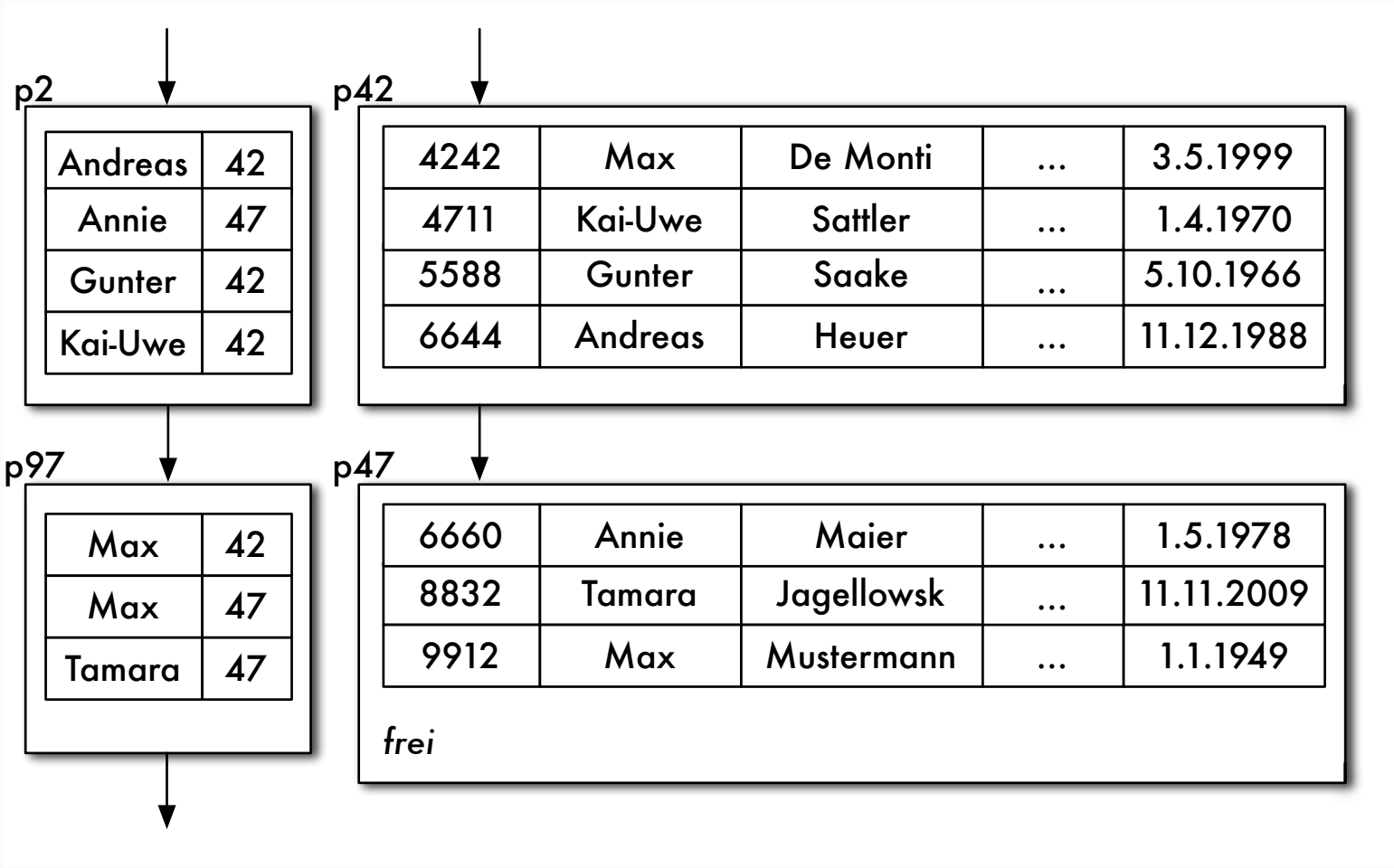
Indiziert-nichtsequenzieller Zugriffspfad

- zur Unterstützung von Sekundärschlüsseln
- mehrere Zugriffpfade dieser Form pro Datei möglich
- einstufig oder mehrstufig: höhere Indexstufen wieder indexsequenziell organisiert

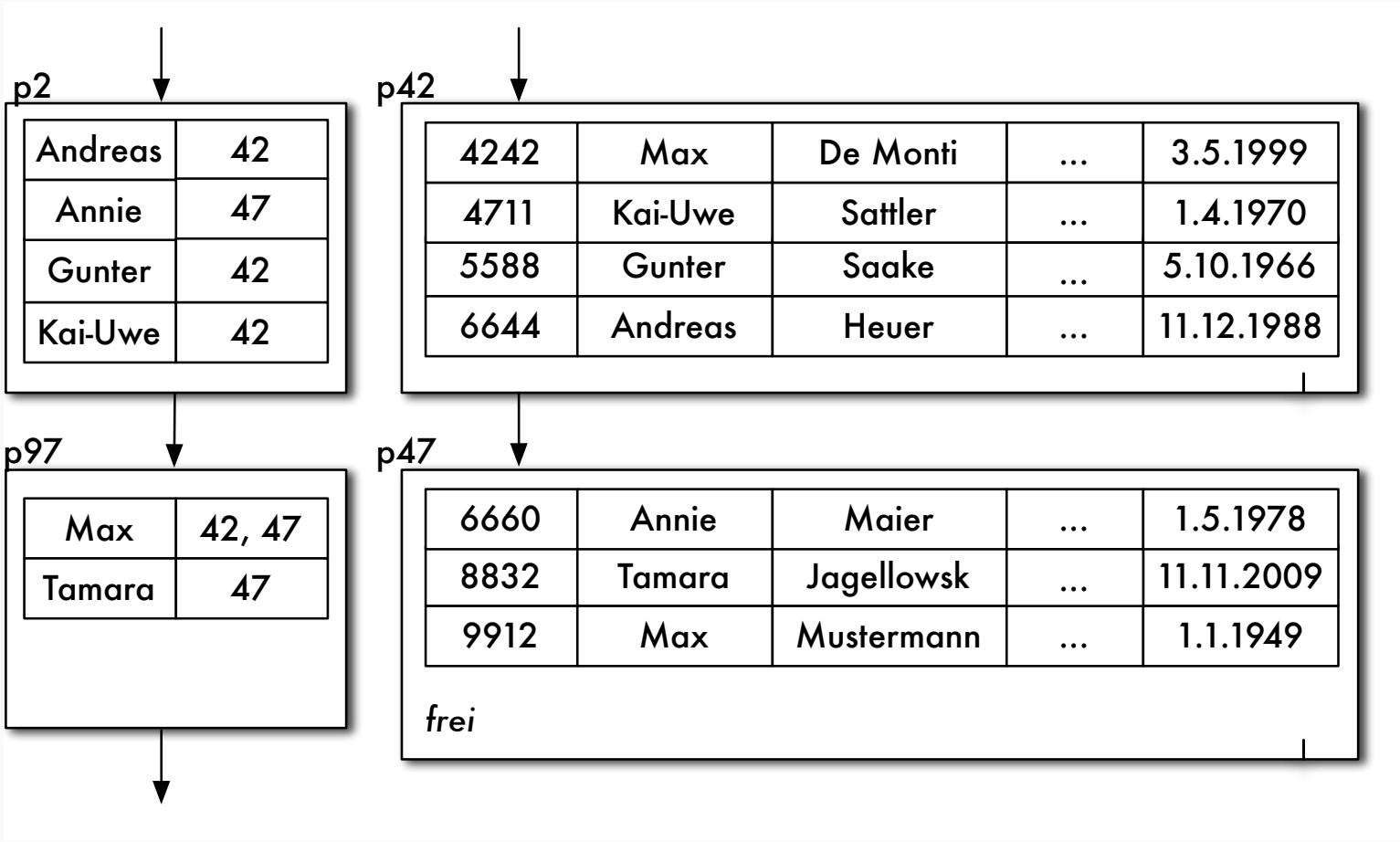
Aufbau der Indexdatei

- Sekundärindex, dichtbesetzter und nicht-geclusteter Index
- zu jedem Satz der Hauptdatei Satz (w, s) in der Indexdatei
- w Sekundärschlüsselwert, s zugeordnete Seite
  - entweder für ein w mehrere Sätze in die Indexdatei aufnehmen
  - oder für ein w Liste von Adresse in der Hauptdatei angeben

Aufbau der Indexdatei /2



Aufbau der Indexdatei /3



Operationen

- `lookup`: w kann mehrfach auftreten, Überdeckungstechnik nicht benötigt
- `insert`: Anpassen der Indexdateien
- `delete`: Indexeintrag entfernen

Probleme statischer Verfahren

- unzureichende Anpassung an wachsende/schrumpfende Datenmengen
- schlechte Ausnutzung von Speicher nach Seitensplits
- Bevorzugung bestimmter Attribute (Schlüssel)
- **daher in den folgenden Kapiteln:**
  - bessere Datenstrukturen zur Schlüsselsuche als zusätzlicher Zugriffspfad = Approximation einer Funktion Schlüssel $\rightarrow$ Speicheradresse, z.B. über **Baumverfahren**
  - Erweiterung von Hashverfahren um Anpassung des Bildbereichs = **dynamische Hashverfahren**
  - Behandlung von zusammengesetzten Schlüsseln = multidimensionale Zugriffsverfahren, z.B. **multidimensionale Bäume** oder **raumfüllende Kurven**

Zusammenfassung

- Dateiorganisation vs. Zugriffsverfahren
- Statische Verfahren
- Literatur: „Datenbanken: Implementierungstechniken“, Kapitel 4