Datenbankindexe

Indexe / Indexstrukturen / Zugriffspfade

- Ein Datenbankindex ist eine Datenstruktur mit deren Hilfe die Abfragen optimiert werden können
- Der Index selbst stellt einen Zeiger dar, der entweder auf einen weiteren Index oder auf einen Datensatz zeigt
- Dateneintrag eines Index → Indexeintrag $k^*=(k,v)$
- Der Index beschleunigt die Suche und das Sortieren nach bestimmten Feldern (die Felder, auf denen der Index erstellt wurde)
- Suchschlüssel = ein Attribut oder Attributkombination einer Relation, die als Suchkriterium dient
- Suchschlüssel ≠ Kandidatschlüssel/Primärschlüssel (Suchschlüssel kommt in der Bedingung einer Abfrage vor, hat nichts mit dem Kandidatschlüssel oder Primärschlüssel einer Relation zu tun)

Eigenschaften der Indexe

- Propagierung der Änderungen
  - Wenn Tupeln eingefügt oder gelöscht werden, müssen alle Indexstrukturen aktualisiert werden
  - Wenn der Suchschlüssel geändert wird, muss die Indexstruktur aktualisiert werden
- Indexgröße – da der Index in dem Primärspeicher aufgeladen werden muss, um eine Suche durchzuführen, muss der Index nicht zu groß sein
- Falls Index zu groß
  - Partielle Indexstruktur
  - Index der auf einen weiteren Index zeigt

Eigenschaften der Indexe

- Fragen bei dem Erstellen des Index:
  - Was speichern wir in jedem Indexeintrag (wie speichern wie die Datensätze) ?
    → ***Index Inhalt***
  - Wie sind die Indexeinträge orgnisiert?
    → ***Indexierungstechnik***

Indexinhalt und Indexierungstechniken

- Drei Alternativen für die Struktur des Indexeintrags $k^*=(k,v)$:
  1. Ein Paar (k, Datensatz) mit dem Suchschlüsselwert k
  2. Ein Paar (k, RID), wobei RID der Datensatzindetifikator eines Datensatzes mit dem Suchschlüsselwert k ist
  3. Ein Paar (k, RID-Liste), wobei RID-Liste eine Liste von Datensatzindentifikatoren von Datensätze mit dem Suchschlüsselwert k ist
- Die Alternative für die Indexeinträge wird „ungefähr“ unabhängig von der Indexierungstechnik ausgewählt; die Indexierungstechnik sorgt nur dafür, Einträge mit einem bestimmten Suchschlüsselwert zu finden
- Beispiele von Indexierungstechniken: B^+ Bäume, hash-basierte Strukturen
- Meistens enthalten Indexe zusätzliche Hilfsinformationen, welche zu der Suche eines Eintrags beitragen

Indexeinträge - Alternative (1)

- Es ist nicht notwendig die Datensätze zusätzlich zu dem Inhalt des Index getrennt abzuspeichern
- Hier haben wir eine spezielle Primärorganisation, die anstelle einer sortierten Datei oder einer Haufendatei (heap file) benutzt werden kann
- Diese Alternative braucht am meisten Speicherplatz für den Index
- Für eine Menge von Datensätze kann es einen einzigen Index geben, der Alternative (1) für die Indexeinträge benutzt (ansonsten würden wir die Datensätze mehrmals speichern → Redundanzen und Inkonsistenzen)

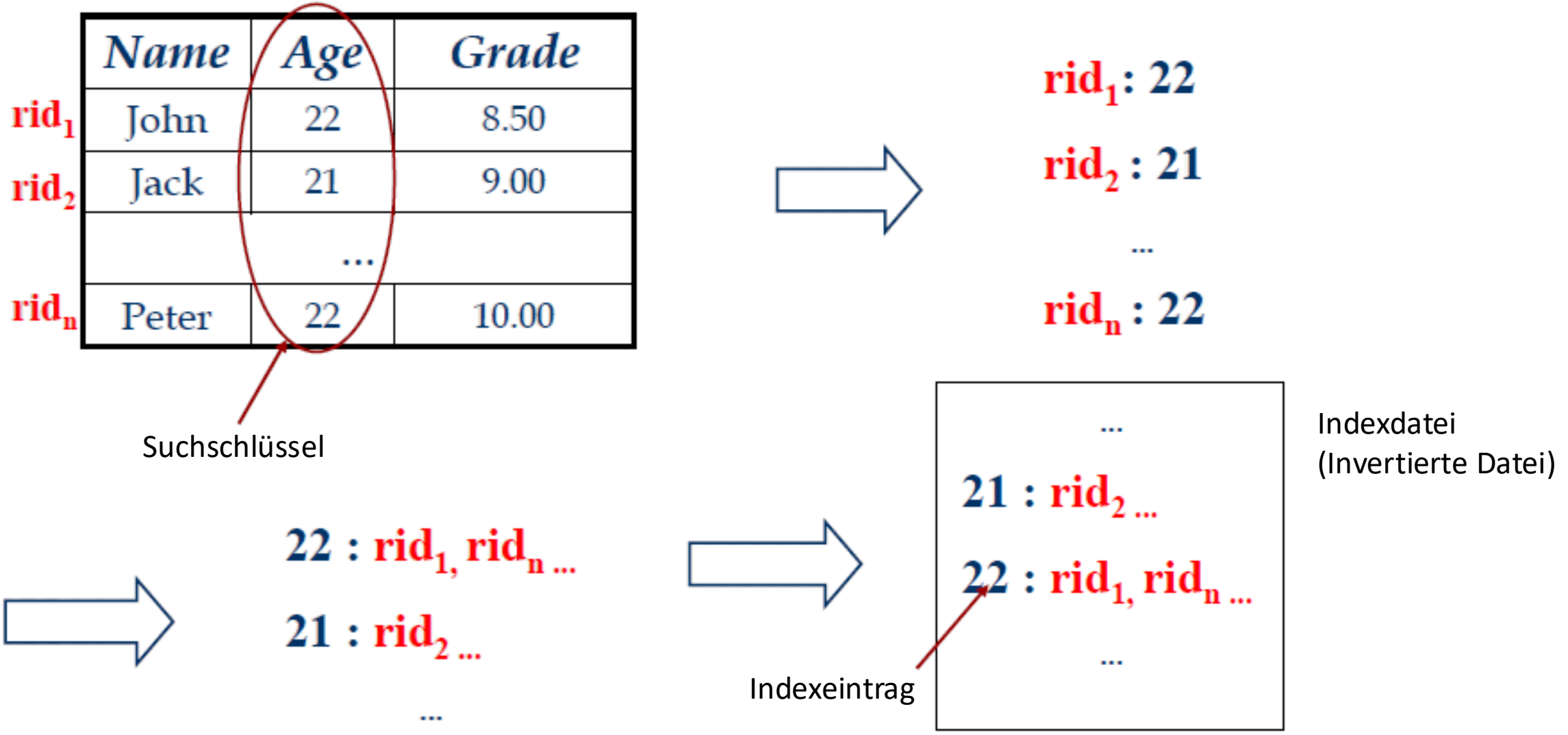
Indexeinträge - Alternative (2), (3)

- Die Indexeinträge „zeigen“ auf die eigentlichen Datensätze (die Datensätze werden nicht in den Indexeinträgen gespeichert)
- Diese Alternativen sind unabhängig von der Struktur der Datei, die die Datensätze enthält
- Es handelt sich also um Sekundärorganisationen
- Die Indexeinträge sind normalerweise viel kleiner als in Alternative (1) (insbesondere wenn die Suchschlüssel viel kleiner als die Datensätze sind)
- Der Teil des Index, der für die eigentliche Suche benutzt wird ist auch viel kleiner als in Alternative (1)

Indexeinträge - Alternative (2), (3)

- Alternative (3) ist speicherplatzeffizienter als Alternative (2), aber Indexeinträge sind von variabler Länge
- Die Länge eines Indexeintrag hängt von der Anzahl der Datensätze mit gleichem Schlüsselwert ab
- Wenn mehrere Indexe für eine Datei gebraucht werden, dann sollen wir ein einziger Index mit Alternative (1) benutzen und die anderen mit Alternative (2) oder (3)

Erstellen eines Index



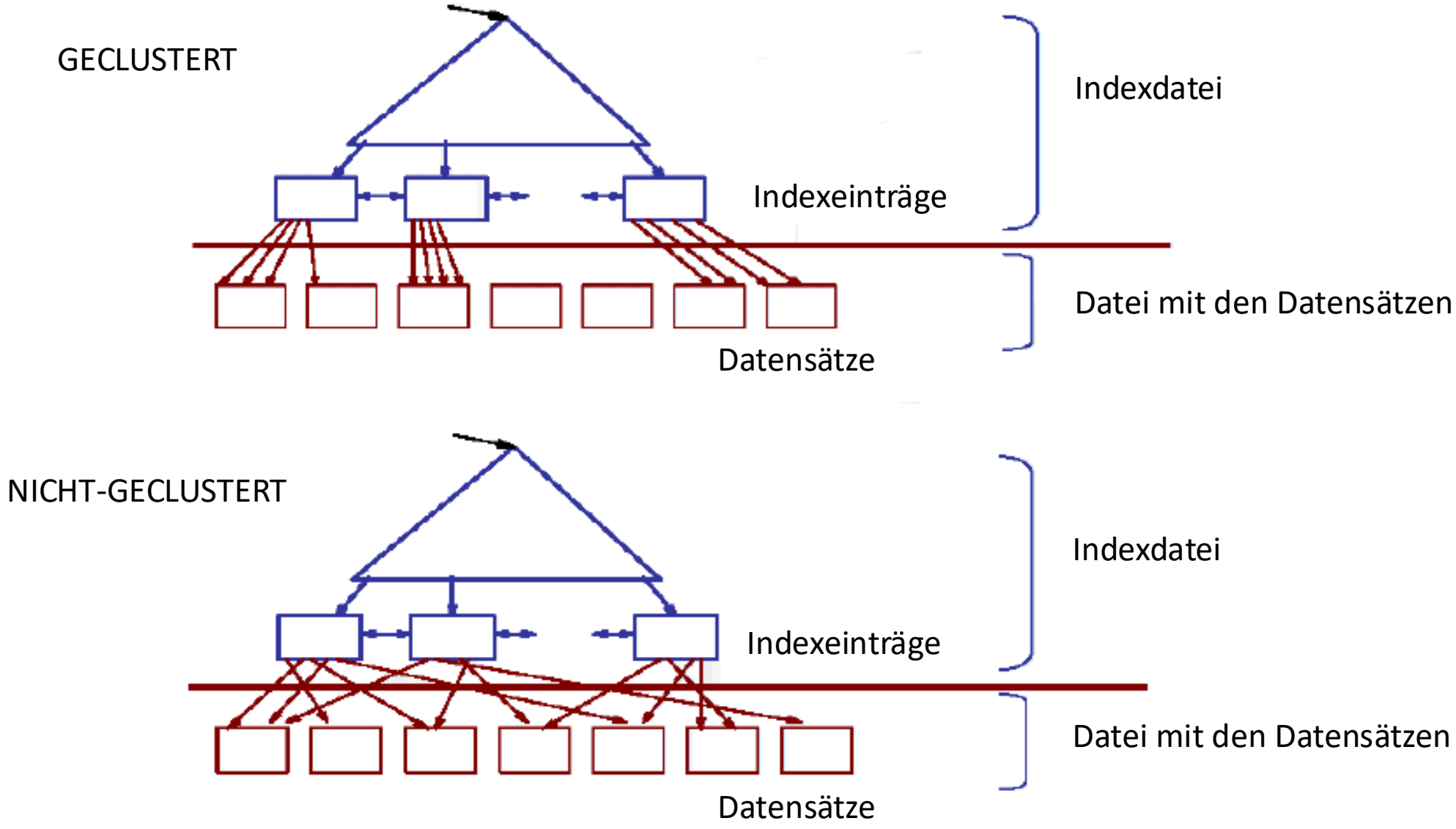
Klassifikation für Indexstrukturen

- Wir betrachten folgende verschiedene Klassen von Indexstrukturen, die größten Teils kombinierbar sind:
  - Gruppierte und nicht-gruppierte Indexe (clustered and unclustered)
  - Dichte und dünne Indexe
  - Primär- und Sekundärindexe
  - Indexe mit einfachen und zusammengesetzten Suchschlüssel
  - Ein- und mehrstufige Indexe

Geclusterte und nicht geclusterte Indexe

- Wenn eine Datei so organisiert ist, dass die Ordnung der Datensätze gleich oder beinahe gleich der Ordnung der Einträge in einem Index ist, so spricht man von einem **geclusterten/gruppierten Index (clustered index)**
- Ein Index, der gemäß Alternative (1) aufgebaut ist, ist per Definition geclustert
- Wenn ein Index, der Alternative (2) oder (3) verwendet, geclustert sein soll, ist dies nur sinnvoll, wenn die Datensätze nach dem Suchschlüssel sortiert sind
- Eine Datei kann höchstens bzgl. eines Suchschlüssels geclustert sein
- Die **Kosten** einer Indexbenutzung für eine **Bereichsanfrage** (Datensätze in einem Bereich finden) hängen viel davon ab, ob der Index geclustert ist oder nicht

Geclusterte und nicht geclusterte Indexe



Geclusterte und nicht geclusterte Indexe

- In der Praxis werden die Datensätze selten in sortierter Reihenfolge gehalten → es ist **sehr aufwendig** die Daten sortiert zu behalten
- Daher, um ein Clustered Index aufzubauen:
  - Erst Datensätze in einer Haufendatei (heap file) sortierten
  - Auf jede Seite bleibt einen gewissen freien Speicherplatzbereich für zukünftige Einfügungen
  - Wenn der freie Platz aufgebraucht wird, dann werden weitere Einfügungen auf **Überlaufseiten** ausgelagert (mit Links zu diesen Seiten)
  - Nach einiger Zeit wird die Datei dann reorganisiert (neu sortiert) um eine gute Effizienz zu behalten

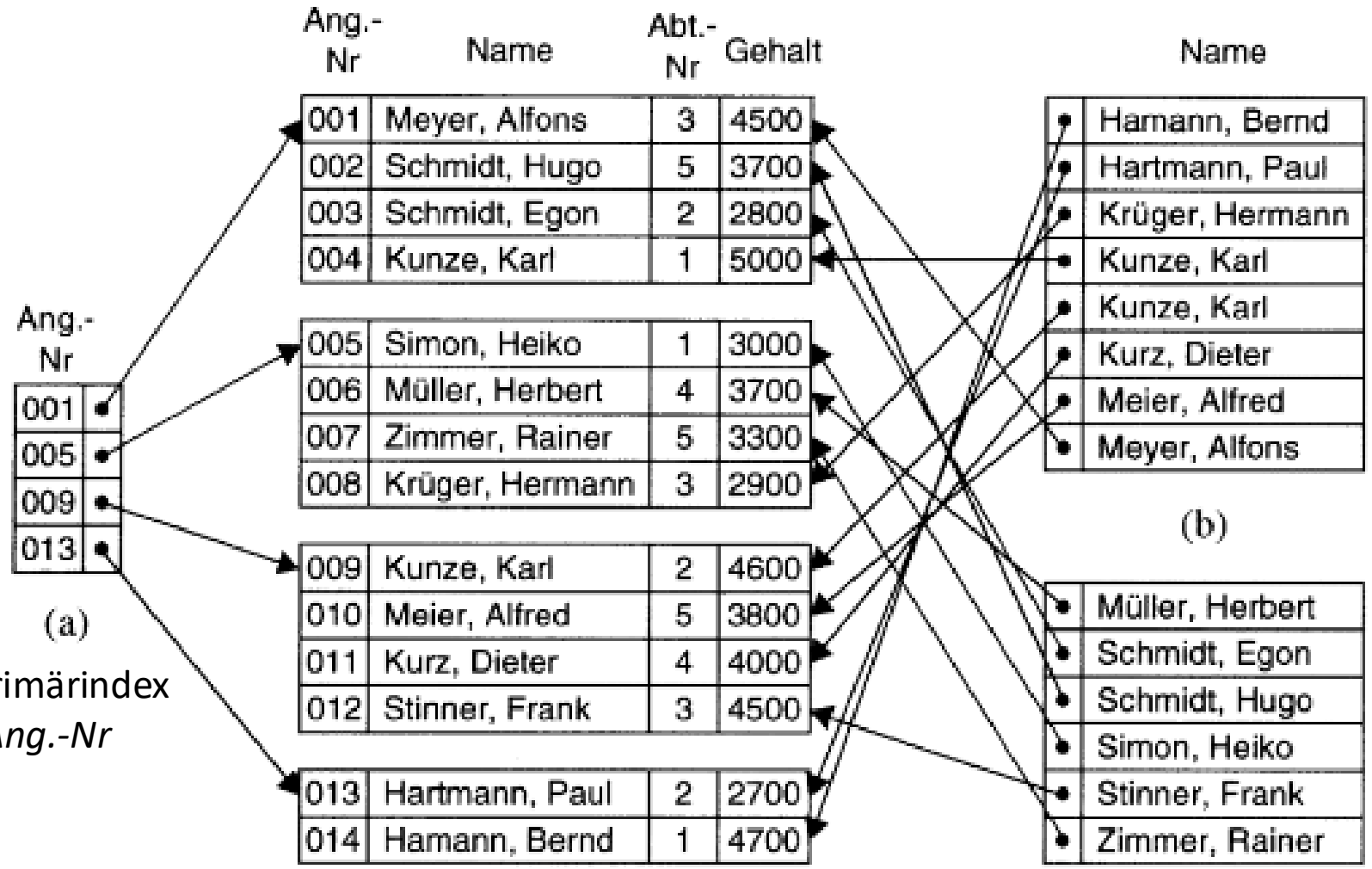
  → die Erhaltung geclusterter Indexe insbesondere bei Aktualisierungen ist relativ teuer

Dichte (dense) und dünne (sparse) Indexe

- Ein Index wird als **dichter Index** bezeichnet, wenn es wenigstens einen Indexeintrag für jeden Suchschlüsselwert (der in einem Datensatz der indizierten Datei auftritt) enthält
  - Alternative (1) für Indexeinträge führt immer zu einem dichten Index
  - Mehrere Indexeinträge können denselben Suchschlüsselwert haben, wenn es Duplikate gibt und wenn wir Alternative (2) benutzen

- Ein **dünner Index** enthält einen Eintrag für jede Seite von Datensätzen der indizierten Datei
  - Üblicherweise referenziert ein Eintrag den ersten Datensatz einer Seite
  - Dünne Indexe setzen notwendigerweise eine Sortierung der Datensätze der indizierten Datei voraus → es kann höchstens einen dünnen Index geben
  - Jeder dünne Index ist gruppiert (clustered)

Dichte und dünne Indexe



Dünner Primärindex über *Ang.-Nr*

Sequentielle Angestellten-Datei

Dichter Sekundärindex über *Name*

Dichte (dense) und dünne (sparse) Indexe

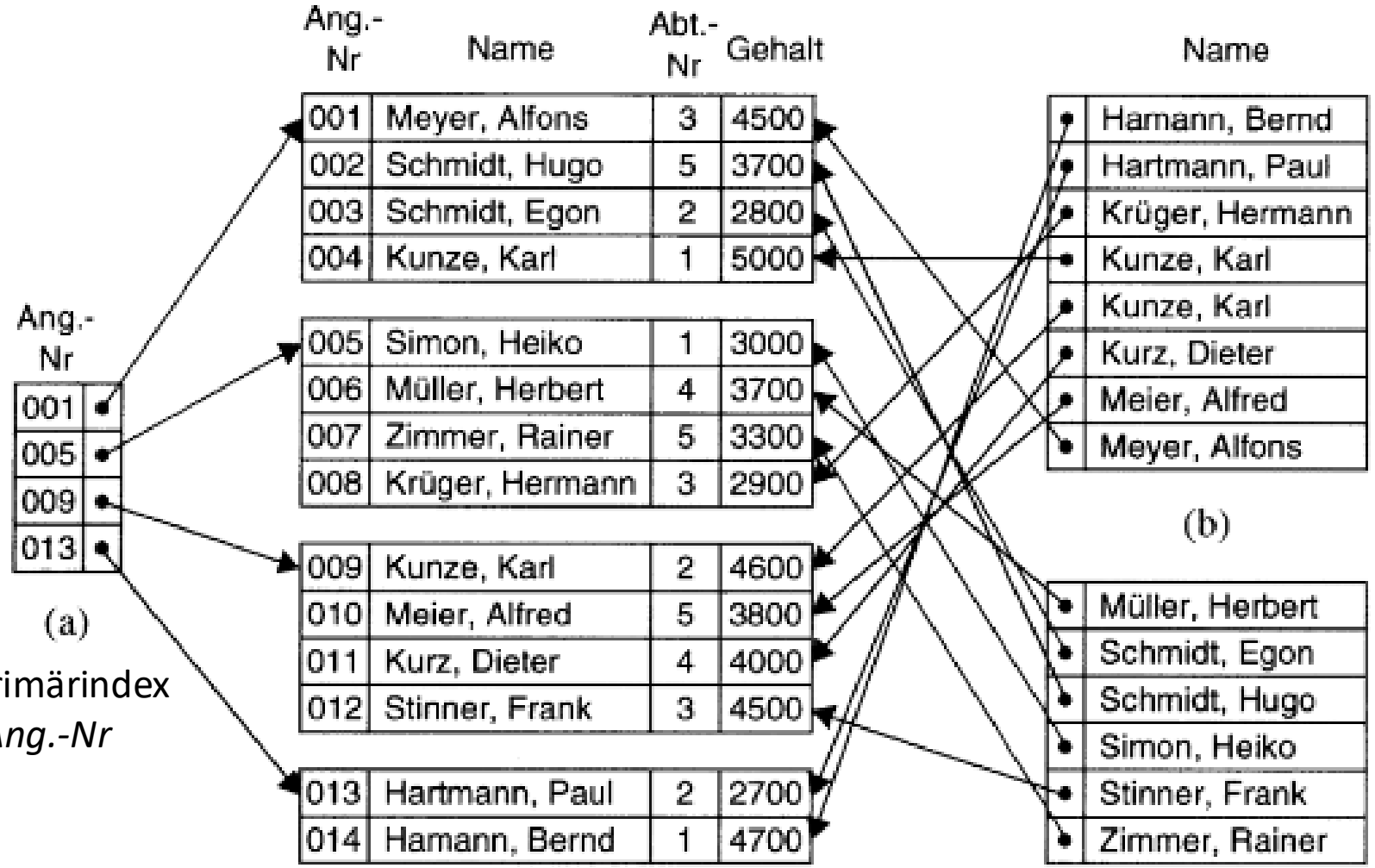
- Das Auffinden eines Datensatzes in einem dünnen Index:
  - zunächst muss der Indexeintrag mit dem größten Suchschlüsselwert, der kleiner oder gleich dem gesuchten Suchschlüsswert ist, ermittelt werden
  - dann beginnen wir bei dem Datensatz, auf der dieser Indexeintrag zeigt, und durchsuchen sequentiell die entsprechenden Seite, bis der gewünschte Datensatz gefunden ist
- Ein dünner Index ist viel kleiner als ein entsprechender dichter Index, da es wenigere Einträge enthält
- Aber, es gibt sehr nützliche Optimisierungstechniken, die auf einem dichten Index beruhen
- Existenztests nehmen bei dünnen Indexen mehr Zeit in Anspruch

Primär- und Sekundärindexe

- Ein Index, der über einen Primärschlüssel definiert und geordnet ist, wird **Primärindex** (primary index) genannt
  - Es kann nur einen Primärindex geben (da es in einer Relation nur einen Primärschlüssel gibt)
  - Es gibt keine Duplikate: nicht mehrere Suchschlüssel mit demselben Wert
  - Hauptproblem: Einfügen und Löschen von Datensätzen
- Ein Index heißt eindeutiger Index, wenn der Suchschlüssel ein Kandidatschlüssel enthält

  → keine Duplikate
- Ein **Sekundärindex** (secondary index) kann Duplikate in den Indexeinträgen enthalten
  - Weil die Datensätze der indexierten Datei (meistens) nicht physisch nach dem Suchschlüssel eines Sekundärindex sortiert sind, kann ein Sekundärindex nur als **dichter Index** auftreten
  - Eine Datei, die einen dichten Sekundärindex bzgl. eines Attributs besitzt, wir auch **invertierte Datei** bzgl. dieses Attributs genannt

Primär- und Sekundärindexe Beispiel



Dünner Primärindex über *Ang.-Nr*

Sequentielle Angestellten-Datei

Dichter Sekundärindex über *Name*

Indexe mit einfachen und zusammengesetzten Suchschlüsseln

- Der Suchschlüssel für einen Index kann mehrere Felder enthalten → solche Feldkombinationen werden als **zusammengesetzte Suchschlüssel (composite search keys)** bezeichnet
- Im Gegensatz, bestehen **einfache Suchschlüssel** aus einem einzigen Feld
- Die entsprechenden Indexe werden **zusammengesetzter Index (composite index)** bzw. **einfacher Index (non-composite index)** genannt

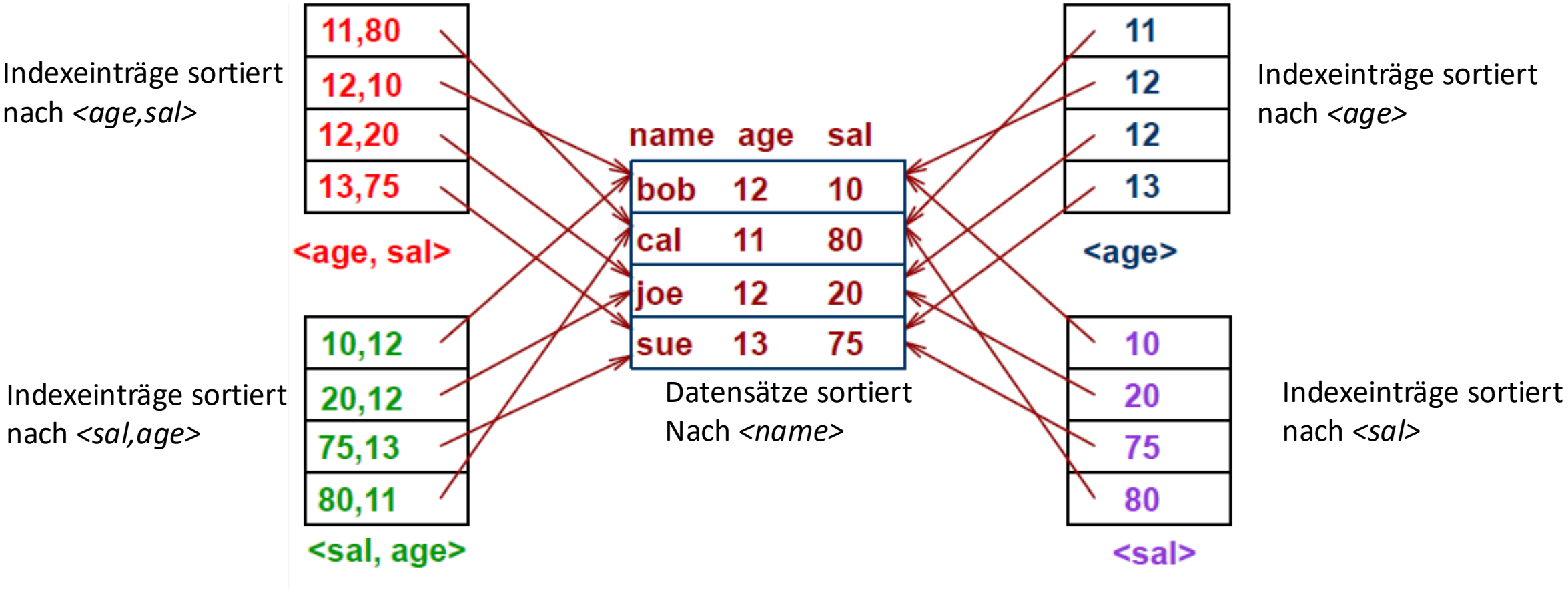
Indexe mit einfachen und zusammengesetzten Suchschlüsseln

- Suche nach einem zusammengesetzen Suchschlüssel:
  - Gleichheitsanfrage (equality query) – jedes Feld des Suchschlüssels wir mit einem Gleichheitsprädikat verknüpft
    - Name = ‘Schmidt’ and Gehalt = 5000
  - Bereichsanfrage (range query) – nicht jedes Feld des Suchschlüssels wird an ein Gleichheitsprädikat gebunden
    - Name = ‘Schmidt’ and Gehalt > 4000
- Bei der Unterstützung von Bereichsanfragen werden zwei Arten von Indexstrukturen unterschieden:
  - **Eindimensionale Indexstrukturen** / one-dimensional index structures
  - **Mehrdimensionale Indexstrukturen** / multi-dimensional index structures

Indexe mit einfachen und zusammengesetzten Suchschlüsseln

- Eindimensionale Indexstrukturen
  - auf die Menge der Suchschlüsselwerte wird eine lineare Ordnung definiert
- Mehrdimensionale Indexstrukturen
  - die Organisation der Indexeinträge erfolgt häufig anhand räumlicher Beziehungen (spatial relationships)
  - Jeder Wert eines zusammengesetzten Suchschlüssels mit k Feldern wird dabei als ein Punkt im k-dimensionalen Raum aufgefasst
  - Einträge werden gemäß ihrer Nähe im zugrundeliegenden k-dimensionalen Raum abgespeichert
  - zB geometrische Indexstrukturen
  - **Nachteil**: Suche auf einem einzigen Feld (Gleichheitsanfrage) bei einer mehrdimensionalen Indexstruktur ist meistens langsamer als bei einer eindimensionalen Indexstruktur

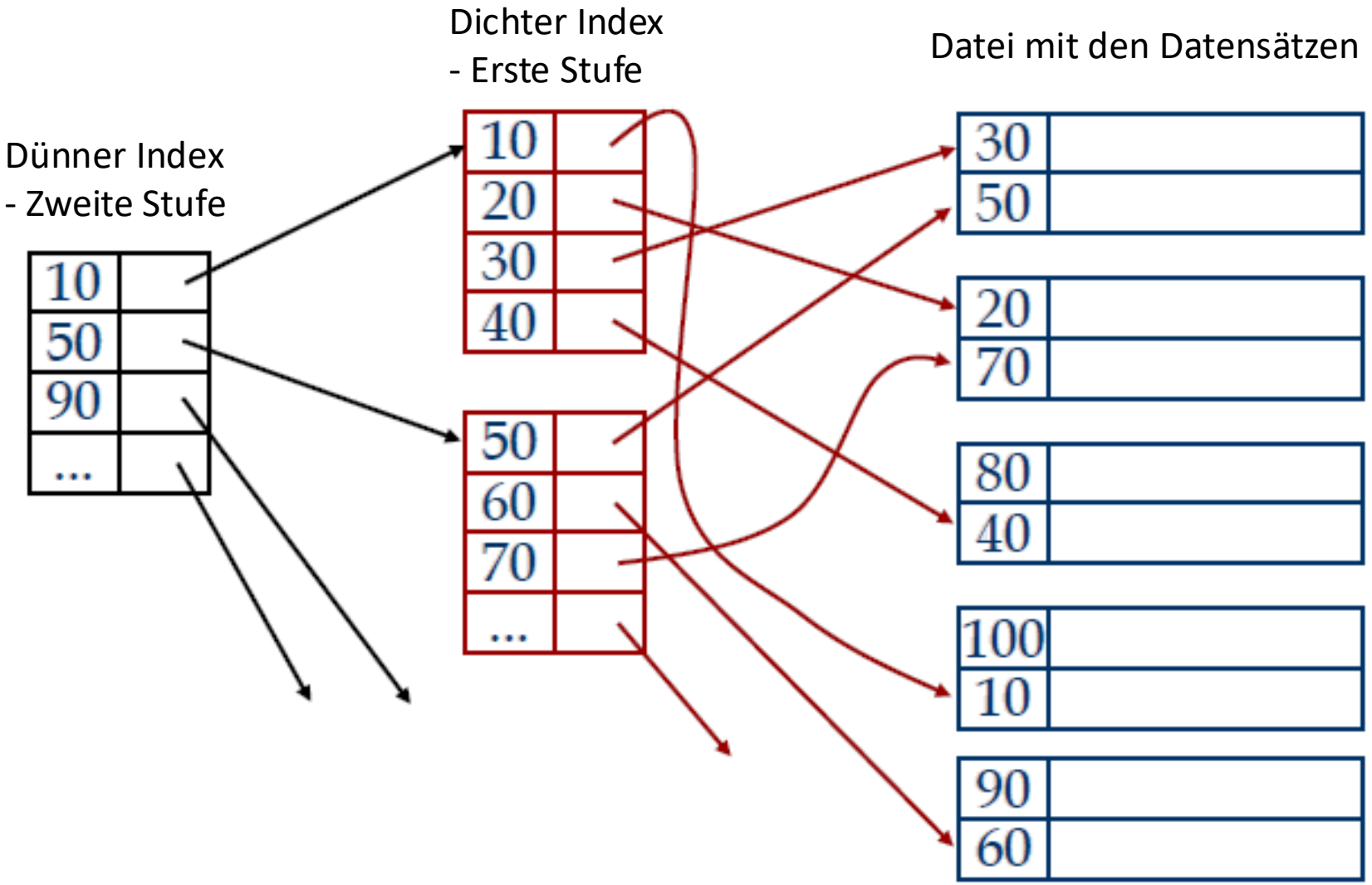
Indexe mit einfachen und zusammengesetzten Suchschlüsseln



Ein- und mehrstufige Indexe

- **Einstufiger Index** - bestehen aus einer einzigen geordneten Datei
  - z.B. alle Indexstrukturen beschrieben bisher
- In der Regel wird binäre Suche auf dem Index angewendet
- Binäre Suche ist effizient, aber um eine höhere Effizienz zu erreichen → mehrstufige Indexe
- Die Idee eines **mehrstufigen Index (multilevel index):** den Teil des zu durchsuchenden Index noch mal zu reduzieren (ein Index zu dem anderen Index)

Mehrstufiger Index - Beispiel



Zugriffsarten, wo Indexe helfen können

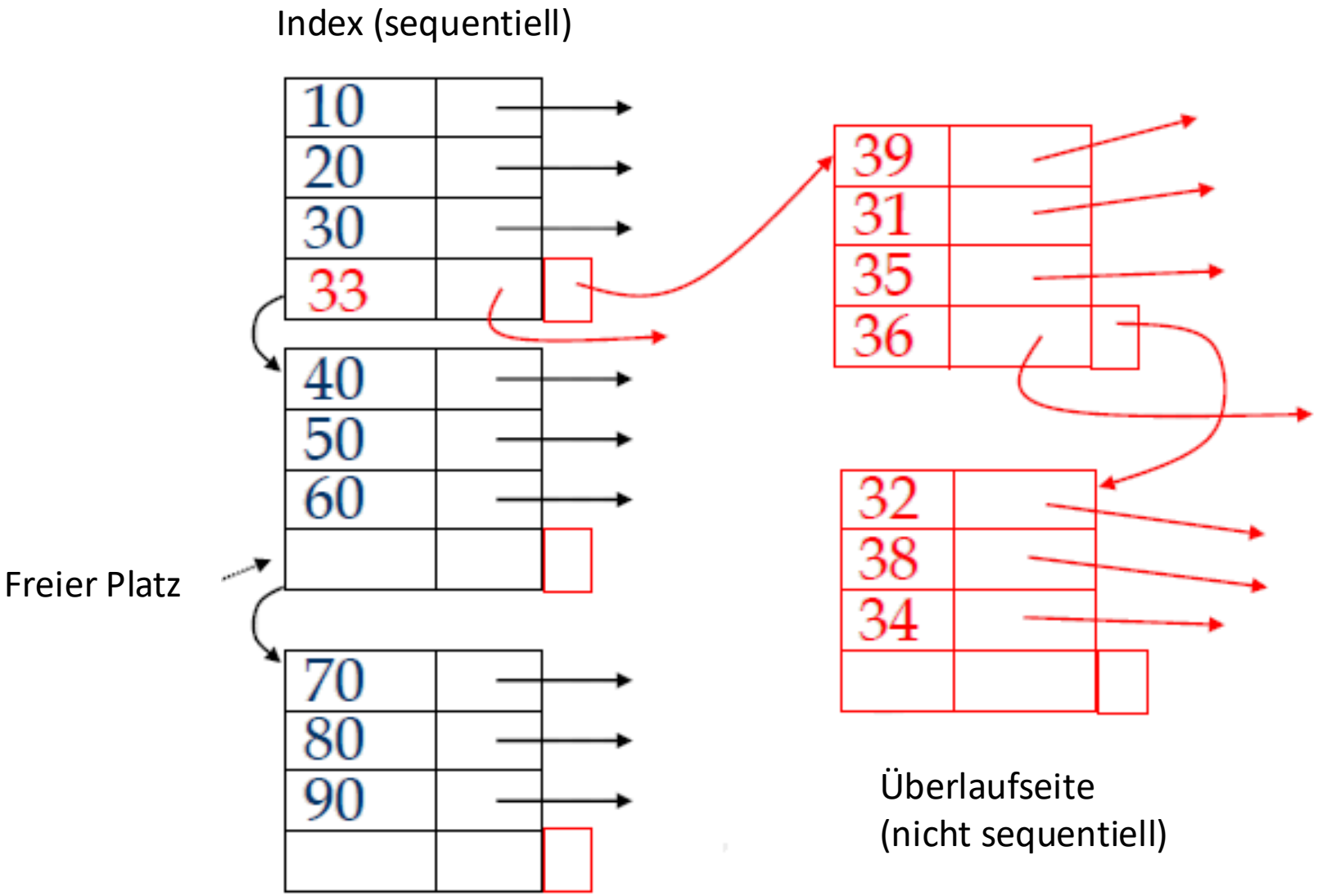
- Sequentieller Zugriff – Indexe erlauben bzgl. eines gegebenen Suchschlüssels einen sequentiellen (sortierten) Zugriff auf eine indizierte Datei
- Direkter Zugriff – auf einzelne oder mehrere Datensätze kann für einen gegebenen Suchschlüssel „direkt“ zugegriffen werden
  - z.B. Finde die Angestellten mit dem ID = 123
- Bereichszugriff – bei Bereichsanfragen
  - z.B. Finde alle Angestellte mit dem Gehalt zwischen 4000 und 5000
- Existenztest – die Frage nach der Existenz eines Datensatzes kann ohne Zugriff auf die indizierte Datei von der Indexstruktur selbst beantwortet werden
  - z.B. Gibt es einen Angestellten mit dem ID = 124?

Konventionelle Indexe

- Vorteile:
  - Einfach
  - Der Index ist eine sequentielle Datei → gut für Scans

- Nachteile
  - Einfügen von Datensätze ist teuer und man kann die Sequentialität verlieren

Konventionelle Indexe



Den Workload verstehen

- Für jede Abfrage (SELECT) aus dem Workload:
  - Welche Relationen braucht die Abfrage?
  - Welche Attribute ruft die Abfrage ab?
  - Welche Attribute kommen in Selektion- oder Join-Bedingungen vor?
    Wie einschränkend (restrictive) sind diese Bedingungen?

- Für jede Änderung (INSERT, DELETE, UPDATE) aus dem Workload:
  - Welche Attribute kommen in Selektion- oder Join-bedingungen vor?
  - Welcher Typ von Änderung und welche Attribute werden geändert

Auswahl des Index

- Welchen Index sollten wir erstellen und mit welcher Indexierungstechnik?
- Eine Möglichkeit:
  - Denke an die wichtigsten Abfragen, der Reihe nach
  - Denke an den besten Ausführungsplan mit den schon existierenden Indexen
  - Überlege ob es einen besseren Ausführungsplan gibt mit zusätzlichen Indexen und falls ja, erstelle den neuen Index
- Natürlich muss man dafür verstehen wie der DBMS den Ausführungsplan berechnet, aber man kann am Anfang einfache Abfragen auf einer einzigen Tabelle berücksichtigen
- Bevor man einen Index erstellt, muss man auch seine Auswirkung auf Änderungen berücksichtigen:
  - Indexe können Abfragen beschleunigen, aber Updates verlangsamen
  - Indexe können auch viel Speicherplatz brauchen

Guidelines für die Index Auswahl

- Attribute in dem WHERE Klausel sind Kandidaten für den Suchschlüssel:
  - Gleichheitsanfrage (exact match) → Hash Index
  - Bereichsanfrage → Baumbasierte Indexstrukturen
    - Clustering ist besonders nützlich für Bereichsanfrage und es kann manchmal auch für Gleichheitsanfragen nützlich sein wenn es viele Duplikate gibt
- Indexe mit zusammengesetzen Suchschlüsseln können nützlich sein wenn der WHERE Klausel mehrere Bedingungen enthält
  - Die Reihenfolge der Attribute ist in einer Bereichsanfrage wichtig
- Versuche den Index auszuwählen, der für die meisten Abfragen nützlich ist
- Da nur ein Index geclustert sein kann, wähle den Index, der für die wichtigen Abfragen, die ein geclusterten Index brauchen könnten, nützlich ist

Beispiel

- Die Datensätze der Tabelle *Studenten* sind in einer sortierten Datei gespeichert (nach dem Attribut *Age)*
- Jede Seite kann 3 Datensätze speichern
- Schreibe die Indexeinträge für jede der folgenden Indexe (ein Datensatz kann mit <page_id, slot_no> identifiziert werden)

ID	Name	Age	Note	Kurs
123	Melanie	18	7.8	DB1
124	Georg	19	8.0	DB1
110	Jan	25	9.4	Alg1
112	Sammy	26	9.2	DB1
100	Sammy	26	9.8	Alg1

ID	Name	Age	Note	Kurs
123	Melanie	18	7.8	DB1
124	Georg	19	8.0	DB1
110	Jan	25	9.4	Alg1
112	Sammy	26	9.2	DB1
100	Sammy	26	9.8	Alg1

1. Age – dicht, Alternative (1)
   - Die Datei selbst
2. Age – dicht, Alternative (2)
   - (18, <1,1>), (19, <1,2>), (25, <1,3>), (26, <2,1>), (26, <2,2>)
3. Age – dicht, Alternative (3)
   - (18, <1,1>), (19, <1,2>), (25, <1,3>), (26, <2,1>, <2,2>)
4. Age – dünn, Alternative (1)
   - Nicht möglich (wegen der Definition)

ID	Name	Age	Note	Kurs
123	Melanie	18	7.8	DB1
124	Georg	19	8.0	DB1
110	Jan	25	9.4	Alg1
112	Sammy	26	9.2	DB1
100	Sammy	26	9.8	Alg1

5. Age – dünn, Alternative (2)
   - (18, <1,1>), (26, <2,1>)
6. Age – dünn, Alternative (3)
   - (18, <1,1>), (26, <2,1>, <2,2>)
7. Note – dicht, Alternative (1)
   - In dem Index werden Datensätze nach Note sortiert: 7.8, 8.0, 9.2, 9.4, 9.8
8. Note – dicht, Alternative (2)
   - (7.8, <1,1>), (8.0, <1,2>), (9.2, <2,1>), (9.4, <1,3>), (9.8, <2,2>)

ID	Name	Age	Note	Kurs
123	Melanie	18	7.8	DB1
124	Georg	19	8.0	DB1
110	Jan	25	9.4	Alg1
112	Sammy	26	9.2	DB1
100	Sammy	26	9.8	Alg1

9. Note – dicht, Alternative (3)
   - (7.8, <1,1>), (8.0, <1,2>), (9.2, <2,1>), (9.4, <1,3>), (9.8, <2,2>)
10. Note – dünn, Alternative (1)
    - Nicht möglich (wegen der Definition)
11. Note – dünn, Alternative (2)
    - Nicht möglich, weil die Datei nach dem Suchschlüssel (*Note*) nicht geordnet ist
12. Note – dünn, Alternative (3)
    - Nicht möglich, weil die Datei nach dem Suchschlüssel (*Note*) nicht geordnet ist