Technische Hochschule Rosenheim

Kapitel 10 – Physische Datenorganisation

Vorlesung Datenbanken

Prof. Dr. Kai Höfig

Kapitel 9: Physische Datenorganisation

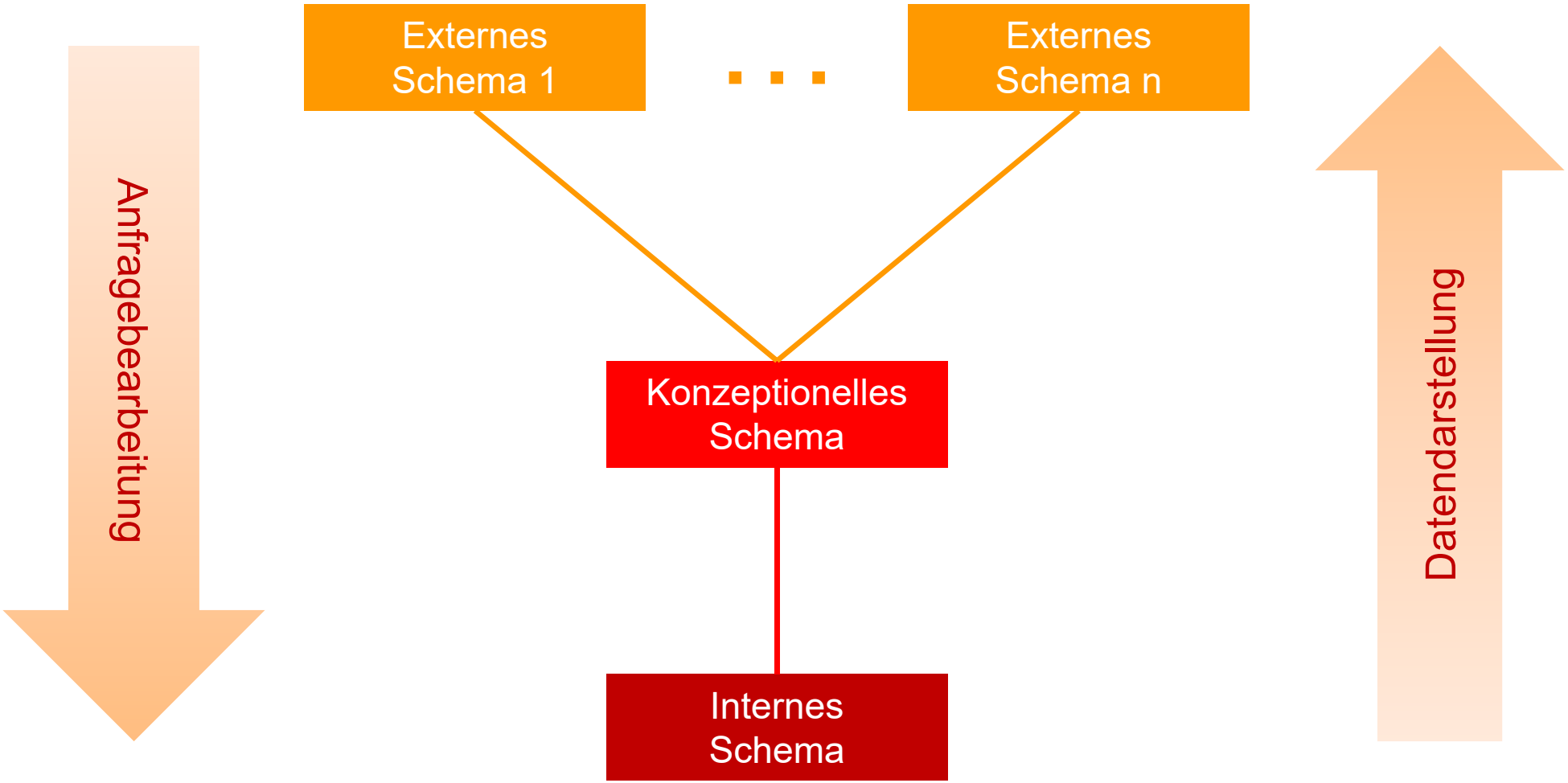
In diesem Kapitel wollen wir folgende Fragen betrachten

- Wie legt ein DBMS die Daten auf einem Sekundärspeicher ab?
- Wie beschleunigt man den Zugriff auf Daten mittels Indexstrukturen?
- Welche sind die wichtigsten Indexstrukturen und wie funktionieren sie?
- Wie unterscheiden sich primäre und sekundäre Indizes?
- Wie unterscheiden sich geclusterte von nicht-geclusterten Indizes?
- Wie legt man Indizes in SQL-Server bzw. Oracle an?

Literatur: Biberbuch Kap --; CompleteBook Chap 13, 14

Kapitel 13: Dateiorganisation und Indizes

Schema-Architektur – allgemein (Wh)



Schema-Architektur – allgemein (Wh)

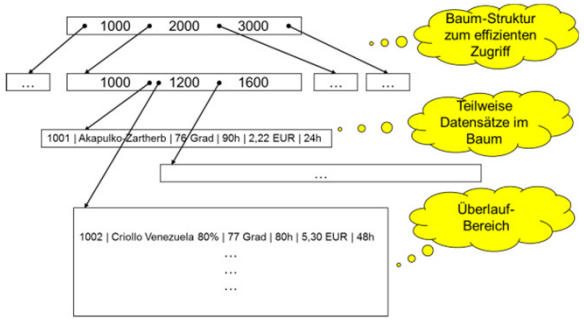
Name	Zutaten	Temperatur	Rührdauer
Akapulko-Zartherb	30% Criollo Venezuela 20% Nacional Ecuador 48% Zucker 2% Vanille	76 Grad	90h
Criollo Venezuela 80%	80% Crillo Venezuela 20% Zucker	77 Grad	80h
Milch Zartschmelzend	35% Arriba Venezuela 65% Zucker	78 Grad	30h

Name	Kakao	Preis	Lieferzeit
Akapulko-Zartherb	50%	2,22 EUR	24h
Criollo Venezuela 80%	80%	5,30 EUR	48h
Milch Zartschmelzend	35%	1,50 EUR	12h

SNr	Name	Temperatur	Rührdauer	Preis	Lieferzeit
1001	Akapulko-Zartherb	76 Grad	90h	2,22 EUR	24h
1002	Criollo Venezuela 80%	77 Grad	80h	5,30 EUR	48h
1003	Milch Zartschmelzend	78 Grad	30h	1,50 EUR	12h

ZNr	Name	IstKakao
2001	Criollo Venezuela	True
2002	Nacional Ecuador	True
2003	Arriba Venezuela	True
2004	Vanille	False
2005	Zucker	false

SNr	ZNr	Anteil
1001	2001	30%
1001	2002	20%
1001	2005	48%
1001	2004	2%
1002	2001	80%
1002	2005	20%
1003	2003	35%
1003	2005	65%

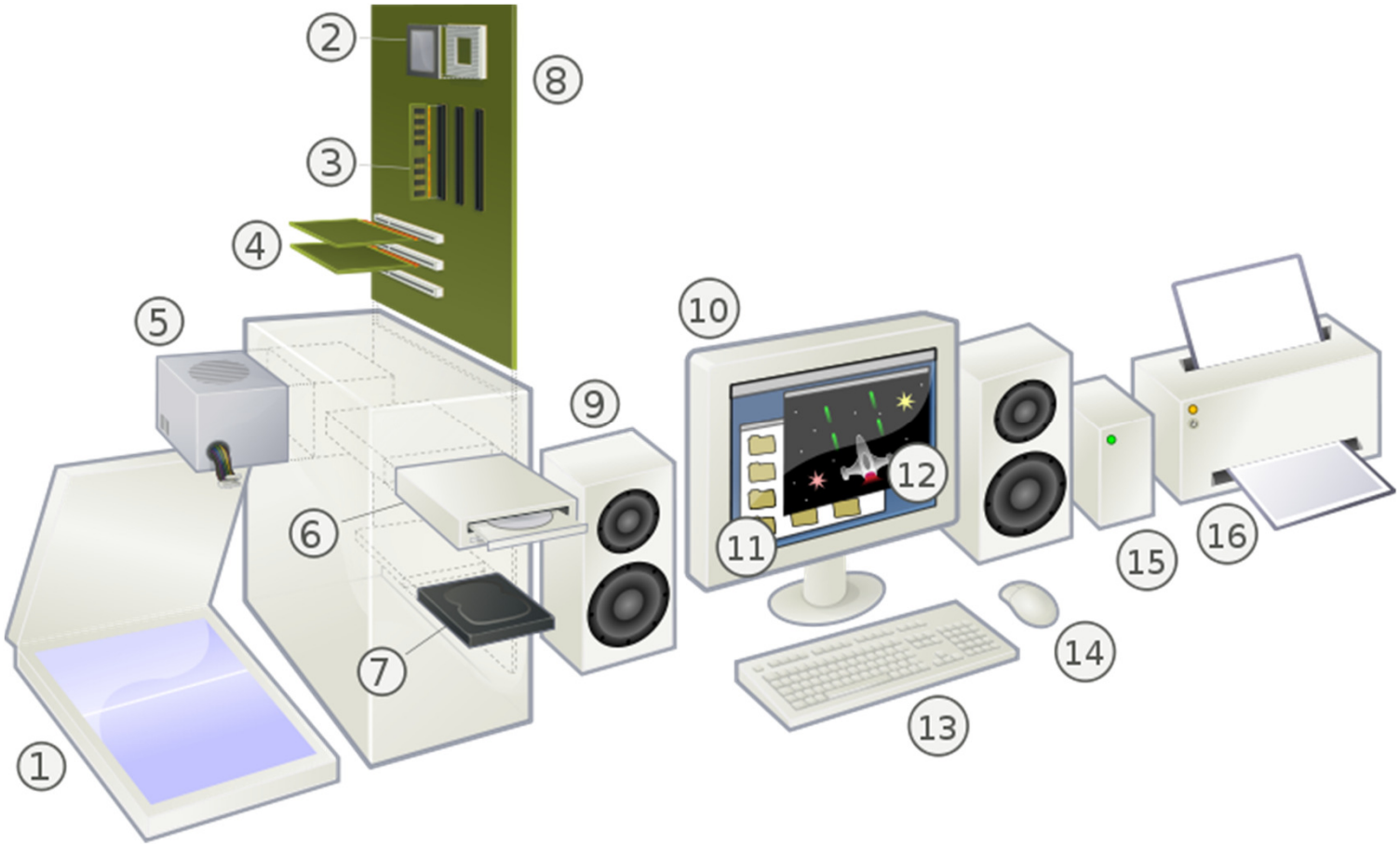


Anfragebearbeitung

Datendarstellung

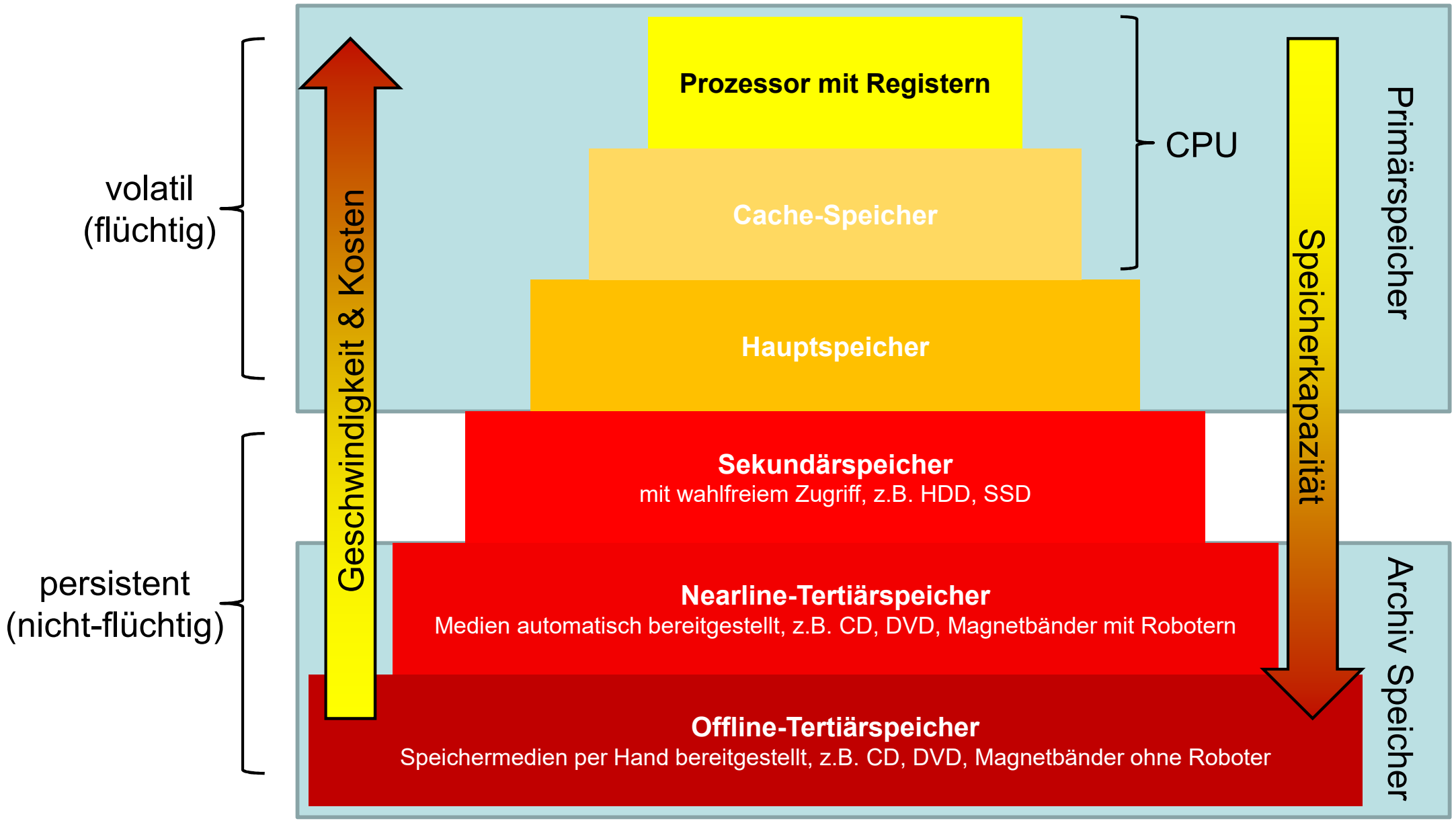
Kapitel 13: Dateiorganisation und Indizes

Aufbau eines modernen Computers



1) Scanner
2) CPU (Microprocessor)
3) Memory (RAM)
4) Expansion cards
5) Power supply
6) Optical disc drive
7) Storage (Hard disk or SSD)
8) Motherboard
9) Speakers
10) Monitor
11) System software
12)Application software
13) Keyboard
14) Mouse
15) External hard disk
16) Printer

Speicherhierarchie



Zielsetzung der Speicherhierarchie

- Trade-Off zwischen Performance und Persistenz
- Verschiedene Zwecke der Speichermedien
  - Daten zur Verarbeitung bereitstellen
    → Register, Cache, Hauptspeicher, Sekundärspeicher
  - Daten langfristig speichern und trotzdem schnell verfügbar halten
    → Sekundärspeicher
  - Daten sehr langfristig und preiswert archivieren unter Inkaufnahme etwas längerer Zugriffszeiten
    → Tertiär/Archivspeicher
- Je nach Anwendung sind unterschiedliche Ansätze sinnvoll
  - Persistenz extrem wichtig → Transaktionale Systeme
  - Performance extrem wichtig → Main-Memory Databases

Kapitel 13: Dateiorganisation und Indizes

DBMS und Betriebssystem (1)

- Datenbankmanagementsysteme arbeiten mit dem Betriebssystem zusammen, um den Sekundärspeicher zu verwalten

- Verschiedene Arten der Zusammenarbeit möglich
  - BS verwaltet Dateien
    DBMS verwendet eine Datei pro Relation und BS Funktionen zum Zugriff
    → selten verwendet

  - BS verwaltet Dateien
    DBMS verwaltete selbständig Relationen und Zugriffspfade innerhalb der Dateien mit eigenem „Dateisystem“
    → typischer, häufigster Ansatz in kleinen/mittleren Systemen

  - BS wird nicht verwendet
    DBMS verwaltet Sekundärspeicher selbständig („raw device“)
    → vor allem in großen OLTP Systemen

DBMS und Betriebssystem (2)

- Warum nicht immer BS-Dateiverwaltung?
  - Betriebssystemunabhängigkeit
  - BS-Dateien auf maximal einem Medium
  - Betriebssystemseitige Pufferverwaltung von Blöcken des Sekundärspeichers im Hauptspeicher genügt nicht den Anforderungen des Datenbanksystems
  - (In 32-Bit-Betriebssystemen: Dateigröße 4 GB maximal)
- Eine Datenbank besteht meist aus mehreren Dateien
  - Trennung von Daten und Log Dateien (Transaktionslogs)
  - Performancesteigerung durch Partitionierung auf mehrere Datenträger
  - Ggf. Überwindung der maximalen BS-Dateigröße

Aufbau einer Festplatte

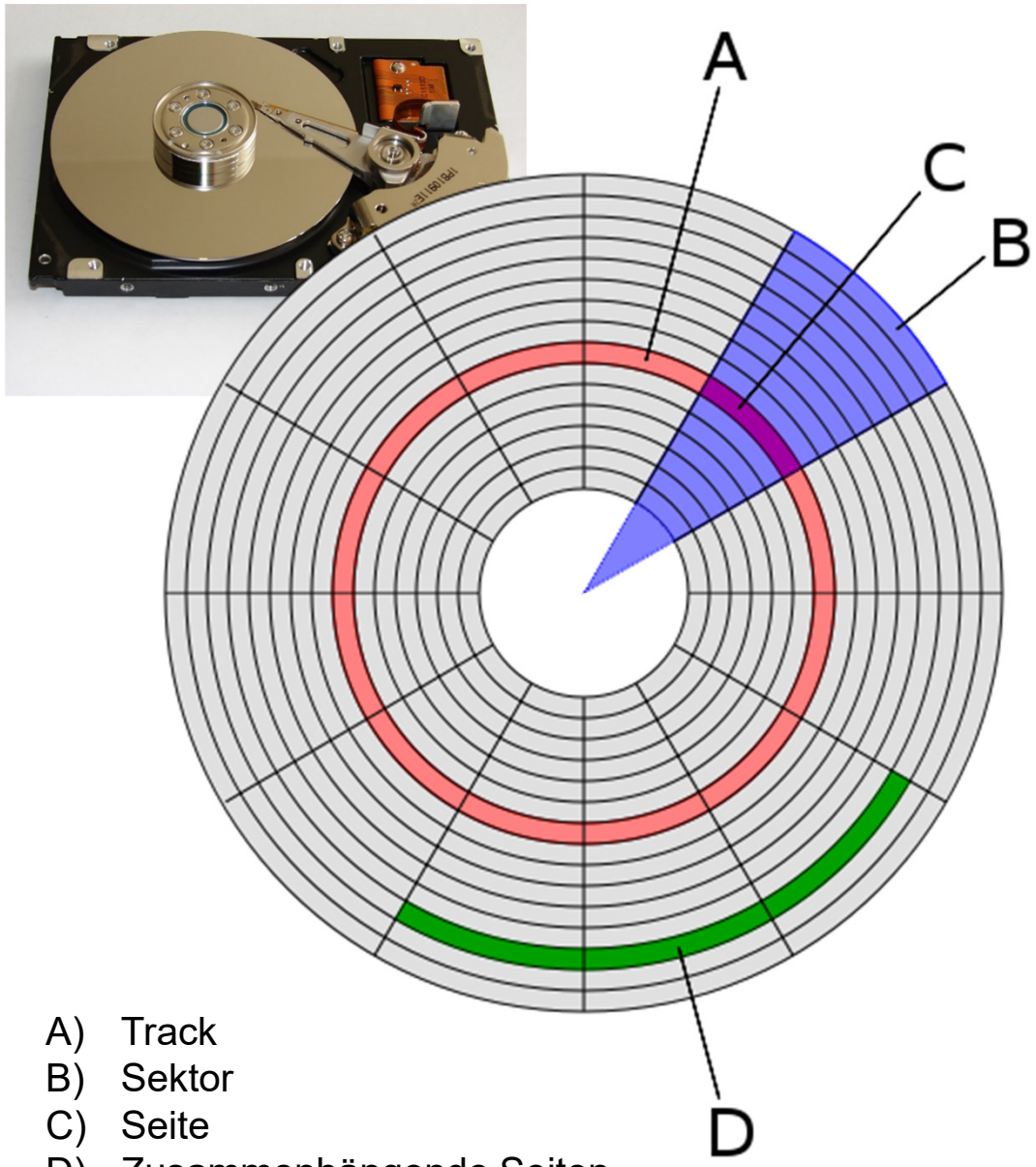
- Schreib-/Lesekopf auf einer bestimmten Spur platzieren: Seek Time

- Bis ein bestimmter Block gelesen wird auf der Spur: Latenzzeit

- Die Seek Time nimmt auf Grund des mechanischen Vorgangs die meiste Zeit beim Lesen eines Datums in Anspruch.

- Daher bedeutet bei Suchvorgängen, wie sie in DBMS häufig vorkommen, die direkte Wahl der richtigen Spur zu einem gesuchten Datum eine hohe Zeitersparnis.

- Dabei helfen uns Indizes



A) Track
B) Sektor
C) Seite
D) Zusammenhängende Seiten

Dateien und Seiten (1)

- Unterste Ebene des DBMS verwaltet Dateien (Files)

- Seite (Page) ist die kleinste physische Einheit der Verwaltung
  - Jede Datei besteht aus Seiten, die von 0 bis n durchnummeriert sind
  - Seiten enthalten i.allg. mehrere Tupel (je nach deren Größe) bzw. Zugriffspfade
  - Seite ist kleinste Einheit, die zwischen Sekundärspeicher und Hauptspeicher-Puffer übertragen wird
  - Seiten haben normalerweise eine feste Größe
    - MS SQL Server: 8KB
    - Oracle, IBM DB2: zwischen 4 und 64 KB

Dateien und Seiten (2)

- Da idR. der Primärspeicher sehr viel kleiner ist als der Sekundärspeicher und im Sekundärspeicher die Daten persistent gehalten werden, werden Seiten aus dem Sekundärspeicher im Primärspeicher gepuffert (Caching).

- Wird eine bestimmte Seite bei einer Datenbankoperation benötigt und befindet sie sich bereits im Puffer, kann auf diese Seite sehr schnell zugegriffen werden (Cache Hit). Befindet sie sich nicht im Puffer (Cache Miss), muss sie aus dem Sekundärspeicher geladen werden und wird ihrerseits wieder im Puffer abgelegt.

- Dabei verdrängt sie möglicherweise eine bereits vorhandene Seite im Puffer nach unterschiedlichen Verdrängungsstrategien, z.B.
  - FIFO, First in first out
  - LRU, Last recently used
  - LFU, Last frequently used
  - Random

Dateien und Seiten (3)

Dienste, die die unterste Ebene des DBMS zur Verfügung stellt

- Allokation oder Deallokation von Speicherplatz (Seiten)
- Allokation möglichst so, dass logisch aufeinanderfolgende Datenbereiche (etwa einer Relation) auch möglichst in aufeinander folgenden Seiten der Platte gespeichert werden
- Holen oder Speichern von Seiteninhalten
- Nach vielen Update-Operationen: Reorganisationsmethoden

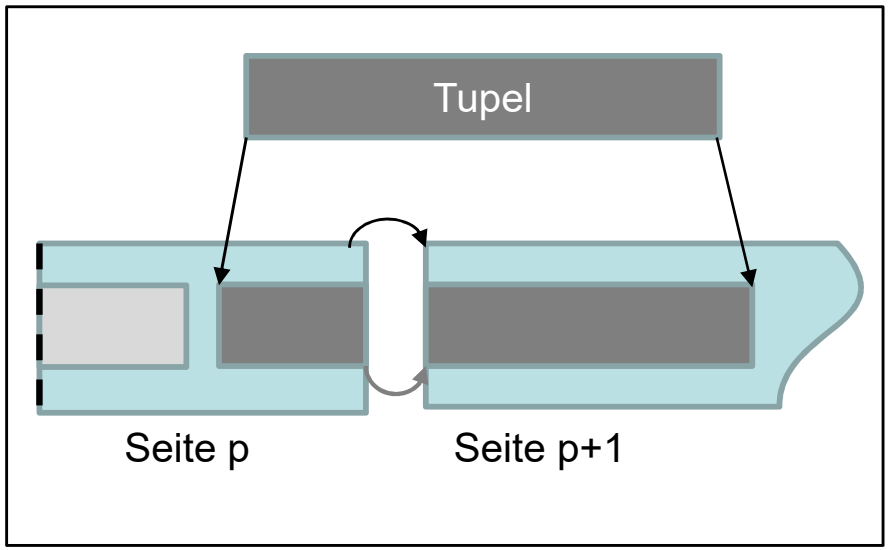
Zuordnung von Tupel zu Seiten (1)

- Seite enthält i.allg. mehrere Tupel
  ➔ Datensätze (eventuell variabler Länge) müssen in die aus einer fest vorgegebenen Anzahl von Bytes bestehenden Seiten eingepasst werden = Blocken

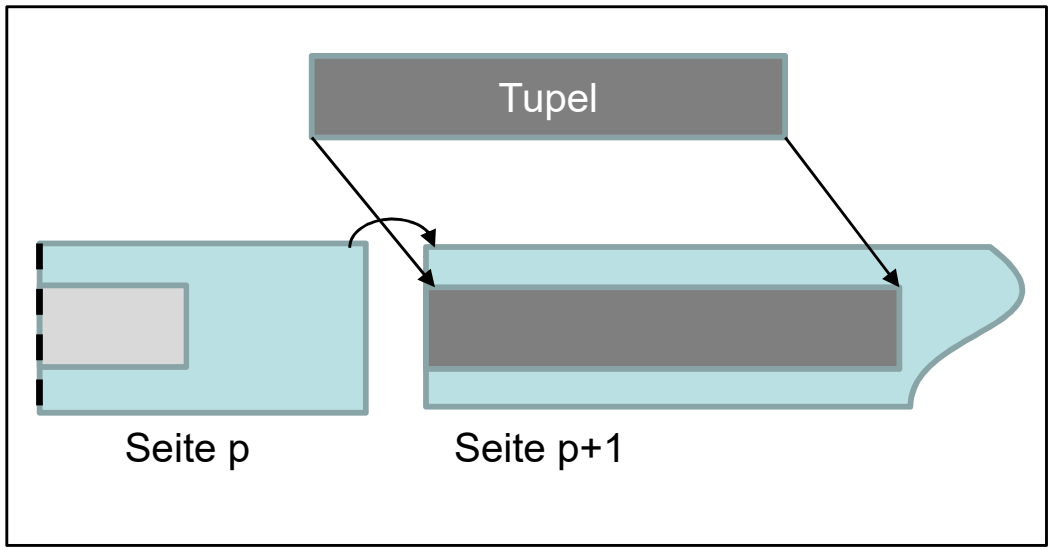
- Blocken abhängig von variabler oder fester Feldlänge der Datenfelder
  - Datensätze (Tupel) mit variabler Satzlänge: höherer Verwaltungsaufwand beim Lesen und Schreiben, Satzlänge immer wieder neu ermitteln
  - Datensätze (Tupel) mit fester Satzlänge: höherer Speicheraufwand

Zuordnung von Tupel zu Seiten (2)

- Blockungs-Techniken
  - Nichtspannsätze: jedes Tupel in maximal einer Seite
  - Spannsätze: Tupel eventuell in mehreren Seiten



Spannsatz

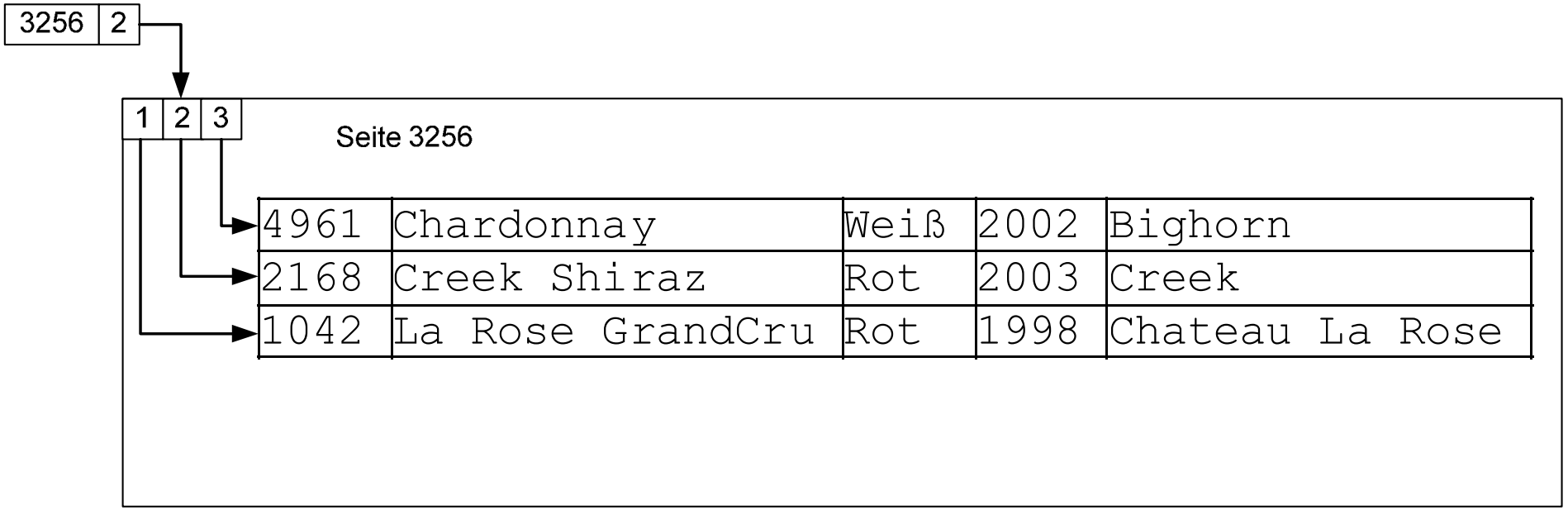


Nichtspannsatz

- Standard: Nichtspannsätze (nur im Falle von BLOBs oder CLOBs Spannsätze üblich)

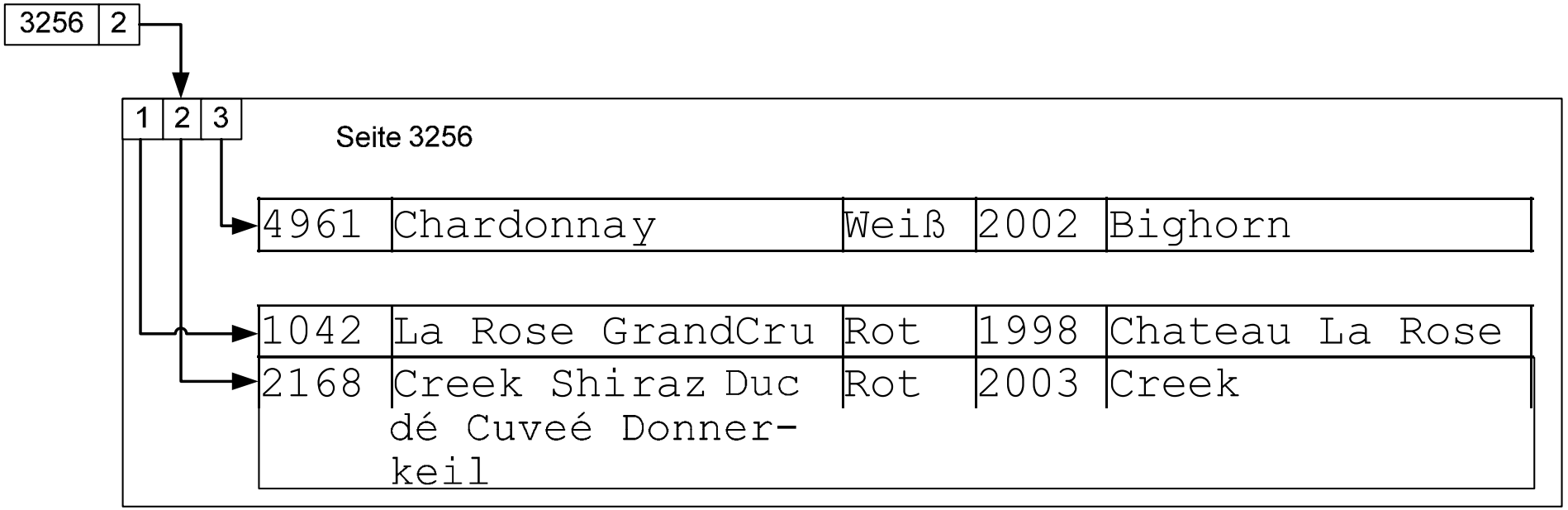
Speichern von Tupeln auf Seiten

- Mehrere Tupel werden in einer Seite gespeichert.
- Ein Tupel-Identifkator (TID) Identifiziert eindeutig ein Tupel in einer Seite. Dies hilft bei der Umsetzung von Indexstrukturen (später).



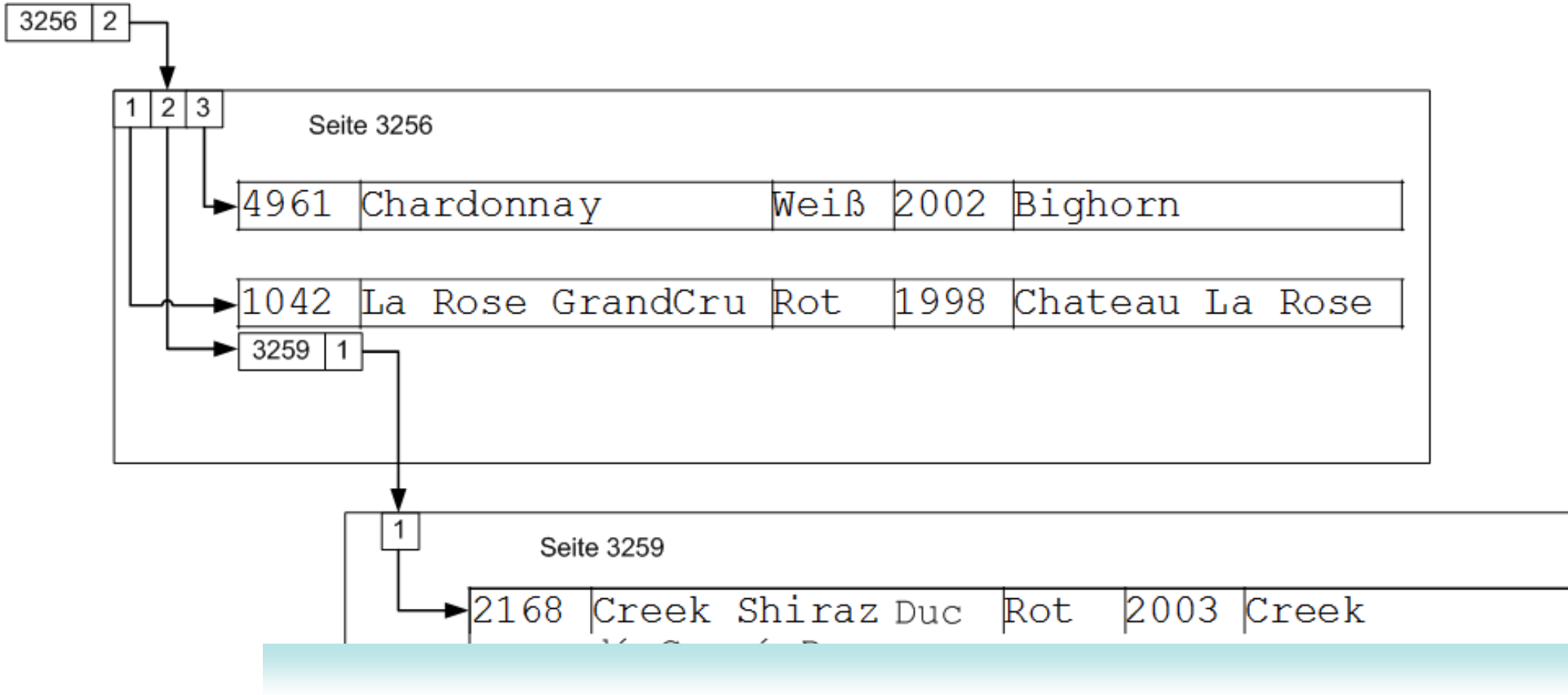
Verschieben von Tupeln innerhalb einer Seite

- Wird ein Tupel geringfügig vergrößert und verschiebt sich innerhalb einer Seite, bleibt der TID unverändert.



Verdrängung eines Tupels von einer Seite

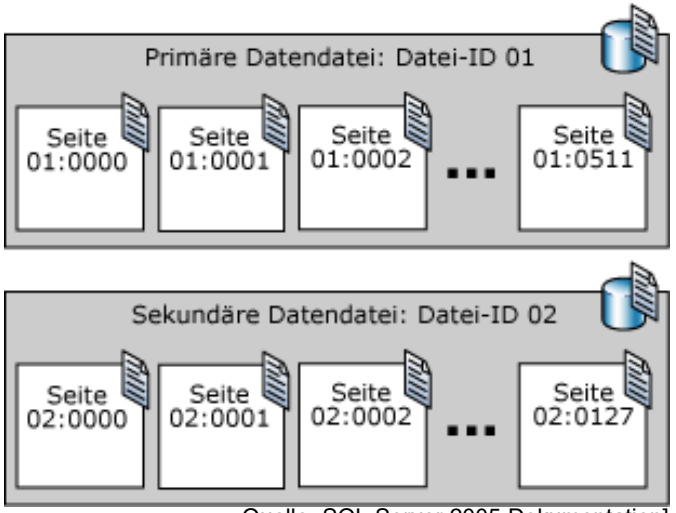
- Wird ein Tupel zu groß für eine Seite, wird es in eine separate Seite ausgelagert. Dabei kann der ursprüngliche TID bestehen bleiben, wenn an die Stelle des ursprünglichen Tupels ein Verweis gesetzt wird.



Beispiel: Physische Datenspeicherung SQL Server (1)

- Jede SQL Server-Datenbank hat mindestens zwei Betriebssystemdateien: eine Datendatei und eine Protokolldatei.

- Datendateien
  - enthalten Daten und Objekte wie z. B. Tabellen, Indizes, gespeicherte Prozeduren und Sichten.
  - enthalten Benutzerdaten und Systemdaten
  - Primäre Datendatei. Endung .mdf
  - Optional: weitere Sekundäre Datendateien. Endung .ndf
  - können für die Zuordnung und Verwaltung in Dateigruppen zusammengefasst werden.



Quelle: SQL Server 2005 Dokumentation]

- Protokolldateien
  - enthalten die Informationen, die zum Wiederherstellen aller Transaktionen in der Datenbank erforderlich sind (Transaktionsprotokoll).
  - Endung: .ldf

Beispiel: Physische Datenspeicherung SQL Server (2)

- Jede Datenbank besitzt eine primäre Dateigruppe.
  - enthält die primäre Datendatei sowie ggf. alle sekundären Dateien, die nicht in anderen Dateigruppen gespeichert werden.
- Benutzerdefinierte Dateigruppen können erstellt werden.
  - Zweck: Datendateien der Verwaltung, Datenzuordnung und -verteilung zu Gruppen zusammenzufassen.
- Beispiel:
  - Drei Dateien (`Wein1.mdf`, `Wein2.ndf` und `Wein3.ndf`) auf drei unterschiedlichen Datenträgern erstellen
  - Zusammenfassen zur Dateigruppe `Primary`
  - Erstellen einer Tabelle `weine` in dieser Dateigruppe

  ➔ Leistungssteigerung: Queries über `weine` werden über alle drei Datenträger verteilt.

Beispiel: Physische Datenspeicherung im SQL Server 2005 (3)

- Beispiel in SQL:

```
CREATE DATABASE WeinDB
  ON PRIMARY
     (NAME = Wein1_dat,
      FILENAME = 'D:\databases\weindd\wein1.mdf', SIZE =100MB),
     (NAME = Wein2_dat,
      FILENAME = 'E:\databases\weindd\wein2.ndf', SIZE =100MB),
     (NAME = Wein3_dat,
      FILENAME = 'F:\databases\weindd\wein3.ndf', SIZE =100MB),
  LOG ON
     (NAME = Wein_log,
      FILENAME = 'D:\databases\weindd\wein.ldf'', SIZE = 5MB);


CREATE TABLE WeinDB.weine (WeinID ...) ON PRIMARY;
```

Kapitel 13: Dateiorganisation und Indizes

Dateiorganisation

- Dateiorganisationsform: Form der Speicherung der internen Relation
  - Unsortierte Speicherung von internen Tupeln: Heap-Organisation
  - Sortierte Speicherung von internen Tupeln: Sequenzielle Organisation
  - Gestreute Speicherung von internen Tupeln: Hash-Organisation
  - Speicherung von internen Tupeln in mehrdimensionalen Räumen: mehrdimensionale Dateiorganisationsformen
- Üblich:
  - Sortierung oder Hash-Funktion über Primärschlüssel;
  - Sortierte Speicherung plus zusätzlicher Primärindex über Sortierattributen: Index-sequenzielle Organisationsform

Indizes - Motivation

- Physische Organisation der Datei hat starke Auswirkung auf die Effizienz von Queries
  - Beispiel: wenn die `WEINE` Tabelle nach `WeinID` sortiert gespeichert ist, können wir sehr effizient alle Tuple mit `34215 < WeinID < 57363` bestimmen. Um jedoch alle Tupel mit `name='Pinot Noir'` zu bestimmen, müssen wir die Datei komplett durchsuchen.

- Datei kann aber nur auf eine Art (Reihenfolge) organisiert sein
  - d.h. maximal in einer Sortierreihenfolge

➔ Notwendigkeit, auf Tupel in unterschiedlichen Sortierreihenfolgen effizient zugreifen zu können
  - ohne die Datei (teuer) physisch neu sortieren/komplett durchsuchen zu müssen

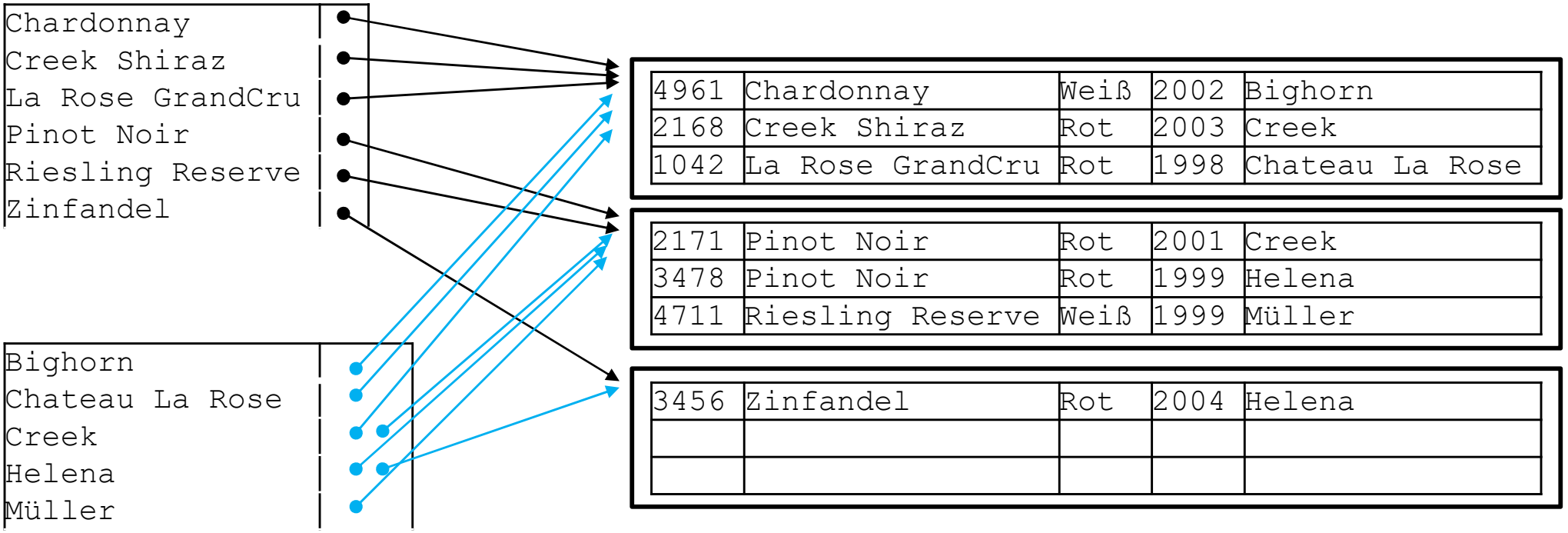
➔ Index: zusätzliche physische Struktur, die den Zugriff beschleunigt

Indizes

- Ein **Index** bildet den Wert einer Spalte/eines Attributes (oder eine Kombination von Attributen) auf die (logischen) Tupel ab.
- Index-Einträge sind nach dem Wert des indizierten Attributes geordnet
- Typische Implementierung von Indizes: Speichern von Schlüssel/Wert Paaren
  - („Attributwert“, „Menge von Seiten, die Tupel mit diesem Attributwert enthalten“)
  - Attributwert wird auch Suchschlüssel (Search Key) genannt
  - Mengen von physischen Seiten meist durch Zeiger (Pointer) umgesetzt
- Dadurch: (erheblich) Reduktion der Anzahl Seiten (d.h. i.allg. Festplattenzugriffe), die das DBMS lesen und prüfen muss.
- Index ist vergleichbar dem Stichwortverzeichnis in einem Buch.

Beispiel

- WEINE Tabelle
  - Physisch nach `name` geordnet.
  - Ein Index über `name` zum schnellen Zugriff darauf
  - Zweiter Index über `weingut` zum schnellen Zugriff darauf



4961	Chardonnay	Weiß	2002	Bighorn
2168	Creek Shiraz	Rot	2003	Creek
1042	La Rose GrandCru	Rot	1998	Chateau La Rose

2171	Pinot Noir	Rot	2001	Creek
3478	Pinot Noir	Rot	1999	Helena
4711	Riesling Reserve	Weiß	1999	Müller

3456	Zinfandel	Rot	2004	Helena

Kapitel 13: Dateiorganisation und Indizes

Klassifikation von Indexstrukturen

- Primärer ⇔ Sekundärer Index
  - Primärer Index: Suchschlüssel des Index ist der Primärschlüssel der Tabelle
  - Sekundärer Index: Suchschlüssel des Index ist ein beliebiges Attribut (oder eine Attributmenge) der Tabelle
- Geclusterter ⇔ Nicht-Geclusterter Index
  - Geclusterter Index: Daten sind physisch in der Reihenfolge des Index sortiert
  - Somit kann es natürlich pro Tabelle maximal einen geclusterten Index geben
- Dünnbesetzter ⇔ Dichtbesetzter Index
  - Dichtbesetzter Index: für jeden vorkommenden Attributwert des Suchschlüssels gibt es einen Indexeintrag
  - Dünnbesetzter Index: nur für mache Attributwerte gibt es Indexeinträge

Was sollte man indizieren?

- Index macht Sinn auf
  - Primärschlüssel – Zugriff auf die natürliche Ordnung
  - Fremdschlüssel – Verbessern der Performance von Join-Operationen
  - Spalten die häufig in WHERE Klauseln vorkommen

- Beispiel

```
SELECT    WeinId, Name, Weine.Weingut, Anbaugebiet
   FROM   Weine, Erzeuger
   WHERE  Weine.Weingut = Erzeuger.Weingut AND
          Name = 'Pinot Noir'
```

Wichtige Indexstrukturen

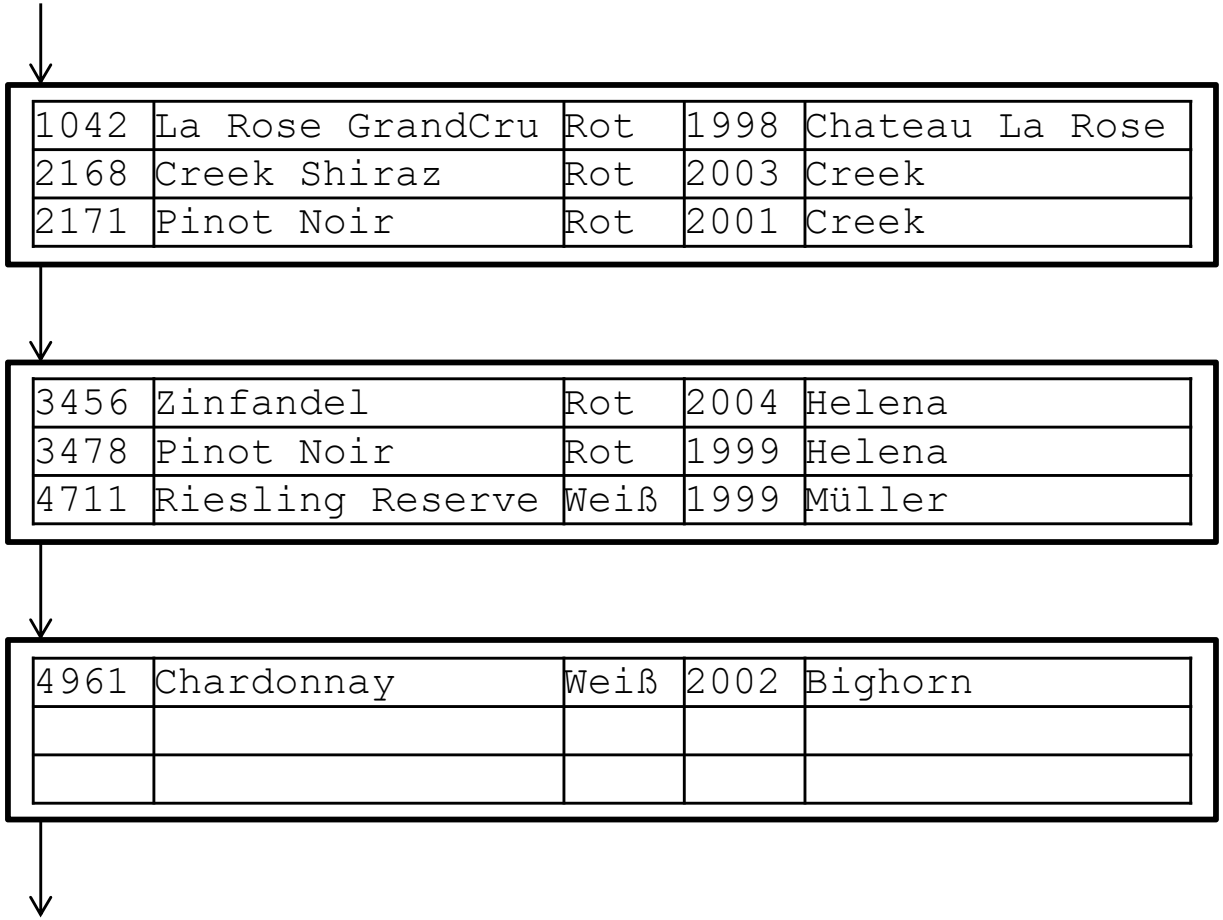
- Sequenzielle Speicherung (eigentlich kein wirklicher Index)
- Index-Sequenzielle Speicherung
- Hash-Index
- Bitmap-Index
- B+-Baum (und Verwandte)

Sequenzielle Speicherung

- Sortiertes Speichern der Datensätze

Nur eine leichte Verbesserung gegenüber einer unsortierten Speicherung.

Bei den typischen Operationen (CRUD) kann es leicht zu hohen Aufwänden bei der Erhaltung der Sortierung kommen.



1042	La Rose GrandCru	Rot	1998	Chateau La Rose
2168	Creek Shiraz	Rot	2003	Creek
2171	Pinot Noir	Rot	2001	Creek

3456	Zinfandel	Rot	2004	Helena
3478	Pinot Noir	Rot	1999	Helena
4711	Riesling Reserve	Weiß	1999	Müller

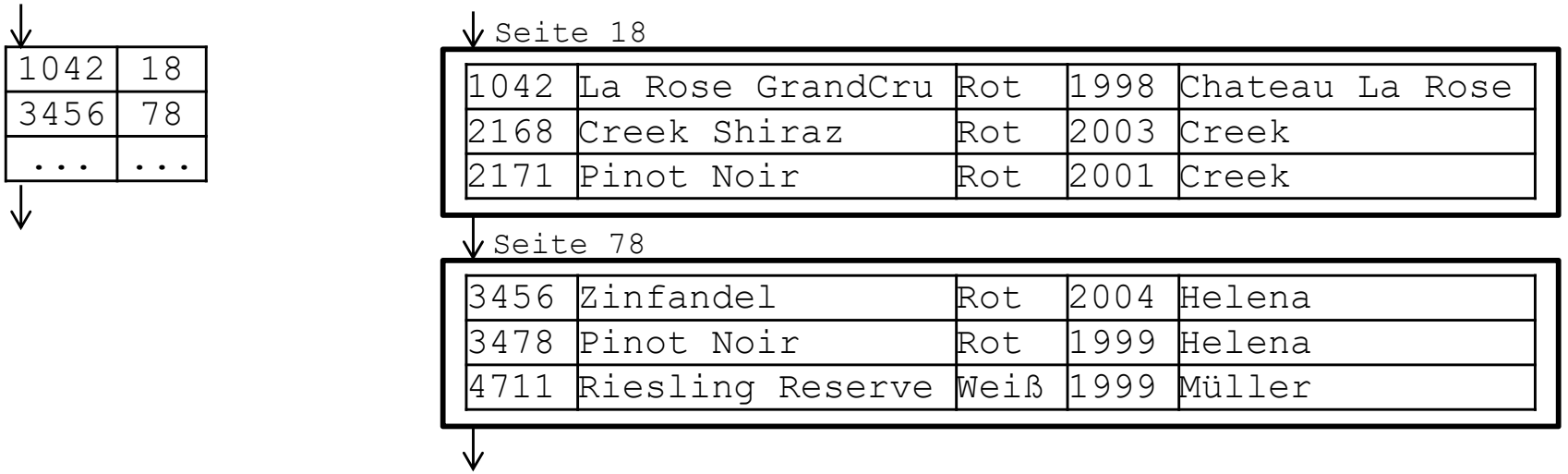
4961	Chardonnay	Weiß	2002	Bighorn

Sequenzielle Speicherung - Operationen

- **Insert**: Seite Suchen, Datensatz einsortieren
  - → Beim Anlegen oder sequentiellen Füllen jede Seite nur bis zu einem gewissen Grad füllen (ca. 66%)
  - → Falls kein Platz: Datensatz mit größtem Wert auf nächste Seite schieben oder neue Seite anlegen und Datensätze 50/50 verteilen
- **Delete**: Seite Suchen, Datensatz als gelöscht markieren (Löschbit)
- **Lookup**: Sequentielles Lesen der Datei bis Datensatz gefunden
  - O(n) wobei n die Anzahl Datensätze ist

Index-Sequenzielle Speicherung

- Index-sequenzielle Dateiorganisationsform (*ISAM, Index-Sequential Access Method*): Kombination von sequenzieller Hauptdatei und Indexdatei:
- Indexdatei kann geclusterter, dünnbesetzter Index sein
- Entspricht „zweistufigem Baum“
  - Blattebene ist Hauptdatei (Datensätze)
  - Andere Stufe („Wurzel“) ist Indexdatei

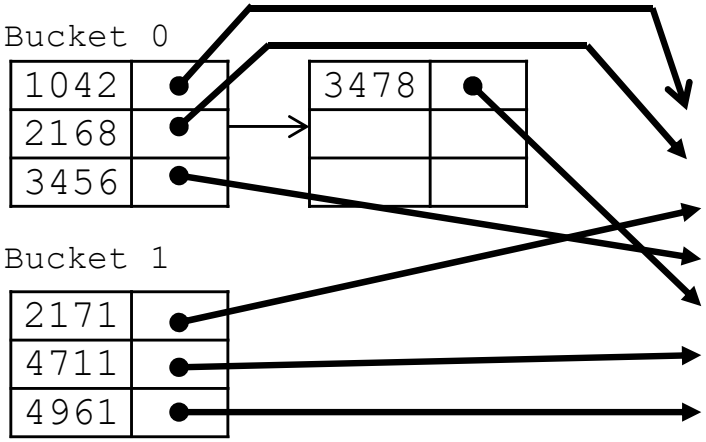


Index-Sequenzielle Speicherung - Operationen

- **Insert & Delete**: Vorgehen wie bei sequenzieller Speicherung, aber
  - Suchen (Lookup) schneller
  - Erhöhter Aufwand, den Index zu verwalten
  - Führt ein Insert zu einer neuen Seite, muss der ganze Index verschoben werden.
- **Lookup**: Binäres Suchen im Index, dann direkter Zugriff auf die richtige Datenseite
  - Somit deutlich schneller als sequenzielle Speicherung: O(log n)

Hash-Index

- Eine Hash-Funktion bildet jeden Schlüsselwert auf eine ganze Zahl ab: $h(key)$: key → [0, …, n]
- Eine Hash-Tabelle besteht aus n **Buckets**. In Bucket j werden die Datensätze mit $h(key)=j$ gespeichert, d.h. die Datensätze, deren key von der Hash-Funktion auf j abgebildet wird.
- Ein Hash-Index ist eine Hash-Tabelle in der als Daten die Datensätze selber (primärere Hash-Index) bzw. Zeiger auf die Datensätze (sekundärer Hash-Index) abgelegt werden.



1042	La Rose GrandCru	Rot	1998	Chateau La Rose
2168	Creek Shiraz	Rot	2003	Creek
2171	Pinot Noir	Rot	2001	Creek
3456	Zinfandel	Rot	2004	Helena
3478	Pinot Noir	Rot	1999	Helena
4711	Riesling Reserve	Weiß	1999	Müller
4961	Chardonnay	Weiß	2002	Bighorn

`h(key)=(key%2)`

Hash-Index - Operationen

- **Insert**: Datensatz ans das Ende der Datei anhängen, und Hash-Index updaten in O(1) möglich
- **Delete**: Analog Insert
- **Lookup**: Unter der Annahme einer guten Hash-Funktion (in konstanter Zeit berechenbar und verteilt die Datensätze gleichmäßig) ist der Lookup in O(1) möglich.
  - In dem Beispiel mit *modulo 2* als Hash-Funktion werden die Buckets natürlich riesig und das Verfahren wird schlechter als ISAM.
  - Ein Hash Index hilft bei der Suche nach einem einzelnen Datensatz, bei ganzen Wertebereichen sieht es auf Grund der Verteilung eines Wertebereiches auf unterschiedliche Buckets schon wieder schlechter aus.

Kapitel 13: Dateiorganisation und Indizes

Bäume wachsen überall!

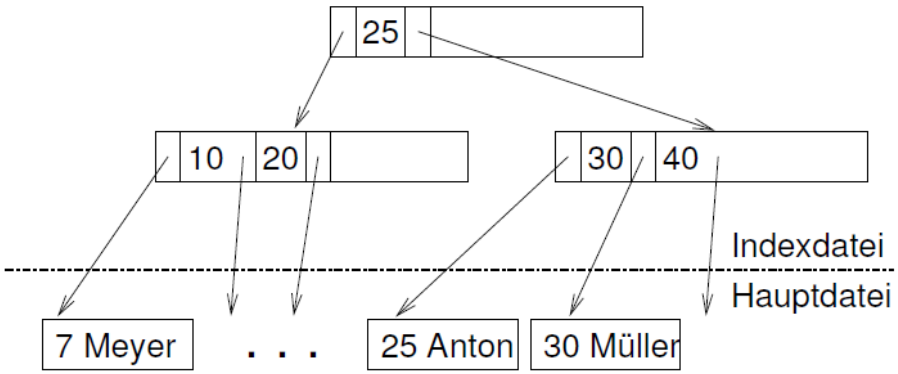
- Bäume sind eine weitverbreitete Datenstruktur
  - Nicht nur als B+-Bäume in Relationalen-DBMS, auch in OO-DBMS
  - Bäume generell als Datenstruktur extrem wichtig.
  - Grund: Insert, Delete, Update, Lookup alle in **garantiert** O(log(n)) möglich
- B-Baum und Varianten
  - B-Baum von Herrn Bayer (TUM) eingeführt. B steht für balanciert (nicht Bayer oder binär!)
  - Unmenge von Varianten wurden entwickelt: B+ Baum, B* Baum, R Baum, UB Baum, …
  - Wichtigster für DBMS: B+ Baum
    - Sehr ähnlich dem ursprünglichen B-Baum
    - Unterschied: Tupel/Tupelzeiger nur in Blättern, und Blätter untereinander verkettet.

Der B+ Baum

- Ordnung eines B+Baumes: min. Anzahl der Einträge auf den Indexseiten (außer Wurzelseite)

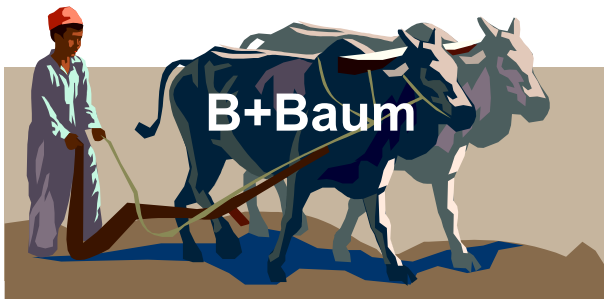
- Definition: Indexbaum ist B+Baum der Ordnung m, wenn:
  - Jede Seite enthält höchstens $2m$ Elemente
  - Jede Seite außer Wurzelseite enthält mind. m Elemente
  - Jede Seite ist entweder Blattseite ohne Nachfolger oder hat i + 1 Nachfolger (i : Anzahl ihrer Elemente)
  - Alle Blattseiten liegen auf der gleichen Stufe, nur Blätter enthalten Tupel/TIDs
  - Blätter untereinander verkettet für sequenziellen (Bereichs-) Durchlauf

- Beispiel



B+ Baum – Eigenschaften

- B+ Baum als Primär/Sekundärindex einsetzbar
  - Als Primärindex können Tupel direkt im Baum gespeichert werden
  - Als Sekundärindex werden nur Tupel-Ids (Zeiger) im Baum gespeichert
- Effizient: Balancierungskriterium führt zu nahezu vollständiger Ausgeglichenheit, d.h. Weg von der Wurzel zu jedem Blatt gleich lang
  - n Datensätze in Hauptdatei→ $\log_m n$ Seitenzugriffe von der Wurzel zum Blatt
  - Einfügen, Löschen, Suchen mit $O(\log_m n)$
  - Bereichsanfragen (a < x < b) durch Blattverkettung effizient möglich
- Vorteil: Baum „reorganisiert“ sich selbständig bei Insert/Delete durch kleine, lokale Änderungen
- Nachteil: zusätzlicher Aufwand bei Insert/Delete, Speicherplatzoverhead
- **Fazit**: Vorteile überwiegen bei weitem, daher ist der B+ Baum der am weitesten verbreitetste Index in **allen** kommerziellen DBMS („Arbeitstier“)



Kapitel 13: Dateiorganisation und Indizes

Indizes in MS SQL Server (1)

- SQL Server kennt nur B+ Baum Indizes, diese können *clustered* (primär) oder *nonclustered* (sekundär) sein
- Indizes können für die PRIMARY KEY oder UNIQUE Spalten direkt im CREATE TABLE Statement erzeugt werden
  - Für PRIMARY KEY wird automatisch ein (CLUSTERED) Index erstellt
- Beispiel

```
create table WEINE (
   WeinID int not null,
   Name varchar(20) not null,
   Farbe varchar(10),
   Jahrgang int,
   Weingut varchar(20),
   primary key nonclustered (WeinID),
   unique clustered (Name, Weingut),
   foreign key(Weingut) references ERZEUGER(Weingut))
```

Indizes in MS SQL Server (2)

- CLUSTERED und NONCLUSTERED Indizes können mittels CREATE INDEX auf beliebigen Spalten(-kombinationen) erzeugt werden

```
CREATE … [ CLUSTERED | NONCLUSTERED ] INDEX <index_name>
   ON <table or view> ( <column> [ ASC | DESC ] [ ,...n ] )
   …
```

- Beispiel

```
create nonclustered index idx_weine_weingut
   on WEINE (Weingut asc)
```

Zusammenfassung

- DBMS speichern Daten auf Sekundärspeichern zur langfristigen, nicht-flüchtigen Speicherung
- Verwaltung der Sekundärspeicher erfolgt in Seiten
- Organisation kann vom Nutzer beeinflusst werden
- Indexstrukturen können die Performance erheblich steigern
- Viele verschiedene Indextypen sind verfügbar
- B+ Baum ist am weitesten verbreitet, sehr gut für relationale DBMS geeignet und daher in jedem kommerziellen System verfügbar