9 Datenbanken

9 Lernziele

- Sie kennen die Komponenten eines DBMS und wissen wie diese zusammenspielen.
- Sie verstehen, wie ein DBMS Benutzeranfragen optimiert.
- Sie können erklären, wie eine effiziente Datenspeicherung erfolgt und wie Indizes funktionieren.
- Sie kennen die Kriterien, um ein DBMS auszuwählen.

9 Aufgaben eines DBMS

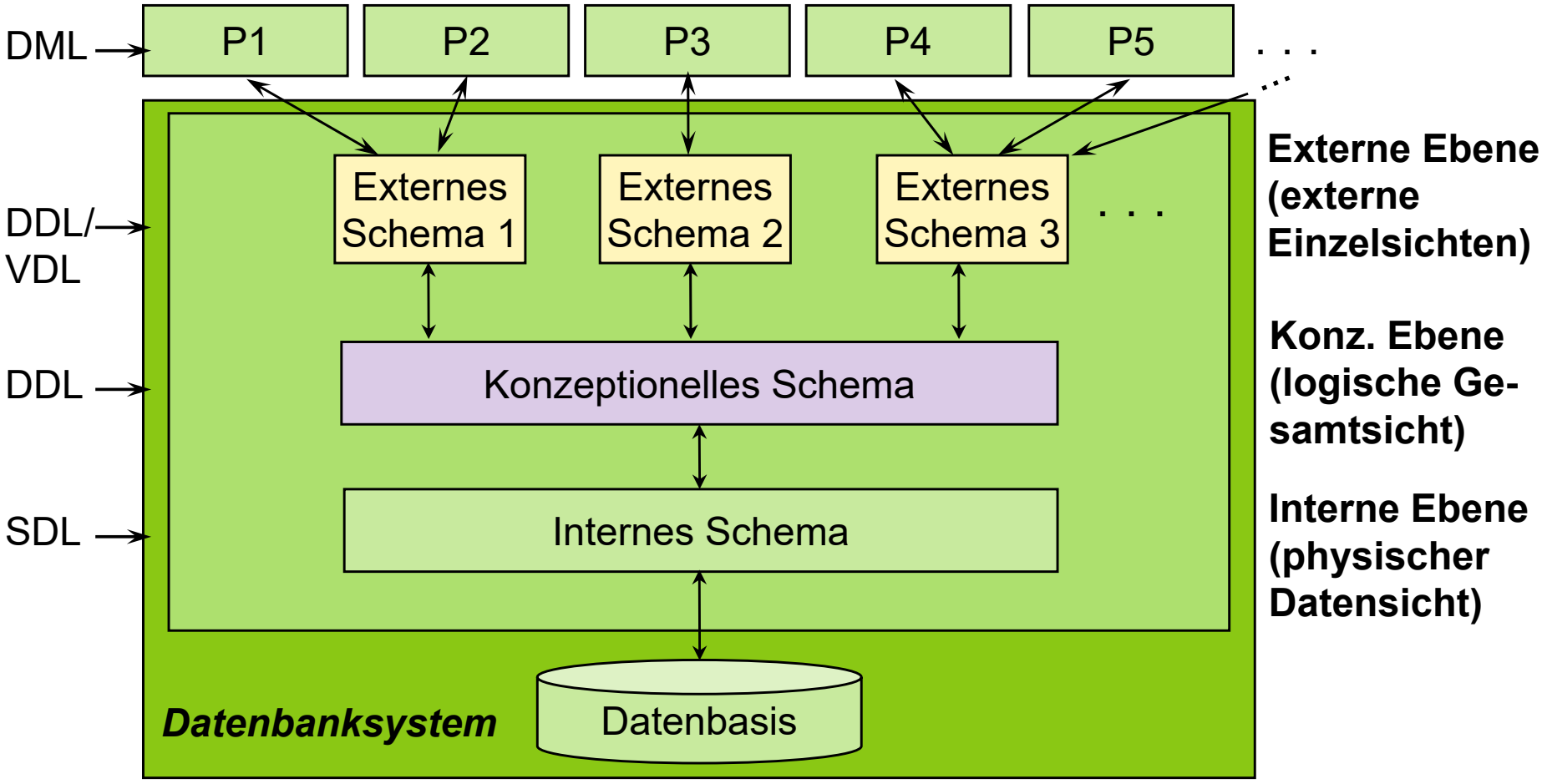
- Gewährleistung von Datenintegrität
  - Semantische Datenbankintegrität
    z.B. referentielle Integrität und weitere Constraints
  - Operationale Datenbankintegrität
    Verhinderung von Zerstörung von Daten bei Mehrbenutzerbetrieb
  - Physische Datenbankintegrität
    Wiederherstellung eines konsistenten Datenbestands nach Anwendungs- oder Systemfehlern
- Schutz der Daten vor unerlaubtem Zugriff
  - Rechtemanagement (Vergabe/Entzug von Zugriffsrechten)
  - Authentifikation (Identifikation und Passwort)
- Verschlüsselungsverfahren
  z.B. bei verteilten Datenbanken

9 Architekturanforderungen

- Jede Architekturkomponente behandelt ein logisch zusammenhängendes Teilproblem.
- Möglichst kleine Abhängigkeit jeder einzelnen Komponente von anderen Komponenten zur Minimierung der Beziehungskomplexität.
- Die Funktionsweise jeder Komponente ist so definiert, dass sie von anderen Komponenten ohne Kenntnis ihrer Realisierungsdetails nutzbar ist.

9 Schichtmodell (ANSI/SPARC-Architekturmodell)

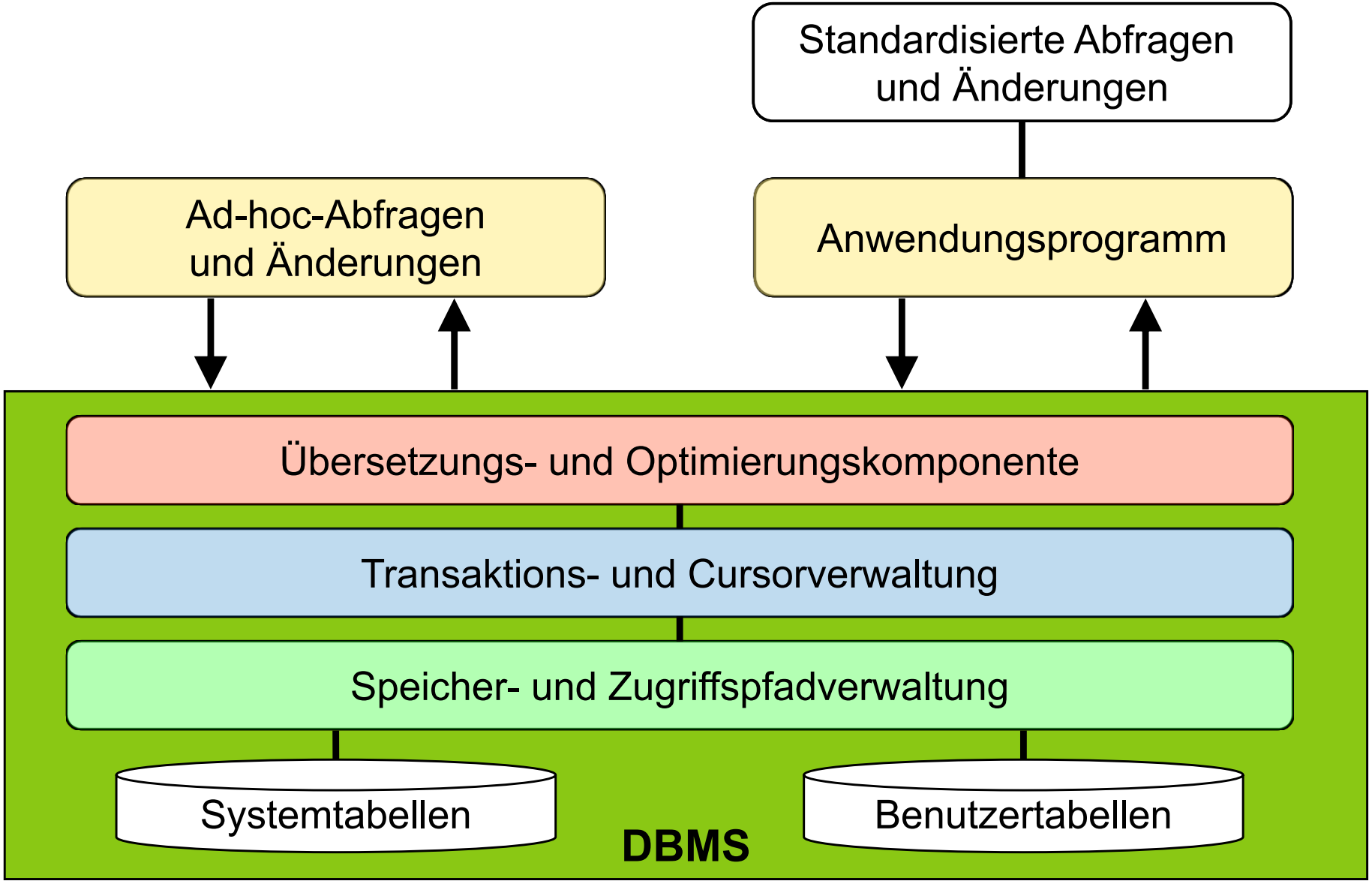
Intention: Unabhängigkeit von logischer und physischer Sicht



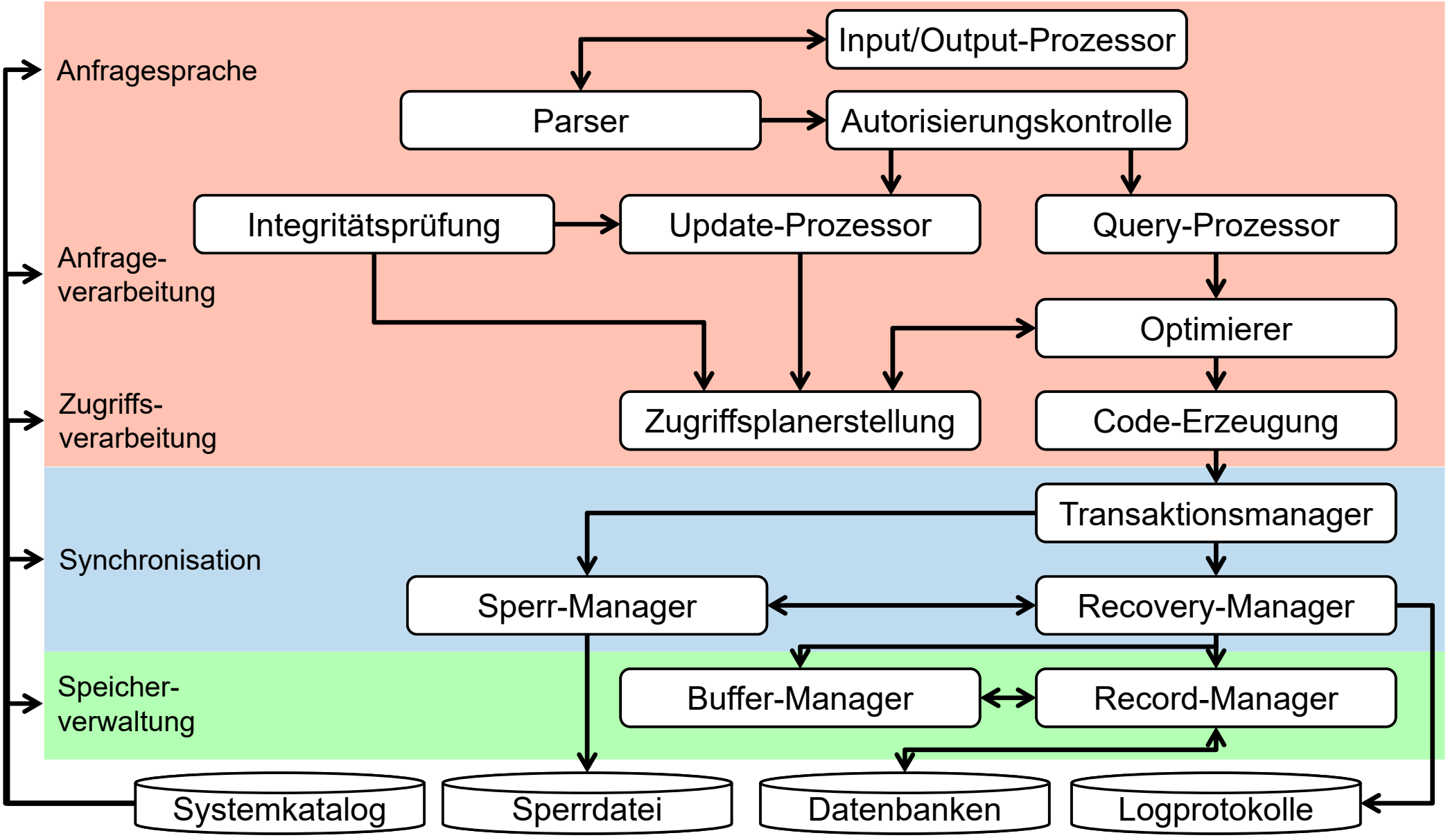
9 Ziele & Aufgaben eines DBMS

- Realisierung des Architekturmodells (ANSI/SPARC)
- Bereitstellung administrativer Werkzeuge
  - Backup
  - Monitoring
  - (Query-)Builder
- Sicherstellung eines effizienten Mehrbenutzer-Betriebes
  - **User Interface Handler:** Schnittstelle zu Anwendungen/Usern, Autorisationskontrolle
  - **Datenkontrolle:** Syntaktisch und Semantisch
  - **Anfrage-Optimizer:** Optimiert Anfragen anhand Metadaten, ermöglicht schnelle Auswertung verknüpfter Daten
  - **Transaktionsmanager:** Mehrbenutzerbetrieb durch Scheduler
  - **Anfragebearbeitung:** Anfrage auf die Datenbank
  - **Recovery-Manager:** Protokollierung und Buffer-Management zur Sicherstellung der Datensicherheit und Datenintegrität
  - **Runtime Support Processor:** Zugriff auf Speichermedium

9 DBMS – Architektur im Überblick



9 DBMS – Architektur im Detail



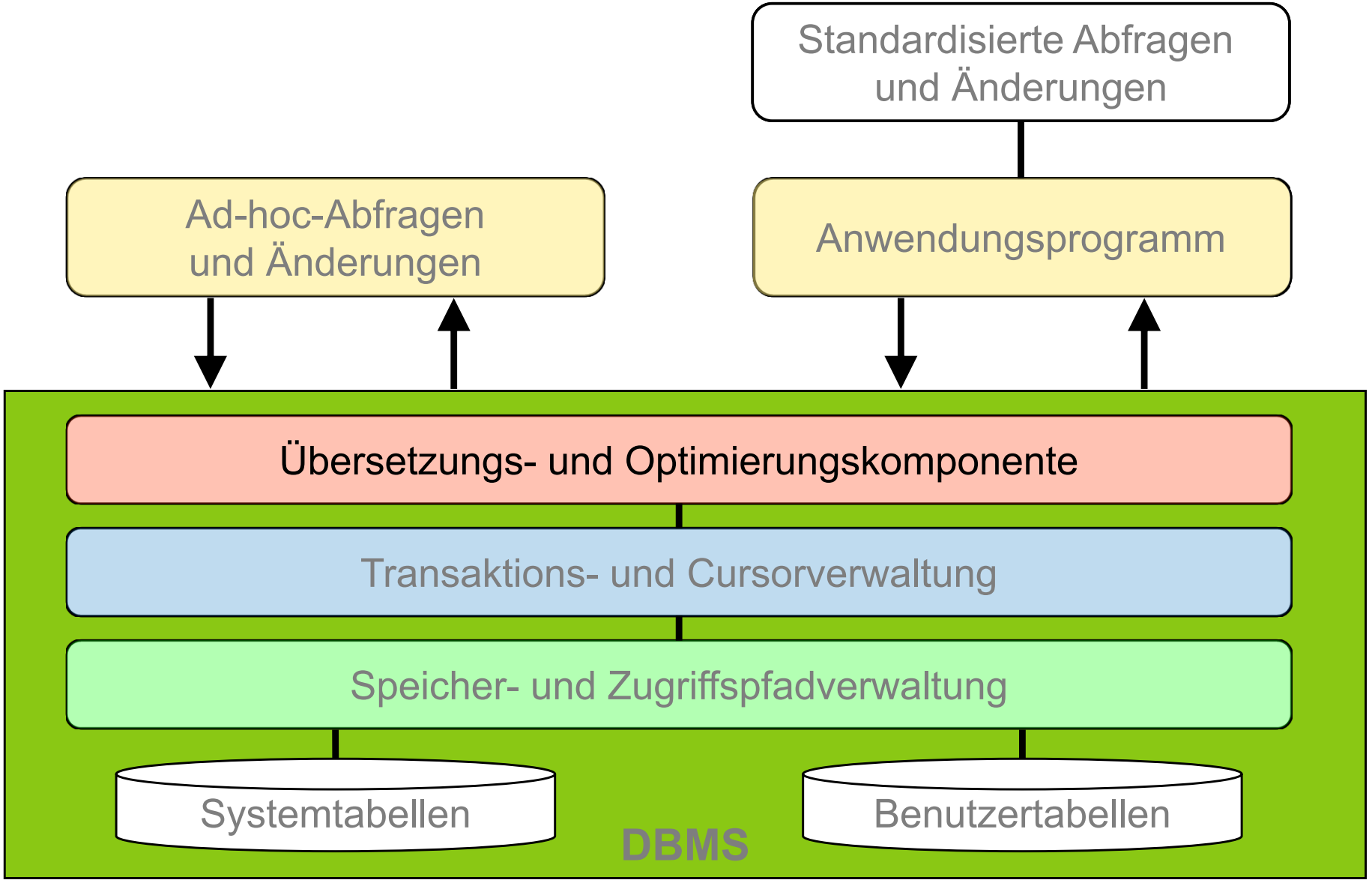
9 Ergänzende Komponenten

- CASE-Werkzeuge (Computer-Aided Software Engineering)
  - Integrierte Systeme zur produktiven, qualitätsgesicherten Entwicklung von Datenbankanwendungen,
    z.B. Freie: MySQL Workbench
    Kommerzielle: Aris, Rational Rose, SAP R/3
  - Funktionen sind z.B.
    - Generierung von Datenbankschemata aus konzeptionellem Entwurf
    - Generierung von Anwendungen (Listen, Masken, Menüs usw.)
    - Alle mit dem Entwicklungsprozess verbunden Informationen werden ein einer Entwicklungsdatenbank gespeichert.
- Browser-Oberfläche
  - zur benutzerfreundlichen Abfrage von Daten, d.h. ohne SQL-Kenntnisse, z.B. phpMyAdmin, …

9 Ergänzende Komponenten

- Administrations-Werkzeuge
  - zur Vereinfachung der Benutzerverwaltung, Festplattverwaltung, Datensicherung oder Datenimport und -export,
    z.B. mysqldump, …
- Monitore
  - zur Aufzeichnung und Analyse des Laufzeitverhaltens einer Datenbank um „Performance“-Probleme rechtzeitig zu erkennen. Hierzu zählt auch Datenbank-Tuning, d.h. Anpassung des konzeptionellen und des internen Schemas an das Anwendungsprofil,
    z.B. Nagios, HP Open View, …. Open-Source: Zabbix, OP5, …

9 Übersetzungs- und Optimierungskomponente



9 Konzeptionelle Befehlsabarbeitung (SELECT)

1. Bildung des kartesischen Produktes der Tabellen der FROM-Klausel
2. Auswertung der Selektionsbedingung in der WHERE-Klausel
3. Projektion auf die in der SELECT-Klausel angegebenen Spaltennamen

Problem mit dieser Vorgehensweise

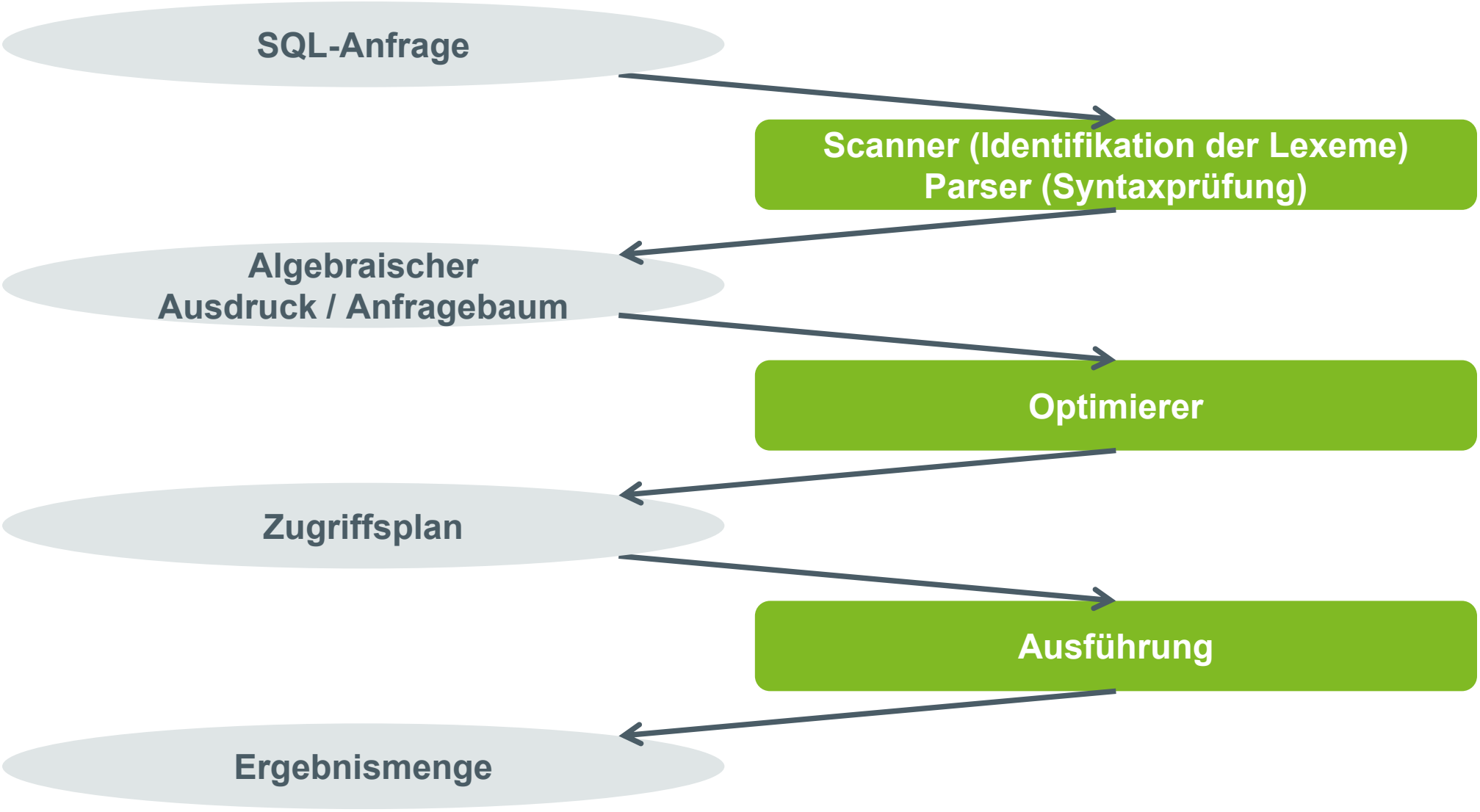
⇒ Kombinatorik

⇒ Viele Speicherzugriffe

⇒ Zugriffslücke in der Speicherhierarchie

Ziel: Optimierung der Laufzeiteffizienz (sog. Performance“) ohne Benutzerintervention

9 Konzeptionelle Befehlsabarbeitung (SELECT)



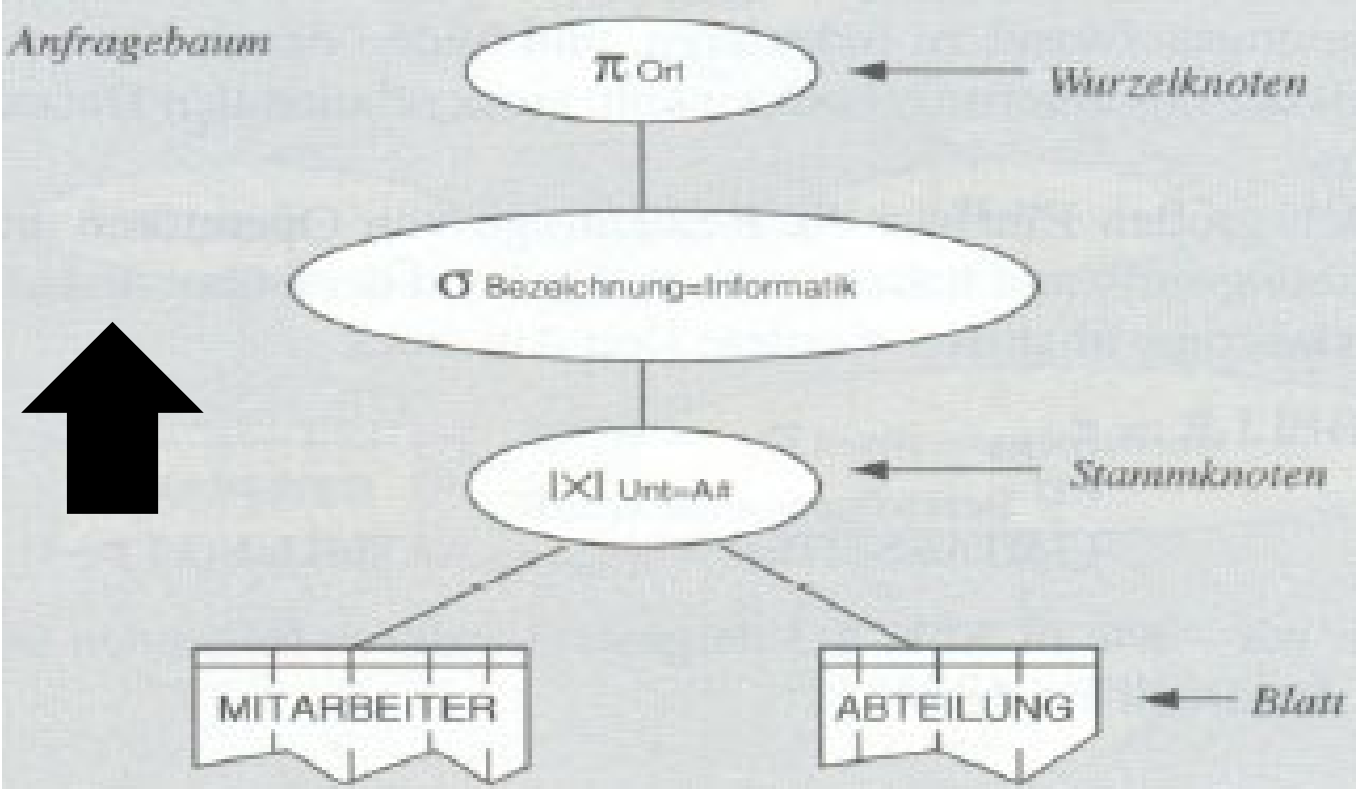
9 Bsp.: SQL-Abfrage u. Anfragebaum

Ein Anfragebaum (engl. *query tree*) visualisiert graphisch eine relationale Abfrage durch den *äquivalenten Ausdruck der Relationenalgebra*. Z.B.

```
SELECT Ort FROM Mitarbeiter, Abteilung WHERE …
```

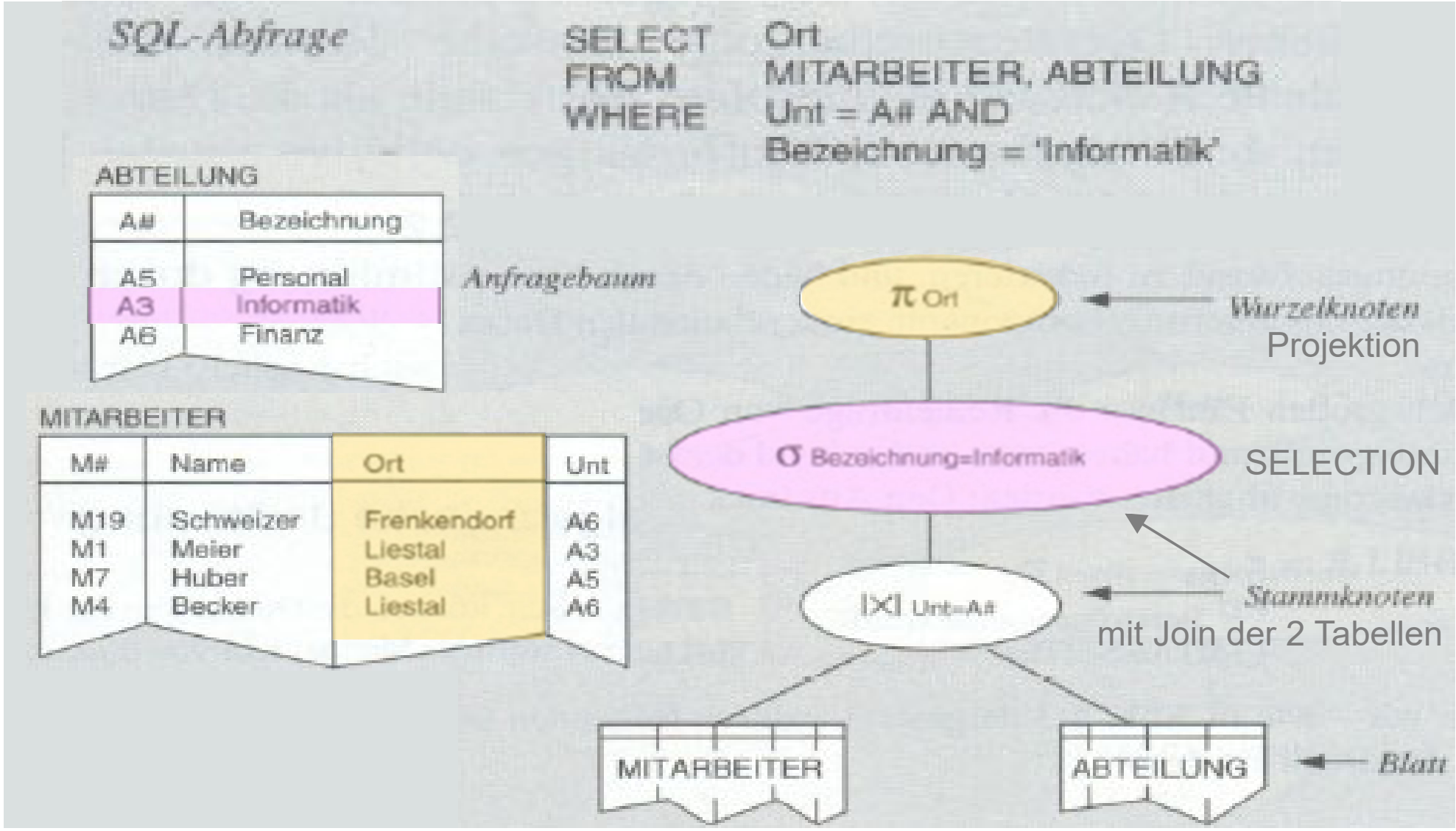
Die Blätter des Anfragebaumes entsprechen den für die Abfrage verwendeten Tabellen, der Wurzel- und die inneren Knoten bezeichnen die algebraischen Operatoren.

Die Berechnung Erfolgt von unten nach oben!



Quelle: Meier, A.: Relationale und postrelationale Datenbanken, 2007, S. 97

9 Bsp.: SQL-Abfrage u. Anfragebaum



Quelle: Meier, A.: Relationale und postrelationale Datenbanken, 2007, S. 97

9 Motivierendes Beispiel

```
select KUNDE.KNr, Nachname
from KUNDE, BESTELLUNG
where KUNDE.KNr = BESTELLUNG.KNr and Datum = date '22-NOV-04'
```

- In der Relation Kunde sind 100 Tupel gespeichert. Auf eine (Computer-) Seite/einen Block von der Speicherverwaltung passen hierbei 5 Tupel.
- In der Relation Bestellung sind 10.000 Tupel gespeichert. Auf eine Seite passen jeweils 10 Tupel.

Quelle: Saake, G.; Sattler, K.-U., Heuer, A.: Datenbanken - Implementierungstechniken, 2011, S. 338

9 Motivierendes Beispiel

```
select KUNDE.KNr, Nachname
from KUNDE, BESTELLUNG
where KUNDE.KNr = BESTELLUNG.KNr and Datum = date '22-NOV-04'
```

- In der Relation Kunde sind 100 Tupel gespeichert. Auf eine (Computer-) Seite/einen Block von der Speicherverwaltung passen hierbei 5 Tupel.
- In der Relation Bestellung sind 10.000 Tupel gespeichert. Auf eine Seite passen jeweils 10 Tupel.
- Typischerweise werden 50 Bestellungen pro Tag aufgegeben.
- Betrachten wir Tupel der Form (KNr, Nachname, so passen 50 von ihnen auf eine Seite. Das Ergebnis der Abfrage passt also auf eine Seite.

Quelle: Saake, G.; Sattler, K.-U., Heuer, A.: Datenbanken - Implementierungstechniken, 2011, S. 338

9 Motivierendes Beispiel

o.B.d.A. (ohne Beschränkung der Allgemeinheit):

- Vereinfachend nehmen wir an, dass für jeden Ausführungsschritt eine Zwischenrelation mit dem Ergebnis angelegt wird und dass der Puffer für jede Relation die Größe 1 hat.
- Auf Seiten werden nur ganze Tupel einer Relation abgespeichert.

Quelle: Saake, G.; Sattler, K.-U., Heuer, A.: Datenbanken - Implementierungstechniken, 2011, S. 338

Direkte Auswertung

- 3 Zeilen des Kreuzprodukts $r(\texttt{KUNDE}) \times r(\texttt{BESTELLUNG})$ passen auf eine Seite.

Die folgenden Schritte beschreiben die Berechnungsabfolge. Bei den einzelnen Schritten werden jeweils die berechneten Kosten für lesende (l) und schreibende (s) Seitenzugriffe aufgeführt.

1. $r_1 := r(\texttt{KUNDE}) \times r(\texttt{BESTELLUNG})$

   Die Berechnung des Kreuzprodukts und anschließendes Schreiben des Zwischenergebnisses benötigt die folgenden Seitenzugriffe:

   - $l : (100/5 \cdot 10.000/10) = 20.000$

     Die Tupel beider Relationen werden seitenweise gelesen. Aufgrund der Puffergröße muss für jede Seite der KUNDE-Relation jeweils jede Seite der BESTELLUNG-Relation erneut gelesen werden.

ℓ=lesen

Quelle: Saake, G.; Sattler, K.-U., Heuer, A.: Datenbanken - Implementierungstechniken, 2011, S. 339

Direkte Auswertung

- 3 Zeilen des Kreuzprodukts $r(\texttt{KUNDE}) \times r(\texttt{BESTELLUNG})$ passen auf eine Seite.

Die folgenden Schritte beschreiben die Berechnungsabfolge. Bei den einzelnen Schritten werden jeweils die berechneten Kosten für lesende (l) und schreibende (s) Seitenzugriffe aufgeführt.

1. $r_1 := r(\texttt{KUNDE}) \times r(\texttt{BESTELLUNG})$

   Die Berechnung des Kreuzprodukts und anschließendes Schreiben des Zwischenergebnisses benötigt die folgenden Seitenzugriffe:

   - $l : (100/5 \cdot 10.000/10) = 20.000$
     Die Tupel beider Relationen werden seitenweise gelesen. Aufgrund der Puffergröße muss für jede Seite der KUNDE-Relation jeweils jede Seite der BESTELLUNG-Relation erneut gelesen werden.
   - $s : (100 \cdot 10.000)/3 = 333.000$ (ca.)
     Alle Seiten des Zwischenergebnisses (Kreuzprodukt!) werden gespeichert.

l=lesen
s=schreiben

Quelle: Saake, G.; Sattler, K.-U., Heuer, A.: Datenbanken - Implementierungstechniken, 2011, S. 339

2. $r_2 := \sigma_{\texttt{KUNDE.KNr=BESTELLUNG.KNr} \wedge \texttt{Datum='22.11.04'}}(r_1)$

   Bei der Selektion muss erst das Zwischenergebnis des vorigen Schrittes gelesen werden. Geschrieben werden alle Tupel, die das Prädikat erfüllen, also insgesamt 50, die auf 17 Seiten passen:

   - l : 333.000 (ca.)
   - s : $50/3 = 17$ (ca.)

3. $r_{erg} := \pi_{\texttt{KNr,Nachname}}(r_2)$

   Bei der abschließenden Projektion müssen alle 17 Zwischenergebnisseiten gelesen und die eine Seite mit dem Endergebnis zurückgeschrieben werden:

   - l : 17
   - s : 1

Insgesamt werden in dieser Variante ca. 687.000 Seitenzugriffe benötigt. Außerdem werden ca. 333.000 Seiten zur Zwischenspeicherung benötigt.

Quelle: Saake, G.; Sattler, K.-U., Heuer, A.: Datenbanken - Implementierungstechniken, 2011, S. 339

9 Optimizer

- Optimizer können kostenbasiert arbeiten:
  - Anfragen werden geparst und analysiert
  - Es gibt Statistiken die für jede Tabelle die Anzahl der Sätze, die Verteilung der Schlüssel etc. angeben. Die Zugriffspläne werden unter Ausnutzung der Statistiken erstellt.

- Optimizer können regelbasiert arbeiten:
  - Anfragen werden geparst und analysiert
  - Es gibt einen Satz von Regeln, der die Umsetzung in einen Zugriffsplan steuert
  - Die Befüllung der Tabellen spielt keine Rolle.

Trotzdem macht es oft Sinn, dass man sich selbst überlegt wie SQL-Queries effizienter gestellt werden können.

Noch ist menschliche Intelligenz der künstlichen Intelligenz überlegen!!

Kostenbasierte Optimierung

Statistiken über die Daten der Tabellen sollten von Zeit zu Zeit aktualisiert werden.

Dies geschieht mit Hilfe des Kommandos:

```
Analyze Table <Tabelle>
```

Das Ergebnis kann mit SHOW INDEX angezeigt werden.

```
Show Index from <Tabelle>
```

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality
projekt_mitarbeit	0	PRIMARY	1	Personal_nr	A	7
projekt_mitarbeit	0	PRIMARY	2	projekt_nr	A	7
projekt_mitarbeit	1	FK_projekt_mitarbeit_2	1	projekt_nr	A	7
projekt_mitarbeit	1	FK_projekt_mitarbeit_1	1	Personal_nr	A	7

9 Regelbasierte Optimierung

- **Mehrere Selektionen** auf ein und dieselbe Tabelle zu einer einzigen verschmelzen, so dass das Selektionsprädikat nur einmal zu prüfen ist.
- **Selektionen** sind so früh wie möglich auszuführen, damit die Zwischenresultattabellen klein bleiben (d.h. Selektionsoperatoren sind in der Nähe der Blätter des Anfragebaums angesiedelt).
- **Projektionen** sind so früh wie möglich auszuführen, jedoch nie vor Selektionen. Sie reduzieren die Anzahl der Spalten und meistens auch die Anzahl der Tupel.
- **Verbundoperatoren** sind möglichst im Wurzelbereich des Anfragebaums zu berechnen, da sie kostenaufwändig sind.

9 Bsp.: SQL-Abfrage u. Anfragebaum

Vorher:

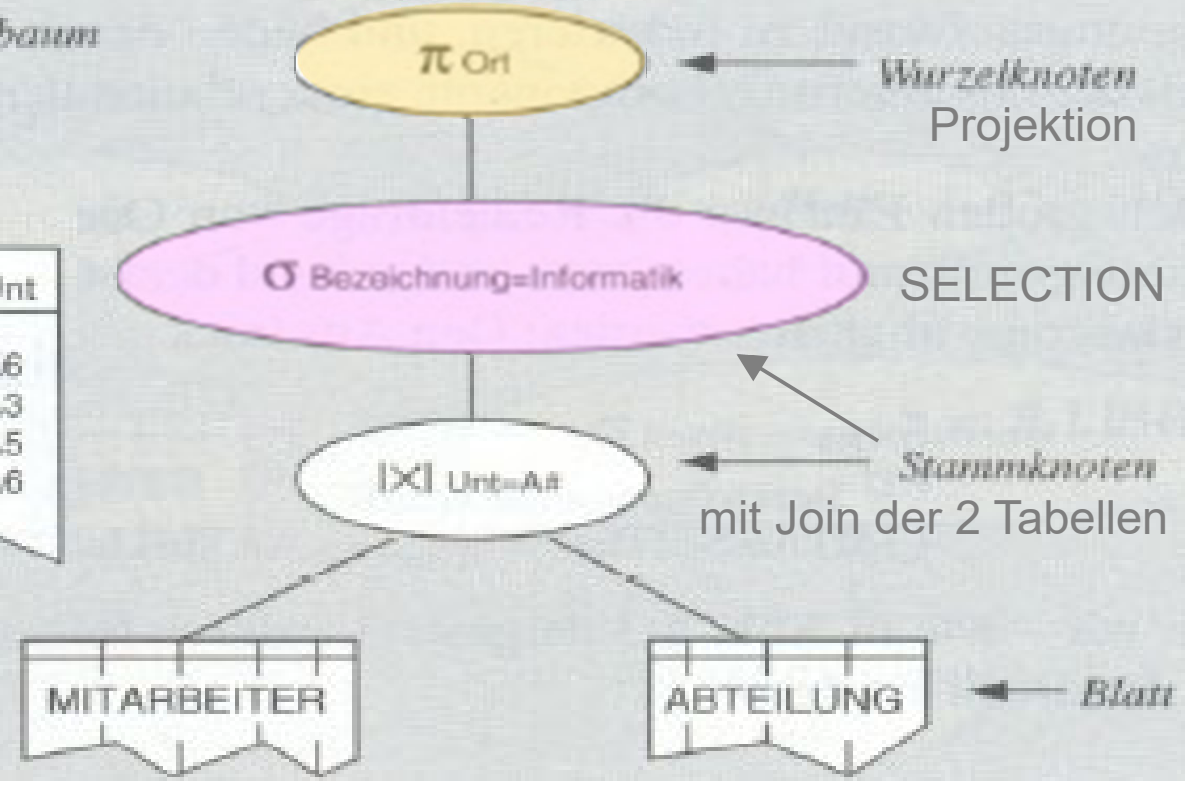
```
SELECT   Ort
FROM     MITARBEITER, ABTEILUNG
WHERE    Unt = A# AND
         Bezeichnung = 'Informatik'
```

ABTEILUNG

A#	Bezeichnung
A5	Personal
A3	Informatik
A6	Finanz

MITARBEITER

M#	Name	Ort	Unt
M19	Schweizer	Frenkendorf	A6
M1	Meier	Liestal	A3
M7	Huber	Basel	A5
M4	Becker	Liestal	A6

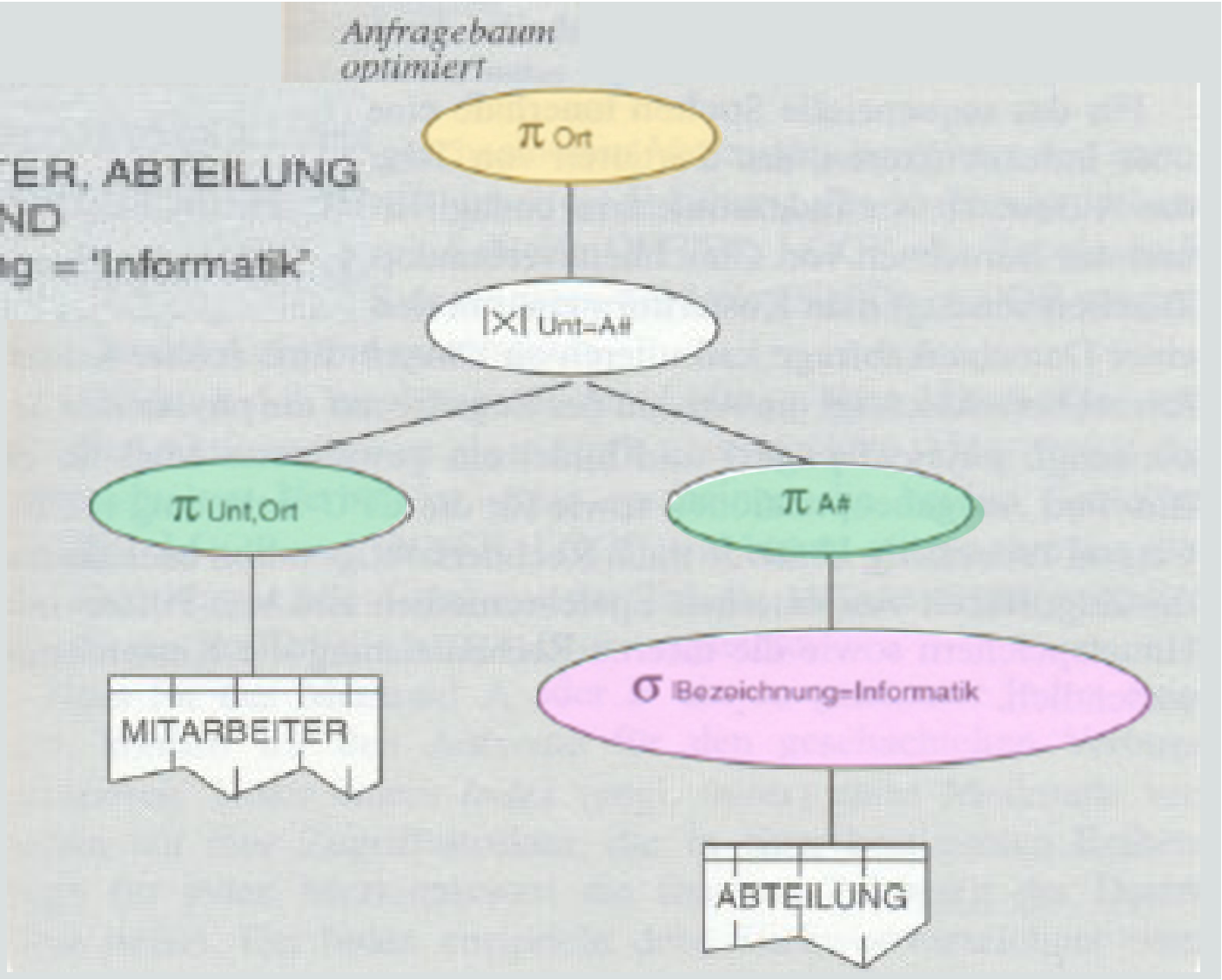


Quelle: Meier, A.: Relationale und postrelationale Datenbanken, 2007, S. 97

9 Bsp.: Regelbasierte Optimierung

Nachher:

```
SELECT   Ort
FROM     MITARBEITER, ABTEILUNG
WHERE    Unt = A# AND
         Bezeichnung = 'Informatik'
```



Quelle: Meier, A.: Relationale und postrelationale Datenbanken, 2007, S. 99

Die enorme Anzahl an zwischenzuspeichernden Tupeln resultiert aus der sinnlosen Berechnung des vollen Kreuzprodukts beider Relationen. Wird statt des Kreuzprodukts ein natürlicher Verbund berechnet, so dürfte das Zwischenergebnis viel kleiner ausfallen. Noch weiter kann man das Zwischenergebnis verkleinern, wenn der Verbund nicht mit der gesamten Relation BESTELLUNG, sondern nur mit den Tupeln, die die Selektionsbedingung Datum = '22.11.04' erfüllen, durchgeführt wird:

1. $r_1 := \sigma_{\texttt{Datum='22.11.04'}}(r(\texttt{BESTELLUNG}))$

   Hier erfolgt eine Vorselektion, um den Verbund mit kleineren Relationen durchführen zu können – alle Seiten werden gelesen, und die 50 Bestelleinträge des Tages geschrieben.

   - $l : 10.000/10 = 1.000$
   - $s : 50/10 = 5$

Quelle: Saake, G.; Sattler, K.-U., Heuer, A.: Datenbanken - Implementierungstechniken, 2011, S. 340

2. $r_2 := r(\texttt{KUNDE}) \bowtie_{\texttt{KNr=KNr}} r_1$

   Die Berechnung des Verbundes erfolgt der Vereinfachung halber mit dem Nested-Loops-Algorithmus:

   - $l : 100/5 \cdot 5 = 100$
   - $s : 50/3 = 17$

3. $r_{erg} := \pi_{\texttt{KNr,Nachname}}(r_2)$

   Abschließend wird die Ergebnisprojektion berechnet:

   - $l : 17$
   - $s : 1$

Insgesamt werden in dieser Variante ca. 1.140 Seitenzugriffe benötigt.

Quelle: Saake, G.; Sattler, K.-U., Heuer, A.: Datenbanken - Implementierungstechniken, 2011, S. 340

2. $r_2 := r(\texttt{KUNDE}) \bowtie_{\texttt{KNr=KNr}} r_1$

   Die Berechnung des Verbundes erfolgt der Vereinfachung halber mit dem Nested-Loops-Algorithmus:

   - $l : 100/5 \cdot 5 = 100$
   - $s : 50/3 = 17$

3. $r_{erg} := \pi_{\texttt{KNr,Nachname}}(r_2)$

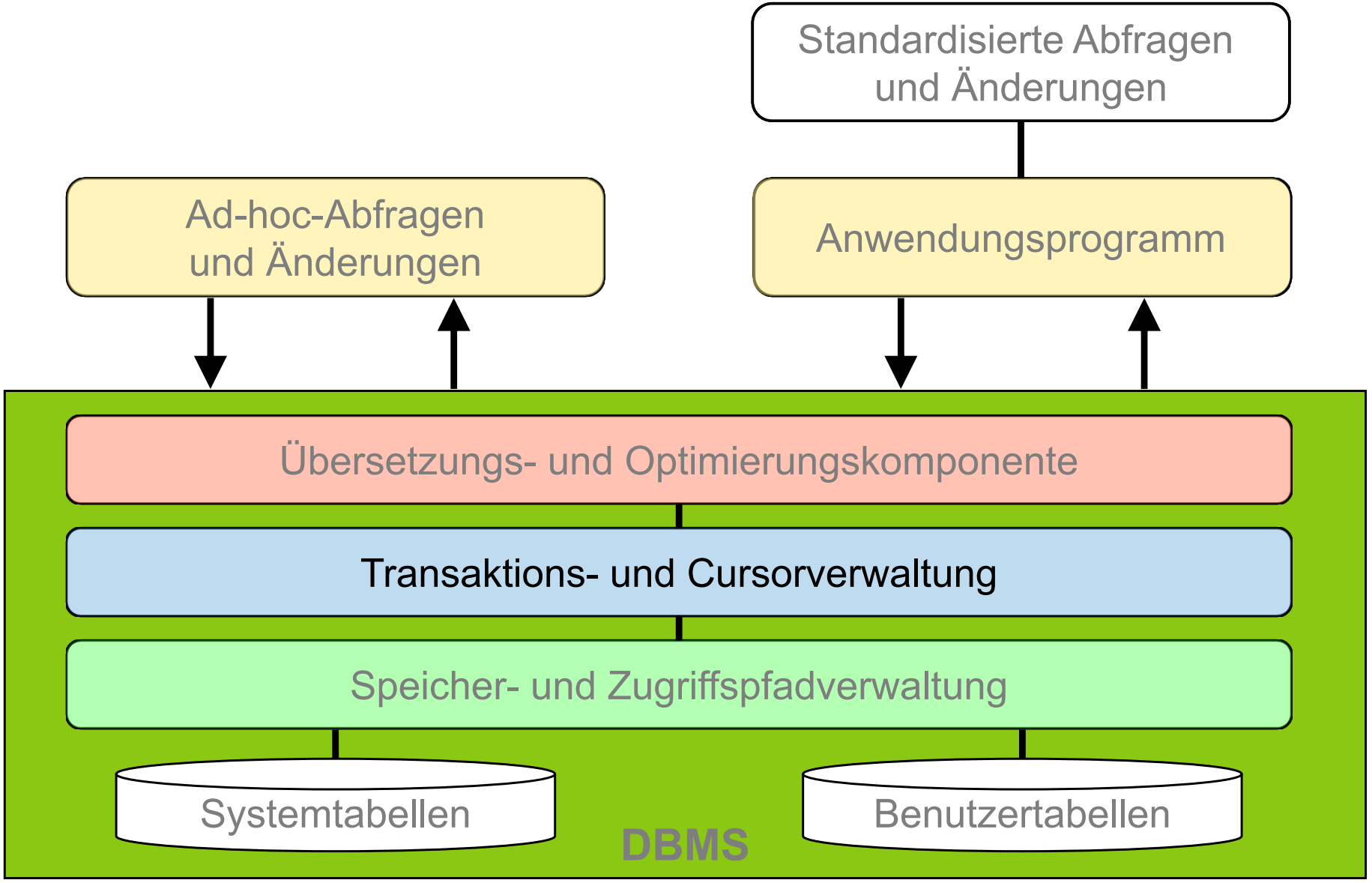
   Abschließend wird die Ergebnisprojektion berechnet:

   - $l : 17$
   - $s : 1$

Insgesamt werden in dieser Variante ca. 1.140 Seitenzugriffe benötigt. Allein durch Umformung des Relationenalgebraterms konnte somit die Ausführungszeit um einen Faktor von über 500 verbessert werden! Man sollte dabei beachten, dass es sich nicht um einen konstanten Verbesserungsfaktor handelt, sondern dass dieser Faktor bei größeren Relationen weiter wächst.

Quelle: Saake, G.; Sattler, K.-U., Heuer, A.: Datenbanken - Implementierungstechniken, 2011, S. 340

9 Übersetzungs- und Optimierungskomponente



Zur Erinnerung

Cursor (Kapitel 5, Stored Procedures)

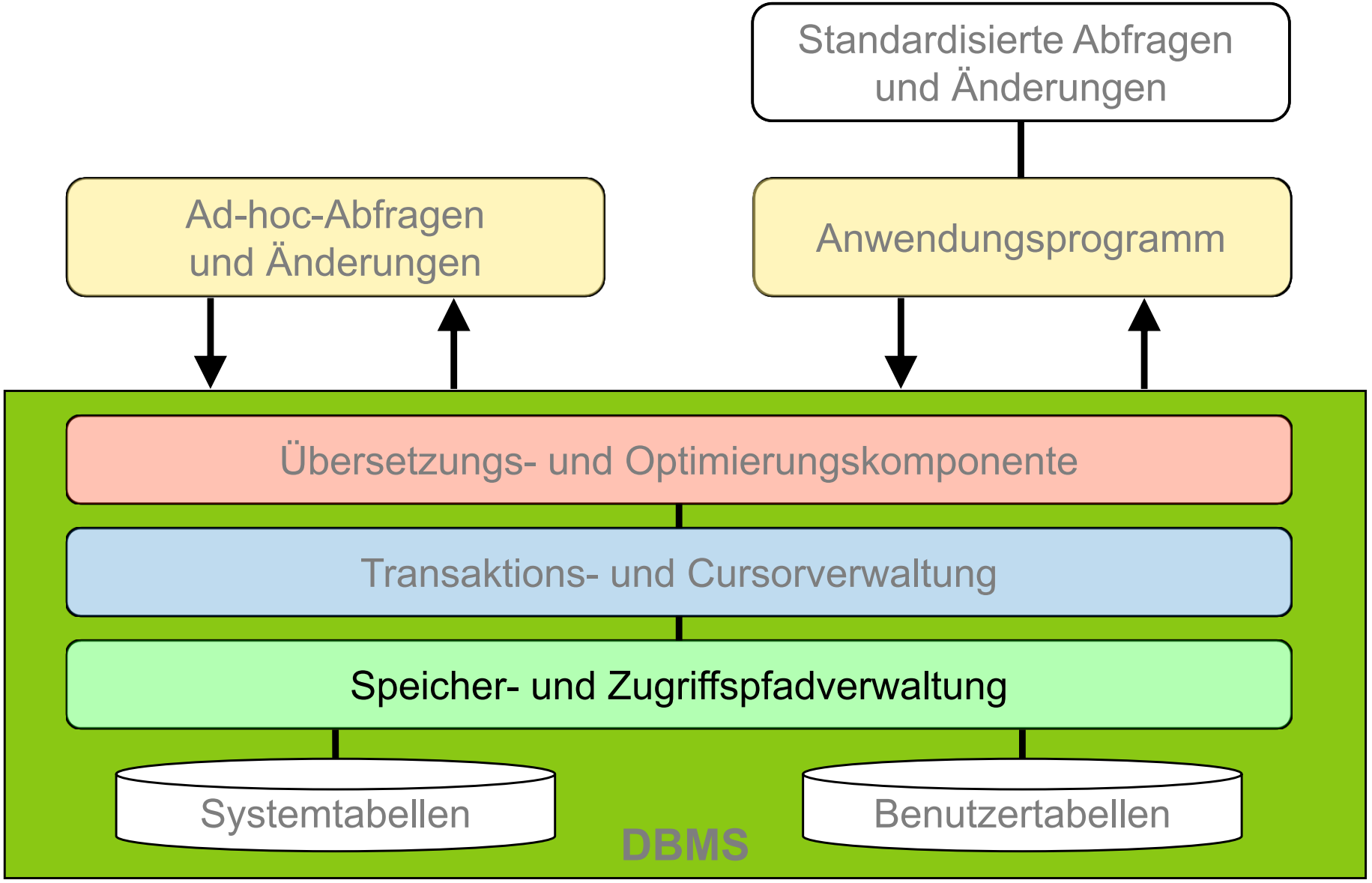
- Eine Methode, um auf die einzelnen Zeilen in einer Tabelle oder einer Ergebnismenge zugreifen zu können. Dabei wird jeweils auf eine Zeile zugegriffen.

Transaktion (Kapitel 6)

- Eine Folge von Operationen, die eine logische Einheit bilden und einen Datenbestand nach fehlerfreier Ausführung in einem konsistenten Zustand hinterlässt.

  ➔*ACID* (Atomicity, Consistency, Isolation, Durability)

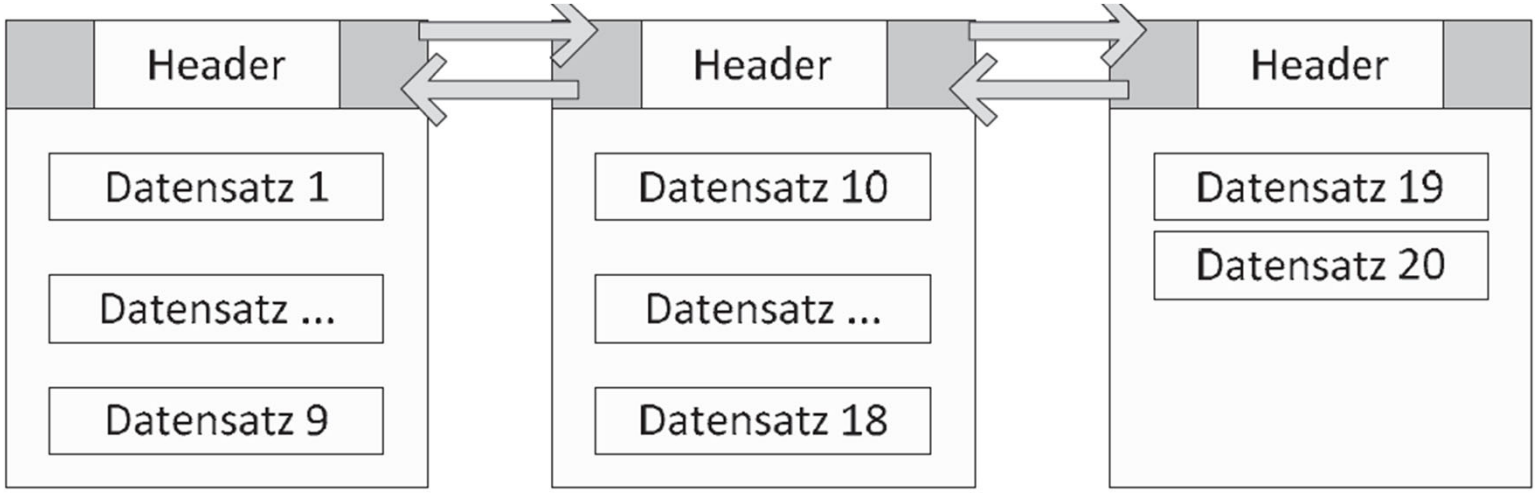
9 Übersetzungs- und Optimierungskomponente



9 Bsp.: Anlegen einer Kundentabelle

```
CREATE TABLE Kunden
(
    Kundennummer        INTEGER,
    Nachname   CHAR(200),
    Vorname             CHAR(200),
    Strasse             CHAR(200),
    Plz                 INTEGER,
    Ort                 CHAR(250)
)
```

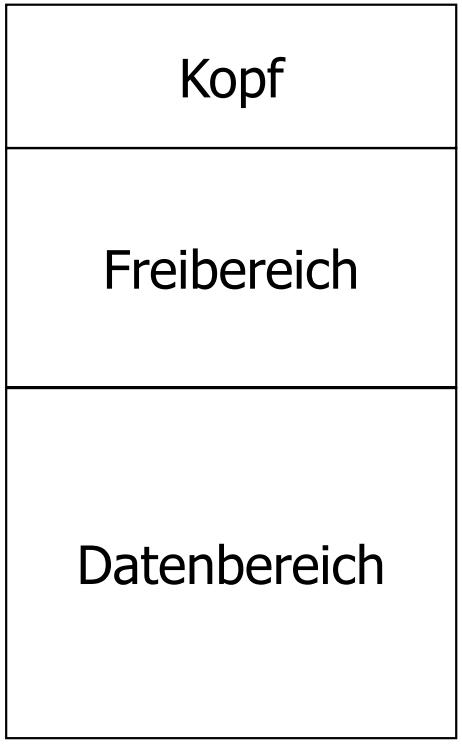
Kundennummer	4 Bytes
Nachname	200 Bytes
Vorname	200 Bytes
Strasse	200 Bytes
Plz	5 Bytes
Ort	250 Bytes
Gesamt	859 Bytes



Quelle: Cordts, S. u.a.: Datenbanken für Wirtschaftsinformatiker, 2011, S. 231

9 Seiten (engl. Pages)

- Die Datenbank speichert Daten in so genannten Seiten / Blöcken.
- Die Seitengröße ist auch ein Datenbankparameter dessen Wert mit dem Betriebssystem abgestimmt werden sollte.
- Die Seitengröße ist abhängig von den Disks bzw. dem RAID-Systemen (RAID steht für Redundant Array of Independent Disks).



Blockinformation (24 Byte)
Tabellen-Verzeichnis (4 Byte pro Tabelle)
Datensatz-Verzeichnis (variabel, abh. von # Tupel)

PCTFREE: Anteil des Freibereichs, der für Update-Operationen bereitgehalten werden muss. Kann für Insert-Operationen nicht genutzt werden.

PCTUSED: Nach Erreichung der PCTFREE-Grenze steht der Block nach Unterschreiten von PCTUSED wieder für Insert-Operationen zur Verfügung.

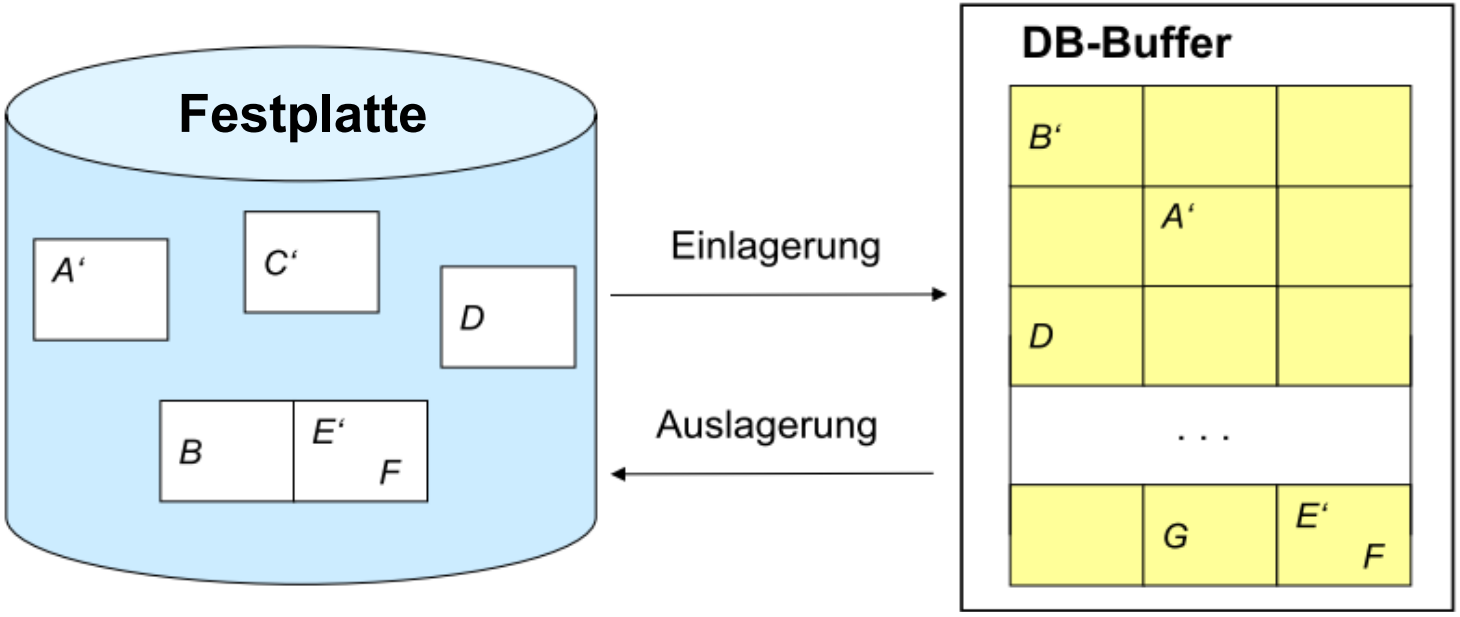
9 Seiten (engl. Pages)

Adressieren von Datenblöcken per Zeiger (engl. Pointer):

- Zum Lesen der Daten muss das DBMS eine Seite komplett in den Hauptspeicher laden. Eine Seite ist die kleinste physische Einheit, die vom DBMS in den Hauptspeicher geladen werden kann.
- Seiten bzw. Blöcke sind grundlegende Einheiten für das Lesen und Transferieren von Daten.
- Die Adressierung einer Seite erfolgt über physische Adressen (relevant für das Betriebssystem) und logischen Adressen (relevant für das DBMS).
- Ein Zeiger kann sich nicht nur auf eine Seite beziehen, sondern auf einen bestimmten Datensatz innerhalb einer Seite. Man nennt einen solchen Zeiger dann auch Record-ID (RID):

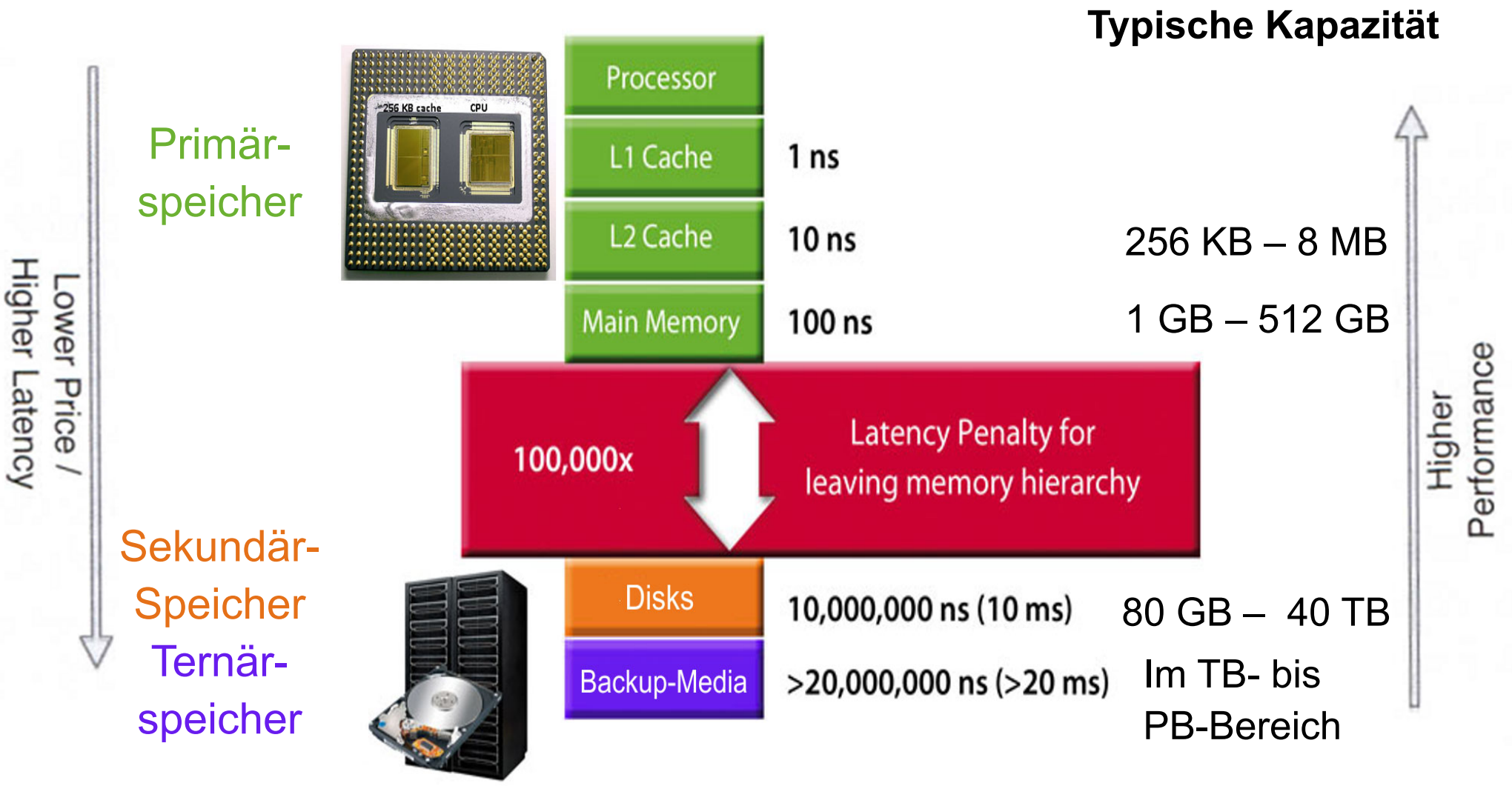
RID = Adresse der Seite + Aufsetzpunkt (Offset) für den Datensatz in der Seite

Puffer
Ausgezeichneter Bereichs des Hauptspeichers, der in Pufferrahmen gegliedert ist. Jeder Rahmen kann eine Seite (engl. Page) aufnehmen

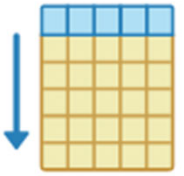


Jede Seite, die von der Platte in den DB-Puffer gelesen wird, wird dort so lange wie möglich vorgehalten, um unnötige I/Os auf die Platte zu vermeiden

Speicherhierachie
Zugriffslücke in Zahlen



9 Ermittlung der Treffermenge



Full Table Scan
Zur Erfüllung einer SELECT-Anfrage muss die ganze Tabelle gelesen werden. (sehr ressourcen-intensiv!!)

Beispiel: `Select * from Kunde where Name = "Müller";`

	KDNR	Name
RID 0000001:	12345	Müller
....		
RID 1000000:	08456	Müller

<u>Treffermenge ermitteln:</u>
Annahme: 1 Zeile hat 2KB,
Tabelle mit 10^6 Zeilen, 2 Treffer
➔ ca. 2GB (ohne Overhead)
müssen für
4KB Nutzdaten gelesen werden!

9 Zugriffsstrukturen / –verfahren

Verschiedene Speichertechniken sollen die Häufigkeit von Full Table Scans minimieren und das Ermitteln der Treffermenge einer Datenbankanfrage möglichst effizient (=schnell) gestalten. Dies sind:

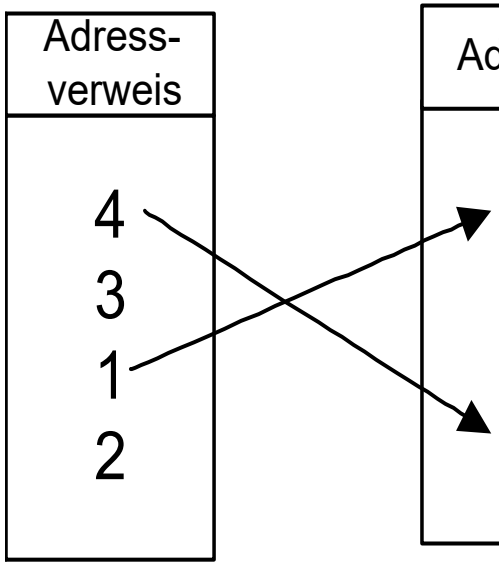
Zugriffspfade / Index
Zugriffstrukturen für den Zugriff auf die Tupel. Ein Eintrag in einem Index ist ein Paar (Schlüssel, Zeiger).

9 Index (Plural Indexe)

- Liefert einen Mechanismus zur Ermittlung der Treffermenge in einer Tabelle, ohne dass alle Zeilen der Tabelle gelesen werden müssen.
- Besitzt Einträge, die einen Schlüssel sowie Verweise auf die Einträge der indizierten Tabellen enthalten.
- Der Schlüssel heißt Suchschlüssel. Er muss nicht eindeutig sein (ist also kein Schlüsselkandidat).
- Ist eigentlich auch eine Art Tabelle und besitzt eine eigene Sortierung nach dem Suchschlüssel.
- Ein Index für einen Primärschlüssel heißt Primärindex.
- Weitere Indizes heißen dann Sekundärindizes mit entsprechenden Sekundärschlüsseln (entspricht dem Suchschlüssel). Sekundärschlüssel besitzen i.d.R. keine Schlüsseleigenschaft.
- Ein Index kann über ein Attribut (non-composite Index) oder über mehrere Attribute (composite Index) gelegt werden.

9 Grundsätzliche Arbeitsweise eines Index

Indexdatei



Hauptdatei

Adresse	Kunden-Nr.	Name	Ort
1	20	Müller	GE
2	30	Schulze	WZ
3	10	Maier	GI
4	40	Huber	OB

Die Indexdatei ermöglicht schnelle Sortierung und binäre Suche bzgl. des Namens.

Wie kann man Tabelle sortiert ausgeben?
Suchzeit?

9 Alternative Arbeitsweise eines Index

- Arbeitsweise (Anwendung z.B. Hauptspeicher)
  - In jedem Indexdatensatz wird die Adresse des nachfolgenden Indexdatensatzes gespeichert.

Indexdatei

RA	Schlüssel Kunden-Nr.	KF	RA
1	200	3	1
2	150	1	3
3	300	-	2
4	**100**	2	4

Hauptdatei

RA	Schlüssel Kunden-Nr.	Name
1	200	Müller
2	300	Maier
3	150	Huber
4	100	Schulte

KF = Kettenfeld (relative Adresse der Indexdatei)
RA = relative Adresse

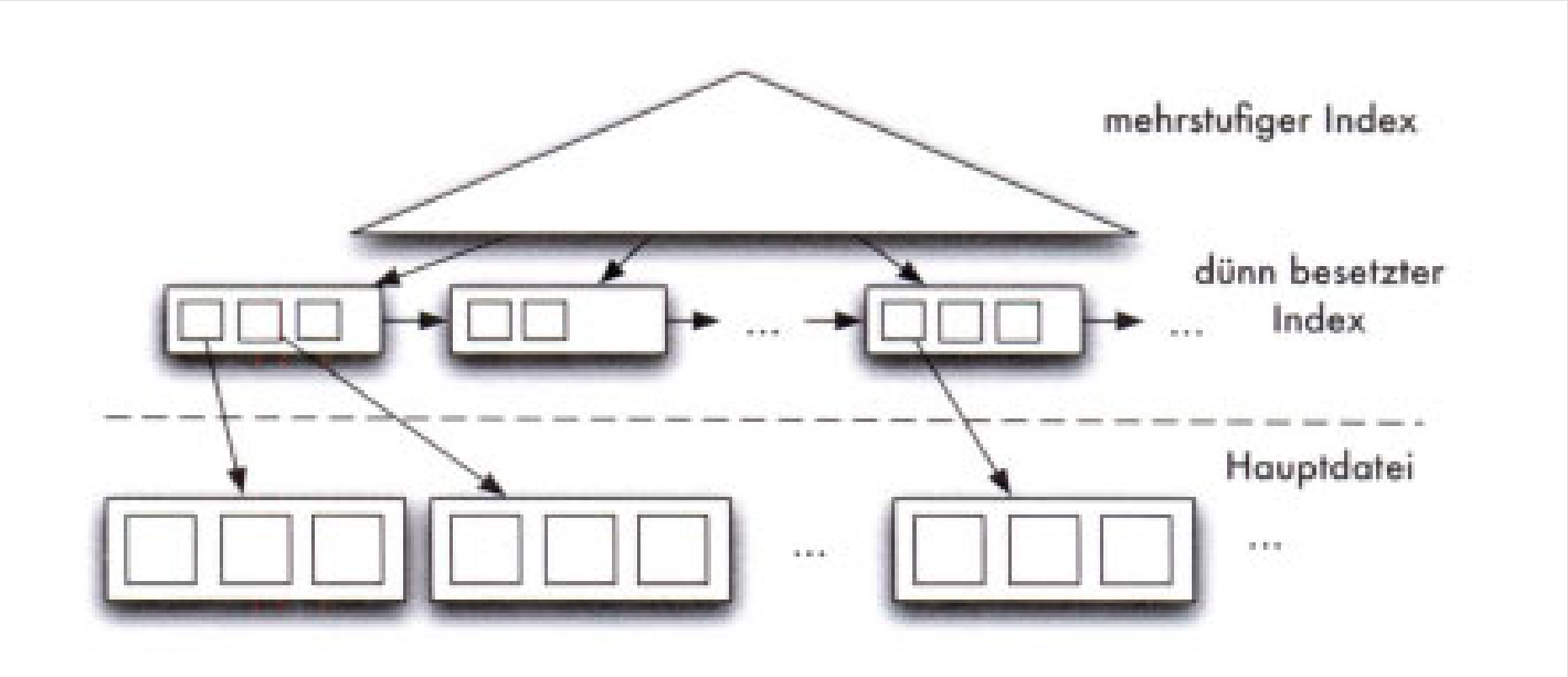
Quelle: Hohmann, Peter: „Datenverarbeitung für Wirtschaftsinformatiker und Betriebswirte“, 2001, S. 187

9 Weitere mögliche Indexe

Ein dünn besetzter Index speichert nicht für jeden Zugriffs-attributwert einen Eintrag in der Indexdatei. Ist die interne Relation sortiert nach dem Zugriffsattributen organisiert, so muss im Index nur jeweils ein Eintrag pro Seite der internen Relation enthalten sein. Der Index verweist auf den Seitenanführer, d.h. auf den bezüglich der Ordnung ersten Wert auf dieser Seite. Der nächste Indexeintrag verweist dann auf den Seitenanführer der Nachfolgeseite.

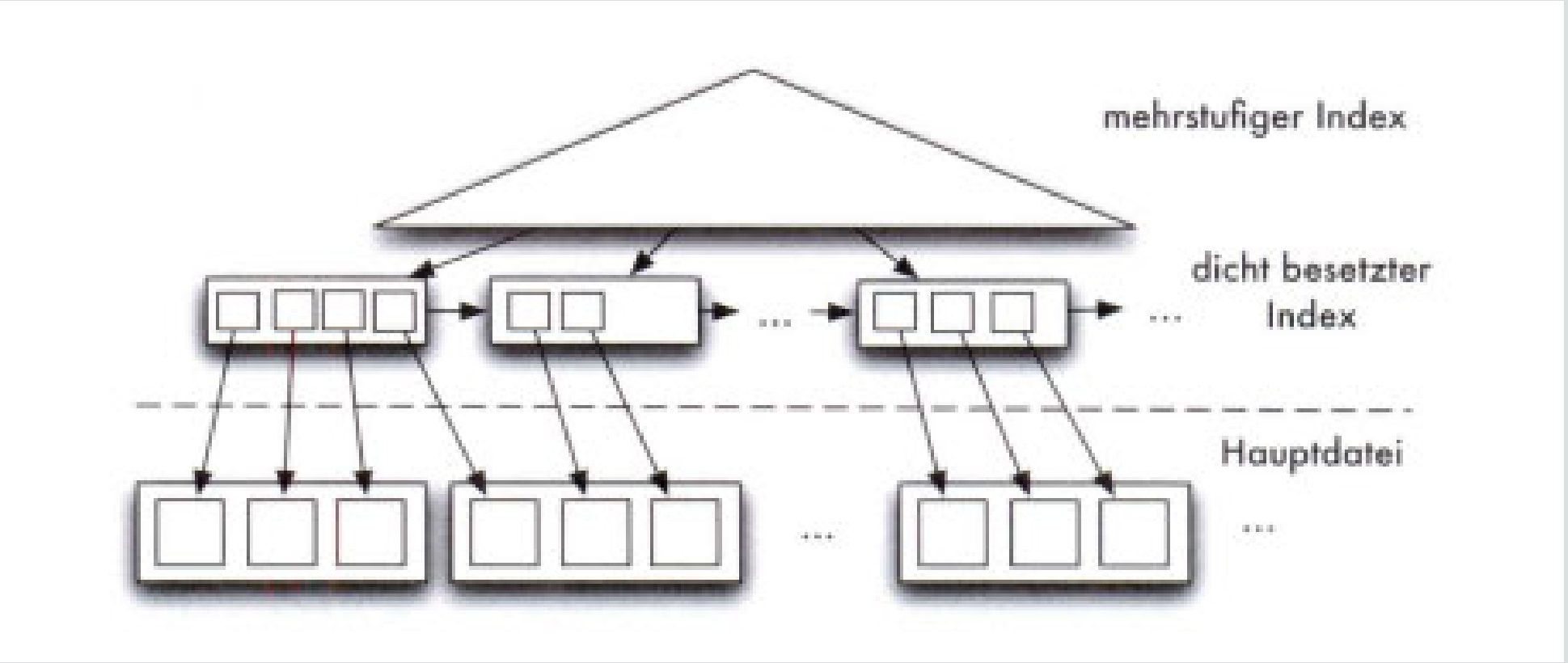
Ein dicht besetzter Index speichert dagegen für jeden Datensatz der internen Relation auch einen Eintrag in der Indexdatei.

9 Dünn besetzter Index



Quelle: Saake, G.; Sattler, K.-U., Heuer, A.: Datenbanken - Implementierungstechniken, 2011, S. 134

9 Dicht besetzter Index



Quelle: Saake, G.; Sattler, K.-U., Heuer, A.: Datenbanken - Implementierungstechniken, 2011, S. 134

9 Zugriffsstrukturen / –verfahren

Verschiedene Speichertechniken sollen die Häufigkeit von Full Table Scans minimieren und das Ermitteln der Treffermenge einer Datenbankanfrage möglichst effizient (=schnell) gestalten. Dies sind:

Zugriffspfade / Index
Zugriffstrukturen für den Zugriff auf die Tupel. Ein Eintrag in einem Index ist ein Paar (Schlüssel, Tupel).

(Interne/Physische) Organisationsformen
Form der Speicherung der Tupel einer der Tabelle.

9 Interne/Physische Organisationsform

- Soll so ausgerichtet sein, dass die Anzahl der Plattenzugriffe bei der Recherche oder der Manipulation von Daten minimal ist.
- Der gewünschte Datensatz ist durch möglichst wenige I/Os zu finden. Hierzu wird die Anordnung der gespeicherten Daten optimiert. Eine solche Anordnung nennt man Speicherstruktur.
- Verschiedene Speicherstrukturen haben verschiedene Laufzeiteffizienz („Performance“)- und Speicherplatz-Charakteristika, die für jeweils unterschiedliche Anwendungen / Datenbankbereiche von Vorteil sein können.
- Eine optimale Datenorganisation kann nur erzielt werden, wenn die Datenstrukturen, statistische Informationen zum Datenvolumen sowie Ausprägungen innerhalb der Wertebereiche bekannt sind und berücksichtigt werden.
- **Keine Speicherstruktur ist optimal für alle Anwendungen!**

9 Interne/Physische Organisationsform

- **Sequentielle Speicherung**
  Die Datensätze werden unmittelbar nacheinander abgespeichert und können nur in der gespeicherten Folge verarbeitet werden (z. B. auf einer Magnetbandkassette, in einer Datei ohne Schlüssel).

  Bemerkung: Die Neuaufnahme von Daten ist mit der Komplexität *O(1)* zwar konkurrenzlos günstig, der Aufwand für die lineare Suche in einer unsortierten Datei mit *n* Datensätzen beträgt *O(n),* d.h. im schlechtesten Fall („worst case“) müssen alle Datensätze durchlaufen werden, bis der gesuchte Satz gefunden ist („full table scan“).

- **Indizierte Organisation**
  Logisch sortierter Index, z.B. als Baumstruktur oder Bitmap-Index

- **Gestreute Organisation**
  Speicherung durch direkte Adressierung oder durch sogenannte Hash-Verfahren

9 Logisch sortierter Index: B* - Bäume

B*-Baum der Ordnung m

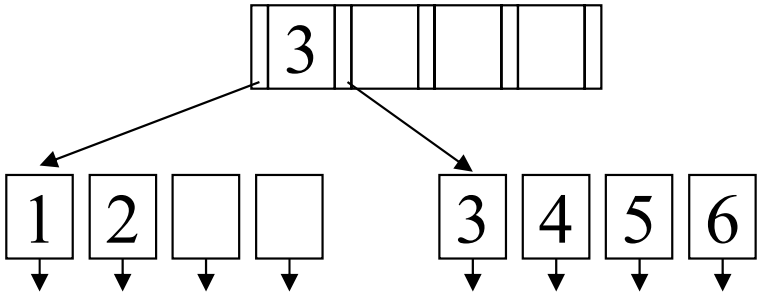
Ein Baum mit den folgenden Eigenschaften

- Alle Blätter sind gleich weit von der Wurzel entfernt sind.
- Jeder Knoten außer der Wurzel hat zwischen *m* und 2*m* Elemente $(x_1,\ldots,x_j)$. Tupel unter *i*-tem Kind haben Schlüssel in $[x_{i-1},x_i -1]$
- Die Wurzel hat maximal 2m Einträge (sie kann ein Blatt sein!). Die Einträge sind nach dem Schlüssel sortiert.

Bemerkungen:

- Verweise auf die Hauptstruktur bzw. die Daten liegen nur in den Blättern (es handelt sich um einen hohlen Baum).
- Der Baum wird bei Modifikation der indizierten Daten reorganisiert.
- Fälschlicherweise werden B* manchmal auch B^+ - Bäume genannt.

9 Bsp.: B*-Baum mit *m*=2



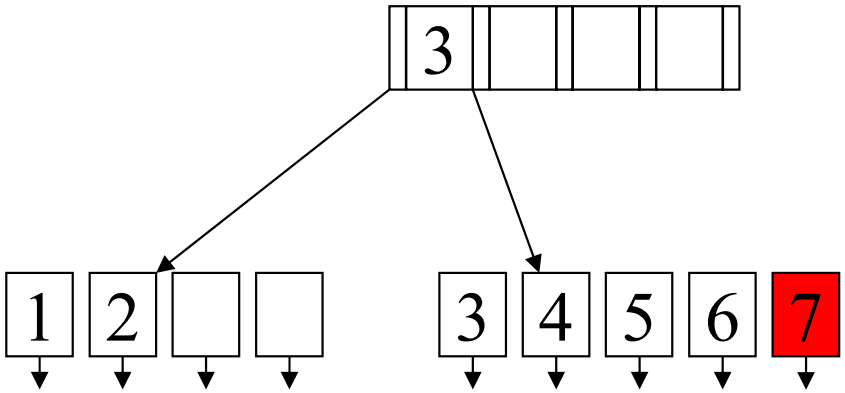
Ordnung m=2, d.h. jeder Knoten hat zwischen m und 2m viele Kinder

Einfügealgorithmus in B*-Baum:

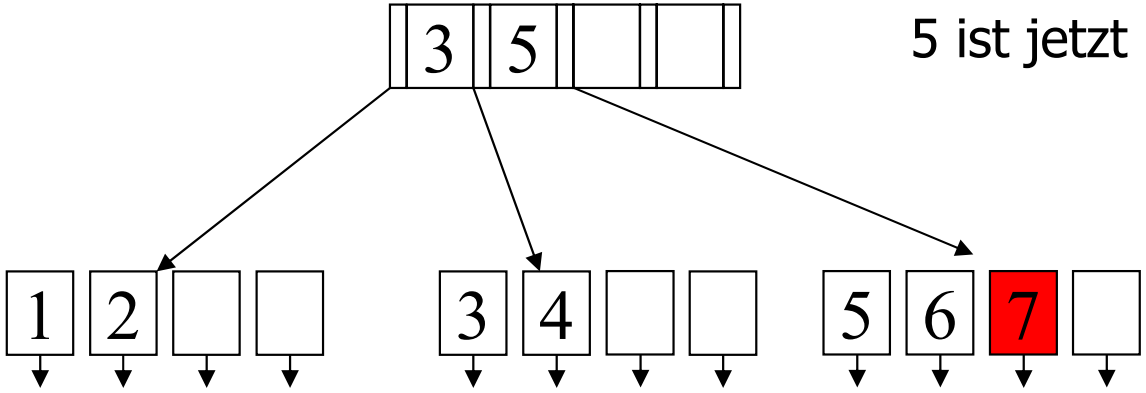
- Suche das Blatt, wo der neue Schlüssel eingefügt werden muss. Füge dort ein.
- Falls das Blatt überfüllt ist:
  - Teile den Knoten. Alle Schlüssel kleiner dem mittleren Schlüssel bleiben in der Seite. Der Rest wandert in die neue Seite.
  - Der mittlere Schlüssel wird mit Verweisen auf die beiden Kind-Knoten in den Vaterknoten eingefügt. Falls es noch keinen gibt, wird einer erzeugt.
  - Falls der Vaterknoten überfüllt ist, wird auch der Vaterknoten geteilt. Hier wandert der mittlere Schlüssel nur nach oben nicht in die neuen Indexseiten auf der gleichen Stufe. Auch weiter „oben“ kann es zu Teilungen kommen.

Wie sieht der obige Baum nach **Einfügen des Schlüssels 7** aus?

9 Bsp.: Einfügen in B*-Baum mit $m=2$



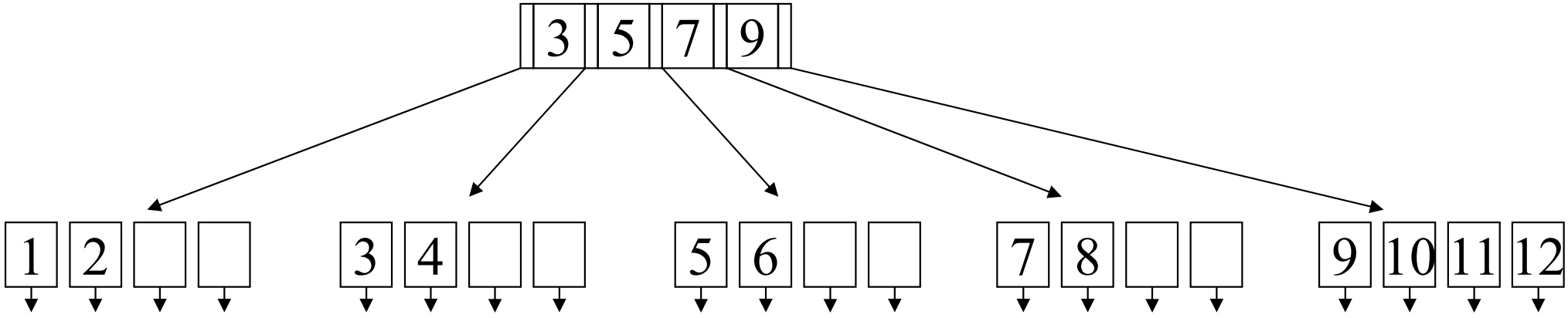
Überfüllt (mehr als 2m Kinder). Also teilen.



5 ist jetzt im Index und im Blatt!

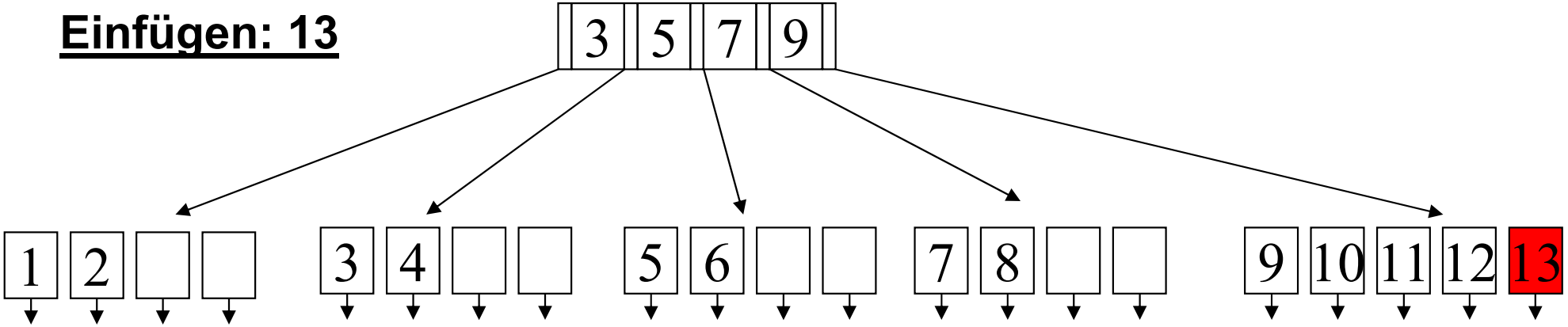
9 Bsp.: Einfügen in B*-Baum mit $m=2$

Baum nach Einfügen von 8, 9, 10, 11 und 12:

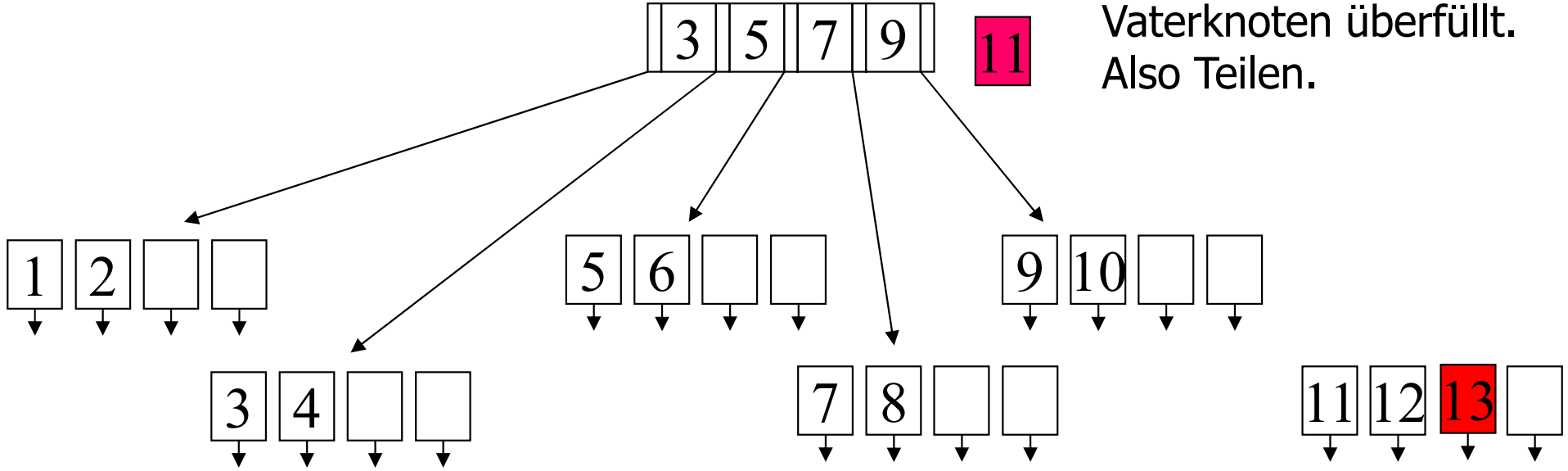


9 Bsp.: Einfügen in B*-Baum mit $m=2$

Einfügen: 13

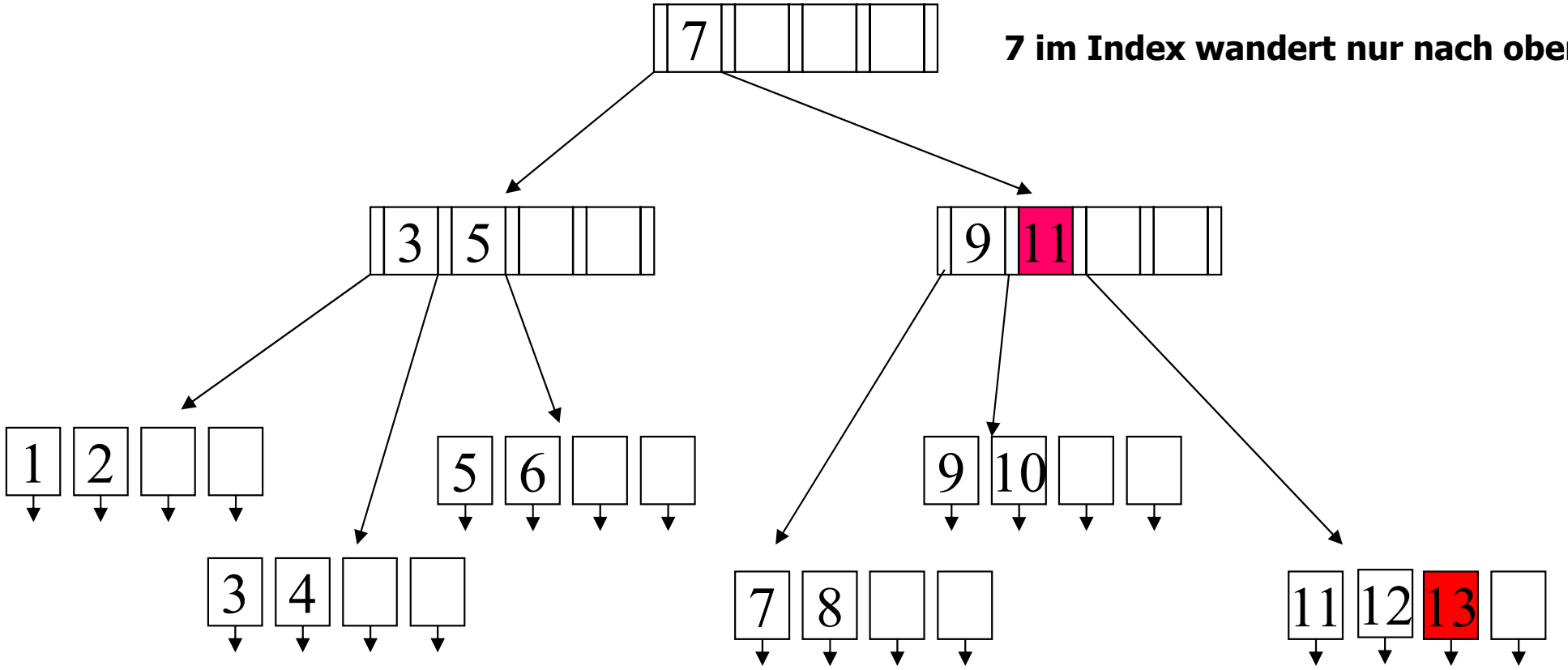


Überfüllt (mehr als 2m Kinder). Also teilen.



Vaterknoten überfüllt. Also Teilen.

9 Bsp.: Einfügen in B*-Baum mit *m*=2

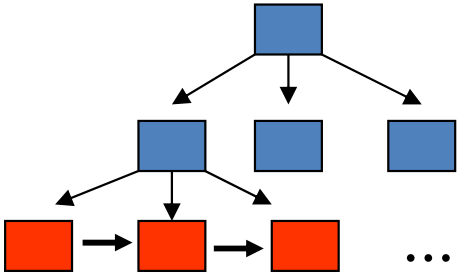


7 im Index wandert nur nach oben!

n=13, m=2: Maximale Höhe des Baumes: ceil($\log_m(n/m)$)=3

9 B*-Baum: Anmerkungen

- Der Wert für *m* ist üblicherweise „sehr groß“. Er hängt von der Block- bzw. Seitengröße und der Größe der indizierten Spalten ab.
- **Daher sind B*-Bäume auf nummerischen Spalten sehr effizient.** Update und Delete werden analog behandelt. Die Komplexität ist gleich.
- Auf Blattebene gibt es in der Regel noch Zeiger von jedem Blatt zum folgenden Blatt. Damit wird ein sequentielles Lesen effizienter (z.B. Bereichsanfragen (a,b): Alle x mit a<=x<=b.).



Bitmap-Index

Bitmap-Index

Eine zweidimensionale Struktur zum Speichern der Werte in Bitvektoren

- Dimension 1: Rowid
- Dimension 2: die Bitleiste

Beispiele:

Attribut 1: Bestellstatus mit den Ausprägungen: A, O

Attribut 2: Region mit den Ausprägungen: Nord, Süd, Ost, West

BI1	A	O
Rowid1	0	1
Rowid2	1	0
Rowid3	0	1
....		

BI2	N	S	O	W
Rowid1	0	1	0	0
Rowid2	1	0	0	0
Rowid3	0	0	0	1
...				

Bitstrings

Es darf nur eine sehr geringe Zahl an Ausprägungen geben.

9 Bitmap-Index Anmerkungen

- Anwendung im Data Warehouse-Bereich
- Bitmap-Indices können aufgrund der geringen Größe im Hauptspeicher verarbeitet werden
- Eher teuer in Erstellung und vor allem in der Wartung (Update, Insert, Delete)
- Erlaubt eine 64-fach schnellere Berechnung auf 64-Bit-Architekturen
- Es gibt Varianten (Kodierte Bitmap-Indices, Mehrkomponenten Bitmap-Indices)

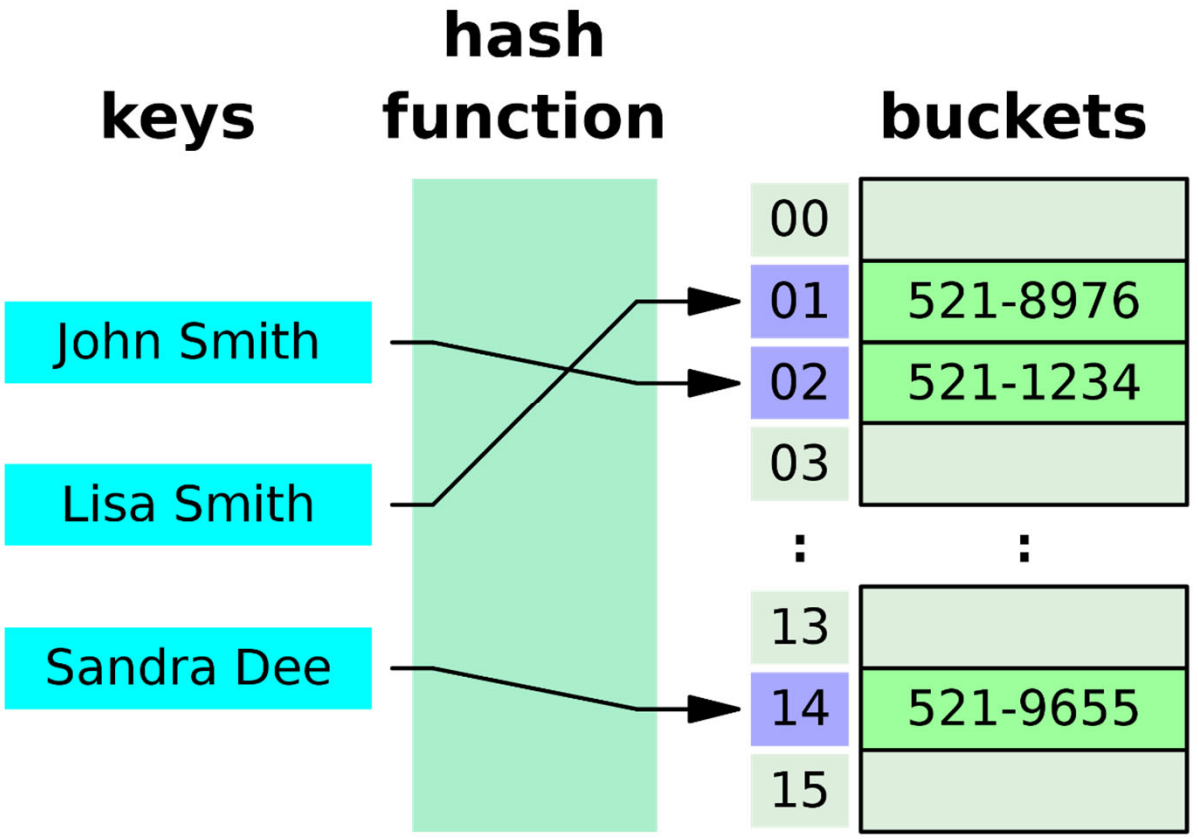
9 Vor- und Nachteile der indizierten Organisation

Vorteile	Nachteile
Direkte Adressierbarkeit	Indextabellen belegen Speicherplatz
Primärschlüssel und/oder Sekundärschlüssel möglich	Update-Zeiten werden langsamer
Breite Anwendung für die Datenorganisation	Integritätsprobleme zwischen Index und Datendatei (durch z.B. Abstürze bei schlechter Implementierung)

Quelle: Hohmann, Peter: „Datenverarbeitung für Wirtschaftsinformatiker und Betriebswirte“, 2001, S. 188

Gestreute Organisation: Hashing

Hash-Algorithmen liefern häufig keine eindeutige Adresse, dadurch kommt es zu Kollisionen, die in einem Überlaufbereich durch Verkettung gespeichert werden müssen.



9 Hash-Strukturen

In manchen Fällen ist es sinnvoll, die Daten auf festgelegte Bereiche zu streuen. Dies geschieht mit sogenannten Hash-Verfahren.

Ein Schlüssel (häufig der PK) wird auf Hash-Buckets umgerechnet (besteht aus einer oder mehreren Seiten).

Dies geschieht durch eine Schlüsseltransformation (engl. hash function), d.h. eine Abbildung einer Menge von Schlüsseln in eine Menge von Adressen, die einen zusammenhängenden Adressraum bilden.

Ist ein Bucket voll, so werden Überlaufseiten verkettet (Überlaufbereich).

9 Anforderungen an Hash-Funktionen

- Die Transformationsvorschrift muss mit einfacher Berechnung und ohne große Kosten eingehalten werden können.
- Die belegten Adressen müssen gleichmäßig über den Adressraum verteilt sein.
- Die Wahrscheinlichkeit für Mehrfachbelegungen, d.h. die Verwendung gleicher Adressen für mehrere Schlüssel, sollte für alle Schlüsselwerte gleich groß sein, unabhängig von den gegebenen Daten.

Bemerkung: Diese drei Anforderungen werden z.B. von sogenannten *universellen Hash*-Funktionen erfüllt.

Quelle: Meier, A.: Relationale und postrelationale Datenbanken, 2007, S. 119

9 Hashing mit Divisionsrest

Eine einfache *Hash*-Funktionen ist zum Beispiel:

Jeder Schlüssel wird als natürliche Zahl interpretiert, indem die Bitdarstellung verwendet wird. Die Schlüsseltransformation oder *Hash-Funktion H für einen Schlüssel k und eine Primzahl p* ist durch die Formel

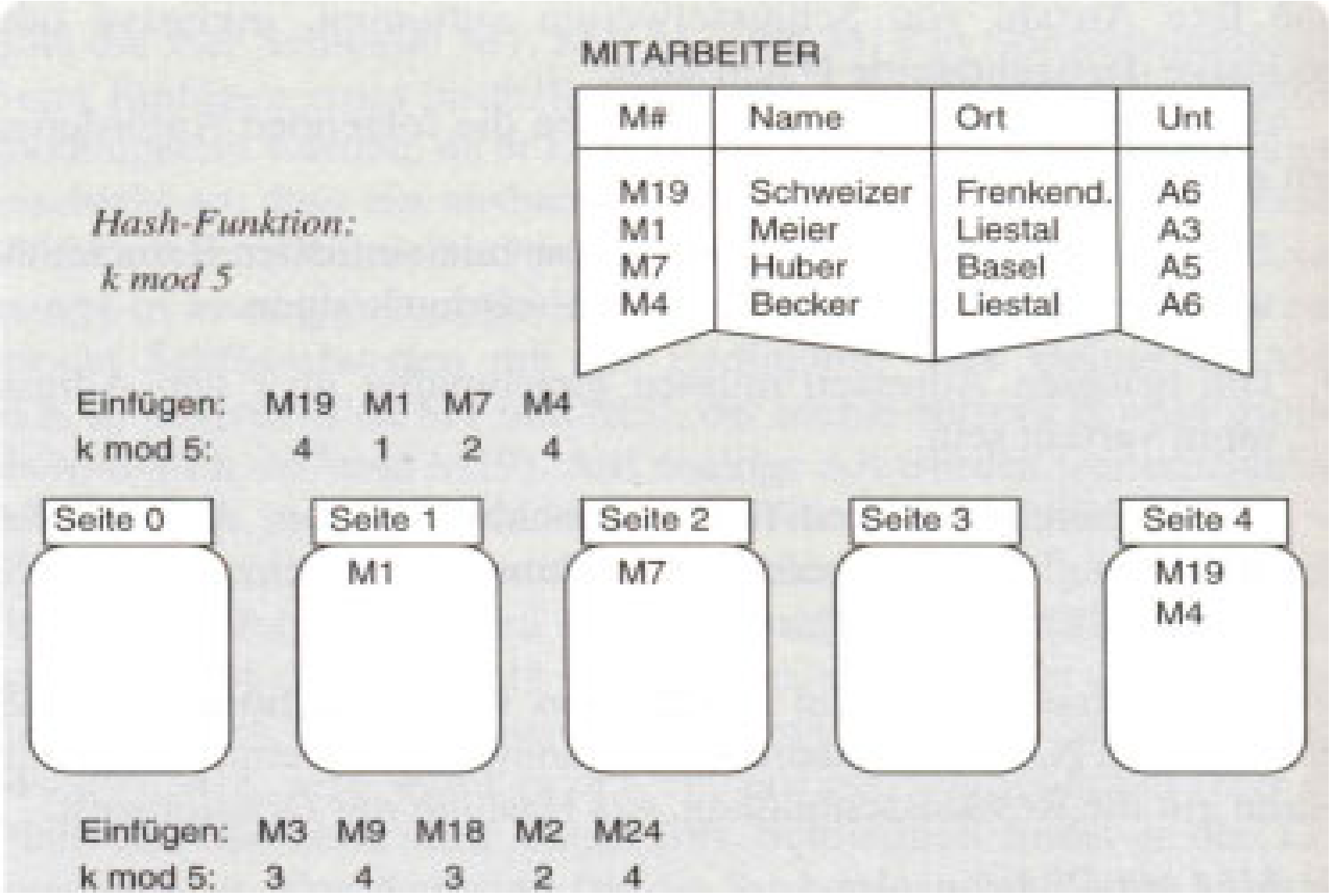
$$H(k) := k \bmod p$$

gegeben. Der ganzzahlige Rest «k mod p» – der Division des Schlüsselwertes k durch die Primzahl p – bildet eine relative Adresse oder Seitennummer. Bei diesem Divisionsrestverfahren bestimmt die Wahl der Primzahl p die Speicherausnutzung und den Grad der Gleichverteilung.

Bemerkung: Die Menge $\boldsymbol{H} = \{ H_a(k) := a*k \bmod p \mid a \in \{1..p\}\}$ bildet eine universelle Hashklasse. Durch zufällige Wahl von a aus {1..p} werden alle 3 Anforderungen an die Hash-Funktionen erfüllt.

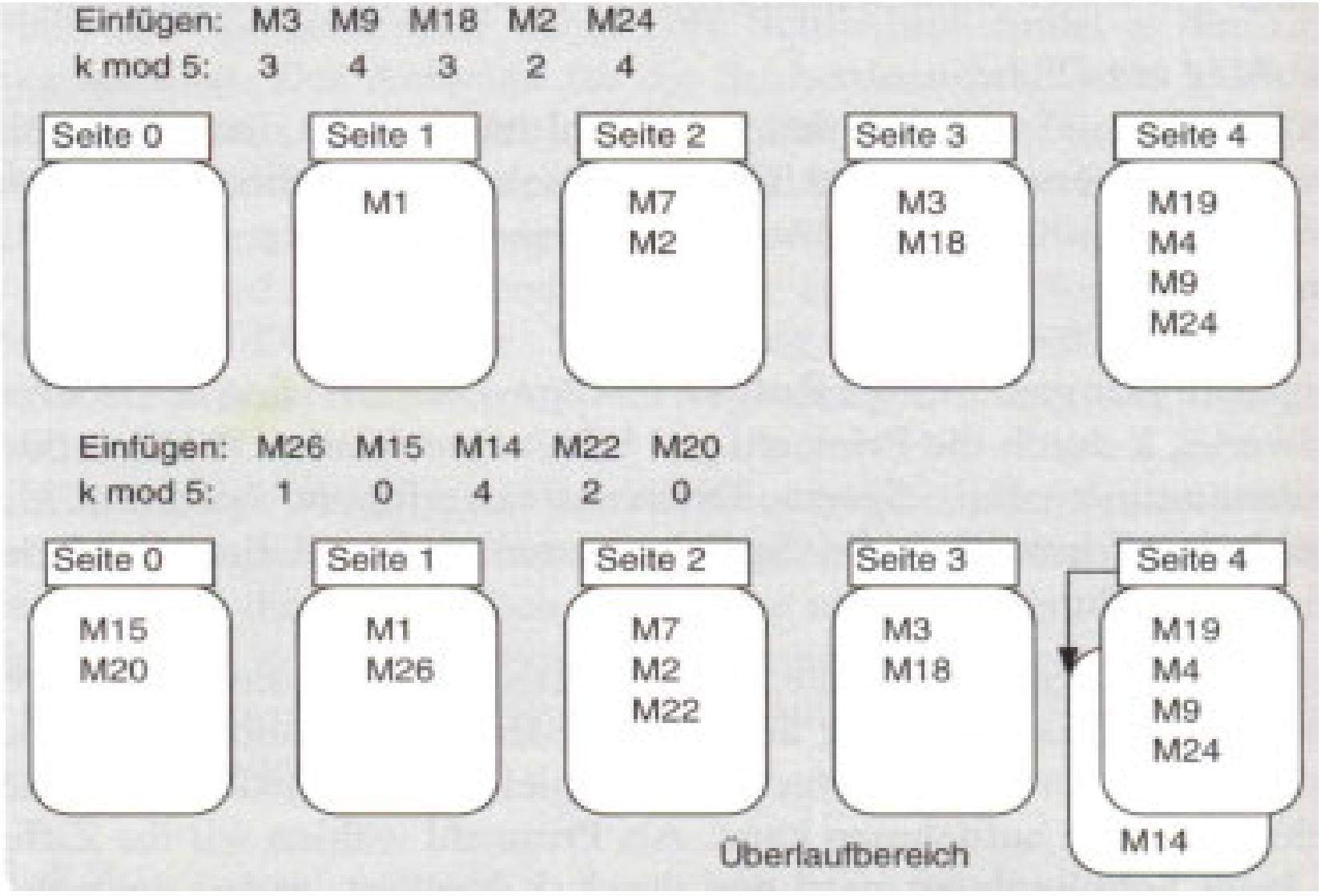
Quelle: Meier, A.: Relationale und postrelationale Datenbanken, 2007, S. 119

9 Bsp.: Hashing mit Divisionsrest



Quelle: Meier, A.: Relationale und postrelationale Datenbanken, 2007, S. 120

9 Bsp.: Hashing mit Divisionsrest



Quelle: Meier, A.: Relationale und postrelationale Datenbanken, 2007, S. 120

9 Vor- und Nachteile vom Hashing

Vorteile	Nachteile
Sehr schneller Zugriff möglich	Man braucht etwa doppelt soviel Speicher, wie für die eigentlichen Daten, damit Kollisionen eher unwahrscheinlich werden.
	Kein Zugriff mit Teilschlüssel möglich, Hash-Funktion braucht volle Infos.

9 Aufgabe 1: Anfrageoptimierung

Gegeben seien folg. Tabellen:

Abteilung:

Abt_Nr	Abt_Name	Abt_Kürzel
1	Organisation	ORG
2	IT-Service	ITS

Mitarbeiter:

Per_Nr	Name	Ausbildung	Gehalt	Abt_Nr
1	Maler	Informatiker	4000.00	1
2	Schulz	Programmierer	3800.00	2
3	Müller	Mathematiker	4100.00	2
4	Schmidt	Medieninform	4200.00	2
5	Krause	Techn.Inform.	4200.00	2
6	Berg	BWL	4000.00	1
7	Schader	Elektrotechnik	3900.00	2
8	Müller	Mathematiker	4200.00	2

9 Aufgabe 1: Anfrageoptimierung

```
SELECT M.Name, A.Abt_Name
        FROM Mitarbeiter M, Abteilung A
        WHERE M.Abt_Nr = A.Abt_Nr
                AND A.Abt_Name = 'Organisation'
```

a) Was leistet die SQL-Abfrage?

b) Geben Sie dieselbe Abfrage algebraisch durch eine Sequenz von Operatoren gemäß der Notation der Relationenalgebra an (Hinweis: Verwenden Sie für das kartesische Produkt das Symbol |x|, für die Selektion σ und für die Projektion π).

c) Erstellen Sie den zugehörigen Anfragebaum und kennzeichnen Sie dabei die Wurzel-, Stammknoten und Blätter. Wie sehen die Zwischen- und Ergebnistabellen aus, d.h. wie viele Tupel (mit Angabe der jeweiligen Spaltenanzahl) werden in jedem Schritt für die konkreten Tabellenbelegungen erzeugt bzw. durchsucht?

d) Erzeugen Sie einen effizienteren, aber zugleich äquivalenten Anfragebaum gemäß den Prinzipien der algebraischen Optimierung. Wie viele Tupel (mit Angabe der jeweiligen Spaltenanzahl) werden nunmehr in jedem Schritt für die konkreten Tabellenbelegungen erzeugt bzw. durchsucht?

9 Aufgabe 2: B*-Baum

Fügen Sie in einen zu Beginn leeren *B*-Baum* vom Typ $m=1$ der Reihe nach die folgenden Schlüssel ein:

9, 6, 2, 8, 4, 13 und 5

Welches Aussehen hat der *B*-Baum* nach den Einfüge-Operationen? Zeichnen Sie den Baum nach jeder Ausführung neu. Unveränderte Knoten dürfen Sie mit „…“ abkürzen.

Bitte beachten Sie: Für jeden Knoten k sind die Schlüsselwerte eines linken Subbaums immer kleiner als der von k. Die des rechten sind größer oder gleich dem von k.

9 Aufgabe 3: Hashing

Gegeben sei die folgende Hash-Funktion $H(k) := k \bmod p$, wobei k der Schlüssel und p eine Primzahl sei. Weiterhin die Tabelle „Mitarbeiter“, die Tupel mit den folgenden Ausprägungen des Primärschlüssels „Personal-Nr“ in der gegebenen Reihenfolge aufweist:

1, 3, 9, 18, 2, 27, 24, 19, 7, 4, 6, 15, 14, 20 und 22

Erzeugen Sie sukzessive für jeden Schlüssel die Seitenzuordnung (Skizze!). Als Primzahl p wird die Zahl 5 gewählt. Gehen Sie dabei davon aus, dass jede Seite maximal vier Schlüsselwerte aufnehmen kann – interpretieren Sie das Ergebnis.