Kapitel 7: Speicherungsstrukturen

7.1 Speicherung von Datensätzen (Records)

Charakteristika moderner Magnetplatten
(z.B. Seagate Cheetah 18 mit Ultra−SCSI− oder Fibre−Channel−Schnittstelle)

Durchmesser	3.5 Zoll
Speicherkapazität	18.2 GBytes
Preis	ca. 4000,− DM
Größe	ca. 14 x 10 x 4 cm
Gewicht	ca. 1.2 kg
Energieverbrauch	15 Watt
Zuverlässigkeit: Mean Time To Failure	800000 Stunden (> 75 Jahre)
Anzahl der Oberflächen	24
Anzahl der Zylinder	6962
Spurkapazität	87 bis 128 KBytes
Rotationszeit	6 msec (10000 U/min)
Mittlere Armpositionierungszeit	5.7 msec
Minimale Armpositionierungszeit	0.6 msec
Übertragungsrate	14.5 bis 21.3 MBytes/sec
Zeit zum wahlfreien Zugriff auf 1 Block à 4 KBytes	ca. 9 msec
Zeit zum sequentiellen Zugriff auf 30 Blöcke à 4 KBytes	ca. 12 msec

Charakteristika von Speicherhierarchien

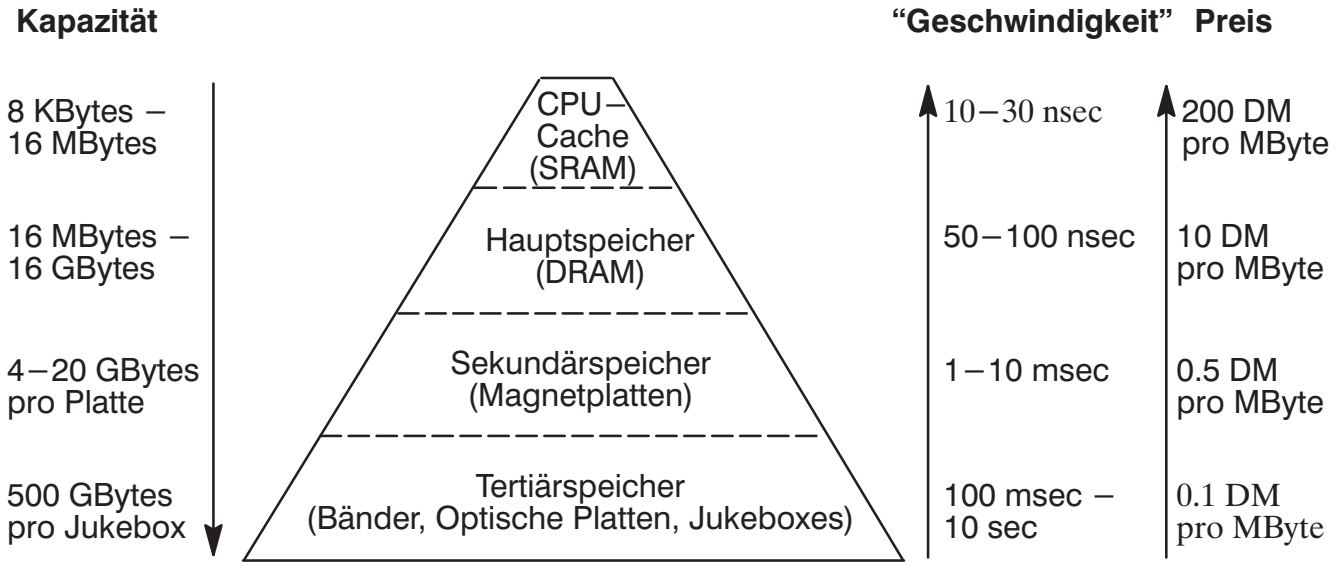
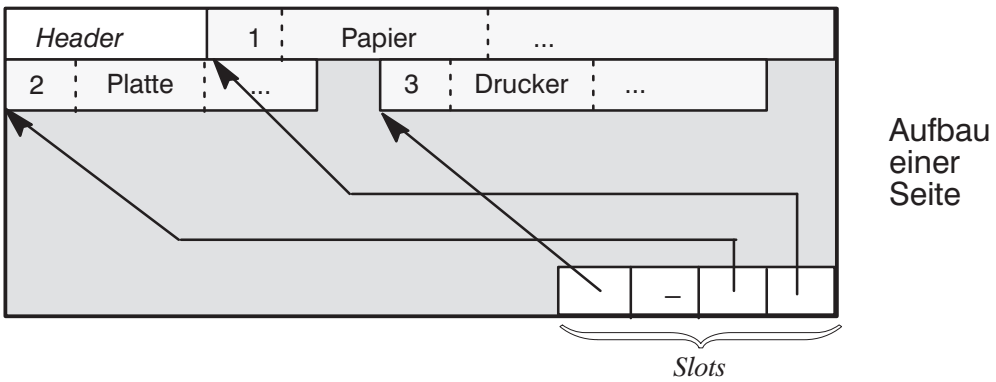


Abbildung von Tupeln (Datensätzen, Records) auf Seiten (Blöcke)

Seiten (Blöcke) sind Speicherbereiche fester Größe (typischerweise 2 bis 8 KBytes), in denen ein oder mehrere Datensätze (Records) gespeichert werden.
Eine Seite ist die kleinste Einheit für den Transport zwischen Sekundärspeicher und Hauptspeicher sowie für die Pufferung im Hauptspeicher.



Aufbau einer Seite

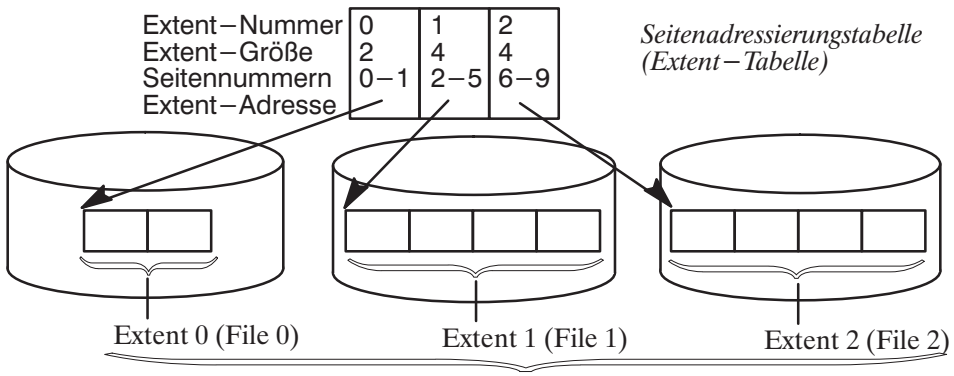
Der durch Datensätze belegte Bereich in einer Seite wächst von vorne nach hinten, das Slot−Array wächst von hinten nach vorne.
Der Header einer Seite enthält u.a. Angaben zur Größe des gesamten freien Platzes in der Seite, zur Größe des zusammenhängenden freien Platzes zwischen Datensätzen und Slot−Array, zur Größe des Slot−Arrays, usw.

Mögliches Speicherungsformat eines Datensatzes:

Bei diesem Format werden Nullwerte überhaupt nicht gespeichert.

Abbildung von Seiten auf Externspeicher

Eine oder mehrere Relationen und/oder Indizes werden in einem *Tablespace* gespeichert.
Ein Tablespace ist ein logisch zusammenhängender Seitenadreßraum.
Logische Seitennummern werden mit Hilfe einer Seitenadressierungstabelle (Extent−Tabelle) in physische Blocknummern auf der Platte übersetzt.



Extents sind typischerweise gegenüber dem Betriebssystem als reguläre Files alloziert.
Unter Unix entsprechen Extents eines DBS in der Regel ganzen Plattenpartitionen (”raw disks”).

Verwaltung von Tablespaces

Allokation von Tablespaces:
CREATE TABLESPACE tablespacename
DATAFILE filename SIZE size {”,” filename SIZE size}

Erweiterung von Tablespaces:
ALTER TABLESPACE tablespacename
ADD DATAFILE filename SIZE size {”,” filename SIZE size}

Zuordnung von Relationen (und Indizes) zu Tablespaces:
CREATE tablename (...
[TABLESPACE tablespacename]
[STORAGE INITIAL size NEXT size] [PCTFREE percent])

Die Parameter der Tablespace−/Table−Spezifikation erlauben eine Einflußnahme auf die Allokation von Daten auf den zugrundeliegenden Speichermedien
(z.B.: Allokation bestimmter Daten auf derselben oder verschiedenen Platten,
Festlegung eines Füllgrades von Seiten beim Laden, usw.).
Tablespaces sind auch eine Einheit für verschiedene Dienstprogramme (z.B. Backup/Restore).

Beispiel für die Wichtigkeit der Zuordnung von Relationen/Indizes zu Tablespaces:

Tablespace TS1 bestehe aus Extents, die alle auf Platte /dev/rsd0a liegen;
Tablespace TS2 bestehe aus Extents, die alle auf Platte /dev/rsd1b liegen.
Es gebe viele Zugriffe der Form
SELECT * FROM Produkte WHERE PNr = :pnr,
die vom DBS wie folgt ausgeführt werden:
1) Zugriff auf den Index ”ProduktPNrIndex” über PNr
2) Zugriff auf das entsprechende Treffertupel.

Sinnvolle Zuordnung:
Produkte → TS1
ProduktePNrIndex → TS2
damit der Zugriffsarm auf der ”Indexplatte” nur wenig bewegt werden muß !

Verwendung von Betriebssystem− oder Hardwaremechanismen zur Datenverteilung über mehrere Platten (Striping)

Sogenannte *Disk−Arrays*, auch *RAID−Speichersysteme* genannt (Redundant Arrays of Independent Disks), unterstützen mit ihren Controllern eine − Striping genannte − automatische Round−Robin−Verteilung der Blöcke eines Files. Im einfachsten Fall liegt bei einem Disk−Array mit n Platten 0 bis n−1 die Seite i auf der Platte i mod n. Darüber hinausgehend wird in der Regel angeboten, die Anzahl logisch aufeinanderfolgender Blöcke, die physisch zusammenhängend auf einer Platte abgelegt werden, als Tuning−Option zu spezifizieren. Diese Größe wird als *Striping−Granulat* bezeichnet. Bei einem Striping−Granulat von s Blöcken liegt also Block i auf der Platte floor(i/s) mod n.

RAID−Speichersysteme beinhalten außerdem eine gewisse Redundanz in den Daten, indem über eine bestimmte Anzahl von Blöcken verschiedener Platten jeweils ein Parity−Block auf einer weiteren Platte mitgeführt wird. Dadurch sind RAID−Systeme in der Lage, den Ausfall einer Platte (abgesehen von Performance−Aspekten) vollständig zu maskieren.

Moderne Betriebssysteme unterstützen diese Art von flexiblem Striping auch unabhängig von spezifischer Hardware. Datenbanksysteme können sich auf diese Art der gleichmäßigen Datenverteilung über mehrere Platten verlassen, verlieren dann aber u.U. die explizite Kontrolle über die Plazierung der Tablespaces.

Beispiel eines Disk−Arrays mit n=4 Platten und einem Striping−Granulat von s=2 Blöcken:

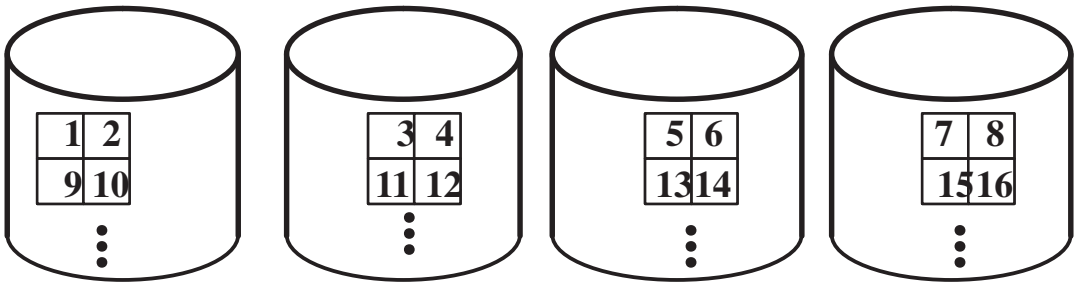


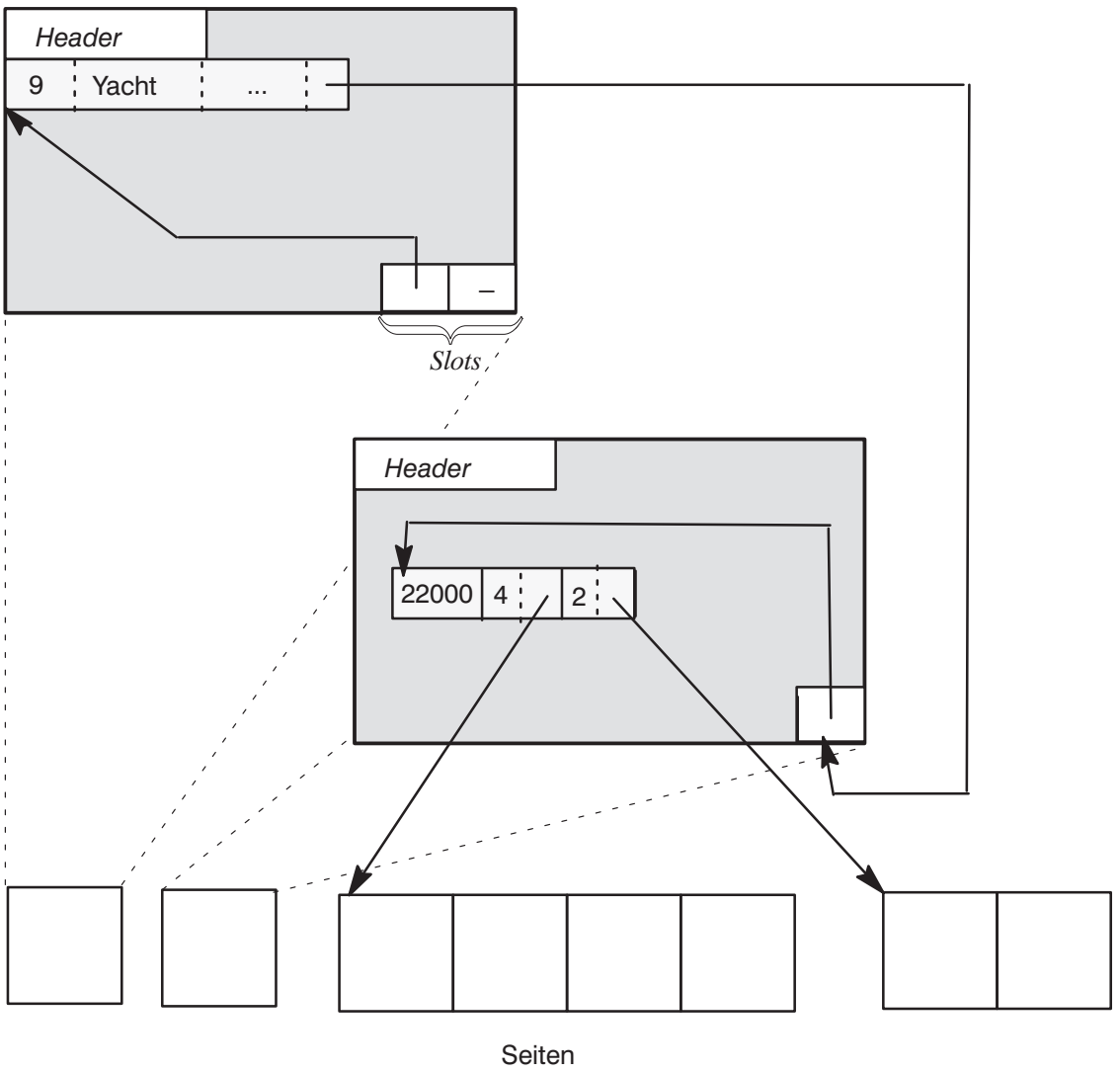
Abbildung von BLOBs (Attribute vom Typ LONG RAW) auf Seiten

Bilder, Videos u.ä. werden in Attributen vom Typ LONG RAW − sogenannten BLOBs ("Binary Large Objects", "Long Fields") − gespeichert. Solche Attribute, auf denen man Operationen Read/Write (Byte−Position, Length) ausführen kann, werden als Pointer−Arrays repräsentiert mit Einträgen der Form

<Anzahl benachbart allozierter Seiten, Zeiger auf Anfangsseite>

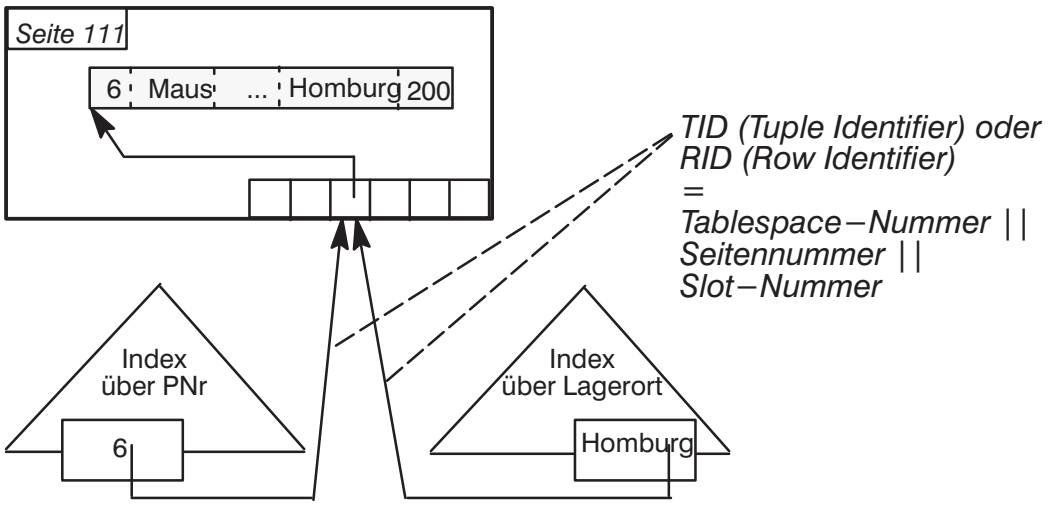
und einem zusätzlichen Längenfeld für den gesamten Attributwert. Die Seiten, auf denen der Attributwert selbst liegt, sind in der Regel für diesen Zweck dediziert (enthalten also keine anderen Tupel) und können daher einfacher aufgebaut sein (z.B. ohne Slot−Array). Diese Seiten sollten soweit wie möglich benachbart alloziert werden; dies ist Aufgabe der Freispeicherverwaltung, die dafür z.B. das Buddy−System verwenden könnte.

Bei besonders langen Feldern muß das Pointer−Array ggf. zu einem Baum − typischerweise einem B*−Baum (siehe Kapitel Zugriffsstrukturen) − mit Bytepositionen als Suchschlüssel generalisiert werden.

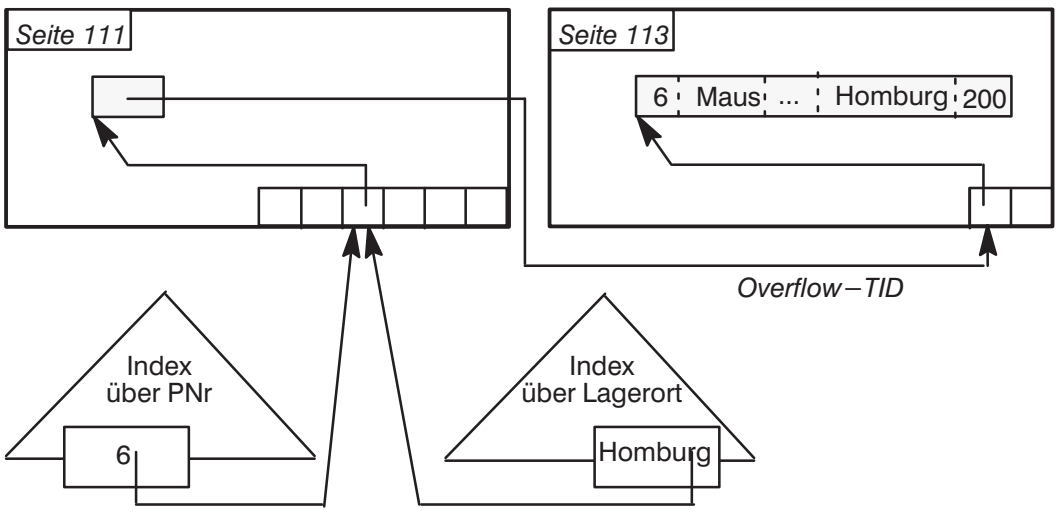


Satzadressierung mit TIDs

Sätze werden indirekt über die Slots einer Seite adressiert.
Vorteil: Sätze sind innerhalb einer Seite verschiebbar, ohne daß Zeiger in *anderen Seiten* geändert werden müssen.



Beim ”Auslagern” eines Satzes in eine andere Seite wird in der ursprünglichen Seite ein ”Overflow−TID” hinterlassen.
⇒ Die meisten Sätze sind mit einem Seitenzugriff adressierbar;
eine Minderheit ist mit 2 Seitenzugriffen adressierbar.
Bei einer Häufung von ”Overflow−TIDs” ist eine Reorganisation nötig (z.B. Unload&Load).



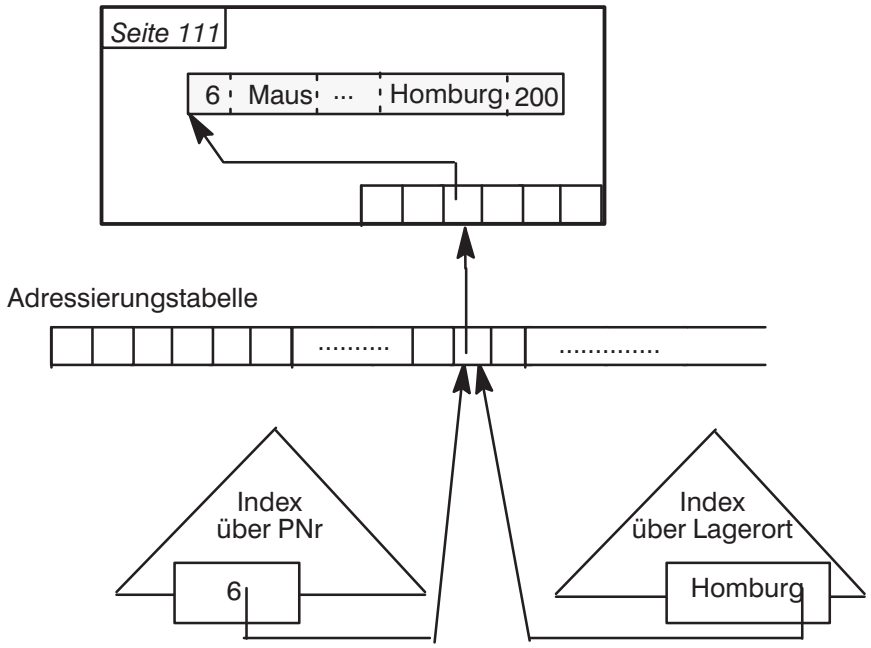
Satzadressierung mit DBKs:

Sätze werden indirekt über eine globale Adressierungstabelle adressiert. Die “logische” Adresse eines Satzes heißt “Database Key” (DBK) oder auch “ Satznummer”.

Vorteil: Verschiebbarkeit von Sätzen wie bei der TID−Adressierung.

Nachteil gegenüber TIDs:

Wenn die Adressierungstabelle nicht zumindest größtenteils hauptspeicherresident ist, löst der Direktzugriff auf einen Satz generell zwei Plattenzugriffe aus.



7.2 Pufferverwaltung

Alle DBS puffern häufig referenzierte Seiten (z.B. einen Teil der Indexseiten) im Hauptspeicher. Die Größe des Datenbankpuffers (Cache, Buffer Pool) ist durch den DBA spezifizierbar.

Schnittstelle und Architektur der Pufferverwaltung

Die Pufferverwaltung (Cache Manager, Buffer Manager) stellt der Anfrageausführung bzw. dem Record−Manager eine Schnittstelle zum Seitenzugriff zur Verfügung. Dazu gehören folgende Operationen:

- *fix (in pageno, out vmaddr)*
  zum Einlesen einer Seite in den Puffer mit der virtuellen Speicheradresse der Seite als Resultat
- *unfix (in pageno, in dirtyflag)*
  zur Freigabe einer gepufferten Seite zum Zurückschreiben, ggf. mit Hinweis auf Änderungen
- *flush (in pageno)*
  zum sofortigen Zurückschreiben einer gepufferten Seite auf die Platte

Zwischen dem fix− und dem unfix−Aufruf für eine Seite ist die Seite weder aus dem Puffer verdrängbar noch im Puffer verschiebar; sie ist auf der von fix zurückgelieferten virtuellen Speicheradresse fixiert. In dieser Zeitspanne kann die Anfrageausführung bzw. der Record−Manager direkt byteadressiert auf den Seiteninhalt zugreifen.

Die Pufferverwaltung ruft ihrerseits den I/O−Manager des Datenbanksystems zum Einlesen oder Zurückschreiben von Seiten auf. Initiiert werden solche Lese− oder Schreibzugriffe auf die Platte einerseits durch fix− oder flush−Aufrufe oder durch vorausschauendes *Prefetching* des Pufferverwalters. Beispielsweise kann der Pufferverwalter bei einem sequentiellen Scan über einen Tablespace selbständig veranlassen, daß jeweils eine oder mehrere Seiten im voraus in den Puffer eingelesen werden (d.h. bevor sie von der Anfrageausführung per fix−Aufruf explizit angefordert werden).

Pufferersetzungsstrategie

Wenn der Puffer voll belegt ist und es wird eine Seite benötigt, die nicht im Puffer vorhanden ist, muß eine gepufferte Seite ersetzt werden.
Falls die ersetzte Seite seit ihrer letzten Einlagerung in den Puffer geändert wurde, muß sie vor ihrer Ersetzung erst noch auf Platte zurückgeschrieben werden.

Optimale Ersetzungsstrategie:
Ersetze diejenige Seite, die am weitesten in der Zukunft wieder benötigt wird.

Zeit	Referenzierte Seite	Pufferinhalt (Puffergröße: 3 Seiten)		
1	1	**1**		
2	2	1	**2**	
3	3	1	2	**3**
4	2	1	2	3
5	4	1	**4**	3
6	3	1	4	3
7	5	1	4	**5**
8	1	1	4	5

Realistische (und typische) Ersetzungsstrategie (basierend auf stochastischem Modell für die Seitenreferenzen der Vergangenheit) – *LRU (Least Recently Used)*:
Ersetze diejenige Seite, auf die am längsten nicht mehr zugegriffen worden ist.

Zeit	Referenzierte Seite	LRU–Kette (Puffergröße: 3 Seiten) alt ..... jung		
1	1	**1**		
2	2	1	**2**	
3	3	1	2	**3**
4	2	1	3	2
5	4	3	2	**4**
6	3	2	4	3
7	5	4	3	**5**
8	1	3	5	**1**

Zusätzliche I/O–Optimierung durch *Prefetching* von Seiten, von denen das DBS weiß, daß sie in naher Zukunft benötigt werden
(z.B. bei einem *Relationen–Scan*: sequentielles Lesen aller Seiten einer Relation).

Probleme mit reinem LRU bei DBS

1) Häufige Anfragen vom Typ
SELECT * FROM Produkte WHERE PNr = :pnr
mit Indexzugriff
Annahme: Es gibt 2 Indexseiten und 20 Datenseiten für Produkte.

Zeit	Referenzierte Seite	LRU−Kette (3 Seiten) alt ..... jung	Optimaler Pufferinhalt
1	I1	**I1**	**I1**
2	D1	I1 **D1**	I1 **D1**
3	I2	I1 D1 **I2**	I1 D1 **I2**
4	D11	D1 I2 **D11**	I1 **D11** I2
5	I1	I2 D11 **I1**	I1 D11 I2
6	D2	D11 I1 **D2**	I1 **D2** I2
7	I2	I1 D2 **I2**	I1 D2 I2
8	D12	D2 I2 **D12**	I1 **D12** I2
...			

Achtung:
Das Problem ist eigentlich noch komplexer, da es innerhalb einer Klassen von Seiten ungleichmäßig verteilte Zugriffshäufigkeiten geben kann und sich diese noch dazu im Laufe der Zeit ändern können.

2) Gleichzeitige Anfragen vom Typ
SELECT * FROM Kunden WHERE KNr = :knr − mit Indexzugriff −
und vom Typ
SELECT SUM(summe) FROM Bestellungen − ohne Indexunterstützung.
Annahme: Es gibt 1 Indexseite für Kunden, 2 Datenseiten für Kunden
und 20 Datenseiten für Bestellungen.

Kommerzielle DBS eliminieren diese zweite Art von Problemen, indem sie Seitenreferenzen von *Relationen−Scans* (sequentielles Lesen aller Seiten einer Relation) speziell behandeln. Solche Seiten werden an das ”wertlose” Ende der (modifizierten) LRU−Kette gehängt.

Zeit	Referenzierte Seite	Modifizierte LRU−Kette (Puffergröße: 4 Seiten) ”wertlos” ..... ”wertvoll”
1	I1	**I1**
2	K1	I1 **K1**
3	B1	**B1** I1 K1
4	I1	B1 K1 I1
5	K2	B1 K1 I1 **K2**
6	B2	**B2** K1 I1 K2
...		

7.3 Spezifische Speicherungsstrukturen für objektorientierte DBS

Leistungsanforderungen an objektorientierte DBS

Die Leistungsanforderungen an objektorientierte DBS sind natürlich sehr stark abhängig von der jeweiligen Anwendungsklasse. Für technische Anwendungen (wie z.B. CAD) wurde der sogenannte OO7−Benchmark definiert, der die wesentlichen Anforderungen in einer synthetischen Last zusammenfaßt. Der Benchmark dient vor allem dem Leistungsvergleich zwischen verschiedenen Systemen.

OO7−Benchmark:

Objekte:

- CompositeParts
  mit PartID, PartType, BuildDate als Attributen und
  IsDescribedIn als Referenz auf Documents sowie
  HasAtomicParts als Referenzen auf AtomicParts
- Documents
  mit DocID, Title, Text als Attributen und
  Describes als Referenz auf CompositeParts
- AtomicParts
  mit PartID, PartType, BuildDate, X, Y als Attributen und
  UsedIn als Referenzen auf CompositeParts
- Assemblies
  d.h. azyklische Graphen mit CompositeParts und AtomicParts als Knoten

Queries:

- Traversierung der Assemblies
  z.B. Ausgabe aller Assemblies, in denen ein bestimmtes AtomicPart verwendet wird
- Suche nach CompositeParts oder AtomicParts mit bestimmten Attributwerten
  z.B. mit einem BuildDate innerhalb eines gegebenen Zeitraums ,
  mit einem bestimmten PartType, usw.
- Kombinationen von Traversierung und attributwertorientierter Suche
- Änderungsoperationen

Traversierung von Objektreferenzen

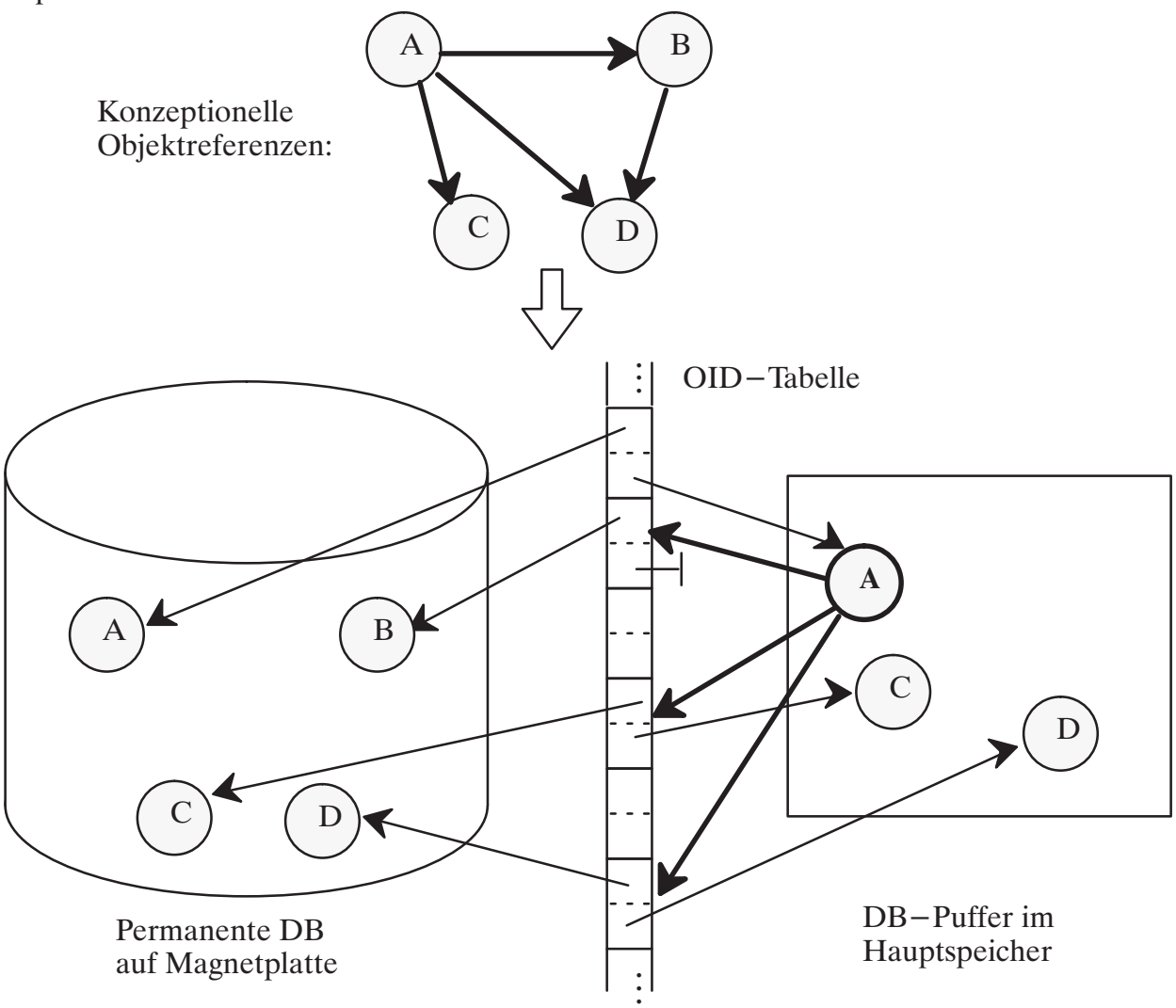
Objektreferenzen (die von Objekten oder auch von Indizes ausgehen) werden – in Anlehnung an die TID–Technik relationaler DBS – als sekundärspeicherorientierte, indirekte Zeiger realisiert. Diese Zeiger heißen OIDs (eine interne Form von “Object IDs”).
Ein OID gibt eine Position in einer persistenten, aber in der Regel immer hauptspeicherresidenten OID–Tabelle (Adreßumsetzungstabelle) an.

Ein Eintrag der OID–Tabelle hat zwei Komponenten:

- die persistente Adresse des Objekts,
  in der Regel in Form eines TIDs (Seiten–Nr. || Slot–Nr.)
- die virtuelle Hauptspeicheradresse des Objekts, falls sich dieses gerade im Datenbankpuffer befindet.

Mit Hilfe der OIDs und der OID–Tabelle können Objektreferenzen mit akzeptabler Effizienz traversiert werden, und gleichzeitig bleiben die Referenzen auch bei Verschiebungen der Objekte innerhalb von Seiten und über Seitengrenzen hinweg stabil. Bei solchen Verschiebungen müssen nur die persistenten Adressen in den entsprechenden Einträgen der OID–Tabelle aktualisiert werden.

Beispiel:

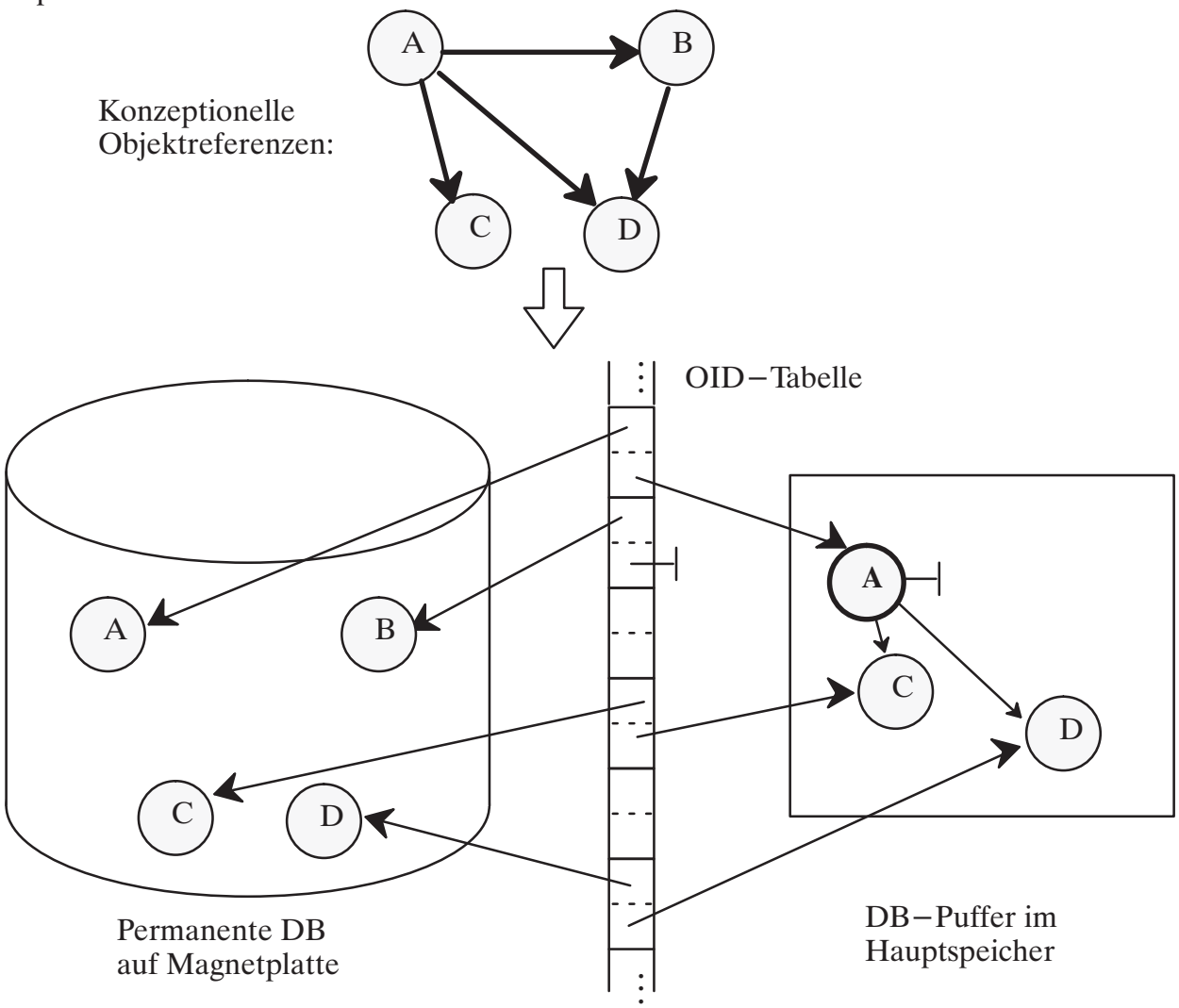


Pointer Swizzling

Bei der Einlagerung von Objekten in den Hauptspeicher werden Referenzen, die von solchen Objekten ausgehen (entweder sofort nach der Einlagerung oder bei der ersten Dereferenzierung) in virtuelle Speicheradressen konvertiert. Damit können Referenzen mit derselben Geschwindigkeit wie C++−Pointer traversiert werden. Diese Leistungssteigerung ist für besonders anspruchsvolle Anwendungen (z.B. CAD) essentiell. Sie wird beispielsweise im objektorientierten Datenbanksystem ObjectStore unterstützt.

Referenzen auf Objekte, die nur auf Platte stehen, werden in den hauptspeicherresidenten Objekten durch ungültige virtuelle Speicheradressen ersetzt (Nil−Zeiger). Bei der Dereferenzierung wird dann ein Signal ausgelöst (eine sog. “Trap”), das vom DBS abgefangen und behandelt wird, indem das referenzierte Objekt in den Hauptspeicher geladen wird.

Beispiel:



Ergänzende Literatur zu Kapitel 7:

T. Härder, E. Rahm, Datenbanksysteme – Konzepte und Techniken der Implementierung, Springer–Verlag, 1998

J. Gray, A. Reuter, Transaction Processing: Concepts and Techniques, Morgan Kaufmann, 1993, Chapters 13–14

T. Härder, Realisierung von operationalen Schnittstellen, in: P.C. Lockemann, J.W. Schmidt (Hrsg.), Datenbank–Handbuch, Springer–Verlag, 1987

A.L. Tharp, File Organization and Processing, John Wiley & Sons, 1988

P. Corrigan, M. Gurry, Oracle Performance Tuning, O'Reilly & Associates, Inc., 1993