

3. Speichersystem / Pufferverwaltung

- Dateiverwaltung
- Direkte vs. indirekte Seitenzuordnung
 - Segmentkonzept
 - Indirekte Einbringstrategien
- DB-Pufferverwaltung: Grundlagen
 - Allgemeine Charakteristika
 - Speicherzuteilung im Puffer
 - Suche im Puffer
 - Schreibstrategien (Force vs. Noforce)
 - Lesestrategien (Prefetching, Demand Fetching)
- DB-Pufferverwaltung: Seitenersetzungsverfahren
 - Klassifikation von Ersetzungsverfahren
 - Einfache Verfahren: LRU, FIFO, CLOCK, GCLOCK, LRD ...
 - LRU-K
 - Adaptives LRU



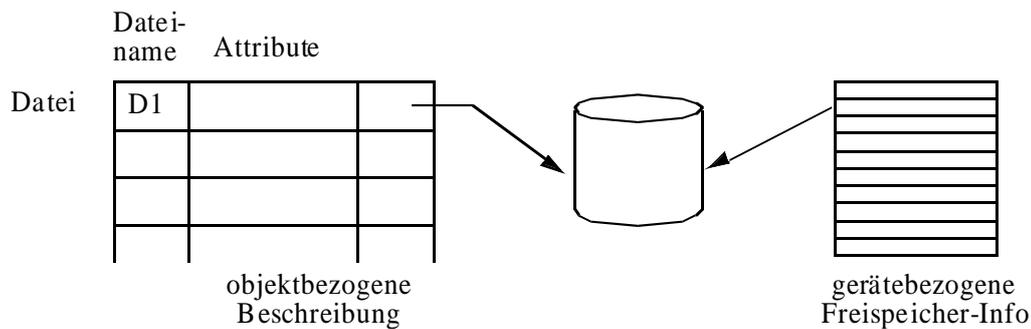
Speichersystem



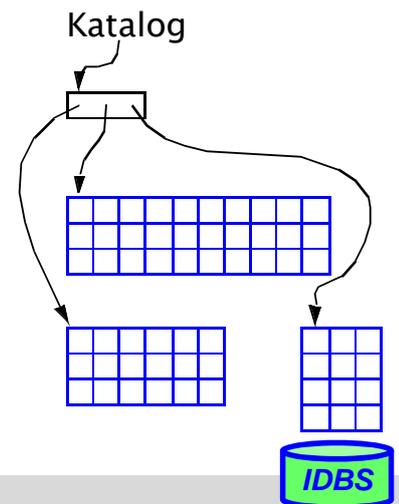
- Dateiverwaltung (Speicherzuordnung)
 - DB-Speicher = Menge von Dateien
 - Dateien repräsentieren externe Speichermedien in geräteunabhängiger Weise
 - Dateisystem: Abbildung von Dateien / physischen Blöcken auf Externspeicher
 - Dateiartern: permanent/temporär, sequentiell/direkt adressierbar, ...
 - wichtig.: schnelle Adressierbarkeit, dynamische Wachstumsfähigkeit von Dateien
- Segment- und Seitenverwaltung (Seitenzuordnung)
 - Indirekte Einbringstrategien



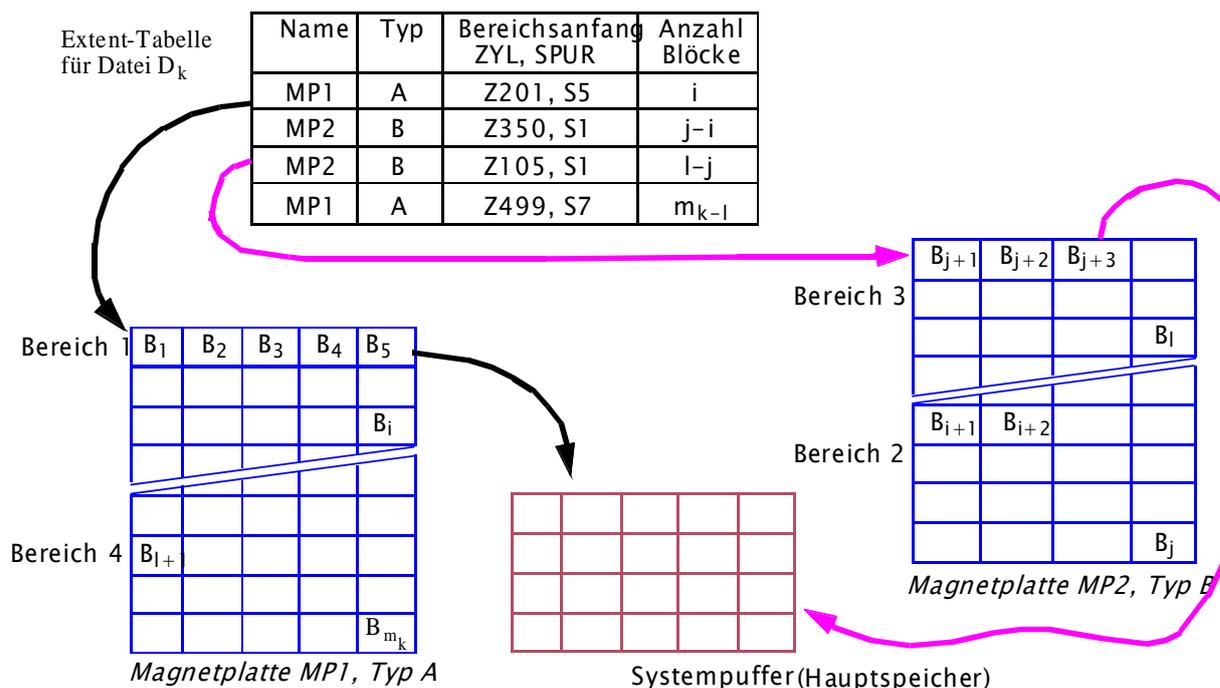
Dateikonzept: Realisierungsaspekte



- Katalog für alle Dateien mit Deskriptor pro Datei
 - Dateimerkmale: Name, Größe, Externspeicherzuordnung, Owner, Erzeugungszeitpunkt, ...
- Freispeicherverwaltung für Externspeicher (z.B. Bitlisten)
 - Anlegen/Reservieren von Speicherbereichen (Extents): Erstzuweisung, Erweitern
- Einheit des physischen Zugriffs: Block
 - feste Blocklänge pro Datei
- Dynamische Extent-Zuordnung
 - kleine Extent-Liste pro Datei ermöglicht geringe Zugriffskosten
 - Schnelle sequentielle Zugriffe innerhalb von Extents
 - Flexibilität hinsichtlich Wachstum



Blockzuordnung über Extent-Tabellen



Dargestellte Aktionen der Zugriffsprimitive: Hole Block B_5 ; Hole Block B_{j+3}

Segmentkonzept

- Aufteilung des logischen DB-Adressraumes in Segmente mit sichtbaren Seitengrenzen
- Realisierung eines Segmentkonzeptes:
 - Ermöglichung indirekter Einbringstrategien
 - Segmente als Einheiten des Sperrens, der Recovery und der Zugriffskontrolle
 - Unterschiedliche Segmentarten: permanent/temporär, öffentlich/privat,
 - Abbildung auf Dateien
- DBS-Unterstützung durch sog. **Tablespaces** (tw. auch Indexspaces)
 - Segmenttyp zur Speicherung von Tabellen (Relationen) sowie ggf. Indexstrukturen
- Tablespace kann i.a. auf mehrere Dateien abgebildet werden


```
CREATE TABLESPACE tablespacename
DATAFILE filename SIZE size { "," filename SIZE size }
```
- Zuordnung von Relationen zu Tablespaces


```
CREATE TABLE tablename ( ...
[ TABLESPACE tablespacename ]
[ STORAGE INITIAL size NEXT size ] [PCTFREE percent ] )
```

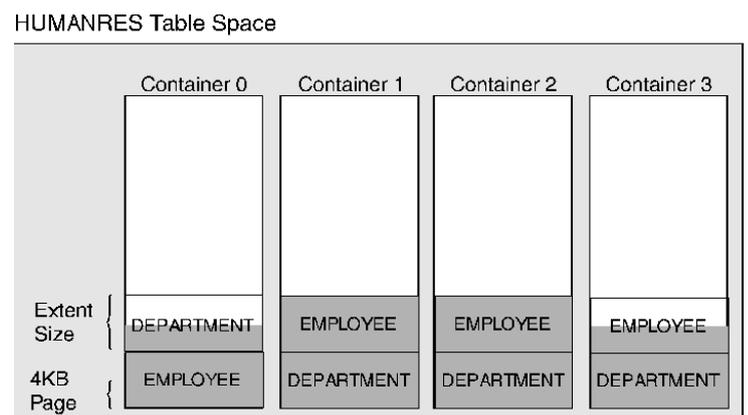
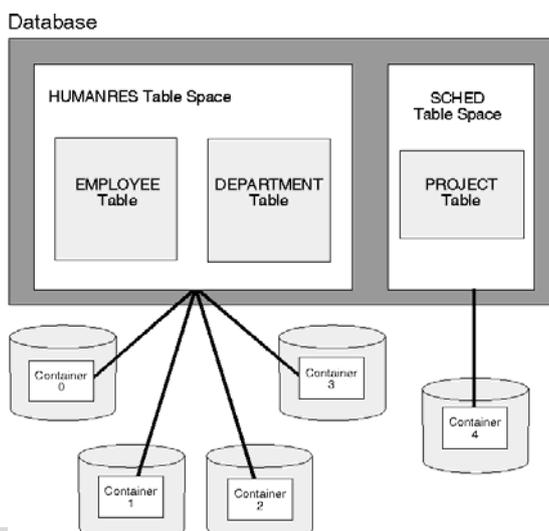


Tablespaces in DB2

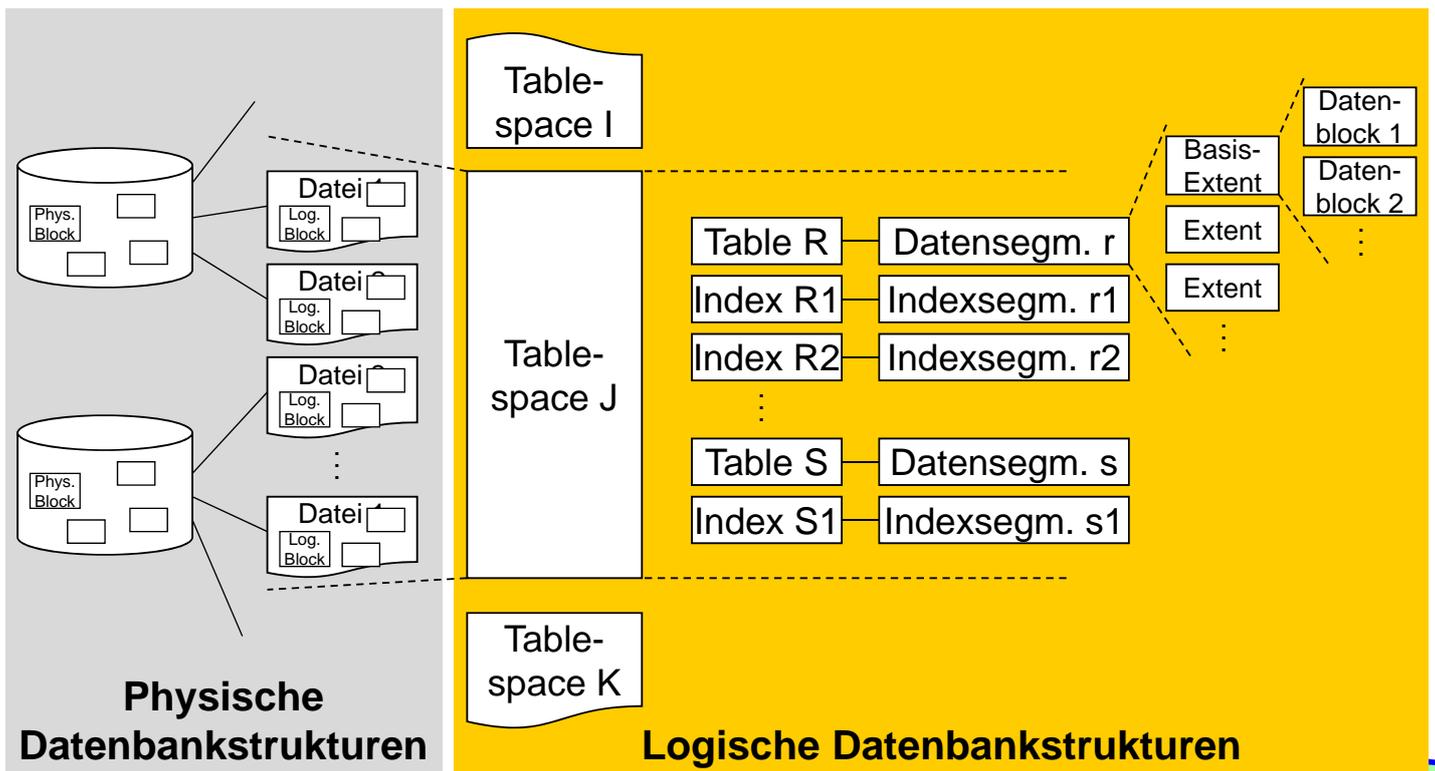
- 3 Segmentarten: Catalog, User, Temporary Tablespace

```
CREATE DATABASE PERSONL
CATALOG TABLESPACE MANAGED BY SYSTEM USING ('d:\pcatalog','e:\pcatalog')
EXTENTSIZE 32 PREFETCHSIZE 16
USER TABLESPACE MANAGED BY DATABASE USING (FILE'd:\db2data\personl' 5000,
FILE'd:\db2data\personl' 5000) EXTENTSIZE 32 PREFETCHSIZE 64
```

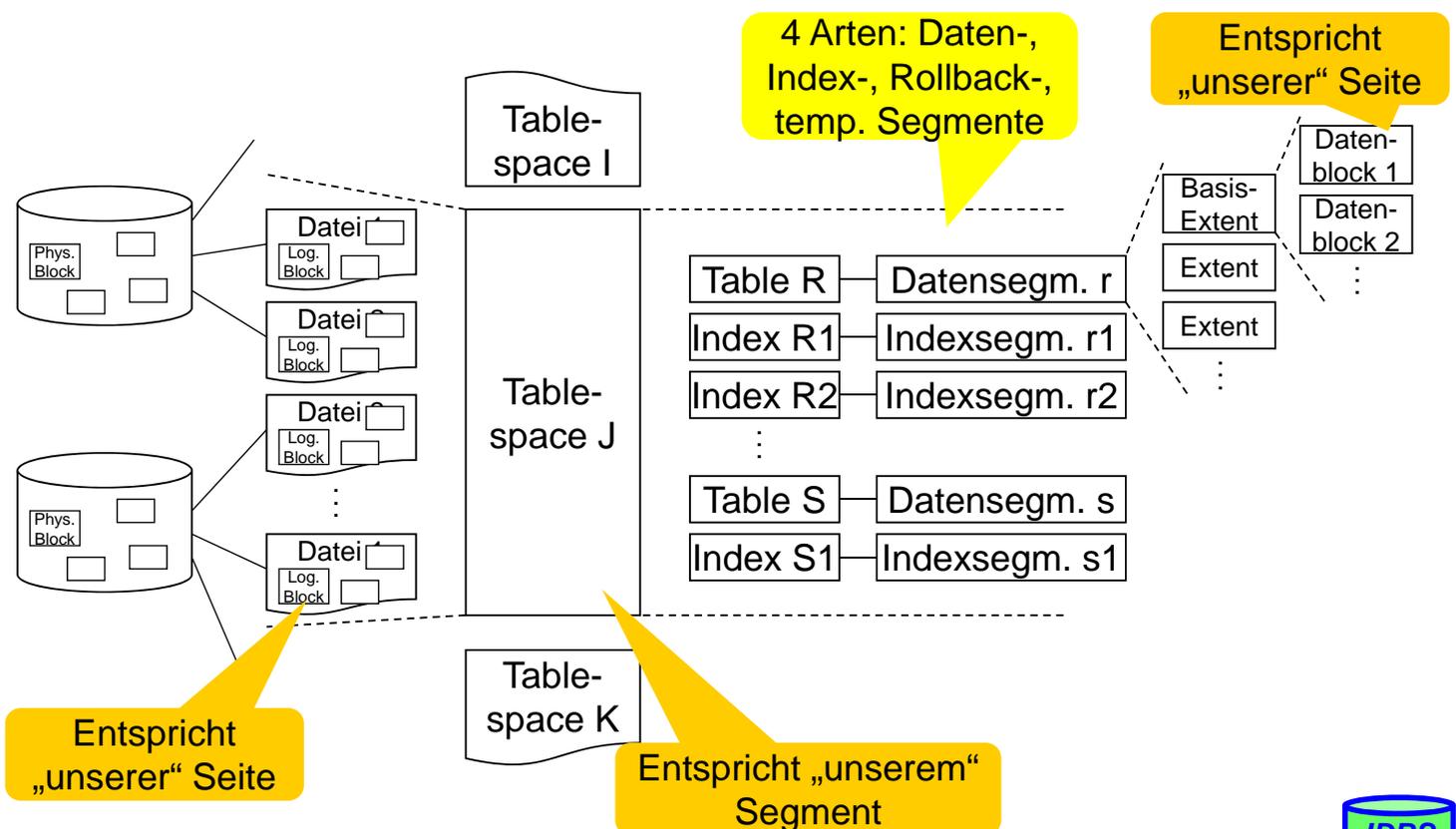
- Tablespace kann Extent-weise (Round-Robin-mäßig) auf mehrere Container/Platten aufgeteilt werden



Speicherorganisation in Oracle



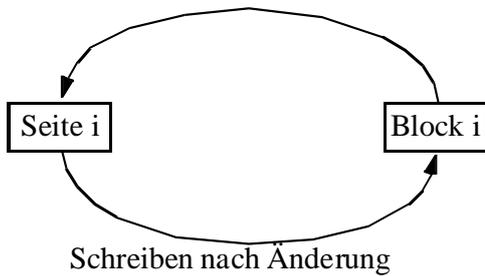
Speicherorganisation in Oracle



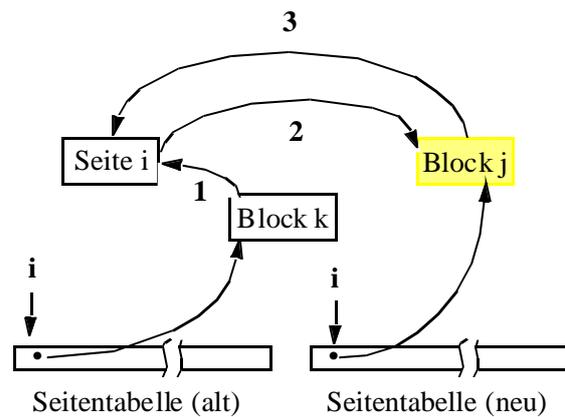
Direkte vs. indirekte Seitenzuordnung

Direkte Seitenzuordnung

(Update in Place)



Indirekte Seitenzuordnung



- 1) Lesen vor Änderung
- 2) Schreiben nach Änderung
- 3) Lesen nach Änderung

■ Indirekte Zuordnung

- Seitentabelle: Abbildung von Seitennr. → Blocknr.
- Einbringen (mengenorientiert): Änderungen schreiben + Umschalten der Seitentabellen
- Realisierungsverfahren: Schattenspeicherkonzept, Zusatzdateiverfahren (s. Lehrbuch)

Indirekte Seitenzuordnung

■ Vorteile

- Rücksetzen auf älteren konsistenten DB-Zustand (Undo-Recovery) einfach möglich
- durch Umschalten einer Seitentabelle können viele Änderungen gleichzeitig gültig gemacht („eingebracht“) werden
- physische DB kann „operationskonsistent“ gehalten werden
=> auf dem physischen DB-Zustand lassen sich DB-Operationen ausführen (ermöglicht für die Recovery ein sog. Operationen-Logging)

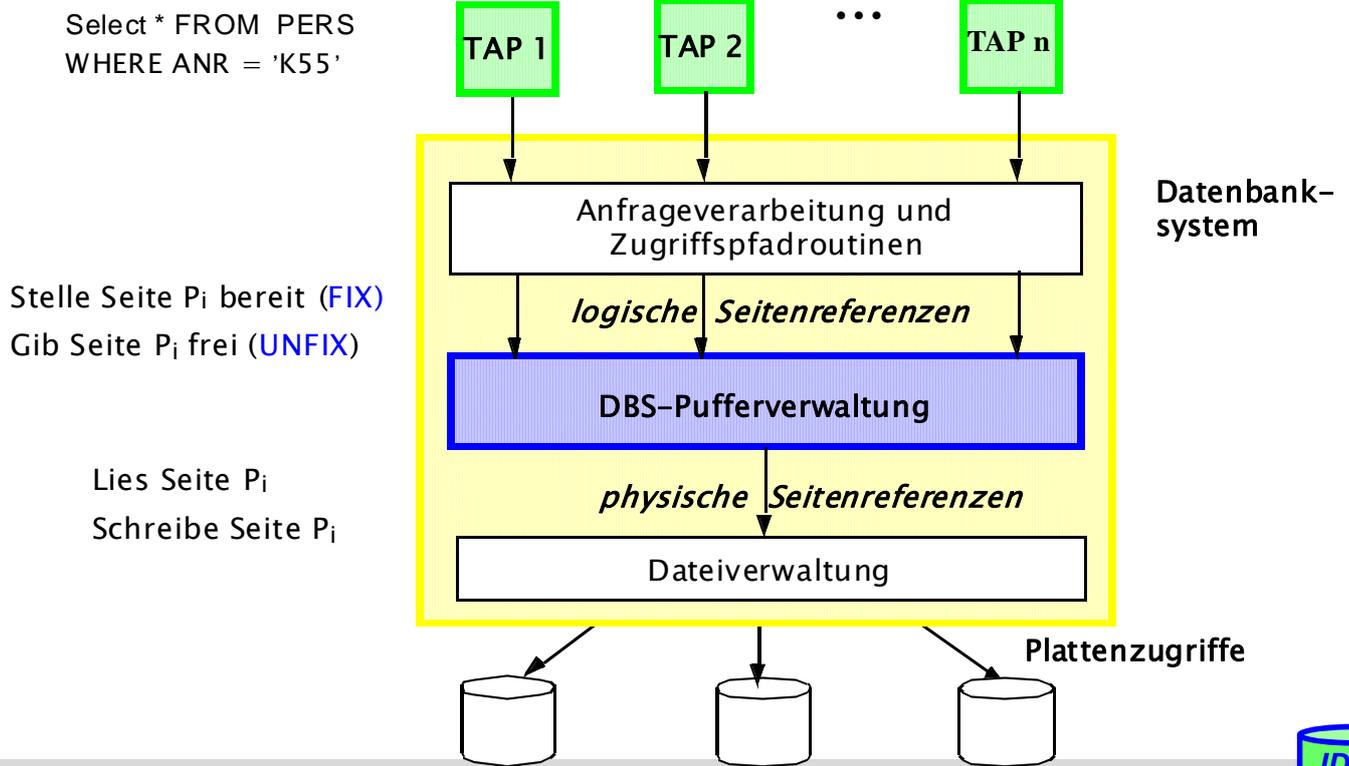
■ Nachteile

- Seitentabellen werden oft zu groß für Hauptspeicher: hohe Zugriffskosten (E/A)
- Doppelspeicherung ungünstig für lange Änderungsprogramme
- physische Clusterbildung logisch zusammengehöriger Seiten wird beeinträchtigt bzw. zerstört -> signifikante Verlangsamung für sequentielle Externspeicherzugriffe

■ In der Praxis werden daher Update-in-Place-Verfahren (direkte Seitenzuordnung) genutzt

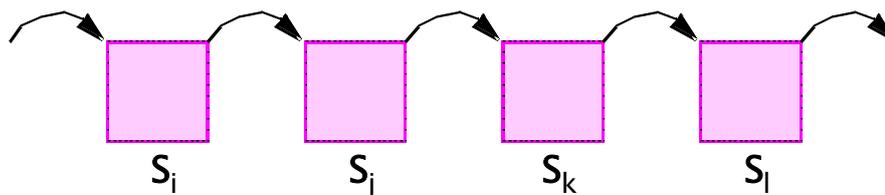
Stellung der Pufferverwaltung innerhalb eines DBS

Transaktionsprogramme, die auf die Datenbank zugreifen

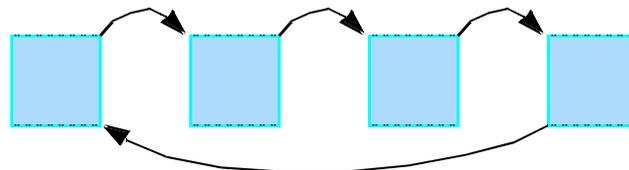


Typische Referenzmuster in DBS

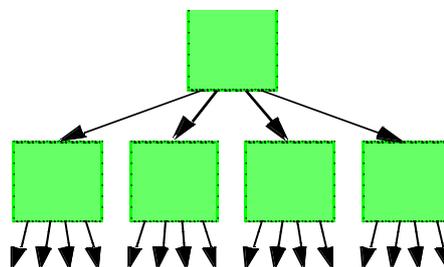
1. Sequentielle Suche (Bsp.: Relationen-Scan): räumliche Lokalität



2. Zyklische Pfade (Bsp.: wiederholte Abarbeitung von Satzmengen): temporale Lokalität



3. Hierarchische Pfade (Bsp.: Suchen über B*-Bäume)



Seitenreferenzstrings

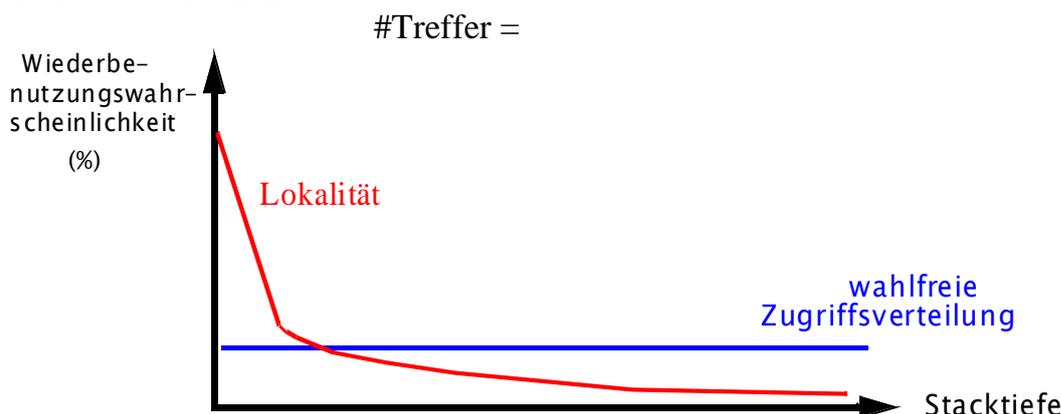
- jede Datenanforderung ist eine *logische Seitenreferenz*
- Aufgabe der Pufferverwaltung: Minimierung der *physischen Seitenreferenzen*
- **Referenzstring** $R = \langle r_1, r_2, \dots, r_i, \dots, r_n \rangle$ mit $r_i = (T_i, D_i, S_i)$

T_i zugreifende Transaktion
 D_i referenzierte DB-Partition
 S_i referenzierte DB-Seite

- Referenzstring-Information ermöglicht
 - Charakterisierung des Referenzverhaltens
 - insgesamt
 - bezüglich bestimmter Transaktionen, Transaktions-Typen und DB-Partitionen
 - Bestimmung von Lokalität und Sequentialität
 - Lokalitätsbestimmung z.B. über LRU-Stacktiefenverteilung

LRU-Stacktiefenverteilung

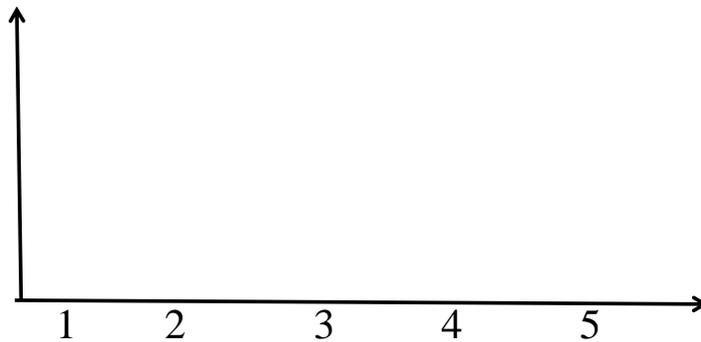
- LRU-Stack enthält bereits referenzierte Seiten in der Reihenfolge ihres Zugriffsalters
 - Unten am längsten nicht mehr referenzierte (d.h. least recently used/LRU) Seite, oben: most recently used (MRU) Seite
- Bestimmung der Stacktiefenverteilung:
 - pro Stackposition i wird Referenzzähler c_i geführt für Seiten an dieser Position
 - Zählerwerte entsprechen der Wiederbenutzungshäufigkeit
 - aus Stacktiefenverteilung läßt sich für LRU-Ersetzung und bestimmte Puffergröße x Trefferrate bestimmen



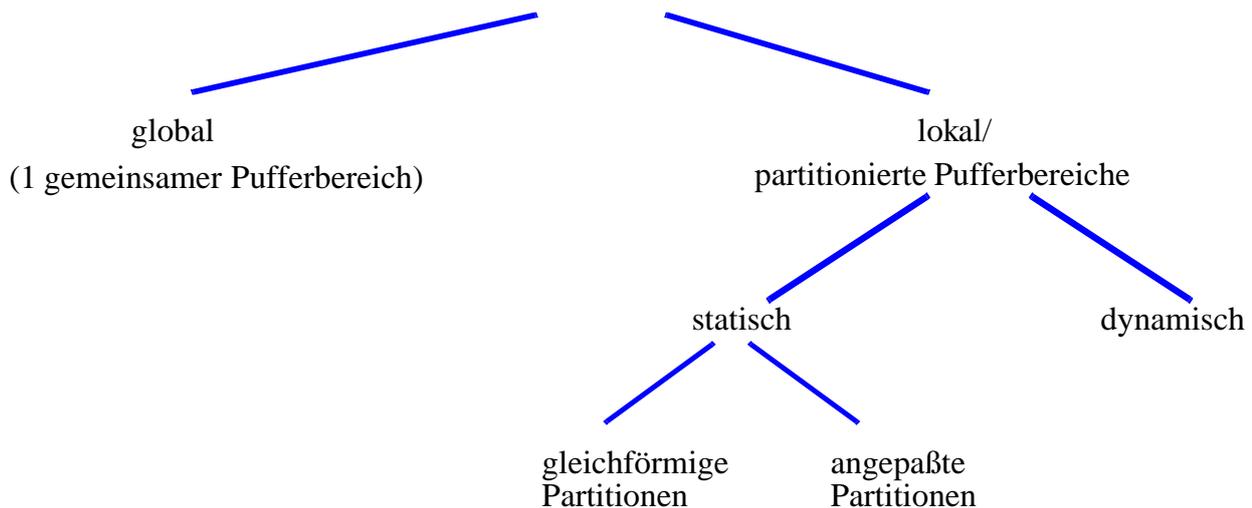
Beispiel

Referenzstring: A B A C C B C C D E

Stack-POS	Start-inhalte										
1	A										
2	B										
3	C										
4	D										
5	E										



Speicherzuteilung im DB-Puffer

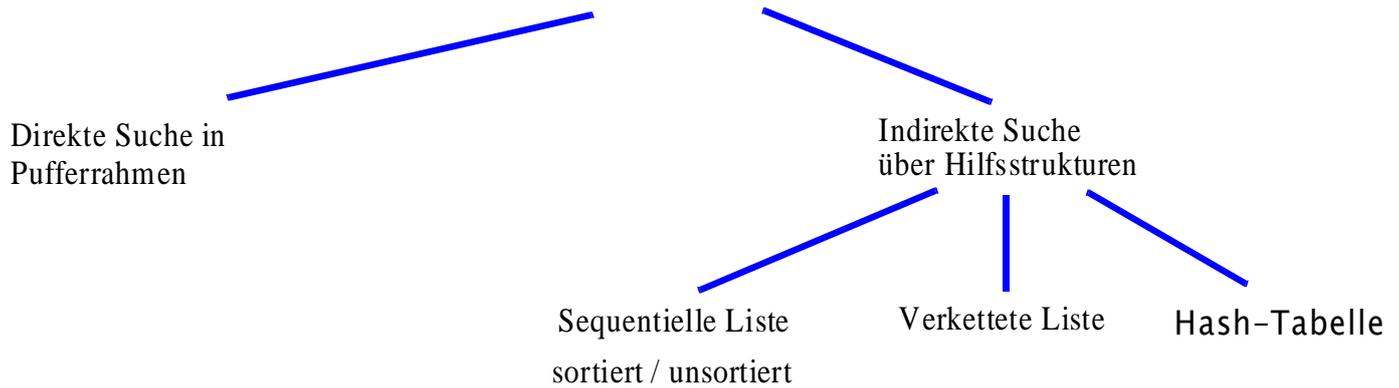


Partitionierungsmöglichkeiten:

- eigener Pufferbereich pro Transaktion bzw. Query
- Transaktionstyp-bezogene Pufferbereiche
- Seitentyp-bezogene Pufferbereiche
- DB (-Partitions)-spezifische Pufferbereiche



Suche im Puffer



■ Probleme der direkten Suche

- hohe lineare Suchkosten proportional zur Puffergröße
- hohe Wahrscheinlichkeit von Paging-I/Os

■ Listenstrukturen: lineare Suchkosten

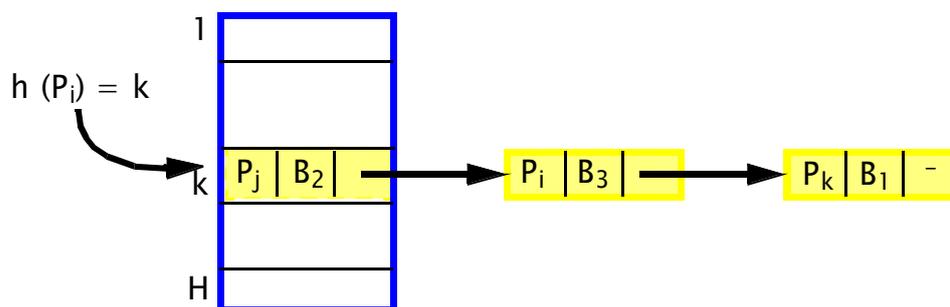
■ Beste Lösung: Hash-Tabelle mit Kollisionsbehandlung

- z.B. durch Überlaufketten

Suche (2)

■ Hash-Tabelle mit Überlaufketten

■ Infos pro Eintrag



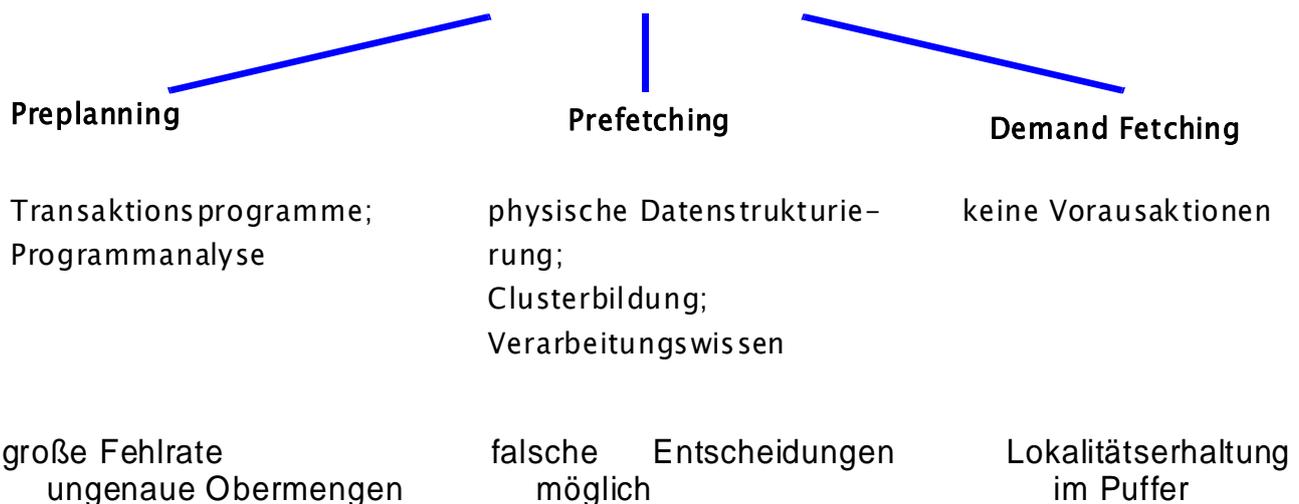
- Seiten-Nummer
- Pufferadresse
- Fix-Zähler
- Änderungsbit (bzw. Änderungszähler)
- evtl. Zeitpunkt der ersten Änderung etc.

Schreibstrategien

- Ersetzung einer geänderten Seite erfordert vorheriges Zurückschreiben der Änderung in permanente DB auf Externspeicher
 - synchrones (=> Antwortzeitverschlechterung) vs. asynchrones Ausschreiben
 - Abhängigkeit zur gewählten Ausschreibstrategie (Force vs. Noforce)
- **FORCE**: alle Änderungen einer Transaktion werden spätestens beim Commit in die DB zurückgeschrieben ("write-through")
 - i.a. stets ungeänderte Seiten zur Ersetzung vorhanden
 - hoher E/A-Overhead
 - starke Antwortzeiterhöhung für Änderungstransaktionen
- **NOFORCE**: kein Durchschreiben der Änderungen bei Commit (verzögertes Ausschreiben, "deferred write-back")
 - Seite kann mehrfach geändert werden, bevor ein Ausschreiben erfolgt (geringerer E/A-Overhead, bessere Antwortzeiten)
 - vorausschauendes (asynchrones) Ausschreiben geänderter Seiten erlaubt auch bei NOFORCE, vorwiegend ungeänderte Seiten zu ersetzen
 - synchrone Schreibvorgänge in die DB können weitgehend vermieden werden

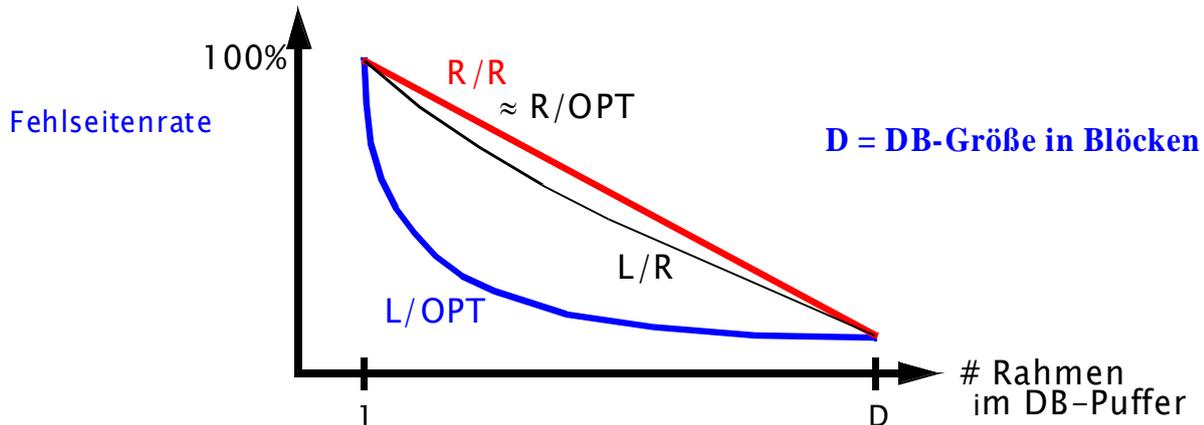


Lesestrategien



Referenzverhalten und Ersetzungsverfahren

- Grundannahme bei Ersetzungsverfahren
 - Referenzverhalten der jüngeren Vergangenheit ähnelt Referenzverhalten in der näheren Zukunft
 - Nutzung der typischerweise hohen Lokalität bei Ersetzung
- **manchmal** Sequentialität oder zufällige Arbeitslast (RANDOM-Referenzen)
- Kombinationen bzgl. Referenzen/Ersetzung: RANDOM/RANDOM, RANDOM/OPT, Lokalität/RANDOM, Lokalität/OPT
- Grenzfälle zeigen Optimierungsmöglichkeiten auf



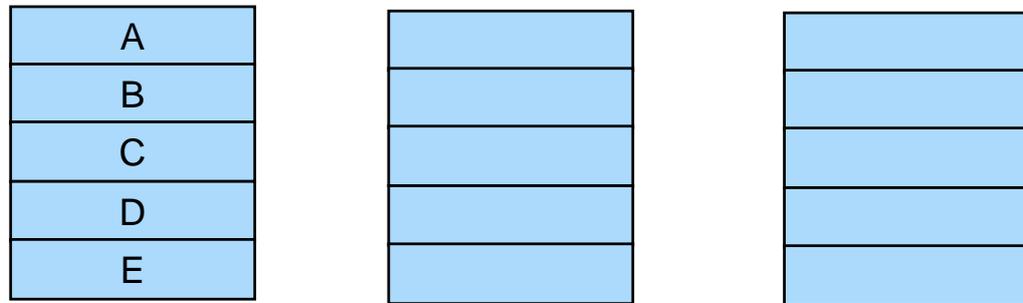
Kriterien für die Auswahl der zu ersetzenden Pufferseite

Verfahren	Alter	Letzte Referenz	Referenzhäufigkeit	Andere Kriterien
OPT	-	-	-	Vorauswissen
RANDOM	-	-	-	-
LRU				
LFU				
FIFO				
CLOCK				
GCLOCK				
LRD V1				
LRD V2				
LRU-K				
Adaptives LRU				



Least-Recently-Used (LRU)

- Ersetzungskriterium: Zeit seit der letzten Referenzierung der Seite.
- Eine referenzierte Seite kommt an die Spitze des LRU-Stacks
 - Falls referenzierte Seite an Position n im Stack war, rutschen alle Seiten an den Positionen 1 bis $n-1$ eine Position tiefer
- Seite am Kellerboden wird ersetzt
- Beispiel (Stackgröße 5): Referenzen von Seiten C und F



- Unterscheidung zwischen *Least-Recently-Referenced* und *Least-Recently-Unfixed*



Least-Frequently-Used (LFU)

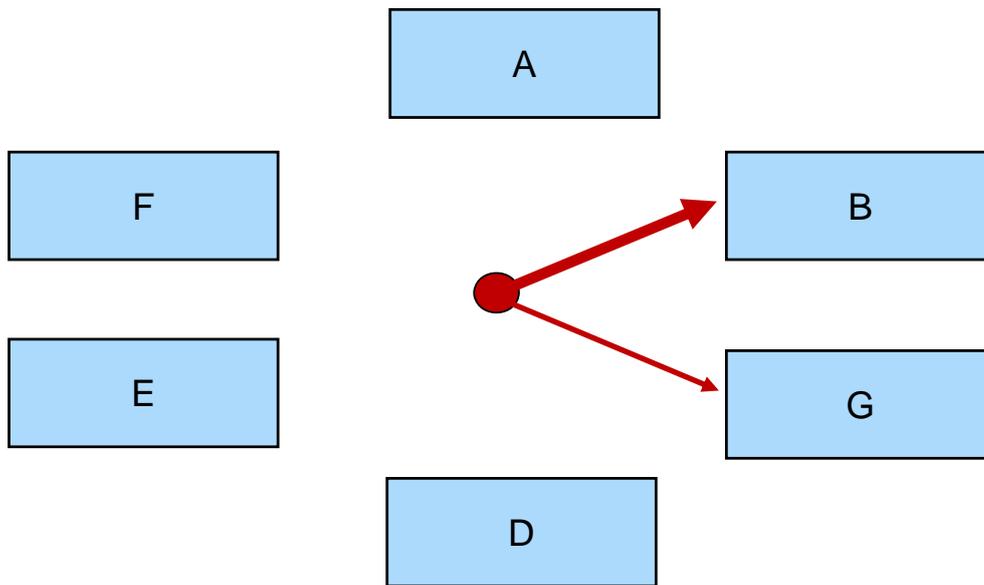
- Führen eines Referenzzählers pro Seite im Puffer
- Ersetzung der Seite mit der geringsten Referenzhäufigkeit

RZ	
2	
4	
1	
3	
3	
6	
1	
3	

- Nachteil: Alter (Zeit seit letzter Einlagerung) einer Seite wird nicht berücksichtigt
 - Seiten mit kurzzeitiger, sehr hoher Referenzierung sind kaum mehr zu verdrängen

FIFO (First-In First-Out)

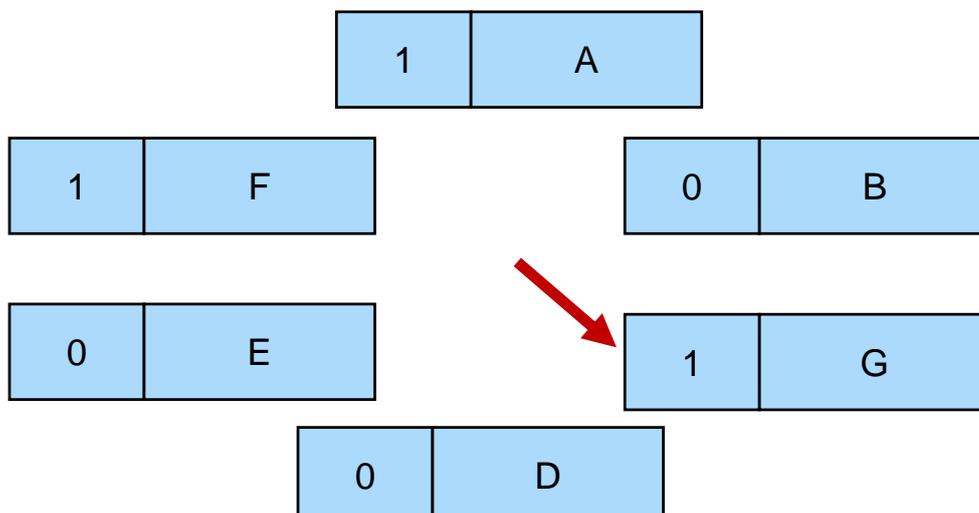
- die älteste Seite im Puffer wird ersetzt



- Referenzierungsverhalten während Pufferaufenthaltes wird nicht berücksichtigt
 - auch häufig referenzierte Seiten werden ersetzt

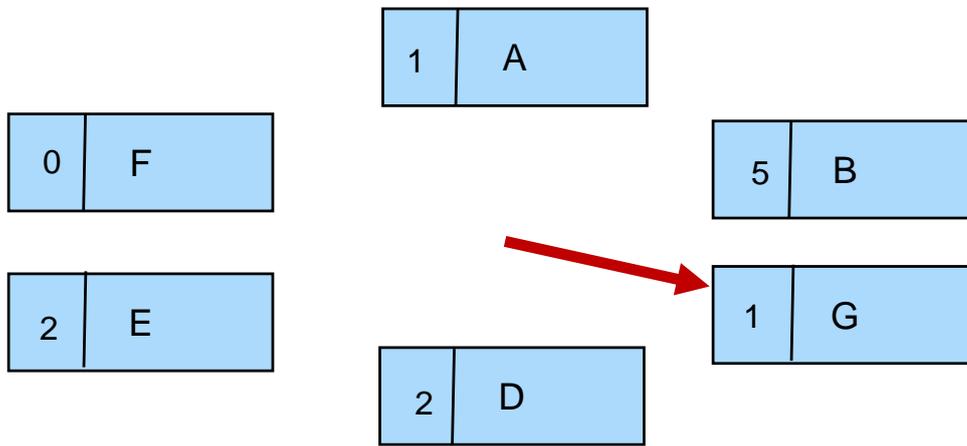
CLOCK (Second Chance)

- Erweiterung von FIFO
- Referenzbit pro Seite, das bei Zugriff gesetzt wird
- Ersetzung erfolgt nur bei zurückgesetztem Bit (sonst erfolgt Zurücksetzen des Bits)



- annähernde Berücksichtigung des letzten Referenzierungszeitpunktes

GCLOCK (Generalized CLOCK)



- pro Seite wird Referenzzähler geführt (statt Bit)
- Ersetzung nur von Seiten mit Zählerwert 0 (sonst erfolgt Dekrementierung des Zählers und Betrachtung der nächsten Seite)
- Verfahrensparameter:
 - Initialwerte für Referenzzähler
 - Wahl des Dekrementes
 - Zählerinkrementierung bei erneuter Referenz
 - Vergabe von seitentyp- oder seitenspezifischen Gewichten



Least-Reference-Density (LRD)

- Referenzdichte: Referenzhäufigkeit während eines bestimmten Referenzintervalls
- LRD Variante 1: Referenzintervall entspricht Alter einer Seite
- Berechnung der Referenzdichte:
 - Globaler Zähler GZ: Gesamtanzahl aller Referenzen
 - Einlagerungszeitpunkt EZ: GZ-Wert bei Einlesen der Seite
 - Referenzzähler RZ

Referenzdichte $RD(j) = \frac{RZ(j)}{GZ - EZ(j)}$



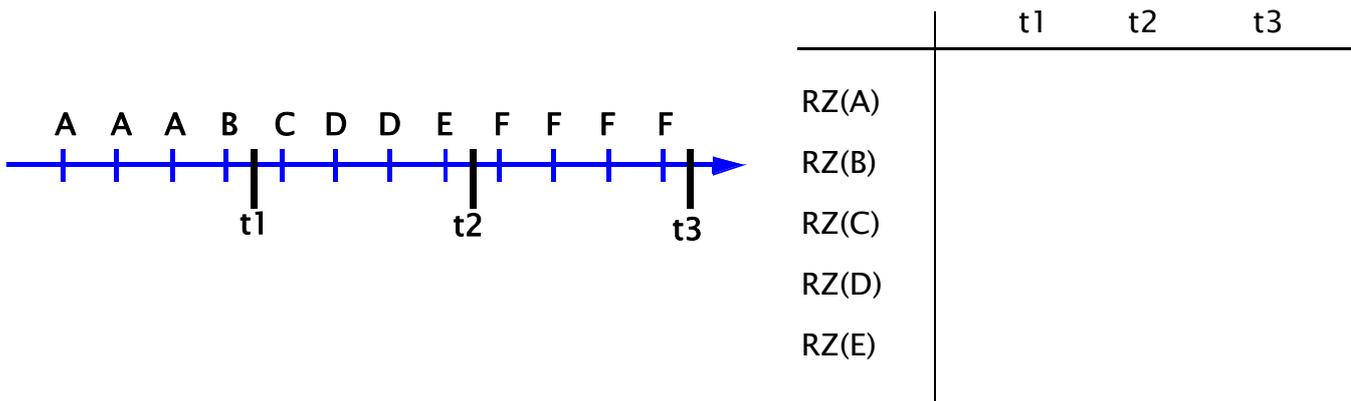
	RZ	EZ	RD
A			
B			
C			
D			
E			
F			



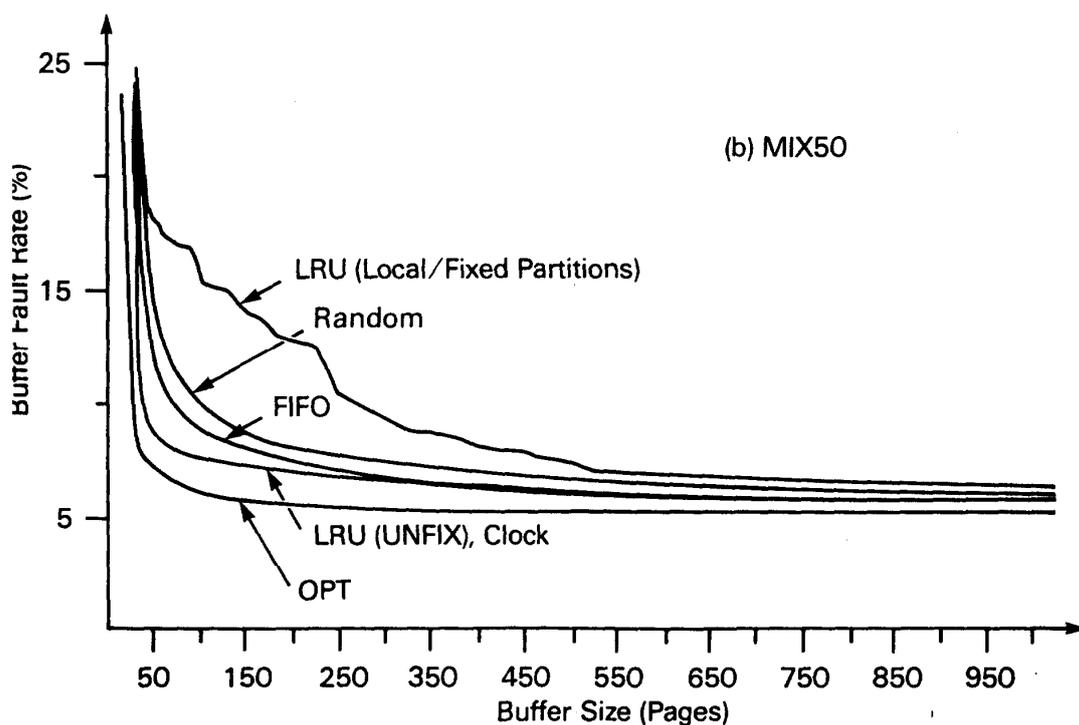
Least-Reference-Density (2)

- LRD Variante 2: konstante Intervallgröße
- periodisches Reduzieren der Referenzzähler, um Gewicht früher Referenzen zu reduzieren
 - Reduzierung von RZ durch Division oder Subtraktion:

$$RZ(i) = \frac{RZ(i)}{K1} \quad (K1 > 1) \text{ oder } RZ(i) = \begin{cases} RZ(i) - K2 & \text{falls } RZ(i) - K2 \geq K3 \\ K3 & \text{sonst} \end{cases} \quad (K2 > 0, K3 \geq 0)$$



Simulationsergebnisse

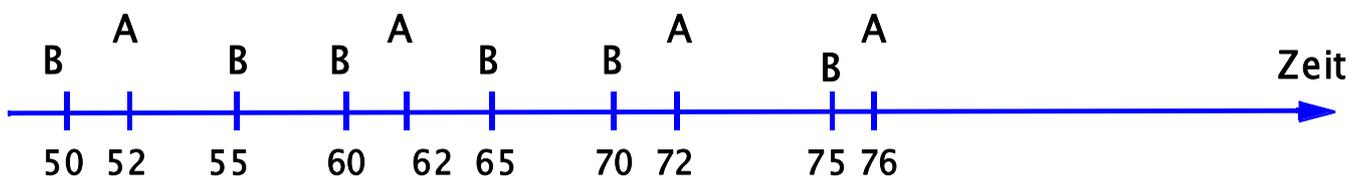


Probleme von LRU

- LRU ungeeignet für sequentielle Zugriffsmuster (z.B. Relationen-Scan)
 - sofortige Ersetzung sinnvoll (z.B. *Most Recently Used, MRU*)
- LRU nutzt kein Wissen über spezielle Referenzfolgen, z.B. Referenzzyklen
 - zyklisches Referenzieren von S Seiten mit $S > \#Rahmen$ → internes Thrashing
 - zyklisches Referenzieren von S Seiten mit $S < \#Rahmen$ und Interferenz durch andere Transaktionen mit schnellerer Anforderung (stealing) → externes Thrashing
- LRU berücksichtigt nicht unterschiedliche Referenzhäufigkeiten, z.B. zwischen Index- und Datenseiten
- LRU berücksichtigt nicht Charakteristika unterschiedlicher Anfrage-/Transaktionstypen
 - Bsp.: Transaktionen mit hoher Referenzlokalität können durch gleichzeitige sequentielle Scans mit schneller Seitenanforderung stark benachteiligt werden
- Alternativen
 - Ausnutzen von Kontextwissen des Query-Optimierers ("hints" an Pufferverwaltung)
 - LRU-Erweiterungen bzgl. Prioritäten, Referenzhäufigkeiten etc.

LRU-K

- Berücksichtigung der K letzten Referenzzeitpunkte einer Seite
 - erlaubt Approximation der Referenzhäufigkeit durch Bestimmung des mittleren Zeitabstands zwischen Referenzen einer Seite
 - Beschränkung auf die K letzten Referenzen ist einfache Methode, Information aktuell zu erhalten (keine zusätzlichen Tuning-Parameter wie bei LRU V2)
- Beispiel ($K=4$)



- zur Ersetzung genügt es, K -letzten Referenzierungszeitpunkt zu berücksichtigen!
- LRU-2 (d.h. $K=2$) stellt i.a. beste Lösung dar
 - ähnlich gute Ergebnisse wie für $K > 2$, jedoch einfachere Realisierung
 - bessere Reagibilität auf Referenzschwankungen als für größere K

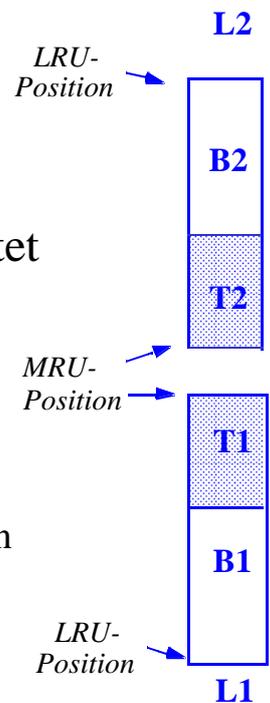
Adaptives LRU*

■ Verwendung von 2 LRU-Ketten

- L1: Seiten die nur 1-mal referenziert wurden (v.a. für sequentielle Zugriffe)
- L2: Seiten, die wenigstens 2-mal referenziert wurden

■ für Cache-Größe c werden 2c Seiten in L1 und L2 verwaltet

- jede Liste verwaltet in einem Top-Bereich (T1 bzw. T2) die gepufferten Seiten sowie
 - in einem Bottom-Bereich (B1 bzw. B2) die nicht mehr gepufferten Seiten
- $|T1|+|T2| \leq c$; $|L1|+|L2| = |T1|+|T2|+|B1|+|B2| \leq 2c$; $|L1| \leq c$
- falls angeforderte Seite in L1 oder L2 wird sie an MRU-Position von L2 gebracht, ansonsten an MRU-Position von L1



■ Parameter $p = |T1|$ bestimmt relatives Verhältnis zwischen L1- und L2-Seiten im Puffer

■ dynamische Anpassung von p gemäß aktueller Verteilung von sequentiellen Zugriffen und Zugriffen mit temporaler Lokalität

* N. Megiddo, D.S. Modha: *Outperforming LRU with an Adaptive Replacement Cache Algorithm*. IEEE Computer, April 2004



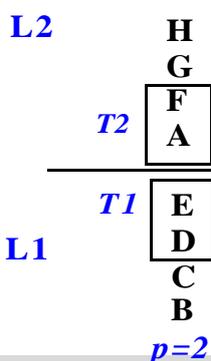
Adaptives LRU (2)

■ Ersetzung für Referenz auf Seite x

- 1: Hit in T1 oder T2: Bringe x an MRU-Position von T2
 - 2: "Hit" in B1: $p := \min(p+k1, c)$ mit $k1 = 1$ falls $|B1| \geq |B2|$, sonst $k1 = |B2|/|B1|$; **Erhöhe p** ($|T1|$)
REPLACE(x,p); lese x ein und bringe x an MRU-Position von T2
 - 3: "Hit" in B2: $p := \max(p-k2, 0)$ mit $k2 = 1$ falls $|B2| \geq |B1|$, sonst $k2 = |B1|/|B2|$;
Reduziere p (erhöhe $|T2|$)
REPLACE(x,p); lese x ein und bringe x an MRU-Position von T2
 - 4: Miss: REPLACE (x,p); lese x ein und bringe x an MRU-Position von T1
- REPLACE (x,p):** IF $|T1| > 0$ and $(p < |T1|$ OR $(p = |T1|$ and $(x$ in B2))
 Ersetze LRU-Seite von T1 (-> Wechsel an MRU-Position von B1)
 ELSE Ersetze LRU-Seite von T2 (-> Wechsel an MRU-Position von B2)

Beispiel (c=4)

G C A K F Referenzen



Adaptives LRU: Beobachtungen

- Adaptivität über erweiterte Listen
- Heuristik: investiere in erfolgreiche Liste (vergrößere T1 bzw. T2 für Treffer in B1 bzw. B2)
- rein sequentielle Zugriffe füllen L1; fehlende Treffer in B1 vergrößern T2



Zusammenfassung

- effizientes und flexibles Dateikonzept erforderlich
 - Blockzuordnung über dynamische Extents für Wachstum und schnelle Adressierung
 - direkter und sequentieller Blockzugriff
- Segmentkonzept: Logische Behälter für DB-Inhalte, Indexstrukturen
- Update-in-Place ist indirekten Einbringstrategien vorzuziehen
- DB-Pufferverwaltung zur Minimierung physischer E/A
 - Nutzung von Lokalität innerhalb und zwischen Transaktionen, Sequentialität, ...
 - Suche im Puffer: durch Hash-Verfahren
 - Globale Speicherzuteilung vorteilhaft (Pufferrahmen für alle Transaktionen)
 - Behandlung geänderter Seiten: NOFORCE, asynchrones Ausschreiben
- Seiteneretzungsverfahren
 - Nutzung mehrerer Kriterien möglich: Alter, letzte Referenz, Referenzhäufigkeit
 - LRU ist guter Default-Ansatz
 - LRU-2 wählt Ersetzungskandidaten aufgrund des vorletzten Referenzzeitpunktes aus
 - Adaptives LRU

