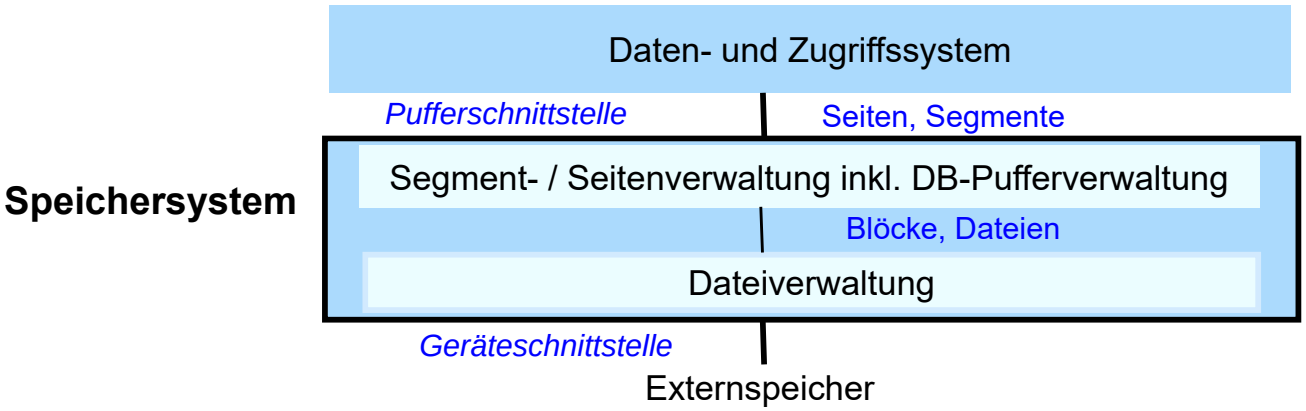
# 3. Speichersystem / Pufferverwaltung

- Dateiverwaltung / Segmente
- Direkte vs. indirekte Seitenzuordnung
- DB-Pufferverwaltung: Grundlagen
  - Referenzverhalten, LRU-Stacktiefenverteilung
  - 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

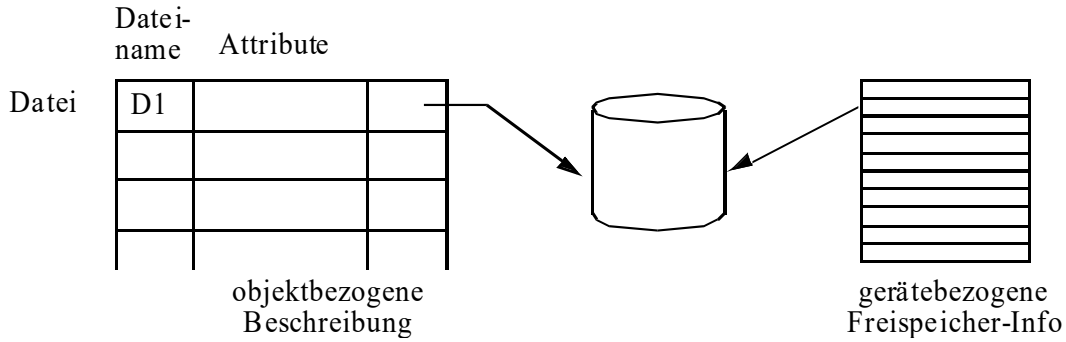


# Speichersystem

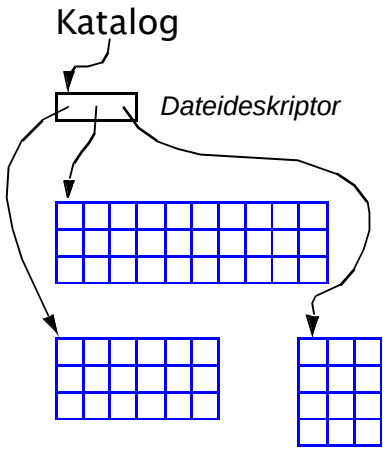



- Dateiverwaltung (Teil des Betriebssystems)
  - DB-Speicher = Menge von Dateien
  - Dateien repräsentieren externe Speichermedien in geräteunabhängiger Weise
  - Dateisystem: Abbildung von Dateien / physischen Blöcken auf Externspeicher
  - wichtig: schnelle Adressierbarkeit, dynamische Wachstumsfähigkeit von Dateien
- Segment- und Seitenverwaltung (im DBS)
  - ermöglicht indirekte Einbringstrategien (unterschiedliche Blöcke pro Seite)

# 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 sequenzielle Zugriffe innerhalb von Extents
  - Flexibilität hinsichtlich Wachstum






# Segmentkonzept

- DB besteht aus mehreren Segmenten mit Seiten
  - Abbildung auf Dateien
  - unterschiedliche Segmentarten: permanent/temporär, öffentlich/privat,
  - Segmente: Einheiten des Sperrens, der Recovery und der Zugriffskontrolle
- 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

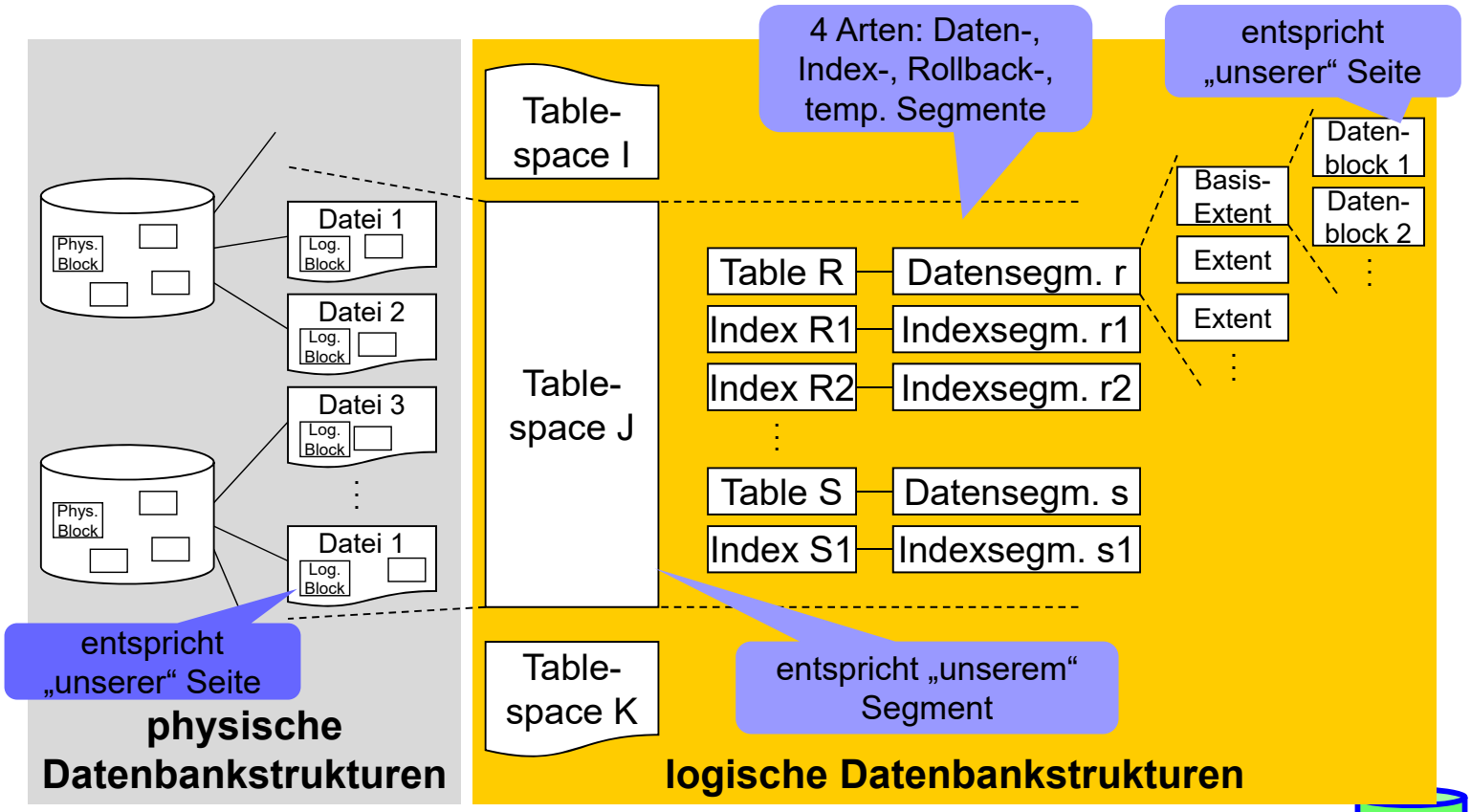
  CREATE TABLESPACE tablespacename
  DATAFILE filename SIZE size { "," filename SIZE size }

- Zuordnung von Relationen zu Tablespaces

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

  - PCTFREE: Prozentsatz der in Seite initial freibleibt

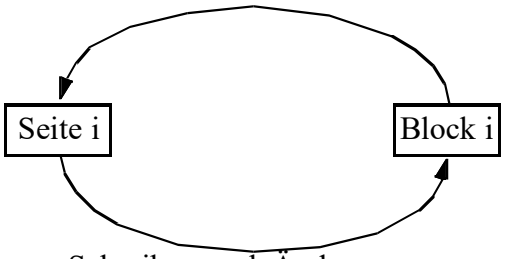
# Speicherorganisation in Oracle



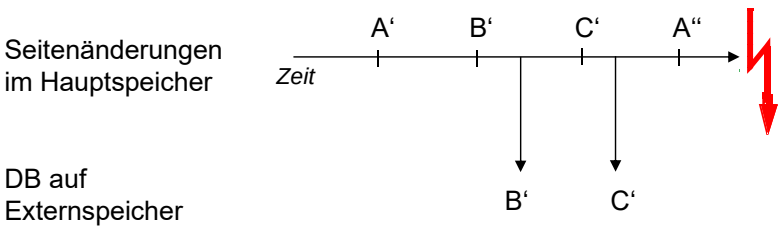



# Direkte vs. indirekte Seitenzuordnung (1)

- direkte Seitenzuordnung (Update in Place)
  - geänderte Seite wird in den Block auf Externspeicher zurückgeschrieben, von wo sie gelesen wurde
  - Clusterung von Seiten/Böcken bleibt erhalten

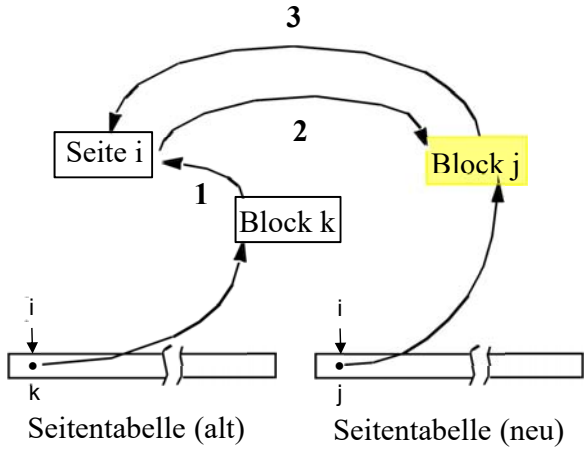



- nach DBS-Absturz kann DB-Zustand auf Externspeicher inkonsistent sein
  - erhöhter Recoveryaufwand (s Vorl. IDBS2)



# Direkte vs. indirekte Seitenzuordnung (2)

- indirekte Seitenzuordnung
  - nach (erster) Änderung wird Seite in neuen Block auf Externspeicher zurückgeschrieben
  - ungeänderte Versionen noch verfügbar (vereinfacht Undo-Recovery)
  - Seitentabelle: Abbildung von Seitennr. → Blocknr.




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

- gelegentliches Umschalten der Seitentabellen
  - alle Änderungen werden ausgeschrieben, so dass Konsistenz erreicht wird
  - Umschalten der Seitentabelle ermöglicht „Einbringen“ aller Änderungen
  - Realisierungsverfahren: Schattenspeicherkonzept, Zusatzdateiverfahren (s. Lehrbuch)



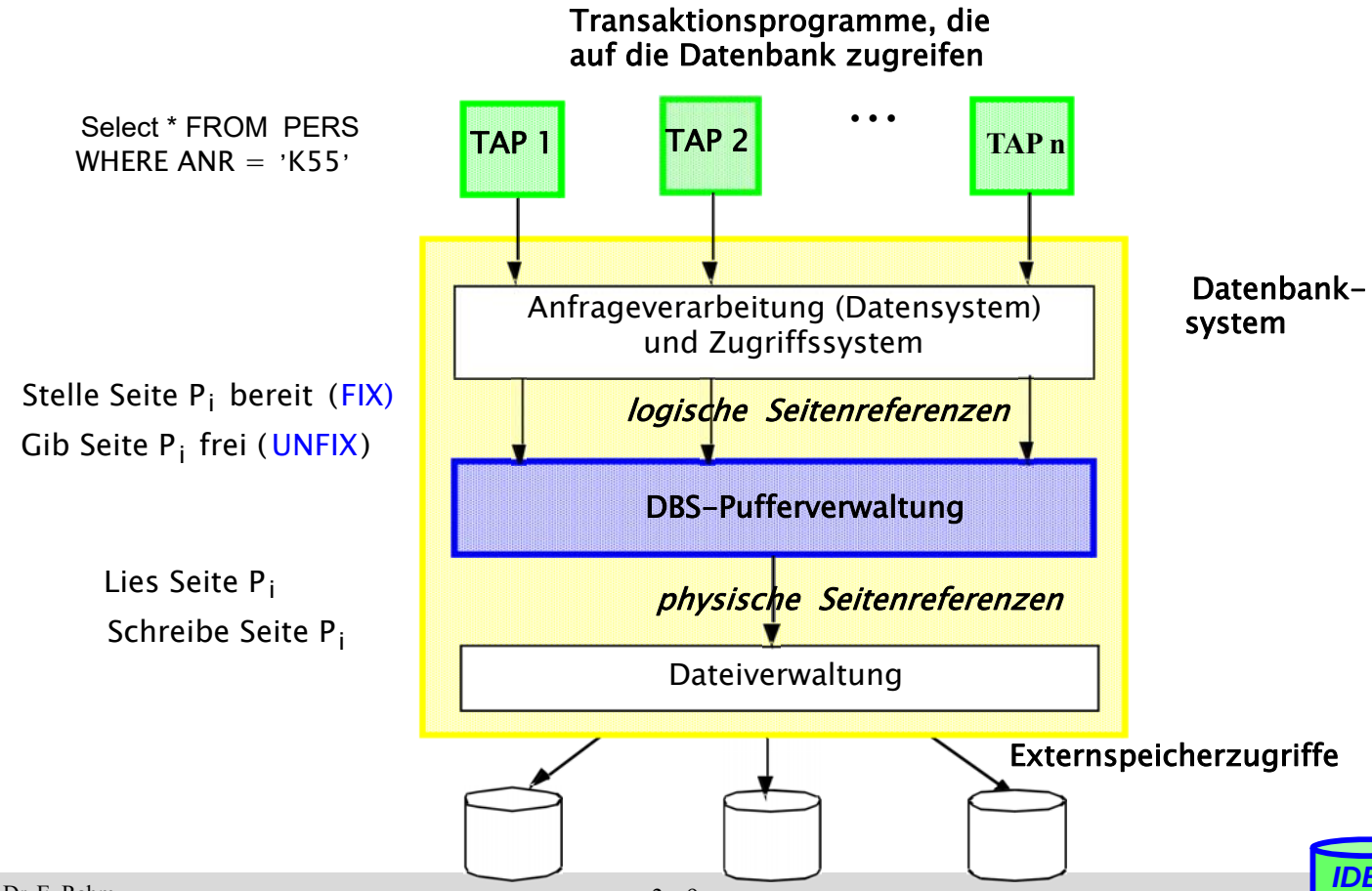
# Indirekte Seitenzuordnung

- 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

- 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 sequenzielle Externspeicherzugriffe

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

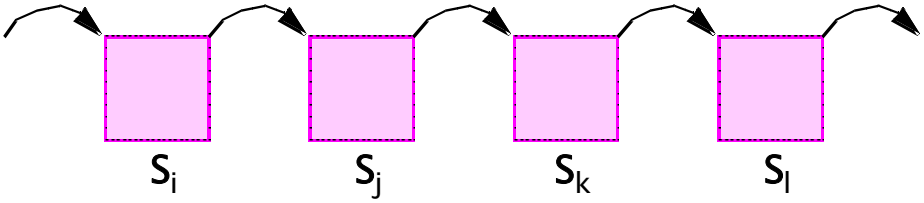
# Stellung der Pufferverwaltung innerhalb eines DBS



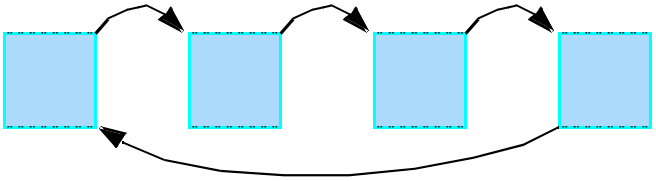



# Typische Referenzmuster in DBS

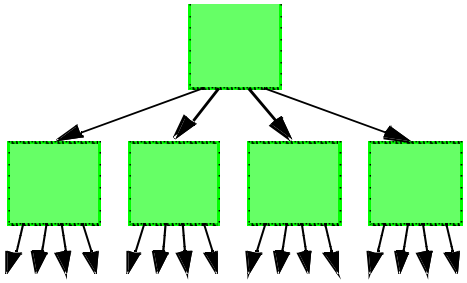
1. sequenzielle Suche (Bsp.: Relationen-Scan): räumliche Lokalität




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 $R = <r_1, r_2, \ldots r_i, \ldots r_n>$ mit $r_i = (T_i, D_i, S_i)$

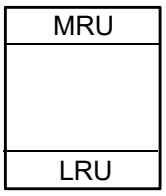
  $T_i$ zugreifende Transaktion
  $D_i$ referenzierte DB-Partition
  $S_i$ referenzierte DB-Seite

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

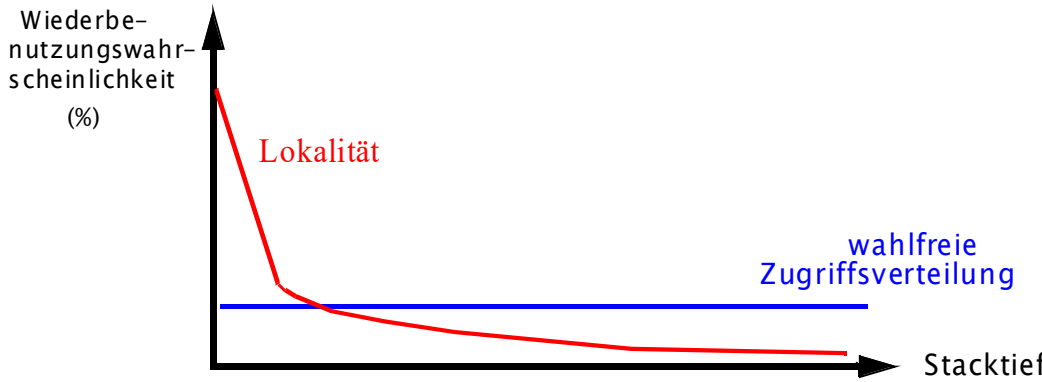


# LRU-Stacktiefenverteilung

- 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



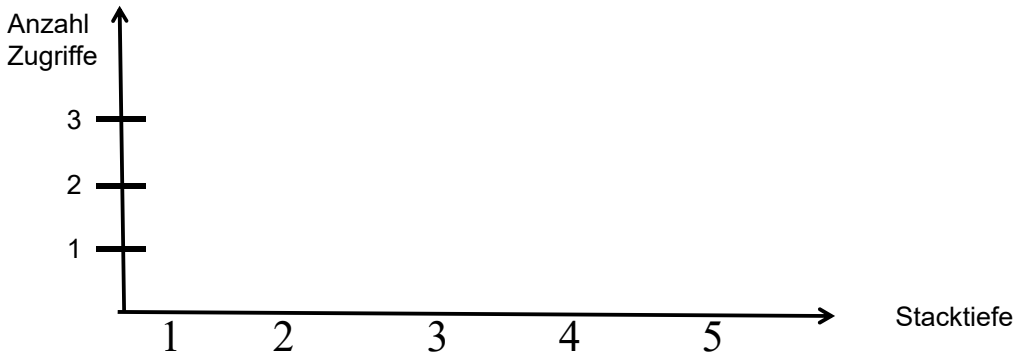

- Bestimmung der Stacktiefenverteilung:
  - pro Stackposition i wird Referenzzähler $c_i$ geführt für Seiten an dieser Position
  - Zählerwerte entsprechen der Wiederbenutzungshäufigkeit
  - Stacktiefenverteilung ermöglicht Bestimmung der Trefferrate für bestimmte Puffergröße x (bei LRU-Ersetzung): $\#\text{Treffer} = \sum_{i=1}^{x} c_i$



# Beispiel

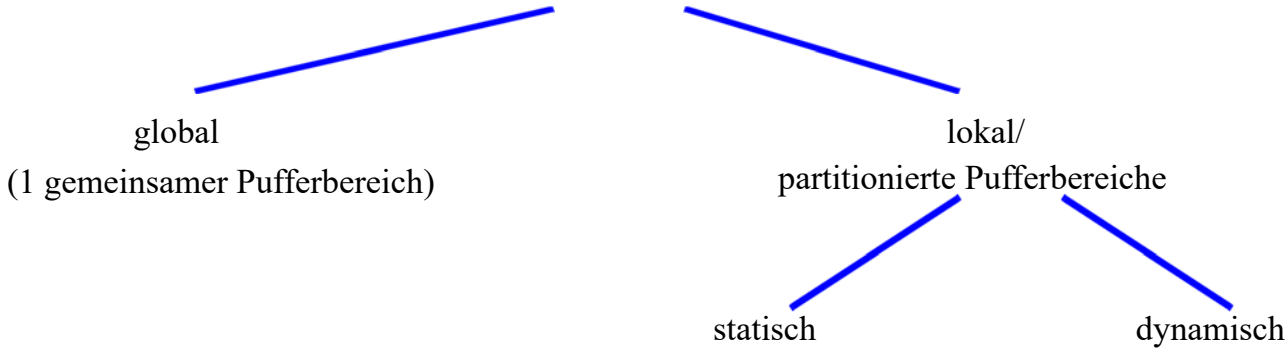
Referenzstring: A B A C C B C C D E

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





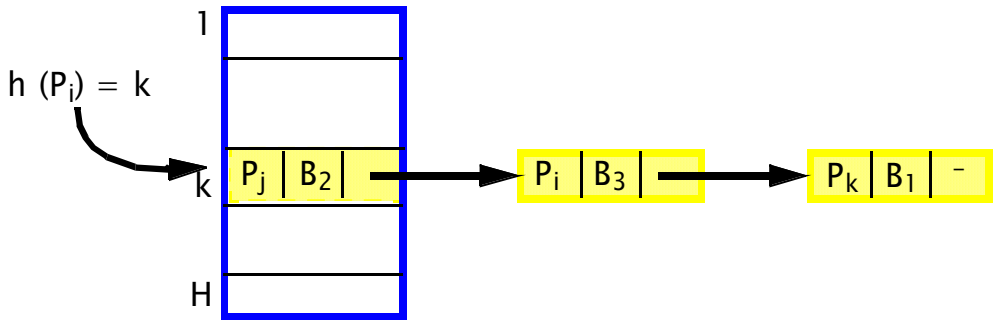

# Speicherzuteilung im DB-Puffer




- globaler Puffer ist meist am besten
  - Nutzung von Lokalität im Referenzverhalten mehrerer Transaktionen
- Partitionierungsmöglichkeiten für lokale Pufferbereiche
  - eigener Pufferbereich pro Transaktion bzw. Query
  - Pufferbereiche pro DB-Partition, Transaktionstyp etc.

# Suche im Puffer

- schnelle Feststellung erforderlich, ob Seite $P_i$ bereits im Puffer und wenn ja an welcher Adresse
- beste Lösung: Hash-Tabelle mit Überlaufketten




- Infos pro Eintrag
  - Seitennummer
  - Pufferadresse
  - Fix-Zähler (Wert > 0 verhindert Ersetzung)
  - Änderungsbit bzw. Änderungszähler
  - evtl. Zeitpunkt der ersten Änderung etc.



# Schreibstrategien

- Ersetzung einer geänderten Seite erfordert vorheriges Zurückschreiben der Änderung in permanente DB auf Externspeicher
- möglichst vorausschauendes, asynchrones Ausschreiben, damit bei Ersetzung nicht auf Abschluss des Schreibvorgangs gewartet werden muss
- geänderte DB-Seiten bleiben üblicherweise im DB-Puffer nach Commit der ändernden Transaktion (verzögertes Ausschreiben, NOFORCE-Strategie)
  - Seite kann mehrfach geändert werden, bevor Ausschreiben erfolgt (geringerer E/A-Aufwand)
- Durchschreiben aller Transaktionsänderungen bei Commit (FORCE) durchgeschrieben werden (FORCE) führt zu hoher Antwortzeitverlängerung und hohem E/A-Aufwand

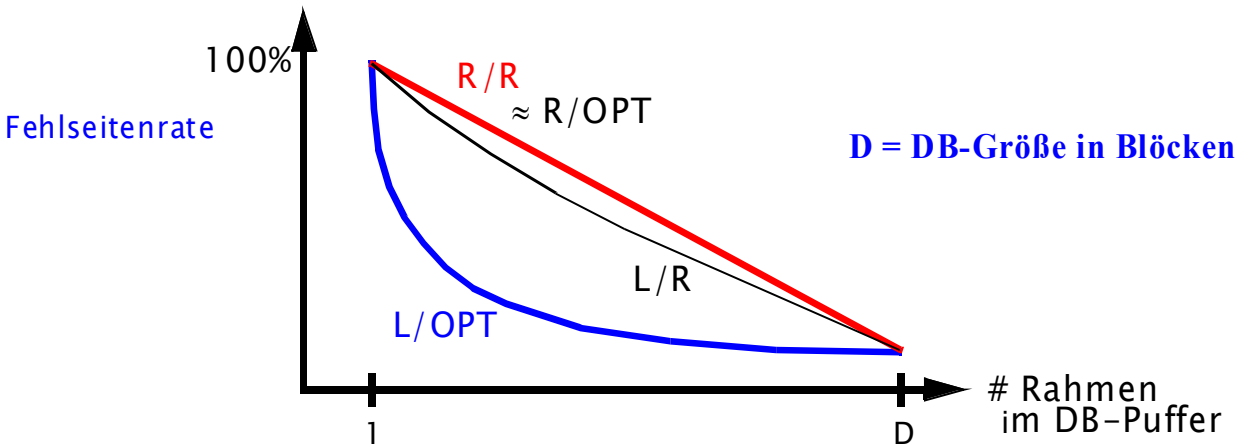
# Lesestrategien

| Preplanning | Prefetching | Demand Fetching |
|---|---|---|
| Transaktionsprogramme; Programmanalyse | physische Datenstrukturierung; Clusterbildung; Verarbeitungswissen | keine Vorausaktionen |
| große Fehlrate ungenaue Obermengen | falsche Entscheidungen möglich | Lokalitätserhaltung im Puffer |



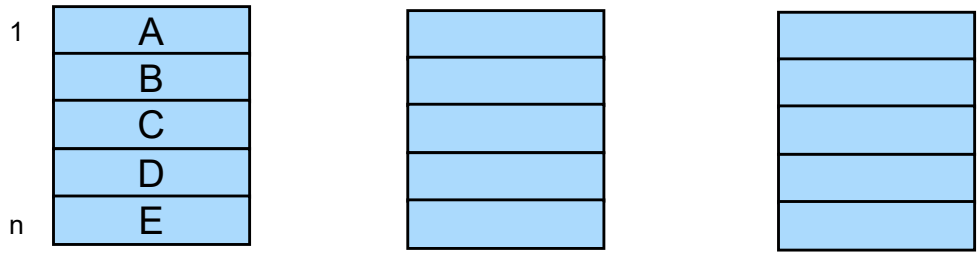
# Referenzverhalten und Ersetzungsverfahren

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



# Least-Recently-Used (LRU)

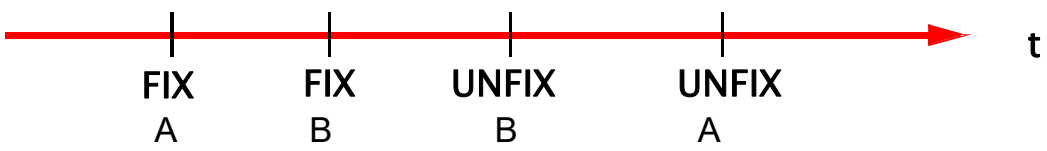
- Ersetzungskriterium: Zeit seit der letzten Referenzierung der Seite.
- eine referenzierte Seite kommt an die Spitze des LRU-Stacks
  - falls referenzierte Seite an Position *i* im Stack war, rutschen alle Seiten an den Positionen 1 bis *i*-1 eine Position tiefer
- Seite am Kellerboden wird ersetzt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






# Least-Recently-Used (2)

- 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

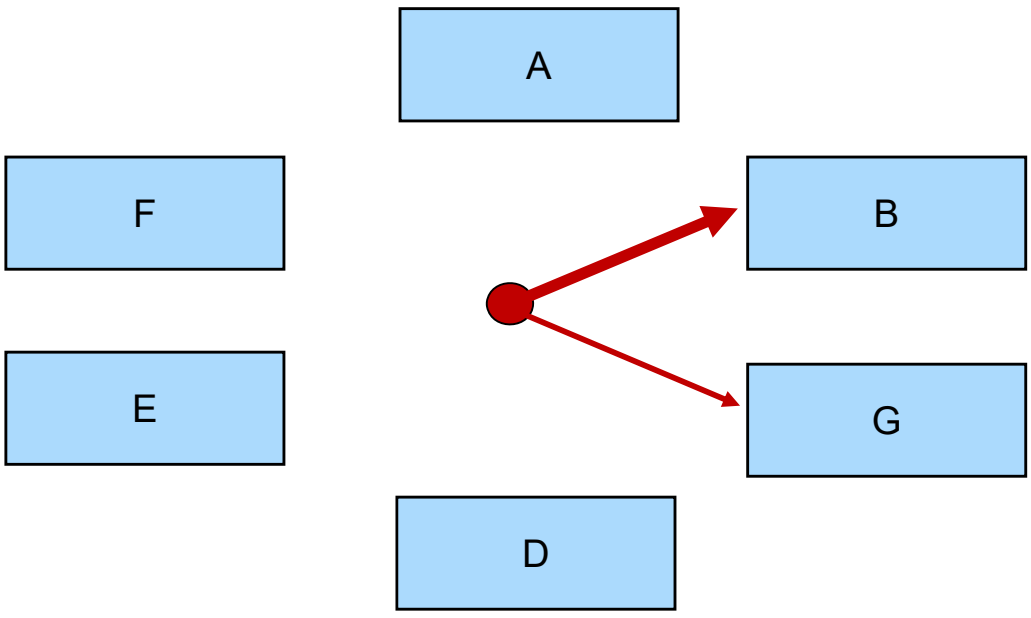
| RZ | |
|---|---|
| 2 | |
| 4 | |
| 1 | |
| 3 | |
| 3 | |
| 6 | |
| 1 | |
| 3 | |

- 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

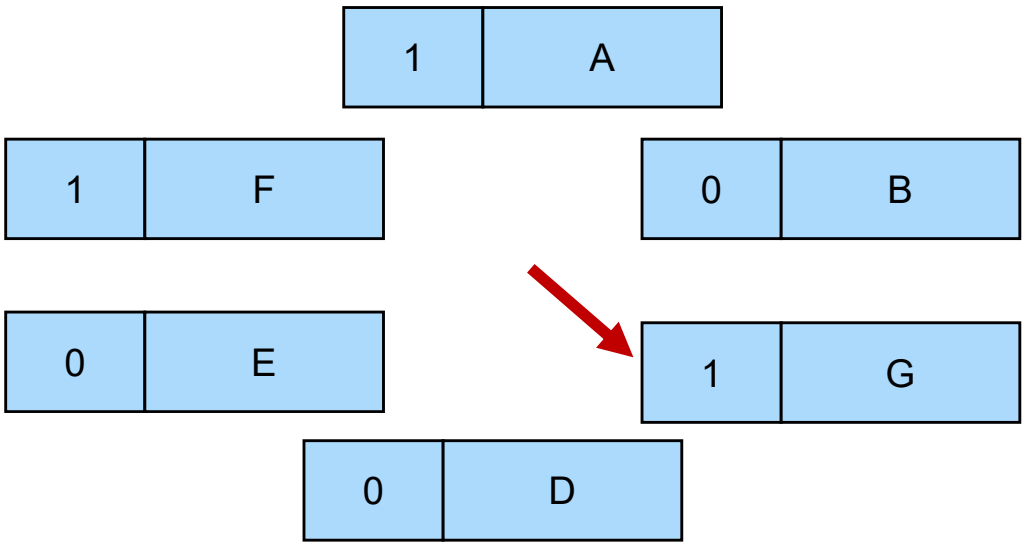
# Kriterien für Auswahl zu ersetzender Pufferseite

| Verfahren | Alter | letzter Referenzierungs-zeitpunkt | Referenzhäufigkeit | andere Kriterien |
|---|---|---|---|---|
| OPT | - | - | - | Vorauswissen |
| RANDOM | - | - | - | - |
| LRU | (x) | **X** | | |
| LFU | | | **X** | |
| FIFO | **X** | | | |
| CLOCK | X | (X) | | |
| GCLOCK | X | (X) | (X) | Gewichte |
| LRD V1 | X | - | (X) | |
| LRD V2 | X | (X) | (X) | Parameter I, K2, K3 |
| LRU-K | (X) | (X) | (X) | k-letzte Referenz |
| Adaptives LRU | (X | X | (X)<br>1 oder 2+ Referenz(en) | doppelte Historie |



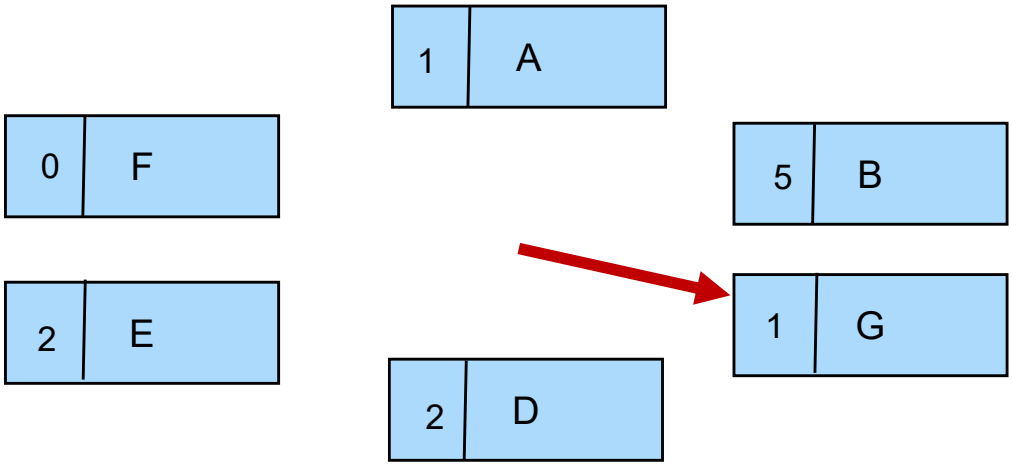
# CLOCK (Second Chance)

- 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

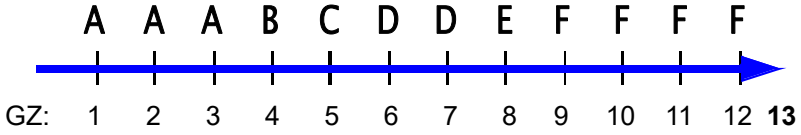


# Least-Reference-Density (LRD)

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

***Referenzdichte (für Seite j )***

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

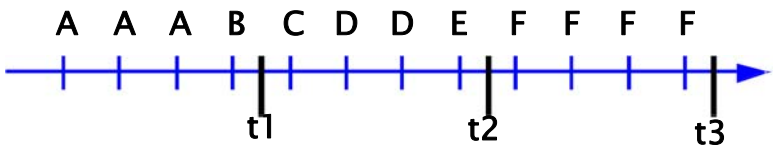



| | RZ | EZ | RD |
|---|---|---|---|
| A | 3 | 1 | 3/12=0,25 |
| B | 1 | 4 | 0,111 |
| C | 1 | 5 | 0.125 |
| D | 2 | 6 | 0,286 |
| E | 1 | 8 | 0,2 |
| F | 4 | 9 | 1,0 |

# Least-Reference-Density (2)

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

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



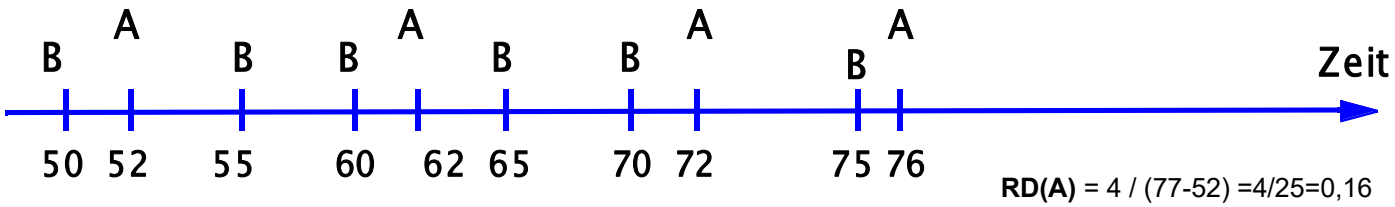

I=4, K2=1, K3=0

| | t1 | t2 | t3 |
|---|---|---|---|
| RZ(A) | 3 | | |
| RZ(B) | 1 | | |
| RZ(C) | | 1 | |
| RZ(D) | | 2 | |
| RZ(E) | | 1 | |
| RZ(F) | | | 4 |



# LRU-K

- Berücksichtigung der K letzten Referenzzeitpunkte einer Seite
  - erlaubt Approximation der Referenzdichte 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 LRD V2)
- Beispiel (K=4)




**RD(A)** = 4 / (77-52) =4/25=0,16

**RD(B)** = 6 / (77-50)=0,22

- 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

# Probleme von LRU

- LRU ungeeignet für sequenzielle 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 → Thrashing

  Beispiel: S=5, 4 Seitenrahmen

| A | B | C | D |
|---|---|---|---|

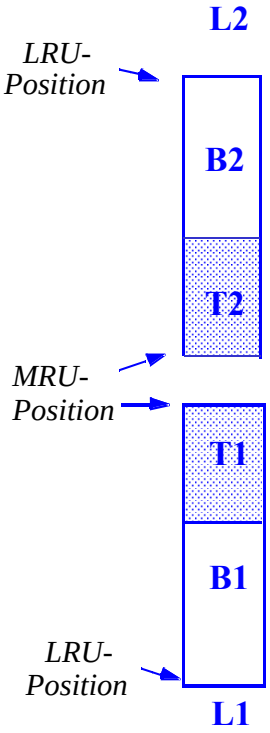
- Mehrbenutzereffekte: Transaktionen mit hoher Referenzlokalität können durch gleichzeitige sequenzielle Scans mit schneller Seitenanforderung stark benachteiligt werden

- Alternativen
  - Ausnutzen von Kontextwissen des Query-Optimierers ("hints" an Pufferverwaltung)
  - LRU-Erweiterungen bzgl. Prioritäten, Referenzhäufigkeiten etc.



# Adaptives LRU*

- Verwendung von 2 LRU-Ketten
  - L1: Seiten die nur 1-mal referenziert wurden (v.a. für sequenzielle Zugriffe)
  - L2: Seiten, die wenigstens 2-mal referenziert wurden
- für Cache-Größe c werden 2c Seiten in L1 und L2 verwaltet
  - Top-Bereiche T1 und T2 für gepufferte Seiten und Bottom-Bereiche B1 und B2 für nicht mehr gepufferte Seiten
    |T1|+|T2|<=c ; |L1|+|L2| = |T1|+|T2|+|B1|+|B2| <=2c; |L1|<=c
  - falls angeforderte Seite in L1 oder L2 wird sie an MRU-Position von L2 gebracht, ansonsten an MRU-Position von L1
- Adaptionsparameter p
  - bestimmt relatives Verhältnis zwischen L1- und L2-Seiten im Puffer (Ziel p ≈ |T1|)
  - dynamische Anpassung von p gemäß aktueller Verteilung von sequenziellen Zugriffen und Zugriffen mit temporaler Lokalität
  - Heuristik: investiere in erfolgreiche Liste (vergrößere T1 bzw. T2 für Treffer in B1 bzw. B2)




* 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

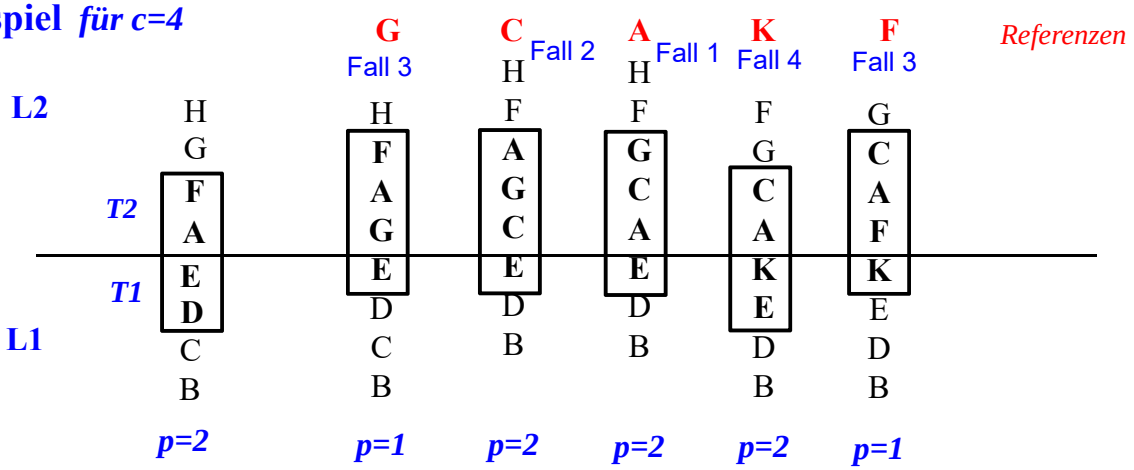
  2:"Hit" in B1: p := min (p+k1,c) mit k1 = 1 *falls* |*B1*| >=| *B2*|, *sonst k1*=|*B2*|/|*B1*|; Erhöhe p (|T1|)
  REPLACE(x,p); lese x ein und bringe x an MRU-Position von T2

  3: "Hit" in B2: p:=max(p-k2,0) mit k2=1 *falls* |*B2*|>=| *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** ***für c=4***






# Zusammenfassung

- Segmentkonzept: Logische Behälter für DB-Inhalte (table spaces), Indexstrukturen
- Update-in-Place ist indirekten Einbringstrategien vorzuziehen
- DB-Pufferverwaltung zur Minimierung physischer E/A
  - Nutzung von Lokalität innerhalb und zwischen Transaktionen, Sequenzialität, ...
  - Suche im Puffer: durch Hash-Verfahren
  - globale Speicherzuteilung vorteilhaft (Pufferrahmen für alle Transaktionen)
  - Behandlung geänderter Seiten: NOFORCE, asynchrones Ausschreiben
- Seitenersetzungsverfahren
  - Nutzung mehrerer Kriterien möglich: Alter, letzte Referenz, Referenzhäufigkeit
  - LRU ist guter Default-Ansatz
  - LRU-2 wählt Ersetzungskandidaten aufgrund des vorletzten Referenzzeitpunktes aus
  - adaptives LRU