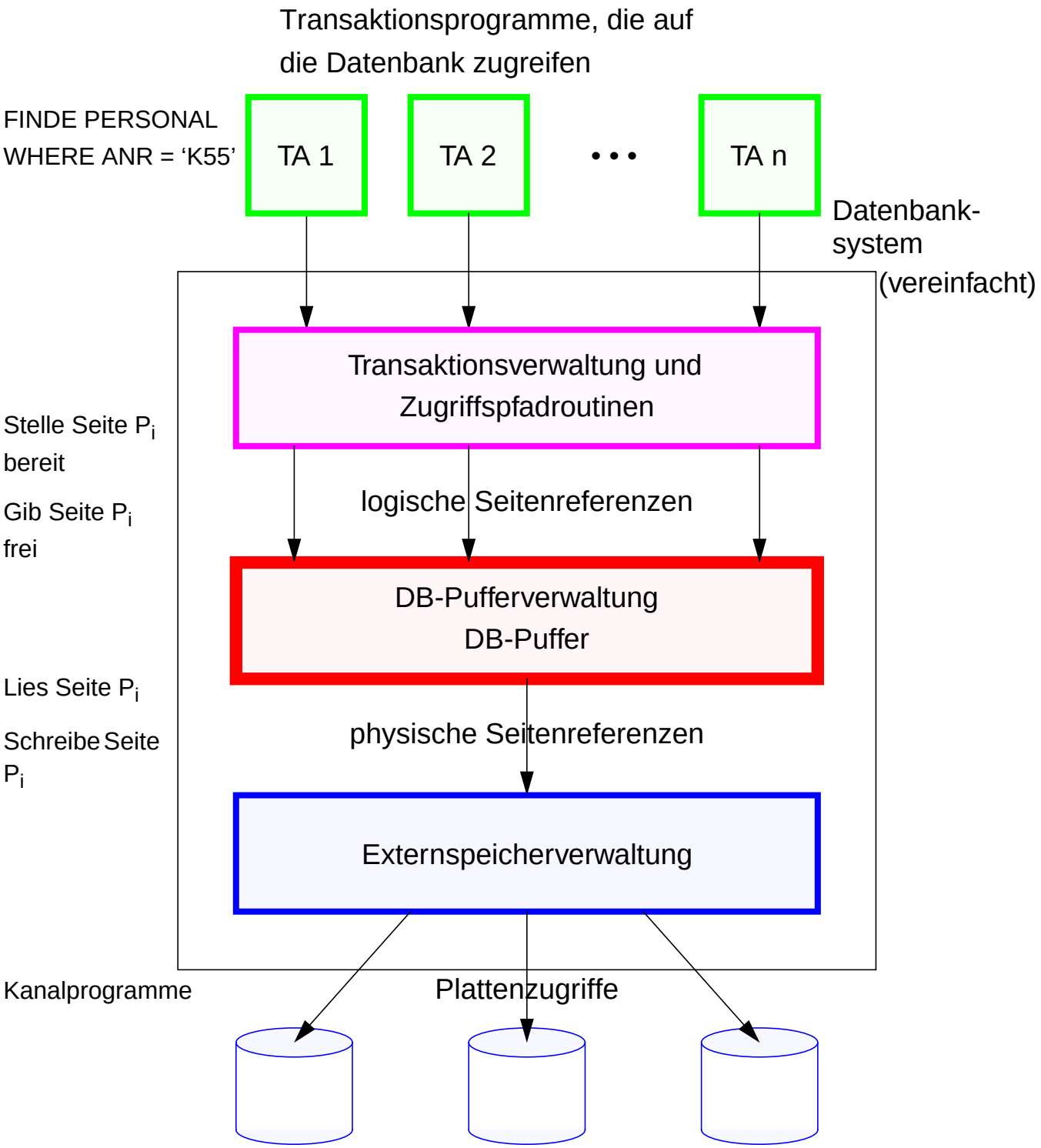
3. DB-Pufferverwaltung

- **Ziel: Realisierung einer effizienten, seitenbasierten Verarbeitungsplattform im Hauptspeicher**
  - größtmögliche Vermeidung von physischer Ein-/Ausgabe
  - Ersetzungsverfahren ohne und mit Kontextwissen
- **Rolle der DB-Pufferverwaltung**[1]
  - Ablauf des Zugriffs auf den DB-Puffer
  - Vergleich mit ähnlicher Funktionalität in Betriebssystemen (BS)
- **Lokalität**
  - Maße für Lokalität
  - Charakterisierung durch LRU-Stacktiefen-Verteilung Und Referenzdichtekurven
- **Speicherzuteilung und Suche im DB-Puffer**
- **Seitenersetzungsverfahren**
  - Klassifikation von Ersetzungsverfahren
  - LRU, FIFO, CLOCK, GCLOCK, LRD, LRU-K ...
- **Ersetzungsverfahren – Einbezug von Kontextwissen**
- **DB-Caching**
  - Klassifikation der Verfahren
  - DBProxy
  - DBCache

---

1. Effelsberg, W., Härder, T.: Principles of Database Buffer Management, in: ACM Transactions on Database Systems 9:4, Dec. 1984, pp. 560-595.

Rolle der DB-Pufferverwaltung in einem Datenbanksystem



Seitenreferenzstrings

- **Jede Datenanforderung ist eine *logische Seitenreferenz***

- **Aufgabe der DB-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

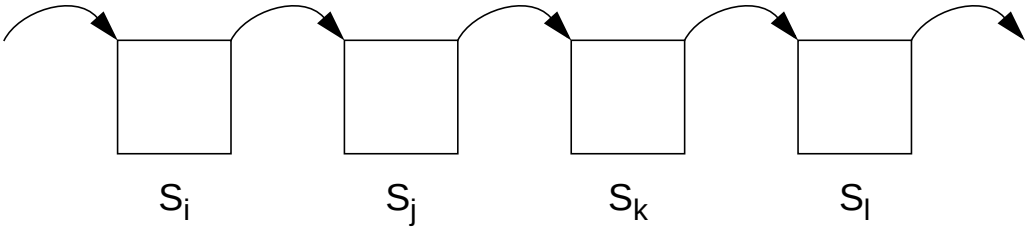
- Bestimmung von Ausschnitten aus R bezüglich bestimmter Transaktionen, Transaktions-Typen und DB-Partitionen sinnvoll zur Analyse des Referenzverhaltens

- **Wie kann Referenzstring-Information verwendet werden für**
  - Charakterisierung des Referenzverhaltens?
  - Bestimmung von Lokalität und Sequentialität?
  - Unterstützung einer effektiven Seitenersetzung?

Eigenschaften von DB-Referenzstrings

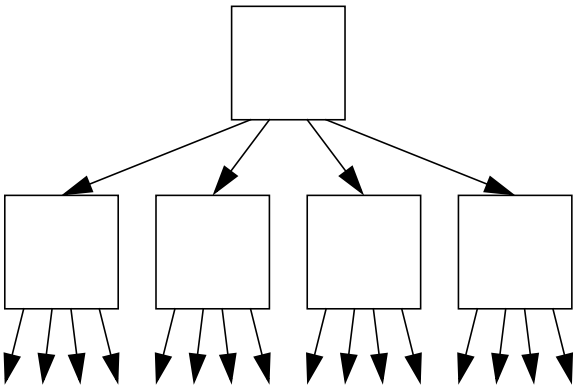
- **Typische Referenzmuster in DBS**

1. Sequentielle Suche



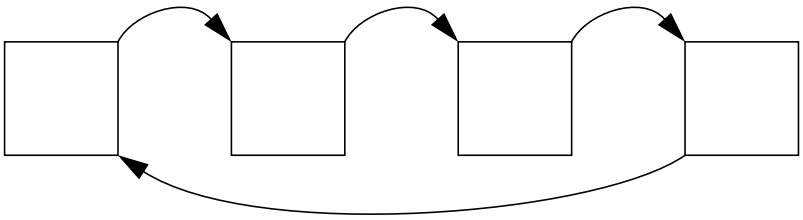
Bsp.: Durchsuchen ganzer Satztypen (Relationen)

2. Hierarchische Pfade



Bsp.: Suchen mit Hilfe von B*-Bäumen

3. Zyklische Pfade



Bsp.: Abarbeiten von Sets ((1:n)-Beziehungen),
Suchen in DBTT-/Datenseiten

Vergleich mit BS-Funktionen

- **Ersetzungsalgorithmen im DB-Puffer in Software implementiert – Seitenersetzung in Adreßräumen bei Virtuellem Speicher ist HW-gestützt**

- **Seitenreferenz vs. Adressierung**

  nach einem FIX-Aufruf kann eine DB-Seite mehrfach bis zum UNFIX referenziert werden

  ➥ unterschiedliches Seitenreferenzverhalten

  ➥ andere Ersetzungsverfahren?

- **Können Dateipuffer des BS als DB-Puffer eingesetzt werden?**

1. **Zugriff auf Dateipuffer ist teuer (SVC:** *supervisor call***)**

2. **DB-spezifische Referenzmuster können nicht gezielt genutzt werden**

   BS-Ersetzungsverfahren sind z. B. nicht auf zyklisch sequentielle oder baumartige Zugriffsfolgen abgestimmt

3. **Normale Dateisysteme bieten keine geeignete Schnittstelle für Prefetching**

   In DBMS ist aufgrund von Seiteninhalten oder Referenzmustern eine Voraussage des Referenzverhaltens (z. B. bei Tabellen-Scans) möglich; Prefetching erzielt in solchen Fällen eine enorme Leistungssteigerung

4. **Selektives Ausschreiben von Seiten zu bestimmten Zeitpunkten (z. B. für Logging) nicht immer möglich in existierenden Dateisystemen**

   ➥ DBVS muß eigene Pufferverwaltung realisieren

Sequentialität

- SRS weisen typischerweise **Phasen von Sequentialität und Lokalität** auf

- **Sequentielle Zugriffsfolge (SZ):**

  Zwei aufeinanderfolgende Referenzen r_i und r_{i+1} gehören zu einer sequentiellen Zugriffsfolge, falls

  $$S_{i+1} - S_i = 0 \text{ oder } 1$$

  d. h., aufeinanderfolgende Zugriffe referenzieren benachbarte DB-Seiten

- **Algorithmus**

  - Seitenreferenzstring wird vollständig durchmustert; alternativ kann die Folge der ankommenden Referenzen analysiert werden
  - Solange obige Bedingung erfüllt ist,
    gehören alle aufeinanderfolgenden Referenzen zu einer SZ,
    sonst beginnt eine neue SZ

- **Länge einer sequentiellen Zugriffsfolge (LSZ):**

  - LSZ ist die Anzahl der verschiedenen in SZ referenzierten Seiten
  - Bsp.:Referenzstring
    A A B B D E E F F H enthält
    (AABB) mit LSZ(1) = 2, (DEEFF) mit LSZ(2) = 3 und (H) mit LSZ(3) = 1

- **Maß für Sequentialität:**

  - Die kumulative Verteilung der SZ-Längen LSZ(i) wird berechnet

    $$S(x) = Pr(\text{SZ-Länge} <= x)$$

  - Für obiges Bsp. gilt: S(1)=0.33, S(2)=0.67, S(3)=1.0

- **Bei Sequentialität Optimierung durch (asynchrones) Prefetching von DB-Seiten möglich**

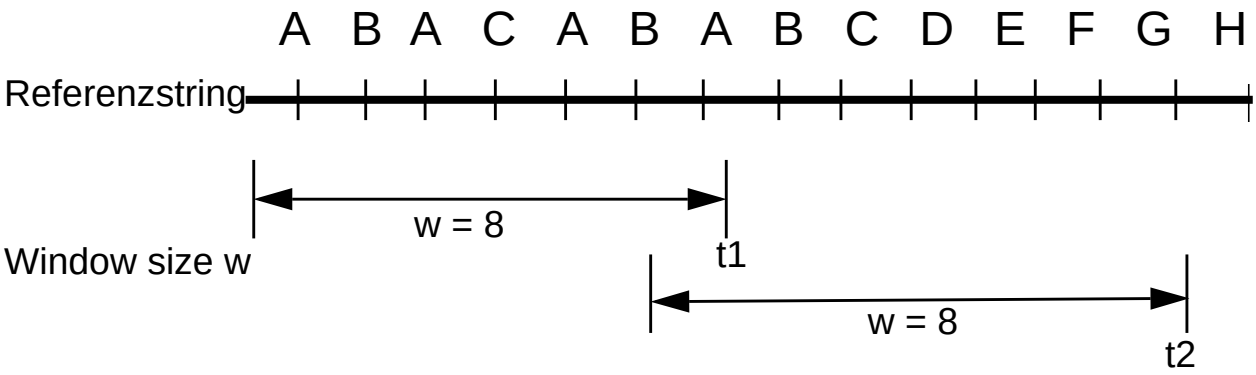
Lokalität

- **Erhöhte Wiederbenutzungswahrscheinlichkeit für gerade referenzierte Seiten (gradueller Begriff)**

- **Grundlegende Voraussetzung für**
  - effektive DB-Pufferverwaltung (Seitenersetzung)
  - Einsatz von Speicherhierarchien

- **Wie kann man Lokalität messen?**

Working-Set-Modell



Working set size W W (t1, w=8) = 3 W (t2, w=8) = 8

Aktuelle Lokalität: $AL(t, w) = \frac{W(t, w)}{w}$

Durchschnittliche Lokalität: $L(w) = \frac{\sum_{t=1}^{n} AL(t, w)}{n}$

Relative Referenzmatrix (DOA-Last)

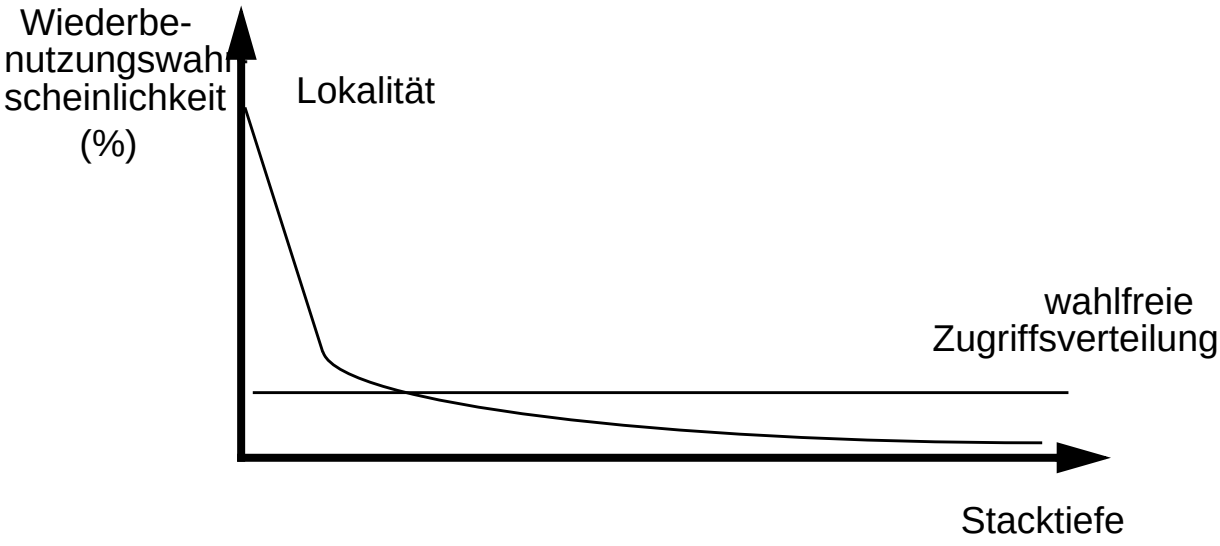
ca. 17 500 Transaktionen, 1 Million Seitenreferenzen auf ca. 66 000 verschiedene Seiten

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	P12	P13	Total
TT1	9.1	3.5	3.3		5.0	0.9	0.4	0.1				0.0		22.3
TT2	7.5	6.9	0.4	2.6	0.0	0.5	0.8	1.0	0.3	0.2	0.0			20.3
TT3	6.4	1.3	2.8	0.0	2.6	0.2	0.7	0.1	1.1	0.4		0.0	0.0	15.6
TT4	0.0	3.4	0.3	6.8			0.6	0.4			0.0			11.6
TT5	3.1	4.1	0.4		0.0		0.5	0.0						8.2
TT6	2.4	2.5	0.6		0.7		0.9	0.3						7.4
TT7	1.3		2.6			2.3	0.1							6.2
TT8	0.3	2.3	0.2		0.0		0.1							2.9
TT9	0.0	1.4	0.0					1.1						2.6
TT10	0.3	0.1	0.3			1.0	0.1					0.0		1.8
TT11		0.9						0.2						1.1
TT12		0.1												0.1
Total	30.3	26.6	11.0	9.4	8.3	4.9	4.1	3.3	1.4	0.6	0.0	0.0	0.0	100.0
partition size (%)	31.3	6.3	8.3	17.8	1.0	20.8	2.6	7.3	2.6	1.3	0.8	0.0	0.0	100.0
% refe-renced	11.1	16.6	8.0	2.5	18.1	1.5	9.5	4.4	5.2	2.7	0.2	13.5	5.0	6.9

LRU-Stacktiefenverteilung

- **Wie läßt sich Lokalität charakterisieren?**
  - LRU-Stacktiefenverteilung liefert Maß für die Lokalität (präziser als Working-Set-Ansatz)
  - LRU-Stack enthält alle bereits referenzierten Seiten in der Reihenfolge ihres Zugriffsalters

- **Bestimmung der Stacktiefenverteilung:**
  - pro Stackposition wird Zähler geführt
  - Rereferenz einer Seite führt zur Zählererhöhung für die jeweilige Stackposition



➥ **Zählerwerte entsprechen der Wiederbenutzungshäufigkeit**

Für LRU-Seitenersetzung kann aus der Stacktiefenverteilung für eine bestimmte Puffergröße unmittelbar die Trefferrate (bzw. Fehlseitenrate) bestimmt werden

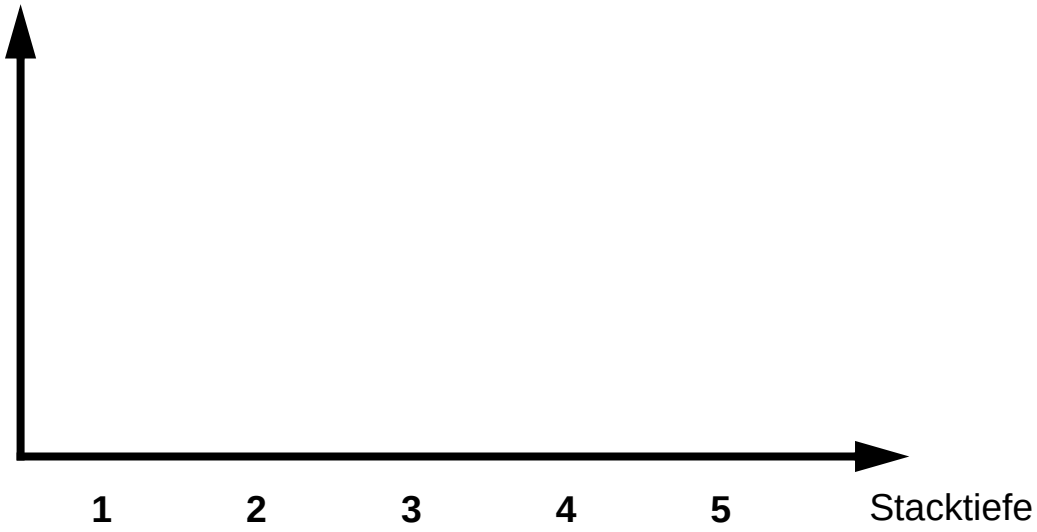
Beispiel: Ermittlung der Stacktiefen-Verteilung

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

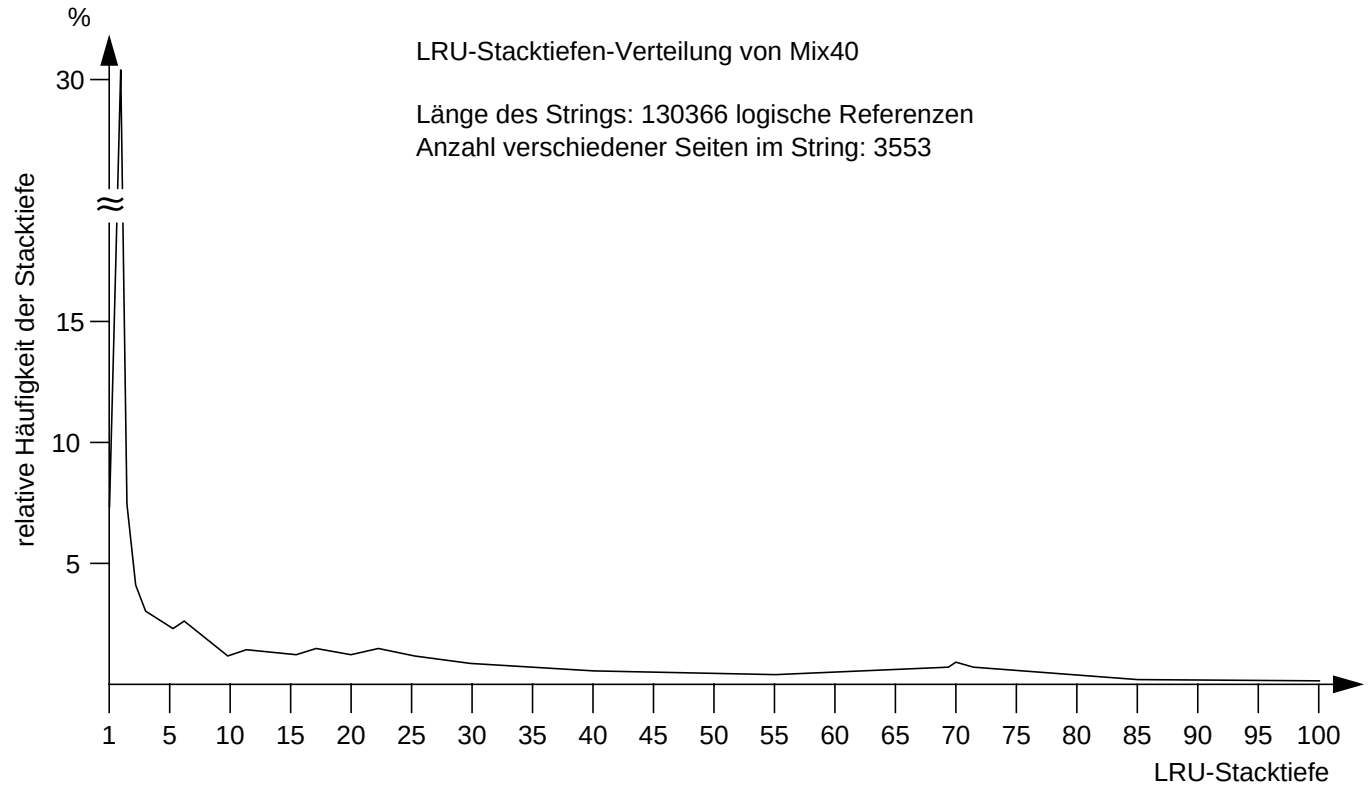
LRU-Stack:

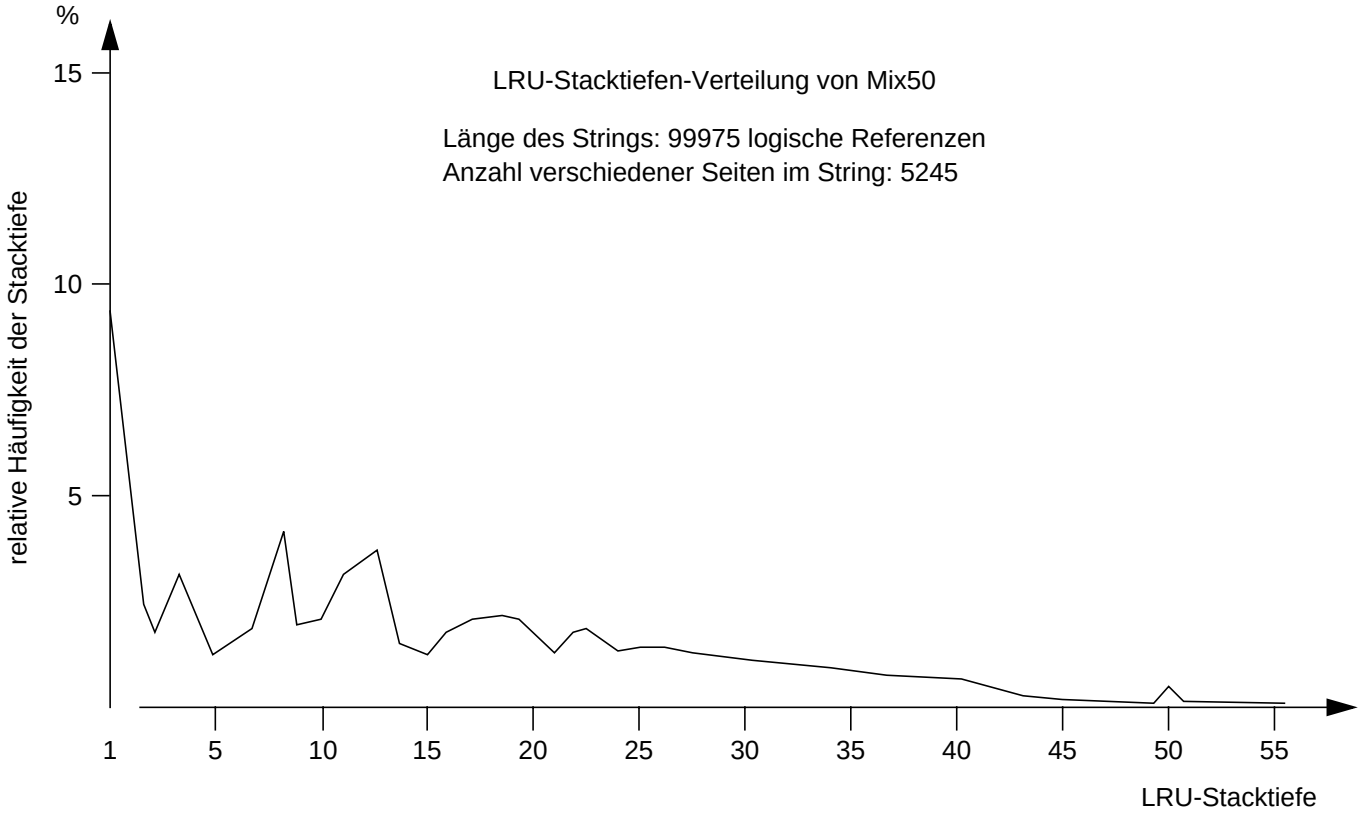
1	**A**
2	**B**
3	**C**
4	**D**
5	**E**

Stacktiefen-Verteilung

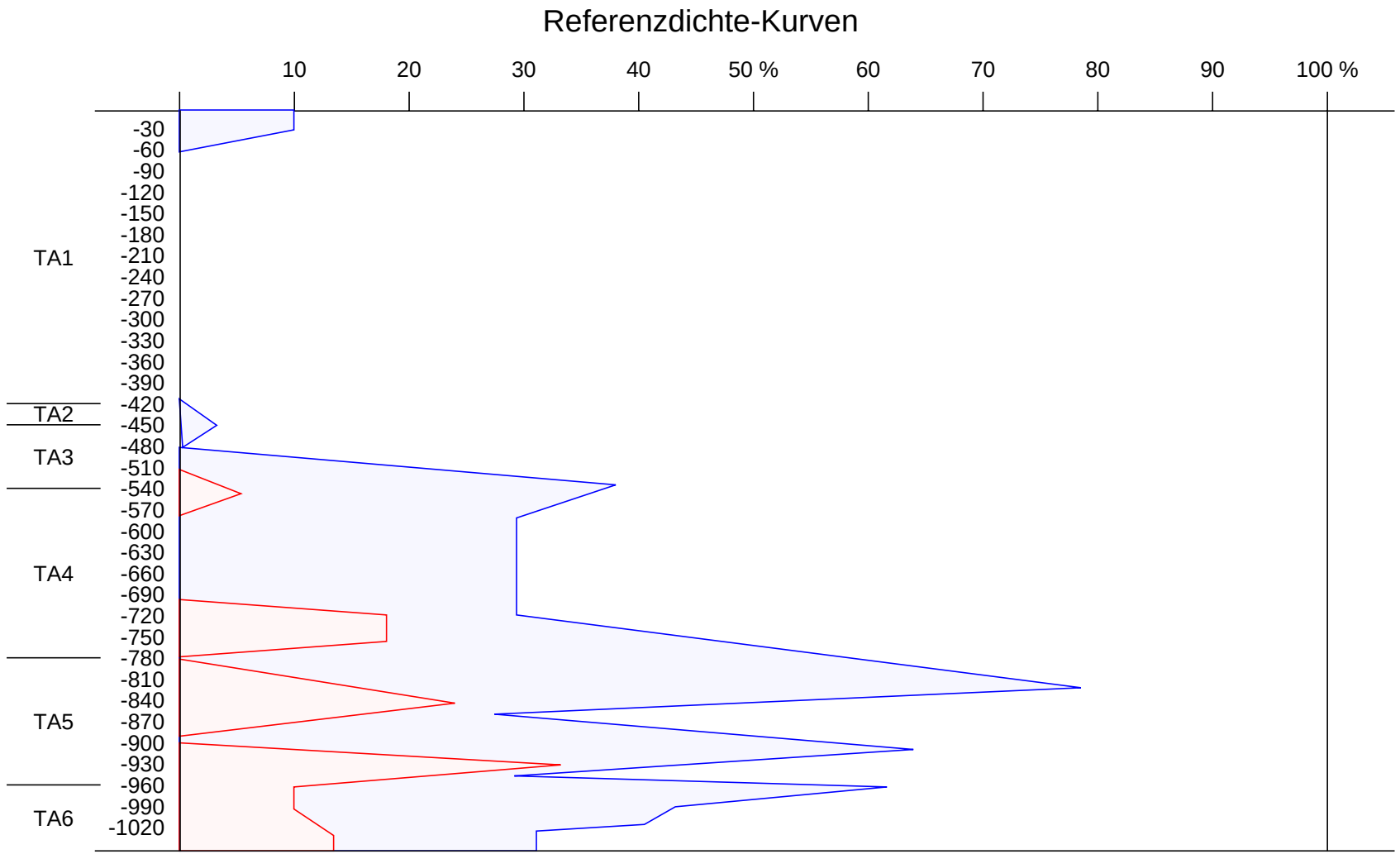


Reale LRU-Stacktiefen-Verteilungen[1]





1. W. Effelsberg, T. Härder: Principles of Database Buffer Management, ACM Transactions on Database Systems, Vol. 9, No. 4, Dec. 1984, pp. 560-595.

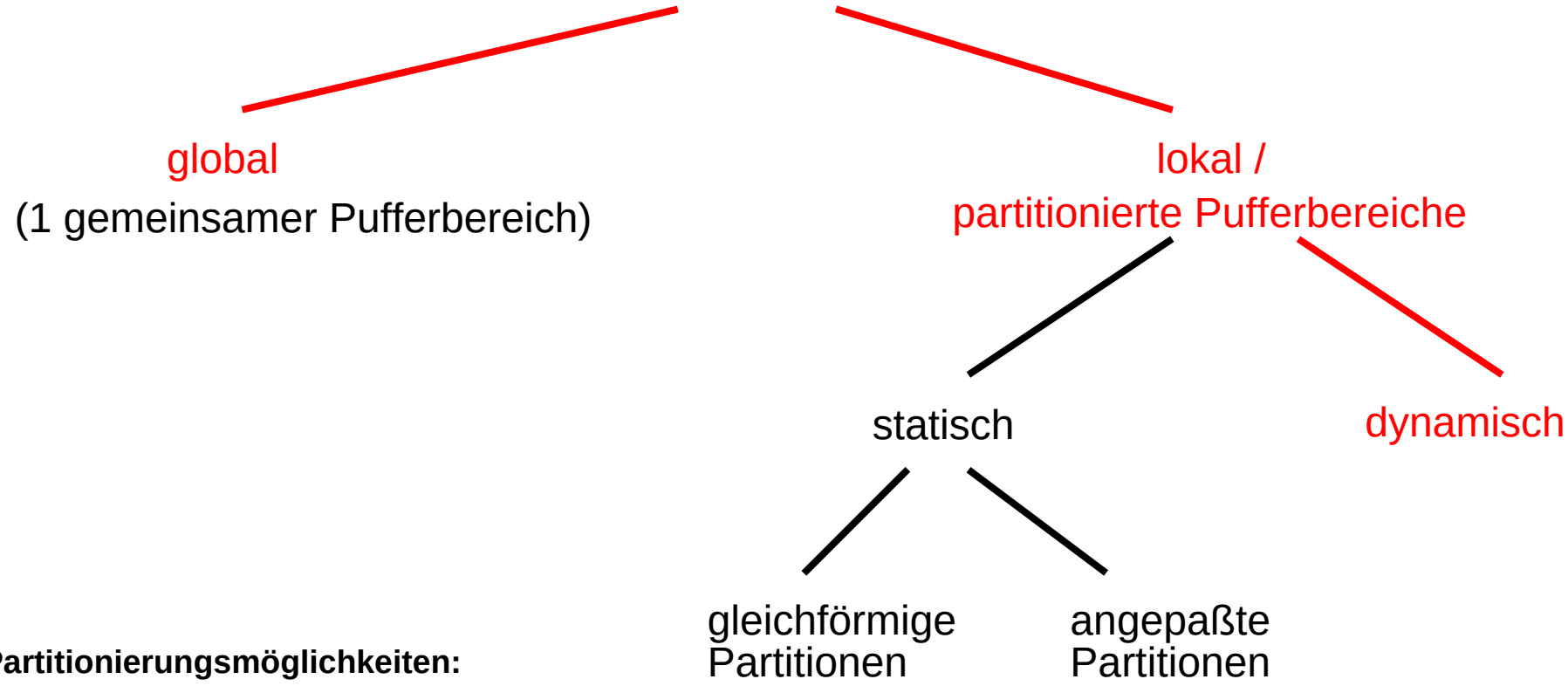


Relative Häufigkeit der Seitentypen im Beispiel

- = Daten und Indexstrukturen: 93,8 %
- = Adressumsetztabellen: 6,1 %
- = Freispeicher-Verwaltung: 0,1 %

Referenzdichte-Kurven

Speicherzuteilung im DB-Puffer



Partitionierungsmöglichkeiten:

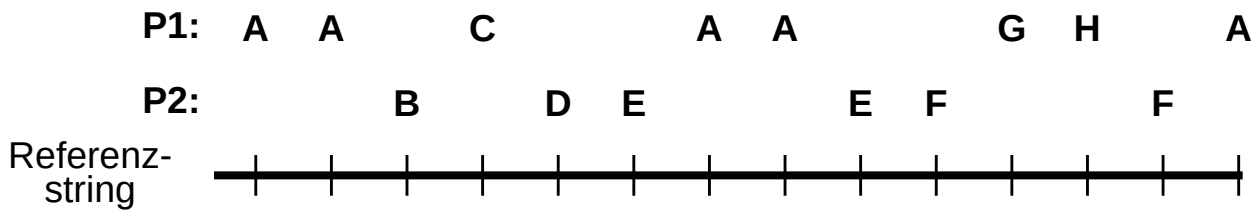
- eigener Pufferbereich pro Transaktion
- TA-Typ-bezogene Pufferbereiche
- Seitentyp-bezogene Pufferbereiche
- DB-(Partitions)spezifische Pufferbereiche

Dynamische Pufferallokation – Working-Set-Ansatz (WS)

- **Pro Pufferpartition P soll Working-Set im Puffer bleiben;**
  Seiten, die nicht zum Working-Set gehören, können ersetzt werden

- **Bei Fehlseitenbedingung muß Working-Set bekannt sein,**
  um Ersetzungskandidat zu bestimmen
  - Fenstergröße (*Window Size*) pro Partition: w (P)
  - Referenzzähler pro Partition: RZ (P)
  - letzter Referenzzeitpunkt für Seite i: LRZ (P, i)
  - ersetzbar sind solche Seiten, für die RZ (P) — LRZ (P, i) > w (P)

- **Fenstergröße kritischer Parameter → Thrashing-Gefahr**



Suche im DB-Puffer

- **Sequentielles Durchsuchen der Pufferrahmen**
  - sehr hoher Suchaufwand
  - Gefahr vieler Paging-Fehler bei virtuellen Speichern

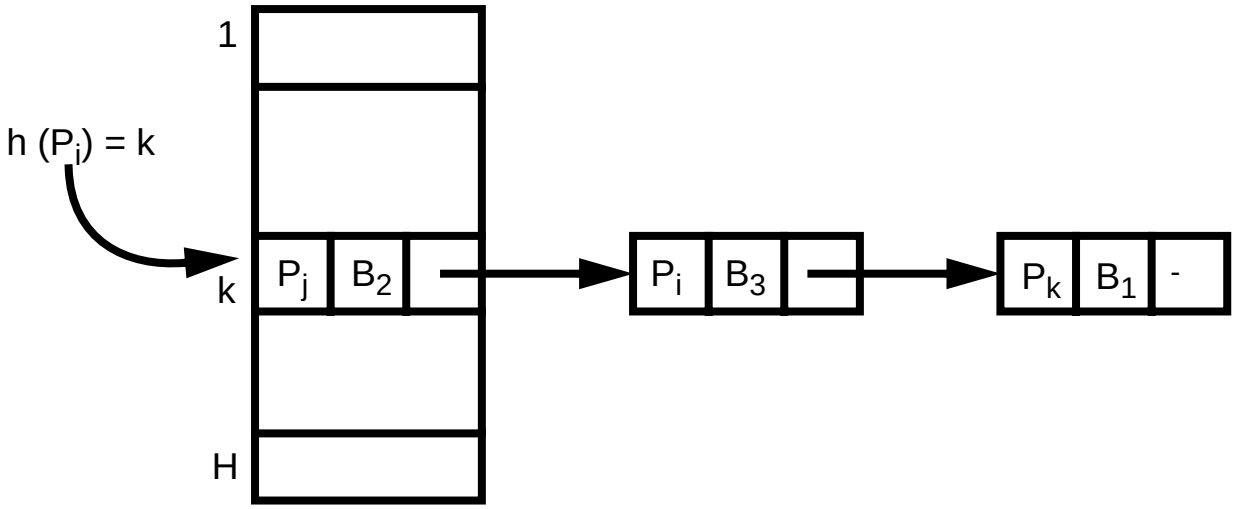
- **Nutzung von Hilfsstrukturen**
  (Eintrag pro Pufferrahmen)

  1. **unsortierte oder sortierte Tabelle**

  2. **Tabelle mit verketteten Einträgen**
     - geringere Änderungskosten
     - Anordnung in LRU-Reihenfolge möglich

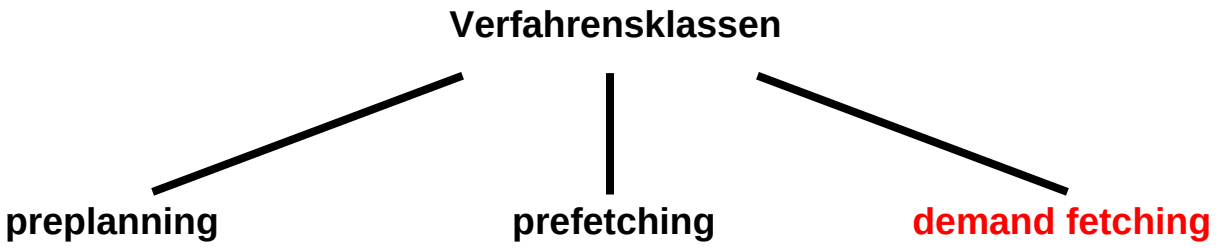
  3. **Suchbäume (z. B. AVL-, m-Weg-Bäume)**

  4. **Hash-Tabelle mit Überlaufketten**
     - beste Lösung



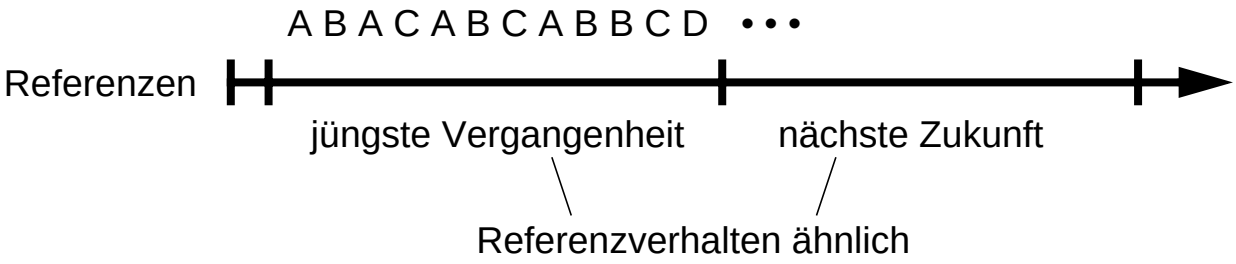
Seitenersetzungsverfahren

- **Klassifikation**



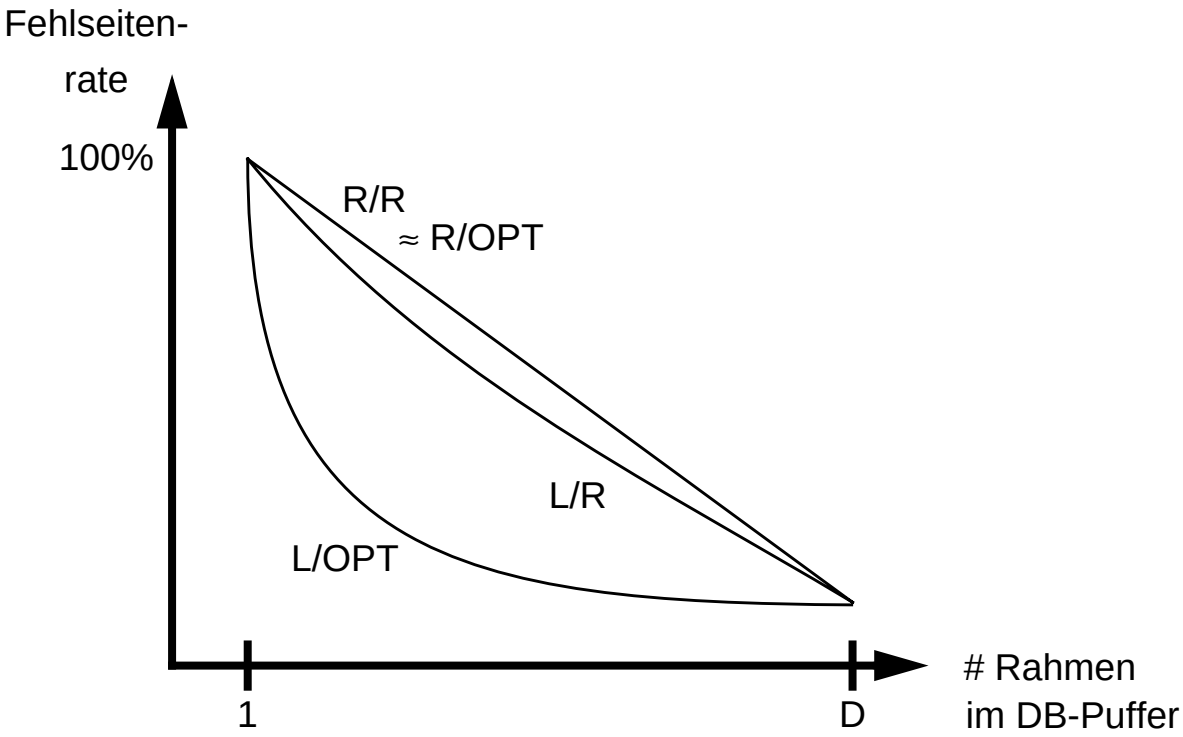
preplanning	prefetching	demand fetching
Programmanalyse, Vorabuntersuchung des Datenbedarfs	physische Datenstrukturierung, Clusterbildung, Verarbeitungswissen	keine Vorausaktionen
➥ große Fehlrate, ungenaue Obermengen	➥ datenmodellbezogen (hierarchisch), spekulative Entscheidungen	➥ Lokalitätserhaltung im DB-Puffer

- **Grundannahme bei Ersetzungsverfahren**:



Referenzverhalten und Ersetzungsverfahren

- **Referenzverhalten in DBS**
  - **typischerweise** hohe Lokalität: Optimierung durch Ersetzungsverfahren
  - manchmal Sequentialität oder zufällige Arbeitslast (RANDOM-Referenzen)
- **Prinzipielle Zusammenhänge**, welche die Fehlseitenrate bestimmen



D = DB-Größe in Blöcken

- **Kombinationen:**

Referenzen:	RANDOM	RANDOM	Lokalität	Lokalität
Ersetzung:	RANDOM	OPT	RANDOM	OPT

➥ Grenzfälle des Referenzverhaltens und der Ersetzungsverfahren zeigen Optimierungsmöglichkeiten auf

Behandlung geänderter Seiten im DB-Puffer

- **Ersetzung einer geänderten Seite erfordert ihr vorheriges (synchrones) Zurückschreiben in die DB**

  ➥ Antwortzeitverschlechterung

- **Abhängigkeit zur gewählten *Ausschreibstrategie*:**

  FORCE: alle Änderungen einer Transaktion werden spätestens beim EOT in die DB zurückgeschrieben („write-through“)

  \+ i. allg. stets ungeänderte Seiten zur Ersetzung vorhanden

  \+ vereinfachte Recovery (nach Rechnerausfall sind alle Änderungen beendeter TA bereits in die DB eingebracht)

  \- hoher E/A-Overhead

  \- starke Antwortzeiterhöhung für Änderungstransaktionen

  NOFORCE: kein Durchschreiben der Änderungen bei EOT (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 DB-Schreibvorgänge lassen sich weitgehend vermeiden

Kriterien für die Auswahl der zu ersetzenden Pufferseite

Verfahren	Kriterien			
	Alter	letzte Referenz	Referenz-häufigkeit	andere Kriterien
OPT	-	-	-	**Vorauswissen**
RANDOM	-	-	-	---
LFU	-	-	X	---
FIFO	X	-	-	---
LRU				
CLOCK				
GCLOCK				
LRD (V1)				
LRD (V2)				
LRU-K				

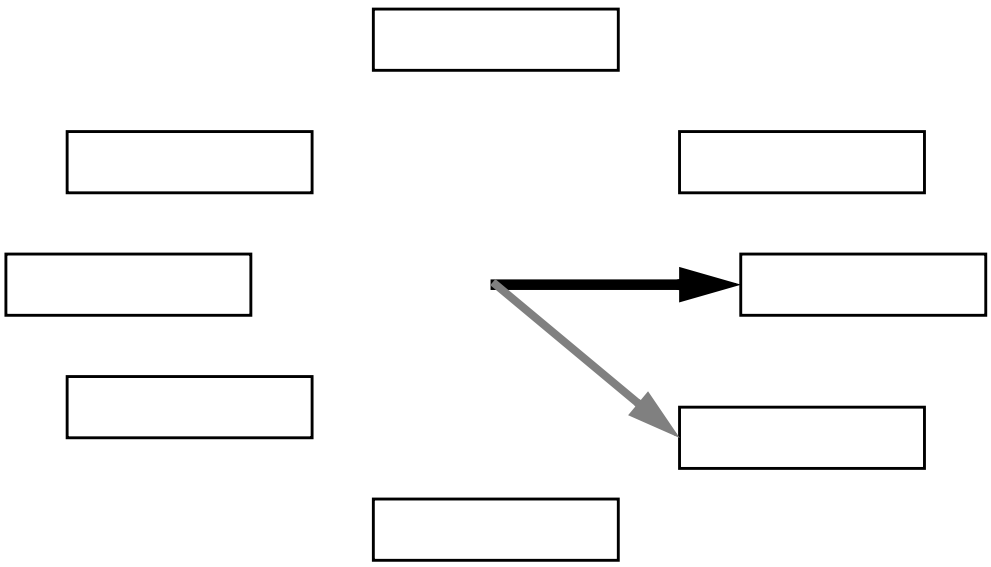
Least Frequently Used und First-In First-Out

- **Algorithmus LFU**
  - Referenzzähler pro Seite wird bei jeder Seitenreferenz inkrementiert
  - Ersetzung der Seite mit der geringsten Referenzhäufigkeit

RZ	
2	
4	
1	
3	
3	
6	
1	
3	

➥ **Alter einer Seite wird nicht berücksichtigt!**

- **Algorithmus FIFO**
  - Die älteste Seite im DB-Puffer wird ersetzt
  - Referenzen während des Pufferaufenthaltes werden nicht berücksichtigt

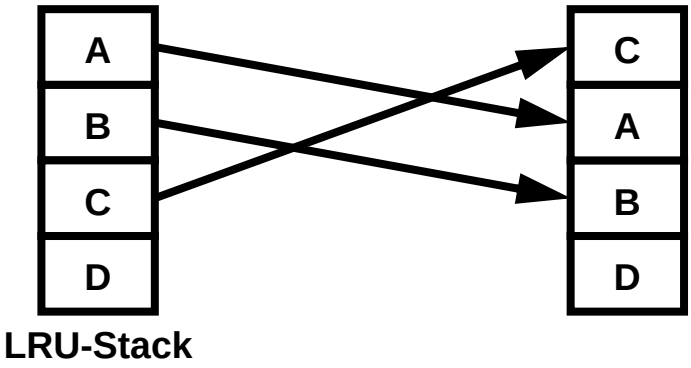


➥ **Nur für strikt sequentielles Referenzierungsverhalten geeignet**

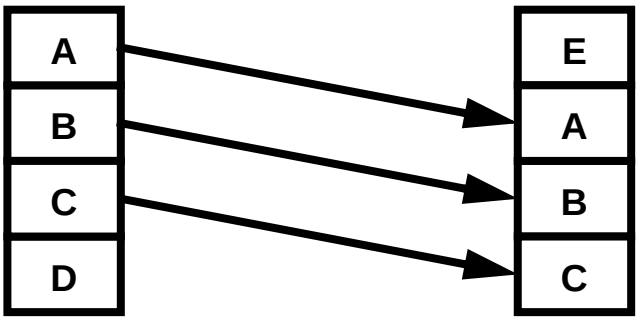
Least Recently Used (LRU)

- **Beispiel (Puffergröße 4):**

  1. Referenz der Seite C



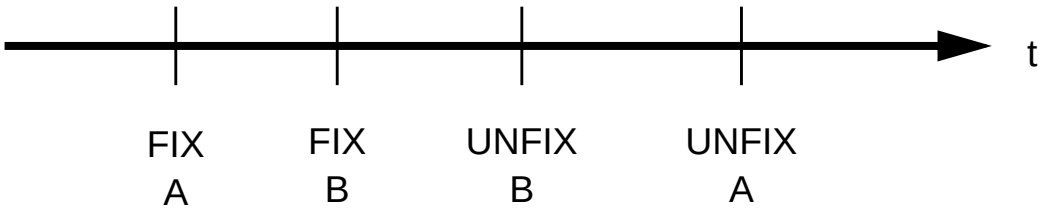
  2. Referenz der Seite E



- **Unterscheidung zwischen**

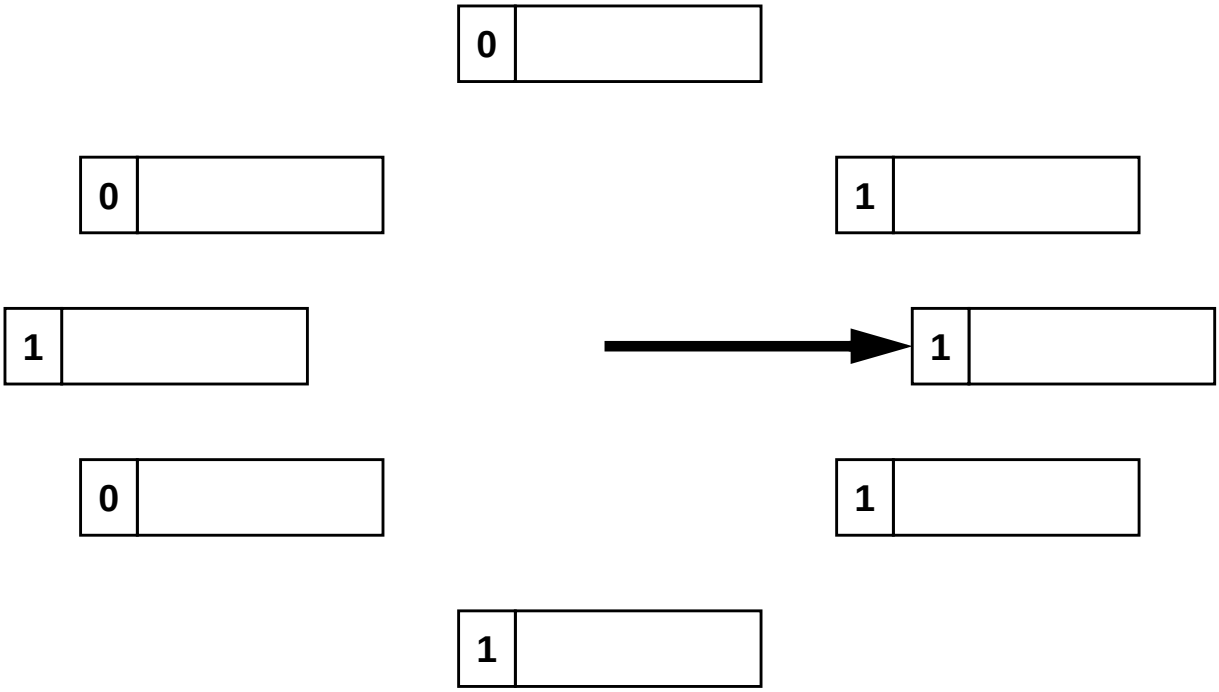
  Least Recently Referenced **und**

  Least Recently Unfixed



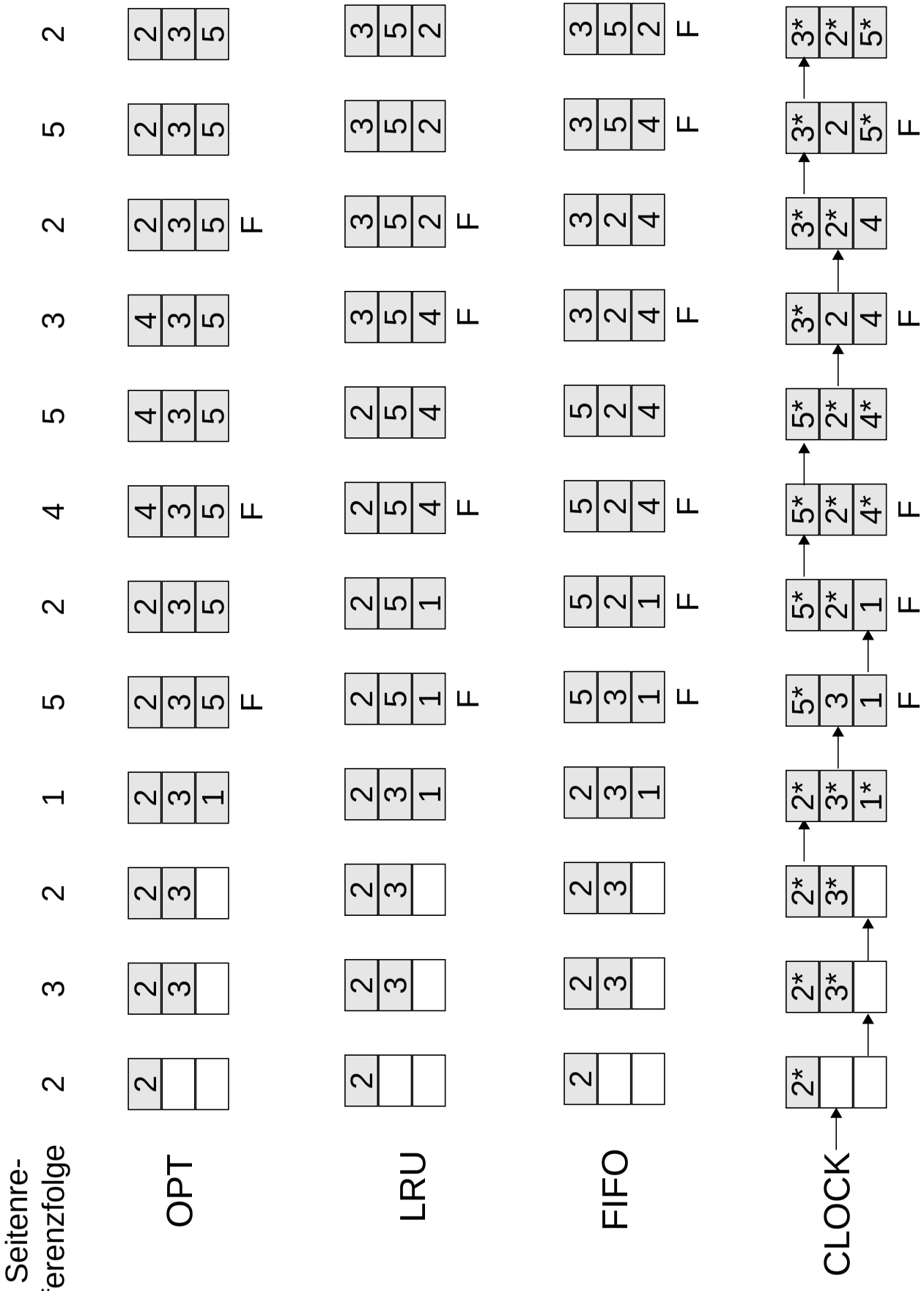
CLOCK (Second Chance)

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



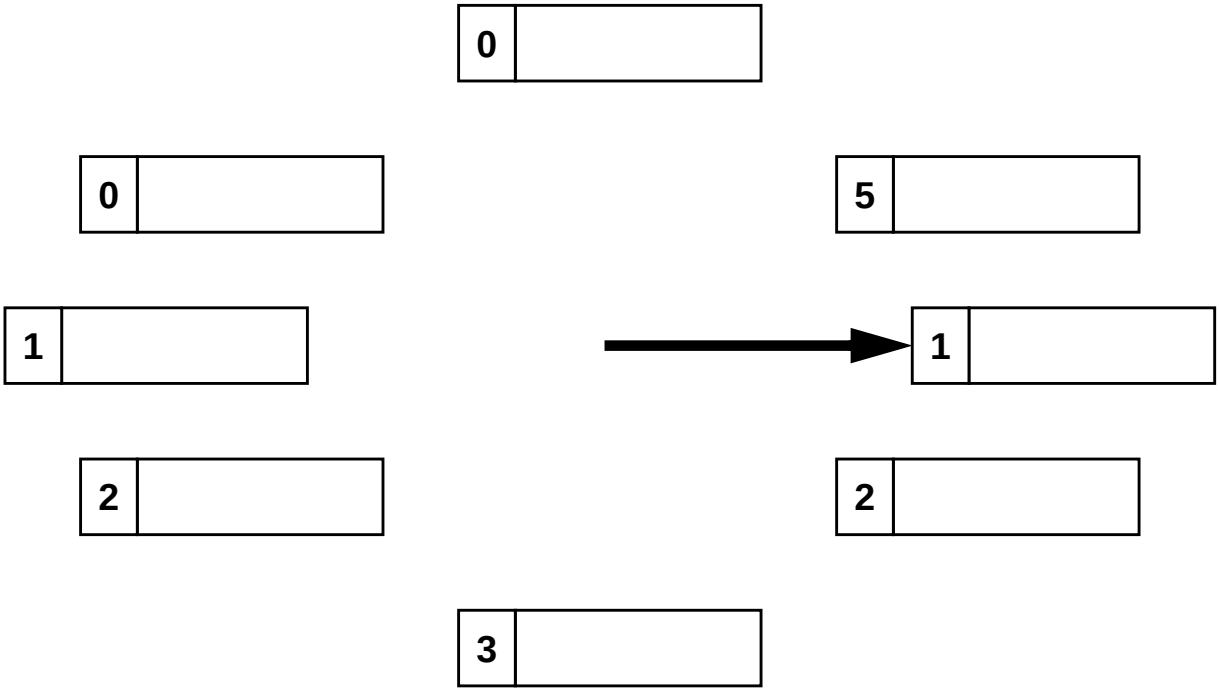
➥ **annähernde Berücksichtigung des letzten Referenzierungszeitpunkts**

Seitenersetzungsverfahren — Beispiel



GCLOCK (Generalized CLOCK)

- **Algorithmus**
  - 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
  - Zähler-Inkrementierung bei erneuter Referenz
  - Vergabe von seitentyp- oder seitenspezifischen Gewichten

Least Reference Density (LRD)

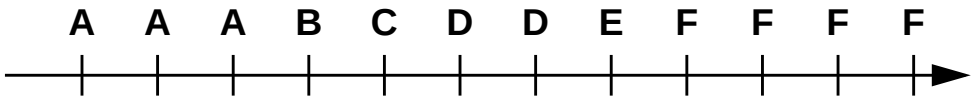
- **Algorithmus**
  - Wenn eine Seite ersetzt werden muß, wird die Referenzdichte aller Seiten im DB-Puffer bestimmt
  - Referenzdichte = Referenzhäufigkeit in einem bestimmten Referenzintervall
  - Ersetzungskandidat ist Seite mit geringster Referenzdichte

- **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			

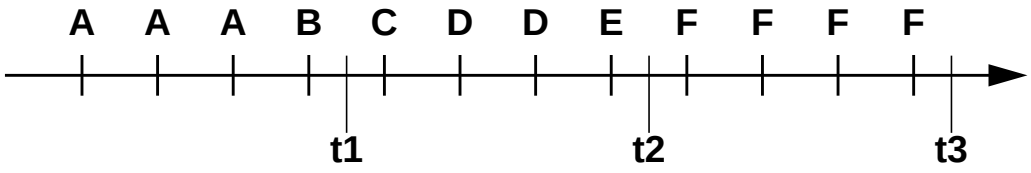
Least Reference Density (2)

- **Variante 2: konstante Intervallgröße**
  - Künstliches Altern von Seiten: Ältere Referenzen werden bei der Bestimmung der Referenzdichte geringer bewertet
  - **Periodisches Reduzieren** der Referenzzähler, um Gewicht früher Referenzen zu reduzieren
  - Reduzierung von RZ durch Division oder Subtraktion:

$$RZ(i) = \frac{RZ(i)}{K1} \qquad (K1 > 1)$$

oder

$$RZ(i) = \begin{cases} RZ(i) - K2 & \text{falls } RZ(i) - K2 \geq K3 \\ K3 & \text{sonst} \end{cases} \qquad (K2 > 0, K3 \geq 0)$$



	t1	t2	t3
RZ(A)			
RZ(B)			
RZ(C)			
RZ(D)			
RZ(E)			
RZ(F)			

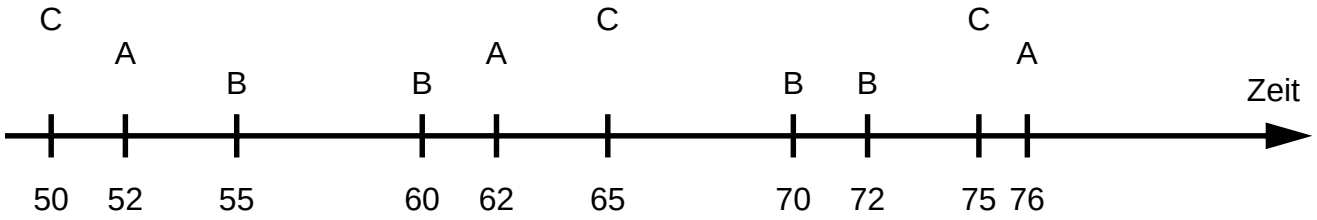
LRU-K

- **Aufzeichnung der K letzten Referenzzeitpunkte** (pro Seite im DB-Puffer)
  - Aufwendigere Aufzeichnung gewährleistet aktuelle Ersetzungsinformation; Methode benötigt kein explizites „Altern“ über Tuning-Parameter wie LRD-V2
  - Gegeben sei bis zum Betrachtungszeitpunkt t der Referenzstring $r_1, r_2, \ldots, r_t$. **Rückwärtige K-Distanz $b_t(P, K)$** ist die in Referenzen gemessene Distanz rückwärts bis zur K-jüngsten Referenz auf Seite P:

    $b_t(P, K) = x$, wenn r_{t-x} den Wert P besitzt und
    es genau K-1 andere Werte i mit $t-x < i \leq t$ mit $r_i = P$ gab.

    $b_t(P, K) = \infty$, wenn P nicht wenigstens K mal in $r_1, r_2, \ldots, r_t$ referenziert wurde

- **Beispiel (K=4)**



- **Zur Ersetzung** genügt es, die $b_t(P_i, K)$ der Pufferseiten zu berücksichtigen!
  - Sonderbehandlung für Seiten mit weniger als K Referenzen erforderlich
  - Wie hängt LRU-K mit LRD zusammen? Approximation der Referenzdichte?

- **LRU-2 (d.h. K=2) stellt i. allg. beste Lösung dar**[1]
  - ähnlich gute Ergebnisse wie für K > 2, jedoch einfachere Realisierung
  - Verfahren reagiert schneller auf Referenzschwankungen als bei größeren K

1. O‘Neil, E.J., O‘Neil, P.E., Weikum, G.: The LRU-K Page Replacement Algorithm for Database Disk Buffering. Proc. ACM SIGMOD Conf. Washington. D.C. 1993. 297–306

Ersetzungverfahren – Einbezug von Kontextwissen

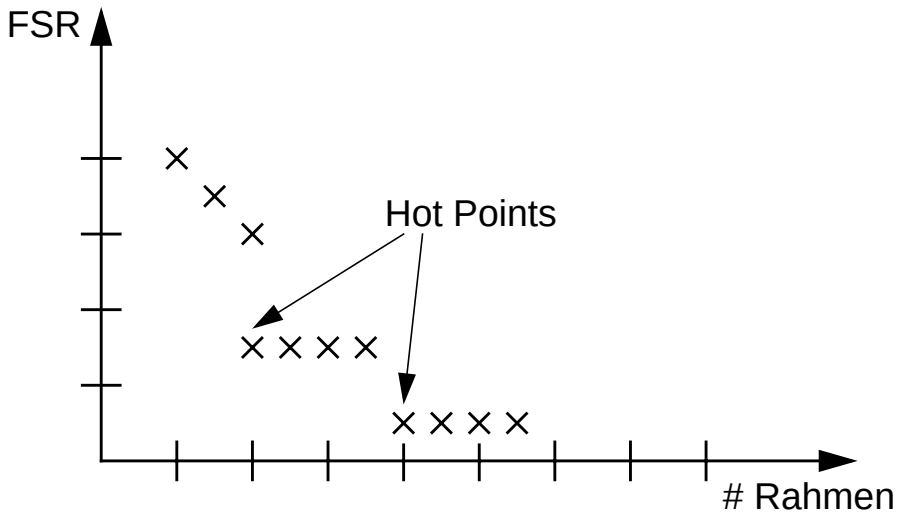
- **Ausnutzung von Kontextwissen bei mengenorientierten Anforderungen**

  ➥ Verbesserung in relationalen DBS möglich

- **Zugriffspläne durch Anfrage-Optimierer**
  - Zugriffscharakteristik/Menge der referenzierten Seiten kann bei der Erstellung von Plänen vorausgesagt/abgeschätzt werden
  - Zugriffsmuster enthält immer Zyklen/Loops (mindestens Kontrollseite – Datenseite, *nested loop join* etc.)
  - Kostenvoranschläge für Zugriffspläne können verfügbare Rahmen berücksichtigen
  - Bei Ausführung wird die Mindestrahmenzahl der Pufferverwaltung mitgeteilt

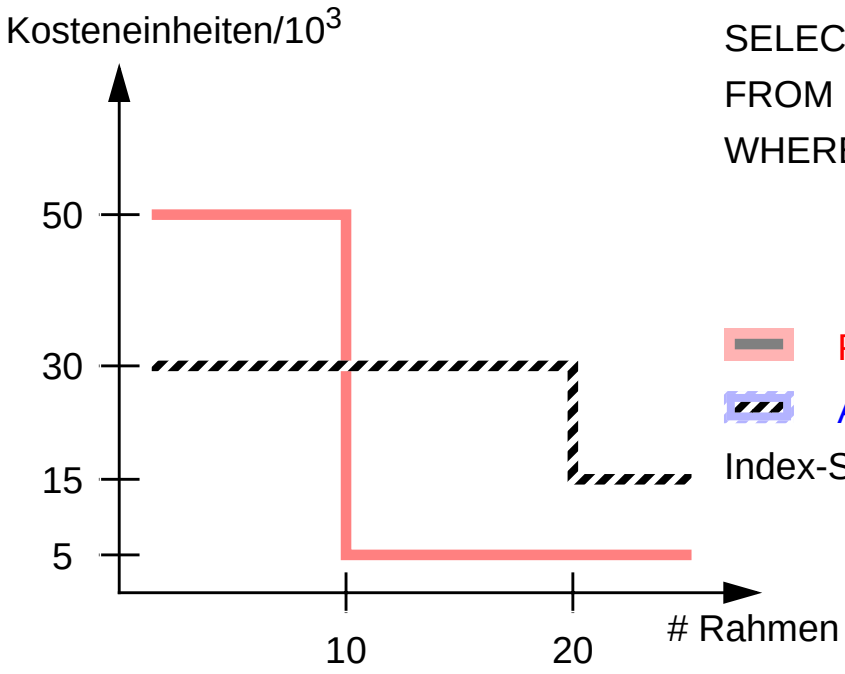
- **Hot Set: Menge der Seiten im Referenzzyklus**

  Prinzipieller Verlauf der Fehlseitenrate (**FSR**) bei speziellen Operationen



„Hot Set“-Modell

- ***Hot Point*:**
  abrupte Veränderung in der FSR, z. B. verursacht durch Schleife beim Verbund
- ***Hot Set Size* (HSS):**
  größter *Hot Point* kleiner als der verfügbare DB-Puffer
- Anfrage-Optimierer berechnet HSS für die verschiedenen Zugriffspläne
  (Abschätzung der #Rahmen)
- Beispiel:




```
SELECT   *
FROM     ABT X, PERS Y
WHERE    X.ANR = Y.ANR
         AND ...
```

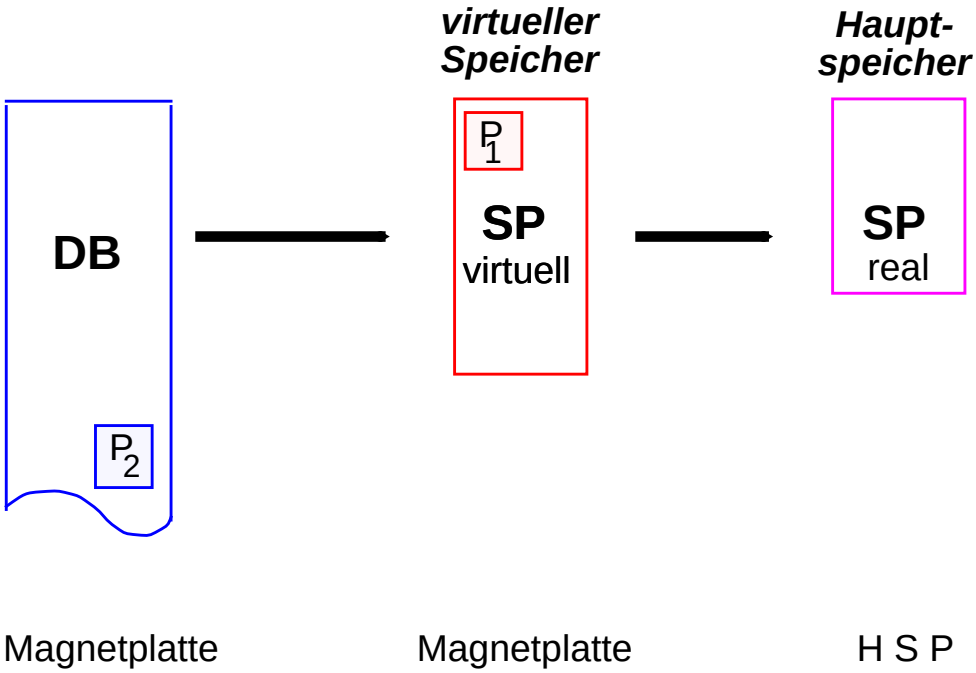
PERS in äußerer Schleife

ABT in äußerer Schleife

Index-Scan für beide Relationen

- **Anwendungscharakteristika**
  - Berücksichtigung der HSS in den Gesamtkosten
  - Auswahl abhängig von verfügbarer DB-Puffergröße
  - Bindung zur Laufzeit möglich

Seitenersetzung bei virtuellem Speicher



- **Page Fault:**

  $P_i(P_1)$ in SP virtuell, aber nicht in SP real (HSP)

- **Database Fault:**

  $P_i(P_2)$ nicht in SP virtuell,

  Seitenrahmen für P_i jedoch in SP real

- **Double Page Fault:**

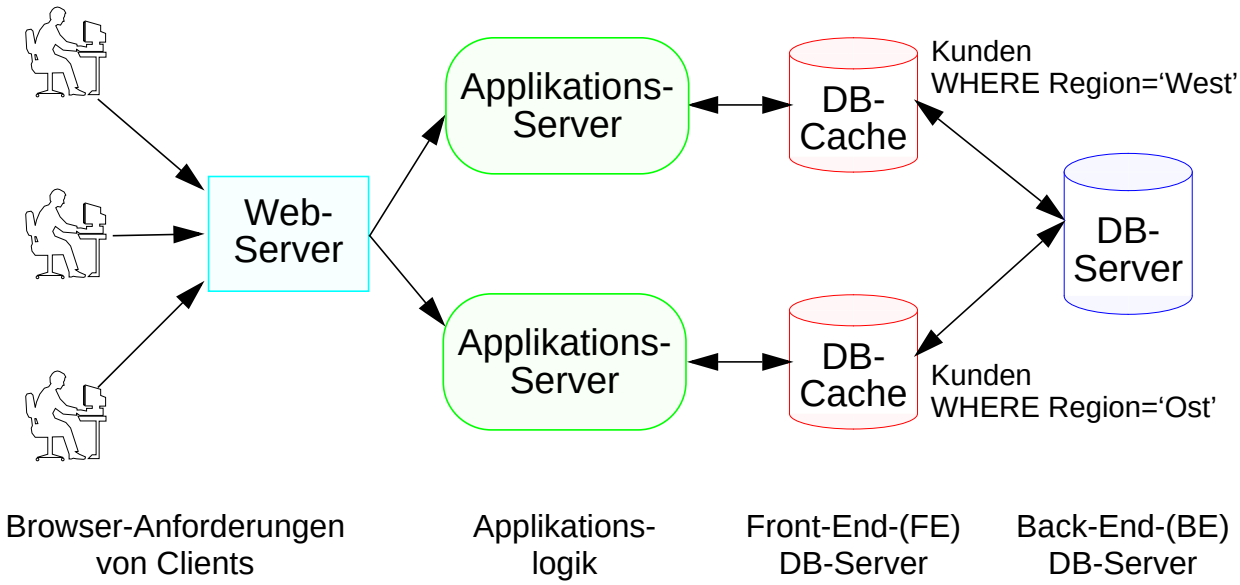
  $P_i(P_2)$ nicht in SP virtuell, ausgewählter

  Seitenrahmen nicht in SP real

DB-Caching

- **Ziel: Unterstützung von Web-basierten DB-Anwendungen**
  - durch Abwicklung von DB-(Teil-)Anfragen im Cache in AW-Nähe
    - Anfrageergebnis-Caching (query result caching)
    - On-demand Caching bei vorhandenen „passenden“ Satzmengen
  - Im DB-Cache muß Vollständigkeit (und Aktualität) der eingelagerten Satzmengen gewährleistet werden

- **Wichtige Caching-Verfahren**
  - **Deklaratives Caching** erfordert BE-Metadaten, um den Cache dynamisch zu laden (Caching mehrerer QRs in gemeinsamen Tabellen)
  - **On-demand Caching** verwendet BE-Metadaten und „Hinweise“ der TA, um Daten dynamisch in den Cache zu füllen oder zu ersetzen



- **DB-Caching „in der Nähe“ des Applikations-Servers**
  - Beschleunigung des lesenden DB-Zugriffs
  - (bislang noch) Weiterleitung von Änderungsanweisungen zum BE
  - Konsistenzprobleme

DB-Caching (2)

- **Ansätze**
  - Replikation
    - DBA definiert, was im Cache zu halten ist
    - FE-Tabellen spiegeln die entsprechenden BE-Tabellen wider
    - Volle DB- oder Tabellen-Replikation ist meist nicht wünschenswert
  - Materialisierte Sichten[1]
    - DBA spezifiziert Sichtdefinitionen für Sichten, die im Cache gehalten werden sollen
    - separate FE-Tabelle für jede Sicht
    - Was sind die „richtigen Sichten“?

  ➥ Wie läßt sich eine dynamische Anpassung des Cache-Inhalts an die TA-Last erreichen?

- **Deklaratives und/oder On-demand Caching[2]**
  - Beide Begriffe werden in der Literatur zur Klassifikation verwendet
  - keine strikte Unterscheidung, Verfahrensübergänge fließend
  - Beide Verfahren sind „dynamisch“ und wollen „adaptiv“ sein
  - Es werden BE-Metadaten und „Hinweise“ gebraucht (deklarativ), um zu wissen, was im Cache zu speichern ist
  - Werden zu speichernde Daten nicht im Cache gefunden, wird die Anfrage in der BE-DB beantwortet. Zugleich werden die entsprechenden Daten in den Cache geladen, damit die Anfrage beim **„nächsten“** Mal im Cache beantwortet werden kann (on-demand)

---

1. Materialized Views werden (in IBM-Publikationen) auch Automated Summary Tables (AST) oder Materiakized Query Tables (MQT) genannt.
2. Wir behalten die englischen Beigriffe Cache, Cache Key, Cache Group usw. bei

DB-Caching (3)

- **DB-Caching — Was ist zu entscheiden?**
  - Was soll im Cache gehalten werden und wozu?
    - Anfrageergebnisse (einzelne Tabellen/Sichten):
      Sie lassen Anfragen zu, die Untermengen als Ergebnis haben!
    - Cache Groups (Cache-Gruppen), die aus mehreren „zusammen-hängenden“ Tabellen bestehen. Sie erlauben die Abwicklung von Anfragen mit einfachen Prädikaten und n Verbunden im Cache
  - Wie wird es spezifiziert?
    - Liste von Anfragen
    - Alle Anfragen, die eine spezifizierte Tabelle/Sicht betreffen
    - Spezifikation von Cache Groups durch sog. Cache Constraints;
      das sind Cache Keys und Referential Cache Constraints (RCCs)
  - Wann werden Daten in den Cache geladen?
    - vorab (statisch)
    - on-demand; d. h., nachdem spezifizierte Daten nachgefragt wurden
    - nach Analyse des Bedarfs?
  - Sind überlappende Daten im Cache zugelassen?
    - Überlappende Sichten oder Cache Groups mit gemeinsamen Tabellen werden disjunkt gespeichert
    - Problem: Caching + Replikation!
  - Wann werden Daten im Cache aktualisiert?
    - zeitgleich zu ihrer Aktualisierung in der BE-DB?
    - Relevante Änderungen werden innerhalb einer Zeitspanne δ propagiert
    - irgendwann
  - Wann werden Daten im Cache ersetzt bzw. invalidiert?
    - nie
    - bei Speichermangel im Cache
    - nach Ablauf eines Zeitintervalls ohne Referenzen

DBProxy

- **DBProxy-Ansatz**[1]
  - Daten werden persistent in den FE-DB-Servern gespeichert
  - Als Hinweise sind zu spezifizieren: common schema tables
  - Daten, die im Cache gehalten werden, sind durch eine Liste von Anfragen in einem Cache-Index beschrieben

  ➥ Jede Anfrage liefert genau eine Tabelle zurück!

  - Aus Gründen der Speicherplatzeffizienz und zur Vermeidung von Replikation im Cache werden Anfrageergebnisse, wenn möglich, in derselben FE-Tabelle gespeichert:
    - Anfragen über dieselbe BE-Tabelle
    - Verbund-Anfragen über dieselbe Menge von BE-Tabellen

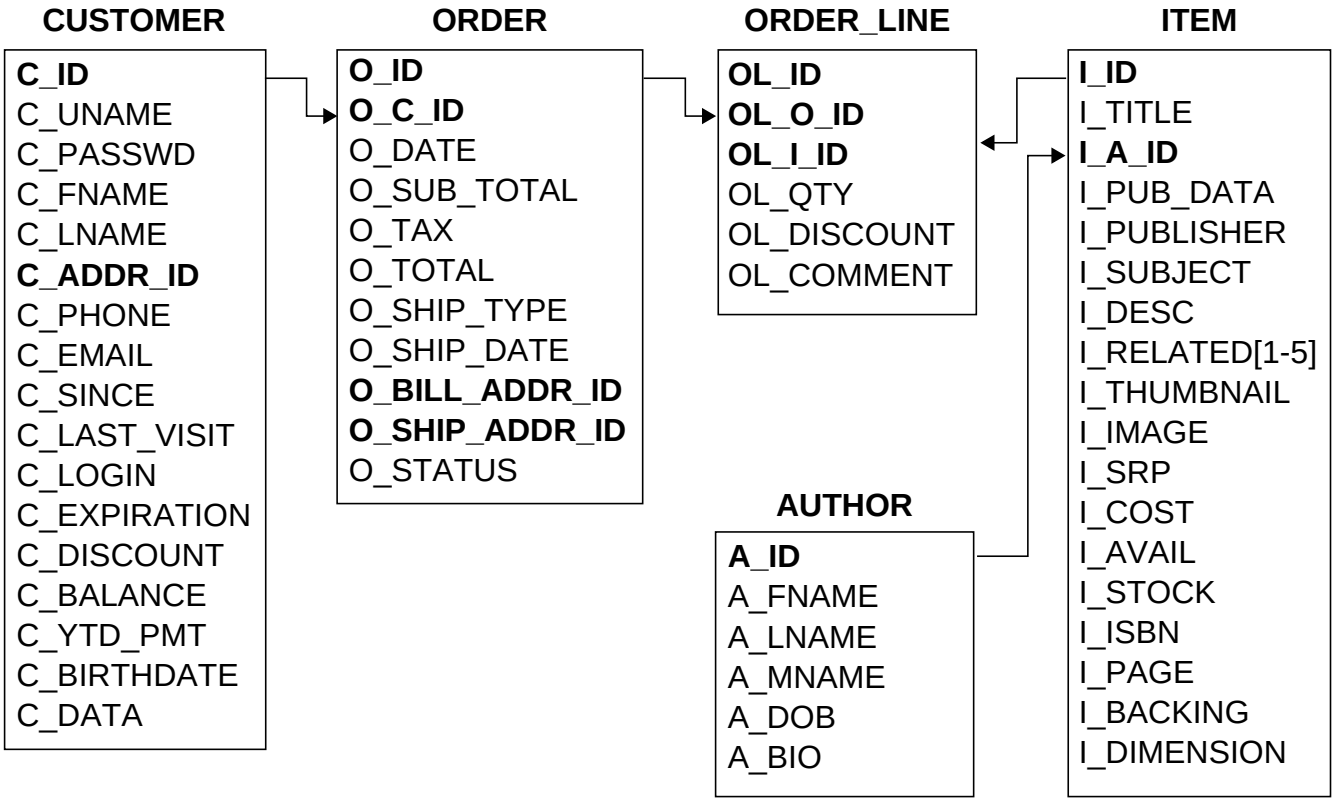
  ➥ Sonst entstehen Replikate im Cache!

  - Anfragen für eine FE-Tabelle beziehen sich auf unterschiedliche Spalten
    - Anfragen können in ihrem Ergebnis überlappen
    - Bei Anfragen, die nicht alle Spaltenwerte zurückliefern, sind die Spalten mit NULL („fake“ NULL values) aufzufüllen
    - Wie lassen sich „echte“ NULL-Werte darstellen?

1. Amiri, K., Park, S., Tewari, R., Padmanabhan, S.: DBProxy: A Self-managing Data Cache for Edge-of-Network Web Applications, in: Proc. CIKM’2002, pp. 177-185.

DBProxy (2)

- **Anwendungsbeispiel**
  - Vereinfachtes DB-Schema eines Web-Buchhändlers (nach TPC-W-Benchmark)



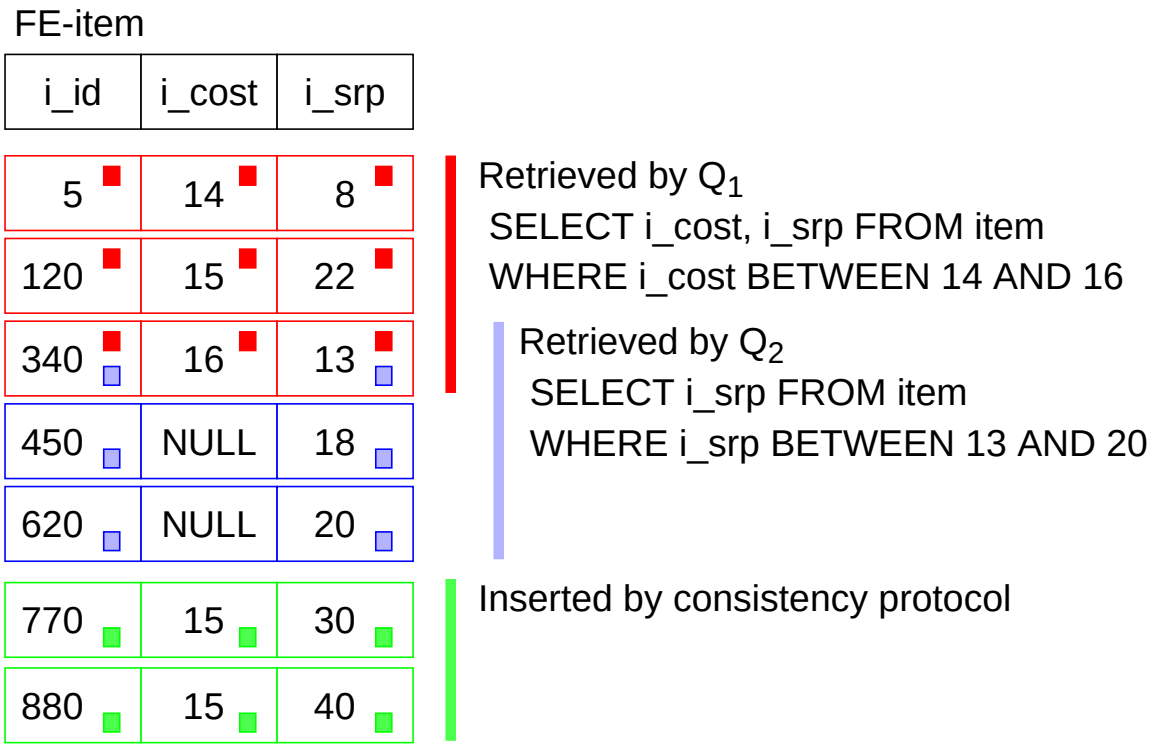
  - BE-item-Tabelle hat 18 Spalten mit i_id als Primärschlüssel

  - Mögliche Anfragen hinsichtlich Kosten und Verkaufspreis (srp: suggested retail price) auf Tabelle item

Q_A:	SELECT	i_avail, i_cost FROM item	WHERE i_cost < 5
Q_B:	SELECT	i_avail, i_cost FROM item	WHERE i_cost > 25
...		...	...
Q_N:	SELECT	i_srp, i_cost FROM item	WHERE i_srp BETWEEN 30 AND 65

DBProxy (3)

- **Verfahrensaspekte**
  - Belegung der ursprünglich leeren item-Tabelle im Cache nach Einfügung der Ergebnisse von Q_1 und Q_2
  - Anfragen werden so umgeschrieben, daß sie den Primärschlüssel i_id enthalten. So lassen sich Zeilenduplikate vermeiden



  - Vor Einfügen vom Q_2-Ergebnis ist zu prüfen,
    - ob FE-item mit Spalten zu erweitern ist
    - Optimierung: Definition einer „umfassenden“ Tabelle mit Vorabwissen
  - Speicherung verschiedener Anfragen in einer FE-Tabelle erzeugt unbelegte Spaltenwerte (NULL-Werte). Spätere Cache-Anfrage darf sie nicht benutzen
  - Einfügen von i_id=340 muß als Duplikat erkannt werden

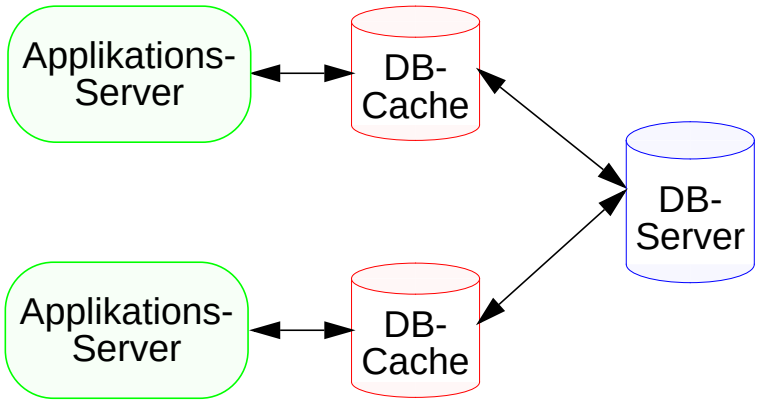
DBProxy (4)

- **Verfahrensaspekte (Forts.)**
  - Enthaltenseinstypen von Anfragen
    - vollständig enthalten in einem früheren Anfrageergebnis (Sicht von Cache-Prädikat Q_i)
    - enthalten in der Vereinigung von mehreren früheren Ergebnissen (Sichten von Q_i und Q_j)
    - nur teilweise enthalten in einer oder mehreren im Cache gehaltenen Sichten
  - Komplexer Matching-Algorithmus
    - Prädikate der im Cache gehaltenen Daten sind in einem Index gespeichert
    - Enthaltensein von Q_B:
      Ergebnis von Q_B ist enthalten in dem von Q_A, wenn das WHERE-Prädikat von Q_B das von Q_A für alle möglichen Werte von item logisch impliziert
    - Q_B.wherep => Q_A.wherep äquivalent zur Anweisung „Q_B.wherep AND (NOT (Q_A.wherep)) ist nicht erfüllbar“
    - (i_cost < 5 AND NOT (i_cost > 25))

➥ Folglich ist Q_B nicht in Q_A enthalten

DBProxy (5)

- **Verfahrensaspekte (Forts.)**
  - Aktualisierung
    - Wo wird aktualisiert?



    - Alle DB-Caching-Ansätze sind nur sinnvoll, wenn sich die DB-Daten nur langsam verändern
    - δ-Konsistenz wird gewährleistet: Relevante Änderungen in BE-item werden nach FE-item innerhalb einer Zeitspanne δ propagiert.
    - i_id = {770, 880} wurden später in die BE-DB eingefügt. Cache-Prädikat Q_1 verlangt das Propagieren dieser Sätze in die FE-DB

  - Ersetzung oder Invalidierung
    - Ersetzung von Q2 darf nur i_id = {450, 620} aus FE-item entfernen
    - i. allg. sehr komplex, da Satzmengen, die durch „überlappende“ Prädikate beschrieben werden, zu entfernen sind

DBCache

- **DBCache-Ansatz[1]**
  - Es sollen **Anfragen mit einfachen Prädikaten und n Verbunden** im Cache unterstützt werden. Das setzt voraus, daß der Cache-Mgr garantieren kann,
    - daß bei einer Anfrageauswertung alle Sätze, die ein gegebenes Prädikat erfüllen, sich in der betreffenden FE-Tabelle befinden
    - daß in den n FE-Tabellen alle zugehörigen Verbundpartner gespeichert sind

  ➥ Neue Herausforderung für Caching!

  - Wie wird die erste Anforderung spezifiziert?
    - Hinreichende Schema-Information von allen BE-Tabellen, von denen eine horizontale Partition als FE-Tabelle gespeichert werden soll
    - Einführung sog. **Cache Keys** (CK)

- **Cache Key**
  - kann für eine FE-Tabelle spezifiziert werden
  - bezieht sich auf eine Spalte und dient als **„Füllpunkt“**
  - besitzt die Eigenschaft „bereichsvollständig“ (domain complete, DC)
  - Mechanismen zur Einschränkung sind **lebenswichtig** (~ Stoppwortliste)

- **Definition: Bereichsvollständigkeit einer Spalte**

  Wenn ein Spaltenwert im Cache gefunden wird, garantiert der Cache-Mgr, daß alle Sätze mit diesem Wert sich im Cache befinden.
  UNIQUE-Spalten sind somit immer bereichsvollständig

  ➥ Folglich wird die Auswertung eines Prädikats <ColName> = <value> durch eine solche Spalte unterstützt!

---

1. Altinel, M., Bornhoevd, Ch., Krishnamurthy, S. Mohan, C., Pirahesh, H., Reinwald, B.: Cache Tables: Paving the Way for an Adaptive Database Cache, in. Proc. VLDB, Berlin, 2003.

DBCache (2)

- **Beispiel**
  - Tabelle CUST habe u. a. zwei UNIQUE-Spalten (U) Cnr und Cid sowie zwei Spalten CType und CLocation vom Typ NON UNIQUE (NU)
  - Im Cache seien für CUST CType und Cnr als Cache Keys deklariert
  - Belegung im BE:

BE_CUST	Cnr	CType	CLocation	Cid	...
	...	...	...		
	789	silver	SF	NULL	
	891	silver	LA	d07	
	333	platinum	SJ	a21	
	444	unspec.	LA	a07	
	123	gold	SJ	a14	
	456	gold	SF	b21	
	...	...	...		

  - Im Cache sei die zugehörige Tabelle FE-CUST zunächst leer. Eine Anfrage mit ’CType = gold’ wird im BE ausgewertet und führt zu folgender Belegung von FE-CUST:

FE_CUST	Cnr	CType	CLocation	Cid	...
	123	gold	SJ	a14	
	456	gold	SF	b21	

  - Erneute Anfragen mit ’CType = gold’ oder Cnr = 123’ oder Cnr = 456’ werden im Cache ausgewertet, weil die Spalten DC sind und die Werte im Cache gefunden werden
  - Achtung: Cid ist eine U-Spalte und deswegen implizit DC. Folglich wird eine Anfrage mit ’Cid = a14’ im Cache ausgewertet, da der Wert a14 im Cache gefunden wird. Eine Anfrage mit diesem Prädikat hätte jedoch nicht zum Laden des Cache geführt, da Cid kein Cache Key ist
  - Cache-Belegung nach einer Anfrage mit ’Cnr = 789’

FE_CUST	Cnr	CType	CLocation	Cid	...
	123	gold	SJ	a14	
	456	gold	SF	b21	
	789	silver	NY	NULL	
	891	silver	LA	d07	

DBCache (3)

- **CK-Regel**
  - Für eine FE-Tabelle dürfen n Cache Keys deklariert werden.
  - Höchstens ein Cache Key darf die Eigenschaft NU besitzen

FE_CUST	Cnr	CType	CLocation	Cid	...
	123	gold	SJ	a14	
	456	gold	SF	b21	

➥ Warum muß diese Einschränkung eingeführt werden?

- **Cache Groups**
  - Wie wird der Zusammenhang zwischen FE-Tabellen spezifiziert?
    - **Referential Cache Constraints** beziehen sich auf Paare von Spalten (in der Regel verschiedener Tabellen) und sind vom Typ U->U, U->NU, NU->U, NU->NU (oder 1:1, 1:n, n:1, n:m)
    - **Alle Sätze** im Cache erfüllen alle spezifizierten RCCs, d. h., wenn z. B. NU->NU (oder U->NU) zwischen den Spalten A und B von Tabelle S bzw. T spezifiziert ist, wird garantiert, daß alle T-Sätze mit einem Wert ($B=v_i$) sich im Cache befinden, sobald ein S-Satz mit dem Wert ($A=v_i$) dort ist

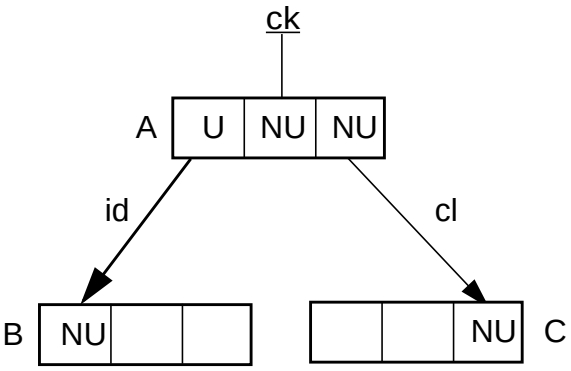
➥ Bedingung läßt sich nicht einschränken, da bei einem Verbund alle Verbundpartner da sein müssen!

DBCache (4)

- **Cache Groups (Forts.)**
  - Zusammenhang zwischen FE-Tabellen
    - Wichtigste Fälle:
      Beziehungen zwischen Primär-/Fremdschlüssel oder Owner-Member
    - **U->NU:**
      Wenn ein PS-Wert im Cache gefunden wird, garantiert der Cache-Mgr, daß **alle Sätze** mit dem gleichen FS-Wert im Cache sind
    - **NU->U:**
      Wenn ein FS-Wert im Cache gefunden wird, ist der zugehörige Satz mit dem gleichen PS-Wert auch da
  - „Verknüpfung“ von Tabellen im Cache durch RCCs
    - FE-Tabelle mit einem Cache Key ist Wurzel-Tabelle einer Cache Group
    - Sie kann mit anderen FE-Tabellen ohne Cache Key über RCCs verknüpft sein
    - So lassen sich **Cache Groups** bilden, die Verbundoperationen und, im Fall von NU->NU, Kreuzprodukte im Cache unterstützen

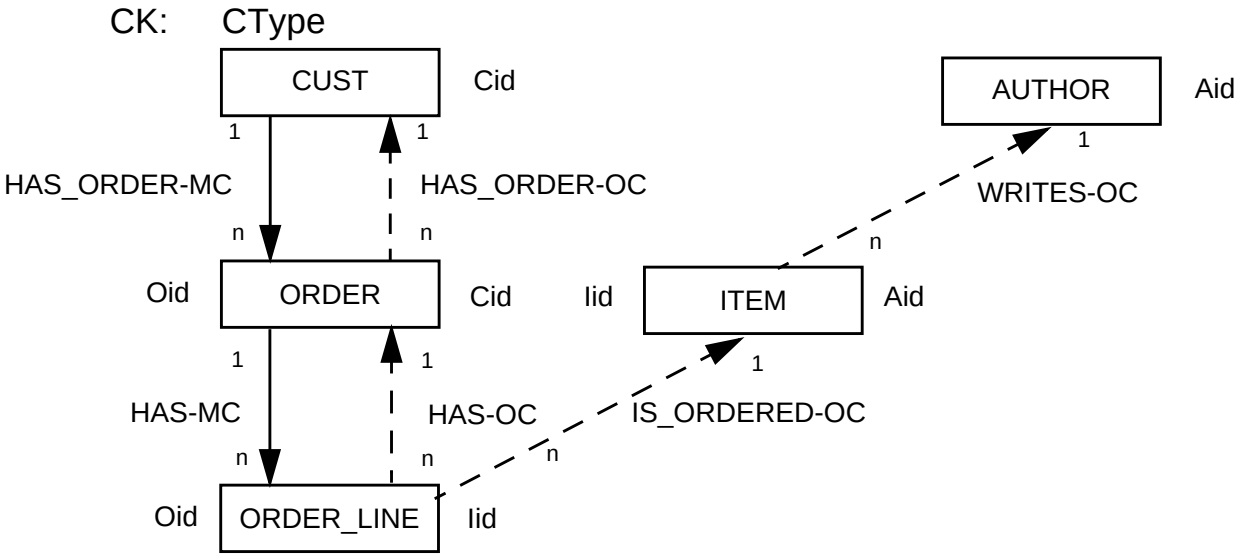
DBCache (5)

- **Prinzip**
  - Definition einer Cache Group über drei Tabellen A, B, C und mit A.ck als Cache Key
  - „Wertbasiertes Tabellenmodell“ im Cache:
    - RCCs sind A.id->B.id (U->NU) und A.cl->C.cl (NU->NU)
    - RCCs können, müssen aber keine entsprechenden Beziehungen in der BE-DB besitzen



    - Was passiert, wenn ein Anfrageprädikat ’A.ck = value’ bei leerem Cache ausgewertet wird?

- **Beispiel**
  - Als BE-DB werde die des Web-Buchhändlers verwaltet (mit ähnlichen Namen)
  - Cache Group: MC entspricht (U->NU)- und OC (NU->U)-Beziehung



Zusammenfassung

- **Referenzmuster in DBS sind Mischformen**
  - sequentielle, zyklische, wahlfreie Zugriff
  - Lokalität innerhalb und zwischen Transaktionen
  - „bekannte“ Seiten mit hoher Referenzdichte
- **Ohne Lokalität ist jede Optimierung** der Seitenersetzung sinnlos (~ RANDOM)
- **Suche im Puffer durch Hash-Verfahren**
- **Speicherzuteilung:**
  - global ⇒ alle Pufferrahmen für alle Transaktionen (Einfachheit, Stabilität, ...)
  - lokal ⇒ Sonderbehandlung bestimmter TAs/Anfragen/ DB-Bereiche
- **Behandlung geänderter Seiten:**
  NOFORCE, asynchrones Ausschreiben
- **Seitenersetzungsverfahren**
  - „zu genaue“ Verfahren sind schwierig einzustellen (⇒ instabil)
  - Nutzung mehrerer Kriterien: Alter, letzte Referenz, Referenzhäufigkeit
  - CLOCK ~ LRU, aber einfachere Implementierung
  - GCLOCK, LRD, LRU-K relativ komplex
  - LRU-2 guter Kompromiß; vorletzter Referenzzeitpunkt bestimmt „Opfer“
- **Erweiterte Ersetzungsverfahren**
  - Nutzung von Zugriffsinformationen des Anfrage-Optimierers
  - „Hot Set“-Modell
- **Double-Paging sollte vermieden werden**
- **DB-Caching**
  - will Skalierbarkeit und Leistungsverhalten bei Web-Anwendungen verbessern
  - Ansätze wie DBProxy und DBCache müssen Praxistauglichkeit noch erweisen