Realization of DBS

4. DB Buffer Management

Theo Härder
www.haerder.de

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

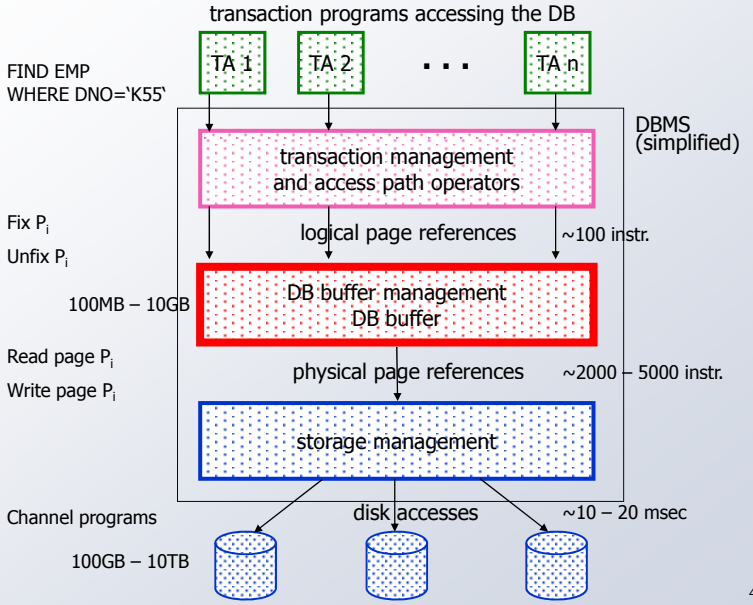
Technische Universität Kaiserslautern

Realization of DBS

DB Buffer Management – Contents

- **Goal: realization of efficient, page-based in-memory processing**
  - maximum possible avoidance of physical I/O
  - replacement algorithms without and with context knowledge
- **Role of DB buffer management**
  - course of access to the DB buffer
  - comparison with similar functionality in operating systems (OS)
- **Locality**
  - various measures for locality
  - characterization by LRU-stack-depth distributions and reference density
- **Memory allocation and search in the DB buffer**
- **Page replacement algorithms**
  - classification of replacement algorithms
  - LRU, CLOCK, GCLOCK
  - LRD, LRU-K ...
- **Replacement algorithms – utilization of context knowledge**
- **DB Buffer Management – further problems**

Role of DB Buffer Management in a DBMS



Page Reference Strings (PRSs)

- **Every data request is a *logical page reference***

  ➔ **Task of DB buffer management: minimization of *physical page references***

- **PRS = $<r_1, r_2, \ldots r_i, \ldots r_n>$ with $r_i = (T_i, D_i, P_i)$**

  T_i accessing transaction
  D_i referenced DB partition
  P_i referenced DB page

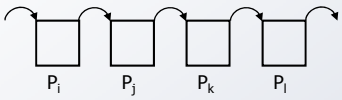
- Determination of windows in PRS w.r.t. specific transactions, transaction types, and DB partitions are useful for the analysis of the reference behavior

- How can we use reference string information for
  - characterization of reference behavior?
  - determination of **locality and sequentiality**?
  - support of an effective page replacement?

Properties of DB Reference Strings

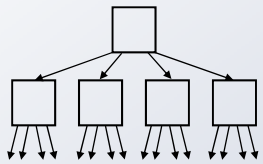
- **Typical reference pattern in DBS**

1. Sequential search



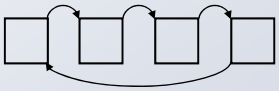
Example: scanning of entire record types (tables)

2. Hierarchical paths



Example: search using B*-trees

3. Cyclic paths



Example: processing of sets ((1:n)-relationships), search in DBTT/data pages

Comparison with OS Functionality

- Replacement algorithms of DB buffer are implemented in SW – page replacement in address spaces of Virtual Storage is HW supported
- **Page reference vs. addressing**
  after a Fix call, a DB page can be referenced several times until Unfix

  ➔ different page reference behavior
  ➔ other replacement algorithms?
- **Can OS file buffers be used as DB buffers?**

  1. **Access to file buffer is expensive (SVC: *supervisor call*)**
  2. **DB-specific reference patterns cannot be exploited systematically**
     for example, OS replacement algorithms are not aligned to cyclic sequential or arborescent access sequences
  3. **Normal file systems do not offer a suitable interface for prefetching**
     based on page contents or reference patterns, DBMS are enabled to predict the reference behavior (e.g., for table scans); prefetching may enormously enhance the performance
  4. **Selective writes of pages at specific points in time (e. g., for logging) are not always possible in existing file systems**

➔ DBMS must implement own buffer management, 64-bit architectures make not a difference!

Sequentiality

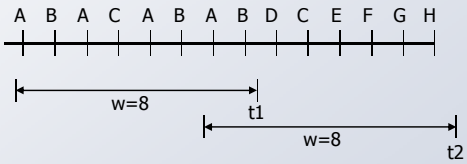
- PRSs typically show phases of sequentiality and locality
- Sequential reference sequence (SRS):
  two subsequent references r_i and r_{i+1} belong to a sequential reference sequence, if
  $P_{i+1} - P_i = 0$ or 1 i. e., subsequent accesses reference adjacent DB pages
- **Algorithm**
  - the page reference string is completely scanned; alternatively, the sequence of arriving references can be analyzed
  - as long as the above condition is satisfied, all consecutive references belong to an SRS; otherwise, a new SRS starts
- Length of a sequential reference sequence (LRS):
  - LRS is the number of different pages referenced in an SRS
  - PRS-1: A A B B D E E F F H contains
    (AABB) with LRS(1) = 2, (DEEFF) with LRS(2) = 3
    and (H) with LRS(3) = 1
- Measure for sequentiality:
  - cumulative distribution of SRS lengths LRS(i)
    S(x) = Pr(SRS length <= x)
  - PRS-1 yields: S(1)=0.33, S(2)=0.67, S(3)=1.0
- Sequentiality allows optimization by (asynchronous) prefetching of pages

Locality

- Enhanced reuse probability for pages just referenced
- **Fundamental prerequisite for**
  - effective DB buffer management (page replacement)
  - use of storage hierarchies
- **How can locality be measured?**

Working-Set model

reference string



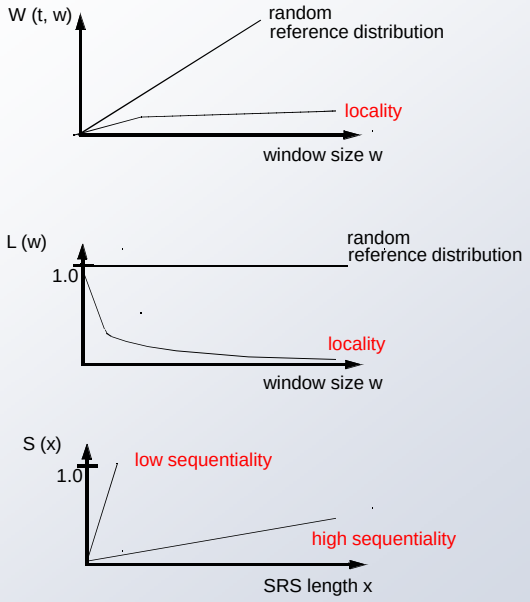
window size w

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

current locality at time t: $AL(t,w) = \frac{W(t,w)}{w}$

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

Sequentiality vs. Locality



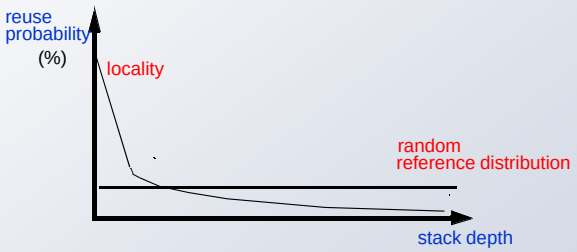
Relative Reference Matrix (DOA Load)

~ 17 500 transactions, 1 million page references on ~ 66 000 pages

	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
% referenced	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-stack-depth Distribution

- **How can locality be characterized?**
  - LRU-stack-depth distribution provides a measure for locality (more precise than Working-Set approach)
  - LRU-stack contains all already referenced pages in the sequence of their "age to their most recent reference"
- **Determination of stack-depth distribution**
  - a counter is maintained per stack position
  - re-reference of a page increases the counter for the resp. stack position



➔ **Counter values correspond to reuse frequency**
when applying LRU page replacement, the hit rate (resp. page fault rate) for a specific buffer size can be directly determined from the stack-depth distribution

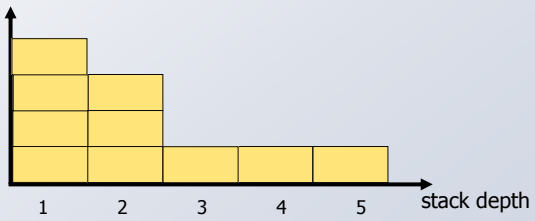
Example: Determination of stack-depth Distribution

Reference string: A B A C A A A D D E C D E A E

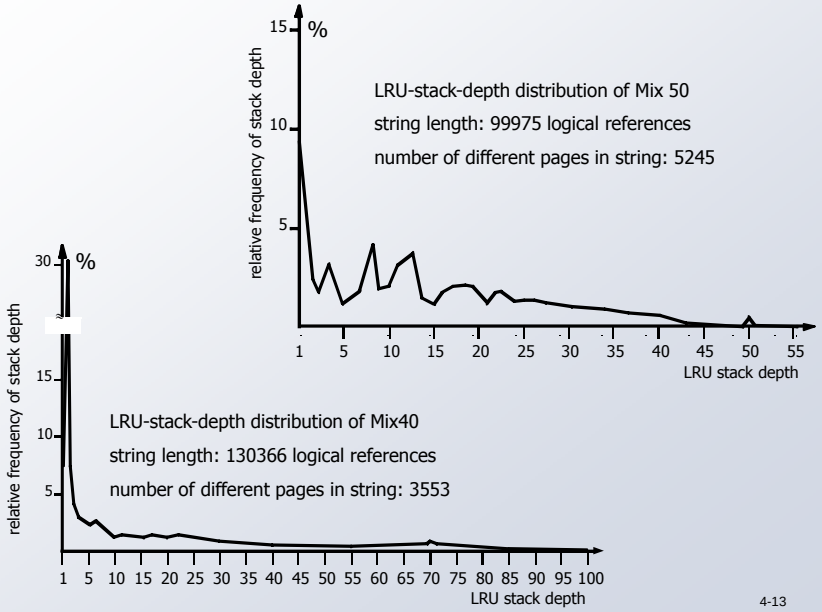
LRU stack:

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

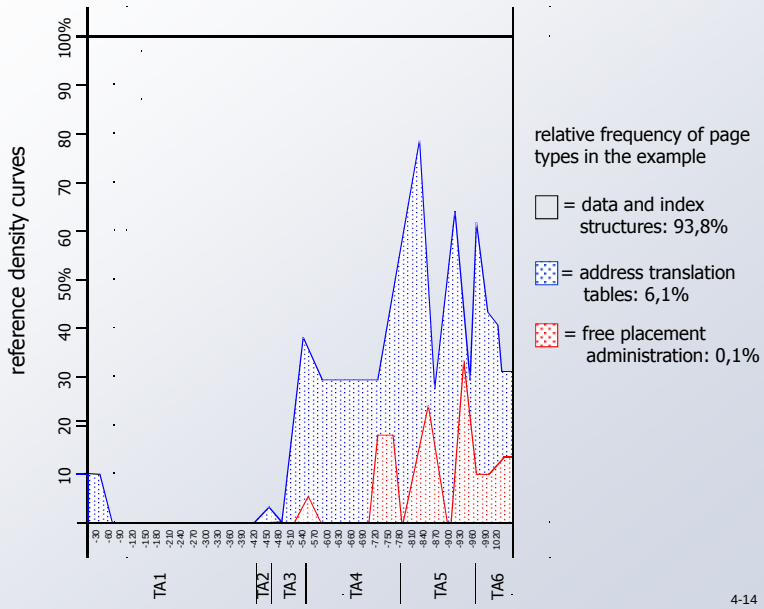
Stack-depth distribution



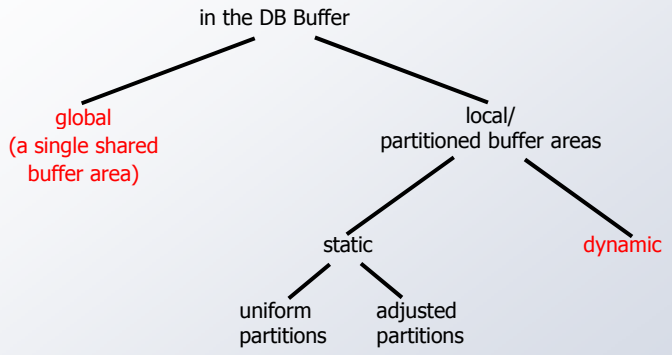
Real LRU-stack-depth Distributions



Reference Density Curves



Memory Allocation



Choice of partitioning:

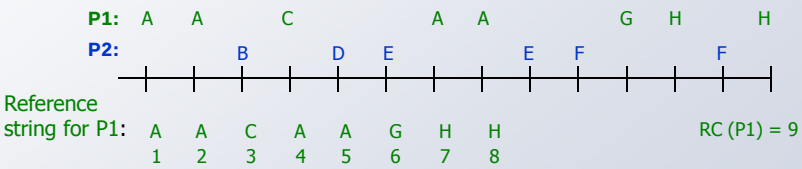
- own buffer area per transaction
- TA-type-related buffer areas
- page-type-related buffer areas
- DB-(partition)-specific buffer areas

Realization of DBS

Role of DB buffer mgmt
Locality
Memory allocation & search
Page replacement algorithms
On-demand replacement
Context knowledge
Variable-length pages

Dynamic Buffer Allocation – Working-Set Approach

- Per buffer partition P, the Working Set (WS) should be kept in the buffer; pages not belonging to the Working Set can be replaced
- When a page fault condition occurs, the Working Set must be known to determine the replacement candidate (victim)
  - window size per partition: w(P)
  - reference counter per partition: RC(P)
  - least reference time for page i: LRT(P, i)
  - replaceable are those pages, for which RC(P) – LRT(P, i) > w(P)
- Window size is critical parameter -> danger of thrashing



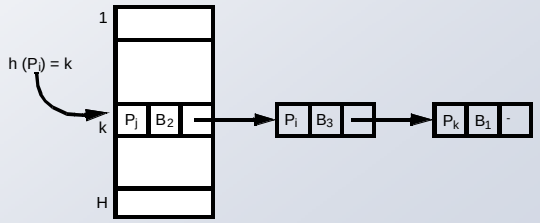
LRT (P1, H) = 8

LRT (P1, A) = 5

LRT (P1, C) = 3

Search in the DB Buffer

- Sequential search of buffer frames
  - very high search overhead
  - danger of many paging exceptions in Virtual Storage systems
- Use of auxiliary structures (entry per buffer frame)
  1. **unsorted or sorted table**
  2. **table with chained entries**
     - lower update overhead
     - allocation in LRU sequence possible
  3. **search trees (e. g., AVL trees, m-way trees)**
  4. **hash table with overflow chain**
     - **best solution**



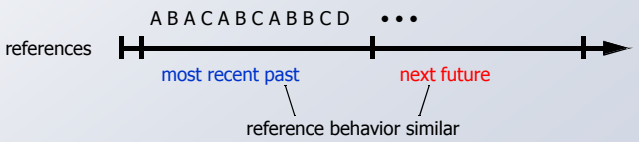
Page Replacement Algorithms

- **Classification**

method classes

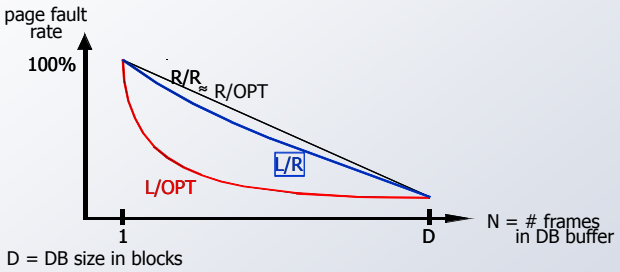
preplanning	**prefetching**	**demand fetching**
Program analysis, Pre-analysis of data demand	physical data structuring, clustering, processing knowledge	no pre-actions
➔ high fault rate, imprecise super-sets	➔ data model related, (hierarchical), speculative decisions	➔ locality preservation in DB buffer

- **Basic assumption for replacement algorithms**



Reference Behavior and Replacement Algorithms

- **Reference behavior in DBS**
  - **typically high** locality: optimization through replacement algorithms
  - sometimes sequentiality or random workload (RANDOM references)
- **Fundamental relationships** determining the page fault rate



- **Combinations**

references:	random	random	locality	locality
replacement:	RANDOM	OPT	RANDOM	OPT

➔ Borderline cases of reference behavior and replacement algorithms reveal optimization opportunities

Dealing with Modified Pages in the DB buffer

- Replacement of a modified page requires its prior (synchronous) propagation into the DB
  ➔ response time degradation
- **Dependency to the chosen propagation strategy:**

Force: all updates of a transaction have to be propagated not later than EOT into the DB („write-through")

+ always unmodified pages available for replacement, in general
+ simplified recovery (after crash, all updates of committed transactions are already propagated into the DB)
- high I/O overhead
- large response time increase for writer transactions

NoForce: no propagation of updated upon EOT („deferred write-back")

+ page can be modified several times before propagation applies (smaller I/O overhead, improved response times)
+ anticipatory (asynchronous) propagation of modified pages enables also in case of NoForce to predominantly replace unmodified pages

➔ **Synchronous DB writes can be avoided as far as possible**

Criteria for the Replacement of Buffer Pages

Algorithm	Criteria: age	most recent reference	reference frequency	other criteria
OPT	-	-	-	pre-knowledge
RANDOM	-	-	-	---
LFU	-	-	x	---
FIFO	x	-	-	---
LRU	-	x	-	---
CLOCK	x	x	-	---
GCLOCK				
LRD (V1)				
LRD (V2)				
LRU-K				

Least Frequently Used and First-In First-Out

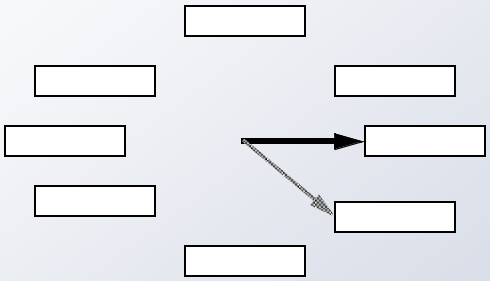
- **Algorithm LFU**
  - Reference counter per page is incremented upon each page reference
  - Replacement of page having the lowest reference frequency

RC	
2	
4	
1	
3	
3	
6	
1	
3	

➔ **Age of a page is not considered !**

Least Frequently Used and First-In First-Out

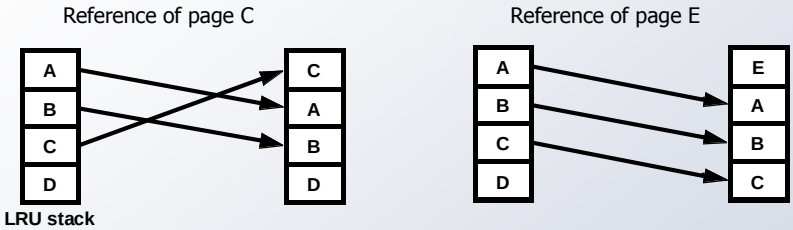
- **Algorithm FIFO**
  - Oldest page in the DB buffer is replaced
  - References while staying the buffer are not considered



➔ **Suitable only for strict sequential reference behavior**

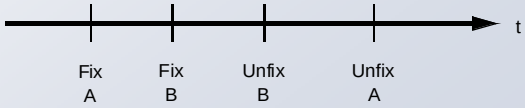
Least Recently Used (LRU)

- **Example (buffer size 4)**



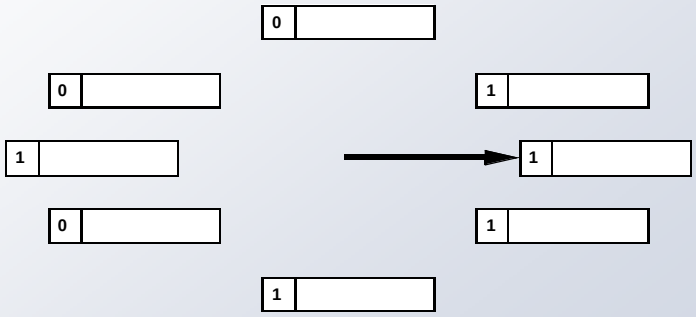
- **Distinction between**

  Least Recently Referenced and
  Least Recently Unfixed



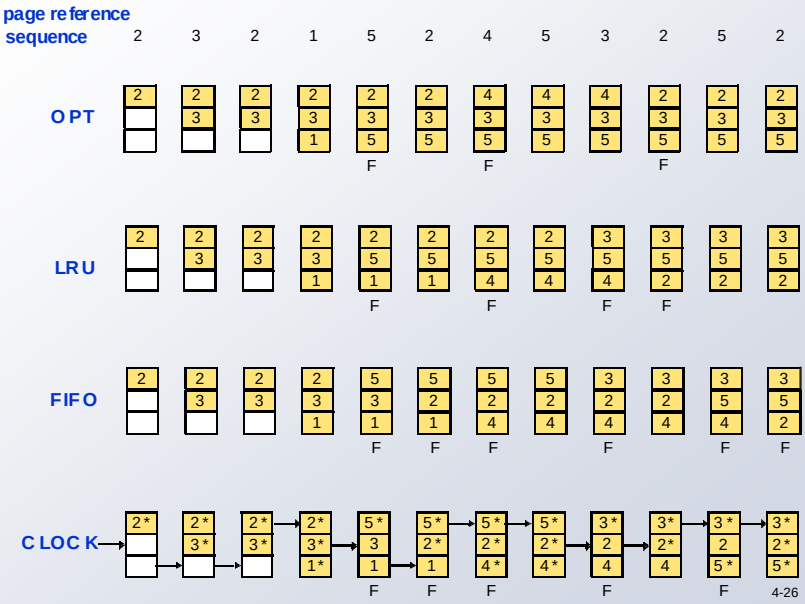
Clock (Second Chance)

- **Algorithm**
  - Extension of FIFO
  - Reference bit per page, which is set (1) upon each reference to the page
  - Replacement only applies when bit is already reset (0); otherwise, the bit is reset



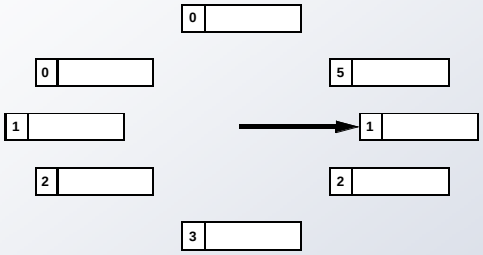
➔ **approximate consideration of the most recent reference time**

Page Replacement Algorithms - Example



GCLOCK (Generalized CLOCK)

- **Algorithm**
  - Reference counter (RC, instead of reference bit) is maintained per page
  - Replacement only if a page has counter value 0
  - Otherwise, the counter is decremented and the next page is inspected



	data	index	addr-tab.
F_i			
R_i			

Versions of weighted increments

V1: $RC(j) = F_i$
$RC(j) = RC(j) + R_i$

V2: $RC(j) = F_i$
$RC(j) = R_i$

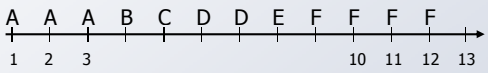
- **Parameters of the algorithm**
  - Initial values for reference counters (F_i)
  - Choice of the decrement value
  - Counter is incremented upon a re-reference R_i
  - Use of page-type of page-specific weights

Least Reference Density (LRD)

- **Algorithm**
  - If a page has to be replaced, reference density is determined for all pages in the DB buffer
  - Reference density = reference frequency for a given reference interval
  - Replacement candidate is page having lowest reference density
- Variant 1: reference interval corresponds to the age of a page
- **Computation of the reference density**

Global counter GC: total number of all references
Arrival time AT: GC value when the page enters the buffer
Reference counter RC

Reference density $RD(j) = \frac{RC(j)}{GC - AT(j)}$



	RC	AT	RD
A	3	1	
B	1	4	
C	1	5	
D	2	6	
E	1	8	
F	4	9	

Least Reference Density (2)

- **Variant 2: constant interval size**
  - artificial aging of pages: older references get a smaller weights, whenever their reference density is determined
  - **periodical reduction** of reference counters to reduce the influence of older references
  - reduction of RC by division or subtraction:

$$RC(i) = \frac{RC(i)}{K1} \quad (K1>1) \quad \text{or} \quad RC(i) = \begin{cases} RC(i) - K2 & \textit{if} \quad RC(i) - K2 \geq K3 \\ K3 & \textit{otherwise} \quad (K2>0, K3 \geq 0) \end{cases}$$

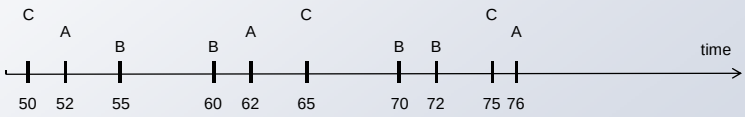
before	t1	t2	t3
RC(A)	3	2	
RC(B)	1	0	
RC(C)	0	1	
RC(D)	0	2	
RC(E)	0	1	
RC(F)	0	0	

Realization of DBS

Role of DB buffer mgmt
Locality
Memory allocation & search
Page replacement algorithms
On-demand replacement
Context knowledge
Variable-length pages

LRU-K

- **Recording of the K past reference times** (per page in the DB buffer)
  - More expensive recording, method does not need explicit "aging"
  - At inspection time t, the reference string $r_1, r_2, \ldots, r_t$ be given. **Backward K-Distance $b_t(P, K)$** is the distance measured in references backwards to the K-recent reference onto page P:
  - **$b_t(P, K) = x$,** if r_{t-x} owns value P and there exist exactly K-1 other values i ($t-x < i \leq t$) with r_i owning P
  - **$b_t(P, K) = \infty$,** if P is not referenced at least K times in $r_1, r_2, \ldots, r_t$
- Example (K=1, 2, 3)



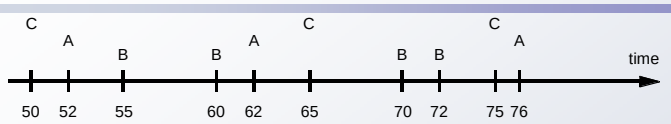
K=1 K=2 K=3 K=4

b_{100} (A,K) =

b_{100} (B,K) =

b_{100} (C,K) =

LRU-K (2)



- $b_t(P_i, K)$ of the buffer pages are needed to select a replacement candidate!
  - Special handling for pages with $b_t(P, K) = \infty$
  - How does LRU-K correspond to LRD? Approximation of reference density?

- LRU-2 (i.e., K=2) represents the best solution, in general[3]
  - Results similarly good as for K > 2, however simpler realization
  - Faster reaction in case of reference variations than for larger K

3) 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

Simulation of Page Replacement Algorithms

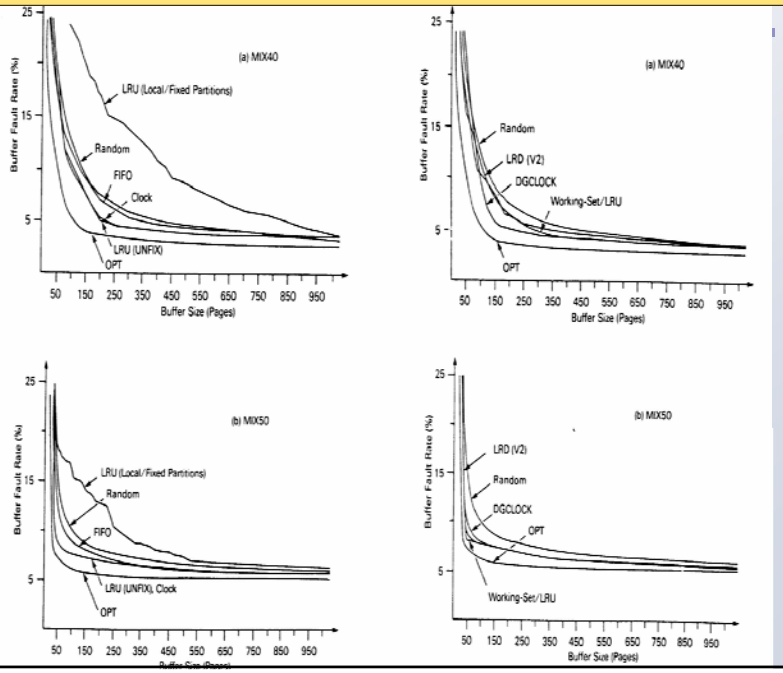
- **Characteristics of DB, TA load and logical page reference strings**

		MIX40	MIX50
Total number of pages (school-DB)		30,000	
Number of different pages in the string		3,553	5,245
Number of logical references		130,366	99,975
Number of page modifications		9,378	2,865
Number of pages being fixed*	maximum	11	10
	average	4.61	6.26
Percentage of references with FIX-duration	= 1	71%	41%
	2-10	22%	41%
	>10	7%	18%
FIX-duration (in logical references)	maximum	1,786	-
	average	4.62	6.26
Percentage of pages of a given type	FPA	0.1%	
	DBTT	6.1%	
	USER	93.8%	
Percentage of references to page-types	FPA	0.9%	0.1%
	DBTT	9.4%	21.7%
	USER	89.7%	78.2%
Percentage of references to most frequently referenced pages (hot spot pages)	1.	12.6%	3.3%
	2.	9.8%	0.5%
	3.	4.8%	0.4%
Relative frequency distribution of references to the other pages	>0.9%	5	1
	0.9%-0.1%	195	293
	0.1%-0.03%	1,606	2,741
	< 0.03%	1,747	2,208
Number of references to shared pages (concurrently fixed)*		1,175	359
cold start buffer fault rate in %		2.72	5.24
number of executed transactions	total	262	39
	parallel:max.	8	8
	avg.	6.31	6.29

*measured with a buffer size of 128 pages

Fig. 12: Characteristics of the DB, transaction load and logical reference strings.

Simulation Results



Replacement Algorithm – Use of Context Knowledge

- Problems for LRU-based methods
  - T1: long sequential scan with very fast page requests
    effects on T_i: pages of T_i are replaced if it proceeds with "slow" requests – even showing high reference locality

    T1: A B C D E F G H

    T2: Z Z Z

  - cyclic referencing (*loop*) of a set of pages (# pages > # frames)

    LRU: A B C D E A B C D E

    MRU: A B C D

    ➔ **internal thrashing**

Replacement Algorithm – Use of Context Knowledge

- Problems for LRU-based methods (cont.)
  - T1: cyclic referencing of a page set (# pages < # frames) interference through T_i if faster requests occur (*stealing*)

T1: A B C D A B C

T2: S T V W X

➔ **external thrashing**

- Mechanisms against thrashing: WS model
  - scheduler tries to allocate at least $\sigma = W(t, w)$ frames for T_i
  - if loop > w, WS tries to reserve w frames;
    in case of a sequential scan, a single frame would suffice
  - WS model: expensive implementation

Replacement Algorithm – Use of Context Knowledge (2)

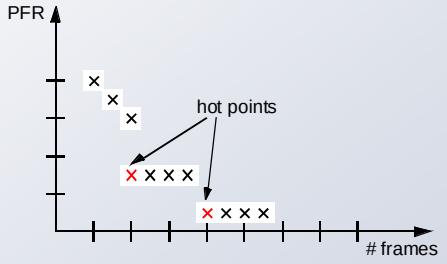
- Exploitation of context knowledge for set-oriented requests

  ➔ improvements in relational DBS possible

- Query execution plans (QEPs) by query optimizer
  - access characteristics/set of referenced pages can be predicted or estimated when QEPs are constructed
  - access pattern always contains cycles/loops
    (at least control page – data page: *nested loop join,* etc.)
  - cost estimate for QEPs can consider available frames
  - upon execution, min. # of frames for a QEP is communicated to buffer mgmt

- Hot Set: set of pages in reference cycle

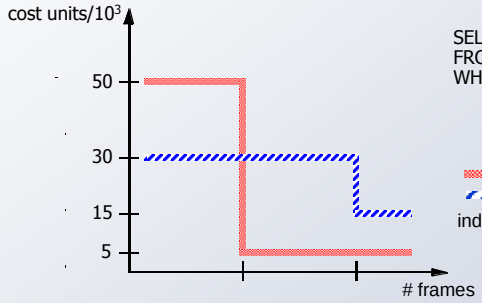
typical characteristics of page fault rate (PFR) for specific operations



Role of DB buffer mgmt
Locality
Memory allocation & search
Page replacement algorithms
On-demand replacement
Context knowledge
Variable-length pages

Hot Set Model

- ***Hot Point*** abrupt change in the PFR, e. g., caused by a loop in a join
- ***Hot Set Size*** (HSS): largest *Hot Point* smaller than the available DB buffer
- Query optimizer computes HSS for various QEPs (estimation of #frames)
- Example




```
SELECT   *
FROM     DEPT X, EMP Y
WHERE    X.DNO = Y.DNO
         AND ...
```

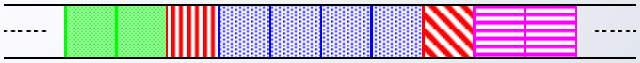
EMP in outer loop
DEPT in outer loop
index scan for both tables

- Application characteristics
  - consideration of HSS in the total costs
  - selection dependent on available DB buffer size
  - binding at runtime possible

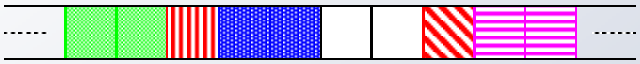
Role of DB buffer mgmt
Locality
Memory allocation & search
Page replacement algorithms
On-demand replacement
Context knowledge
Variable length pages

DB Buffer Management / Variable-Length Pages

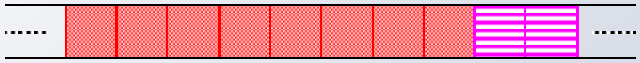
- Allocation example with variable-size pages and fixed-size frames



- Problems
  - buffer fragmentation



  - replacement of several pages to satisfy a new page request

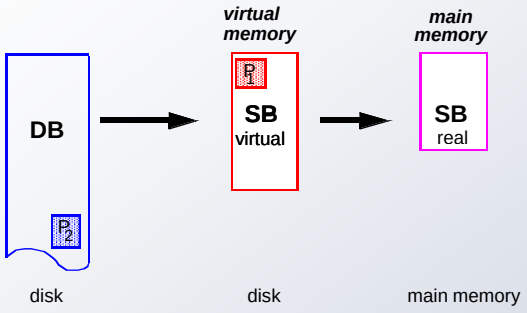


- Goals
  - maximization of buffer use (memory optimization)
  - minimization of I/O through use of locality in the reference behavior
- Proposal for an algorithm: VAR-PAGE-LRU* - what does LRU here mean?
- Further task: Caching and speculative prefetching of documents in storage hierarchies (tertiary storage)**

* A. Sikeler: VAR-PAGE-LRU ù A Buffer Replacement Algorithm Supporting Different Page Sizes. in Proc. EDBT, 1988, Springer, LNCS 303, pp. 336-351

** A. Kraiss, G. Weikum: Integrated Document Caching and Prefetching in Storage Hierarchies Based on Markov-Chain Predictions, in: VLDB Journal 7:3, pp. 141-162, 1998.

Page Replacement in Virtual Memory



- **Page fault**

  P_i (P_1) in system buffer (SB) virtual, but not in SB real (main memory)
- **Database fault**

  P_i (P_2) not in SB virtual,

  page frame for P_i however in SB real
- **Double page fault**

  P_i (P_2) not in SB virtual, chosen page frame not in SB real

Summary

- Reference patterns in DBS are hybrids
  - sequential cyclic, random access
  - locality within and between transactions, "known" pages with high reference density
  - detection of scan-based processing
- Without locality, optimization of page replacement is useless (~ RANDOM)
- Search in the buffer via hashing
- Memory allocation
  - global ⇒ all buffer frames for all transactions (simplicity, stability, ...)
  - local ⇒ special handling of specific TAs/queries/ DB areas
- Handling modified pages: NoForce, asynchronous propagation
- Page replacement algorithms
  - prefetching and pipelining using alternating buffers
  - "too accurate" methods are difficult to configure (⇒ instable)
  - exploitation of several criteria: age, most recent reference, reference frequency
  - CLOCK ~ LRU, but simpler implementation; GCLOCK, LRD, LRU-K relatively complex
  - LRU-2 good trade-off; most recent but one reference determines "victim"
- Extended replacement algorithms
  - Use of access information of the query optimizer
  - Hot Set model
  - Multiple buffer use to separate loads of different types, optimized for specific data types
- Double Paging should be avoided