Architecture of DB Systems
03 Data Layouts and Bufferpools

Matthias Boehm

Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMK endowed chair for Data Management

SCIENCE
PASSION
TECHNOLOGY

Last update: Oct 20, 2020

Announcements/Org

- **#1 Video Recording**
  - Link in TeachCenter & TUbe (lectures will be public)
  - Optional attendance (independent of COVID)

- **#2 COVID-19 Restrictions** (HS i5)
  - Max 25% room capacity (TC registrations)

max 18/74

- **#3 Open Position**
  - Student research assistant in ExDRa project
  - 10/20h per week

- **#4 Programming Projects**
  - Initial test suite, benchmark, and make file
  - Reference implementation Dexter released (you need to implement ./server.h)

Caching – An Old and Fundamental CS Concept

4.0. The Memory Organ

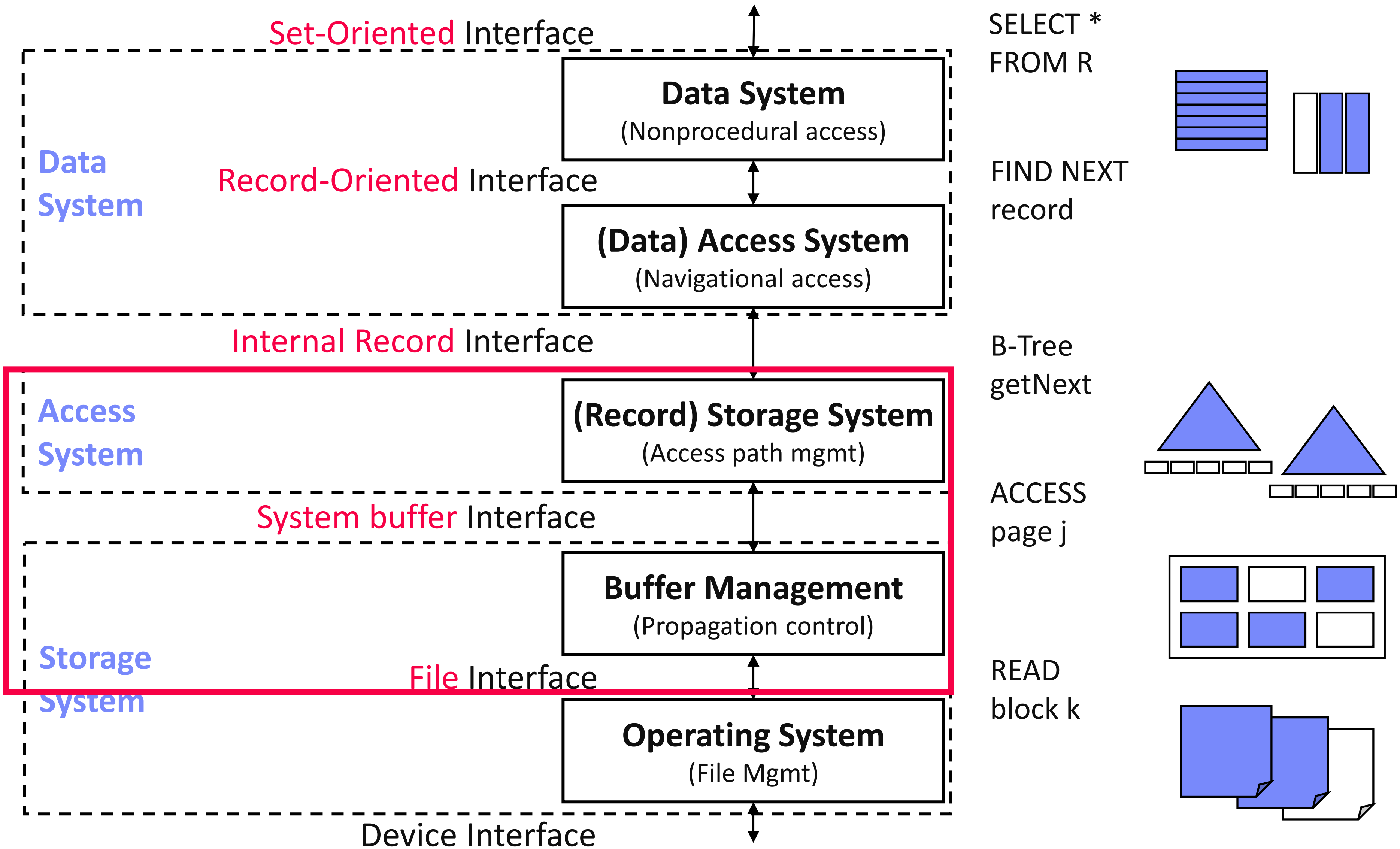
4.1. Ideally one would desire an indefinitely large memory capacity such that any particular 40 binary digit number or word would be immediately - i.e., in the order of 1 to 100 μs - available and that words could be replaced with new words at about the same rate. It does not seem possible physically to achieve such a capacity. We are therefore forced to recognize the possibility of constructing a hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.

[Arthur W. Burks, Herman H. Goldstine, John von Neumann: Preliminary Discussion of the Logical Design of an Electronic Computing Instrument, Part I, Vol. I, Report prepared for U.S. Army Ord. Dept., **28 June 1946**]

[**Credit:** Nimrod Megiddo and Dharmendra S. Modha (ARC paper)]

DBMS Architecture, cont.

[Theo Härder, Erhard Rahm: Datenbanksysteme: Konzepte und Techniken der Implementierung, **2001**]



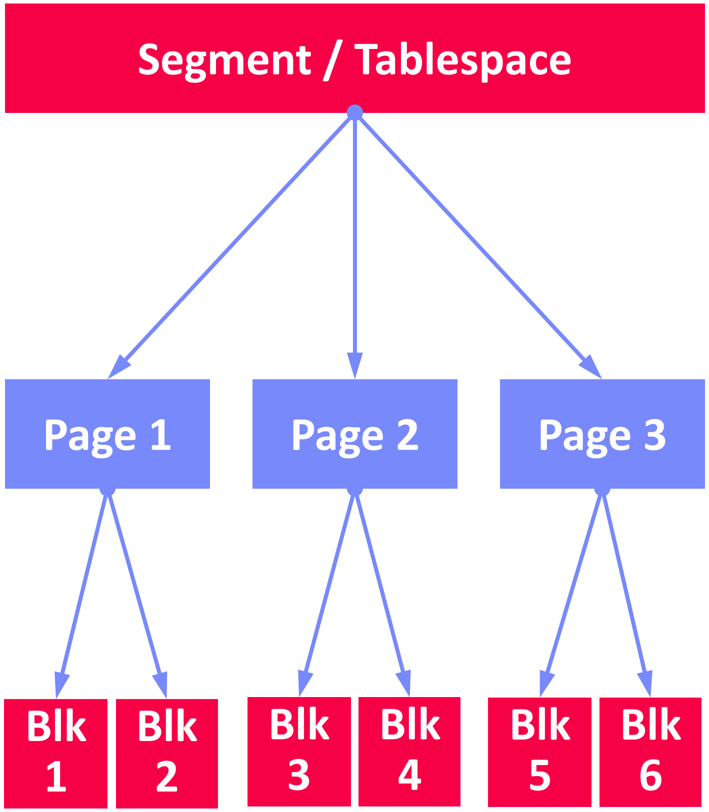
Agenda

- **Page Layouts and Record Management**
- **Buffer Pool Management**
- **Page Replacement Strategies**
- **In-Memory DBMS Eviction**

Page Layout and Record Management

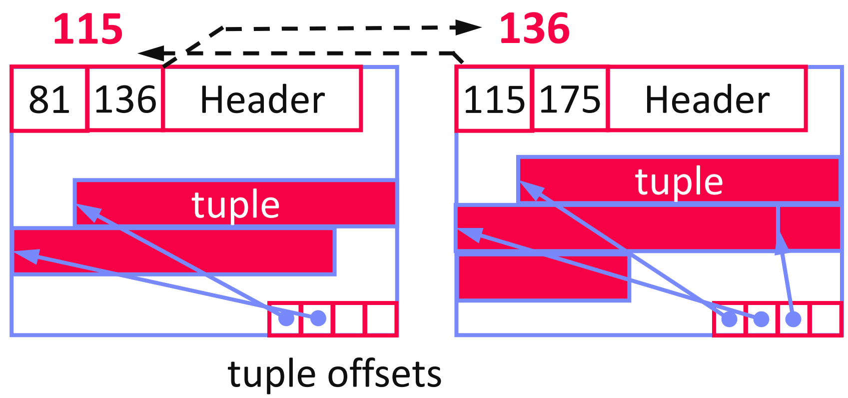
Segments, Pages, and Blocks

- **Segment**
  - Storage unit of DB objects like relations (heap), and indexes
  - Allocate/iterate pages, drop all
  - Often separate file
- **Page**
  - Smallest unit in DB buffer pool
  - Page: fixed-sized memory region
  - Frame: meta data on data page
- **Block** (and/or disk sector)
  - Smallest addressable unit on disk (e.g., POSIX block devices)

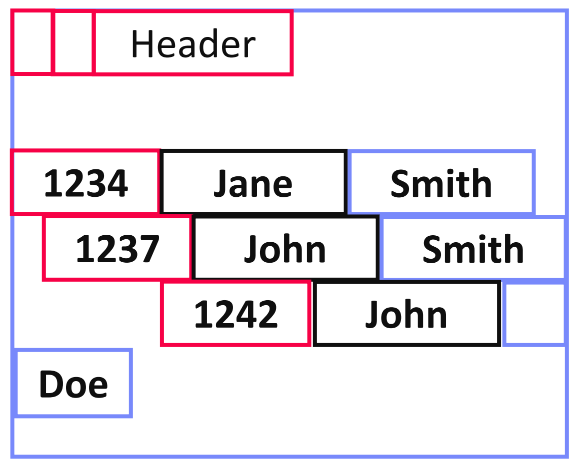


Recap: Page Layout of Row Stores

- **Background: Storage System**
  - Buffer and storage management (incl. I/O) at granularity of **pages**
  - PostgreSQL default: **8KB**
  - Different table/page layouts



- **Row Storage**
  - **NSM** (nary storage model)
  - Store tuple attributes in contiguous form
  - Fast get/insert/delete
  - Slow column aggregates



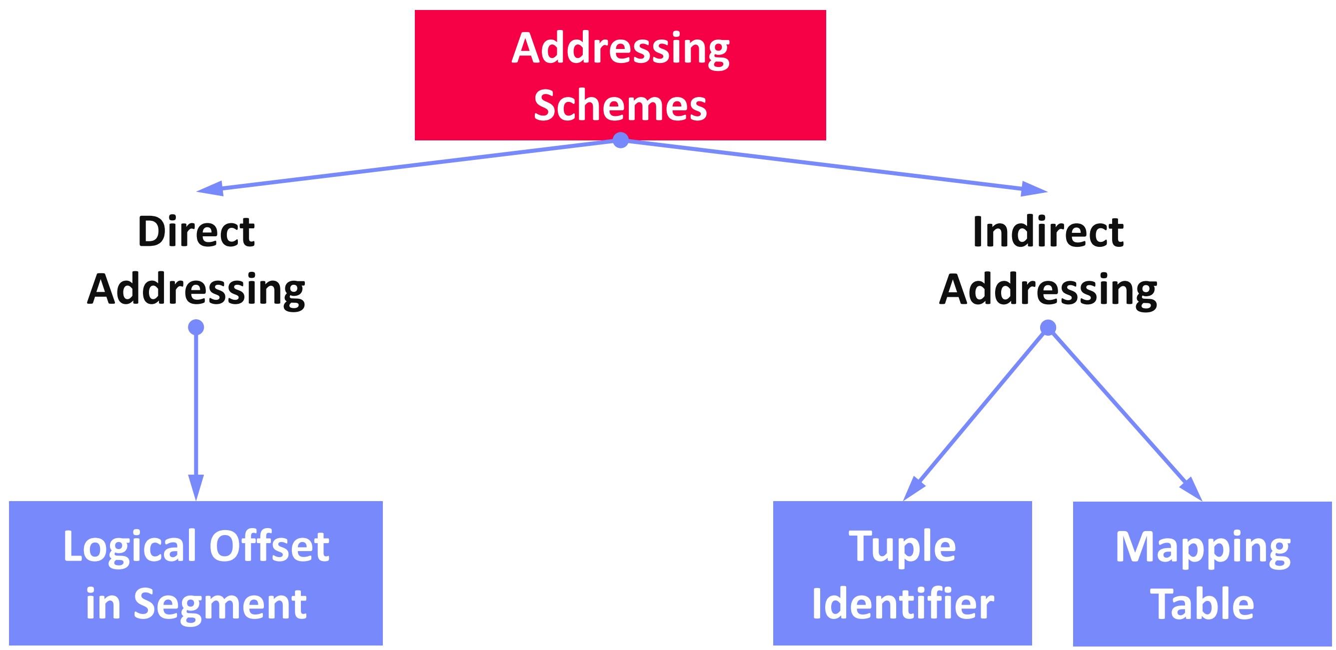
- **Other: DSM, PAX**

Motivation Fixed-size Pages

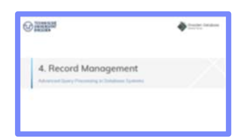
- **#1 Alignment with Disk Blocks**
  - Typically 512B to 4KB (AF) blocks as minimum storage unit
  - A single DB page should map to 1..N physical disk blocks/sectors
- **#2 Sequential Reads/Writes**
  - Recap: HDD seek times vs sequential read/write
  - Similar: SSD sequential read/write w/ higher bandwidth
- **#3 Simplified Buffer Manager**
  - Fixed-size pages removes need for reasoning about sizes for eviction
  - Fixed-size pages avoid main memory fragmentation
- **Recent Perspective: Variable-Size Pages**
  - Large objects (strings, dictionaries) across pages complicates/slows down DBMS components

[Thomas Neumann, Michael J. Freitag: Umbra: A Disk-Based System with In-Memory Performance. **CIDR 2020**]

Classification of Record Addressing Schemes



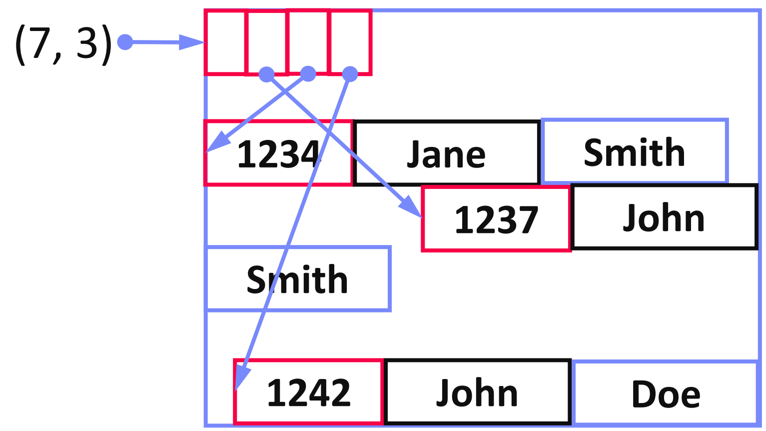
[Dirk Habich: Advanced Query Processing in Database Systems – Record Management, TU Dresden, **WS 2019**]



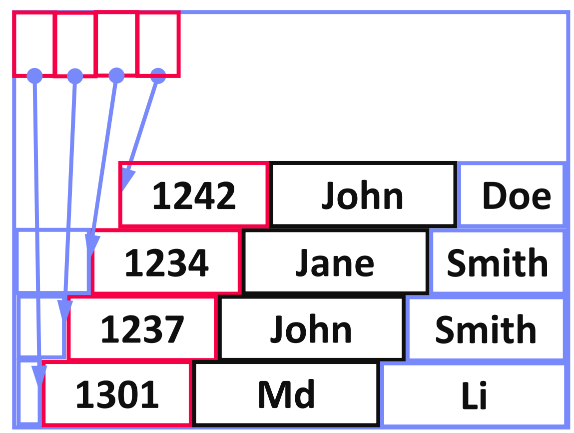
TID (Tuple Identifier) Concept

- **Problem:** Internal TID should be stable, even if records reorganized

- **TID Concept (p, s)**
  - TID := (page number, slot index)
  - Page slot directory holds tuple offsets (byte position) within page
  - Variable number of slots
  - Single page access for internal row



- **Reorganization**
  - Compact free space between records via page-local record movements
    ➔ Updates of page-local directory sufficient
  - Inserts: use free slot or add new slot



TID (Tuple Identifier) Concept, cont.

- **Example PostgreSQL**
  - Recap: Papers(PKey, Title, Pages, CKey, JKey)
  - Hidden CTID system column (not shown on *, but usable)

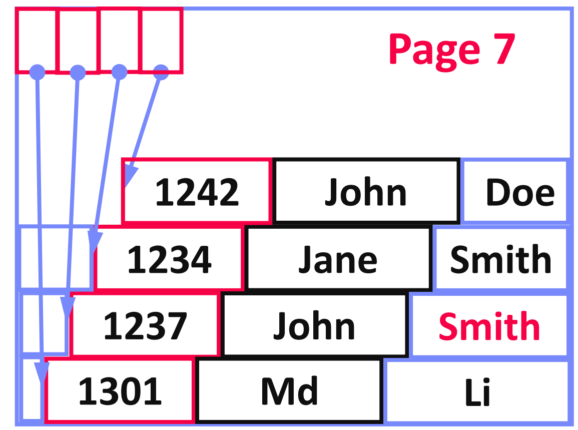
```
SELECT CTID, PKey,
       Title, Pages
  FROM Papers
```

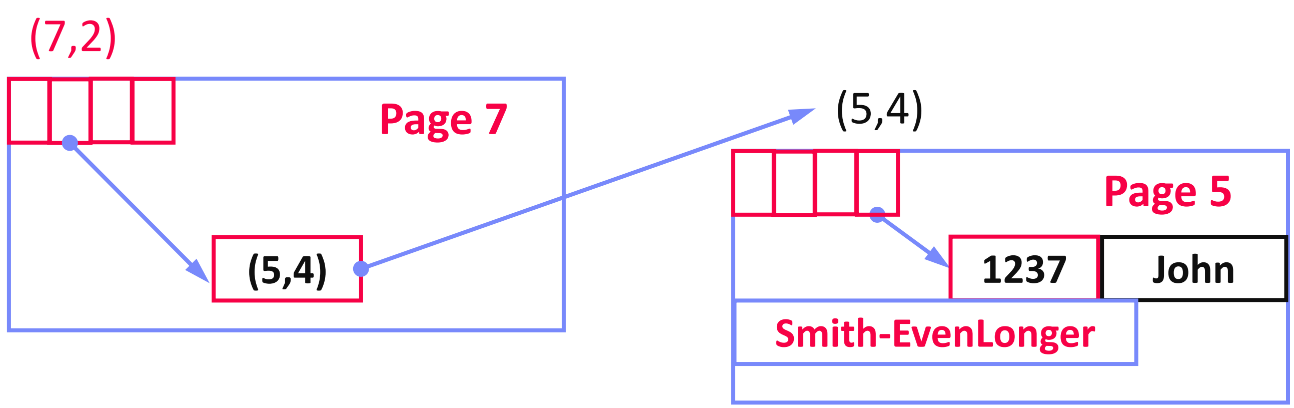
	ctid tid	pkey integer	title character varying (512)	pages character va
5681	(78,21)	731118	MV-IDX: Multi-Version Index in Action	671-674
5682	(78,22)	731121	Hochperformante Analyse von Graph-Dat...	311-330
5683	(78,23)	731122	SPARQling Pig - Processing Linked Data wi...	279-298
5684	(78,24)	731123	RelaX: A Webbased Execution and Learnin...	503-506
5685	(78,25)	731129	Efficient In-Memory Indexing with General...	227-246
5686	(78,26)	731130	Datensicherheit in mandantenfähigen Clo...	477-489
5687	(78,27)	731131	In-Database Machine Learning: Gradient ...	247-266
5688	(78,28)	731133	FlexY: Flexible; datengetriebene Prozessm...	503-506
5689	(78,29)	731134	Extending the MPSM Join	57-71
5690	(78,30)	731137	Orthogonal key-value locking	237-256

- **Other Hidden System Columns**
  - oid, tableoid
  - xmin, cmin (insert), xmax, cmax (delete)

TID (Tuple Identifier) Concept, cont.

- **Overflow Handling**
  - On updates, tuple might need additional space (more than available on page)
  - **Example:** Rename “Smith” to “Smith-EvenLonger”
  - Reference new page, to preserve original TID (chains longer than 1 can be internally avoided)

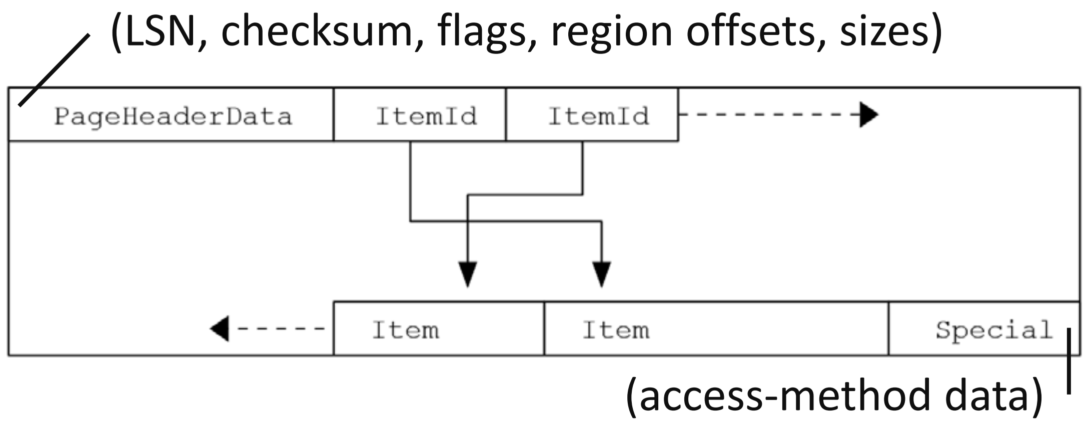




Example Page Layouts

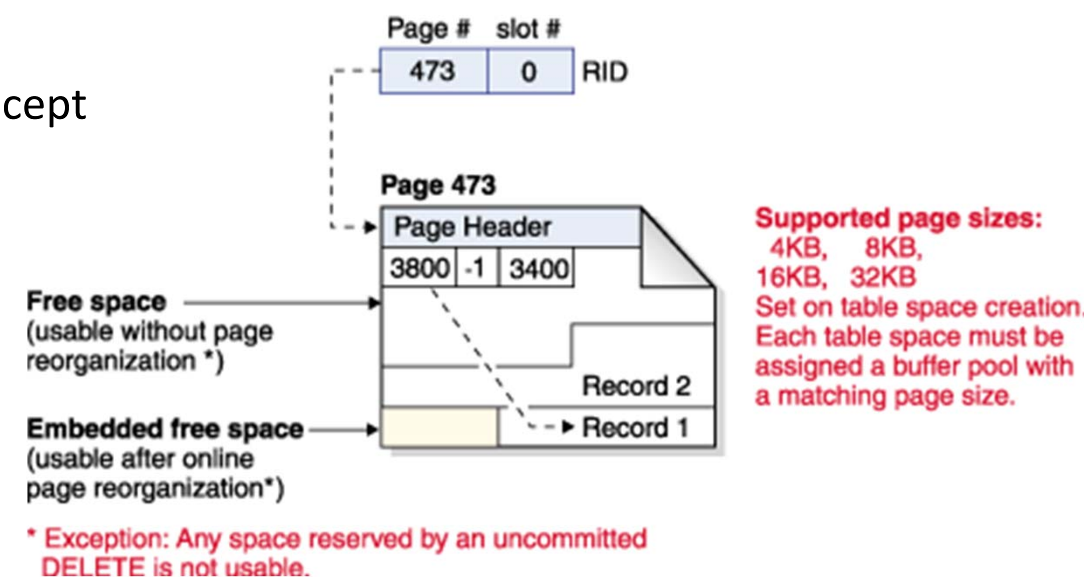
- **PostgreSQL 13.5**
  - Uses TID concept

[https://www.postgresql.org/docs/13/storage-page-layout.html]



- **IBM DB2 11.5**
  - Uses TID (aka RID) concept

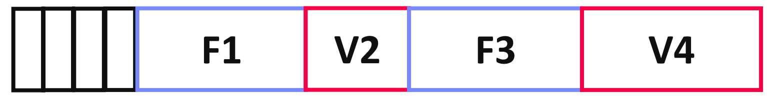
[https://www.ibm.com/support/knowledgecenter/SSEPGG_11.5.0/com.ibm.db2.luw.admin.perf.doc/doc/c0005424.html]



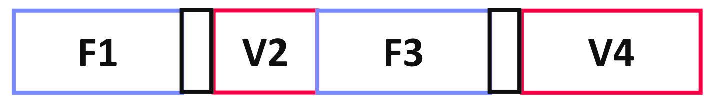
Common Record Layouts

- **#1 Fixed-Size Fields**
  - Concatenated fields, directly accessible

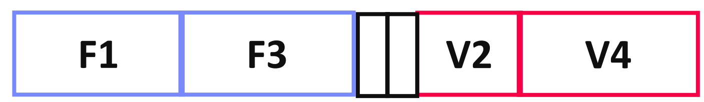
- **#2 Offsets**
  - Prefix with relative offsets of all fields



- **#3 Embedded Length Fields**
  - Length fields only for variable-size fields
  - Cannot access a specific field w/o record scan



- **#4 Partitioned**
  - Partition 1: Fixed-sized fields
  - Partition 2: Offsets and variable-sized fields

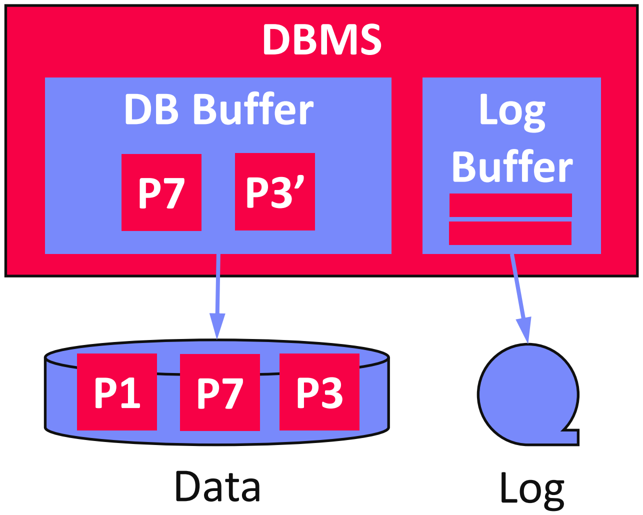


- **Other:** Sometimes bitmap field (#cols/8 bytes) for NULL indicator, etc

Buffer Pool Management

Buffer Pool Overview

- **Buffer Pool**
  - Holds fraction of DB pages in memory
  - Find pages via addressing scheme
  - Allocate memory (local, global)
  - Page replacement (exact, approximate)



- **Example Configuration (PostgreSQL)**
  - `block_size`: size of disk block, i.e., page (default **8KB**)
  - `shared_buffers`: size of cross-session buffer pool (default **128MB**)
    → Recommended tuning: 25% of available memory
  - `temp_buffers`: size of session-local memory for tmp tables (default **8MB**)
  - `work_mem`: size of operation-local memory for sort/hash tables (default **4MB**)

[https://www.postgresql.org/docs/13/runtime-config.html]

DB Buffer Pool vs Operating System

- **#1 Why not Memory-Mapped Files (mmap)**
  - ACID Atomicity and Durability (flush TX log before dirty pages)
  - ACID Isolation (locking of pages)
  - Context knowledge of query processing / access paths; portability
- **#2 Why no Swapping**
  - No durability of changes after restart
  - With DB buffer pool danger of **double page faults**
    (requested page not in DB buffer - load, victim page swapped – load, replace)
- **#3 Why no OS File Cache**
  - **#1** Bypass via direct I/O (O_DIRECT) to avoid redundant caching
  - **#2** Leverage via small buffer pool and otherwise OS file cache (see Postgres)

Buffer Pool Interface

[Thomas Neumann: Datenbanksysteme und moderne CPU-Architekturen - Storage, TU Munich, **2019**]

- **Pin/Fix**
  - `fix(pageID, exclusive)`
  - Pins page for read/write access, guards against replacement
  - If page not in buffer, read and replace victim page in buffer pool
- **Unpin/Unfix**
  - `unfix(pageID, dirty)`
  - Unpins page to release guard against replacement
  - Dirty flag indicates if page has been modified → async write to disk
- **Others Aspects**
  - Additional operations: Get via `fix(pageNo,false)`, Mark dirty, Flush
  - Lookup via hash map (pageID, buffer frame), load/replace via put/remove

Buffer Frame Allocation

- **Global and Local Memory Allocation**
  - Global: shared buffer pool used by all transactions, sessions, and users
  - Local: transaction/session-local buffers for temporary tables and operations

- **PostgreSQL Buffer Frame** (Buffer Descriptor)
  - Access to data page via buf_id (hash table lookup)

[https://github.com/postgres/postgres/blob/master/src/include/storage/buf_internals.h]

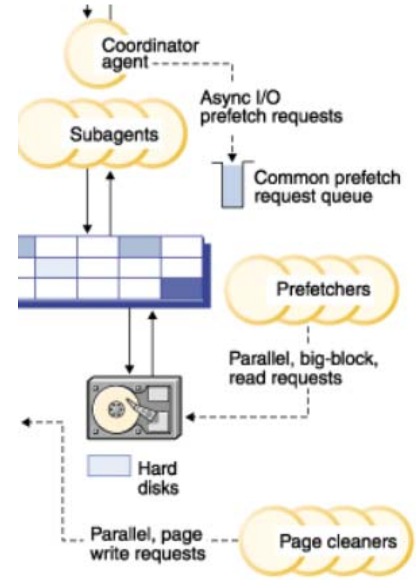
```
// Extracted as of Oct 18, 2020
typedef struct BufferDesc {
    BufferTag        tag;           /* ID of page contained in buffer */
    int              buf_id;        /* buffer's index number (from 0) */
    pg_atomic_uint32 state;         /* tag state, flags, ref/usage counts */

    int              wait_backend_pid;  /* backend PID of pin-count waiter */
    int              freeNext;      /* link in freelist chain */

    LWLock           content_lock;  /* to lock access to buffer contents */
} BufferDesc;
```

Pre-Fetching, Cleaning, and Scan Sharing

- **Pre-Fetching** (Async)
  - Overlay computation w/ speculative sequential read of multiple pages
  - Based on physical data structures, and query plan



- **Cleaning** (Async)
  - Asynchronous sequential write of changed (dirty) pages → moved out of critical path of TX processing

- **Scan Sharing**
  - Multiple queries can piggyback on existing table scan, w/ compensations
  - **Red Brick:** coordinated table scan
  - **Crescando:** continuous scan

[Phillip M. Fernandez: Red Brick Warehouse: A Read-Mostly RDBMS for Open SMP Platforms. **SIGMOD 1994**]

[Philipp Unterbrunner, Georgios Giannikis, Gustavo Alonso, Dietmar Fauser, Donald Kossmann: Predictable Performance for Unpredictable Workloads. **PVLDB 2(1) 2009**]

Excursus: Automatic Buffer Pool Tuning

- **IBM DB**
  - Self-tuning memory manager
  - Caches, ops, buffer pool

[Adam J. Storm, Christian Garcia-Arellano, Sam Lightstone, Yixin Diao, Maheswaran Surendra: Adaptive Self-tuning Memory in DB2. **VLDB 2006**]

- **Oracle**
  - Automatic tuning of SGA/PGA (System/Process Global Memory)

[Benoît Dageville, Mohamed Zaït: SQL Memory Management in Oracle9i. **VLDB 2002**]

- **Microsoft**
  - Multi-tenant page replacement (MR-LRU)

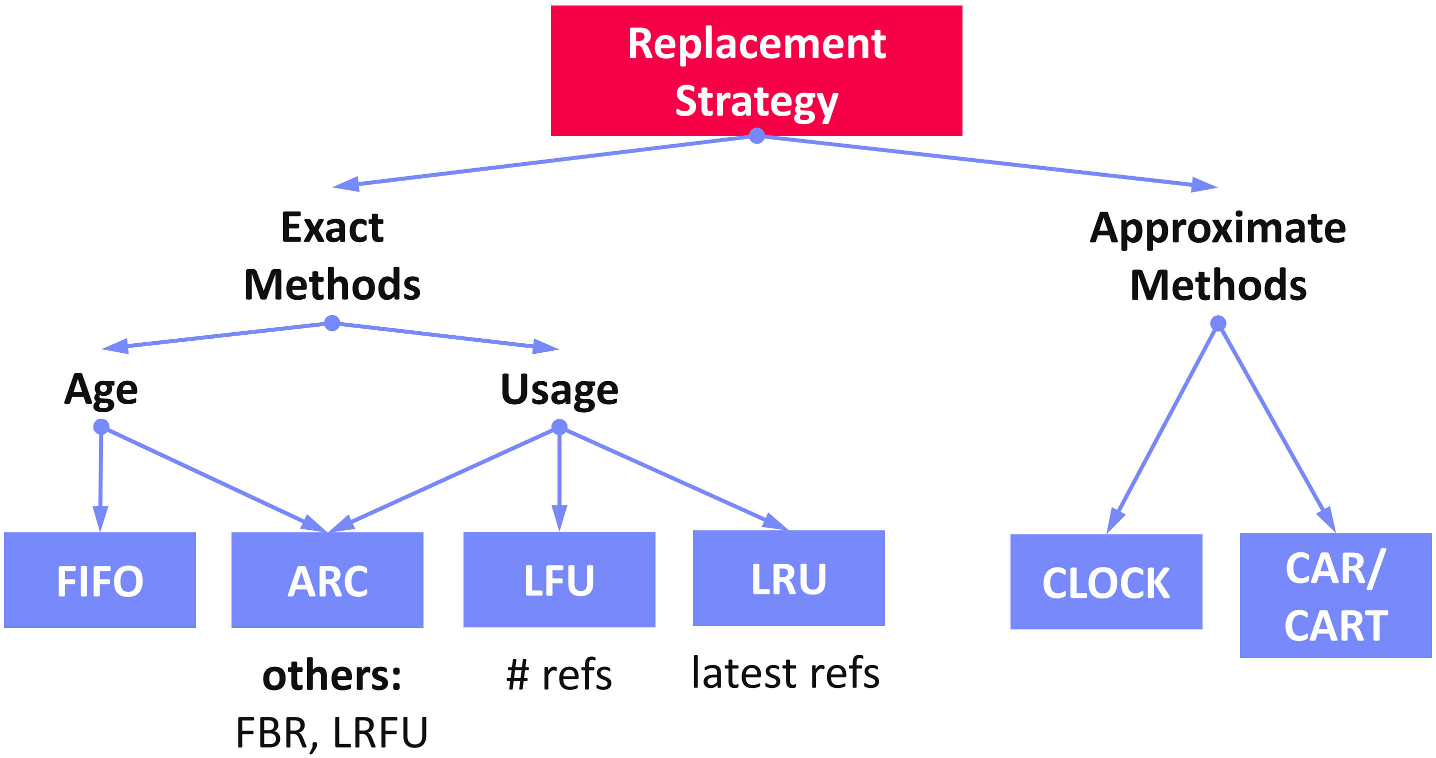
[Vivek R. Narasayya, Ishai Menache, Mohit Singh, Feng Li, Manoj Syamala, Surajit Chaudhuri: Sharing Buffer Pool Memory in Multi-Tenant Relational Database-as-a-Service. **PVLDB 8(7), 2015**]

- **OtterTune**
  - ML-based tuning of DB configurations

[Dana Van Aken, Andrew Pavlo, Geoffrey J. Gordon, Bohan Zhang: Automatic Database Management System Tuning Through Large-scale Machine Learning. **SIGMOD 2017**]

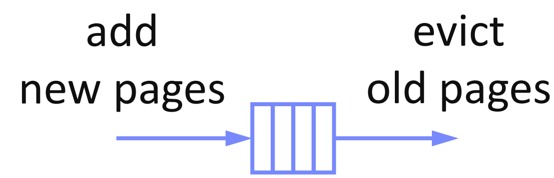
Page Replacement Strategies

Classification of Replacement Strategies

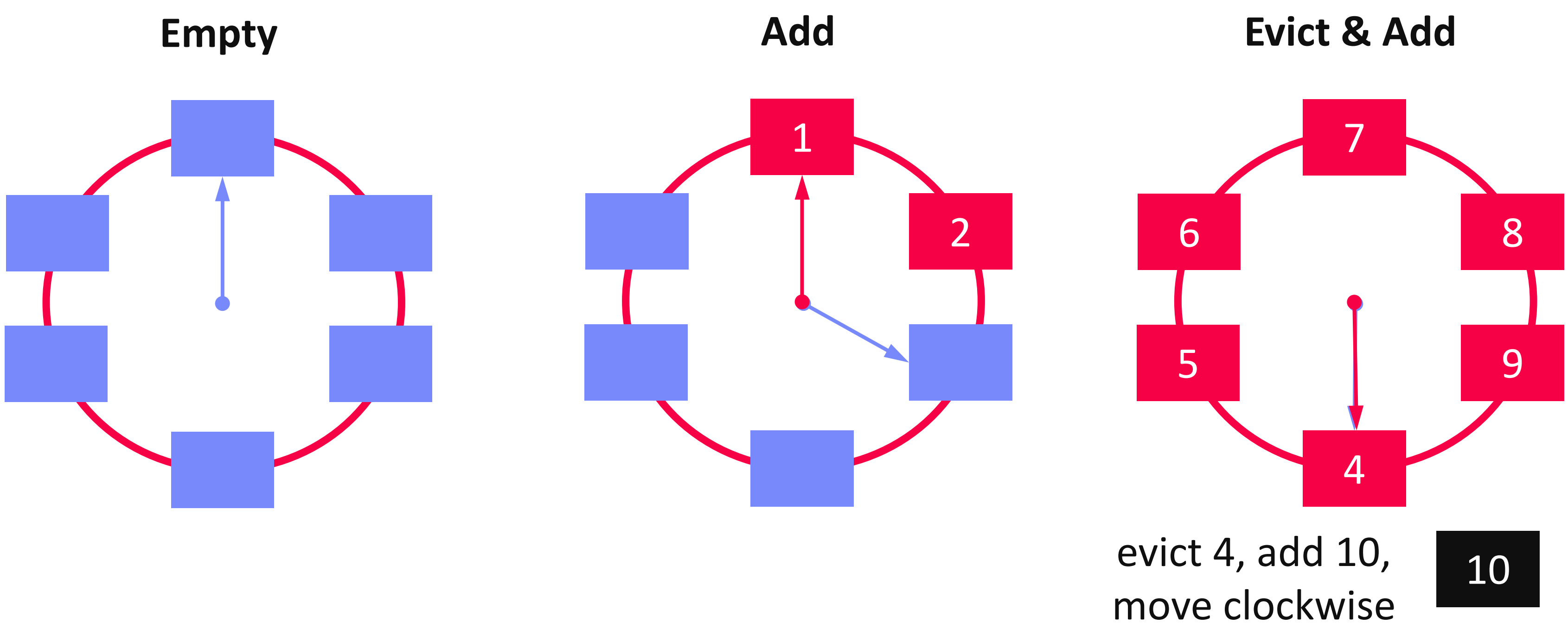


[Dirk Habich: Advanced Query Processing in Database Systems – Storage Management and System Buffer, TU Dresden, **WS 2019**]

FIFO (First-in, first-out)

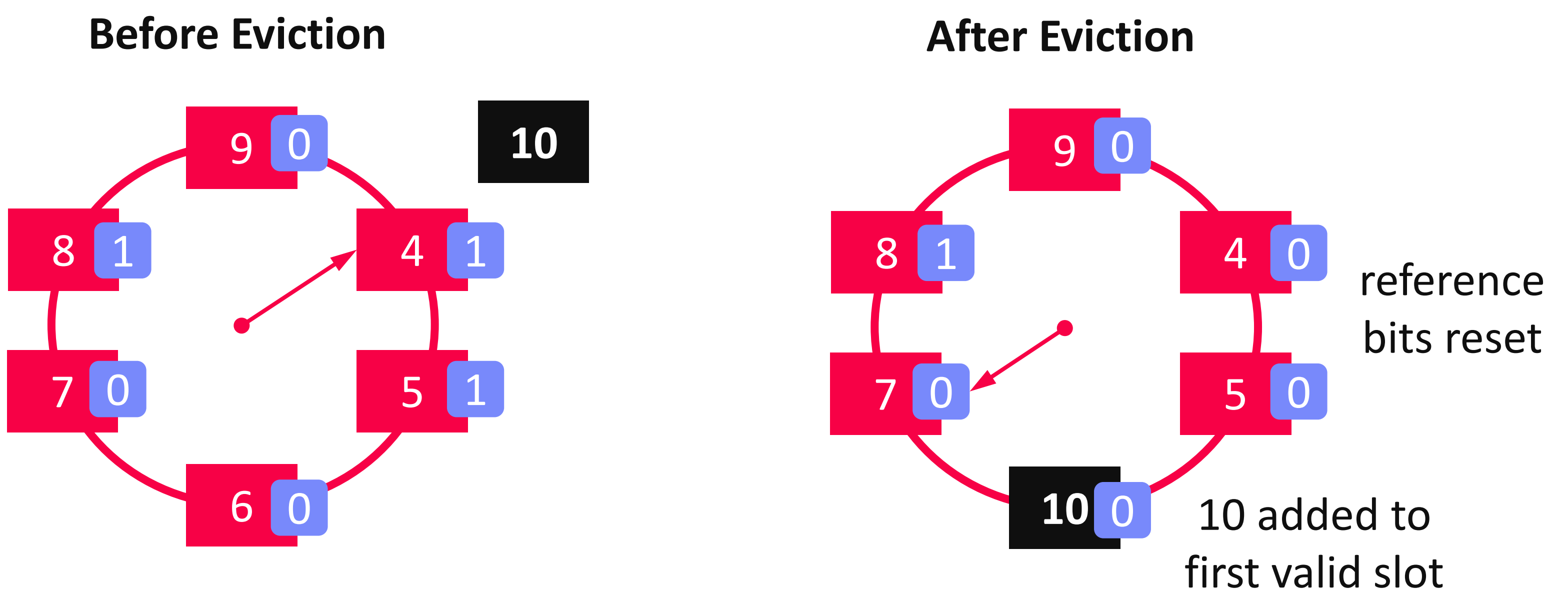


- **Strategy**
  - Evict oldest page (time in buffer) from pool
  - Implementation as basic ring buffer of size c (capacity)
  - Ignores frequent and recent page references



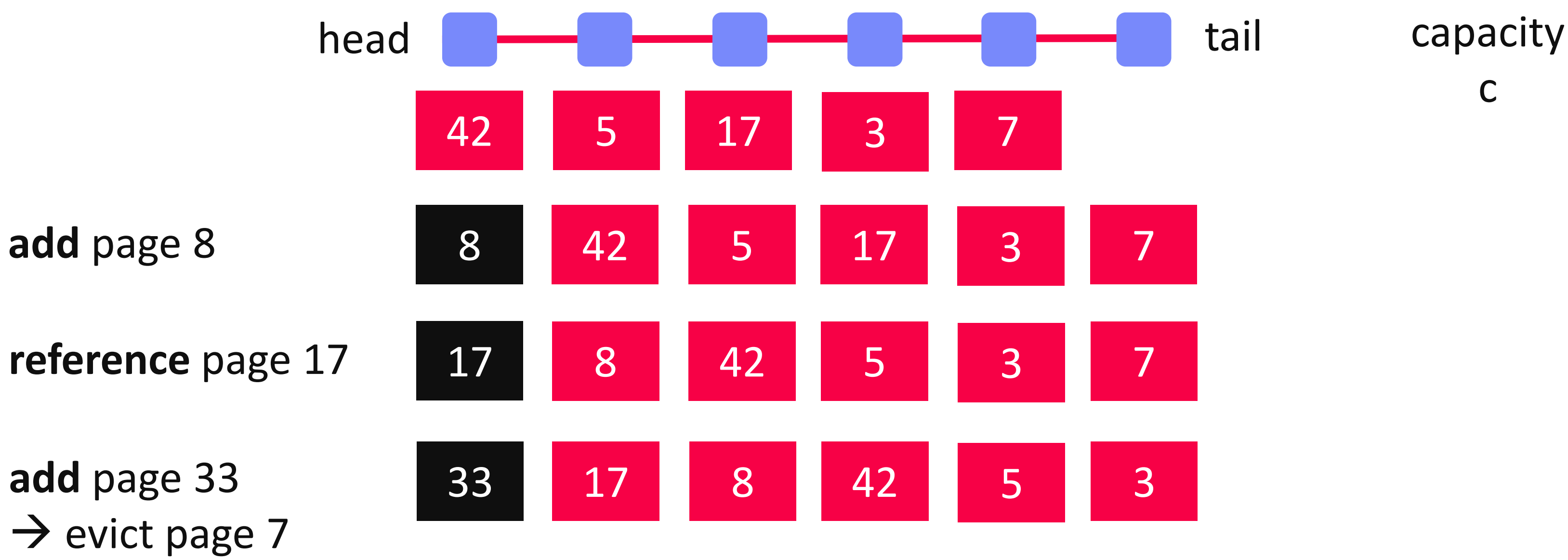
CLOCK (Second Chance)

- **Strategy**
  - Each page has a **reference bit R**, indicating if it was referenced in the last cycle
  - Evict oldest page (time in buffer) **with R=0** from pool
  - FIFO extension with coarse-grained accounting of page references
  - **Variant:** GCLOCK (Generalized CLOCK) w/ ref counter (PostgreSQL **clock sweep**)



LRU (Least Recently Used)

- **Strategy**
  - Evict least recently used page (last page reference)
  - Implementation as **basic list/queue** (head: new pages, tail: LRU page)
  - Equivalent to FIFO for sequential scans (might evict hot data pages)



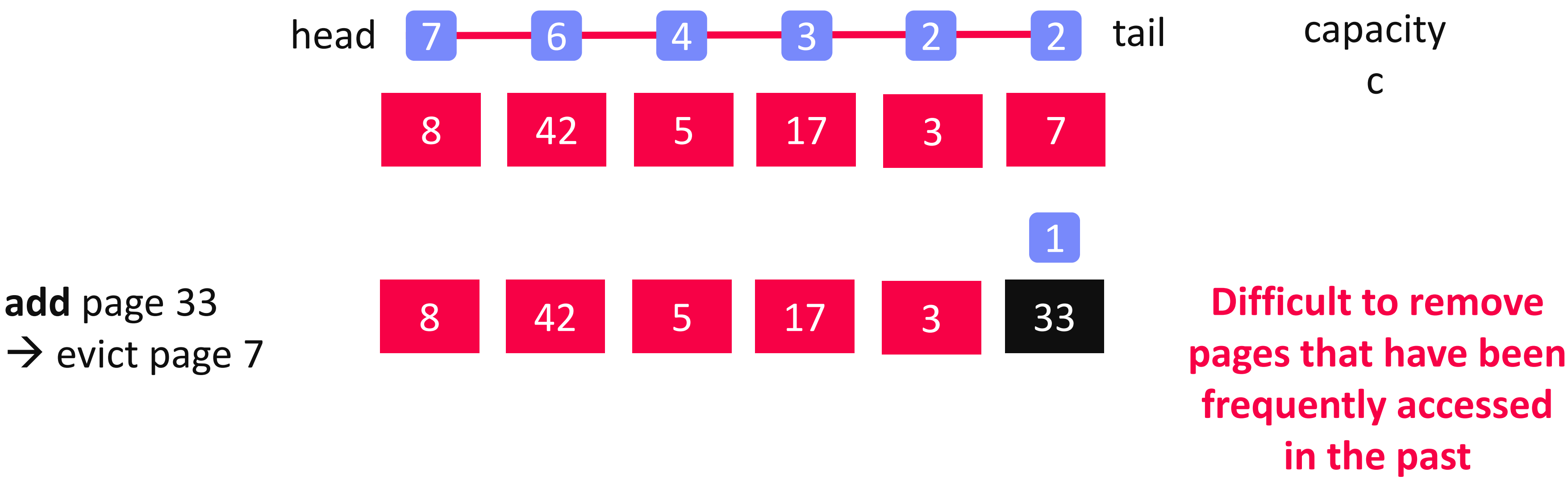
LRU-K (Least Recently Used K)

- **Strategy**
  - Evict page with max backward K-distance (k^{th}-last reference, ∞ if <k refs)
  - LRU-1 equivalent to LRU, in practice: often LRU-2
  - **Variants:** timestamp as of page reference, or of page UNFIX operation

head → tail (capacity c)	8	42	5	17	3	7
K last references	(23, 17)	(24, 15)	(14, 12)	(15, 9)	(10, 7)	(5)
K=2 Distance at T=25	**8**	**10**	**13**	**16**	**18**	**∞**

LFU (Least Frequently Used)

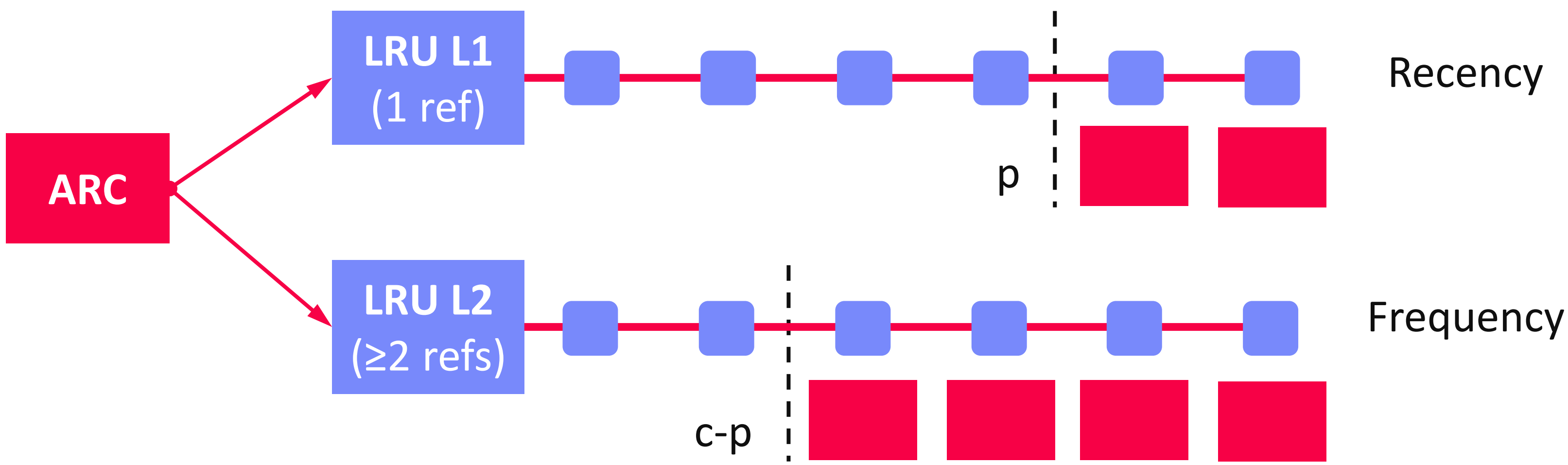
- **Strategy**
  - Evict page with min **reference count** since brought in buffer pool
  - Draws resolved with secondary strategy (e.g., FIFO)
  - Implement as list with swaps of neighbors on access



ARC (Adaptive Replacement Cache)

[Nimrod Megiddo, Dharmendra S. Modha: ARC: A Self-Tuning, Low Overhead Replacement Cache. **FAST 2003**]

- **Strategy**
  - Maintain two LRU lists of pages: L1 and L2
  - Keep cache directory of length c (cache size) for both lists
  - Keep c pages in cache, p in L1 and (c-p) L2
  - Replacement: evict LRU L1 if |L1|>p, evict LRU L2 if |L1|<p
  - Adaptively tune p based on hits and size of L1/L2 lists w/o pages



- **Note:** Linux page cache w/ 'active' and 'inactive' LRU page lists + migration

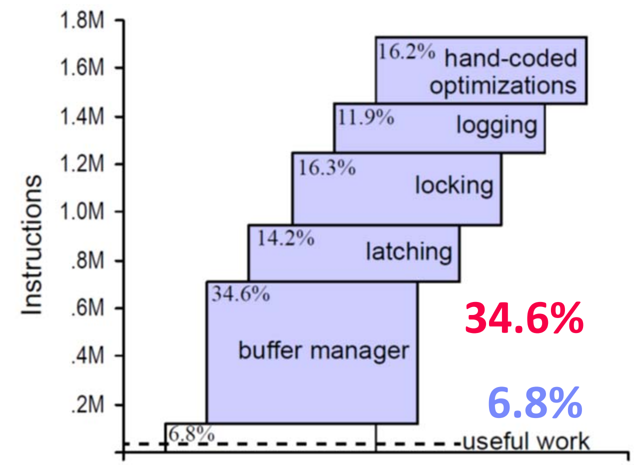
In-Memory DBMS Eviction

Motivation In-Memory DBMS

- **Common Misconception: So an in-memory database system is just a regular database system with unlimited buffer pool capacity?**

- **Disk-based DBMS Overhead**
  - OLTP workloads bottlenecked on buffer pool, latching, locking, logging
  - Evaluated on Shore-MT research prototype

[Stavros Harizopoulos, Daniel J. Abadi, Samuel Madden, Michael Stonebraker: OLTP through the looking glass, and what we found there. **SIGMOD 2008**]



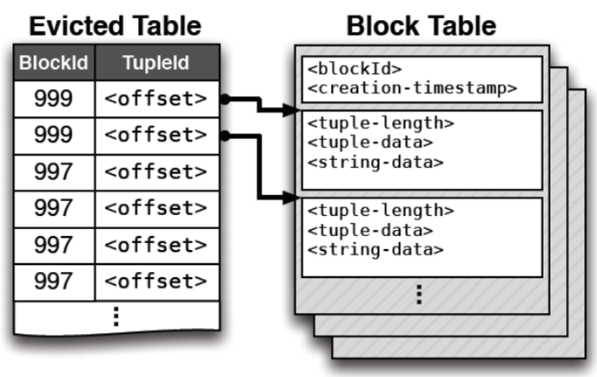
- **In-Memory DBMS**
  - Eliminates one of the main bottlenecks (disk I/O, and buffer pool)
  - Requires improvements for modern hardware, locking/latching, etc
  - However, storage cost-perf trade-off (DRAM vs SSD/HDD)
    ➔ **How to enable graceful evictions, without reintroducing overhead**?

Anti Caching (Andy Pavlo et al.)

- **Fine-grained Eviction**
  - Online identification of cold tuples
  - Threshold of ~80% triggers anti-caching
  - Abort TX on “page fault”, retrieve, and restart TX (no blocking of other TXs)
  - Pre-pass to identify all page faults of TX

[Justin DeBrabant, Andrew Pavlo, Stephen Tu, Michael Stonebraker, Stanley B. Zdonik: Anti-Caching: A New Approach to Database Management System Architecture. **PVLDB 6(14) 2013**]

- **Anti-Cache**
  - Construct fixed-size blocks via LRU chain
  - Evicted Table: in-mem map of evicted tuples (granularity of individual data accesses)
  - Block Table: on-disk map of evicted blocks

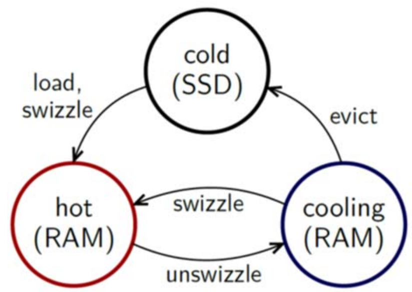


- **Excursus: SystemDS Buffer Pool**
  - Similarly, eviction of live variables under memory pressure
  - DIA projects: **#44** Lineage-Exploitation in Buffer Pool

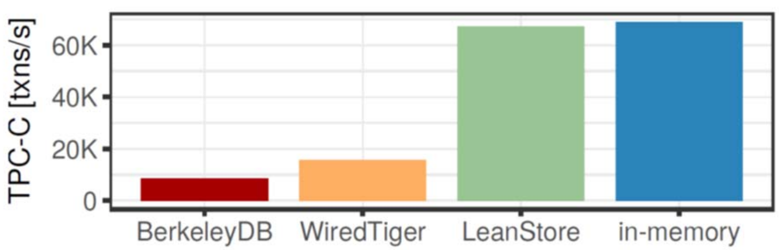
LeanStore (Viktor Leis et al.)

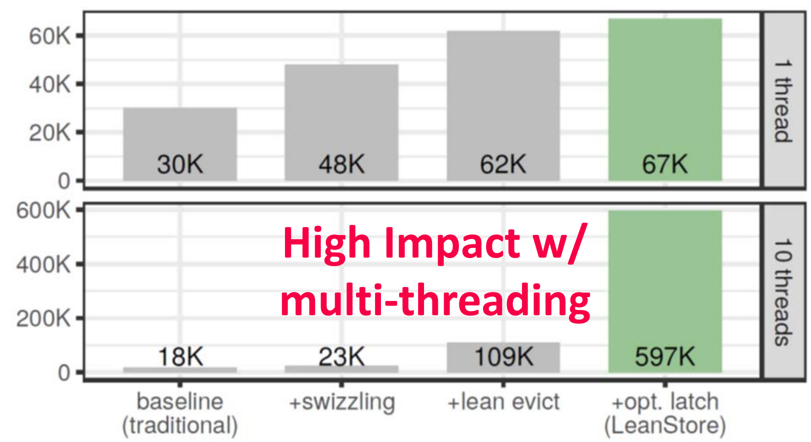
- **Coarse-Grained Eviction**
  - Motivation: avoid buffer pool overhead
  - Pointer swizzling (direct page references)
  - Avoid LRU overhead per page access by tracking infrequently access pages
  - Speculative unswizzling w/o eviction
  - CLOCK eviction unswizzled pages

[Viktor Leis, Michael Haubenschild, Alfons Kemper, Thomas Neumann: LeanStore: In-Memory Data Management beyond Main Memory. **ICDE 2018**]



- **Experimental Results**
  - TPC-C 10 WH (initially 10GB)





Summary and **Q&A**

- **Page Layouts and Record Management**
- **Buffer Pool Management**
- **Page Replacement Strategies**
- **In-Memory DBMS Eviction**

- **Programming Projects**
  - Initial test suite, benchmark, make file, and reference implementation
  - Try compiling it, and start **your own implementation** in next weeks

- **Next Lectures** (Part A)
  - 04 **Index Structures and Partitioning** [Oct 28]
  - 05 **Compression Techniques** [Nov 04]