Data Storage Structures 2

Instructor: Beom Heyn Kim

beomheynkim@hanyang.ac.kr

Department of Computer Science

Overview

- Organization of Records in Files
- Database Buffer
- Assignments

Organization of Records in Files

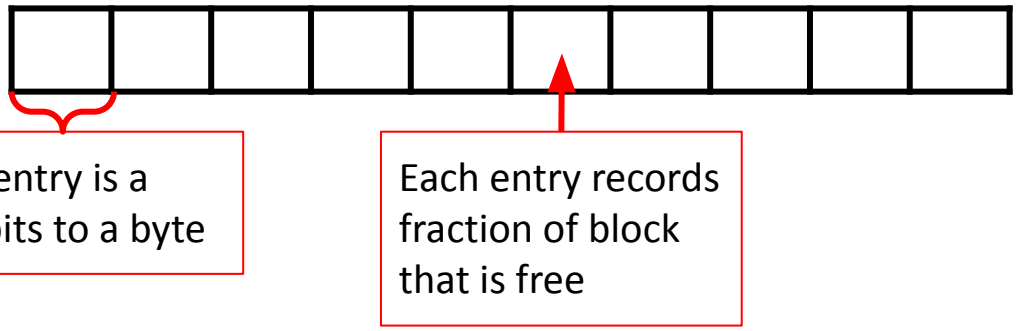
- How can we organize records in files? There are several ways:
  - **Heap**
    - Store records anywhere in the file where there is space
  - **Sequential**
    - Store records in sequential order based on the value of the search key of each record
  - **Multitable clustering file organization**
    - Store records of several different relations in the same file
    - To store related records on the same block to minimize I/O
  - **B+tree file organization** (Ch14)
    - Ordered storage even with inserts/deletes
  - **Hashing** (Ch14)
    - A hash function computed on search key
    - The result specifies in which block of the file the record should be stored

Heap File Organization

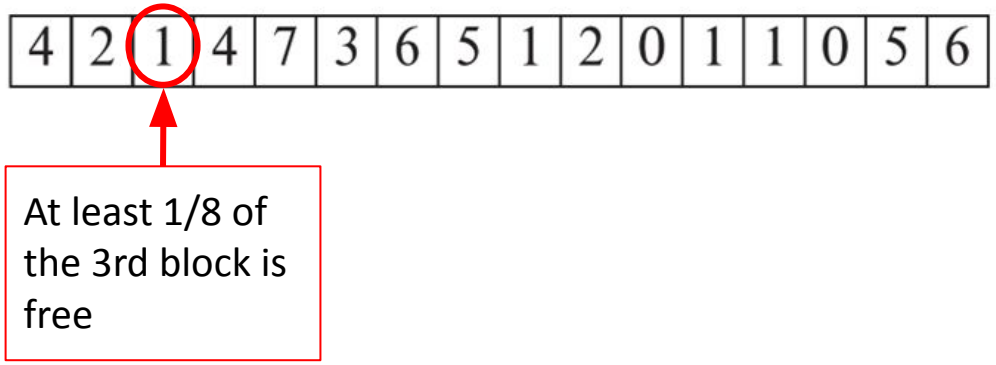
- In a heap file organization,
  - records can be stored anywhere in the file where there is free space
  - records usually do not move once allocated
- Problem is to efficiently find free spaces
  - For insertion, we can store a new record at the end of file
  - What if some record in the middle of the file gets deleted?
    - How can we efficiently find these free spaces in the file to store records without sequential search?
      - Most database systems uses a space-efficient data structure called **free-space map**

Heap File Organization

- **Free-space map**
  - Array with 1 entry per block

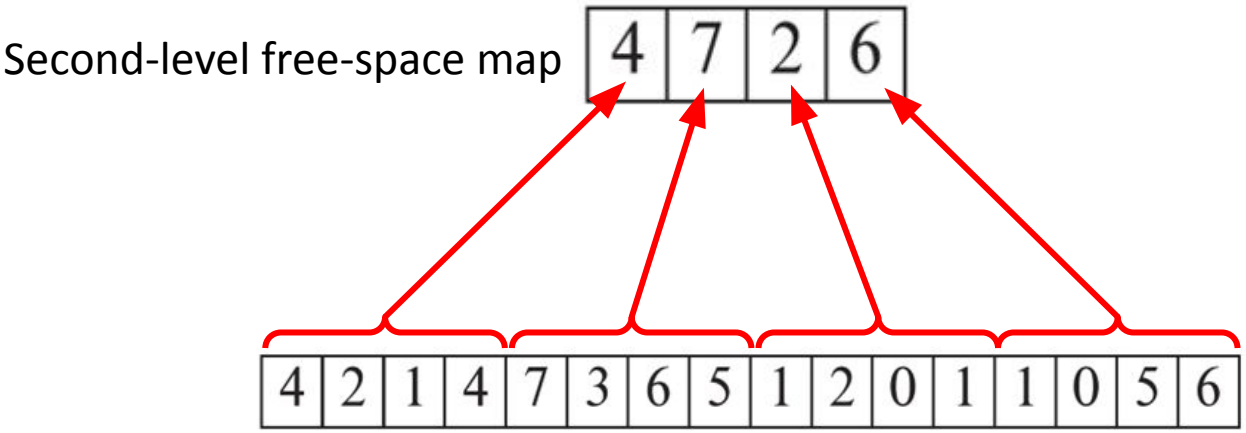


  - Example:
    - 3 bits per block
    - Value divided by 8 indicates the fraction of block that is free



Heap File Organization

- Can have second-level free-space map
  - Example
    - Each entry stores maximum from 4 entries of first-level free-space map

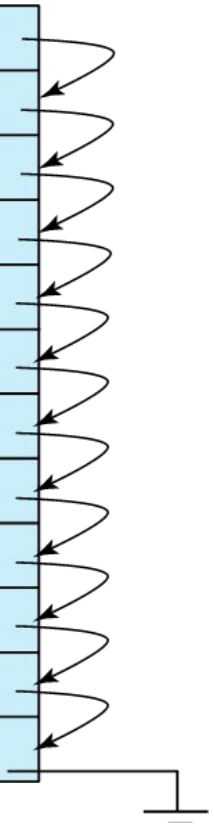


- Free-space map written to disk periodically, OK to have wrong (old) values for some entries (will be detected and fixed)

Sequential File Organization

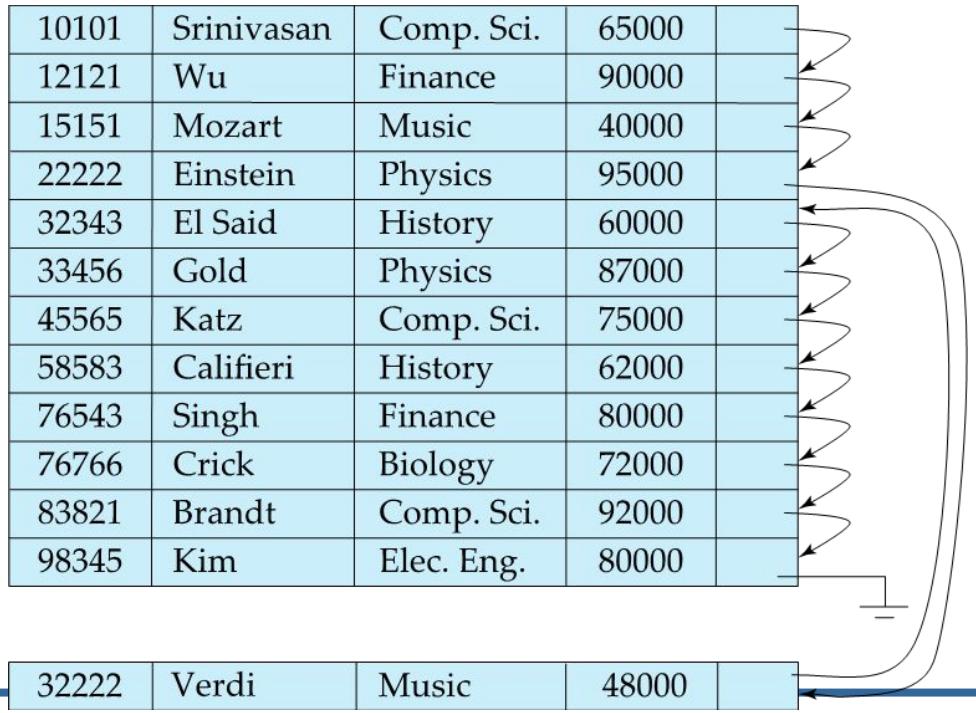
- Suitable for applications that require sequential processing of the entire file
- The records in the file are ordered by a search-key

10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	



Sequential File Organization (Cont.)

- Deletion – use pointer chains
- Insertion –locate the position where the record is to be inserted
  - if there is free space insert there
  - if no free space, insert the record in an overflow block
  - In either case, pointer chain must be updated
- Need to reorganize the file from time to time to restore sequential order



10101	Srinivasan	Comp. Sci.	65000	
12121	Wu	Finance	90000	
15151	Mozart	Music	40000	
22222	Einstein	Physics	95000	
32343	El Said	History	60000	
33456	Gold	Physics	87000	
45565	Katz	Comp. Sci.	75000	
58583	Califieri	History	62000	
76543	Singh	Finance	80000	
76766	Crick	Biology	72000	
83821	Brandt	Comp. Sci.	92000	
98345	Kim	Elec. Eng.	80000	

32222	Verdi	Music	48000	

Multitable Clustering File Organization

- Store several relations in one file using a **multitable clustering** file organization

department

dept_name	*building*	*budget*
Comp. Sci.	Taylor	100000
Physics	Watson	70000

instructor

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

multitable clustering of *department* and *instructor*

Comp. Sci.	Taylor	100000	
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000
Physics	Watson	70000	
33456	Gold	Physics	87000

Multitable Clustering File Organization (Cont.)

- Good for queries involving *department* $\bowtie$ *instructor*, and for queries involving one single department and its instructors
- Bad for queries involving only *department*
- Results in variable size records
- Can add pointer chains to link records of a particular relation

Partitioning

- **Table partitioning**: Records in a relation can be partitioned into smaller relations that are stored separately
- E.g., *transaction* relation may be partitioned into
  transaction_2018, transaction_2019, etc.
- Queries written on *transaction* must access records in all partitions
  - Unless query has a selection such as *year*=2019, in which case only one partition in needed
- Partitioning
  - Reduces costs of some operations such as free space management
  - Allows different partitions to be stored on different storage devices
    - E.g., *transaction* partition for current year on SSD, for older years on magnetic disk

Data Dictionary Storage

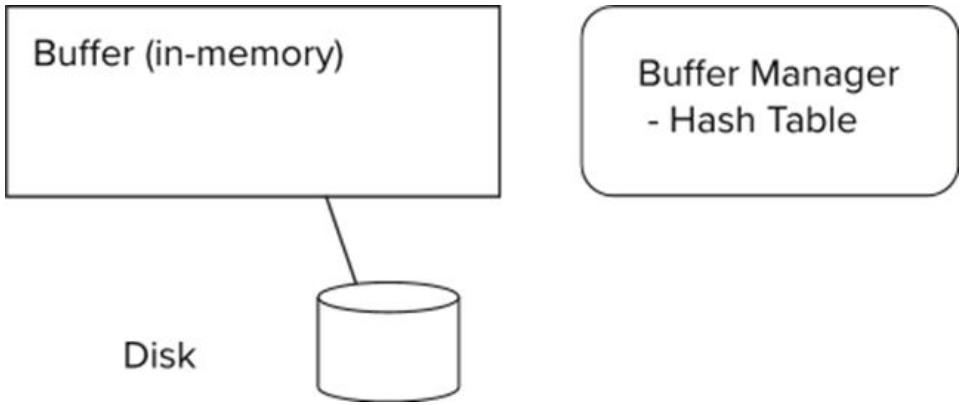
- The **Data dictionary** (also called **system catalog**) stores **metadata**; that is, data about data, such as
  - Information about relations
    - names of relations
    - names, types and lengths of attributes of each relation
    - names and definitions of views
    - integrity constraints
  - User and accounting information, including passwords
  - Statistical and descriptive data
    - number of tuples in each relation
  - Physical file organization information
    - How relation is stored (sequential/hash/…)
    - Physical location of relation
  - Information about indices (Chapter 14)

Overview

- Organization of Records in Files
- Database Buffer
- Assignments

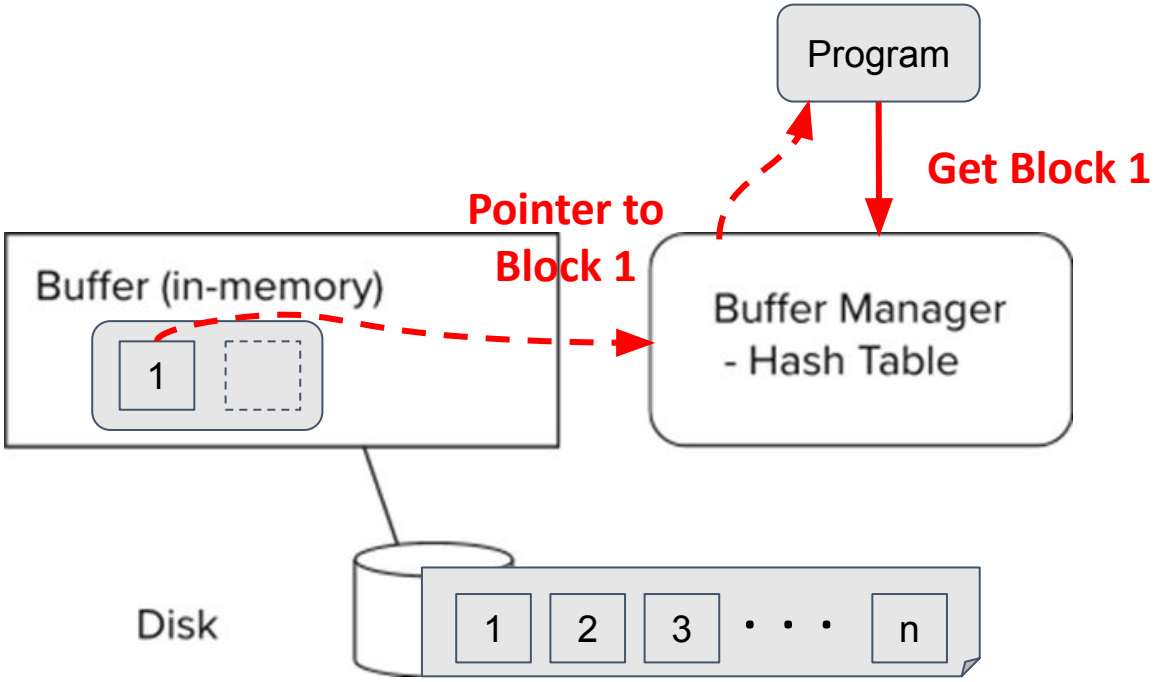
Buffer Manager

- Blocks are units of both storage allocation and data transfer
- Database system seeks to minimize the number of block transfers between the disk and memory.
- We can reduce the number of disk accesses by keeping as many blocks as possible in main memory.
- **Buffer** – portion of main memory available to store copies of disk blocks.
- **Buffer manager** – subsystem responsible for allocating buffer space in main memory.



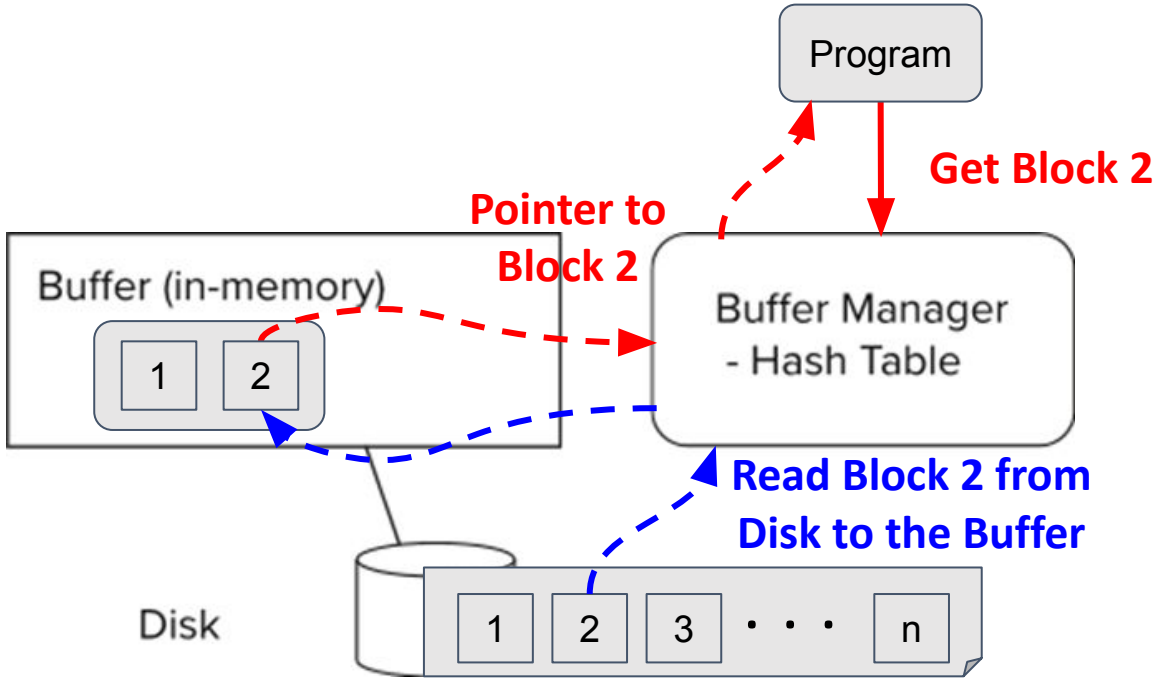
Buffer Manager

- Programs call on the buffer manager when they need a block from disk
  - If the block is already in the buffer, buffer manager returns the address of the block in main memory



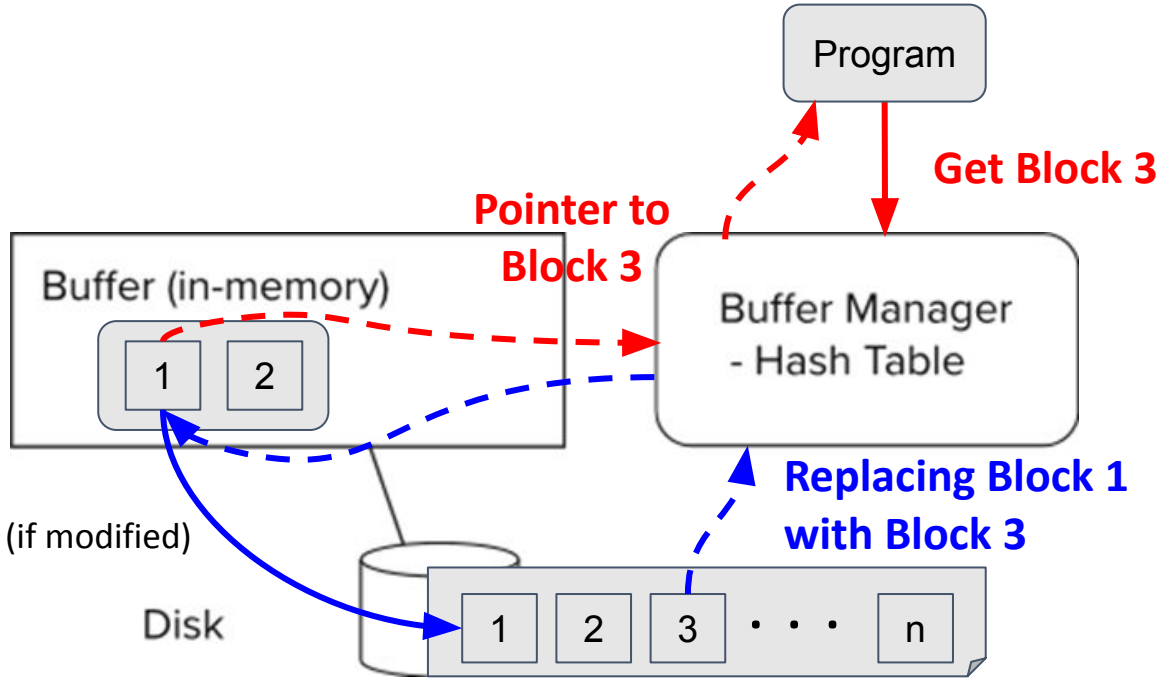
Buffer Manager

- Programs call on the buffer manager when they need a block from disk
  - If the block is not in the buffer, the buffer manager reads the block from the disk to the buffer and returns the address of the block in main memory to requester



Buffer Manager

- Buffer Manager may need to replace some other block to make space for the new block
  - Replaced block written back to disk only if it was modified since the most recent time that it was written to/fetched from the disk



Buffer Replacement Strategy

- A block to be replaced with a new one is **evicted** from the buffer before the new one can be read in
  - Problem: a process may see incorrect data or corrupt the block if eviction and replacement is done during reading/writing.
  - To solve this issue:
    - **Pin** the block before reading/writing data from a block
    - **Unpin** done when read/write is complete
- **Pinned block:** memory block that is not allowed to be written back to disk
  - Multiple concurrent processes may pin/unpin operations
    - Keep a pin count
      - buffer block can be evicted only if pin count = 0

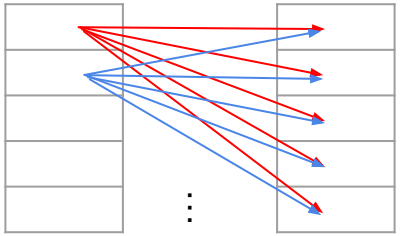
Buffer Replacement Strategy

- Shared and exclusive locks on buffer
  - Needed to prevent concurrent operations from reading page contents as they are moved/reorganized, and to ensure only one move/reorganize at a time
  - Readers get shared lock, updates to a block require exclusive lock
  - **Locking rules:**
    - Only one process can get exclusive lock at a time
    - Shared lock cannot be granted concurrently with exclusive lock
    - Multiple processes may be given shared lock concurrently

Buffer Replacement Policies

- Most operating systems replace the block **least recently used** (LRU strategy)
  - Idea behind LRU – use past pattern of block references as a predictor of future references
  - LRU can be bad for some queries
- Queries have well-defined access patterns (such as sequential scans), and a database system can use the information in a user's query to predict future references
- Mixed strategy with hints on replacement strategy provided by the query optimizer is preferable
- Example of bad access pattern for LRU: when computing the join of 2 relations r and s by a nested loops

```
for each tuple tr of r do
        for each tuple ts of s do
                if the tuples tr and ts match …
```

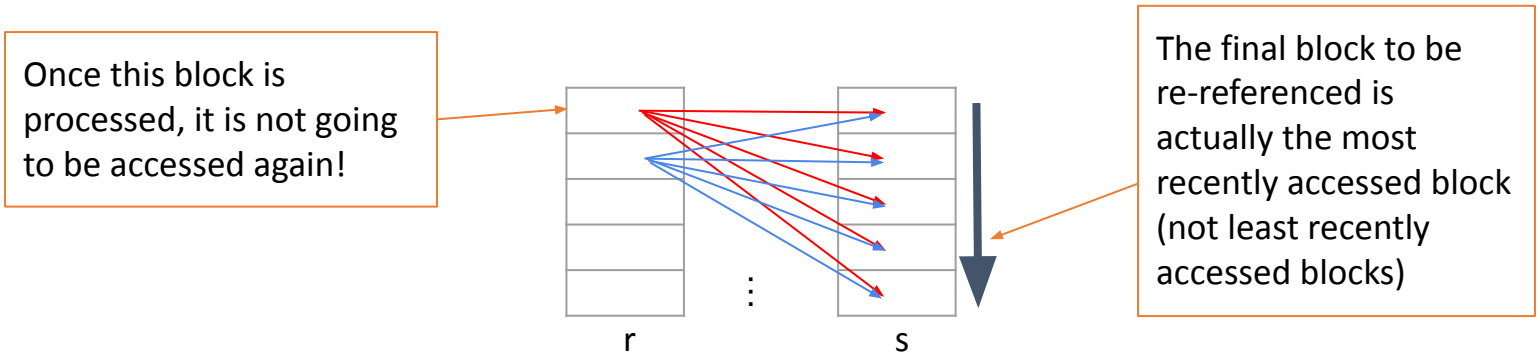


Every iteration of outer loop will start scanning s from the beginning again. LRU may have already evicted the required blocks.

Example Query

```
select *
from instructor natural join department;
     r                       s
```

Buffer Replacement Policies (Cont.)



- **Toss-immediate** strategy – frees the space occupied by a block as soon as the final tuple of that block has been processed
- **Most recently used (MRU) strategy** – after the final tuple of the block has been processed, the block becomes the most recently used block which will be the first block to be evicted if necessary.

Buffer Replacement Policies (Cont.)

- Buffer manager can also use statistical information regarding the probability that a request will reference a particular relation
  - E.g., the data dictionary is frequently accessed. Heuristic: keep data-dictionary blocks in main memory buffer
- Operating system or buffer manager may reorder writes
  - Can lead to corruption of data structures on disk
    - E.g., linked list of blocks with missing block on disk
    - File systems perform consistency check to detect such situations
  - Careful ordering of writes can avoid many such problems

Optimization of Disk Block Access (Cont.)

- Buffer managers support **forced output** of blocks for the purpose of recovery (more in Chapter 19)
- **Nonvolatile write buffers** speed up disk writes by writing blocks to a non-volatile RAM or flash buffer immediately
  - *Writes can be reordered to minimize disk arm movement*
- **Log disk** – a disk devoted to writing a sequential log of block updates
  - Used exactly like nonvolatile RAM
    - Write to log disk is very fast since no seeks are required
- **Journaling file systems** write data in-order to NV-RAM or log disk
  - Reordering without journaling: risk of corruption of file system data

Assignments

- Reading: Ch13.3-13.5
- Practice Excercises: 13.4, 13.5, 13.6, 13.7, 13.8

Solutions to the Practice Excercises:
https://www.db-book.com/Practice-Exercises/index-solu.html

Announcement

Quiz 1 on next Wed!

- 15 min for 10 questions - multiple choices, true/false, etc.
- Ch 12 & 13; every lecture material that we have covered upto today (excluding C++ tutorials and labs)
- Online Quiz on LMS - So, bring your laptops
- Not an open book; no electronic devices are allowed during the quiz except for the device you use to take the quiz.
- Please remain seated until the time is up
- Keep full screen all the time and do not chat with others (both online and offline)
  - Don’t cheat! You will get penalties for that (F grade and more)!

The End