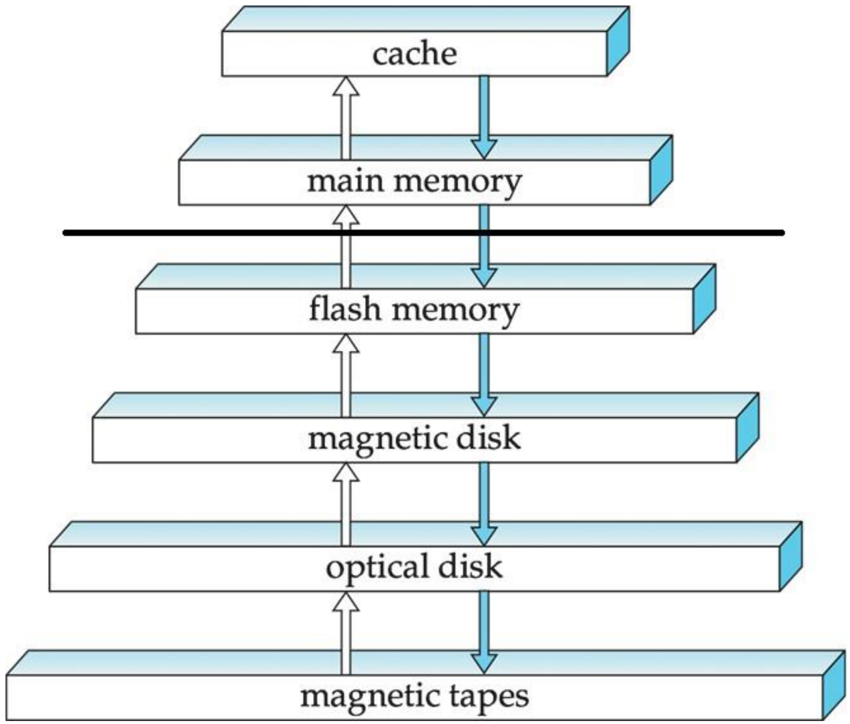
Buffer Manager and Sorting

Miao Qiao
The University of Auckland

Recap: Storage Hierarchy

Volatile
Random Access
Byte Addressable



Faster,
Smaller,
Expensive.

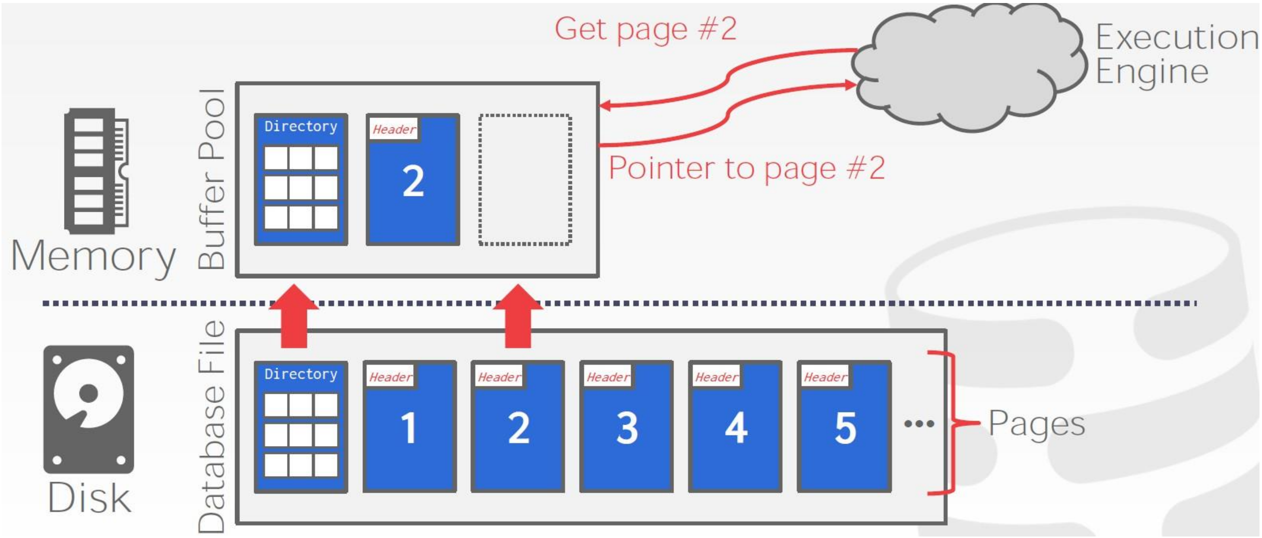
Non-volatile
Sequential Access
Block-Addressable

Slower,
Larger,
Cheaper.

- Data is transfered between the main memory and the disk in blocks.

Storge

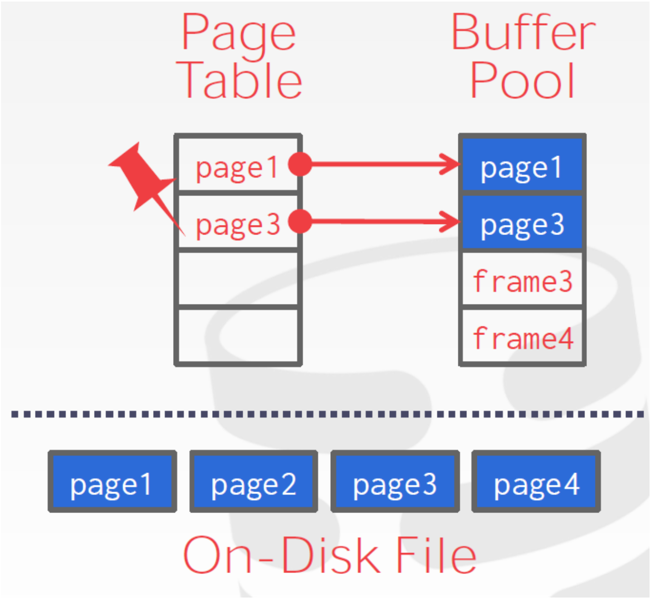
- **Buffer Pool**: available main memory used for storing copies of disk blocks.
- **Buffer Manager**: a DBMS subsystem that manages the buffer pool, aiming at minimizing I/O – the number of blocks transfered between the memory and the disk.



Buffer Manager

THE UNIVERSITY OF AUCKLAND
Te Whare Wānanga o Tāmaki Makaurau
NEW ZEALAND
SCIENCE
DEPARTMENT OF COMPUTER SCIENCE

- The buffer pool is an array of frames, the size of a frame is the size of a page.
- When the DBMS requests a page, a copy is placed in a frame.
- The page table maps the IDs of the pages that are currently in the buffer pool to the corresponding frames, and maintains the meta-data for each page
  - Dirty flag: a binary state. The page is dirty if the page is updated since loaded from the disk.
  - Pin: an integer, representing the number of threads that is using the page.
    - The page is unpinned if *pin* = 0.



Page Table VS Page Directory
Page directory: non-volatile mapping from Page IDs to Page locations.

Buffer Manager - Read

THE UNIVERSITY OF AUCKLAND Te Whare Wānanga o Tāmaki Makaurau NEW ZEALAND | SCIENCE DEPARTMENT OF COMPUTER SCIENCE

Read request: a page ID *X* .

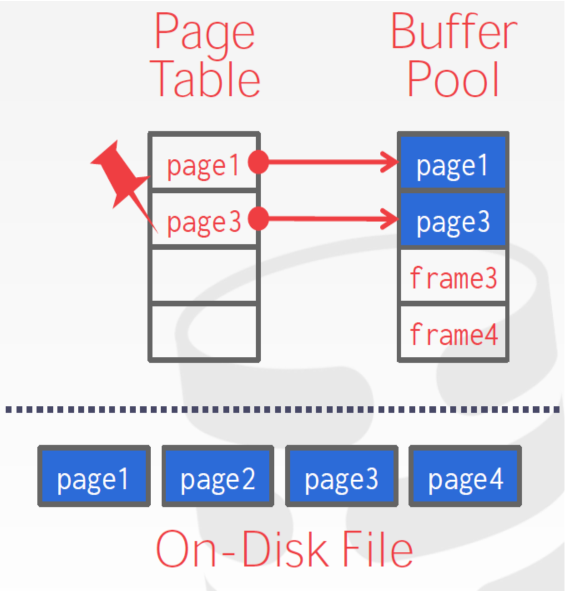
Buffer manager, upon receiving a read request

- Check the page table, if the page is not in the buffer pool,
  - if the buffer pool is full, perform buffer replacement to get an empty frame

  load *X* to the buffer and update the page table.
- Return the the content of the page from the corresponding frame.

Buffer replacement: Choose, among all the unpinned pages, one page *Y* based on a replacement policy

- If Page *Y* is dirty, write back *Y* to the disk, then kick the page out of the buffer pool.



Write back: either overwrite the original block, or write to a new location and then mark the original block as invalid.

Buffer Manager - Replacement Policy

Buffer replacement policy. When the DBMS needs to free up a frame to make room for a new page, it must decide which page to evict from the buffer pool.

- Goal: Increase the “hit rate” — the proportion of read requests whose page is in the buffer pool without triggering an I/O.

Policy 1: Least Recently Used (LRU).

- Maintain a single timestamp of when each page was last accessed.
- Select the one with the oldest timestamp to be evicted.

Heuristic: The page that is used more recently is more likely to be used later.

Buffer Manager - Replacement Policy

Buffer replacement policy. When the DBMS needs to free up a frame to make room for a new page, it must decide which page to evict from the buffer pool.

- Goal: Increase the “hit rate” — the proportion of read requests whose page is in the buffer pool without triggering an I/O.

Policy 2: Clock.

- Maintain a binary bit for each page, when access the page, set the bit to 1.
- Reset to bit to 0 using a “sweeping clock hand”, in particular,
- When the clock hand reaches a page and the bit of the page is 0 – the page has not been accessed for the past whole circle, evict the page; otherwise, set the bit to 0.

Heuristic: Approximate LRU without keeping a timestamp per page.

Buffer Manager - Replacement Policy

- Sequential flooding of Policies 1-2.
  - A query performs a sequential scan that reads every page.
  - This pollutes the buffer pool with pages that are read once and then never again.

- Example:
- Q1: SELECT AVG(VAL) FROM A
- Q2: SELECT $*$ FROM A WHERE VAL = 100
- Both queries are executed by sequentially scanning the blocks of relation *A*.

- Policy 3: Mose Recently Used (MRU).
  - Maintain a single timestamp of when each page was last accessed.
  - Select the one with the youngest timestamp to be evicted.
- Heuristic: The page that is used more recently is less likely to be used later.

Buffer Manager

THE UNIVERSITY OF AUCKLAND Te Whare Wānanga o Tāmaki Makaurau NEW ZEALAND | SCIENCE DEPARTMENT OF COMPUTER SCIENCE

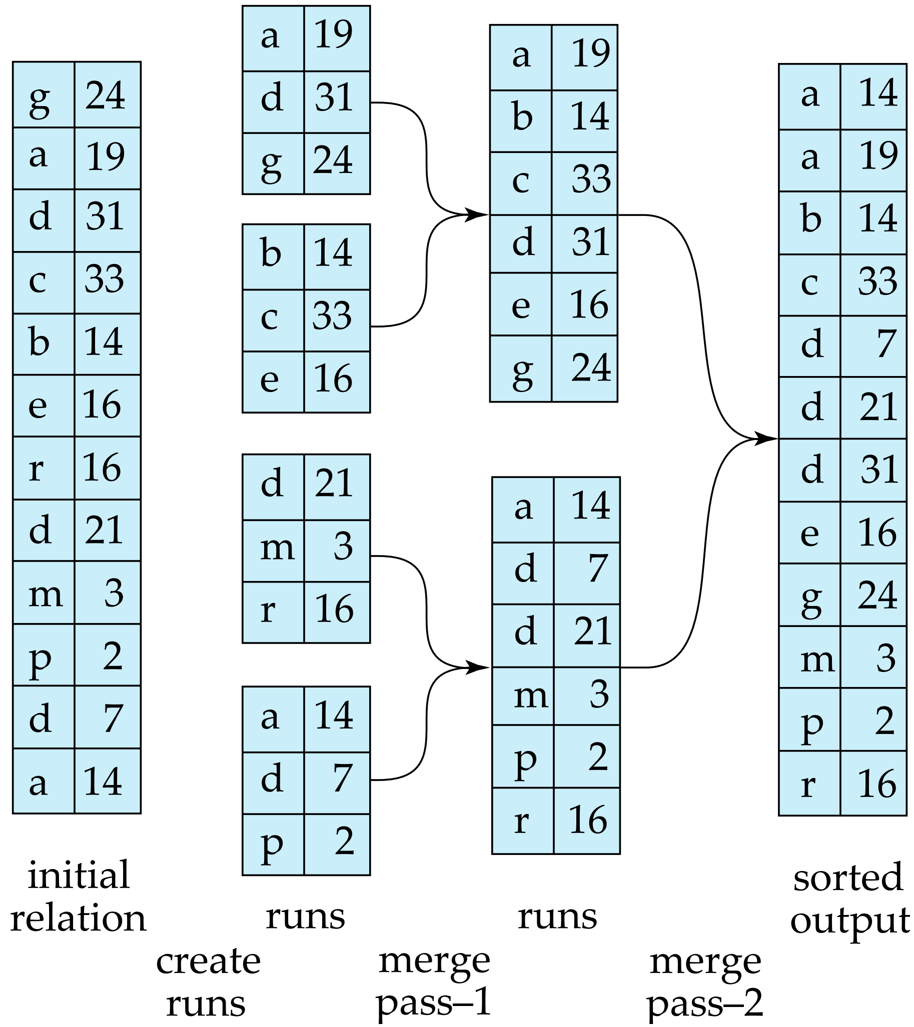
Other heuristics in improving the buffer management.

- Run-time statistics and query execution algorithms can be analyzed to predict which page is less likely to be used in future.
- Replacing a dirty page is slower than replacing a clean page.
- Background writing: the DBMS can periodically walk through the page table, write dirty pages back to the disk and reset the dirty flag.
- Engaging multiple buffer pools with different replacement policies
  - Per-database buffer pool,
  - Per-page buffer pool,
  - Sorting and join buffers,
  - Log buffers,
  - Query caches

Sorting

- We may build an index on the relation, and then use the index to read the relation in sorted order. May lead to one disk block access for each tuple.
- For relations that fit in memory, techniques like quicksort can be used.
  - For relations that don't fit in memory, **external sort-merge** is a good choice.

Example: External Sorting Using Sort-Merge



External Sort-Merge

Let M denote memory size (in pages).

1. **Create sorted runs**. Let i be 0 initially.
   Repeatedly do the following till the end of the relation:
   (a) Read M blocks of relation into memory
   (b) Sort the in-memory blocks
   (c) Write sorted data to run R_i; increment i.

Let the final value of i be N

2. **Merge the runs (N-way merge)**. We assume (for now) that $N < M$.
   1. Use N blocks of memory to buffer input runs, and 1 block to buffer output.
      Read the first block of each run into its buffer page
   2. **repeat**
      1. Select the first record (in sort order) among all buffer pages
      2. Write the record to the output buffer. If the output buffer is full write it to disk.
      3. Delete the record from its input buffer page.
         If the buffer page becomes empty **then**
         read the next block (if any) of the run into the buffer.
   3. **until** all input buffer pages are empty:

External Sort-Merge (Cont.)

- If $N \geq M$, several merge *passes* are required.
  - In each pass, contiguous groups of M - 1 runs are merged.
  - A pass reduces the number of runs by a factor of M -1, and creates runs longer by the same factor.
    - E.g. If M=11, and there are 90 runs, one pass reduces the number of runs to 9, each 10 times the size of the initial runs
  - Repeated passes are performed till all runs have been merged into one.

External Merge Sort (Cont.)

- Cost analysis for a relation with b_r blocks of tuples:
  - Total number of merge passes required: $\lceil \log_{M-1}(b_r/M) \rceil$.
  - Block transfers for initial run creation as well as in each pass is $2b_r$
    - for final pass, we don't count write cost
      - we ignore final write cost for all operations since the output of an operation may be sent to the parent operation without being written to disk
    - Thus, the total number of I/Os for external sorting:

$$b_r(2\lceil \log_{M-1}(b_r/M) \rceil + 1)$$

Exercise: EM Sort

- Suppose you need to sort a relation of 40 gigabytes, with 8-kilobyte blocks, using a memory size of 40 megabytes. Suppose the cost of a seek is 5 milliseconds, while the disk transfer rate is 40 megabytes per second.
  1. What is the time of transferring one block of data (without seeking)?
  2. Suppose a flash storage device is used instead of a disk, and it has a latency of 20 microsecond and a transfer rate of 400 megabytes per second. What is the time of transferring one block of data (without seeking)?
  3. How many merge passes are required?
  4. Find the cost of sorting the relation, in the number of I/Os.
  5. [After class exercise]

     What is the largest file size that can be sorted in at most 2 passes?

FIN

Any questions?