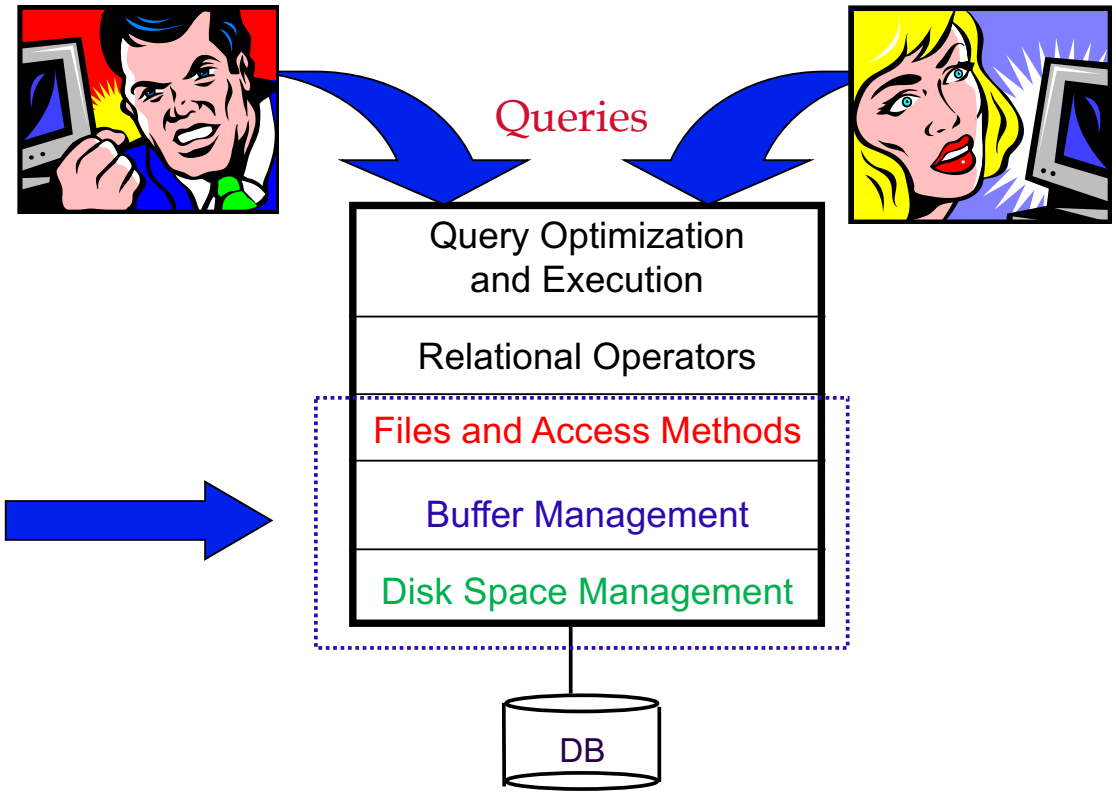
Database Management Systems

Buffer and File Management

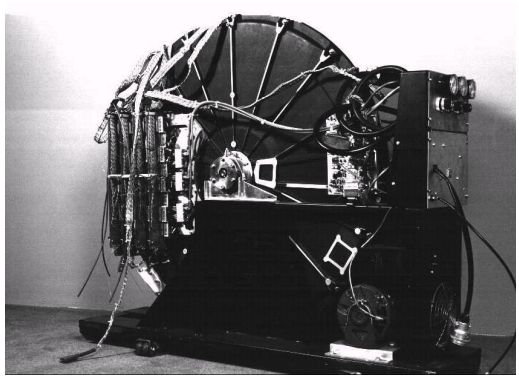
Fall 2017

"Yea, from the table of my memory
I'll wipe away all trivial fond records."
-- Shakespeare, *Hamlet*

The BIG Picture



Disks and Files



- DBMS stores information on disks.
  - In an electronic world, disks are a mechanical anachronism!
- This has major implications for DBMS design!
  - READ: transfer data from disk to main memory (RAM).
  - WRITE: transfer data from RAM to disk.
  - Both are high-cost operations, relative to in-memory operations, so must be planned carefully!

Why Not Store It All in Main Memory?

- *Costs too much*. $65 will buy you either around 8 GB of RAM DDR3 or around 2000 GB (2 TB) of disk today.
  - High-end Databases today can be in the Petabyte (1000TB) range.
  - Approx 60% of the cost of a production system is in the disks.
- *Main memory is volatile*. We want data to be saved between runs. (Obviously!)

- Note, some specialized systems do store entire database in main memory.
  - Vendors claim 10x speed up vs. traditional DBMS running in main memory.

Storage Access

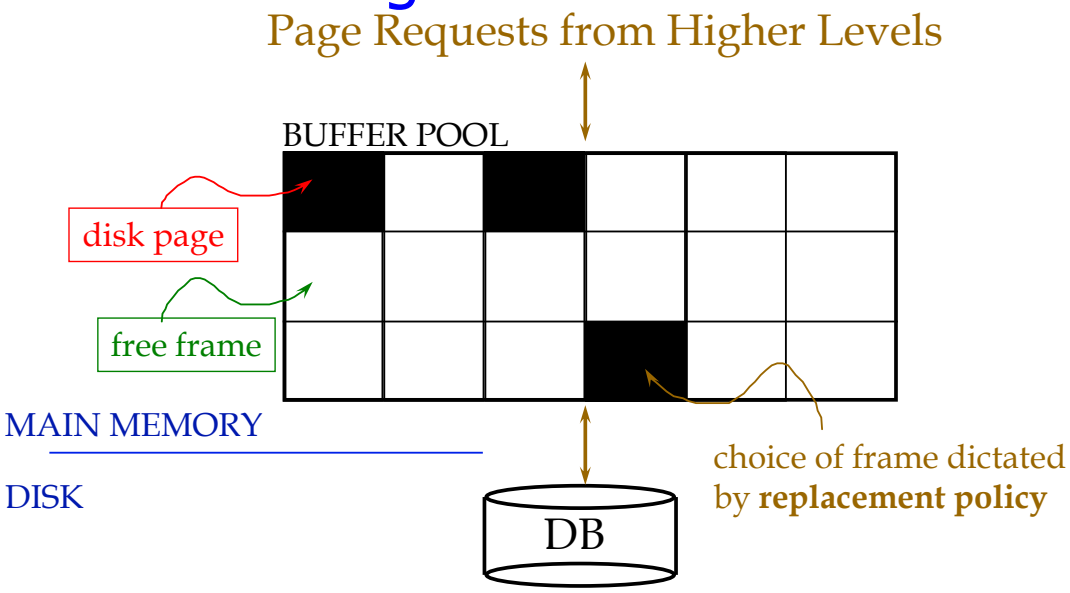
- A database file is partitioned into fixed-length storage units called **blocks**
  - Units of both storage allocation and data transfer.
  - Database system seeks to minimize the number of block transfers between the disk and memory.
  - Can reduce the number of disk accesses by keeping as many blocks 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.

More Terminology…

- Disk Page – the unit of transfer between the disk and memory
  Typically set as a config parameter for the DBMS.
  Typical value between 4 KBytes to 32 KBytes.
- Frame – a unit of memory
  Typically the same size as the Disk Page Size
- Buffer Pool –
  - *An area of memory into which database pages are read, modified, and held during processing*
  - A collection of frames used by the DBMS to temporarily keep data for use by the query processor.
    - note: We will sometime use the term "buffer" and "frame" synonymously.
- Pinned block – memory block that is not allowed to be written back to disk.

Question: When would you use a larger page size rather than a smaller one?

Buffer Management in a DBMS



- *Data must be in RAM for DBMS to operate on it!*
  - *The query processor refers to data using virtual memory addresses.*
- *Buffer Mgr hides the fact that not all data is in RAM*

When a Page is Requested ...

- If requested page IS in the pool:
  - *Pin* the page and return its address.
- Else, if requested page IS NOT in the pool:
  - If a free frame exists, choose it, Else:
    - Choose a frame for *replacement (only un-pinned pages are candidates)*
    - If chosen frame is "dirty", write it to disk
  - Read requested page into chosen frame
  - *Pin* the page and return its address.

Buffer Control Blocks (BCBs):

<frame#, pageid, pin_count, dirty>

- A page may be requested many times, so
  - a *pin count* is used.
  - To pin a page, pin_count++
  - A page is a candidate for replacement iff *pin_count* == 0 (*"unpinned"*)
- Requestor of page must eventually unpin it.
  - pin_count--
- Must also indicate if page has been modified:
  - *dirty* bit is used for this.

  Q: Why is this important?

Additional Buffer Manager Notes

- BCB's are hash indexed by pageID
- Concurrency Control & Recovery may entail additional I/O when a frame is chosen for replacement.

  (*Write-Ahead Log* protocol; more later.)
- If requests can be predicted (e.g., sequential scans) pages can be pre-fetched several pages at a time.

Buffer Replacement Policy

- Frame is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU), MRU, Clock, etc.
- This policy can have big impact on the number of disk reads and writes.
  - Remember, these are slooooooooooow.
- **BIG IDEA** – throw out the page that you are least likely to need in the future.
  - Q: How do you predict the future?
- Efficacy depends on the *access pattern*.

LRU Replacement Policy

- *Least Recently Used (LRU)*

1) for each page in buffer pool, keep track of time last *unpinned*

- *What else might you keep track off?*
- *How would that impact performance?*

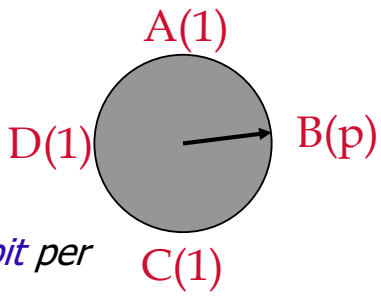
2)Replace the frame that has the oldest (earliest) time

- Most common policy: intuitive and simple
  - Based on notion of "Temporal Locality"
  - Works well to keep "working set" in buffers.
- Implemented through doubly linked list of BCBs
  - Requires list manipulation on unpin

Some issues with LRU

- *Problem: Sequential flooding*
  - LRU + repeated sequential scans.
  - # buffer frames < # pages in file means each page request causes an I/O. *MRU* much better in this situation (but not in all situations, of course).
- *Problem: "cold" pages can hang around a long time before they are replaced*
  - Cold pages are pages that have been touched only once recently

"Clock" Replacement Policy



- An approximation of LRU
- Arrange frames into a cycle, store one *reference bit per frame*
  - Can think of this as the 2nd chance bit
- When pin count reduces to 0, turn on ref. bit
- When replacement necessary

```
do for each page in cycle {
        if (pincount == 0 && ref bit is on)
                turn off ref bit;
        else if (pincount == 0 && ref bit is off)
                choose this page for
                        replacement;
} until a page is chosen;
```

Questions:
How like LRU?
Problems?

"2Q" Replacement Policy

- One Queue (A1) has pages that have been referenced only once.
  - new pages enter here
- A second, LRU Queue (Am) has pages that have been referenced (pinned) multiple times.
  - pages get promoted from A1 to here
- Replacement victims are usually taken from A1
  - Q: Why????