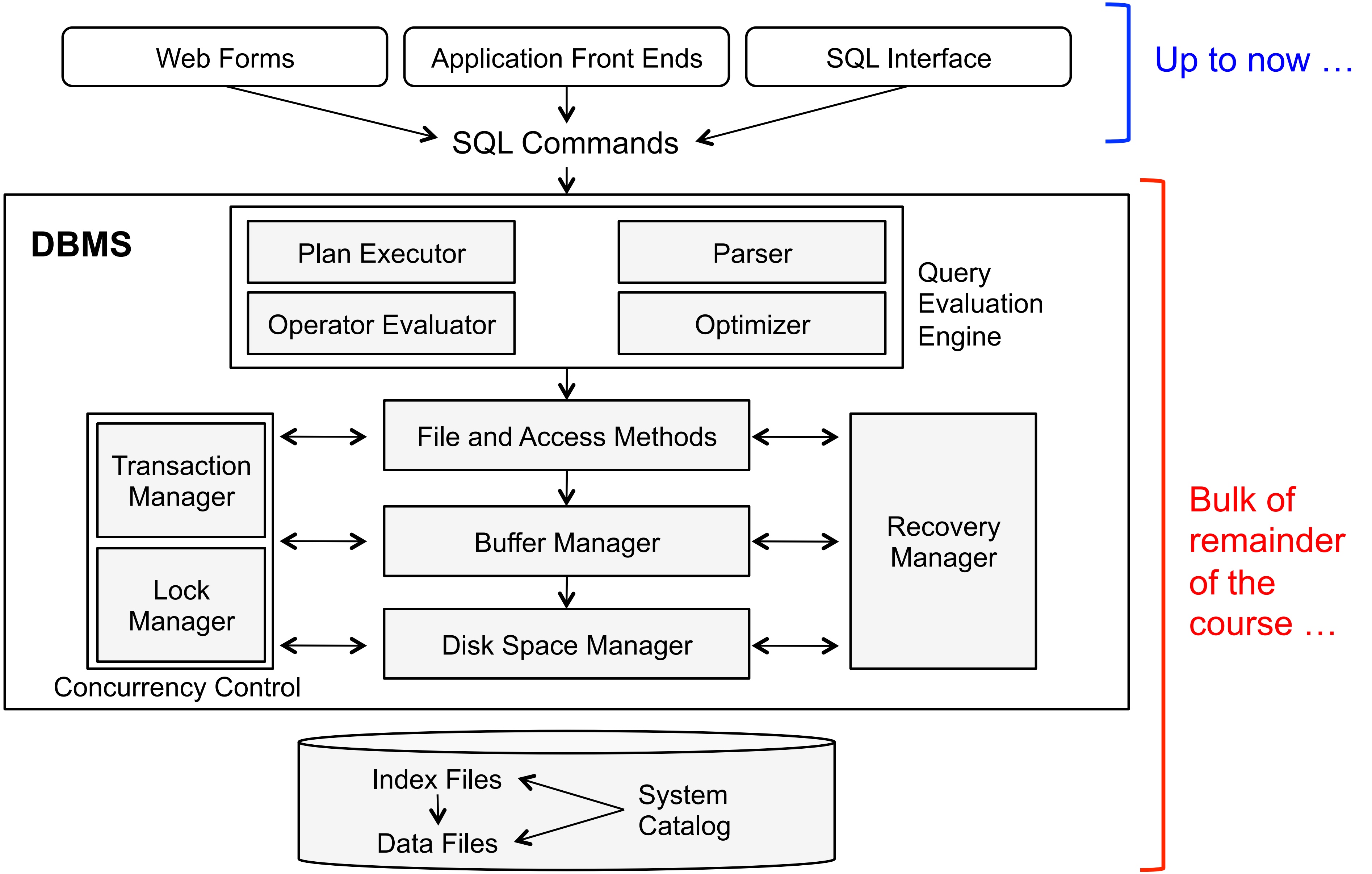
Announcements & Overview

- Midterm results: Wednesday
- HW3 due Wednesday @ 11:59pm
- Start on Database Internals: Data Storage on Disk
  - Warning: broad overview … "typical" cases, generalities
  - Many of these topics covered in more depth in 165B
- Reading
  - Chapter 13

Basic Database Architecture



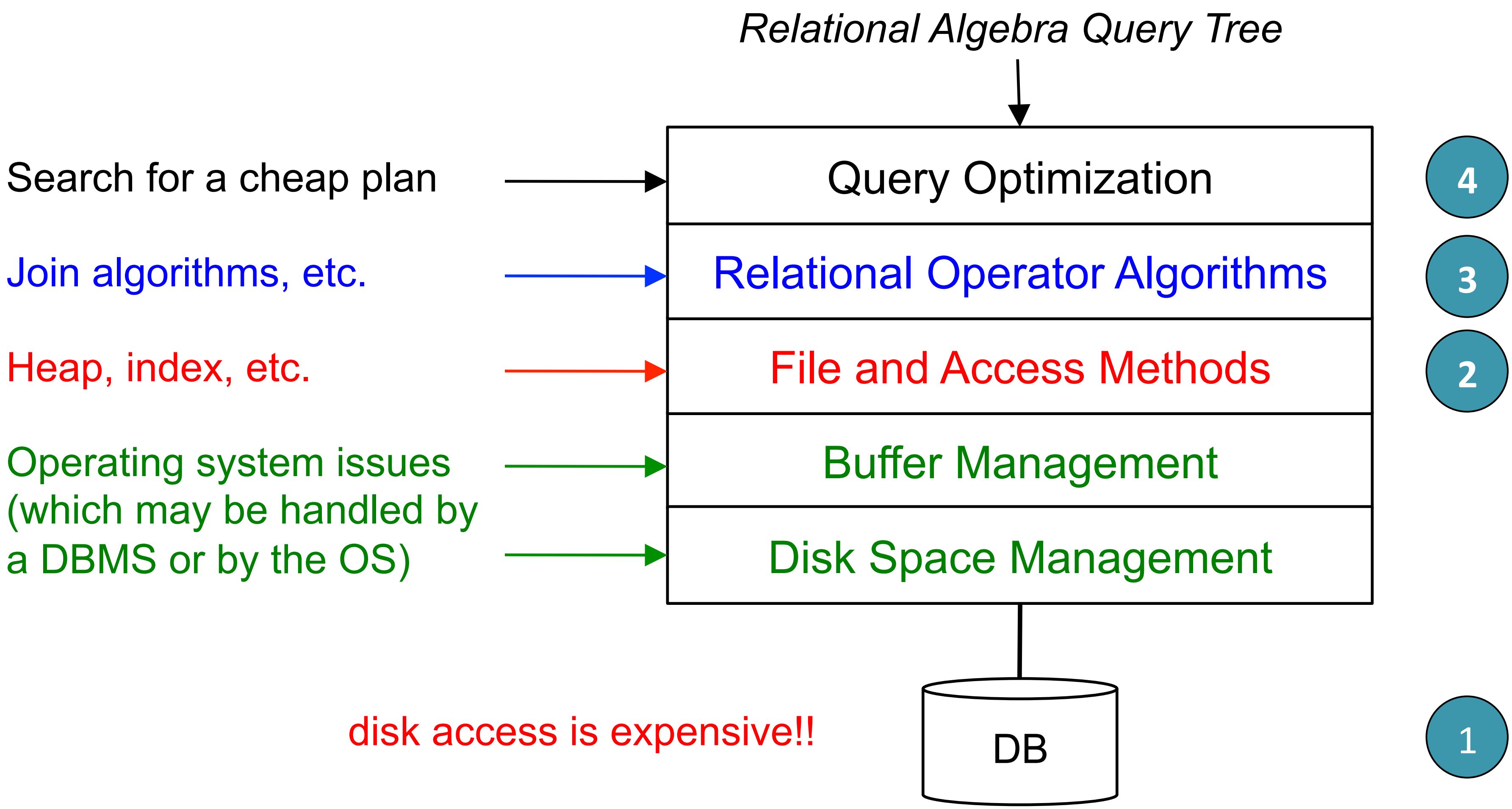
10,000 Foot View of Query Optimization

- Given an SQL query
- Translate it into relational algebra
- Find equivalent query plans
  - different ways to order operators
  - different ways to implement each operator
- Pick a cheap plan (per estimated cost)
- Execute the plan …

How are operators implemented?

How is data stored on disk?

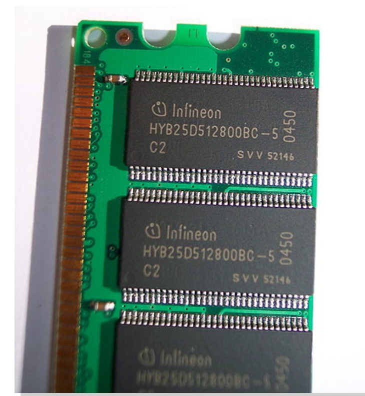
The Plan


Relational Algebra Query Tree
Search for a cheap plan
Query Optimization
4
Join algorithms, etc.
Relational Operator Algorithms
3
Heap, index, etc.
File and Access Methods
2
Operating system issues (which may be handled by a DBMS or by the OS)
Buffer Management
Disk Space Management
disk access is expensive!!
DB
1

Types of Physical Storage

- Cache
  - fastest and most costly form of storage
  - volatile … content lost if power failure, system crash, etc.
  - managed by the hardware and/or operating system

- Main memory
  - fast access
  - in most applications, too small to store an entire DB
  - Volatile



<u>Note</u>: many "main memory only" databases are available … and used increasingly for applications with small storage requirements and as memory sizes increase

Types of Physical Storage

- Magnetic ("Hard") Disk Storage
  - primary medium for long-term storage of data
  - typically can store entire database (all relations and access structures)
    - data must be moved from disk to main memory for access and written back for storage
    - direct-access, i.e., it is possible to read data on disk in any order
    - usually survives power failures and system crashes (disk failure can occur, but less frequently)

We focus on disk storage!

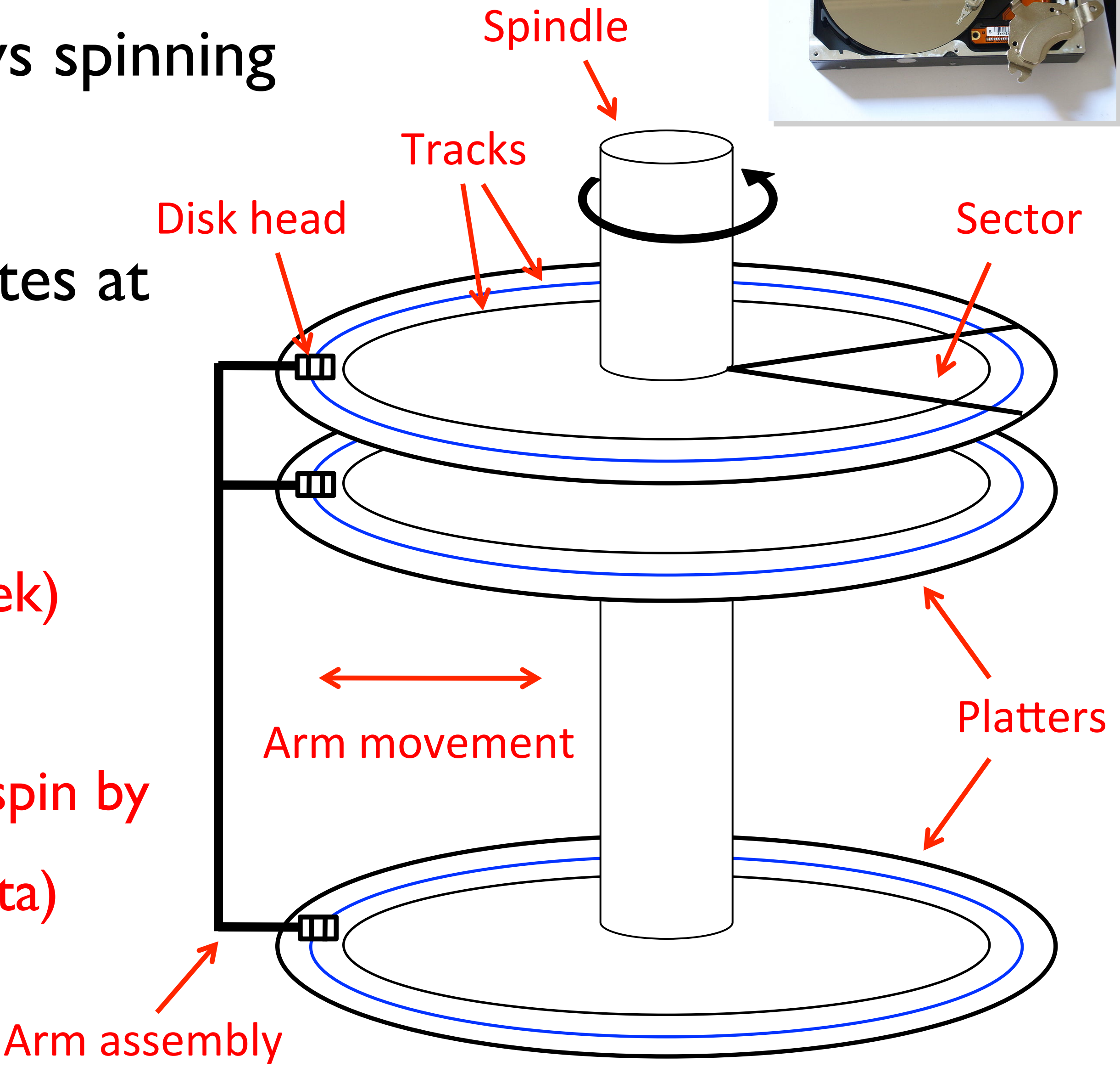
Types of Physical Storage

- Optical Storage
  - non volatile
  - e.g., CD-ROM, DVD
  - write-once-read-many optical disks used for archival storage

- Tape Storage
  - non volatile
  - used primarily for backup and export (to recover from disk failures and restore data)
  - often used for archival
  - tapes are typically much cheaper storage

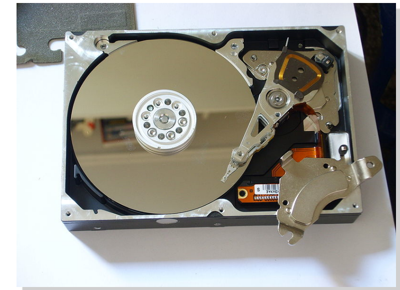
Components of a Disk

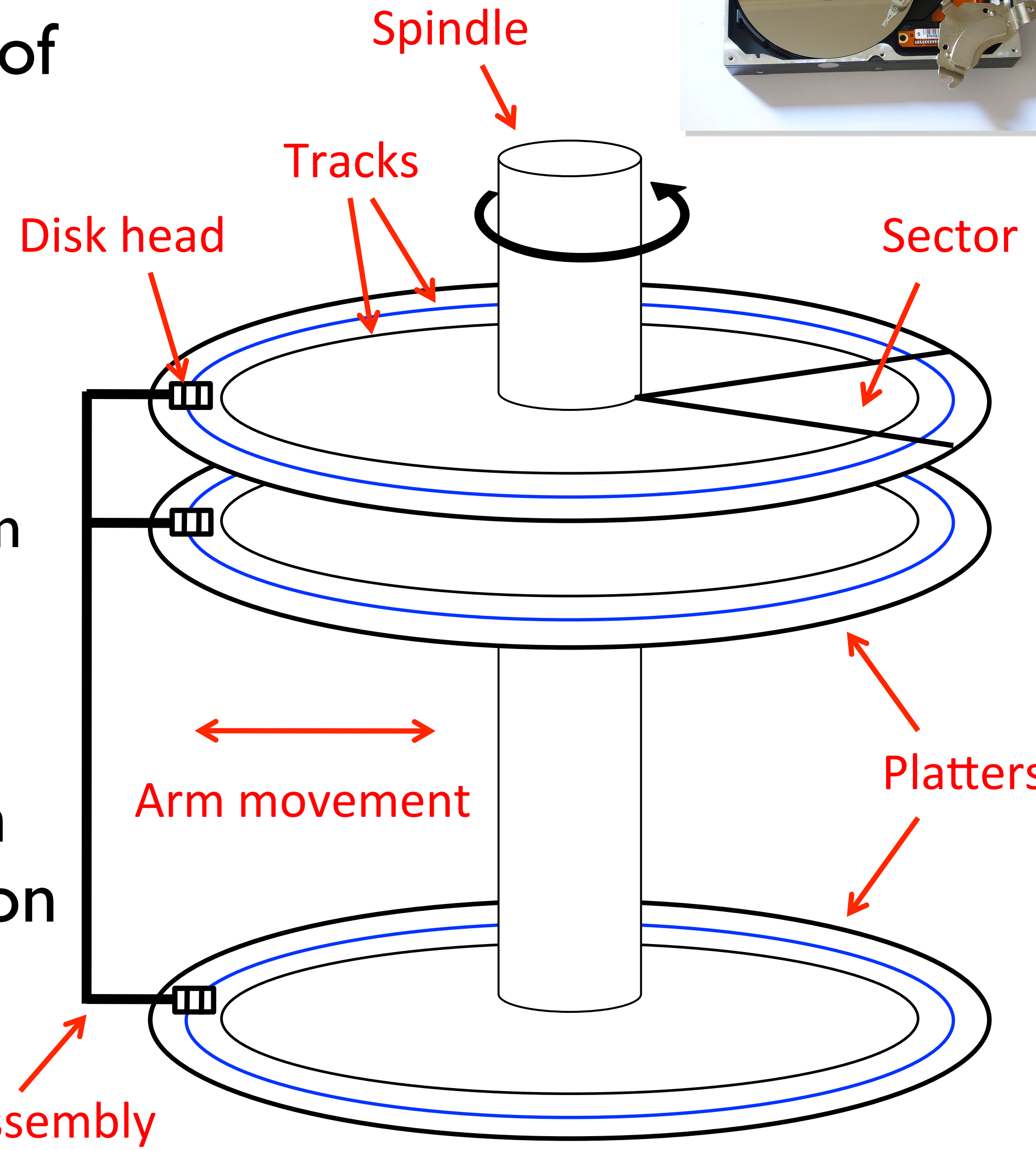
- Platters are always spinning (e.g., 7,200 rpm)
- A head reads/writes at any one time
- To read a record:
  - position arm (seek)
  - engage head
  - wait for data to spin by
  - read (transfer data)



Components of a Disk

- Each *track* is made up of fixed size *sectors*
- *Page size* is a multiple of *sector size*
  - unit of transfer
  - size depends on system and configuration
  - e.g., 4kb
- All tracks that you can reach from one position of the arm is called a *cylinder* (imaginary!)





Cost of Accessing Data on a Disk

- Time to access (read/write) data

  seek time = moving arms to position disk head on track

  rotational delay = waiting for sector to rotate under head

  transfer time = actually moving data to/from disk surface

- Key to lower I/O cost: *reduce seek & rotational delays!*
  - you have to wait for the transfer time, no matter what

- Query cost often measured in number of page I/Os
  - often simplified to assume each page I/O costs the same
  - random I/O is more expensive than Sequential I/O

Memory versus Disk Access Time

- Lets say disk access time (all three costs together) is about 5 milliseconds (ms) … and memory access time is about 50 nanoseconds (ns)
  - 5 ms = 5,000,000 ns
  - therefore disk access is 100,000 times slower than memory access!
- Contrast this with:
  - 1 second (e.g. pick up a piece of paper)
  - 100,000 seconds ~ 28 hours (a looooong drive…)

Block (Page) Size vs. Record Size

- The terms “block” and “page” are often used interchangeably …

  … (e.g., depending on how a DBMS is implemented)

- A block generally refers to a contiguous sequence of sectors from a single track

  – unit of (physical) storage on a disk, and transfer between main memory and disk

  – a page is a “block” in logical memory … smallest unit of transfer supported by an OS (virtual memory, paging)

  – Pages/blocks range in size (typically around 512b to 8kb)

Block (Page) Size vs. Record Size

- A database system seeks to *minimize* the number of block transfers between disk and main memory

- Transfer can be reduced by keeping as many blocks as possible in main memory
  - Buffer is the portion of main memory available to store copies of disk blocks
  - Buffer manager is responsible for allocating and managing buffer space

- If possible, store file blocks sequentially:
  - Consecutive blocks on same track, followed by
  - Consecutive tracks on same cylinder, followed by
  - Consecutive cylinders adjacent to each other
  - First two incur no seek time or rotational delay, seek for third is only one track

Buffer Manager

- Program calls buffer manager when it needs blocks from disk
  - the program is given the address of the block in main memory, if it is already in the buffer
  - if block not in buffer, the buffer manager adds it …
    - Replaces (*throws out*) other blocks to make space
    - The thrown out block is written back to the disk if it was modified (since last write to disk)
    - Once space is allocated, the buffer manager reads in the block from disk to the buffer and returns the address

Buffer Replacement Policies

- Operating systems often replace the block *least recently used* (LRU strategy)
  - In LRU, past (use) is a predictor of future (use)
- Alternatively, queries have well-defined access patterns (e.g., sequential scans)
  - A database system can exploit user queries to predict block accesses
  - LRU can be an inefficient strategy for certain access patterns that involve (e.g., repeated) sequential scans
  - The query optimizer can provide hints on replacement strategies

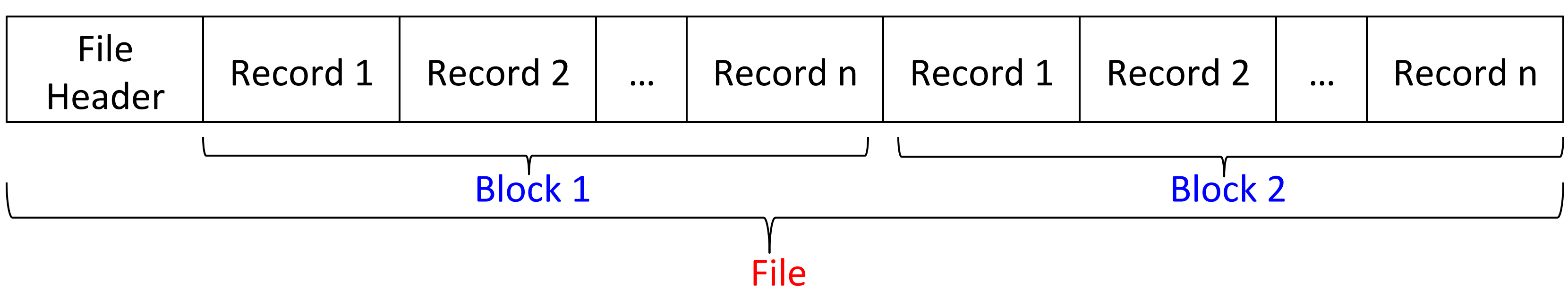
Buffer Replacement Policies

Pinned block = not allowed to be written back to disk

- *Most recently used* (MRU) strategy
  - Pin the block currently being processed
  - After final tuple of that block processed, the block is unpinned and becomes the most recently used block
  - Keeps older blocks around longer (good for scan problem)
- Buffer manager can use statistics regarding the probability that a request will reference a particular relation

File Organization

- A database can be stored as a collection of files
- Record-oriented storage
  - Each file is a sequence of records
  - Each record is a sequence of fields
- Typical organization of records in files
  - Assume the record size is ***fixed*** (not always the case …)
  - Each file has records of one particular type only
  - … different files used for different relations



File Organization

- We also will draw blocks like this:

Block 1

<v1, v2, …, vn>
<v1, v2, …, vn>
<v1, v2, …, vn>
<v1, v2, …, vn>
<v1, v2, …, vn>

Block 2

<v1, v2, …, vn>
<v1, v2, …, vn>
<v1, v2, …, vn>
<v1, v2, …, vn>
<v1, v2, …, vn>

Fixed-Length Records

- Simple approach
  - Store record i starting at byte $n * (i - 1)$, where n is the size of each (fixed-length) record
  - Record access is simple, but records may span blocks
- Deletion of record i (to avoid fragmentation)
  - Move (shift) records $i + 1, \ldots, n$ to $i, \ldots, n - 1$
  - Move record n to i
  - Maintain positions of free records in a *free list*

Fixed-Length Records

Free Lists

– In the file header, store the address of the first record whose content is deleted

– Use this first record to store the address of the second available record, and so on

– These stored addresses act as "pointers" … they "point" to the location of a record (like a linked list)

– Tricky to get right (often the case with pointers)

Variable-Length Records

- Variable-length records are often needed
  - for record types that allow a variable length for one or more fields (e.g., varchar)
  - if a file is used to store more than one relation

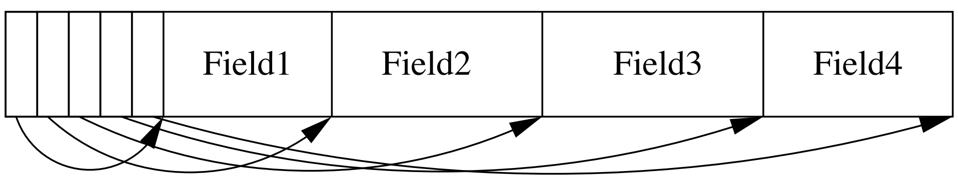
Approaches for storing variable length records

- End-of-record markers
  - Fields "packed" together
  - Difficult to reuse space of deleted records (fragmentation)
  - No space for record to grow (e.g., due to an update)
  - … in this case, must move the record

Variable-Length Records

- Field delimiters
  - Requires scan of record to get to n-th field value
  - Requires a field for a NULL value

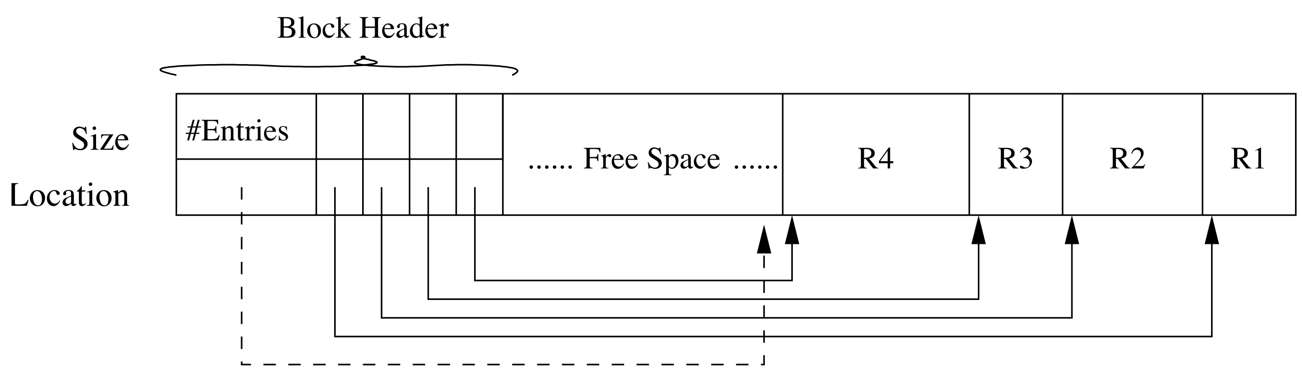
- Each record as an array of field offsets
  - For overhead of the offset, we get direct access to any field
  - NULL values represented by assigning begin and end pointers of a field to the same address



Variable-Length Records

- Can cause problems when attributes are modified
  - growth of a field requires shifting all other fields
  - a modified record may no longer fit into the block
  - a (large) record can span multiple blocks

- Block headers
  - maintain pointers to records
  - contain pointers to free space area
  - records inserted from end of the block
  - records can be moved around to keep them contiguous



Organization of Records within a File

- Requirements: must efficiently support
  - insert/delete/update of a record
  - access to a record (typical using rid)
  - scan of all records
- Heap File (unsorted file)
  - Simplest file structure
  - Records are unordered
  - Record can be placed anywhere in the file where there is space
- Sequential File
  - Records are ordered according to a search key
- Clustered Index
  - related to seqential files, we'll discuss in Section 7

Heap File Organization

- At DB runtime, pages/blocks are allocated and deallocated

- Information to maintain for a heap file includes pages, free space on pages, records on a page

- A typical implementation is based on two doubly-linked lists of pages, starting with a header block

- Two lists can be associated with header block: (1) full page list, and (2) list of pages having free space

Sequential File Organization

- Suitable for applications that require sequential processing of the entire file
- Records in file are ordered by a *search key*
- Deletions of records managed using pointer chains
- Insertions: must locate the position in the file where the record is to be inserted
  - if there is free space, insert record there
  - if no free space, insert record in an *overflow block*
  - in either case, pointer chain must be updated
- If many record modifications (esp. insertions and deletions), correspondence between search key order and physical order can be totally lost => file reorganization