Disks, Memories & Buffer Management

“The two offices of memory are collection and distribution.”
- Samuel Johnson

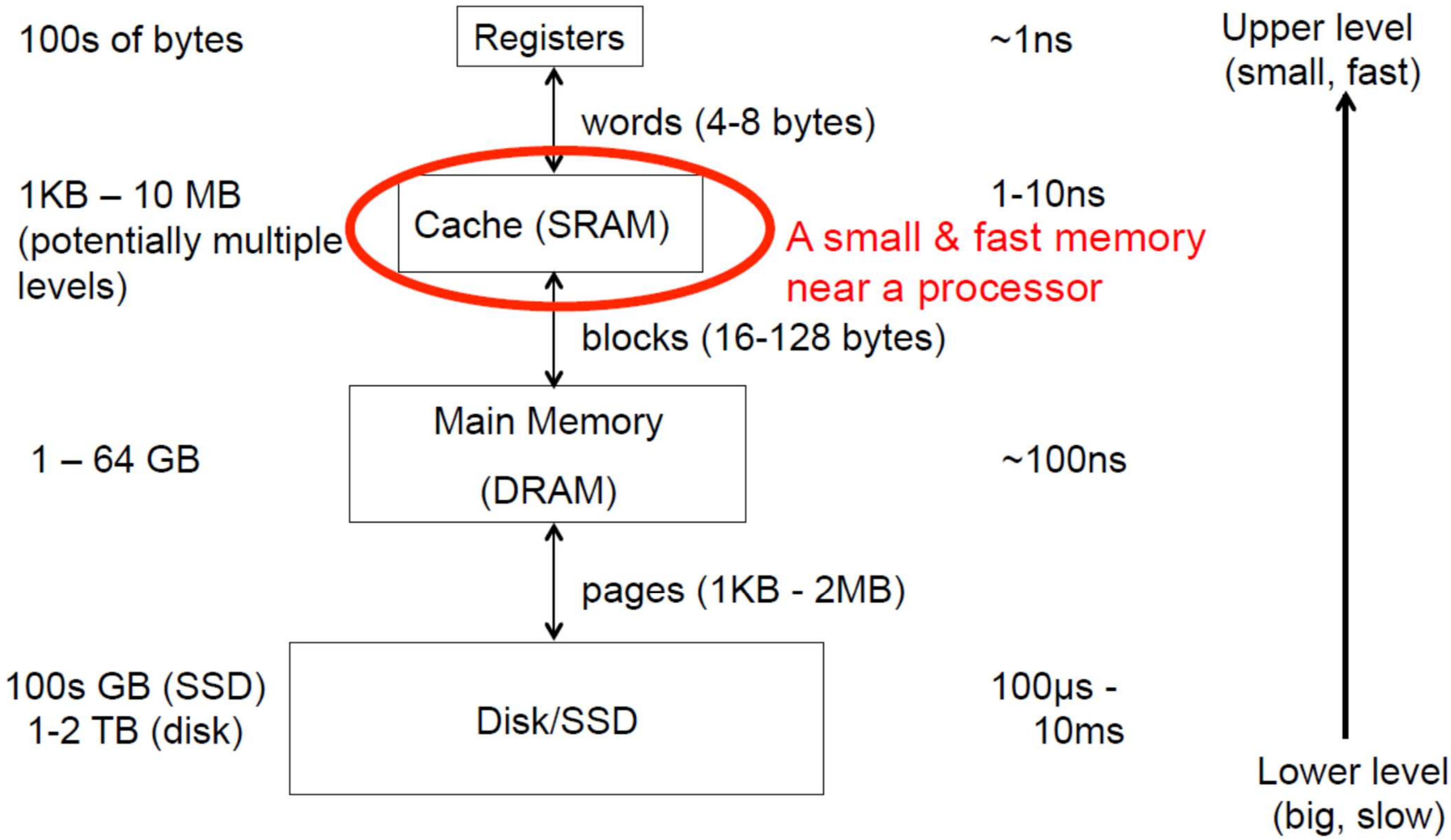
What does a DBMS Store?

- Relations – Actual data
- Indexes – Data structures to speed up access to relations
- System catalog (a.k.a. data dictionary) stores metadata about relations
  - Relation schemas – structure of relations, constraints, triggers
  - View definitions
  - Statistical information about relations for use by query optimizer
  - Index metadata
- Log files – information maintained for data recovery

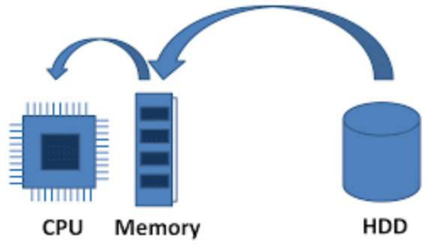
Where are the data stored?

- Memory Hierarchy
  - Primary memory: registers, static RAM (caches), dynamic RAM (physical memory)
    - Currently used data
  - Secondary memory: magnetic disks (HDD), solid state disks (SSD)
    - Main database
    - SSD can also be used as an intermediary between disk and RAM
  - Tertiary memory: optical disks, tapes, jukebox
    - Archiving older versions of the data
    - Infrequently accessed data
- Tradeoffs:
  - Capacity
  - Cost
  - Access speed
  - Volatility vs non-volatility

Memory Hierarchy



Data Access

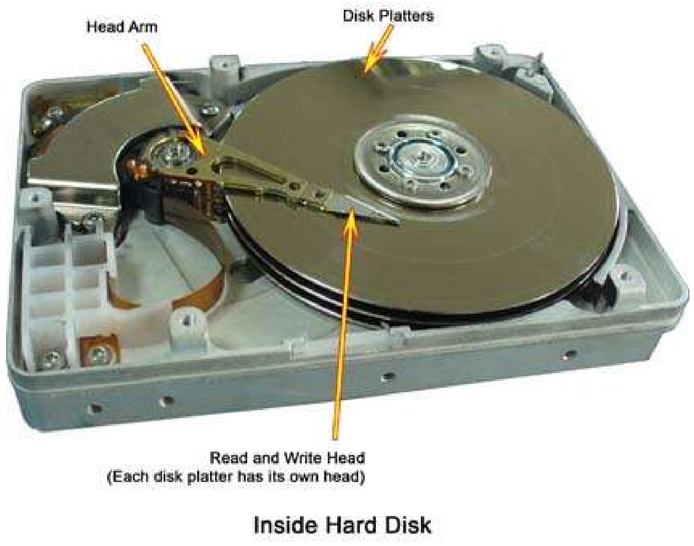


- DBMS *stores* information on non-volatile ("hard") disks
- DBMS *processes* data in main memory (RAM)
- This has major implications for DBMS design!
  - READ: transfer data from disk to main memory (RAM)
  - WRITE: transfer data from RAM to disk
  - Both are *high-cost* operations, relative to in-memory operations, so must be planned carefully!

Disks

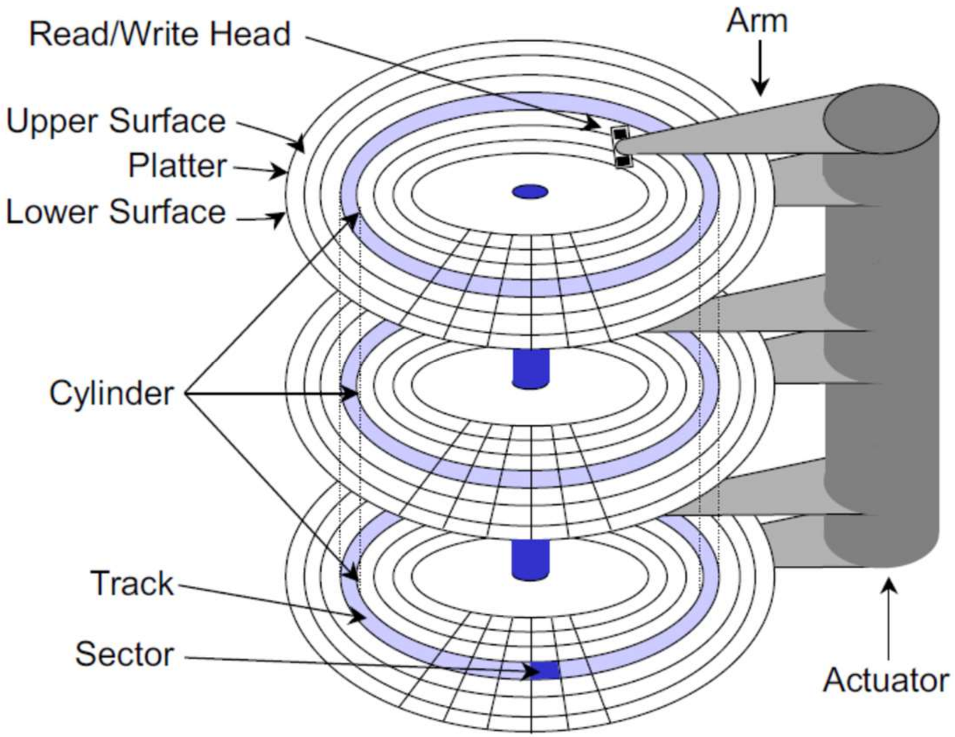
- Secondary storage device of choice
- Offers *random access* to data
- Data is stored and retrieved in units called *disk pages or blocks (consecutive number of pages)*
  - Typical page size is 4KB – 1MB
  - Typical block size is 1MB – 64MB
- Unlike RAM, time to retrieve a disk page varies depending upon its "relative" *location* on disk *at the time of access*
  - Therefore, relative placement of pages on disk has major impact on DBMS performance!

Components of a Disk



The platters spin (say, 120rps)

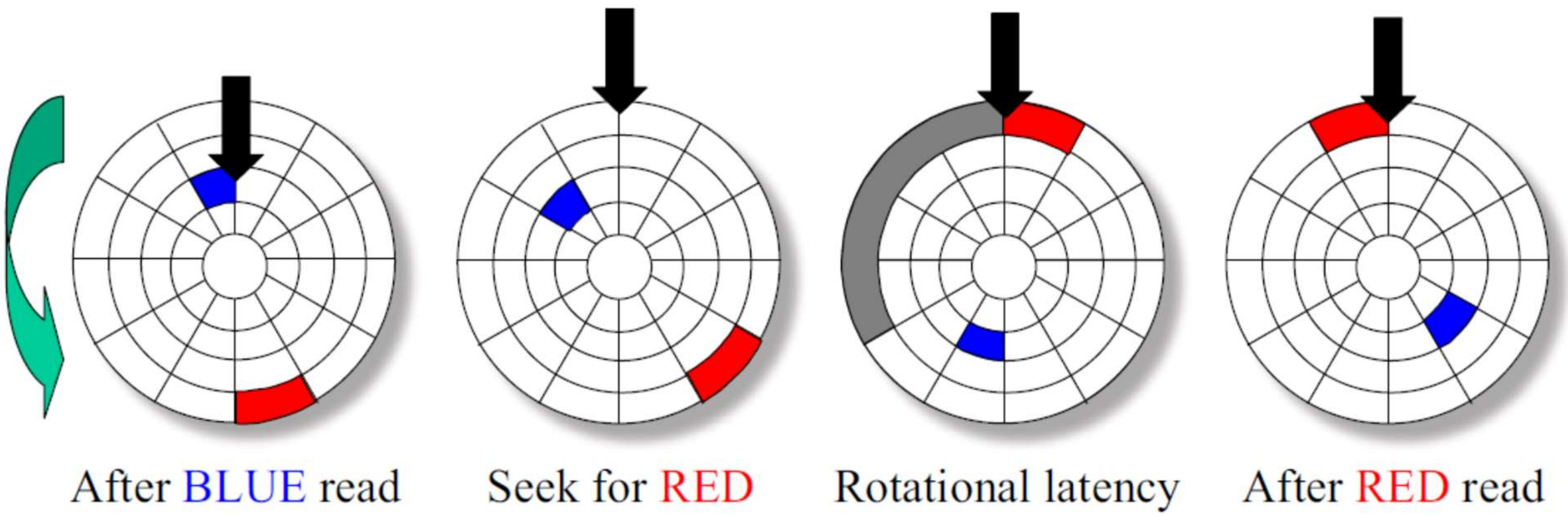
The arm assembly is moved in or out to position a read/write head on a desired track. Tracks under the head make a (imaginary) *cylinder*



Only one head reads/writes at any one time

Block size is a multiple of *sector size* (which is fixed)

Components of Disk Access Time



Source: R. Burns' slides on storage systems

Accessing a Disk Page

- Time to access (read/write) a disk block:
  - *seek time* (moving arms to position disk head on track)
  - *rotational delay* (waiting for block to rotate under head)
  - *transfer time* (actually moving data to/from disk surface)
- Seek time and rotational delay dominate
  - Seek time varies from about 0.3 to 10msec
  - Rotational delay varies from 0 to 4msec
  - Transfer rate is about 0.05msec per 8KB page
- Key to lower I/O cost: reduce seek/rotation delays!

Improving Access Time of Secondary Storage

- Organization of data on disk
- Disk scheduling algorithms
- Multiple disks or Mirrored disks
- Prefetching and large-scale buffering
- Algorithm design

An Example

- How long does it take to read a 2,048,000-byte file that is divided into 8,000 256-byte records assuming the following disk characteristics?

average seek time	18 ms
track-to-track seek time	5 ms
average rotational delay	8.3 ms
maximum transfer rate	16.7 ms/track
bytes/sector	512
sectors/track	40
tracks/cylinder	11
tracks/surface	1,331

- 1 track contains 40*512 = 20,480 bytes, the file needs 100 tracks (~10 cylinders)

Design Issues

- Randomly store records
  - suppose each record is stored randomly on the disk
  - reading the file requires 8,000 random accesses
  - each access takes 18 (average seek) + 8.3 (average rotational delay) + 0.8 (transfer two sectors*) = 27.1 ms
  - total time = 8,000*27.1 = 216,800 ms = 216.8 s

* Assume that one page/block consists of 2 sectors

Design Issues

- Store on adjacent cylinders
  - need 100 tracks ~ 10 cylinders
  - read first cylinder = 18 + 8.3 + 11*16.7 = 210 ms
  - read next 9 cylinders = 9*(5+8.3+11*16.7) = 1,773* ms
  - total = 1,983 ms = 1.983 s

- Blocks in a file should be arranged sequentially on disk to minimize seek and rotational delay!

* The actual value is smaller as the last cylinders does not need to read all 11 tracks

Why Not Store Everything in Main Memory?

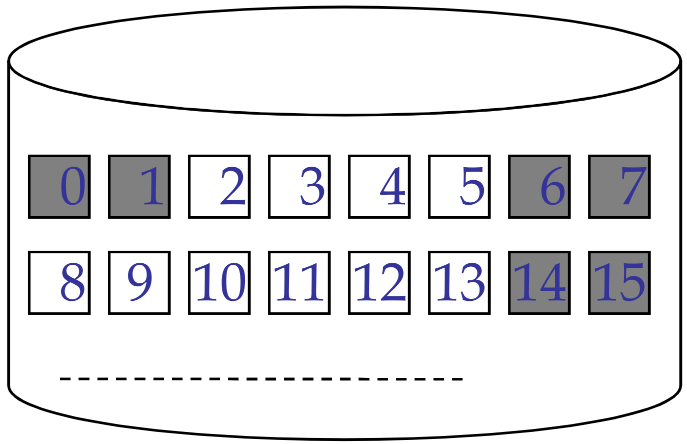
Disk Space Management

- Many files will be stored on a single disk
- Need to allocate space to these files so that
  - disk space is effectively utilized
  - files can be quickly accessed
- Several issues
  - How is the free space on a disk managed?
    - system maintains a *free space list* -- implemented as *bitmaps or link lists*
  - How is the free space allocated to files?
    - granularity of allocation (blocks, extents)
    - allocation methods (*contiguous, linked*)
  - How is the allocated space managed?

Managing Free Space: Bitmap

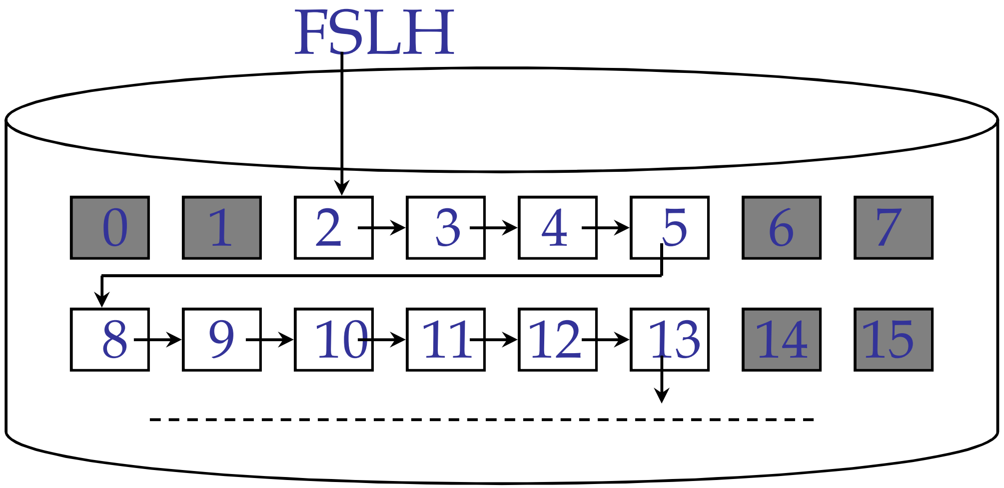
- Each block (one or more pages) is represented by one bit
- A bitmap is kept for all blocks in the disk
  - if a block is free, its corresponding bit is 0
  - if a block is allocated, its corresponding bit is 1
- To allocate space, scan the map for 0s

- Consider a disk whose blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, etc. are free. The bitmap would be
  - 110000110000001...



Managing Free Space: Link Lists

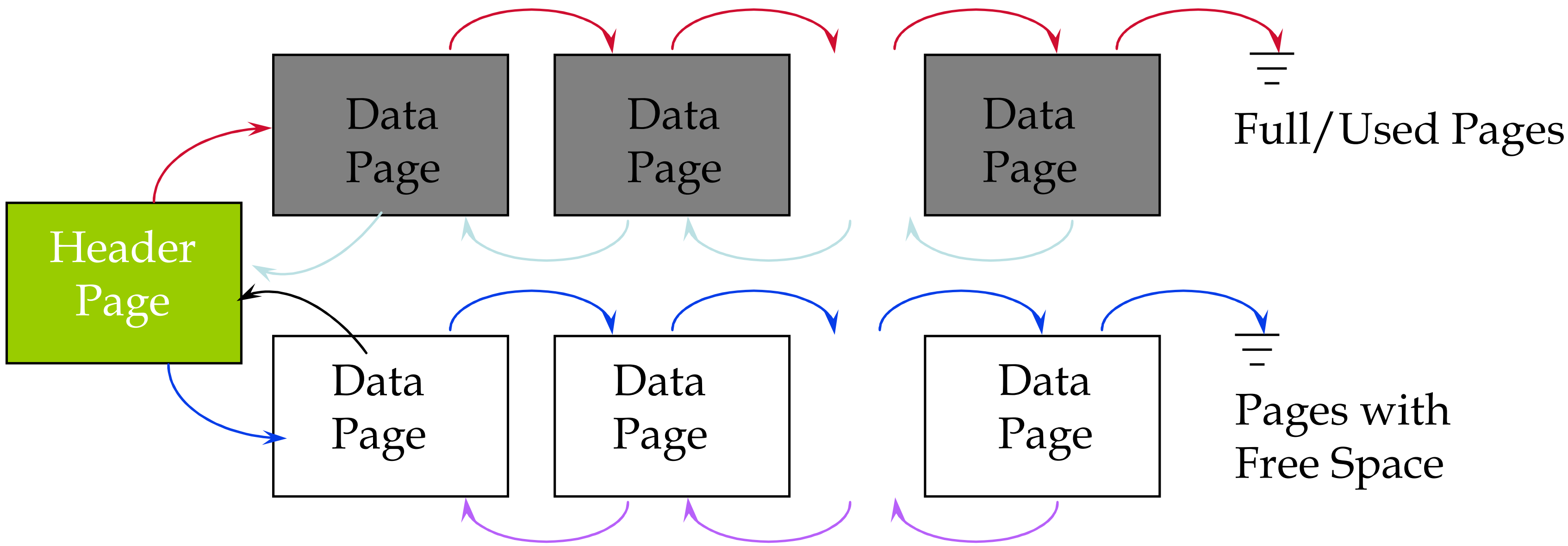
- Link all the free disk blocks together
  - each free block points to the next free block
- DBMS maintains a *free space list head* (FSLH) to the first free block
- To allocate space
  - look up FSLH
  - follow the pointers
  - reset the FSLH



Allocation of Free Space

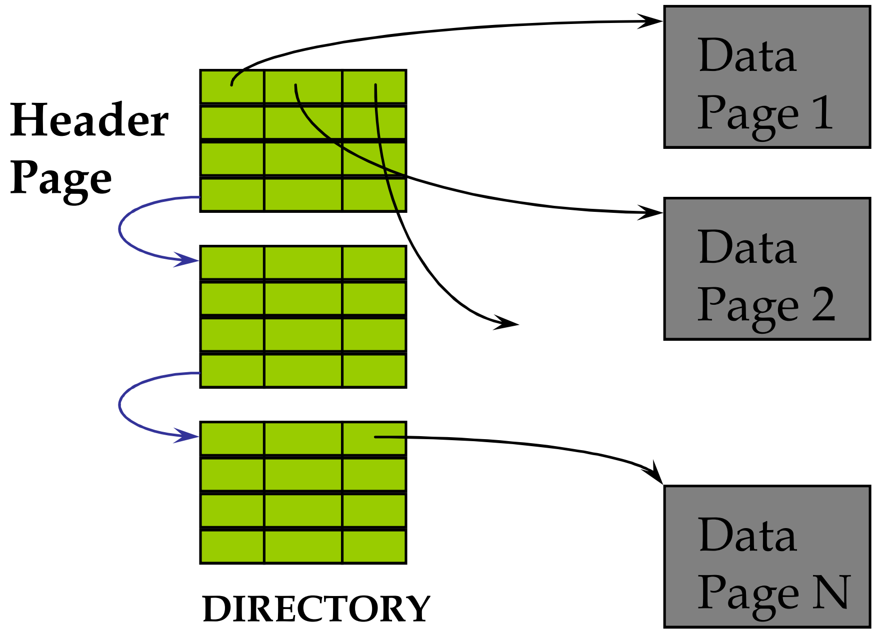
- Granularity
  - pages vs blocks (multiple consecutive pages) vs extents (multiple consecutive blocks)
    - smaller granularity more fragmented
    - larger granularity leads to lower space utilization; good as file grows in size
- Allocation methods
  - contiguous: all pages/blocks/extents are close by
    - may need to reclaim space frequently
  - linked lists: simple but may be fragmented

Managing Space Allocated to Files: Heap (Unordered) File Implemented as a List



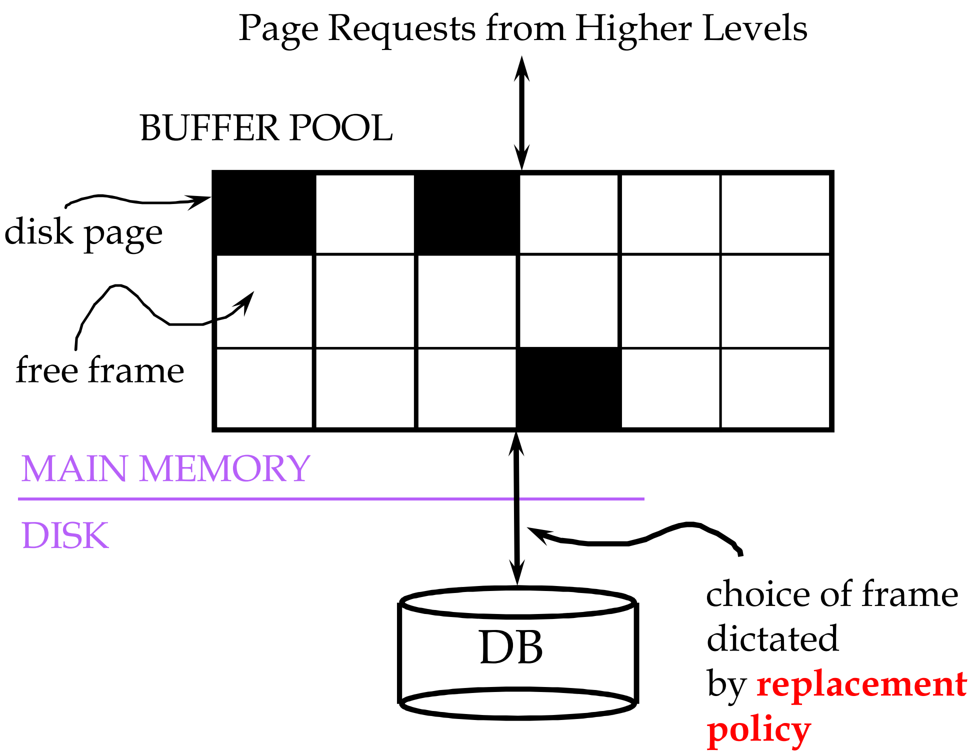
- The header page id and Heap file name must be stored at some place
  - Database "catalog"
- Each page contains 2 pointers plus data

Managing Space Allocated to Files: Heap File Using a Page Directory



- The entry for a page can include the number of free bytes on the page.
- The directory is a collection of pages; linked list implementation is just one alternative
  - *Much smaller than linked list of all HF pages*!

Buffer Management in a DBMS



- *Data must be in RAM for DBMS to operate on it!*
- Buffer pool = main memory allocated for DBMS
- Buffer pool is partitioned into pages called frames
- *Table of <frame#, pageid> pairs is maintained*
- Each frame has two values: pin count and dirty flag

When a Page is Requested ...

- If requested page is not in the buffer pool:
  - If no free frames available
    - Choose a frame for *replacement*
      - *What are such frames?? How to choose?*
    - If frame is *dirty*, write it to disk
  - Read requested page into chosen frame
- *Pin* the page (or increase pin count) and return its address
- What if
  - a page is requested/shared by multiple transactions?
  - no page can be replaced? (when will this happen?)
- Cost to access a page??

If requests can be predicted (e.g., sequential scans)
pages can be pre-fetched several pages at a time!

Replacement Policies

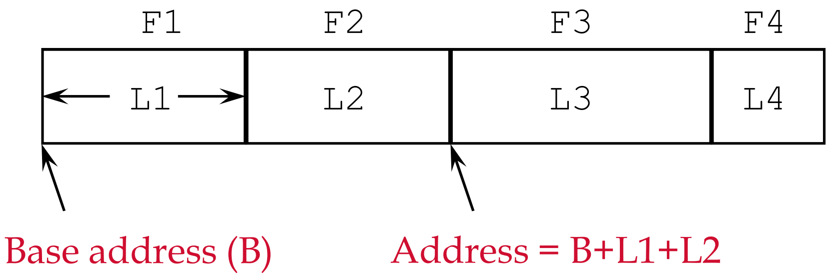
- FIFO: replaces the oldest buffer page (age: first reference)
  - good only for sequential access behavior
- LFU (Least Frequently Used): replaces the buffer page with the lowest reference frequency
  - pages with high reference activity in a short interval may never be replaced!
- LRU (Least Recently Used): replaces the buffer page that is least recently used, i.e., age: last reference
  - worst policy when sequential flooding occurs (MRU is best here!)

Files of Records

- Page or block is OK when doing I/O, but higher levels of DBMS operate on *records*, and *files of records*.
- FILE: A collection of pages, each containing a collection of records. Must support:
  - Create/insert/delete/modify record
  - Read a particular record (specified using *record id*)
  - Scan all records (possibly with some conditions on the records to be retrieved)

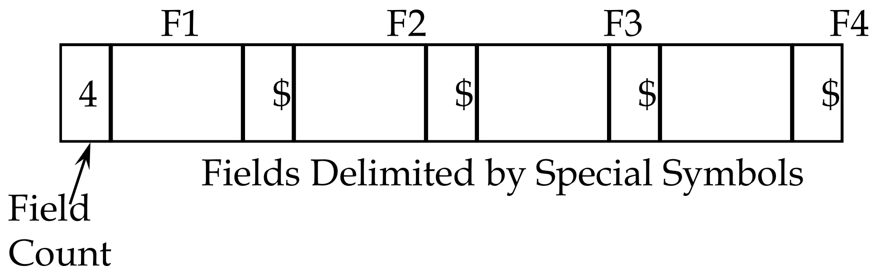
How are records stored? Record Formats

Fixed Length

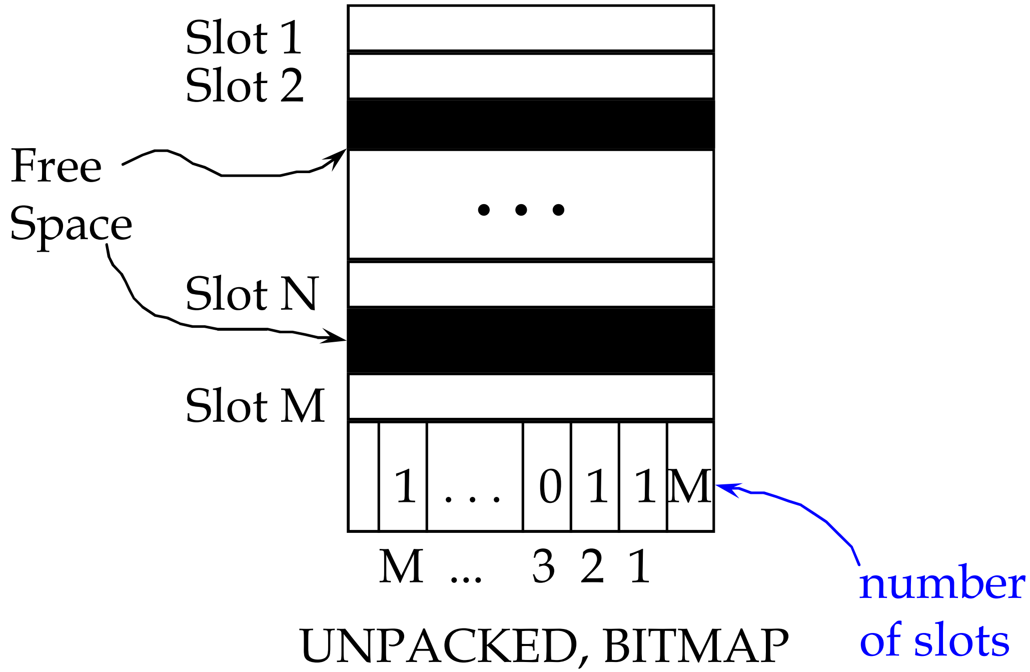


- Information about field types *same* for all records in a file; stored in *system catalogs*

Variable Length:

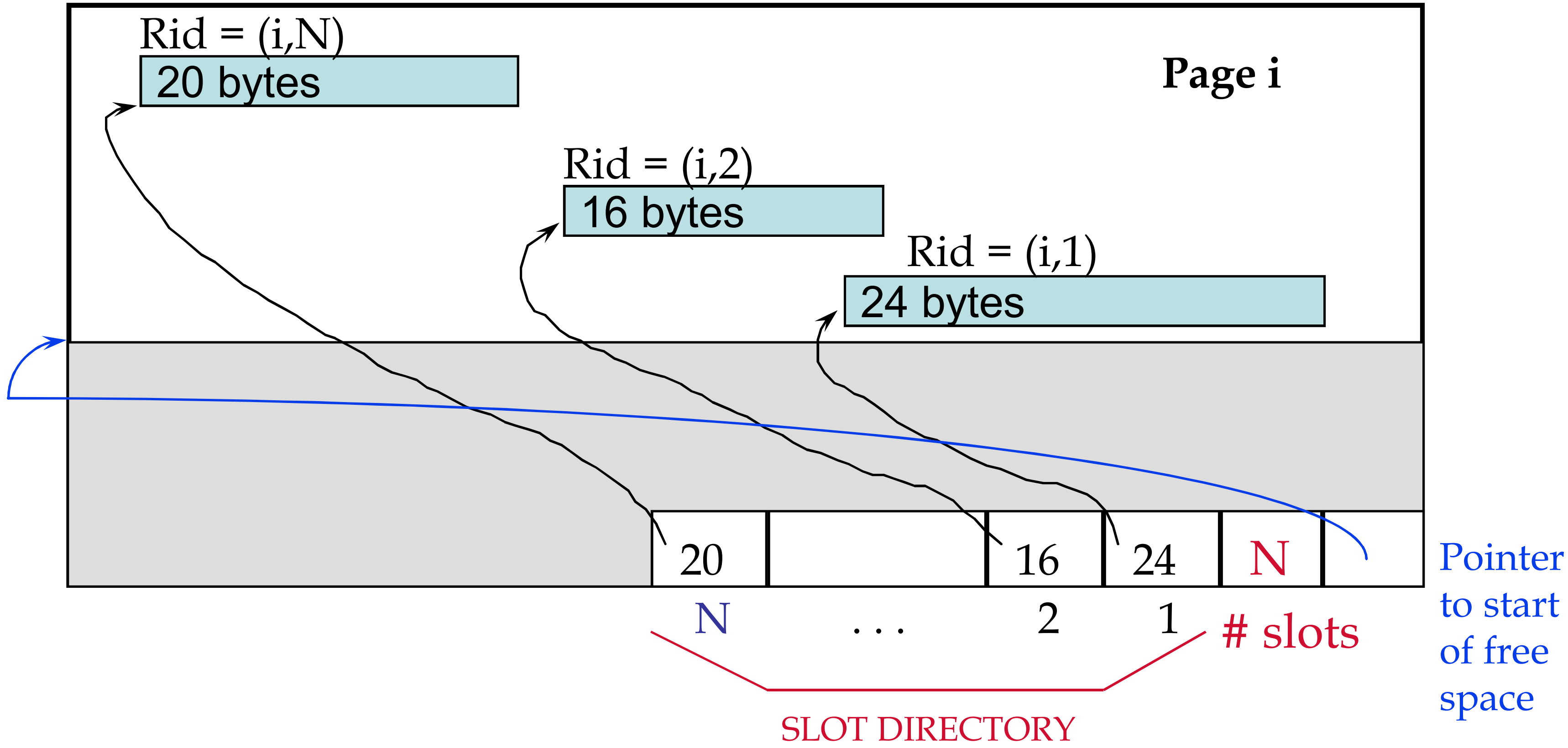


How are pages structured?
Page Formats: Fixed Length Records



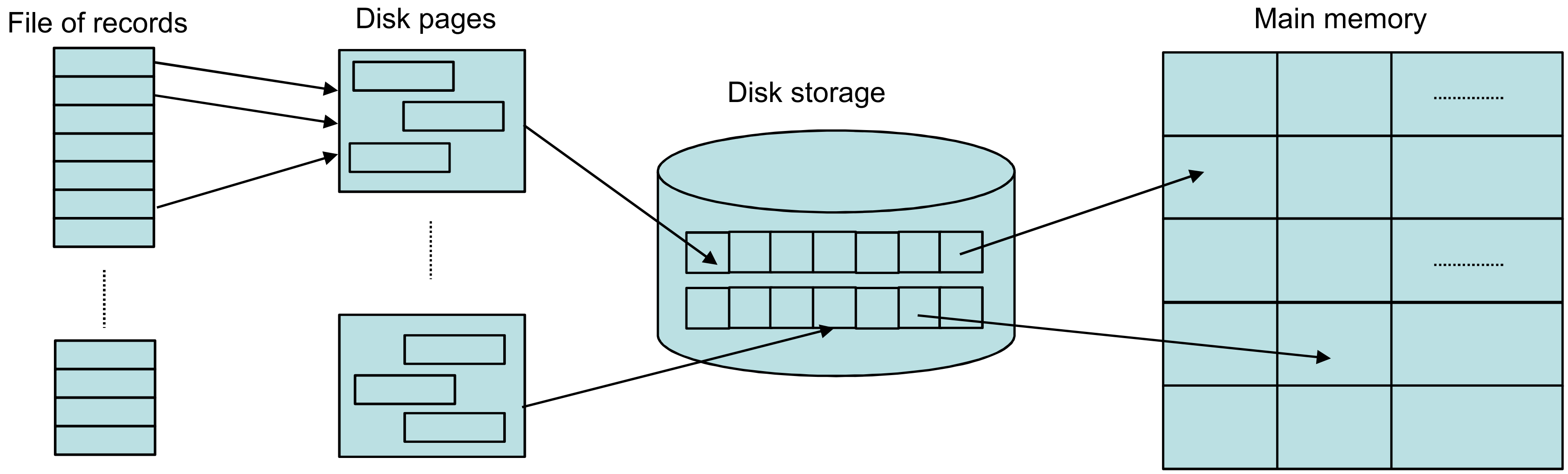
- *<u>Record id</u> = <page id, slot #>. Records within a page cannot be shifted around within the page without changing record id.*

Page Formats: Variable Length Records



- *Can move records in page without changing rid; so, attractive for fixed-length records too.*

Summary



- Disk accesses are expensive operations
- Effective buffer management is crucial to performance
- Buffer management in DBMS vs OS
  - page reference patterns are predictable
  - pages can be pinned, and forced to disk
  - can prefetch multiple pages in advance