# Database Organization: Tables on Disk

Garcia-Molina, Ullman, Widom

Ramakrishnan/Gehrke Ch. 9

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

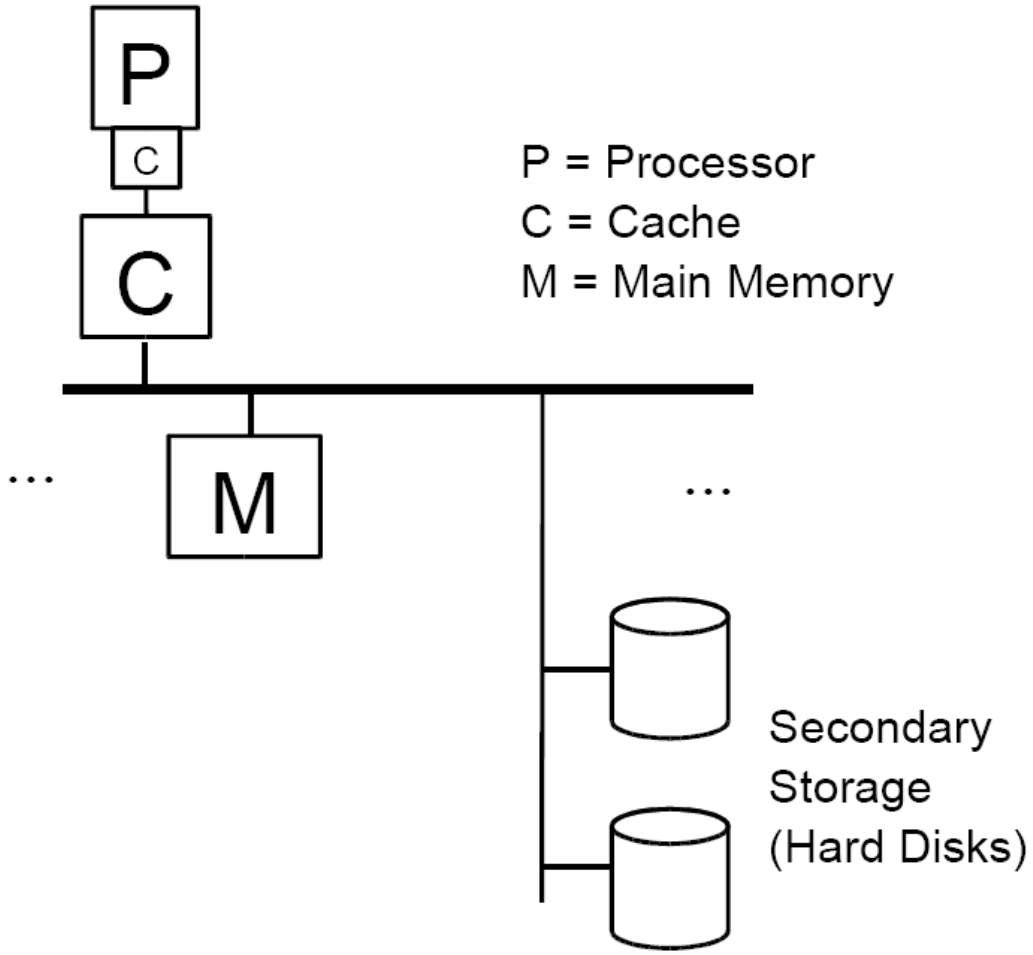
# Roadmap

- Disks
- Files
- Records
- Storage management
- Catalogs
- Summary

# Why Not Everything in Main Memory?

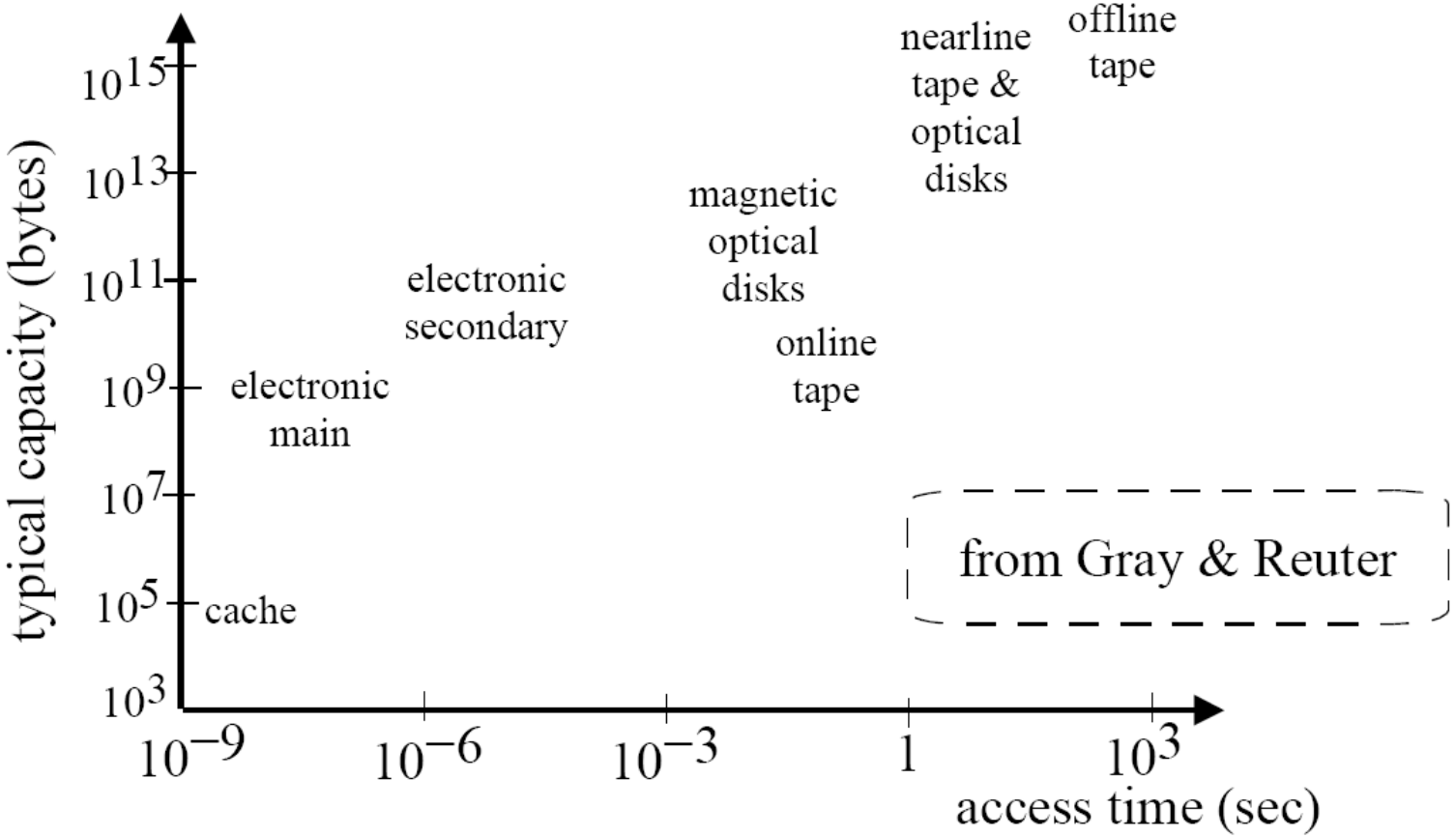
- Costs too much
  - [Rama/Gehrke] $1000 will buy you either 128MB of RAM or 7.5GB of disk
  - Today: 80 EUR will buy you either 4 GB of RAM or 1 TB of disk
  - …but today we have multi-Terabyte databases!
- Main memory is volatile
  - want data to be saved between runs (obviously!)
- Typical storage hierarchy:
  - Main memory (RAM) for currently used data
  - Disk for main database (secondary storage)
  - Tapes for archiving older versions of data (tertiary storage)

# A Typical Computer



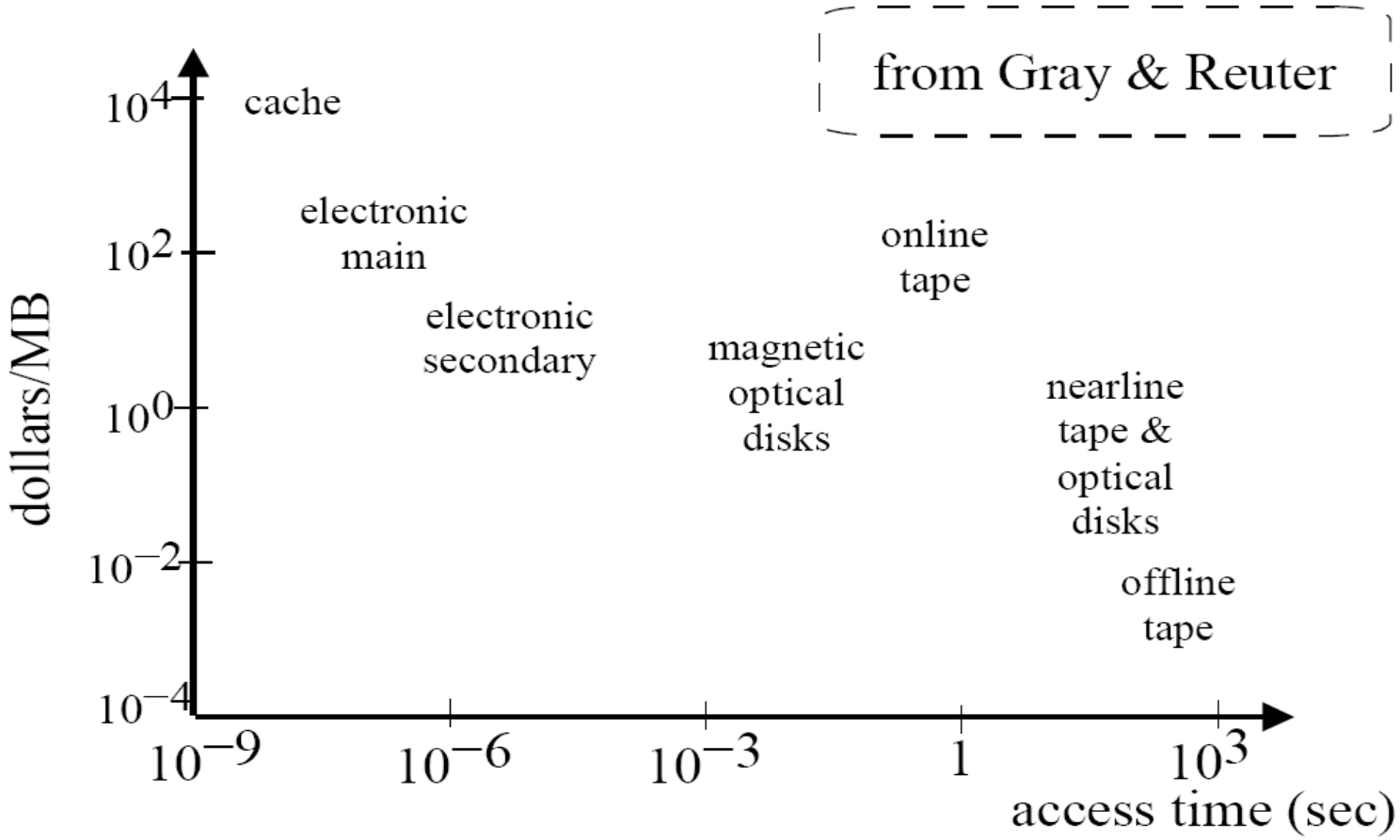
# Storage Capacity

- Absolute times as of 2003, but ratios still ~ same

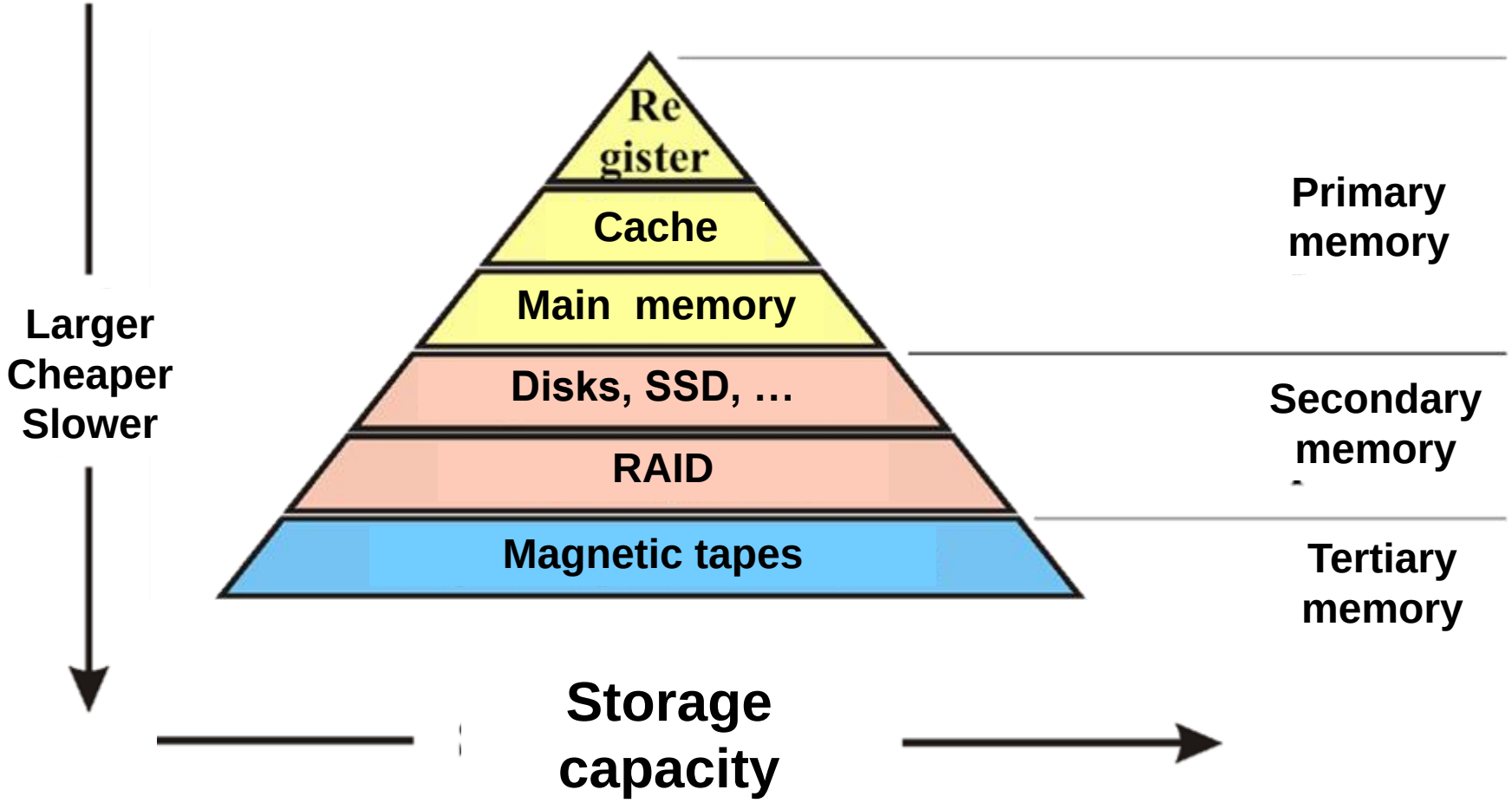


# Storage Cost

- Again, absolute values as of 2003, but ratios still ~ same



# Storage Hierarchies



# Numbers

| | |
|---|---|
| CPU Register | 1 ns |
| Main Memory | 100 ns |
| Flash Drive | 100,000 ns |
| Hard Drive | 10,000,000 ns |

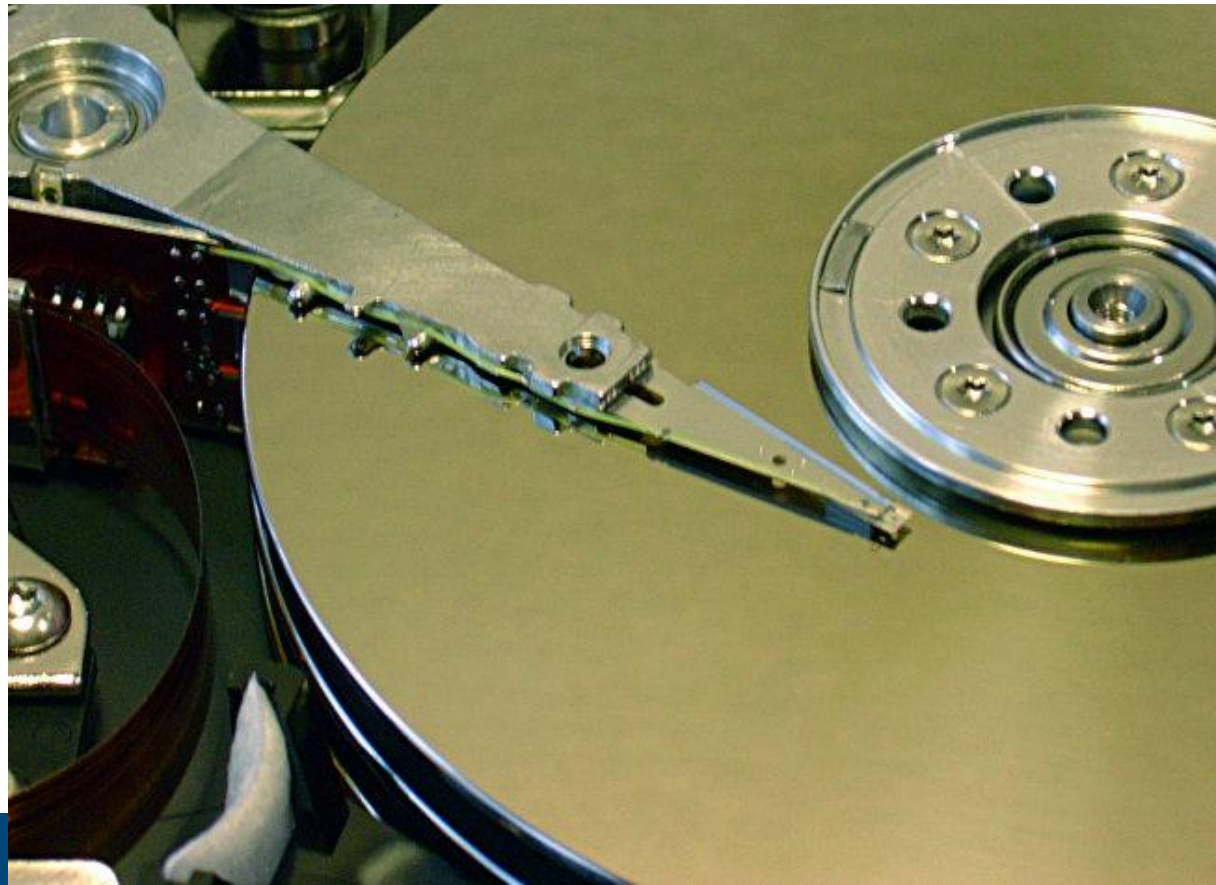
source

# Caching & Virtual Memory

- Cache: Fast memory, holding frequently used parts of a slower, larger memory
  - small (L1) cache holds a few kilobytes of the memory "most recently used" by the processor
  - Most operating systems keep most recently used "pages" of memory in main memory, put the rest on disk
- Virtual memory
  - programs don't know whether accessing main memory or a page on secondary memory page (most operating systems)
- Database systems usually take explicit control over 2ndary memory access

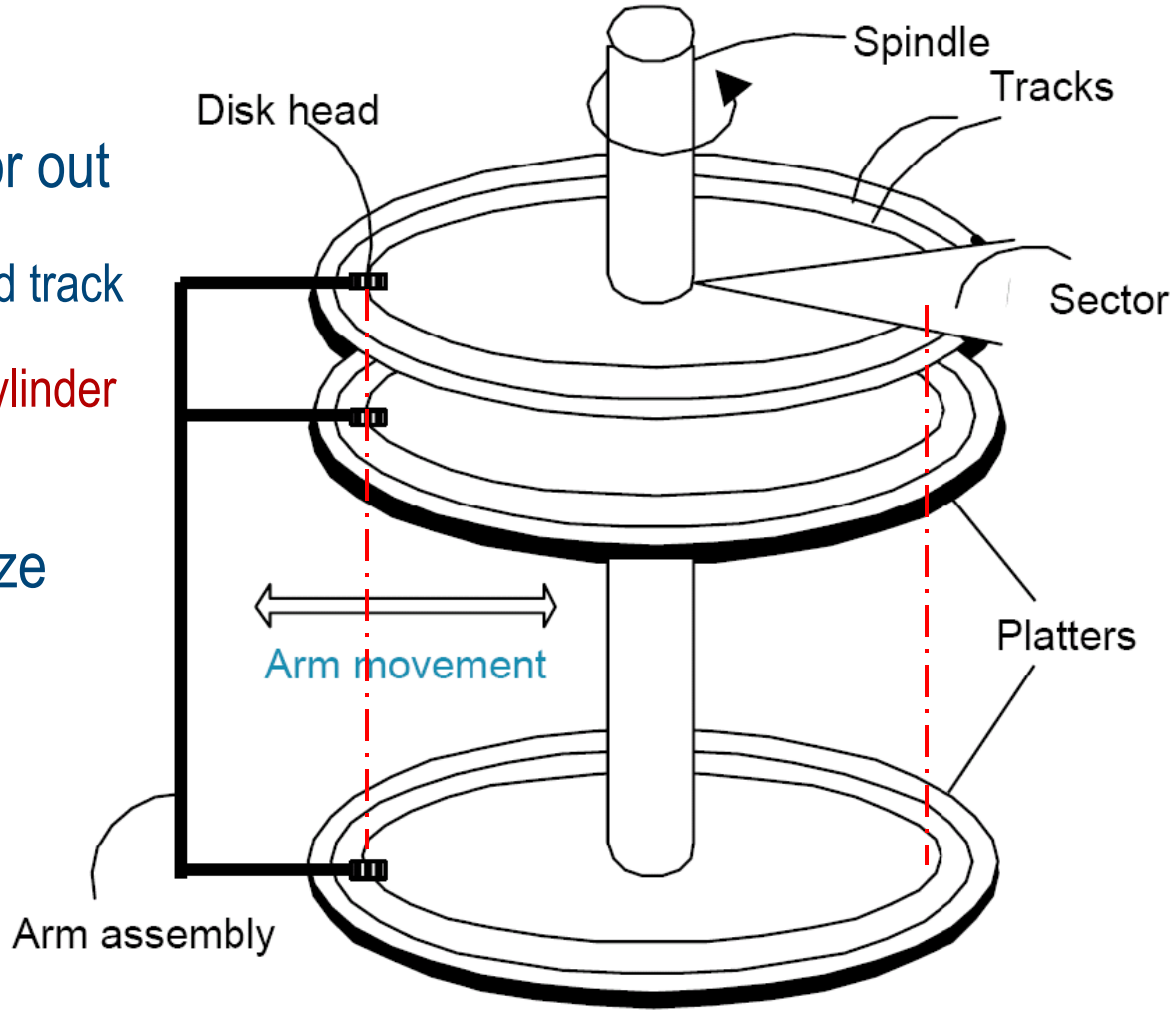
# The Miracle Called "Hard Disk"

- Disk head contains magnet, hovering over spinning platter
  - flight height: 10-20 nm
  - (x 5,000 gives one hair!)
- → relative placement of pages on disk has major impact on DBMS performance!



# Components of a Disk

- platters spin
- arm assembly moves in or out
  - to position head on desired track
  - Tracks under heads = a cylinder (imaginary!)
- Sector size = N * block size (fixed)
- *...typical numbers?*



# Nearline (Tertiary) Storage

- Usually tape
  - Reel, today: cartridge
  - Capacity 10...220 GB per tape
- Tape robots
  - HSM system = Hierarchical storage management system
  - Capacity several Petabytes

# Roadmap

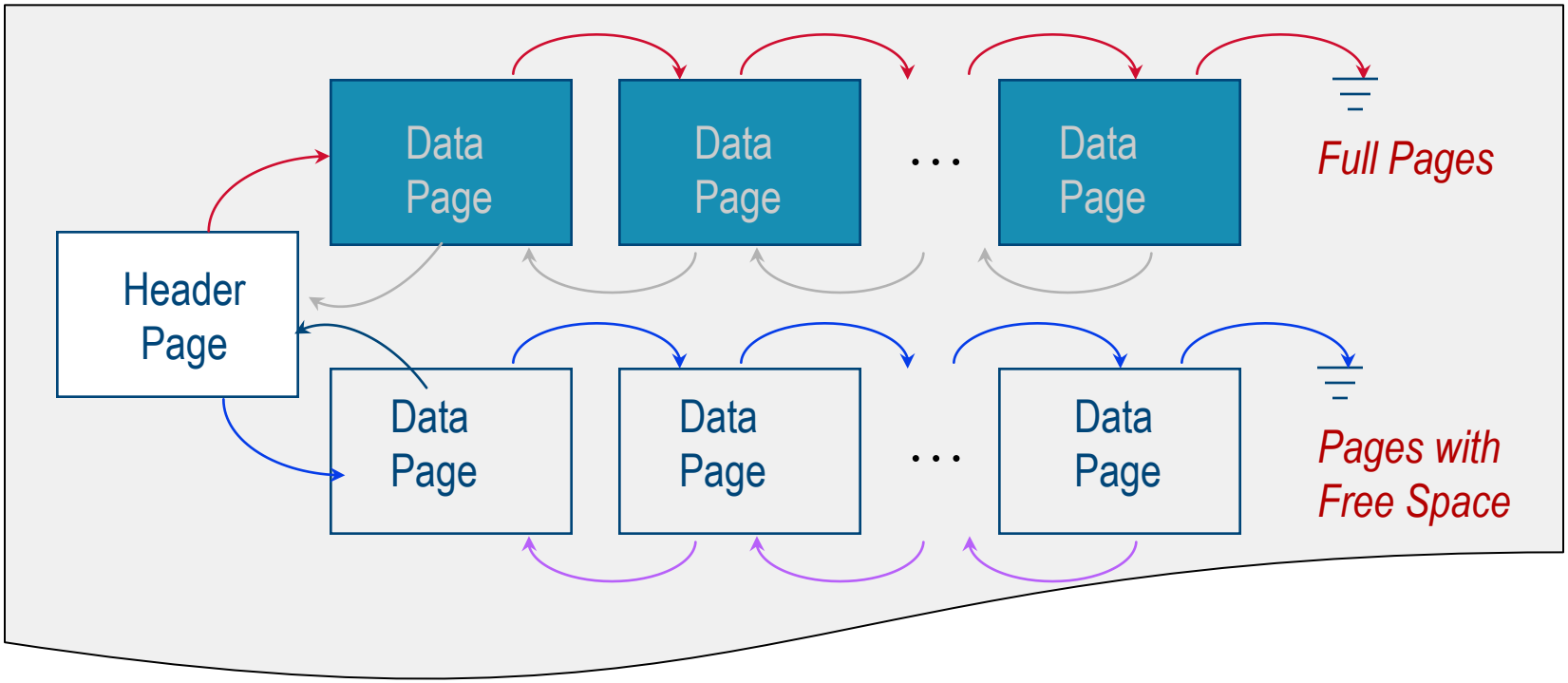
- Disks
- Files
- Records
- Storage management
- Catalogs
- Summary

# Table of Tuples → Files of Records

- File = collection of pages, each containing a collection of records
- Operations:
  - find record by id
  - scan through all records (possibly with predicate)
  - insert/delete/modify record
- I/O operates on pages / blocks  higher levels operate on records
- Heap File: Simplest file structure, records in no particular order
  - many implementation alternatives

# Heap File Implemented as a List

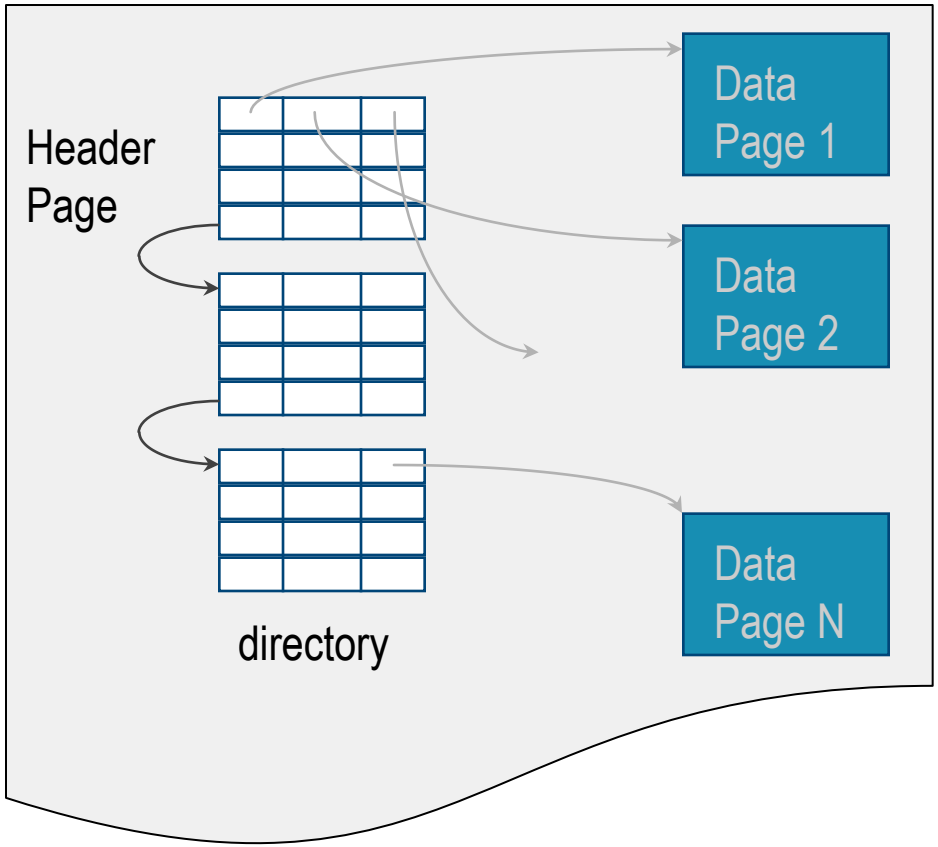
- /data/.../students.dbf:




- Heap File name stored in system catalog ↳
- page = 2 `pointers' + records

# Heap File Using a Page Directory

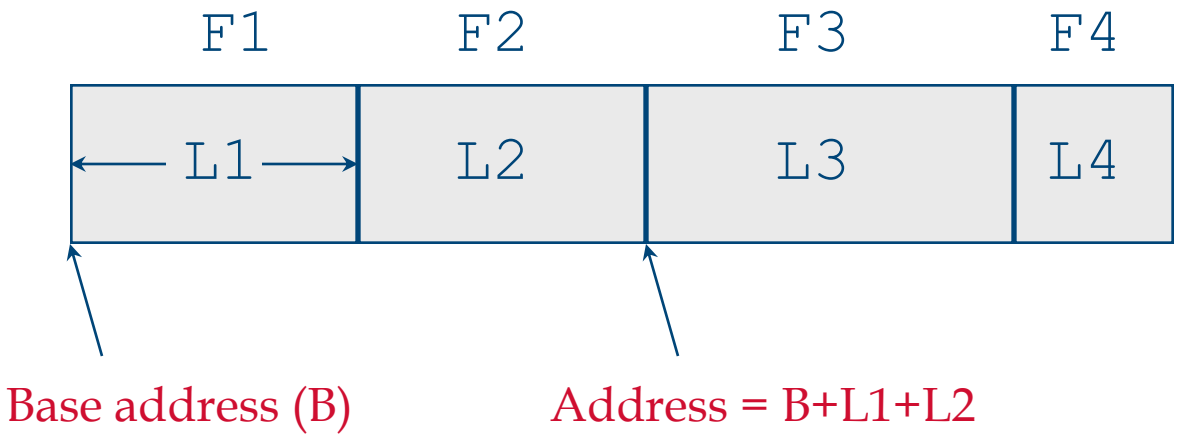
- directory = collection of pages
  - linked list, hash table, …
- Optimization:
  add # free bytes on page



# Roadmap

- Disks
- Files
- Records
- Storage management
- Catalogs
- Summary

# Record Formats: Fixed Length

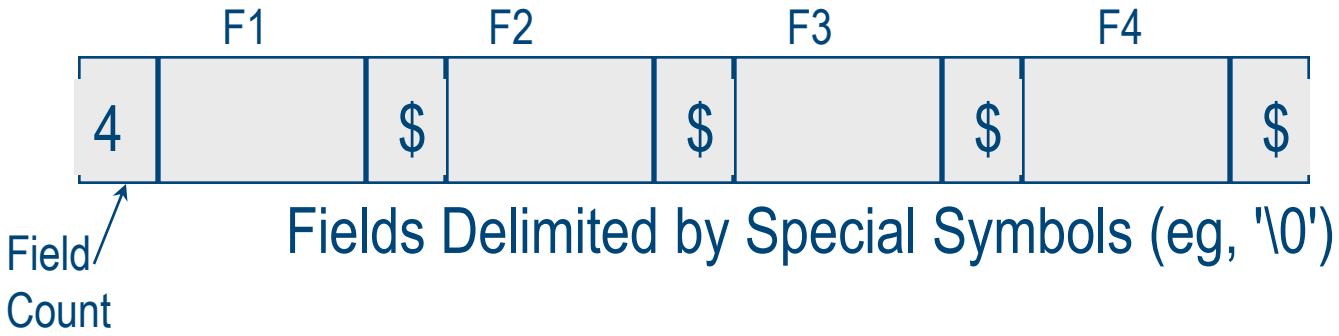



- Information about field types same for all records in a file; stored in *system catalogs*
- Finding *i'th* field does not require scan of record

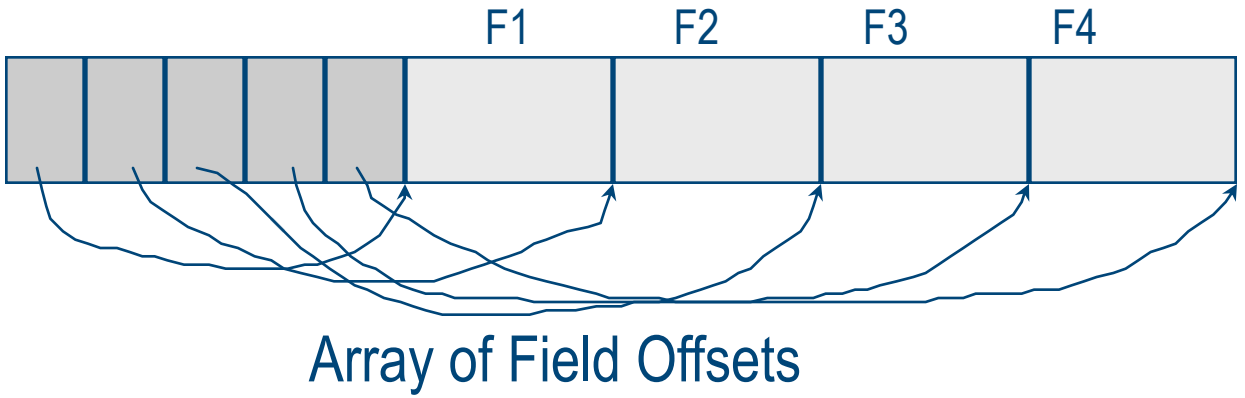
# Record Formats: Variable Length

- Two alternative formats (# fields fixed):

Variant 1: delimiters

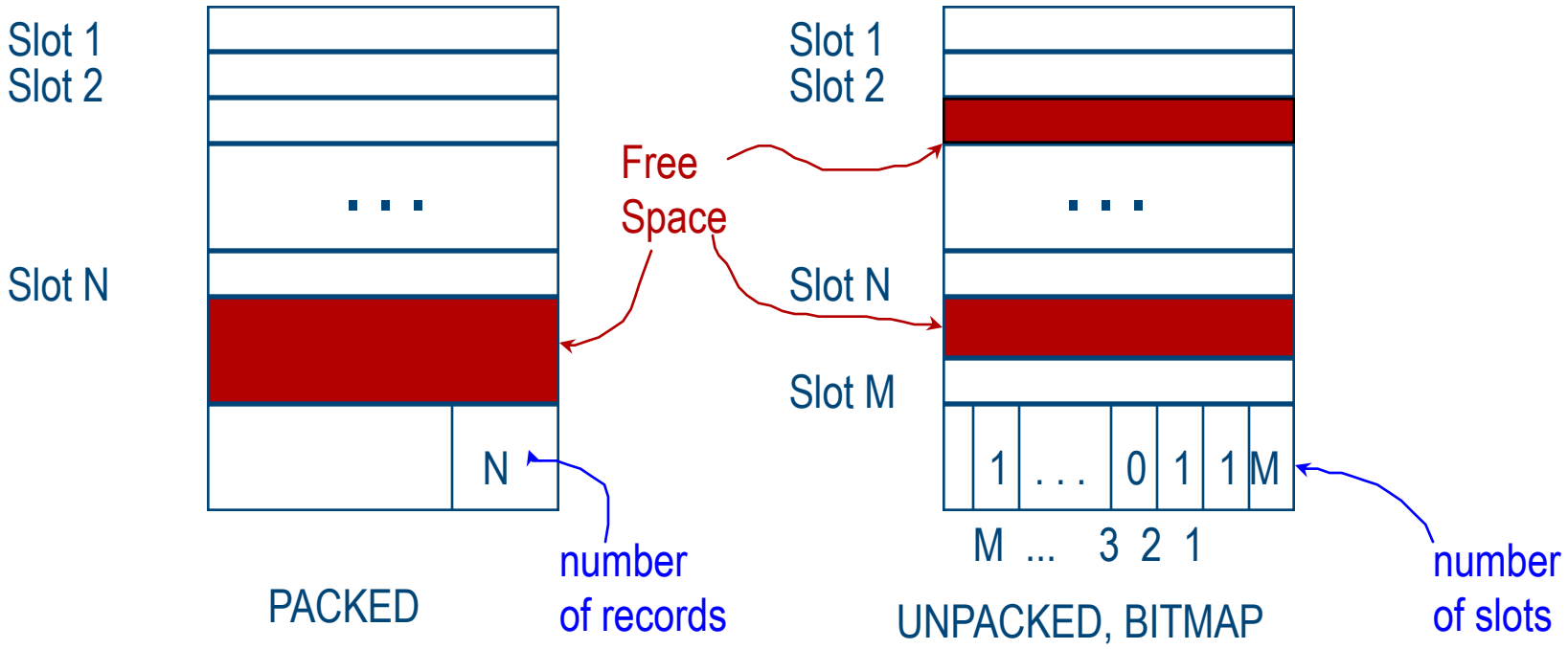



Variant 2: offset ptrs



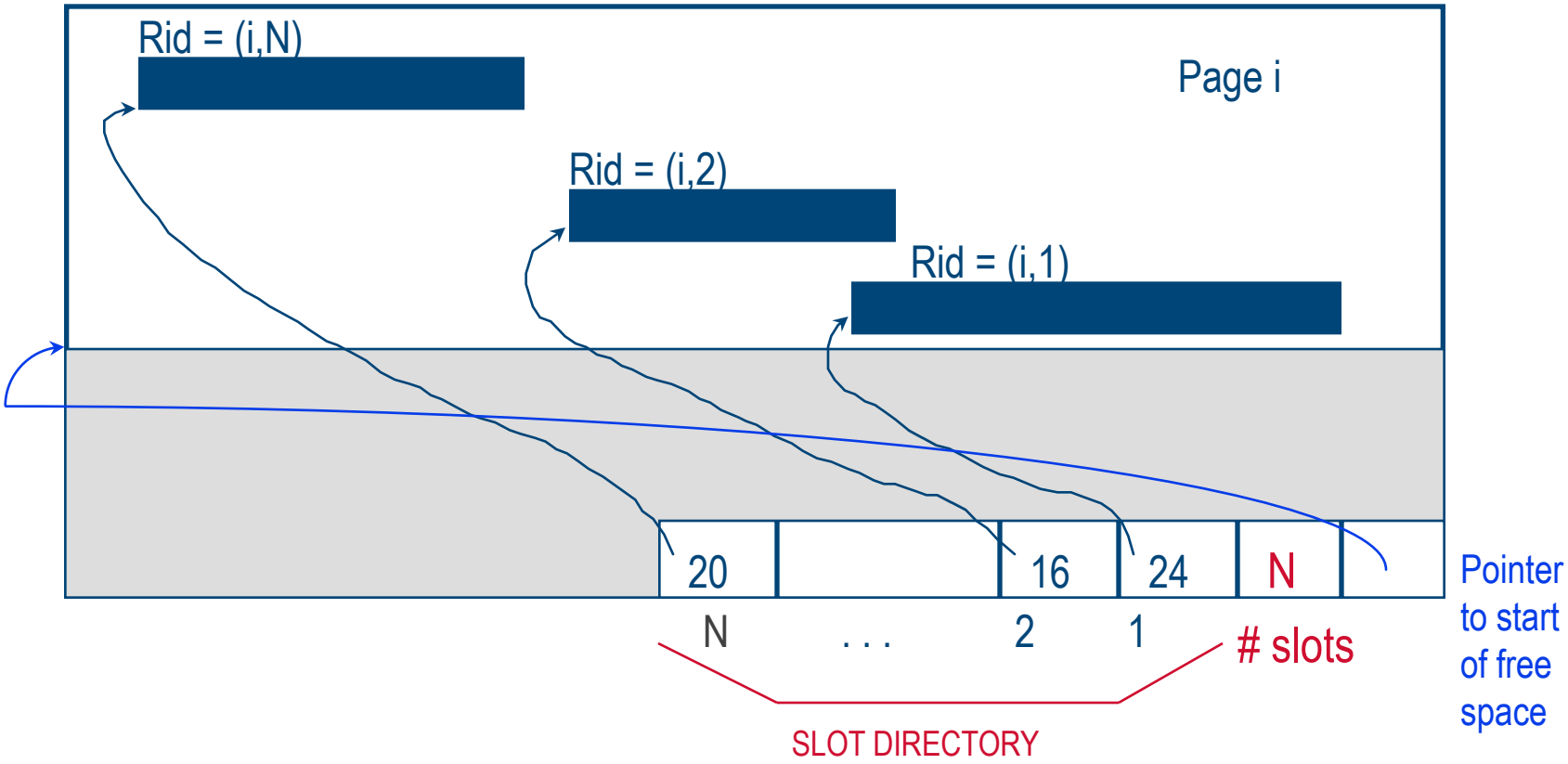

- Var2: direct access to i'th field; efficient storage of *nulls*; small directory overhead

# Page Formats: Fixed Length Records




- Record id = <page id, slot #>
- In first alternative, moving records for free space mgmnt changes rid
  - may not be acceptable

# Page Formats: Variable Length Records




Indirection → can move records on page without changing rid

→ attractive for fixed-length records too

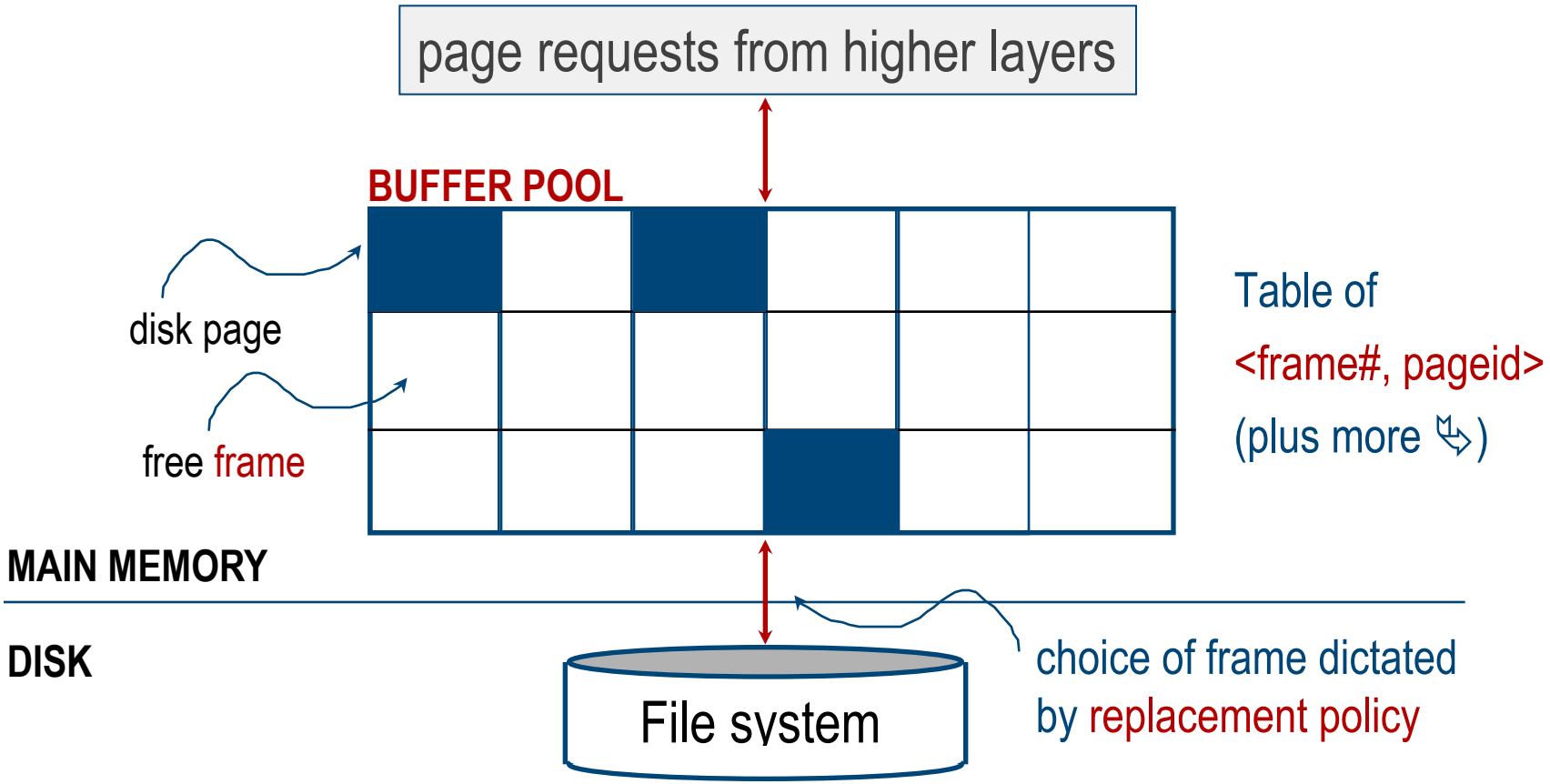
# Roadmap

- Disks
- Files
- Records
- Storage management
- Catalogs
- Summary

# Disk Space Management

- Persistent Storage Manager
  - Lowest layer of DBMS software, manages space on disk
  - Higher levels call this to allocate/de-allocate page, read/write page, …
- Optimization: allocate sequence of pages (table!) sequentially on disk!
  - Allocate *n* pages at once
- DBMS maintains RAM cache: Buffer manager ↳

# DBMS Buffer Management



# When a Page is Requested ...

- If requested page is not in pool:
  - Choose a frame for replacement
  - If frame is dirty, write it to disk
  - Read requested page into chosen frame
- Pin page and return its address

- *If possible, arrange blocks sequentially on disk*
  - *minimize seek and rotational delay*
- *For sequential scan (access predictable!), pre-fetching is a big win*

NB:
'page' ≈'block'

# More on Buffer Management

- Page requestor must unpin it
  & indicate whether page has been modified
  - *dirty* bit
- Page in pool may be requested many times
  - *pin count:* page is candidate for replacement iff *pin count* == 0
- CC & recovery: additional I/O when replacing frame
  - *Write-Ahead Log* protocol

# Buffer Replacement Policy

- Frame is chosen for replacement by a *replacement policy:*
  - Least-recently-used (LRU), Clock, MRU etc.
- Policy can have big impact on # of I/O's; depends on the *access pattern*
- *Sequential flooding*:
  Nasty situation caused by 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, of course)

# Roadmap

- Disks
- Files
- Records
- Storage management
- Catalogs
- Summary

# System Catalogs

- For each relation:
  - name, file name, file structure (e.g., Heap file)
  - attribute name and type, for each attribute
  - index name, for each index
  - integrity constraints
- For each index:
  - structure (e.g., B+ tree) and search key fields
- For each view:
  - view name and definition
- Plus statistics, authorization, buffer pool size, etc.

*Catalogs themselves stored as relations!*

# Sample Catalog Table

**Attribute_Cat:**

| attr_name | rel_name | type | position |
|---|---|---|---|
| attr_name | Attribute_Cat | string | 1 |
| rel_name | Attribute_Cat | string | 2 |
| type | Attribute_Cat | string | 3 |
| position | Attribute_Cat | integer | 4 |
| sid | Students | string | 1 |
| name | Students | string | 2 |
| login | Students | string | 3 |
| age | Students | integer | 4 |
| gpa | Students | real | 5 |
| fid | Faculty | string | 1 |
| fname | Faculty | string | 2 |
| sal | Faculty | real | 3 |

*1st entry?*
*Key(s)?*

# Roadmap

- Disks
- Files
- Records
- Storage management
- Catalogs
- Summary

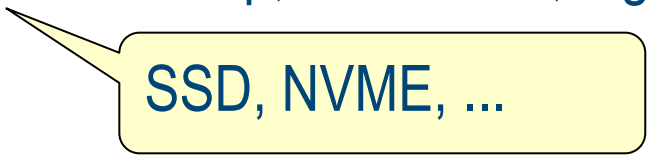
# DBMS vs. OS File System

OS does disk space & buffer mgmt:
why not let OS manage these tasks?

- Differences in OS support: portability issues
- Some limitations
  - e.g., files can't span disks
- Buffer management in DBMS requires ability to:
  - pin page in buffer pool, force page to disk (CC & recovery!)
  - adjust replacement policy + pre-fetch pages
    based on access patterns in typical DB operations

# Databases & Disk: Practitioner's Hints

- Choose file system wisely: ext3, ext4, xfs, ...
  - Many discussions
- Place redo logs (and Oracle control files) on separate partitions
  - If possible, same for index (higher traffic!)
- Fastest disks for /tmp, cache files, log, and other high-traffic dirs




- Big RAM is never wrong
- Still money left? RAID!

# Summary

- Disks provide cheap, non-volatile storage
  - Random access, but cost depends on location of page on disk
  - important to arrange data sequentially to minimize seek and rotation delays
- Buffer manager brings pages into RAM
  - Page stays in RAM until released by requestor
  - Written to disk when frame chosen for replacement (which is sometime after requestor releases the page)
  - Choice of frame to replace based on replacement policy
  - Tries to pre-fetch several pages at a time

# Summary (Contd.)

- DBMS vs. OS File Support: DBMS needs features not found in many OS's
  - forcing page to disk
  - controlling order of page writes to disk
  - files spanning disks
  - control pre-fetching + page replacement policy based on predictable access patterns, etc.
- Variable length record format with field offset directory
  - supports direct access to i'th field, null values
- Slotted page format
  - supports variable length records, allows records to move on page

# Summary (Contd.)

- File layer
  - keeps track of pages in file + supports abstraction of "collection of records"
  - Pages with free space identified via linked list or directory structure
  - similar to how pages in file are kept track of
- Indexes
  - support efficient retrieval of records based on the values in some fields
- Catalog relations
  - store information about relations, indexes and views
  - Information that is common to all records in a given collection