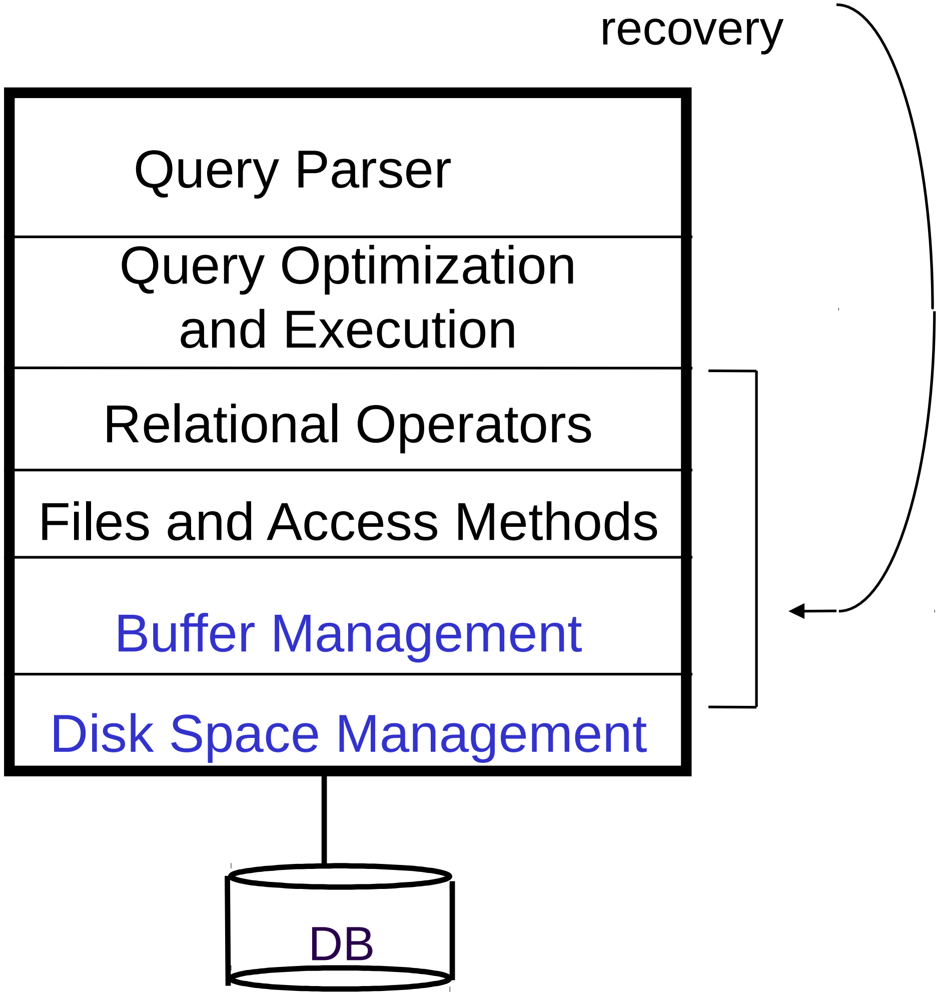
Structure of a DBMS

- A typical DBMS has a layered architecture
- The figure does not show the concurrency control and recovery components
- This is one of several possible architectures; each system has its own variations



These layers must consider concurrency control and recovery

Query Parser

Query Optimization and Execution

Relational Operators

Files and Access Methods

Buffer Management

Disk Space Management

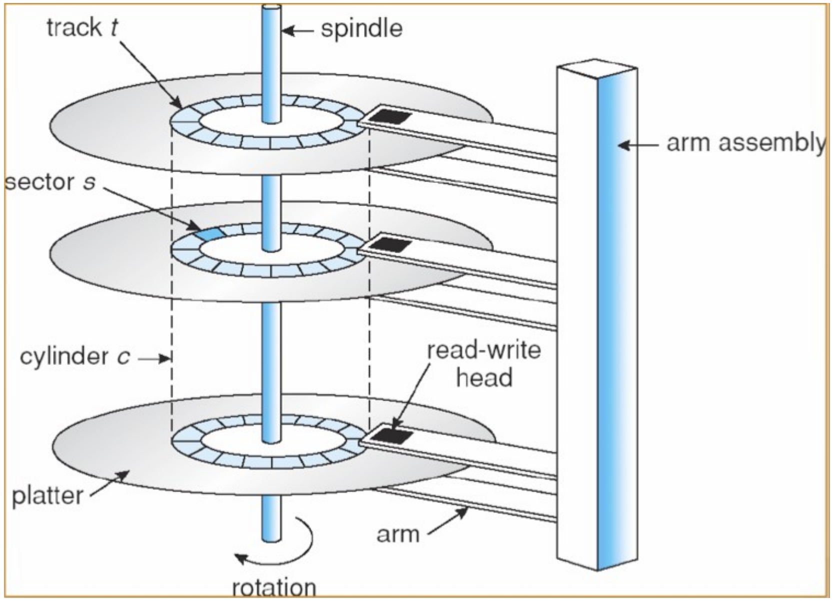
DB

6-layer model

Data Storage

- DBMS deals with a very large amount of data
  - How does a computer system store and manage very large volumes of data?
  - What representations and data structures best support efficient manipulations of this data?

Storage Hierarchy



- cylinder:track:sector
- about 200 times slower than DRAM
- Access time
  - seek time
  - transfer time

CPU

CACHE KB/MB

MAIN MEMORY MB/GB

HARD DISK (RAID) GB/TB

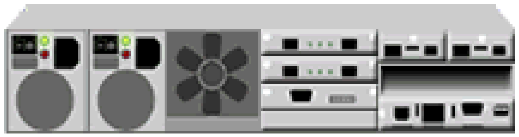
OPTICAL DISK GB/TB

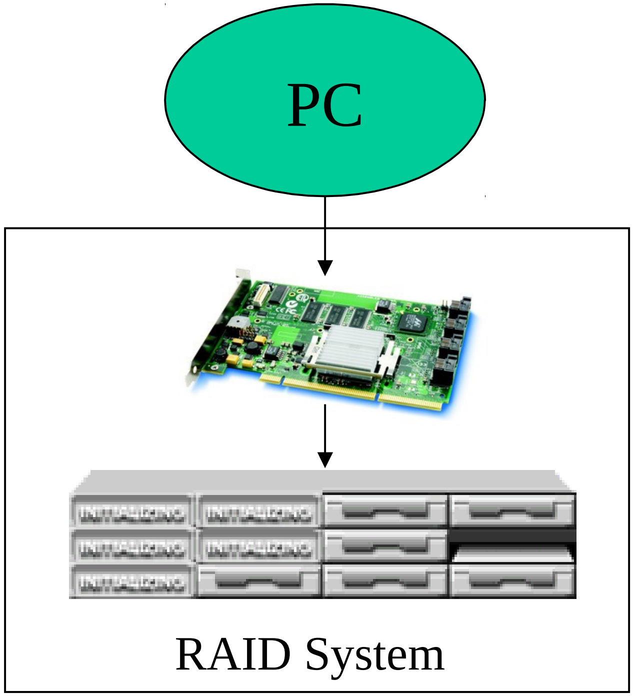
TAPE TB

Redundant Array Inexpensive Disk 1987

Redundant Array of Independent Disks

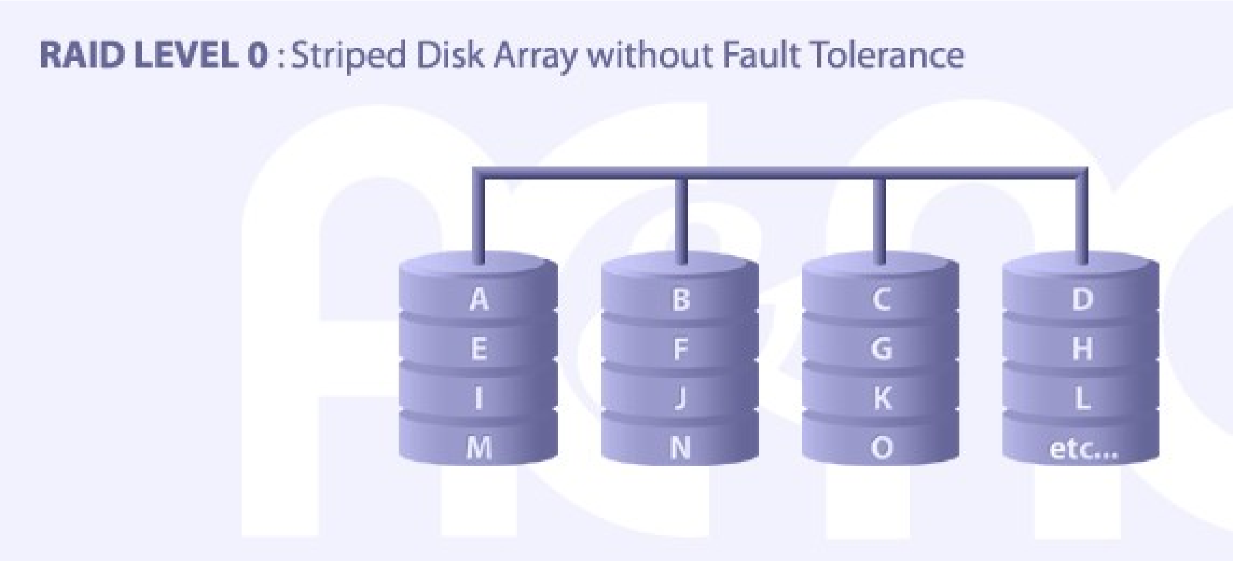
- RAID: A number of disks is organized and appears to be a single one to the OS
    - Aggregate disk capacity
    - Increase I/O throughput
    - Fault-tolerant (hot swapping)
- Some well-known configurations
    - RAID 0: striping, no mirroring or parity
        - improve throughput
    - RAID 1: mirroring
        - Improve throughput and ensure redundancy
    - RAID 2: bit-level striping with parity coding
        - Fault-tolerate with good storage efficiency
    - RAID 5: block-level striping with distributed parity coding





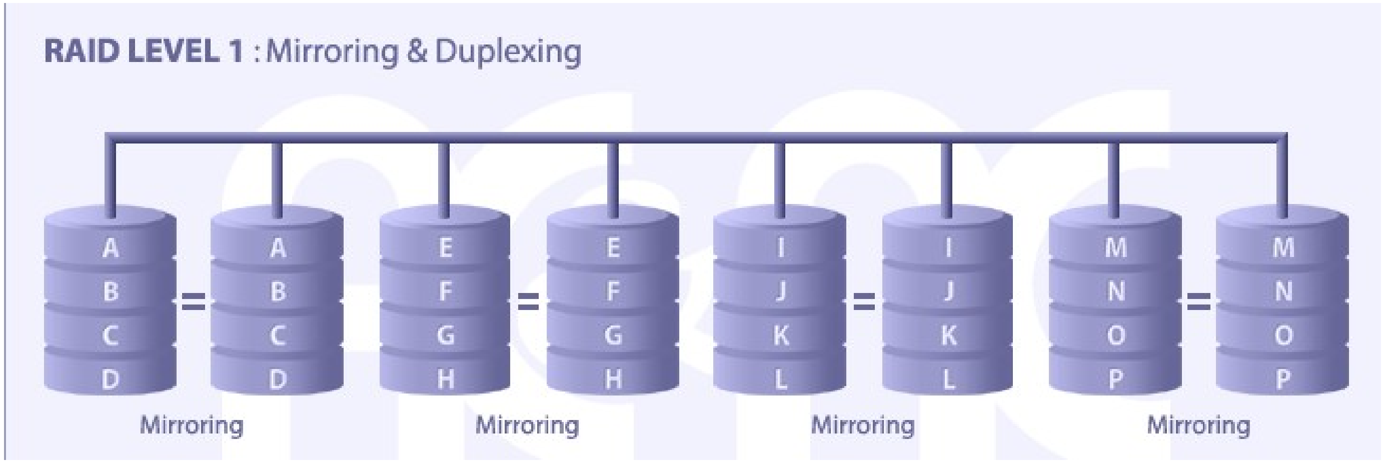
https://www.youtube.com/watch?v=Aa0RTgxJJy8

RAID - Level 0



- RAID 0 implements a striped disk array, the data is broken down into blocks and each block is written to a separate disk drive
  - PRO: I/O performance is greatly improved by spreading the I/O load across many channels and drives
  - CON: NOT fault-tolerant
  - Applications: video editing, etc. (requiring high speed)

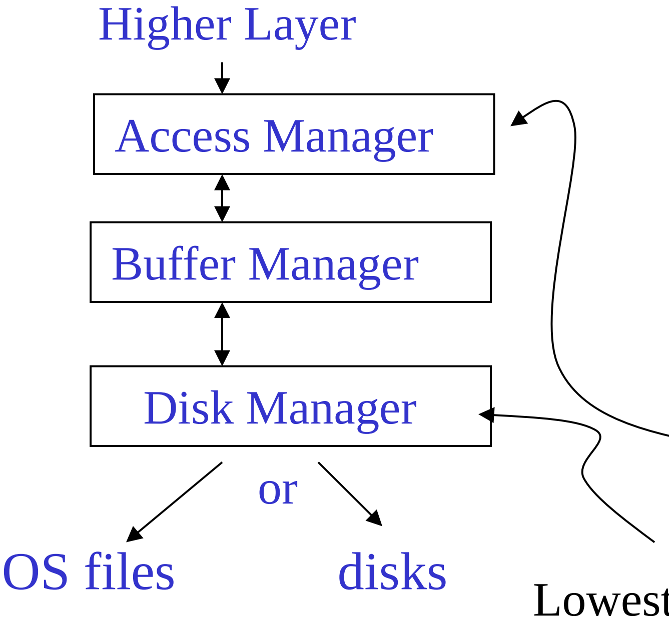
RAID - Level 1



- RAID 1: Each data item is duplicated on two disks (mirror)
  - PRO: 1W/2R possible per mirrored pair, sustain simultaneous disk failures
  - CON: inefficient -- 100% overhead of disk space
  - Applications: accounting, etc. (mission critical)

Storing databases/tables/records

- Each database contains a number of tables
- Each table contains a number of records
- Each record has a unique identifier called a *record id*, or *rid*
    - Records can be stored in files based on the underlying OS
    - Records can be stored directly to disk blocks bypassing file systems (raw devices)



Higher Layer

Access Manager

Buffer Manager

Disk Manager

or

OS files

disks

Why raw devices?

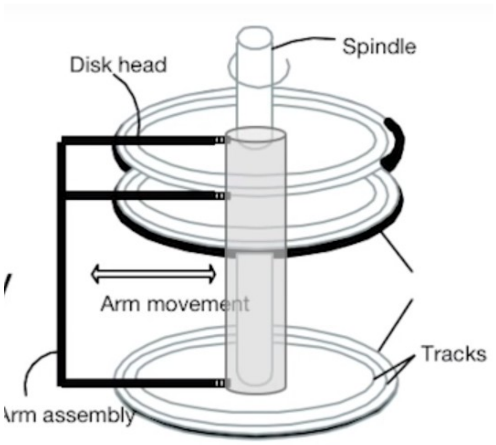
- Differences in OS support: portability issues
- Some limitations, e.g., files can’t span disks.
- Performance

Arrange records in a page, the size of which is usually 4k or 8k (old number, nowadays, usually 64k -128k)

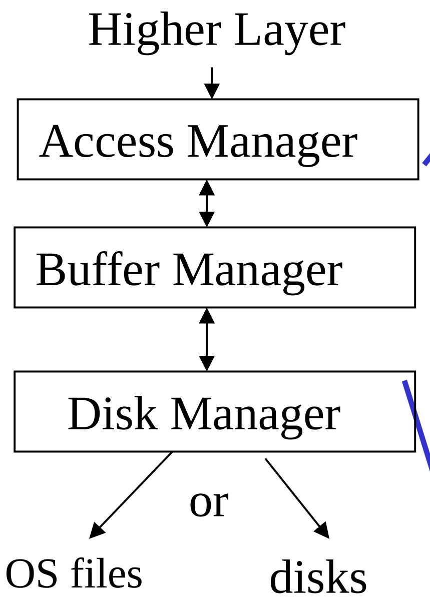
Lowest level in DBMS software architecture

A note of terminology

- Block/Page
  - Unit of transfer for disk read/write
  - I/O is costly
    - Access time = seek time + rotational delay + transfer time
  - Like a backpack



An Overview at 30,000-foot High

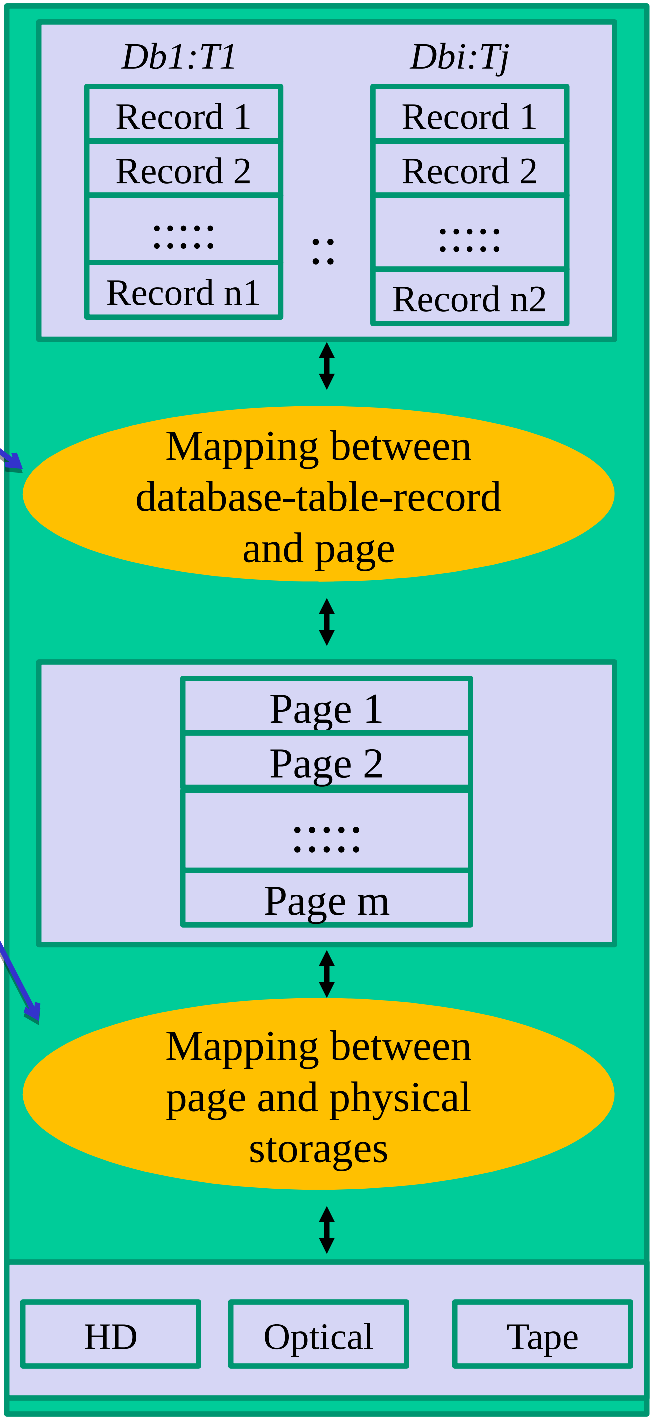


record-level operations

CreateTable(DB, tName, field[])
ReadRecord(DB, tName, rID)
WriteRecord(DB, tName, rID, r)
DeleteRecord(DB, tName, rID)
Append(DB, tName, record)

page-level operations

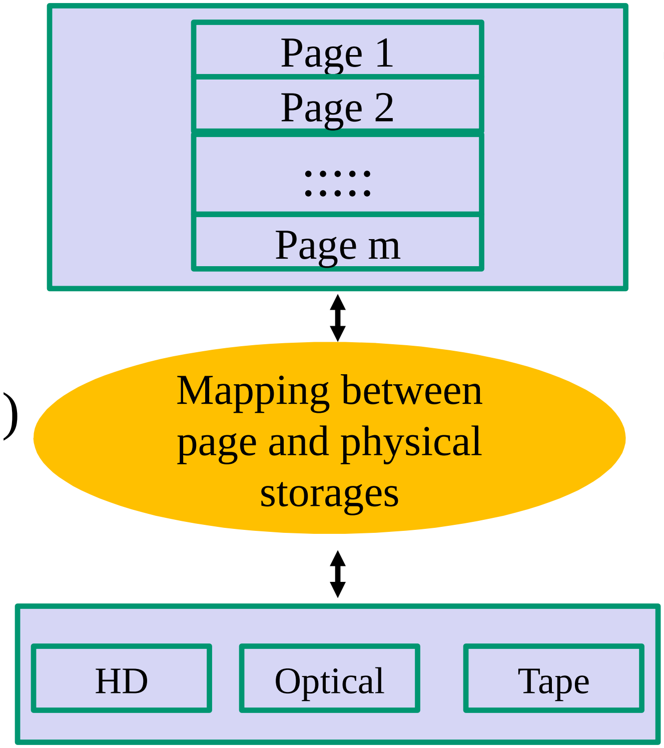
AllocatePage()
DeletePage(pID)
WritePage(pID, page)
ReadPage(pID)



1. A programmer sees storage at record level, he can create/modify/delete a record
2. Access Manager sees storage at page level, it can allocate/read/write/deallocate a page
3. Disk Manager sees storage at device level (e.g., disk/cylinder/sector)

Disk Manager

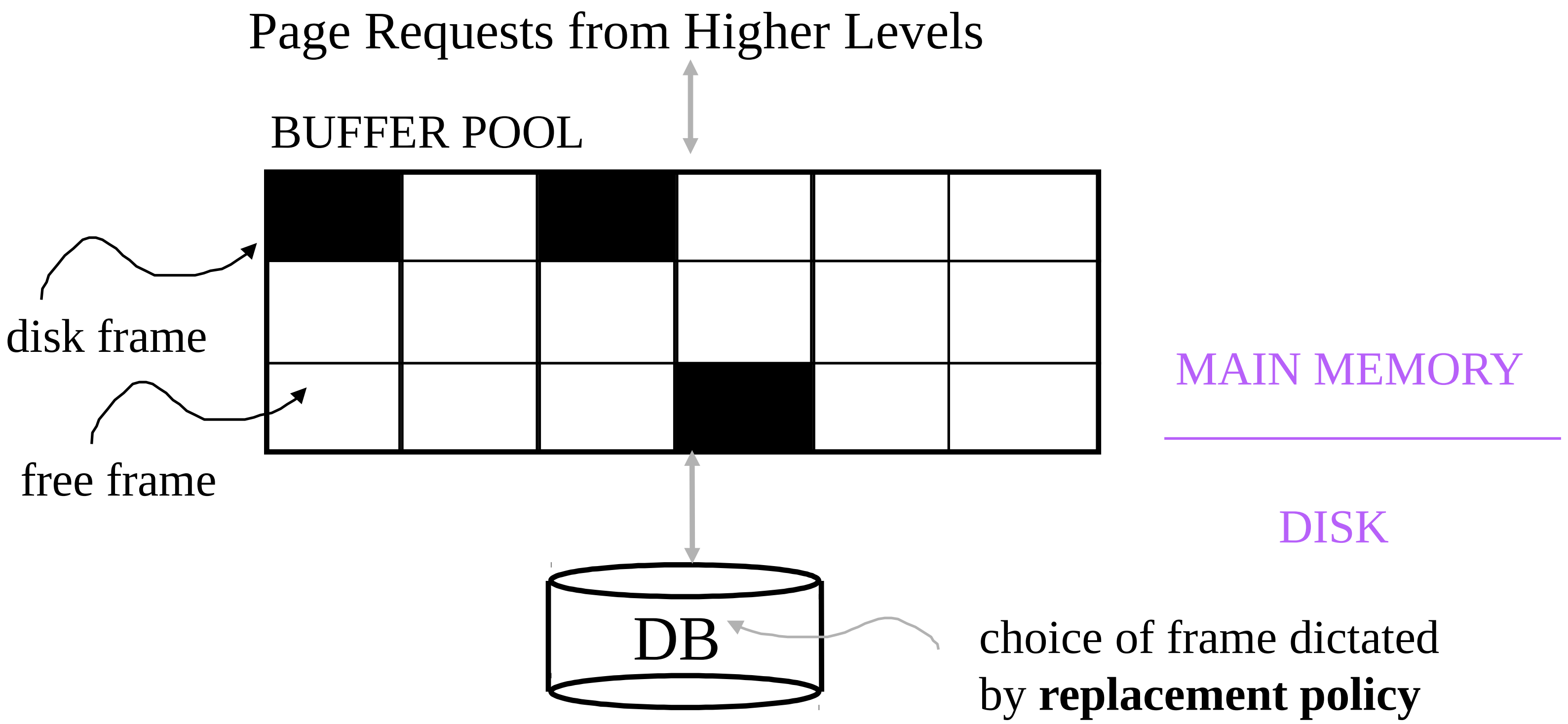
- Lowest layer of DBMS, manages space on disk
  - Mapping between page id and disk id
  - Managing pages (used or free)
  - Higher levels call upon this layer to:
    - allocate/de-allocate a page on disk
    - read/write a page (or a block or a unit of disk retrieval)



These pages are physically located in different devices (RAID, optical, and/or tape)

- AllocatePage()
- DeletePage(pID)
- WritePage(pID, page)
- ReadPage(pID)

Buffer Manager


Page Requests from Higher Levels
BUFFER POOL
disk frame
free frame
MAIN MEMORY
DISK
DB
choice of frame dictated by **replacement policy**

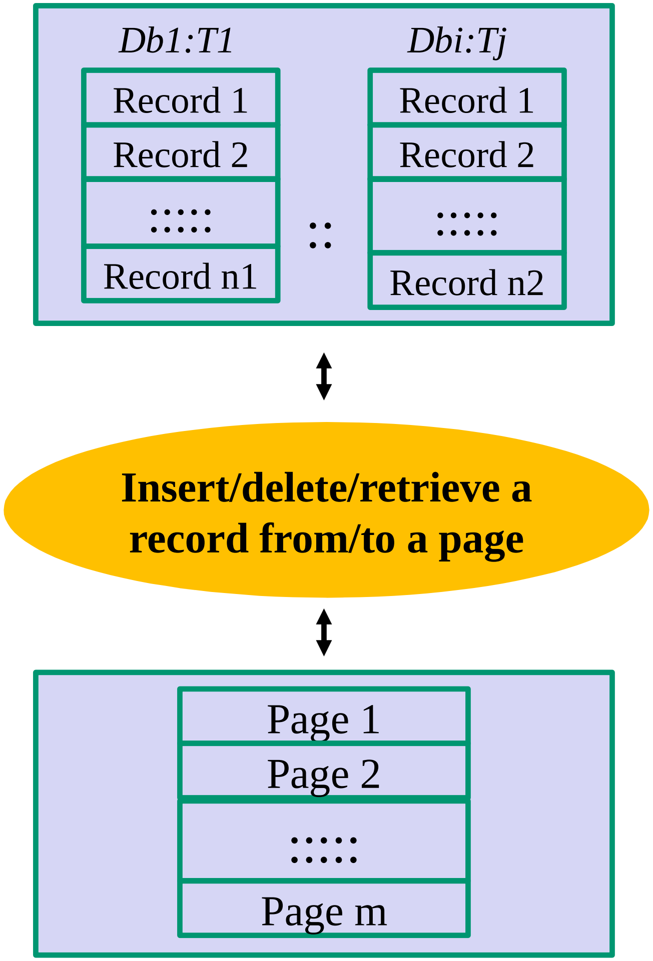
- Size of a frame equal to size of a disk page
- Two variables associated with each frame/page:
  - Pin_count: number of current users of the page
  - Dirty: whether the page has been modified since it has been brought into the buffer pool

When a page is requested

- If a requested page is not in the buffer pool:
  - Choose a frame for replacement if the buffer is full
  - If the frame is dirty, write it to disk
  - Read the requested page into the chosen frame
- *Pin* the page and return its address to the requester
  - Pinning: Incrementing a pin_count
  - Unpinning: Release the page and the pin_count is decremented
- Page is chosen for replacement by a replacement policy
  - Least-recently-used (LRU): based on time of last reference
  - First-In-First-Out (FIFO): based on time of reading the page in
  - Last-in-first-out (LIFO): based on time of reading the page
  - Most-recently-used (MRU): based on time of last reference
  - etc.

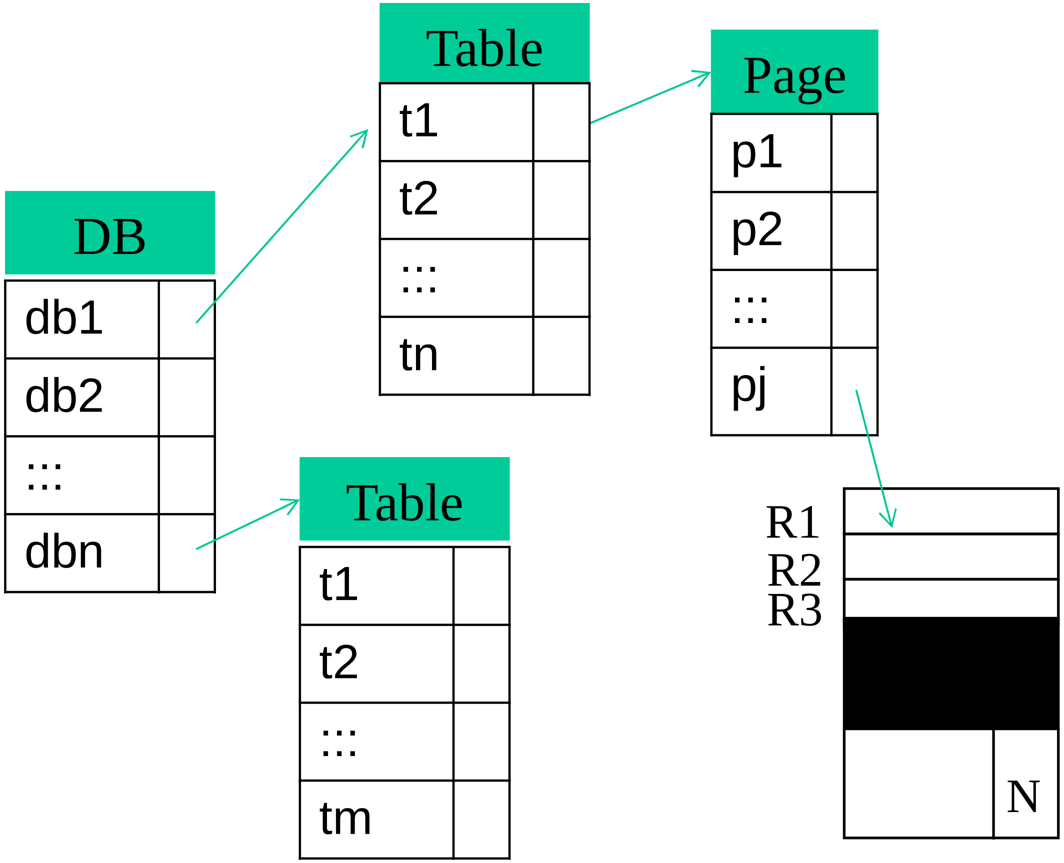
Access Manager

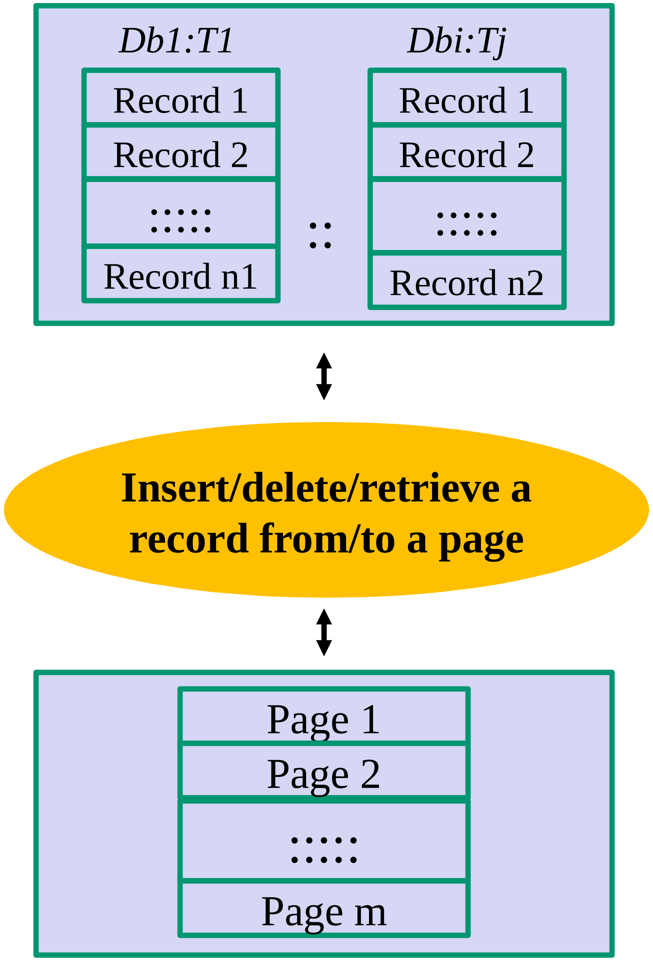
- Mapping between record id (db:table:rid) and page id
- Arrange records into a page
- Retrieve records from a page
- What to concern
  - Mapping between record and page
  - Record formats
  - Page formats
  - File formats



Record-Page Mapping

- Maintain a mapping table



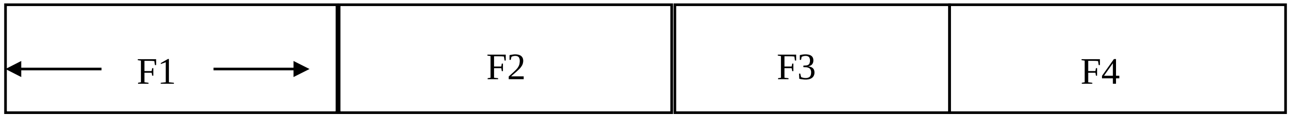


Insert/delete/retrieve a record from/to a page

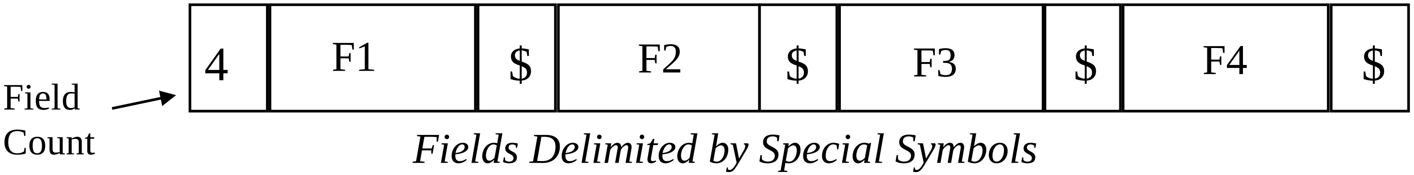
Record Formats

- Fixed length
  - Field types and lengths are stored in *system catalogs*

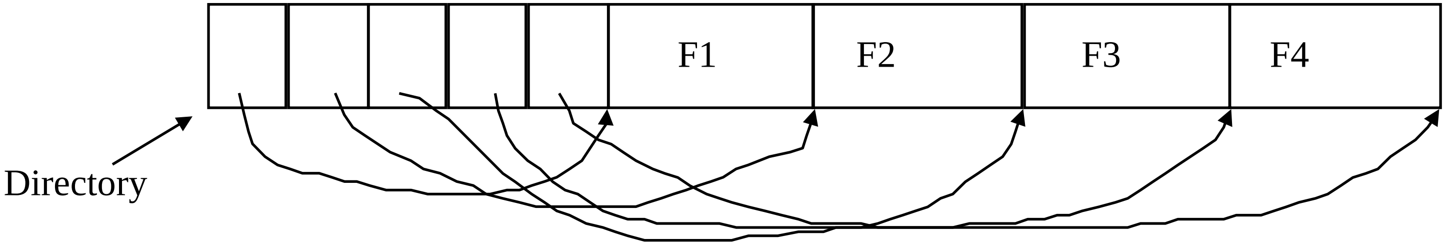
Name: char (40), Address: char (100), Phone: char (10), Email: char (100)



- Variable length (VARCHAR, BLOB-binary large object, etc.)
  - Two alternative formats (number of fields is fixed)



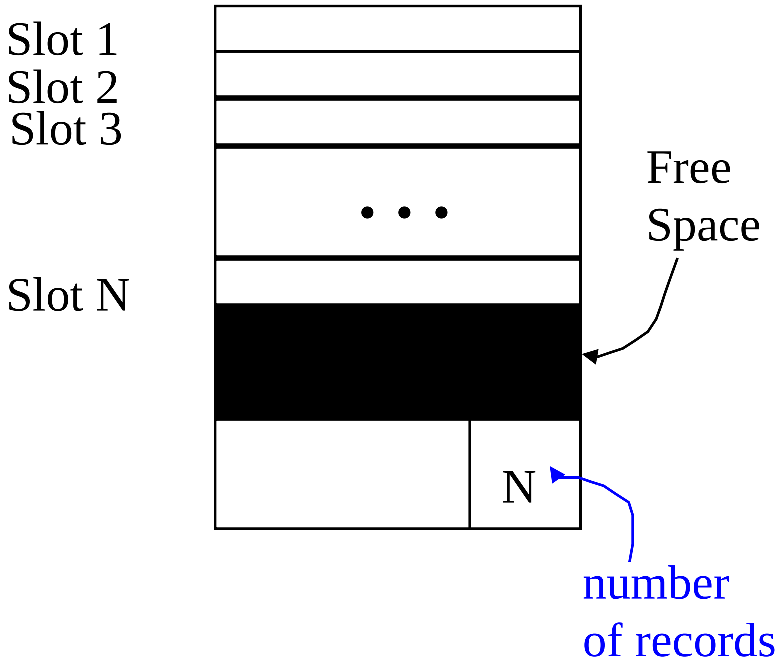
Fields Delimited by Special Symbols



Array of Field Offsets

Page Formats: Fixed Length Records

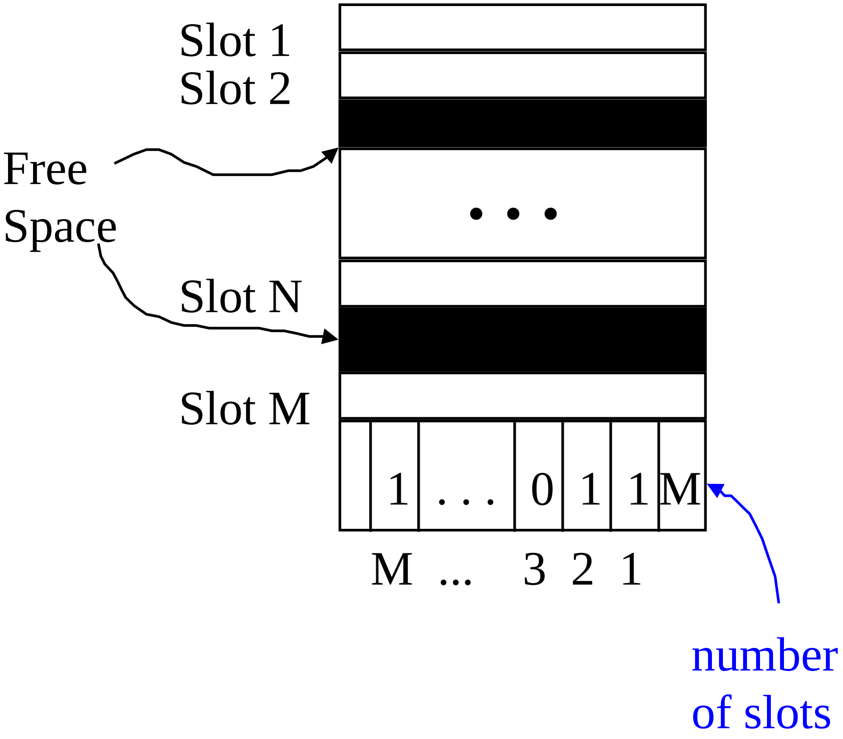
- Manage records within a page
  - Partition a page into a number of slots, each holding one record
  - Record the number of records (total = PageSize/RecordSize)
  - How to store records among slots?



SOLUTION 1: PACKED

- Need a variable N to record the number of slots being used
- The free slot starts from N+1
- When appending a new record, allocate slot N+1, then N++
- When deleting a record at slot i, move the last record to the slot i. If sorted, all records after slot i must be moved up

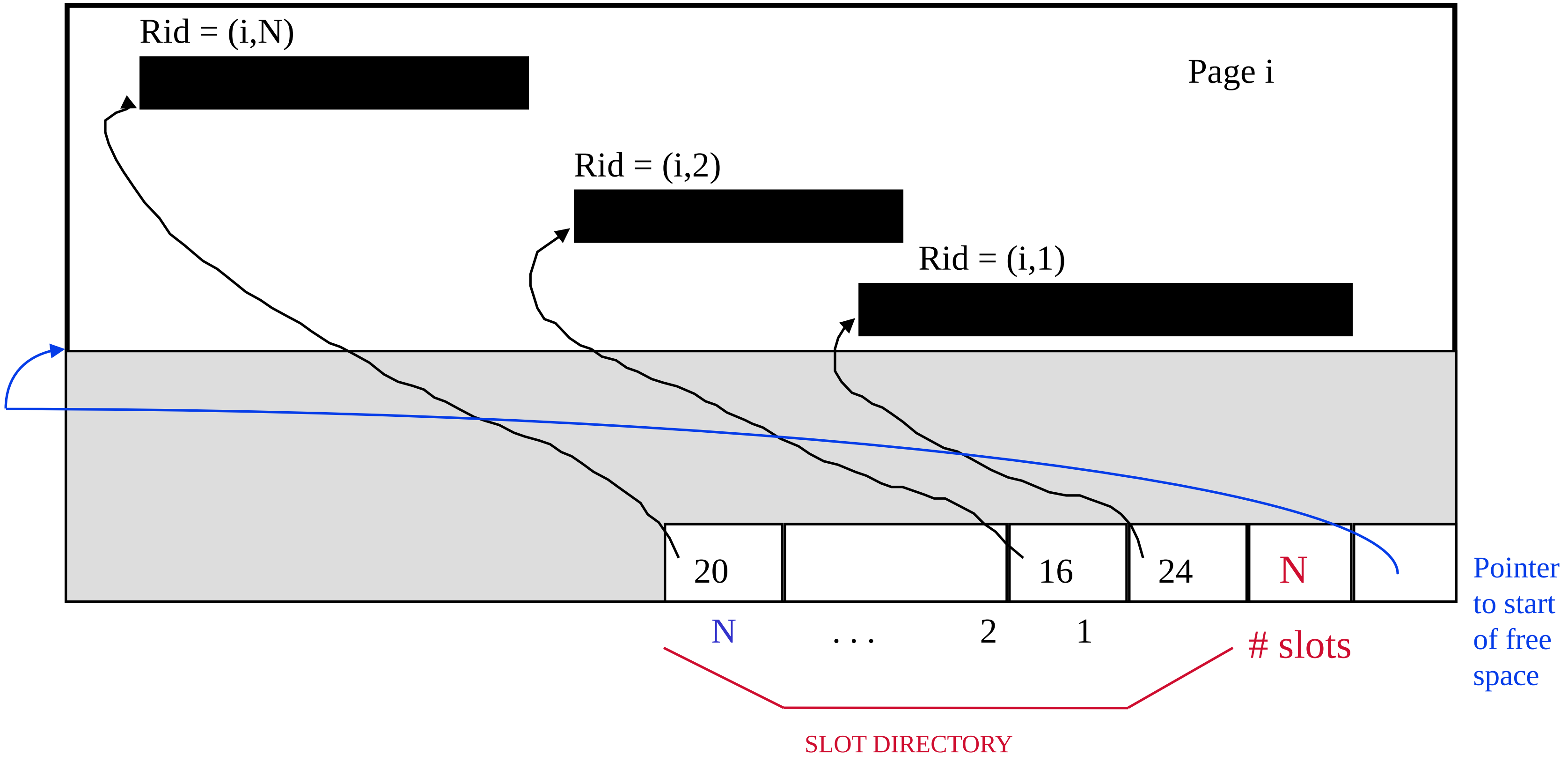
Page Formats: Fixed Length Records



SOLUTION 2: UNPACKED

- Need a bitmap to record if a slot is occupied or not
- If bit[i]==1, slot i is occupied
- When inserting a record, search the bitmap to find a bit that is 0, then allocate the corresponding slot for the record
- When deleting a record, simply reset the corresponding bit

Page Formats: Variable Length Records



- Maintain a directory of slots, which is a number of entries
- Each record stored in the page occupies one entry, which keep the information of the record, i.e., (record offset, record length)

File Formats and Operation Costs

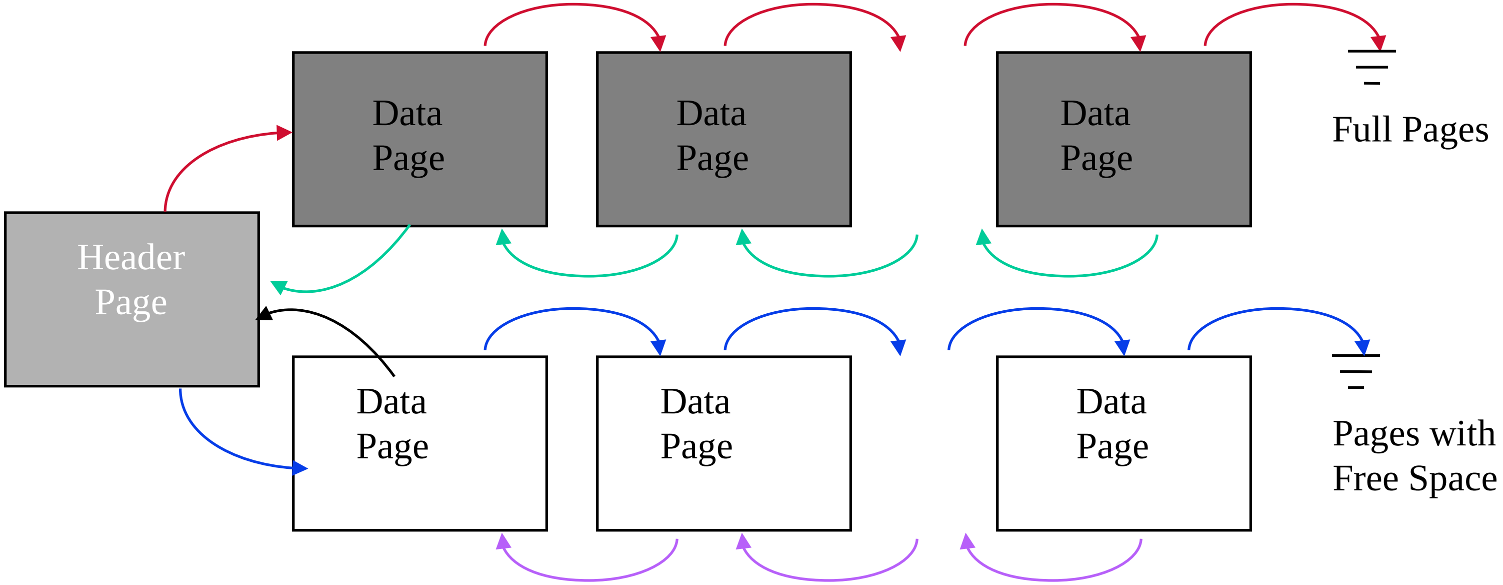
- Manage pages within a file/table, different file formats
  - Heap File: Suitable when typical access is a file scan retrieving all records.
  - Sorted File: Best if records must be retrieved in some order, or only a `range’ of records is needed.
  - Hashed File: Good for equality selections.

- Cost factors (we ignore CPU costs, for simplicity)
  - **P:** The number of data pages
  - **R:** Number of records per page
  - **D:** (Average) time to read or write disk page
  - Measuring number of page I/O’s ignores gains of pre-fetching blocks of pages; thus, even I/O cost is only approximated.

Heap Files

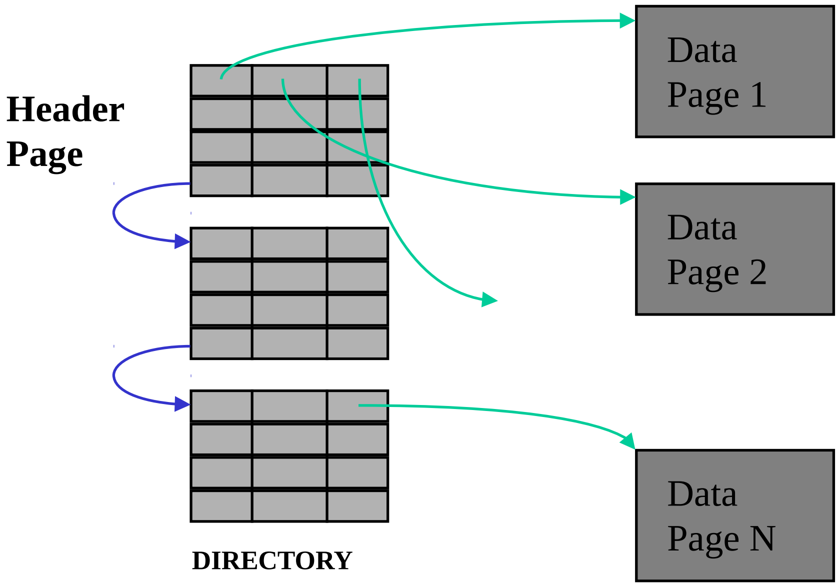
- The data in a heap file is not ordered.
  - How to find a page that has some free space
  - How to find the free space inside a page
- Two types of implementations
  - Link-based
  - Directory-based

Heap File Implemented as a List



- To insert a record, one searches the pages with free space and find the one that has sufficient space
  - Many pages may contain some tiny free space
  - A long list of pages may have to loaded in order to find an appropriate one

Heap File Using a Page Directory



- The directory is a collection of pages;
  - Each page contains a number of entry
  - Each entry contains a pointer linking to the page and a variable recording the free space
- The number of directory pages is much smaller than that of data pages

Heap Files and Associated Costs

P: The number of data pages

R: Number of records per page

D: (Average) time to read or write disk page

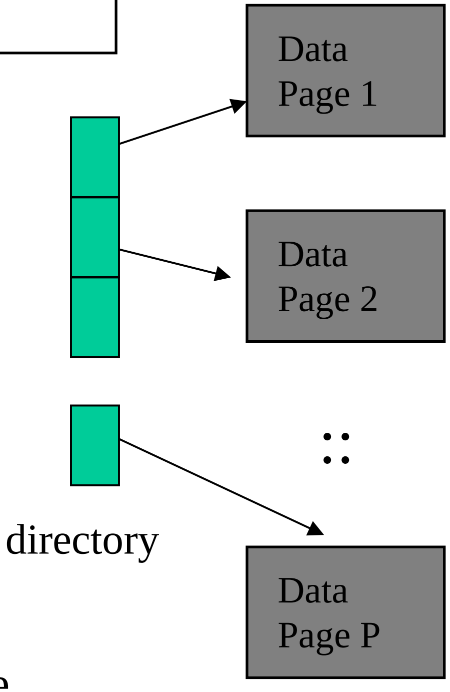
Scan: P*D

Search with equality selection: If a selection is based on a candidate key, on average, we must scan half the file, assuming that the record exists 0.5*P*D.

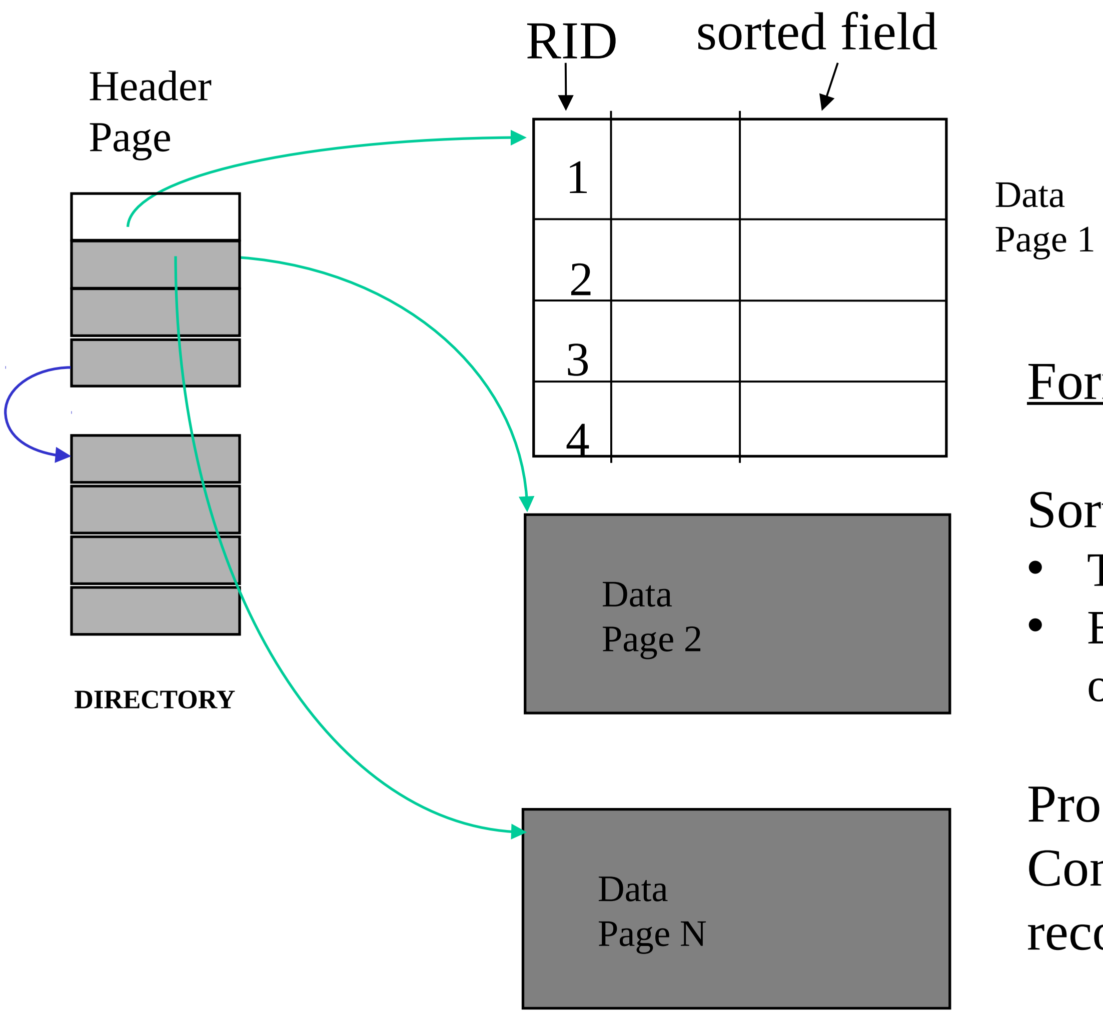
Search with range selection: The entire file must be scanned. The cost is P*D.

Insert: Assume that records are always inserted at the end of the file. We fetch the last page in the file, add the record, and write the page back. The cost is 2D.

Delete:. The cost also depends on the number of qualifying records. The cost is search cost + D.



Sorted File



RID

sorted field

Header Page

1

2

3

4

Data Page 1

Data Page 2

Data Page N

DIRECTORY

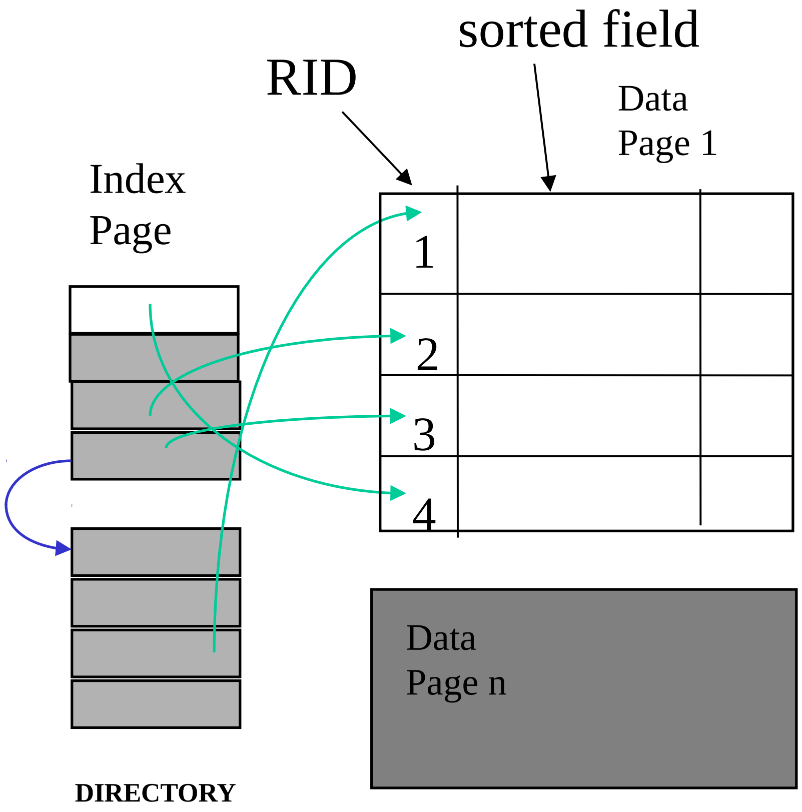
<u>Format 1</u>

Sort records directly

- The records in each page is sorted
- Each entry in a header page link to one page

Pro: minimum page header overhead
Con: Expensive when inserting a record

Sorted File



Format 2

Keep the sorted field using index page

- Each entry points to a record
    - May contain the value of the sorted field
    - May contain a valid bit for deleting operation
- The entries are sorted according to the sorted field

PRO: Inserting a record just need to reorganize the index page, the size of which is much smaller

CON: Each entry in an index page links only one record, so the size of index page is much larger than that in "Format 1"

Sorted Files and Associated Costs

Scan: P * D

Search with equality selection: Assume that the selection is specified on the field by which the file is sorted. The cost is D * log P, assuming that the sorted file is stored sequentially.

Search with range selection:

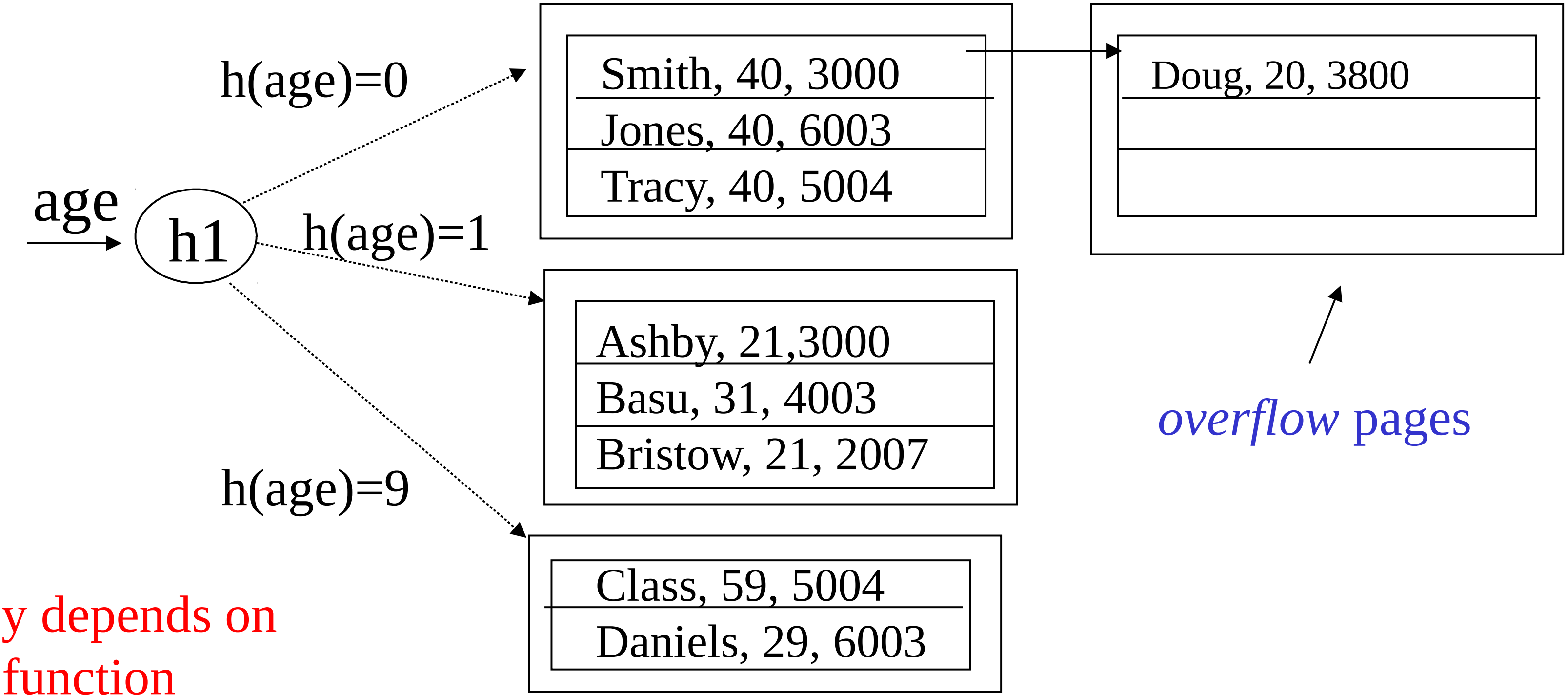
D * log P + cost of retrieving qualified records.

Insert: Search cost + 2*0.5*P*D; the assumption is that the inserted record belongs in the middle of the file.

Delete: Search cost + 2*0.5*P*D; the assumption is that we need to pack the file and the record to be deleted is in the middle of the file.

Hashed files

- File is a collection of *buckets*.
- Bucket = *primary* page plus zero or more *overflow* pages.
- Hashing function **h**: **h**(*r*) = bucket in which record *r* belongs. **h** looks at only some of the fields of *r*, called the *search fields*.



age

h1

h(age)=0

h(age)=1

h(age)=9

Smith, 40, 3000
Jones, 40, 6003
Tracy, 40, 5004

Doug, 20, 3800

Ashby, 21,3000
Basu, 31, 4003
Bristow, 21, 2007

Class, 59, 5004
Daniels, 29, 6003

overflow pages

File hashed on age

Efficiency depends on

- Hash function
- Data skew factor

Hashed Files and Associated Costs

Assume that there is no overflow page.

Scan: 1.25*P*D if pages are kept at 80% occupancy

Search with equality selection:D

Search with range selection: 1.25*P*D

Insert: Search cost + D = 2D

Delete: Search cost + D = 2D

Cost Comparison

	Heap File	Sorted File	Hashed File
Scan all records	PD	PD	1.25 PD
Equality Search	0.5 PD	D log $_2$P	D
Range Search	PD	D (log $_2$P + # of pages with matches)	1.25 PD
Insert	2D	Search + PD	2D
Delete	Search + D	Search + PD	2D

- P: number of pages, D: the cost of read/write a page
- Several assumptions underlie these (rough) estimates!

Quick Review

1. Storage hierarchy
2. Storage management
   1) Access manager
      - db:table:record □□page
   2) Disk manager
      - page □□ physical storage
   3) Buffer manager
      - replacement policy
3. Data format
   1) Record format (fixed length, variable length)
   2) Page format (packed, unpacked, etc.)
   3) File format (heap, sort, hash)
4. Operation cost w.r.t. File formats