Database Systems

Data Storage and Buffer Management III

何明昕 HE Mingxin, Max

Send your email to **c.max@yeah.net** with

a subject like: **DBS***345-Andy: On What …*

Download from **c.program@yeah.net**

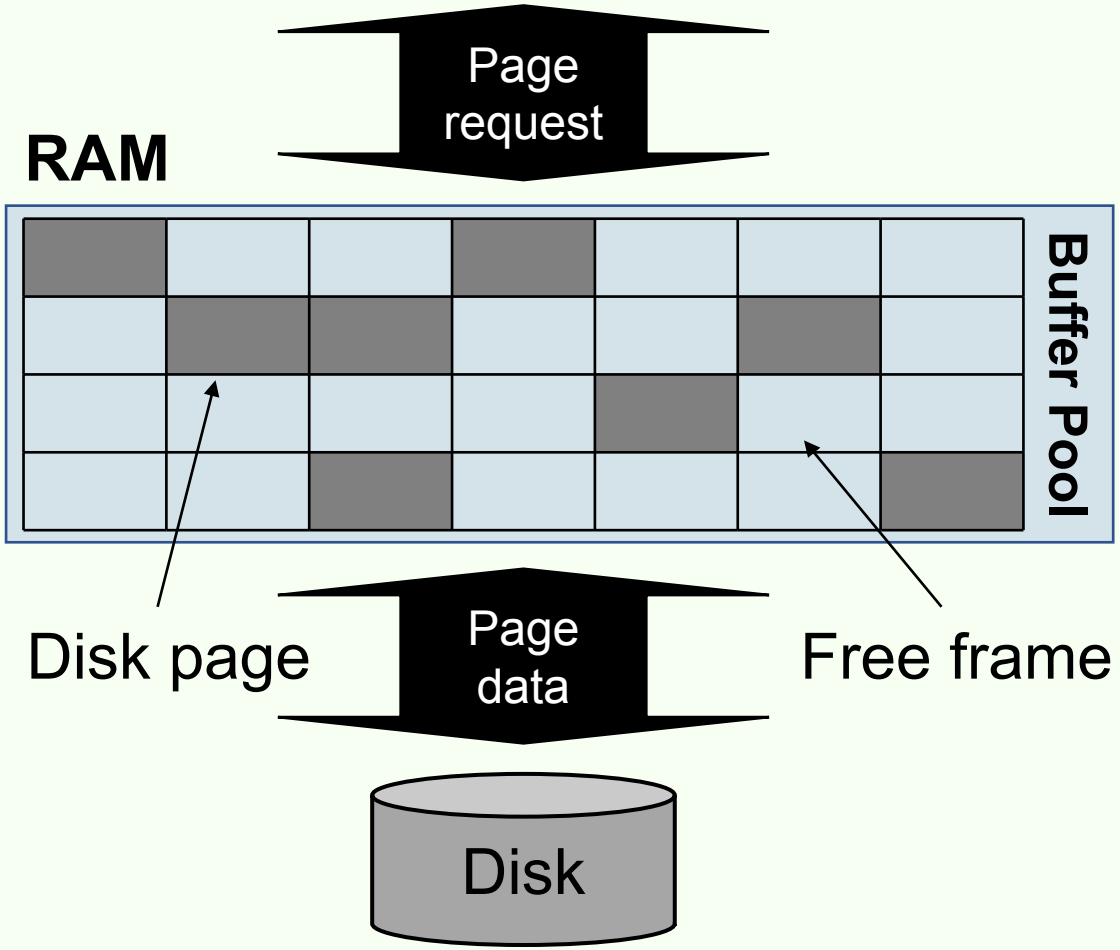
/文件中心/网盘/DatabaseSystems2021

Data Storage and Buffer Management III

Where it hits the metal

Recap: Buffer Manager

- Requests to buffer manager:
  - Request a page (*pin*)
  - Release a page when it is no longer needed (*unpin*)
  - Notify the buffer manager when a page is modified (set *dirty bit*)
- Buffer replacement policies
  - Least recently used (LRU)
  - Clock
  - Most recently used (MRU)
  - FIFO
  - Random, …
- Sequential flooding

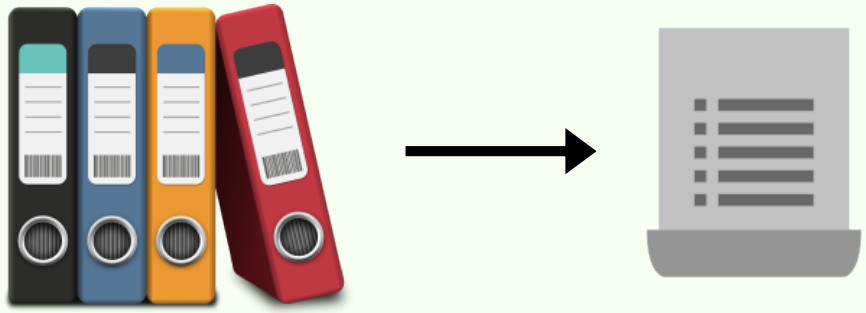


Recap: Tables on Disk

- Data is stored in tables
- Each table is stored in a file on disk
- Each file consists of multiple pages
- Each page consists of multiple records (i.e. tuples)
- Each records consists of multiple (attribute, value) pairs
- Data is allocated/deallocated in increments of pages
- Logically-close pages should be nearby in the disk

Files of (Pages of) Records

- I/O is performed in page units
- However, higher level data management operations require accessing and manipulating records and files of records
- File = collection of pages
  Page = collection of records
- File operations
  - Insert/delete/modify a record
  - Read a particular record (specified using a *record ID*)
  - Scan all records (possibly with some condition(s) on the records to be retrieved)
- Records are organized in files using various formats, i.e. *file organizations*
  - Each file organizations makes certain operations efficient but other operations expensive

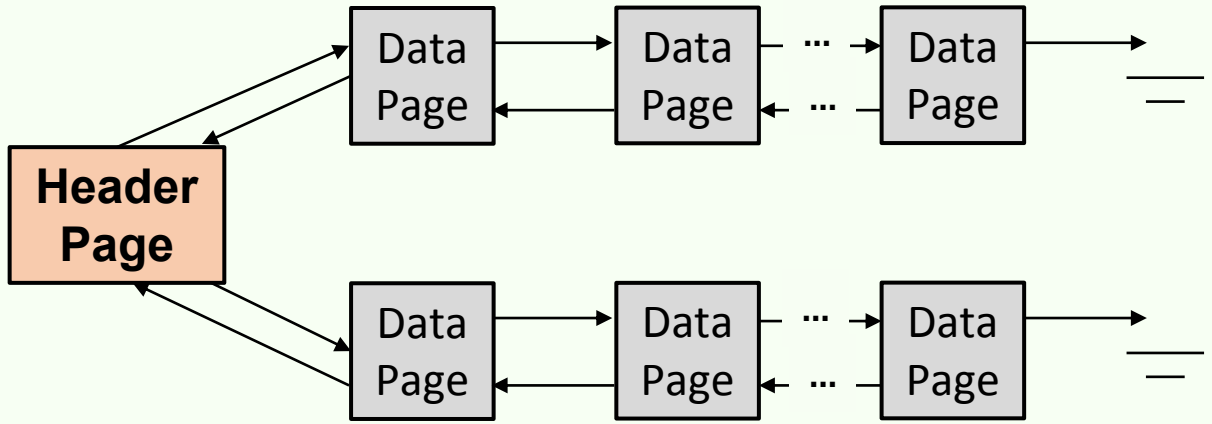


Unordered (Heap) File

- Simplest file organization
- Each page contains records in no particular order
- As the file grows and shrinks, pages are allocated and de-allocated
- To support record level operations, we must keep track of:
  - Pages in a file (using page ID (pid))
  - Free space on pages
  - Records on a page (using record ID (rid))
- Operations: create/destroy file, insert/delete record, fetch a record with a specified rid, scan all records

Heap File Implemented as a List

- (heap file name, header page id) are stored somewhere
- Each (data) page has two pointers + data
  - Each pointer stores a pid
- Pages in the free space list have “some” free space



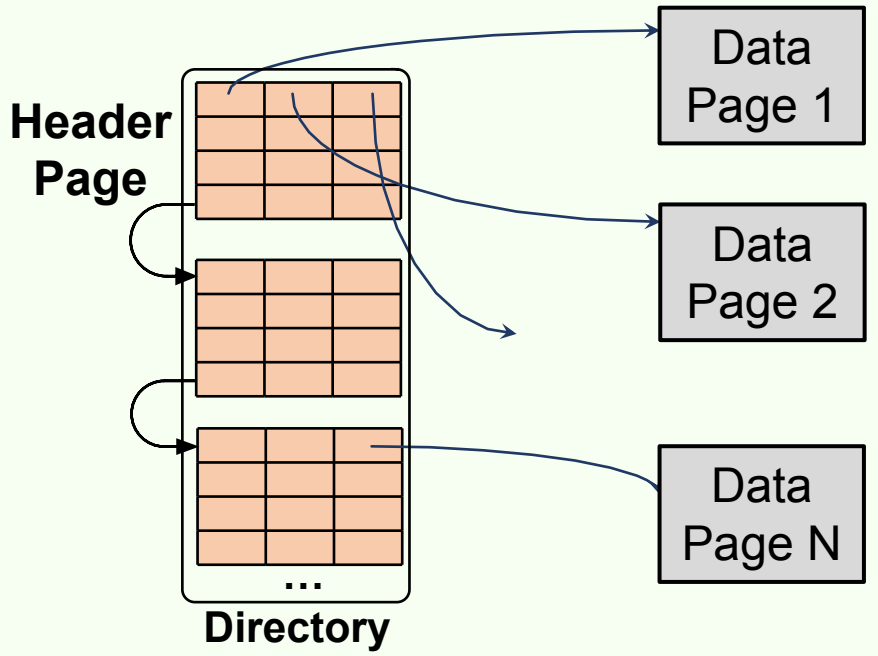
Q: What happens with fixed- vs. variable-length records in terms of full and non-full pages?

Q: How would you find a record with a specific rid?

Q: Why do you need the backward pointers?

Heap File Implemented Using a Page Directory

- Each entry for a page also keeps track of
  - whether the page full or not, or
  - number of free bytes on the page
- Can locate pages for new tuples faster
- Directory is a collection of pages; linked list implementation is just one alternative



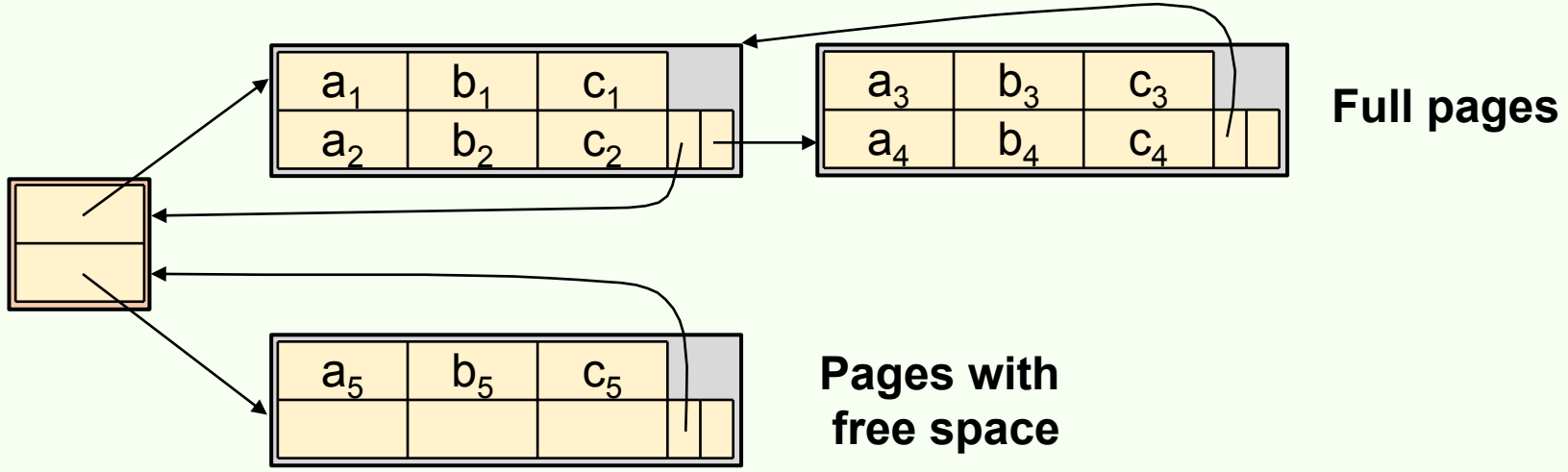
Q: What happens with fixed- vs. variable-length records in terms of full and non-full pages?

Q: How would you find a record with a specific rid?

(Grossly Simplified) Example

R

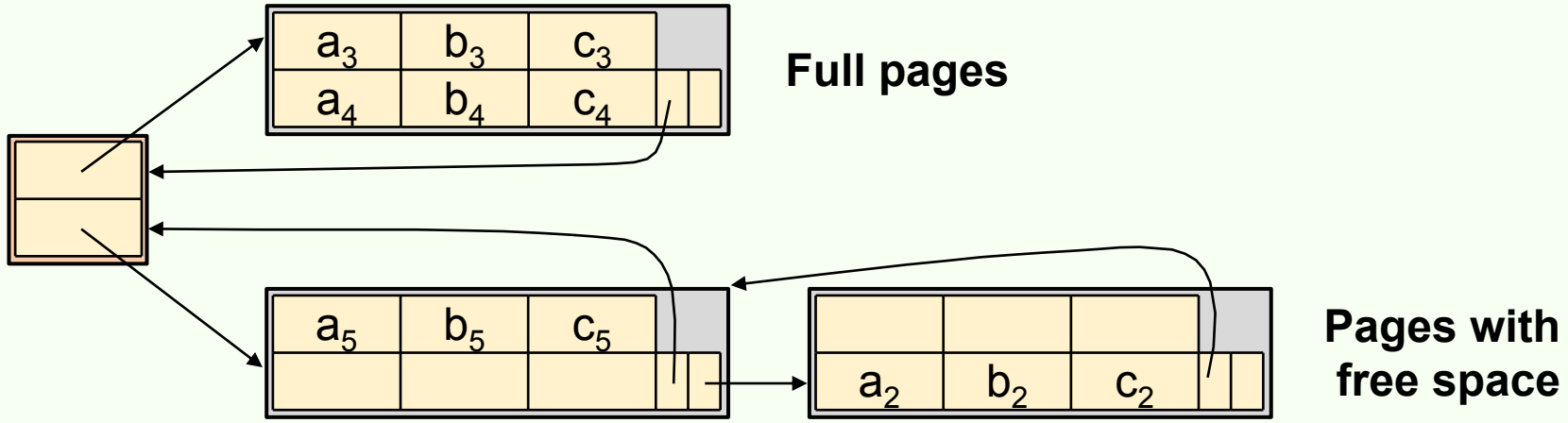
a	b	c
a_1	b_1	c_1
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_5	c_5




```
DELETE FROM R WHERE a = a1;
```

R

a	b	c
a_2	b_2	c_2
a_3	b_3	c_3
a_4	b_4	c_4
a_5	b_5	c_5



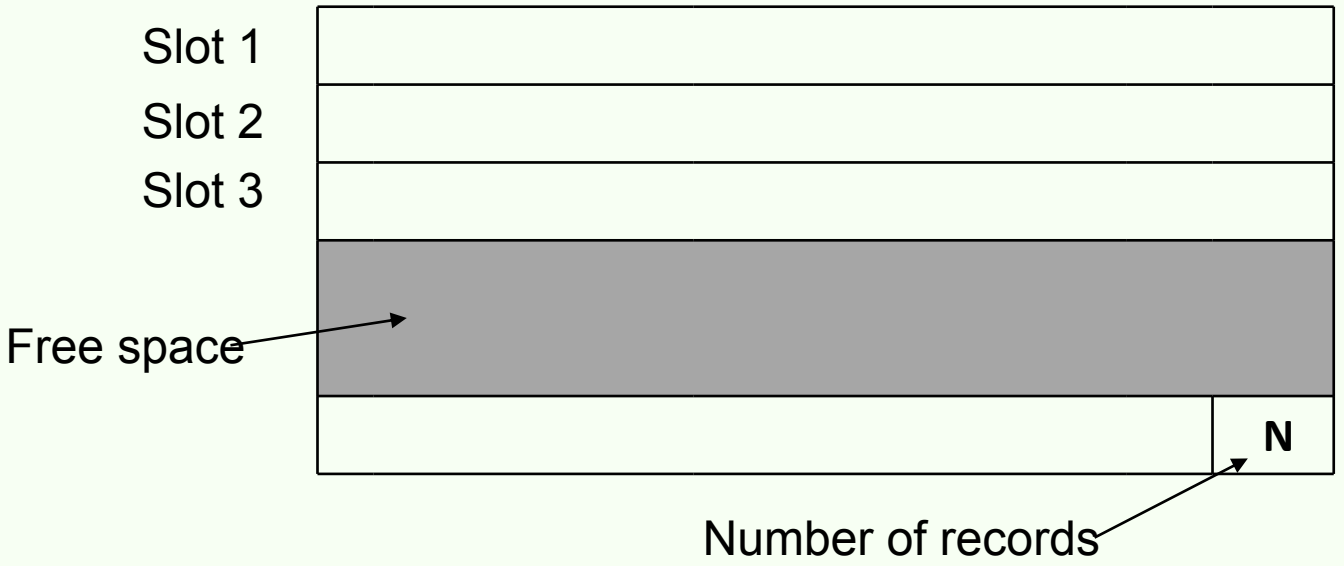
Page Organizations

- A page as a collection of records/tuples
- Queries deal with tuples
- Hence, the *slotted page format*
  - A page is a collection of *slots*
  - Each slot may contain a record
- rid = <pid, slot number>
- Many slotted page organizations
  - Need to support search, insert and delete records on a page
- Page organizations for various *record organizations*
  - Fixed-length records
  - Variable-length records

Other ways of generating rids?

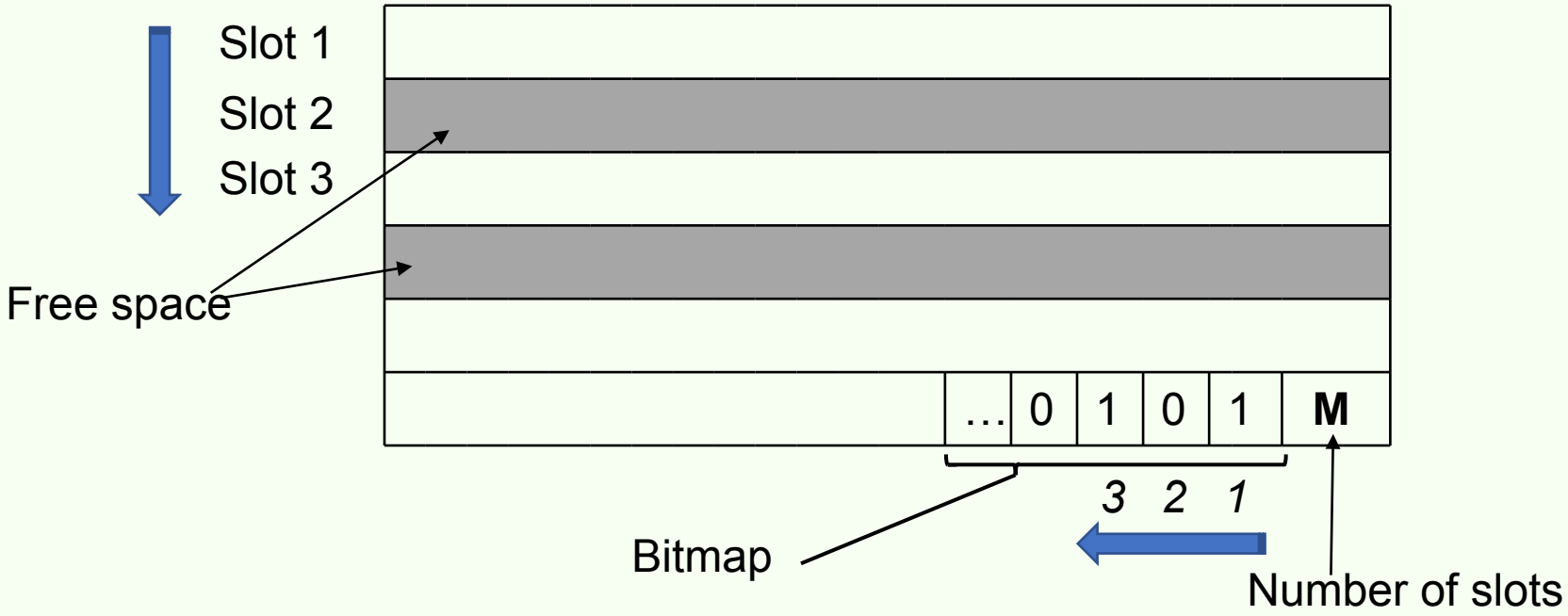
Organization of Pages of Fixed-length Record

1. Packed organization
   - N records are always stored in the first N slots
- Problem: moving records (for free space management) changes the record ID
   - Might not be acceptable



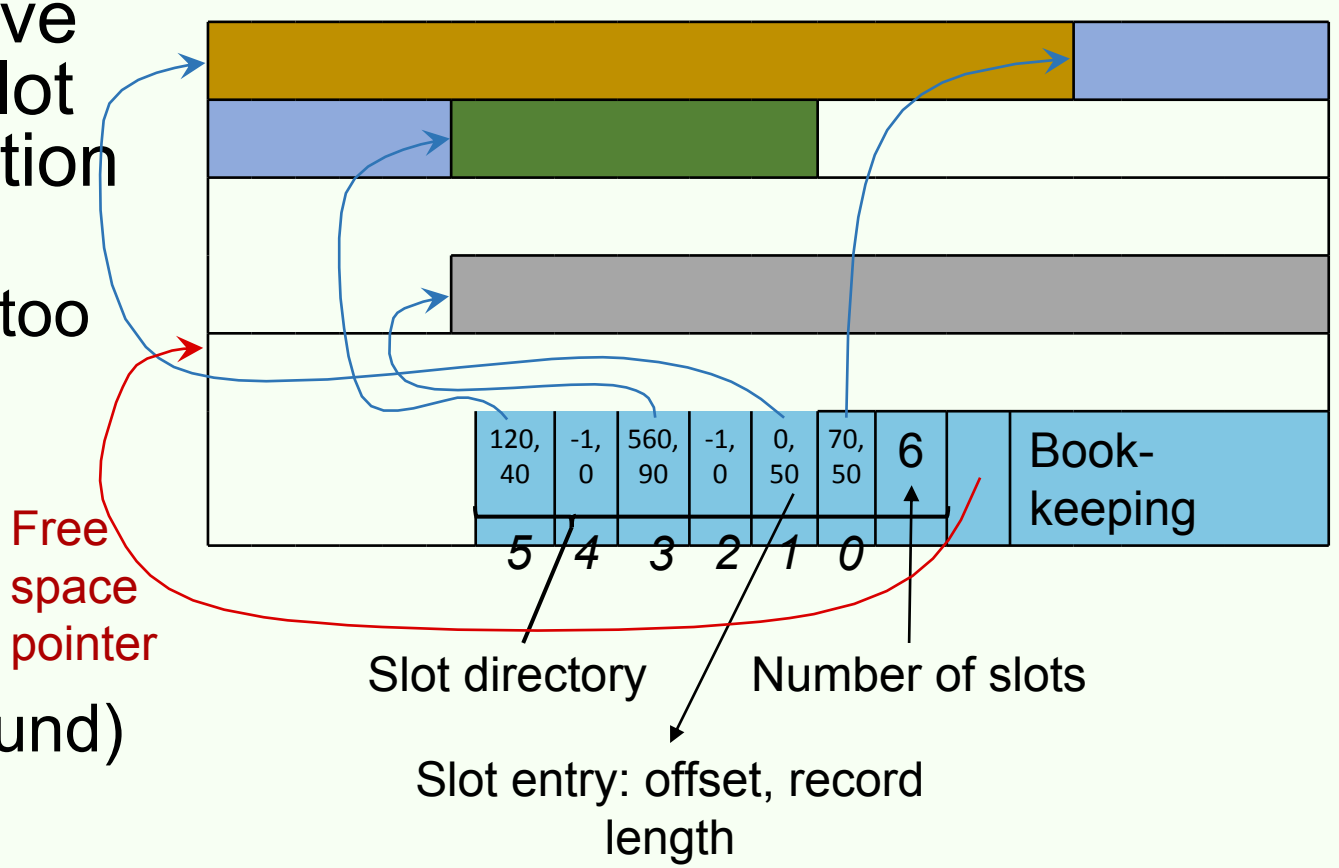
Organization of Pages of Fixed-length Record (Cont.)

2. Unpacked organization
   - Use a *bitmap* to locate records in the page



Organization of Pages of Variable-length Record

- Directory grows backwards
- rid does not change if you move a record on the same page (slot number is determined by position in slot directory)
  - Good for fixed-length records too
- Deletion: offset is set to -1
- Insertion
  - Use any available slots
  - if no space is available, reorganize (move records around)



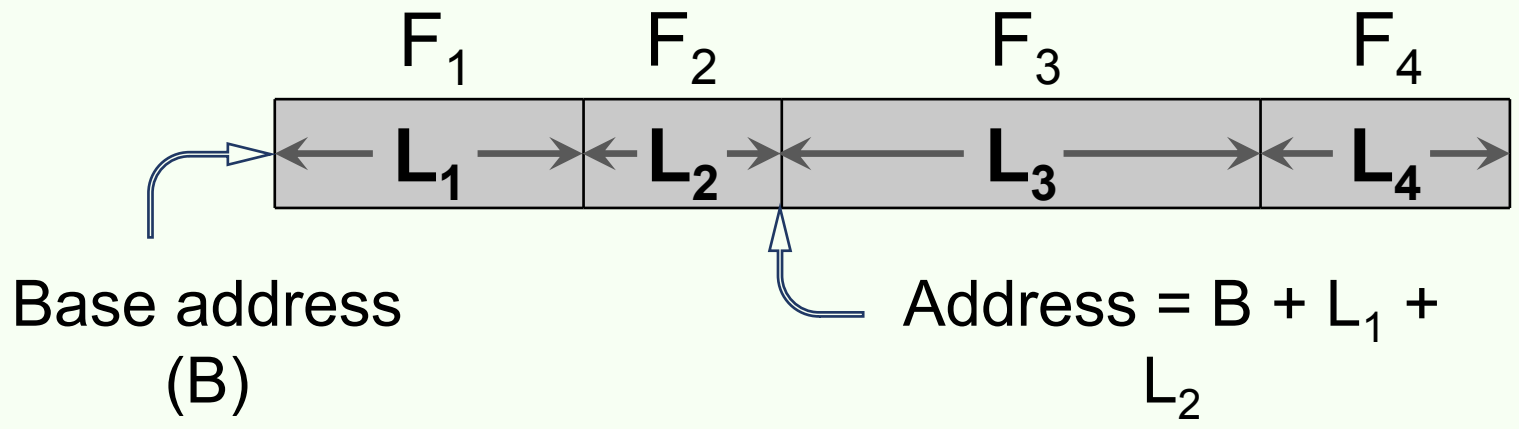
Record Organization (Format)

- Recap
  - File organization
    - Heap file
      - As doubly-linked list
      - Using page directory
  - Page organization
    - For fixed-length records
      - Packed
      - Unpacked
    - For variable-length records

Let's see fixed- and variable-length record formats now.

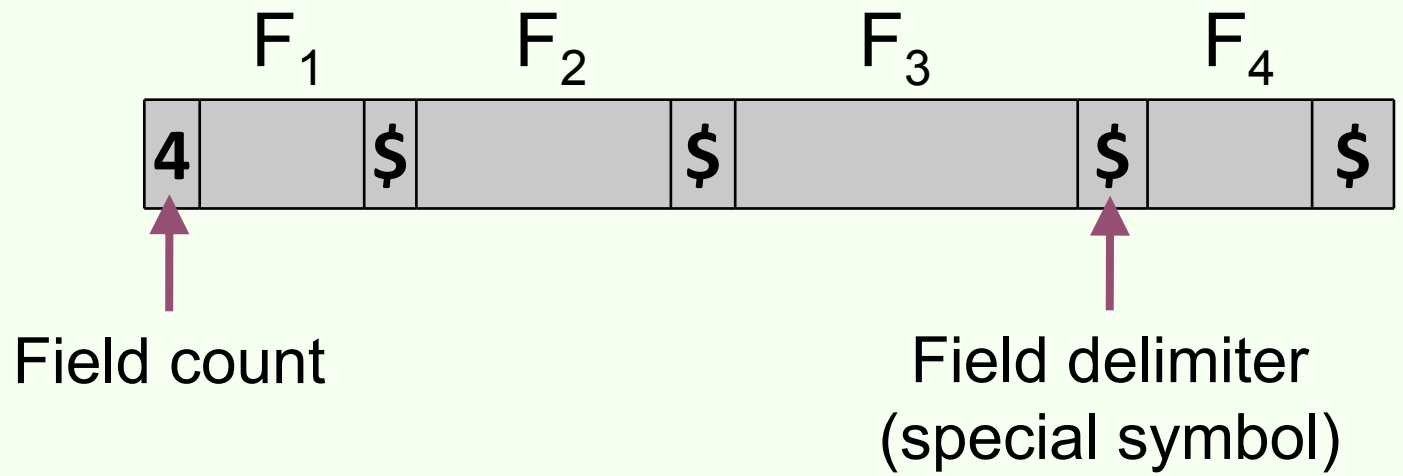
Fixed-length Records

- All records are of the same length, number of fields and field types
  - These information are stored in *system catalog*
- The address of each field can be computed from the information in the catalog



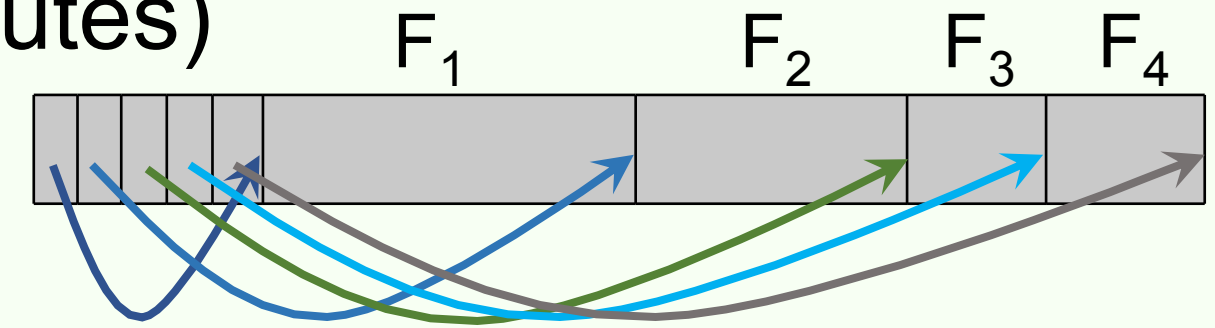
Variable-length Records, Option 1

- Store fields consecutively
- Use delimiters to denote the end of each field
- Need to scan the whole record to locate a field



Variable-length Records, Option 2

- Store fields consecutively
- Use an array of integer offsets in the beginning
  - Efficient storage of NULLs, small directory overhead, direct access to the i^{th} field
- Issues with growing records (change in attribute value, add/drop attributes)



Motivating Example: Looking Up One Attribute

CREATE TABLE T (a INTEGER, b INTEGER, c VARCHAR(255), …);

```
SELECT a
FROM T
WHERE a > 10;
```

- What can potentially go wrong if we use any of the previous record formats?

Column Stores

- Store columns *vertically*
- Each column of a relation is stored in a different file
  - Can be compressed as well
- Contrast with a *row store* that stores all the attributes of a tuple/record contiguously

File 1

1234	45	Here goes a very long sentence 1
4657	2	Here goes a very long sentence 2
3578	45	Here goes a very long sentence 3

Row store

File 1 *File 2* *File 3*

1234	45	Here goes a very long sentence 1
4657	2	Here goes a very long sentence 2
3578	45	Here goes a very long sentence 3

Column store

Recap

- Architecture of a typical DBMS
- Memory hierarchy
  - CPU cache, main memory, SSD, disk, tape
- Secondary storage (disk and SSD)
- Buffer manager
  - Buffer replacement policies: LRU, clock, MRU, FIFO, random, …
- File organization
  - Heap file (as doubly-linked list or directory)
- Page organization
  - For fixed-length (packed and unpacked) and variable-length records
- Record organization
  - Fixed-length and variable-length (two variations) records
- Row-store vs. column-store

Next Up

Indexing

Questions?