ICOM 6005 – Database Management Systems Design

Dr. Manuel Rodríguez-Martínez

Electrical and Computer Engineering Department

Readings

- Read
  - New Book: Chapter 9
  - Paper by Stone Braker

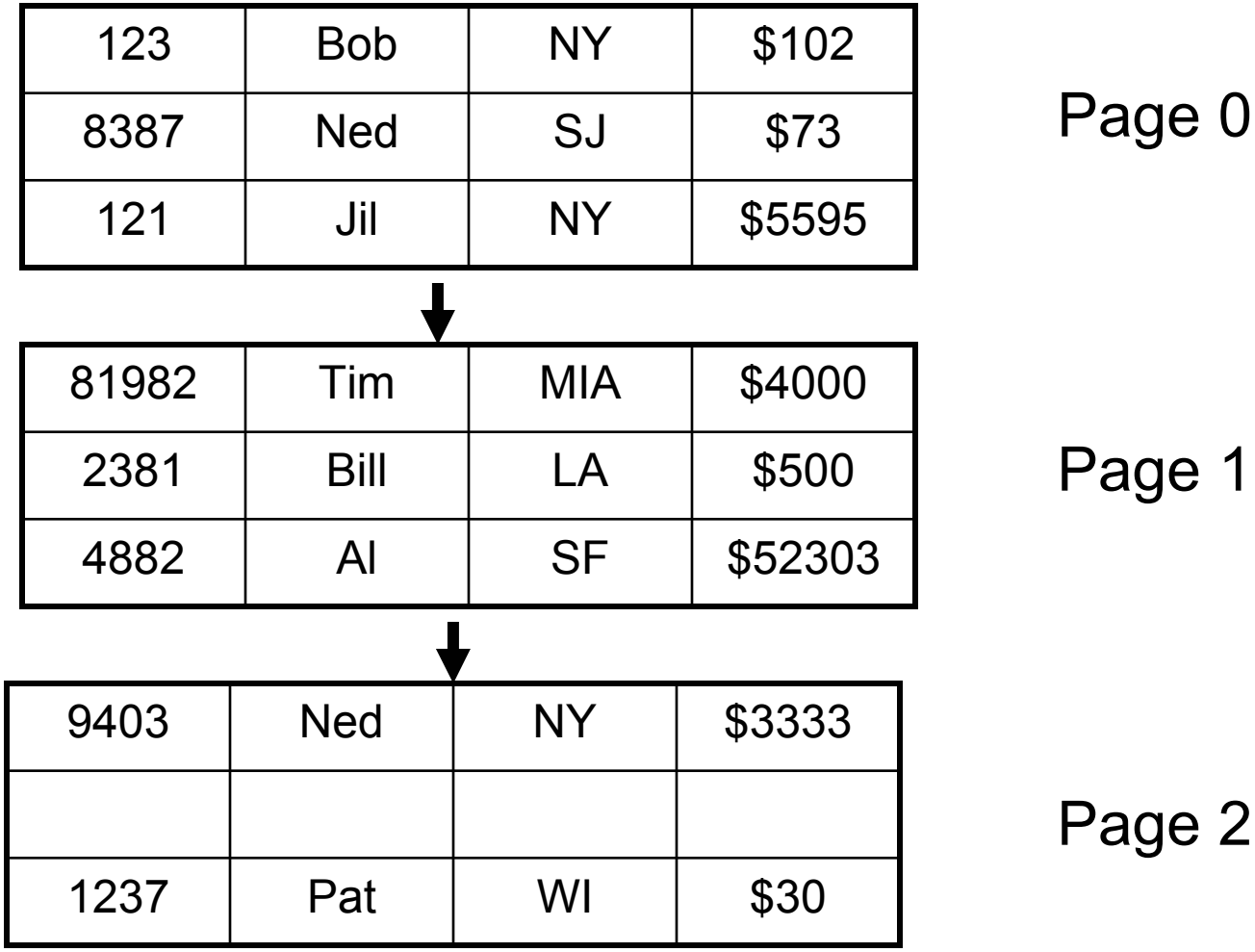
File and Access Methods Layer

- Buffer Manager provides a stream of pages
- But higher layers of DBMS need to see a stream of records
- A DBMS file provides this abstraction
  - File is a collection of records that belong to a relation R.
    - For example: Relation students might be stored in relation students.dat
  - File is made out of pages, and records are taken from pages
- File and Access Methods Layer implements various types of files to access the records
  - Access method – mechanism by which the records are extracted from the DBMS

File Types

- **Heap File** - Unordered collection of records
  - Records within a page a not ordered
  - Pages are not ordered
  - Simple to use and implement
- **Sorted File** – sorted collection or records
  - Within a page, records are ordered
  - Pages are ordered based on record contents
  - Efficient access to data, but expensive to maintain
- **Index File** – combines storage + data structure for fast access and lookups
  - Index entries – store value of attributes as search keys
  - Data entries – hold the data in the index file

Heap File


123 Bob NY $102
8387 Ned SJ $73
121 Jil NY $5595
Page 0
81982 Tim MIA $4000
2381 Bill LA $500
4882 Al SF $52303
Page 1
9403 Ned NY $3333
1237 Pat WI $30
Page 2

Sorted File

121	Jil	NY	$5595
123	Bob	NY	$102
1237	Pat	WI	$30

Page 0

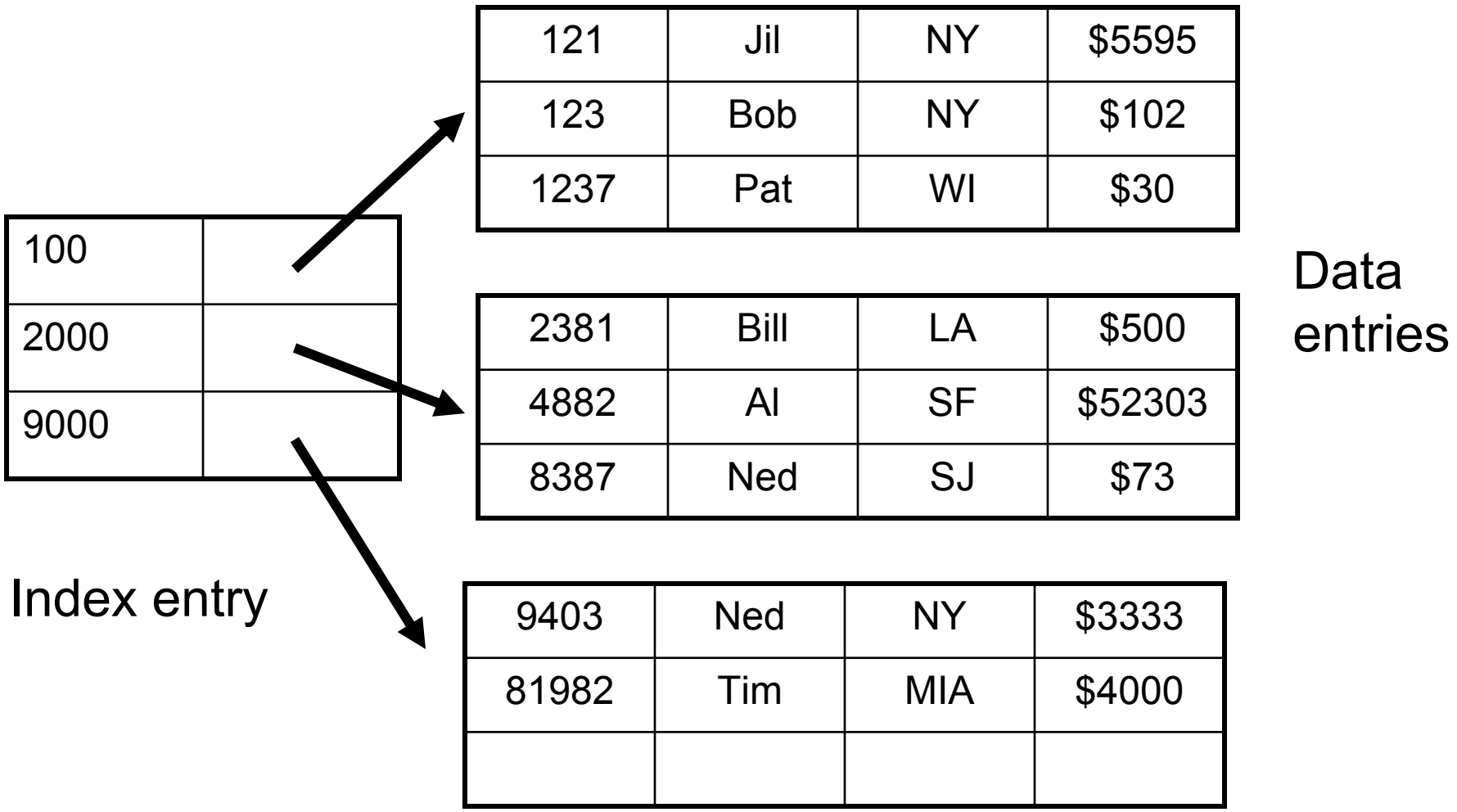
2381	Bill	LA	$500
4882	Al	SF	$52303
8387	Ned	SJ	$73

Page 1

9403	Ned	NY	$3333
81982	Tim	MIA	$4000

Page 2

Index File


121 Jil NY $5595
123 Bob NY $102
1237 Pat WI $30
2381 Bill LA $500
4882 Al SF $52303
8387 Ned SJ $73
9403 Ned NY $3333
81982 Tim MIA $4000
100
2000
9000
Data entries
Index entry

Index files structure

- Index entries
  - Store search keys
  - Search key – a set of attributes in a tuple can be used to guide a search
    - Ex. Student id
  - Search key do not necessarily have to be candidate keys
    - For example: gpa can be a search key on relation:

      Students(sid, name, login, age, gpa)
- Data entries
  - Store the data records in the index file
  - Data record can have
    - Actual tuples for the table on which index is defined
    - Record identifier for tuples that match a given search key

Issues with Index files

- Index files for a relation R can occur in three forms:
  - Data entries store the actual data for relation R.
    - Index file provides both indexing and storage.
  - Data entries store pairs <k, rid>:
    - k – value for a search key.
    - rid – rid of record having search key value k.
    - Actual data record is stored somewhere else, perhaps on a heap file or another index file .
  - Data entries store pairs <k, rid-list>
    - K – value for a search key
    - Rid-list – list of rid for all records having search key value k
    - Actual data record is stored somewhere else, perhaps on a heap file or another index file.

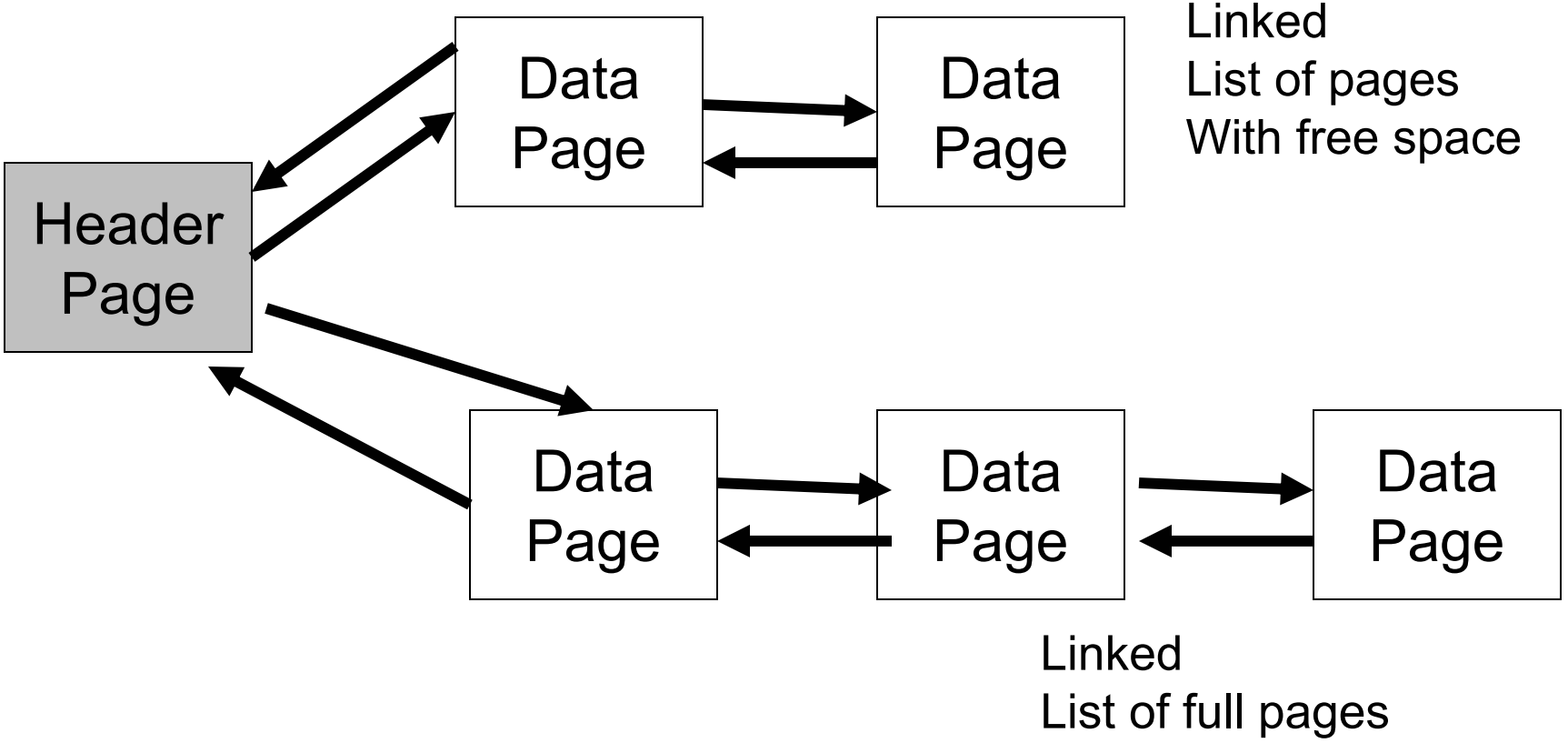
Operations on files

- Allocate file
- Scan operations
  - Grab each records one after one
    - Can be used to step through all records
- Insert record
  - Adds a new record to the file
    - Each record as a unique identifier called the record id (rid)
- Update record
- Find record with a given rid
- Delete record with a given rid
- De-allocate file

Implementing Heap Files

- Heap file links a collection of pages for a given relation R.
- Heap files are built on top of Buffer Manager.
- Each page has a page id
  - Often, we need to know the page size (e.g. 4KB)
    - All pages for a given file have the same size.
  - Page id and page size can be used to compute an offset in a cooked file where the page is located.
  - In raw disk partition, page id should enable DBMS to find block in disk where the page is located.

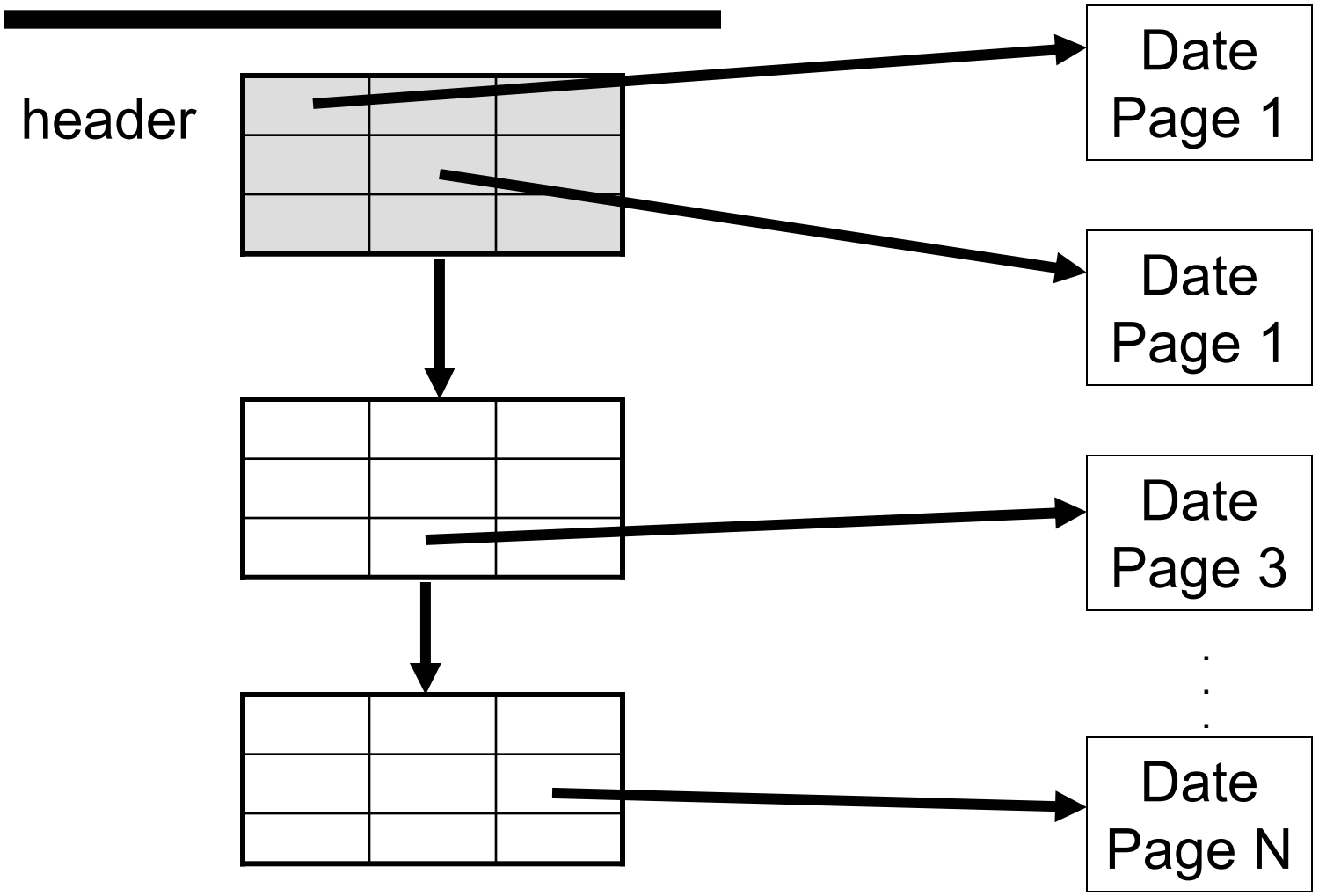
Linked Implementation of Heap Files



Linked List of pages

- Each page has:
  - records
  - pointer to next page
  - pointer to previous page
    - Pointer here means the integer with the page id of the next page.
- Header has two pointers
  - First page in the list of pages with free space
  - First page in the list of full pages
- Tradeoffs
  - Easy to use, good for fixed sized records
  - Complex to find space for variable length records
    - need to iterate over list with space

Directory of pages



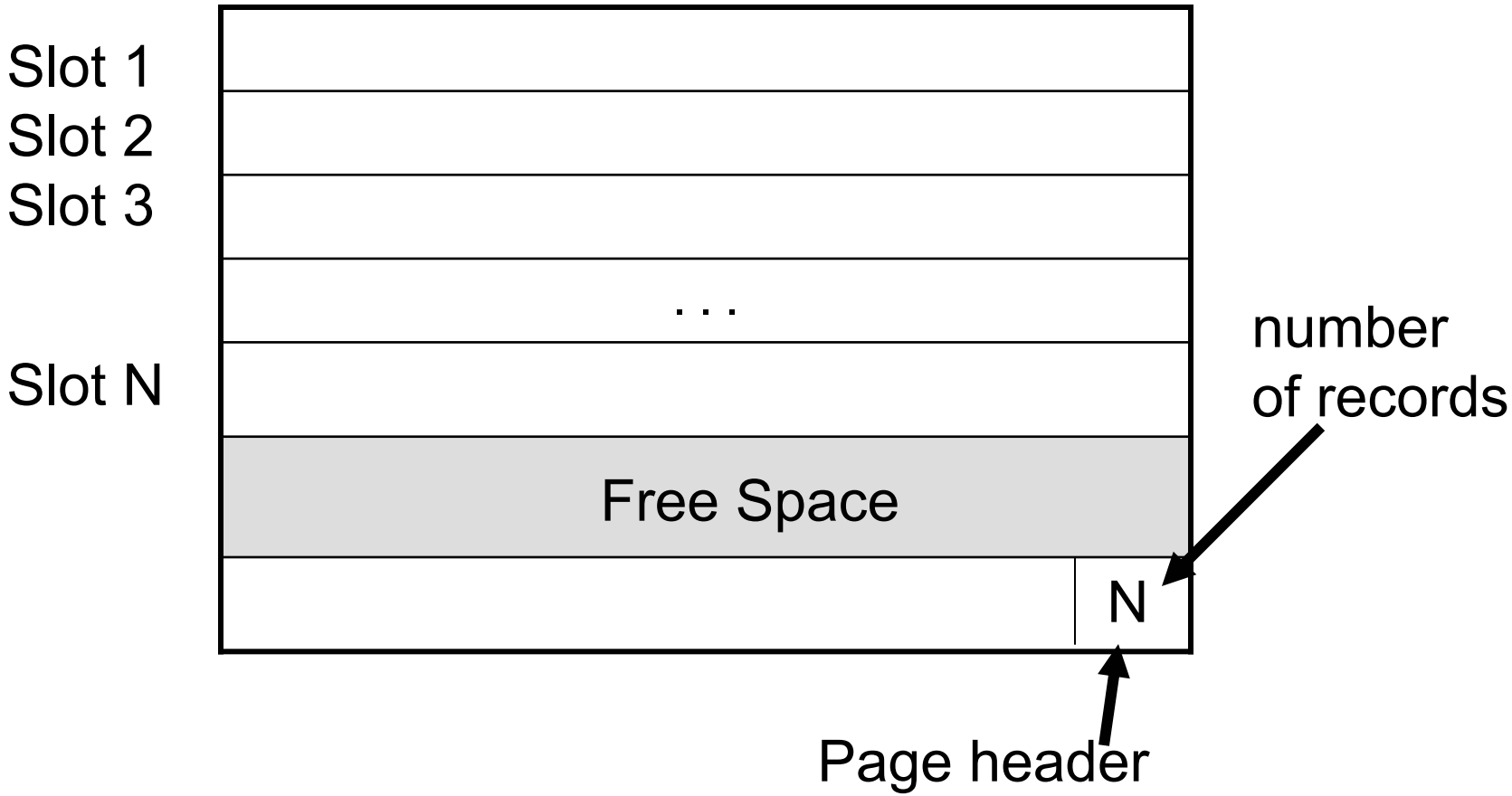
Directory of pages

- Linked list of directory pages
- Directory page has
  - Pointer to a given page
  - Bit indicating if page is full or not
  - Alternatively, have amount of space that is available
- More complex to implement
- Makes it easier to find page with enough room to store a new record

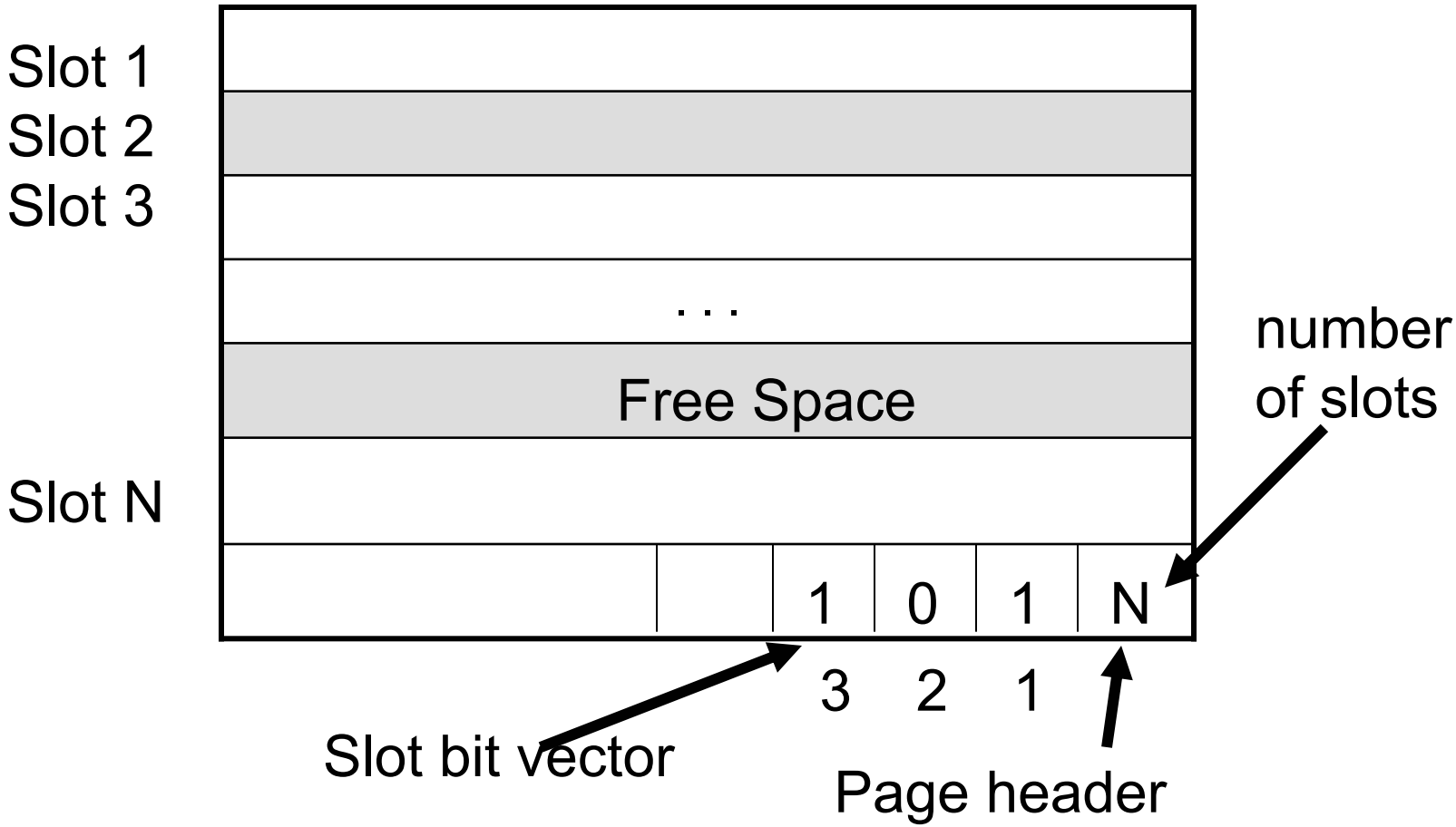
Page formats

- Each page holds
  - records
  - Optional metadata for finding records within the page
- Page can be visualized as a collection of slots where records can be placed
- Each record has a record id in the form:
  - <page_id, slot number>
  - page_id – id of the page where the record is located.
  - slot number – slot where the record is located.

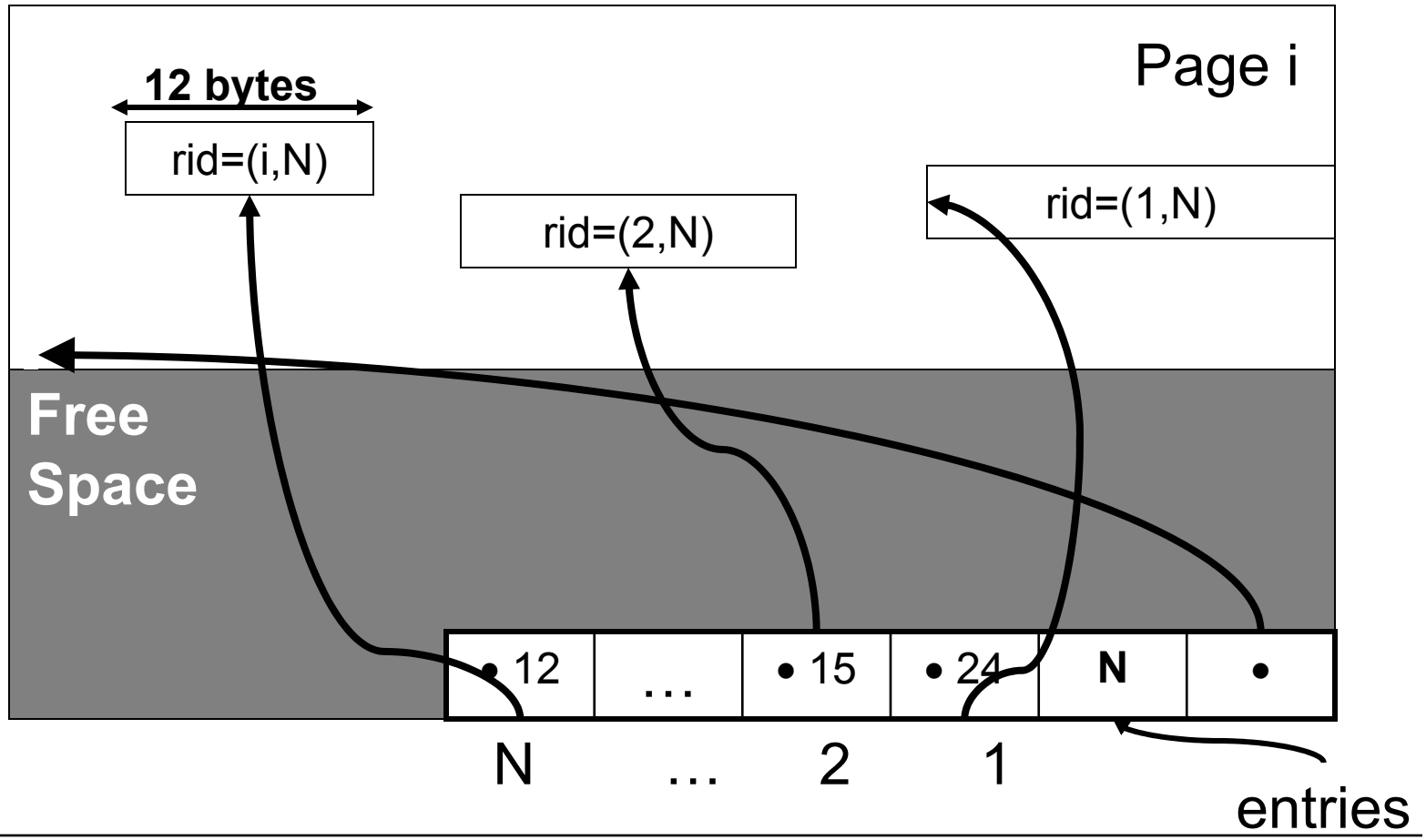
Packed Fixed-Length Record



Unpacked Fixed-Length Record



Variable-Length page



Fixed-Length records

- Size of each record is determined by maximum size of the data type in each column

Size in bytes	2	6	2	4

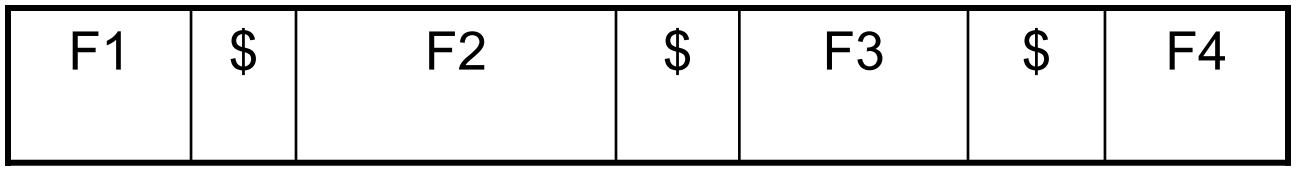
Offset of F1: 0
Offset of F2: 2
Offset of F3: 8
Offset of F4: 10

Need to understand the schema and sizes to find a given column

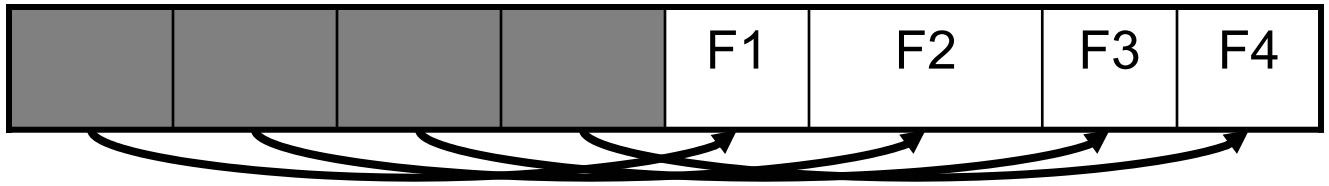
Variable-length records

- Either
  1. use a special symbol to separate fields
  2. use a header to indicate offset of each field

Option 1



Option 2



Option 1 has the problem of determining a good $
Option 2 handles NULL easily