

Fall 2013

BUFFER MANAGER, FILES AND RECORDS

(LOOSELY BASED ON THE COW BOOK: 9.4 – 9.7)

View of the disks



Query Execution

A bunch of files. Each file can be opened, scanned, (searched if its an index file), and modified

Storage Manager

Has to make this mapping happen

Disk

A bunch of sectors

Managing Disk Space

- How does the database IO layer work with the disk device. Two ways:
 1. OS exports a “raw” device interface, which essentially looks like one big file that is a large byte array
 2. OR, the DBMS grabs a big file/directory space in the OS and then uses the OS file as a container for the database

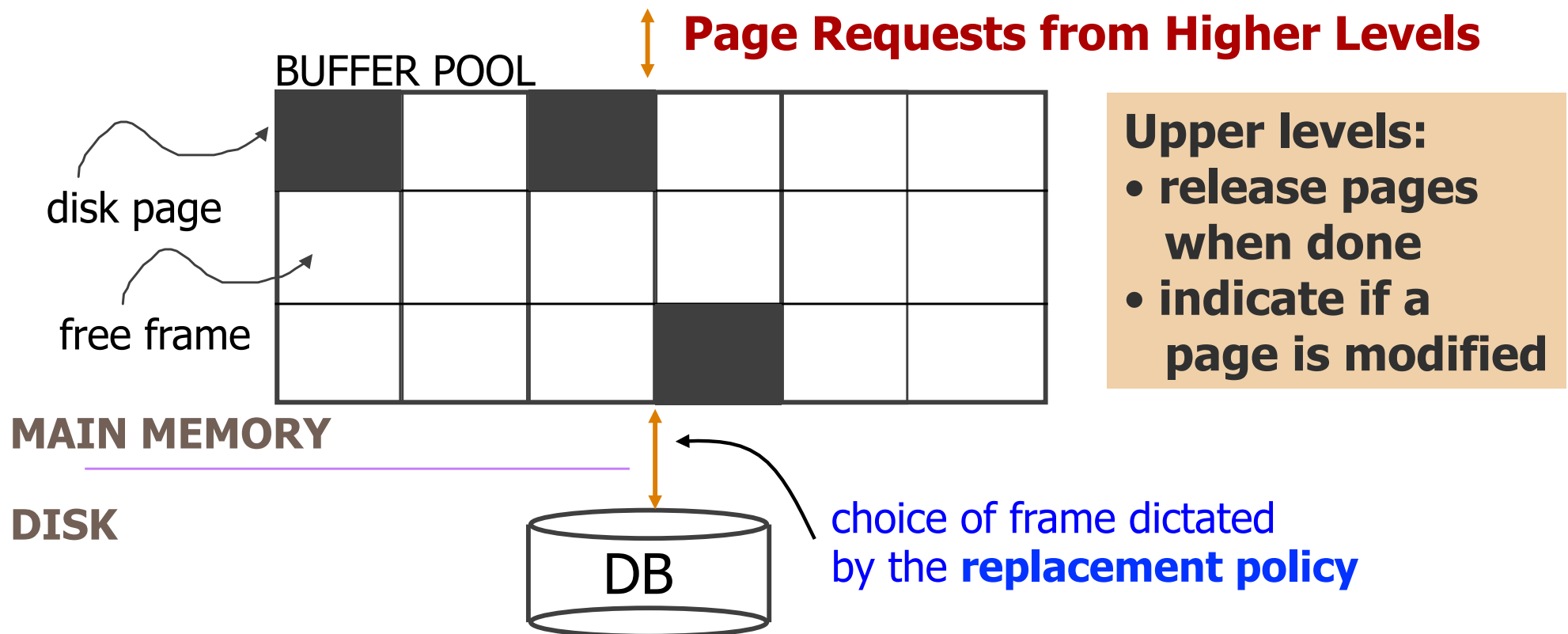
Either way, the raw disk space is organized into **files**.
Files are made up of **pages**, and pages contain **records**

Data is allocated and deallocated in increments of pages.

Logically “near” pages should be kept physically close

Buffer Management in a DBMS

- Data must be in RAM for DBMS to operate on it!
 - Can't keep all the DBMS pages in main memory
- Buffer Manager: Efficiently uses main memory
 - Memory divided into **buffer frames**: slots for holding disk pages



Buffer Manager

2 requestors want to modify the same page?

- Bookkeeping per frame:
 - *pin count* : # users of the page in the frame
 - *Pinning* : Indicate that the page is in use
 - *Unpinning* : Release the page, and also indicate if the page is *dirtied*
 - *dirty bit* : Indicates if changes must be propagated to disk
- When a Page is requested:
 - In buffer pool -> return a handle to the frame. Done!
 - Increment the pin count
 - Not in the buffer pool:
 - Choose a frame for *replacement*
(Only replace pages with pin count == 0)
 - If frame is dirty, write it to disk
 - Read requested page into chosen frame
 - Pin the page and return its address

Can you tell the # current users of a page in the BP?

Buffer Replacement Policy

- Chose a frame for replacement
 - Least-recently-used (LRU), Clock, MRU etc.
- LRU: queue of pointers to “empty” frames
 - Add to end of queue, grab frames from front of queue
- Clock: variant of LRU, but lower overhead
- 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

•

DBMS vs. OS File System

Why not let the OS handle disk space and buffer mgmt.?

- DBMS better at predicting the reference patterns
- Buffer management in DBMS requires ability to:
 - pin a page in buffer pool
 - force a page to disk (required to implement CC & recovery)
 - adjust *replacement policy*
 - pre-fetch pages based on predictable access patterns
 - Pages available when needed later
 - Amortize rotational and seek costs
- Can better control the overlap of I/O with computation
- DBMS can leverage multiple disks more effectively

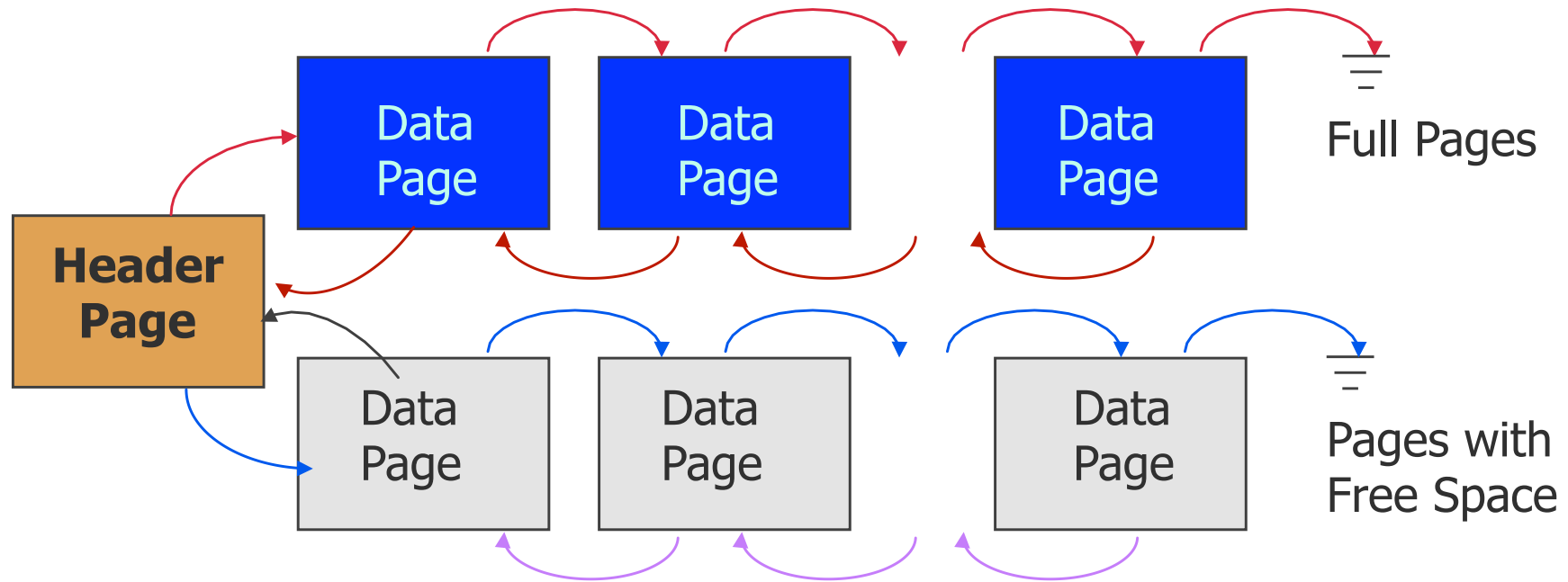
Files of Records

- Page or block is OK for I/O, but higher levels operate on *records*, and *files of records*.
- **File**: A collection of pages
Page: a collection of records.
- File operations:
 - insert/delete/modify record
 - read a particular record (specified using the *record id*)
 - scan all records (possibly with some conditions on the records to be retrieved)

Unordered (Heap) Files

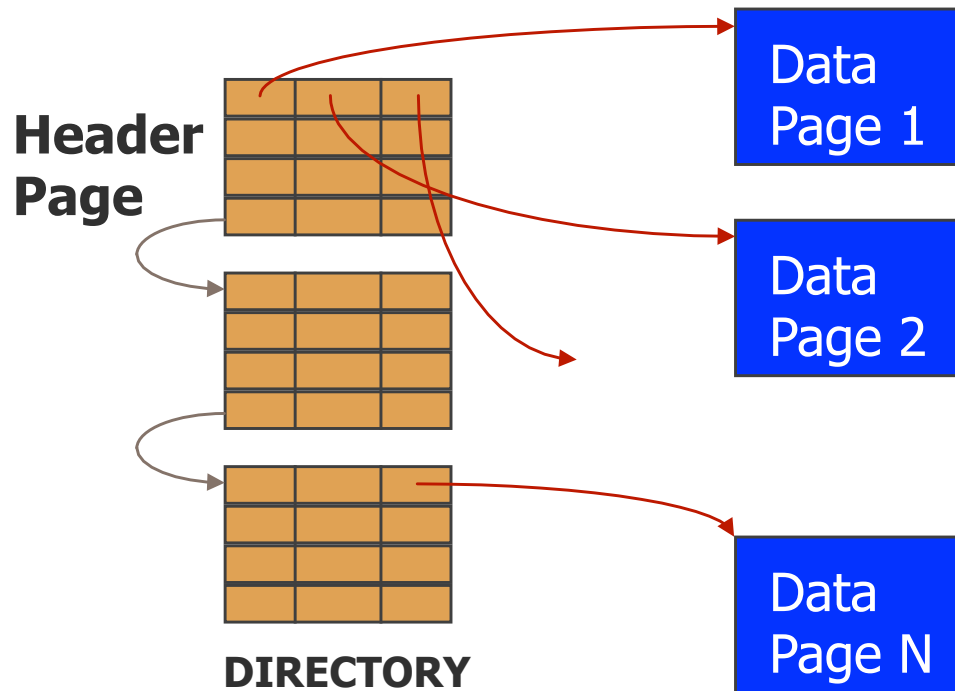
- Simplest file structure contains records in no particular order.
- As file grows and shrinks, disk pages are allocated and de-allocated.
- To support record level operations, we must:
 - keep track of the *pages* in a file: **page id (pid)**
 - keep track of *free space* on pages
 - keep track of the *records* on a page: **record id (rid)**
 - Many alternatives for keeping track of this information
- 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) recorded in a known location
- Each page contains two **pointers** plus data: Pointer = **Page ID (pid)**
- Pages in the free space list have “some” free space
- What happens with variable length records?
- Fetch a record with rid

Heap File Using a Page Directory

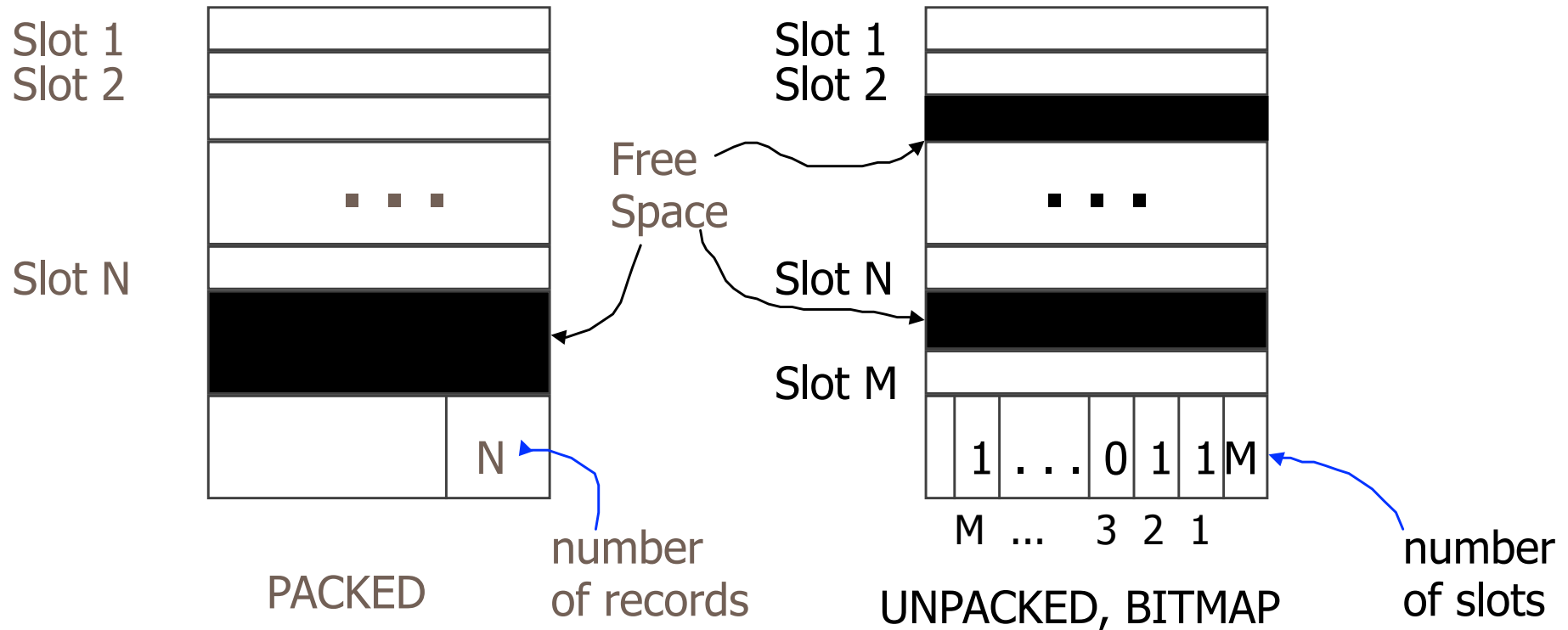


- Entry for a page:
 - Free/full
 - Number of free bytes
- Can locate pages for new tuples faster!

Page Formats

- File -> collection of pages
- Page -> collection of tuples/records
- Query operators deal with tuples
- Slotted page format
 - Page a collection of slots
 - Each slot contains a record
- RID: <page id, slot number>
- Other ways of generating rids
- Many slotted page organizations. Must support
 - Search, insert, delete records on a page

Page Formats: Fixed Length Records

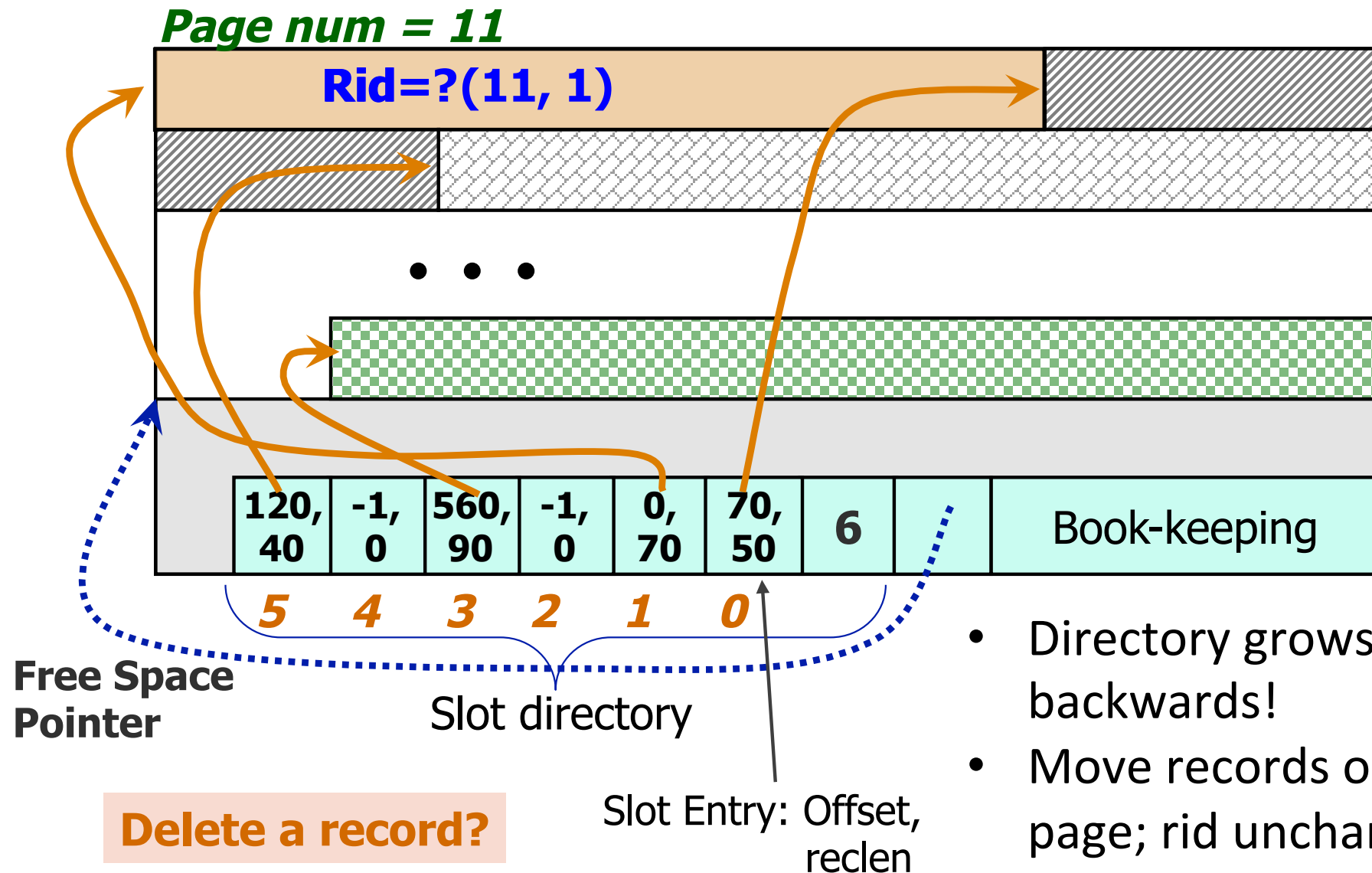


☐ Record id = <page id, slot #>

☐ First alternative: moving records changes rid

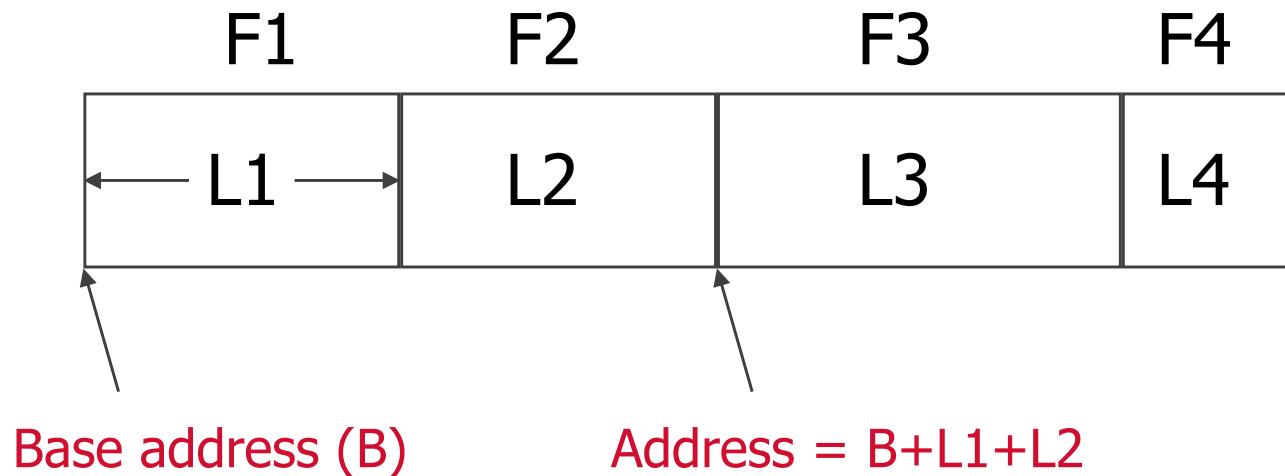
☐ may not be acceptable.

Page Formats: Variable Length Records



- Directory grows backwards!
- Move records on same page; rid unchanged! Good for fixed-length records too.

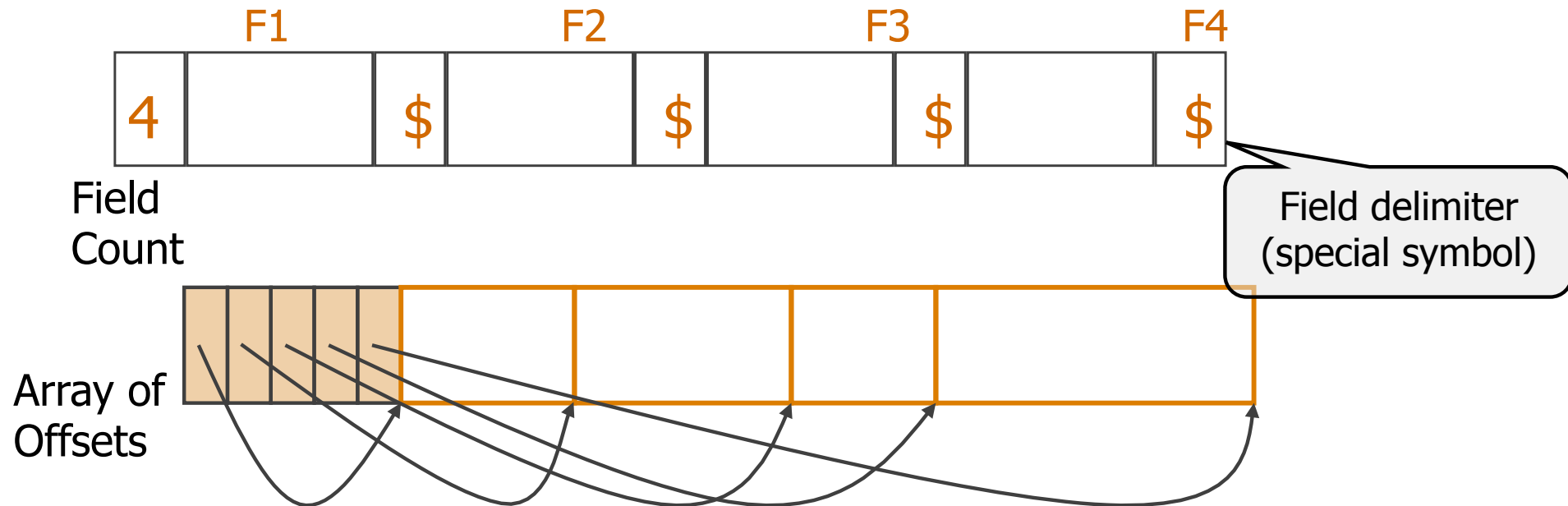
Record Formats: Fixed Length



- All records on the page are the same length
- Information about field types same for all records in a file; stored in *system catalogs*.

Record Formats: Variable Length

Two alternative formats (# fields is fixed):



- Second alternative offers direct access to i 'th field
 - Efficient storage of nulls
 - Small directory overhead.
- Issues with growing records!
 - changes in attribute value, add/drop attributes
- Records larger than pages

Column Stores: Motivation

- Consider a table:
 - Foo (a INTEGER, b INTEGER, c VARCHAR(255), ...)
- And the query:
 - SELECT a FROM Foo WHERE a > 10
- What happens with the previous record format in terms of the bytes that have to be read from the IO subsystem?

Column Stores

- Store data “vertically”
- Contrast that with a “row-store” that stores all the attributes of a tuple/record contiguously
 - The previous record formats are “row stores”

111	212	It was a cold morning
222	222	Warm and sunny here
333	232	Arctic winter conditions
444	242	Tropical weather



111	212	It was a cold morning
222	222	Warm and sunny here
333	232	Arctic winter conditions
444	242	Tropical weather



File 1



File 2



File 3

Column Stores

- Are there any disadvantages associated with column stores?