CSE 444: Database Internals

Lecture 4

Data storage and buffer management

Homework Logistics

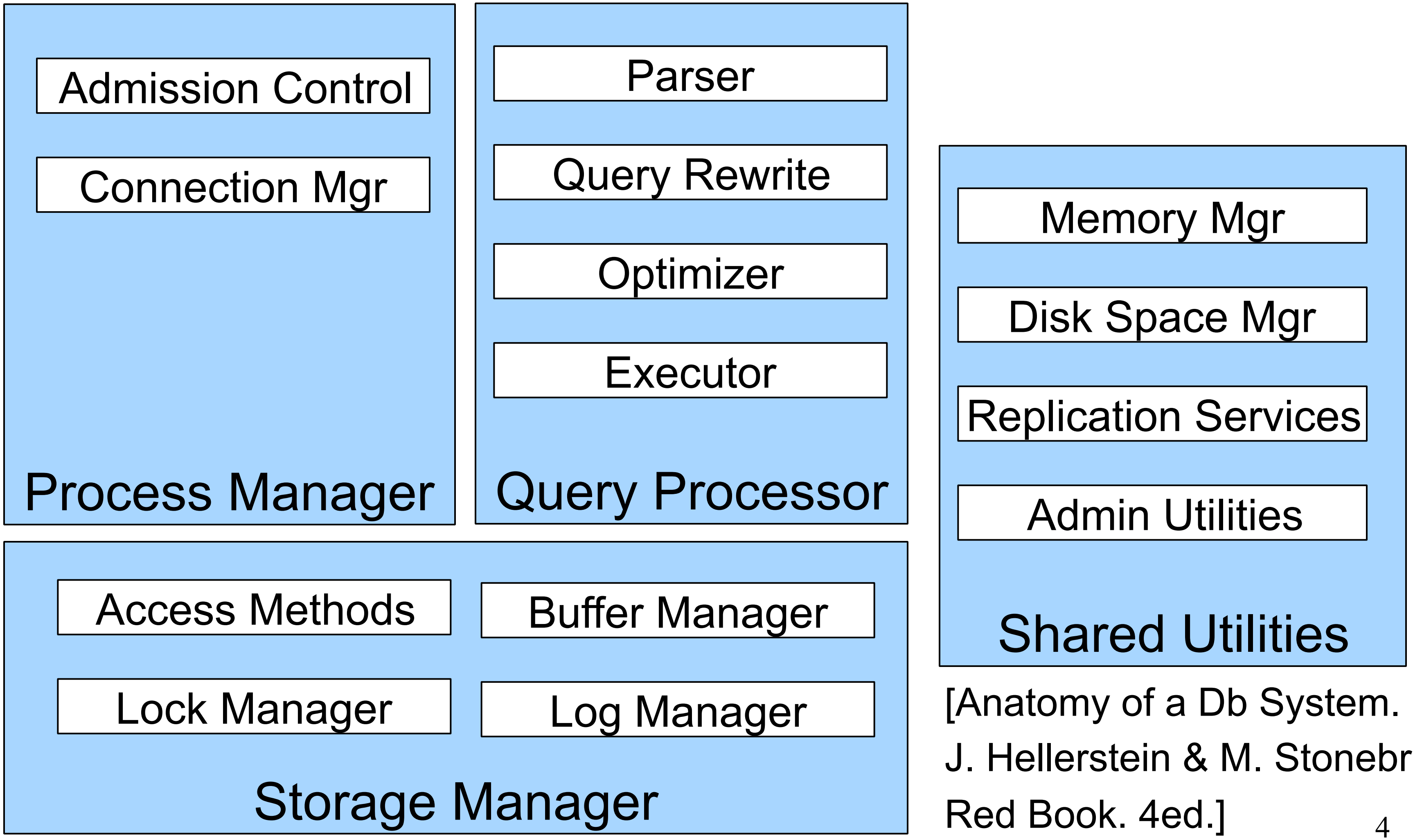
- Homework instructions are in a pdf file

- Submit a single pdf or word file with your solution, or
- Submit a hard copy by 4pm on Wednesday in my office 662

Important Note

- Lectures show principles
- You need to think through what you will actually implement in SimpleDB!
  - Try to implement the simplest solutions
- If you are confused, tell us!

DBMS Architecture



[Anatomy of a Db System.
J. Hellerstein & M. Stonebraker.
Red Book. 4ed.]

Today: Starting at the Bottom

Consider a relation storing tweets:

```
Tweets(tid, user, time, content)
```

How should we store it on disk?

Design Exercise

- Design choice: **One OS file for each relation**
  - This does not always have to be the case! (e.g., SQLite uses one file for whole database)
  - DBMSs can also use disk drives directly

- An OS file provides an API of the form
  - Seek to some position (or “skip” over B bytes)
  - Read/Write B bytes

First Principle: Work with Pages

- Reading/writing to/from disk
  - Seeking takes a long time!
  - Reading sequentially is fast

- To simplify buffer manager, want to cache a collection of same-sized objects

- Solution: Read/write **pages** of data
  - A page should correspond to a disk block

Continuing our Design

Next key questions:

- How do we organize pages into a file?
- How do we organize data within a page?

First, how could we store some tuples on a page?

Let’s first assume all tuples are of the same size

```
Tweets(tid int, user char(10),
       time int, content char(140))
```

Page Formats

Issues to consider

- 1 page = 1 disk block = fixed size (e.g. 8KB)
- Records:
  - Fixed length
  - Variable length
- Record id = RID
  - Typically RID = (PageID, SlotNumber)

Why do we need RID's in a relational DBMS ?

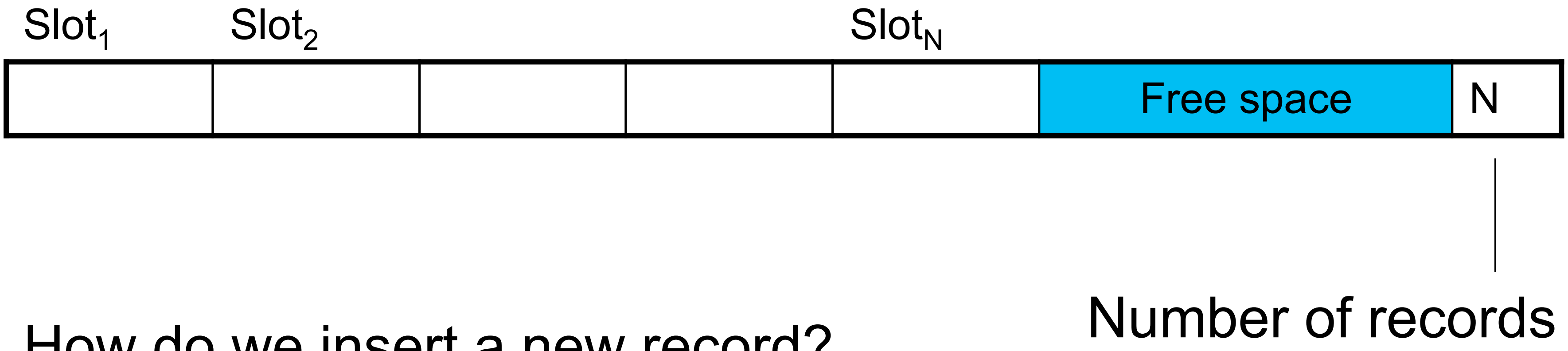
See future discussion on indexes and transactions

Page Format Approach 1

Fixed-length records: packed representation

Divide page into slots. Each slot can hold one tuple

Record ID (RID) for each tuple is (PageID,SlotNb)



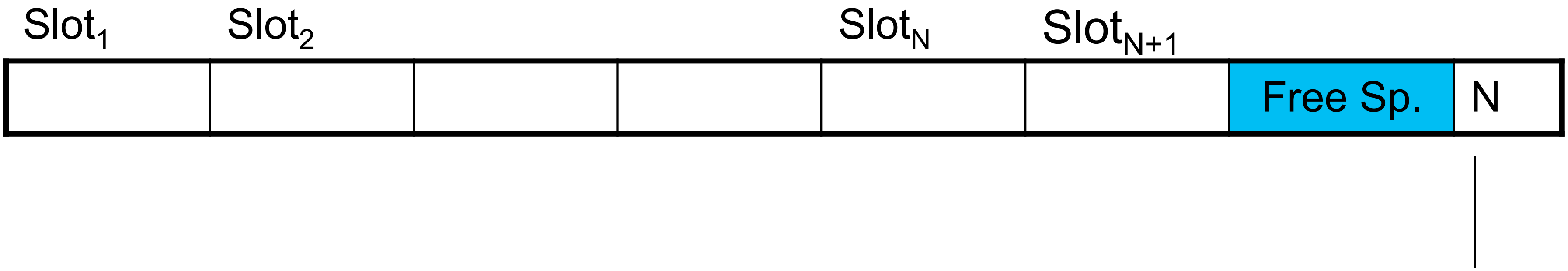
How do we insert a new record?

Page Format Approach 1

Fixed-length records: packed representation

Divide page into slots. Each slot can hold one tuple

Record ID (RID) for each tuple is (PageID,SlotNb)



Number of records

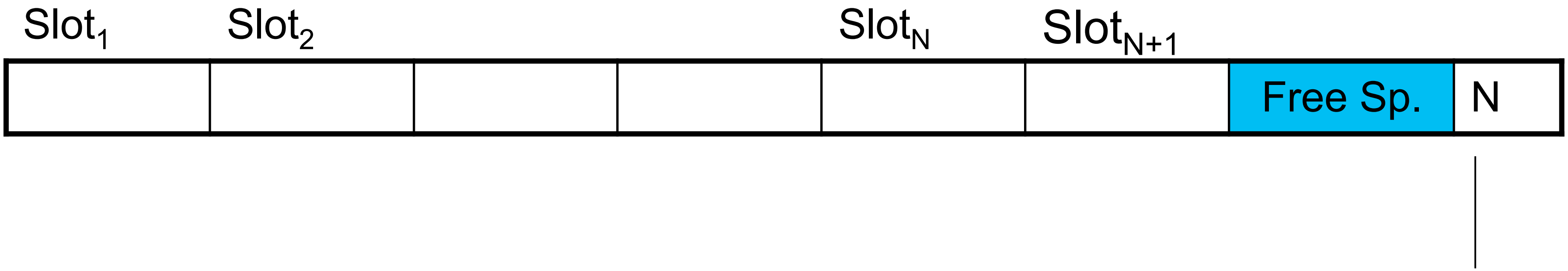
How do we insert a new record?

Page Format Approach 1

Fixed-length records: packed representation

Divide page into slots. Each slot can hold one tuple

Record ID (RID) for each tuple is (PageID,SlotNb)



Slot$_1$ Slot$_2$ Slot$_N$ Slot$_{N+1}$

Free Sp. N

Number of records

How do we insert a new record?

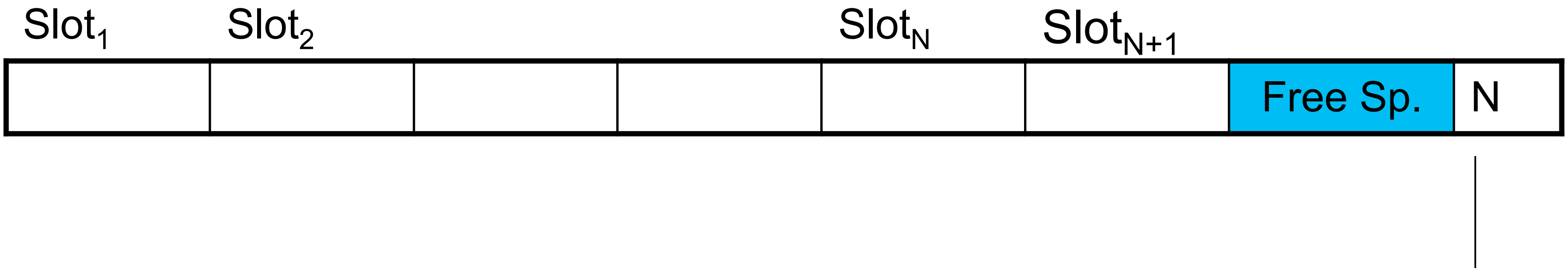
How do we delete a record?

Page Format Approach 1

Fixed-length records: packed representation

Divide page into slots. Each slot can hold one tuple

Record ID (RID) for each tuple is (PageID,SlotNb)



Number of records

How do we insert a new record?

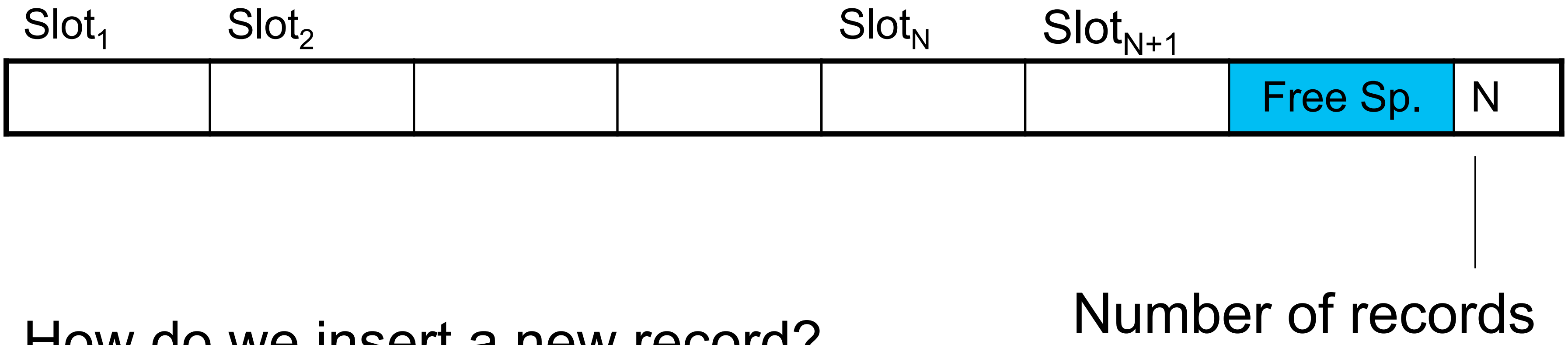
How do we delete a record? Cannot move records! (Why?)

Page Format Approach 1

Fixed-length records: packed representation

Divide page into slots. Each slot can hold one tuple

Record ID (RID) for each tuple is (PageID,SlotNb)

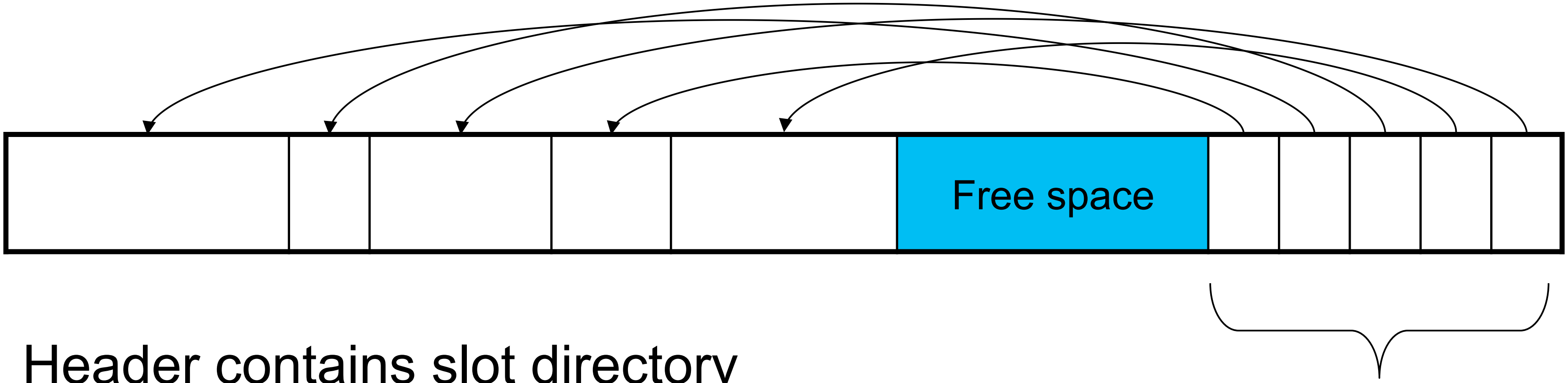


How do we insert a new record?

How do we delete a record? Cannot move records! (Why?)

How do we handle variable-length records?

Page Format Approach 2



Slot directory

Each slot contains
<record offset, record length>

Header contains slot directory

+ Need to keep track of nb of slots

+ Also need to keep track of free space

Can handle variable-length records

Can move tuples inside a page without changing RIDs

RID is (PageID, SlotID) combination

Record Formats

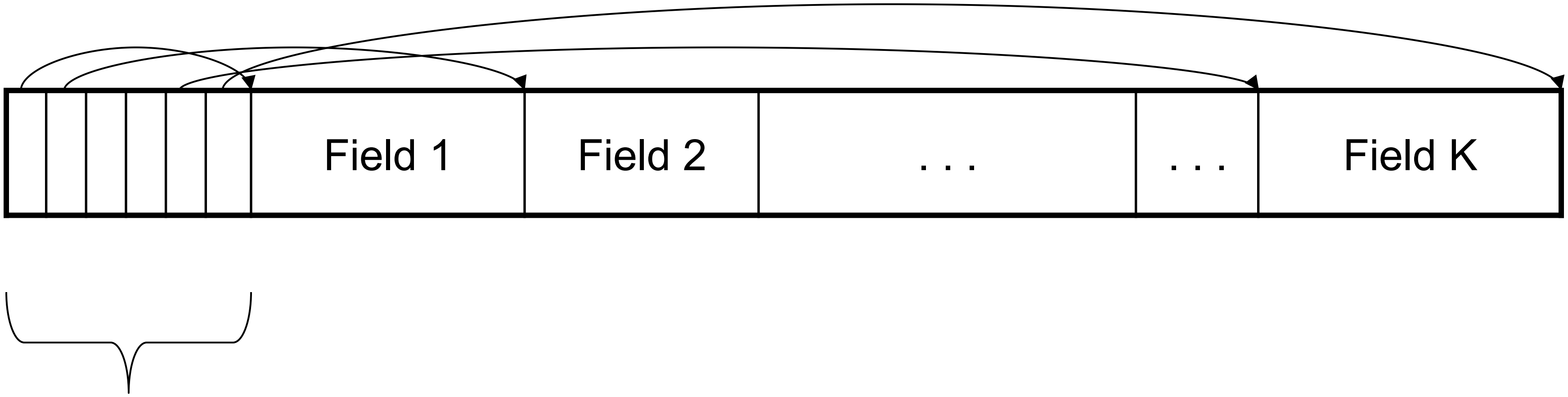
Fixed-length records → Each field has a fixed length (i.e., it has the same length in all the records)

Field 1	Field 2	. . .	. . .	Field K

Information about field lengths and types is in the catalog

Record Formats

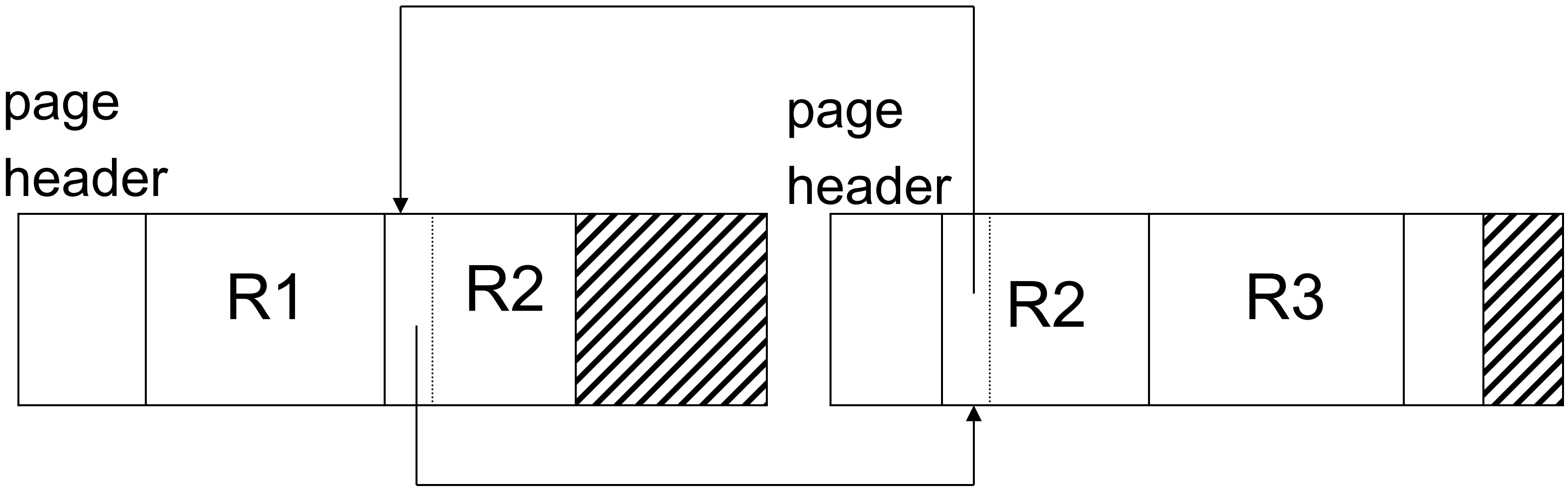
Variable length records



Record header

Remark: NULLS require no space at all (why ?)

Long Records Across Pages



- When records are very large
- Or even medium size: saves space in blocks
- Commercial RDBMSs avoid this

LOB

- Large objects
  - Binary large object: BLOB
  - Character large object: CLOB

- Supported by modern database systems
- E.g. images, sounds, texts, etc.

- Storage: attempt to cluster blocks together

Continuing our Design

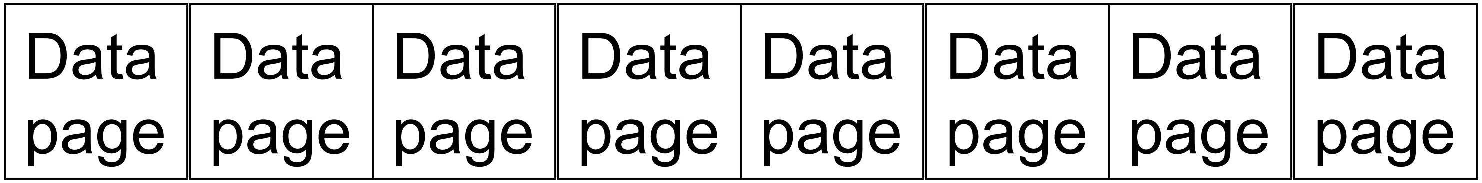
Next key questions:

- How do we organize pages into a file?
- How do we organize data within a page?

Now, how should we group pages into files?

Heap File Implementation 1

A sequence of pages (implementation in SimpleDB)



Data page | Data page | Data page | Data page | Data page | Data page | Data page | Data page

Some pages have space and other pages are full

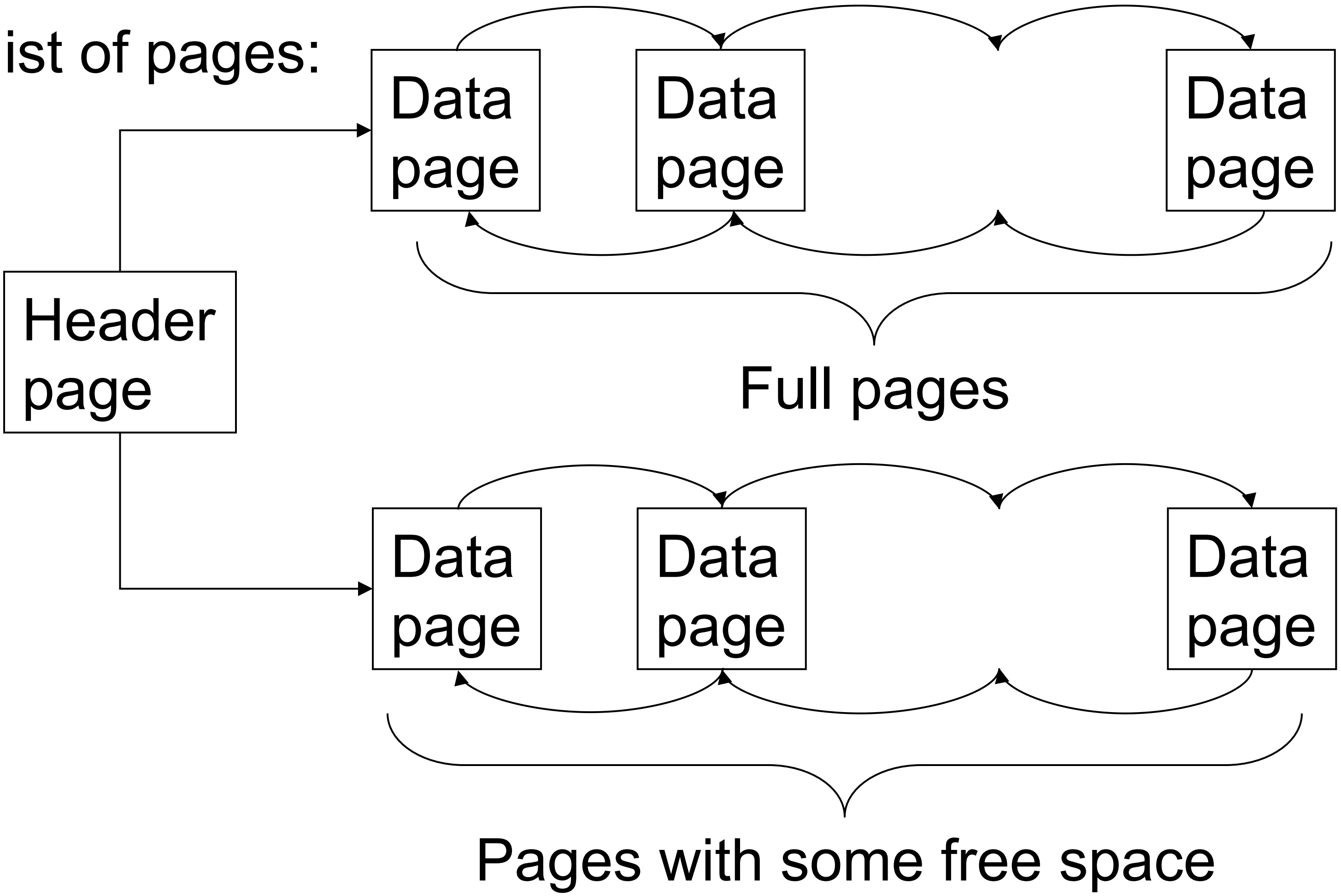
Add pages at the end when need more space

Works well for small files

But finding free space requires scanning the file

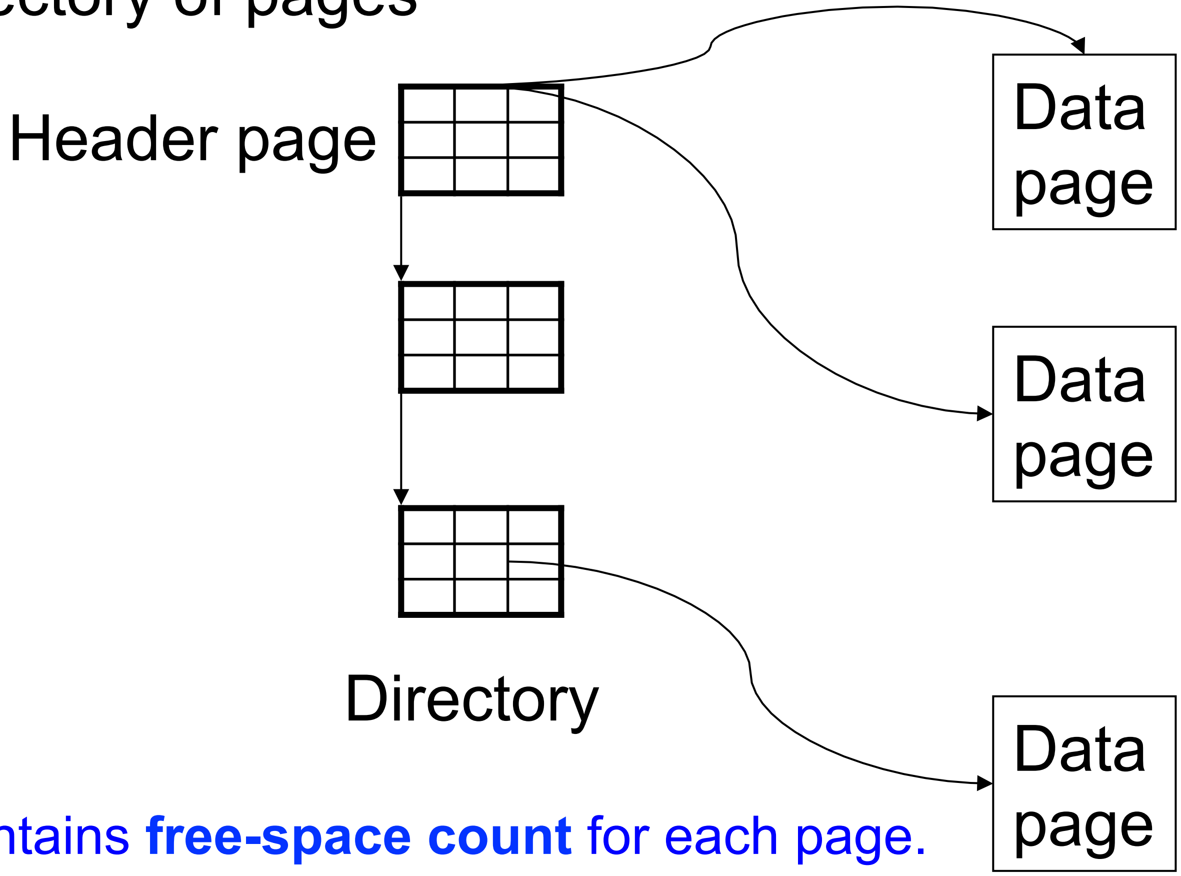
Heap File Implementation 2

Linked list of pages:



Heap File Implementation 3

Better: directory of pages



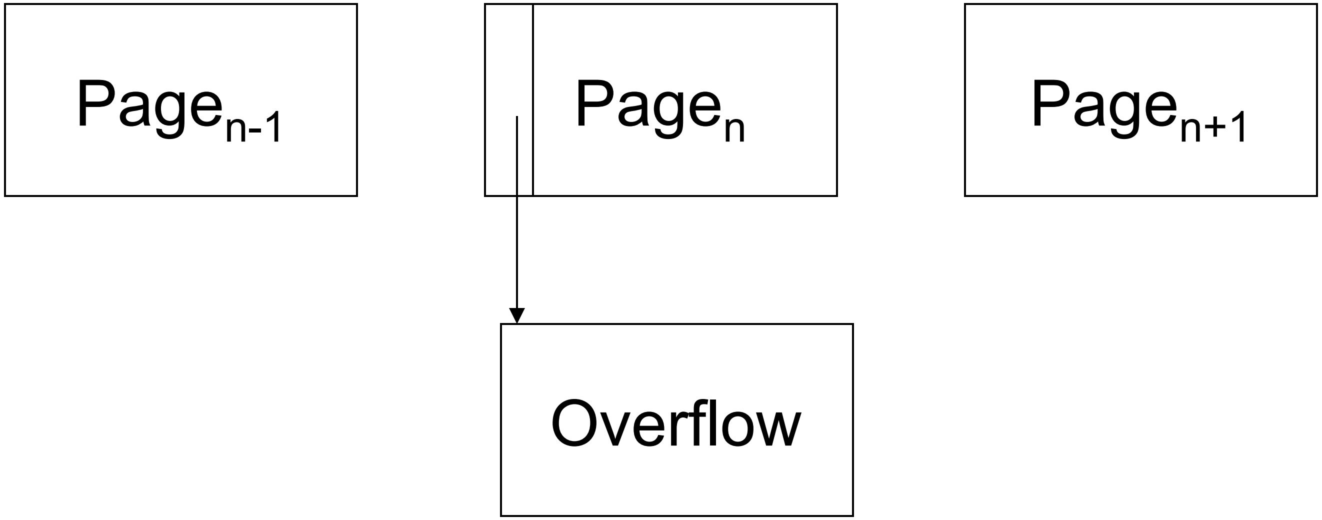
Directory contains **free-space count** for each page.

Faster inserts for variable-length records

Modifications: Insertion

- File is unsorted (= ***heap file***)
  - add it wherever there is space (easy ☺)
  - add more pages if out of space
- File is sorted
  - Is there space on the right page ?
    - Yes: we are lucky, store it there
  - Is there space in a neighboring page ?
    - Look 1-2 pages to the left/right, shift records
  - If anything else fails, create ***overflow page***

Overflow Pages



- After a while the file starts being dominated by overflow pages: time to reorganize

Modifications: Deletions

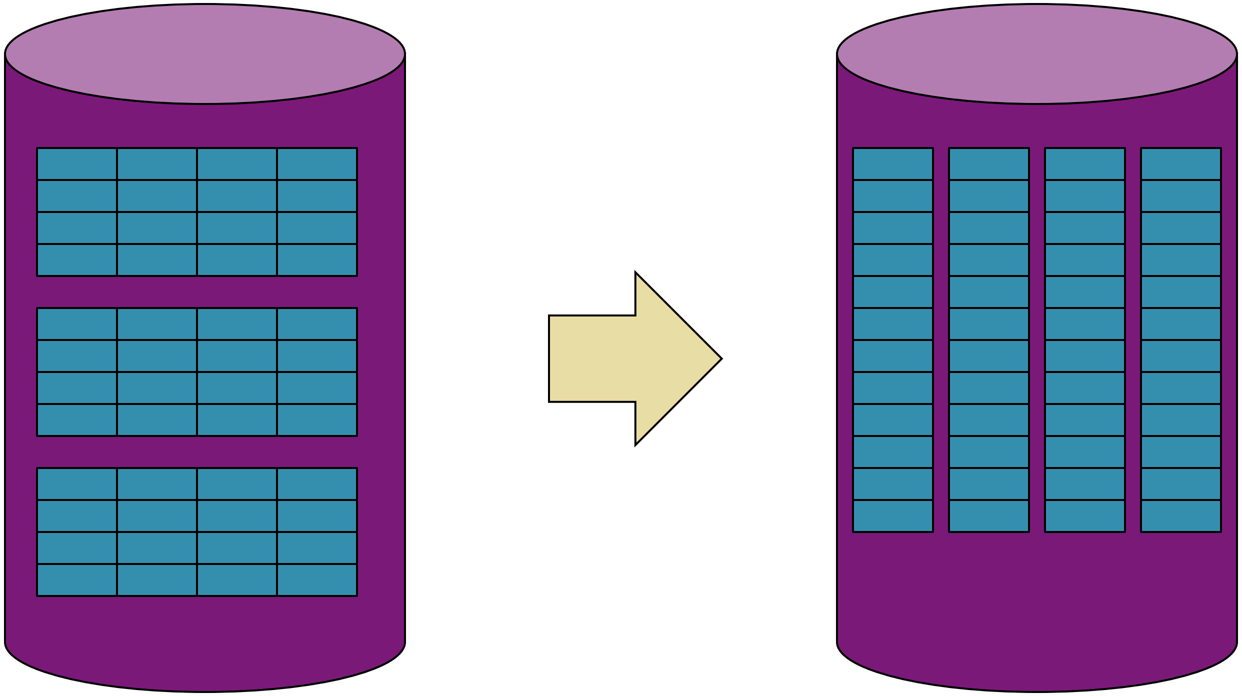
- Free space in page, shift records
  - Be careful with slots
  - RIDs for remaining tuples must NOT change

- May be able to eliminate an overflow page

Modifications: Updates

- If new record is shorter than previous, easy ☺
- If it is longer, need to shift records
  - May have to create overflow pages

Alternate Storage Manager Design: Column Store



Rows stored contiguously on disk (+ tuples headers)

Columns stored contiguously on disk (no headers needed)

More Detailed Example

Row-based
(4 pages)

Page

A	1
A	2

A	2
A	2

B	2
B	4

C	4
C	4

Column-based
(4 pages)

A
A
A
A

1
2
2
2

B
B
C
C

2
4
4
4

Page

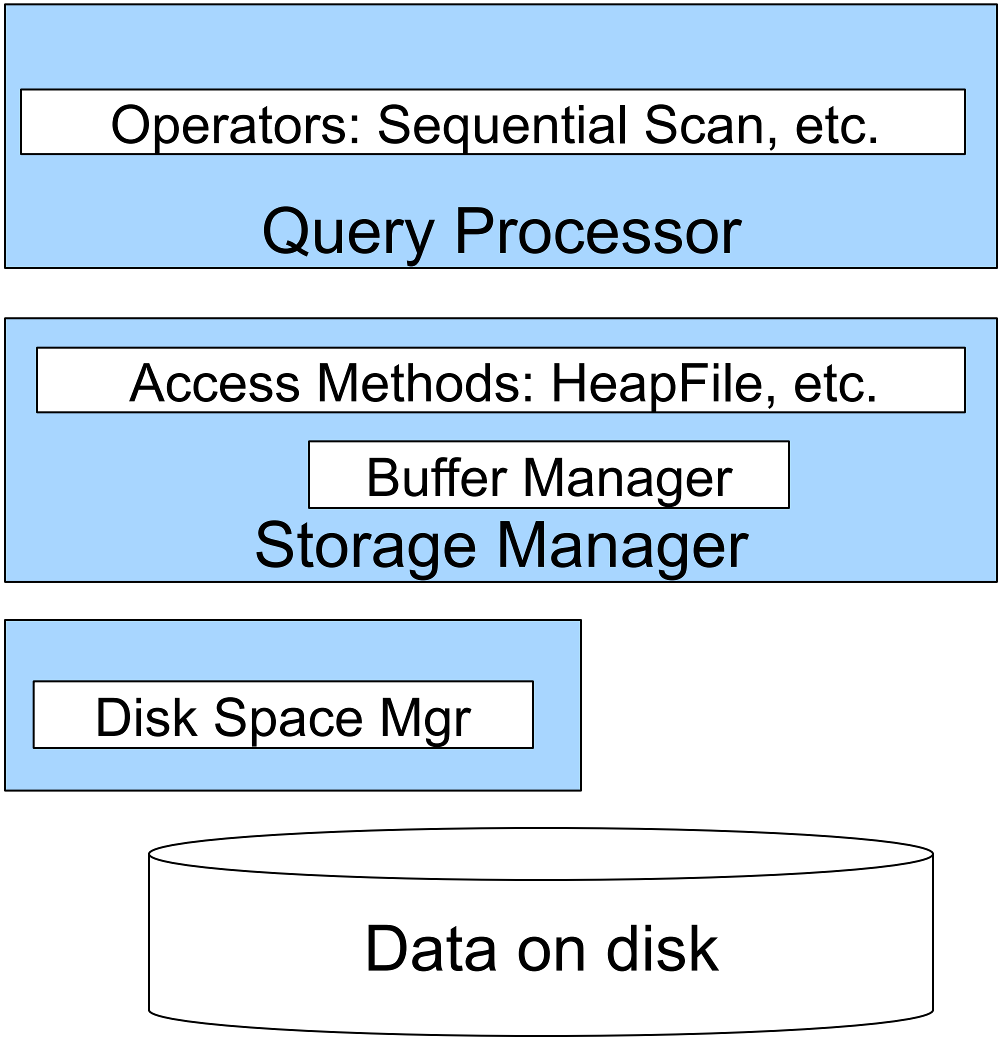
C-Store also avoids large tuple headers

Continuing our Design

We know how to store tuples on disk in a heap file

How do these files interact with rest of engine?

How Components Fit Together



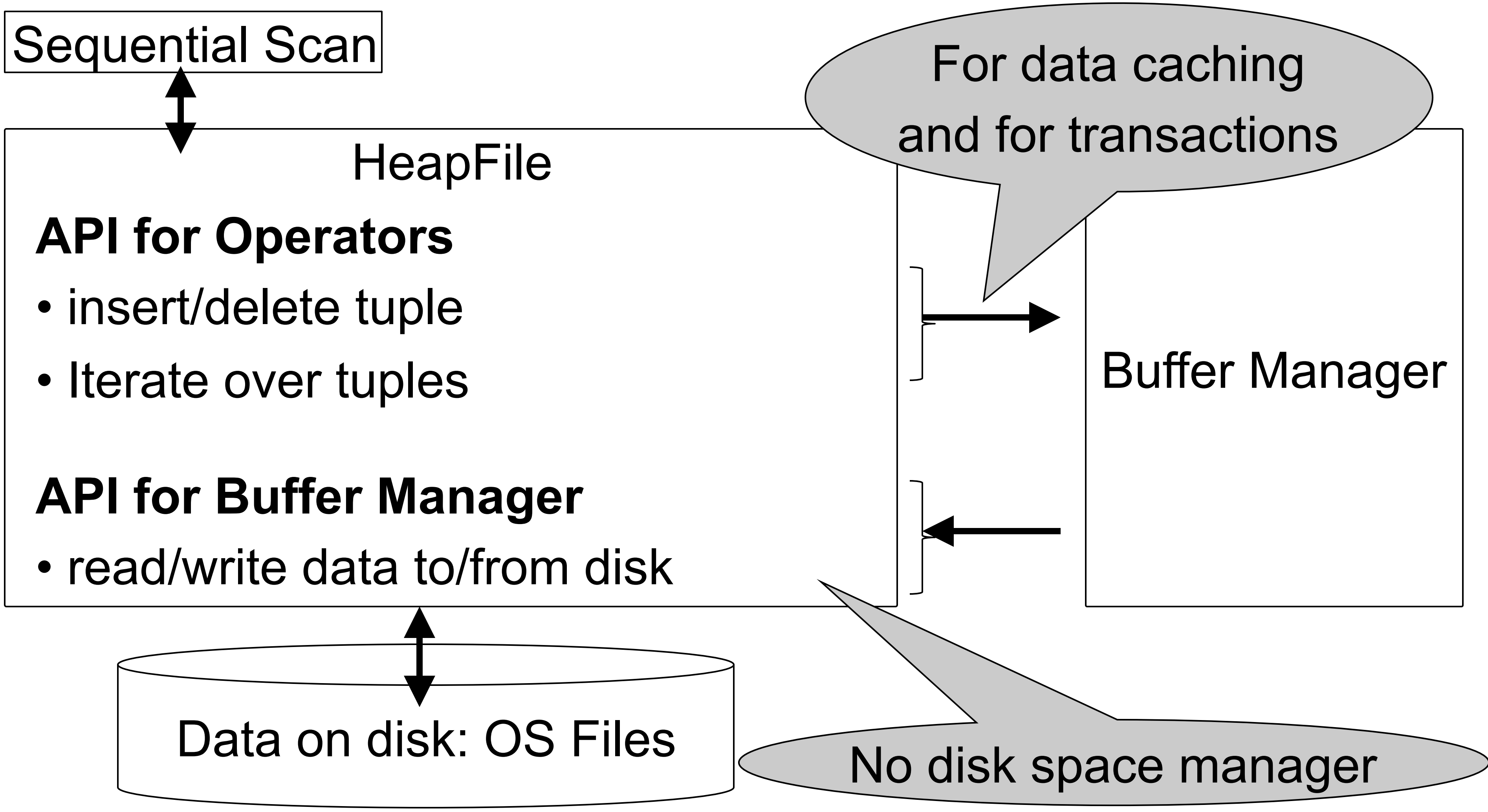
- **Operators:** Process data
- **Access methods**: Organize data to support fast access to desired subsets of records
- **Buffer manager**: Caches data in memory. Reads/ writes data to/from disk as needed
- **Disk-space manager**: Allocates space on disk for files/access methods

Access Methods

- Operators view relations as collections of records

- The access methods worry about how to organize these collections

HeapFile In SimpleDB



Sequential Scan

HeapFile

API for Operators

- insert/delete tuple
- Iterate over tuples

API for Buffer Manager

- read/write data to/from disk

Buffer Manager

For data caching and for transactions

Data on disk: OS Files

No disk space manager

HeapFile In SimpleDB

- Data is stored on disk in an OS file. HeapFile class knows how to “decode” its content
- Control flow:
  - SeqScan calls methods such as "iterate" on the DbFile Access Method
  - During the iteration, the DbFile object needs to call the BufferManager.getPage method to ensure that necessary pages get loaded into memory.
  - The BufferManager will then call DbFile.read/write page to actually read/write the page.

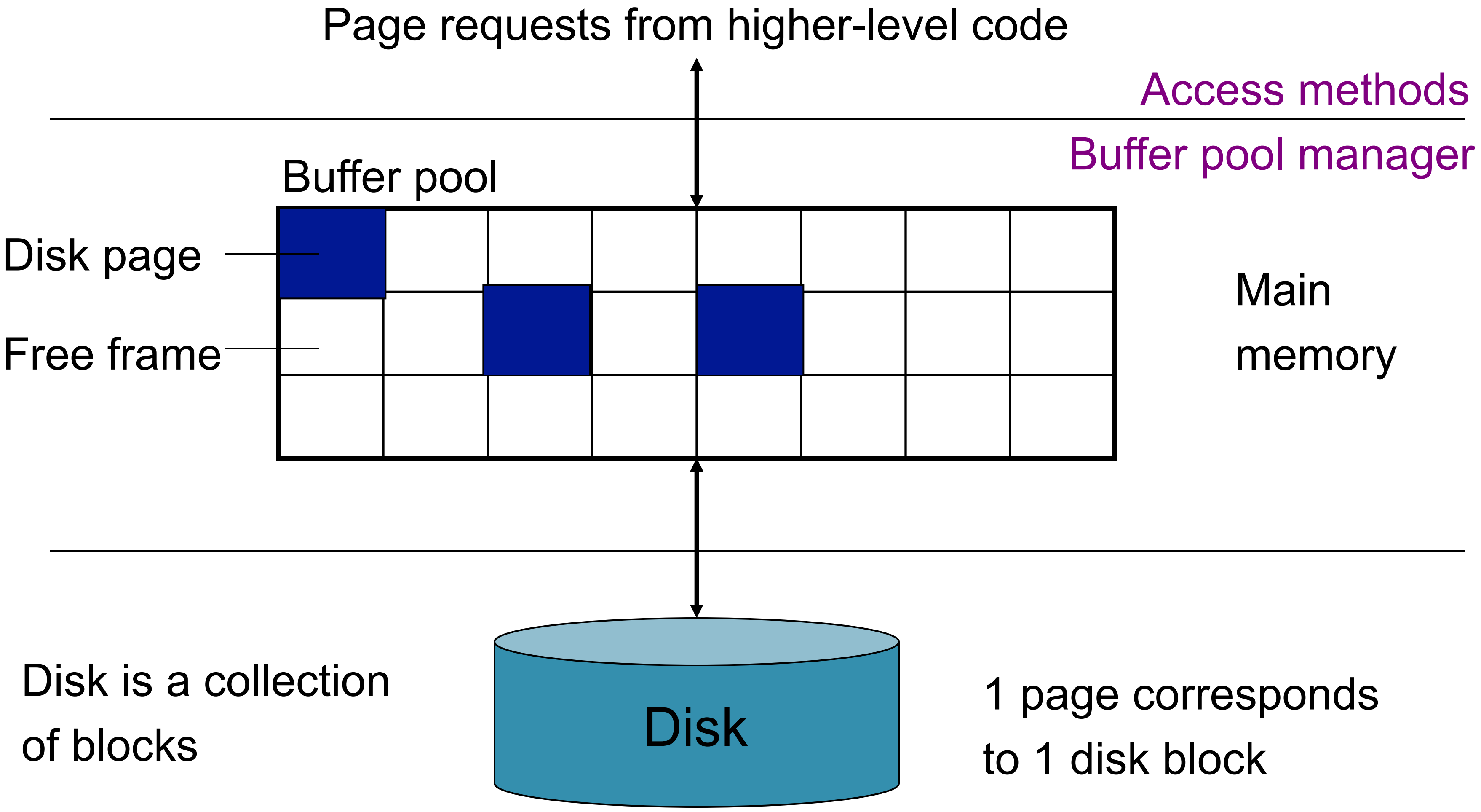
Heap File Access Method API

- **Create** or **destroy** a file
- **Insert** a record
- **Delete** a record with a given rid (rid)
  - rid: unique tuple identifier (more later)
- **Get** a record with a given rid
  - Not necessary for sequential scan operator
  - But used with indexes (more next lecture)
- **Scan** all records in the file

Pushing Updates to Disk

- When inserting a tuple, HeapFile inserts it on a page but does not write the page to disk
- When deleting a tuple, HeapFile deletes tuple form a page but does not write the page to disk
- The buffer manager worries when to write pages to disk (and when to read them from disk)
- When need to add a new page to the file, HeapFile adds page to the file on disk and then gets it again through the buffer manager

Buffer Manager


Page requests from higher-level code
Access methods
Buffer pool manager
Buffer pool
Disk page
Free frame
Main memory
Disk
Disk is a collection of blocks
1 page corresponds to 1 disk block

Buffer Manager

- Brings pages in from memory and caches them
- Eviction policies
  - Random page (ok for SimpleDB)
  - Least-recently used
  - The “clock” algorithm (see whiteboard or book)
- Keeps track of which **pages are dirty**
  - A dirty page has changes not reflected on disk
  - Implementation: Each page includes a dirty bit

Conclusion

- Row-store storage managers are most commonly used today
- They offer high-performance for transactions
- But column-stores win for analytical workloads
- They are gaining traction in that area

- Final discussion: OS vs DBMS
  - OS files vs DBMS files
  - OS buffer manager vs DBMS buffer manager