# CS186 Discussion 4

(Buffer Management, Heap Files, File Organization)

Matthew Deng

# Buffer Management

# Buffer Management

- Buffer pool
  - In-memory cache for database tables
- Not all data can fit in memory at once
- Which data should we keep?
  - Buffer replacement policies
    - Least Recently Used (LRU)
    - Most Recently Used (MRU)
    - Clock

# Least Recently Used (LRU)

- Evict page that has been least recently used
- Must keep track of when pages were used
- Prone to sequential flooding
- Example: A, B, C, D, E, A, B, C, D

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | | | | | |
| **2** | | B | | | | | | | |
| **3** | | | C | | | | | | |
| **4** | | | | D | | | | | |

Hit Rate:

# Least Recently Used (LRU)

- Evict page that has been least recently used
- Must keep track of when pages were used
- Prone to sequential flooding
- Example: A, B, C, D, E, A, B, C, D

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | E | | | | D |
| **2** | | B | | | | A | | | |
| **3** | | | C | | | | B | | |
| **4** | | | | D | | | | C | |

Hit Rate: 0/9

# Most Recently (LRU)

- Evict page that has been most recently used
- Must keep track of when pages were used
- Solves sequential flooding
- Example: A, B, C, D, E, A, B, C, D

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | | | | | |
| **2** | | B | | | | | | | |
| **3** | | | C | | | | | | |
| **4** | | | | D | | | | | |

Hit Rate:

# Most Recently (LRU)

- Evict page that has been most recently used
- Must keep track of when pages were used
- Solves sequential flooding
- Example: A, B, C, D, E, A, B, C, D

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | | * | | | |
| **2** | | B | | | | | * | | |
| **3** | | | C | | | | | * | D |
| **4** | | | | D | E | | | | |

Hit Rate: 3/9

# Clock

- Approximation for LRU
- Use reference bits
- Each page gets a second chance
- Example: A, B, C, D, E, A, B, C, D

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | | | | | |
| **2** | | B | | | | | | | |
| **3** | | | C | | | | | | |
| **4** | | | | D | | | | | |

Hit Rate:

# Clock

```
if HIT:
   reference bit = 1
else:
   while reference bit != 0:
      reference bit = 0
      move hand
   replace page
   reference bit = 1
   move hand
```

# Clock

- Approximation for LRU
- Use reference bits
- Each page gets a second chance
- Example: A, B, C, D, E, A, B, C, D

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | E | | | | D |
| **2** | | B | | | | A | | | |
| **3** | | | C | | | | B | | |
| **4** | | | | D | | | | C | |

Hit Rate: 0/9

# Buffer Replacement Worksheet

# Buffer Replacement Exercises

Fill in the following tables for the given buffer replacement policies. You have 4 buffer pages, with the access pattern
A B C D A F A D G D G E D F

1. Least Recently Used (LRU)

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | | | | | | | | | |
| **2** | | B | | | | | | | | | | | |
| **3** | | | C | | | | | | | | | | |
| **4** | | | | D | | | | | | | | | |

Hit Rate:

# Buffer Replacement Exercises

Fill in the following tables for the given buffer replacement policies. You have 4 buffer pages, with the access pattern
A B C D A F A D G D G E D F

1. Least Recently Used (LRU)

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | * | | * | | | | | | | F |
| **2** | | B | | | | F | | | | | | E | | |
| **3** | | | C | | | | | | G | | * | | | |
| **4** | | | | D | | | | * | | * | | | * | |

Hit Rate: 6/14

# Buffer Replacement Exercises

Fill in the following tables for the given buffer replacement policies. You have 4 buffer pages, with the access pattern
A B C D A F A D G D G E D F

2. Most Recently Used (MRU)

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | | | | | | | | | |
| **2** | | B | | | | | | | | | | | |
| **3** | | | C | | | | | | | | | | |
| **4** | | | | D | | | | | | | | | |

Hit Rate:

# Buffer Replacement Exercises

Fill in the following tables for the given buffer replacement policies. You have 4 buffer pages, with the access pattern
A B C D A F A D G D G E D F

2. Most Recently Used (MRU)

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | * | F | A | | | | | | | |
| **2** | | B | | | | | | | | | | | | |
| **3** | | | C | | | | | | | | | | | |
| **4** | | | | D | | | | * | G | D | G | E | D | F |

Hit Rate: 2/14

# Buffer Replacement Exercises

Fill in the following tables for the given buffer replacement policies. You have 4 buffer pages, with the access pattern
A B C D A F A D G D G E D F

3. Clock

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | | | | | | | | | |
| **2** | | B | | | | | | | | | | | |
| **3** | | | C | | | | | | | | | | |
| **4** | | | | D | | | | | | | | | |

Hit Rate:

# Buffer Replacement Exercises

Fill in the following tables for the given buffer replacement policies. You have 4 buffer pages, with the access pattern
A B C D A F A D G D G E D F

3. Clock

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | A | | | | * | F | | | | | | D | |
| **2** | | B | | | | | A | | | | | | F |
| **3** | | | C | | | | | | G | | * | | |
| **4** | | | | D | | | | * | | * | | E | | |

Hit Rate: 4/14

# Buffer Replacement Exercises

When is LRU the worst possible replacement policy?

# Buffer Replacement Exercises

When is LRU the worst possible replacement policy?

Sequential Scans – Flooding

# Buffer Replacement Exercises

Why would we use a clock replacement policy over LRU?

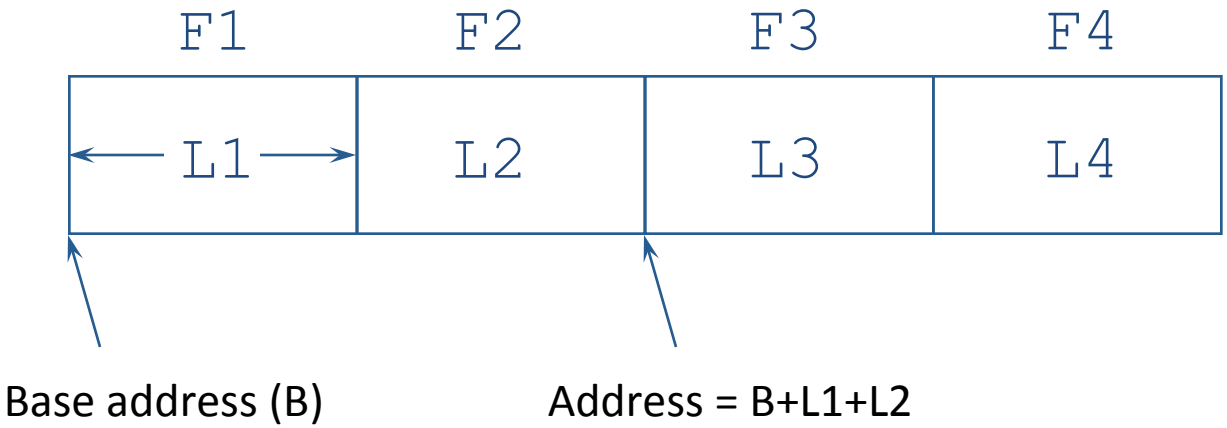
# Buffer Replacement Exercises

Why would we use a clock replacement policy over LRU?

Efficiency (Clock approximates LRU)

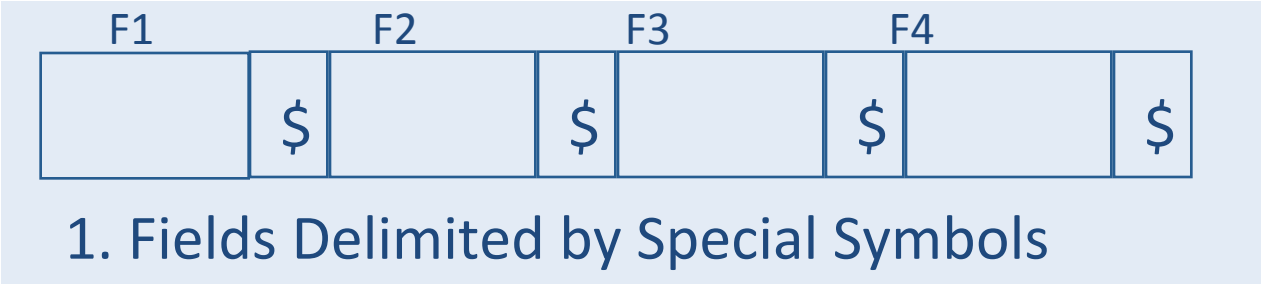
# Heap Files

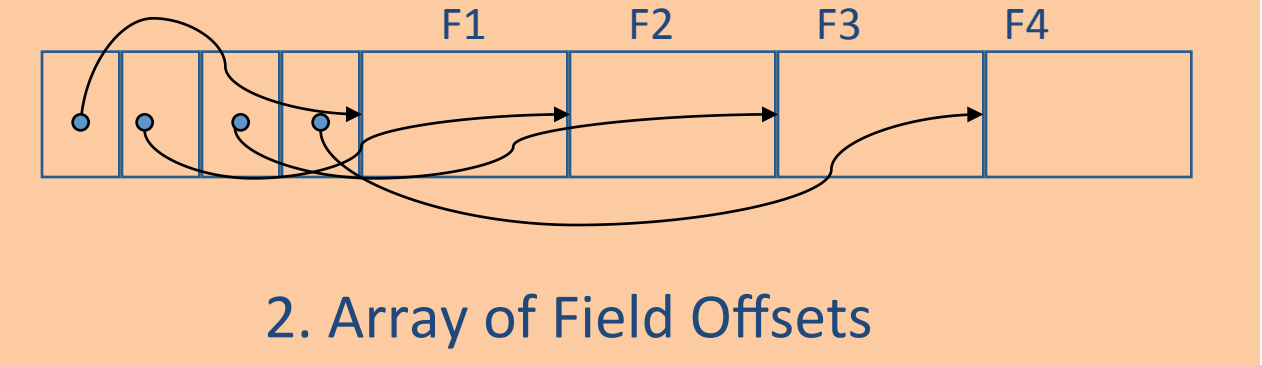
# Record Formats: Fixed Length




use this when you can!

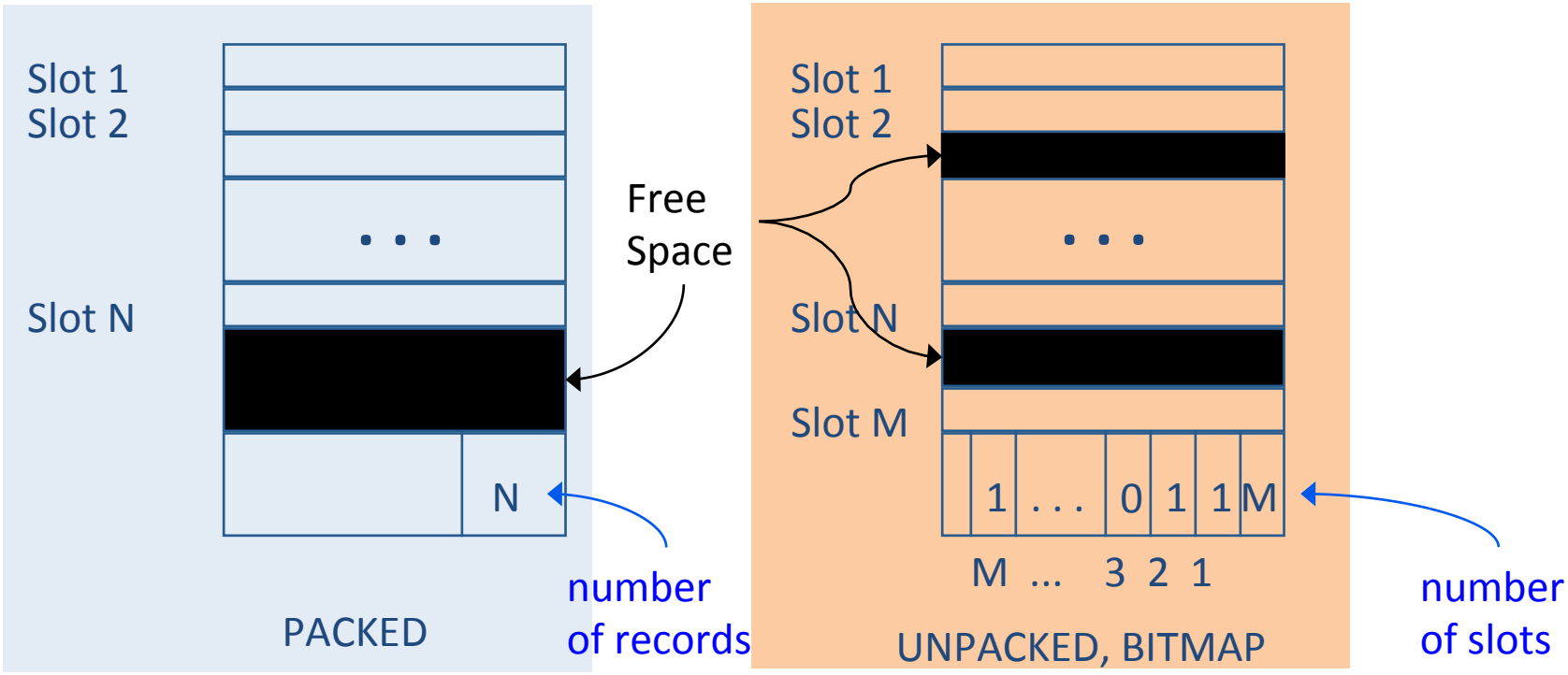
# Record Formats: Variable Length




symbols can cause problems

# Page Formats: Fixed Length

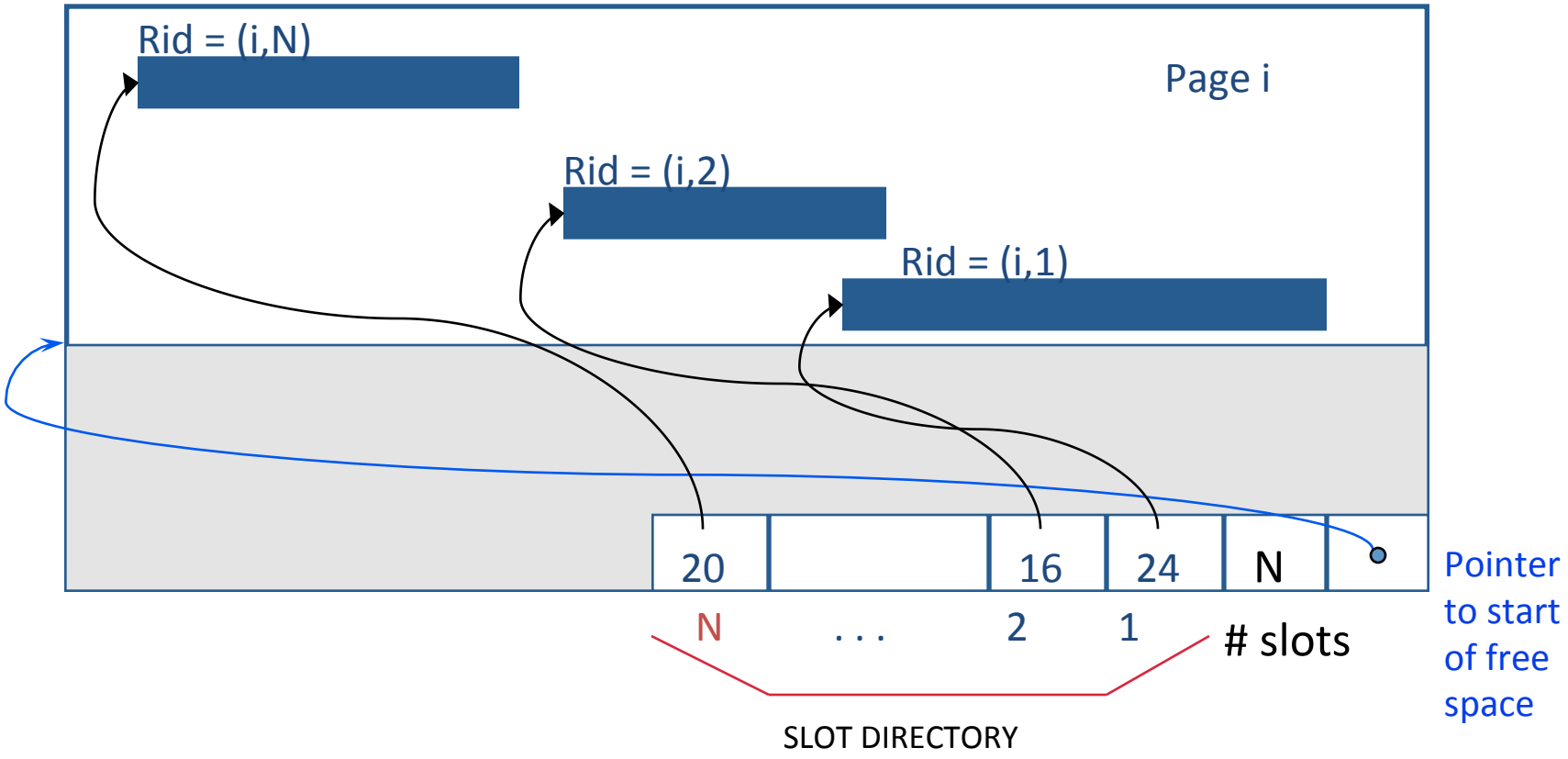



record id = <page id, slot number>

moving records in packed changes rid

free space in unpacked spread out

# Page Formats: Variable Length




can move records without changing rid

# Heap Files Worksheet

# Heap Files Exercises

1. What can we do to support variable length records (over fixed length records)?

# Heap Files Exercises

1. What can we do to support variable length records (over fixed length records)?

   Delimit with special symbols

   or use an array of field offsets

# Heap Files Exercises

2. What are the advantages and disadvantages of using slotted pages or bitmaps versus tightly packing records together?

# Heap Files Exercises

2. What are the advantages and disadvantages of using slotted pages or bitmaps versus tightly packing records together?

advantages: can move records without changing rid

supports variable length records

disadvantages: needs a page directory

# Heap Files Exercises

3. You have a slotted page with 80 bytes of free space, and it costs 4 bytes to store a directory entry.

   a. What is the size of the largest record you can insert?

# Heap Files Exercises

3. You have a slotted page with 80 bytes of free space, and it costs 4 bytes to store a directory entry.

   a. What is the size of the largest record you can insert?

      80 – 4 = 76 bytes

# Heap Files Exercises

3. You have a slotted page with 80 bytes of free space, and it costs 4 bytes to store a directory entry.

   b. At most, how many 1 byte large records can you insert?

# Heap Files Exercises

3. You have a slotted page with 80 bytes of free space, and it costs 4 bytes to store a directory entry.

   b. At most, how many 1 byte large records can you insert?

   80 / (4 + 1) = 16 records

# File Organization

# File Organization

| | Heap File | Sorted File |
|---|---|---|
| Scan All | | |
| Equality Search | | |
| Range Search | | |
| Insertion | | |
| Deletion | | |

# File Organization

| | Heap File | Sorted File |
|---|---|---|
| Scan All | B | B |
| Equality Search | .5B | $\log_2(B)$ |
| Range Search | B | $\log_2(B)$ + num_matches |
| Insertion | 2 | $\log_2(B)$ + 2 * .5B |
| Deletion | .5B + 1 | $\log_2(B)$ + 2 * .5B |

# File Organization Worksheet

# File Organization Exercises

Consider the table Enrolled(sid, course, grade) with 6,000 tuples spread evenly over 500 pages, and the query `SELECT * FROM Enrolled WHERE sid > 4500`.

a. How many I/Os would this query take if the table was stored in a heap file?

# File Organization Exercises

Consider the table Enrolled(sid, course, grade) with 6,000 tuples spread evenly over 500 pages, and the query `SELECT * FROM Enrolled WHERE sid > 4500`.

a. How many I/Os would this query take if the table was stored in a heap file?

I/Os = B = 500

# File Organization Exercises

Consider the table Enrolled(sid, course, grade) with 6,000 tuples spread evenly over 500 pages, and the query `SELECT * FROM Enrolled WHERE sid > 4500`.

b. If the table was stored in a file sorted on grade?

# File Organization Exercises

Consider the table Enrolled(sid, course, grade) with 6,000 tuples spread evenly over 500 pages, and the query `SELECT * FROM Enrolled WHERE sid > 4500`.

b. If the table was stored in a file sorted on grade?

I/Os = B = 500

# File Organization Exercises

Consider the table Enrolled(sid, course, grade) with 6,000 tuples spread evenly over 500 pages, and the query `SELECT * FROM Enrolled WHERE sid > 4500`.

c. If the table was stored in a file sorted on sid?

# File Organization Exercises

Consider the table Enrolled(sid, course, grade) with 6,000 tuples spread evenly over 500 pages, and the query `SELECT * FROM Enrolled WHERE sid > 4500`.

c. If the table was stored in a file sorted on sid?

I/Os = $\log_2(B)$ + [# matching pages]
= $\log_2(500)$ + 500/4
= 134

Or… I/OS = B = 125