CS 186 Section 4: Buffers, Files, and Indexes

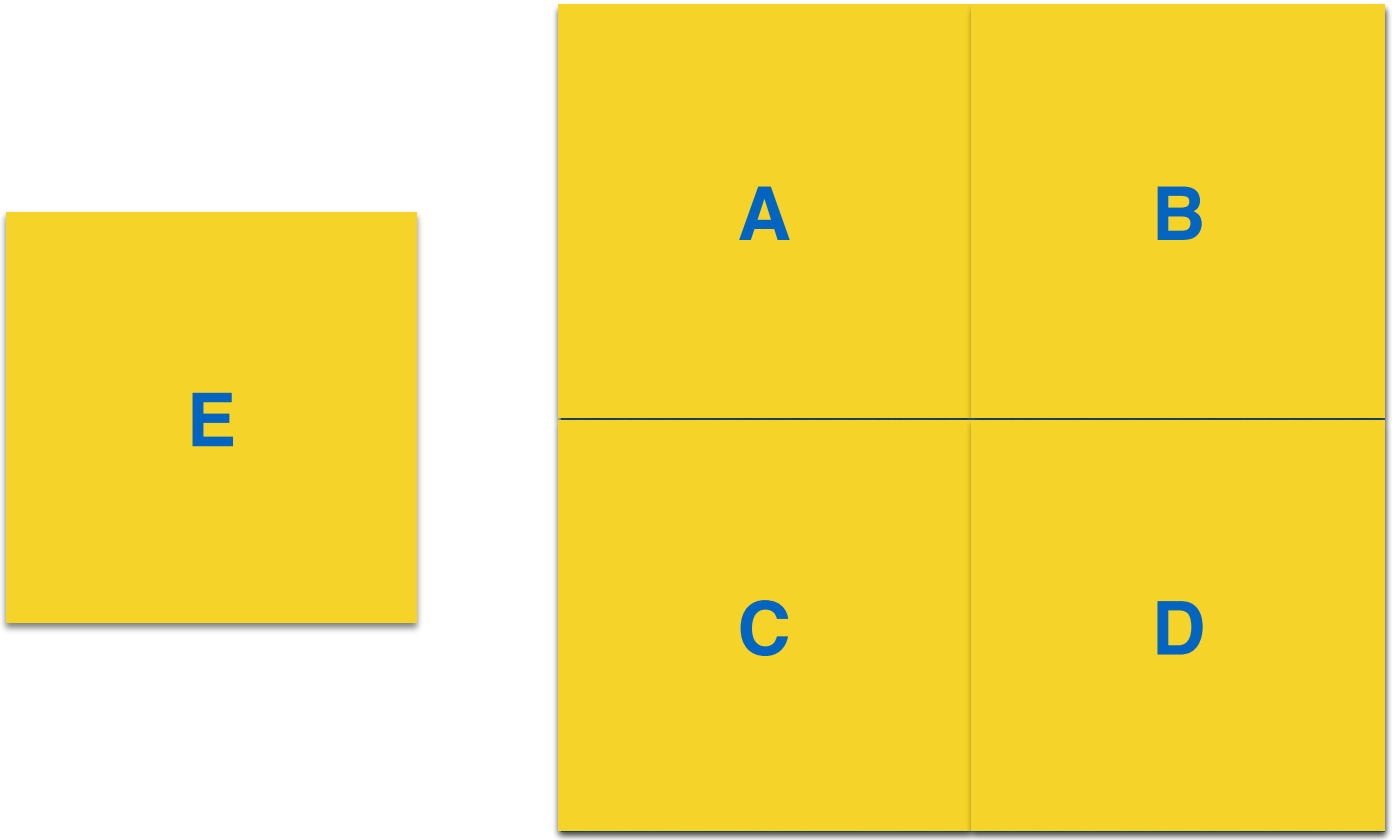
Vikram Sreekanti

Buffer Replacement

- Buffer pool is your memory manager.
  - Data must be in memory to operate on it; buffer manager hides the fact that not all data in memory.
    - This should sound familiar to everyone (especially anyone who's taken 162).
- We've said it a million times: your data doesn't fit in memory.
- What do you do memory is full, and you want to load something else in?
  - You kick something out!
  - There are many, many different buffer replacement algorithms.
  - Common algorithms:
    - LRU
    - MRU
    - Clock
    - etc.

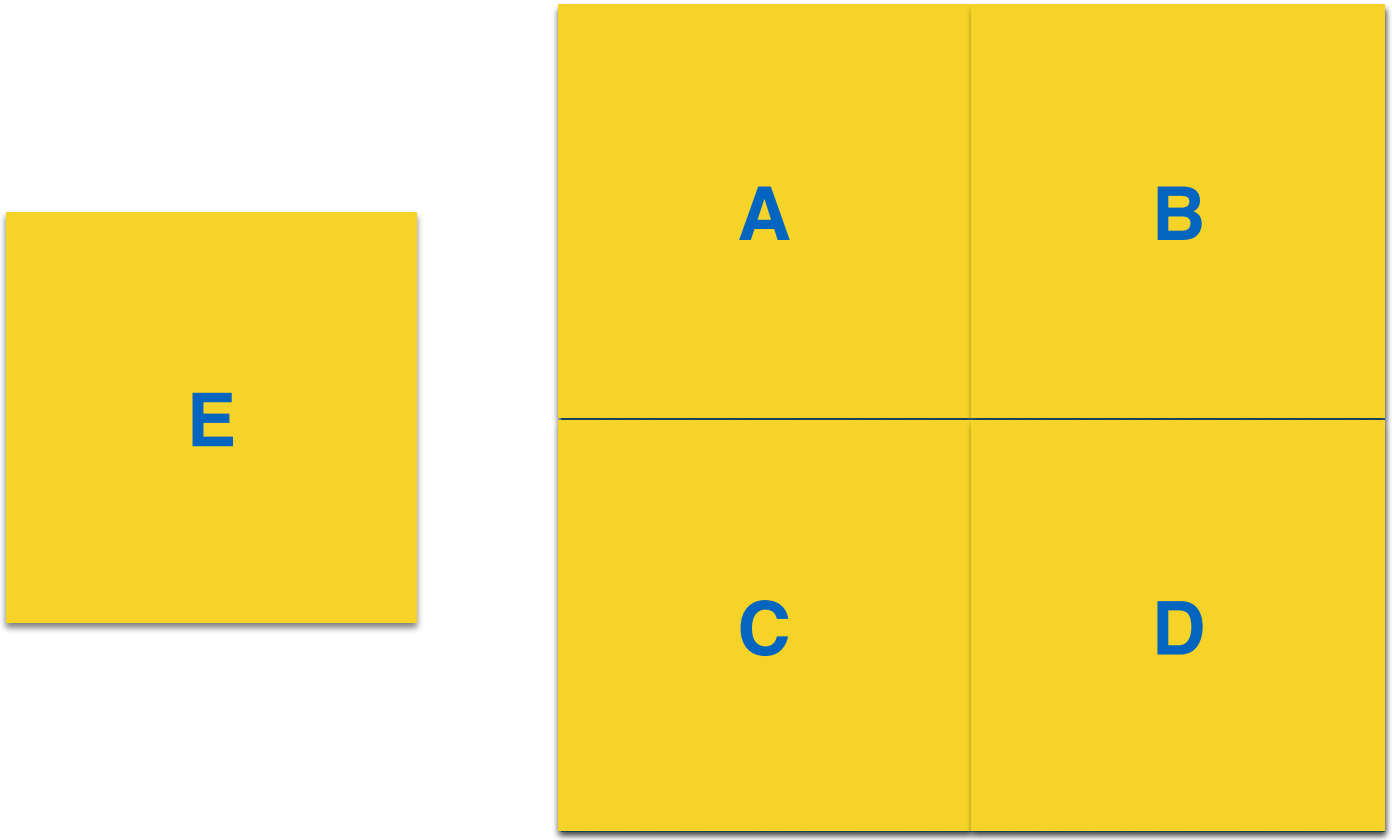
Buffer Replacement: Least Recently Used

Main Idea: When you need to evict something, you evict the page that was least-recently used.



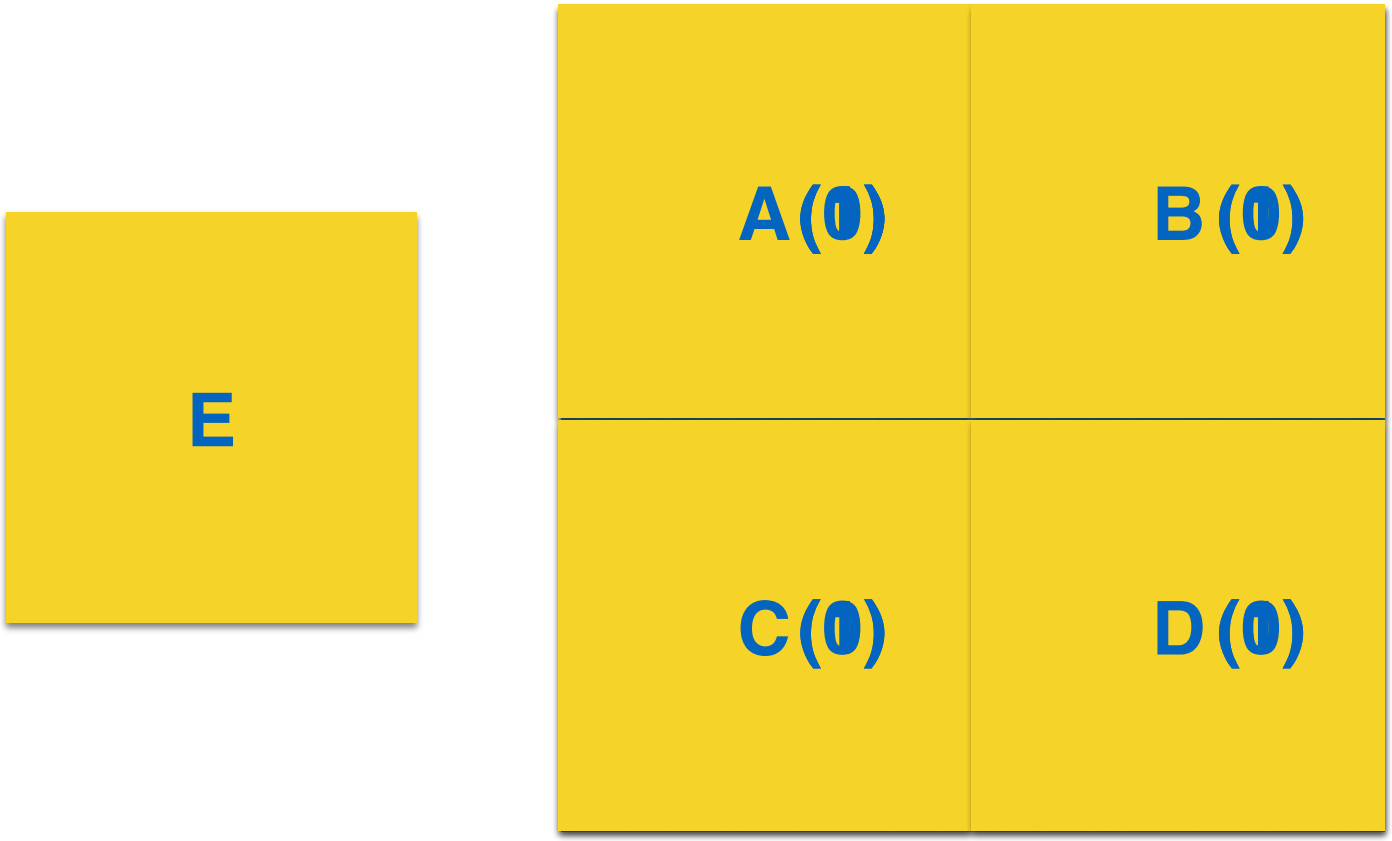
Buffer Replacement: Most Recently Used

Main Idea: When you need to evict something, you evict the page that was most-recently used.



Buffer Replacement: Clock

Main Idea: Close approximation of LRU, but each page gets a second chance bit.



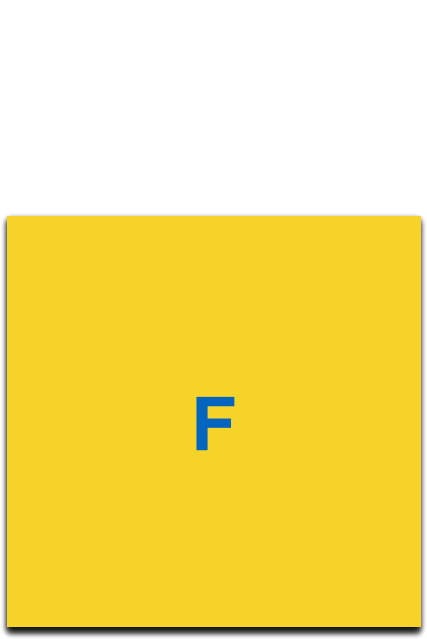
Buffer Replacement Exercises

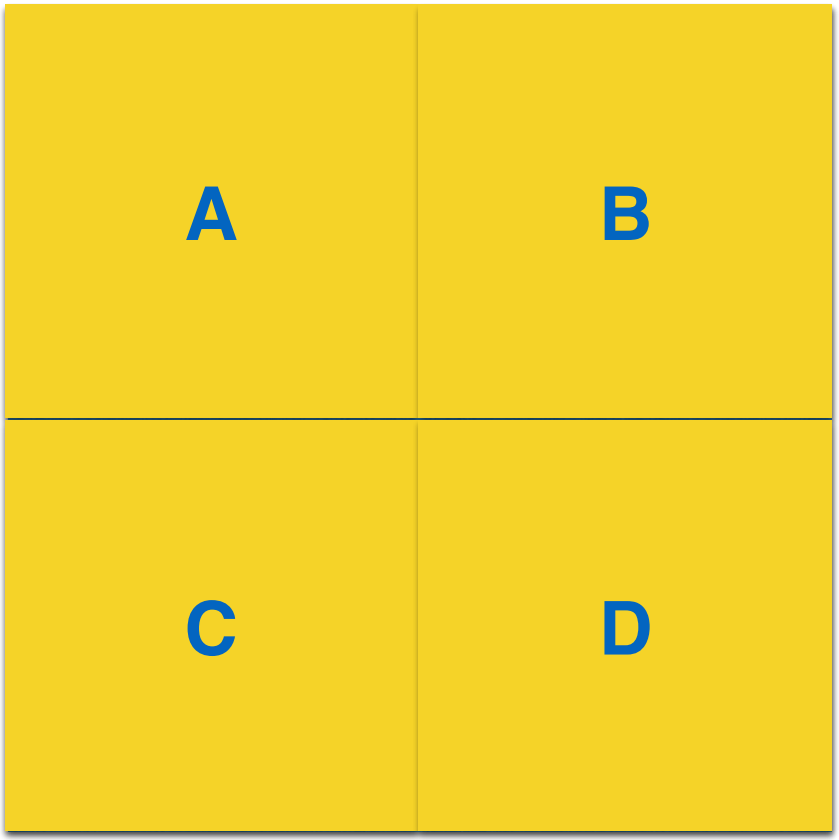
You have four buffer pages. What is the end state of the buffer pool for {LRU, MRU, Clock} after the following access pattern? What is your hit rate?

A B C D A F A D G D G E D F

Buffer Replacement Exercise: LRU

A B C D A F A D G D G E D F

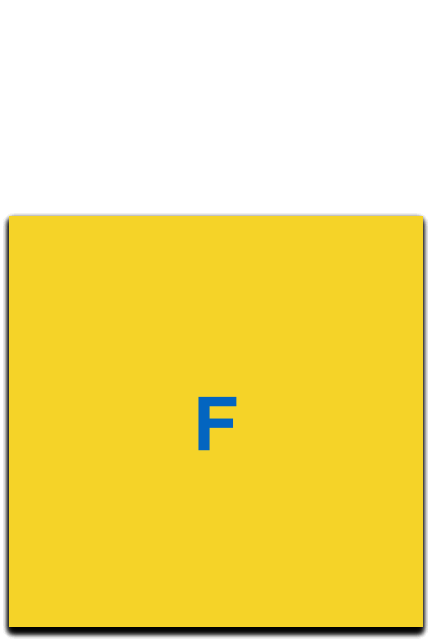


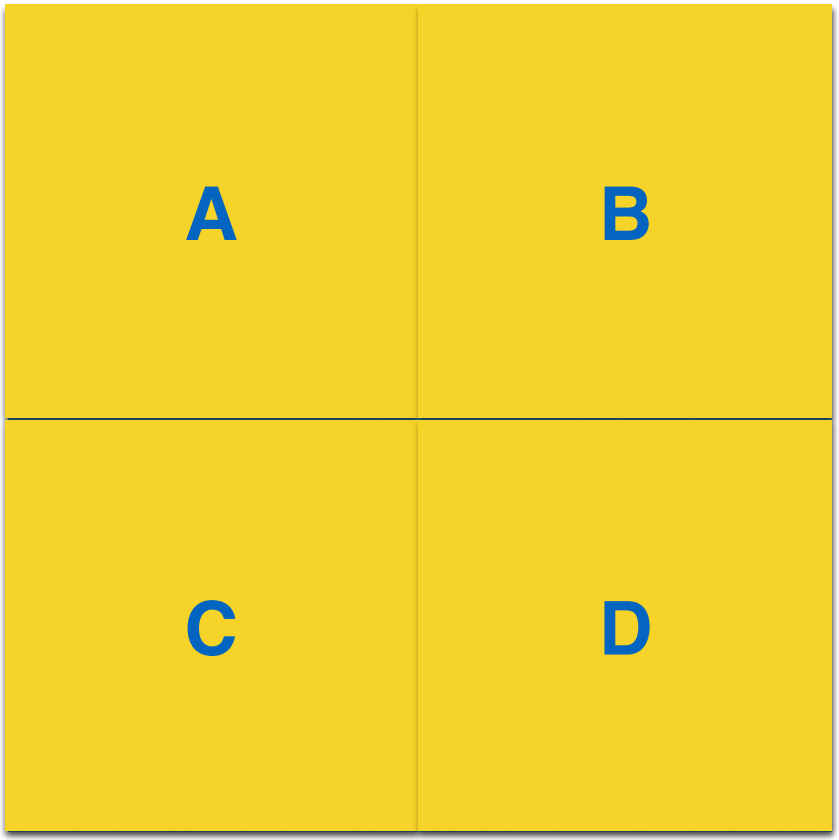


Number of Hits:

Buffer Replacement Exercise: MRU

A B C D A F A D G D G E D F



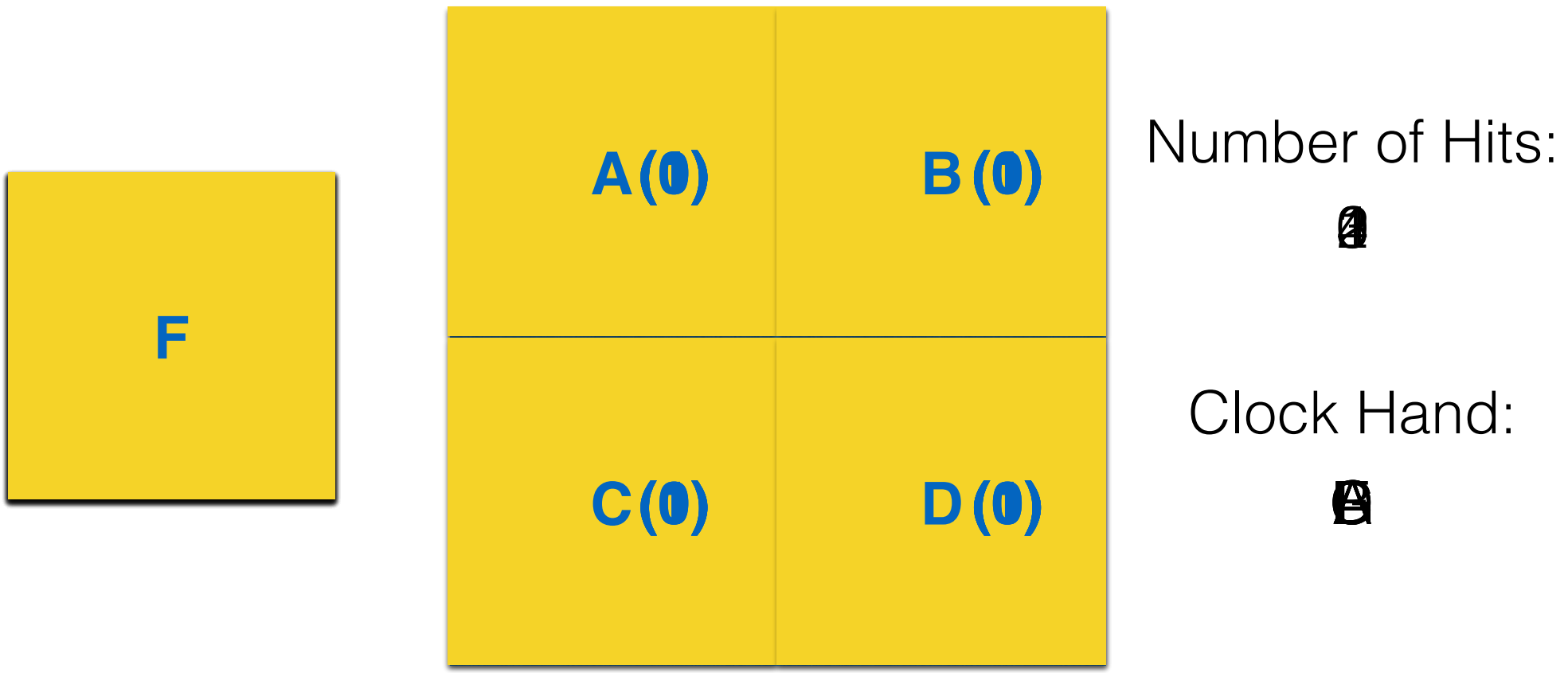


Number of Hits:

0
1
2

Buffer Replacement Exercise: Clock

A B C D A F A D G D G E D F



Heap (Unsorted) File

IMPORTANT: Do not confuse this with the heap data structure — there no relation.

	Heap File (IOs)
Sequential Scan	B
Equality Search	.5B
Range Search	B
Insertion	2
Deletion	.5B + 1

Sorted File

	Heap File (IOs)	Sorted File (IOs)
Sequential Scan	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	(same as Insertion)

Indexes (Indices?)

Disk-based data structure for fast value-based lookups of data in tables.

Indices come in many shapes and sizes: tree-based, hash; clustered vs. unclustered; alternatives 1, 2, and 3

Three Storage Alternatives for Indexing

1. The pages of the index store the actual tuples.

2. The pages of the index store <key, rid> pairs, where rid is one record with key.

3. The pages of the index store <key, list[rid]> pairs, where list[rid] is all rids with key.
   A. Can anyone see any issues with this?

Clustered vs. Unclustered Indexes

- Clustered: Entries in the index are stored (approximately) in sorted order over key.
  - Alternative 1 indices are always clustered.

- Unclustered: … not clustered.

Note: Clustering affects the way that the actual file is stored on disk.

Index and Storage Exercises

What are important factors in determining whether or not you should add an index to a table?

Can’t use hash indexing with range queries. Should know which field to cluster on (calculate I/Os based on typical queries that you’ll need to run). Decide if you even want to cluster (high maintenance cost).

Index and Storage Exercises

Consider the table:

```
Enrolled(sid, course, grade)
```

and the query:

```
SELECT * FROM Enrolled where sid > 4500;
```

Assume SIDs are unique and are in the range [1, 6000]

Index and Join Exercises

How many IOs would this query take if the table was stored in a heap file?

B = 500

Index and Join Exercises

How many IOs would this query take if the table was stored in a sorted filed sorted on grades?

B = 500

Index and Join Exercises

How many IOs would this query take if the table was stored in a sorted filed sorted on SID?

$$B = \log_2(500) + .25\,(500)$$
$$= 9 + 125$$
$$= 134 \text{ IOs}$$

Index and Join Exercises

True or False? Given the table `Students(sid, gpa, age)`, a hash index on GPA will significantly increase the performance of the following query:

```
SELECT * FROM Students WHERE age > 20;
```

False

Index and Join Exercises

True or False? Given the table `Students(sid char(20), gpa float, age integer)`, a clustered tree based index on gpa will increase the performance of the following query:

```
SELECT * FROM Students where age > 20 AND
gpa > 3.5;
```

True