Welcome to CS 186, Section 4!

TA: Bryan Munar

OH: Mondays 11-12pm and Thursdays 2:30-3:30pm (651 Soda)

DISC: Tuesdays 11-12am (136 Barrows) and Wednesdays 10-11am (130 Wheeler)

Announcements and Such

- Project/Homework 2 due on Monday!!

- Sign up to be partners with someone if you would like!

- Vitamin due tonight (if you are in Tuesday section)!!

Discussion 4: Disks, Buffers, Files!

Overview:

1. Buffer Management Policies
2. Worksheet exercises
3. File Organization
4. Worksheet exercises

(A majority of the slides are from Michelle and lecture!)

Buffer Management

What happens if a DBMS wants to scan a file?

What happens if a higher level in a DBMS requests to access a page in disk?

A brief note on terminology

Block = Page

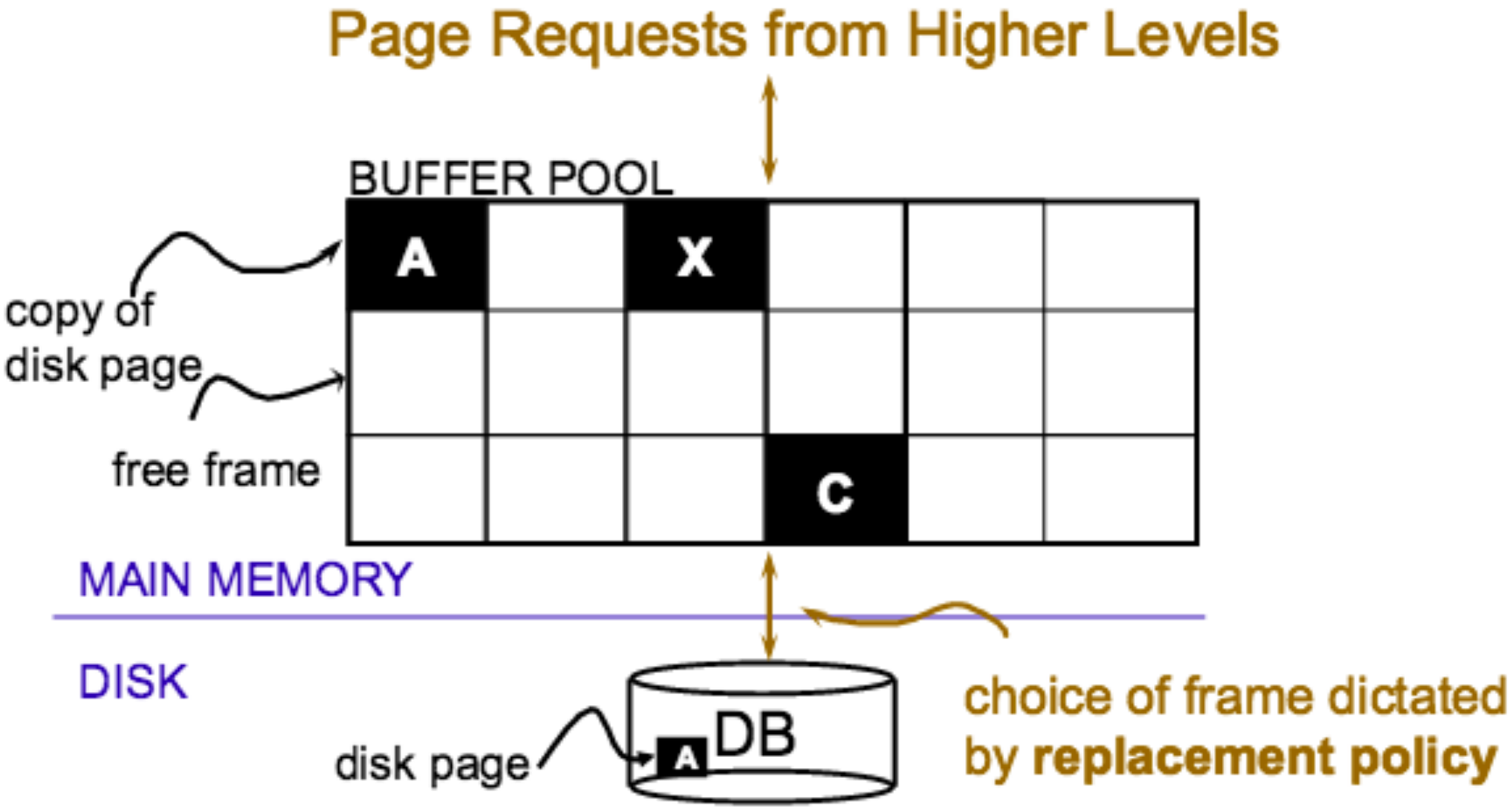
- Unit of transfer for disk read/write
- Typically 4KB in the book
  - And hence in formulas in slides
- 64KB is a good number today

Relation = Table

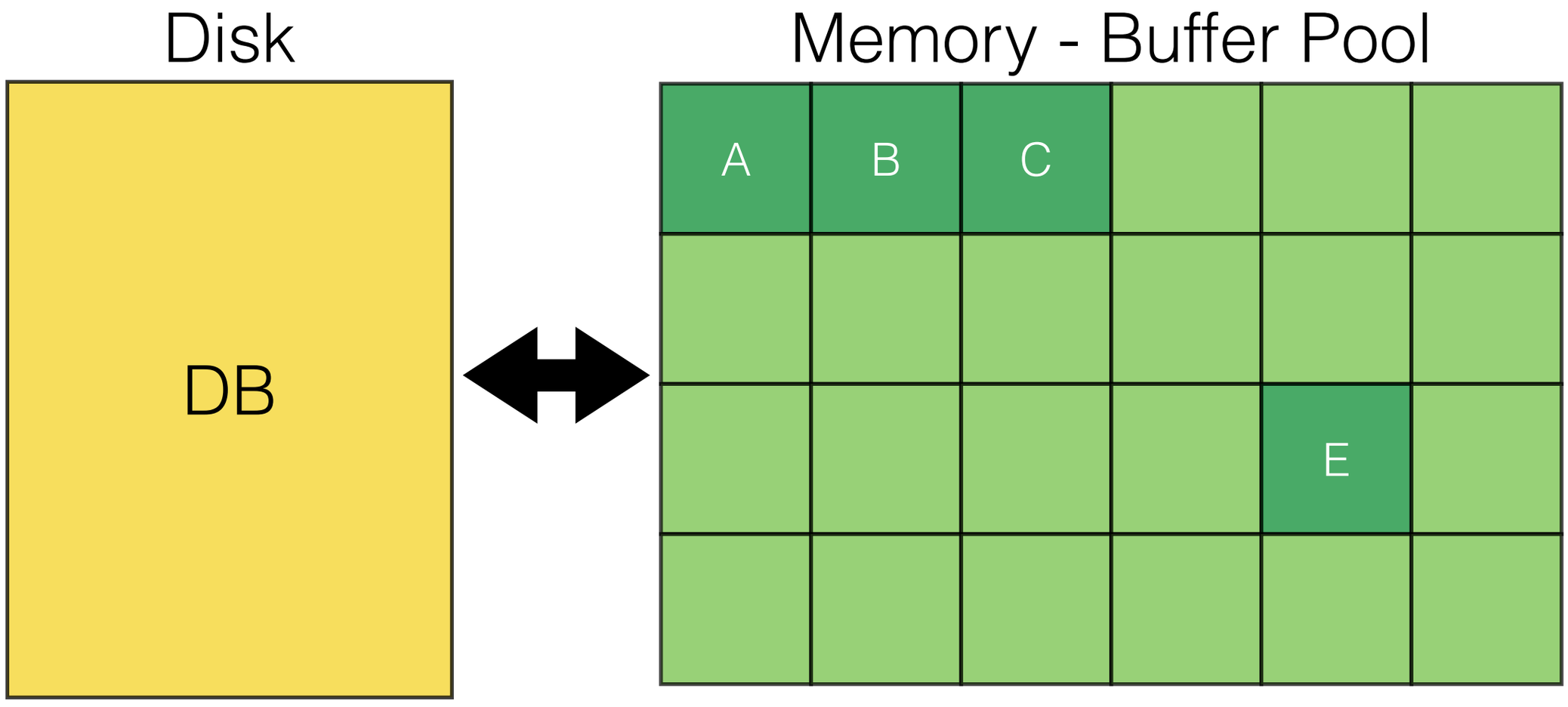
Tuple = Row = Record

Attribute = Column = Field

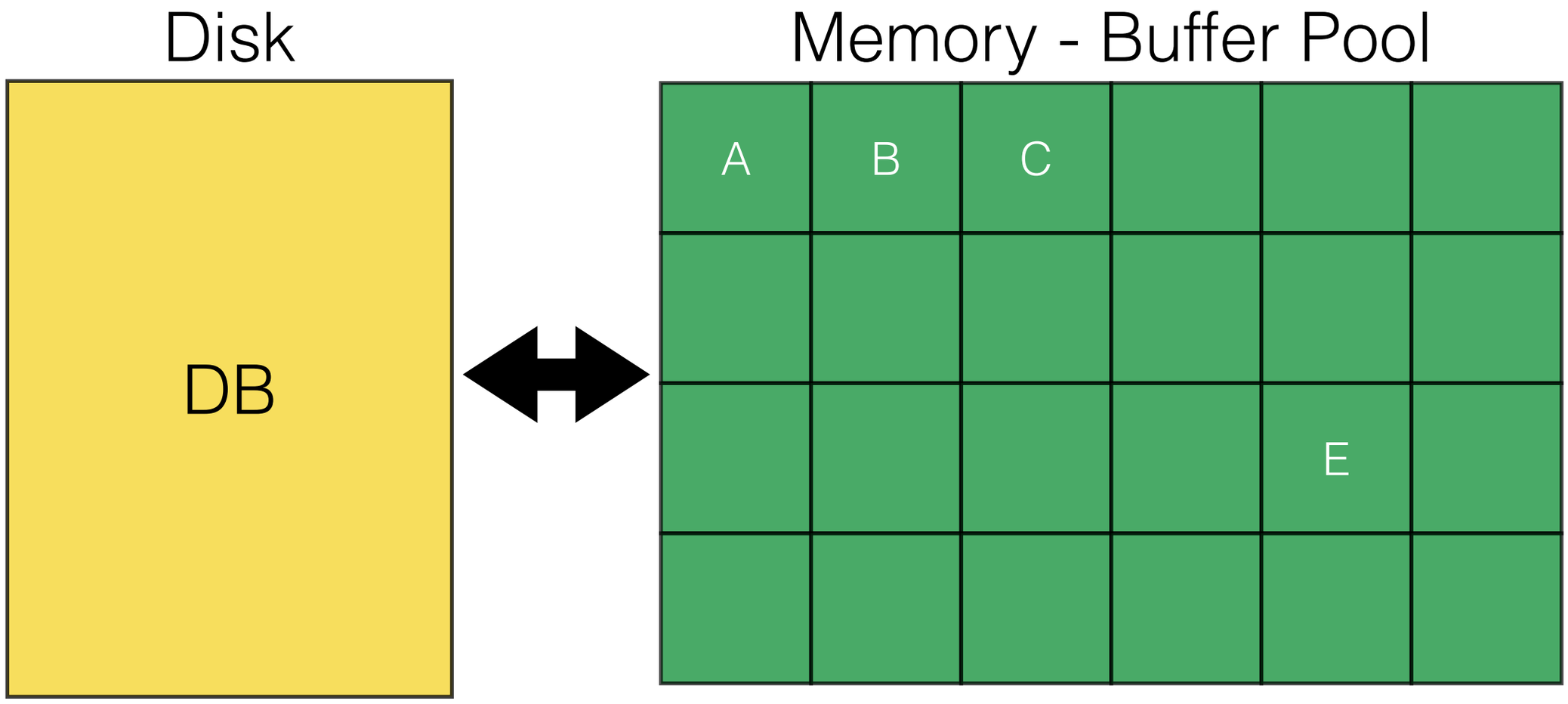
Buffer Management in a DBMS



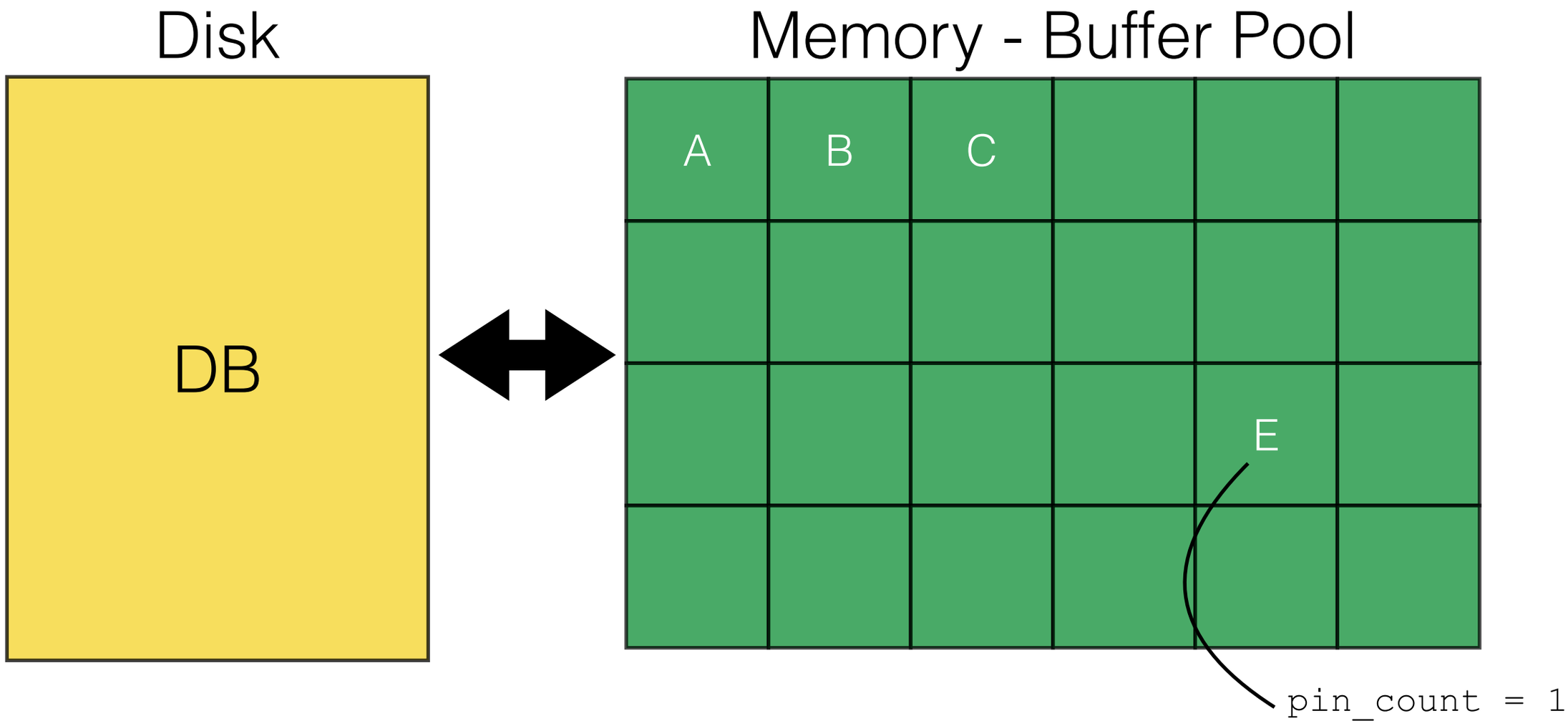
- *Data must be in RAM for DBMS to operate on it!*
- *BufMgr hides the fact that not all data is in RAM*


Disk
DB
Memory - Buffer Pool
A
B
C
E

What happens when our buffer pool is full?
Which pages can we replace?



"Pin" a page (`pin_count++`) when page is requested. Only replace if `pin_count == 0`.



Buffer Replacement Policy

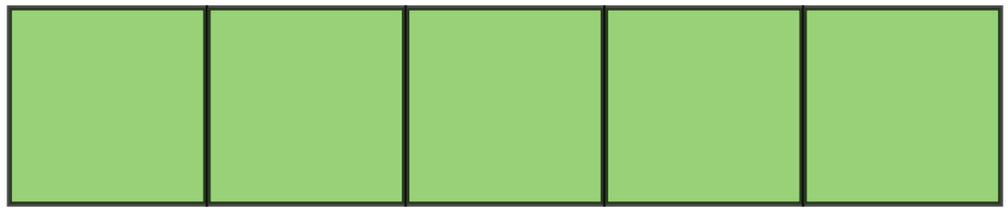
- Frame chosen for replacement using replacement policy (LRU, MRU, Clock, etc.)
- Policy can have a big impact on I/O's

Least Recently Used (LRU)

- Replace page that has been unused for the longest amount of time
  - Assumes pages used recently will be used again
- Must keep track of last time page was used/pinned

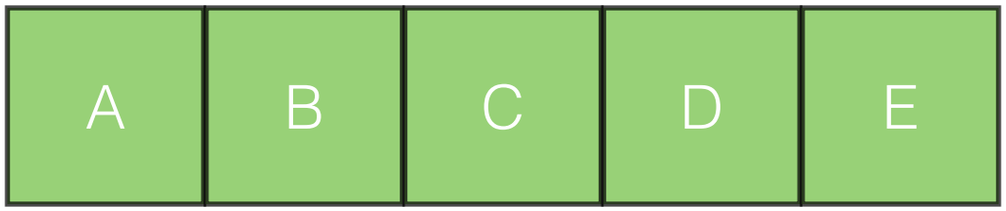
- Prone to sequential flooding
  - Reading all pages in a file multiple times
  - # buffer pages < # pages in file

LRU - Sequential Flooding



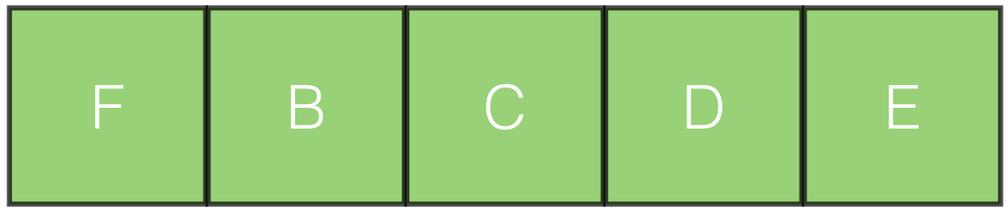
A, B, C, D, E, F, A, B, C, D

LRU - Sequential Flooding



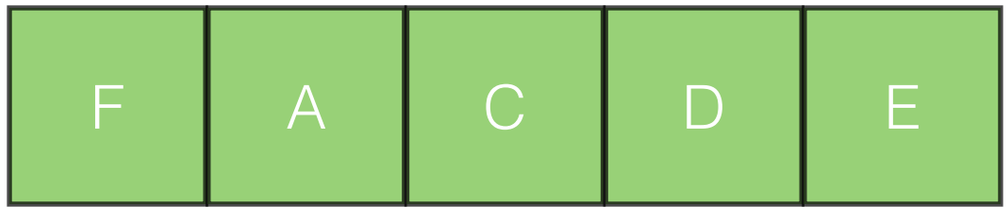
F, A, B, C, D

LRU - Sequential Flooding



A, B, C, D

LRU - Sequential Flooding

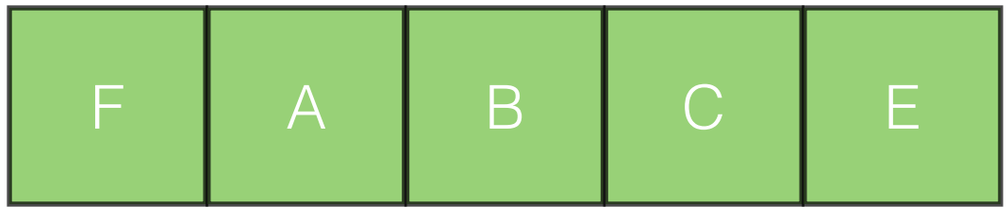


B, C, D

LRU - Sequential Flooding

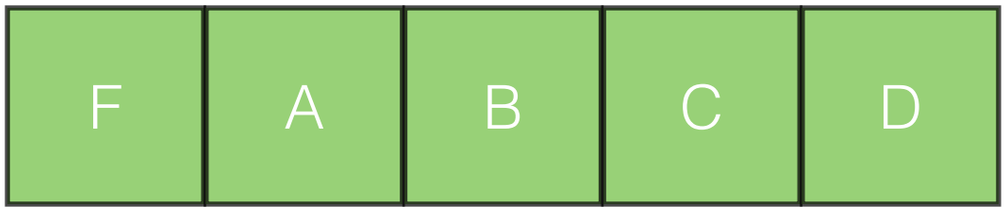
C, D

LRU - Sequential Flooding



D

LRU - Sequential Flooding

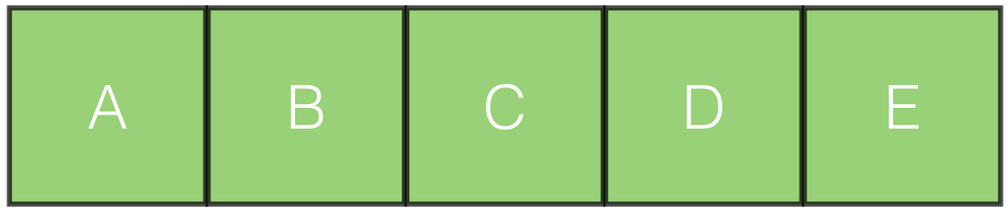


Every page request results in a cache miss!

Most Recently Used (MRU)

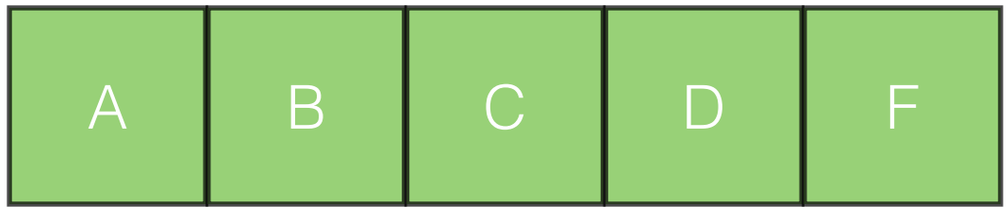
- Replace page that has just been used
- Fixes sequential flooding

MRU - Sequential Flooding



F, A, B, C, D

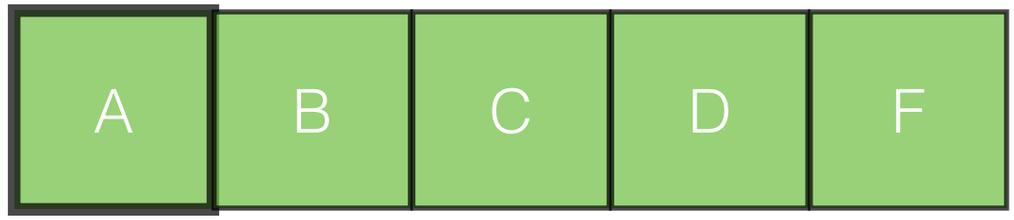
MRU - Sequential Flooding



A, B, C, D

MRU - Sequential Flooding

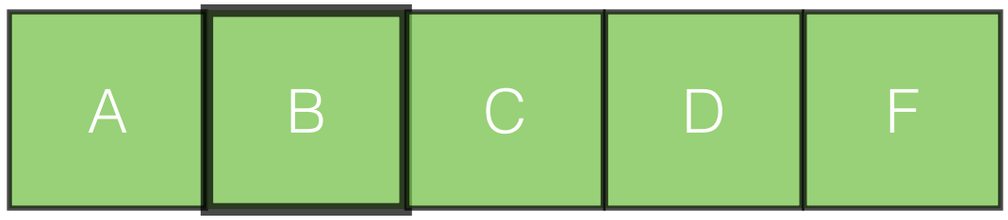
Cache hit!



B, C, D

MRU - Sequential Flooding

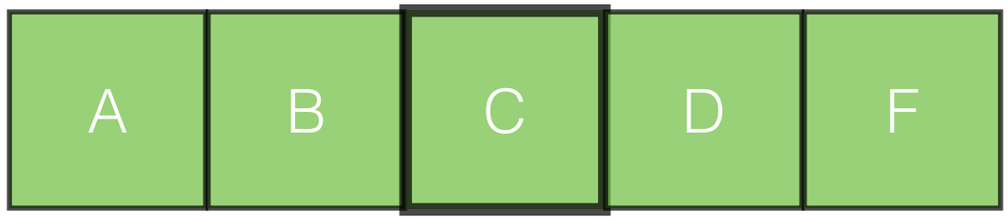
Cache hit!



C, D

MRU - Sequential Flooding

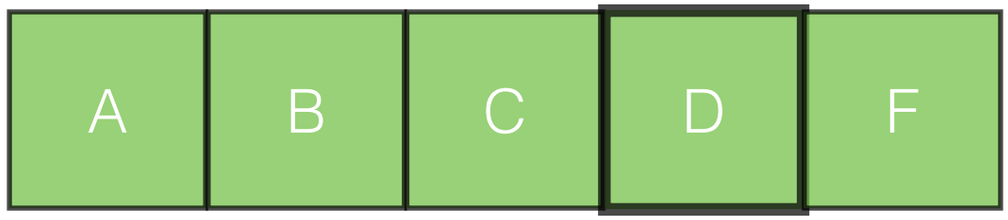
Cache hit!



D

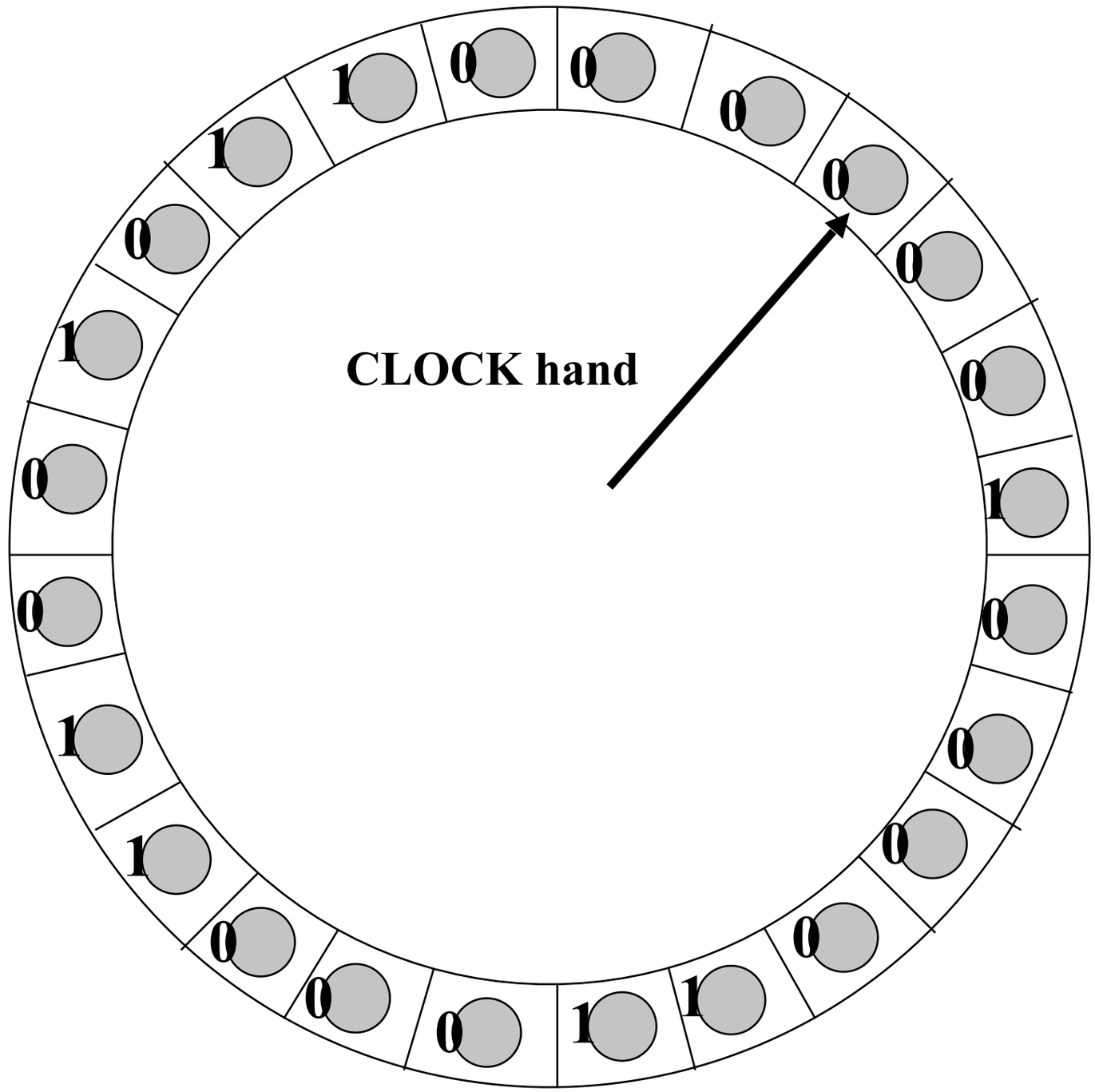
MRU - Sequential Flooding

Cache hit!



Clock Replacement

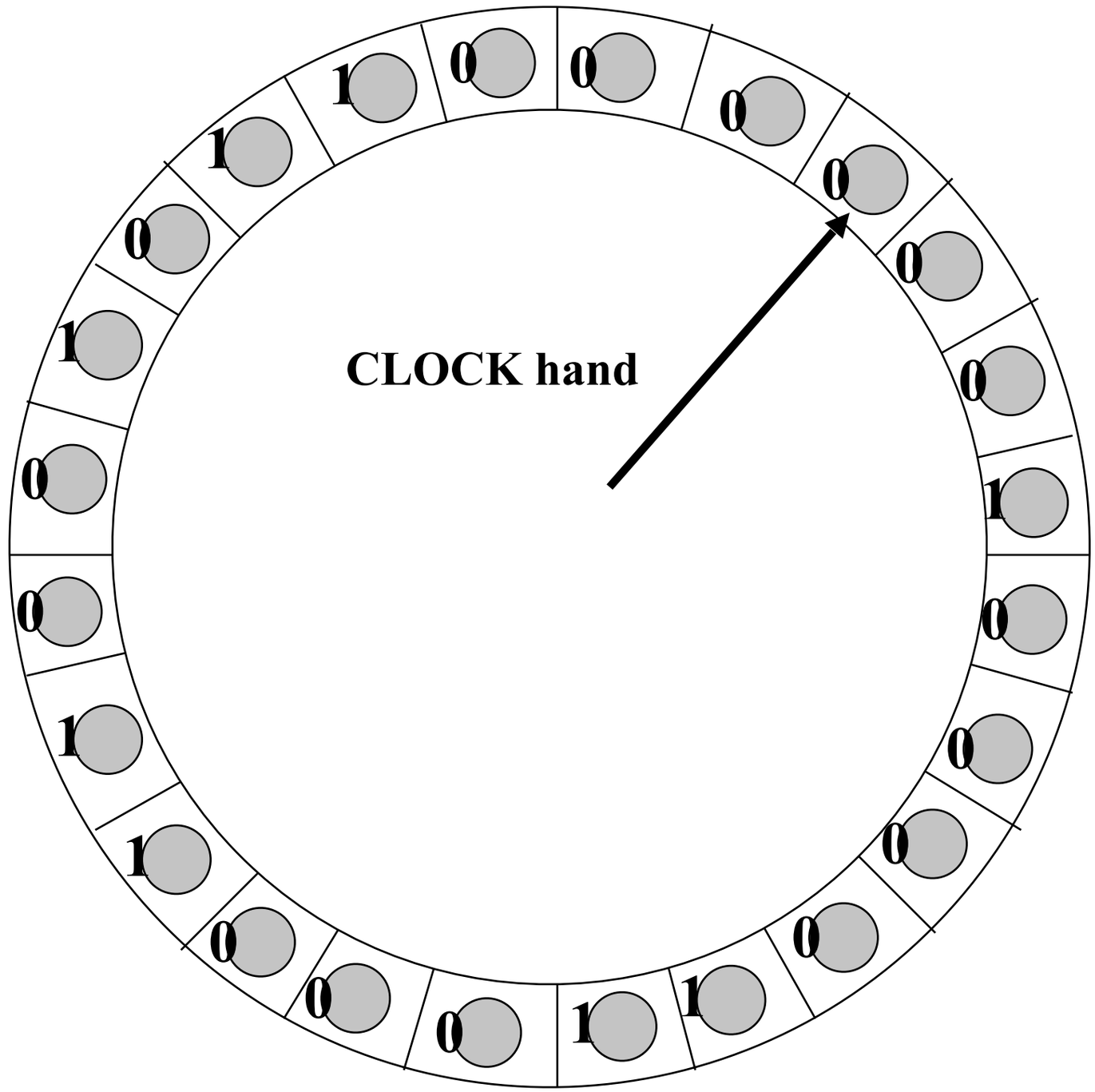
- All pages placed in a circular list.
- Each page has reference bit (“second-chance” bit) indicating if page has been accessed.



IMPORTANT

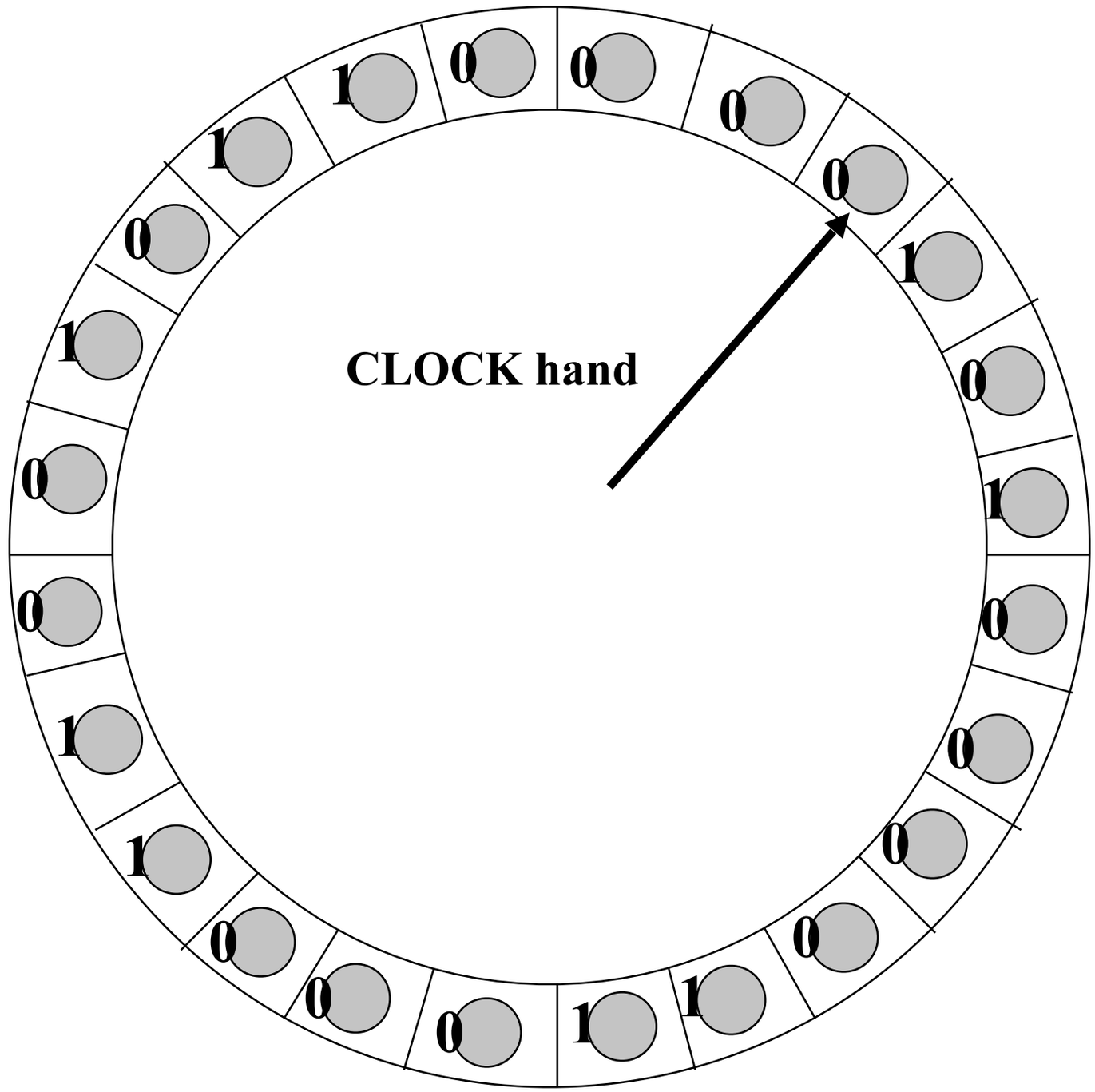
Clock Replacement

- On a HIT, set reference bit to 1.



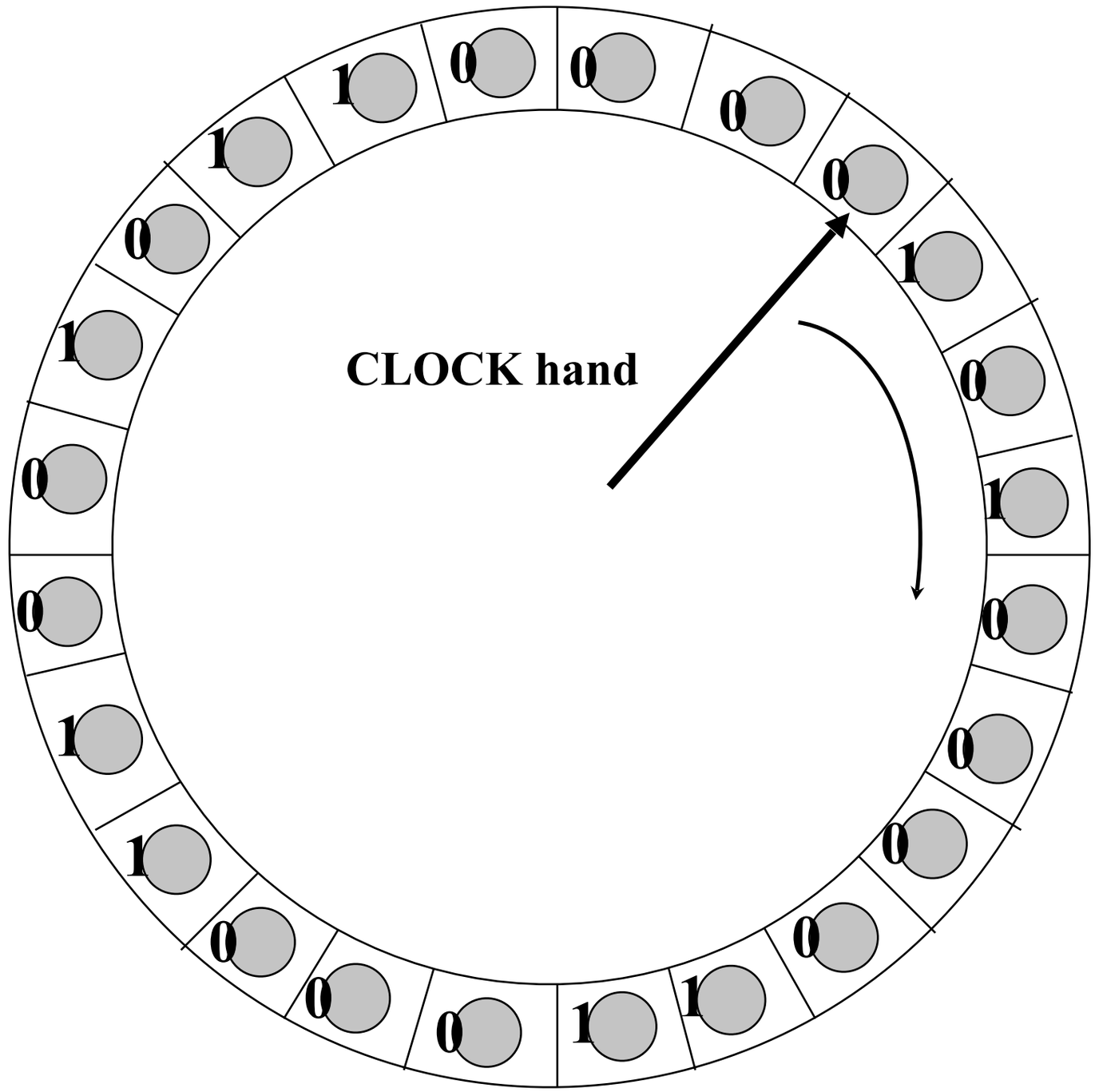
Clock Replacement

- On a HIT, set reference bit to 1.



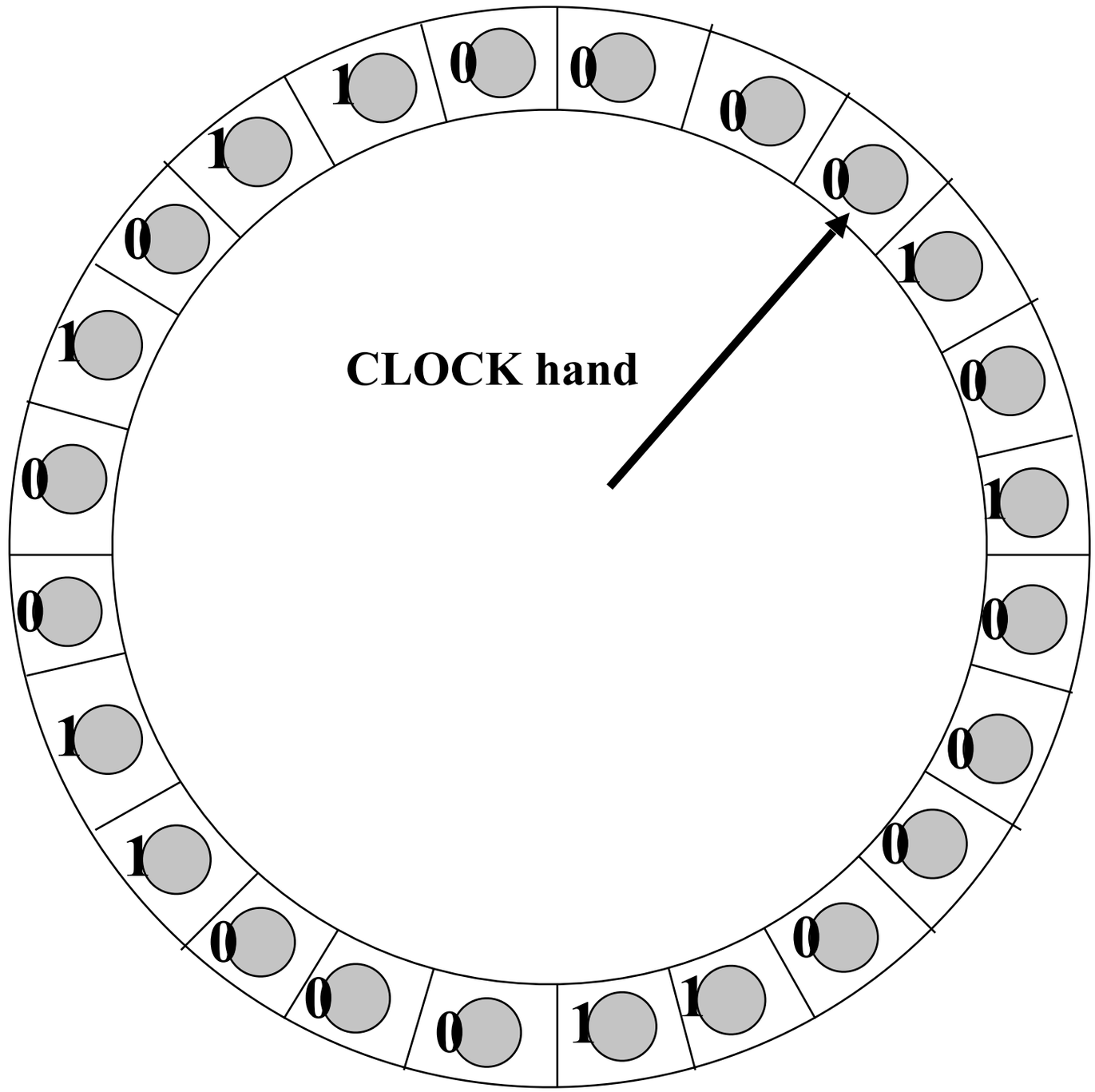
Clock Replacement

- On a MISS, move clock hand until reaches a page with “0” bit.
- Gives “1” bit pages a second chance and does not evict, but resets “1” to “0”.



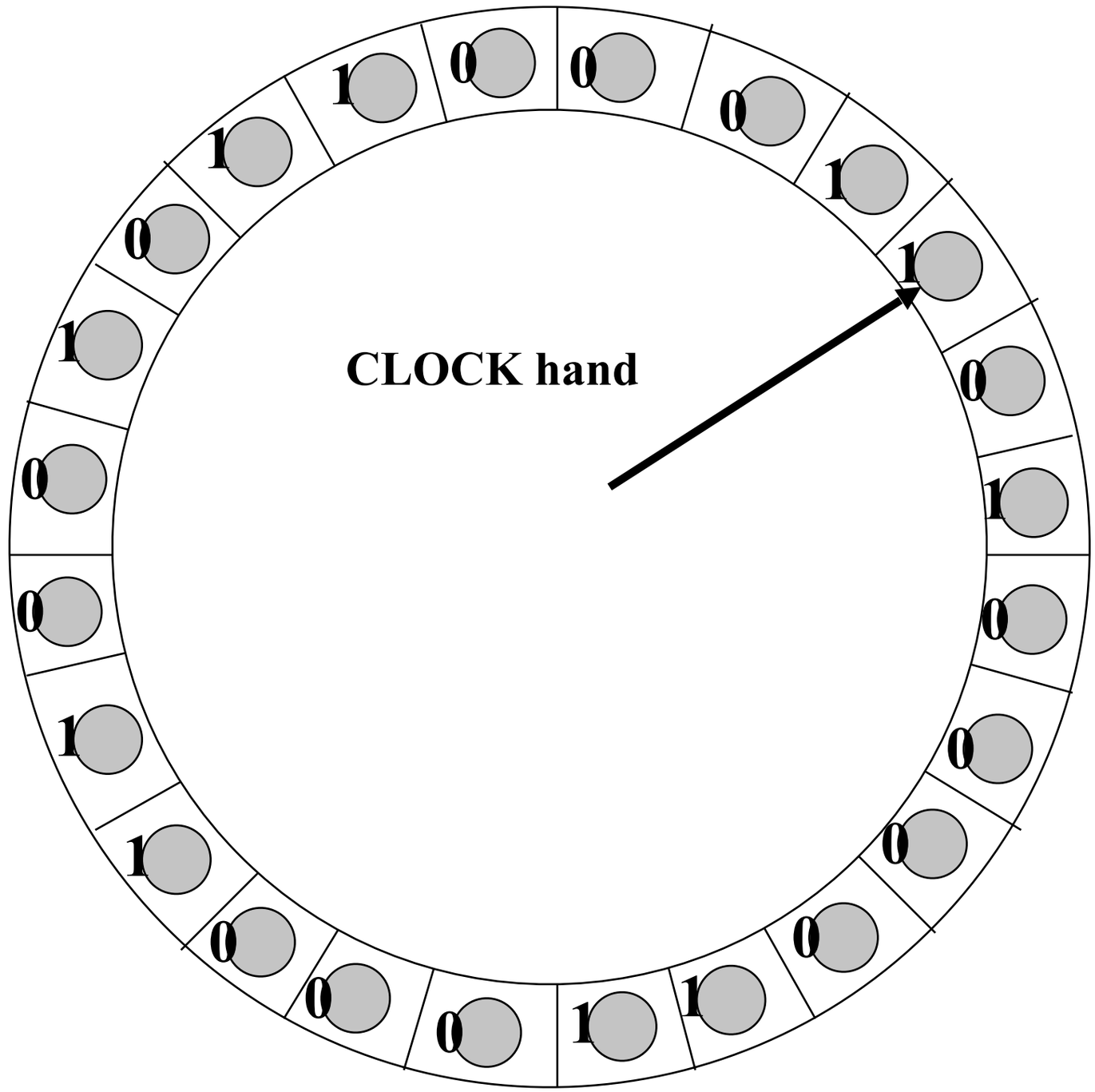
Clock Replacement

- 1 MISS



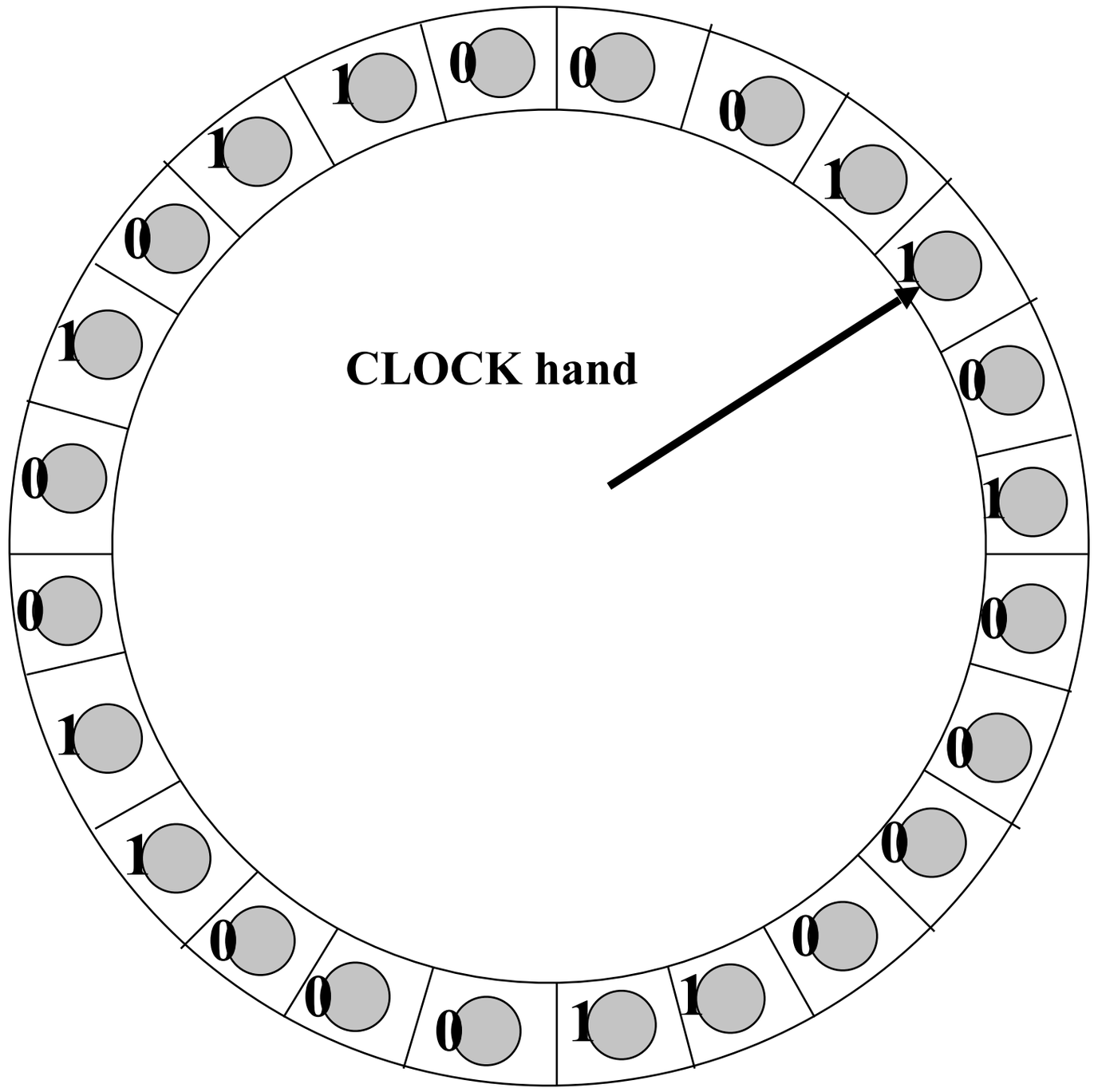
Clock Replacement

- 1 MISS
- when a page is replacement, move clock pointer forward.



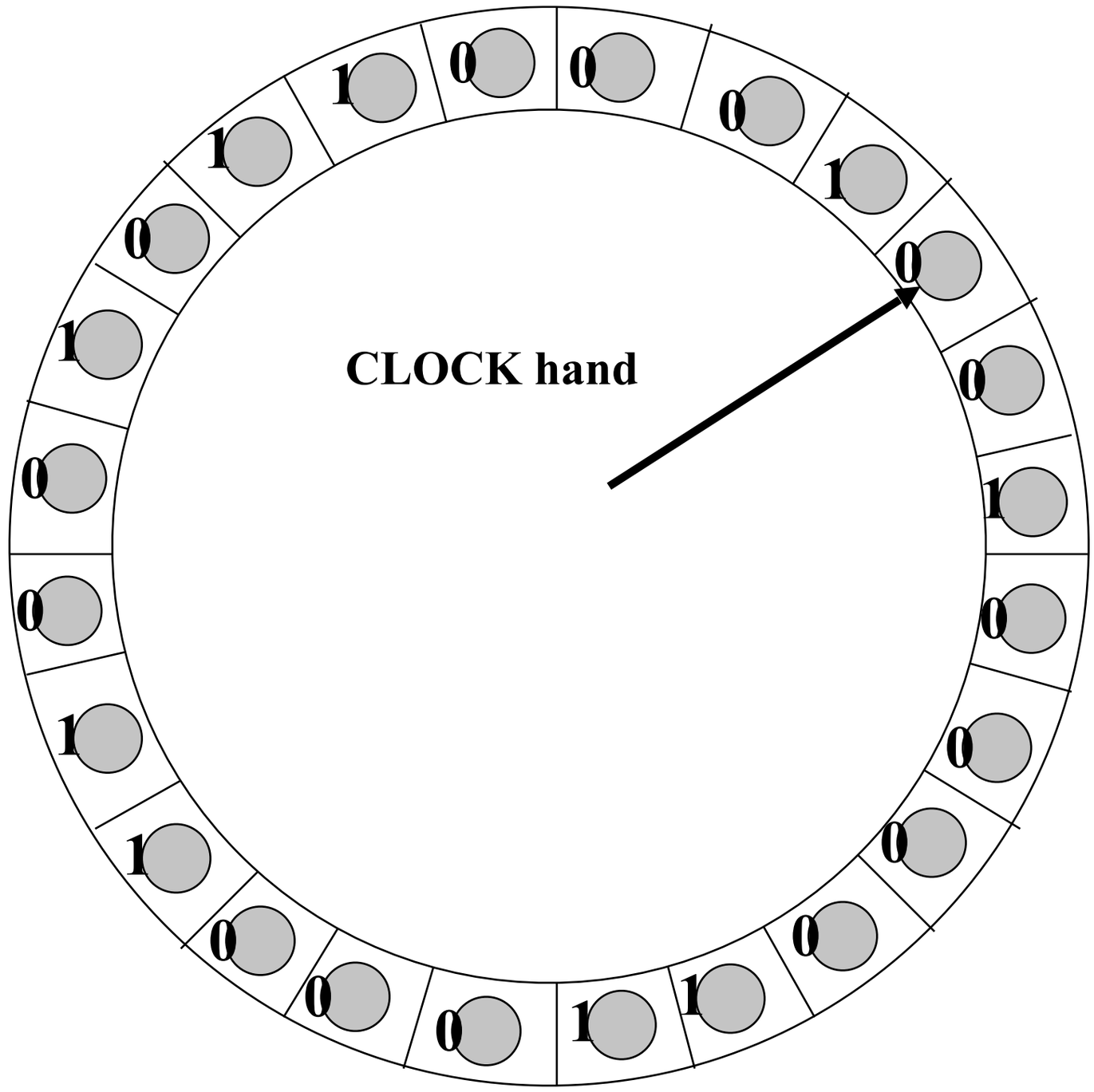
Clock Replacement

- Another MISS



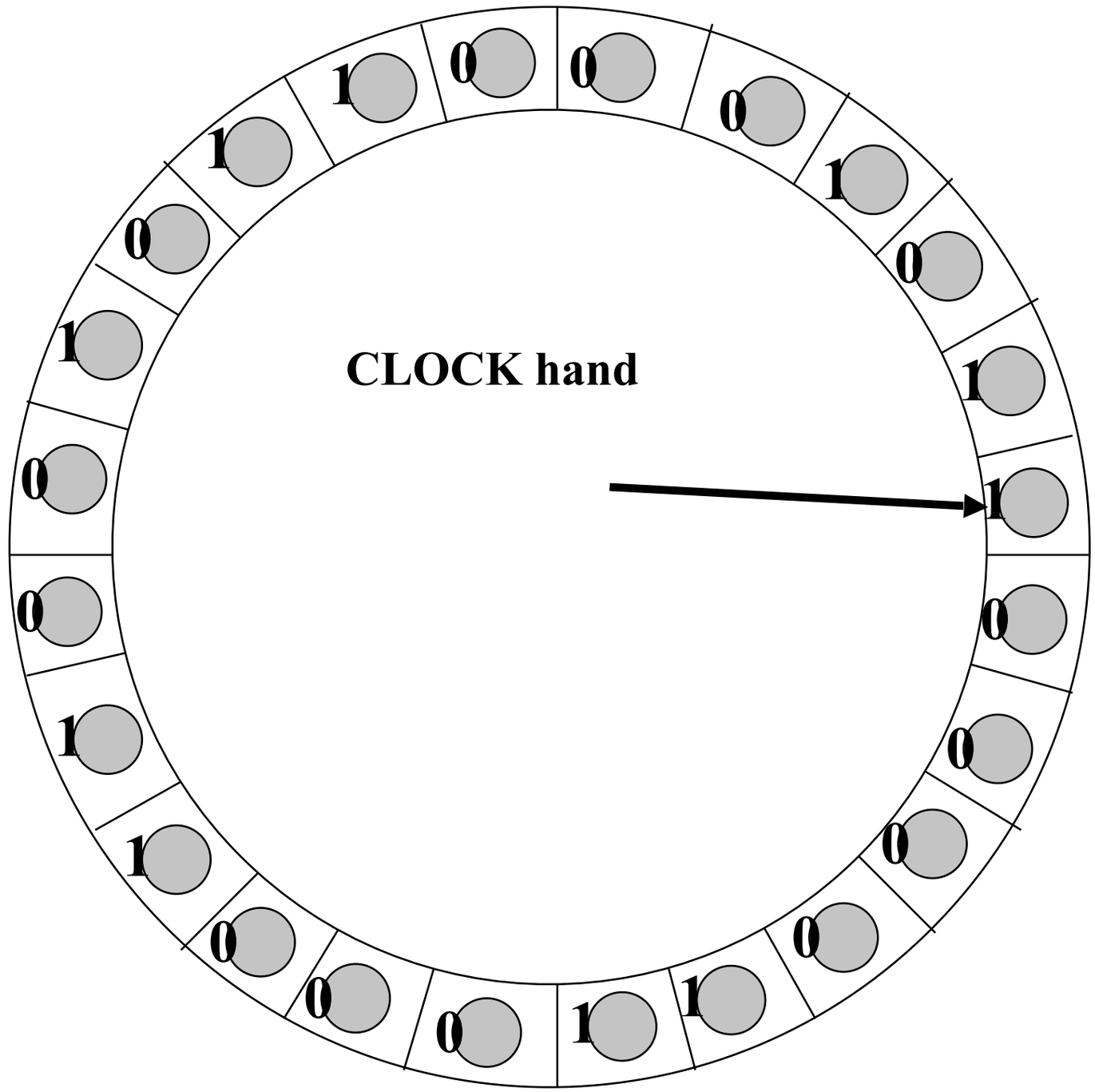
Clock Replacement

- Another MISS



Clock Replacement

- Another MISS



Do first page of worksheet!

File Organization

Files of Records

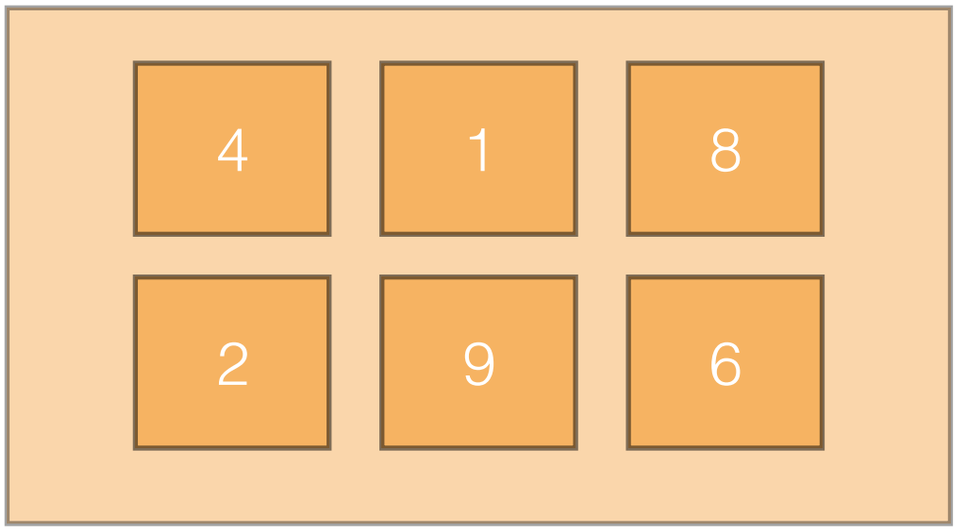
- Pages are the interface for I/O, but…
- Higher levels of DBMS operate on *records*, and *files of records*.

- FILE: A collection of pages, each containing a collection of records. Must support:
  - insert/delete/modify record
  - fetch a particular record (specified using *record id*)
  - scan all records (possibly with some conditions on the records to be retrieved)

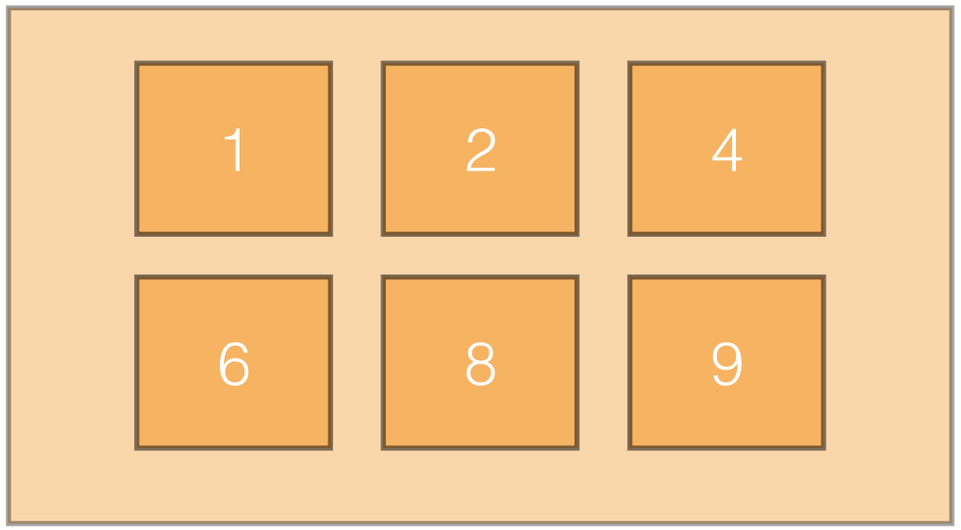
- Typically implemented as multiple OS "files"
  - Or "raw" disk space
- To support record level operations, we must:
  - keep track of the *pages* in a file
  - keep track of *free space* on pages
  - keep track of the *records* on a page

File Organization

- Heap files: unordered set of records
- Sorted file: ordered set of records

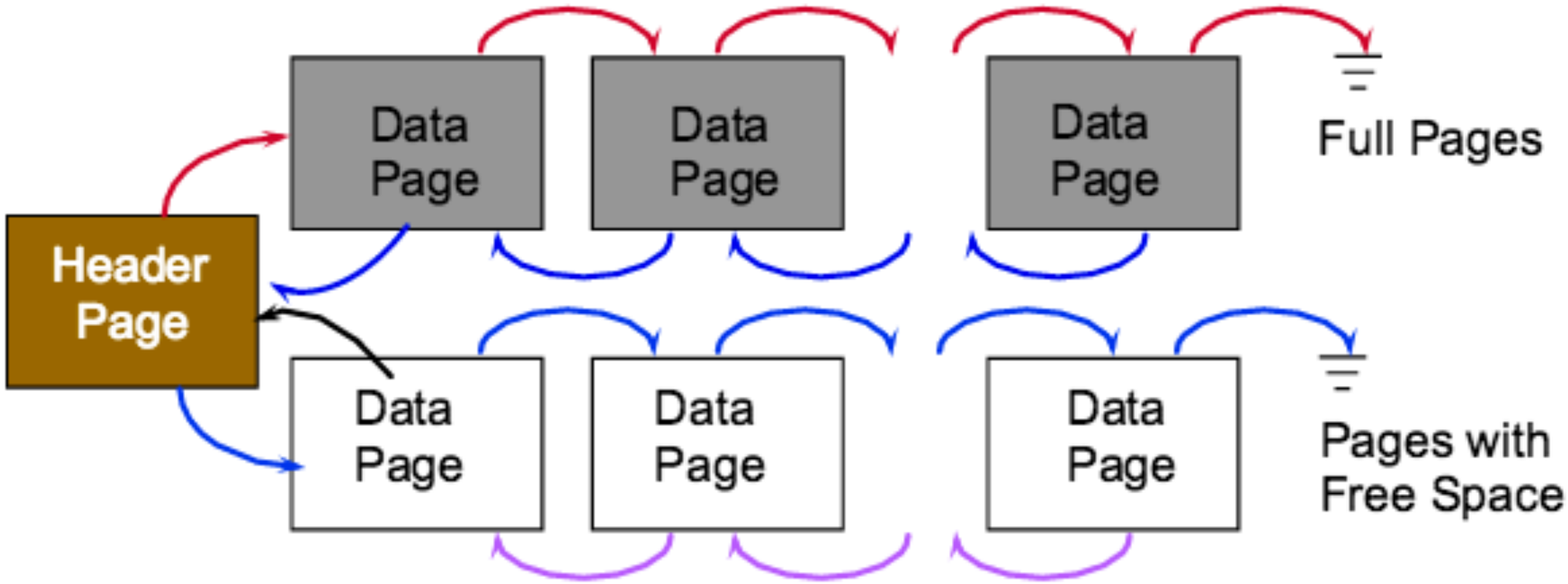


Heap file



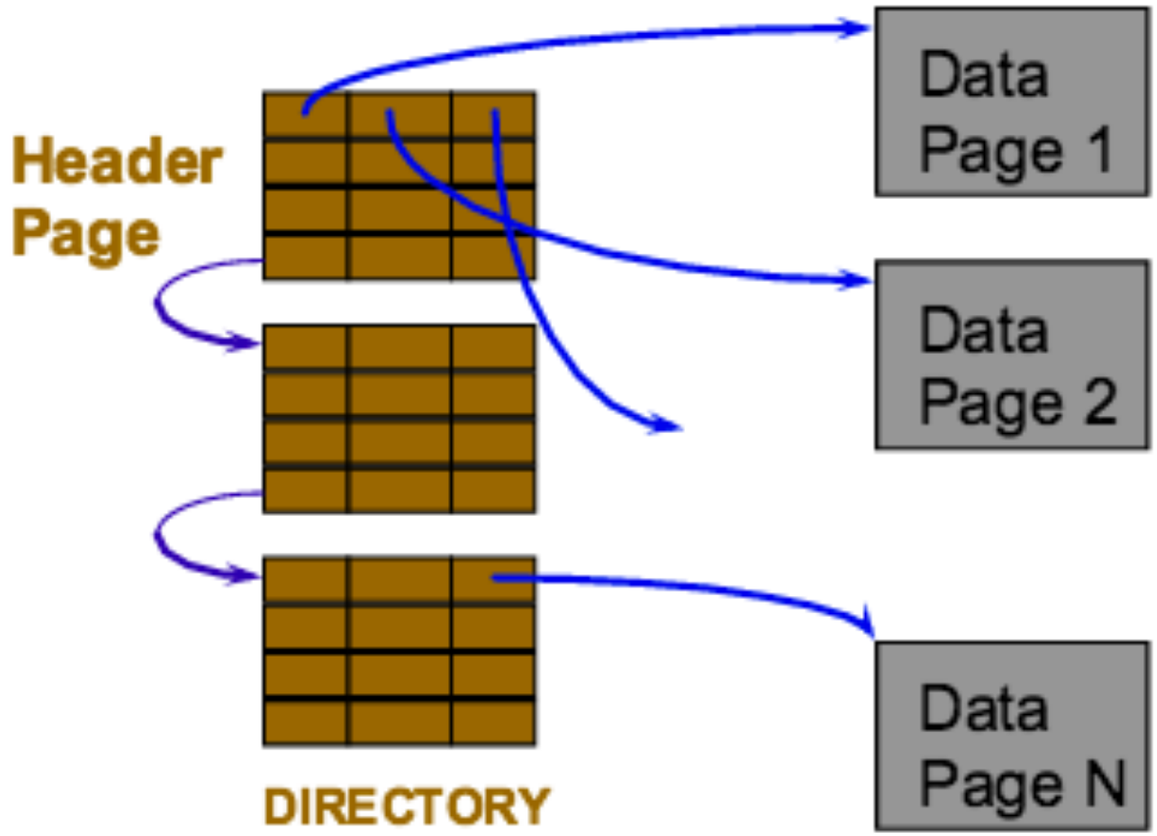
Sorted file

Heap File Implemented as a List



- Header page ID and Heap file name stored elsewhere
  - Database "catalog"
- Each page contains 2 "pointers" plus data
  - Problem for multi-page objects (blobs) – how to read blobs?

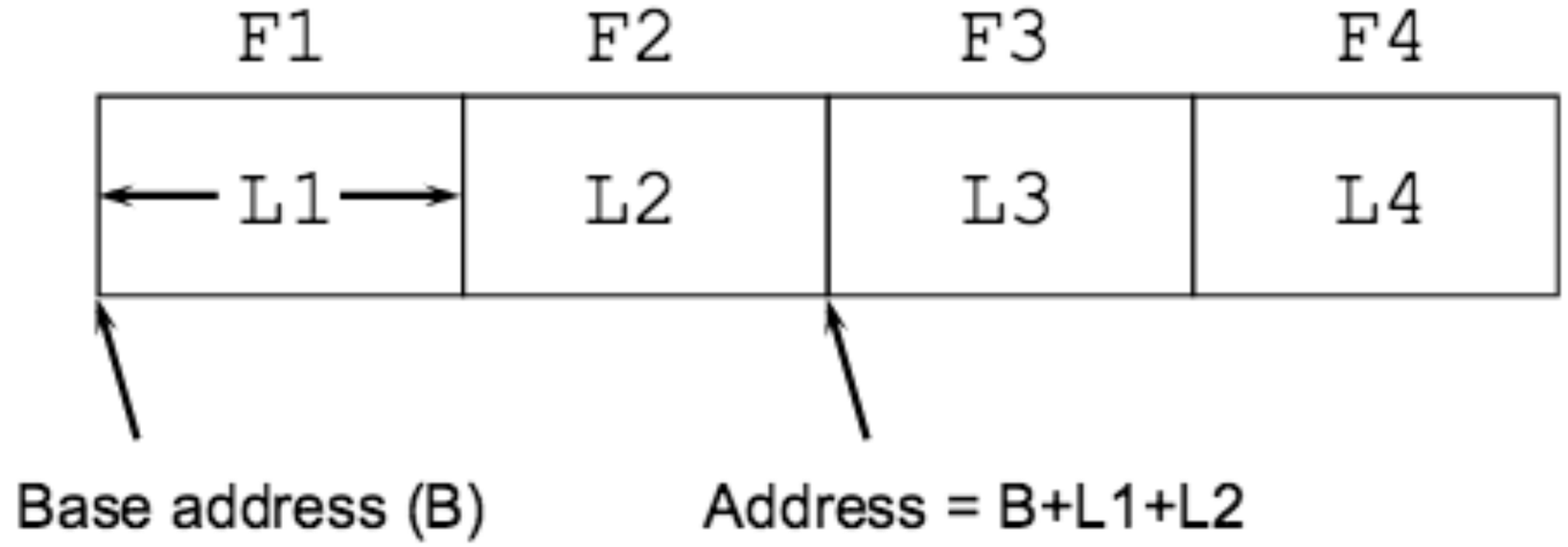
Better: Use a Page Directory



- Directory entries include #free bytes on the page.

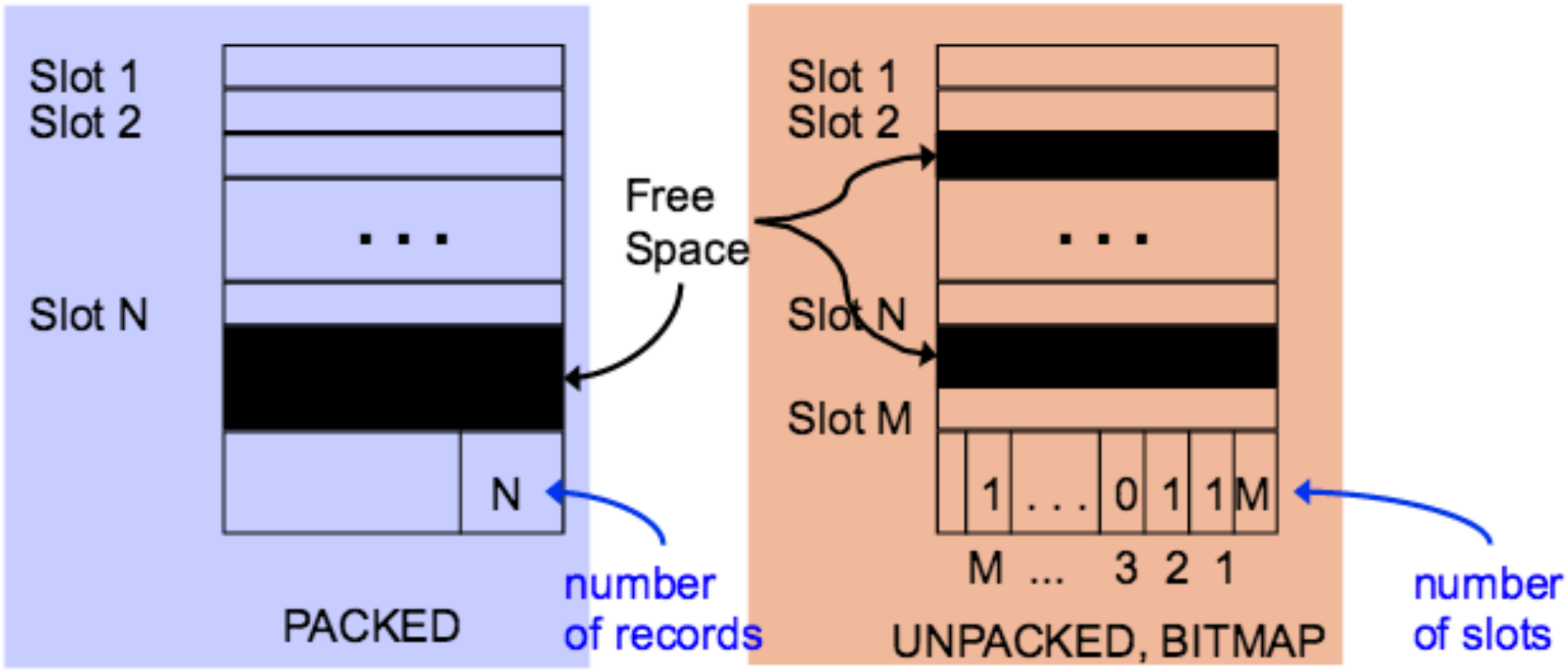
- Directory is a collection of pages; linked list implementation is just one alternative.
  - *Much smaller than linked list of all HF pages*!
  - Can also point to groups of pages (say 64k chunks)

Record Formats: Fixed Length



- Field types same for all records in a file.
  - Type info stored separately in *system catalog.*
- Finding *i'th* field done via arithmetic like arrays

Page Formats: Fixed Length Records

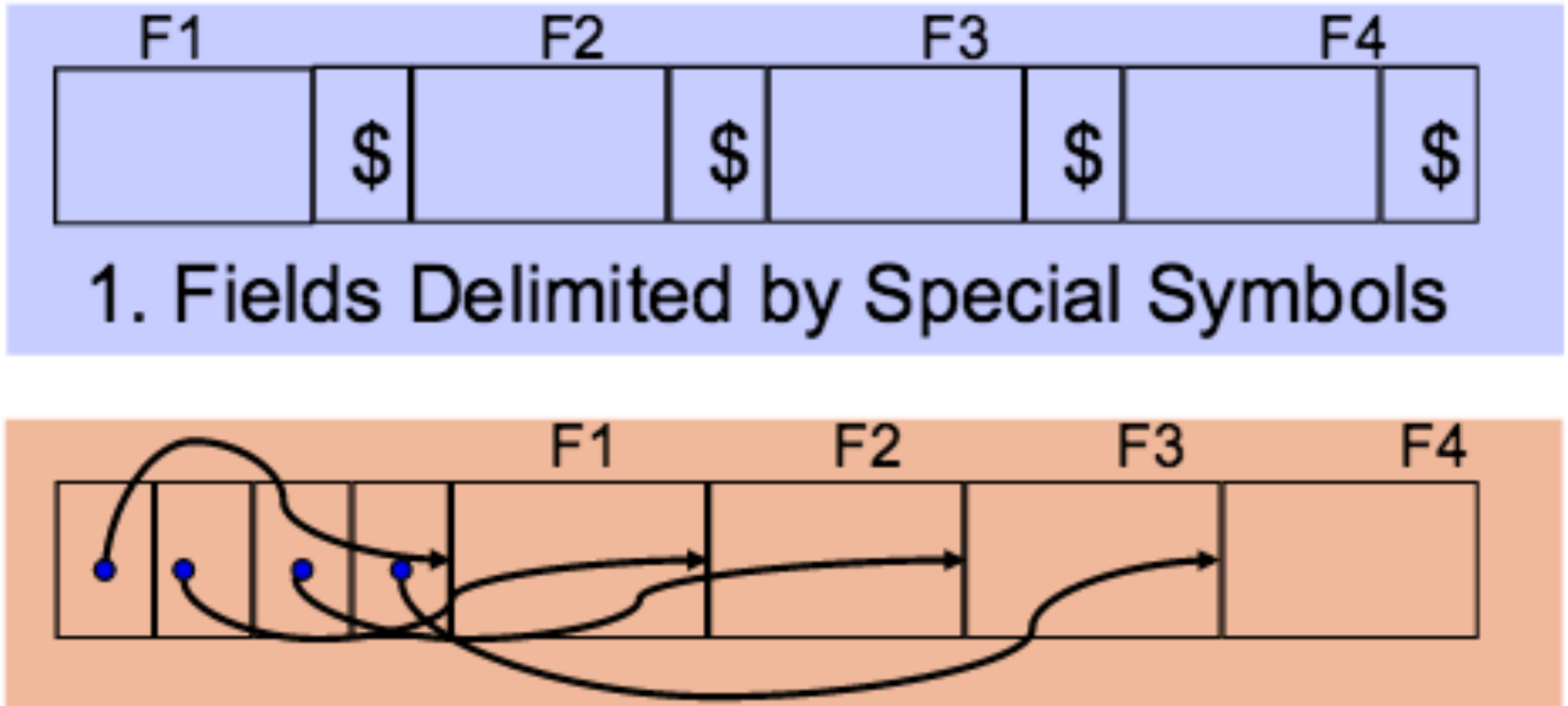


Record id = <page id, slot #>.

In first alternative, moving records for free space management *changes rid*; may be problematic!

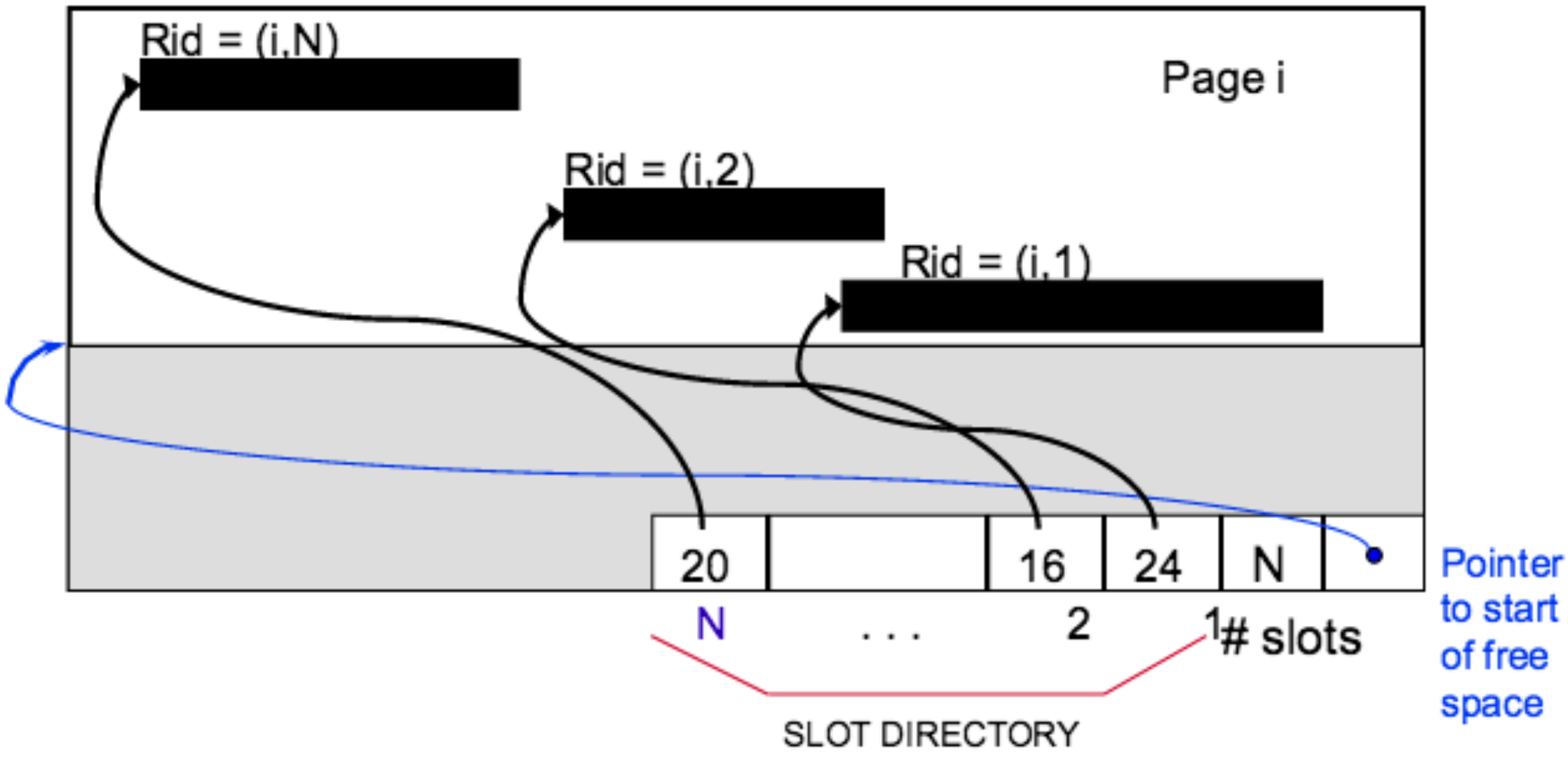
Record Formats: Variable Length

- Two alternative formats (# fields is fixed):



Second offers direct access to i'th field, efficient storage of *nulls* (special *unknown* value); small directory overhead.

"Slotted Page" Format: Variable Length Records



Can move records on page without changing rid!
So, attractive for fixed-length records too.

B = # of data blocks
D = avg. time to read/write disk block
*multiply everything by D!

I/O Costs

Operation	Heap File	Sorted File
Scan all records	B	B
Equality Search	0.5B	log_2(B)
Range Search	B	log_2(B) + # pages matched
Insert	2	log_2(B)+ (B/2) *2
Delete	0.5B+1	log_2(B)+ (B/2) *2

Do second page of worksheet!