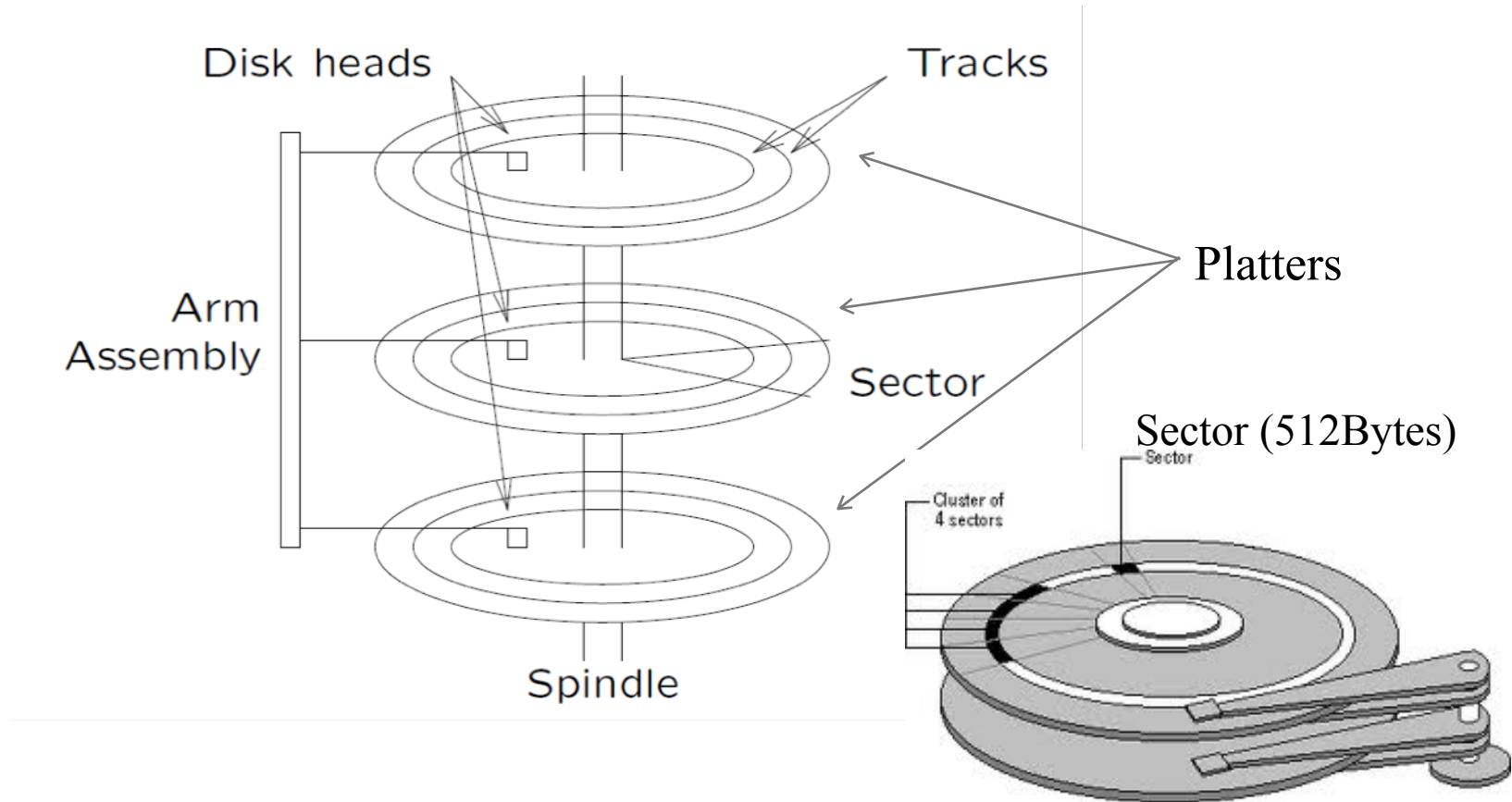


# Storing Data: Disks and Files

# 11.1 Memory Hierarchy

- Primary Storage: main memory.
  - Fast access, expensive.
  - Before processing any query, the data must be in the main memory.
- Secondary storage: hard disk.
  - Slower access, less expensive.
  - Most widely used.
- Tertiary storage: tapes, cd, etc.
  - Slowest access, cheapest.
  - Rarely used.

# 11.2 Disks



- We need to transfer the data from disk to main memory;
- Smallest transferring unit is Block (4KB): If a single record in a block is needed, the entire block is transferred.

# 11.2 Disks

Access time includes:

- seek time (find the right sector, e.g.  $10\text{msec}$ )
- rotational delay (find the right sector, e.g.  $5\text{msec}$ )
- transfer time (read/write block, e.g.  $10\mu\text{sec}$ )

**Random access is dominated by seek time and rotational delay**

- Data Space Management
- Buffer pool

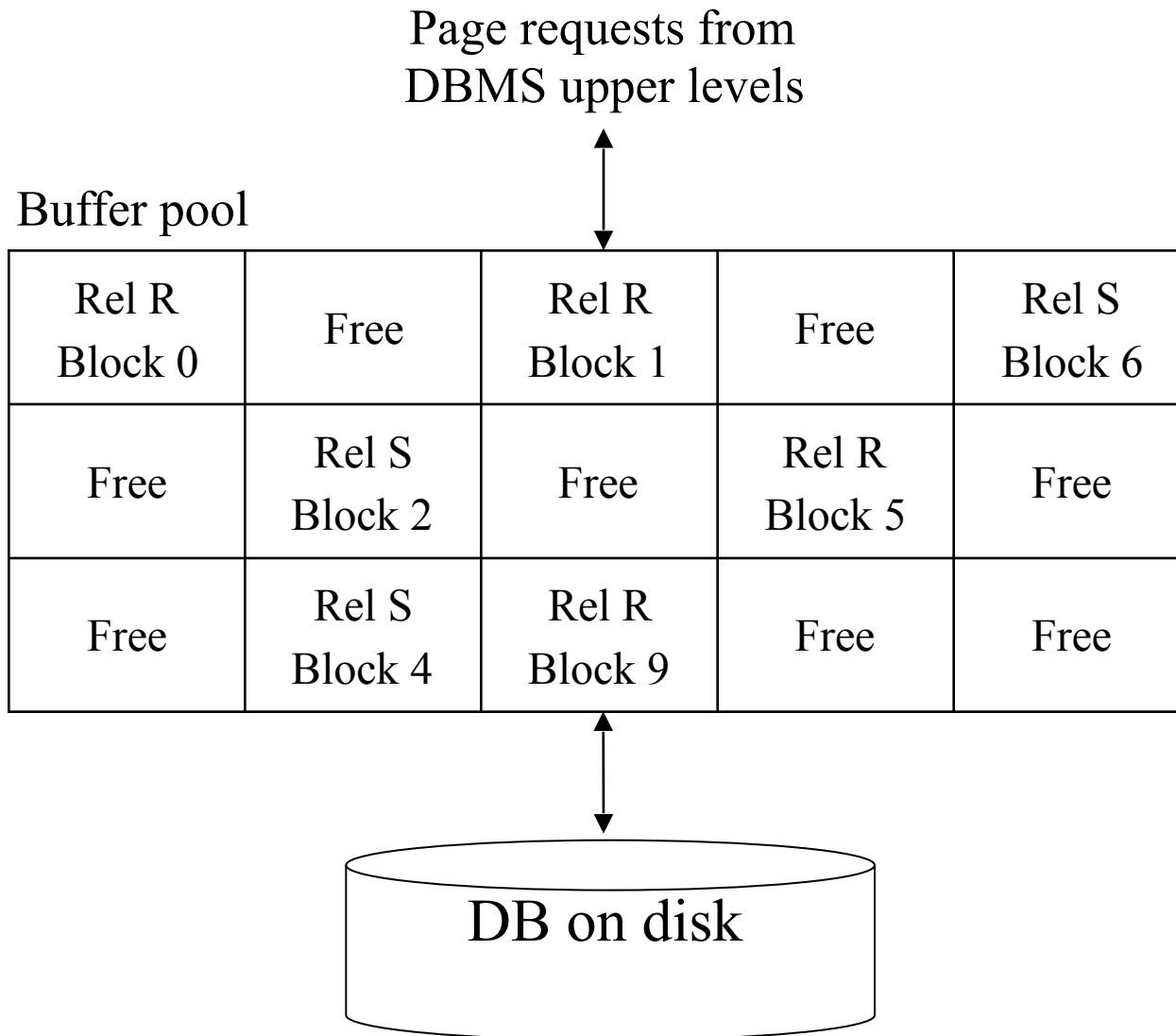
# 11.3 Disk Space Management

- Improving Disk Access:
  - Use knowledge of data access patterns.
    - E.g. two records often accessed together: put them in the same block (clustering)
    - E.g. records scanned sequentially: place them in consecutive sectors on same track
  - Keeping Track of Free Blocks
    - Maintain a list of free blocks.
    - Use bitmap.

# 11.4 Buffer Management

- Manages traffic between disk and memory by maintaining a ***buffer pool*** in main memory.
- Buffer pool
  - Collection of *page slots* (frames) which can be filled with copies of disk block data.
  - One page = 4096Bytes = One block

# 11.4.1 Buffer Pool



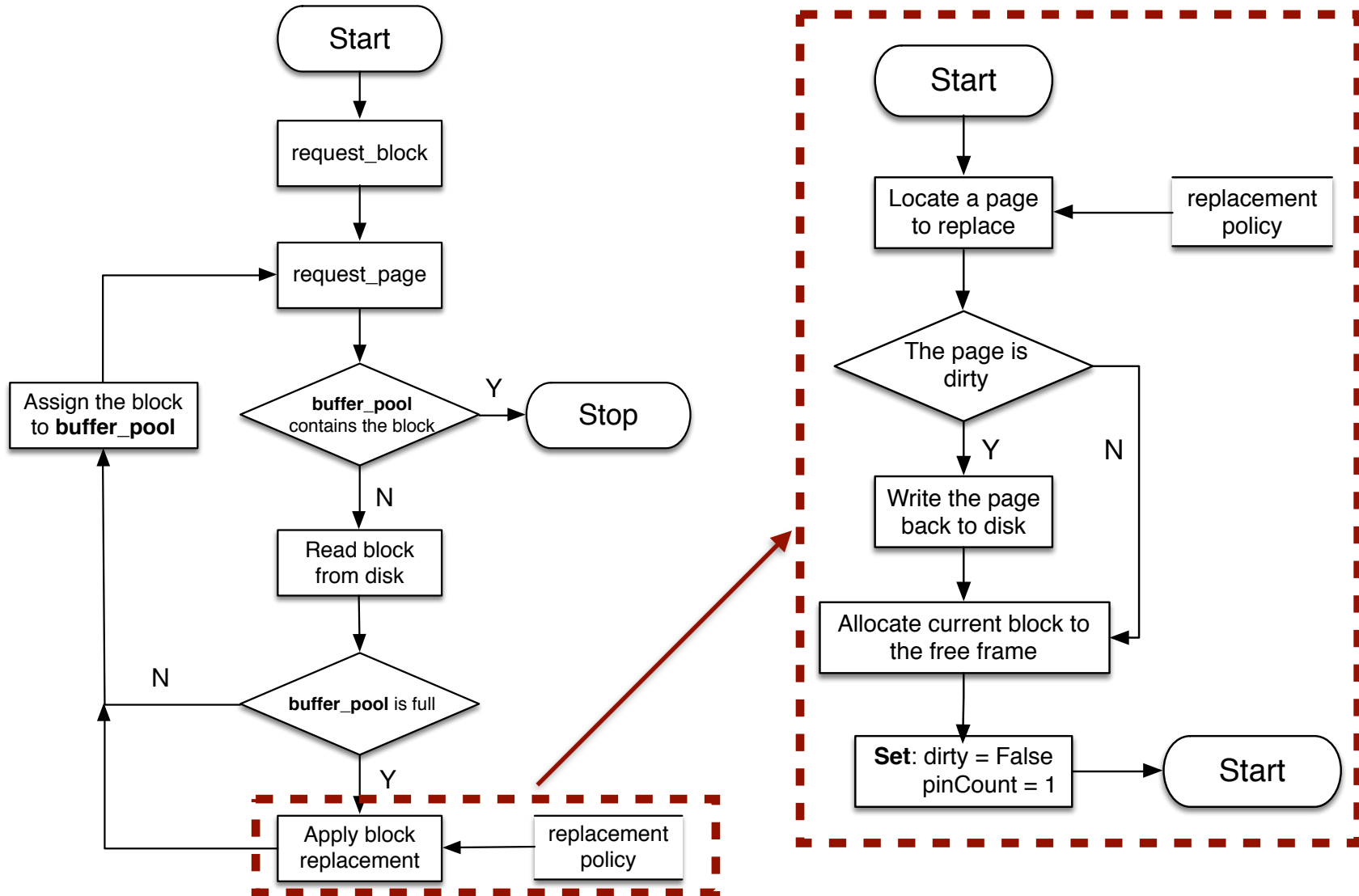
# 11.4.1 Buffer Pool

- The *request\_block* operation:
  - If block *is* already in buffer pool:
    - no need to read it again
    - use the copy there (unless write-locked)
  - If block is *not* already in buffer pool:
    - need to read from hard disk into a free frame
    - if no free frames, need to remove block using *a buffer replacement policy*.
  - The *release\_block* function indicates that block is no longer in use
    - good candidate for removal (or replacing)



# 11.4.1 Buffer Pool

## The *request\_block* Operation



# 11.4.1 Buffer Pool

## The *release\_block* Operation

1. Decrement pin count for specified page.

**Note:** No real effect until replacement required.

## The *write\_block* Operation

1. Updates contents of page in pool (memory)

**Note:** Doesn't actually write to disk, until been replaced, or forced to commit.

# 11.4.1 Buffer Pool

For each frame, we need to know:

- whether it is currently in use
- whether it has been modified since loading (*dirty bit*)
- how many transactions are currently using it (*pin count*)
- (maybe) time-stamp for most recent access

# 11.4.2 Buffer Replacement Policies

- Least Recently Used (LRU)
  - release the frame that has not been used for the longest period.
  - intuitively appealing idea but can perform badly
- First in First Out (FIFO)
  - need to maintain a queue of frames
  - enter tail of queue when read in
- Most Recently Used (MRU)
  - release the frame used most recently
- Random

No one is guaranteed better than the other.

Quite dependent on applications.

## Example1:

**Data pages:** P1, P2, P3, P4

**Queries:**

Q1: read P1; Q2: read P2;

Q3: read P3; Q4: read P1;

Q5: read P2;

**Buffer:**

P1 <sub>Q4</sub>	P2 <sub>Q5</sub>	P3 <sub>Q3</sub>
------------------	------------------	------------------

Q6: read P4:

- **LRU:** Replace P3
- **MRU:** Replace P2
- **FIFO:** Replace P1
- Random: randomly choose one buffer to replace

## Example 2:

**Data pages:** P1, P2, ..., P11

**Queries:**

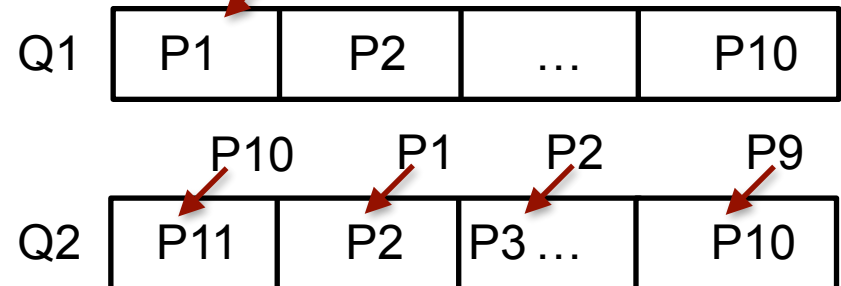
Q1: read P1, P2,..., P11;

Q2: read P1, P2,..., P11;

Q3: read P1, P2,..., P11;

**Buffer:** 10 pages like Example 1

LRU/FIFO: P11



**Boom: We need to get in/out every page**

MRU: Perform the best in this case.

**Practice yourself!!**

# 11.5 Record Formats

Records are stored within fixed-length blocks.

- *Fixed-length*: each field has a fixed length as well as the number of fields.

33357462	Neil Young	Musician	0277
4 bytes	20 bytes	10 bytes	4 bytes

- Easy for intra-block space management.
- Possible waste of space.

- *Variable-length*: some field is of variable length

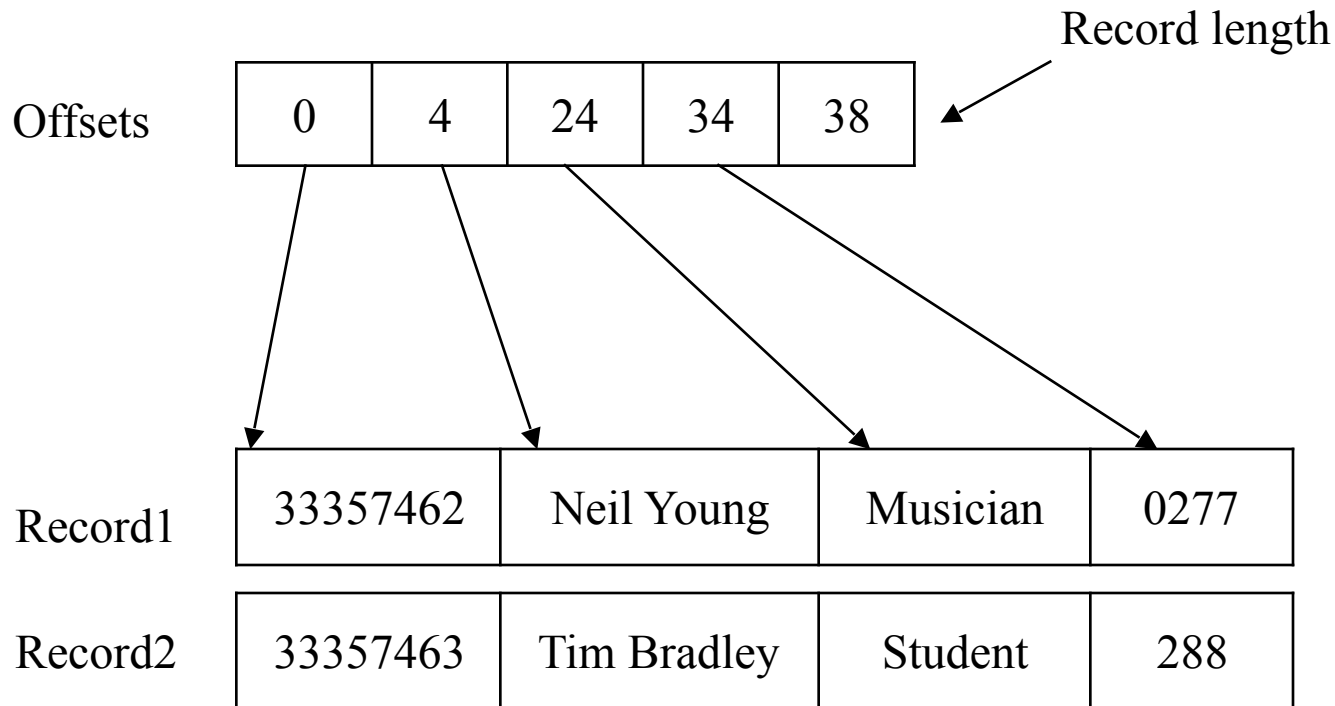
33357462	Neil Young	Musician	0277
4 bytes	10 bytes	8 bytes	4 bytes

- complicates intra-block space management
- does not waste (as much) space.

# 11.5.1 Fixed-Length

Encoding scheme for fixed-length records:

- length + offsets stored in header



# 11.5.2 Variable-Length

Encoding schemes for variable-length records:

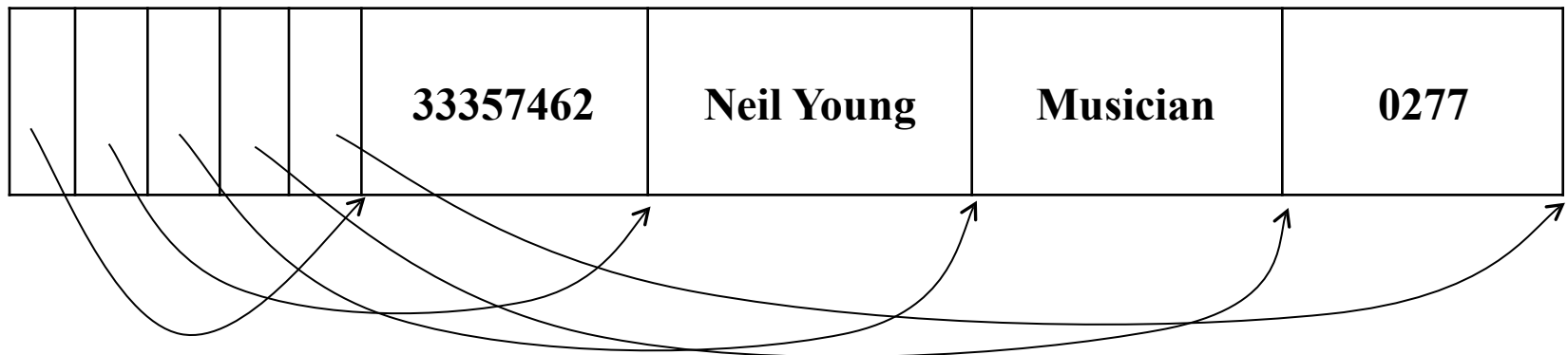
- Prefix each field by length

4 xxxx 10 Neil Young 8 Musician 4 xxxx

- Terminate fields by delimiter

33357462/Neil Young/Musician/0277/

- Array of offsets





# 11.6 Block (Page) Formats

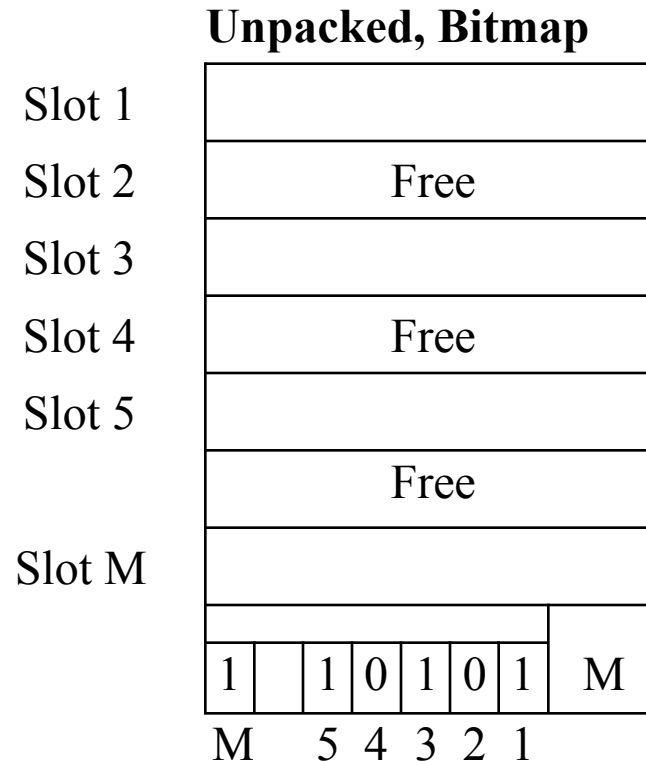
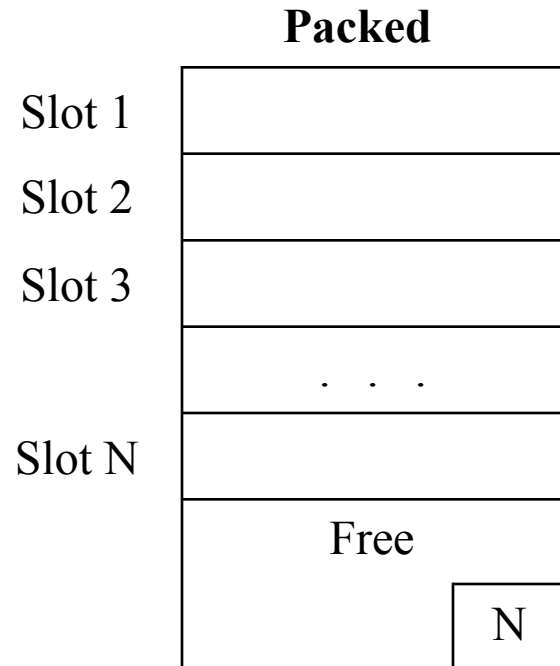
A block is a collection of *slots*.

Each slot contains a record.

A record is identified by  $\text{rid} = \langle \text{page id, slot number} \rangle$ .

# 11.6.1 Fixed Length Records

For fixed-length records, use record slots:



Insertion: occupy first free slot; packed more efficient.

Deletion: (a) need to compact, (b) mark with 0; unpacked more efficient.

# 11.6.2 Variable-Length Records

For variable-length records, use **slot directory**.

Possibilities for handling free-space within block:

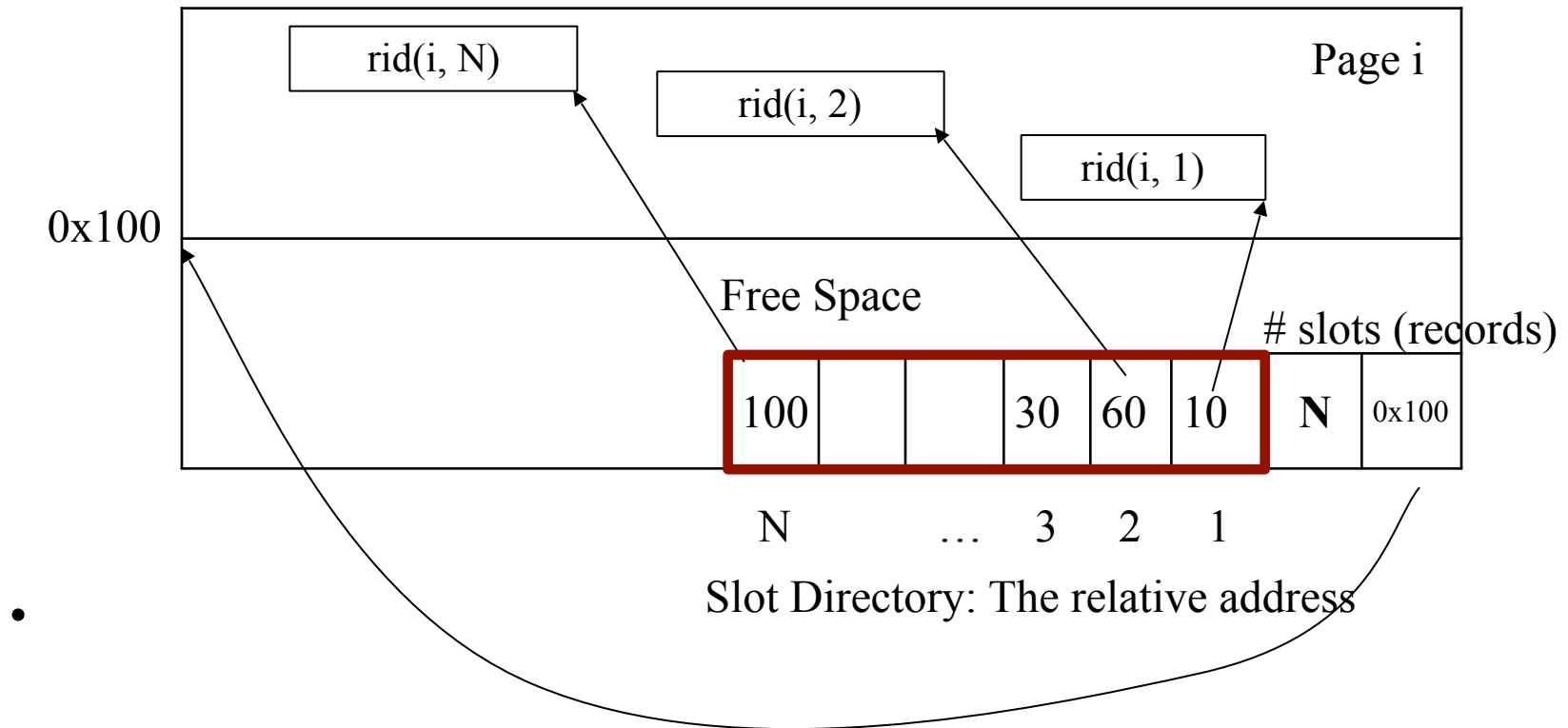
- compacted (one region of free space)
- fragmented (distributed free space)

In practice, probably use a combination:

- normally fragmented (cheap to maintain)
- compact when needed (e.g. record won't fit)

# 11.6.2 Variable-Length Records

- Compacted free space:

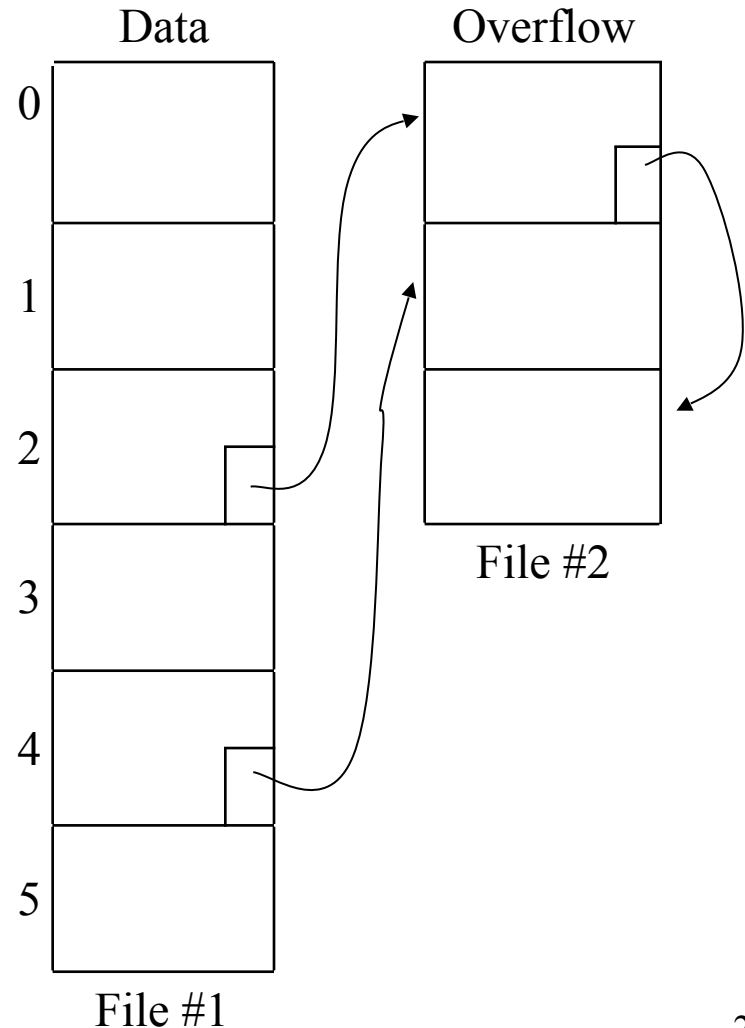


# 11.6.2 Variable-Length Records

- Overflows
  - Some file structures (e.g. hashing) allocate records to specific blocks.
  - What happens if specified block is already full?
  - Need a place to store “excess” records.

# 11.6.2 Variable-Length Records

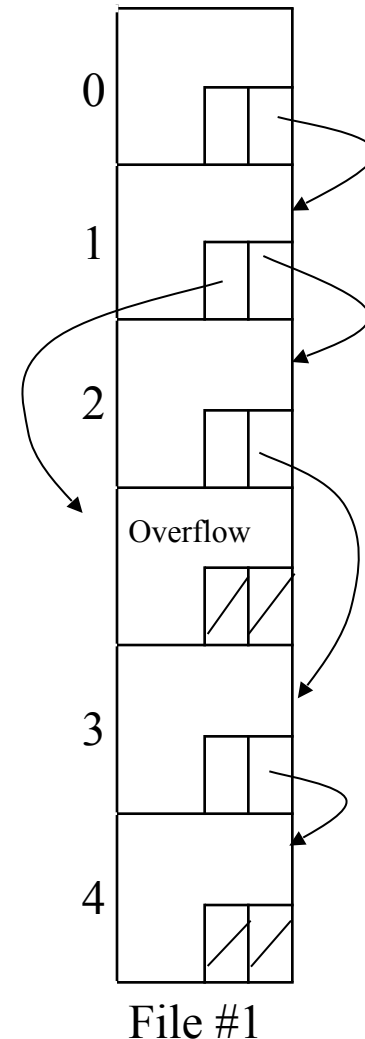
- Overflow blocks in a separate file:
- Note: “pointers” are implemented as file offsets.



# 11.6.2 Variable-Length Records

- Overflow blocks in a single file:
- Not suitable if accessing blocks via offset (e.g. hashing).

Data + overflows



# 11.7 Files

*A file* consists of several data blocks.

*Heap Files*: unordered pages (blocks).

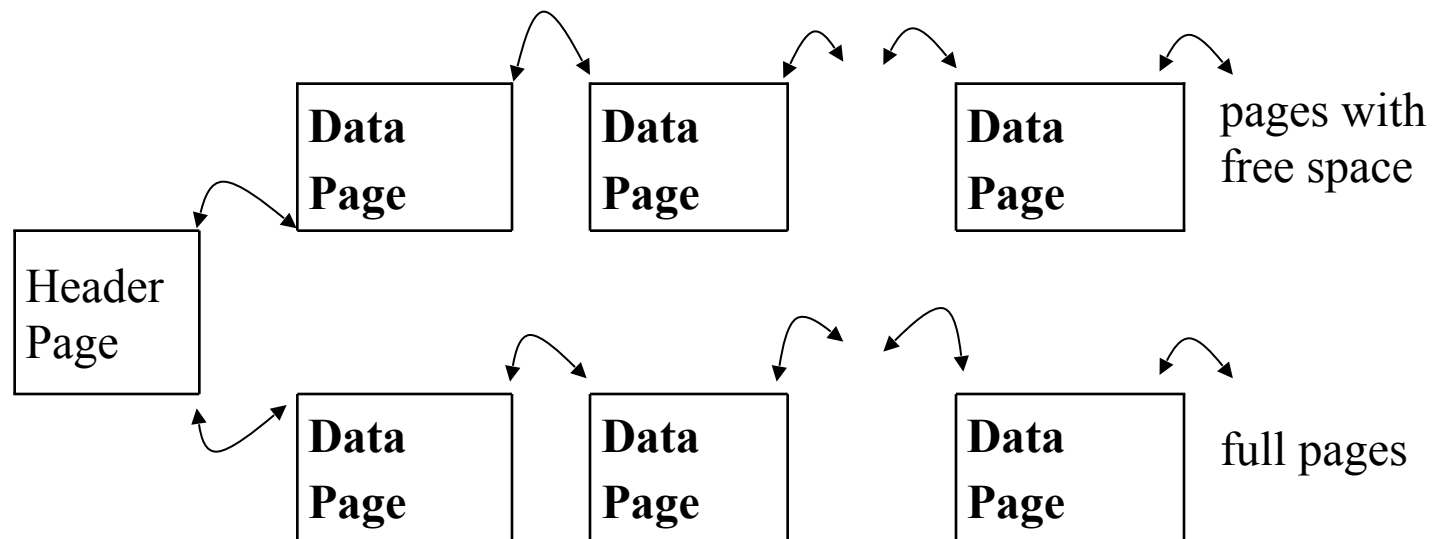
Two alternatives to maintain the block information:

- Linked list of pages.
- Directory of pages.



# 11.7.1 Linked List of Pages

- Maintain a heap file as a doubly linked list of pages.



Organised by a Linked List

- Disadvantage:** To insert a record, several pages may be retrieved and examined.

# 11.7.2 Directory of Pages

Maintain a directory of pages.

- Each directory entry identifies a page (or a sequence of pages) in the heap file.
- Each entry also maintains a bit to indicate if the corresponding page has any free space.

