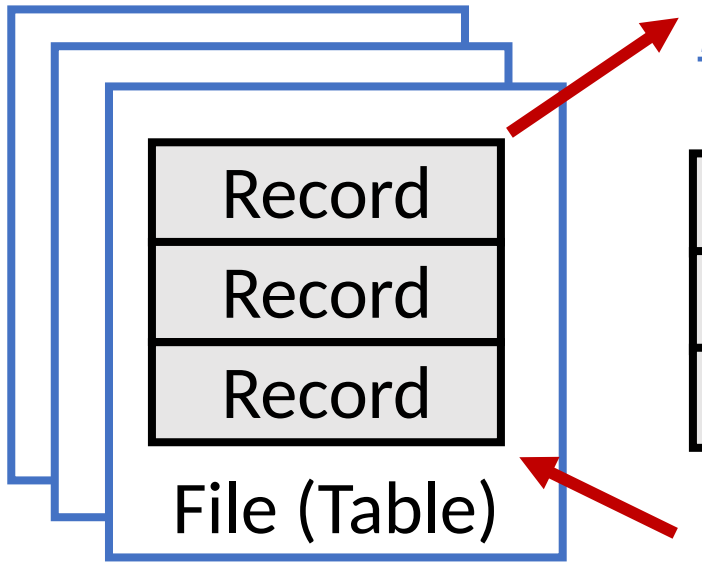
CSE 541: Database Systems I

Data Organization

Slides Credit: Tianzheng Wang

Organizing Data on Block Devices

- *Database:* a collection of tables (files)



Record: collection of fields

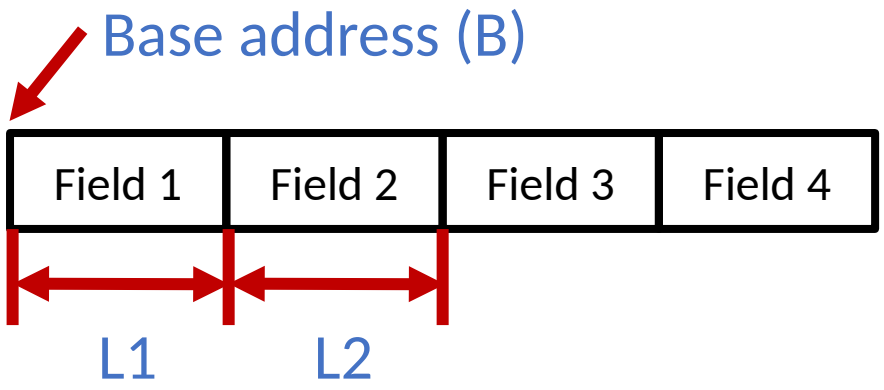
StudentID	Name	GPA
100	John	3.0
101	Hao	3.0
102	May	3.5

Table: collection of records

- Field ⊏ Record ⊏ Table ⊏ Database ⊏ Storage device
- Stored permanently in storage device as fixed-size pages

Placing Fields in Records

- **Fixed-Length Records**: each field has the same length



Address of Field 3

= B + L1 + L2

Placing Fields in Records

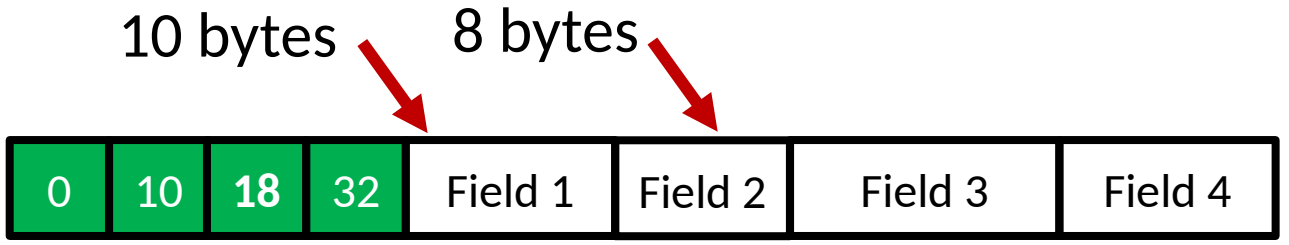
Variable-Length Records: fields may have different lengths

But the number of fields is fixed – determined by schema

Alternative 1: Store all fields consecutively with delimiters

- Requires a sequential scan to find a specific field

Alternative 2: Use an array of field offsets to locate fields



Offset for Field 3: 10 + 8 = **18**

Storing Records in Pages

- The size of each page is fixed, e.g., 4KB, 8KB
  - Multiply of block device blocks.

- Records can be **variable** or **fixed** length
- Each page contains
  - A page header
  - A collection of slots, each slot stores a record
- Each record identified by a unique Record ID (RID)
  - RID: `<Page ID, Slot Number>` Pair

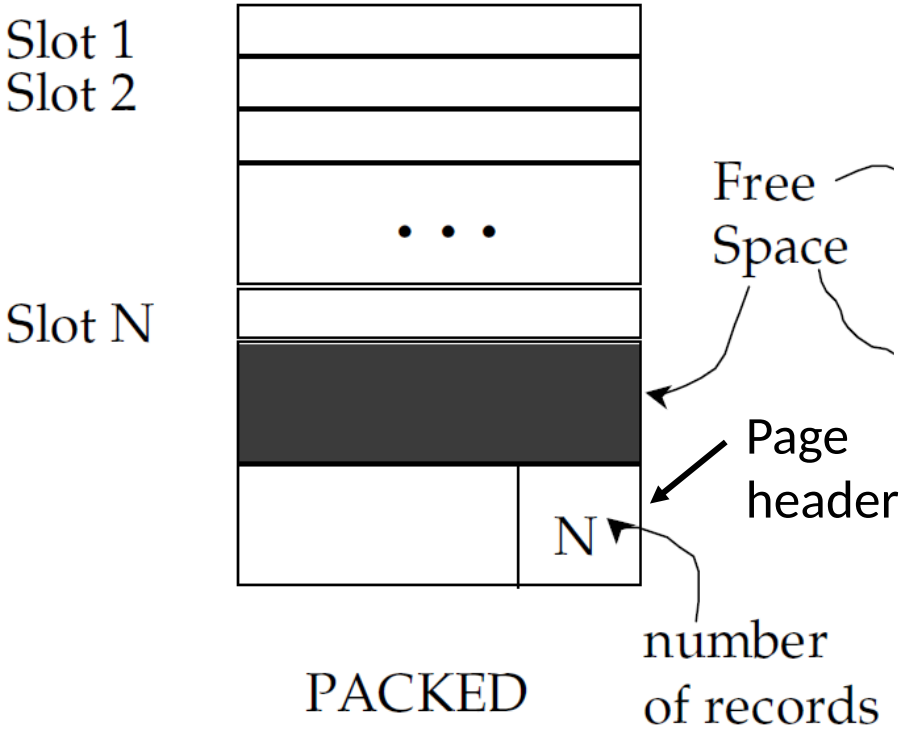
Page Format: Fixed-Length Records

All records are fixed-length in the page

Alternative 1: Packed Page

- Offset arithmetic to find a record

- **What's the problem?**
  - Move subsequent records upon deletion
  - RID may change as records are moved
    – complicates external references



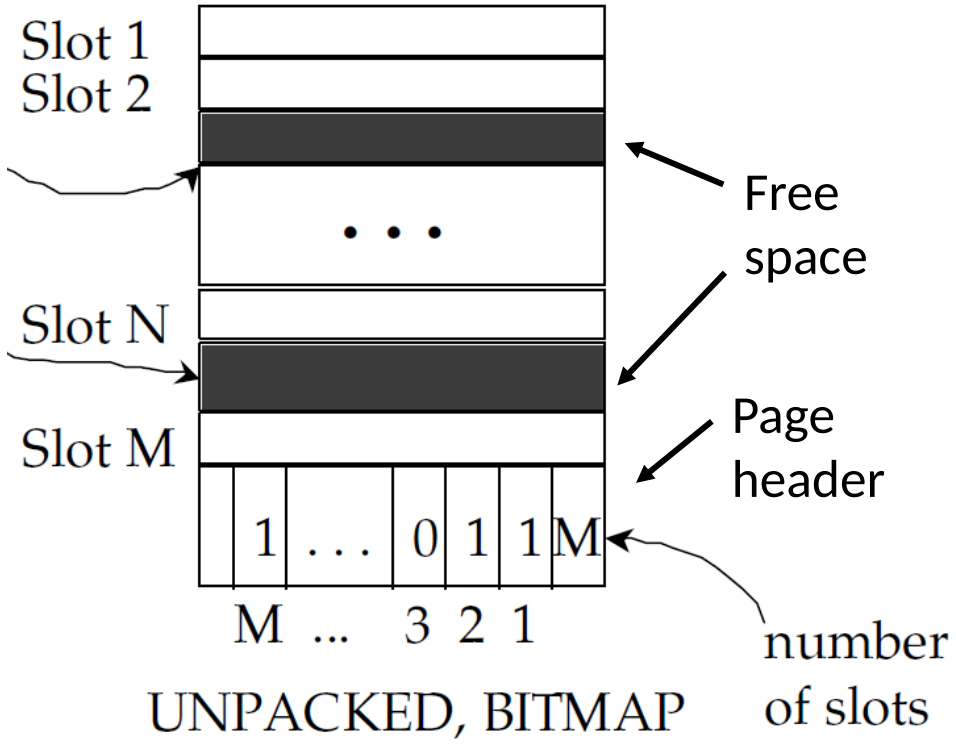
Page Format: Fixed-Length Records

All records are fixed-length in the page

Key Problem: handling record deletions

Alternative 2: Bit Array

- Scan the bit array to locate a record
- One bit per slot to denote whether slot contains a record: bit-on means record alive
- Toggle-off the bit upon deletion



Page Format: Variable-Length Records

Records could be of any size (though usually smaller than a page)

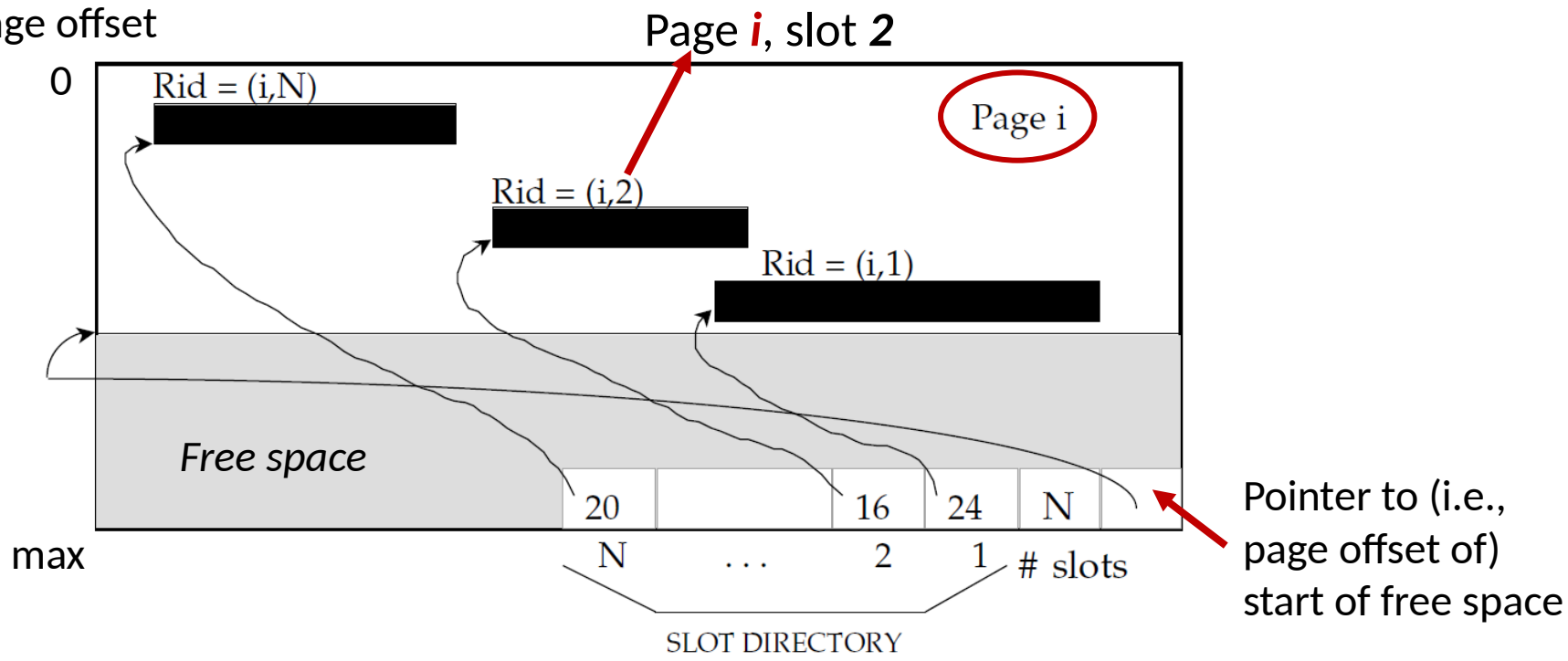
- Need to allocate just the right amount of space for a record
  - Space too big – waste space
  - Space too small – not enough to store the record
- Records may grow or shrink
  - Important to allow records to be moved easily

- Solution: Maintain a slot directory in each page
  - “Slotted page structure”

Page Format: Variable-Length Records

Slotted page structure

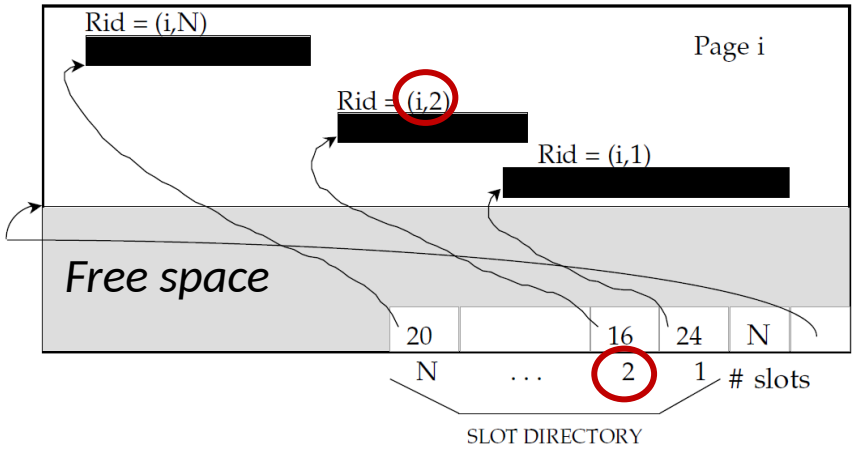
- Each slot entry contains record offset
  - I.e., offset in bytes from start of data area to start of record
  - Also contain record length (not shown in figure)
- Records can be moved without changing RID (just modify slot entry)



Page Format: Variable-Length Records

Slotted page structure

- Flexible, but more complex

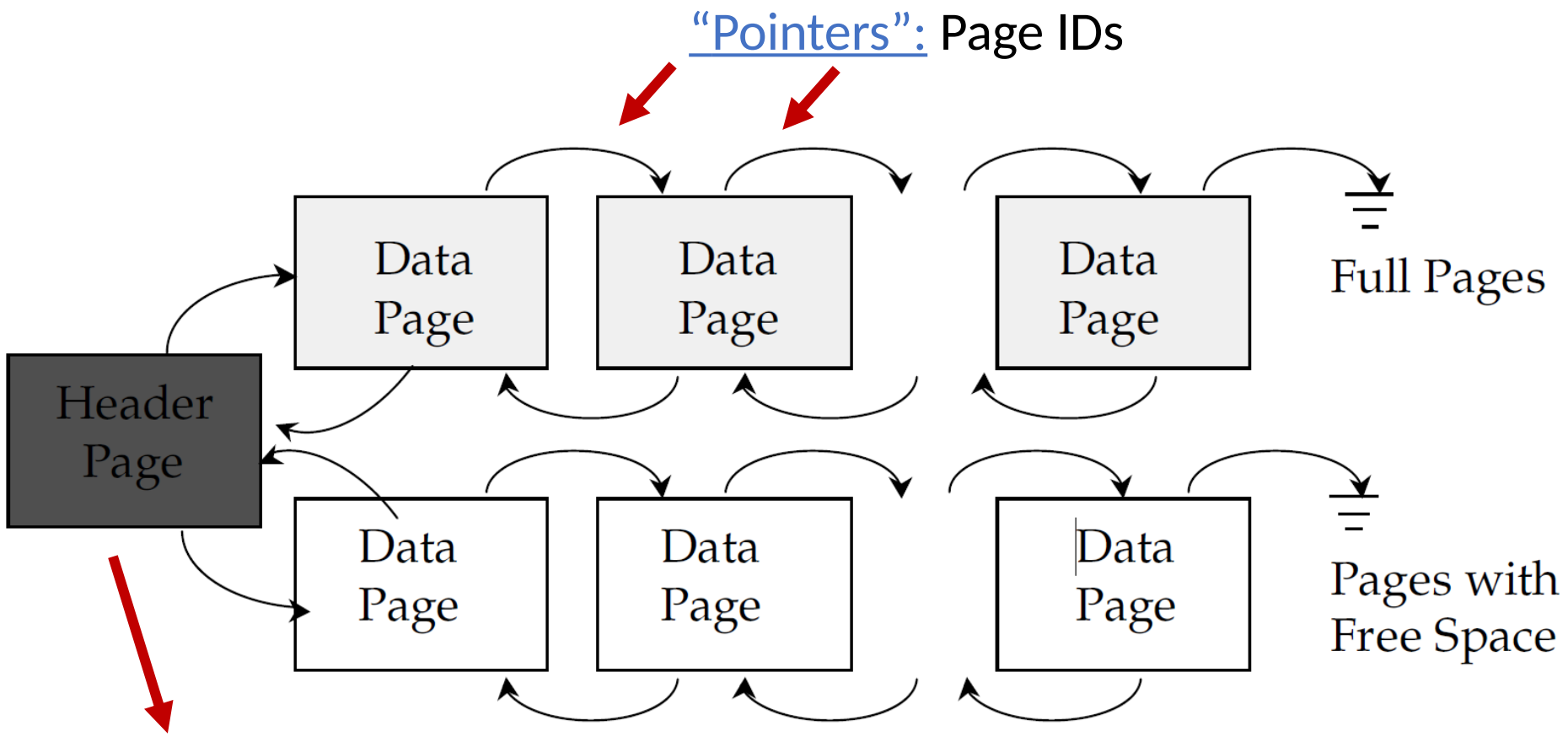


- Delete operation
  - Decrement the number of “valid” slots
  - Change the offset/size in slot entry to invalid
  - But cannot remove the slot entry – RIDs will all change
    - Exception: the record being deleted is the last entry
- Insert operation
  - Try to reuse previously created slot (record deleted)
  - Need to scan existing slot entries

Organizing Pages into a Heap File

- Consists of fixed-size pages
- Pages are allocated and deallocated as the file grows and shrinks
- Pages are unordered in the heap file
  - Only guarantees we can find all pages
- A heap file must support the following operations
  - Insert a record, delete a record, update a record
  - Search for a record, scan for a range of records
- Key issues
  - Keep track of all pages **(Scan)**
  - Keep track of free space **(Insert/Delete)**

Heap File Alternative 1: Linked List

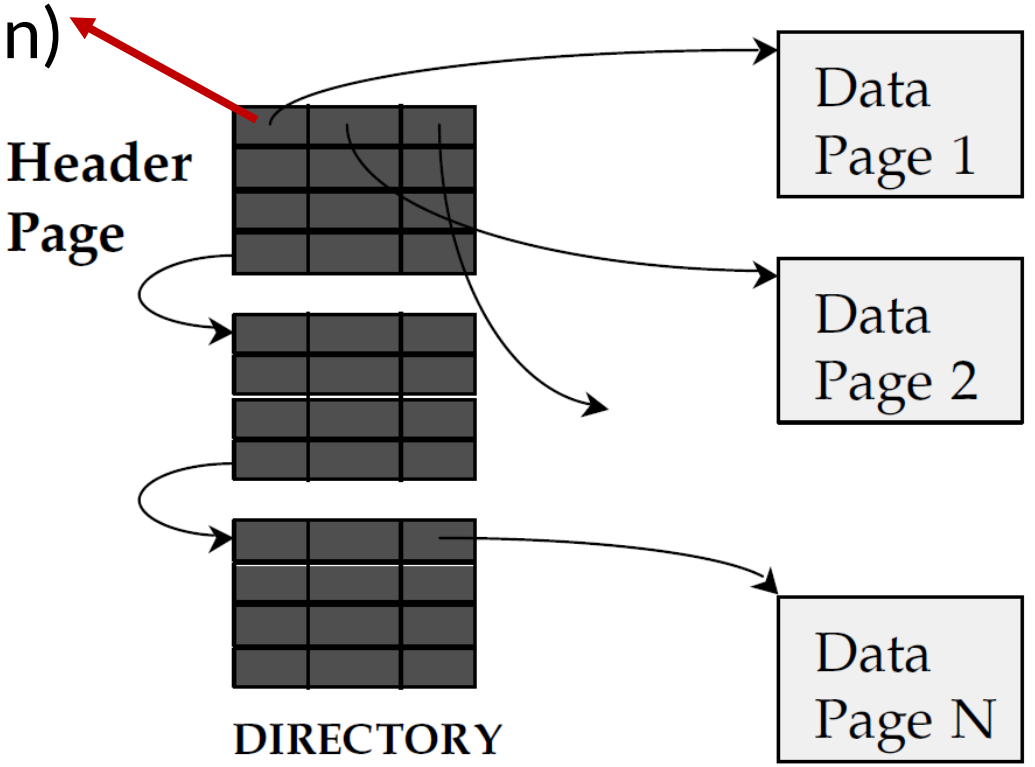


Header page and file name stored in "well-known" locations

- E.g., using empty files (reliable: file metadata is typically journaled by the file system)

Heap File Alternative 2: Directory

Directory entry: Identifies a page (may include the page's free space information)



Directory:

- Consists of directory pages
  - May be a file by itself
  - May be part of the "main" file
- Much smaller than the linked list in Alternative 1

Putting Things Together

- Field → Records → Table → Database → File → Storage Device

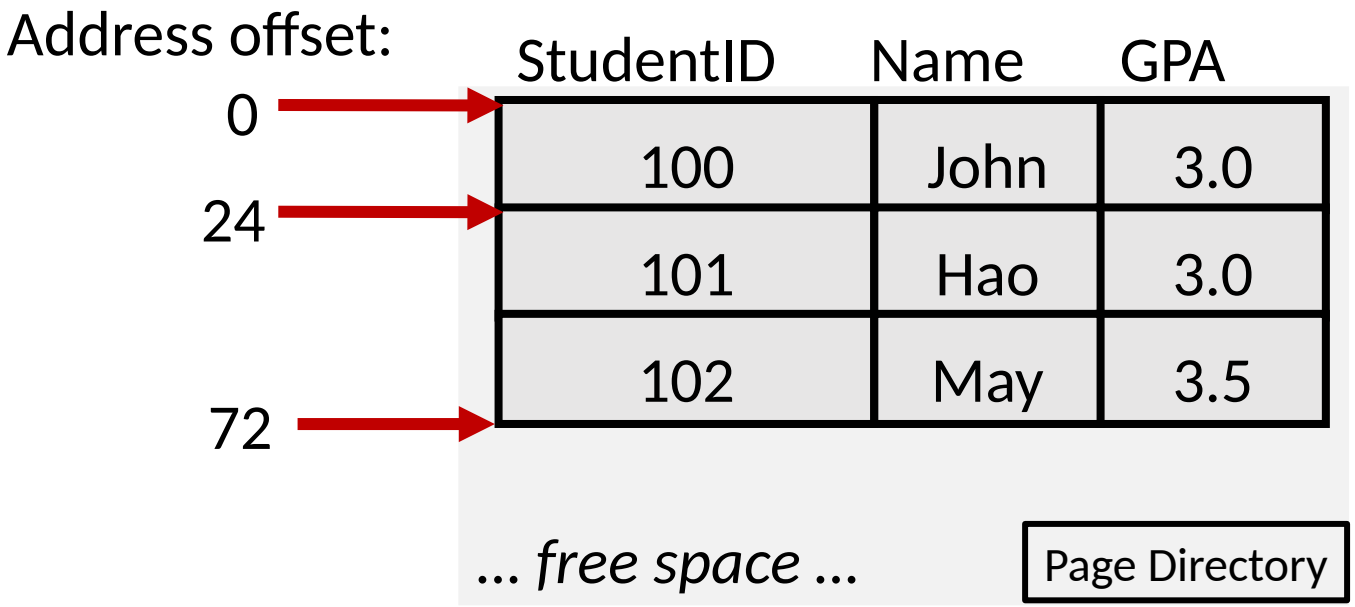
- How do you think DMBS organize multiple DBs and Tables?
- Is an FS essential for DBMS?
- Where and how DBMS store meta-info (e.g. Schema, etc.)?

- Does it have to be record/row oriented?
- Is there other ways to organize “data → page → file”?

Row-Oriented Storage

Records stored one by one in page space

- Aka N-ary storage model (NSM)
- Cheap to get all fields in a row by a sequential read
- Good if we need to access many fields in a record

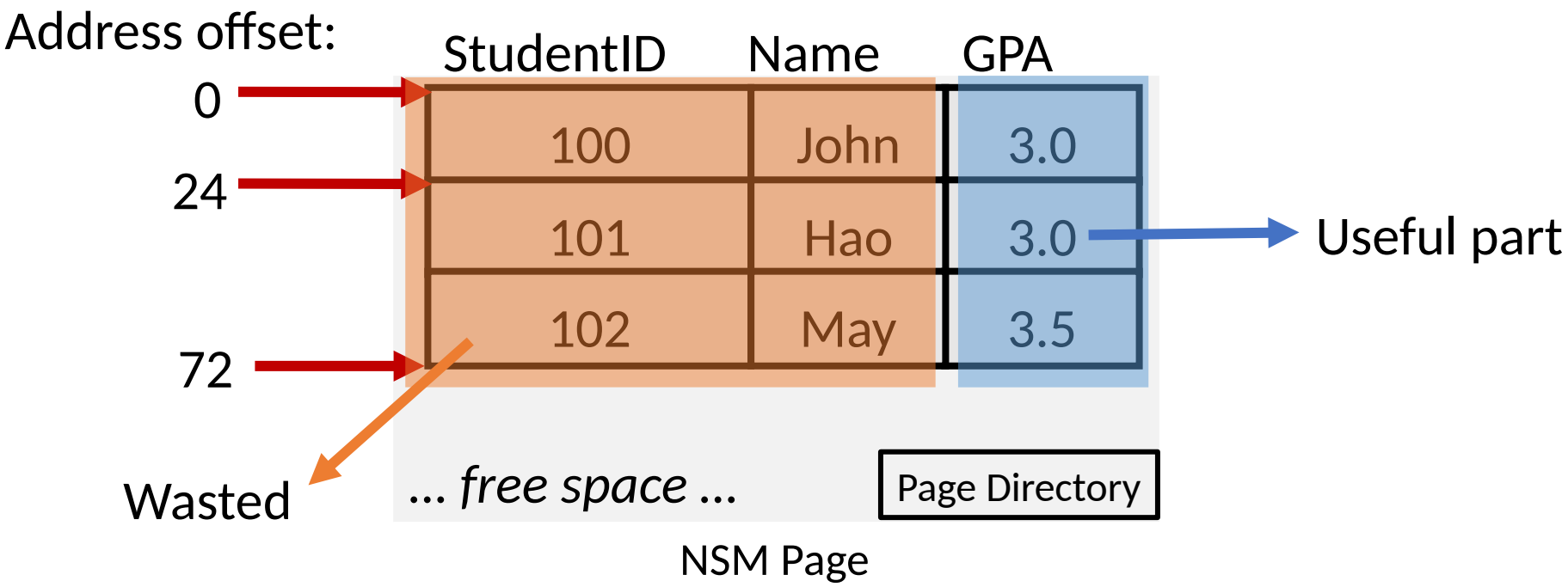


NSM Page

Wasted I/O in Row-Oriented Storage

Many workloads only look at a few fields

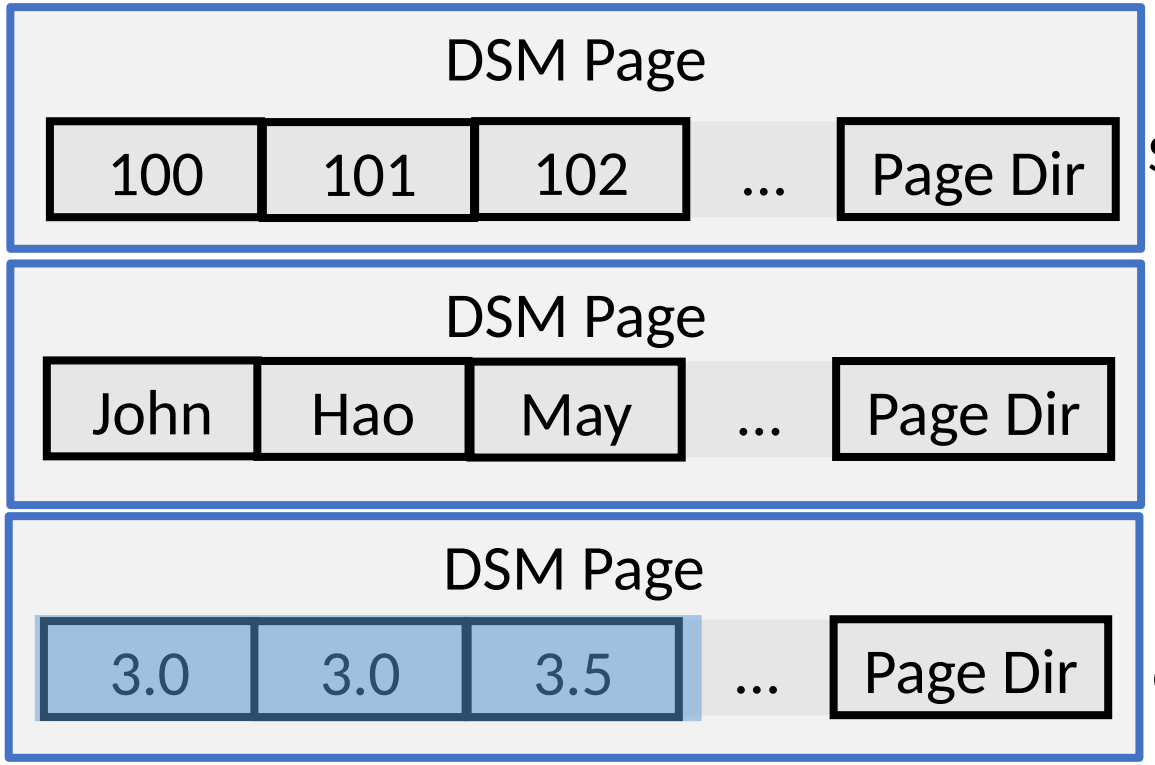
- Analytics, decision making, etc.
  - Example: Get all GPAs
- Many useless I/Os for unused fields



Column-Oriented Storage

Co-locate by fields, instead of rows

- Aka “Decoupled Storage Model” (DSM)



Sub-relations:

- StudentID
- Name
- GPA

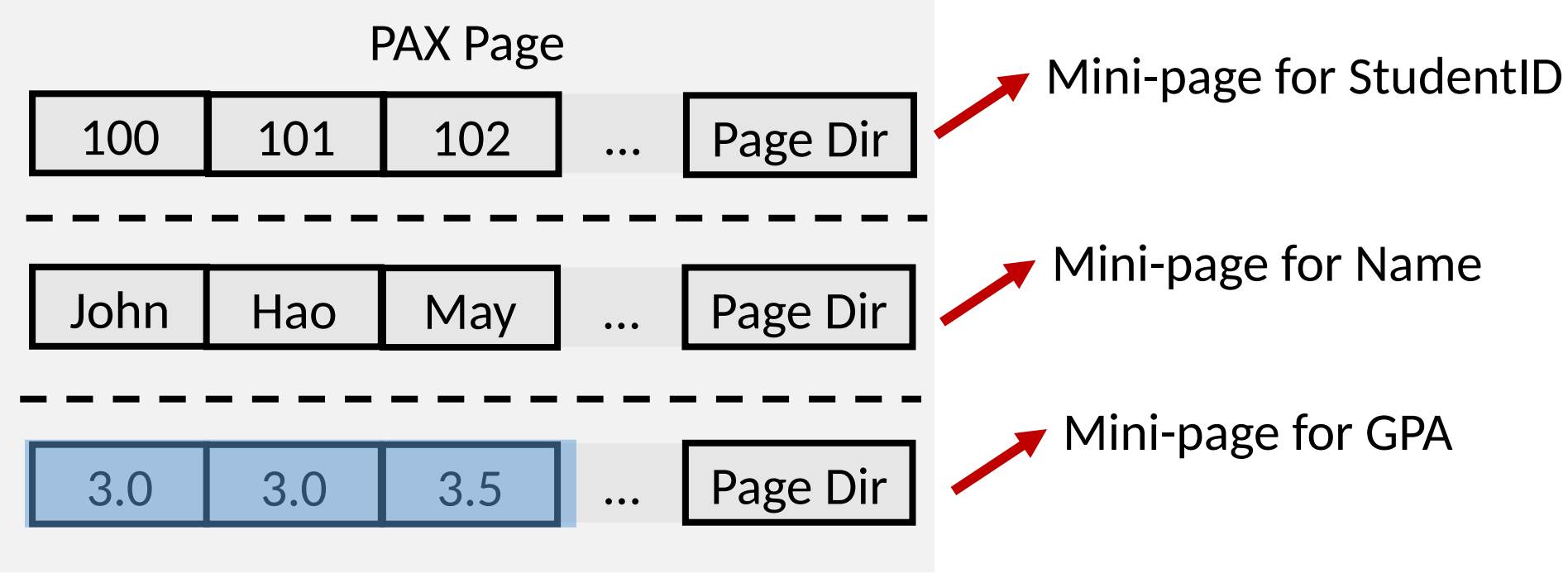
Scan GPA sub-relation only, no wasted I/O

- Getting multiple attributes is more complex, need additional I/O

Partition Attributes Across (PAX)

Hybrid NSM and DSM → Approach best of both worlds.

- Each page partitioned into mini-pages
- One mini-page per attribute, i.e., DSM inside a mini-page



Summary

- Traditional RDBMS: **IO on the critical path**
- Organizing records into pages
  - Fixed-length: packed and bit array
  - Variable-length: using a slot directory
- Organizing pages into heap files
  - Using linked list
  - Using directory pages (directory itself may be a heap file)
- Row- vs. column-oriented storage
  - Row store: good for workloads touching full records
  - Column store good for analytics focusing on columns
    - More work to get full records
  - Hybrid approach may be able to achieve benefits of both