Managing and Accessing Records of a Table

Managing the tuples of a relation

- File: Collection of pages
  - For instance: Pages hold records of one relation
- Management Interface:
- Insert
  - `INSERT INTO Students VALUES (23, ‘Bertino”,2016,… )`
  - Typically returns a rid

dont mix relations records in a page

Retrieving the tuples of a relation

- Get a record by rid
  - Returns record
  - (or rather returns the start position of record in main memory

More general

- Scan over all records

  -
    ```
    SELECT * FROM Students
    ```
- Point Query

  -
    ```
    SELECT * FROM Students WHERE sid = 100
    ```
- Equality Query

  -
    ```
    SELECT * FROM Students WHERE starty = 2015
    ```
- Range Search

  -
    ```
    SELECT * FROM Students
       WHERE starty > 2012 and starty <= 2014
    ```

Retrieval and management

-
  ```
  DELETE FROM Students WHERE sid = 100
  ```

-
  ```
  DELETE FROM Students WHERE endyear < 1950
  ```

} search then delete

Managing relation in a file

- As file grows and shrinks, disk pages are allocated and de-allocated.
- 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
- There are many alternatives for keeping track of this.

Cost Model for Execution

❑ How should we estimate the costs for executing a statement?

- ☆ Number of I/Os
- ☆ CPU Execution Cost
- ☆ Network Cost in distributed system (ignore for now)

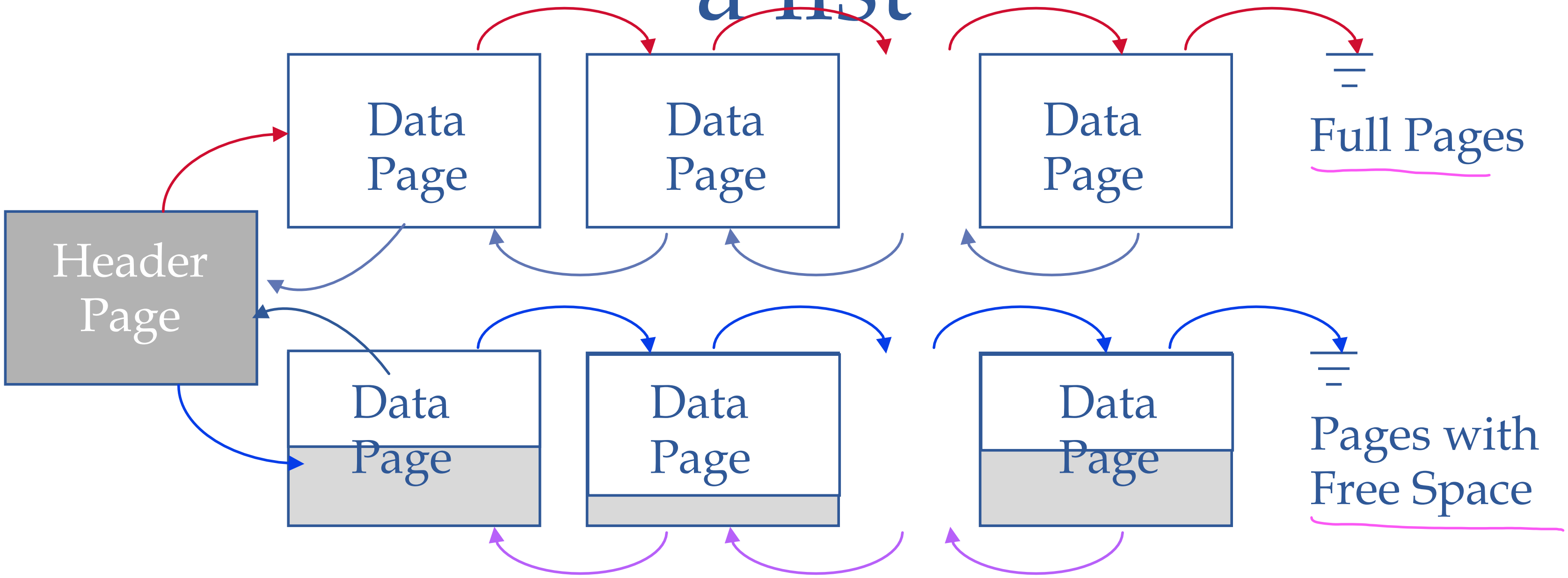
☆ Assumption in this course

- ☆ I/O cost >>> CPU cost
- ☆ Real systems also consider CPU

❑ Simplifications

- ☆ only consider disk reads (ignore writes -- assume read-only workload)
- ☆ only consider number of I/Os and not the individual time for each read (ignores page pre-fetch)
- ☆ Average-case analysis; based on several simplistic assumptions.
  - ☛ Good enough to show the overall trends!

Unsorted heap file implemented as a list



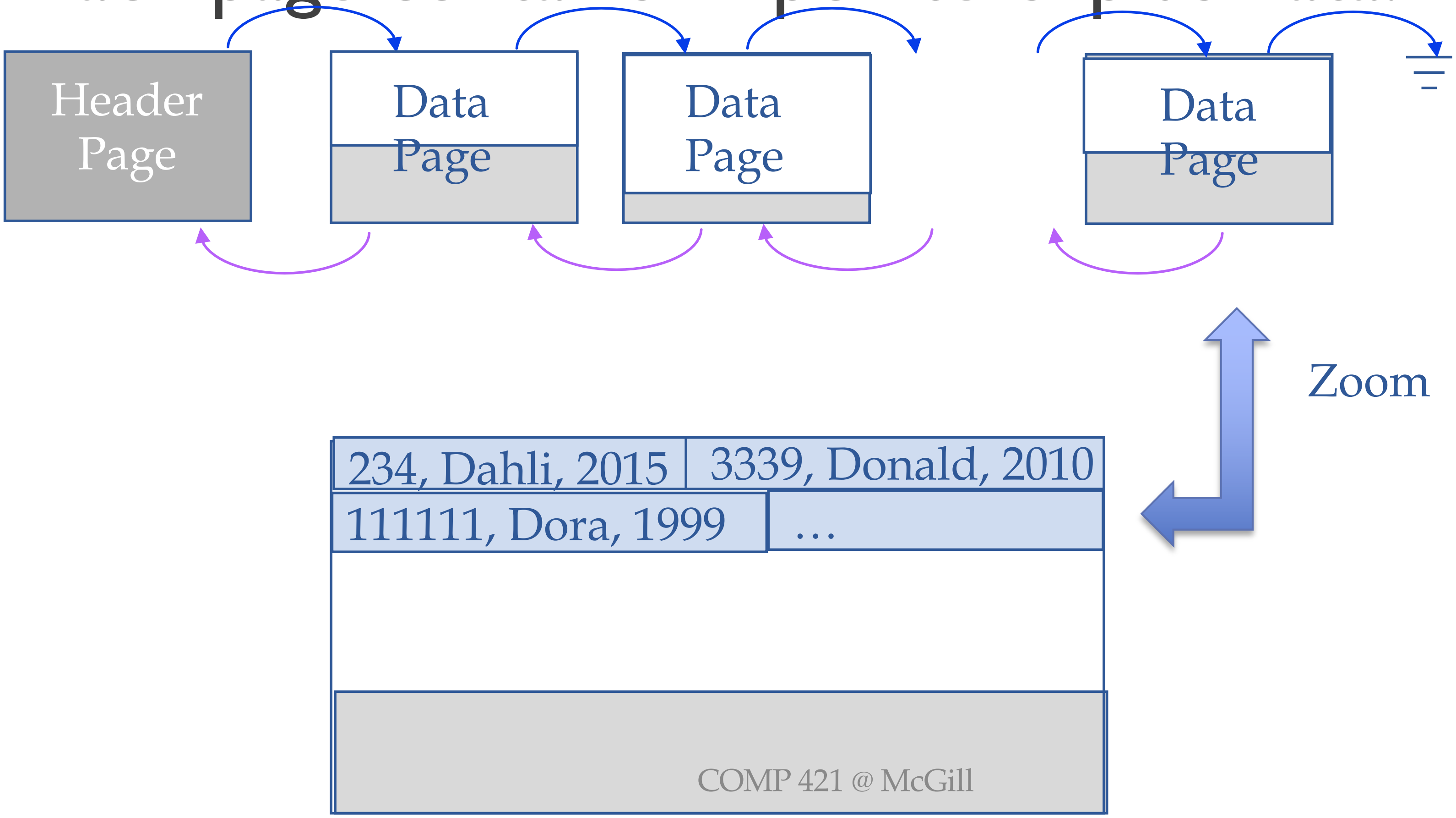
- The header page id and Heap file name must be stored someplace.
- Each page contains 2 `pointers' plus data.

Heap File

- ☆ Linked, unordered list of all pages of the file
- ☆ How well does it support the different operations?
  - ● insert
    - ▲ Insert in any free page
    - ▲ Cost is low (insert anywhere) ←
  - ● scan retrieving all records (SELECT *)?
    - ▲ go from page and page and return each tuple
    - ▲ Not much optimization possible have to load all pages one at a time
    - ▲ But tuples can be stored in a compact way ←
  - ● Point query on unique attributes ← find specific record (could be anywhere)
    - ● Go from page to page, look at each record and return once record is found
    - ▲ have to read on avg. half the pages to return one record
  - ● range search or equality search on non-primary key
    - ▲ have to read all pages to return subset of records.
  - ● delete/update
    - ▲ same as for equality/range search -- depends on WHERE clause

Sorted file

- Records are sorted by one of the attributes (e.g., name).
- Each page contains 2 `pointers' plus data.



234, Dahli, 2015 | 3339, Donald, 2010
111111, Dora, 1999 | …

Sorted File

- insert
  - have to find proper page
  - Algorithm to find proper page? ➔ binary search in log2(number-of-pages)
  - overflow possible
  - Keep empty space on each page → less compact that unsorted heapt
- scan retrieving all records (SELECT *)?
  - you have to retrieve all pages anyways
- Equality/point search on sort attribute
  - find first qualifying page with binary search in log2(number-of-pages)
- range search on sort attribute
  - find first qualifying page with binary search in log2(number-of-pages); adjacent pages might have additional matching records
- delete/update
  - finding tuple same as equality/range search depending on WHERE clause
  - update itself might lead to restructuring of pages
- Sorted output: (ORDER BY)
  - good if on sorted attribute
- Search/sort on attributes other than sort attribute
  - Similar to unsorted heap