L17-23: Query Processing & Database Internals

CS3200 Database design (sp18 s2)

https://course.ccs.neu.edu/cs3200sp18s2/

3/19/2018

L17: The I/O model and External Sort

CS3200 Database design (sp18 s2)

https://course.ccs.neu.edu/cs3200sp18s2/

3/19/2018

I/O model and external sort

1) Buffer
2) External sort
3) External merge

1. The Buffer

Transition to Mechanisms

1. So you can understand what the database is doing!
   - Understand the CS challenges of a database and how to use it.
   - Understand how to optimize a query

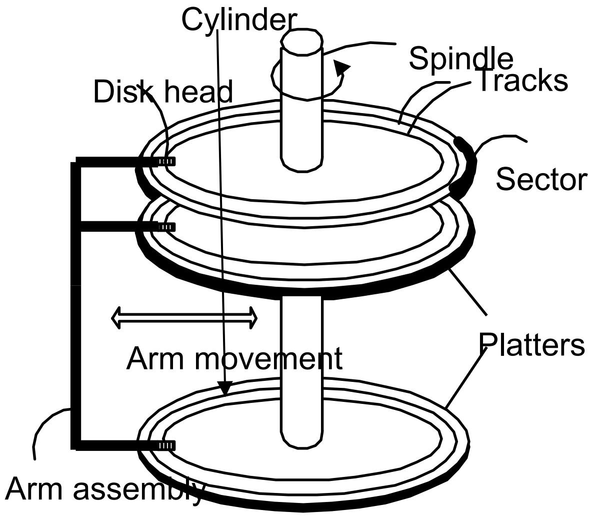
2. Many mechanisms have become stand-alone systems
   - Indexing to Key-value stores
   - Embedded join processing
   - SQL-like languages take some aspect of what we discuss (PIG, Hive)

What we will learn about next

- RECAP: Storage and memory model
- Buffer primer

High-level: Disk vs. Main Memory

Disk



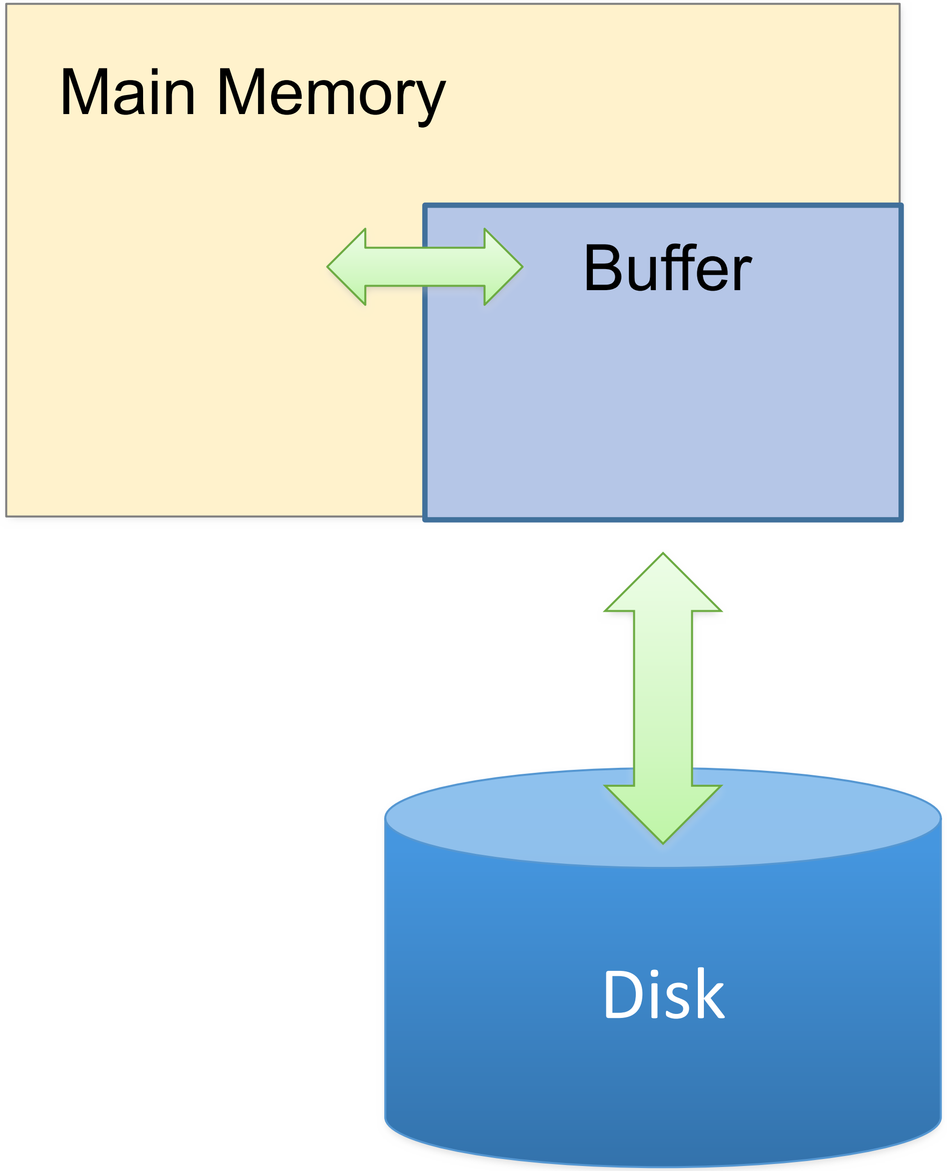
Random Access Memory (RAM) or Main Memory:

- **Slow**: Sequential block access
  - Read a blocks (not byte) at a time, so sequential access is cheaper than random
  - Disk read / writes are expensive!
- **Durable**: We will assume that once on disk, data is safe!
- **Cheap**

- **Fast:** Random access, byte addressable
  - ~10x faster for <u>sequential access</u>
  - ~100,000x faster for <u>random access!</u>
- **Volatile:** Data can be lost if e.g. crash occurs, power goes out, etc!
- **Expensive:** For $100, get 16GB of RAM vs. 2TB of disk!

The Buffer

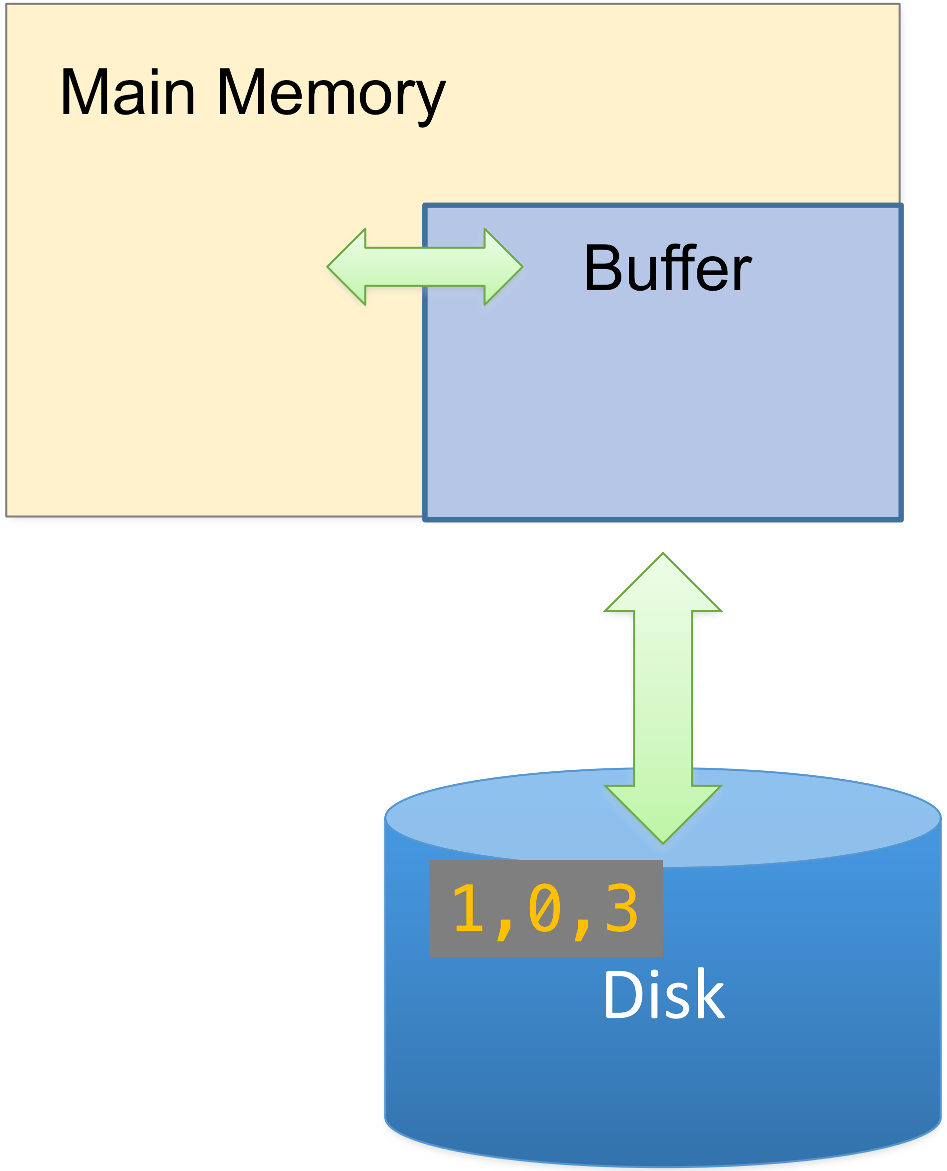
- A **buffer** is a region of physical memory used to store temporary data
    - In this lecture: a region in main memory used to store intermediate data between disk and processes
- Key idea: Reading / writing to disk is slow- need to cache data!



The (Simplified) Buffer

- In this class: We’ll consider a buffer located in main memory that operates over pages and files:

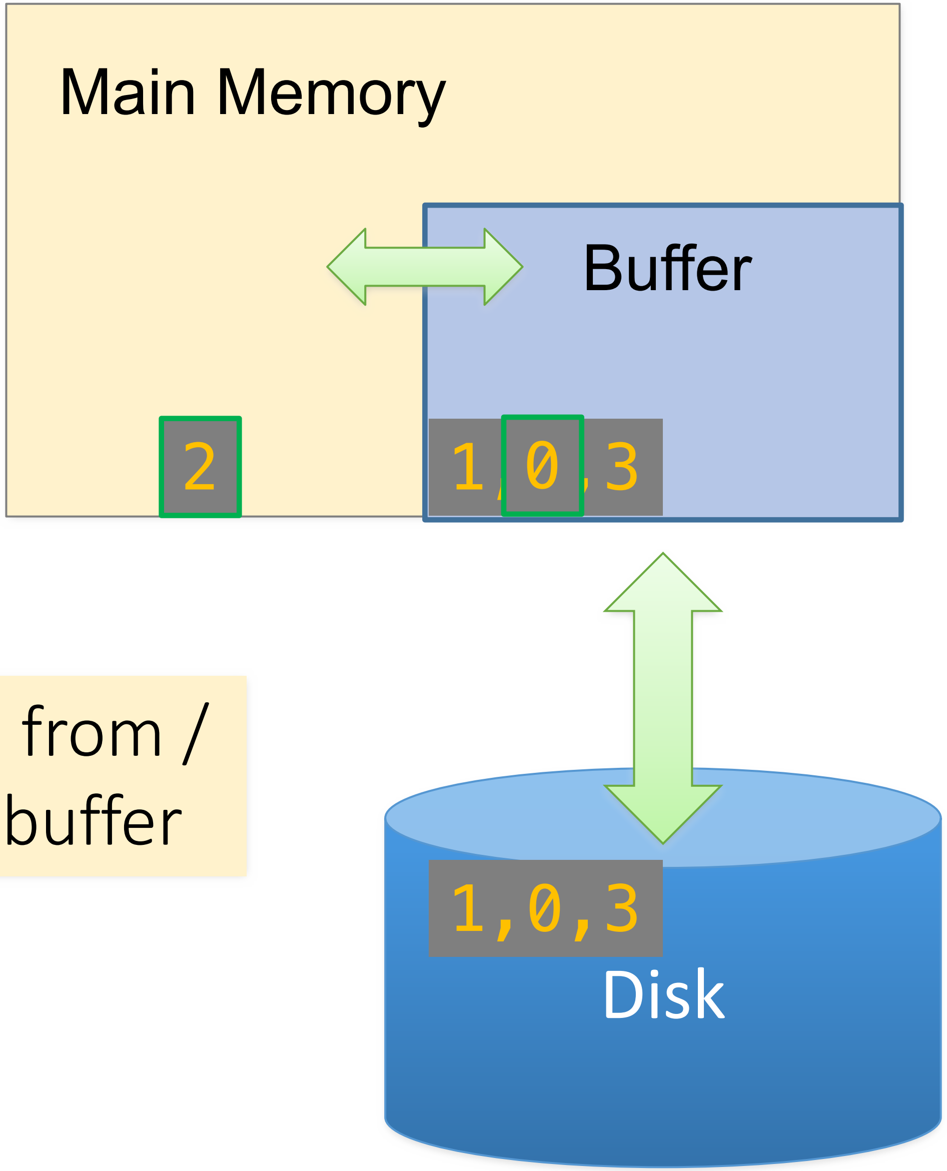
  - **<u>Read(page):</u>** Read page from disk -> buffer *if not already in buffer*



The (Simplified) Buffer

- In this class: We’ll consider a buffer located in main memory that operates over pages and files:

  - **<u>Read(page):</u>** Read page from disk -> buffer *if not already in buffer*



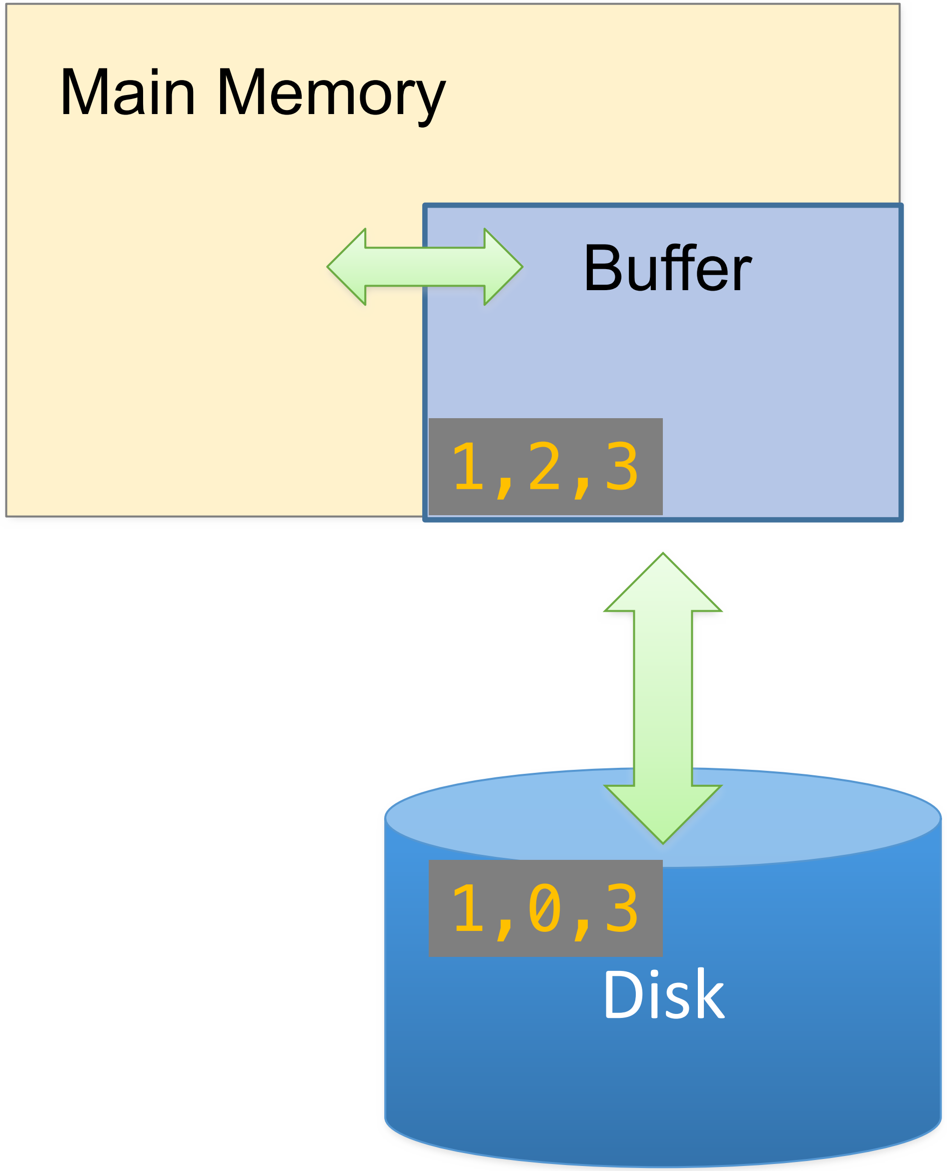
Processes can then read from / write to the page in the buffer

The (Simplified) Buffer

- In this class: We'll consider a buffer located in main memory that operates over pages and files:

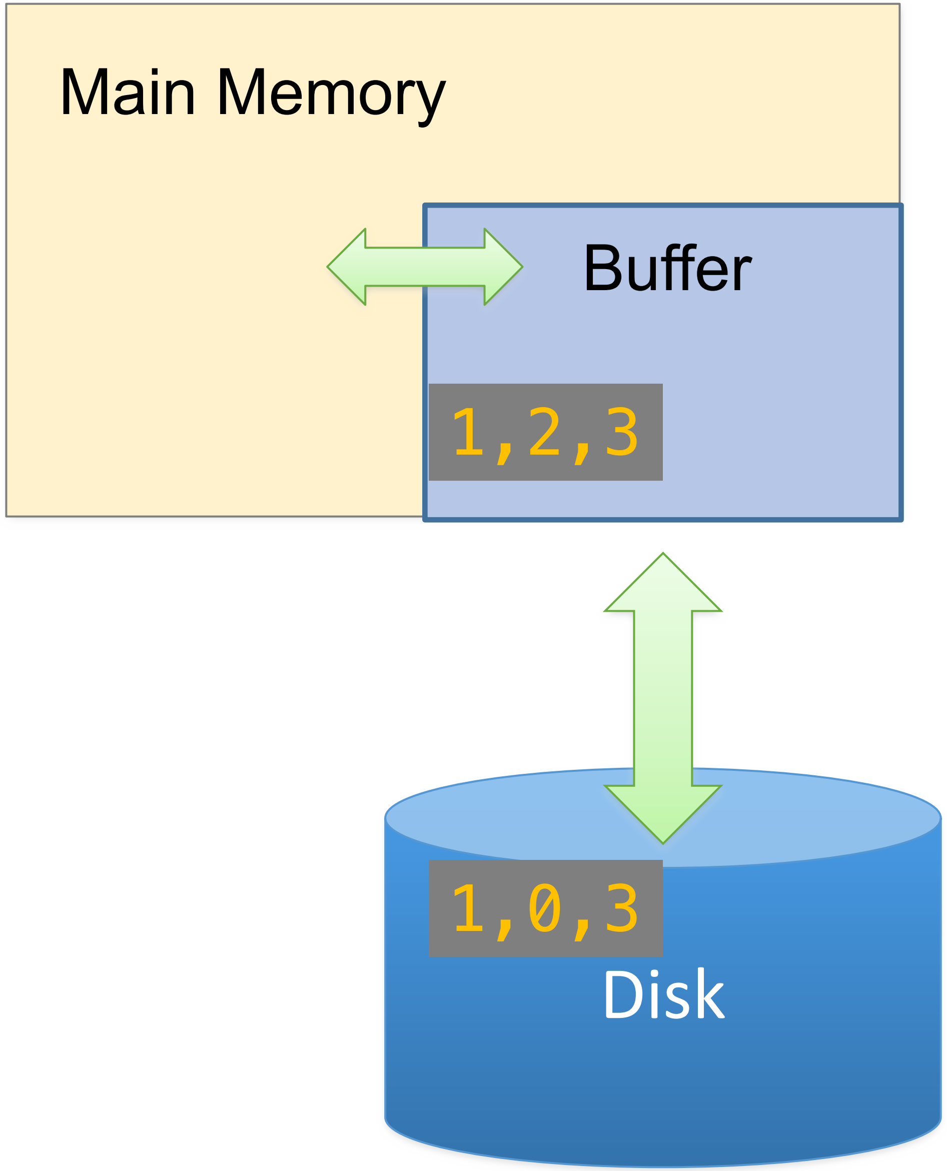
  - **<u>Read(page):</u>** Read page from disk -> buffer *if not already in buffer*

  - **<u>Flush(page):</u>** Evict page from buffer & write to disk



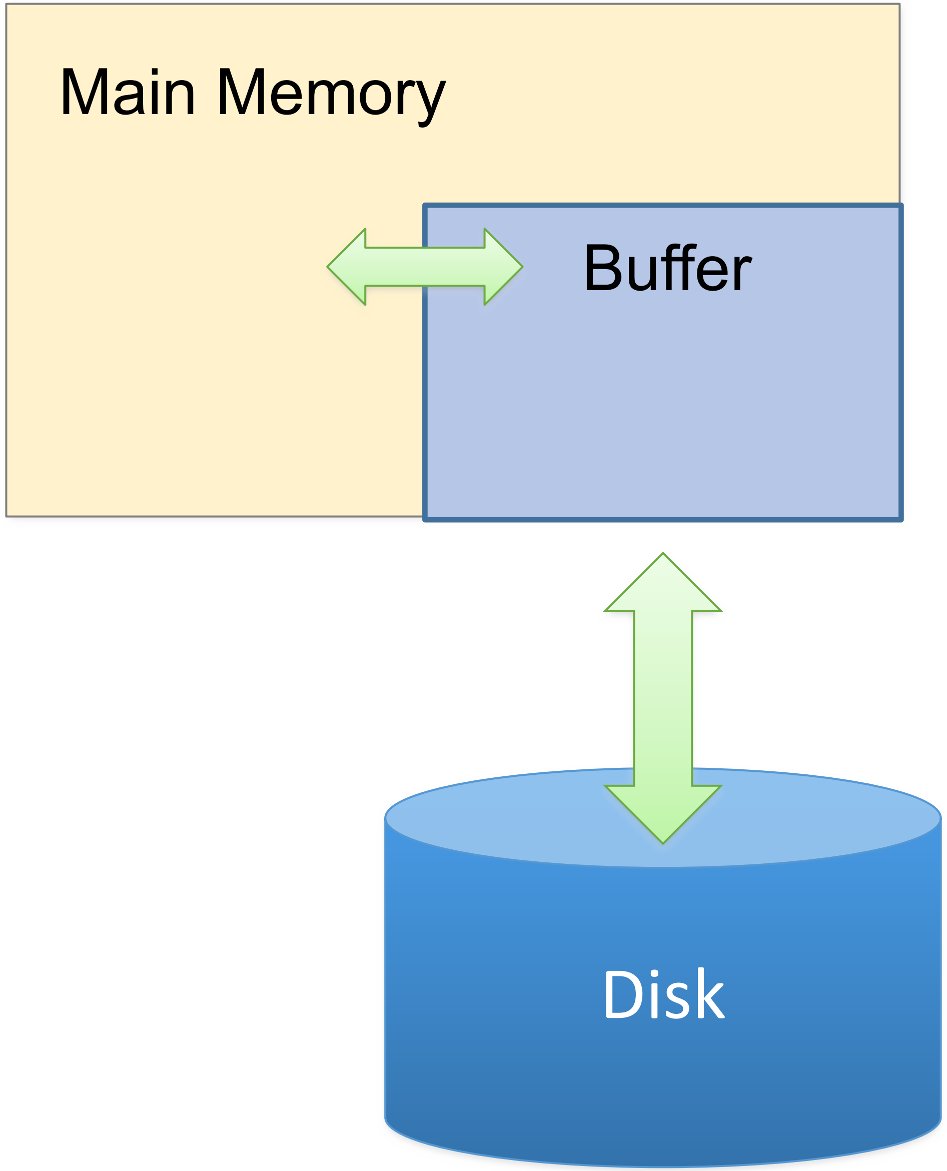
The (Simplified) Buffer

- In this class: We'll consider a buffer located in main memory that operates over pages and files:
  - **<u>Read(page):</u>** Read page from disk -> buffer *if not already in buffer*
  - **<u>Flush(page):</u>** Evict page from buffer & write to disk
  - **<u>Release(page):</u>** Evict page from buffer *without* writing to disk



Managing Disk: The DBMS Buffer

- Database maintains its own buffer
  - Why? The OS already does this…
- Because:
  - DB knows more about access patterns.
    - Watch for how this shows up! Recovery and logging require ability to flush to disk.

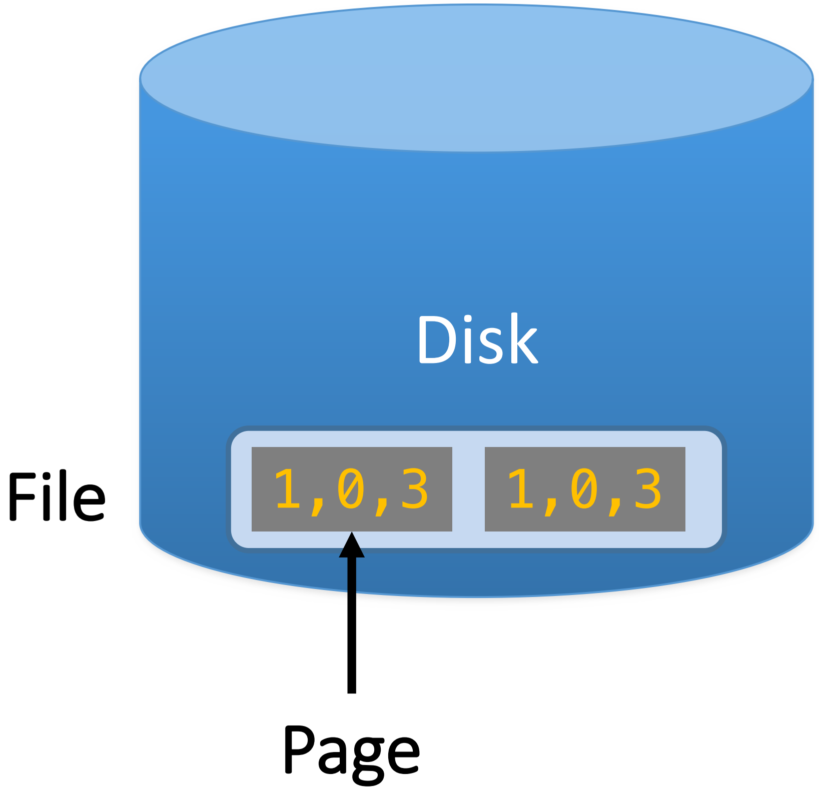


The Buffer Manager

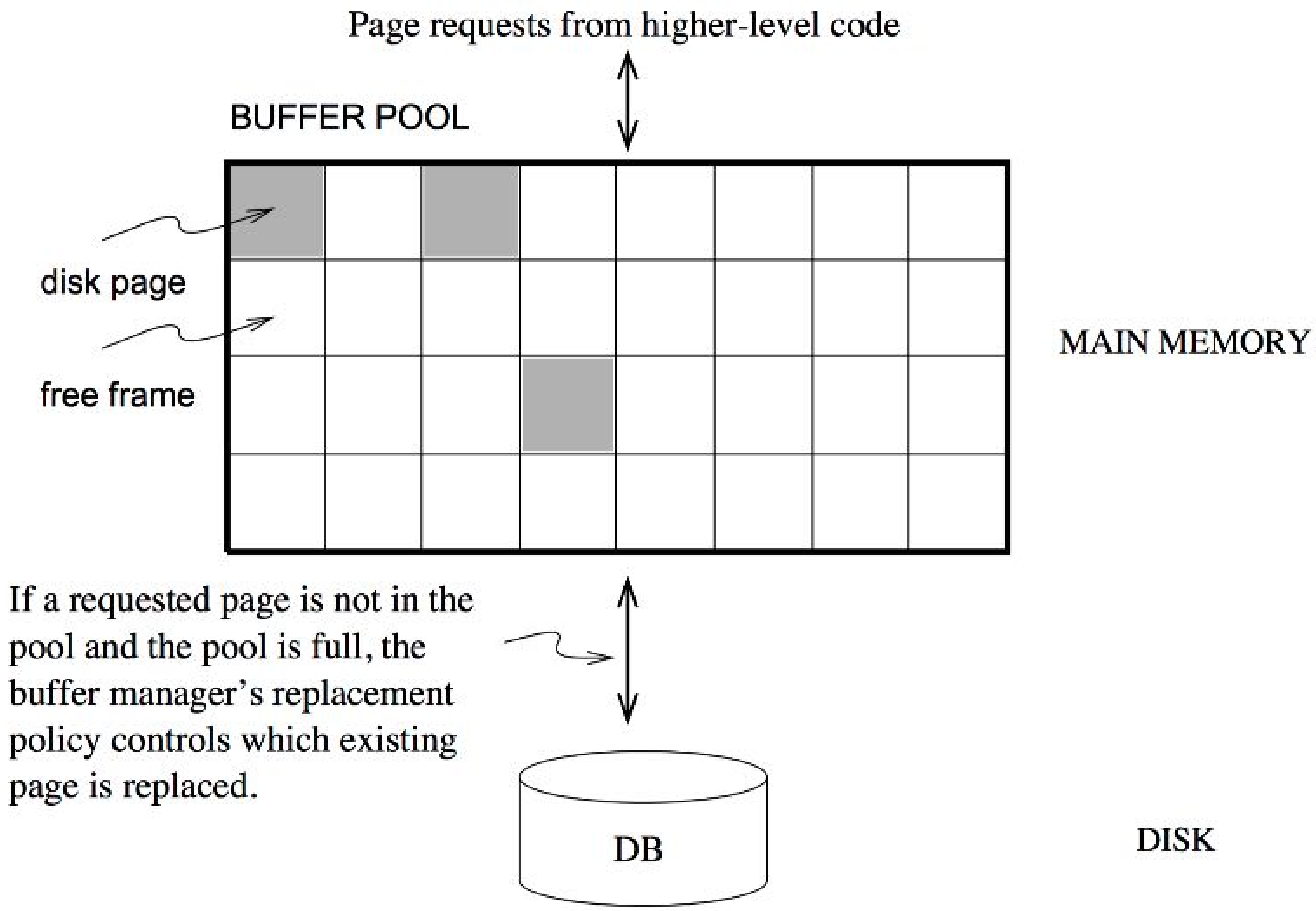
- A buffer manager handles supporting operations for the buffer:
  - Primarily, handles & executes the “replacement policy”
    - i.e. finds a page in buffer to flush/release if buffer is full and a new page needs to be read in
  - DBMSs typically implement their own buffer management routines

A Simplified Filesystem Model

- For us, a page is a fixed-sized array of memory
  - Think: One or more disk blocks
  - Interface:
    - write to an entry (called a slot) or set to “None”
  - DBMS also needs to handle variable length fields
    - Page layout is important for good hardware utilization as well
- And a file is a variable-length list of pages
  - Interface: create / open / close; next_page(); etc.



The Buffer Pool



If a requested page is not in the pool and the pool is full, the buffer manager's replacement policy controls which existing page is replaced.

Source: "Cow book" (Ramakrishan, Gehrke, Database management systems, 2003)

2. External Merge Algorithm

Challenge: Merging Big Files with Small Memory

- How do we efficiently merge two sorted files when both are much larger than our main memory buffer?

- Key point: Disk IO (R/W) dominates the algorithm cost

Our first example of an **“IO aware”** algorithm / cost model

External Merge Algorithm

- Input: 2 sorted lists of length m and n
- Output: 1 sorted list of length m + n
- Required: At least … (?) Buffer Pages
- IOs: … (?)