Lecture 9:
The Buffer & External Merge

Today's Lecture

1. The Buffer
2. External Merge
3. External Merge Sort & Sorting Optimizations

1. The Buffer

Transition to **Mechanisms**

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

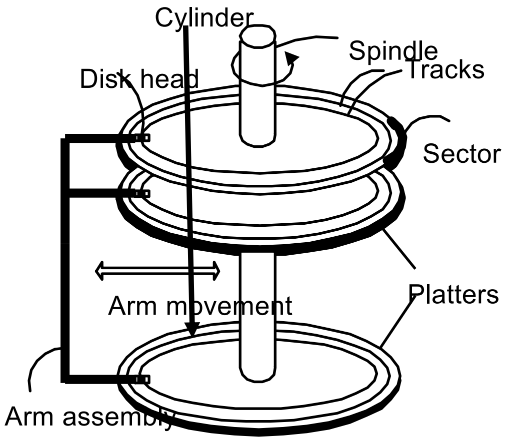
2. Many **mechanisms** have become **stand-alone systems**
   - **Indexing** to Key-value stores
   - Embedded join processing
   - SQL-like languages take some aspect of what we discuss (PIG, Hive)

What you will learn about in this section

1. RECAP: Storage and memory model

2. Buffer primer

High-level: Disk vs. Main Memory



Disk:

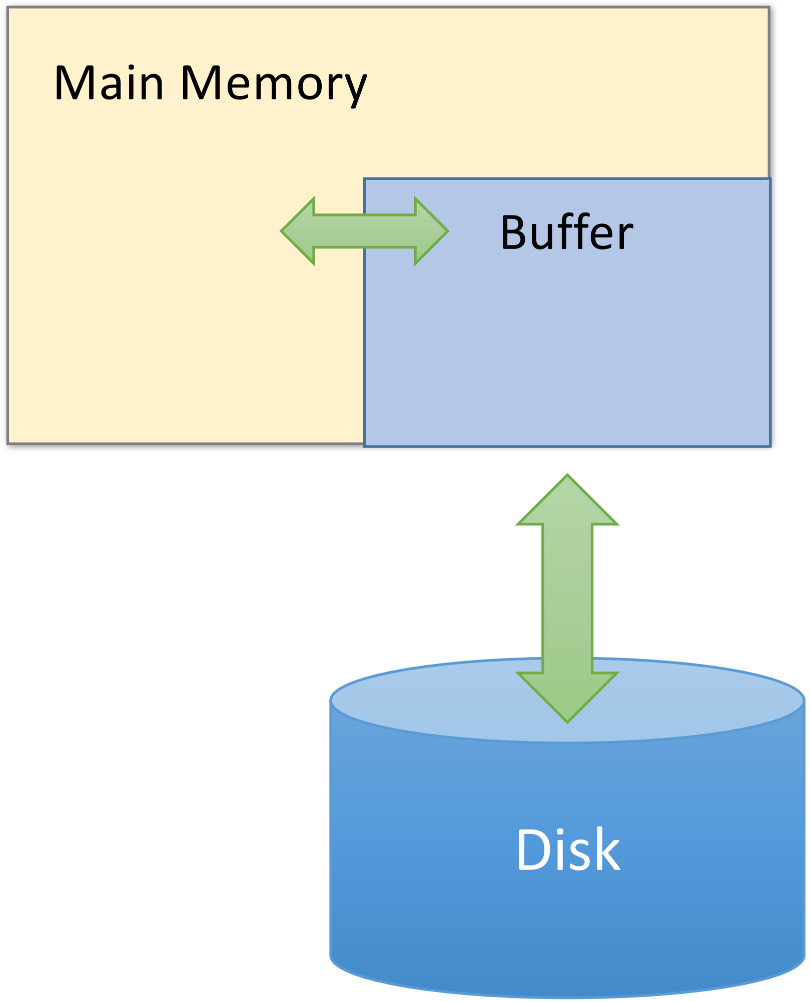
- ***Slow:*** Sequential *block* access
  - Read a blocks (not byte) at a time, so sequential access is cheaper than random
  - **Disk read / writes are expensive!**
- ***Durable:*** We will assume that once on disk, data is safe!
- ***Cheap***

Random Access Memory (RAM) or Main Memory:

- ***Fast:*** Random access, byte addressable
  - ~10x faster for sequential access
  - ~100,000x faster for random access!
- ***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

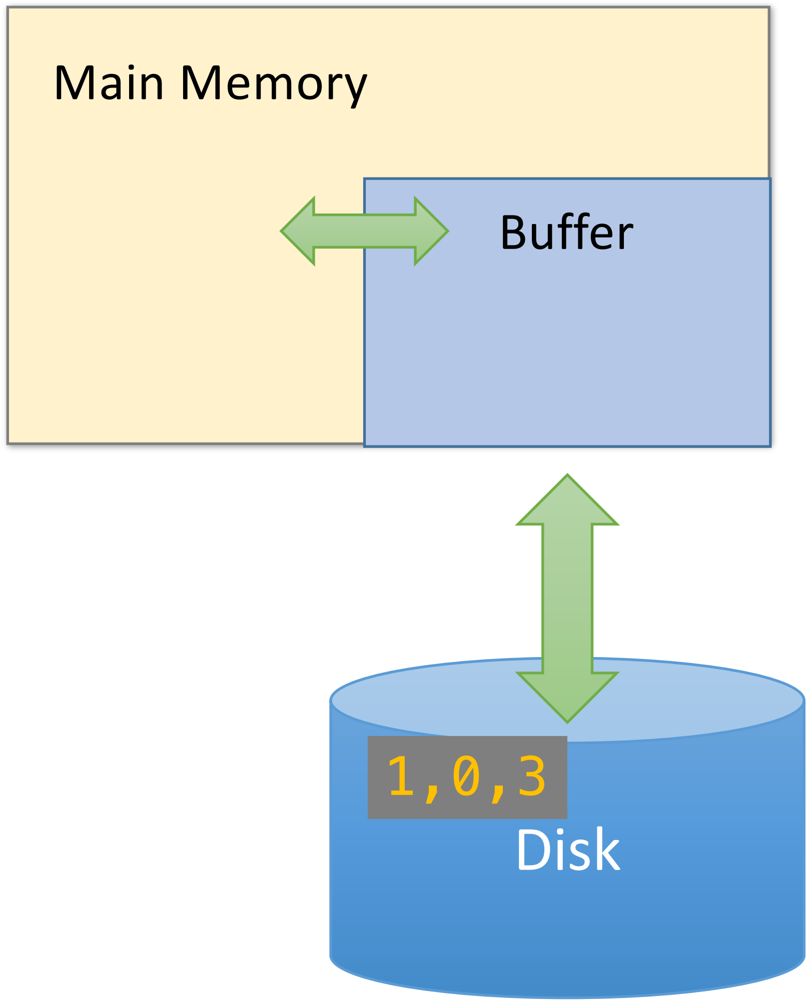
- A **buffer** is a region of physical memory used to store *temporary data*
  - *In this lecture:* a region in main memory used to store **intermediate data between disk and processes**
- *Key idea:* Reading / writing to disk is slow- need to cache data!



The (Simplified) Buffer

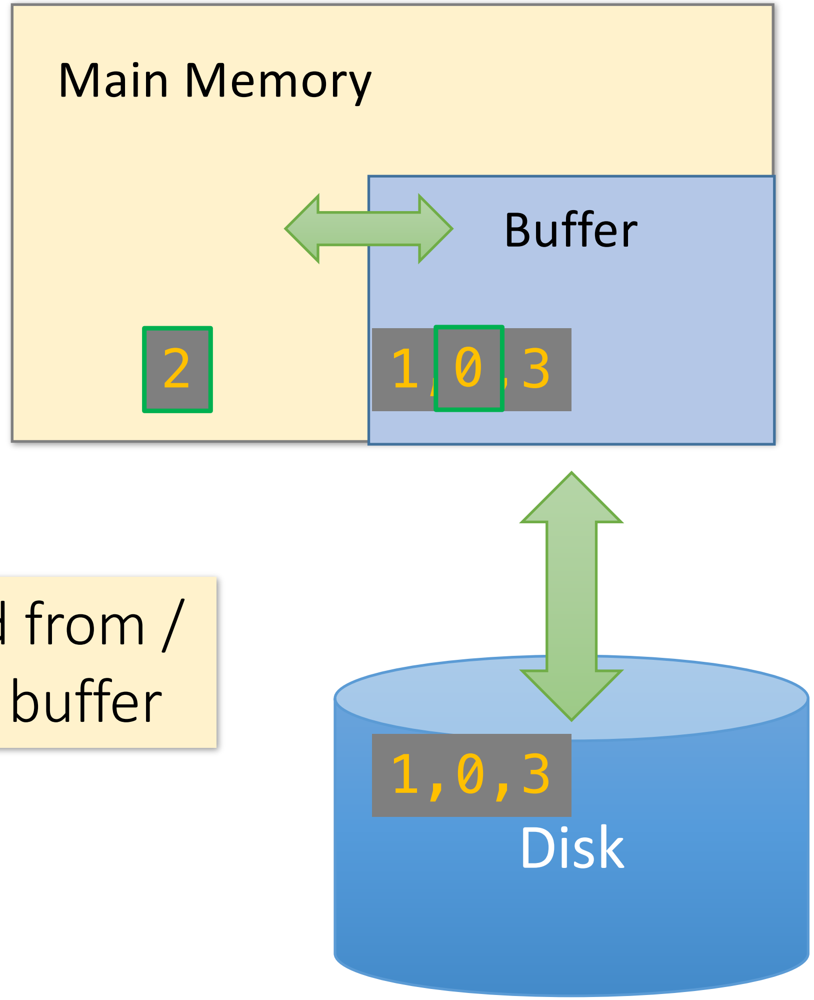
- In this class: We'll consider a buffer located in **main memory** that operates over **pages** and **files**:

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



The (Simplified) Buffer

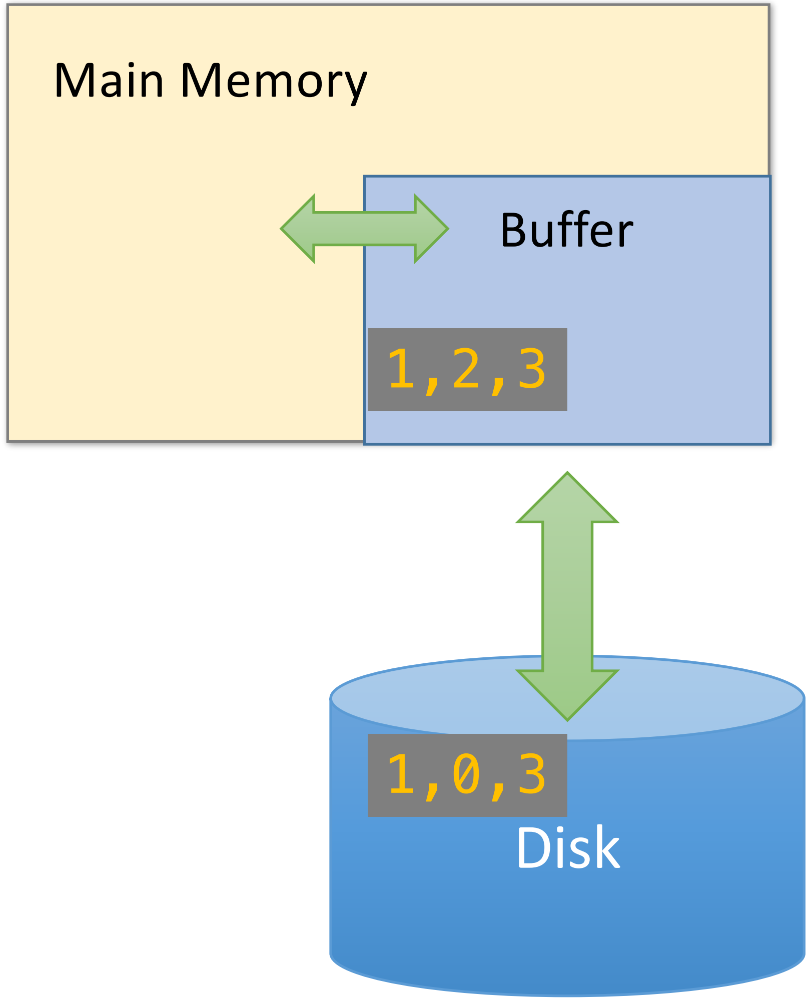
- In this class: We'll consider a buffer located in **main memory** that operates over **pages** and **files**:
  - <u>**Read(page):**</u> Read page from disk -> buffer *if not already in buffer*



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

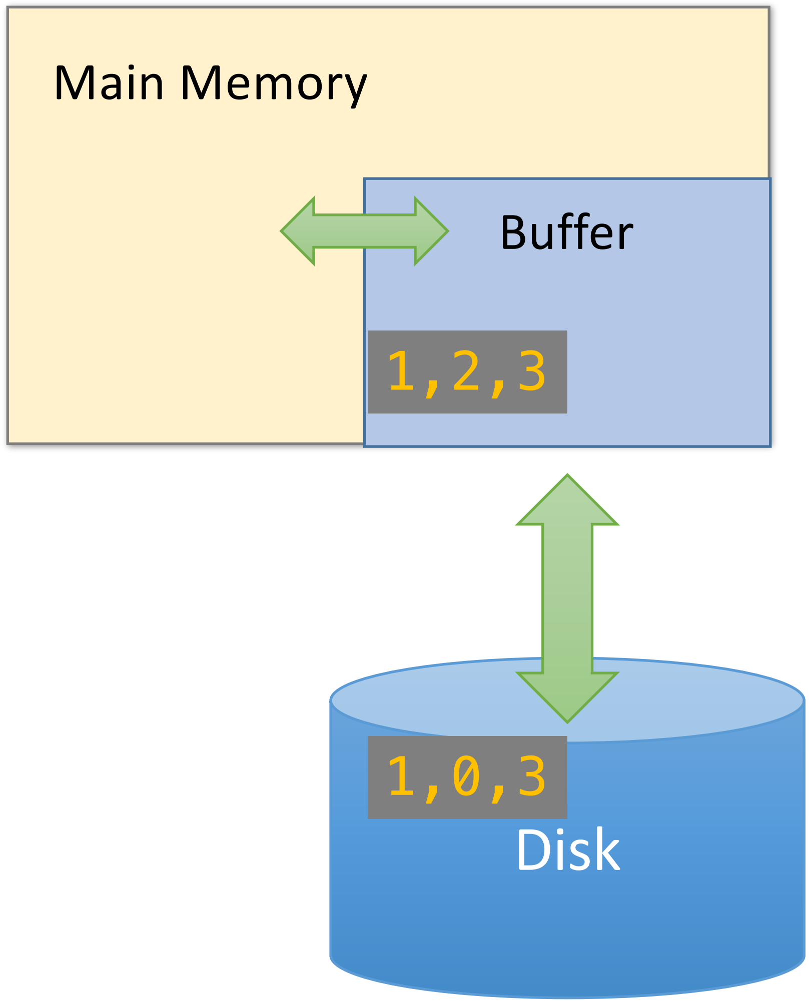
The (Simplified) Buffer

- 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



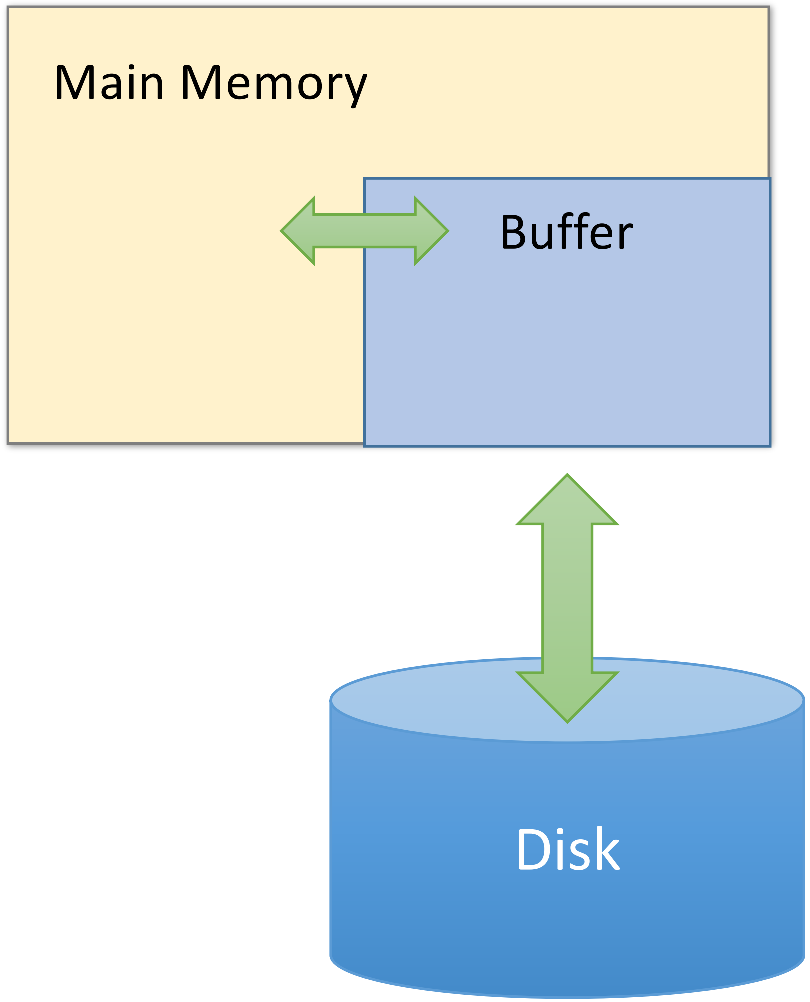
The (Simplified) Buffer

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



Managing Disk: The DBMS Buffer

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

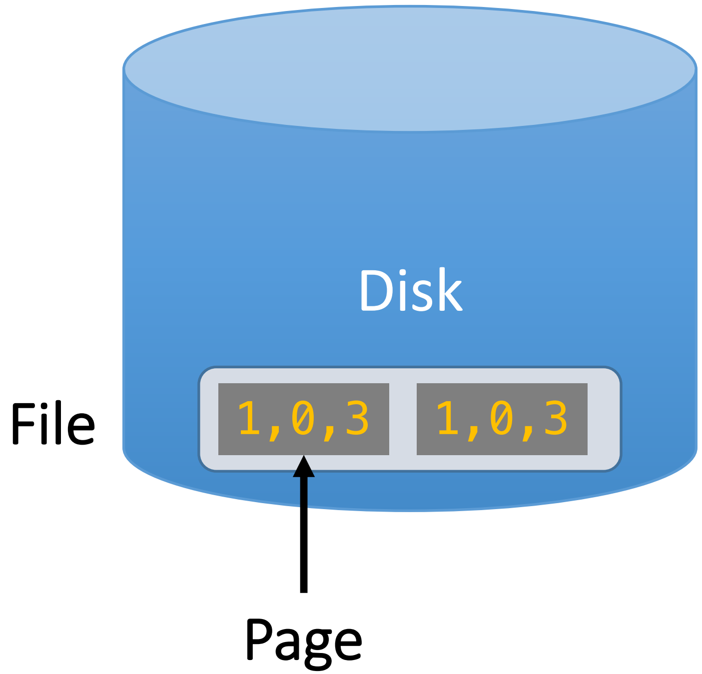


The Buffer Manager

- 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 (see 346)
- And a **file** is a *variable-length list* of pages
  - Interface: create / open / close; next_page(); etc.



DB-WS09a.ipynb

2. External Merge

What you will learn about in this section

1. External Merge- Basics

2. External Merge Algorithm

3. ACTIVITY: External Merge Sort- Demo

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?

External Merge Algorithm

- **Input**: 2 sorted lists of length M and N
- **Output:** 1 sorted list of length M + N
- **Required:** At least 3 Buffer Pages
- **IOs**: 2(M+N)

Key (Simple) Idea

To find an element that is no larger than all elements in two lists, one only needs to compare minimum elements from each list.

If:

$$A_1 \leq A_2 \leq \cdots \leq A_N$$
$$B_1 \leq B_2 \leq \cdots \leq B_M$$

Then:

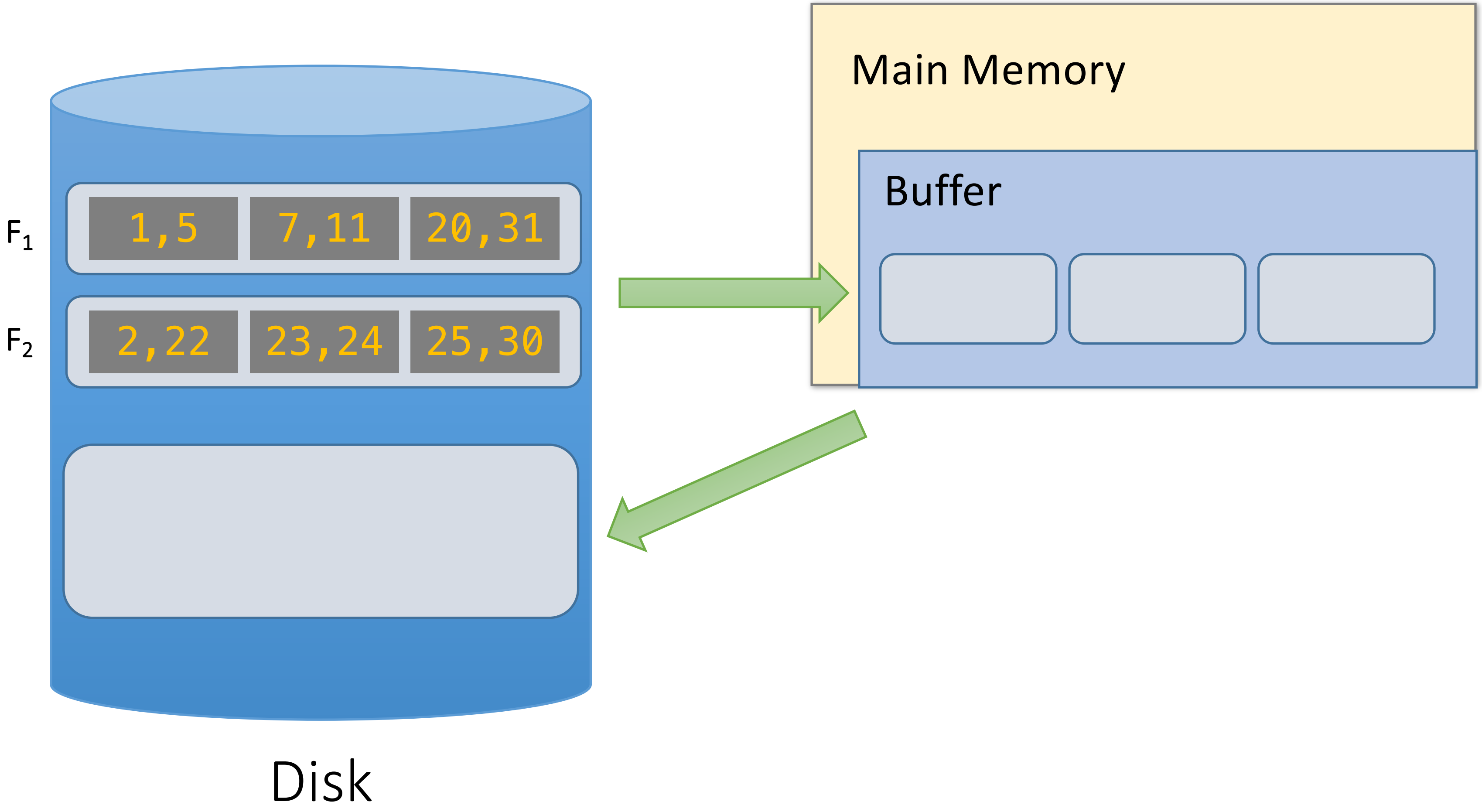
$$Min(A_1, B_1) \leq A_i$$
$$Min(A_1, B_1) \leq B_j$$

for i=1….N and j=1….M

External Merge Algorithm

Input:
Two sorted
files

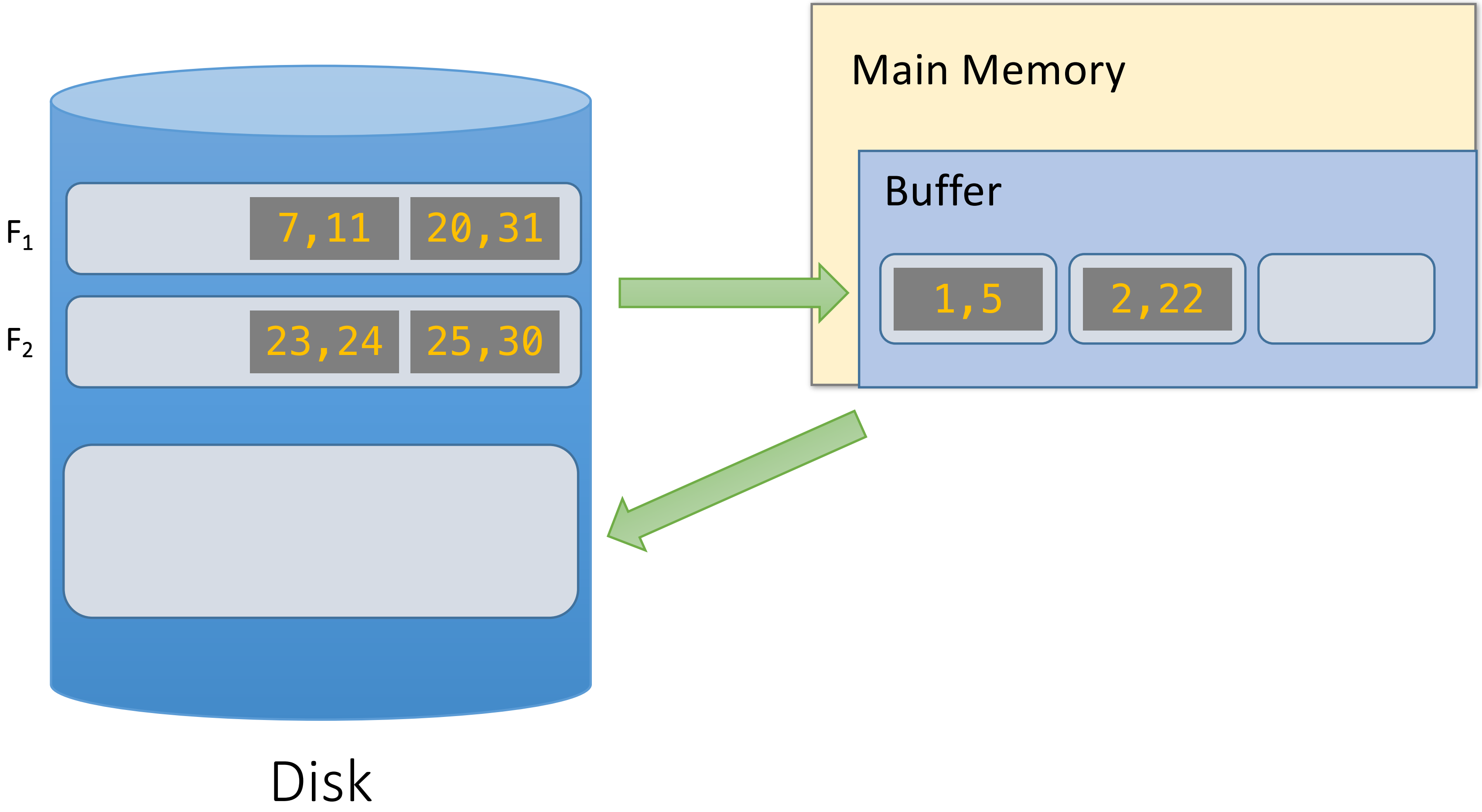
Output:
One *merged*
sorted file



External Merge Algorithm

Input:
Two sorted files

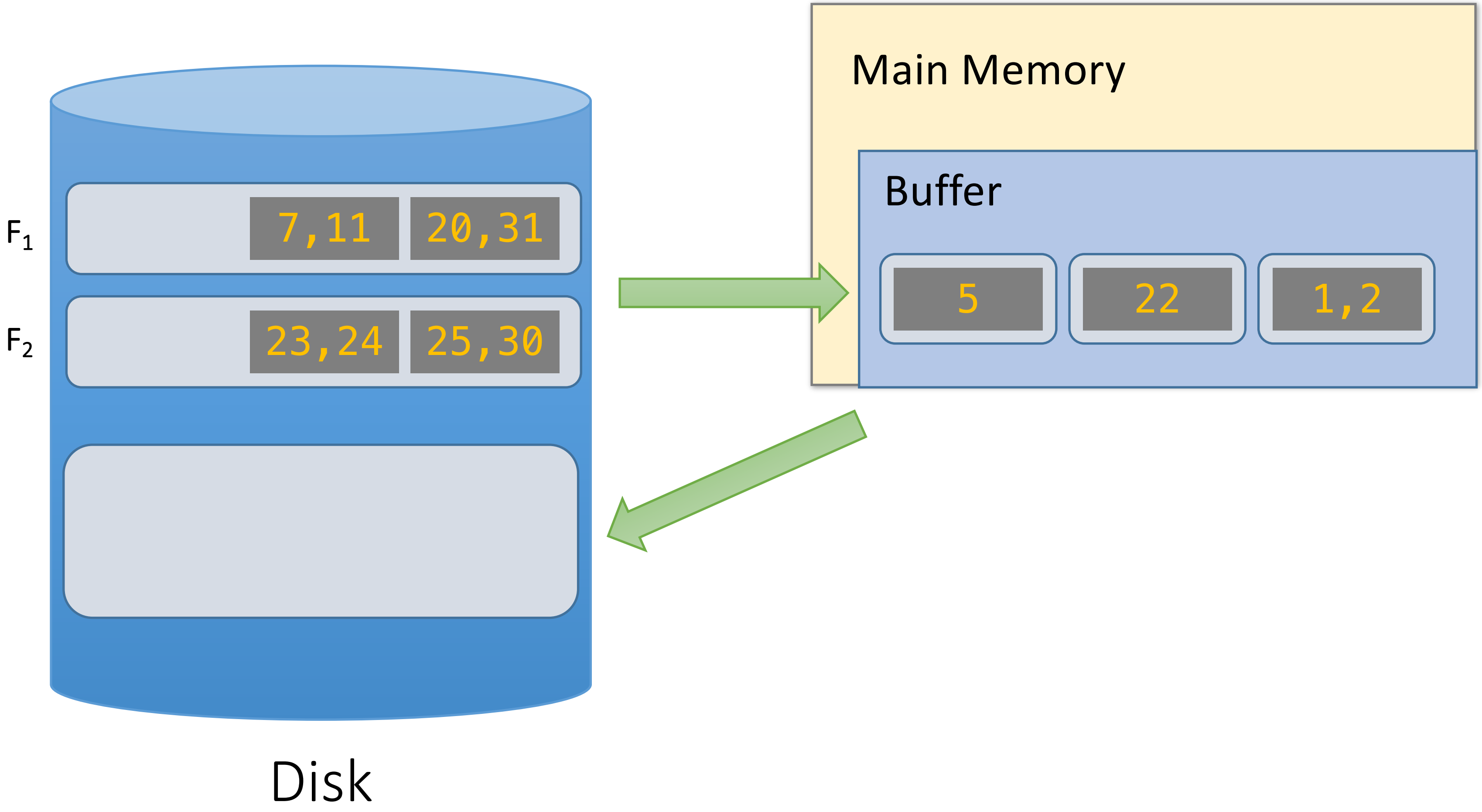
Output:
One *merged* sorted file



External Merge Algorithm

Input:
Two sorted
files

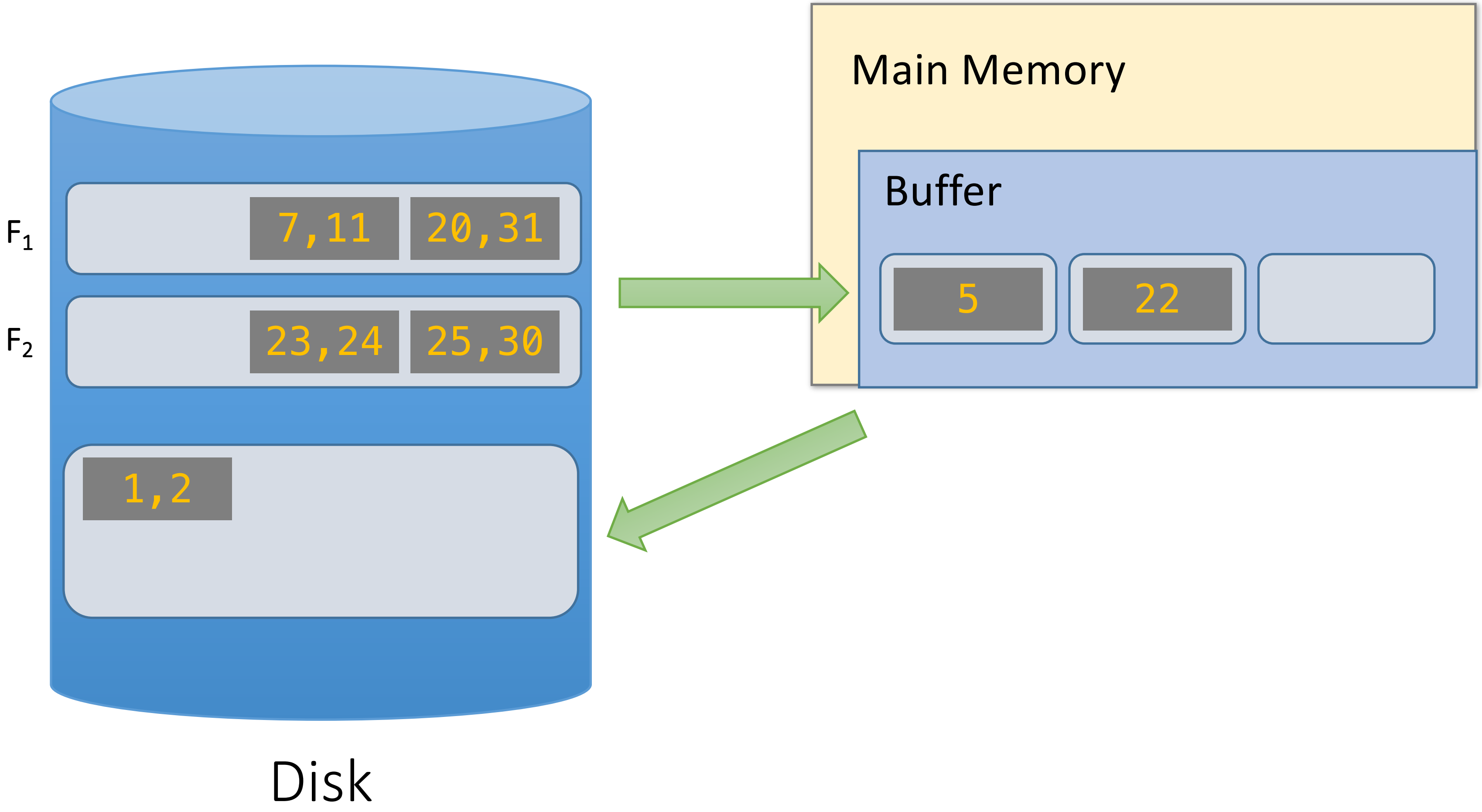
Output:
One *merged*
sorted file



External Merge Algorithm

Input:
Two sorted
files

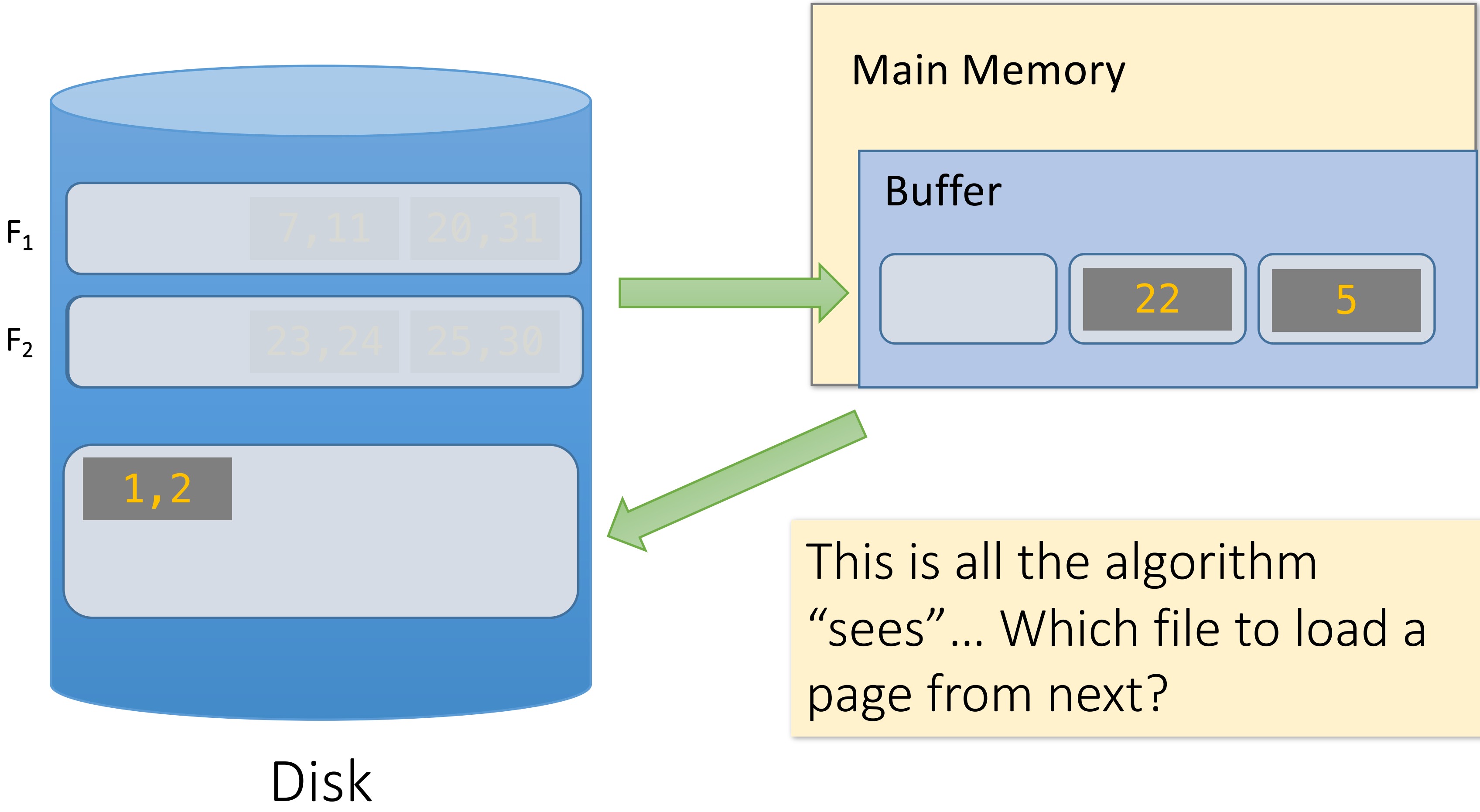
Output:
One *merged*
sorted file



External Merge Algorithm

Input:
Two sorted
files

Output:
One *merged*
sorted file

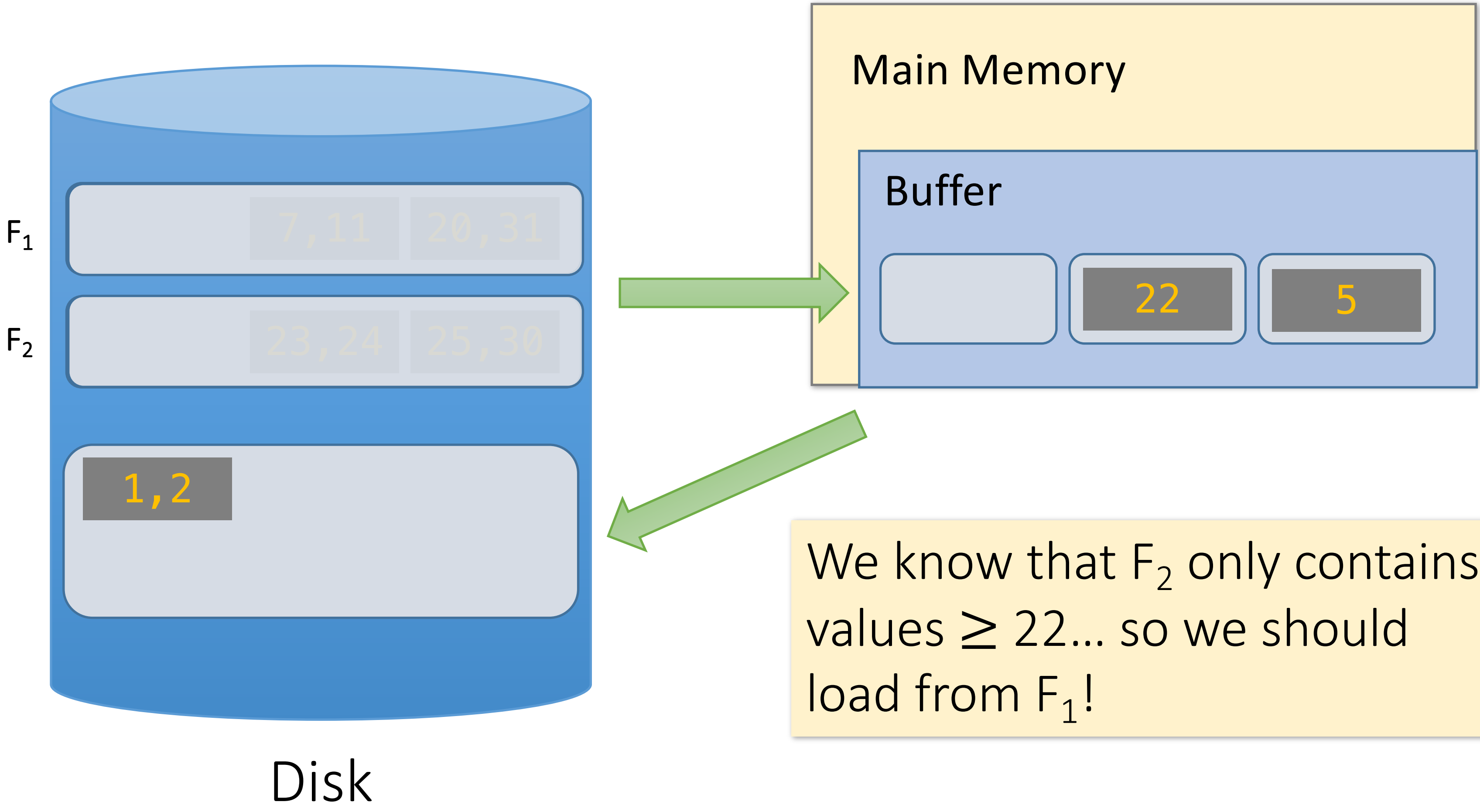


This is all the algorithm "sees"… Which file to load a page from next?

External Merge Algorithm

Input:
Two sorted files

Output:
One *merged* sorted file

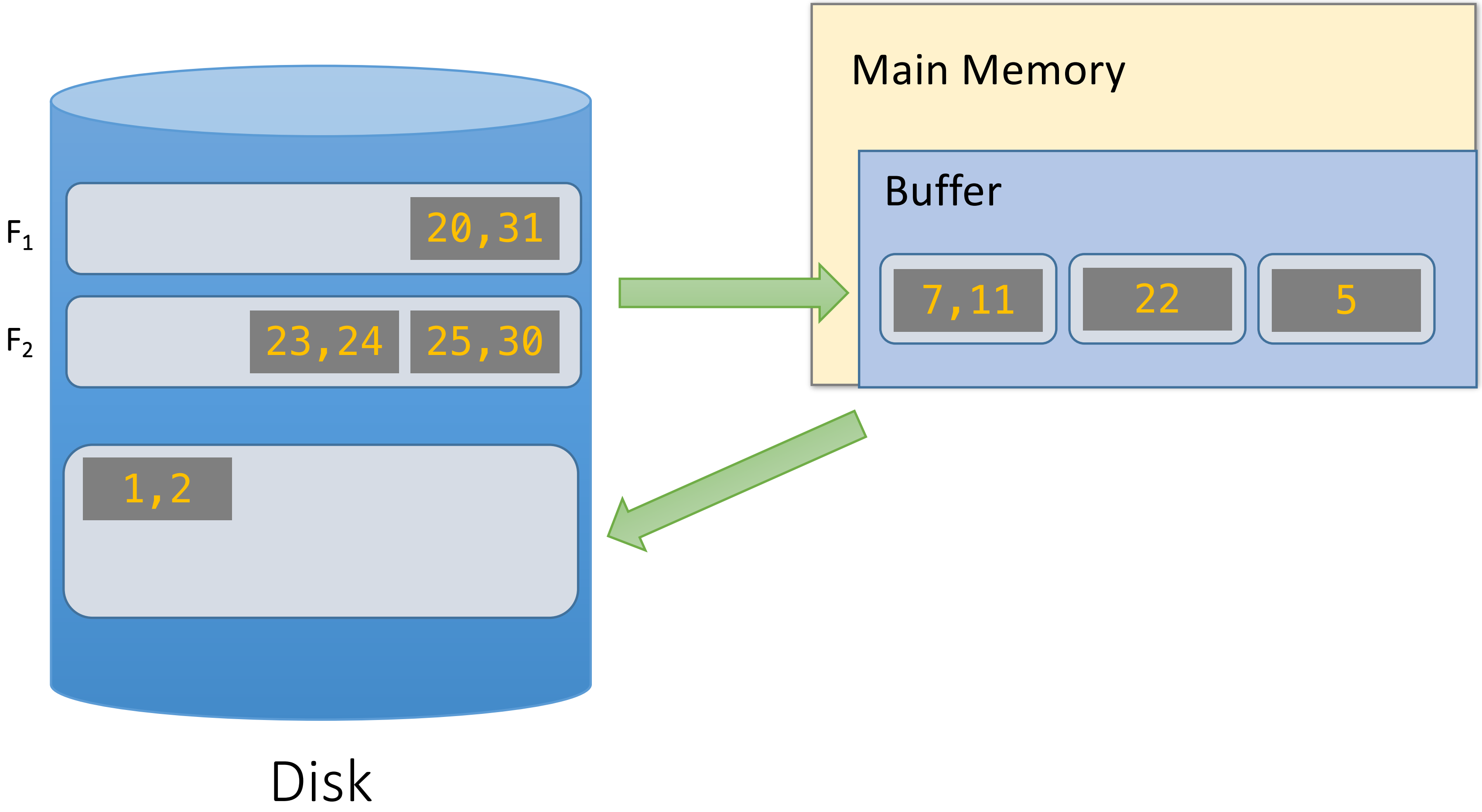


We know that F_2 only contains values ≥ 22… so we should load from F_1!

External Merge Algorithm

Input:
Two sorted
files

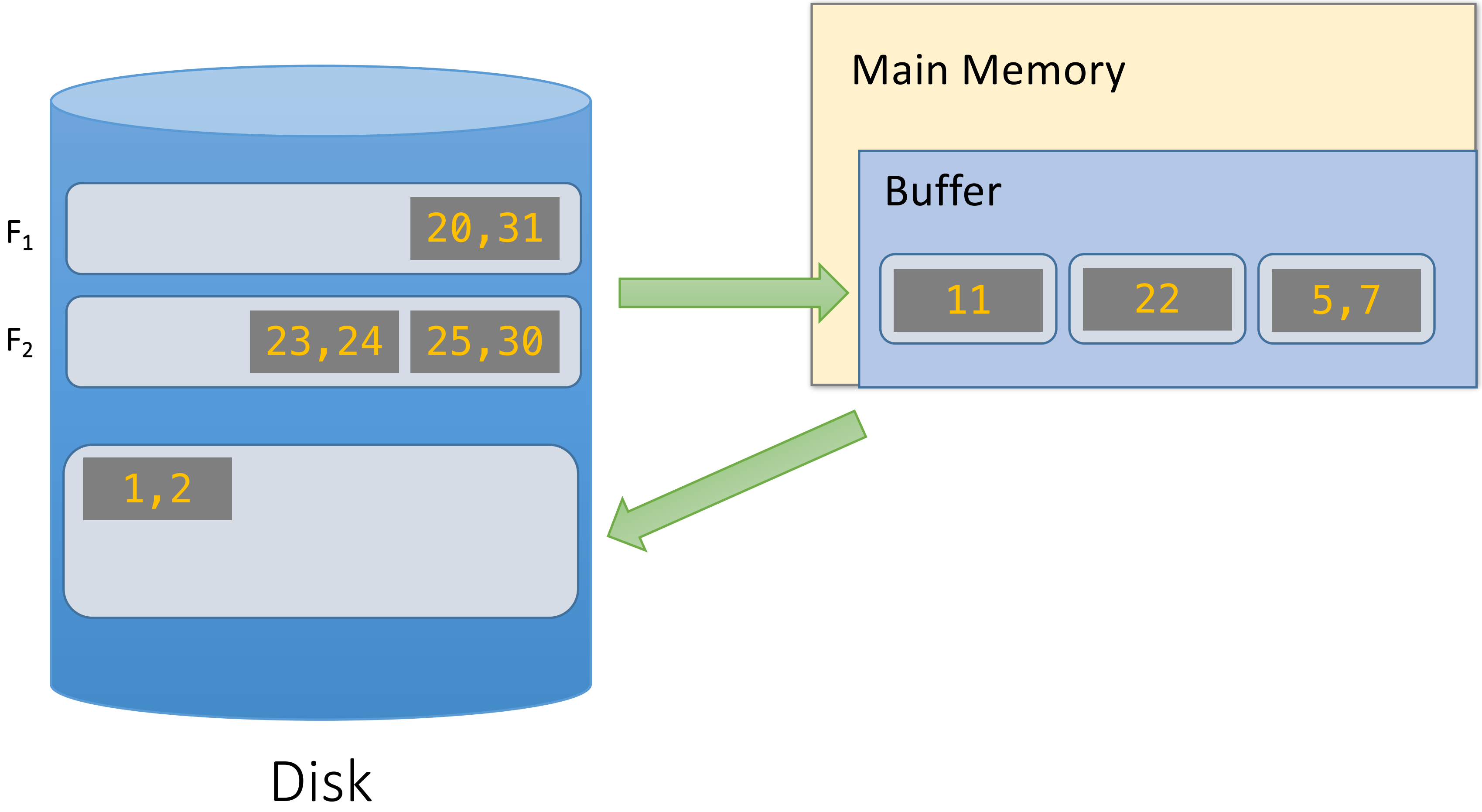
Output:
One *merged*
sorted file



External Merge Algorithm

Input:
Two sorted files

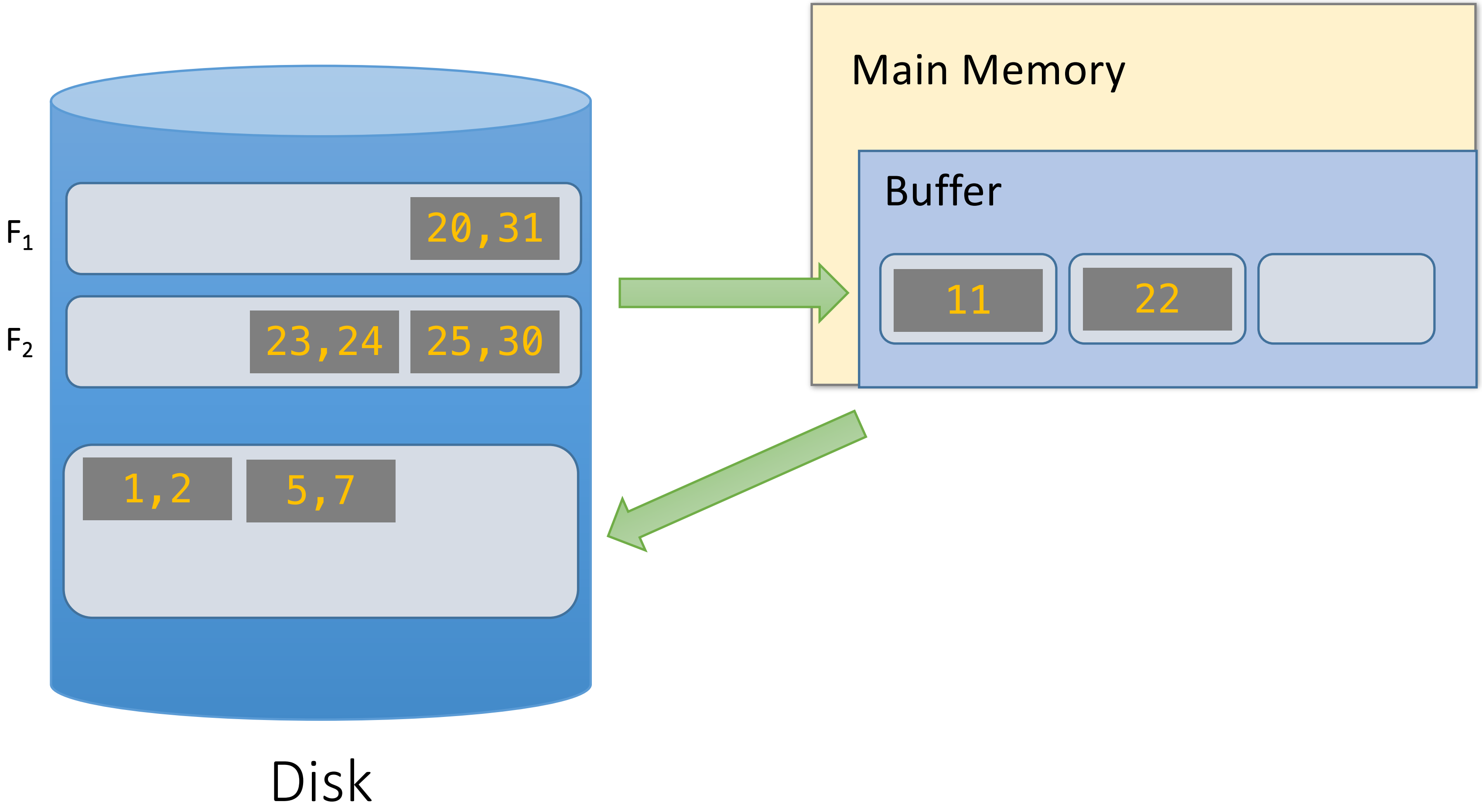
Output:
One *merged* sorted file



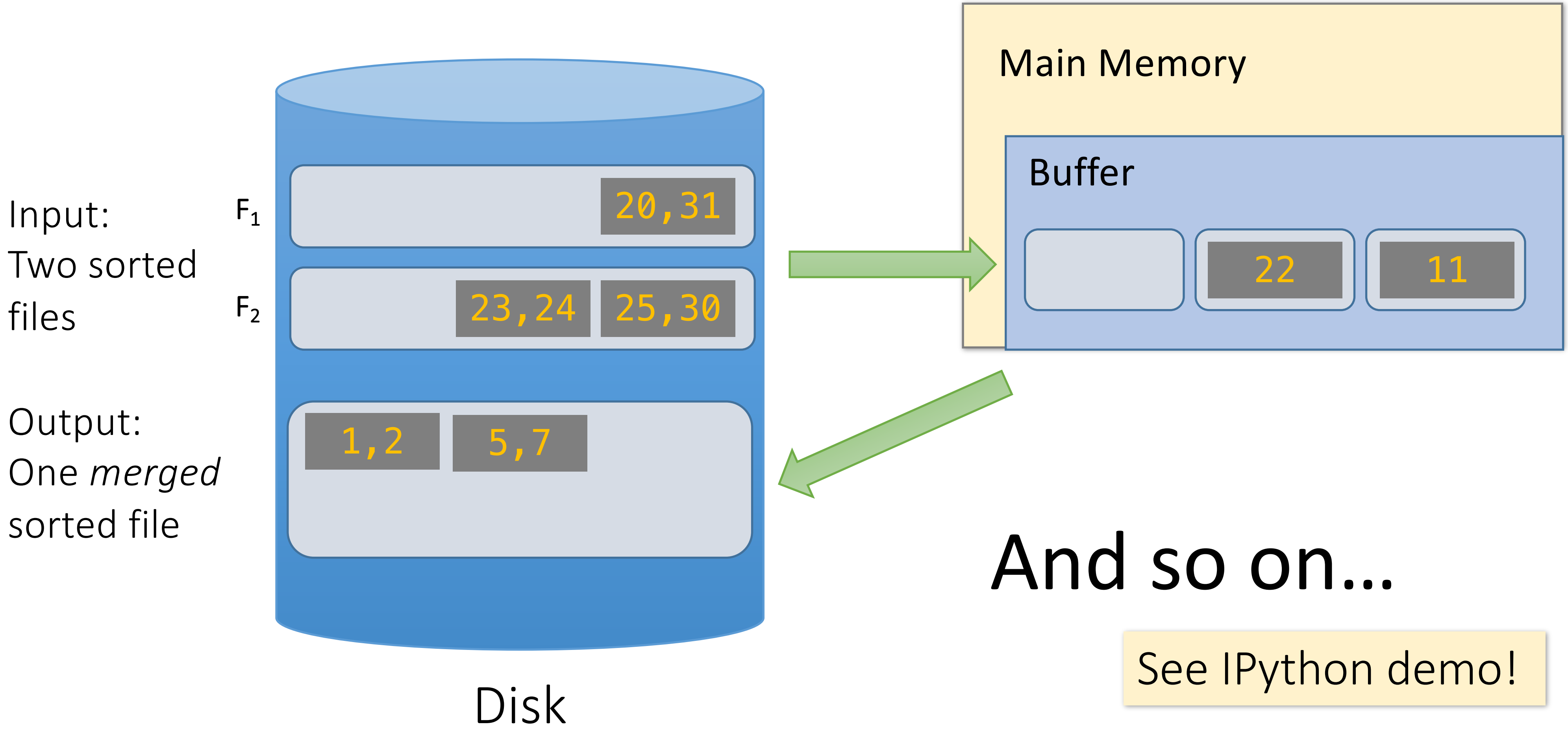
External Merge Algorithm

Input:
Two sorted
files

Output:
One *merged*
sorted file



External Merge Algorithm



And so on…

See IPython demo!

We can merge lists of **arbitrary length** with *only* 3 buffer pages.

If lists of size M and N, then
Cost: 2(M+N) IOs
Each page is read once, written once

With B+1 buffer pages, can merge B lists. How?

EMS-Demo.ipynb