Lecture 11: External Sorting

Announcements

1. M
2. Pr
3. M
    1.
    2.
4. Ou

Lecture 11: External Sorting

What you will learn about in this section

1. External Merge (of sorted files)

2. External Merge - Sort

1. External Merge

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:

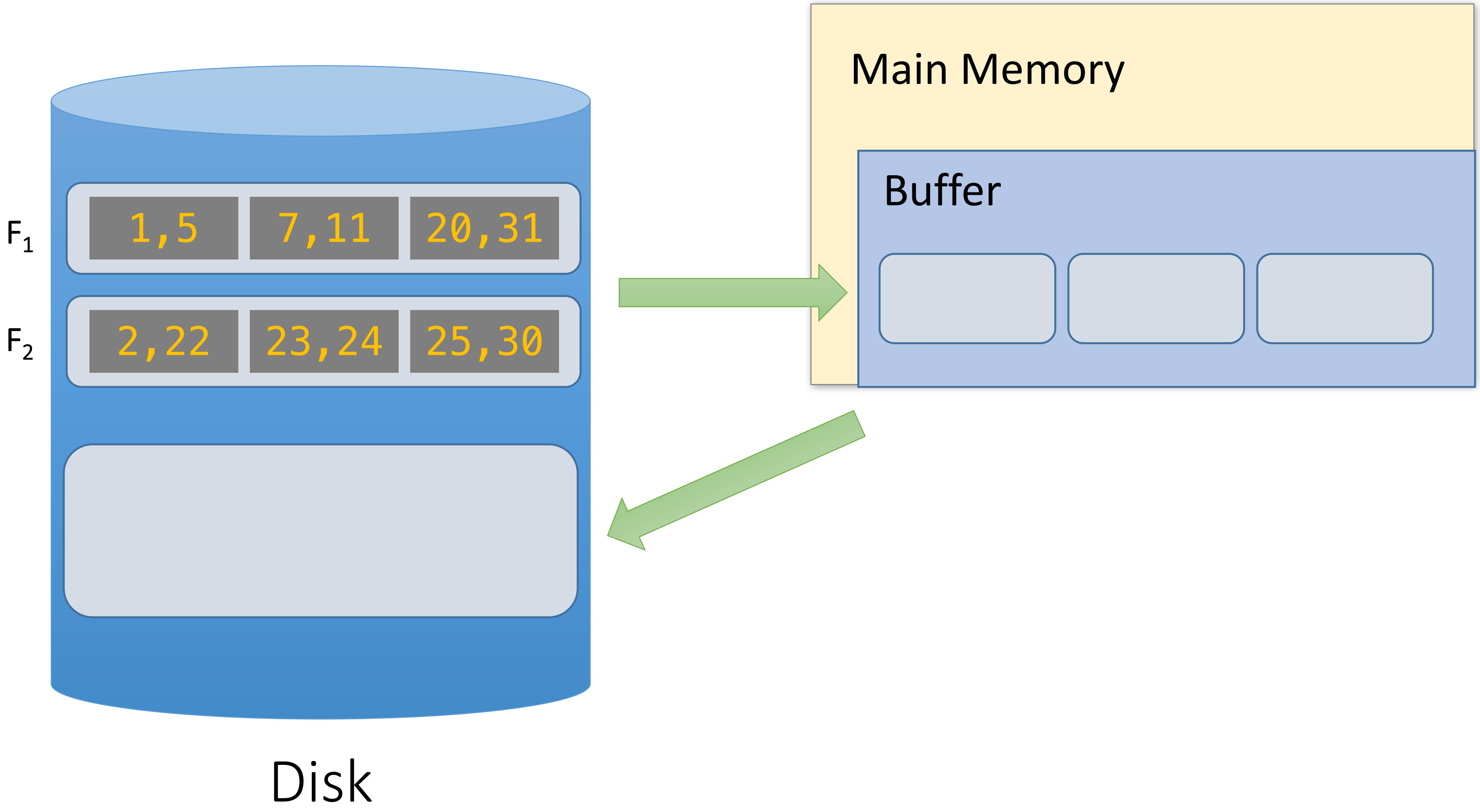
$$Min(A_1, B_1) \leq A_i$$
$$Min(A_1, B_1) \leq B_j$$

for i=1….N and j=1….M

External Merge Algorithm

Input:
Two sorted
files

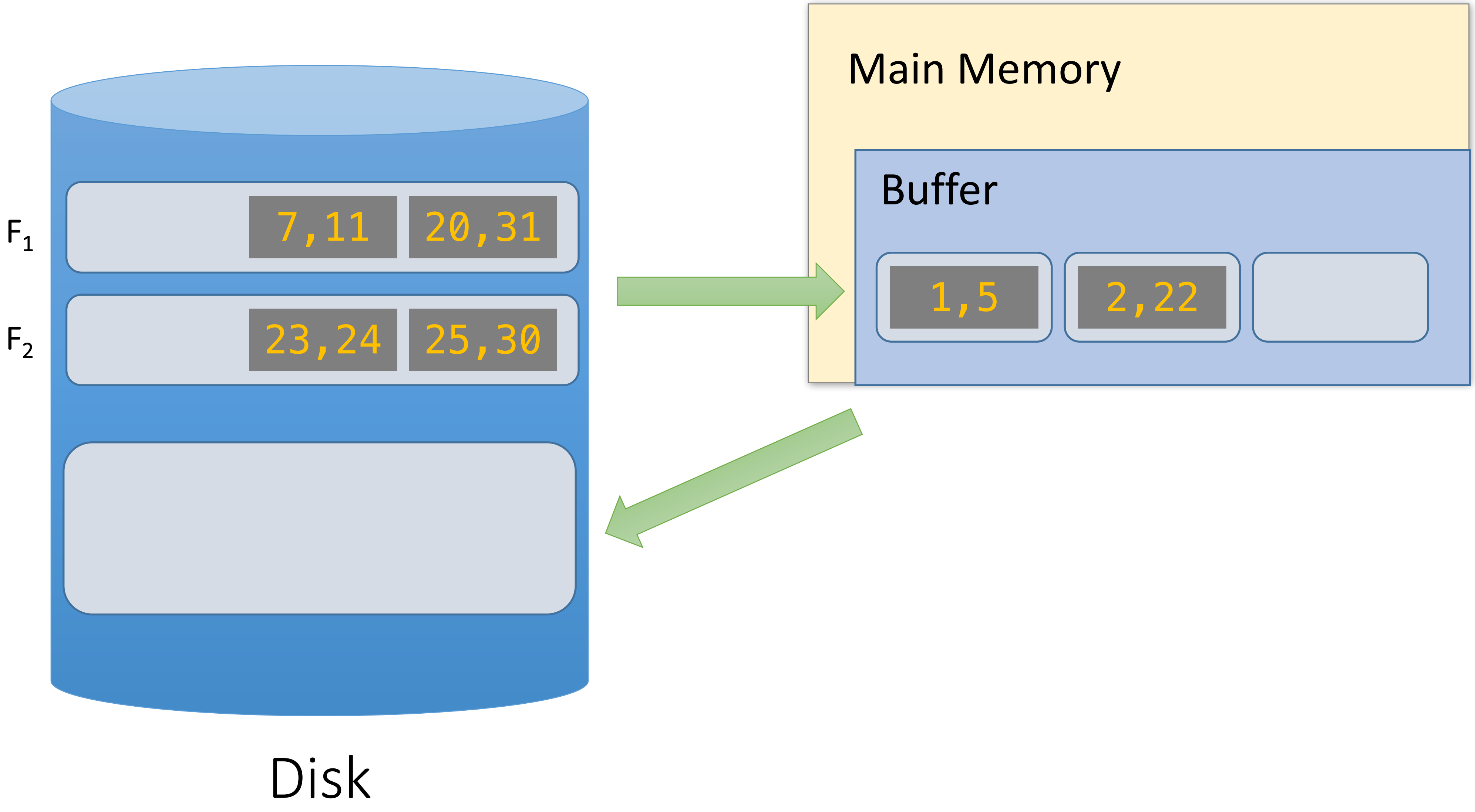
Output:
One *merged*
sorted file



External Merge Algorithm

Input:
Two sorted files

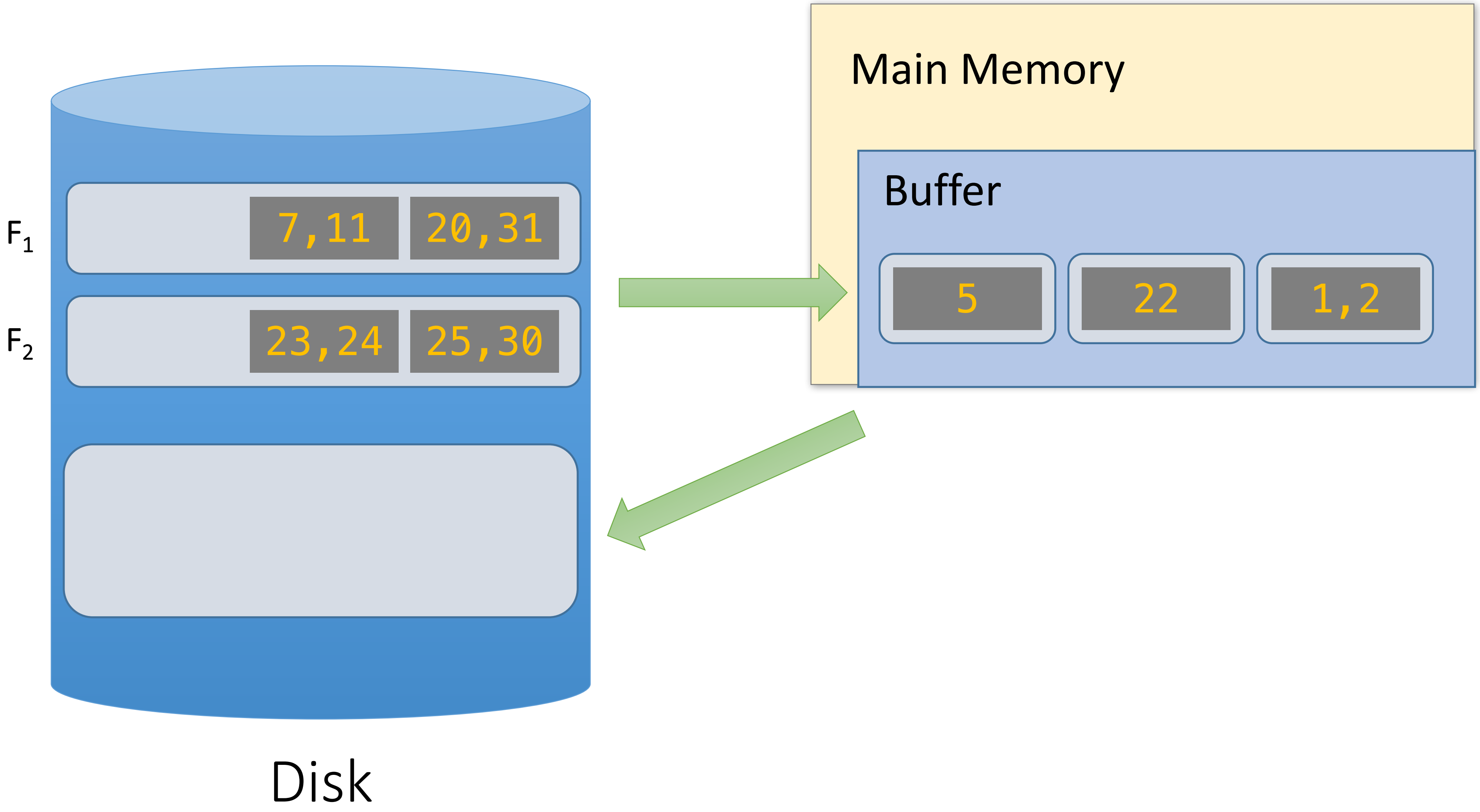
Output:
One *merged* sorted file



External Merge Algorithm

Input:
Two sorted files

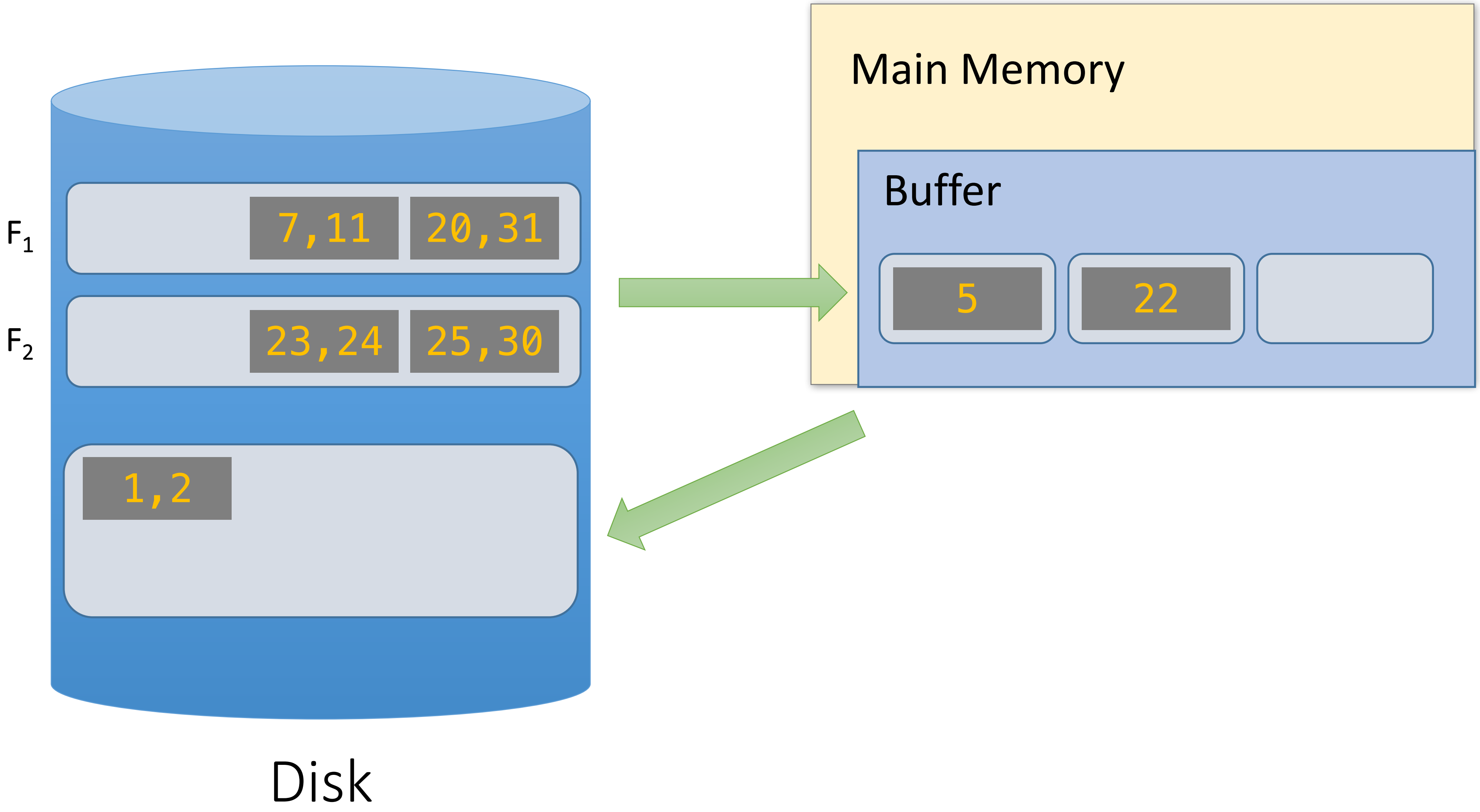
Output:
One *merged* sorted file



External Merge Algorithm

Input:
Two sorted files

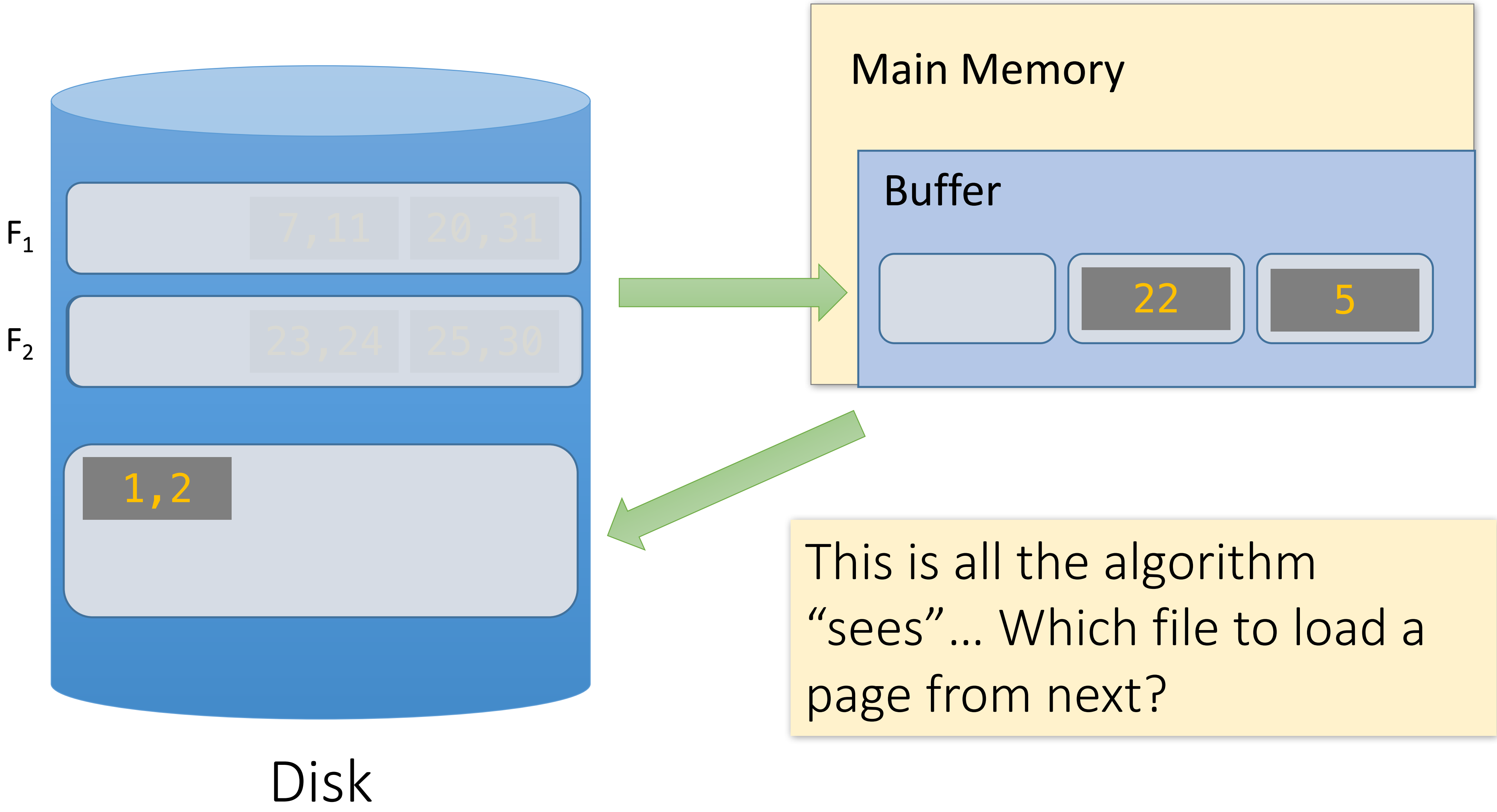
Output:
One *merged* sorted file



External Merge Algorithm

Input:
Two sorted files

Output:
One *merged* sorted file

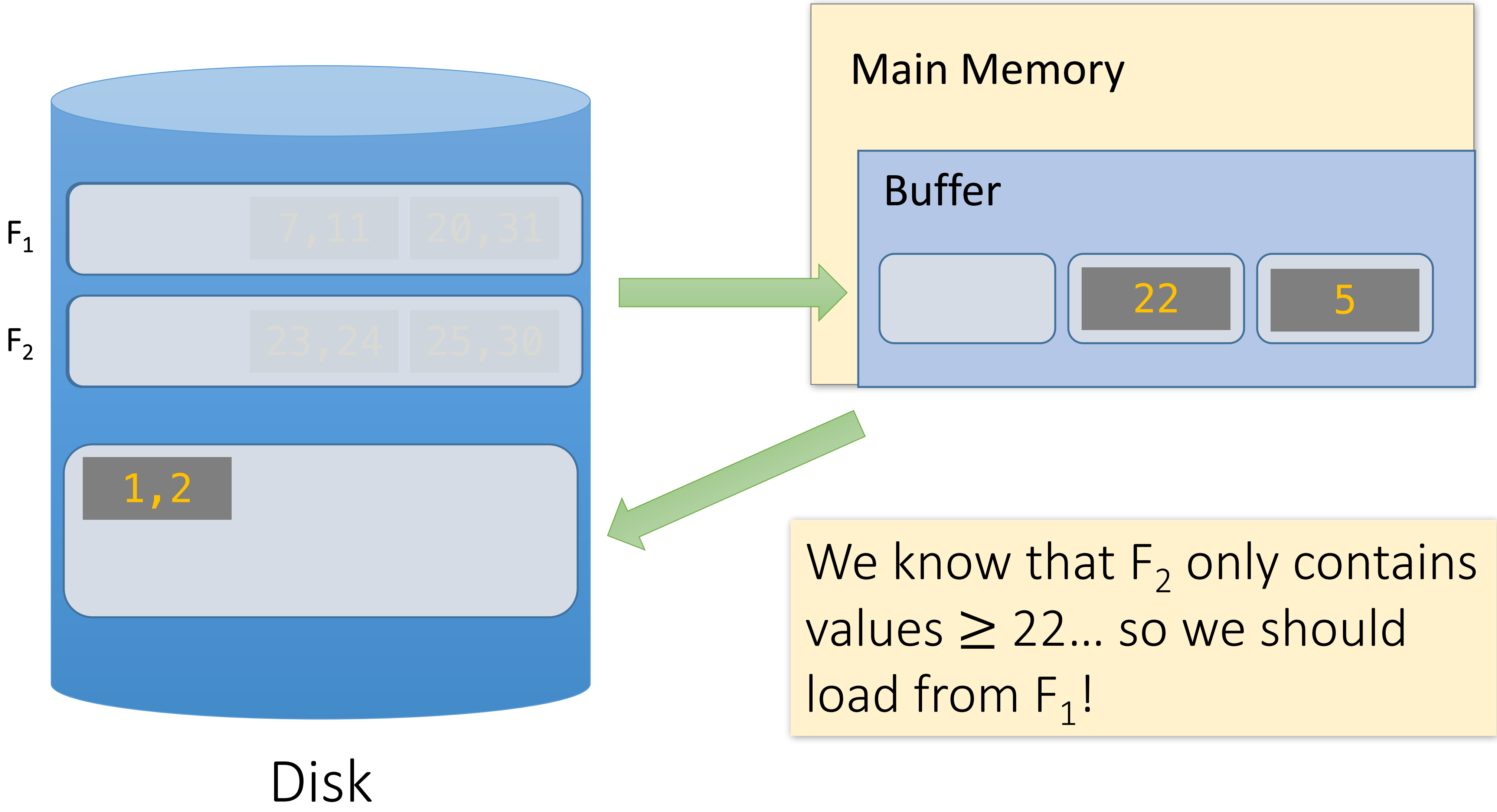


This is all the algorithm "sees"… Which file to load a page from next?

External Merge Algorithm

Input:
Two sorted files

Output:
One *merged* sorted file

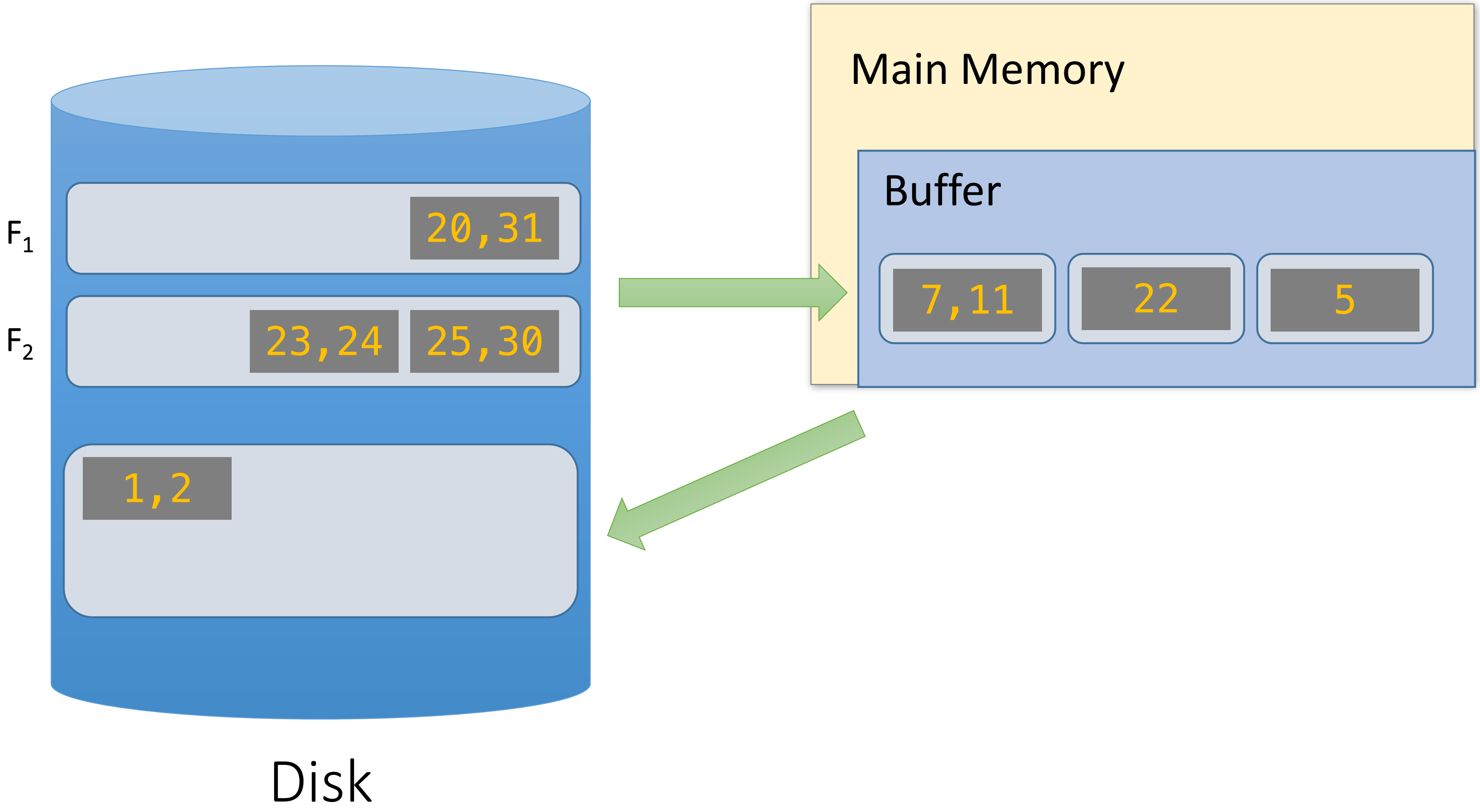


We know that F_2 only contains values ≥ 22… so we should load from F_1!

External Merge Algorithm

Input:
Two sorted files

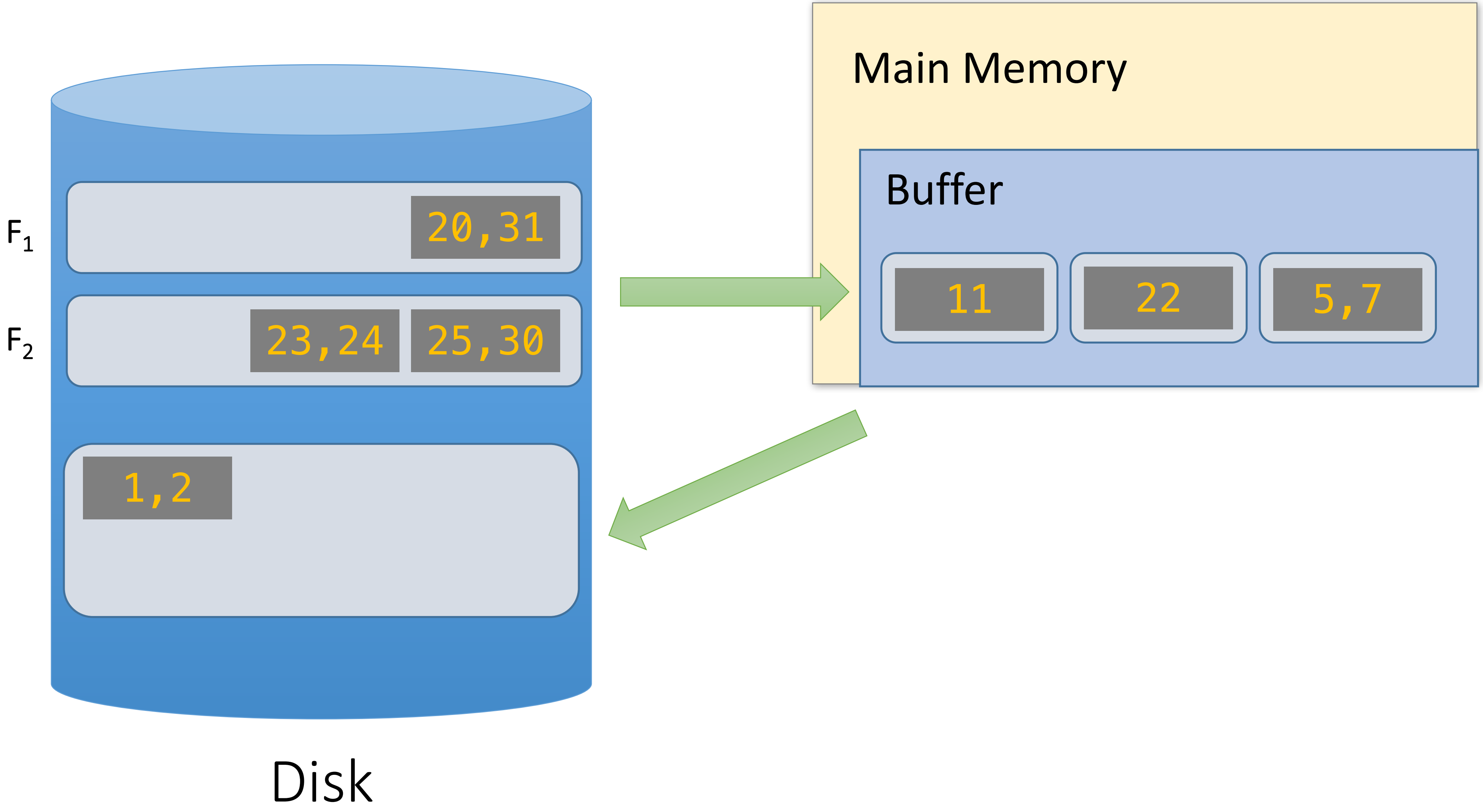
Output:
One *merged* sorted file



External Merge Algorithm

Input:
Two sorted files

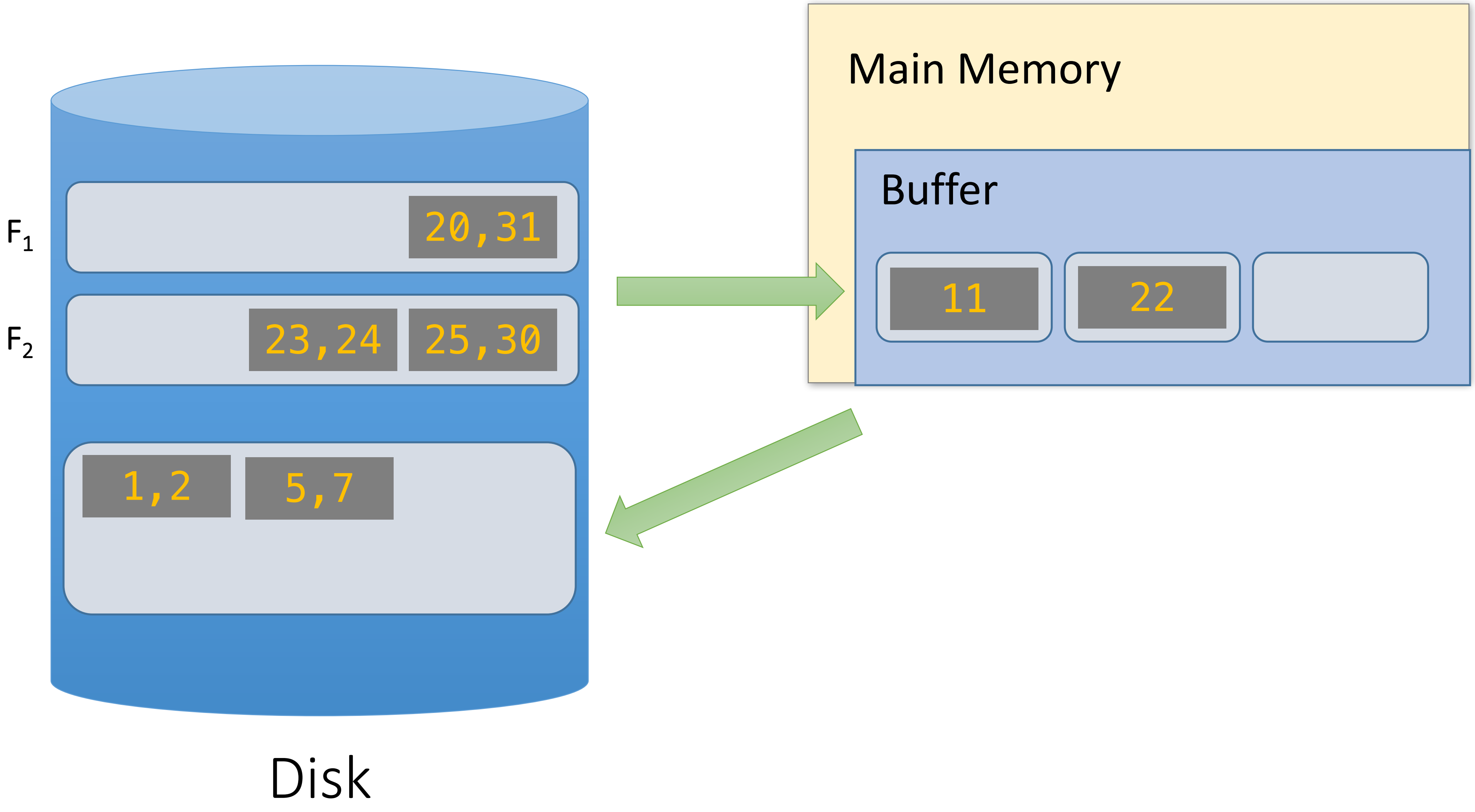
Output:
One *merged* sorted file



External Merge Algorithm

Input:
Two sorted files

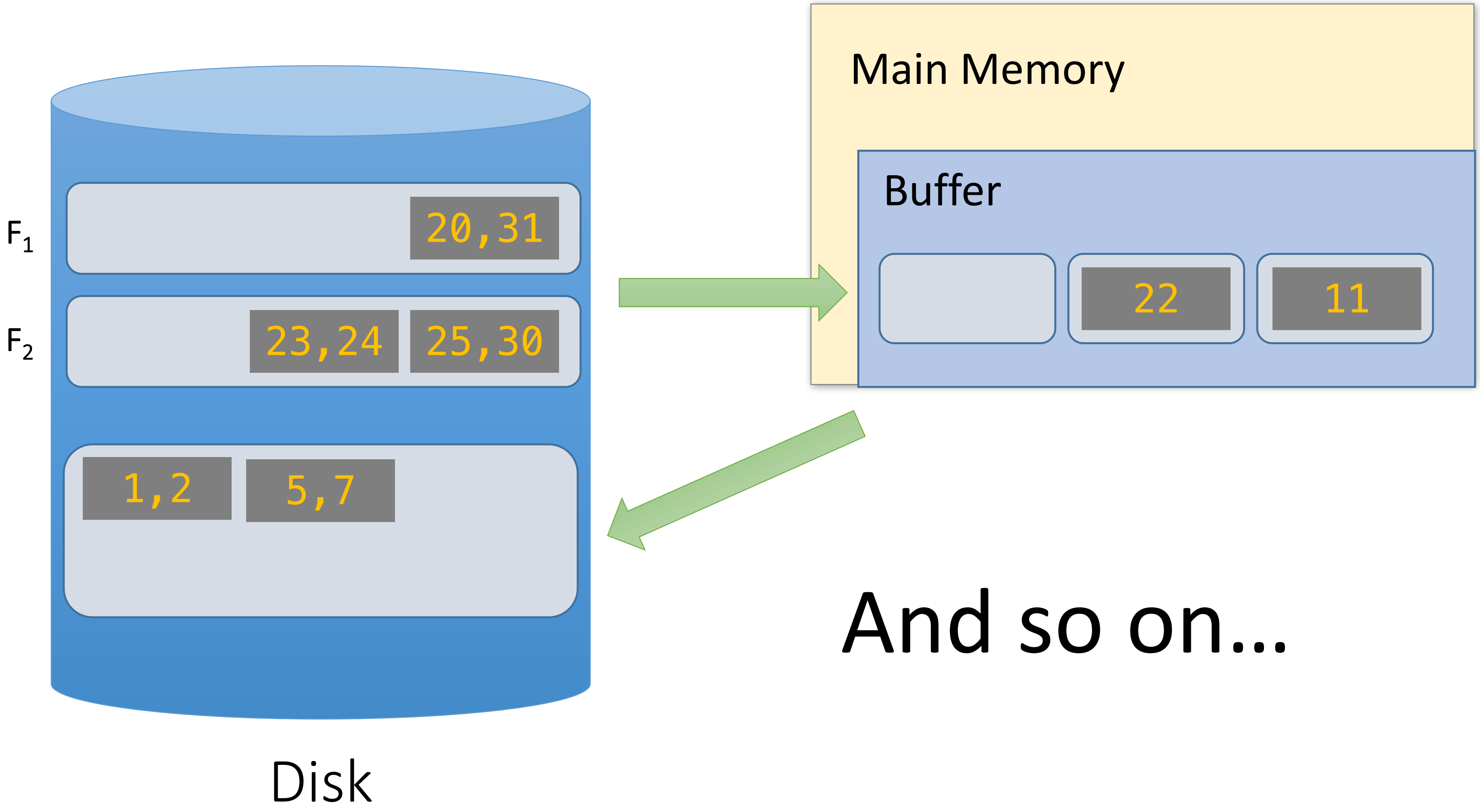
Output:
One *merged* sorted file



External Merge Algorithm

Input:
Two sorted files

Output:
One *merged* sorted file



And so on…

We can merge 2 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?

2. External Merge Sort

What you will learn about in this section

1. External merge sort (2-way sort)

2. External merge sort on larger files

3. Optimizations for sorting

External Merge Algorithm

- Suppose we want to merge two **sorted** files both much larger than main memory (i.e. the buffer)

- We can use the **external merge algorithm** to merge files of ***arbitrary length*** in **2*(N+M) IO** operations with only **3 buffer pages**!

Our first example of an "IO aware" algorithm / cost model

Why are Sort Algorithms Important?

- Data requested from DB in sorted order is **extremely common**
  - e.g., find students in increasing GPA order

- **Why not just use quicksort in main memory??**
  - What about if we need to sort 1TB of data with 1GB of RAM…

A classic problem in computer science!

More reasons to sort…

- Sorting useful for eliminating *duplicate copies* in a collection of records (Why?)

- Sorting is first step in *bulk loading* B+ tree index.

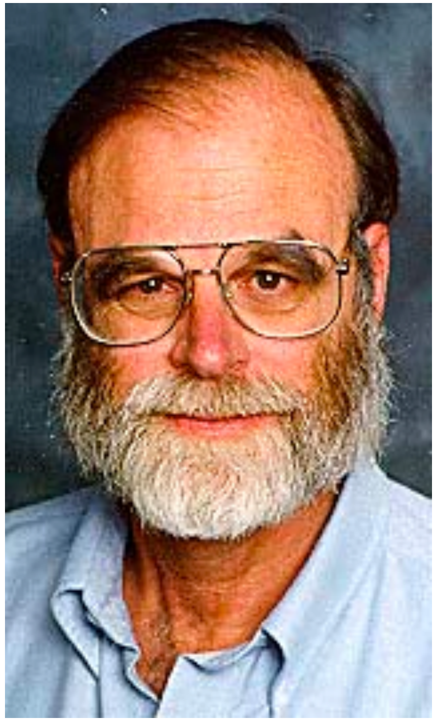
Coming up…

Coming up…

- *Sort-merge* join algorithm involves sorting

Do people care?

http://sortbenchmark.org



Sort benchmark bears his name

External Merge Sort

So how do we sort big files?

1. Split into chunks small enough to **sort in memory *("runs")***

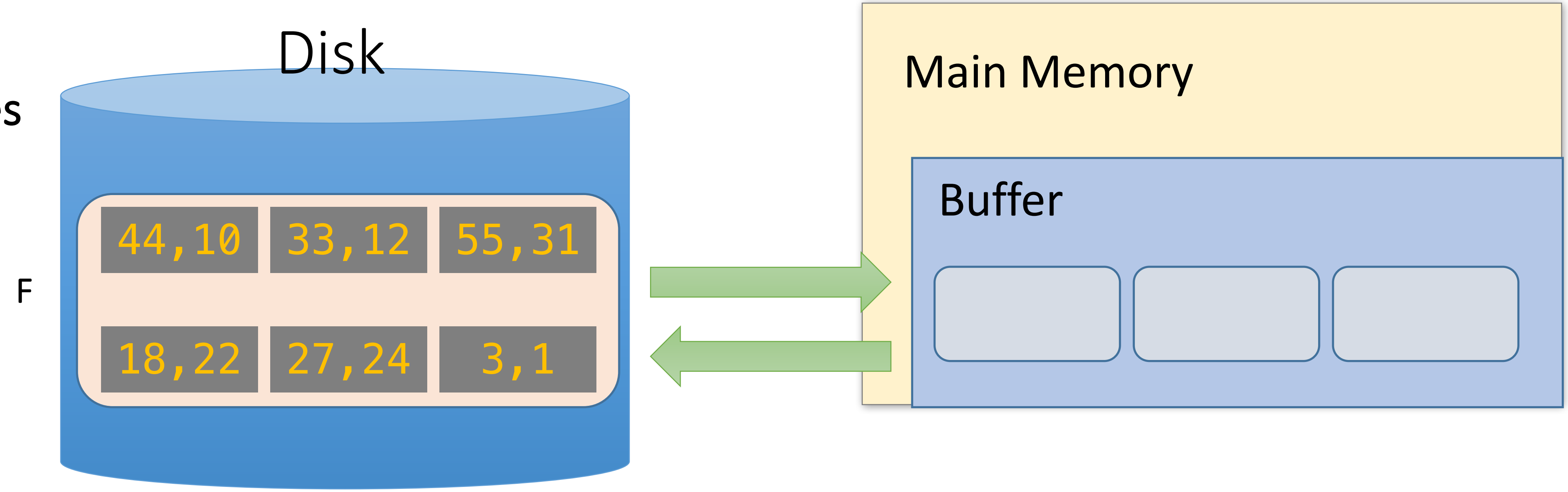
2. **Merge** pairs (or groups) of runs ***using the external merge algorithm***

3. **Keep merging** the resulting runs ***(each time = a "pass")*** until left with one sorted file!

External Merge Sort Algorithm (2-way sort)

Example:
- 3 Buffer pages
- 6-page file

Orange file = unsorted

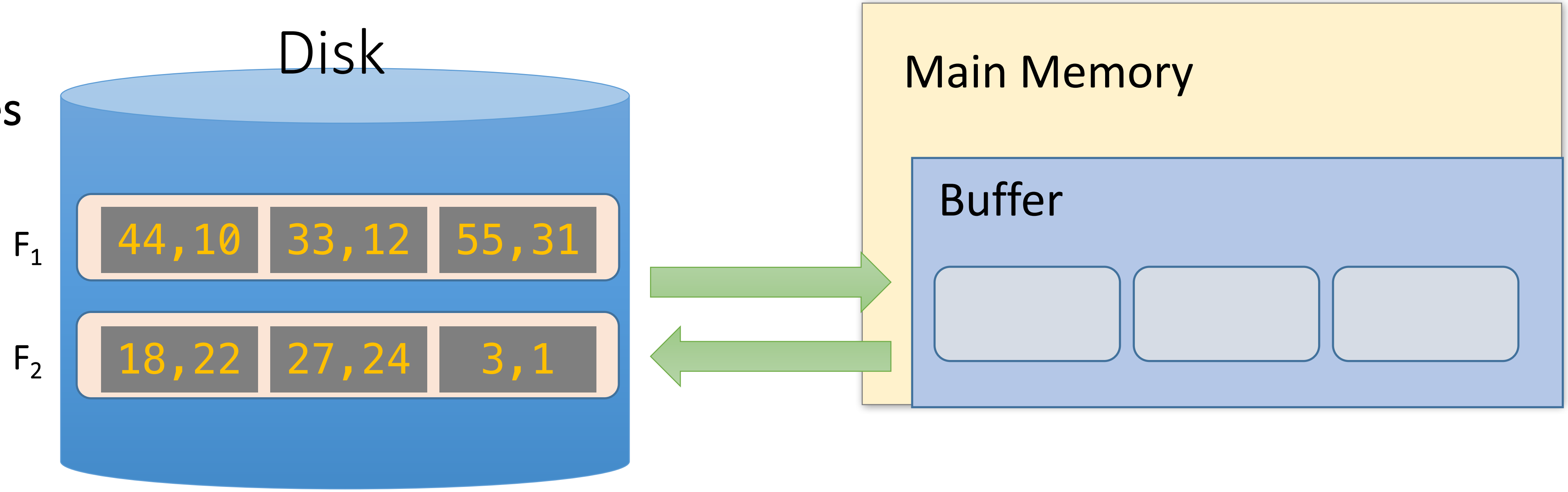


1. Split into chunks small enough to **sort in memory**

External Merge Sort Algorithm (2-way sort)

Example:
- 3 Buffer pages
- 6-page file

Orange file = unsorted

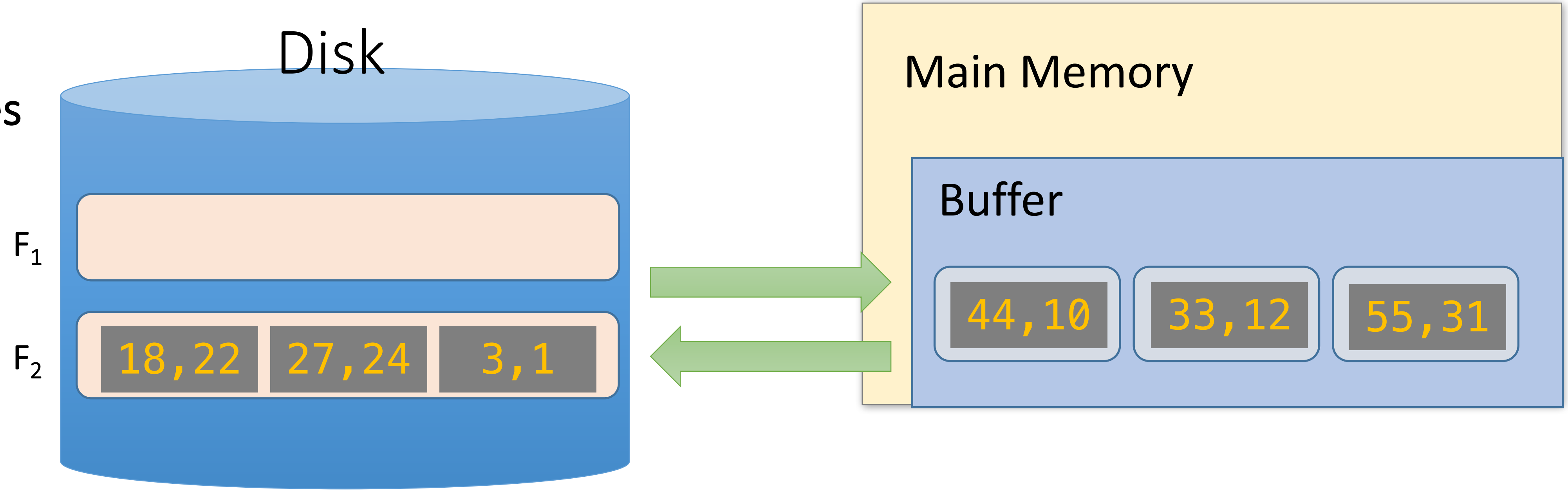


1. Split into chunks small enough to **sort in memory**

External Merge Sort Algorithm (2-way sort)

Example:
- 3 Buffer pages
- 6-page file

Orange file = unsorted

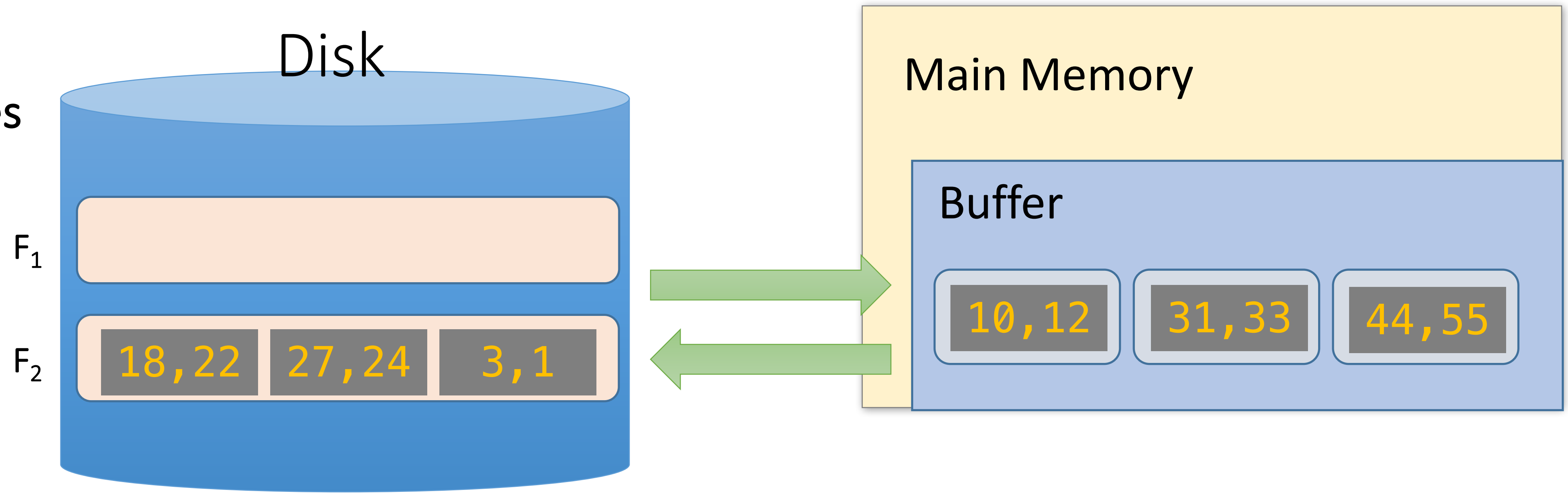


1. Split into chunks small enough to **sort in memory**

External Merge Sort Algorithm (2-way sort)

Example:
- 3 Buffer pages
- 6-page file

Orange file = unsorted

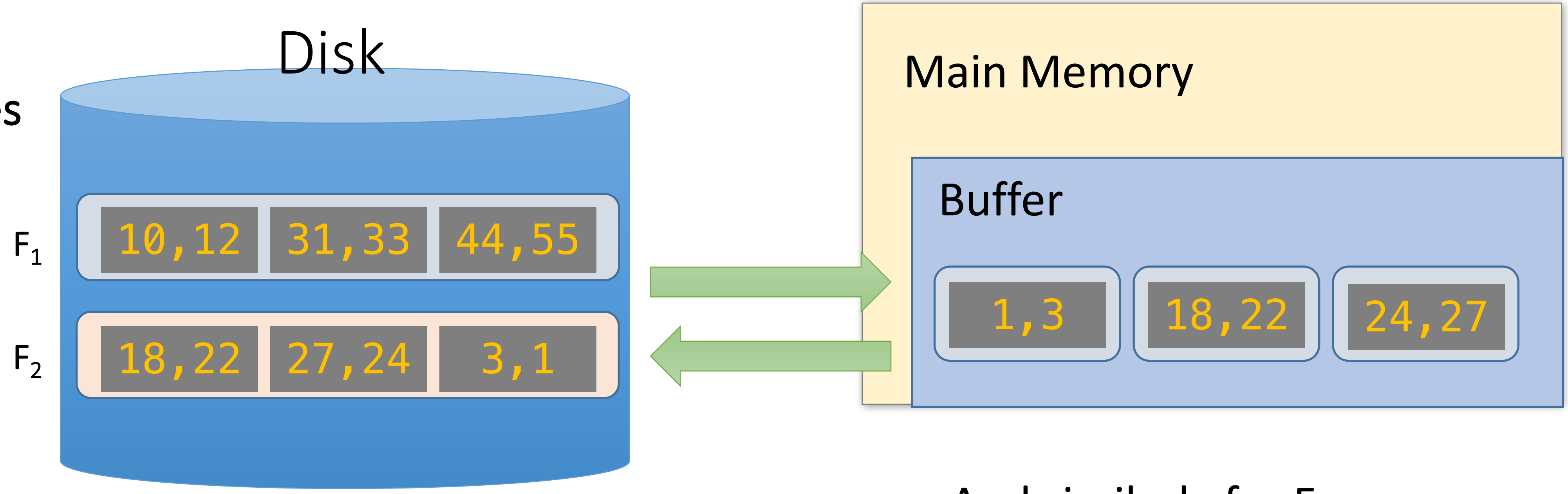


1. Split into chunks small enough to **sort in memory**

External Merge Sort Algorithm (2-way sort)

Example:
- 3 Buffer pages
- 6-page file

Each sorted file is a *run*



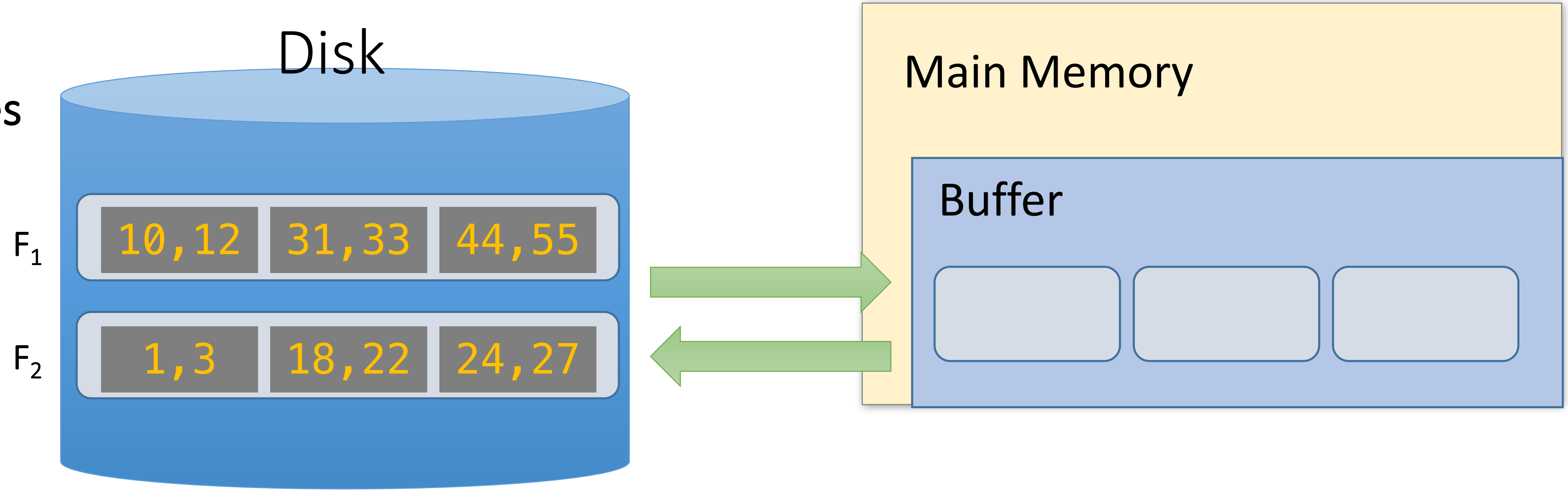
And similarly for F_2

1. Split into chunks small enough to **sort in memory**

External Merge Sort Algorithm (2-way sort)

Example:

- 3 Buffer pages
- 6-page file



2. Now just run the **external merge** algorithm & we're done!

Calculating IO Cost

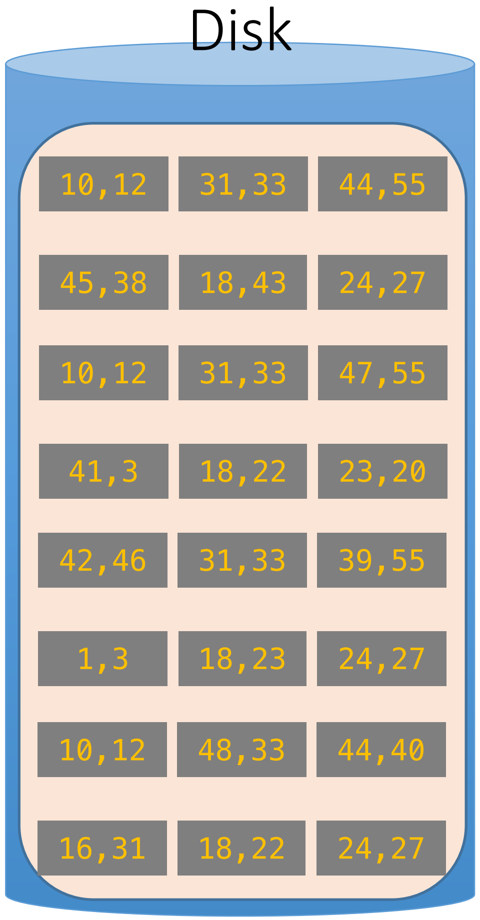
For 3 buffer pages, 6 page file:

1. Split into **two 3-page files** and **sort in memory**
    1. **= 1 R + 1 W for each file = 2*(3 + 3) = 12 IO operations**

2. **Merge** each pair of sorted chunks ***using the external merge algorithm***
    1. **= 2*(3 + 3) = 12 IO operations**

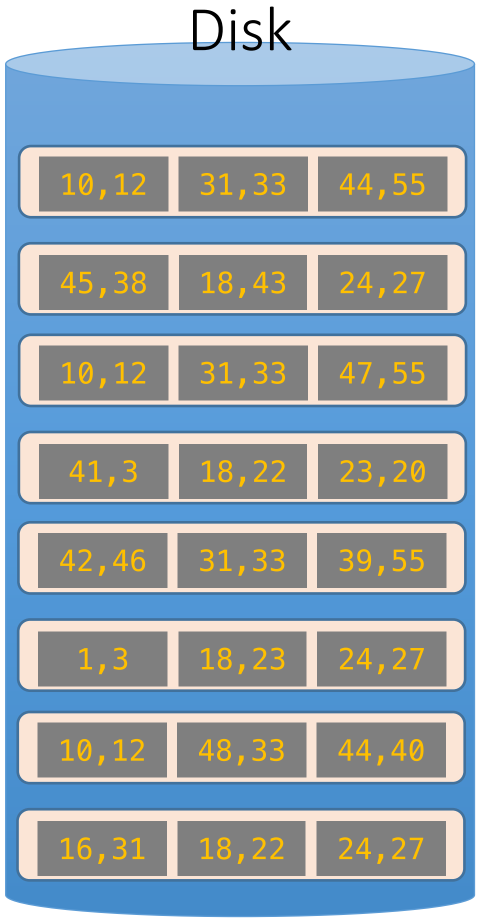
3. **Total cost = 24 IO**

Running External Merge Sort on Larger Files



Assume we still only have *3* buffer pages *(Buffer not pictured)*

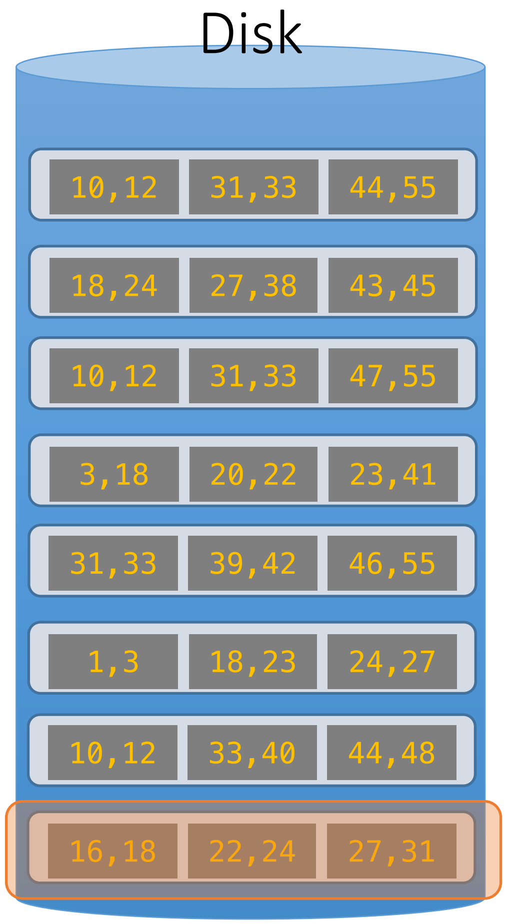
Running External Merge Sort on Larger Files



1. Split into files small enough to sort in buffer…

Assume we still only have 3 buffer pages *(Buffer not pictured)*

Running External Merge Sort on Larger Files

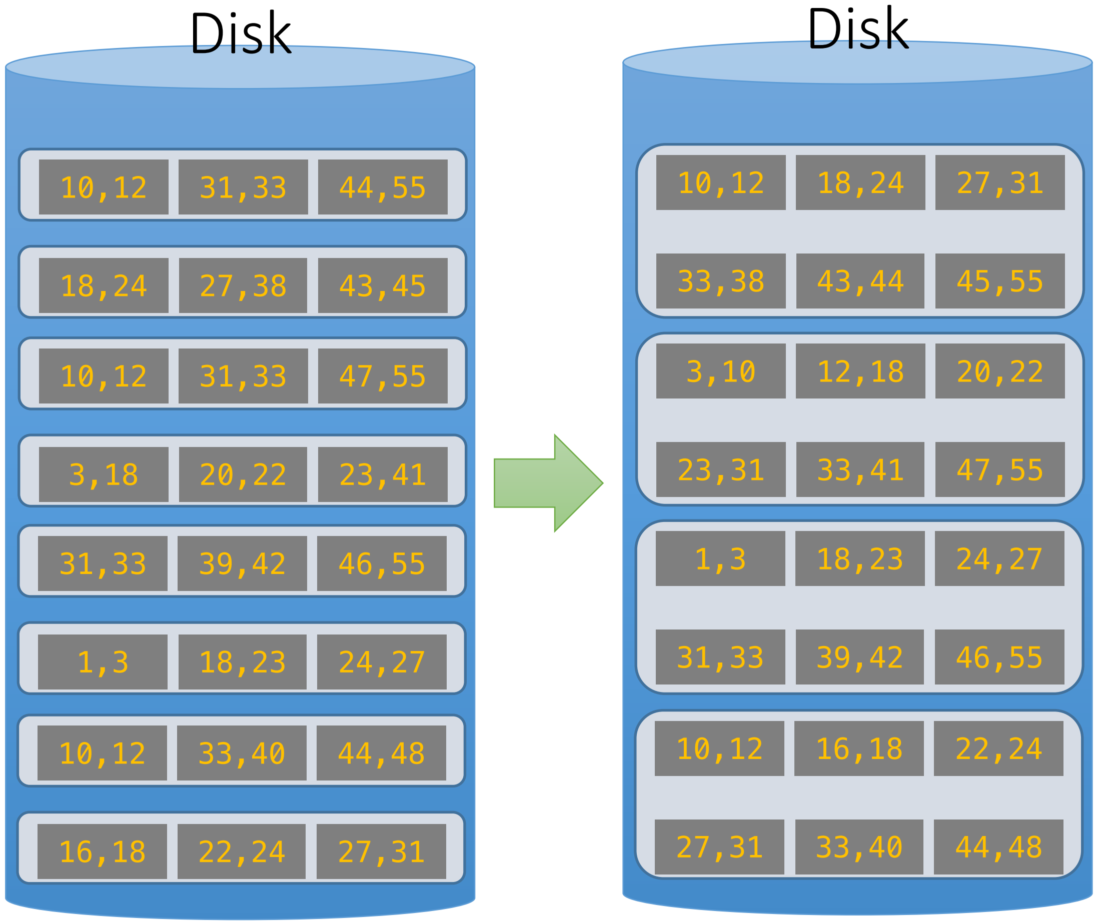


1. Split into files small enough to sort in buffer… and sort

Assume we still only have *3* buffer pages *(Buffer not pictured)*

Call each of these sorted files a ***run***

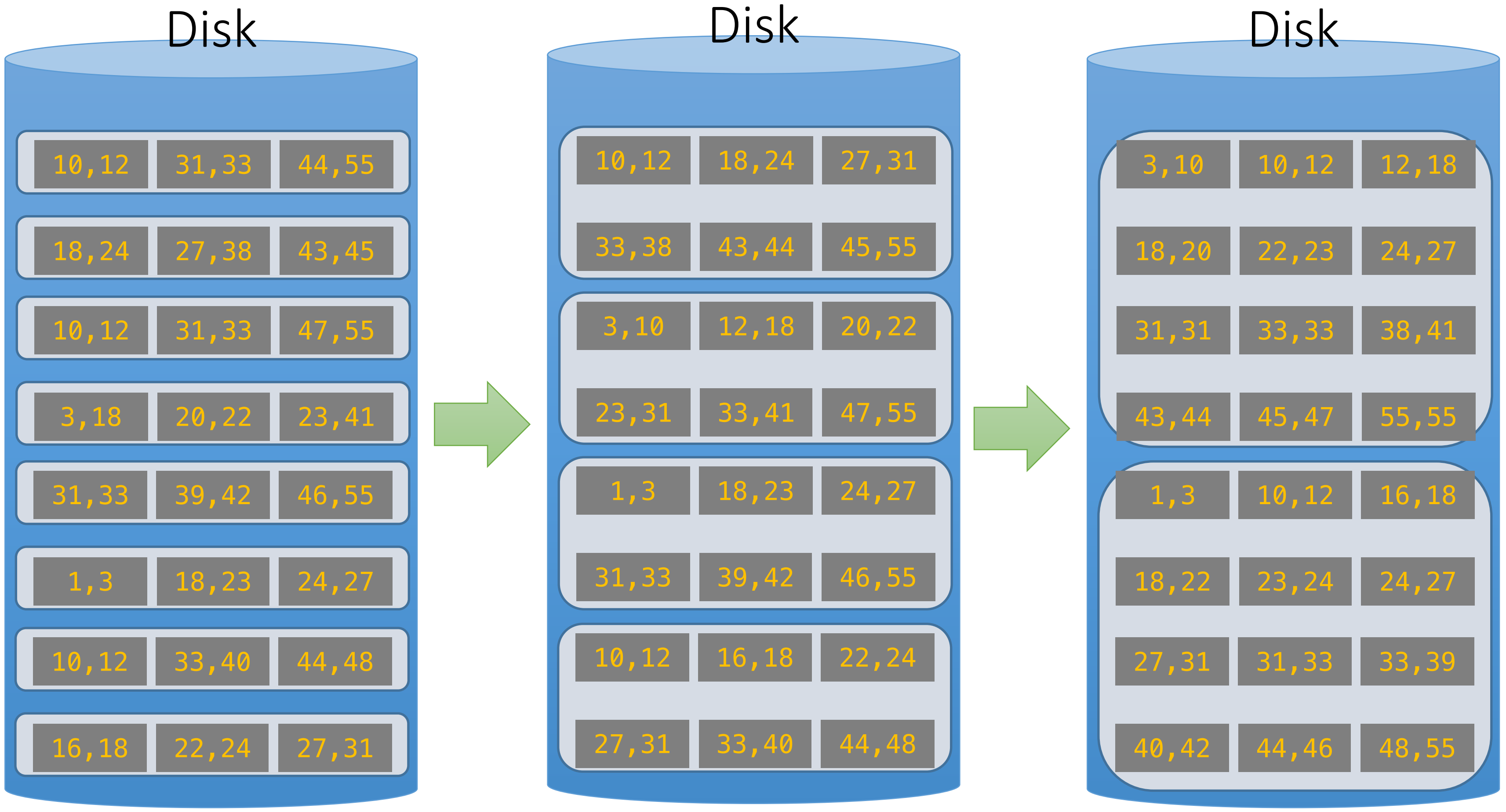
Running External Merge Sort on Larger Files



Assume we still only have *3* buffer pages *(Buffer not pictured)*

2. Now merge pairs of (sorted) files... **the resulting files will be sorted!**

Running External Merge Sort on Larger Files

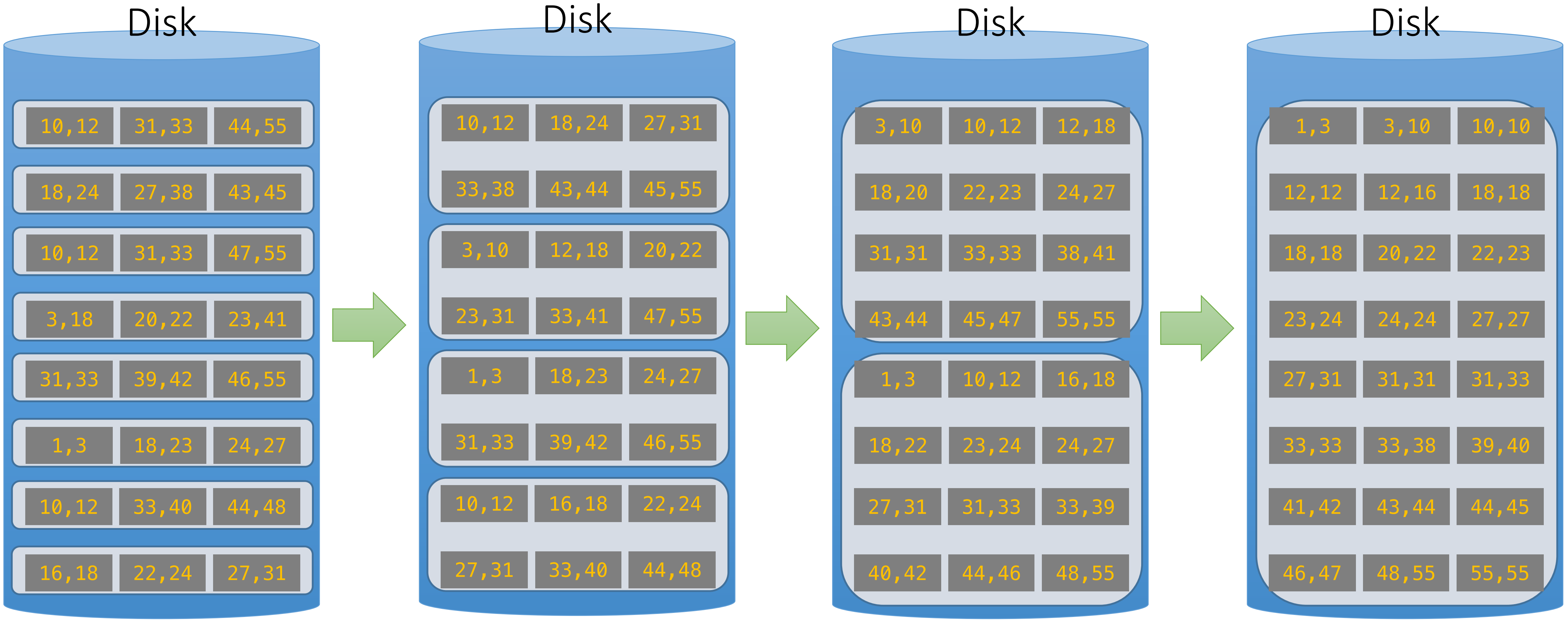


Assume we still only have *3* buffer pages *(Buffer not pictured)*

3. And repeat…

Call each of these steps a ***pass***

Running External Merge Sort on Larger Files

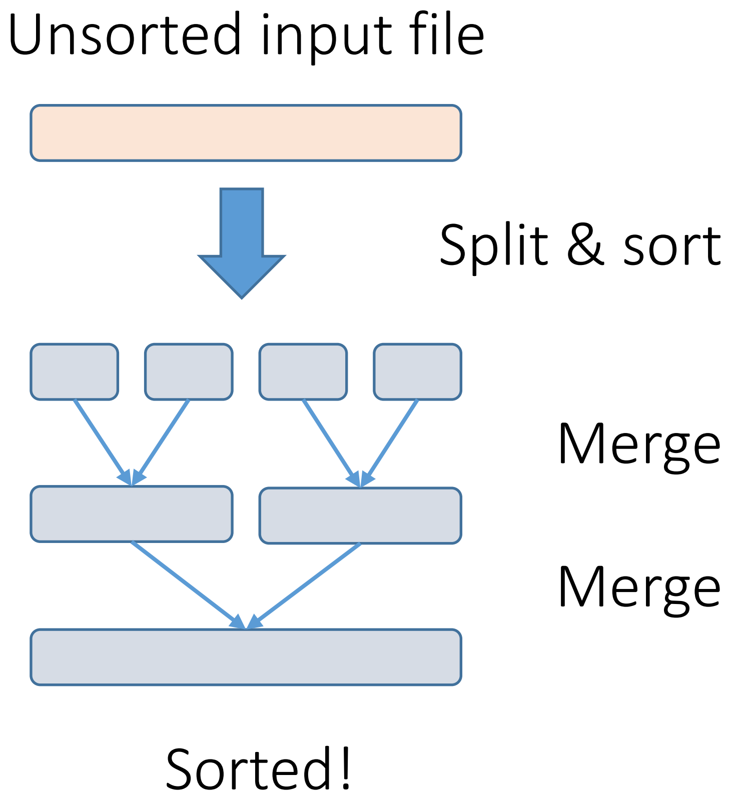


4. And repeat!

Simplified 3-page Buffer Version

Assume for simplicity that we split an N-page file into N single-page ***runs*** and sort these; then:

- First pass: Merge **N/2 *pairs* of runs** each of length **1 page**
- Second pass: Merge **N/4 *pairs* of runs** each of length **2 pages**
- In general, for **N** pages, we do $\lceil \log_2 N \rceil$ passes
  - +1 for the initial split & sort
- Each pass involves reading in & writing out all the pages = ***2N IO***



→ 2N*($\lceil \log_2 N \rceil$+1) total IO cost!

Using B+1 buffer pages to reduce # of passes

Suppose we have B+1 buffer pages now; we can:

1. **Increase length of initial runs**. Sort B+1 at a time!

At the beginning, we can split the N pages into runs of length B+1 and sort these in memory

IO Cost:

$$2N(\lceil \log_2 N \rceil + 1) \Rightarrow 2N\left(\left\lceil \log_2 \frac{N}{B+1} \right\rceil + 1\right)$$

Starting with runs of length 1

Starting with runs of length ***B+1***

Using B+1 buffer pages to reduce # of passes

Suppose we have B+1 buffer pages now; we can:

2. Perform a B-way merge.

On each pass, we can merge groups of ***B*** runs at a time (vs. merging pairs of runs)!

<u>IO Cost:</u>

$$2N(\lceil \log_2 N \rceil + 1) \rightarrow 2N\left(\left\lceil \log_2 \frac{\boldsymbol{N}}{\boldsymbol{B+1}} \right\rceil + 1\right) \rightarrow 2N\left(\left\lceil \log_B \frac{\boldsymbol{N}}{\boldsymbol{B+1}} \right\rceil + 1\right)$$

Starting with runs of length 1

Starting with runs of length ***B+1***

Performing ***B***-way merges

Repacking

Repacking for even longer initial runs

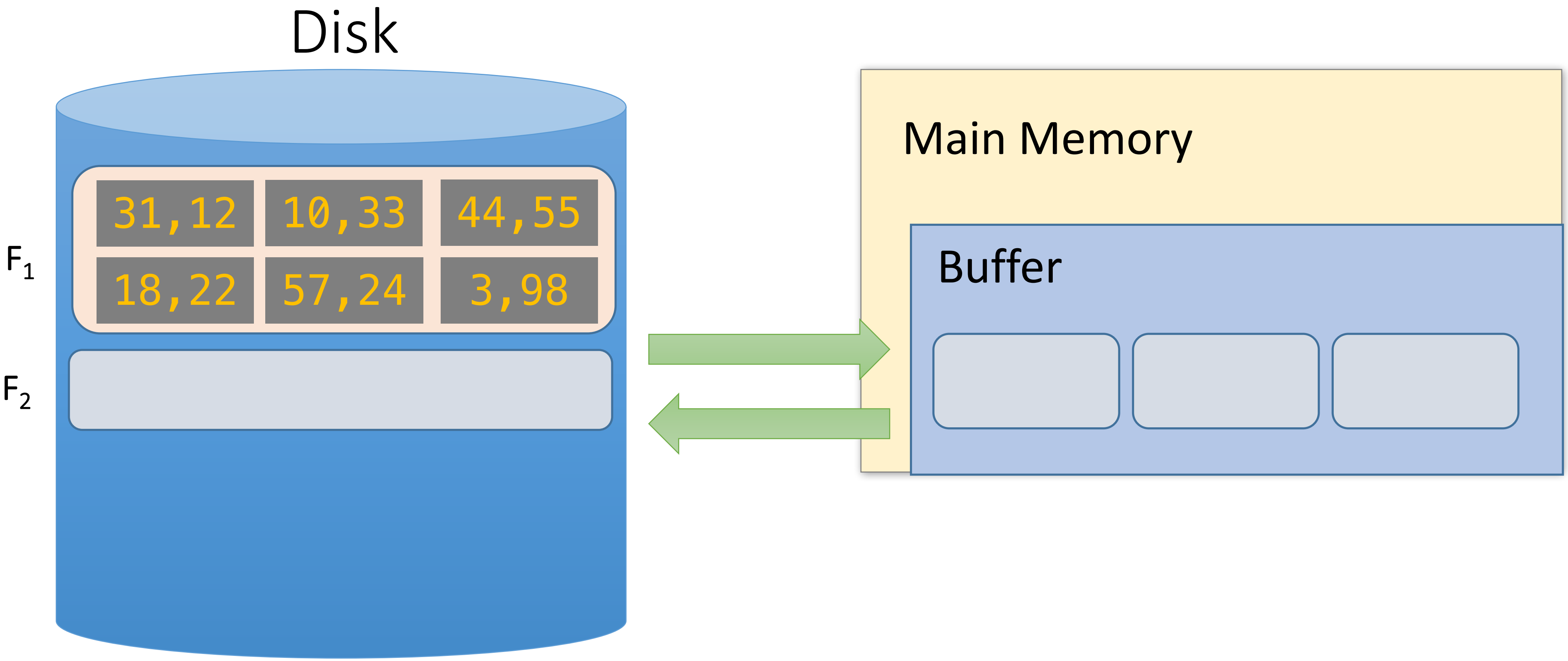
- With B+1 buffer pages, we can now start with ***B+1-length initial runs*** (and use ***B-way merges***) to get $2N(\left\lceil \log_B \frac{\boldsymbol{N}}{\boldsymbol{B+1}} \right\rceil + 1)$ IO cost…

- Can we reduce this cost more by getting even longer initial runs?

- Use **<u>repacking</u>**- produce longer initial runs by “merging” in buffer as we sort at initial stage

Repacking Example: 3 page buffer

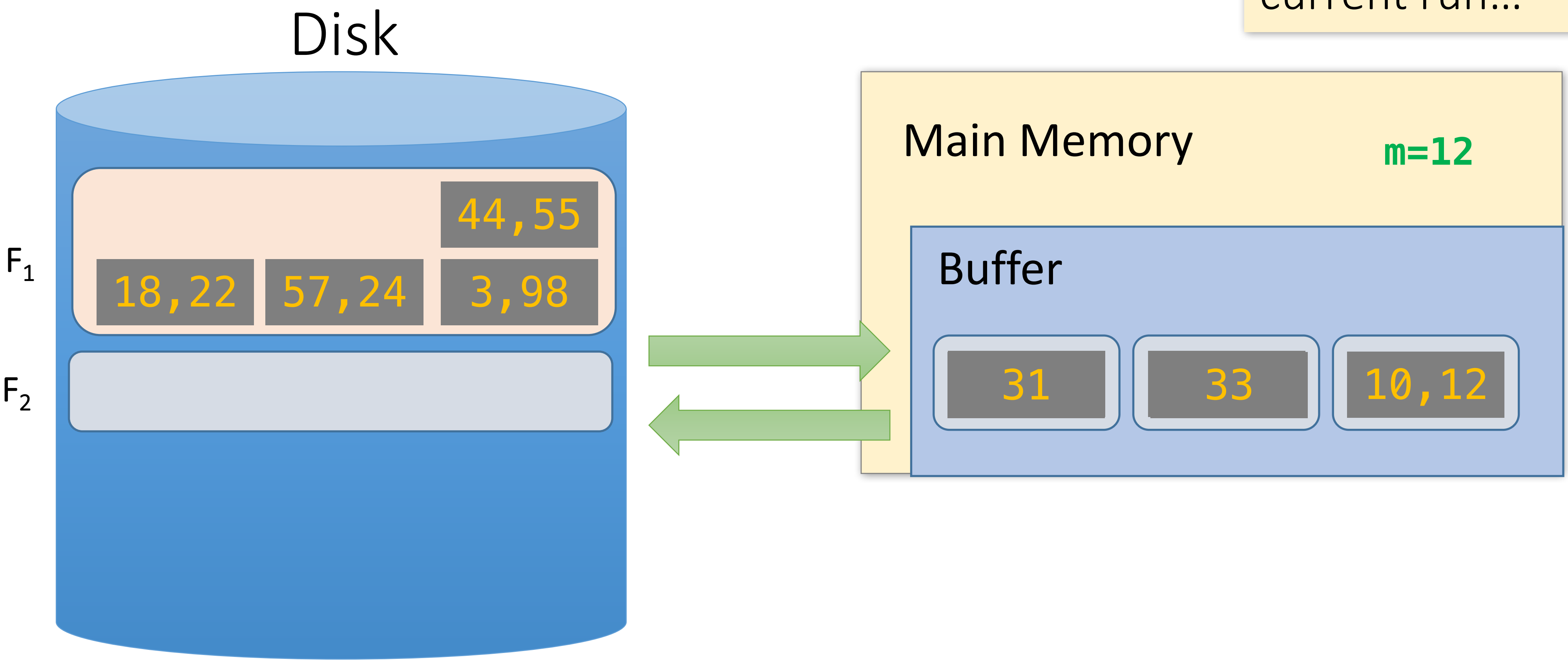
- Start with unsorted single input file, and load 2 pages



Repacking Example: 3 page buffer

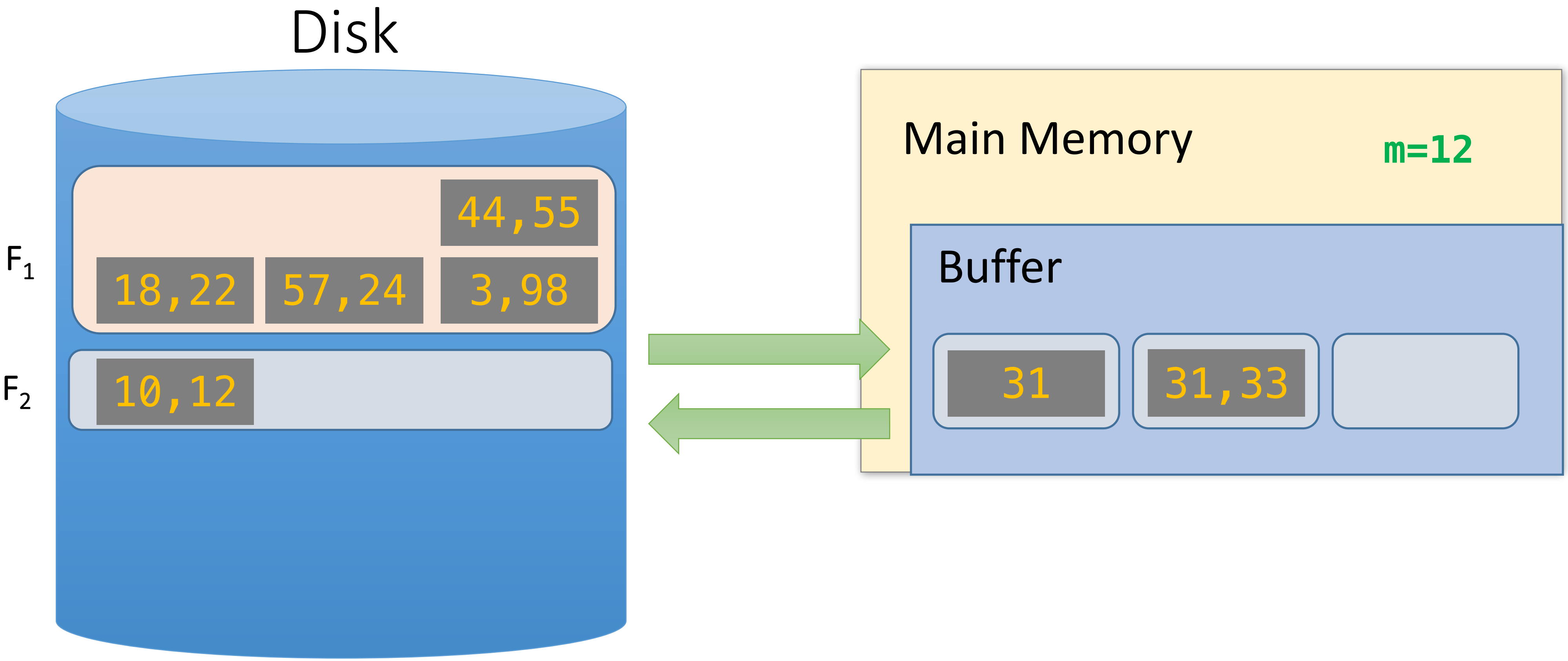
- Take the minimum two values, and put in output page

Also keep track of max (last) value in current run…



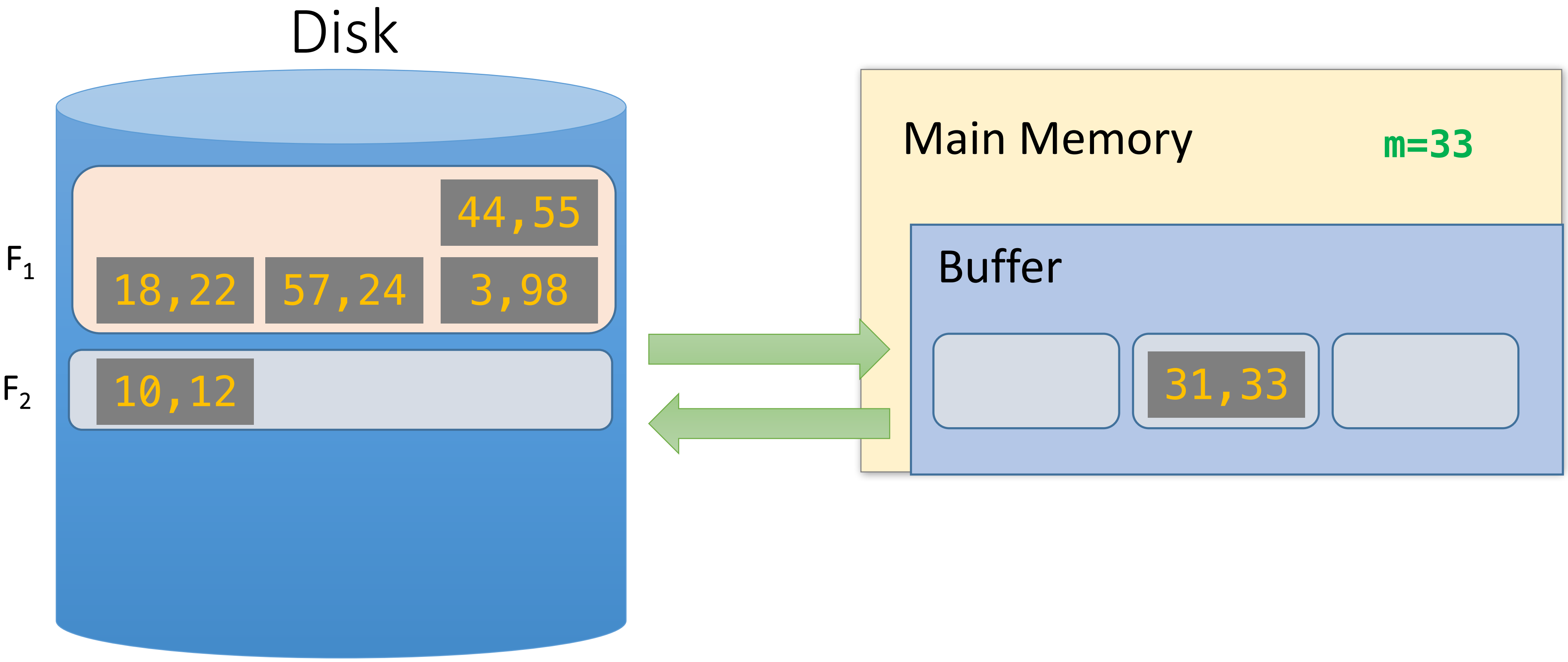
Repacking Example: 3 page buffer

- Next, ***repack***



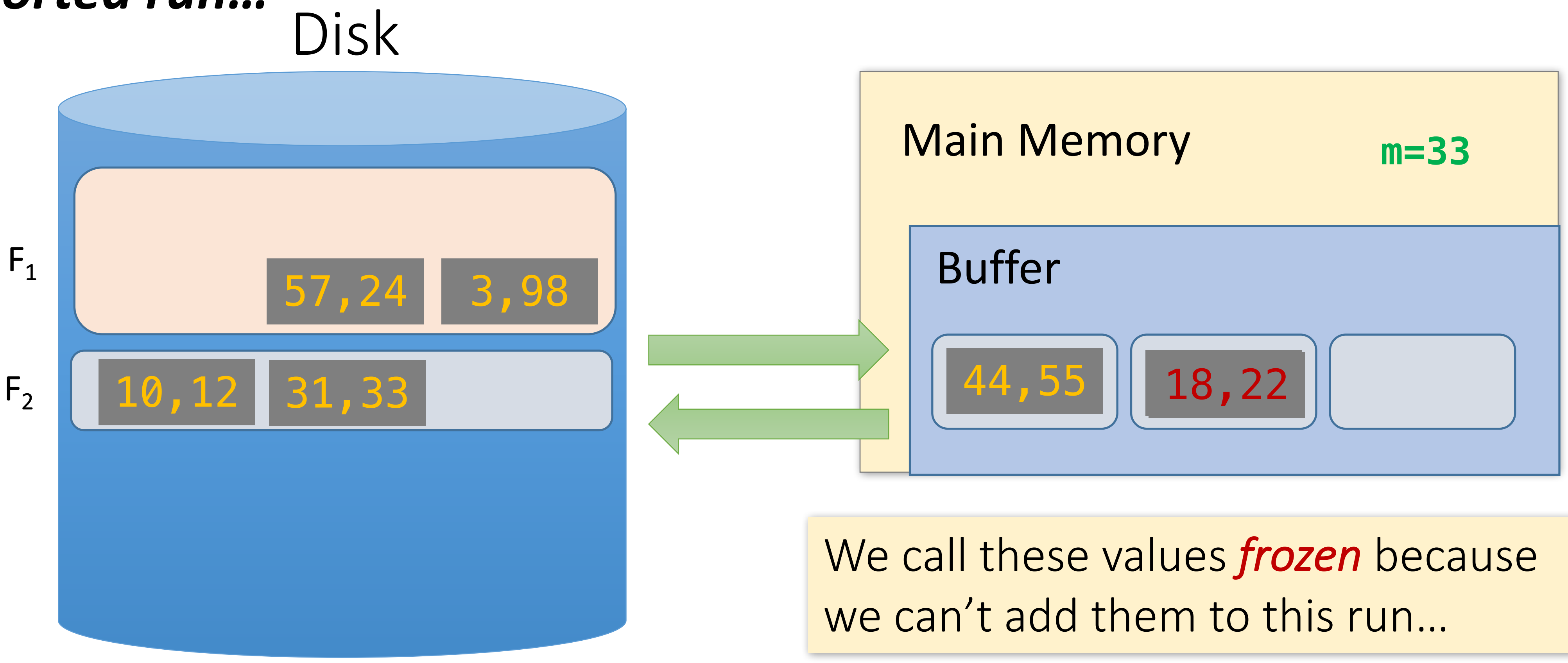
Repacking Example: 3 page buffer

- Next, ***repack***, then load another page and continue!



Repacking Example: 3 page buffer

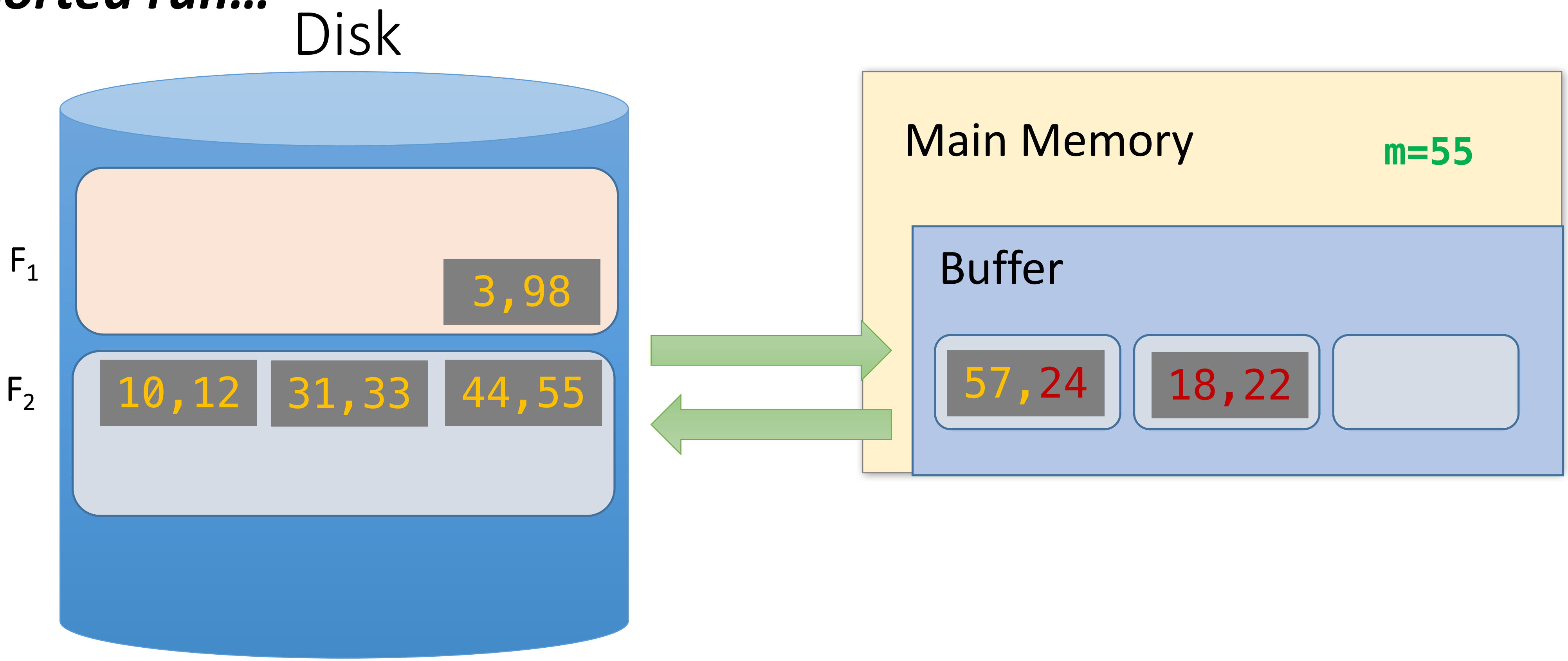
- Now, however, ***the smallest values are less than the largest (last) in the sorted run…***



We call these values ***frozen*** because we can't add them to this run…

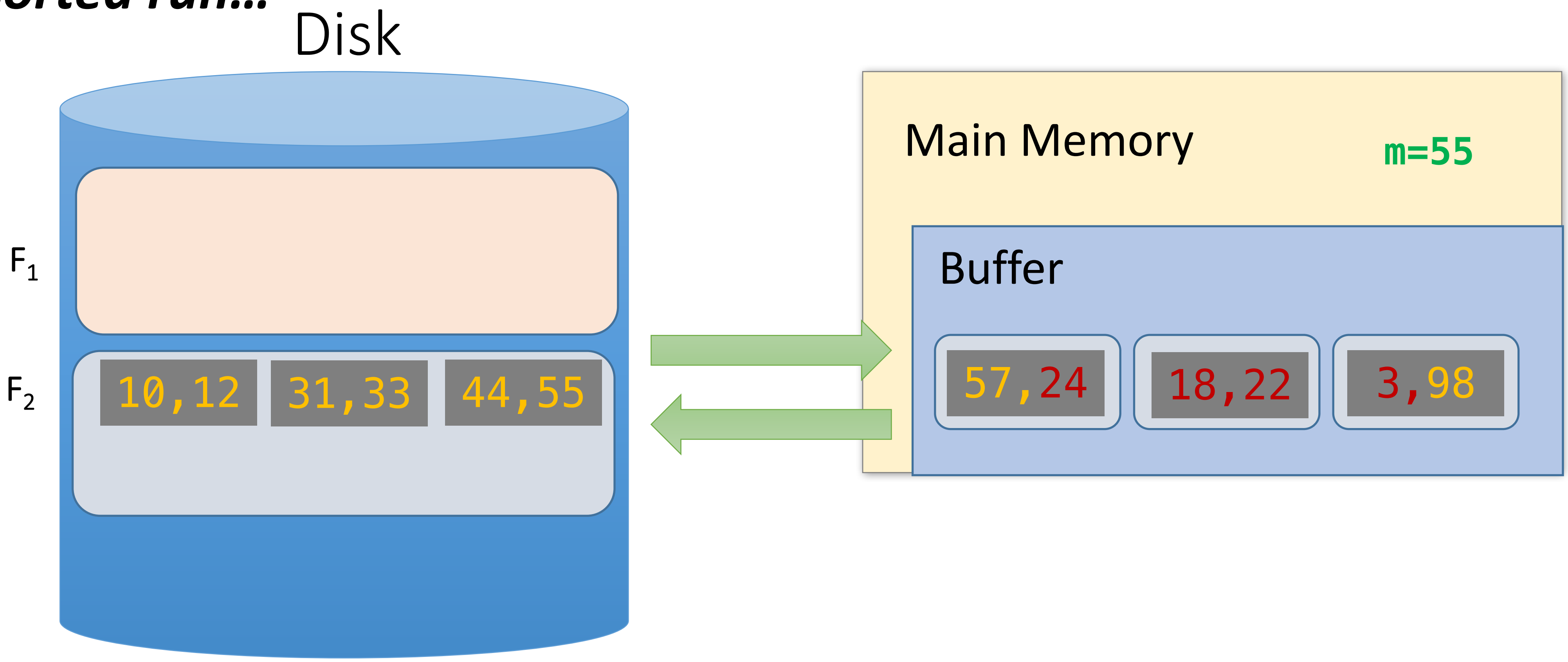
Repacking Example: 3 page buffer

- Now, however, ***the smallest values are less than the largest (last) in the sorted run…***



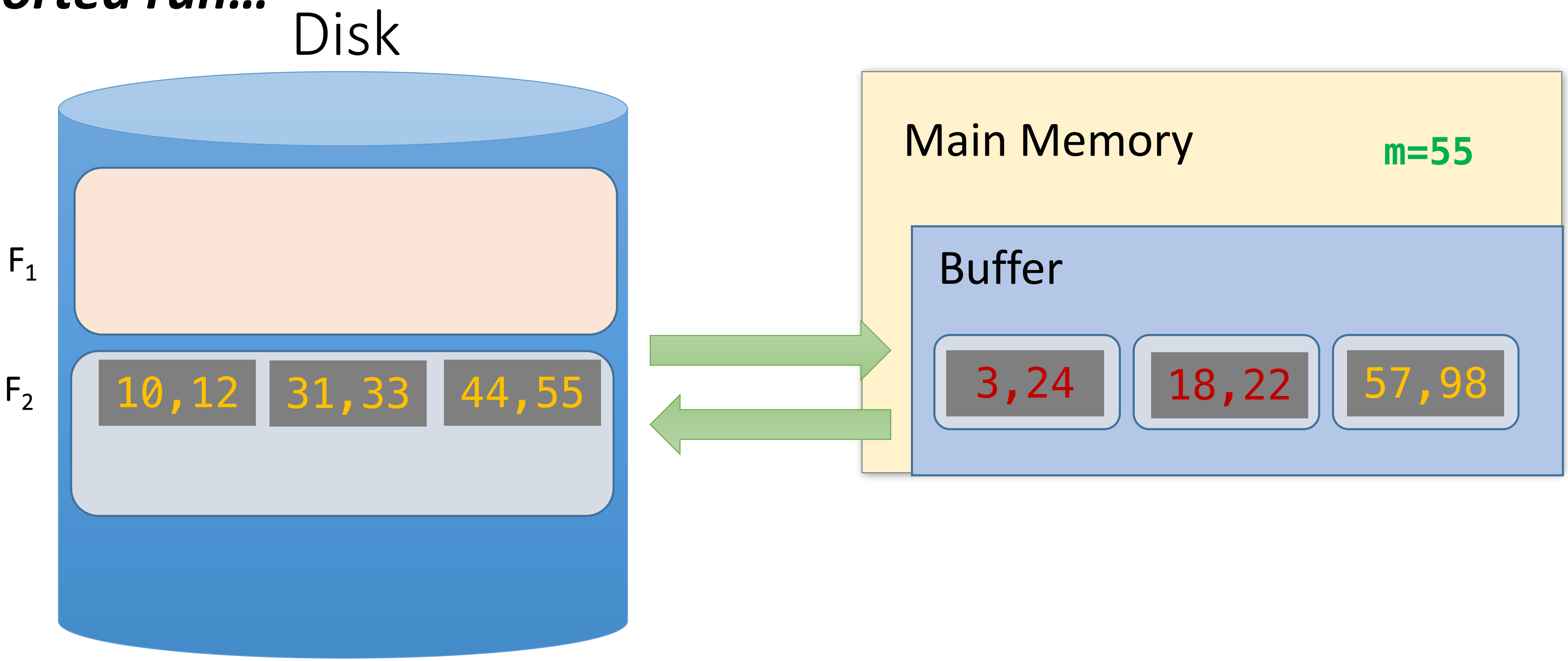
Repacking Example: 3 page buffer

- Now, however, ***the smallest values are less than the largest (last) in the sorted run…***



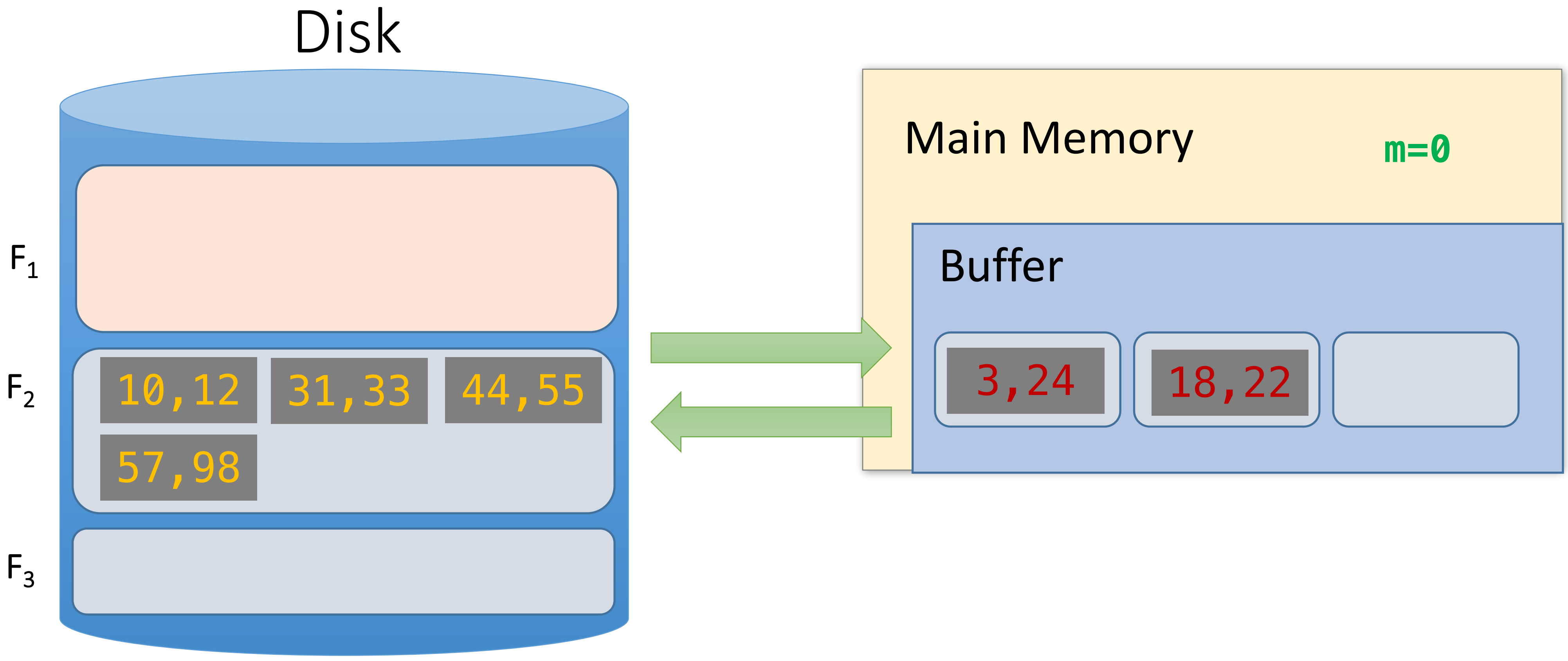
Repacking Example: 3 page buffer

- Now, however, ***the smallest values are less than the largest (last) in the sorted run…***



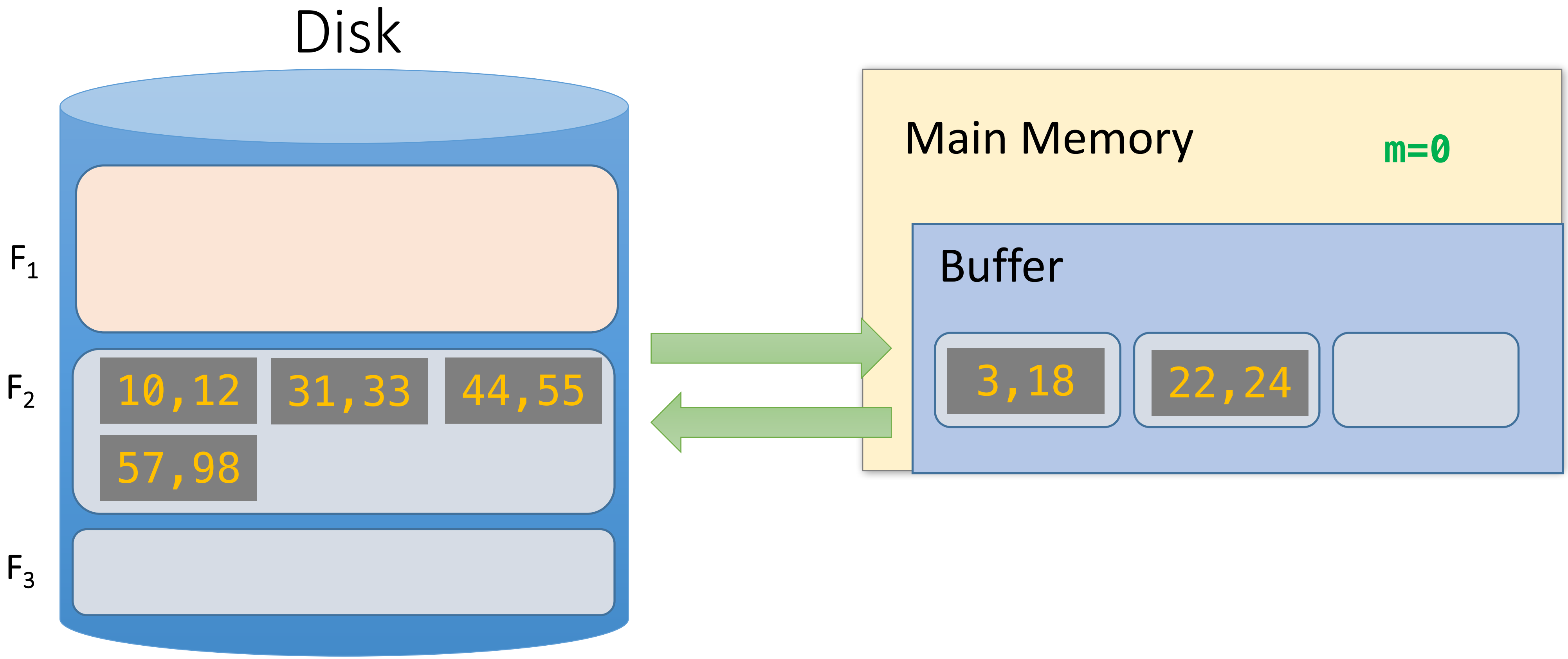
Repacking Example: 3 page buffer

- Once ***all buffer pages have a frozen value,*** or input file is empty, start new run with the frozen values



Repacking Example: 3 page buffer

- Once ***all buffer pages have a frozen value,*** or input file is empty, start new run with the frozen values



Repacking

- Note that, for buffer with B+1 pages:
  - If input file is sorted → nothing is frozen → we get **a single** run!
  - If input file is reverse sorted (worst case) → everything is frozen → we get runs of length **B+1**
- In general, with repacking we do **no worse** than without it!
- What if the file is already sorted?
- Engineer's approximation: runs will have **~2(B+1)** length

$$\sim 2N\left(\left\lceil \log_B \frac{\boldsymbol{N}}{\mathbf{2}(\boldsymbol{B}+\mathbf{1})} \right\rceil + 1\right)$$

Summary

- Basics of IO and buffer management.
- We introduced the IO cost model using **sorting**.
    - Saw how to do merges with few IOs,
    - Works better than main-memory sort algorithms.
- Described a few optimizations for sorting