DB Normalization and Buffer

Prof. Hyuk-Yoon Kwon
https://sites.google.com/view/seoultech-bigdata

Most parts are based on slides used in Stanford (http://web.stanford.edu/class/cs145)

Boyce-Codd Normal Form

Superkeys and Keys

Keys and Superkeys

A **superkey** is a set of attributes $\mathbf{A_1, \ldots, A_n}$ s.t.
for *any other* attribute **B** in R,
we have $\{\mathbf{A_1, \ldots, A_n}\} \rightarrow \mathbf{B}$

I.e. all attributes are *functionally determined* by a superkey

A **key** is a *minimal* superkey

This means that no subset of a key is also a superkey (i.e., dropping any attribute from the key makes it no longer a superkey)

Finding Keys and Superkeys

- **For each set of attributes X**

1. Compute X^+
2. If X^+ = set of all attributes then X is a **superkey**
3. If X is minimal, then it is a **key**

Example of Finding Keys

Product(name, price, category, color)

{name, category} → price
{category} → color

What is a key?

Example of Keys

Product(name, price, category, color)

{name, category} $\rightarrow$ price
{category} $\rightarrow$ color

$\{name, category\}^+ = \{name, price, category, color\}$

= the set of all attributes

$\Rightarrow$ this is a **superkey**

$\Rightarrow$ this is a **key**, since neither name nor category alone is a superkey

Practice #6: Finding superkeys

1. Consider relation R(A,B,C,D,E) with functional dependencies:
   - $D \rightarrow C$, $CE \rightarrow A$, $D \rightarrow A$, $AE \rightarrow D$
   - Which of the following is a superkey?
     1) A
     2) BD
     3) CDE
     4) BDE

2. Consider relation R(A,B,C,D,E,F) with functional dependencies:
   - $CDE \rightarrow B$, $ACD \rightarrow F$, $BEF \rightarrow C$, $B \rightarrow D$
   - Which of the following is a superkey?
     1) BDF
     2) ABE
     3) ADEF
     4) ABEF

3. Consider relation R(A,B,C,D,E,F,G) with functional dependencies:

- AB → C, CD → E, EF → G, FG → E, DE → C, and BC → A
- Which of the following is a superkey?
  1) ABEF
  2) BDF
  3) BDEG
  4) BDFG

Practice #7: Finding super keys and keys

- **Given the schema R={A,B,C}, define two FDs such that it has only two keys**
  - Remember the definition of super key: $\{A_1, \ldots, A_n\} \rightarrow$ B where B is any other attribute (NOT set) except for $\{A_1, \ldots, A_n\}$

- **Now, given the below relation R, define a set of FDs to result in the most keys possible.**
  - R = {A,B,C,D,E}
  - How many keys can you make? Largest number wins it all!

```
F = [(set(['A','B']), set(['C','D','E'])),
     (set(['A','C']), set(['B','D','E'])),
     (set(['A','D']), set(['C','B','E'])),
     (set(['A','E']), set(['C','D','B'])),
     (set(['B','C']), set(['A','D','E'])),
     (set(['B','D']), set(['A','C','E'])),
     (set(['B','E']), set(['A','D','C'])),
     (set(['C','D']), set(['A','B','E'])),
     (set(['C','E']), set(['A','B','D'])),
     (set(['D','E']), set(['A','B','C']))]
```

Back to Conceptual Design

Now that we know how to find FDs, it's a straight-forward process:

1. Search for "bad" FDs

2. If there are any, then *keep decomposing the table into sub-tables* until no more bad FDs

3. When done, the database schema is *normalized*

Boyce-Codd Normal Form (BCNF)

- **Main idea is that we define "good" and "bad" FDs as follows:**
  - X → A is a *"good FD" if X is a superkey*
    - In other words, if A is the set of all attributes
  - X → A is a *"bad FD"* otherwise

- **We will try to eliminate the "bad" FDs!**

Boyce-Codd Normal Form (BCNF)

- **Why does this definition of “good” and “bad” FDs make sense?**

- **If X is *not* a (super)key, it functionally determines *some* of the attributes; therefore, those other attributes can be duplicated**
  - Recall: this means there is <u>redundancy</u>
  - And redundancy like this can lead to data anomalies!

EmpID	Name	Phone	Position
E0045	Smith	1234	Clerk
E3542	Mike	9876	Salesrep
E1111	Smith	9876	Salesrep
E9999	Mary	1234	Lawyer

Example

Name	SSN	PhoneNumber	City
Fred	123-45-6789	206-555-1234	Seattle
Fred	123-45-6789	206-555-6543	Seattle
Joe	987-65-4321	908-555-2121	Westfield
Joe	987-65-4321	908-555-1234	Westfield

{SSN} → {Name,City}

This FD is *bad* because it is **not** a superkey

⟹ **Not** in BCNF

What is the key? {SSN, PhoneNumber}

Example

Name	SSN	City
Fred	123-45-6789	Seattle
Joe	987-65-4321	Madison

{SSN} → {Name,City}

This FD is now *good* because it is the key

SSN	PhoneNumber
123-45-6789	206-555-1234
123-45-6789	206-555-6543
987-65-4321	908-555-2121
987-65-4321	908-555-1234

Let's check anomalies:

- Redundancy ?
- Update ?
- Delete ?

Now in BCNF!

BCNF Decomposition Algorithm

```
BCNFDecomp(R):
   Find X s.t.: X⁺ ≠ X and X⁺ ≠ [all attributes]

   if (not found) then Return R

   let Y = X⁺ - X,  Z = (X⁺)ᶜ
   decompose R into R1(X ∪ Y) and R2(X ∪ Z)

   Return BCNFDecomp(R1), BCNFDecomp(R2)
```

BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find *a set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) **then** **Return** R

  let $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into **R1(X ∪ Y)** and **R2(X ∪ Z)**

  Return BCNFDecomp(R1), BCNFDecomp(R2)

BCNF Decomposition Algorithm

```
BCNFDecomp(R):
   Find a set of attributes X s.t.: X⁺ ≠ X and X⁺ ≠ [all attributes]

   if (not found) then Return R

   let Y = X⁺ - X,  Z = (X⁺)ᶜ
   decompose R into R1(X ∪ Y) and R2(X ∪ Z)

   Return BCNFDecomp(R1), BCNFDecomp(R2)
```

BCNF Decomposition Algorithm

BCNFDecomp(R):
  Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  if (not found) **then** **Return** R

  let $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into **R1($X \cup Y$)** and **R2($X \cup Z$)**

  Return BCNFDecomp(R1), BCNFDecomp(R2)

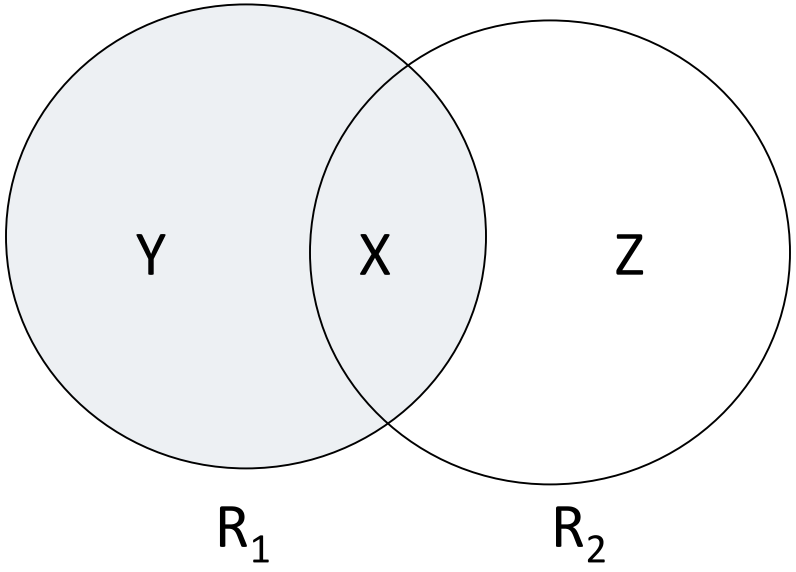
BCNF Decomposition Algorithm

BCNFDecomp(R):
Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

if (not found) **then** **Return** R

let $Y = X^+ - X$, $Z = (X^+)^C$
decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

Return BCNFDecomp(R1), BCNFDecomp(R2)



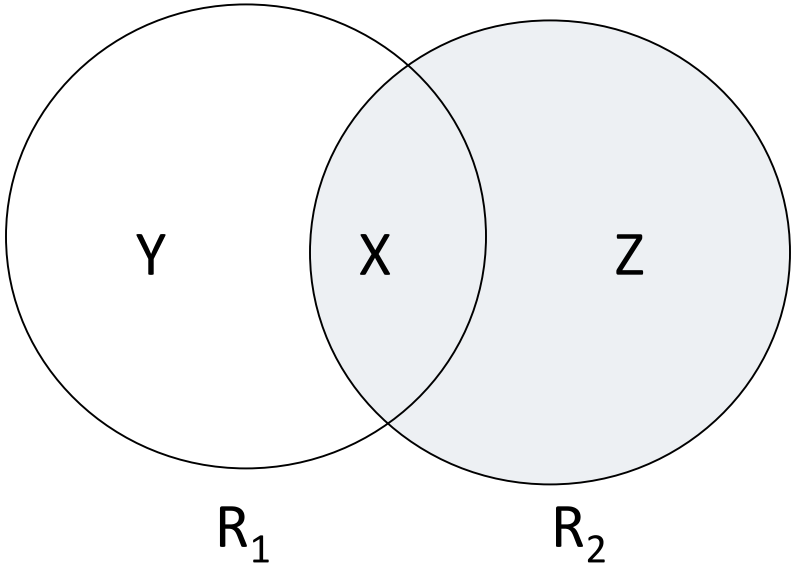
BCNF Decomposition Algorithm

BCNFDecomp(R):
   Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

   if (not found) **then** **Return** R

   let $Y = X^+ - X$, $Z = (X^+)^C$
   decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

   Return BCNFDecomp(R1), BCNFDecomp(R2)



BCNF Decomposition Algorithm

BCNFDecomp(R):
Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

if (not found) **then** **Return** R

let $Y = X^+ - X$, $Z = (X^+)^C$
decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

Return BCNFDecomp(R_1), BCNFDecomp(R_2)

Example

BCNFDecomp(R):
  Find a *set of attributes* X s.t.: $X^+ \neq X$ and $X^+ \neq$ [all attributes]

  <u>if</u> (not found) **<u>then</u> Return** R

  <u>let</u> $Y = X^+ - X$, $Z = (X^+)^C$
  decompose R into $\mathbf{R_1(X \cup Y)}$ and $\mathbf{R_2(X \cup Z)}$

  Return BCNFDecomp(R_1), BCNFDecomp(R_2)

R(A,B,C,D,E)

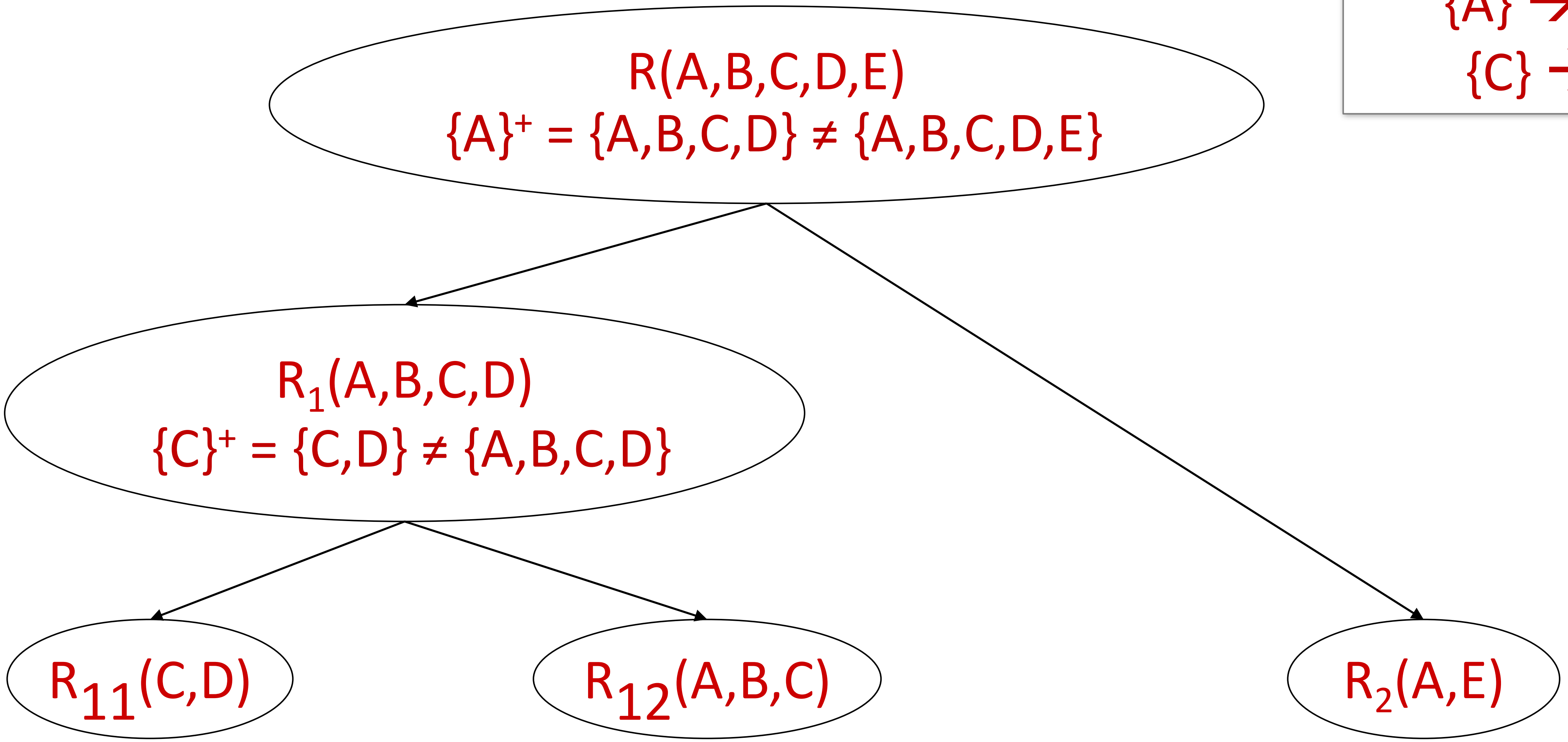
$\{A\} \rightarrow \{B,C\}$
$\{C\} \rightarrow \{D\}$

Example

R(A,B,C,D,E)

{A} → {B,C}

{C} → {D}



Practice: BCNF

1. Let R(A,B,C,D,E) be a relation in Boyce-Codd Normal Form (BCNF). Suppose ABC is the only key for R. Which of the following functional dependencies is guaranteed to hold for R?
   1) BCD -> E
   2) ACDE -> B
   3) ABC → D
   4) BCDE → A

2. Consider a relation R(A,B,C,D). For which of the following sets of FDs is R in Boyce-Codd Normal Form (BCNF)?
   1) ABC → D, BCD → A, D → C, ACD → B
   2) A → D, C → A, D → B, AC → B
   3) BD → C, AB → D, AC → B, BD → A
   4) C → D, CD → A, AB → C, BD → A

3. Consider a relation R(A,B,C,D,E). For which of the following sets of FDs is R in Boyce-Codd Normal Form (BCNF)?

1) BE -> D, B -> E, D -> E, CD -> A
2) ACD → E, AE → C, CE → B, A → D
3) ABD → C, ACD → E, ACE → B, BC → E
4) AC → D, BCE → A, CD → E, CE → B

Practice: BCNF

1. **Make the following table using CREATE TABLE and INSERT**

emp_id	emp_nationality	emp_dept	dept_type	dept_no_of_emp
1001	Austrian	Production and planning	D001	200
1001	Austrian	stores	D001	250
1002	American	design and technical support	D134	100
1002	American	Purchasing department	D134	600

2. **Decompose the table above into multiple tables to satisfy BCNF in the schema level**
   - Functional dependencies in the table above:

     emp_id -> emp_nationality

     emp_dept -> {dept_type, dept_no_of_emp}

3. **Make the actual tables and insert values**
   - Compare decomposed tables with original table

Today's Lecture

1. **The Buffer**

2. **External Merge Sort**

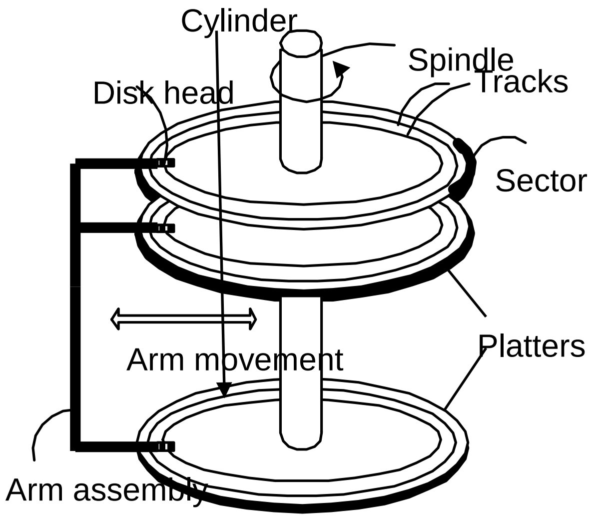
The Buffer

What you will learn about in this section

1. **Storage and memory model**

2. **Buffer primer**

High-level: Disk vs. Main Memory



Disk:

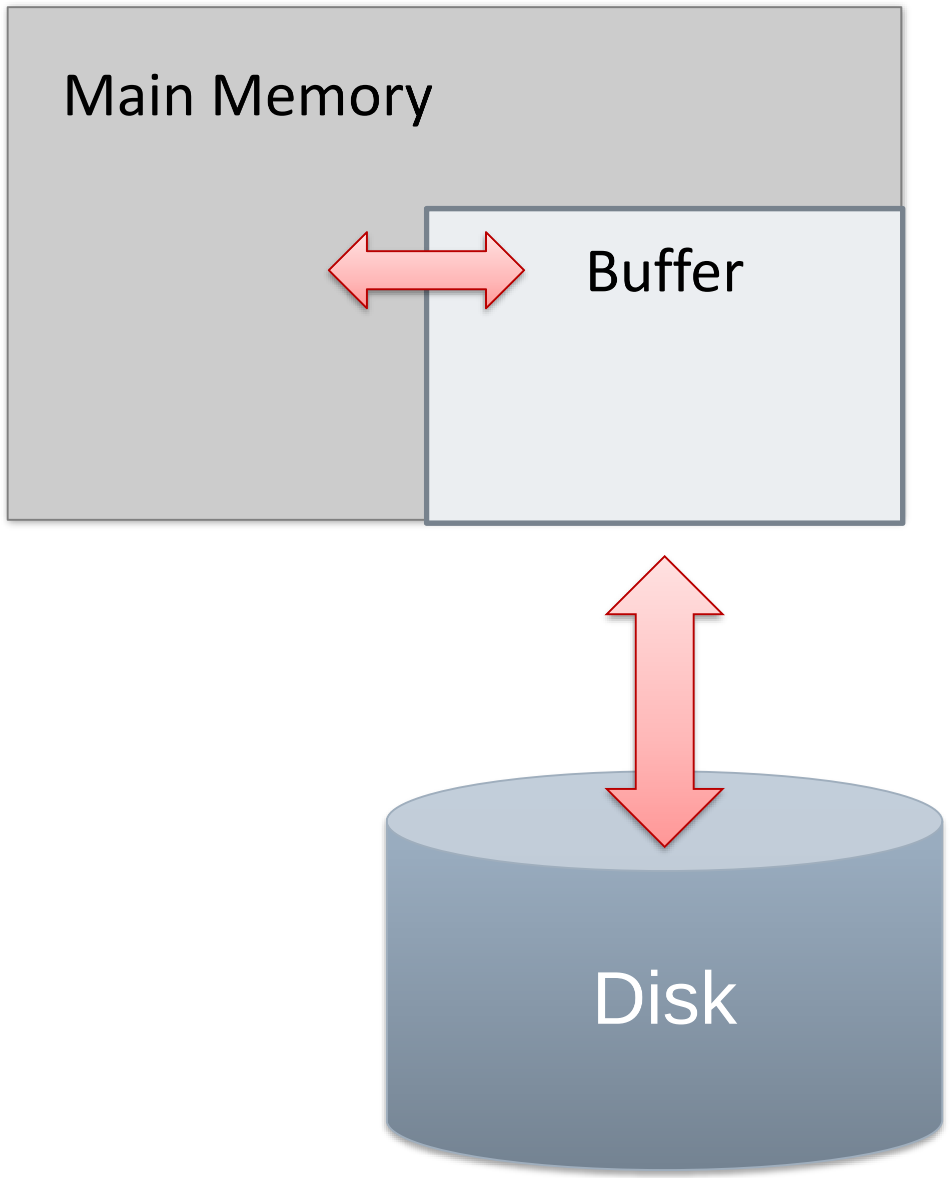
- ***Slow:* Sequential *block* access**
  - Read a blocks (not byte) at a time, so sequential access is c heaper than random
  - **Disk read / writes are expensive!**
- ***Durable:* We will assume that once on disk, data is safe!**
- ***Cheap***

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!

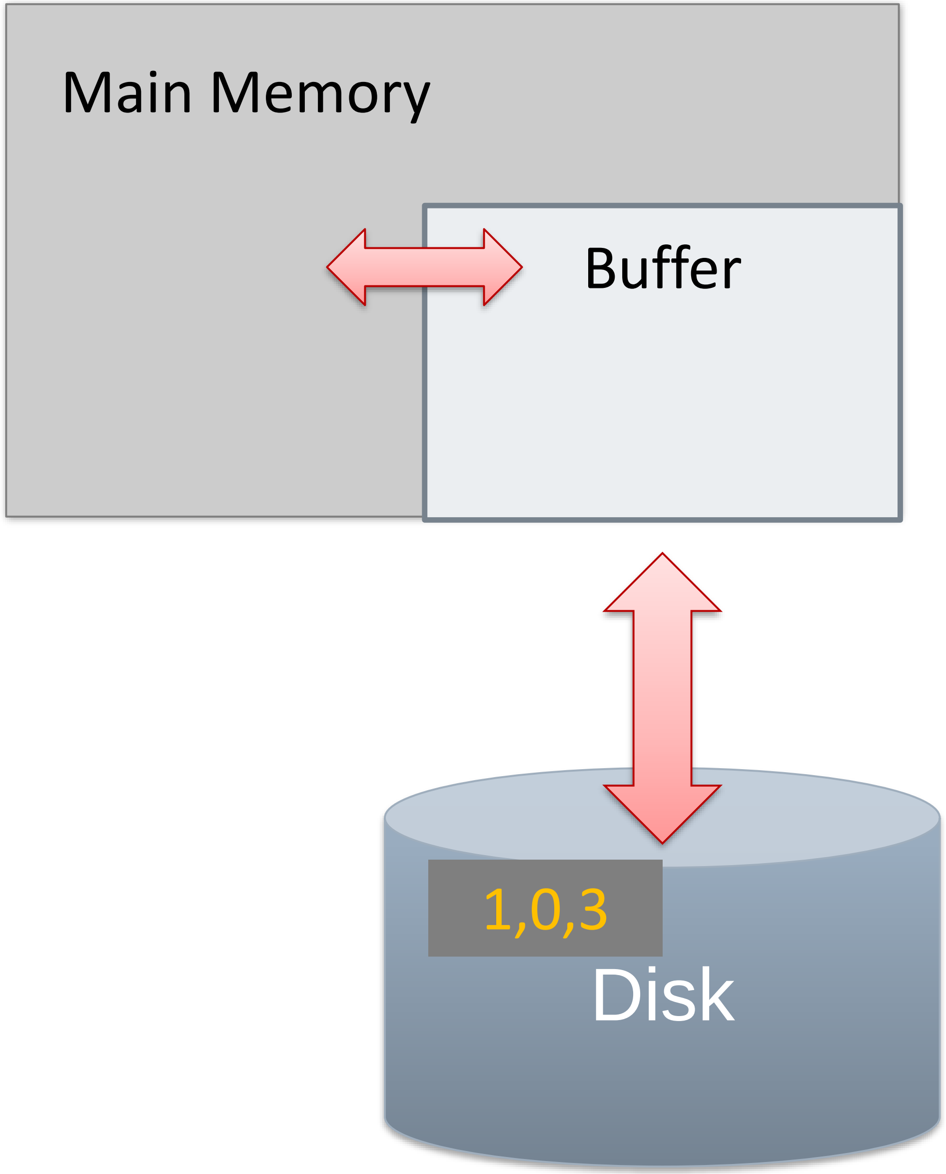
The Buffer

- A <u>buffer</u> is a region of physical memory used to store *te mporary data*
  - *In this lecture:* a region in main memor y used to store **intermediate data bet ween disk and processes**
- *Key idea:* Reading / writing to disk is slow- need to cache data!



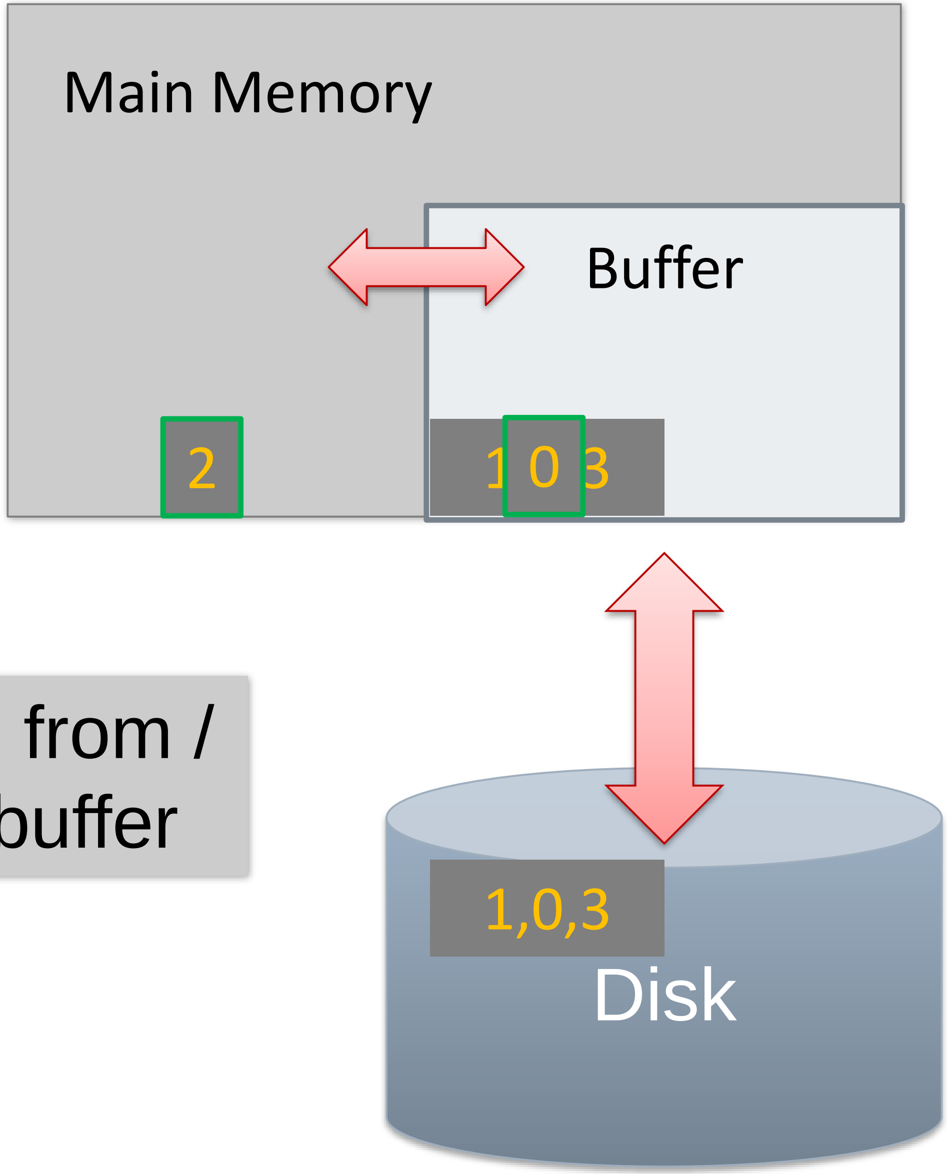
The (Simplified) Buffer

- **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

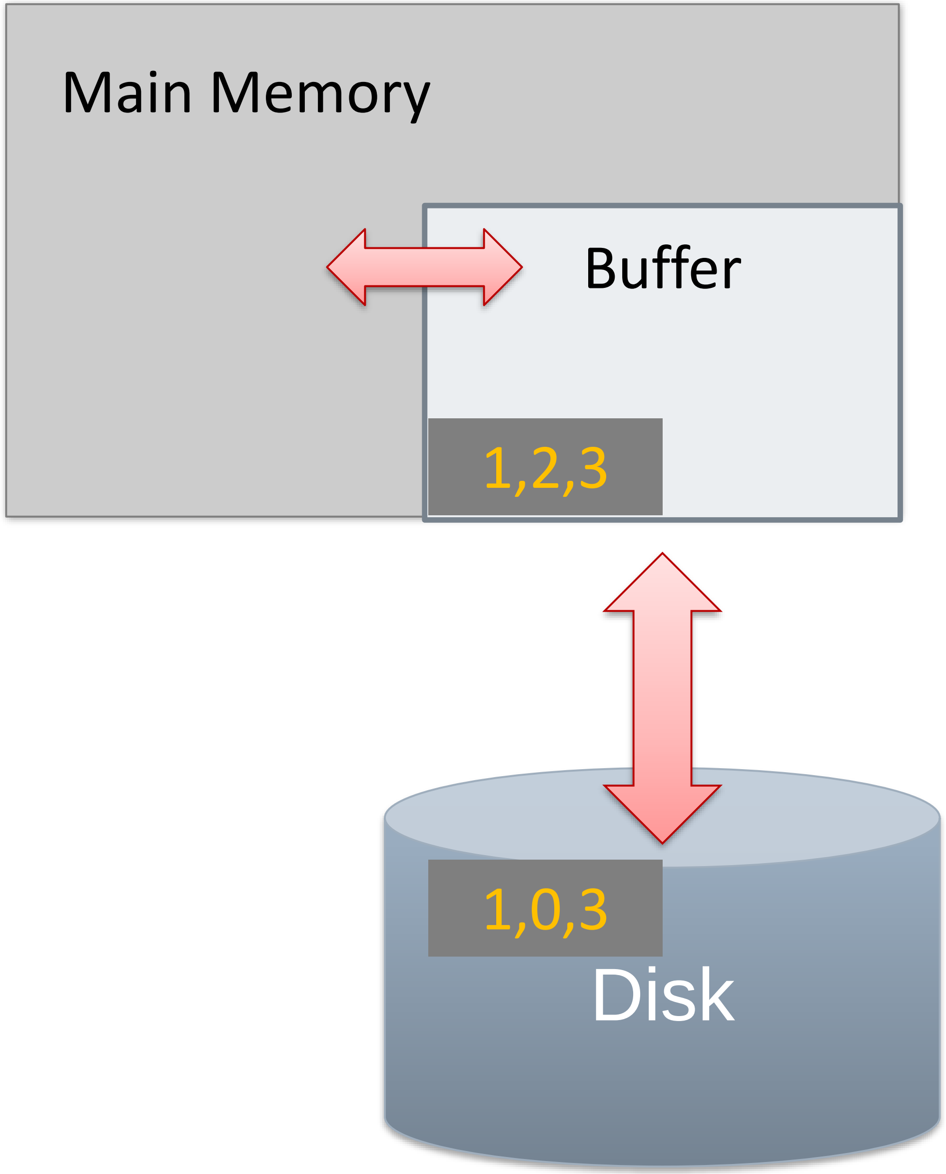
- **In this class: We'll consider a buffer located in main memory that operates over pages and files:**
  - **<u>Read(page):</u>** Read page from disk -> buffer *if not already in buffer*



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

The (Simplified) Buffer

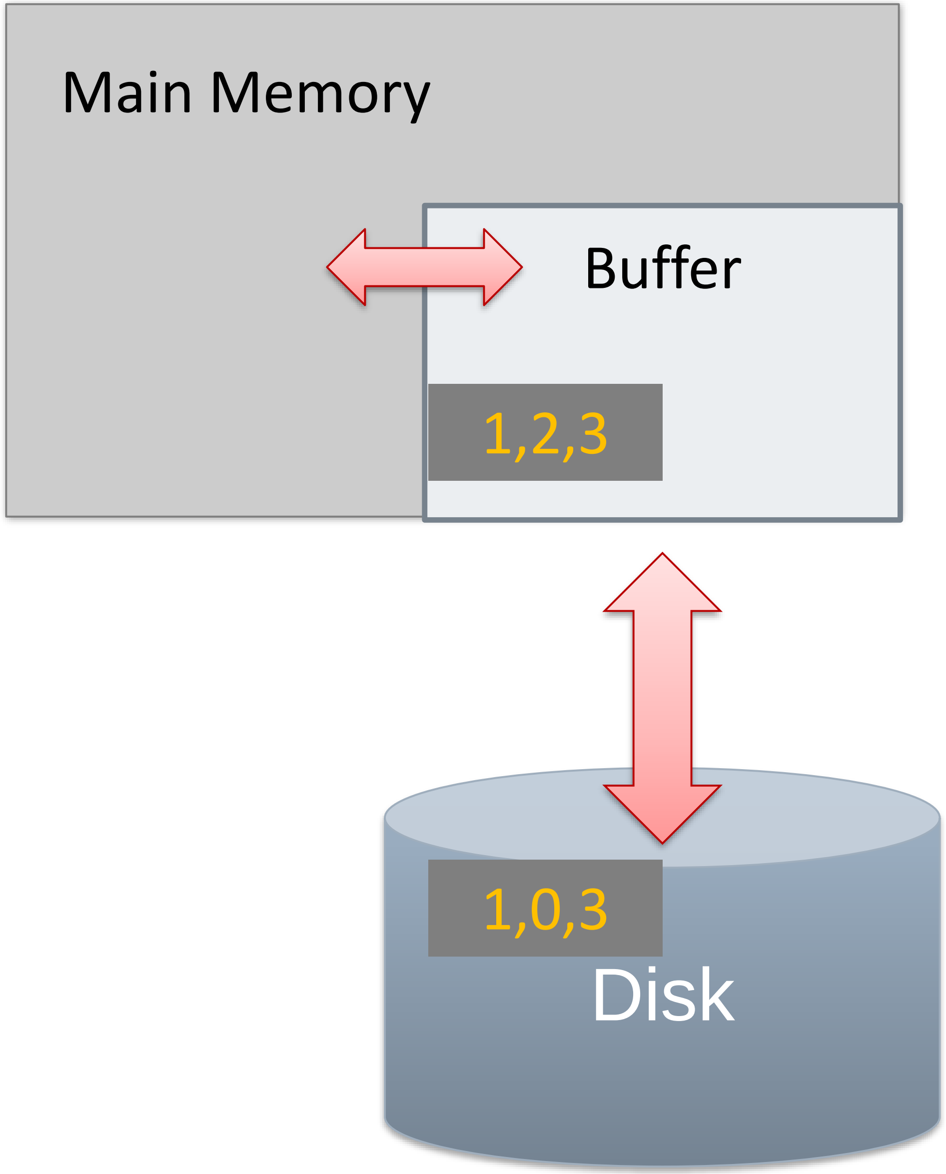
- **In this class: We'll consider a buffer located in main memory that operates over pages and files:**
  - **<u>Read(page):</u>** Read page from disk -> buffer *if not already in buffer*
  - **<u>Flush(page):</u>** Evict page from buffer & write to disk



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



Practice – Commit/Rollback

- **Make a simple table as follows.**

A	B	C
1	2	3

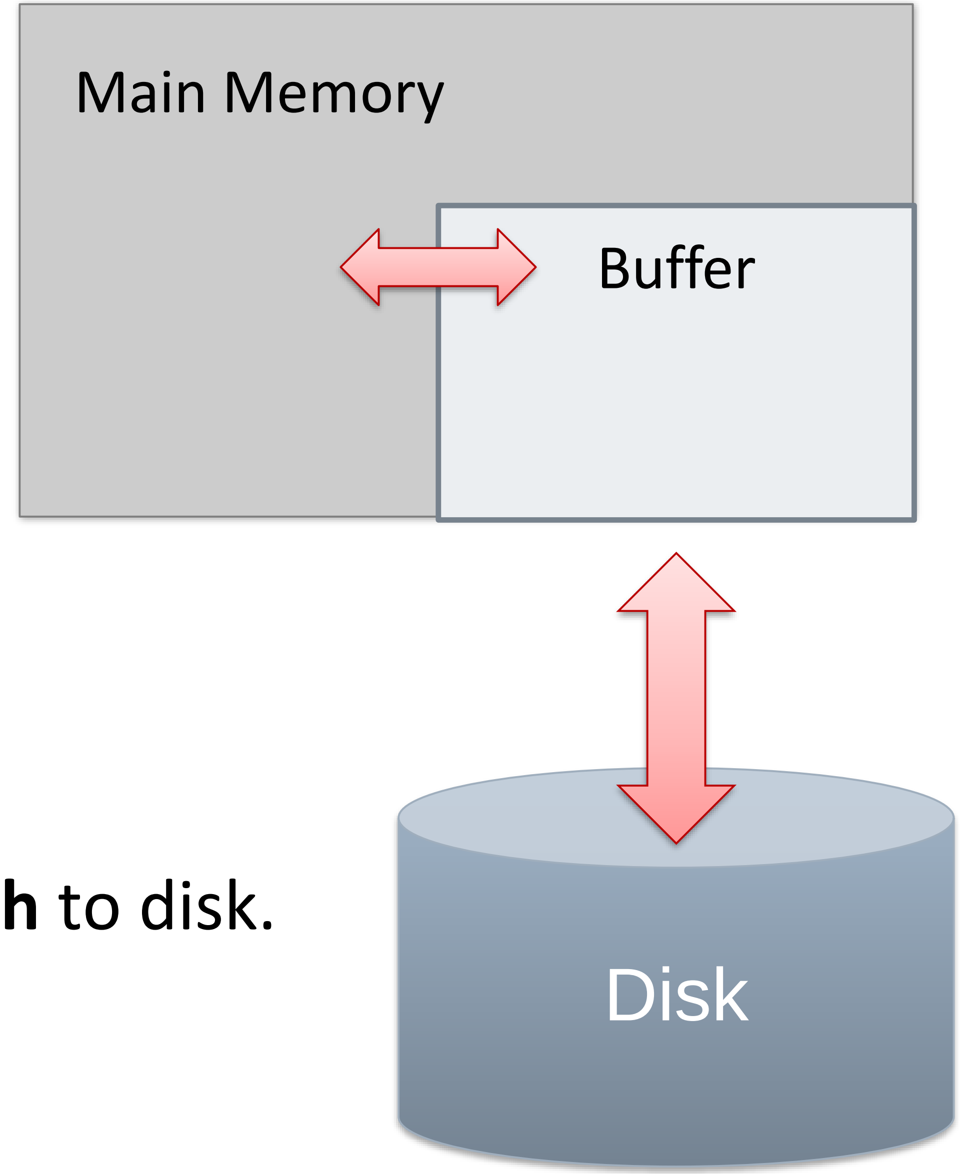
- **After make the table above, then input the following commands the compare the results**

1. commit;
2. rollback;

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.



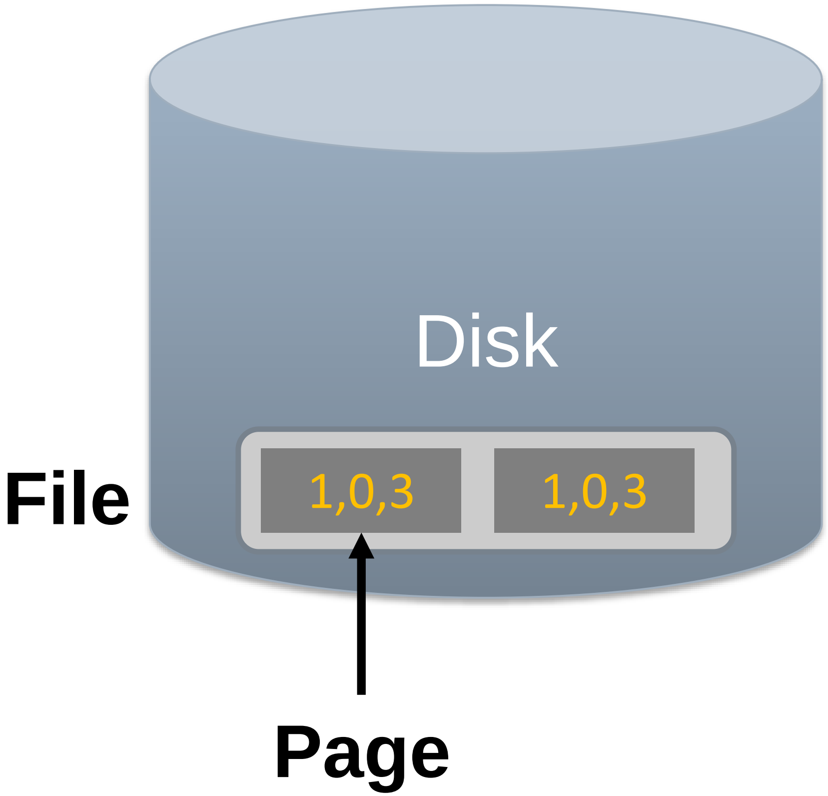
The Buffer Manager

- A <u>buffer manager</u> 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

- A page is a *fixed-sized array* of memory

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



Practice – Buffering Effects

- **Install Python 2.7**
  - Type “download python 2.7” in Google
- **Using Python Script, make a simple database having 100,000 tuples**
  - Simple schema - e.g., SampleT (ID, value1, value2)
  - Write Python script for INSERT INTO statements to insert 100,000 tuples into the database
    - ID is a unique value
    - Value1 and value2 could be random values
  - Inserting 100,000 tuples will take some times (about a couple of minutes)

Execute Python

```
c:\Python27>
c:\Python27>
c:\Python27>python
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:53:40) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
^C
c:\Python27>python
Python 2.7.13 (v2.7.13:a06454b1afa1, Dec 17 2016, 20:53:40) [MSC v.1500 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> a = 1
>>> b = 2
>>> print a+b
3
>>>
```

```
c:\Python27>type test.py
a = 1
b = 2

print a + b
c:\Python27>python test.py
3

c:\Python27>
```

Sample Python Script

```python
from random import randint

for i in range(0, 10):
    print "random value: %d" %(randint(0, 10000))
```

명령 프롬프트

```
>>> a = 1
>>> b = 2
>>> print a+b
3
>>> print a+b
^C
c:₩Python27>python test.py
3

c:₩Python27>type test.py
a = 1
b = 2

print a + b
c:₩Python27>python test.py
3

c:₩Python27>python test.py
random value: 4751
random value: 119
random value: 7958
random value: 4034
random value: 2914
random value: 4868
random value: 4134
random value: 1811
random value: 2492
random value: 2865

c:₩Python27>
```

Check the Buffering Effects of Oracle Database

- Commands to remove the buffer effect in Oracle
  - alter system flush buffer_cache;
  - alter system flush shared_pool;
- Command to check the time to execute a given SQL
  - SET TIMING ON;
- Obtain the executed time to check the buffering effects for the following query
  - select t1.value1 from SampleT t1, SampleT t2 where t1.value2 > t2.value2 group by t1.value1 having t1.value1 > 9990;

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 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 mini mum elements from each list.

If:

$$A_1 \leq A_2 \leq \cdots \leq A_N$$

$$B_1 \leq B_2 \leq \cdots \leq B_M$$

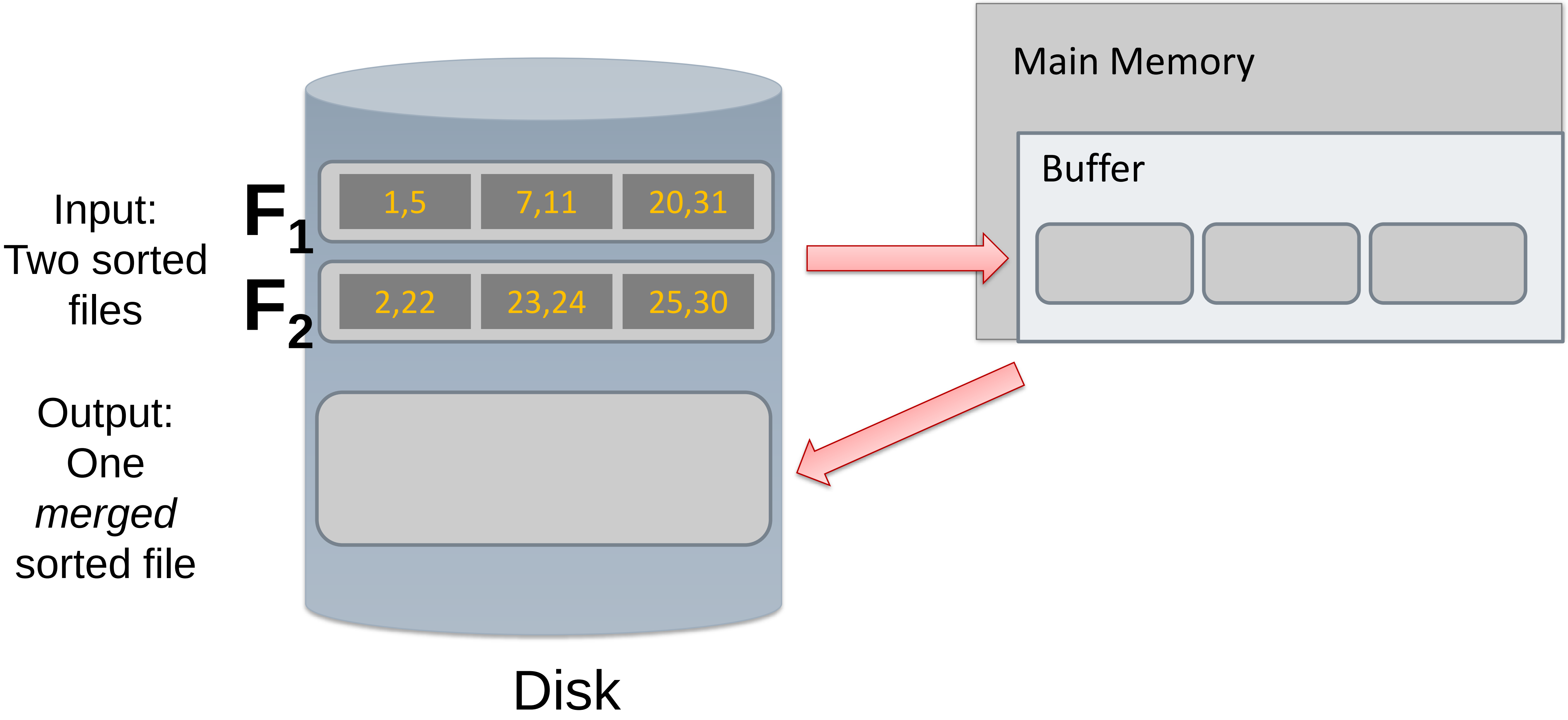
Then:

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

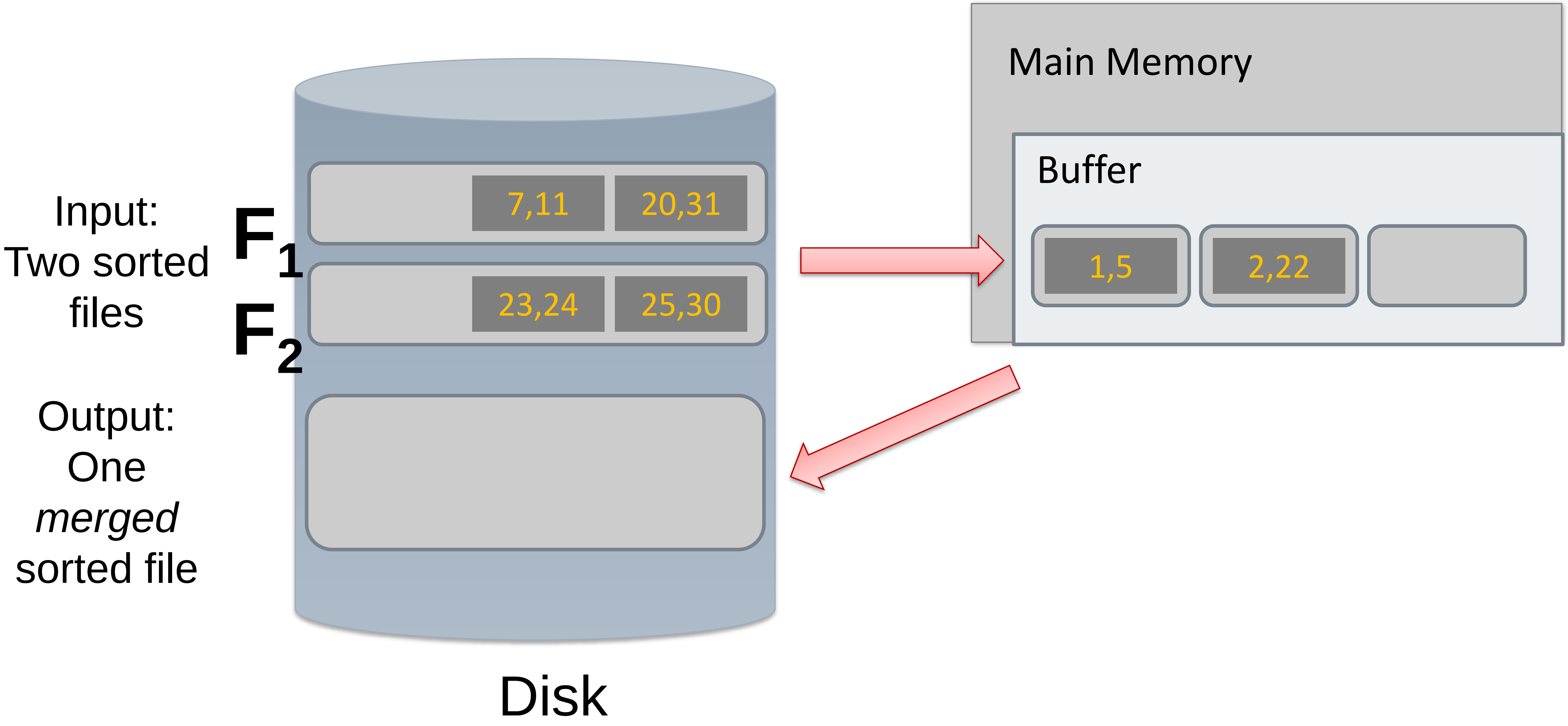
$$Min(A_1, B_1) \leq B_j$$

for i=1….N and j=1….M

External Merge Algorithm



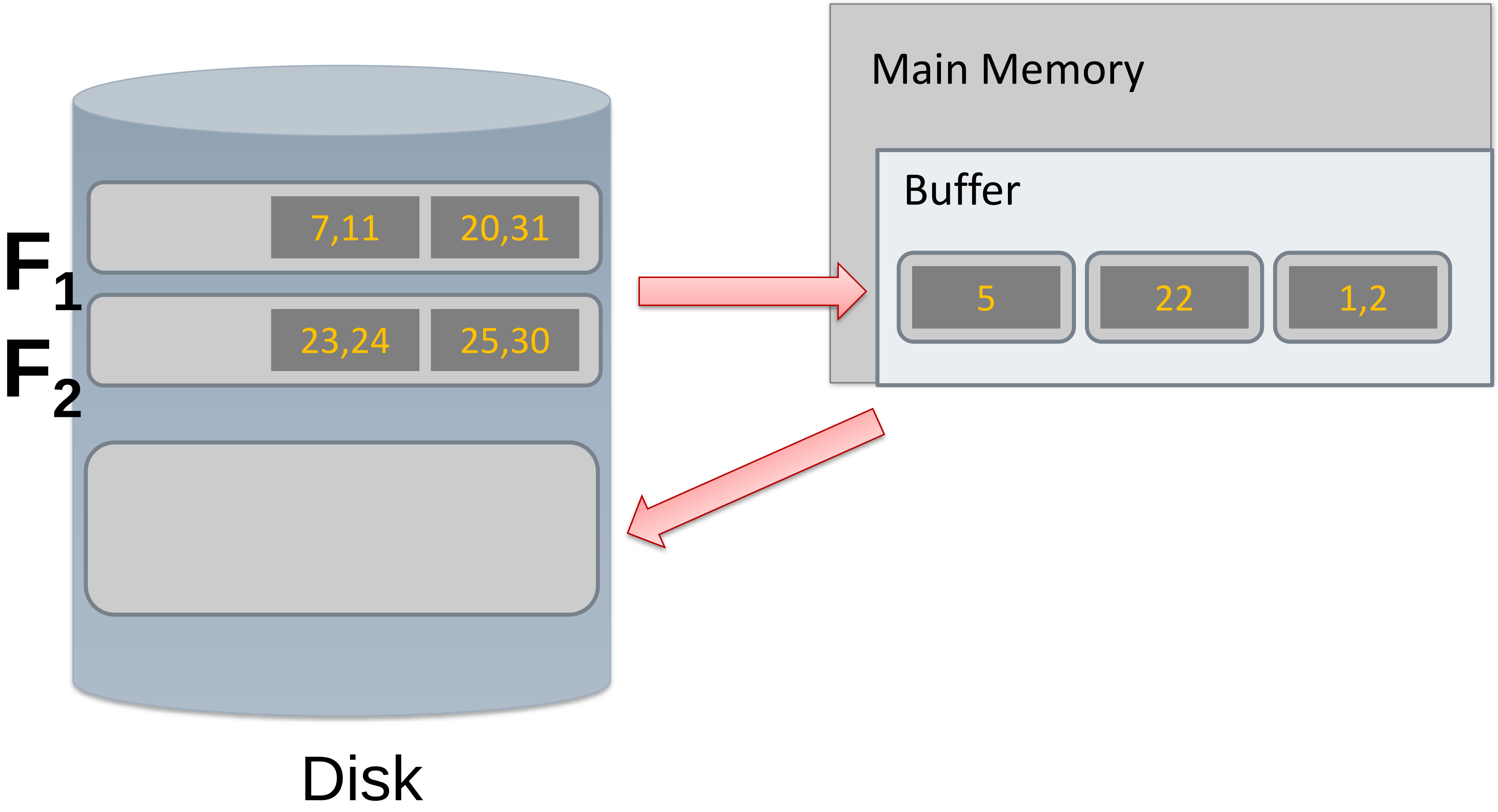
External Merge Algorithm



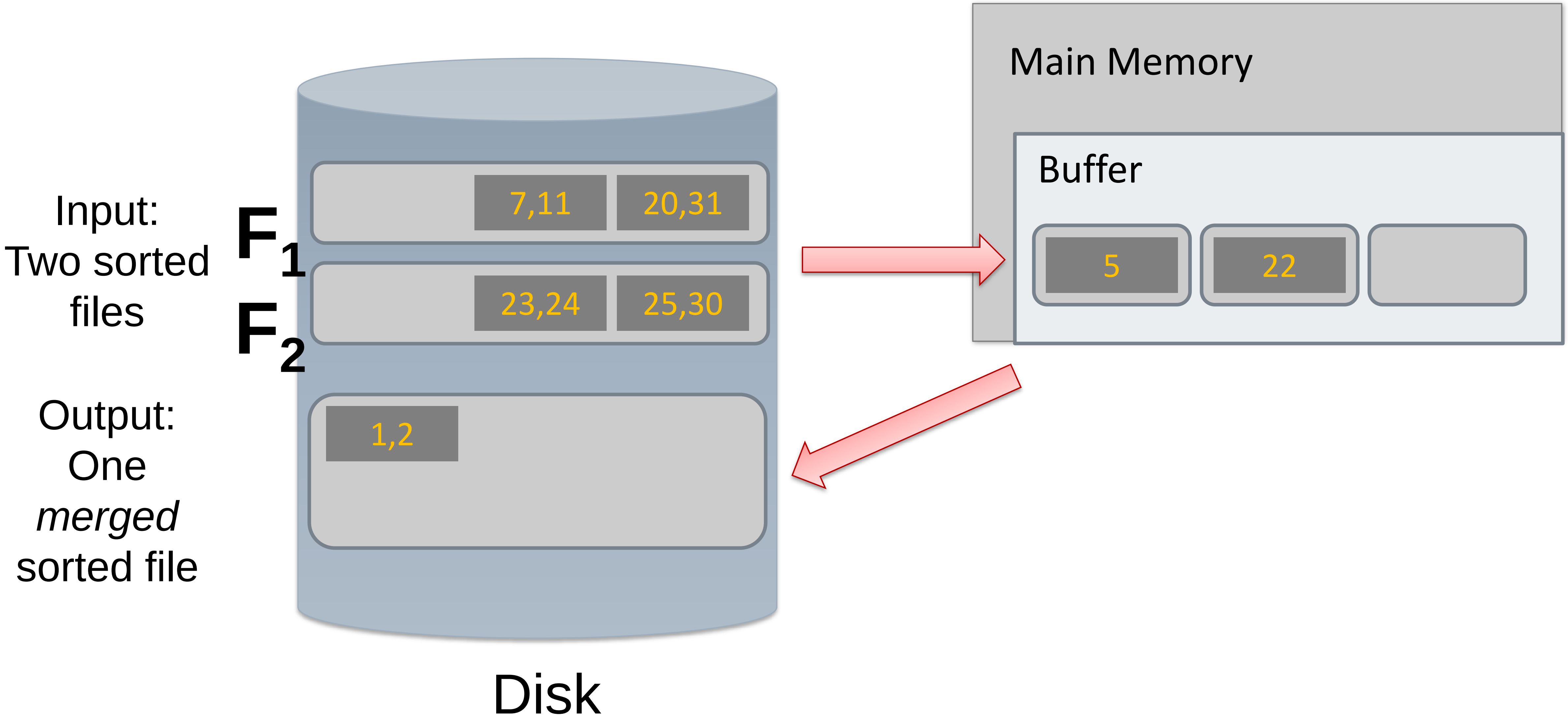
External Merge Algorithm

Input: Two sorted files

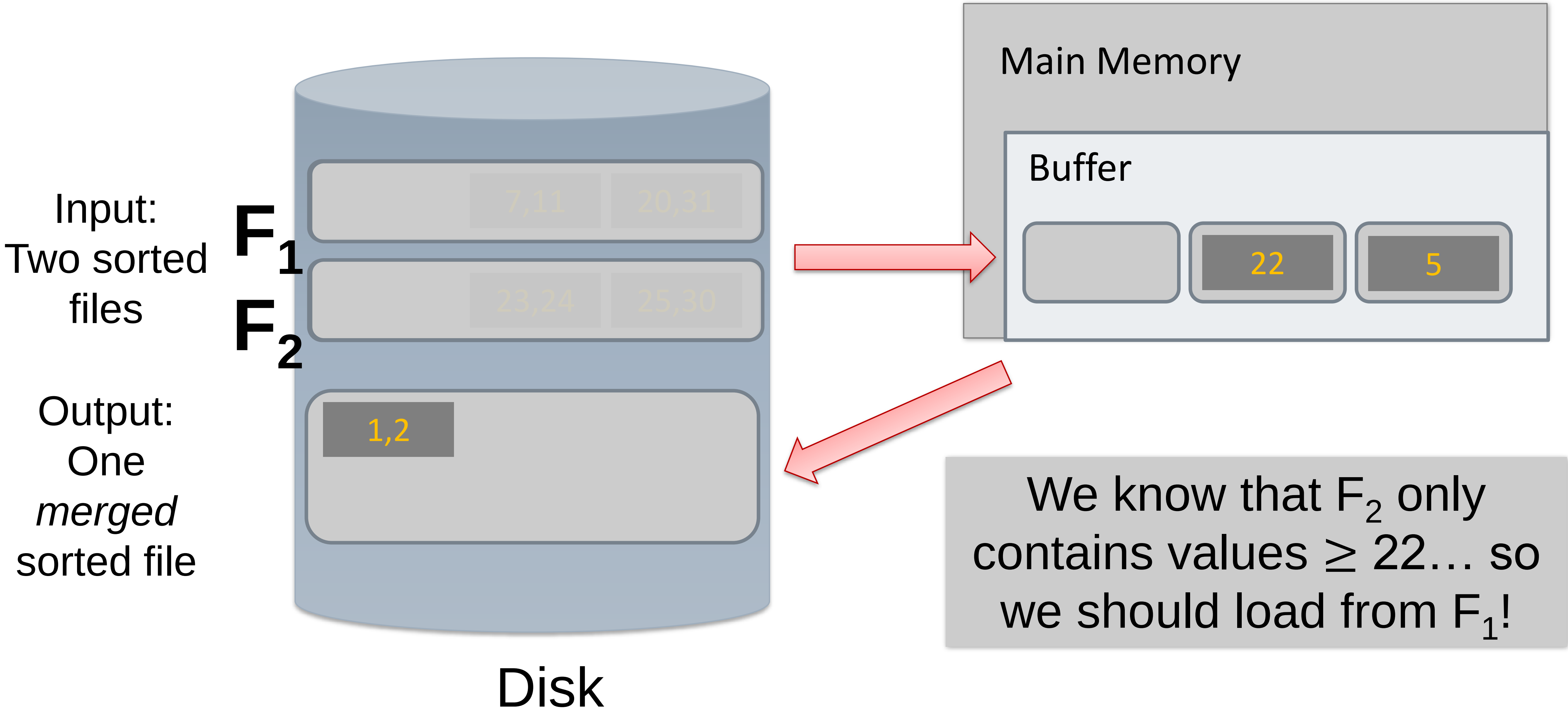
Output: One *merged* sorted file



External Merge Algorithm

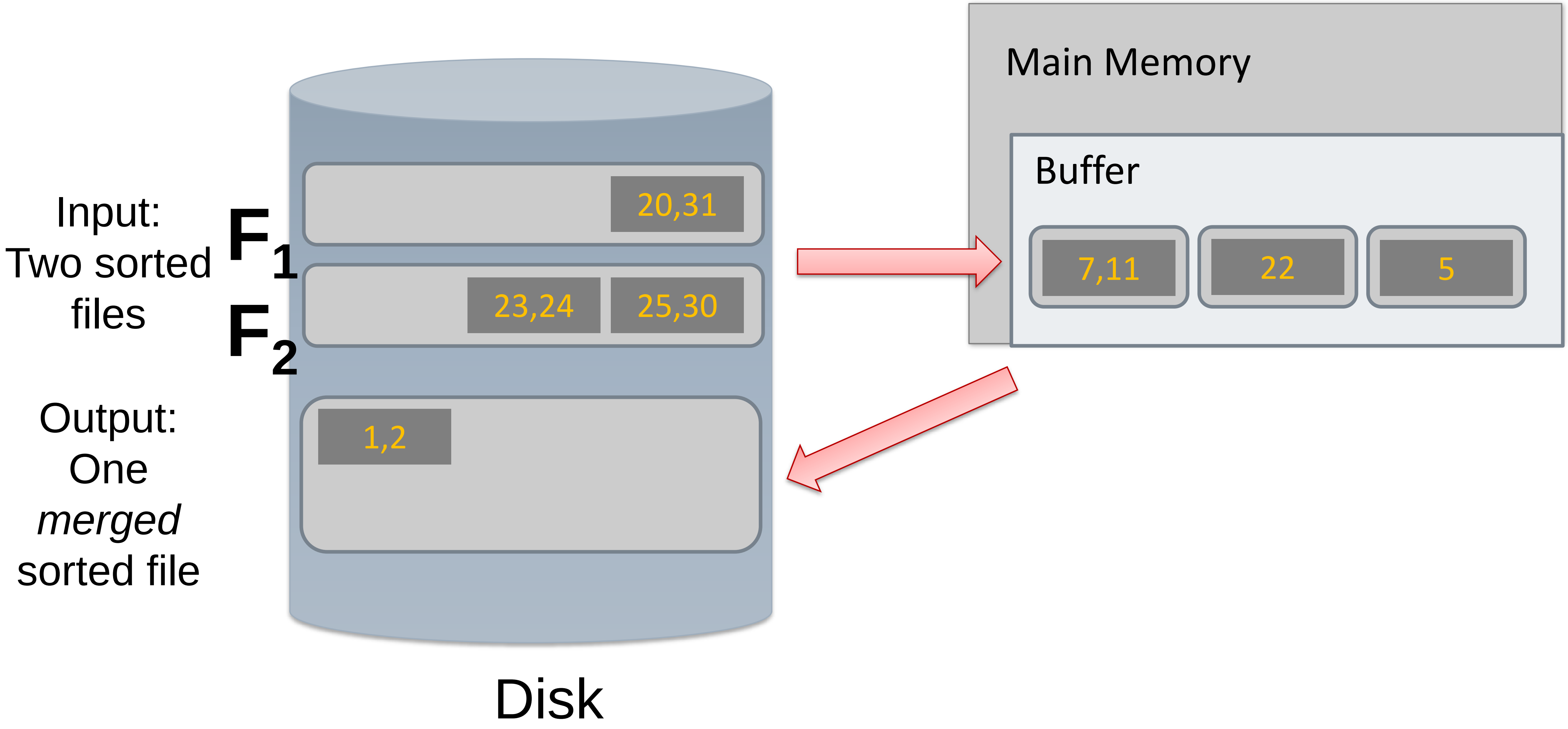


External Merge Algorithm

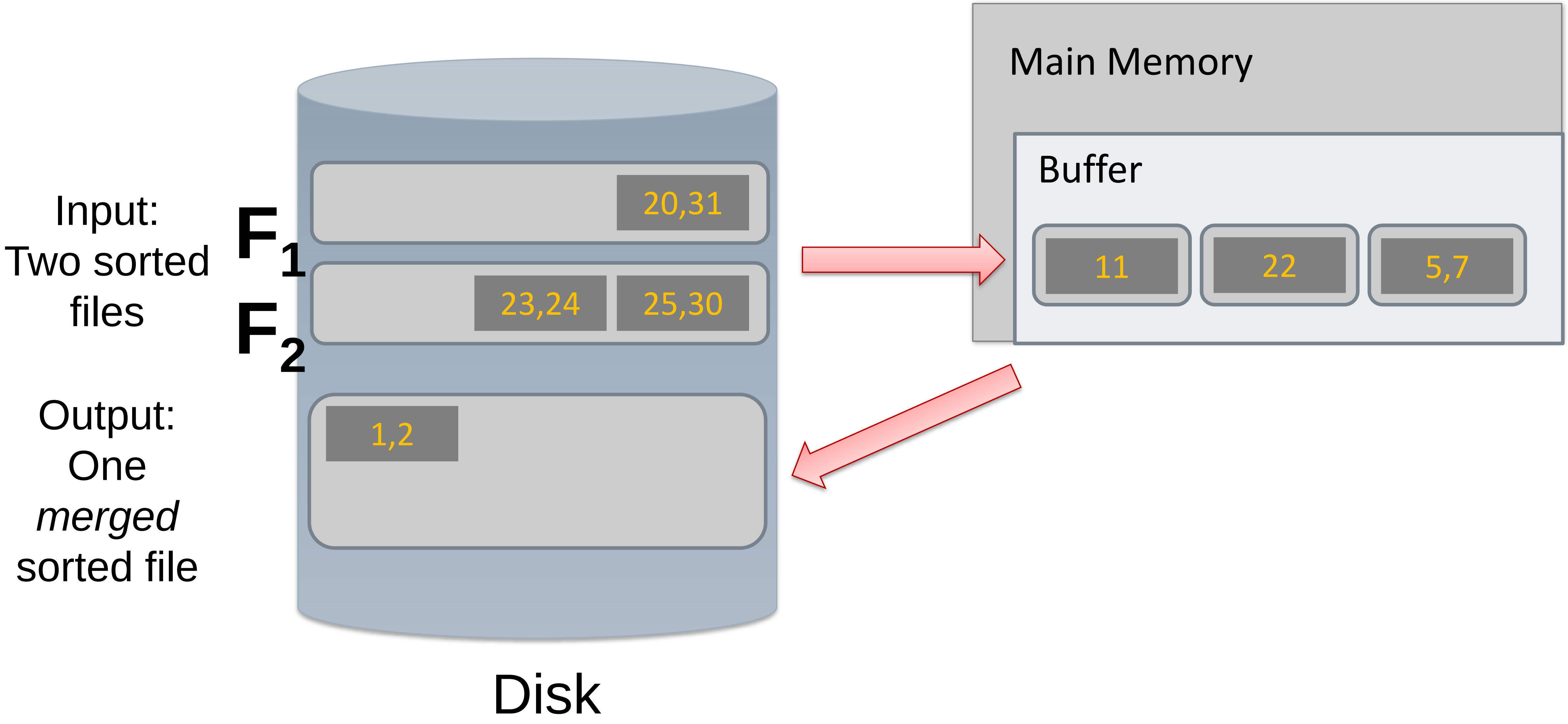


We know that F_2 only contains values $\geq$ 22… so we should load from F_1!

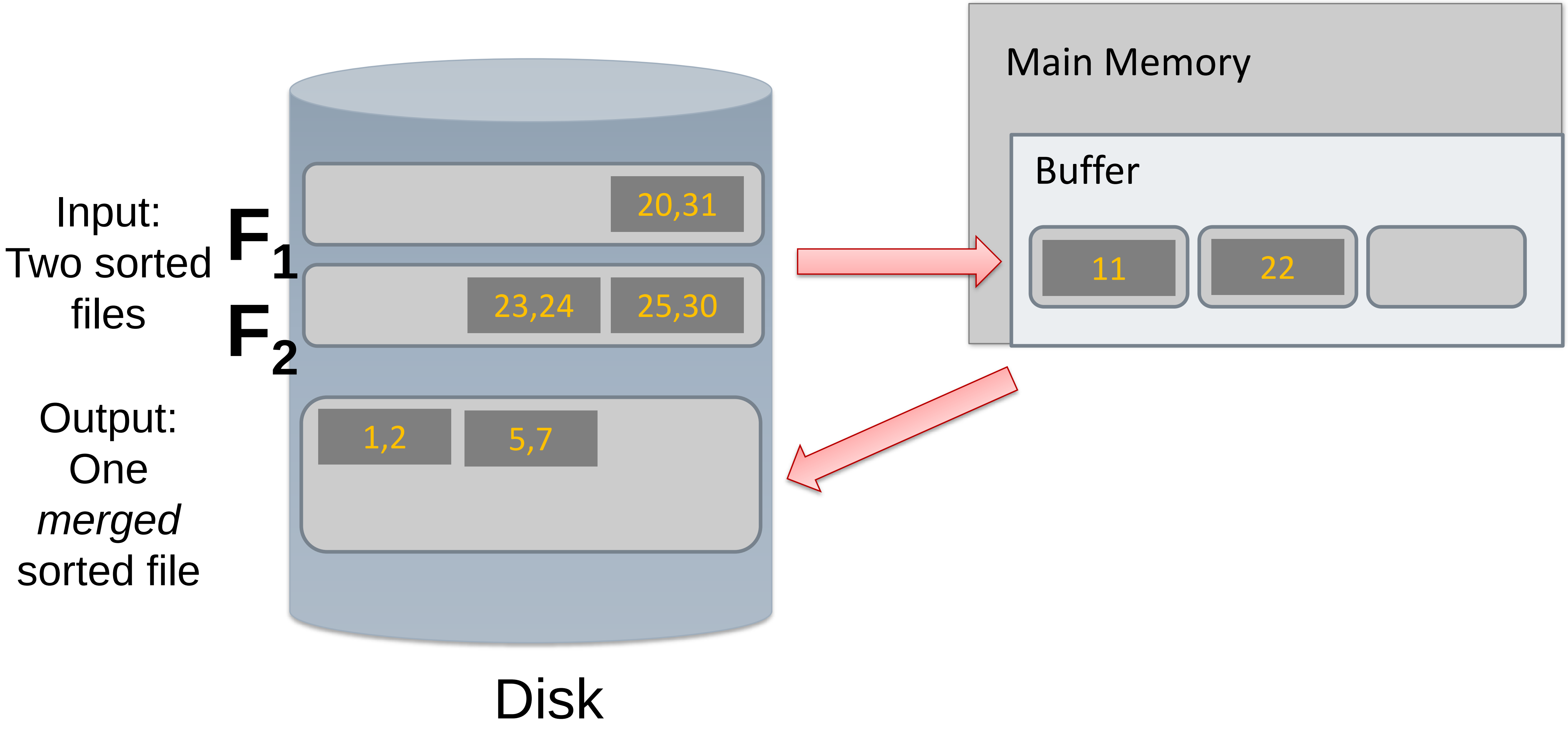
External Merge Algorithm



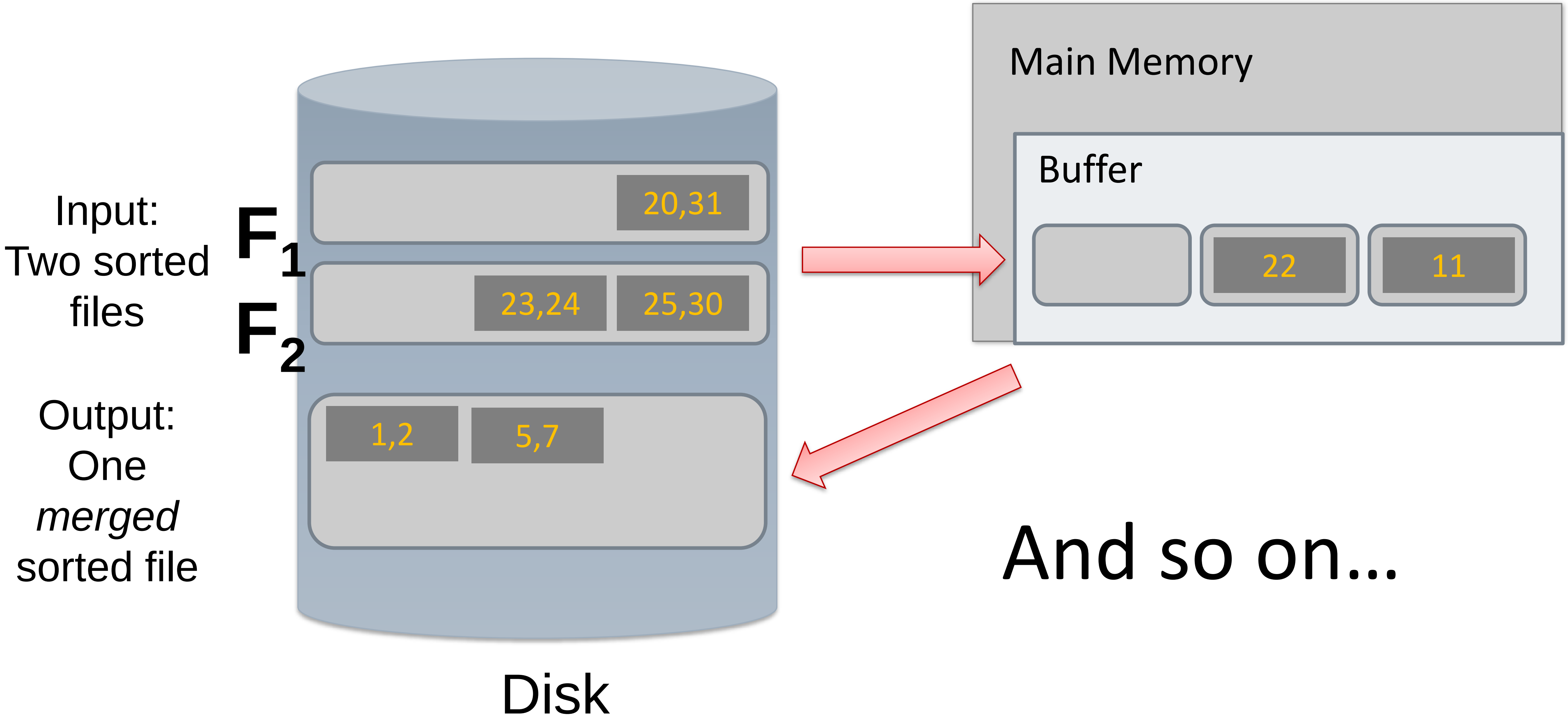
External Merge Algorithm



External Merge Algorithm



External Merge Algorithm



And so on…

External Merge Sort

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!

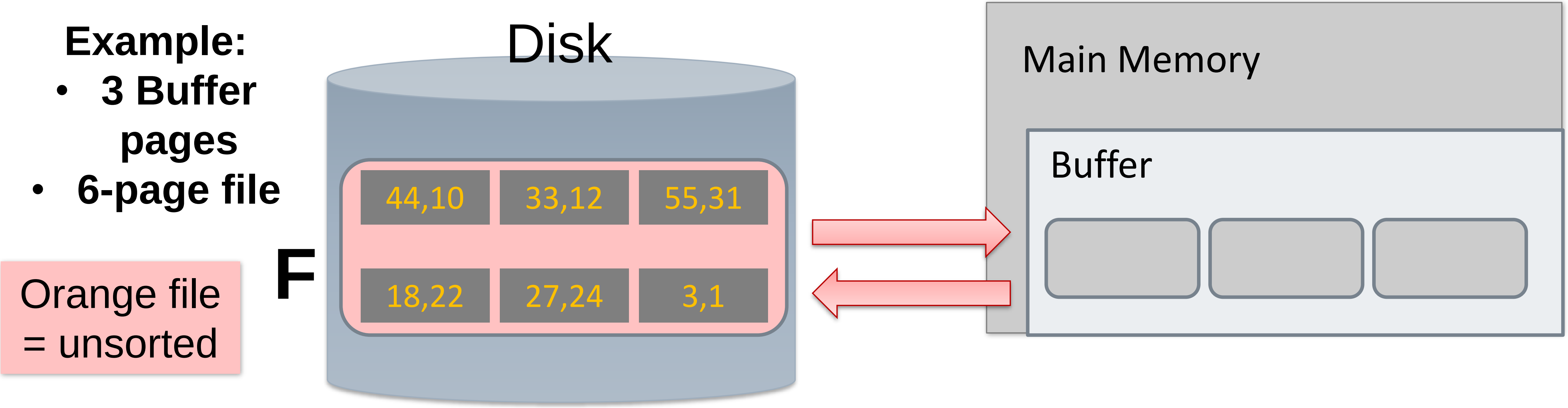
More reasons to sort…

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

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

External Merge Sort Algorithm

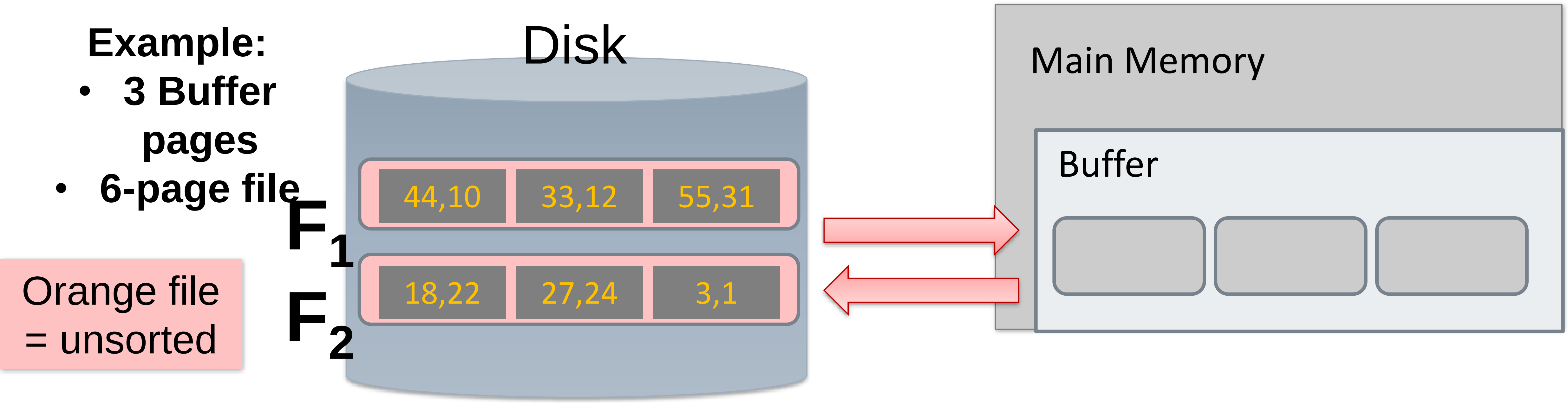
Example:
- **3 Buffer pages**
- **6-page file**



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

External Merge Sort Algorithm

Example:
- **3 Buffer pages**
- **6-page file**

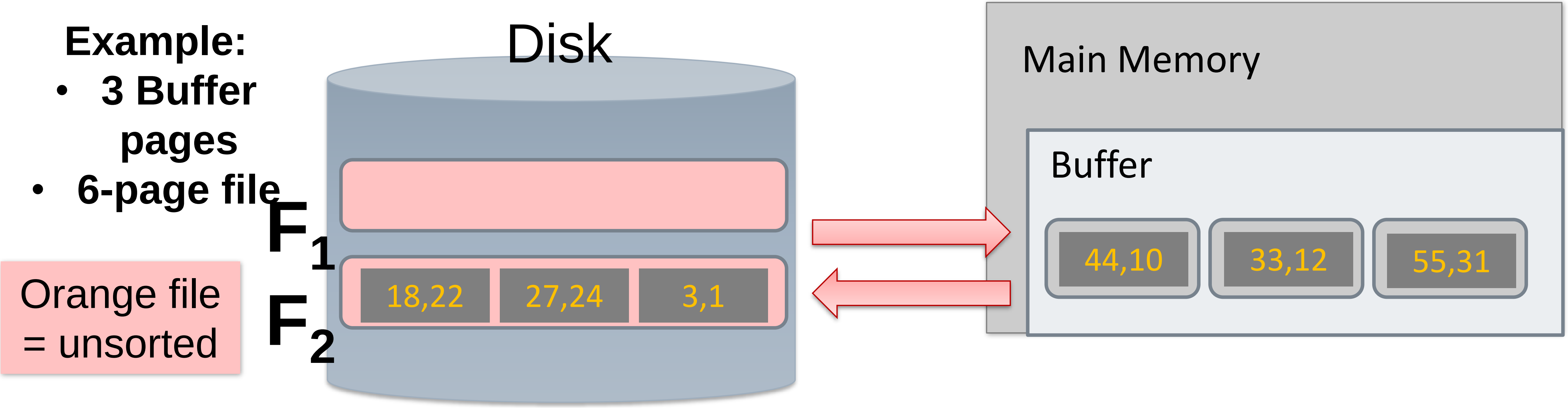


1. **Split into chunks small enough to sort in memory**

External Merge Sort Algorithm

Example:

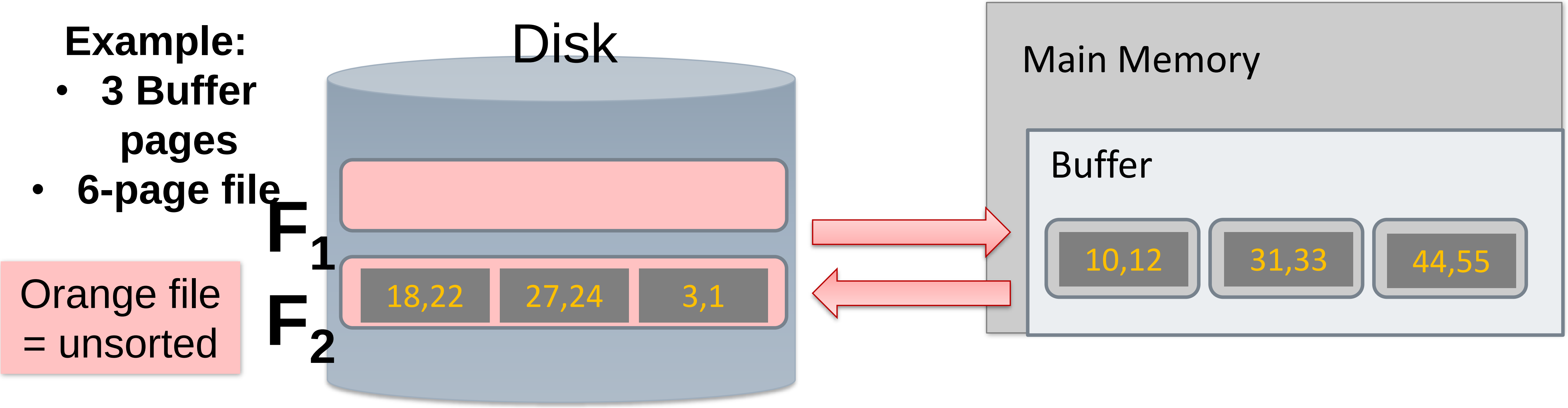
- **3 Buffer pages**
- **6-page file**



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

External Merge Sort Algorithm

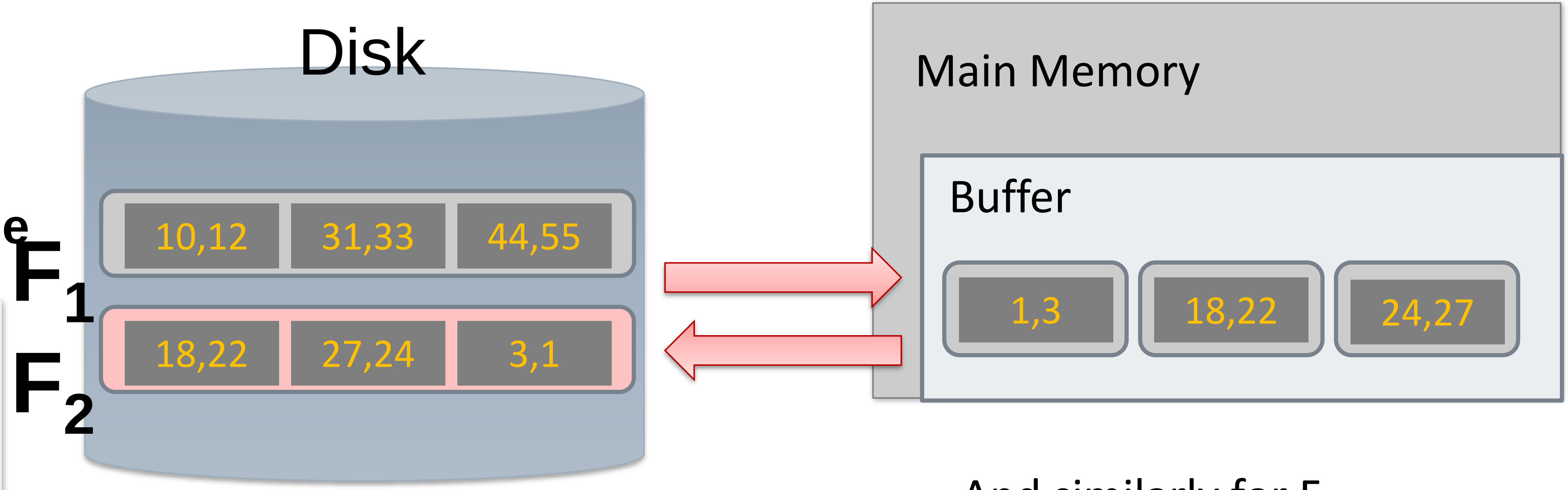
Example:
- **3 Buffer pages**
- **6-page file**



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

External Merge Sort Algorithm

Example:
- **3 Buffer pages**
- **6-page file**



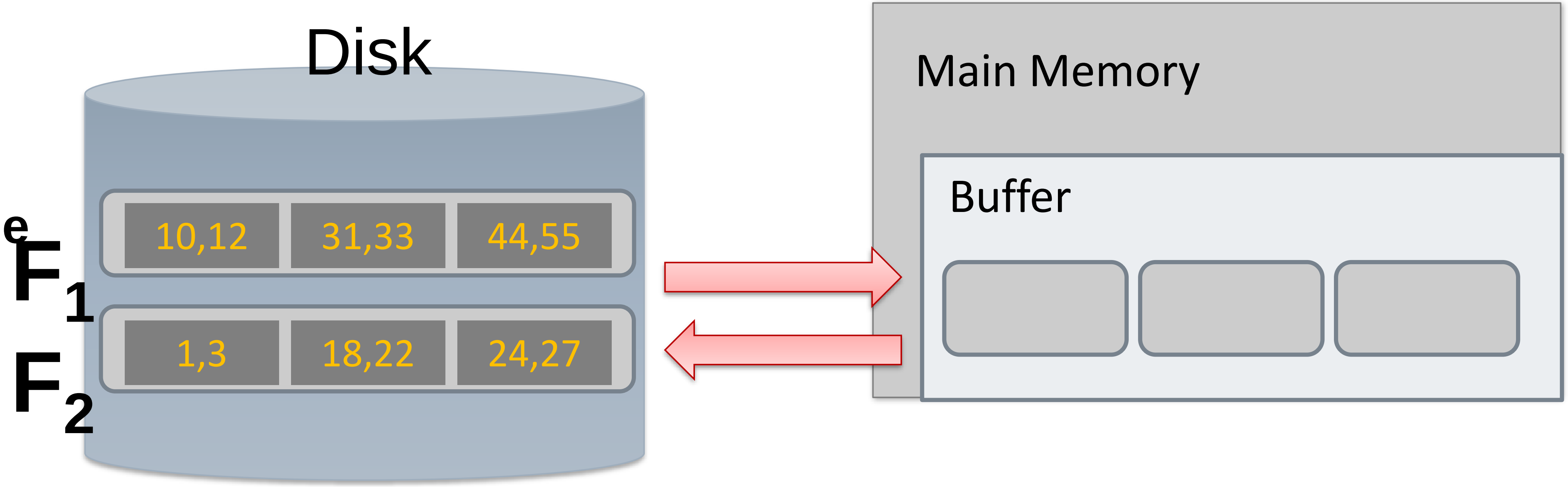
Each sorted file is a called a ***run***

And similarly for F_2

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

External Merge Sort Algorithm

Example:
- **3 Buffer pages**
- **6-page file**



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