Fundamental File Structure Concepts

Outline

- Field and record organization
- Sequential Files
- Sorted Sequential Files

Files

A file can be seen as

1. A stream of bytes (no structure), or
2. A collection of records with fields

The need for fields

Consider the function “write Book info as a stream of bytes”

```
ostream & operator << (ostream & outputFile, Book &b)
{  // insert (write) fields into stream
    outputFile << b.isbn  << b.author    << b.title;
    return outputFile;
}
```

Input:

```
87359  Carroll  Alice in wonderland
38180  Folk     File Structures
```

Output:

```
87359CarrollAlice in wonderland38180FolkFile
Structures .....
```

A Stream File

- File as a sequence of bytes:

```
87359CarrollAlice in wonderland38180FolkFile Structures ...
```

- Data semantics is lost: there is no way to get it apart again.

Field Structures

- Fixed-length fields

```
87359 Carroll   Alice in wonderland
38180 Folk      File Structures
```

- Begin each field with a length indicator

```
058735907Carroll19Alice in wonderland
053818004Folk15File Structures
```

- Place a delimiter at the end of each field

```
87359|Carroll|Alice in wonderland|
38180|Folk|File Structures|
```

- Store field as `keyword = value`

```
ISBN=87359|AU=Carroll|TI=Alice in wonderland|
ISBN=38180|AU=Folk|TI=File Structures|
```

Fixed length fields

- The size of each character array in the above example is fixed. If the value of the data requires less space than reserved, then characters representing spaces (‘ ‘) are added to fill the array.
- For example a string such as “Carroll” would have to be padded with 8 blanks (and terminated with \0) to fill the array:

  char author[15]

Separating Fields with Length Indicators

- This method requires that each field data be preceded with an indicator of its length (in bytes).

```
058735907Carroll19Alice in wonderland
053818004Folk15File Structures
```

- One of the disadvantages of this method is that it is more complex since it requires extracting of numbers and strings from a single string representing a record.

Separating Fields with Delimiters

- This method requires that the fields be separated by a selected special character or a sequence of characters called a *delimiter*.

```
87359|Carroll|Alice in wonderland|
38180|Folk|File Structures|
```

- The method of separating fields with a delimiter is often used. However choosing a **right** delimiter is very important.

Type	Advantages	Disadvantages
Fixed	Easy to read/write	Waste space with padding
Length-based	Easy to jump ahead to the end of the field	Long fields require more than 1 byte
Delimited	May waste less space than with length-based	Have to check every byte of field against the delimiter
Keyword	Fields are self describing. Allows for missing fields	Waste space with keywords

Record Structures

Files may be viewed as collections of records which are sets of fields.

Two approaches to implement records:

1. Fixed-length records.
2. Variable-length records.

Fixed-length records

Two ways of making fixed-length records:

1. Fixed-length records with *fixed-length fields.*

87359	Carroll	Alice in wonderland
03818	Folk	File Structures

2. Fixed-length records with *variable-length fields.*

87359|Carroll|Alice in wonderland|	*unused*
38180|Folk|File Structures|	*unused*

Variable-length records

1. Fixed number of fields:

```
87359|Carroll|Alice in wonderland|38180|Folk|File Structures| ...
```

2. Record beginning with length indicator:

```
3387359|Carroll|Alice in wonderland|2638180|Folk|File Structures| ..
```

3. Placing a delimiter: e.g. end-of-line char

4. Use an index file to keep track of record addresses:

– The index file keeps the byte offset for each record; this allows us to search the index (which have fixed length records) in order to discover the beginning of the record.

Question

- What is the byte offset or just offset of a file?

Type	Advantages	Disadvantages
Fixed length record	Easy to jump to the i-th record	Waste space with padding
Variable-length record	Saves space when record sizes are diverse	Cannot jump to the i-th record, unless through an index file

File Organization

Four basic types of organization:

1. Sequential
2. Indexed
3. Indexed Sequential
4. Hashed

File Operations

- Typical Operations:
  - Retrieve a record
  - Insert a record
  - Delete a record
  - Modify a field of a record

Sequential files

- Records are stored ***contiguously*** on the storage device.
- Sequential files are read from beginning to end.
- Some operations are very efficient on sequential files (e.g. finding averages)
- Organization of records:

1. Unordered sequential files (**pile files**)
2. **Sorted** sequential files (records are ordered by some field)

Pile Files : Unordered Sequential Files

- A **pile file** is a succession of records, simply placed one after the other with no additional structure.
- Records may vary in length.
- Typical Request:
  - Print all records with a given field value
    - e.g. print all books by *Folk*.
  - We must examine each record in the file, in order, starting from the first record.

Searching Pile Files

```
SELECT *
FROM Students S
WHERE S.sid = 1234;
```

- To look-up a record, given the value of one or more of its fields, we must search the whole file.
- In general, (***b*** is the total <u>number of blocks</u> in file):
  - At least 1 block is accessed
  - At most *b* blocks are accessed.
  - On average $1/b * b (b + 1) / 2 \Rightarrow b/2$
- Thus, time to find and read a record in a pile file is approximately : $\mathbf{T_F = (b/2) * btt}$

Time to fetch one record

Exhaustive Reading of the File

- Read and process all records

(reading order is not important)

$$T_X = b * \mathbf{btt}$$

(approximately twice the time to fetch one record)

- e.g. Finding averages, min or max, or sum.
  - Pile file is the best organization for this kind of operations.

```
SELECT MAX(S.gpa)
FROM Students S;
```

Inserting a new record

- Just place the new record at the end of the file (assuming that we don’t worry about duplicates).
  - Read the last block
  - Put the record at the end.

=> Q) What if the last block is full?

Updating a record

- For fixed length records:

  $$\mathbf{T_U\ (fixed\ length) \approx T_F}$$

- For variable length records update is treated as a combination of delete and insert.

$$\mathbf{T_U\ (variable\ length) = T_D + T_I}$$

Deleting Records

- Operations like create a file, add records to a file and modify a record can be performed physically by using basic file operations (open, seek, write, etc.)
- What happens if records are deleted? There is no basic operation that allows us to “remove part of a file”.
- Record deletion should be taken care by the program responsible for file organization.

Strategies for record deletion

1. **Record deletion and Storage compaction:**
   - Deletion can be done by "marking" a record as deleted.
     - e.g. Place '*' (tombstone) at the beginning of the record
   - The space for the record is not released, but the program must include logic that checks if record is deleted or not.
   - After a lot of records have been deleted, a special program is used to squeeze the file – this is called Storage Compaction.

Strategies for record deletion (cont.)

2. **Deleting fixed-length records and reclaiming space dynamically.**

- How to use the space of deleted records for storing records that are added later?
  - Use an "AVAIL LIST", a linked list of available records.
  - A header record (at the beginning of the file) stores the beginning of the AVAIL LIST
  - When a record is deleted, it is marked as deleted and inserted into AVAIL LIST

Strategies for record deletion (cont.)

3. **Deleting variable length records**
   - Use AVAIL LIST as before, but take care of the variable-length difficulties.
   - The records in AVAIL LIST must store its size as a field. Exact byte offset must be used.
   - Addition of records must find a large enough record in AVAIL LIST

Placement Strategies

There are several placement strategies for selecting a record in AVAIL LIST when adding a new record:

1. First-fit Strategy
   – AVAIL LIST is not sorted by size; first record large enough is used for the new record.
2. Best-fit Strategy
   – List is sorted by size. Smallest record large enough is used.
3. Worst-fit strategy
   – List is sorted by decreasing order of size; largest record is used; unused space is placed in AVAIL LIST again.

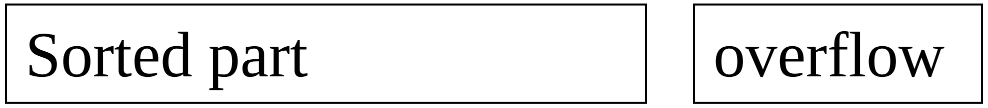
Sorted Sequential Files

- Sorted files are usually read sequentially to produce lists, such as mailing lists, invoices.etc.
- A sorted file cannot stay in order after additions (usually it is used as a temporary file).
- A sorted file will have an overflow area of added records. Overflow area is not sorted.
- To find a record:
  - First look at sorted area
  - Then search overflow area
  - If there are too many overflows, the access time degenerates to that of a sequential file.

Searching for a record

```
SELECT *
FROM Students S
WHERE S.sid = 1234;
```

- We can do binary search (assuming fixed-length records) in the sorted part.



x blocks y blocks $(x + y = b)$

- Average case to fetch a record in the ***sorted area***:

$$T_F = \log_2 x * (s + r + btt).$$

- If the record is ***not found in the sorted part***, search the ***overflow area*** too. Thus, total time is:

$$T_F = \log_2 x * (s + r + btt) + s + r + (y/2) * btt$$

	T_F	T_X	T_I	T_D	T_u
Pile File					
Sorted Sequential File					

Problem 1

- Given the following:
  - Block size = 2400 bytes
  - File size = 40M
  - Block transfer time (btt) = 0.84ms
  - s = 16ms
  - r = 8.3 ms

Q1) Calculate T_F for a certain record

a) in a pile file

b) in a sorted file (no overflow area)

Q2) Calculate the time to look up 10000 names.

Solution

40 M file => b = 16,667

Q1 a) T_F for pile file = b/2 * 0.84 ≈ 7 sec.

b) T_F for sorted file = log b * (s +r+btt)

log b = 14

=> T_F for sorted file = 14 * (16+8.3+0.84) = 352 ms

Q2) 10000 names:

Pile file => 19 hrs.

Sorted file => 59 min.

Problem 2

- Given two sorted files A and B with number of records n=100,000 and record size R=400 bytes each, we want to create an intersection file. Assume that 70% of the records are in common and the available memory for this operation is 10M.
- Calculate a timing estimate for deriving and writing the intersection file (Use s = 16 ms, r = 8.3ms, btt = 0.84ms, B=2400bytes.)

Problem 2

sorted files A and B

n=100,000

R=400 bytes

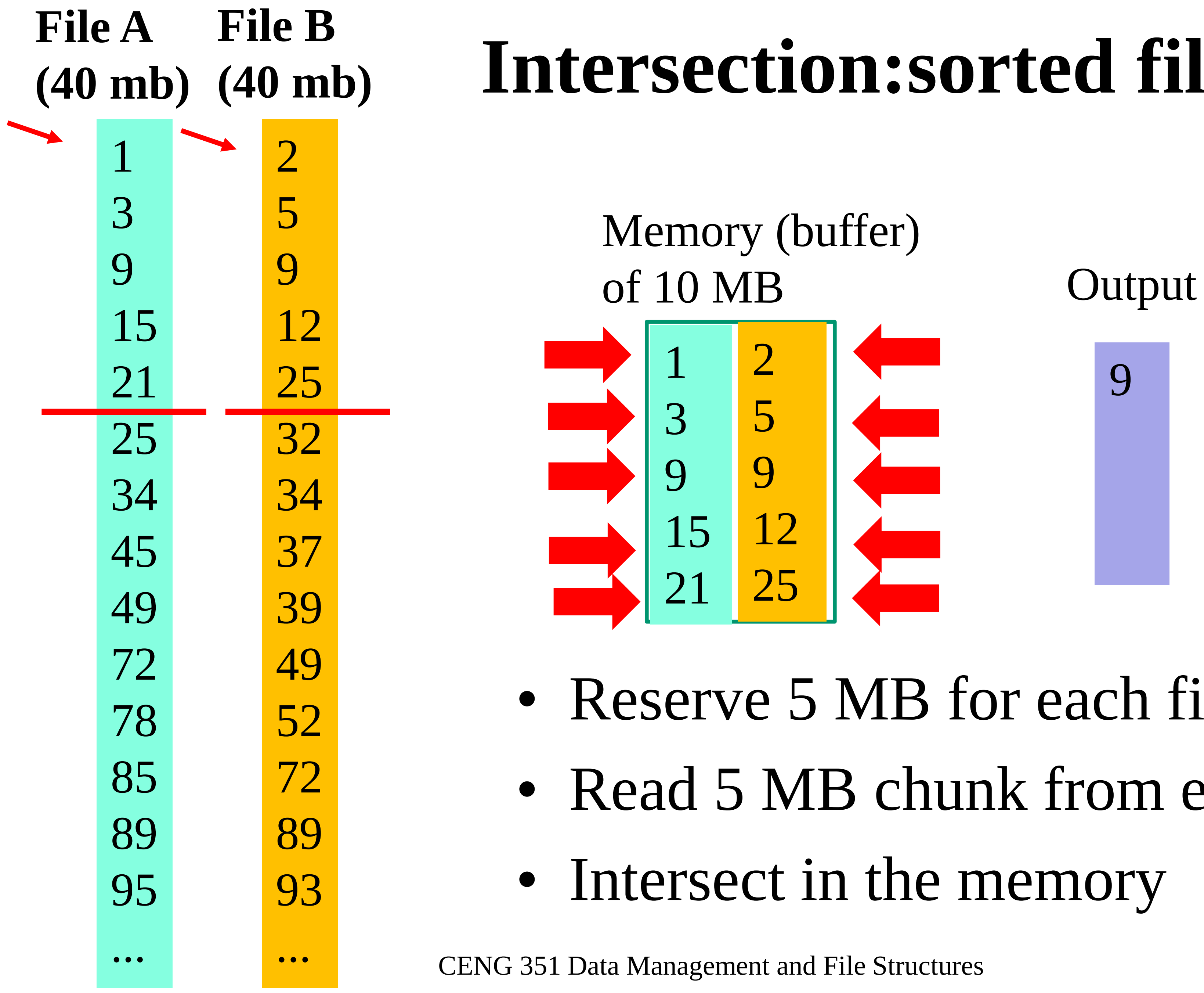
70% in common

Buffer size 10M.

s = 16 ms, r = 8.3ms,

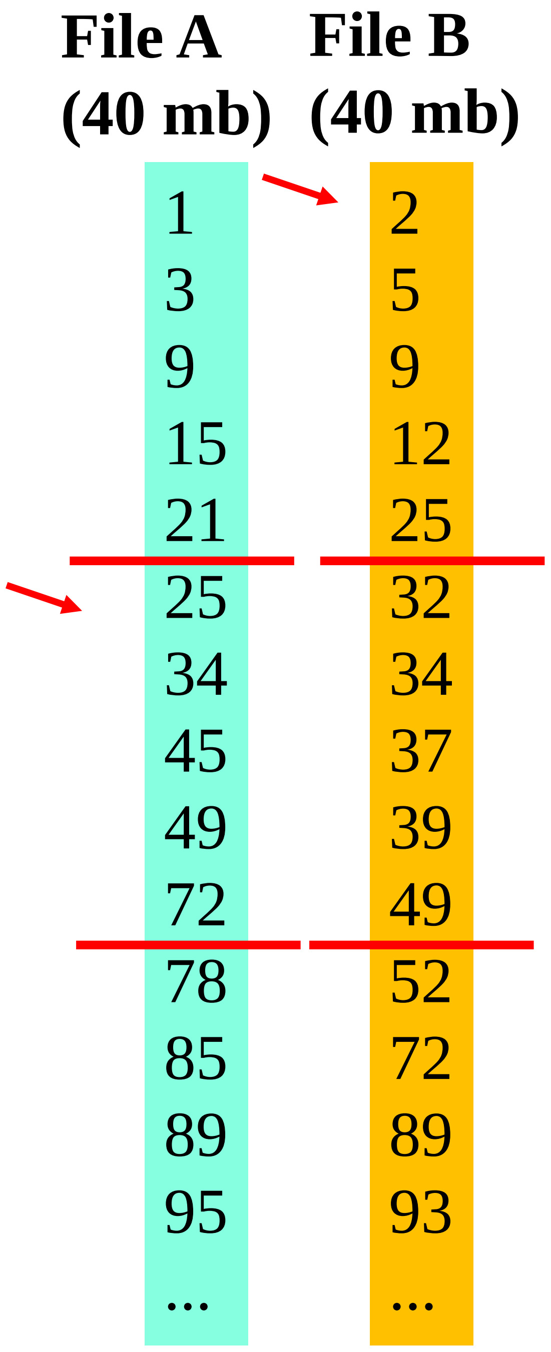
btt = 0.84ms,
B=2400bytes

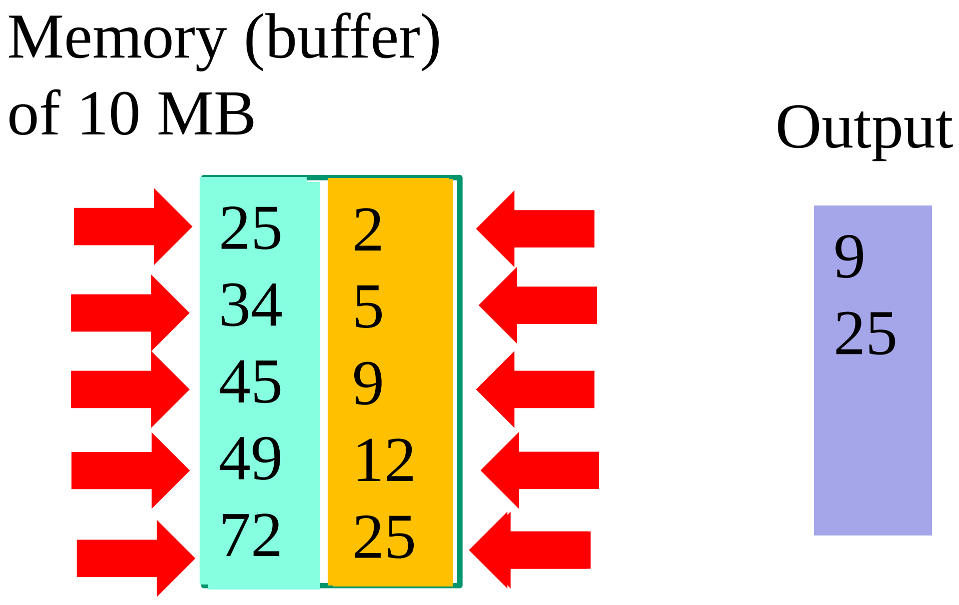
Intersection:sorted files



- Reserve 5 MB for each file
- Read 5 MB chunk from each
- Intersect in the memory

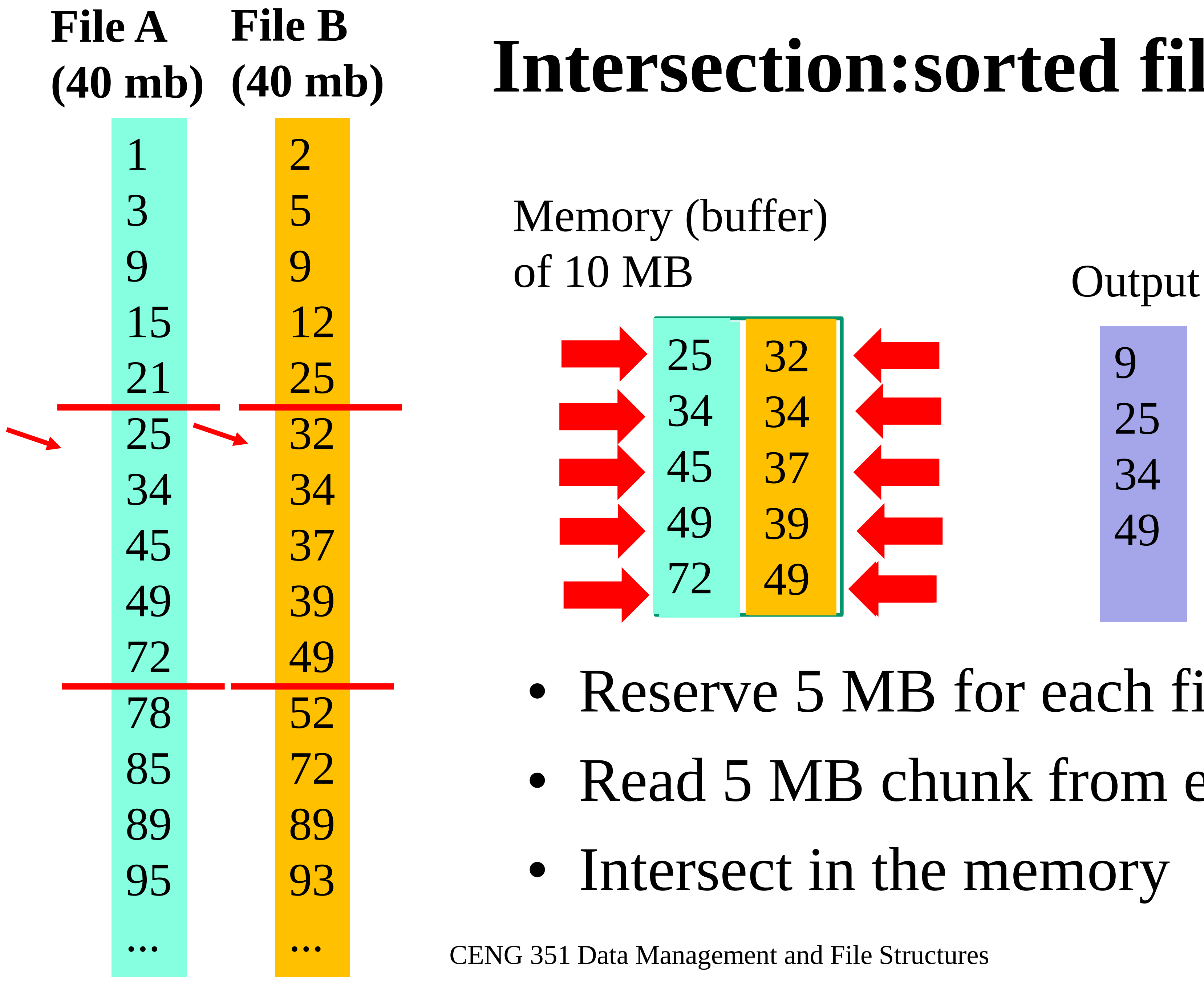
Intersection:sorted files





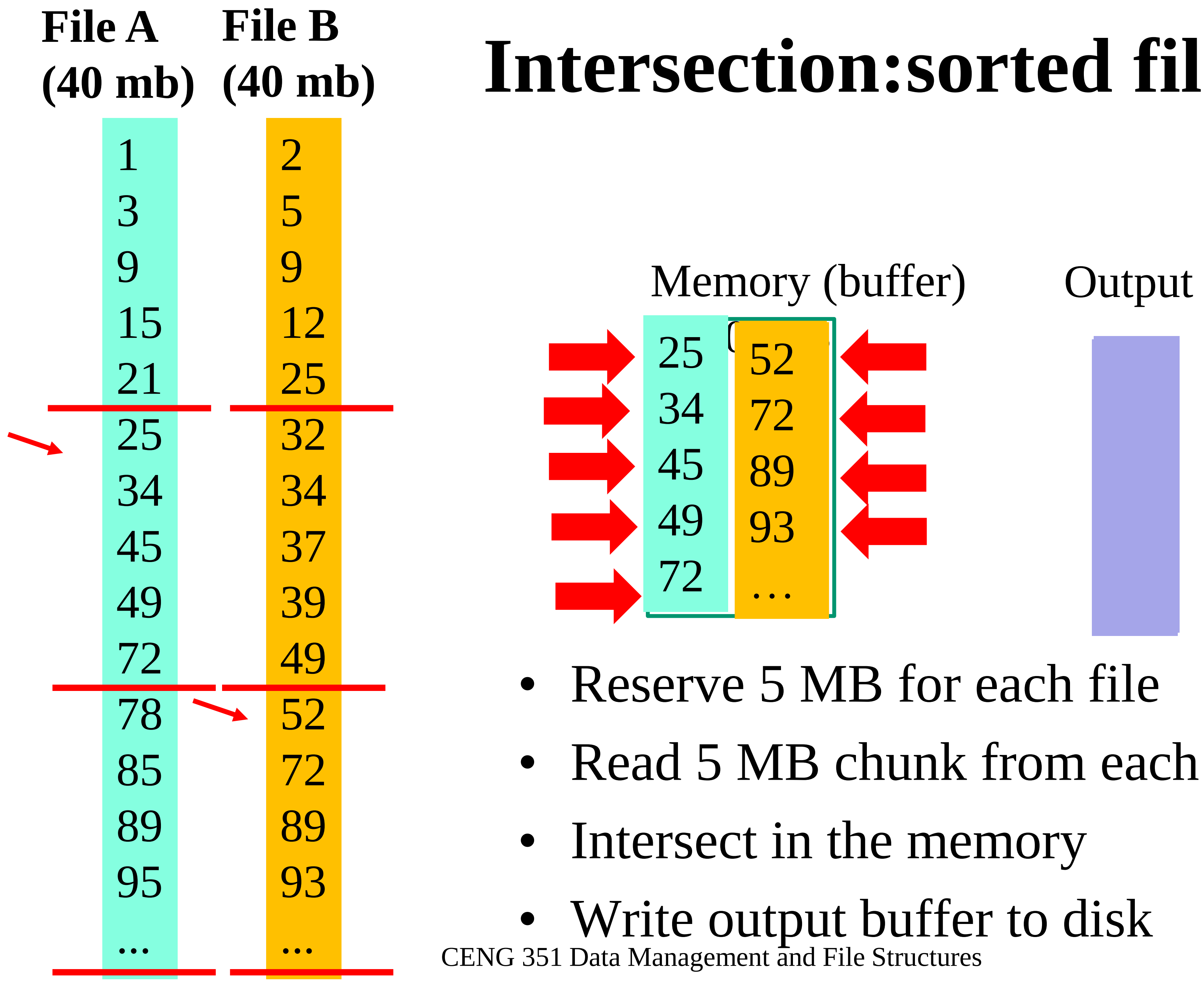
- Reserve 5 MB for each file
- Read 5 MB chunk from each
- Intersect in the memory

Intersection:sorted files



- Reserve 5 MB for each file
- Read 5 MB chunk from each
- Intersect in the memory

Intersection:sorted files



- Reserve 5 MB for each file
- Read 5 MB chunk from each
- Intersect in the memory
- Write output buffer to disk

- <u>Solution:</u>
  - Read a segment from A.
  - Start reading file B and compare records, marking common records.
  - As soon as we find a record in B whose key is greater than the largest value in memory write out the matched records from A.
  - Read the next segment from A ...
- Thus we read through each file once:
  - Read file A => 14 sec.
  - Read file B => 14 sec.
  - Write intersection => 10 sec.
  - Total = 38 sec.