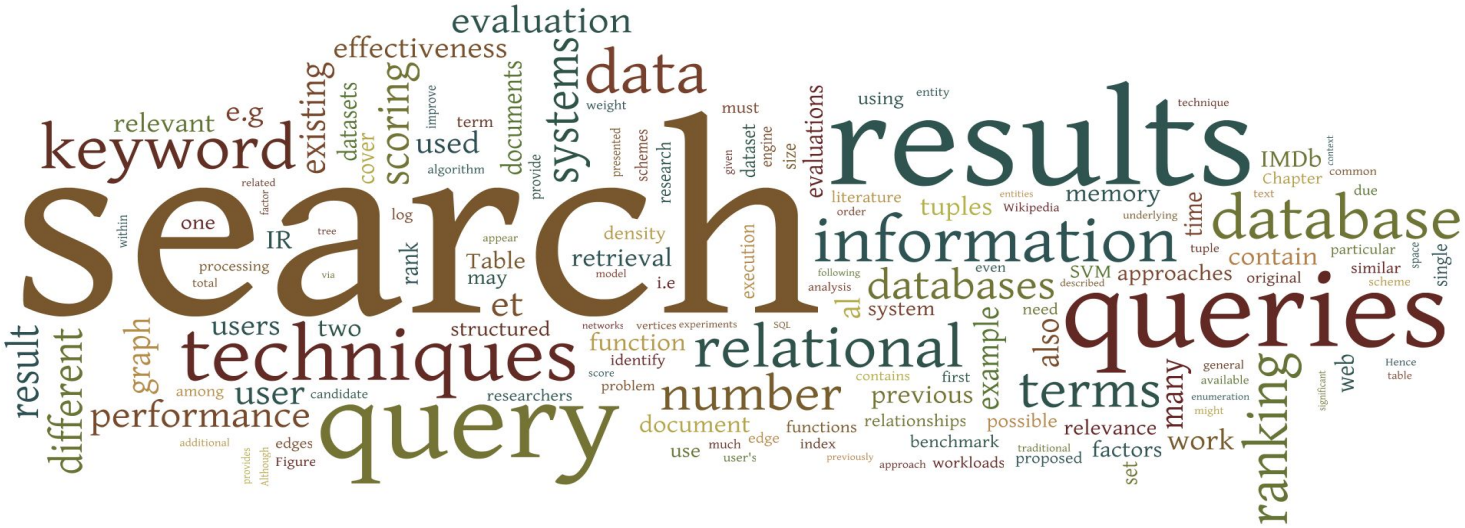
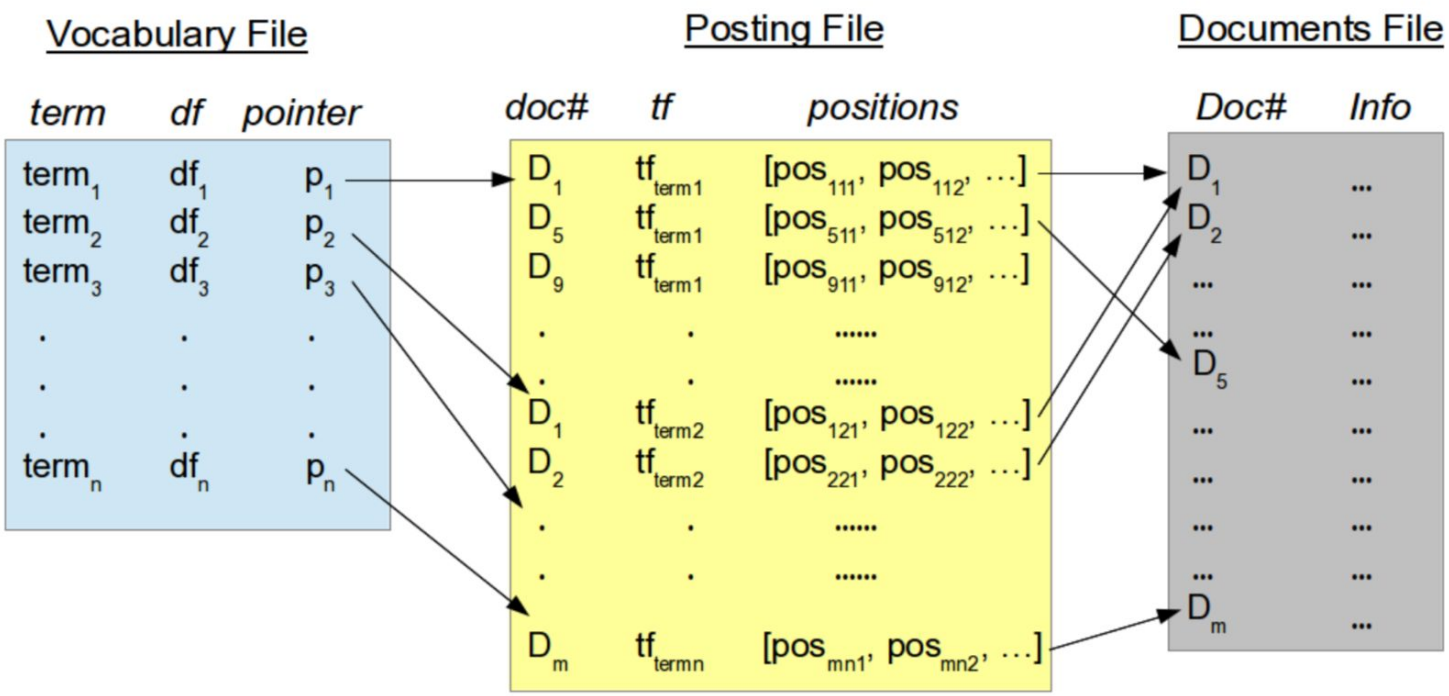
# Partial Indexing and Merging

Nikos Myrtakis and Panagiotis Papadakos

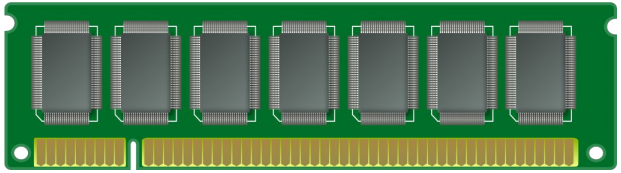


# The Inverted File



# Why Partial Indexing ?

- In memory index construction does not scale
- We should be able to construct indexes for very large collections
- Posting file occupies the largest amount of memory
- Think a collection of **5 gb** size
  - For an inverted file 3 times the collection size, we would need 15 gb!
  - Solution: Write chunks of postings to disk and merge them afterwards to a single posting file

# The Two Extremes

Write partially the inverted file and merge it with other partial inverted files to construct the final one
→ Fast and can adapt on the available resources

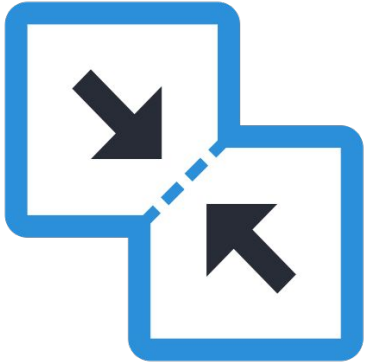
Write each record immediately when is indexed
Too many I/Os
→ Low need in resources but requires lot of time to finish

Keep all inverted file to main memory and write it directly after reading the collection
Requires too much resources
→ Extremely fast but requires huge amount of main memory

# An Overview

Read the text collection once

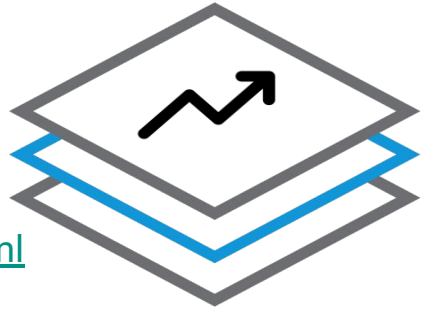
Write the partial indices sorted when RAM is going to exceed or the inverted file reaches a certain number of records



Merge the partial indices by keeping the sorted order to construct the final inverted file

# Increasing Heap Size

- In order to utilize better the main memory during indexing, you should increase the heap size of JVM
- Increase heap size from IDE
  - Intellij: https://www.jetbrains.com/help/idea/increasing-memory-heap.html
  - Netbeans: https://stackoverflow.com/questions/15460779/how-to-increase-the-java-heap-size-in-netbeans
  - Eclipse: http://www.planetofbits.com/eclipse/increase-jvm-heap-size-in-eclipse
- Increase heap size through terminal
  - https://stackoverflow.com/questions/1565388/increase-heap-size-in-java



# Partial Indexing Process - Introduction

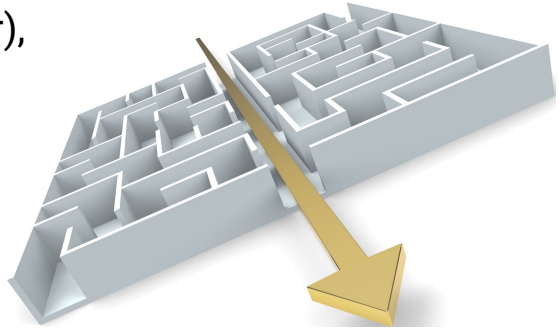
- Choosing when the partial indices should be written
  - **Option 1**: Write the inverted file each time heap size is going to exceed
    - Periodically check if heap size is going to exceed. E.g. every 5 seconds check if heap size reaches 80% usage
      - https://stackoverflow.com/questions/12807797/java-get-available-memory
    - This approach could be slow so choose a period of time to ask the remaining available memory

# Partial Indexing Process - Introduction

- Choosing when the partial indices should be written
  - **Option 2**: Choose how many records in the inverted file will be kept in memory
    - Define a good threshold
      Assume that:
      - Posting file contains: Tf (double = 8 bytes) and a pointer to document (long = 8 bytes)
      - Vocabulary file contains: word (string = 2 bytes/char), df (long = 8 bytes), a pointer to posting file (long = 8 bytes) and an offset (int = 4 bytes) indicating how many records to read in posting file

      If a word has approximately 10 chars (=20 bytes) then we need 56 bytes for each record in vocabulary and posting file. If the heap size is 4 GB **a good threshold could be 50 million records utilizing the 80% of available heap size**

More Simple



# Partial Indexing Process - Introduction

- A partial index may contain
  - A partial vocabulary file that is sorted lexicographically based on each word and stores
    - Word
    - DF of the word
    - Pointer to the partial posting file
  - A partial posting file
    - TF
    - A document id
    - Pointer to the document file

# Partial Indexing Process - Example

Partial Vocabulary 1

| Word | DF | Pointer |
|---|---|---|
| Brutus | 1 | P |
| Caesar | 1 | P |
| Julius | 1 | P |
| Killed | 1 | P |

Partial Posting File 1

| TF | #Doc | Pointer |
|---|---|---|
| 1.5 | D2 | P |
| 2.33 | D1 | P |
| 5.3 | D1 | P |
| 4 | D2 | P |

Inverted Index 1

Partial Vocabulary 2

| Word | DF | Pointer |
|---|---|---|
| Brutus | 1 | P |
| Caesar | 1 | P |
| Noble | 1 | P |
| With | 1 | P |

Partial Posting File 2

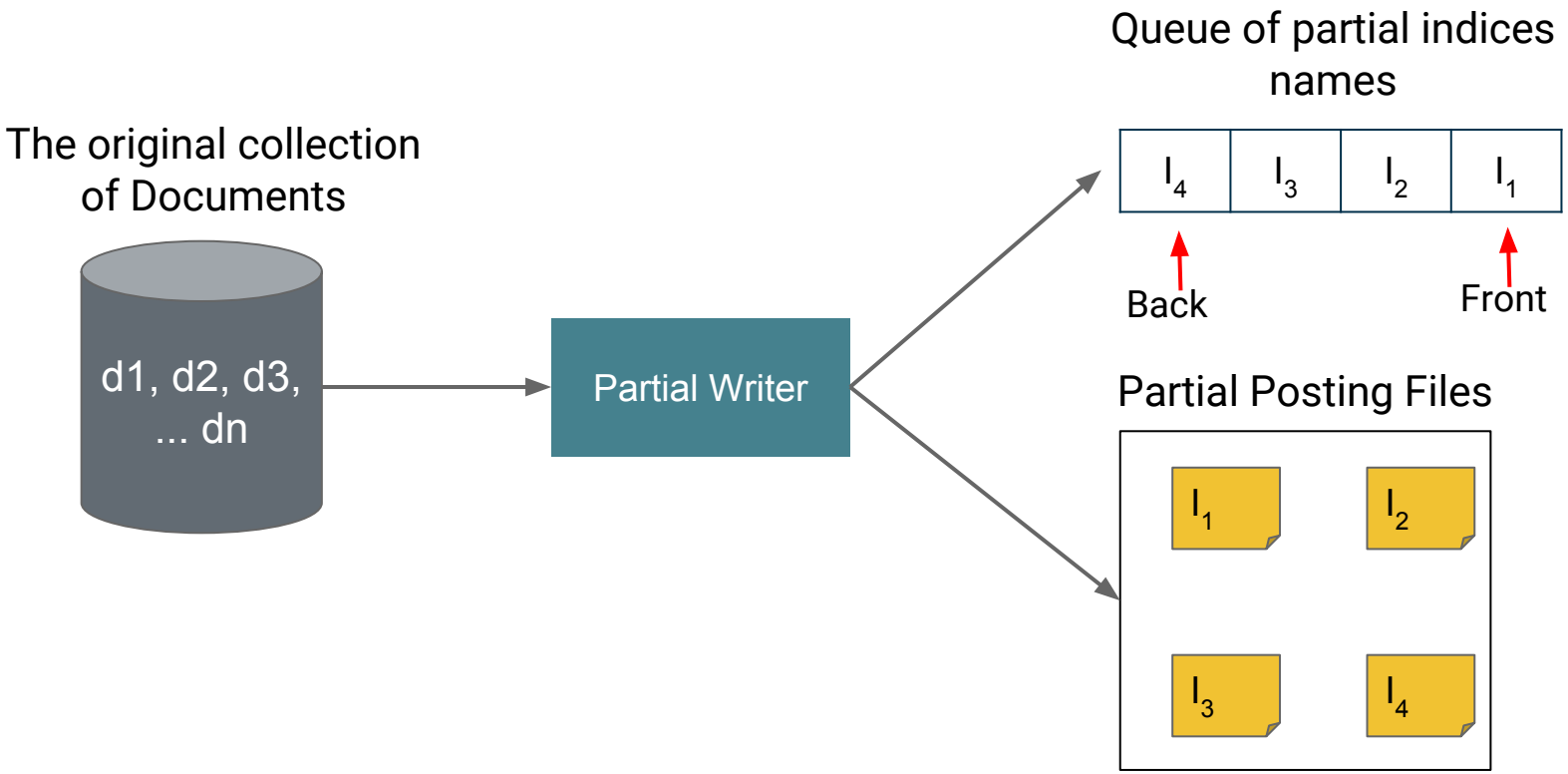
| TF | #Doc | Pointer |
|---|---|---|
| 2 | D3 | P |
| 3.33 | D4 | P |
| 6.3 | D3 | P |
| 4.5 | D4 | P |

Inverted Index 2

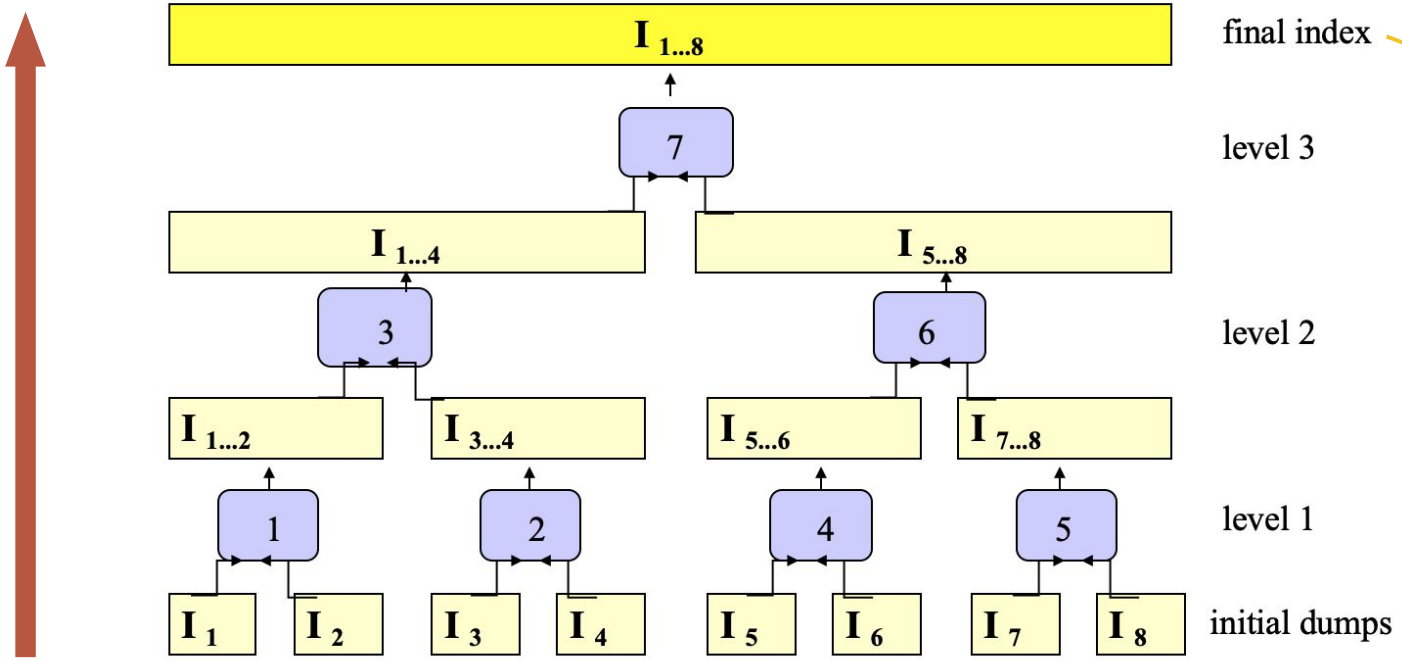
# Partial Indexing Process - Pseudocode

```
for each document d in collection
    for each word w of d
        if inverted file size reaches the threshold t and the current document ends
            Sort the records in current vocabulary file lexicographically
            Start writing the current posting file and for each word keep the bytes to its first
            record in posting file
            Write the partial vocabulary file and use the stored bytes to point in the first
            record of each word in its first record in posting file
            Keep a data structure (e.g. a queue) in main memory to store the file names of
            the partial indexes
            Clear the inverted file from memory
        endif
        Insert the word w to the inverted file
    endfor; endfor
```

# Partial Indexing Process - Output



# Merging Process - Overview




The final inverted file

The Partial Indexes

# Merging Process - Example

Partial Vocabulary 1

| Word | DF | Pointer |
|---|---|---|
| Brutus | 1 | P |
| Caesar | 1 | P |
| Julius | 1 | P |
| Killed | 1 | P |

Partial Vocabulary 2

| Word | DF | Pointer |
|---|---|---|
| Brutus | 1 | P |
| Caesar | 1 | P |
| Noble | 1 | P |
| With | 1 | P |

Merged Vocabulary from 1 and 2

| Word | DF | Pointer |
|---|---|---|
| Brutus | 2 | P’ |

Merged Posting file

| TF | #Doc | Pointer |
|---|---|---|
| 2.33 | D1 | P |
| 3.33 | D4 | P |

# Merging Process - Example

Partial Vocabulary 1

| Word | DF | Pointer |
|---|---|---|
| Brutus | 1 | P |
| Caesar | 1 | P |
| Julius | 1 | P |
| Killed | 1 | P |

Partial Vocabulary 2

| Word | DF | Pointer |
|---|---|---|
| Brutus | 1 | P |
| Caesar | 1 | P |
| Noble | 1 | P |
| With | 1 | P |

Merged Vocabulary from 1 and 2

| Word | DF | Pointer |
|---|---|---|
| Brutus | 2 | P’ |
| Caesar | 2 | P’ |

Merged Posting file

| TF | #Doc | Pointer |
|---|---|---|
| 2.33 | D1 | P |
| 3.33 | D4 | P |
| 1.5 | D2 | P |
| 2 | D3 | P |

# Merging Process - Example

Partial Vocabulary 1

| Word | DF | Pointer |
|---|---|---|
| Brutus | 1 | P |
| Caesar | 1 | P |
| Julius | 1 | P |
| Killed | 1 | P |

Partial Vocabulary 2

| Word | DF | Pointer |
|---|---|---|
| Brutus | 1 | P |
| Caesar | 1 | P |
| Noble | 1 | P |
| With | 1 | P |

Merged Vocabulary from 1 and 2

| Word | DF | Pointer |
|---|---|---|
| Brutus | 2 | P’ |
| Caesar | 2 | P’ |
| Julius | 1 | P’ |

Merged Posting file

| TF | #Doc | Pointer |
|---|---|---|
| 2.33 | D1 | P |
| 3.33 | D4 | P |
| 1.5 | D2 | P |
| 2 | D3 | P |
| 5.3 | D1 | P |

# Merging Process - Example

Partial Vocabulary 1

| Word | DF | Pointer |
|---|---|---|
| Brutus | 1 | P |
| Caesar | 1 | P |
| Julius | 1 | P |
| Killed | 1 | P |

Partial Vocabulary 2

| Word | DF | Pointer |
|---|---|---|
| Brutus | 1 | P |
| Caesar | 1 | P |
| Noble | 1 | P |
| With | 1 | P |

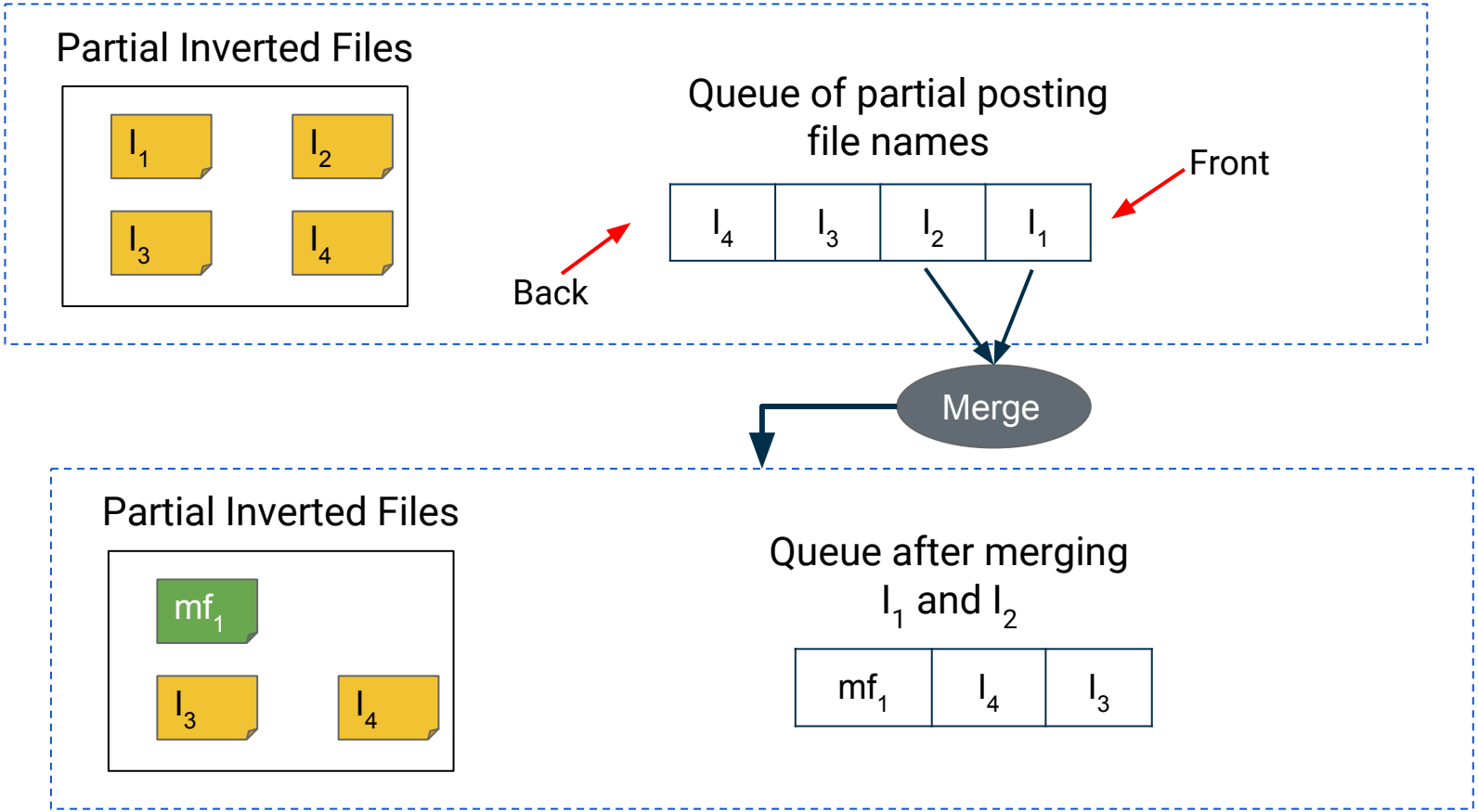
Merged Vocabulary from 1 and 2

| Word | DF | Pointer |
|---|---|---|
| Brutus | 2 | P’ |
| Caesar | 2 | P’ |
| Julius | 1 | P’ |
| ... | ... | ... |

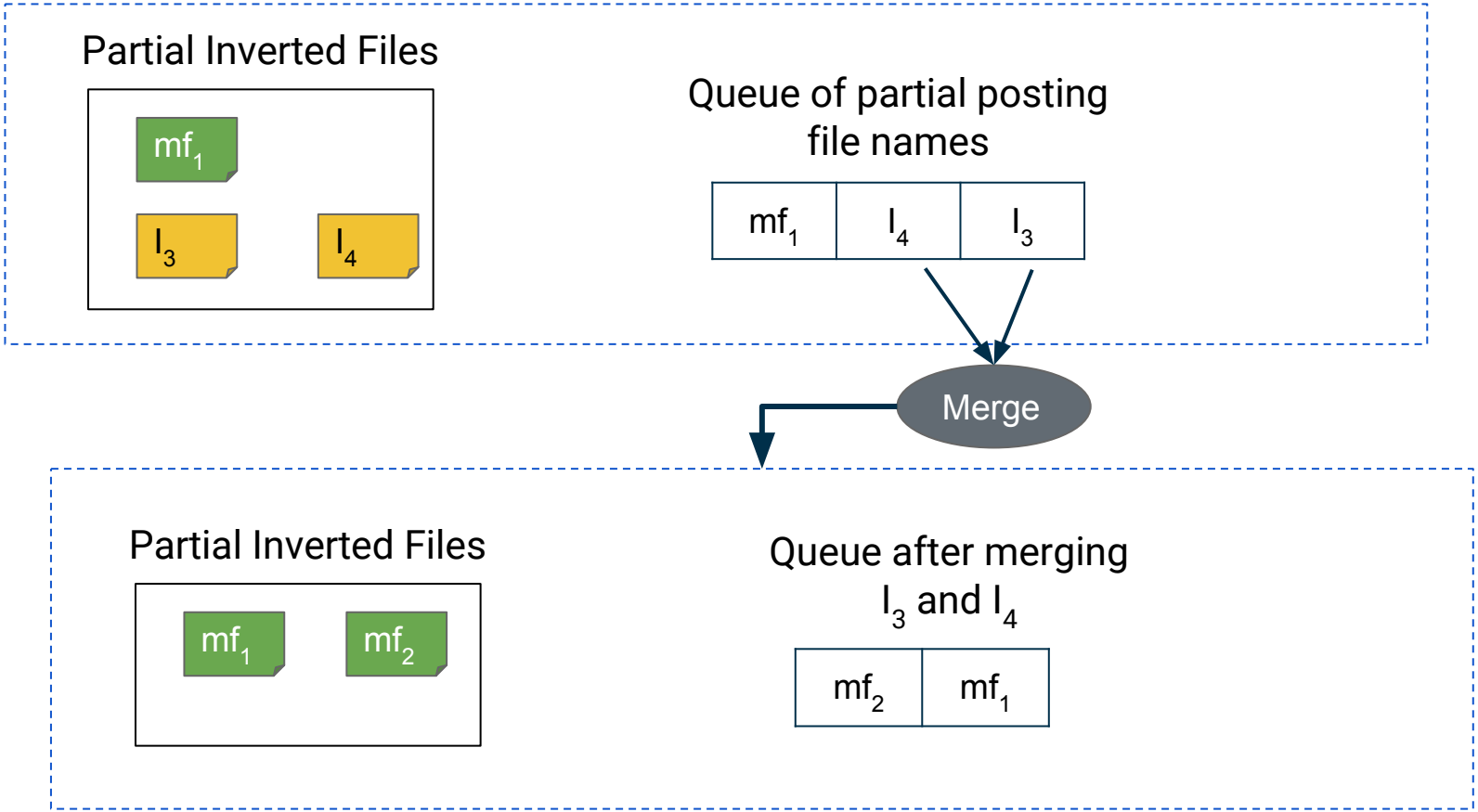
Merged Posting file

| TF | #Doc | Pointer |
|---|---|---|
| 2.33 | D1 | P |
| 3.33 | D4 | P |
| 1.5 | D2 | P |
| 2 | D3 | P |
| 5.3 | D1 | P |
| ... | ... | ... |

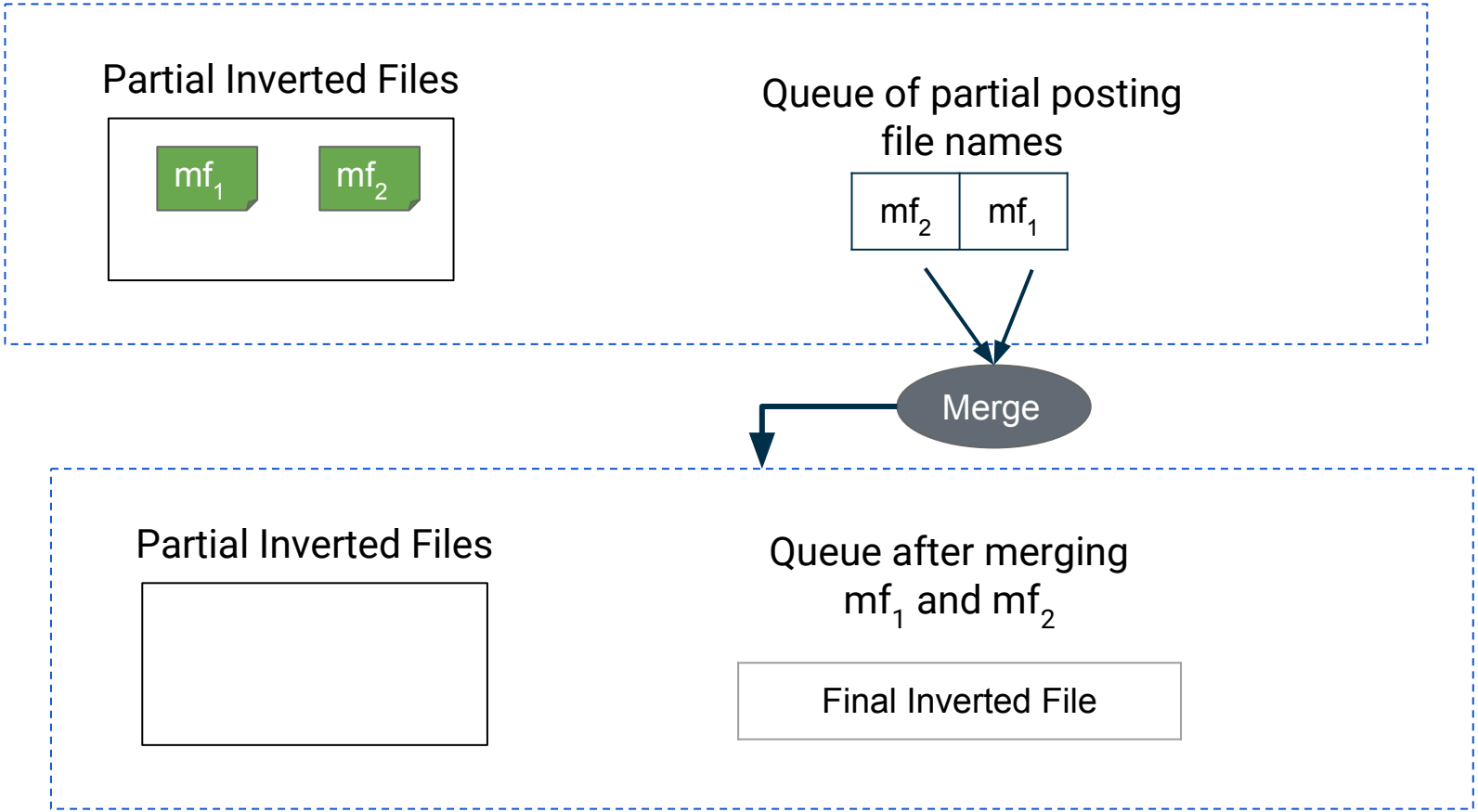
# Merging Process - Schema



# Merging Process - Schema



# Merging Process - Schema



# Updating Vocabulary File

- **Case 1**: If two words are not equal
  - Find the smaller word lexicographically
  - Write its data from partial posting in the common posting file
  - Write the data from partial vocabulary file in the common vocabulary file and move the file pointer from file with the smaller word
- **Case 2**: If two words are equal
  - Merge two lines and update word's DF. Add DF of the first line with DF of the second
  - Keep the word's data in main memory until a different word is discovered
  - Write the word's data in the common vocabulary file and move the file pointers of both files

# Merging Process - Pseudocode

```
Let Q be a queue that keeps every file name from partial inverted indices
for each pair of partial indices I1, I2 in Q
    Pop I1 and I2 from Q and get the partial vocabulary and posting file names
    for each word w_i of file vocab1 and w_j of file vocab2
        if w_i < w_j
            Write posting data from the pointer to posting file of w_i vocab1 record, write w_i to the
            merged vocab file and move file pointer of file vocab1
        elseif w_i > w_j
            Write posting data from the pointer to posting file of w_j vocab2 record, write w_j to the
            merged vocab file and move file pointer of file vocab2
        else    ▷ The two words are equal
            Keep the record in memory, update DF and write the word's data when a new word is discovered. Move
            the file pointers from both files
        endif
        Close and delete the files used for merging
        Add the merged file name to Q
        endfor; endfor
```

# Useful Tips

- Build indexes in a small collection for testing (check the partial indexing using a small threshold e.g. 10 records). Then, scale it to the full collection.
- Give the JVM at least 4 gb of heap space (if available)
- Perform plenty of tests
  - Each word in vocabulary file must have its posting data
  - Each record in posting file should correspond to a document
- Use **random access file API** from java to obtain the bytes and seek into posting and document file

# Conclusion

- ✓ Partial indexing is a good technique to index large collections
- ✓ It is preferable to define a threshold and tune it according to the available resources.
- ✓ Keep a data structure (e.g. a queue) in main memory that stores the file names of partial indices and use it in the merging phase
- ✓ Be careful while merging. The lexicographic order should be preserved!