Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

Ecole supérieure en sciences et technologies de l'informatique et du numérique

Sequential structures : Part 1

Presented by : Dr. Daoudi Meroua

Academic year: 2024/2025

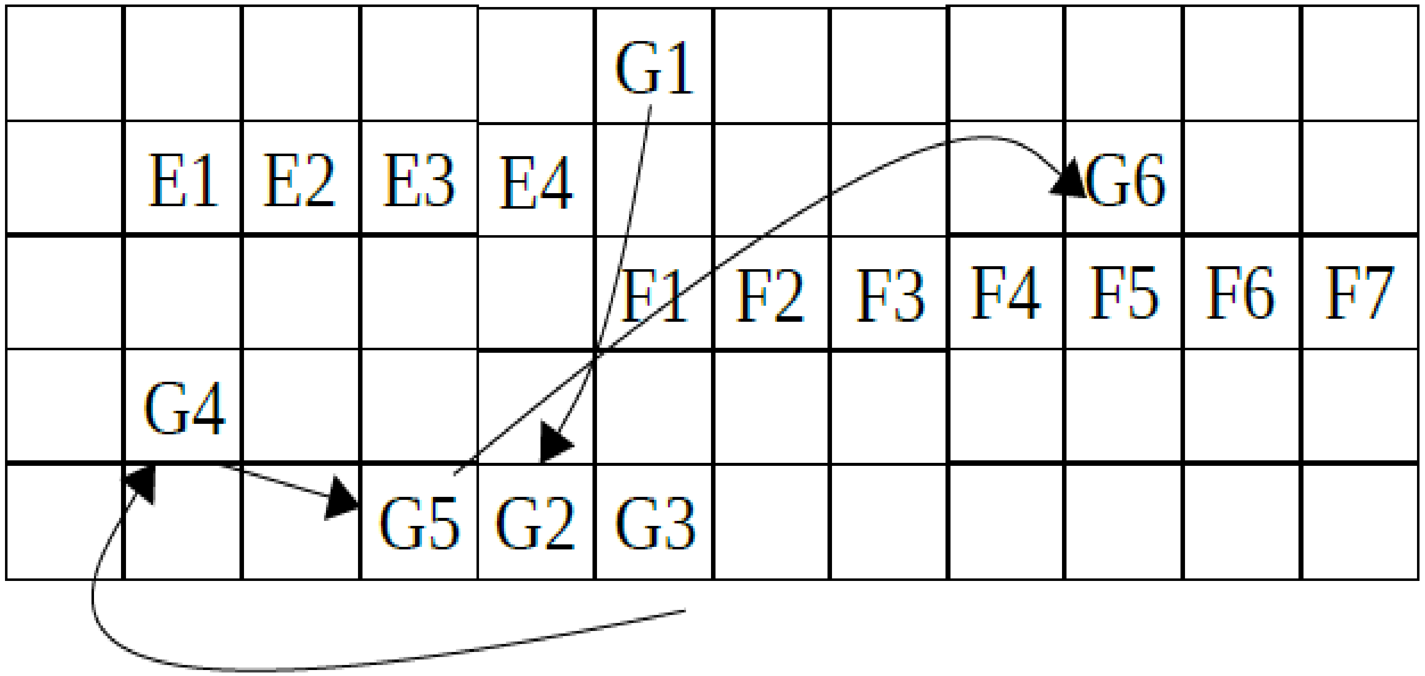
File structures

File structures are the combination of:

- The organization of data within files
- The methods of file access

The goal is to define and design data structures and associated algorithms tc organize the blocks of a file on storage media (SM) and to implement certair access operations for data manipulation within the file.

File modeling



Performance of file structures

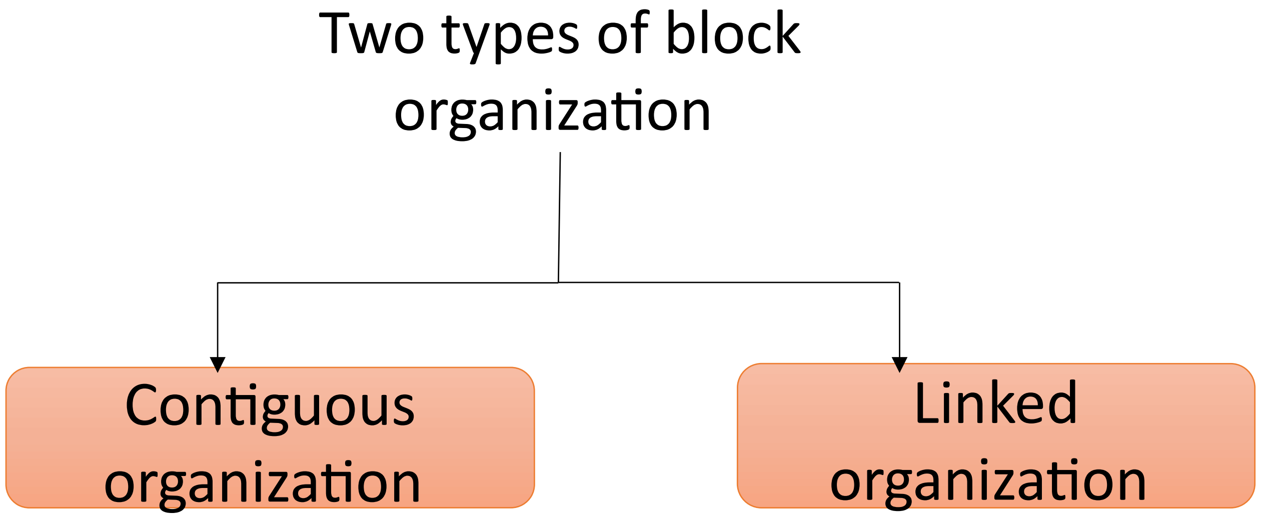
- Number of physical Input/Output operations performed: the algorithmi complexity is directly related to the number of Input/Output operations.
- Another performance criterion concerns memory space usage:
- Memory overhead: ⇒ Size of the data structures used / Size of the data store ≥ 100% (the smaller the ratio, the more efficient it is)
- Loading factor: ⇒ Number of data inserted / Number of available slots In]0% 100%] (the larger the ratio, the better)

Organization methods

The organization of files refers to how data is stored within a file. File organization is crucial as it determines access methods, efficiency, flexibility, and the types of storage devices to be used. There are four methods of file organization on a storage medium, which include:

- ➔ Sequential organization
- ➔ Indexed organization
- ➔ Hierarchical organization (tree methods)
- ➔ Random organization (hashing methods)

Global organization of blocks

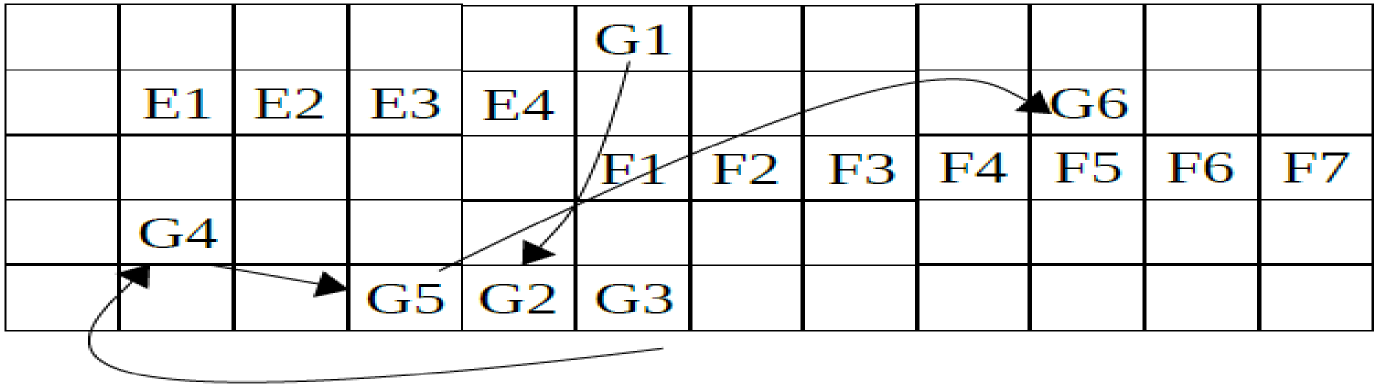


The file is viewed as an array of blocks.

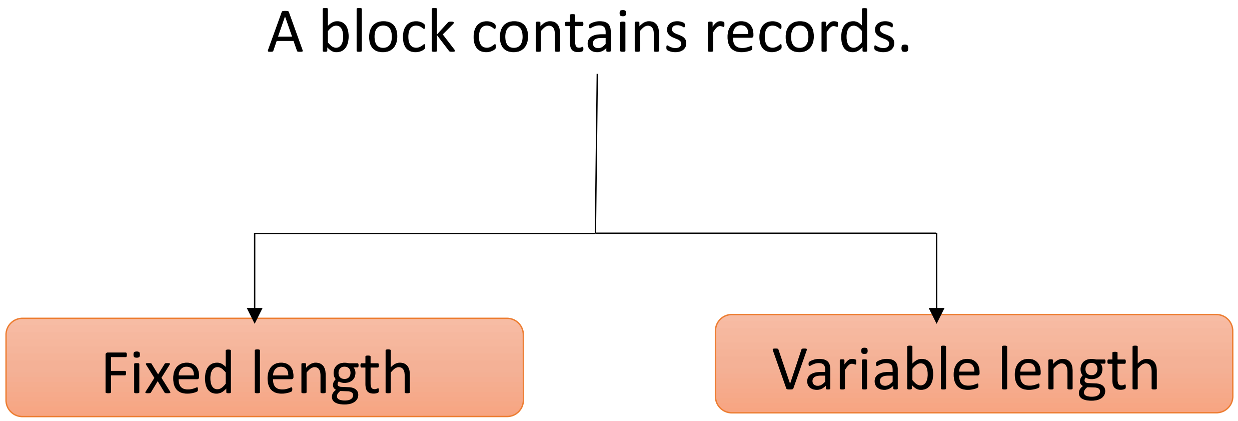
The file is viewed as a linked list of blocks.

Global organization of blocks

In the figure below, we have two files viewed as arrays (E and F) and one file viewed as a list (G). The blocks F1, F2, … F7 are contiguous, as are the blocks E1, E2, E3, and E4. However, the blocks G1, G2, … G6 are not contiguous; they are linked together, forming a list of blocks



Internal organization of blocks



Internal organization of blocks

Fixed-length records

Each block can then contain an array of records of the same type

```
Type Tenreg = structure // Structure of a file record
matricule : chaine(10);
nom : chaine(20);
age : entier;
...
fin;
```

```
Type Tbloc = structure // Structure of a file block
tab : tableau[1..b] de Tenreg;
NB : entier;
fin;
```

Internal organization of blocks

Variable-length records

Each record is viewed as a string of characters (of variable length).

➔It has one or more fields with variable sizes,

➔the number of fields varies from one record to another.

To separate the fields within the record, we can either use a special character ('#') that cannot be used as a valid value or prefix the beginning of the fields with their size (over a fixed number of positions).

Internal organization of blocks

Variable-length records

In the example below, we use 3 positions to indicate the size of the fields

...|0|2|7|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|a|0|1|7|b|b|b|b|b|b|b|b|b|b|b|b|b|b|0|
3|7|c|c|c|.......|c|c|.....

<--> <--------------------------------> <---------------->

A field with 27 characters and a field with 17 characters... In the case of variable-length records, the block cannot be defined as an array of records, because the elements of an array must always be of the same size.

Internal organization of blocks

Variable-length records

```
Type Tbloc = structure
        tab : tableau[1..b'] de caractères; // Character array for records
        suiv : entier; // Number of the next block in the list
Fin;
```

Note: Even though the records are of variable lengths, the size of the blocks remains fixed.

Internal organization of blocks

Overlapping

To minimize wasted space in blocks (in the case of variable format only), one can opt for an organization with overlapping between two or more blocks: when inserting a new record into a block that is not yet full and where the remaining empty space is not sufficient to fully contain that record, it will be split into two parts. The first part will occupy all the empty space of the block in question, while the remaining part (the second part) will be inserted into a newly allocated block for the file. This is referred to as the record being 'split across two blocks.

Simple file structures

Taxonomy of simple file structures

Notation:

T : For a file viewed as an array (tableau), use L: for a file viewed as a list

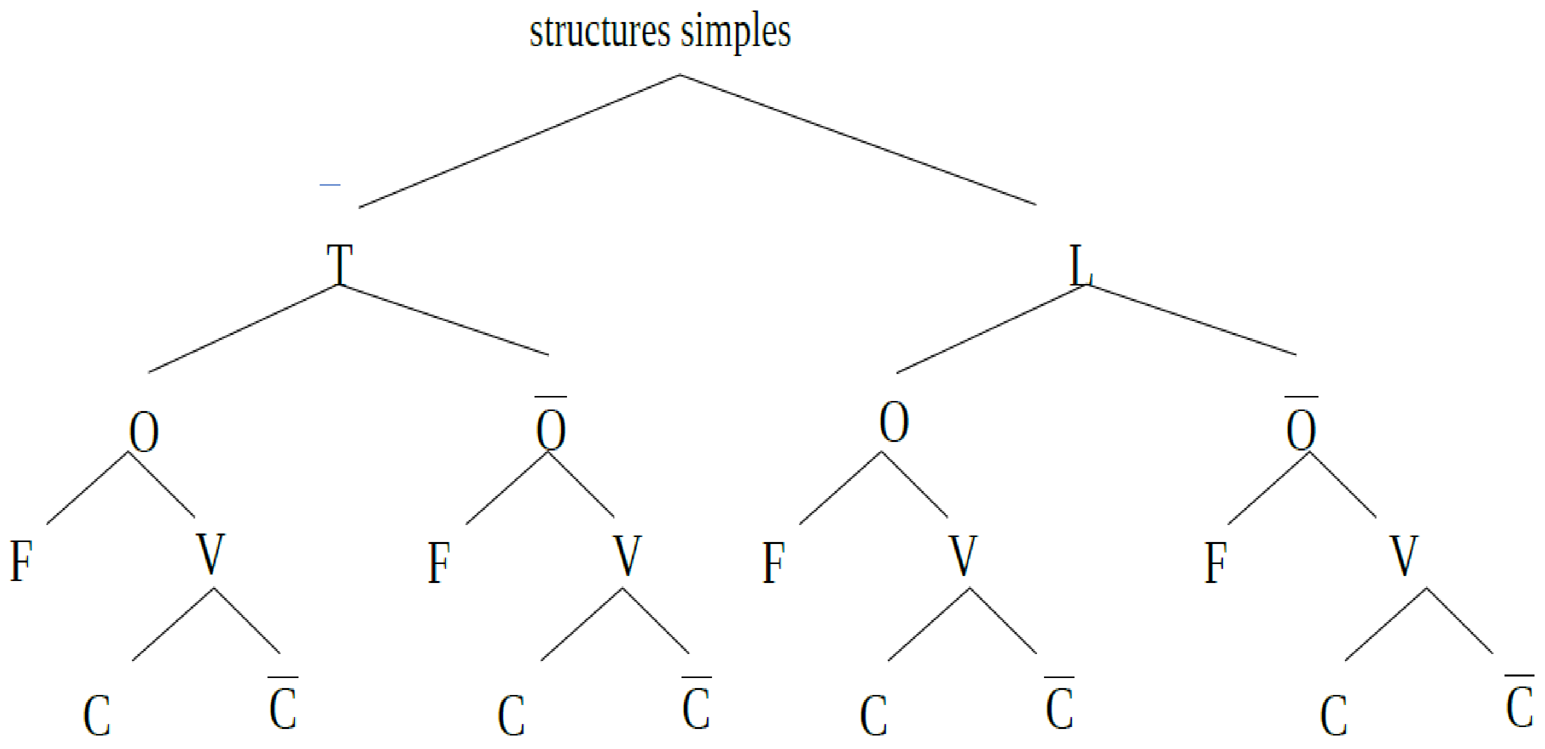
O: for ordered file, Ō: for unordered file.

F: for fixed record format, V: for variable record format.

C: with overlapping records between blocks, :Ć without overlapping. (C chevauchement)

Simple file structures

Taxonomy of simple file structures



Simple file structures

Examples :

T O VC method :

- Represents the organization of a file viewed as an array (T), unordered (O), with variable-length records (V) and allowing overlaps between blocks (C).
- The search is sequential, insertion occurs at the end of the file, and deletion is logical.

LOF method:

- File viewed as a list, ordered with fixed-length records.
- The search is sequential,
- The insertion causes intra-block shifts (to maintain the order of the records), and deletion can be either logical or physical.

TŌF   Example : Initial loading

```
TypeBloc = struct
        tab : tableau[b] de TypeEnreg
        NB : entier
Fin

F : FICHIER de TypeBloc BUFFER buf ENTETE ( entier , entier ) /* nb_bloc ,
nb_enreg */
e : TypeEnreg ;
i, j, k, n : entier ;
Debut
OUVRIR ( F , « fichier.dat », ’N’ ) // un nouveau fichier
Lire(n) ; i ← 1 ; j ← 1 // le remplir avec n enreg
```

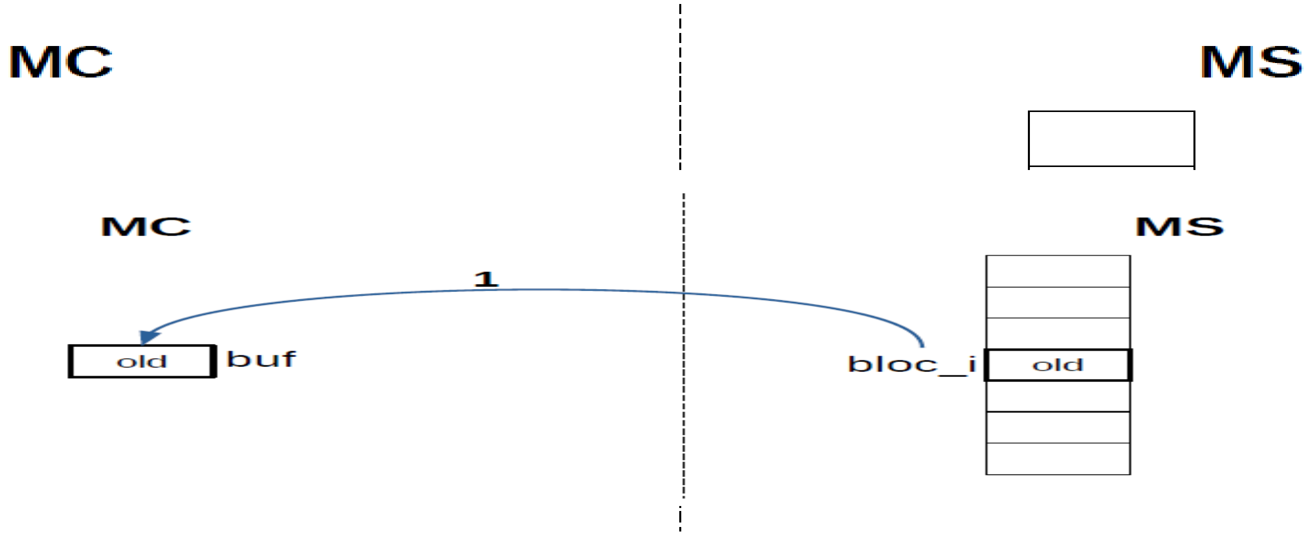
TŌF Example : Initial loading

```
Pour k = 1,n
Lire(e) // récupérer un enreg de l'entrée standard
SI ( j ≤ b*0.7 )
        buf.tab[ j ] ← e ; j++ // et le mettre dans le buffer buf
SINON // s'il n'y a pas de place dans buf,
        buf.NB ← j-1 ;
        ECRIREDIR( F , i , buf ) ; // le sauvegarder en MS et
        buf.tab[1] ← e ;
         i++ ;
         j ← 2 // remplir un nouveau
FSI
FP
buf.NB ← j-1
ECRIREDIR( F , i , buf) // écriture du dernier bloc
AFF_ENTETE( F , 1 , i ) ; AFF_ENTETE( F , 2 , n ) ; // m-a-j les caractéristiques
FERMER( F ) // et fermer le fichier
Fin
```

TŌF Example : Initial loading

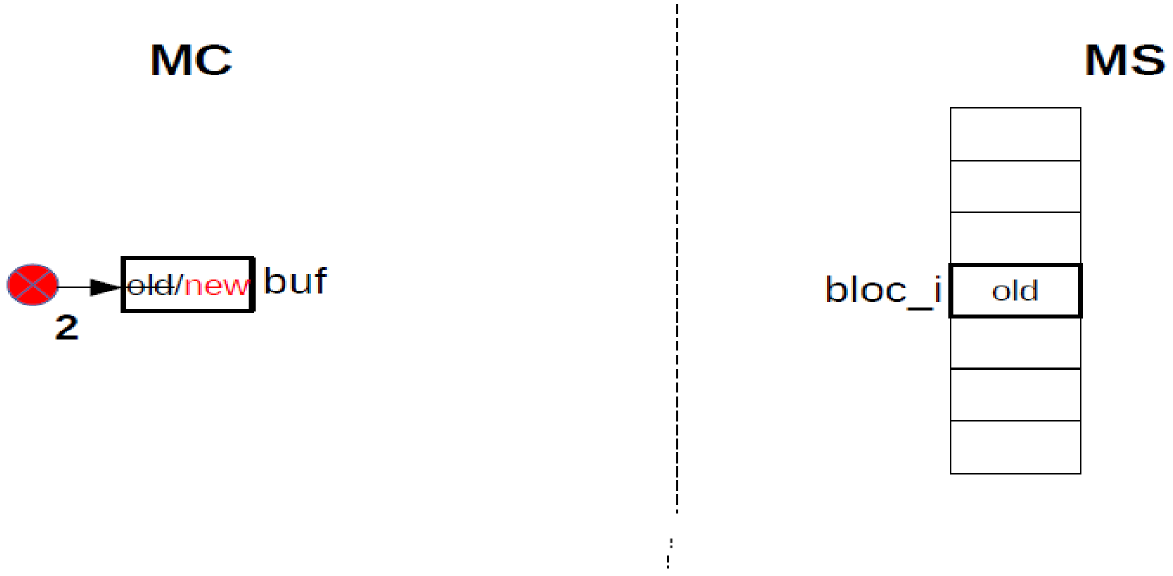
To modify the content of a block, you need to:

1- Load the content of the block into memory (in a buffer). ⇒ ***LireDir(F , i , buf)***



TŌF Example : Initial loading

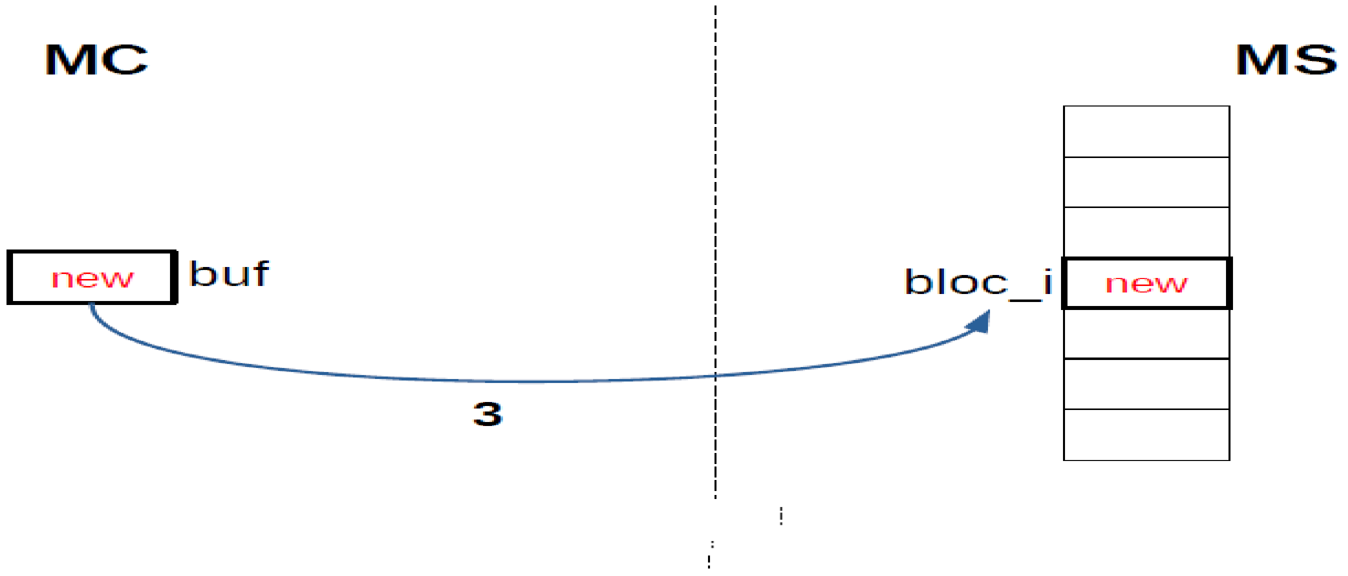
2- Make the modifications in the buffer (in MC) ⇒ *buf.tab[ ] ← … ; buf.NB ← …*



TŌF Example : Initial loading

3- Write the modified content of the buffer back into the block in MS ⇒

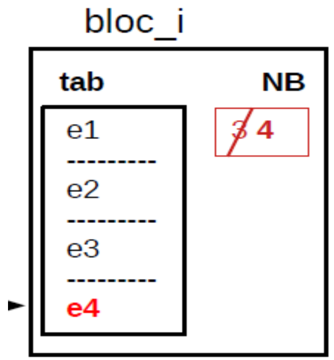
EcrireDir(F , i , buf)



TŌF Example : Initial loading

⇒ Insert the record e4 into block number i of F.

```
*OUVRIR( F , « … » , ‘A’ )*
…
*LIREDIR( F , i , buf )* // 1. Read the content of block i into memory.
SI ( buf.NB < Capacité_max )
      buf.NB++ //2.  Update the buffer in MC
      buf.tab[ buf.NB ] ← e4 // the field NB and the array tab
*ECRIREDIR( F , i , buf )* //3.  Write the content of the buffer into block i in MS
SINON
       écrire(«   Cannot insert into block i : it is already full   »)
FSI
…
*FERMER( F )*
```

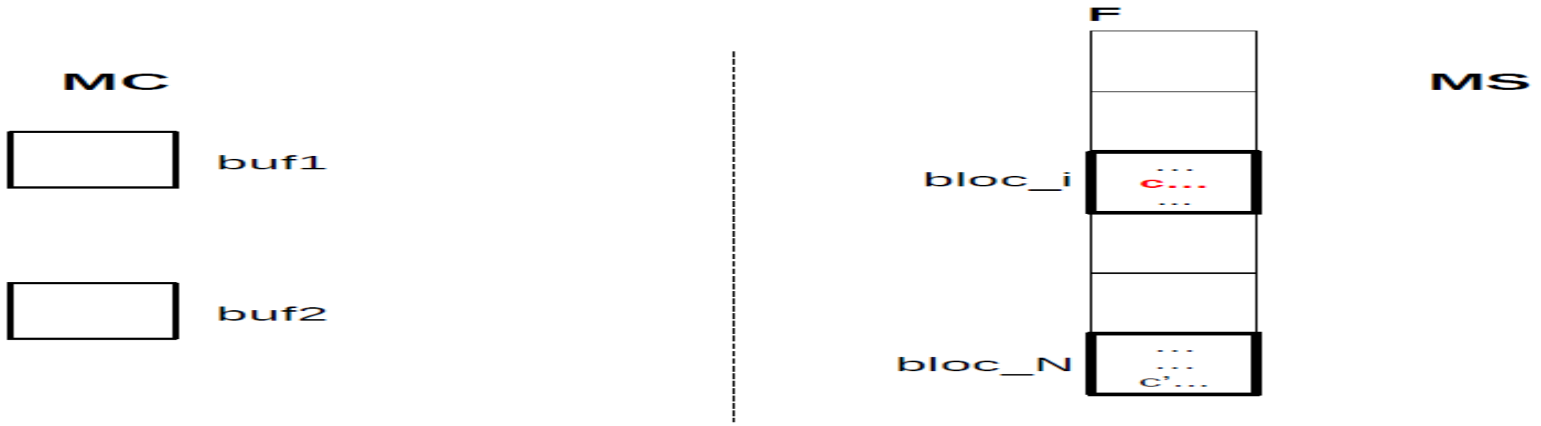


TŌF Example : Physical deletion

Physical deletion of a record by overwriting it with the last one (Using 2 buffers, buf1 and buf2 in MC).

To physically delete the record with key c, you need to:

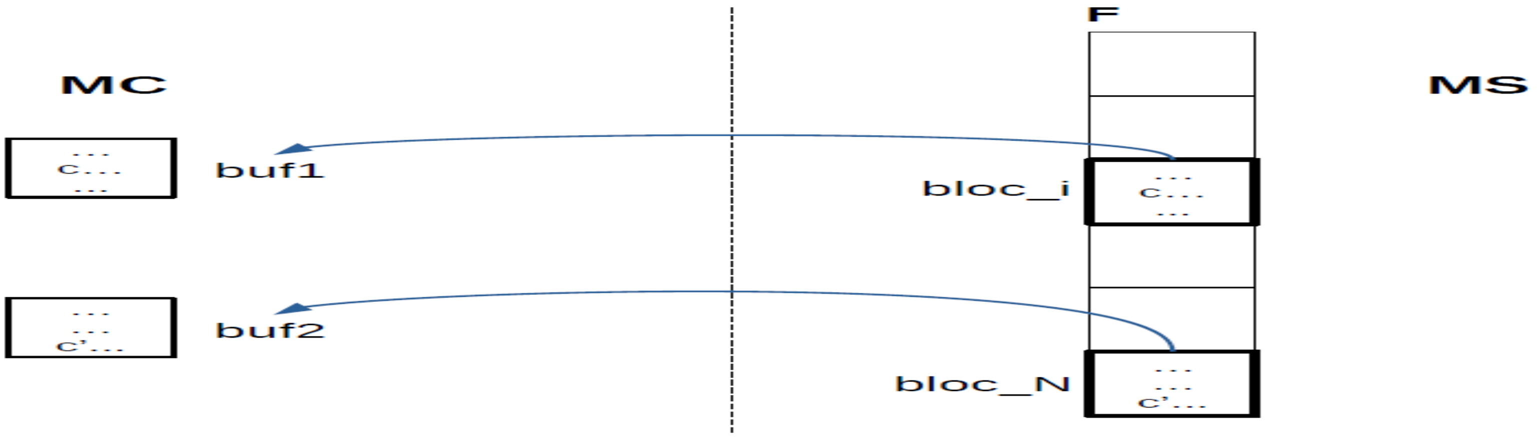
Locate the number (i) of the block containing the record c ⇒ for example, by searching for c.



TŌF Example : Physical deletion

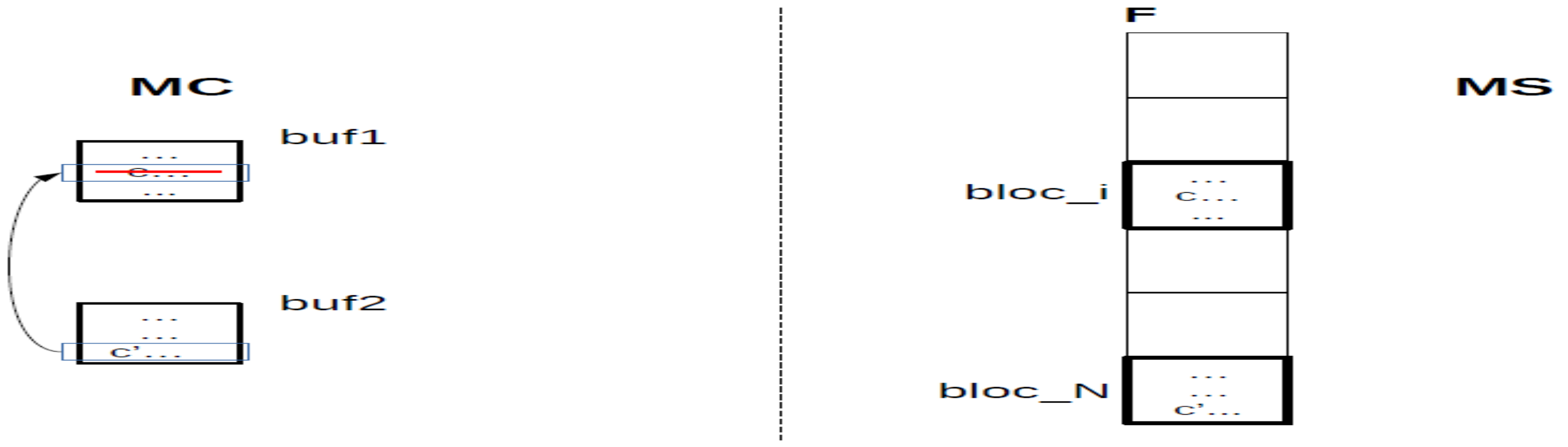
Retrieve the number (N) of the last block in file F ⇒ N ← Entete(F,1)

Read blocks i and N into MC ⇒ lireDir(F, i, buf1); EcrireDir(F, N, buf2)



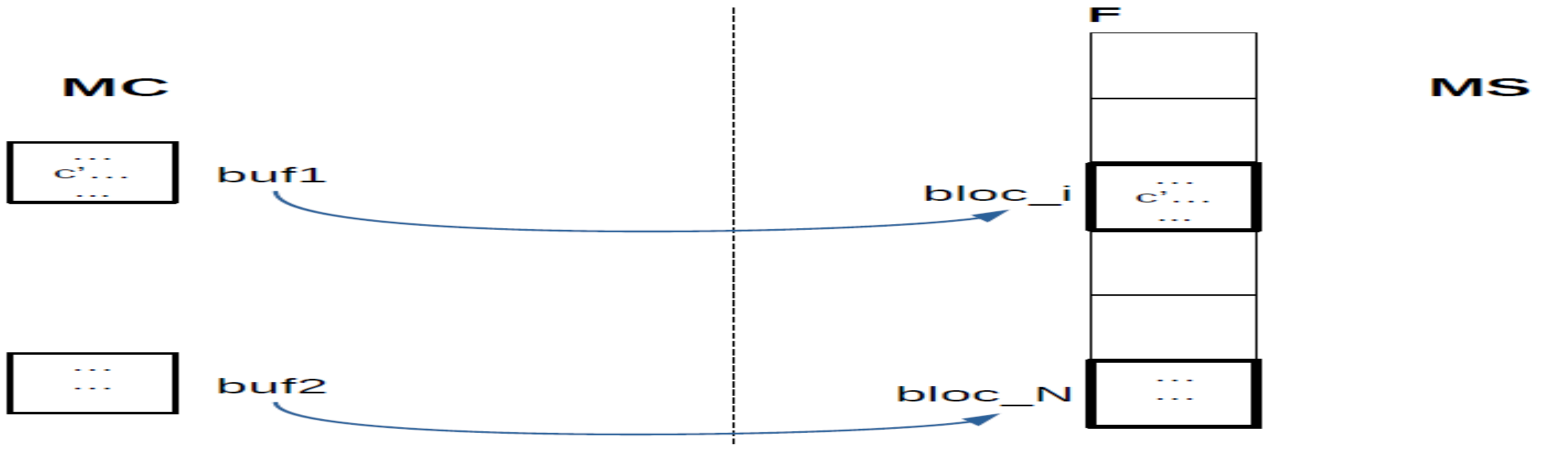
TŌF Example : Physical deletion

Overwrite c with c' in memory ⇒ by moving c' from buf2 to buf1.



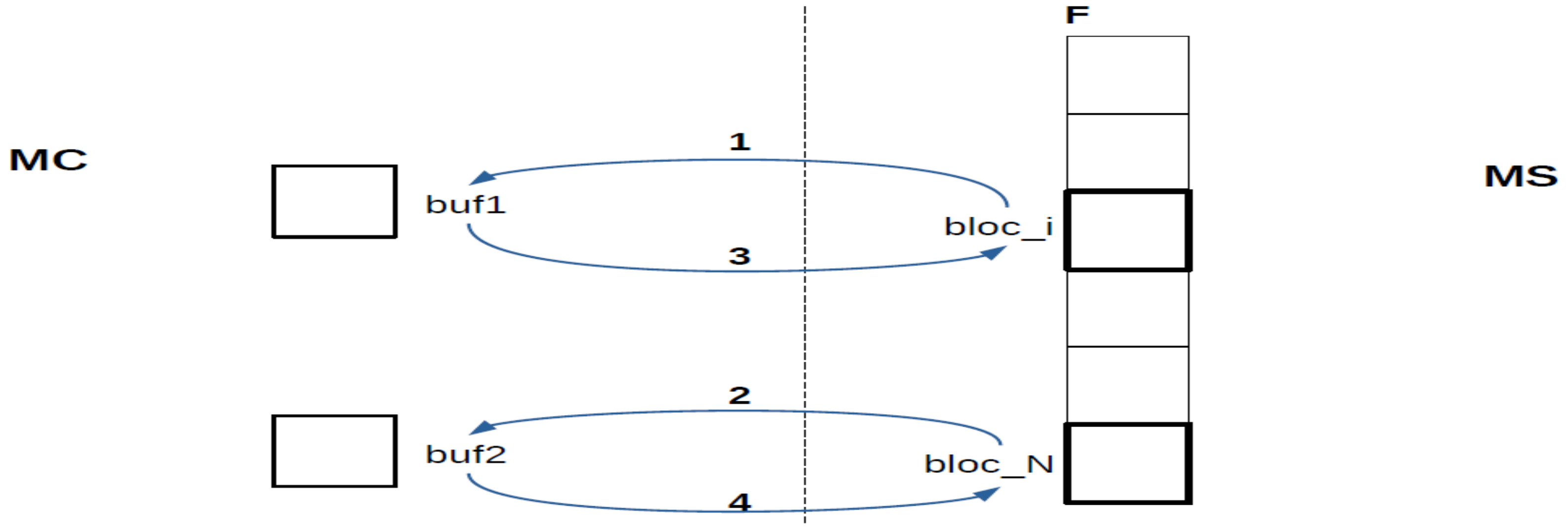
TŌF Example : Physical deletion

Write the contents of buf1 and buf2 to MC ⇒ EcrireDir(F, i, buf1); EcrireDir(F, N, buf2).



TŌF Example : Physical deletion

Cost in the worst case



TŌF Example : Physical deletion

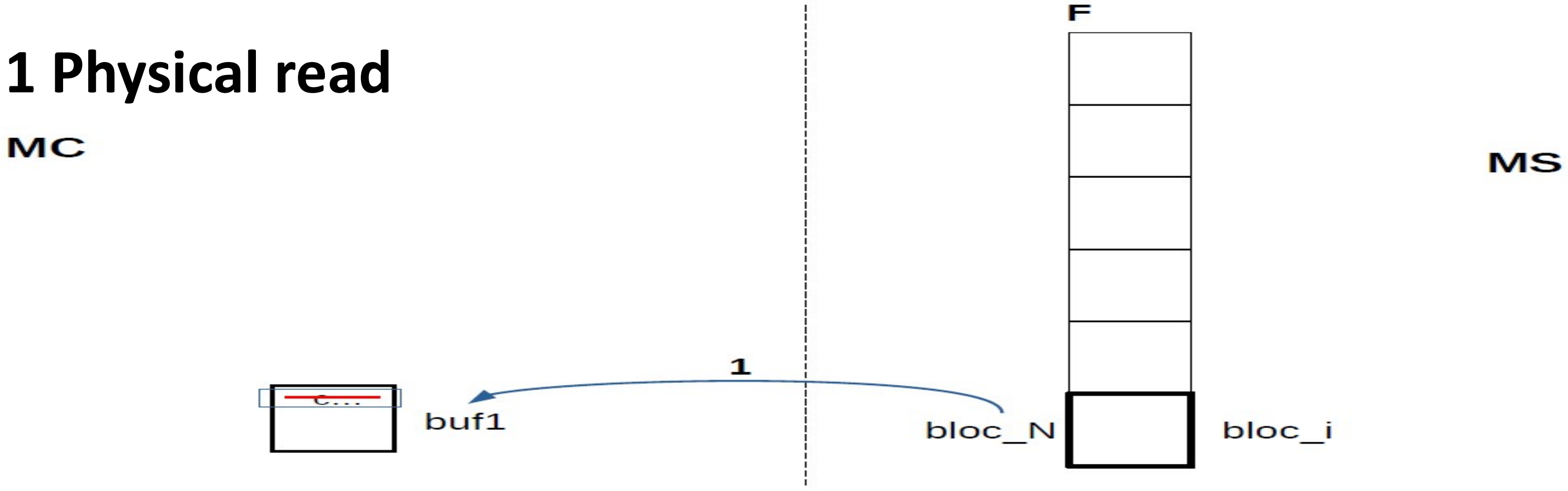
Cost in the best case

(i == N and block N contains only one record.)

- Read block i (or N) into MC ⇒ ***LireDir(F , N , buf1) ;***

- SI (buf1.NB == 1) Aff_Entete(F , 1 , Entete(F , 1) - 1) SINON …. FSI

Cost= **1 Physical read**


F
MC
MS
1
c...
buf1
bloc_N
bloc_i