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

Indexed sequential structures

Presented by : Dr. Daoudi Meroua

Academic year: 2024/2025

Files with Indexes

➔Searching for a record in a sequential file structure is generally costly

- → sequential search
- → binary search in a (very) large file

➔Indexing is a data structure technique that allows efficient retrieval of file records based on certain attributes on which the indexing has been performed.

Files with Indexes

The attribute (or group of attributes) used to search for records is called a "search key."

For example, in a meteorological measurements file:

File of meteorological measurements

< city, date, temperature >

Search examples:

→ Find the record(s) where city = 'DJELFA'

Result:

'DJELFA', '2015-06-23', 21

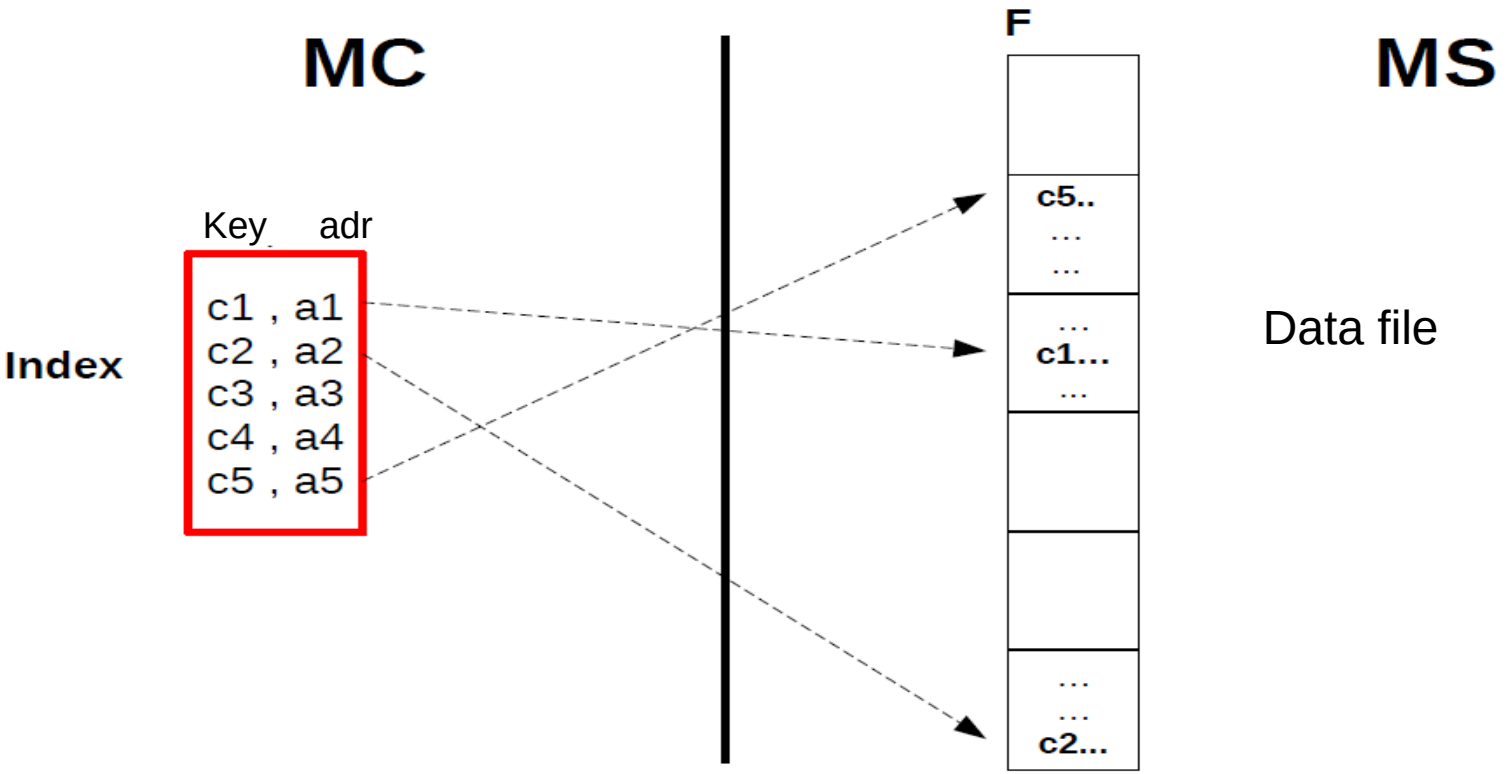
'DJELFA', '2013-10-04', 15

'DJELFA', '2015-06-22', 20

'DJELFA', '2020-07-16', 29

Files with Indexes

An index is an ordered table in main memory (MC), containing, among other things, pairs: < key, address >

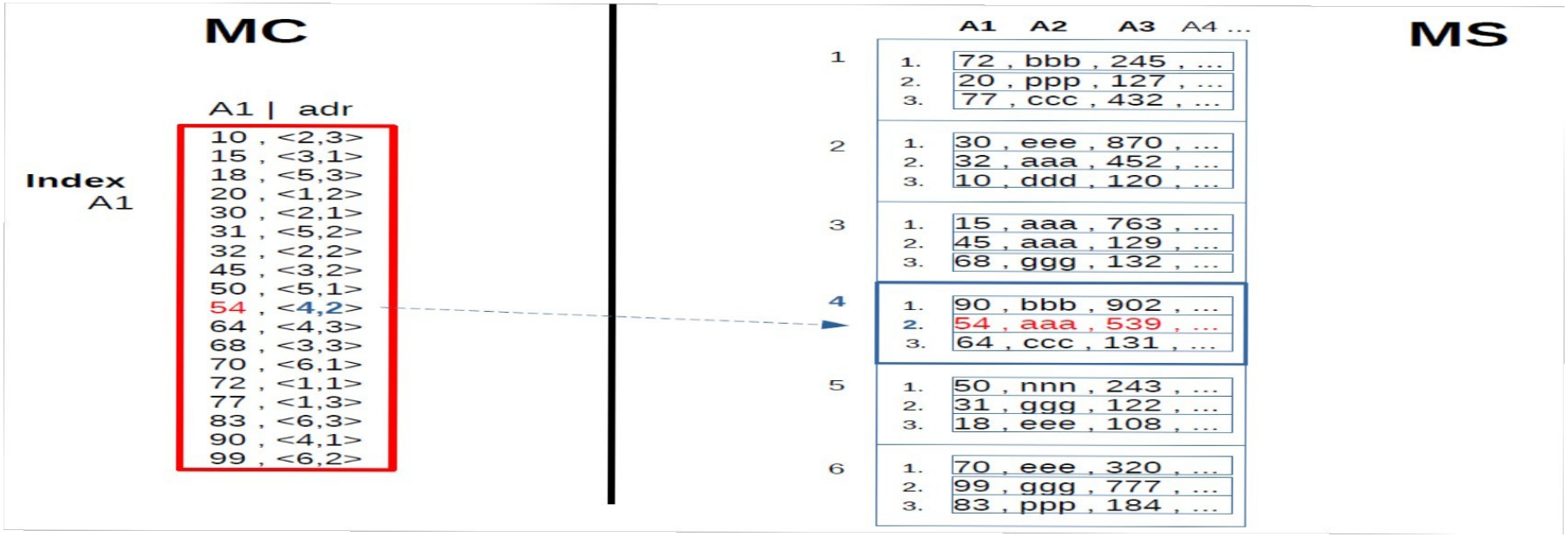


Files with Indexes

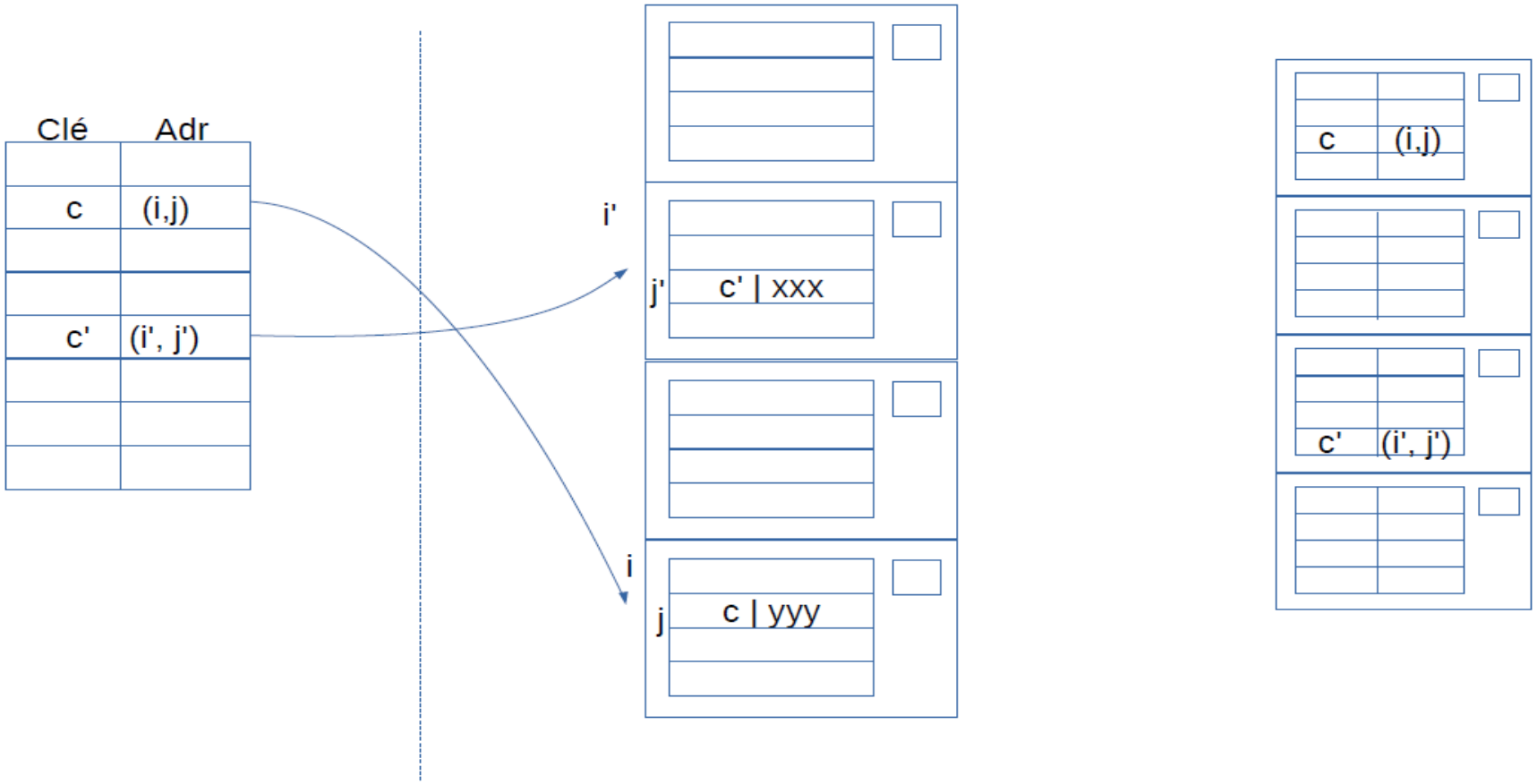
Example: Search for the record with the attribute value A1 = 54

→ Perform a binary search for 54 in the index table in main memory (MC): result adr = <4,2>

→ LireDir(F, 4, buf) and retrieve the record buf.tab[2]



Files with Indexes

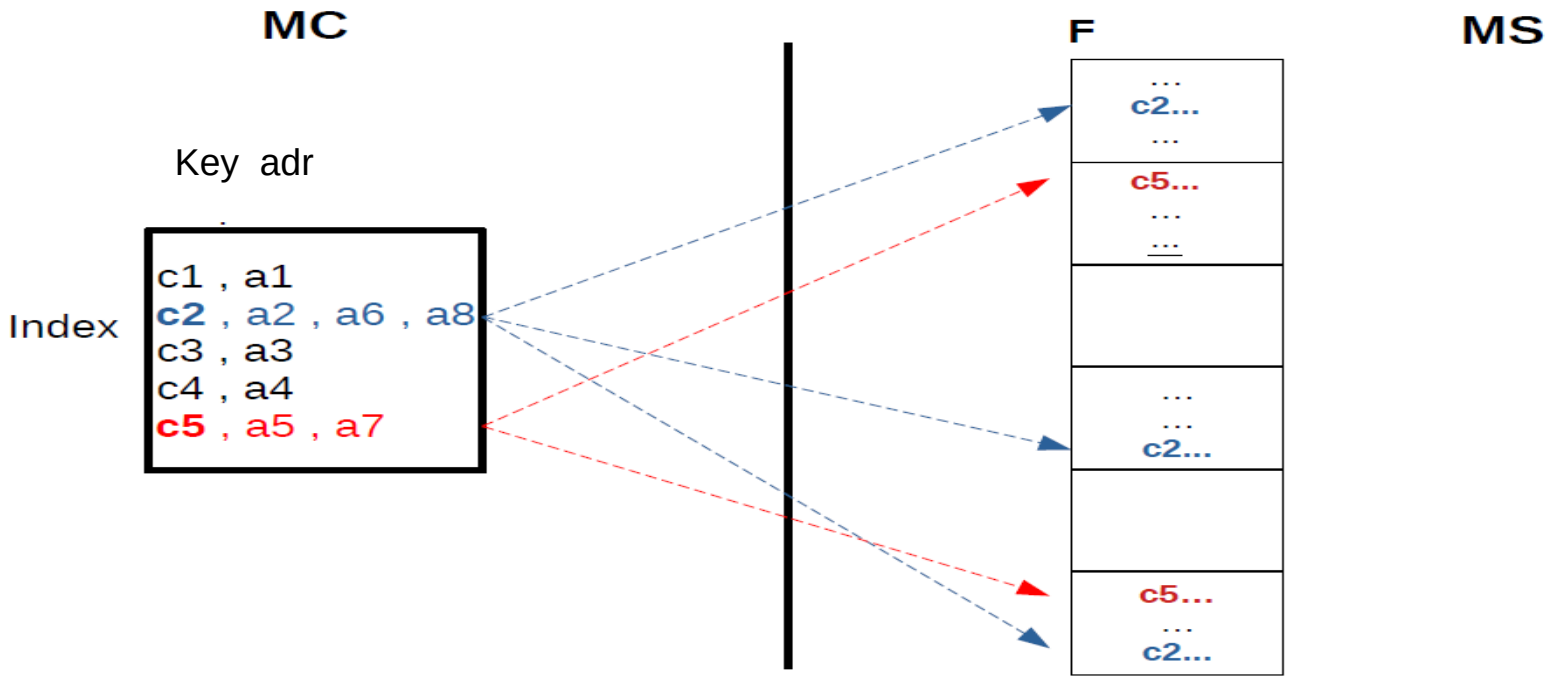


Index table (MC) Data file (MS) Index file (MS)

Files with Indexes

The key can have unique values or not (multiple values).

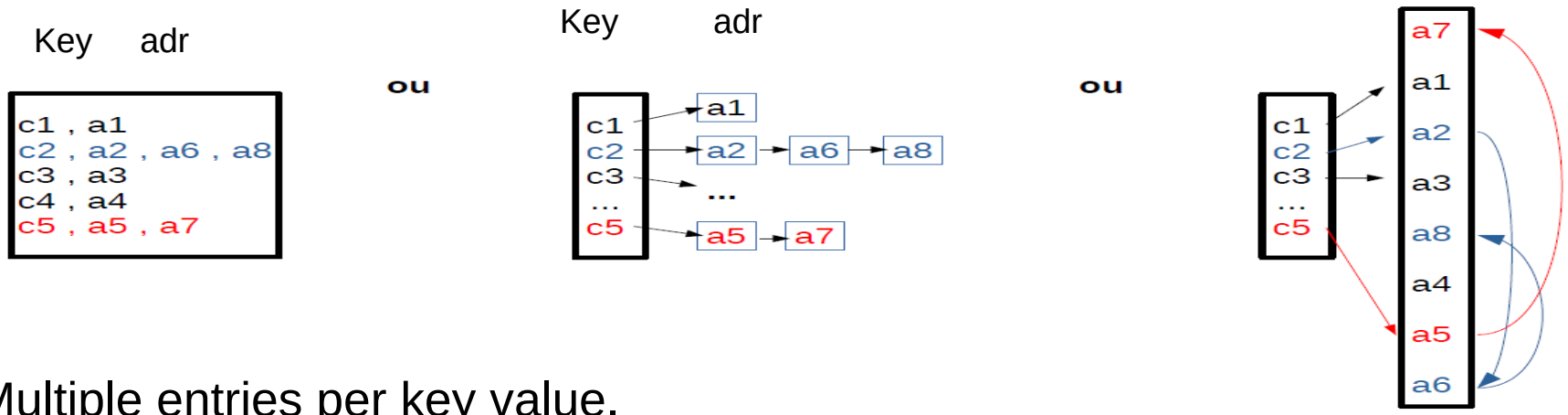
Example of an index on a key attribute with multiple values



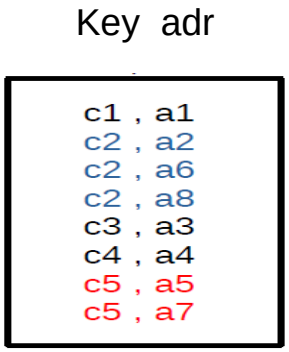
Files with Indexes

Different representations of index tables with multiple values:

1) One entry per key value.



2) Multiple entries per key value.



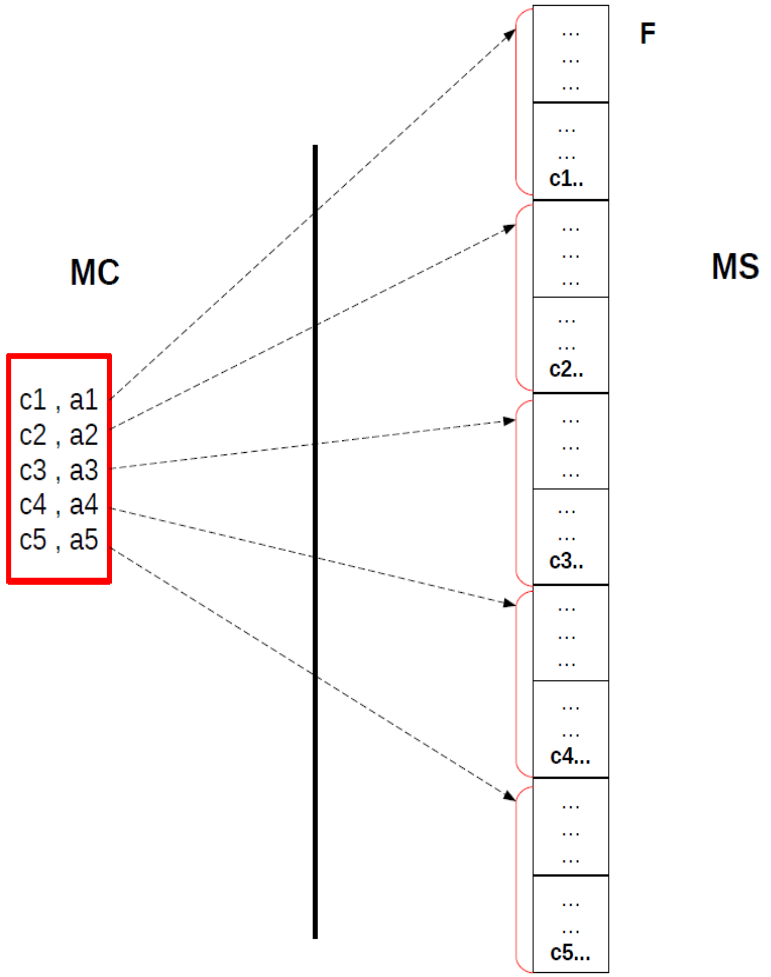
Files with Indexes

The data file can be ordered by the key or not.

If the data file is ordered (by the key attribute)

⇒ Non-dense index (Clustered Index) does not contain all the values of the key attribute.

In this example, each entry in the index table contains the largest key of a group of two consecutive blocks.



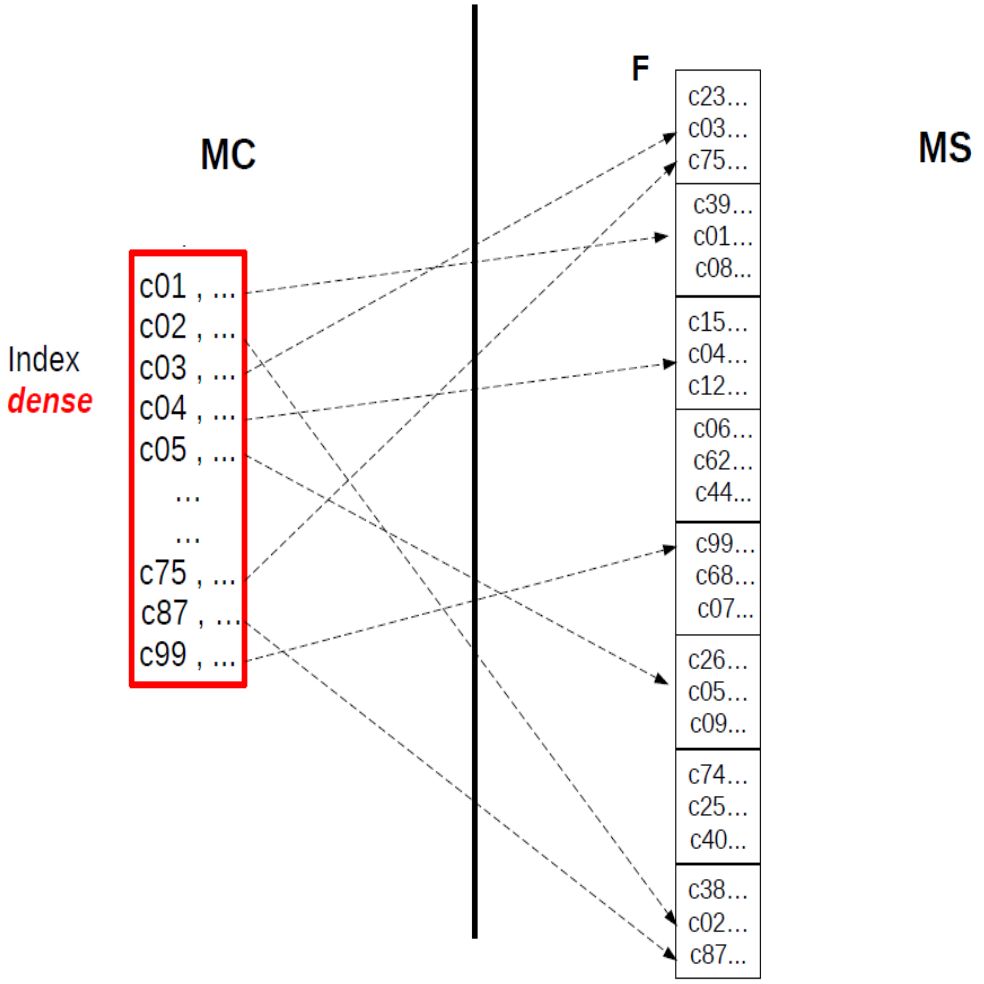
Files with Indexes

The data file can be ordered by the key or not.

2) If the data file is not ordered (by the key attribute)

⇒ Dense index (Non-Clustered Index)

contains all the values of the key attribute.



Files with Indexes : basic operations

Record Search

Search in the index in main memory (MC), then access the data file.

- Exact query (key = value) → binary search for the exact value.
- Interval query (key ∈ [a, b]) → binary search for 'a' + sequential search for the following values up to 'b'.

Insertion / Deletion of Records

Insertions/deletions of records in the data file and, if necessary, update the index in MC.

Case of Ordered File:

More efficient interval query.
Deletion is more costly.

Files with Indexes : : basic operations

Example: Insertion in T~OF with a dense index and unique key values.

```
Type Tbloc = Struct
   tab : tableau[ b ] de typeEnreg
   NB : entier
Fin
Var F : FICHIER de Tbloc BUFFER buf ENTETE ( entier )
Index : tableau [ MaxIndex ] de Tcouple
NbE : entier // number of elements in the index table (== number of records in the file F)
```

```
Tcouple = Struct
    cle : typeqlq ;
    numBlc , depl : entier
Fin
```

Ins(e:TypeEnreg)

Rech(e.cle , trouv , k) *// Search (binary) in the index table*

SI (Non trouv)

// Insertion at the end of the data file ...

OUVRIR(F, « donnees.dat » , 'A')

i ← *Entete(F , 1)*

LireDir(F , i , buf)

Files with Indexes

```
SI ( buf.NB < b ) buf.NB++ ; j ← buf.NB ; buf.tab[ j ] ← e
   EcrireDir( F , i , buf )
SINON
    i++ ; j ← 1 ;
    buf.NB ← 1 ;
    buf.tab[ j ] ← e
    Aff_entete( F, 1, i ) ; EcrireDir( F , i , buf )
FSI
FERMER( F )
// Insertion in the index table ...
NbE++ ; m ← NbE
TQ ( m > k )
      Index[ m ] ← Index[ m-1 ] ;
      m–
 FTQ
 Index[ k ] ← < e.c , i , j > // clé, numBlc, depl
 FSI
```

Files with Indexes

Same example but with non-unique key values.

```
Type Tcouple = Struct
    cle : typeqlq ;
    tete : ptr(maillon)
Fin
Var Index : tableau [ MaxIndex ] de Tcouple

maillon = struct
    val : struct (numblc , depl : entier) ;
    adr : ptr(maillon)
Fin
```

Ins(e:TypeEnreg)

// Insertion at the end of the data file ...

OUVRIR(F, « donnees.dat » , 'A')

i ← *Entete(F , 1)*

LireDir(F , i , buf)

SI (buf.NB < b) buf.NB++ ; **j ← buf.NB** ; buf.tab[j] ← e

EcrireDir(F , i , buf)

SINON

i++ ; j ← 1 ; buf.NB ← 1 ; buf.tab[j] ← e

Aff_entete(F, 1, i) ; *EcrireDir(F , i , buf)*

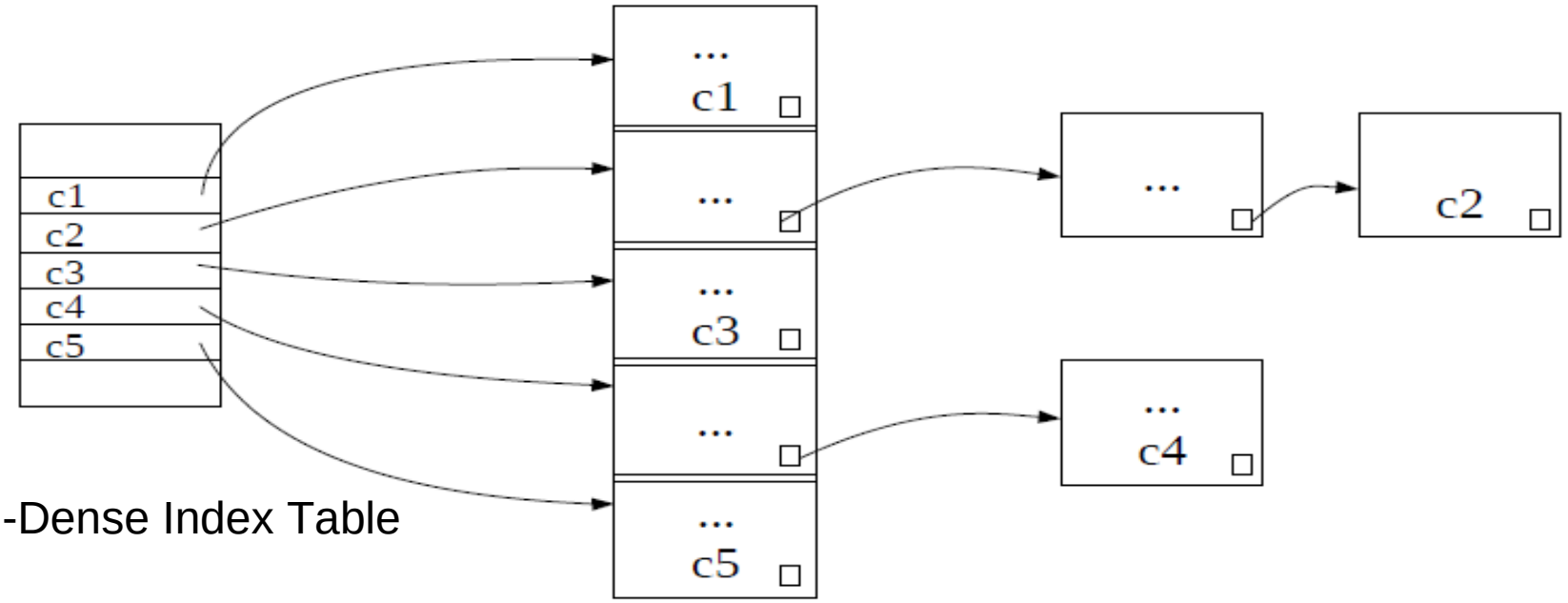
FSI

Files with Indexes

```
FERMER( F )
// Insertion in the index table ...
Rech( e.cle , trouv , k )
SI ( trouv ) // Add a link <i, j> to the list index[k].head
        Allouer( p ) ;
        Affval( p , < i , j > ) ;
        Affadr( p , Index[ k ].tete ) ;
        Index[ k ].tete = p
SINON // Insert a new entry <key, <i, j>> in the index at position k.
        NbE++ ;
        m ← NbE ;
        Allouer(p) ;
        Affval(p, < i , j >) ;
        Affadr(p,nil)
TQ ( m > k ) Index[ m ] ← Index[ m-1 ] ; m-- FTQ
Index[ k ] ← < e.c , p > // key = e.c, head = p
FSI
```

Files with Indexes

Management of an Overflow Area



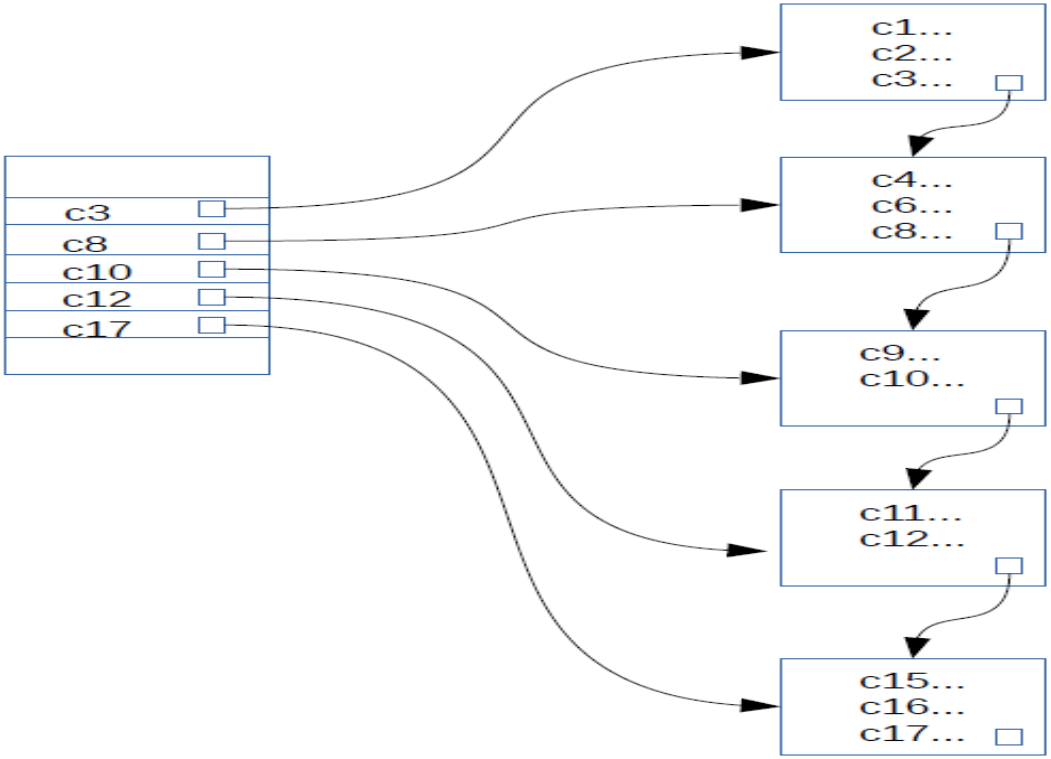
Non-Dense Index Table

Primary Area Data File

Overflow Area Data File

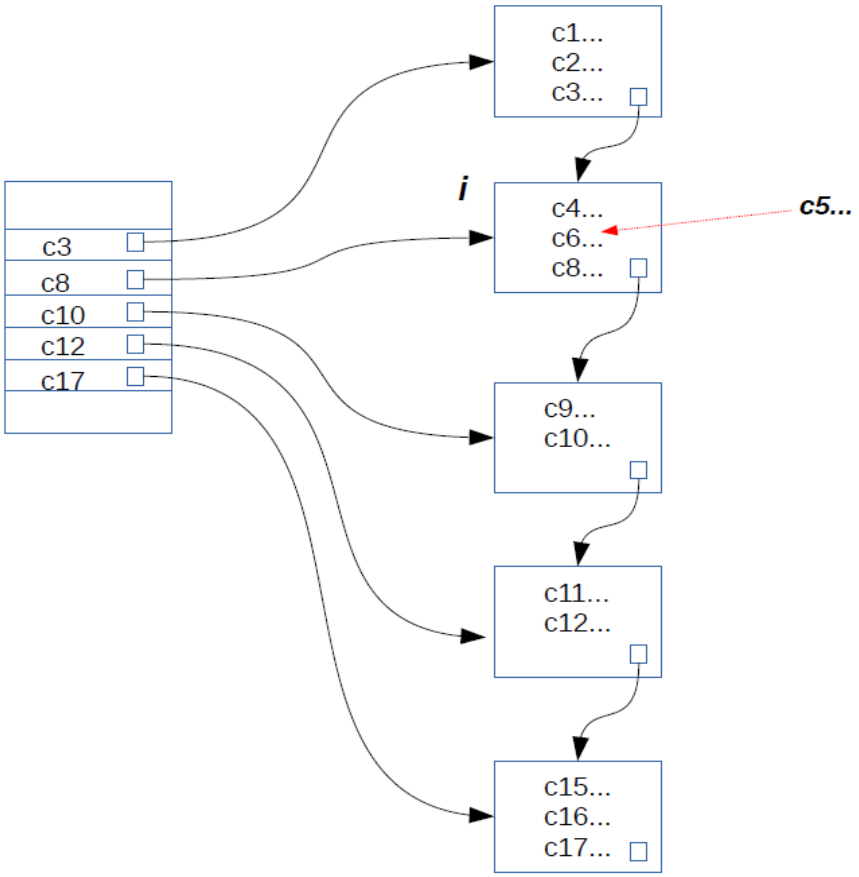
Files with Indexes

Example: Index for LOF File
(no inter-block offsets and no overflow area)



Files with Indexes

Exemple :LOF File / Insertion



The insertion of c5 causes the overflow of block i:

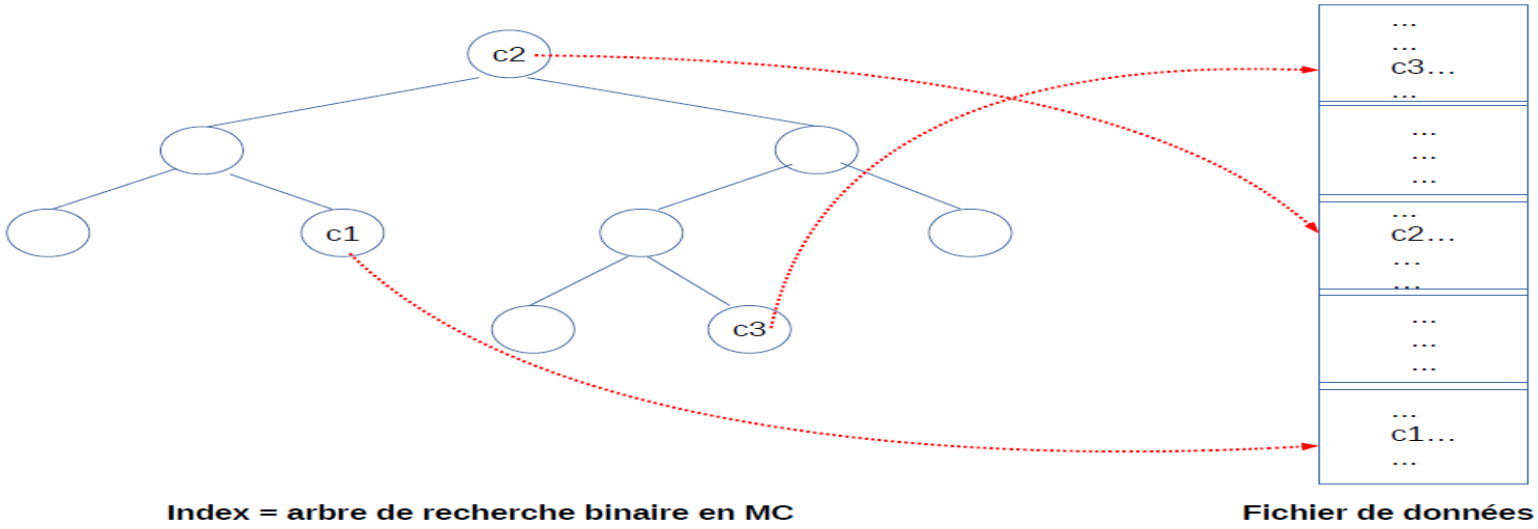
Add a new block → i'

Split the content of i into two halves

Update the index by inserting a new entry for block i'

Files with Indexes

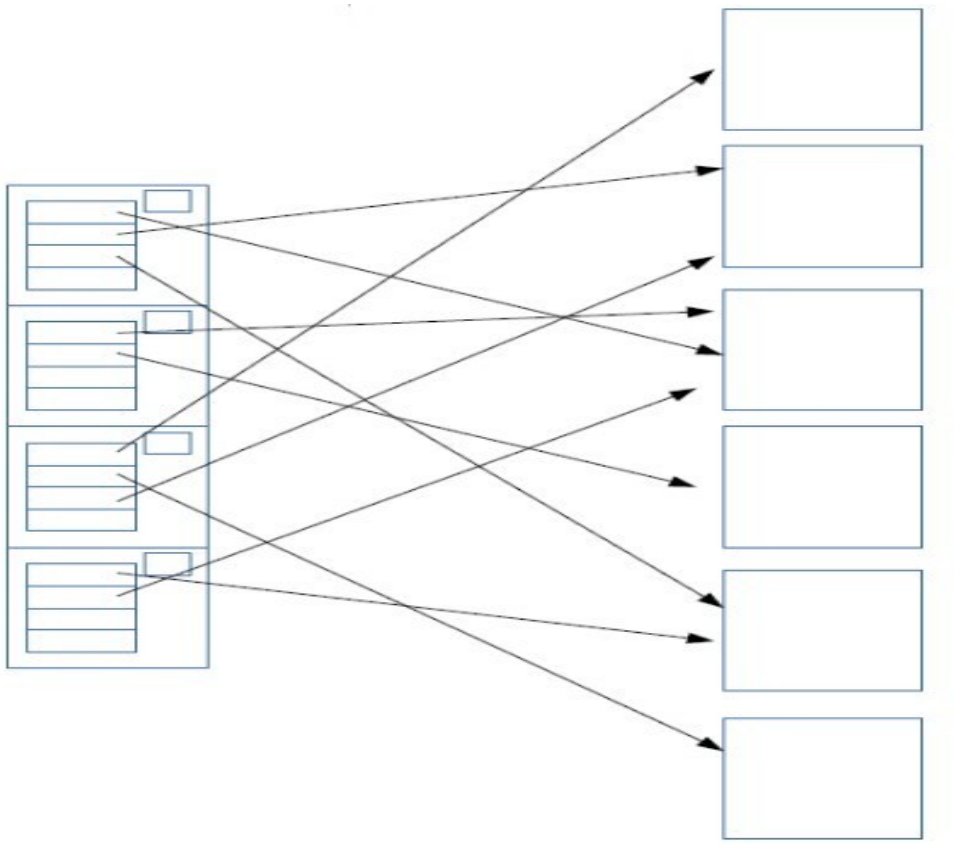
Index in Main Memory in the form of BST




```
Type Tnoeud = struct
        cle : typeqlq
        numBlc , depl : entier
        fg , fd : ptr(Tnoeud)
Fin
```

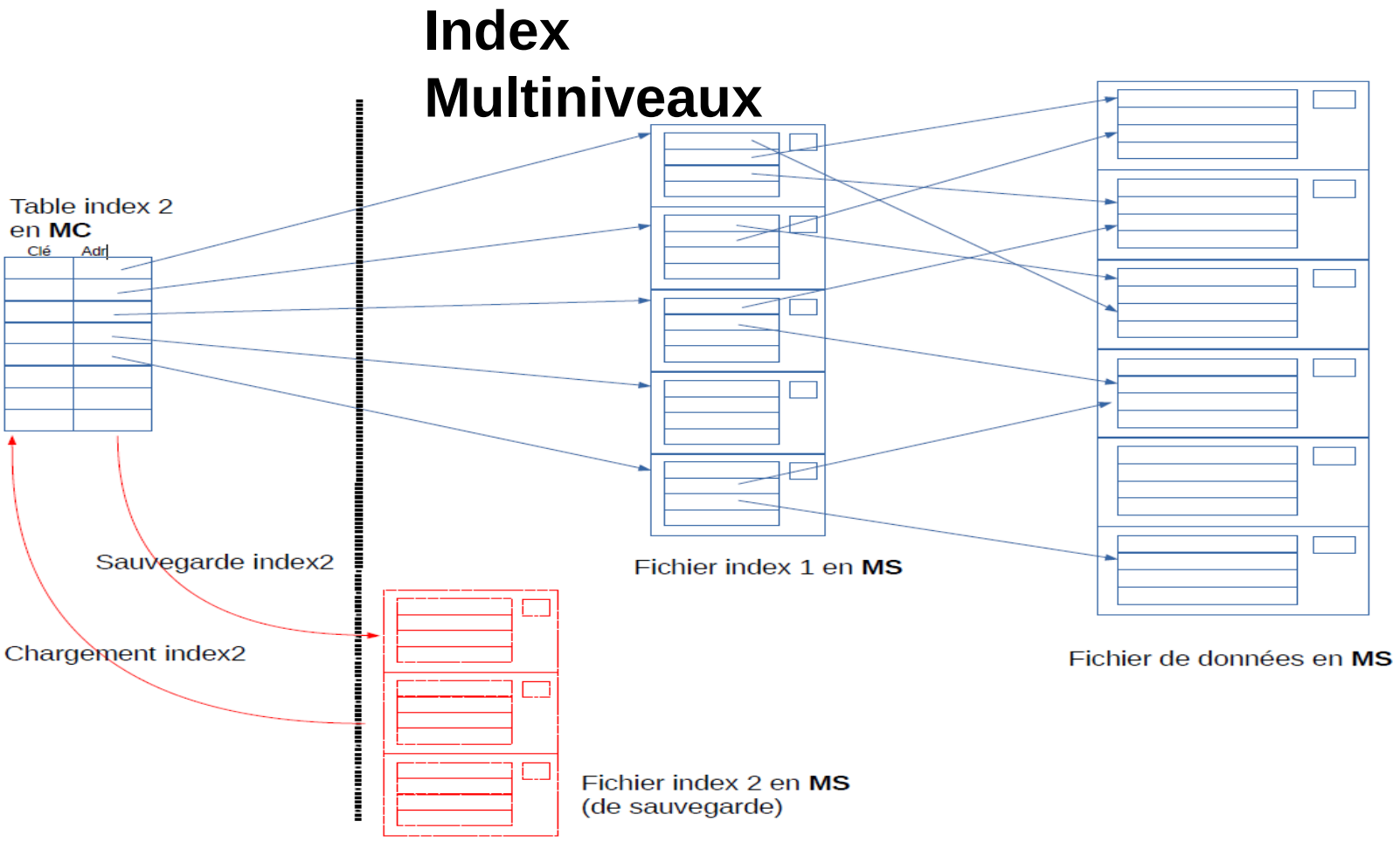
Files with Indexes : Large Index

Index in central memory in the form of an ordered file with contiguous blocks

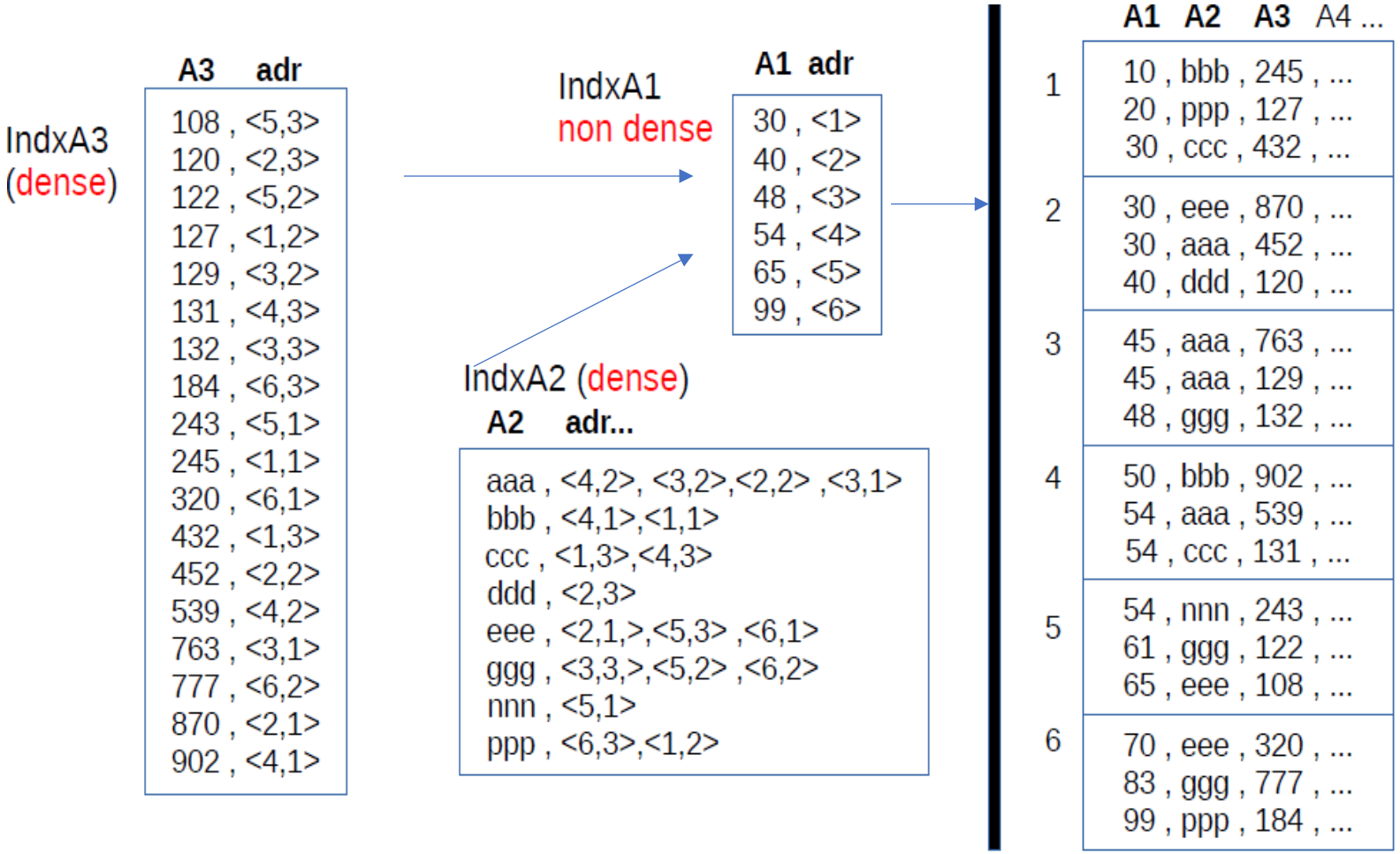


file in main memory (MC)

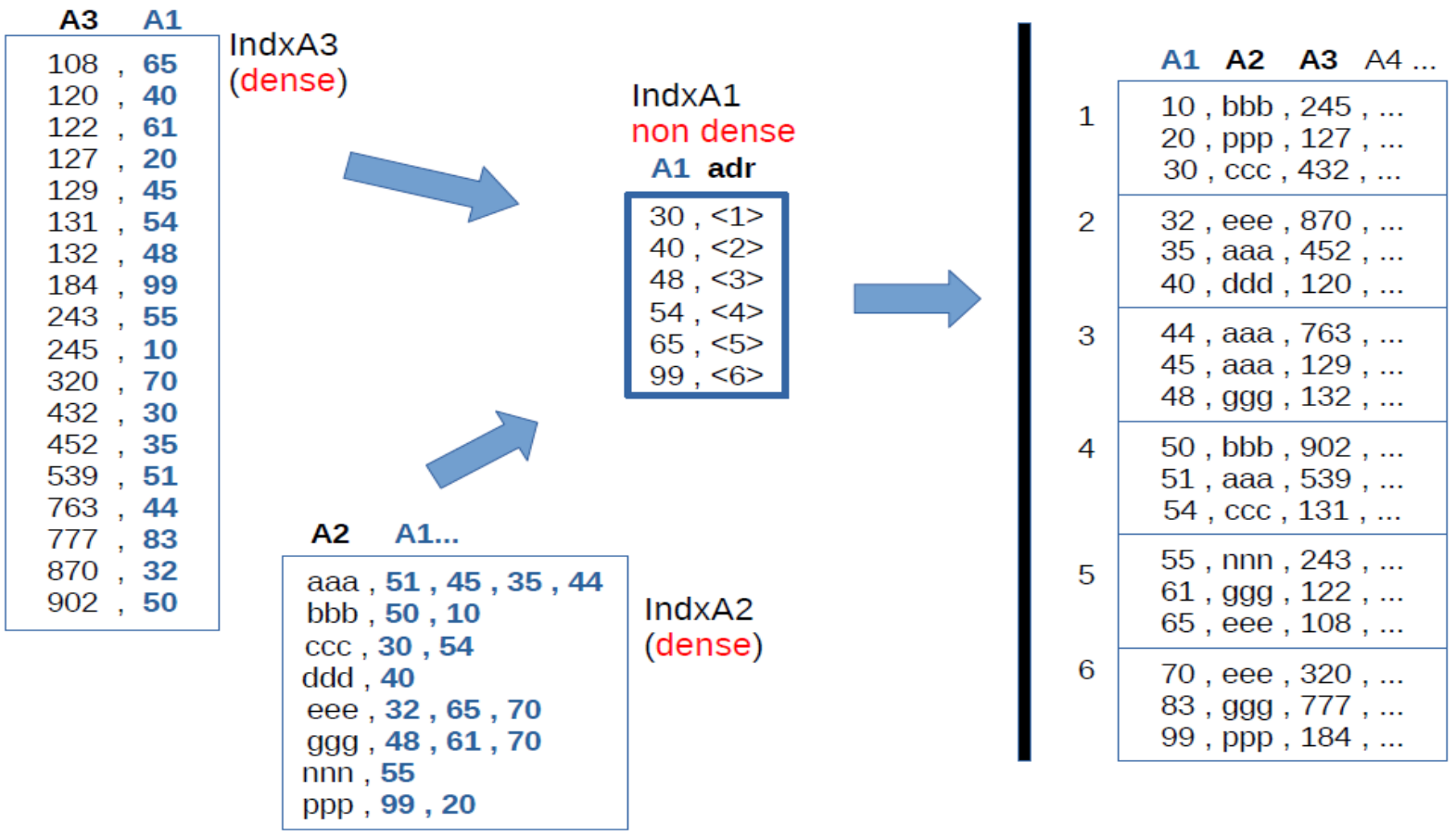
Files with Indexes : Large Index



Files with Indexes : Multi-Key Query



Files with Indexes : Multi-Key Query



Files with Indexes : Multi-Key Query

Find all records where the value of X = vx AND the value of Y = vy AND …" with X, Y, ... as 'secondary keys' (For each secondary key, there is a corresponding secondary index):

- Using the secondary index X, find the list Lx of primary keys associated with the value vx.
- (Repeat the same action for each secondary key mentioned in the query…)
- Perform the intersection of the primary key lists Lx, Ly, ... to find the primary keys associated with each secondary key value mentioned in the query.
- Use the primary index to retrieve the records from the data file (by first sorting the sequence of block numbers before performing the physical transfers).

Files with Indexes : Multi-Key Query

If we are searching for all records where A2 = ‘eee’ and A3 = 870, the multi-key query algorithm will proceed as follows:

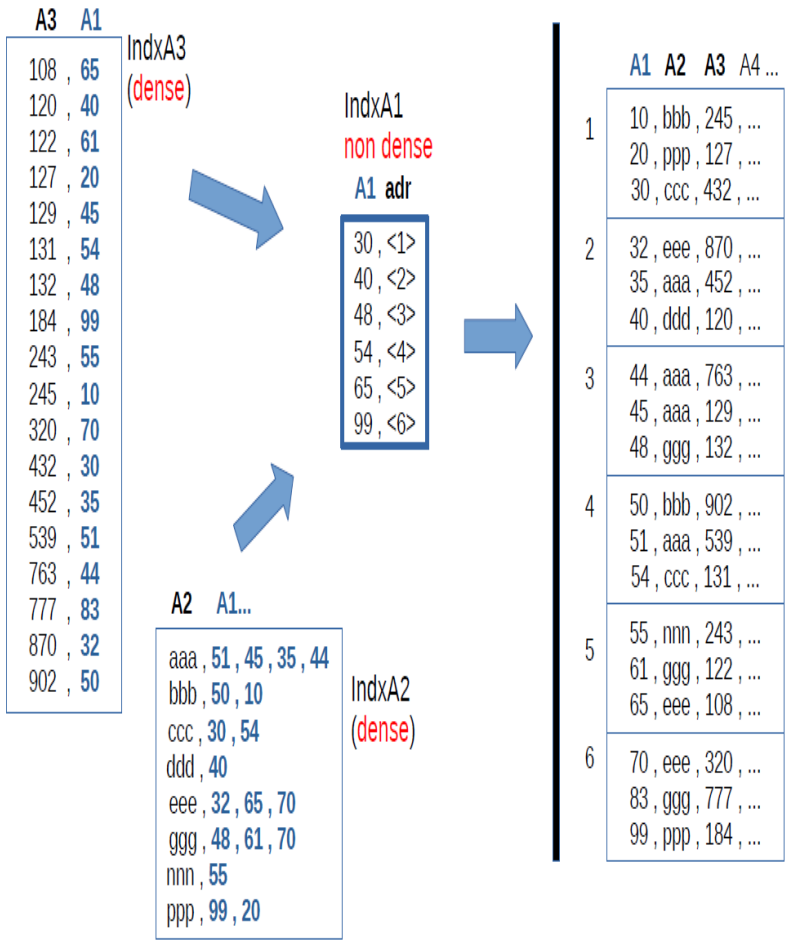
a. Search for ‘eee’ in the index IndA2 → result: LA2 = [32, 65, 70]

b. Search for 870 in the index IndA3 → result: LA3 = [32]

c. Intersection of LA2 and LA3 → result: Final L: [32]

d. Search for 32 in IndA1 → result: block number <2>

e. ReadDir(F, 2, buf) and retrieve the record “<32, bbb, 870, …>”



Files with Indexes : Multi-Key Query

Insertion of a record < c, vx, vy, ... >

- Search for c in the primary index → ip: the index where this key should be inserted (binary search).
- Insert the record into the data file → adr: the address where the record has been inserted.
- Insert in the primary index, at position ip, the entry < c, adr > if it is a dense index, or update the entry at index ip if it is a non-dense index.
- Search for the value vx in the secondary index X.
- If vx exists, add c to the list pointed to by vx.
- If vx does not exist, insert vx in the secondary index X.
- → In this case, the new entry vx will point to a list formed by a single primary key (c).
- Repeat step 4) for each remaining secondary key (vy, ...).

Files with Indexes : Multi-Key Query

Deletion of a record < c, vx, vy, ... >

- To logically delete a record with primary key c, it is sufficient to set a deletion bit (or character) in the data file or in the primary index table for the entry c.

- To physically delete a record with primary key c, you must first physically remove the record from the data file, and then update the primary index table either by deleting the entry related to c (in the case of a dense index) or by modifying the key and/or address of the representative of the group to which the deleted record belongs (in the case of a non-dense index).

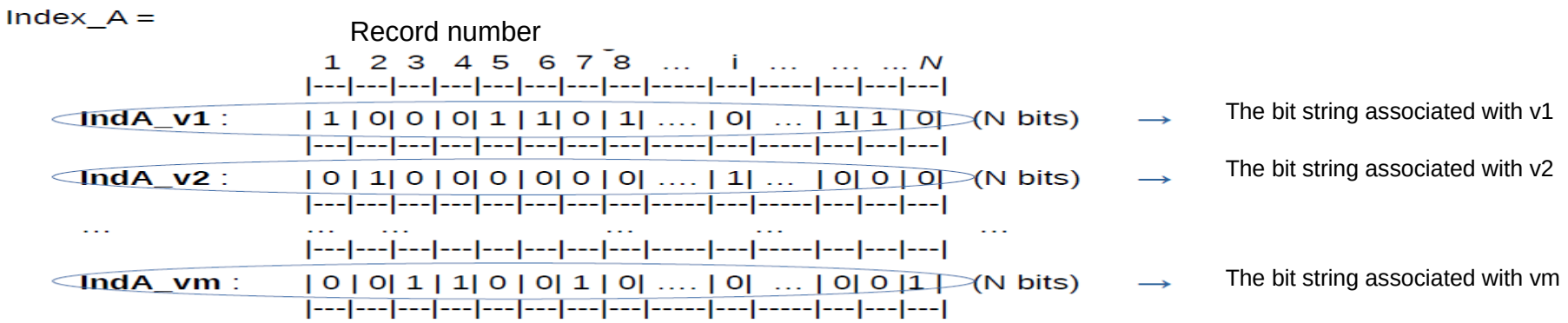
In both types of deletion (logical or physical), it is not necessary to update the secondary indexes.

Index Bitmap

A bitmap index on an attribute A (formed by m different values: v1, v2, … vm) consists of m binary strings, each with N bits (IndA_v1, IndA_v2, ... IndA_vm):

Each string IndA_vj is associated with the value vj of attribute A.

- If (IndA_vj[k] = 1), then in record number k, attribute A equals vj.
- If (IndA_vj[k] = 0), then in record number k, attribute A is different from vj.



Examples:

A = v2 in record number 2 and record number i of the data file.

A = v1 in records number 1, 5, 6, 8, … N-2 and N-1.

Files with Indexes : Index Bitmap

Bitmap indexes can be useful for attributes with low cardinality (e.g., < 20 distinct values).

The different bit strings can be loaded into main memory (MC) independently of each other.

They are primarily used for multi-key queries on attributes with low cardinality.

Example: “Find records where A = v2 and B = w4.

Index_A =

	1	2	3	4	5	6	7	8	...	i	...	...	N
IndA_v1 :	1	0	0	0	0	1	0	1	....	0	...	1	1	0
IndA_v2 :	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**....**	**1**	**...**	**0**	**0**	**0**
IndA_v3 :	0	0	1	1	0	0	0	0	....	0	...	0	0	1

Cardinality of A = 3

Index_B =

	1	2	3	4	5	6	7	8	...	i	...	...	N
IndB_w1 :	0	0	1	0	0	0	0	1	....	0	...	1	0	0
IndB_w2 :	0	1	0	0	0	1	0	0	....	0	...	0	0	1
IndB_w3 :	0	0	0	1	1	0	0	0	....	0	...	0	1	0
IndB_w4 :	**1**	**0**	**0**	**0**	**0**	**0**	**1**	**0**	**....**	**1**	**...**	**0**	**0**	**0**

Cardinality of w = 4

The result of the query is given by the binary operation: (IndA_v2 AND IndB_w4)

→ Records number 7 and number i.