Chapitre 3

Les structures d'index

(Le séquentiel-indexé)

Plan du chapitre

1) Généralités

Définitions , Clé de recherche , Utilisation

2) Accès mono-clé

Index en MC, Index en MS, Index multi-niveaux

2) Accès multi-clés

Index indépendants, Index inversées, Bitmaps

Généralités

La recherche d'un enregistrement dans une structure de fichier séquentielle est généralement coûteuse

→ recherche séquentielle
→ recherche dichotomique dans un fichier (très) volumineux

On appelle '***clé de recherche***' (*Search Key*) l'attribut (ou groupe d'attributs) utilisé pour rechercher les enregistrements :

Fichier de mesures météorologiques
*< **ville** , date, température >*

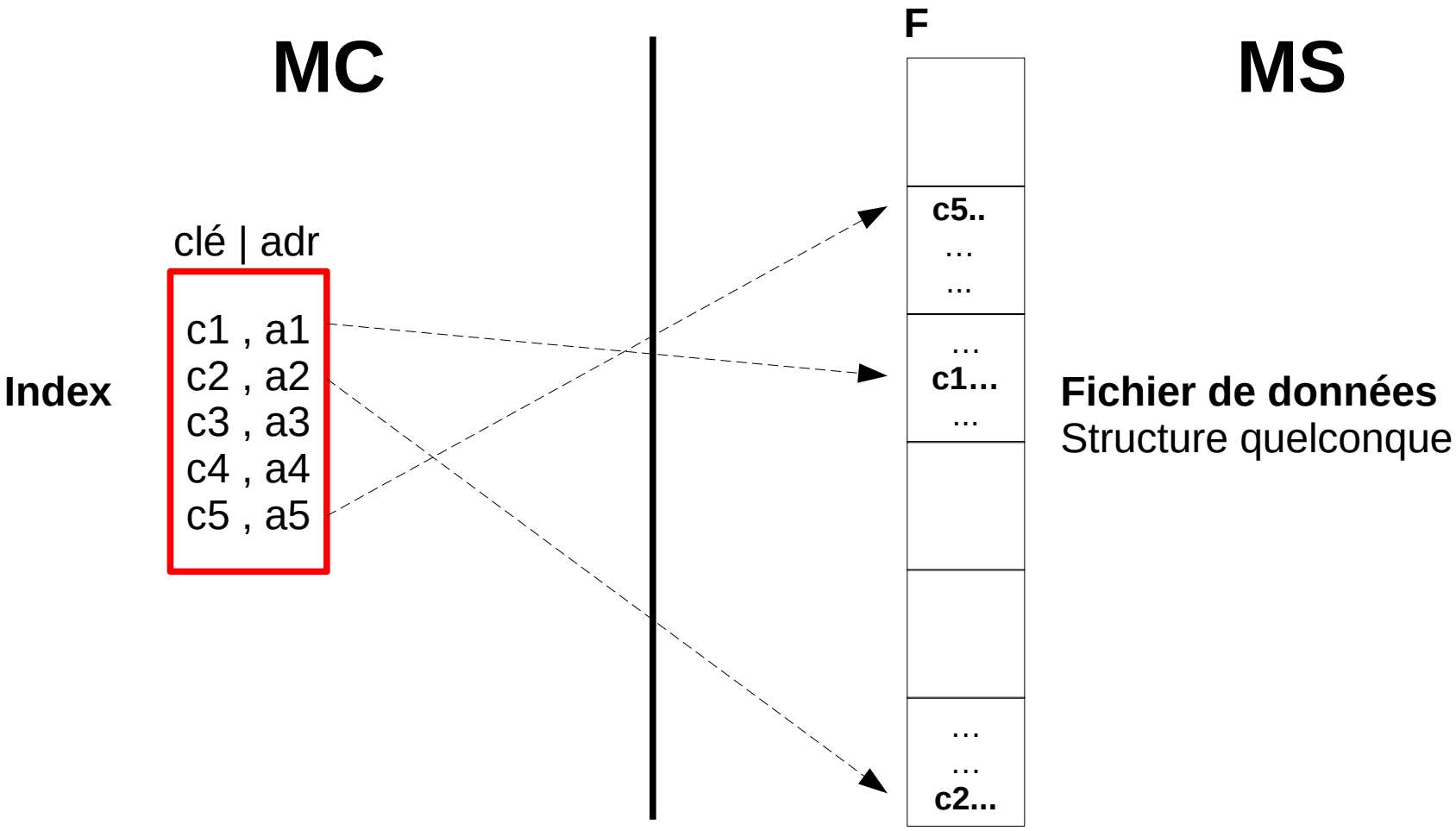
Exemples de recherche :
→ *trouver le (ou les) enregistrement(s) tel que **ville** = 'DJELFA'*
résultat : *'DJELFA' , '2015-06-23' , 21*
'DJELFA' , '2013-10-04' , 15
'DJELFA' , '2015-06-22' , 20
'DJELFA' , '2020-07-16' , 29
...

→ *trouver le (ou les) enregistrement(s) tel que **ville** < 'MZZZ' ET **ville** > 'ME'*
résultat : *'MSILA' , '2011-04-23' , 22*
'MEDEA , '2019-02-04' , 11
'MEFTAH , '2016-08-25' , 32
'MILA' , '2011-04-23' , 14
'MOSTAGANEM' , '2020-03-08' , 19
...

Généralités

Un index est une structure de données (en MC et/ou en MS) permettant de rendre la recherche d'enregistrements plus rapide.

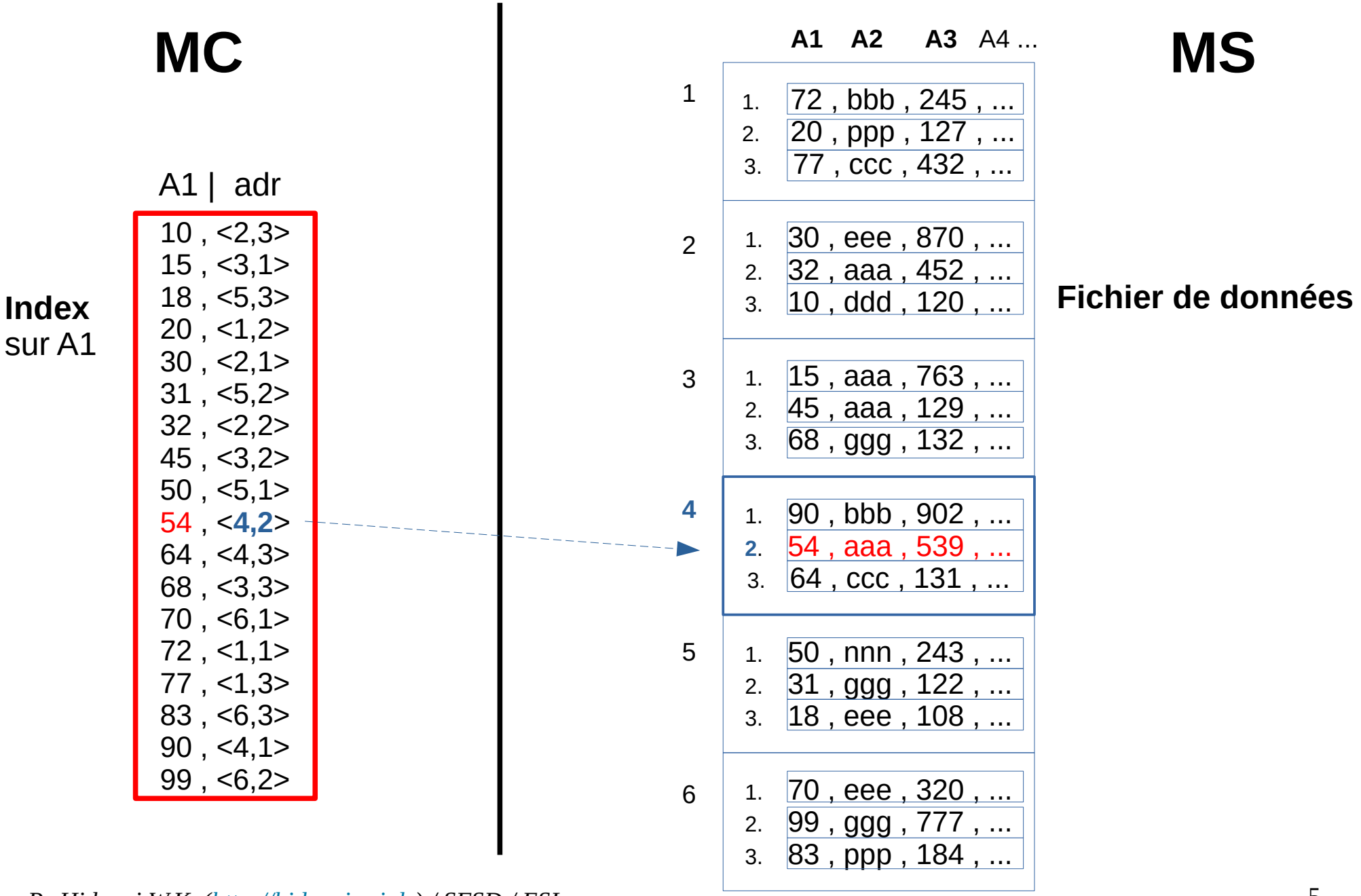
Souvent un ***index*** est une ***table en MC, ordonnée***, contenant entre autre, des couples : < *val_clé* , *adr_enreg* >



Exemple : rechercher l'enregistrement ayant comme valeur d'attribut ***A1 = 54***

→ recherche dichotomique de ***54*** dans la table index en MC : résultat ***adr = <4,2>***

→ ***LireDir(F, 4, buf)*** et récupérer l'enregistrement ***buf.tab[2]***



Utilisation des index en MC

Recherche d'enregistrement

Rechercher dans l'index en MC, ensuite accéder aux fichier de données
Requête exacte (clé = valeur) → recherche dichotomique de la valeur cherchée
Requête à intervalle (clé ∈ [a,b]) → recherche dichotomique de 'a' + les suivants en séquentiel jusqu'à 'b'

Insertion / Suppression d'enregistrements

insertions/suppressions d'enreg dans le fichier de données et éventuellement **m-a-j de l'index en MC**

Création d'un index

- quand le fichier de données est encore vide
  → **créer en MC une table d'index vide**

- à partir d'un fichier de données déjà existant
  → **remplir en MC une table d'index en parcourant le fichier de données**

Sauvegarde d'un index en MS

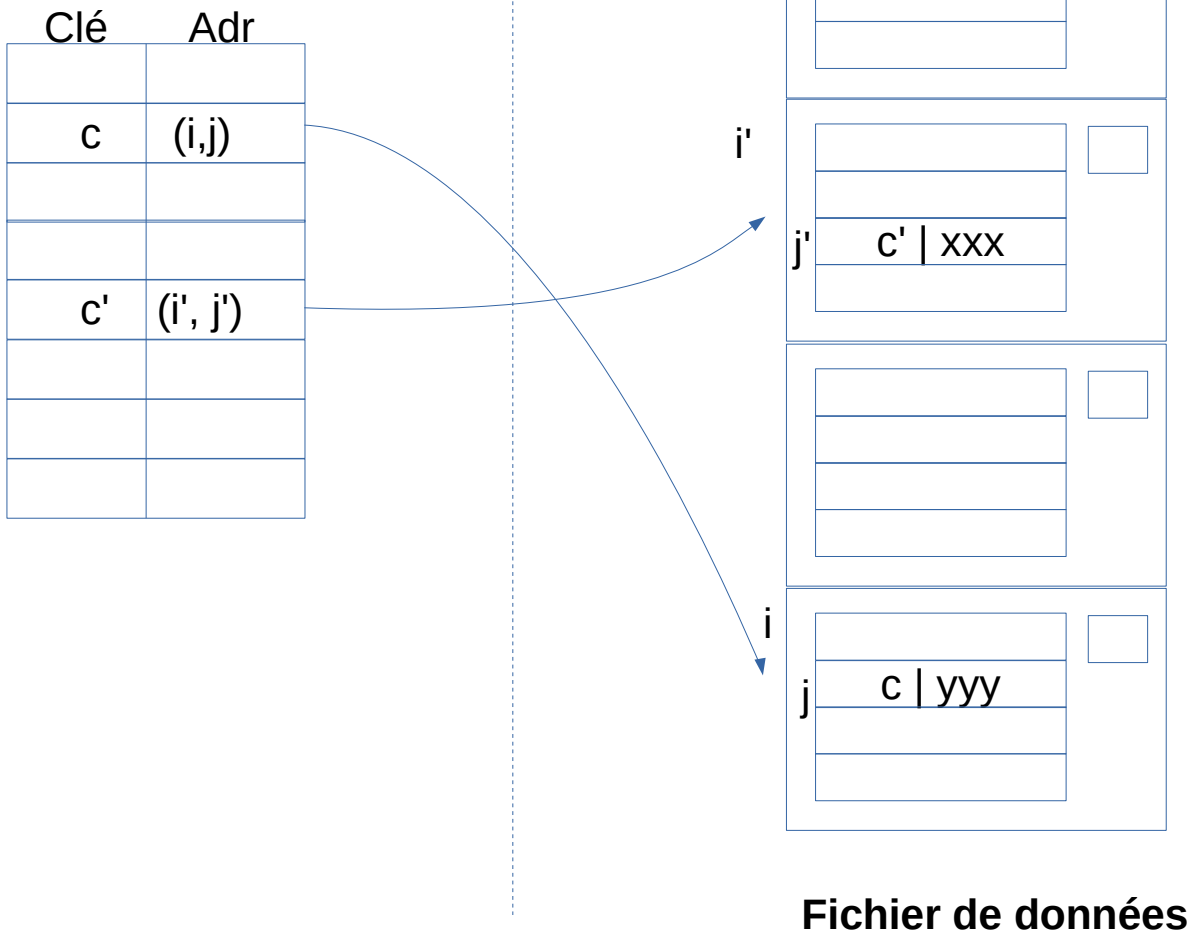
→ sauvegarder le contenu de la table d'index dans un (nouveau) fichier index

Chargement d'un index depuis la MS

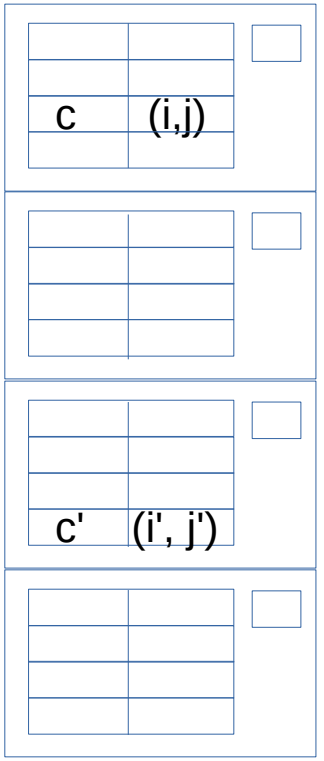
→ charger le contenu du fichier index dans la table d'index en MC

Utilisation des index en MC

Table d'index en MC



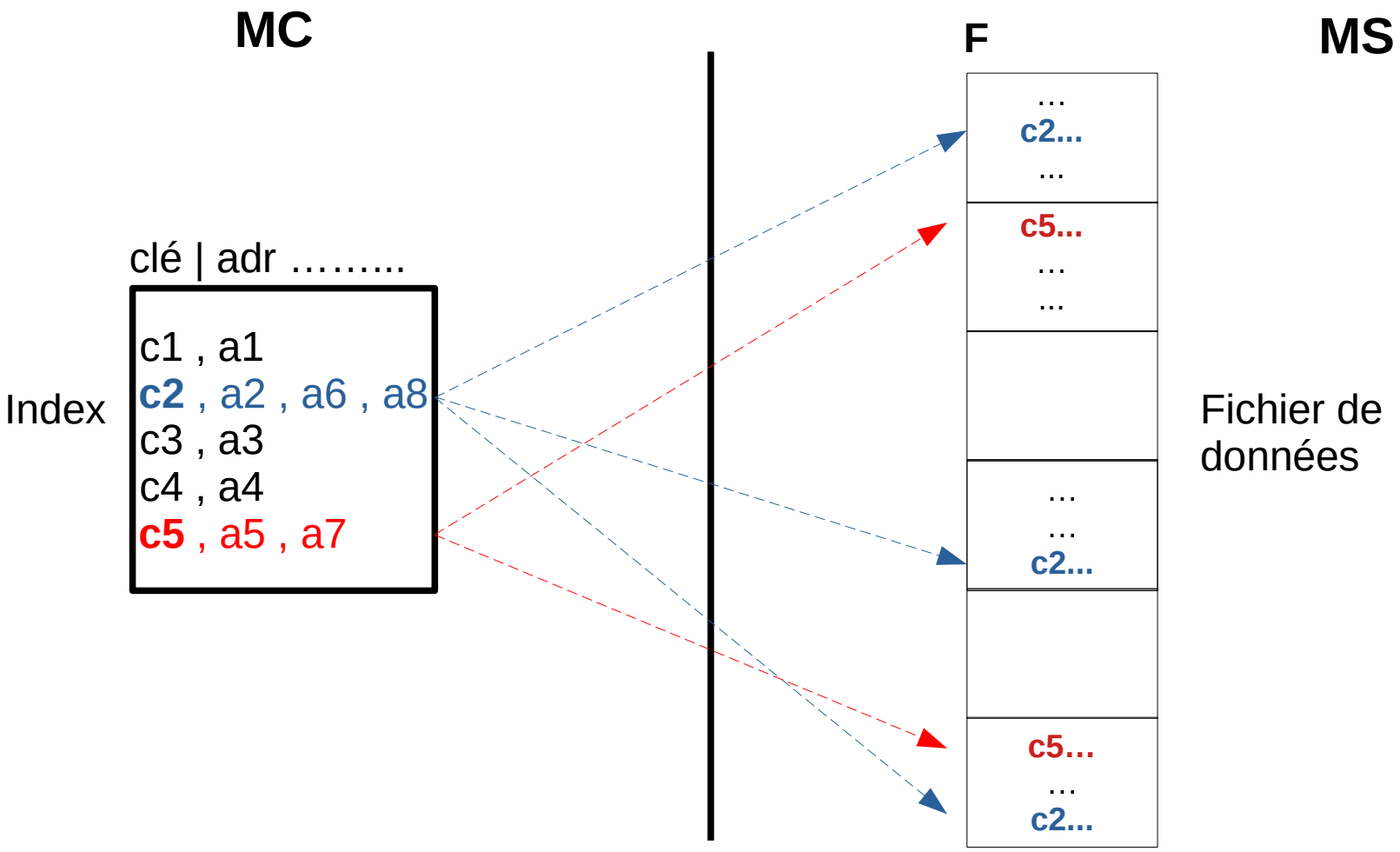
Fichier de données en MS



Fichier index (de sauvegarde) en MS

La clé peut être à valeurs uniques ou non (valeurs multiples)

Exemple d'un index sur un ***attribut clé à valeurs multiples*** :



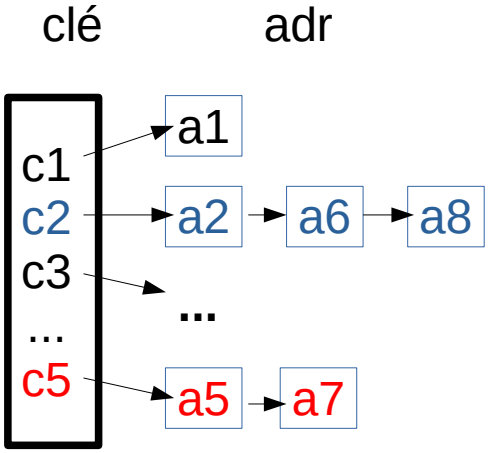
Différentes représentations des tables d'index à valeurs multiples

1) Une entrée par valeur de clé

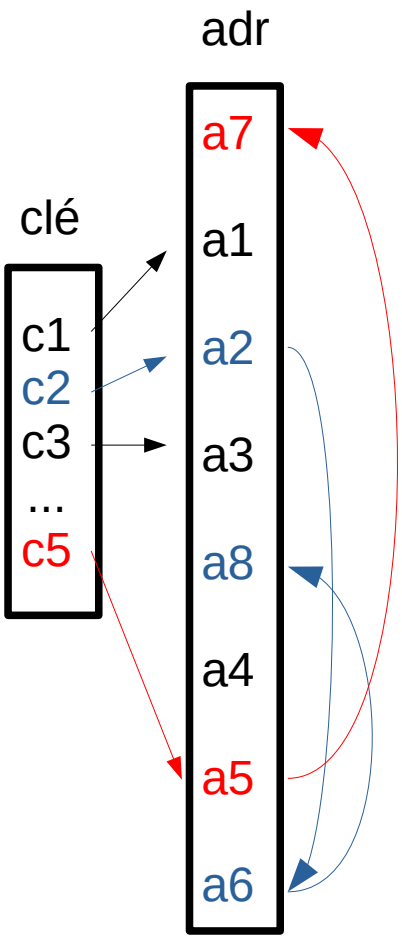
clé | adr ……..

c1 , a1
c2 , a2 , a6 , a8
c3 , a3
c4 , a4
c5 , a5 , a7

ou



ou



2) Plusieurs entrées par valeur de clé

clé | adr

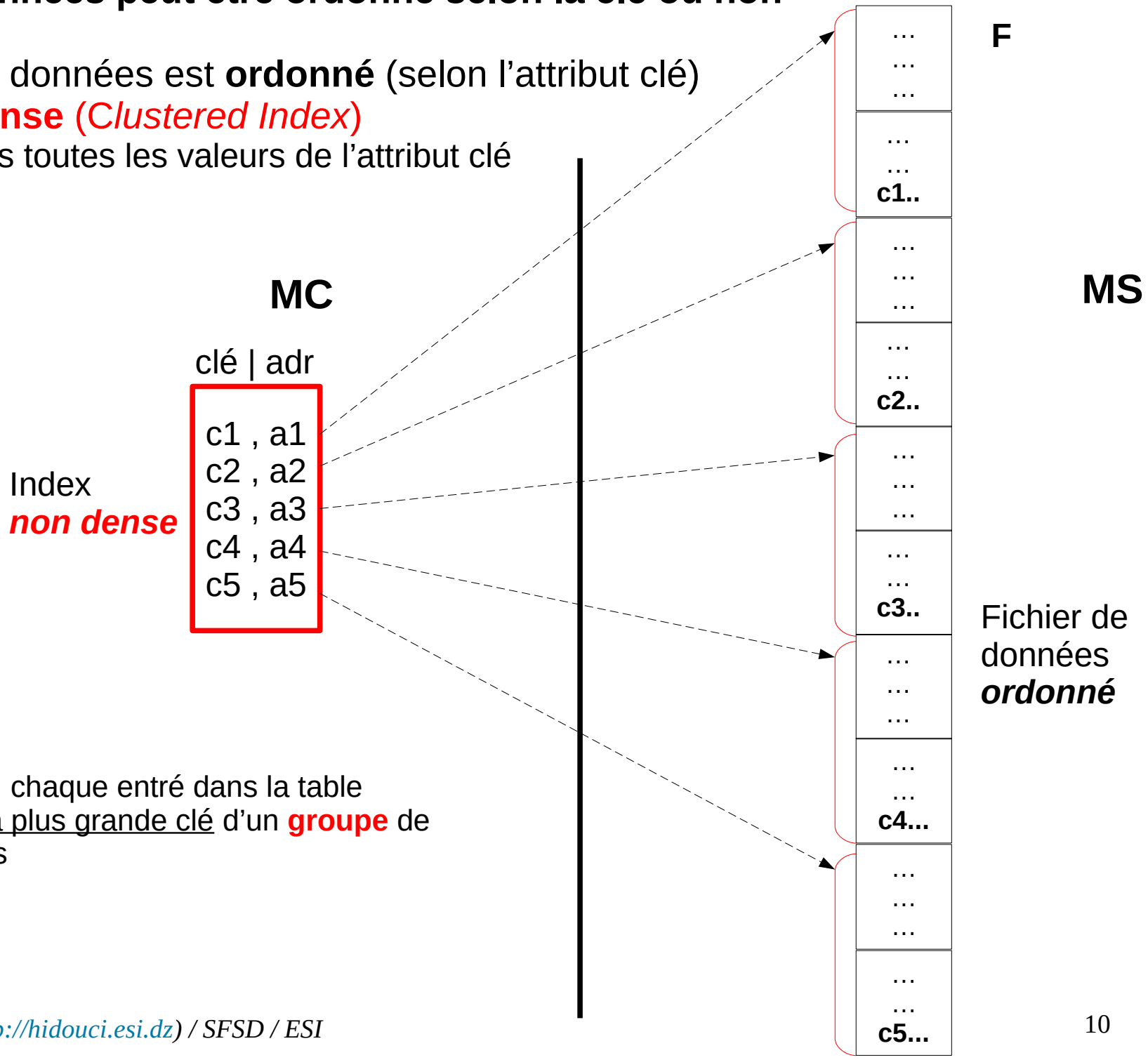
c1 , a1
c2 , a2
c2 , a6
c2 , a8
c3 , a3
c4 , a4
c5 , a5
c5 , a7

Le fichier de données peut être ordonné selon la clé ou non

1) Si le fichier de données est **ordonné** (selon l'attribut clé)

⇒ **Index non dense** (*Clustered Index*)

ne contient pas toutes les valeurs de l'attribut clé



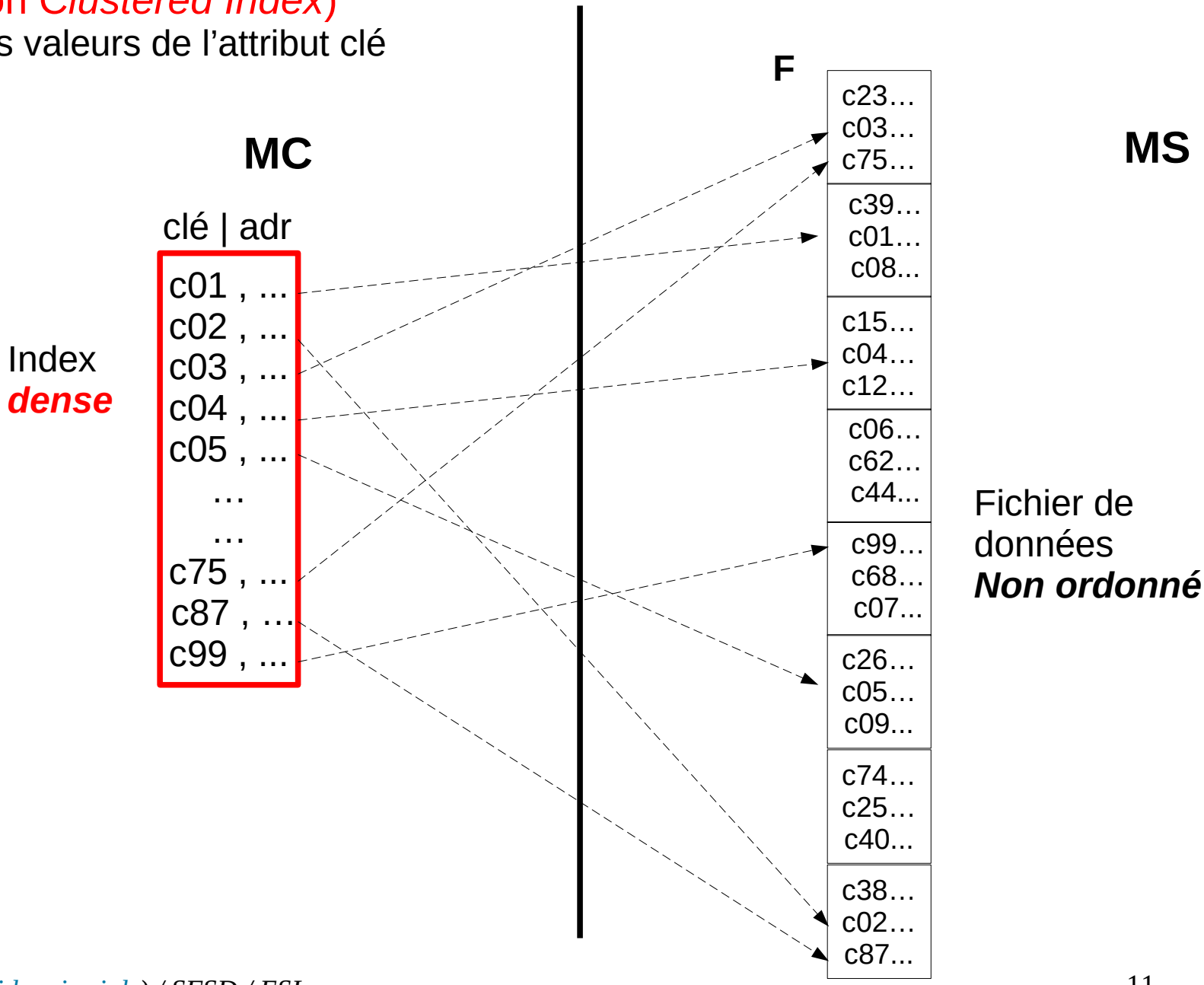
Dans cet exemple, chaque entré dans la table d'index, contient la plus grande clé d'un **groupe** de 2 blocs consécutifs

Le fichier de données peut être ordonné selon la clé ou non

2) Si le fichier de données **n'est pas ordonné** (selon l'attribut clé)

⇒ **Index dense** (non *Clustered Index*)

contient toutes les valeurs de l'attribut clé



Exemple : Insertion dans TOF avec index dense et clés à valeurs uniques

```
Type Tbloc = Struct tab : tableau[ b ] de typeEnreg ;   NB : entier  Fin
       Tcouple = Struct   clé : typeqlq ;   numBlc , depl : entier   Fin

Var    F : FICHIER de Tbloc BUFFER buf ENTETE ( entier )
       Index : tableau [ MaxIndex ] de Tcouple
       NbE : entier        // nombre d'éléments dans la table index ( == nombre d'enreg dans le fichier F)

   Ins( e:TypeEnreg )
      Rech( e.cle , trouv , k )  // Recherche (dichotomique) dans la table index
      SI ( Non trouv )
                  // insertion à la fin du fichier de données ...
                  OUVRIR( F, « donnees.dat » , 'A' )
                  i ← Entete( F , 1 )                    // n° du dernier bloc de F
                  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
                  FERMER( F )
                  // insertion dans la table d'index ...
                  NbE++ ;   m ← NbE
                  TQ  ( m > k )   Index[ m ] ← Index[ m-1 ] ;  m--   FTQ
                  Index[ k ] ← <  e.c  ,  i  ,  j  >    // clé, numBlc, depl
         FSI
```

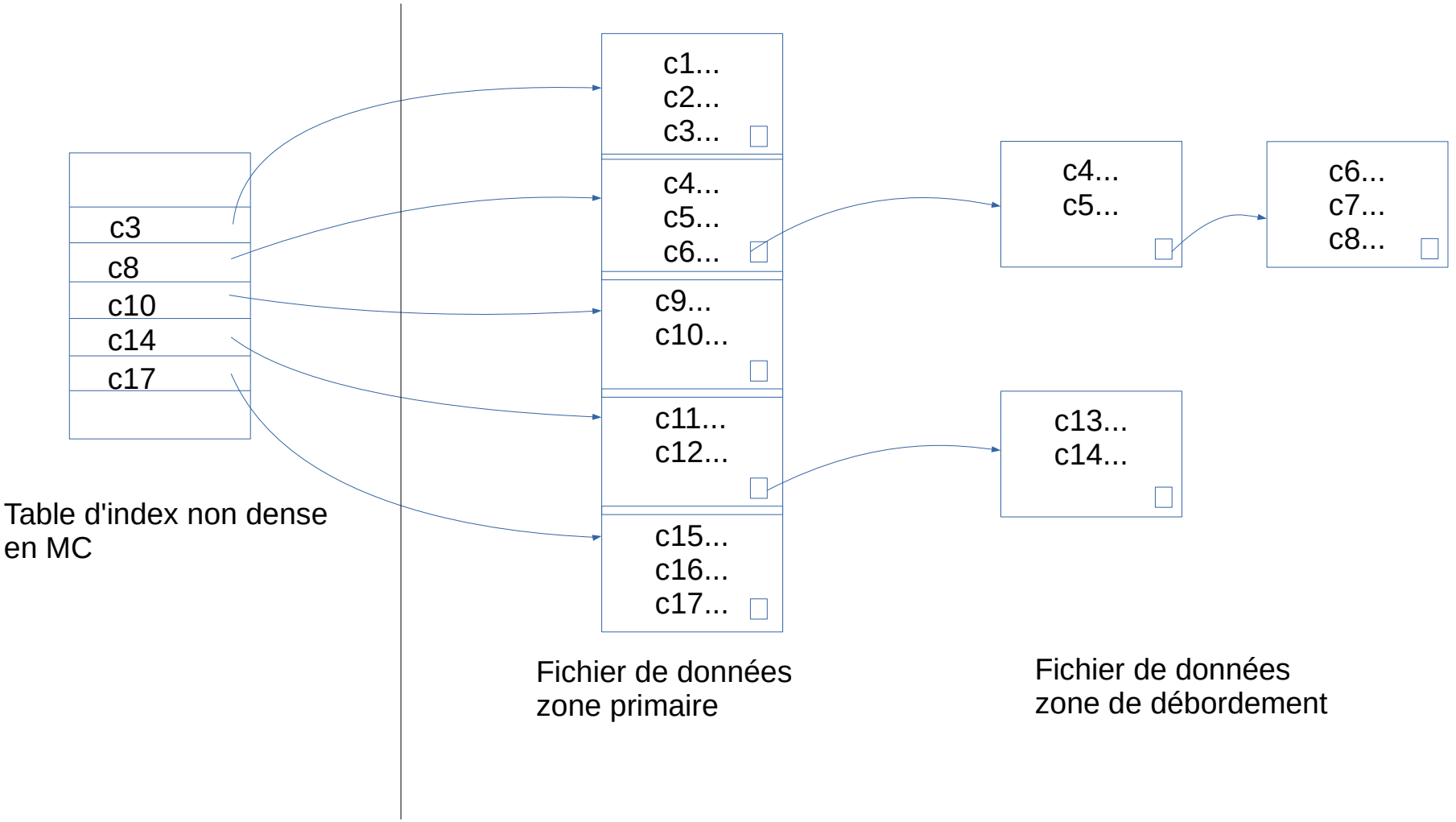
Même exemple mais clés à valeurs non uniques

```
Type  Tcouple = Struct   clé : typeqlq ;   tete : ptr(maillon)   Fin
        maillon = struct   val : struct (numblc , depl : entier) ;  adr : ptr(maillon) Fin
Var     Index : tableau [ MaxIndex ] de Tcouple
```

Ins(e:TypeEnreg)

```
   // insertion à la fin du fichier de données ...
   OUVRIR( F, « donnees.dat » , 'A' )
   i ← Entete( F , 1 )                          // n° du dernier bloc de F
   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
   FERMER( F )
   // insertion dans la table d'index …
   Rech( e.cle , trouv , k )
   SI ( trouv )   // rajouter un maillon <i,j> la liste index[k].tete
                  Allouer( p ) ; Affval( p , < i , j > ) ; Affadr( p , Index[ k ].tete ) ;
                  Index[ k ].tete = p
   SINON         // insérer une nouvelle entrée <clé,<i,j>> dans l'index à la pos 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  >    // clé=e.c , tete=p
   FSI
```

Exemple : Index pour Fichier TOF avec zone de débordement




```
Type Tbloc = struct
                tab : tableau[ b ] de Tenreg
                NB : entier
                Lien : entier
             Fin
```

Exemple : Index pour Fichier LOF

(pas de décalages inter-blocs et pas de zone de débordement)

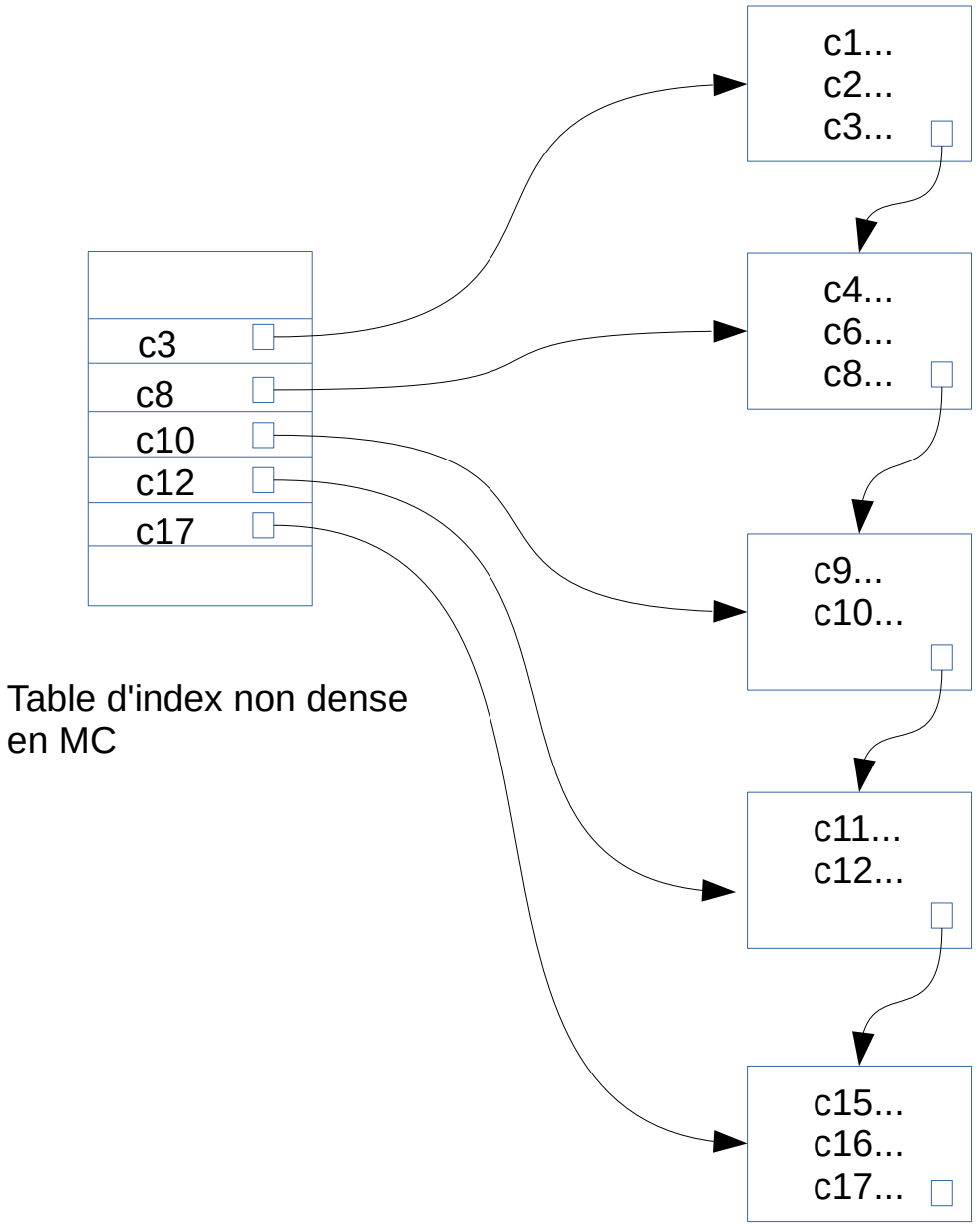


Table d'index non dense en MC

Fichier de données

Exemple : Fichier LOF / Insertion

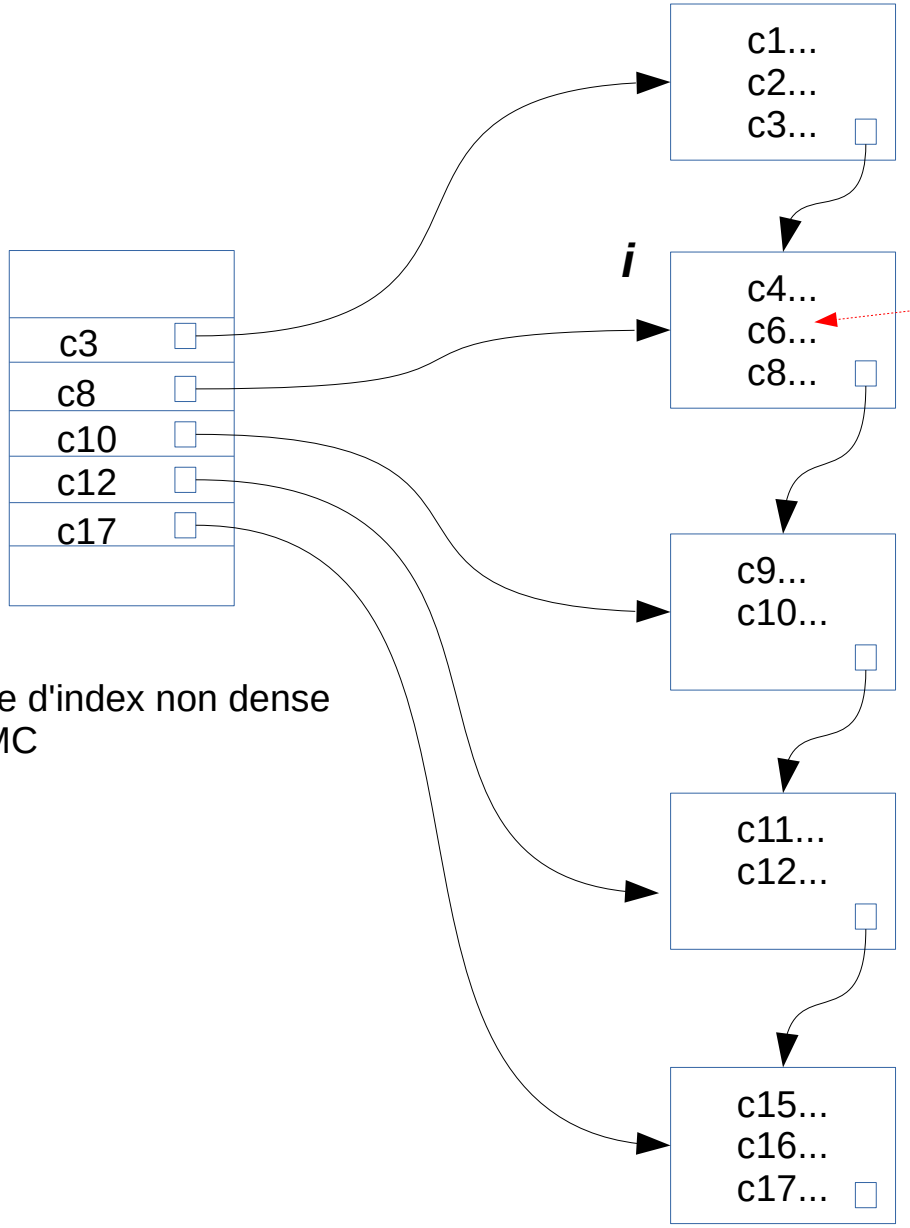
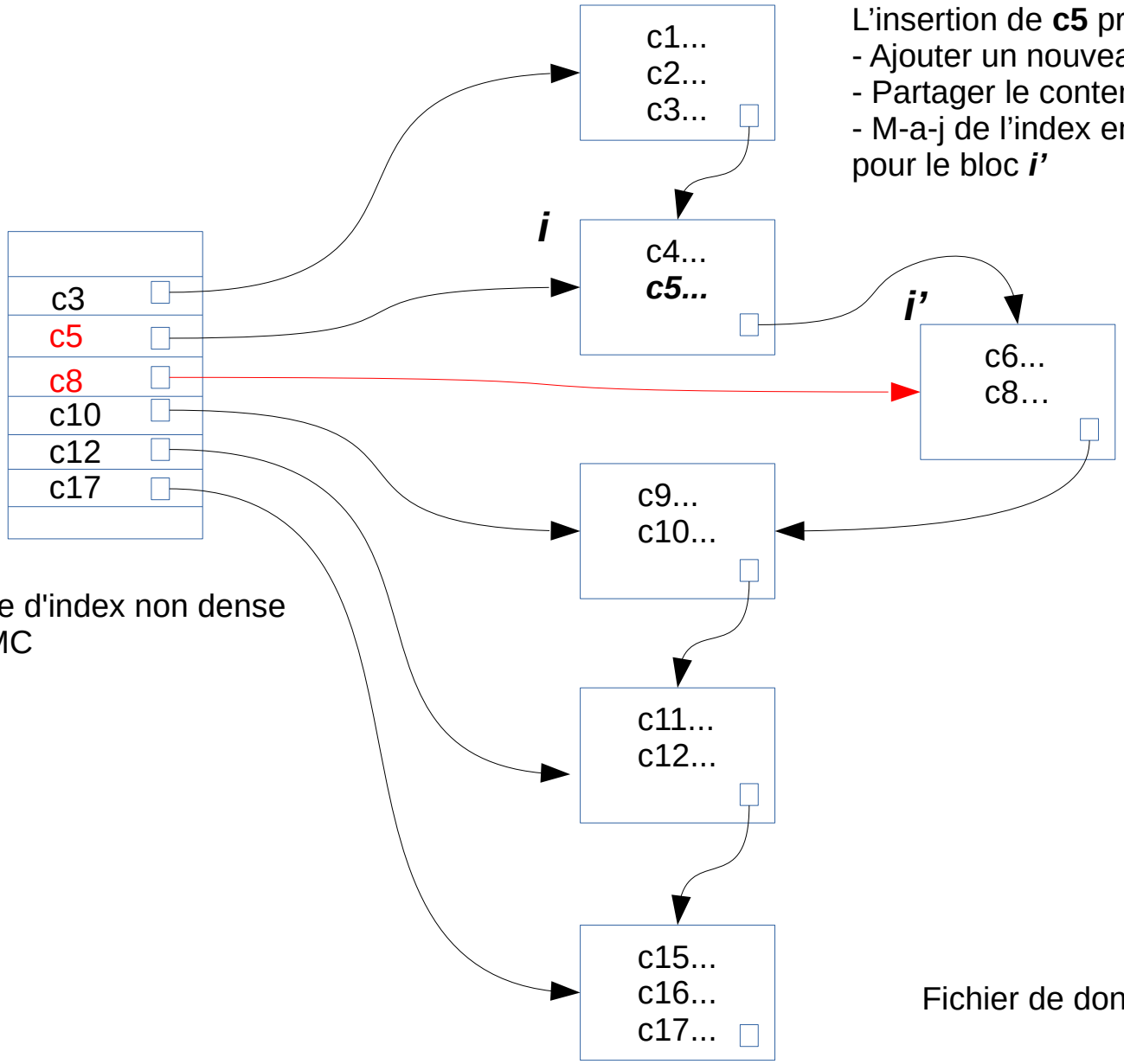


Table d'index non dense en MC

L'insertion de **c5** provoque l'éclatement du bloc ***i*** :
- Ajouter un nouveau bloc → ***i'***
- Partager le contenu de ***i*** en 2 moitiés
- M-a-j de l'index

Fichier de données

Exemple : Fichier LOF / Insertion

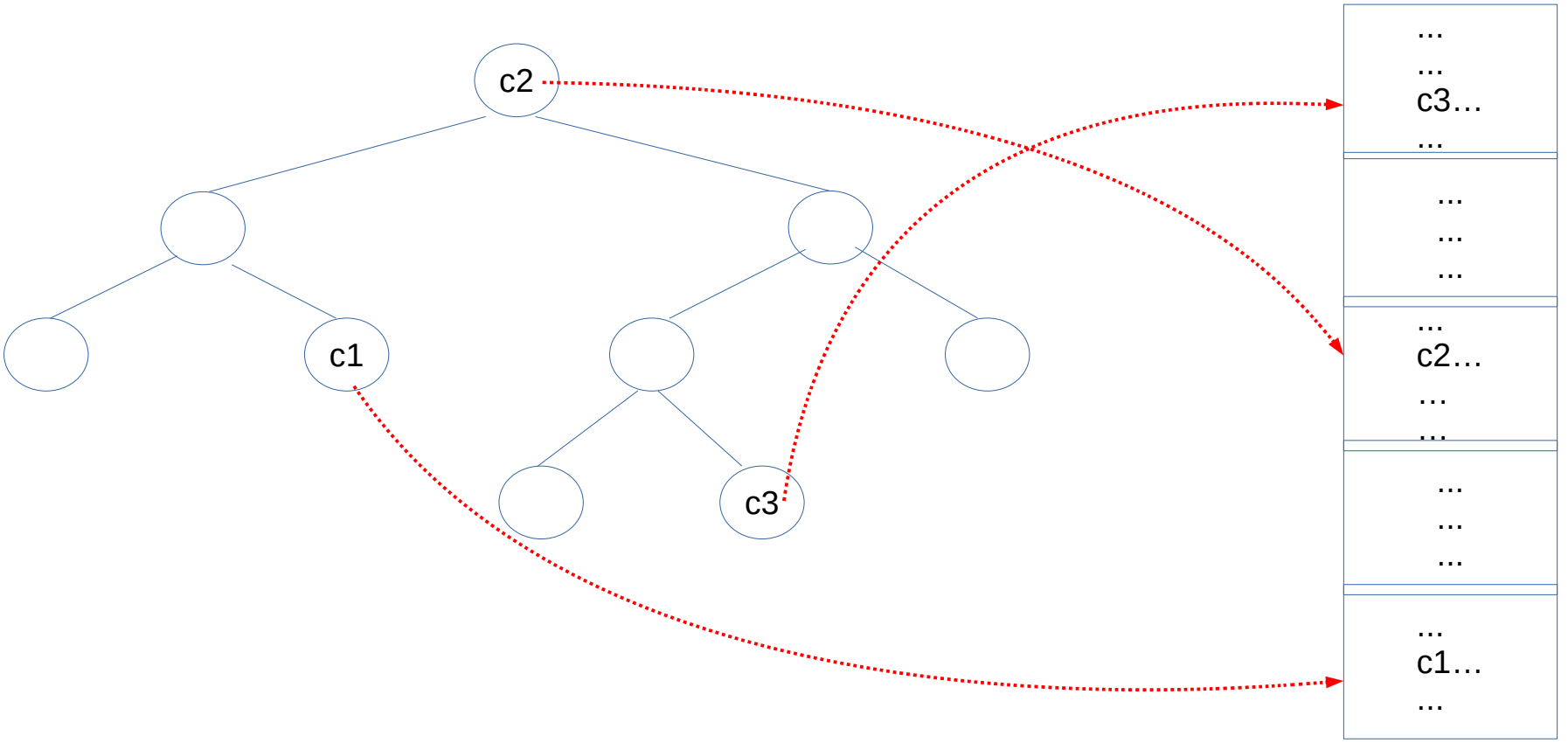


L'insertion de **c5** provoque l'éclatement du bloc ***i*** :
- Ajouter un nouveau bloc → ***i'***
- Partager le contenu de ***i*** en 2 moitiés
- M-a-j de l'index en insérant une nouvelle entrée pour le bloc ***i'***

Table d'index non dense en MC

Fichier de données

Index en MC sous forme de BST



Index = arbre de recherche binaire en MC **Fichier de données**

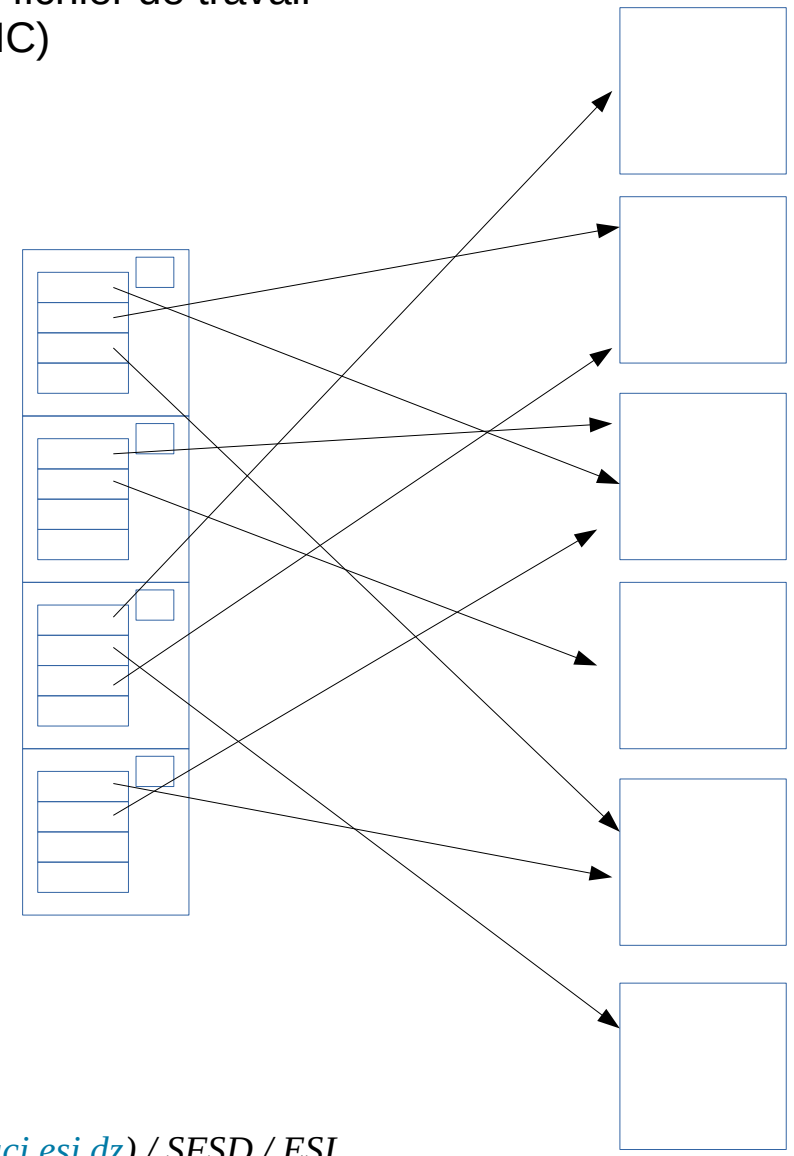
```
Type Tnoeud = struct
                clé : typeqlq
                numBlc , depl : entier
                fg , fd : ptr(Tnoeud)
        Fin
```

Index de grande taille

Si le fichier de données est volumineux (en nombre d'enregistrements) la taille de l'index peut devenir trop grande pour résider en MC.

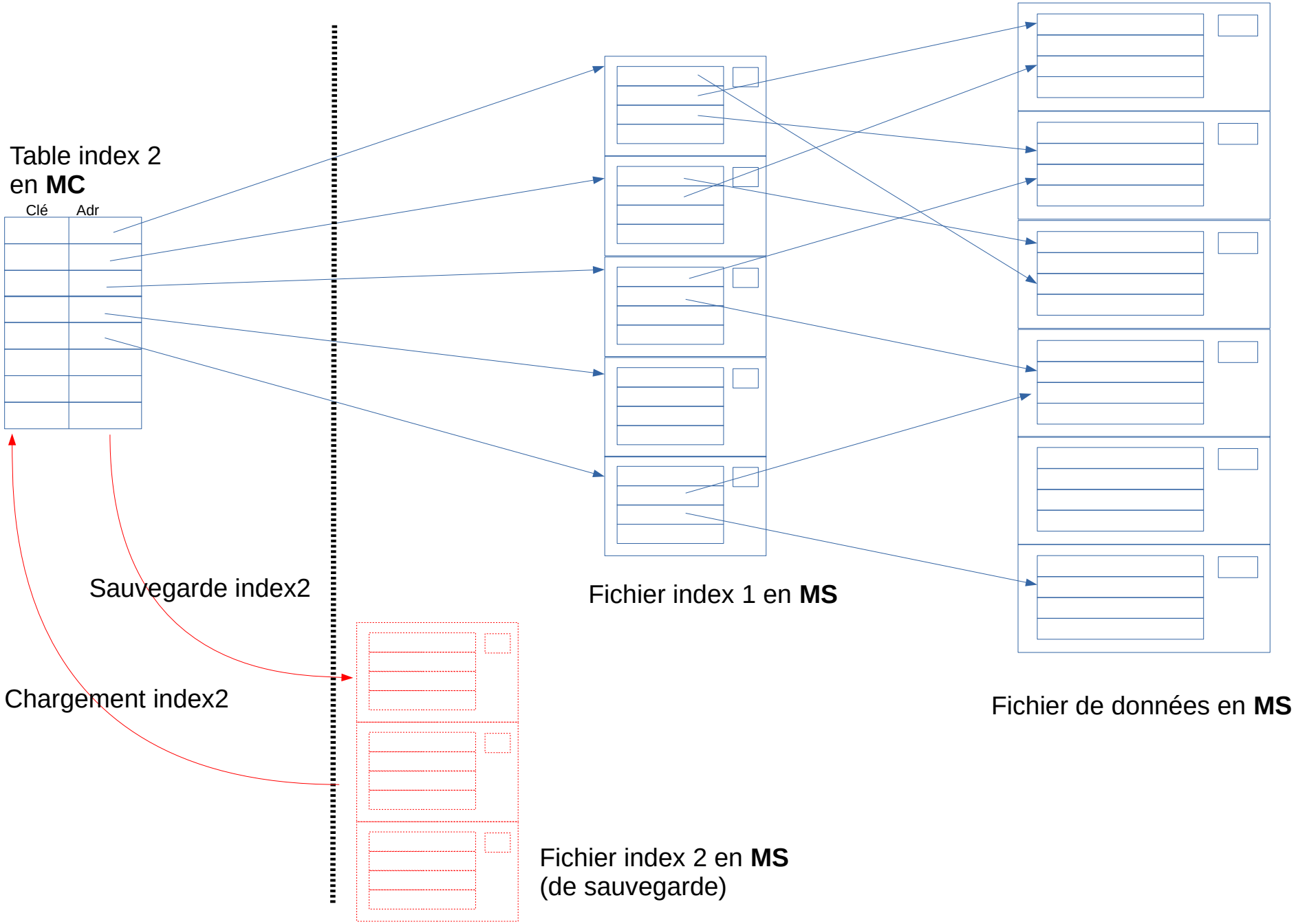
Une solution consiste à maintenir l'index en MS sous forme de fichier TOF (ou TOV$\overline{C}$)
⇒ Le fichier index devient un fichier de travail
(il n'y a plus de table en MC)

Index en MS
sous forme de
fichier ordonné
avec des blocs
contigus



Fichier de données
quelconque en **MS**

Index Multiniveaux



Accès Multiclés

Lorsque plusieurs attributs A1, A2, … An sont souvent utilisés comme conditions de recherche, il peut être intéressant de maintenir plusieurs index : IndA1, IndA2 … IndAn (un sur chaque attribut clé)

Les requêtes multiclés peuvent ainsi bénéficier du parcours de plusieurs index pour retrouver les enregistrements résultats → c'est l'accès multiclés

Si le fichier est ordonné selon l'un des attributs clés, l'index associé (non dense) sera appelé index primaire (Primary Index). Les autres index (forcément denses) seront considérés comme index secondaires (Secondary Indices).

IndxA3 (dense)

A3	adr
108	<5,3>
120	<2,3>
122	<5,2>
127	<1,2>
129	<3,2>
131	<4,3>
132	<3,3>
184	<6,3>
243	<5,1>
245	<1,1>
320	<6,1>
432	<1,3>
452	<2,2>
539	<4,2>
763	<3,1>
777	<6,2>
870	<2,1>
902	<4,1>

IndxA1 non dense

A1	adr
30	<1>
40	<2>
48	<3>
54	<4>
65	<5>
99	<6>

IndxA2 (dense)

A2	adr...
aaa	<4,2>, <3,2>,<2,2> ,<3,1>
bbb	<4,1>,<1,1>
ccc	<1,3>,<4,3>
ddd	<2,3>
eee	<2,1,>,<5,3> ,<6,1>
ggg	<3,3,>,<5,2> ,<6,2>
nnn	<5,1>
ppp	<6,3>,<1,2>

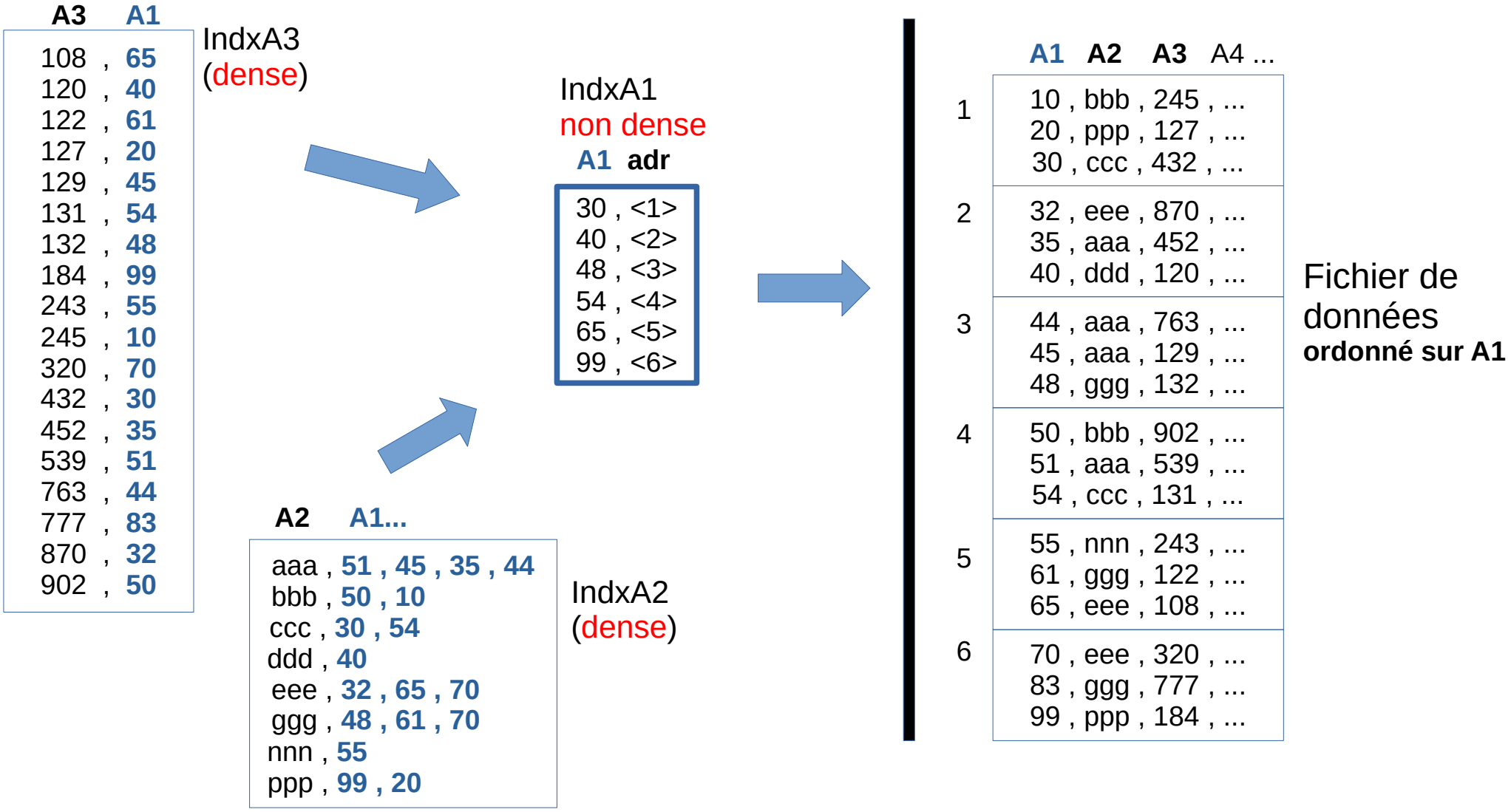
	A1	A2	A3	A4 ...
1	10	bbb	245	...
	20	ppp	127	...
	30	ccc	432	...
2	30	eee	870	...
	30	aaa	452	...
	40	ddd	120	...
3	45	aaa	763	...
	45	aaa	129	...
	48	ggg	132	...
4	50	bbb	902	...
	54	aaa	539	...
	54	ccc	131	...
5	54	nnn	243	...
	61	ggg	122	...
	65	eee	108	...
6	70	eee	320	...
	83	ggg	777	...
	99	ppp	184	...

Fichier de données **ordonné sur A1**

Une variante est souvent utilisée, lorsque l'index primaire est basée sur une **clé primaire** (à valeurs uniques) en se basant sur les ***index inversés*** :

L'idée est que les index secondaires utilisent à la place des adresses, des valeurs de clés primaires.

Donc l'accès par une valeur de clé secondaire, commence au niveau de l'index secondaire associé, puis passe par l'index primaire (pour récupérer les adresses des enreg) et se termine par l'accès au fichier de données.



Etapes d'une requête multi-clés avec la méthode des index inversés
« trouver tous les enregistrements dont la valeur de X = vx ET la valeur de Y = vy ET ... »
avec X, Y, ... des 'clés secondaires' (Pour chaque clé secondaire, il y a donc un index secondaire) :

1- En utilisant l'index secondaire X, trouver la liste **Lx** de clés primaires associées à la valeur vx.
(Refaire la même action pour chaque clé secondaire mentionnée dans la requête…)

2- Faire l'intersection des listes de clés primaires **Lx**, **Ly**, ... pour trouver les clés primaires associées avec chaque valeur de clé secondaire mentionnée dans la requête.

3- Utiliser l'index primaire pour retrouver les enregistrements du fichier de données (en triant d'abord la séquence des numéros de blocs avant d'effectuer les transferts physiques).

Dans la figure de l'exemple précédent, si on cherche tous les enregistrements tel que A2 = 'eee' et A3 = 870,

L'algorithme de la requête multiclés procédera comme suit :

a- **Recherche** de 'eee' dans l'index **IndA2** → résultat : L_{A2} = [32 , 65 , 70]

b- **Recherche** de 870 dans l'index **IndA3** → résultat : L_{A3} = [32]

c- Intersection de L_{A2} et L_{A3} → résultat : L_{finale} : [32]

d- **Recherche** de 32 dans **IndA1** → résultat : bloc_N° <2>

e- **LireDir(F , 2 , buf)** et récupérer, l'enreg « 32 , bbb , 870 , … »

Insertion d'un enregistrement < ***c*** , ***vx*** , ***vy*** , … >

1- Rechercher ***c*** dans l'index primaire → *ip* : l'indice où doit être insérée cette clé (recherche dichotomique).
2- Insérer l'enregistrement dans le fichier de données → *adr* : l'adresse où l'enreg a été inséré
3- Insérer dans l'index primaire, à la position *ip* l'entrée < ***c*** *, adr* > s'il s'agit d'un index dense
ou alors, éventuellement, mettre à jour l'entrée qui se trouve à l'indice *ip* s'il s'agit d'un index non dense.

4- Rechercher la valeur ***vx*** dans l'index secondaire X,
si ***vx*** existe, rajouter **c** à la liste pointée par ***vx***
si ***vx*** n'existe pas, insérer ***vx*** dans l'index de secondaire X.
→ dans ce cas la nouvelle entrée ***vx***, pointera une liste formé par une seule clé primaire (**c**).

Refaire l'étape 4) pour chaque clé secondaire restante (***vy***, ...).

Suppression d'un enregistrement < ***c*** , ***vx*** , ***vy*** , … >

Pour supprimer logiquement un enregistrement de clé primaire ***c***, il suffit de positionner un bit (ou caractère) d'effacement au niveau du fichier de données ou de la table d'index primaire, pour l'entrée ***c***.

Pour supprimer physiquement un enregistrement de clé primaire ***c***, il faut d'abord supprimer physiquement l'enregistrement dans le fichier de données, ensuite il faut mettre à jour la table d'index primaire soit en supprimant l'entrée relative à c (cas d'un index dense) ou alors modifier la clé et/ou l'adresse du représentant du groupe auquel appartient l'enregistrement supprimé (cas d'un index non dense).

Dans les deux types de suppression (logique ou physique), **il n'est pas nécessaire de mettre à jour les index secondaires**.

Index Bitmap

Un index bitmap sur un attribut A (formé par ***m*** différentes valeurs : v_1, v_2, … v_m)
est composé de ***m*** chaînes binaires, de ***N*** bits chacune ($IndA\_v_1$, $IndA\_v_2$ ... $IndA\_v_m$) :

Chaque chaîne $IndA\_v_j$ est associée à la valeur v_j de l'attribut A

Si ($IndA\_v_j[\ k\ ] = 1$), alors dans l'enregistrement n°k, l'attribut A vaut v_j

Si ($IndA\_v_j[\ k\ ] = 0$), alors dans l'enregistrement n°k, l'attribut A est différent de v_j

Index_A =

n° enregistrement (le fichier contient ***N*** enregistrements)

	1	2	3	4	5	6	7	8	…	i	…	…	… *N*	
IndA_v1 :	1	0	0	0	1	1	0	1	….	0	…	1	1	0	(N bits) → la chaîne de bits associée à v1
IndA_v2 :	0	1	0	0	0	0	0	0	….	1	…	0	0	0	(N bits) → la chaîne de bits associée à v2
…	…		…			…			…		…		…		… ...
IndA_vm :	0	0	1	1	0	0	1	0	….	0	…	0	0	1	(N bits) → la chaîne de bits associée à vm

Exemples : A = v2 dans l'enregistrement n°2 et l'enregistrement n°i du fichier de donnée
A = v1 dans les enregistrements n° 1, 5, 6, 8, … N-2 et N-1
...

Les index bitmaps peuvent être utiles pour les **attributs à faible cardinalité** (ex < 20 valeurs distinctes).

Les différentes chaînes de bits peuvent être **chargées en MC** indépendamment les unes des autres.

Ils sont utilisés principalement pour les requêtes multiclés sur des attributs à faible cardinalité

Exemple : *« trouver les enregistrements tel que A = v2 et B = w4 »*

Index_A =

(cardinalité de A = 3 valeurs : v1,v2,v3)

	n° enr													
	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

Index_B =

(cardinalité de B = 4 valeurs : w1,w2,w3,w4)

	n° enr													
	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**

Le résultat de la requête est donné par **l'opération binaire : (*IndA_v2* AND *IndB_w4*)**
→ Les enregistrements n° **7** et n° **i**