Democratic and Popular Republic of Algeria

Ministry of Higher Education and Scientific Research

ESTIN

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

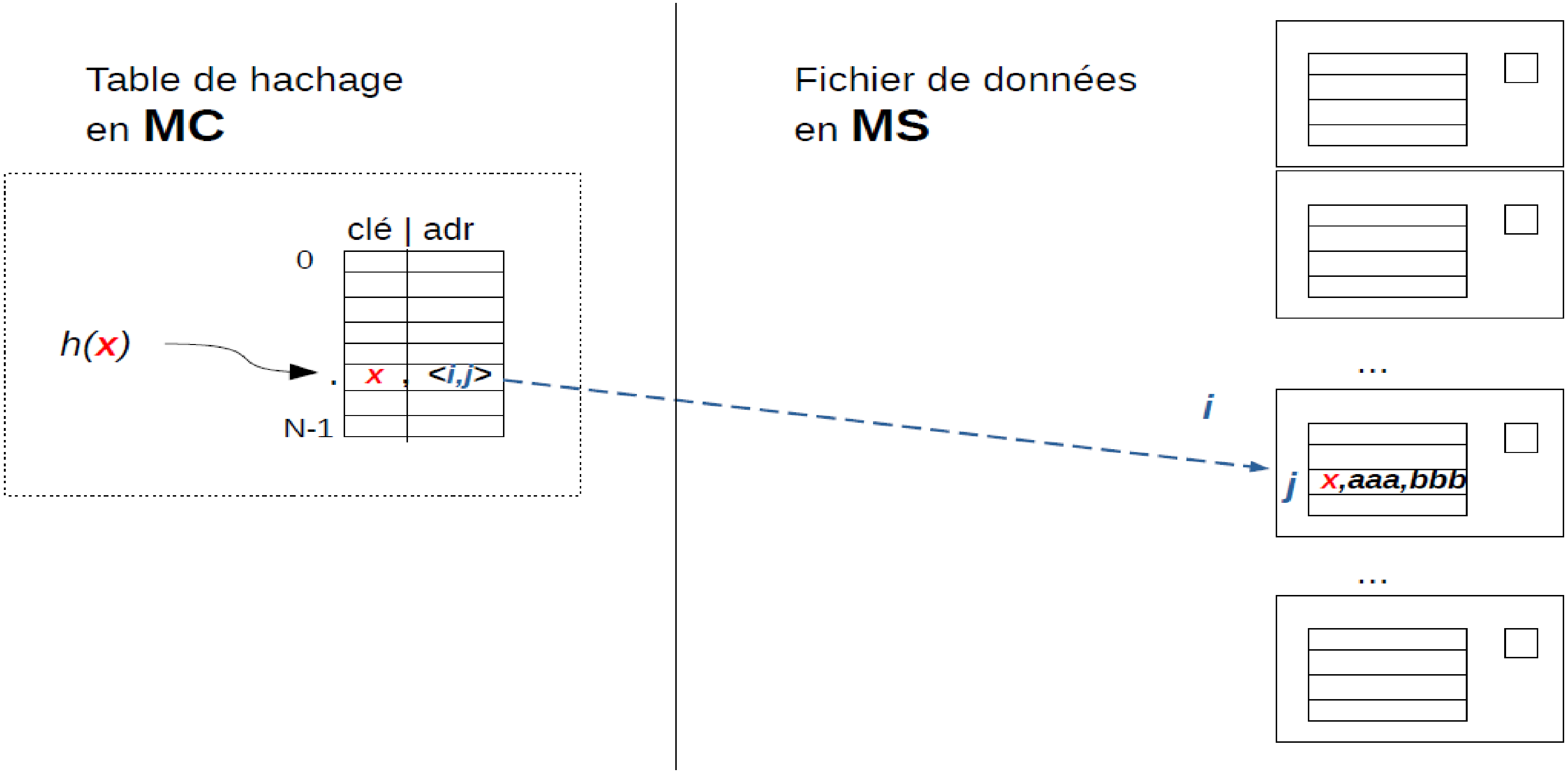
Hashing methods

Presented by : Dr. Daoudi Meroua

Academic year: 2024/2025

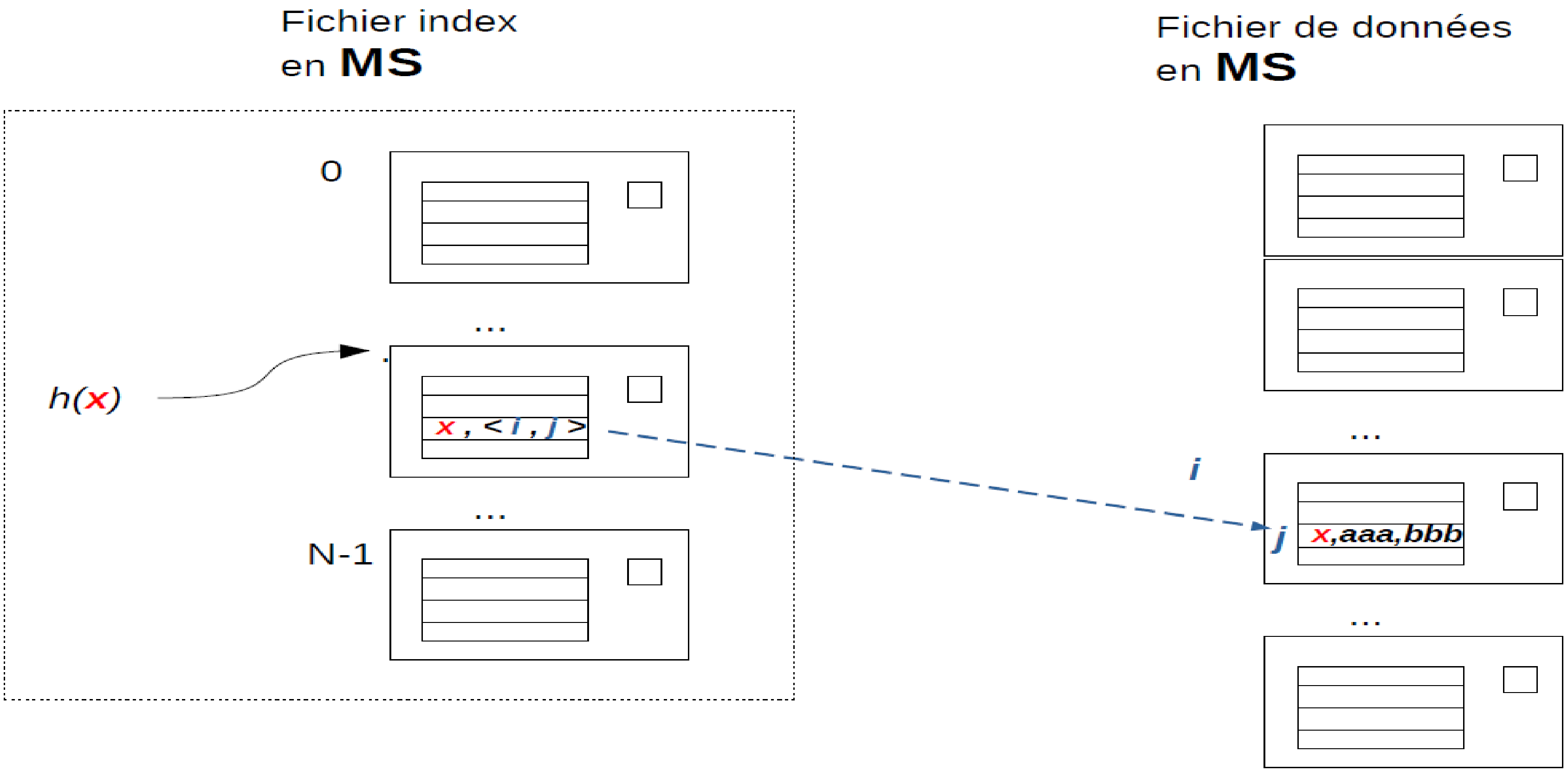
Use of hash tables

1. Use a hash table as an index, in MC, to speed up access to data files.



Use of hash tables

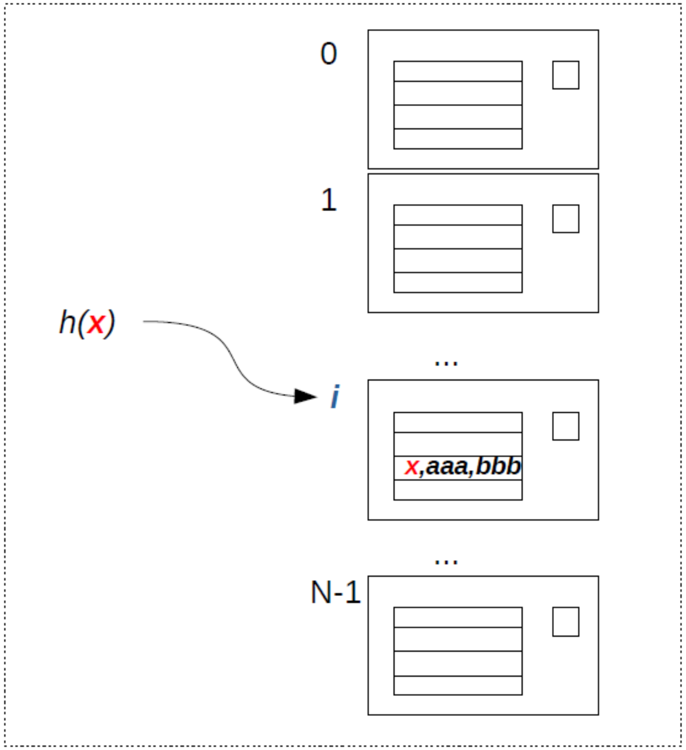
2. Use an index in MS, managed by a hashing method.



Use of hash tables

3. Manage the data file using a hashing method.

Fichier de données
en **MS**



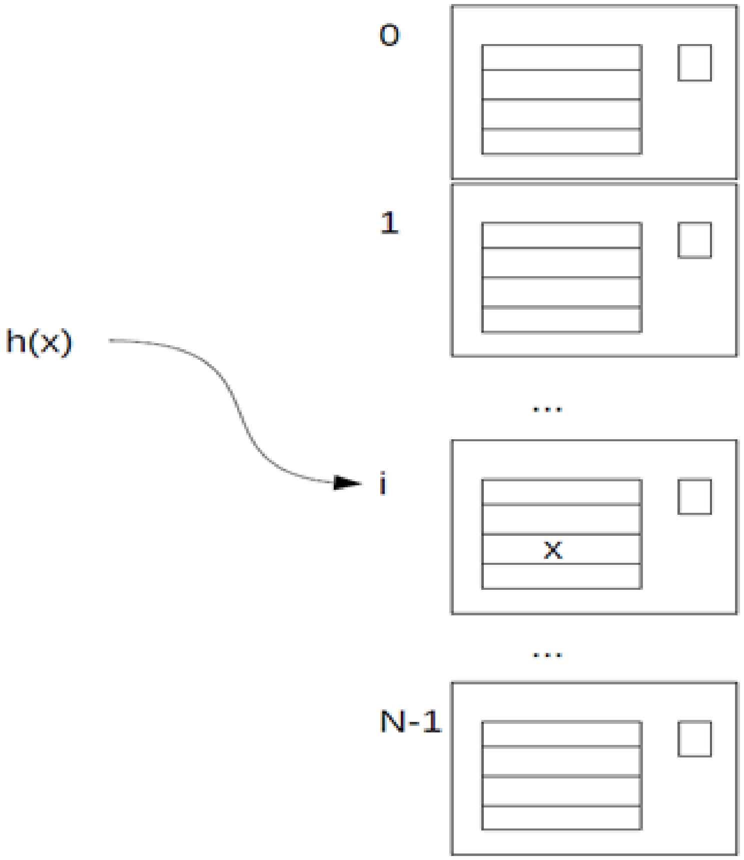
Use of hash tables

File with hashing

The primary address of the record with key x is the block number h(x).

If the block is full, one of the collision resolution methods is used.

The capacity of a block is b records



Collision resolution methods

- **Linear Probing**

Sequence of probes: blocks numbered h(x) , h(x)-1 , h(x)-2 , … 0 , N-1 , … < non_full_block >

- **External Chaining**

Sequence of probes: block number h(x) and those in its overflow list that are located outside the range addressable by the hash function h.

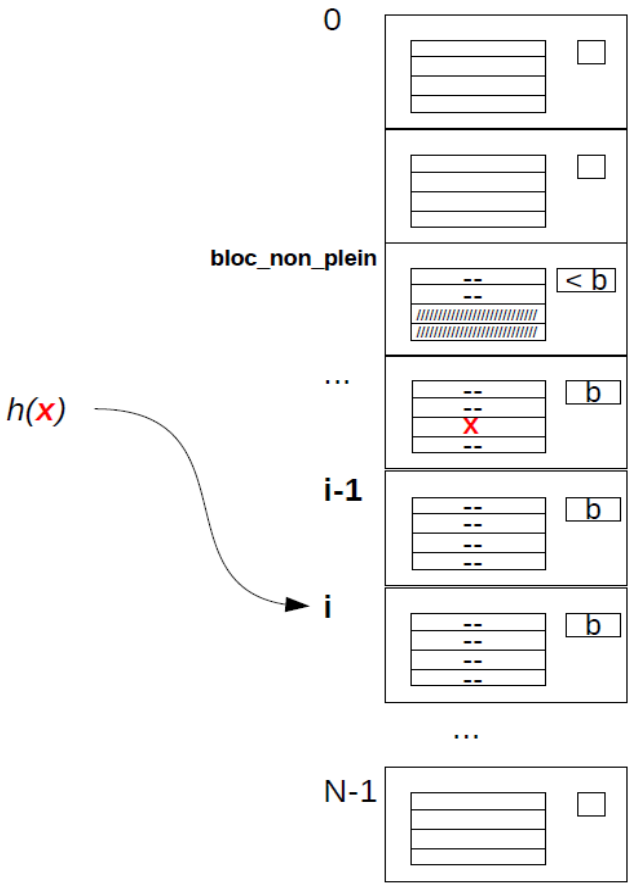
Linear Probing

There must always be at least one non-full block.

The sequence of probes may eventually be circular.

The insertion is made in the first non-full block found in the sequence of probes.

Logical or Physical Deletion



Linear Probing : Search algorithm

The characteristics:

1- N : the number of blocks forming the file

2- nbIns : the number of data entries inserted

```
Rech( entrée : x sorties : trouv, i, j )
i ← h(x) ; trouv ← faux ; stop ← faux ; N ← Entete( F, 1 )
TQ ( Non trouv && Non stop )
    LireDir( F, i, buf )
    j ← 1 // Internal search within block i
    TQ ( j ≤ buf.NB && Non trouv )
        SI ( x = buf.tab[ j ].cle ) trouv ← vrai SINON j ← j+1 FSI
    FTQ
    SI ( buf.NB < b ) // If there is an empty slot (non-full block)
        stop ← vrai // Then end of the probe sequence
    SINON i ← i - 1 ; SI ( i < 0 ) i ← N-1 FSI // Otherwise, continue the
probes
    FSI
FTQ
```

Linear Probing : insertion

The characteristics:

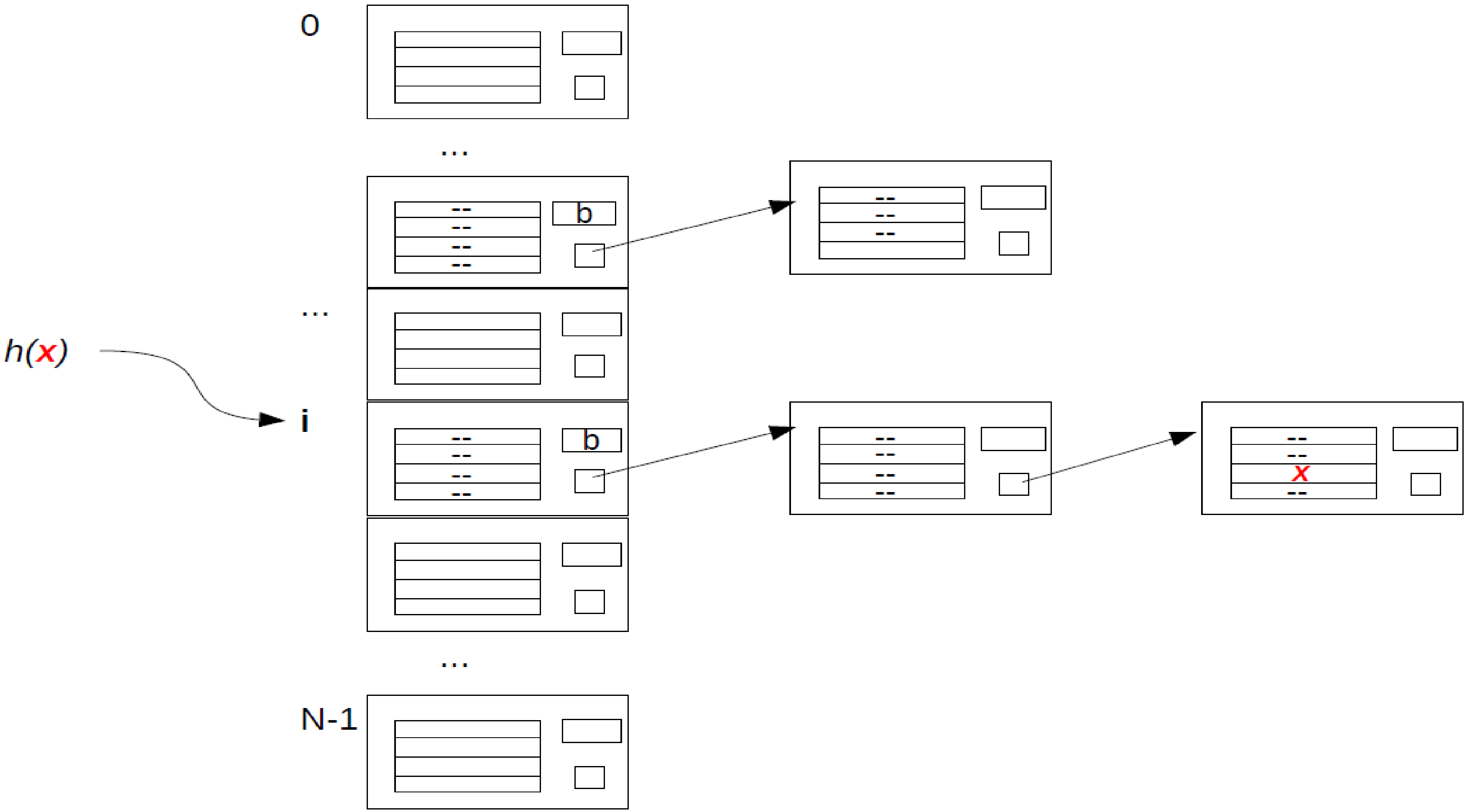
1- N : the number of blocks forming the file

2- nbIns : the number of data entries inserted

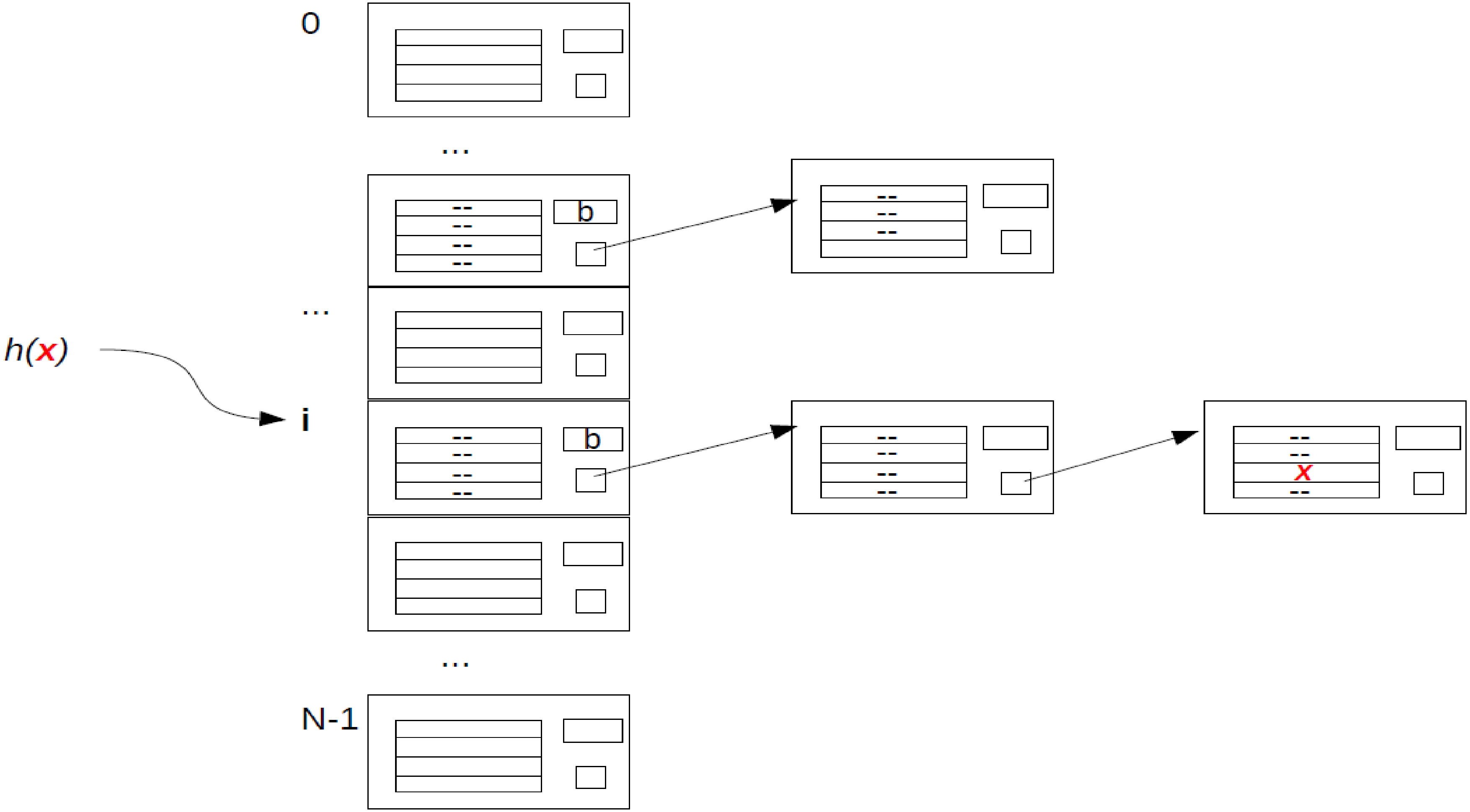
Ins(entrée : e)

```
N ← Entete( F, 1 ) ; nbIns ← Entete( F, 2 )
   SI ( nbIns = N * b – 1 ) Insertion impossible // No available space
  SINON
      Rech( e.clé , trouv , i , j )
      SI ( Non trouv )
         buf.NB++
         buf.tab[ buf.NB ] ← e
         EcrireDir( F, i, buf )
         Aff_Entete( F , 2 , nbIns+1 )
      FSI
   FSI
```

External Chaining



External Chaining



External Chaining : Search algorithm

We assume that F contains the main area (the blocks between 0 and N-1) and the overflow area (the blocks from number N to the end of the file)

The characteristics:

N: the number of blocks forming the main area

M: the total number of blocks in F (main area + overflow area)

nbIns: the number of data entries inserted

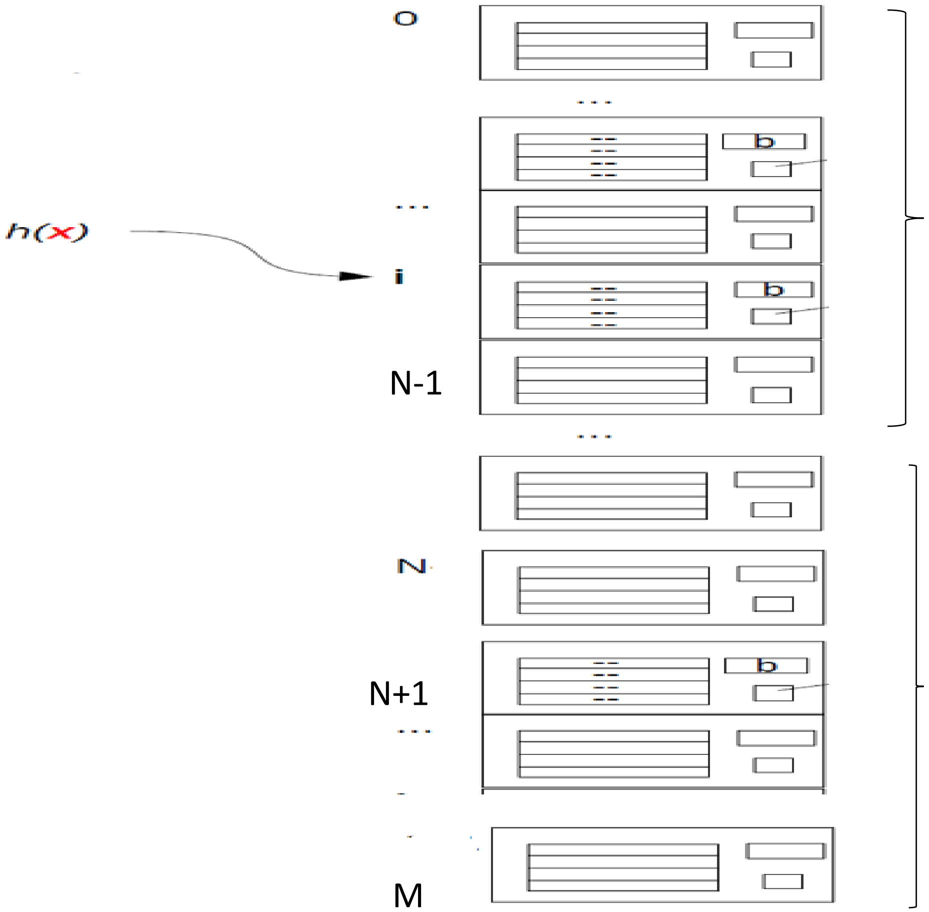
Rech(entrée : x sorties : trouv, i, j)

```
i ← h(x) ; trouv ← faux ; stop ← faux ; LireDir( F, i, buf )
TQ ( Non trouv && Non stop )
       j ← 1 // Internal search within block i
       TQ ( j ≤ buf.NB && Non trouv )
              SI ( x = buf.tab[ j ].cle ) trouv ← vrai SINON j++ FSI
       FTQ
      SI ( Non trouv )
         SI ( buf.lien <> -1 ) i ← buf.lien ; LireDir( F, i, buf ) SINON stop ← vrai FSI
      FSI
FTQ
```

External Chaining : insertion algorithm

```
Ins( entrée : e , nomFich : chaîne )
Ouvrir( F , nomFich , 'A' )
Rech( e.clé, trouv, i, j )
SI ( Non trouv )
// If there is space in the last visited block, insert e there
    SI ( buf.NB < b )
        buf .NB++ ; buf.tab[ buf.NB ] ← e ; EcrireDir( F, i, buf )
    SINON
// If the last block is already full, allocate a new overflow block"
          nouvBloc ← Entete( F , 2 ) + 1
          buf.lien ← nouvBloc // Chain the new block with the previous one (i)
          EcrireDir( F, i, buf )
          buf .NB ← 1 ; buf.tab[ 1 ] ← e // Insert e into the new block
          buf.lien ← -1
          EcrireDir( F, nouvBloc, buf )
          Aff_Entete( F , 2 , Entete( F , 2 ) + 1 ) // The total number of blocks
     FSI
     Aff_Entete( F , 3 , Entete( F , 3 ) + 1 ) // The number of insertions
FSI  Fermer( F )
```

Internal Chaining



Zone primaire

Zone secondaire ou
de débordement