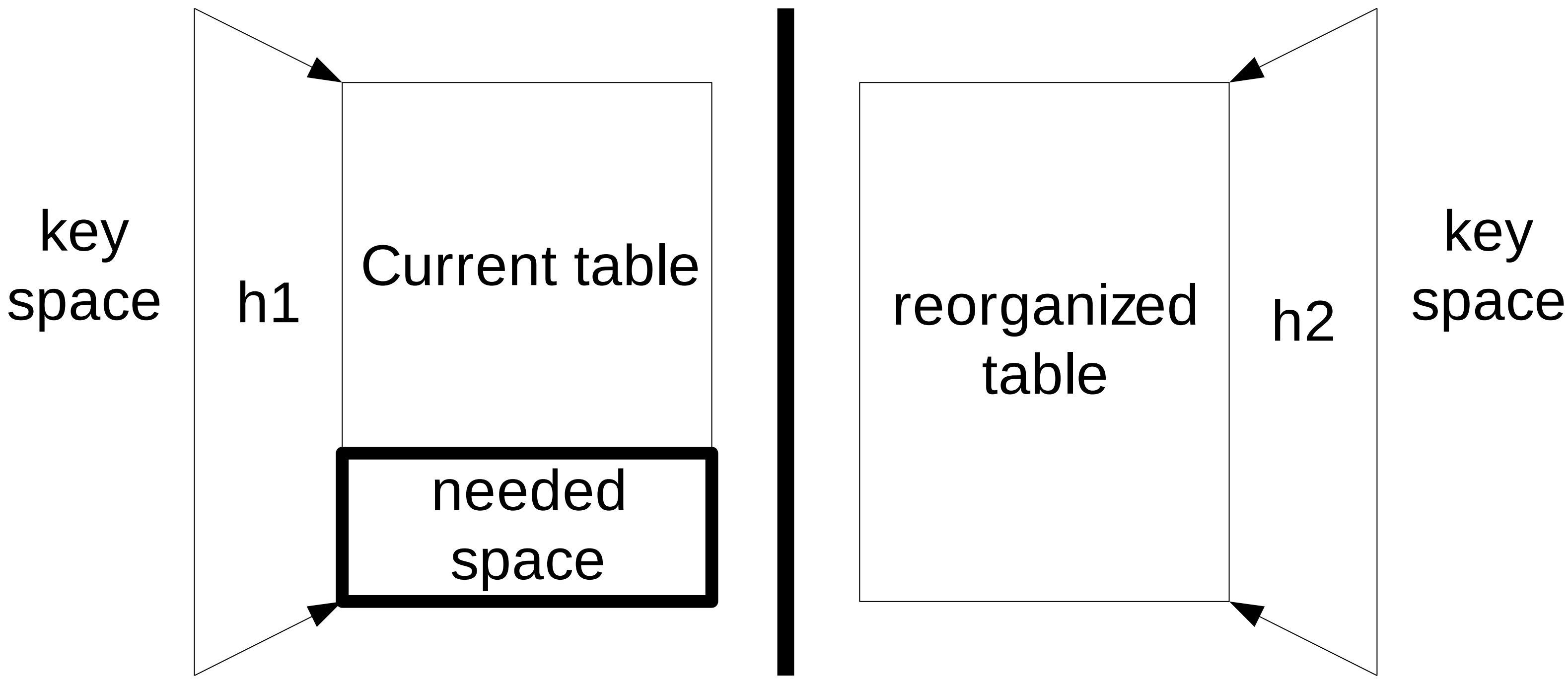
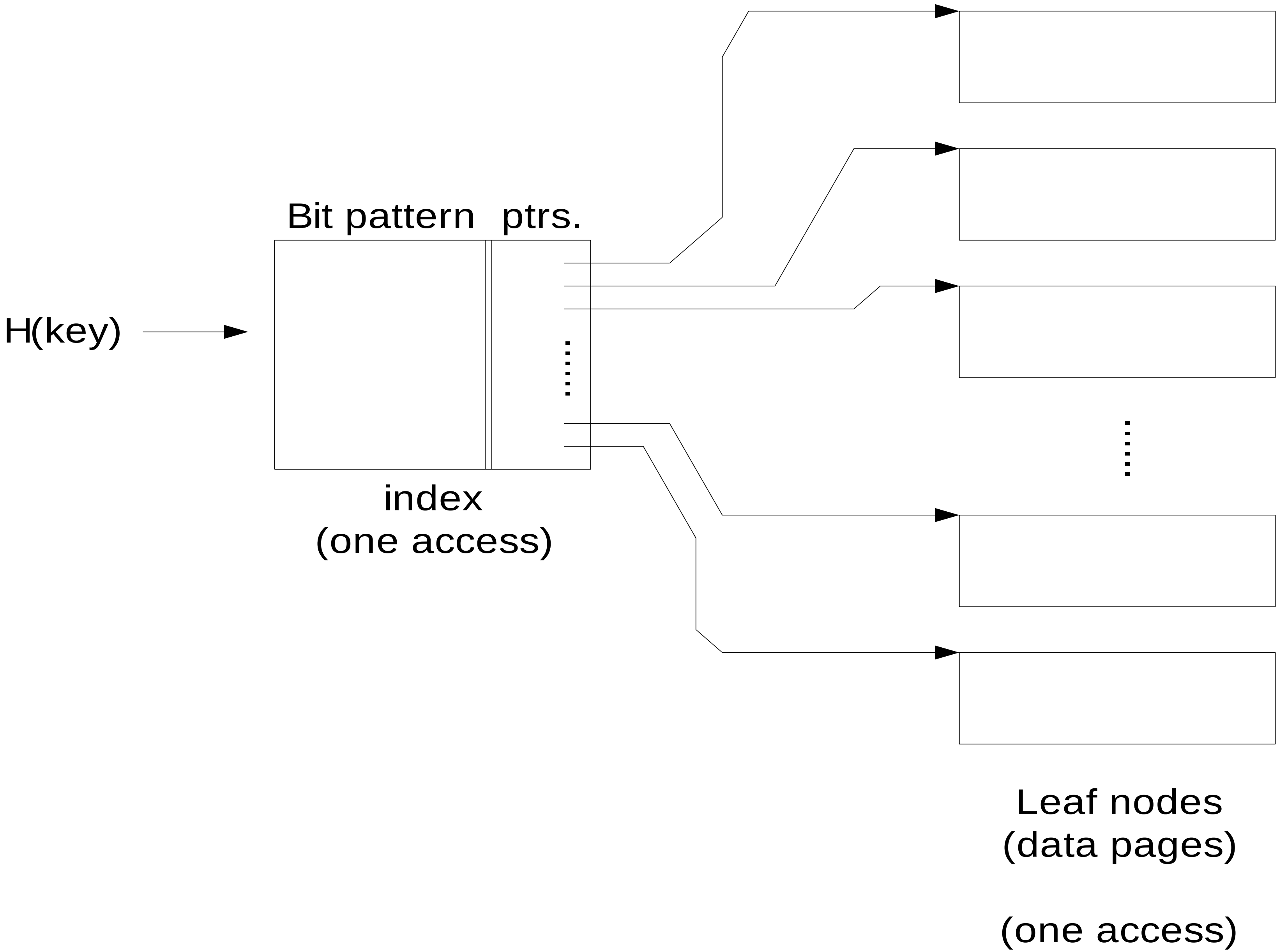
Dynamic Hashing

- **Static hashing schemes**
  - Static table
  - Unload all the data and reenter them
  - Reorganization



- **Dynamic hashing schemes**
  - Extendible hashing
  - Dynamic hashing
  - Linear hashing

- **Extendible Hashing:**
  - Use index
  - Records are stored in terminal nodes
  - Page overflow → split → modify index
  - H(key) = pseudokey
  - Page depth = the number of bits to distinguish the pseudokey on which page
  - Index depth = maximun of all the page depth
  - Not a sequential order


Bit pattern
ptrs.
H(key)
index
(one access)
Leaf nodes
(data pages)
(one access)

Algorithm

- Extendible Hashing Insertion
  - Hashing_function(key) = pseudokey
  - N(index depth) most significant bits of the pseudokey = address of entry in index
  - Pointer in index → proper page
  - If space available in page, insert record, else

A. Split the overflowing page
B. Place each group of records into a separate page
C. Determine the page depths of these two pages
D. If these page depths < the index depth, then update index pointers

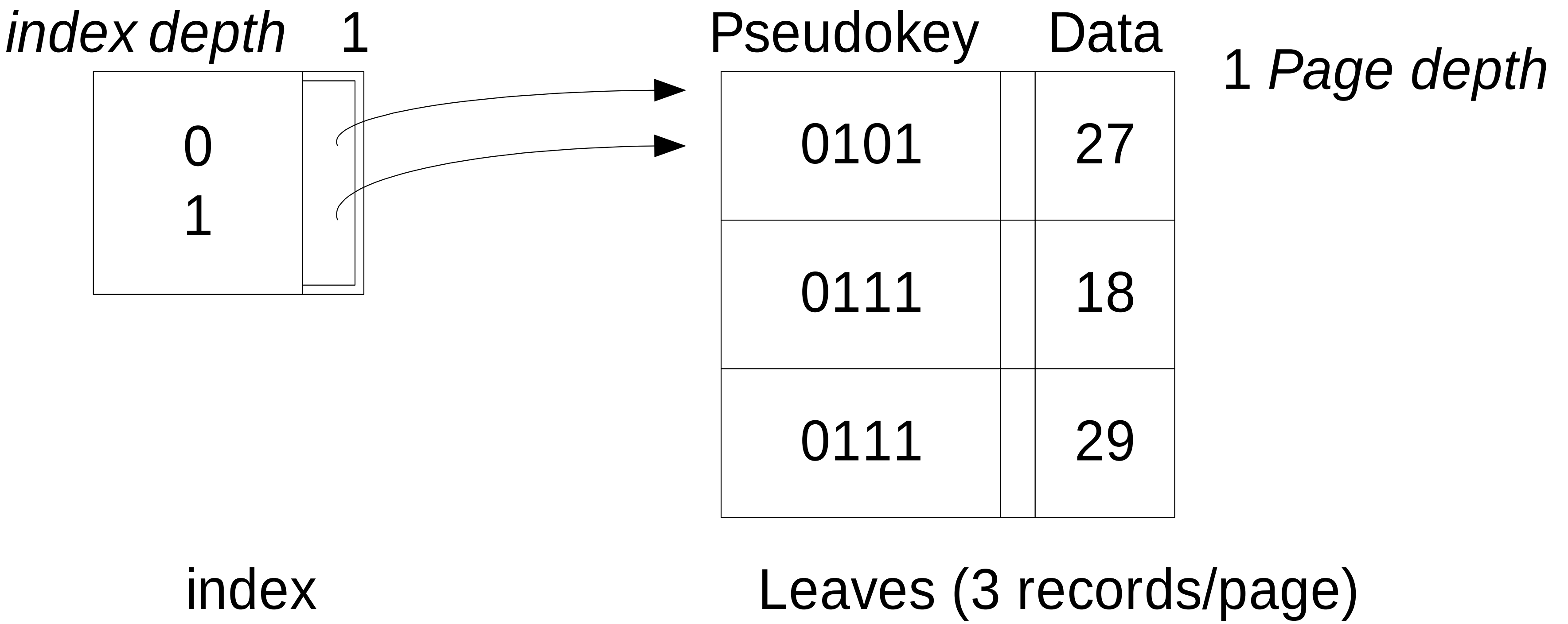
   Else, index pointer = maximum(k)

   → expand index size

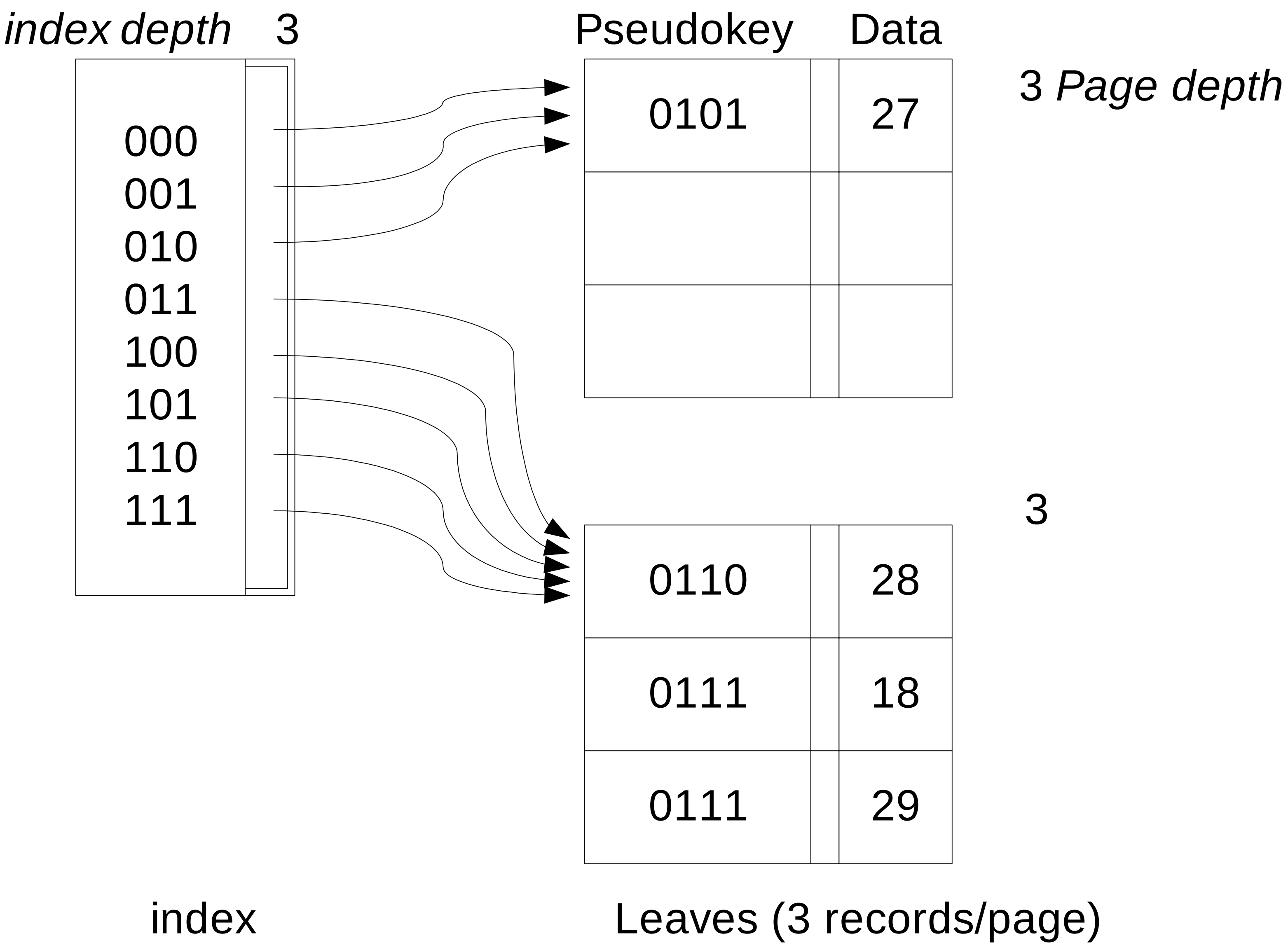
   → adjust index pointer

An Example

- Data page = 3 records
- H(key) = key mod 11
- Data is 27, 18, 29, 28, 42, 13, 16

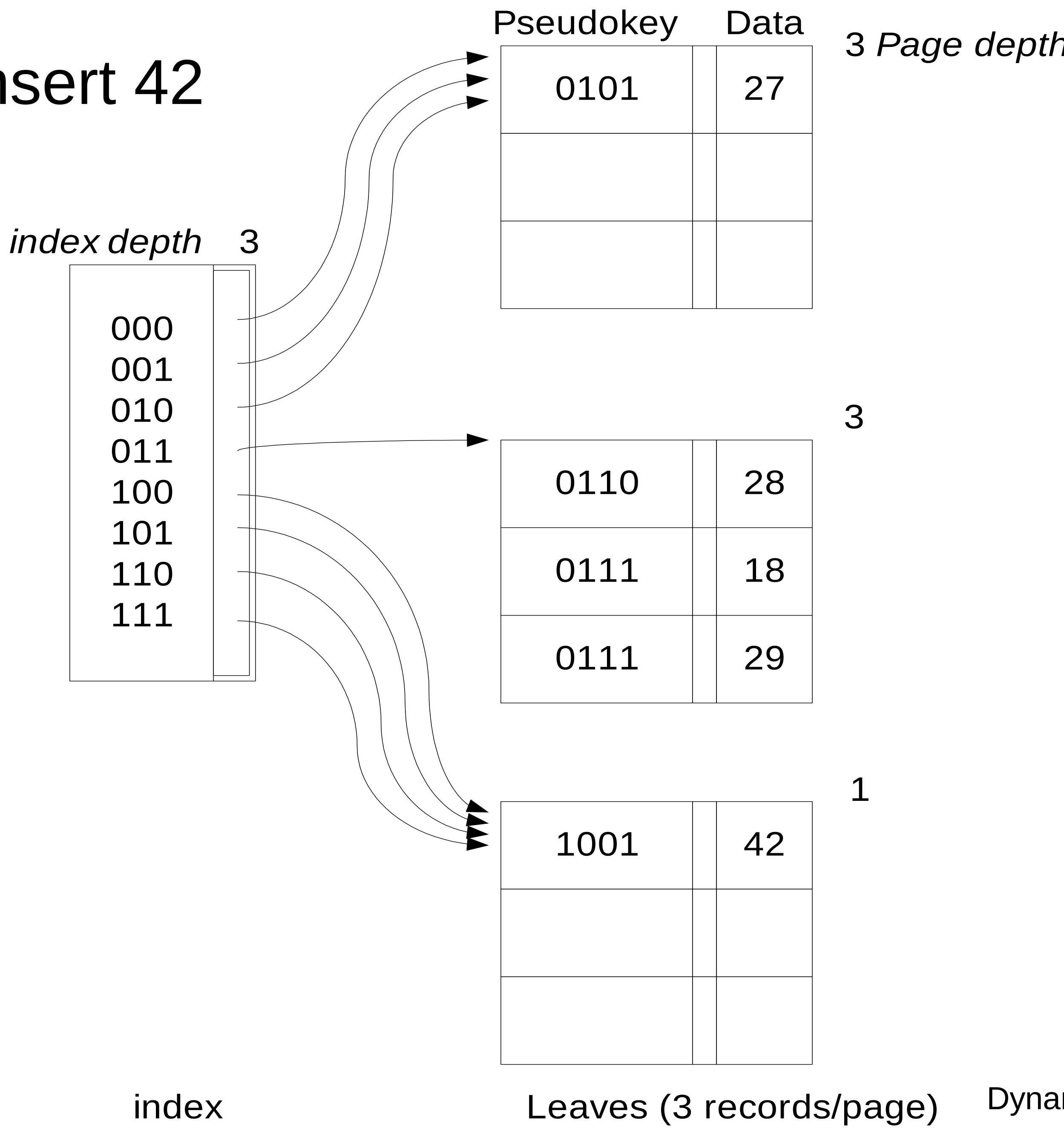


• Insert 28



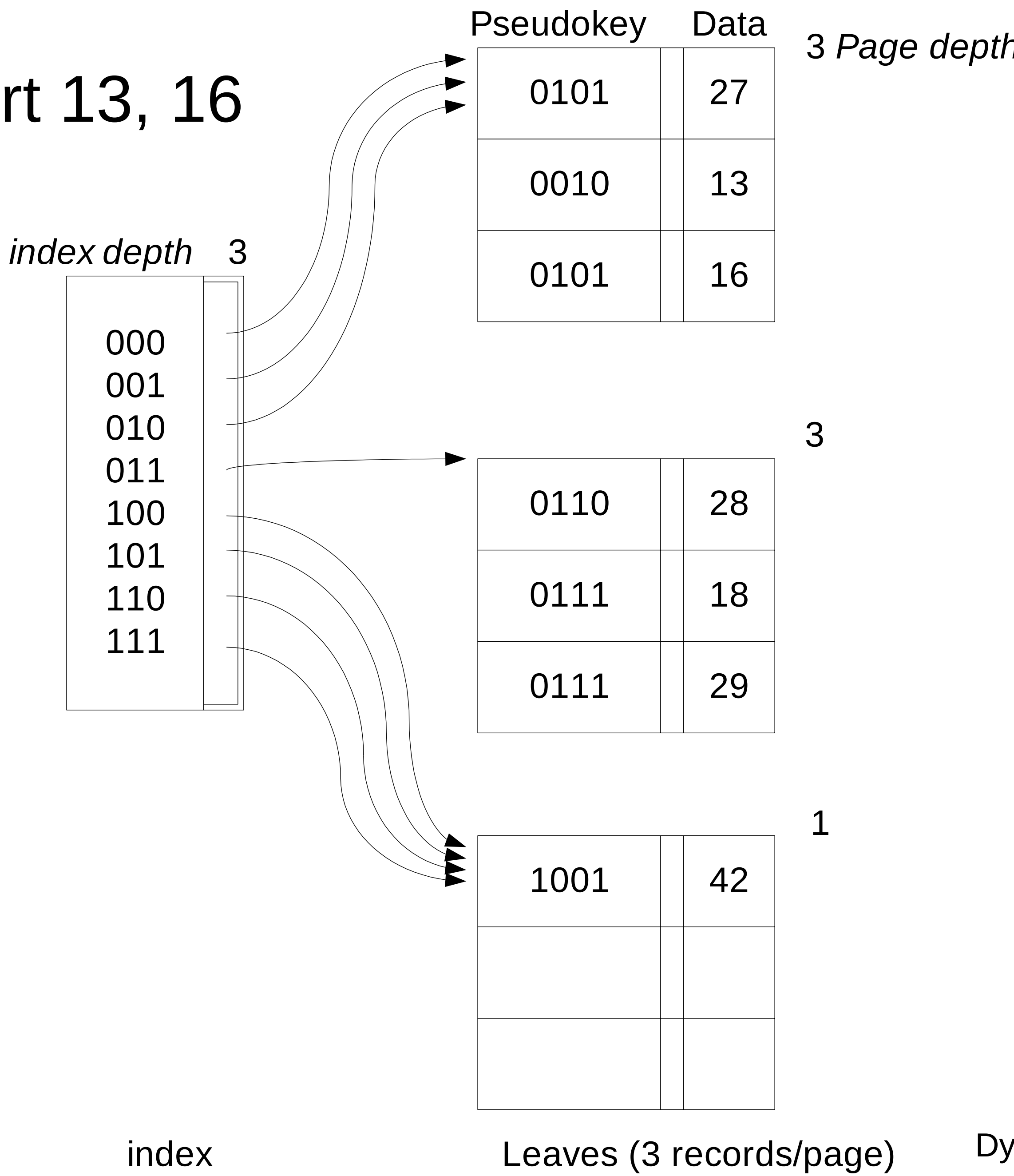
index depth 3
Pseudokey
Data
3 Page depth
000
001
010
011
100
101
110
111
0101
27
3
0110
28
0111
18
0111
29
index
Leaves (3 records/page)

- Insert 42



Pseudokey
Data
3 *Page depth*
0101
27
index depth 3
000
001
010
011
100
101
110
111
3
0110
28
0111
18
0111
29
1
1001
42
index
Leaves (3 records/page)

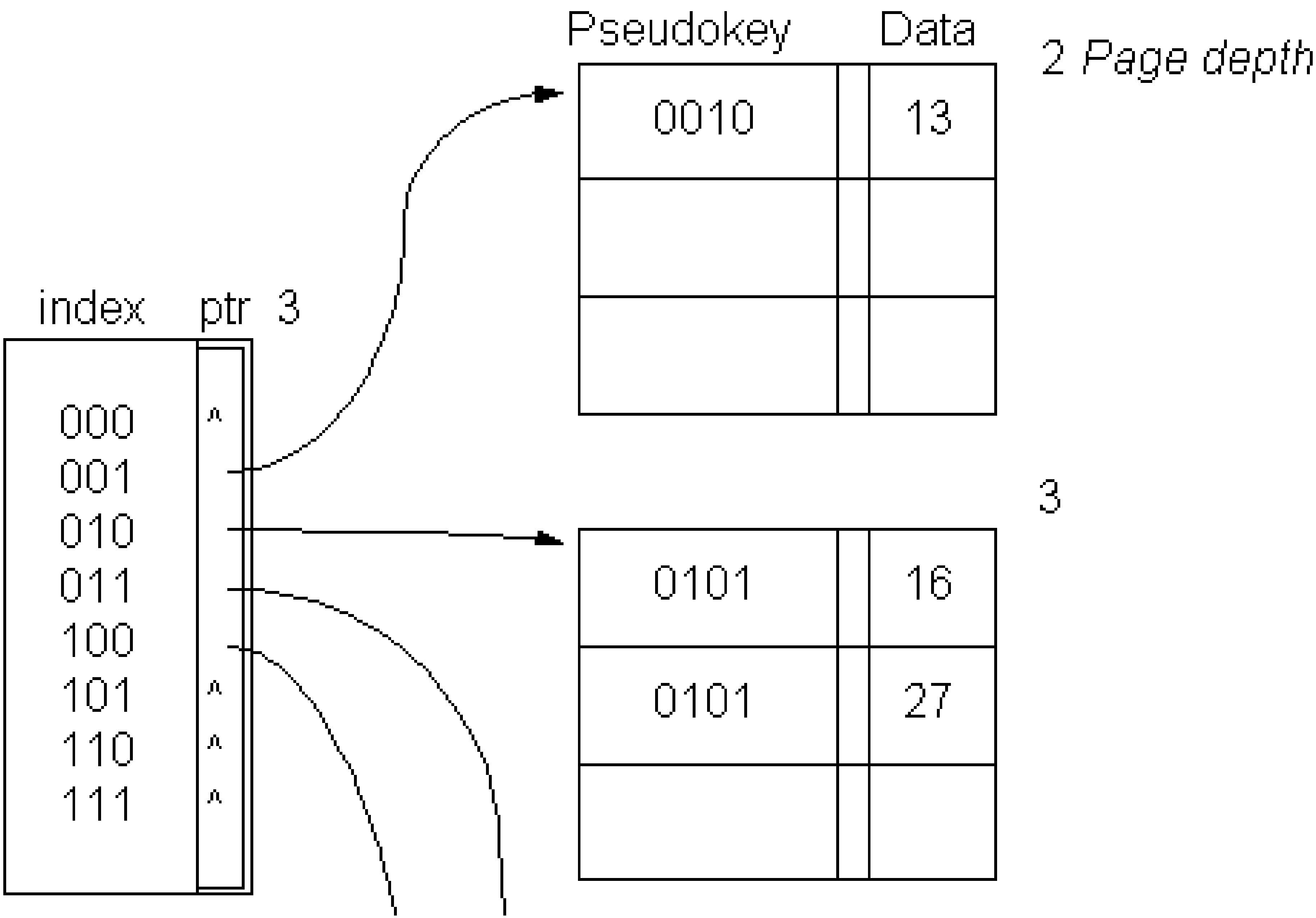
- Insert 13, 16

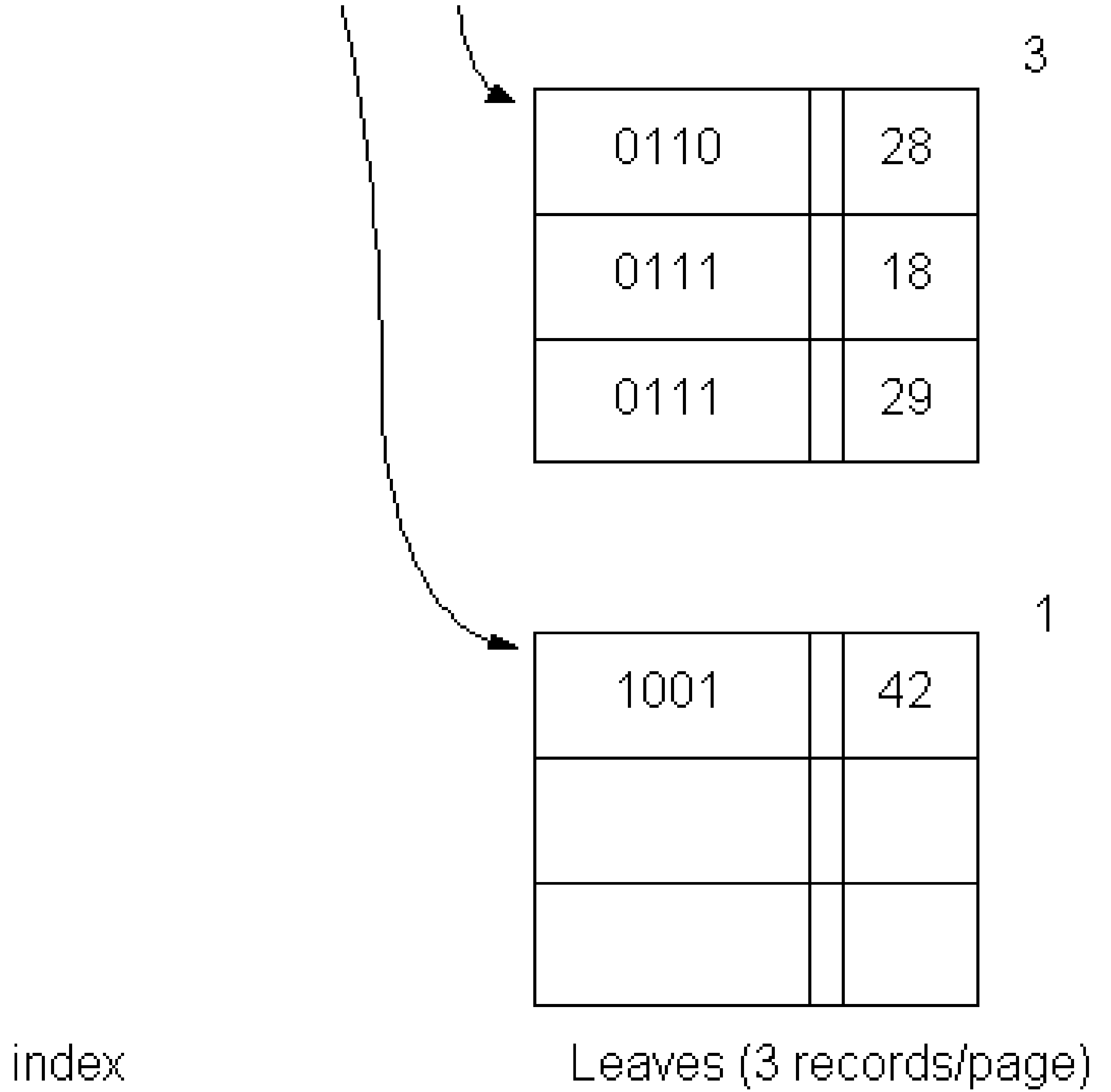

Pseudokey
Data
3 Page depth
0101
27
0010
13
0101
16
index depth 3
000
001
010
011
100
101
110
111
3
0110
28
0111
18
0111
29
1
1001
42
index
Leaves (3 records/page)

Another Implementation of Extendible Hashing

- A pointer or null pointer to a data page
- All records on a data page have the same N most significant bits in their pseudokeys

- Delete 16, 29

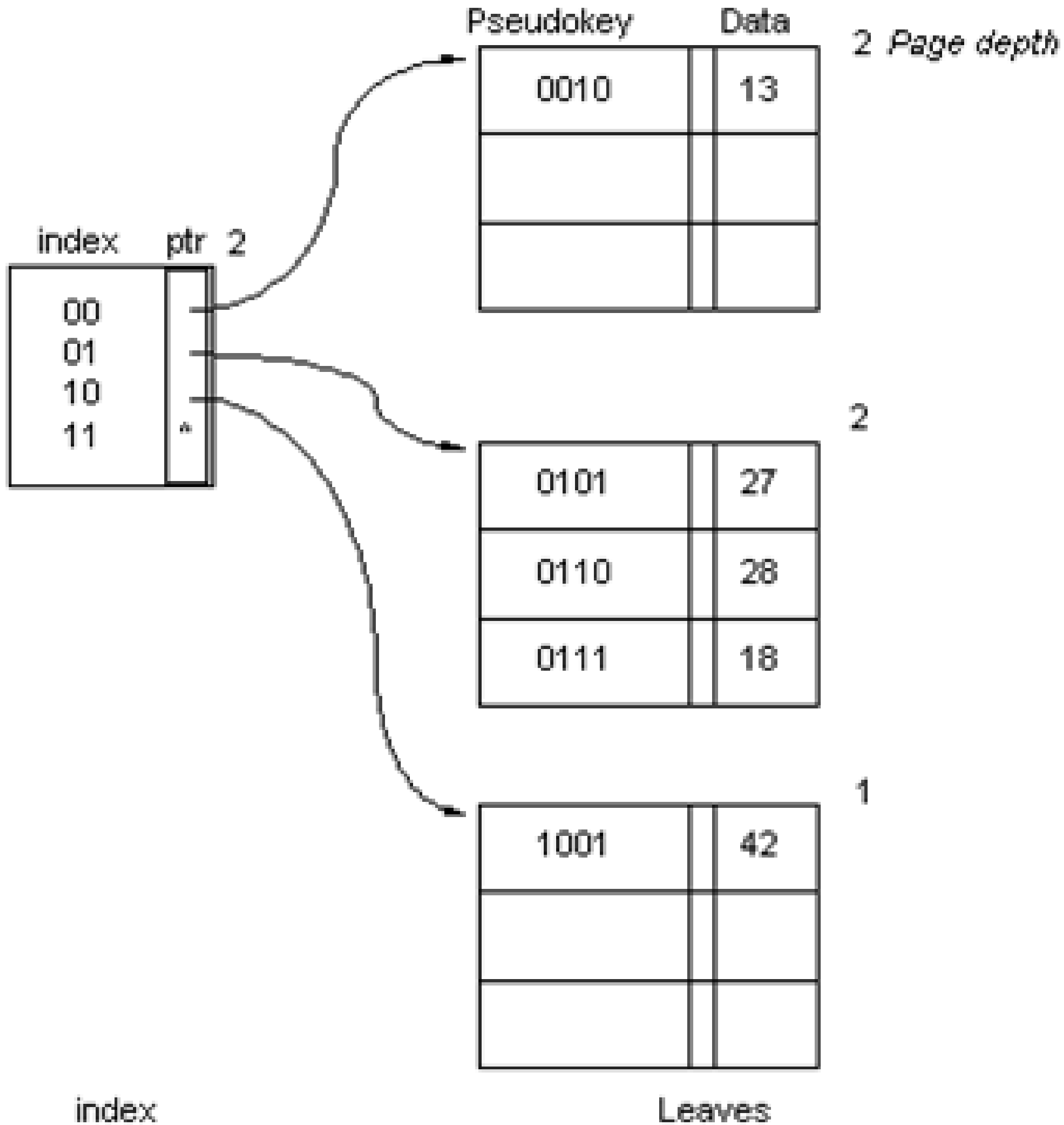

Pseudokey
Data
2 Page depth
0010
13
index
ptr 3
000
001
010
011
100
101
110
111
^
^
^
^
3
0101
16
0101
27


3
0110
28
0111
18
0111
29
1
1001
42
index
Leaves (3 records/page)

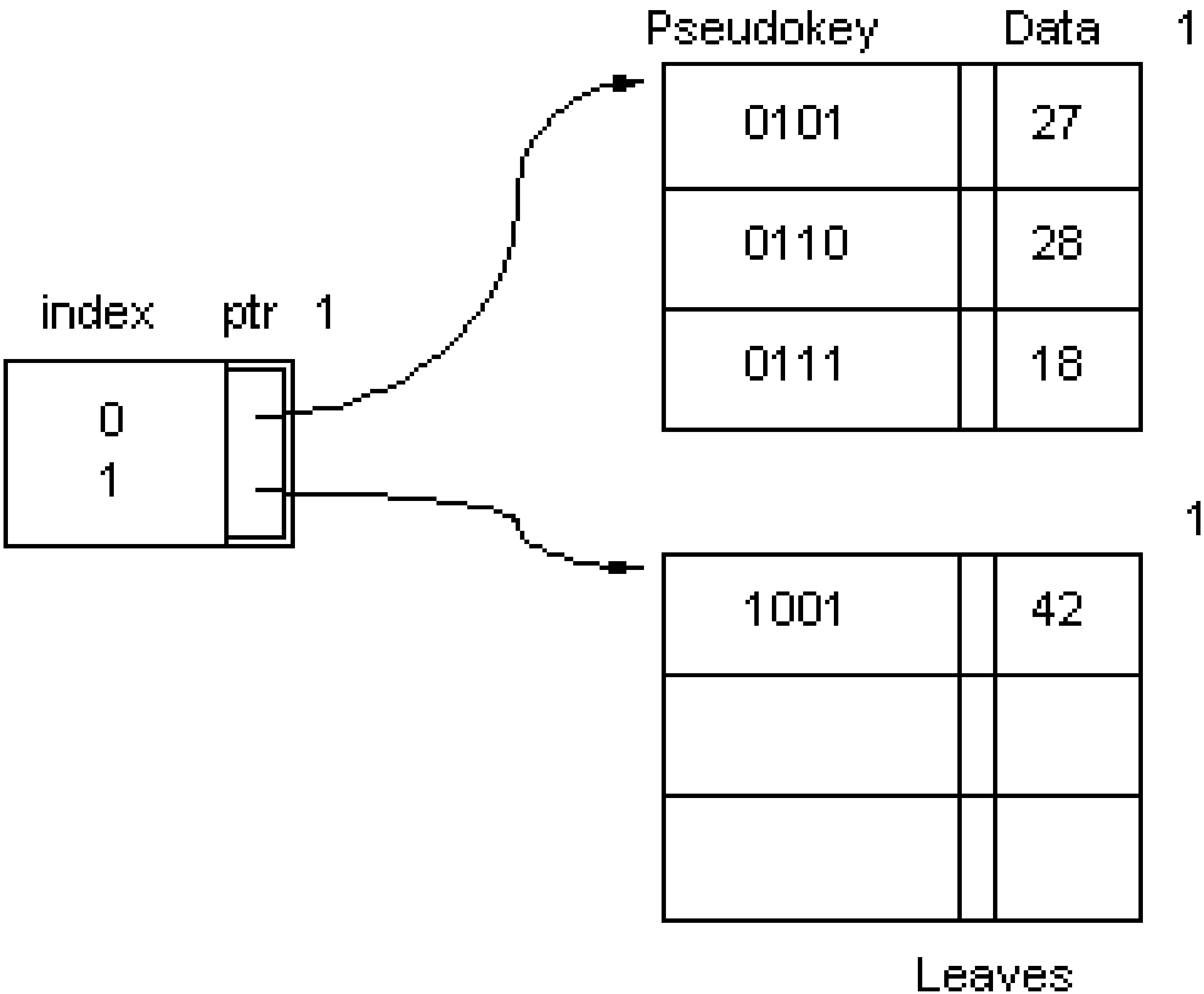
Deletion

- Page coalesce → reduce the amount of storage
  → reduce the depth of the index
- delete record → check buddy page

- Delete 13



index
ptr
2
00
01
10
11
*
Pseudokey
Data
2 *Page depth*
0010
13
2
0101
27
0110
28
0111
18
1
1001
42
index
Leaves


Pseudokey
Data
1
0101
27
0110
28
0111
18
index
ptr
1
0
1
1
1001
42
Leaves

Dynamic Hashing

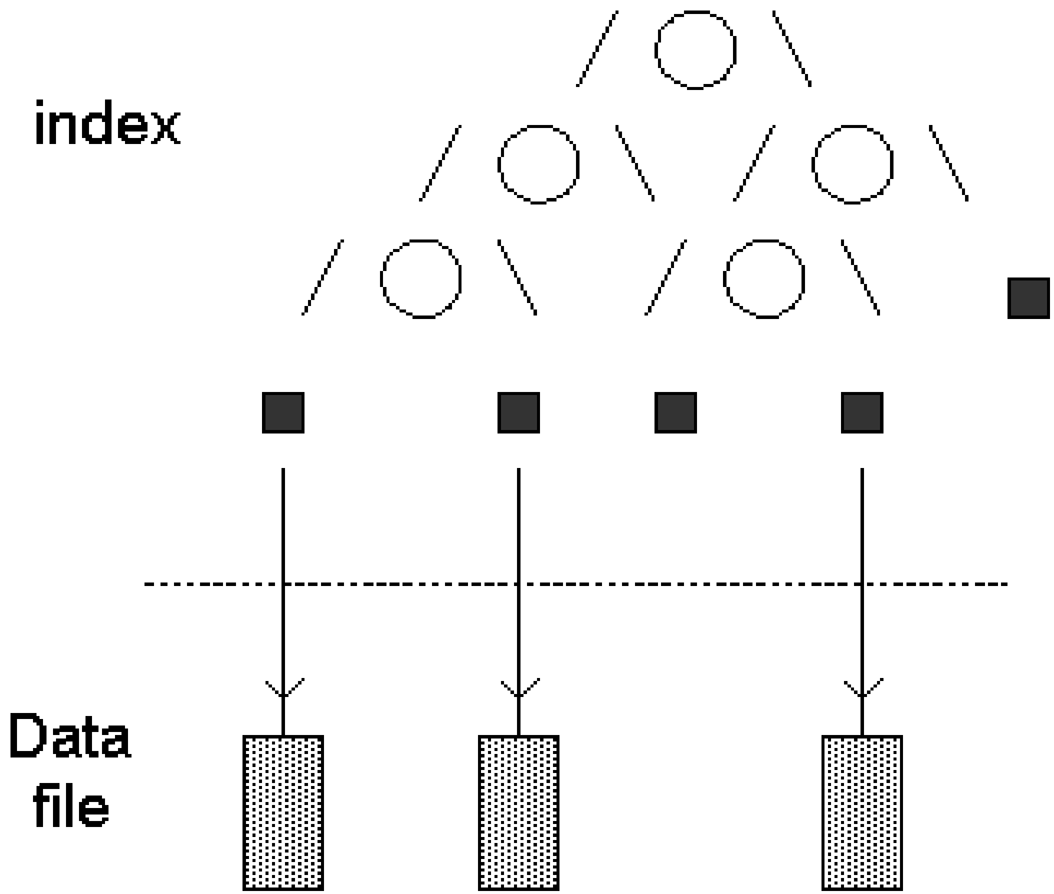
- Index grows gradually (not doubling)
- Use pseudo_random_number generator
  → $H_1(key_i)$
  → $B(H_1(key_i))=(b_{i0}, b_{i1}, b_{i2}, \ldots)$ $b_{ij}=\{0,1\}$ for all j
- Use forest of binary trees
  - $H_0(key_i)$ → $\{0, 1, \ldots, n\}$ determine which subtree

Algorithm

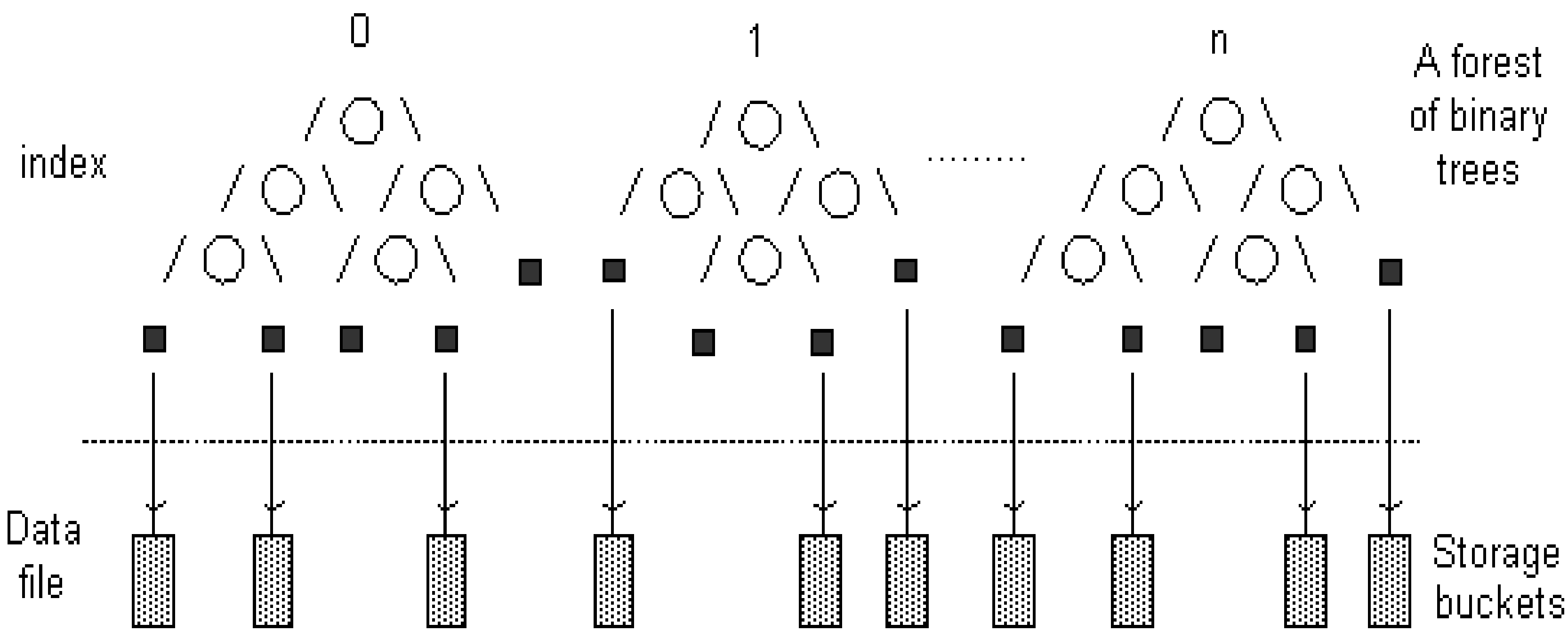
- Dynamic Hashing Insertion

1. H_0(key) = subdirectory
2. The current node is not an external node
   → use $B(H_1(\text{key}))$ to navigate subdirectory
   → 0->left, 1->right
3. if not full, insert the new record, else repeat until an overflow no longer exits
   A. external → internal node
      create two offspring external nodes
   B. reinsert the record using the next bit of
      pseudorandom sequence

- $H_1(Key_1)$
- $B(H_1(Key_1)) = (b_{i0}, b_{i1}, b_{i2}, \ldots)\ b_{ij} \in \{0,1\}$ for all j



– $H_0(Key_i)$ – {0, 1, …, n} (forest)



Example

- Data 27, 18, 29, 28, 39, 13, 16, 36
- H0 = key mod 3, H1 = key mod 11
- Data page = 2 records

PSEUDORANDOM BIT SEQUENCES FOR SEEDS 0-10

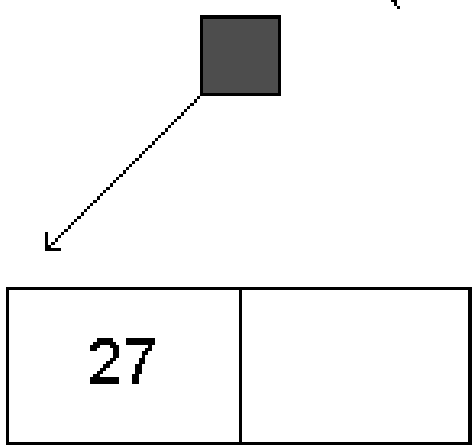
B(0) = 1011	B(5) = 0101
B(1) = 0000	B(6) = 0001
B(2) = 0100	B(7) = 1110
B(3) = 0110	B(8) = 0011
B(4) = 1111	B(9) = 0111
B(10) = 1001	

Insert 27

0 (subdirectory)

$H_0(27) = 0$

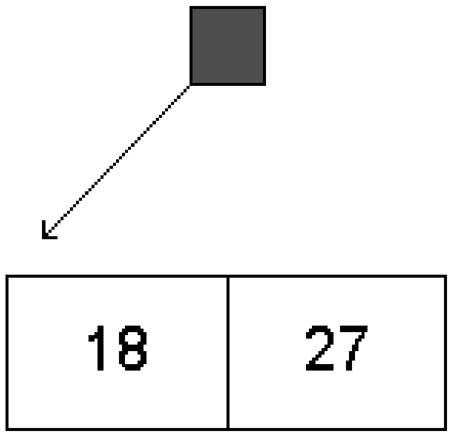
$H_1(27) = 5$ B(5) = 0101



Insert 18 0 (subdirectory)

$H_0(18) = 0$

$H_1(18) = 7$ $B(7) = 1110$



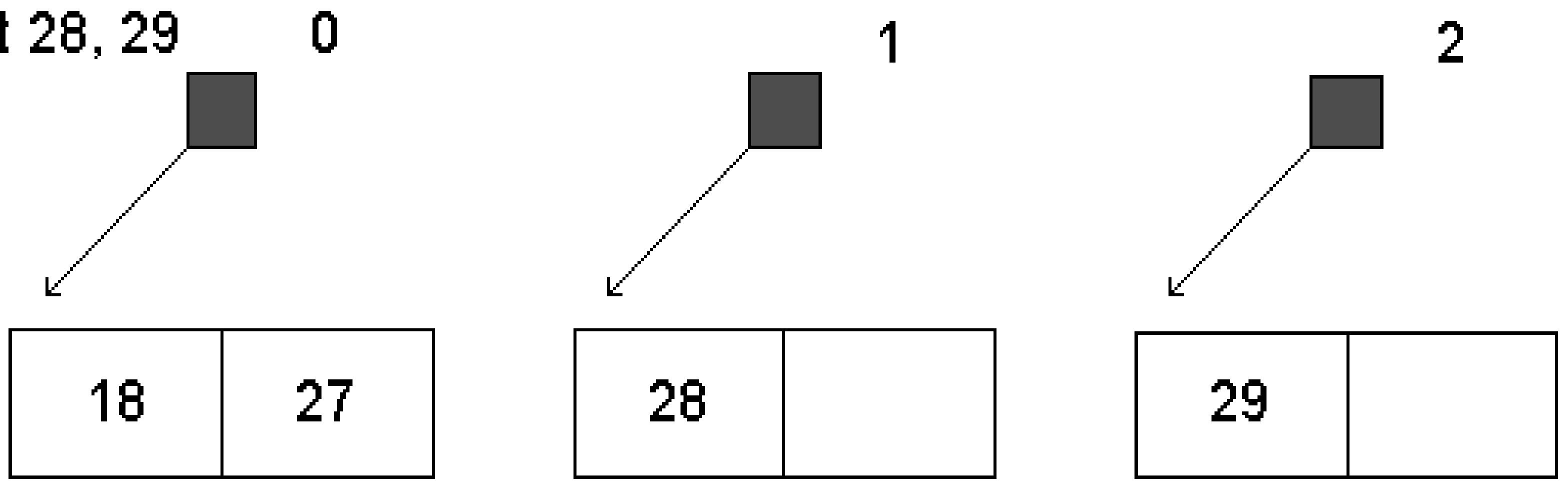
$H_0(28) = 1$

$H_1(28) = 6$ $B(6) = 0001$

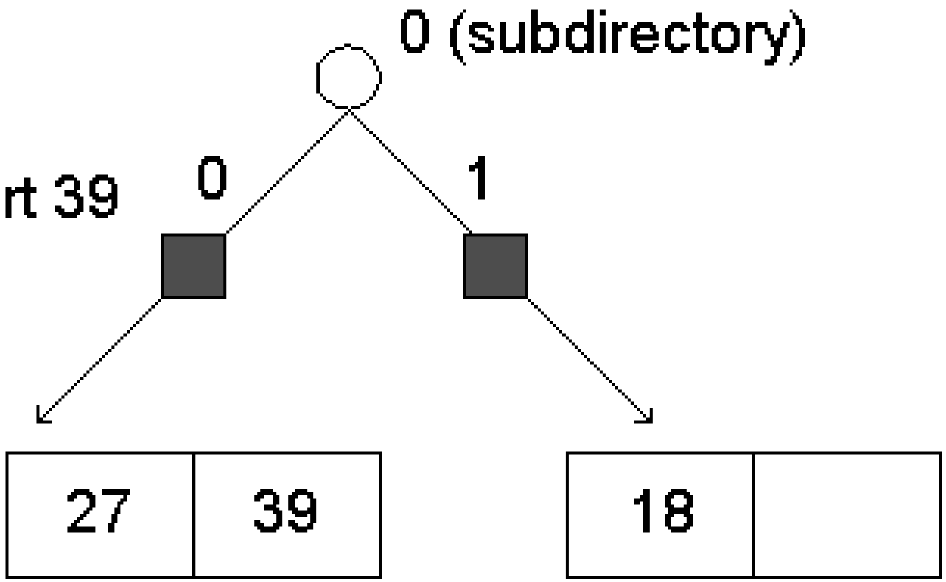
$H_0(29) = 2$

$H_1(29) = 7$ $B(7) = 1110$

Insert 28, 29



Insert 39

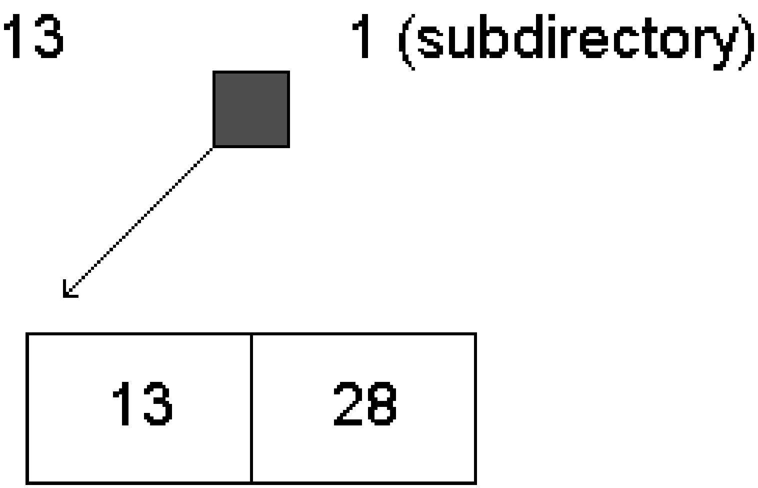


$H_0(39) = 0$

$H_1(39) = 6$

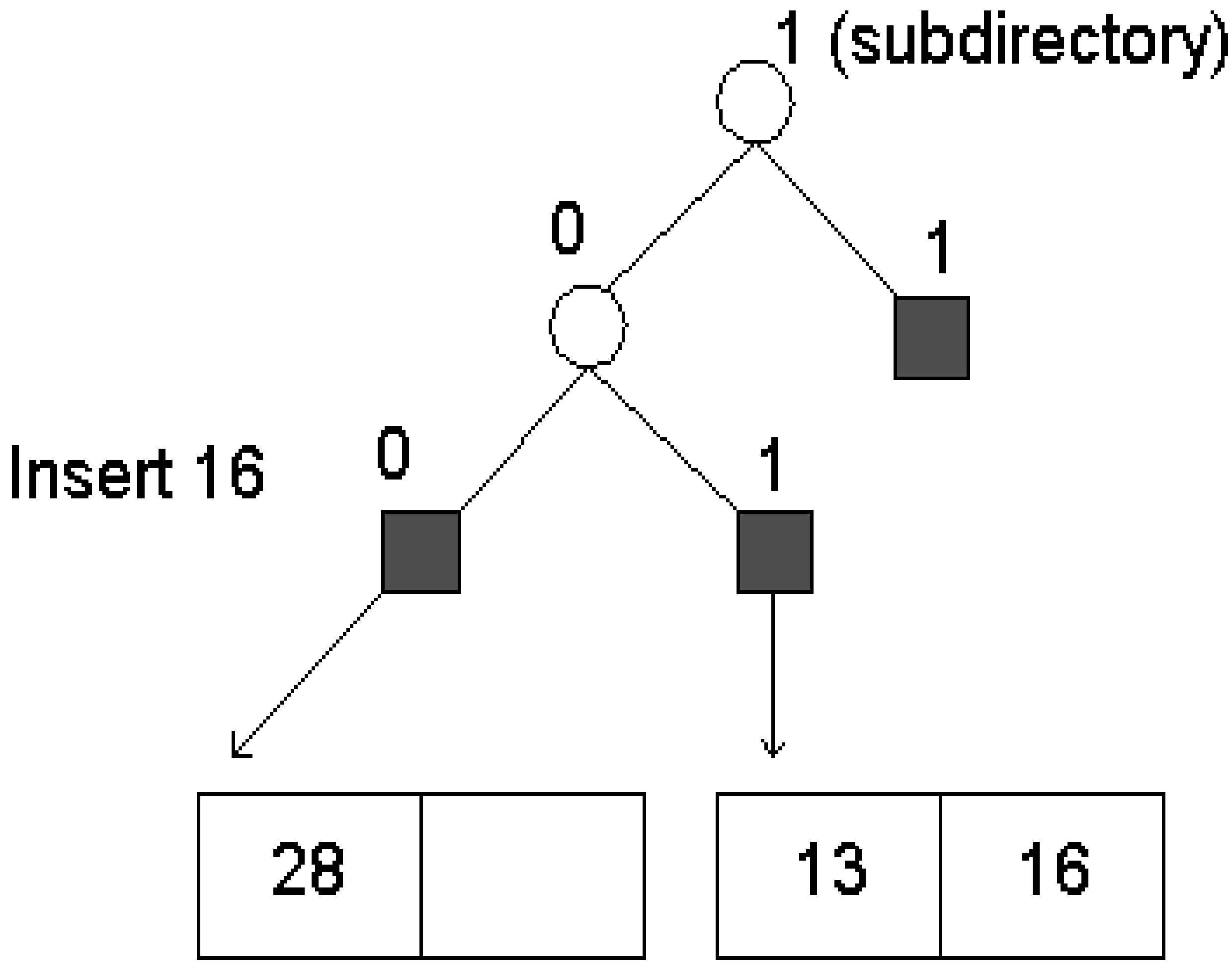
$B(6) = 0001$

Insert 13



$H_0(13) = 1$

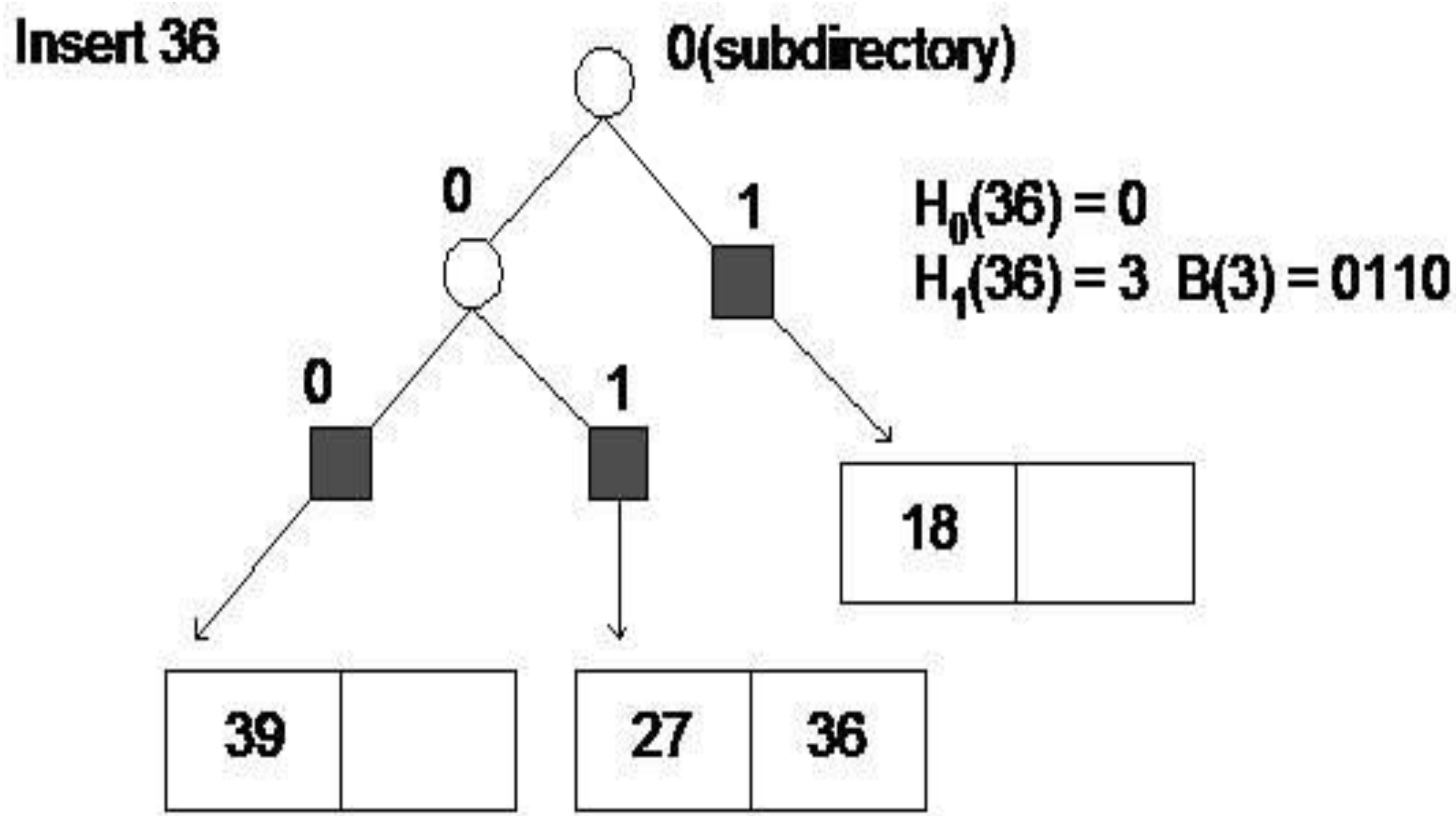
$H_1(13) = 2 \quad B(2) = 0100$



$H_0(16) = 1$

$H_1(16) = 5$

$B(5) = 0101$


Insert 36
0(subdirectory)
0
1
$H_0(36) = 0$
$H_1(36) = 3$ B(3) = 0110
0
1
18
39
27
36

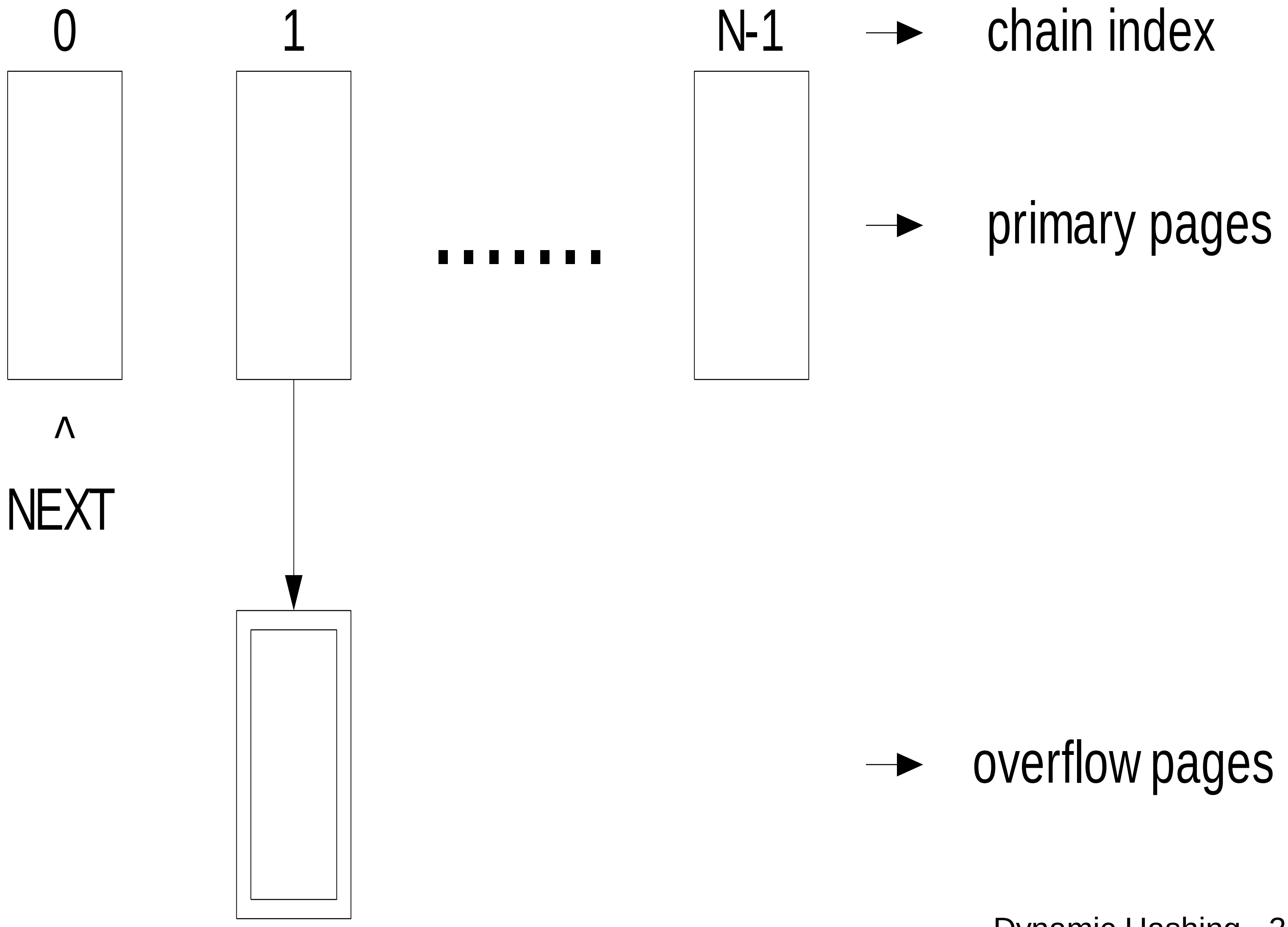
Linear Hashing

- To permit file expansion without reorganization.
- No index.
- How to allow the file expand?

  - Changing the hashing function.

  - Has two hashing functions active at a time.

structure



- A pointer NEXT points the next chain to be split
- N is the number of chains initially; fixed
- H_{level} is the hashing function for the current level
- m is the output from the hashing function

Algorithm

1. Determine the chain, m, which the record maps to using $m = h_{level}(key)$.
2. Check where the chain has split by comparing m with NEXT, if m < NEXT, then $m = h_{level+1}(key)$.
3. Insert the record into chain m.

4. Check the upper space utilization bound, while it is exceeded then

   4.1 Create a new chain with index equal to NEXT + $N*2^{level}$

   4.2 For each record on the chain NEXT, determine whether to move it.

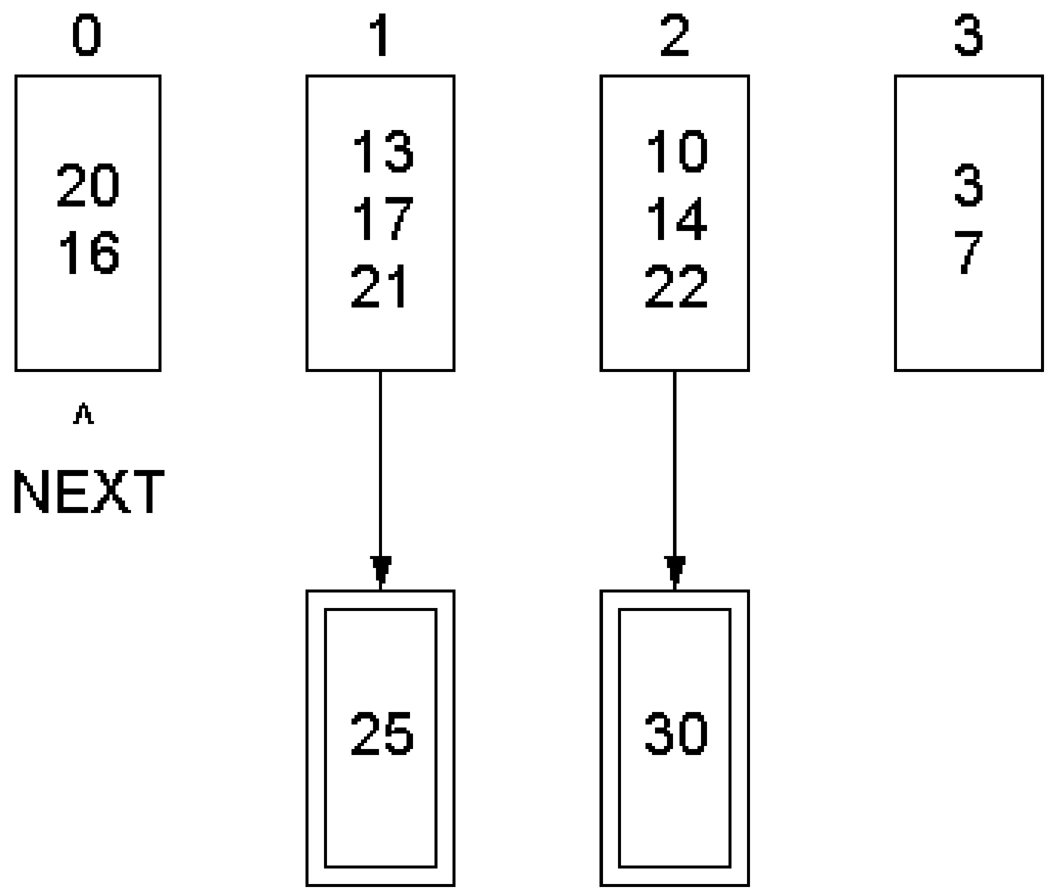
   4.3 Update parameters, NEXT = NEXT + 1, if NEXT >= $N*2^{level}$, then reset NEXT to 0 and level = level + 1.

   - On each level, we split the chains in the order from 0 to the maximum chain ($N*2^{level}$ –1).
   - After all the chains on the current level have been split, we increment the current level and begin the split process over again with chain 0.
   - Where h_{level}(key) = key mod [$N*2^{level}$]

– Example 1:

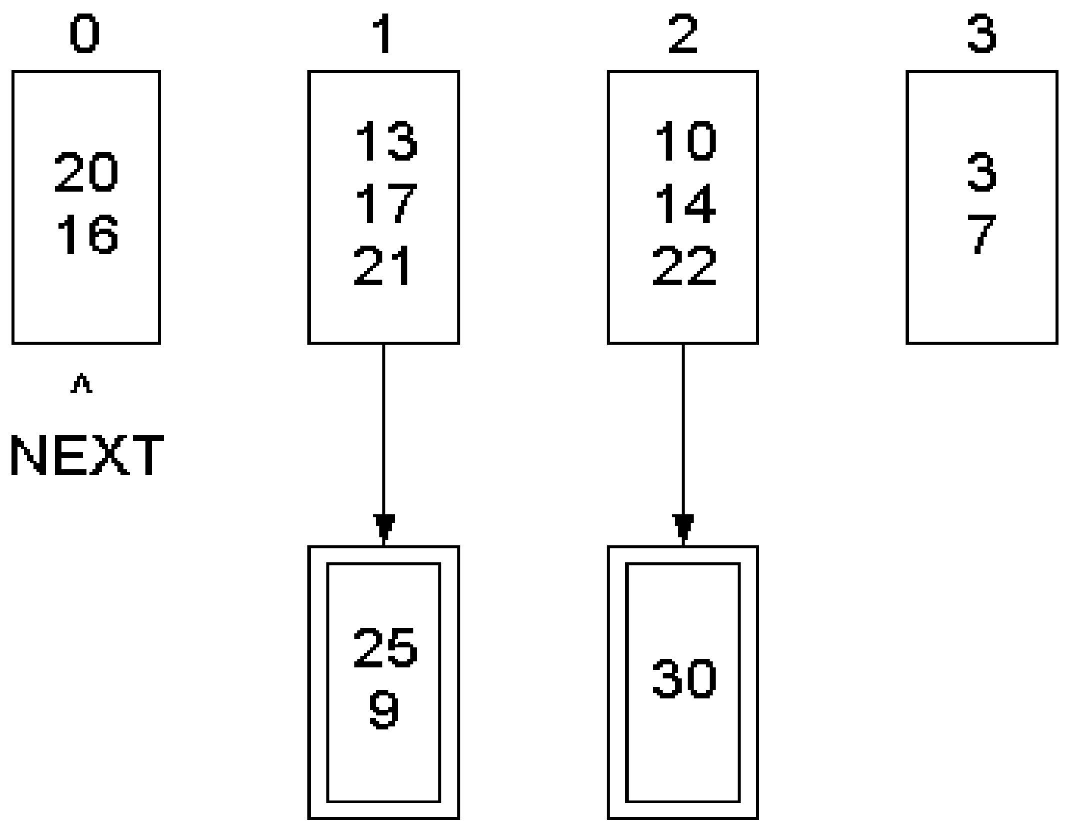
(40% <-> 80%)

$h_0(key) = key \bmod 4$

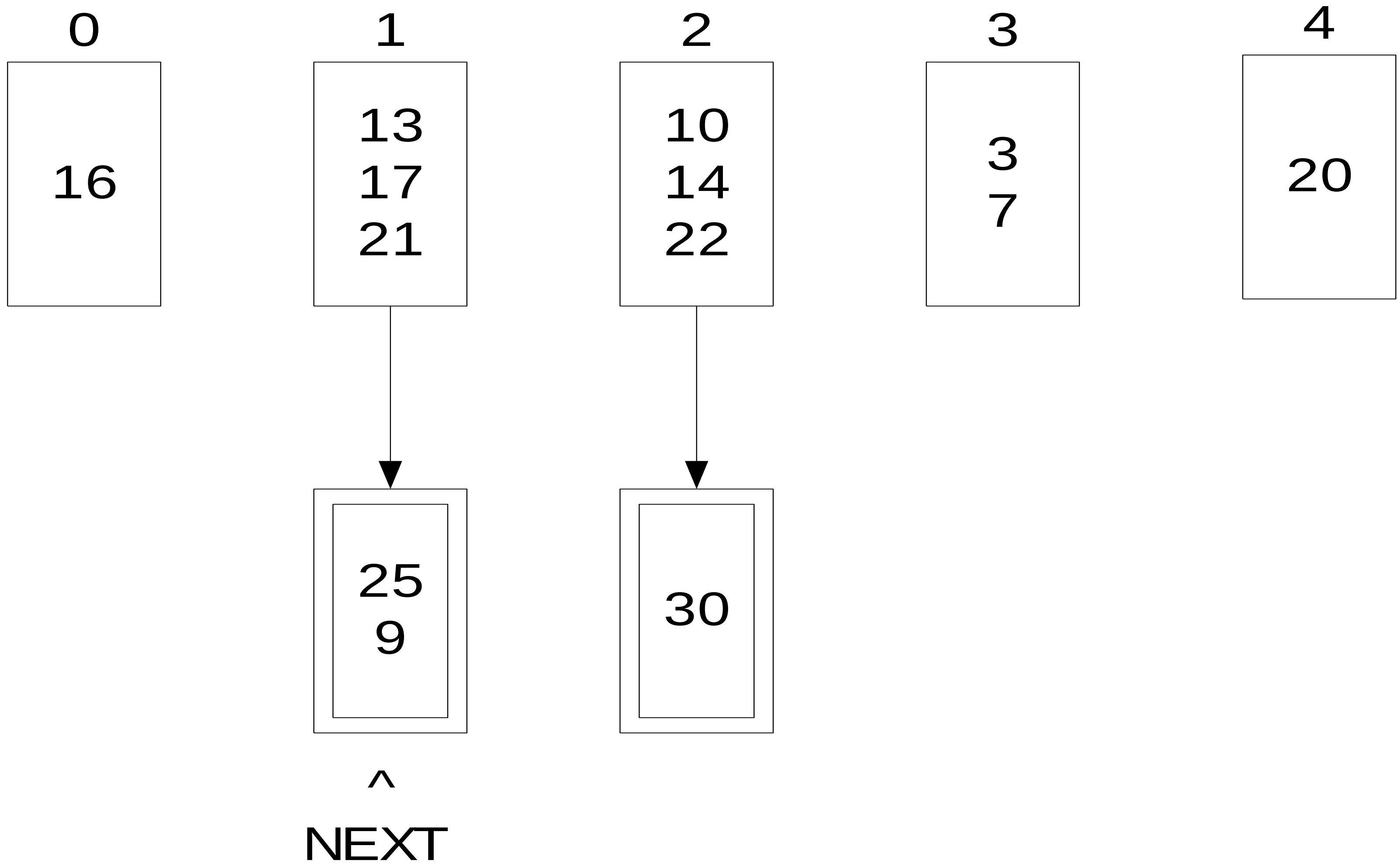


- Insert a record with a key of 9

$h_0(9) = 1$



- Storage utilization = 81%
- To split the NEXT chain



- To retrieve a record with key 20

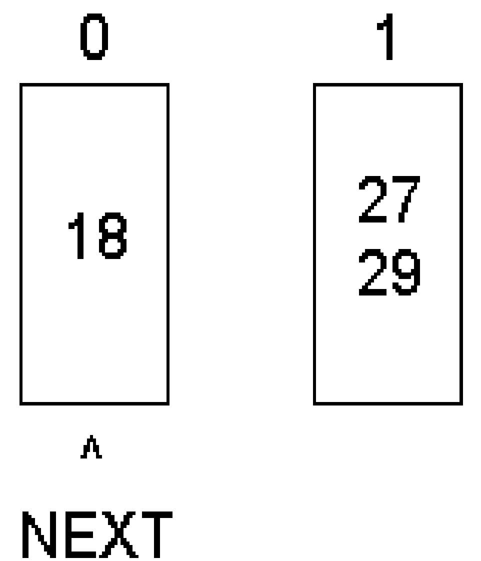
  1. Apply h0 => h0(20) = 0 < (NEXT = 1)
  2. Apply h1 => h1(20) = 20 mod (4*2) = 4
  3. We locate 20 on chain 4.

– Example 2:

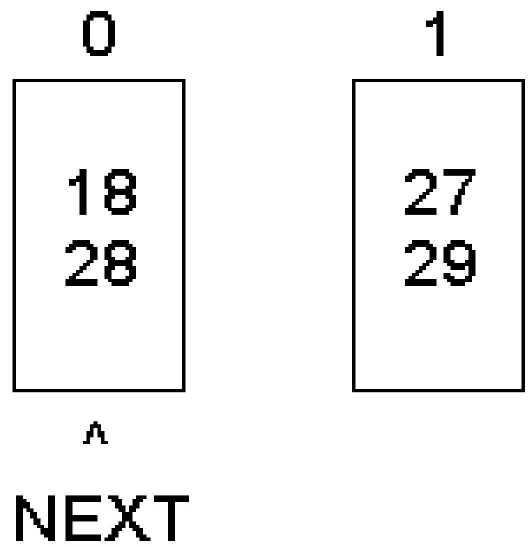
The keys that we insert are 27, 18, 29, 28, 39, 13, 16, 51, 19.

- 40% <-> 80%
- primary pages hold two records
- overflow pages hold one record
- N = 2

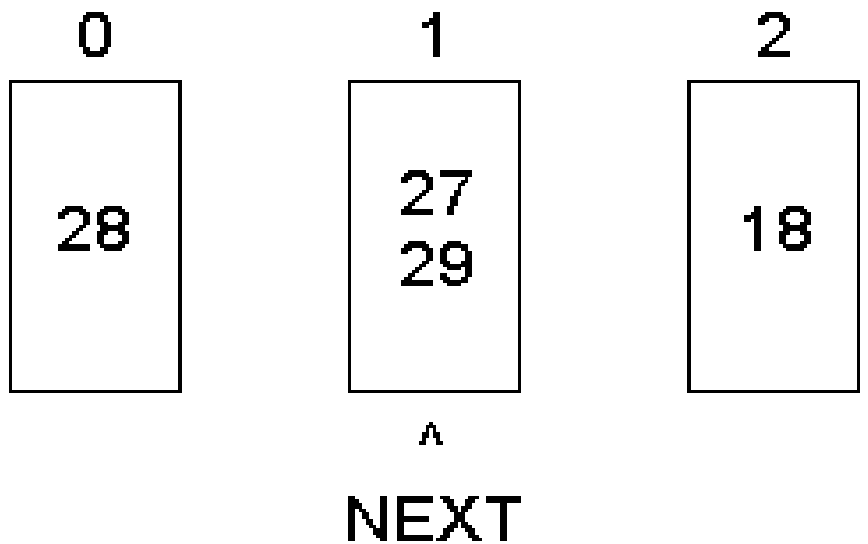
1) $h_0(27) = 1$
2) $h_0(18) = 0$
3) $h_0(29) = 1$



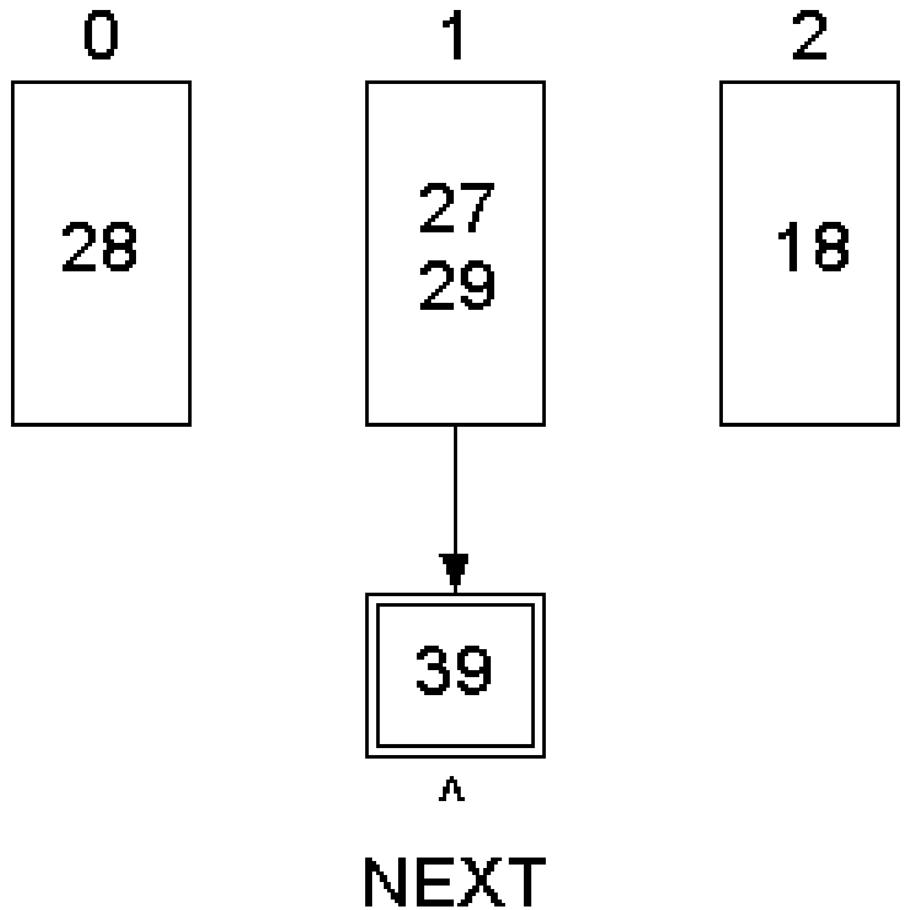
4) $h_0(28) = 0$



- Utilization = 100%
- Split 0

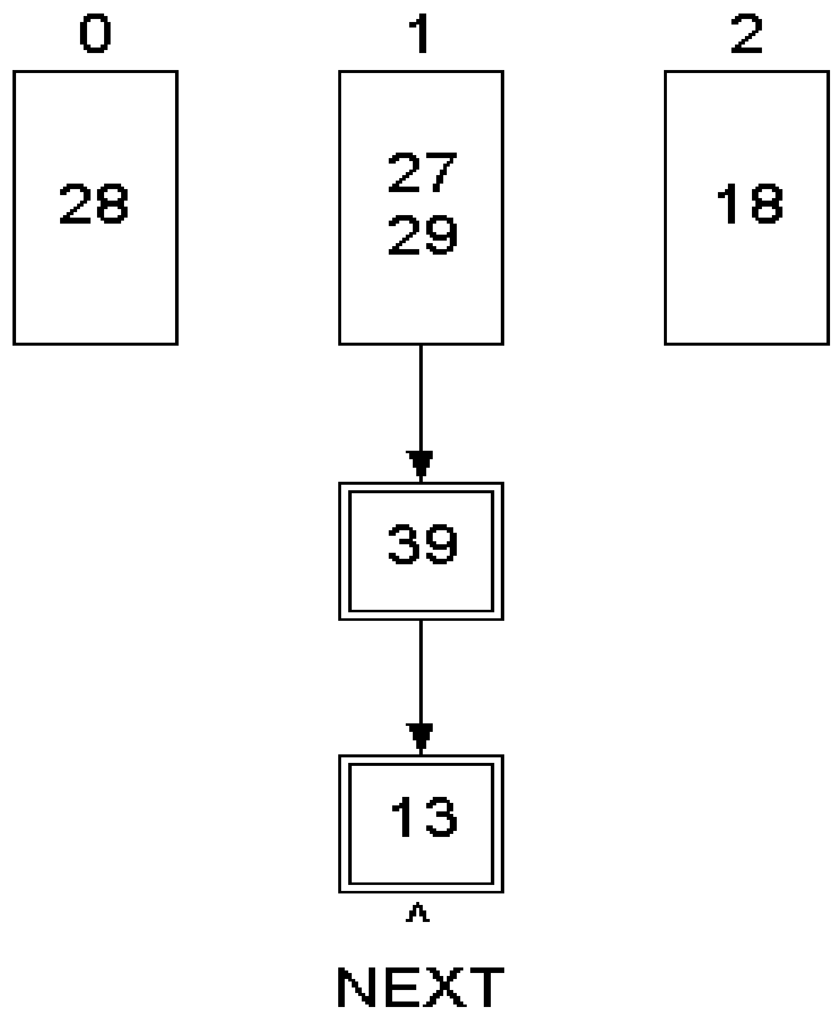


5) $h_0(39) = 1$

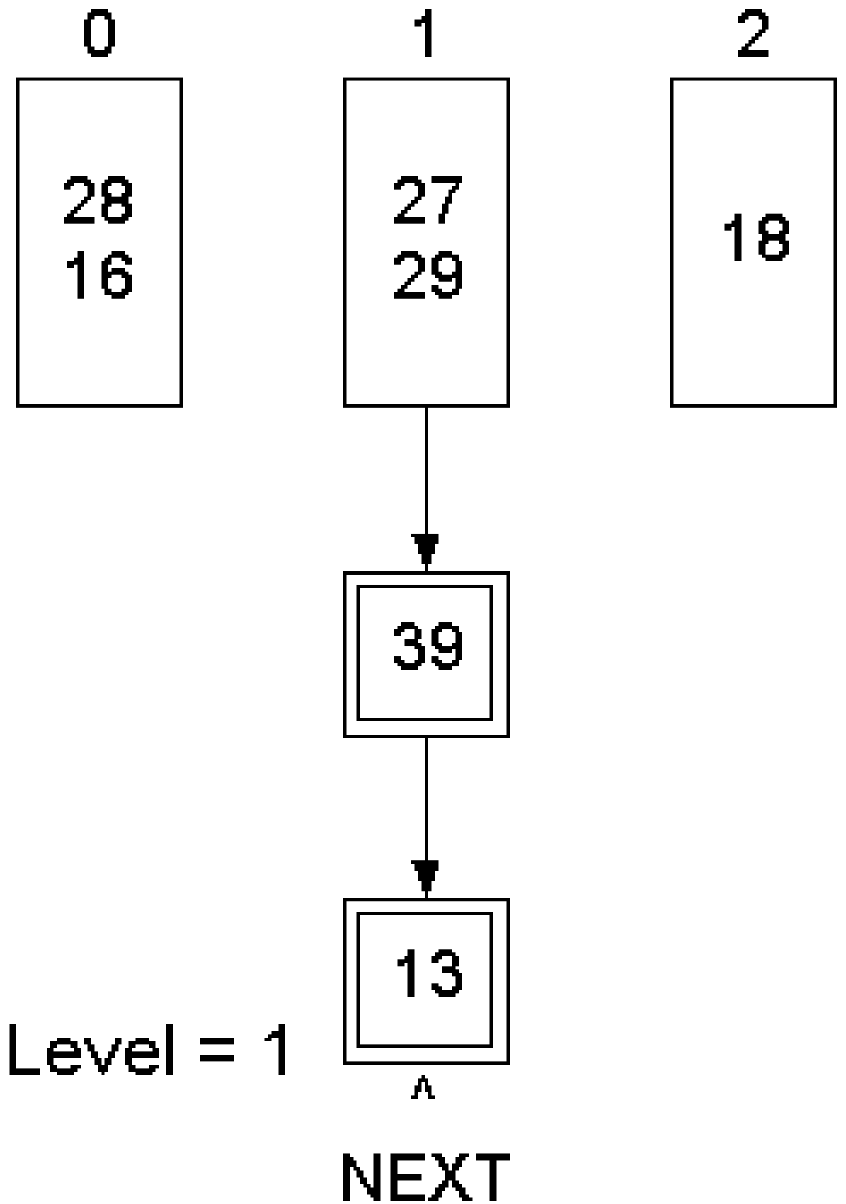


- Utilization = 71%

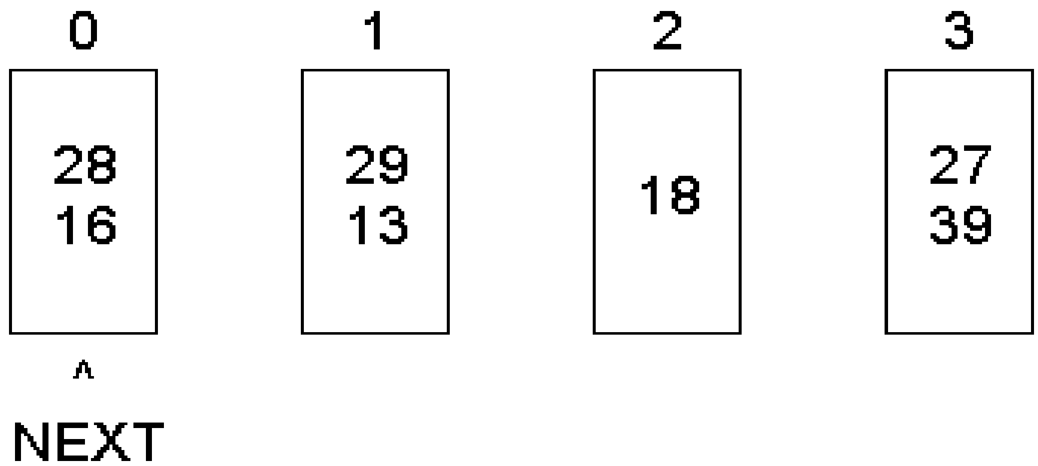
6) $h_0(13) = 1$



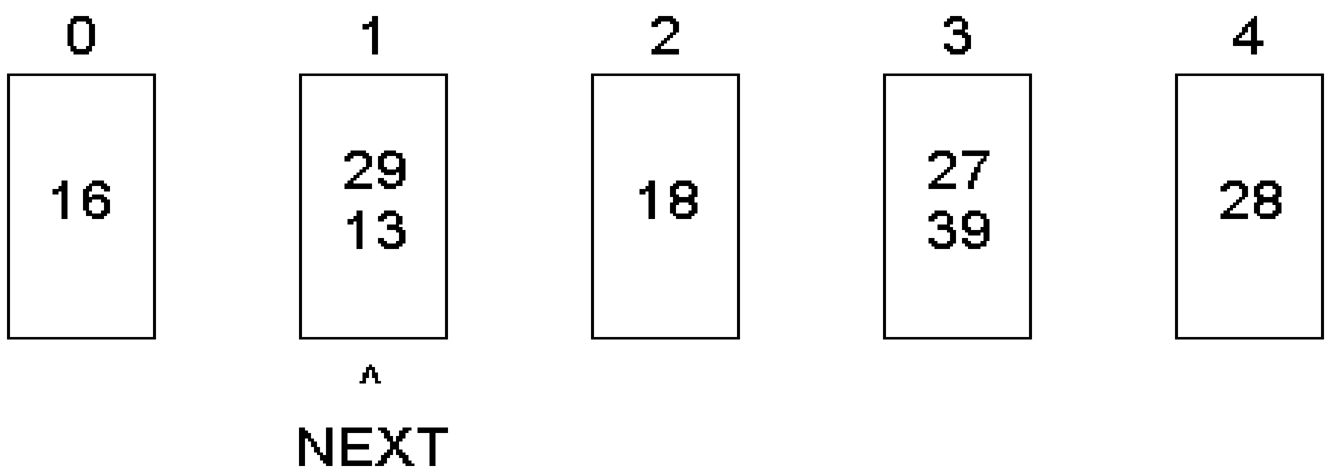
7) $h_0(16) = 0$



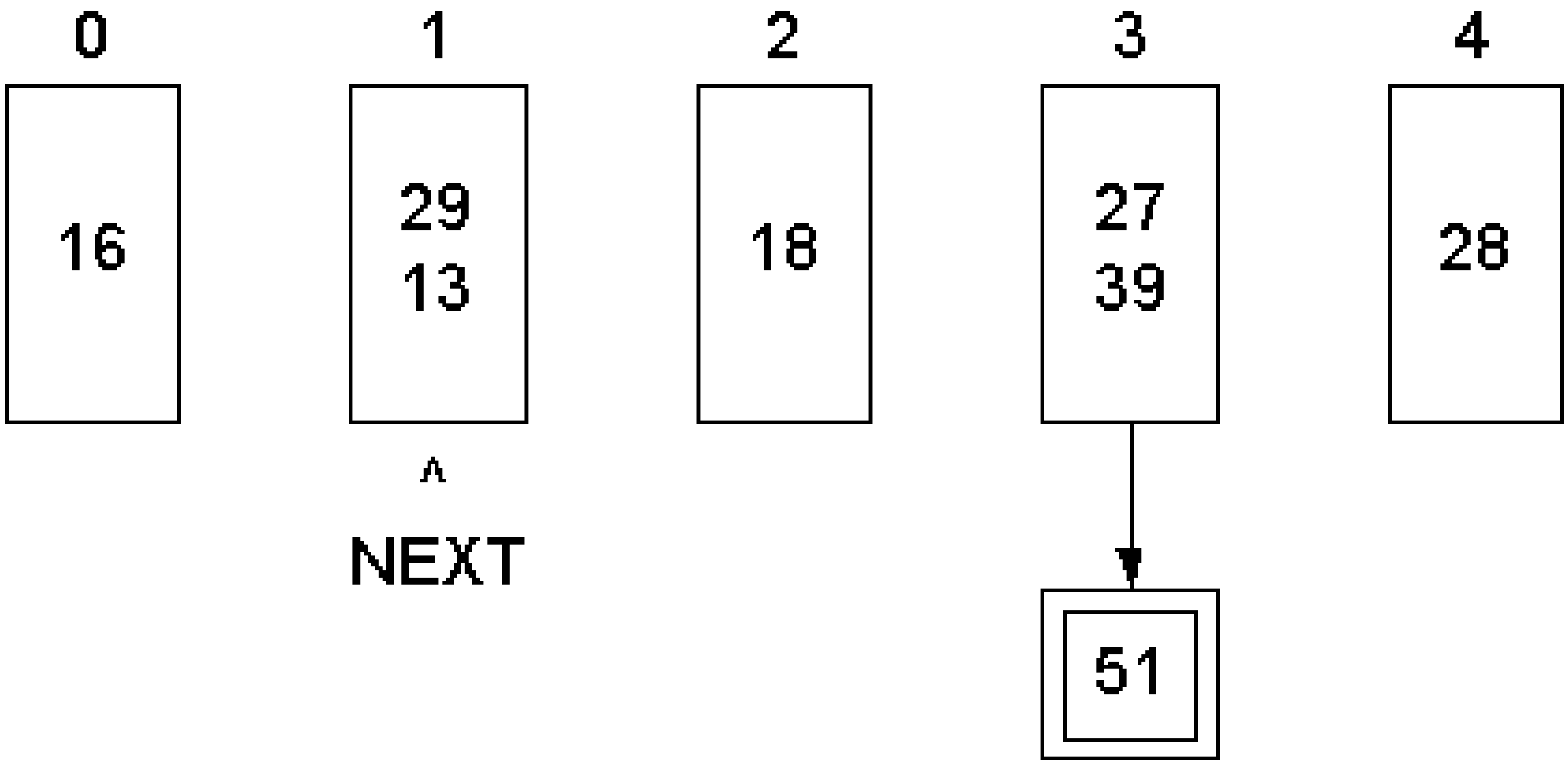
- Utilization = 87%
- Split 1
- NEXT >=2,NEXT = 0,Level = 1



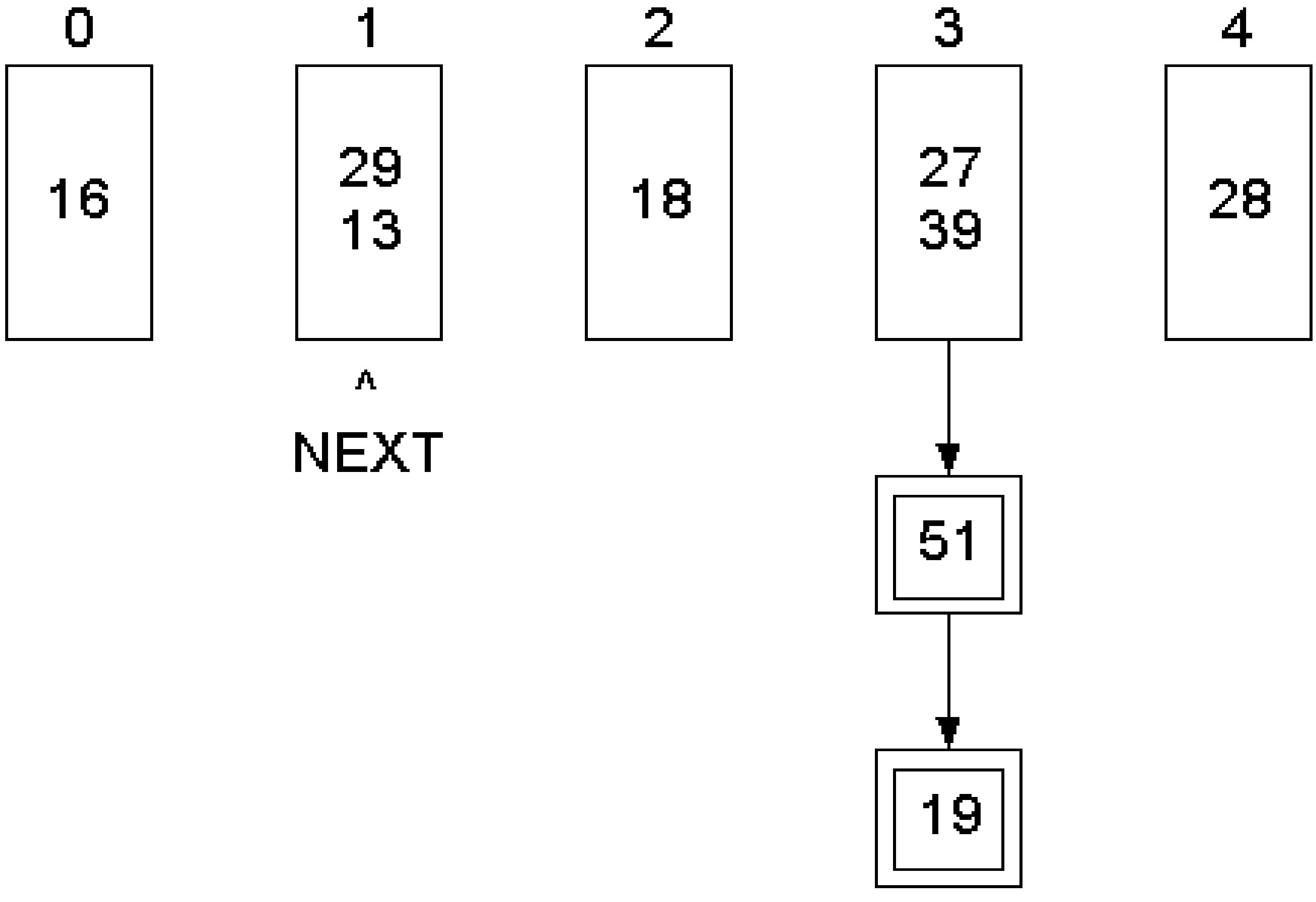
- Split 0



8) $h_1(51) = 3$



9) $h_1(19) = 3$



Discussion

1)Retrieve 28

Current level = 1

h1(28) = 0 < NEXT (=1) ➔ h2(28) = 4

2)Retrieve 19

h1(19) = 3 >= NEXT (=1)