Hash-Based Indexes

Chapter 11

Index Design Space

Organization Structure for k*

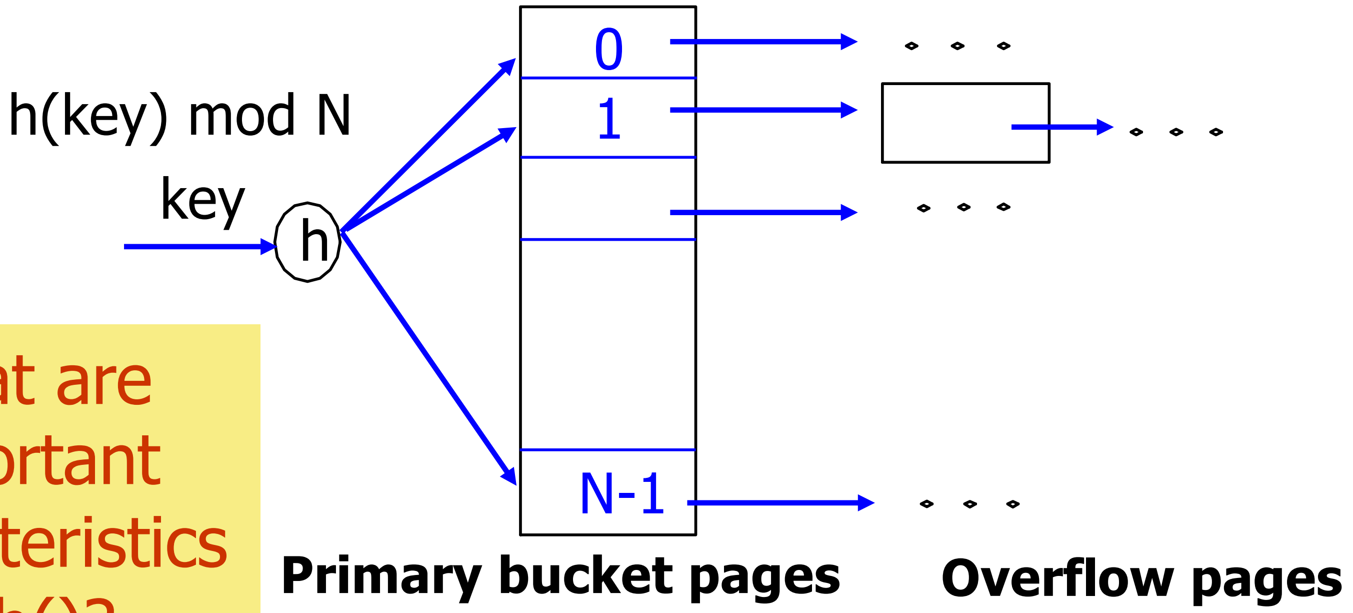
- **Hash-based**
  - + Equality search
    - Static hashing
    - Extensible hashing
    - Linear hashing
- **Tree-based**
  - + Range, equality search

Data Entry (k*) Contents

1. Actual Data record
   - index = file
2. <k, rid>
   - actual records in a different file
3. <k, list of rids>

Static Hashing

- # primary bucket pages fixed, allocated sequentially, never de-allocated; overflow pages if needed.
- **h**(*key*) mod N = bucket to which data entry with *key* belongs. (N = # of buckets)



What are important characteristics of h()?

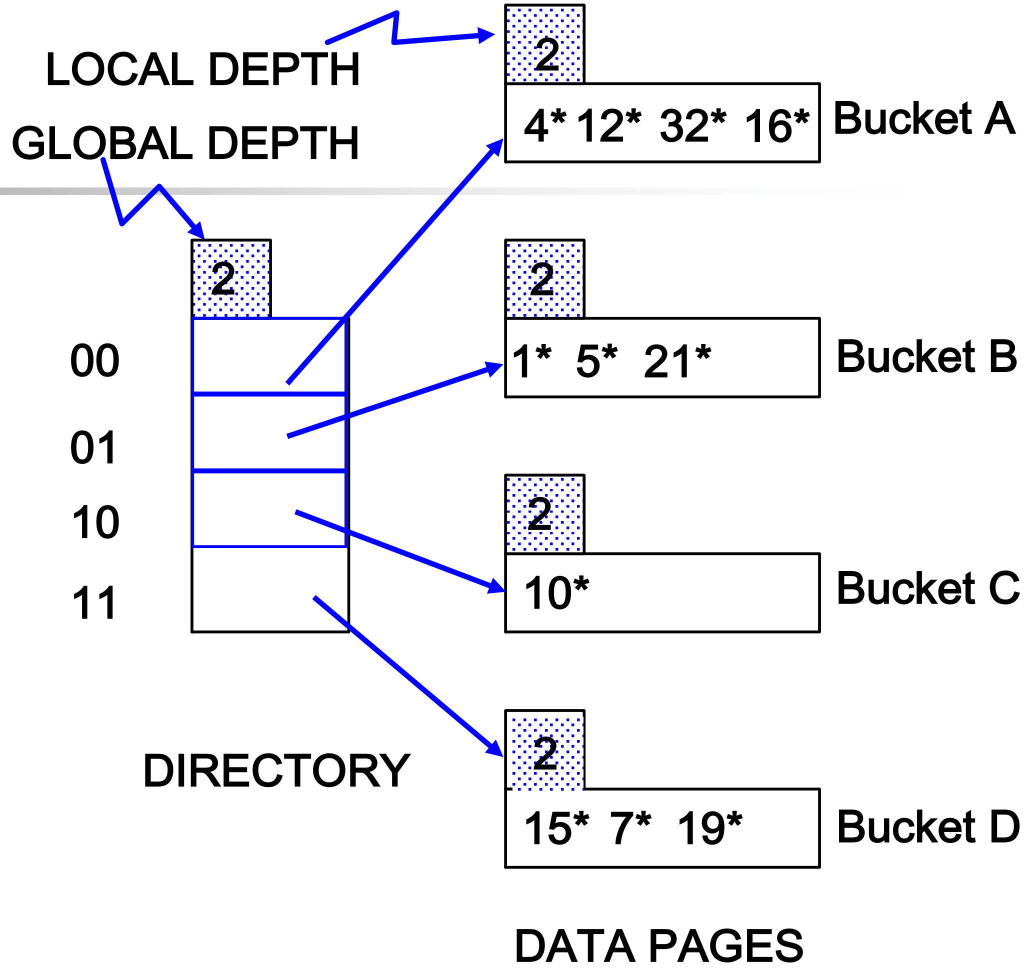
Static Hashing (Contd.)

- Buckets contain *data entries*
- Number of buckets (N) is fixed ahead of time
- Static structure can be problematic
  - Consider many insertions
  - Long overflow chains can develop (and degrade performance!)
- Might consider periodically doubling N and “rehashing” file
  - Entire file has to be read and written
  - Index unavailable while rehashing
  - *Extensible* and *Linear Hashing*: Dynamic techniques to fix this problem.

Extensible Hashing

- **Main Idea:** Use a directory of pointers to buckets
- On overflow, double the directory (not # number of buckets)
- Why does this help?
  - Directory much smaller than file
  - Only one page of data entries is split at a time
  - No overflow pages

Example

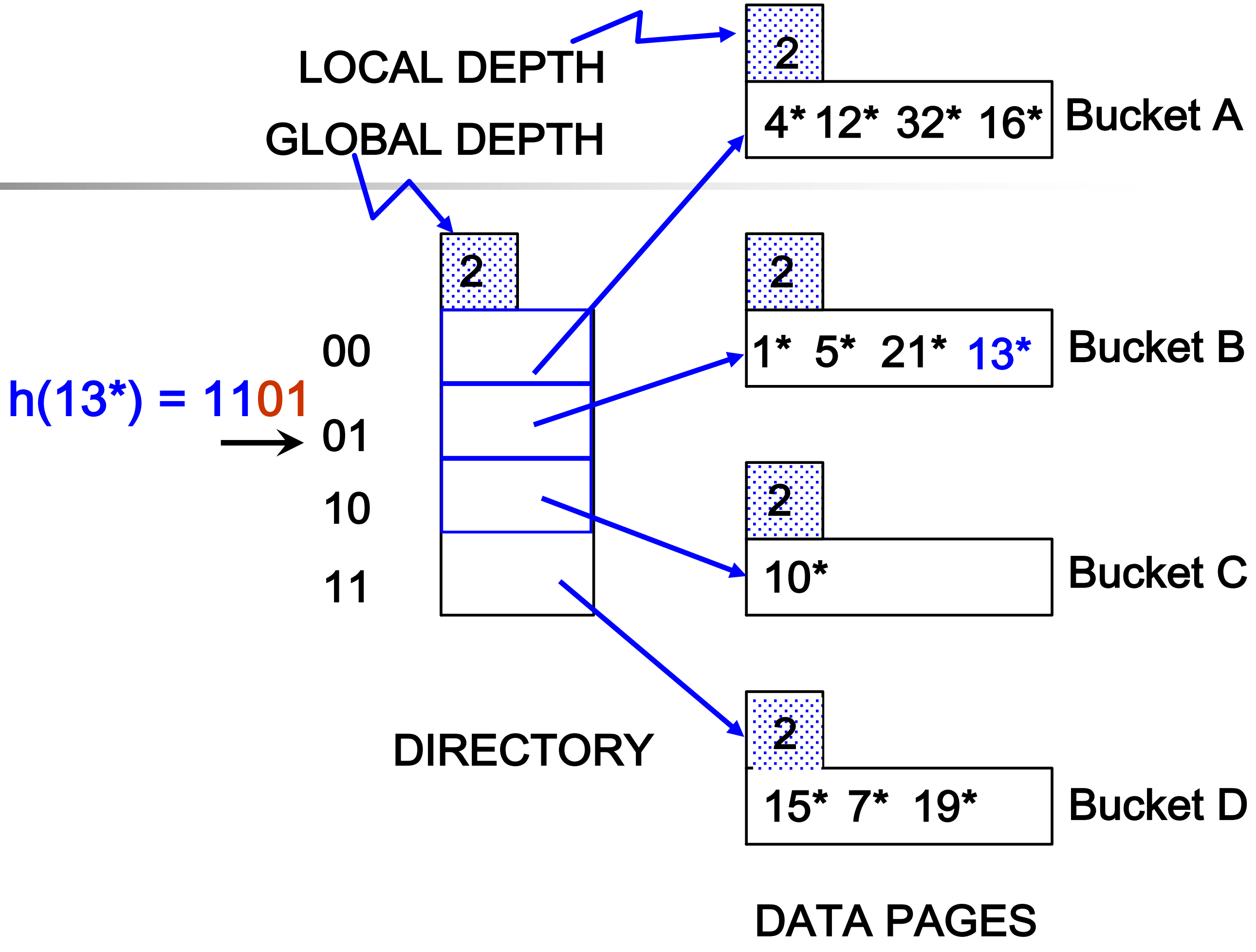


- Directory an array
- Search for k:
  - Apply hash function h(k)
  - Take last *global depth* # bits of **h**(*k*)
- Insert:
  - If bucket has space, insert, done
  - If bucket if full, split it, re-distribute
  - If necessary, double the directory

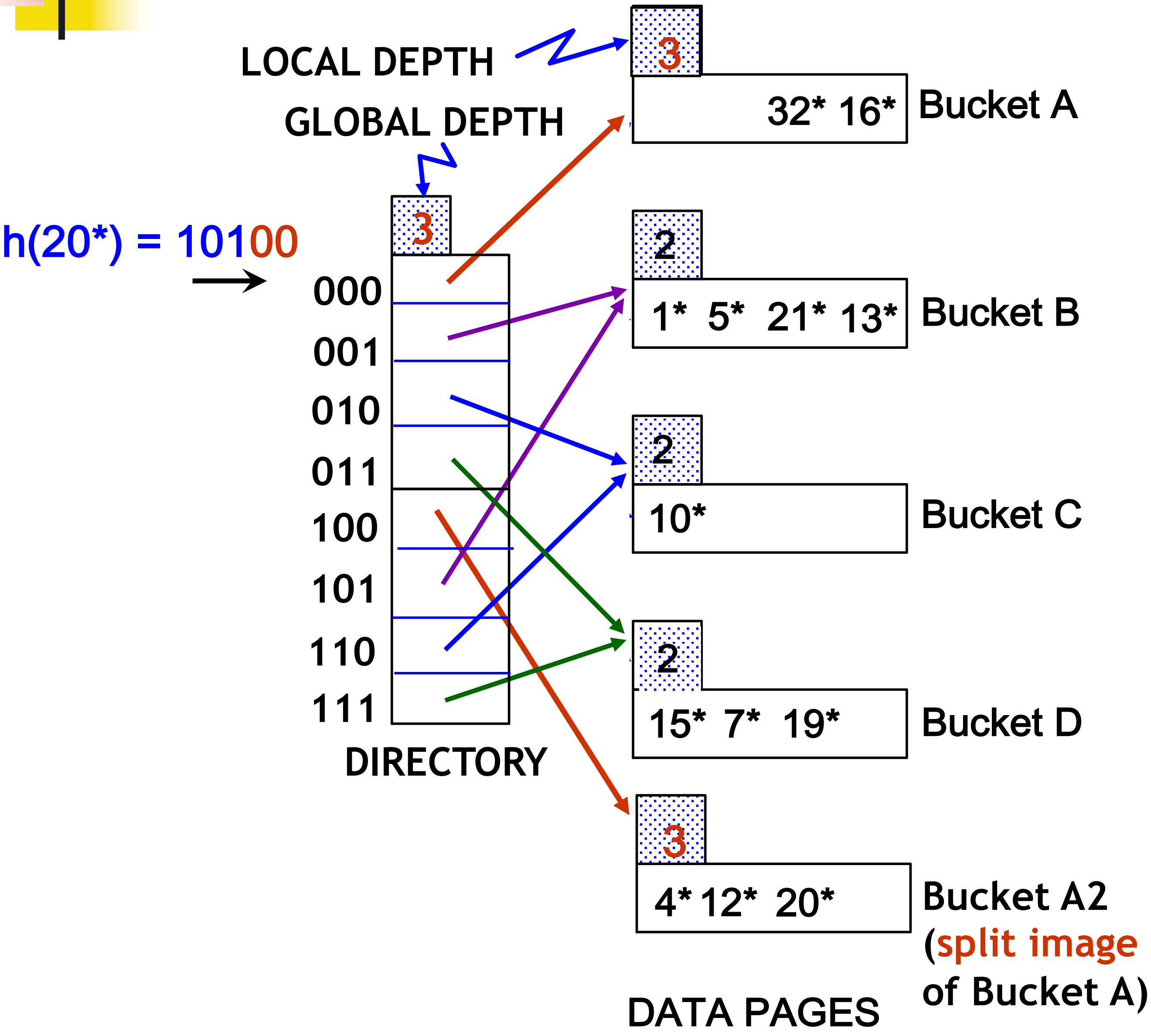
Example

- Insert 13*
- Suppose

  h(13*) = 1101



Insert 20



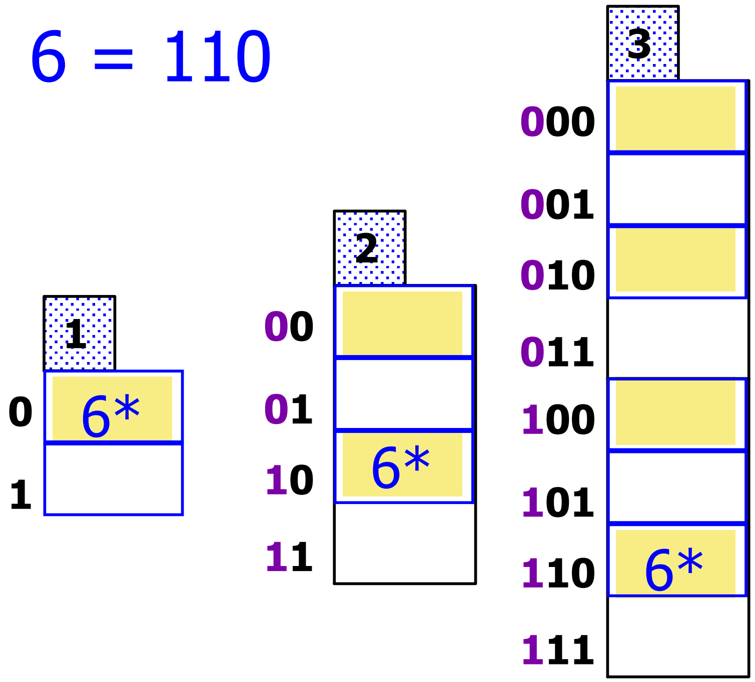
Notice that splitting a bucket only requires doubling the directory when LD = GD

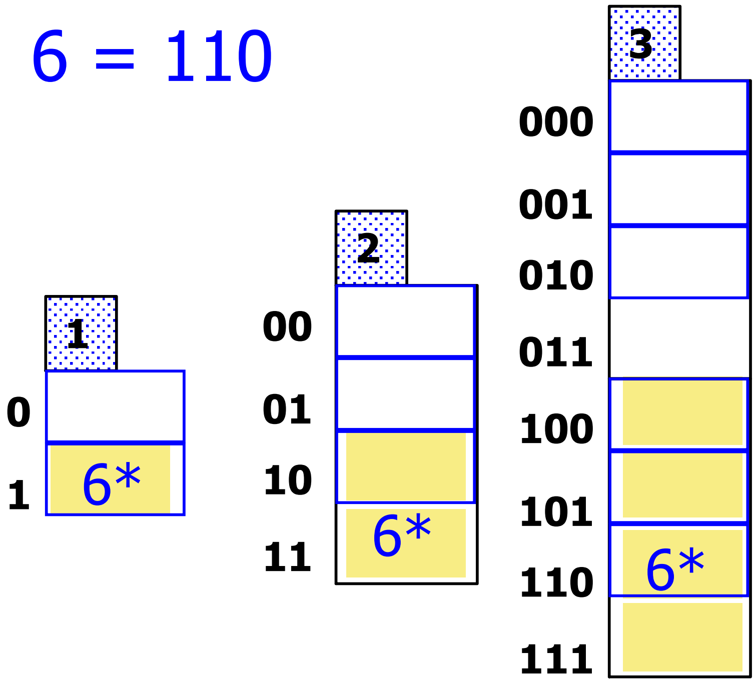
Directory doubled by *copying it over* and fixing pointer to split image page

Directory Doubling

Why use least significant bits in directory?

Allows for doubling via copying!



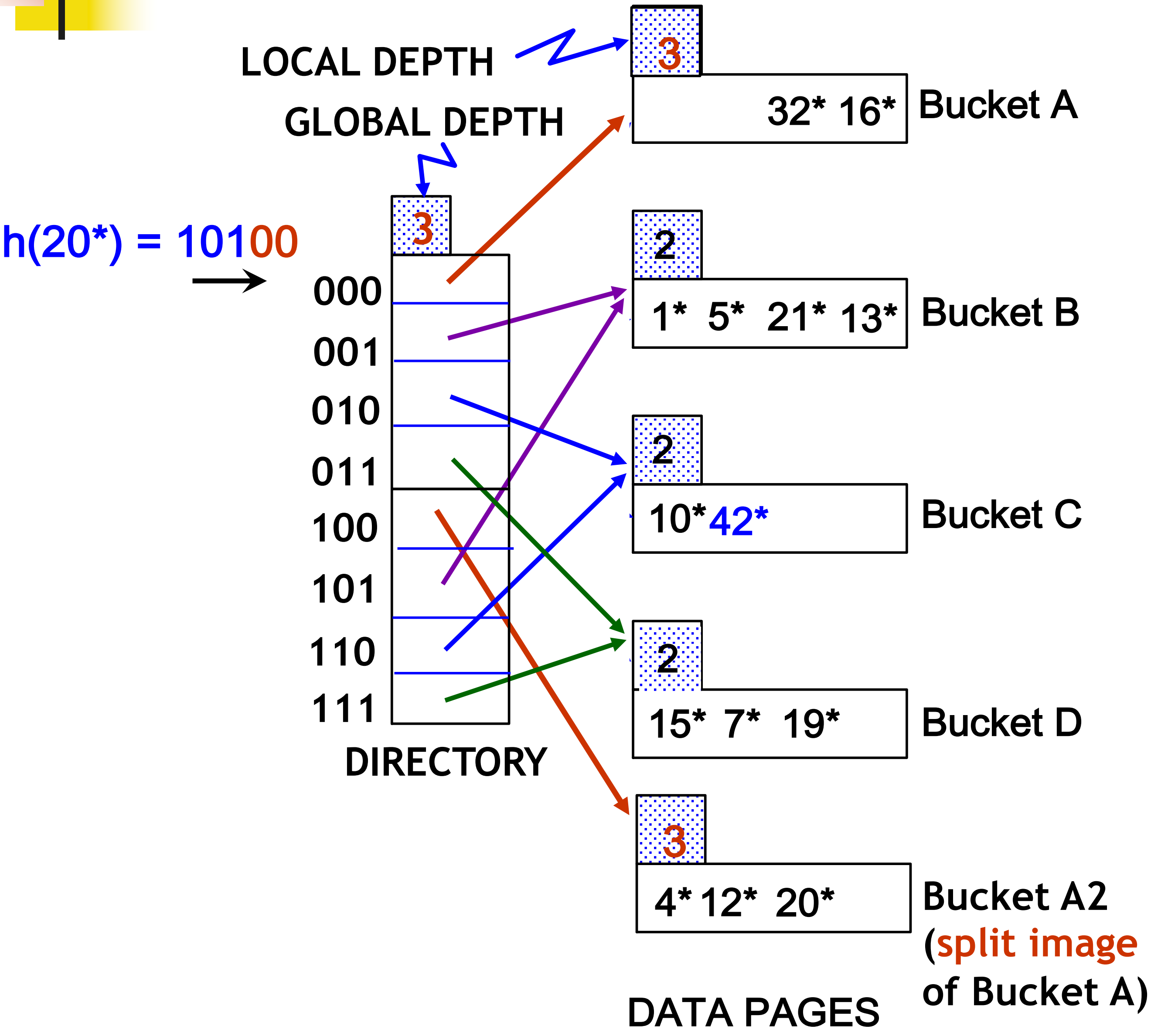


Least Significant vs. Most Significant

Comments on Extensible Hashing

- How many disk accesses for equality search?
  - One if directory fits in memory, else two.
- Directory grows in spurts, and, if the distribution *of hash values* is skewed, directory can grow large.
- Do we ever need overflow pages?
  - Multiple entries with same hash value cause problems!
- **Delete:** Reverse of inserts – see textbook.

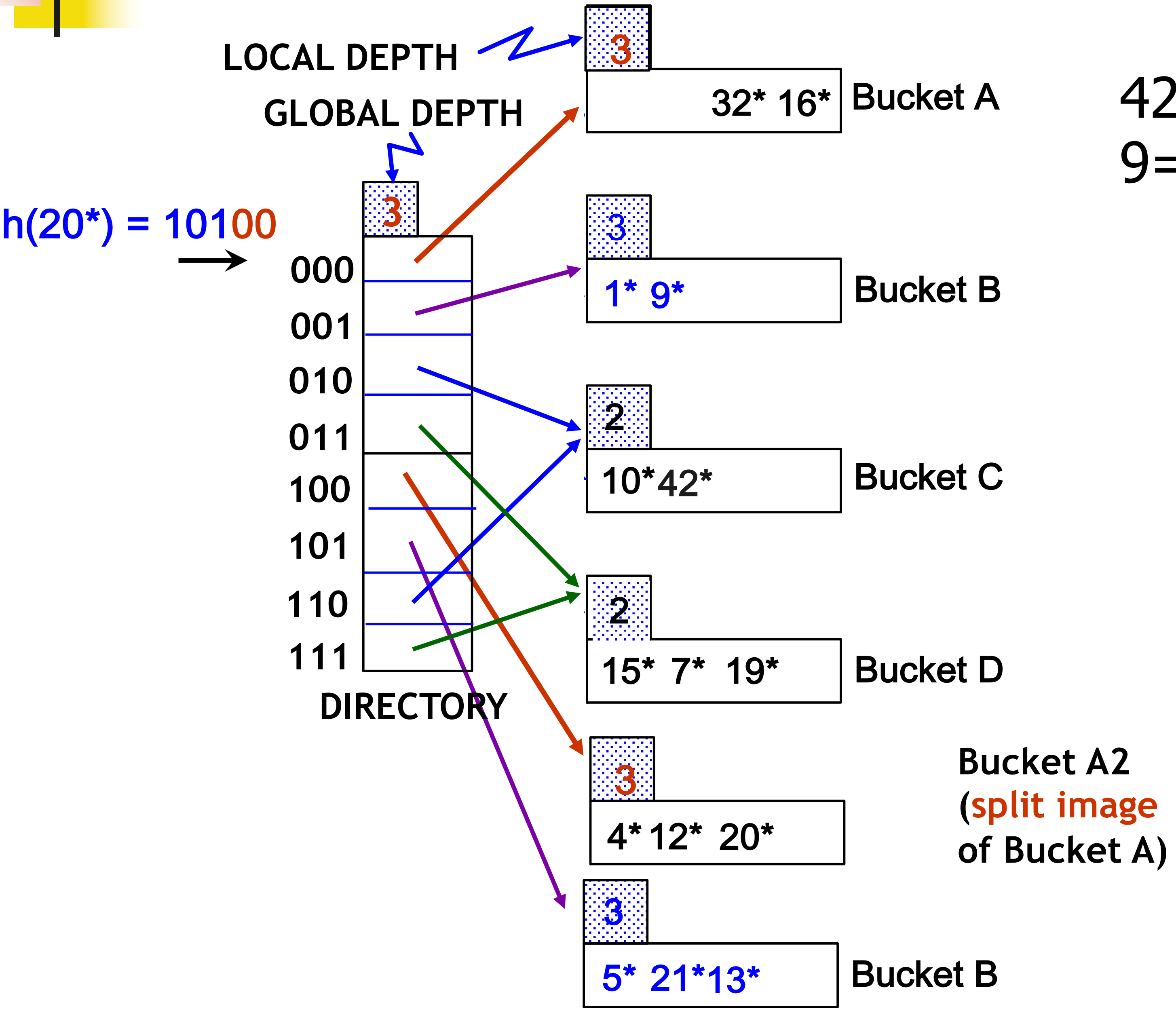
Quiz: Insert 42, 9



$42=(101010)_2$

$9=(1001)_2$

Answer: Insert 42, 9



$42=(101010)_2$

$9=(1001)_2$

Linear Hashing

- Another dynamic hashing scheme
- Eliminates long overflow chains without using a directory.
- **Main idea:** Use a family of hash functions $\mathbf{h}_0$, $\mathbf{h}_1$, $\mathbf{h}_2$, …
  - $\mathbf{h}_{i+1}$ doubles the range of $\mathbf{h}_i$ (similar to directory doubling)
  - Hash family typically obtained by choosing hash function *h()* and initial number of buckets *N*
  - Define $h_i(value) = h(value)mod(2^iN)$
  - If N is a power of 2, apply hash function *h(),* and look at last d_i bits
    - d_0 number of bits needed to represent *N*
    - $d_i = d_0 + i$

Linear Hashing

- Splitting proceeds in rounds
  - During round *level*, only $\mathbf{h}_{\text{level}}$ and $\mathbf{h}_{\text{level+1}}$ are in use
- Variables
  - Level: Initialized to 0
  - Next: Pointer to the bucket being split
- At the beginning of round # Level, the # buckets in the file = $N * 2^{\text{Level}}$
  - N is initial number of buckets

Linear Hashing Example

Level 0, N = 4

What happens if we add 37*?
37=(100101)$_2$

Next=0

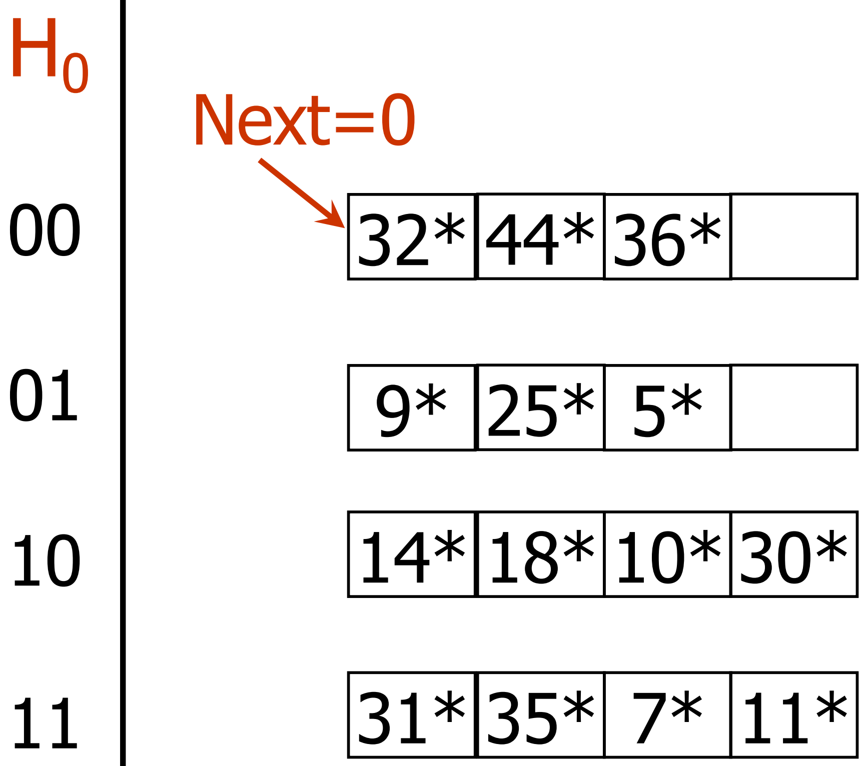
H_0				
00	32*	44*	36*	
01	9*	25*	5*	
10	14*	18*	10*	30*
11	31*	35*	7*	11*

Primary bucket Pages in the Hash File

This is not actually stored

Linear Hashing Example

Level 0, N = 4



Next=0

H_0				
00	32*	44*	36*	
01	9*	25*	5*	
10	14*	18*	10*	30*
11	31*	35*	7*	11*

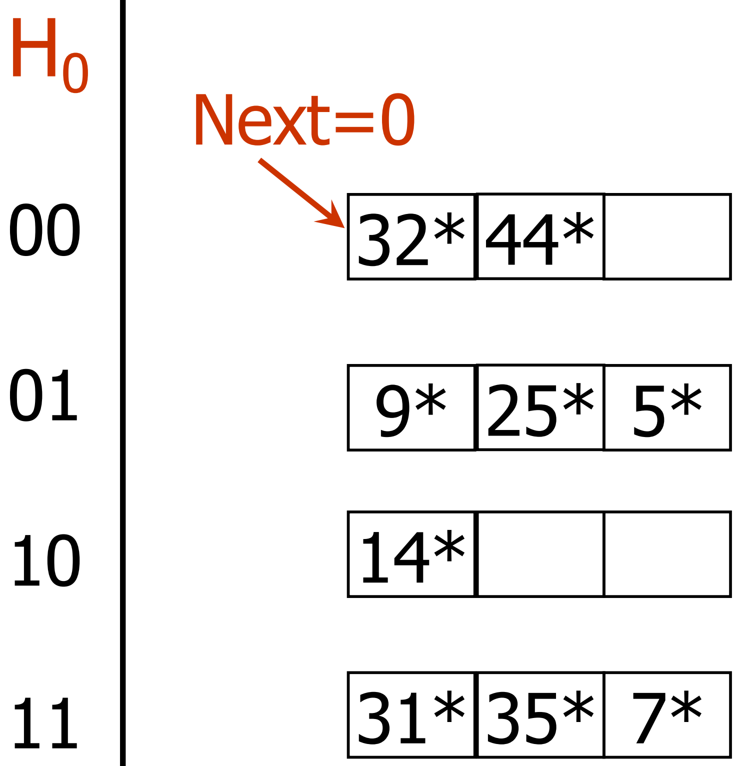
Primary bucket Pages in the Hash File

This is not actually stored

Linear Hashing Example

Level 0, N = 4

What happens if we add 37*?
$37=(100101)_2$



Next=0

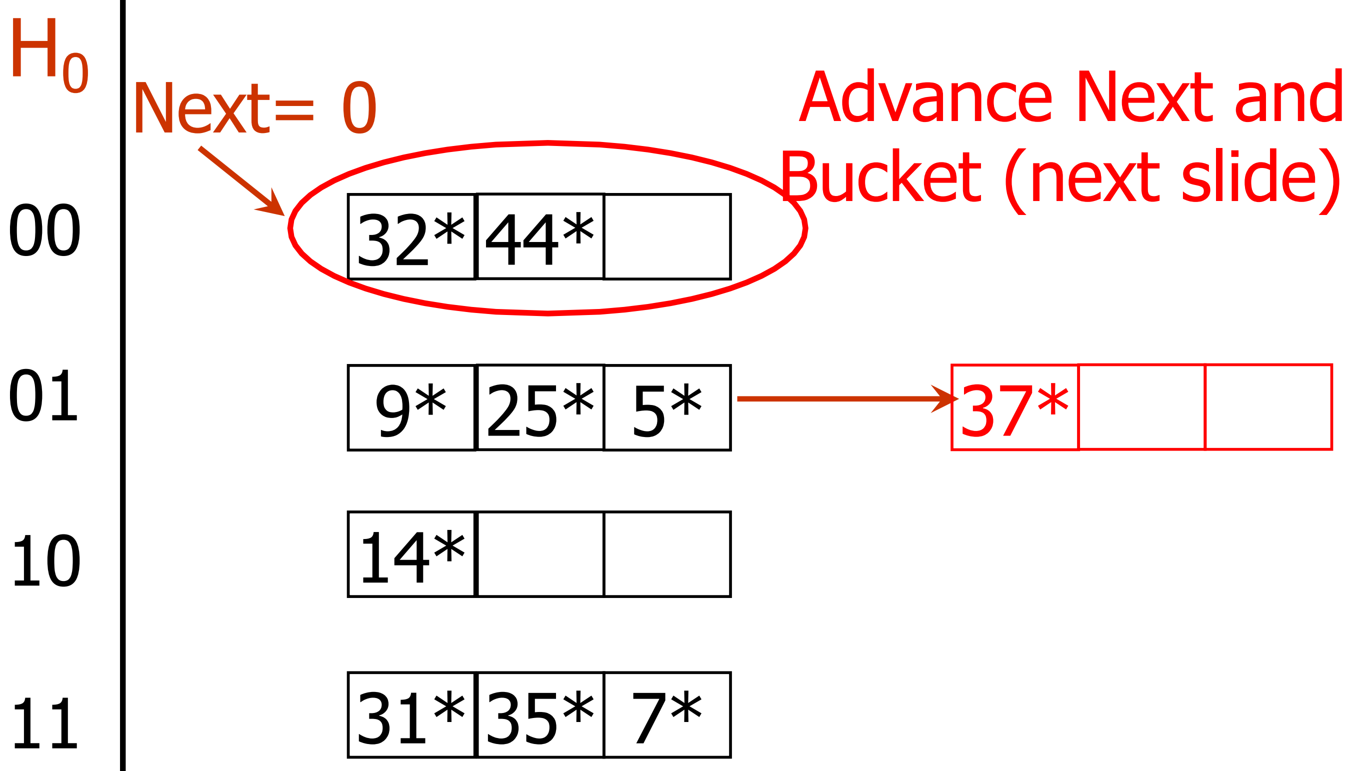
H_0			
00	32*	44*	
01	9*	25*	5*
10	14*		
11	31*	35*	7*

This is not actually stored

Primary bucket Pages in the Hash File.
Next points to the next bucket to be split on an overflow

Linear Hashing Example

Level 0, N = 4



H_0

Next= 0

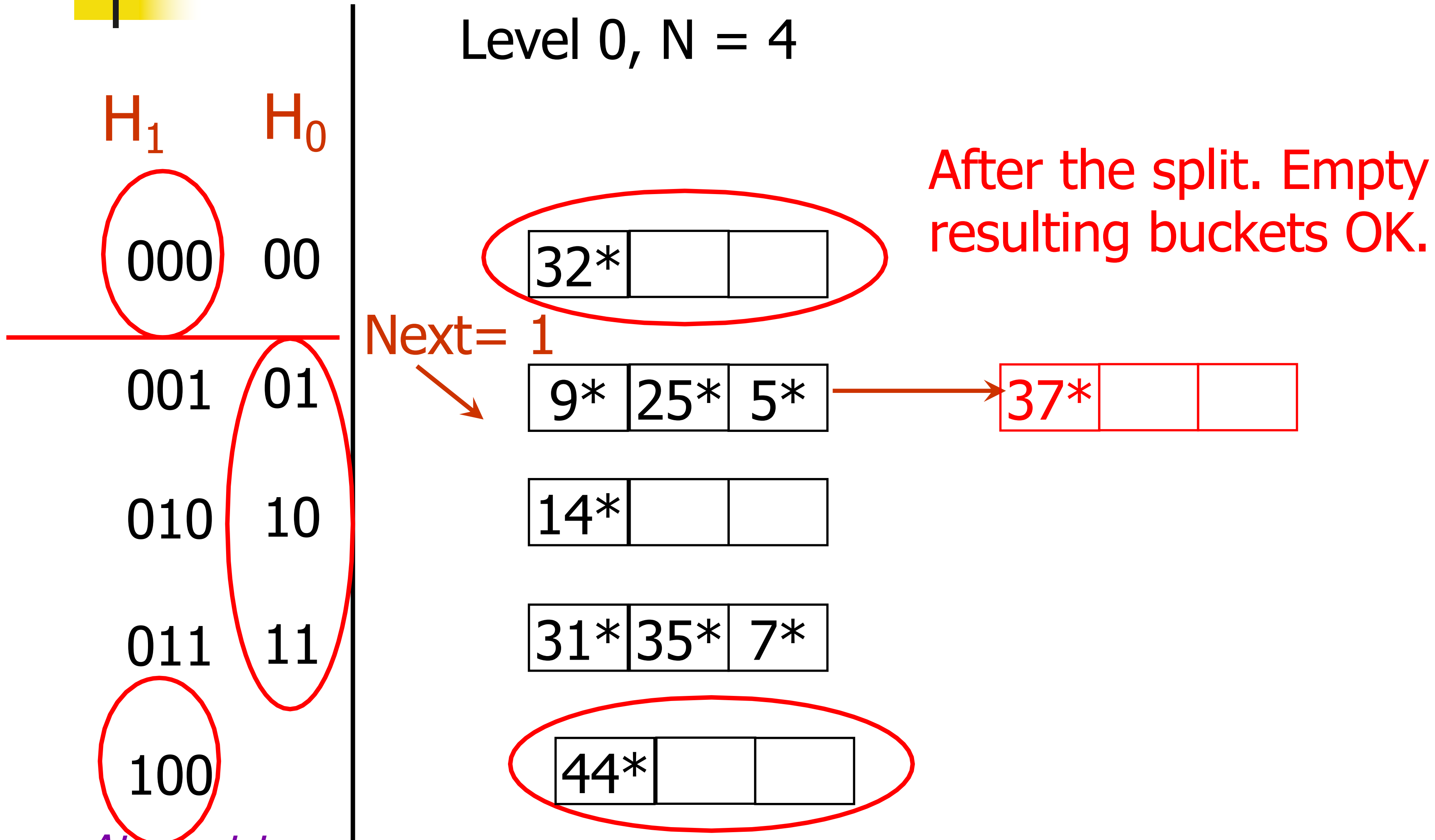
Advance Next and Split this Bucket (next slide)

00	32*	44*				
01	9*	25*	5*	37*		
10	14*					
11	31*	35*	7*			

This is not actually stored

Note: 37 added to bucket 01. But bucket 00 will be split, since Next points to 00.

Linear Hashing Example



Level 0, N = 4

After the split. Empty resulting buckets OK.

Next= 1

H_1	H_0				
000	00	32*			
001	01	9*	25*	5*	37*
010	10	14*			
011	11	31*	35*	7*	
100		44*			

At most two hash functions at a time

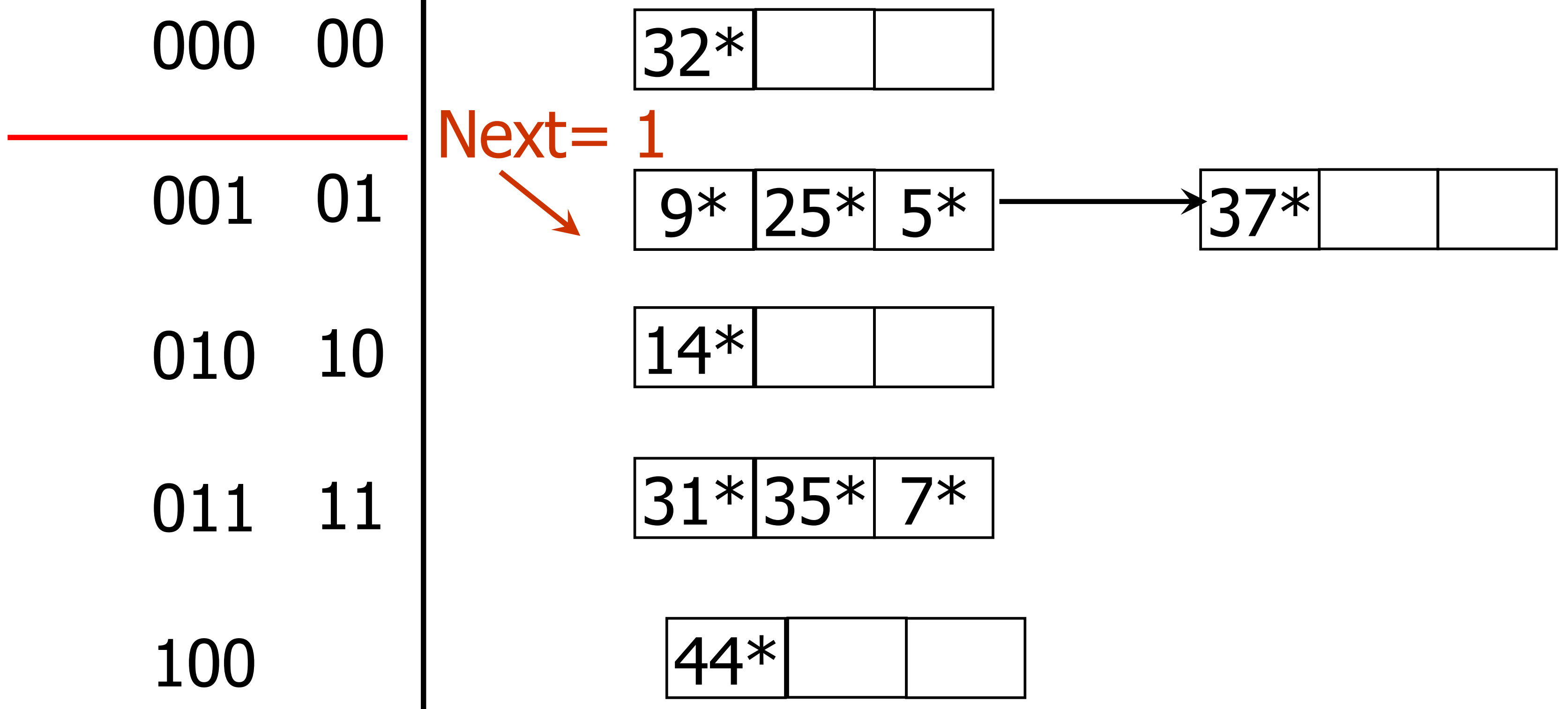
To hash, first use H_0. If H_0(key) < Next, then use H_1(key) instead. Thus, 44 hashed to 100.

Linear Hashing Example

Level 0, N = 4

What happens if we add 13*?
13=$(1101)_2$

H_1 H_0



000 00
32*

Next= 1

001 01
9* 25* 5* → 37*

010 10
14*

011 11
31* 35* 7*

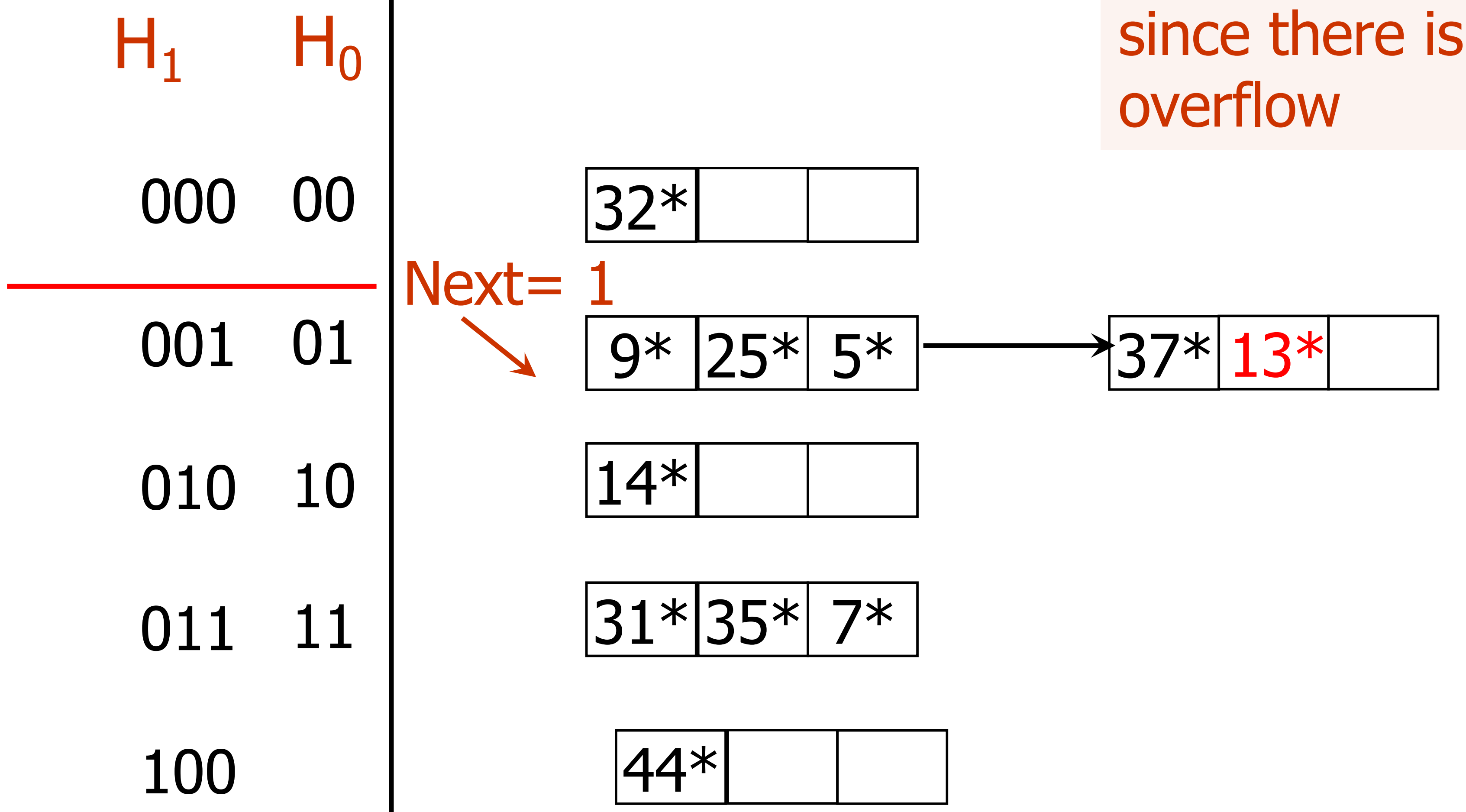
100
44*

At most two hash functions at a time

Linear Hashing Example

Level 0, N = 4

Next not advanced since there is no overflow



H_1	H_0						
000	00	32*					
001	01	9*	25*	5*	37*	13*	
010	10	14*					
011	11	31*	35*	7*			
100		44*					

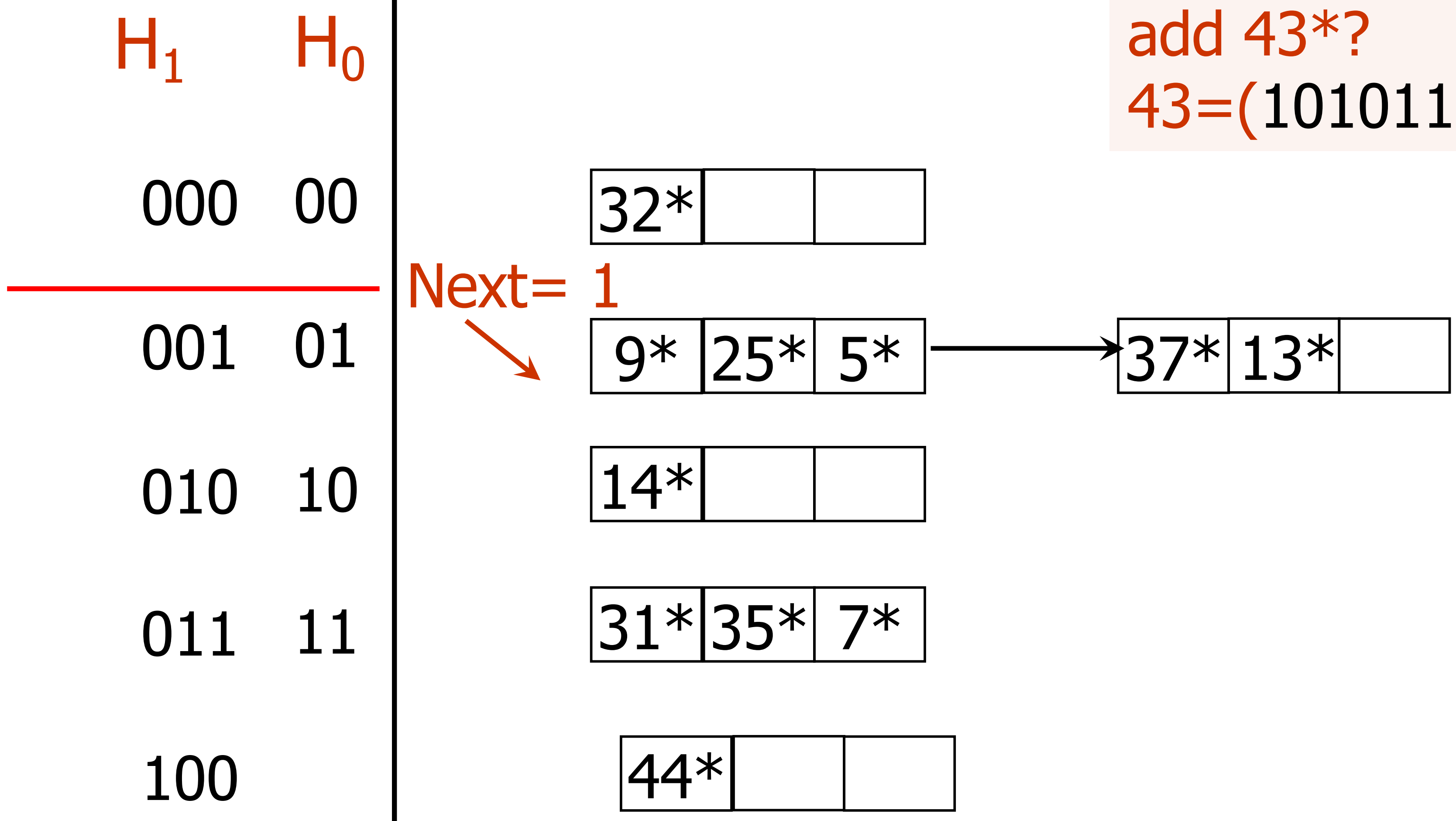
Next= 1

At most two hash functions at a time

Linear Hashing Example

Level 0, N = 4

What happens if we add 43*?
$43=(101011)_2$



Next= 1

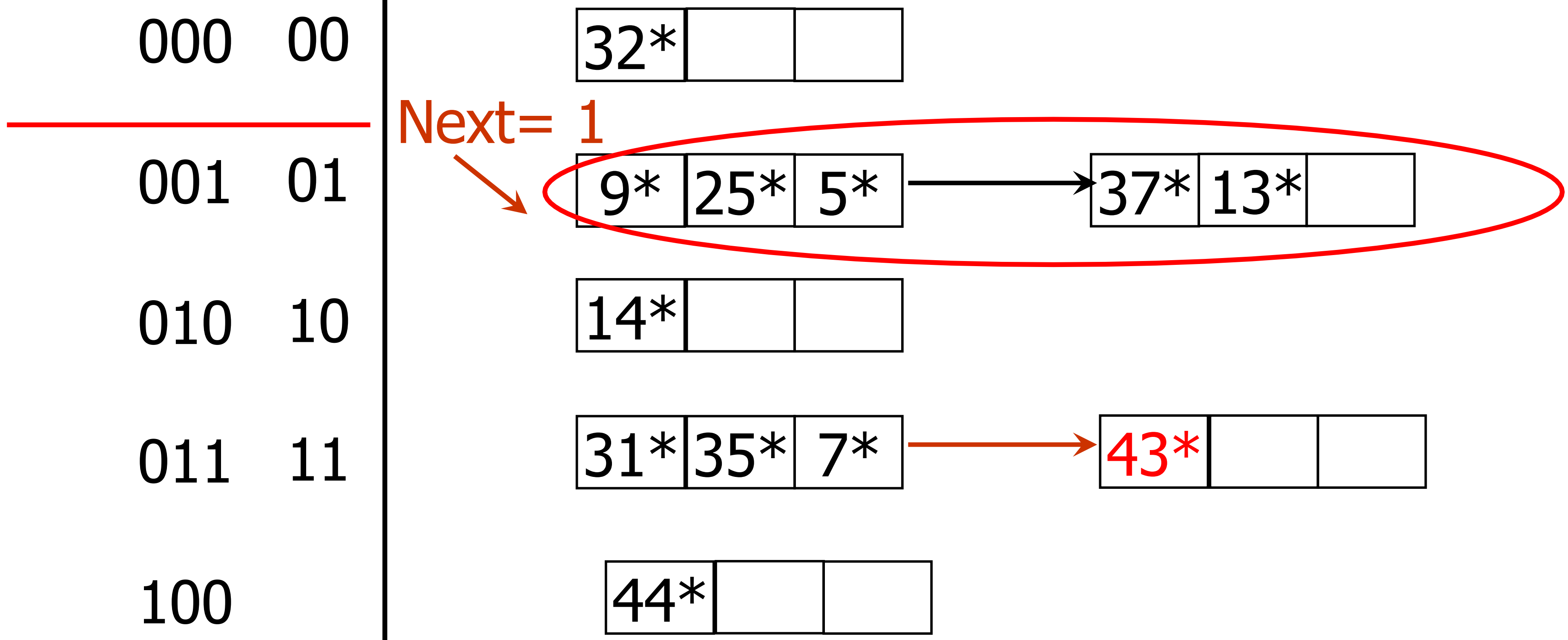
H_1	H_0						
000	00	32*					
001	01	9*	25*	5*	37*	13*	
010	10	14*					
011	11	31*	35*	7*			
100		44*					

At most two hash functions at a time

Linear Hashing Example

Level 0, N = 4

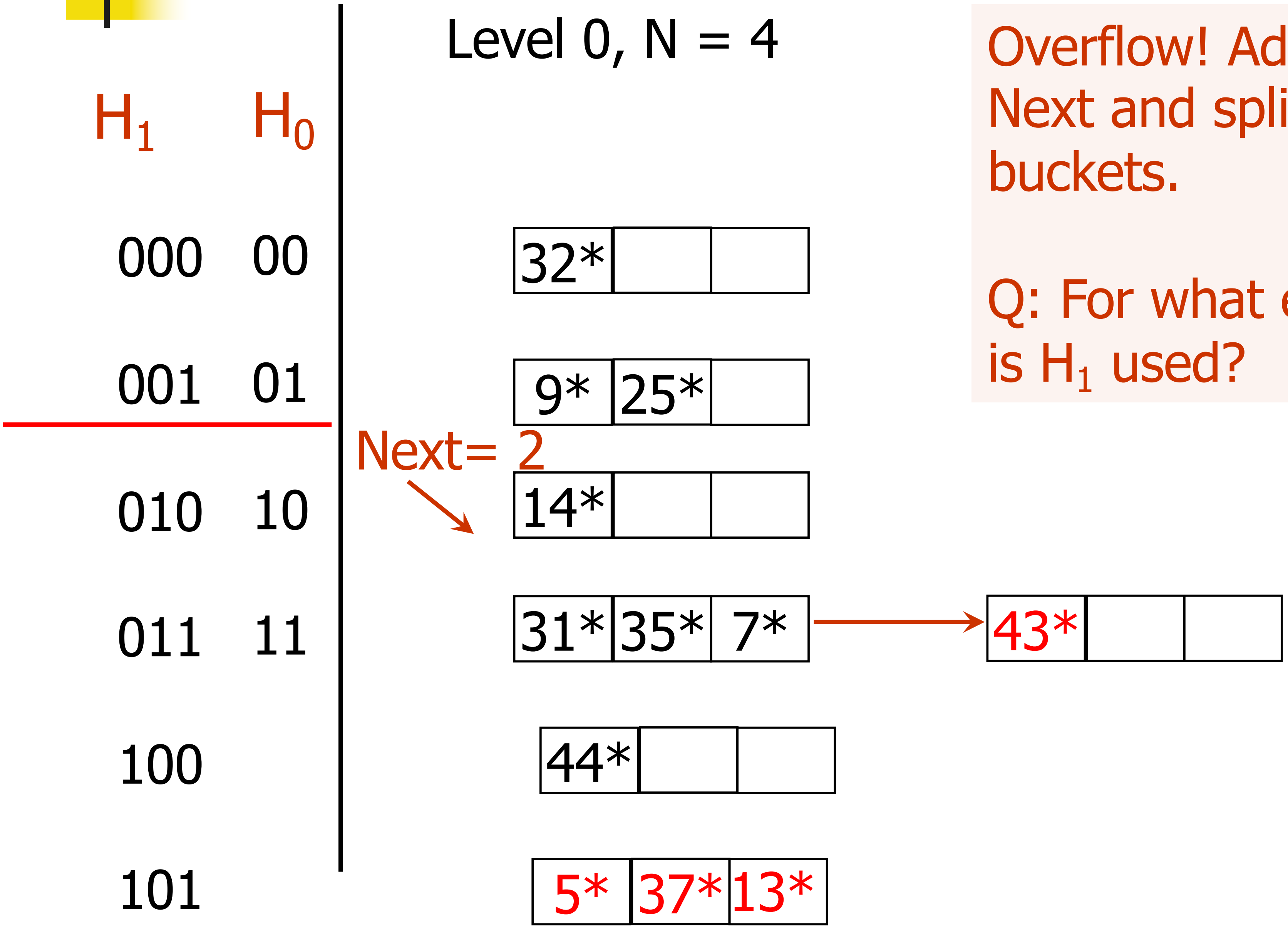
Overflow! Advance Next and split the buckets.



H_1	H_0		
000	00	32*	
001	01	9* 25* 5*	37* 13*
010	10	14*	
011	11	31* 35* 7*	43*
100		44*	

At most two hash functions at a time

Linear Hashing Example



Level 0, N = 4

H_1	H_0						
000	00	32*					
001	01	9*	25*				
010	10	14*					
011	11	31*	35*	7*	43*		
100		44*					
101		5*	37*	13*			

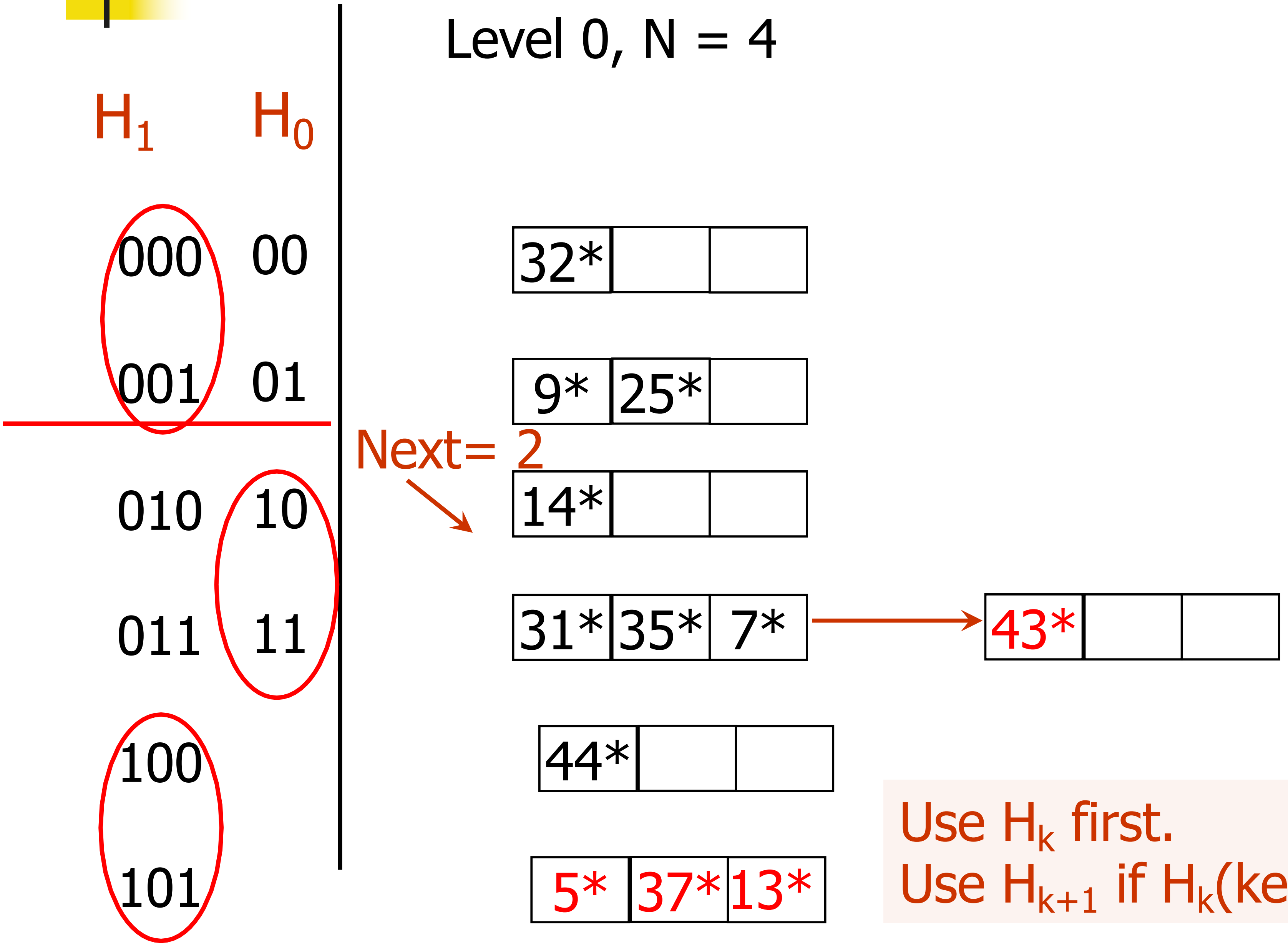
Next= 2

Overflow! Advance Next and split the buckets.

Q: For what entries is H_1 used?

Linear Hashing Example

Level 0, N = 4



H_1	H_0				
000	00	32*			
001	01	9*	25*		
010	10	14*			
011	11	31*	35*	7*	43*
100		44*			
101		5*	37*	13*	

Next= 2

Use H_k first.
Use H_{k+1} if $H_k(key)$ < Next.

Linear Hashing Example

Level 0, N = 4

H_1	H_0
000	00
001	01
010	10
011	11
100	
101	
110	

Next= 3

What happens on the next split?
(Imagine the buckets)

Linear Hashing Example

Level 0, N = 4

H_1	H_0
000	00
001	01
010	10
011	11
100	
101	
110	
111	

Next= 4?

No. At this point, H0 is useless – will never be used. Reset Next to and advance Level.

Linear Hashing Example

Level 1, N = 4

H_1

000 Next= 0

001

010

011

100

101

110

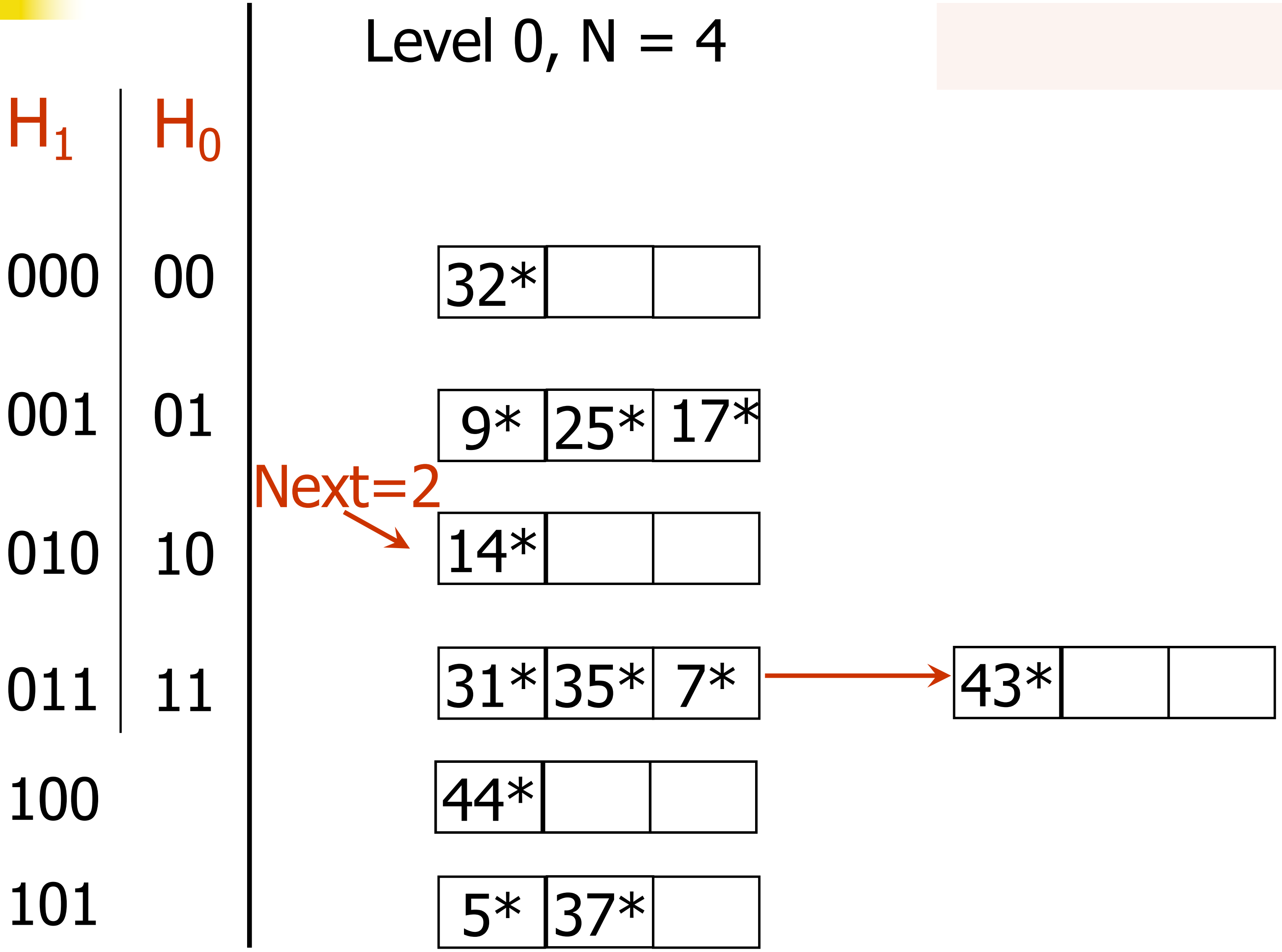
111

Level advanced. H_0 no longer needed.
H_2 will be used when Next advances further.

Linear Hashing (Generalization)

- *Can choose any criterion to trigger split.*
  - Split on a overflow (as we assumed)
  - Space utilization on the page > 90%
- Since buckets are split round-robin, long overflow chains don’t develop!
- **Deletes**: see textbook

Quiz: Add 13* and 29*



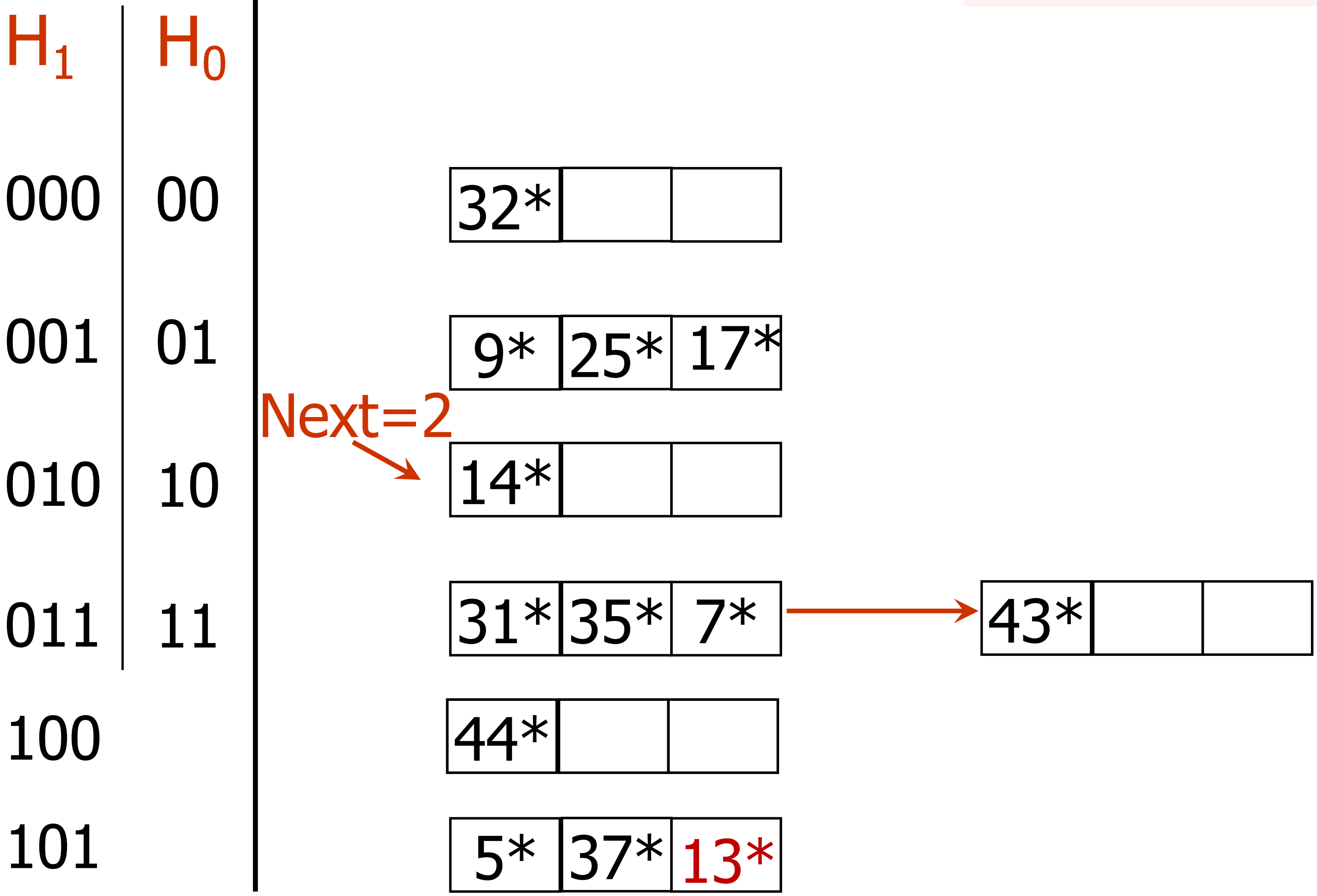
Level 0, N = 4

Next=2

H_1	H_0						
000	00	32*					
001	01	9*	25*	17*			
010	10	14*					
011	11	31*	35*	7*	43*		
100		44*					
101		5*	37*				

Quiz: After adding 13

Level 0, N = 4

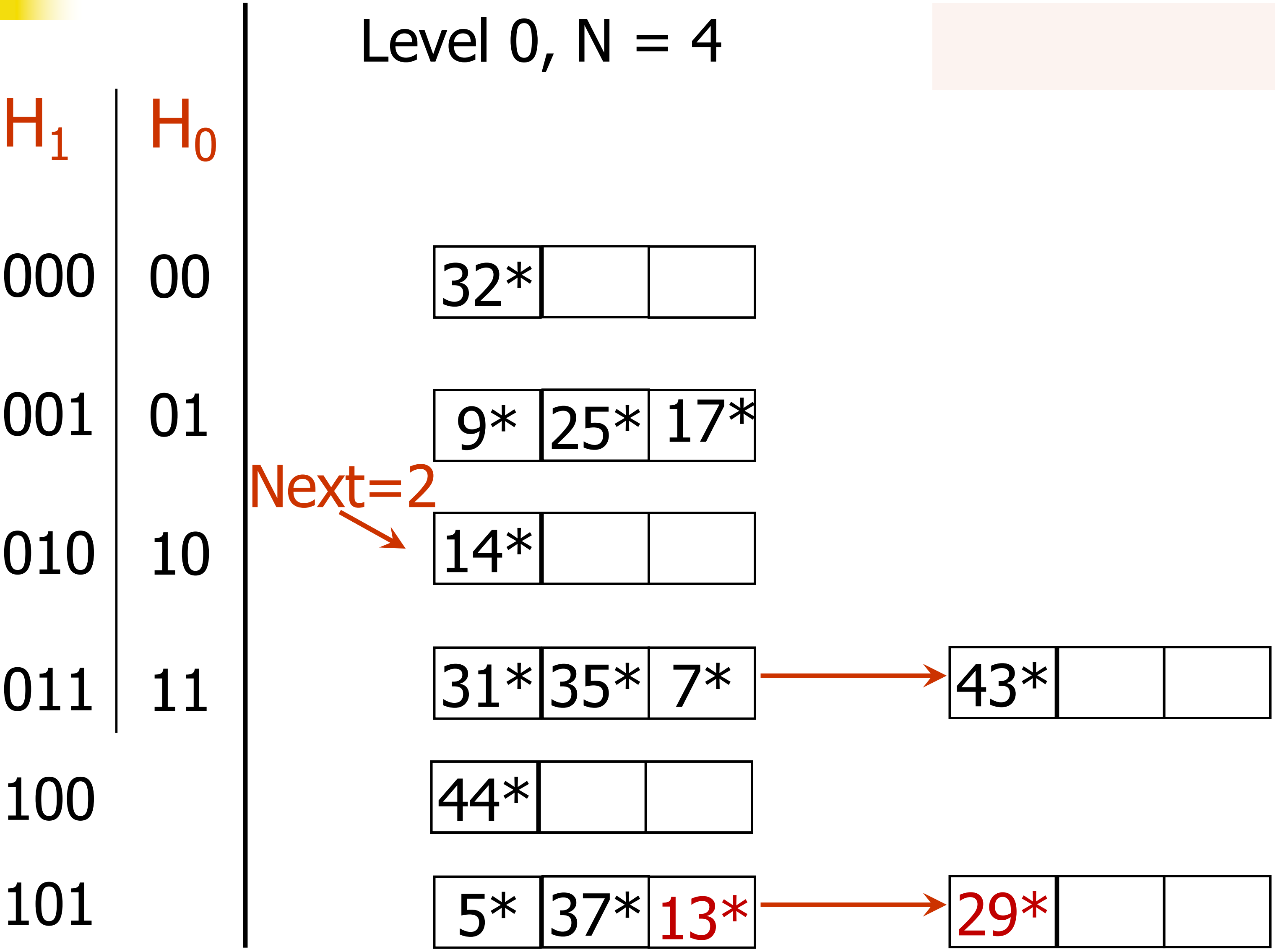


H_1	H_0						
000	00	32*					
001	01	9*	25*	17*			
010	10	14*					
011	11	31*	35*	7*	43*		
100		44*					
101		5*	37*	13*			

Next=2

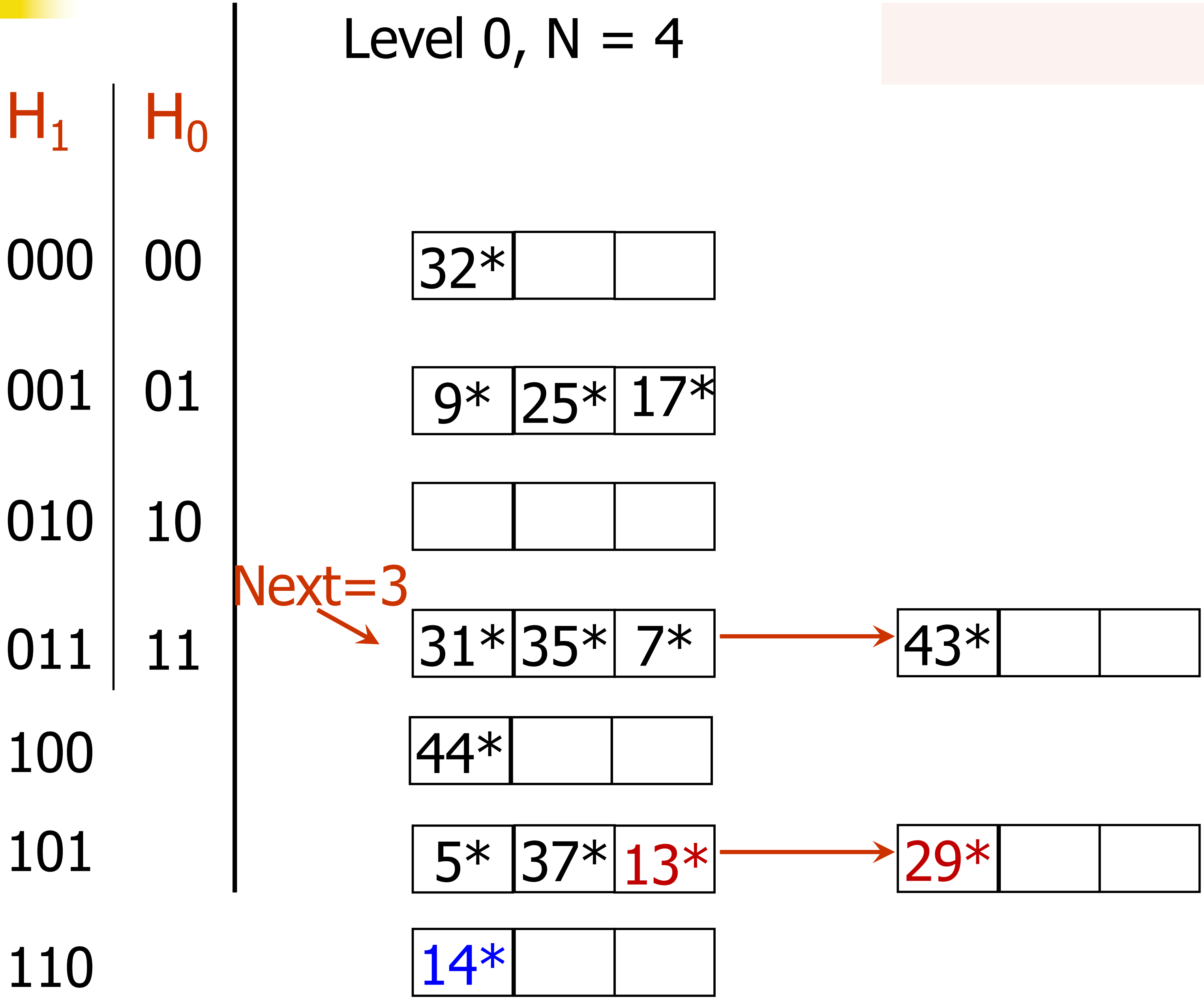
Quiz: After adding 29* (Step 1)

Level 0, N = 4



H_1	H_0							
000	00	32*						
001	01	9*	25*	17*				
010	10	14*						
011	11	31*	35*	7*	→	43*		
100		44*						
101		5*	37*	13*	→	29*		

Quiz: After adding 29* (done)



Level 0, N = 4

H_1	H_0						
000	00	32*					
001	01	9*	25*	17*			
010	10						
011	11	31*	35*	7*	43*		
100		44*					
101		5*	37*	13*	29*		
110		14*					

Next=3

Summary

- Discussed 3 kinds of hash-based indexes
- Static Hashing can lead to long overflow chains.
- Extensible Hashing
  - Directory to keep track of buckets, doubles periodically.
  - Always splits the “right” bucket.
- Linear Hashing
  - Split buckets round-robin, and use overflow pages.
  - Space utilization could be lower than EH.

Announcements

- Review problems for hash indexes
    - 11.1, 11.3, 11.7, 11.9