

CS460: Intro to Database Systems

Class 11: Hash Indexing

Instructor: Manos Athanassoulis

<https://bu-disc.github.io/CS460/>

Hash Indexing

Static Hashing

Extendible Hashing

Linear Hashing

Introduction

1. Actual data record (with key value **k**)
2. $\langle \mathbf{k}, \text{rid of matching data record} \rangle$
3. $\langle \mathbf{k}, \text{list of rids of matching data records} \rangle$

Choice is orthogonal to the indexing technique

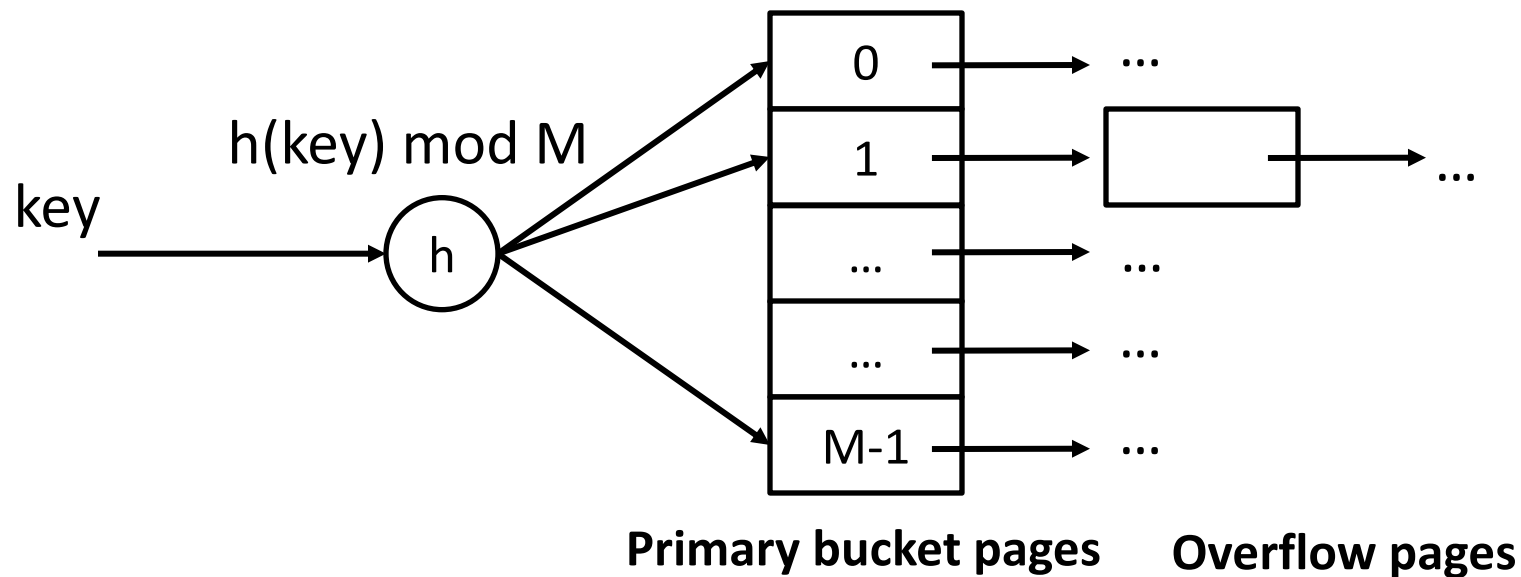
Hash-based indexes \rightarrow *equality selections*
Cannot support range searches

Static and dynamic hashing techniques exist

Static Hashing

#primary bucket pages fixed, allocated sequentially, never de-allocated; overflow pages if needed

$h(k) \bmod M$ = bucket to insert data entry with key k (M : #buckets)



Static Hashing (Contd.)

Buckets contain **data entries**

Hash function on *search key* field of record r

Must distribute values over range $0 \dots M-1$

What is a good hash function?

$h(key) = (a * key + b)$ usually works well

a and b are constants; lots known about how to tune h

Static Hashing (Problems!)



Long overflow chains can develop and degrade performance



Ways to solve?

- Reorganization is expensive and may block queries
- *Extendible* and *Linear Hashing*: Dynamic techniques to fix this problem

Hash Indexing

Static Hashing

Extendible Hashing

Linear Hashing

Extendible Hashing



Why not double the number of buckets?

Note that reading and writing all pages is expensive!

Idea:

Use directory of pointers to buckets

On overflow, double the directory (not the # of buckets)

Why does this help?

- Directory is much smaller than the entire index file

- Only one page of data entries is split

- No overflow page! (caveat: duplicates w.r.t. the hash function)*

Trick lies in how the hash function is adjusted!

Extendible Hashing

Directory: an array

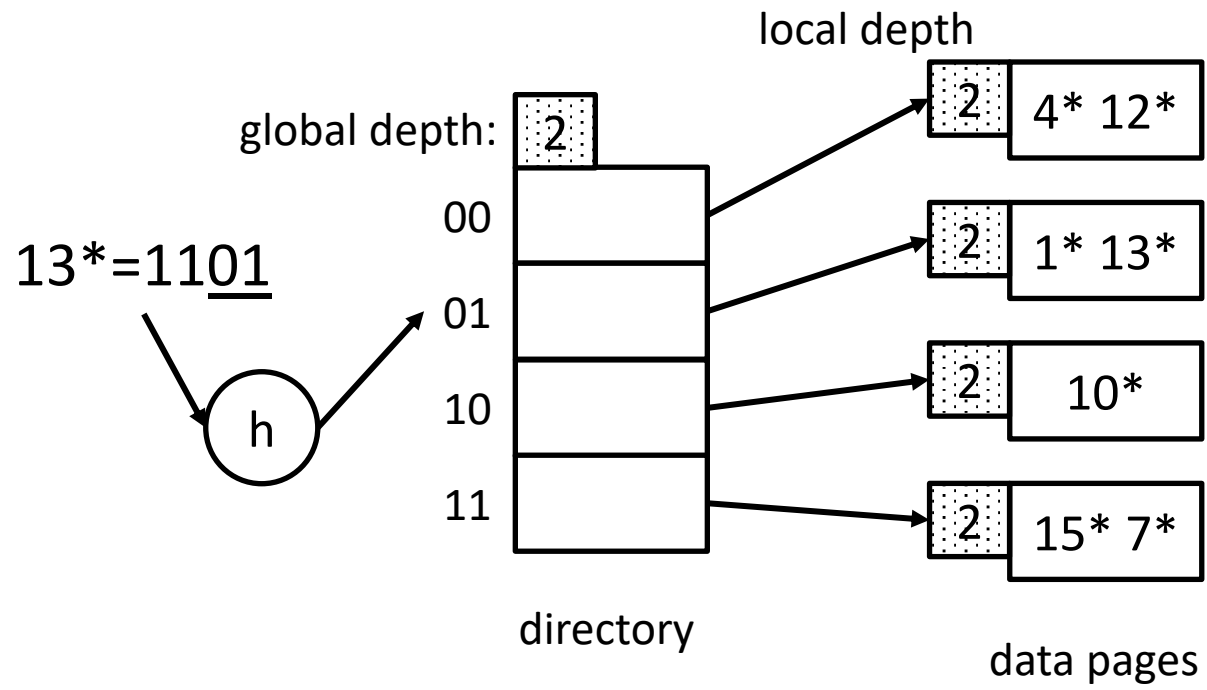
Search for k :

- Apply hash function $h(k)$
- Take last **global depth** # bits of $h(k)$

Insert:

- If the bucket has space, insert, done
- If the bucket is full, **split** it, re-distribute – If necessary, double the directory

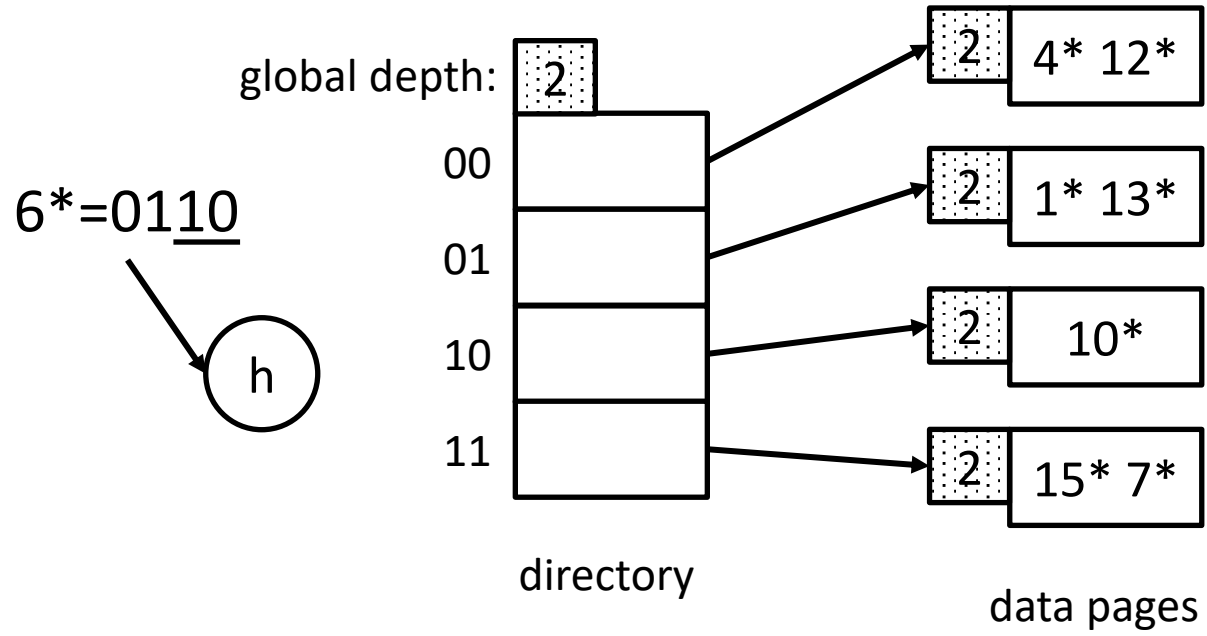
Example



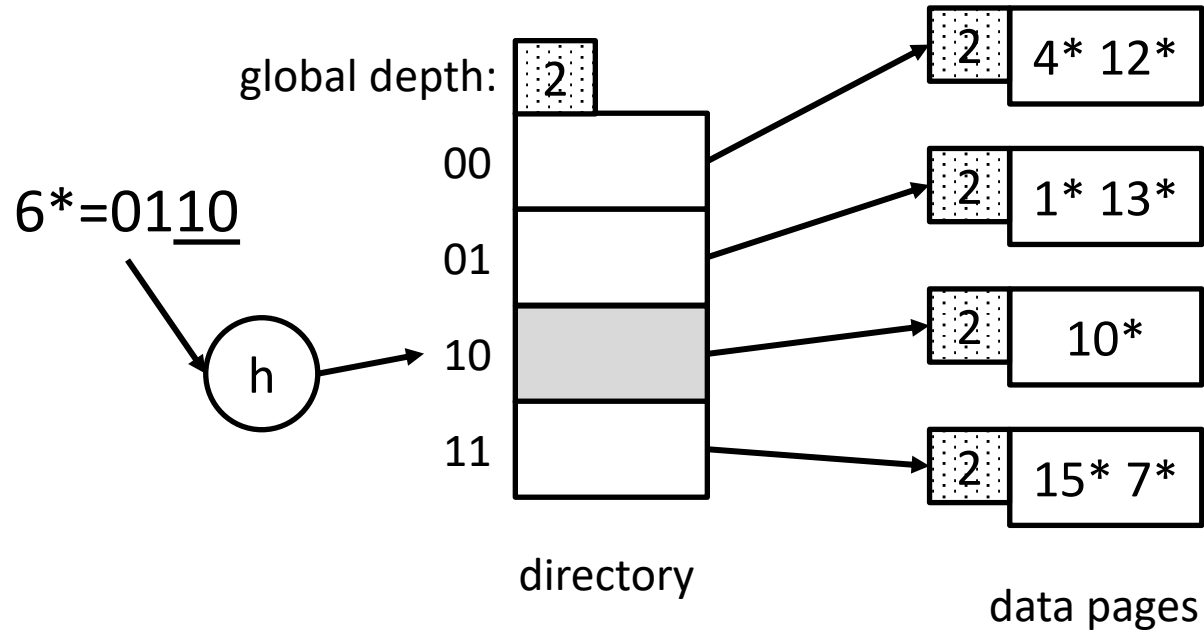
what is the hash function?



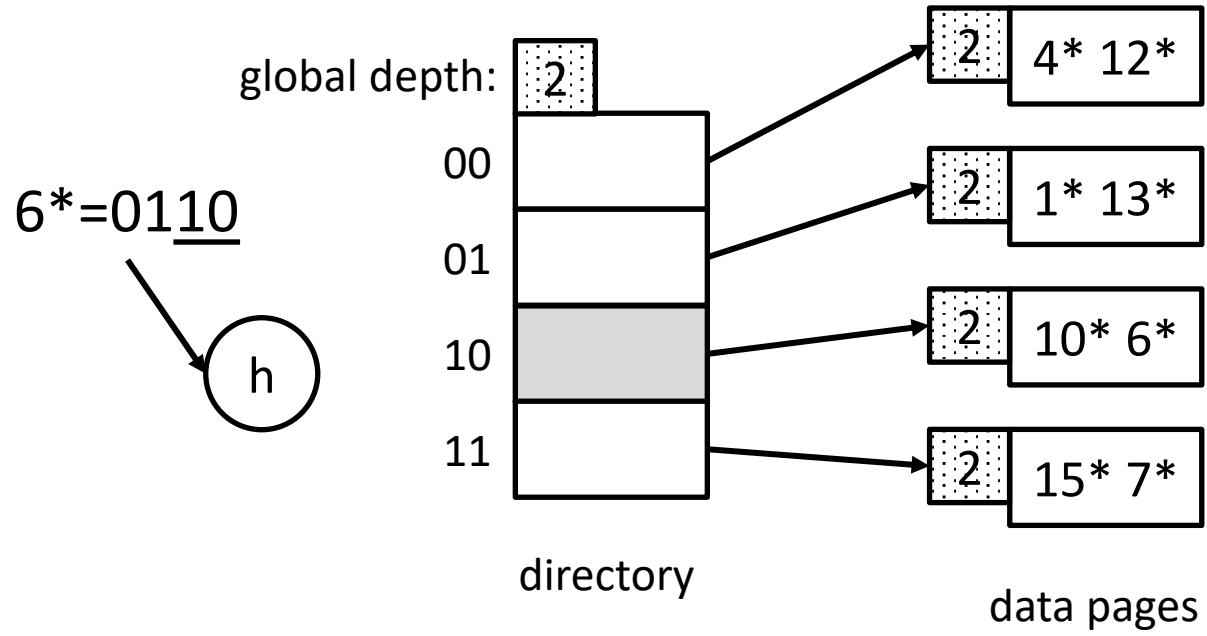
Example: Insert 6



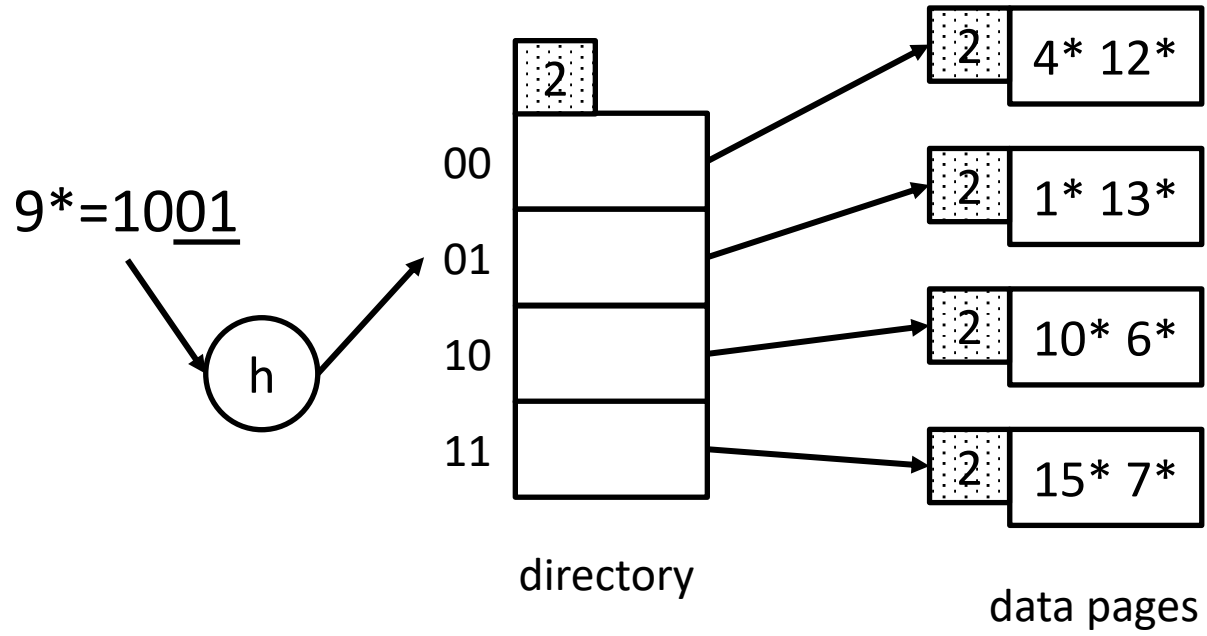
Example: Insert 6



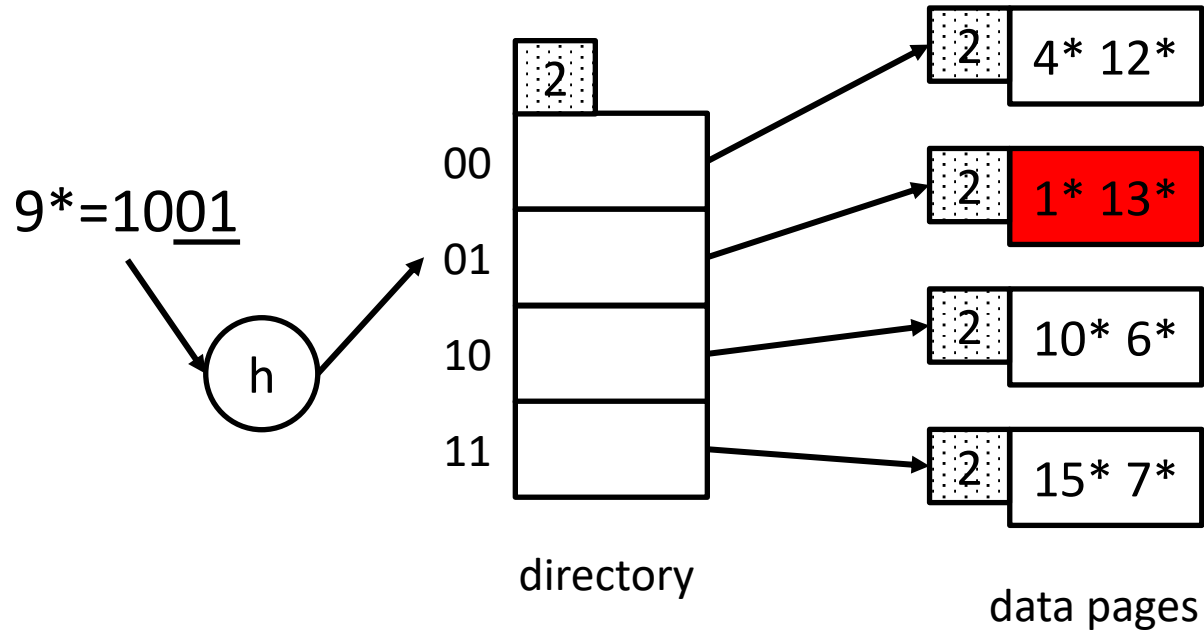
Example: Insert 6



Example 2: Insert 9



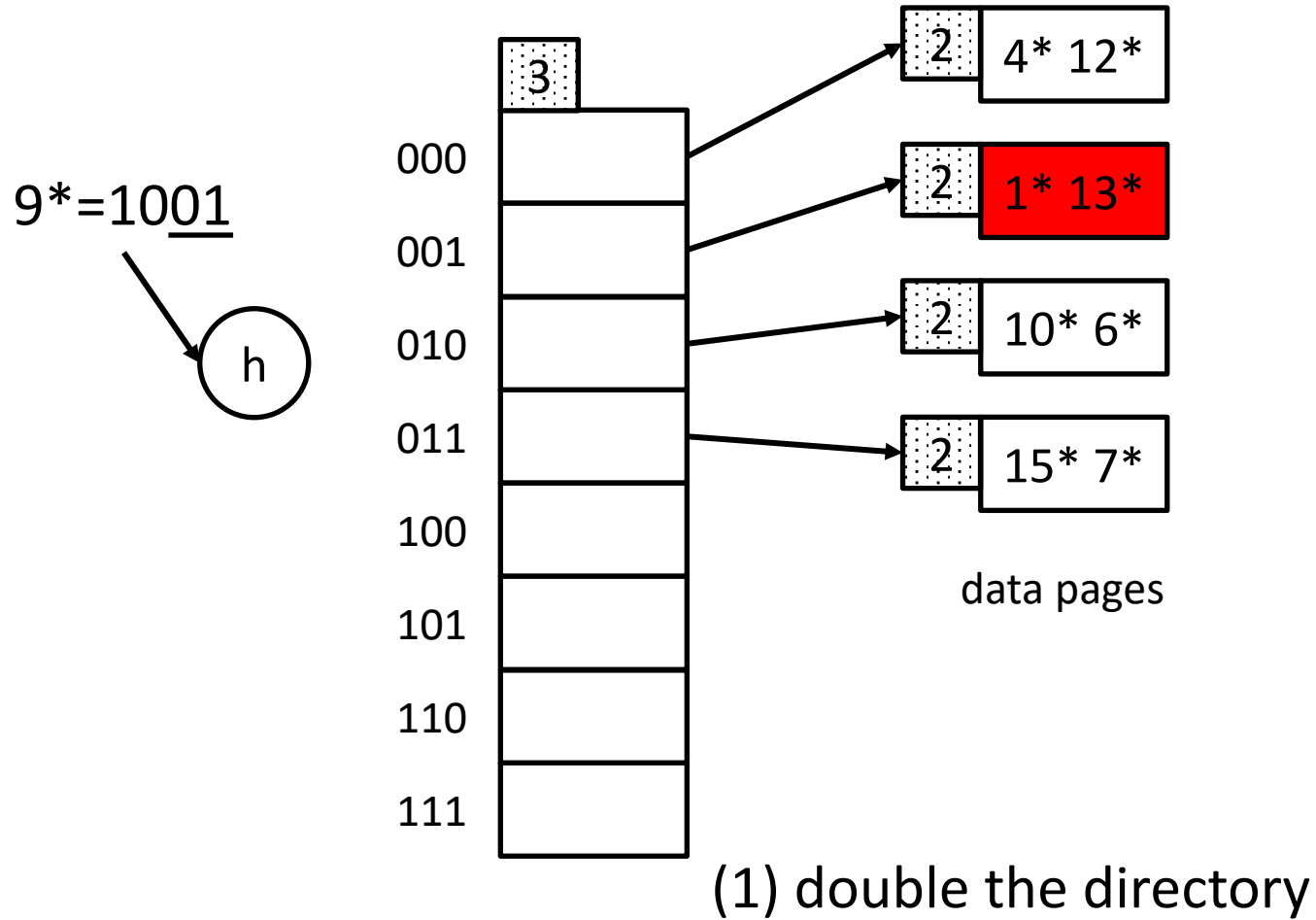
Example 2: Insert 9



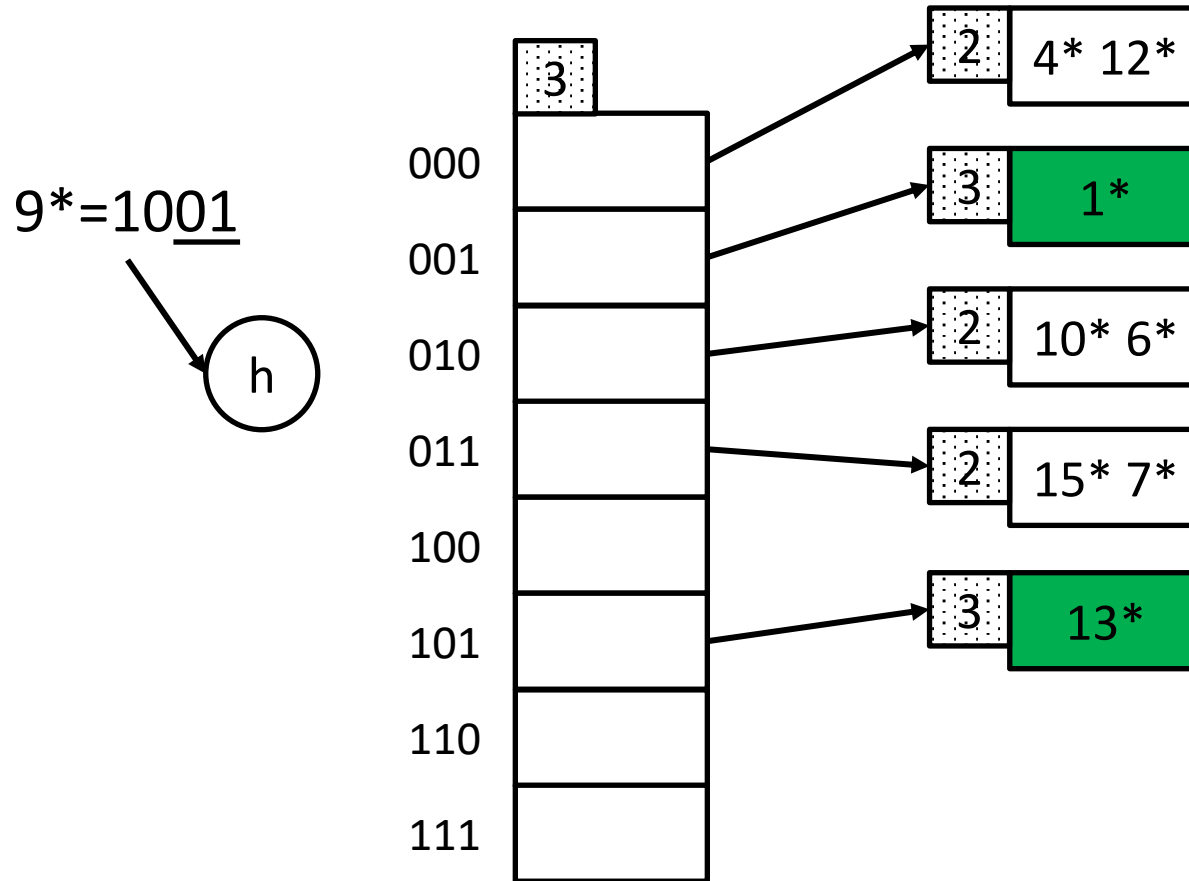
now what??



Example 2: Insert 9

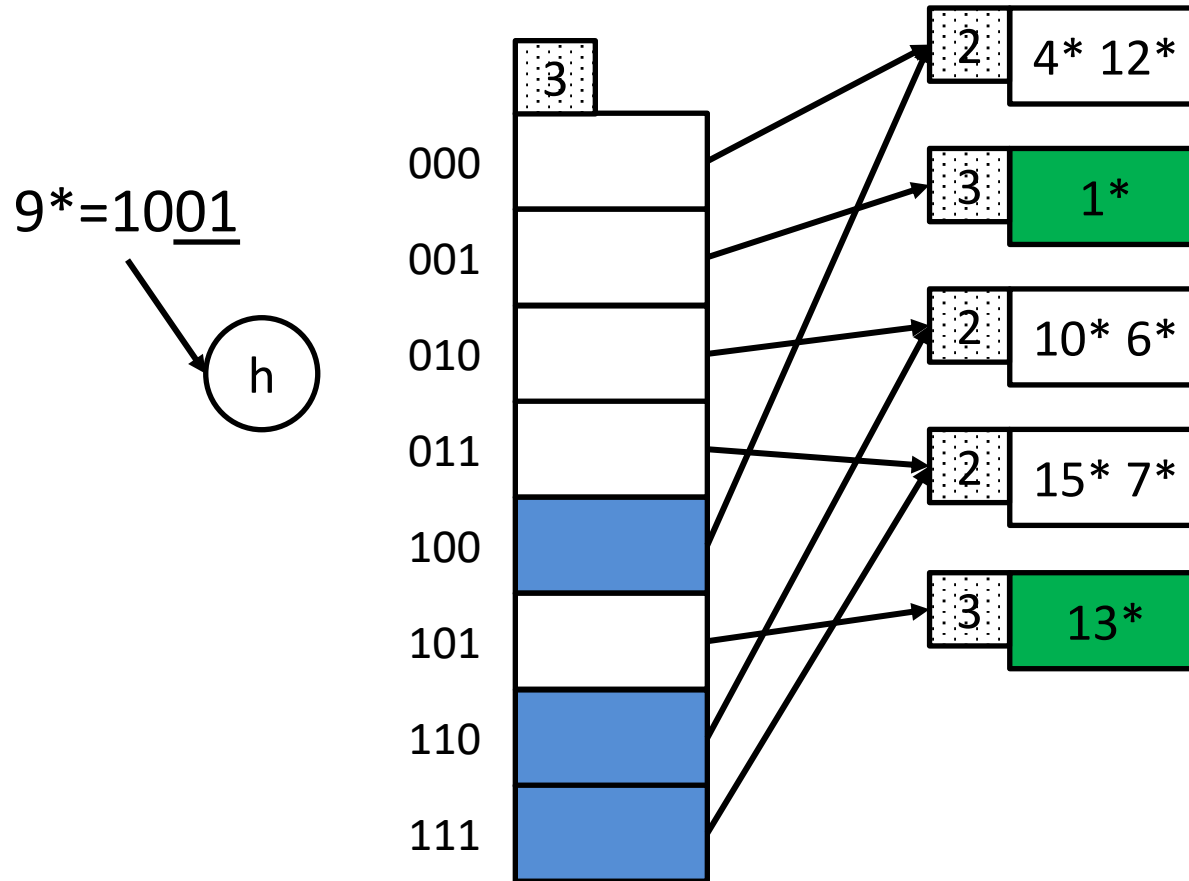


Example 2: Insert 9



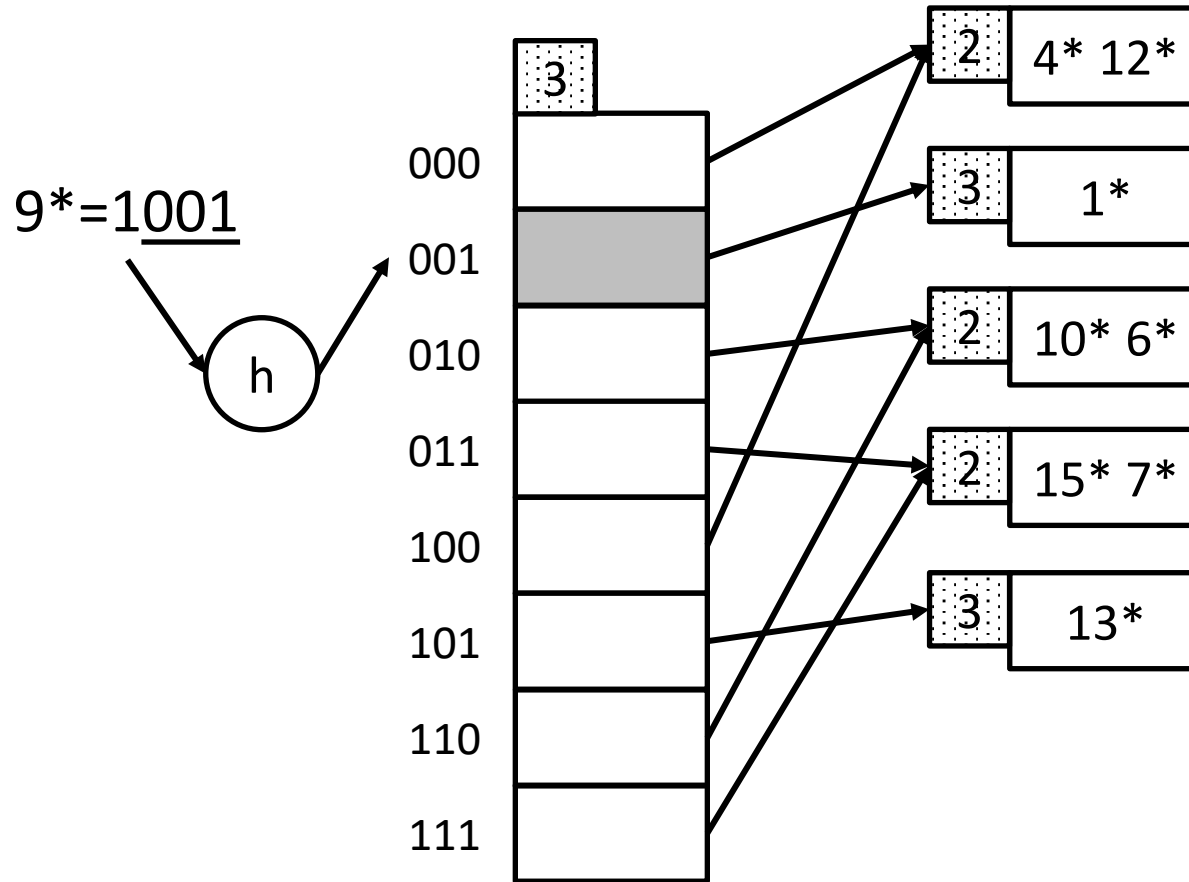
- (1) double the directory
- (2) re-distribute the split bucket

Example 2: Insert 9

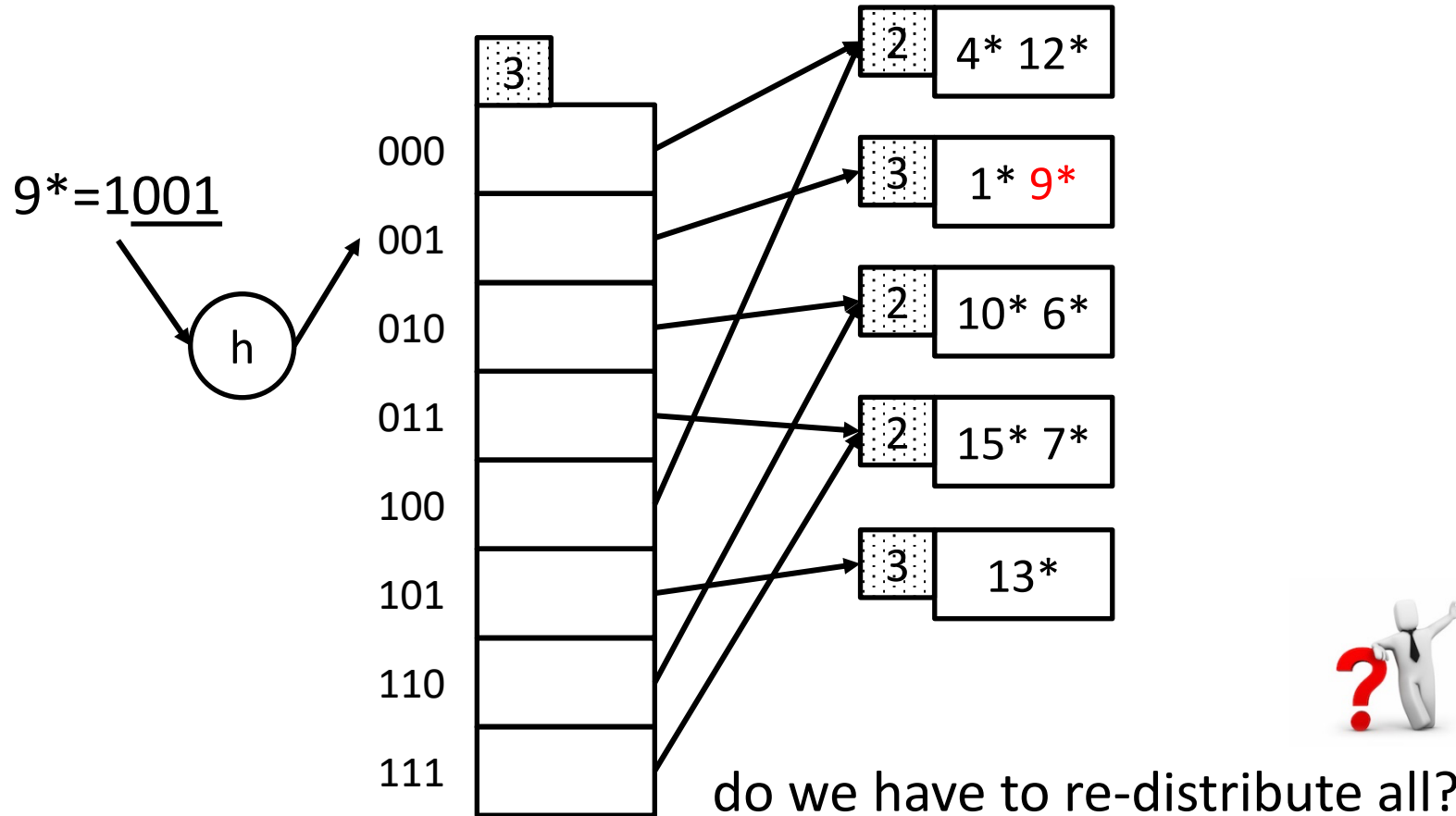


- (1) double the directory
- (2) re-distribute the split bucket
- (3) connect corresponding buckets

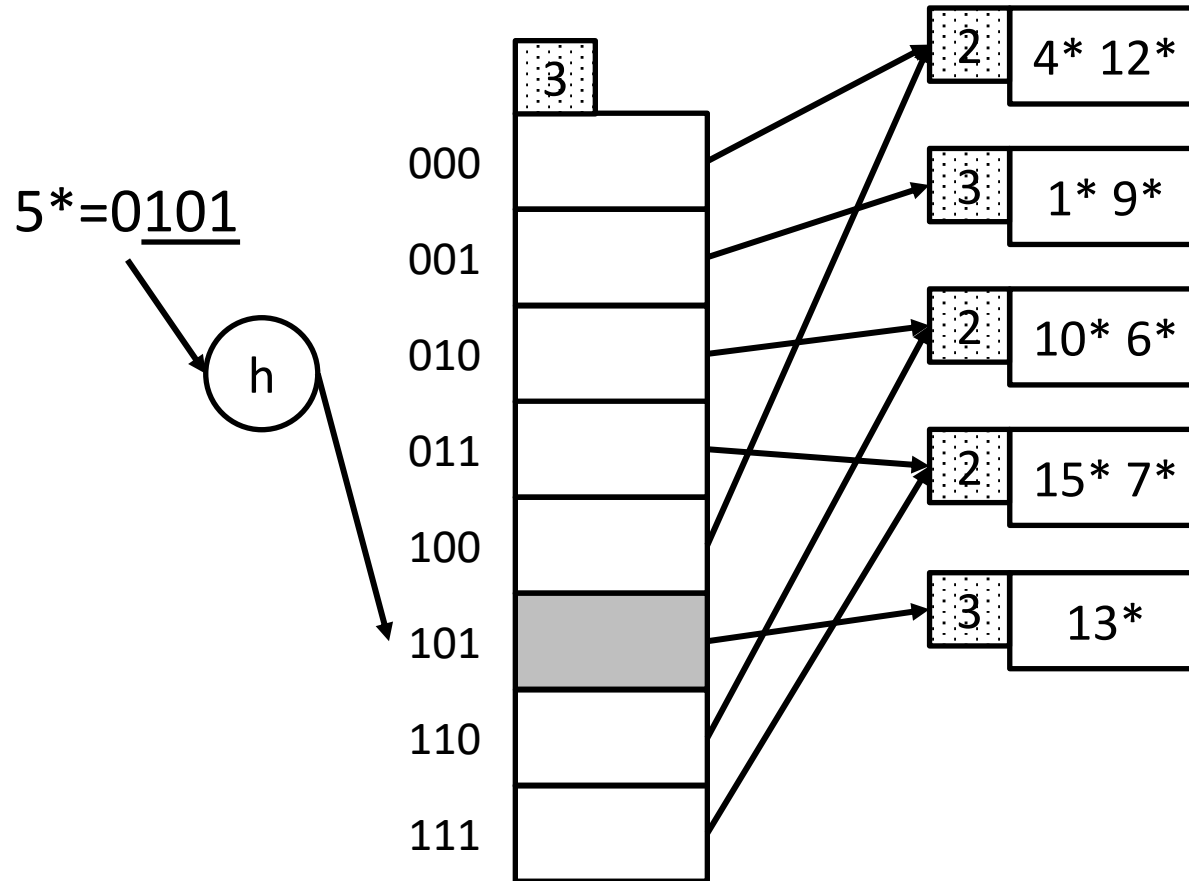
Example 2: Insert 9



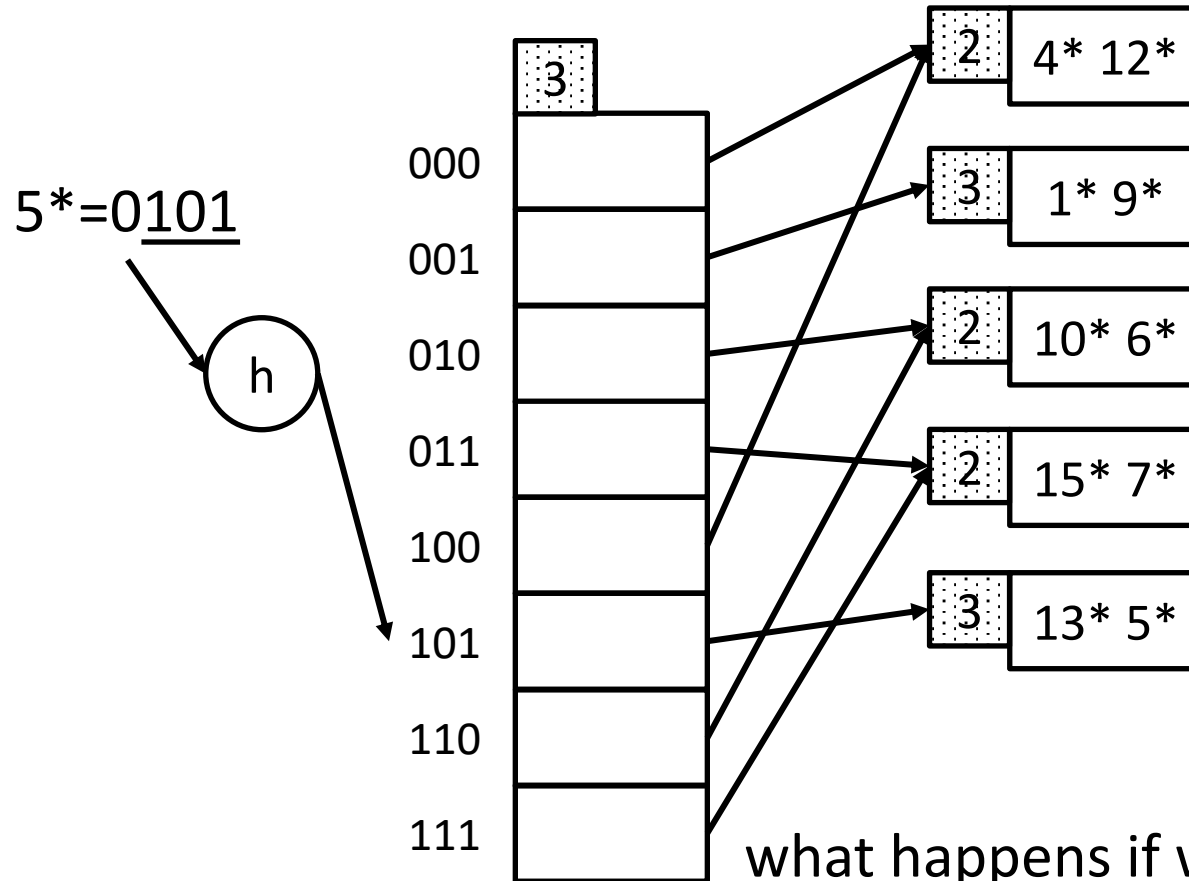
Example 2: Insert 9



Example 3: Insert 5



Example 3: Insert 5

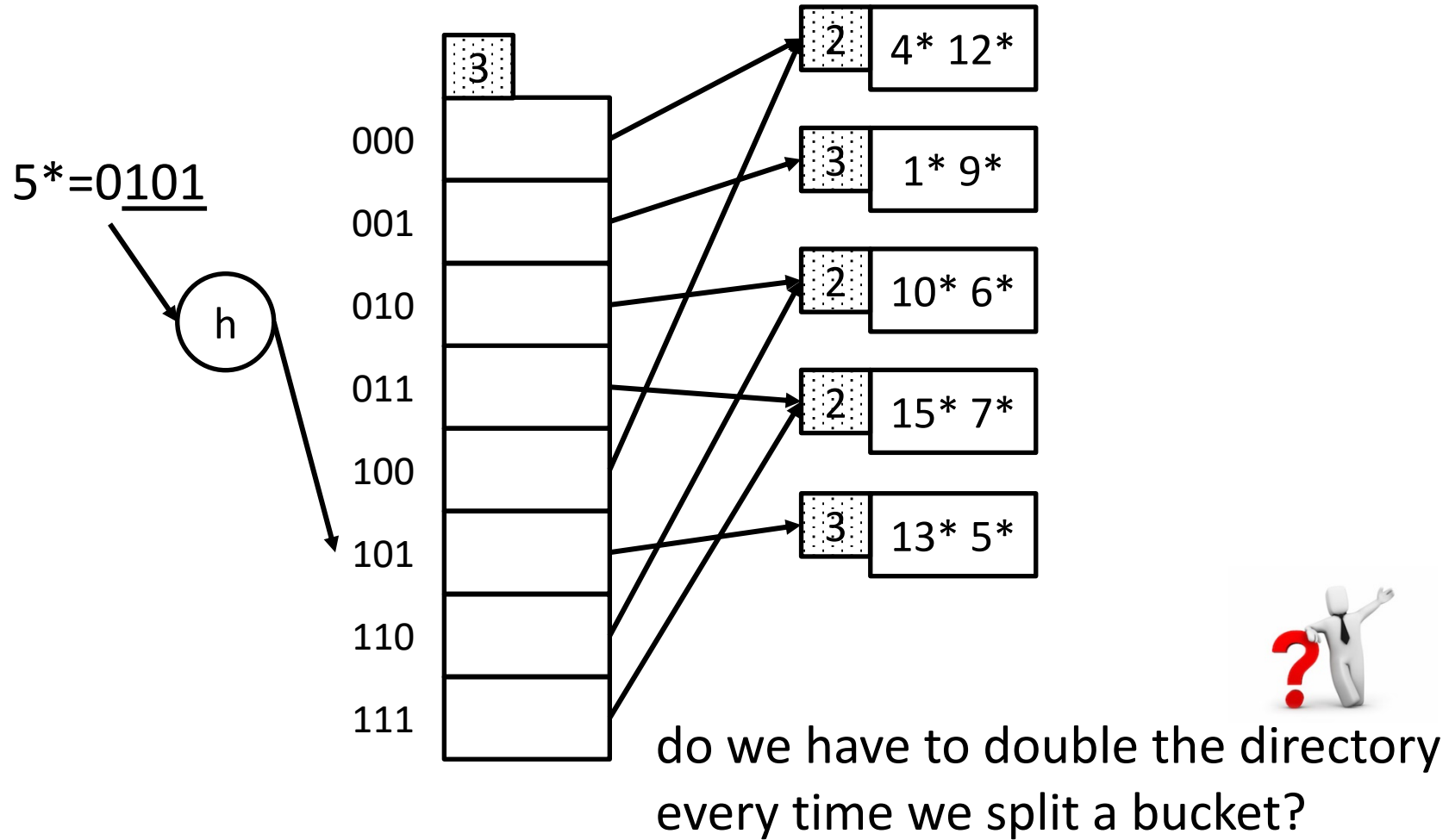


what happens if we want to insert 17?

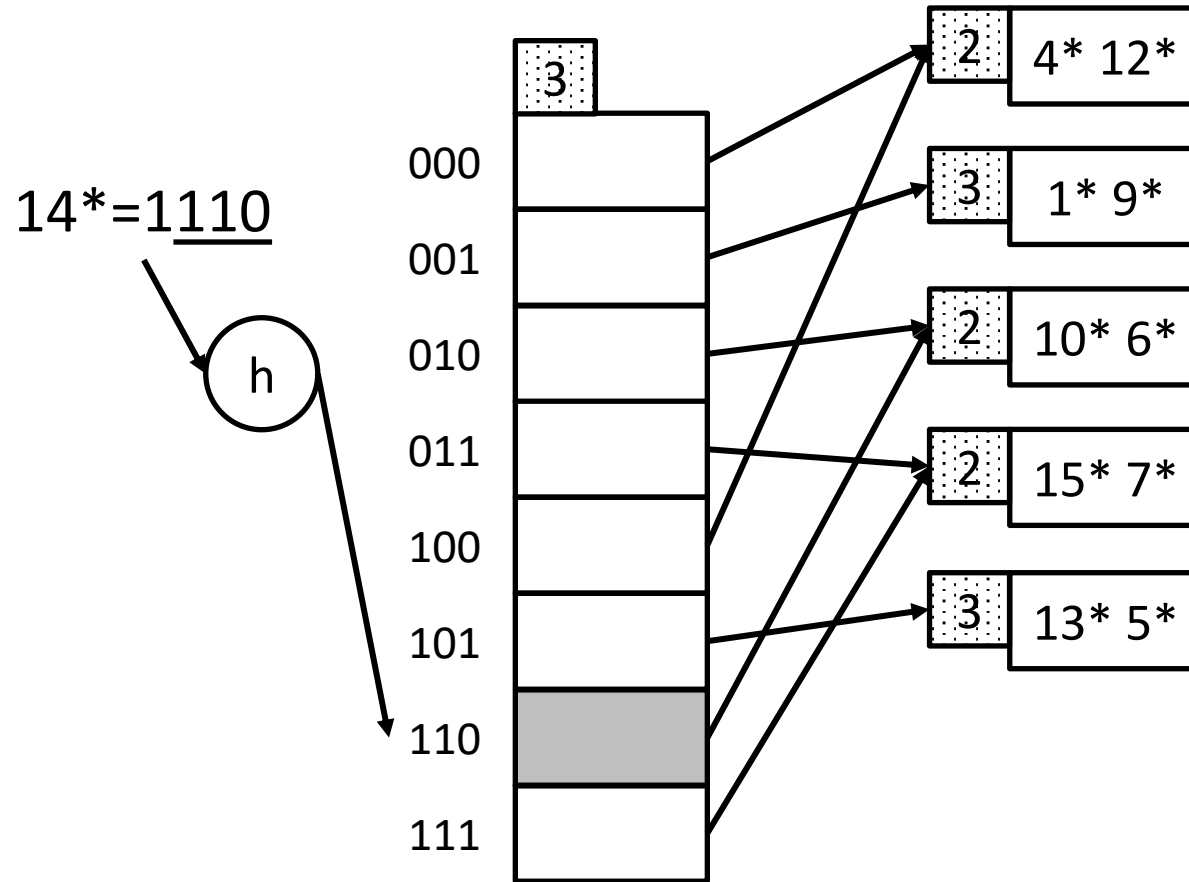
do we have to re-distribute all?

[17→10001] so, double the dir again!

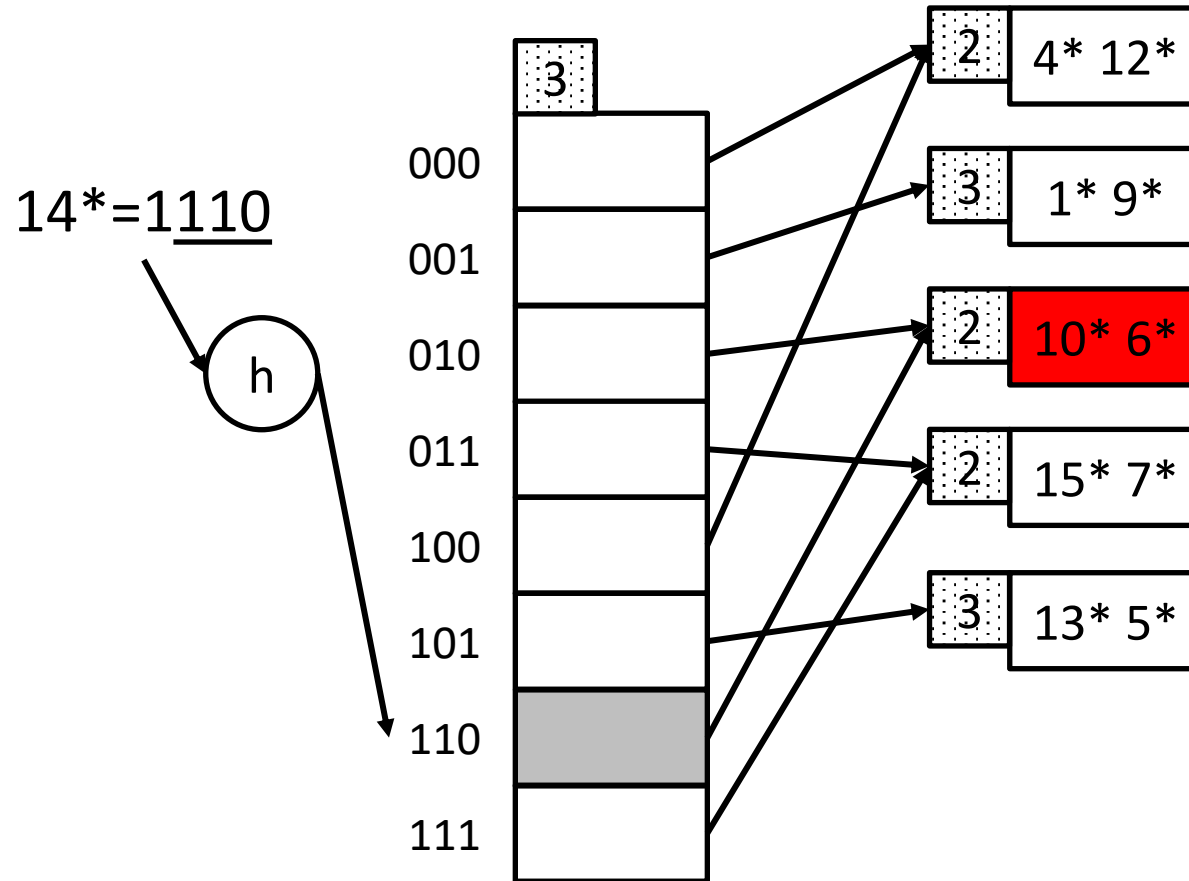
Example 3: Insert 5



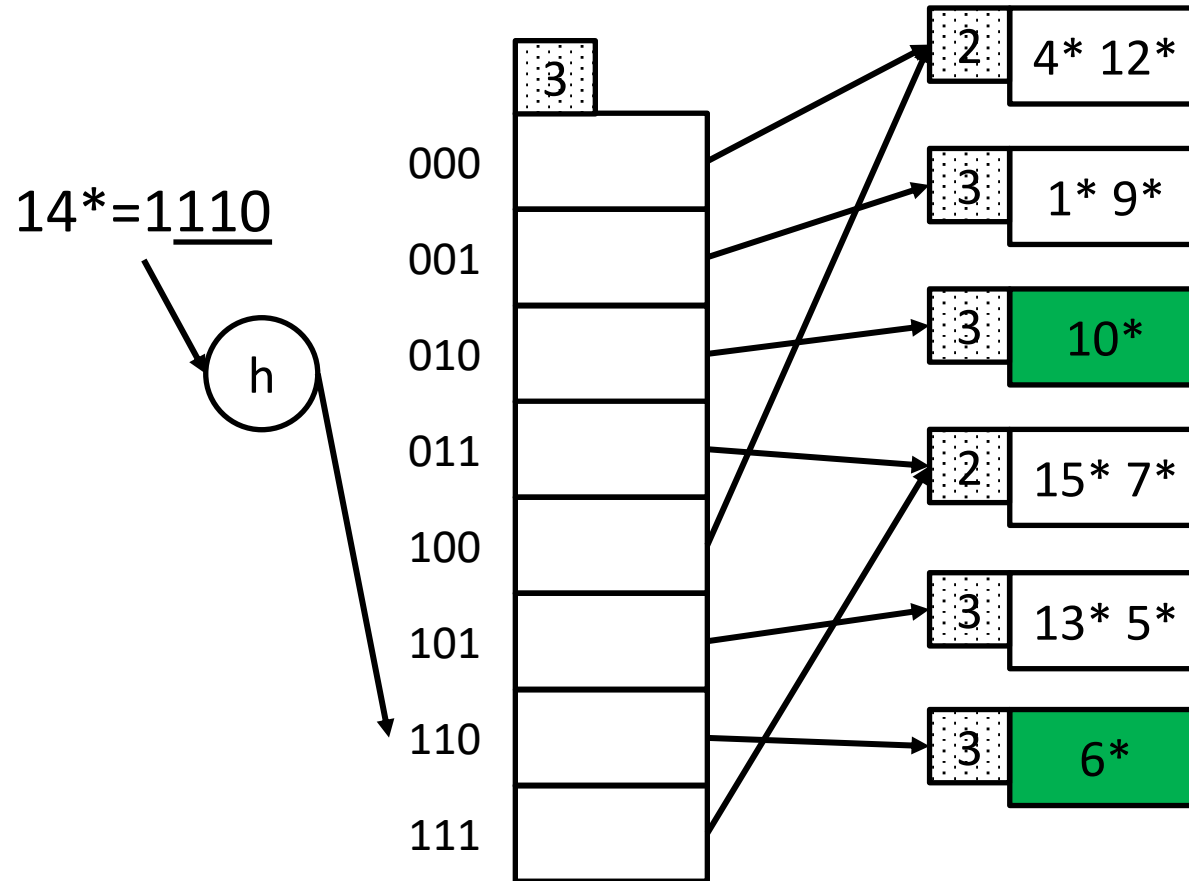
Example 3: Insert 14



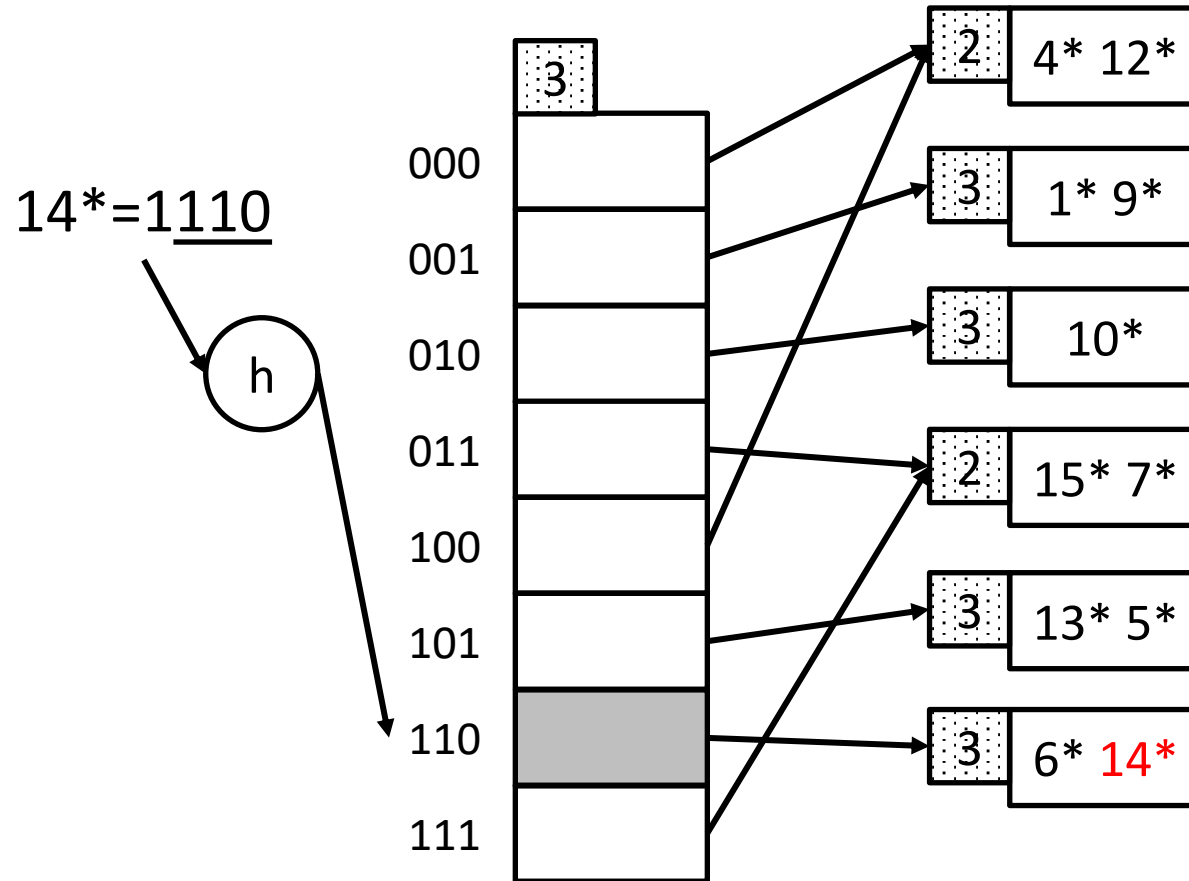
Example 3: Insert 14



Example 3: Insert 14



Example 3: Insert 14



Notes on Extendible 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, can grow large

Notes on Extendible Hashing

Do we ever need overflow pages?

- Multiple entries with same hash value cause problems!

Delete: Reverse of inserts

- Can merge with split image
- Can shrink the directory by half. When?
Each directory element points to same bucket as its split image
- Is shrinking/merging a good idea?



Hash Indexing

Static Hashing

Extendible Hashing

Linear Hashing

Linear Hashing

another dynamic hashing scheme

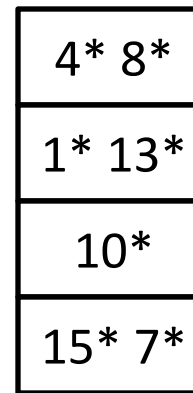
LH handles overflow chains without a directory

Idea: Use overflow pages, and split pages in a round-robin fashion

Example

this for information reasons!
it is not really kept.

h_1	h_0
000	00
001	01
010	10
011	11



Next bucket to split

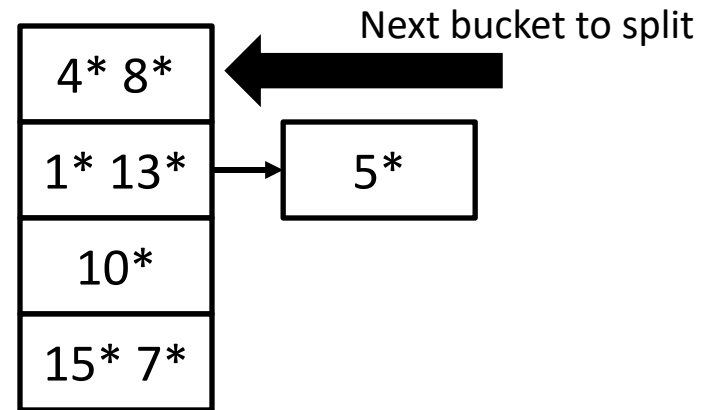


what happens when we insert 5?

Example

this for information reasons!
it is not really kept.

h_1	h_0
000	00
001	01
010	10
011	11



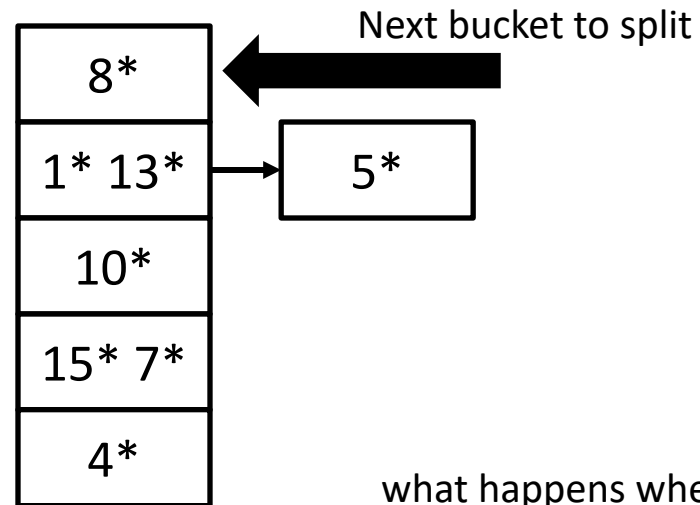
what happens when we insert 5?

(1) 5 goes to an overflow page

Example

this for information reasons!
it is not really kept.

h_1	h_0
000	00
001	01
010	10
011	11
100	



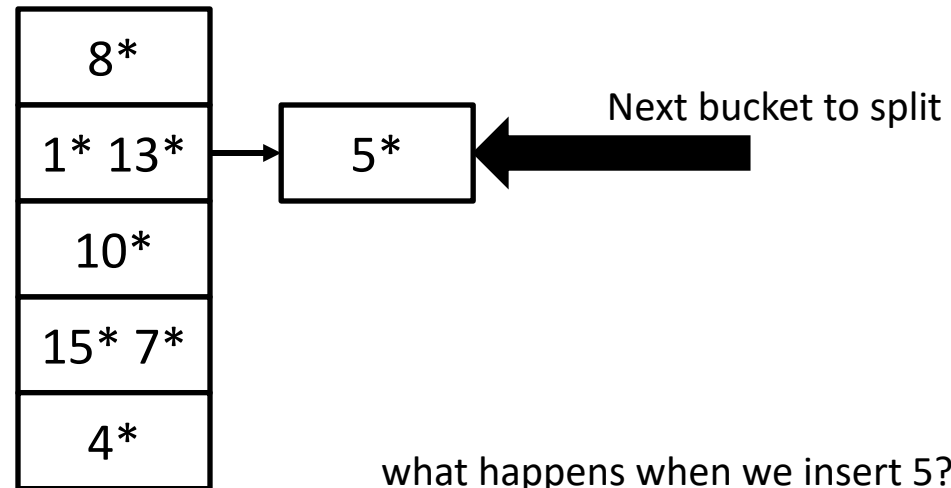
what happens when we insert 5?

- (1) 5 goes to an overflow page
- (2) we split the "next" page

Example

this for information reasons!
it is not really kept.

h_1	h_0
000	00
001	01
010	10
011	11
100	



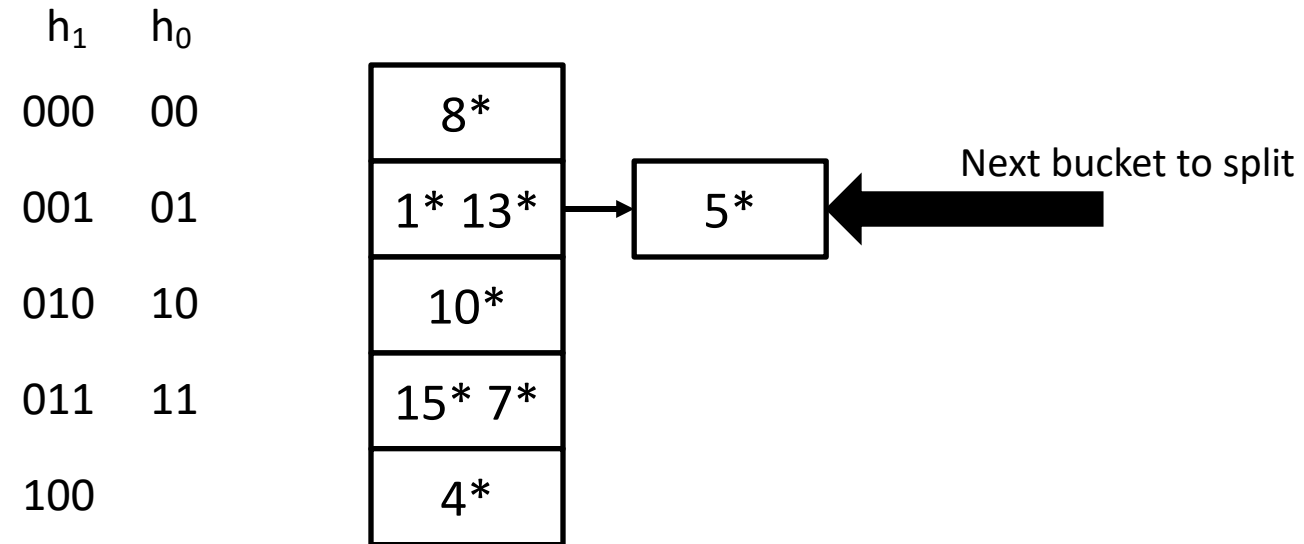
what happens when we insert 5?

- (1) 5 goes to an overflow page
- (2) we split the "next" page
- (3) we move the "next" pointer

Example: Insert 2

this for information reasons!

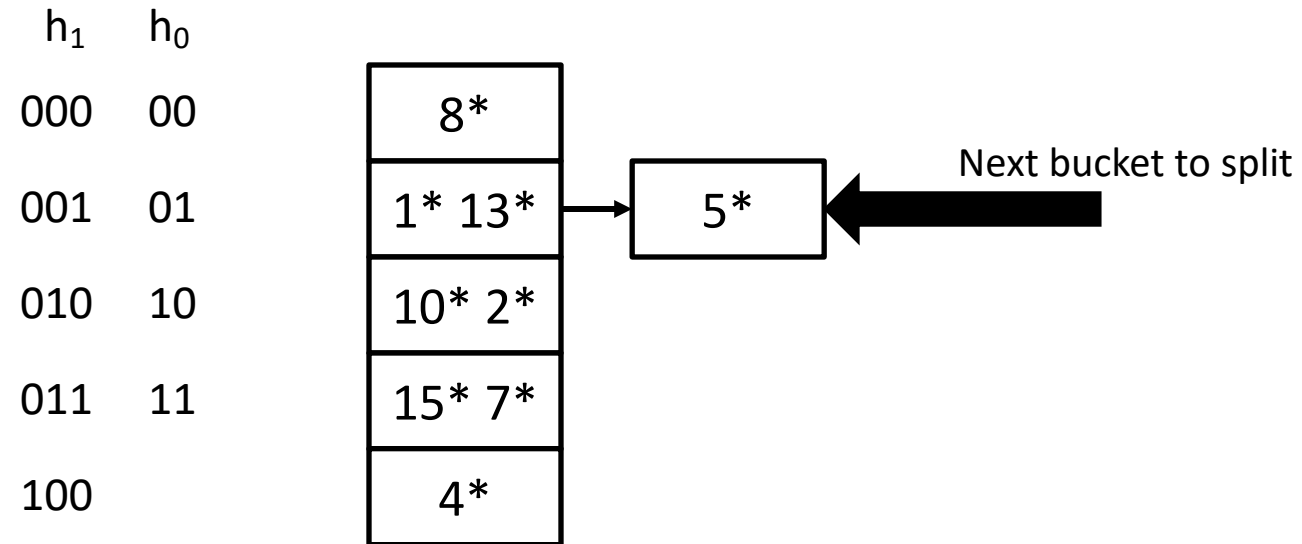
it is not really kept.



Example: Insert 2

this for information reasons!

it is not really kept.



Example: Insert 3

this for information reasons!

it is not really kept.

h_1 h_0

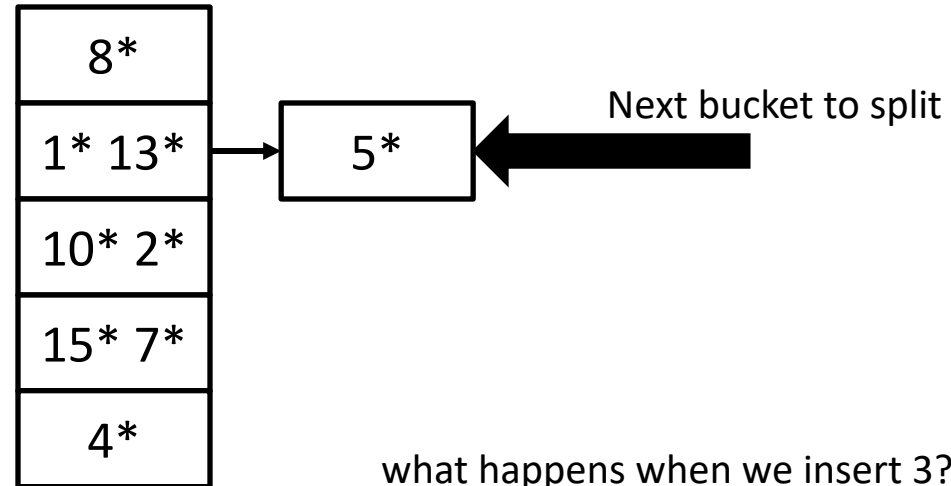
000 00

001 01

010 10

011 11

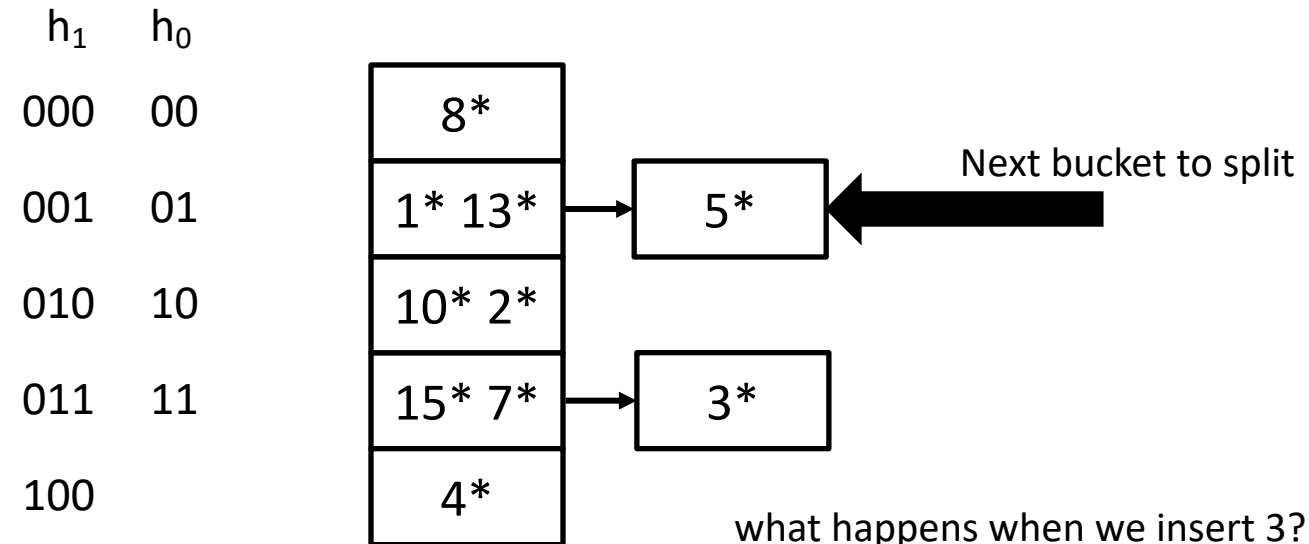
100



Example: Insert 3

this for information reasons!

it is not really kept.



what happens when we insert 3?

(1) 3 goes to an overflow page

Example: Insert 3

this for information reasons!

it is not really kept.

h_1 h_0

000 00

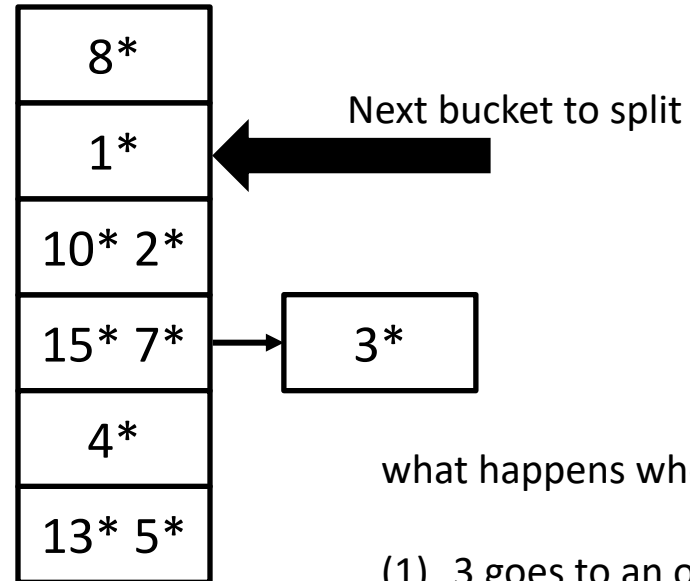
001 01

010 10

011 11

100

101



what happens when we insert 3?

- (1) 3 goes to an overflow page
- (2) we split the "next" page

Example: Insert 3

this for information reasons!

it is not really kept.

h_1 h_0

000 00

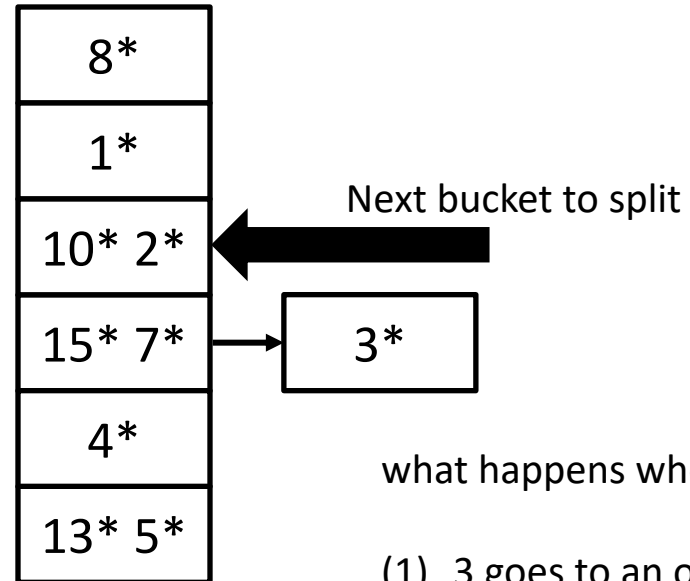
001 01

010 10

011 11

100

101



what happens when we insert 3?

- (1) 3 goes to an overflow page
- (2) we split the "next" page
- (3) we move the "next" pointer

Linear Hashing

$h_0, h_1, h_2 \dots$ can be more general hash functions

when h_0 hits on a split buffer we employ h_1 and we have to look in both buffers

if the second is also split we use h_2 and so on

Benefit: buckets are split round-robin
→ no long chains

Hash Indexing

Hash indexes: best for equality searches

Static Hashing can lead to long overflow chains

Extendible Hashing

- avoids overflow pages by splitting a bucket when full
- directory to keep track of buckets
- dir. can get too large (>memory) when data is skewed

Linear Hashing

- avoids directory by splitting buckets round-robin
- uses overflow pages
- overflow pages not likely to be long