Hash-Based Indexes

Chapter 11

Index Design Space

Organization Structure for k*

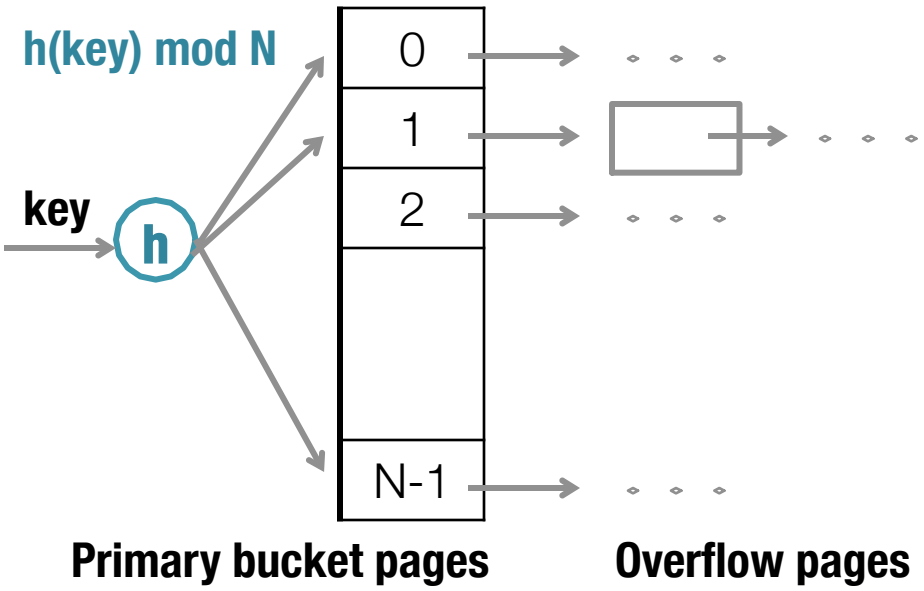
- Tree-based
  (+) Range, equality search
- Hash-based
  (+) Equality search
  - Static hashing
  - Extendible hashing
  - Linear hashing

Data Entry (k*) Contents

1. Actual Data record
   index = file
2. <k, rid>
   actual records in a diff 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 downsides of static hashing?

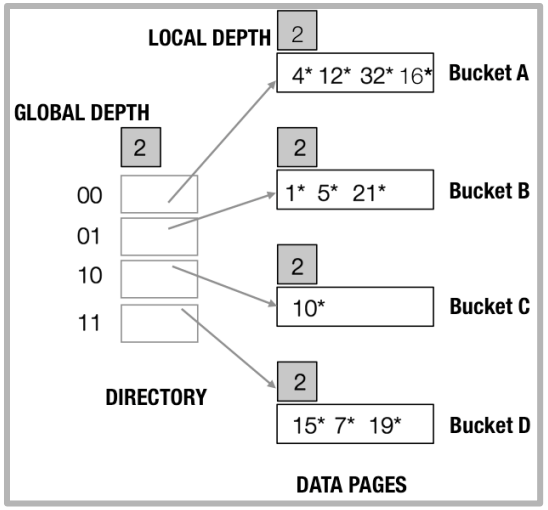
Static Hashing (Contd.)

- 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
  - Dynamic hashing fixes this problem: Extendible and Linear Hashing

Why not start w/ a very large N?

Extendible Hashing

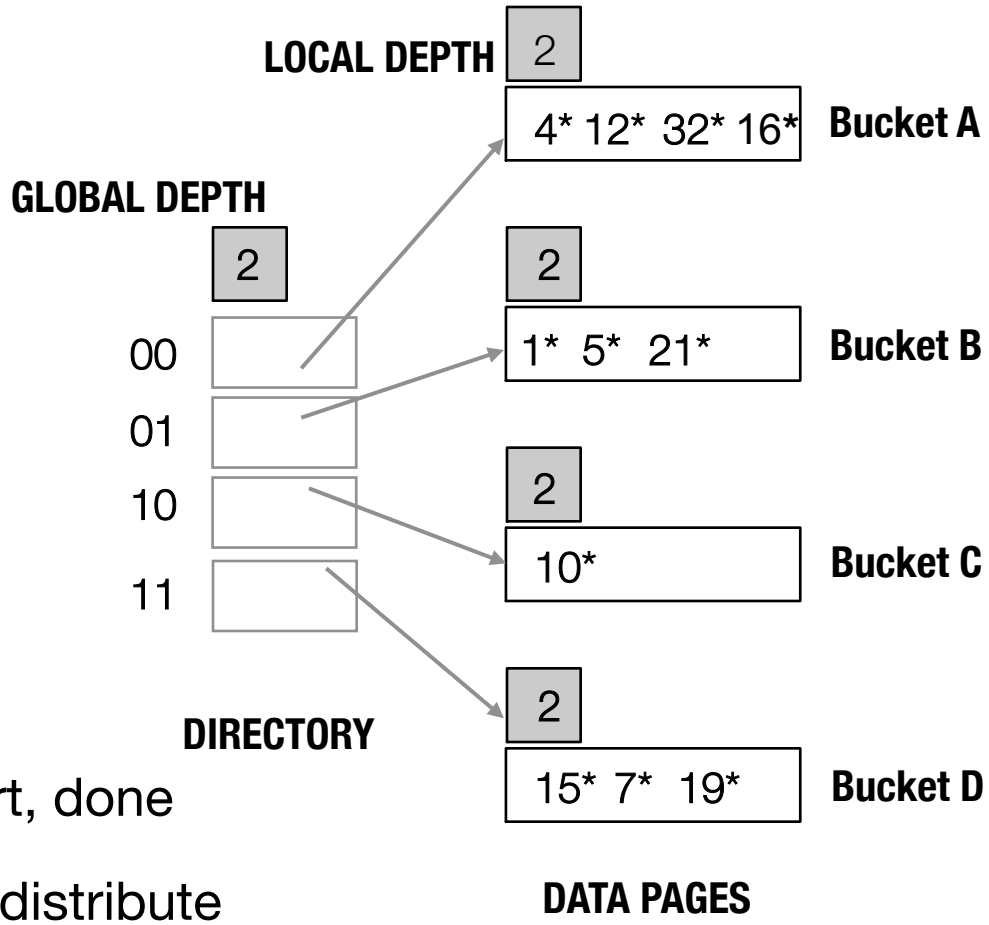
- Main Idea: Use a directory of pointers to buckets
- On overflow, double the directory (not # number of buckets)



- Why does this help?
  - Directory much smaller than the file
  - Only one page of data entries is split at a time
  - (Usually*) No overflow pages

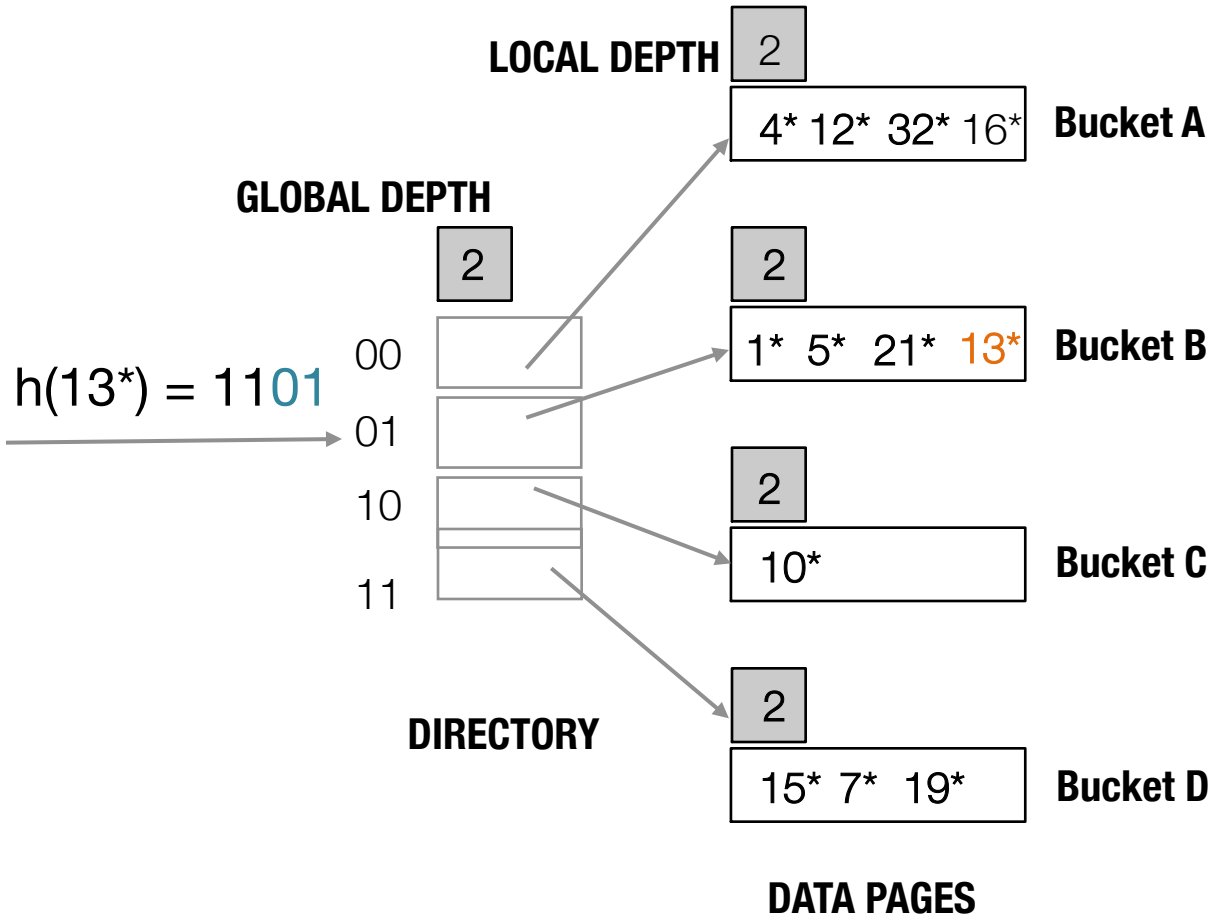
Extendible Hashing: Algorithm

- 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

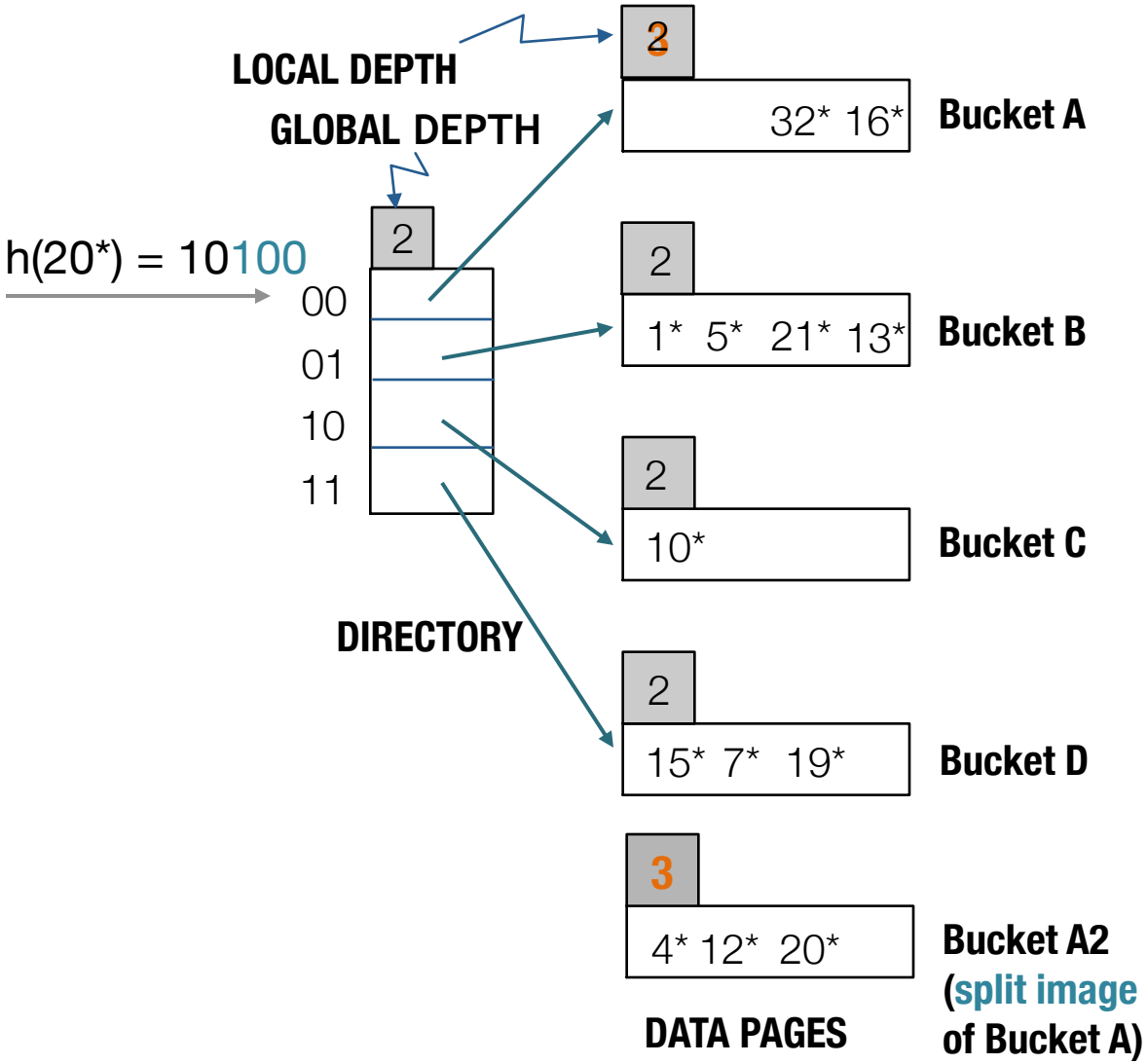


Example: Insertion into a free bucket

- Insert 13*
- Suppose
  h(13*) = 1101



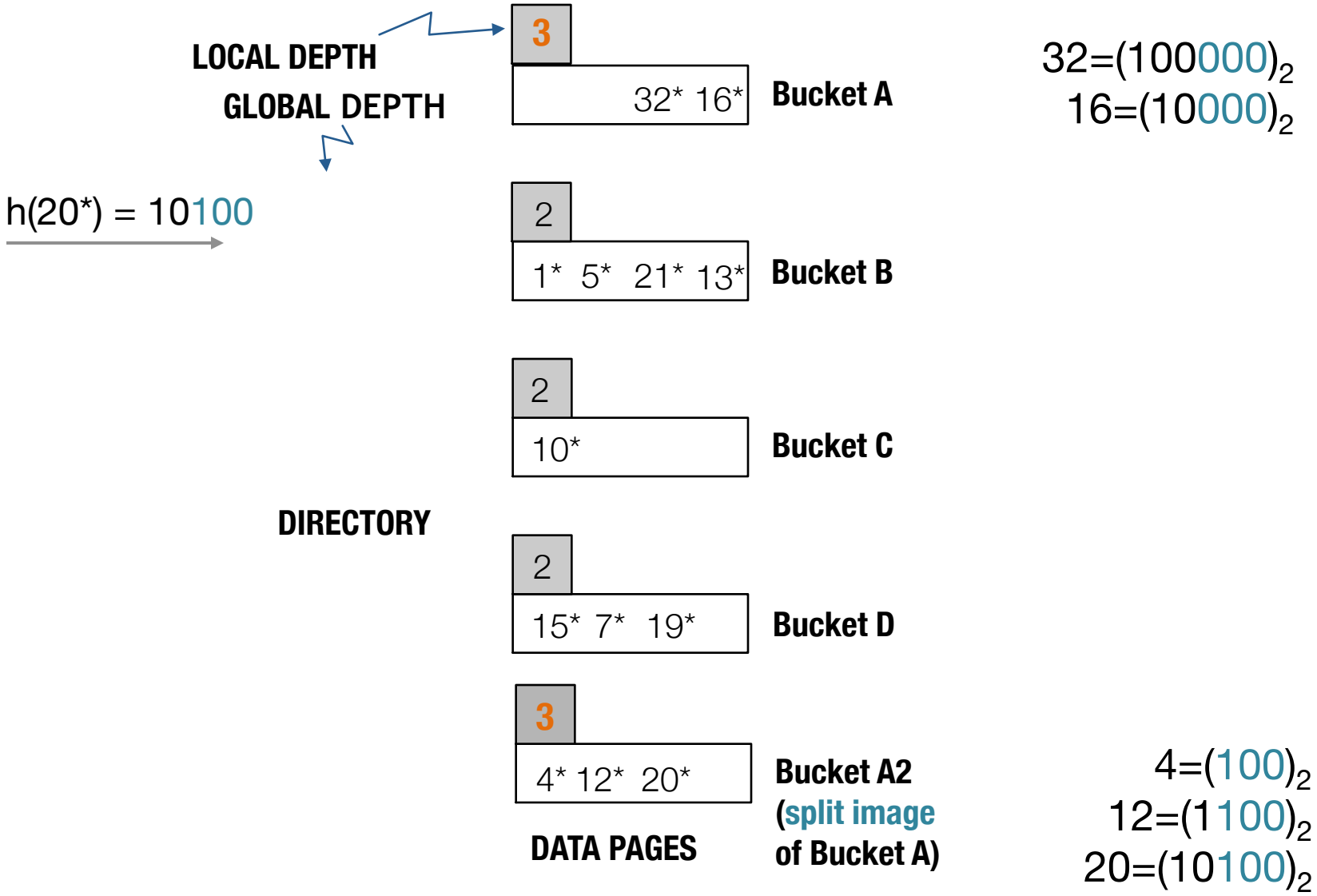
Example: Insertion into a full bucket
Insert 20 = $(10100)_2$



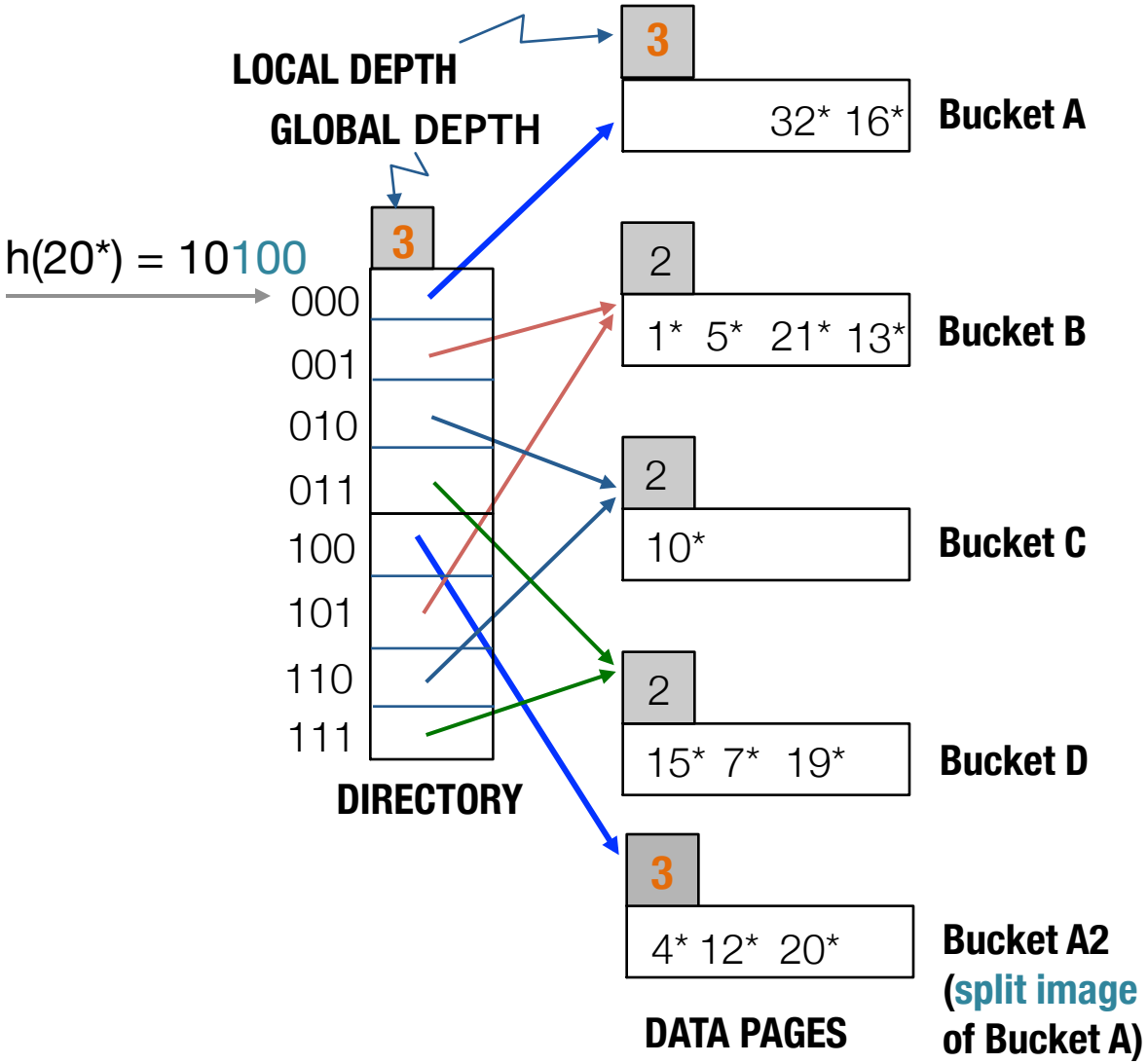
$32=(100000)_2$
$16=(10000)_2$

$4=(100)_2$
$12=(1100)_2$
$20=(10100)_2$

Example: Insertion into a full bucket
Insert 20 = $(10100)_2$



Example: Insertion into a full bucket
Insert 20 = $(10100)_2$



$32=(100000)_2$
$16=(10000)_2$

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

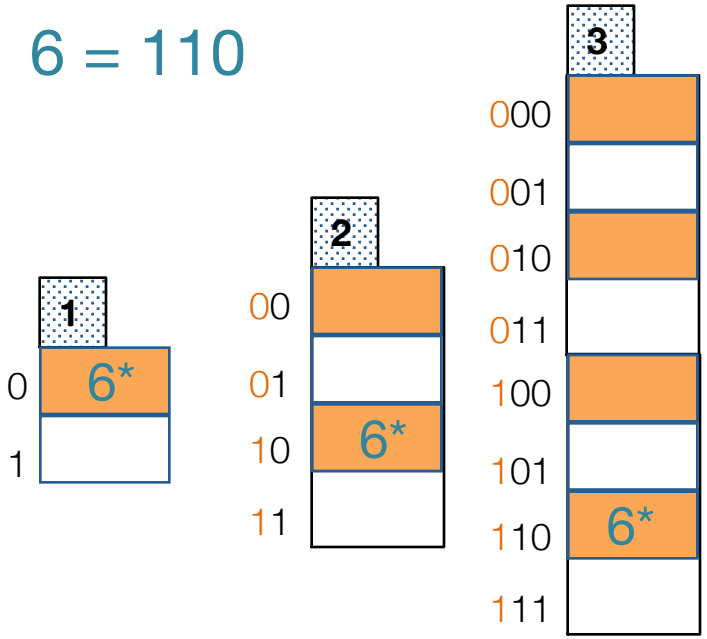
Directory doubled by *copying it over* and fixing pointer to split image page

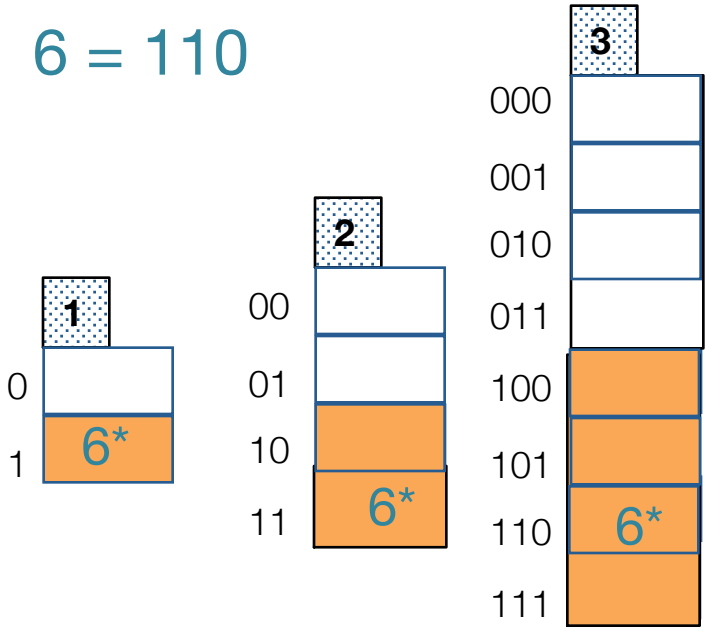
$4=(100)_2$
$12=(1100)_2$
$20=(10100)_2$

Directory Doubling

Why use least significant bits in directory?

Allows for doubling via copying!



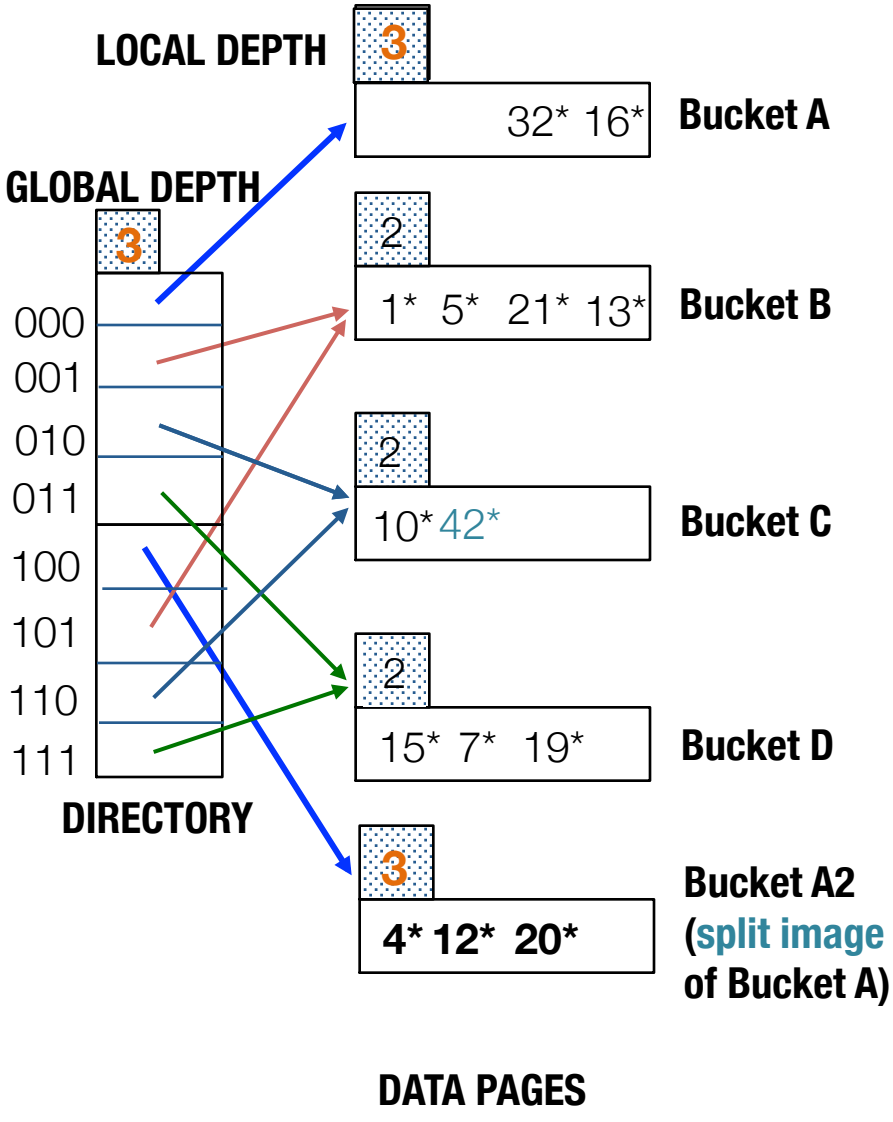


Least Significant vs. Most Significant

Comments 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, directory can grow large
- Do we ever need overflow pages?
  - Multiple entries with same hash value cause problems!
    => We need overflow pages
- Delete: Reverse of inserts – see textbook

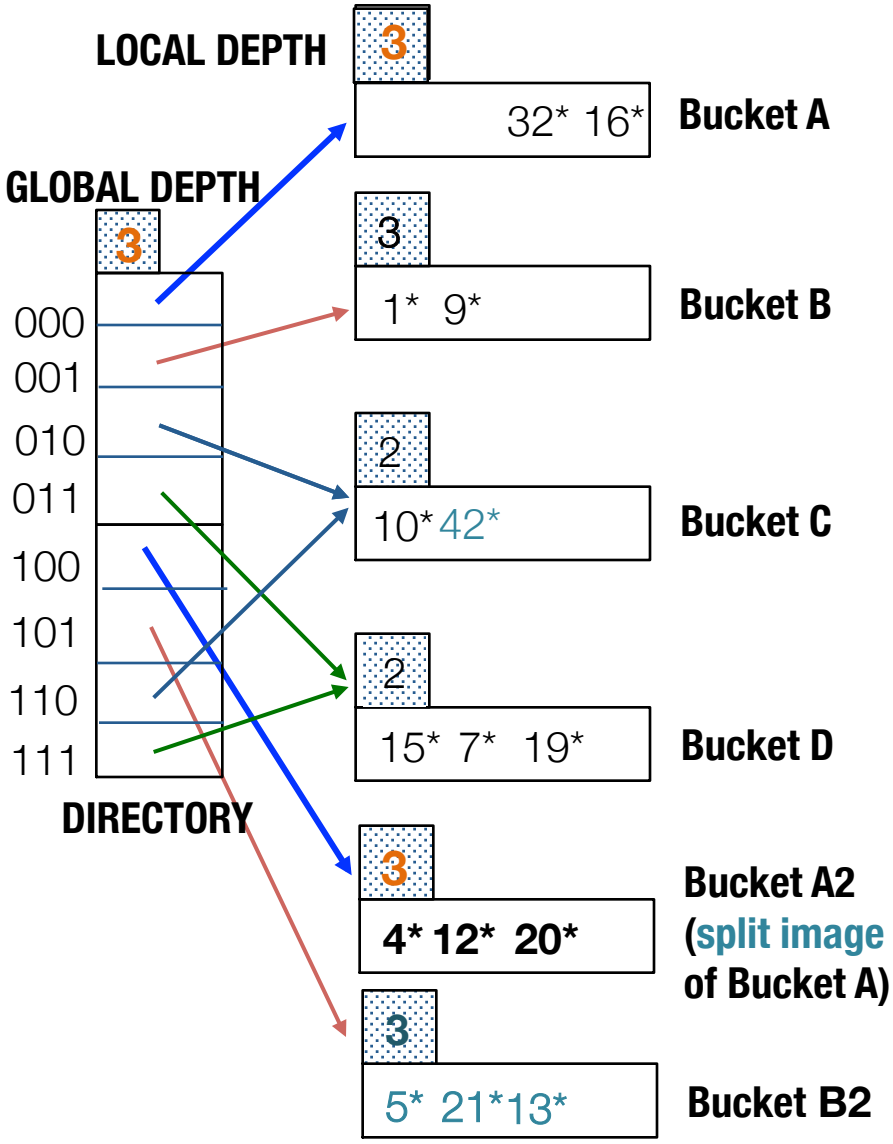
Extendible Hashing Quiz: Insert 42, 9



$42=(101010)_2$

$9=(1001)_2$

Answer: Insert 42, 9

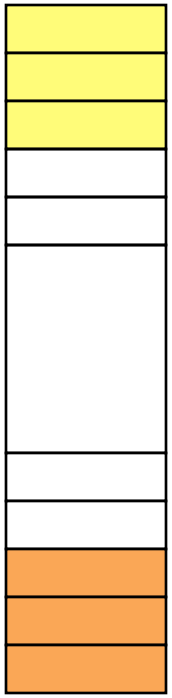


$42=(101010)_2$

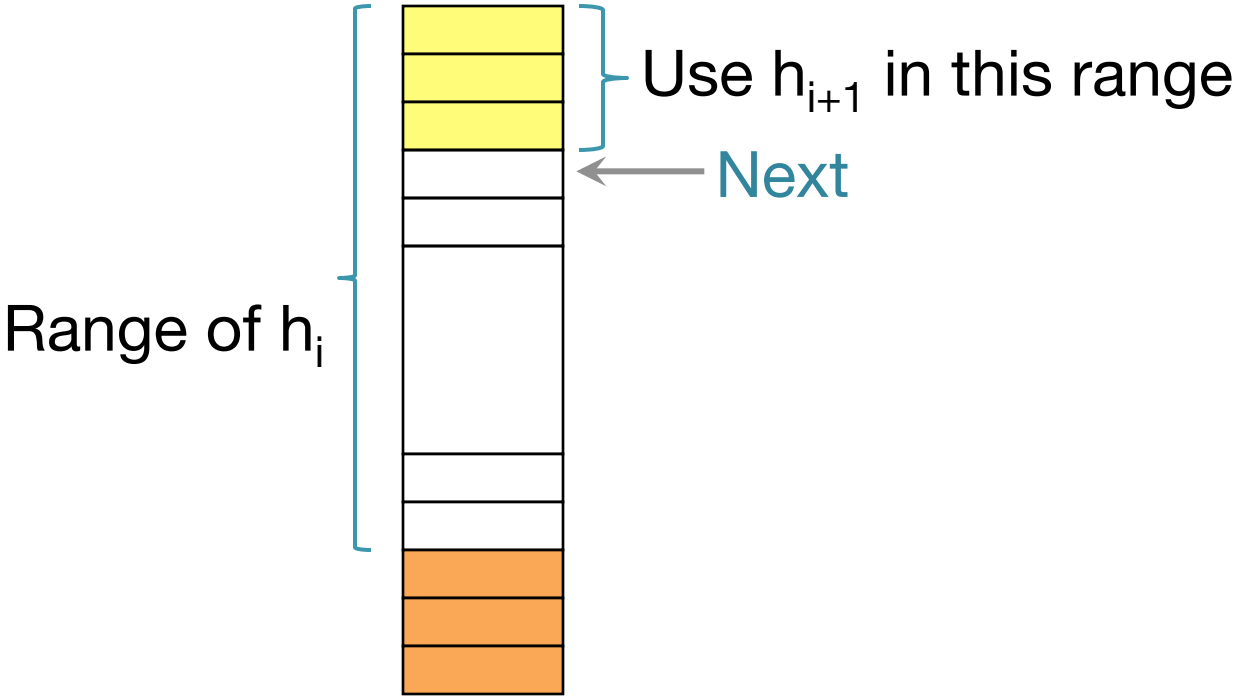
$9=(1001)_2$

Linear Hashing

- Tries to overcome some of the drawbacks of Extendible Hashing
- Works in rounds
- Grows more gradually

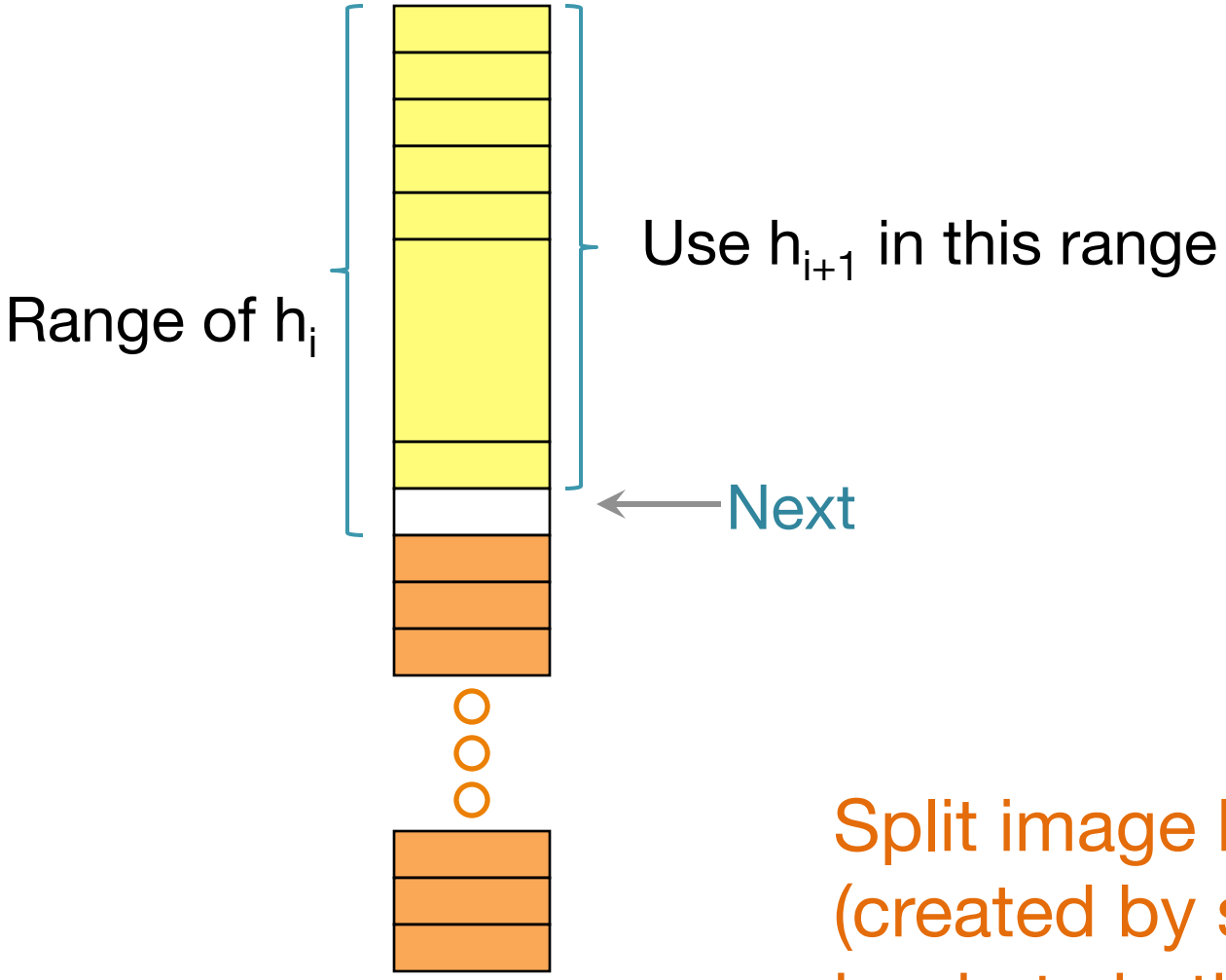


Buckets During Round *i*



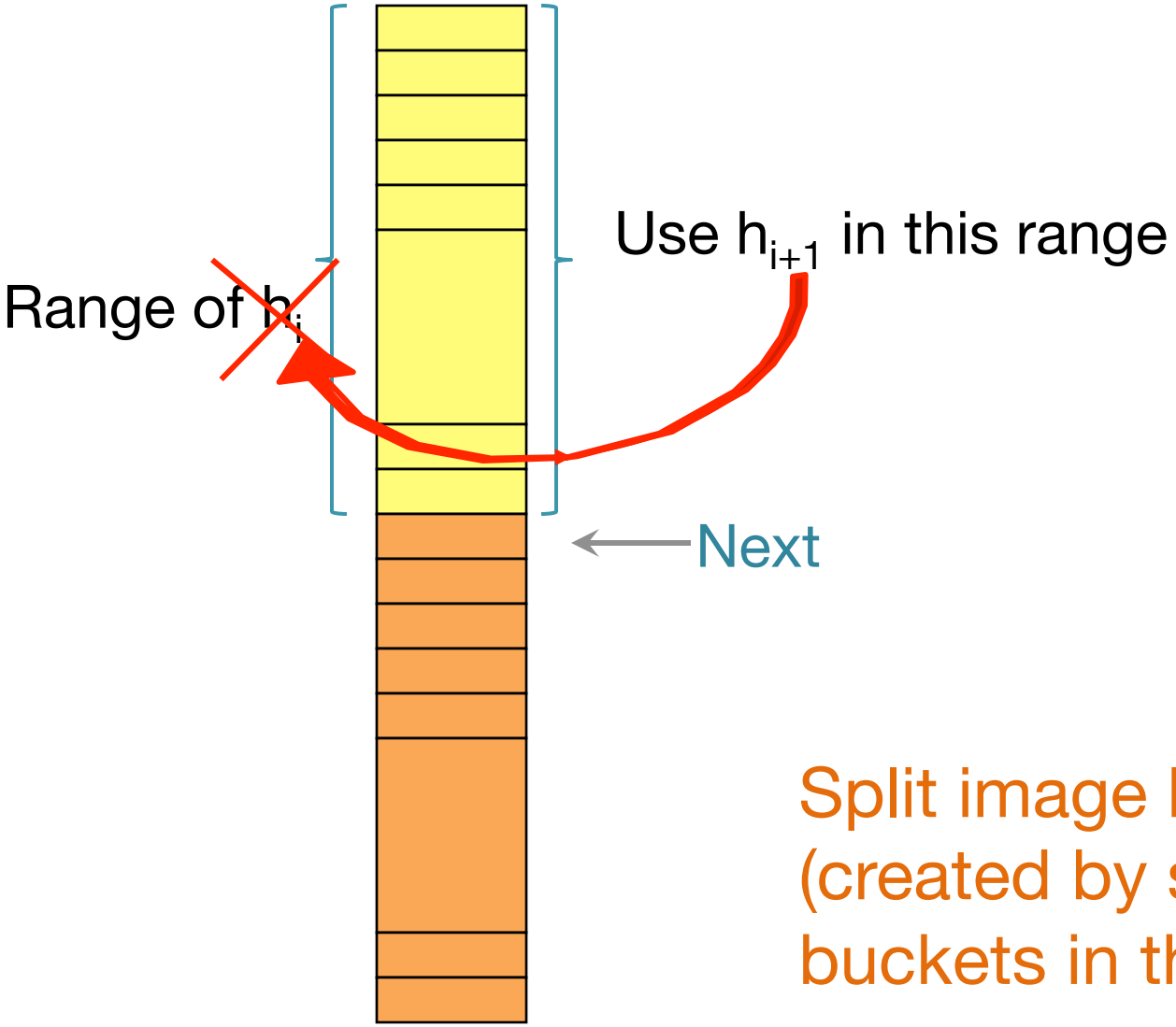
Split image buckets (created by splitting other buckets in this round)

Buckets During Round *i*



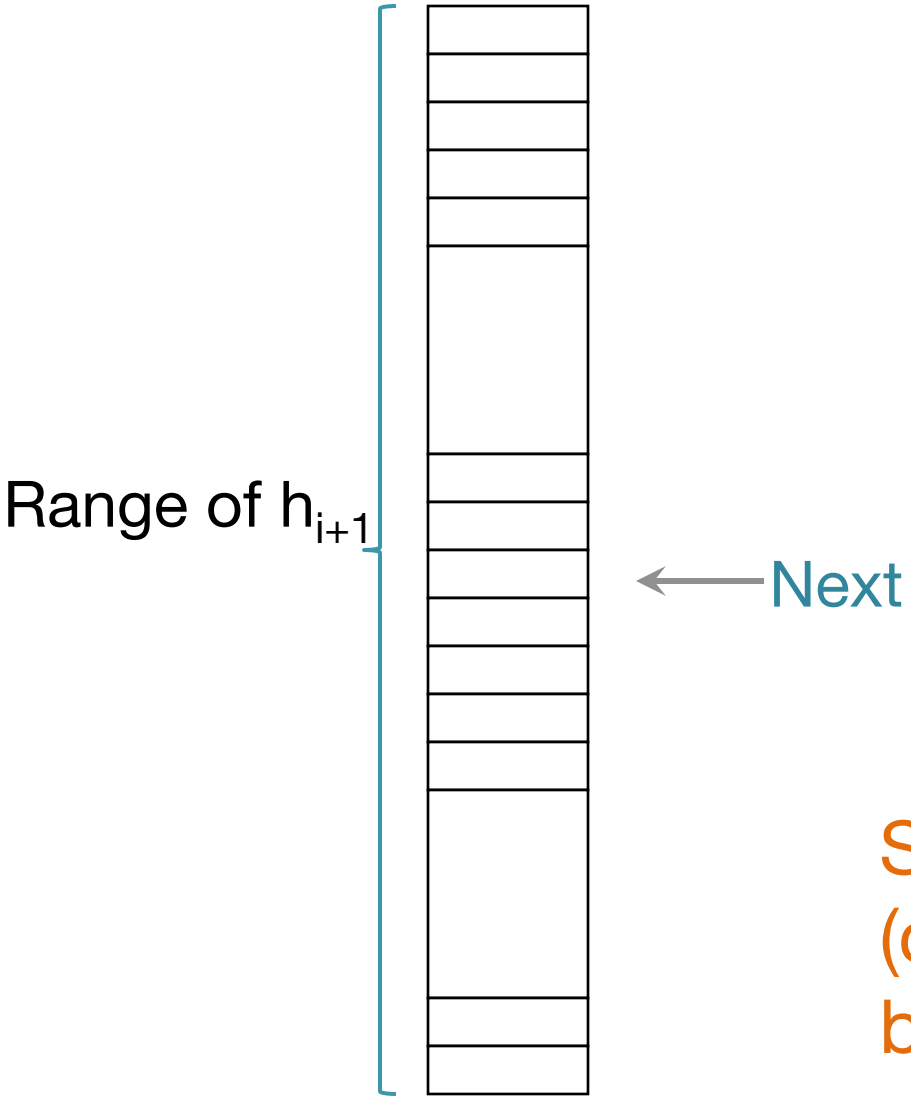
Split image buckets (created by splitting other buckets in this round)

End of Round *i* = Start of Round *i+1*



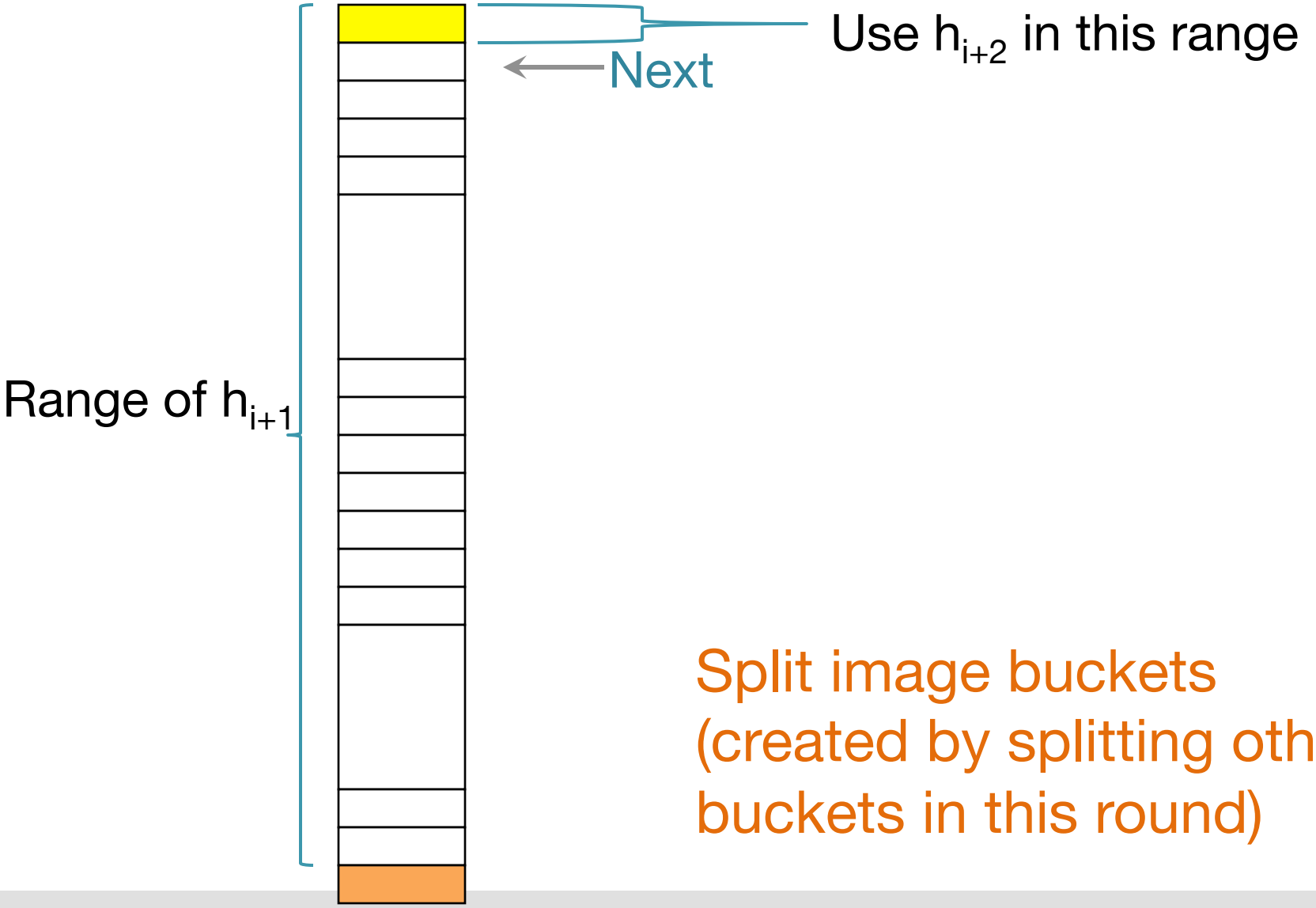
Split image buckets (created by splitting other buckets in this round)

End of Round *i* = Start of Round *i+1*



Split image buckets (created by splitting other buckets in this round)

Round *i+1* after the First Split



Split image buckets (created by splitting other buckets in this round)

Linear Hashing: Main Idea

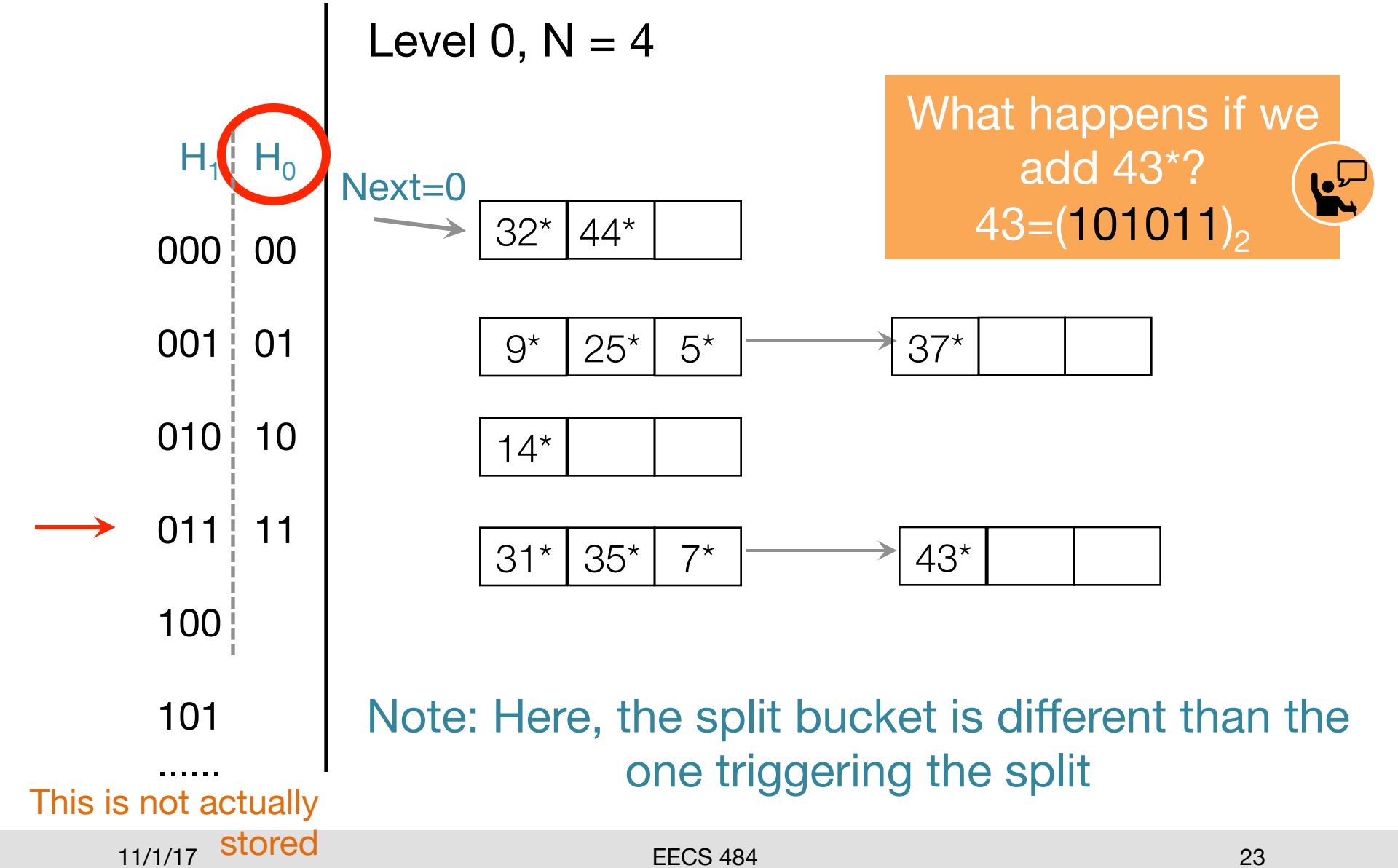
- Another dynamic hashing scheme
- Eliminates long overflow chains without using a directory
- **Idea:** Use a family of hash functions: $\mathbf{h_0}$, $\mathbf{h_1}$, $\mathbf{h_2}$, ...
  - h_{i+1} doubles the range of h_i (similar to directory doubling)
  - Hash family typically obtained by choosing hash function $h(\ )$ and initial number of buckets N
  - Define $h_i(value) = h(value)\ mod\ (2^i N)$
  - If N is a power of 2, say $\mathbf{N=2^{d_0}}$, apply hash function $h(\ )$, and look at last $\boldsymbol{d_i}$ bits where $\mathbf{d_i = d_0 + i}$

Linear Hashing: Details

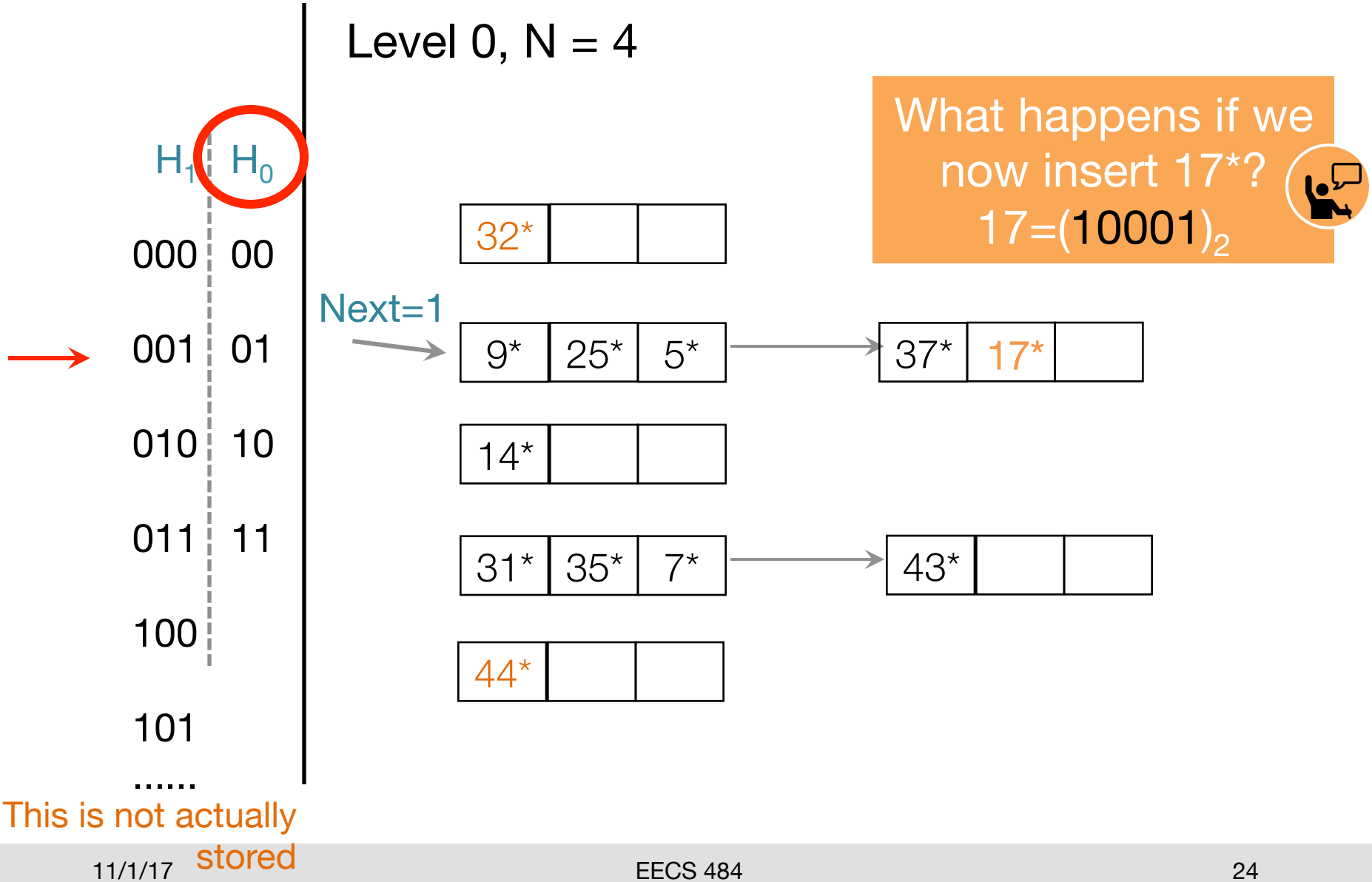
- Splitting proceeds in rounds
  - During round *Level*, only $\mathbf{h}_{Level}$ and $\mathbf{h}_{Level+1}$ are in use
- Variables
  - Level: Initialized to 0
  - Next: Pointer to the bucket being split
- At the beginning of round # Level,
  - # buckets in the file = $N_{Level} = N * 2^{Level}$

  where N is initial number of buckets

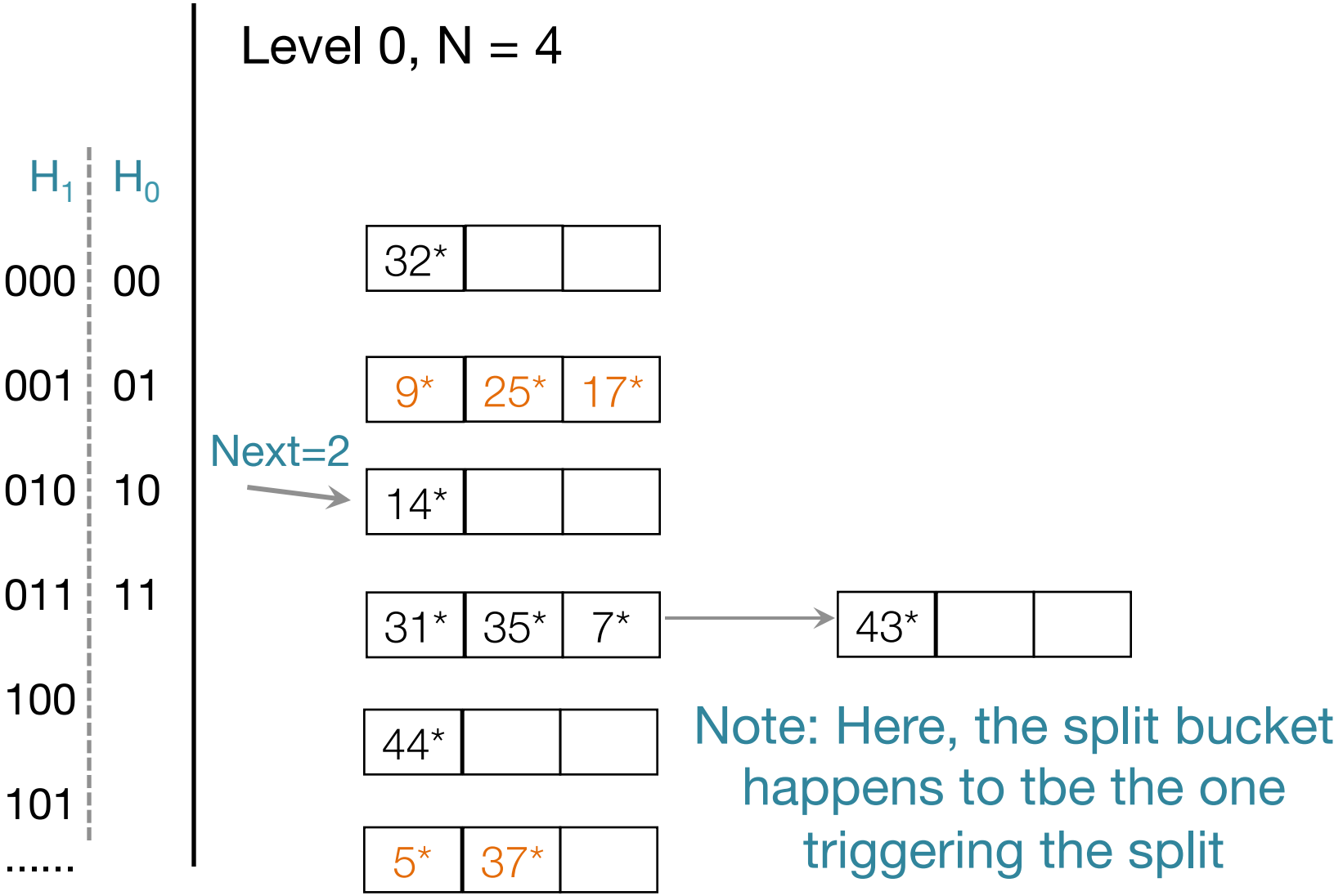
Linear Hashing Example



Linear Hashing Example



Linear Hashing Example



Linear Hashing Algorithm

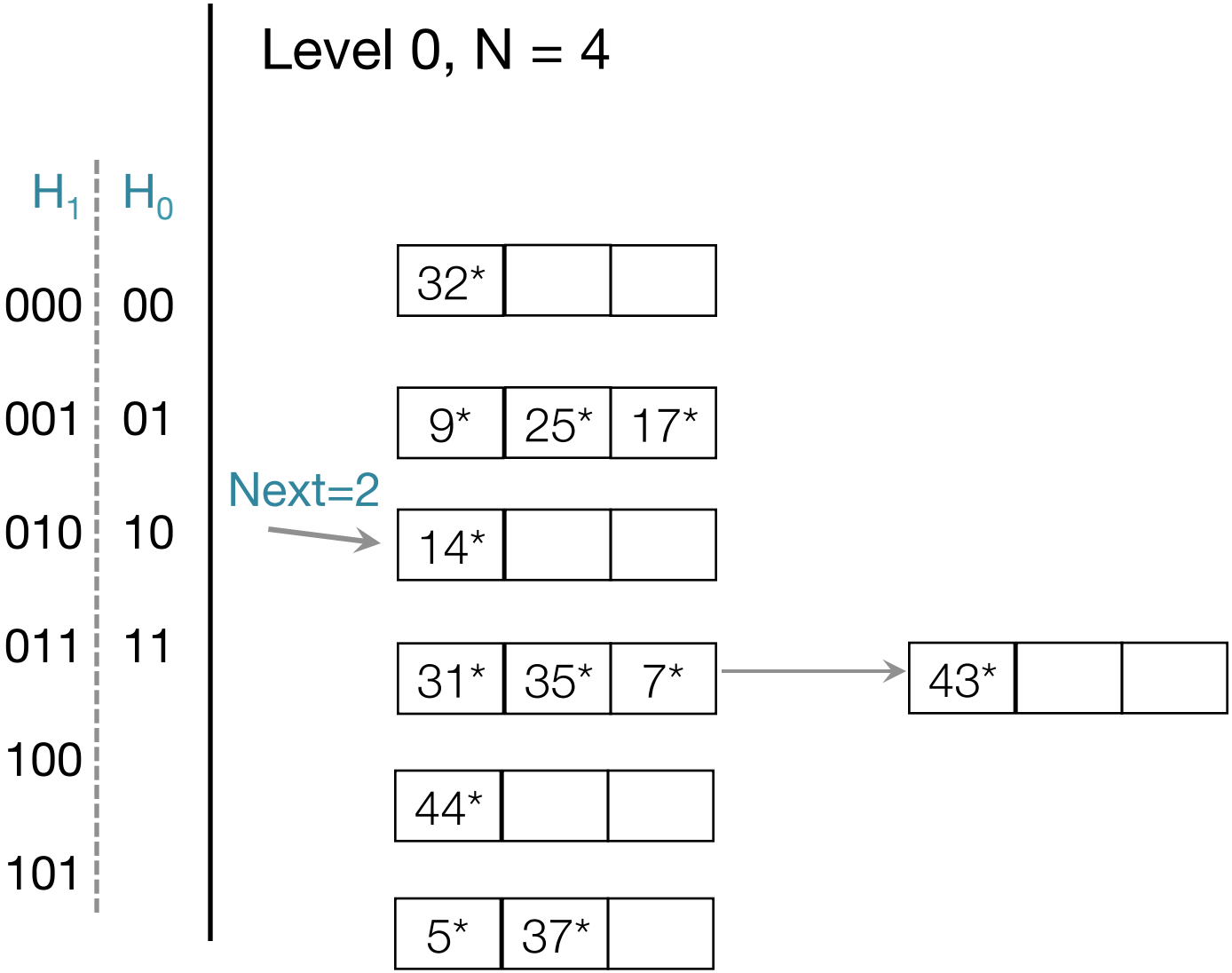
Insert:

- Find bucket by applying $\mathbf{h}_{Level}$
  - Apply $\mathbf{h}_{Level+1}$ if the value returned by $\mathbf{h}_{Level}$ < Next
- If bucket being inserted into is full:
  - Add overflow page and insert data entry
    - If bucket is Next, split first and then see if you still need overflow
  - Split *Next* bucket and increment Next
    - If Next = N_{Level} – 1 and a split is triggered
      1. We split the bucket # N_{Level}-1 (i.e. the last bucket before the split image buckets)
      2. Next = 0 (reset the pointer)
      3. Level = Level + 1 (the size has doubled at this point)

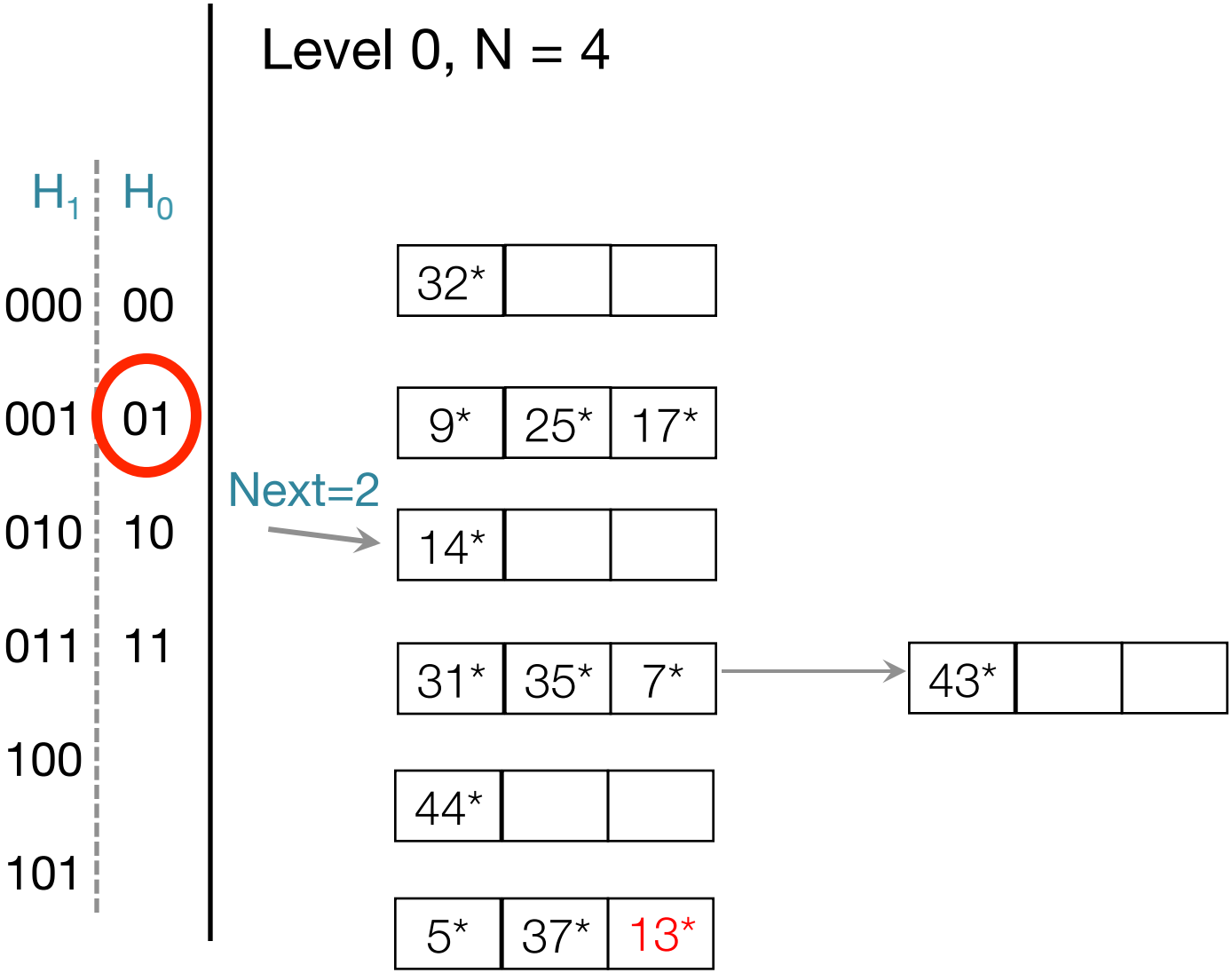
Linear Hashing (Contd.)

- Can choose any criterion to trigger split
  - e.g., split on an overflow (as in example)
  - e.g., space utilization on the page > 90%
- Since buckets are split round-robin, long overflow chains don't develop!
- Deletes: see textbook

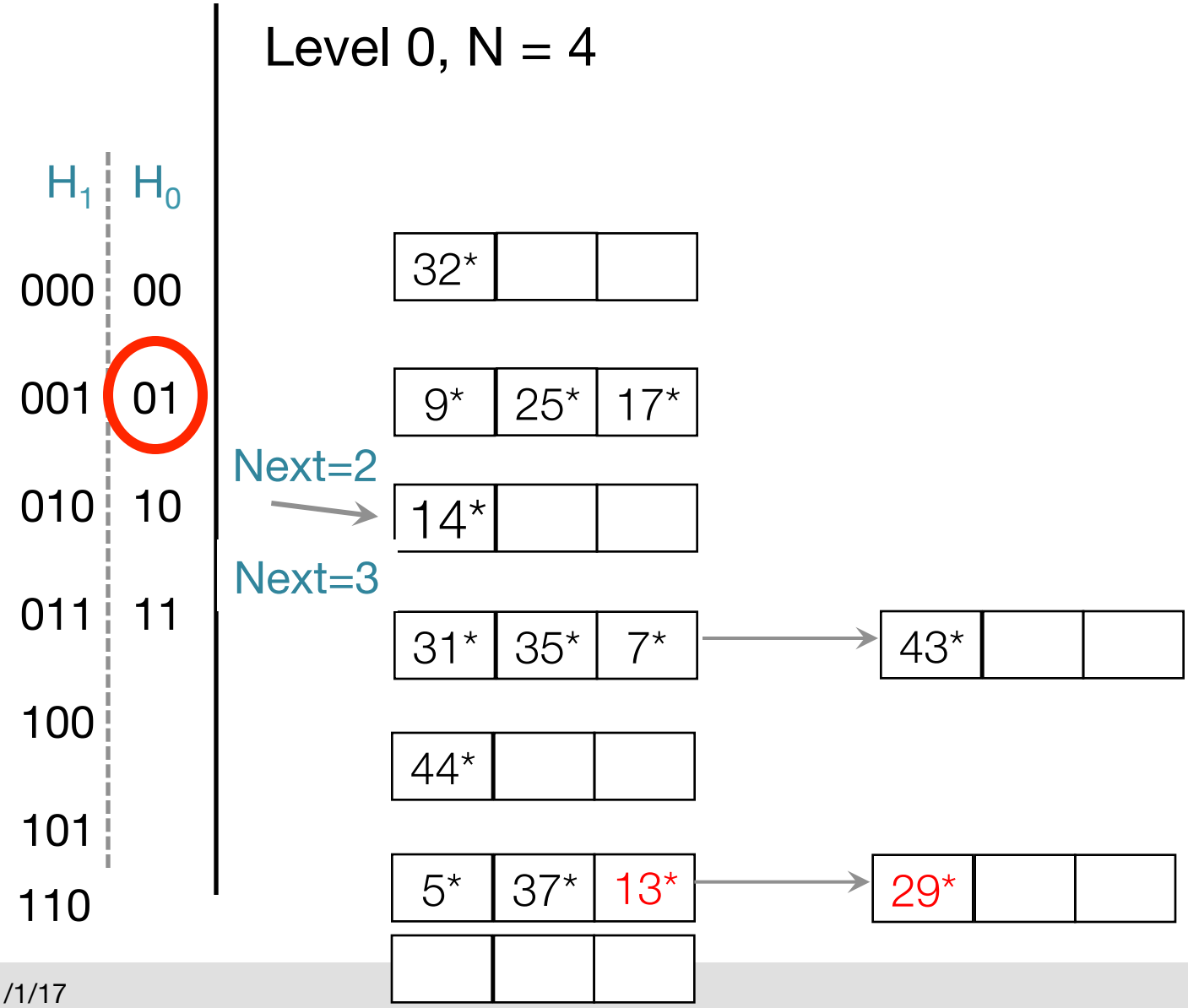
Quiz: Insert 13=$(1101)_2$ & 29=$(11101)_2$



Quiz: Insert 13=$(1101)_2$ & 29=$(11101)_2$



Quiz: Insert 13=$(1101)_2$ & 29=$(11101)_2$



Summary

- Discussed 3 kinds of hash-based indexes
- Static Hashing can lead to long overflow chains
- Extendible 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 Extensible Hashing

Optional Exercises

11.1, 11.3, 11.7, 11.9