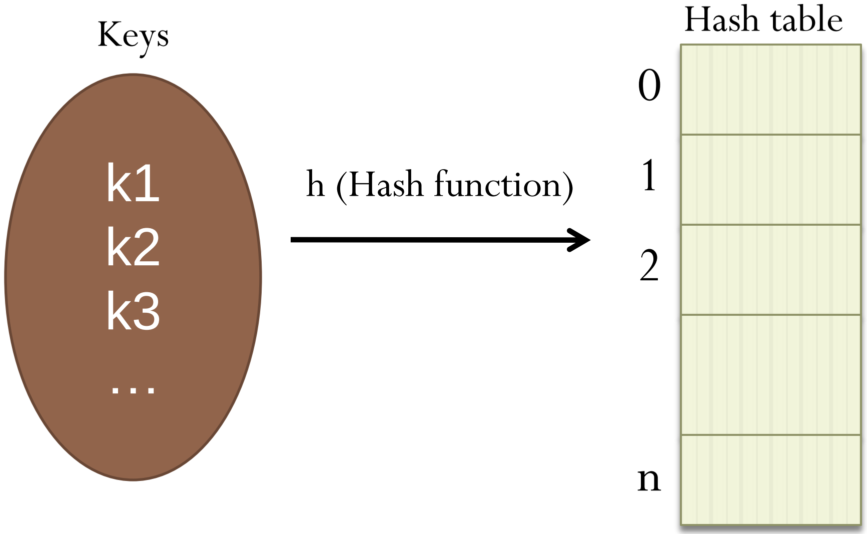
# Dynamic Hashing

# Dynamic Hashing

- Also called as Extendible hashing.
- Motivation
  - Limitations of static hashing
    - When the table is to be full, overflows increase. As overflows increase, the overall performance decreases.
    - We cannot just copy entries from smaller table into a corresponding buckets of a bigger table.
    - The use of memory space is not flexible.

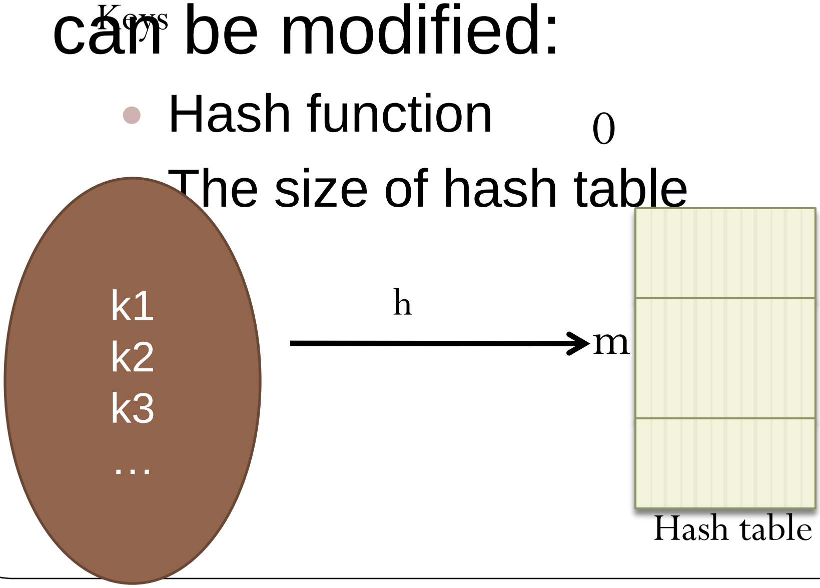


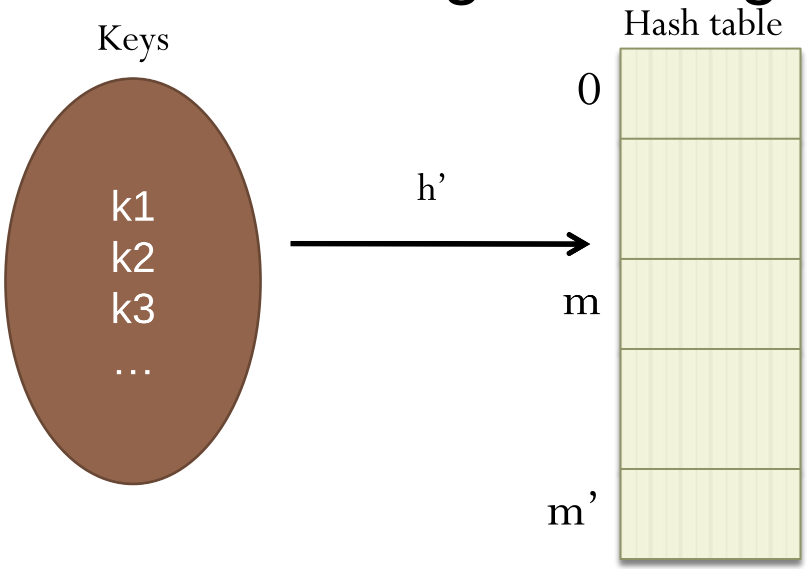
# Properties of Dynamic Hashing

- Allow the size of dictionary to grow and shrink.
- The size of the hash table **d** is depends on the no. of bits **r** of h(x) used as index.

$$\mathbf{D= 2^r}$$

- The size of hash table can be changed *dynamically*.
- The term “dynamically” implies the following two things can be modified:
  - Hash function
  - The size of hash table



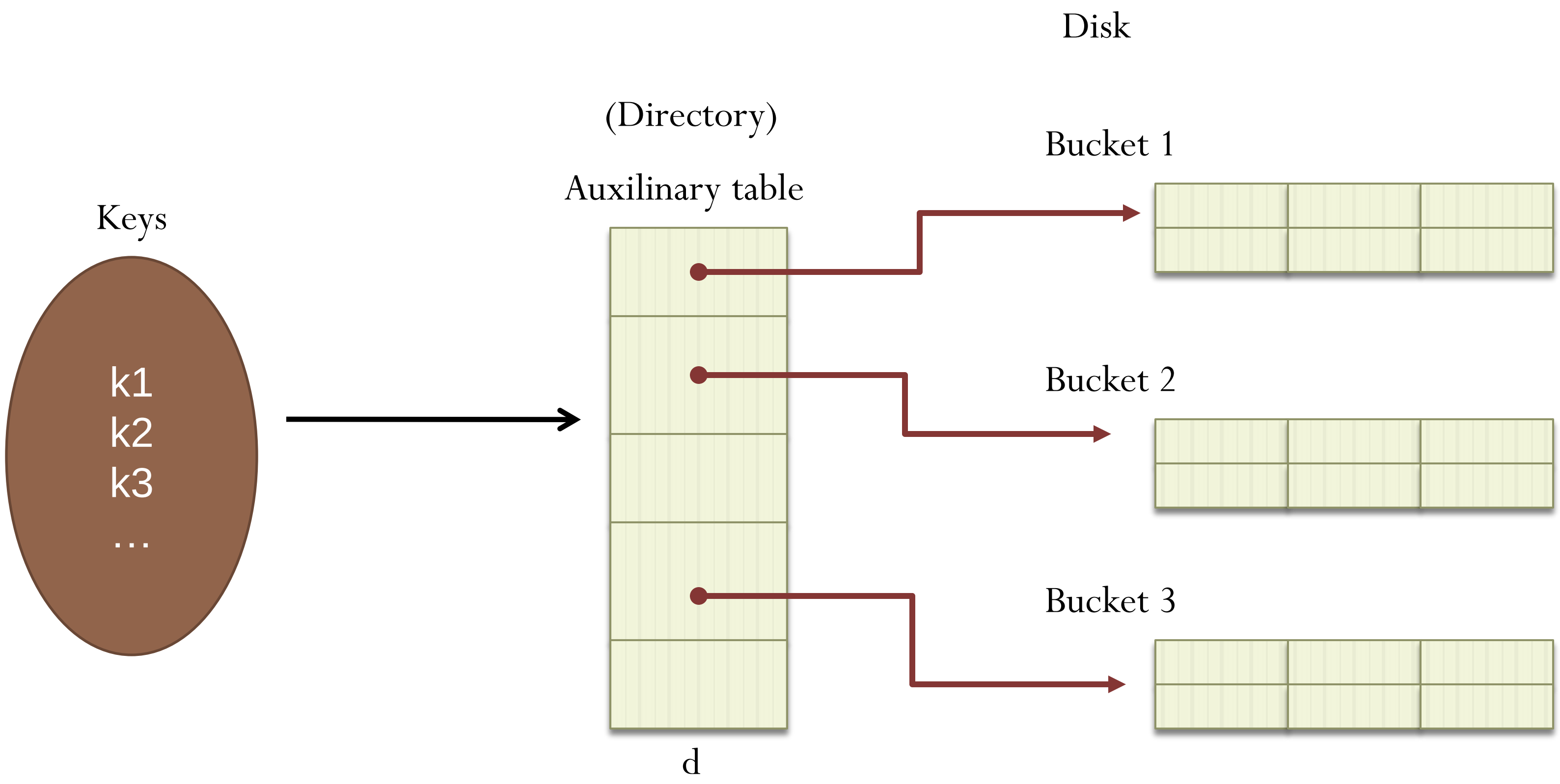


# Two Types

1. **Dynamic Hashing using Directories**
2. **Directory less Dynamic Hashing (Linear Dynamic Hashing)**

# Dynamic Hashing Using Directories

- Use an directory table to record the pointer of each bucket.



# Dynamic Hashing Using Directories

- Define the hash function h(k) transforms k into 6-bit binary integer.
- For example:

| k | h(k) |
|---|---|
| A0 | 100 000 |
| A1 | 100 001 |
| B0 | 101 000 |
| B1 | 101 001 |
| C1 | 110 001 |
| C2 | 110 010 |
| C3 | 110 011 |
| C5 | 110 101 |

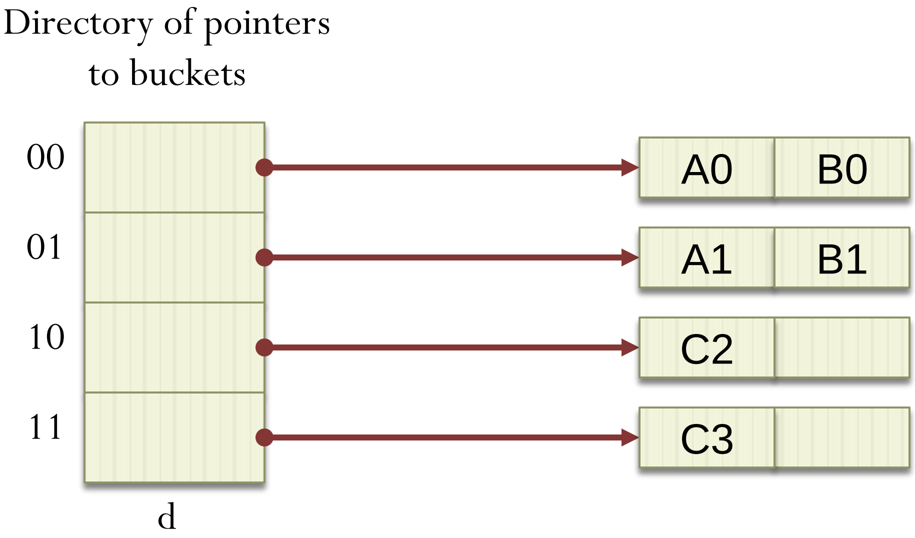
# Dynamic Hashing Using Directories

- The size of d is $2^r$, where r is the number of bits used to identify all h(x).
  - Initially, Let r = 2. Thus, the size of d = $2^2$ = 4.
- Suppose h(k, p) is defined as the p least significant bits in h(k), where p is also called dictionary depth.
- E. g.
  - h(C5) = 110 101
  - h(C5, 2) = 01
  - h(C5, 3) = 101

# Process to Expand the Directory

- Consider the following keys have been already stored. The least r is 2 to differentiate all the input keys.

| k | h(k) |
|---|---|
| A0 | 100 000 |
| A1 | 100 001 |
| B0 | 101 000 |
| B1 | 101 001 |
| C2 | 110 010 |
| C3 | 110 011 |

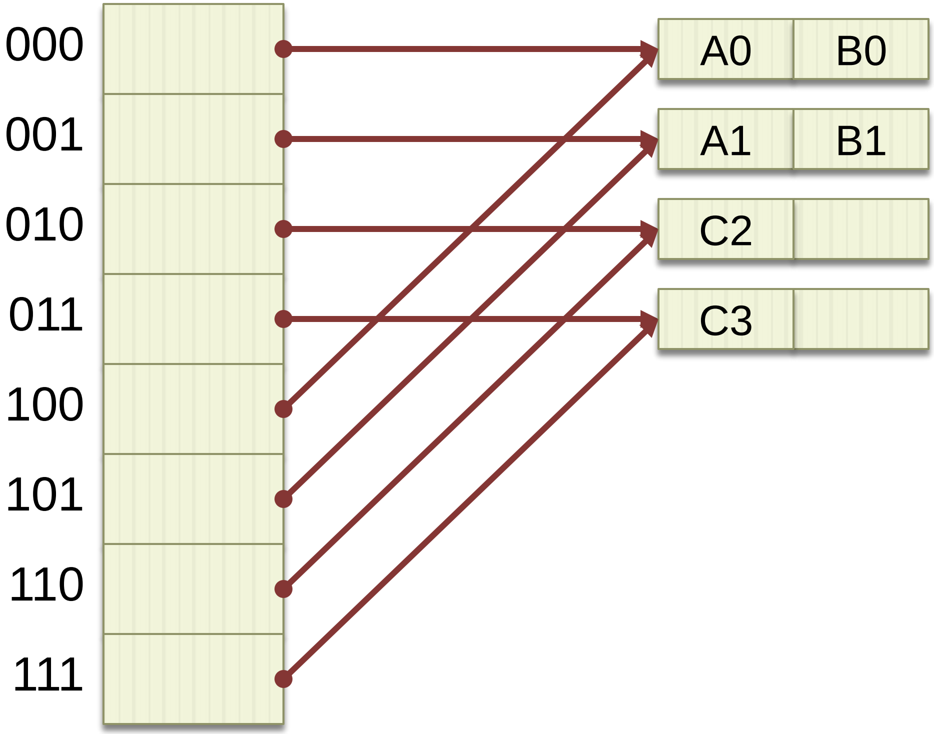


# When C5 (110101) is to enter

1. Since r=2 and h(C5, 2) = 01, follow the pointer of d[01].
2. A1 and B1 have been at d[01]. Bucket overflows.
   - Find the least u such that h(C5, u) is not the same with some keys in h(C5, 2) (01) bucket.
     - In this case, u = 3.
     - Step 2-1

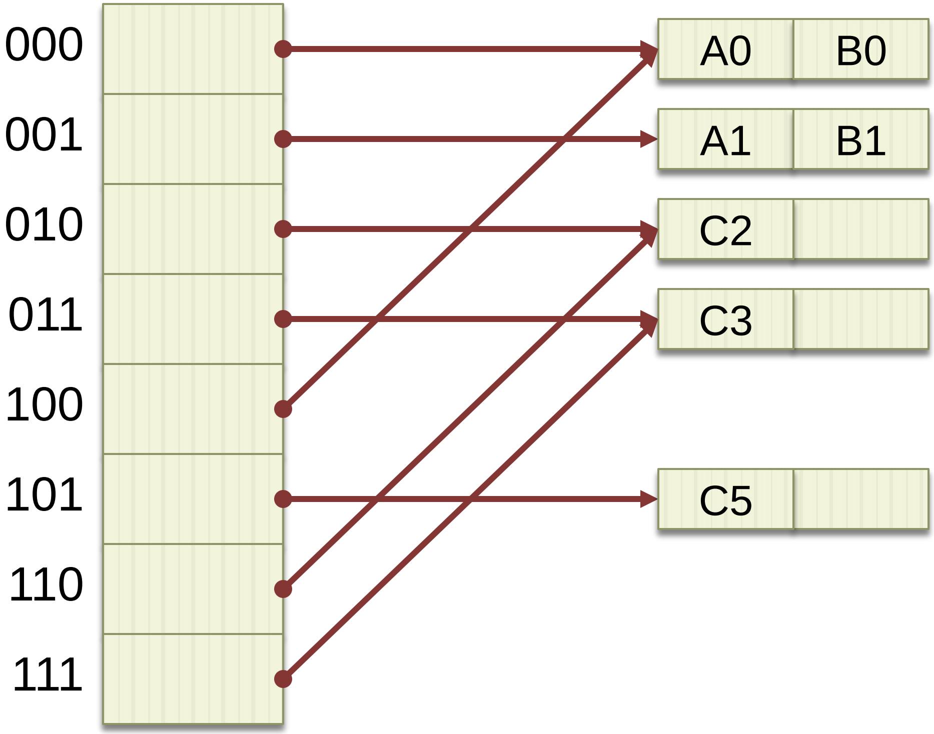
       Since u > r, expand the size of d to $2^u$ and duplicate the pointers to the new half (why?).

# When C5 (110101) is to enter



# When C5 (110101) is to enter

- Step 2-2
  - Rehash identifiers 01 (A1 and B1) and C5 using new hash function h(k, u).

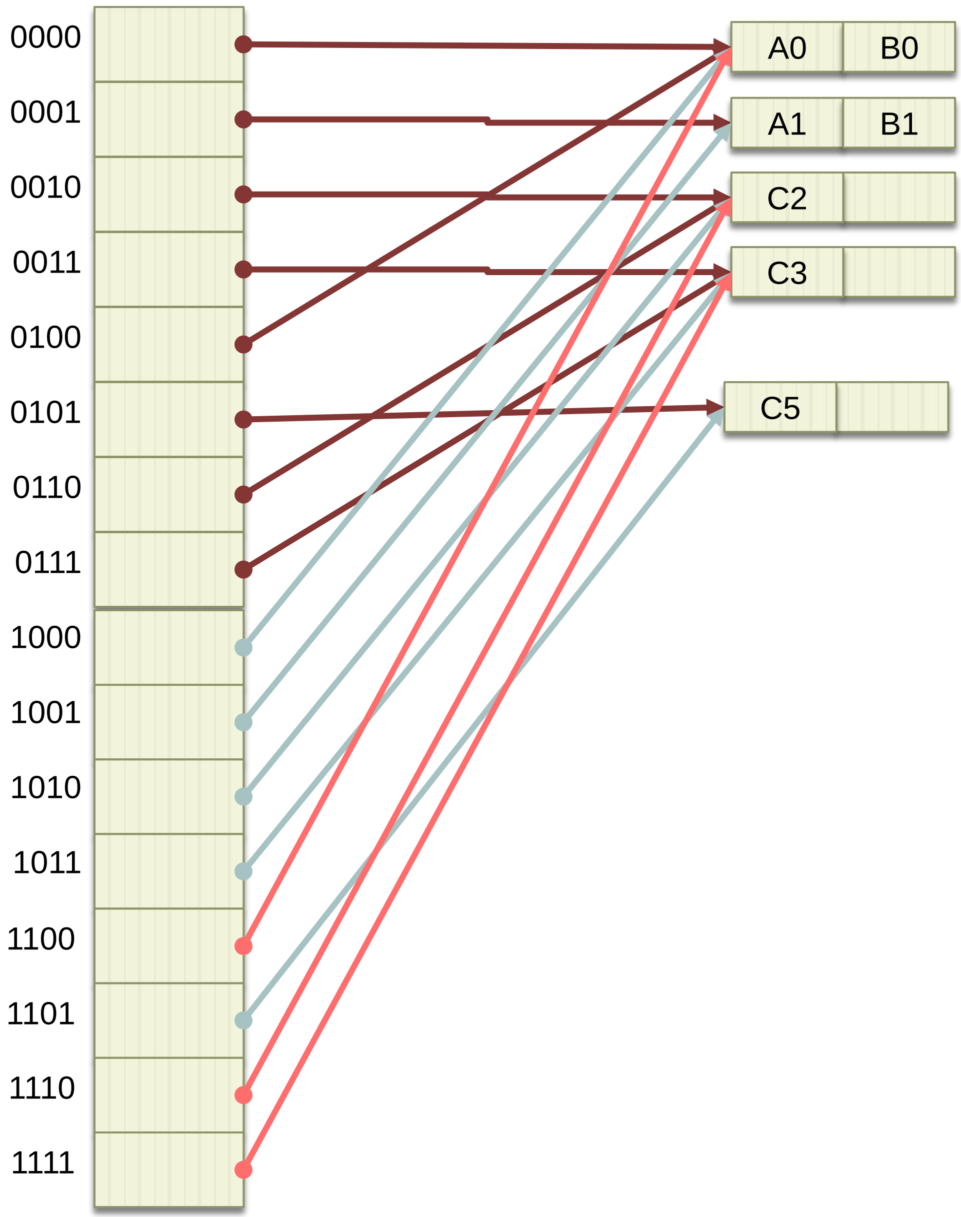


- Step 2-3
  - Let r = u = 3.

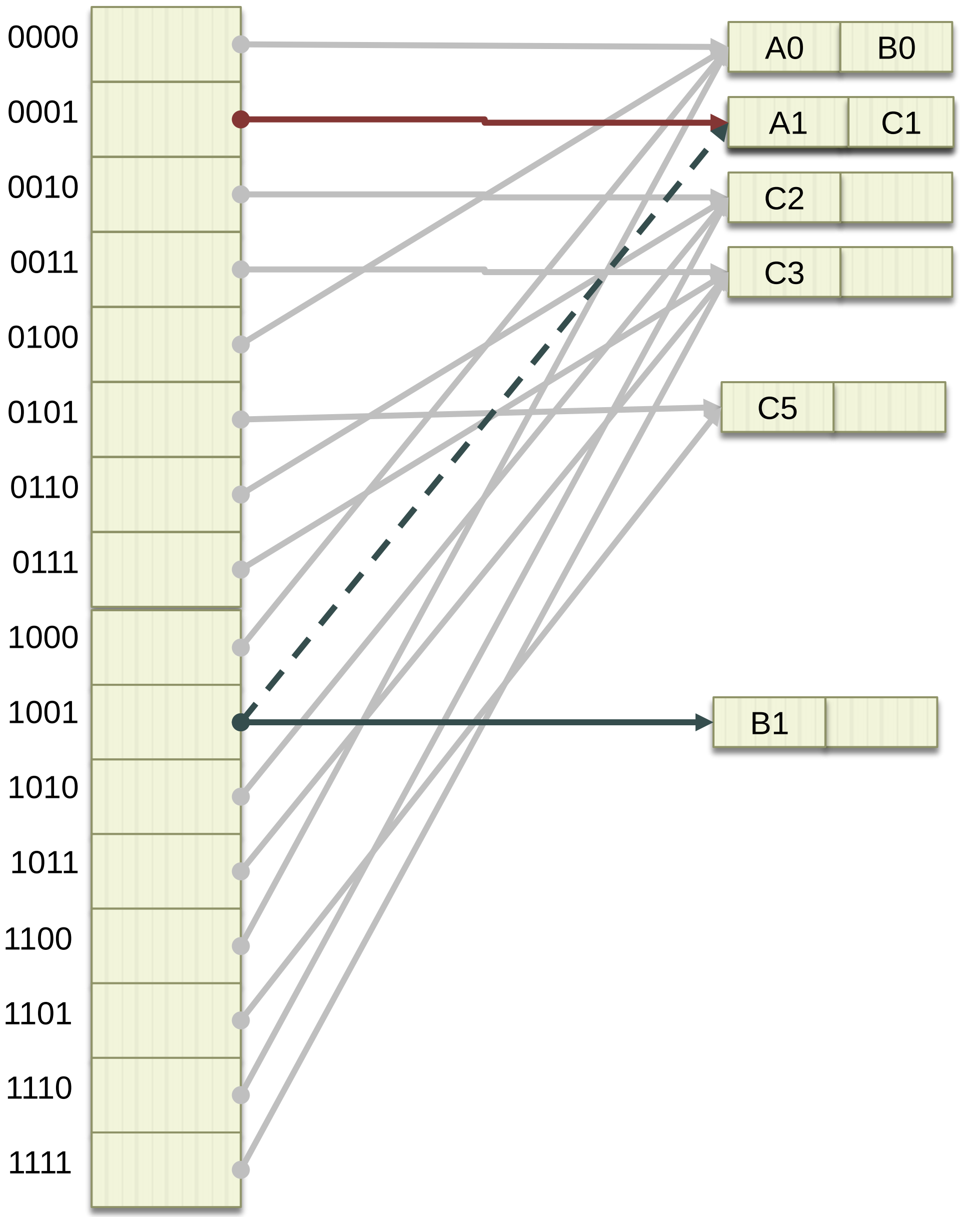
# When C1 (110001) is to enter

1. Since r=3 and h(C1, 3) = 001, follow the pointer of d[001].
2. A1 and B1 have been at d[001]. Bucket overflows.
   - Find the least u such that h(C1, u) is not the same with some keys in h(C1, 3) (001) bucket.
     - In this case, u = 4.
     - Step 2-1

       Since u > r, expand the size of d to $2^u$ and duplicate the pointers to the new half.


0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
A0
B0
A1
B1
C2
C3
C5

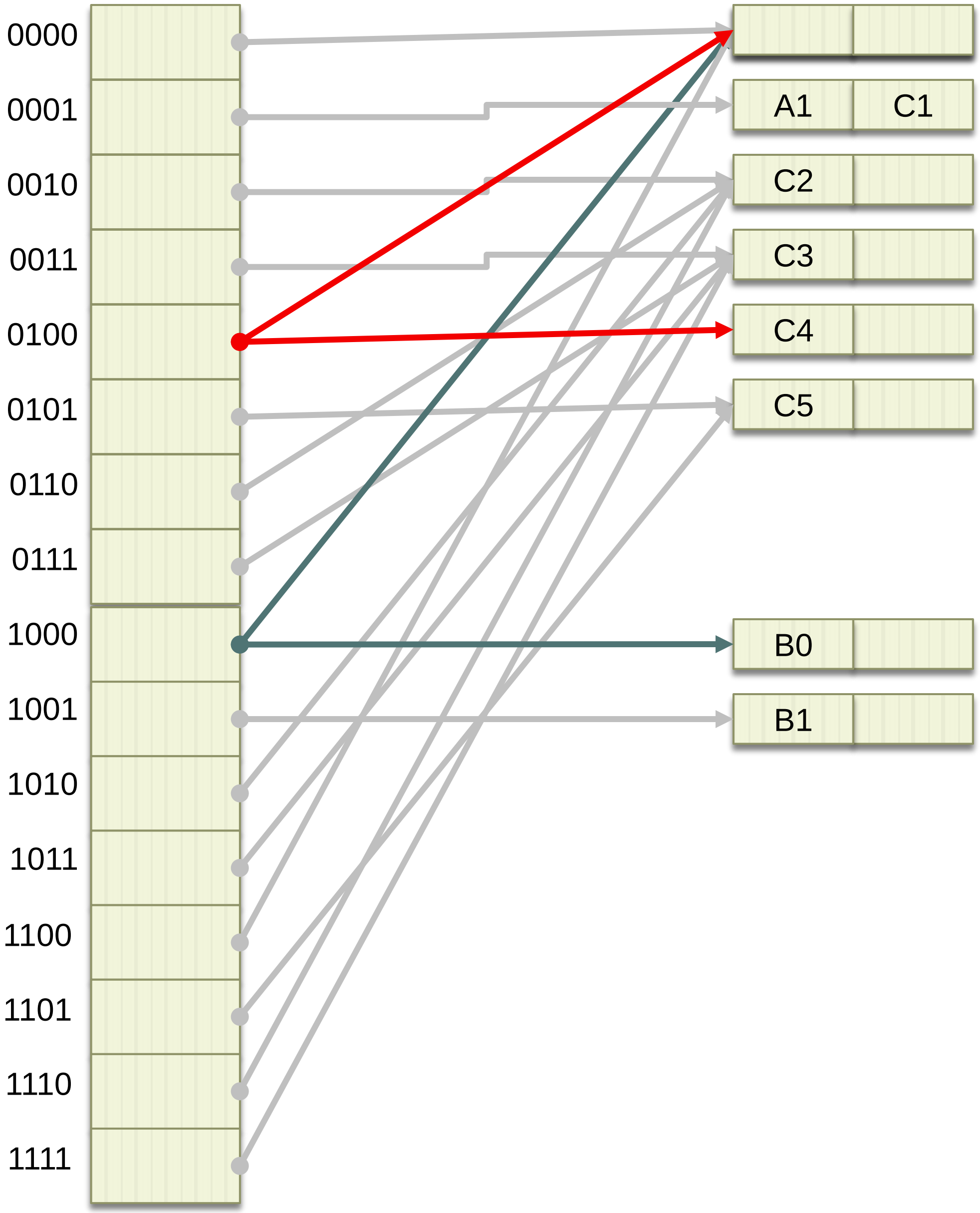
- Step 2-2
  - Rehash identifiers 001 (A1 and B1) and C1 using new hash function h(k, u).
- Step 2-3
  - Let r = u = 4.



# When C4 (110100) is to enter

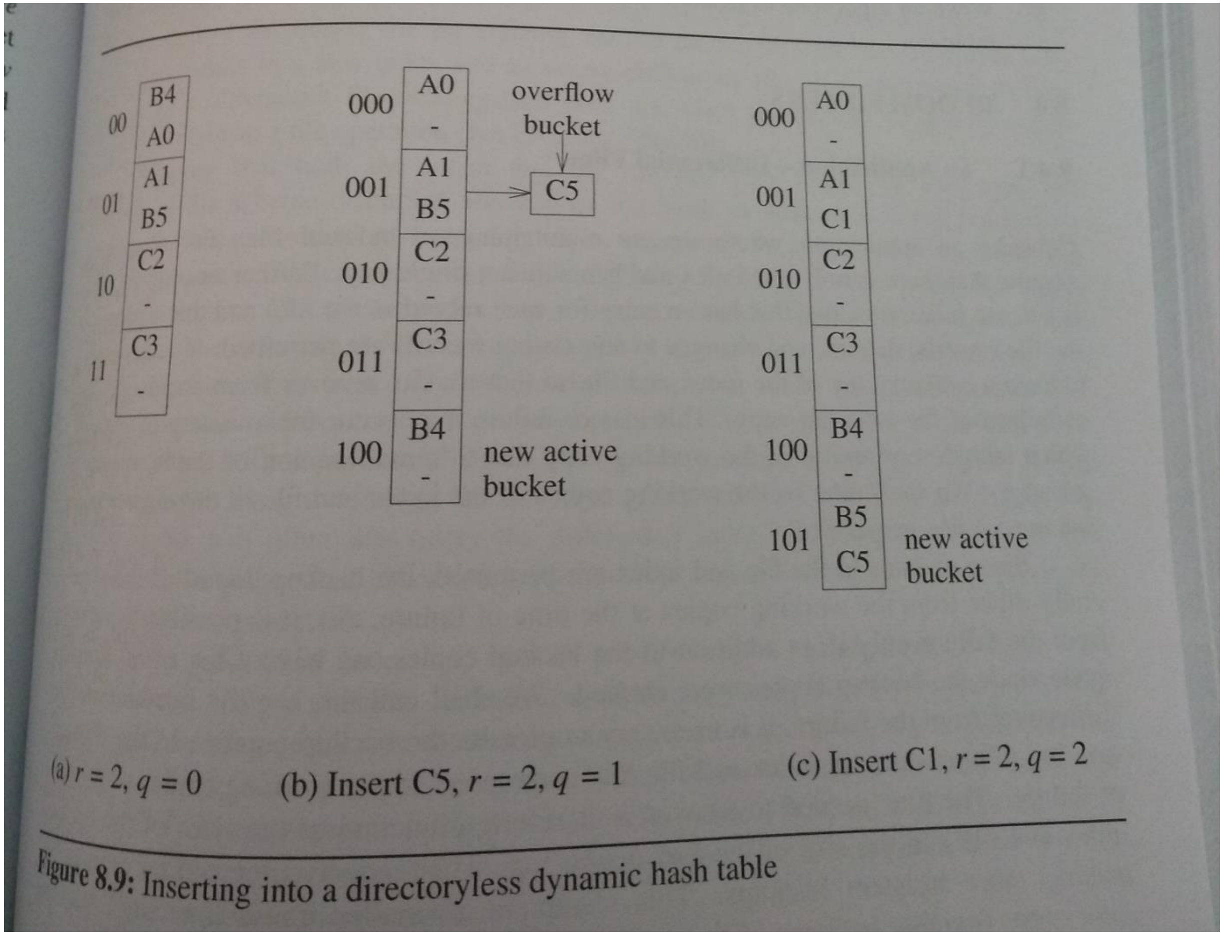
1. Since r=4 and h(C4, 4) = 0100, follow the pointer of d[0100].
2. A0 (100000) and B0 ((101000)) have been at d[0100]. Bucket overflows.
   - Find the least u such that h(C1, u) is not the same with some keys in h(C1, 4) (0100) bucket.
     - In this case, u = 3.
     - Step 2-1

       Since u = 3 < r = 4, d is not required to expand its size.


0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
A1
C1
C2
C3
C4
C5
B0
B1

# Directory less dynamic hashing

- Also called as Linear dynamic hashing.



(a) $r = 2, q = 0$ (b) Insert C5, $r = 2, q = 1$ (c) Insert C1, $r = 2, q = 2$

**Figure 8.9:** Inserting into a directoryless dynamic hash table

# Advantages

- Only doubling directory rather than the whole hash table used in static hashing.
- Only rehash the entries in the buckets that overflows.