Data Structure and Algorithms

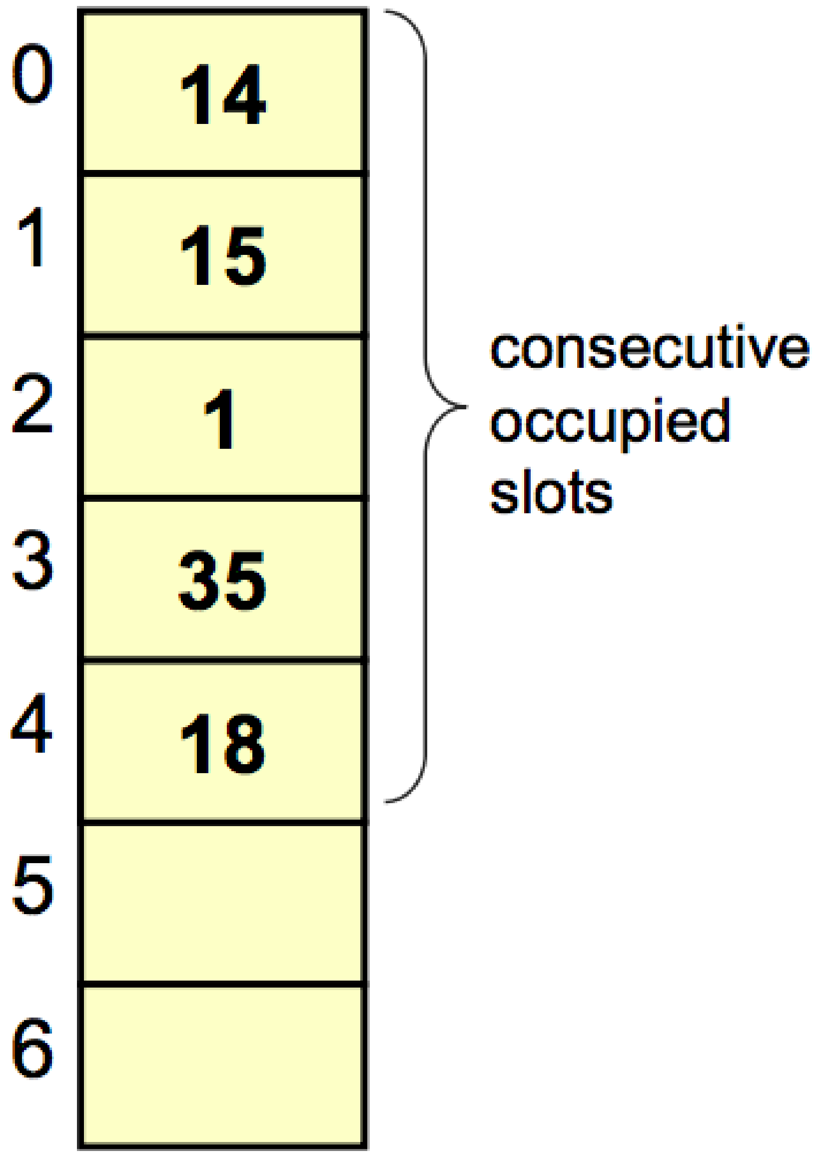
Session-32

Dr. Subhra Rani Patra
SCOPE, VIT Chennai

Clustering Problem

- As long as table is big enough, a free cell can always be found, but the time to do so can get quite large.
- Worse, even if the table is relatively empty, blocks of occupied cells start forming.
- This effect is known as *primary clustering*.
- Any key that hashes into the cluster will require several attempts to resolve the collision, and then it will add to the cluster.

- A **cluster** is a collection of consecutive occupied slots
- A cluster that covers the home address of a key is called the **primary cluster** of the key
- Linear probing can create large primary clusters that will increase the running time of find/insert/delete operations



Random probing

$$i = (i + m) \% h + 1$$

i = index

m = prime number (2, 3, 5, 7 ...)

h = size of Hash table

$i = (i + m) \% h + 1$

i = index

m = prim number (2, 3, 5, 7 ...)

h = size of Hash table

$i \leftarrow (key \% 11) + 1$

key = 23

$i = (23 \% 11) + 1$

$= 1 + 1 = 2$

1	
2	23
3	
4	
5	
6	
7	
8	
9	
10	
11	

key = 45

$$i = 45 \% 11 + 1$$
$$= 1 + 1$$
$$= 2$$

1	
2	23
3	
4	
5	
6	
7	
8	
9	
10	
11	

key = 45

$i = 45 \% 11 + 1$

$= 1 + 1$

$= 2$

$i = (i + m) \% h + 1$

$m = 5$

$i = (2 + 5) \% 11 + 1$

$= 7 \% 11 + 1$

$= 8$

$i =$

1	
2	23
3	
4	
5	
6	
7	
8	
9	
10	
11	

$key = 45$

$i = 45 \% 11 + 1$

$= 1 + 1$

$= 2$

$i = (i + m) \% n + 1$

$m = 5$

$i = (2 + 5) \% 11 + 1$

$= 7 \% 11 + 1$

$= 8$

$i = (8 + 5) \% 11 + 1$

$= (13) \% 11 + 1$

$= 2 + 1 = 3$

1	
2	23
3	
4	
5	
6	
7	
8	
9	
10	
11	

Key = 45

1	
2	23
3	
4	
5	
6	
7	
8	
9	
10	
11	

$i = 45 \% 11 + 1$

$= 1 + 1$

$= 2$

$i = (i + m) \% h + 1$

$m = 5$

$i = (2 + 5) \% 11 + 1$

$= 7 \% 11 + 1$

$= 8$

$i = (8 + 5) \% 11 + 1$

$= (13) \% 11 + 1$

$= 2 + 1 = 3$

$i = (3 + 5) \% 11 + 1$

$= 8 + 1 = 9$

$i = (9 + 5) \% 11 + 1$

$= 14 \% 11 + 1$

$= 4$

$i = 9 \% 11 + 1$

$= 10$

$i = 15 \% 11 + 1$

$= 4 + 1$

$i = (5 + 5) \% 11 + 1$

$= 10 + 1$

$i = 16 \% 11 + 1 = 5$

8, 3, 9, 4,
10, 5, 11
6, 1, 7, 2

$i = 11 \% 11 + 1$

$= 1$

$i = 6 \% 11 + 1$

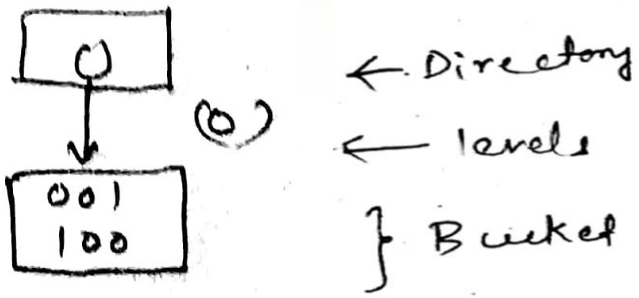
$= 7$

$i = 12 \% 11 + 1$

$= 2$

Extendible Hashing

- Handles large amount of data.
- The data to be placed in the hash table is by extracting certain no. of bits.



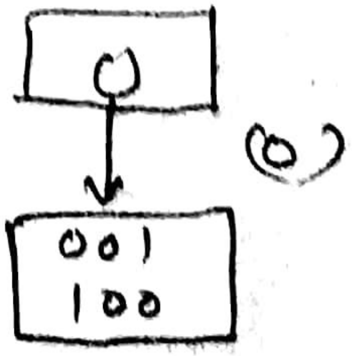
- The bucket can hold the data of its global depth.
- If data in bucket is more than global depth, then split the bucket and double the directory

Insert 1, 4, 5, 7, 8, 10 and assume each page can hold 2 data entries (2 is the depth).

step I: Insert 1, 4

1 = 001

4 = 100



Extendible Hashing

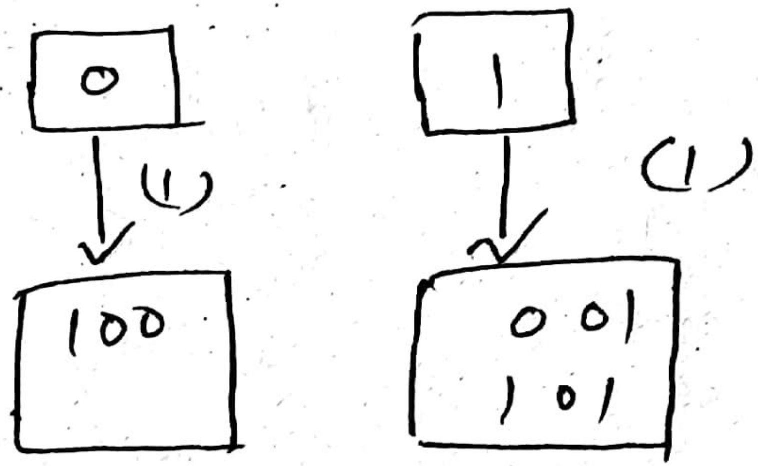
→ Insert 5

Bucket is full. Hence double the directory

1 = 001

4 = 100

5 = 101



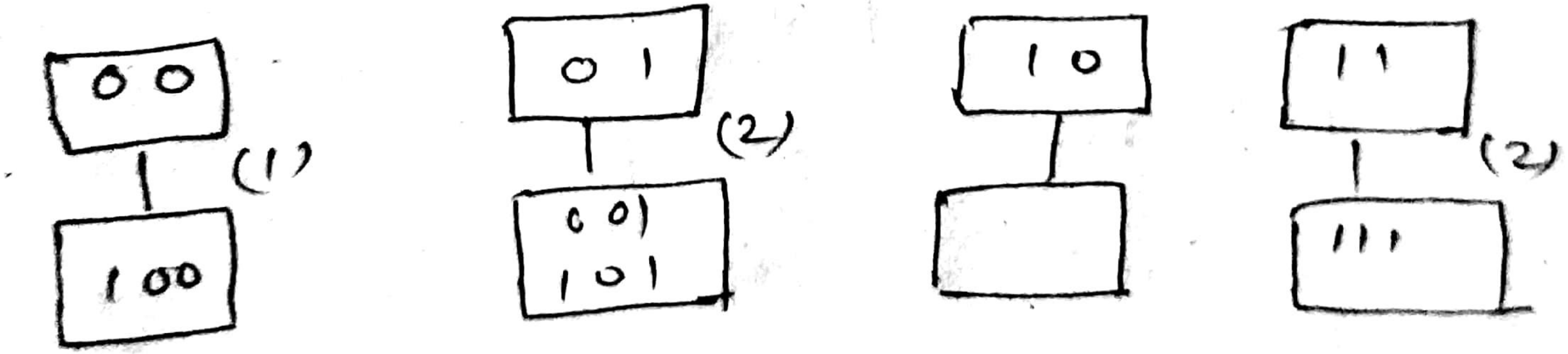
We will examine last bit of data and insert the data in bucket.

Step II : Insert 7

7 = 111

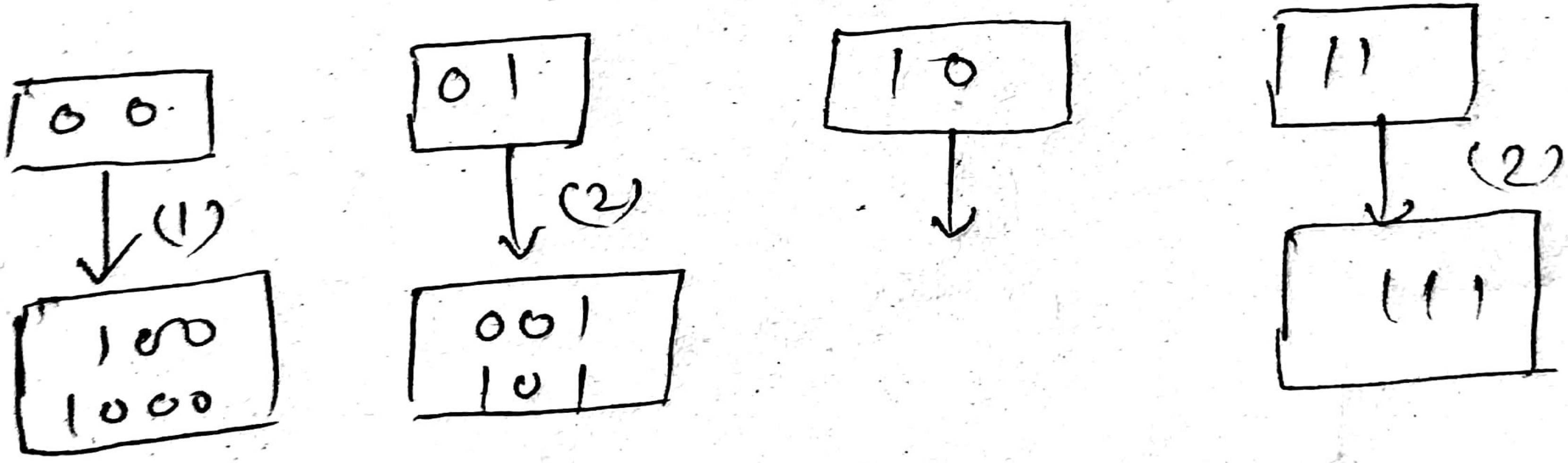
Bucket is full. Hence double the directory and split the bucket

After insertion of 7, consider last two bits.



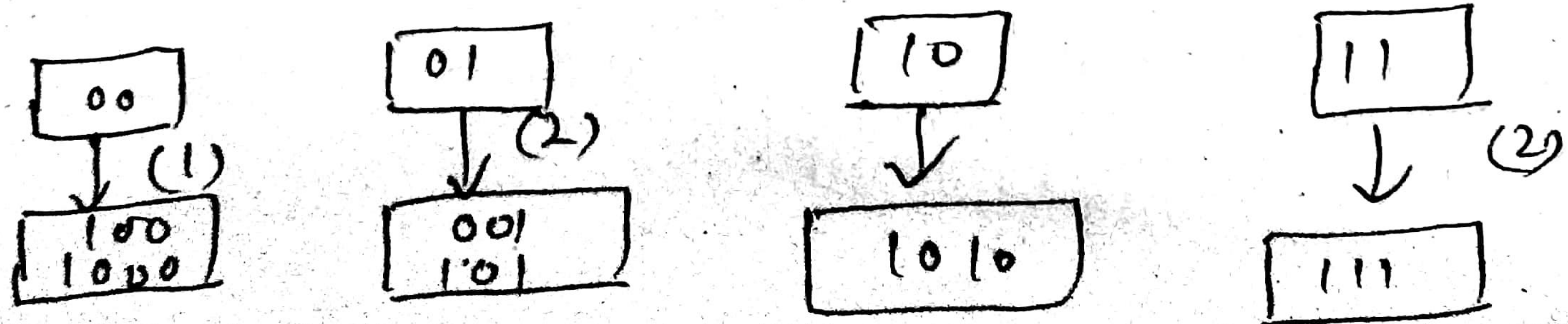
step 3 : Insert 8

8 = 1000



setp 4 : Insert 10

10 = 1010



Telephonic directory

Phone Book

Design a data structure to store your contacts: names of people along with their phone numbers. The data structure should be able to do the following quickly:

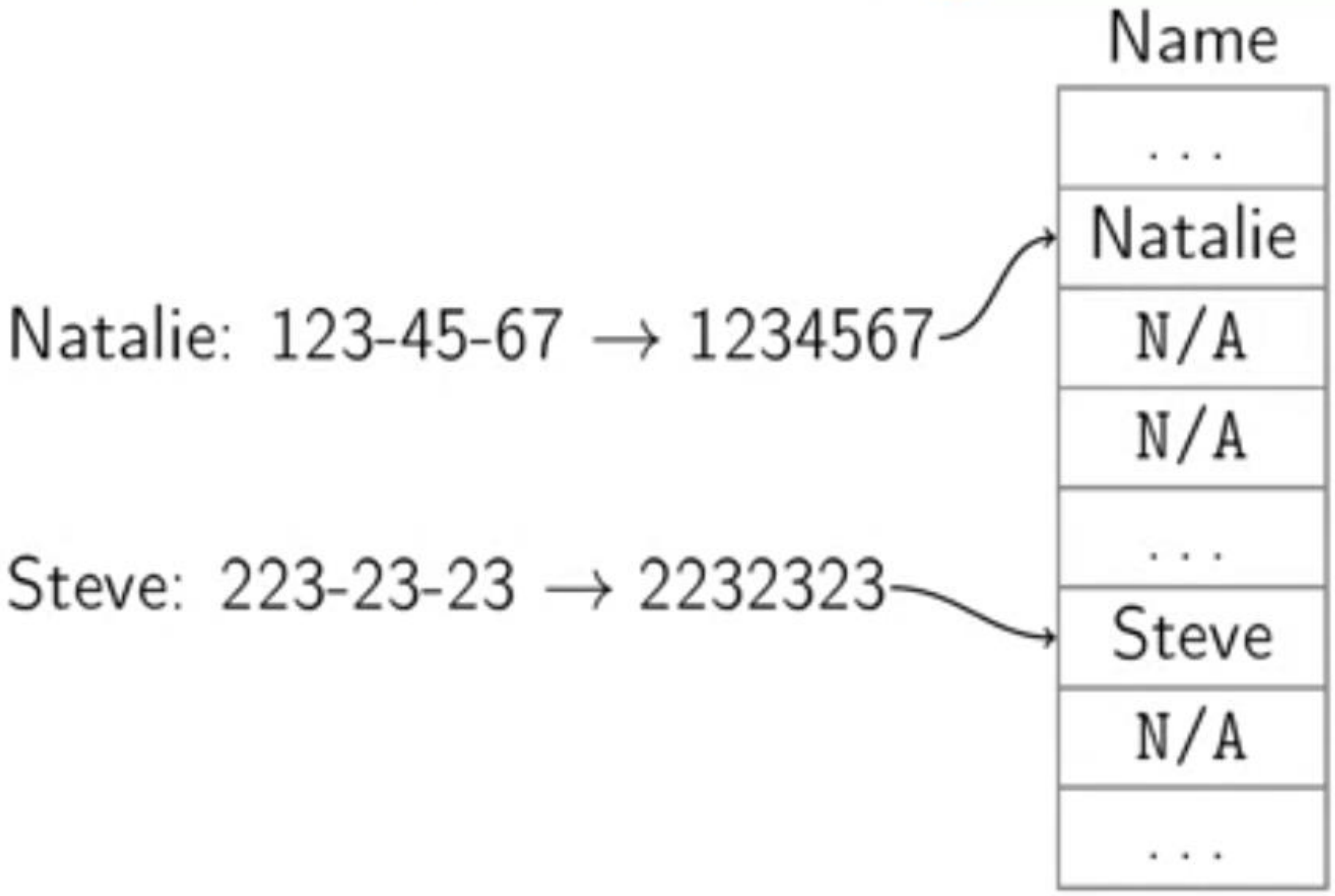
- Add and delete contacts,
- Lookup the phone number by name,
- Determine who is calling given their phone number.

- We need two Maps:
  (phone number $\to$ name) and
  (name $\to$ phone number)
- Implement these Maps as hash tables
- First, we will focus on the Map from phone numbers to names

Direct Addressing

- $\mathtt{int}(123\text{-}45\text{-}67) = 1234567$
- Create array *Name* of size 10^L where L is the maximum allowed phone number length
- Store the name corresponding to phone number P in $Name[\mathtt{int}(P)]$
- If no contact with phone number P, $Name[\mathtt{int}(P)] = \mathtt{N/A}$

Direct Addressing



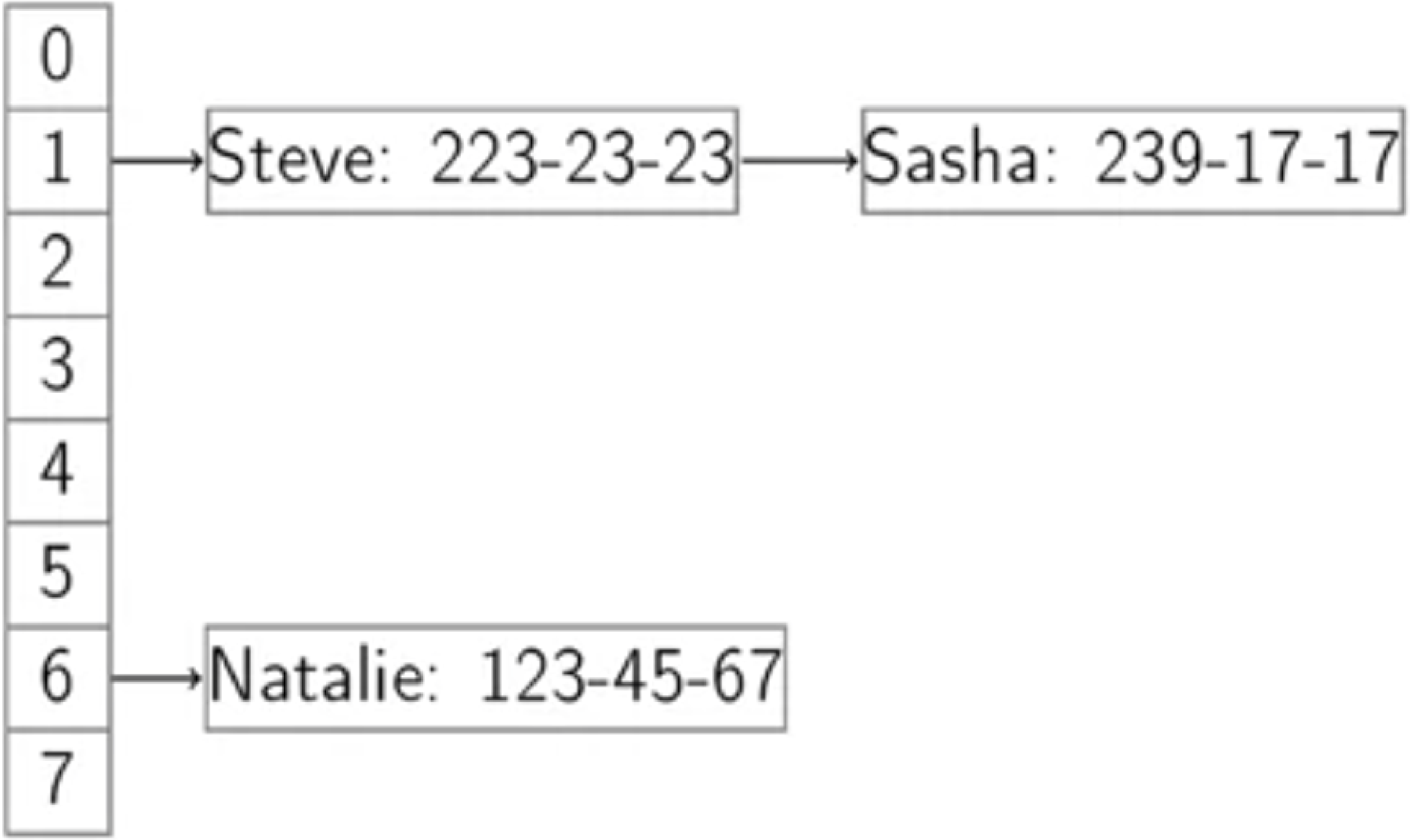
Direct Addressing

- Operations run in $O(1)$
- Memory usage: $O(10^L)$, where L is the maximum length of a phone number
- Problematic with international numbers of length 12 and more: we will need 10^{12} bytes $=$ 1TB to store one person's phone book — this won't fit in anyone's phone!

Chaining

- Select hash function h with cardinality m
- Create array *Name* of size m
- Store chains in each cell of the array *Name*
- Chain $Name[h(\mathtt{int}(P))]$ contains the name for phone number P

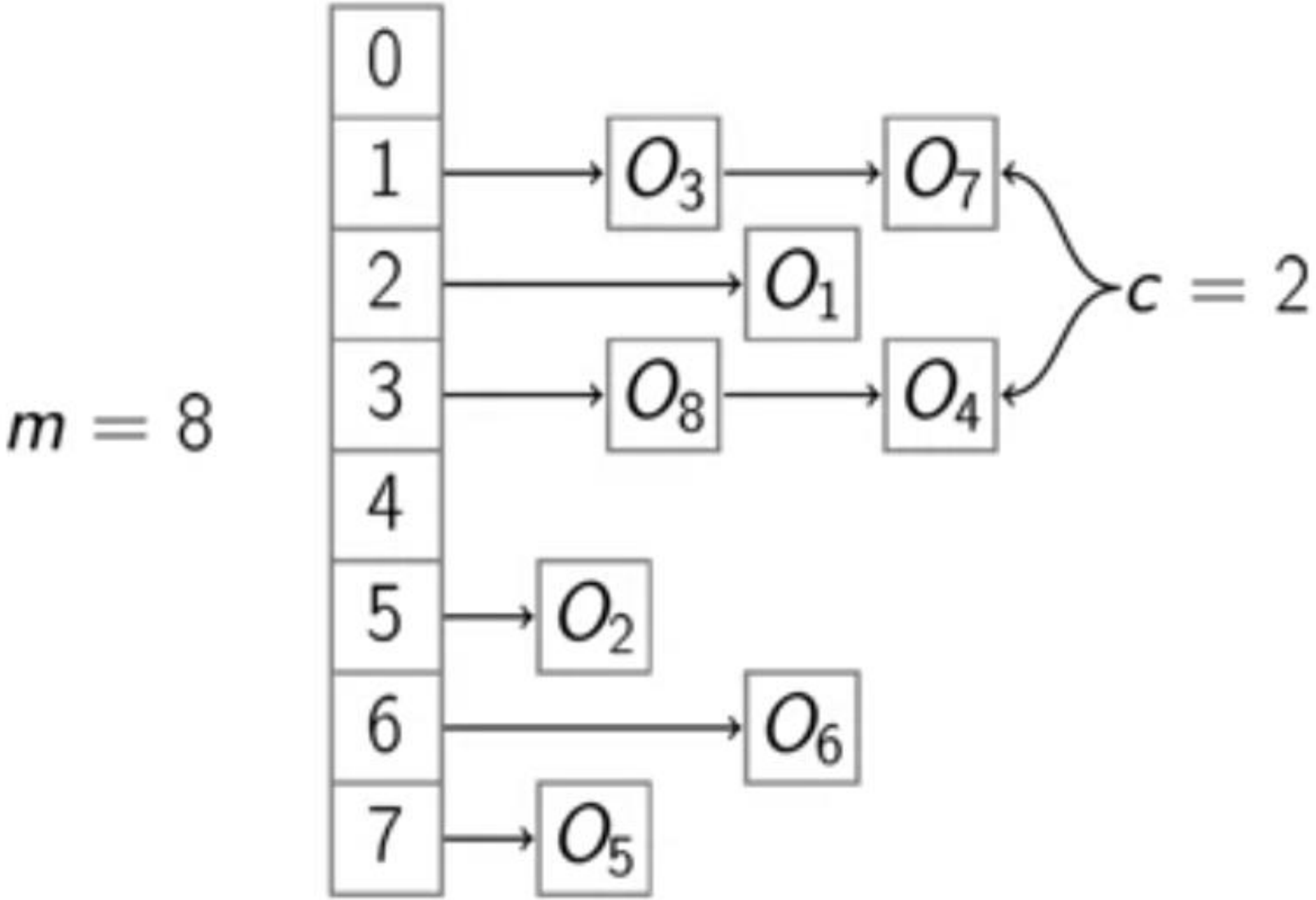
Chaining



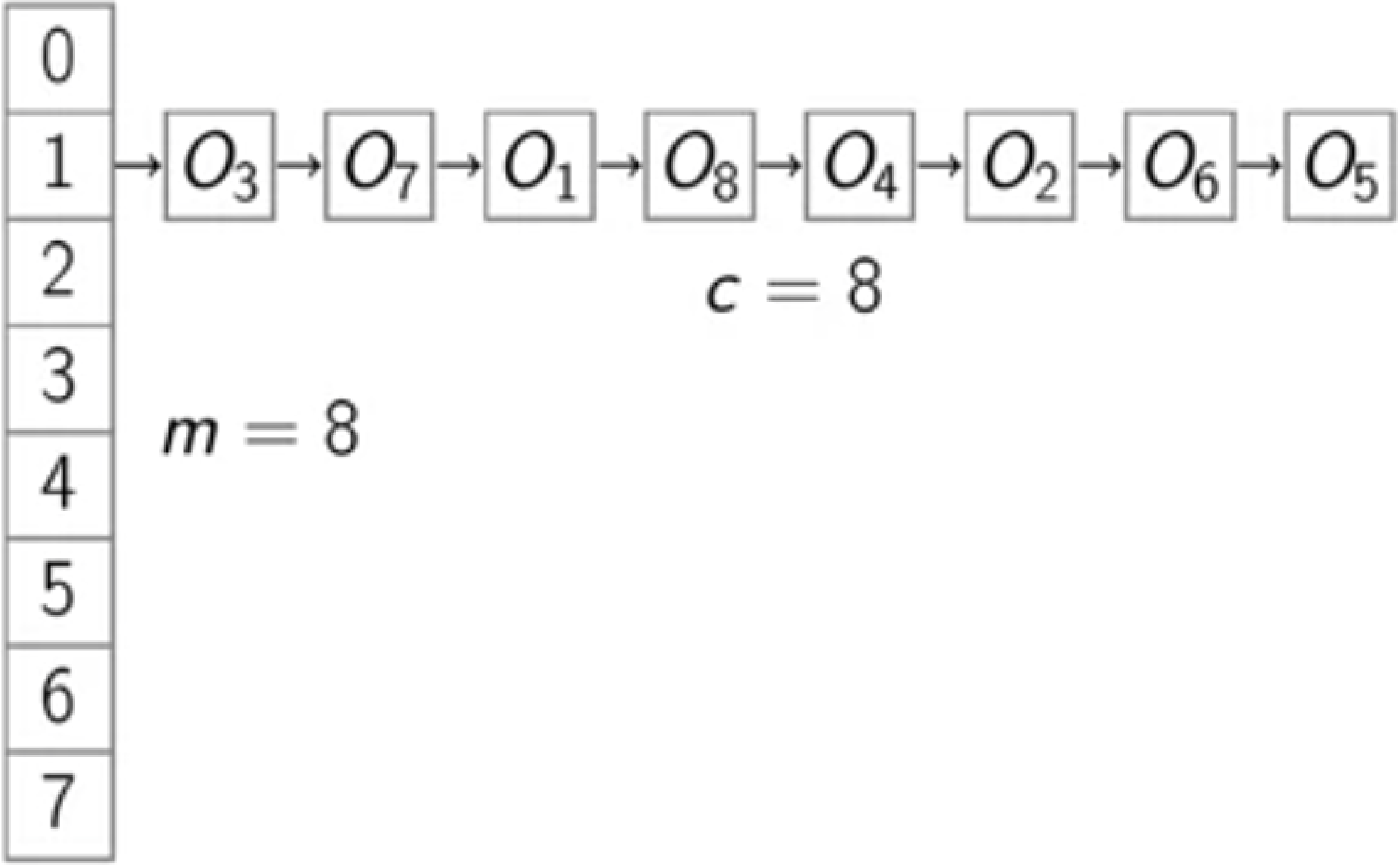
Parameters

- n phone numbers stored
- m — cardinality of the hash function
- c — length of the longest chain
- $O(n+m)$ memory is used
- $\alpha = \frac{n}{m}$ is called load factor
- Operations run in time $O(c+1)$
- You want small m and c!

Good Example



Bad Example



First Digits

- For the map from phone numbers to names, select $m = 1000$
- Hash function: take first three digits
- $h(800\text{-}123\text{-}45\text{-}67) = 800$
- Problem: area code
- $h(425\text{-}234\text{-}55\text{-}67) =$
  $h(425\text{-}123\text{-}45\text{-}67) =$
  $h(425\text{-}223\text{-}23\text{-}23) = \cdots = 425$

$h(425\text{-}234\text{-}55\text{-}67) =$
$h(425\text{-}123\text{-}45\text{-}67) =$
$h(425\text{-}223\text{-}23\text{-}23) = \cdots = 425$

Last Digits

- Select $m = 1000$
- Hash function: take last three digits
- $h(800\text{-}123\text{-}45\text{-}67) = 567$
- Problem if many phone numbers end with three zeros

Random Value

- Select $m = 1000$
- Hash function: random number between 0 and 999
- Uniform distribution of hash values
- Different value when hash function called again — we won't be able to find anything!
- Hash function must be deterministic

Good Hash Functions

- Deterministic
- Fast to compute
- Distributes keys well into different cells
- Few collisions

Thank you