Hashing

Lecture Outline

- What is hashing?
- How to hash? – Direct Addressing
- What is collision?
- How to resolve collision?
  - Separate chaining
  - Linear probing
  - Quadratic probing
  - Double hashing
- Load factor
- Primary and secondary clustering

What is Hashing?

- An algorithm that uses a **function (hash function)** to map **large** data sets (**variable length**), called **keys**, to **smaller** data sets of a **fixed length**
- **Hash table (hash map):** A data structure that uses a hash function to efficiently map keys to values
- Very efficient search and retrieval
- Used in many computer software: e.g. associative arrays, database indexing, caches, and sets.

Complexity

Operations	Sorted Array
Insertion	O(*n*)
Deletion	O(*n*)
Retrieval	O(log *n*)

Complexity

Operations	Sorted Array	Balanced BST
Insertion	O(*n*)	O(log *n*)
Deletion	O(*n*)	O(log *n*)
Retrieval	O(log *n*)	O(log *n*)

Complexity

Operations	Sorted Array	Balanced BST	Hashing
Insertion	O(*n*)	O(log *n*)	**O(1) avg**
Deletion	O(*n*)	O(log *n*)	**O(1) avg**
Retrieval	O(log *n*)	O(log *n*)	**O(1) avg**

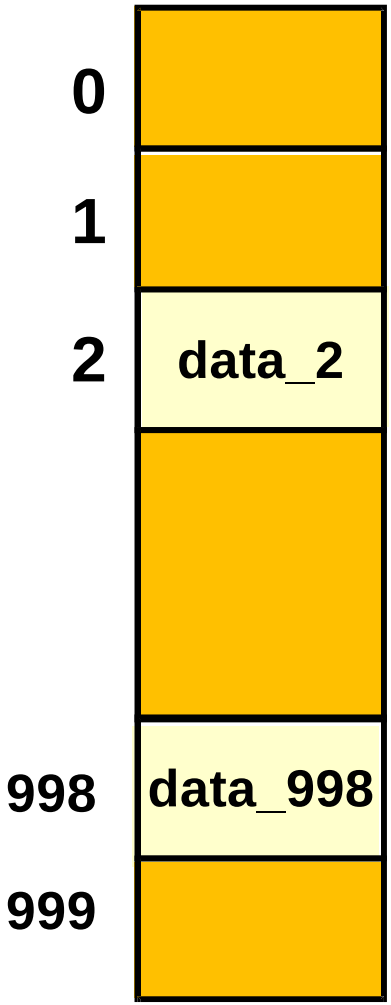
Constant time, on an average, for the above operations (terms and conditions apply…)

Direct Addressing

The easiest form

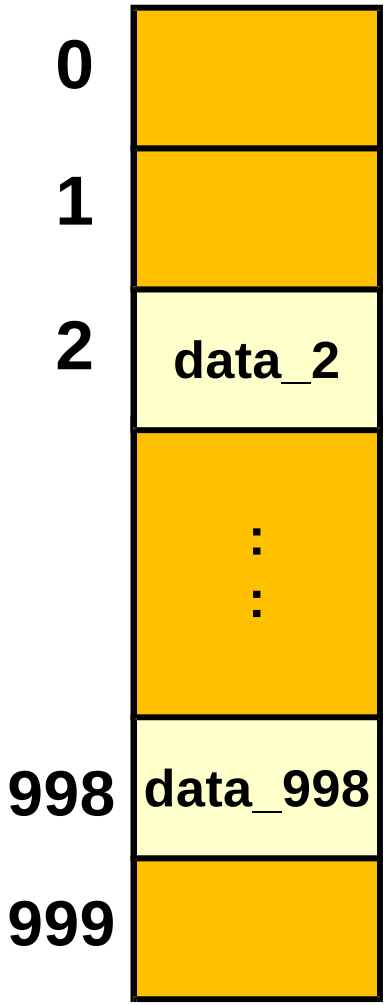
Example: Bus Services (1/2)

- **Operations**
  - **Retrieval**: find (NUM)
    - Find the bus route of bus service number = **NUM**
  - **Insertion**: insert (NUM)
    - Introduce a new bus service number = **NUM**
  - **Deletion**: delete (NUM)
    - Remove bus service number = **NUM**
- If bus numbers are integers **0 – 999**, we can use an **array with 1000 entries**



Example: Bus Services (2/2)

```
// a[] is an array (the table)

 insert(key, data)
     a[key] = data

 delete(key)
     a[key] = NULL

 find(key)
     return a[key]
```



Direct Addressing: Limitations

- Keys must be non-negative integer values
- Range of keys needs to be small
- Keys must be dense, i.e. not many gaps in key values
- How to overcome these restrictions?

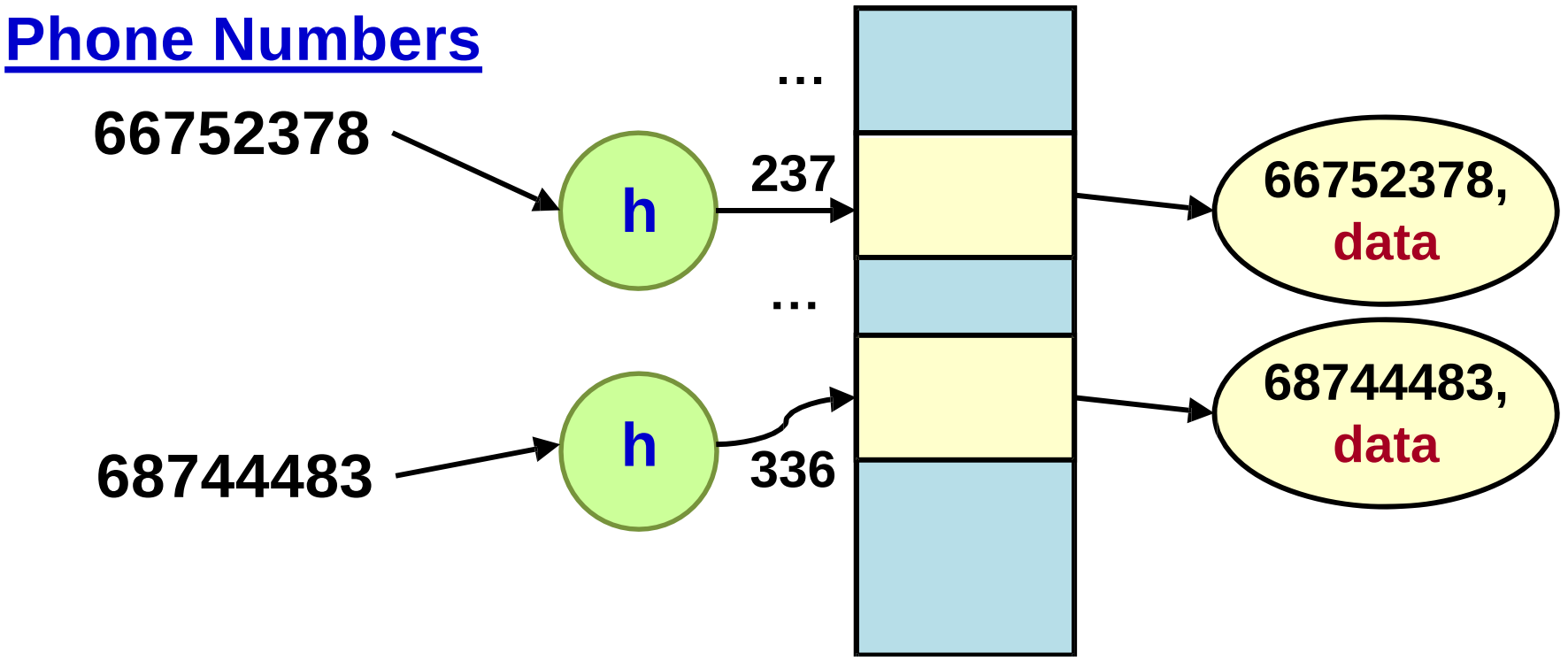
Hash Table

The true form of hashing

Hash Table: Key Ideas

- Map large integers to smaller integers
- Map non-integer keys to integers

Hash Table: Example



- **h** is a **hash function, h(x) = x%997; (x = phone number)**
- **Note**: we must store the key values
- **Why?**

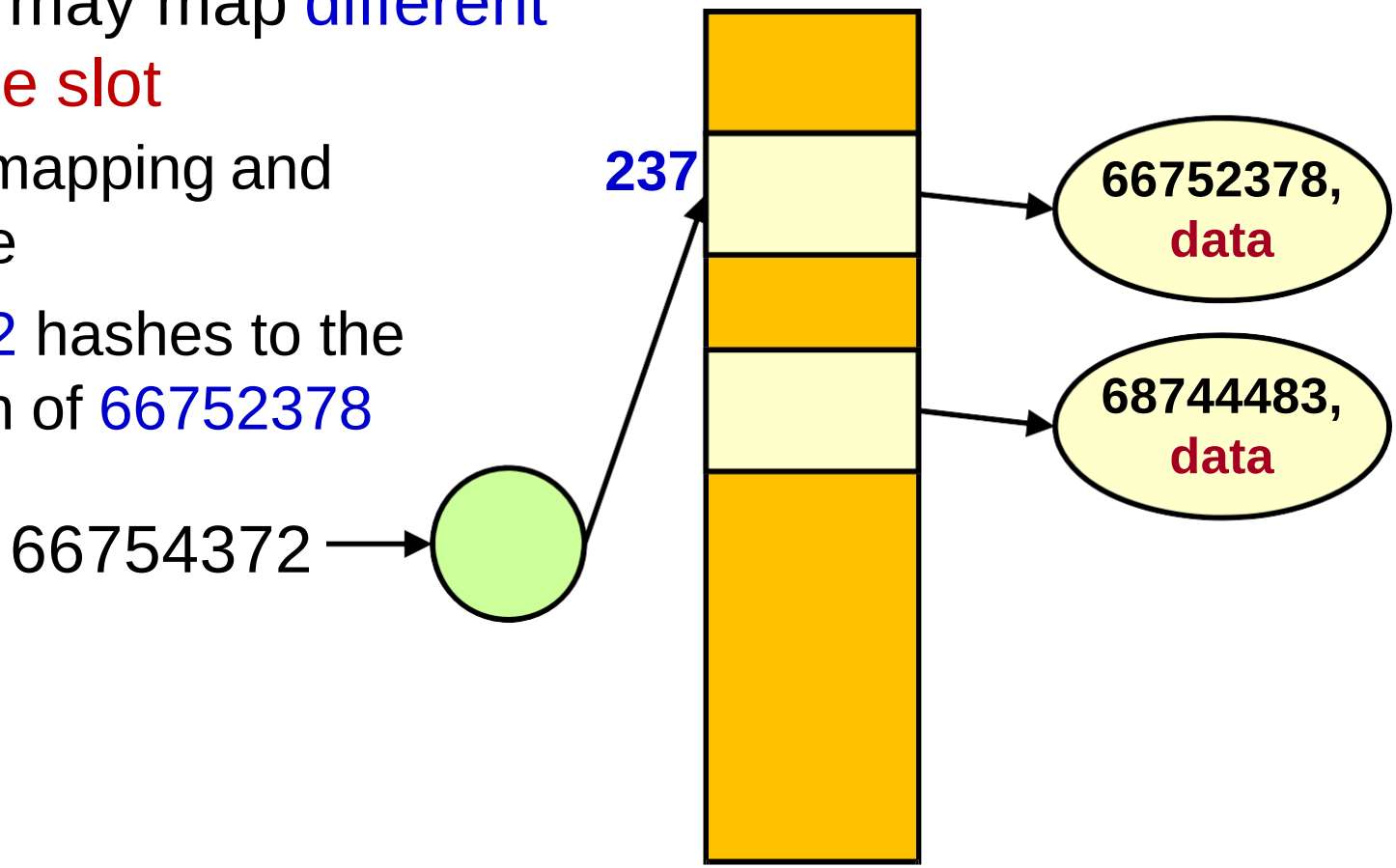
Hash Table: Operations

```
// a[] is an array (the table)
// h is a hash function

insert(key, data)
   a[h(key)] = data
delete(key)
   a[h(key)] = NULL

find(key)
   return a[h(key)]
```

- However, this does not work for all cases!
- Why?

Hash Table: Collision

- A hash function may map different keys to the same slot
  - Many-to-one mapping and not one-to-one
  - E.g. 66754372 hashes to the same location of 66752378



- **Collision**: Two keys have the **same hash value**

Two Major Issues

- How to **hash**?
- How to **resolve collisions**?

Hash Functions

Create a good function!

Hash Functions and Hash Values

- **Assume**: A **hash table** of size ***N***
- **Keys:** Used to identify the data
- **Hash function:** Used to compute a **hash value**
- **Hash value (hash code)**
  - Computed from the key using hash function to get a number in the range 0 ~ N–1
  - Used as the index (address) of the table entry for the data
  - Regarded as the "home address" of a key
- **Goal**: Addresses are different and spread evenly
- When two keys have same hash value — **collision**

Good Hash Functions

- Fast to compute, i.e. complexity: **O(1)**
- **Scatter keys evenly** throughout the hash table
- **Less collisions**
- Need **less slots** (space)

Bad Hash Functions: Example

- **Digit Selection**
  - e.g. choose the 4th and 8th digits of a phone number
  - hash(677**5**437**8**) = 58
  - hash(634**9**782**0**) = 90

- What happens when house phone numbers are hashed by selecting the first three digits?

Perfect Hash Functions

- **Perfect hash function: One-to-one** mapping between keys and hash values. Hence, no collision occurs
- Possible only if all keys are known
- **Applications**: Compiler and interpreter search for reserved words; shell interpreter searches for built-in commands
- Example: GNU gperf is a freely available perfect hash function, written in C++. It automatically constructs perfect functions from a user supplied list of keywords

Thank you!