Hash functions

Hash functions

- Direct hashing
- Modulo division
- Digit extraction
- Mid-square
- Folding
- Rotation
- Pseudo-random

Direct Hashing

The address is the key itself:

$$hash(Key) = Key$$

Direct Hashing

- Advantage: there is no collision.
- Disadvantage: the address space (storage size) is as large as the key space.

Modulo division

$$Address = Key \ mod \ listSize$$

- Fewer collisions if $listSize$ is a prime number.
- Example:
  Numbering system to handle 1,000,000 employees
  Data space to store up to 300 employees
  $hash(121267) = 121267 \ mod \ 307 = 2$

Digit extraction

$$Address = selected\ digits\ from\ Key$$

Example:

379452→394

121267→112

378845→388

160252→102

045128→051

Mid-square

$$Address = middle\ digits\ of\ Key^2$$

Example:

9452 * 9452 = 89340304→3403

Mid-square

- Disadvantage: the size of the Key^2 is too large.
- Variations: use only a portion of the key. Example:
  379452: 379 * 379 = 143641→364
  121267: 121 * 121 = 014641→464
  045128: 045 * 045 = 002025→202

Folding

The key is divided into parts whose size matches the address size.

Example:
Key = 123—456—789
fold shift
$123 + 456 + 789 = 1368$
$\rightarrow 368$

Folding

The key is divided into parts whose size matches the address size.

Example:
Key = 123—456—789
fold shift
$123 + 456 + 789 = 1368$
$\rightarrow 368$

fold boundary
$321 + 456 + 987 = 1764$
$\rightarrow 764$

Rotation

- Hashing keys that are identical except for the last character may create synonyms.
- The key is rotated before hashing.

original key	rotated key
600101	160010
600102	260010
600103	360010
600104	460010
600105	560010

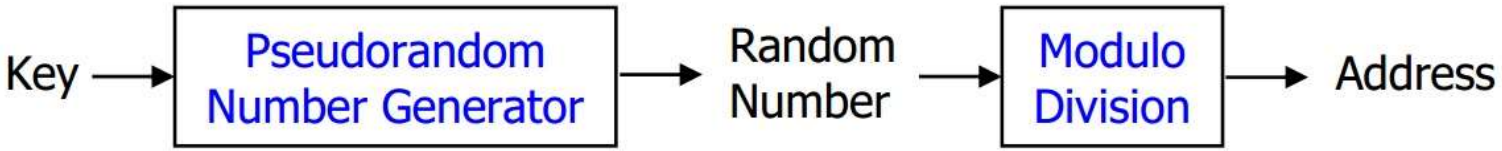
Rotation

- Used in combination with fold shift.

original key		rotated key
600101	$\rightarrow$ 62	160010 $\rightarrow$ 26
600102	$\rightarrow$ 63	260010 $\rightarrow$ 36
600103	$\rightarrow$ 64	360010 $\rightarrow$ 46
600104	$\rightarrow$ 65	460010 $\rightarrow$ 56
600105	$\rightarrow$ 66	560010 $\rightarrow$ 66

Spreading the data more evenly across the address space.

Pseudo-random



$$y = ax + c$$

For maximum efficiency, a and c should be prime numbers.

Pseudo-random

Example:

Key = 121267

a = 17

c = 7

listSize = 307

Address = ((17*121267 + 7) mod 307

= (2061539 + 7) mod 307

= 2061546 mod 307

= 41

Collision resolution

Collision resolution

- Except for the direct hashing, none of the others are one-to-one mapping
  $\rightarrow$ Requiring collision resolution methods
- Each collision resolution method can be used independently with each hash function

Collision resolution

- Closed Hashing
  - Open addressing
  - Bucket hashing
- Open Hashing
  - Linked list resolution

Open addressing

When a collision occurs, an unoccupied element is searched for placing the new element in.

Open addressing

Hash function:

$$h : U \rightarrow \{0, 1, 2, ..., m - 1\}$$

set of keys addresses

Open addressing

Hash and probe function:

$hp : U \times \{0, 1, 2, ..., m-1\} \rightarrow \{0, 1, 2, ..., m-1\}$

set of keys probe numbers addresses

Open Addressing

```
1  Algorithm hashInsert(ref T ¡array¿, val k ¡key¿)
2  Inserts key k into table T.

3  i = 0
4  while i ¡ m do
5  |   j = hp(k, i)
6  |   if T[j] = nil then
7  |   |   T[j] = k
8  |   |   return j
9  |   else
10 |   |   i = i + 1
11 |   end
12 end
13 return error: "hash table overflow"
14 End hashInsert
```


Open Addressing

```
1  Algorithm hashSearch(val T ¡array¿, val k ¡key¿)
2  Searches for key k in table T.

3  i = 0
4  while i ¡ m do
5  |   j = hp(k, i)
6  |   if T[j] = k then
7  |   |   return j
8  |   else if T[j] = nil then
9  |   |   return nil
10 |   else
11 |   |   i = i + 1
12 |   end
13 end
14 return nil
15 End hashSearch
```


Open Addressing

There are different methods:

- Linear probing
- Quadratic probing
- Double hashing
- Key offset

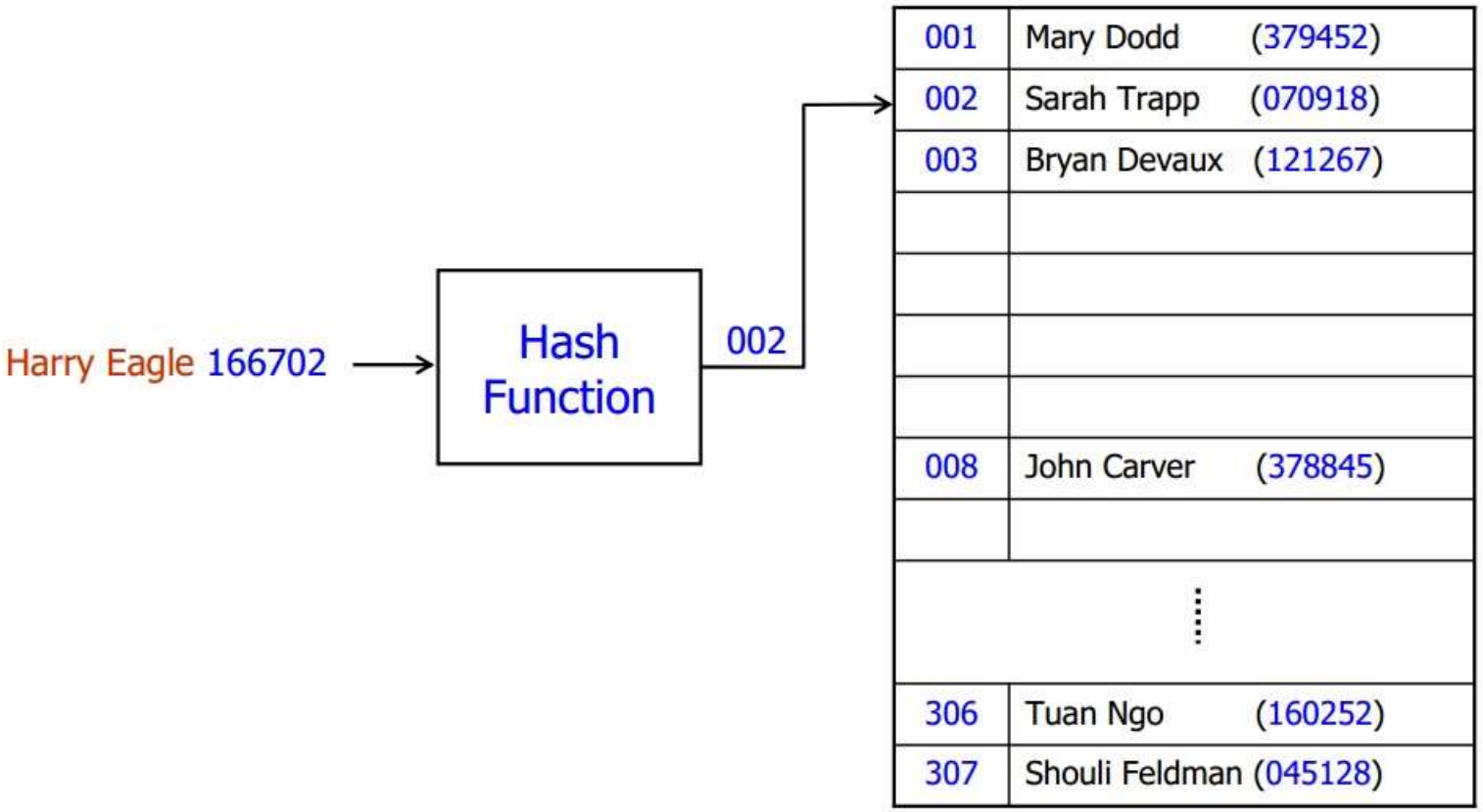
Linear Probing

- When a home address is occupied, go to the next address (the current address + 1):
  $hp(k, i) = (h(k) + i) \ mod \ m$

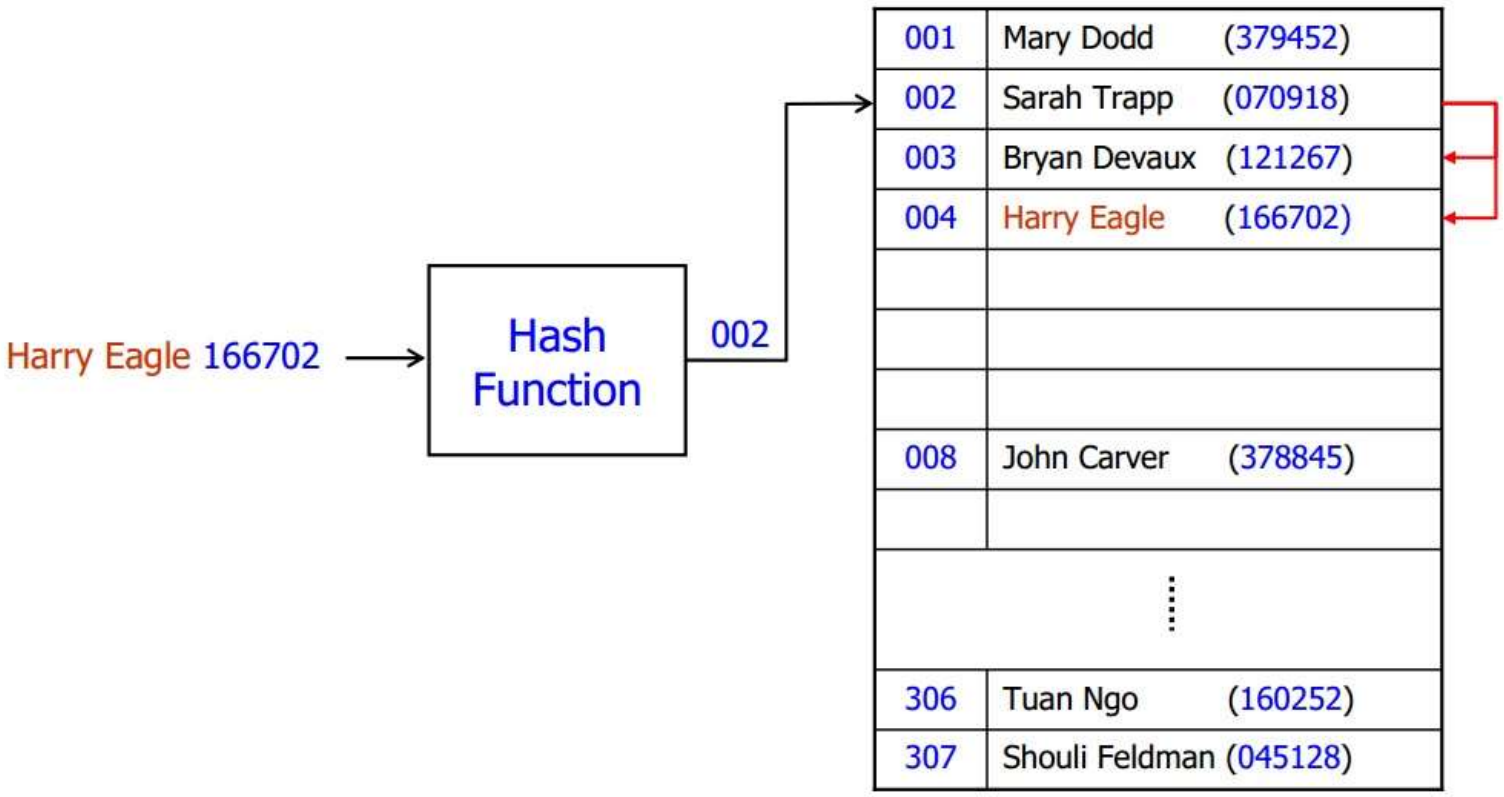
Linear Probing

- When a home address is occupied, go to the next address (the current address + 1):

$$hp(k, i) = (h(k) + i) \ mod \ m$$



Linear Probing



Linear Probing

- Advantages:
  - quite simple to implement
  - data tend to remain near their home address (significant for disk addresses)
- Disadvantages:
  - produces primary clustering

Quadratic Probing

- The address increment is the collision probe number squared:
$$hp(k, i) = (h(k) + i^2) \ mod \ m$$

Quadratic Probing

- Advantages:
  - works much better than linear probing
- Disadvantages:
  - time required to square numbers
  - produces secondary clustering
    $h(k_1) = h(k_2) \rightarrow hp(k_1, i) = hp(k_2, i)$

Double Hashing

- Using two hash functions:
  $hp(k, i) = (h_1(k) + ih_2(k)) \ mod \ m$

Key Offset

- The new address is a function of the collision address and the key.

$$offset = [key/listSize]$$
$$newAddress = (collisionAddress + offset)\ mod\ listSize$$

Key Offset

- The new address is a function of the collision address and the key.

$$offset = [key/listSize]$$
$$newAddress = (collisionAddress + offset)\ mod\ listSize$$

$$hp(k,i) = (hp(k,i-1) + [k/m])\ mod\ m$$

Open addressing

Hash and probe function:

$hp : U \times \{0, 1, 2, ..., m-1\} \rightarrow \{0, 1, 2, ..., m-1\}$

set of keys probe numbers addresses

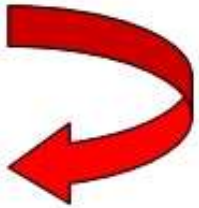
$\{hp(k, 0), hp(k, 1), \ldots, hp(k, m-1)\}$ is a permutation of $\{0, 1, \ldots, m-1\}$

Bucket hashing

- Hashing data to buckets that can hold multiple pieces of data.
- Each bucket has an address and collisions are postponed until the bucket is full.

Bucket hashing

001	Mary Dodd (379452)
002	Sarah Trapp (070918)
	Harry Eagle (166702)
	Ann Georgis (367173)
003	Bryan Devaux (121267)
	Chris Walljasper(572556)
⋮	
307	Shouli Feldman (045128)

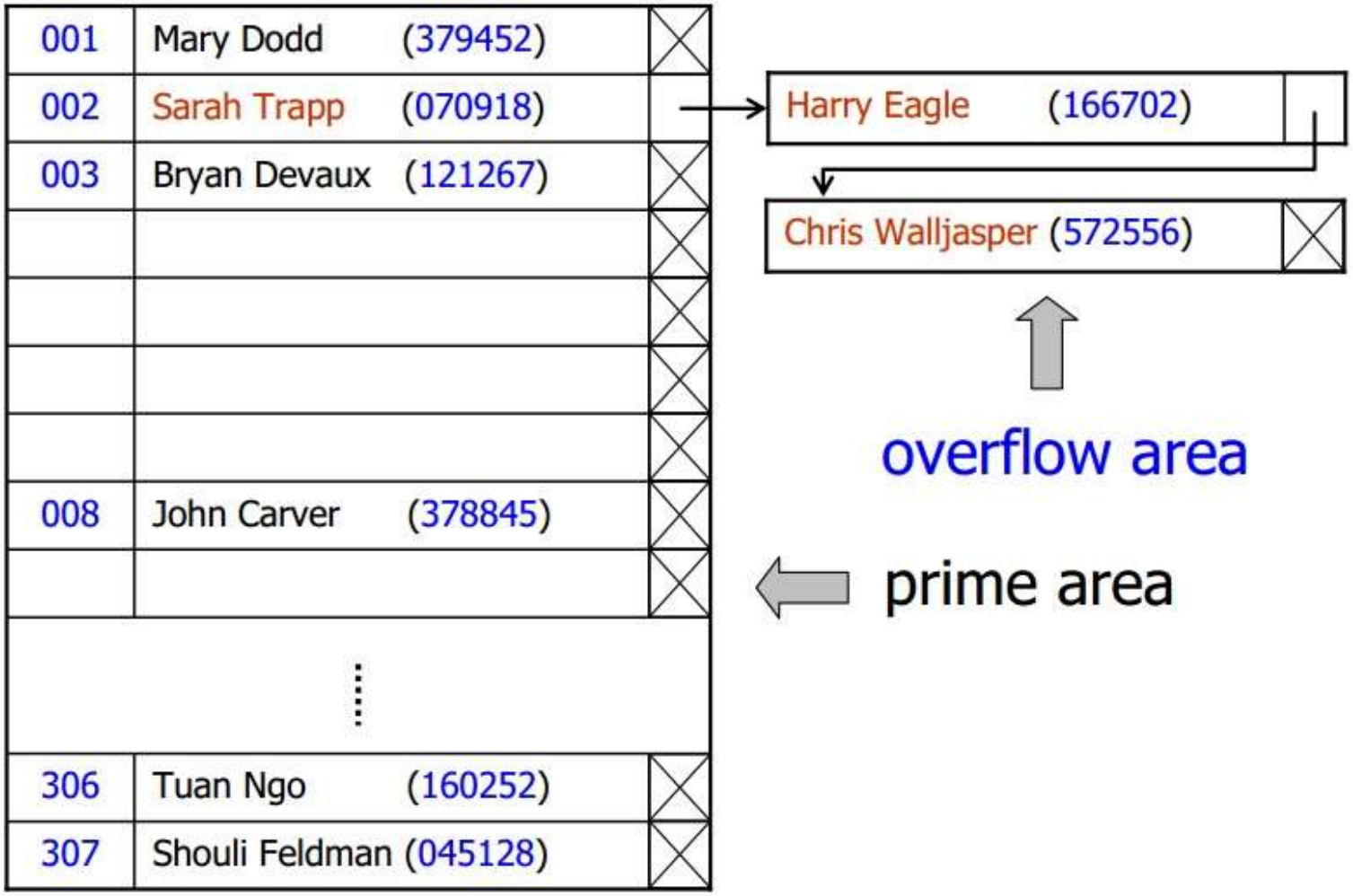


linear probing

Linked List Resolution

- Major disadvantage of Open Addressing: each collision resolution increases the probability for future collisions.
  $\longrightarrow$ use linked lists to store synonyms

Linked list resolution



THANK YOU.