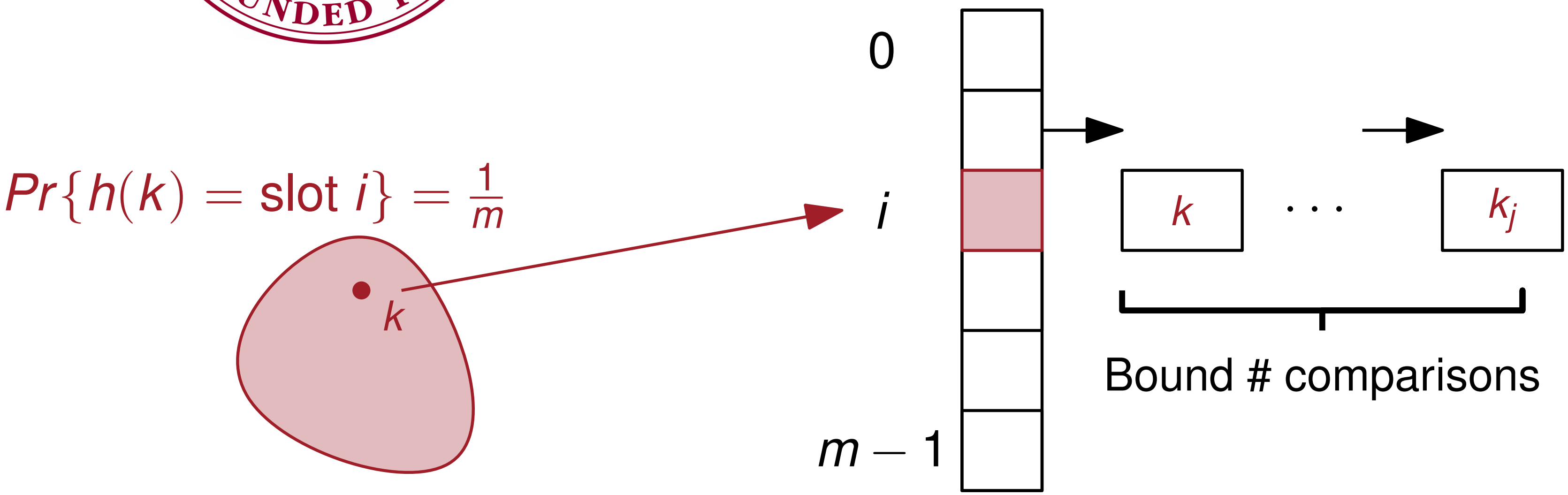
COSC 302, Spring 2018
Lecture 6.2: Hash tables

Prof. Darren Strash

Department of Computer Science
Colgate University



Breaking the search lower bound

Just like with linear-time sorting, use value of elements.

Example: Integers 0 to $m-1$, can store in Boolean array $A[0..m-1]$
$\rightarrow O(1)$-time search

A:	0	0	0	1	0	1	0	0	1	0	0	1	0	1	0	1
	0															$m-1$

What if elements aren't integers 0 to $m-1$?

If integers, can map to 0 to $m-1$ with $\bmod m$ operator

If two integers are same after mapping? $\rightarrow$ handle *collisions*.

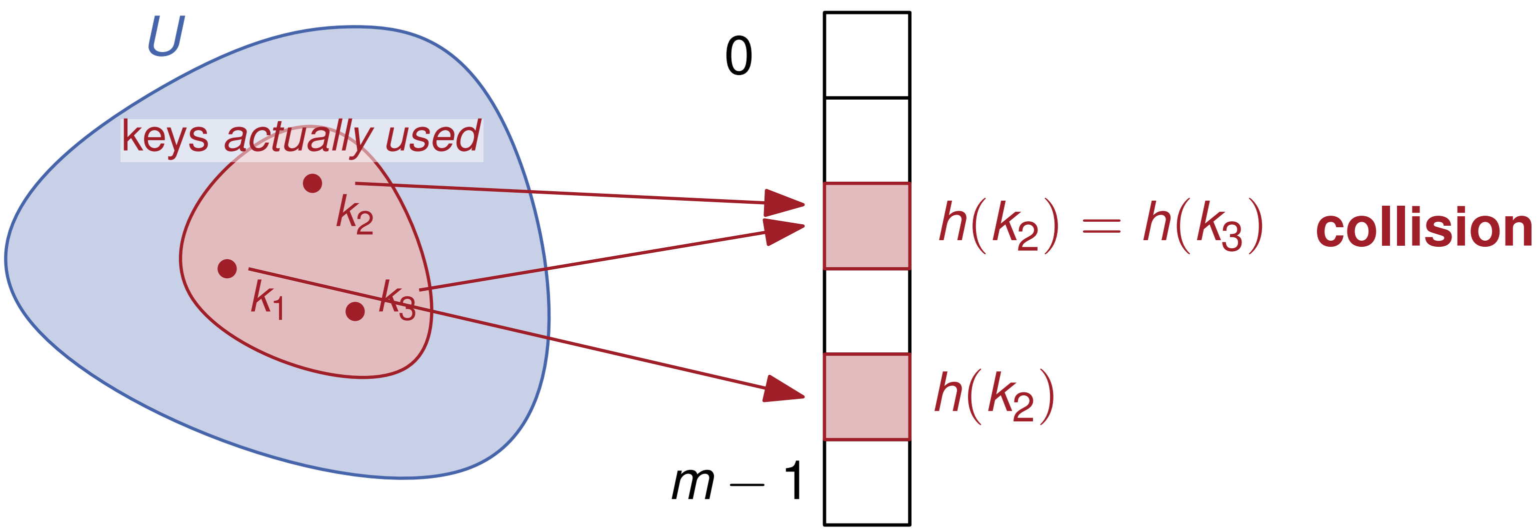
If not integers, map to integer! $\rightarrow$ *hash function*

Hash functions and direct addressing

Map a universe of values U to their key (index) in $\{0, 1, \ldots m-1\}$

$$h : U \to \{0, 1, \ldots, m-1\}$$

Store values in a *direct address table*, in *slots*.



Hash functions and direct addressing

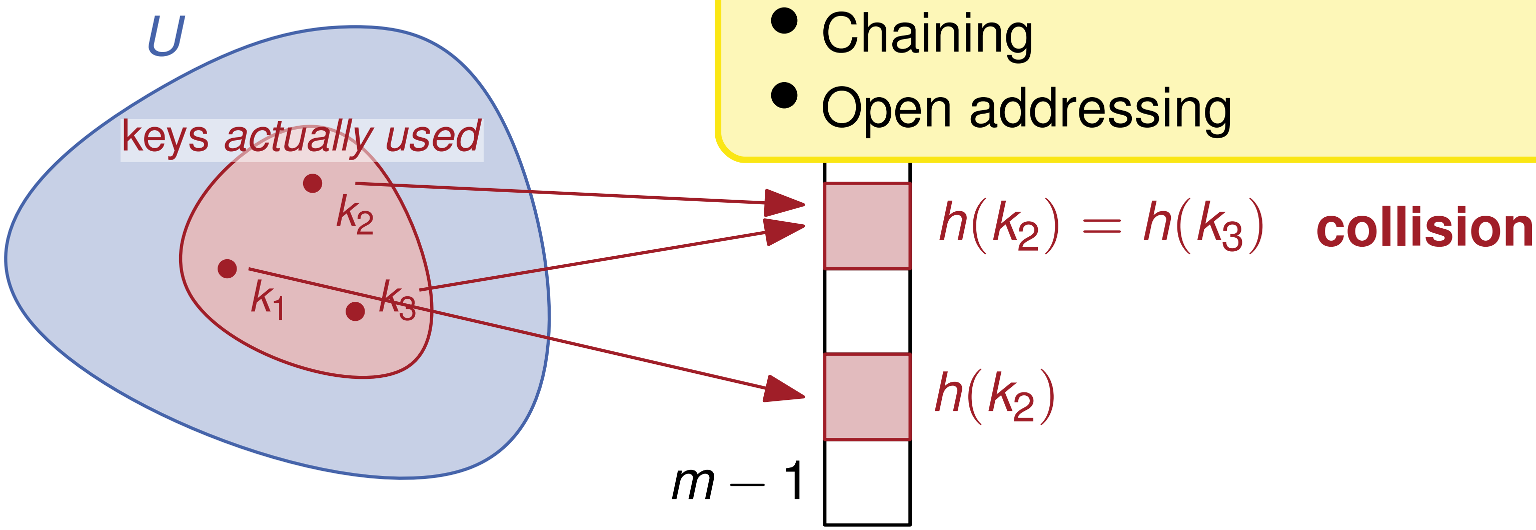
Map a universe of values U to their key (index) in $\{0, 1, \ldots m-1\}$

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

Store values in a *direct address table*, in *slots*

Two methods to handle collisions:

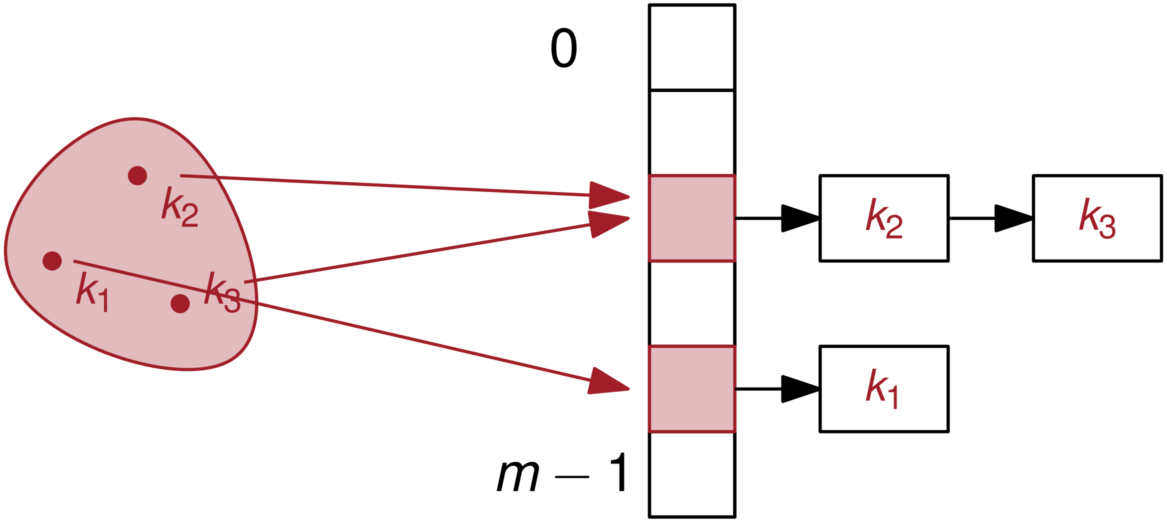
- Chaining
- Open addressing



Handling collisions: chaining

Keys with colliding hash values are stored in **linked list** in the corresponding cell

Keys are **added** and **removed** from the linked lists



Chaining: analysis

We assume **simple uniform hashing**:

> Any given key is equally likely to hash to any of the m slots independently of where any key is hashed to.

Chaining: analysis

We assume **simple uniform hashing**:

> Any given key is $\boxed{\text{equally likely}}$ to hash to any of the *m* slots $\boxed{\text{independently}}$ of where any key is hashed to.

$\rightarrow$ this assumption is critical to analysis

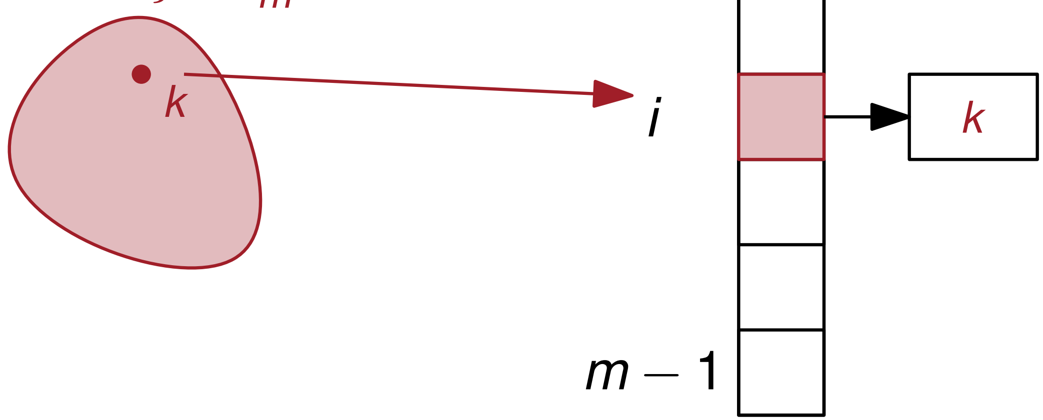
Chaining: analysis

We assume **simple uniform hashing**:

> Any given key is $\boxed{\text{equally likely}}$ to hash to any of the m slots $\boxed{\text{independently}}$ of where any key is hashed to.

$\rightarrow$ this assumption is critical to analysis

$$Pr\{h(k) = \text{slot } i\} = \frac{1}{m}$$



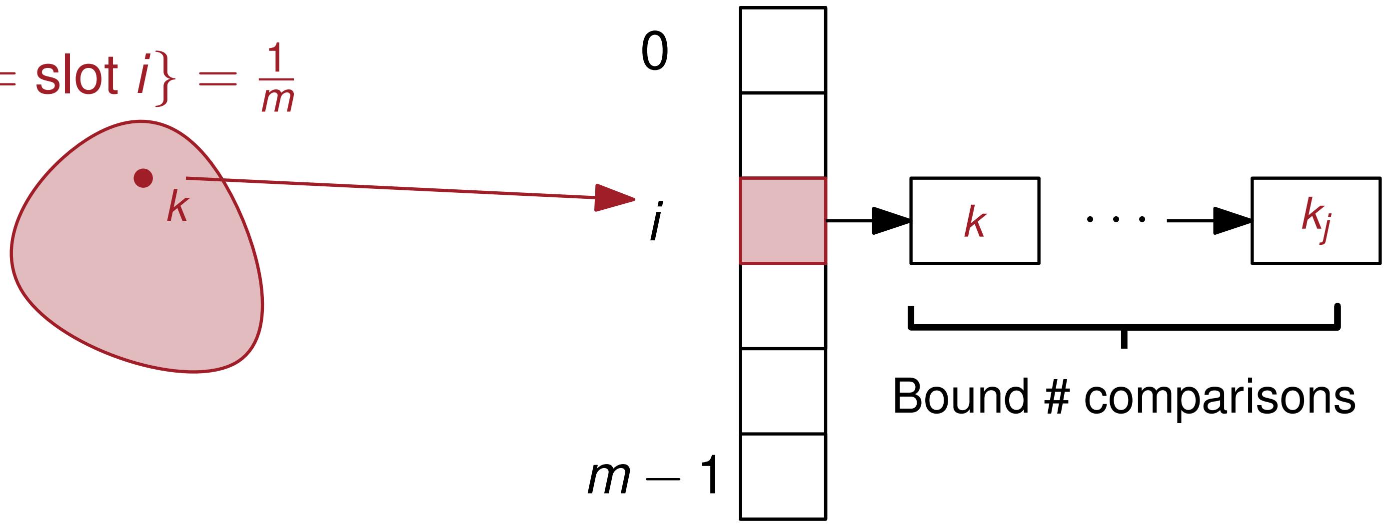
Chaining: analysis

We assume **simple uniform hashing**:

> Any given key is equally likely to hash to any of the m slots independently of where any key is hashed to.

→ this assumption is critical to analysis

$Pr\{h(k) = \text{slot } i\} = \frac{1}{m}$

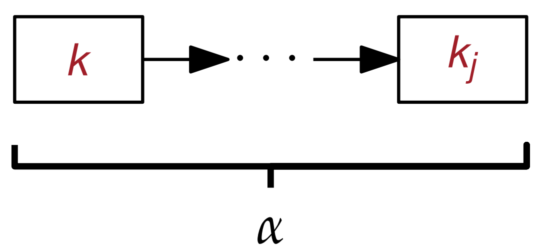


Chaining: unsuccessful searches

Let $\alpha = \frac{n}{m}$ be the **load factor** for the hash table.

Thm In hashing by chaining, *unsuccessful* searches take expected time $\Theta(1 + \alpha)$

Expected length of chain is $\frac{n}{m} = \alpha$ with simple uniform hashing



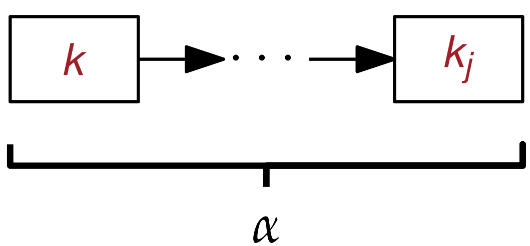
- $\Theta(1)$ for hash + access
- $\Theta(\alpha)$ to traverse and compare keys of *entire chain*

Chaining: unsuccessful searches

Let $\alpha = \frac{n}{m}$ be the **load factor** for the hash table.

Thm In hashing by chaining, *unsuccessful* searches take expected time $\Theta(1 + \alpha)$

Expected length of chain is $\frac{n}{m} = \alpha$ with simple uniform hashing



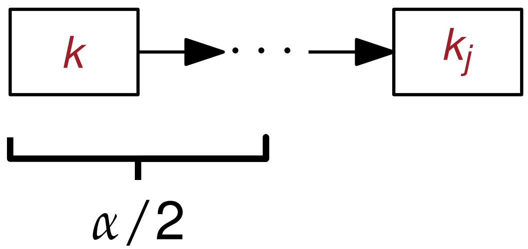
- $\Theta(1)$ for hash + access
- $\Theta(\alpha)$ to traverse and compare keys of *entire chain*

$\rightarrow$ constant time if α is a constant!

Chaining: successful searches

Thm In hashing by chaining, *successful* searches take expected time $\Theta(1+\alpha)$

Intuition: expect to be in middle of chain $\approx \alpha/2$ comparisons



- $\Theta(1)$ for hash + access
- $\approx \alpha/2 = \Theta(\alpha)$ to traverse and compare keys in chain

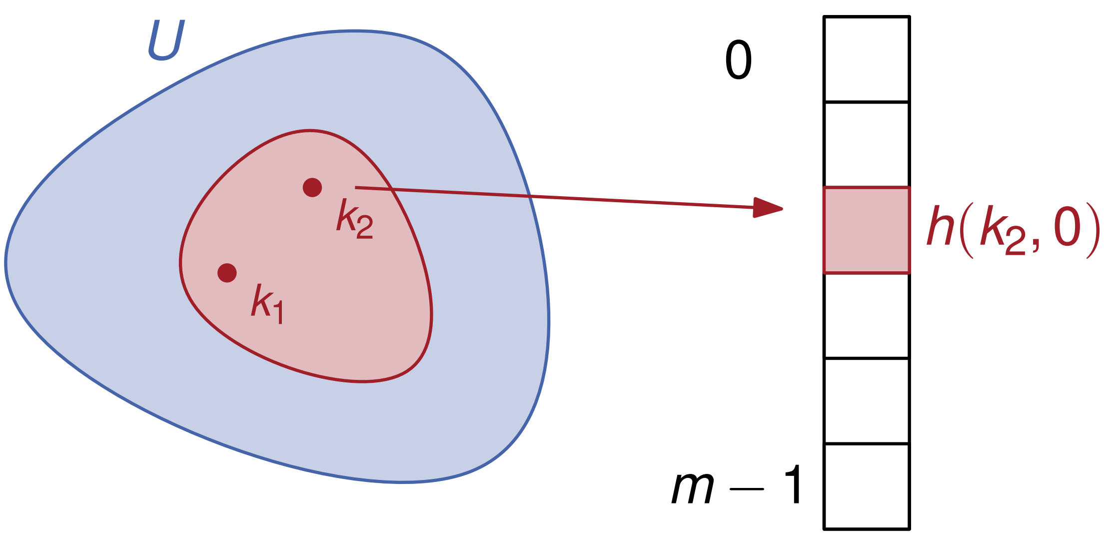
Open addressing: probe sequences

Open addressing: Store keys directly in slots of hash table

$\rightarrow$ if collision, choose new slot, new hash function:

$$h : U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}$$

$$h(k, i) := (i+1)\text{-th slot probed}$$



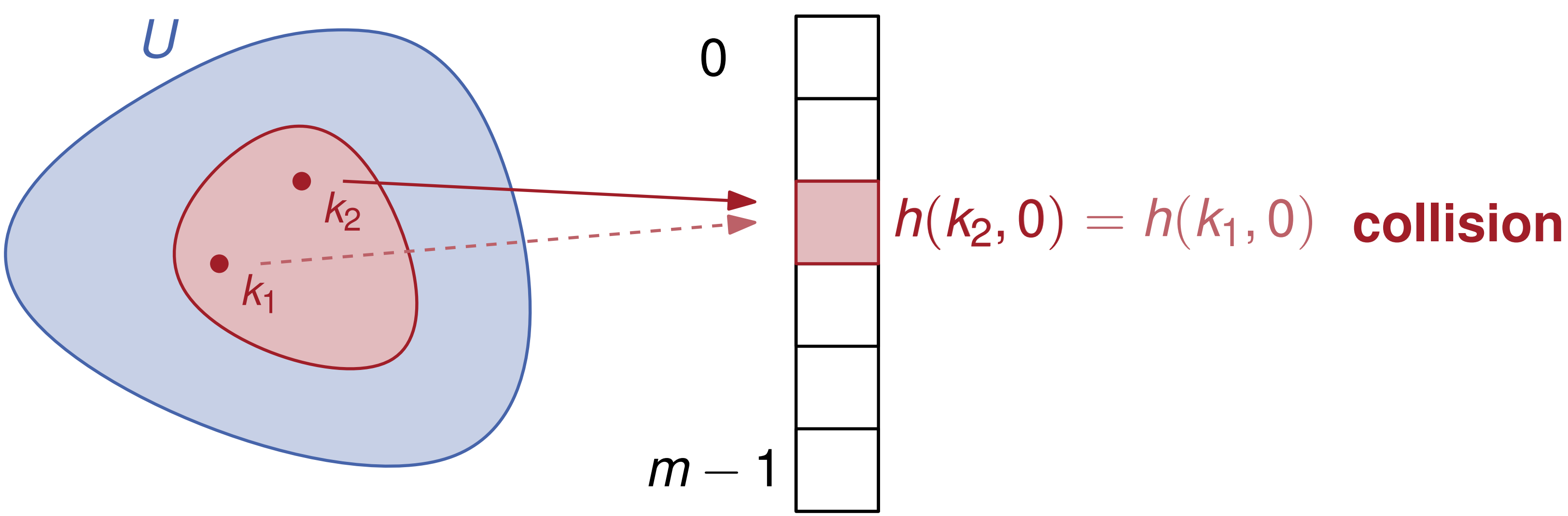
Open addressing: probe sequences

Open addressing: Store keys directly in slots of hash table

$\rightarrow$ if collision, choose new slot, new hash function:

$$h : U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}$$

$$h(k, i) := (i + 1)\text{-th slot probed}$$



Department of Computer Science
Colgate University

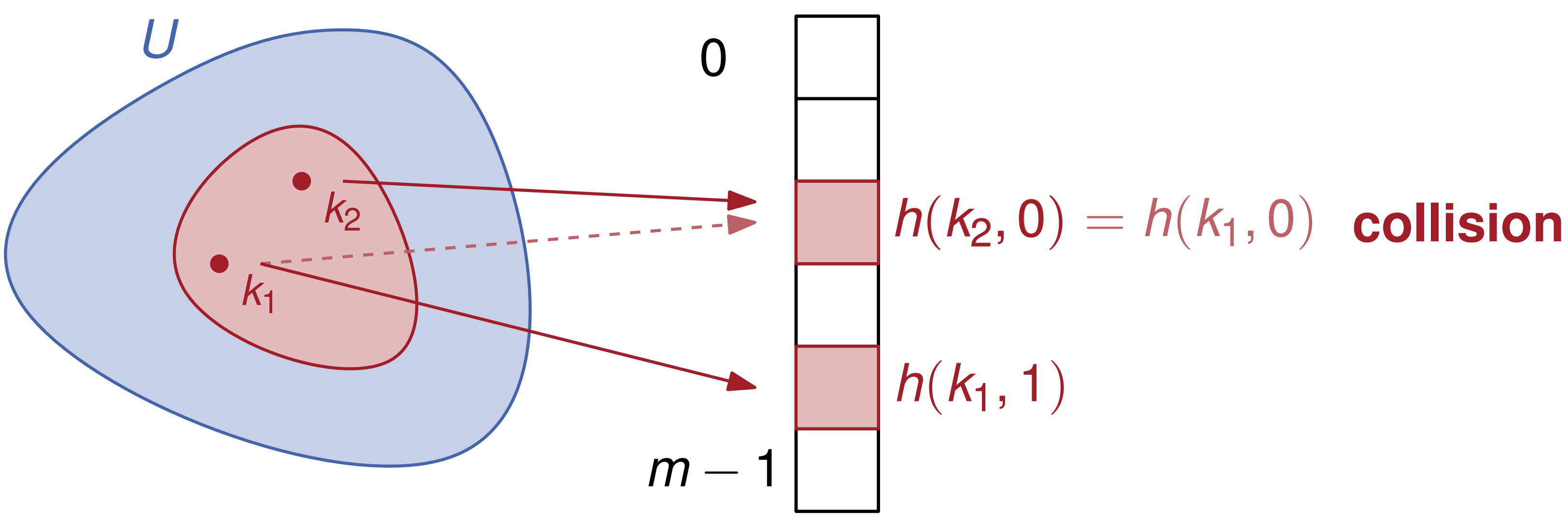
Open addressing: probe sequences

Open addressing: Store keys directly in slots of hash table

$\rightarrow$ if collision, choose new slot, new hash function:

$$h : U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}$$

$$h(k, i) := (i+1)\text{-th slot probed}$$



Open addressing: probe sequences

Probe sequence: $\langle h(k,0), h(k,1), \ldots, h(k,m-1)\rangle$

$\rightarrow$ a permutation of $\langle 0, 1, \ldots, m-1\rangle$

Described by a *probing strategy*: decides how to probe after collision

Open addressing: probe sequences

Probe sequence: $\langle h(k,0), h(k,1), \ldots, h(k,m-1) \rangle$

$\rightarrow$ a permutation of $\langle 0, 1, \ldots, m-1 \rangle$

Described by a *probing strategy*: decides how to probe after collision

Strategies:

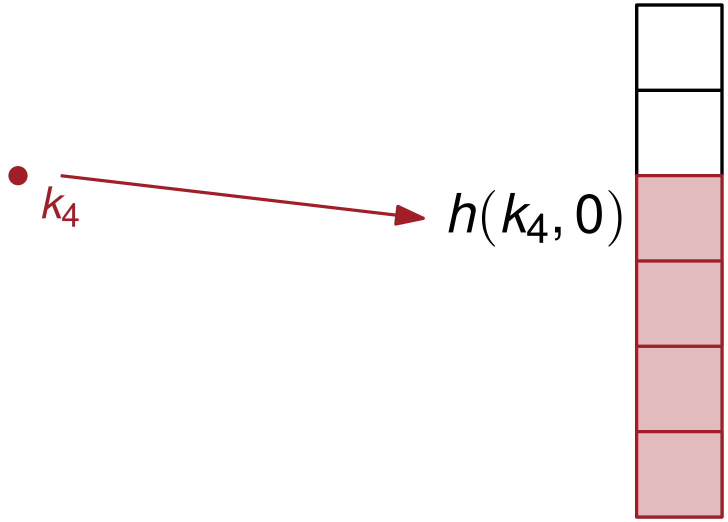
- Linear probing $\rightarrow$ probe cells sequentially
- Quadratic probing $\rightarrow$ probe cells with quadratic function offset
- Double hashing $\rightarrow$ probe cells using *another* hash function

Goal: keep number of probes during search *small*

Linear probing

Use auxiliary hash function $h'(k)$:

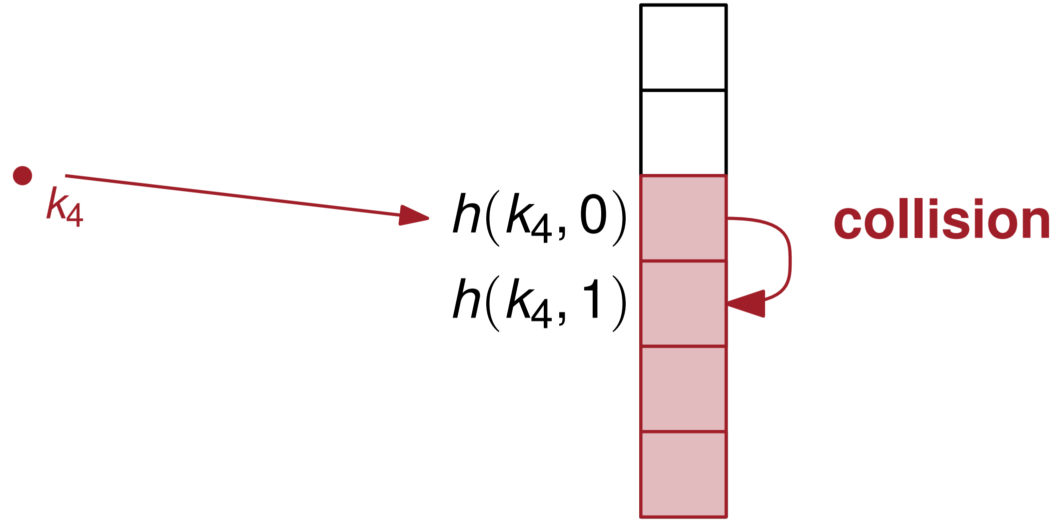
$$h(k,i) = (h'(k) + i) \mod m$$



Linear probing

Use auxiliary hash function $h'(k)$:

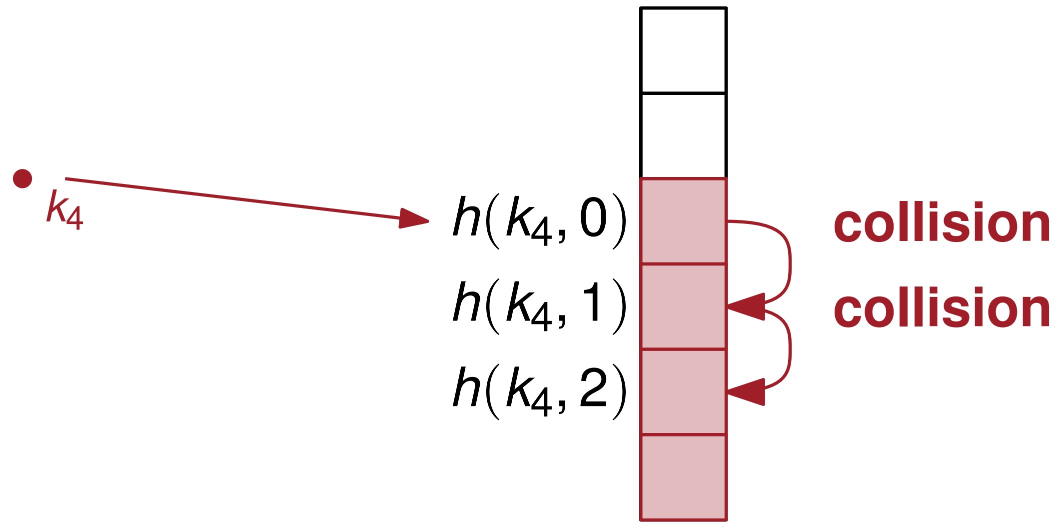
$$h(k, i) = (h'(k) + i) \mod m$$



Linear probing

Use auxiliary hash function $h'(k)$:

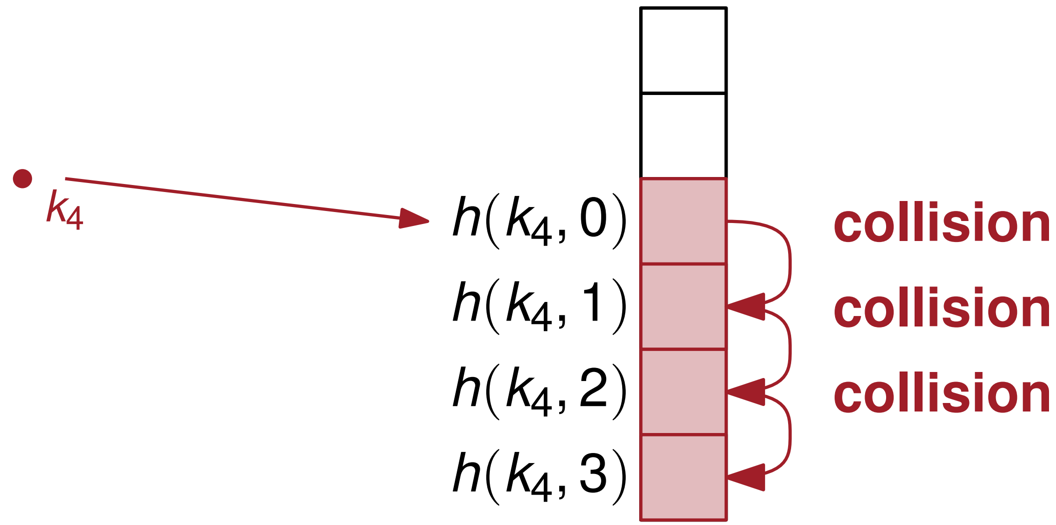
$$h(k, i) = (h'(k) + i) \mod m$$



Linear probing

Use auxiliary hash function $h'(k)$:

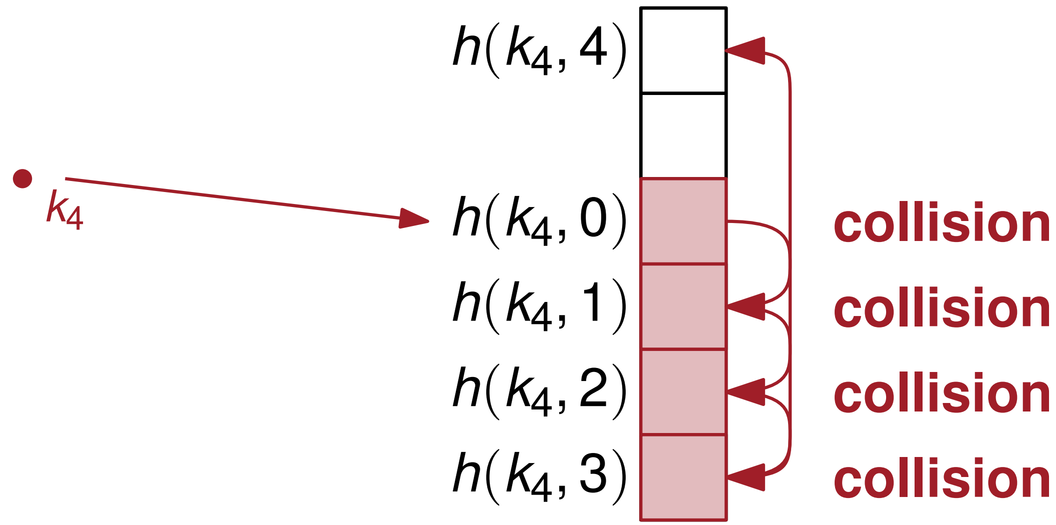
$$h(k, i) = (h'(k) + i) \mod m$$



Linear probing

Use auxiliary hash function $h'(k)$:

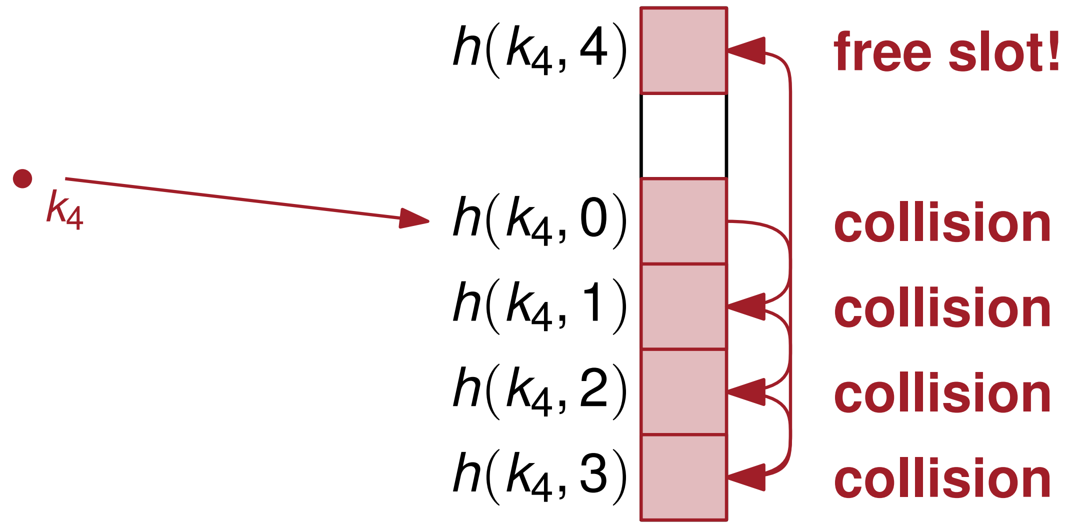
$$h(k, i) = (h'(k) + i) \mod m$$



Linear probing

Use auxiliary hash function $h'(k)$:

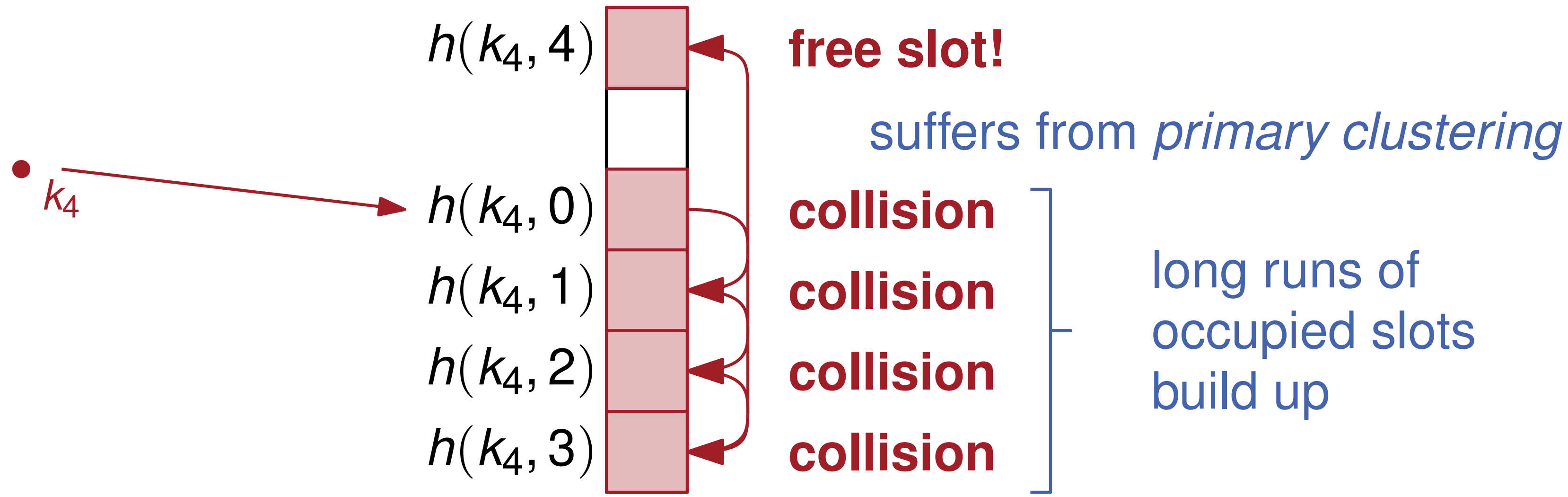
$$h(k,i) = (h'(k) + i) \mod m$$



Linear probing

Use auxiliary hash function $h'(k)$:

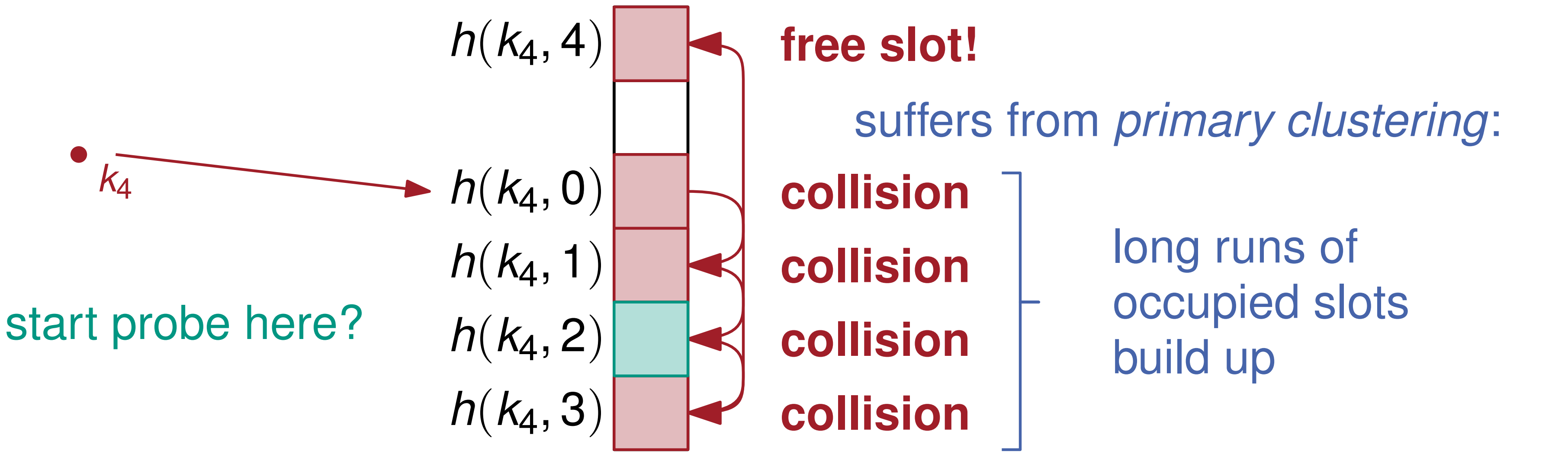
$$h(k, i) = (h'(k) + i) \mod m$$



Linear probing

Use auxiliary hash function $h'(k)$:

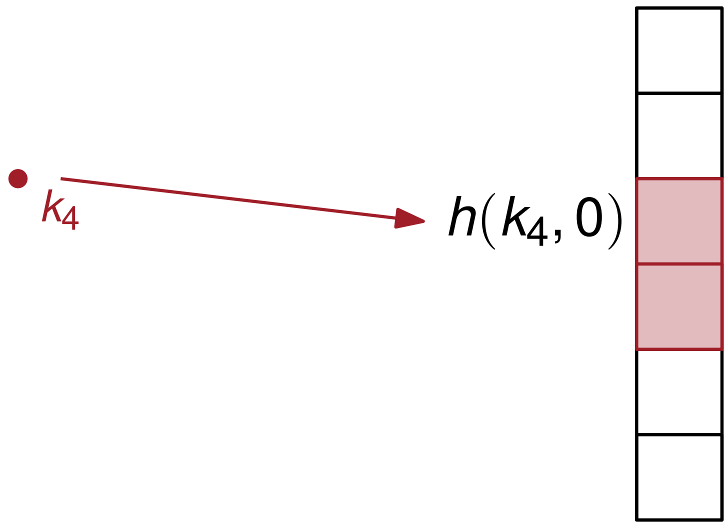
$$h(k,i) = (h'(k) + i) \mod m$$



Quadratic probing

Use auxiliary hash function $h'(k)$:

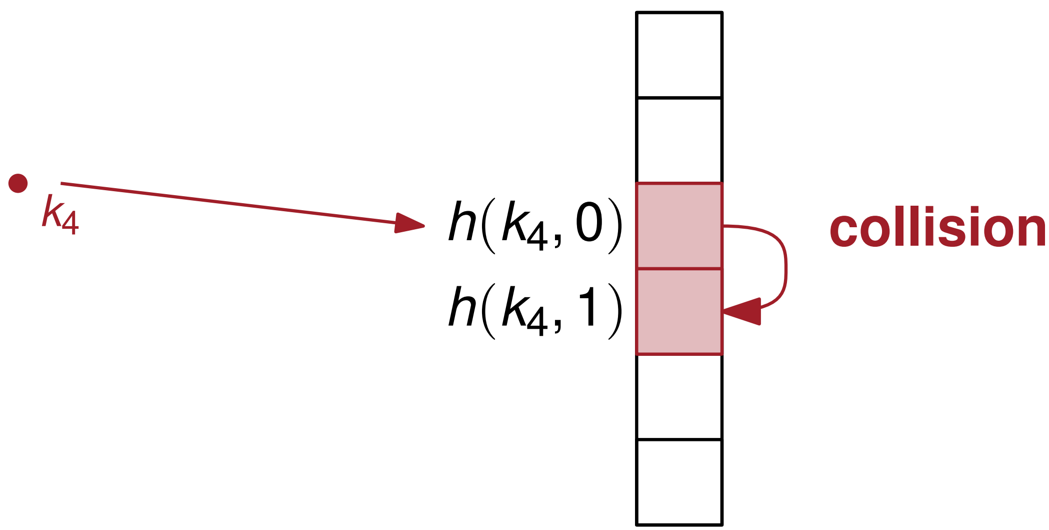
$$h(k, i) = (h'(k) + \underbrace{c_1 i + c_2 i^2}_{\text{auxiliary constants chosen to make probe sequence a permutation}}) \mod m$$



Quadratic probing

Use auxiliary hash function $h'(k)$:

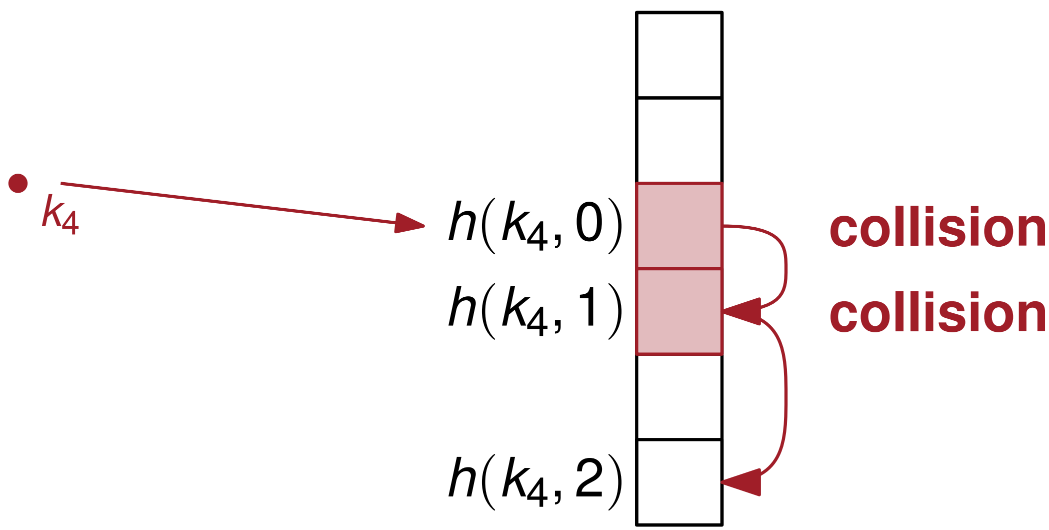
$$h(k,i) = (h'(k) + \underbrace{c_1 i + c_2 i^2}_{\text{auxiliary constants chosen to make probe sequence a permutation}}) \mod m$$



Quadratic probing

Use auxiliary hash function $h'(k)$:

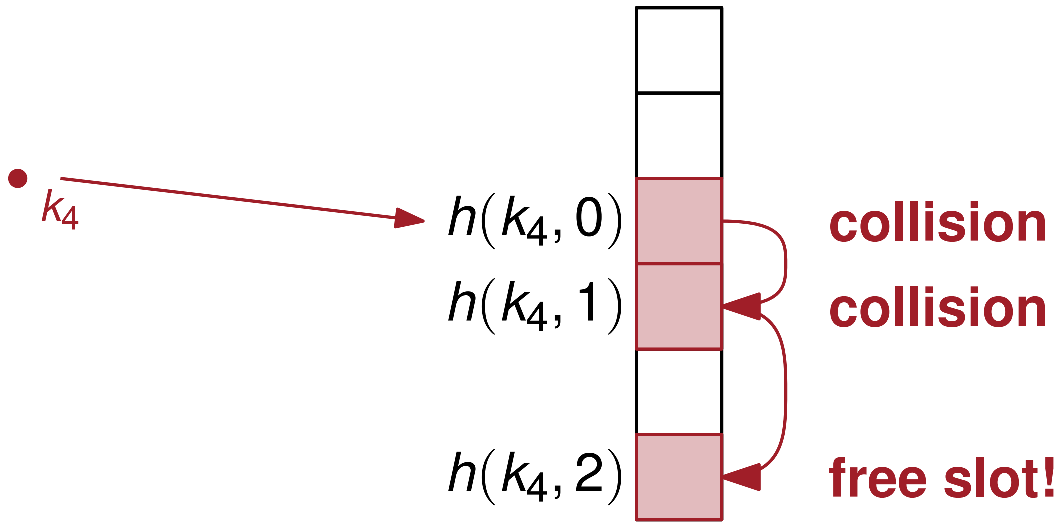
$$h(k, i) = (h'(k) + \underbrace{c_1 i + c_2 i^2}_{\text{auxiliary constants chosen to make probe sequence a permutation}}) \mod m$$



Quadratic probing

Use auxiliary hash function $h'(k)$:

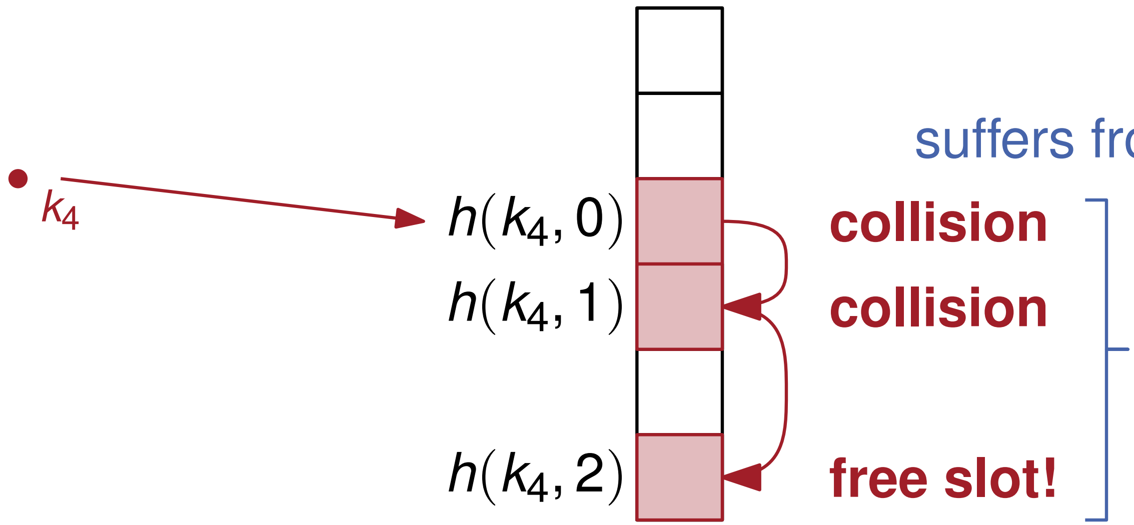
$$h(k, i) = (h'(k) + \underbrace{c_1 i + c_2 i^2}_{\text{auxiliary constants chosen to make probe sequence a permutation}}) \mod m$$



Quadratic probing

Use auxiliary hash function $h'(k)$:

$$h(k,i) = (h'(k) + \underbrace{c_1 i + c_2 i^2}_{\text{auxiliary constants chosen to make probe sequence a permutation}}) \mod m$$



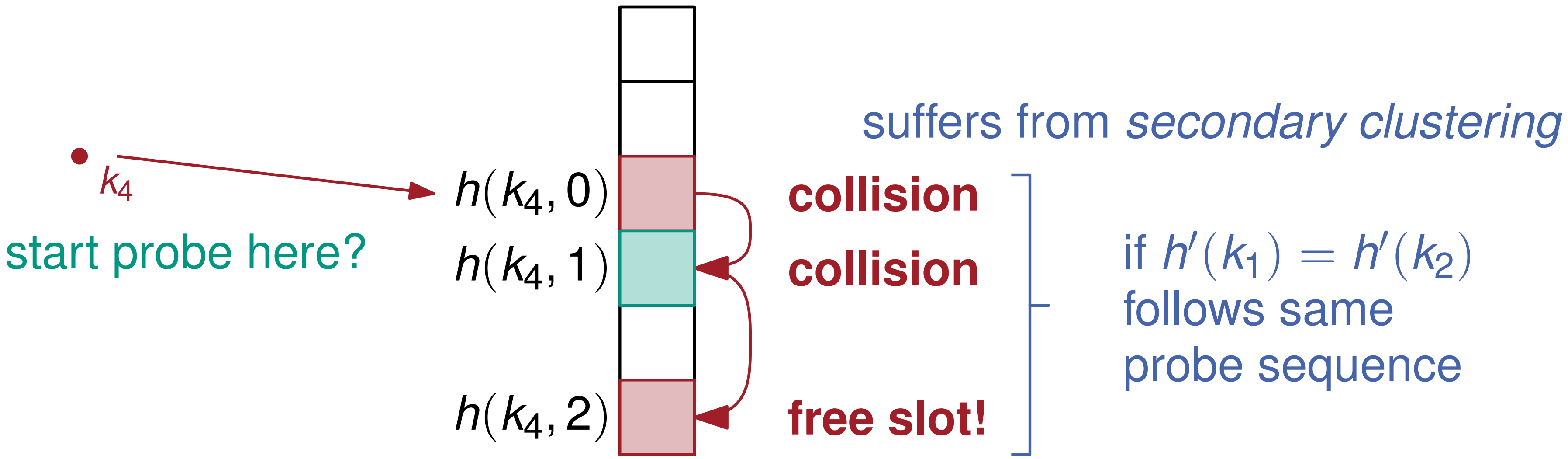
suffers from *secondary clustering*:

if $h'(k_1) = h'(k_2)$ follows same probe sequence

Quadratic probing

Use auxiliary hash function $h'(k)$:

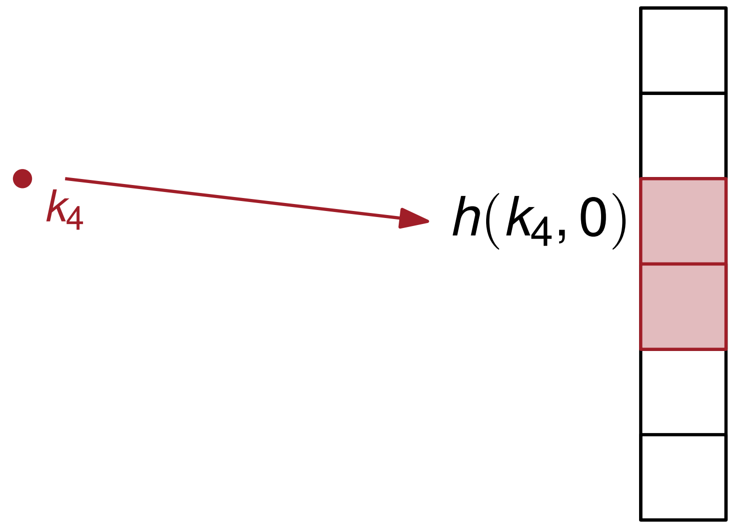
$$h(k, i) = (h'(k) + \underbrace{c_1 i + c_2 i^2}_{\text{auxiliary constants chosen to make probe sequence a permutation}}) \mod m$$



Double hashing

Use auxiliary hash functions $h_1(k)$ and $h_2(k)$:

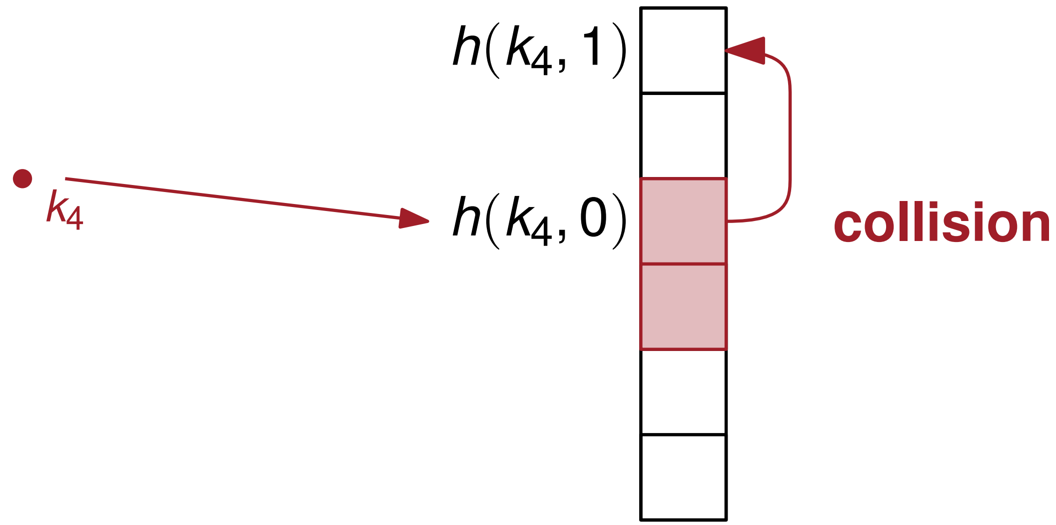
$$h(k, i) = (h_1(k) + \underbrace{ih_2(k)}_{\text{offset also depends on key } k}) \mod m$$



Double hashing

Use auxiliary hash functions $h_1(k)$ and $h_2(k)$:

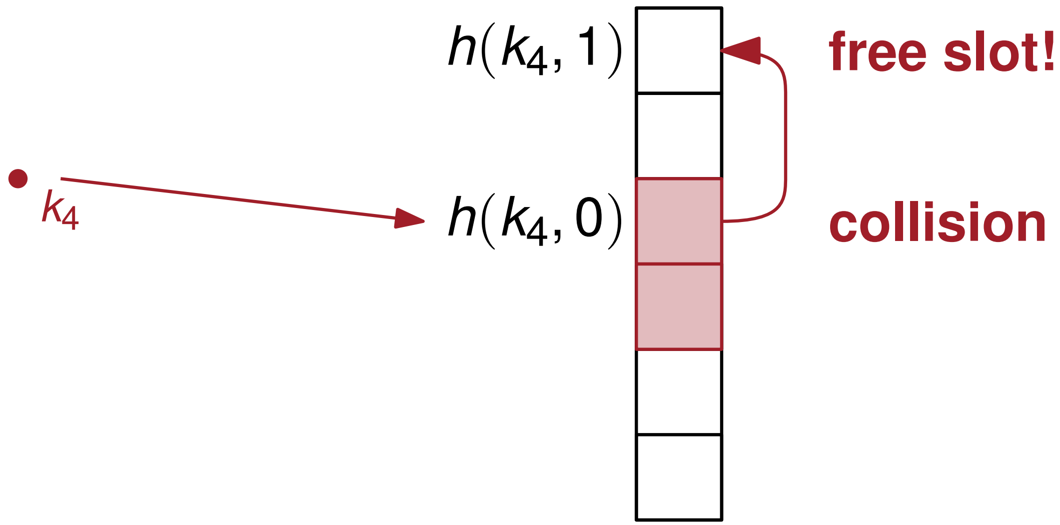
$$h(k, i) = (h_1(k) + \underbrace{ih_2(k)}_{\text{offset also depends on key } k}) \mod m$$



Double hashing

Use auxiliary hash functions $h_1(k)$ and $h_2(k)$:

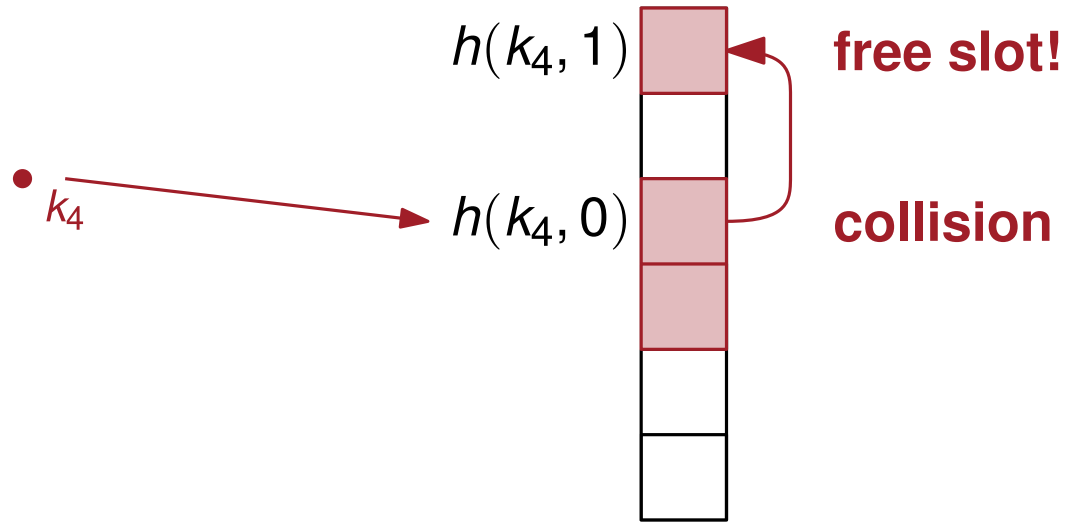
$$h(k,i) = (h_1(k) + \underbrace{ih_2(k)}_{\text{offset also depends on key } k}) \mod m$$



Double hashing

Use auxiliary hash functions $h_1(k)$ and $h_2(k)$:

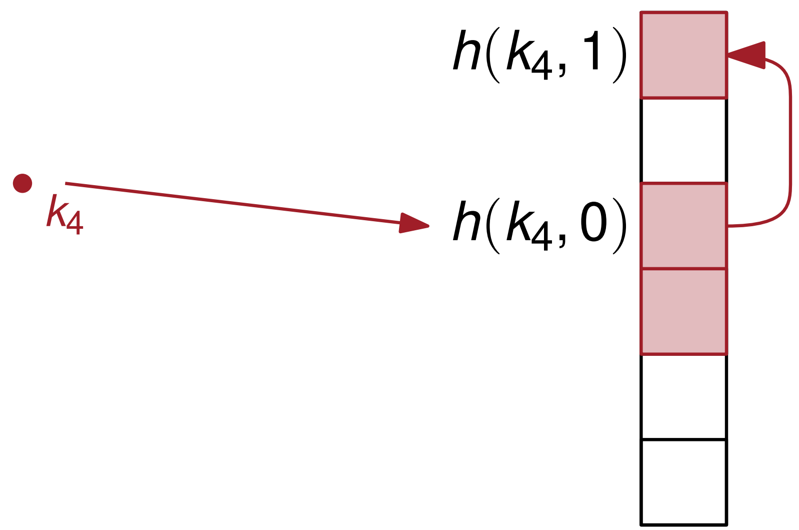
$$h(k, i) = (h_1(k) + \underbrace{ih_2(k)}_{\text{offset also depends on key } k}) \mod m$$



Double hashing

Use auxiliary hash functions $h_1(k)$ and $h_2(k)$:

$$h(k,i) = (h_1(k) + \underbrace{ih_2(k)}_{\text{offset also depends on key } k}) \mod m$$



free slot!

clustering?

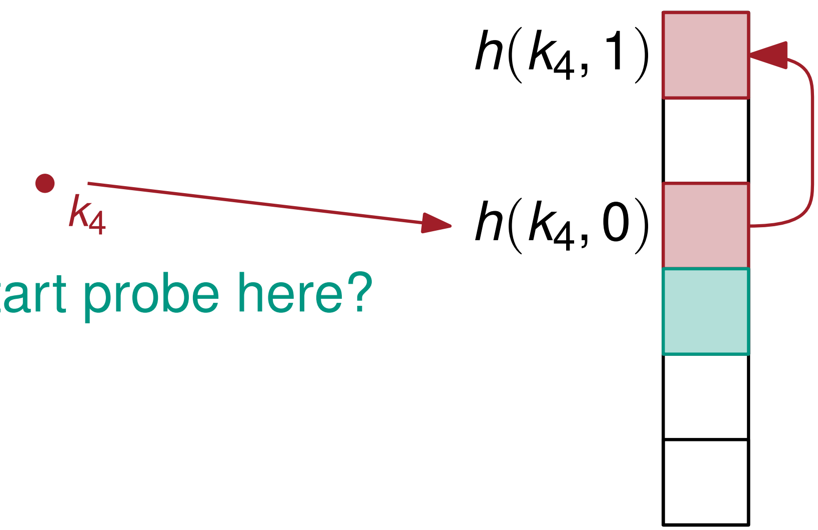
collision

if $h_1(k_1) = h_1(k_2)$
and
$h_2(k_1) = h_2(k_2)$
follows same
probe sequence

Double hashing

Use auxiliary hash functions $h_1(k)$ and $h_2(k)$:

$$h(k,i) = (h_1(k) + \underbrace{ih_2(k)}_{\text{offset also depends on key } k}) \mod m$$



free slot!

clustering?

collision

start probe here?

if $h_1(k_1) = h_1(k_2)$
and
$h_2(k_1) = h_2(k_2)$
follows same
probe sequence

Open addressing: analysis

We assume **uniform hashing**:

> Any given key is equally likely to have any of the $m!$ permutations of $\langle 0, 1, \ldots, m-1 \rangle$ as its probe sequence.

$\rightarrow$ linear probing, quadratic probing do **not** meet this requirement!

$\rightarrow$ double hashing is *closest*

Open addressing: analysis

We assume **uniform hashing**:

> Any given key is equally likely to have any of the $m!$ permutations of $\langle 0, 1, \ldots, m-1 \rangle$ as its probe sequence.

$\rightarrow$ linear probing, quadratic probing do **not** meet this requirement!

$\rightarrow$ double hashing is *closest*

Open addressing: analysis

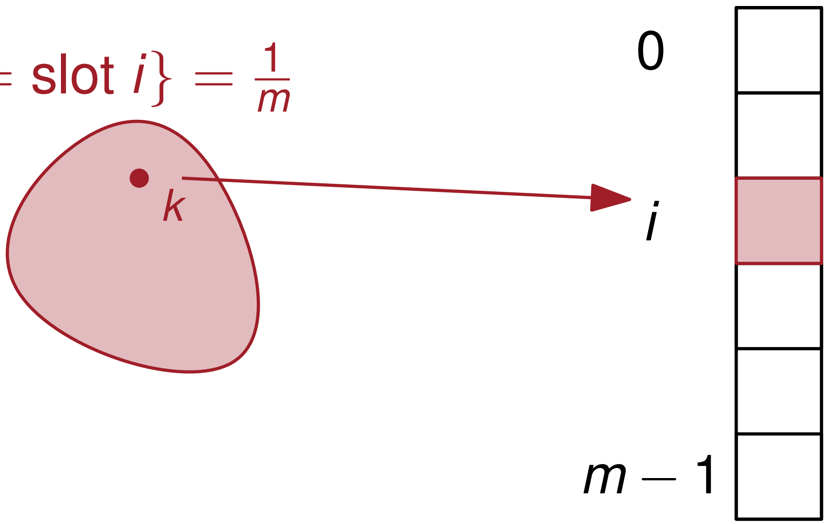
We assume **uniform hashing**:

> Any given key is equally likely to have any of the $m!$ permutations of $\langle 0, 1, \ldots, m-1 \rangle$ as its probe sequence.

$\rightarrow$ linear probing, quadratic probing do **not** meet this requirement!

$\rightarrow$ double hashing is *closest*

$$Pr\{h(k) = \text{slot } i\} = \frac{1}{m}$$



Open addressing: analysis

We assume **uniform hashing**:

> Any given key is equally likely to have any of the $m!$ permutations of $\langle 0, 1, \ldots, m-1 \rangle$ as its probe sequence.

$\rightarrow$ linear probing, quadratic probing do **not** meet this requirement!

$\rightarrow$ double hashing is *closest*

$Pr\{h(k) = \text{slot } i\} = \frac{1}{m}$