THE UNIVERSITY OF VERMONT
COLLEGE OF ENGINEERING & MATHEMATICAL SCIENCES

Hash Tables: Double Hashing

CS 124 / Department of Computer Science

Double hashing

So far we've seen three collision resolution policies, separate chaining, linear probing, and quadratic probing.

Double hashing is another approach to resolving hash collisions.

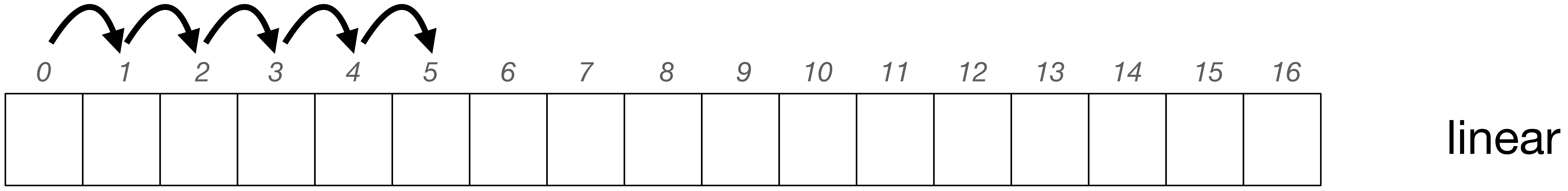
Double hashing

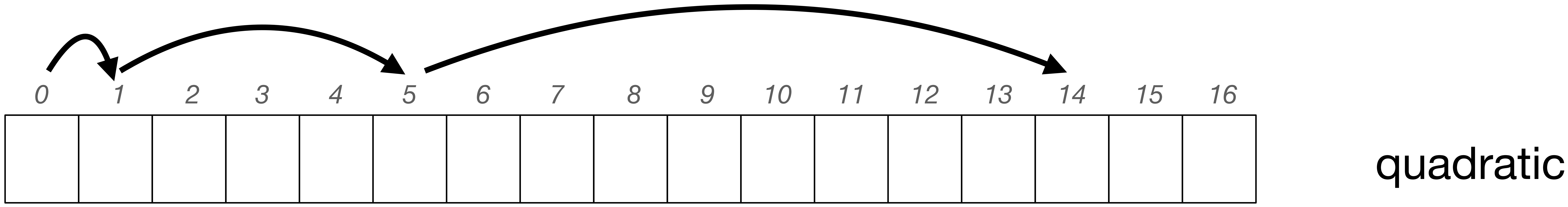
We've seen that linear probing is prone to primary clustering.

Quadratic probing is designed to eliminate primary clustering, but we've seen that quadratic probing is prone to secondary clustering.

Double hashing is designed to address both these problems.

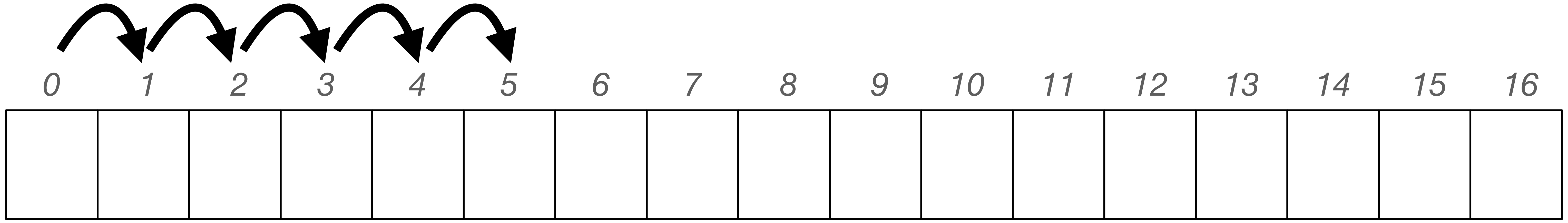
Quadratic probing vs linear probing



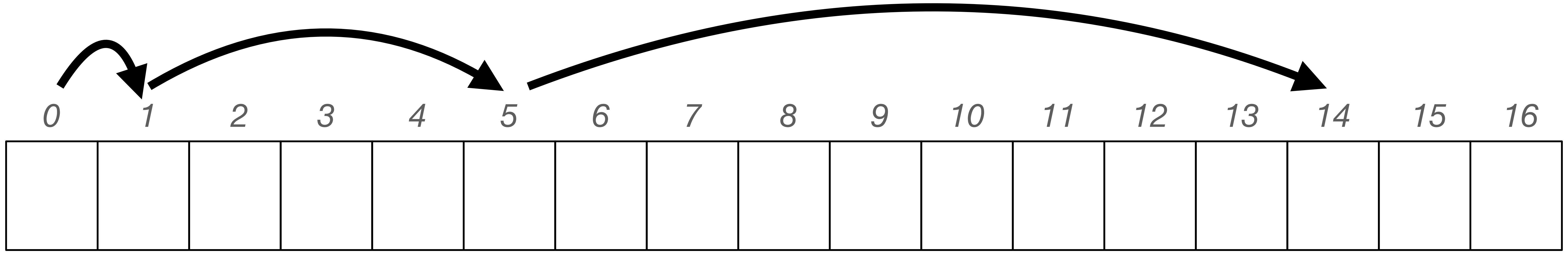


$$1^2 = 1,\ 2^2 = 4,\ 3^2 = 9,\ 4^2 = 16, \ldots$$

Quadratic probing vs linear probing vs double hashing

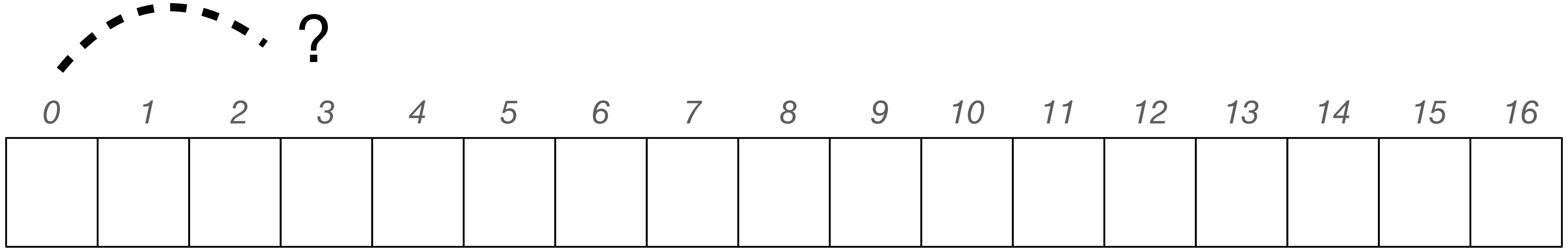


linear



quadratic

$$1^2 = 1,\ 2^2 = 4,\ 3^2 = 9,\ 4^2 = 16, \ldots$$



double hashing

Stride is calculated by a *secondary* hash function

The secondary hash function

- Should be different from hash function used to get the index
- Output of primary hash function and secondary hash function should be *pairwise independent* -- that is, *uncorrelated*
- Should return values in the range 1 to (table size - 1)
- Should distribute values as uniformly as possible within this range

The secondary hash function

- Calculate some number, *h*, (by Horner's method or some other method)
- Return *p* - (*h* mod *p*), where *p* is some prime number less than the table size

Primary and secondary hash functions

- The primary hash function gives us the starting point of our probe sequence
- The secondary hash function gives us the *stride* (if we need to probe)

The secondary hash function

	primary hash function	secondary hash function
key	f(x) = x % 17	g(x) = 13 - (x % 13)
0	0	13
1	1	12
2	2	11
3	3	10
...	...	...
25	8	1
...	...	...
32	15	7
...	...	...
163	10	6
...	...	...

The secondary hash function

	primary hash function	secondary hash function
key	f(x) = x % 17	g(x) = 13 - (x % 13)
0	0	13
17	0	9
34	0	5
51	0	1
68	0	10
...	...	...

The secondary hash function

	primary hash function	secondary hash function
key	f(x) = x % 17	g(x) = 13 - (x % 13)
0	0	13
17	0	9
34	0	5
51	0	1
68	0	10
...	...	...

Example: double hashing

Primary hash function
$f(x) = x \% 17$

Secondary hash function
$g(x) = 13 - (x \% 13)$

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*

Example: double hashing

Primary hash function
$f(x) = x \ \% \ 17$

Secondary hash function
$g(x) = 13 - (x \ \% \ 13)$

Insert 5 : f(x) = 5; g(x) = 8

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Example: double hashing

Primary hash function
$f(x) = x \% 17$

Secondary hash function
$g(x) = 13 - (x \% 13)$

Insert 5 : f(x) = 5; g(x) = 8

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
					5											

Example: double hashing

Primary hash function
$f(x) = x \% 17$

Secondary hash function
$g(x) = 13 - (x \% 13)$

Insert 56 : f(x) = 5; g(x) = 9

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
					5											

Example: double hashing

Primary hash function
$f(x) = x \ \% \ 17$

Secondary hash function
$g(x) = 13 - (x \ \% \ 13)$

Insert 56 : f(x) = 5; g(x) = 9

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
					5											

Example: double hashing

Primary hash function
$f(x) = x \% 17$

Secondary hash function
$g(x) = 13 - (x \% 13)$

Insert 56 : f(x) = 5; g(x) = 9

0	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
					5									56		

Example: double hashing

Primary hash function
$f(x) = x \% 17$

Secondary hash function
$g(x) = 13 - (x \% 13)$

Insert 39 : f(x) = 5; g(x) = 13

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
					5									56		

Example: double hashing

Primary hash function
$f(x) = x \% 17$

Secondary hash function
$g(x) = 13 - (x \% 13)$

Insert 39 : f(x) = 5; g(x) = 13

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
					5									56		

Example: double hashing

Primary hash function
$f(x) = x \% 17$

Secondary hash function
$g(x) = 13 - (x \% 13)$

Insert 39 : f(x) = 5; g(x) = 13

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	39				5									56		

Example: double hashing

Primary hash function
$f(x) = x \% 17$

Secondary hash function
$g(x) = 13 - (x \% 13)$

Insert 22 : f(x) = 5; g(x) = 4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	39				5									56		

Example: double hashing

Primary hash function
$f(x) = x \% 17$

Secondary hash function
$g(x) = 13 - (x \% 13)$

Insert 22 : f(x) = 5; g(x) = 4

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	39				5				22					56		

What just happened?

	primary hash function	secondary hash function
key	f(x) = x % 17	g(x) = 13 - (x % 13)
5	5	8 (not used)
56	5	9
39	5	13
22	5	4

Each of these inserts follows a different probe sequence

What's the probability of hash collisions having the same stride?

In order for hash collisions to have the same stride for their probe sequence, both the primary hash function and the secondary hash function would have to return the same value for two different keys.

In an ideal world, with "perfect" hash functions, the outputs would be distributed uniformly, just as if the hash functions were random. Then we'd have

$$\frac{1}{n} \times \frac{1}{n} = \frac{1}{n^2}$$

What's the probability of hash collisions having the same stride?

$$p > \frac{1}{n^2}$$

Comparison of CRPs

<table>
<tr><th>Linear probing</th><th>Quadratic probing</th><th>Double hashing</th><th>Separate chaining</th></tr>
<tr><td colspan="3">On collisions we probe</td><td>On collisions
we extend the chain</td></tr>
<tr><td colspan="3">Fixed upper limit on number of objects we can insert
(size of hash table)</td><td>Only limited by memory / system constrants</td></tr>
<tr><td>Fixed stride (typically 1)</td><td>Stride changes on each step ($step^2$)</td><td>Fixed stride calculated by second hash</td><td>n/a</td></tr>
<tr><td>Prone to primary clustering</td><td>Prone to secondary clustering</td><td>Reduces clustering</td><td>Clustering does not occur</td></tr>
</table>

Double hashing: summary

- We only allow a single object at a given index.
- Upon hash collisions, we probe our hash table, one step at a time, with a stride that's calculated by a second hash function.
- Because we use a second hash function, the stride *depends on the data*. This makes it very unlikely that two insertions, with the same hash value for the first index, would follow the same probe sequence. They'd have to have, in effect, two concurrent hash collisions!
- Double hashing has a fixed limit on the number of objects we can insert into our hash table.

Questions

- We have two basic strategies for hash collision: chaining and probing (linear probing, quadratic probing, and double hashing are of the latter type).
  - Which do you think uses more memory?
  - Which do you think is faster?
  - How would you calculate their complexities?