Hash Tables: Separate Chaining

CS 124 / Department of Computer Science

What have we left out?

There are quite a few implementation details we've left out but the most important thing we've left out of our discussion so far is: what to do when hashing two di!erent keys yields the same value? This is a challenge for hash tables called "*hash collisions*" or just "*collisions.*""

We'll learn more about collisions and what to do when they occur in future lectures. It turns out there are many di!

What have we left out?

There are quite a few implementation details we've left out but the most important thing we've left out of our discussion so far is: what to do when hashing two di!erent keys yields the same value? This is a challenge for hash tables called "*hash collisions*" or just "*collisions.*""

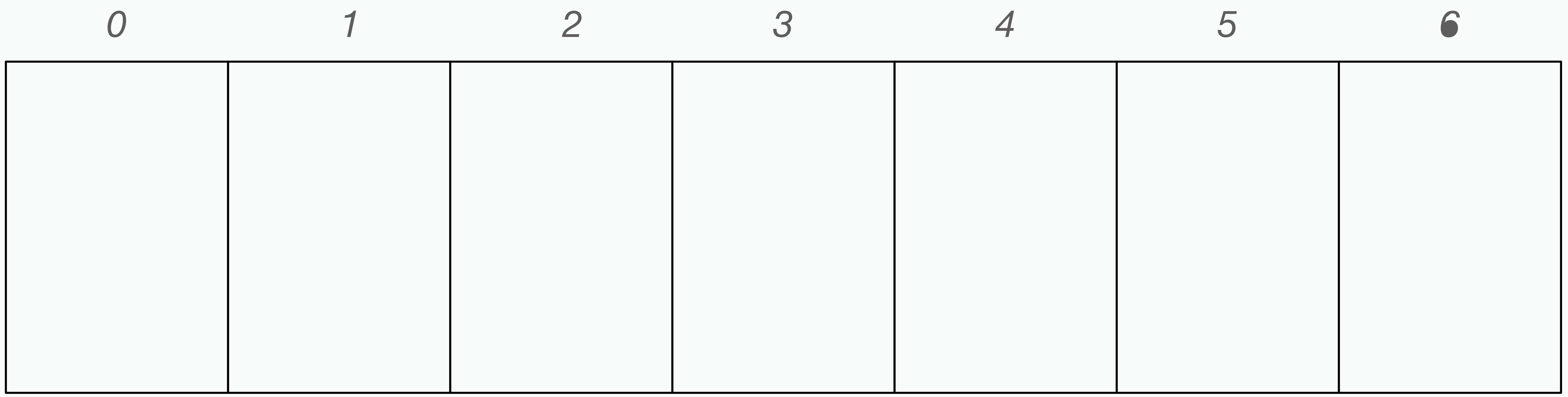
We'll learn more about collisions and what to do when they occur in future lectures. It turns out there are many di!erent strategies -- called "*collision resolution policies*," and we'll look at some of the most common ones.

Collisions are inevitable

Separate Chaining

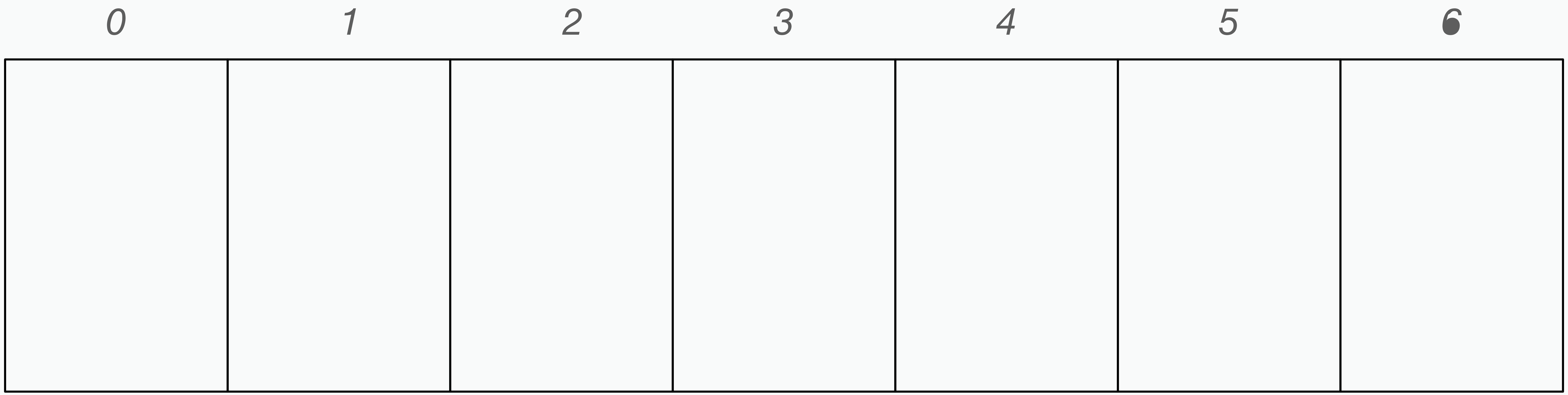
Instead of holding just one object, allow elements in our hash table to hold *more than one object*.

Separate Chaining



Separate Chaining

Hash function:#
$f(x) = x \bmod 7$



Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

13 : 13 mod 7 = 6

0	*1*	*2*	*3*	*4*	*5*	*6*

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

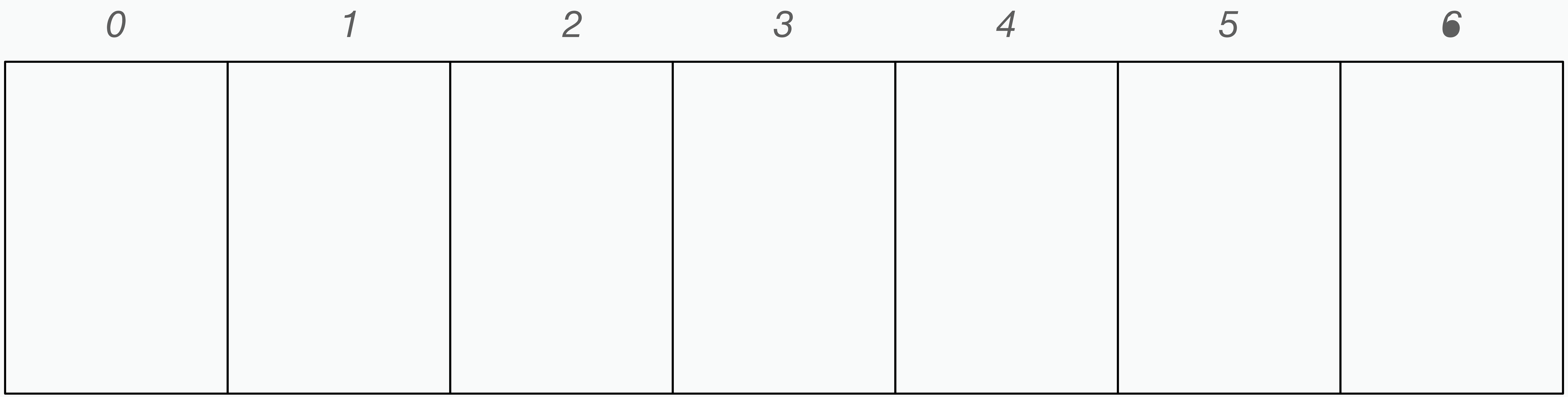
179 : 179 mod 7 = 4

0	1	2	3	4	5	6
						13

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

179 : 179 mod 7 = 4



Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

114 : 114 mod 7 = 2

0	1	2	3	4	5	6
				179		13

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

114 : 114 mod 7 = 2

0	1	2	3	4	5	6
		114		179		13

Separate Chaining

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

5 : 5 mod 7 = 5

0	1	2	3	4	5	6
		114		179	5	13

Separate Chaining

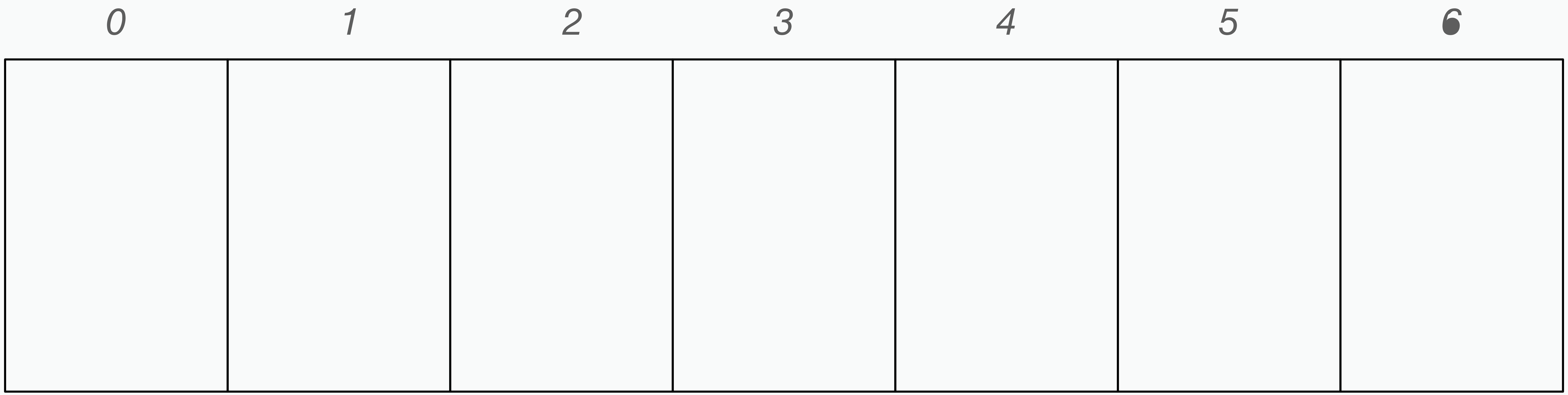
Hash function:#
$f(x) = x \bmod 7$

20 : 20 mod 7 = 6

0	1	2	3	4	5	6
		114		179	5	13

Separate Chaining

Separate Chaining



Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

73 : 73 mod 7 = 3

0	*1*	*2*	*3*	*4*	*5*	*6*
		114	73	179	5	13# 20

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

180 : 180 mod 7 = 5

0	*1*	*2*	*3*	*4*	*5*	*6*
		114	73	179	5	13# 20

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

48 : 48 mod 7 = 6

0	1	2	3	4	5	6
		114	73	179	5# 180	13# 20

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

48 : 48 mod 7 = 6

0	1	2	3	4	5	6
		114	73	179	5# 180	13# 20# 48

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

46 : 46 mod 7 = 4

0	1	2	3	4	5	6
		114	73	179	5# 180	13# 20# 48

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

46 : 46 mod 7 = 4

0	1	2	3	4	5	6
		114	73	179# 46	5# 180	13# 20# 48

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

88 : 88 mod 7 = 4

0	1	2	3	4	5	6
		114	73	179# 46	5# 180	13# 20# 48

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

88 : 88 mod 7 = 4

0	1	2	3	4	5	6
		114	73	179# 46# 88	5# 180	13# 20# 48

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

196 : 196 mod 7 = 0

0	1	2	3	4	5	6
		114	73	179# 46# 88	5# 180	13# 20# 48

Separate Chaining

Hash function:#
$f(x) = x \bmod 7$

196 : 196 mod 7 = 0

0	1	2	3	4	5	6
196		114	73	179# 46# 88	5# 180	13# 20# 48

Separate Chaining

Insertion takes constant time"

$ Calculating hash takes constant time"

$ Inserting into vector takes constant time"

But what about duplicate values?"

What about find and remove?

Separate Chaining

Insertion takes constant time"

$ Calculating hash takes constant time"

$ Inserting into vector takes constant time"

But what about duplicate values? We have to search through the bucket."

What about find and remove?

Separate Chaining

Insertion takes constant time"

$ Calculating hash takes constant time"

$ Inserting into vector takes constant time"

But what about duplicate values? We have to search through the bucket."

What about find and remove? We have to search through the bucket.

Separate Chaining

If we have a table of size *b* (*b* for the number of buckets) and we have n objects we wish to store, then on average a bucket will store *n* / *b* objects."

If we use linear search to check to see if an object is already in our bucket before insertion that's *O* (*n* / *b*).

Separate Chaining

If we have a table of size *b* (*b* for the number of buckets) and we have n objects we wish to store, then on average a bucket will store *n* / *b* objects."

If we use linear search to check to see if an object is already in our bucket before insertion that's O (*n* / *b*). "

We also have to search through a bucket when finding or removing."

Note that the book uses a linked list for buckets; here we're using vectors. But this doesn't change the fact that in either case we still need to search.

Summary

$ Separate chaining uses a vector of vectors (or a vector of linked lists) to handle collisions."

$ Objects with the same index calculated from the hash function wind up in the same bucket (again, whether it's a vector or linked list)."

$ This requires us to search on each insertion, find, or remove operation."

$ Separate chaining is easy to implement.

Questions

- If we sorted our buckets, we could improve search time to $O(\log(n / b))$ using binary search or $O(\log \log(n / b))$ using interpolation search. Does it make sense to do this? Why or why not?"

- Can you think of other ways we might handle collisions that don't require the use of buckets?