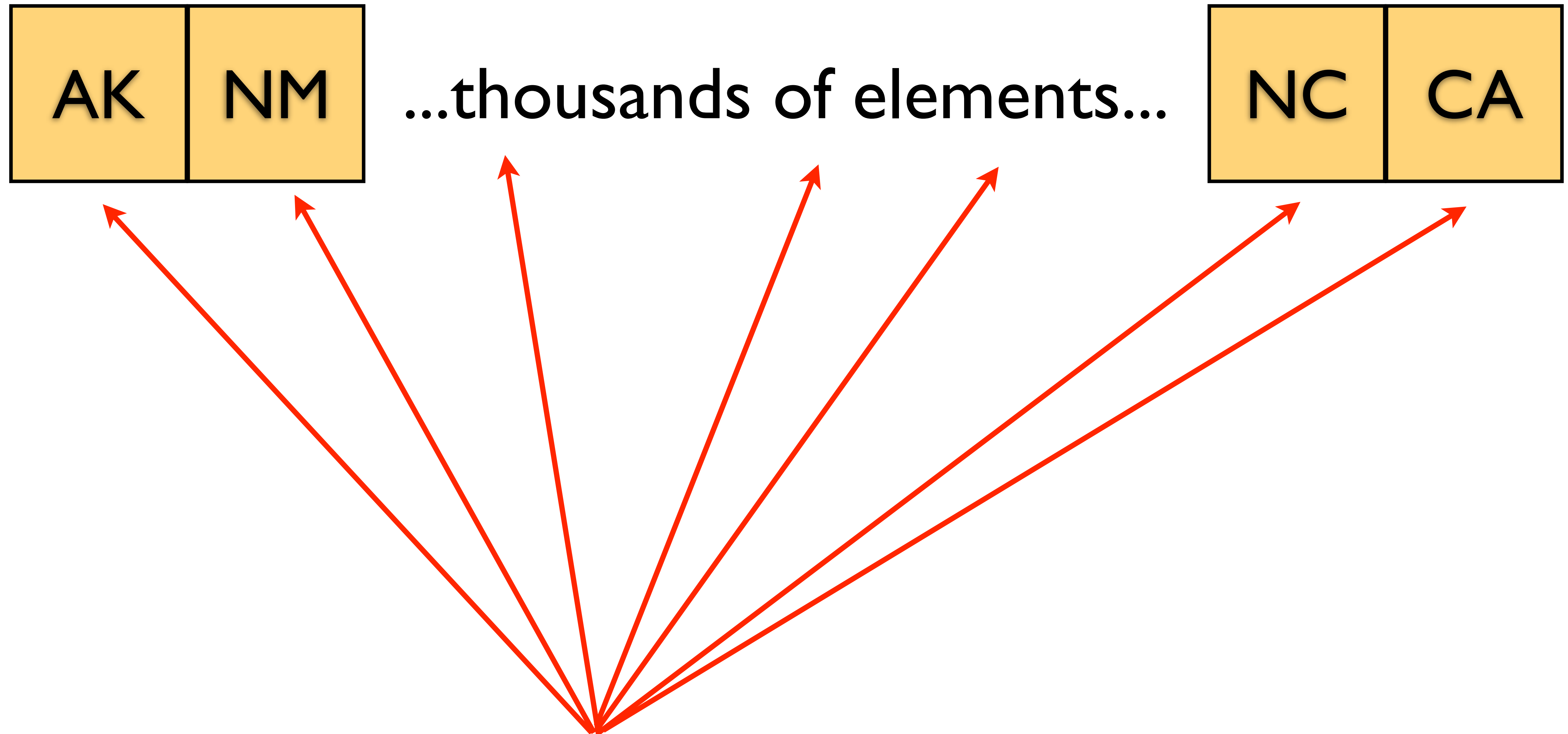
Hashing!
(At long last)

So you want to write a set...

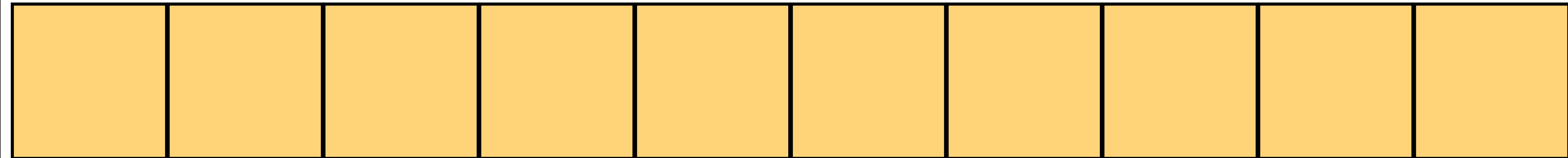
And you want to ask “Is DC in this set?”

So you want to write a set...



And you want to ask "Is DC in this set?"

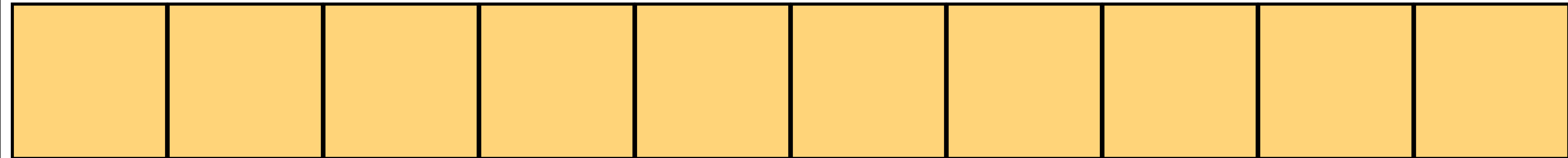
HashCode: "I'm near index *k*"



0 1 2 3 4 5 6 7 8 9

At most a constant distance *from index* k, *in fact.*

HashCode: “I’m near index *k*”



0 1 2 3 4 5 6 7 8 9

AK: 17

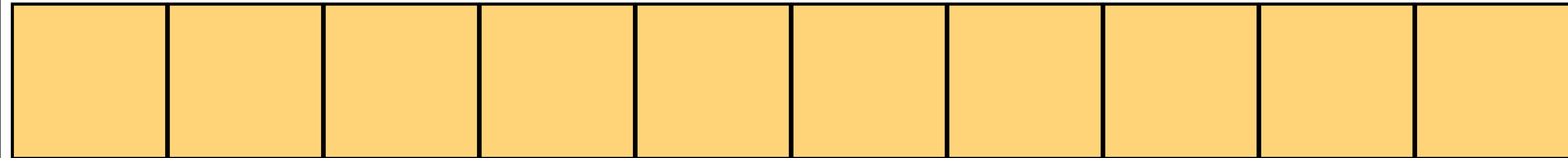
NC: 31

NY: 729

TX: 312425

WA: 53623238

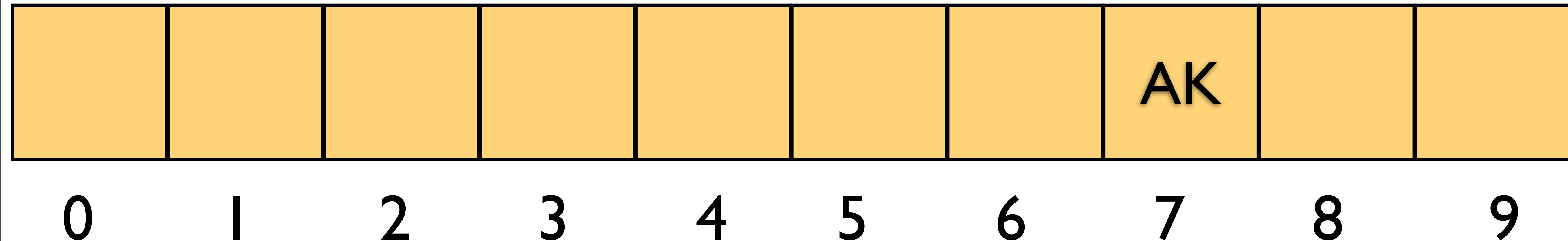
HashCode: “I’m near index *k*”



AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

```
table[hashCode % table.length] = v;
```

HashCode: “I’m near index *k*”



AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

```
table[hashCode % table.length] = v;
```

HashCode: "I'm near index *k*"

0	1	2	3	4	5	6	7	8	9
	NC						AK		

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

```
table[hashCode % table.length] = v;
```

HashCode: “I’m near index *k*”

	NC						AK		NY
0	1	2	3	4	5	6	7	8	9

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

```
table[hashCode % table.length] = v;
```

HashCode: "I'm near index *k*"

	NC				TX		AK		NY
0	1	2	3	4	5	6	7	8	9

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

```
table[hashCode % table.length] = v;
```

HashCode: "I'm near index *k*"

0	1	2	3	4	5	6	7	8	9
	NC				TX		AK	WA	NY

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

```
table[hashCode % table.length] = v;
```

HashCode: "I'm near index *k*"

0	1	2	3	4	5	6	7	8	9
	NC				TX		AK	WA	NY

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

What's the Big O of lookup?

```
table[hashCode % table.length] = v;
```

HashCode: “I’m near index *k*”

NC				TX		AK	WA	NY
0 1 2 3 4 5 6 7 8 9

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75

```
table[hashCode % table.length] = v;
```

HashCode: "I'm near index *k*"

	NC				TX	OH	AK	WA	NY
0	1	2	3	4	5	6	7	8	9

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75

```
table[hashCode % table.length] = v;
```

HashCode: "I'm near index *k*"

	NC	OR			TX	OH	AK	WA	NY
0	1	2	3	4	5	6	7	8	9

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75

```
table[hashCode % table.length] = v;
```

HashCode: "I'm near index *k*"

	NC	OR	FL		TX	OH	AK	WA	NY
0	1	2	3	4	5	6	7	8	9

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75

```
table[hashCode % table.length] = v;
```

HashCode: "I'm near index *k*"

SC	NC	OR	FL		TX	OH	AK	WA	NY
0	1	2	3	4	5	6	7	8	9

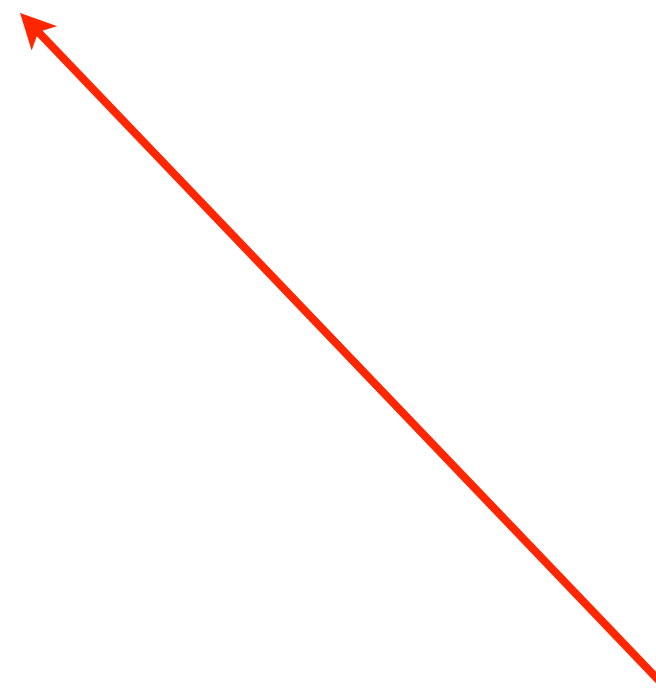
AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75

```
table[hashCode % table.length] = v;
```

HashCode: “I’m near index *k*”

SC	NC	OR	FL		TX	OH	AK	WA	NY
0	1	2	3	4	5	6	7	8	9

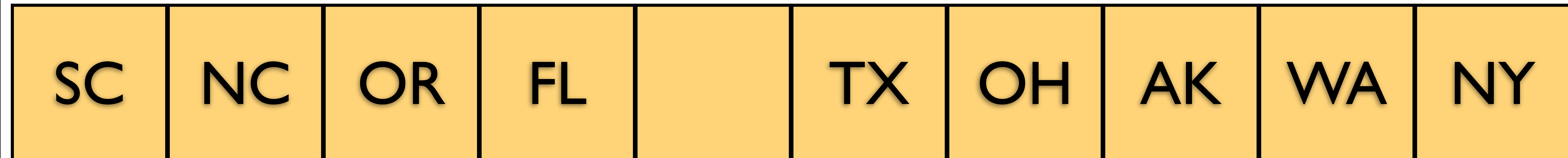


AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75

```
table[hashCode % table.length] = v;
```

HashCode: “I’m near index *k*”



0 1 2 3 4 5 6 7 8 9

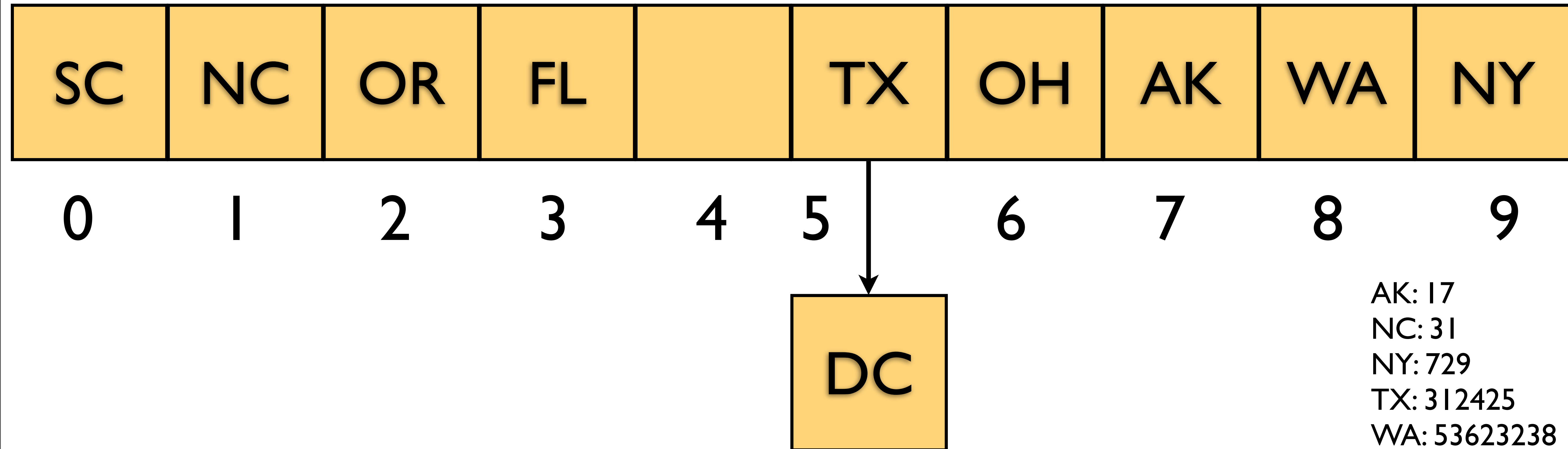
DC

AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238

OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75

```
table[hashCode % table.length] = v;
```

HashCode: "I'm near index *k*"

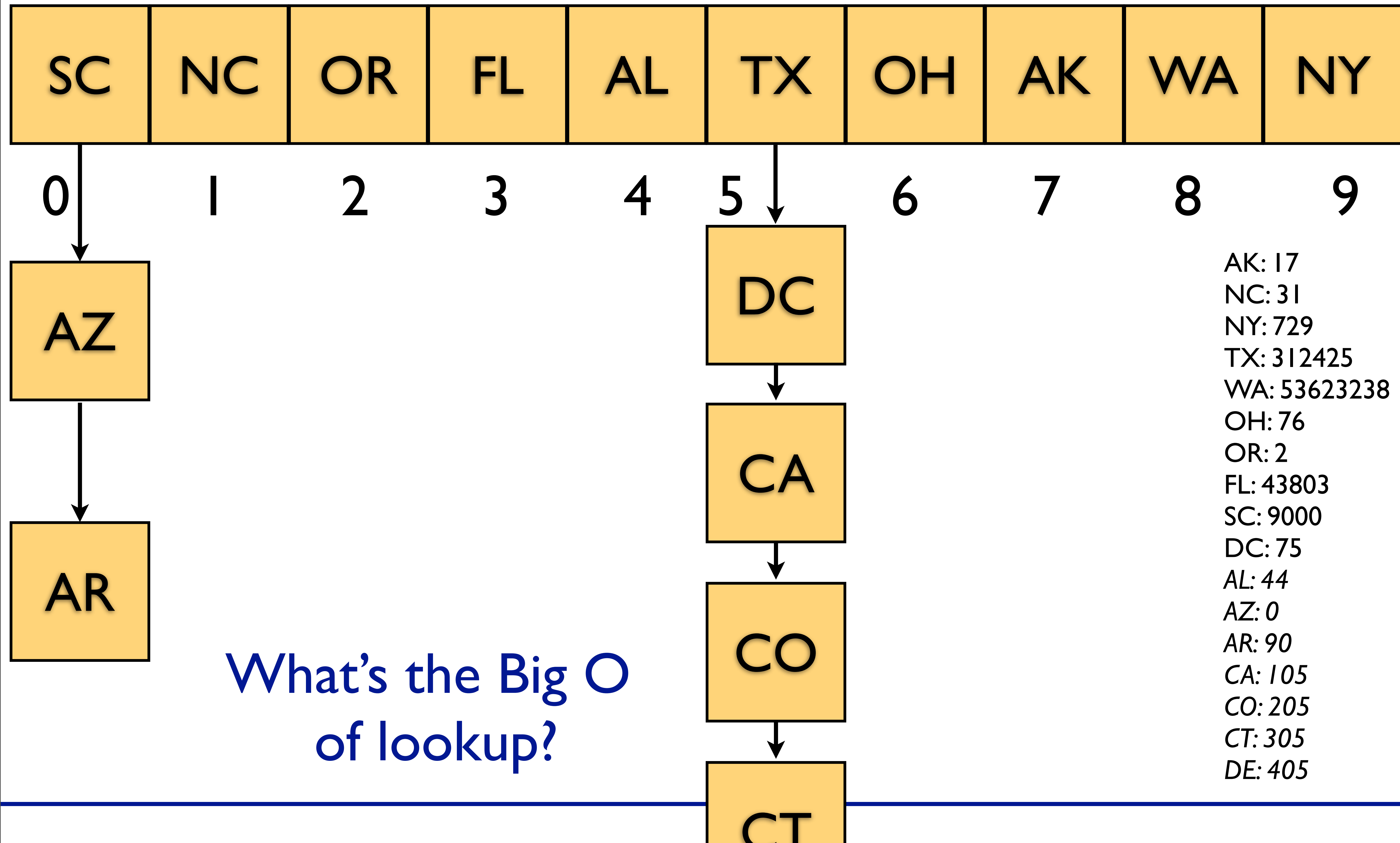


AK: 17
NC: 31
NY: 729
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

Draw the resulting table.

```
table[hashCode % table.length] = v;
```

HashCode: “I’m near index *k*”

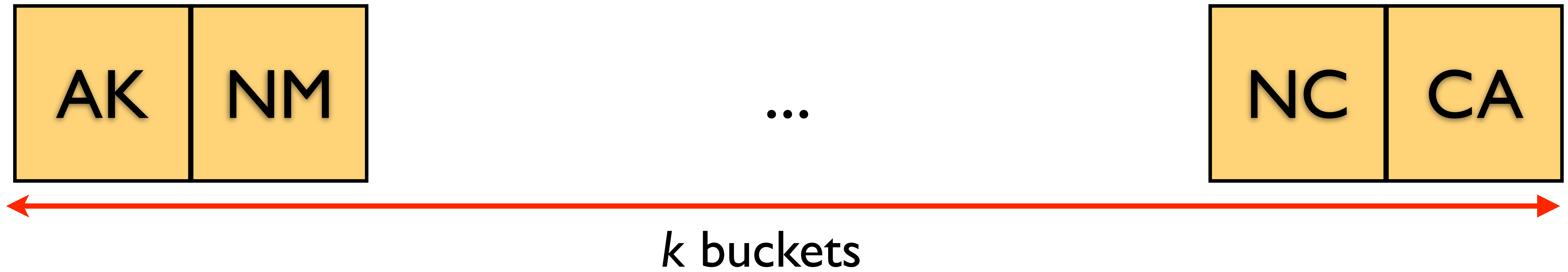


What do we do about collisions?

Pausing for a moment.

http://goo.gl/qB9DP

Back to collisions

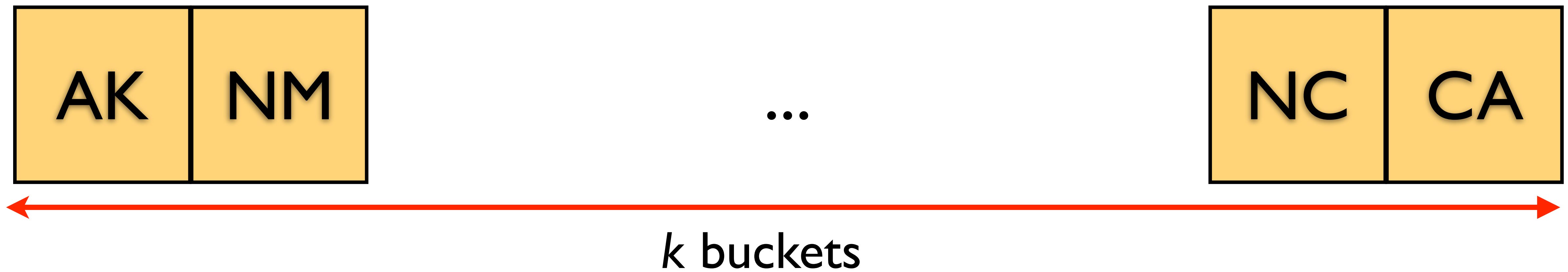


After adding n elements to the table, we have $\approx n/k$ elements in each bucket.

$O(n/k)$

Aside: this can be formalized. If we assume that our hash function is drawing from the buckets uniformly at random, we will have n/k elements per bucket in expectation. As you know, actually using random won't work, so this is a slightly optimistic analysis.

Back to collisions



After adding n elements to the table, we have $\approx n/k$ elements in each bucket.

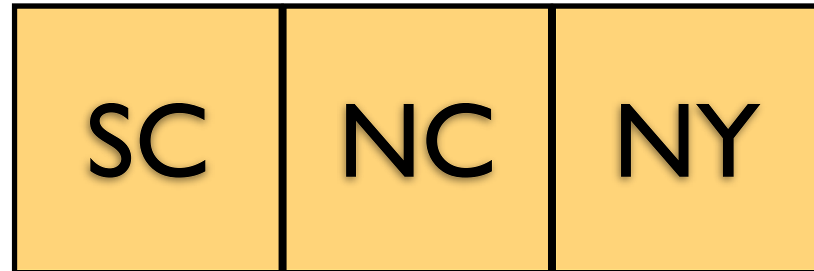
$O(n/k)$

We control k!

Load factor

Aside: this can be formalized. If we assume that our hash function is drawing from the buckets uniformly at random, we will have n/k elements per bucket in expectation. As you know, actually using random won't work, so this is a slightly optimistic analysis.

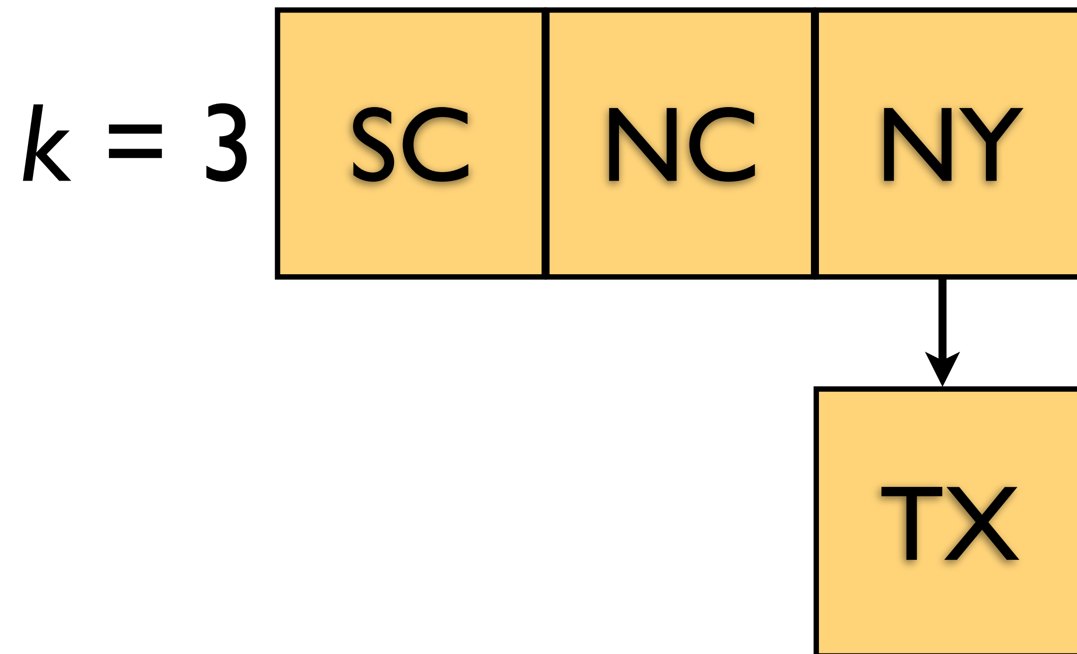
Rehashing

$k = 3$ 

$L = 1.0$

SC: 9
NC: 37
NY: 98
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

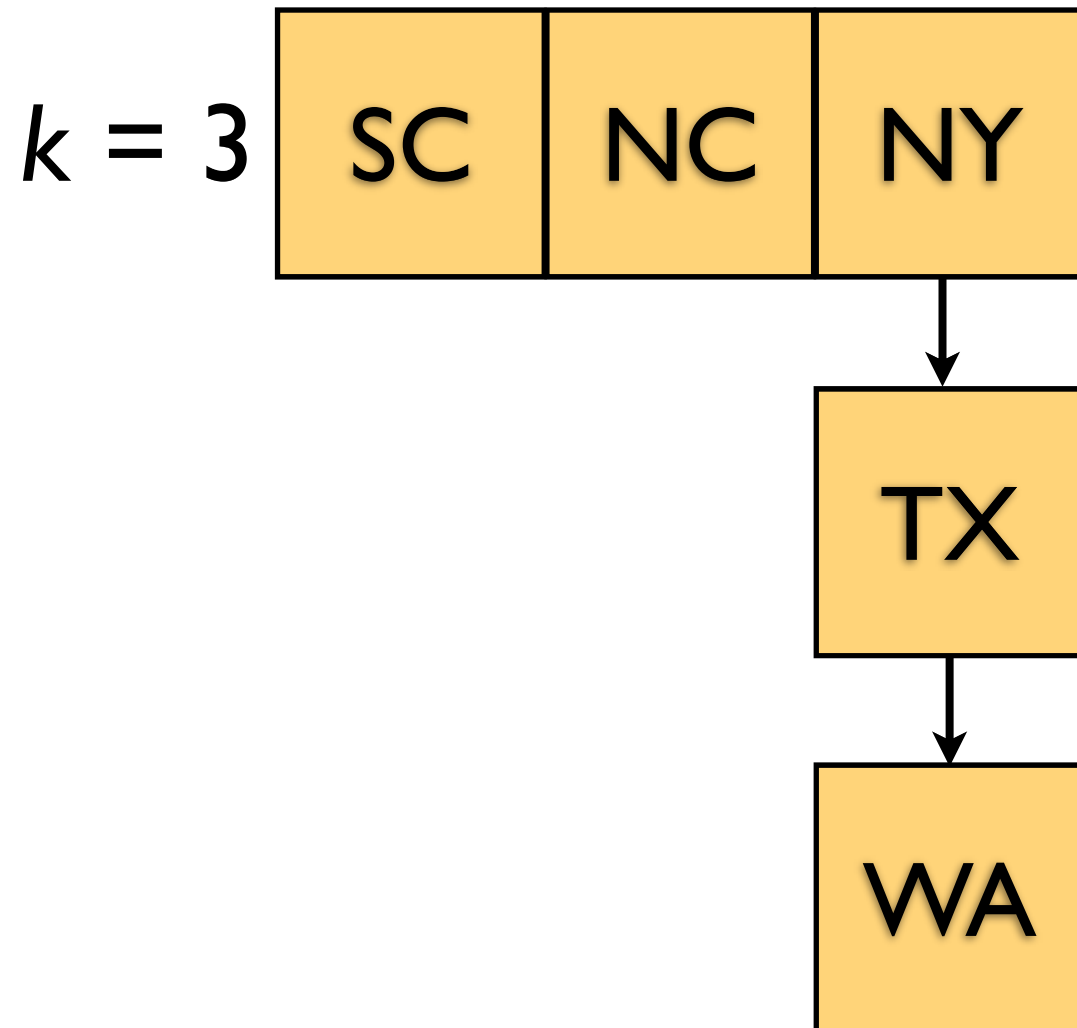
Rehashing



L = *4/3*

SC: 9
NC: 37
NY: 98
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

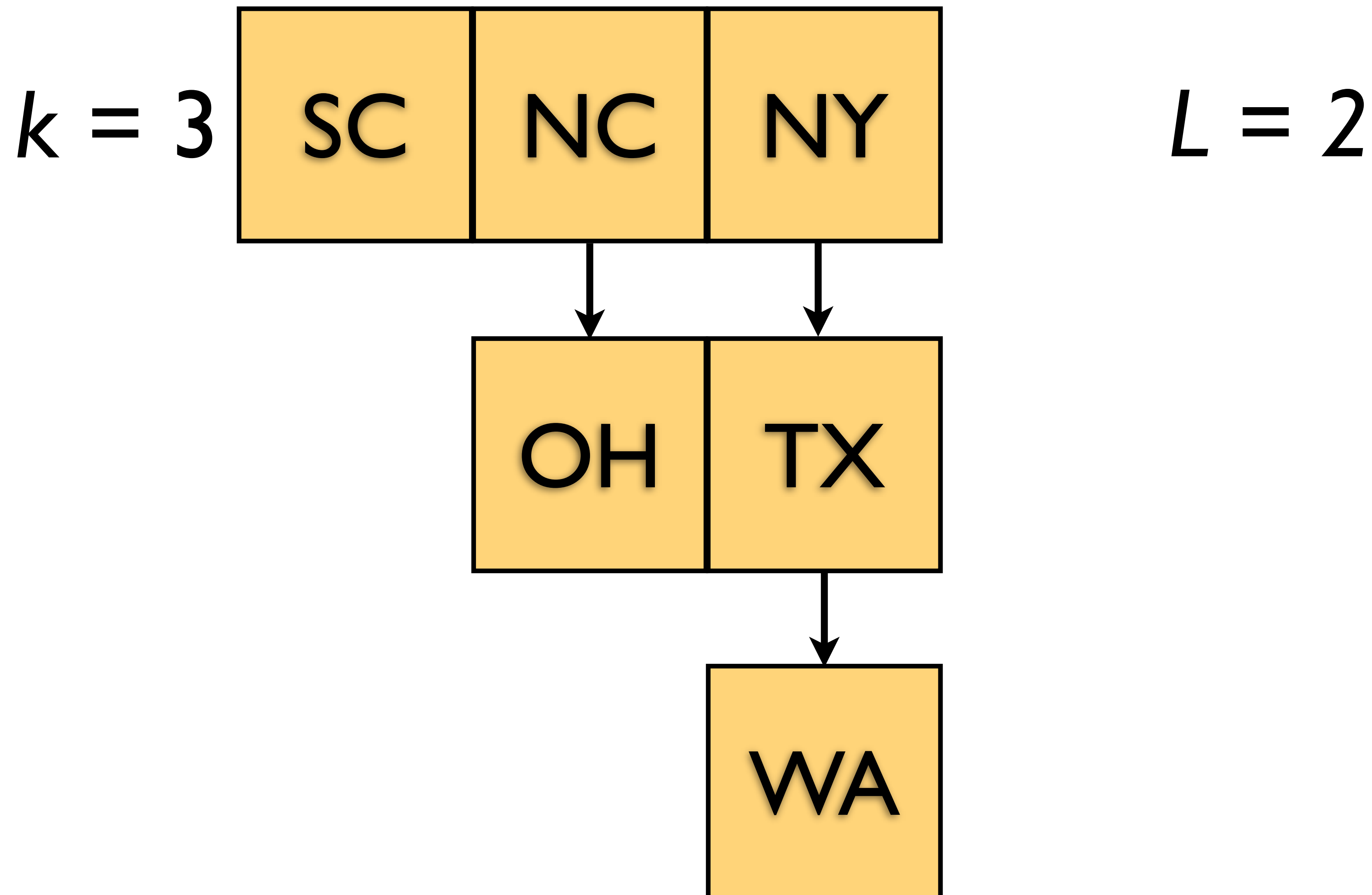
Rehashing



$L = 5/3$

SC: 9
NC: 37
NY: 98
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

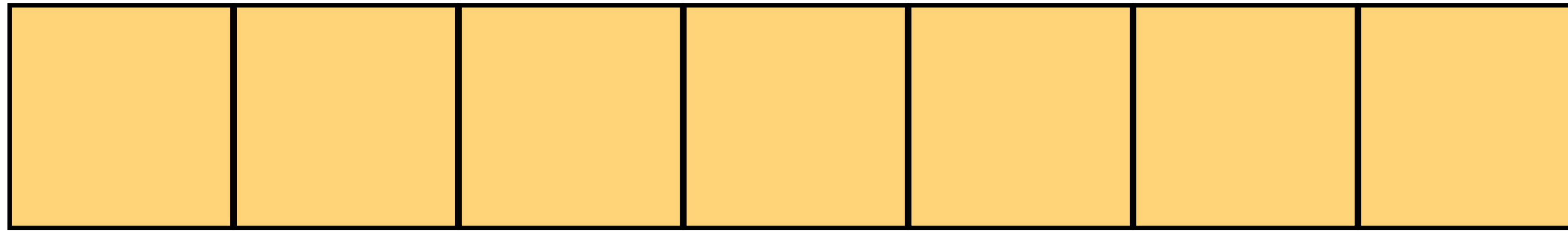
Rehashing

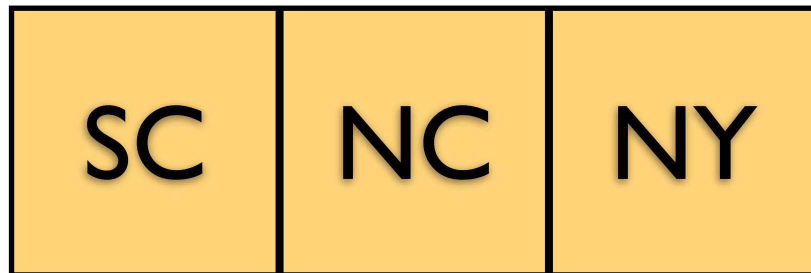


$L = 2$

SC: 9
NC: 37
NY: 98
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

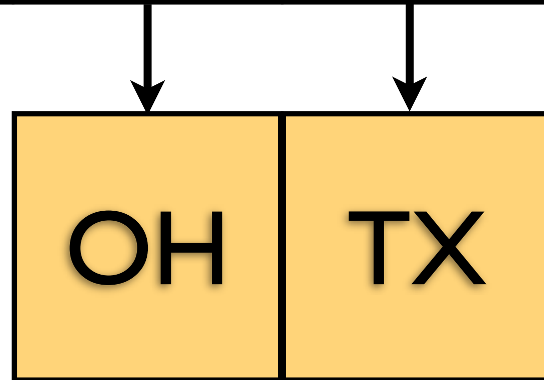
Rehashing

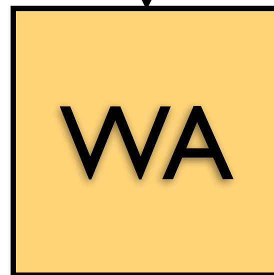




$k = 3$

$L = 2$





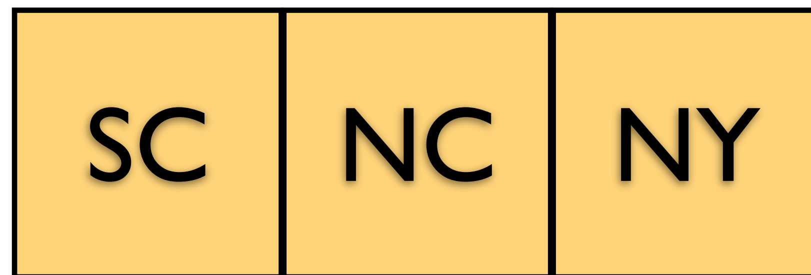
SC: 9
NC: 37
NY: 98
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

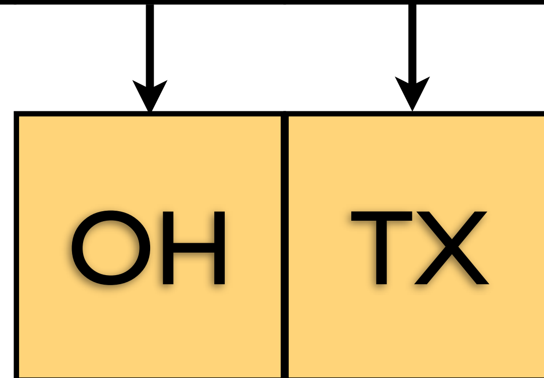
```
table[hashCode % table.length] = v;
```

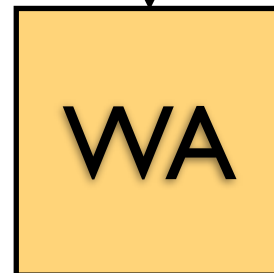
A thoughtful person would ponder why I'm using prime-number-sized tables, and not powers of two.

Rehashing

$k = 3$







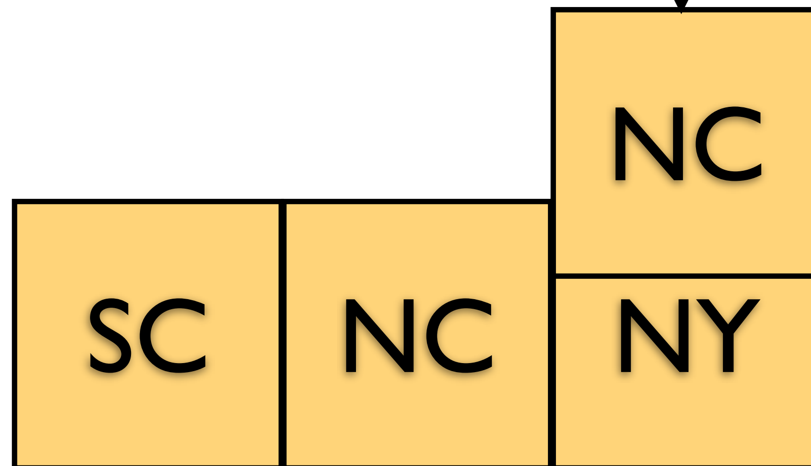
$L = 2$

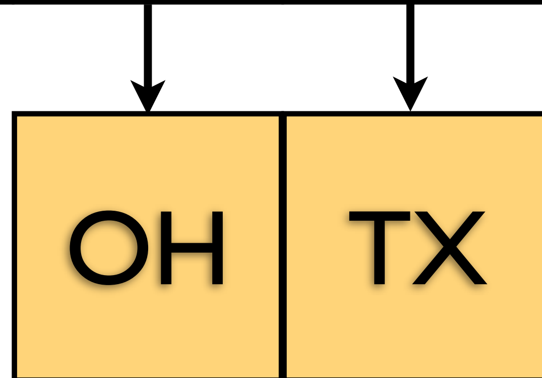
SC: 9
NC: 37
NY: 98
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

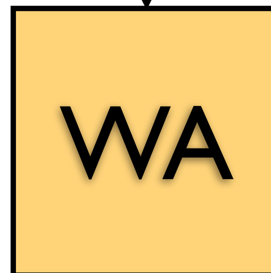
```
table[hashCode % table.length] = v;
```

A thoughtful person would ponder why I'm using prime-number-sized tables, and not powers of two.

Rehashing







$k = 3$

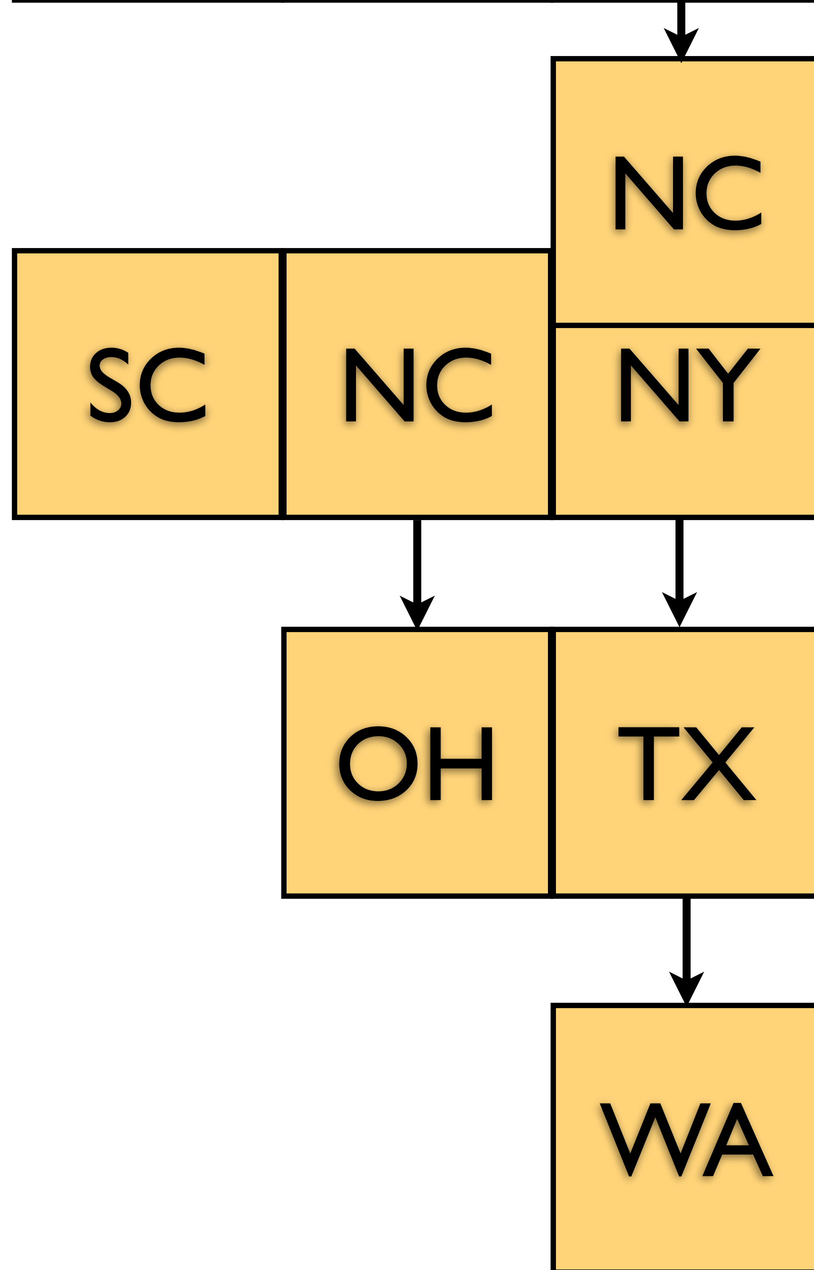
$L = 2$

SC: 9
NC: 37
NY: 98
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

```
table[hashCode % table.length] = v;
```

A thoughtful person would ponder why I'm using prime-number-sized tables, and not powers of two.

Rehashing



$k = 3$

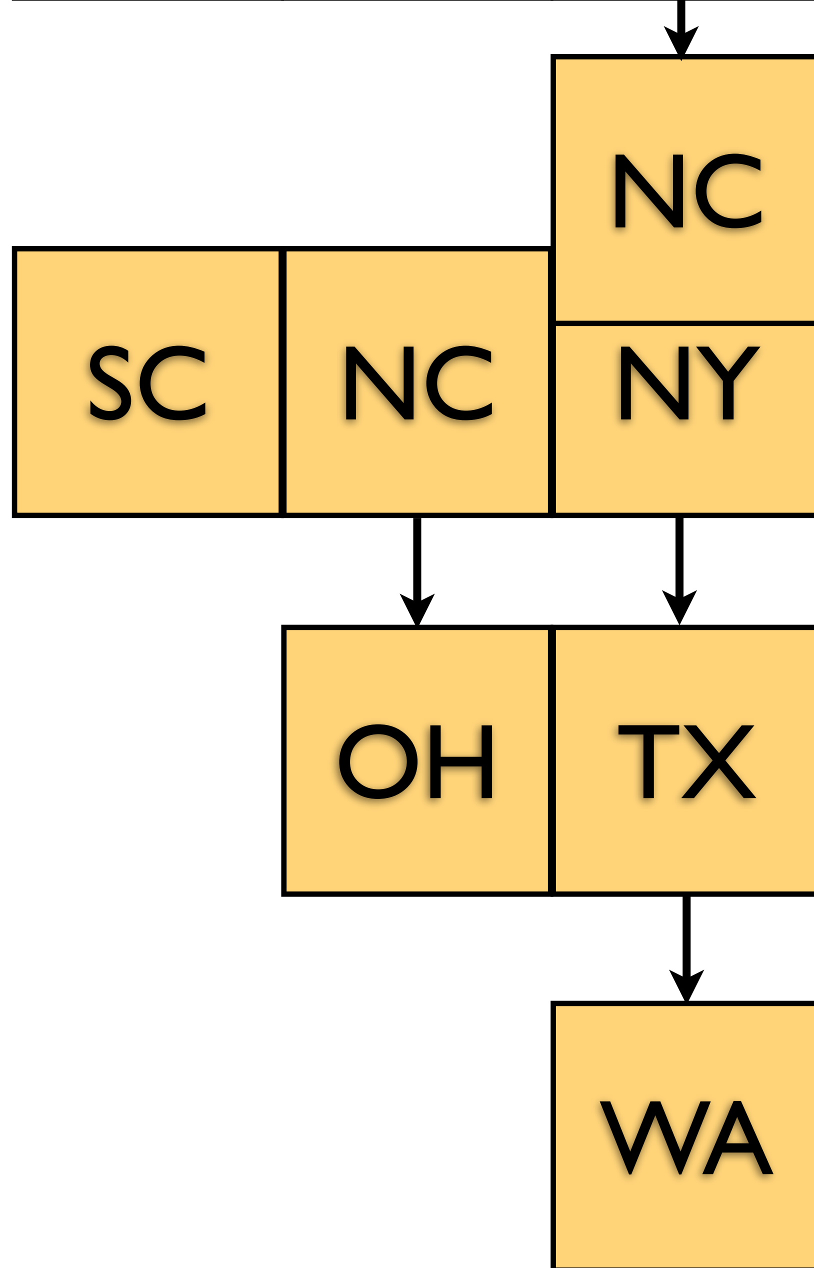
$L = 2$

SC: 9
NC: 37
NY: 98
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

```
table[hashCode % table.length] = v;
```

A thoughtful person would ponder why I'm using prime-number-sized tables, and not powers of two.

Rehashing



$k = 3$

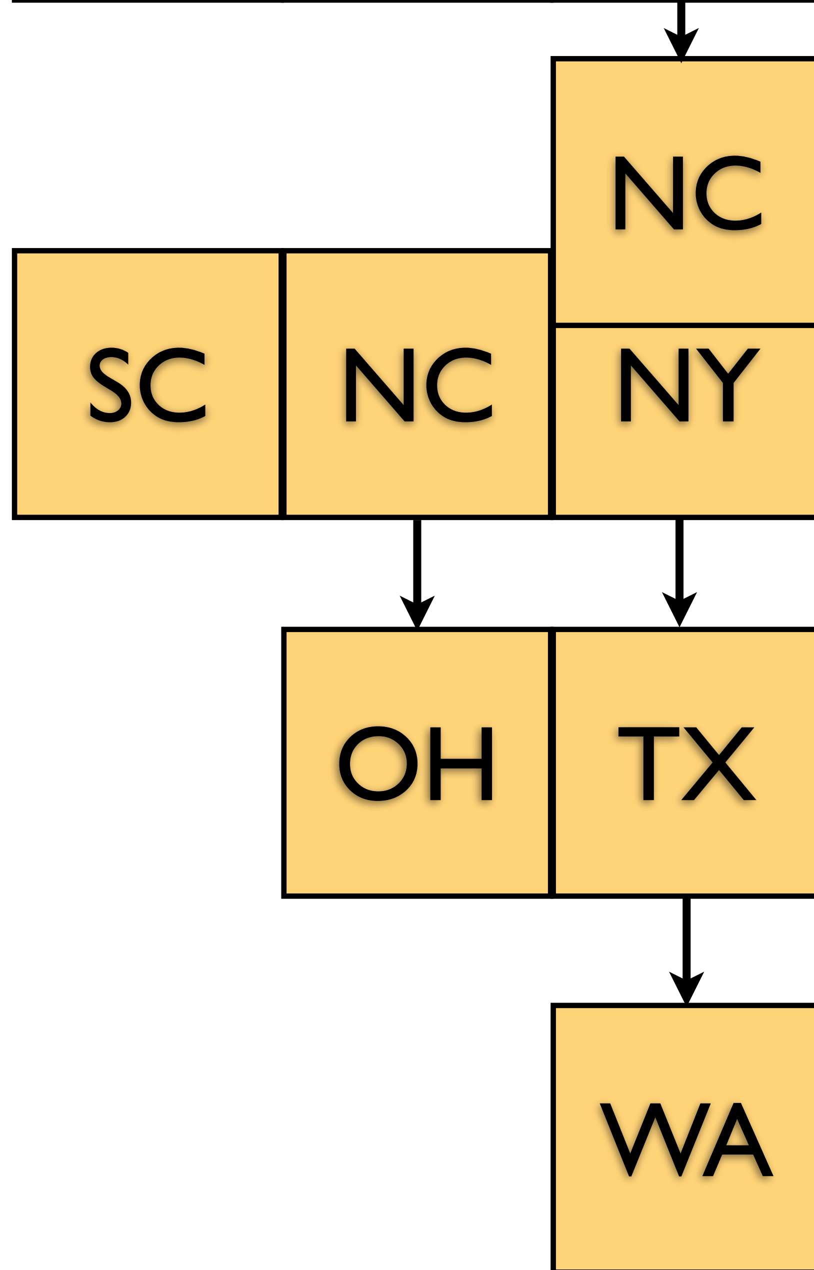
$L = 2$

SC: 9
NC: 37
NY: 98
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

```
table[hashCode % table.length] = v;
```

A thoughtful person would ponder why I'm using prime-number-sized tables, and not powers of two.

Rehashing



$k = 3$

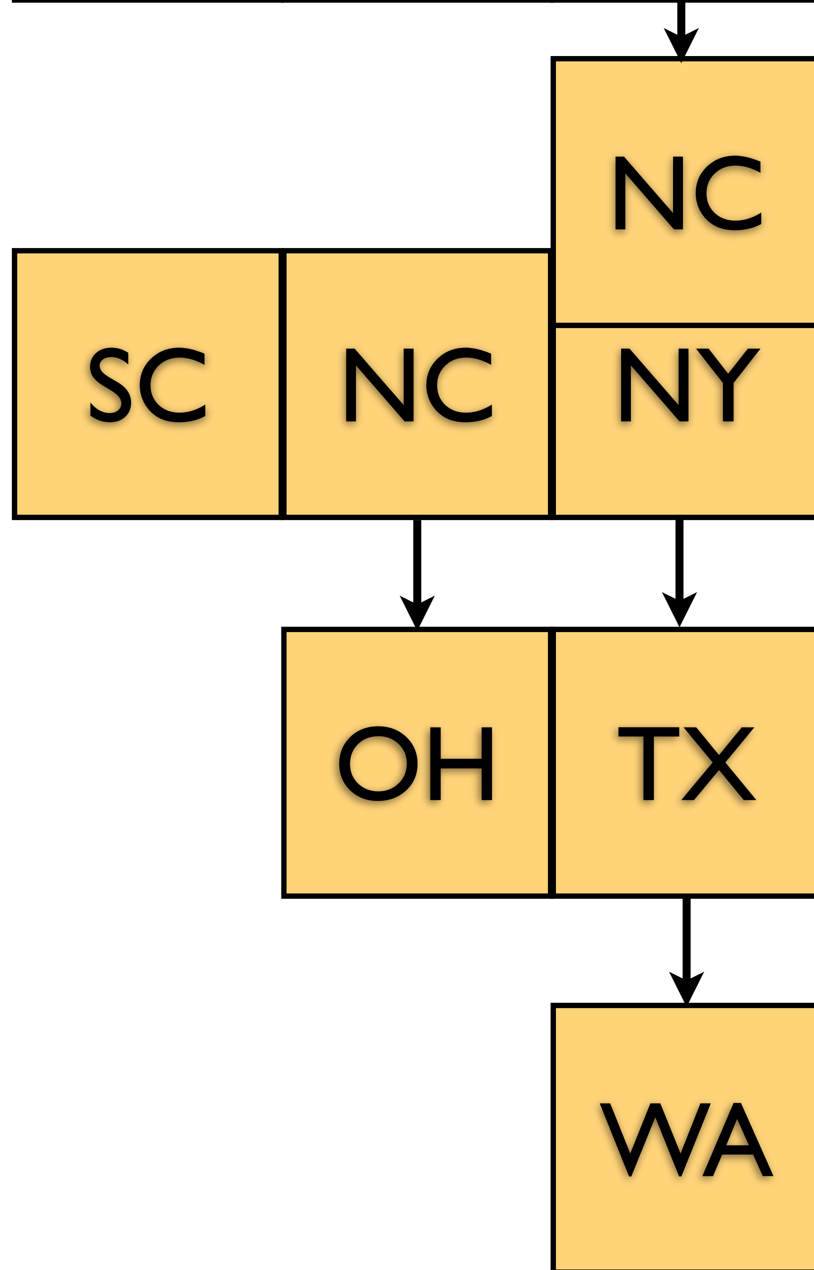
$L = 2$

SC: 9
NC: 37
NY: 98
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

```
table[hashCode % table.length] = v;
```

A thoughtful person would ponder why I'm using prime-number-sized tables, and not powers of two.

Rehashing



$k = 3$

$L = 2$

SC: 9
NC: 37
NY: 98
TX: 312425
WA: 53623238
OH: 76
OR: 2
FL: 43803
SC: 9000
DC: 75
AL: 44
AZ: 0
AR: 90
CA: 105
CO: 205
CT: 305
DE: 405

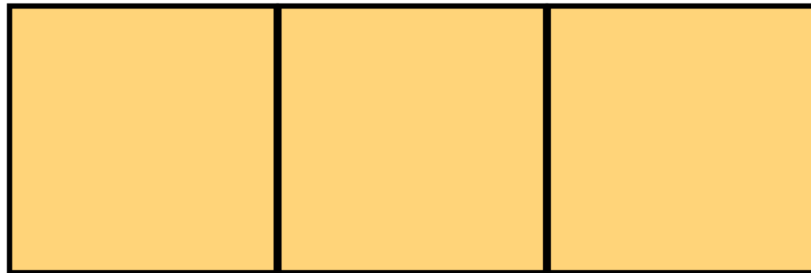
```
table[hashCode % table.length] = v;
```

A thoughtful person would ponder why I'm using prime-number-sized tables, and not powers of two.

Load Factor

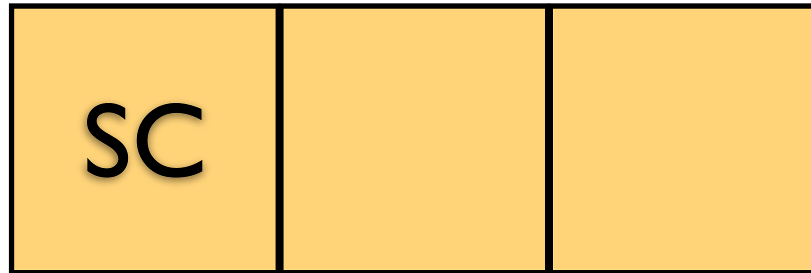
http://goo.gl/fI7wt

Not just chains

$k = 3$ 

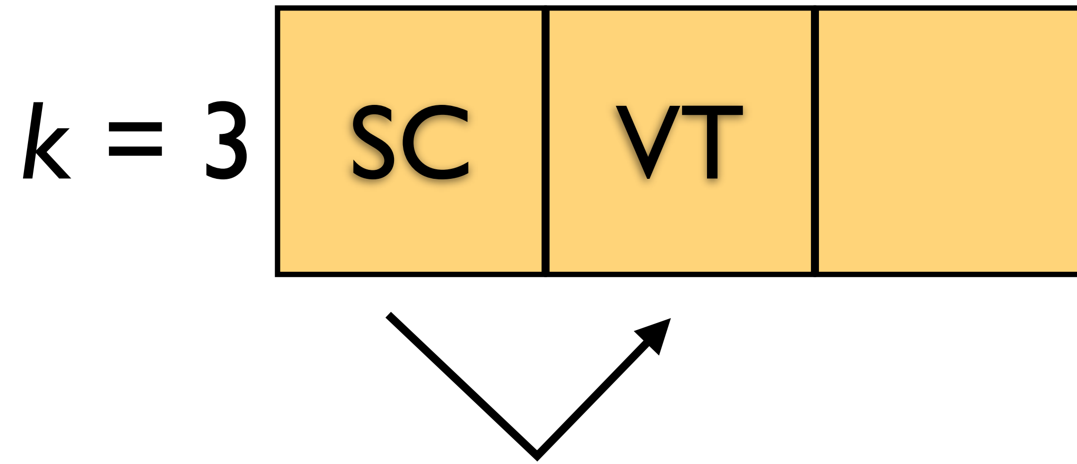
SC: 9
VT: 9000
DC: 99

Not just chains



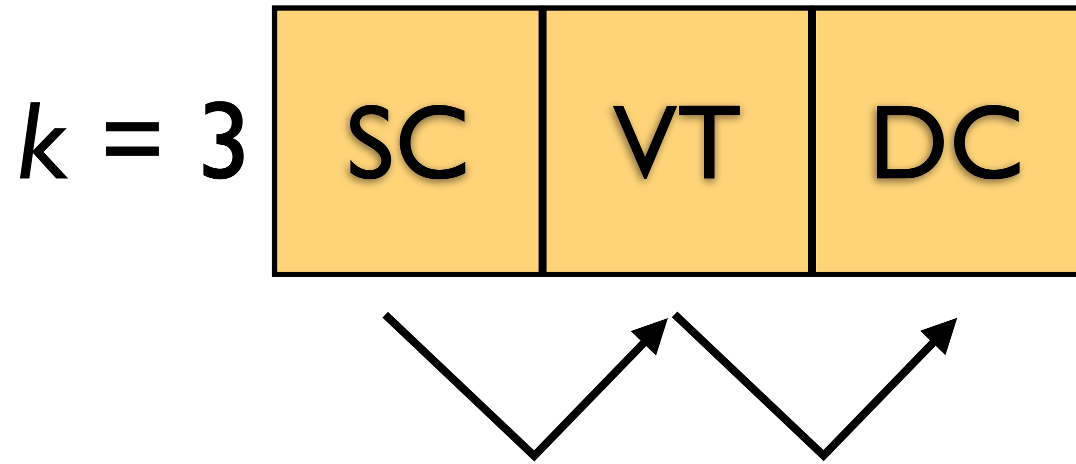
SC: 9
VT: 9000
DC: 99

Not just chains



SC: 9
VT: 9000
DC: 99

Not just chains: *probing*



SC: 9
VT: 9000
DC: 99