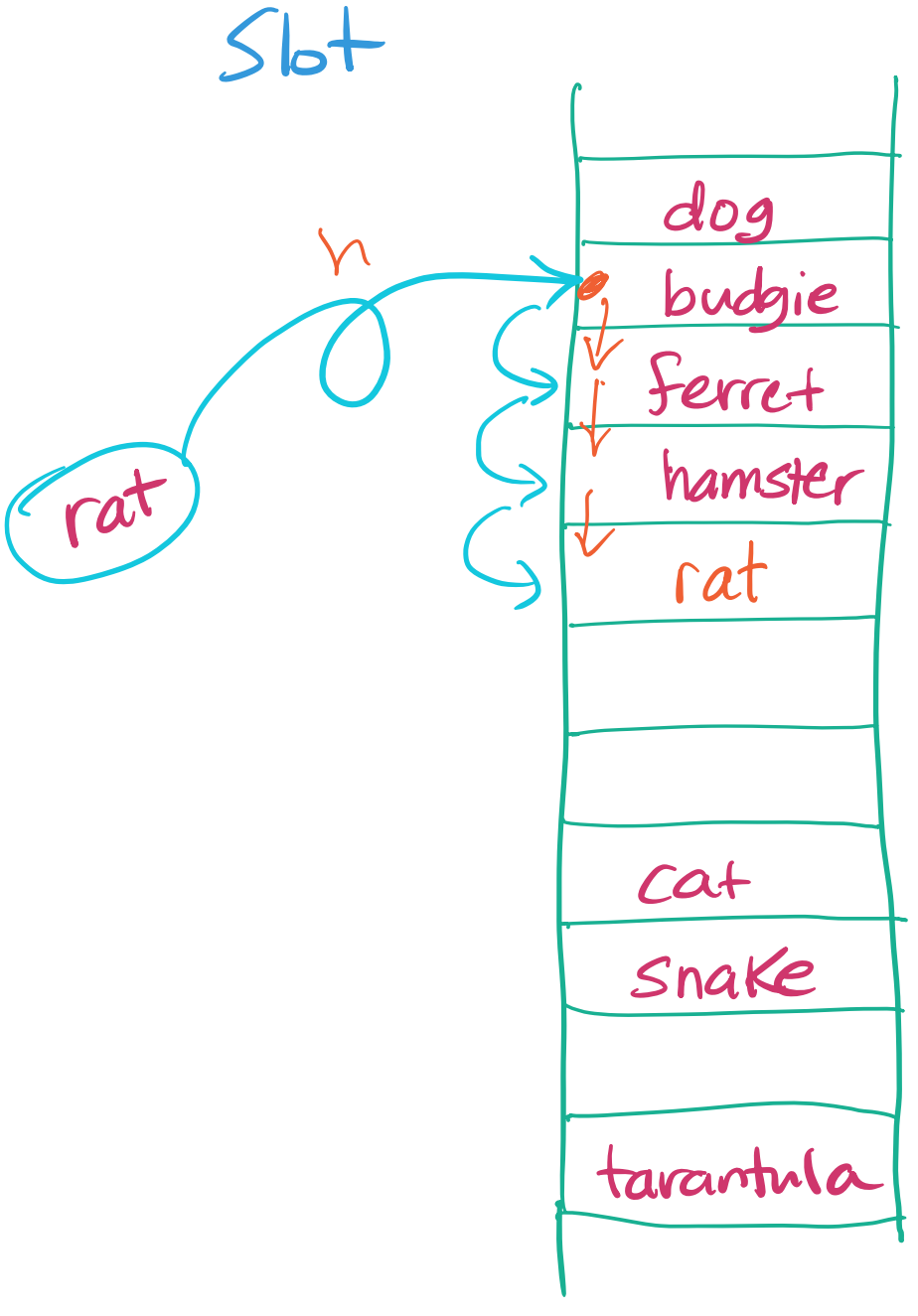
II Hashtables: ~~Open Addressing~~
(closed hashing)

- all elements (or pointers to them) are stored in the table itself

$\Rightarrow \alpha \leq 1$

$\alpha = \frac{n}{m}$ (n: # elements, m: # slots)

Simple method of handling collisions:
- add 1 to slot # until find an empty slot

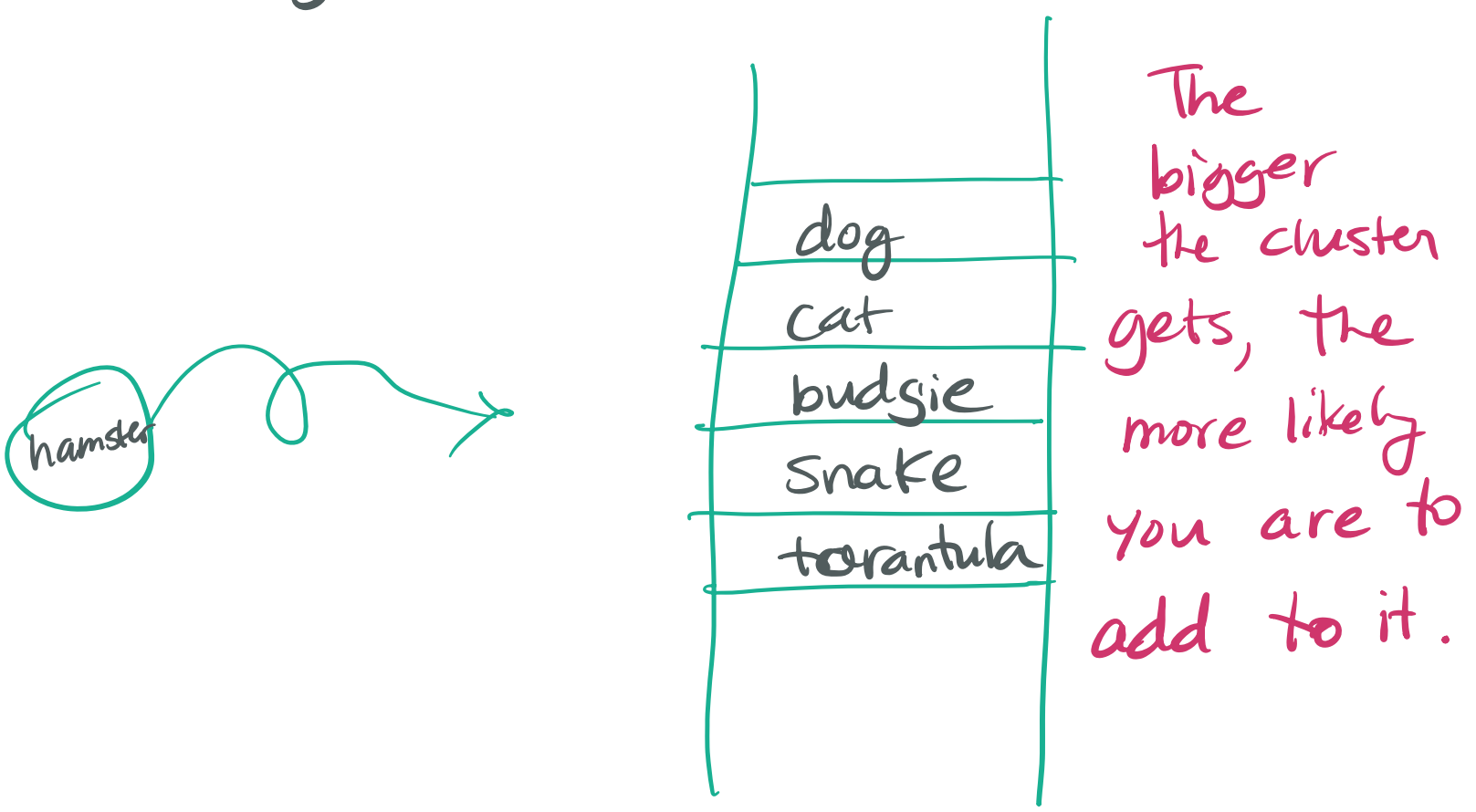


To find a key: K:
- Successively probe slots starting at h(K) until either find the key or encounter an empty slot

To Insert(K), do the same but when encounter an empty slot, store element there.

Problem: "Clustering"

- a few collisions will fill adjacent slots
  - as cluster grows, the cluster is more likely to be location of a collision, and becomes more likely to grow....



In our simple scheme, once you hash to a cluster, the operations take time Θ(size of cluster)

We can do better by hashing to determine the probe sequence.

probe sequence for key k
- slots searched when looking for key k.

Our simple strategy above yields a probe sequence of

$$\langle h(k), h(k)+1 \bmod m, h(k)+2 \bmod m, \ldots \rangle$$

Instead, lets use a different hash function depending on the probe:

$$\langle h(k,0), h(k,1), h(k,2), \ldots h(k,m-1) \rangle$$

We require that the probe sequence for any key be a permutation of

0 1 2 3 ... m-1

That way, we guarantee that

if ∃ an empty slot ⇒ an Insert will eventually find it and insert new key.

```
HashInsert (e, k)  // e an element with key k

for (int i = 0; i < m; i++)
    slot = h(k, i)
    if T[slot] = NULL
        T[slot] = (e, k)
        return slot
error "hash table overflow"
```

Linear probing

auxiliary hash function.

$$h(K,i) = \left(h'(K) + i\right) \bmod m$$

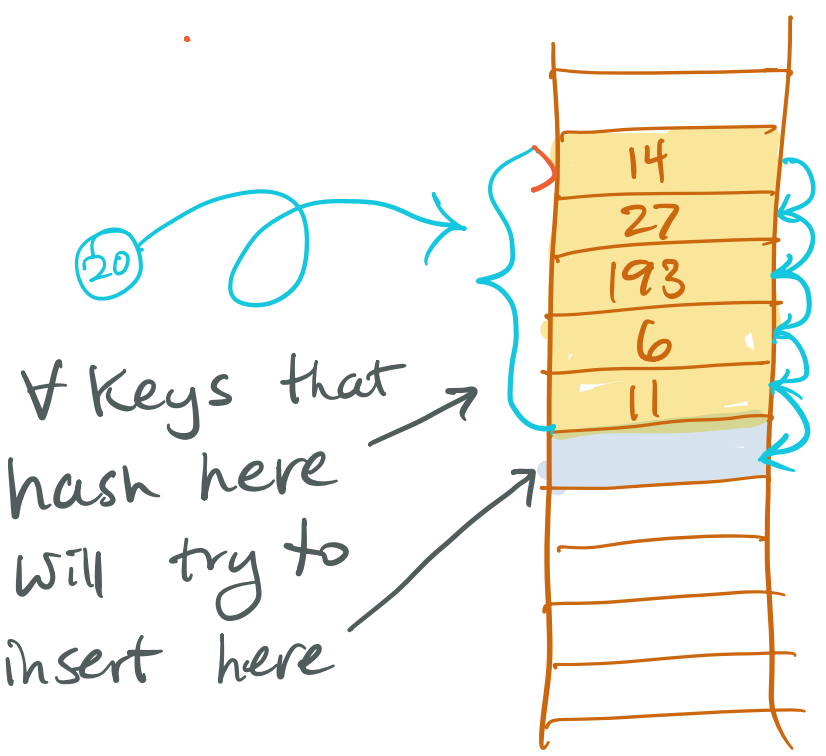
ie. this is our simple strategy and it has the problem of "primary" clustering: long runs of contiguous occupied slots

Quadratic probing

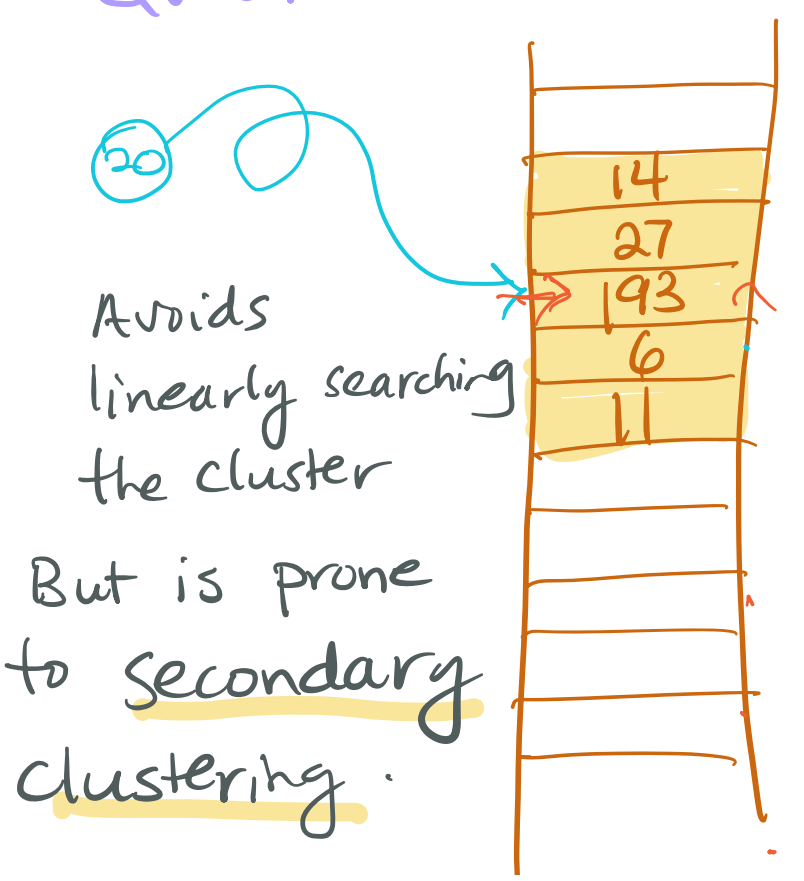
$$h(k,i) = \left(h'(K) + c_1 i + c_2 i^2\right) \bmod m$$

Better:

Linear



Quadratic



Double Hashing

$$h(k,i) = \left(h_1(k) + i\, h_2(k) \right) \bmod m$$

h_1, h_2 are auxiliary hash functions.

Two keys K_1 and K_2 may have a collision: $h_1(K_1) = h_1(K_2)$

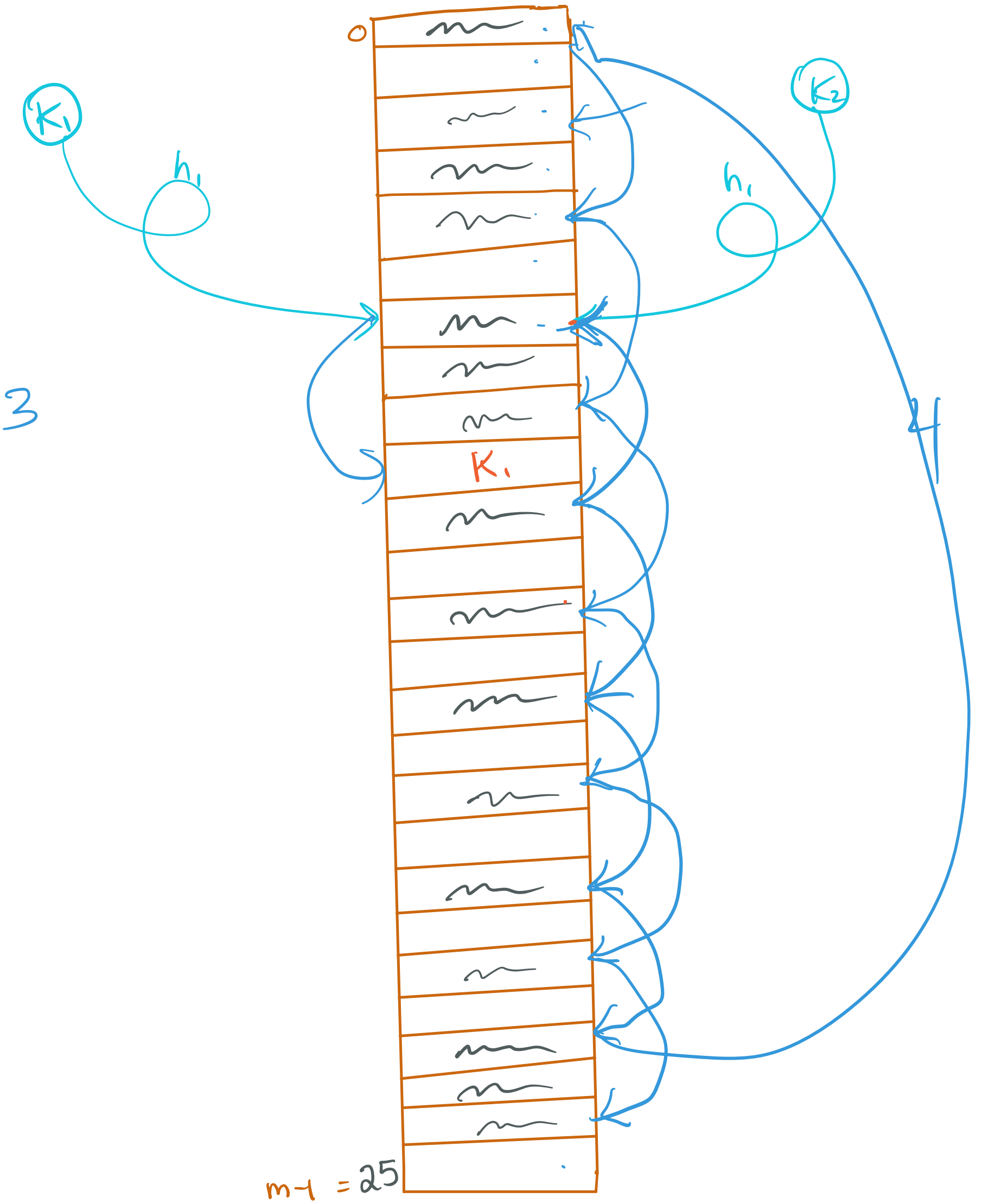
But what are the chances they <u>also</u> collide in their second probe? $h_2(K_1) \stackrel{?}{=} h_2(K_2)$

$\Rightarrow \frac{1}{m}$ likelihood

... low if you select them well
(so their outputs "look independent")

$h_2(k)$ = "offset" for successive probing.

eg if $h_1(K_1) = h_1(K_2) = 6$
and $h_2(K_1) = 3$ and $h_2(K_2) = 4$


0
K_1
h_1
K_2
h_1
3
K_1
m-1 = 25

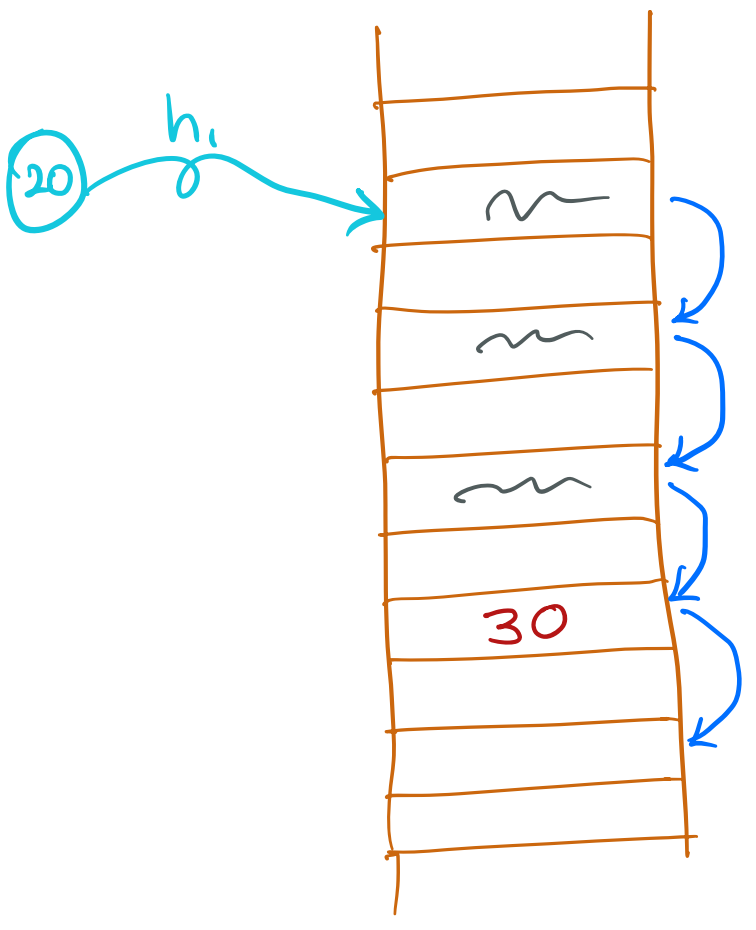
The value of $h_2(k)$ must be relatively prime with m in order that the probe sequence is a permutation of $\{0, \ldots, m-1\}$

- ie so that

If $\exists$ empty slot $\rightarrow$ probing will find it.

Easy solution: Use prime m

How to Delete when Double Hashing?



Just set to NULL?

Insert(20)

Delete(30)

Find(20)

during Insert.

How good is it?

Theorem 12.5 (CLRS)

Assuming uniform hashing, and closed hashing ~~open-address~~ hash table with load factor $\alpha = \frac{n}{m} < 1$,

$\Rightarrow$ # of probes expected in unsuccessful search is $\leq \frac{1}{1-\alpha}$

$= \frac{1}{1-\frac{n}{m}} = \frac{1}{\frac{m-n}{m}} = \frac{m}{m-n}$

Insert takes same time as unsuccessful search.

Theorem 12.7 (CLRS)

Assuming uniform hashing, and closed hashing open address hash table with load factor $\alpha = \frac{n}{m} < 1$,

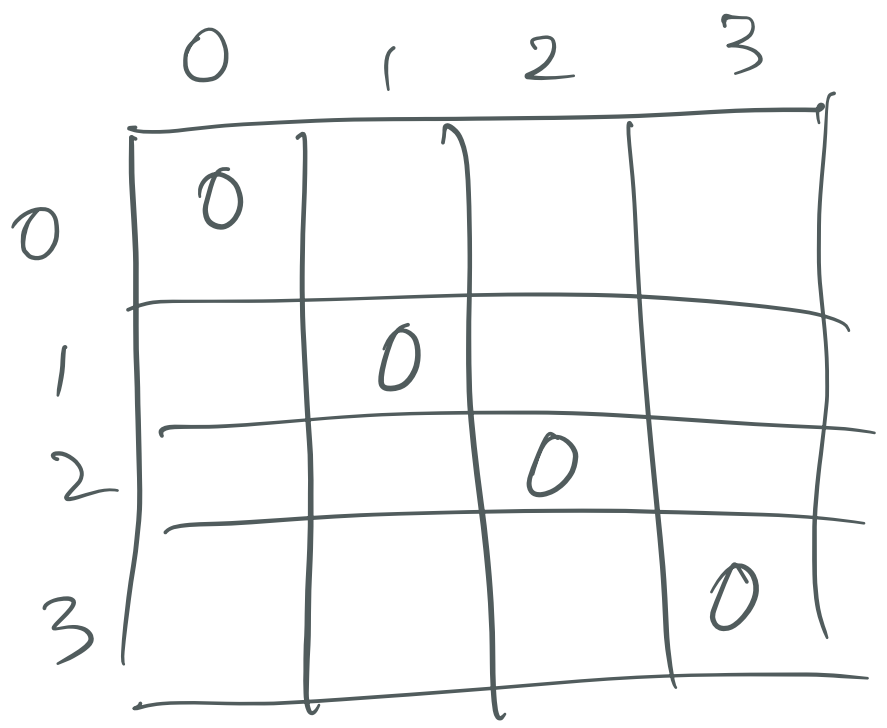
$\Rightarrow$ # of probes expected in successful search is $\frac{1}{\alpha} \ln \frac{1}{1-\alpha} + \frac{1}{\alpha}$

∴ if the hashtable is half full

then expect ≤ 3.39 probes to success!

90% : $\frac{1}{.9} * \ln(10) + \frac{1}{.9} \approx 3.67$

$\ln(10)$: 2.3



$O(n^n)$

Θ

$\Theta(n+m)$ is size of Adj Lists.

$\Theta(n^2)$ is size of Adj matrix