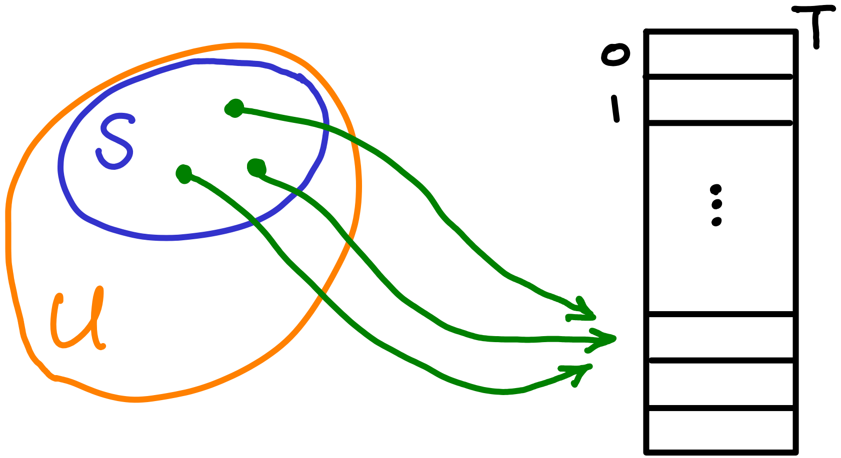
HASHING

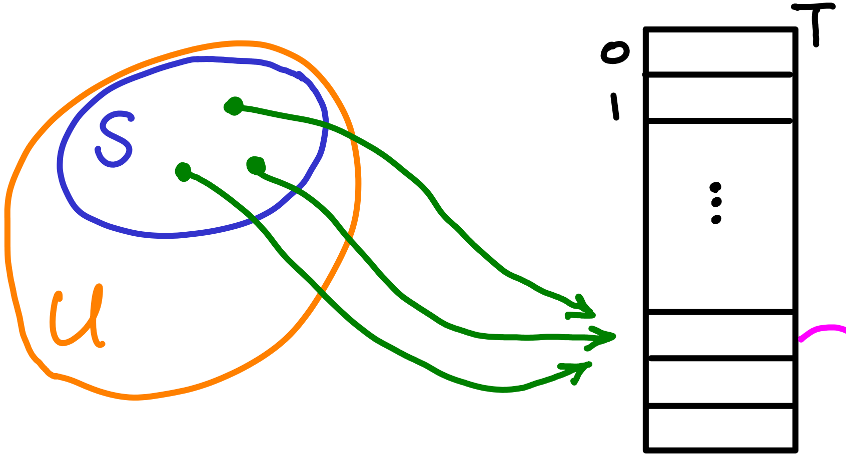
(Chaining)

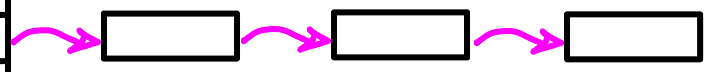


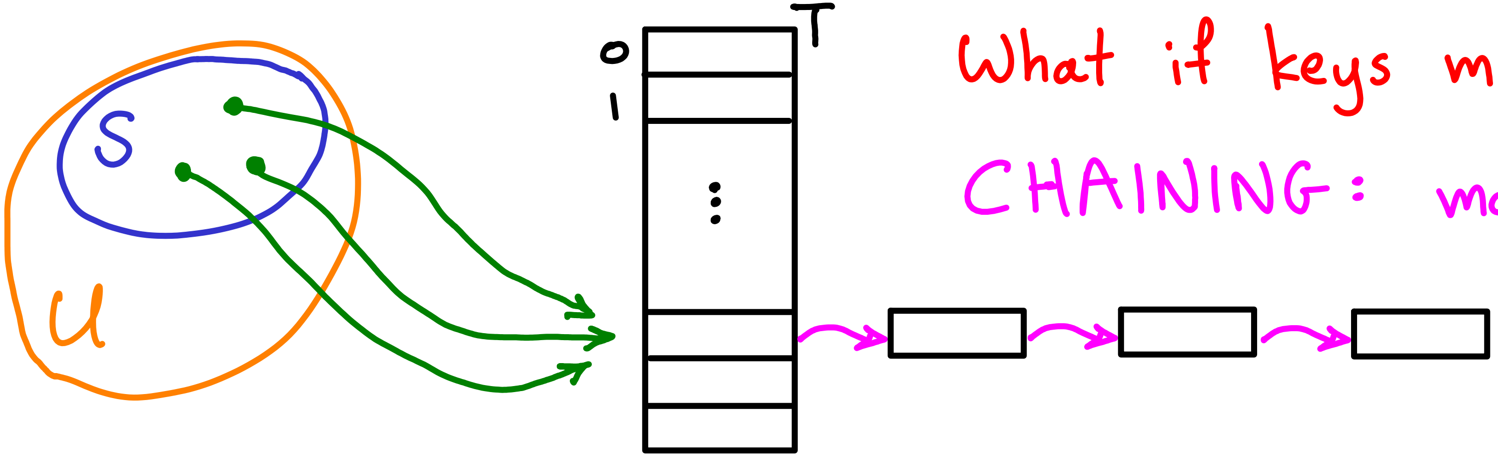
What if keys map to the same slot?

What if keys map to the same slot?

CHAINING: make a linked list







What if keys map to the same slot?

CHAINING: make a linked list

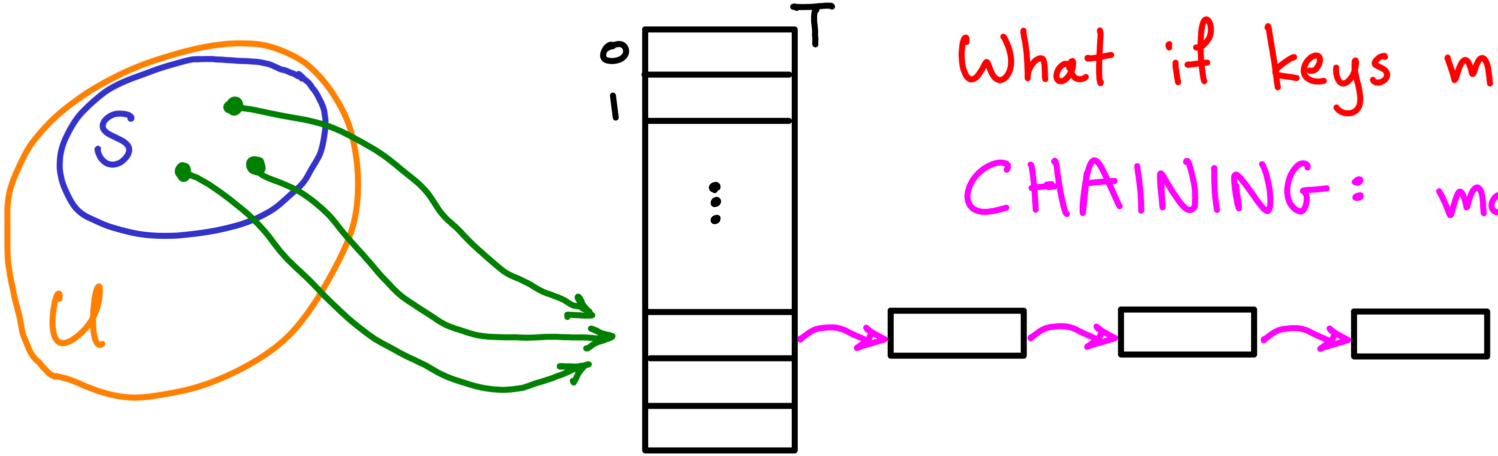
Insert: $\Theta(1)$

Search/Delete: $O(n)$

$|S| = n$

What if keys map to the same slot?

CHAINING: make a linked list



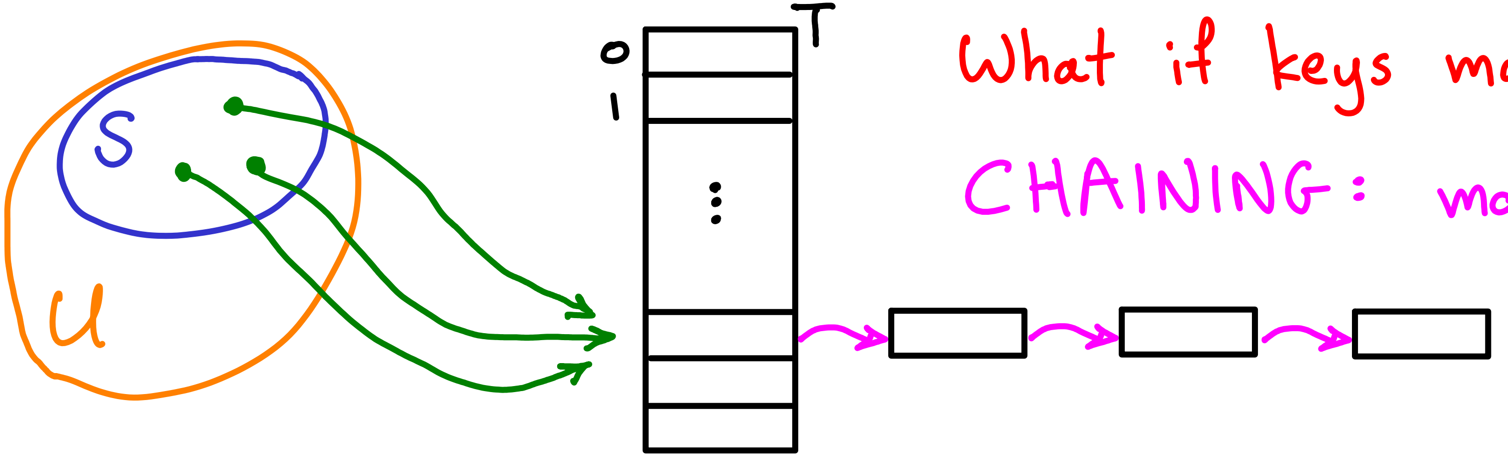
Insert: $\Theta(1)$

Search/Delete: $O(n)$

$|S| = n$

If we could spread S into T uniformly,

we would minimize max chain size (minimize worst-case time)



What if keys map to the same slot?

CHAINING: make a linked list

Insert: $\Theta(1)$

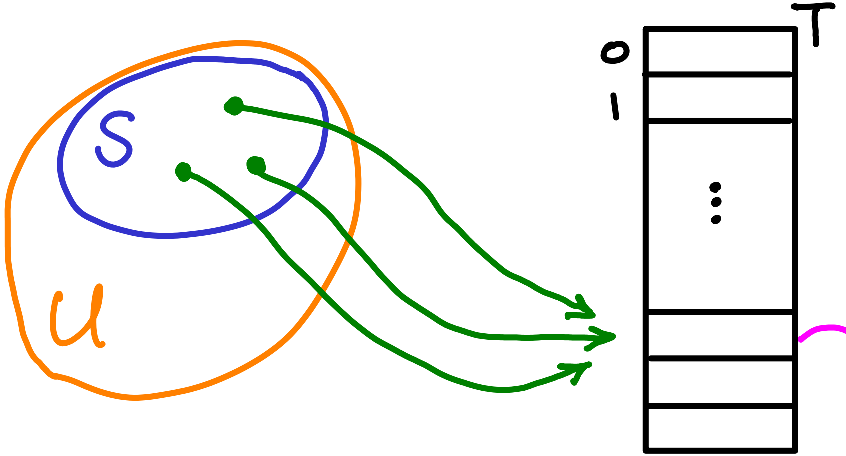
Search/Delete: $O(n)$

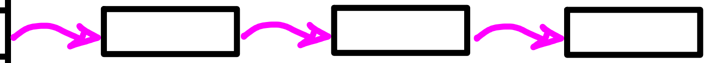
$|S| = n$

If we could spread S into T uniformly,

we would minimize max chain size (minimize worst-case time)

A random h() would do that





What if keys map to the same slot?

CHAINING: make a linked list

Insert: $\Theta(1)$

Search/Delete: $O(n)$

$|S| = n$

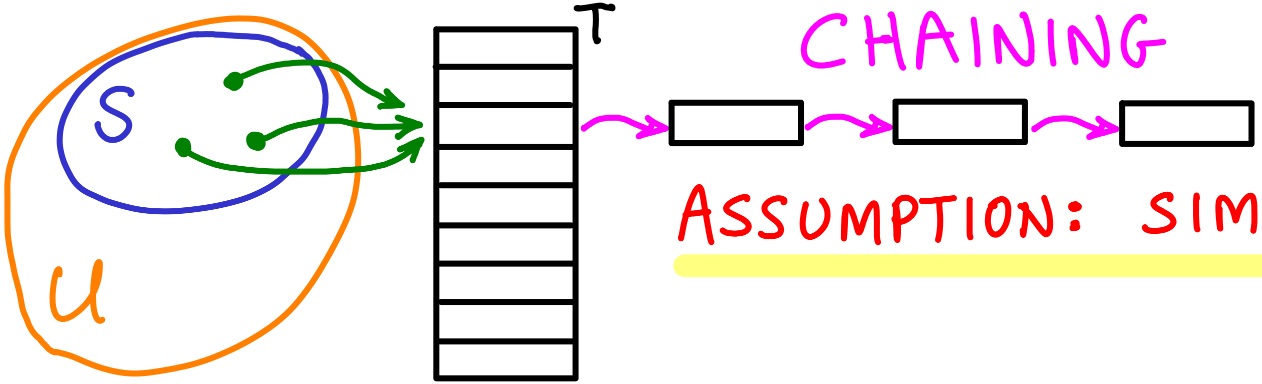
If we could spread S into T uniformly,

we would minimize max chain size (minimize worst-case time)

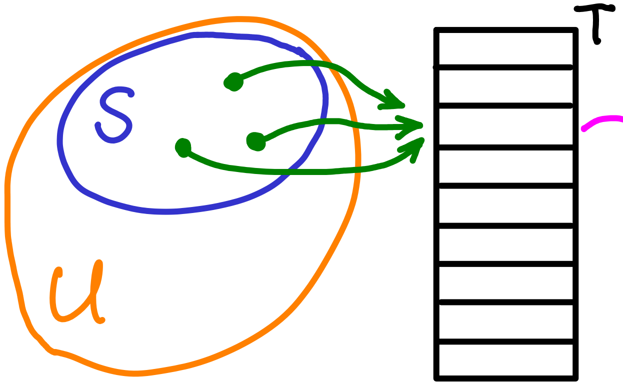
A random h() would do that but

we need h() to be consistent/deterministic (same key → same slot)

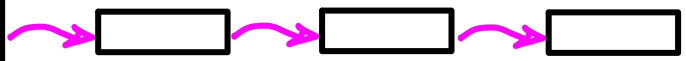
CHAINING



ASSUMPTION: SIMPLE UNIFORM HASHING

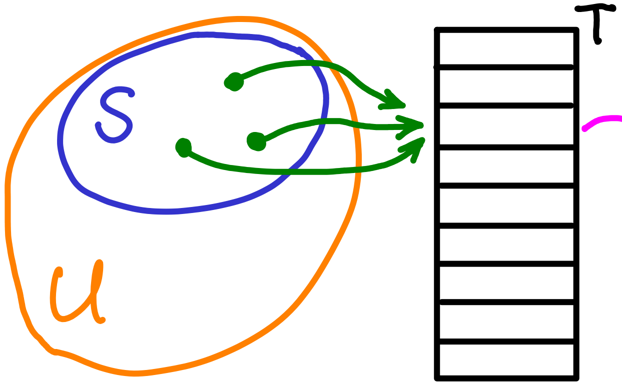


CHAINING

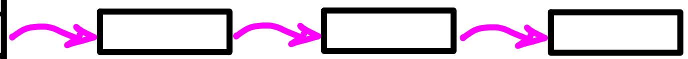


ASSUMPTION: SIMPLE UNIFORM HASHING

h( ) maps data uniformly like a random function



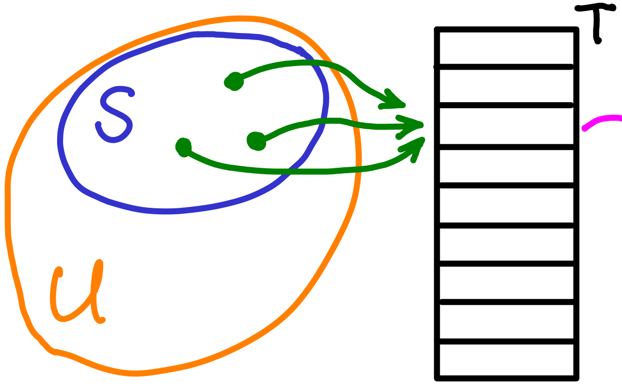
CHAINING



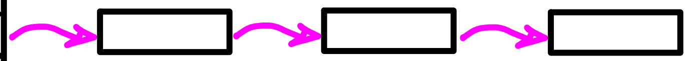
ASSUMPTION: SIMPLE UNIFORM HASHING

h( ) maps data uniformly like a random function

even though h() is consistent / deterministic



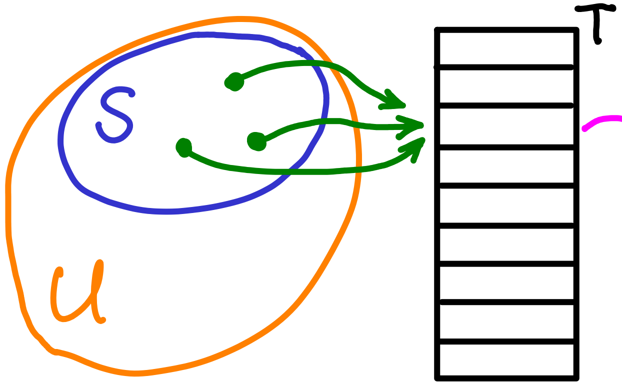
CHAINING



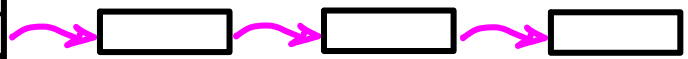
ASSUMPTION: SIMPLE UNIFORM HASHING

h( ) maps data uniformly like a random function

Probability two given keys collide: $\frac{1}{m}$



CHAINING

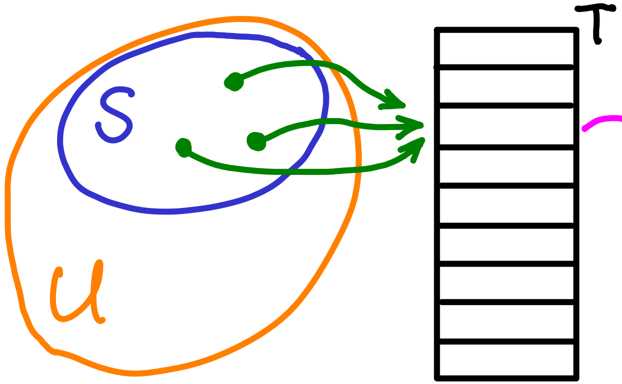


ASSUMPTION: SIMPLE UNIFORM HASHING

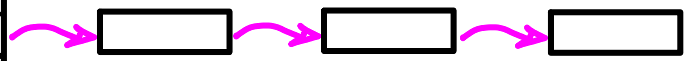
h( ) maps data uniformly like a random function

Probability two given keys collide: $\frac{1}{m}$

Expected list size $= \frac{n}{m}$



CHAINING

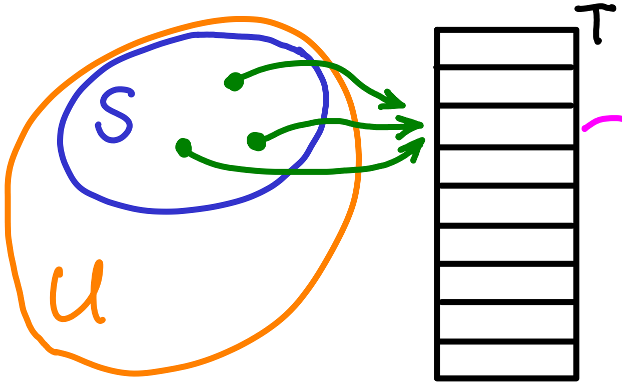


ASSUMPTION: SIMPLE UNIFORM HASHING

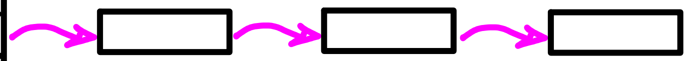
h( ) maps data uniformly like a random function

Probability two given keys collide: $\frac{1}{m}$

Expected list size $= \frac{n}{m} = \alpha =$ "load factor"



CHAINING



ASSUMPTION: SIMPLE UNIFORM HASHING

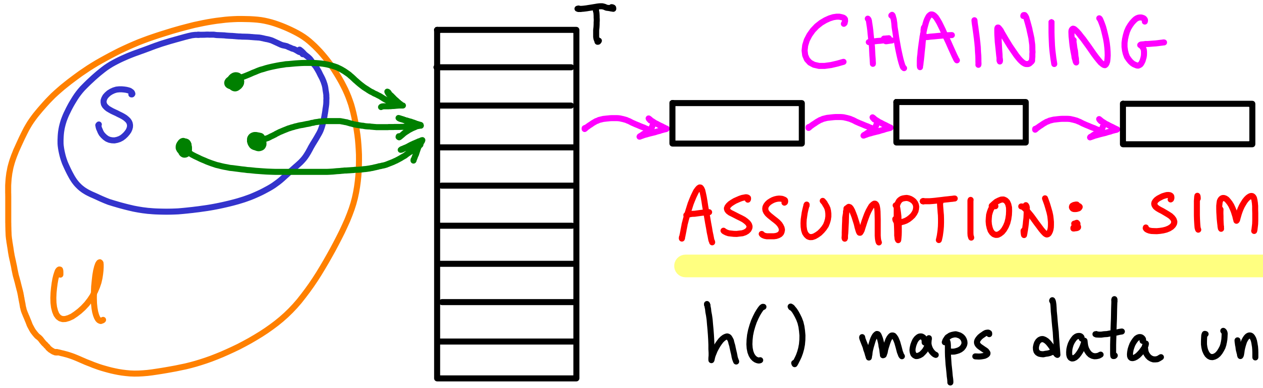
h( ) maps data uniformly like a random function

Probability two given keys collide: $\frac{1}{m}$

Expected list size $= \frac{n}{m} = \alpha$ = "load factor"

Expected time of search (and delete)

?



CHAINING

ASSUMPTION: SIMPLE UNIFORM HASHING

h( ) maps data uniformly like a random function

Probability two given keys collide: $\frac{1}{m}$

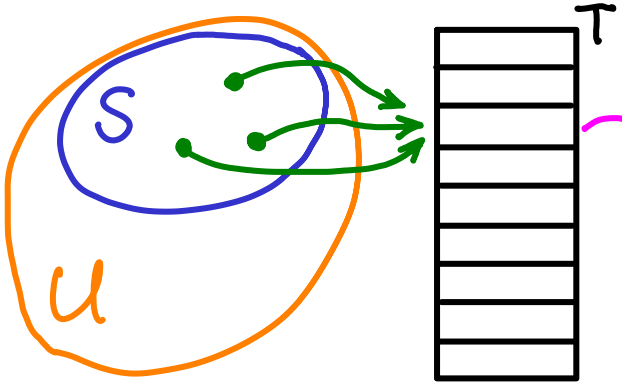
Expected list size $= \frac{n}{m} = \alpha$ = "load factor"

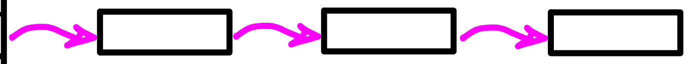
Expected time of search (and delete)

1) Map to T

2) Scan list

CHAINING





ASSUMPTION: SIMPLE UNIFORM HASHING

h( ) maps data uniformly like a random function

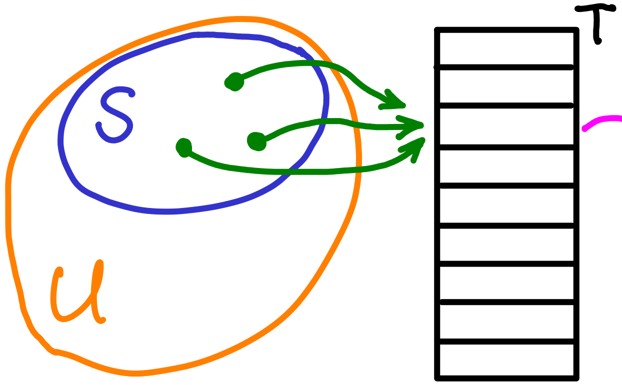
Probability two given keys collide: $\frac{1}{m}$

Expected list size $= \frac{n}{m} = \alpha$ = "load factor"

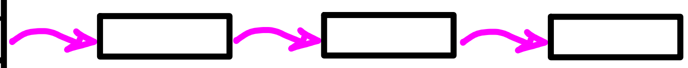
Expected time of search (and delete)

1) Map to T : assume $\Theta(1)$ to evaluate h( ).

2) Scan list



CHAINING



ASSUMPTION: SIMPLE UNIFORM HASHING

h( ) maps data uniformly like a random function

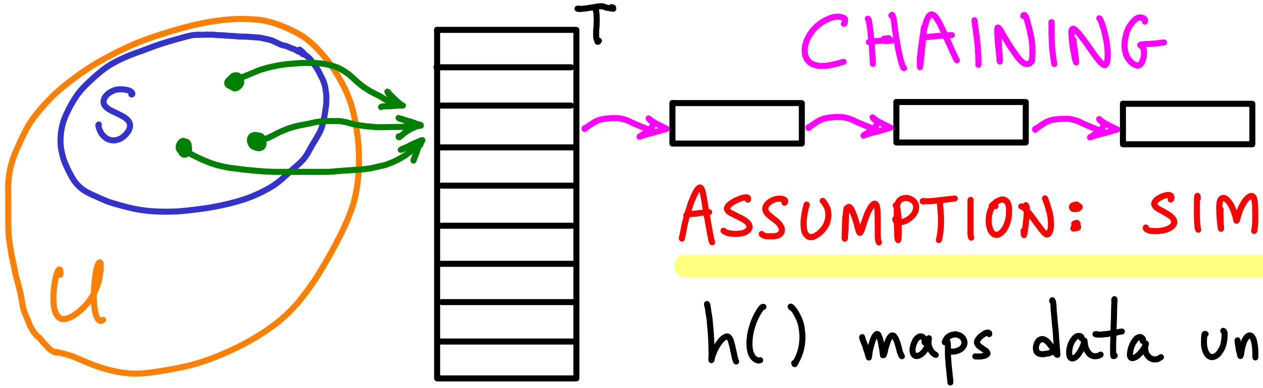
Probability two given keys collide: $\frac{1}{m}$

Expected list size $= \frac{n}{m} = \alpha$ = "load factor"

Expected time of search (and delete)

1) Map to T : assume $\Theta(1)$ to evaluate h( ).

2) Scan list (on average, half a list) $\Theta(\alpha)$



CHAINING

ASSUMPTION: SIMPLE UNIFORM HASHING

h( ) maps data uniformly like a random function

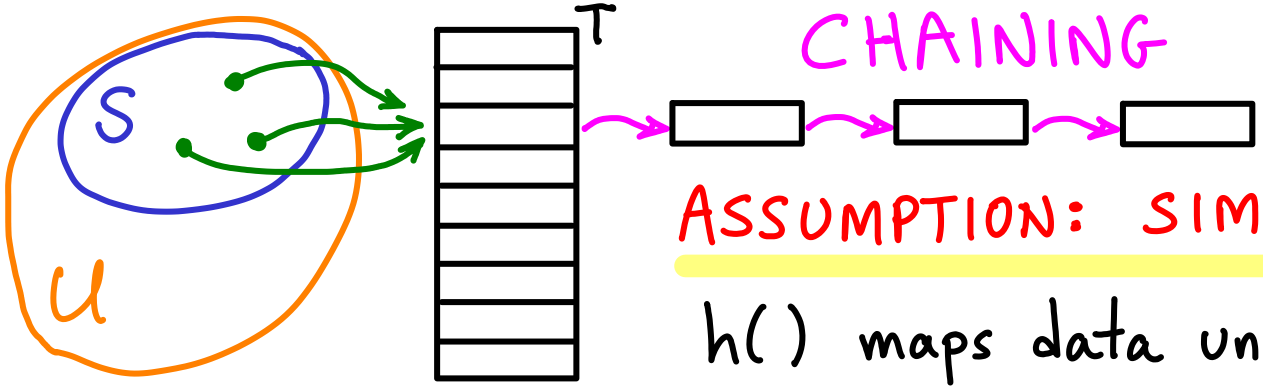
Probability two given keys collide: $\frac{1}{m}$

Expected list size $= \frac{n}{m} = \alpha$ = "load factor"

Expected time of search (and delete)

$$\left.\begin{array}{l} \text{1) Map to } T\text{: assume } \Theta(1) \text{ to evaluate } h(\,). \\ \text{2) Scan list (on average, half a list) } \Theta(\alpha) \end{array}\right\} \Theta(1+\alpha)$$

CHAINING



ASSUMPTION: SIMPLE UNIFORM HASHING

h( ) maps data uniformly like a random function

Probability two given keys collide: $\frac{1}{m}$

Expected list size $= \frac{n}{m} = \alpha$ = "load factor"

Expected time of search (and delete)

1) Map to T : assume $\Theta(1)$ to evaluate h( ).

2) Scan list (on average, half a list) $\Theta(\alpha)$

$\Theta(1+\alpha)$

great if $\alpha = O(1)$