



CS 225

Data Structures

October 26 – Hashing

G Carl Evans



What if $O(\log_m n)$ is not fast enough?

Do you feel lucky?

A Hash Table based Dictionary

Client Code:

```
1 Dictionary<KeyType, ValueType> d;  
2 d[k] = v;
```

A **Hash Table** consists of three things:

1. A hash function, $f(k)$
2. An array
3. Something to handle chaos when it occurs!



Hash Function

Our **hash function** consists of two parts:

- A **hash**:
- A **compression**:

Choosing a good hash function is tricky...

- Don't create your own (yet*)
- Very smart people have created very bad hash functions

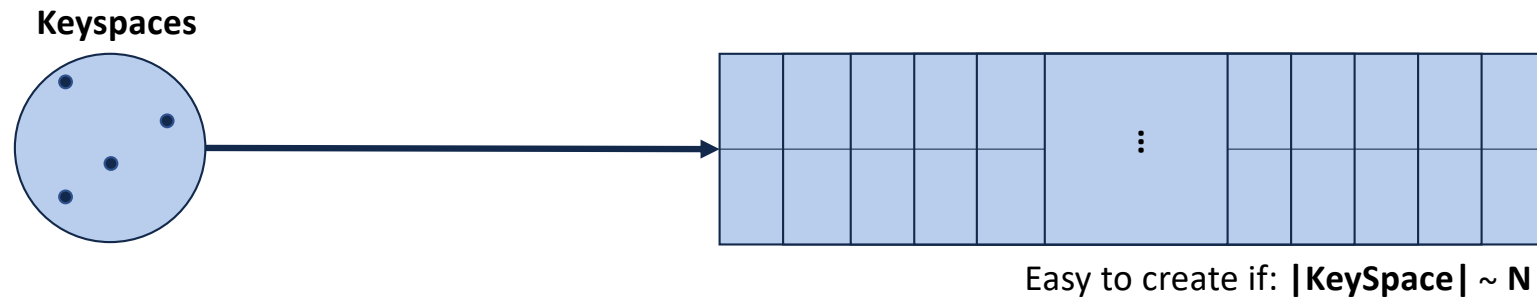


Hash Function

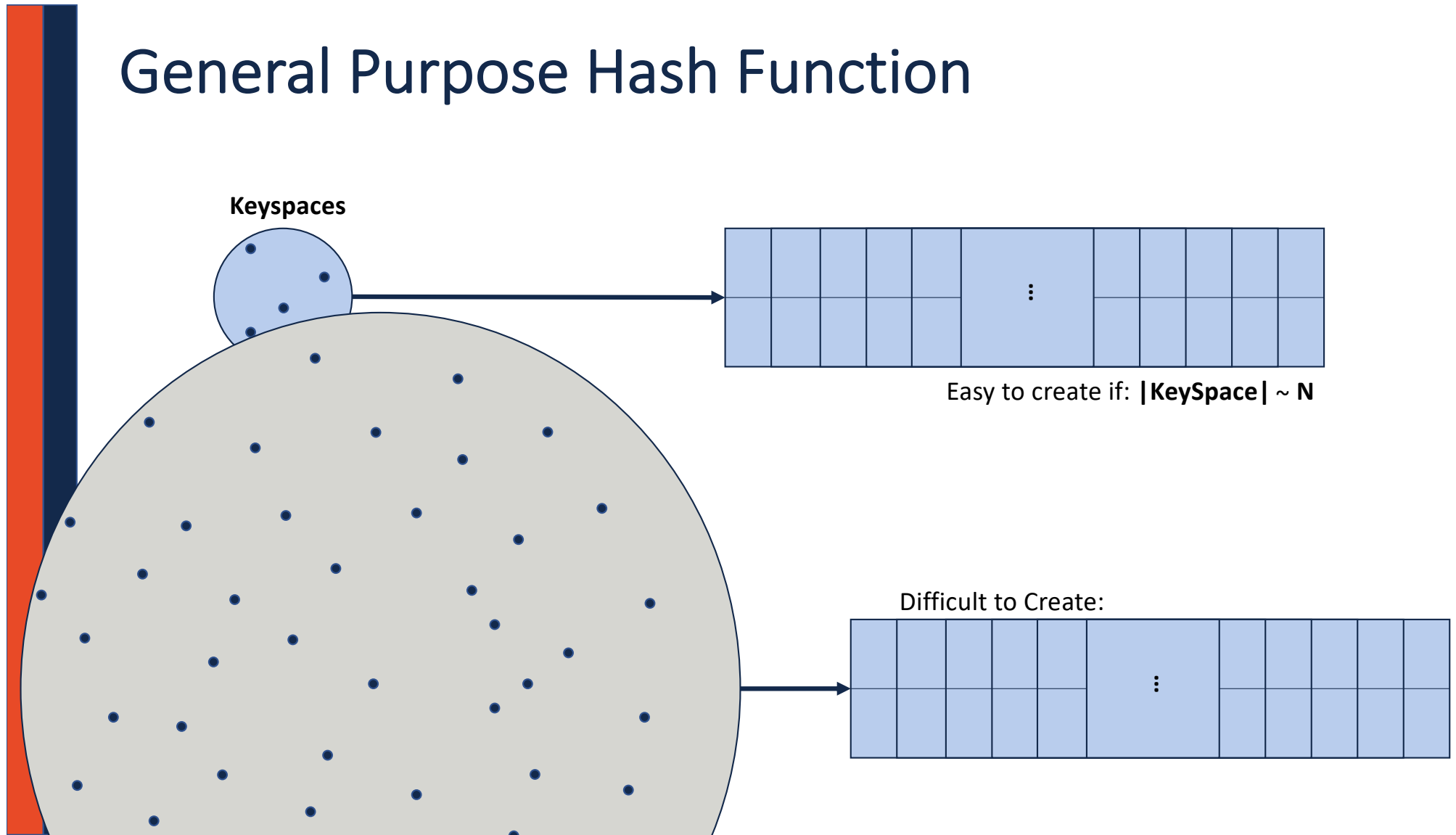
Characteristics of a good hash function:

1. Computation Time:
2. Deterministic:
3. Satisfy the SUHA:

General Purpose Hash Function

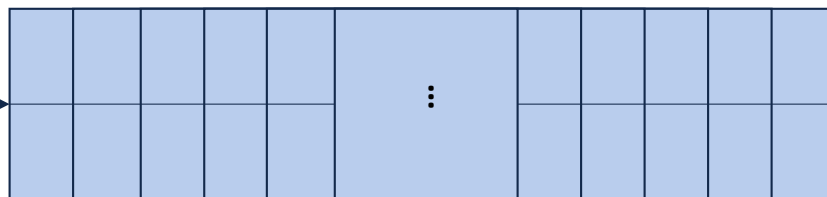
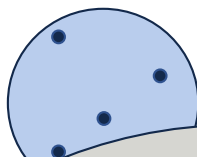


General Purpose Hash Function

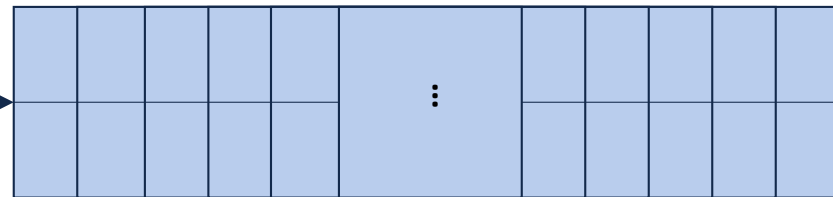
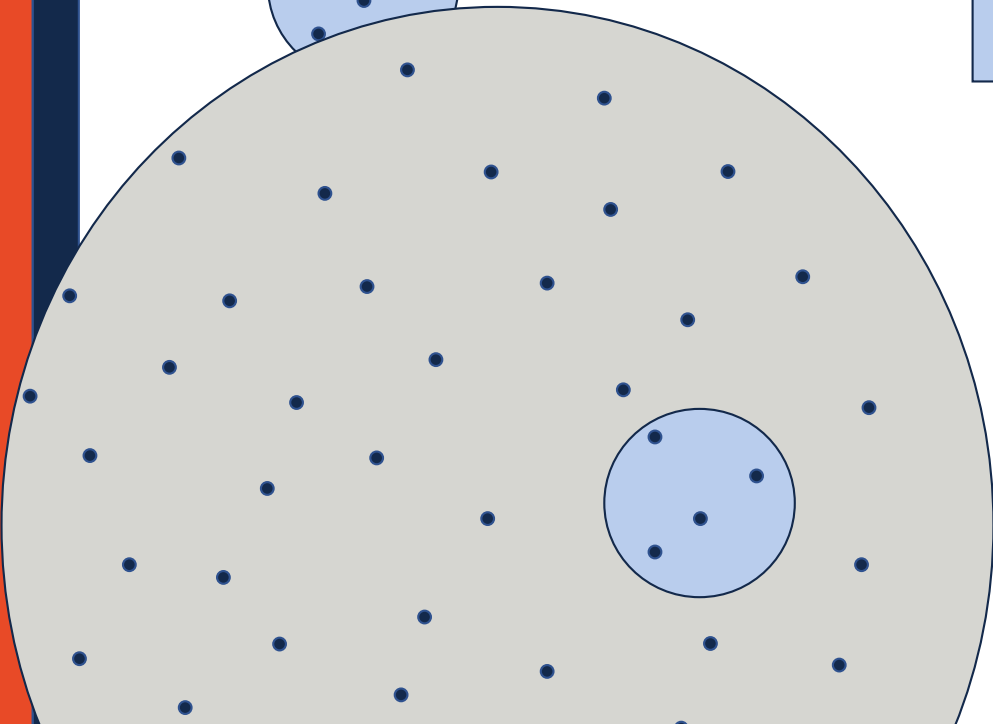


General Purpose Hash Function

Keyspaces



Easy to create if: $|\text{KeySpace}| \sim N$





Hash Function

In CS 225, we focus on **general purpose** hash functions.

Other hash functions exist with different properties
(eg: cryptographic hash functions)

(Example of open hashing)

Collision Handling: Separate Chaining

$S = \{ 16, 8, 4, 13, 29, 11, 22 \}$

$|S| = n$

$h(k) = k \% 7$

$|\text{Array}| = N$



	Worst Case	SUHA
Insert		
Remove/Find		

(Example of closed hashing)

Collision Handling: Probe-based Hashing

$S = \{ 16, 8, 4, 13, 29, 11, 22 \}$

$|S| = n$

$h(k) = k \% 7$

$|\text{Array}| = N$



(Example of closed hashing)

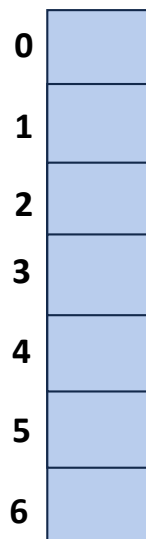
Collision Handling: Linear Probing

$S = \{ 16, 8, 4, 13, 29, 11, 22 \}$

$|S| = n$

$h(k) = k \% 7$

$|\text{Array}| = N$



Try $h(k) = (k + 0) \% 7$, if full...

Try $h(k) = (k + 1) \% 7$, if full...

Try $h(k) = (k + 2) \% 7$, if full...

Try ...

	Worst Case	SUHA
Insert		
Remove/Find		



A Problem w/ Linear Probing

Primary clustering:



Description:

Remedy:

(Example of closed hashing)

Collision Handling: Double hashing

$S = \{ 16, 8, 4, 13, 29, 11, 22 \}$

$|S| = n$

$h(k) = k \% 7$

$|\text{Array}| = N$



Try $h(k) = (k + 0 * h_2(k)) \% 7$, if full...

Try $h(k) = (k + 1 * h_2(k)) \% 7$, if full...

Try $h(k) = (k + 2 * h_2(k)) \% 7$, if full...

Try ...

$h(k, i) = (h_1(k) + i * h_2(k)) \% 7$

Running Times

The expected number of probes for find(key) under SUHA

Linear Probing:

- Successful: $\frac{1}{2}(1 + 1/(1-\alpha))$
- Unsuccessful: $\frac{1}{2}(1 + 1/(1-\alpha))^2$

(Don't memorize these equations, no need.)

Double Hashing:

- Successful: $1/\alpha * \ln(1/(1-\alpha))$
- Unsuccessful: $1/(1-\alpha)$

Instead, observe:

- **As α increases:**

Separate Chaining:

- Successful: $1 + \alpha/2$
- Unsuccessful: $1 + \alpha$

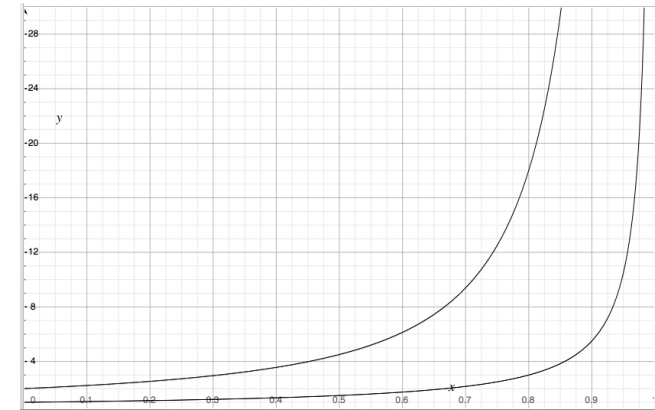
- **If α is constant:**

Running Times

The expected number of probes for find(key) under SUHA

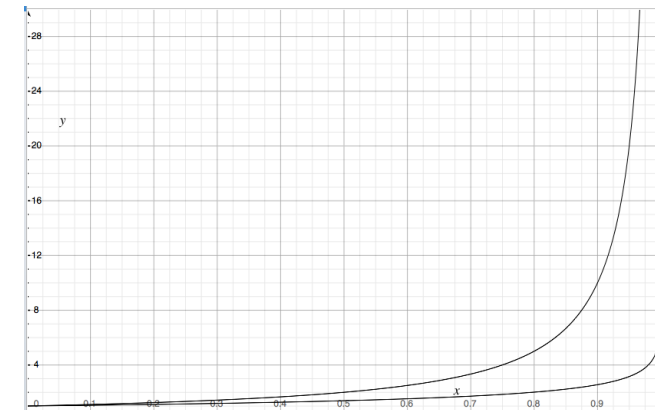
Linear Probing:

- Successful: $\frac{1}{2}(1 + \frac{1}{1-\alpha})$
- Unsuccessful: $\frac{1}{2}(1 + \frac{1}{1-\alpha})^2$



Double Hashing:

- Successful: $\frac{1}{\alpha} * \ln(\frac{1}{1-\alpha})$
- Unsuccessful: $\frac{1}{1-\alpha}$





ReHashing

What if the array fills?





Which collision resolution strategy is better?

- Big Records:
- Structure Speed:

What structure do hash tables replace?

What constraint exists on hashing that doesn't exist with BSTs?

Why talk about BSTs at all?

Running Times

	Hash Table	AVL	Linked List
Find	Amortized: Worst Case:		
Insert	Amortized: Worst Case:		
Storage Space			



std data structures

std::map



std data structures

std::map

::operator[]

::insert

::erase

::lower_bound(key) → Iterator to first element \leq key

::upper_bound(key) → Iterator to first element $>$ key



std data structures

std::unordered_map

::operator[]

::insert

::erase

~~— ::lower_bound(key) → Iterator to first element \leq key~~

~~— ::upper_bound(key) → Iterator to first element $>$ key~~

std data structures

std::unordered_map

::operator[]

::insert

::erase

~~::lower_bound(key) → Iterator to first element \leq key~~

~~::upper_bound(key) → Iterator to first element $>$ key~~

::load_factor()

::max_load_factor(ml) → Sets the max load factor