CS 400

Hashing – Introduction

ID: 09-01

Hashing

Goals:

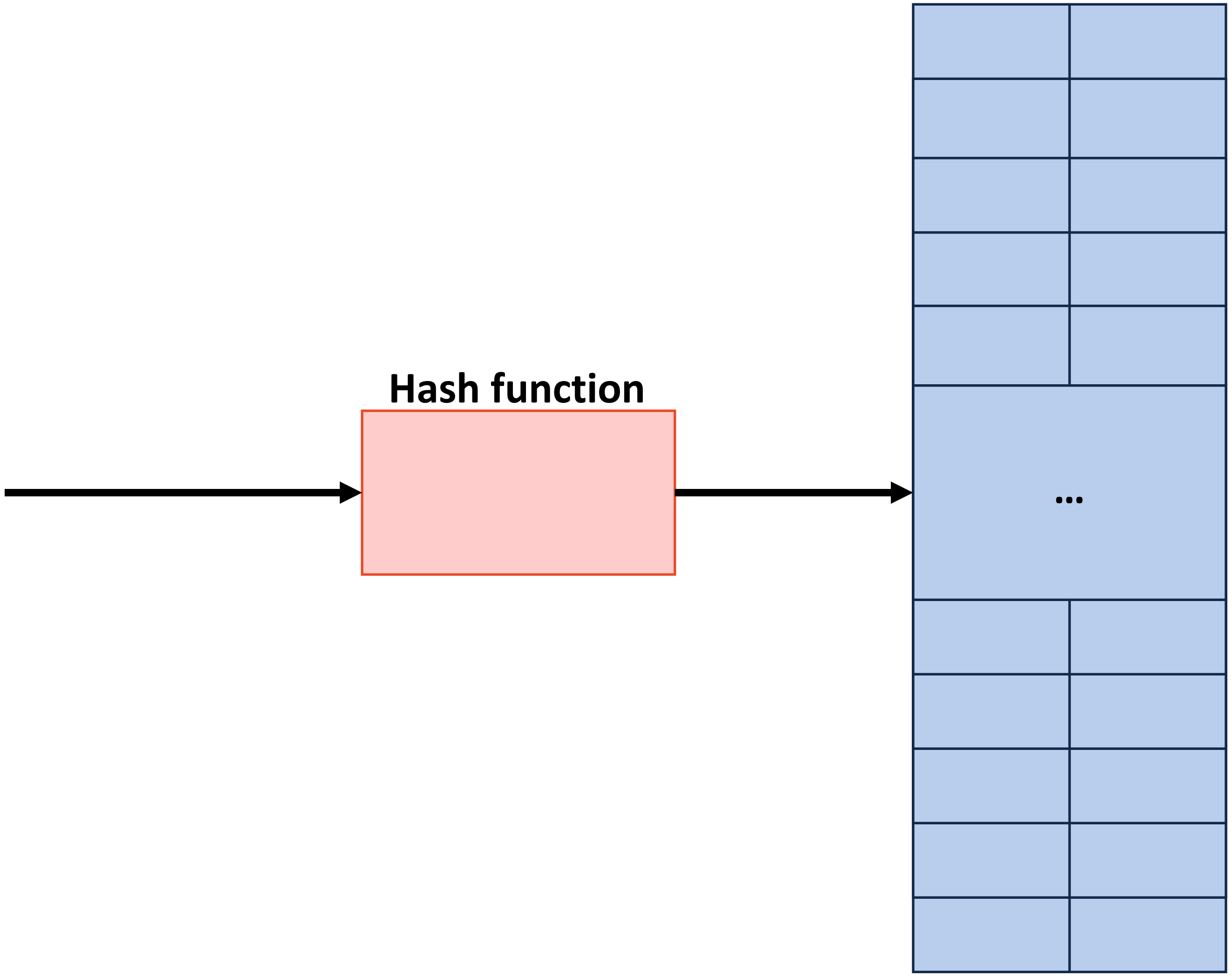
We want to define a **keyspace**, a (mathematical) description of the keys for a set of data.

…use a function to map the **keyspace** into a small set of integers.

Hashing

Locker Number	Name
103	
92	
330	
46	
124	

Hashing



A Hash Table based Dictionary

Client Code:

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

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

1.

2.

3.

CS 400

Hashing – Hash Function

ID: 09-02

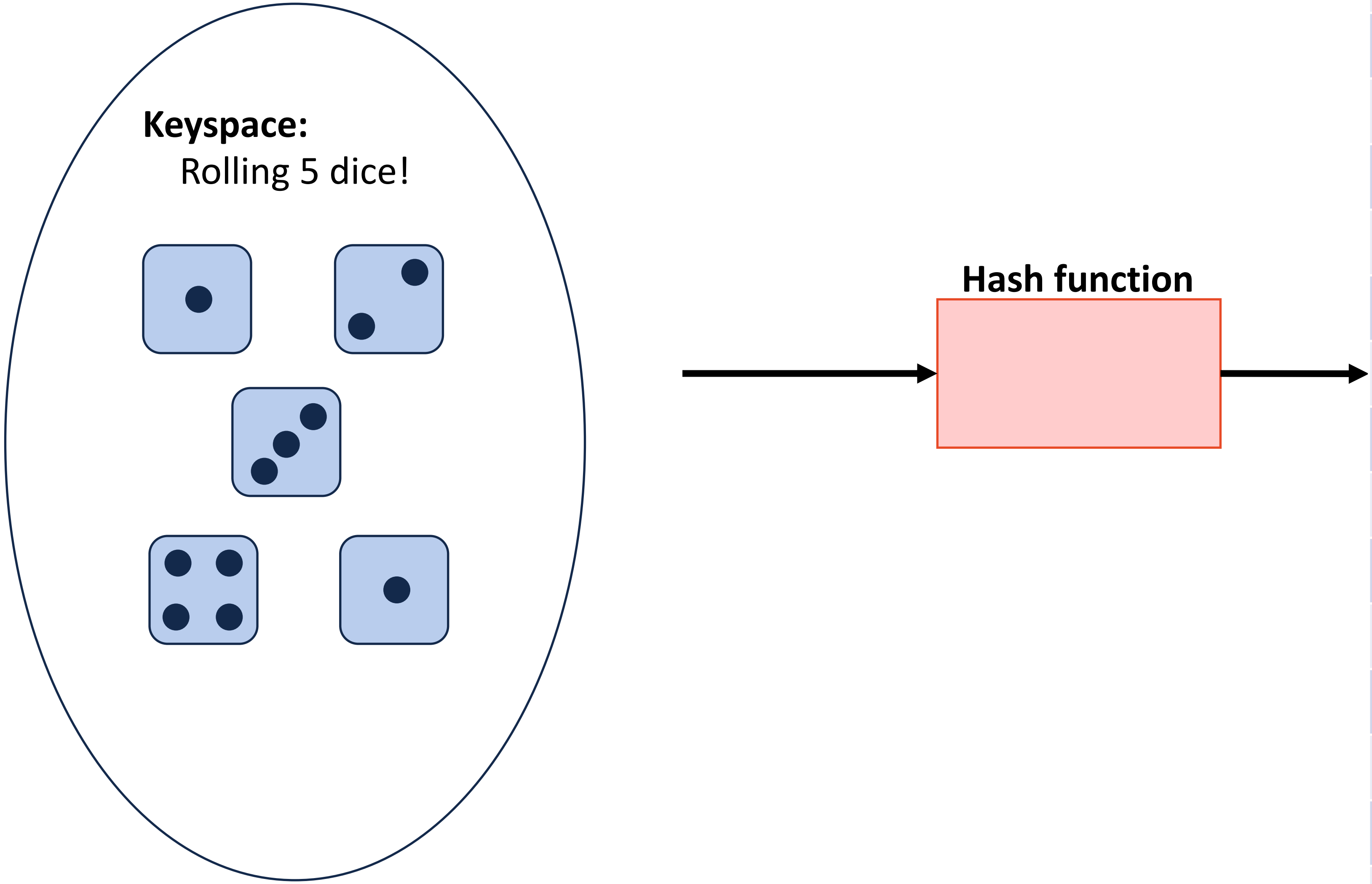
A Perfect Hash Function

(Angrave, CS 241)
(Beckman, CS 421)
(Cunningham, CS 210)
(Davis, CS 101)
(Evans, CS 126)
(Fagen-Ulmschneider, CS 225)
(Gunter, CS 422)
(Herman, CS 233)

Hash function

Key	Value

A Perfect Hash Function



Key	Value
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

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:

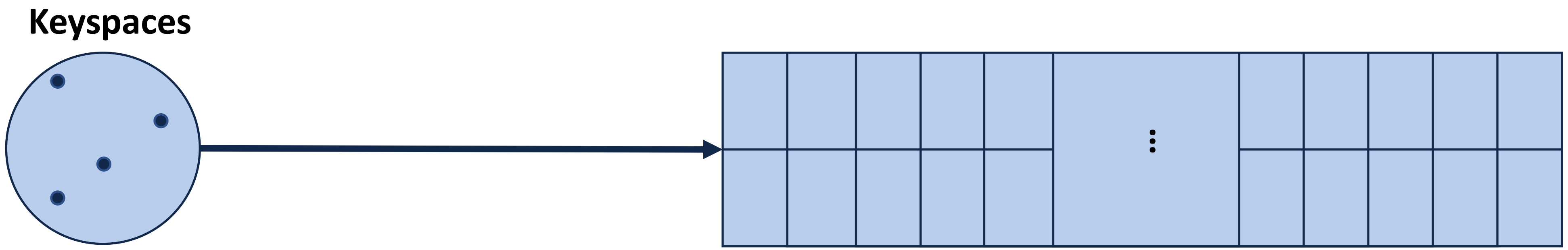
3. Satisfy the SUHA:

CS 400

Hash Function Examples

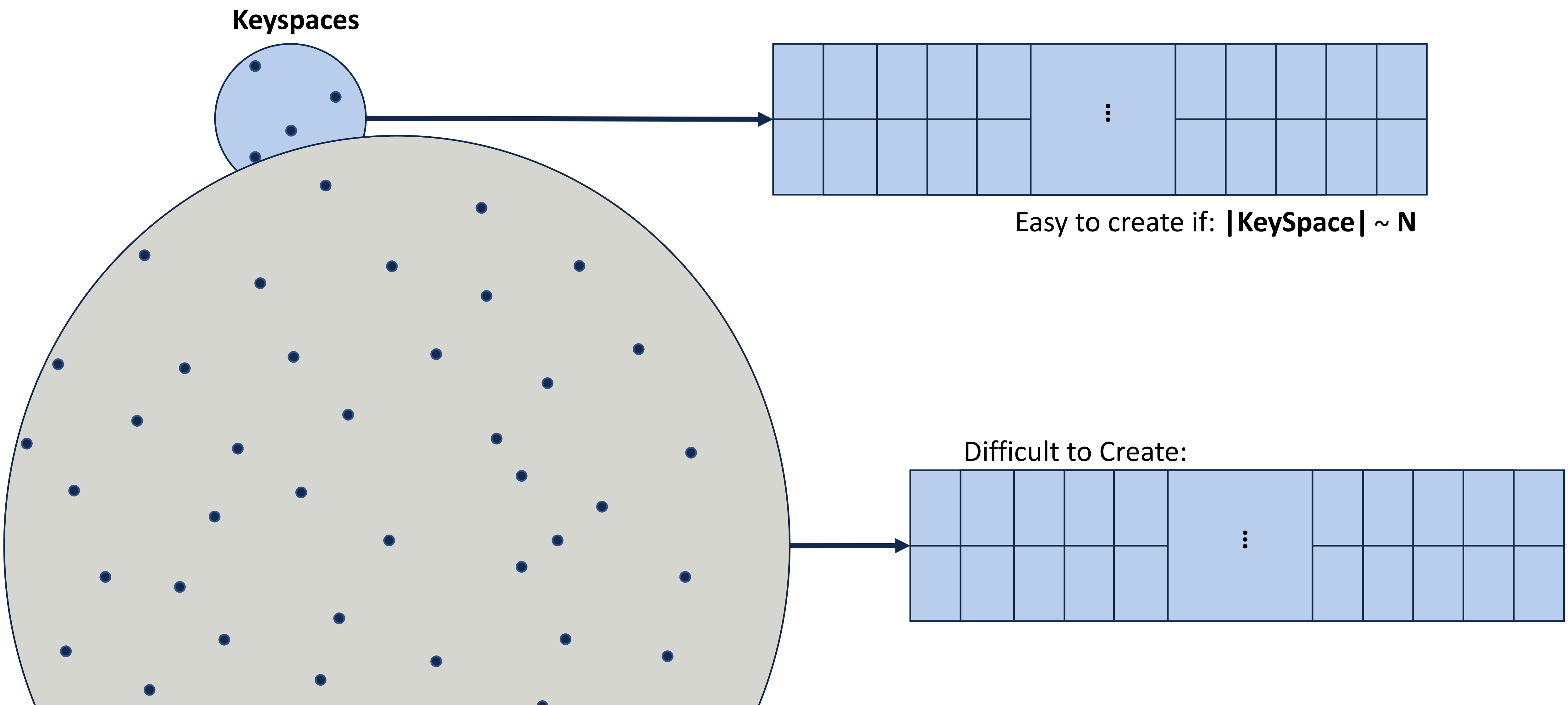
ID: 09-03

General Purpose Hash Function

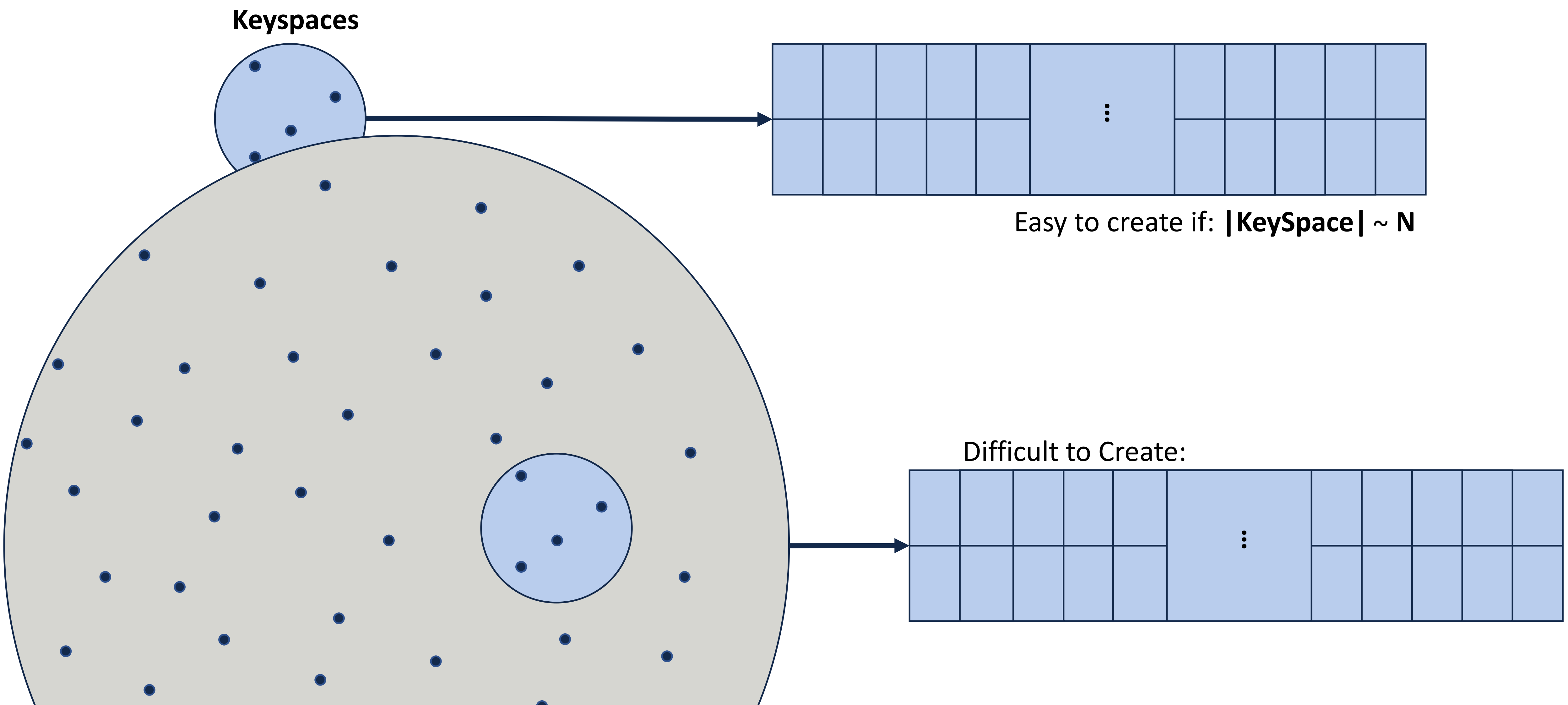


Easy to create if: **|KeySpace|** ~ **N**

General Purpose Hash Function



General Purpose Hash Function



Hash Function

Given: Easy to create a hash function of strings of length 8.

Idea: Map 40 character things to length 8:

```
Alice was beginning to get very tired of
 sitting by her sister on the bank, and 
of having nothing to do: once or twice s
he had peeped into the book her sister w
as reading, but it had no pictures or co
nversations in it, ‘and what is the use 
of a book,’ thought Alice ‘without pictu
res or conversations?’  So she was consi
dering in her own mind (as well as she c
ould, for the hot day made her feel very
 sleepy and stupid), whether the pleasur
e of making a daisy-chain would be worth
 the trouble of getting up and picking t
he daisies, when suddenly a White Rabbit
 with pink eyes ran close by her.  There
 was nothing so very remarkable in that;
 nor did Alice think it so very much out
 of the way to hear the Rabbit say to it
self, ‘Oh dear! Oh dear! I shall be late
!’ (when she thought it over afterwards,
 it occurred to her that she ought to ha
```

Idea: Map 40 character things to length 8:

```
https://en.wikipedia.org/wiki/Main_Page
https://en.wikipedia.org/wiki/Battle_of_
https://en.wikipedia.org/wiki/Vector_Gen
https://en.wikipedia.org/wiki/2017_Austr
https://en.wikipedia.org/wiki/19th_Natio
https://en.wikipedia.org/wiki/Japanese_g
```

Hash Function

In CS 400, we will focus on **general purpose** hash functions.

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

CS 400

Collision Handling

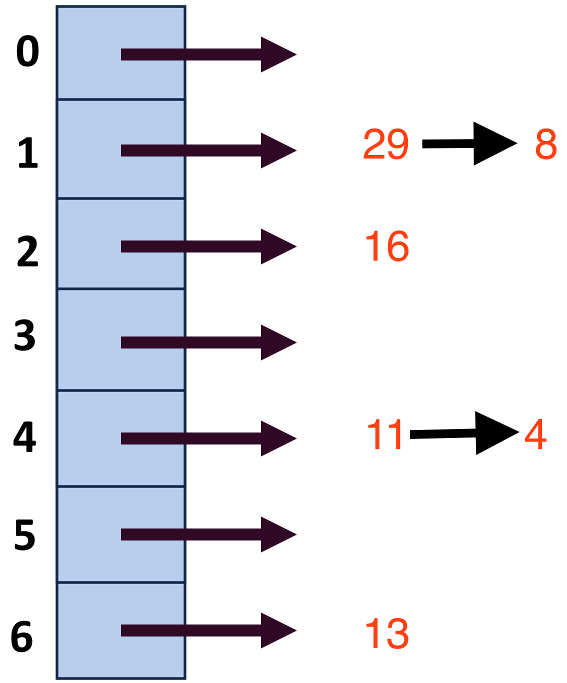
ID: 09-04-PartA

(Example of open hashing)

Collision Handling: Separate Chaining

S = { 16, 8, 4, 13, 29, 11, 22 } **|S| = n**

h(k) = k % 7 **|Array| = N**



alpha= n/N

use linked list to handle collision

	Worst Case	SUHA
Insert	O(1)	O(1)
Remove/Find	O(n)	O(α)

CS 400

Collision Handling

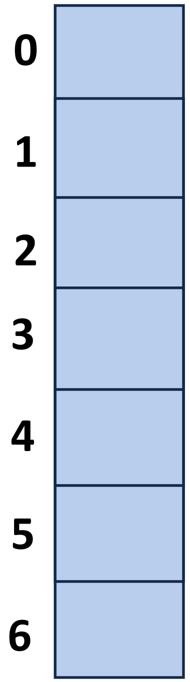
ID: 09-04-PartB

(Example of closed hashing)

Collision Handling: Probe-based Hashing

S = { 16, 8, 4, 13, 29, 11, 22 } **|S| = n**

h(k) = k % 7 **|Array| = N**

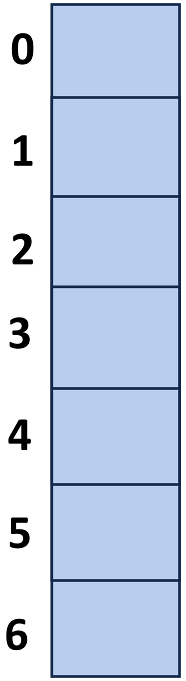


(Example of closed hashing)

Collision Handling: Linear Probing

S = { 16, 8, 4, 13, 29, 11, 22 } |S| = n

h(k) = k % 7 |Array| = N



Try h(k) = (k + 0) % 7, if full…

Try h(k) = (k + 1) % 7, if full…

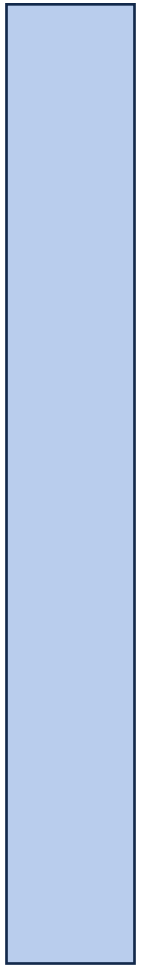
Try h(k) = (k + 2) % 7, if full…

Try …

	Worst Case	SUHA
Insert		
Remove/Find		

A Problem w/ Linear Probing

Primary clustering:



Description:

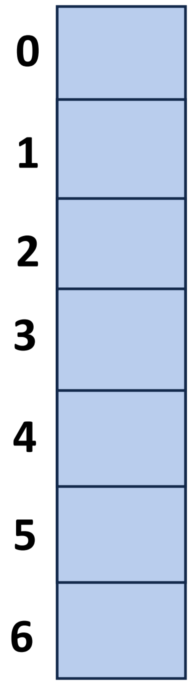
Remedy:

(Example of closed hashing)

Collision Handling: Double hashing

S = { 16, 8, 4, 13, 29, 11, 22 } **|S| = n**

h(k) = k % 7 **|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: $\frac{1}{2}(1 + 1/(1-\alpha))^2$

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

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

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

Instead, observe:

- **As α increases:**

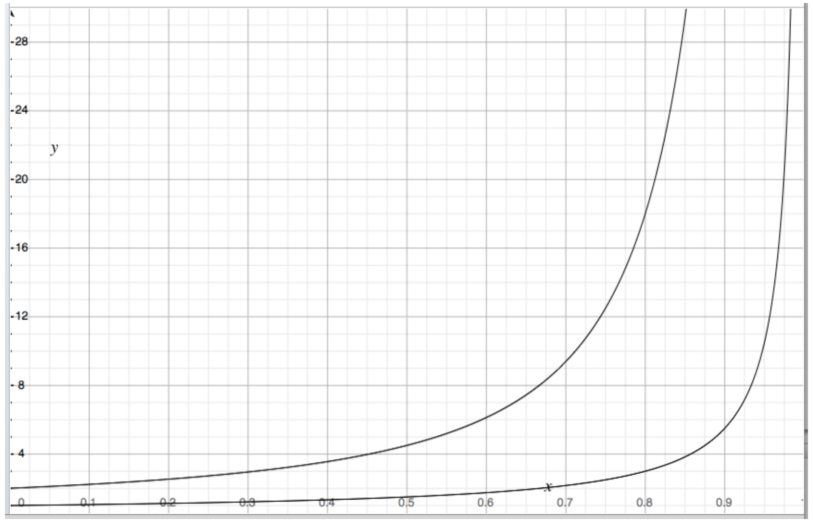
- **If α is constant:**

Running Times

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

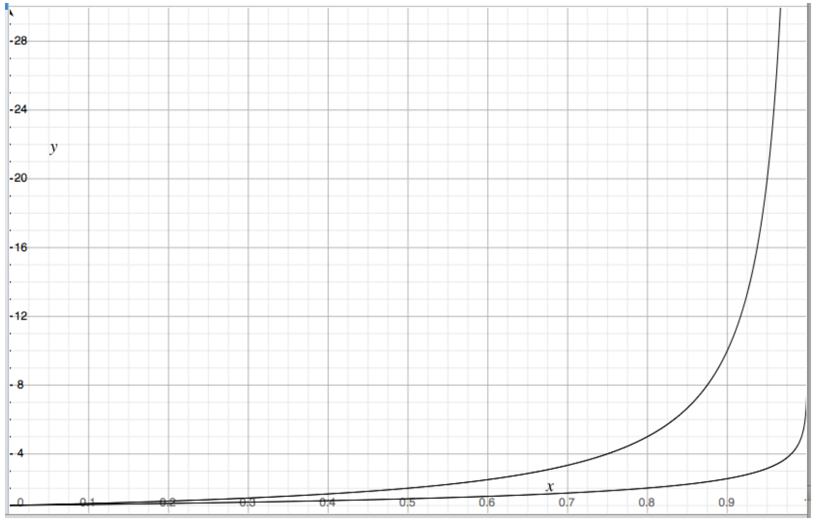
Linear Probing:

- Successful: **½(1 + 1/(1-α))**
- Unsuccessful: **½(1 + 1/(1-α))^2**



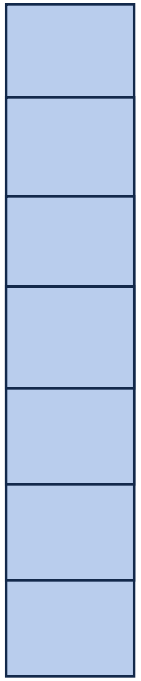
Double Hashing:

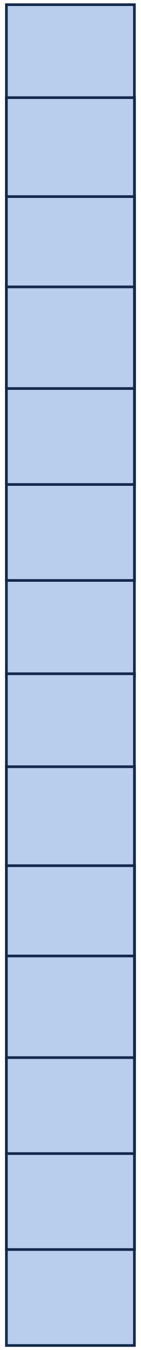
- Successful: **1/α * ln(1/(1-α))**
- Unsuccessful: **1/(1-α)**



ReHashing

What if the array fills?





CS 400

Hashing Analysis

ID: 09-05

Which collision resolution strategy is better?

- Big Records: Separate Chaining
- Structure Speed: Double hashing

What structure do hash tables replace? dictionary

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			

CS 400

Hash Tables in C++

ID: 09-06

std data structures

std::map

std data structures

std::map

::operator[]

::insert

::erase

::lower_bound(key) ➔ Iterator to first element ≤ 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 ≤ 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 ≤ key~~
~~::upper_bound(key) ➔ Iterator to first element > key~~

::load_factor()
::max_load_factor(ml) ➔ Sets the max load factor