Late class start: Enjoy the solar eclipse!

Class begins at 2:15 PM.

Algorithms and Data Structures for Data Science
Hashing

CS 277
Brad Solomon

April 8, 2024

Graphs Algorithms ???

Dijkstra

Hashing

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

(Dictionary in Python)

Motivate and define a hash table

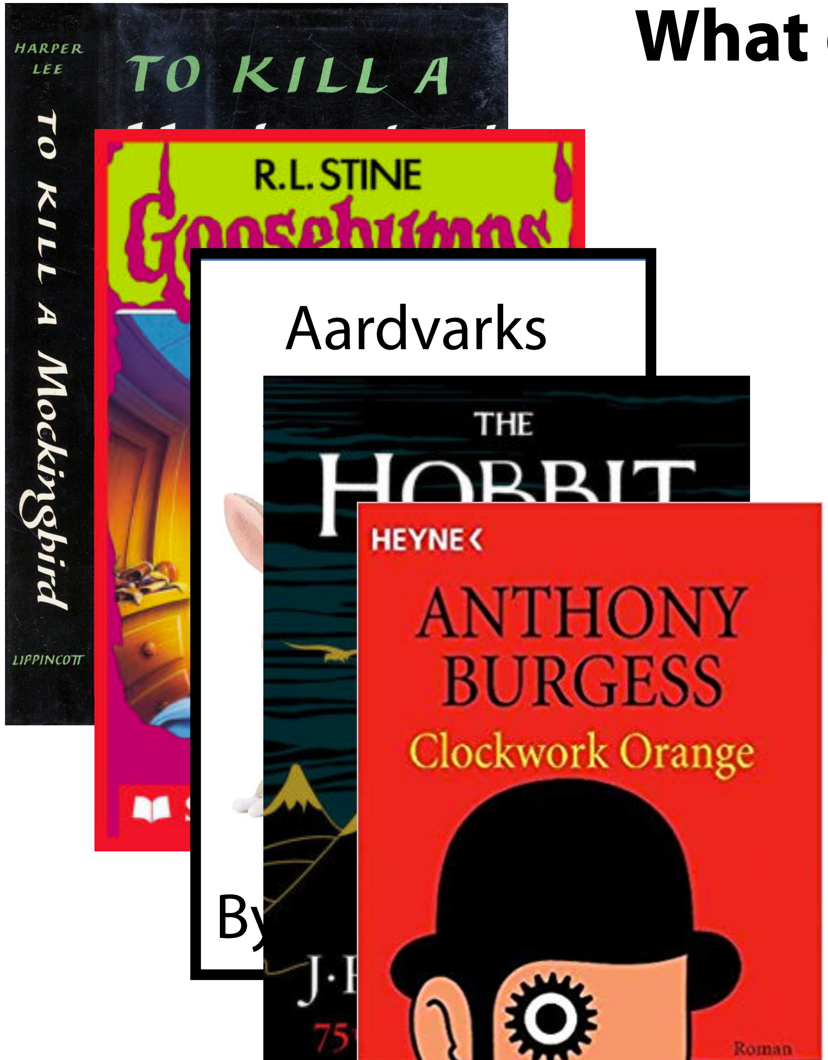
Discuss what a 'good' hash function looks like

Identify a key weakness of the hash table

Introduce strategies to 'correct' this weakness

Data Structure Review

I have a collection of books and I want to store them in a dictionary!



What data structures can I use here?

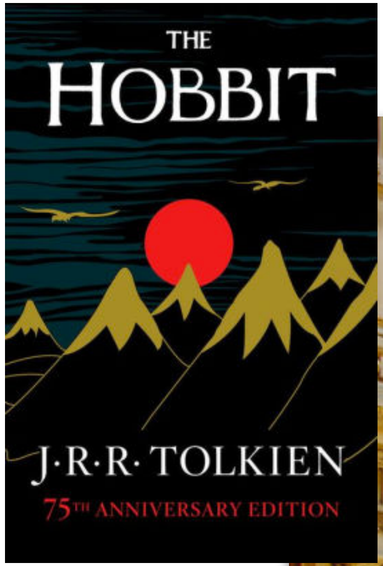
Key → Value

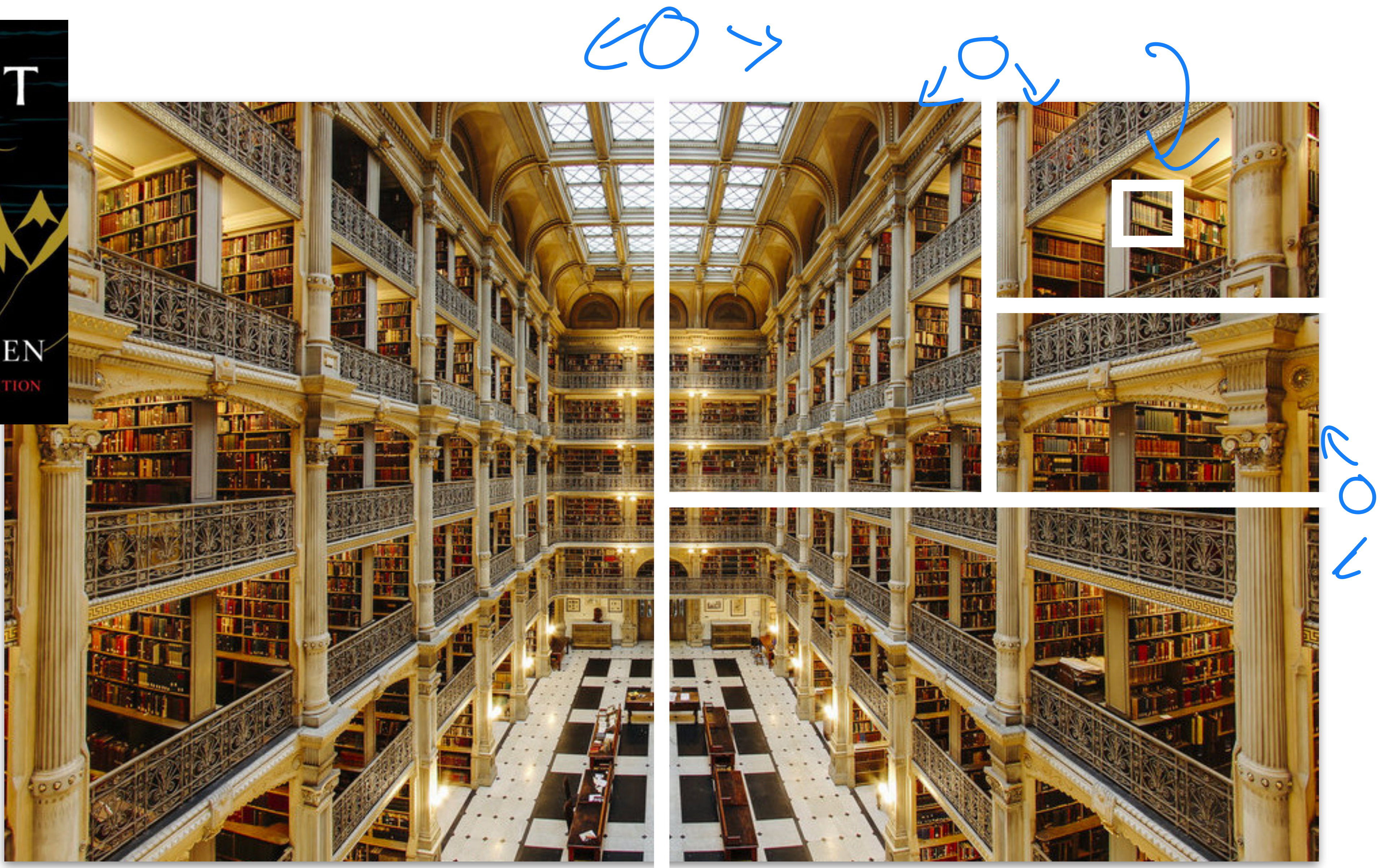
Book   contents

2 lists   keys ← match their indexing
values ←

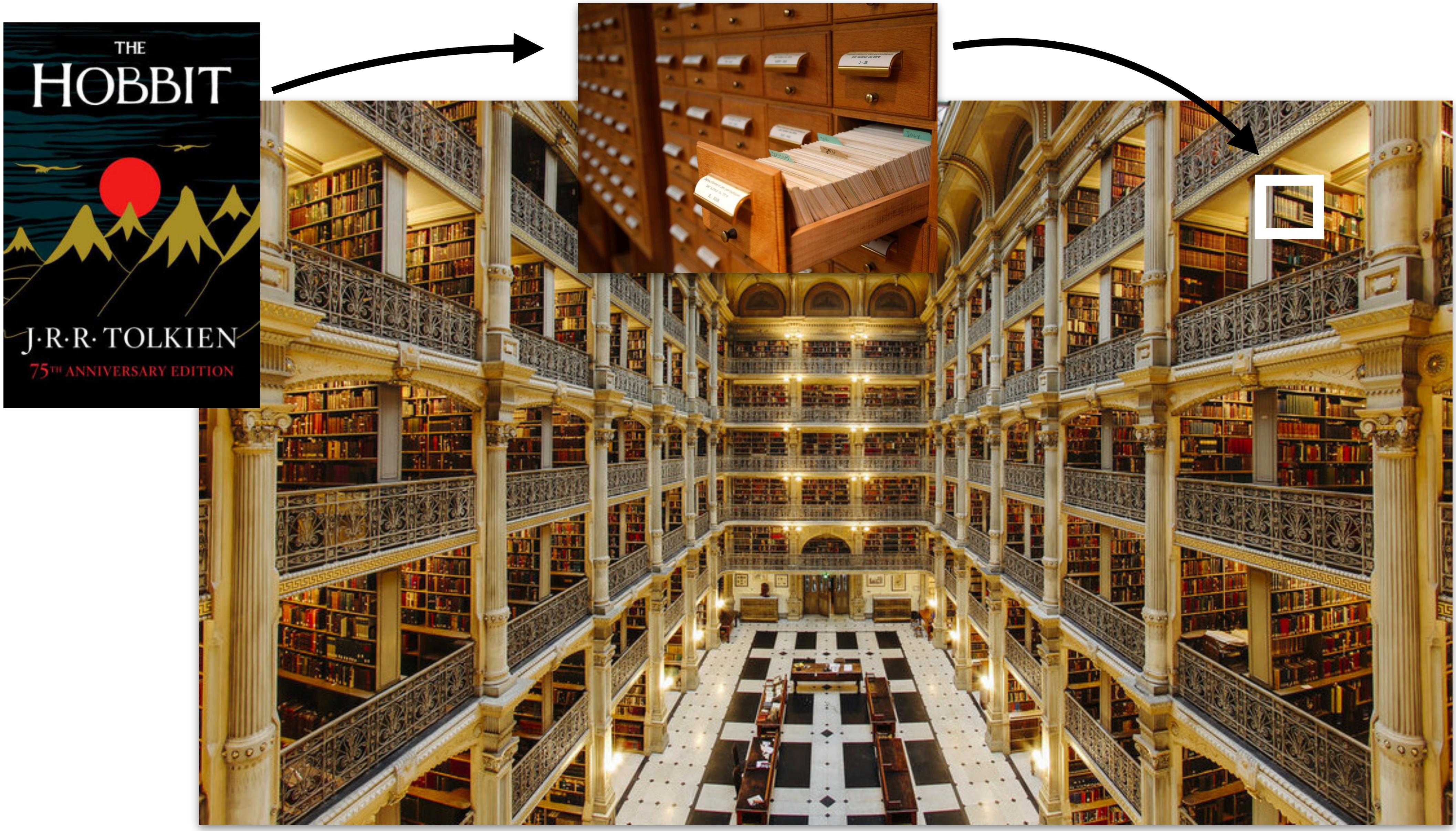
Tree (Binary Search Tree)

If we recognize that libraries are ordered: $O(\log n)$





What if $O(log\ n)$ isn't good enough?



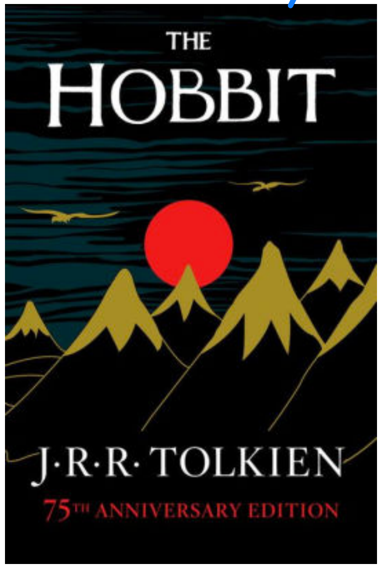
A Hash Table based Dictionary

/ Hash function

(address)

integer

Key



O(1)

Hash #
address

ISBN: 9781526602381

Call #: PR
6068.O93
H35 1998

O(1)

ISBN: 9781526602381

Call #: PR
6068.O93
H35 1998

Array
list

Chapter I

AN UNEXPECTED PARTY

In a hole in the ground there lived a hobbit. Not a nasty, dirty, wet hole, filled with the ends of worms and an oozy smell, nor yet a dry, bare, sandy hole with nothing in it to sit down on or to eat: it was a hobbit-hole, and that means comfort.

It had a perfectly round door like a porthole, painted green, with a shiny yellow brass knob in the exact middle. The door opened on to a tube-shaped hall like a tunnel: a very comfortable tunnel without smoke, with panelled walls, and floors tiled and carpeted, provided with polished chairs, and lots and lots of pegs for hats and coats—the hobbit was fond of visitors. The tunnel wound on and on, going fairly but not quite straight into the side of the hill—The Hill, as all the people for many miles round called it—and many little round doors opened out of it, first on one side and then on another. No going upstairs for the hobbit: bedrooms, bathrooms, cellars, pantries (lots of these), wardrobes (he had whole rooms devoted to clothes), kitchens, dining-rooms, all were on the same floor, and indeed on the same passage. The best rooms were all on the left-hand side (going in), for these were the only ones to have windows, deep-set round windows looking over his garden, and meadows beyond, sloping down to the river.

This hobbit was a very well-to-do hobbit, and his name

1

A Hash Table based Dictionary

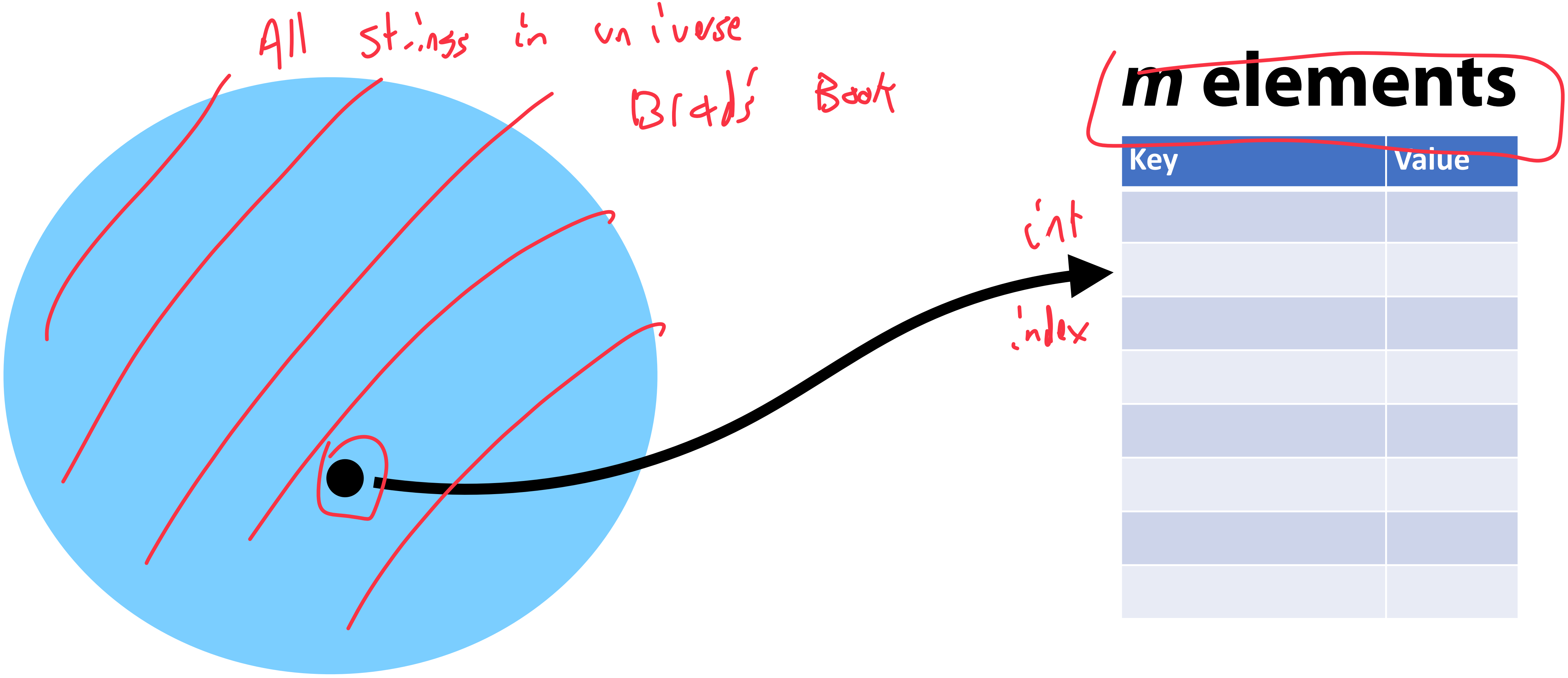
```python
d = {}
d[k] = v
print(d[k])
```

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

1. A hash function (key → int)
2. A list (stores our data @ int)
3. ???

Hash Function

Maps a **keyspace**, a (mathematical) description of the keys for a set of data, to a set of integers.



Hash Function

A hash function ***must*** be:

- **Deterministic:** Given same key twice, return same value
- **Efficient:** O(1)
- **Defined for a certain size table:** Universe → 0, ..., M-1

  M unique values

Hash Function

(Angrave, CS 241)
(Beckman, CS 421)
(Challon, CS 125)
(Davis, CS 101)
(Evans, CS 225)
(Fagen-Ulmschneider, CS 107)
(Gunter, CS 422)
(Herman, CS 233)

Hash function

K[0] – 'A'

A=0
B=1
C=2
...

	Key	Value
0	Angrave	241
	Beckman	421
	Challon	125
	...	

Hash Function

1 to 1 mapping of these specific names

to

this specific array

(Angrave, CS 241)

(Beckman, CS 421)

(Challon, CS 125)

(Davis, CS 101)

(Evans, CS 225)

(Fagen-Ulmschneider, CS 107)

(Gunter, CS 422)

(Herman, CS 233)

Akwini

Solomon 277

Hash function

(key[0] - ‘A’)

Key	Value
Angrave	241
Beckman	421
Challon	125
Davis	101
Evans	225
Fagen-U	107
Gunter	422
Herman	233

H ?

General Hash Function

An $O(1)$ deterministic operation that maps all keys in a universe U to a defined range of integers $[0, \ldots, m-1]$

- A **hash:** Function that converts any # in universe to a # (# could be any number)

  hash # % m

- A **compression:** Takes our # and converts to $[0, \ldots, m-1]$

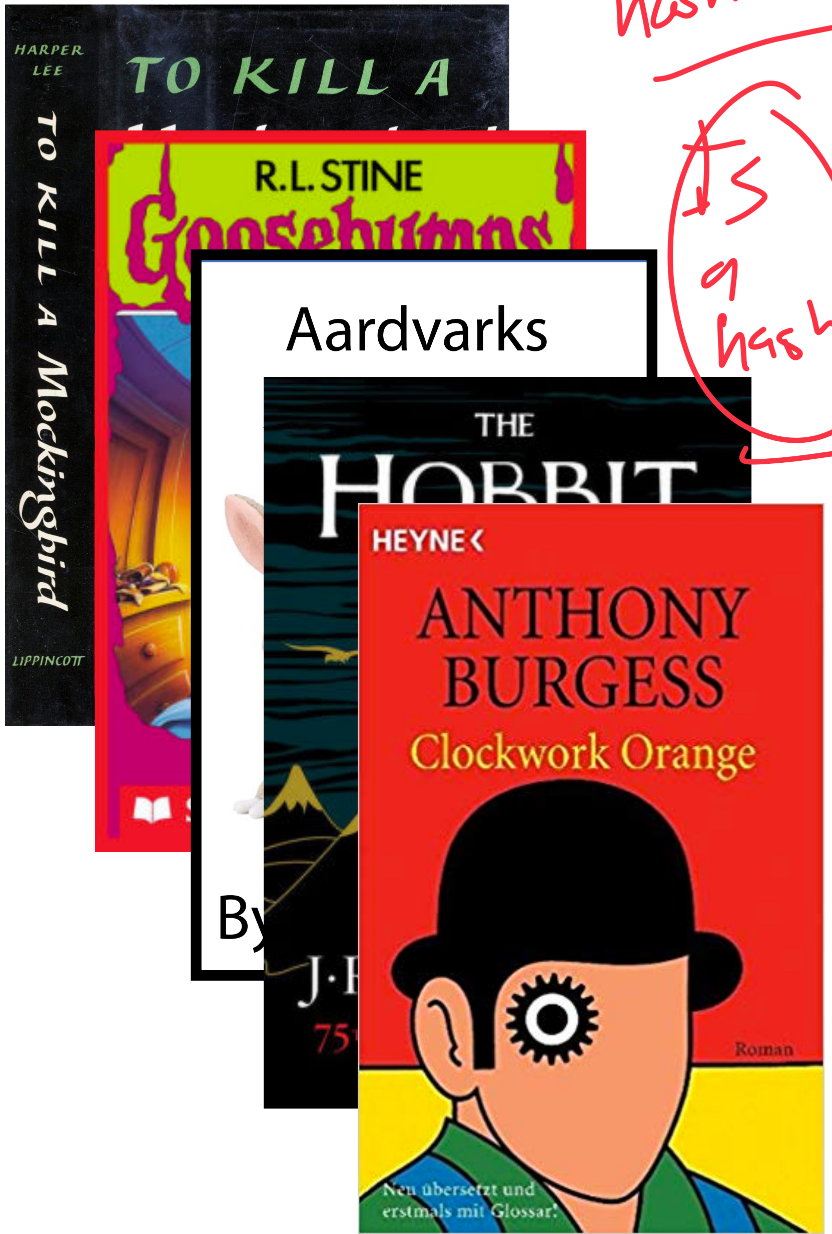
  $[0, 1] \leftarrow x \% 2$ (Modulo or Remainder)

Choosing a good hash function is tricky…

- Don’t create your own!

Hash Function

Is it hash function?



Is a hash

Authors 2 Authors

$$h(k) = (k.firstName[0] + k.lastName[0])\,\%\,m$$

↳ Deterministic ✓ ↳ {0, ..., m-1}

↳ Efficient ✓

↳ universe of all books? need work ???

$$h(k) = (rand() * k.numPages)\,\%\,m$$

↳ Not a hash function

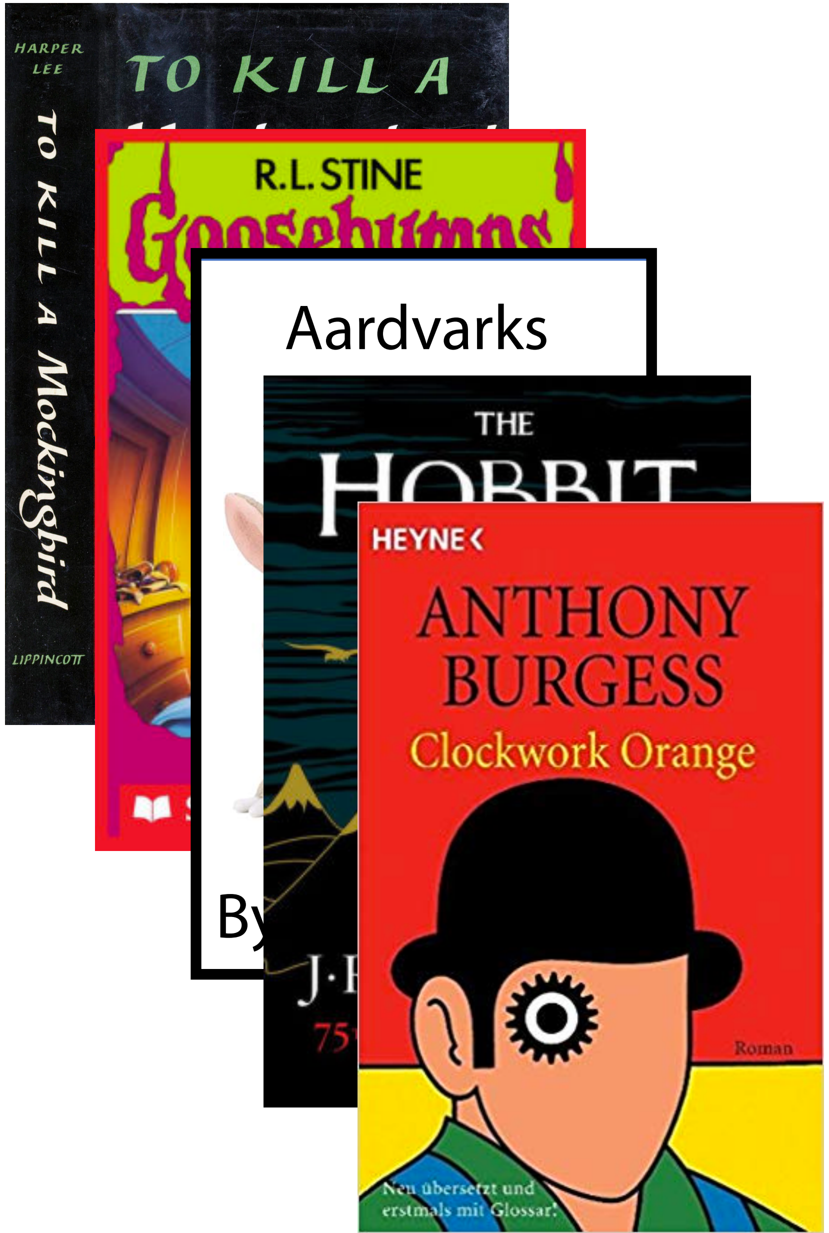
↳ Not deterministic

$$h(k) = (\text{Order I insert [Order seen]})\,\%\,m$$

↳ Not a hash function

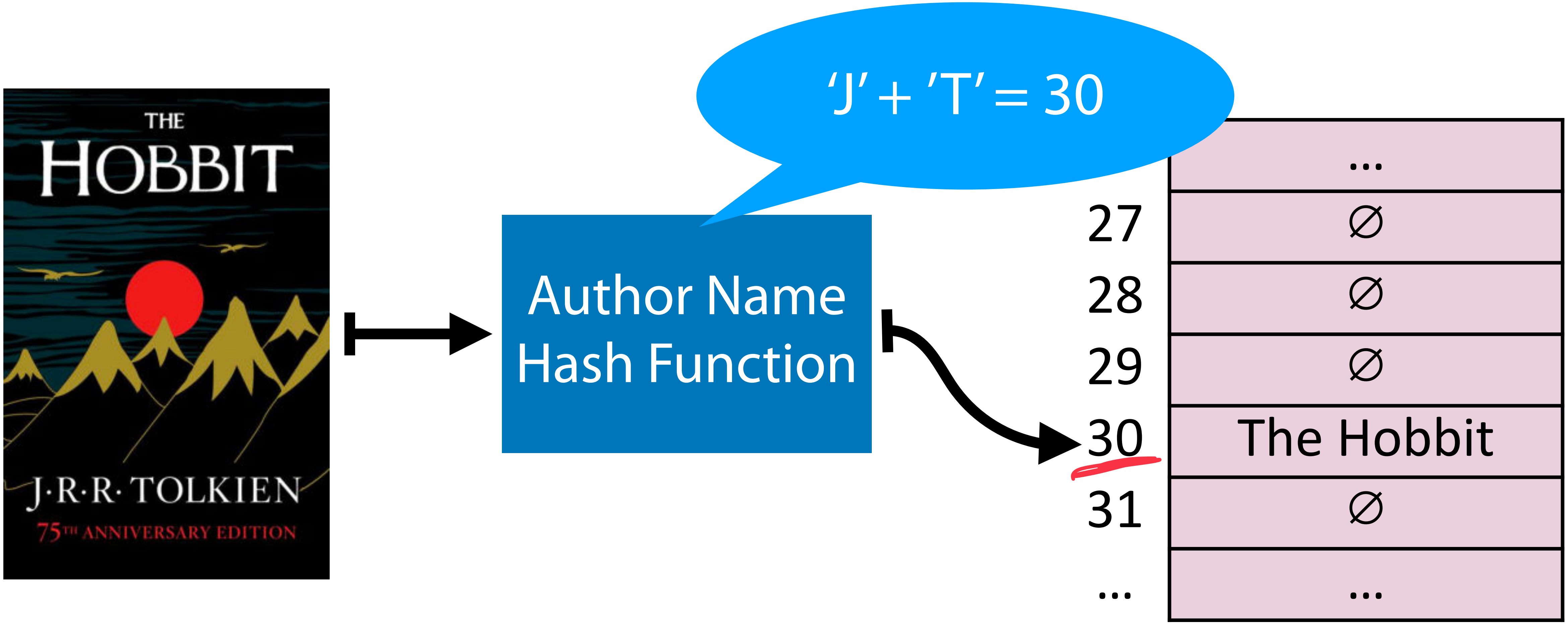
↳ Not deterministic

Hash Function

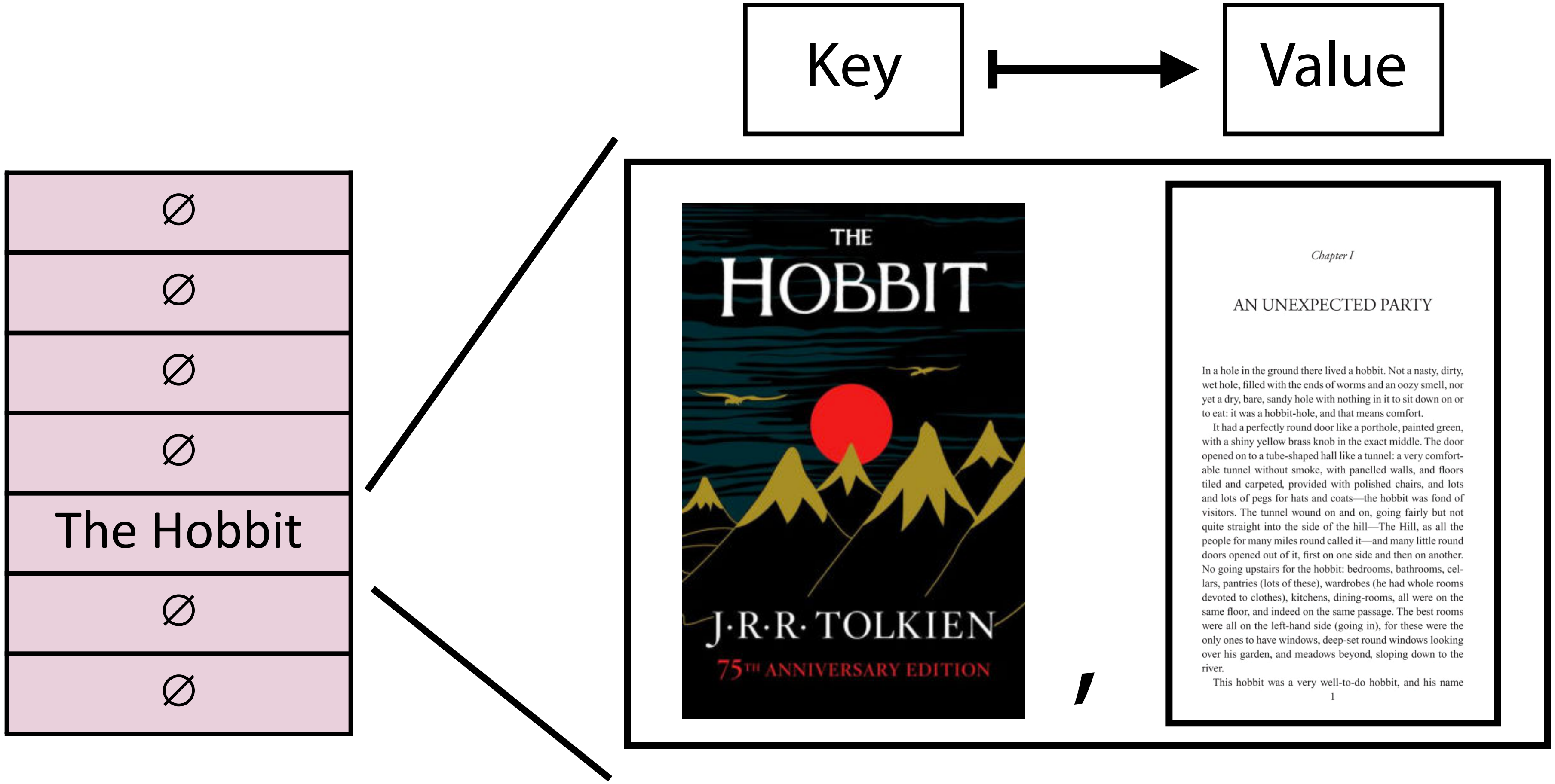


Exercise for viewer
↳ Some hash function for books

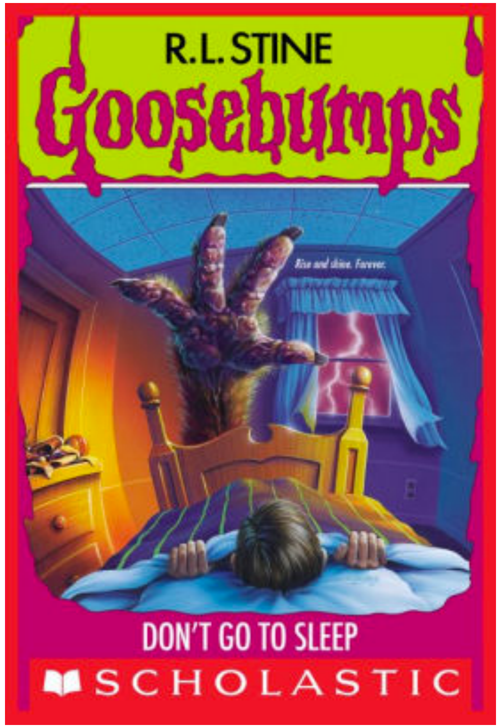
A Hash Table based Dictionary



A Hash Table based Dictionary



A Hash Table based Dictionary

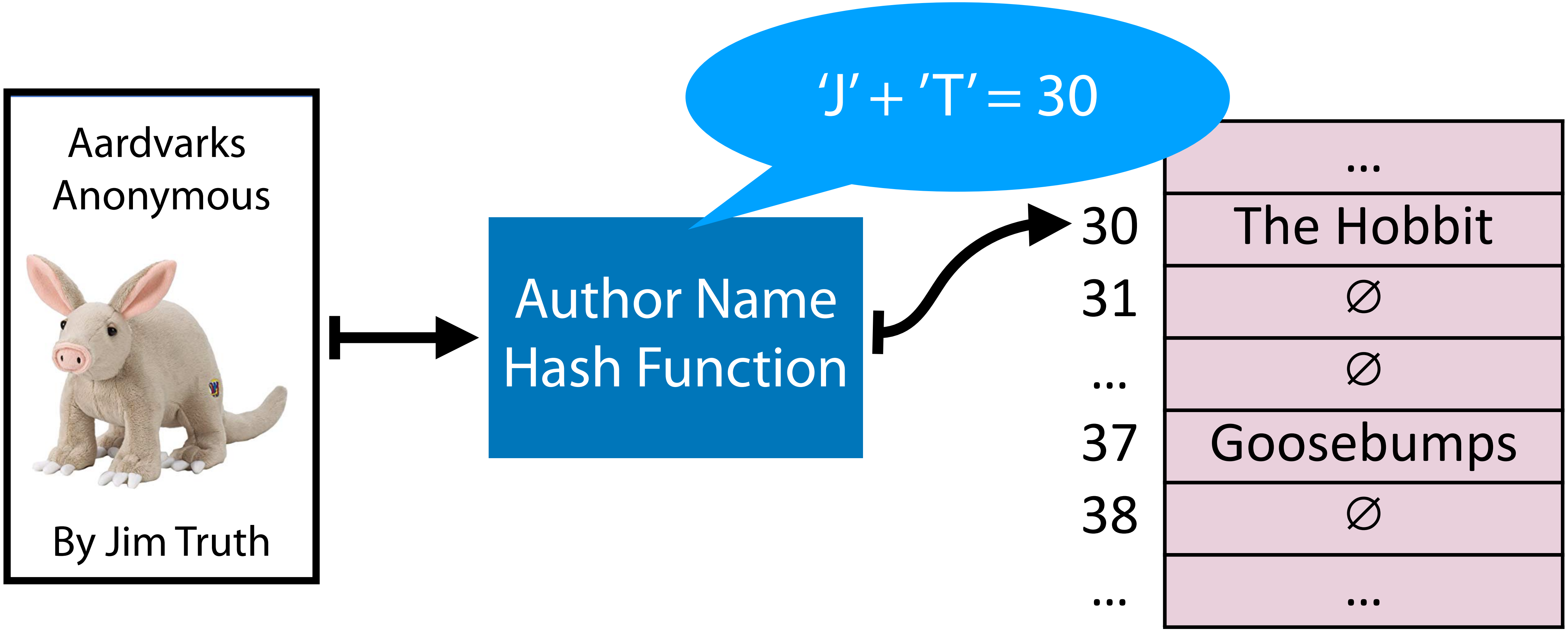


'R' + 'S' = 37

Author Name
Hash Function

	…
30	The Hobbit
31	∅
…	∅
37	Goosebumps
38	∅
…	…

A Hash Table based Dictionary

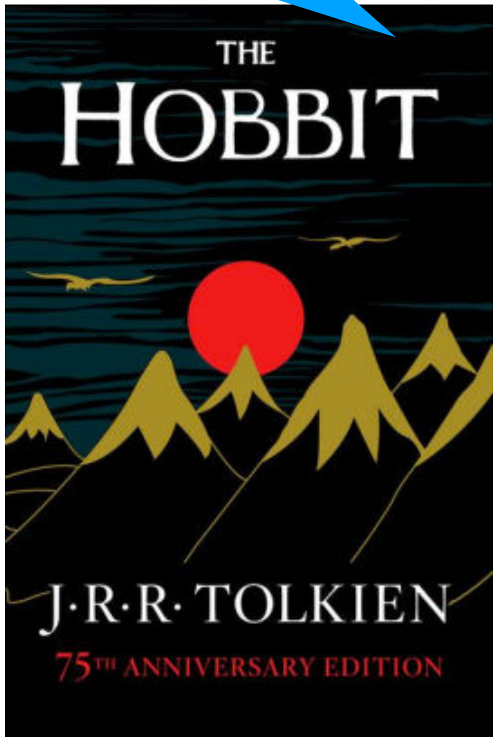


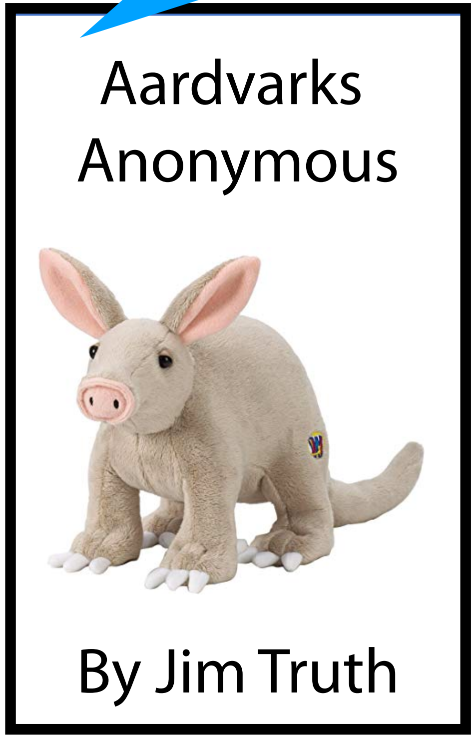
Hash Collision

A ***hash collision*** occurs when multiple unique keys hash to the same value

J.R.R Tolkien = 30!

Jim Truth = 30!

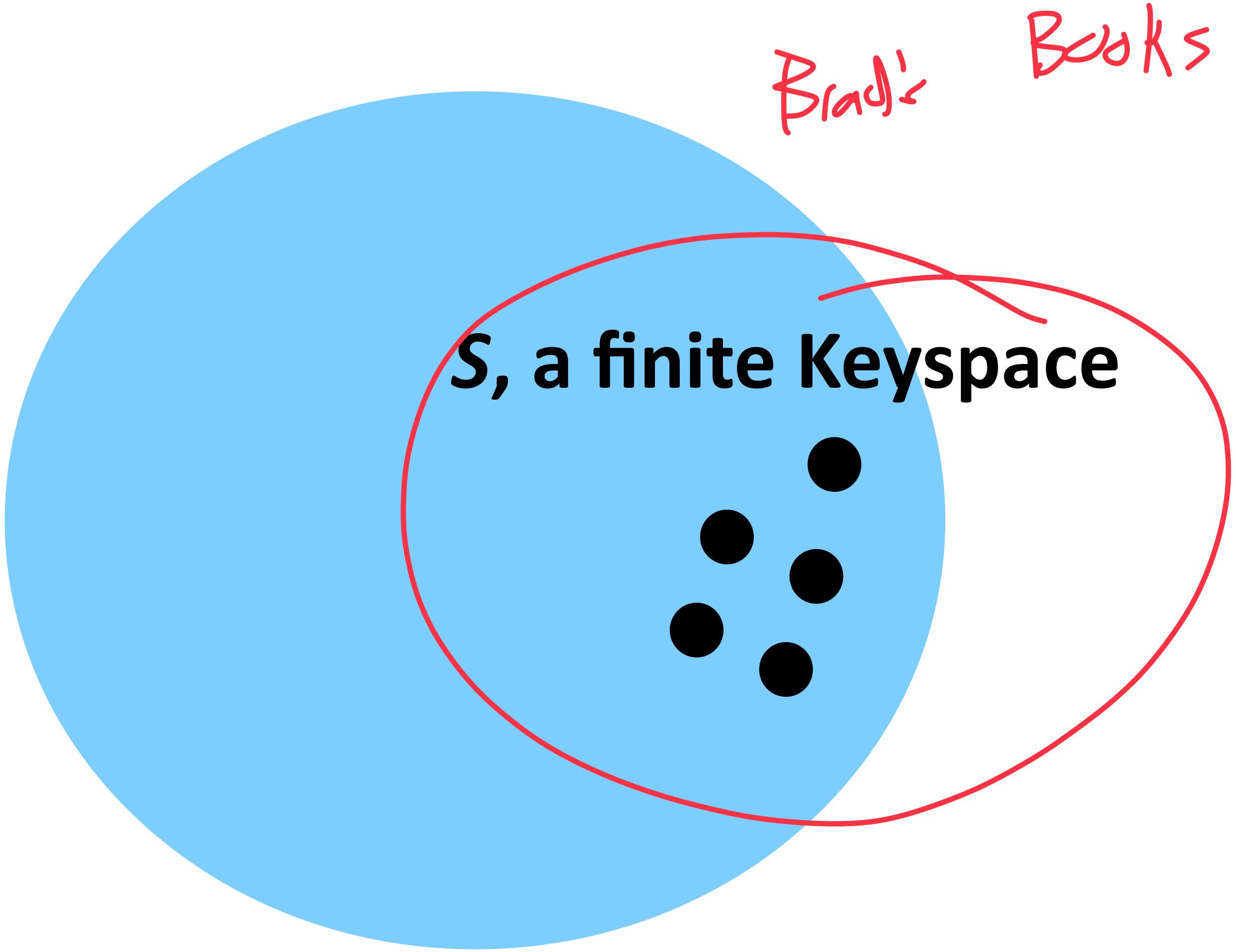




…	…
30	???
31	∅
…	∅
37	Goosebumps
38	∅
…	…

Perfect Hashing

If $m \geq S$, we can write a *perfect* hash with no collisions

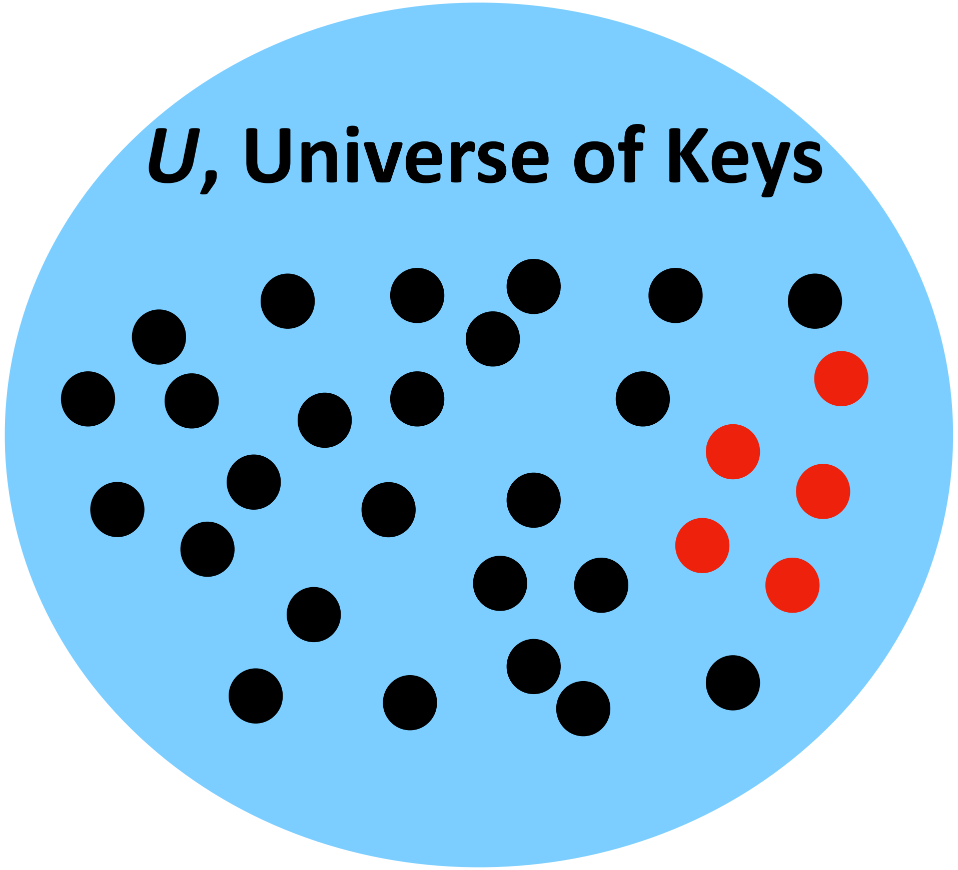


***m* elements**

Key	Value

General Purpose Hashing

In CS 277, we want our hash functions to work *in general*.

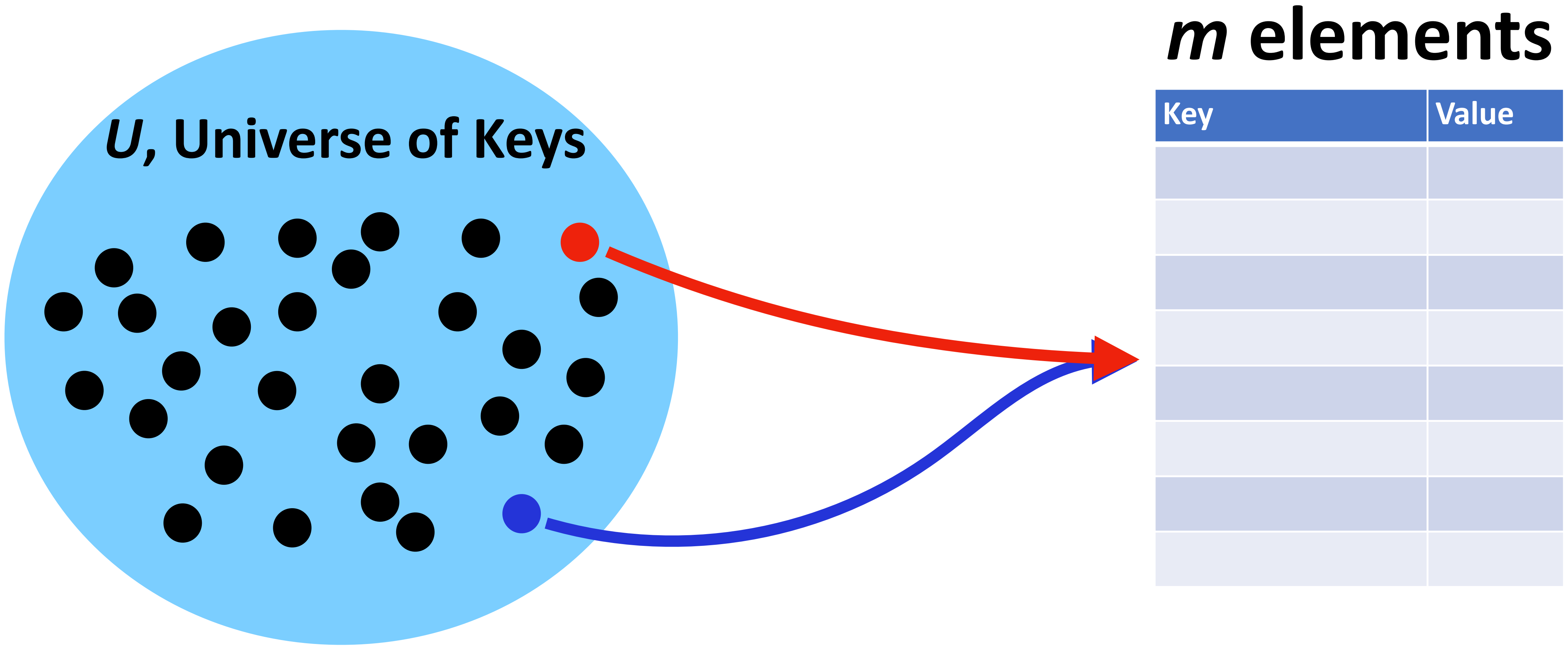


m elements

Key	Value

General Purpose Hashing

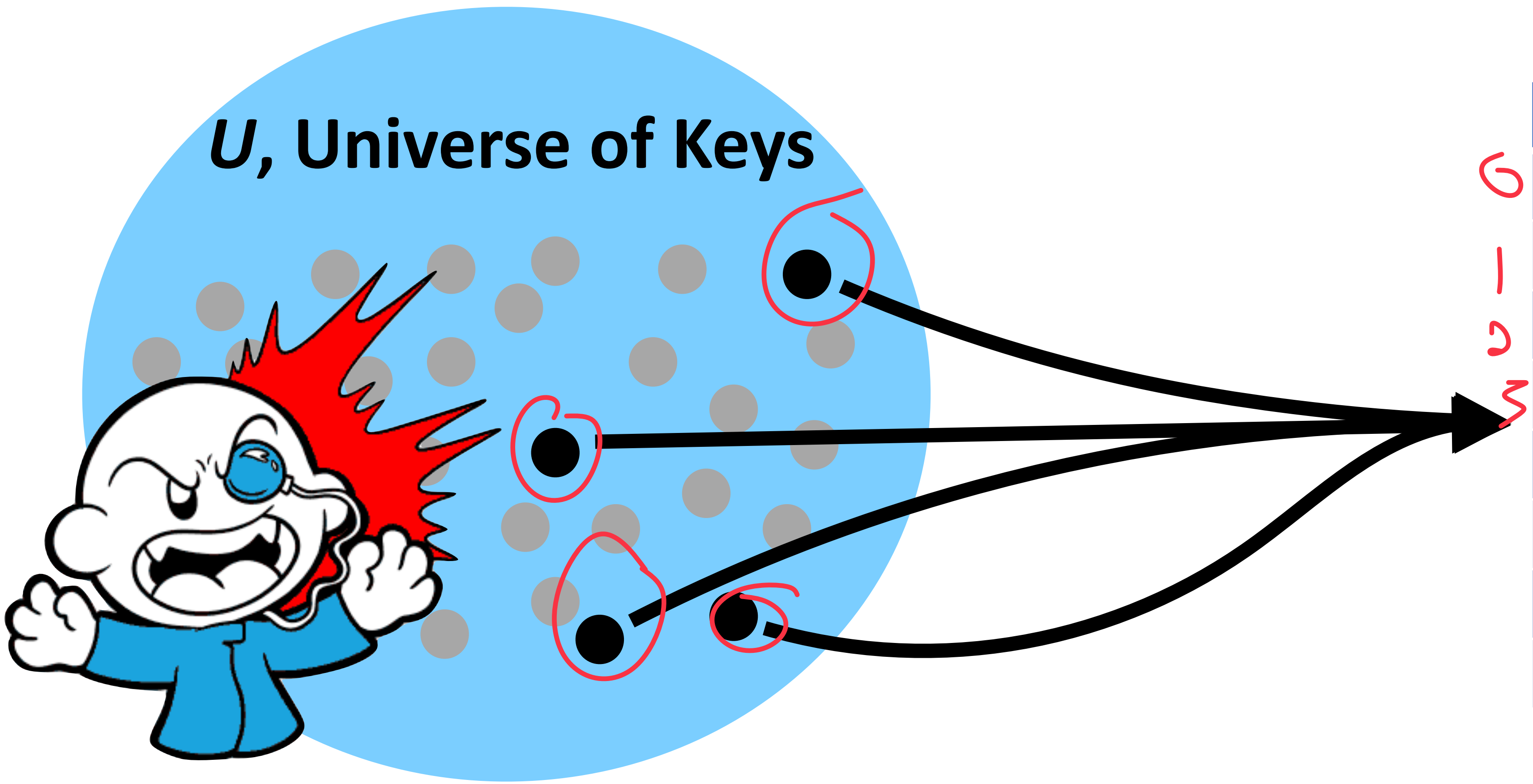
If $m < U$, there must be at least one hash collision.



General Purpose Hashing

By fixing h, we open ourselves up to adversarial attacks.

Data set



m elements

	Key	Value
0		
1		
2		
3		

Image by Matthew Loffhagen

A Hash Table based Dictionary

User Code (is a map):

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

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

1. A hash function

2. A data storage structure

3. A method of addressing *hash collisions*

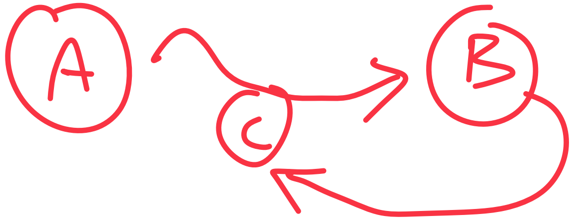
Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

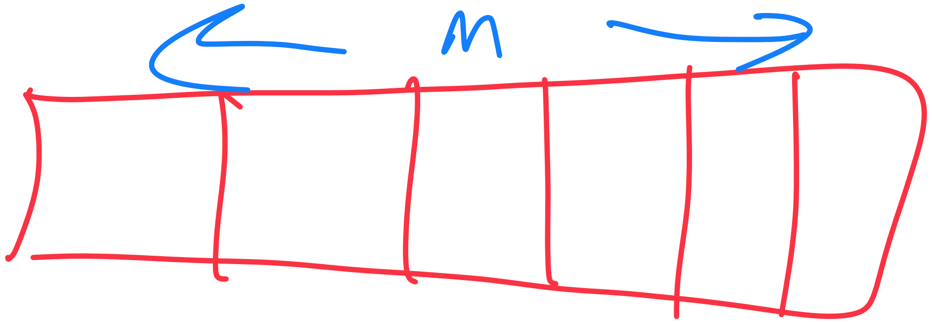
- **Open Hashing:** Stores key, value externally

  ↳ Linked list

  Can store individual nodes anywhere



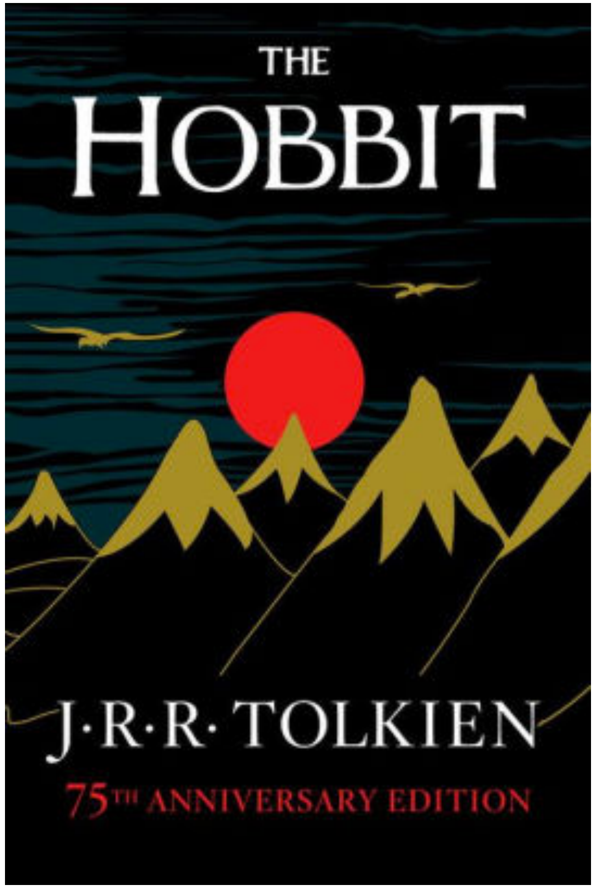
- **Closed Hashing:** Stores k, v internally

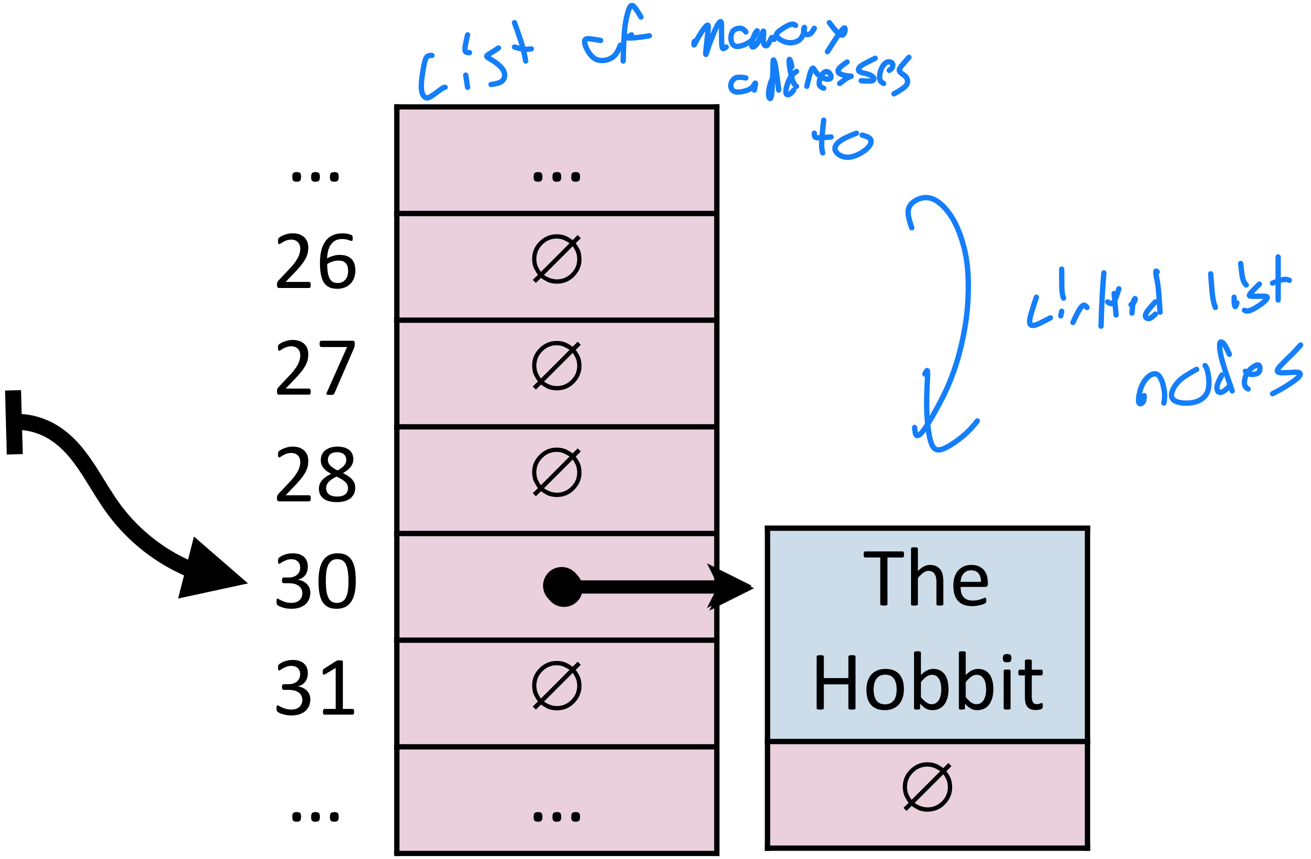


allocate fixed memory

Open Hashing

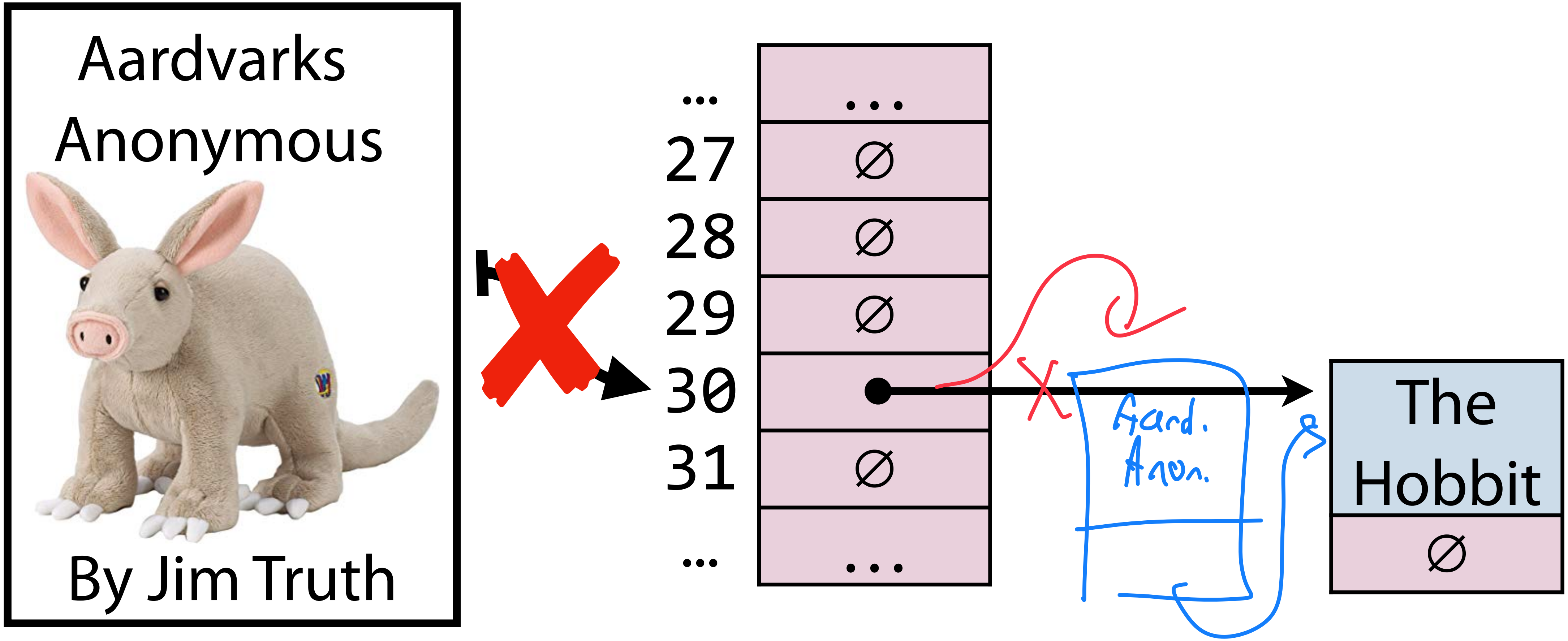
In an ***open hashing*** scheme, key-value pairs are stored externally (for example as a linked list).





Hash Collisions (Open Hashing)

A ***hash collision*** in an open hashing scheme can be resolved by adding to the linked list. This is called ***separate chaining***.



Insertion (Separate Chaining)

`_insert("Bob")`

`_insert("Anna")`

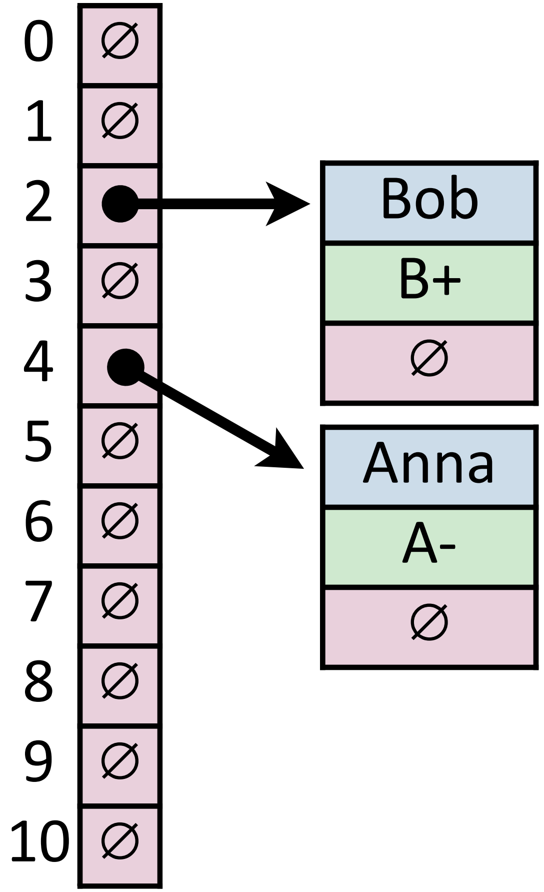
Key	Value	Hash
Bob	**B+**	**2**
Anna	**A-**	**4**
Alice	A+	4
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7

0	∅
1	∅
2	∅ → Bob / B+ / ∅ → None
3	∅
4	∅
5	∅
6	∅
7	∅
8	∅
9	∅
10	∅

Insertion (Separate Chaining)

`_insert("Alice")`

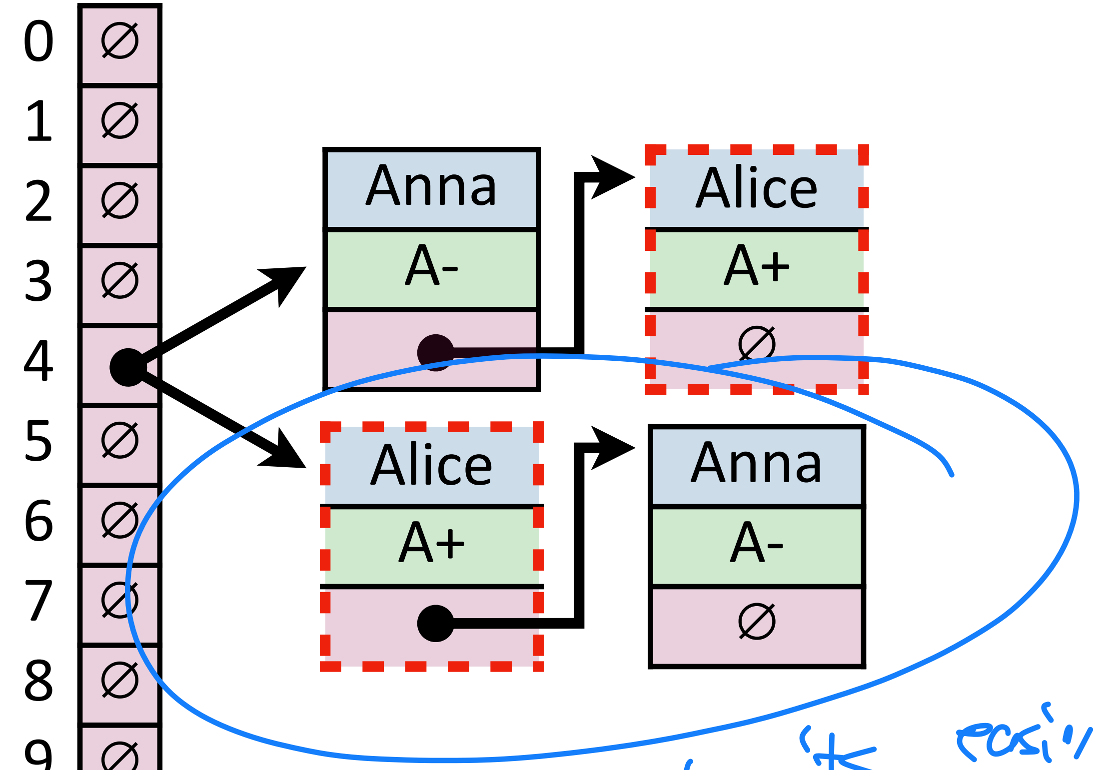
Key	Value	Hash
Bob	B+	2
Anna	A-	4
Alice	**A+**	**4**
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7



Insertion (Separate Chaining)

Where does Alice end up relative to Anna in the chain?

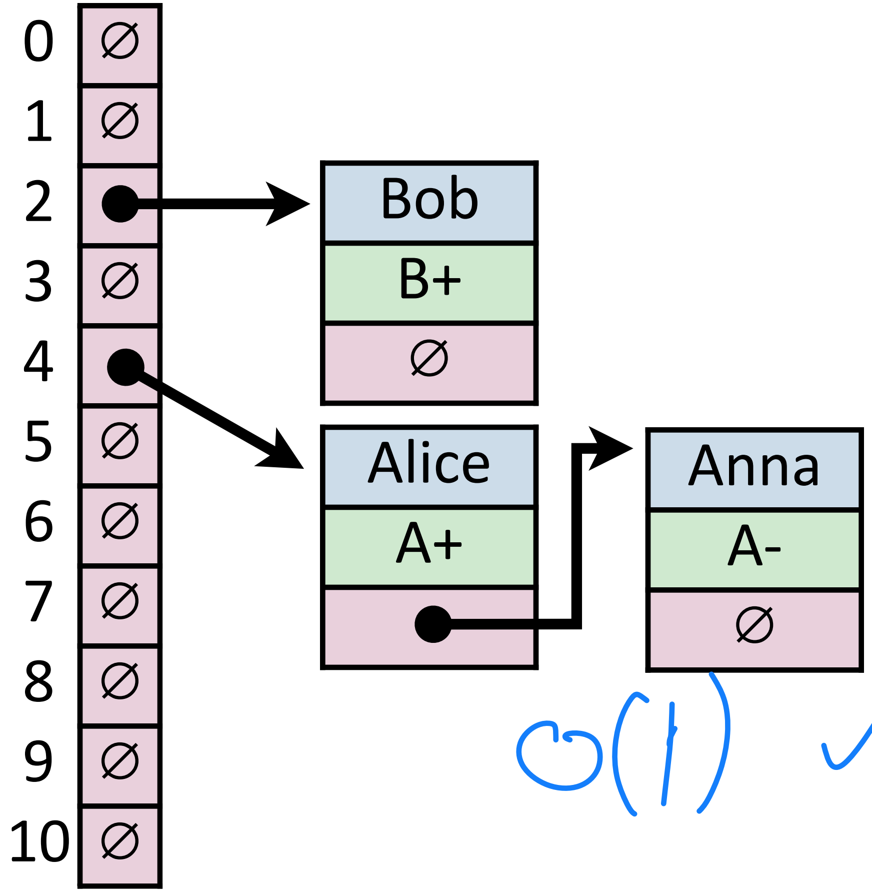
Key	Value	Hash
Bob	B+	2
Anna	A-	4
Alice	**A+**	**4**
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7



Before b/c it's easier to
insert at head / front

Insertion (Separate Chaining)

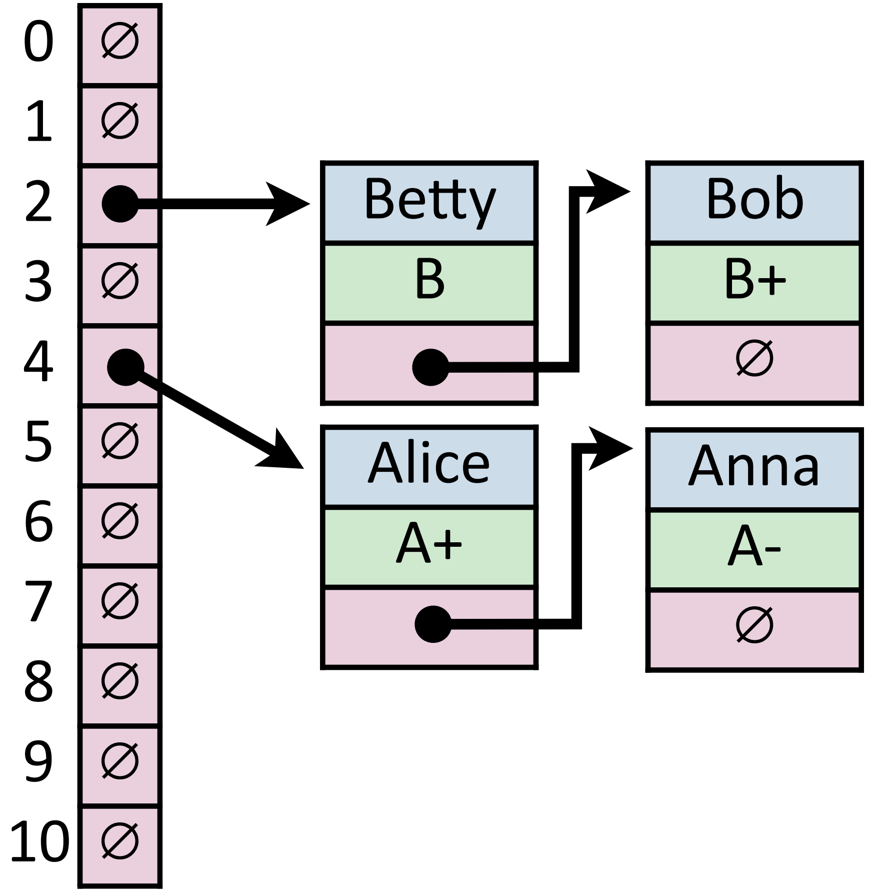
Key	Value	Hash
Bob	B+	2
Anna	A-	4
Alice	A+	4
Betty	**B**	**2**
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7



O(1) ✓

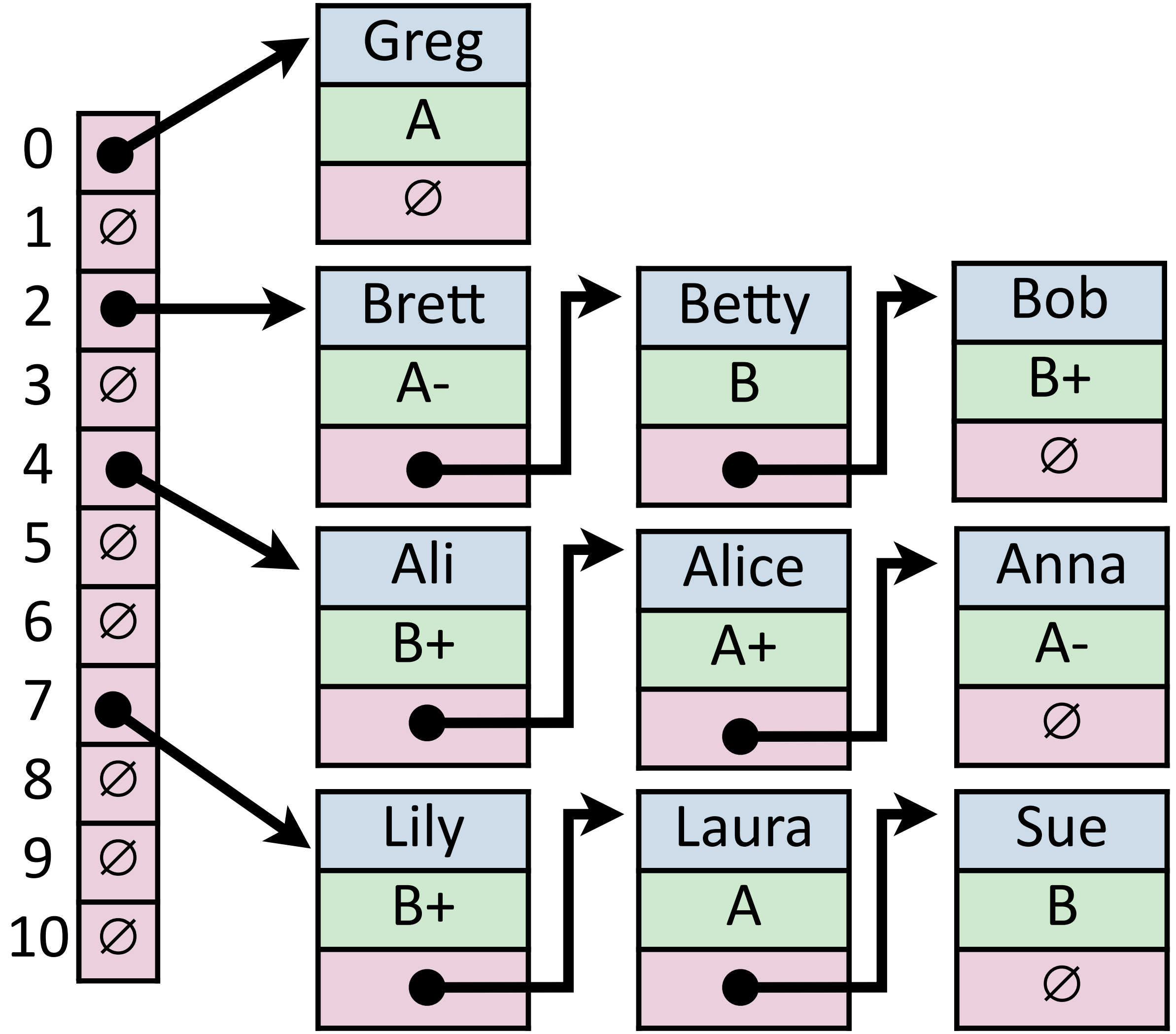
Insertion (Separate Chaining)

Key	Value	Hash
Bob	B+	2
Anna	A-	4
Alice	A+	4
Betty	**B**	**2**
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7



Insertion (Separate Chaining)

Key	Value	Hash
Bob	B+	2
Anna	A-	4
Alice	A+	4
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7



Find (Separate Chaining)

`_find("Sue")`

↳ O(n)

:(

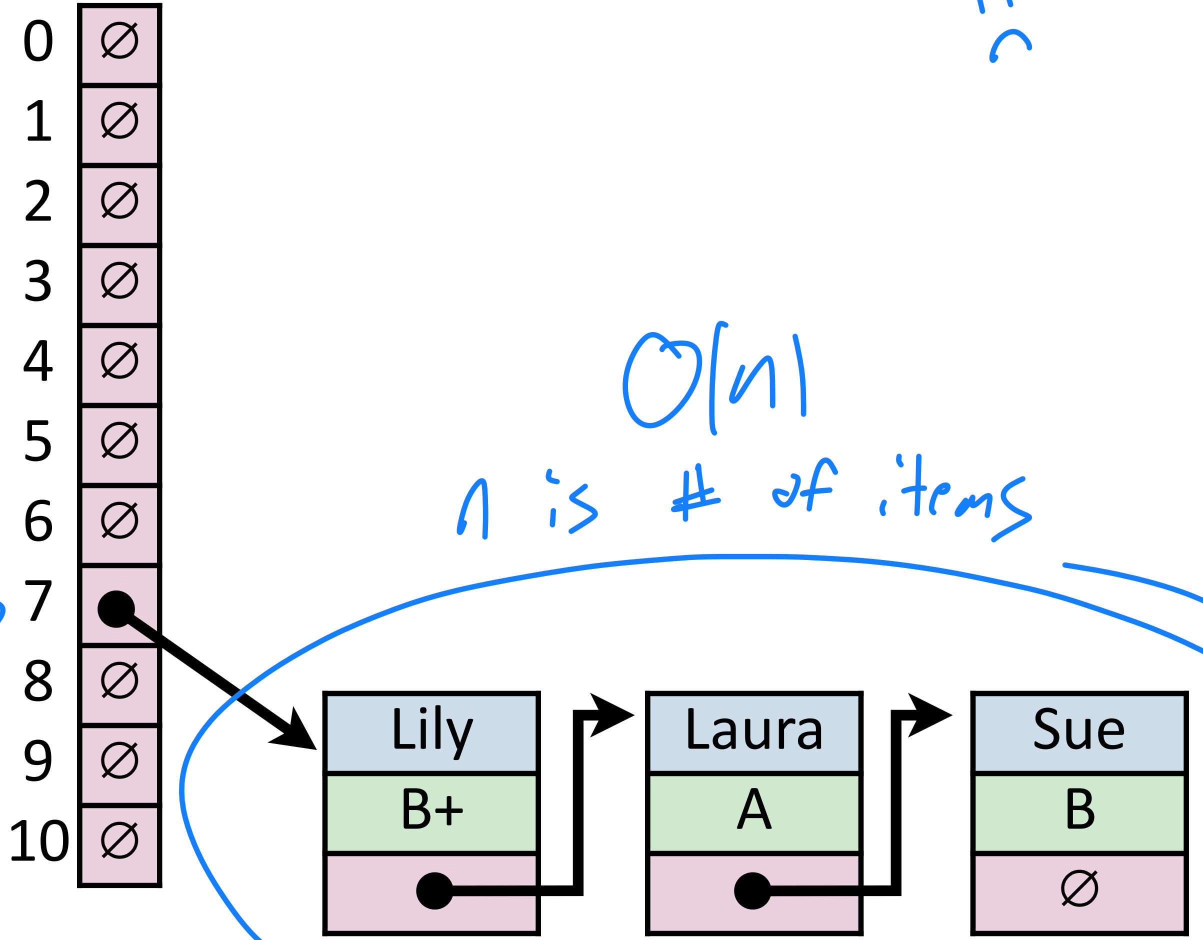
Key	Hash
Sue	7

Hash

O(1)

lookup

O(1)



O(n)

n is # of items

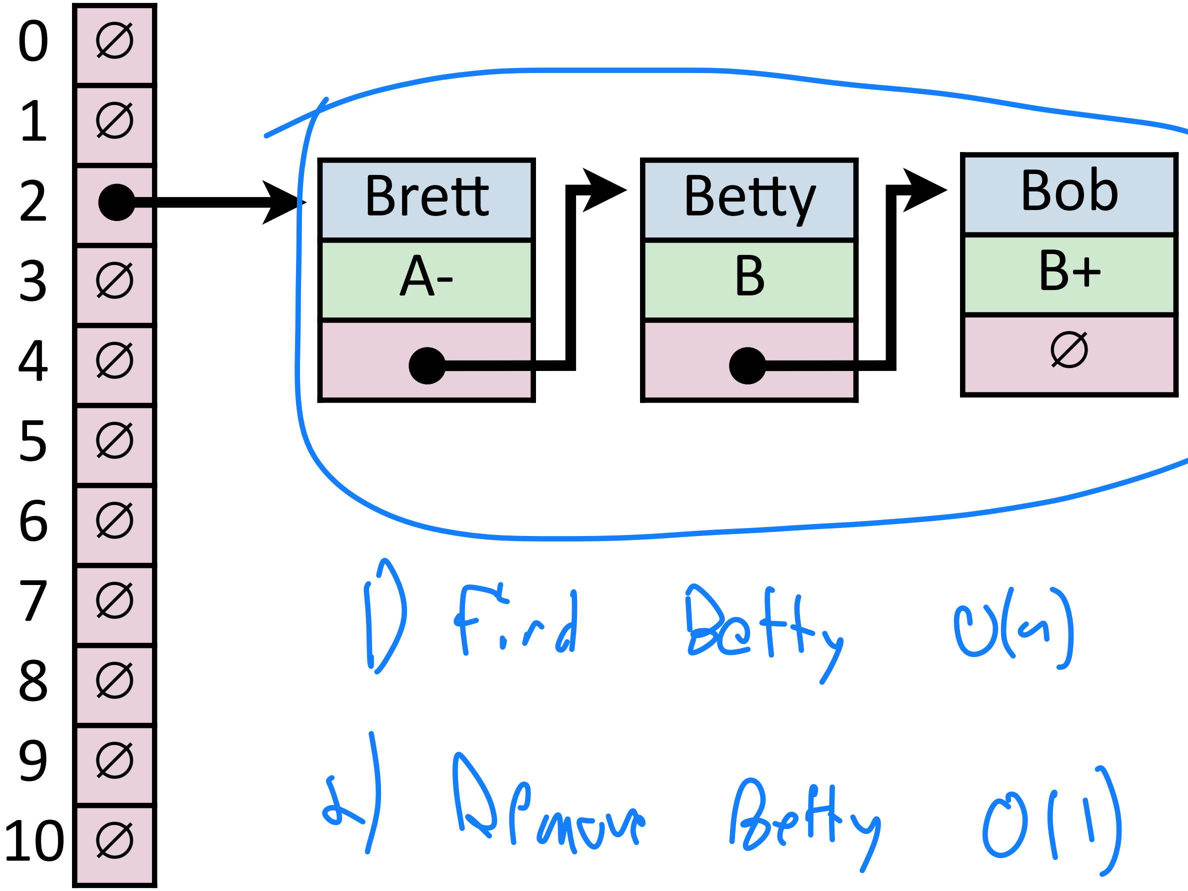
Remove (Separate Chaining)

`_remove("Betty")`

↳ O(n)

Key	Hash
Betty	2

O(1)



1) Find Betty O(n)

2) Remove Betty O(1)

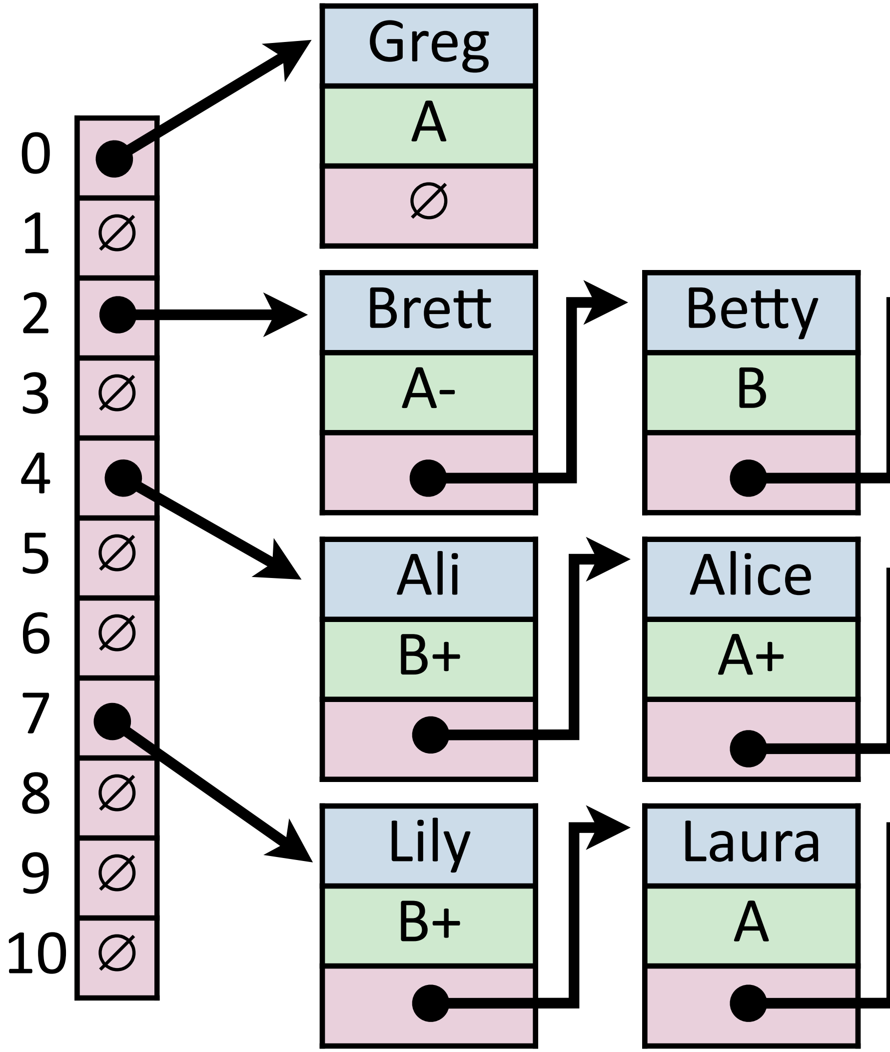
Hash Table (Separate Chaining)

For hash table of size *m* and *n* elements:

Find runs in: $O(n)$

Insert runs in: $O(1)$

Remove runs in: $O(n)$



Fundamentals of Probability

Imagine you roll a pair of six-sided dice.

The **sample space** Ω is the set of all possible outcomes.

	1	2	3	4	5	6
1	2	3	4	5	6	7
2						
3						
4						
5						
6						12

An **event** $E \subseteq \Omega$ is any subset.

↳ D1 rolls 1

↳ D1 & D2 roll 12

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

The **expectation** of a (discrete) random variable is:

$$E[X] = \sum_{x \in \Omega} \left(Pr\{X = x\} \cdot x \right.$$

Average expected outcome

$$\frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \ldots$$

$$= 3.5$$

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$$E[X + Y] = E[X] + E[Y]$$

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$E[X + Y] = E[X] + E[Y]$

Prob

value

$$= \underbrace{\sum_x \sum_y}\ \underbrace{Pr\{X = x, Y = y\}}(x + y)$$

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$$E[X + Y] = E[X] + E[Y]$$

$$= \sum_x \sum_y Pr\{X = x, Y = y\}(x + y)$$

$$= \sum_x x \sum_y Pr\{X = x, Y = y\} + \sum_y y \sum_x Pr\{X = x, Y = y\}$$

Sums up to 1

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$$E[X+Y] = E[X] + E[Y]$$

$$= \sum_x \sum_y Pr\{X = x, Y = y\}(x + y)$$

$$= \sum_x x \sum_y Pr\{X = x, Y = y\} + \sum_y y \sum_x Pr\{X = x, Y = y\}$$

$$= \sum_x x \cdot Pr\{X = x\} + \sum_y y \cdot Pr\{Y = y\}$$

Fundamentals of Probability

Imagine you roll a pair of six-sided dice. What is the expected value?

Linearity of Expectation: For any two random variables X and Y,

$$E[X + Y] = E[X] + E[Y]$$

3.5 + 3.5

= 7

Hash Table

Worst-Case behavior is bad — but what about randomness?

1) **Fix *h***, our hash, and assume it is good **for *all keys***:

↳ simple uniform hash assumption

↳ every o

2) Create a ***universal hash function family:***

Simple Uniform Hashing Assumption (SUHA)

Given table of size m, a simple uniform hash, h, implies

$$\forall k_1, k_2 \in U \text{ where } k_1 \neq k_2 \,,\ \underbrace{Pr(h[k_1] = h[k_2])}_{\text{same hash value}} = \boxed{\frac{1}{m}}$$

m is table size

Uniform: Is a flat line

↳ everything is equally likely to hash to every position

Independent:

↳ Every item hashes independently of every other item

Separate Chaining Under SUHA

Table Size: m

Num objects: n

Claim: Under SUHA, expected length of chain is $\frac{n}{m}$

$\alpha_j =$ expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} \text{1 if item i hashes to j} \\ \text{0 otherwise} \end{cases}$$

Separate Chaining Under SUHA

Table Size: m

Num objects: n

Claim: Under SUHA, expected length of chain is $\dfrac{n}{m}$

α_j = expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} \text{1 if item i hashes to j} \\ \text{0 otherwise} \end{cases}$$

$$E[\alpha_j] = E\Big[\sum_i H_{i,j}\Big]$$

Separate Chaining Under SUHA

Table Size: m

Num objects: n

Claim: Under SUHA, expected length of chain is $\frac{n}{m}$

$\alpha_j =$ expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j} \qquad H_{i,j} = \begin{cases} \text{1 if item i hashes to j} \\ \text{0 otherwise} \end{cases}$$

$$E[\alpha_j] = E\left[\sum_i H_{i,j}\right]$$

$$E[\alpha_j] = \sum_i Pr(H_{i,j} = 1) * 1 + \cancel{Pr(H_{i,j} = 0) * 0}$$

Separate Chaining Under SUHA

Table Size: m

Num objects: n

Claim: Under SUHA, expected length of chain is $\frac{n}{m}$

$\alpha_j =$ expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} 1 \text{ if item i hashes to j} \\ 0 \text{ otherwise} \end{cases}$$

$$E[\alpha_j] = E[\sum_i H_{i,j}]$$

$\frac{1}{m}$

$$E[\alpha_j] = \sum_i Pr(H_{i,j} = 1) * 1 + Pr(H_{i,j} = 0) * 0$$

$$E[\alpha_j] = n * Pr(H_{i,j} = 1) \cdot 1$$

Separate Chaining Under SUHA

Table Size: m

Num objects: n

Claim: Under SUHA, expected length of chain is $\frac{n}{m}$

$\alpha_j =$ expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} \text{1 if item i hashes to j} \\ \text{0 otherwise} \end{cases}$$

$$Pr[H_{i,j} = 1] = \frac{1}{m}$$

$$E[\alpha_j] = E\left[\sum_i H_{i,j}\right]$$

$$E[\alpha_j] = n * Pr(H_{i,j} = 1)$$

Separate Chaining Under SUHA

Claim: Under SUHA, expected length of chain is $\frac{n}{m}$

Table Size: m

Num objects: n

α_j = expected # of items hashing to position j

$$\alpha_j = \sum_i H_{i,j}$$

$$H_{i,j} = \begin{cases} \text{1 if item i hashes to j} \\ \text{0 otherwise} \end{cases}$$

$$E[\alpha_j] = E[\sum_i H_{i,j}]$$

$$Pr[H_{i,j} = 1] = \frac{1}{m}$$

$$E[\alpha_j] = n * Pr(H_{i,j} = 1)$$

$$\boxed{\mathbf{E}[\alpha_\mathbf{j}] = \frac{\mathbf{n}}{\mathbf{m}}}$$

Separate Chaining Under SUHA

Under SUHA, a hash table of size *m* and *n* elements:

Find runs in: $O(1+\alpha)$.

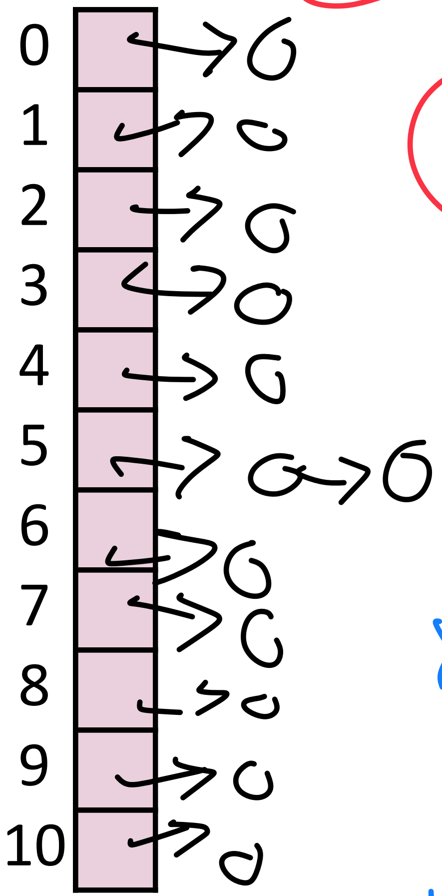
$\alpha = n/m$

Insert runs in: $O(1)$.

$O(1)^*$

* – we control constant factors

Remove runs in: $O(1+\alpha)$.



Expected # of items is

n/m

is a constant

$X = n/m$

$m = n/X$

You give me n & I set m to be good

Separate Chaining Under SUHA

Pros:

Cons:

Next time: Closed Hashing

Closed Hashing: store *k,v* pairs in the hash table

S = { 1, 8 , 15}

h(k) = k % 7

0	
1	1
2	
3	
4	
5	
6	

8?