# CS 225

**Data Structures**

March 21 – Hashing

Brad Solomon

# Active Assignments

lab_huffman due March 21st (today!)

mp_traversal due March 28th

Don’t forget about your final project!

Exam 2 (March 22 — 24)

# Exam Review Session

Main topics of exam:

Trees (Traversal, tree properties, search strategies)

Binary Search Trees (Find, insert, delete, tree properties)

AVL Trees (Find, insert, delete, tree properties, rotations)

B-Trees (Structural properties)

# Team Contract and Proposal Due March 25th

**Team Contract:**

Be sure to ‘sign’ electronically.

Non-participants will not get credit (see grading rubric)!

**Project Proposal:**

One of your three algorithms should be completed by ***mid-project*** check-in.

# Learning Objectives

Motivate and formally define a hash table

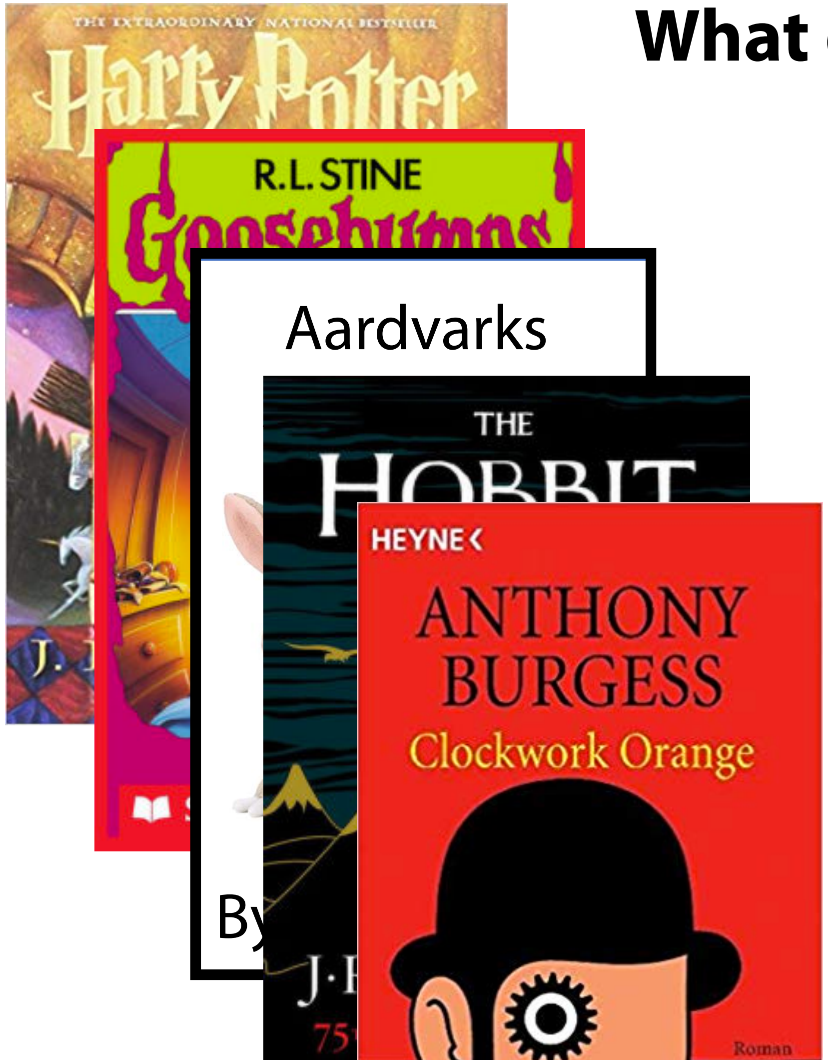
Discuss what a ‘good’ hash function looks like

Identify the key weakness of a hash table

Introduce strategies to “correct” this weakness

# Data Structure Review

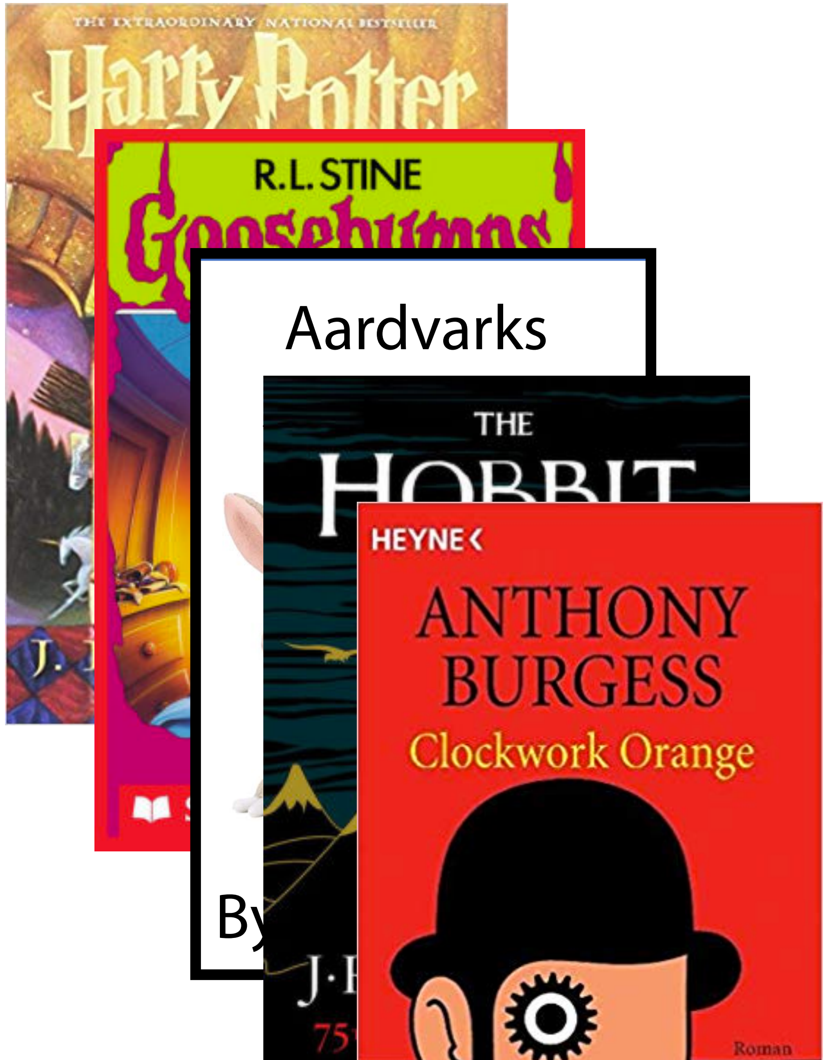
I have a collection of books and I want to store them in a dictionary!




**What data structures can I use here?**

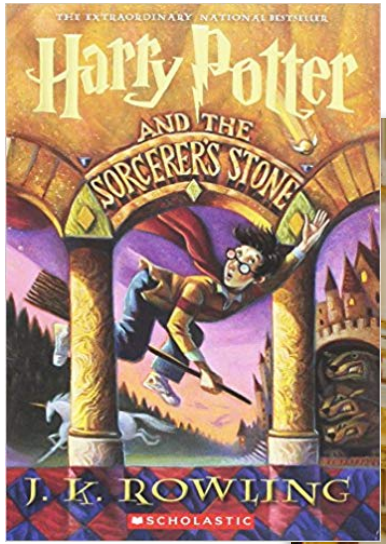
# Data Structure Review

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

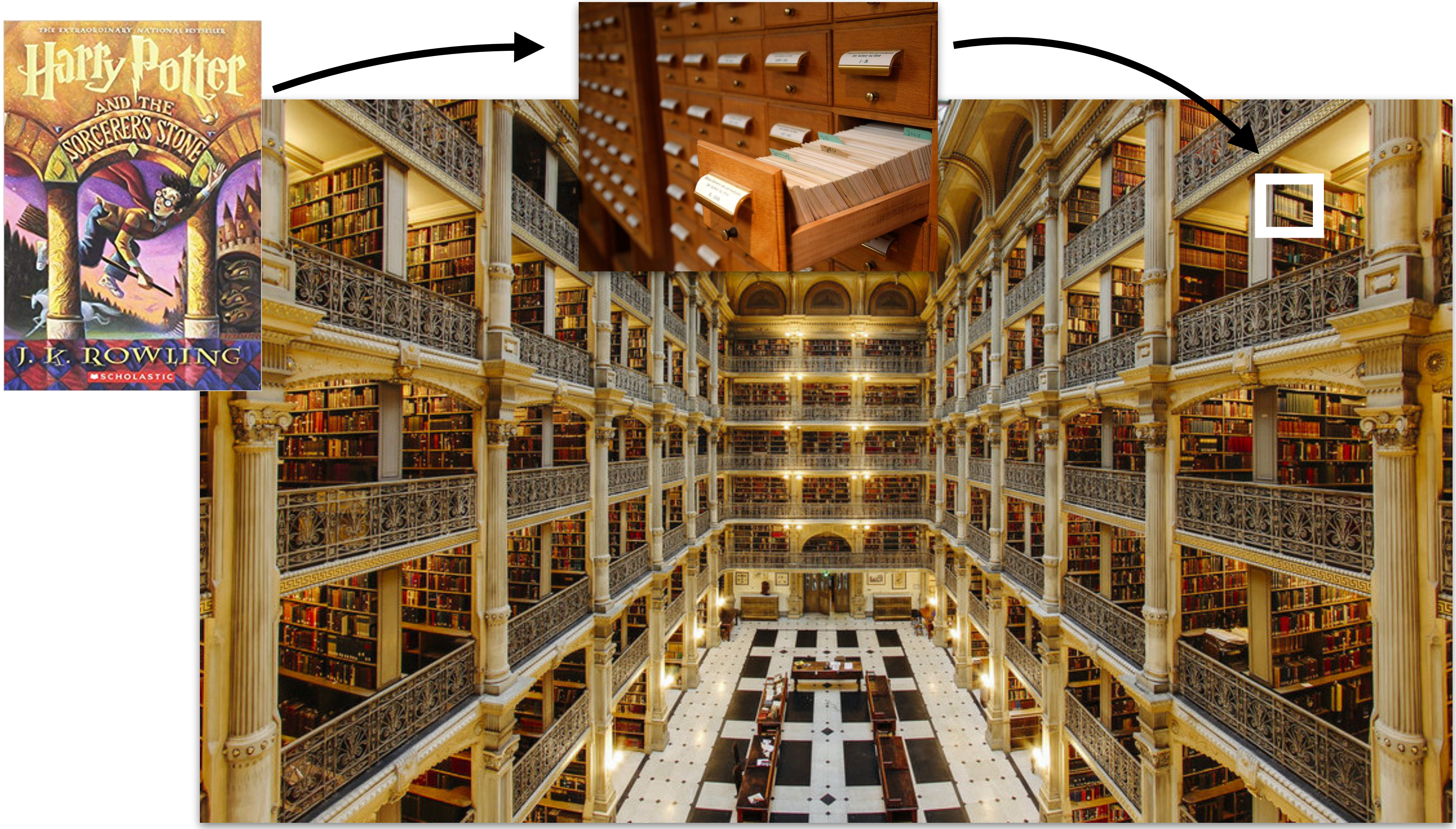



| | Sorted Array | BST | AVL Tree |
|---|---|---|---|
| Find | | | |
| Insert | | | |
| Remove | | | |

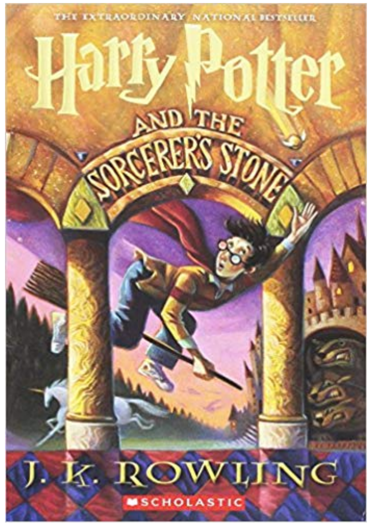
# What if $O(log\ n)$ isn't good enough?




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



# A Hash Table based Dictionary



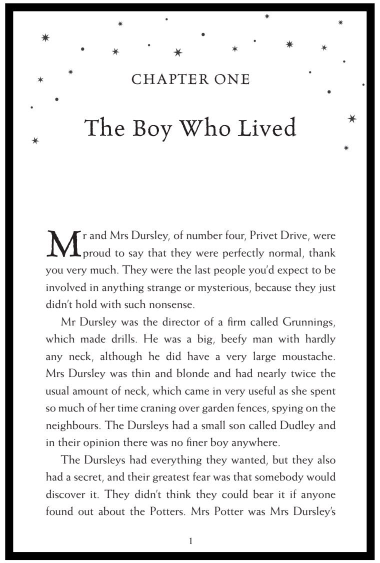


ISBN: 9781526602381

Call #: PR
6068.O93
H35 1998

ISBN: 9781526602381

Call #: PR
6068.O93
H35 1998



CHAPTER ONE

The Boy Who Lived

Mr and Mrs Dursley, of number four, Privet Drive, were proud to say that they were perfectly normal, thank you very much. They were the last people you'd expect to be involved in anything strange or mysterious, because they just didn't hold with such nonsense.

Mr Dursley was the director of a firm called Grunnings, which made drills. He was a big, beefy man with hardly any neck, although he did have a very large moustache. Mrs Dursley was thin and blonde and had nearly twice the usual amount of neck, which came in very useful as she spent so much of her time craning over garden fences, spying on the neighbours. The Dursleys had a small son called Dudley and in their opinion there was no finer boy anywhere.

The Dursleys had everything they wanted, but they also had a secret, and their greatest fear was that somebody would discover it. They didn't think they could bear it if anyone found out about the Potters. Mrs Potter was Mrs Dursley's

1

# A Hash Table based Dictionary

**Client Code:**

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

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

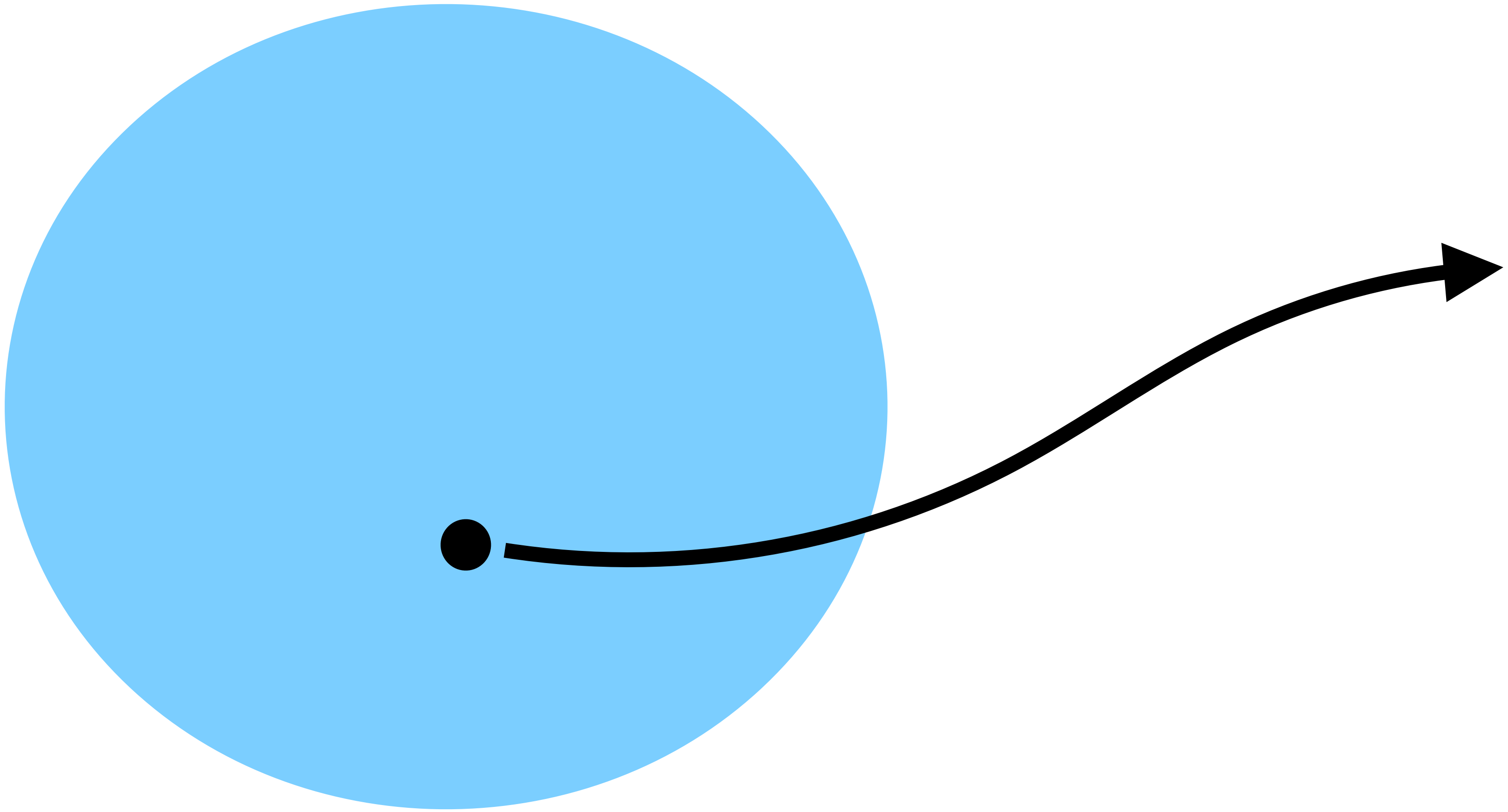
1.

2.

3.

# Hash Function

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



***m* elements**

| Key | Value |
| --- | --- |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

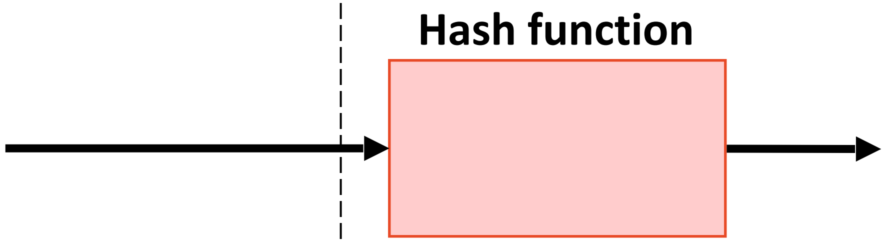
# Hash Function

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

- **Deterministic**:
- **Efficient**:
- **Defined for a certain size table:**

# 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)




| Key | Value |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

# 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**

(key[0] - ‘A’)

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

# General Hash Function

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

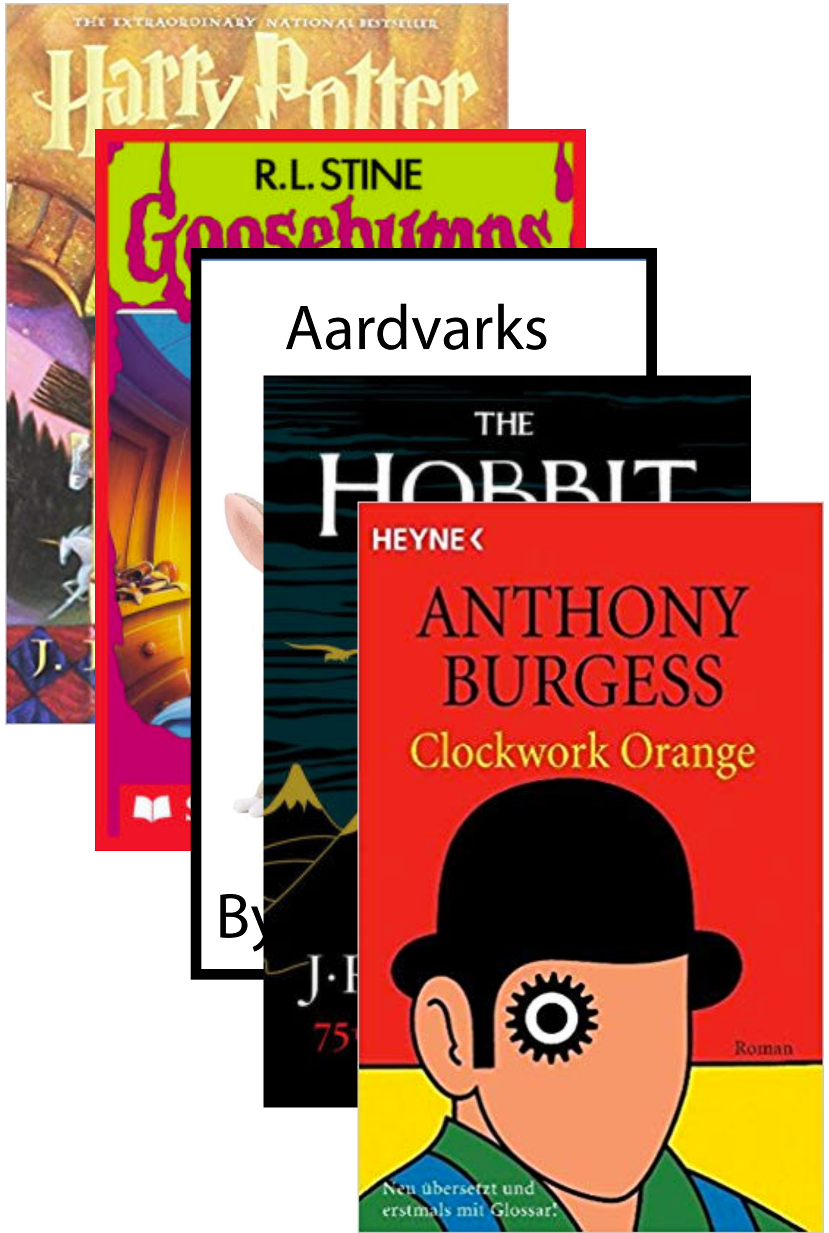
- A **hash**:

- A **compression**:

**Choosing a good hash function is tricky…**

- Don’t create your own (yet*)

# Hash Function

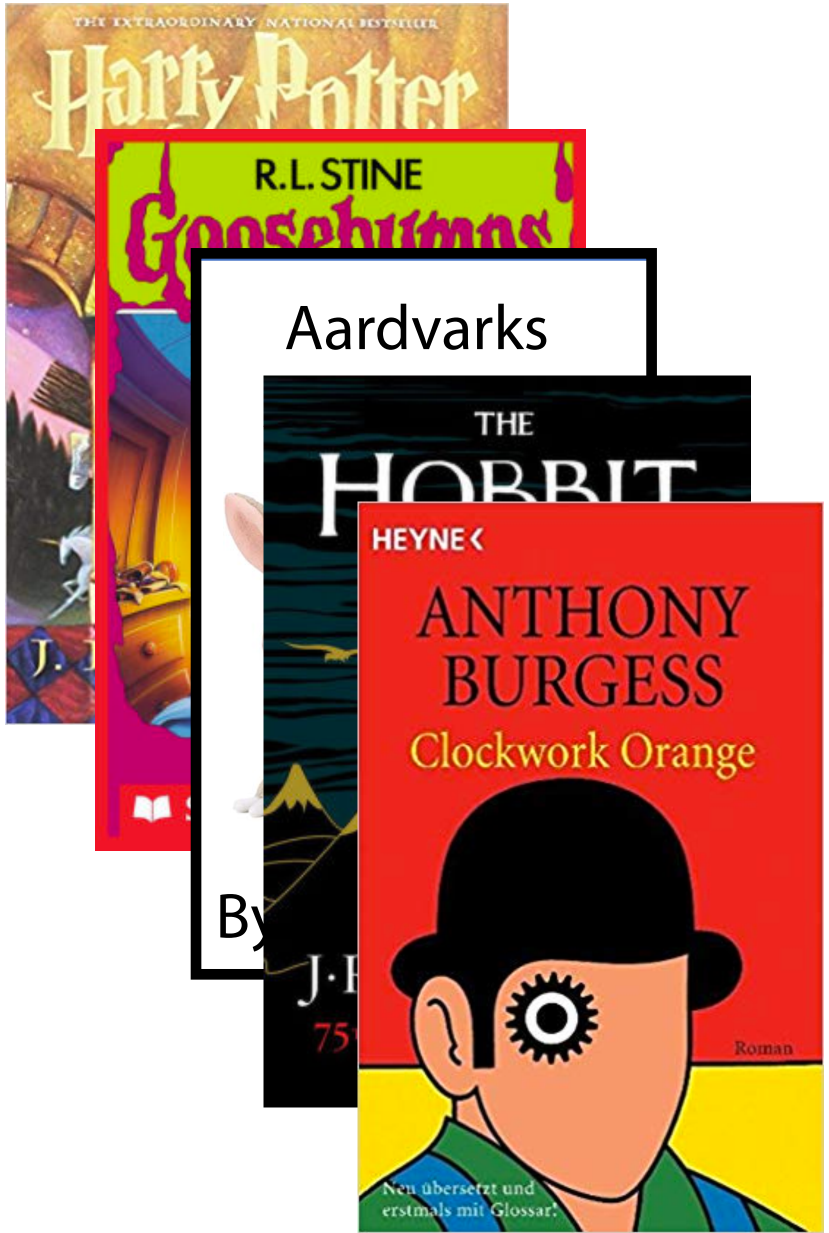



$$h(k) = \left(k.firstName[0] + k.lastName[0]\right) \% m$$

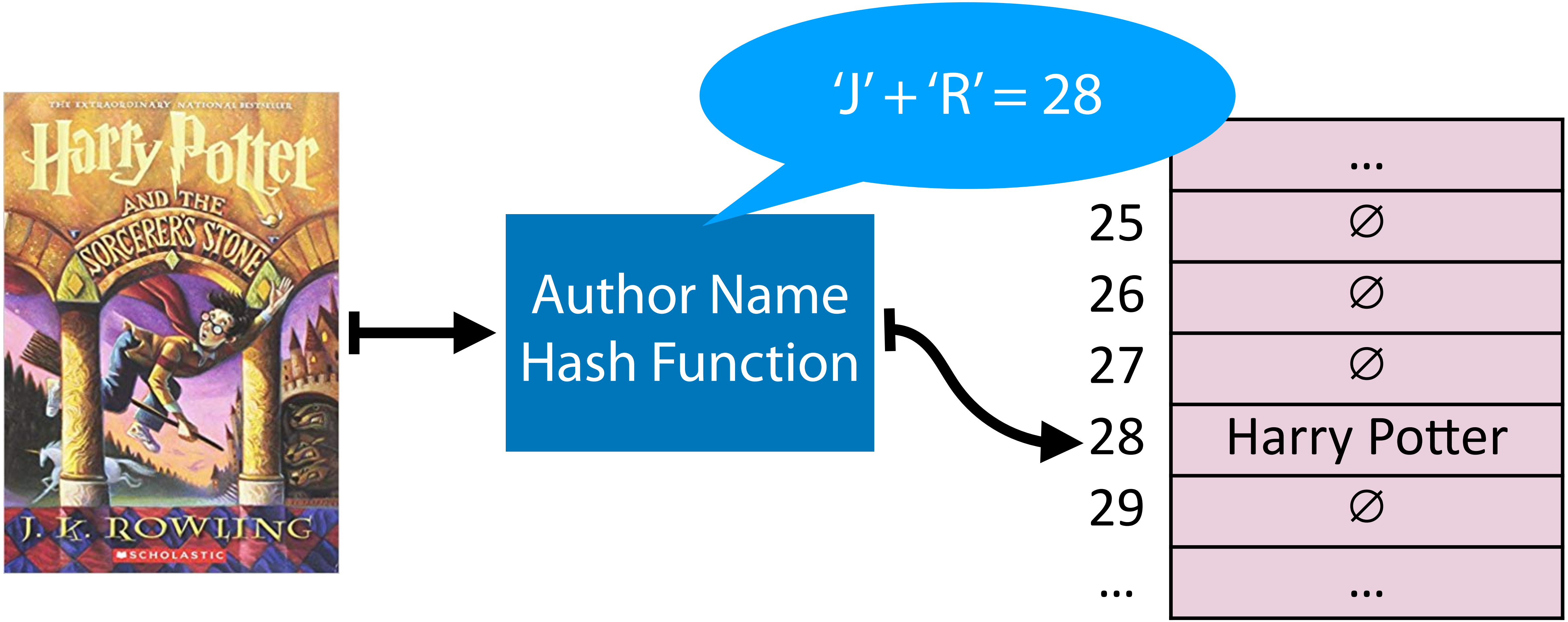
$$h(k) = \left(rand() * k.numPages\right) \% m$$

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

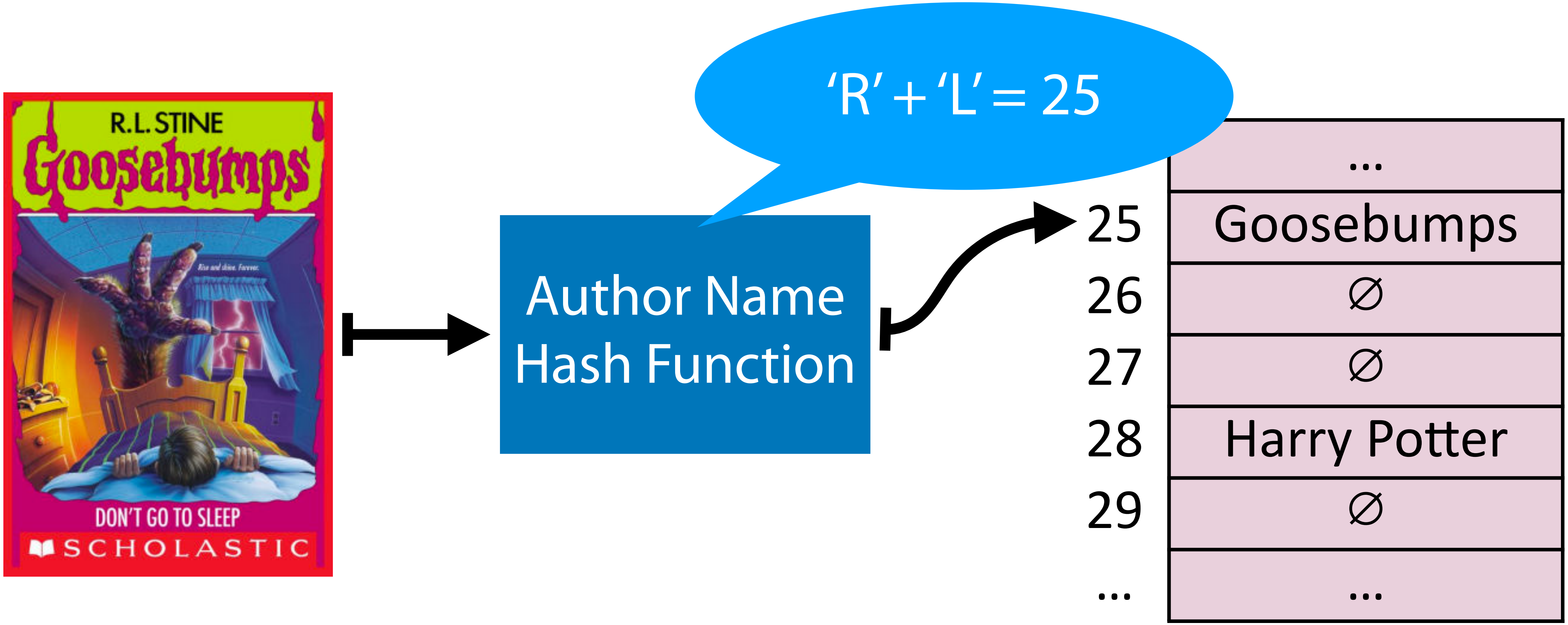
# Hash Function



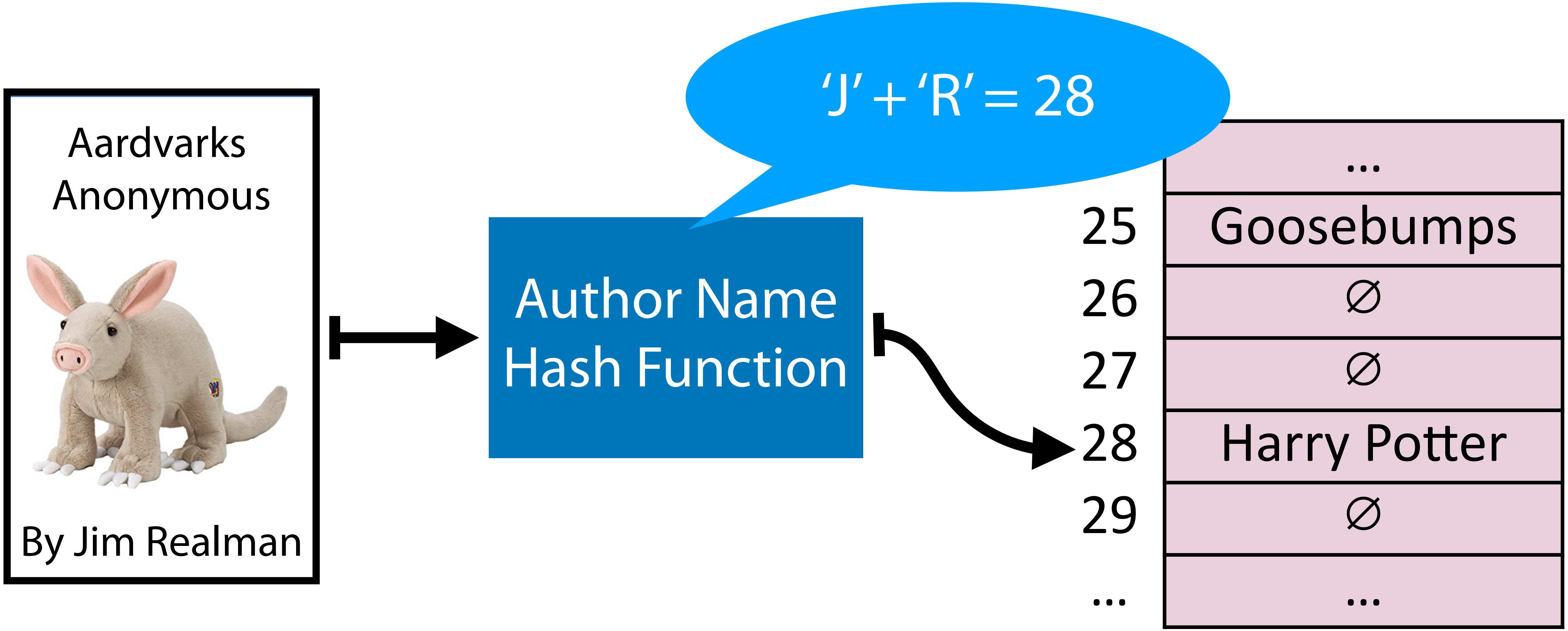
# Hash Function



# Hash Function



# Hash Function

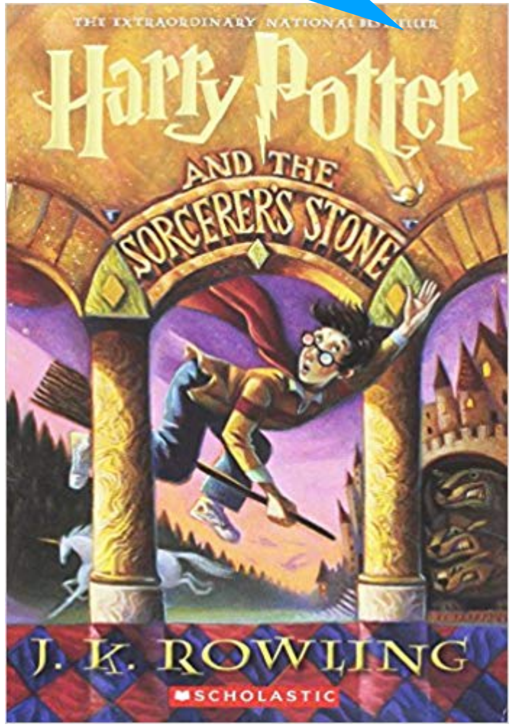


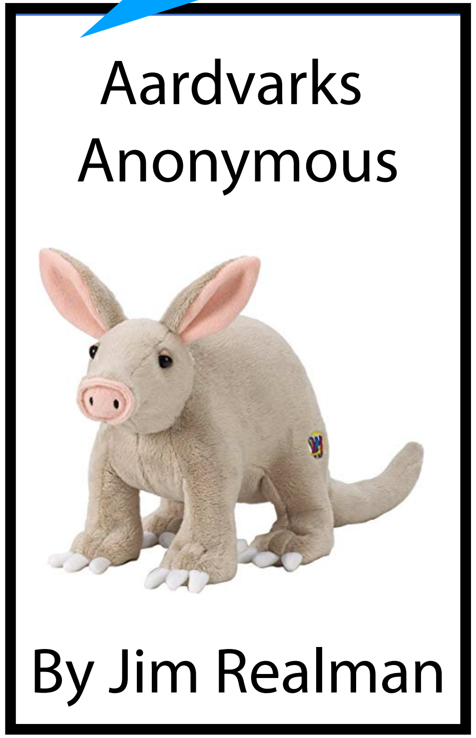
# Hash Collision

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

J.K Rowling = 28!

Jim Realman = 28!

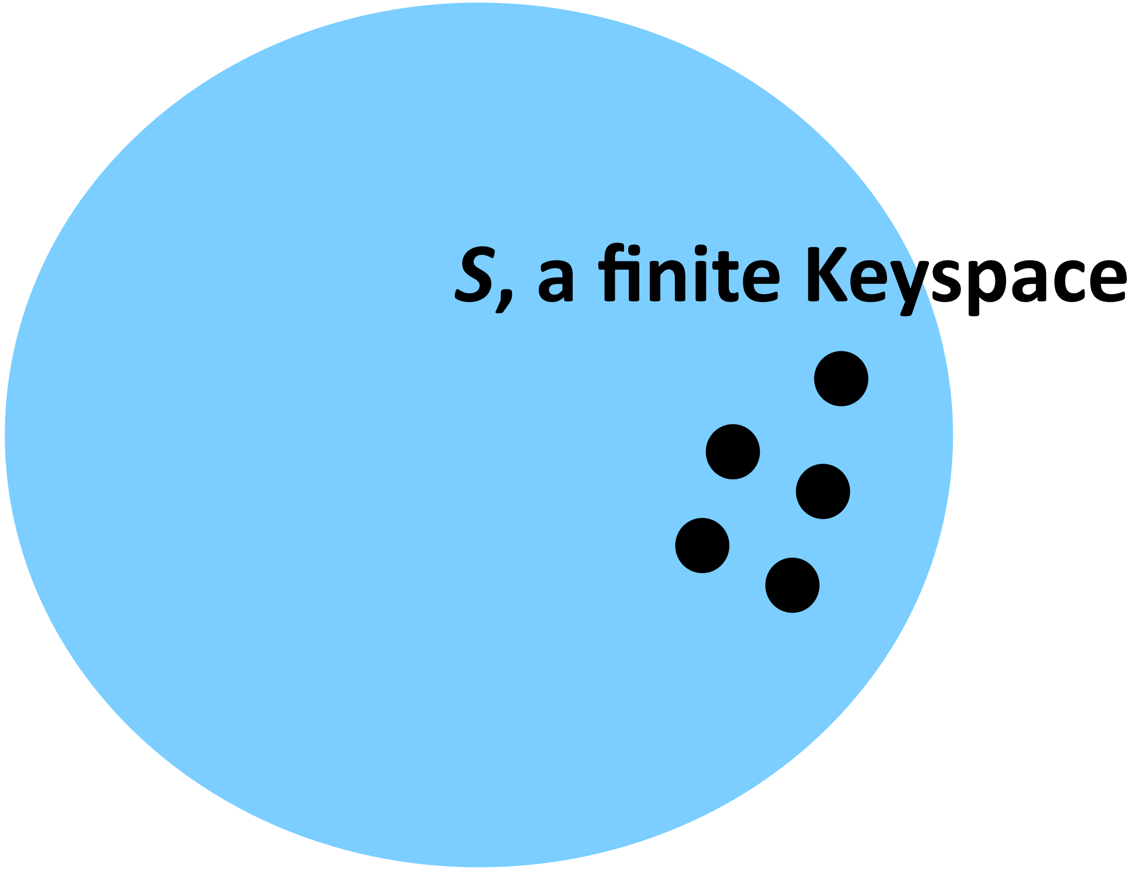



| | |
|---|---|
| ... | ... |
| 25 | Goosebumps |
| 26 | ∅ |
| 27 | ∅ |
| 28 | ??? |
| 29 | ∅ |
| ... | ... |

# Perfect Hashing

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

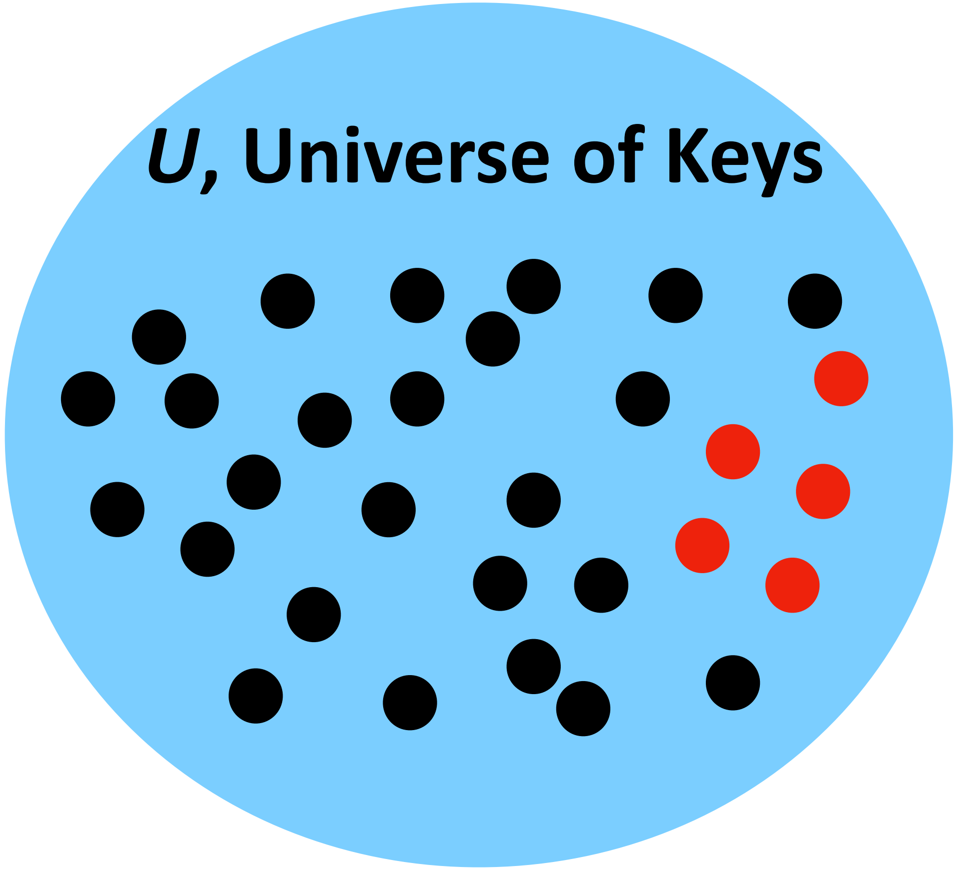



***m* elements**

| Key | Value |
|---|---|
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |

# General Purpose Hashing

In CS 225, we want our hash functions to work *in general*.

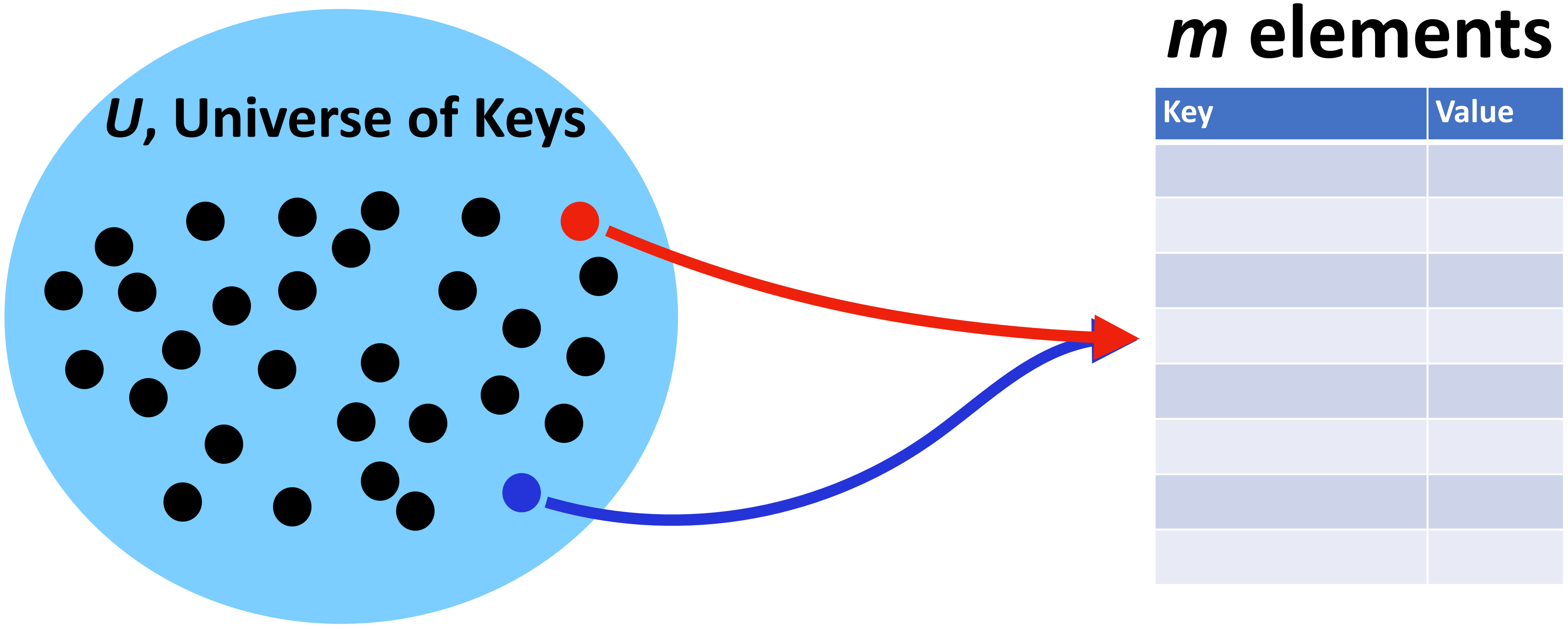



***m* elements**

| 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.

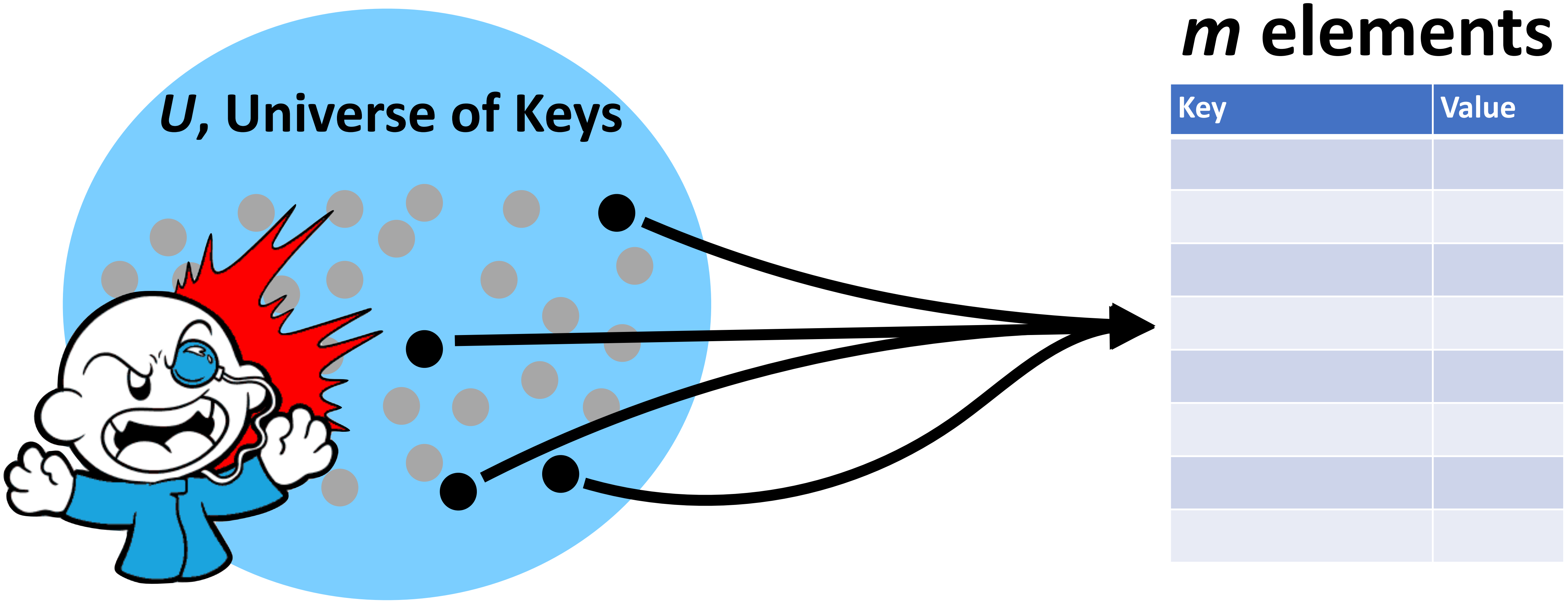



Image by Matthew Loffhagen

# A Hash Table based Dictionary

**Client Code:**

```
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

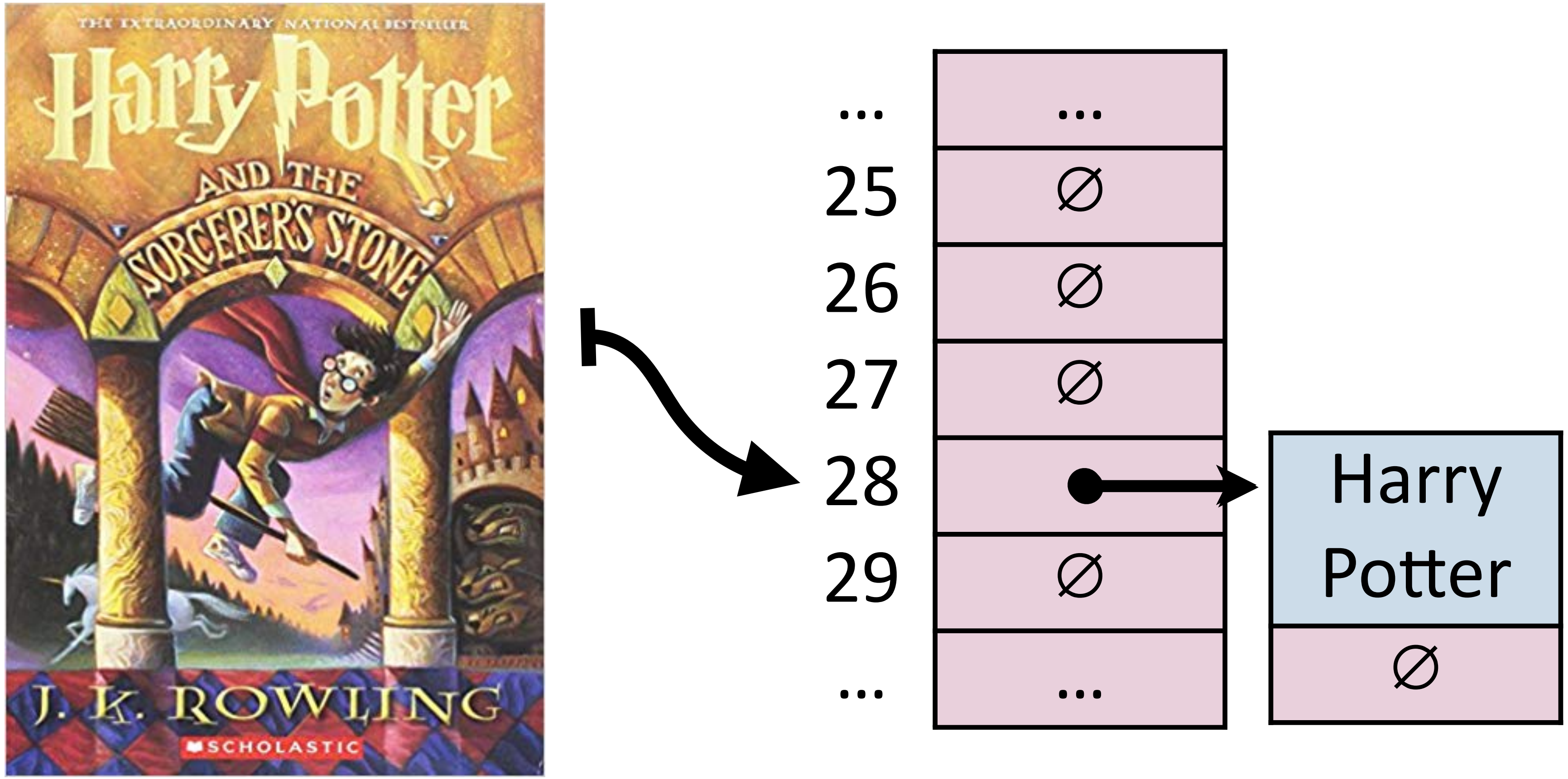
Addressing hash collisions depends on your storage structure.

- **Open Hashing:**

- **Closed Hashing:**

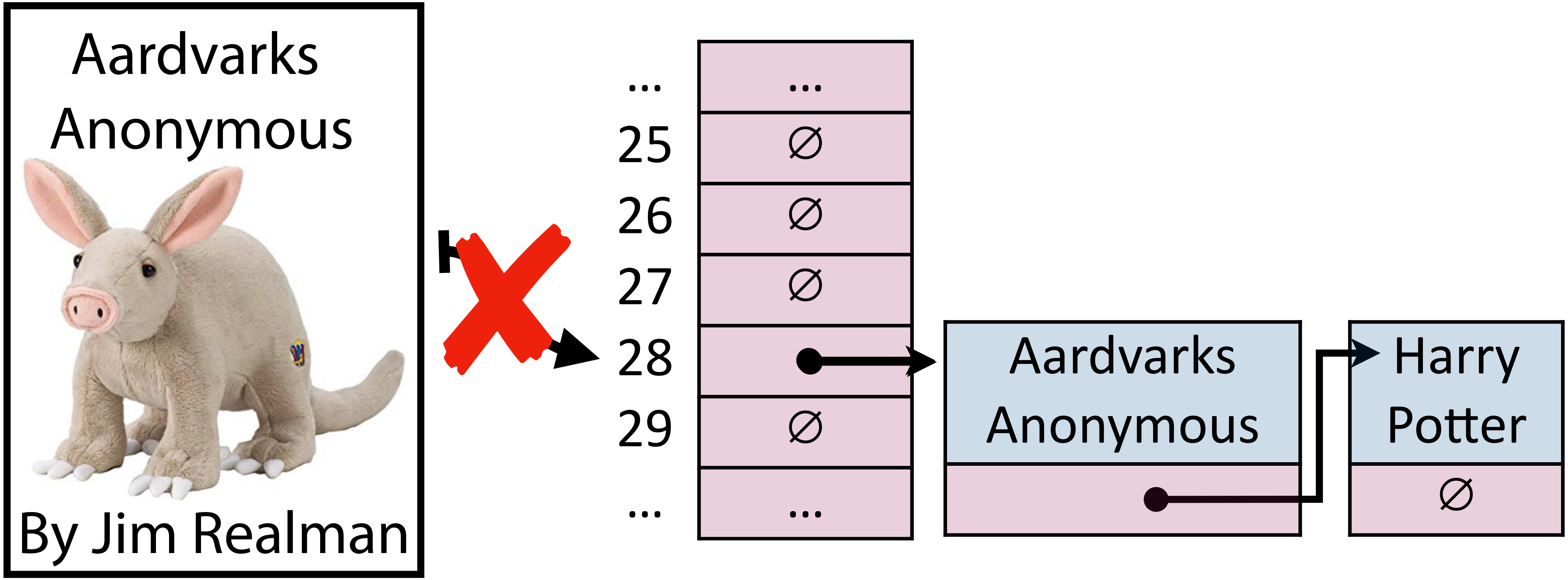
# Open Hashing

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 ______________________________. This is called ***separate chaining***.



# Insertion (Separate Chaining)

`_insert("Bob")`

`_insert("Anna")`

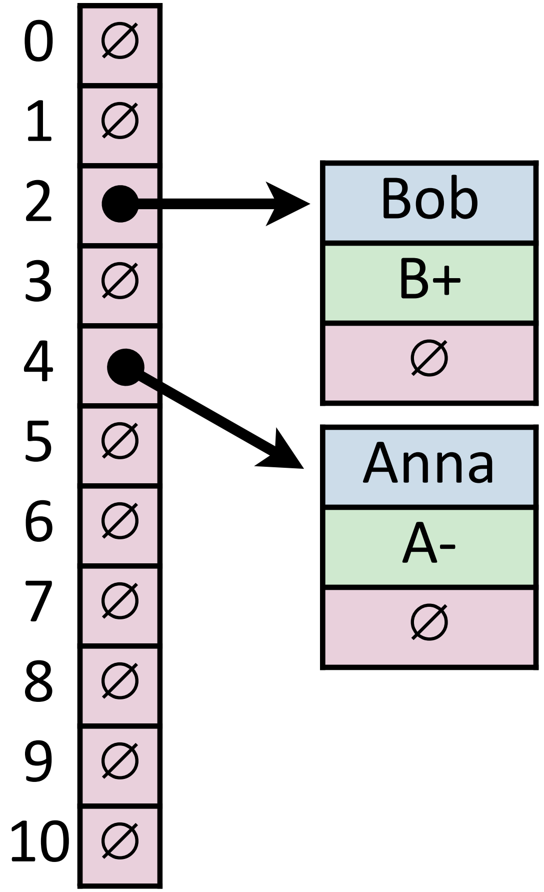
| Key | Value | Hash |
|---|---|---|
| **Bob** | **B+** | **2** |
| **Anna** | **A-** | **4** |
| Alice | A+ | 4 |
| Betty | B | 2 |
| Brett | A- | 2 |
| Greg | A | 0 |
| Sue | B | 7 |
| Ali | B+ | 4 |
| Laura | A | 7 |
| Lily | B+ | 7 |

| | |
|---|---|
| 0 | ∅ |
| 1 | ∅ |
| 2 | ∅ |
| 3 | ∅ |
| 4 | ∅ |
| 5 | ∅ |
| 6 | ∅ |
| 7 | ∅ |
| 8 | ∅ |
| 9 | ∅ |
| 10 | ∅ |

# Insertion (Separate Chaining)

`_insert("Alice")`

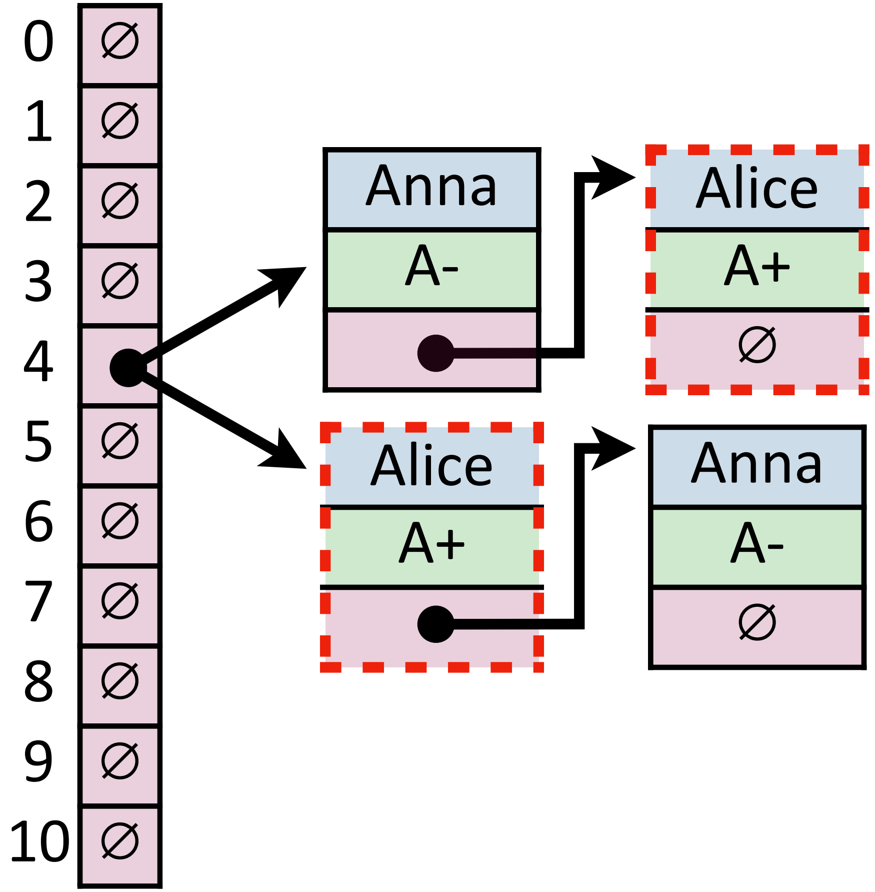
| Key | Value | Hash |
| --- | --- | --- |
| Bob | B+ | 2 |
| Anna | A- | 4 |
| **Alice** | **A+** | **4** |
| Betty | B | 2 |
| Brett | A- | 2 |
| Greg | A | 0 |
| Sue | B | 7 |
| Ali | B+ | 4 |
| Laura | A | 7 |
| Lily | B+ | 7 |



# Insertion (Separate Chaining)

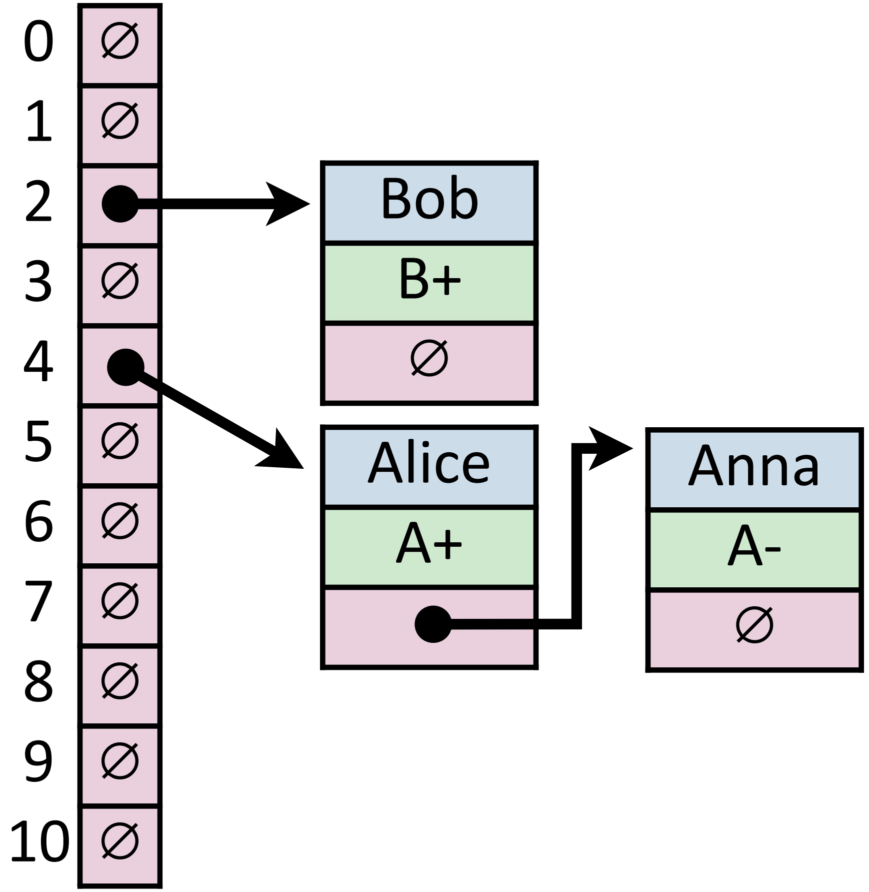
Where does Alice end up relative to Anna in the chain?

| 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 |



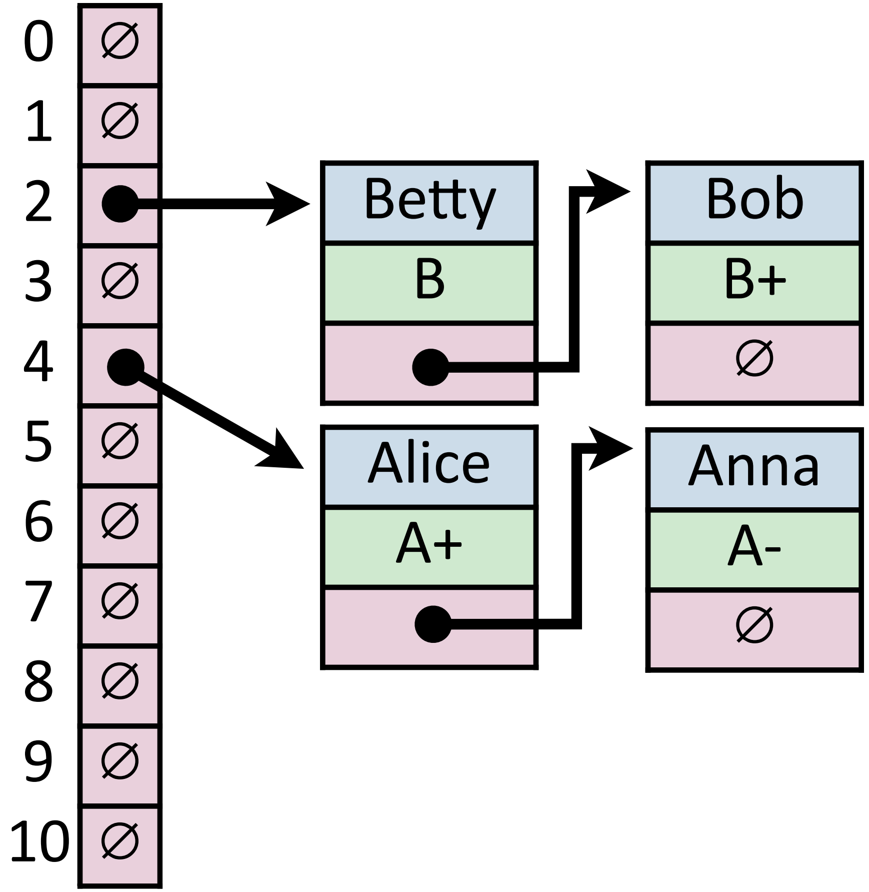
# Insertion (Separate Chaining)

| 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 |



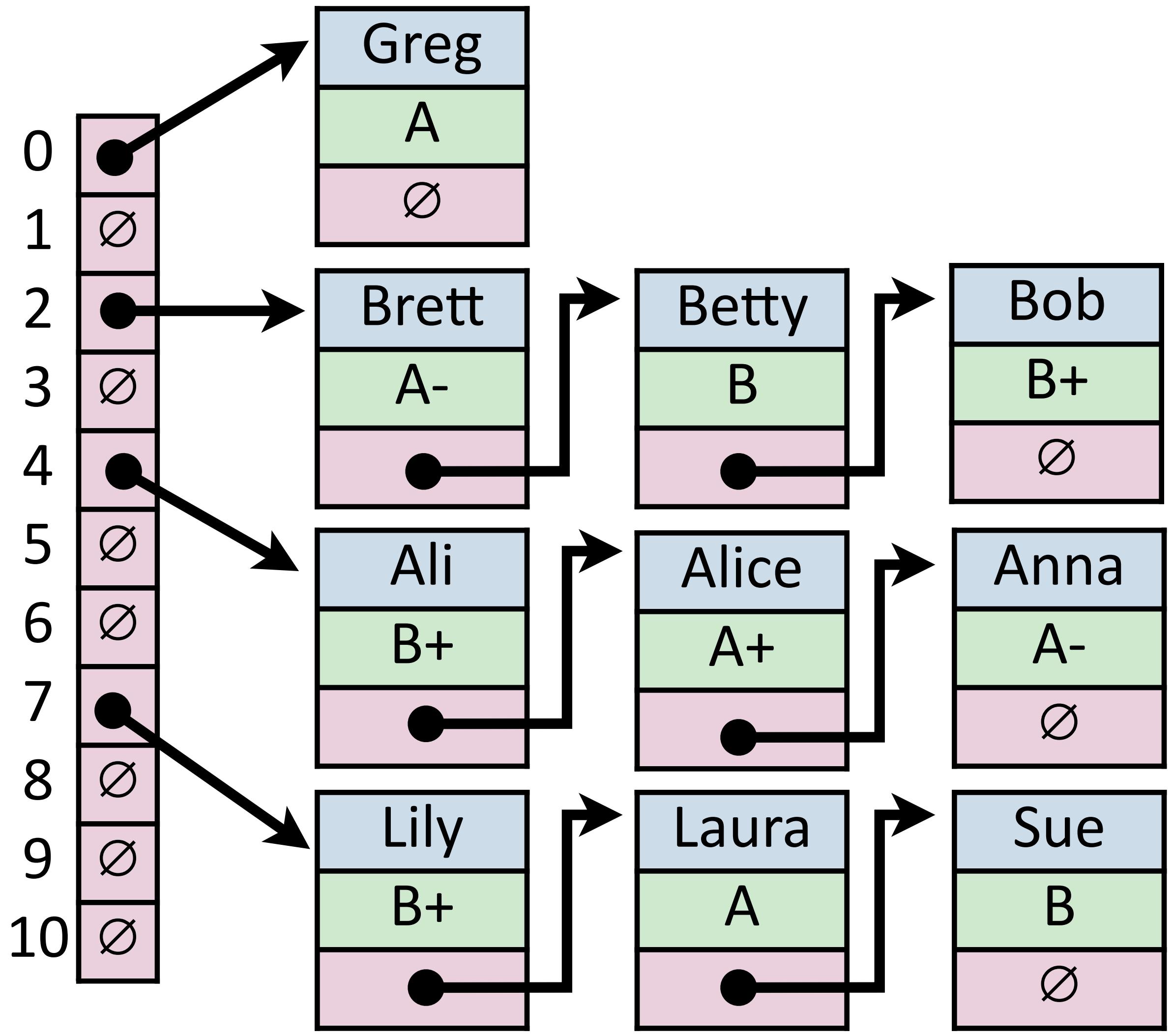
# Insertion (Separate Chaining)

| 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 |



# Insertion (Separate Chaining)

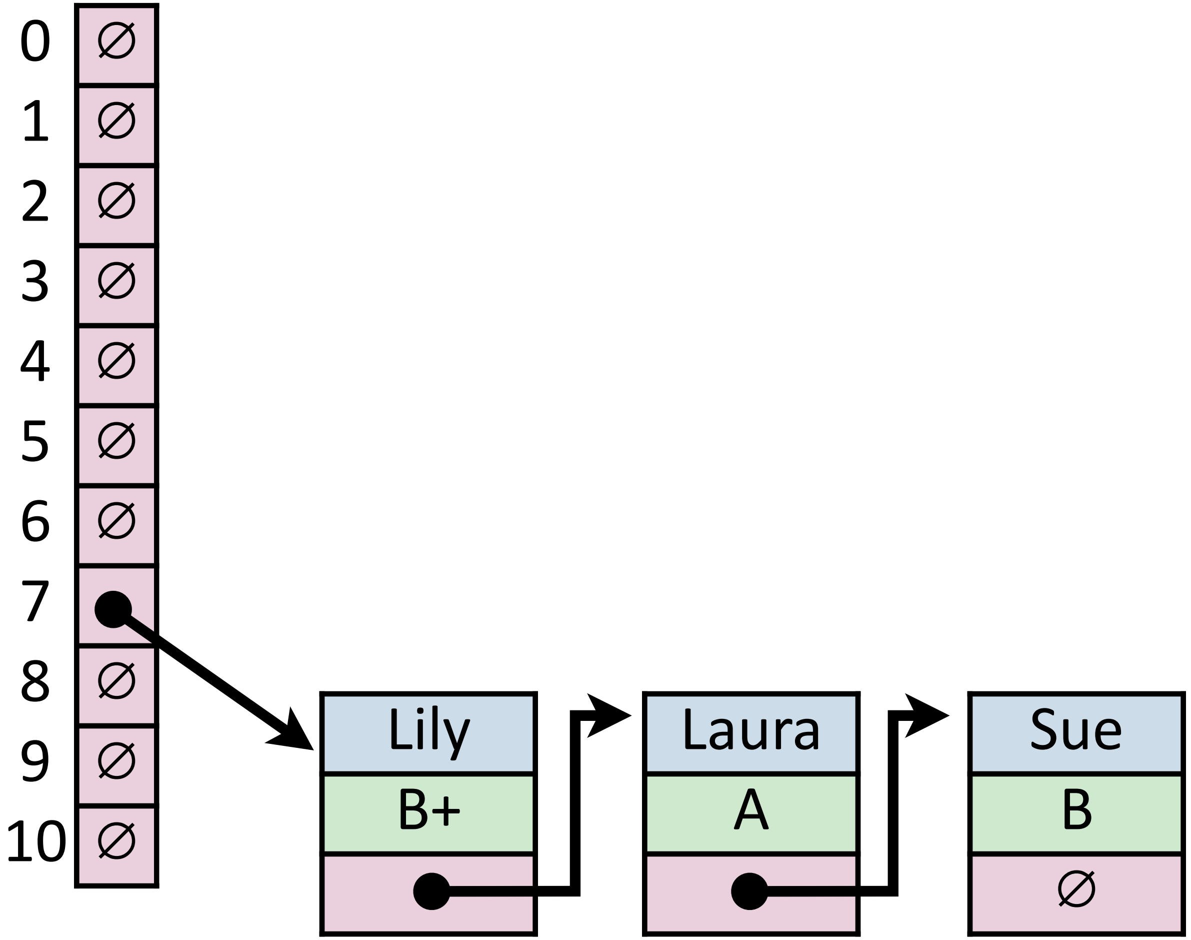
| Key | Value | Hash |
|---|---|---|
| Bob | B+ | 2 |
| Anna | A- | 4 |
| Alice | A+ | 4 |
| Betty | B | 2 |
| Brett | A- | 2 |
| Greg | A | 0 |
| Sue | B | 7 |
| Ali | B+ | 4 |
| Laura | A | 7 |
| Lily | B+ | 7 |



# Find (Separate Chaining)

`_find("Sue")`

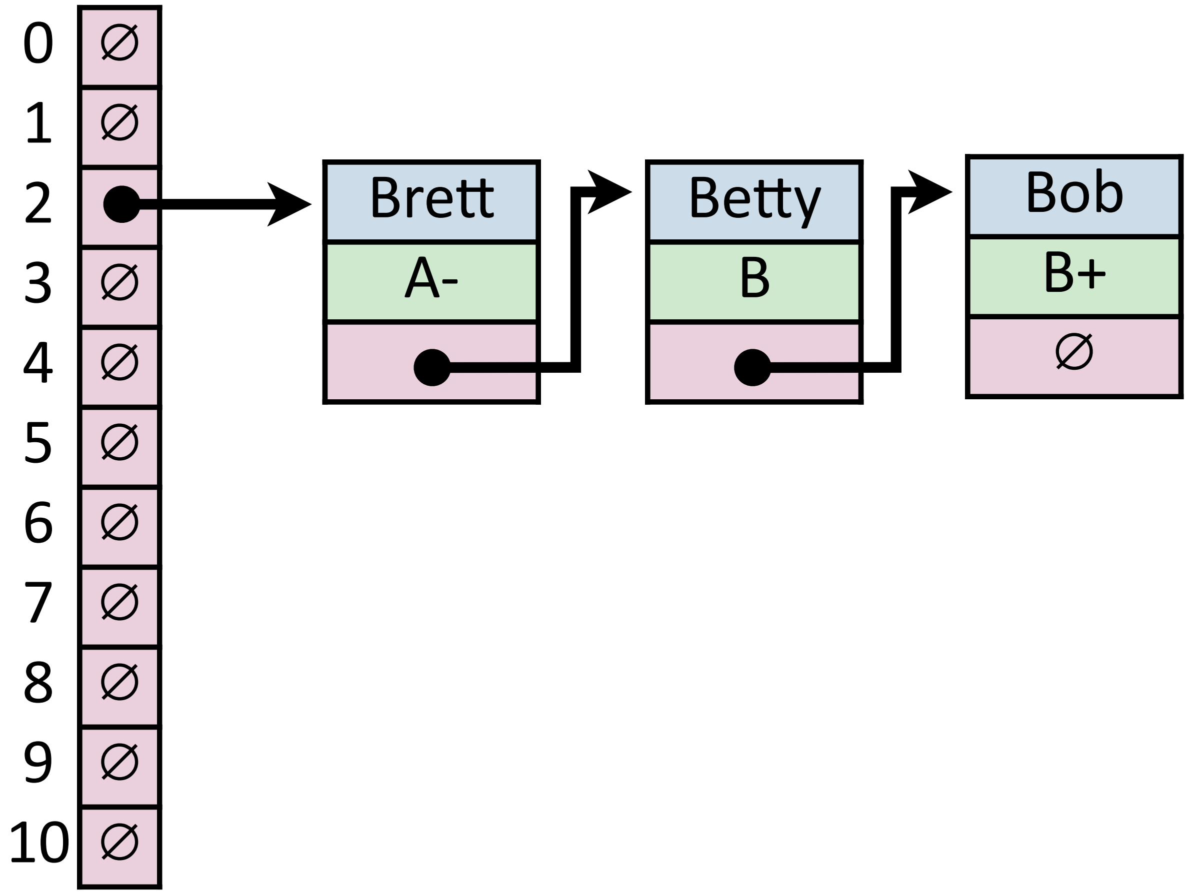
| Key | Hash |
|---|---|
| Sue | 7 |



# Remove (Separate Chaining)

`_remove("Betty")`

| Key | Hash |
| --- | --- |
| Betty | 2 |



# Hash Table (Separate Chaining)

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

find runs in: ____________.

insert runs in: ____________.

remove runs in: ____________.

# Hash Table

Two ways forward:

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

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

# Simple Uniform Hashing Assumption

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

$$\forall k_1, k_2 \in U \text{ where } k_1 \neq k_2 \text{ , } Pr(h[k_1] = h[k_2]) = \frac{1}{m}$$

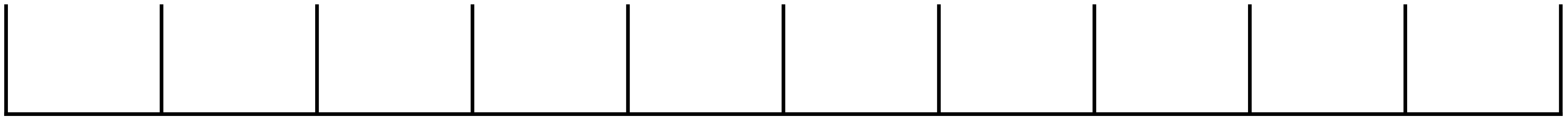
**Uniform:**

**Independent:**

# Separate Chaining Under SUHA

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

Expected length of chain is _______________.



# Separate Chaining Under SUHA

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

find runs in: ___________.

insert runs in: ___________.

remove runs in: ___________.

# 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 | |
| 2 | |
| 3 | |
| 4 | |
| 5 | |
| 6 | |