Algorithms and Data Structures for Data Science
Hashing

CS 277
Brad Solomon

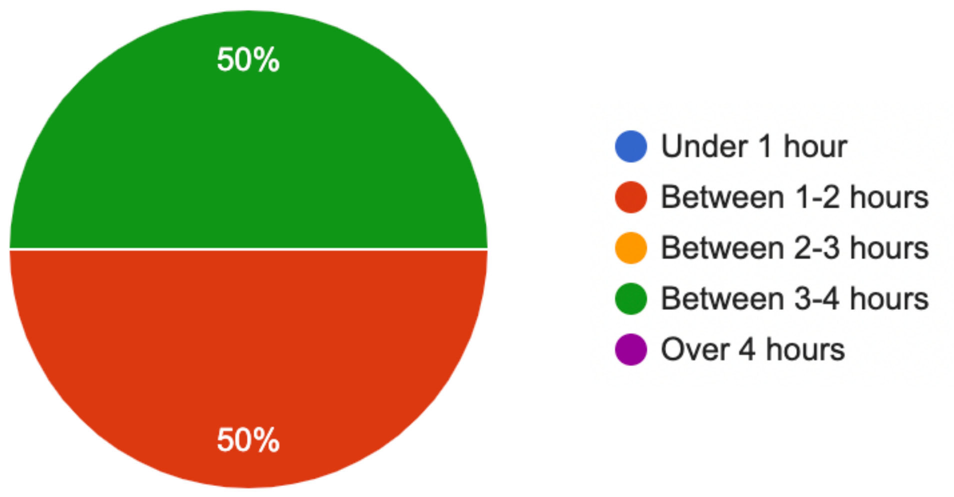
February 15, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Lab_cipher Feedback

Average score: 104%

PL average time: 73 minutes



Class helpful to all students (who filled out survey)

Lab taught learning objectives (but did not improve coding confidence)

People liked the lab as it required logic to solve

Learning Objectives

Motivate and define a hash table

Discuss what a ‘good’ hash function looks like

Identify a key weakness of the hash table

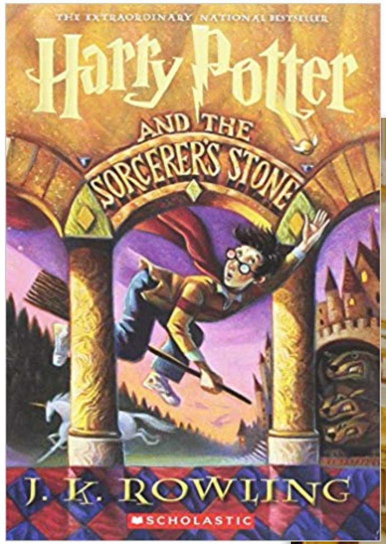
Introduce strategies to ‘correct’ this weakness

Optimal Find

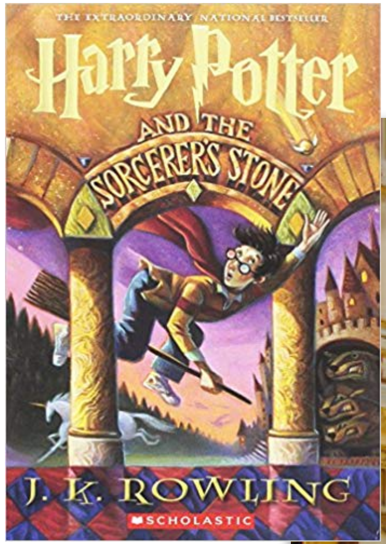
Imagine you have an arbitrary collection of numbers and want to store them in an efficient data structure designed for **find**.

What would you do?

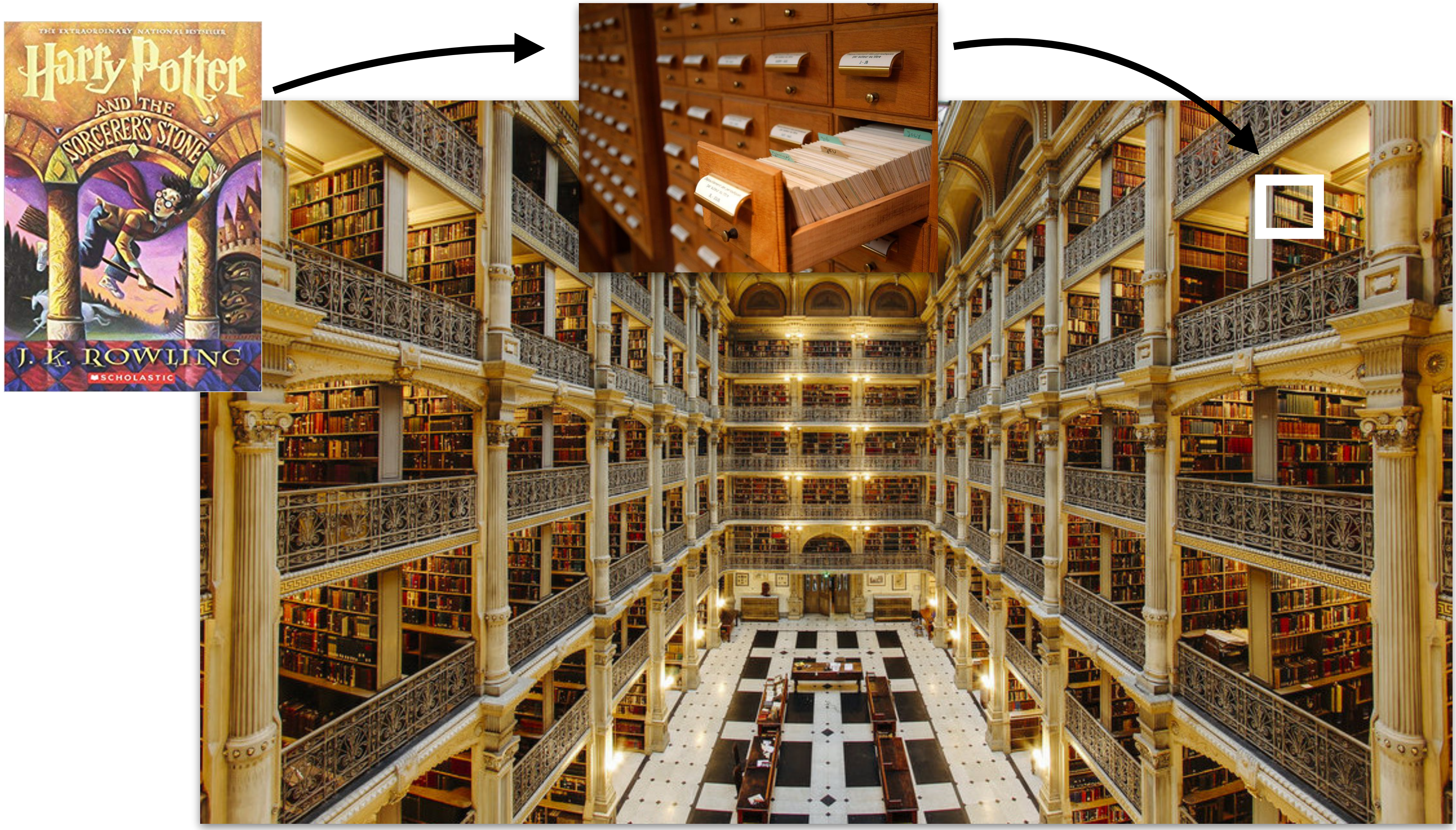
Optimal currently is $O(n)$



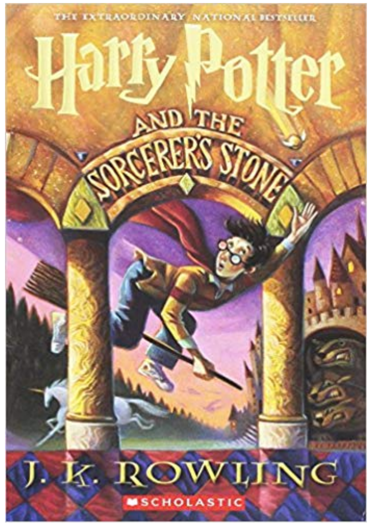
If we recognize that libraries are ordered: $O(log\ n)$



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

A Hash Table based Dictionary

```
d = {}
d[k] = v
```

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

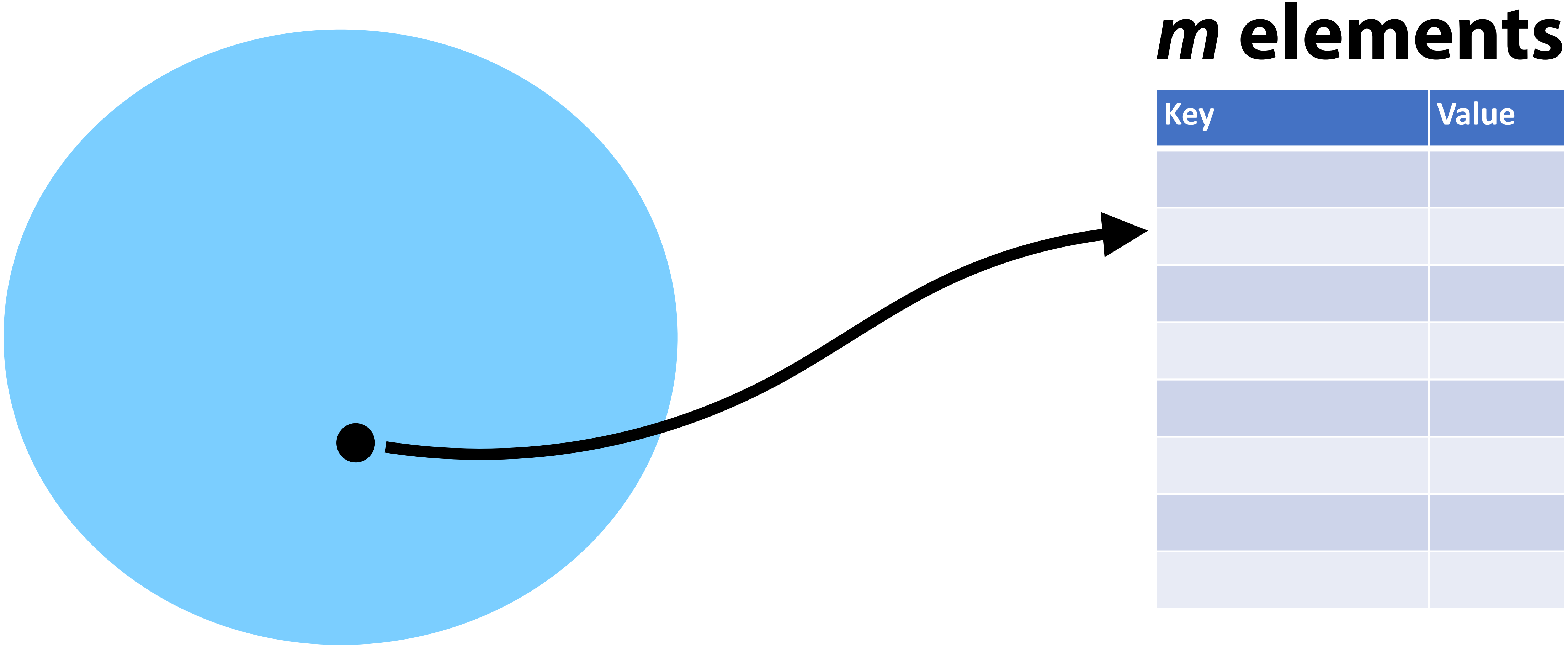
1.

2.

3.

Hash Function

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



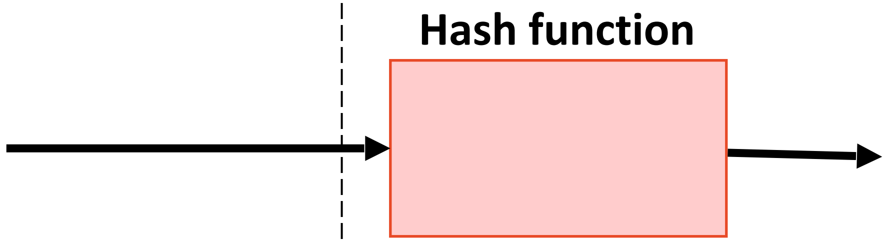
Hash Function

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 $[0, \ldots, m-1]$

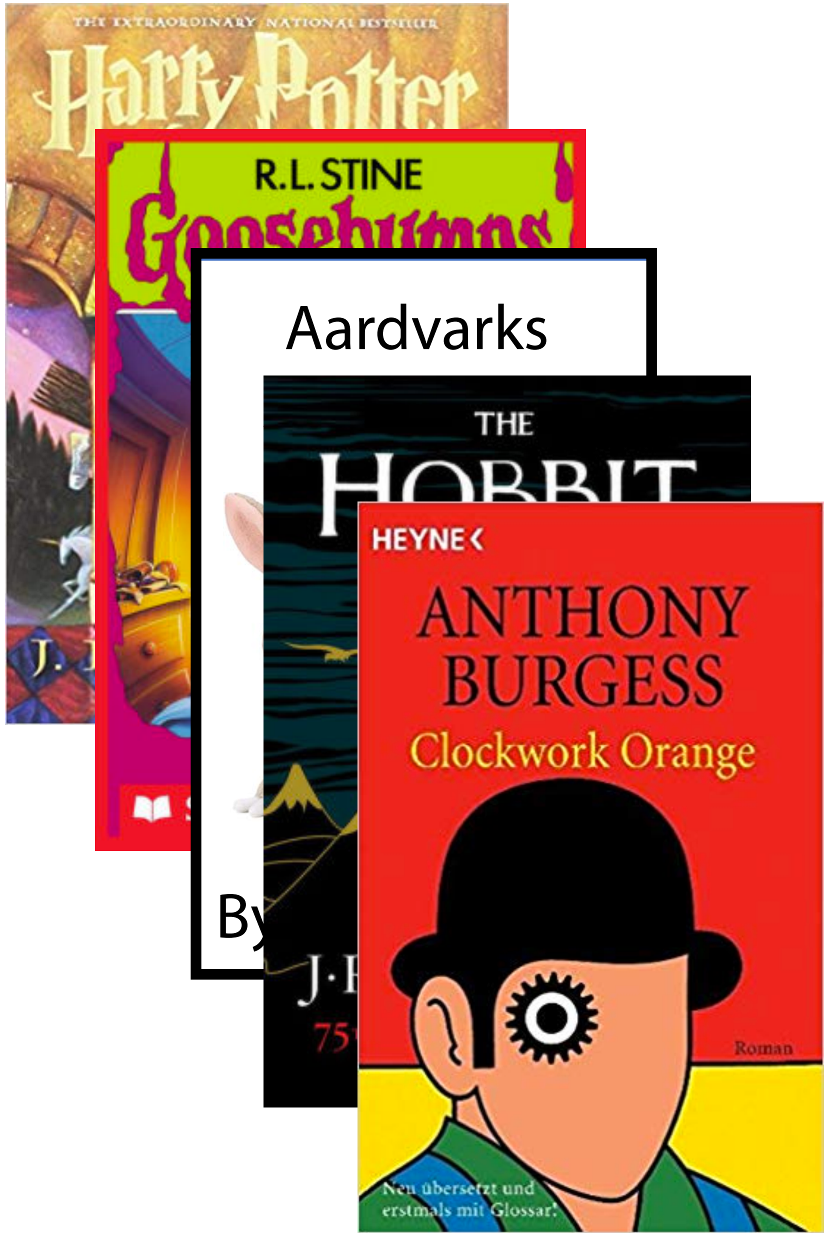
- A **hash**:

- A **compression**:

Choosing a good hash function is tricky…

- Don't create your own!

Hash Function

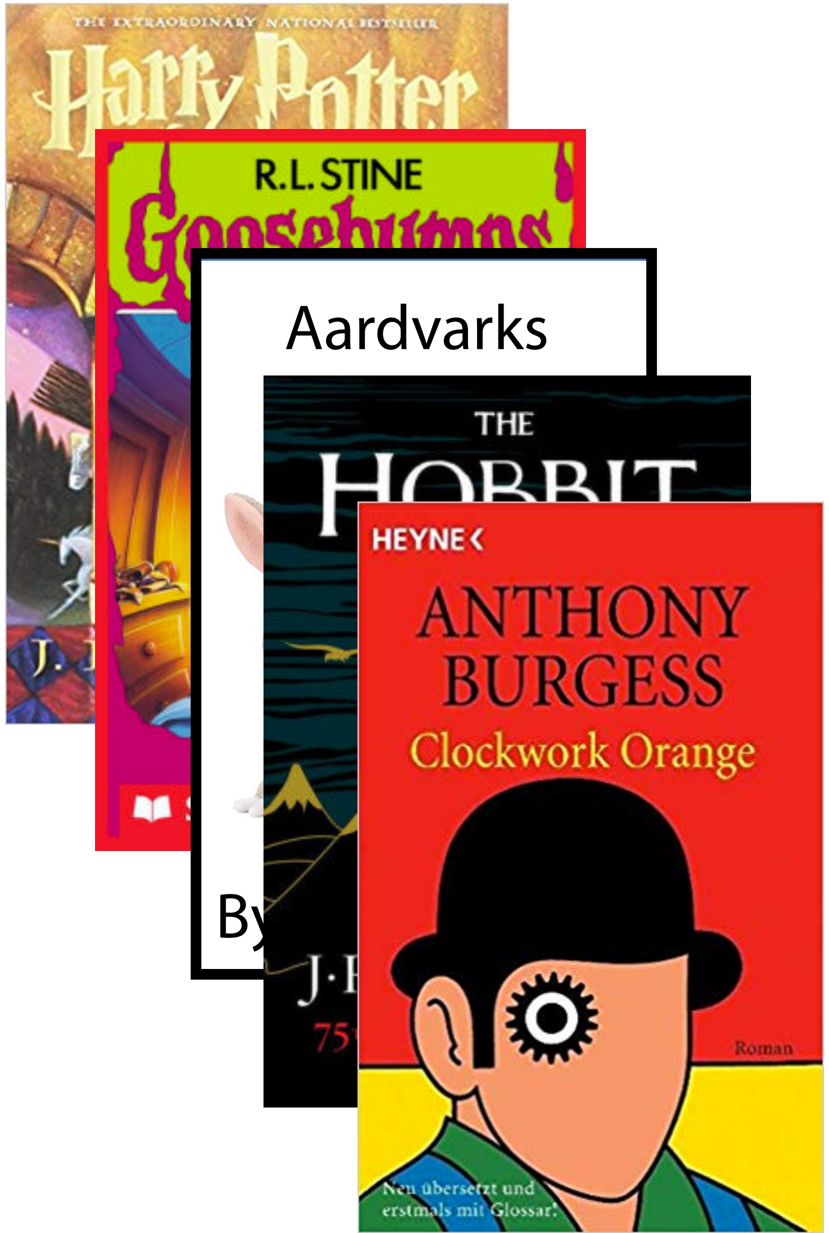


$$h(k) = \left(k\,.\,firstName[0] + k\,.\,lastName[0]\right)\,\%\,m$$

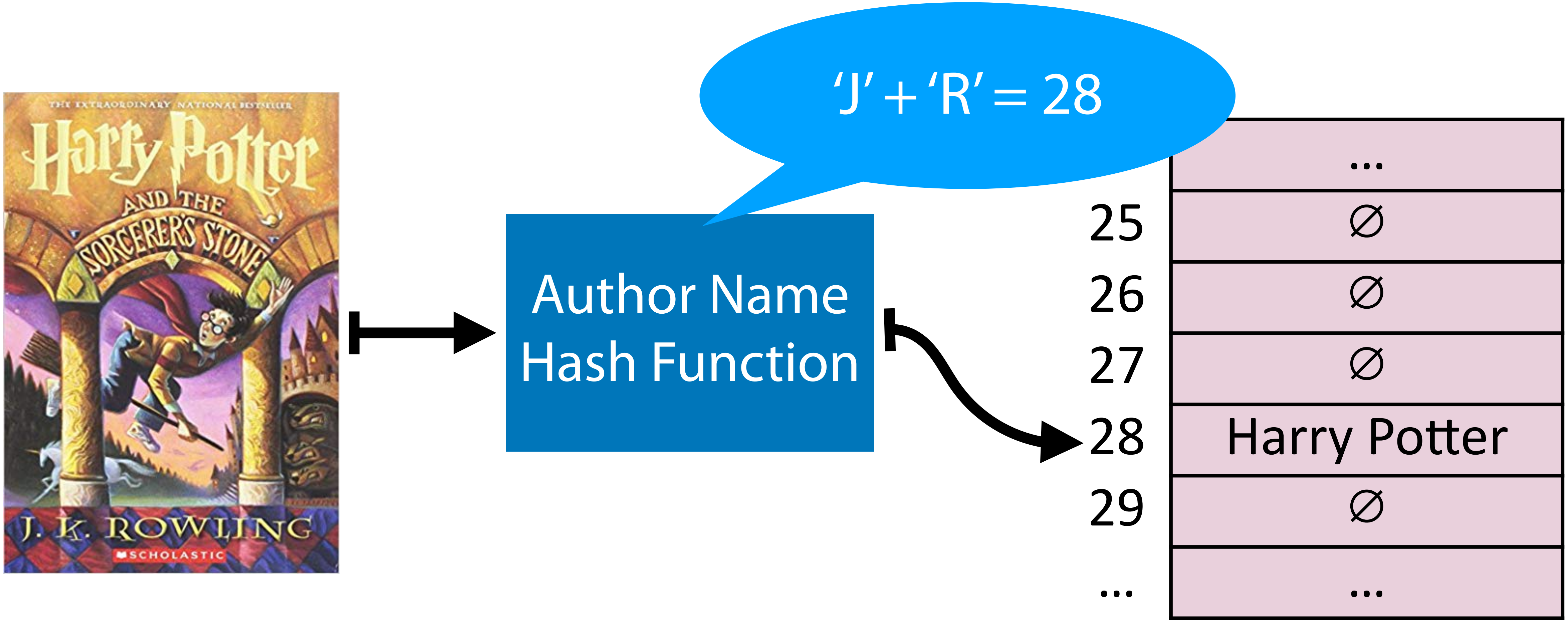
$$h(k) = \left(rand() * k\,.\,numPages\right)\,\%\,m$$

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

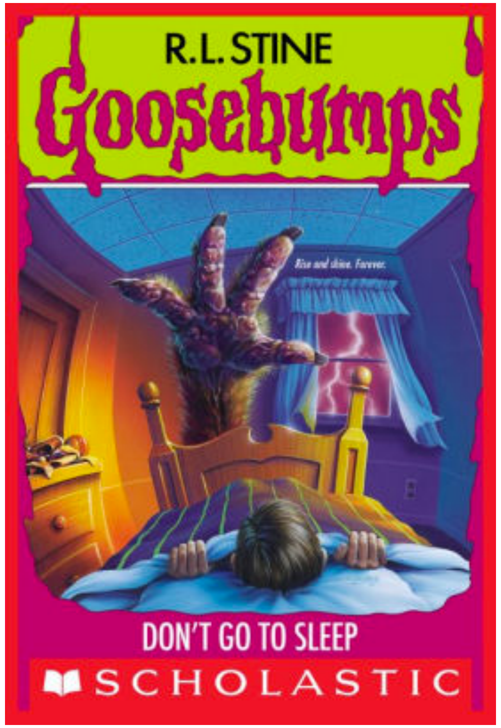
Hash Function



Hash Function



Hash Function

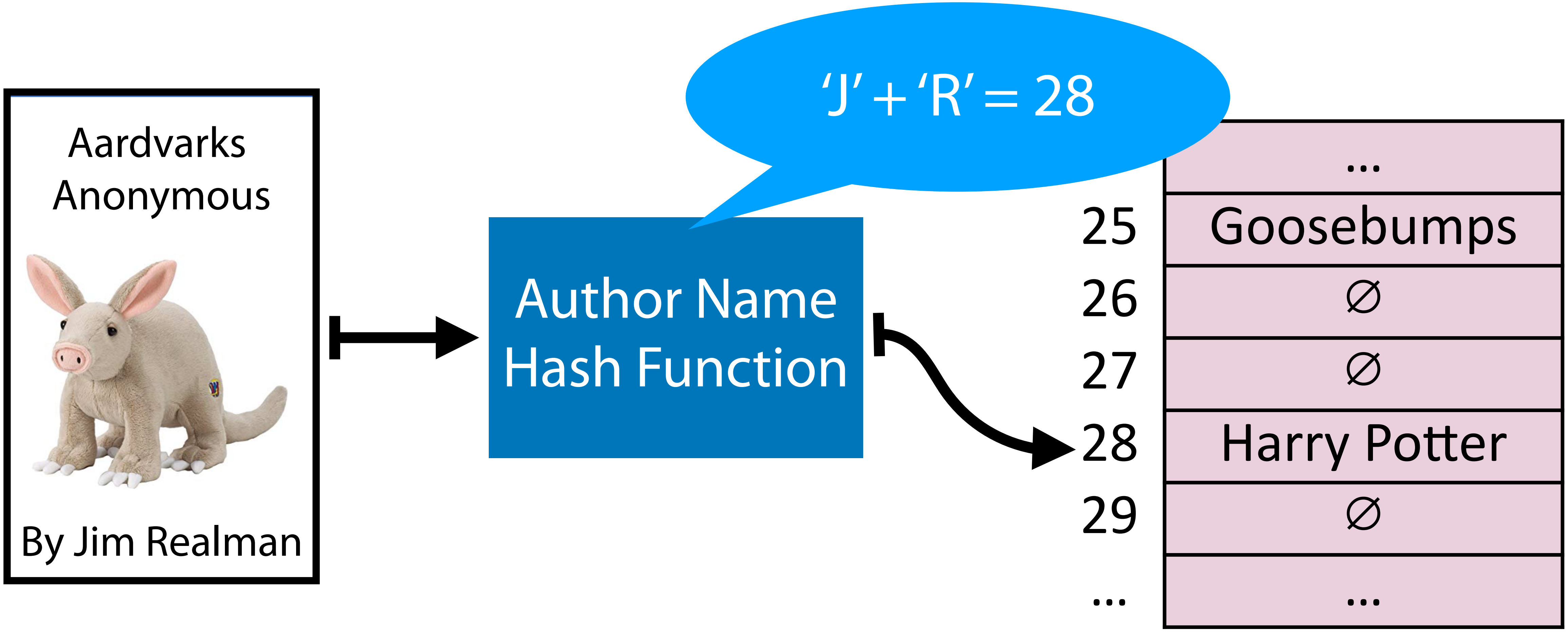


Author Name Hash Function

'R' + 'L' = 25

	…
25	Goosebumps
26	∅
27	∅
28	Harry Potter
29	∅
…	…

Hash Function

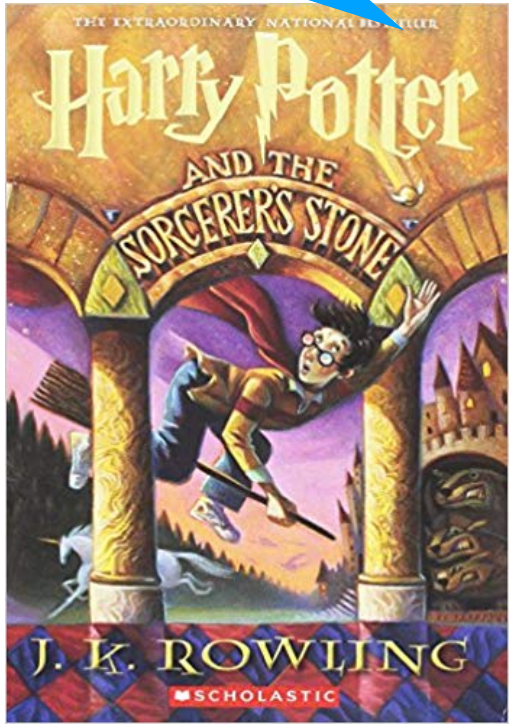


Hash Collision

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

J.K Rowling = 28!

Jim Realman = 28!



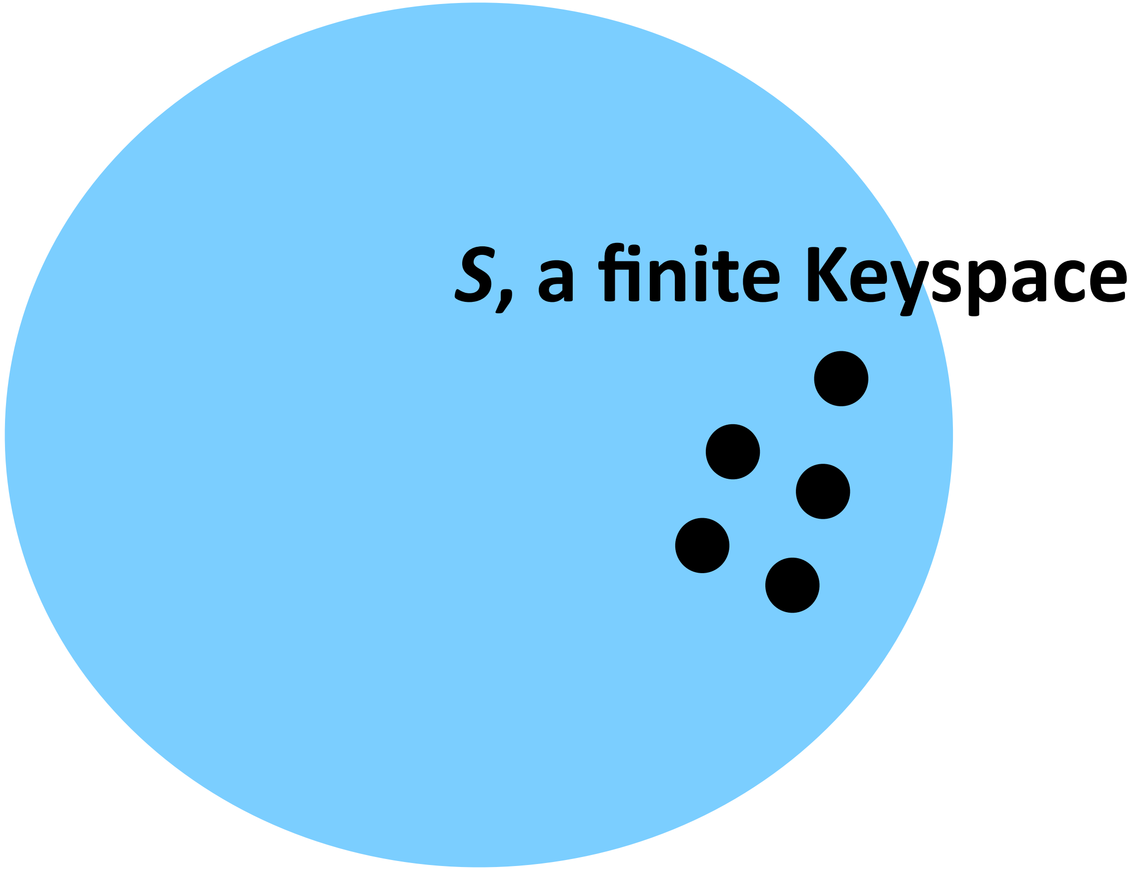
Aardvarks Anonymous

By Jim Realman

…	…
25	Goosebumps
26	∅
27	∅
28	???
29	∅
…	…

Perfect Hashing

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

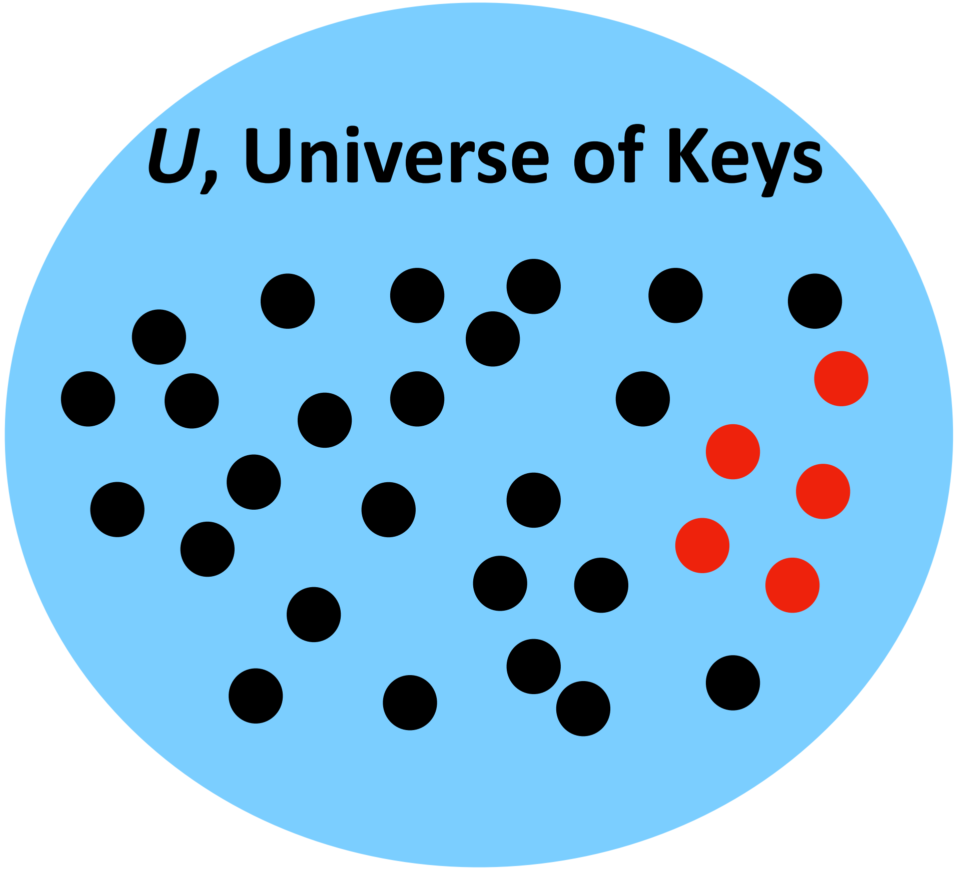


m elements

Key	Value

General Purpose Hashing

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

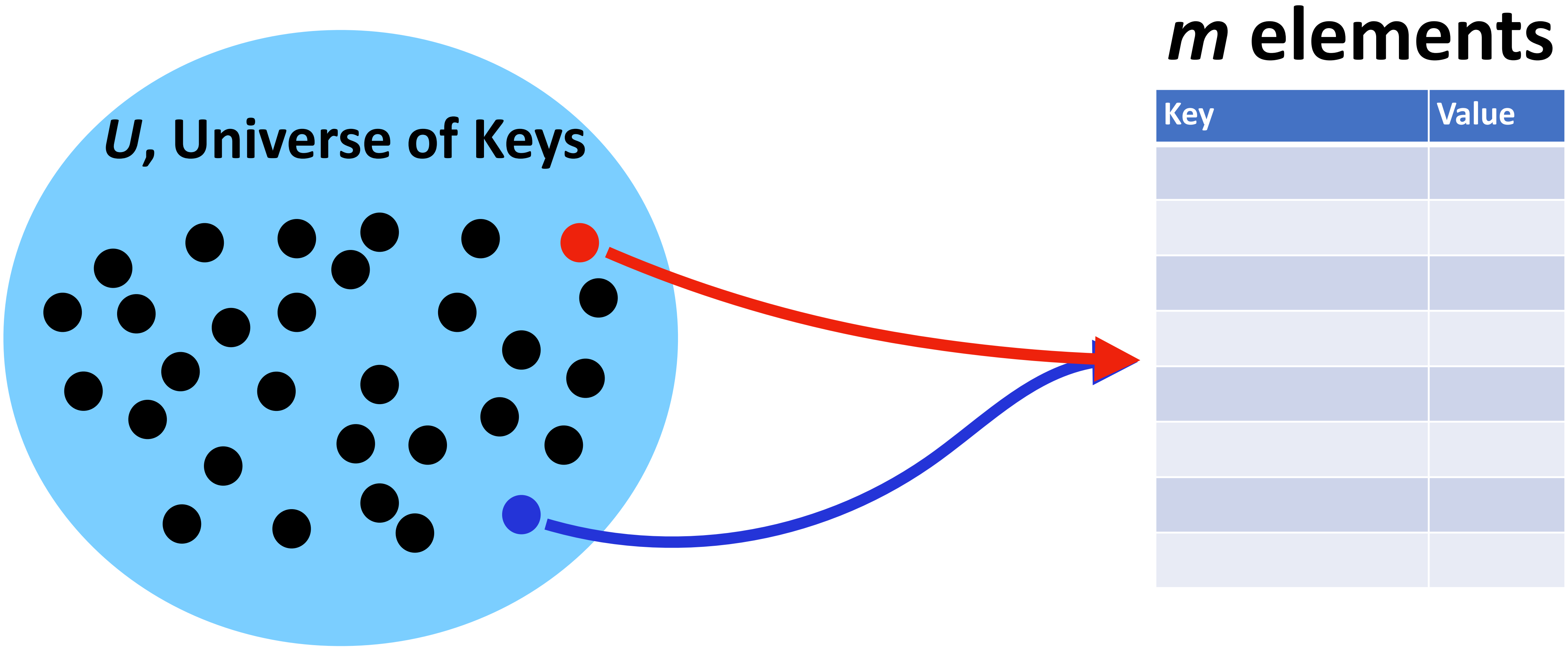


***m* elements**

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.

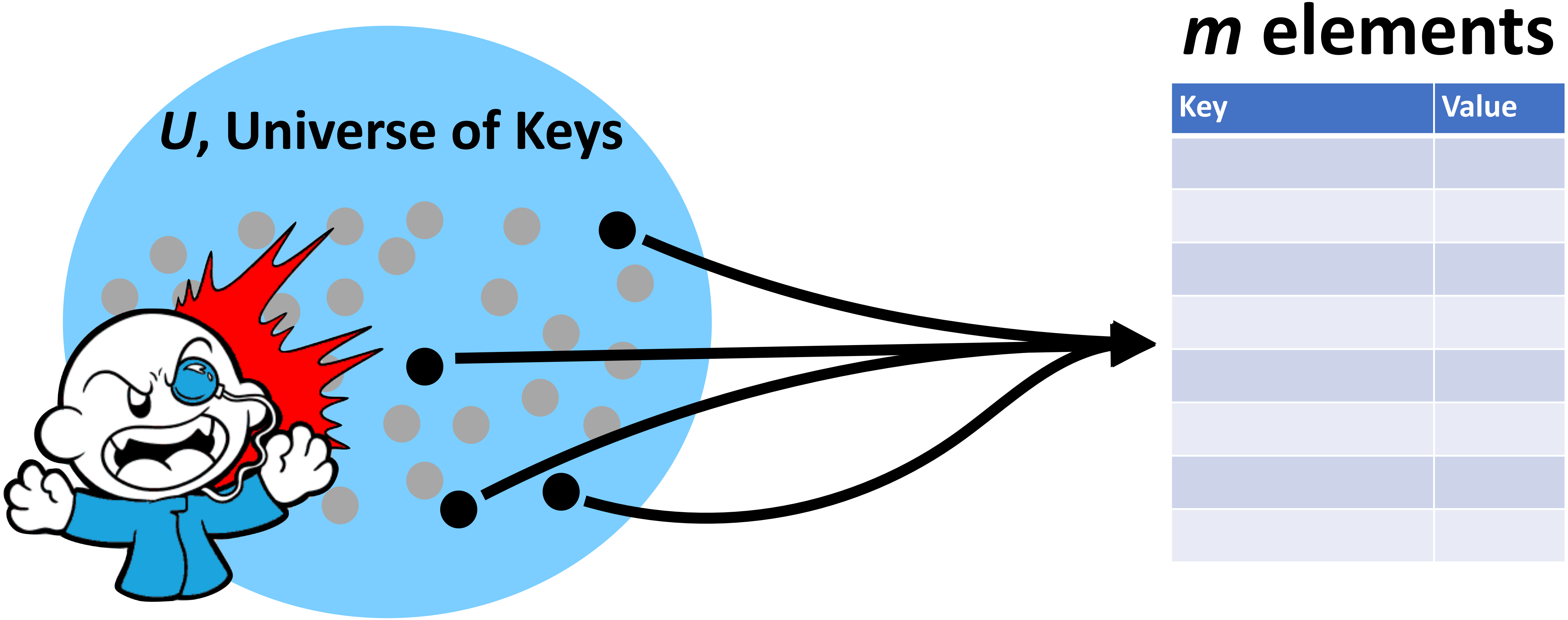


Image by Matthew Loffhagen

A Hash Table based Dictionary

```
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

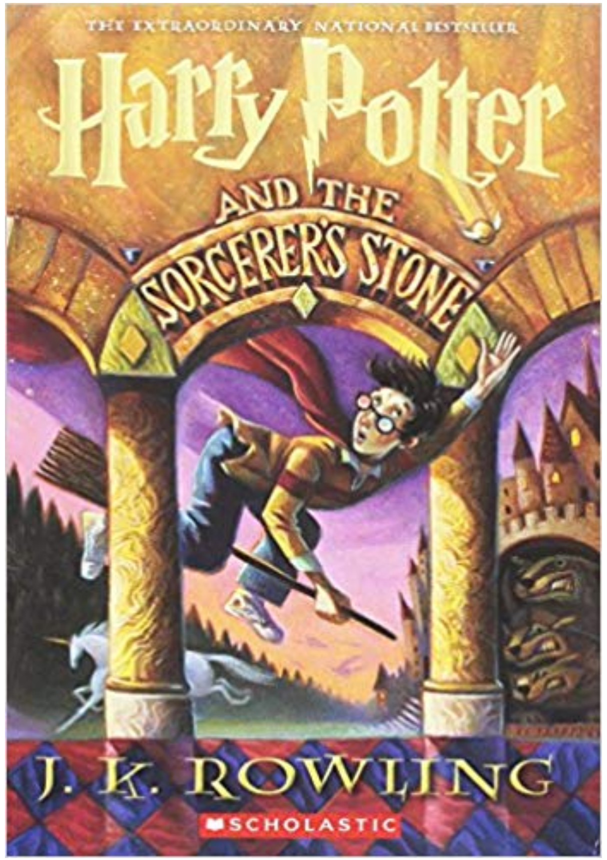
Addressing hash collisions depends on your storage structure.

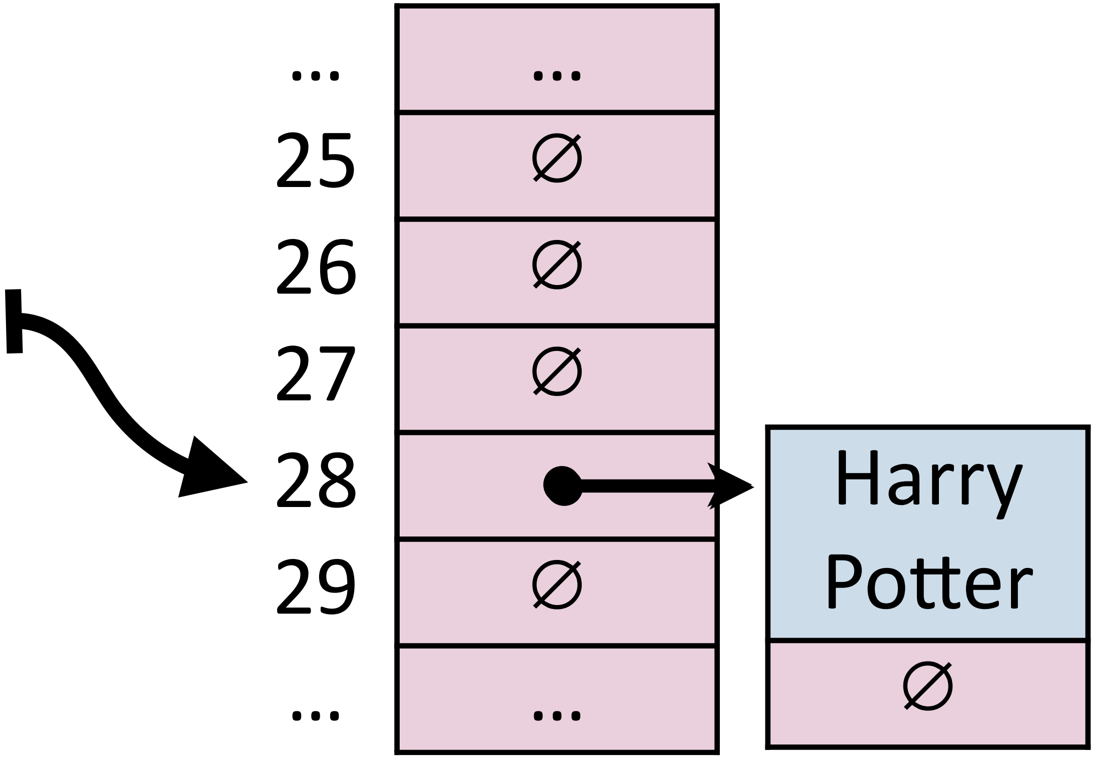
- **Open Hashing:**

- **Closed Hashing:**

Open Hashing

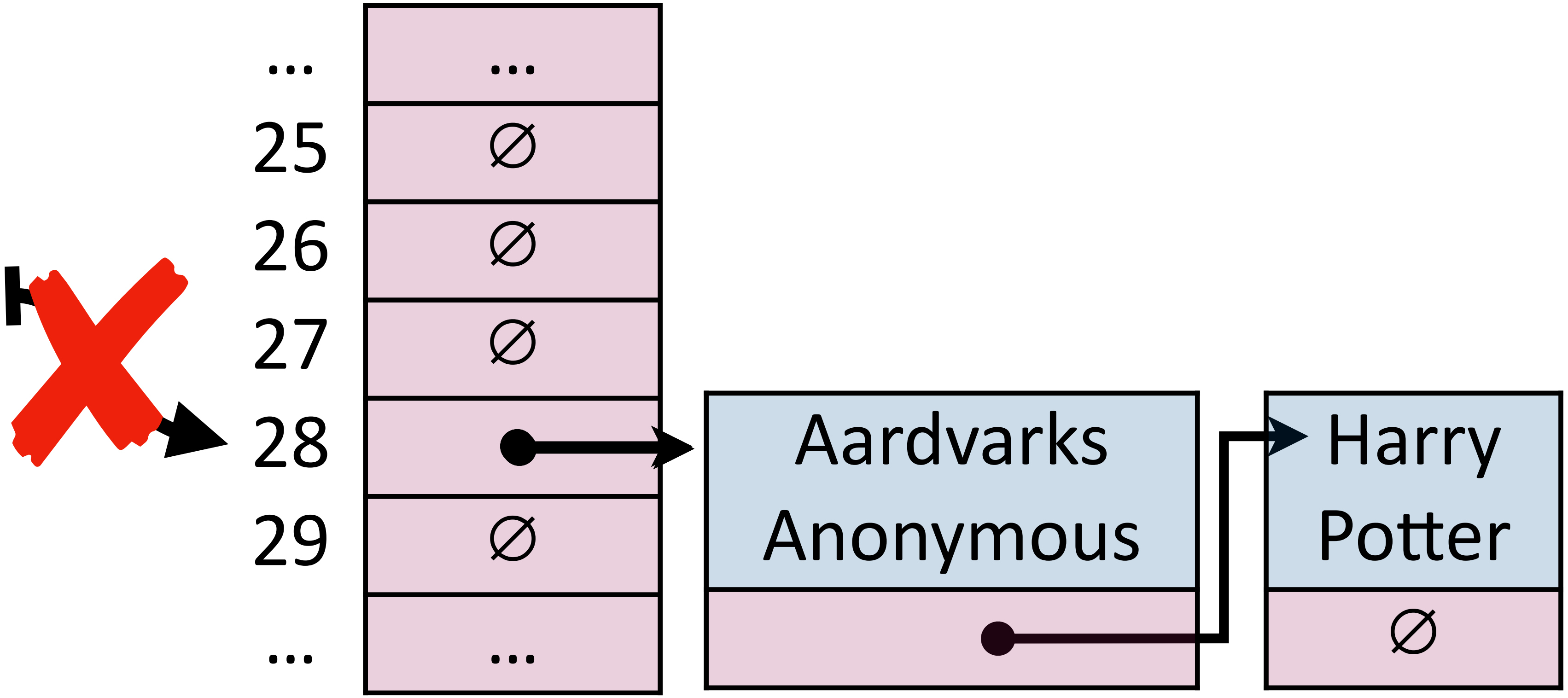
In an ***open hashing*** scheme, key-value pairs are stored externally (for example as a linked list).





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")
```

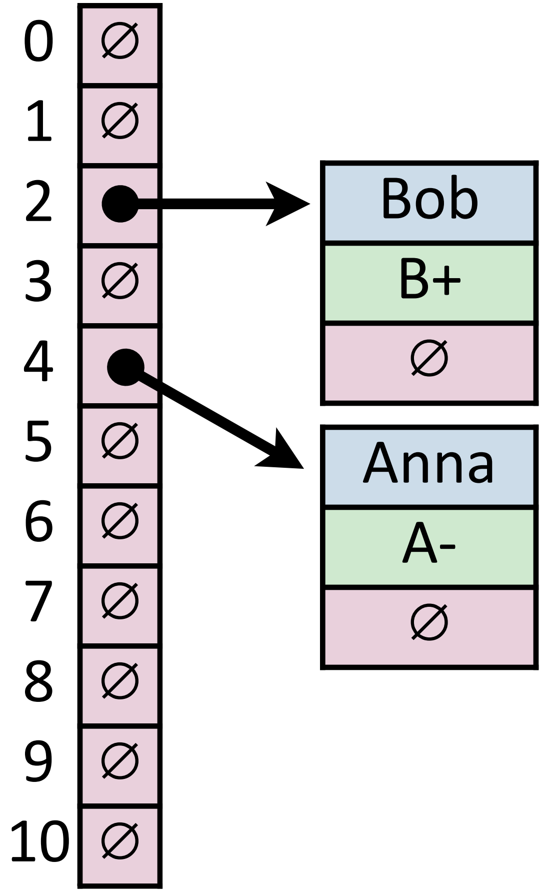
Key	Value	Hash
Bob	**B+**	**2**
Anna	**A-**	**4**
Alice	A+	4
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7

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

Insertion (Separate Chaining)

`_insert("Alice")`

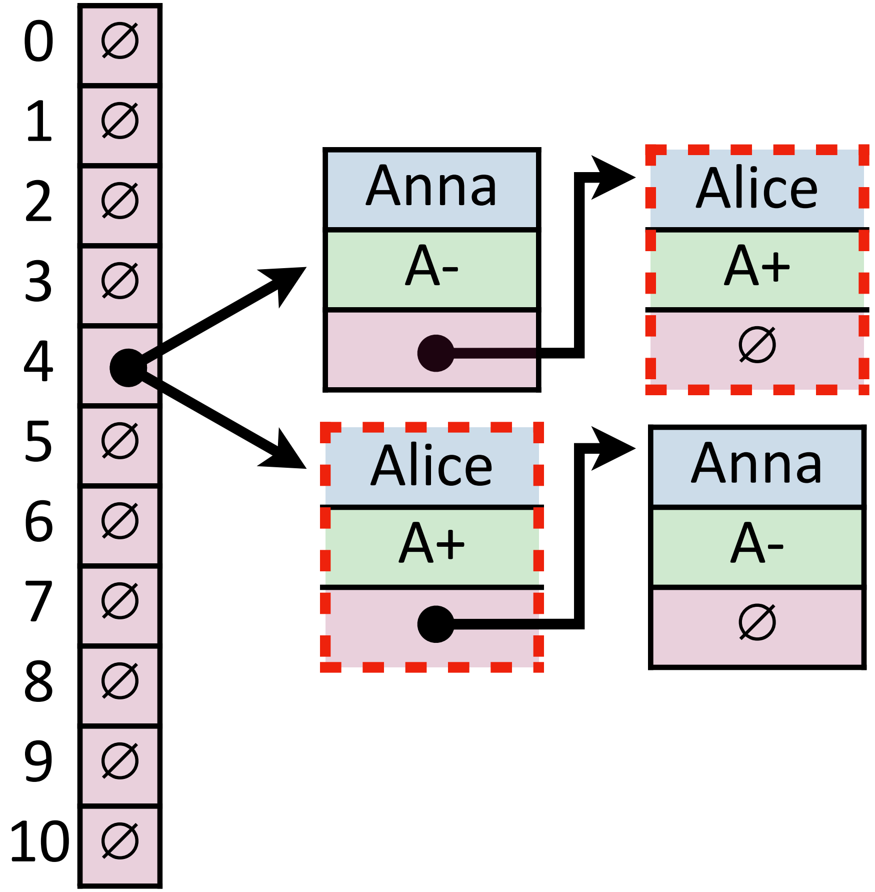
Key	Value	Hash
Bob	B+	2
Anna	A-	4
Alice	**A+**	**4**
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7



Insertion (Separate Chaining)

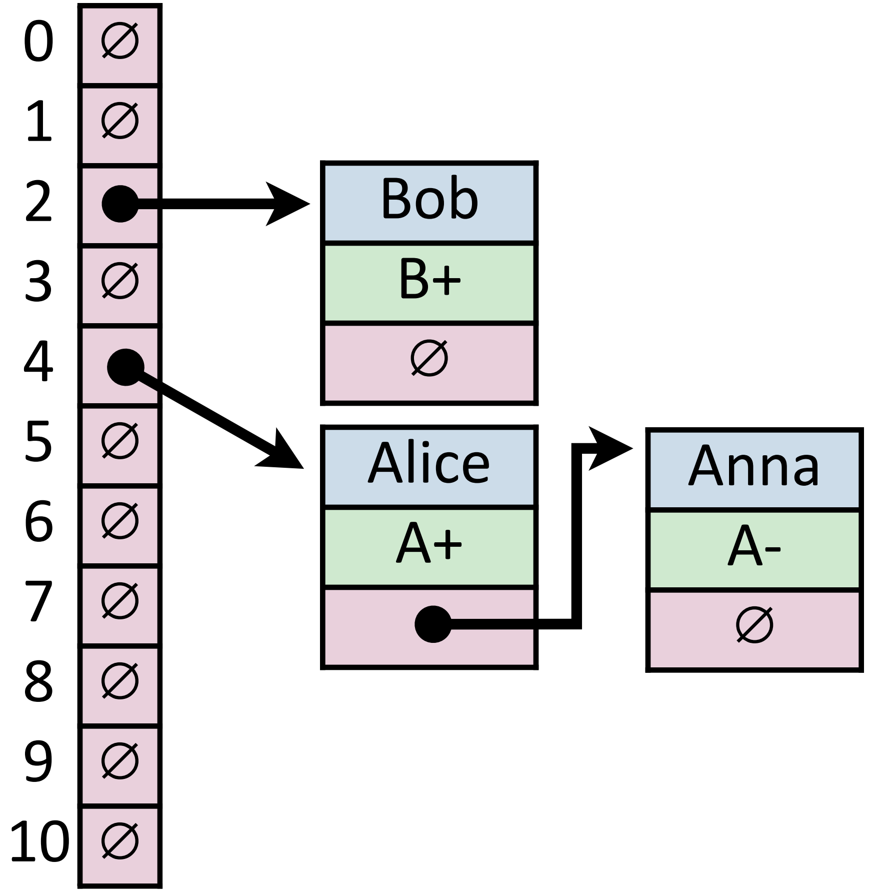
Where does Alice end up relative to Anna in the chain?

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



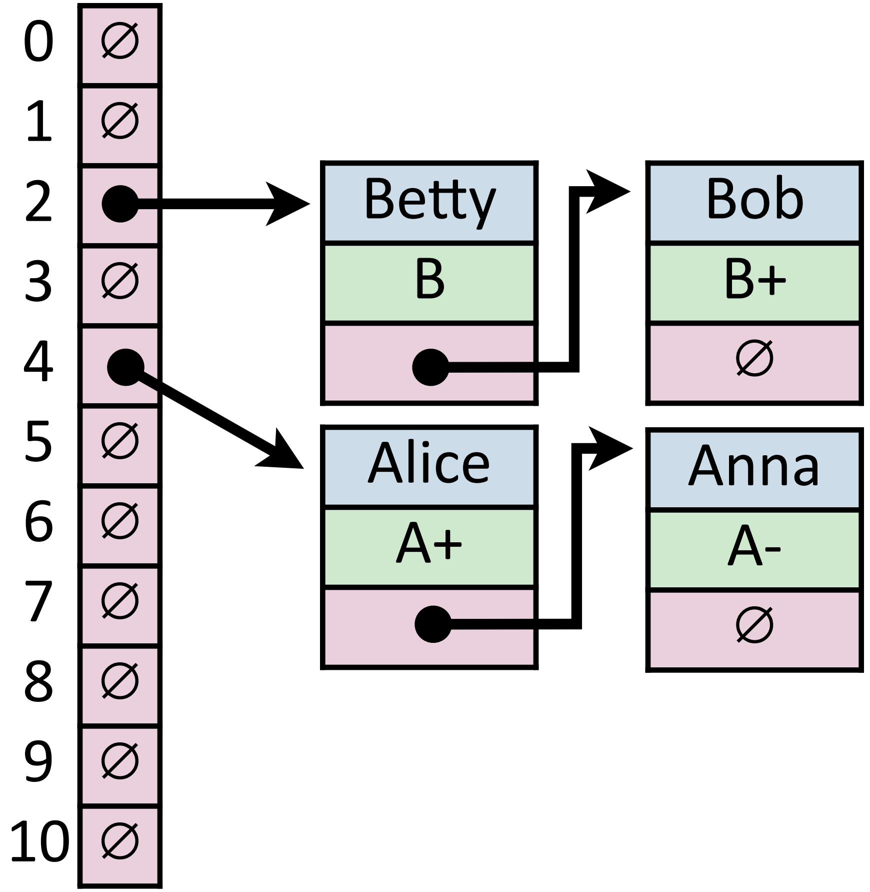
Insertion (Separate Chaining)

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



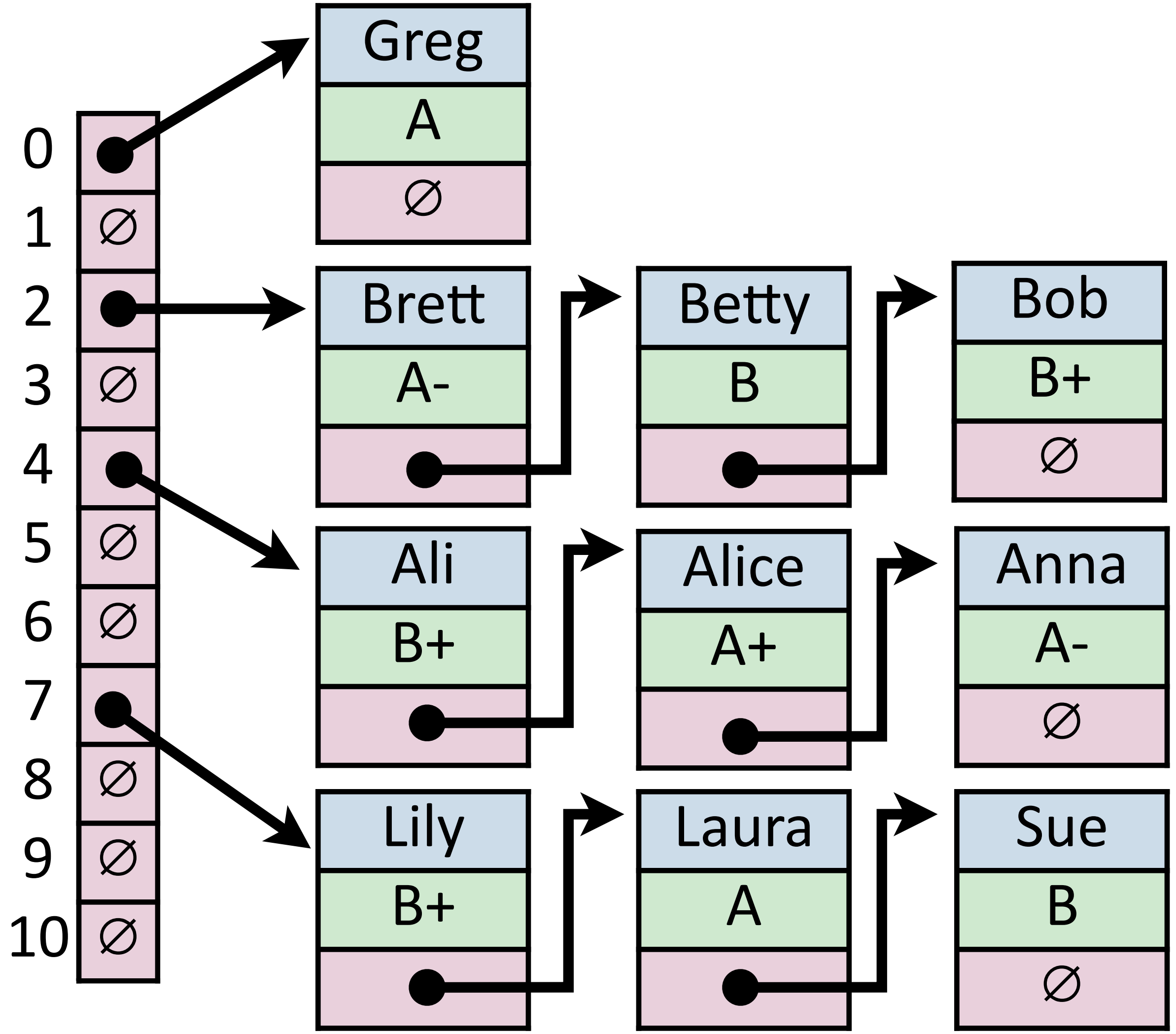
Insertion (Separate Chaining)

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)

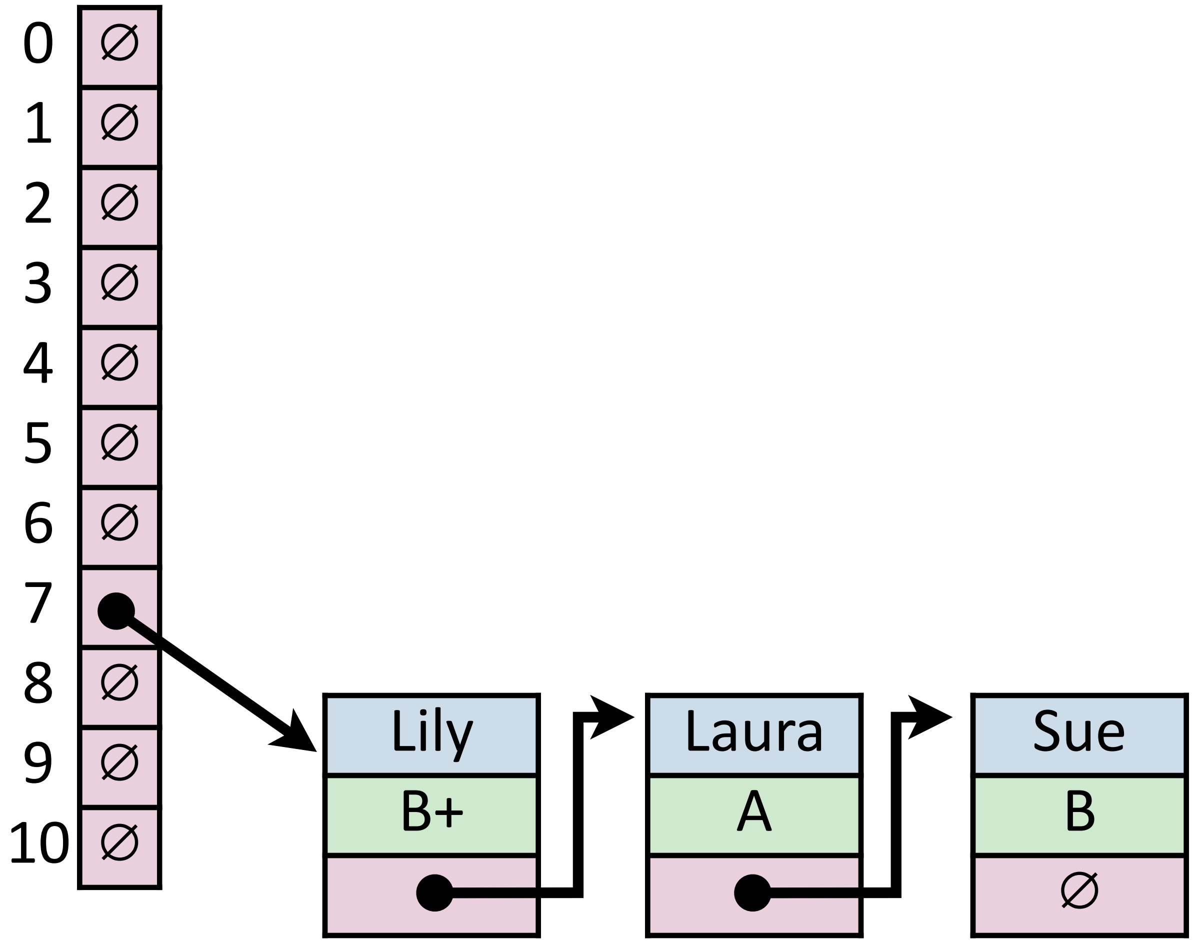
Key	Value	Hash
Bob	B+	2
Anna	A-	4
Alice	A+	4
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7



Find (Separate Chaining)

`_find("Sue")`

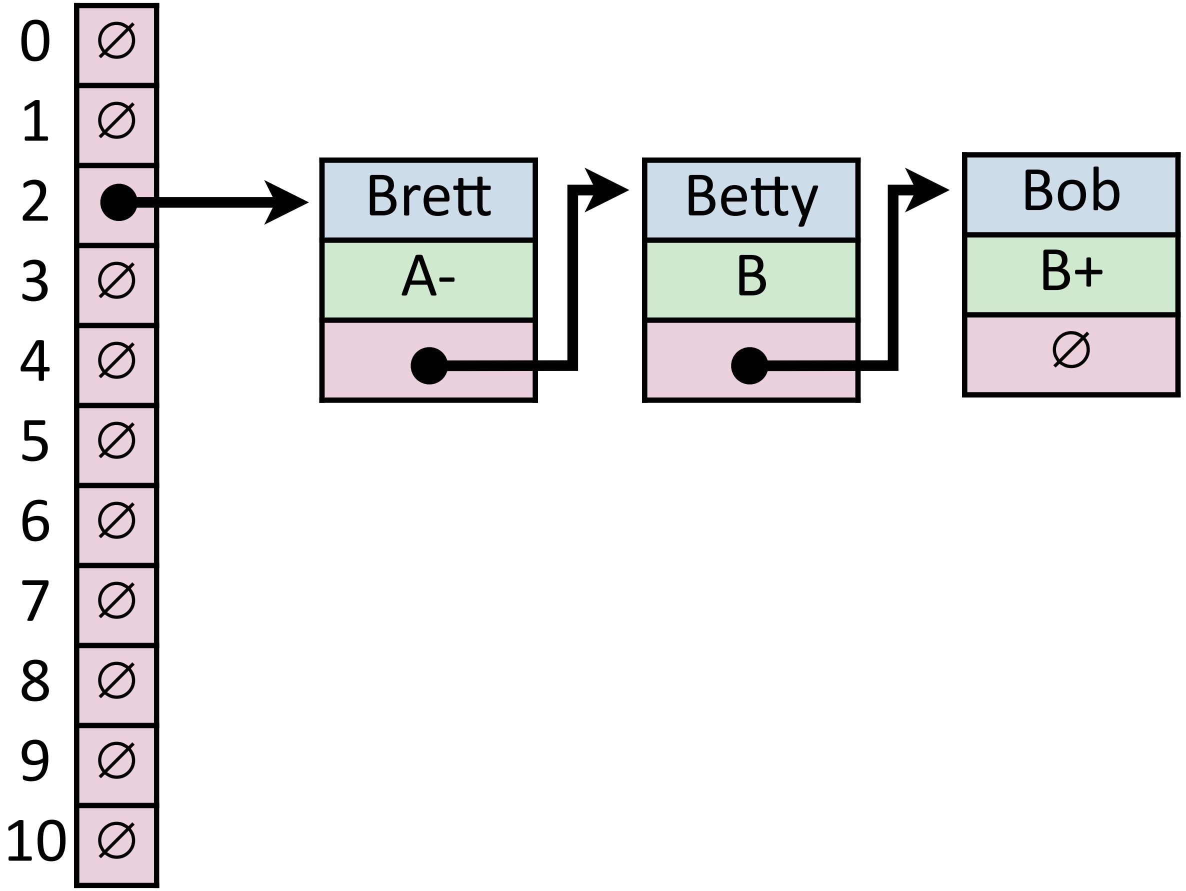
Key	Hash
Sue	7



Remove (Separate Chaining)

`_remove("Betty")`

Key	Hash
Betty	2



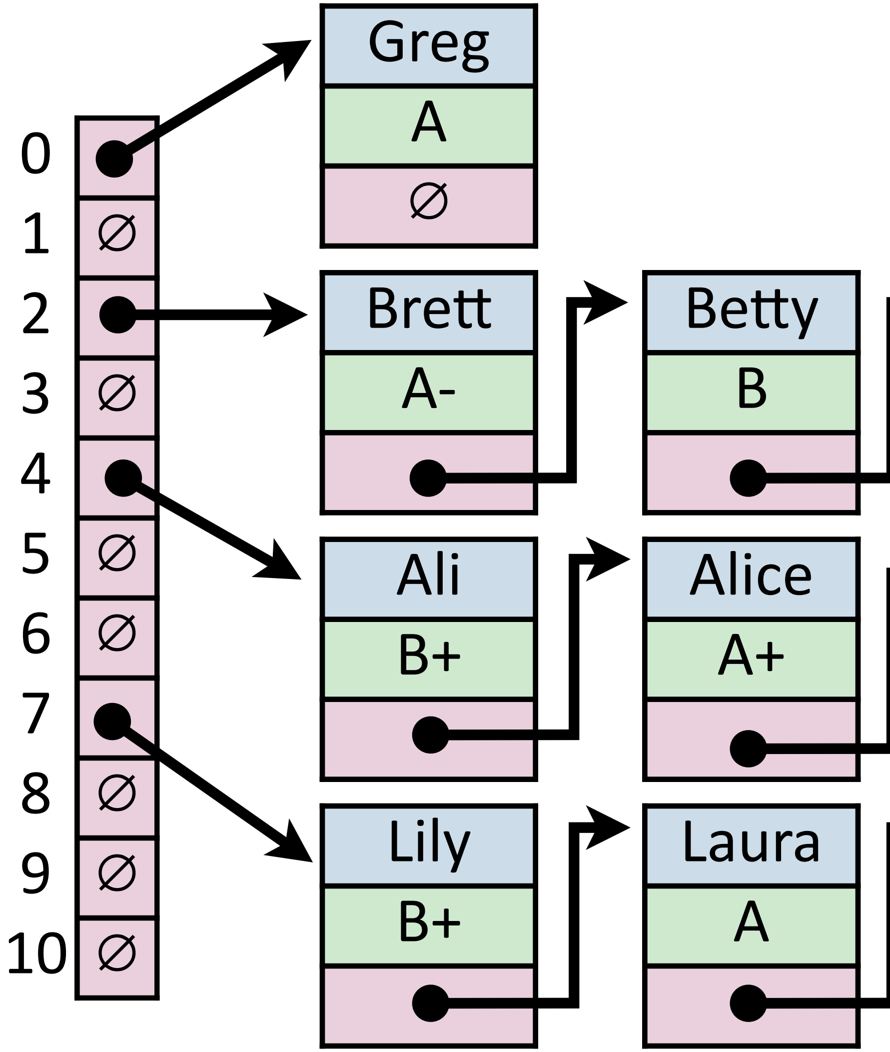
Hash Table (Separate Chaining)

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

Find runs in: ______________________

Insert runs in: ______________________

Remove runs in: ______________________



Fundamentals of Probability

Imagine you roll a pair of six-sided dice.

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

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

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:

$$E[X] = \sum_{x \in \Omega} Pr\{X = x\} \cdot x$$

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,

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

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

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

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

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,

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

Hash Table

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

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

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

Uniform:

Independent:

Separate Chaining Under SUHA

Given table of size m and n inserted objects

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

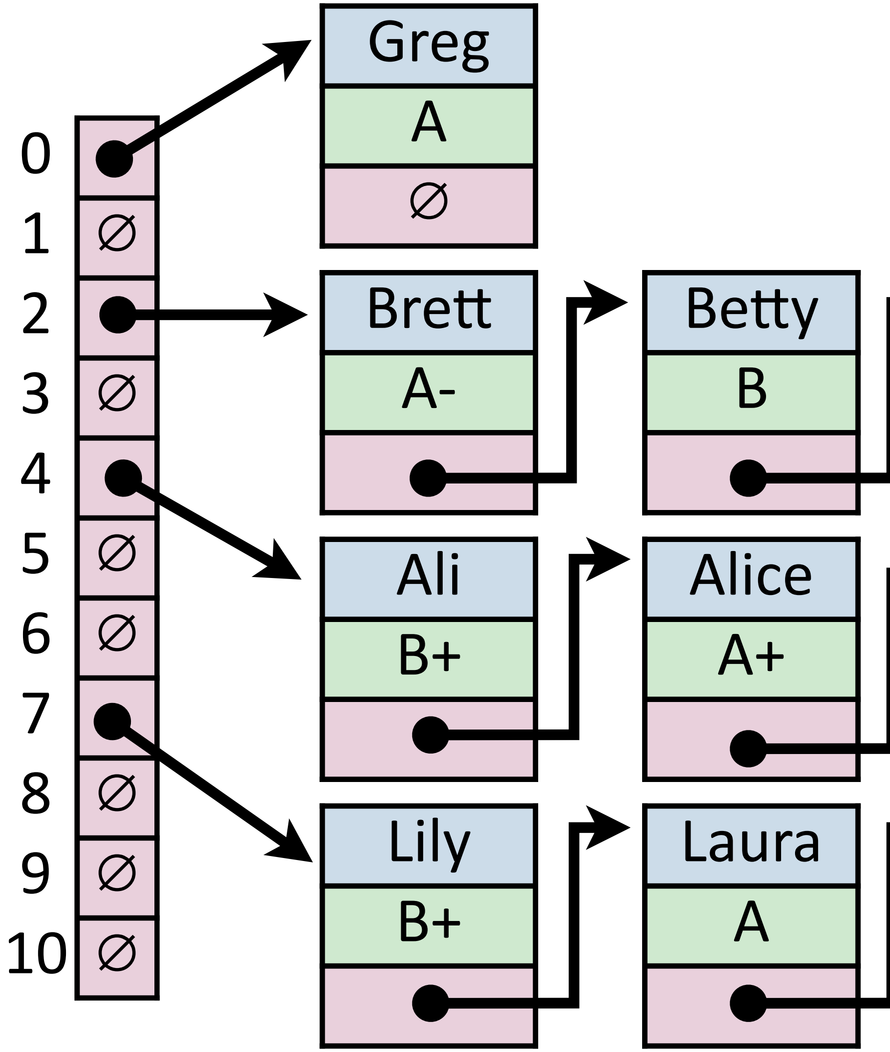
Hash Table (Separate Chaining w/ SUHA)

For 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	