Data Structures and Algorithms

Hashing

CS 225
Brad Solomon

October 27, 2023

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN

Department of Computer Science

Extra Credit Projects — Submit by 10/31

Randomization in Algorithms

1. Assume input data is random to estimate average-case performance

2. Use randomness inside algorithm to estimate expected running time

3. Use randomness inside algorithm to approximate solution in fixed time

Learning Objectives

Motivate and formally define a hash table

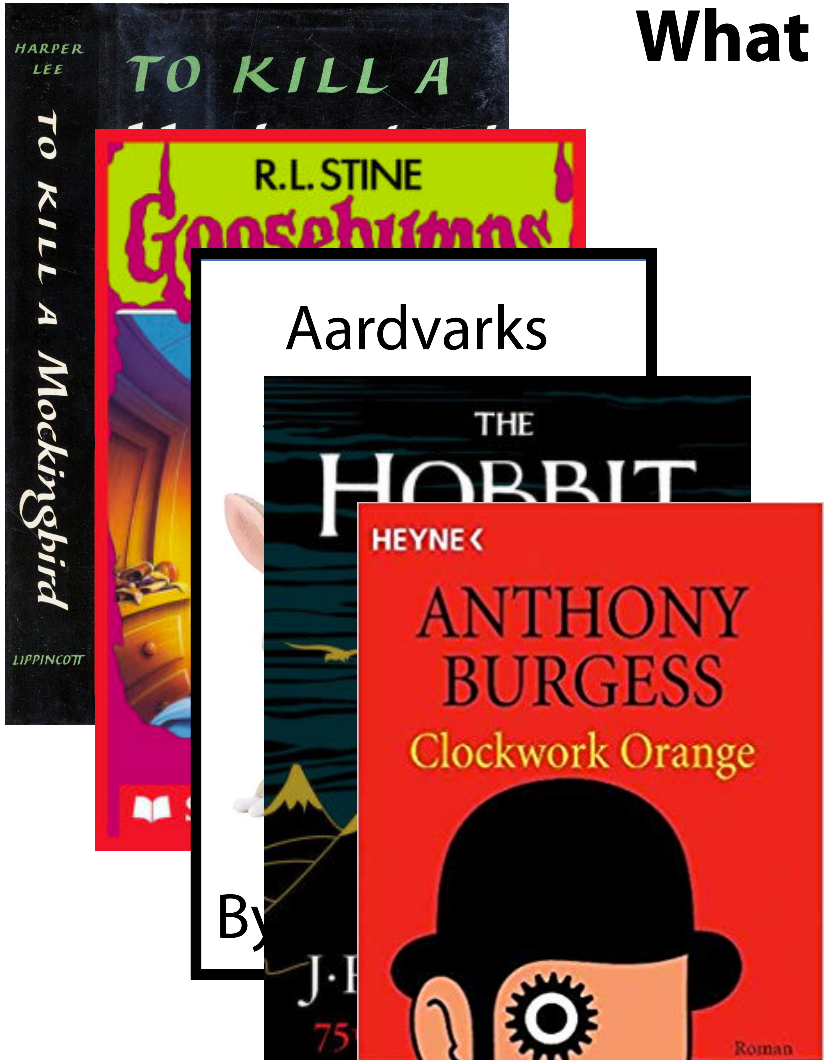
Discuss what a 'good' hash function looks like

Identify the key weakness of a hash table

Introduce strategies to "correct" this weakness

Data Structure Review

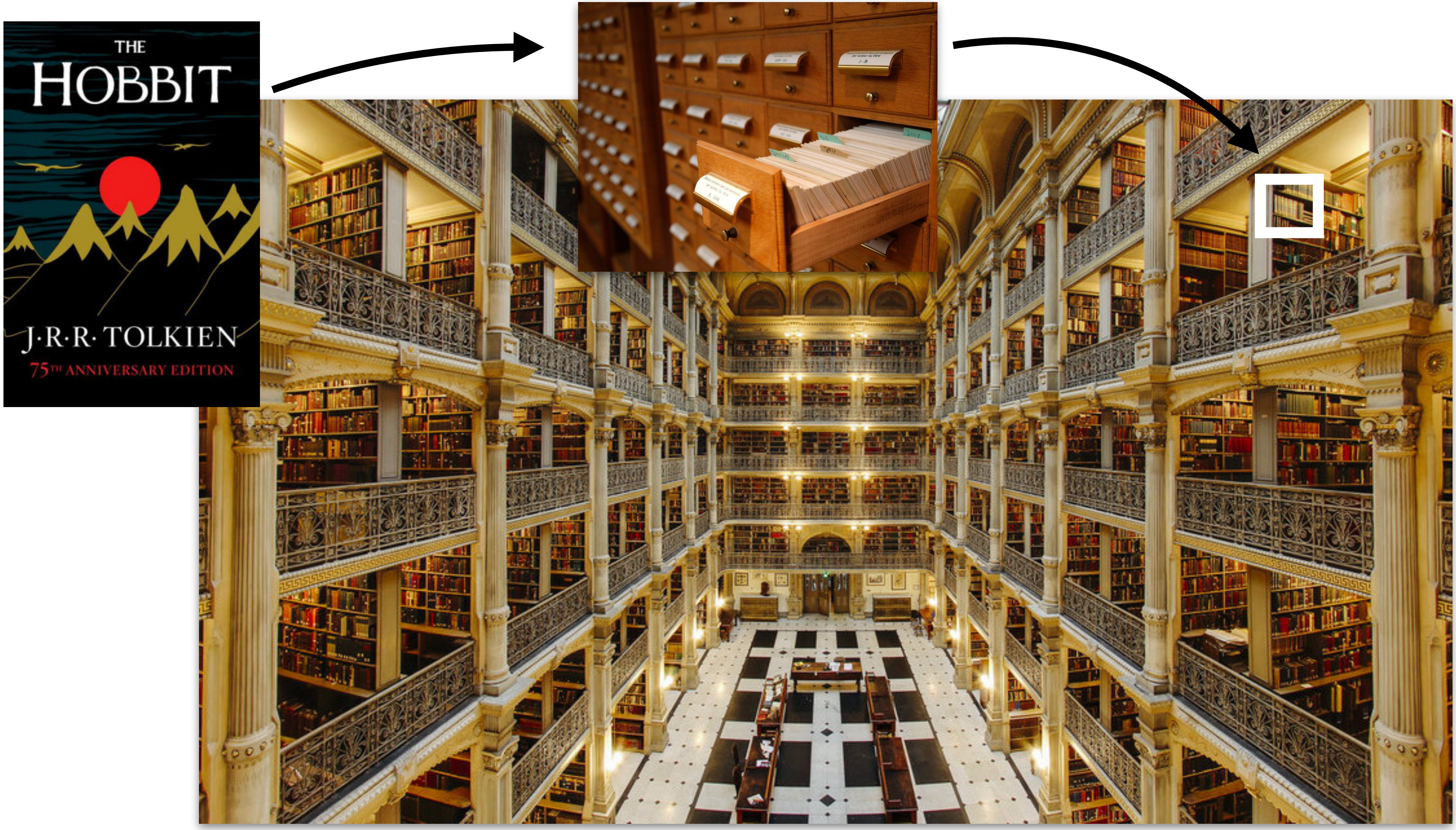
I have a collection of books and I want to store them in a dictionary!



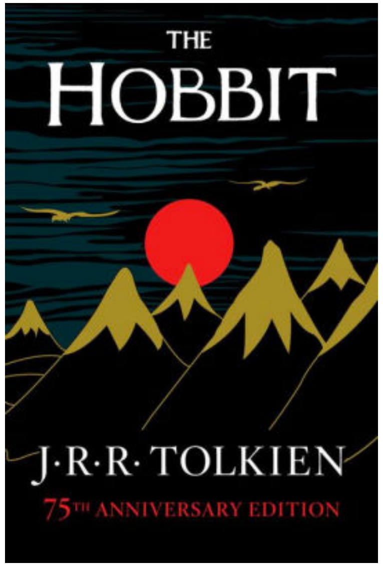
What data structures can I use here?

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

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



A Hash Table based Dictionary

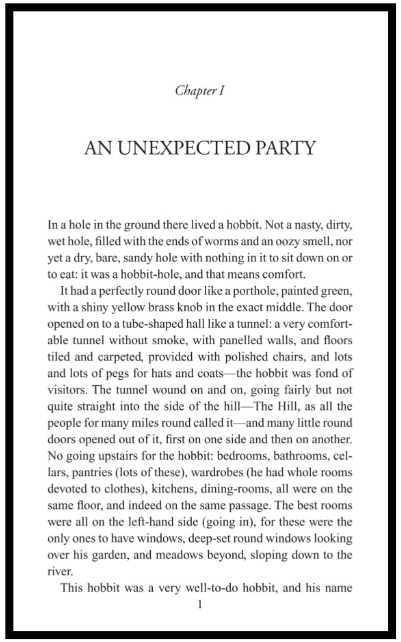


ISBN: 9780062265722

Call #: PR
6068.O93
H35 1937

ISBN: 9780062265722

Call #: PR
6068.O93
H35 1937



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

Randomized Data Structures

Sometimes a data structure can be **too ordered / too structured**

Randomized data structures rely on **expected** performance

Randomized data structures ‘cheat’ tradeoffs!

A Hash Table based Dictionary

User Code (is a map):

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

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

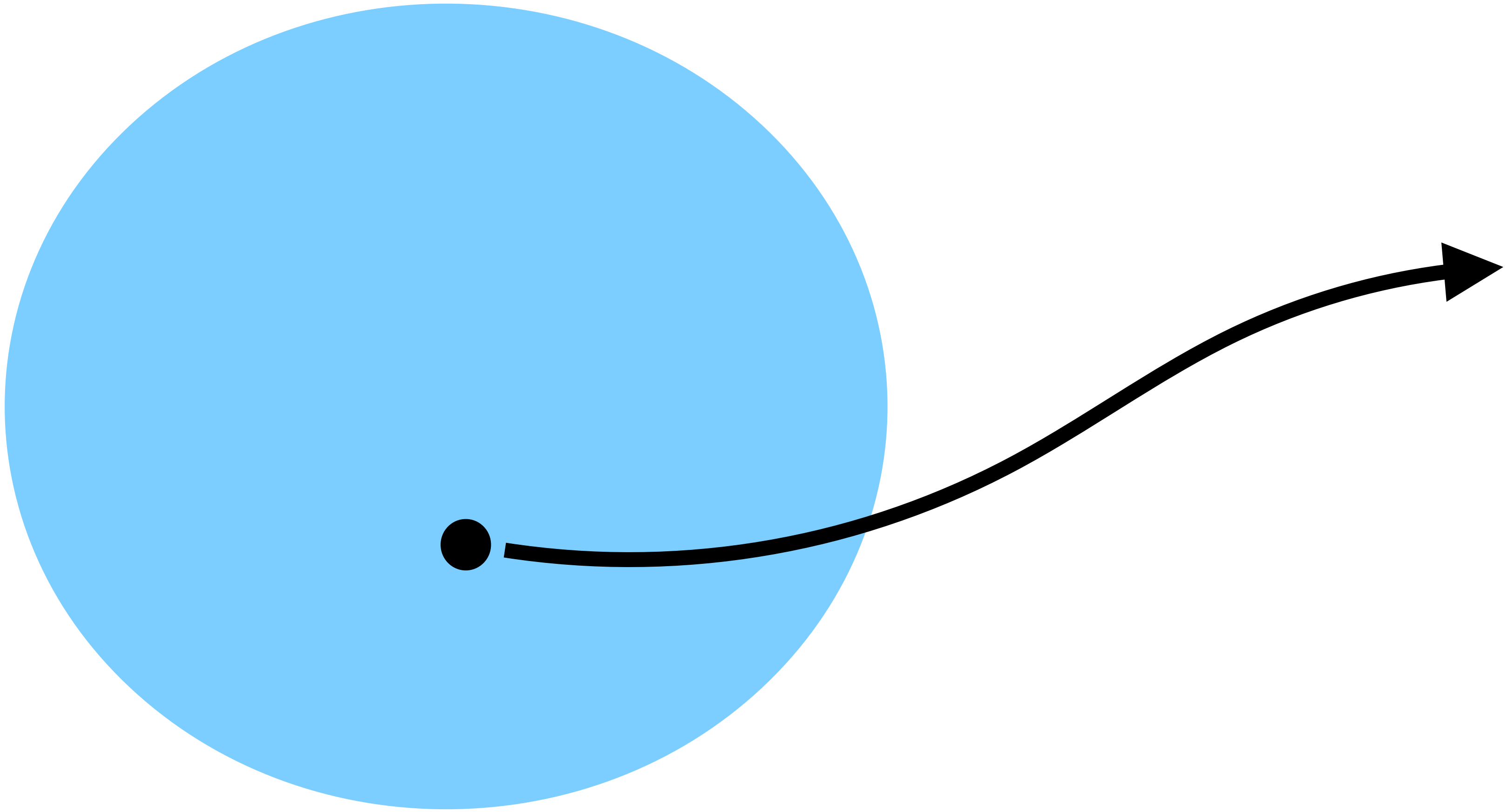
1.

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

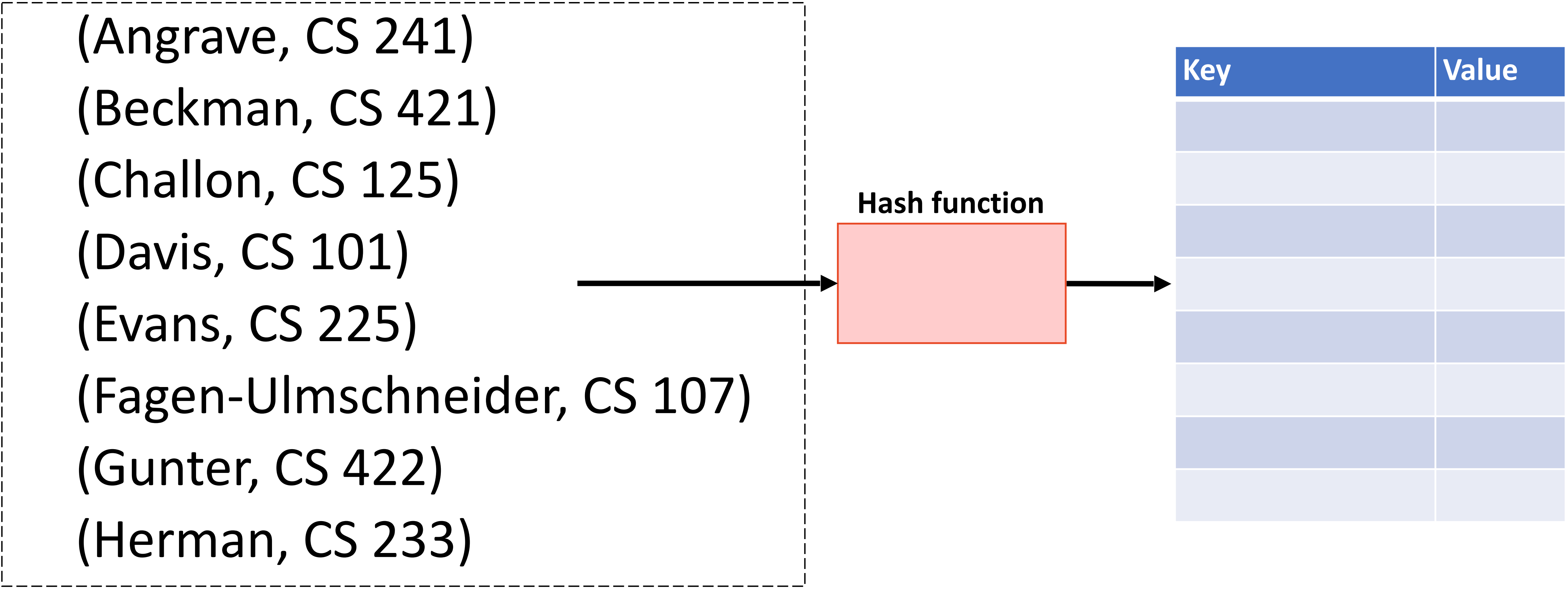
A hash function ***must*** be:

- **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)

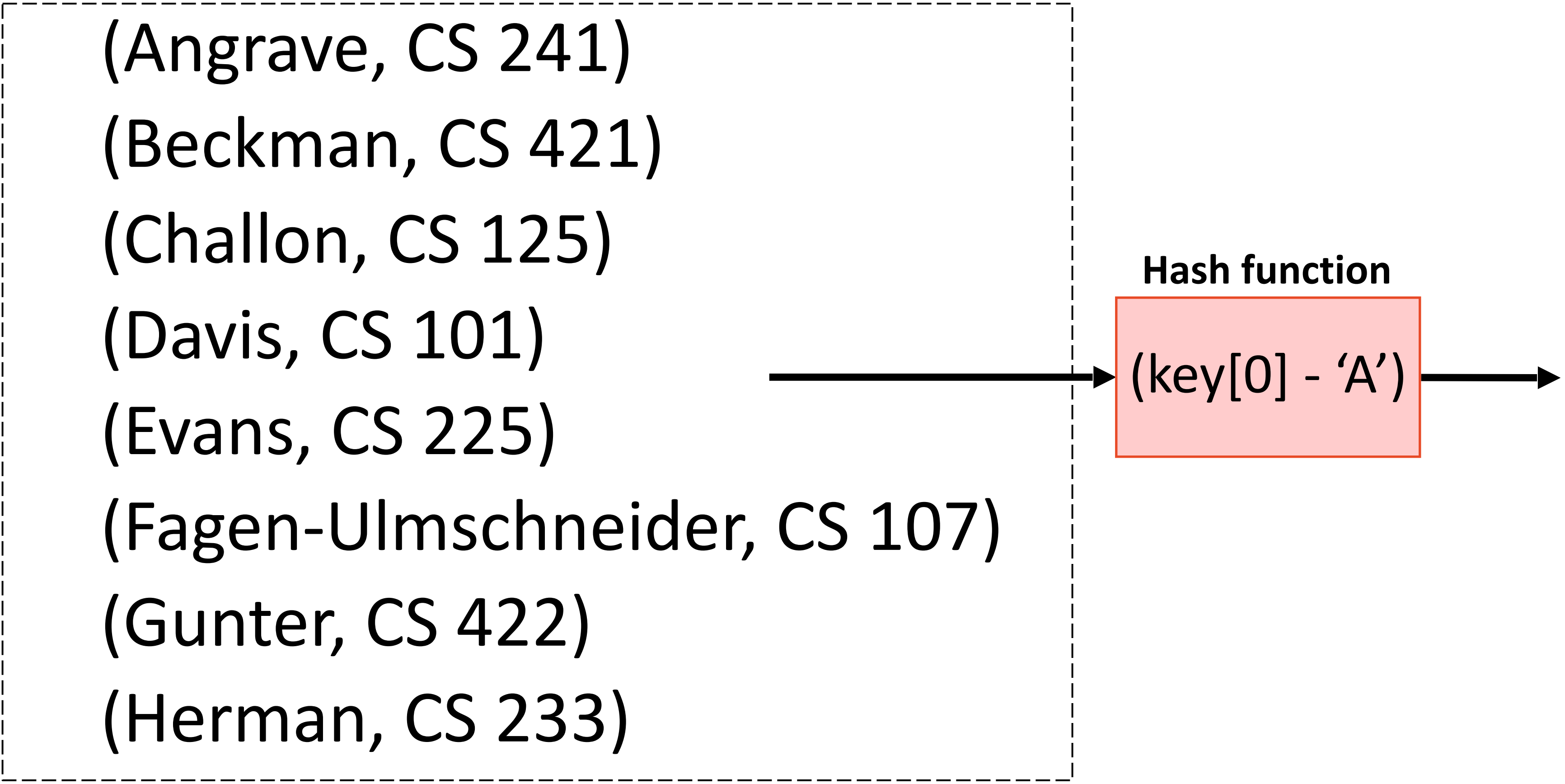
(Fagen-Ulmschneider, CS 107)

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

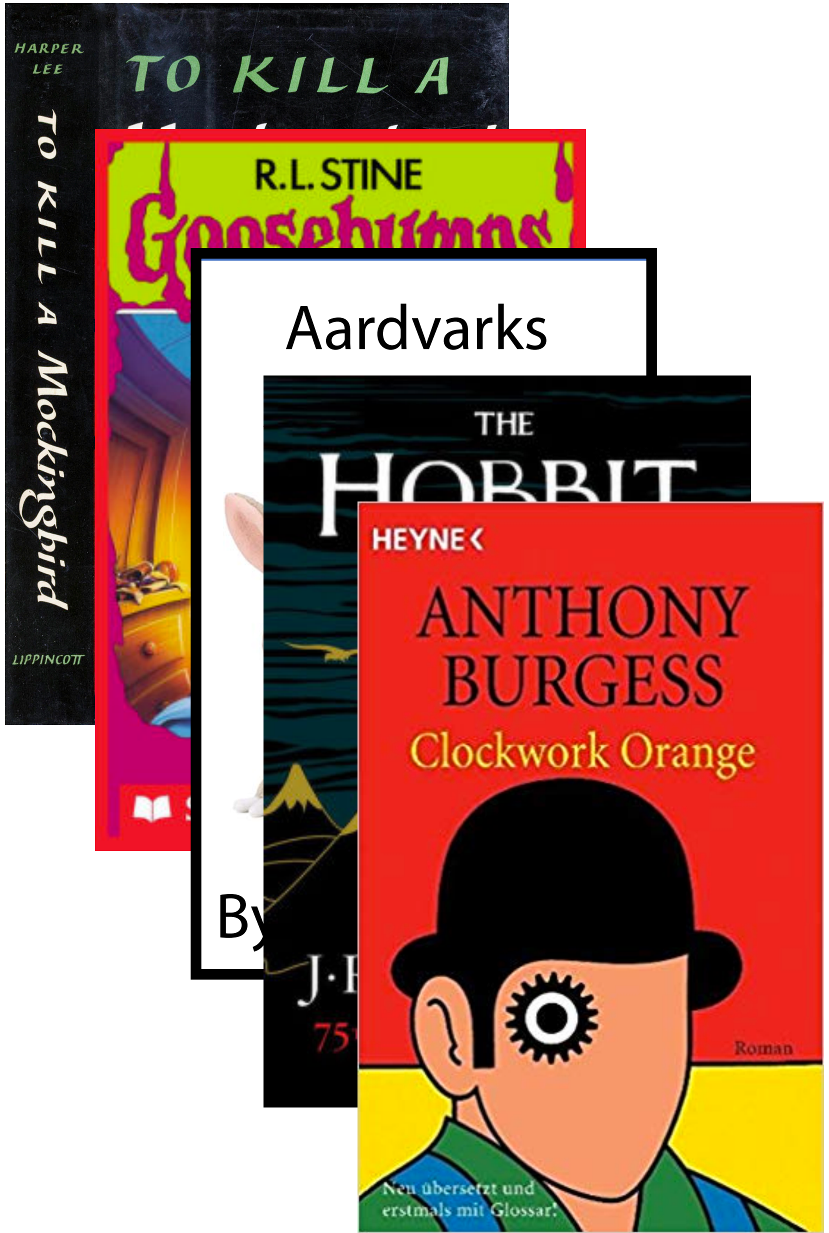
- A **hash**:

- A **compression**:

Choosing a good hash function is tricky…

- Don’t create your own (yet*)

Hash Function

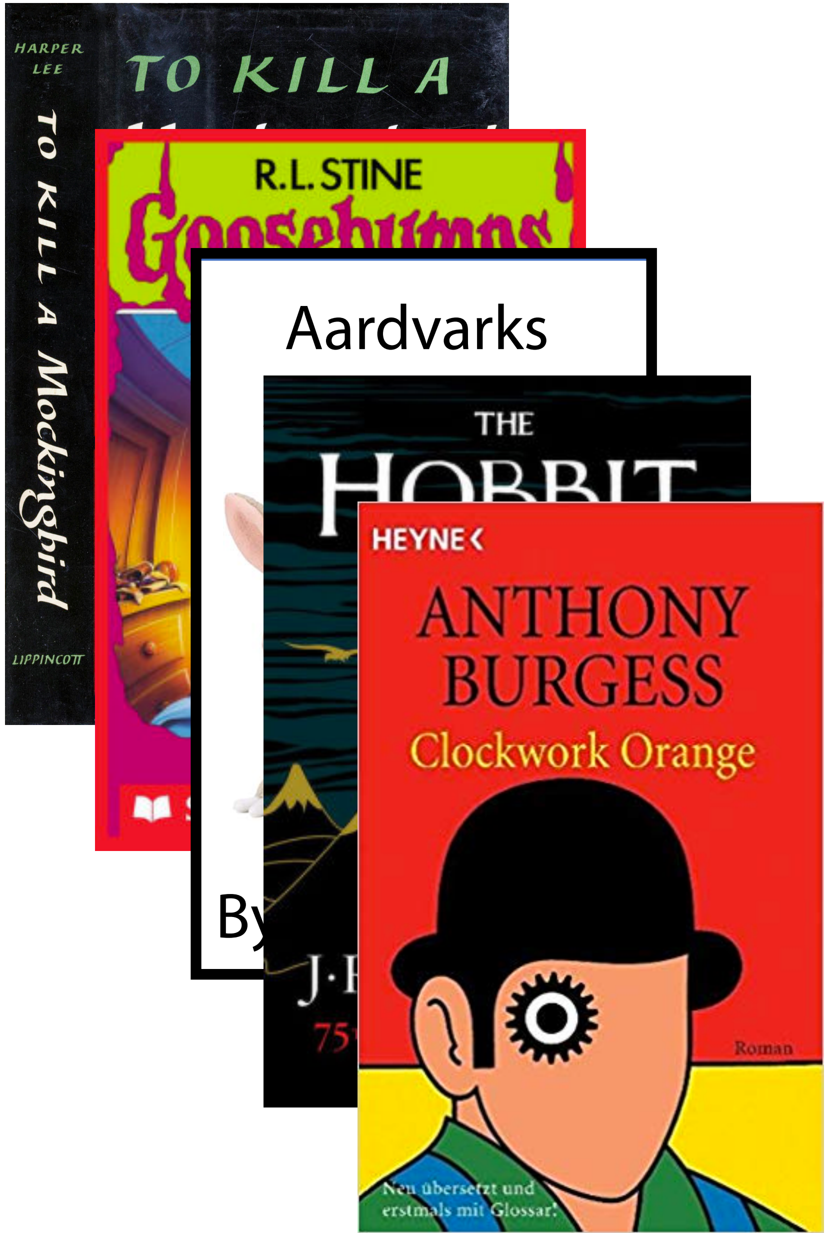


$$h(k) = \left(k\,.\,firstName[0] + k\,.\,lastName[0]\right)\,\%\,m$$

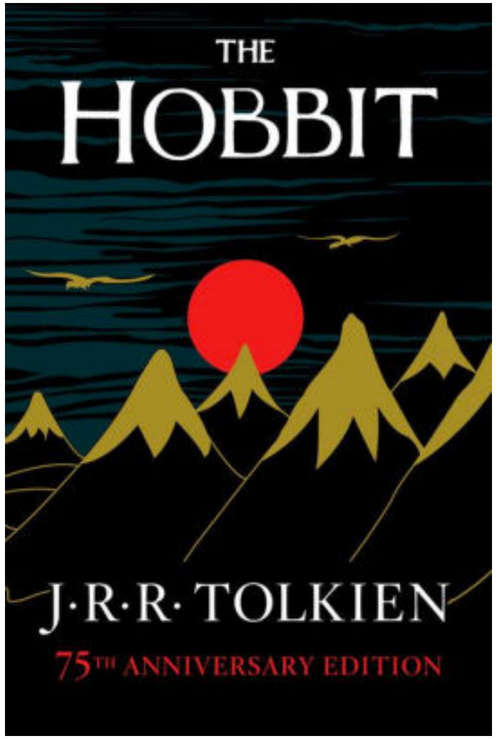
$$h(k) = \left(rand() * k\,.\,numPages\right)\,\%\,m$$

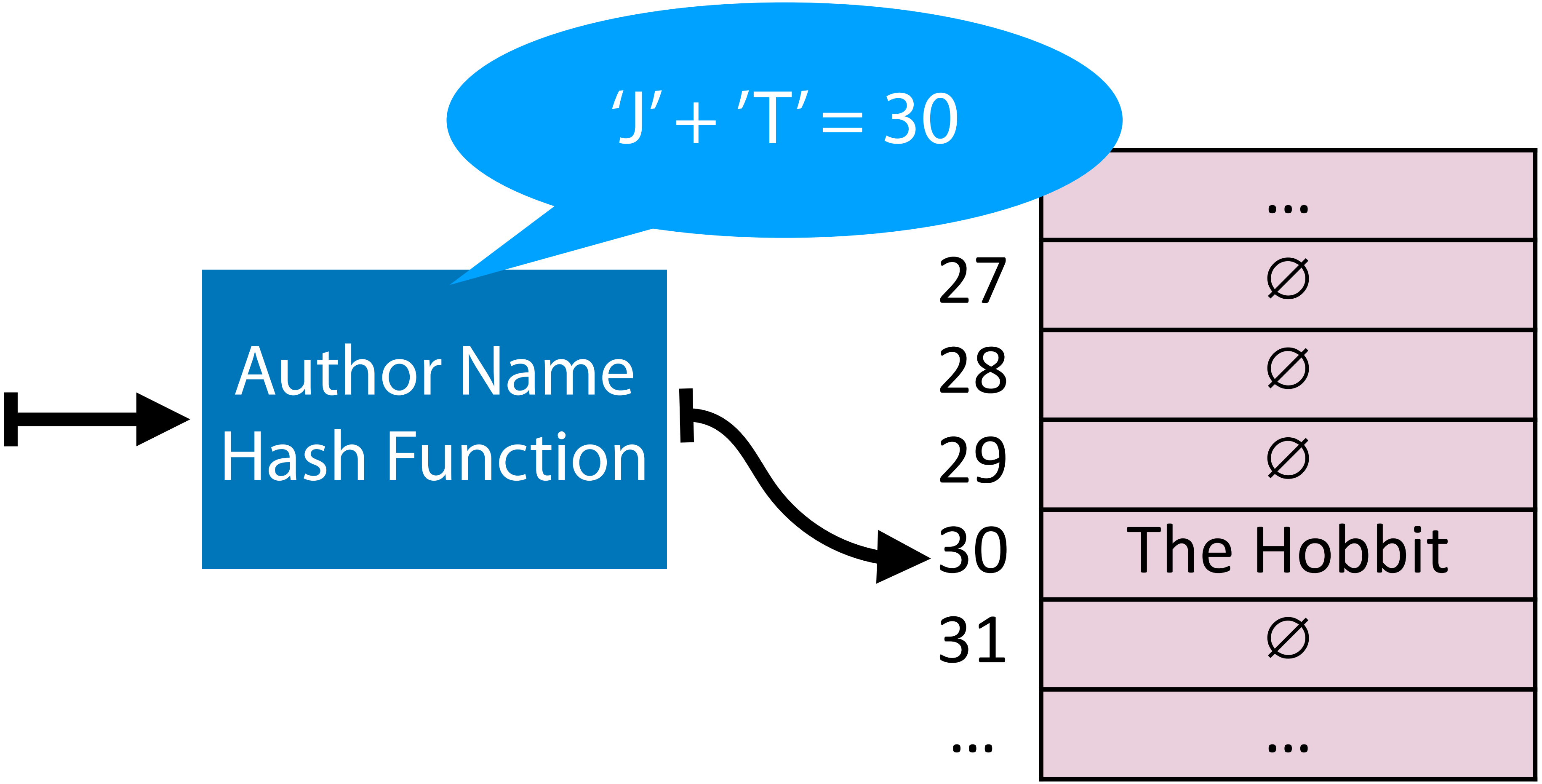
$$h(k) = (\text{Order I insert [Order seen]})\,\%\,m$$

Hash Function



A Hash Table based Dictionary



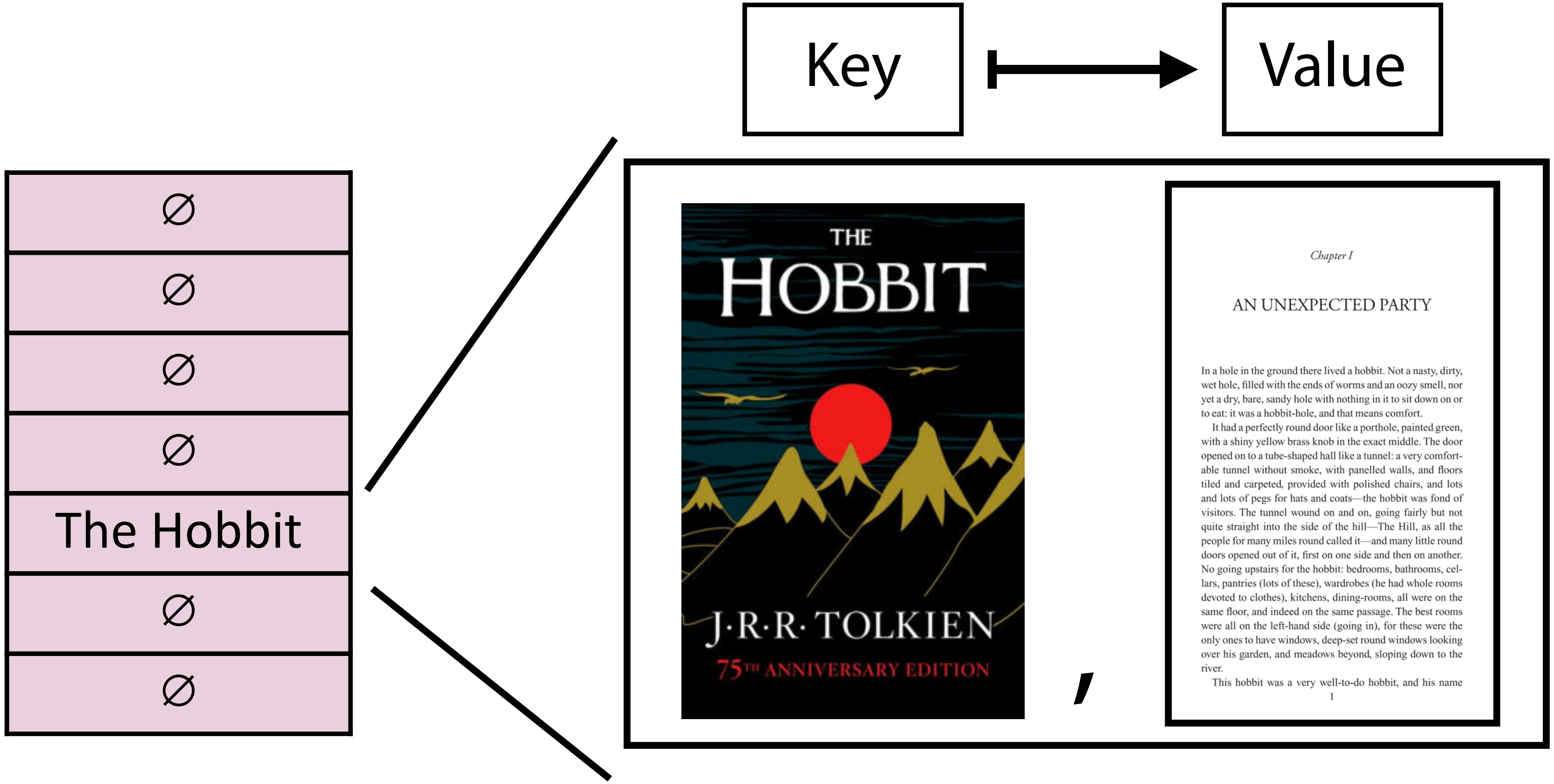


'J' + 'T' = 30

Author Name Hash Function

	…
27	∅
28	∅
29	∅
30	The Hobbit
31	∅
…	…

A Hash Table based Dictionary



Chapter I

AN UNEXPECTED PARTY

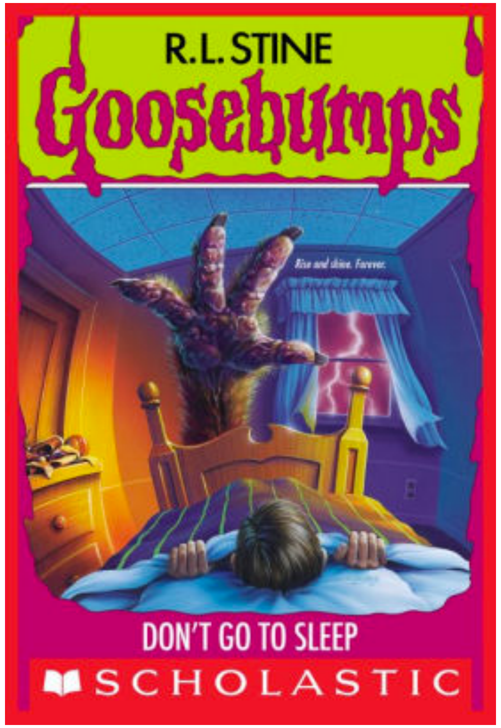
In a hole in the ground there lived a hobbit. Not a nasty, dirty, wet hole, filled with the ends of worms and an oozy smell, nor yet a dry, bare, sandy hole with nothing in it to sit down on or to eat: it was a hobbit-hole, and that means comfort.

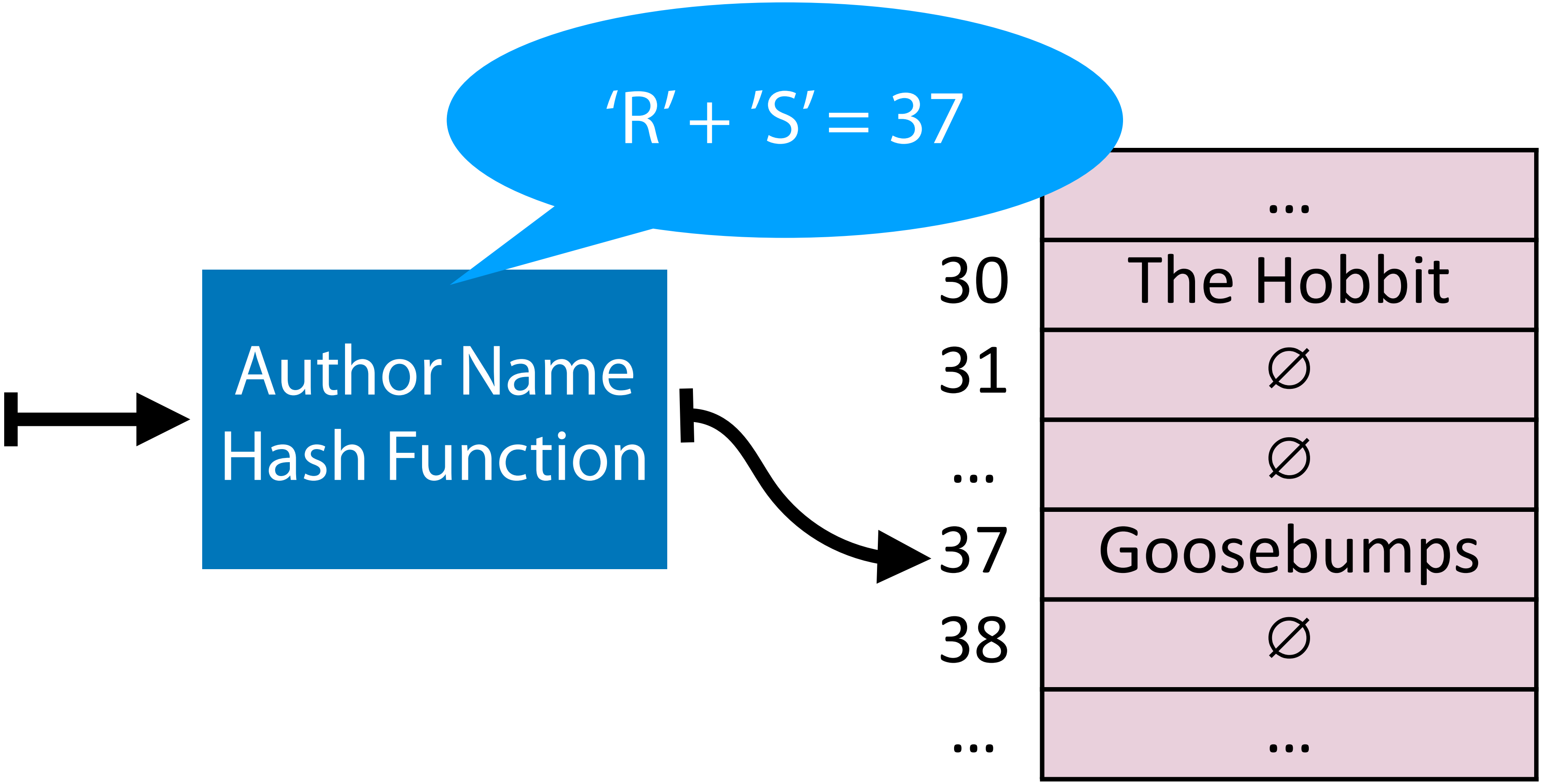
It had a perfectly round door like a porthole, painted green, with a shiny yellow brass knob in the exact middle. The door opened on to a tube-shaped hall like a tunnel: a very comfortable tunnel without smoke, with panelled walls, and floors tiled and carpeted, provided with polished chairs, and lots 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

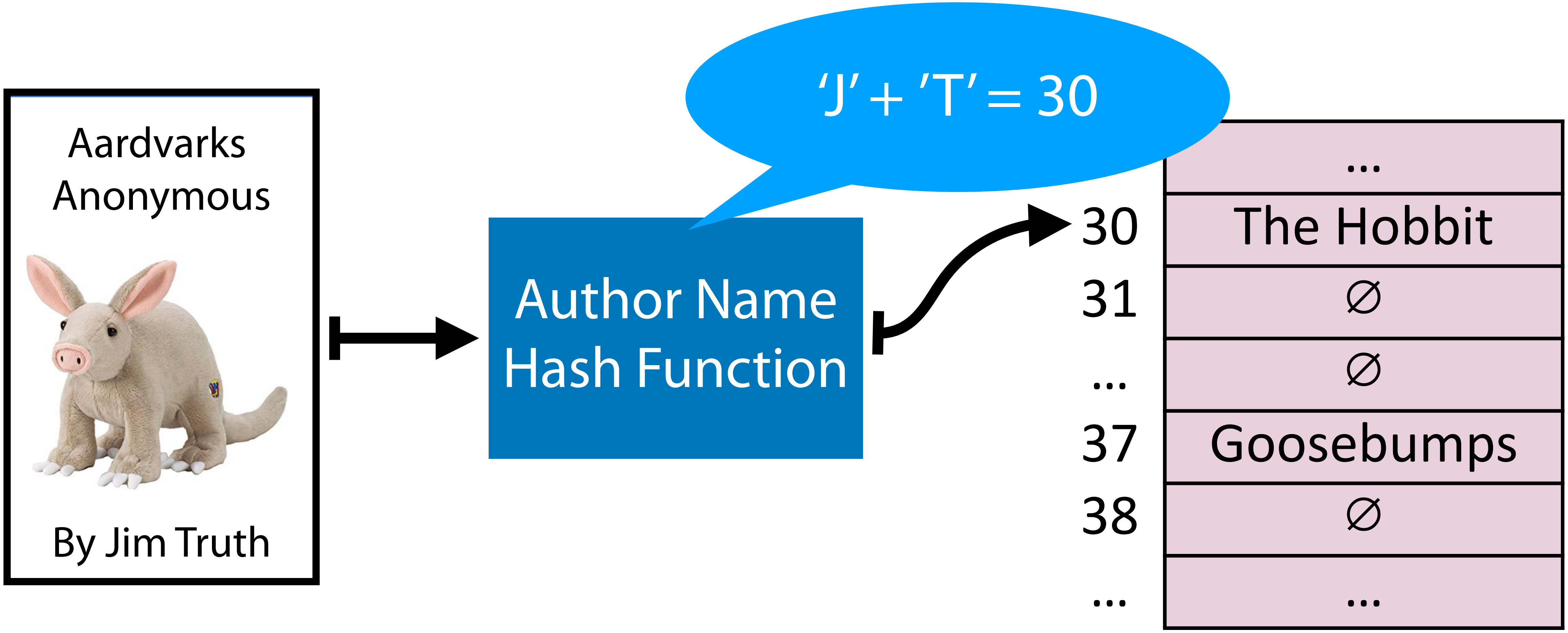
A Hash Table based Dictionary





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

A Hash Table based Dictionary



Aardvarks Anonymous

By Jim Truth

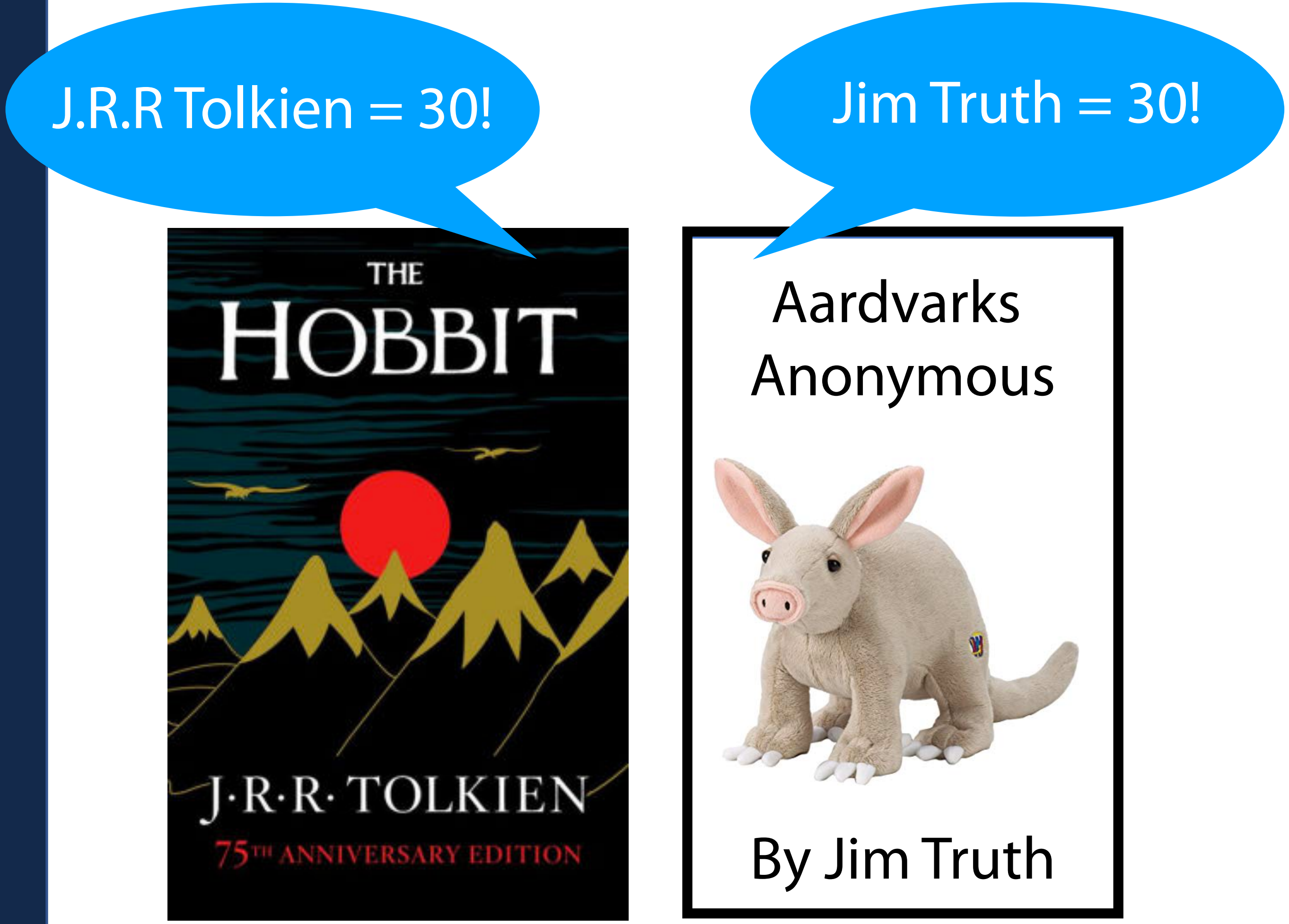
Author Name Hash Function

‘J’ + ’T’ = 30

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

Hash Collision

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



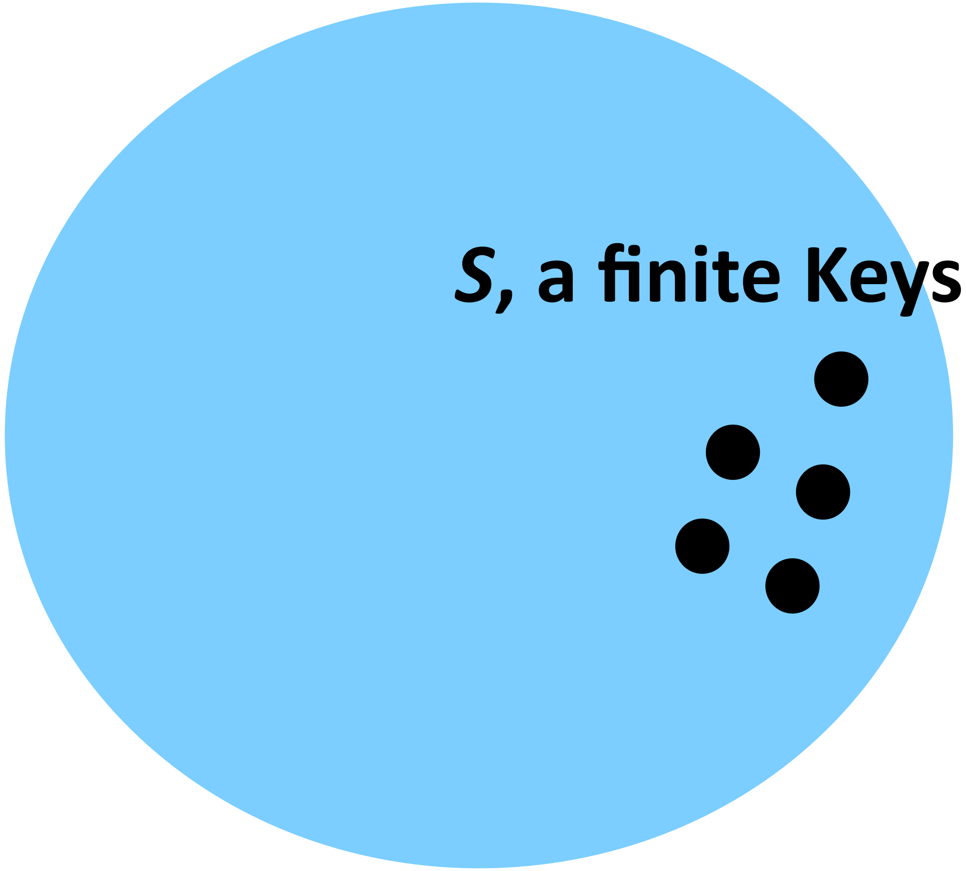
J.R.R Tolkien = 30!

Jim Truth = 30!

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

Perfect Hashing

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



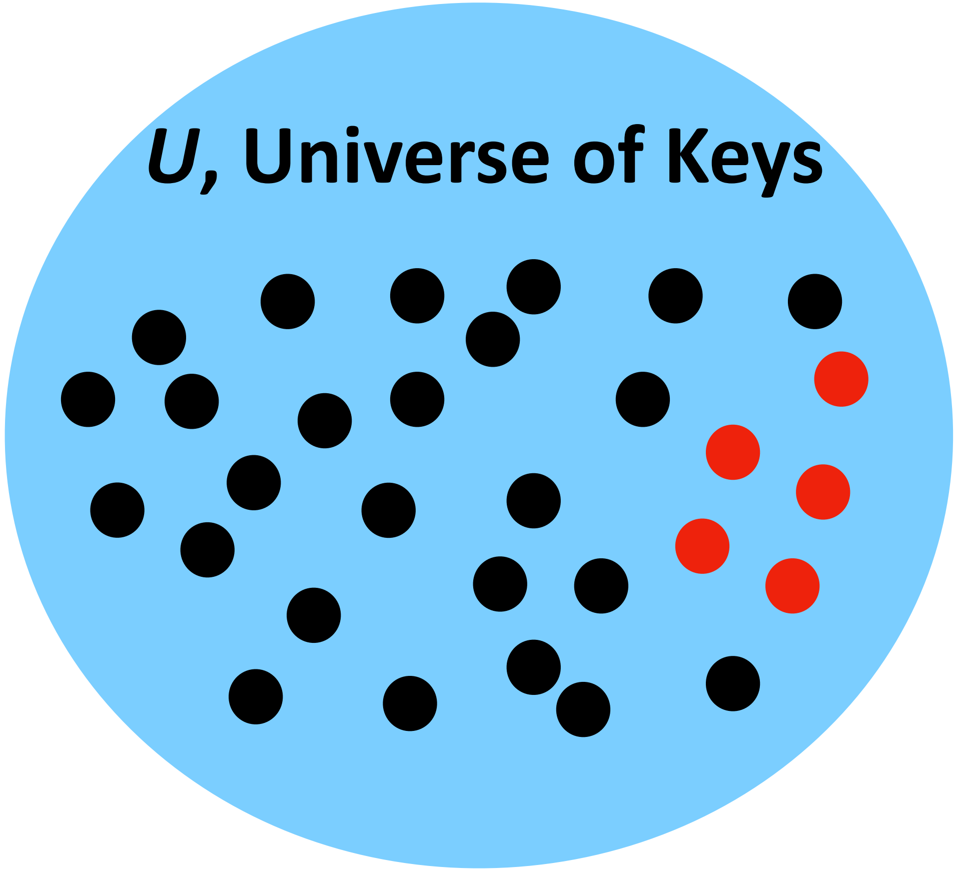
***S*, a finite Keyspace**

***m* elements**

Key	Value

General Purpose Hashing

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

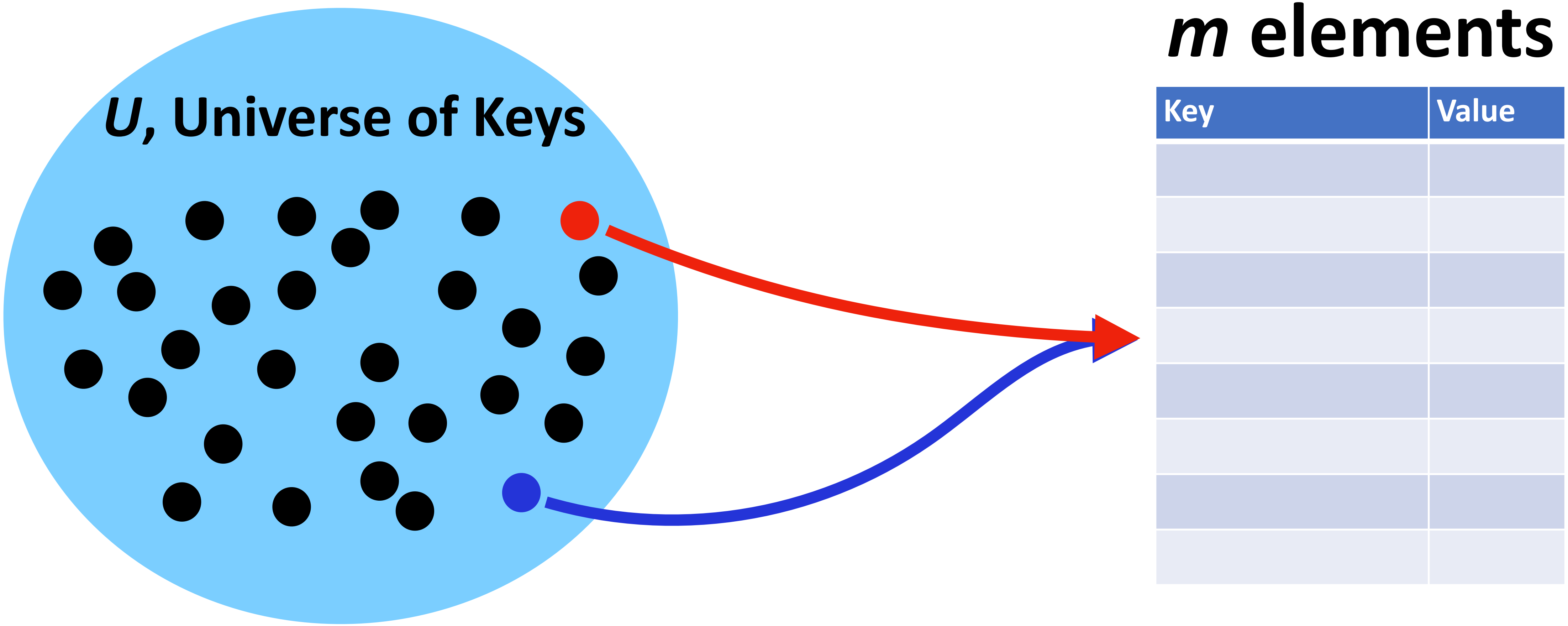


***m* elements**

Key	Value

General Purpose Hashing

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



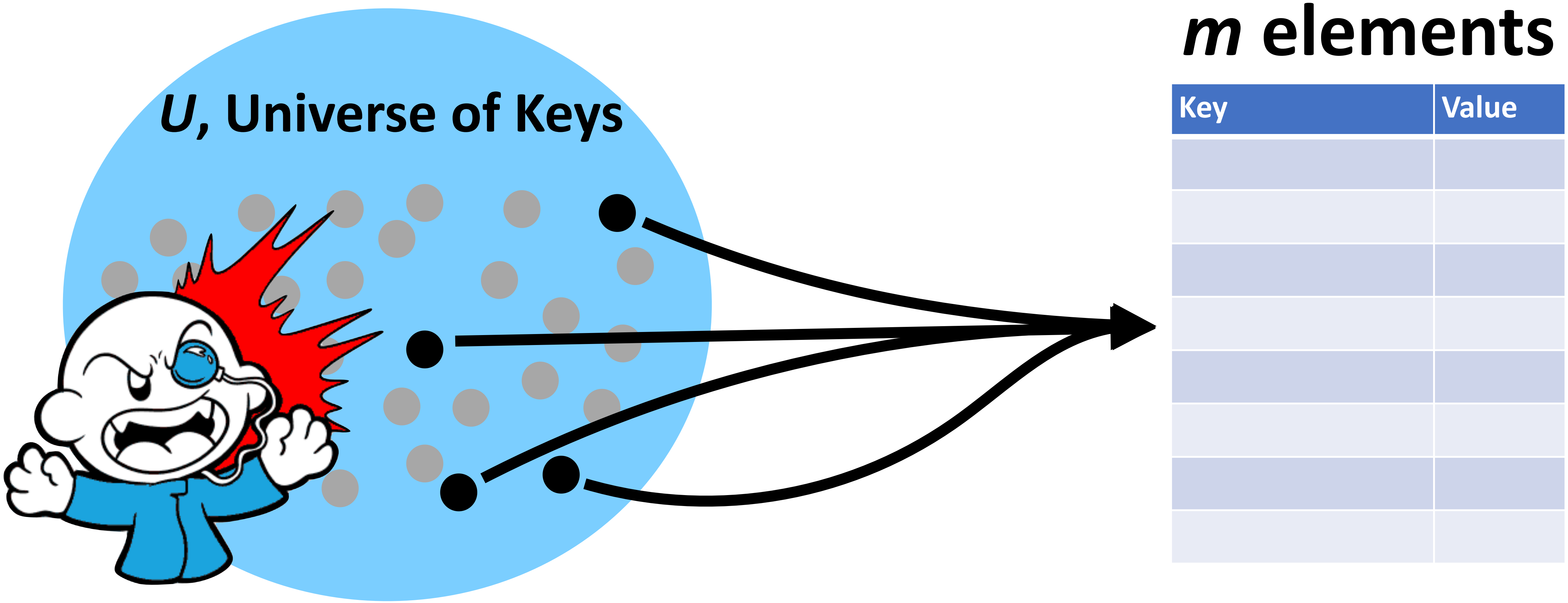
***U*, Universe of Keys**

***m* elements**

Key	Value

General Purpose Hashing

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



***U*, Universe of Keys**

***m* elements**

Key	Value

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

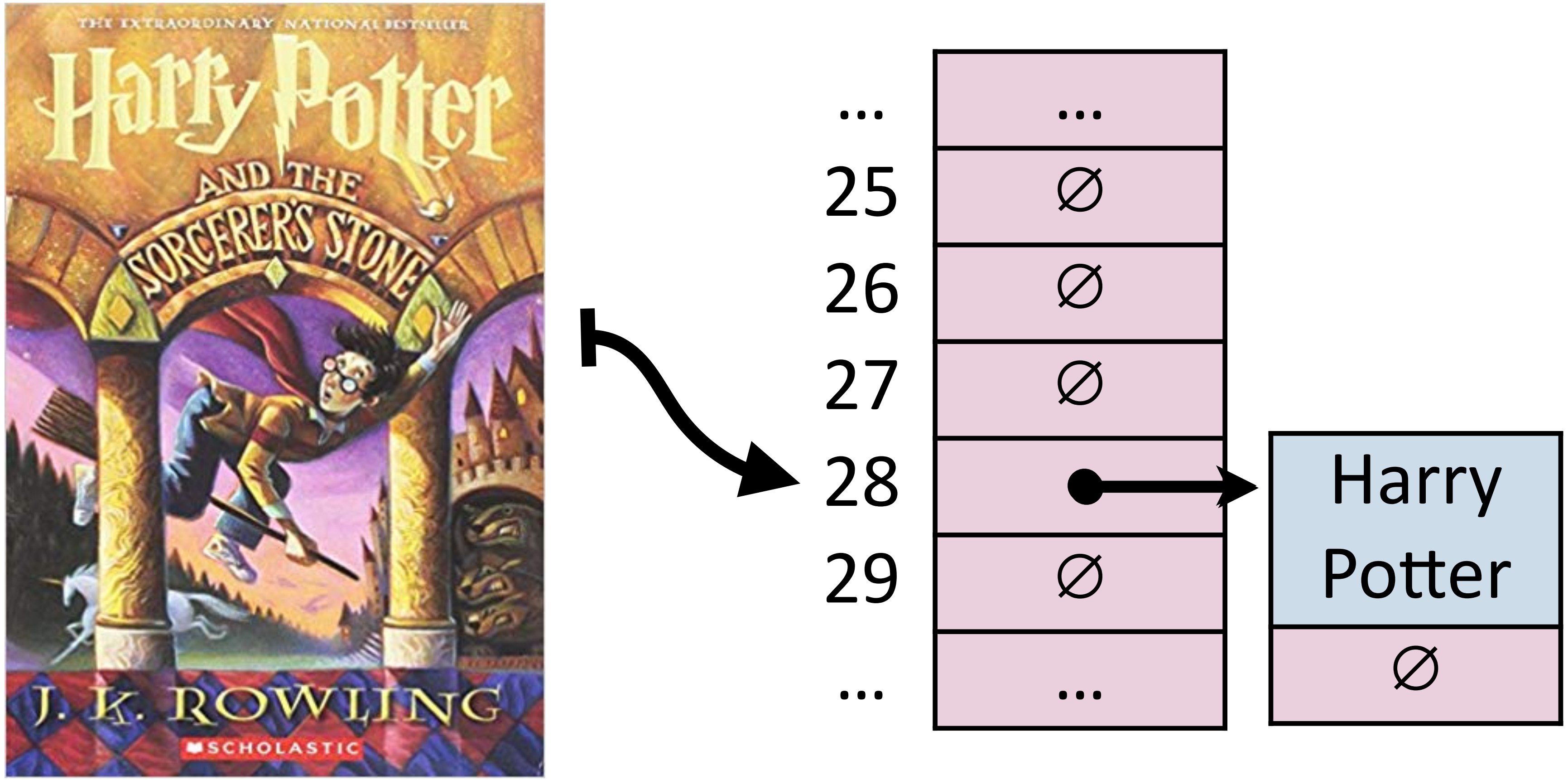
Addressing hash collisions depends on your storage structure.

- **Open Hashing:**

- **Closed Hashing:**

Open Hashing

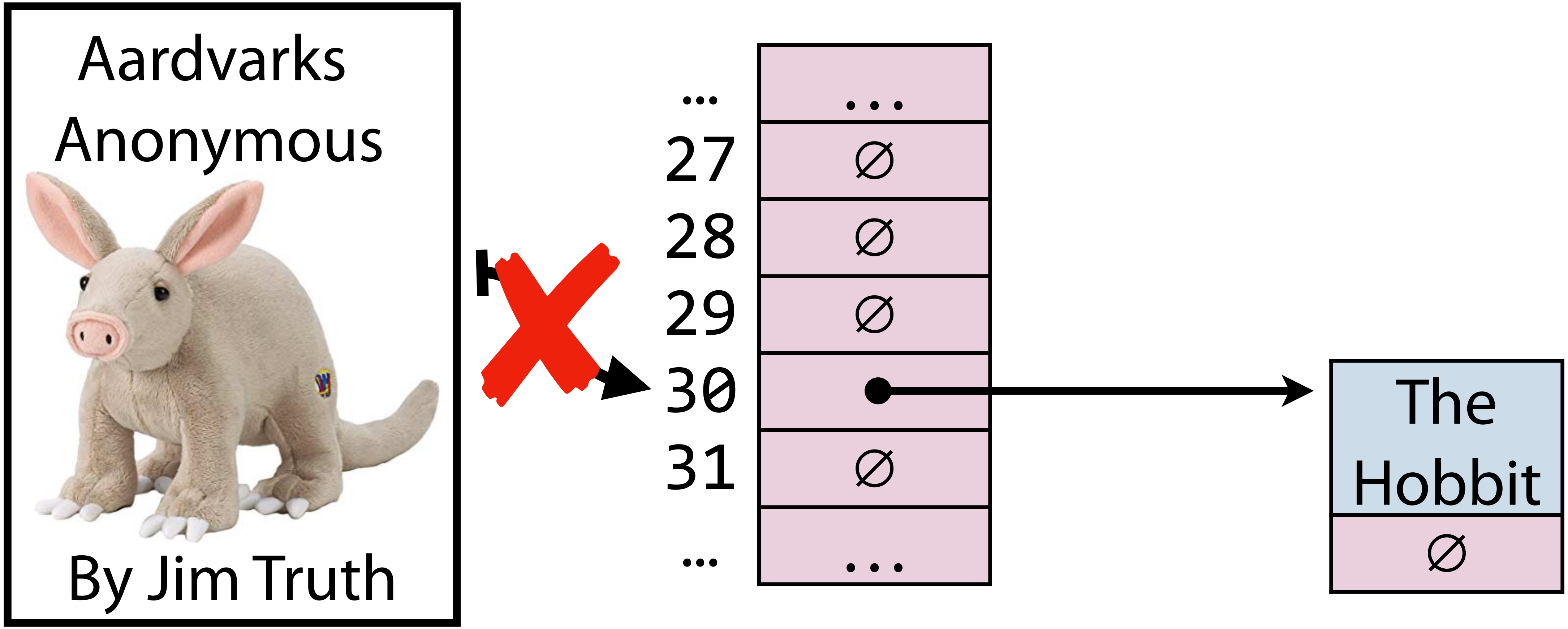
In an ***open hashing*** scheme, key-value pairs are stored externally (for example as a linked list).



…	…
25	∅
26	∅
27	∅
28	● → Harry Potter ∅
29	∅
…	…

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”)
```

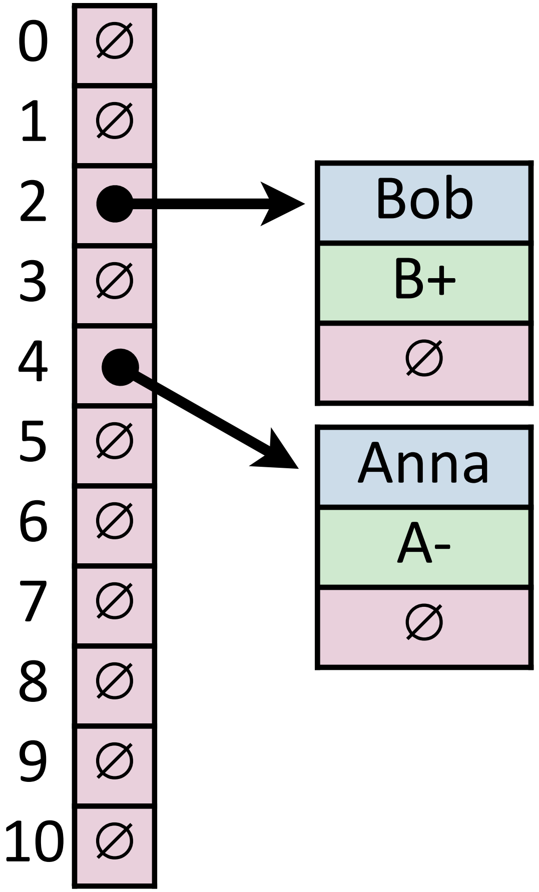
Key	Value	Hash
Bob	**B+**	**2**
Anna	**A-**	**4**
Alice	A+	4
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7

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

Insertion (Separate Chaining)

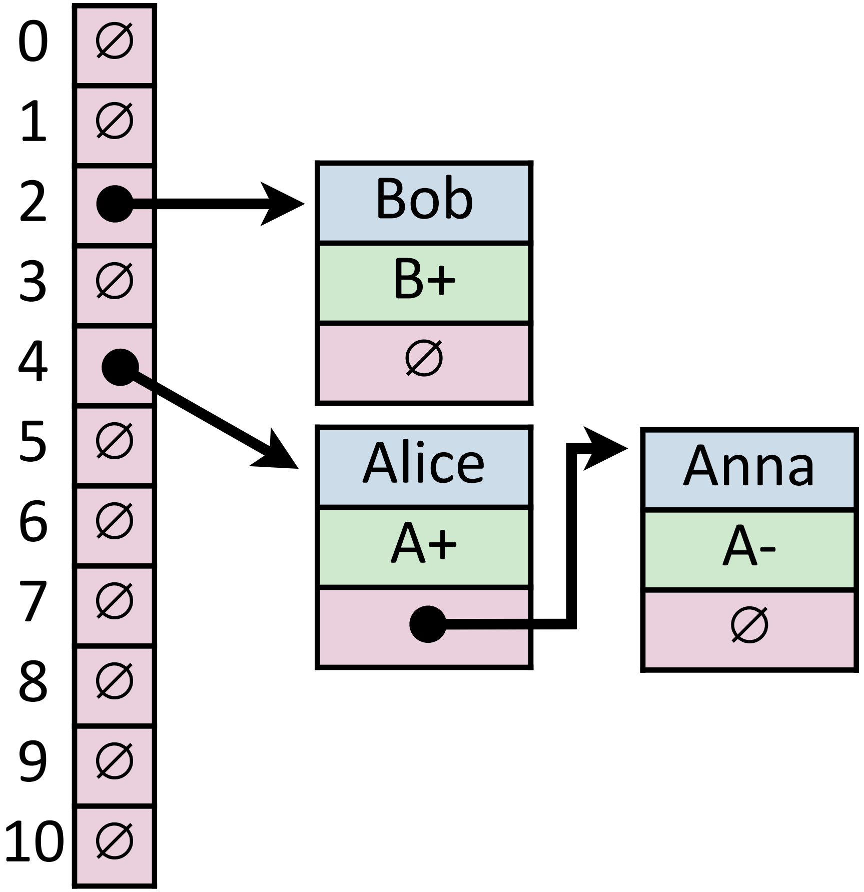
`_insert("Alice")`

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



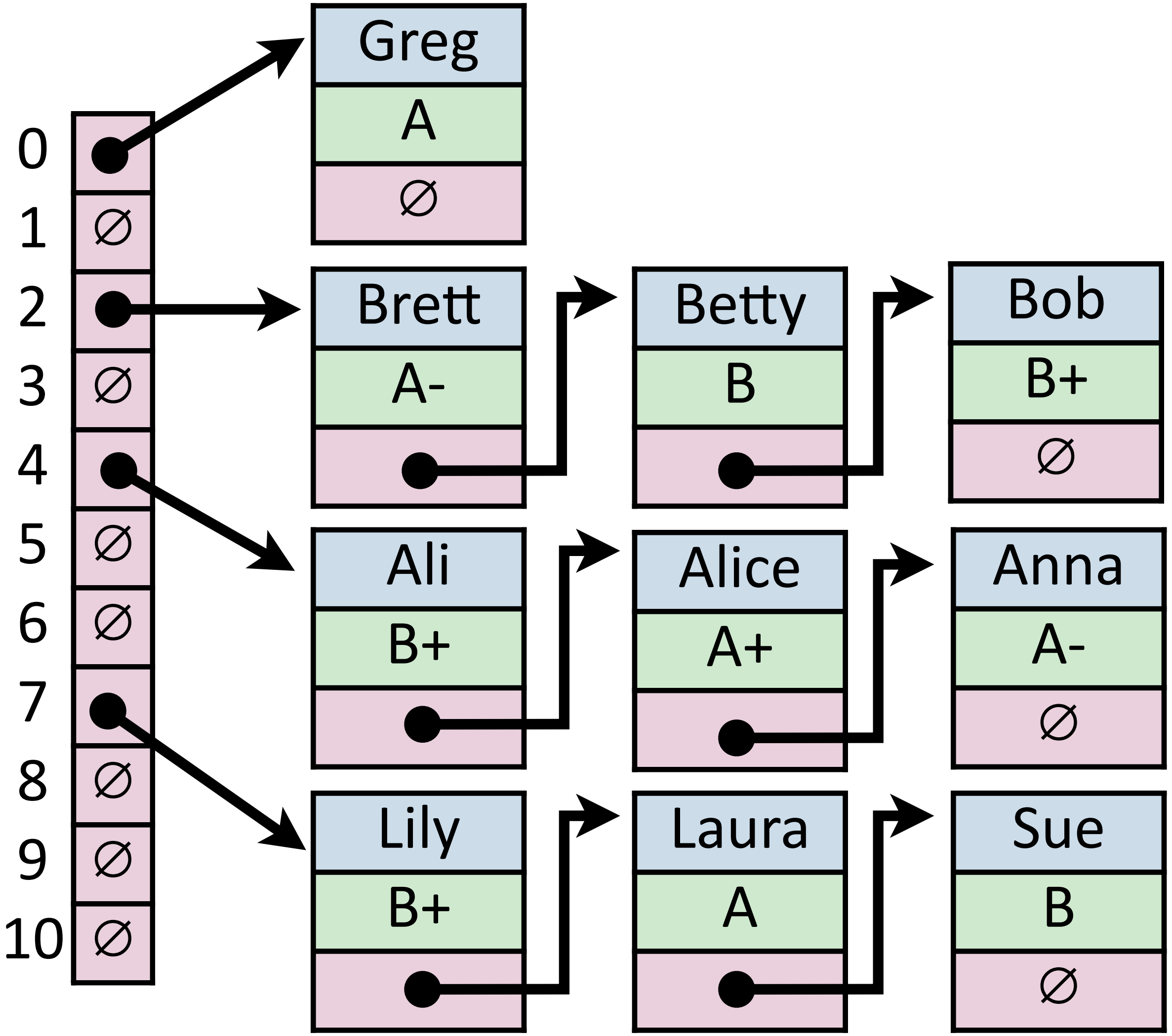
Insertion (Separate Chaining)

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)

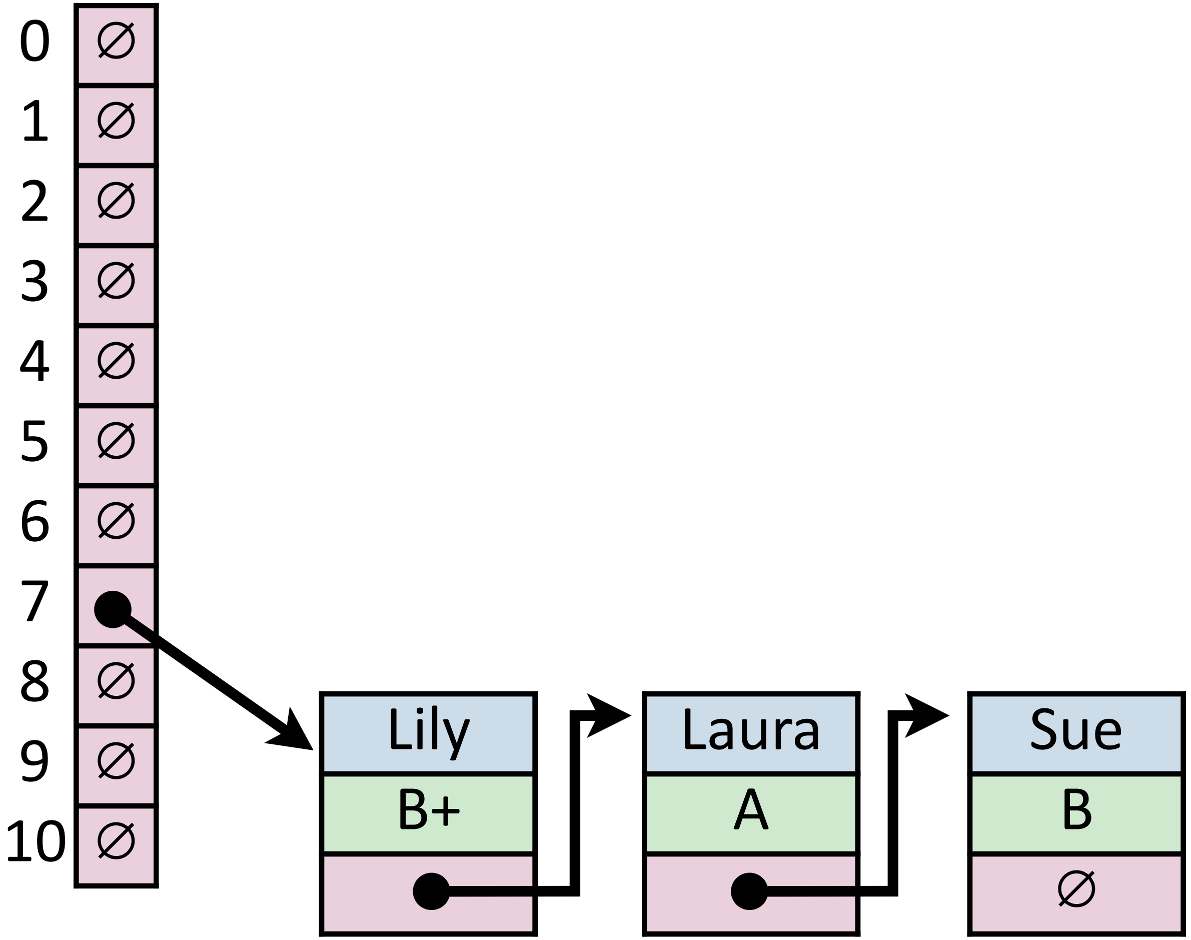
Key	Value	Hash
Bob	B+	2
Anna	A-	4
Alice	A+	4
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7



Find (Separate Chaining)

`_find(“Sue”)`

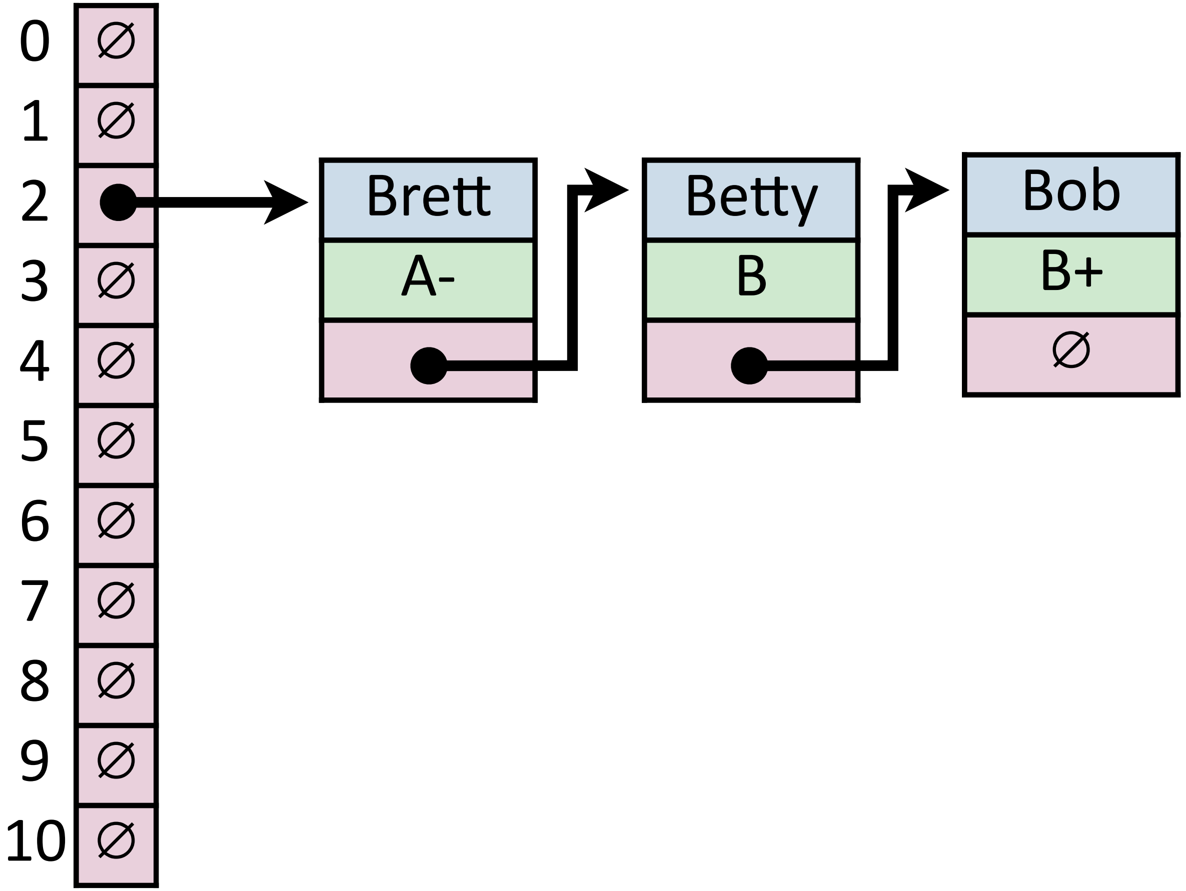
Key	Hash
Sue	7



Remove (Separate Chaining)

`_remove("Betty")`

Key	Hash
Betty	2



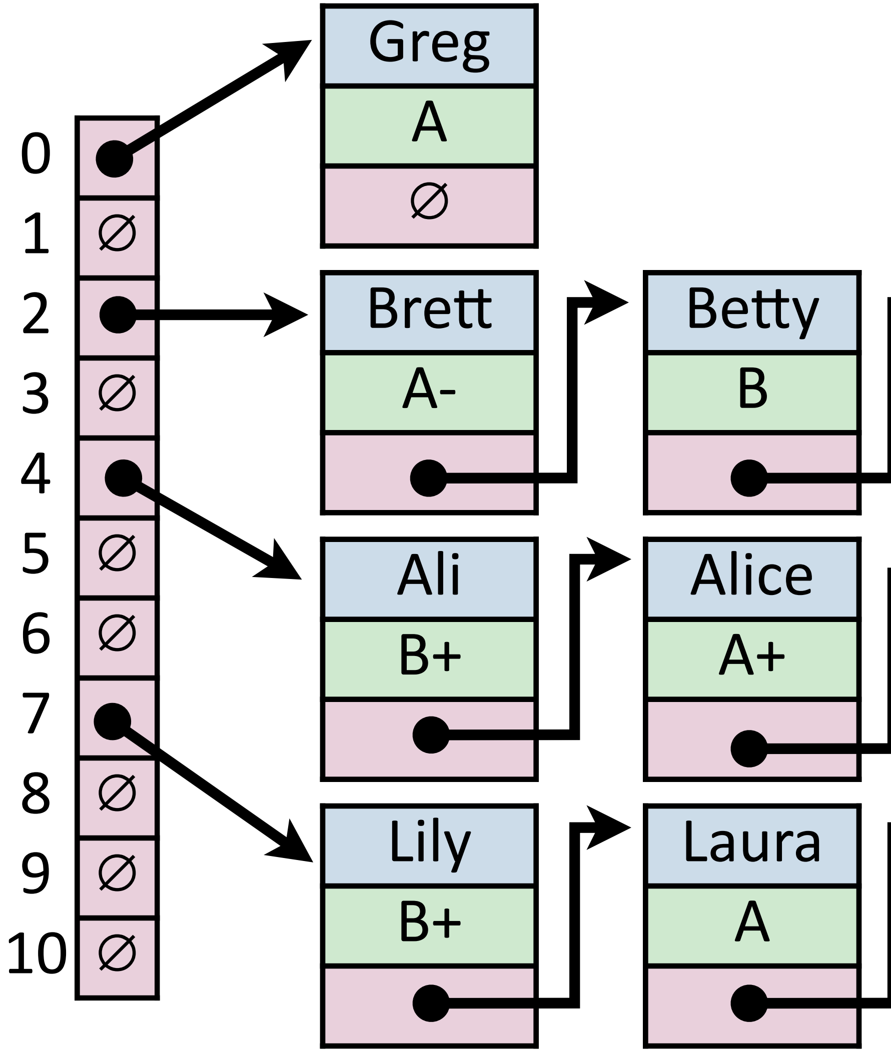
Hash Table (Separate Chaining)

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

Find runs in: ____________________

Insert runs in: ____________________

Remove runs in: ____________________



Hash Table

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

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

Uniform:

Independent:

Separate Chaining Under SUHA

Given table of size m and n inserted objects

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

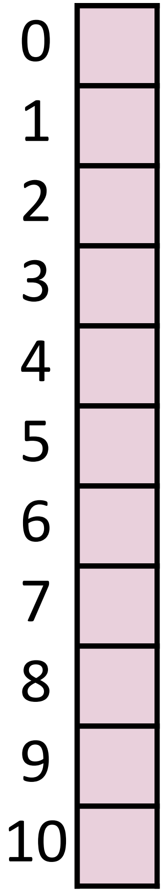
Separate Chaining Under SUHA

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