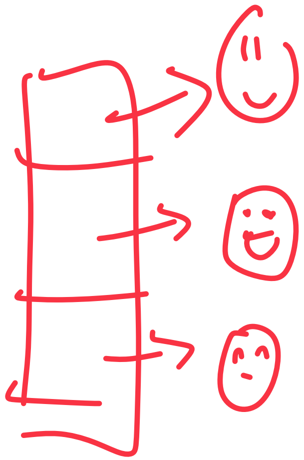
Data Structures and Algorithms
Hashing

CS 225
Brad Solomon

November 11, 2024



Department of Computer Science

Randomization in Algorithms

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

   ↳ BST avg path length (weakest argument)

2. Use randomness inside algorithm to estimate expected running time

   ↳ Randomized quicksort

   ↳ 100% correctness

   ↳ Randomness is runtime

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

   ↳ Fermat's Primality Test

   ↳ Might not be correct

   O(1) time

   Randomness is always

Learning Objectives

Motivate and formally define a hash table

my favorite core D.S.

Discuss what a 'good' hash function looks like

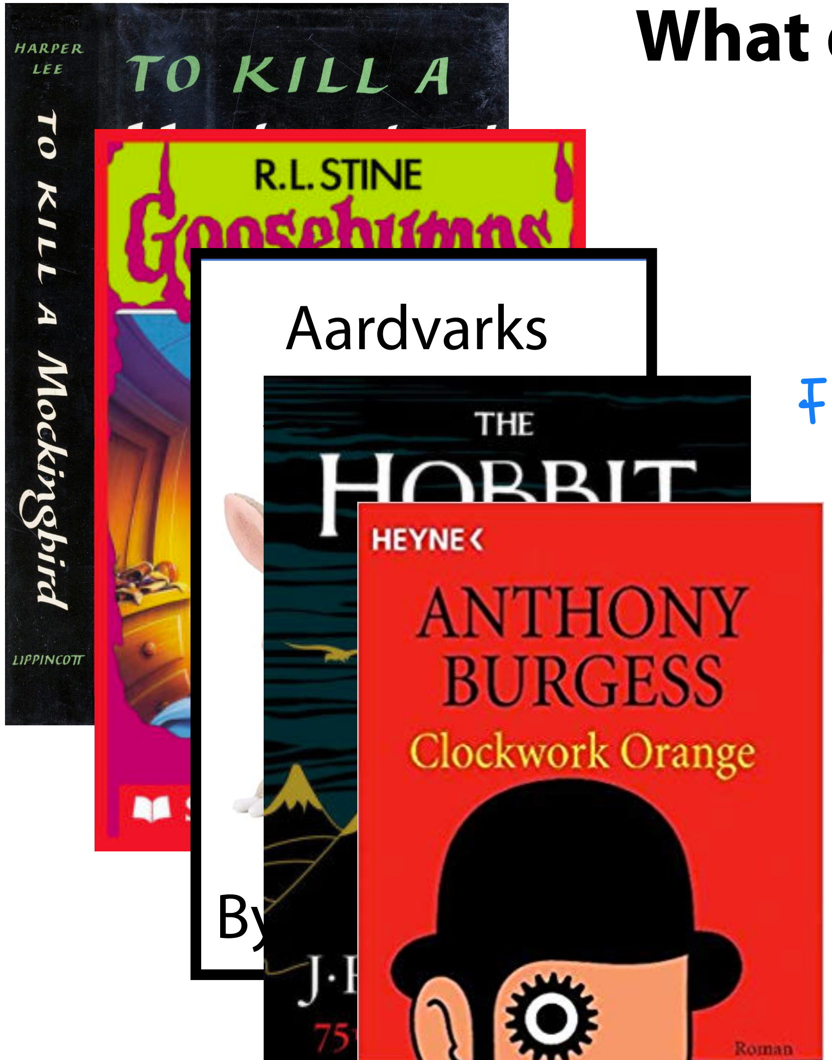
Identify the key weakness of a hash table

Introduce strategies to "correct" this weakness

Data Structure Review

Key: title
Value: text

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

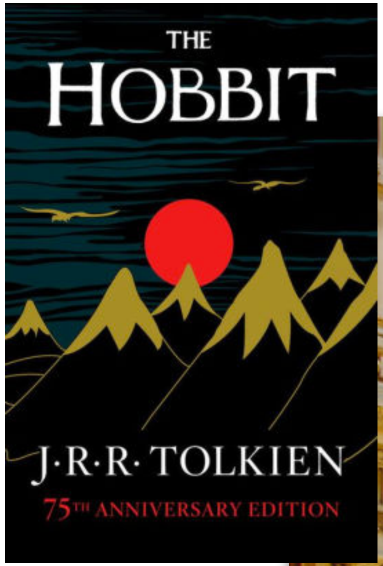


What data structures can I use here?

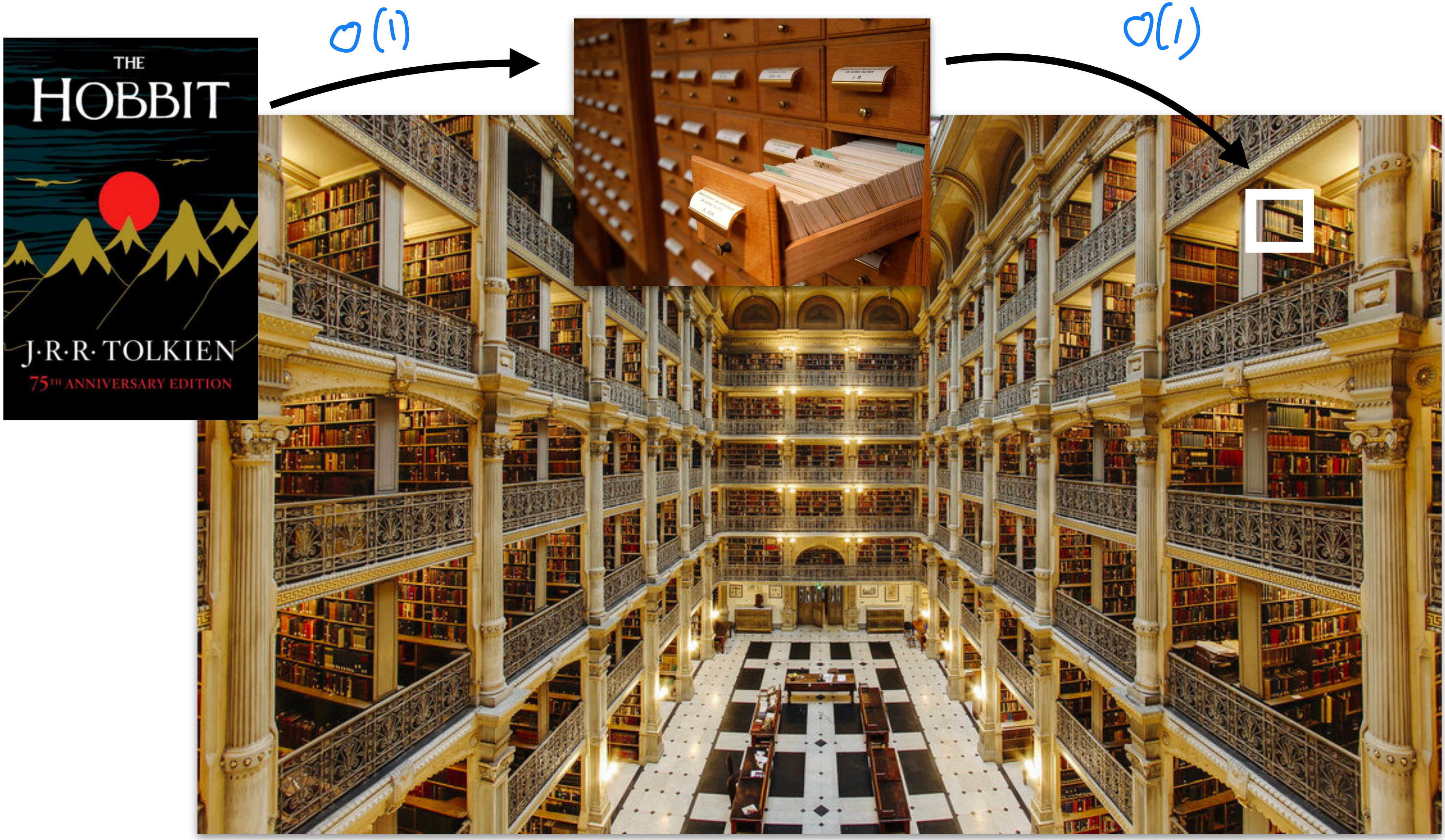
	Sorted Array (List)	AVL	Unsorted List
Find	log(n)	log n	O(n)
Insert	O(log n) or O(n)	log n	O(1)
Remove	(same as Insert)	log n	O(n)

No resize → O(log n); resize → O(n)

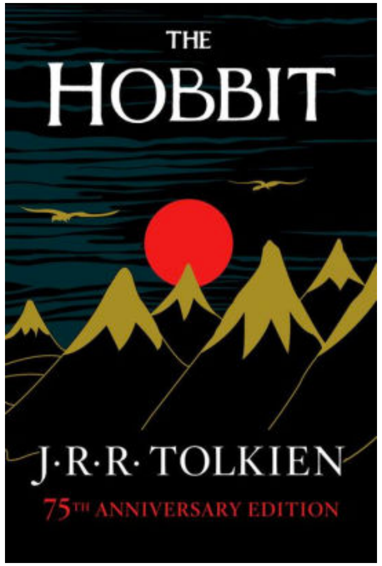
What if $O(log\ n)$ isn't good enough?



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



A Hash Table based Dictionary



O(1)

"Address"

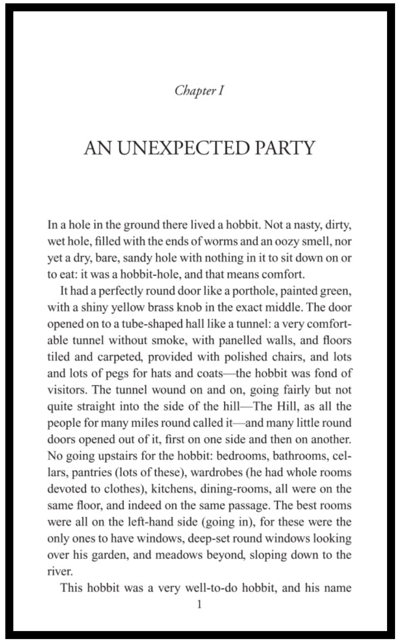
ISBN: 9780062265722

Call #: PR
6068.O93
H35 1937

O(1)

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**

AVL tree (sorted DS) log(n) insert

Randomized data structures rely on **expected** performance

Randomized data structures 'cheat' tradeoffs!

A Hash Table based Dictionary

User Code (is a map):

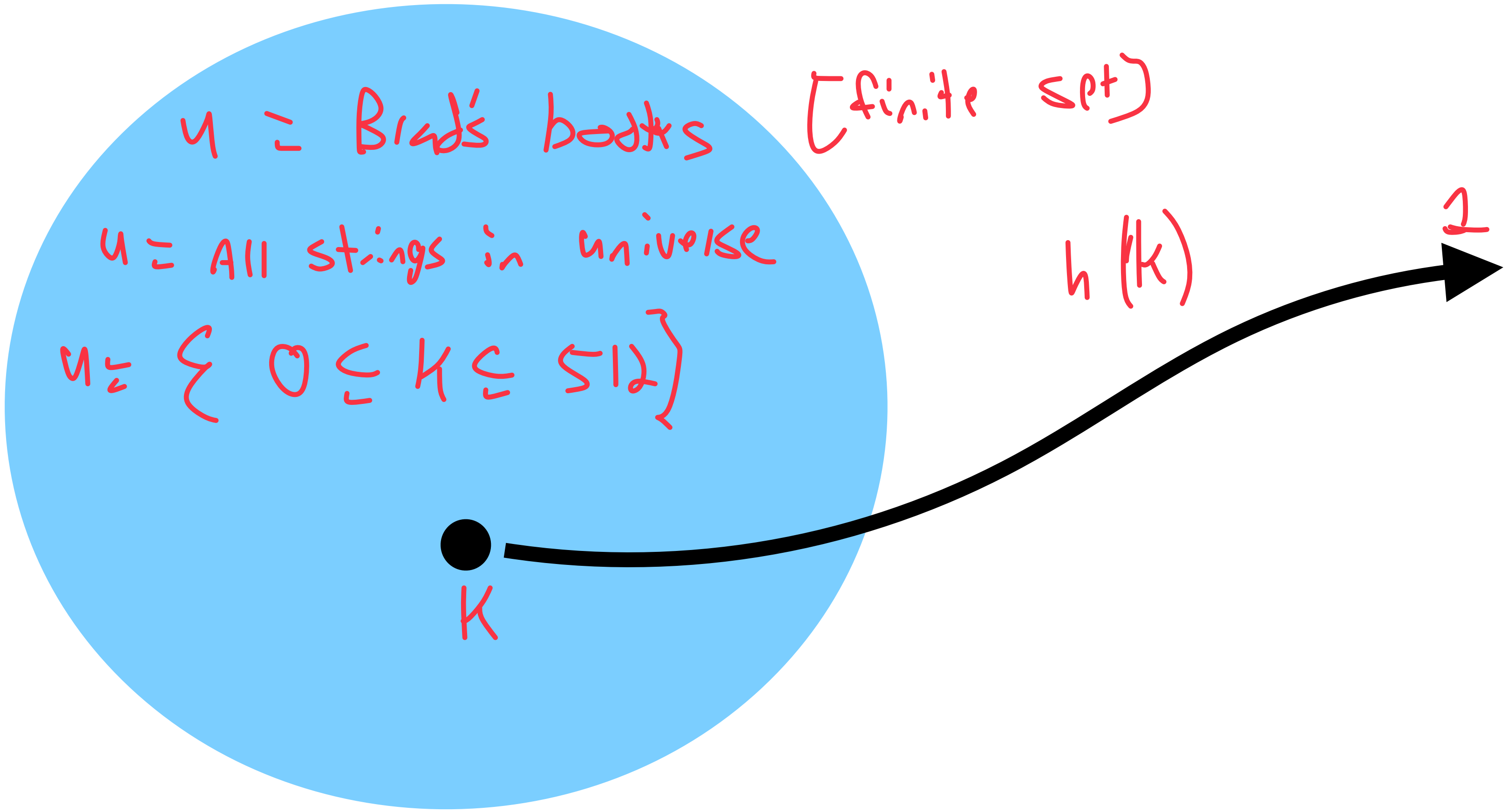
```
Dictionary<KeyType, ValueType> d;
d[k] = v;
```

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

1. A hash function (key → int "address")
   input
2. A data storage structure (Array)
3. ?? Handle chaos

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

Output of hash
hash value
⇕

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

- **Deterministic:** Given same key, hash will always return same address
- **Efficient:** Goal is O(1), then hash must be O(1)
- **Defined for a certain size table:** $h(k)$: Universe → $[0, \ldots m-1]$

↑ Array of size m

Hash Function

Key Value

(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'

	Key	Value
0	Angrave	241
1	Beckman	421

Hash Function

Good

Perfect bijection / Perfect hash for this exact dataset

0 (Angrave, CS 241) Alawini

1 (Beckman, CS 421)

2 (Challon, CS 125)

3 (Davis, CS 101)

4 (Evans, CS 225)

5 (Fagen-Ulmschneider, CS 107)

6 (Gunter, CS 422)

7 (Herman, CS 233)

Solomon 225

collision!

O(1) hash

Hash function

(key[0] - 'A')

h(Name)

Bad

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

Solomon 225 ?

No room!

Not all letters equally likely!

General Hash Function

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

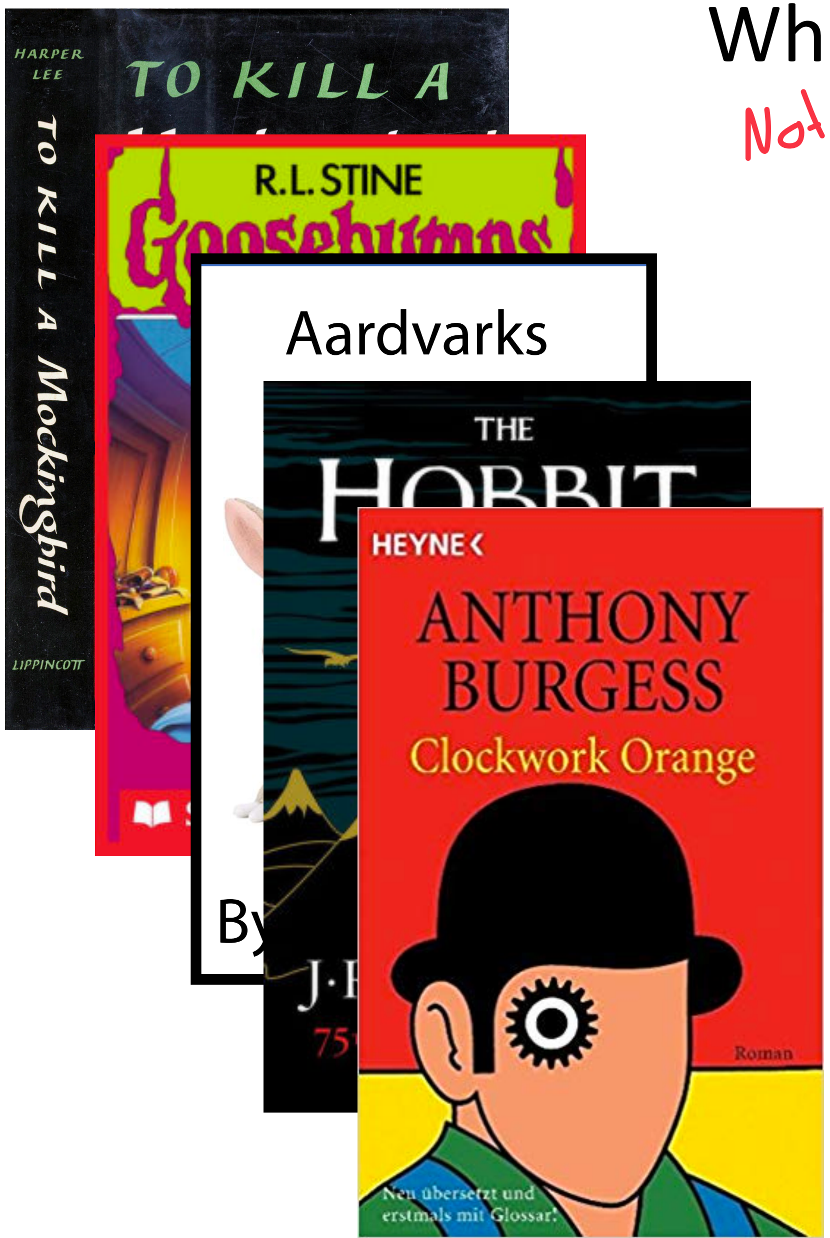
- A **hash**:
- A **compression**:

Choosing a good hash function is tricky…

- Don't create your own (yet*)

Hash Function

Collisions are Ok!



Which of the following are valid hashes?

Not a good hash function

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

↳ Is a hash function

O(1) = Yes

u → [0, ... m-1] = Yes

Deterministic = Yes

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

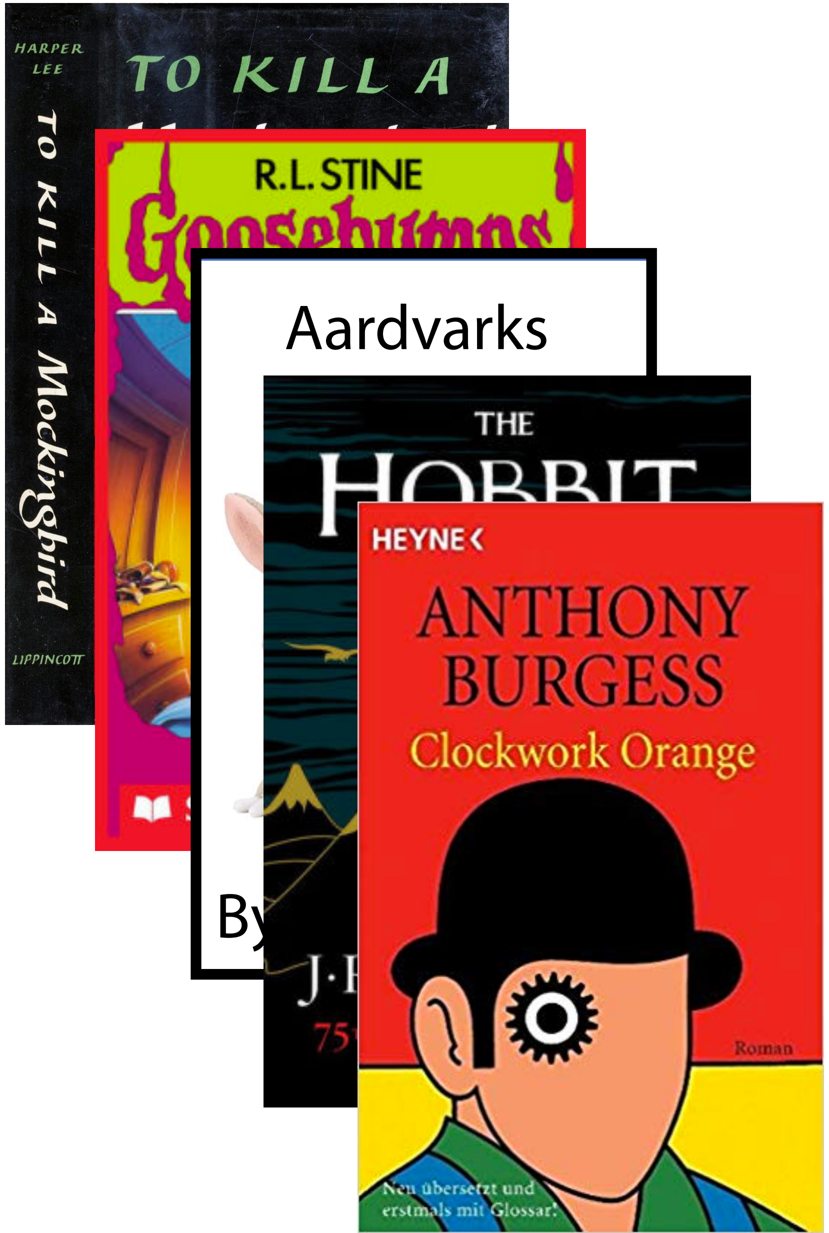
↑ Not deterministic

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

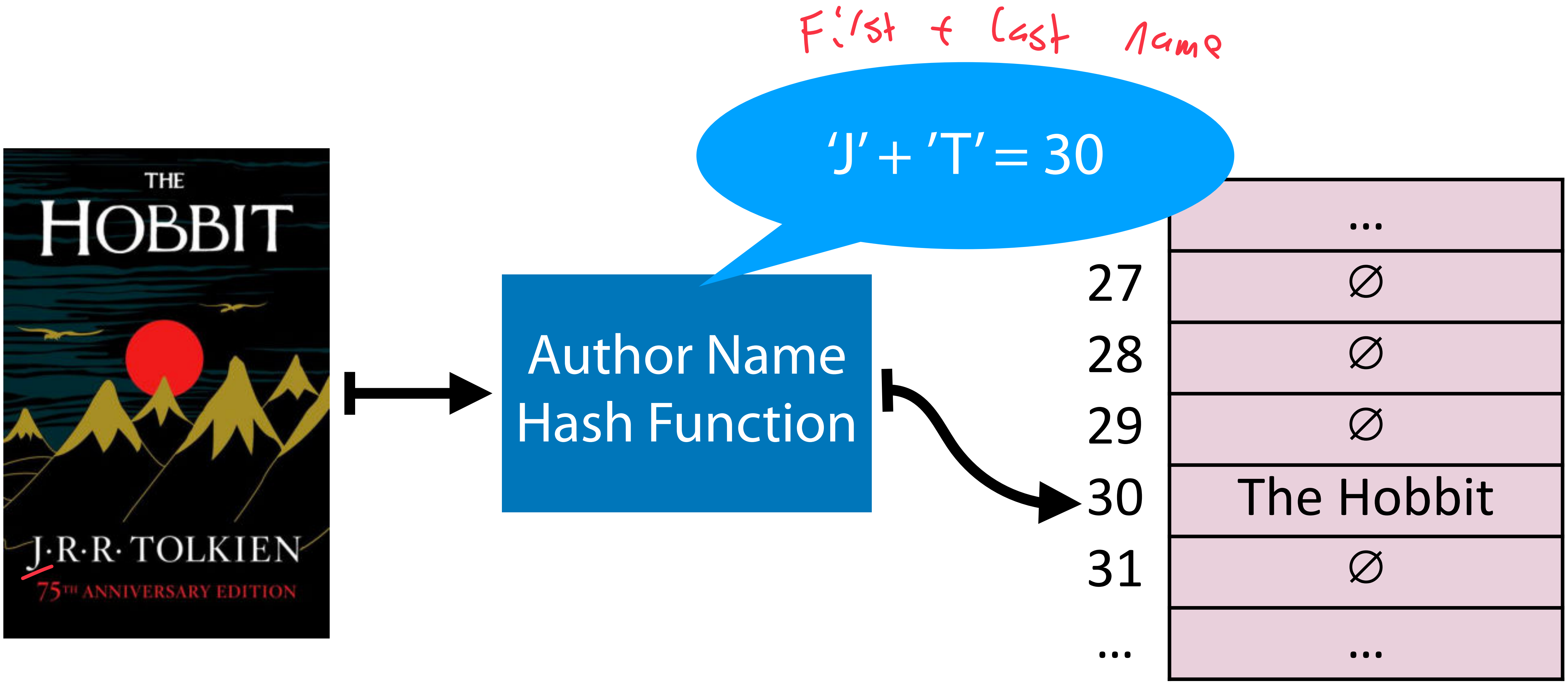
What if same book twice?

Not deterministic b/c input order matters

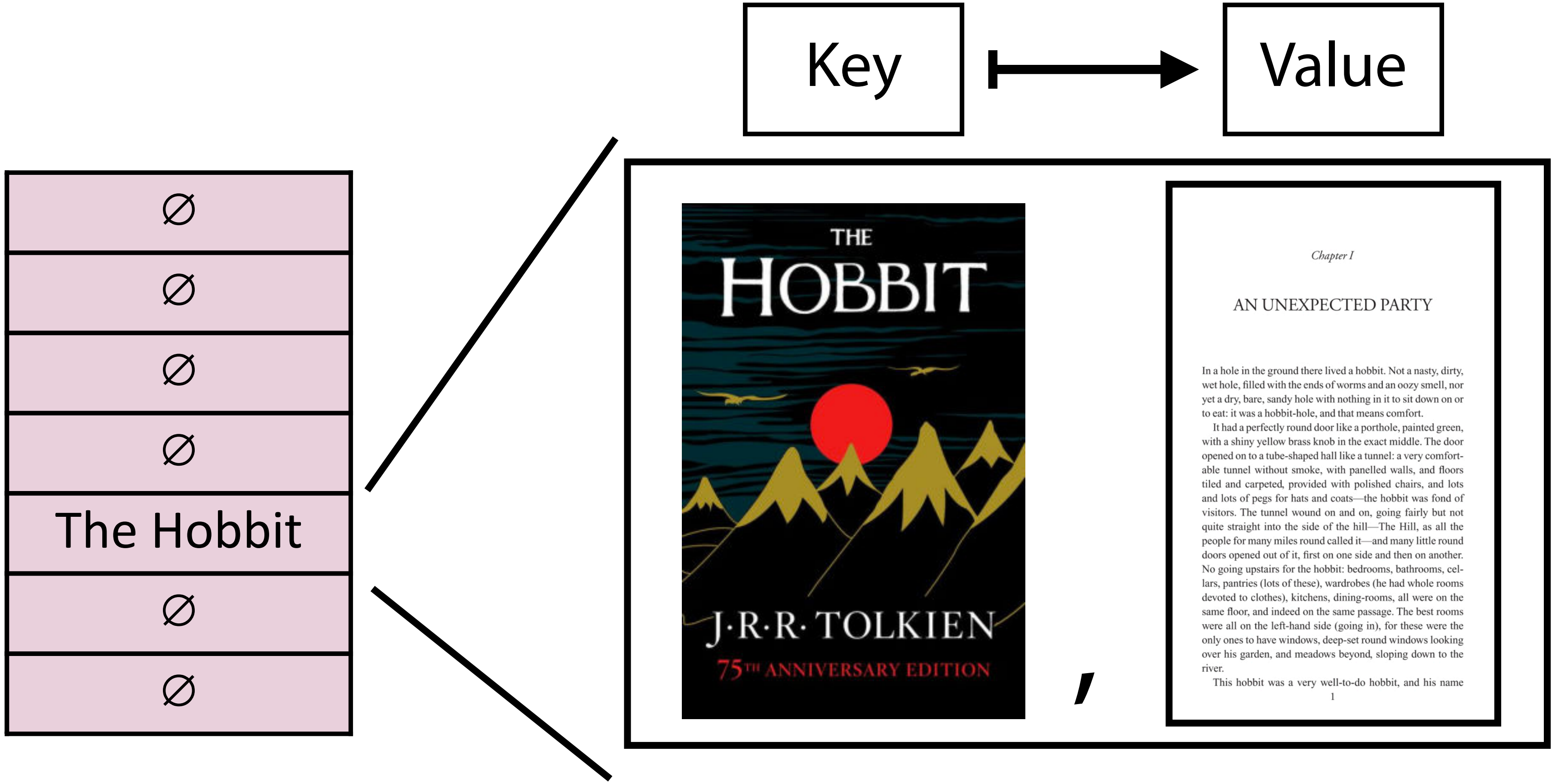
Hash Function



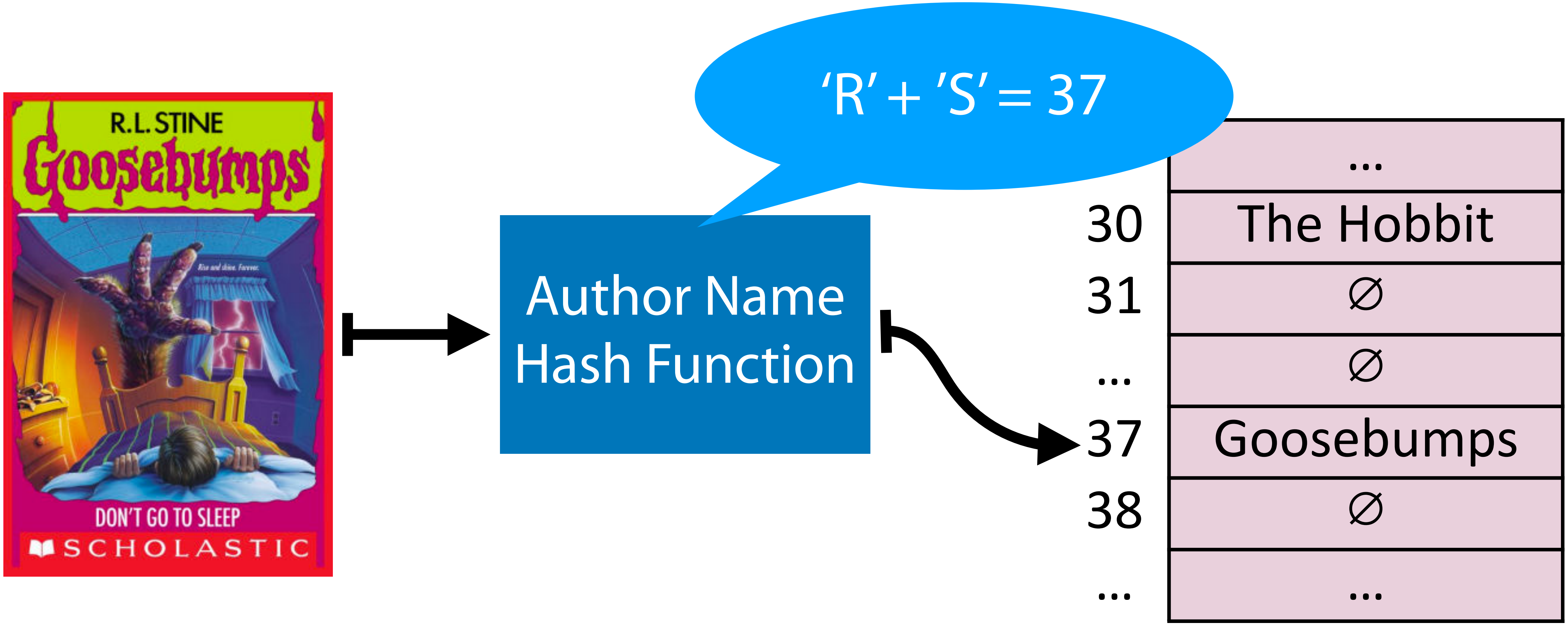
A Hash Table based Dictionary



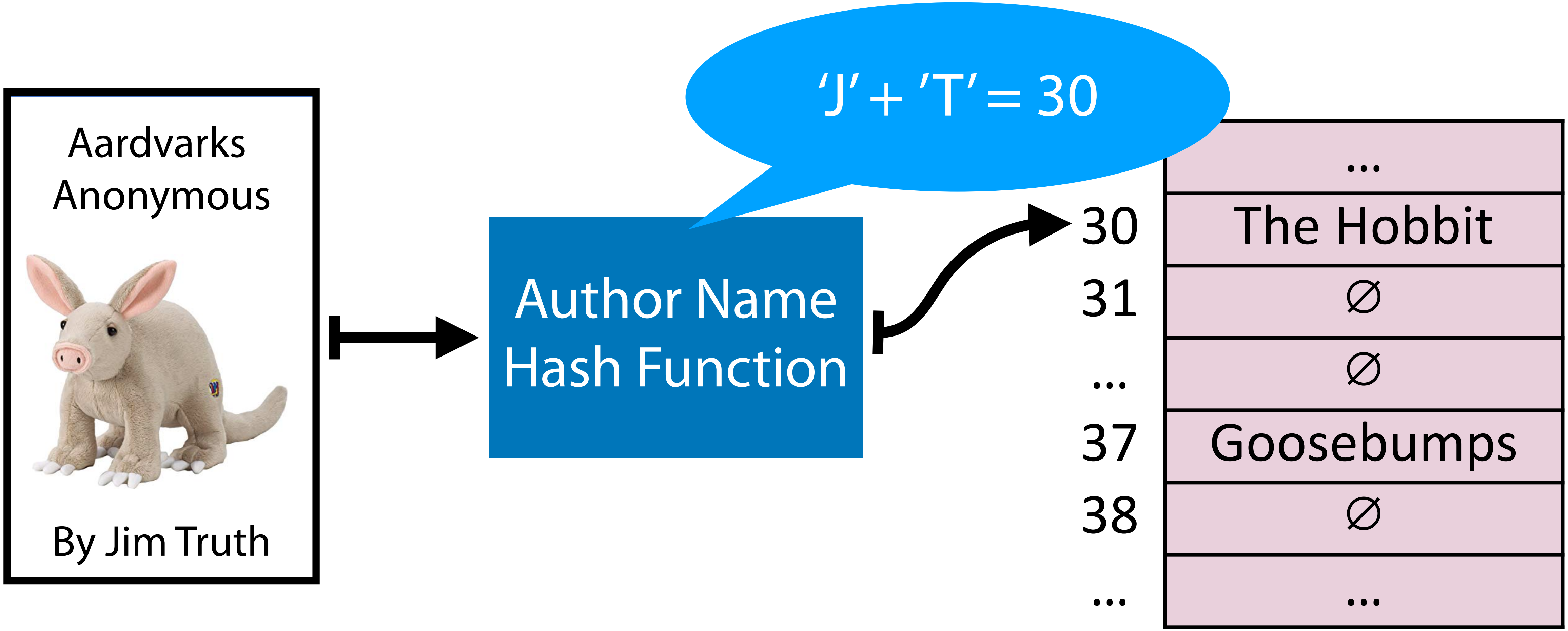
A Hash Table based Dictionary



A Hash Table based Dictionary

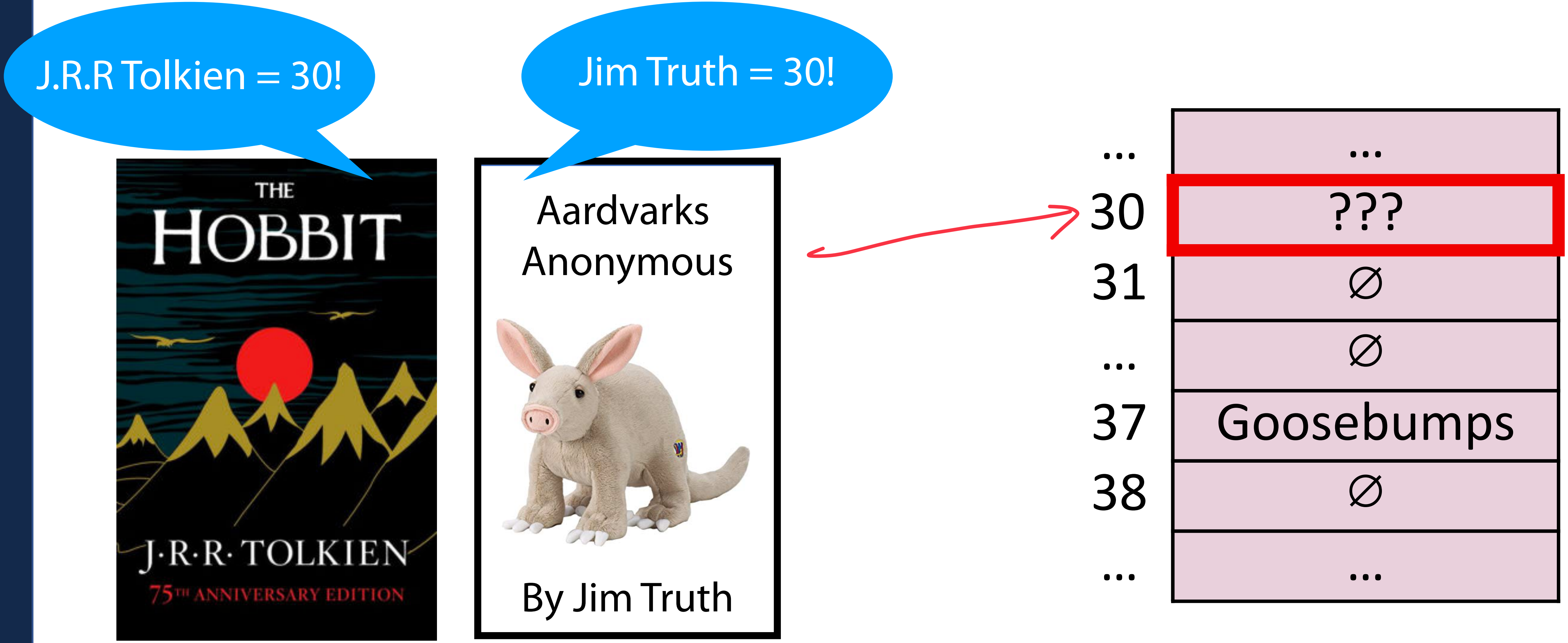


A Hash Table based Dictionary



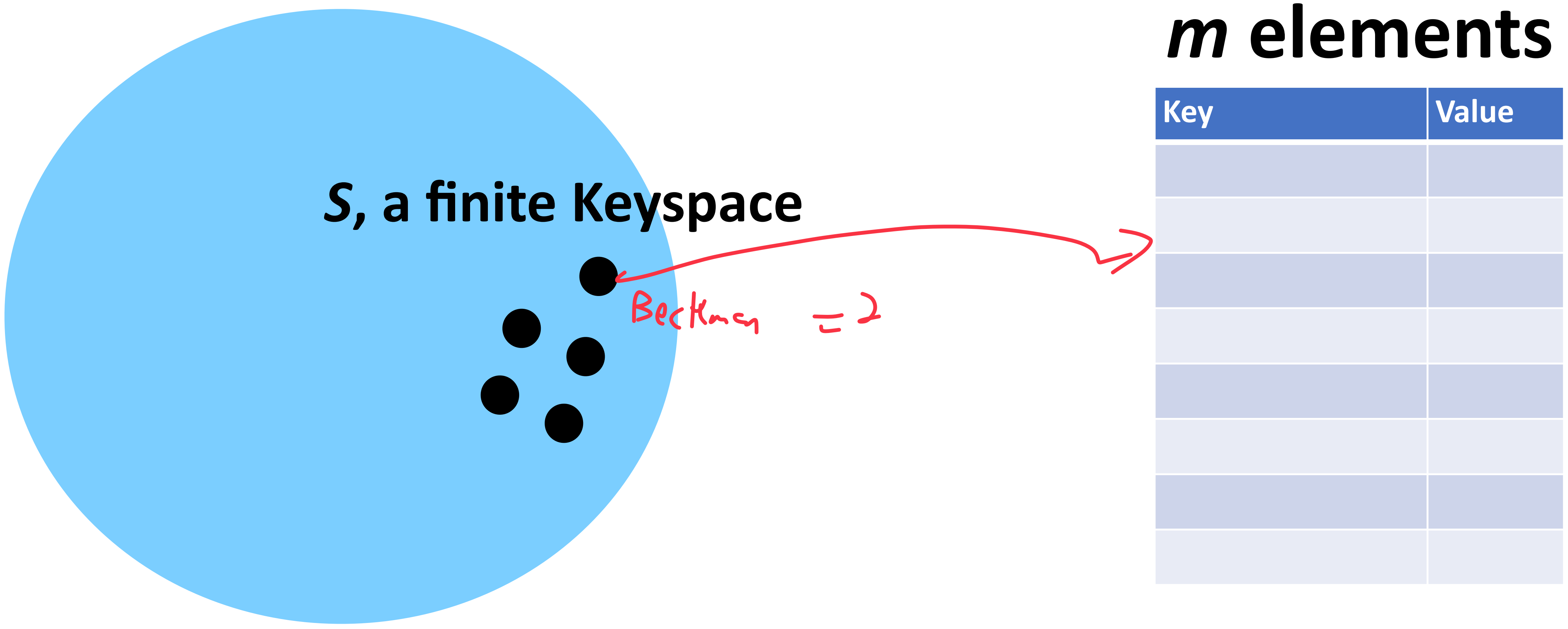
Hash Collision

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



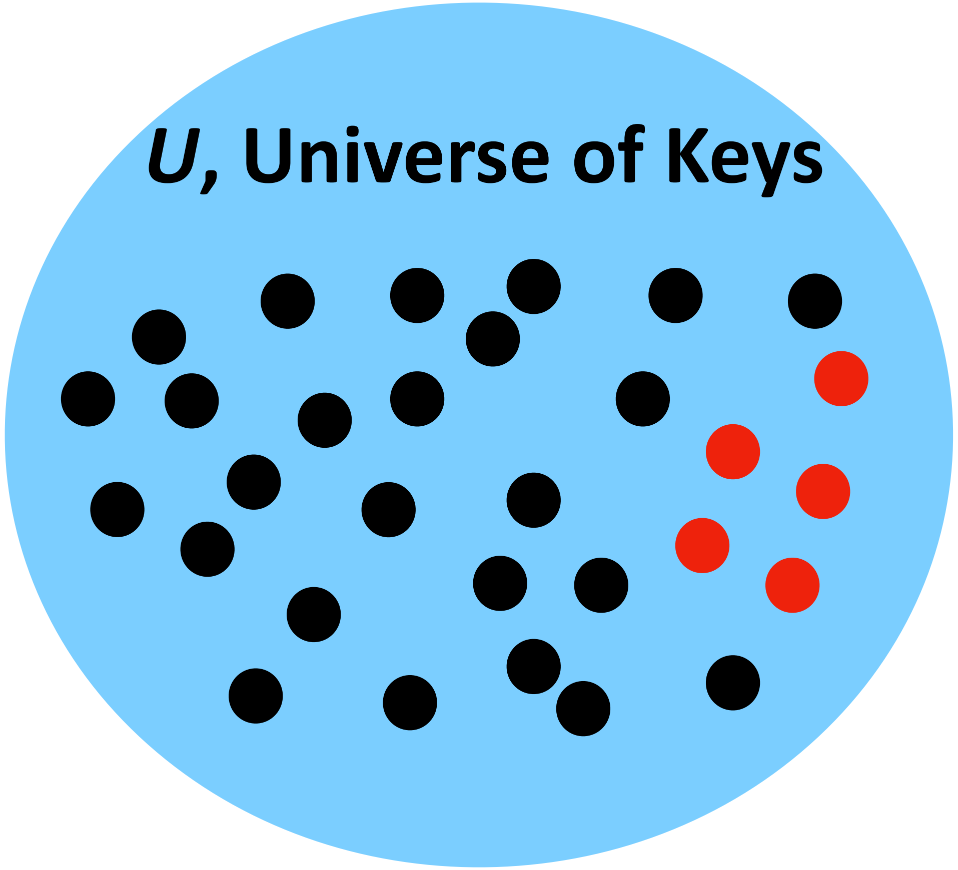
Perfect Hashing

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



General Purpose Hashing

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

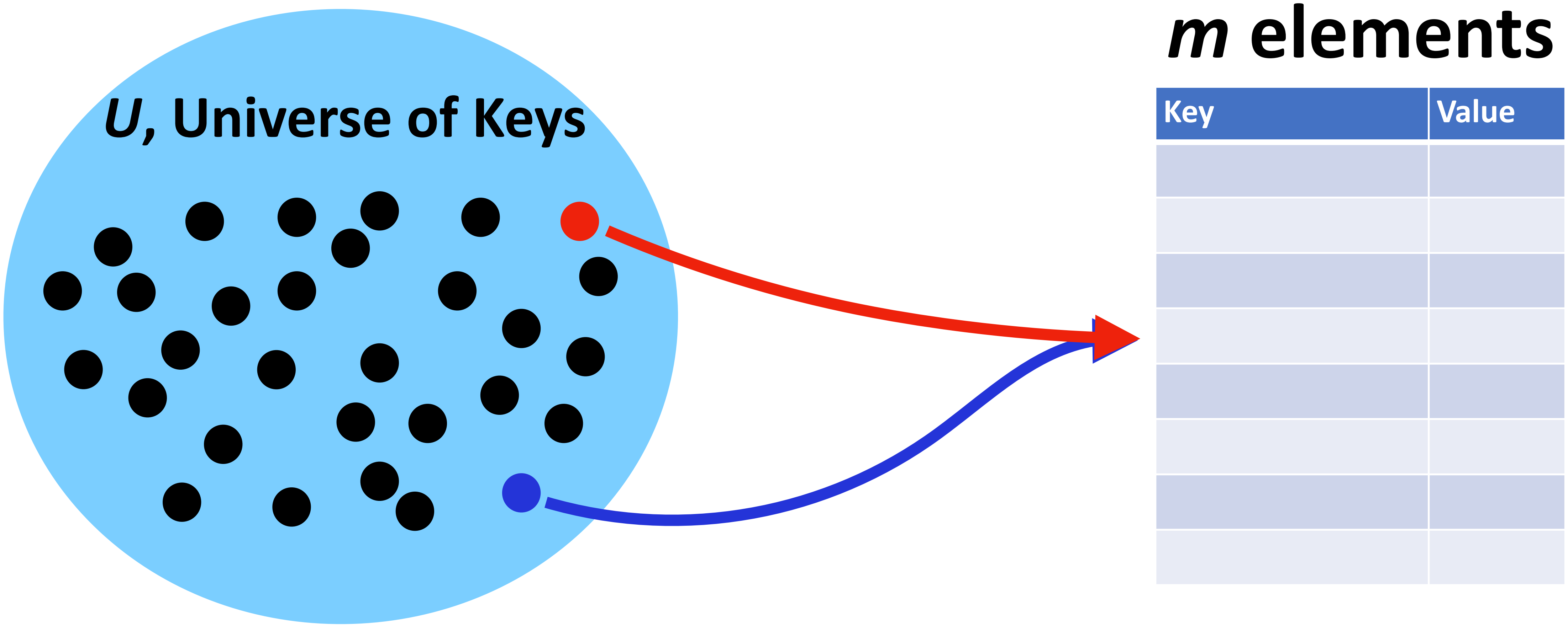


m elements

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.

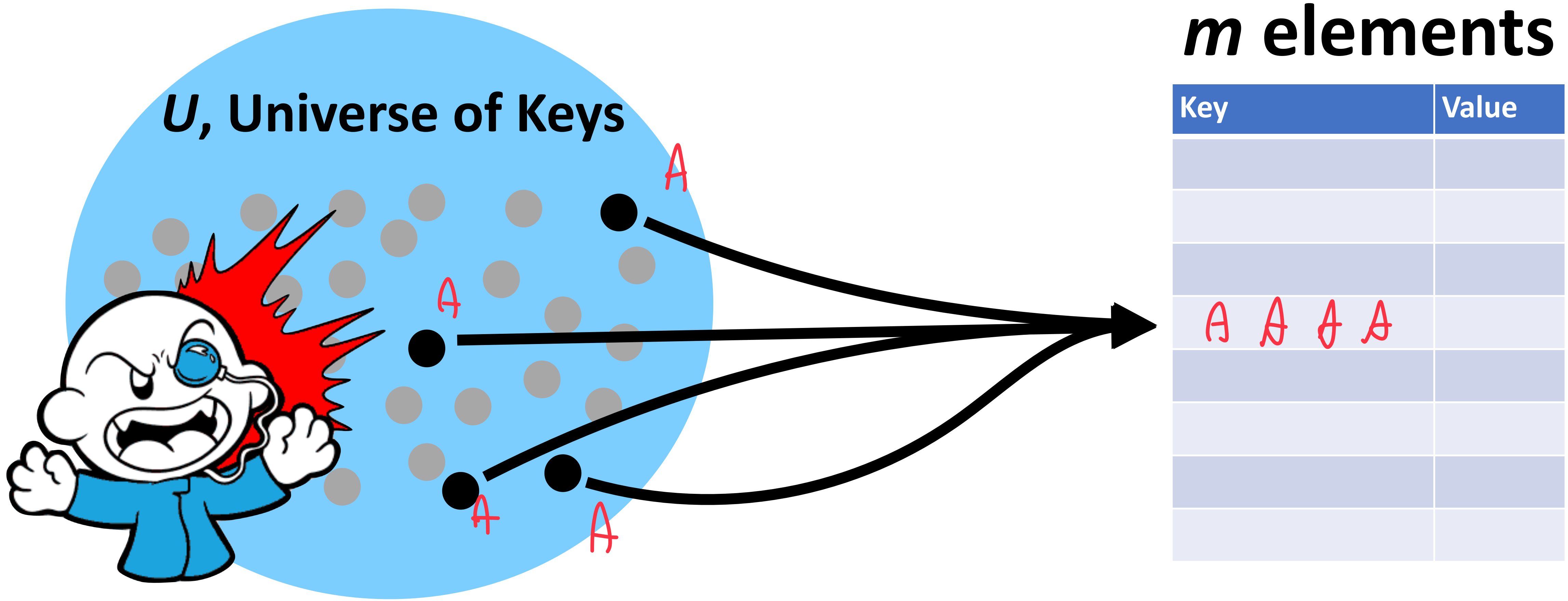


Image by Matthew Loffhagen

A Hash Table based Dictionary

User Code (is a map):

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

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

1. A hash function

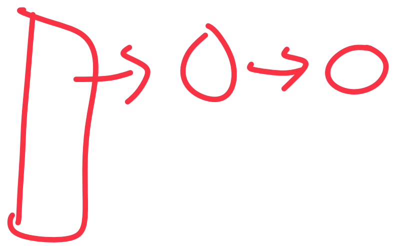
2. A data storage structure

3. A method of addressing *hash collisions*

Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

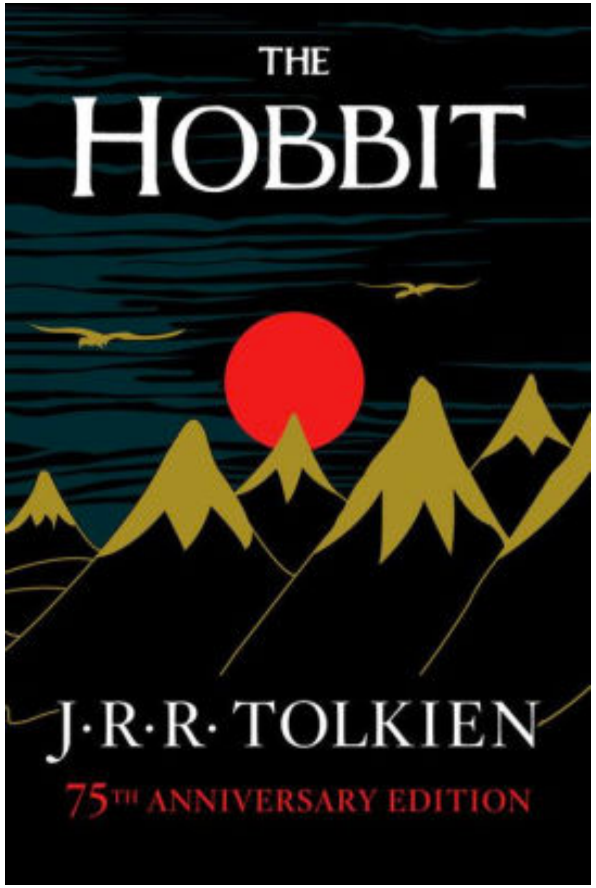
- **Open Hashing:** Store (k, v) pairs externally

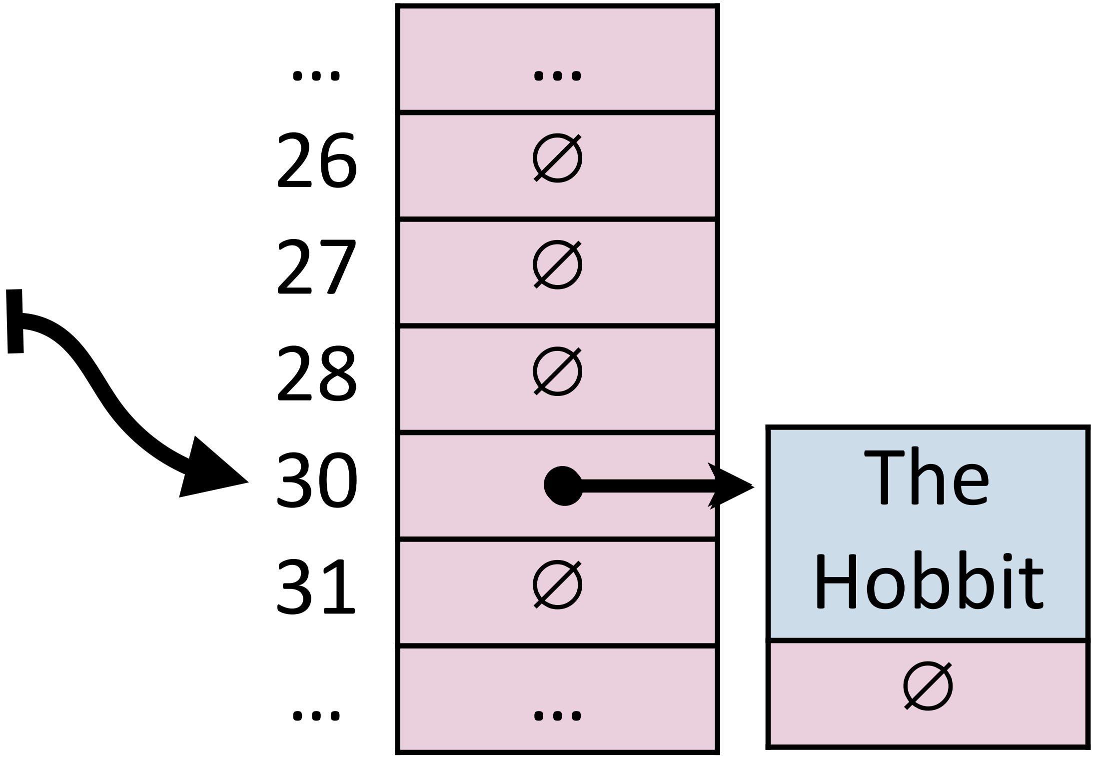


- **Closed Hashing:** store (k, v) pairs in hash table

Open Hashing

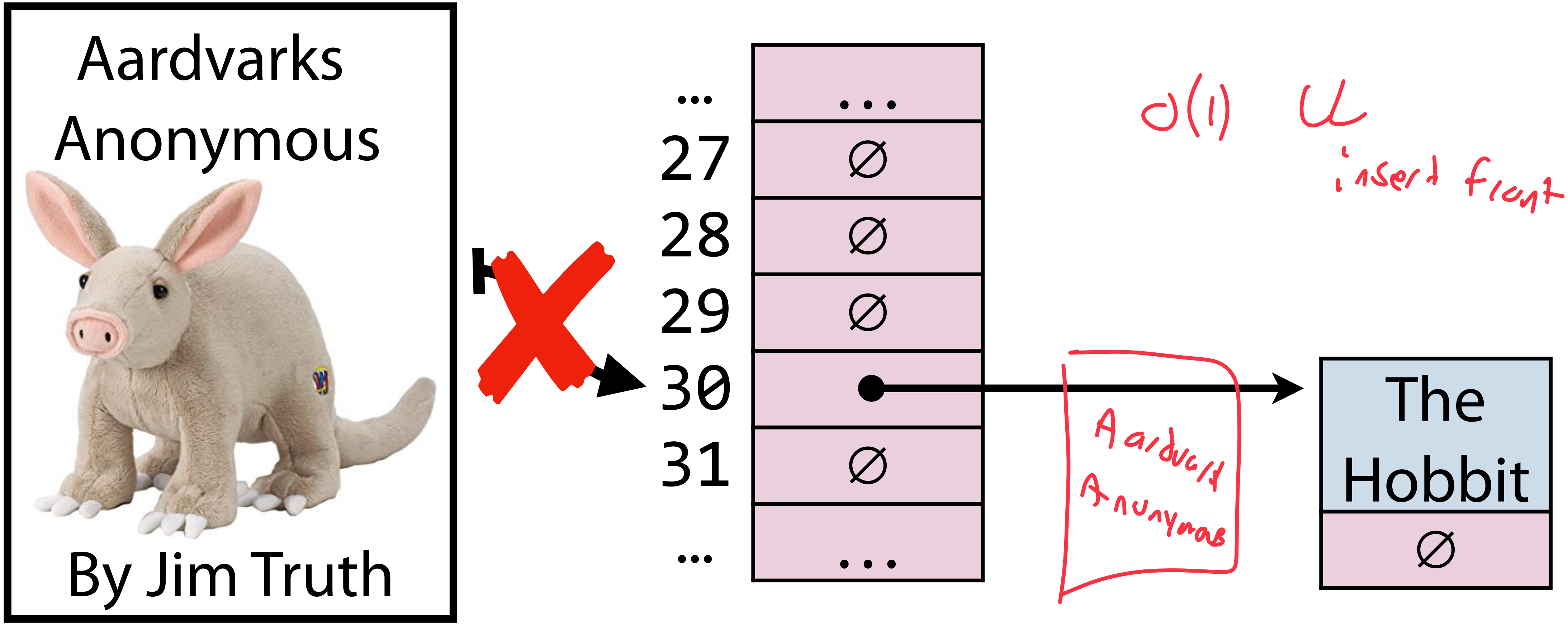
In an ***open hashing*** scheme, key-value pairs are stored externally (for example as a linked list).





Hash Collisions (Open Hashing)

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

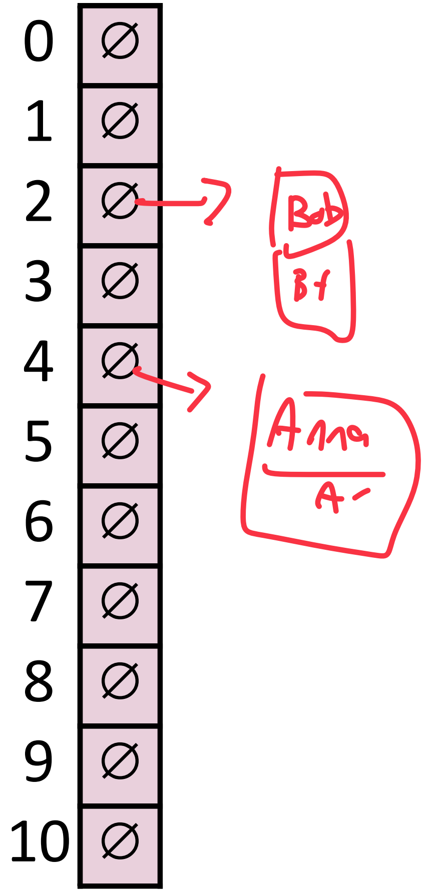


Insertion (Separate Chaining)

`_insert("Bob")`

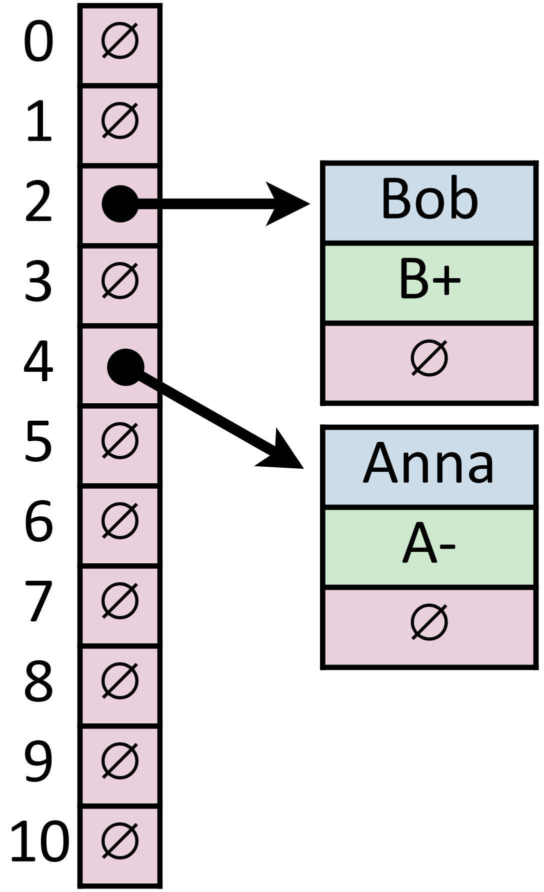
`_insert("Anna")`

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



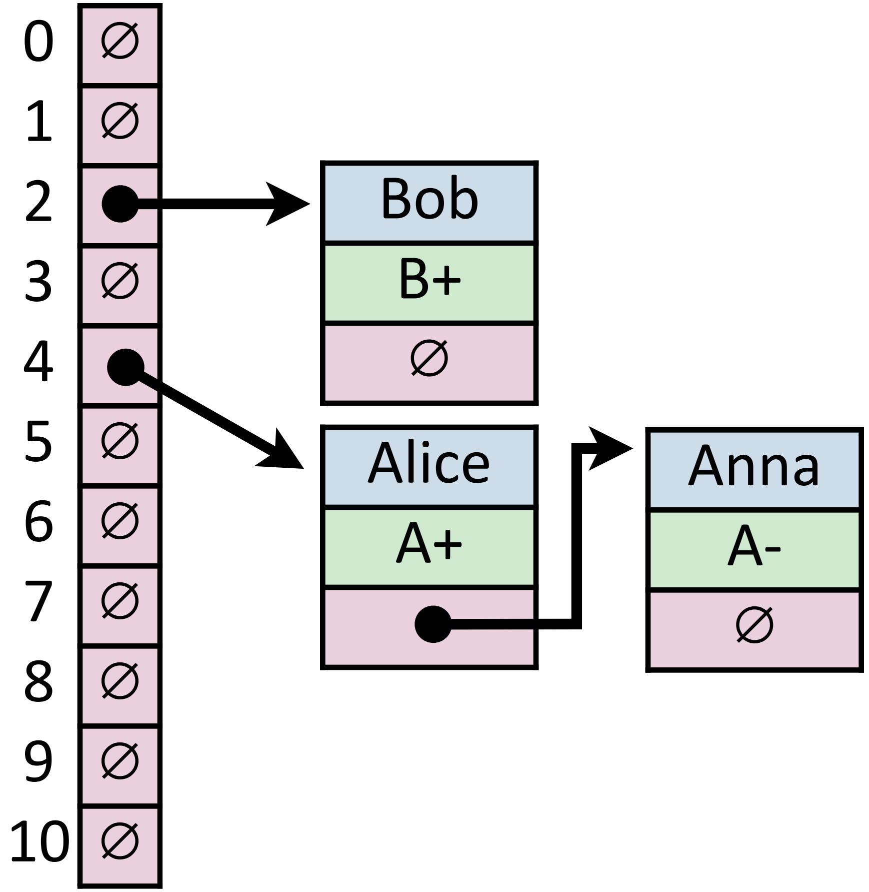
Insertion (Separate Chaining) _insert("Alice")

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



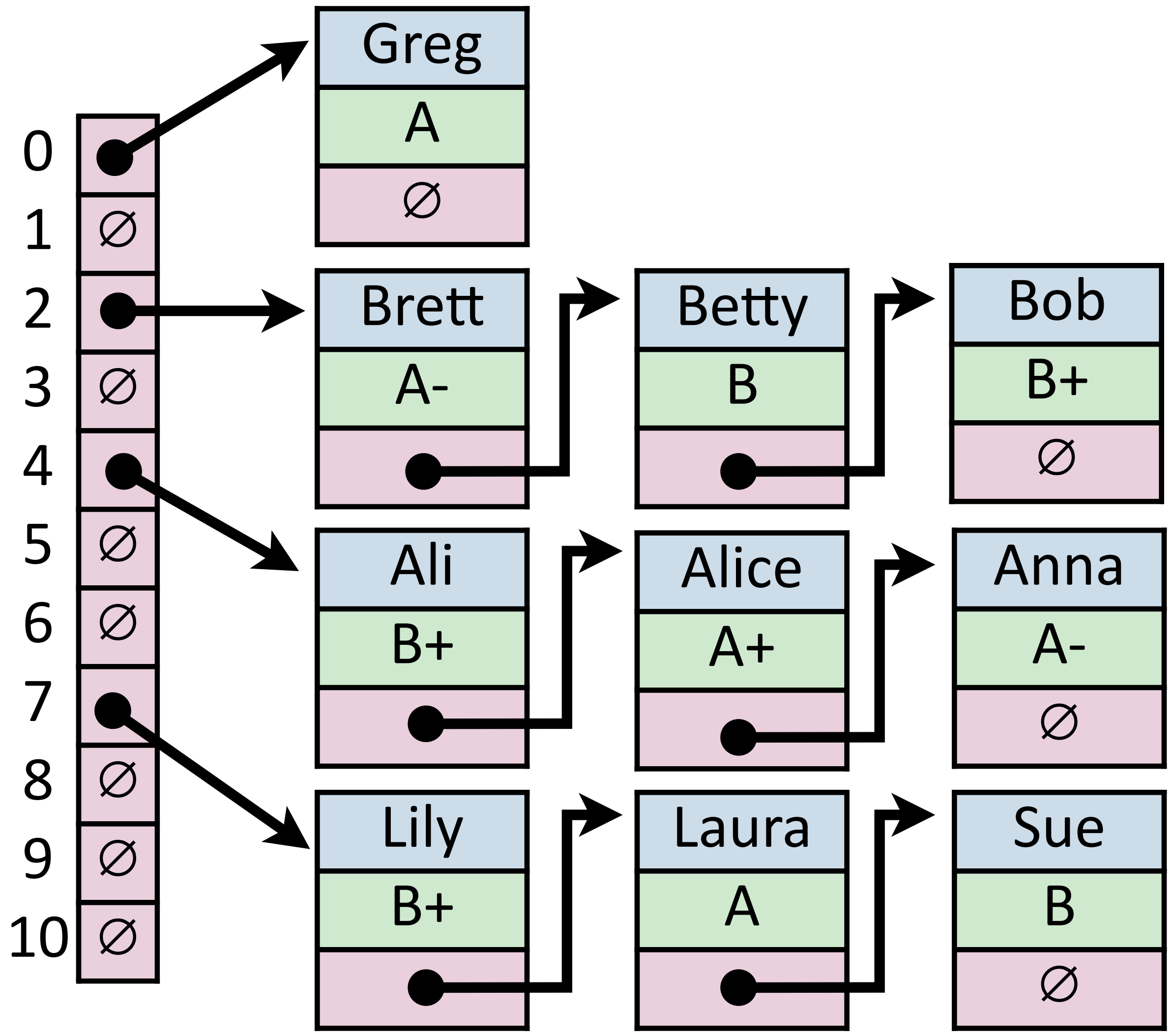
Insertion (Separate Chaining)

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



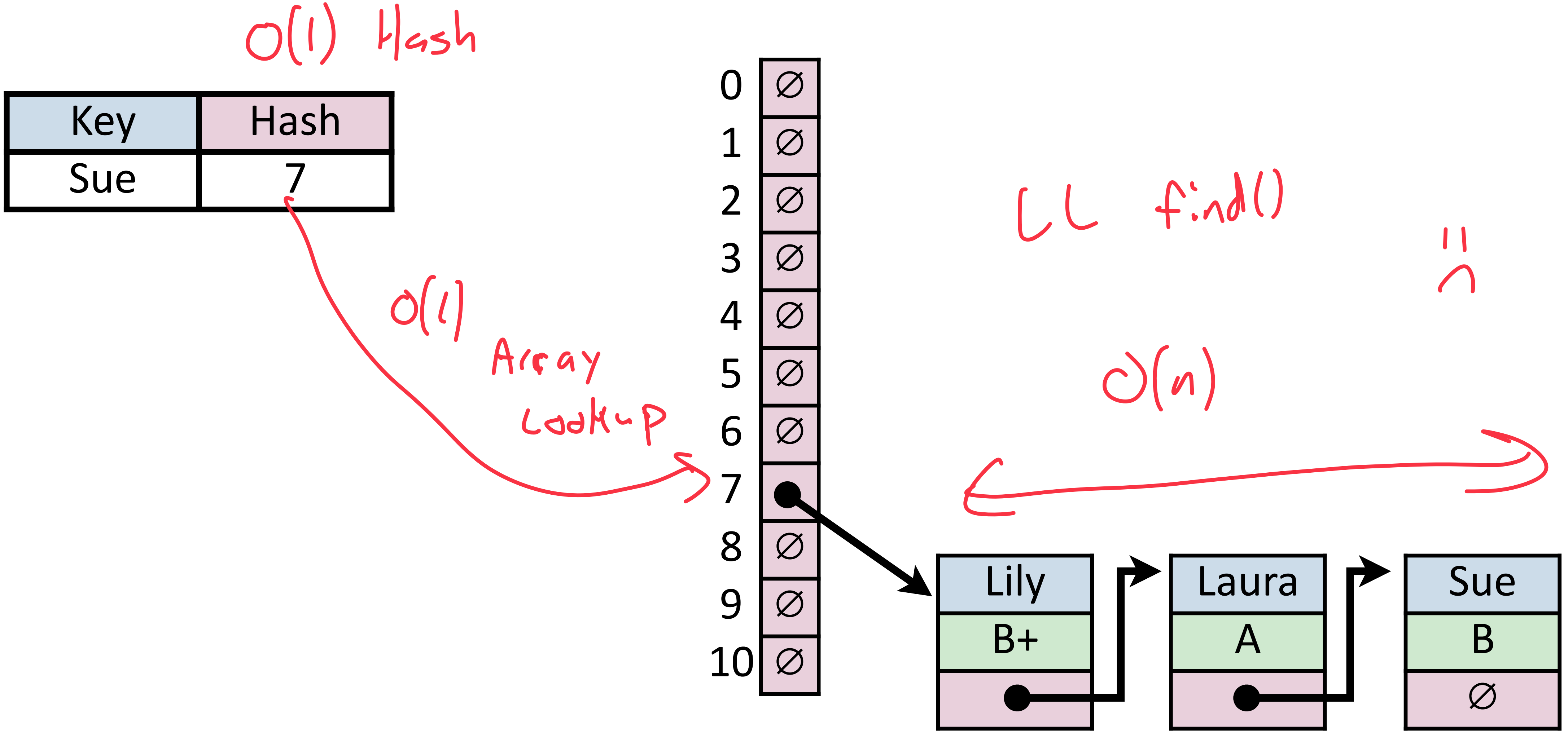
Insertion (Separate Chaining)

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")`



Remove (Separate Chaining)

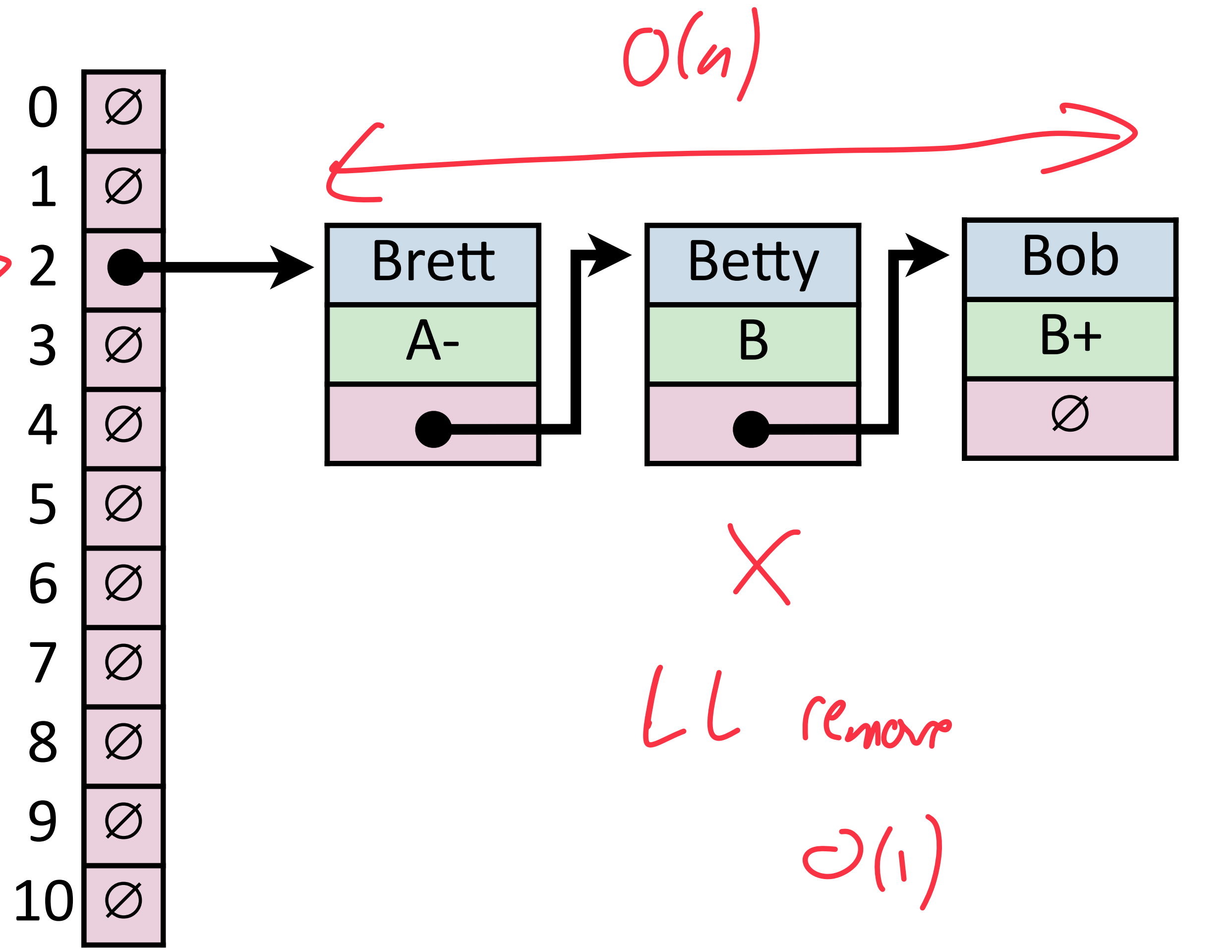
`_remove("Betty")`

Find

O(1)

Key	Hash
Betty	2

O(1)



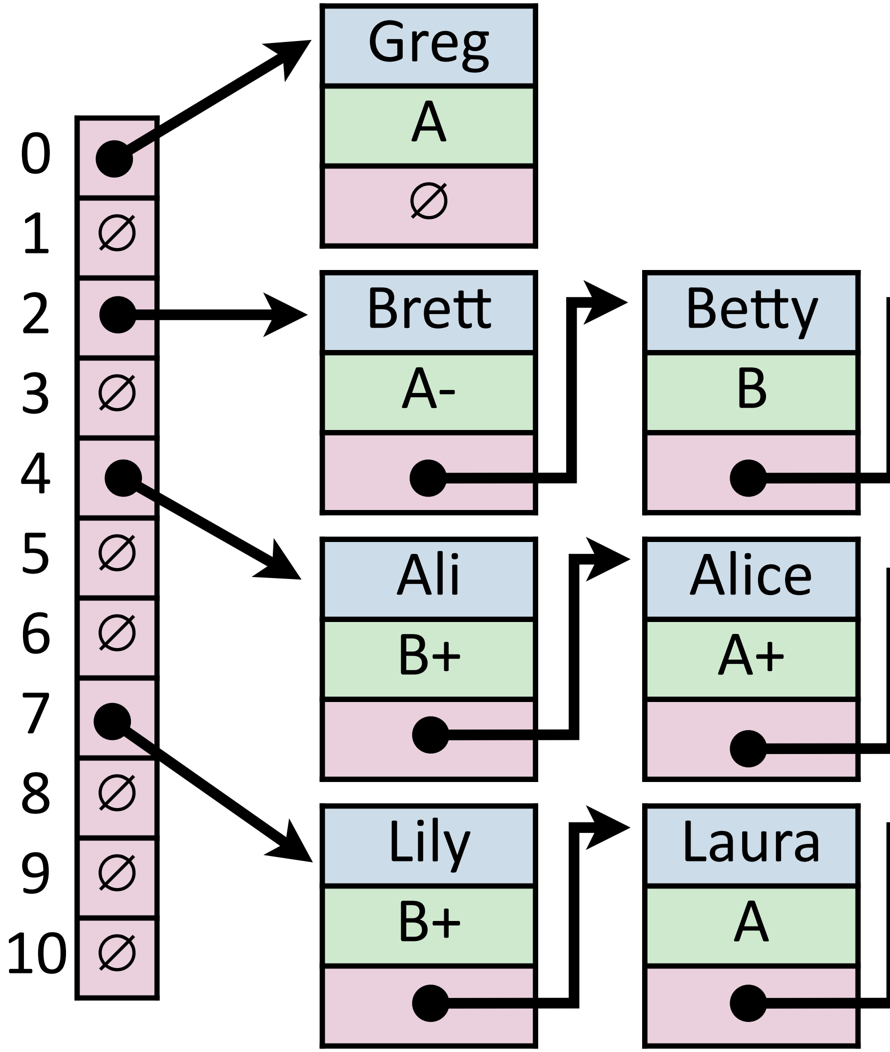
Hash Table (Separate Chaining)

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

Find runs in: O(n)

Insert runs in: O(1)

Remove runs in: O(n)



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 Hashing Assumption (Weak assumption)

↳ Randomness is on data

In reality pretty good!

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

↳ Real world way to get SUHA

↳ If we randomly pick 1 hash from family of hashes

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

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 $\frac{n}{m}$

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\mathbf{E[\alpha_j] = \frac{n}{m}}$$

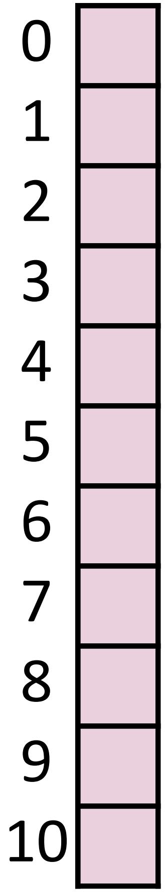
Separate Chaining Under SUHA

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

0	
1	
2	
3	
4	
5	
6	