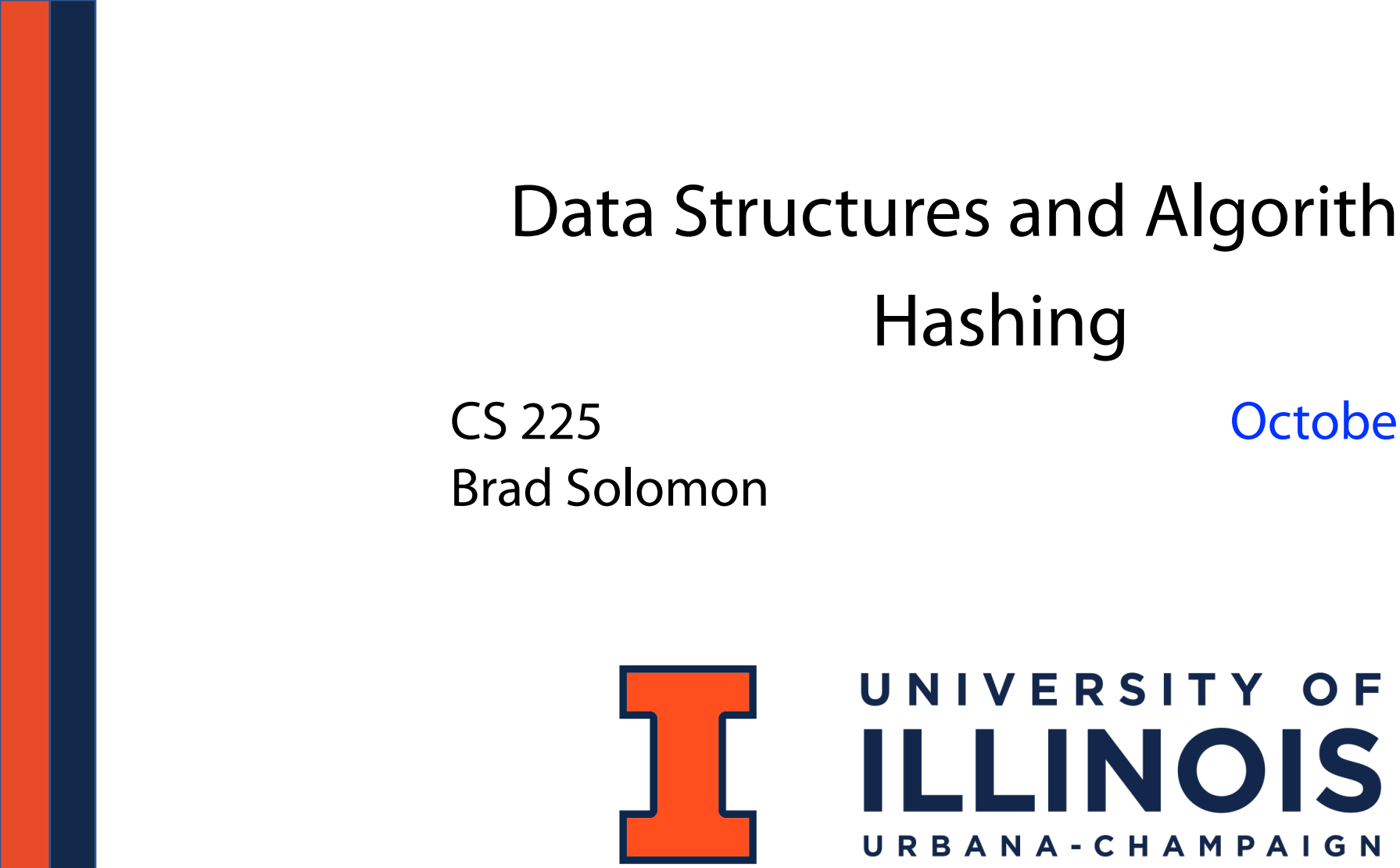
Data Structures and Algorithms

Hashing

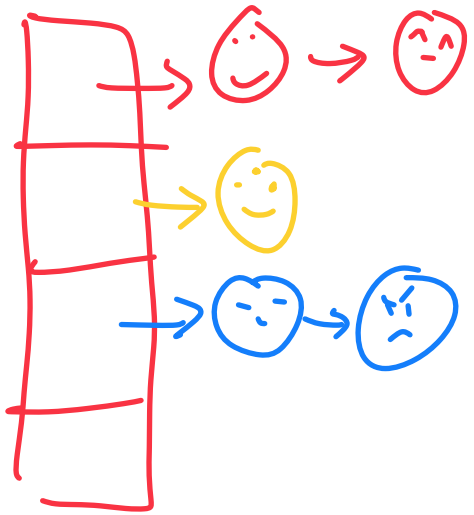
CS 225

Brad Solomon

October 27, 2023



Department of Computer Science



Extra Credit Projects — Submit by 10/31

"Approved by Nov 1"

↳ Submitted by 10/31 → Ideally 10/24

Its fine!

Randomization in Algorithms

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

2. Use randomness inside algorithm to estimate expected running time

↳ In the slides!

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

Learning Objectives

My fav core D.S.

Motivate and formally define a hash table

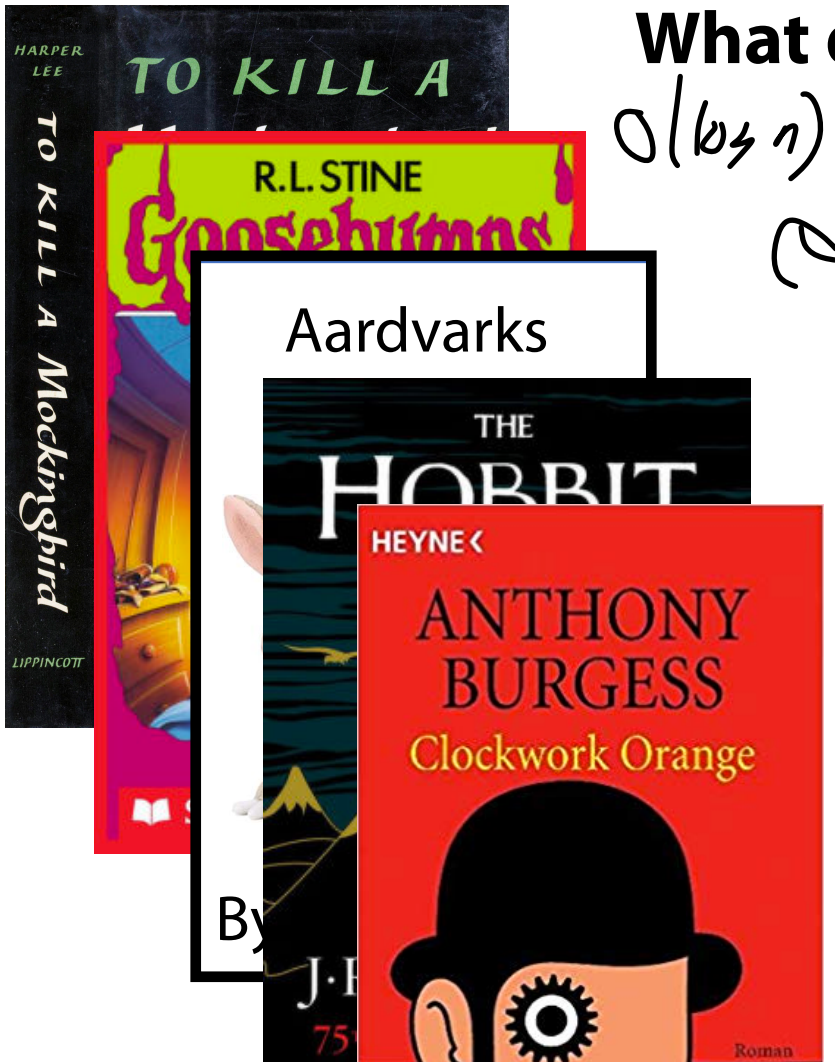
Discuss what a 'good' hash function looks like

Identify the key weakness of a hash table

Introduce strategies to "correct" this weakness

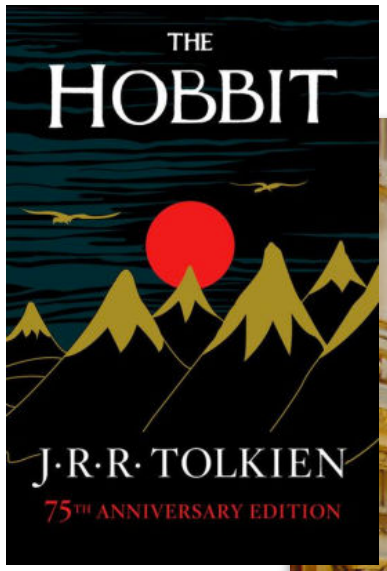
Data Structure Review

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

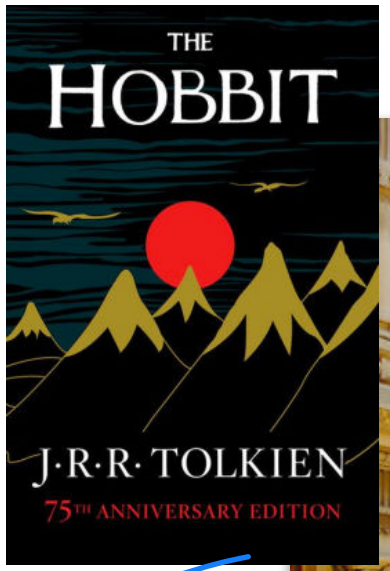


What data structures can I use here?

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



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



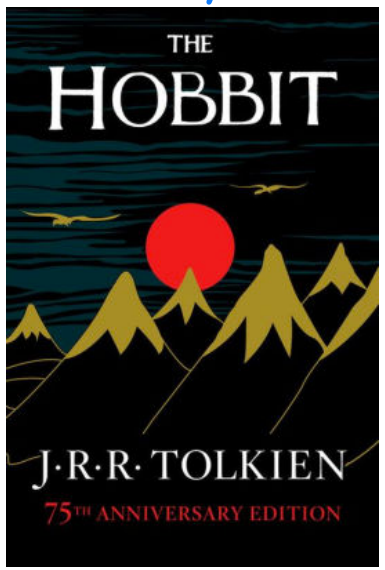
O(1) →

O(1)

A Hash Table based Dictionary

(Memory)

Key



Black Box

O(1)

Address

ISBN: 9780062265722

Call #: PR
6068.O93
H35 1937

address

ISBN: 9780062265722

Call #: PR
6068.O93
H35 1937

O(1)

Value

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

Randomized Data Structures

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

e.g. AVL tree (log n)

Randomized data structures rely on **expected** performance

Expected performance good!

Randomized data structures 'cheat' tradeoffs!

Worst case is bad

A Hash Table based Dictionary

User Code (is a map):

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

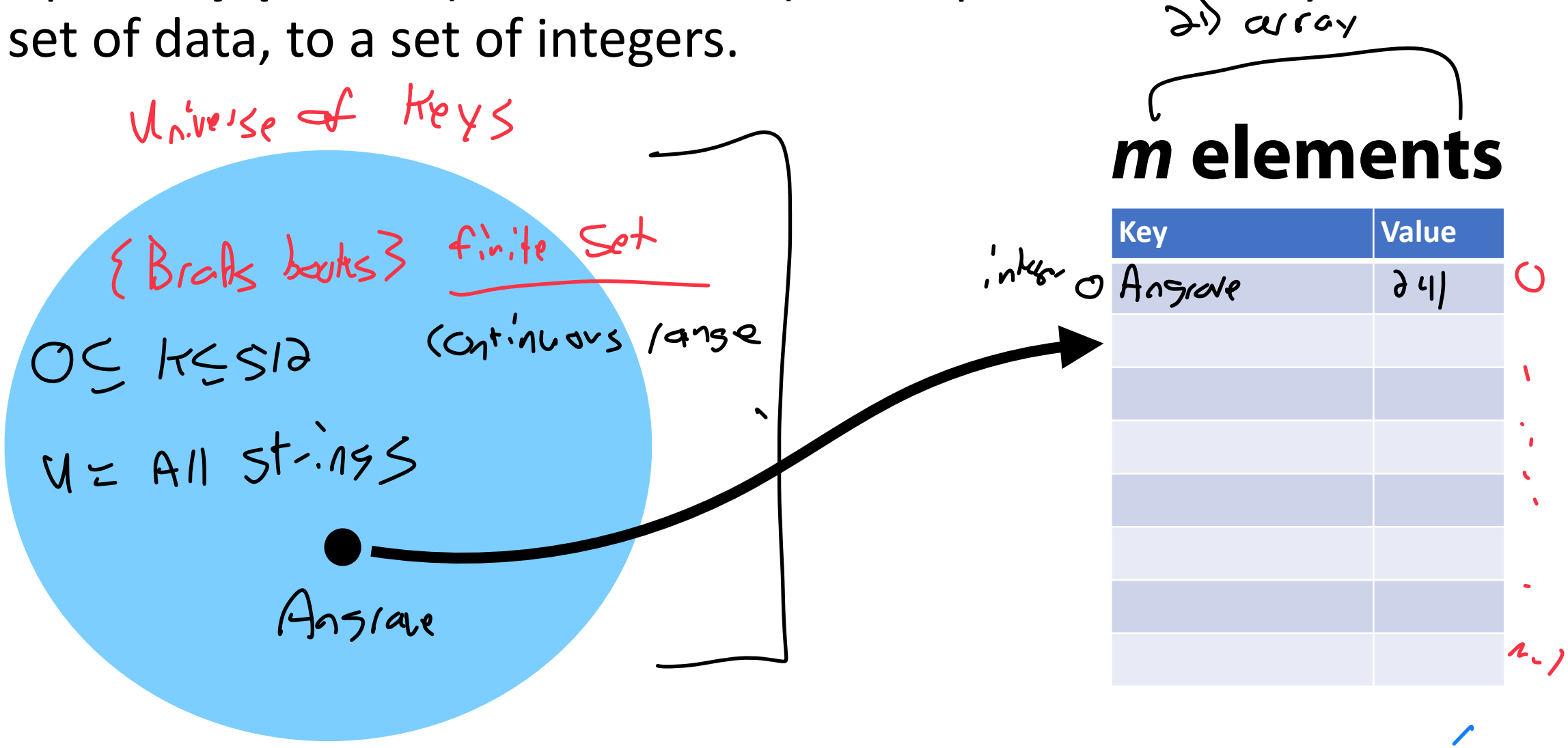
(KeyType) → int
uint64_t
etc

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

1. A hash function — key as input → returns an address given index → O(1) lookup
2. A data storage structure (An array is good!)
3. ??? A way of handling chaos

Hash Function

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



Hash Function

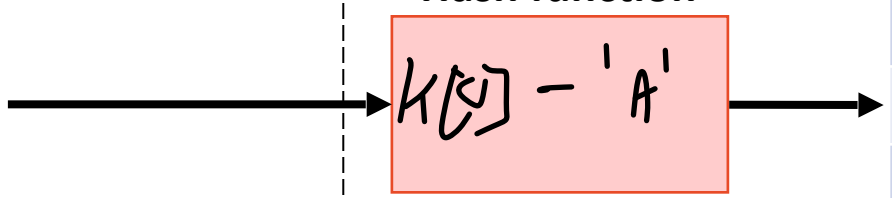
A hash function ***must*** be:

- **Deterministic**: Given the same key, return the same value
  - If math major obvious!
- **Efficient**: Our goal is O(1). The hash function is run every time [Be aware of our key size but indep of key size]
- **Defined for a certain size table:**
  - % m (size of table)

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)



	Key	Value
0	Angrave	241
	Beckma	421

Hash Function

Good

↳ A perfect bijection!

Are all letters equally likely?

↳ Bad hash function!

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

Only for 26 letters

Hash function

(key[0] - 'A')

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

Bad: Can't have two people w/ same first letter.

General Hash Function

H : KeyType → int

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

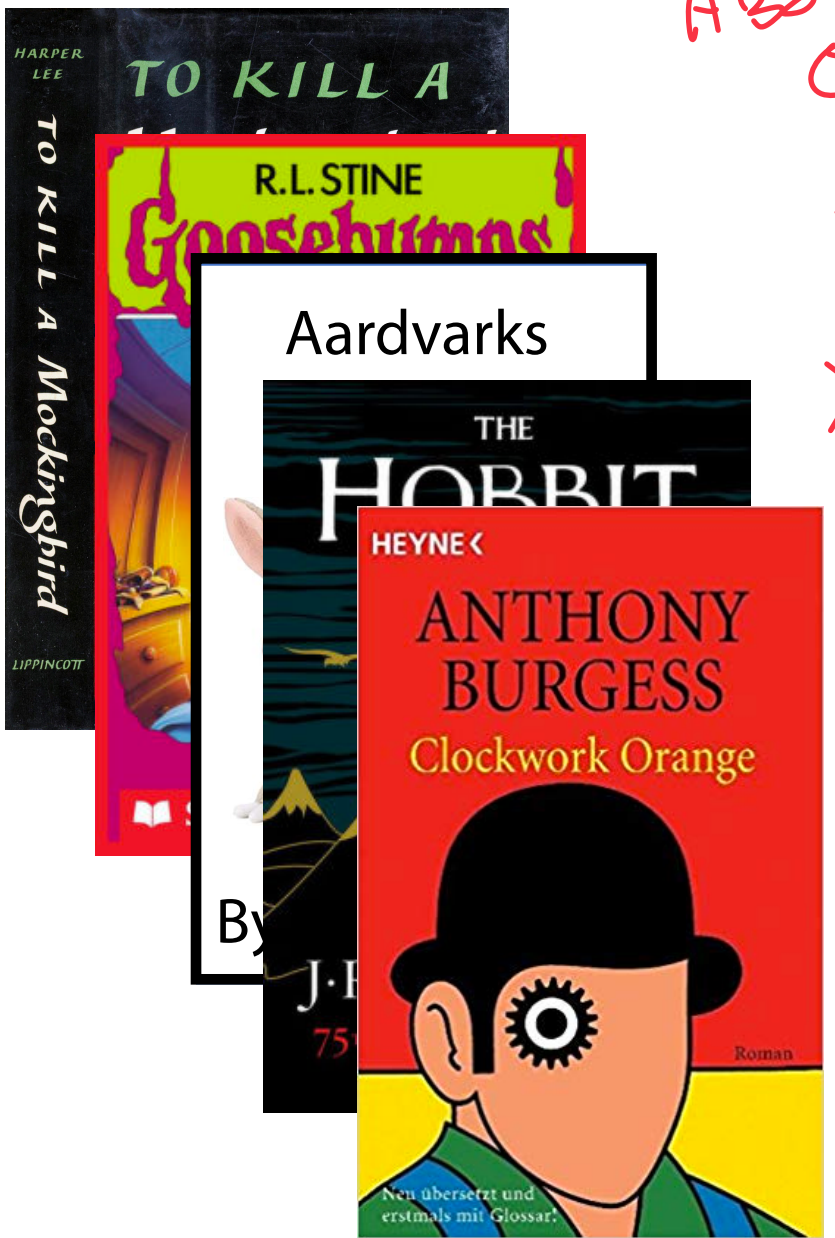
- A **hash**: Key → int
- A **compression**: Mod. H(x) % m

Choosing a good hash function is tricky…

- Don't create your own (yet*)

Lab_Hom
↳ C++ (int) → int
↳ return x

Hash Function



Are those hash functions?

Also I look for author

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

↳ Is O(1) ✓

First & last names are not uniformly distributed

deterministic ✓

✗ $h(k) = \left(\cancel{rand()} * k.numPages\right) \% m$

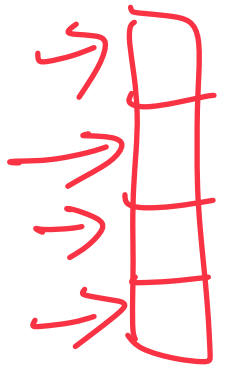
↳ Not a hash function!

Can't have randomness inside function!

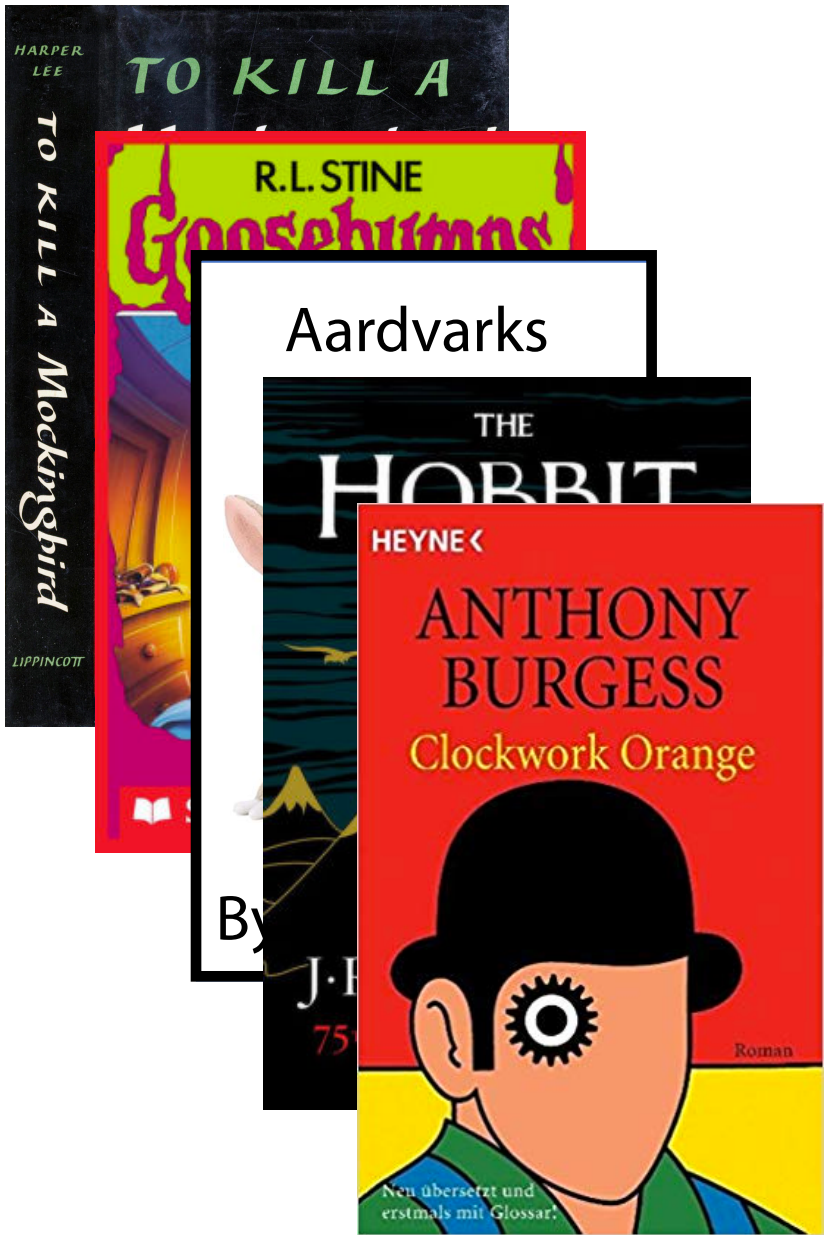
Started my program by generating one #

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

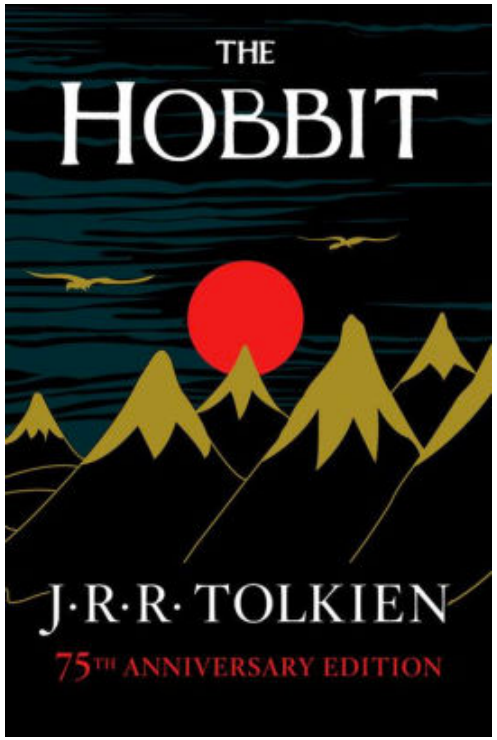
↳ Not deterministic!



Hash Function



A Hash Table based Dictionary



Key

'J' + 'T' = 30

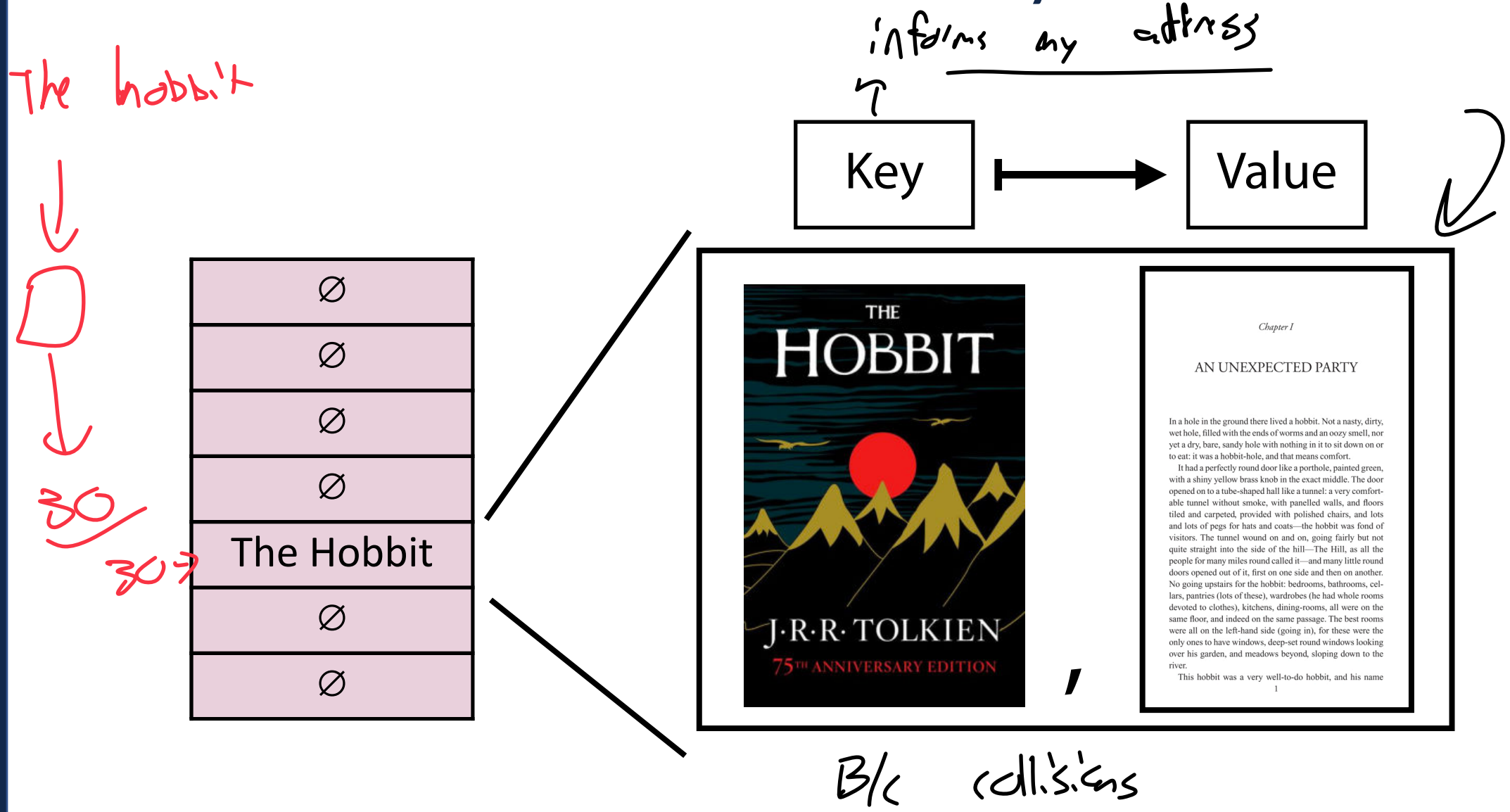
Author Name Hash Function

O(1)

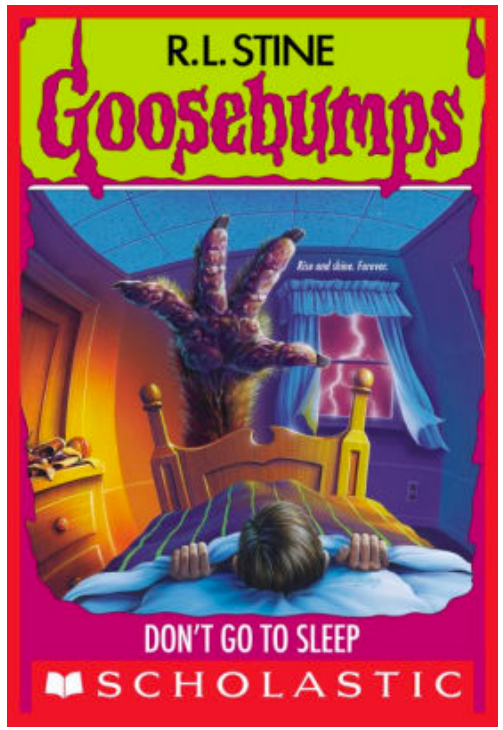
int

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

A Hash Table based Dictionary



A Hash Table based Dictionary

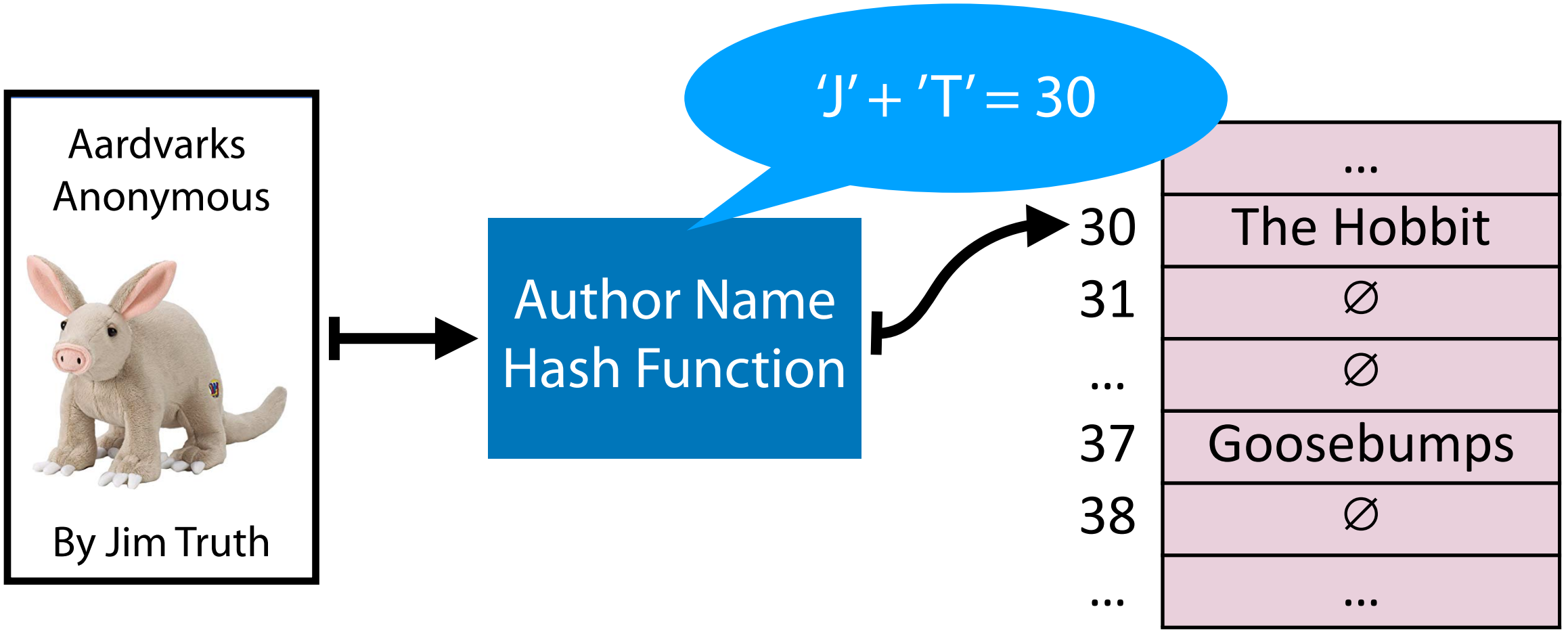


'R' + 'S' = 37

Author Name Hash Function

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

A Hash Table based Dictionary

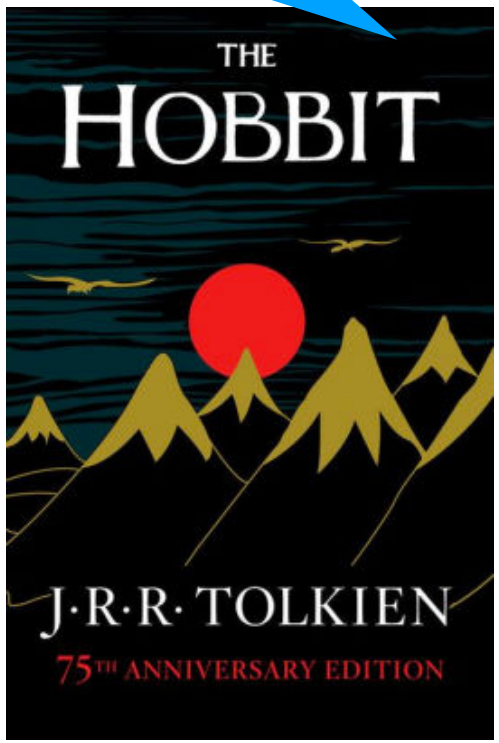


Hash Collision

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

hash

J.R.R Tolkien = 30!



Jim Truth = 30!

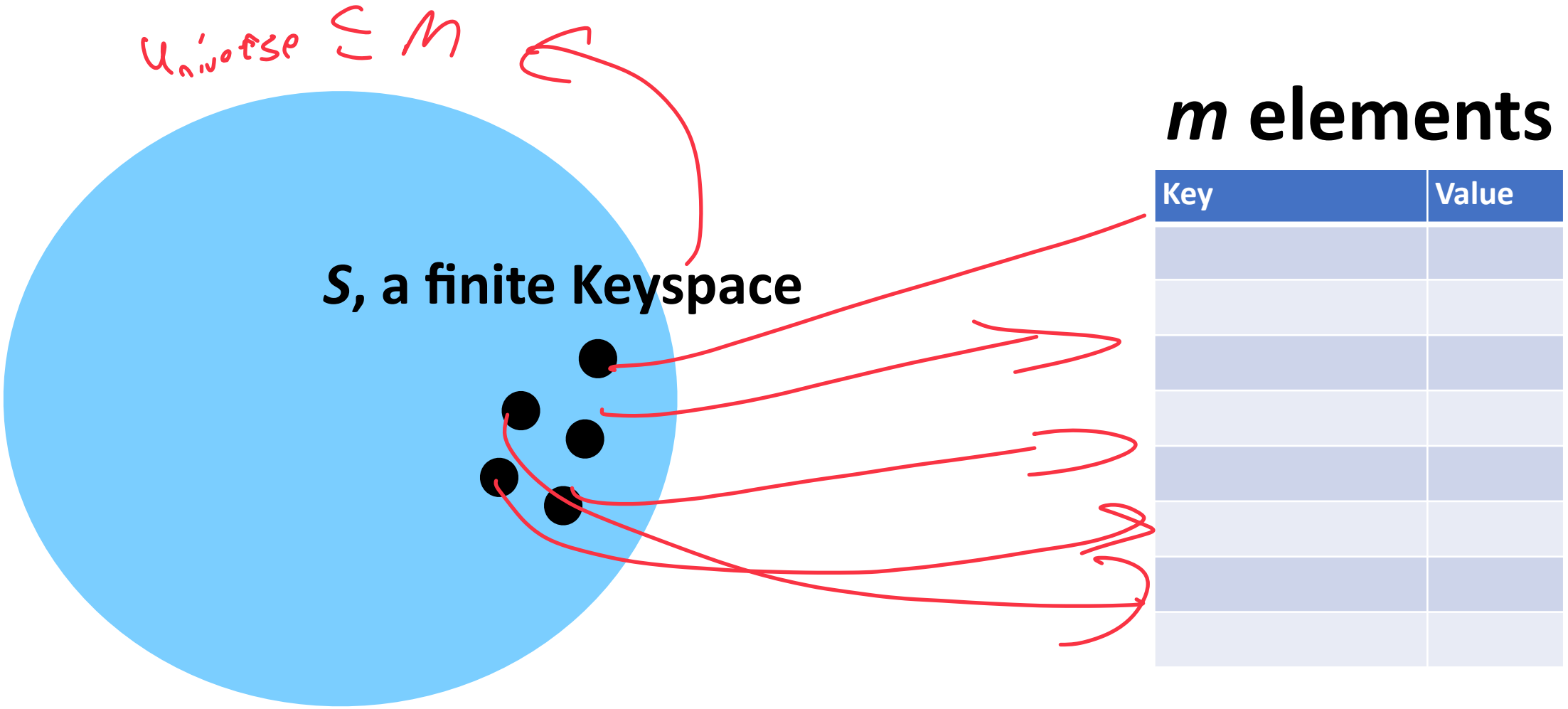
Aardvarks Anonymous

By Jim Truth

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

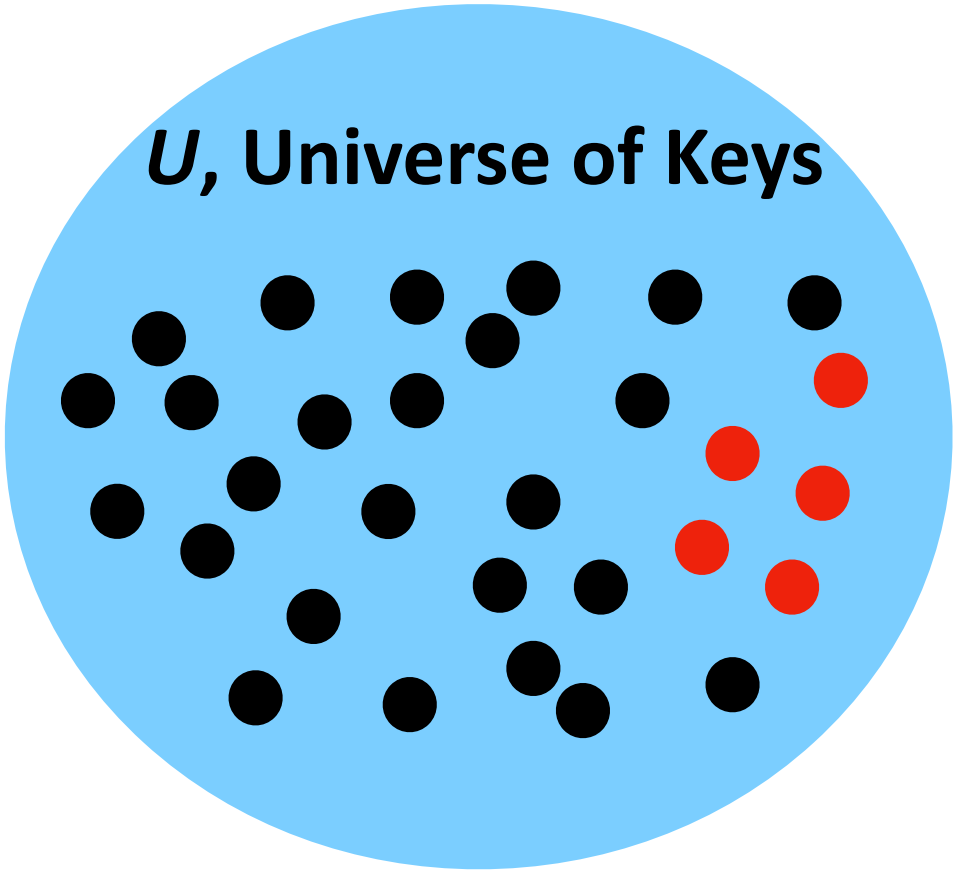
Perfect Hashing

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*.

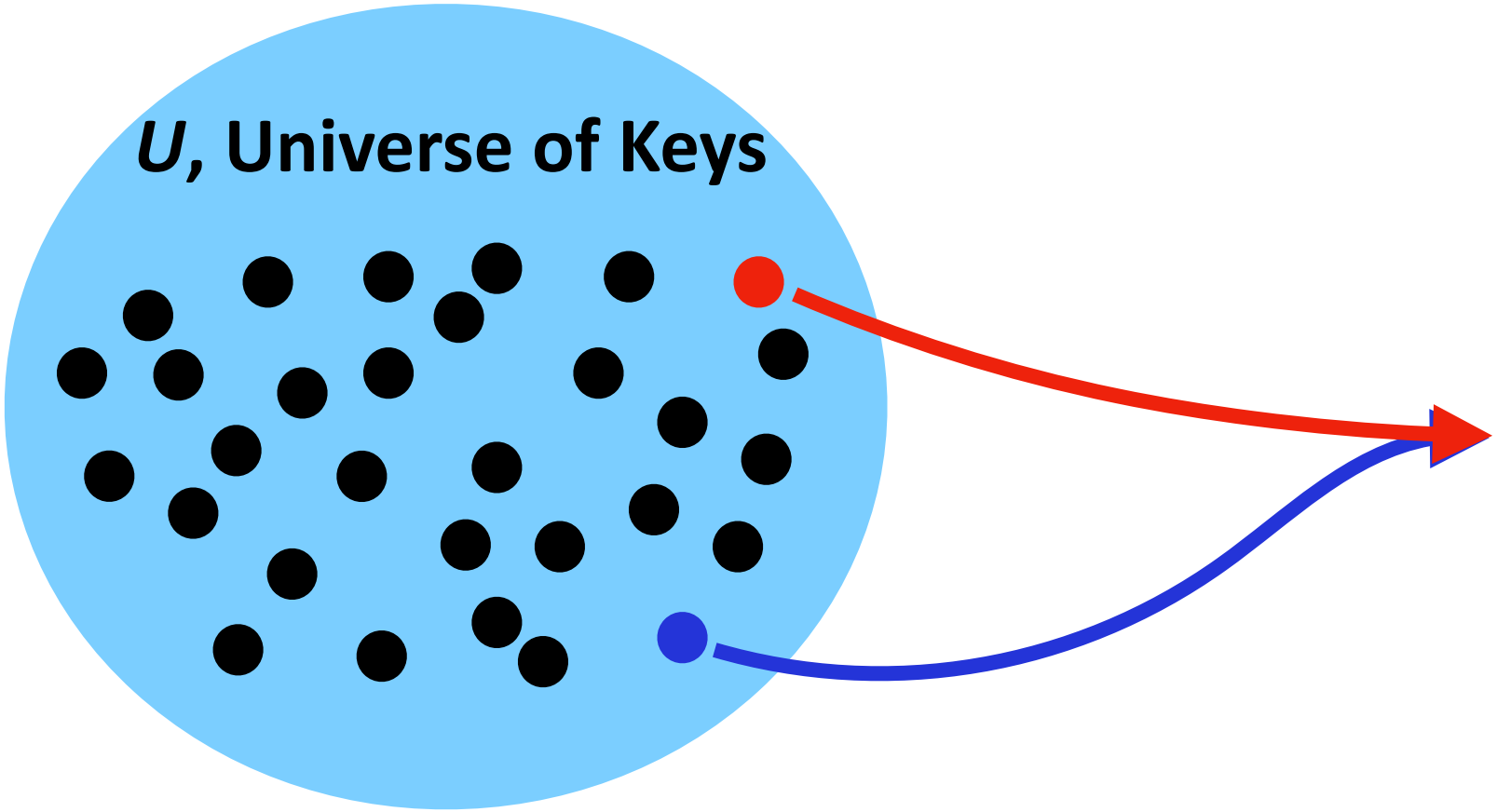


***m* elements**

Key	Value

General Purpose Hashing

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

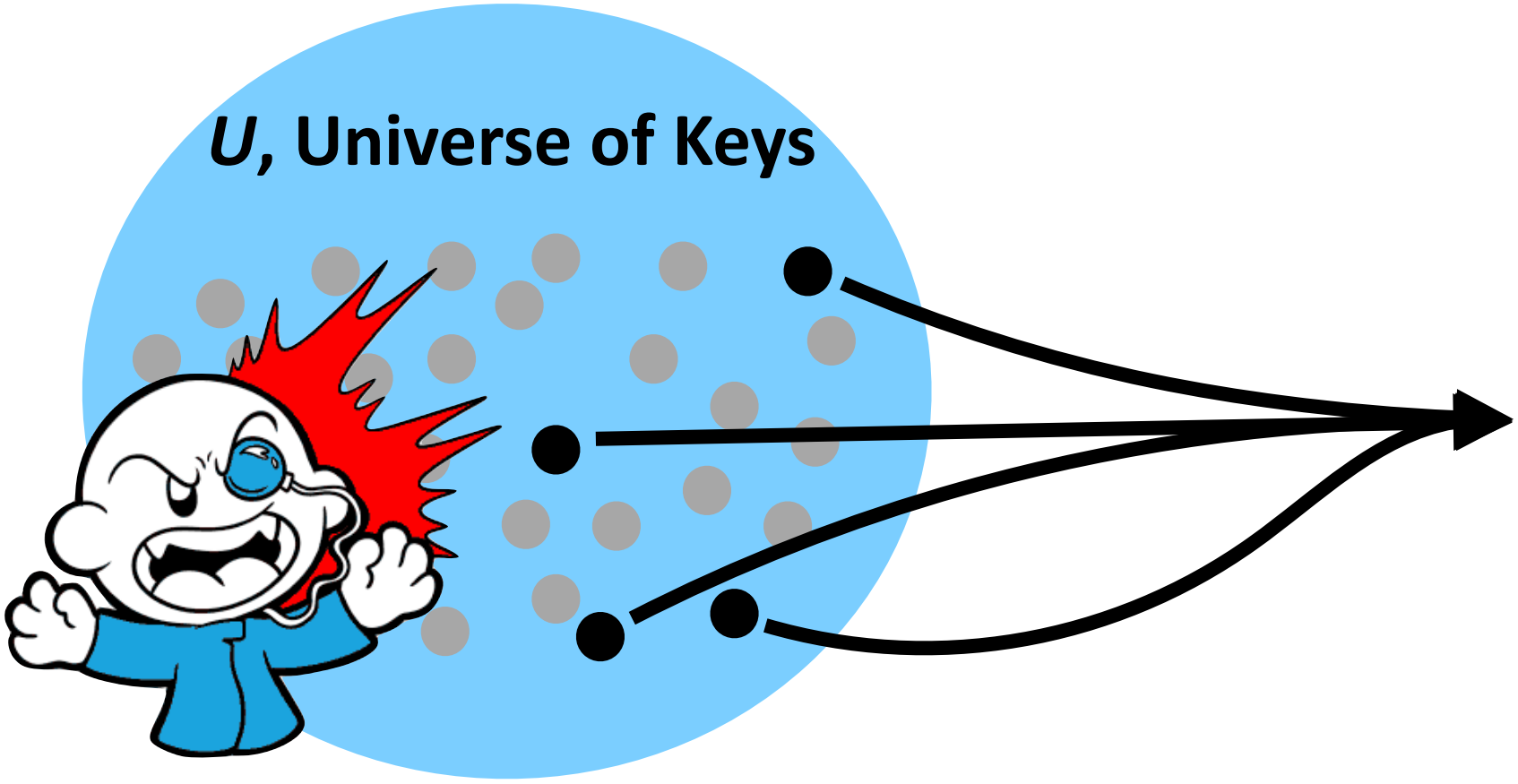


m elements

Key	Value

General Purpose Hashing

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



***m* elements**

Key	Value

Image by Matthew Loffhagen

A Hash Table based Dictionary

User Code (is a map):

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

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

1. A hash function

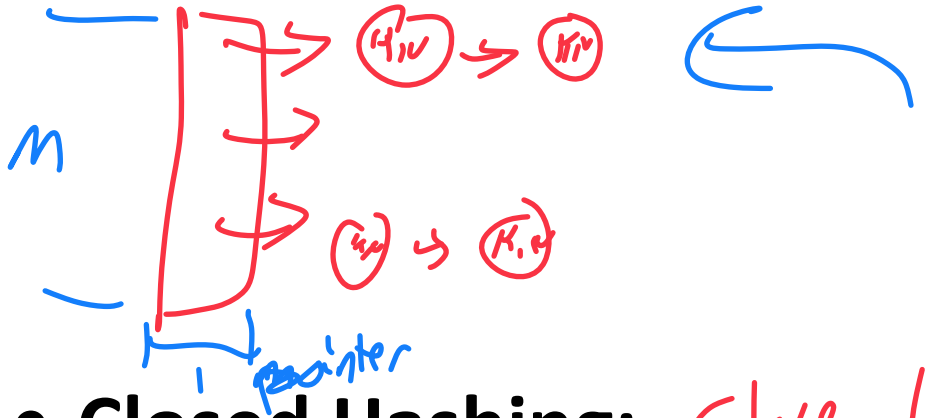
2. A data storage structure

3. A method of addressing *hash collisions*

Open vs Closed Hashing

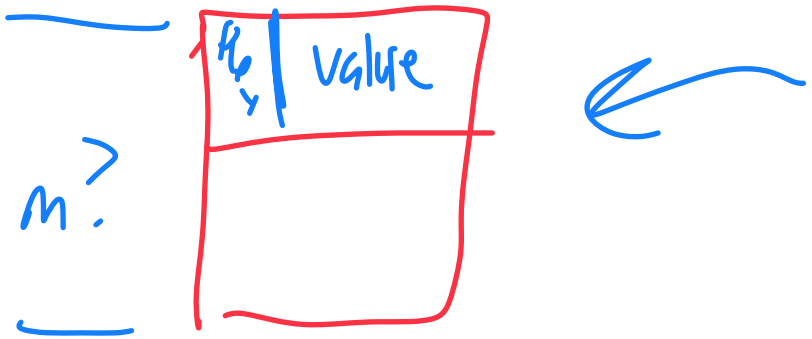
Addressing hash collisions depends on your storage structure.

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



Easy!

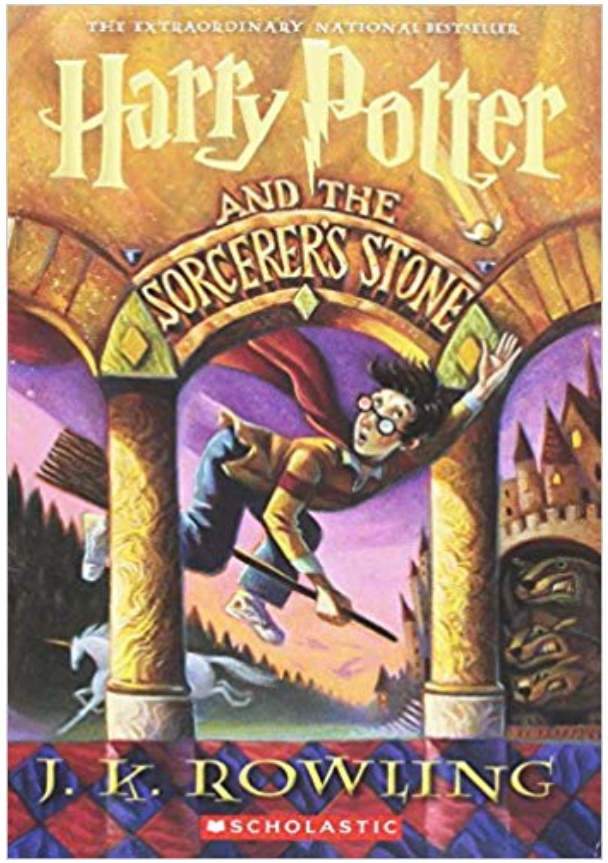
- **Closed Hashing:** Store (k,v) internally (in my table)

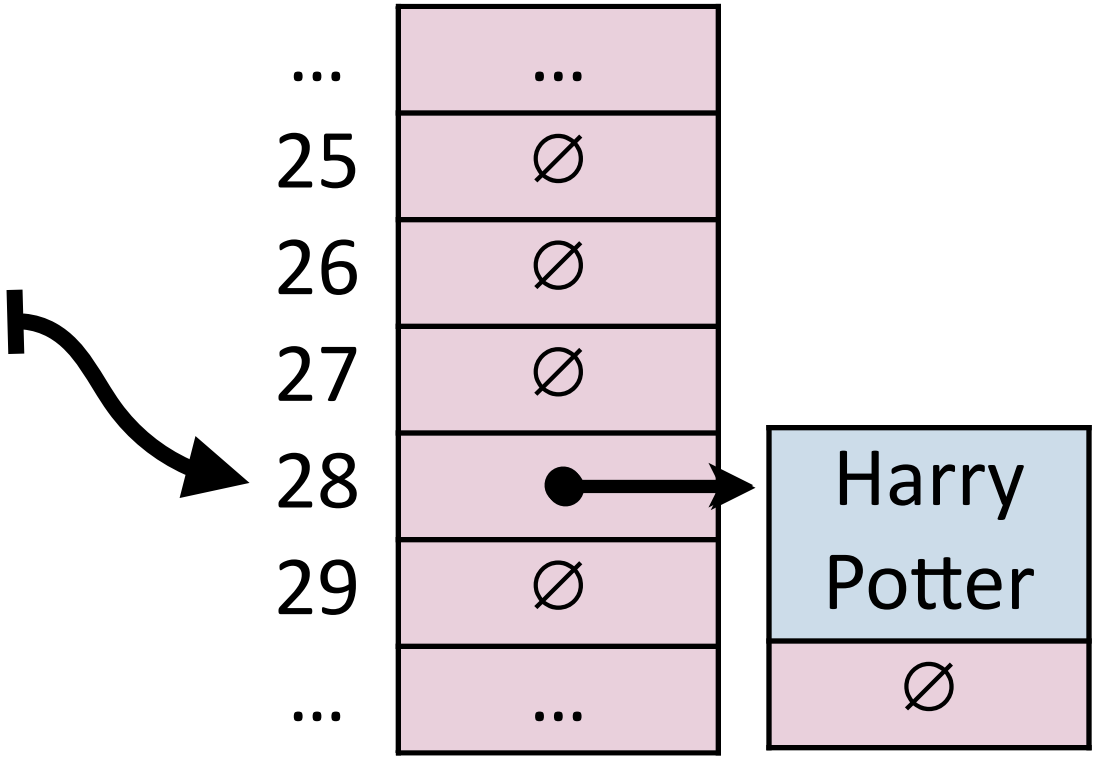


Hard!

Open Hashing

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





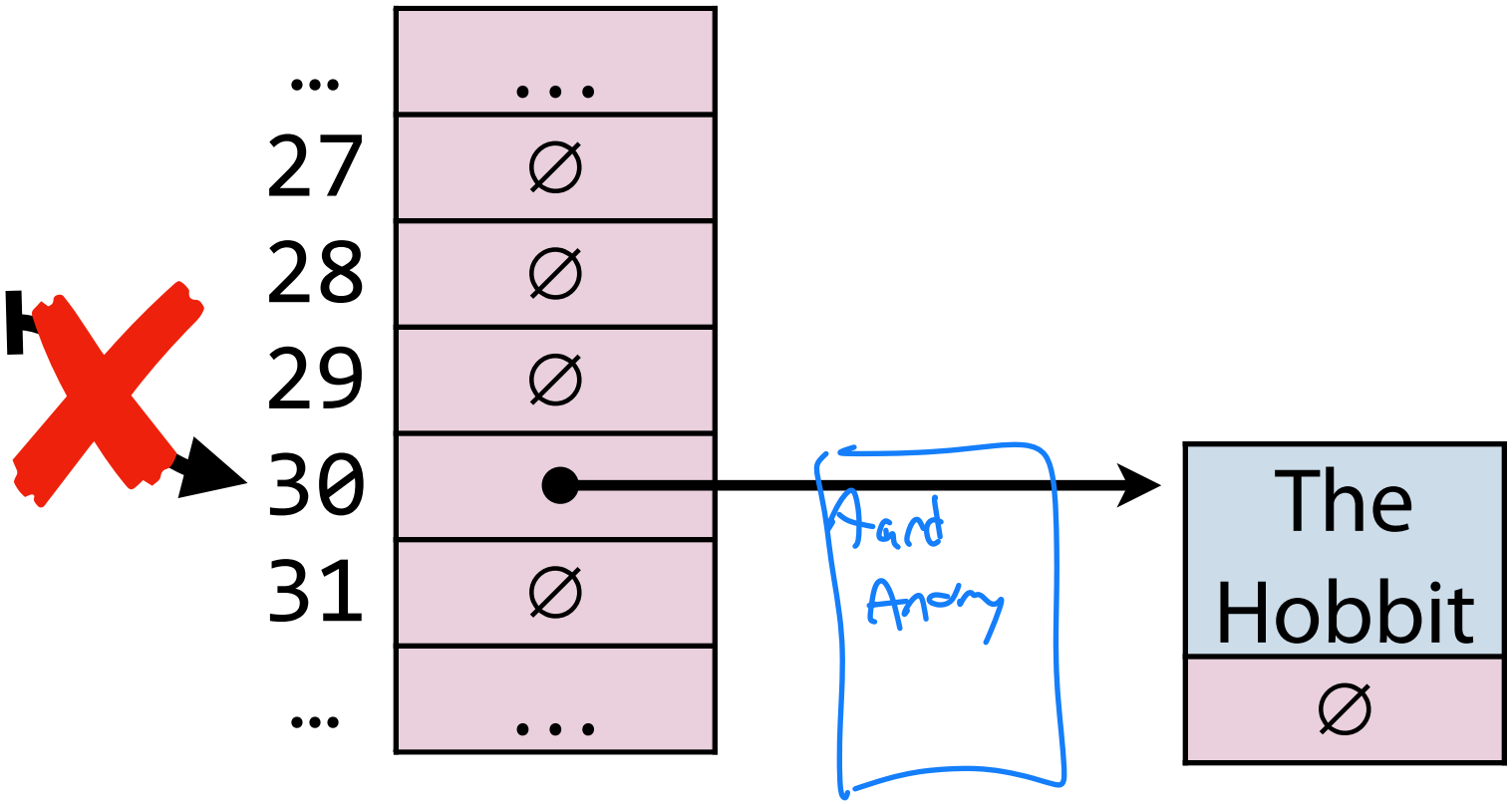
Hash Collisions (Open Hashing)

A ***hash collision*** in an open hashing scheme can be resolved by inserting at front. This is called ***separate chaining***.

O(1) insert at front

Aardvarks Anonymous

By Jim Truth



Insertion (Separate Chaining)

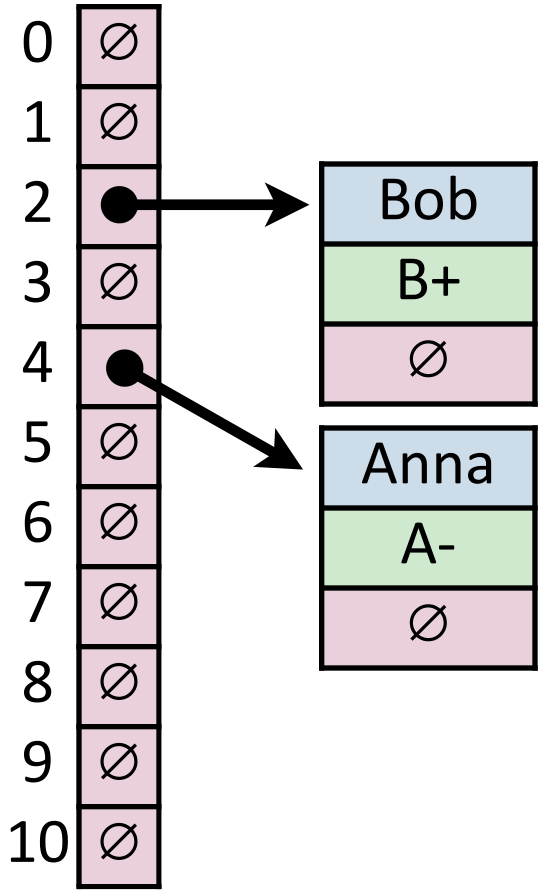
```
_insert("Bob")

_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")`

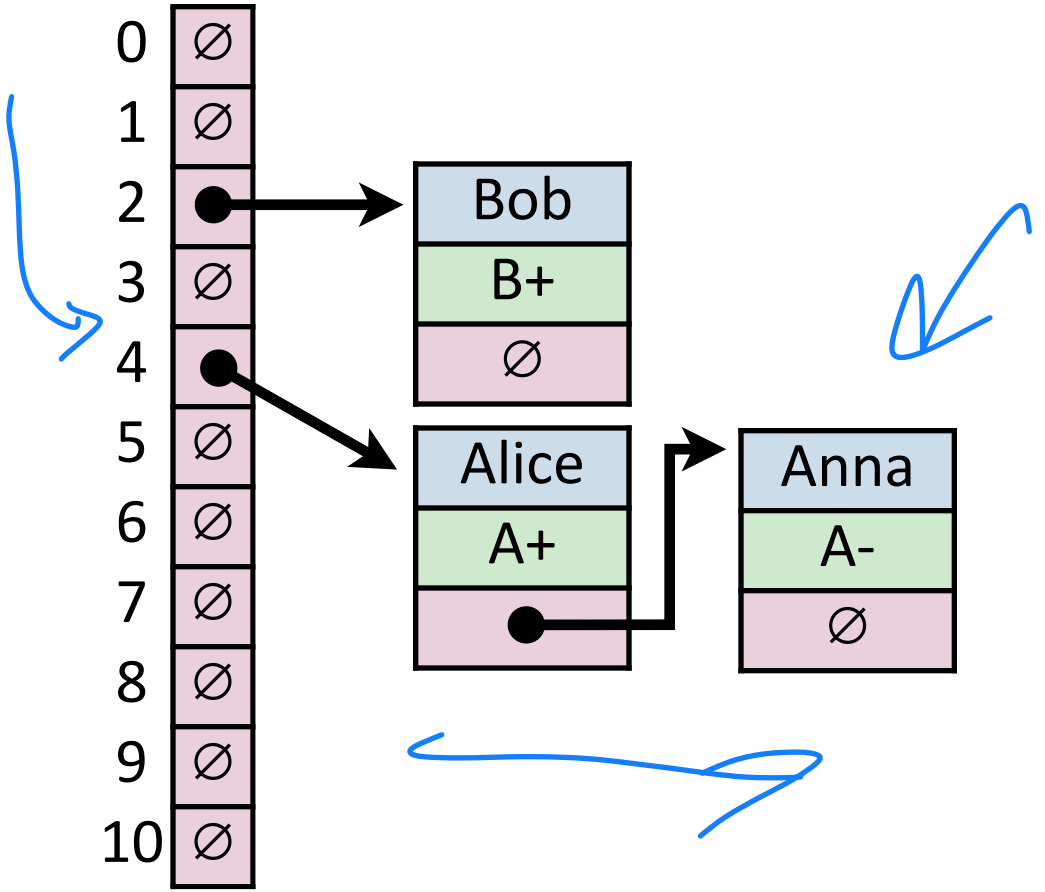
Key	Value	Hash
Bob	B+	2
Anna	A-	4
Alice	**A+**	**4**
Betty	B	2
Brett	A-	2
Greg	A	0
Sue	B	7
Ali	B+	4
Laura	A	7
Lily	B+	7



Insertion (Separate Chaining)

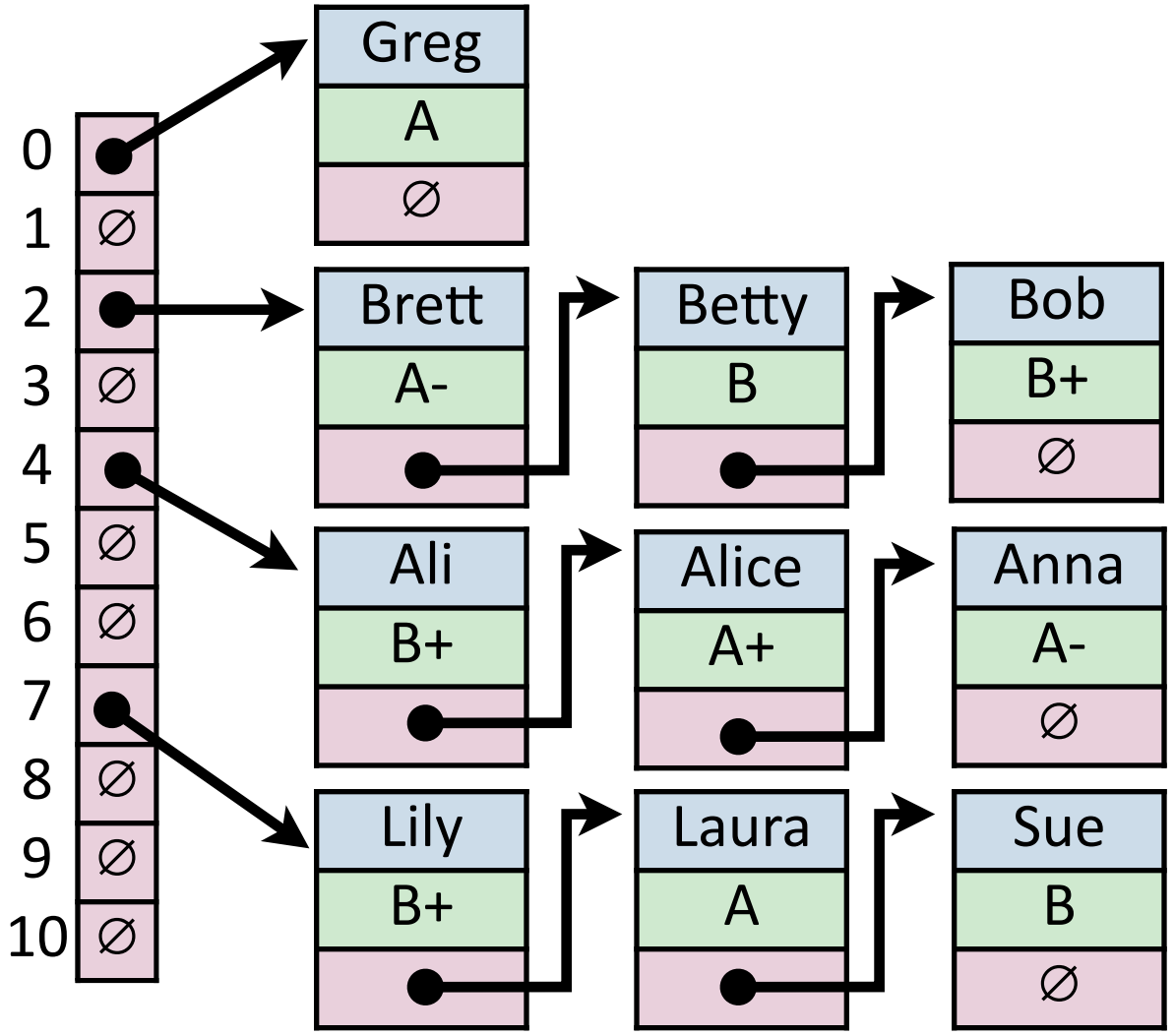
find(key)

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



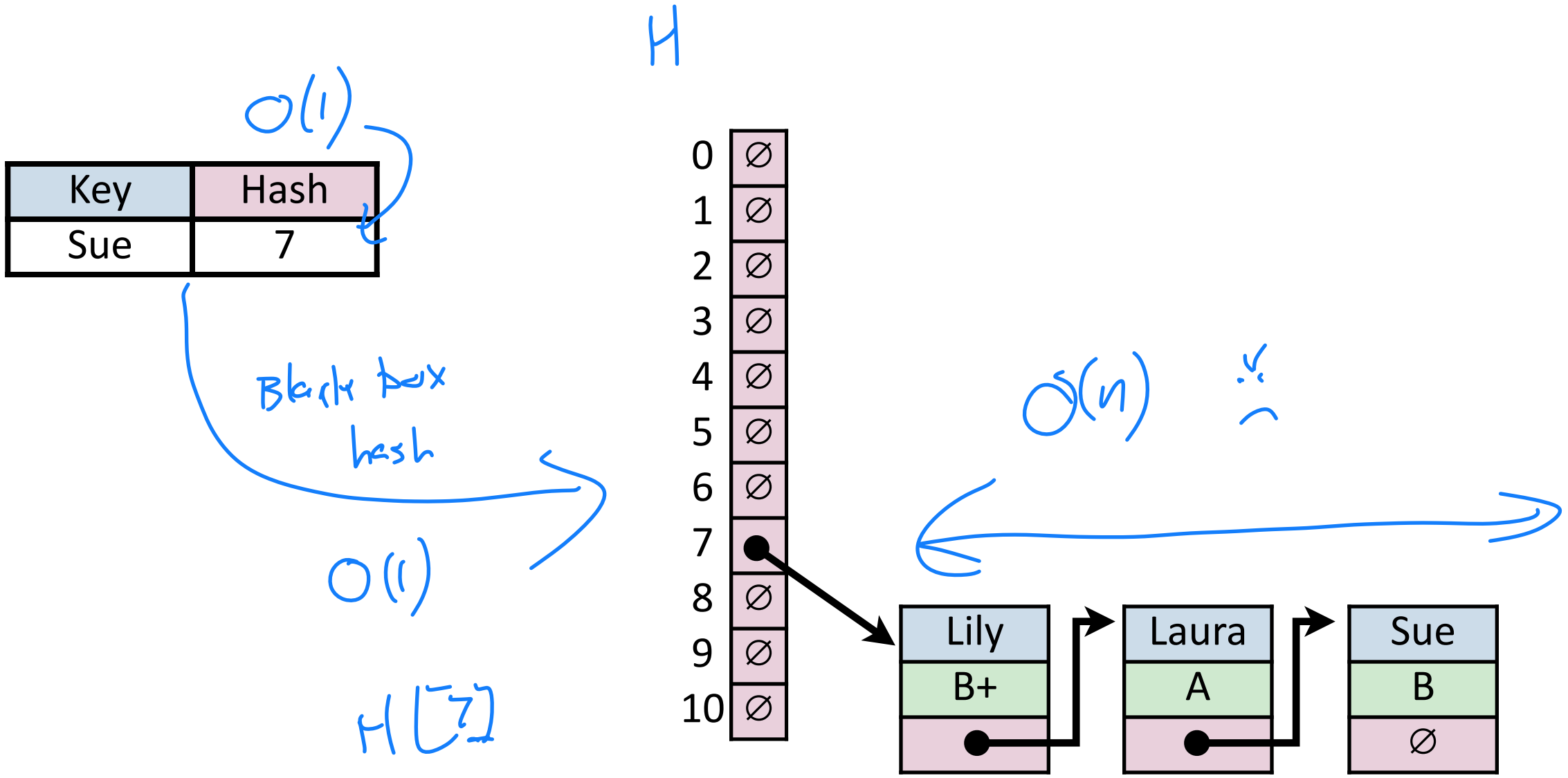
Insertion (Separate Chaining)

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")
```



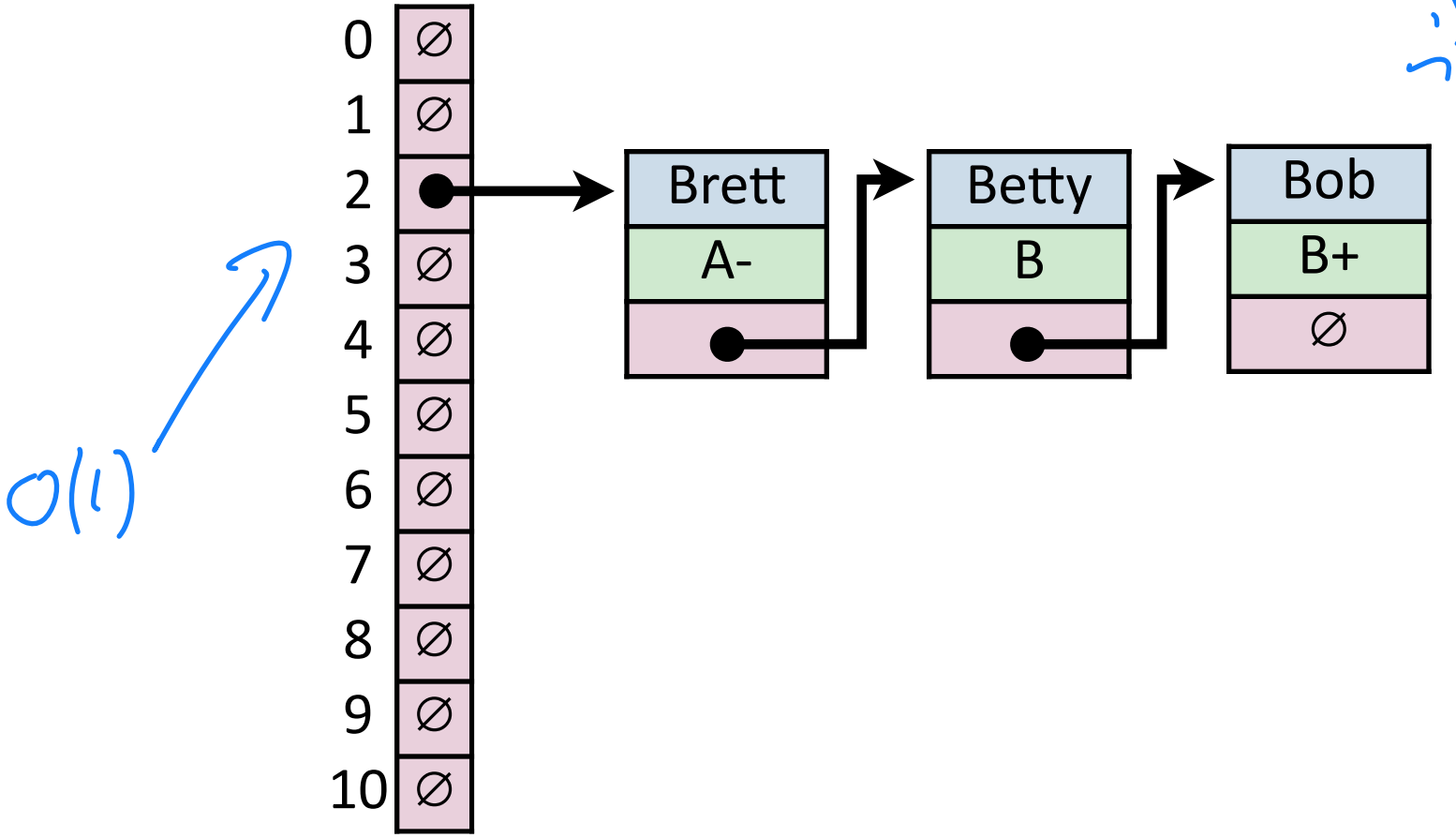
Remove (Separate Chaining)

`_remove("Betty")`

Key	Hash
Betty	2

O(1)

LL remove O(n)



Hash Table (Separate Chaining)

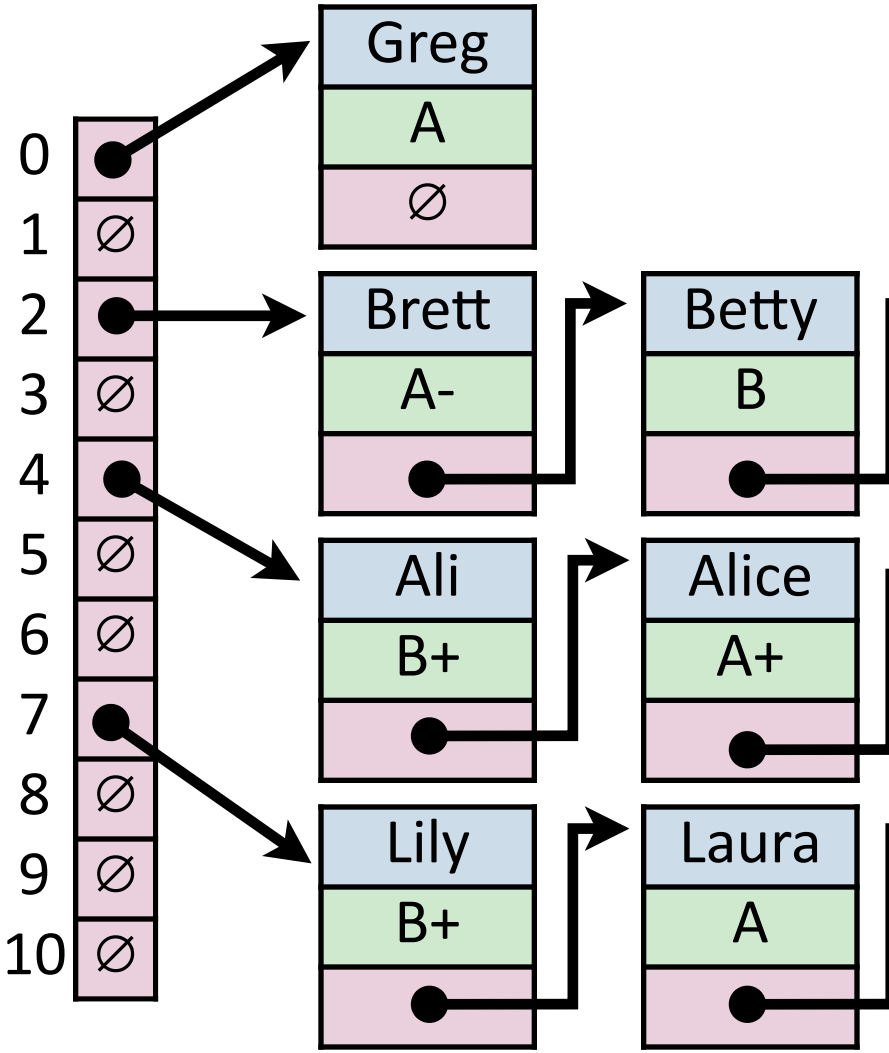
For hash table of size *m* and *n* elements:

Find runs in: O(n)

Insert runs in: O(1) :)

Remove runs in: O(n)

↳ B/c of collisions!



Hash Table

In expectation

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

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

by secret (SUHA in real life)

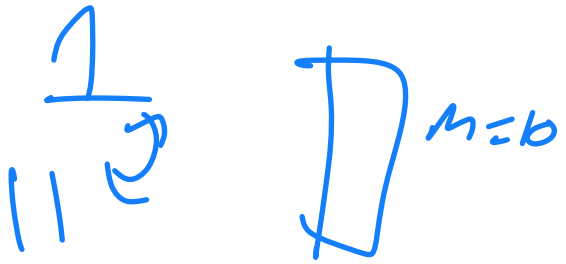
Simple Uniform Hashing Assumption

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: My hash is uniform

Independent: All keys hash independently



Separate Chaining Under SUHA

Given table of size m and n inserted objects

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

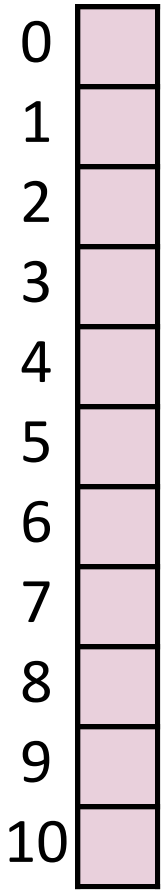
Separate Chaining Under SUHA

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