Your Next Week

Saturday June 20	Sunday June 21	Monday June 22	Tuesday June 23
9:00 AM **— DUE Class 24 Lab** **— DUE Class 24 Mock Interviews** **— DUE Class 25 Reading** — Class 25 — Interview Prep *MIDNIGHT* **— DUE Class 25 Learning Journal**	*MIDNIGHT* **— DUE CCW #2 Prep - Behavioral Questions** **— DUE CCW #2 Prep - Mock Interviews** **— DUE CCW #2 Prep - Star Methodology** **— DUE CCW #2 Prep - Winning the Interview** **— DUE Class 24-25 Feedback**	*6:30 PM* — Career Coaching Workshop 2A	*6:30 PM* **— DUE Class 25 Lab** **— DUE Class 26 Reading** — Class 26A
Wednesday June 24	**Thursday June 25**	**Friday June 26**	**Saturday June 27**
6:30 PM — Class 26B *MIDNIGHT* **— DUE Class 26 Learning Journal**	*6:30 PM* — Co-working		*9:00 AM* **— DUE Class 26 Lab** **— DUE Class 26 Code Challenge** **— DUE Class 27 Reading** — Class 27 — Interview Prep *MIDNIGHT* **— DUE Class 27 Learning Journal**

What We've Covered

Module 01

Javascript Fundamentals and Data Models

C01 — Node Ecosystem, TDD, CI/CD
C02 — Classes, Inheritance, Functional Programming
C03 — Data Modeling & NoSQL Databases
C04 — Advanced Mongo/Mongoose
C05 — DSA: Linked Lists

Module 02

API Servers

C06 — HTTP and REST
C07 — Express
C08 — Express Routing & Connected API
C09 — API Server
C11 — DSA: Stacks and Queues

Module 03

Auth/Auth

C10 — Authentication
C12 — OAuth
C13 — Bearer Authorization
C14 — Access Control (ACL)
C15 — DSA: Trees

Module 04

Realtime

C16 — Event Driven Applications
C17 — TCP Server
C18 — Socket.io
C19 — Message Queues
C20 — Midterms Prep

Midterms

Module 05

React Basics

C21 — Component Based UI
C22 — React Testing and Deployment
C23 — Props and State
C24 — Routing and Component Composition
C25 — DSA: Sorting and HashTables

Module 06

Advanced React

C26 — Hooks API
C27 — Custom Hooks
C28 — Context API
C29 — Application State with Redux
C30 — DSA: Graphs

Module 07

Redux State Management

C31 — Combined Reducers
C32 — Asynchronous Actions
C33 — Additional Topics
C34 — React Native
C35 — DSA: Review

Module 08

UI Frameworks

C36 — Gatsby and Next
C37 — JavaScript Frameworks
C38 — Finals Prep

Finals

Code Challenge 24 Review

Lab 24 Review

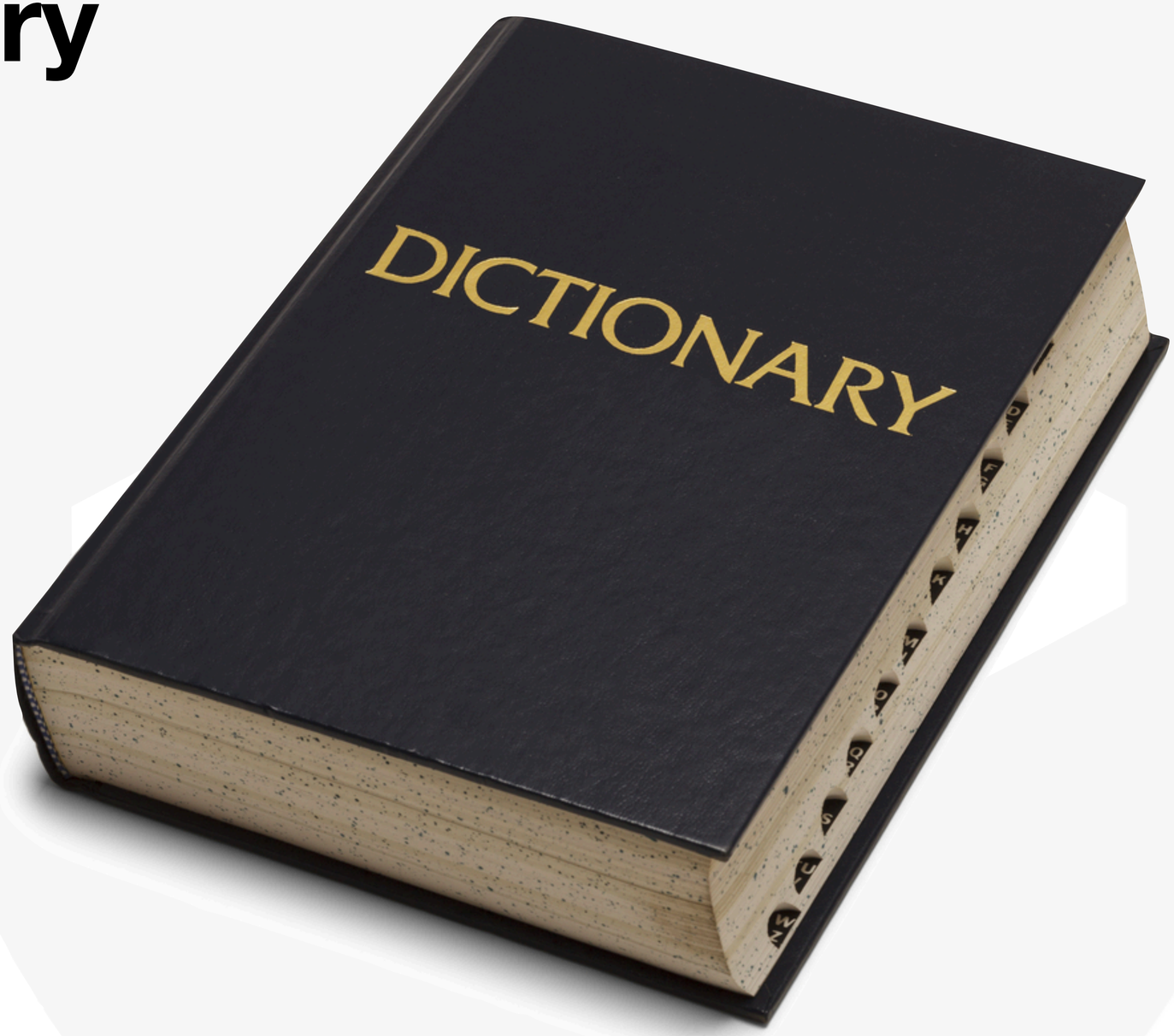
Class 24

DSA: HashTables

seattle-javascript-401n16

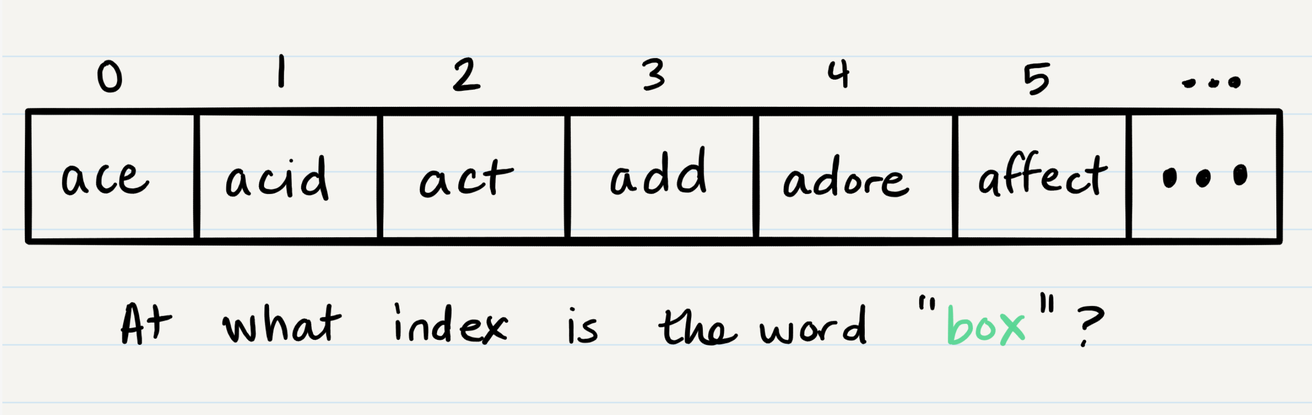
Storing a Dictionary

- I have a dictionary with definitions for each word
- I want to be able to *quickly* look up any word's definition
- What data structure should I use?



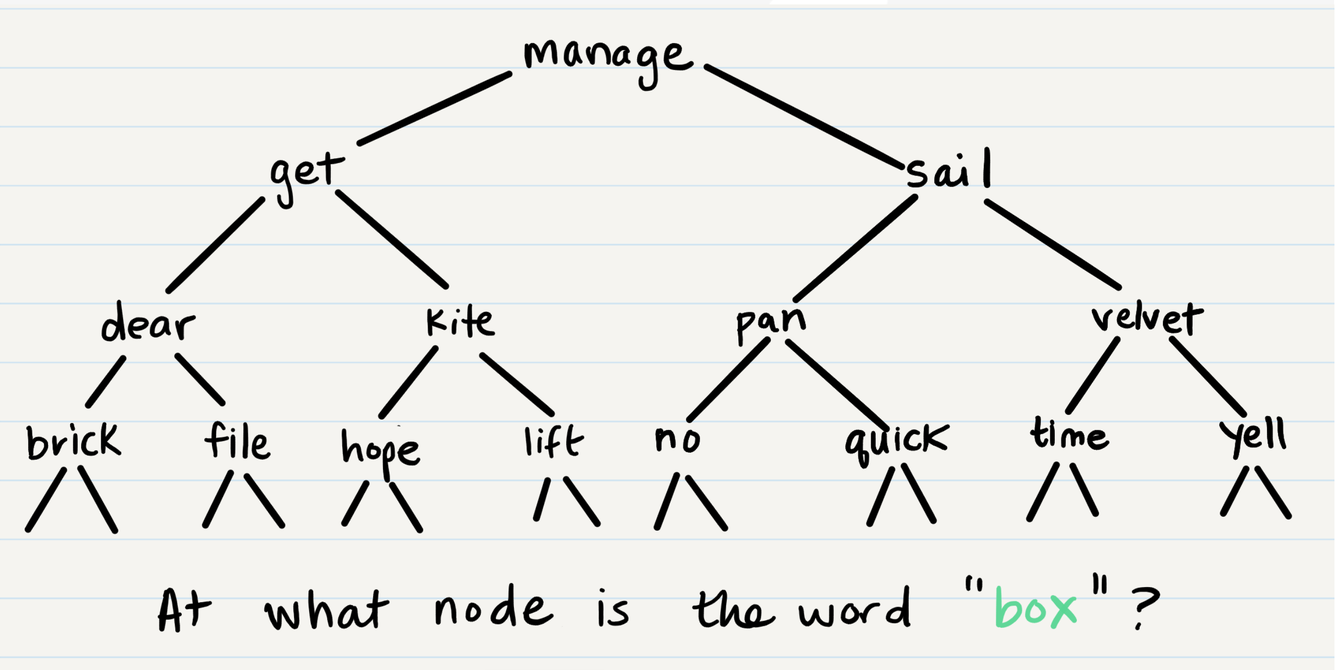
Array or Linked List

- Each time I see a new word in the dictionary, I push it to Array
- Sorting is possible (alphabetically)
- Searching is possible
  - But I still need to *search*
  - Can’t directly jump to the word I want
  - Worst case search is at least `O(logn)`



Stack, Queue, Tree

- Probably either the same or worse than an Array
- We can get a binary search tree at `O(logn)`
- Stacks and Queues are not really an option
    - You have to remove items to get to what you want
    - Otherwise, `O(n)`



Object

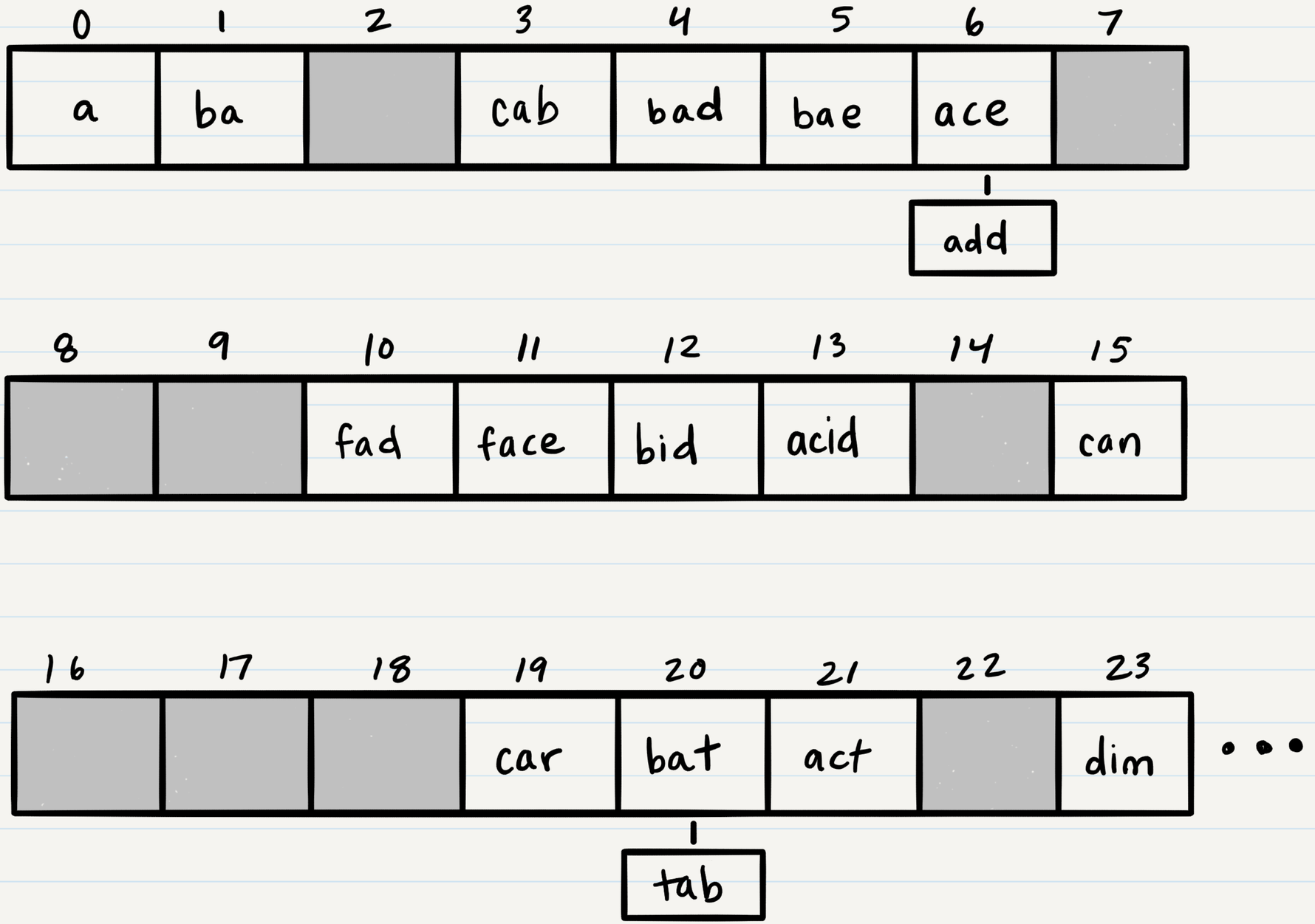
- Seems like a better solution
- We can have each word and definition stored as a key-value
- If we have a word with multiple definitions, store them as an array
- What if our key was something more complicated / an invalid key name?

```
const words = {
    ace: ['The best at', 'A playing card'],
    acid: 'A dangerous chemical',
    act: 'Pretend to be someone else',
    add: 'Combine two numbers into a sum',
    adore: 'Love something',
    ...
};

const box = words['box'];
```

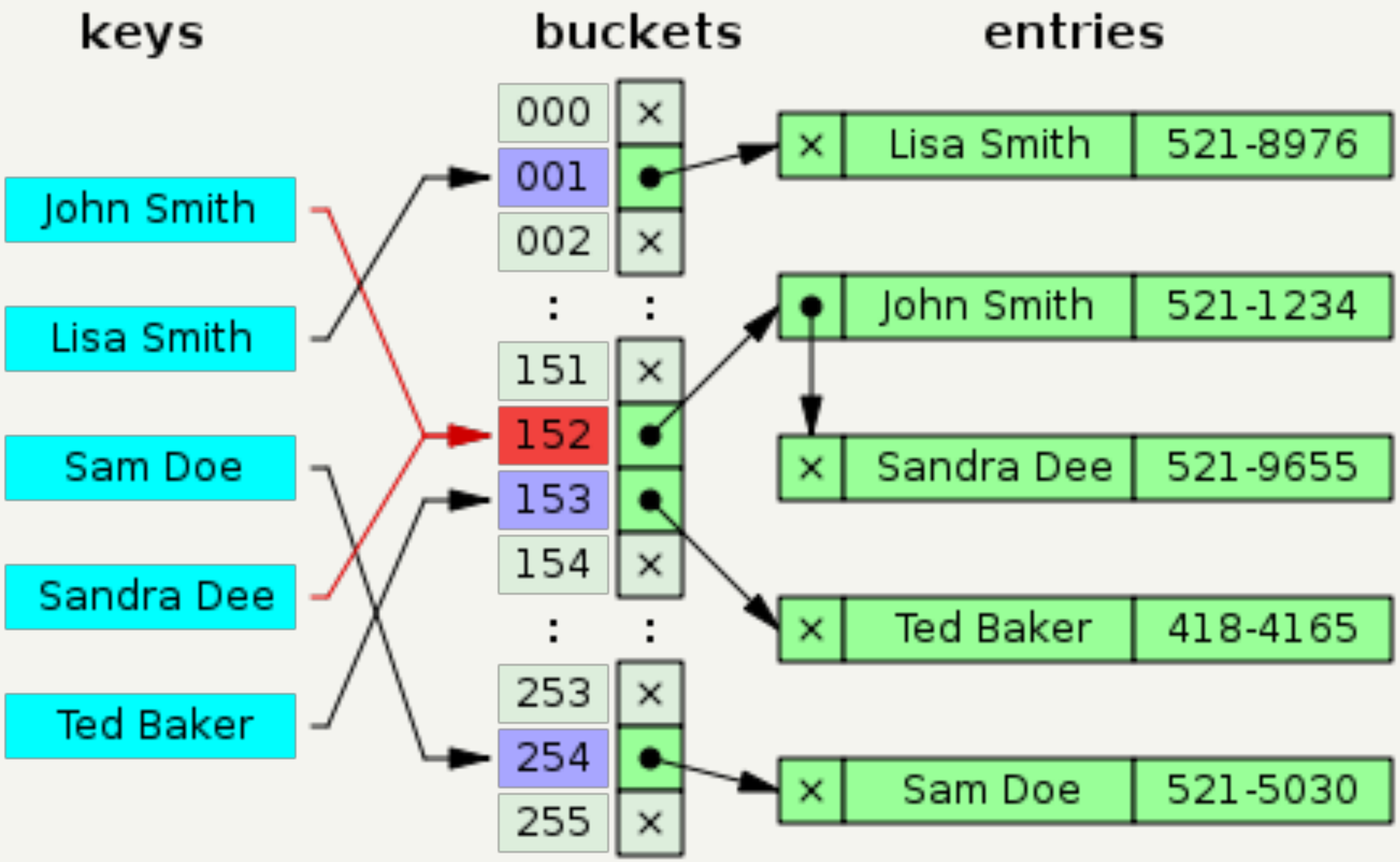
Something Smarter?

- What if I had a function that converted my words to the index of an array?
  - Set a number value for each letter: `a = 0, b = 1, c = 2,` …
  - For every word, sum up the number values:
    - `'ace' >> 0 + 2 + 4 = 6`
    - `'act' >> 0 + 2 + 19 = 21`
    - `'box' >> 1 + 14 + 23 = 38`
  - If there is a *collision*, at that index in the array create a linked list
    - `'bat'` and `'tab'` are both equal to the index `20`


0
1
2
3
4
5
6
7
a
ba
cab
bad
bae
ace
add
8
9
10
11
12
13
14
15
fad
face
bid
acid
can
16
17
18
19
20
21
22
23
car
bat
act
dim
tab

HashTables

- **HashTables** are large array-like structures
  - Essentially arrays, but indexes are called “**buckets**”, and each bucket is initialized to null
  - Why is this distinction important?
    - In some languages arrays take up more space than buckets
- Hash Tables have `insert` and `search` time complexity of `O(1)`!
  - Perfect **hash functions** never create a **collision**
  - If you have collisions, technically, `O(max collision length)`
  - Bad hash table can have `O(n)`, but most have `O(2)` or `O(5)`, which is about `O(1)`


keys
buckets
entries
John Smith
Lisa Smith
Sam Doe
Sandra Dee
Ted Baker
000
001
002
151
152
153
154
253
254
255
Lisa Smith
521-8976
John Smith
521-1234
Sandra Dee
521-9655
Ted Baker
418-4165
Sam Doe
521-5030

Lab 25 Overview