LEC 7

CSE 122

Nested Collections

Questions during Class?

Raise hand or send here

sli.do #cse122

Slido vote & chat with neighbors:

The weather is hot, what're your plans to cool off?

Instructor: Ivory & Colin

TAs: Connor
Wesley
Yang
Dani
Rohan

Agenda (1/4)

- Announcements
- Recap: Nested Collections
- Practice: Search Engines
- Practice: Social Network

Announcements

- Programming Assignment 1 (P1) due tomorrow, July 23rd 11:59pm!
- Quiz 1 on July 28th in your registered Quiz Section
  - Topics: (Reference Semantics), Stacks and Queues, Sets, Maps
  - Practice Quiz 1 available soon
- Resubmission Cycle 2 (R2) form released tomorrow, due July 28th by 11:59pm PT
  - Available assignments: **C0**, P0, C1
  - Reminder: to use a resubmission cycle you need to
    - (1) submit your work (big blue "Submit" button on Ed)
    - **AND** (2) fill out the resubmission form (linked from Ed + course calendar)

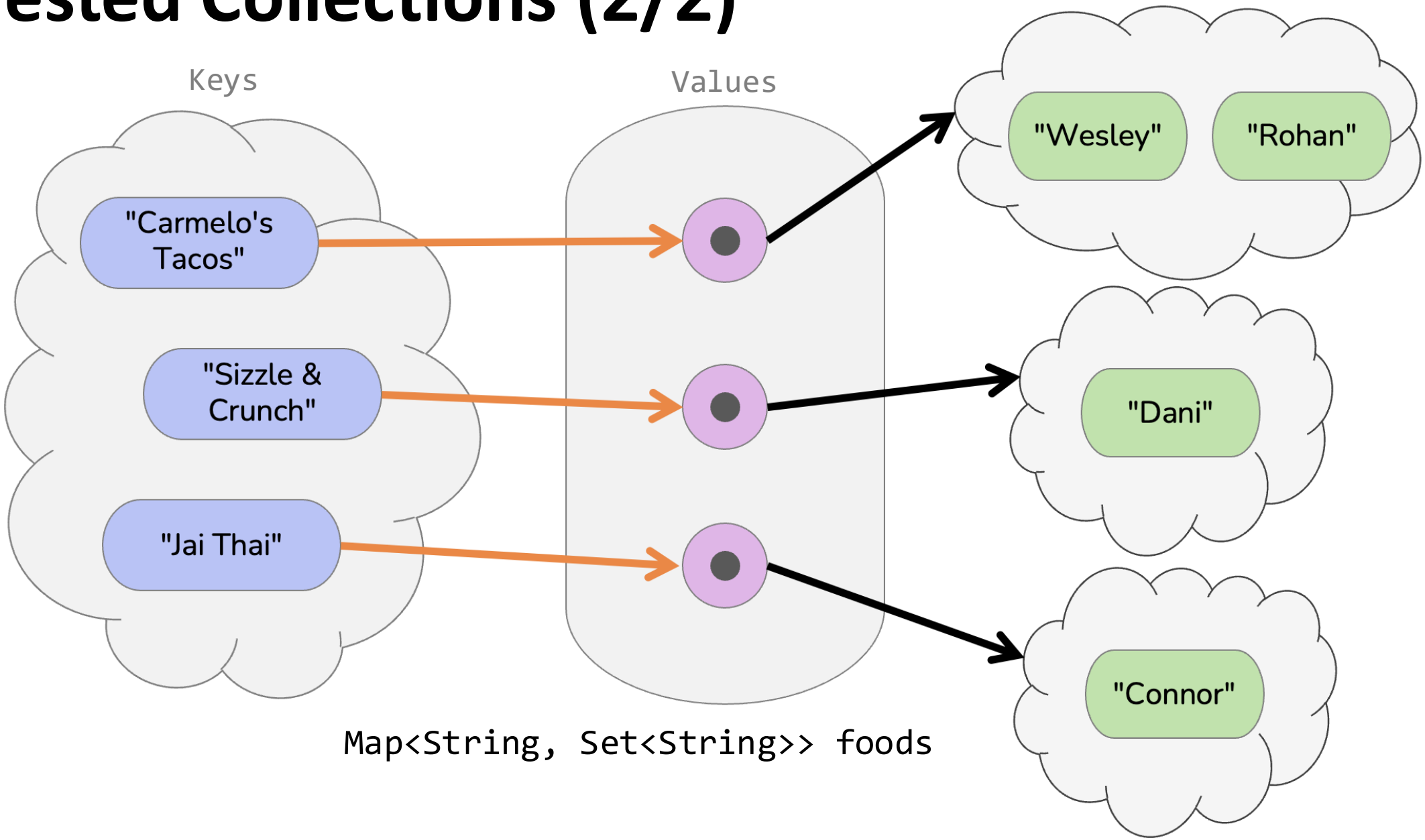
Agenda (2/4)

- Announcements
- **Recap: Nested Collections**
- Practice: Search Engines
- Practice: Social Network

Nested Collections (1/2)

- The values inside a Map (or any data structure!) can be any type, including more data structures
- Common examples:
  - Mapping: Section ➔ **Set of students** in that section
  - Mapping: Recipe ➔ **Set of ingredients** in that recipe
    (Or even Map<String, Map<String, Double>> for units!)

Nested Collections (2/2)




```
Map<String, Set<String>> foods
```

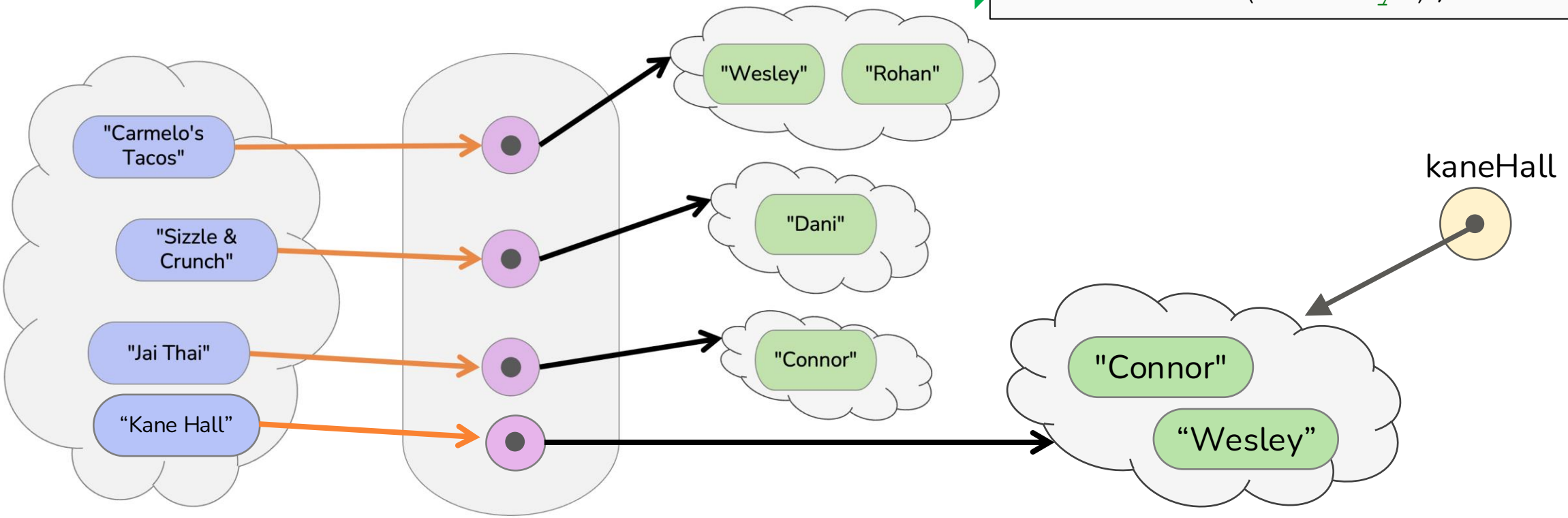
Updating Nested Collections

The "value" inside the Map is a reference to the data structure!

- Think carefully about number of references to a particular object

```
foods.put("Kane Hall",
        new HashSet<String>());
foods.get("Kane Hall").add("Connor");

Set<String> kaneHall =
        foods.get("Kane Hall");
kaneHall.add("Wesley");
```



Practice : Think

sli.do #cse122

Suppose map had the following items. What would its items be after running this code?

```
map: {"KeyA"=[1, 2], "KeyB"=[3], "KeyC"=[4, 5, 6]}
```

```
Set<Integer> nums = map.get("KeyA");
nums.add(7);
map.put("KeyB", nums);
map.get("KeyA").add(8);
map.get("KeyB").add(9);
```

A. `{"KeyA"=[1, 2], "KeyB"=[1, 2, 7], "KeyC"=[4, 5, 6]}`

B. `{"KeyA"=[1, 2, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}`

C. `{"KeyA"=[1, 2, 7, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}`

D. `{"KeyA"=[1, 2, 7, 8, 9], "KeyB"=[1, 2, 7, 8, 9], "KeyC"=[4, 5, 6]}`

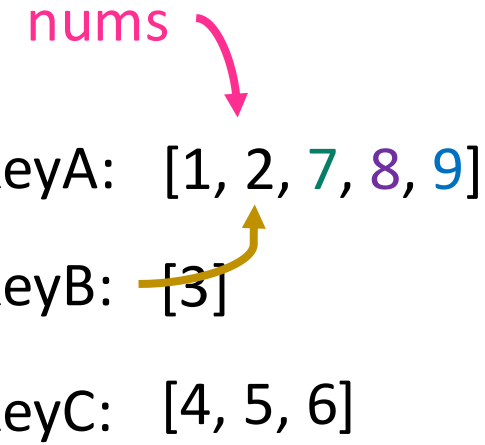
Practice : Pair

sli.do #cse122

Suppose map had the following items. What would its items be after running this code?

```
map: {"KeyA"=[1, 2], "KeyB"=[3], "KeyC"=[4, 5, 6]}
```

```
Set<Integer> nums = map.get("KeyA");
nums.add(7);
map.put("KeyB", nums);
map.get("KeyA").add(8);
map.get("KeyB").add(9);
```



A. `{"KeyA"=[1, 2], "KeyB"=[1, 2, 7], "KeyC"=[4, 5, 6]}`

B. `{"KeyA"=[1, 2, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}`

C. `{"KeyA"=[1, 2, 7, 8], "KeyB"=[1, 2, 7, 9], "KeyC"=[4, 5, 6]}`

D. `{"KeyA"=[1, 2, 7, 8, 9], "KeyB"=[1, 2, 7, 8, 9], "KeyC"=[4, 5, 6]}`

Agenda (3/4)

- Announcements
- Recap: Nested Collections
- **Practice: Search Engines**
- Practice: Social Network

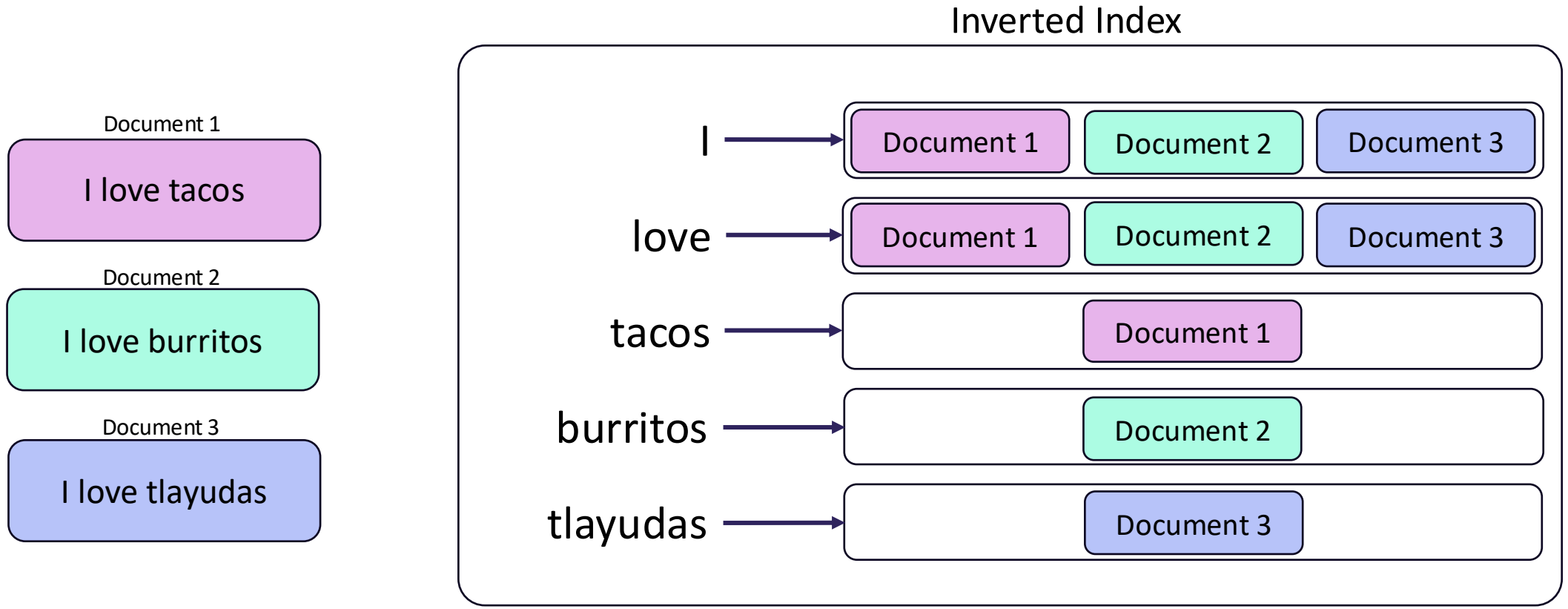
Background: Search Engines

- A **search engine** receives a **query** and returns a set of relevant **documents.** Examples: Google, Mac Finder, more.
  - Queries often can have more
- A search engine involves two main components
  - An **index** to efficiently find the set of documents for a query
    - Will focus on “single word queries” for today’s example
  - A **ranking algorithm** to order the documents from most to least relevant
    - Not the focus of this example

- Goal: Precompute a data structure that helps find the relevant documents for a given query

Inverted Index

- An **inverted index** is a Mapping from possible query words to the set of documents that contain that word
  - Answers the question:
    "What documents contain the word 'burritos'?"



Ranking Results

- There is no one right way to define which documents are “most relevant” There are approximations, but make decisions about what relevance means
- Idea 1: Documents that have more hits of the query should come first
  - Pro: Simple
  - Con: Favors longer documents (query: “the dogs” will favor long documents with lots of “the”s)
- Idea 2: Weight query terms based on their “uniqueness”. Often use some sort of score for “Term Frequency – Inverse Document Frequency (TF-IDF)
  - Pro: Doesn’t put much weight on common words like “the”
  - Cons: Complex, many choices in how to compute that yield pretty different rankings
- Idea 3: Much more! Most companies keep their ranking algorithms very very secret ☺

Agenda (4/4)

- Announcements
- Recap: Nested Collections
- Practice: Search Engines
- **Practice: Social Network**