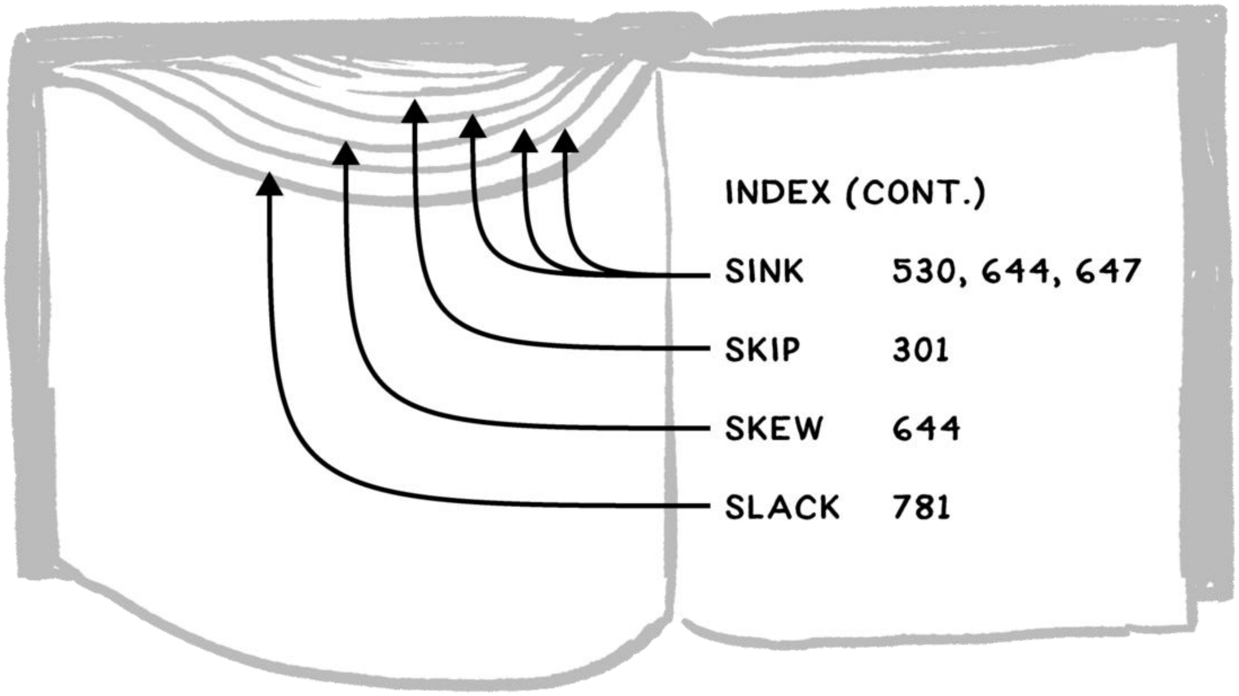
Introducing Indexes

By

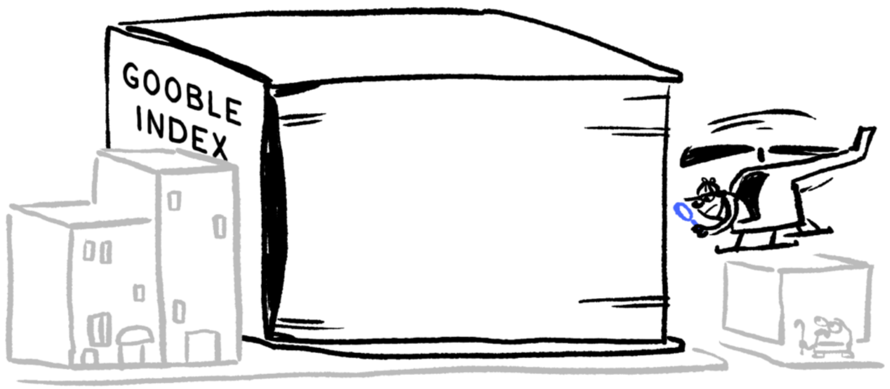
Dr Syed Khaldoon Khurshid

- In searching a word in a document that you could get away with searching the simplest possible way: reading through the whole document from beginning to end, looking for the right words.
- That's *not* how the any search engine you used worked, and it wouldn't be remotely possible for Google to work this way!
- But we will develop our understanding of indexes by using this simple approach.

- **There's "one weird trick" behind any search engine: building and using an *index*.**
- A book's index allows you to access a book in an "inside out" manner. Instead of scanning the book to find references to a topic, the index is an alphabetical listing of topics that explains where in the book that topic appears.



- Even huge internet search engines containing many trillions of documents ultimately **rely on storing all that information in <u>an index</u> that is stored in a computer.**
- **The most fundamental operation on an index is *looking up a word*. If the index is built just like a book's index, then a word can be found by looking in the middle of the index, and then (depending on the word) looking at the word in the middle of the first half or the middle of the second half.**
- By always looking in the middle, you can look up a word in an index containing trillions of web pages by just checking a few dozen entries! That's the magic of **binary search!**

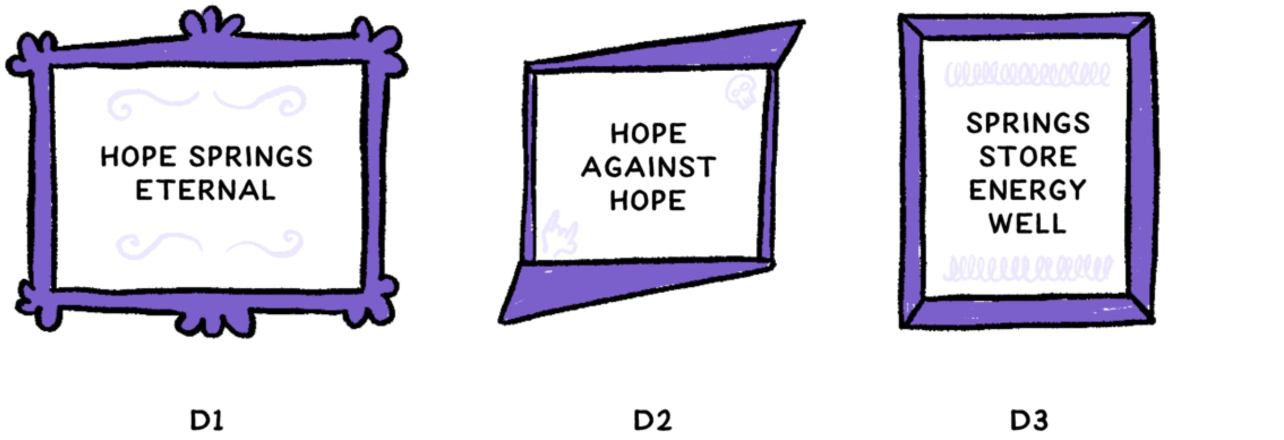


- In a book, an index is a static document, created one time for one book. There's no space in a book's index to add new topics. That won't work for a search engine: the web is changing constantly!
- A search engine's index has to support one other critical operation: ***adding a new word***. You can think of an index that supports addition as a huge organized wall full of sticky notes. It's always possible to **add** a new sticky note.

- In further lecture, you’re going to assume that you have an index that supports these two critical operations:
- **quickly looking up words and**
- **quickly inserting new words.**
- Based on these two operations, you’ll learn **how to build a index** for search engine. Search Engine that efficiently supports complex queries over trillions of documents.

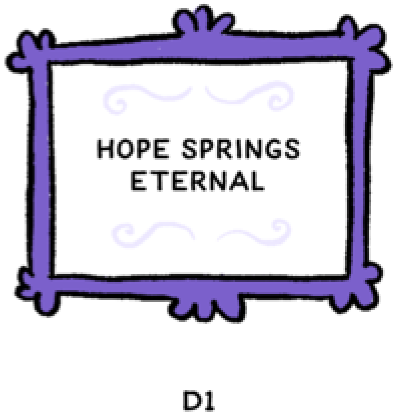
Building An Index

- But to get big, **you'll start very, very small**. In this module, you'll build an index that lets you search these three "documents":



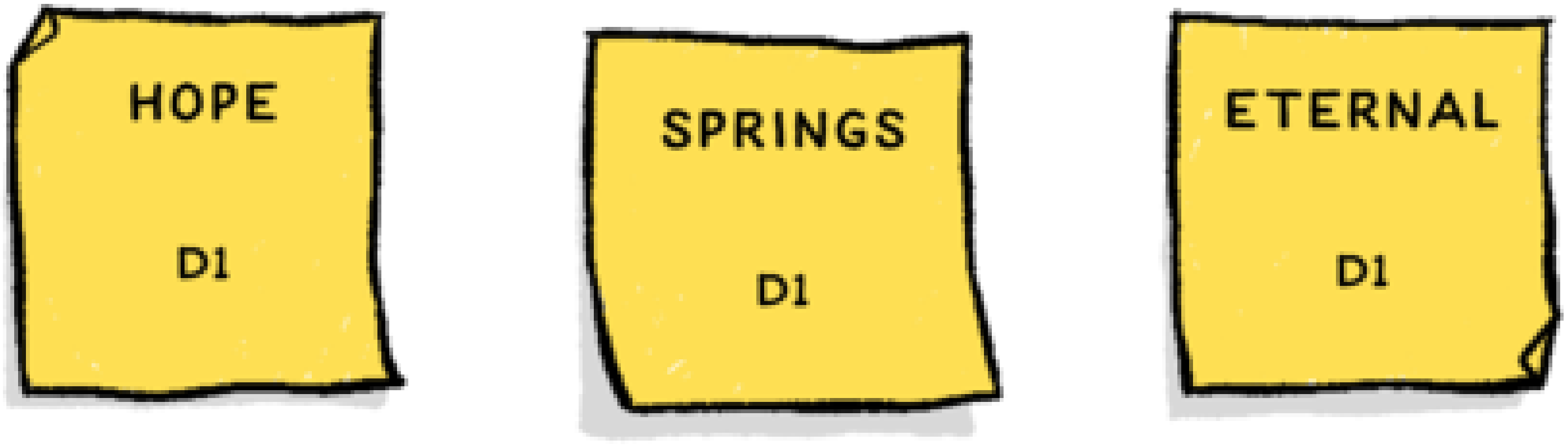
- This collection contains 3 documents, with a total of ten words.

- Your goal is to create an easily searchable record of where words appear. In addition to your documents, you have a pad of sticky notes. Start with the first document:

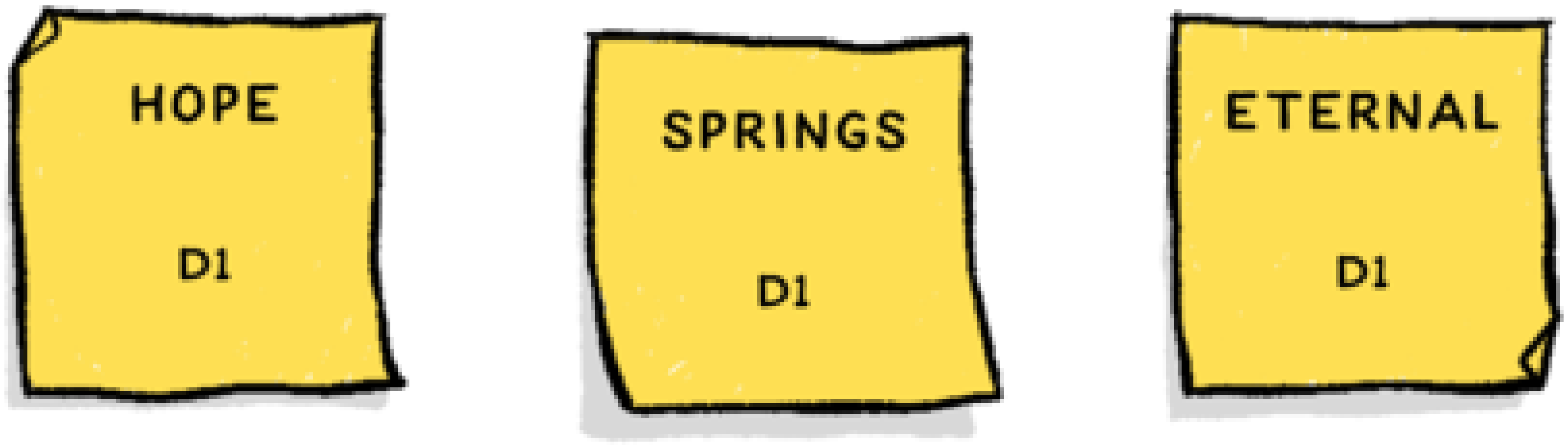


- Ignoring punctuation and capitalization, the first step is to write the first word, “**Hope**" on a sticky note, along with the document identifier, **D1**:

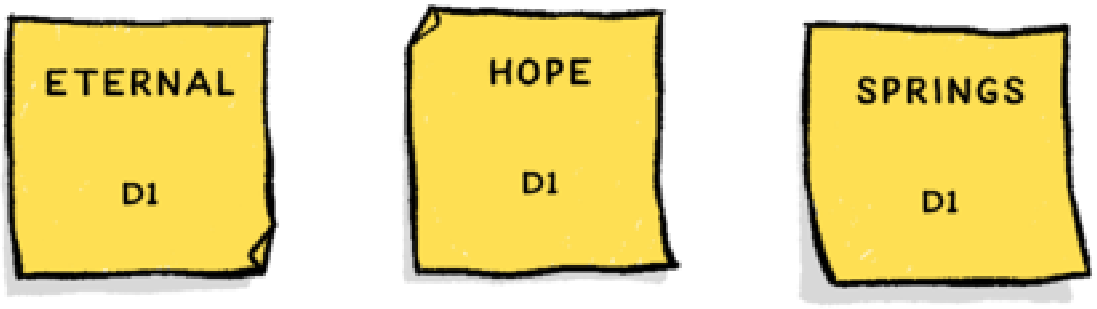
- Repeat this process with the second and third words in the document, and your wall will look like this:



- Your goal is to use these sticky notes to quickly look up individual words. Which of the following is a good way to organize the notes to facilitate this? (Remember that sticky notes can be moved if and when necessary.)



- After putting the sticky notes in **alphabetical order**, the wall looks like this:

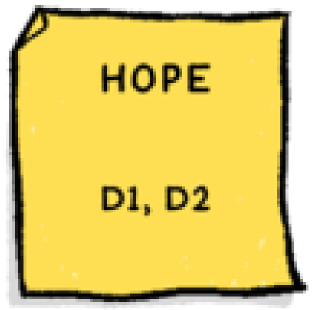


- Some information has been lost in representing the first document this way. Which of the following alternative documents would result in the same arrangements of sticky notes? (Select *all* that apply.)

- Now you can start on the second document.

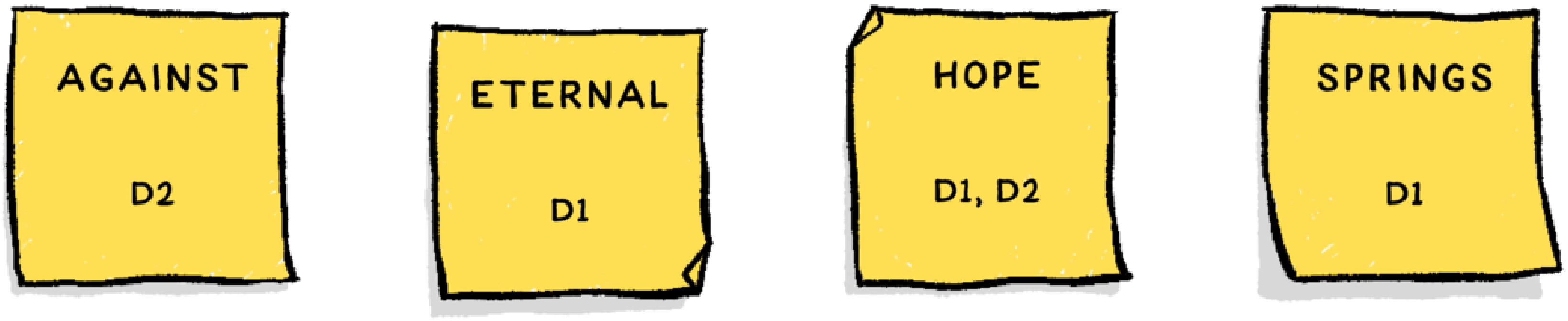
HOPE
AGAINST
HOPE
D2

- The first word is a word you've encountered before: "hope".
- You already have a sticky note for "hope" on the wall. Instead of adding a new sticky note, you use the same one and just add the document number D2 to the end of it, like this.



HOPE
AGAINST
HOPE
D2

- The next word in D2 is "against." You'll need a new sticky note for that one:



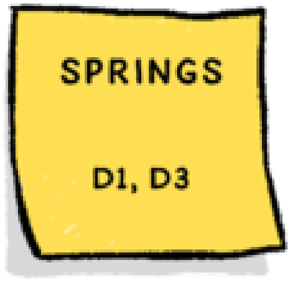
- The last word in D2 is "hope" again. The sticky note for "hope" already has D2 on it, so you have a choice to make!
- You could either write 'D2' on the sticky note again or not. If you choose not to write down D2 a second time on the "hope" sticky note, and if you do the same thing for future repeated words, what information will be lost?

- Continue on to the third document.

SPRINGS STORE ENERGY WELL

D3

- The first word is "springs” In context, this has a different meaning from the one you've encountered before: it's a **noun** describing a bouncy piece of metal, not a **verb** meaning “originates or arises.”
- An automated indexer cannot make that distinction (at least at this stage). For the purposes of the index, these are treated as the same word and given only one sticky note.



- This is really how it must be. If you search for **"jaguar"** in Google, the search engine cannot possibly know whether you meant the car or the wild cat, and it must therefore find pages about both.
- (In reality, Google does have sophisticated algorithms to use external information such as you and other people's search histories to try to predict what you are probably looking for, but a discussion of that is outside the scope of our current studies.)

- You've now built an index for a small database. The sticky notes and the wall are, of course, a metaphor for something more sophisticated happening within a computer system. **The basic principles, however, remain the same**!
- An index is a way to keep records for each word in the database so that individual words can be easily looked up and the record points to all the documents containing the word.

An Algorithm for Indexing

- Informally, you've seen how to create a search engine out of sticky notes. In this exploration, you're going to take a closer look at how this process can be turned into a precise algorithm.
- Remember that the data structures behind a search engine's index will support two kinds of operations quickly: **lookup** and **insert**. In our sticky note metaphor, these two operations look like this:
- **Lookup**: find a sticky note with a certain word on the wall, or determine that no sticky note exists.
- **Insert**: place a sticky note on the wall so that it can be found later.
- There's a third operation you'll want to pay attention to. When you have found a sticky note on the wall or created a new one, you can **write** to it to record new information.

HOPE SPRINGS ETERNAL

D1

HOPE AGAINST HOPE

D2

SPRINGS STORE ENERGY WELL

D3

Here is an algorithm for building a search engine's index. For each document in our database and for each word in that document, run the following procedure

```
lookup the word

if there is already a sticky note
for this word on the wall:
    if the current document's id
    is not on the sticky note:
        write the document's id on
        the sticky note
else:
    get a new sticky note
    write the word and document
    id on the sticky note
    insert new note on the wall
```

- Why count the number of lookups and inserts performed by the index-building algorithm at all?
- One reason is that understanding the number of operations an algorithm performs is a good way to better understand the algorithm.
- Another reason is that the **lookup**, **insert**, and **write** operations, while hopefully quite fast, take up most of the time of creating an index. This is true whether you're building the index with powerful computers or with sticky notes on your wall.
- Counting the number of operations enables us to ignore the *specifics* of how we're putting together the search engine's index, while still understanding important details about how much time it will take to build the index.

Case Study I for IR: Design a Simple Document Search Engine

Design a Simple Document Search Engine

- Creating a simple document search engine for information retrieval involves several steps. Here's a step-by-step approach:

- **Step 1: Define Your Goals**

  Understand what you want your search engine to do. Do you want to search for specific words in documents? Do you want to find documents related to certain topics?

- **Step 2: Gather Documents**

  Collect the documents you want to search through. These could be text files, PDFs, or web pages. Make sure they are in one place, like a folder on your computer.

- **Step 3: Choose a Programming Language**

  Decide on a programming language you are comfortable with. Python is a good choice for beginners.

- **Step 4: Create a Program**

  Write a computer program in your chosen language. This program will help you search through your documents.

- **Step 5: Read and Index Documents**

  Teach your program to read the text in your documents. It needs to understand what's inside them.

  Create an index. This is like a list of words and where they appear in each document. It helps your search engine find things faster.

- **Step 6: Build a Search Function**

  Write a search function in your program. This function takes a user's query (the words they want to search for) and looks it up in the index you created.

  When a user searches for something, your program will find matching documents.

- **Step 7: Display Results**

  Show the search results to the user. You can display a list of documents that match their query.

- **Step 8: Test Your Search Engine**

  Try different searches to make sure your search engine works correctly. Make improvements if needed.

- **Step 9: Make it User-Friendly**

  Add a simple user interface. It could be a web page or a command-line interface where users can type their queries.

- **Step 10: Refine and Optimize**

  Keep improving your search engine. Make it faster and more accurate.

  You can add features like ranking results based on relevance.

- **Step 11: Document Your Work**

  Write down how your search engine works, so others can understand and use it.

- **Step 12: Share Your Search Engine**

  If you want others to use your search engine, make it accessible to them. You can host it online or share the program.

Note:

- Remember, this is a simple search engine, and there are more advanced features you can add as you become more comfortable with the basics. Start small, learn, and build on your project over time.

Any
Questions