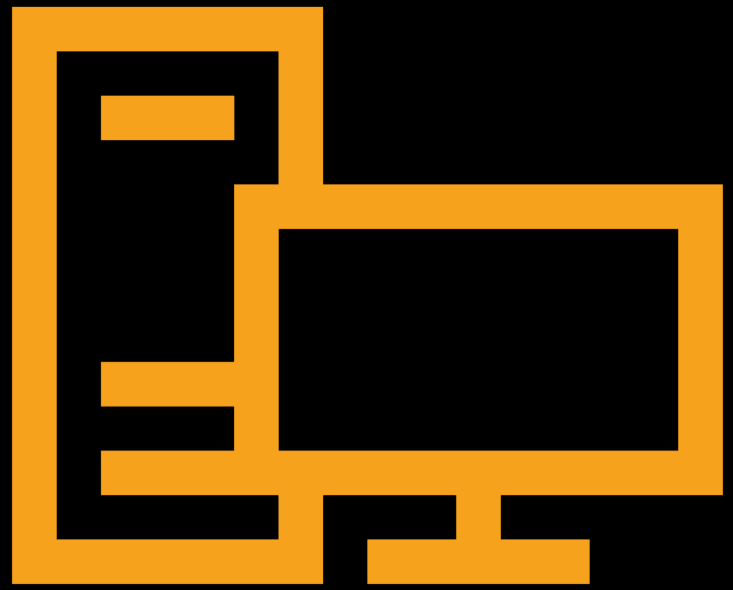
APPLIED ALGORITHMS

Lecture 10: Working remotely, Bloom Filters, and Cuckoo Filters



ADMIN

- Mini-midterm due 10PM today
- Assignment 3 will be optional; I'll release it next week
  - Due over spring break

- Questions?

ONE MINI-MIDTERM CLARIFICATION

- Question 7 asks about sorting
- I'm referring to the $O(n^2)$ algorithm (not the space-efficient one)

WORKING REMOTELY

- Upsides and downsides to this in this class
  - Assignments likely don't change much
  - This course doesn't have a textbook---that's a problem
- Unclear what will happen with lectures
- It seems likely to me that I'll need to start making lecture notes regardless

- Let's talk about assignments

RESOURCES

Resources for remote work:

https://www.cs.williams.edu/system/

Commonly-used software:

https://www.cs.williams.edu/system/softwareused.html

Let me know if you have questions!!!

WHAT YOU NEED

- You need a way to code.
  - Text editor
  - Way to make
  - Way to run your program
  - Access to git somehow
- You also need a way to access the lab computers for testing

CODING

- If you don't have a semi-decent setup on your own computer, now is the time
- make is trivial to get working on mac/linux. There are options for Windows (which I haven't used)
- Get your favorite IDE working; I can help.
- **You need to have git working on your computer.** Not copy-pasting into github
  - This should be reasonably straightforward to install for mac, linux, and windows.
  - Resources page has links that walk you through it

GIT

- You need to be able to use command-line git on the lab computers even if your IDE takes care of it

- `git clone nameofrepo` to create a folder with an external repo called "nameofrepo"
- `git pull` to pull remote changes into an already-existant repo
- `git add blah.c` adds the file blah.c (starts tracking changes---if you don't add it won't be included)
- `git commit –m "blah"` commits with message "blah"
  - Commit is LOCAL to your computer. Does not put it on the repo
- `git push` pushes the changes (I can see them after this)

EDITOR AND RUNNING

- I think everyone has this up on their own computer? Let me know if not

SSH

- ssh is an incredible protocol and you should take this opportunity to learn it if you haven't
- Secure shell—basically, you can run command line programs on a computer remotely
- Easy to use: `ssh sam@devon.cs.williams.edu`
- You can very literally use a lab computer from your home
- YOU NEED A VPN TO DO THIS!
- Download the VPN right now. This is why you brought your laptops.
- https://oit.williams.edu/help-guides/wifi-and-wired-connections/vpn/
- (Or google "Williams VPN")

SSH

- Already installed on Mac and most Linux I think
- For Windows: download Putty
  - Putty is the worst but it's what's available
  - Supposedly you can also use git to do ssh?
  - Also: apparently Putty is better than it used to be

- If you just want to log in to lab computers to git pull, make, and run, Putty is definitely perfectly fine. It's just not very nice to use for extended periods of time

SSH

- Upsides:
  - Literally program on a lab computer
  - access to lots of great unix programs
  - You can access your data from anywhere on any machine so long as you have ssh. (You can literally ssh from your phone)
- Downsides:
  - No GUI!!!
  - No (or extremely limited) mouse support, limited colors, etc.

ACTUALLY CODING ON LAB COMPUTER

- You would need a text-based editor (since ssh is purely text-based)
  - emacs (very extensible; taught in the department. Some simple operations are surprisingly difficult. Also powerful enough to get you into trouble)
  - nano (relatively very easy to use)
  - vim (high learning curve, but extremely rewarding. Very easy to get yourself in trouble.)
    - I am always happy to answer questions about vim
- All of these already exist on the computers
- Coding in vim or emacs is a real thing professionals do (probably not through SSH, but proficiency is a really valuable skill)

ACTUALLY CODING ON A LAB COMPUTER

- This actually is doable. When I was in college all coding was through SSH (putty) using emacs
- I learned a lot from it
  - Valuable skills
  - Also a deep hate for putty and emacs

EXAMPLE WORKFLOW FROM HOME

1. Get a working program on your computer
2. Upload it to the repo using git
3. Login to a lab computer using ssh
4. Git pull your changes (you already set up the repo using git clone)
5. Run make, run the program

UNIX ACCESS

- Only having Windows is a big disadvantage (coding through ssh, or coding locally and transferring files using git, helps alleviate this. FWIW: ssh is way nicer through unix-based terminals)
- Some ways to get unix running on your computer other than SSH:
- Dual boot. (This is easy. Back up your data first. Mary has ubuntu on a thumb drive. Downsides: takes up room, problems do affect your real computer)
- Virtual Machine. We have access to vmware (talk to Mary); Oracle VirtualBox is free. Sandbox--very safe; can throw out and redo if things do wrong. Downsides: speed

ANY QUESTIONS OR COMMENTS?

Please (please please) get as much of this set up as possible before you leave so that you can get whatever help you need.

I am willing to work with you. Also, send me emails if you need it.

COMPRESSION

- CS361: You can't compress data in the worst case
- Today we will!
- Bloom filters: a special kind of compression that works in the worst case
- Essentially: if we're compressing we're losing information. Bloom filters lose information in a specific (and extremely useful) way

BLOOM FILTER

- Store a set S of n elements
- Only answer membership queries
  - For a query q, is q a member of S? In other words: is $q \in S$?
- Build a compressed data structure
- For any query where $q \in S$, the Bloom Filter gives the correct answer always
- For any query where $q \notin S$, the Bloom Filter makes a mistake with probability ϵ
  - With probability ϵ, the Bloom Filter says "yes, $q \in S$"
  - With probability $1 - \epsilon$, the Bloom Filter correctly says "no, $q \notin S$"

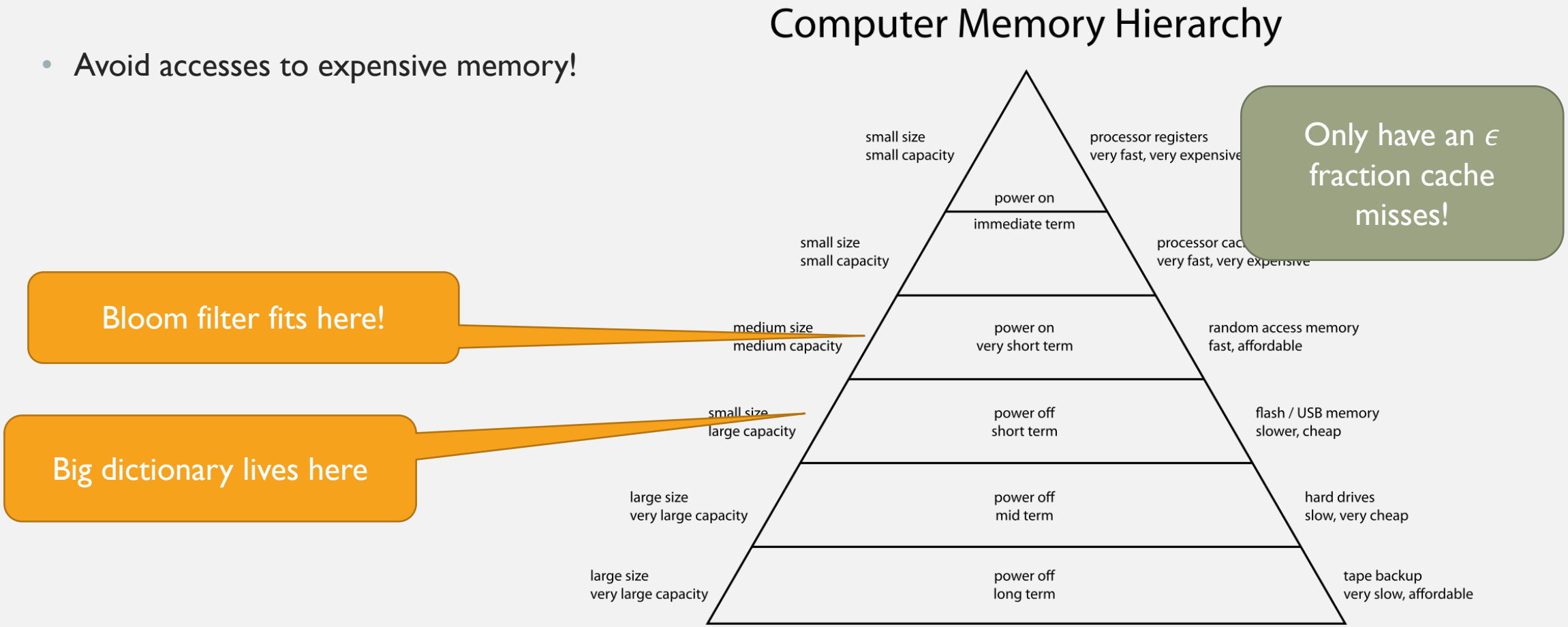
"No false negatives"

BLOOM FILTER: WHY?

- Invented in 1970
- Original motivation: irregular hyphenation rules
- Want to spellcheck a text document
- For each word, does it have irregular hyphenation rules?
- In 1970, storing a dictionary of all English words was very expensive!
- Most words don't have irregular hyphenation rules
- Idea: Bloom filter can "filter out" most words; don't need to query the dictionary

BLOOM FILTER USE CASE #1

- Avoid accesses to expensive memory!



BLOOM FILTLER USE CASE #2

- May not have whole set---maybe all we want is approximate answers to queries!
- Early "approximate" spell checkers (might miss some misspellings)

ACTUAL USES

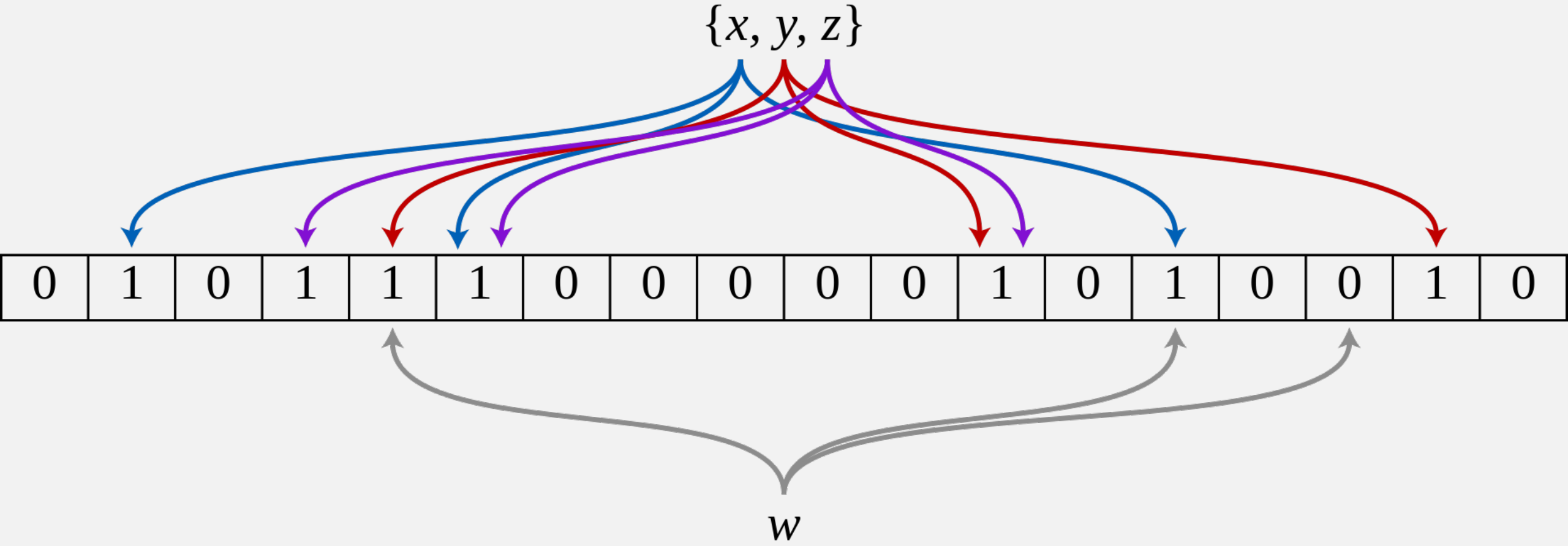
- Storing malicious URLs (Google Chrome)
- Akamai uses for caching (only want to cache things seen twice)
- Bitcoin, Ethereum (this is how you know it's important)
- Can help with brute force (i.e. crypto)
- Databases use extensively
  - Useful for things like checking membership before insert
- Also theoretical uses! Count number of items in set, etc. And inspired other data structures…

HOW IT WORKS

- Bloom filter consists of a table of $1.44n \log_2 1/\epsilon$ bits
- We will use $\log 1/\epsilon$ hash functions
- To put an item x in the bloom filter:
  - For each hash function h_i, set $h_i(x) = 1$

Can we have a false negative?

- How can we answer a query q?
  - For each hash function h_i, if $h_i(q) = 0$ answer "$q \notin S$"
  - If all hash functions have $h_i(q) = 1$, answer "$q \in S$"



Bloom filter with 18 bit positions and 3 hash functions. x, y, and z are in the set; w is a query

HOW IT WORKS

- I won't go over the false positive probability in detail (doing it better than this slide is fairly easy; doing it really properly is a lot of work)
- Short version:
- With k hash functions and a table of size m, each bit is 1 with probability $\sim e^{-kn/m}$
- Plugging in $k = \log_2 1/\epsilon$ and $m = 1.44n \log 1/\epsilon$, we get probability $\sim e^{-1/1.44} = 1/2$

Need to take my word for it, but this is relatively simple probability

- If each bit is 1 with probability $1/2$, then what is the probability that all $\log\_2\, 1/\epsilon$ bits are 1?
- ϵ! So we're done

LOWER BOUND

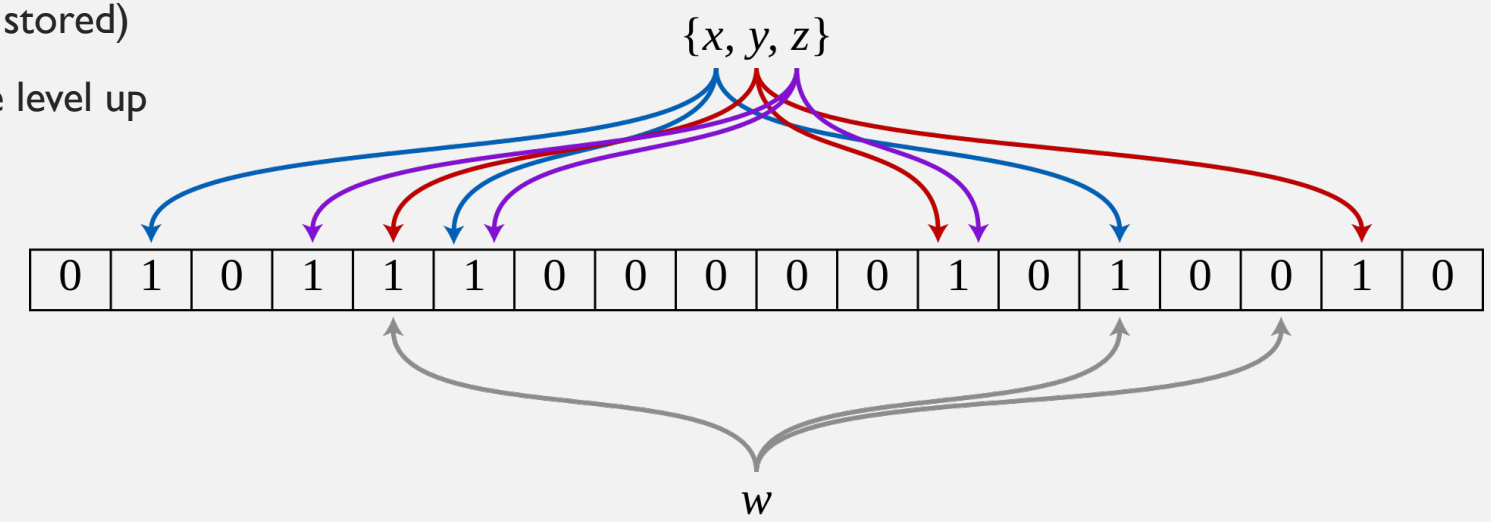
- Carter et al (1978): need $n \log 1/\epsilon$ bits at least
- Lovett and Porat (2010): need $> n \log 1/\epsilon$ bits if want to insert and delete

INSERTING AND DELETING IN A BF

- Can we insert?
  - Yes! If there's space
  - False positive probability will eventually start getting terrible
- Can we delete?
  - No, not even once
  - (Ways around this, like "counting bloom filter"- store actual counts at each position)

WHAT'S SLOW ABOUT A BF?

- Hashing can be pretty slow!
- Need to hash a decent number of times, especially for small ϵ
- Also need to store a ton of hashes (which is a pain)
- Terrible cache efficiency
  - *Within* the L2 cache (or wherever it's stored)
  - BF just moves the problem one cache level up



CUCKOO FILTER

- "Cuckoo Filter: Practically better than Bloom" (Fan et al. 2014)
- Uses cuckoo hashing to build a better Bloom Filter
- Better space usage (better constant)
- Faster lookup, better cache efficiency

CUCKOO FILTER

- Two hash functions h_1 and h_2, map to one of m slots
- Also a *fingerprint* hash function f which maps each element to $\log_2 1/\epsilon$ bits
- Table consisting of $m = 2n$ "slots", each of which is $\log_2 1/\epsilon$ bits

- Our analysis will assume that these are all *perfectly* random
- (As we saw yesterday, murmurhash essentially meets this)

- Draw on board

CUCKOO FILTER: HOW IT WORKS

- To insert x:
- Check slot $h_1(x)$ and see if it contains another element's fingerprint. If not, store $f(x)$ there.
- Otherwise, check slot $h_2(x)$ to see if it can store $f(x)$
- If both slots are full, cuckoo!
  - Kick current element out, and put it in its other slot

- Invariant: if $x \in S$, then either $f(x)$ is stored in slot $h_1(x)$, or $f(x)$ is stored in slot $h_2(x)$

- Example on board

CUCKOO FILTER: HOW TO QUERY

- How can we query an element q?
- Check if $f(q)$ is stored in slot $h_1(q)$. If so, return $q \in S$
- Check if $f(q)$ is stored in slot $h_2(q)$. If so, return $q \in S$
- Otherwise, return $q \notin S$

- If the correct answer is "$q \in S$", are we always correct?

CUCKOO FILTER: FALSE POSITIVES

- Let's say we're querying $q \notin S$. What is the probability the cuckoo filter returns $q \in S$?
- First: (background) Union bound
  - For two events A and B, the probability that A OR B happens is at most Probability(A) + Probability(B)
  - ALWAYS true.

CUCKOO FILTER: FALSE POSITIVES

- Let's say we're querying $q \notin S$. What is the probability the cuckoo filter returns $q \in S$?
- Break it down: look at one hash at a time
- There is a false positive for h_1 if:
  - Some x has its fingerprint stored in $h_1(x)$, AND
  - $f(x) = f(q)$

- Note: these events are independent if $q \neq x$

CUCKOO FILTER: FALSE POSITIVES

- There is a false positive for h_1 if:

- Some x has its fingerprint stored in $h_1(q)$
  - Since half the slots store elements, this occurs with probability $1/2$
- $f(x) = f(q)$
  - Since the fingerprint is $\log_2 1/\epsilon$ bits, what is the probability of this?
  - $1/2^{\log 1/\epsilon} = \epsilon$
- Probability of both = $\epsilon/2$ (because they are independent)

CUCKOO FILTER: FALSE POSITIVES

- Probability of false positive for h_1 is $\epsilon/2$
- Probability of false positive for h_2 is $\epsilon/2$
- By union bound: probability of a false positive = probability of either = $\frac{\epsilon}{2} + \frac{\epsilon}{2} = \epsilon$
- Done! (woo)

CUCKOO FILTER: FILLING IN GAPS

- How do we insert again?
- What's wrong with that?
  - Answer: we don't know the other hash!

- Solution: define $h_2(x) = h_1(x)\ \hat{}\ f(x)$
  - Since $a\ \hat{}\ b\ \hat{}\ b = a$, this means that $h_1(x) = h_2(x)\ \hat{}\ f(x)$
  - Xor slot with fingerprint to get the other slot
- Good news: this means that we can insert elements into the cuckoo filter!
- Bad news: kills our analysis since h_1 and h_2 are independent (generally OK in practice though)

CUCKOO FILTER: FILLING IN GAPS

- How much space do we use?
  - $2n$ slots, each of size $\log 1/\backslash\text{epsilon} = 2n \log\_2 1/\epsilon$ bits
  - This is WORSE than a Bloom Filter!
- Solve using bins!
- Have $m = 1.05n/4$ bins, each consisting of 4 slots
- Hash functions $h_1(x)$ and $h_2(x)$ map to one of m bins
- Store the fingerprint of x in an empty slot in bin $h_1(x)$ or bin $h_2(x)$

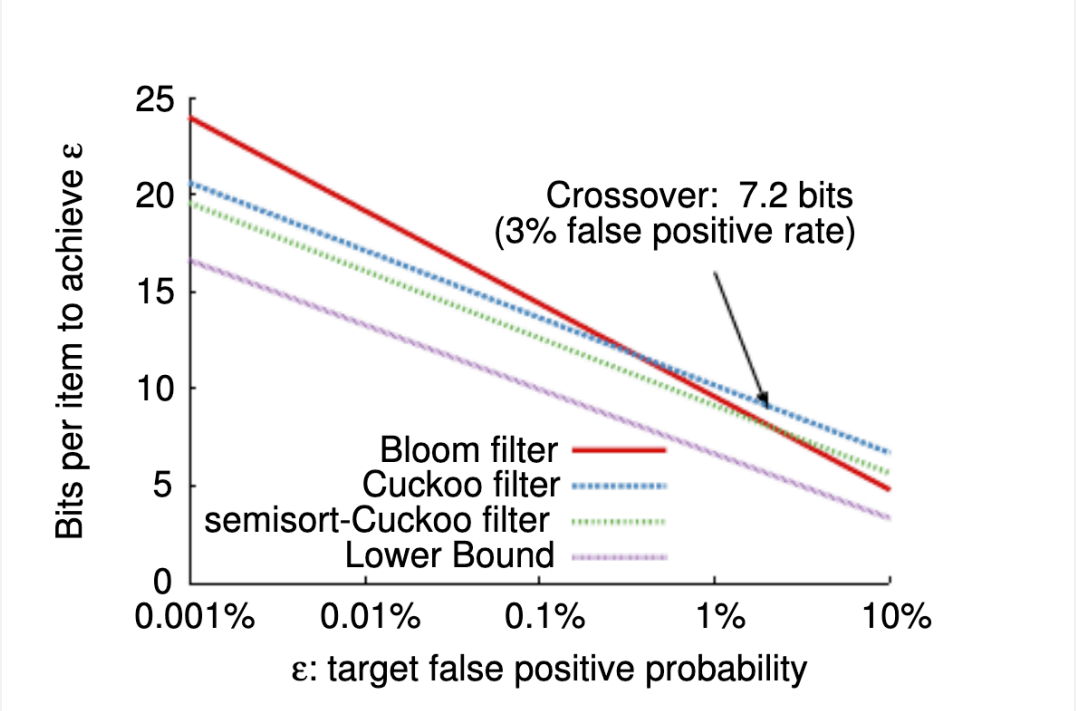
CUCKOO FILTER: FILLING IN GAPS

- It turns out we still get good performance (few swaps, rebuilds are rare)
- Disadvantage: need our fingerprints to be larger (since there are up to 8 elements in these slots; each of these elements can share a fingerprint with our query)
- Fingerprint of length $\log_2 8/\epsilon$
- Total space: $1.05n \log 8/\epsilon = 1.05n \log 1/\epsilon + 3.15n$

CUCKOO VS BLOOM

- Advantages of Cuckoo?
  - Better space usage for small epsilon
  - Fewer hash functions
  - Better cache performance
- Disadvantages?
  - Can be worse for very large epsilon
  - More complicated! (Important for small systems)

CUCKOO VS BLOOM



From Fan et al. 2014

CAN WE DO BETTER?

- Cuckoo filters have not-great constants
- TWO I/Os per lookup
- Did we talk about a way to hash that has one I/O per lookup for small items?
  - Aren't fingerprints small?

- Can we use linear probing to get an even better filter?
- Yes: Quotient Filter (Bender et al. 2012)

BEYOND CUCKOO FILTERS

- Quotient filters have similar performance; similar level of "complicated"
- Also: can get best of both worlds
- Morton Filters (Breslow Jayasena 2018)

NEXT CLASS (AFTER BREAK)

- Bloom filters are all about saving a ~log factor
  - Require a constant number of bits to store each element, no matter how long the original element is
- Can we use less than one bit per element?
- If we're willing to sacrifice accuracy: yes, much much less