Algorithms and Data Structures for Data Science
Bloom Filters

CS 277
Brad Solomon

April 22, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

Learning Objectives

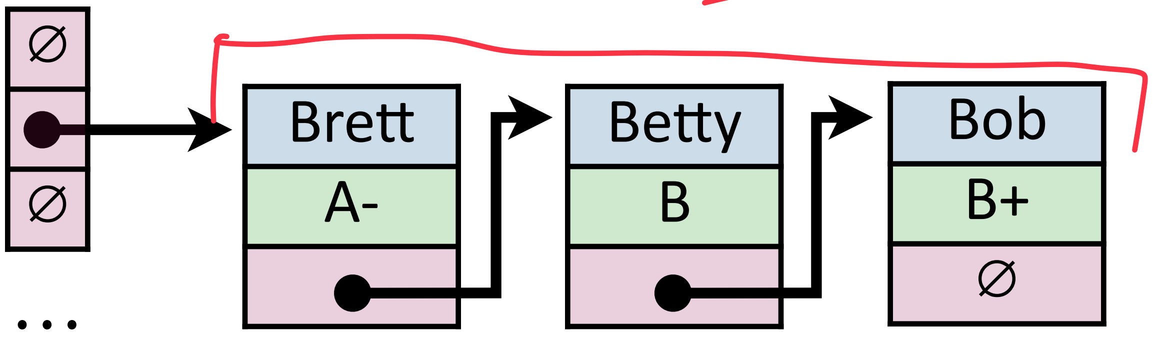
Review fundamentals of hash tables

Introduce probabilistic data structure with bloom filters

Open vs Closed Hashing

Addressing hash collisions depends on your storage structure.

- **Open Hashing:** store *k,v* pairs externally



O(1) insertion

- **Closed Hashing:** store *k,v* pairs in the hash table

m

K, V
K, V
K, V

(Example of closed hashing)

Collision Handling: Double Hashing

S = { 16, 8, 4, 13, 29, 11, 22 }

$|S| = n$

$h_1(k) = k \% 7$

$|Array| = m$

$h_2(k) = 5 - (k \% 5)$

0	22
1	8
2	16
3	29
4	4
5	11
6	13

$h(k, i) = \boxed{(h_1(k) + i*h_2(k))} \% 7$

Try $h(k) = (k + 0*h_2(k)) \% 7$, if full...

Try $h(k) = (k + 1*h_2(k)) \% 7$, if full...

Try $h(k) = (k + 2*h_2(k)) \% 7$, if full...

Try ...

"Stop after m tries"

Running Times *(Don't memorize these equations, no need.)*

The expected number of probes for find(key) under SUHA

Linear Probing:

- Successful: **½(1 + 1/(1-α))**
- Unsuccessful: **½(1 + 1/(1-α))²**

Double Hashing:

- Successful: **1/α * ln(1/(1-α))**
- Unsuccessful: **1/(1-α)**

Separate Chaining:

- Successful: **1 + α/2**
- Unsuccessful: **1 + α**

$\alpha = \frac{n}{m}$ = load factor

Instead, observe:

- As α increases: (gets closer to 1)

Run time → ∞

CH

OH → closer to ∞

- If α is constant:

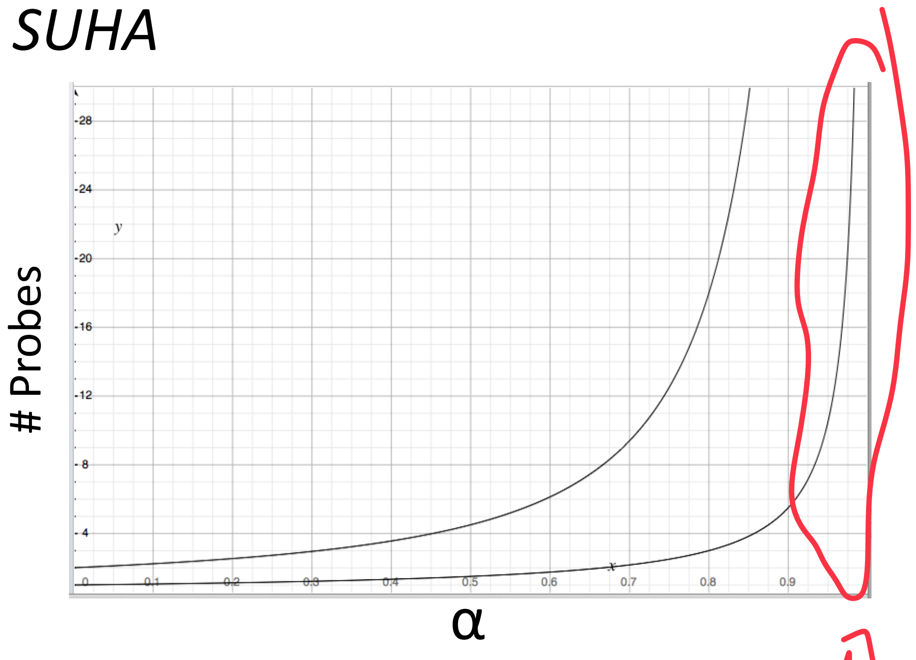
Hashing ops are all constants

Running Times

The expected number of probes for find(key) under SUHA

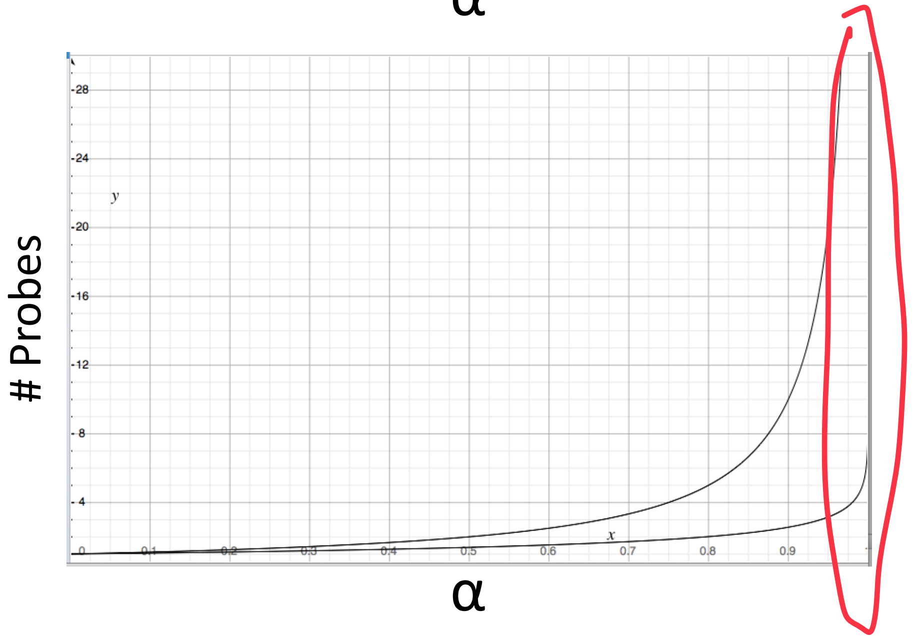
Linear Probing:

- Successful: **½(1 + 1/(1-α))**
- Unsuccessful: **½(1 + 1/(1-α))²**



Double Hashing:

- Successful: **1/α * ln(1/(1-α))**
- Unsuccessful: **1/(1-α)**



When do we resize? $\alpha \approx 0.7 - 0.9$

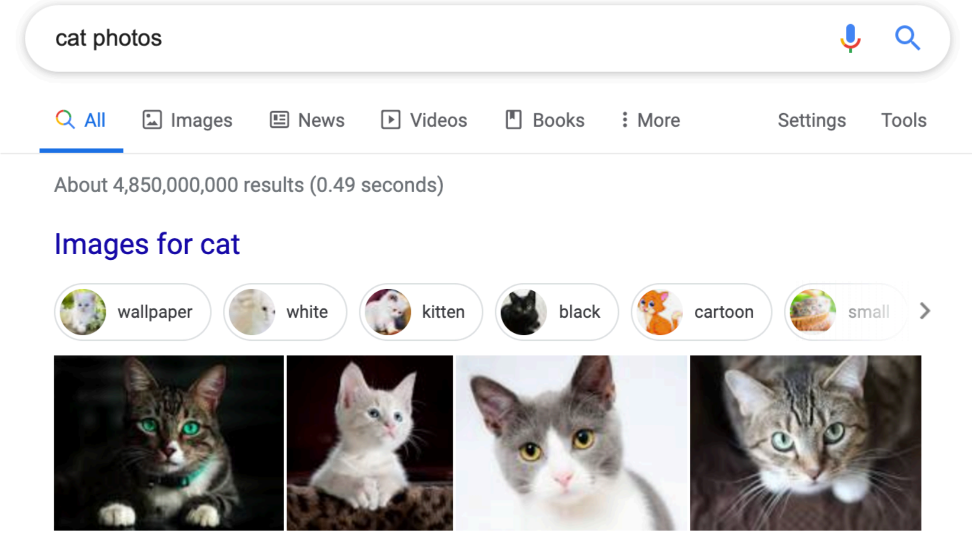
Running Times

	Hash Table	AVL	Linked List
Find	Expectation*: $O(1)$ Worst Case: $O(n)$	$O(\log n)$	$O(n)$
Insert	Expectation*: $O(1)$ Worst Case: $O(n)$ CH $O(1)$ OH	$O(\log n)$	$O(1)$
Storage Space	$O(n)$	$O(n)$	$O(n)$

Memory-Constrained Data Structures

What method would you use to build a search index on a collection of objects *in a memory-constrained environment*?

Constrained by Big Data (Large N)



Google Index Estimate: >60 billion webpages

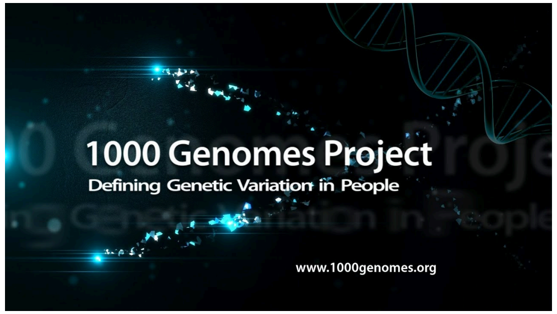
Google Universe Estimate (2013): >130 trillion webpages

Memory-Constrained Data Structures

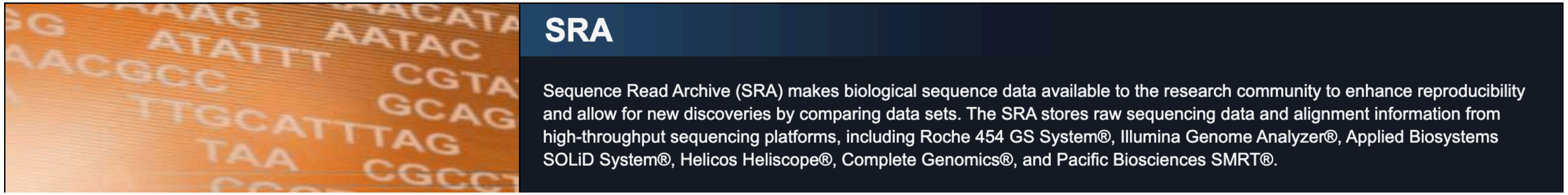
What method would you use to build a search index on a collection of objects *in a memory-constrained environment*?

Constrained by Big Data (Large N)

ENA
European Nucleotide Archive



GTEx



Sequence Read Archive Size: >60 petabases (10^{15})

Memory-Constrained Data Structures

What method would you use to build a search index on a collection of objects *in a memory-constrained environment*?

Constrained by Big Data (Large N)

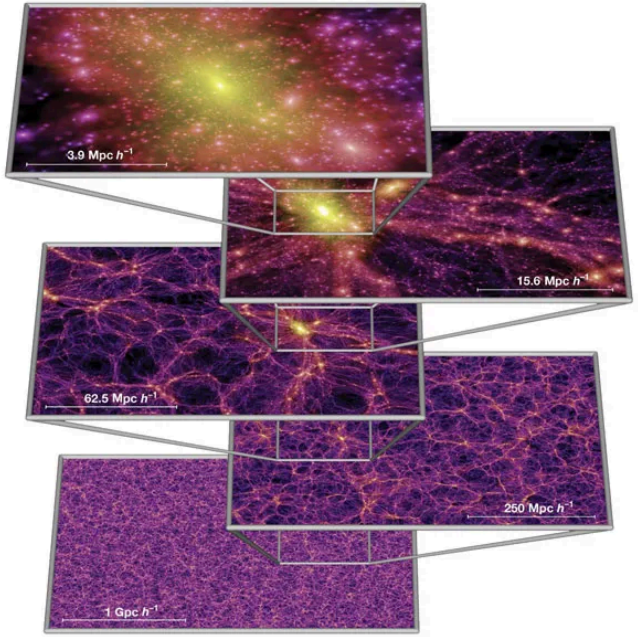


Image: https://doi.org/10.1038/nature03597

Sky Survey Projects	Data Volume
DPOSS (The Palomar Digital Sky Survey)	3 TB
2MASS (The Two Micron All-Sky Survey)	10 TB
GBT (Green Bank Telescope)	20 PB
GALEX (The Galaxy Evolution Explorer)	30 TB
SDSS (The Sloan Digital Sky Survey)	40 TB
SkyMapper Southern Sky Survey	500 TB
PanSTARRS (The Panoramic Survey Telescope and Rapid Response System)	~ 40 PB expected
LSST (The Large Synoptic Survey Telescope)	~ 200 PB expected
SKA (The Square Kilometer Array)	~ 4.6 EB expected

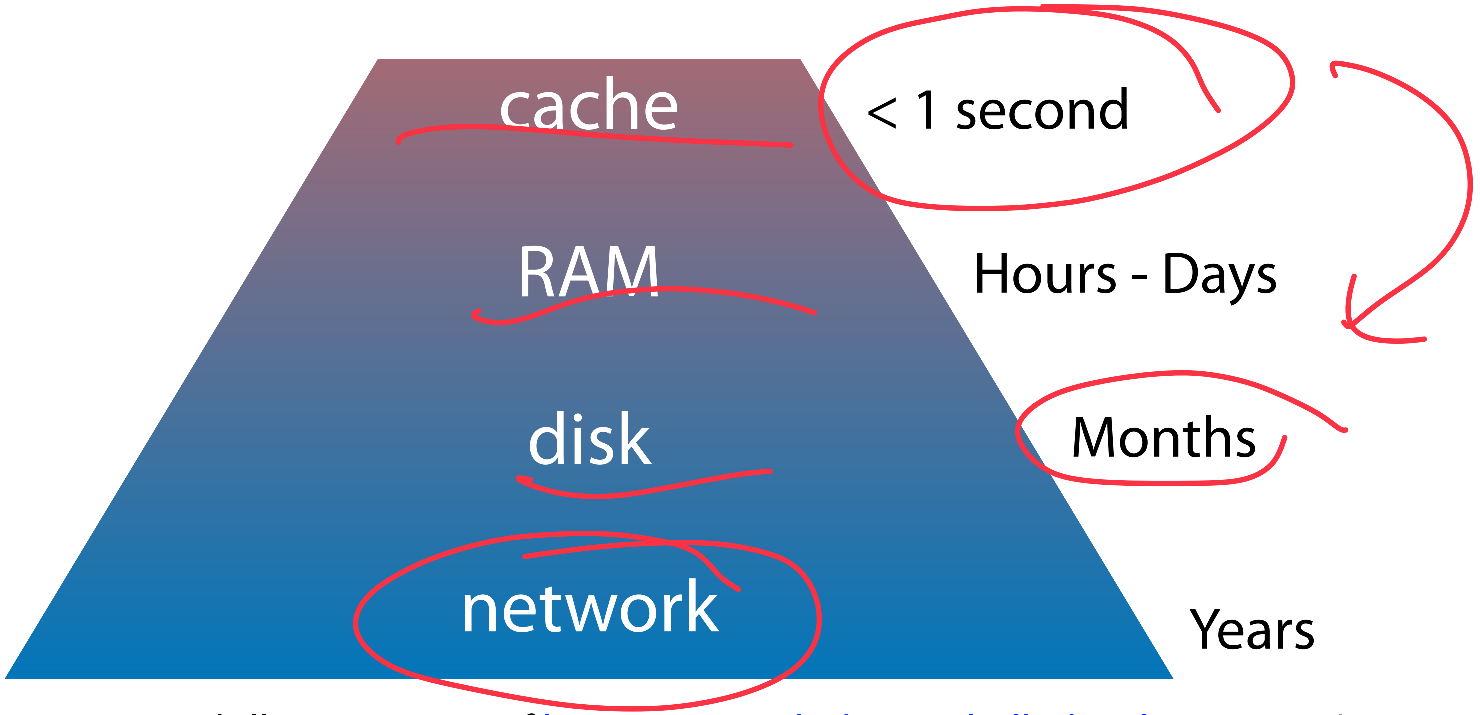
Table: http://doi.org/10.5334/dsj-2015-011

Estimated total volume of one array: 4.6 EB

Memory-Constrained Data Structures

What method would you use to build a search index on a collection of objects *in a memory-constrained environment*?

Constrained by resource limitations



(Estimates are Time x 1 billion courtesy of https://gist.github.com/hellerbarde/2843375)

Memory-Constrained Data Structures

What method would you use to build a search index on a collection of objects *in a memory-constrained environment*?

ADT → Find, insert, remove

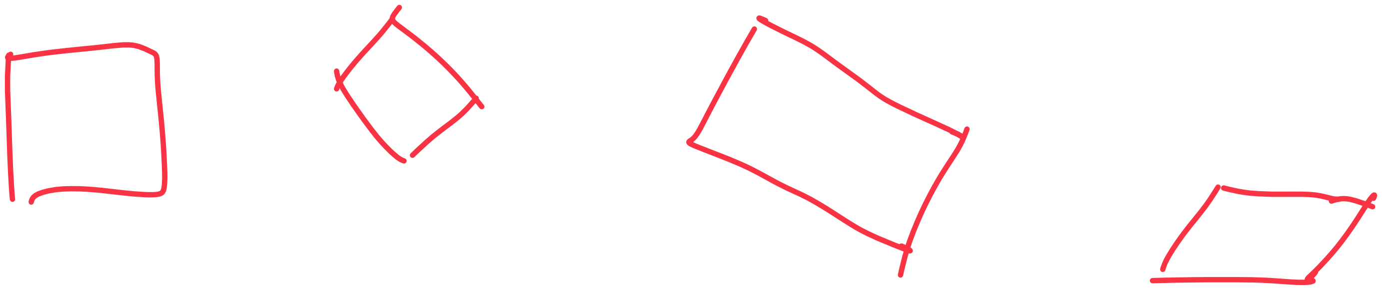
Tree → AVL tree / KD-tree

↳ O(log n)

Reducing storage costs

1) Throw out information that isn't needed (Lossy)

4 count of
4 sided shapes



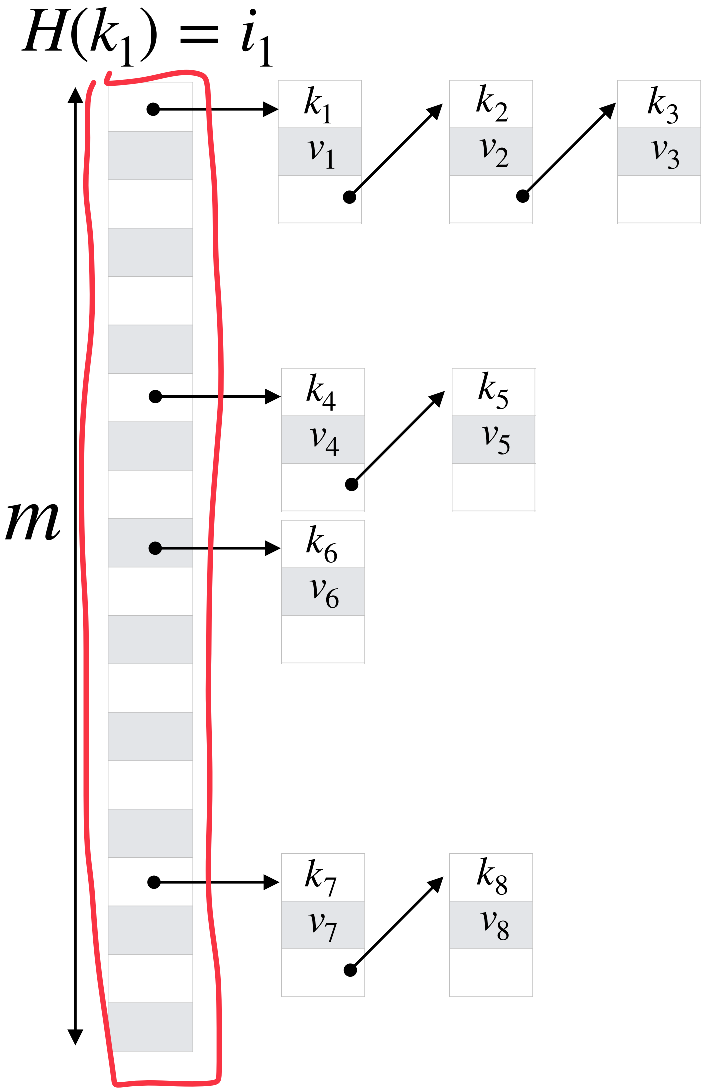
2) Compress the dataset (Exact)

AAAAGGG

A4 G3

Reducing a hash table

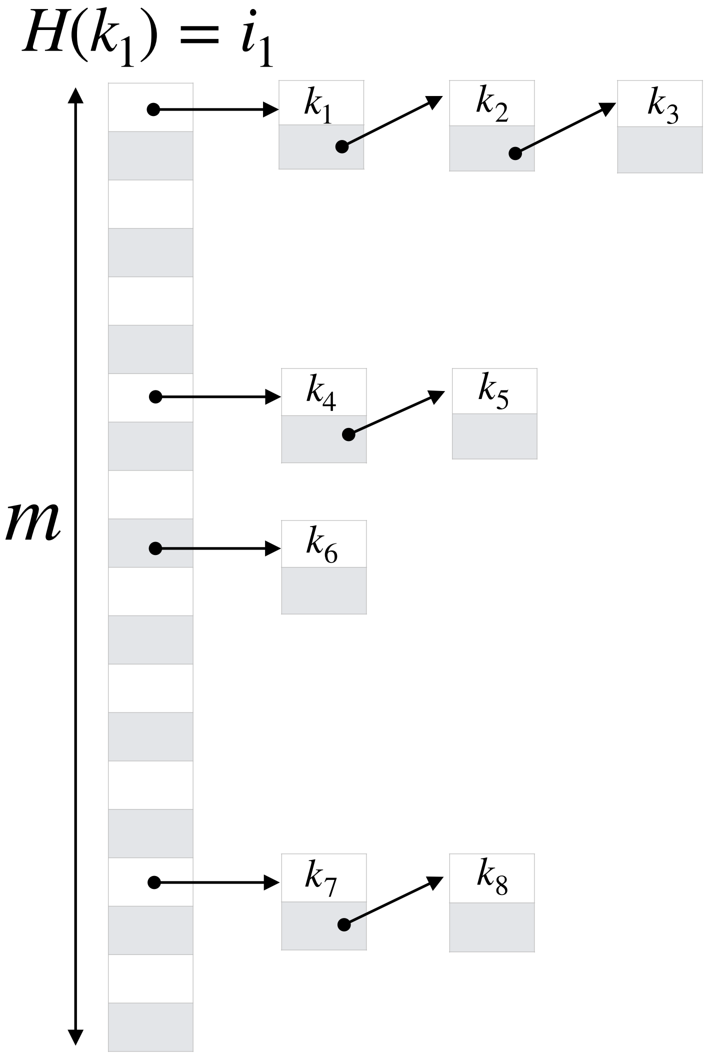
What can we remove from a hash table?



Reducing a hash table

What can we remove from a hash table?

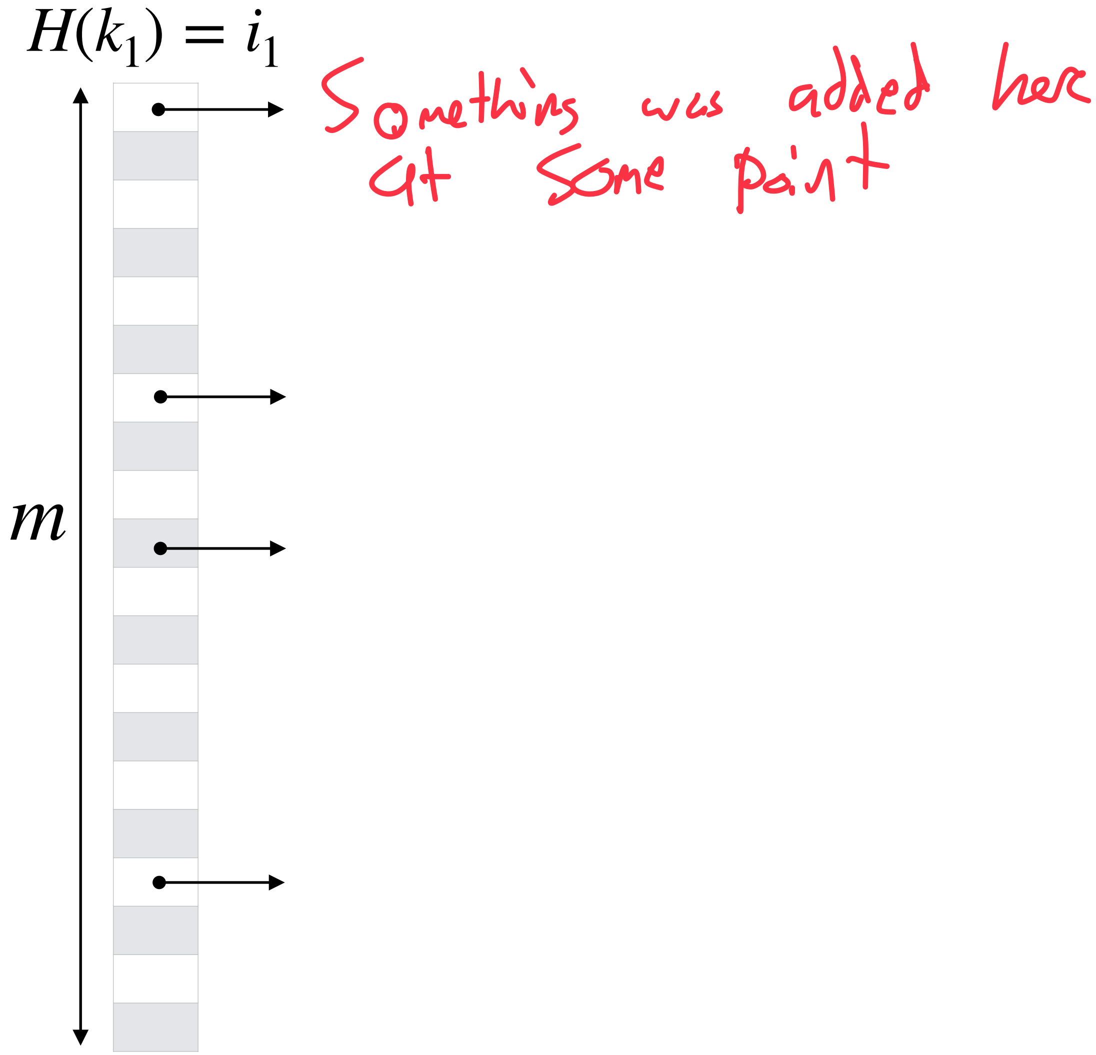
Take away values



Reducing a hash table

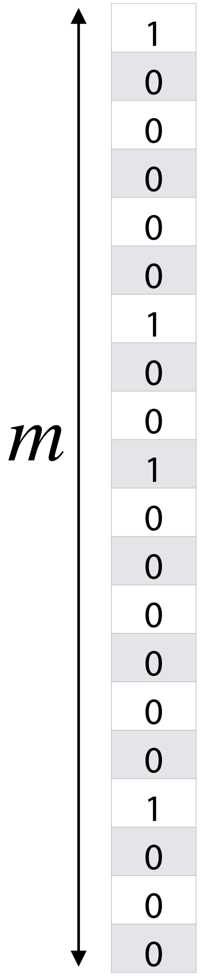
What can we remove from a hash table?

Take away values and keys



Reducing a hash table

$H(k_1) = i_1$



What can we remove from a hash table?

Take away values and keys

This is a ***bloom filter***

Bloom Filter ADT

Constructor

Insert

Find

Delete

Bloom Filter: Insertion

$O(1)$

S = { 16, 8, 4, 13, 29, 11, 22 }

h(k) = k % 7

$16 \% 7 = 2$

$29 \% 7 = 1$

0	0
1	~~0~~ 1
2	~~0~~ 1
3	0
4	~~0~~ 1
5	0
6	~~0~~ 1

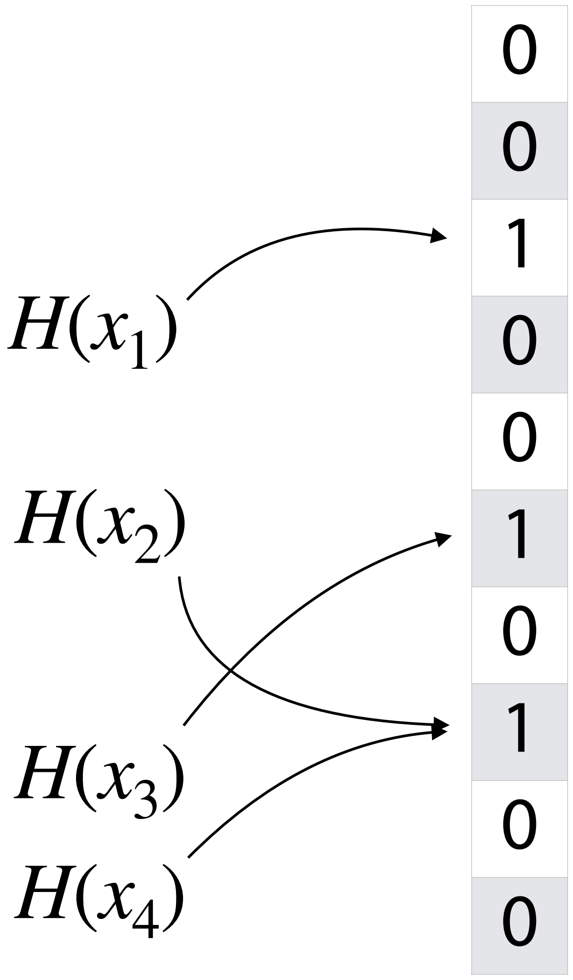
If 1 at pos, stays 1

Collisions don't matter!

Bloom Filter: Insertion

An item is inserted into a bloom filter by hashing and then setting the hash-valued bit to 1

If the bit was already one, it stays 1



Bloom Filter: Deletion

S = { 16, 8, 4, 13, 29, 11, 22 }

h(k) = k % 7

0	0
1	~~1~~ 0
2	1
3	0
4	1
5	0
6	~~1~~ 0

`_delete(13)`

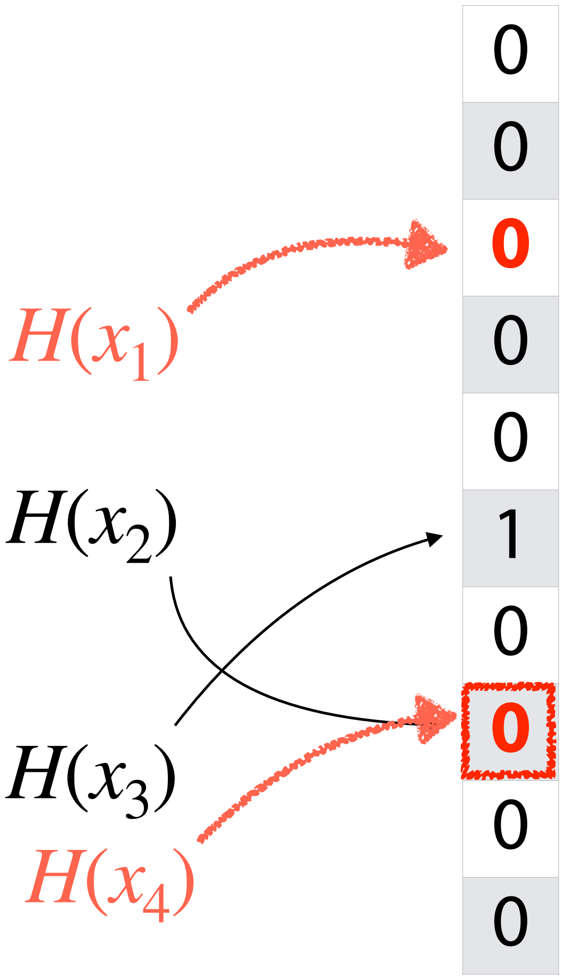
2

`_delete(`~~29~~`)`

_find(8)

Bloom Filter: Deletion

Due to hash collisions and lack of information, items cannot be deleted!



Bloom Filter: Search

S = { 16, 8, 4, ~~13,~~ 29, 11, 22 }

h(k) = k % 7

0	0
1	1
2	1
3	0
4	1
5	0
6	1

`_find(16)` → exists!

16 % 7 = 2

1) Hash value

2) Look at value

`_find(20)` exists

20 % 7 = 6

`_find(3)` doesn't exist!

Bloom Filter: Search

The bloom filter is a *probabilistic* data structure!

If the value in the BF is 0:

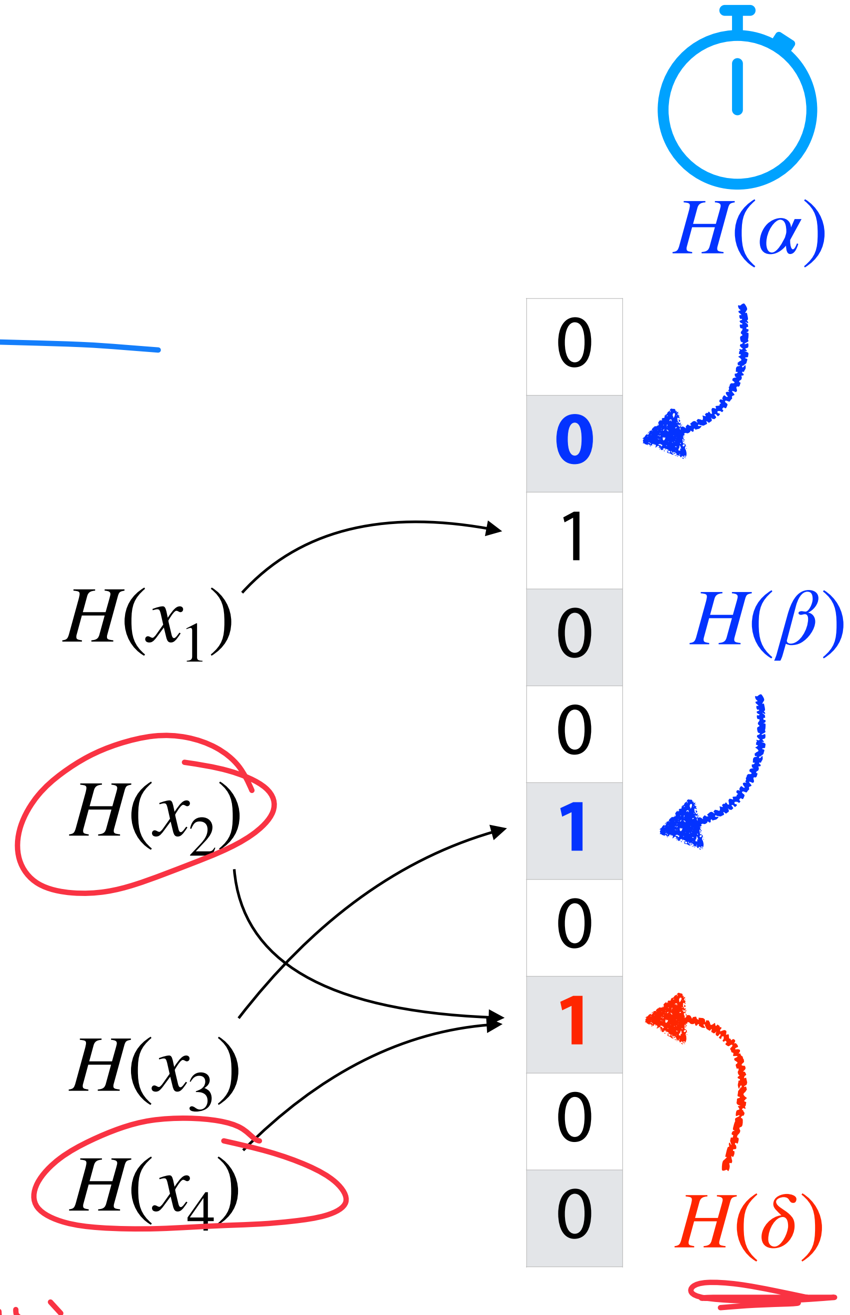
item is 100% not in dataset

↳ No deletions! If item was inserted would be 1 for ever

If the value in the BF is 1:

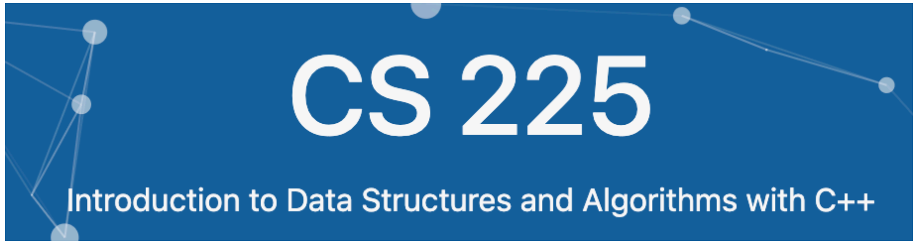
item might be present

↳ we dont know if query was inserted or a collision

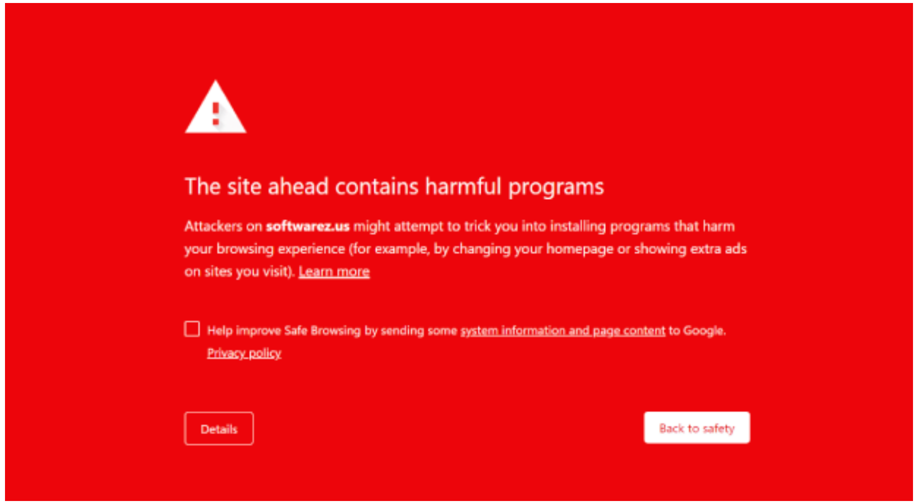


Probabilistic Accuracy: Malicious Websites

Imagine we have a detection oracle that identifies if a site is malicious



"Not malicious"



"Malicious"

Probabilistic Accuracy: Malicious Websites

Imagine we have a detection oracle that identifies if a site is malicious

True Positive: Oracle says Malicious | website is Malicious

False Positive: Malicious | website is Safe

False Negative: Safe (Not Malicious | website is malicious

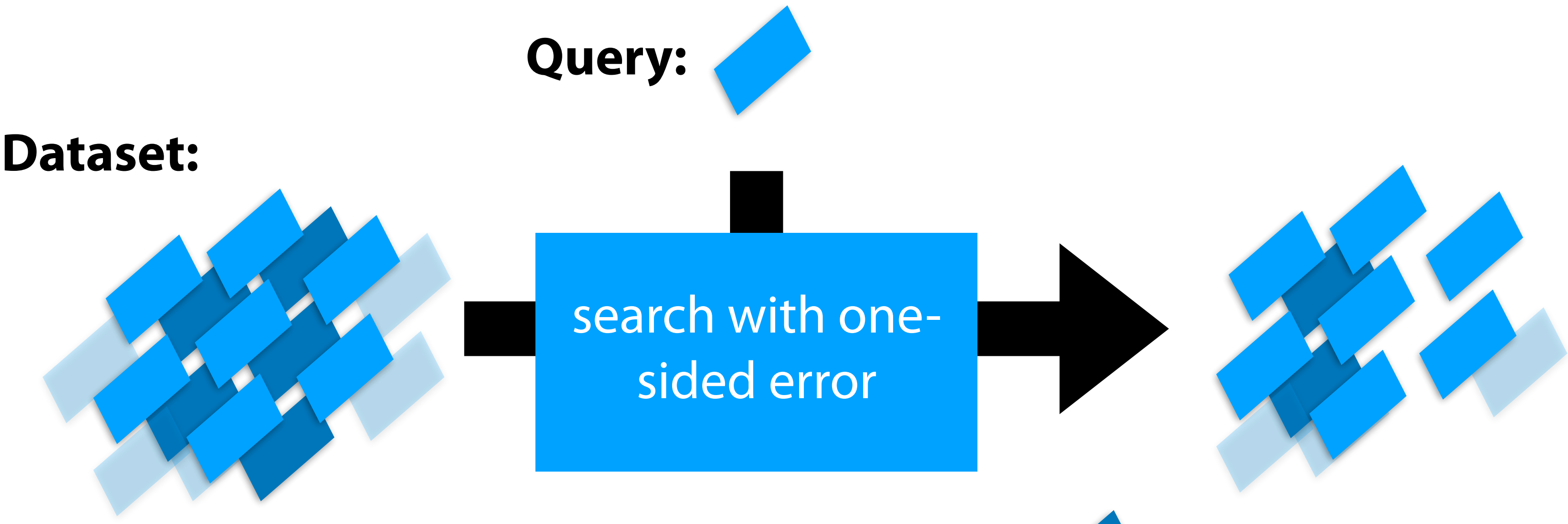
True Negative: Safe | Safe

Imagine we have a **bloom filter** that **stores malicious sites…**

	Bit Value = 1	**Bit Value = 0**
Item Inserted (Malicious sites)	$H(z)$ → 0, 1, 0, 0, 1 'Yes' — True Positive	~~$H(z)$ → 0, 0, 0, 0, 1 'No' — False Negative~~
Item NOT inserted	H(F) → 0, 1, 0, 0, 1 'Yes' H(x) — False Positive	0, 0, 0, 0, 1 'No' — True Negative

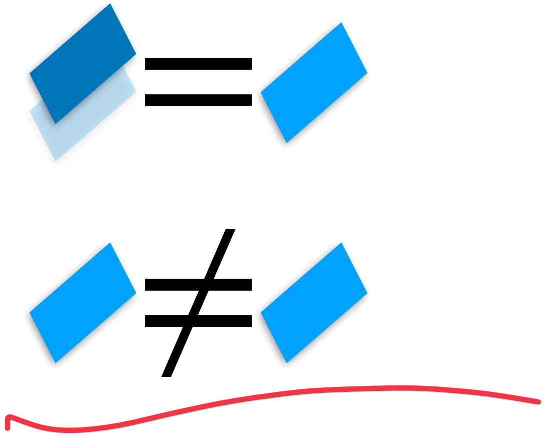
No deletion

Probabilistic Accuracy: One-sided error

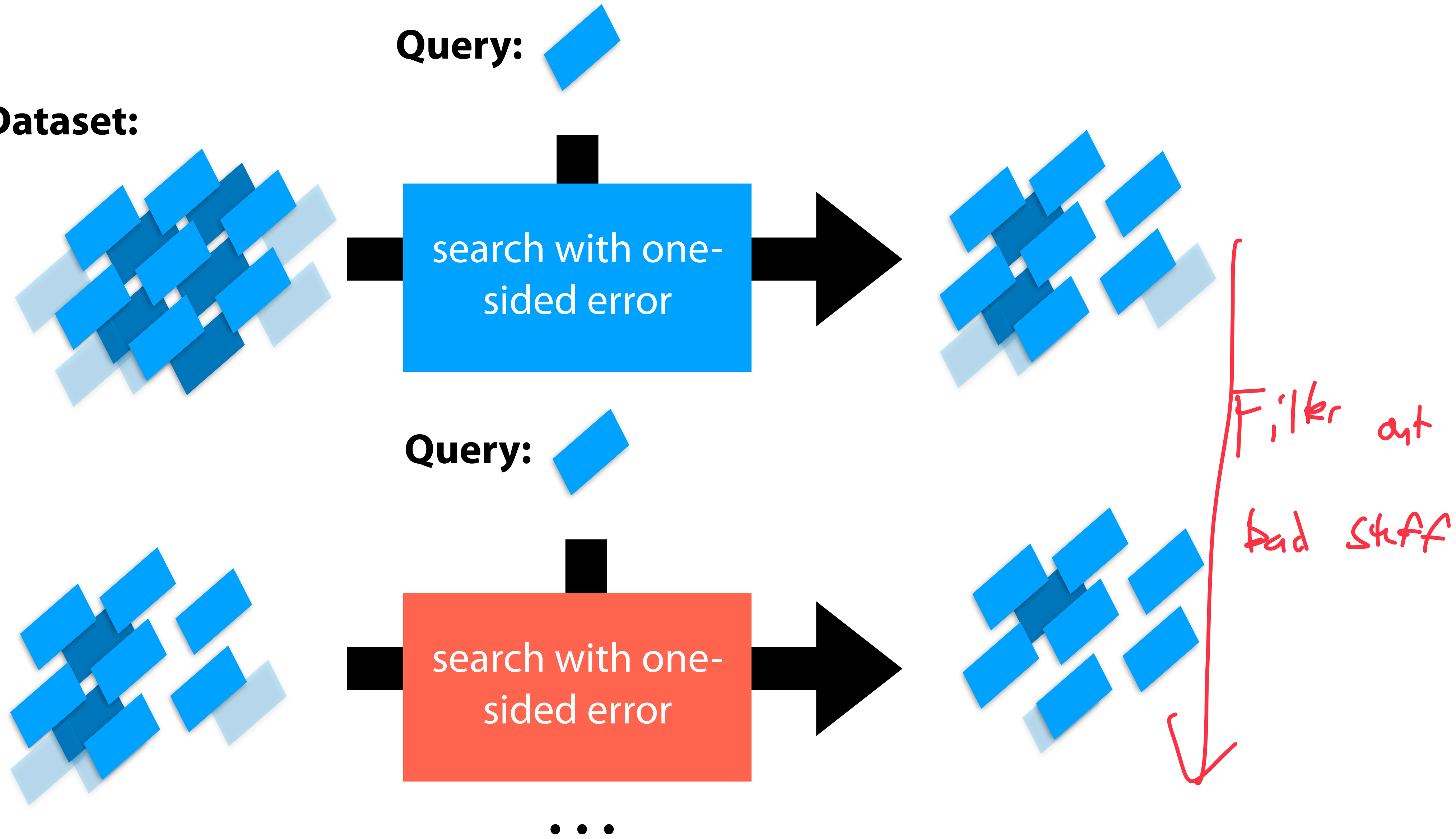


We will get some False Positives:

We will NEVER have a False Negative:



Probabilistic Accuracy: One-sided error

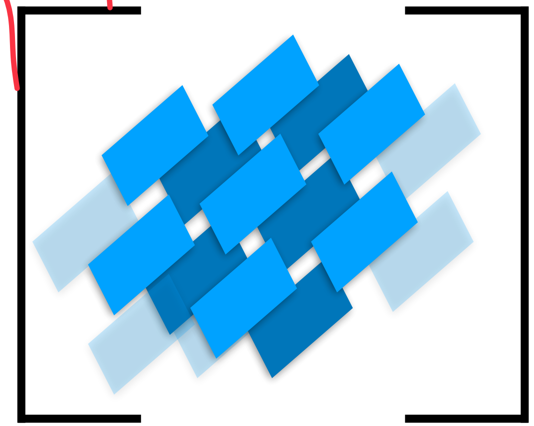


Bloom Filter: Repeated Trials

Use many hashes/filters; add each item to each filter

0
1
0
1
0
0
0
1
0
1
1
0
1
0
1
1
0
1
0
1

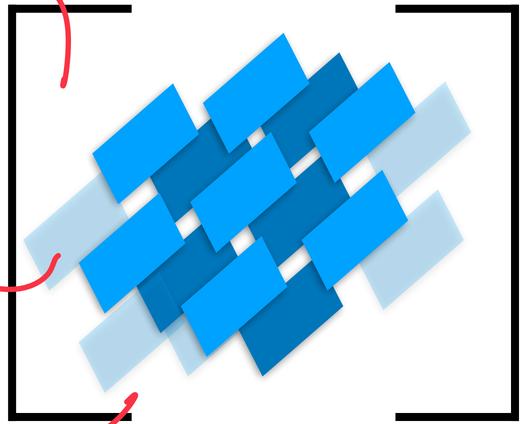
h_1



Bloom Filter: Repeated Trials

Use many hashes/filters; add each item to each filter

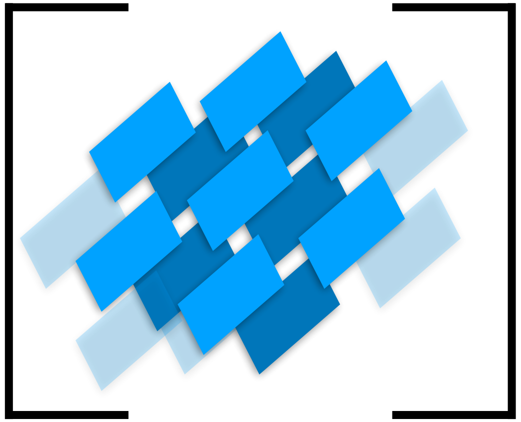
h_1	h_2
0	0
1	0
0	0
1	**1**
0	0
0	0
0	0
1	**1**
0	0
1	**1**
1	0
0	0
1	**1**
0	**1**
1	**1**
1	0
0	0
1	**1**
0	0
1	0



Bloom Filter: Repeated Trials

Use many hashes/filters; add each item to each filter

h_1	h_2	h_3
0	0	0
1	0	**1**
0	0	**1**
1	1	**1**
0	0	0
0	0	0
0	0	**1**
1	1	**1**
0	0	0
1	1	**1**
1	0	**1**
0	0	0
1	1	**1**
0	1	0
1	1	**1**
1	0	**1**
0	0	0
1	1	**1**
0	0	0
1	0	**1**



Bloom Filter: Repeated Trials

Use many hashes/filters; add each item to each filter

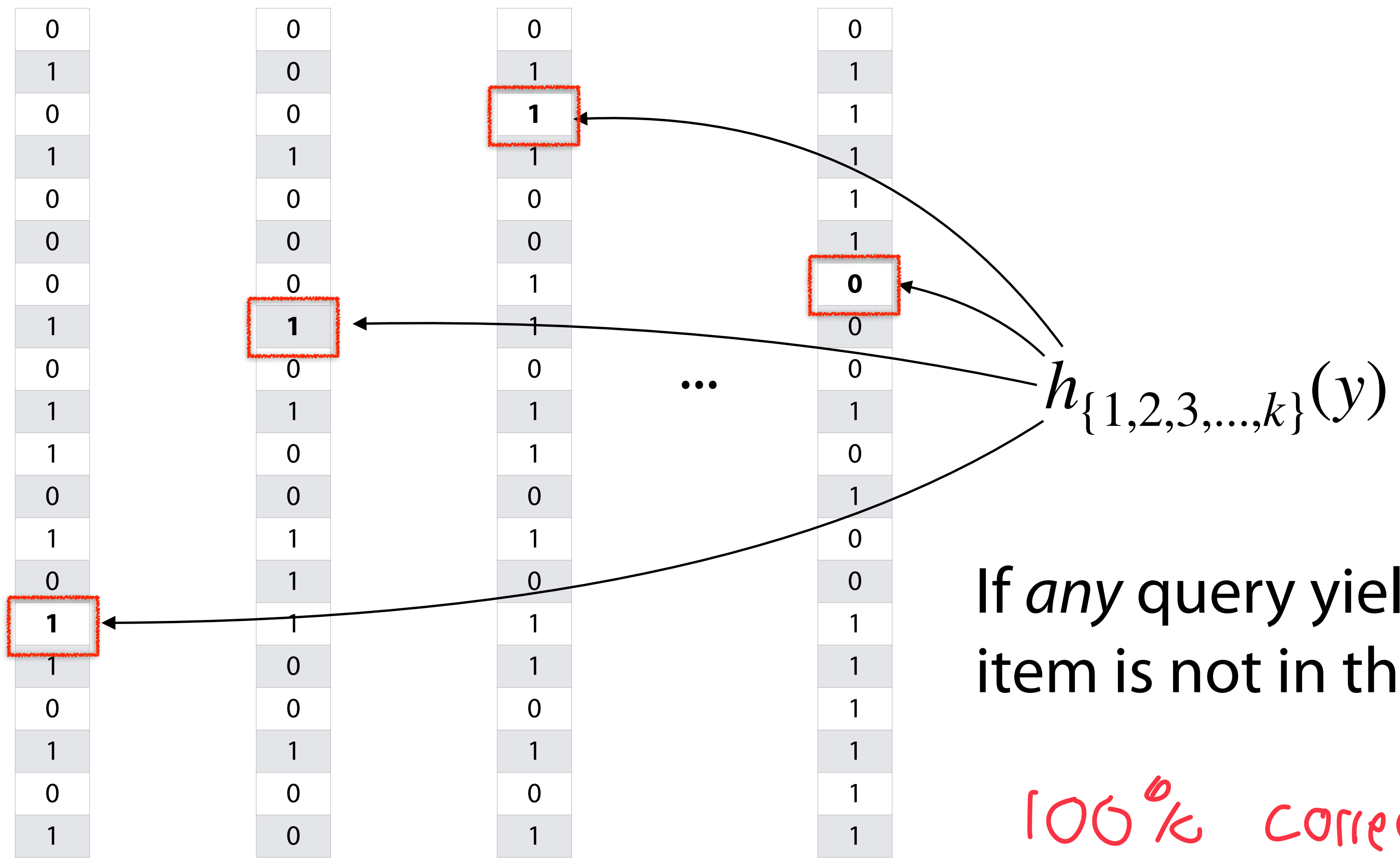
h_1	h_2	h_3	…	h_k
0	0	0		0
1	0	1		1
0	0	1		1
1	1	1		1
0	0	0		1
0	0	0		1
0	0	1		0
1	1	1		0
0	0	0		0
1	1	1		1
1	0	1		0
0	0	0		1
1	1	1		0
0	1	0		0
1	1	1		1
1	0	1		1
0	0	0		1
1	1	1		1
0	0	0		1
1	0	1		1

Bloom Filter: Repeated Trials

0	0	0		0
1	0	1		1
0	0	1		1
1	1	1		1
0	0	0		1
0	0	0		1
0	0	1		0
1	1	1		0
0	0	0	…	0
1	1	1		1
1	0	1		0
0	0	0		1
1	1	1		0
0	1	0		0
1	1	1		1
1	0	1		1
0	0	0		1
1	1	1		1
0	0	0		1
1	0	1		1

$$h_{\{1,2,3,\ldots,k\}}(y)$$

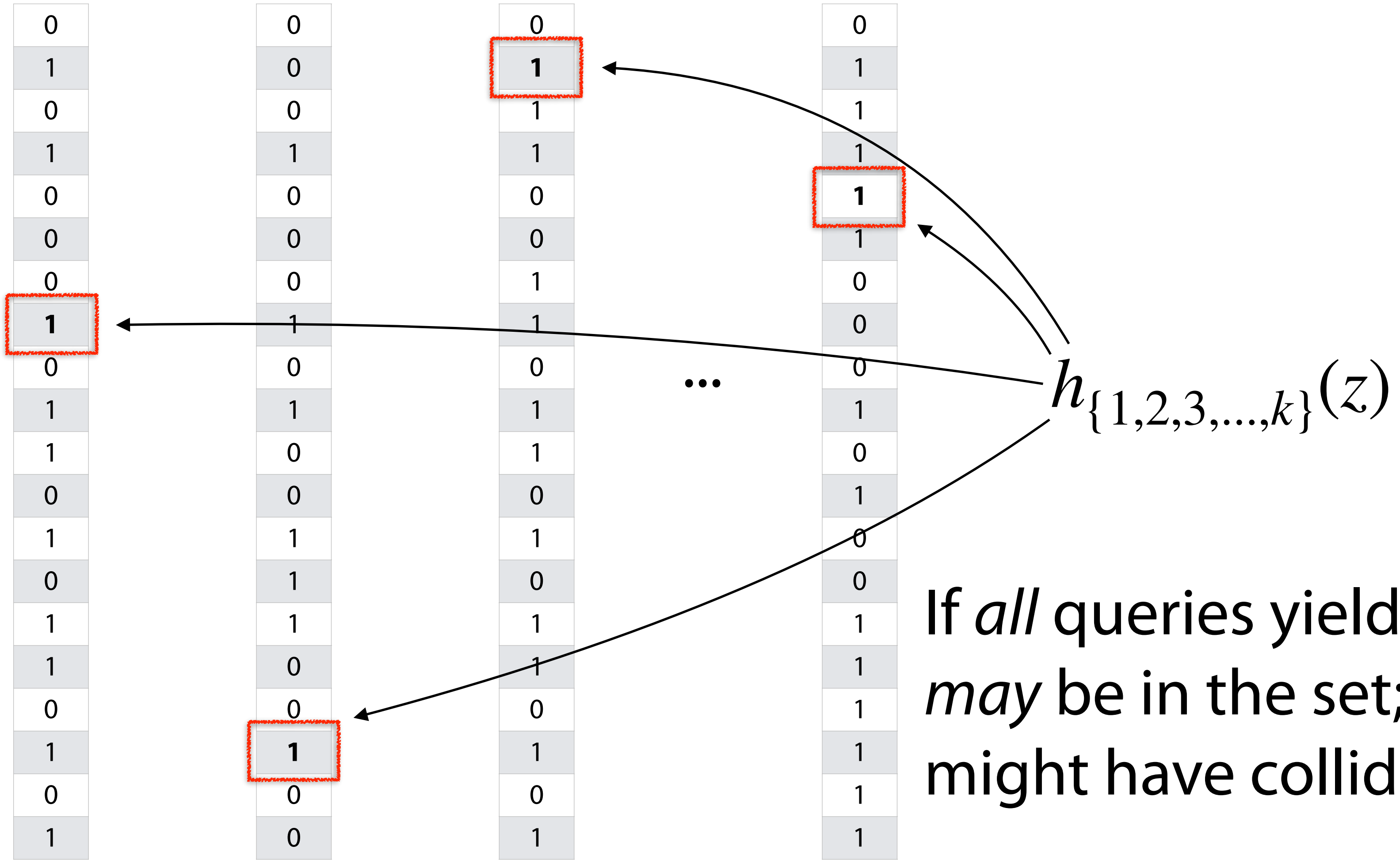
Bloom Filter: Repeated Trials



If *any* query yields 0, item is not in the set

100% correct

Bloom Filter: Repeated Trials



If *all* queries yield 1, item *may* be in the set; or we might have collided *k* times

Bloom Filter: Repeated Trials

$\boxed{\begin{array}{c}1\\ 0\end{array}}$

Using repeated trials, even a very bad filter can still have a very low FPR!

If we have k bloom filter, each with a FPR p, what is the likelihood that ***all*** filters return the value '1' for an item we didn't insert?

$p = 50\%$ $\quad$ $k = 10$ $\quad$ $.5 \cdot .5 = .25$

$(.5)^{10} = 0.00097$

Bloom Filter: Repeated Trials

But doesn't this hurt our storage costs by storing k separate filters?

m

h_1	h_2	h_3	…	h_k
0	0	0		0
1	0	1		1
0	0	1		1
1	1	1		1
0	0	0		1
0	0	0		1
0	0	1		0
1	1	1		0
0	0	0		0
1	1	1		1
1	0	1		0
0	0	0		1
1	1	1		0
0	1	0		0
1	1	1		1
1	0	1		1
0	0	0		1
1	1	1		1
0	0	0		1
1	0	1		1

Bloom Filter: Repeated Trials

Rather than use a new filter for each hash, one filter can use k hashes

0	0
1	0
2	1
3	1
4	1
5	0
6	1
7	1
8	1
9	1

$S = \{ 6, 8, 4 \}$

	$h_1(x) = x \% 10$	**$h_2(x) = 2x \% 10$**	**$h_3(x) = (5+3x) \% 10$**
6:	6	2	3
8:	8	6	9
4:	4	8	7

Bloom Filter: Repeated Trials

Rather than use a new filter for each hash, one filter can use k hashes

0	0
1	0
2	1
3	1
4	1
5	0
6	1
7	1
8	1
9	1

$h_1(x) = x \% 10$　**$h_2(x) = 2x \% 10$**　**$h_3(x) = (5+3x) \% 10$**

`_find(1)` → 100% not present

1　2　8

`_find(16)` → Maybe present!

6　2　3

Bloom Filter

$H = \{h_1, h_2, \ldots, h_k\}$

A probabilistic data structure storing a set of values

Built from a bit vector of length m and k hash functions

Insert / Find runs in: $O(k) \approx O(1)$

Delete is not possible (yet)!

0
0
1
0
0
1
0
1
0
0

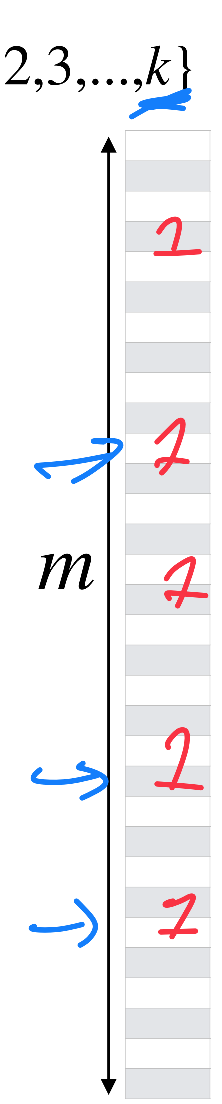
Bloom Filter: Error Rate

Given bit vector of size m and k SUHA hash function

$k=3$

What is our expected FPR after n objects are inserted?

False positive is when we look up some query and by chance k times we look up a 1



Bloom Filter: Error Rate

$h_{\{1,2,3,\ldots,k\}}$

Given bit vector of size m and 1 SUHA hash function

What's the probability a specific bucket is 1 after one object is inserted?

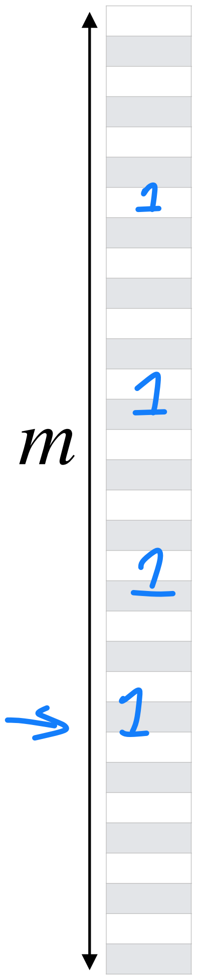
$$\frac{1}{m}$$

$k = 4$

1

1

1



Same probability given k SUHA hash function?

↳ collisions make this math hard!

Bloom Filter: Error Rate

$h_{\{1,2,3,\ldots,k\}}$

Given bit vector of size m and 1 SUHA hash function

Probability a specific bucket is 0 after one object is inserted?

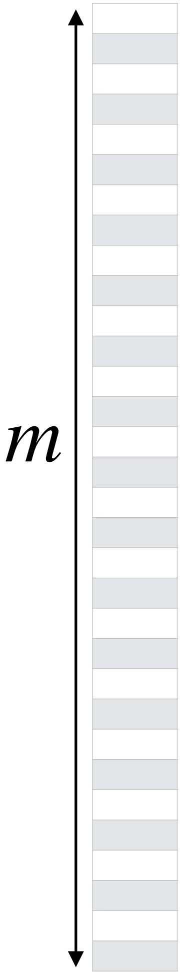
$1 - \frac{1}{m}$

After n objects are inserted? Using k hashes?

Pr Bucket is 0 after $n \cdot k$ inserts

$\left(1 - \frac{1}{m}\right)^{n \cdot k}$

$n \cdot k$ total inserts

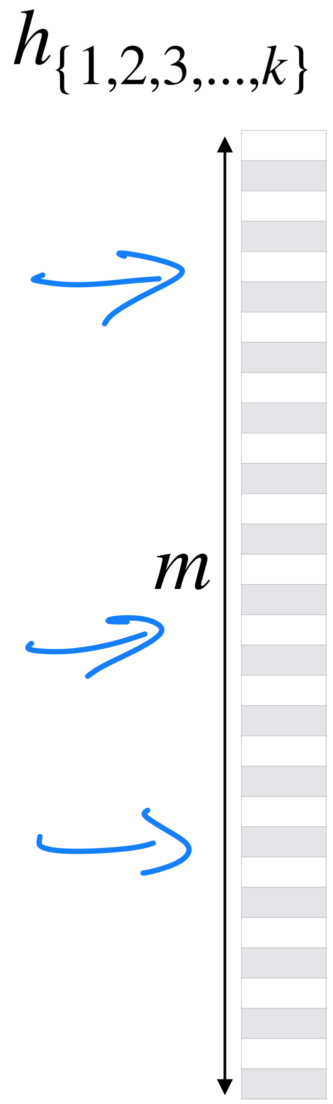


Bloom Filter: Error Rate

Given bit vector of size m and k SUHA hash function

What's the probability a specific bucket is 1 after n objects are inserted? How about k buckets?

$$\left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^{k}$$



Bloom Filter: Error Rate

Given bit vector of size m and k SUHA hash function

What is our expected FPR after n objects are inserted?

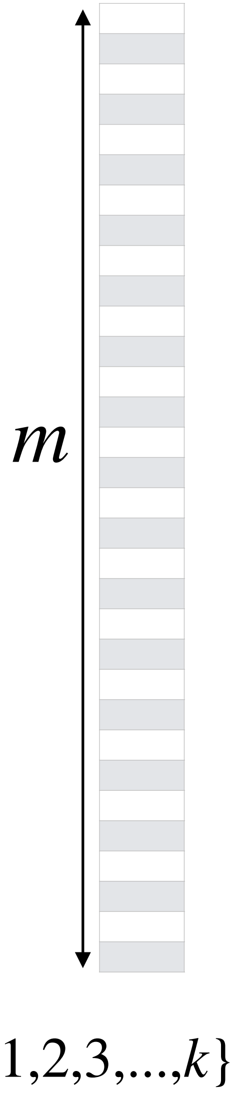
The probability my bit is 1 after n objects inserted ≈ 0.5

$$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k}$$

The number of [assumed independent] trials

1
2
2
1
1

1
0
1
0
1
0



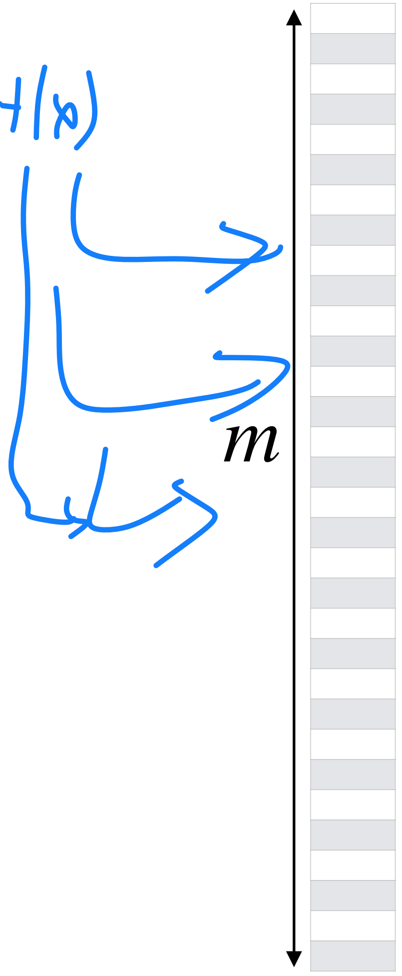
Bloom Filter: Error Rate

$h_{\{1,2,3,\ldots,k\}}$

Vector of size m, k SUHA hash function, and n objects

insert(x)

Add k hash values



To minimize the FPR, do we prefer…

(A) large k **(B) small k**

$$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k}$$

Bloom Filter: Error Rate

Vector of size m, k SUHA hash function, and n objects

(A) large k

$$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k}$$

1
1
1
1
1

As k increases, this gets smaller!

(The percentage of BF which is 1s grows)

(B) small k

$$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k}$$

0
0
1
0
0
1

As k decreases, this gets smaller!

\# of random trials (filter steps)

Bloom Filter: Optimal Error Rate

To build the optimal hash function, fix **m** and **n**!

Claim: The optimal hash function is when $k* = ln\ 2 \cdot \frac{m}{n}$

(1) $\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k} \approx \left(1-e^{\frac{-nk}{m}}\right)^{k}$

(2) $\frac{d}{dk}\left(1-e^{\frac{-nk}{m}}\right)^{k} \approx \frac{d}{dk}\left(k\ \ ln(1-e^{\frac{-nk}{m}})\right)$

Bloom Filter: Optimal Error Rate

Claim 1: $\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k} \approx \left(1-e^{\frac{-nk}{m}}\right)^{k}$

$$\left(1-\frac{1}{m}\right)^{nk} = e^{ln\left[\left(1-\frac{1}{m}\right)^{nk}\right]}$$

$$e^{ln(x)} = x$$

Bloom Filter: Optimal Error Rate

Claim 1: $$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k} \approx \left(1-e^{\frac{-nk}{m}}\right)^{k}$$

$$\left(1-\frac{1}{m}\right)^{nk} = e^{ln\left[\left(1-\frac{1}{m}\right)^{nk}\right]}$$

$$ln(x^{y}) = y\; ln(x)$$

$$= e^{ln\left[\left(1-\frac{1}{m}\right)\right] nk}$$

Bloom Filter: Optimal Error Rate

Claim 1: $$\left(1-\left(1-\frac{1}{m}\right)^{nk}\right)^{k} \approx \left(1-e^{\frac{-nk}{m}}\right)^{k}$$

$$\left(\frac{1}{m}\right)^2$$

$$\left(1-\frac{1}{m}\right)^{nk} = e^{ln\left[\left(1-\frac{1}{m}\right)^{nk}\right]}$$

$$ln(1+x) = x - \cancel{\frac{x^2}{2} + \frac{x^3}{3}} - \dots$$

$$x = -\frac{1}{m}$$

$$= e^{ln\left[\left(1-\frac{1}{m}\right)\right]nk}$$

$$= -\frac{1}{m}$$

$$\approx e^{\frac{-nk}{m}}$$

Bloom Filter: Optimal Error Rate

Claim 2: $$\frac{d}{dk}\left(1 - e^{\frac{-nk}{m}}\right)^k \approx \frac{d}{dk}\left(k \;\; ln(1 - e^{\frac{-nk}{m}})\right)$$

$$min\left[f(x)\right] = min\left[\ln\; f(x)\right]$$

Bloom Filter: Optimal Error Rate

Claim 2: $$\frac{d}{dk}\left(1 - e^{\frac{-nk}{m}}\right)^k \approx \frac{d}{dk}\left(k \;\; ln(1 - e^{\frac{-nk}{m}})\right)$$

Derivative is zero when $k^* = \ln 2 \cdot \dfrac{m}{n}$

$$\frac{d}{dx}\ln\; f(x) = \frac{1}{f(x)}\frac{df(x)}{dx}$$

… and math!

Bloom Filter: Error Rate

K is small

$$\left(1 - e^{\frac{-nk}{m}}\right)^k$$

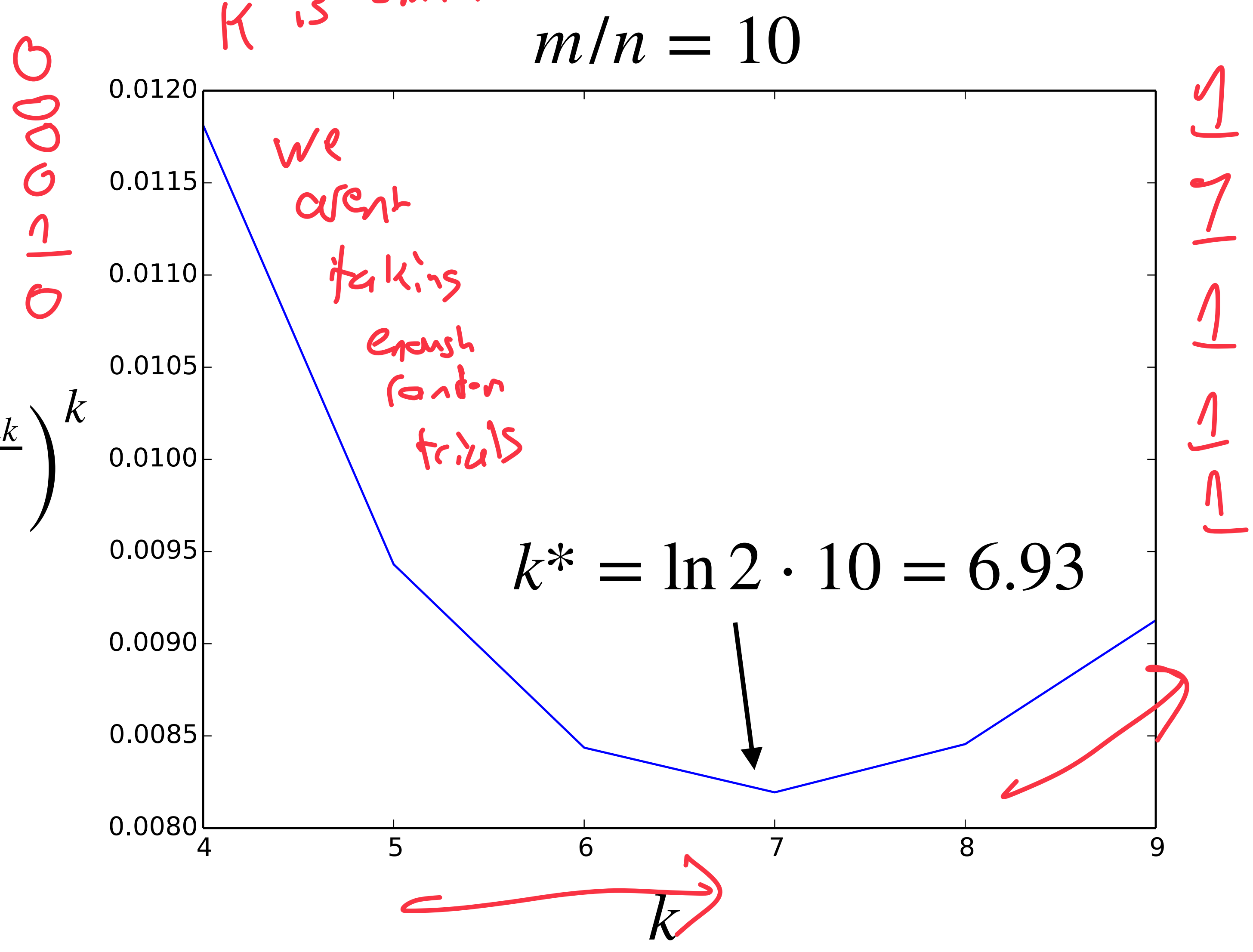


Figure by Ben Langmead

Bloom Filter: Optimal Parameters

$$\boxed{k^* = \ln 2 \cdot \frac{m}{n}}$$

Given any two values, we can optimize the third

$n = 100$ items $k = 3$ hashes $m =$ 433 bits

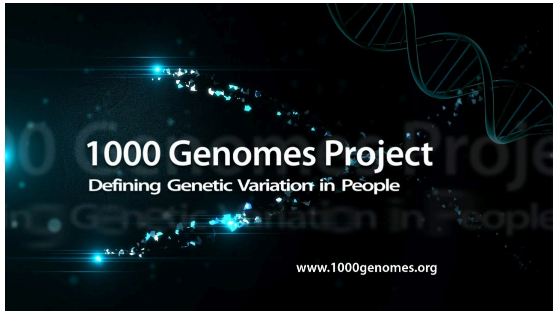
$m = 100$ bits $n = 20$ items $k =$ 3.47 ≈ 4 hashes

$m = 100$ bits $k = 2$ items $n =$ ~~34~~ 34.5 ≈ 35 items

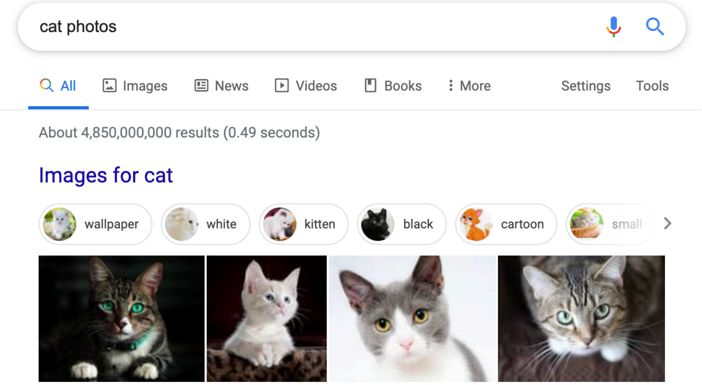
Bloom Filter: Optimal Parameters

$$\boxed{m = \frac{nk}{\ln 2} \approx 1.44 \cdot nk}$$

Optimal hash function is still O(m)!



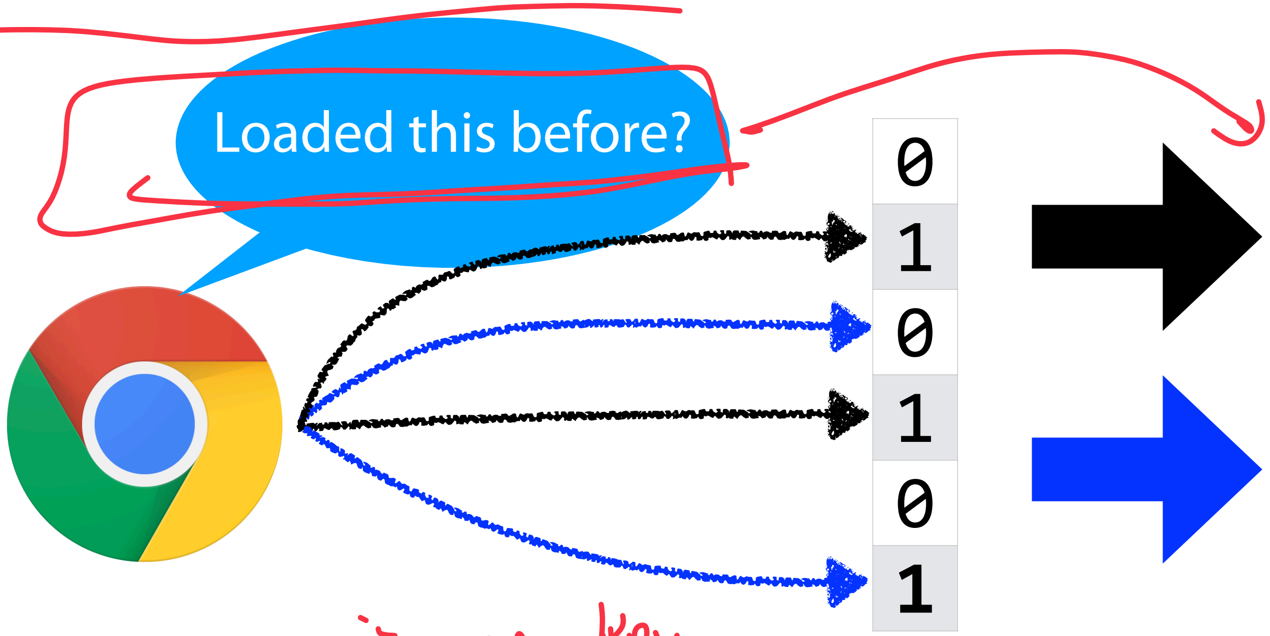
n = 250,000 files vs ~10^{15} nucleotides vs 260 TB



n = 60 billion — 130 trillion

Bloom Filter: Website Caching

Yes I did (But we were wrong)

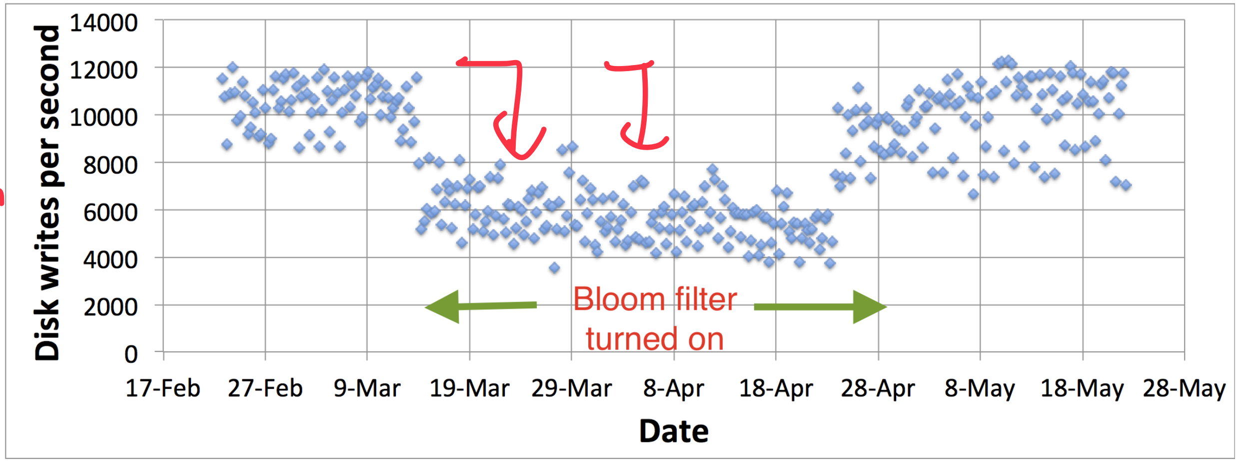


Cache this page!

Add to filter (but don't cache!)

1) Speed and size are key

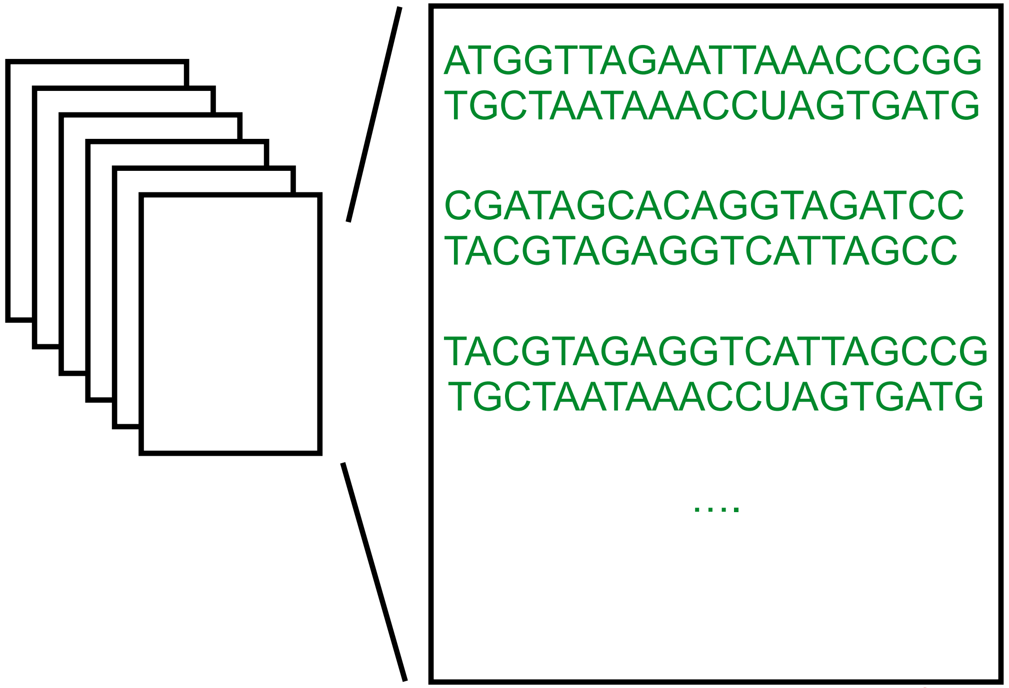
2) False positives aren't a problem



Maggs, Bruce M., and Ramesh K. Sitaraman. **Algorithmic nuggets in content delivery.** *ACM SIGCOMM Computer Communication Review* 45.3 (2015): 52-66.

Sequence Bloom Trees

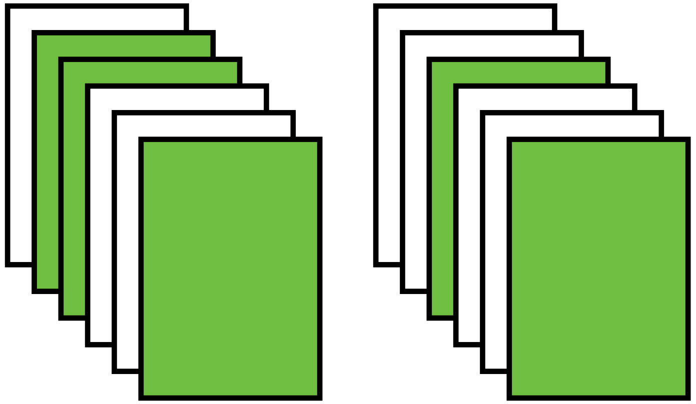
Imagine we have a large collection of text…



↳ collection of words

Insert all words into BF

And our goal is to search these files for a query of interest…



a subset of words

Bloom Filters: Unioning

Bloom filters can be trivially merged using bit-wise union.

0	1		0	0		0	1
1	0		1	1		1	1
2	1		2	1		2	1
3	1		3	0		3	
4	0	∪	4	0	=	4	0
5	0		5	0		5	
6	1		6	1		6	
7	0		7	1		7	
8	0		8	1		8	
9	1		9	1		9	

T T = T
T F = T
F T = T
F F = F

A or B

Bloom Filters: Intersection

X and Y

T T = T

T F = F

F T = F

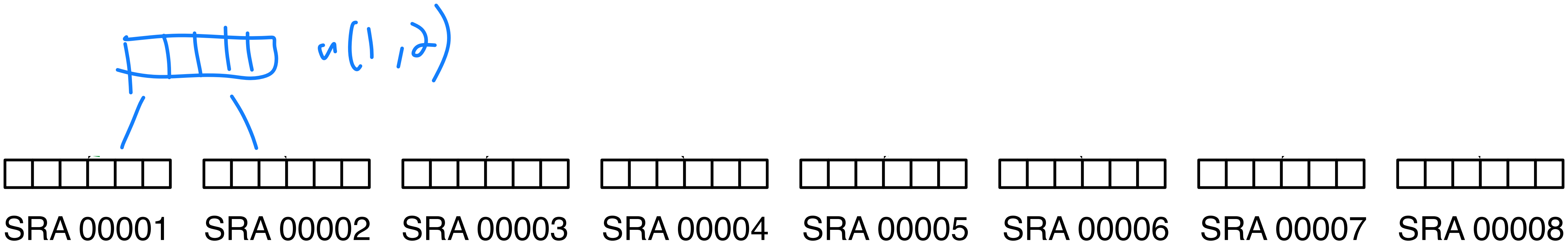
F F = F

Bloom filters can be trivially merged using bit-wise intersection.

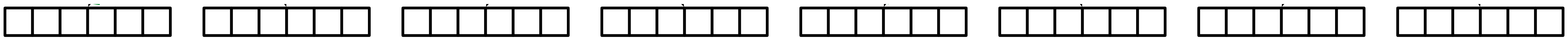
0	1		0	0		0	0
1	0		1	1		1	0
2	1		2	1		2	1
3	1		3	0		3	0
4	0	U	4	0	=	4	0
5	0		5	0		5	
6	1		6	1		6	
7	0		7	1		7	
8	0		8	1		8	
9	1		9	1		9	

Bit Vector Merging

What is the conceptual meaning behind **union** and **intersection**?



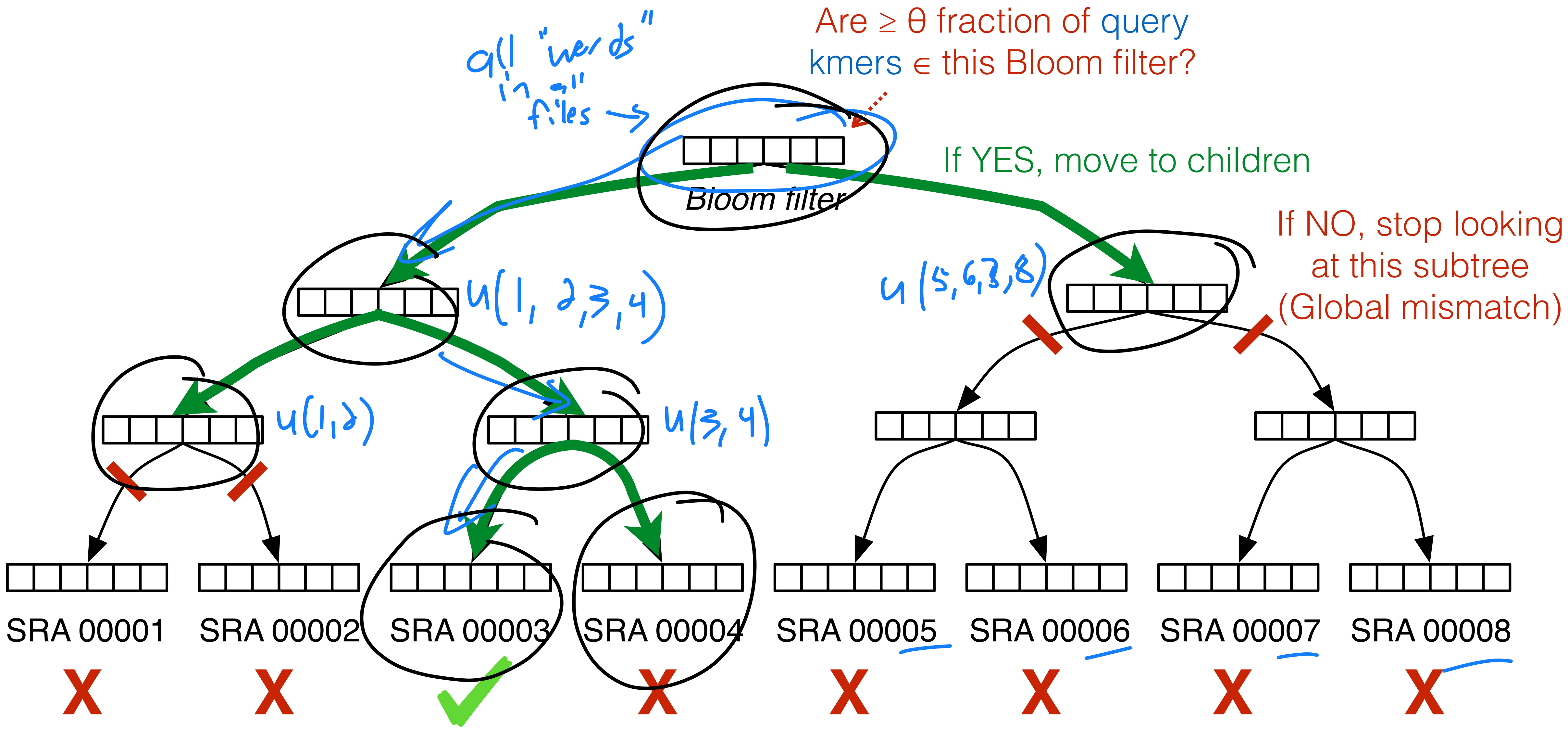
Sequence Bloom Trees



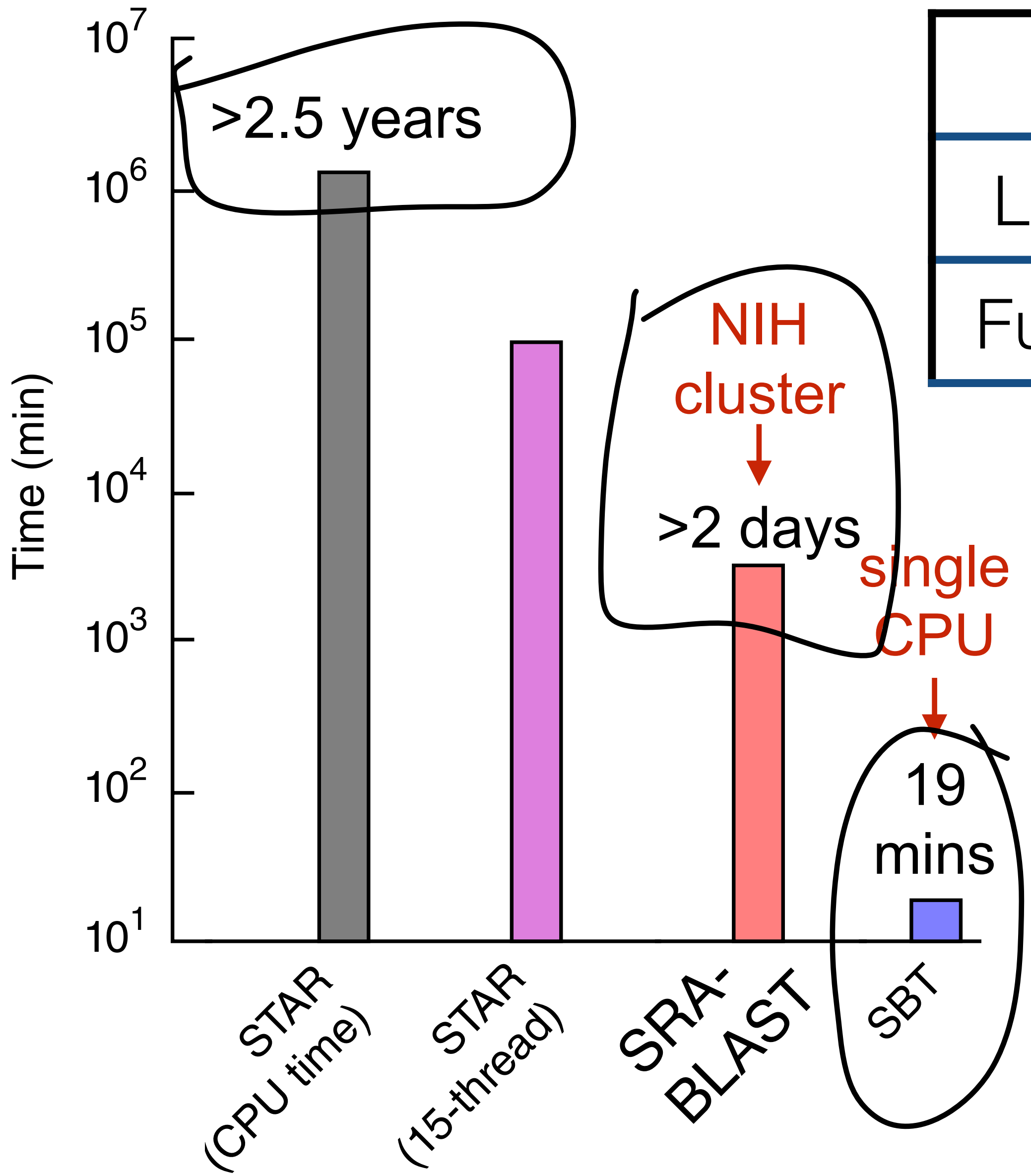
SRA 00001 SRA 00002 SRA 00003 SRA 00004 SRA 00005 SRA 00006 SRA 00007 SRA 00008

Sequence Bloom Trees

70% of query words



Sequence Bloom Trees



	SRA	FASTA.gz	SBT
Leaves	4966 GB	2692 GB	63 GB
Full Tree	-	-	200 GB

Solomon, Brad, and Carl Kingsford. "Fast search of thousands of short-read sequencing experiments." *Nature biotechnology* 34.3 (2016): 300-302.

Solomon, Brad, and Carl Kingsford. "Improved search of large transcriptomic sequencing databases using split sequence bloom trees." *International Conference on Research in Computational Molecular Biology*. Springer, Cham, 2017.

Sun, Chen, et al. "Allsome sequence bloom trees." *International Conference on Research in Computational Molecular Biology*. Springer, Cham, 2017.

Harris, Robert S., and Paul Medvedev. "Improved representation of sequence bloom trees." *Bioinformatics* 36.3 (2020): 721-727.

Bloom Filters: Tip of the Iceberg

Cohen, Saar, and Yossi Matias. "Spectral bloom filters." *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*. 2003.

Fan, Bin, et al. "Cuckoo filter: Practically better than bloom." *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. 2014.

Nayak, Sabuzima, and Ripon Patgiri. "countBF: A General-purpose High Accuracy and Space Efficient Counting Bloom Filter." *2021 17th International Conference on Network and Service Management (CNSM)*. IEEE, 2021.

Mitzenmacher, Michael. "Compressed bloom filters." *IEEE/ACM transactions on networking* 10.5 (2002): 604-612.

Crainiceanu, Adina, and Daniel Lemire. "Bloofi: Multidimensional bloom filters." *Information Systems* 54 (2015): 311-324.

Chazelle, Bernard, et al. "The bloomier filter: an efficient data structure for static support lookup tables." *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*. 2004.

There are many more than shown here…