# Data Structures and Algorithms
# Bloom Filters

CS 225
Brad Solomon

April 19, 2024

UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
Department of Computer Science

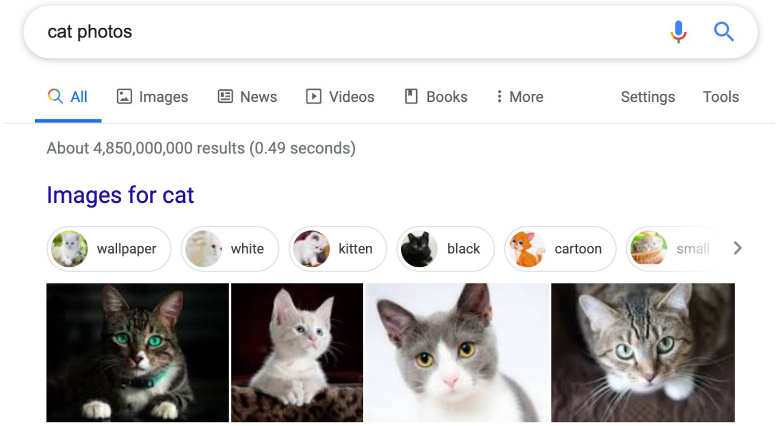
# Data Structures Review

What method would you use to build a search index on a collection of objects?

# 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

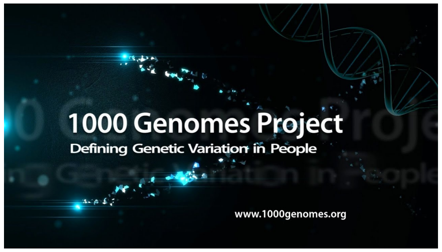
Google Universe Estimate (2013): >130 trillion webpages

# Memory-Constrained Data Structures

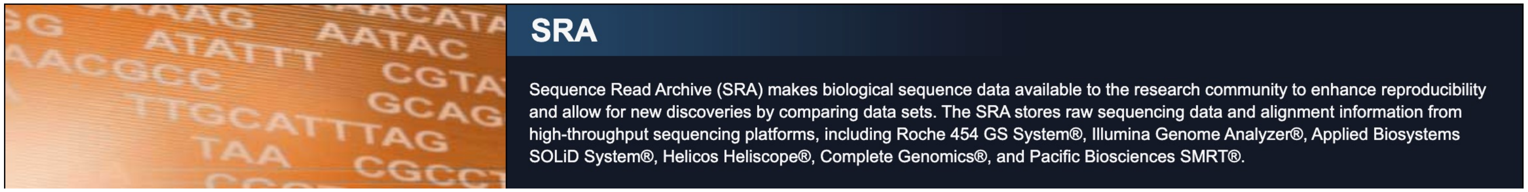
What method would you use to build a search index on a collection of objects *in a memory-constrained environment*?

**Constrained by Big Data (Large $N$)**

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 $N$)

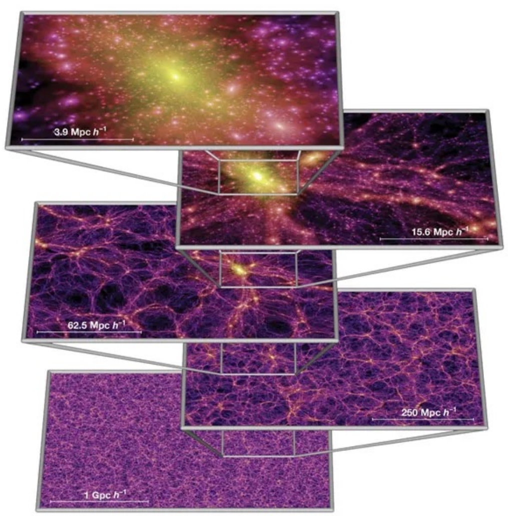



Image: https://doi.org/10.1038/nature03597

| 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 |

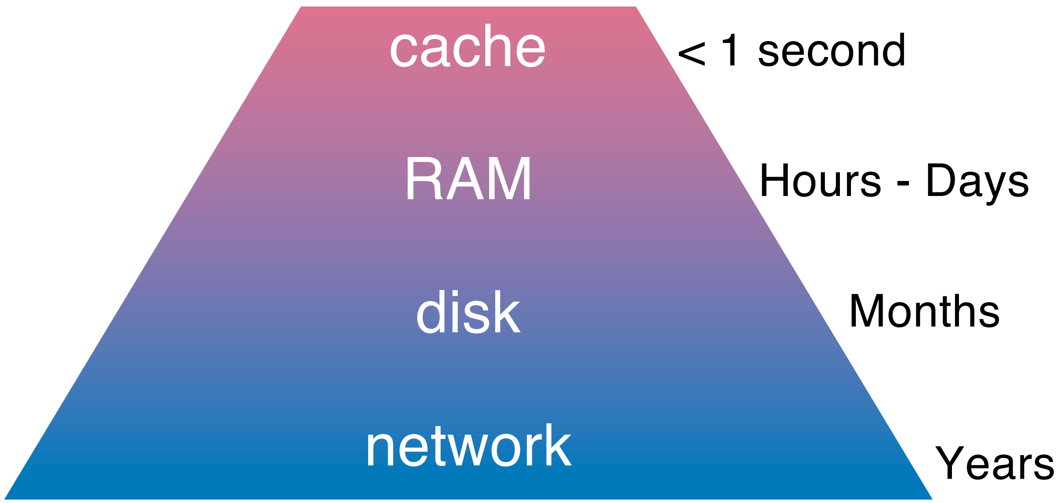
Table: http://doi.org/10.5334/dsj-2015-011

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*?

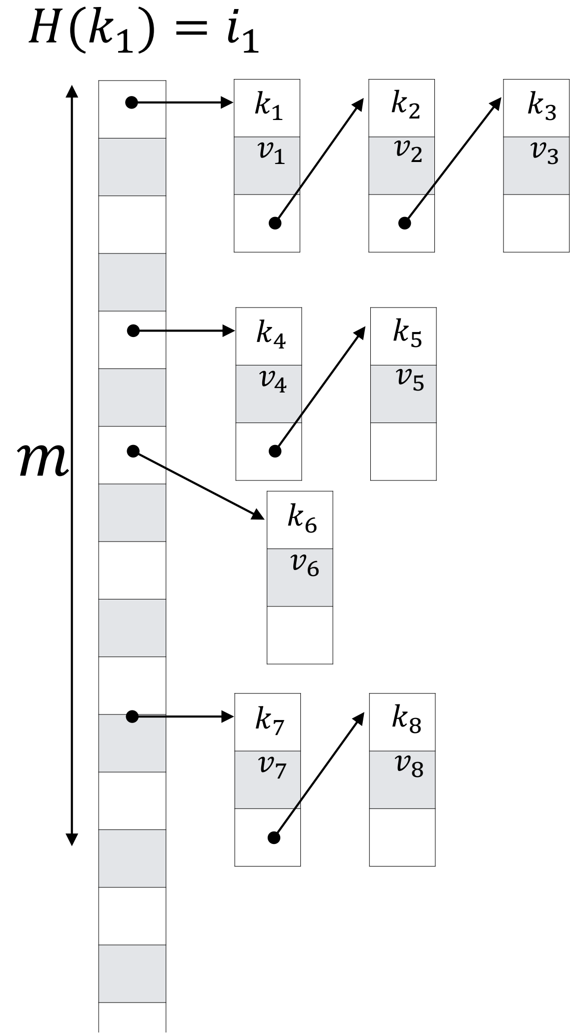
# Reducing storage costs

1) Throw out information that isn't needed

2) Compress the dataset

# Reducing a hash table

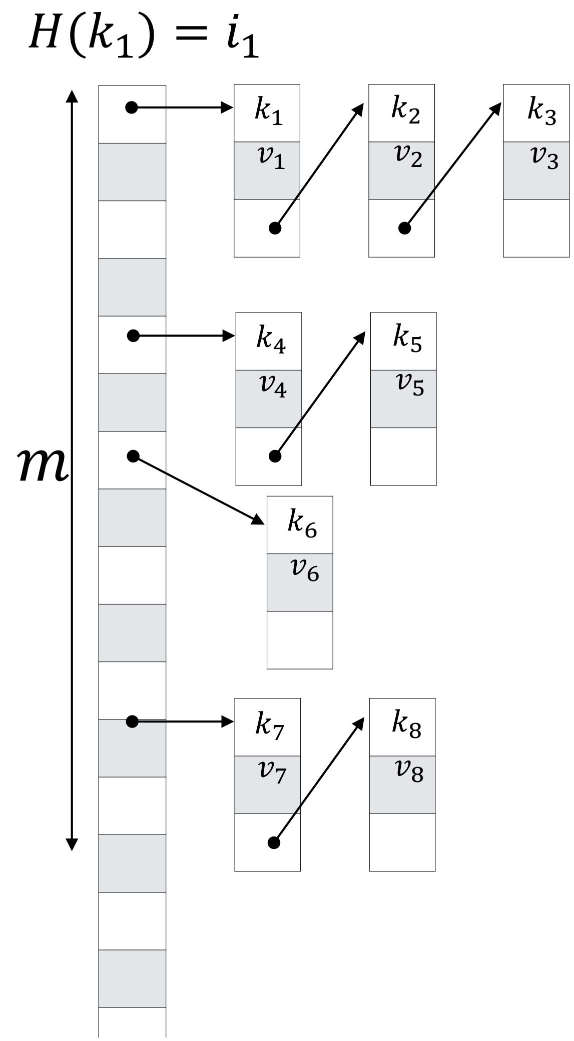
What can we remove from a hash table?



# Reducing a hash table

What can we remove from a hash table?

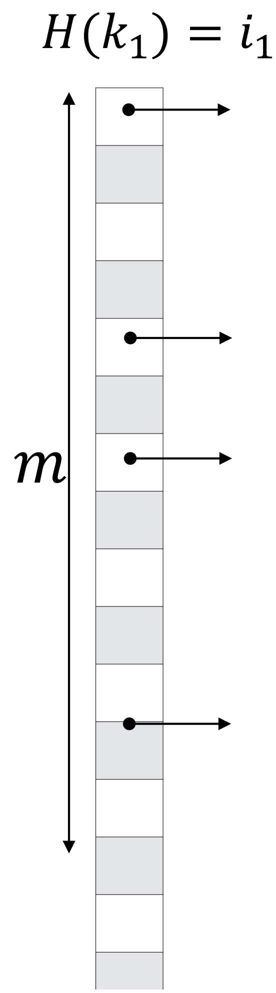
Take away values



# Reducing a hash table

What can we remove from a hash table?

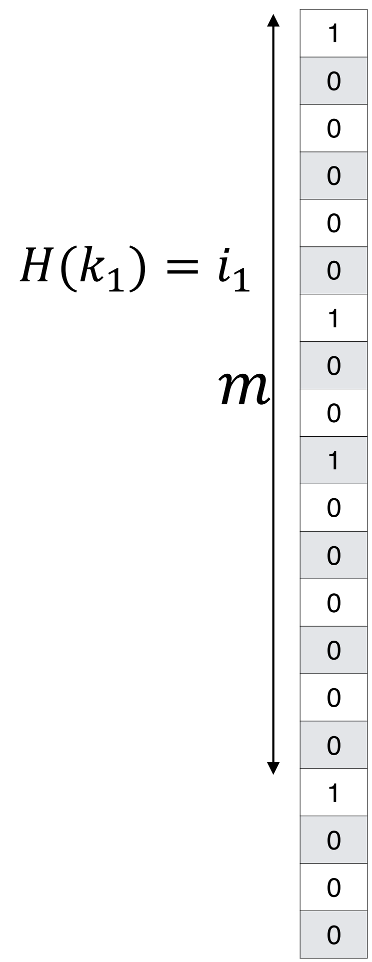
Take away values and keys



# Reducing a hash table

What can we remove from a hash table?

Take away values and keys



# Bloom Filter: Insertion

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

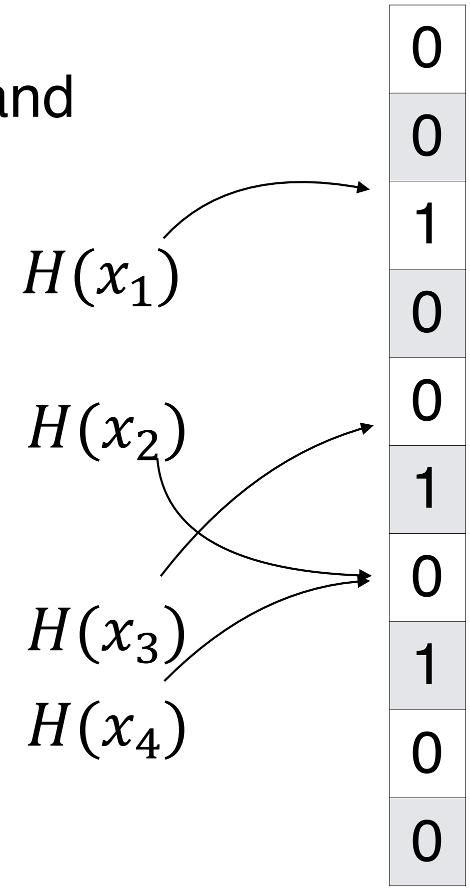
**h(k) = k % 7**

| | |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 2 | 0 |
| 3 | 0 |
| 4 | 0 |
| 5 | 0 |
| 6 | 0 |

# 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 |
| 2 | 1 |
| 3 | 0 |
| 4 | 1 |
| 5 | 0 |
| 6 | 1 |

`_delete(13)`

`_delete(29)`

# 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)`

`_find(20)`

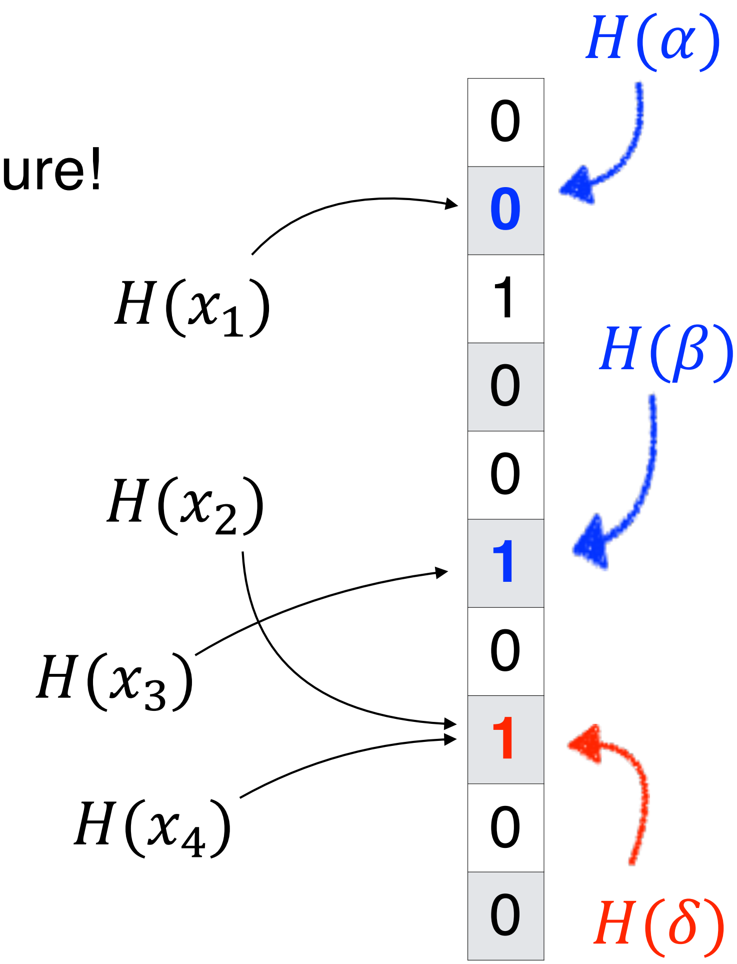
`_find(3)`

# Bloom Filter: Search

The bloom filter is a *probabilistic* data structure!

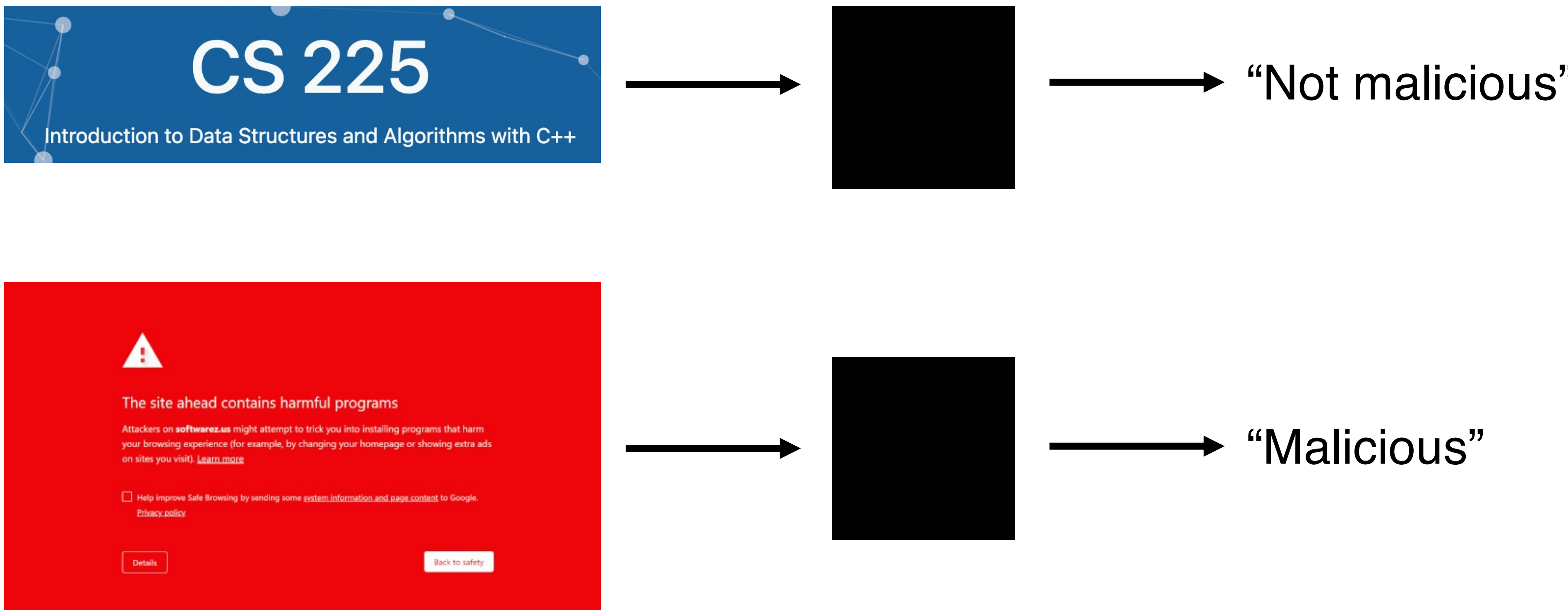
If the value in the BF is 0:

If the value in the BF is 1:



# Probabilistic Accuracy: Malicious Websites

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



# Probabilistic Accuracy: Malicious Websites

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

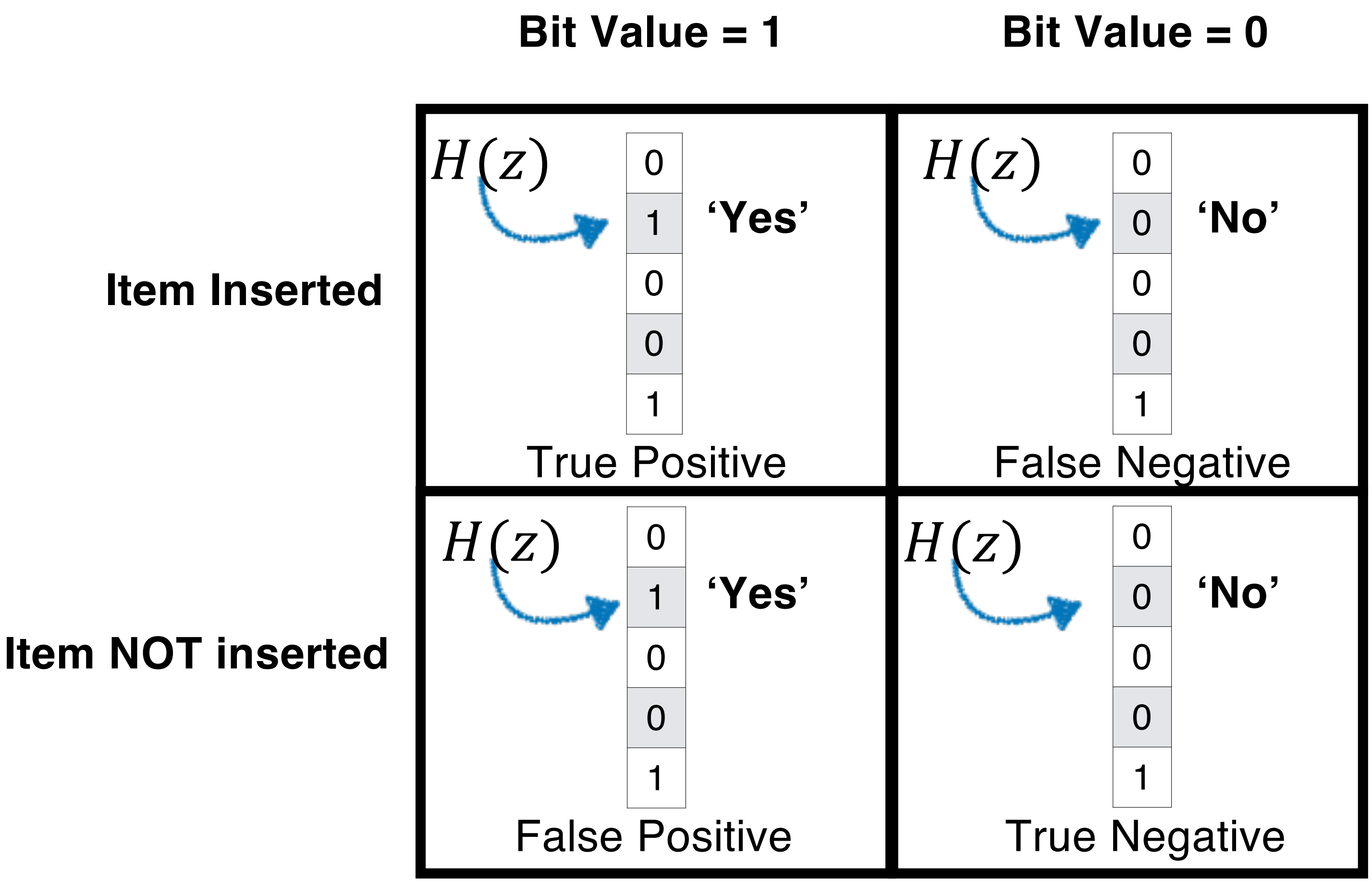
True Positive:

False Positive:

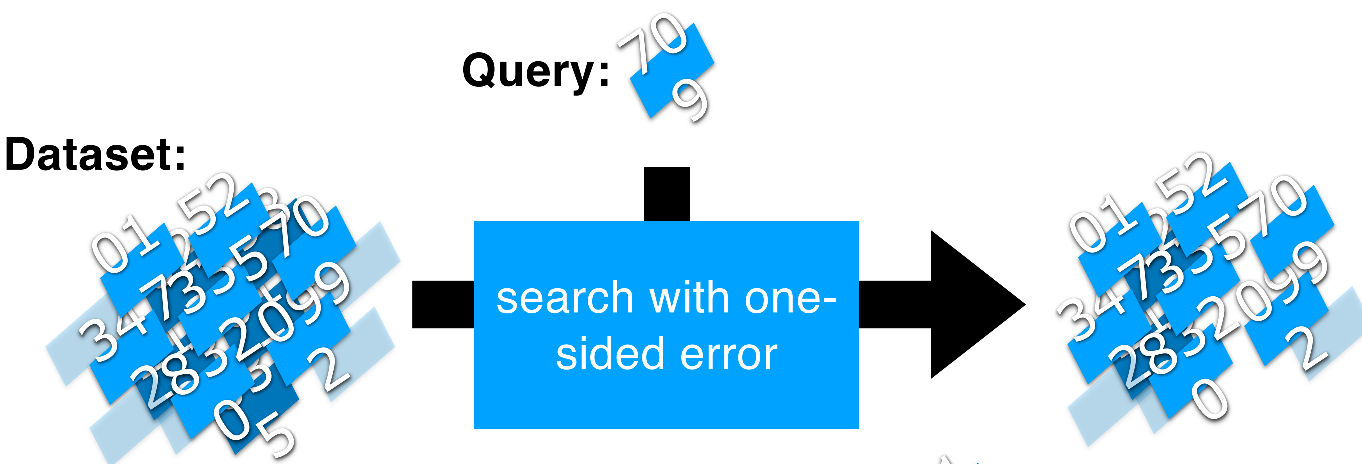
False Negative:

True Negative:

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



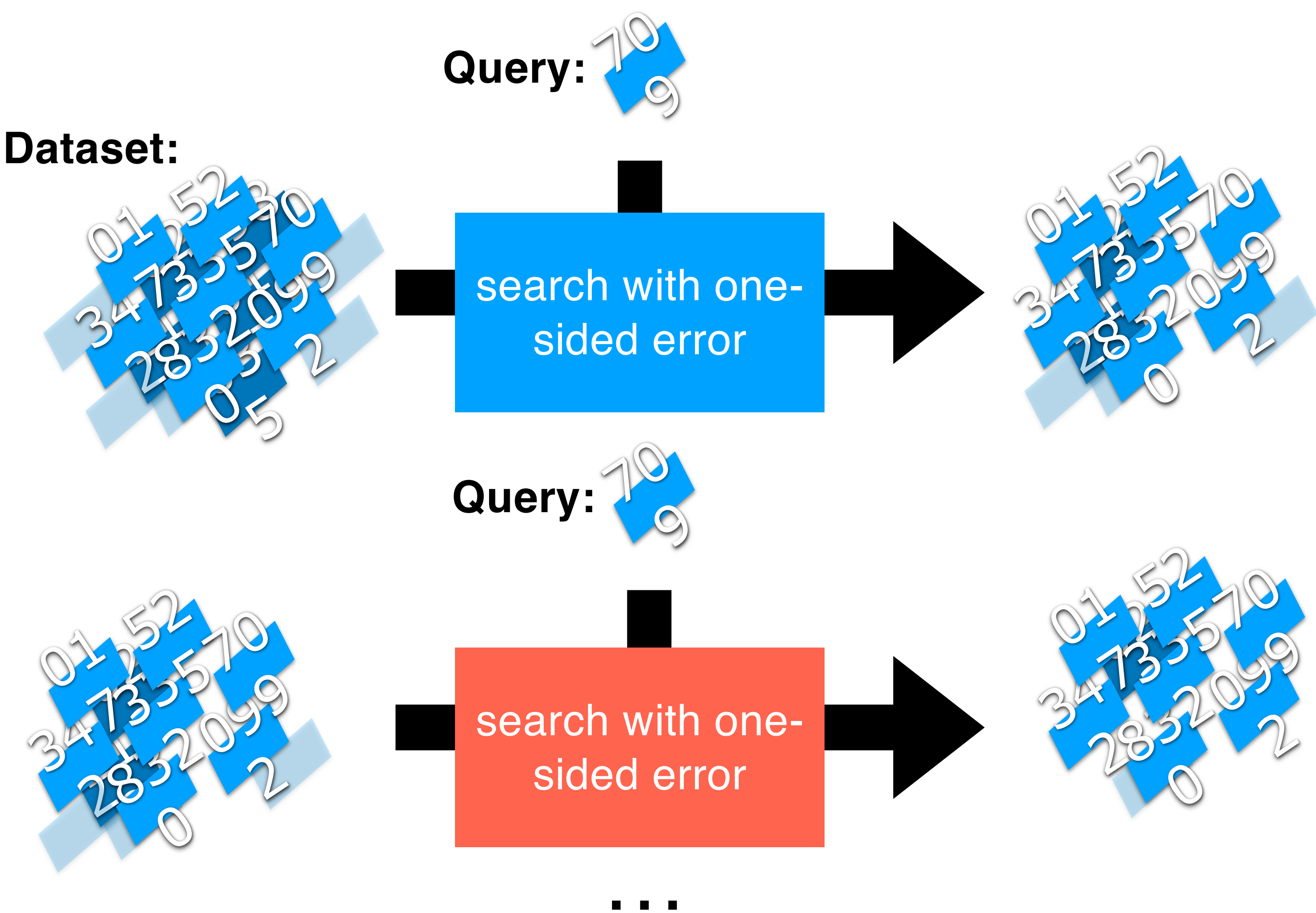
# Probabilistic Accuracy: One-sided error




We will get some False Positives:

We will NEVER have a False Negative:

# Probabilistic Accuracy: One-sided error

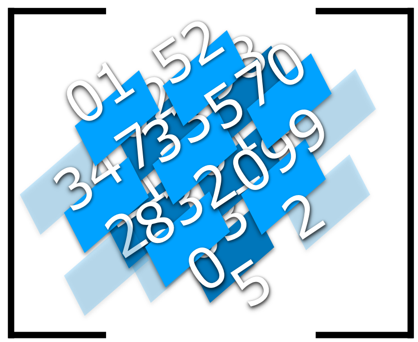


# Bloom Filter: Repeated Trials

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

$h_1$

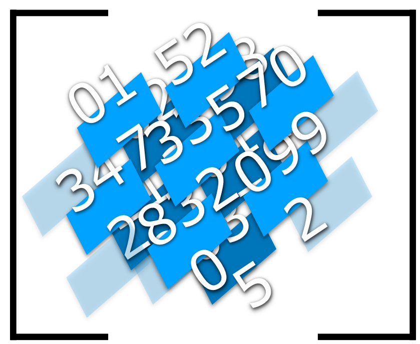
| |
|---|
| 0 |
| 1 |
| 0 |
| 1 |
| 0 |
| 0 |
| 0 |
| 1 |
| 0 |
| 1 |
| 1 |
| 0 |
| 1 |
| 0 |
| 1 |
| 1 |
| 0 |
| 1 |
| 0 |
| 1 |



# Bloom Filter: Repeated Trials

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

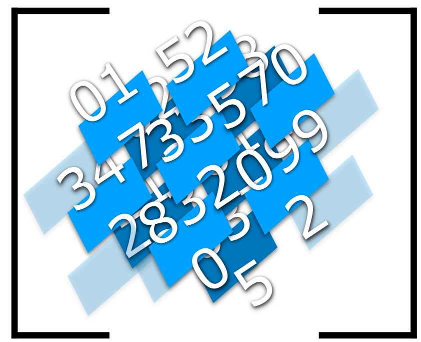
| $h_1$ | $h_2$ |
|---|---|
| 0 | 0 |
| 1 | 0 |
| 0 | 0 |
| 1 | 1 |
| 0 | 0 |
| 0 | 0 |
| 0 | 0 |
| 1 | 1 |
| 0 | 0 |
| 1 | 1 |
| 1 | 0 |
| 0 | 0 |
| 1 | 1 |
| 0 | 1 |
| 1 | 1 |
| 1 | 0 |
| 0 | 0 |
| 1 | 1 |
| 0 | 0 |
| 1 | 0 |



# 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

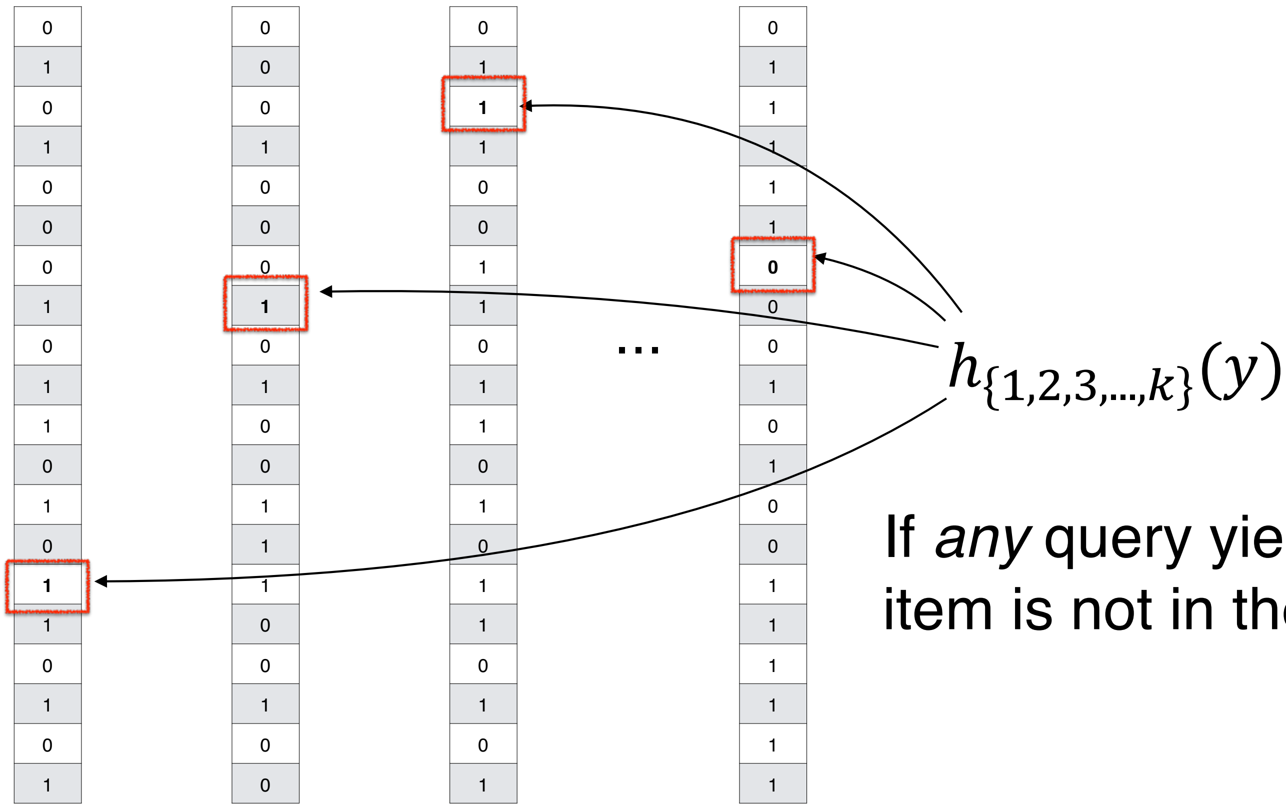
| $h_1$ | $h_2$ | $h_3$ | ⋯ | $h_k$ |
|---|---|---|---|---|
| 0 | 0 | 0 | | 0 |
| 1 | 0 | 1 | | 1 |
| 0 | 0 | 1 | | 1 |
| 1 | 1 | 1 | | 1 |
| 0 | 0 | 0 | | 1 |
| 0 | 0 | 0 | | 1 |
| 0 | 0 | 1 | | 0 |
| 1 | 1 | 1 | | 0 |
| 0 | 0 | 0 | | 0 |
| 1 | 1 | 1 | | 1 |
| 1 | 0 | 1 | | 0 |
| 0 | 0 | 0 | | 1 |
| 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 |

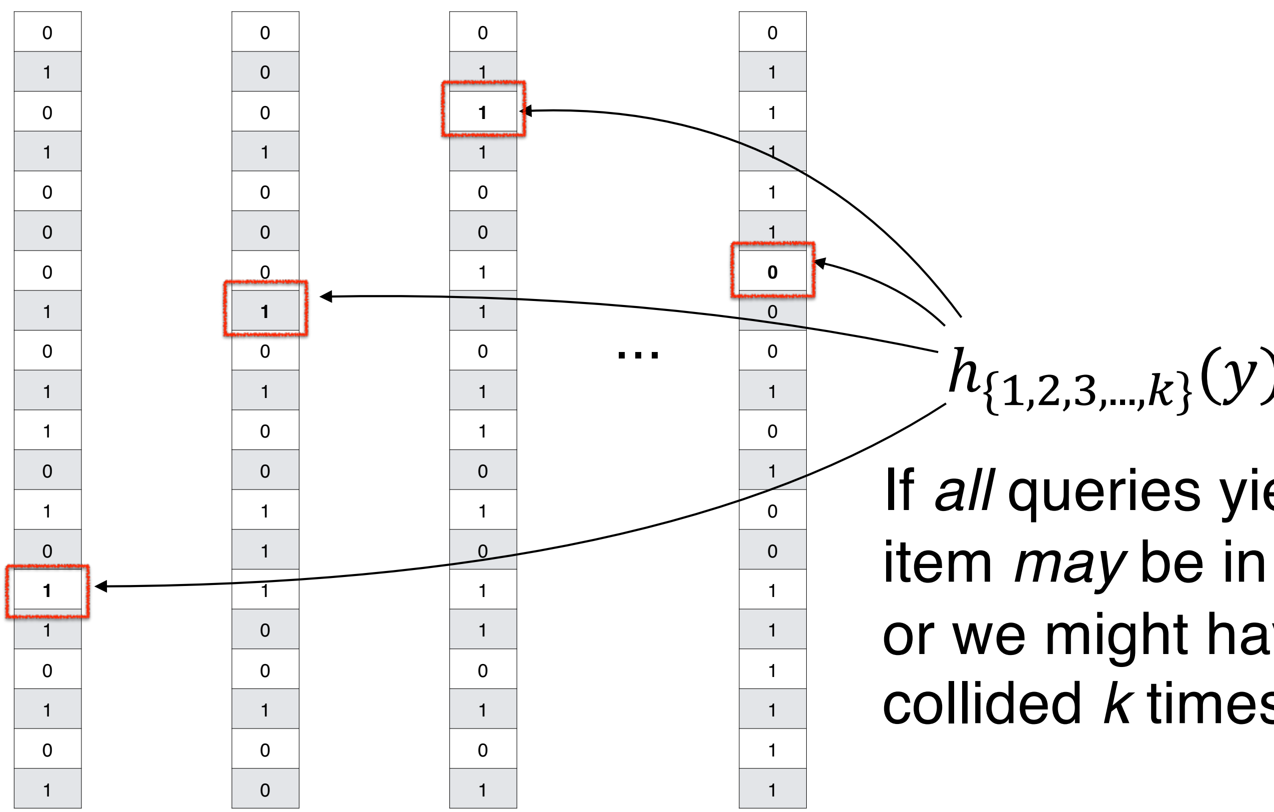
$$h_{\{1,2,3,\ldots,k\}}(y)$$

# Bloom Filter: Repeated Trials




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

# Bloom Filter: Repeated Trials




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

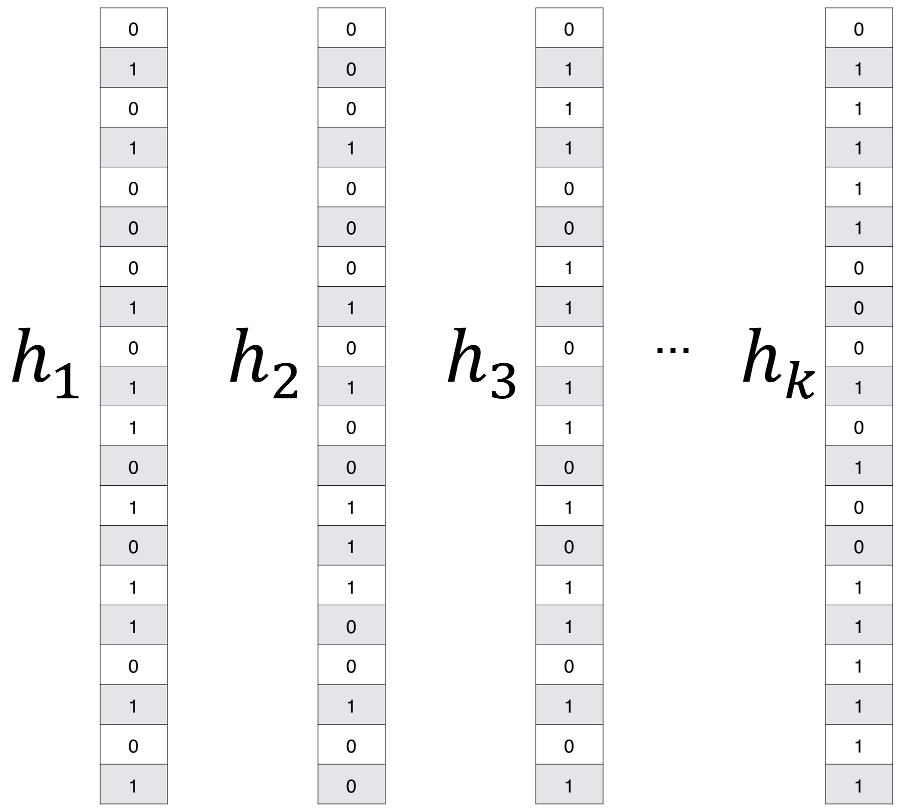
# Bloom Filter: Repeated Trials

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?

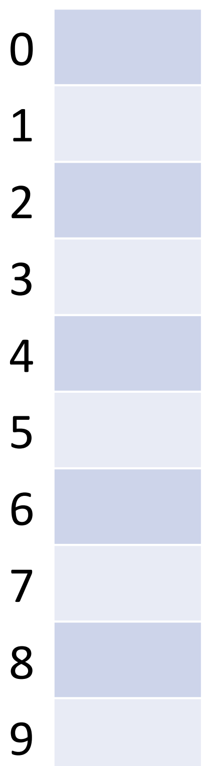
# Bloom Filter: Repeated Trials

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



# Bloom Filter: Repeated Trials

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




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

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

# 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)
```

```
_find(16)
```

# Bloom Filter

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: ________________

Delete is not possible (yet)!

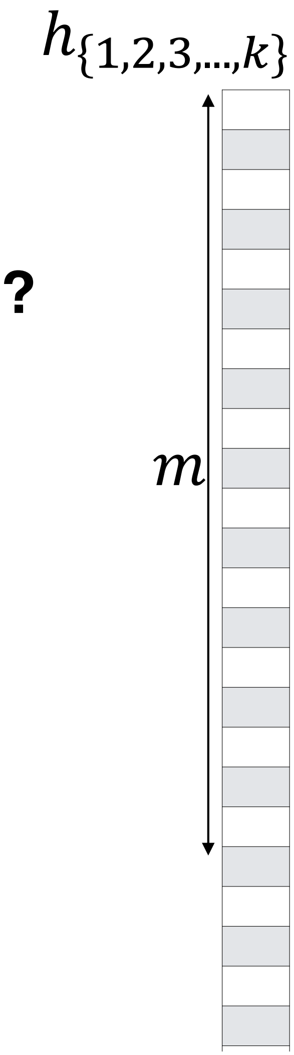
$$H = \{h_1, h_2, \ldots, h_k\}$$

| |
|---|
| 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

**What is our expected FPR after $n$ objects are inserted?**

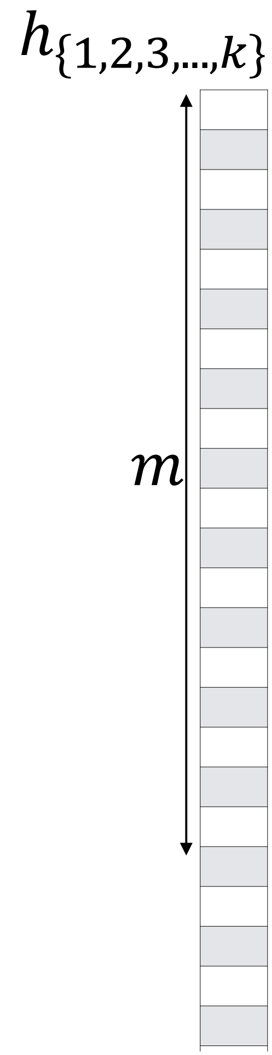


# Bloom Filter: Error Rate

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?

Same probability given $k$ SUHA hash function?

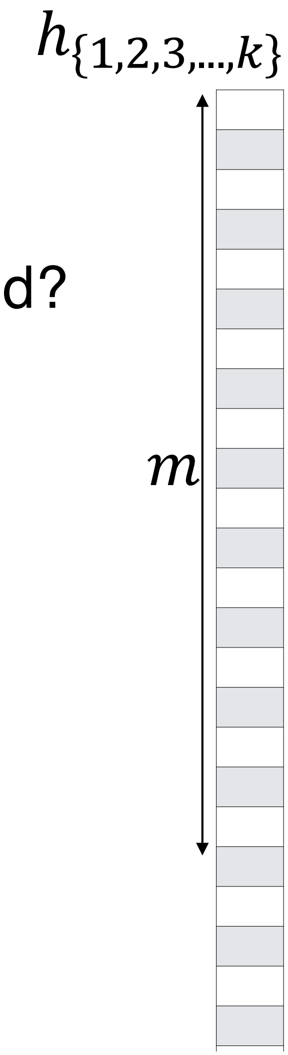


# Bloom Filter: Error Rate

Given bit vector of size $m$ and $k$ SUHA hash function

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

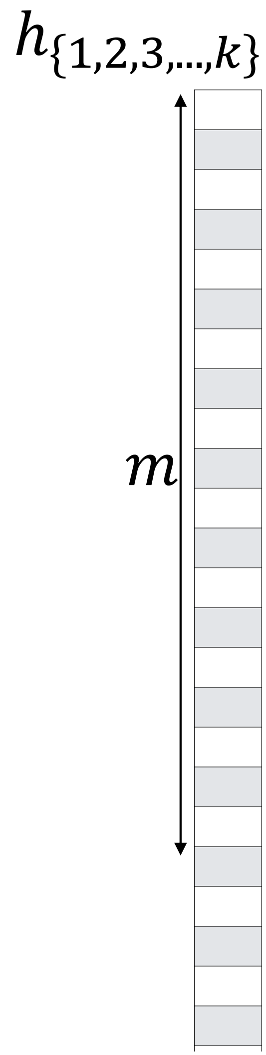
After $n$ objects are inserted?



# 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?



# Bloom Filter: Error Rate

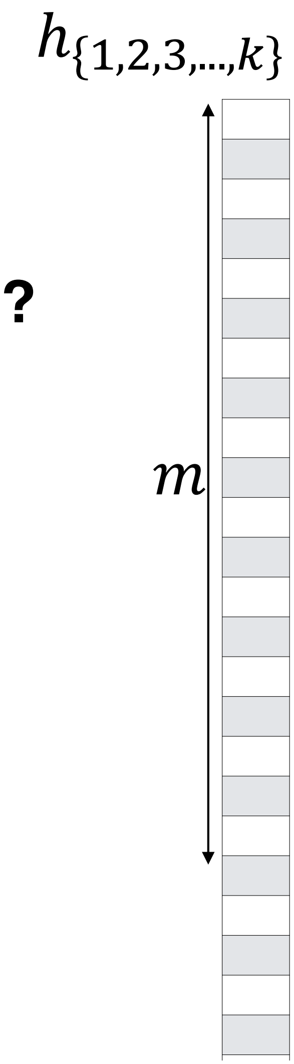
Given bit vector of size $m$ and $k$ SUHA hash function

**What is our expected FPR after $n$ objects are inserted?**

The probability my bit is 1 after $n$ objects inserted

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

The number of [assumed independent] trials



# Bloom Filter: Error Rate

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

**To minimize the FPR, do we prefer…**

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

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