CS2020
Data Structures and Algorithms

Welcome!

Coding Quiz

Problem 1: Inversions

Expected solution:

```
count = 0;
for every (i < j):
    if (A[i] > A[j]) then count++;
```

Running time: $O(n^2)$

Careful: empty list, one element list, array bounds

Coding Quiz

Problem 1: Inversions

Alternate solution: MergeSort

Coding Quiz

Problem 2: FriendSet

- Expected solution:
  - Store names in an array[100].
  - Unsorted.
  - Intersection: $O(n^2)$ double loop.

Coding Quiz

Problem 2: FriendSet

- Basic solution:
  - Store names in an array[100].
  - Unsorted.
  - Add: check for repeats.
  - Intersection: $O(n^2)$ double loop.

Coding Quiz

Problem 2: FriendSet

- Better solution:
  - Store names in an array[100].
  - **Sorted.**
  - Add: O(n).
  - Intersection: O(n) merge.

Coding Quiz

Problem 2: FriendSet

- Better solution:
  - Store names in an array[100].
  - **Sorted order, spread out at random.**
  - **Re-spread when needed.**
  - Add: $O(\log^2 n)$ amortized.
  - Intersection: $O(n)$ merge.

Coding Quiz

Problem 2: FriendSet

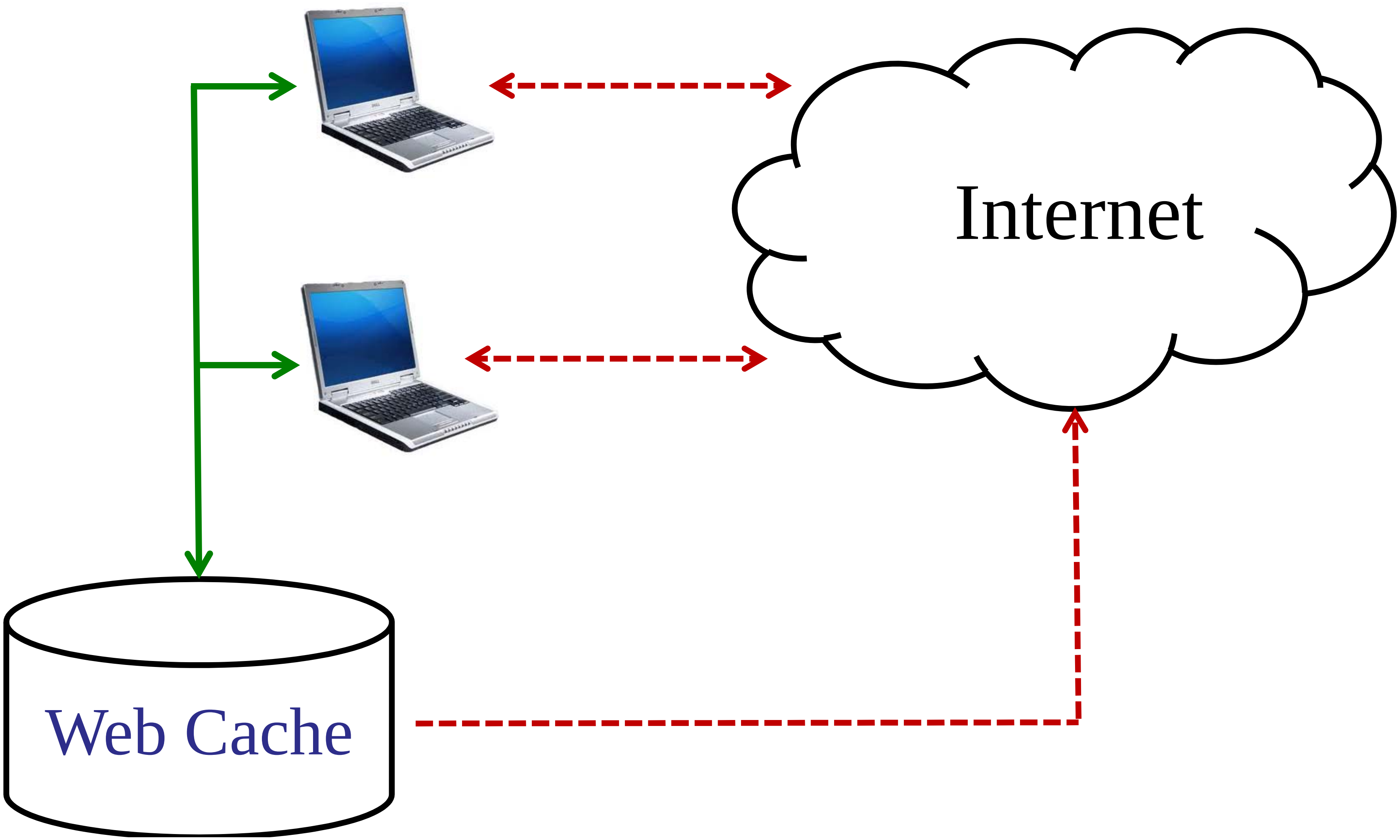
- Even better solution:
  - Store names in a hash table.
  - add: O(1)
  - intersection: O(n)

Coding Quiz

Problem 2: UserDB

- Expected solution: HashMap or ArrayList
  - Each user contains a name and an ISet.
  - Maintain array/hashmap of users.

Web Caching

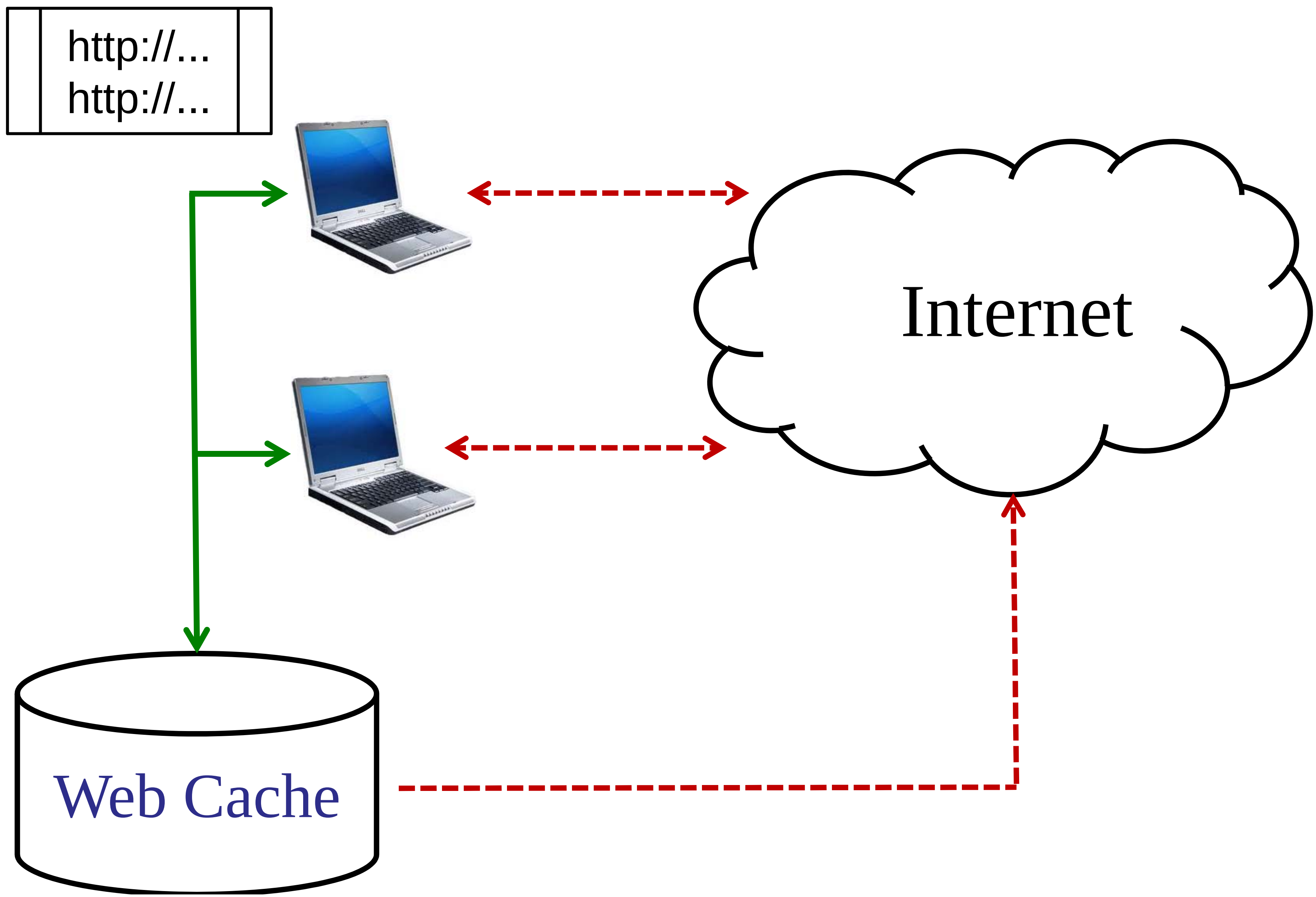


Web Cache

- List of cached URLs:
  - http://cs2020.ddns.comp.nus.edu.sg/
  - http://www.comp.nus.edu.sg
  - http://gmail.com
  - Etc.

- Idea: Web Proxy occasionally sends list of cached URLs to nearby computers.

Web Caching



Web Cache

- List of cached URLs:
  - http://cs2020.ddns.comp.nus.edu.sg/
  - http://www.comp.nus.edu.sg
  - http://gmail.com
  - Etc.

- Idea: Web Proxy occasionally sends list of cached URLs to nearby computers.

- Problem: list of URLs is really, really big.

Web Cache

Use a hash table!

- Only store/send *m* bits!

hash("`www.gmail.com`")

hash("`www.apple.com`")

hash("`www.microsoft.com`")

hash("`www.nytimes.com`")

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

Web Cache

What happens on collision?

hash(“`www.gmail.com`”)

hash(“`www.apple.com`”)

hash(“`www.microsoft.com`”)

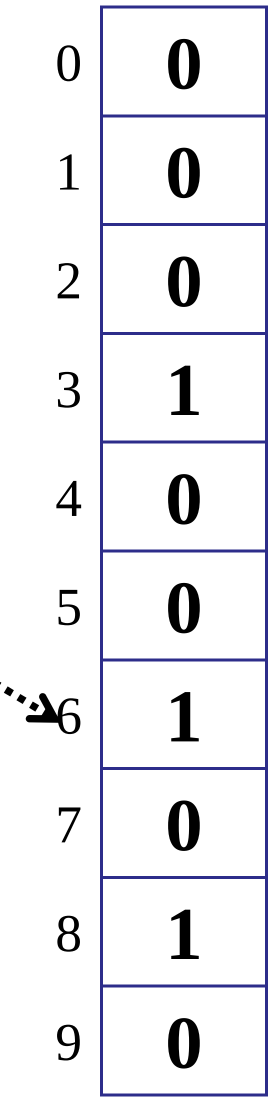
hash(“`www.nytimes.com`”)

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

Web Cache

Lookup operation:

hash(“`www.microsoft.com`”)

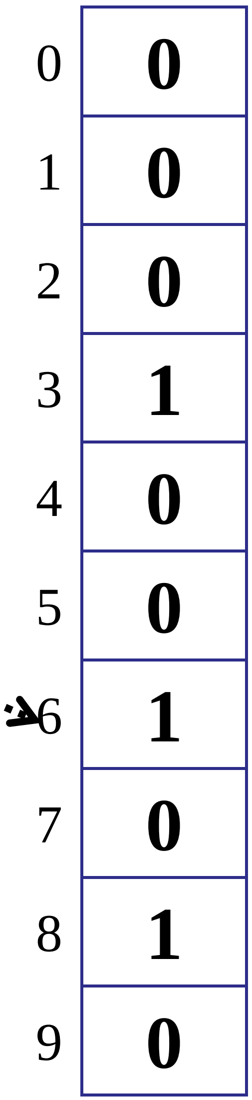


If the URL is in the web cache, it will always report **true**.
(No false negatives.)

Web Cache

Lookup operation:

hash(“`www.rugby.com`”)



Even if the URL is NOT in the web cache,
it may *sometimes* report **true**.
(False positives.)

Web Cache Analysis

Probability of a false negative: 0

Probability of a false positive:

$$\left(1 - \frac{1}{m}\right)^{n} \leq \left(\frac{1}{2}\right)^{n/m}$$

Web Cache Analysis

Probability of a false negative: 0

Probability of no false positive: (simple uniform hashing assumption)

$$\left(1 - \frac{1}{m}\right)^{n} \leq \left(\frac{1}{2}\right)^{n/m}$$

Probability of a false positive:

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

Web Cache Analysis

Assume you want:

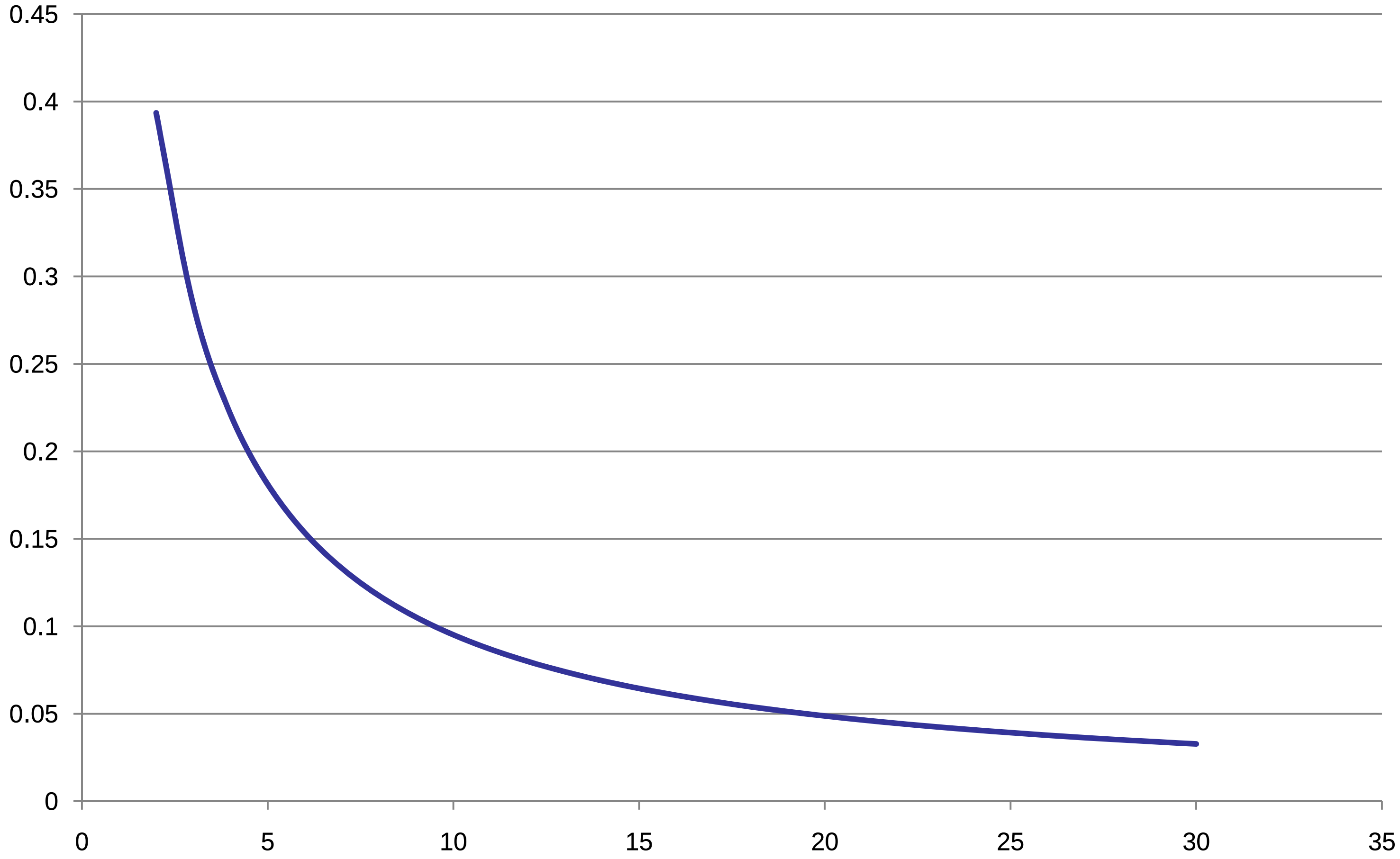
- probability of false positives < p
  - Example: at most 1% of queries return false positive.

    $p = .01$

- Need: $\frac{n}{m} \leq \log\left(\frac{1}{1-p}\right)$

  - Example: $m >= (68.97)n$

Web Cache



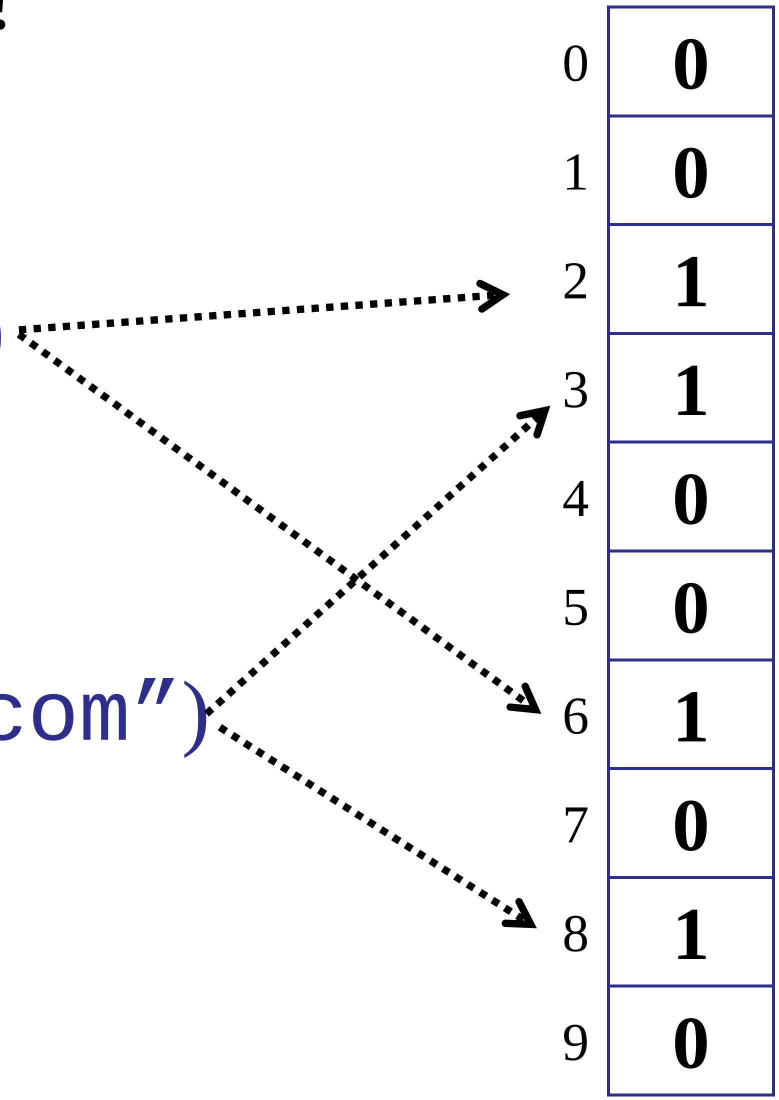
probability of false positive vs (m/n)

Web Cache

Idea 2: use 2 hash functions!

hash(“`www.gmail.com`”)

hash(“`www.microsoft.com`”)



Web Cache

Idea 2: use 2 hash functions!

hash(“`www.gmail.com`”)

insert(URL)

$k_1 = \text{hash}_1(\text{URL});$

$k_2 = \text{hash}_2(\text{URL});$

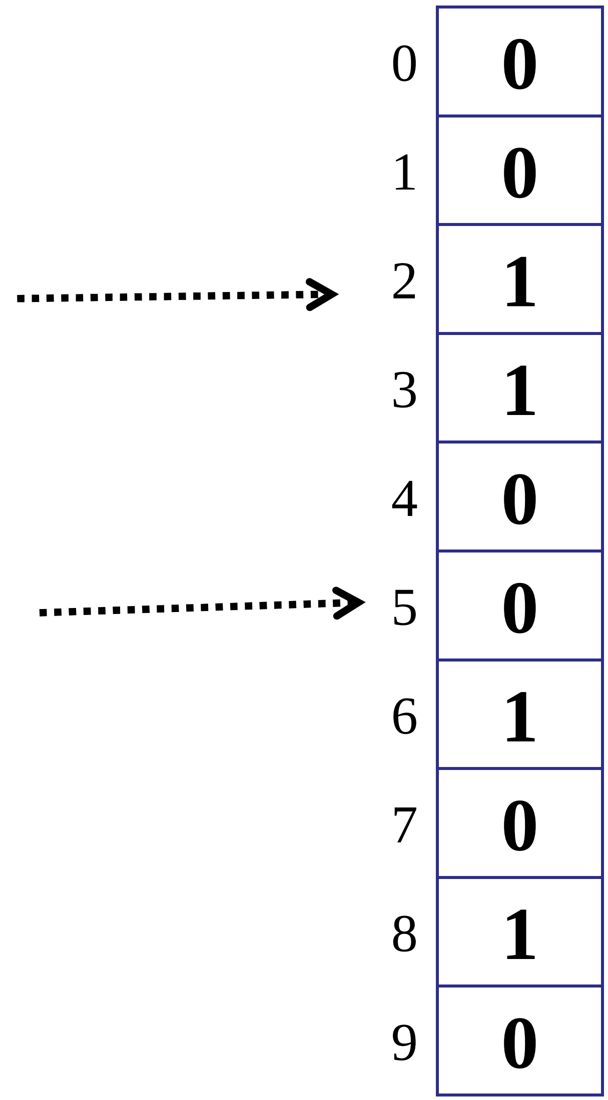
$T[k_1] = 1;$

$T[k_2] = 1;$

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

Web Cache

Idea 2: use 2 hash functions!



query(URL)

k_1 = hash$_1$(URL);

k_2 = hash$_2$(URL);

if (T[k_1] && T[k_2])

return true;

else return false;

Web Cache

Idea 2: use 2 hash functions!

Trade-off:

- Each item takes more “space” in the table.
- Requires two collisions for a false positive.

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

Web Cache Analysis

Probability a given bit is 0:

$$\left(1-\frac{1}{m}\right)^{2n} \leq \left(\frac{1}{2}\right)^{2n/m}$$

Probability of a false positive :

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

Web Cache Analysis

Assume you want:

- probability of false positives $< p$
  - Example: at most 1% of queries return false positive.

    $p = .01$

- Need: $$\frac{n}{m} \leq \frac{1}{2}\log\left(\frac{1}{1-p^{1/2}}\right)$$
  - Example: $m >= (13.36)n$

Web Cache Analysis

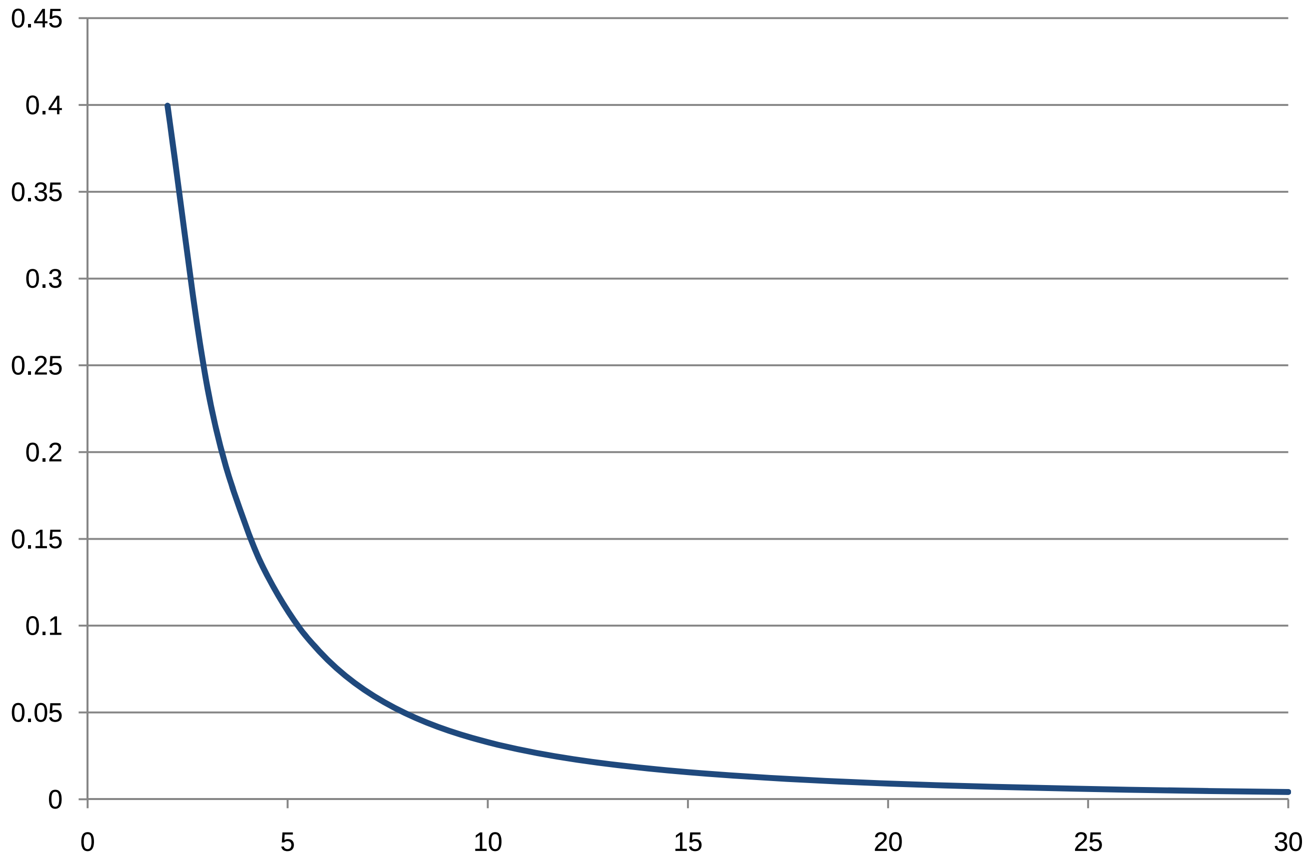
Beware:

- Sloppy math on the last few slides!

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

- Actual results: m > 19n

Web Cache

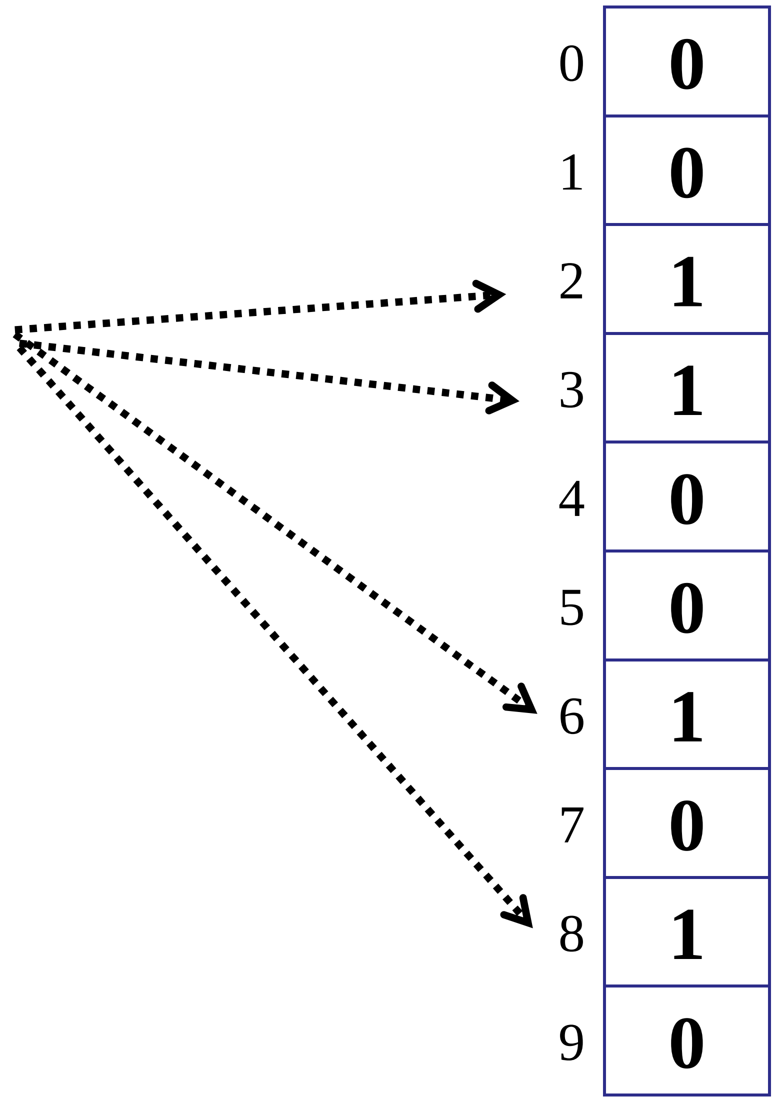


False positives rate vs. (m/n)

Bloom Filters

Use *k* hash functions!

hash(“`www.gmail.com`”)



Bloom Filter Analysis

Probability a given bit is 0:

$$\left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$$

Bloom Filter Analysis

Probability a given bit is 0:

$$\left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$$

Probability of a collision at one spot:

$$1 - e^{-kn/m}$$

Web Cache Analysis

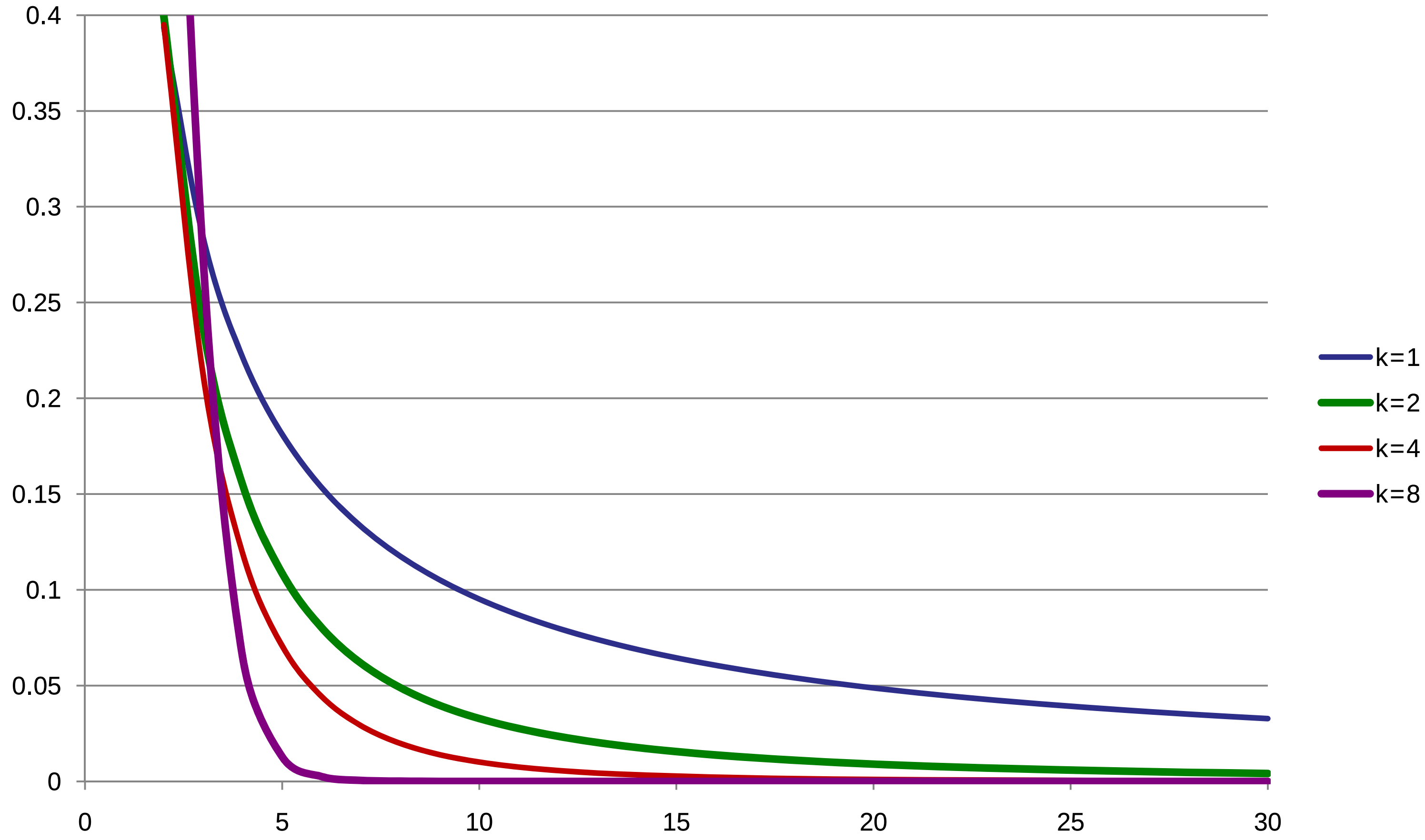
Probability of a collision at one spot:

$$1 - e^{-kn/m}$$

Probability of a collision at all *k* spots:

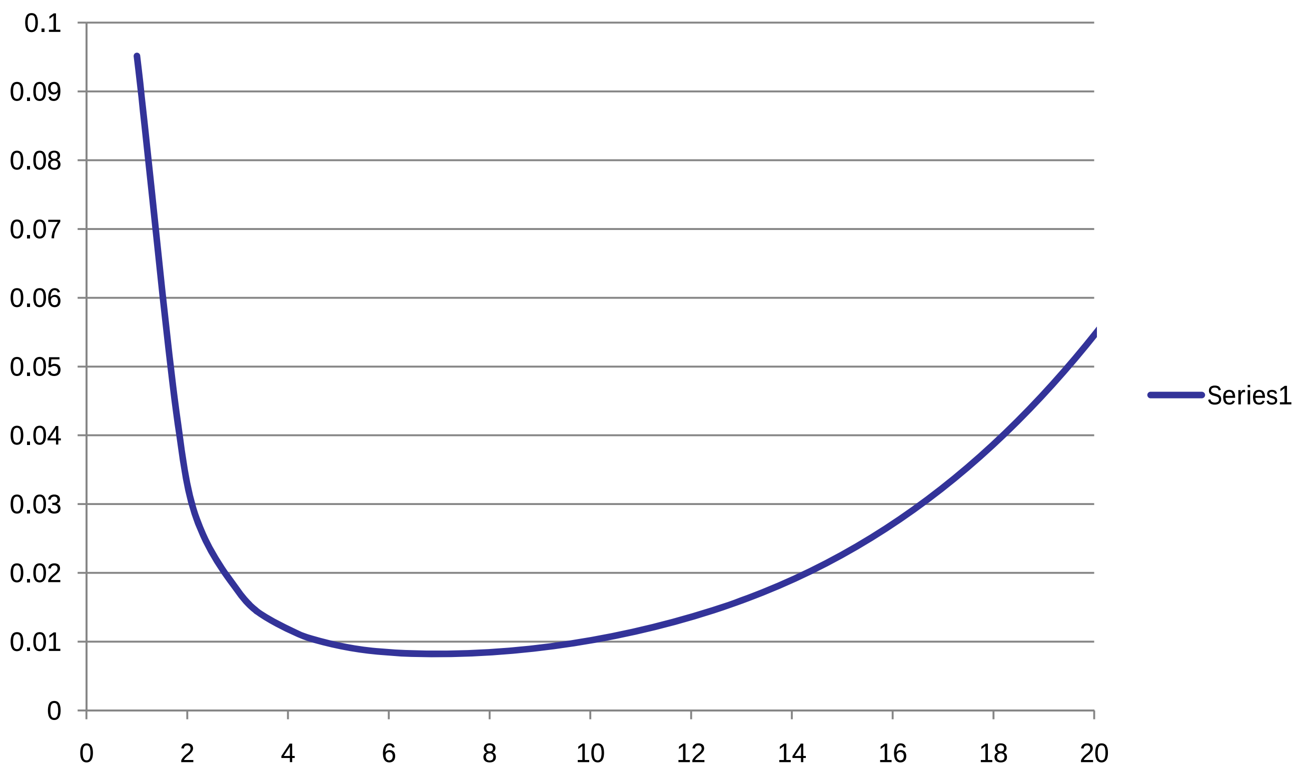
$$\left(1 - e^{-kn/m}\right)^k$$

Bloom Filter



false positive rate vs. (m/n)

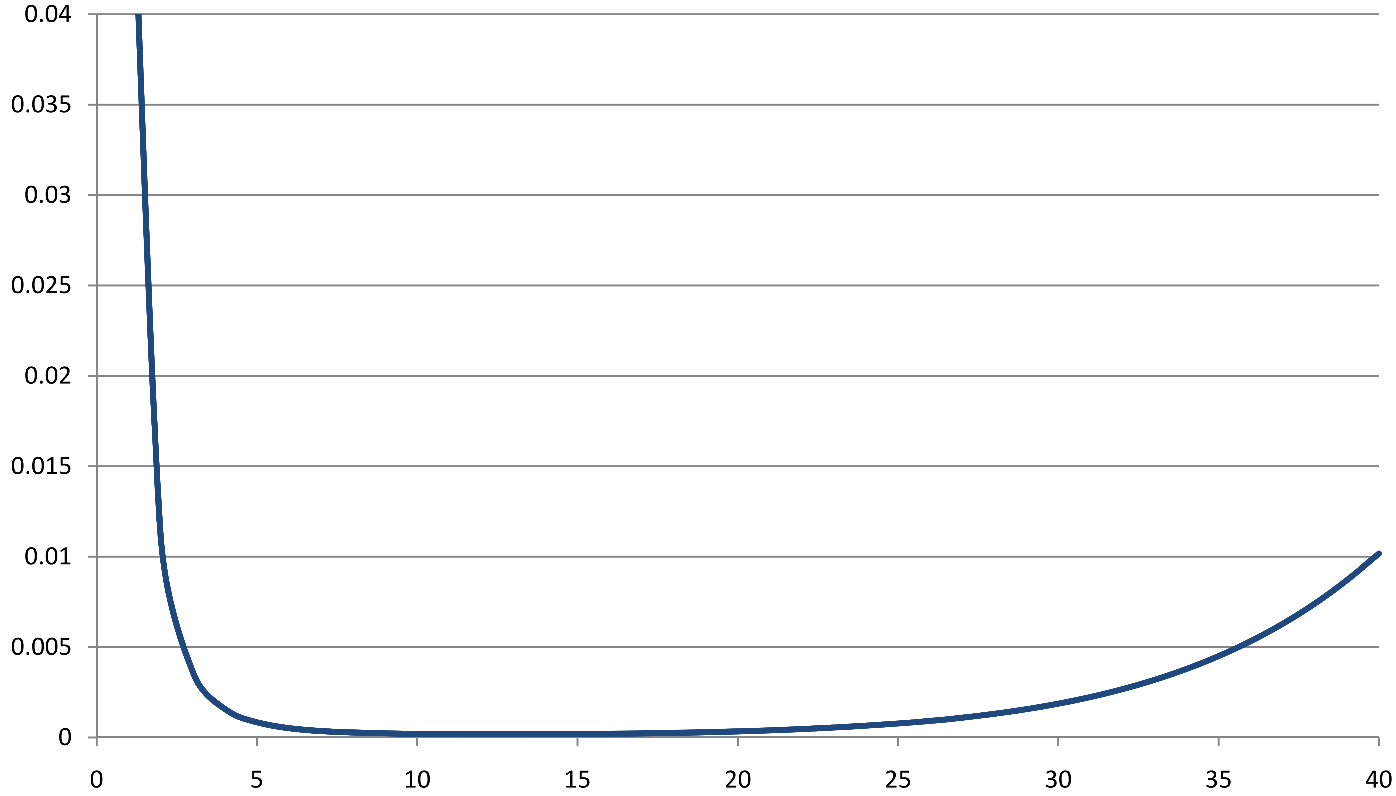
Bloom Filter



false positive rate vs *k*

m = 10n

Bloom Filter



false positive rate vs *k*

m = 18n

Bloom Filter

What is the optimal value of *k*?

- Probability of false positive:

$$\left(1 - e^{-kn/m}\right)^k$$

- Choose: $k = \frac{m}{n} \ln 2$

- Error probability: 2^{-k}

Bloom Filters

What about deleting a URL from the proxy list?

- Store counter instead of 1 bit.
- On insert: increment.
- On delete: decrement.

Bloom Filters

Implementation of Set ADT:

- insert: O(k)
- delete: O(k)
- query: O(k)

Bloom Filters

Implementation of Set ADT:

- intersection
  - Bitwise AND of two Bloom filters: O(m)
- union
  - Bitwise OR of two Bloom filters: O(m)

Other applications

- Chrome browser safe-browsing
  - Maintains list of “bad” websites.
  - Occasionally retrieves updates from google server.

- Spell-checkers
  - Storing all words takes a lot of space.
  - Instead, store a Bloom filter of the words.

- Weak password dictionaries

Summary

When to use Bloom Filters?

- Storing a set of data.
- Space is important.
- False positives are ok.

Interesting trade-offs:

- Space
- Time
- Error probability