Balls and Bins, Hashing and Filters

Balls and Bins

- Consider the process of throwing m balls into n bins
- Each ball is thrown into a uniformly random bin, independent of other balls
- What does the distribution of balls in bins looks like?
- This process helps analyze many applications, e.g. hashing, distributed load balancing (assigning requests to servers), etc.

Balls & Bins: Birthday Paradox

- **Question.** How large must m be that it is likely there is a bin with at least two balls?
- Probability that all m balls fall into different bins:
- $$= \left(1 - \frac{1}{n}\right) \cdot \left(1 - \frac{2}{n}\right) \cdot \left(1 - \frac{3}{n}\right) \ldots \left(1 - \frac{m-1}{n}\right)$$
- $$= \prod_{j=1}^{m-1} \left(1 - \frac{j}{n}\right) = \prod_{j=1}^{m-1} \left(1 - \frac{j}{n}\right) = \prod_{j=1}^{m-1} e^{-j/m} \approx e^{-m^2/2n}$$
- Useful inequality $1 - x \approx e^{-x}$ when x is small
- $m \approx \sqrt{2n \ln 2}$ for probability to be $1/2$; for $n = 365$, we get $m = 22.49$
- Thus, with around 23 people in this class, we have a 50% chance of two people having the same birthday

Useful approximations:
$(1 + x) \approx e^{x}$
$(1 - x) \approx e^{-x}$

Balls and Bins: Coupon Collector

- **Question.** How many balls m should be thrown until all bins have at least one ball?
- $\Pr[\text{bin } b \text{ is empty}] = \left(1 - \frac{1}{n}\right)^m \approx e^{-m/n}$

Useful approximations:

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

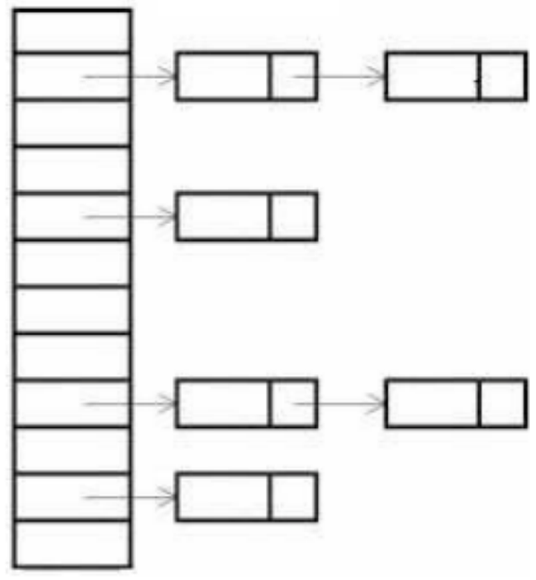
- $\Pr[\text{exists an empty bin}] = \Pr[\cup_{b=1}^{n} \text{bin } b \text{ is empty}]$

$$\leq \sum_{b=1}^{n} \Pr[\text{bin } b \text{ is empty}] = n \cdot e^{-m/n}$$

- E.g. $m = 3n \ln n$, the probability is at most $\frac{1}{n^2}$
- That is, if we buy $O(n \log n)$ packs, we get all Pokemons **w.h.p.**

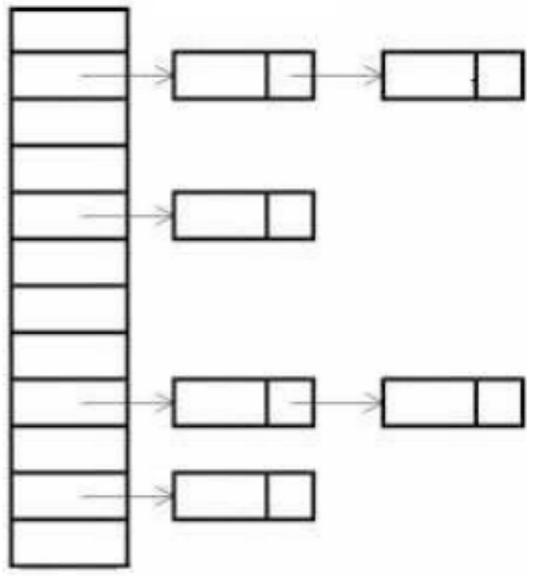
Balls and Bins: Hashing

- **Dictionary problem.** Store a subset S of size m from a large universe U so as to support the following operations efficiently for any x:
  - insert(x)
  - delete(x)
  - search(x)
- Key features: U is very large and set S is not known ahead of time
- A hash function $h : U \rightarrow [1, n]$ can be thought of placing items from the universe into n bins
- **Chain hashing.**
  Elements in a particular bin are stored in a linked list



Balls and Bins: Hashing

- Assume **uniform random hash function**, that is, location of each element is uniformly distributed independent of others
    - For each $x,$ the probability that $h(x) = j$ is $1/n$ and $h(x), h(y)$ are independent for any $x \neq y$
- Very strong and impractical assumption (but we can get pretty close to such hash functions in practice, will discuss later)



Balls and Bins: Hashing

- **Question.** Expected number of balls in a particular bin b?
- Let X_i denote the indicator random variable that ball i lands in bin b
- $X = \sum_{i=1}^{m} X_i$ denotes the number of balls in bin b
- $E[X] = \sum_{i=1}^{m} E[X_i] = \sum_{i=1}^{m} \Pr[X_i = 1] = \sum_{i=1}^{m} \frac{1}{n} = \frac{m}{n}$
- Expected search cost in a hash table with chaining?
    - $O(1 + E[X]) = O(1 + m/n) = O(1 + \alpha)$ where α is sometimes called the load factor
    - As long as table size n is constant factor of number of elements m, expected cost of hashing with chaining is $O(1)$

Hashing: With High Probability Bounds

- What is the worst case search time in a chained hash table?
- Assume $m = n$ for simplicity
- **Lemma.** If n balls are thrown independently and uniformly into n bins, then with high probability, the fullest bin contains $O(\frac{\log n}{\log \log n})$

- To prove this we first answer a few helpful questions
- What is the probability that a bin has exactly k balls?
  - $\binom{n}{k}\left(\frac{1}{n}\right)^k\left(1-\frac{1}{n}\right)^{n-k}$
- Let X be the number of balls in a bin, then X has a binomial distribution

Hashing: With High Probability Bounds

- What is the worst case search time in a chained hash table?
- Assume $m = n$ for simplicity
- **Lemma.** If n balls are thrown independently and uniformly into n bins, then with high probability, the fullest bin contains $O(\frac{\log n}{\log \log n})$
  - The probability that a bin has at least k balls? $\leq \binom{n}{k} \left(\frac{1}{n}\right)^k$
- Select k balls, and multiply with probability they land in the bin, the other $n - k$ balls can land anywhere

Hashing: With High Probability Bounds

- **Lemma.** If n balls are thrown independently and uniformly into n bins, then with high probability, the fullest bin contains $O(\frac{\log n}{\log \log n})$

- The probability that a bin has at least k balls? $\leq \binom{n}{k}\left(\frac{1}{n}\right)^k$

$$\leq \left(\frac{en}{k}\right)^k \frac{1}{n^k} = \left(\frac{e}{k}\right)^k$$

- Recall **death-bed formula** $\left(\frac{y}{x}\right)^x \leq \binom{y}{x} \leq \left(\frac{ey}{x}\right)^x$

- Let $k = \frac{4 \ln n}{\ln \ln n}$

Hashing: With High Probability Bounds

- **Lemma.** If n balls are thrown independently and uniformly into n bins, then with high probability, the fullest bin contains $O(\frac{\log n}{\log \log n})$
- Probability that a bin has at least $\frac{4 \ln}{\ln \ln n}$ balls

$$\leq \left(\frac{e}{k}\right)^k = \left(\frac{e \ln \ln n}{4 \ln n}\right)^{\frac{4 \ln n}{\ln \ln n}} \leq \left(\frac{e \ln \ln n}{\ln n}\right)^{\frac{4 \ln n}{\ln \ln n}} = e^{\frac{4 \ln n}{\ln \ln n} \cdot \ln \frac{\ln \ln n}{\ln n}}$$

$$= e^{\frac{4 \ln n}{\ln \ln n} \cdot (\ln \ln \ln n - \ln \ln n)}$$

$$= e^{-4 \ln n + \frac{4 \ln n \ln \ln \ln n}{\ln \ln n}} \leq e^{-3 \ln n} = \frac{1}{n^3} \text{ for sufficiently large } n$$

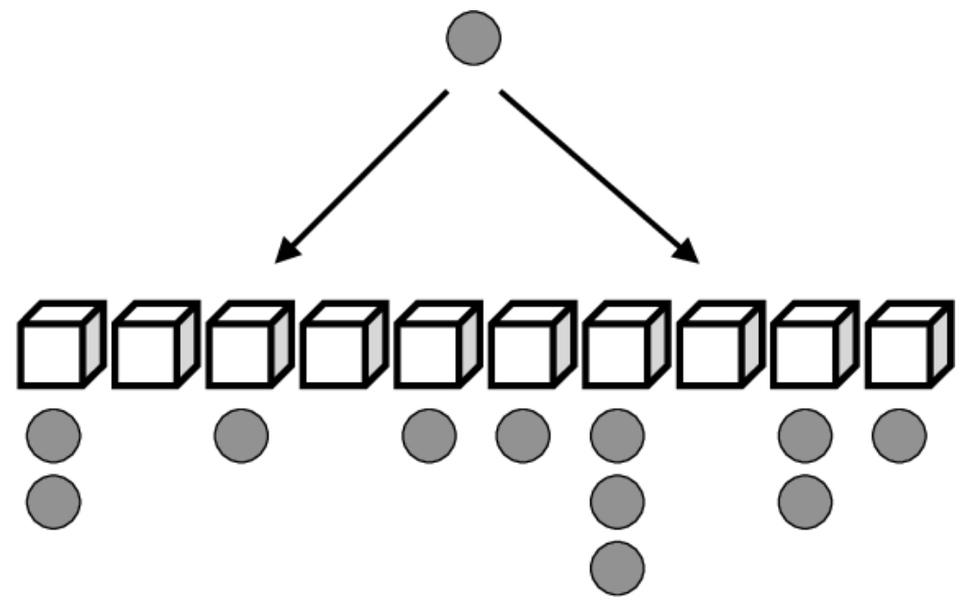
- Thus, Pr[bin b has at least $4 \ln n/(\ln \ln n)$ balls] $\leq 1/n^3$

Hashing: With High Probability Bounds

- **Lemma.** If n balls are thrown independently and uniformly into n bins, then with high probability, the fullest bin contains $O(\frac{\log n}{\log \log n})$
- **By union bound**, the probability that any bin has at least $k = 4\frac{\ln n}{\ln \ln n}$ balls is $\sum_{b=1}^{n} \Pr[\text{bin } b \text{ has at least } 4 \ln n/(\ln \ln n) \text{ balls}] \leq \sum_{i=1}^{n} \frac{1}{n^3} = \frac{1}{n^2}$
- Thus, **with high probability** the fullest bin contains $O(\frac{\log n}{\log \log n})$ balls
- **Note.** This bound is tight: fullest bins contains $\Theta(\frac{\log n}{\log \log n})$ balls w.h.p.

Power of Two Choices

- If we thrown n balls into n bins independently and uniformly at random then the max load is $\Theta\left(\frac{\log n}{\log \log n}\right)$
- **Fun fact.** For each balls, if we instead pick two bins independently and uniformly at random, and put the ball in the less loaded bin, then the max load is at most $\frac{\log \log n}{\log 2} + O(1)$
- Exponential improvement
- Can be generalized to any d choices



Hashing: Good Hash Functions

- Storing a truly random hash function. How many possible hash functions from a universe of size u to a universe of $[1,\ldots n]$
  - n^u
  - Takes $u \log n$ bits to represent/store such a hash function
- We want that probability of collision is at most $1/n$, can we achieve that without truly random hash function?
- **Universal classes of hash function.** The idea is to choose a hash function not from all possible random hash functions, but from a carefully selected class of hash functions
- A class of hash functions $\mathscr{H}$ is universal if for any pair of elements $x, y \in U$ the probability that a randomly chosen $h \in \mathscr{H}$ satisfies $h(x) = h(y)$ is at most $1/n$

Designing a Universal Hash Function

- Size of table: prime number p
- Identify each element $x \in U$ with a base-p integer of r digits:
  $x \equiv (x_1, \ldots, x_r)$
- Hash function. Let A = set of all r-digit, base-p integers. For each $a = (a_1, \ldots, a_r)$ where, define

$$h_a(x) = \left(\sum_{i=1}^{r} a_i x_i \right) \mod p$$

- Hash function family. $\mathscr{H} = \{h_a \mid a \in A\}$.
- **Claim.** $\mathscr{H}$ is universal.

The Dictionary Problem Revisited

- Goal: Queries, inserts and deletes to a set $\mathbf{S} \subseteq \mathcal{U}$
- Set $\mathbf{S}$ is **large** (does not fit in local RAM)

E.g. disk, network

Large, remote memory

$a \in \mathbf{S}?$

No

$c \in \mathbf{S}?$

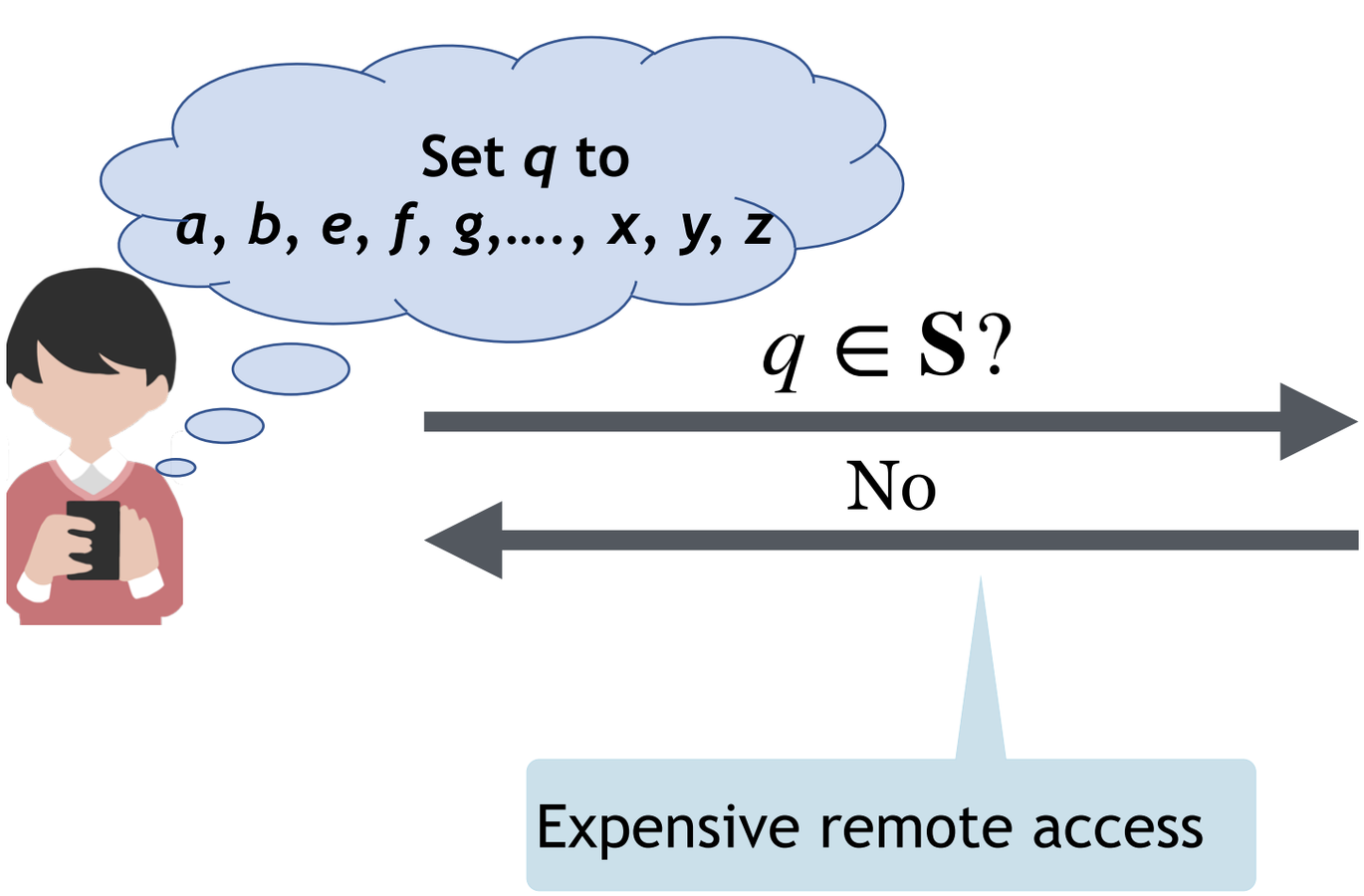
Yes

Expensive remote accesses

Dictionary for set $\mathbf{S} = \{c, d\}$

The Dictionary Problem

- Goal: Queries, inserts and deletes to a set $\mathbf{S} \subseteq \mathcal{U}$
- Set $\mathbf{S}$ is **large** (does not fit in local RAM)
- Most of the time, items are **not** in $\mathbf{S}$



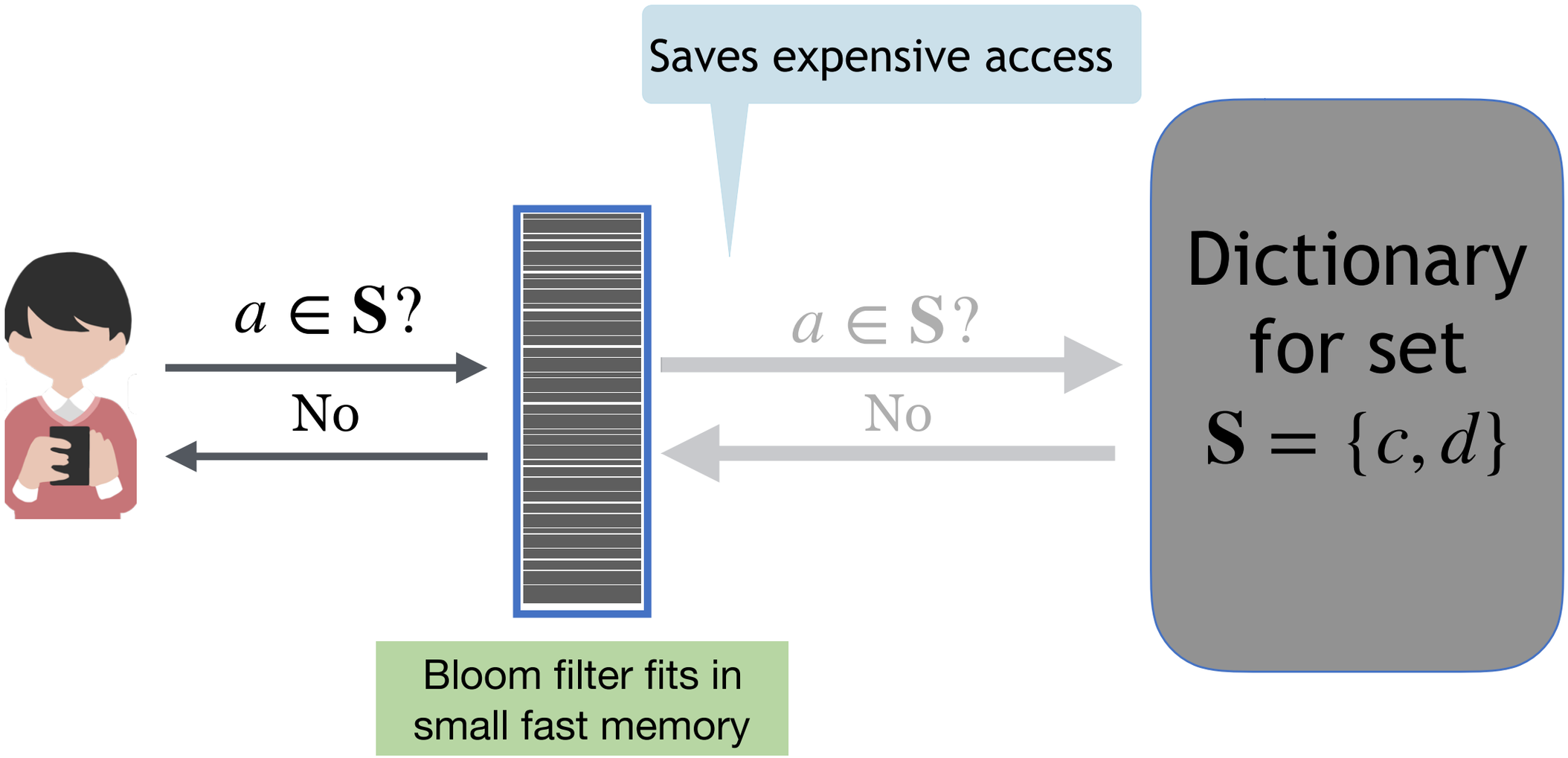
E.g. disk, network

Large, remote memory

Dictionary for set $\mathbf{S} = \{c, d\}$

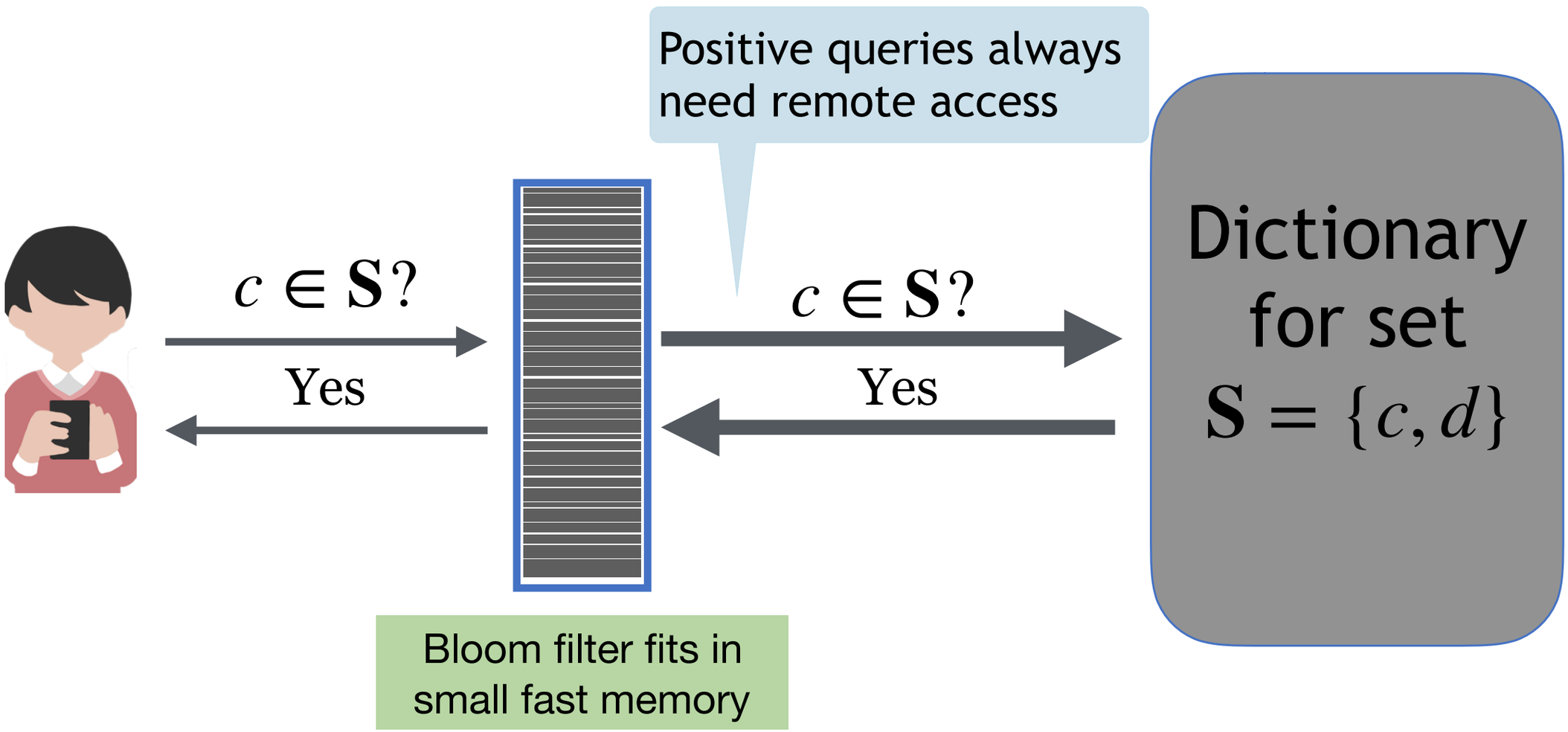
Bloom Filters Speed up Dictionaries

- Bloom filter [B70]: a lossy compressed dictionary
- **Advantage**: fit in small memory, filter out most negative queries



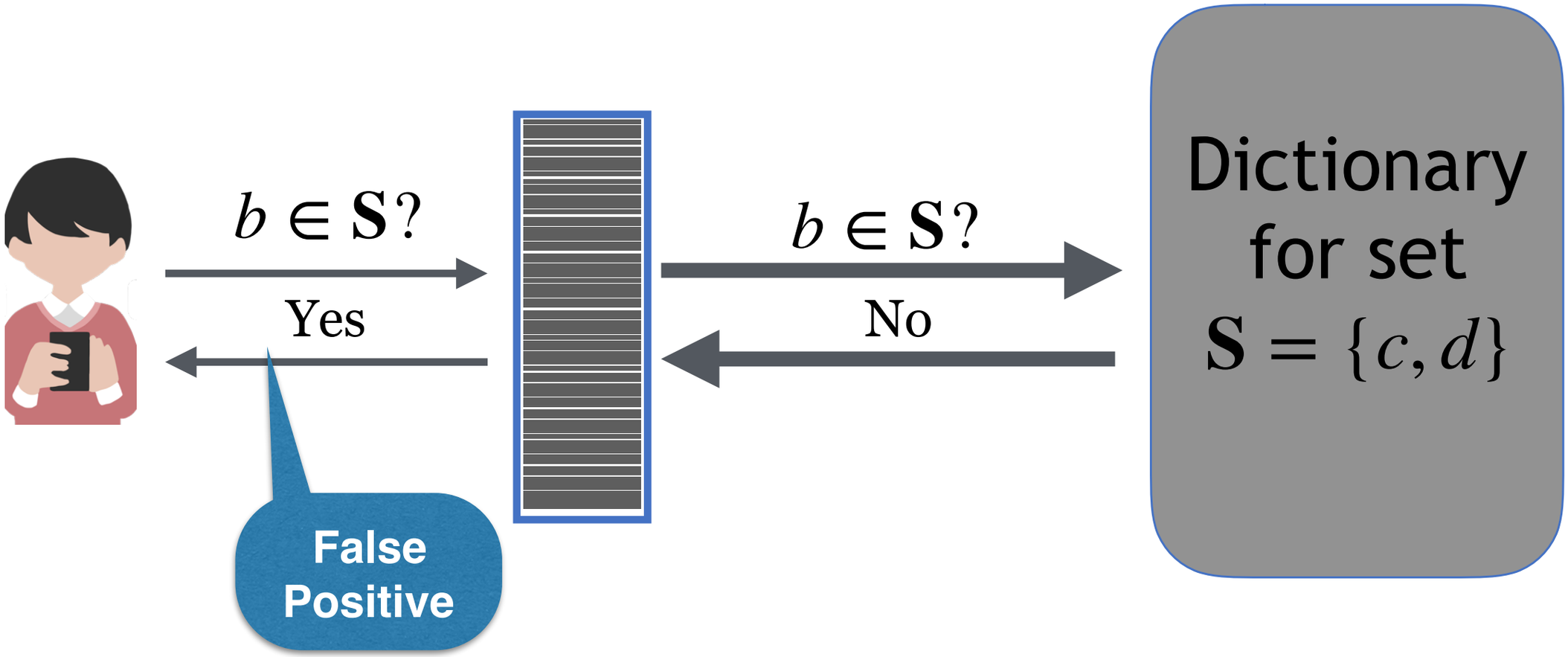
Bloom Filters Speed up Dictionaries

- Bloom filters are a lossy compressed dictionary
- Still need to consult remote dictionary on positive queries



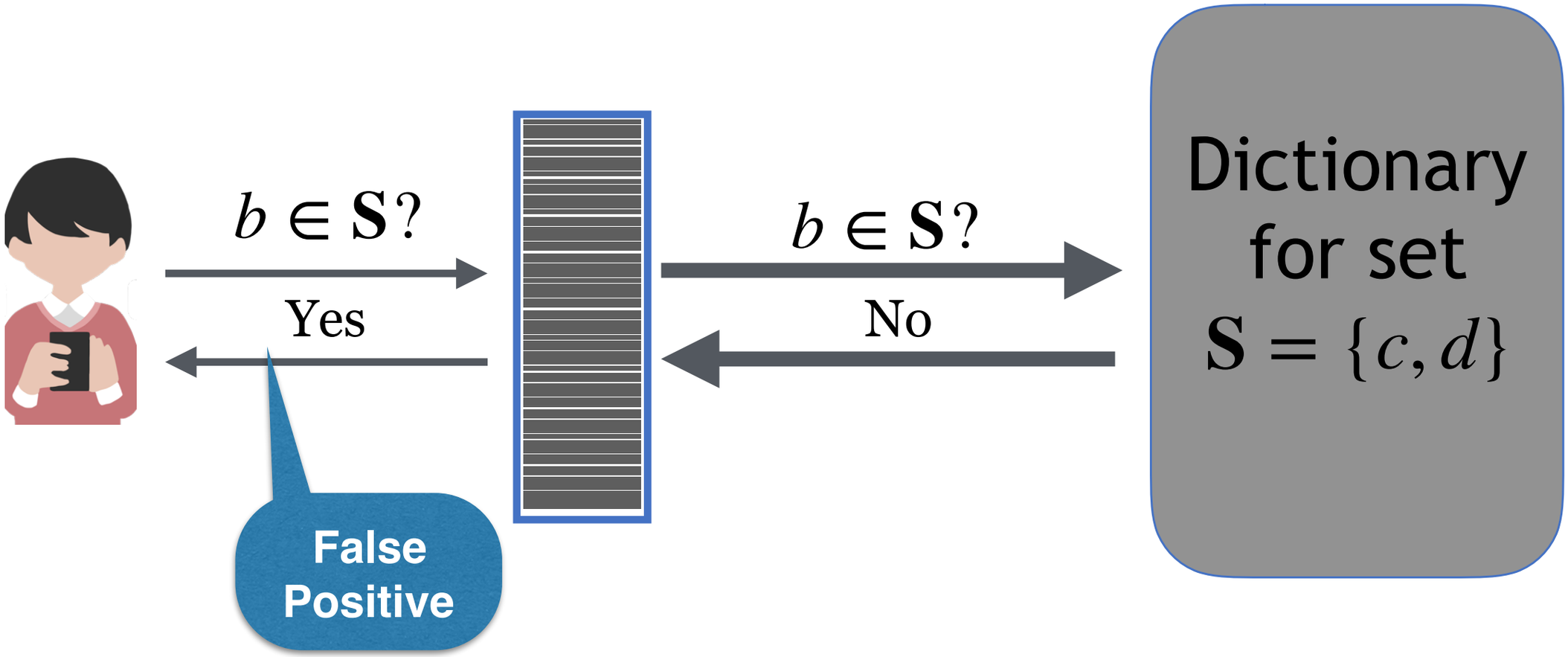
Bloom Filters Speed up Dictionaries

- Bloom filters trade-off space for accuracy: may have false positives



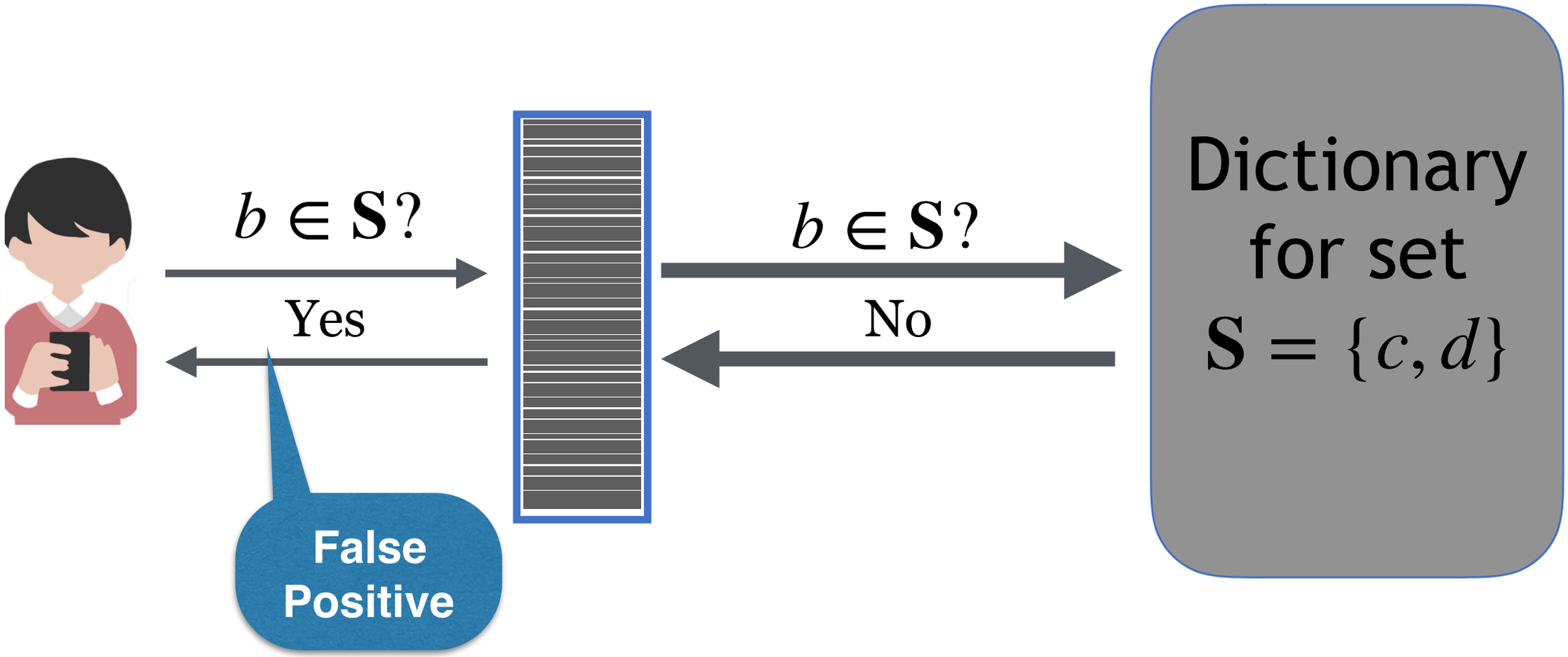
Bloom Filters Guarantees

- **No false negative:** If $q \in \mathbf{S}$, always return “present”
- **Few false positives**: If $q \notin \mathbf{S}$, returns “present” with prob $\leq \epsilon$



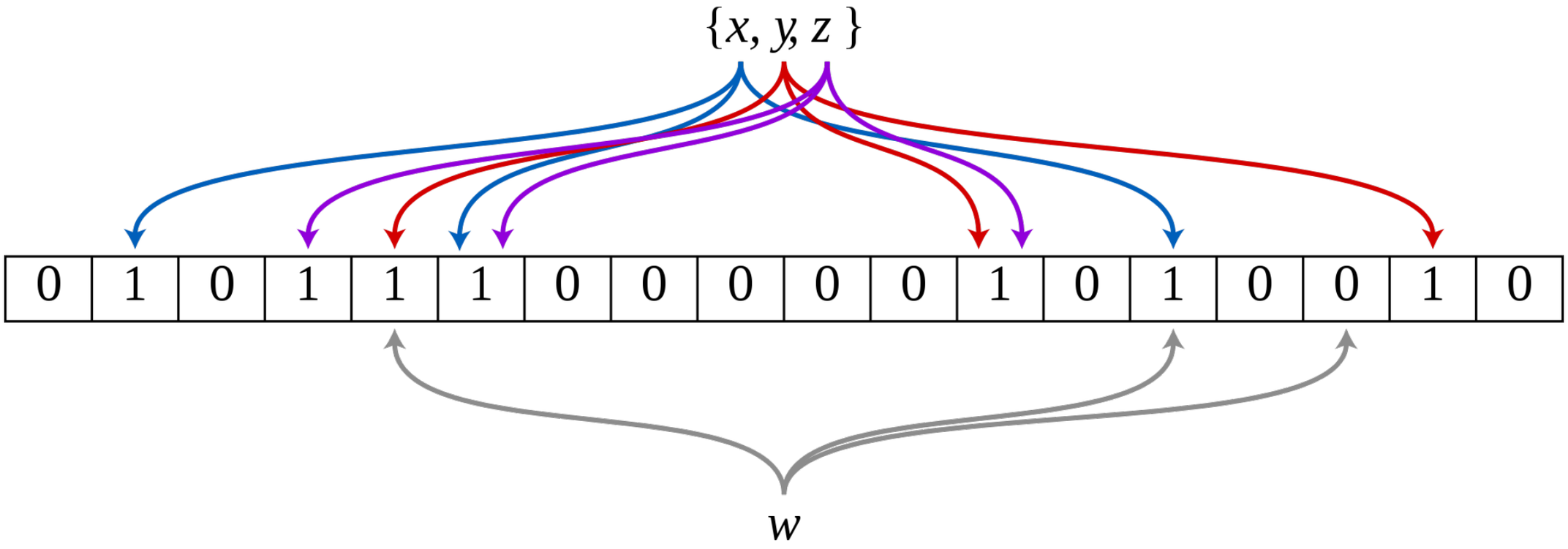
Bloom Filters Guarantees

- **No false negative:** If $q \in \mathbf{S}$, always return “present”
- **Few false positives**: If $q \notin \mathbf{S}$, returns “present” with prob $\leq \epsilon$
- **Space:** $O(n \log 1/\epsilon)$ bits where n is the size of $\mathbf{S}$



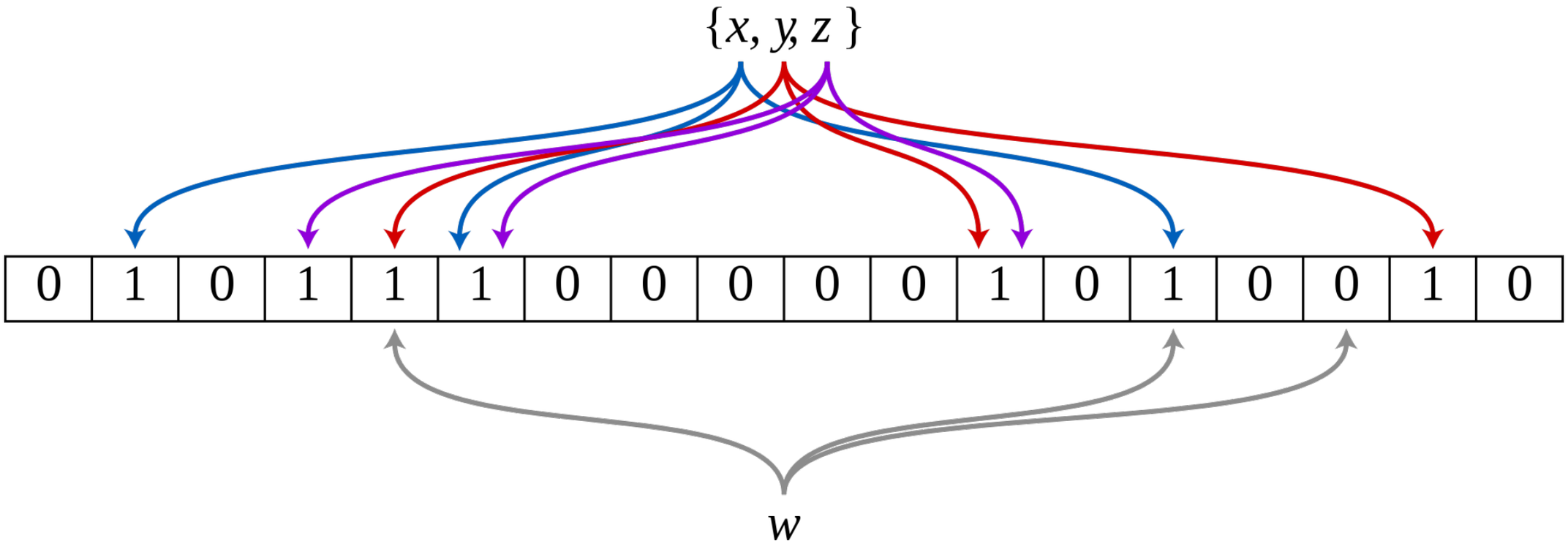
Bloom Filter: Details

- A Bloom filter consists of an array of m bits together with k hash functions $h_1, \ldots, , h_k : U \to [1,\ldots, m]$
- Assume h_i are mutually independent, ideal random functions
- Insert(x). Set bit $h_1(x), h_2(x), \ldots, h_k(x)$ to 1 (regardless of what the bit was set to before)



Bloom Filter: Details

- Can initialize S by inserting every $x \in S$
- Search(y). Check each bit: $h_1(y), \ldots h_k(y)$. If any of them is set to 0 then report no, that is, $y \notin S$, else report yes.
- No deletes in original Bloom filters; improved variants can handle it
- Observe: no false negatives, that is, every **no** answer is correct



Bloom Filter: Analysis

- Let us estimate the false positive rate given parameters n, m and k
- What is the probability that a given bit in the Bloom filter is set to 0?
- Same as if we throw kn balls in m bins and the probability that a particular bin is empty
- Each ball misses the bin with probability $1 - (1/m)$ and each throw is independent so probability a bin is empty is $p = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-kn/m}$
- This is just an approximation but simplifies arguments
- We get a false positive if all k bins are set to 1
    - Pr[false positive] $\approx (1 - e^{-kn/m})^k$

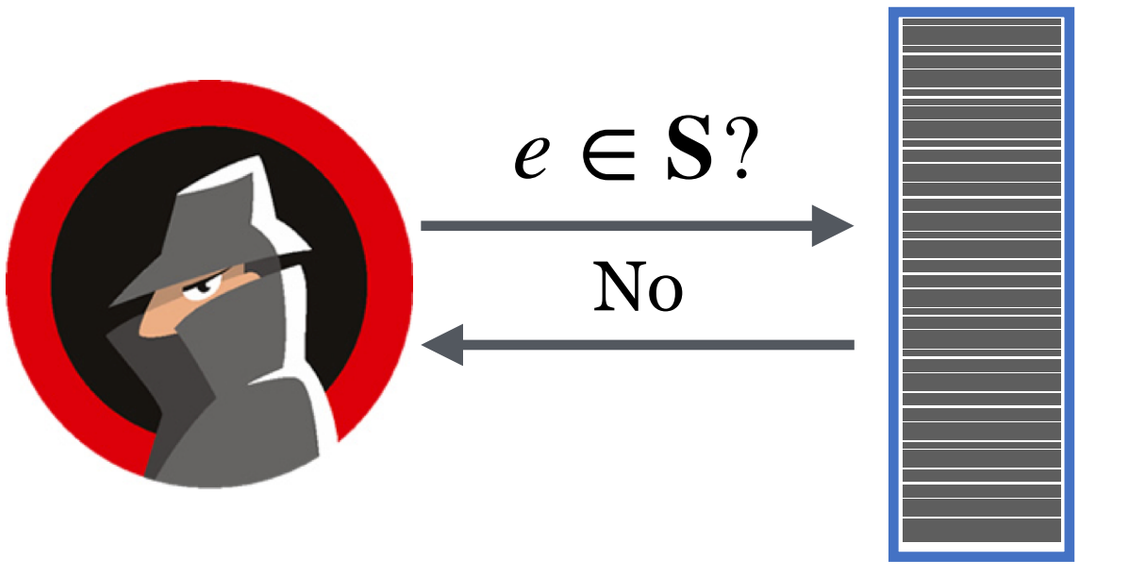
Bloom Filter: False Positive Probability

- Pr[false positive] $\approx (1 - e^{-kn/m})^k$
- **Notice.** False-positive probability increases with n (number of items) and decreases with m (the number of bits)
- Best value of k for a given n and m: Take logs of both side

$$k \ln(1-p) = (-m/n) \ln p \ln(1-p)$$

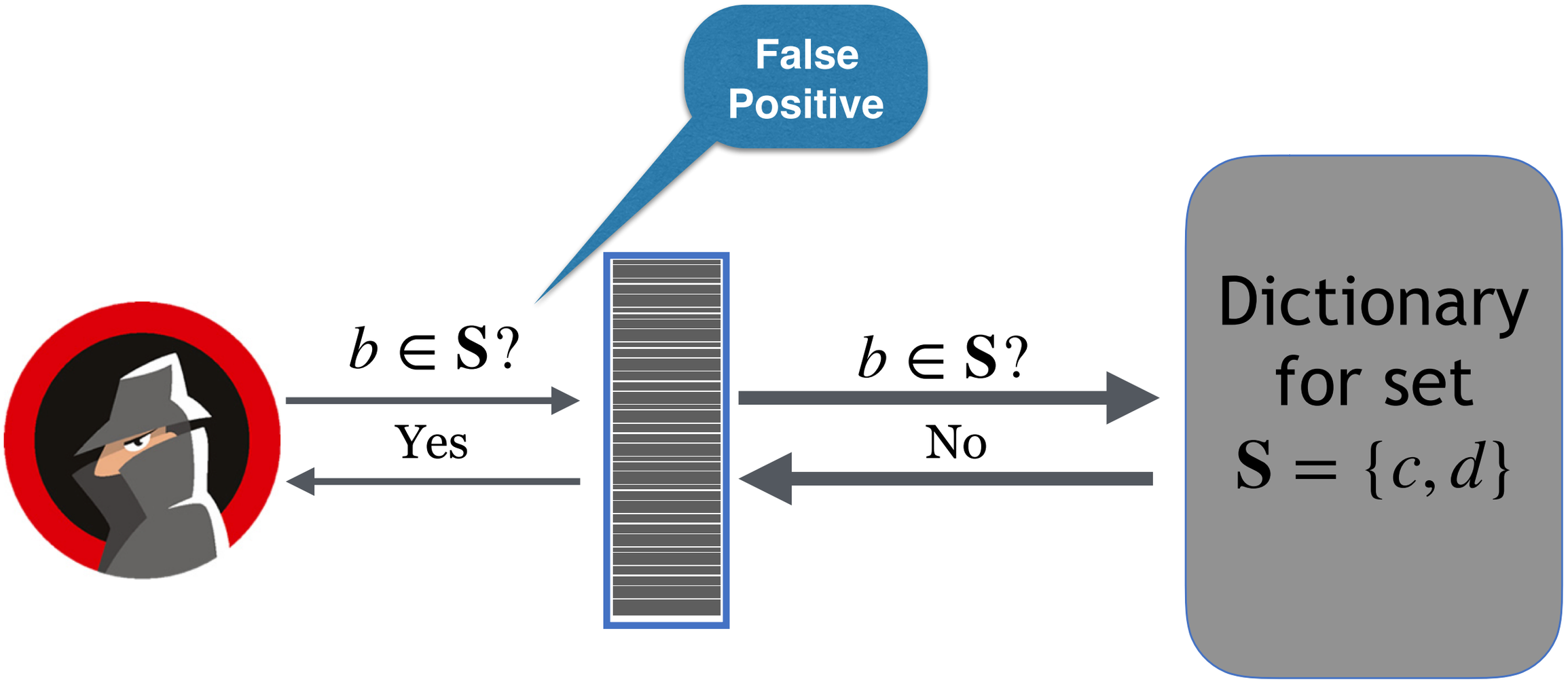
- By symmetry, this is minimized at $p = 1/2$, that is, $k = \ln 2 \cdot (m/n)$
- Pr[false positive] $\approx (1/2)^{\ln 2 (m/n)} = \epsilon$
- For $m = 8n$ (only 8 bits per element), $\epsilon \approx 0.02$!
- Thus, for a desired false positive probability ϵ, $m = O(n \log 1/\epsilon)$

Bloom Filters Don’t Fix Mistakes

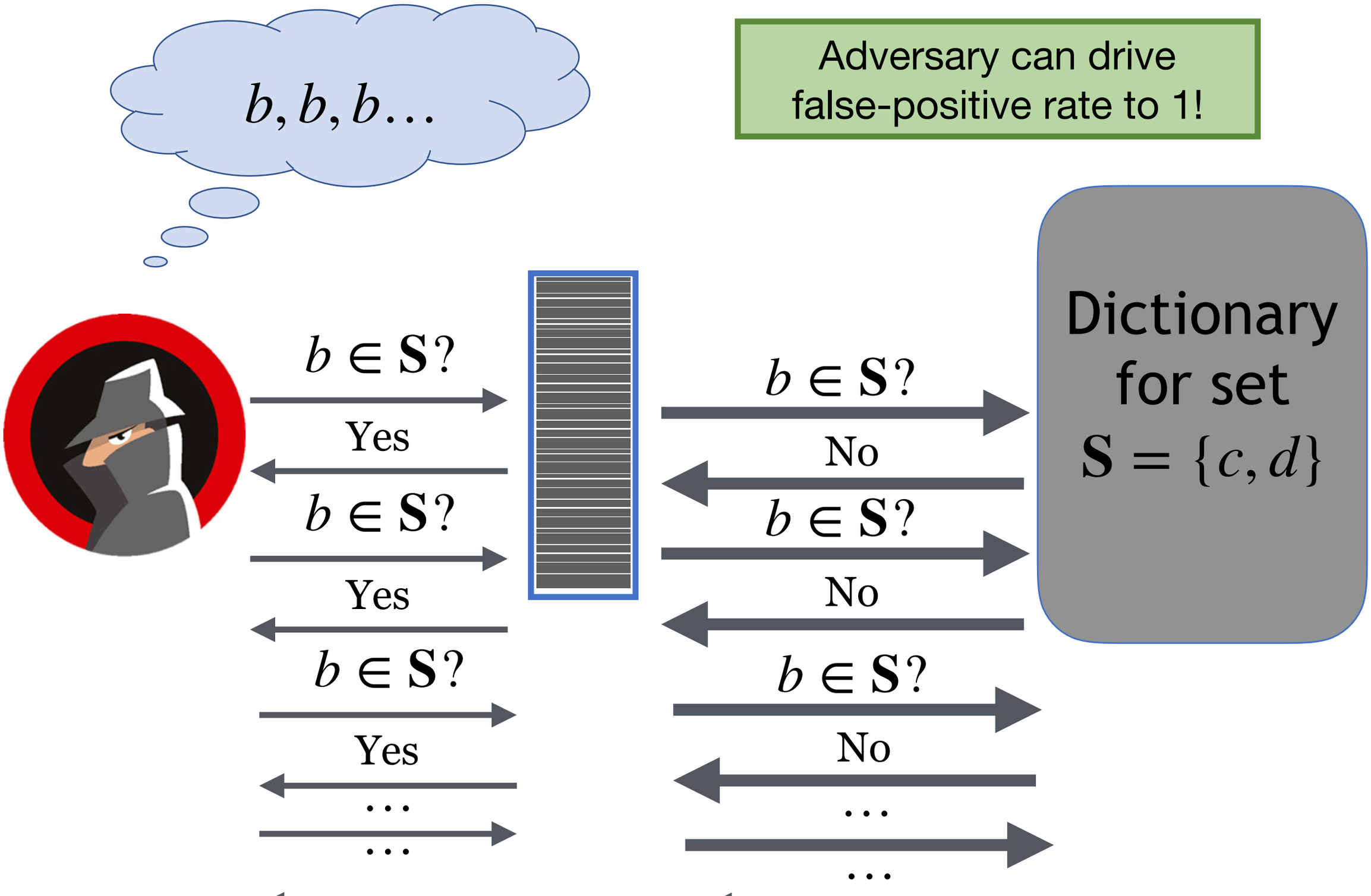


Dictionary for set $\mathbf{S} = \{c, d\}$

Bloom Filters Don’t Fix Mistakes



Bloom Filters Don’t Fix Mistakes



Problem with Bloom Filters

- Once a false positive, always a false positive

Adaptive Bloom Filters [2018]

2018 IEEE 59th Annual Symposium on Foundations of Computer Science

Bloom Filters, Adaptivity, and the Dictionary Problem

Michael A. Bender*, Martín Farach-Colton†, Mayank Goswami‡,
Rob Johnson§, Samuel McCauley¶, and Shikha Singh¶

* Stony Brook University, Stony Brook, NY 11794-2424 USA. Email: bender@cs.stonybrook.edu.
† Rutgers University, Piscataway, NJ 08856 USA. Email: martin@farach-colton.com.
‡ Queens College, CUNY, NY 11367 USA. Email: mayank.goswami@qc.cuny.edu.
§ VMware Research, Creekside F, 3425 Hillview Ave, Palo Alto, CA 94304 USA. Email: robj@vmware.com.
¶ Wellesley College, Wellesley, MA 02481 USA. Email: {smccaule, shikha.singh}@wellesley.edu.

***Abstract*—An approximate membership query data structure (AMQ)—such as a Bloom, quotient, or cuckoo filter—maintains a compact, probabilistic representation of a set $\mathcal{S}$ of keys from a universe $\mathcal{U}$. It supports lookups and inserts. Some AMQs also support deletes. A query for $x \in \mathcal{S}$ returns PRESENT. A query for $x \notin \mathcal{S}$ returns PRESENT with a tunable *false-positive probability* ε, and otherwise returns ABSENT.**

AMQs are widely used to speed up dictionaries that are stored remotely (e.g., on disk or across a network). The AMQ is stored locally (e.g., in memory). The remote dictionary is only accessed when the AMQ returns PRESENT. Thus, the primary performance metric of an AMQ is how often it returns ABSENT for negative queries.

Existing AMQs offer weak guarantees on the number of false positives in a sequence of queries. The false-positive probability ε holds only for a single query. It is easy for an adversary to drive an AMQ's false-positive rate towards 1 by simply repeating false positives.

This paper shows what it takes to get strong guarantees on the number of false positives. We say that an AMQ is *adaptive* if it guarantees a false-positive probability of ε for every query, *regardless of answers to previous queries*.

also support deletes. A positive query for $x \in \mathcal{S}$ returns PRESENT. A negative query for $x \notin \mathcal{S}$ returns PRESENT with a tunable ***false-positive probability*** ε, and otherwise returns ABSENT.

AMQs are used because they are small. An optimal AMQ can encode a set $\mathcal{S} \subseteq \mathcal{U}$, where $|\mathcal{S}| = n$ and $|\mathcal{U}| = u$, with a false-positive probability ε using $\Theta(n \log(1/\varepsilon))$ bits [6]. In contrast, an error-free representation of $\mathcal{S}$ takes $\Omega(n \log u)$ bits.

One of the main uses of AMQs is to speed up dictionaries [5, 8, 10, 11, 13, 20, 21]. Often, there is not enough local storage (e.g., RAM) to store the dictionary's internal state, $\mathbf{D}$. Thus, $\mathbf{D}$ must be maintained remotely (e.g., on-dis
to $\mathbf{D}$ are expensive. E
set $\mathcal{S}$ of keys occurr
accessing $\mathbf{D}$ on most
that a key is not in $\mathcal{S}$

Thus, the primary

Broom Filters: Bloom filters that clean up their mistakes

Acknowledgments

- Some of the material in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
  - Lecture slides: https://web.stanford.edu/class/archive/cs/cs161/cs161.1138/