Overview

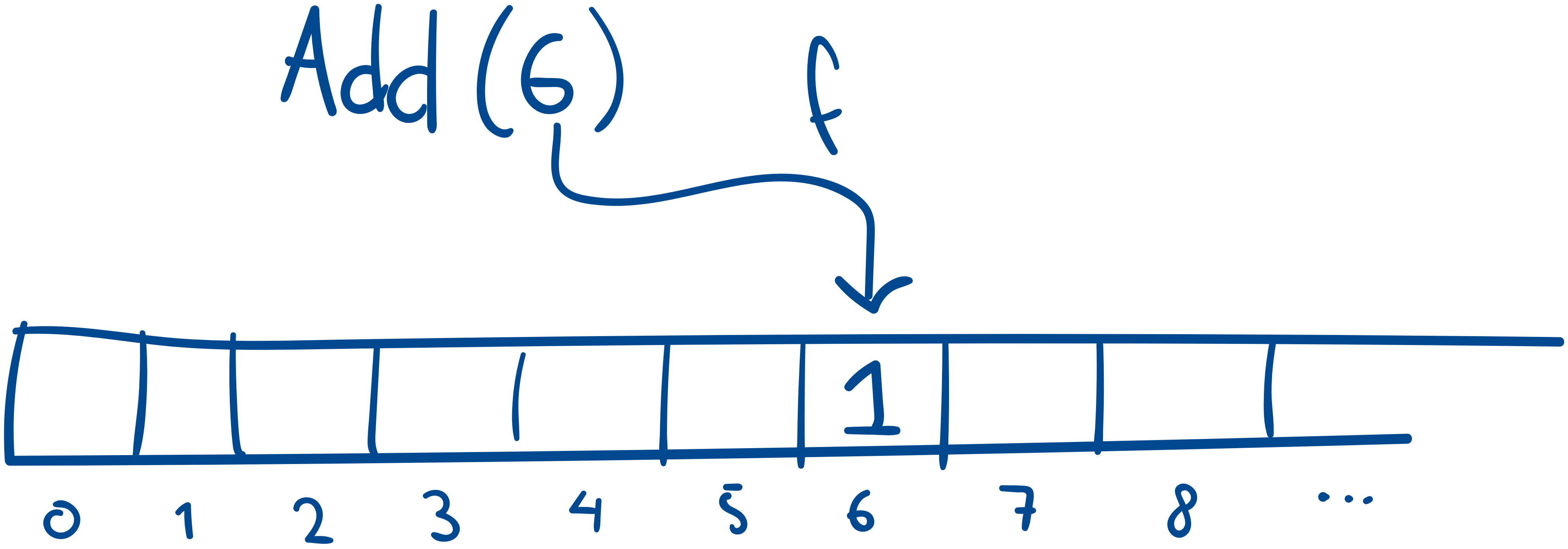
- Set membership data structures
- Why are false positives acceptable
- A Bloom filter in a few steps
- Bloom filter tricks
- GloBiMaps

SetMembership

	LIST	TREE	HASH MAP
$Add(x : U)$	$O(1)$	$O(\log n)$	$O(1)$ $O(n)$
$Remove(x : U)$	$O(n)$	$O(\log n)$	$O(1)$ $O(n)$
$Test(x : U) : bool$ "Search"	$O(n)$	$O(\log n)$	$O(1)$ $O(n)$
Needs:	$=$	$\leq$	hash $=$

Bit Set

- Bijection between elements and array of bits



SetMembership

	LIST	TREE	HASH MAP	BIT SET
$Add(x : U)$	$O(1)$	$O(\log n)$	$O(1)$ $O(n)$	$O(1)$
~~$Remove(x : U)$~~	~~$O(n)$~~	~~$O(\log n)$~~	~~$O(1)$ $O(n)$~~	~~$O(1)$~~
$Test(x : U) : bool$	$O(n)$	$O(\log n)$	$O(1)$ $O(n)$	$O(1)$
Needs:	$=$	$\leq$	hash	index
Space	$O(n)$	$O(n)$	$O(n)$	$\lvert U \rvert$ bits

allow false positives

Also store the elements

Space/Time Trade-offs in Hash Coding with Allowable Errors

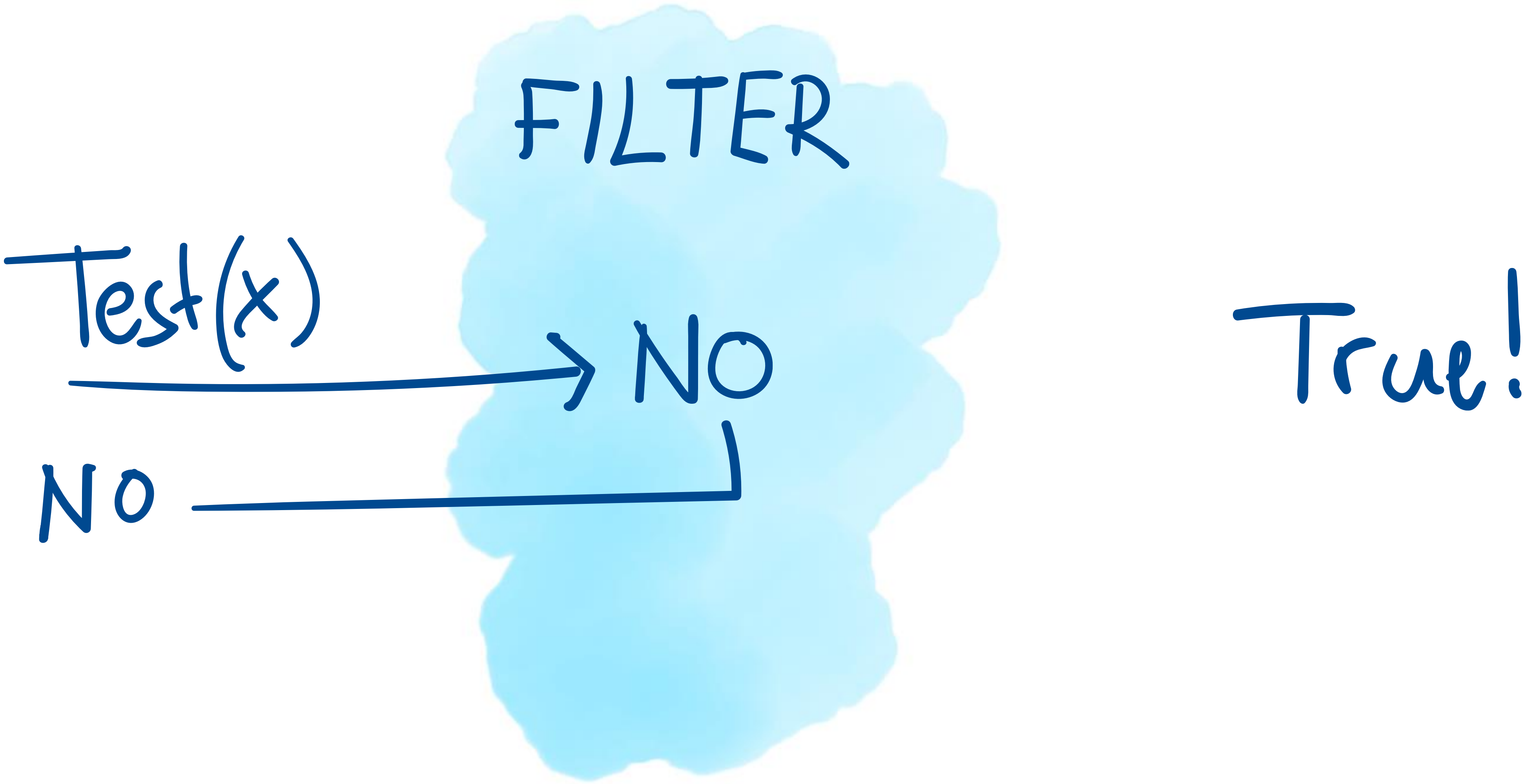
Burton H. Bloom
Computer Usage Company, Newton Upper Falls, Mass.

Communications of the ACM • July 1970

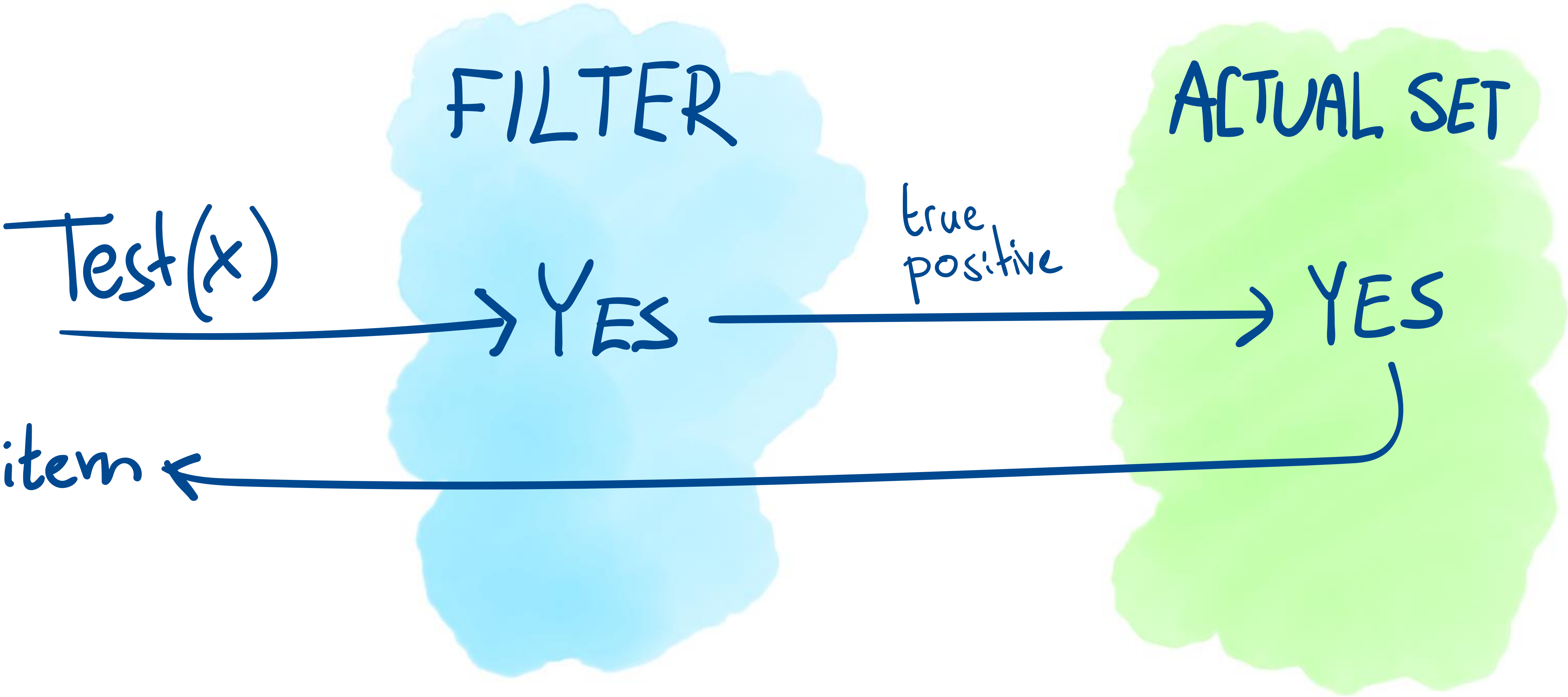
In this paper trade-offs among certain computational factors in hash coding are analyzed. The paradigm problem considered is that of testing a series of messages one-by-one for membership in a given set of messages. Two new hash-coding methods are examined and compared with a particular conventional hash-coding method. The computational factors considered are the size of the hash area (space), the time required to identify a message as a nonmember of the given set (reject time), and an allowable error frequency.

In such applications, it is envisaged that overall performance could be improved by using a smaller core resident hash area in conjunction with the new methods and, when necessary, by using some secondary and perhaps time-consuming test to "catch" the small fraction of errors associated with the new methods. An example is discussed which illustrates possible areas of application for the new methods.

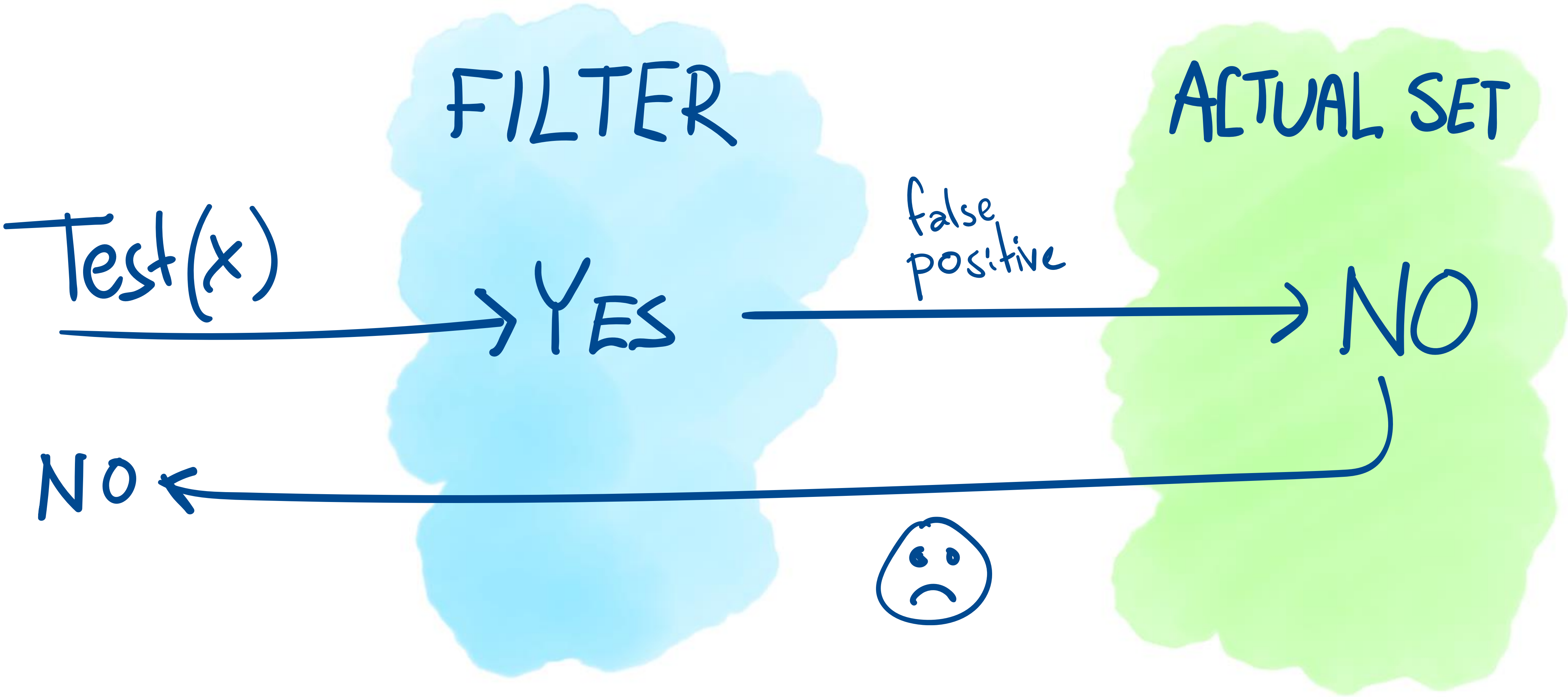
Allow false positives



Allow false positives



Allow false positives



Is this a good idea?

- Depends on “Filter” versus “Actual Set” cost
- Depends on hit rates
- If filter can be amazingly small
  - Maybe you don’t actually have the set!
- Fast rejection most of the time

	True	False
Negative	Filter	Filter
Positive	Filter + Set Hit	Filter + Set Miss

Applications 1/3

- **Bloom ‘70**: hyphenation
  - Most words covered by a few rules
  - Make a set containing the exceptions
  - Hypenation algo: check set, else use rules
  - Let’s add a filter! False positives?
    - Unnecessary lookup; still correct

Applications 2/3

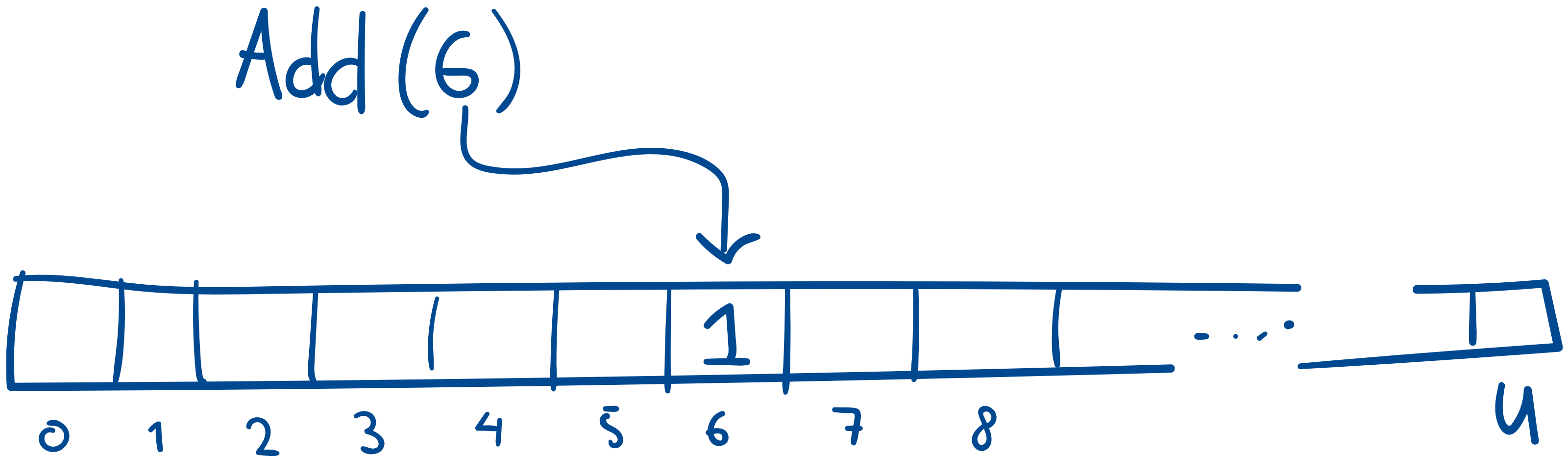
- **MrIlroy ’82**: early UNIX spell-checkers
  - Store correct words. False positives?
    - Just accept them 
  - Amazingly small filter!
- **Spafford ‘92**: unsuitable passwords
  - Store the set. False positives?
    - Not really harmful

Applications 3/3

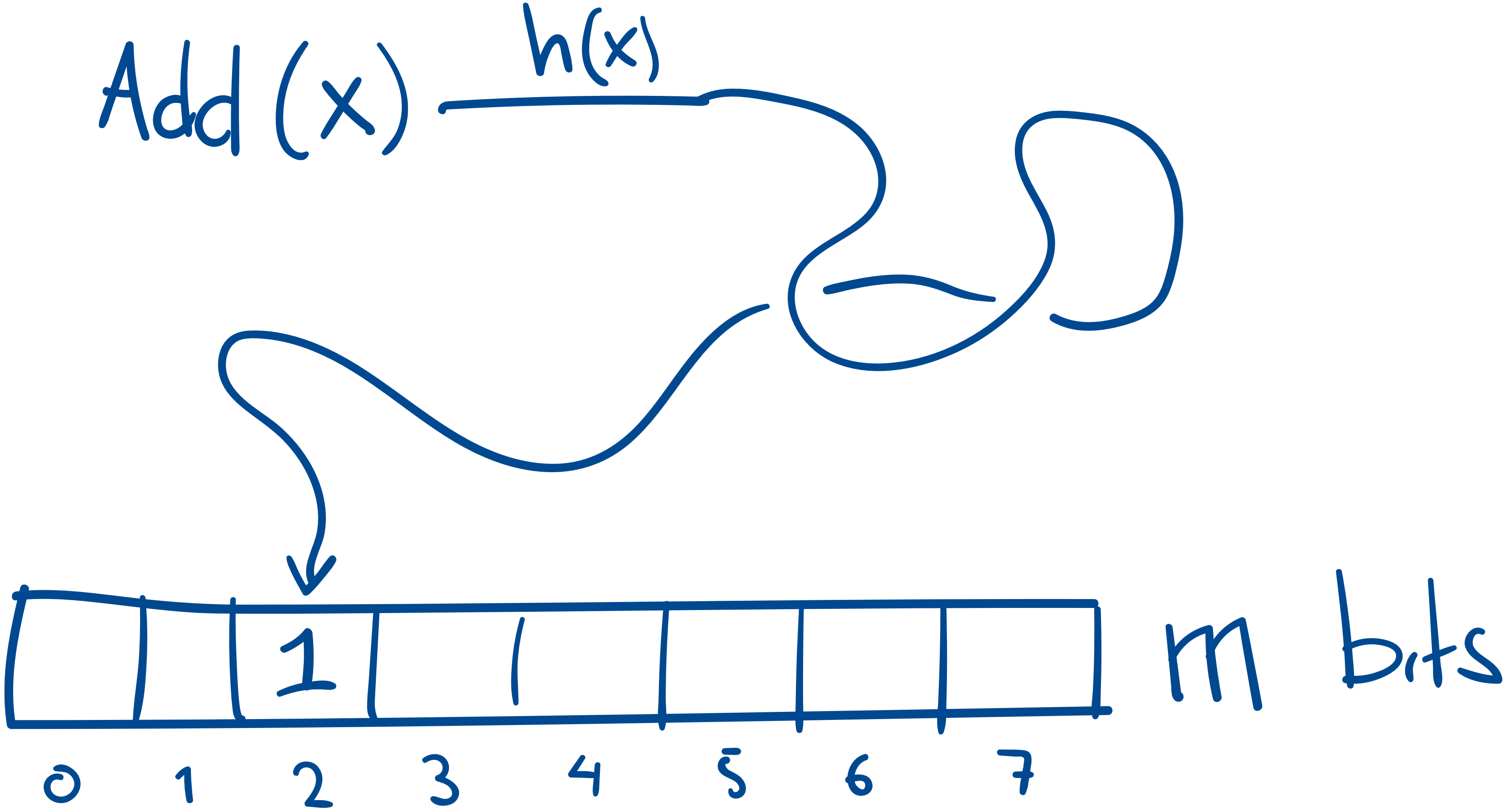
- **Chrome**: local filter for malicious URLs
  - Mostly misses
  - Google doesn’t see where you try to go
  - You don’t get the list
  - False positives?
    - Unnecessary warning; or ask Google.

Bit Set

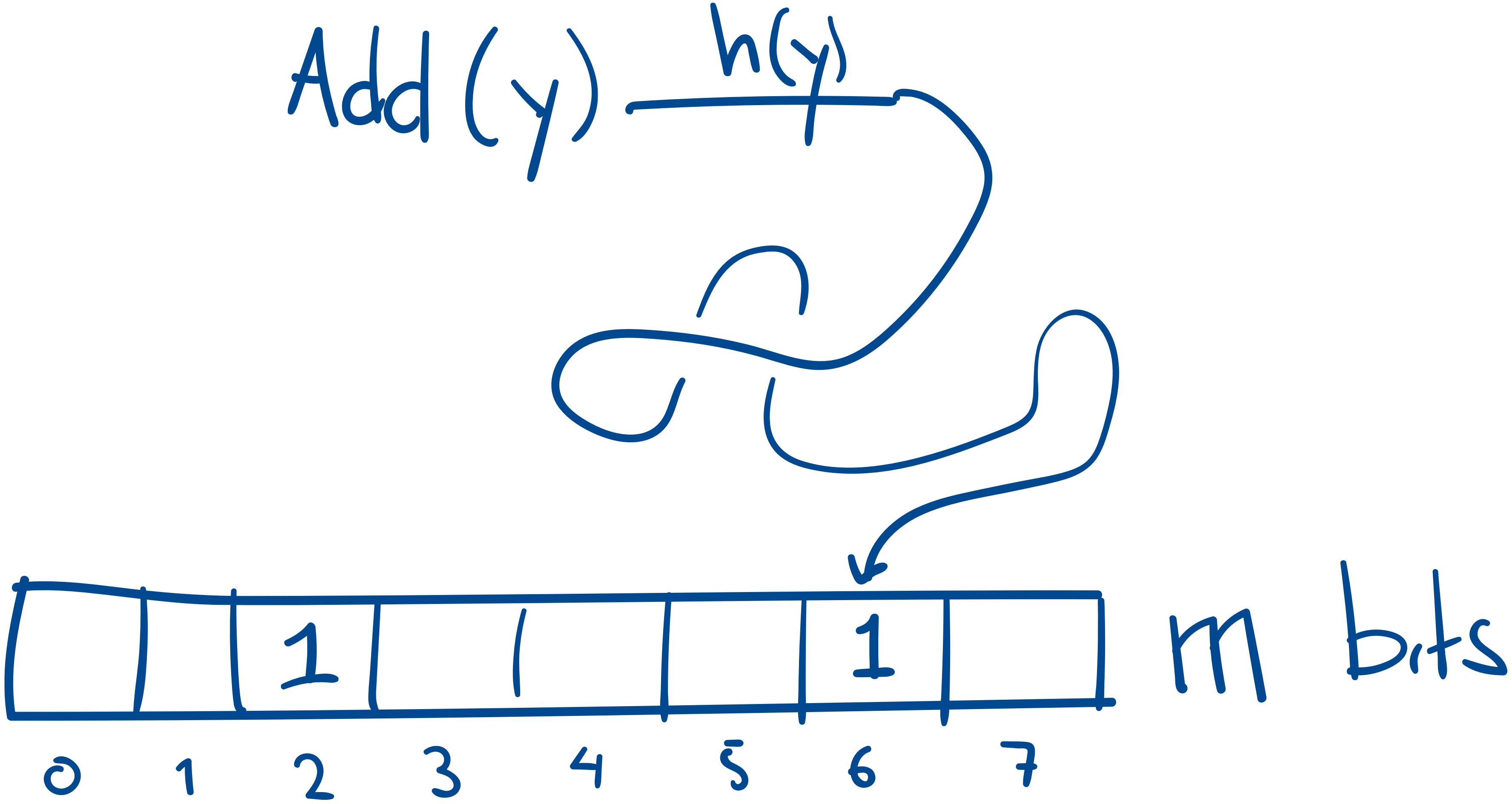
- Good constant factors
- Too large when universe U is large
  - Especially annoying if n << |U|



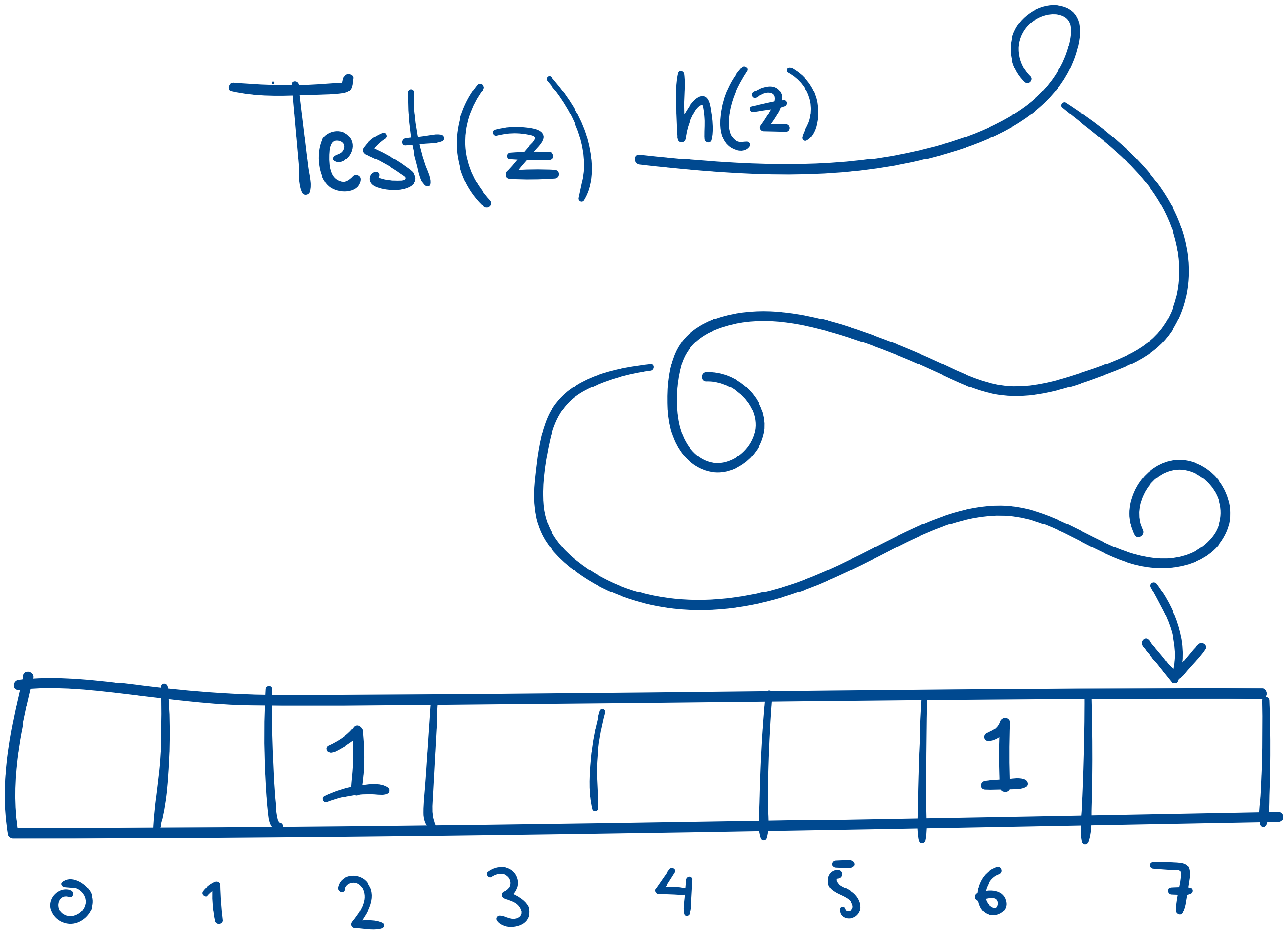
$h : U \to \mathbb{Z}_m$



$h : U \to \mathbb{Z}_m$



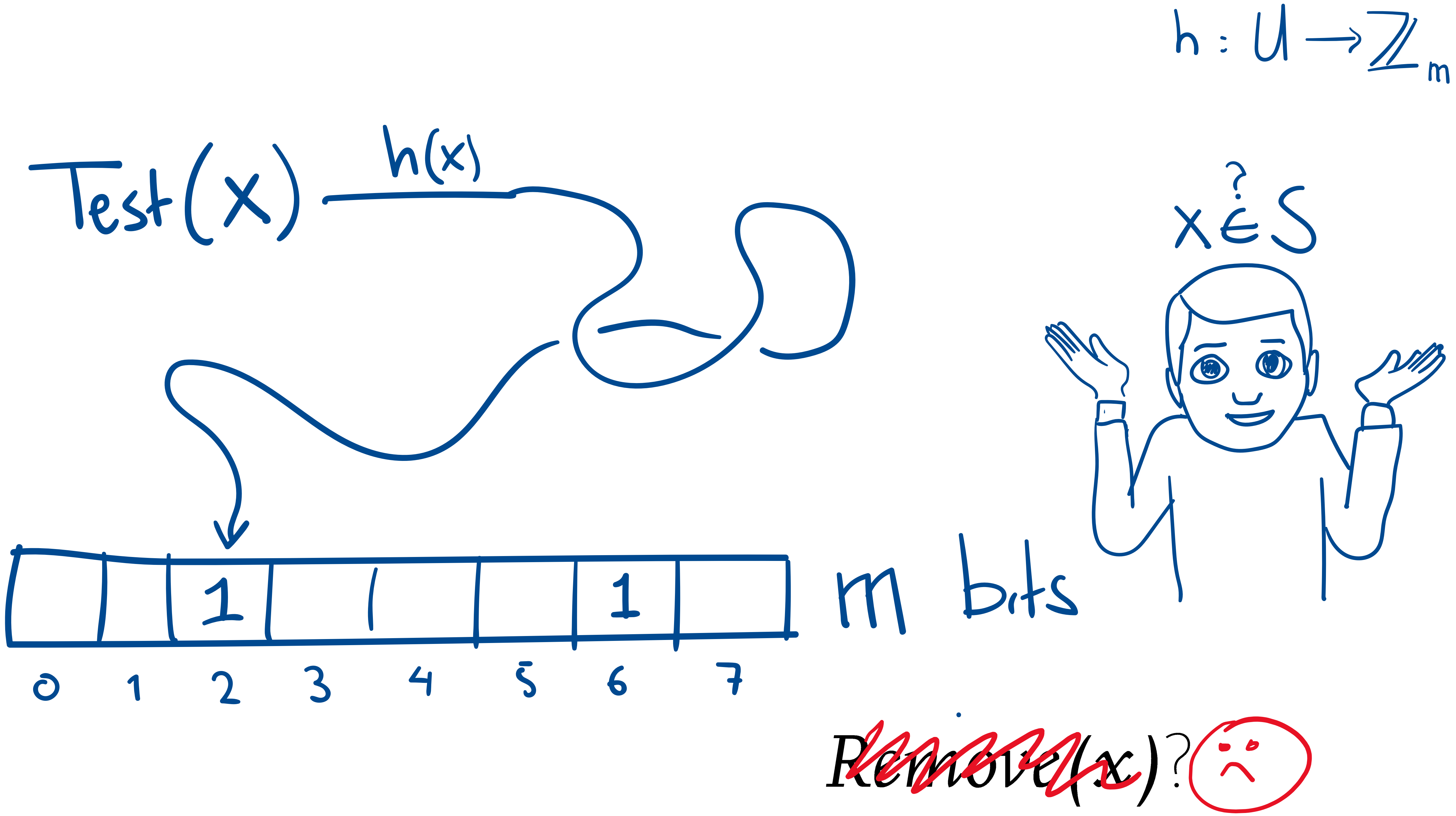
$h : U \to \mathbb{Z}_m$



$z \overset{?}{\in} S$

NO!

m bits


$h : U \to \mathbb{Z}_m$
Test(x)
h(x)
1
1
0 1 2 3 4 5 6 7
m bits
$x \overset{?}{\in} S$
~~Remove(x)~~?

The function h

- Take a “hash function”

- The function is deterministic
- For the analysis, we will assume it gives independent uniformly random indices

“Probabilistic data structure”

– Hashmaps: deterministic correctness
expected runtime

?

– Bloom filters: expected correctness
deterministic runtime

False positive probability

P[bit i is **0** after the first insertion]

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

$$\lim_{m \to \infty} \left(1 - \frac{1}{m}\right)^m = \frac{1}{e}$$

P[bit i is still **0** after the first n insertions]

$$\left(1 - \frac{1}{m}\right)^n = \left[\left(1 - \frac{1}{m}\right)^m\right]^{n/m} \approx e^{-\frac{n}{m}}$$

False positive probability

P[bit i is **1** after n insertions]

$$\approx 1 - e^{-\frac{n}{m}}$$

- Consider $Test(x)$ for nonmember x
- Bit $h(x)$ is set with probability…

Parameters

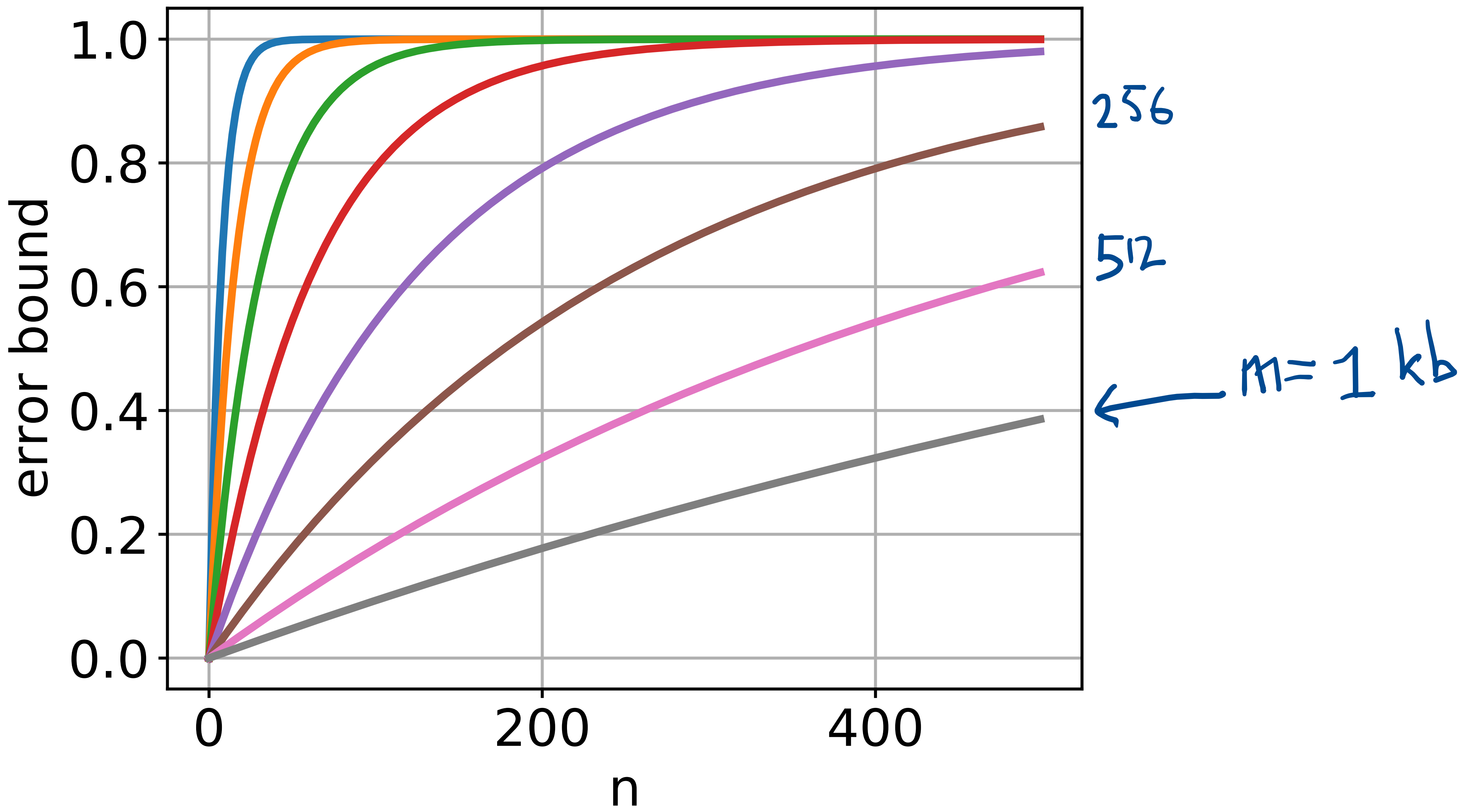
Probably fixed

n Number of items

- Maybe fixed
- Costs space

m Number of bits

ε Error bound

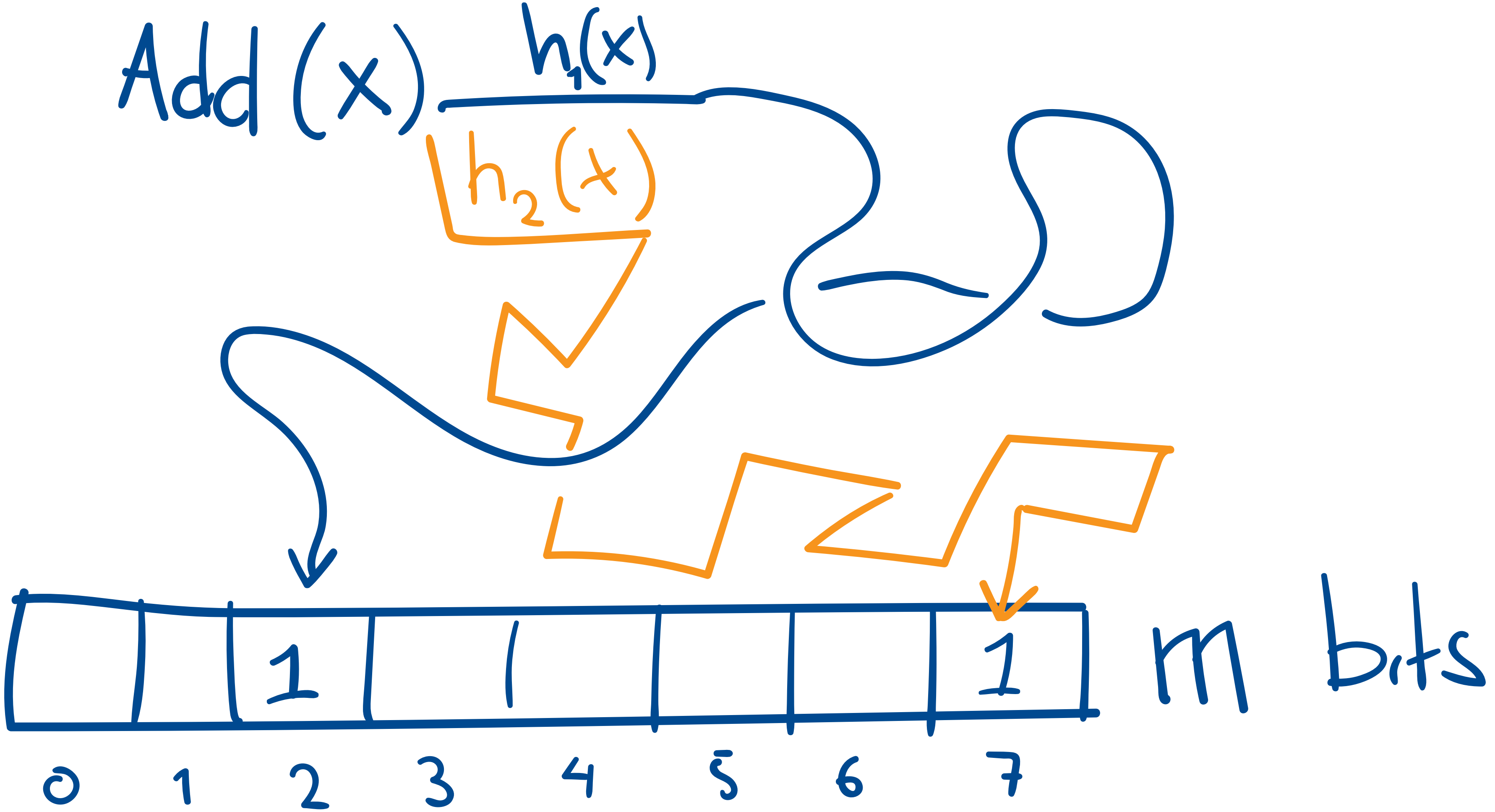


error bound
1.0
0.8
0.6
0.4
0.2
0.0
0
200
400
n
256
512
m = 1 kb

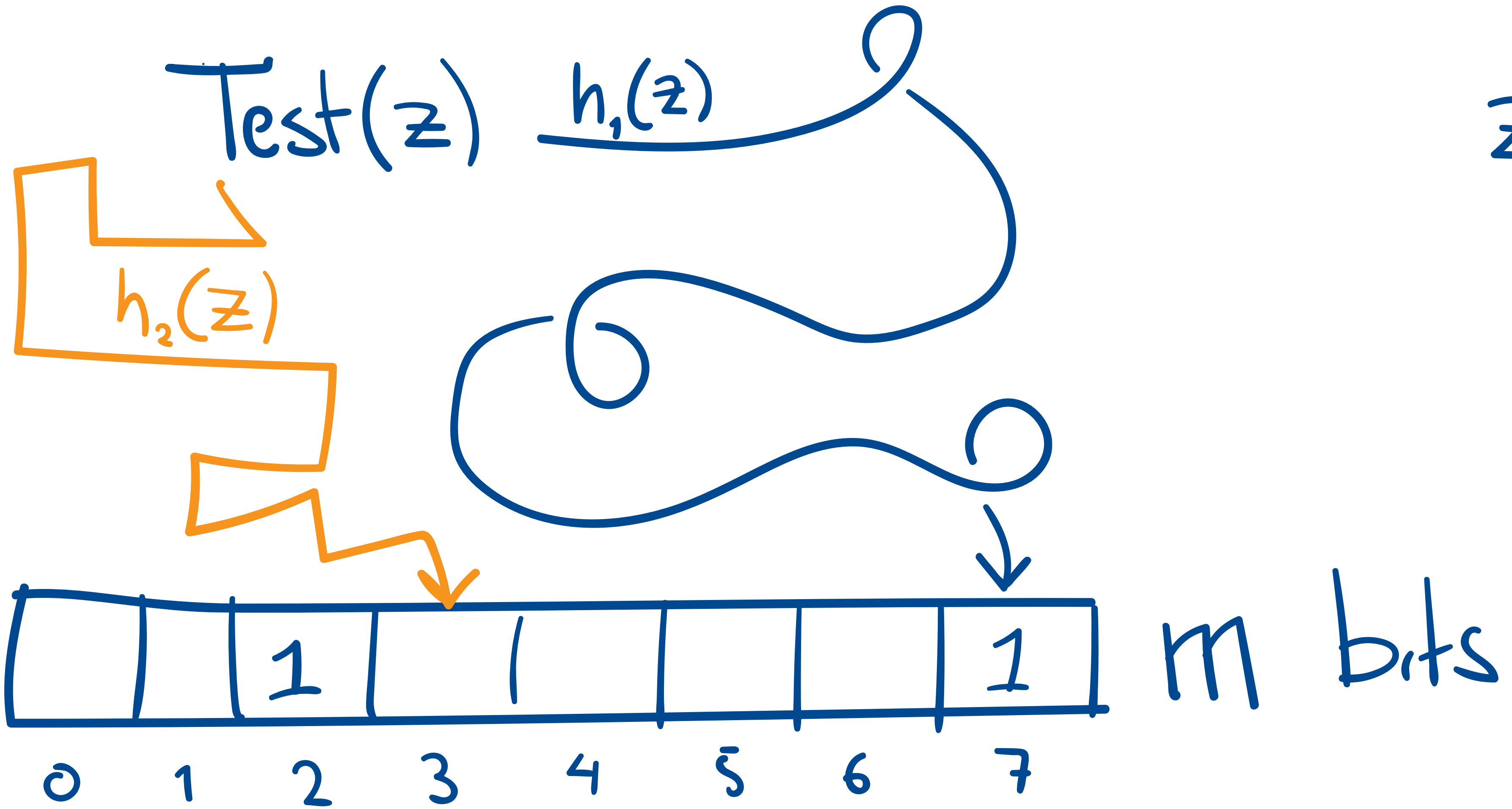
Dictionary example

	n	m	ε
English dictionary	500.000	100 kB	≈ 46%
(≈3 MB ASCII)		1 MB	≈ 5%
		3 MB	≈ 1.9%

$h_1, h_2 : U \to \mathbb{Z}_m$



$h_1, h_2 : U \to \mathbb{Z}_m$



$z \overset{?}{\in} S$

NO!

Bloom filter

- Fix **k** hash functions h_i
- Storage: array of **m** bits, all start unset

$Add(x)$: set all bits $h_i(x)$
$Test(x)$: are all bits $h_i(x)$ set?

What is the effect of k?

- Increases runtime
- It does not affect the space!

- Error probability?
  - Check more bits: accidents *less* likely
  - Set more bits: accidents are *more* likely

False positive probability

- A particular bit is **0** after the first insertion:

$$\left(1-\frac{1}{m}\right)^{k} = \left[\left(1-\frac{1}{m}\right)^{m}\right]^{k/m} \approx e^{-\frac{k}{m}}$$

- A particular bit is still **0** after *n* insertions:

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

False positive probability

- A particular bit is **1** after n insertions:

$$\approx 1 - e^{-\frac{kn}{m}}$$

- False positive $Test(x)$

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

(handwave!)

Parameters

Probably fixed

n Number of items

• Maybe fixed
• Costs space

m Number of bits

k Number of hash functions ← • Costs time

ε Error bound

Picking k

$n, m, k \in \mathbb{N}$

– Idea: given $\boldsymbol{n}$ and $\boldsymbol{m}$, pick $\boldsymbol{k}$ to minimize ε.

$$\varepsilon \approx \left(1 - e^{-\frac{kn}{m}}\right)^k = \exp\left(k \ln\left(1 - e^{-\frac{kn}{m}}\right)\right)$$

$\ln$

$$g \overset{\text{def}}{=} k \ln(1 - p)$$

$$\frac{dg}{dk} \overset{!}{=} 0$$

Picking k

$n, m, k \in \mathbb{N}$

– Idea: given $\boldsymbol{n}$ and $\boldsymbol{m}$, pick $\boldsymbol{k}$ to minimize ε.

$$\varepsilon \approx \left(1 - e^{-\frac{kn}{m}}\right)^k = \exp\left(k \ln\left(1 - e^{-\frac{kn}{m}}\right)\right)$$

$\ln$

$$g \overset{\text{def}}{=} k \ln(1-p) = -\frac{m}{n} \ln(p) \ln(1-p)$$

$$= m \frac{\cancel{n}}{\cancel{n}} \cdot \cancel{\ln(p)} \left(\frac{m k \cancel{n}}{\cancel{n}} \right) \ln(1-p)$$

Picking k

$n, m, k \in \mathbb{N}$

– Idea: given $\boldsymbol{n}$ and $\boldsymbol{m}$, pick $\boldsymbol{k}$ to minimize ε.

$$\varepsilon \approx \left(1 - e^{-\frac{kn}{m}}\right)^k = \exp\left(k \ln\left(1 - e^{-\frac{kn}{m}}\right)\right)$$

$$\xleftarrow{\ln} \quad g \overset{\text{def}}{=} k \ln(1-p) = -\frac{m}{n} \underbrace{\ln(p)\ln(1-p)}_{\min \Rightarrow p = 1/2}$$

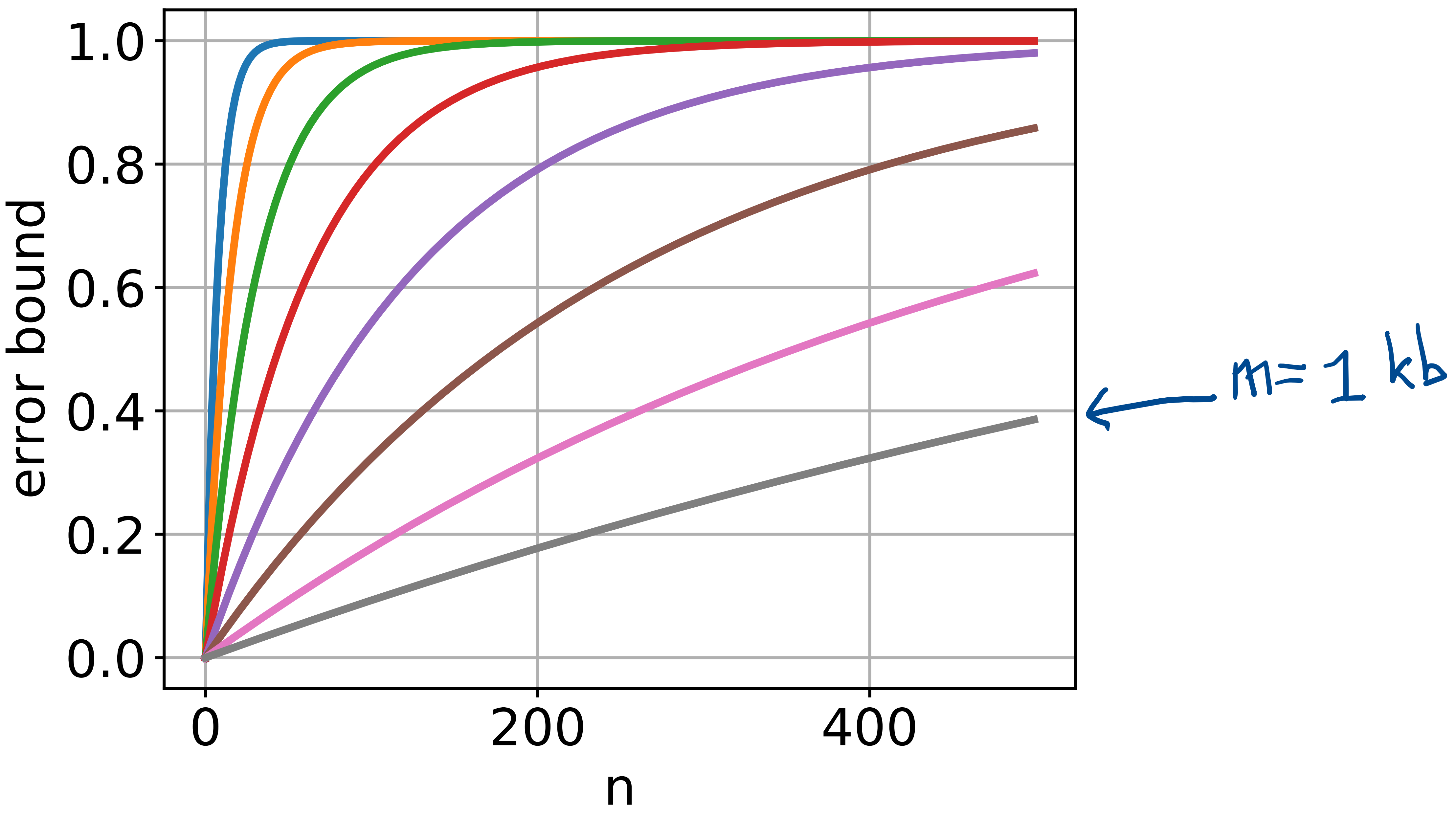
$$e^{-\frac{kn}{m}} = \frac{1}{2} \Rightarrow k = \ln 2 \cdot \frac{m}{n}$$

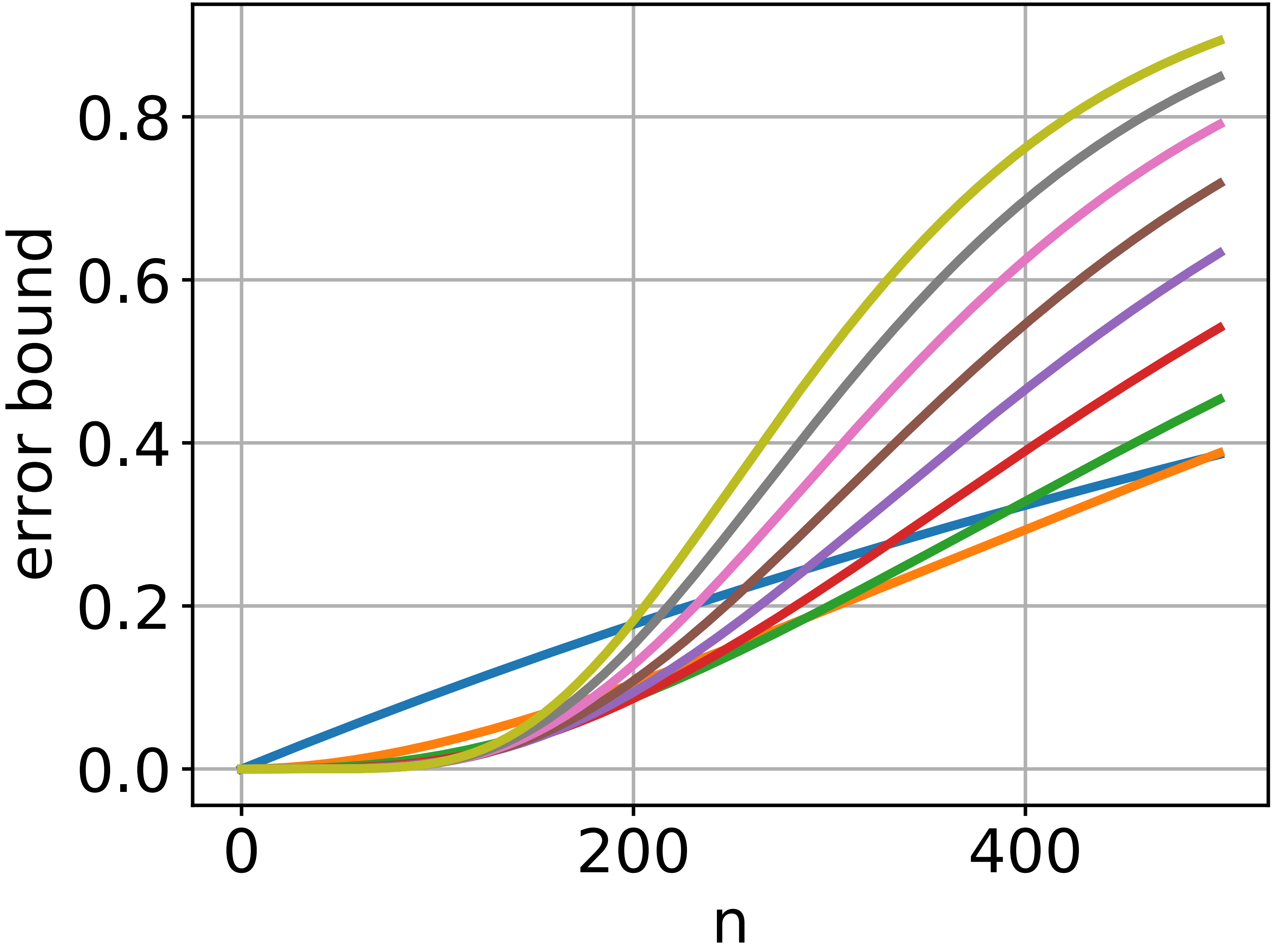
Picking k

$n, m, k \in \mathbb{N}$

- Idea: given $\boldsymbol{n}$ and $\boldsymbol{m}$, pick $\boldsymbol{k}$ to minimize ε.
- Optimal: $k = \ln 2 \cdot \frac{m}{n}$

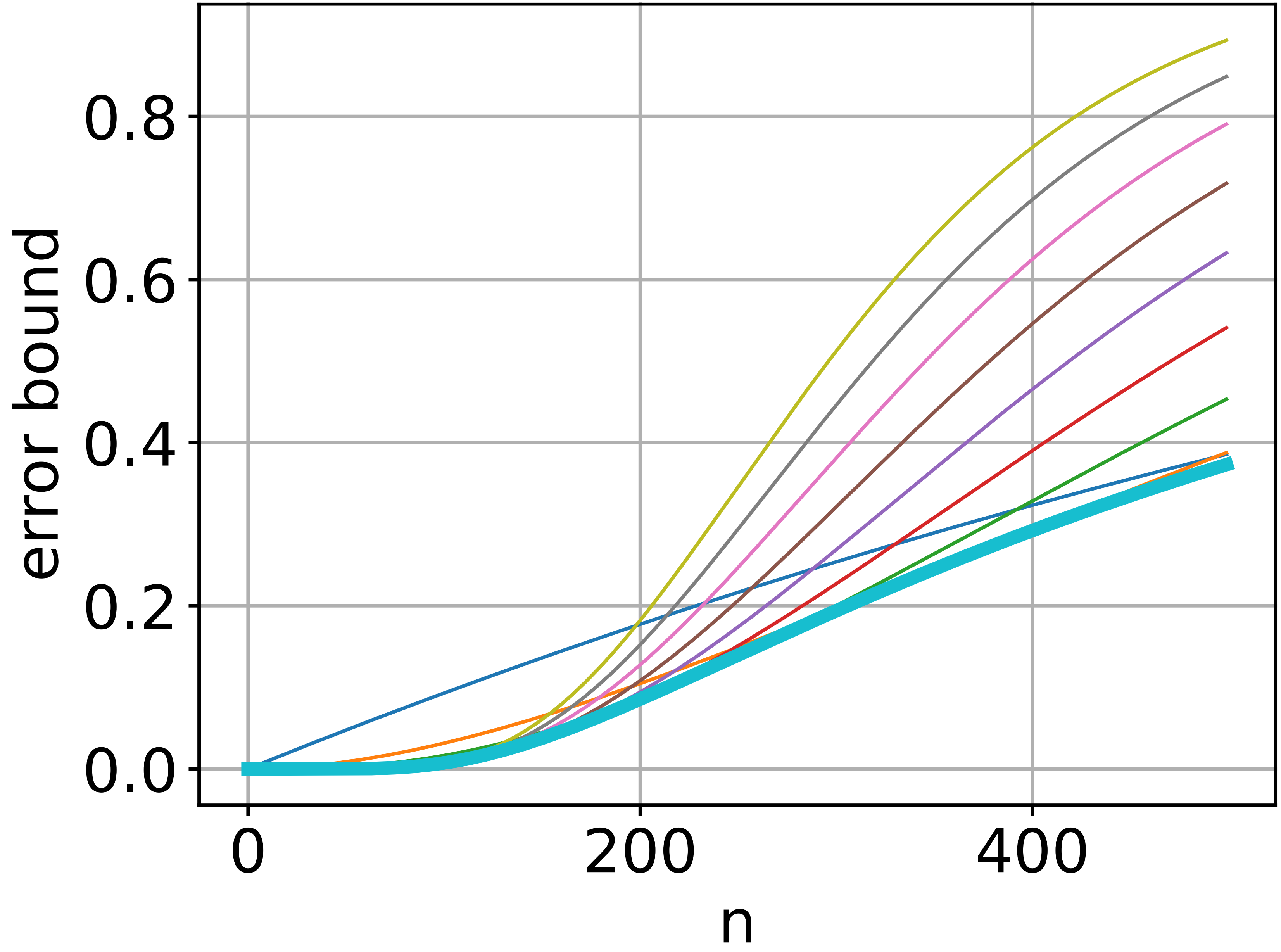
$$\varepsilon = \left(\tfrac{1}{2}\right)^k \approx 0.6185^{m/n}$$


1.0
0.8
0.6
0.4
0.2
0.0
error bound
0
200
400
n
m= 1 kb



Values of k

$m = 1$ kb



Optimal

k

$m = 1$ kb

Dictionary example

	n	m	k	ε
English dictionary	500.000	100 kB	1	≈ 46 %
(≈3 MB ASCII)		1 MB	1	≈ 5 %
		3 MB	1	≈ 1.9 %
		512 kB	1	≈ 11 %
			2	≈ 4 %
			6*	≈ 1 %
		1 MB	12*	≈ 0.03 %
		3 MB	35*	$< 10^{-10}$