Count Sketches and Bloom Filters

Recap from Last Time

Distribution Property:
Each element should have an equal probability of being placed in each slot.

For any $x \in \mathscr{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over $[m]$.

Independence Property:
Where one element is placed shouldn't impact where a second goes.

For any distinct $x, y \in \mathscr{U}$ and random $h \in \mathcal{H}$, $h(x)$ and $h(y)$ are independent random variables.

A family of hash functions $\mathscr{H}$ is called ***2-independent*** (or ***pairwise independent***) if it satisfies the distribution and independence properties.

Suppose there are two tunable values

$$\varepsilon \in (0, 1]$$
$$\delta \in (0, 1]$$

where ε represents ***accuracy*** and δ represents ***confidence***.

Goal: Make an estimator $\hat{A}$ for some quantity A where

With probability at least $1 - \delta$, ← ***Probably***

$|\hat{A} - A| \leq \varepsilon \cdot size(input)$ ← ***Approximately Correct***

for some measure of the size of the input.

What does it mean for an approximation to be "good"?

How to Build an Estimator

	Count-Min Sketch
Step One: Build a Simple Estimator	Hash items to counters; add +1 when item seen.
Step Two: Compute Expected Value of Estimator	Sum of indicators; 2-independent hashes have low collision rate.
Step Three: Apply Concentration Inequality	One-sided error; use expected value and Markov's inequality.
Step Four: Replicate to Boost Confidence	Take min; only fails if all estimates are bad.

New Stuff!

The Count Sketch

Frequency Estimation

- ***Recall:*** A frequency estimator is a data structure that supports
  - ***increment***(x), which increments the number of times that we've seen x, and
  - ***estimate***(x), which returns an estimate of how many times we've seen x.
- ***Notation:*** Assume that the elements we're processing are $x_1, \ldots, x_n$, and that the true frequency of element x_i is $\boldsymbol{a}_i$.
- Remember that the frequencies are not random variables – we're assuming that they're not under our control. Any randomness comes from hash functions.

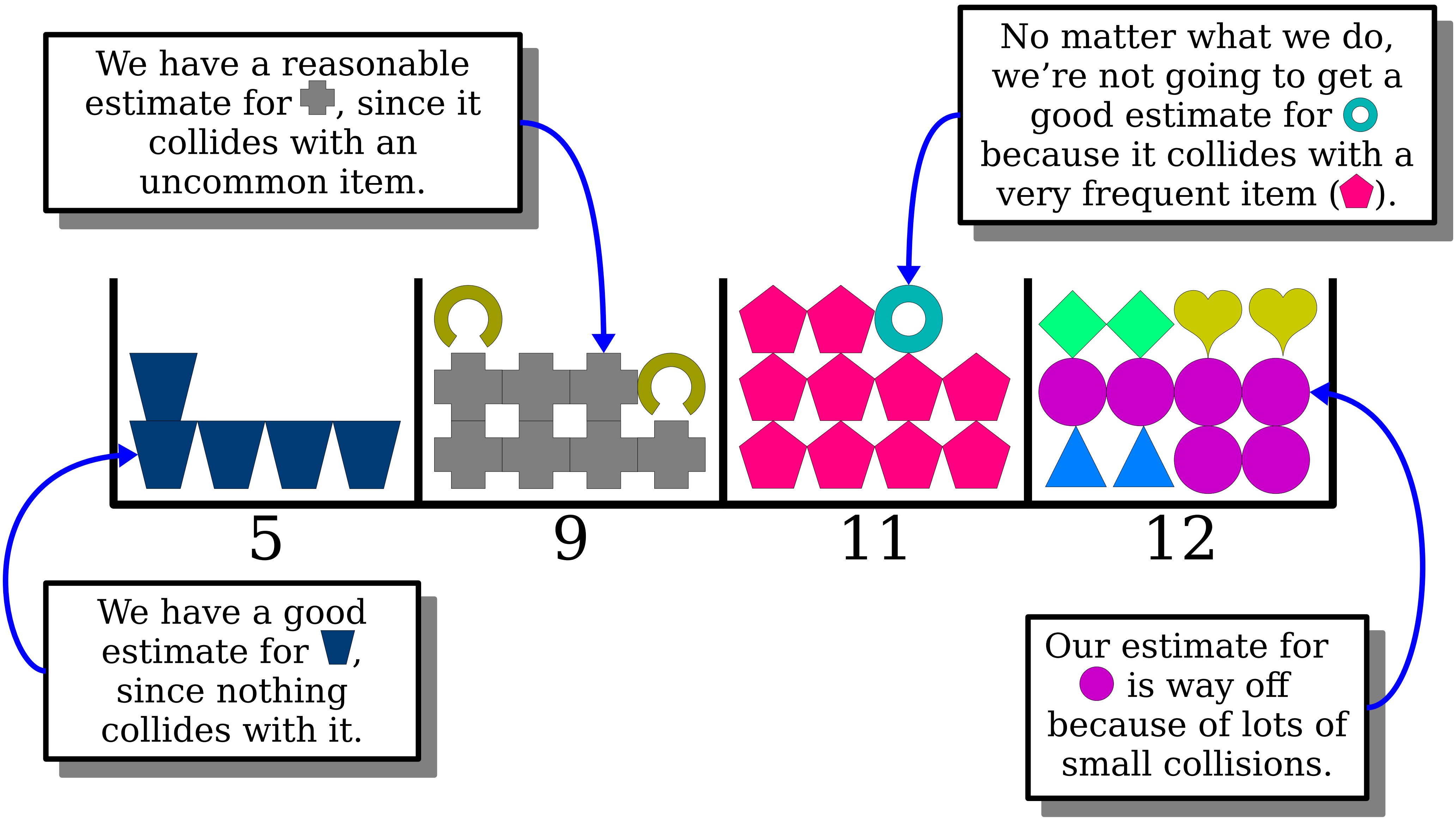
How to Build an Estimator

	Count-Min Sketch	***Count Sketch***
Step One: Build a Simple Estimator	Hash items to counters; add +1 when item seen.	
Step Two: Compute Expected Value of Estimator	Sum of indicators; 2-independent hashes have low collision rate.	
Step Three: Apply Concentration Inequality	One-sided error; use expected value and Markov's inequality.	
Step Four: Replicate to Boost Confidence	Take min; only fails if all estimates are bad.	

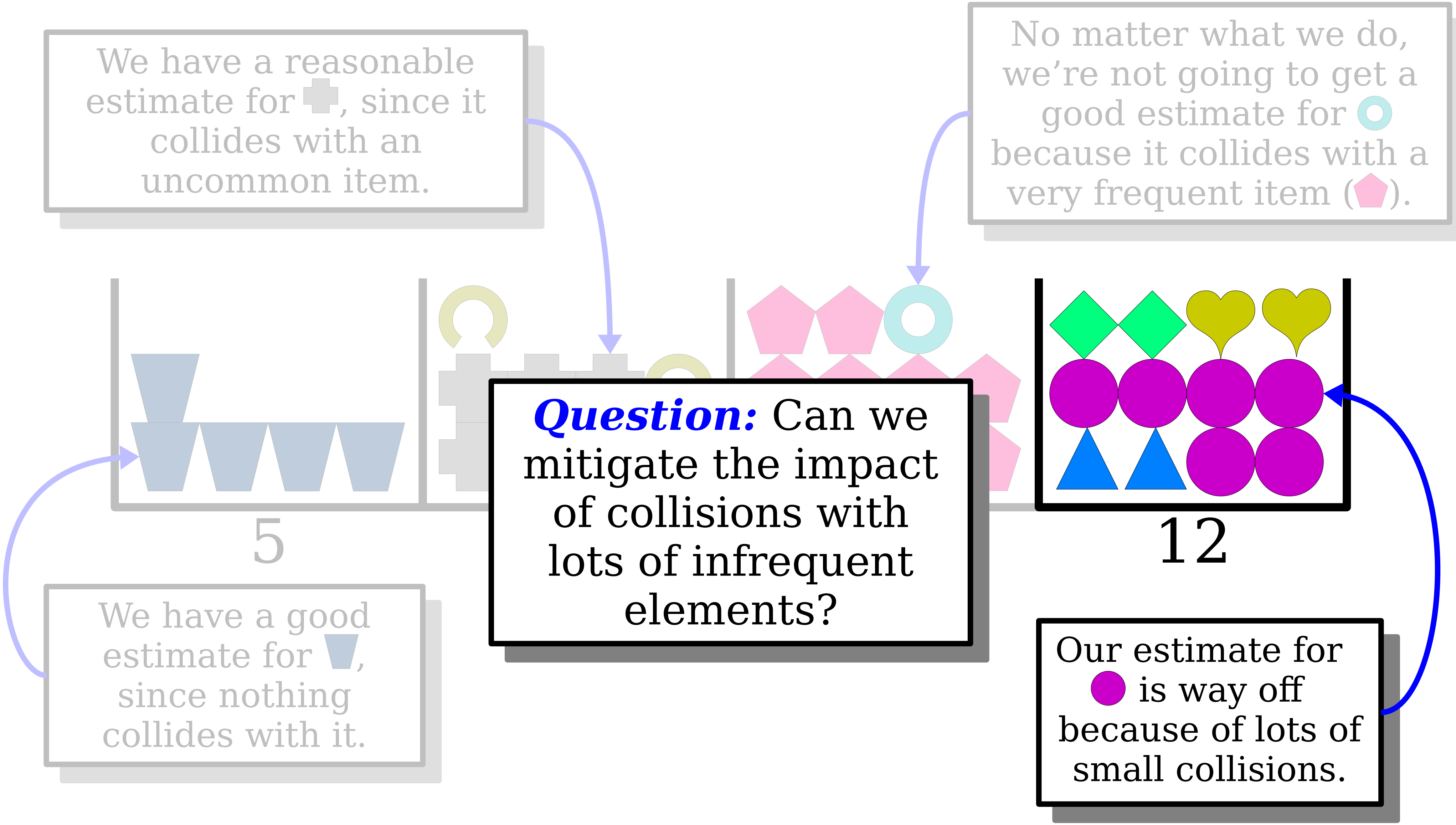
How to Build an Estimator

	Count-Min Sketch	***Count Sketch***
Step One: Build a Simple Estimator	Hash items to counters; add +1 when item seen.	
Step Two: Compute Expected Value of Estimator	Sum of indicators; 2-independent hashes have low collision rate.	
Step Three: Apply Concentration Inequality	One-sided error; use expected value and Markov's inequality.	
Step Four: Replicate to Boost Confidence	Take min; only fails if all estimates are bad.	

Revisiting Count-Min

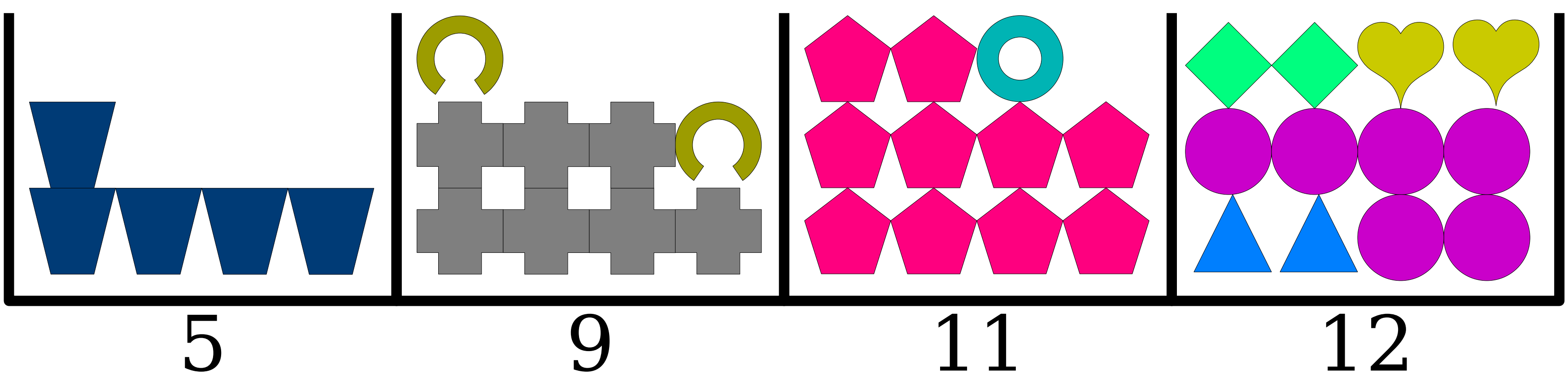


Revisiting Count-Min



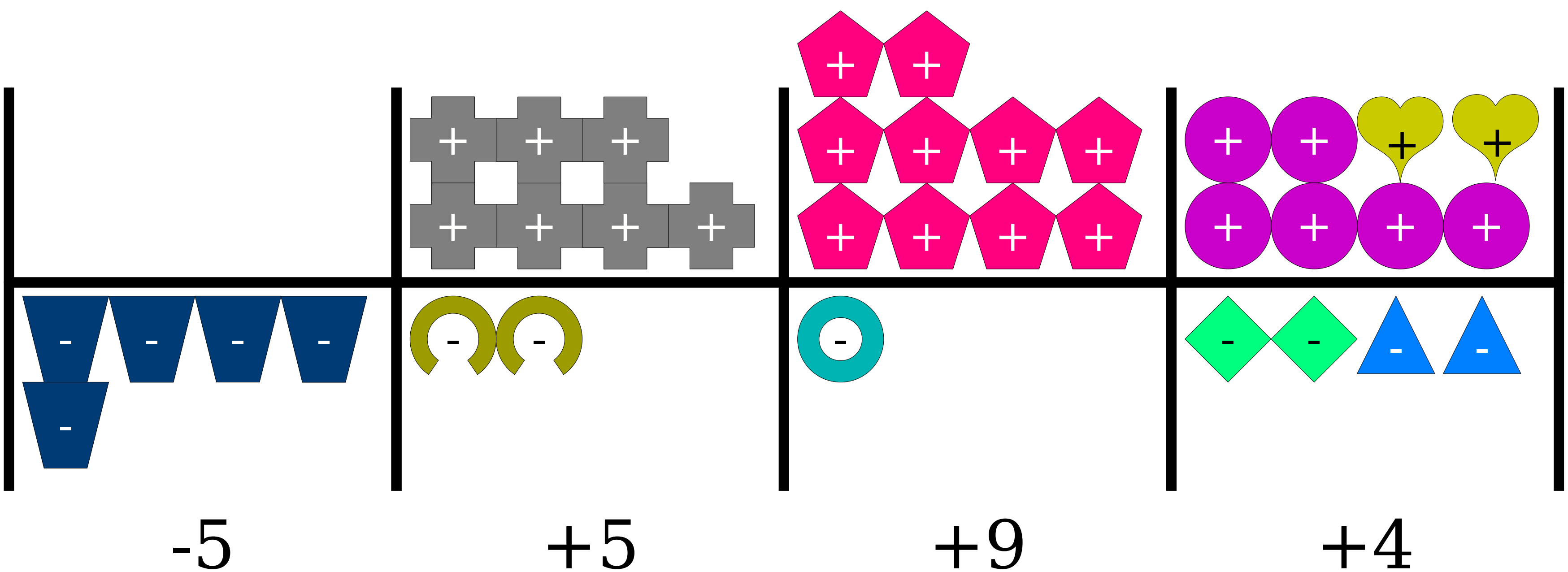
The Setup

- As before, create an array of counters and assign each item a counter.
- ***Key New Step:*** For each item x, assign x either +1 or -1.



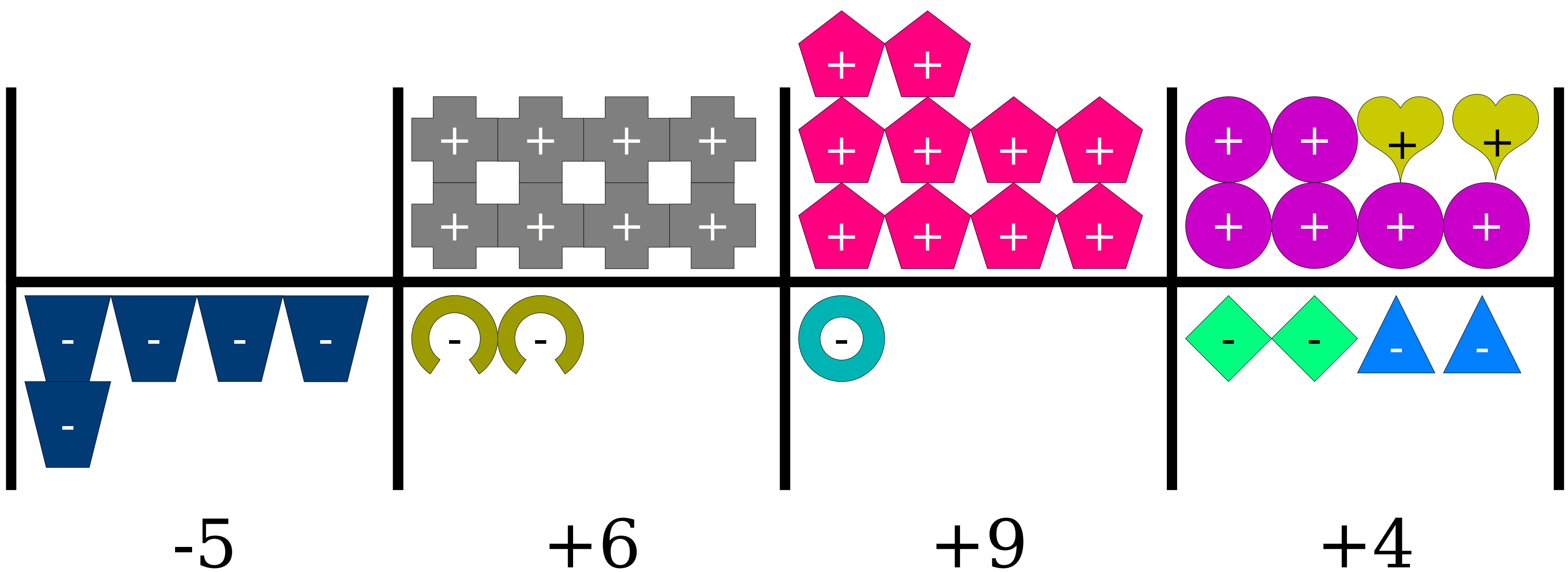
The Setup

- As before, create an array of counters and assign each item a counter.
- ***Key New Step:*** For each item x, assign x either +1 or -1.
  - To ***increment***(x), go to **`count`**[$h(x)$] and add ±1 as appropriate.
  - To ***estimate***(x), return **`count`**[$h(x)$], multiplied by ±1 as appropriate.



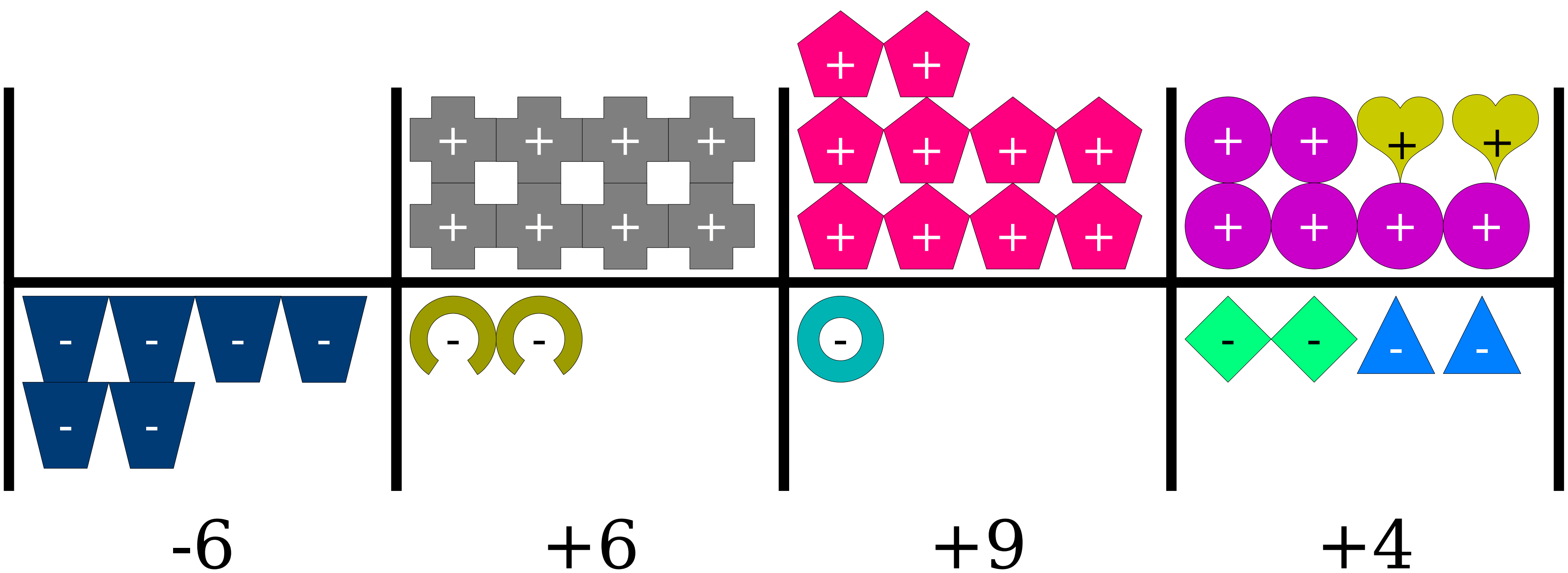
The Setup

- As before, create an array of counters and assign each item a counter.
- ***Key New Step:*** For each item x, assign x either +1 or -1.
  - To ***increment***(x), go to **`count`**[$h(x)$] and add ±1 as appropriate.
  - To ***estimate***(x), return **`count`**[$h(x)$], multiplied by ±1 as appropriate.



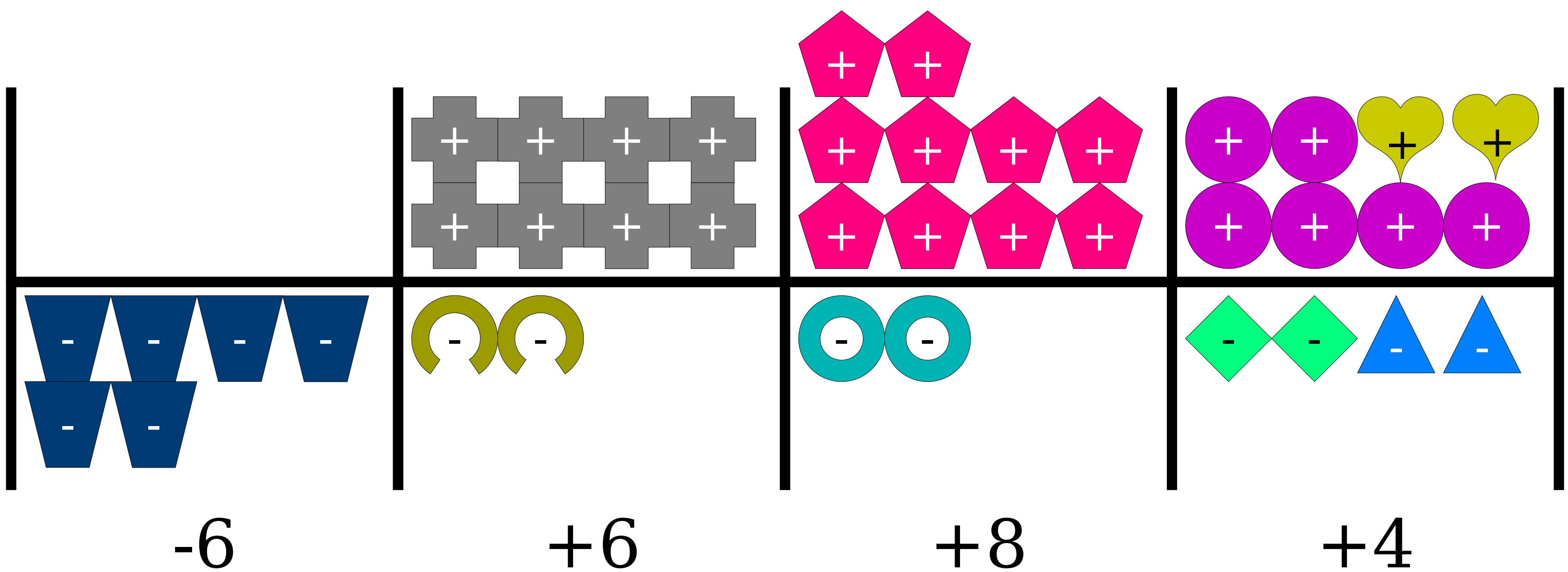
The Setup

- As before, create an array of counters and assign each item a counter.
- ***Key New Step:*** For each item x, assign x either +1 or -1.
  - To ***increment***(x), go to **`count`**[h(x)] and add ±1 as appropriate.
  - To ***estimate***(x), return **`count`**[h(x)], multiplied by ±1 as appropriate.

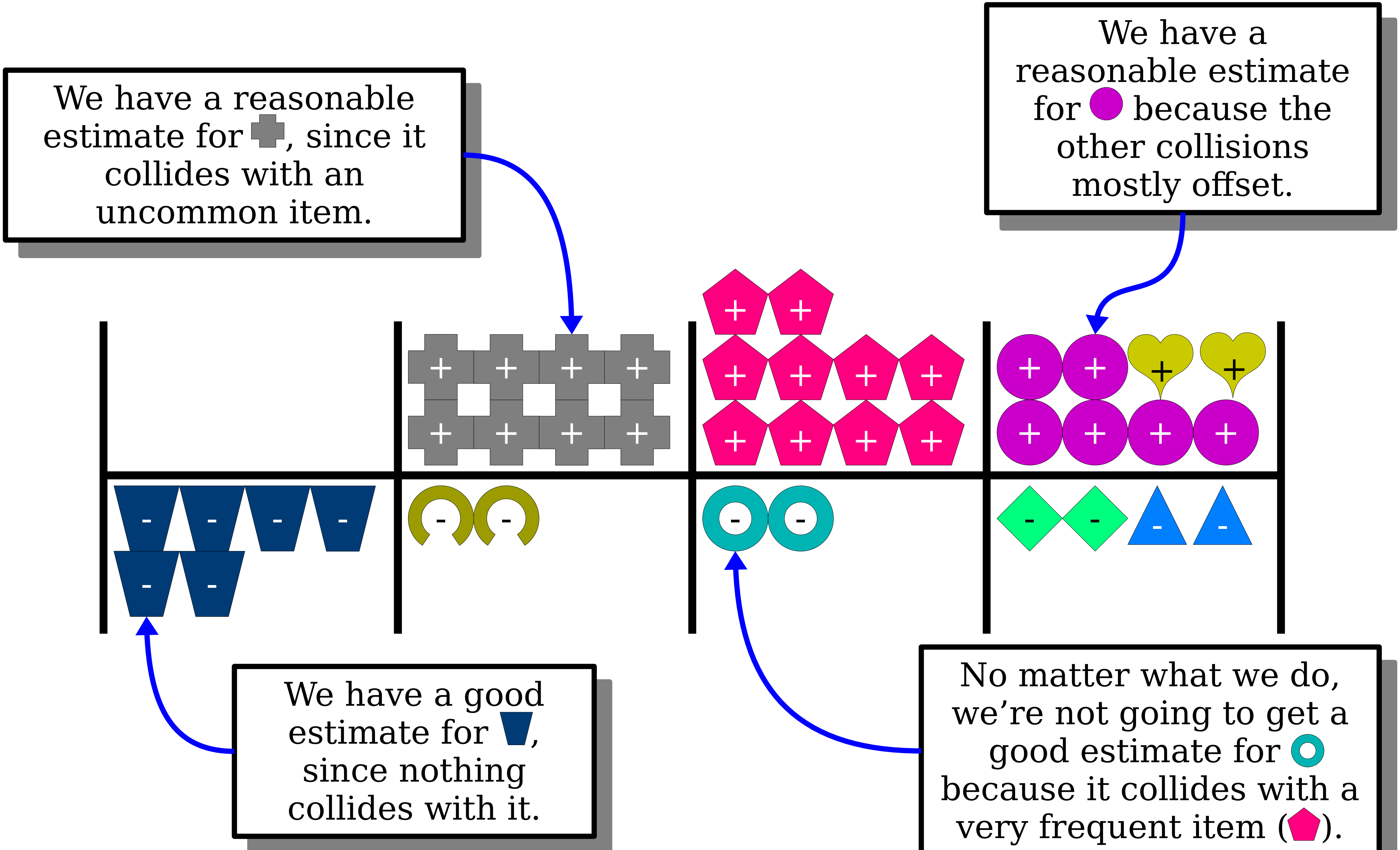


The Setup

- As before, create an array of counters and assign each item a counter.
- ***Key New Step:*** For each item x, assign x either +1 or -1.
  - To ***increment***(x), go to **`count`**[$h(x)$] and add ±1 as appropriate.
  - To ***estimate***(x), return **`count`**[$h(x)$], multiplied by ±1 as appropriate.

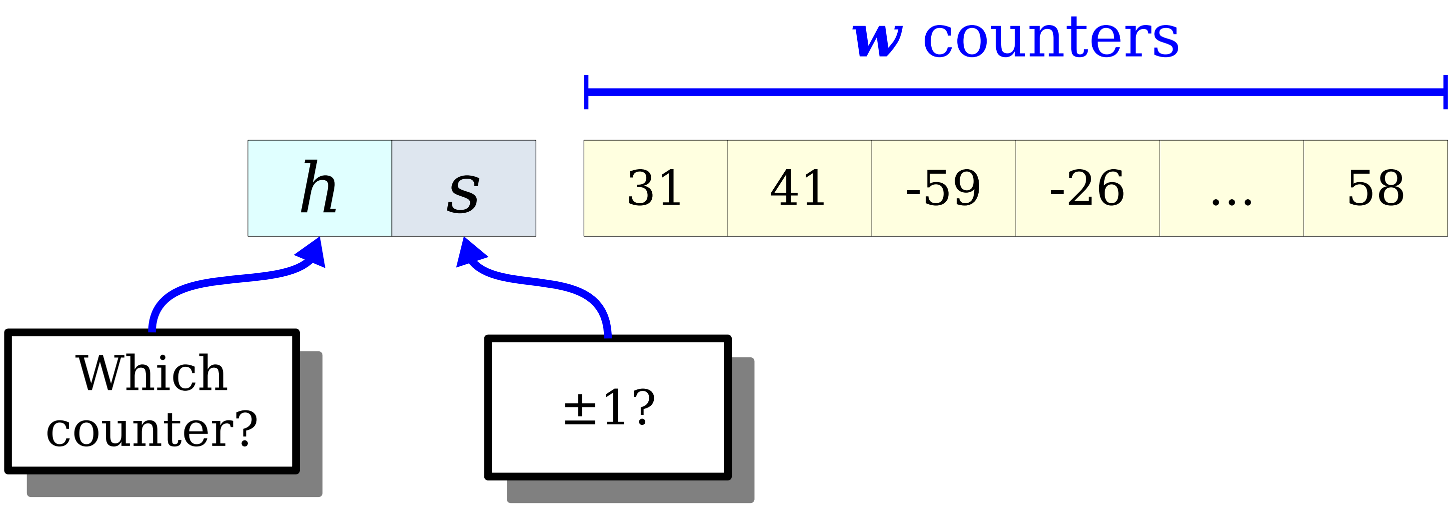


The Setup


We have a reasonable estimate for , since it collides with an uncommon item.
We have a reasonable estimate for because the other collisions mostly offset.
We have a good estimate for , since nothing collides with it.
No matter what we do, we're not going to get a good estimate for because it collides with a very frequent item ( ).

Formalizing This

- Maintain an array of counters of length w.
- Pick $h \in \mathscr{H}$ chosen uniformly at random from a 2-independent family of hash functions from $\mathscr{U}$. to w.
- Pick $s \in \mathscr{U}$ uniformly randomly and independently of h from a 2-independent family from $\mathscr{U}$ to {-1, +1}.
- ***increment***(x): **`count`**$[h(x)]$ += $s(x)$.
- ***estimate***(x), return $s(x) \cdot$ **`count`**$[h(x)]$.



How to Build an Estimator

	Count-Min Sketch	***Count Sketch***
Step One: Build a Simple Estimator	Hash items to counters; add +1 when item seen.	Hash items to counters; add ±1 when item seen.
Step Two: Compute Expected Value of Estimator	Sum of indicators; 2-independent hashes have low collision rate.	
Step Three: Apply Concentration Inequality	One-sided error; use expected value and Markov's inequality.	
Step Four: Replicate to Boost Confidence	Take min; only fails if all estimates are bad.	

How to Build an Estimator

	Count-Min Sketch	*Count Sketch*
Step One: Build a Simple Estimator	Hash items to counters; add +1 when item seen.	Hash items to counters; add ±1 when item seen.
Step Two: Compute Expected Value of Estimator	Sum of indicators; 2-independent hashes have low collision rate.	
Step Three: Apply Concentration Inequality	One-sided error; use expected value and Markov’s inequality.	
Step Four: Replicate to Boost Confidence	Take min; only fails if all estimates are bad.	

Formalizing the Intuition

- Define $\hat{\boldsymbol{a}}_i$ to be our estimate of $\boldsymbol{a}_i$.
- As before, $\hat{\boldsymbol{a}}_i$ will depend on how the other elements are distributed. Unlike before, it now also depends on signs given to the elements by s.
- Specifically, for each other x_j that collides with x_i, the estimate $\hat{\boldsymbol{a}}_i$ includes an error term of

$$s(x_i) \cdot s(x_j) \cdot \boldsymbol{a}_j$$

- Why?

Formalizing the Intuition

- Define $\hat{\boldsymbol{a}}_i$ to be our estimate of $\boldsymbol{a}_i$.
- As before, $\hat{\boldsymbol{a}}_i$ will depend on how the other elements are distributed. Unlike before, it now also depends on signs given to the elements by s.
- Specifically, for each other x_j that collides with x_i, the estimate $\hat{\boldsymbol{a}}_i$ includes an error term of

$$s(x_i) \cdot s(x_j) \cdot \boldsymbol{a}_j$$

- Why?
  - The counter for x_i will have $s(x_j)\ \boldsymbol{a}_j$ added in.
  - We multiply the counter by $s(x_i)$ before returning it.

Formalizing the Intuition

- Define $\hat{\boldsymbol{a}}_i$ to be our estimate of $\boldsymbol{a}_i$.
- As before, $\hat{\boldsymbol{a}}_i$ will depend on how the other elements are distributed. Unlike before, it now also depends on signs given to the elements by s.
- Specifically, for each other x_j that collides with x_i, the estimate $\hat{\boldsymbol{a}}_i$ includes an error term of

$$s(x_i) \cdot s(x_j) \cdot \boldsymbol{a}_j$$

- Why?
  - If $s(x_i)$ and $s(x_j)$ point in the same direction, the terms add to the total.
  - If $s(x_i)$ and $s(x_j)$ point in different directions, the terms subtract from the total.

Formalizing the Intuition

- In our quest to learn more about $\boldsymbol{\hat{a}}_i$, let's have X_j be a random variable indicating whether x_i and x_j collided with one another:

$$X_j = \begin{cases} 1 & \text{if } h(x_i) = h(x_j) \\ 0 & \text{if } h(x_i) \neq h(x_j) \end{cases}$$

Formalizing the Intuition

- In our quest to learn more about $\hat{\boldsymbol{a}}_i$, let's have X_j be a random variable indicating whether x_i and x_j collided with one another:

$$X_j = \begin{cases} 1 & \text{if } h(x_i) = h(x_j) \\ 0 & \text{if } h(x_i) \neq h(x_j) \end{cases}$$

- We can then express $\hat{\boldsymbol{a}}_i$ in terms of the signed contributions from the items x_i collides with:

$$\hat{\boldsymbol{a}}_i = \sum_j \boldsymbol{a}_j s(x_i) s(x_j) X_j$$

This is how much the collision impacts our estimate.

We only care about items we collided with.

Formalizing the Intuition

- In our quest to learn more about $\hat{\boldsymbol{a}}_i$, let's have X_j be a random variable indicating whether x_i and x_j collided with one another:

$$X_j = \begin{cases} 1 & \text{if } h(x_i) = h(x_j) \\ 0 & \text{if } h(x_i) \neq h(x_j) \end{cases}$$

- We can then express $\hat{\boldsymbol{a}}_i$ in terms of the signed contributions from the items x_i collides with:

$$\hat{\boldsymbol{a}}_i = \sum_j \boldsymbol{a}_j s(x_i) s(x_j) X_j = \boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j$$

This is how much the collision impacts our estimate.

We only care about items we collided with.

$$\mathrm{E}[\hat{\boldsymbol{a}}_i] = \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j]$$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j]
\end{aligned}
$$

Hey, it's linearity of expectation!

$$
\begin{aligned}
E[\hat{\boldsymbol{a}}_i] &= E[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= E[\boldsymbol{a}_i] + E[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} E[\boldsymbol{a}_j s(x_i) s(x_j) X_j]
\end{aligned}
$$

Remember that $\boldsymbol{a}_i$ and the like aren't random variables.

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j s(x_i) s(x_j) X_j]
\end{aligned}
$$

We chose the hash functions h and s independently of one another.

$$
X_j = \begin{cases} 1 & \text{if } h(x_i) = h(x_j) \\ 0 & \text{if } h(x_i) \neq h(x_j) \end{cases}
$$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i) s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j]
\end{aligned}
$$

We chose the hash functions h and s independently of one another.

$$
X_j = \begin{cases} 1 & \text{if } h(x_i) = h(x_j) \\ 0 & \text{if } h(x_i) \neq h(x_j) \end{cases}
$$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j\neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j\neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j\neq i} \mathrm{E}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j\neq i} \mathrm{E}[s(x_i) s(x_j)]\mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \boldsymbol{a}_i + \sum_{j\neq i} \mathrm{E}[s(x_i)]\mathrm{E}[s(x_j)]\mathrm{E}[\boldsymbol{a}_j X_j]
\end{aligned}
$$

Since s is drawn from a 2-independent family of hash functions, we know $s(x_i)$ and $s(x_j)$ are independent random variables.

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j\neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j\neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j\neq i} \mathrm{E}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j\neq i} \mathrm{E}[s(x_i) s(x_j)]\mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \boldsymbol{a}_i + \sum_{j\neq i} \mathrm{E}[s(x_i)]\mathrm{E}[s(x_j)]\mathrm{E}[\boldsymbol{a}_j X_j]
\end{aligned}
$$

$\mathrm{E}[s(x_i)] =$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i) s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i)] \mathrm{E}[s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j]
\end{aligned}
$$

$\mathrm{E}[s(x_i)] =$

s is drawn from a 2-independent family of hash functions.

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i) s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i)] \mathrm{E}[s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j]
\end{aligned}
$$

$\mathrm{E}[s(x_i)] =$

s is drawn from a 2-independent family of hash functions.

$s(x_i)$ is uniform over $\{-1, +1\}$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i) s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i)] \mathrm{E}[s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j]
\end{aligned}
$$

$\mathrm{E}[s(x_i)] =$

s is drawn from a 2-independent family of hash functions.

$s(x_i)$ is uniform over {-1, +1}

$\Pr[s(x_i) = -1] = ½$ $\Pr[s(x_i) = +1] = ½$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i) s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i)] \mathrm{E}[s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j]
\end{aligned}
$$

$\mathrm{E}[s(x_i)] = ½ \cdot (-1) + ½ \cdot (+1)$

s is drawn from a 2-independent family of hash functions.

$s(x_i)$ is uniform over $\{-1, +1\}$

$\Pr[s(x_i) = -1] = ½$ $\Pr[s(x_i) = +1] = ½$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i) s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i)] \mathrm{E}[s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[s(x_i)] &= \tfrac{1}{2} \cdot (-1) + \tfrac{1}{2} \cdot (+1) \\
&= 0
\end{aligned}
$$

s is drawn from a 2-independent family of hash functions.

$s(x_i)$ is uniform over $\{-1, +1\}$

$\Pr[s(x_i) = -1] = \tfrac{1}{2}$ $\Pr[s(x_i) = +1] = \tfrac{1}{2}$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i) s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i)] \mathrm{E}[s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} 0
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[s(x_i)] &= \tfrac{1}{2} \cdot (-1) + \tfrac{1}{2} \cdot (+1) \\
&= 0
\end{aligned}
$$

s is drawn from a 2-independent family of hash functions.

$s(x_i)$ is uniform over $\{-1, +1\}$

$\Pr[s(x_i) = -1] = \tfrac{1}{2}$ $\Pr[s(x_i) = +1] = \tfrac{1}{2}$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i] &= \mathrm{E}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{E}[\boldsymbol{a}_i] + \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i) s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} \mathrm{E}[s(x_i)] \mathrm{E}[s(x_j)] \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \boldsymbol{a}_i + \sum_{j \neq i} 0 \\
&= \boldsymbol{a}_i
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[s(x_i)] &= \tfrac{1}{2} \cdot (-1) + \tfrac{1}{2} \cdot (+1) \\
&= 0
\end{aligned}
$$

s is drawn from a 2-independent family of hash functions.

$s(x_i)$ is uniform over $\{-1, +1\}$

$\Pr[s(x_i) = -1] = \tfrac{1}{2}$ $\Pr[s(x_i) = +1] = \tfrac{1}{2}$

How to Build an Estimator

	Count-Min Sketch	***Count Sketch***
Step One: Build a Simple Estimator	Hash items to counters; add +1 when item seen.	Hash items to counters; add ±1 when item seen.
Step Two: Compute Expected Value of Estimator	Sum of indicators; 2-independent hashes have low collision rate.	2-independence breaks up products; ±1 variables have zero expected value.
Step Three: Apply Concentration Inequality	One-sided error; use expected value and Markov's inequality.	
Step Four: Replicate to Boost Confidence	Take min; only fails if all estimates are bad.	

How to Build an Estimator

	Count-Min Sketch	*Count Sketch*
Step One: Build a Simple Estimator	Hash items to counters; add +1 when item seen.	Hash items to counters; add ±1 when item seen.
Step Two: Compute Expected Value of Estimator	Sum of indicators; 2-independent hashes have low collision rate.	2-independence breaks up products; ±1 variables have zero expected value.
Step Three: Apply Concentration Inequality	One-sided error; use expected value and Markov’s inequality.	
Step Four: Replicate to Boost Confidence	Take min; only fails if all estimates are bad.	

A Hitch

- In the count-min sketch, we used Markov's inequality to bound the probability that we get a bad estimate.
- This worked because we had a ***one-sided error***: the distance $\hat{a}_i - a_i$ from the true answer was nonnegative.
- With the count sketch, we have a ***two-sided error***: $\hat{a}_i - a_i$ can be negative in the count sketch because collisions can *decrease* the estimate $\hat{a}_i$ below the true value a_i.
- We'll need to use a different technique to bound the error.

Chebyshev to the Rescue

- ***Chebyshev's inequality*** states that for any random variable X with finite variance, given any $c > 0$, we have

$$\Pr\left[\ |X - \mathrm{E}[X]| \geq c\ \right] \leq \frac{\mathrm{Var}[X]}{c^2}.$$

- If we can get the variance of $\hat{\boldsymbol{a}}_i$, we can bound the probability that we get a bad estimate with our data structure.

$$\mathrm{Var}[\hat{\boldsymbol{a}}_i] \quad = \quad \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j]$$

$$\mathrm{Var}[\hat{\boldsymbol{a}}_i] = \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j]$$

$$\boxed{\mathrm{Var}[a + X] = \mathrm{Var}[X]}$$

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j]
\end{aligned}
$$

$$
\boxed{\mathrm{Var}[a + X] = \mathrm{Var}[X]}
$$

$$\begin{aligned}\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j\neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\ &= \boxed{\mathrm{Var}[\sum_{j\neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j]}\end{aligned}$$

In general, Var is *not* a linear operator.

However, if the terms in the sum are ***pairwise uncorrelated***, then Var is linear.

Lemma: The terms in this sum are uncorrelated. *(Prove this!)*

$$\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \boxed{\mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j]} \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j]
\end{aligned}$$

In general, Var is *not* a linear operator.

However, if the terms in the sum are ***pairwise uncorrelated***, then Var is linear.

Lemma: The terms in this sum are uncorrelated. *(Prove this!)*

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j]
\end{aligned}
$$

The “Sum-o’-Var” Samovar!

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Var}[Z] &= \mathrm{E}[Z^2] - \mathrm{E}[Z]^2 \\
&\leq \mathrm{E}[Z^2]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j\neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j\neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \sum_{j\neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&\leq \sum_{j\neq i} \mathrm{E}[(\boldsymbol{a}_j s(x_i) s(x_j) X_j)^2]
\end{aligned}
$$

$$
\boxed{\begin{aligned}\mathrm{Var}[Z] &= \mathrm{E}[Z^2] - \mathrm{E}[Z]^2 \\ &\leq \mathrm{E}[Z^2]\end{aligned}}
$$

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&\leq \sum_{j \neq i} \mathrm{E}[(\boldsymbol{a}_j s(x_i) s(x_j) X_j)^2] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j^2 s(x_i)^2 s(x_j)^2 X_j^2]
\end{aligned}
$$

$$\boxed{\begin{array}{c} s(x) = \pm 1, \\ \text{so} \\ s(x)^2 = 1 \end{array}}$$

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&\leq \sum_{j \neq i} \mathrm{E}[(\boldsymbol{a}_j s(x_i) s(x_j) X_j)^2] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j^2 s(x_i)^2 s(x_j)^2 X_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j^2]
\end{aligned}
$$

$$
\boxed{\begin{array}{c} s(x) = \pm 1, \\ \text{so} \\ s(x)^2 = 1 \end{array}}
$$

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&\leq \sum_{j \neq i} \mathrm{E}[(\boldsymbol{a}_j s(x_i) s(x_j) X_j)^2] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j^2 s(x_i)^2 s(x_j)^2 X_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j^2]
\end{aligned}
$$

$$
\boxed{X_j = \begin{cases} 1 & \text{if } h(x_i) = h(x_j) \\ 0 & \text{if } h(x_i) \neq h(x_j) \end{cases}}
$$

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&\leq \sum_{j \neq i} \mathrm{E}[(\boldsymbol{a}_j s(x_i) s(x_j) X_j)^2] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j^2 s(x_i)^2 s(x_j)^2 X_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j^2]
\end{aligned}
$$

$$
\boxed{X_j^{\mathbf{2}} = \begin{cases} 1^{\mathbf{2}} & \text{if } h(x_i) = h(x_j) \\ 0^{\mathbf{2}} & \text{if } h(x_i) \neq h(x_j) \end{cases}}
$$

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&\leq \sum_{j \neq i} \mathrm{E}[(\boldsymbol{a}_j s(x_i) s(x_j) X_j)^2] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j^2 s(x_i)^2 s(x_j)^2 X_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j^2]
\end{aligned}
$$

Useful Fact: If X is an indicator, then $X^2 = X$.

$$
X_j^{\mathbf{2}} = \begin{cases} 1 & \text{if } h(x_i) = h(x_j) \\ 0 & \text{if } h(x_i) \neq h(x_j) \end{cases}
$$

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) \boldsymbol{X}_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) \boldsymbol{X}_j] \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) \boldsymbol{X}_j] \\
&\leq \sum_{j \neq i} \mathrm{E}[(\boldsymbol{a}_j s(x_i) s(x_j) \boldsymbol{X}_j)^2] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j^2 s(x_i)^2 s(x_j)^2 \boldsymbol{X}_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j]
\end{aligned}
$$

Useful Fact: If X is an indicator, then $X^2 = X$.

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&\leq \sum_{j \neq i} \mathrm{E}[(\boldsymbol{a}_j s(x_i) s(x_j) X_j)^2] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j^2 s(x_i)^2 s(x_j)^2 X_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j]
\end{aligned}
$$

$$
\boxed{X_j = \begin{cases} 1 & \text{if } h(x_i) = h(x_j) \\ 0 & \text{if } h(x_i) \neq h(x_j) \end{cases}}
$$

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&\leq \sum_{j \neq i} \mathrm{E}[(\boldsymbol{a}_j s(x_i) s(x_j) X_j)^2] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j^2 s(x_i)^2 s(x_j)^2 X_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j] \\
&= \frac{1}{w} \sum_{j \neq i} \boldsymbol{a}_j^2
\end{aligned}
$$

$$
\boxed{X_j = \begin{cases} 1 & \text{if } h(x_i) = h(x_j) \\ 0 & \text{if } h(x_i) \neq h(x_j) \end{cases}}
$$

$$
\begin{aligned}
\mathrm{Var}[\hat{\boldsymbol{a}}_i] &= \mathrm{Var}[\boldsymbol{a}_i + \sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \mathrm{Var}[\sum_{j \neq i} \boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&= \sum_{j \neq i} \mathrm{Var}[\boldsymbol{a}_j s(x_i) s(x_j) X_j] \\
&\leq \sum_{j \neq i} \mathrm{E}[(\boldsymbol{a}_j s(x_i) s(x_j) X_j)^2] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j^2 s(x_i)^2 s(x_j)^2 X_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j^2] \\
&= \sum_{j \neq i} \boldsymbol{a}_j^2 \mathrm{E}[X_j] \\
&= \frac{1}{w} \sum_{j \neq i} \boldsymbol{a}_j^2
\end{aligned}
$$

I know this might look really dense, but many of these substeps end up being really useful techniques. These ideas generalize, I promise.

Think of [$\boldsymbol{a}_1$, $\boldsymbol{a}_2$, $\boldsymbol{a}_3$, …] as a vector.

What does the following quantity represent?

$$\sum_j \boldsymbol{a}_j^2$$

$$\mathrm{Var}[\hat{\boldsymbol{a}}_i] \leq \frac{1}{w}\sum_{j \neq i} \boldsymbol{a}_j^2$$

Think of [$\boldsymbol{a}_1$, $\boldsymbol{a}_2$, $\boldsymbol{a}_3$, …] as a vector.

What does the following quantity represent?

$$\sum_j \boldsymbol{a}_j^2$$

This is the square of the magnitude of the vector!

$$\mathrm{Var}[\hat{\boldsymbol{a}}_i] \leq \frac{1}{w} \sum_{j \neq i} \boldsymbol{a}_j^2$$

Think of [$\boldsymbol{a}_1$, $\boldsymbol{a}_2$, $\boldsymbol{a}_3$, …] as a vector.

What does the following quantity represent?

$$\sum_j \boldsymbol{a}_j^2$$

This is the square of the magnitude of the vector!

The magnitude of a vector is called its ***L_2 norm*** and is denoted $\|\boldsymbol{a}\|_2$.

$$\|\boldsymbol{a}\|_2 = \sqrt{\sum_j \boldsymbol{a}_j^2}$$

$$\mathrm{Var}[\hat{\boldsymbol{a}}_i] \le \frac{1}{w} \sum_{j \neq i} \boldsymbol{a}_j^2$$

Think of [$\boldsymbol{a}_1$, $\boldsymbol{a}_2$, $\boldsymbol{a}_3$, …] as a vector.

What does the following quantity represent?

$$\sum_j \boldsymbol{a}_j^2$$

This is the square of the magnitude of the vector!

The magnitude of a vector is called its ***L₂ norm*** and is denoted $\|\boldsymbol{a}\|_2$.

$$\|\boldsymbol{a}\|_2 \quad = \quad \sqrt{\sum_j \boldsymbol{a}_j^2}$$

Therefore, our above sum is $\|\boldsymbol{a}\|_2^2$.

$$\mathrm{Var}[\hat{\boldsymbol{a}}_i] \quad \leq \quad \frac{1}{w} \sum_{j \neq i} \boldsymbol{a}_j^2$$

Think of [$\boldsymbol{a}_1$, $\boldsymbol{a}_2$, $\boldsymbol{a}_3$, …] as a vector.

What does the following quantity represent?

$$\sum_j \boldsymbol{a}_j^2$$

This is the square of the magnitude of the vector!

The magnitude of a vector is called its ***L_2* norm** and is denoted $\|\boldsymbol{a}\|_2$.

$$\|\boldsymbol{a}\|_2 = \sqrt{\sum_j \boldsymbol{a}_j^2}$$

Therefore, our above sum is $\|\boldsymbol{a}\|_2^2$.

$$\mathrm{Var}[\hat{\boldsymbol{a}}_i] \leq \frac{1}{w}\sum_{j \neq i} \boldsymbol{a}_j^2 \leq \frac{\|\boldsymbol{a}\|_2^2}{w}$$

Think of [$\boldsymbol{a}_1$, $\boldsymbol{a}_2$, $\boldsymbol{a}_3$, …] as a vector.

What does the following quantity represent?

$$\sum_j \boldsymbol{a}_j^2$$

This is the square of the mag

The magnitude of a vector is c

is denoted $\|$

Great exercise: Prove that the L_2 norm of a vector is never greater than the L_1 norm.

$$\|\boldsymbol{a}\|_2 = \sqrt{\sum_j \boldsymbol{a}_j^2}$$

Therefore, our above sum is $\|\boldsymbol{a}\|_2^2$.

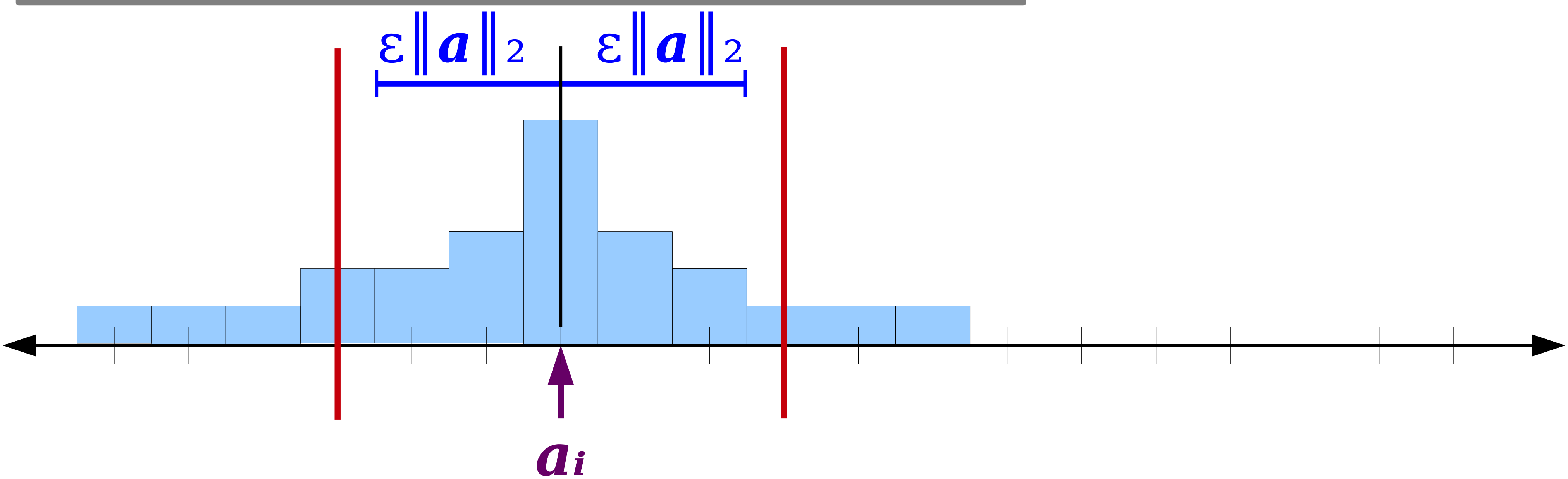
$$\mathrm{Var}[\hat{\boldsymbol{a}}_i] \le \frac{1}{w}\sum_{j \ne i} \boldsymbol{a}_j^2 \le \frac{\|\boldsymbol{a}\|_2^2}{w}$$

Goal: Make an estimator $\hat{\boldsymbol{a}}$ for some quantity $\boldsymbol{a}$ where

With probability at least $1 - \delta$, — ***Probably***

$|\hat{\boldsymbol{a}} - \boldsymbol{a}| \le \varepsilon \cdot \mathit{size}(\mathit{input})$ — ***Approximately Correct***

for some measure of the size of the input.



$$\mathrm{Var}[\hat{\boldsymbol{a}}_i] \le \frac{\|\boldsymbol{a}\|_2^2}{w}$$

$$\Pr\left[\left|\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right| > \varepsilon \|\boldsymbol{a}\|_2\right]$$

$$\Pr\left[\left|\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right| > \varepsilon \|\boldsymbol{a}\|_2\right]$$

Chebyshev's inequality says that

$$\Pr\left[\ |X - \mathrm{E}[X]| \geq c\ \right] \leq \frac{\mathrm{Var}[X]}{c^2}.$$

$$\Pr\left[\left| \hat{\boldsymbol{a}}_i - \boldsymbol{a}_i \right| > \varepsilon \|\boldsymbol{a}\|_2 \right]$$

$$\leq \frac{\mathrm{Var}[\hat{\boldsymbol{a}}_i]}{\left(\varepsilon \|\boldsymbol{a}\|_2\right)^2}$$

Chebyshev's inequality says that

$$\Pr\left[\left| X - \mathrm{E}[X] \right| \geq c \right] \leq \frac{\mathrm{Var}[X]}{c^2}.$$

$$\Pr\left[\left|\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right| > \varepsilon \|\boldsymbol{a}\|_2\right]$$

$$\leq \frac{\mathrm{Var}[\hat{\boldsymbol{a}}_i]}{\left(\varepsilon \|\boldsymbol{a}\|_2\right)^2}$$

$$\boxed{\mathrm{Var}[\hat{\boldsymbol{a}}_i] \leq \frac{\|\boldsymbol{a}\|_2^2}{w}}$$

$$\Pr\left[\left|\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right| > \varepsilon \|\boldsymbol{a}\|_2\right]$$

$$\leq \quad \frac{\mathrm{Var}[\hat{\boldsymbol{a}}_i]}{\left(\varepsilon \|\boldsymbol{a}\|_2\right)^2}$$

$$\leq \quad \frac{\|\boldsymbol{a}\|_2^2}{w} \cdot \frac{1}{\left(\varepsilon \|\boldsymbol{a}\|_2\right)^2}$$

$$\boxed{\mathrm{Var}[\hat{\boldsymbol{a}}_i] \quad \leq \quad \frac{\|\boldsymbol{a}\|_2^2}{w}}$$

$$
\begin{aligned}
& \Pr\left[\left|\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right| > \varepsilon \|\boldsymbol{a}\|_2\right] \\
\leq \quad & \frac{\mathrm{Var}[\hat{\boldsymbol{a}}_i]}{\left(\varepsilon \|\boldsymbol{a}\|_2\right)^2} \\
\leq \quad & \frac{\|\boldsymbol{a}\|_2^2}{w} \cdot \frac{1}{\left(\varepsilon \|\boldsymbol{a}\|_2\right)^2} \\
= \quad & \frac{1}{w\varepsilon^2}
\end{aligned}
$$

Goal: Make an estimator $\hat{\boldsymbol{a}}$ for some quantity $\boldsymbol{a}$ where

With probability at least $1 - \delta$, ***Probably***

$|\hat{\boldsymbol{a}} - \boldsymbol{a}| \le \varepsilon \cdot \mathit{size}(\mathit{input})$ ***Approximately Correct***

for some measure of input size.

$$\Pr\left[\left|\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right| > \varepsilon \left\|\boldsymbol{a}\right\|_2\right] \le \frac{1}{w\varepsilon^2}$$

Pick $w = 4 \cdot \varepsilon^{-2}$. Then

$$\Pr\left[\left|\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right| > \varepsilon \left\|\boldsymbol{a}\right\|_2\right] \le \frac{1}{4}.$$

We now have a single estimator with a not-so-great chance of giving a good estimate.

How do we fix this?

How to Build an Estimator

	Count-Min Sketch	***Count Sketch***
Step One: Build a Simple Estimator	Hash items to counters; add +1 when item seen.	Hash items to counters; add ±1 when item seen.
Step Two: Compute Expected Value of Estimator	Sum of indicators; 2-independent hashes have low collision rate.	2-independence breaks up products; ±1 variables have zero expected value.
Step Three: Apply Concentration Inequality	One-sided error; use expected value and Markov's inequality.	Two-sided error; compute variance and use Chebyshev's inequality.
Step Four: Replicate to Boost Confidence	Take min; only fails if all estimates are bad.	

How to Build an Estimator

	Count-Min Sketch	***Count Sketch***
Step One: Build a Simple Estimator	Hash items to counters; add +1 when item seen.	Hash items to counters; add ±1 when item seen.
Step Two: Compute Expected Value of Estimator	Sum of indicators; 2-independent hashes have low collision rate.	2-independence breaks up products; ±1 variables have zero expected value.
Step Three: Apply Concentration Inequality	One-sided error; use expected value and Markov's inequality.	Two-sided error; compute variance and use Chebyshev's inequality.
Step Four: Replicate to Boost Confidence	Take min; only fails if all estimates are bad.	

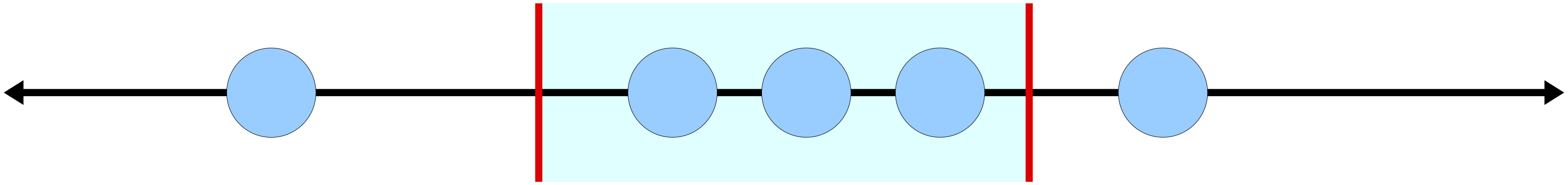
Running in Parallel

- Let's suppose that we run ***d*** independent copies of this data structure. Each has its own independently randomly chosen hash function.
- To ***increment***(x) in the overall structure, we call ***increment***(x) on each of the underlying data structures.
- The probability that at least one of them provides a good estimate is high.
- ***Question:*** How do you know which one?

Estimator 1:	*Estimator 2:*	*Estimator 3:*	*Estimator 4:*	*Estimator 5:*
137	271	166	103	261

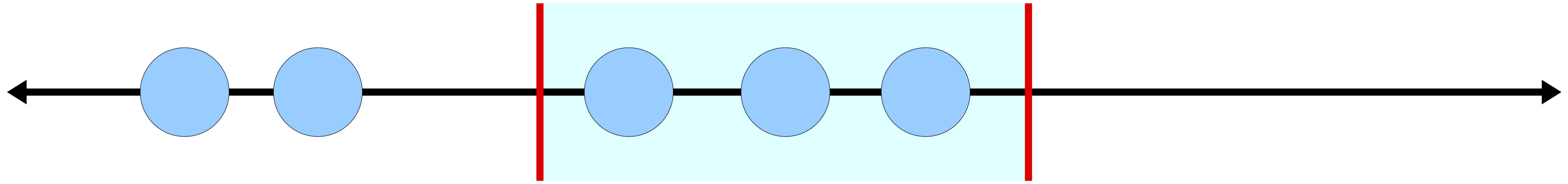
Working with the Median

- ***Claim:*** If we output the median estimate given by the data structures, we have high probability of giving the right answer.
- ***Intuition:*** The only way that the median isn't in the "good" area is if ***at least half*** the estimates are in the "bad" area.
- Each individual data structure has a "reasonable" chance to be good, so this is very unlikely.



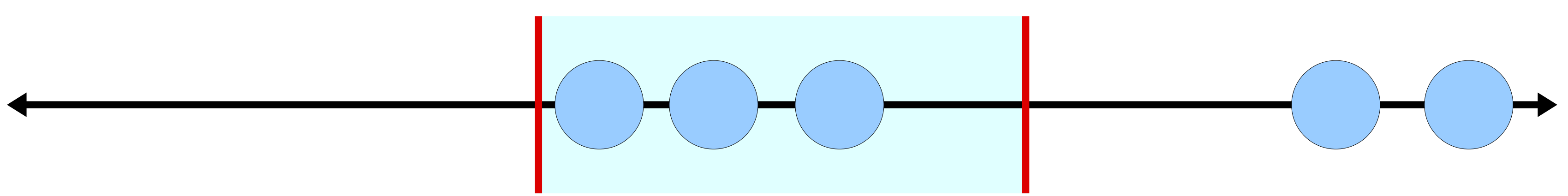
Working with the Median

- ***Claim:*** If we output the median estimate given by the data structures, we have high probability of giving the right answer.
- ***Intuition:*** The only way that the median isn't in the "good" area is if ***at least half*** the estimates are in the "bad" area.
- Each individual data structure has a "reasonable" chance to be good, so this is very unlikely.



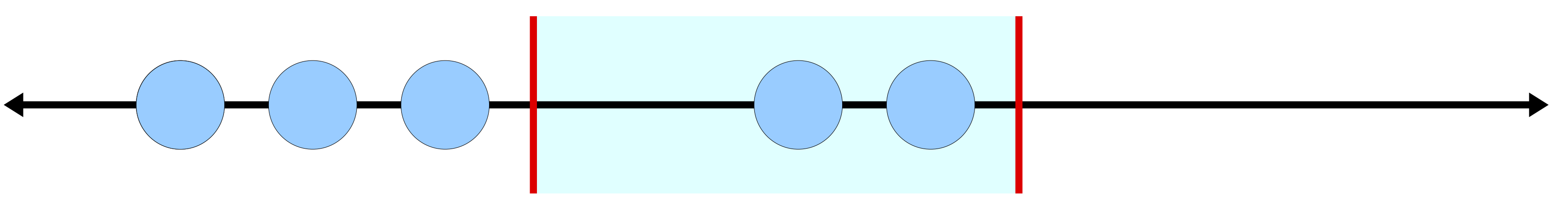
Working with the Median

- ***Claim:*** If we output the median estimate given by the data structures, we have high probability of giving the right answer.
- ***Intuition:*** The only way that the median isn't in the "good" area is if ***at least half*** the estimates are in the "bad" area.
- Each individual data structure has a "reasonable" chance to be good, so this is very unlikely.



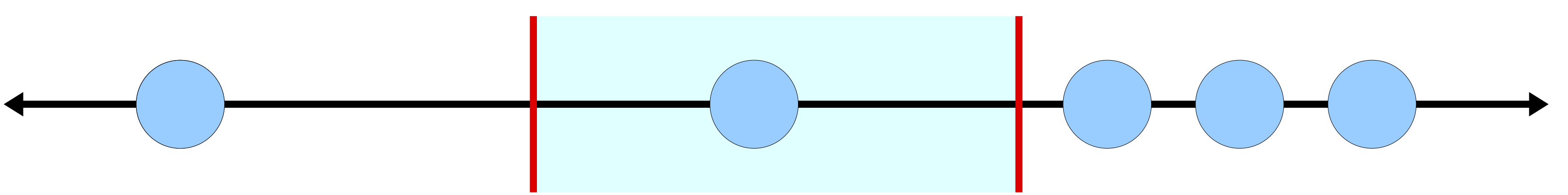
Working with the Median

- ***Claim:*** If we output the median estimate given by the data structures, we have high probability of giving the right answer.
- ***Intuition:*** The only way that the median isn't in the "good" area is if ***at least half*** the estimates are in the "bad" area.
- Each individual data structure has a "reasonable" chance to be good, so this is very unlikely.



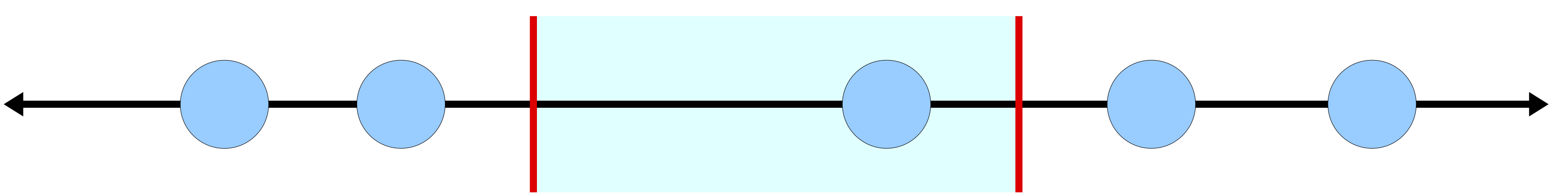
Working with the Median

- ***Claim:*** If we output the median estimate given by the data structures, we have high probability of giving the right answer.
- ***Intuition:*** The only way that the median isn't in the "good" area is if ***at least half*** the estimates are in the "bad" area.
- Each individual data structure has a "reasonable" chance to be good, so this is very unlikely.



Working with the Median

- ***Claim:*** If we output the median estimate given by the data structures, we have high probability of giving the right answer.
- ***Intuition:*** The only way that the median isn't in the "good" area is if ***at least half*** the estimates are in the "bad" area.
- Each individual data structure has a "reasonable" chance to be good, so this is very unlikely.



The Setup

- Let D denote a random variable equal to the number of data structures that produce an answer *not* within $\varepsilon||\boldsymbol{a}||_2$ of the true answer.
- Since each independent data structure has failure probability at most ¼, we can upper-bound D with a Binom(d, ¼) variable.
- We want to know $\Pr[D > d / 2]$.
- How can we determine this?

Chernoff Bounds

- The ***Chernoff bound*** says that if $X \sim \text{Binom}(n, p)$ and $p < 1/2$, then

$$\Pr[X \geq n/2] < \boxed{e^{-n \cdot z(p)}}$$

where $z(p) = (½ – p)^2 / 2p$.

Intuition: For any fixed value of p, this quantity decays exponentially quickly as a function of n. It's extremely unlikely that more than half our estimates will be bad.

Chernoff Bounds

- The ***Chernoff bound*** says that if $X \sim \text{Binom}(n, p)$ and $p < 1/2$, then

$$\Pr[X \geq n/2] < e^{-n \cdot z(p)}$$

where $z(p) = (½ – p)^2 / 2p$.

- In our case, $D \sim \text{Binom}(d, ¼)$, so we know that

$$\Pr[D \geq \frac{d}{2}] \leq e^{-n \cdot z(1/4)} = e^{-d/8}$$

- Therefore, choosing $\boldsymbol{d = 8 \log \delta^{-1}}$ ensures that

$$\Pr[|\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i| > \varepsilon \|\boldsymbol{a}\|_2] \leq \Pr[D \geq \frac{d}{2}] \leq \delta$$

How to Build an Estimator

	Count-Min Sketch	***Count Sketch***
Step One: Build a Simple Estimator	Hash items to counters; add +1 when item seen.	Hash items to counters; add ±1 when item seen.
Step Two: Compute Expected Value of Estimator	Sum of indicators; 2-independent hashes have low collision rate.	2-independence breaks up products; ±1 variables have zero expected value.
Step Three: Apply Concentration Inequality	One-sided error; use expected value and Markov's inequality.	Two-sided error; compute variance and use Chebyshev's inequality.
Step Four: Replicate to Boost Confidence	Take min; only fails if all estimates are bad.	Take median; only can fail if half of estimates are wrong; use Chernoff.

The Count Sketch

$w = \lceil 4 \cdot \varepsilon^{-1} \rceil$

h_1	s_1
h_2	s_2
h_3	s_3
…	…
h_d	s_d

31	41	-59	-26	…	58
27	-18	28	-18	…	-45
16	-18	-3	39	…	-75
…					
69	-31	47	-18	…	59

$d = \lceil 8 \ln \delta^{-1} \rceil$

Sampled uniformly and independently from 2-independent families of hash functions

The Count Sketch

$w = \lceil 4 \cdot \varepsilon^{-1} \rceil$

h_1	s_1
h_2	s_2
h_3	s_3
…	…
h_d	s_d

31	41	-59	-26	…	58
27	-18	28	-18	…	-45
16	-18	-3	39	…	-75
…					
69	-31	47	-18	…	59

$d = \lceil 8 \ln \delta^{-1} \rceil$

```
increment(x):
   for i = 1 … d:
      count[i][hi(x)] += si(x)
```

The Count Sketch

$w = \lceil 4 \cdot \varepsilon^{-1} \rceil$

h_1	s_1
h_2	s_2
h_3	s_3
…	…
h_d	s_d

31	**41**	-59	-26	…	58
27	-18	28	**-18**	…	-45
16	-18	-3	**39**	…	-75
			…		
69	-31	47	-18	…	**59**

$d = \lceil 8 \ln \delta^{-1} \rceil$

```
increment(x):
   for i = 1 … d:
      count[i][hᵢ(x)] += sᵢ(x)
```

The Count Sketch

$w = \lceil 4 \cdot \varepsilon^{-1} \rceil$

h_1	s_1
h_2	s_2
h_3	s_3
…	…
h_d	s_d

31	**40**	-59	-26	…	58
27	-18	28	**-19**	…	-45
16	-18	-3	**40**	…	-75
		…			
69	-31	47	-18	…	**58**

$d = \lceil 8 \ln \delta^{-1} \rceil$

```
increment(x):
   for i = 1 … d:
      count[i][hᵢ(x)] += sᵢ(x)
```

The Count Sketch

$w = \lceil 4 \cdot \varepsilon^{-1} \rceil$

$d = \lceil 8 \ln \delta^{-1} \rceil$

h_1	s_1
h_2	s_2
h_3	s_3
…	…
h_d	s_d

31	40	-59	-26	…	58
27	-18	28	-19	…	-45
16	-18	-3	40	…	-75
…					
69	-31	47	-18	…	58

```
increment(x):
   for i = 1 … d:
      count[i][hᵢ(x)] += sᵢ(x)
```

The Count Sketch

$w = \lceil 4 \cdot \varepsilon^{-1} \rceil$

$d = \lceil 8 \ln \delta^{-1} \rceil$

h_1	s_1
h_2	s_2
h_3	s_3
…	…
h_d	s_d

31	40	-59	-26	…	58
27	-18	28	-19	…	-45
16	-18	-3	40	…	-75
…					
69	-31	47	-18	…	58

```
increment(x):
   for i = 1 … d:
      count[i][hᵢ(x)] += sᵢ(x)
```

```
estimate(x):
   options = []
   for i = 1 … d:
      options += count[i][hᵢ(x)] * sᵢ(x)
   return medianOf(options)
```

The Count Sketch

$w = \lceil 4 \cdot \varepsilon^{-1} \rceil$

h_1	s_1	31	40	-59	-26	…	58
h_2	s_2	27	-18	28	-19	…	-45
h_3	s_3	16	-18	-3	40	…	-75
…	…			…			
h_d	s_d	69	-31	47	-18	…	58

$d = \lceil 8 \ln \delta^{-1} \rceil$

```
increment(x):
   for i = 1 … d:
      count[i][h_i(x)] += s_i(x)
```

```
estimate(x):
   options = []
   for i = 1 … d:
      options += count[i][h_i(x)] * s_i(x)
   return medianOf(options)
```

The Final Analysis

- Here's a comparison of these two structures.
- ***Question to ponder:*** When is a count-min sketch better than a count sketch, and vice-versa?

Count-Min Sketch

Space: $\Theta(\varepsilon^{-1} \log \delta^{-1})$

increment: $\Theta(\log \delta^{-1})$

estimate: $\Theta(\log \delta^{-1})$

Accuracy: within $\varepsilon||\boldsymbol{a}||_1$.

Count Sketch

Space: $\Theta(\varepsilon^{-2} \log \delta^{-1})$

increment: $\Theta(\log \delta^{-1})$

estimate: $\Theta(\log \delta^{-1})$

Accuracy: within $\varepsilon||\boldsymbol{a}||_2$

Bloom Filters

Exact Membership Queries

- Suppose you're in a memory-constrained environment where every bit of memory counts, and you want to store a set of n items drawn from some universe U.
- Examples:
  - You're working on an embedded device with some maximum amount of working RAM.
  - You're working with large n (say, $n = 10^9$) on a modern machine.
  - You're building a consumer application like a web browser and don't want to hog all system resources.
- ***Question:*** How many bits of memory are needed to do this?

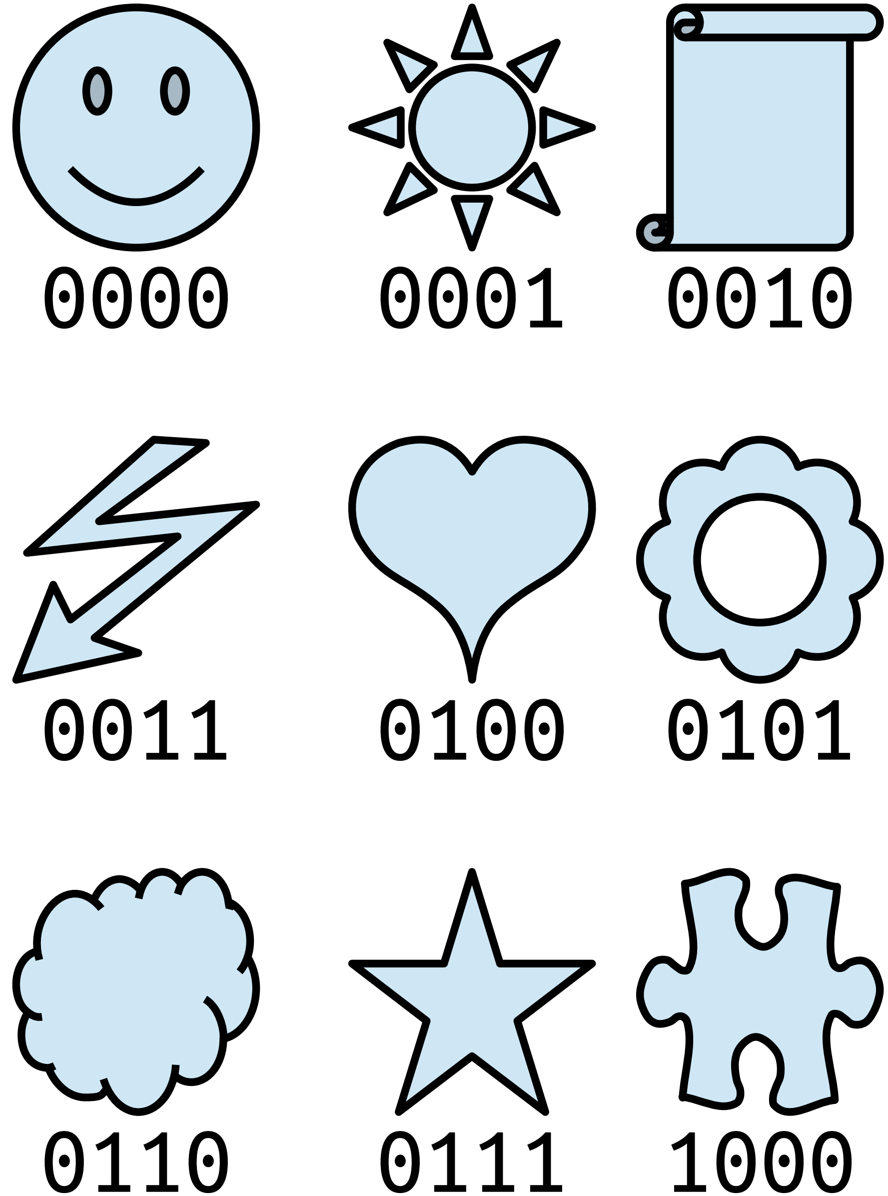
A Quick Detour

Goal: Design a simple data structure that can hold a single one of the objects shown to the right.

What is the minimum number of *bits* (not *words*) required for this data structure in the worst case?

We can get away with four bits by numbering each item and just storing the number.

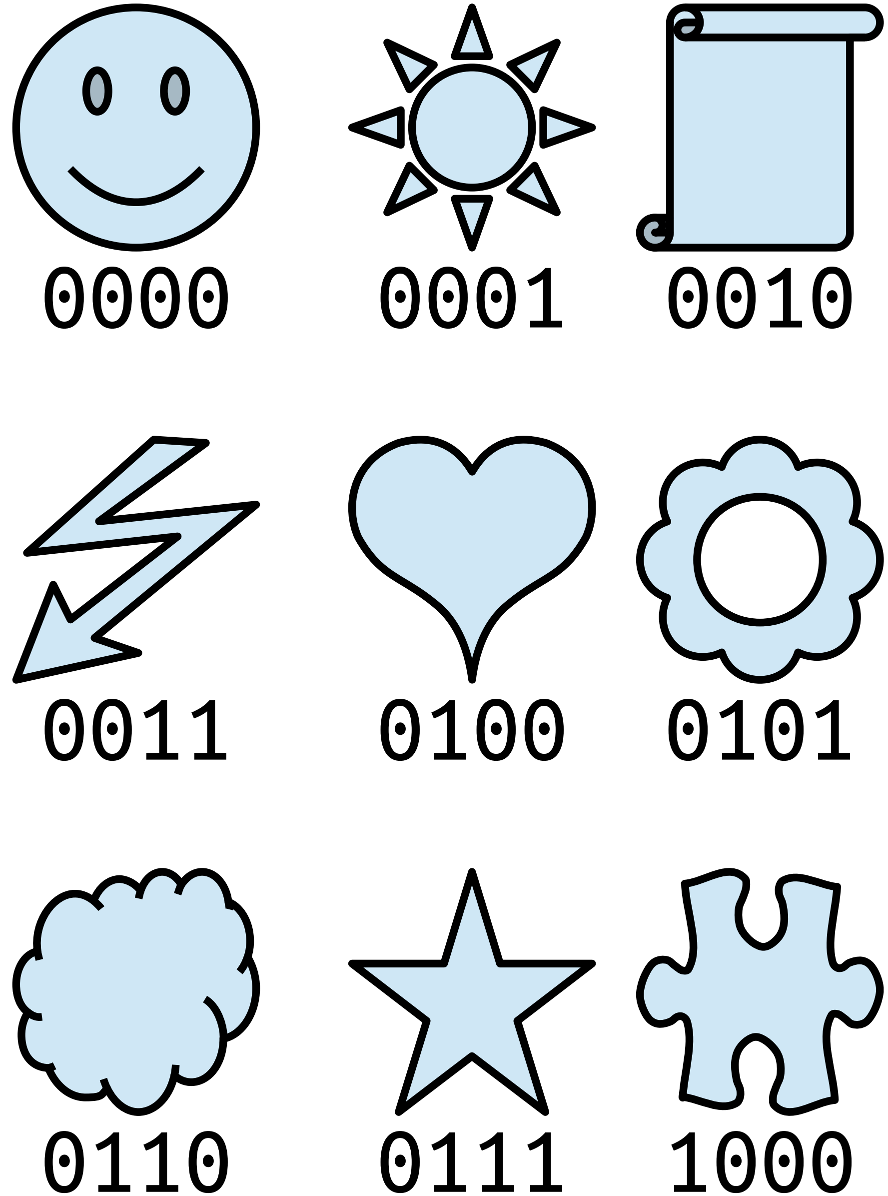
Question: Can we do better?



Goal: Design a simple data structure that can hold a single one of the objects shown to the right.

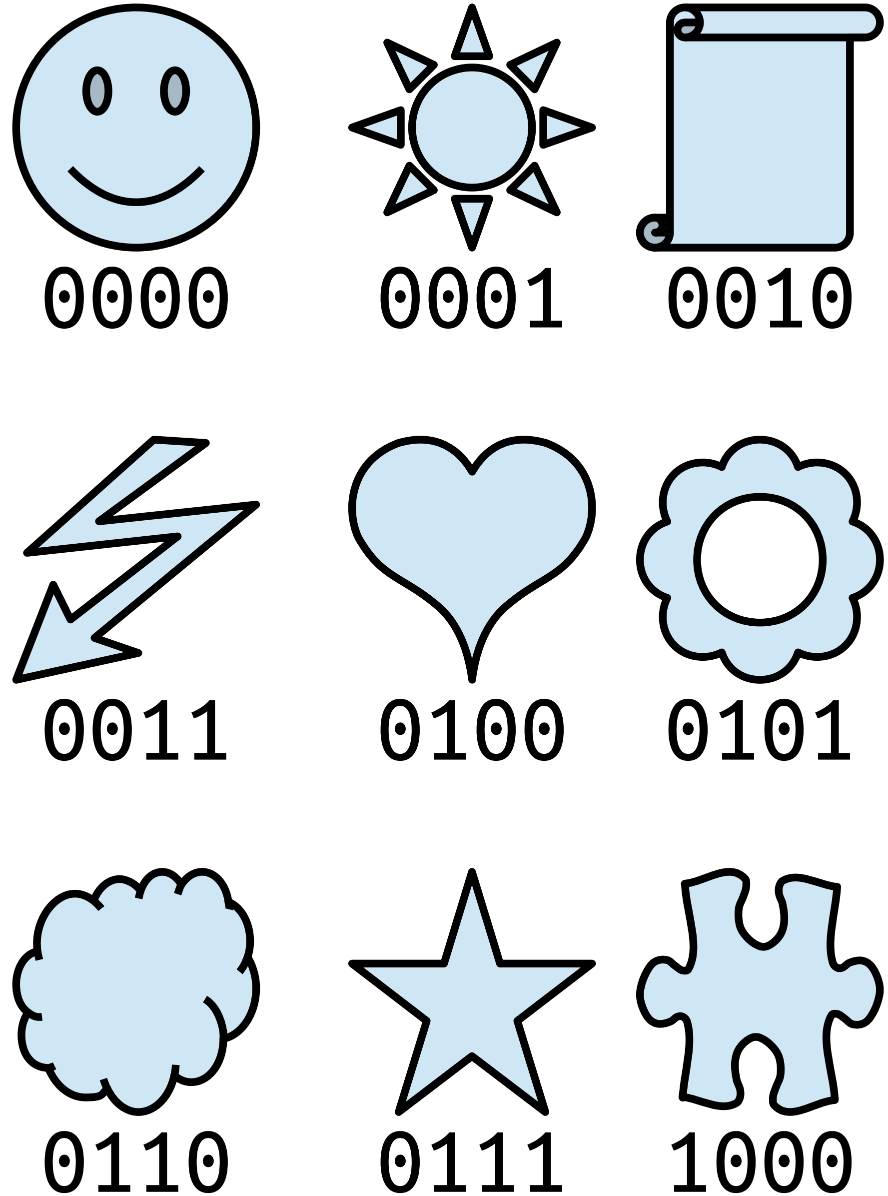
Claim: Every data structure for this problem must use at least four bits of memory in the worst case.

Proof: If we always use three or fewer bits, there are at most $2^3 = 8$ combinations of those bits, not enough to uniquely identify one of the nine different items.



Theorem: A data structure that stores one object out of a set of k possibilities must use at least $\lg k$ bits in the worst case.

Proof: Using fewer than $\lg k$ bits means there are fewer than $2^{\lg k} = k$ possible combinations of those bits, not enough to uniquely identify each item out of the set.



Question: How much memory is needed to solve the exact membership query problem?

Suppose we want to store a set $S \subseteq U$ of size n. How many bits of memory do we need?

Number of n-element subsets of universe U:

$$\binom{|U|}{n}$$

Bits needed:

$\Omega(n \lg |U| - n \lg n)$

$$
\begin{aligned}
& \lg \binom{|U|}{n} \\
= \; & \lg \left(\frac{|U|\,!}{n\,!\,(|U|-n)\,!} \right) \\
\geq \; & \lg \left(\frac{(|U|-n)^n}{n^n} \right) \\
= \; & n \lg \left(\frac{|U|-n}{n} \right) \\
= \; & n \lg \left(\frac{|U|}{n} - 1 \right) \\
\geq \; & n \lg \left(\frac{|U|}{n} \right) - n \\
\geq \; & n \lg |U| - n \lg n - n \\
= \; & \boldsymbol{\Omega\left(n \lg |U| - n \lg n \right)}
\end{aligned}
$$

Bitten by Bits

- Assuming $|U| \gg n$, we need $\boldsymbol{\Omega(n \log |U|)}$ bits to encode a solution to the exact membership query problem.
- If we're resource-constrained, this might be way too many bits for us to fit things in memory.
  - Think $n = 10^8$ and U is the set of all possible URLs or human genomes.
- Can we do better?

Approximate Membership Queries

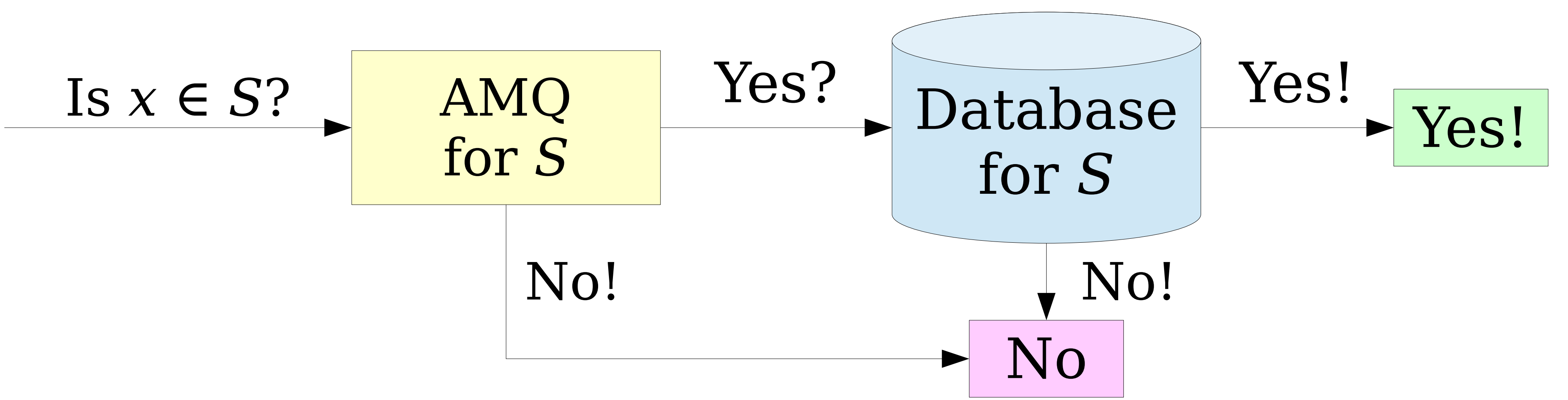
- The ***approximate membership query*** problem is the following:

  Maintain a set S in a way that gives approximate answers to queries of the form "is x ∈ S?"

- Questions we need to answer:
  - How do you give an "approximate" answer to the question "is $x \in S$?"
  - Does this relaxation let us save memory?
- We'll address each of these in turn.

Our Model

- ***Goal:*** Design our data structures to allow for false positives but not false negatives.
- That is:
  - if $x \in S$, we always return true, but
  - if $x \notin S$, we have a small probability of returning true.
- This is often a good idea in practice.



Our Model

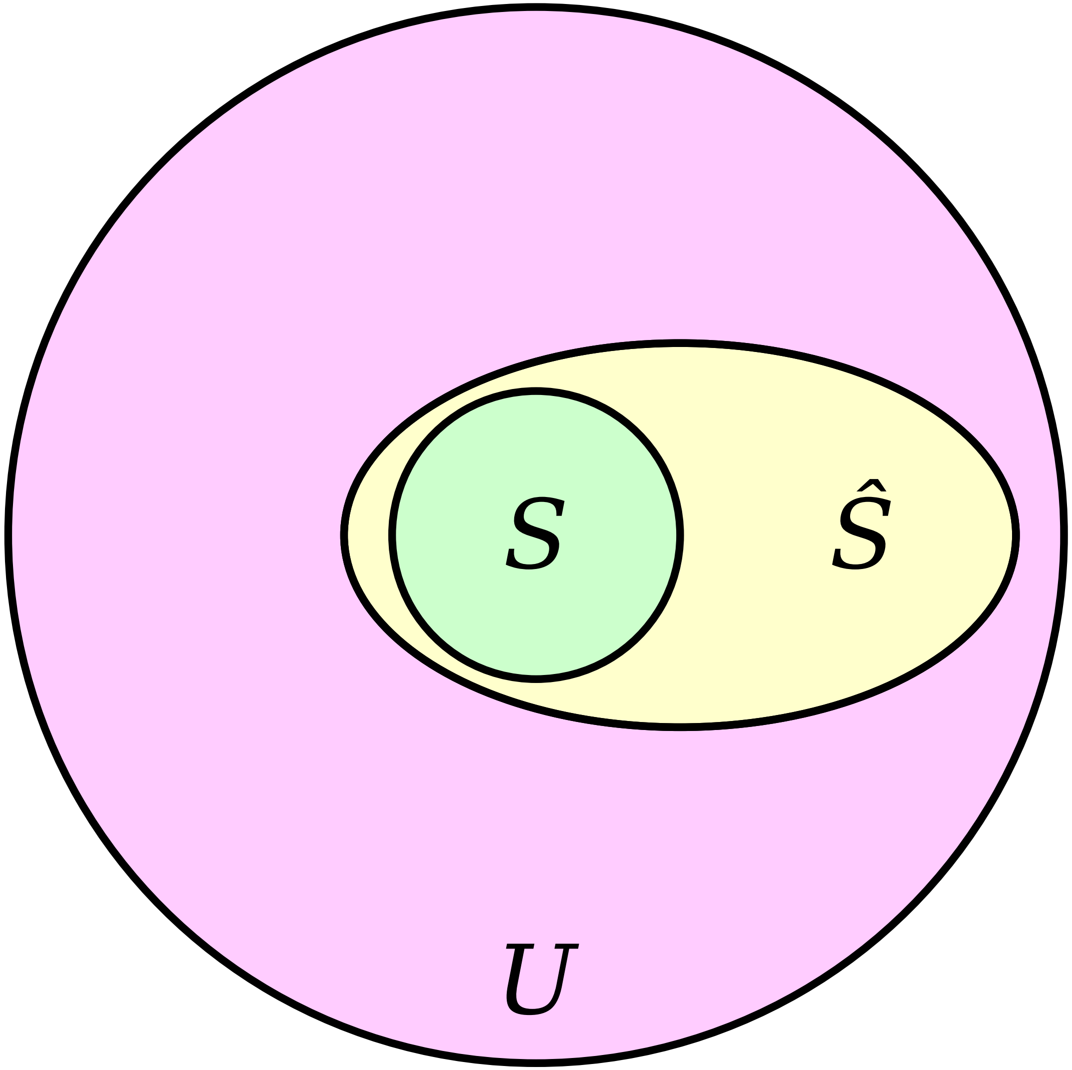
- Let’s assume we have a tunable accuracy parameter $\varepsilon \in (0, 1)$.
- ***Goal:*** Design our data structure so that
  - if $x \in S$, we always return true;
  - if $x \notin S$, we return false with probability at least $1 - \varepsilon$; and
  - the amount of space we need depends only on n and ε, not on the size of the universe.
- Is this even possible?

Suppose we're storing an n-element set S with error rate ε.

How much memory is needed to solve the approximate membership query problem?

Suppose we're storing an n-element set S with error rate ε.

Intuition: An AMQ structure stores a set $\hat{S}$: S plus approximately $\varepsilon|U|$ extra elements due to the error rate.

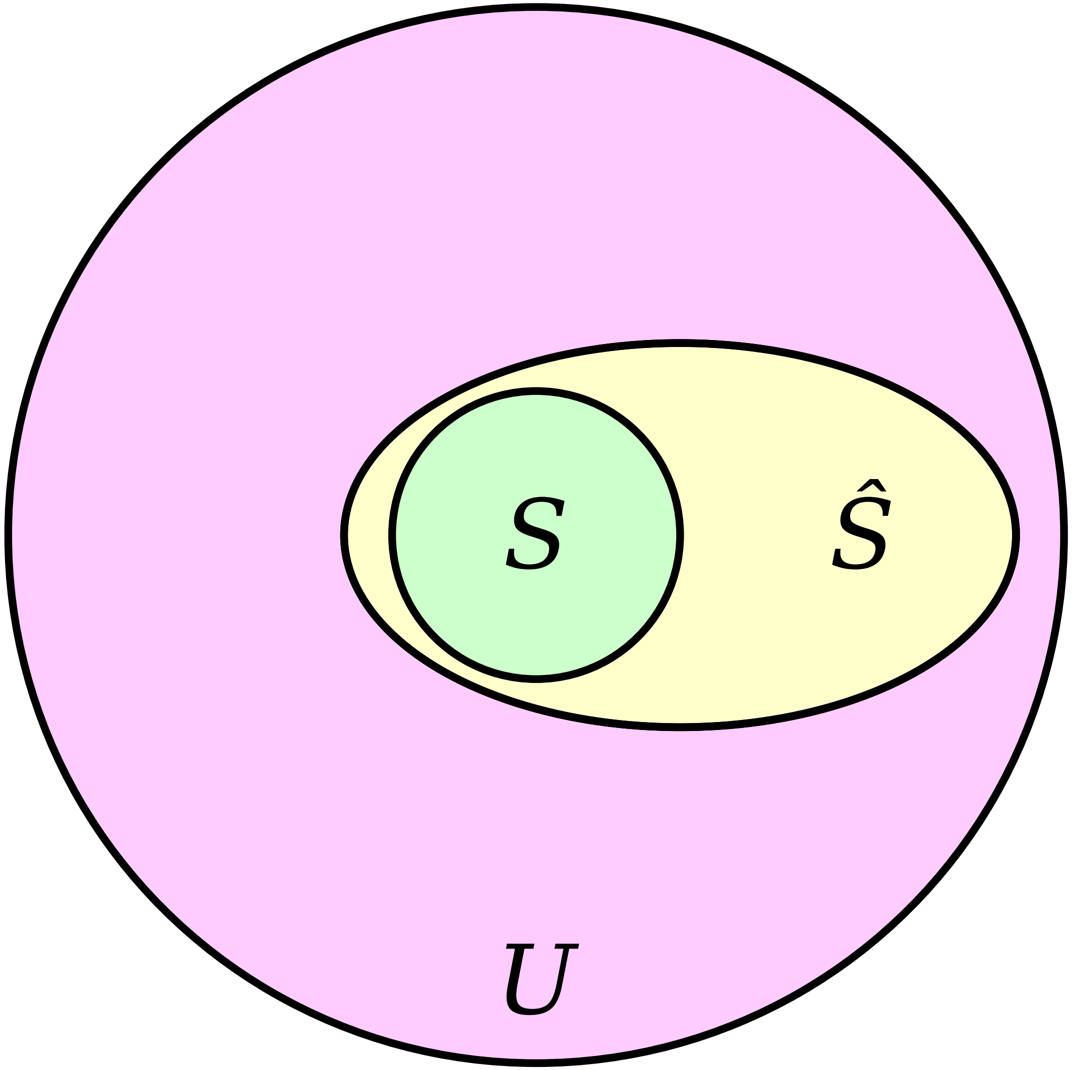


How much memory is needed to solve the approximate membership query problem?

Suppose we're storing an n-element set S with error rate ε.

Intuition: An AMQ structure stores a set $\hat{S}$: S plus approximately $\varepsilon|U|$ extra elements due to the error rate.

Importantly, we don't care *which* $\varepsilon|U|$ elements those are.

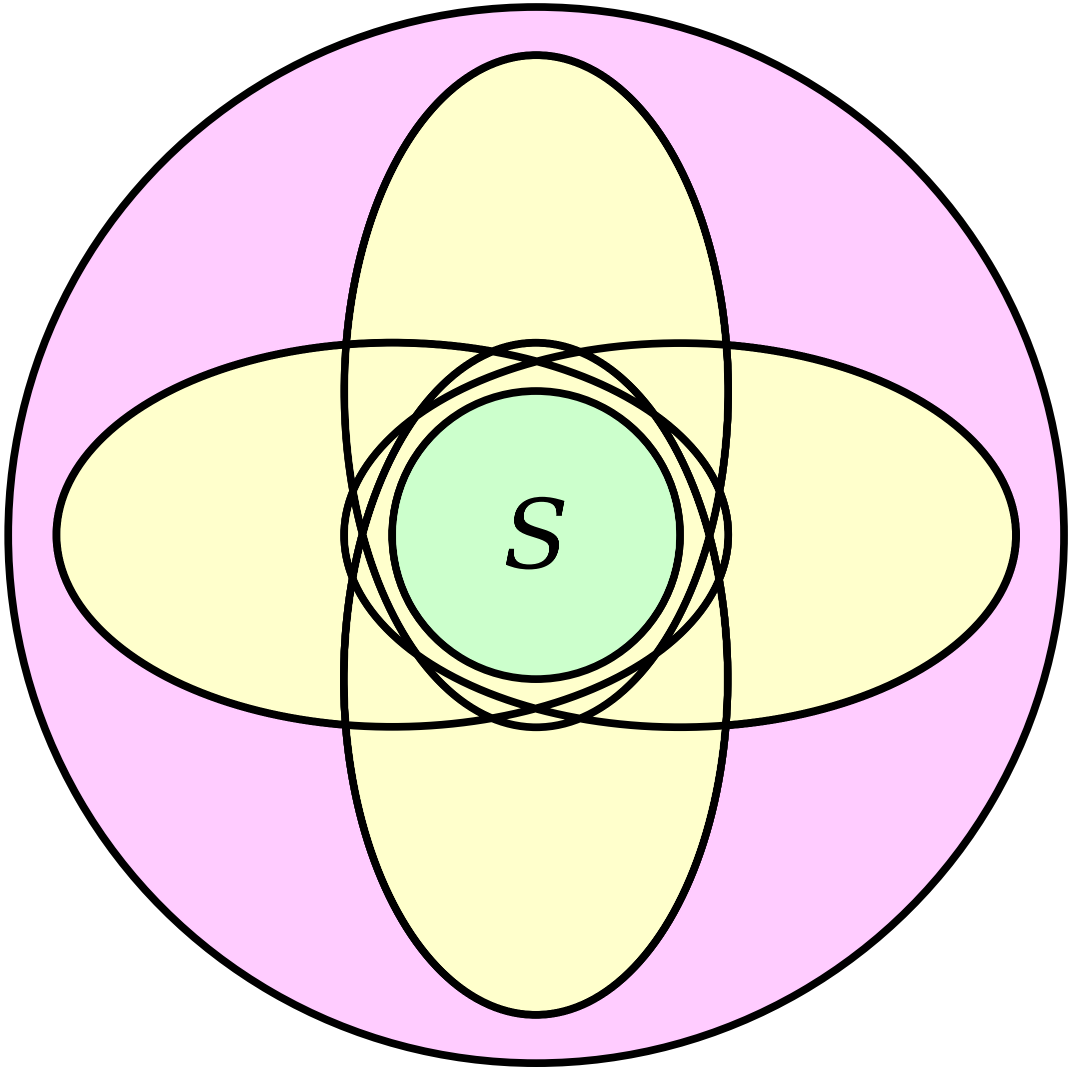


How much memory is needed to solve the approximate membership query problem?

Suppose we're storing an n-element set S with error rate ε.

Intuition: An AMQ structure stores a set $\hat{S}$: S plus approximately $\varepsilon|U|$ extra elements due to the error rate.

Importantly, we don't care *which* $\varepsilon|U|$ elements those are.

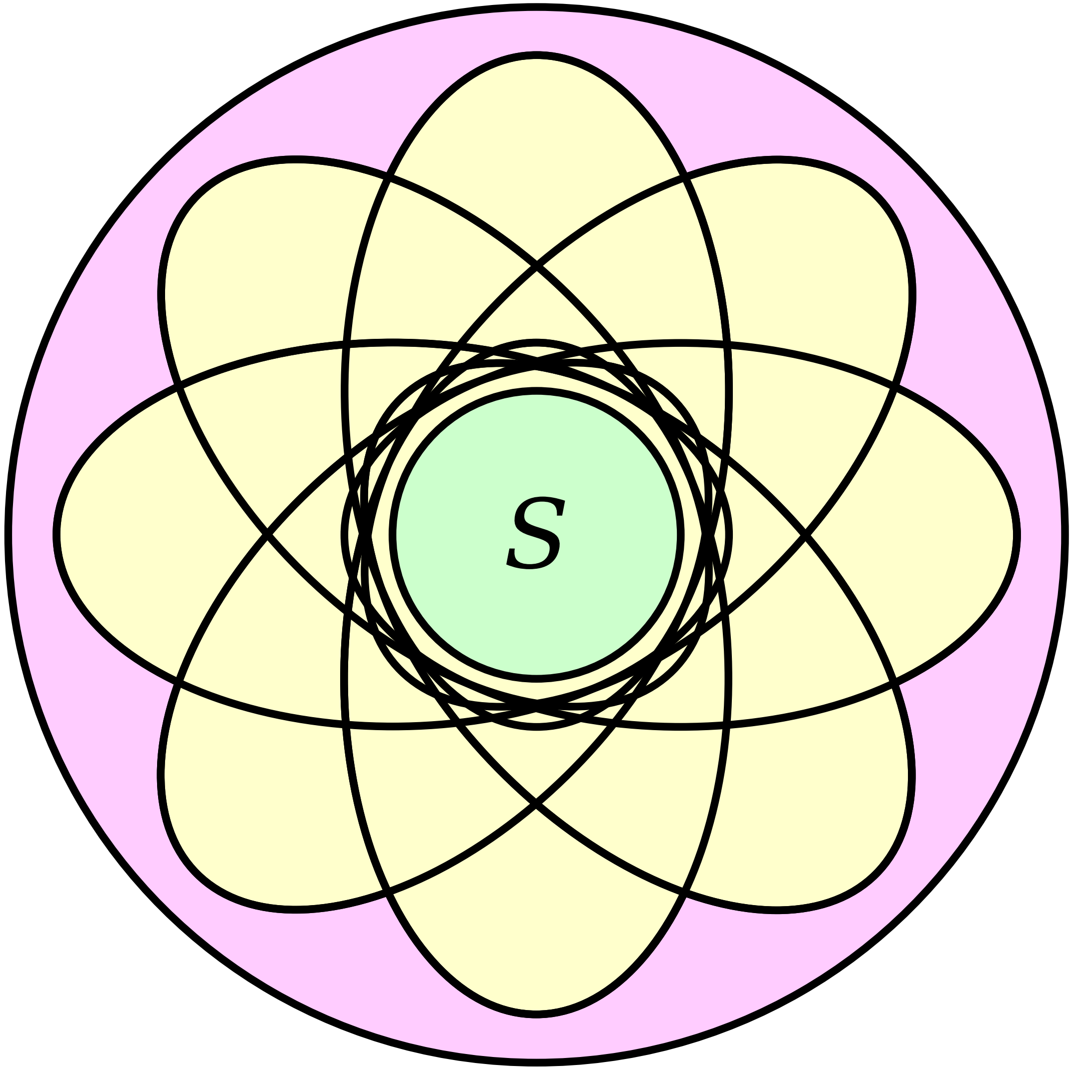


How much memory is needed to solve the approximate membership query problem?

Suppose we're storing an n-element set S with error rate ε.

Intuition: An AMQ structure stores a set $\hat{S}$: S plus approximately $\varepsilon|U|$ extra elements due to the error rate.

Importantly, we don't care *which* $\varepsilon|U|$ elements those are.



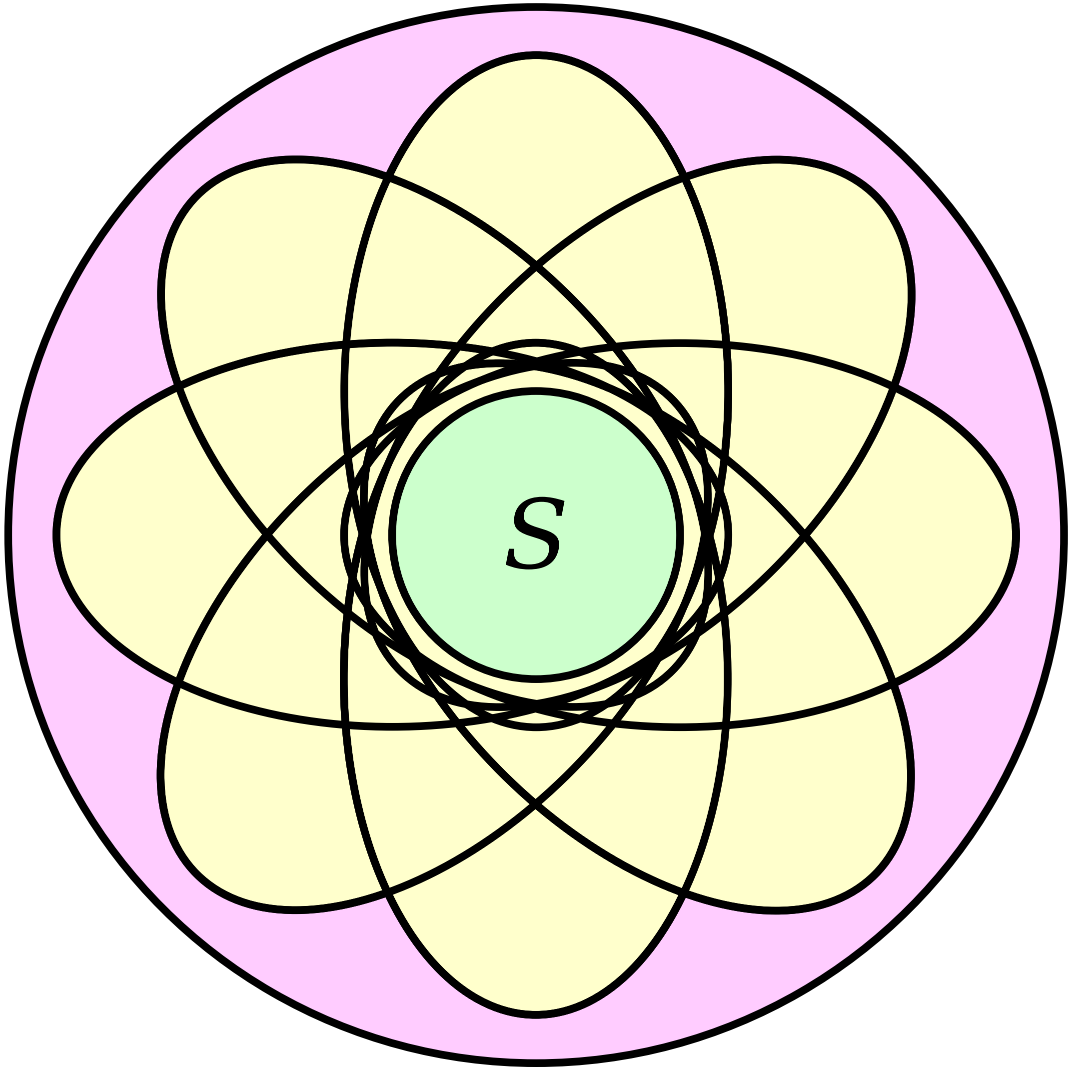
How much memory is needed to solve the approximate membership query problem?

Suppose we're storing an n-element set S with error rate ε.

Intuition: An AMQ structure stores a set $\hat{S}$: S plus approximately $\varepsilon|U|$ extra elements due to the error rate.

Importantly, we don't care *which* $\varepsilon|U|$ elements those are.

How does that affect our lower bound?



How much memory is needed to solve the approximate membership query problem?

How much memory is needed to solve
the approximate membership query problem?

Idea: We can use an AMQ, plus some more bits, to describe a set S.

How much memory is needed to solve the approximate membership query problem?

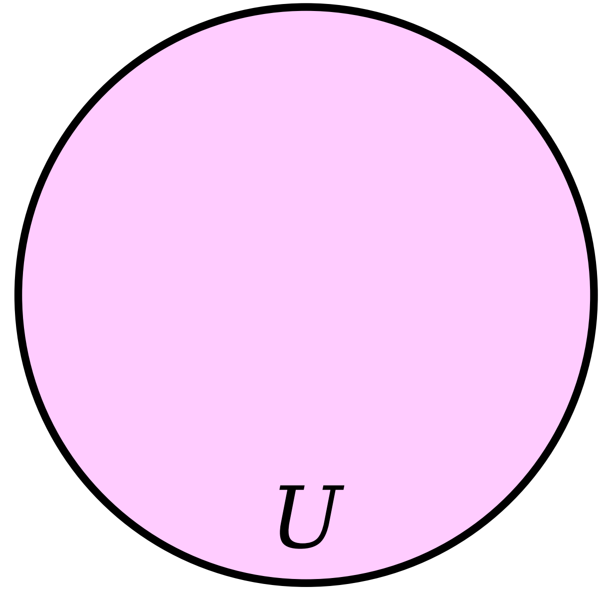
Idea: We can use an AMQ, plus some more bits, to describe a set S.

With b bits, write down an AMQ structure. This describes a set $\hat{S} \subseteq U$ of size around $\varepsilon|U|$.

How much memory is needed to solve the approximate membership query problem?

Idea: We can use an AMQ, plus some more bits, to describe a set S.

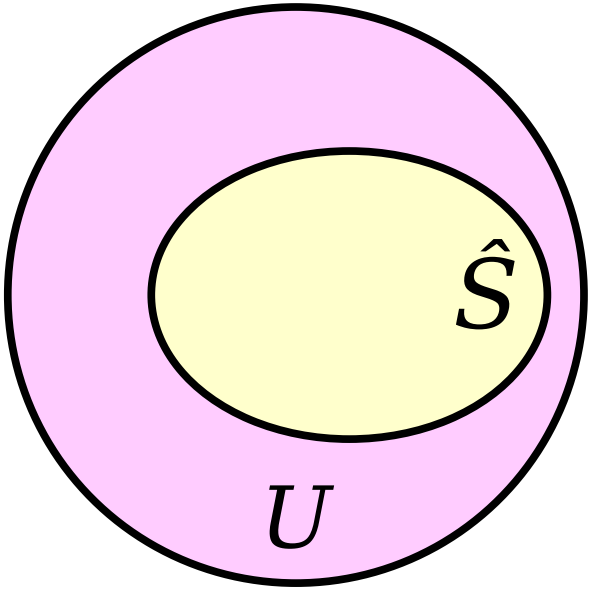
With b bits, write down an AMQ structure. This describes a set $\hat{S} \subseteq U$ of size around $\varepsilon|U|$.



How much memory is needed to solve the approximate membership query problem?

Idea: We can use an AMQ, plus some more bits, to describe a set S.

With b bits, write down an AMQ structure. This describes a set $\hat{S} \subseteq U$ of size around $\varepsilon|U|$.

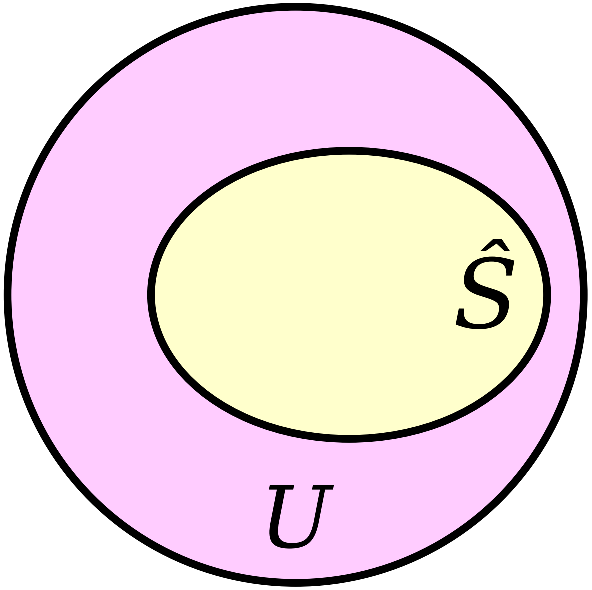


How much memory is needed to solve the approximate membership query problem?

Idea: We can use an AMQ, plus some more bits, to describe a set S.

With b bits, write down an AMQ structure. This describes a set $\hat{S} \subseteq U$ of size around $\varepsilon|U|$.

Write down some more bits to identify which n elements in $\hat{S}$ make up the set S.



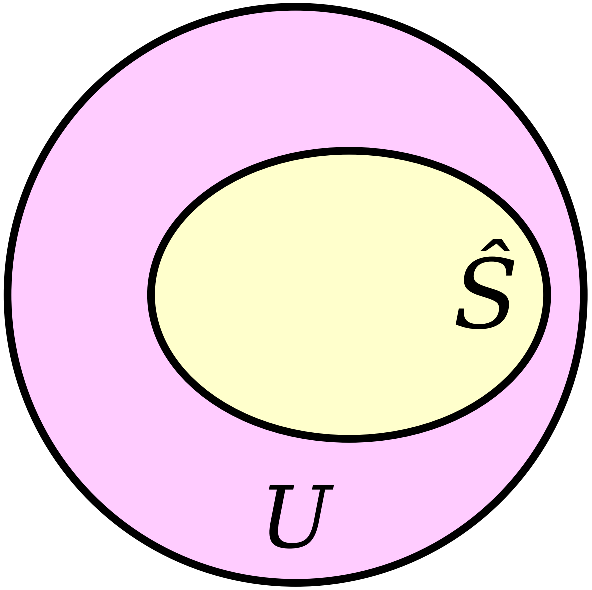
How much memory is needed to solve the approximate membership query problem?

Idea: We can use an AMQ, plus some more bits, to describe a set S.

With b bits, write down an AMQ structure. This describes a set $\hat{S} \subseteq U$ of size around $\varepsilon|U|$.

Write down some more bits to identify which n elements in $\hat{S}$ make up the set S. Bits needed:

$$\lg \binom{\varepsilon |U|}{n}$$



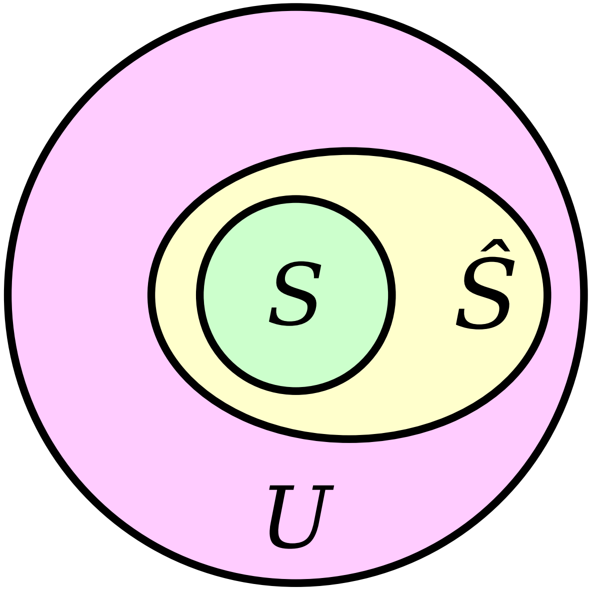
How much memory is needed to solve the approximate membership query problem?

Idea: We can use an AMQ, plus some more bits, to describe a set S.

With b bits, write down an AMQ structure. This describes a set $\hat{S} \subseteq U$ of size around $\varepsilon|U|$.

Write down some more bits to identify which n elements in $\hat{S}$ make up the set S. Bits needed:

$$\lg \binom{\varepsilon |U|}{n}$$



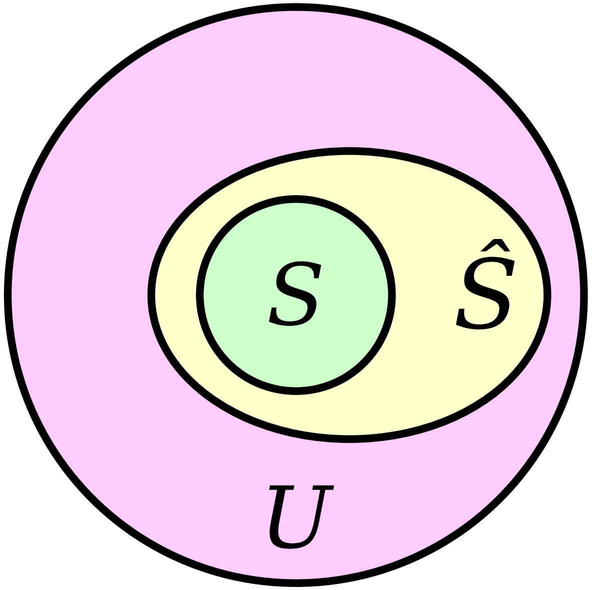
How much memory is needed to solve the approximate membership query problem?

Idea: We can use an AMQ, plus some more bits, to describe a set S.

With b bits, write down an AMQ structure. This describes a set $\hat{S} \subseteq U$ of size around $\varepsilon|U|$.

Write down some more bits to identify which n elements in $\hat{S}$ make up the set S. Bits needed:

$$\lg \binom{\varepsilon|U|}{n}$$



$$b + \lg \binom{\varepsilon|U|}{n}$$

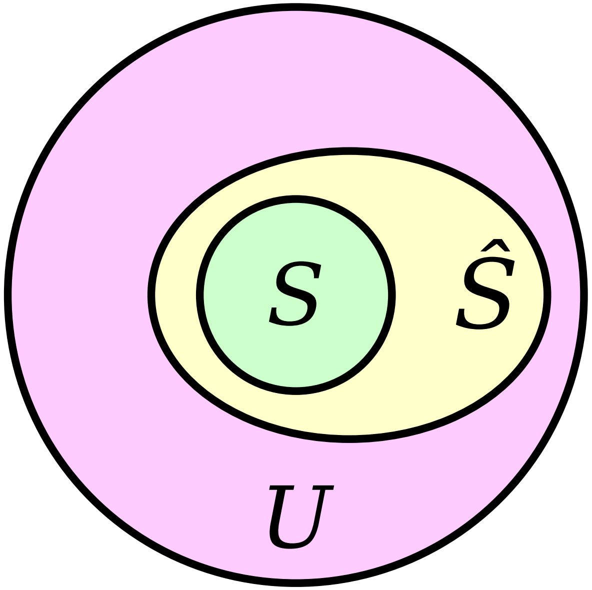
How much memory is needed to solve the approximate membership query problem?

Idea: We can use an AMQ, plus some more bits, to describe a set S.

With b bits, write down an AMQ structure. This describes a set $\hat{S} \subseteq U$ of size around $\varepsilon|U|$.

Write down some more bits to identify which n elements in $\hat{S}$ make up the set S. Bits needed:

$$\lg \binom{\varepsilon|U|}{n}$$



$$b + \lg \binom{\varepsilon|U|}{n}$$

(Number of bits needed to describe S.)

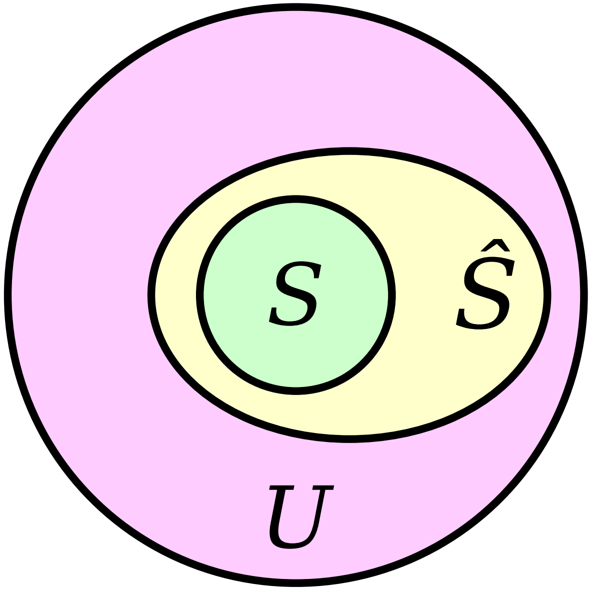
How much memory is needed to solve the approximate membership query problem?

Idea: We can use an AMQ, plus some more bits, to describe a set S.

With b bits, write down an AMQ structure. This describes a set $\hat{S} \subseteq U$ of size around $\varepsilon|U|$.

Write down some more bits to identify which n elements in $\hat{S}$ make up the set S. Bits needed:

$$\lg \binom{\varepsilon |U|}{n}$$



$$b + \lg \binom{\varepsilon |U|}{n} \geq \lg \binom{|U|}{n}$$

(Number of bits needed to describe S.)

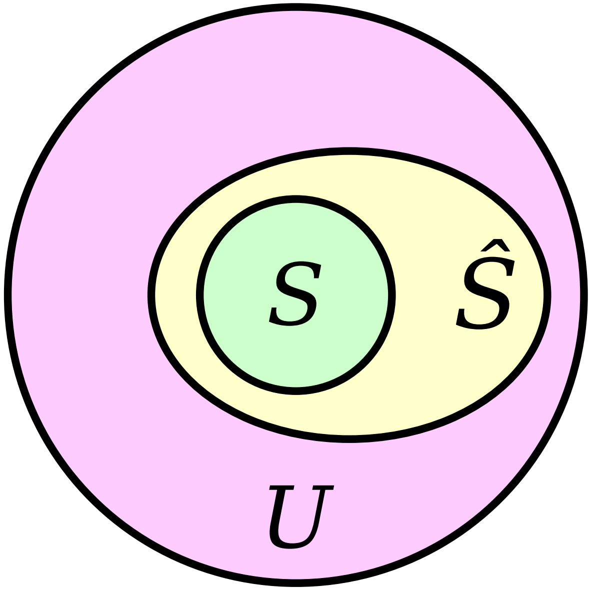
How much memory is needed to solve the approximate membership query problem?

Idea: We can use an AMQ, plus some more bits, to describe a set S.

With b bits, write down an AMQ structure. This describes a set $\hat{S} \subseteq U$ of size around $\varepsilon|U|$.

Write down some more bits to identify which n elements in $\hat{S}$ make up the set S. Bits needed:

$$\lg \binom{\varepsilon|U|}{n}$$



$$b + \lg \binom{\varepsilon|U|}{n} \geq \lg \binom{|U|}{n}$$

(Number of bits needed to describe S.)

(Lower bound on number of bits needed.)

How much memory is needed to solve the approximate membership query problem?

Theorem: Assuming $\varepsilon|U| \gg n$, any AMQ structure needs at least approximately $\boldsymbol{n \lg \varepsilon^{-1}}$ bits in the worst case.

This lower bound suggests we should aim for $\Theta(n \lg \varepsilon^{-1})$ bits of storage space.

How might we use them?

The math, if you're curious:

$$
\begin{aligned}
b &\geq \lg \left(\frac{\binom{|U|}{n}}{\binom{\varepsilon |U|}{n}} \right) \\
&\approx \lg \left(\frac{|U|^n / n!}{(\varepsilon |U|)^n / n!} \right) \\
&= \lg \varepsilon^{-n} \\
&= n \lg \varepsilon^{-1}
\end{aligned}
$$

How much memory is needed to solve the approximate membership query problem?

Bloom Filters

Idea 1: Adapt the "hash to a bucket" idea of the count-min and count sketches.

As an example, let's have

S = {103, 137, 166, 271, 314 }

`0000000000000000000000000000000000000000`

Number of bits: $\boldsymbol{cn \lg \varepsilon^{-1}}$

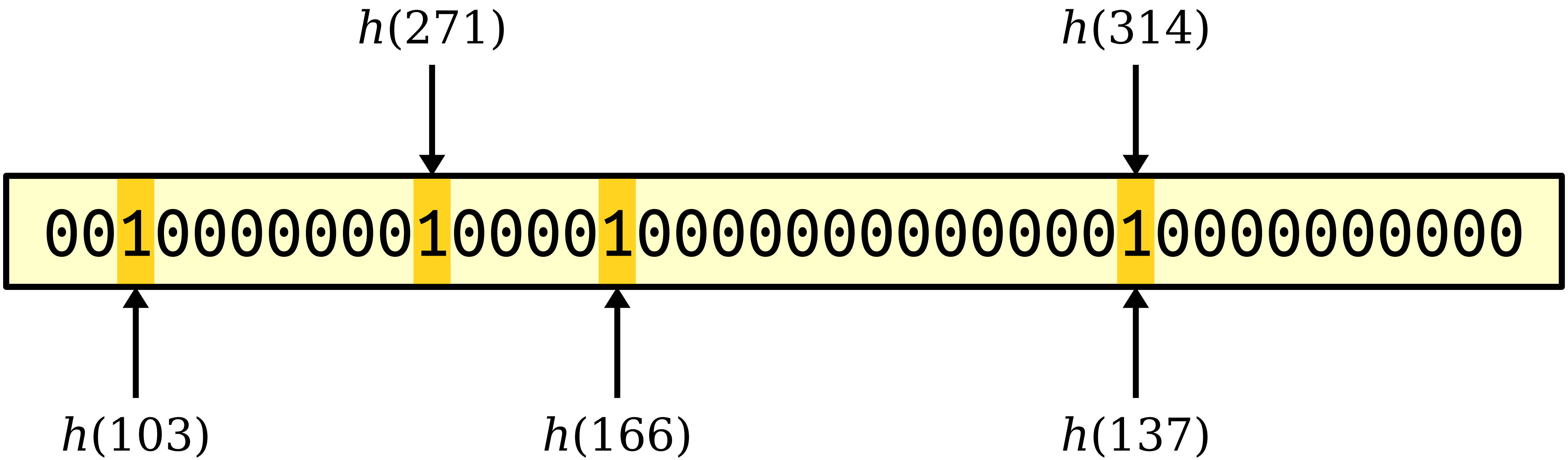
(We'll pick c later.)

How can we approximate a set in a small number of bits and with a low error rate?

Idea 1: Adapt the "hash to a bucket" idea of the count-min and count sketches.

As an example, let's have

$$S = \{103, 137, 166, 271, 314\}$$

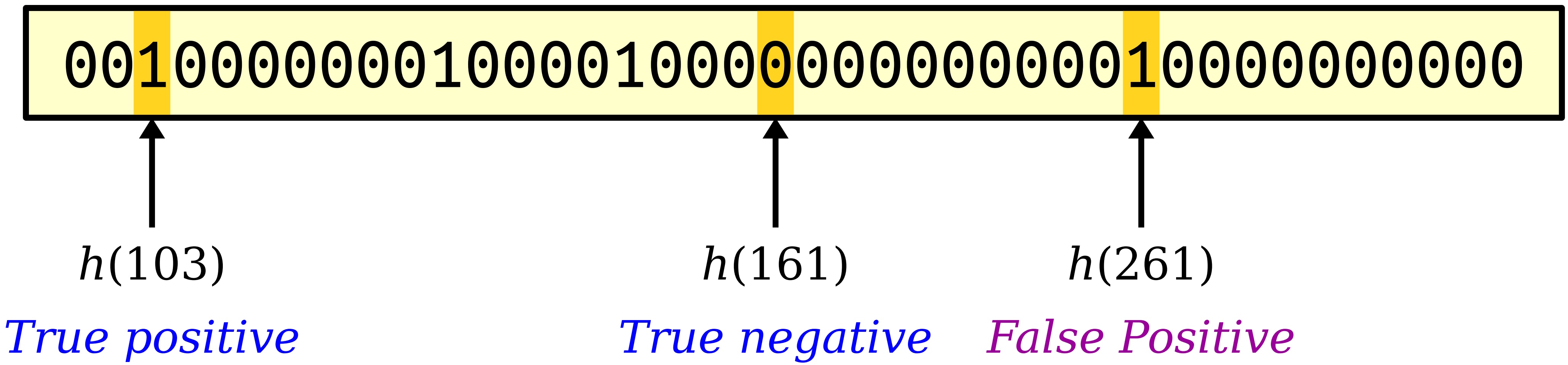


How can we approximate a set in a small number of bits and with a low error rate?

Idea 1: Adapt the "hash to a bucket" idea of the count-min and count sketches.

As an example, let's have

S = {103, 137, 166, 271, 314 }



How can we approximate a set in a small number of bits and with a low error rate?

Idea 1: Adapt the "hash to a bucket" idea of the count-min and count sketches.

Suppose we store a set of n elements in collection of $cn \lg \varepsilon^{-1}$ bits.

Question: What is our false positive rate?

`0010000000100001000000000000010000000000`

Intuition: At most n of our $cn \lg \varepsilon^{-1}$ bits will be 1. We only have false positives if we see a 1. So our false positive rate will be about $1 / c \lg \varepsilon^{-1}$. That's not great.

How can we approximate a set in a small number of bits and with a low error rate?

Idea 2: Adapt the "run in parallel" approach of the count-min sketch.

Make several copies of the previous data structure, each with a random hash function.

`0001000000000000000011000000010000000001`

`0010000000100001000000000000010000000000`

`0000010000000000010000001000001000000100`

Can we get the same accuracy while using fewer bits?

Idea 2: Adapt the "run in parallel" approach of the count-min sketch.

Question: Each copy provides its own estimate. Which one should we pick?

```
00010000000000000000 … 0001
00100000001000010000 … 0000
0000010000000000010000001000001000000100
```

This entry is 0, so it's not possible for this element to be in our set *S*.

Can we get the same accuracy while using fewer bits?

Idea 2: Adapt the "run in parallel" approach of the count-min sketch.

Question: Each copy provides its own estimate. Which one should we pick?

```
0001000000000000000011000000010000000001
```

```
0010000000100001000000000000010000000000
```

```
0000010000000000010000001000001000000100
```

Can we get the same accuracy while using fewer bits?

Idea 2: Adapt the "run in parallel" approach of the count-min sketch.

Question: Each copy provides its own estimate. Which one should we pick?

```
0001000000000000000011000000010000000001
```

```
0010000000100001000000000000010000000000
```

```
0000010000000000010000001000001000000100
```

Can we get the same accuracy while using fewer bits?

Idea 2: Adapt the “run in parallel” approach of the count-min sketch.

Question: Each copy provides its own estimate. Which one should we pick?

`0001000000000000000011000000010000000001`

`0010000000100001000000000000010000000000`

`00000100000000000100000010000`

We only say “yes” if all bits are 1’s.

Can we get the same accuracy while using fewer bits?

Idea 2: Adapt the "run in parallel" approach of the count-min sketch.

We have some fixed number of bits to use. How should we split them across these copies?

`0010000000100001000000000000010000000000`

Can we get the same accuracy while using fewer bits?

Idea 2: Adapt the "run in parallel" approach of the count-min sketch.

We have some fixed number of bits to use. How should we split them across these copies?

`0010010000100000100010`

`1001100001000000001000`

Can we get the same accuracy while using fewer bits?

Idea 2: Adapt the "run in parallel" approach of the count-min sketch.

We have some fixed number of bits to use. How should we split them across these copies?

```
0010001100100100
0101000010001010
0000011100000010
```

Can we get the same accuracy while using fewer bits?

Idea 2: Adapt the "run in parallel" approach of the count-min sketch.

We have some fixed number of bits to use. How should we split them across these copies?

```
001000110010
001000110010
010100001000
000001110000
```

More copies means fewer bits per copy. The error rate per copy is higher, but there's more copies.

Can we get the same accuracy while using fewer bits?

Idea 2: Adapt the "run in parallel" approach of the count-min sketch.

Approach: Use one giant array. Have all hash functions edit and read that array.

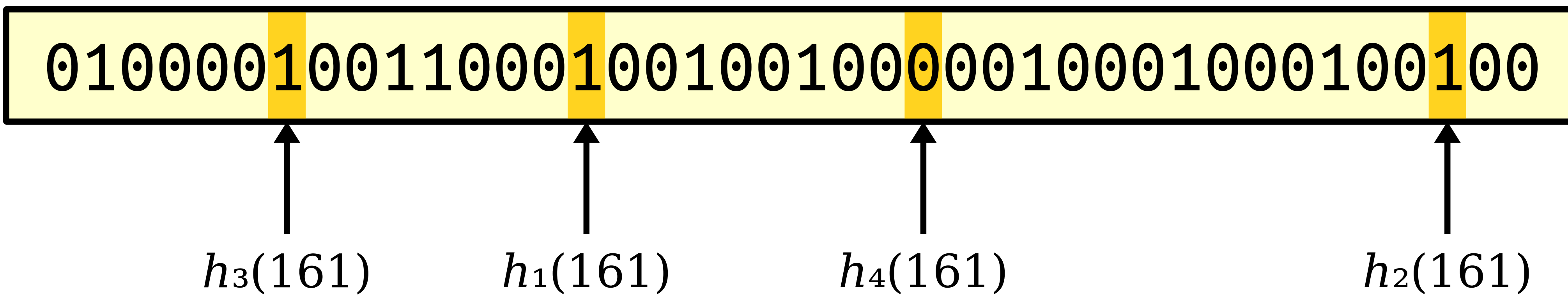
This is called a ***Bloom filter***, named after its inventor.

`01000010011000100100100000010001000100100`

Can we get the same accuracy while using fewer bits?

Assume we use k hash functions, each of which is chosen independently of the others. We'll pick k later on.

(In this example, $k = 4$.)

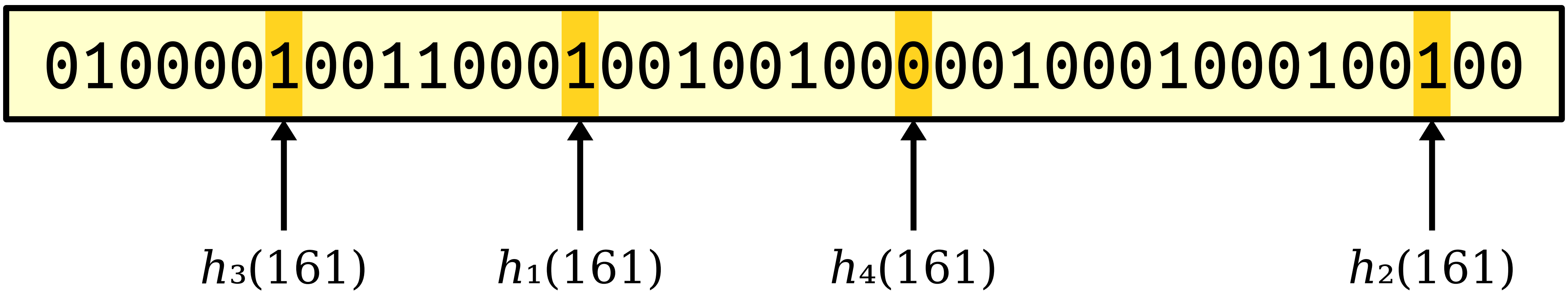


Can we get the same accuracy while using fewer bits?

create(S): Select k hash functions. Hash each element with all hash functions and set the indicated bits to 1.

query(x): Hash x with all k hash functions.

Return whether all the indicated bits are 1.



Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If c is too low, we'll get too many false positives. If c is too large, we'll use too much memory.

`01000010011000100100100000010001000100100`

Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If c is too low, we'll get too many false positives. If c is too large, we'll use too much memory.

`1000100010001100101010001000101`

Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If c is too low, we'll get too many false positives. If c is too large, we'll use too much memory.

`110011001111000111000101`

Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If c is too low, we'll get too many false positives. If c is too large, we'll use too much memory.

111111

Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If c is too low, we'll get too many false positives. If c is too large, we'll use too much memory.

`01000010011000100100100000010001000100100`

Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If k is either too low *or* too high, we'll get too many false positives.

`0100001001100010010010000001000100010010`

Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If k is either too low *or* too high, we'll get too many false positives.

`010100100110101001101000101000100011010 0`

Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If k is either too low *or* too high, we'll get too many false positives.

`0101001111101010011010001011001000110111`

Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If k is either too low *or* too high, we'll get too many false positives.

1111111111111111111111111111111111111111

Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If k is either too low *or* too high, we'll get too many false positives.

`0100001001100010010010000010001000100100`

Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If k is either too low *or* too high, we'll get too many false positives.

```
0000001001000010000000000000001000000100
```

Can we get the same accuracy while using fewer bits?

We have two knobs to turn: the coefficient c in the number of bits $cn \lg \varepsilon^{-1}$, and the number of hash functions k.

Intuition: If k is either too low *or* too high, we'll get too many false positives.

Question: How do we tune k, the number of hash functions?

`0100001001100010010010000001000100010100`

Can we get the same accuracy while using fewer bits?

In what follows, let's have

$$\boldsymbol{m = cn \lg \varepsilon^{-1}}$$

be the number of bits in our array.

How do we quantify our error rate?

Question: In what circumstance do we get a false positive?

Answer: Each of the element's bits are set, but the element isn't in the set S.

Question: What is the probability that this happens?

`0100001001100010010010000001000100010 0100`

How do we quantify our error rate?

Question 1: What is the probability that any particular bit is set?

`0011010100101000`

Focus on a bit at index i.

How do we quantify our error rate?

Question 1: What is the probability that any particular bit is set?

`00110101000101000`

Focus on a bit at index i.

Pick some $x \in S$ and hash function h.

How do we quantify our error rate?

Question 1: What is the probability that any particular bit is set?

`0011010100101000`

Focus on a bit at index i.

Pick some $x \in S$ and hash function h.

What's the probability that $h(x) \neq i$? (Assume truly random hash functions.)

How do we quantify our error rate?

Question 1: What is the probability that any particular bit is set?

`00110101000101000`

Focus on a bit at index i.

Pick some $x \in S$ and hash function h.

What's the probability that $h(x) \neq i$? (Assume truly random hash functions.)

Answer: $1 - 1/m$.

How do we quantify our error rate?

Question 1: What is the probability that any particular bit is set?

`0011010100 0101000`

Focus on a bit at index i.

Pick some $x \in S$ and hash function h.

What's the probability that $h(x) \neq i$? (Assume truly random hash functions.)

Answer: $1 - {}^{1}/_{m}$.

What's the probability that, across all n elements and k hash functions, bit i isn't set?

How do we quantify our error rate?

Question 1: What is the probability that any particular bit is set?

`0011010100101000`

Focus on a bit at index i.

Pick some $x \in S$ and hash function h.

What's the probability that $h(x) \neq i$? (Assume truly random hash functions.)

Answer: $1 - {}^1/_m$.

What's the probability that, across all n elements and k hash functions, bit i isn't set?

Answer: $(1 - {}^1/_m)^{kn}$.

How do we quantify our error rate?

Question 1: What is the probability that any particular bit is set?

Useful fact: $(1 - {}^{1}/_{p})^{p} \approx e^{-1}$.

```
0011010100101000
```

How do we quantify our error rate?

Question 1: What is the probability that any particular bit is set?

```
0011010100010100
```

Useful fact: $(1 - {}^{1}/_{p})^{p} \approx e^{-1}$.

Probability that bit i is unset after inserting n elements:

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

How do we quantify our error rate?

Question 1: What is the probability that any particular bit is set?

```
0011010100101000
```

Useful fact: $(1 - {}^{1}/_{p})^{p} \approx e^{-1}$.

Probability that bit *i* is unset after inserting *n* elements:

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

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

How do we quantify our error rate?

Question 1: What is the probability that any particular bit is set?

`0011010100 0101000`

Useful fact: $(1 - {}^{1}/_{p})^{p} \approx e^{-1}$.

Probability that bit *i* is unset after inserting *n* elements:

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

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

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

How do we quantify our error rate?

Question 1: What is the probability that any particular bit is set?

```
00110101000101000
```

Useful fact: $(1 - {}^{1}/_{p})^{p} \approx e^{-1}$.

Probability that bit i is unset after inserting n elements:

$$
\begin{aligned}
& \left(1 - \frac{1}{m}\right)^{kn} \\
= \quad & \left(\left(1 - \frac{1}{m}\right)^{m}\right)^{\frac{kn}{m}} \\
\approx \quad & e^{-\frac{kn}{m}} \\
= \quad & e^{-k/c \lg \varepsilon^{-1}}
\end{aligned}
$$

How do we quantify our error rate?

Question 2: What is the probability of a false positive?

`0011010100010100 0`

Probability that a fixed bit is 1 after n elements have been added:

$$\approx \; 1 - e^{-k / c \lg \varepsilon^{-1}}$$

How do we quantify our error rate?

Question 2: What is the probability of a false positive?

`00110101000101000`

Probability that a fixed bit is 1 after n elements have been added:

$$\approx \; 1 - e^{-k / c \lg \varepsilon^{-1}}$$

False positive probability is approximately

How do we quantify our error rate?

Question 2: What is the probability of a false positive?

`0011010100010101000`

Probability that a fixed bit is 1 after n elements have been added:

$$\approx 1 - e^{-k / c \lg \varepsilon^{-1}}$$

False positive probability is approximately

How do we quantify our error rate?

Question 2: What is the probability of a false positive?

`00110101000101000`

Probability that a fixed bit is 1 after n elements have been added:

$$\approx 1 - e^{-k / c \lg \varepsilon^{-1}}$$

False positive probability is approximately

$$(1 - e^{-k / c \lg \varepsilon^{-1}})^k$$

How do we quantify our error rate?

Question 2: What is the probability of a false positive?

`0011010100001010000`

This value isn't exactly correct because certain bits being 1 decrease the probability that other bits are 1. With a more advanced analysis we can show that this is very close to the true value.

Probability that a fixed bit is 1 after n elements have been added:

$$\approx \ 1 - e^{-k / c \lg \varepsilon^{-1}}$$

False positive probability is approximately

$$(1 - e^{-k / c \lg \varepsilon^{-1}})^k$$

How do we quantify our error rate?

Question 2: What is the probability of a false positive?

`00110101000101000`

This value isn't exactly correct because certain bits being 1 decrease the probability that other bits are 1. With a more advanced analysis we can show that this is very close to the true value.

Probability that a fixed bit is 1 after n elements have been added:

$$\approx \; 1 - e^{-k / c \lg \varepsilon^{-1}}$$

False positive probability is approximately

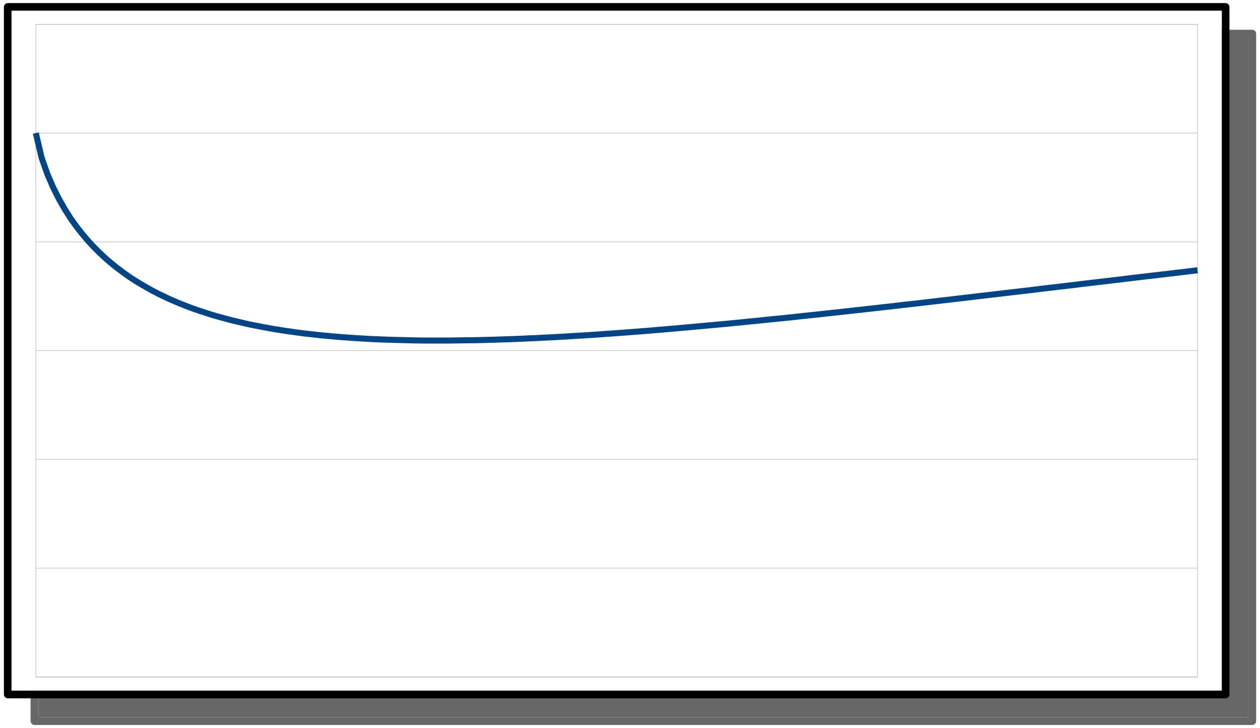
$$(1 - e^{-k / c \lg \varepsilon^{-1}})^k$$

Question: What choice of k minimizes this expression?

How do we quantify our error rate?

Goal: Pick k to minimize

$$(1 - e^{-k / c \lg \varepsilon^{-1}})^k.$$



How many hash functions should we use?

Goal: Pick k to minimize

$$(1 - e^{-k / c \lg \varepsilon^{-1}})^k.$$

Claim: This expression is minimized when

$$\boldsymbol{k = c \lg \varepsilon^{-1} \ln 2}.$$

You can show this using some symmetry arguments or calculus.

How many hash functions should we use?

Goal: Pick k to minimize

$$(1 - e^{-k / c \lg \varepsilon^{-1}})^k.$$

Claim: This expression is minimized when

$$\boldsymbol{k = c \lg \varepsilon^{-1} \ln 2}.$$

You can show this using some symmetry arguments or calculus.

Good exercise: This claim is often repeated and seldom proved. Confirm I am not perpetuating lies.

How many hash functions should we use?

Goal: Pick k to minimize

$$(1 - e^{-k / c \lg \varepsilon^{-1}})^k.$$

Claim: This expression is minimized when

$$\boldsymbol{k = c \lg \varepsilon^{-1} \ln 2}.$$

You can show this using some symmetry arguments or calculus.

Good exercise: This claim is often repeated and seldom proved. Confirm I am not perpetuating lies.

Challenge: Give an explanation for this result that is “immediately obvious” from the original expression.

How many hash functions should we use?

The false positive rate is

$$(1 - e^{-k / c \lg \varepsilon^{-1}})^k.$$

and we know to pick

$$k = c \lg \varepsilon^{-1} \ln 2.$$

Plugging this value into the expression gives a false positive rate of

$$\varepsilon^{c \ln 2}$$

(The derivation, for those of you who are curious.)

$$\begin{aligned}
& \left(1 - e^{-k/c \lg \varepsilon^{-1}}\right)^k \\
= \; & \left(1 - e^{-c \lg \varepsilon^{-1} \ln 2 / c \lg \varepsilon^{-1}}\right)^k \\
= \; & \left(1 - e^{-\ln 2}\right)^k \\
= \; & \left(1 - \frac{1}{2}\right)^k \\
= \; & 2^{-k} \\
= \; & 2^{-c \lg \varepsilon^{-1} \ln 2} \\
= \; & 2^{\lg \varepsilon^{c \ln 2}} \\
= \; & \varepsilon^{c \ln 2}
\end{aligned}$$

Knowing what we know now, how many bits do we need to get a false positive rate of ε?

Our false positive rate, as a function of c, is

$$\varepsilon^{c \ln 2}.$$

Our goal is to get a false positive rate of ε.

To do so, pick

$$\boldsymbol{c = 1 / \ln 2}$$
$$\boldsymbol{= \lg e}$$

Which is about 1.44.

Knowing what we know now, how many bits do we need to get a false positive rate of ε?

Given a number of elements n and an error rate ε, pick

$$m = cn \lg \varepsilon^{-1}$$

$$k = c \lg \varepsilon^{-1} \ln 2.$$

Optimal c:

$$c = \lg e$$

How did we do overall?

Given a number of elements n and an error rate ε, pick

$$m = n \lg \varepsilon^{-1} \lg e$$

$$k = \lg \varepsilon^{-1}$$

Optimal c:

$$c = \lg e$$

How did we do overall?

Given a number of elements n and an error rate ε, pick

$$m = n \lg \varepsilon^{-1} \lg e$$

$$k = \lg \varepsilon^{-1}$$

Space usage: **$\Theta(n \log \varepsilon^{-1})$**. *(Within a factor of 1.44 of optimal!)*

Query time: **$O(\log \varepsilon^{-1})$**.

More recent data structures, like the ***quotient filter*** and ***cuckoo filter***, push this coefficient down and are still viable in practice.

We know that it's possible to reduce this coefficient to $1 + o(1)$ (***Pagh, Pagh, and Rao's optimal AMQ structure***), though this is currently only of theoretical interest.

How did we do overall?

To Summarize

- ***Chebyshev's inequality*** is a useful concentration inequality for two-sided errors. It involves bounding the variance of the underlying variable.
- To simplify variance expressions, look for pairwise independence.
- Some random variables (± 1 variables, indicator variables, etc.) simplify nicely when squared.
- Taking the median of many estimators with two-sided error is a good way to amplify success probabilities. The ***Chernoff bound*** is helpful for showing how effective this strategy is.
- Information-theoretic lower bounds can be helpful in determining how much progress can be made with a data structure.

Next Time

- ***Cuckoo Hashing***
  - Hashing with worst-case O(1) lookups.
- ***Random Graph Theory***
  - Properties of random bipartite graphs.