Hashing and Sketching

Part One

Outline for Today

- ***Hash Functions***
  - Understanding our basic building blocks.
- ***Frequency Estimation***
  - Estimating how many times we've seen something.
- ***Concentration Inequalities***
  - "Correct on expectation" versus "correct with high probability."
- ***Probability Amplification***
  - Increasing our confidence in our answers.

Preliminaries: ***Hash Functions***

Hashing in Practice

- Hash functions are used extensively in programming and software engineering:
  - They make hash tables possible: think C++ `std::hash`, Python's `__hash__`, or Java's `Object.hashCode()`.
  - They're used in cryptography: SHA-256, HMAC, etc.
- ***Question:*** When we're in Theoryland, what do we mean when we say "hash function?"

Hashing in Theoryland

- In Theoryland, a hash function is a function from some domain called the ***universe*** (typically denoted $\mathscr{U}$) to some codomain.
- The codomain is usually a set of the form

$$\mathbf{[m]} = \{0, 1, 2, 3, \ldots, m - 1\}$$

$$h : \mathscr{U} \to [m]$$

Hashing in Theoryland

- ***Intuition:*** No matter how clever you are with designing a specific hash function, that hash function isn't random, and so there will be pathological inputs.
  - You can formalize this with the pigeonhole principle.
- ***Idea:*** Rather than finding the One True Hash Function, we'll assume we have a collection of hash functions to pick from, and we'll choose which one to use randomly.

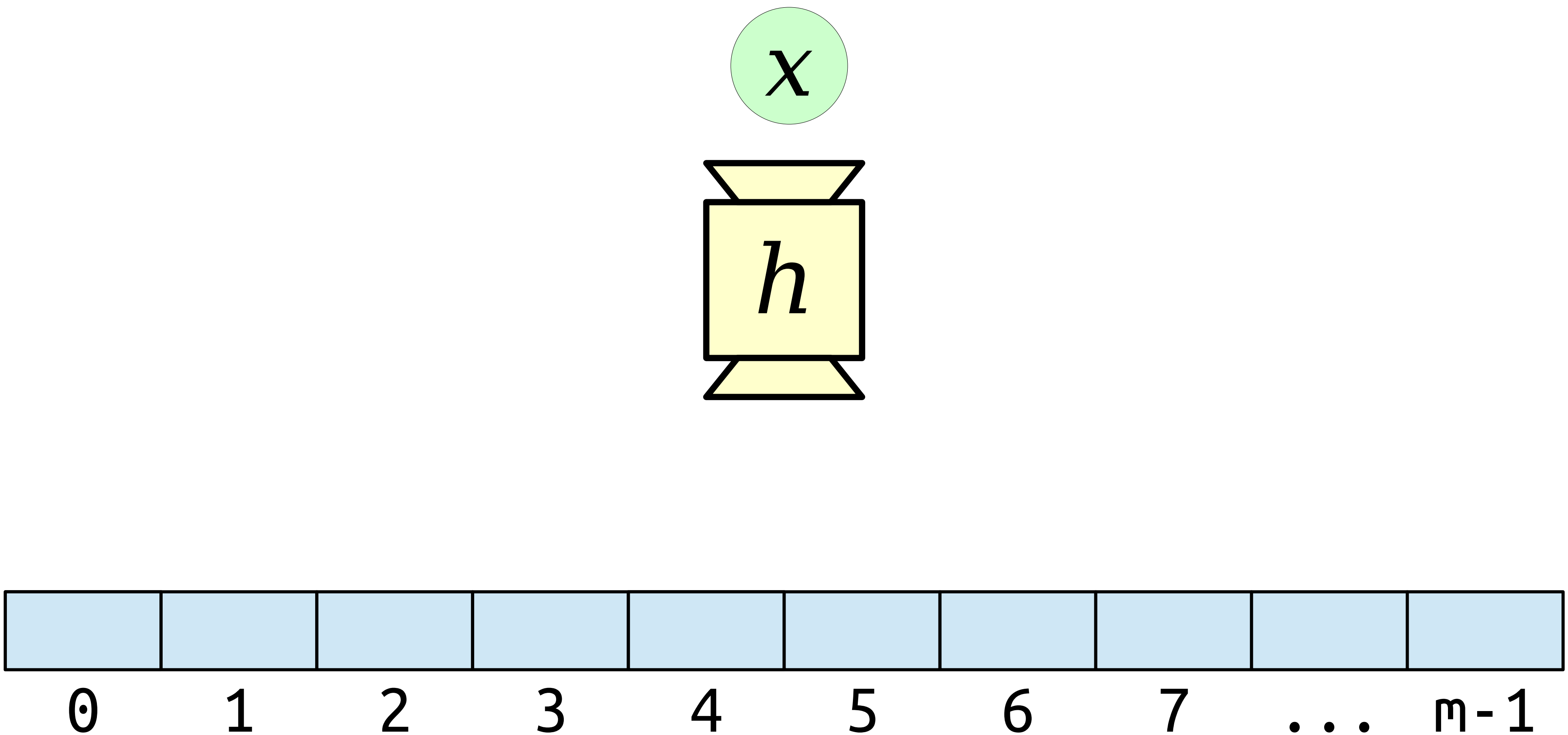
Families of Hash Functions

- A ***family*** of hash functions is a set $\mathscr{H}$ of hash functions with the same domain and codomain.
- We can then introduce randomness into our data structures by sampling a random hash function from $\mathscr{H}$.
- ***Key Point:*** The randomness in our data structures almost always derives from the random choice of hash functions, not from the data.

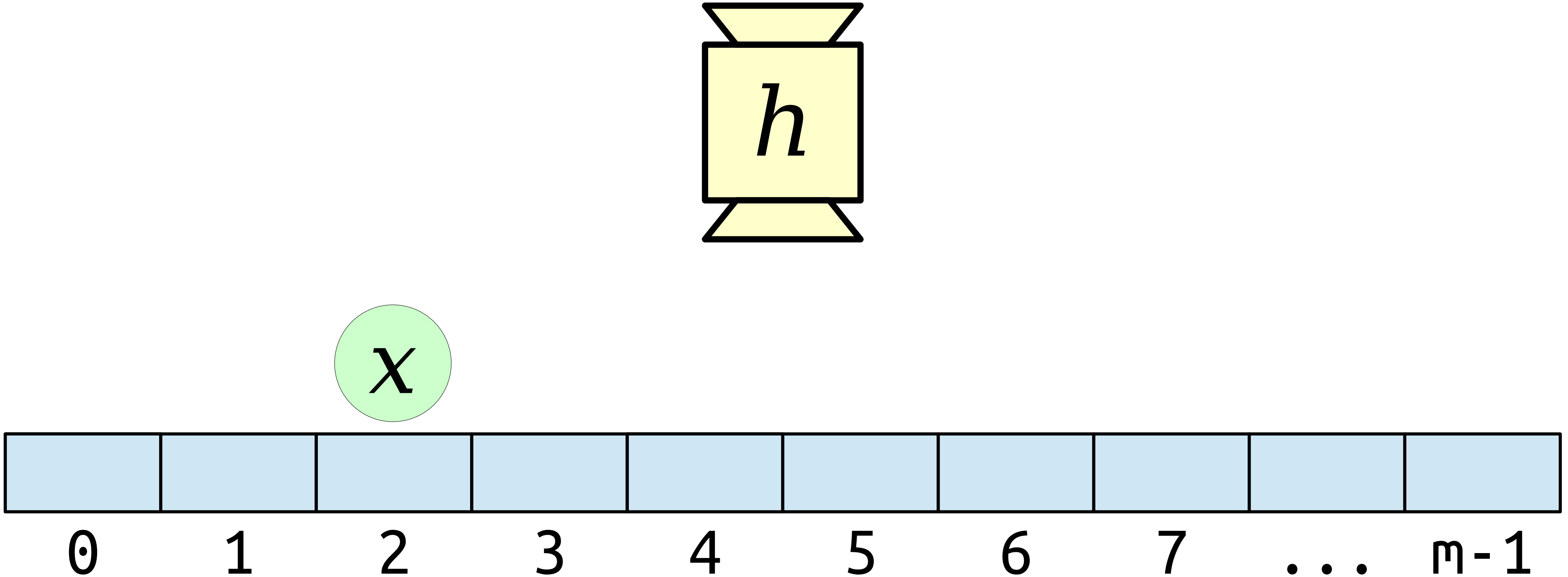
Data is adversarial.
Hash function selection is random.

- ***Question:*** What makes a family of hash functions $\mathscr{H}$ a "good" family of hash functions?

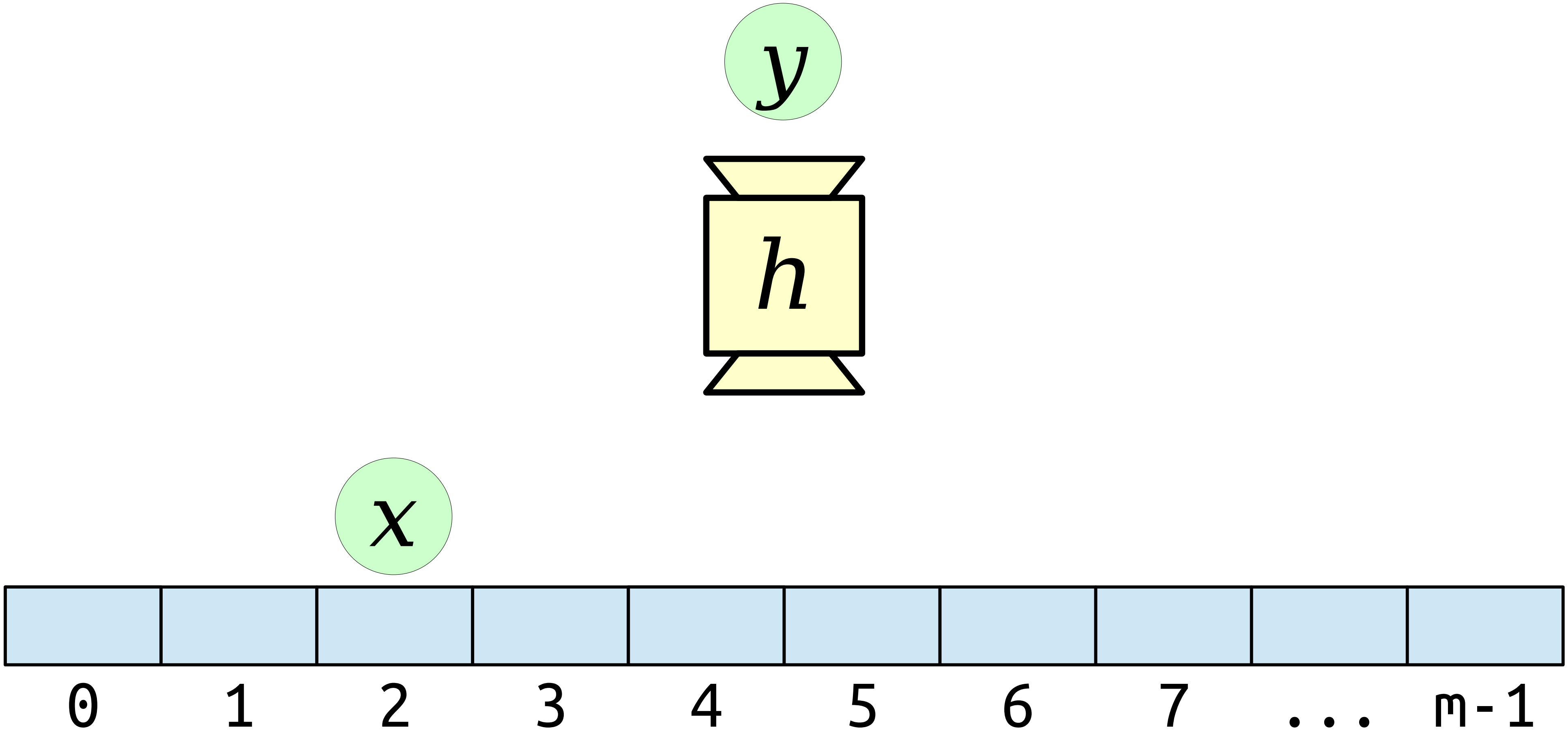
Goal: If we pick $h \in \mathscr{H}$ uniformly at random, then h should distribute elements uniformly randomly.



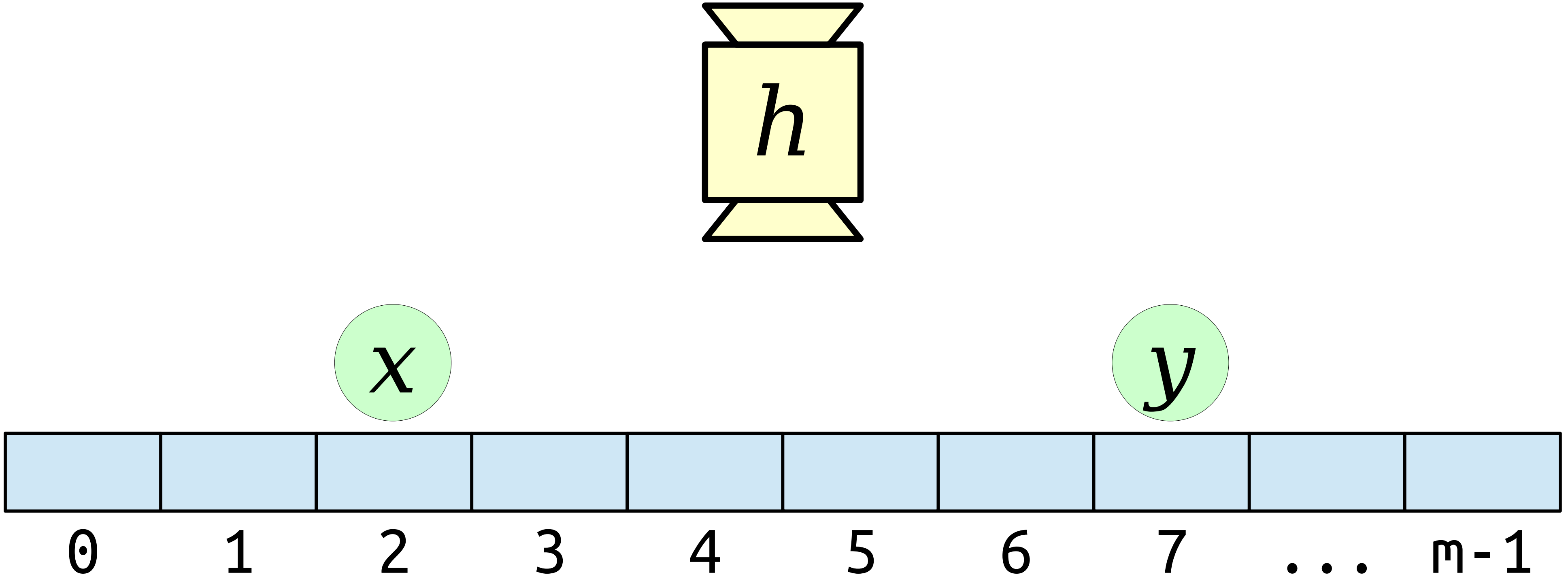
Goal: If we pick $h \in \mathscr{H}$ uniformly at random, then h should distribute elements uniformly randomly.



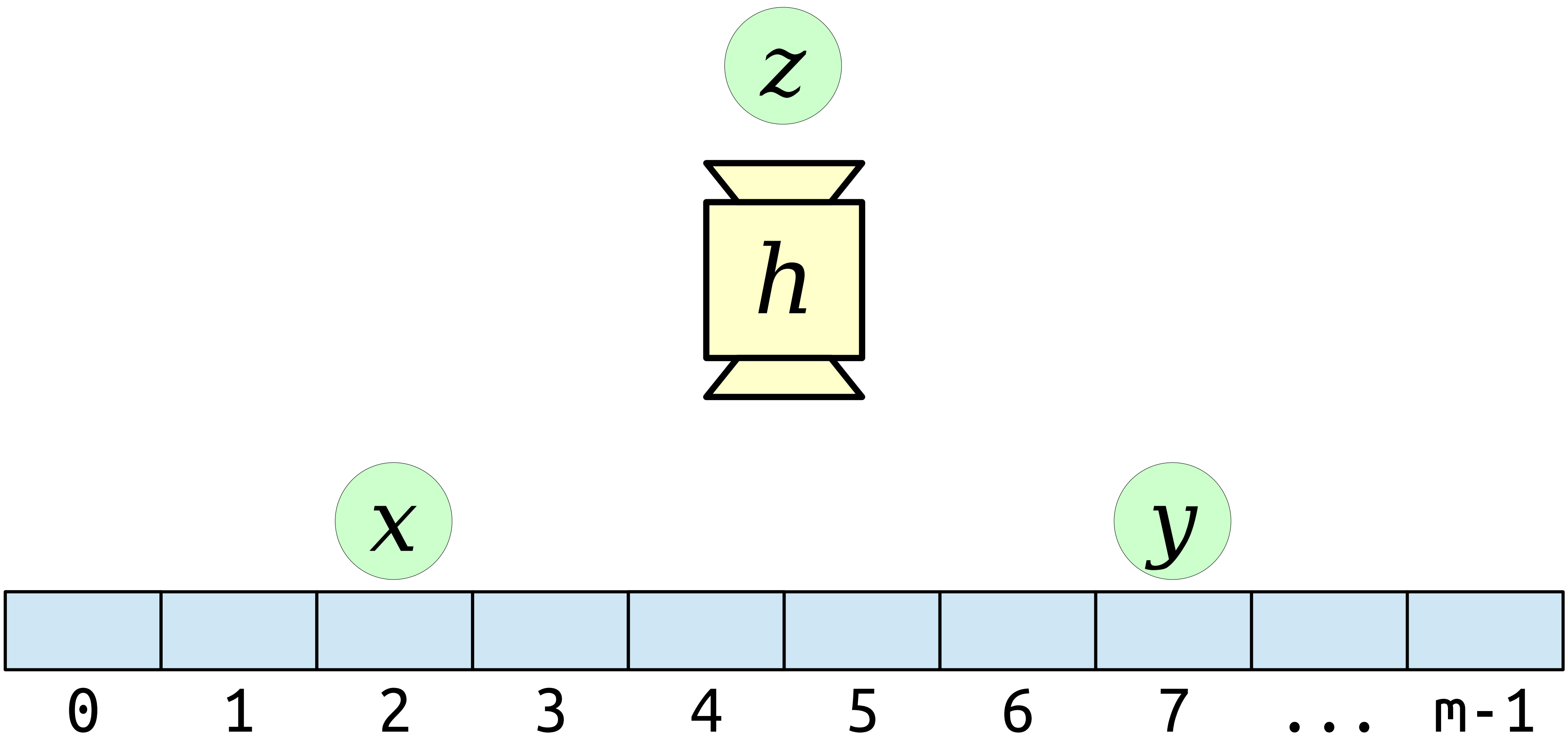
Goal: If we pick $h \in \mathscr{H}$ uniformly at random, then h should distribute elements uniformly randomly.



Goal: If we pick $h \in \mathscr{H}$ uniformly at random, then h should distribute elements uniformly randomly.



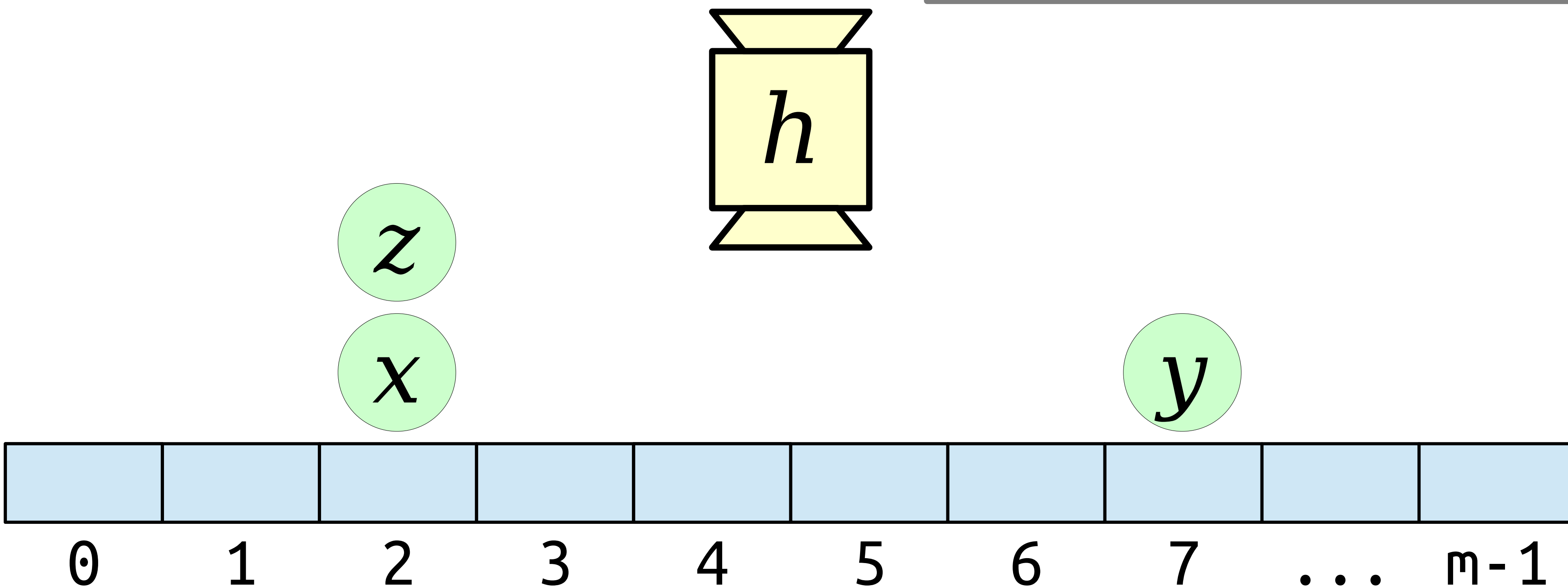
Goal: If we pick $h \in \mathscr{H}$ uniformly at random, then h should distribute elements uniformly randomly.



Goal: If we pick $h \in \mathscr{H}$ uniformly at random, then h should distribute elements uniformly randomly.

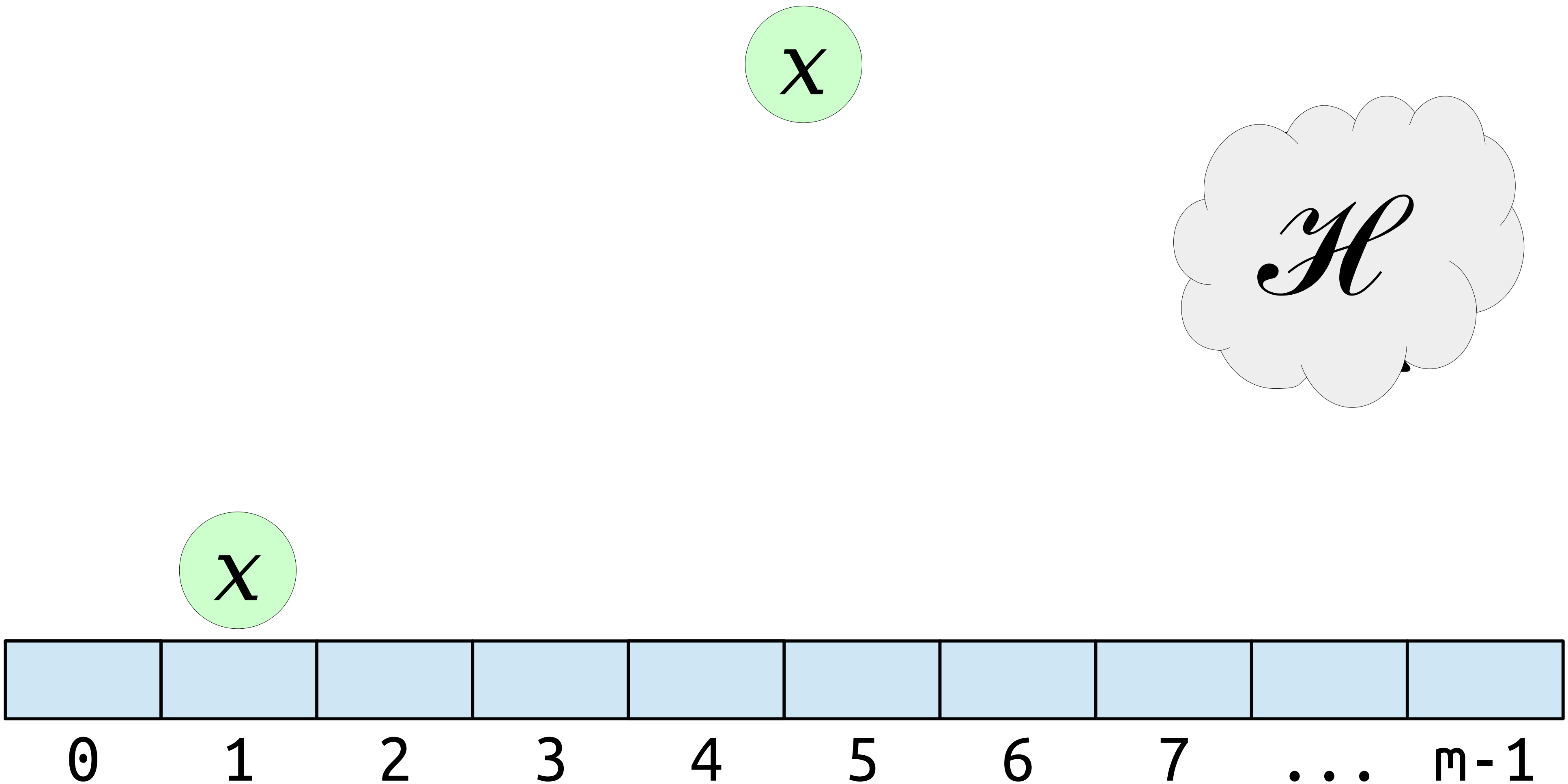
Problem: A hash function that distributes n elements uniformly at random over $[m]$ requires $\Omega(n \log m)$ space in the worst case.

Question: Do we actually need true randomness? Or can we get away with something weaker?



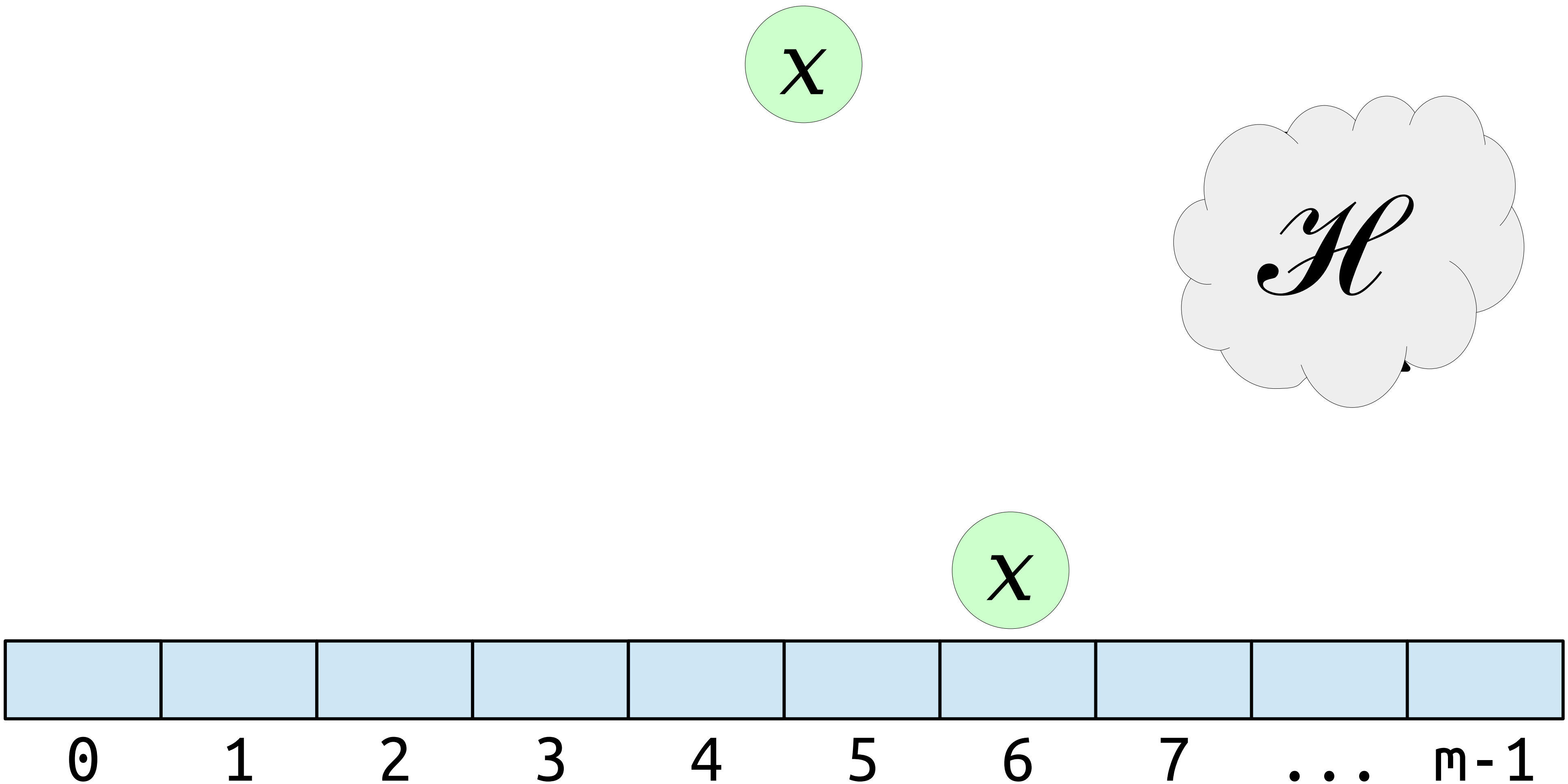
Distribution Property:
Each element should have an equal probability of being placed in each slot.

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



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 $[m]$.



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

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

Find an “obviously bad” family of hash functions that satisfies the distribution property.

Formulate a hypothesis, but ***don’t post anything in chat just yet***.

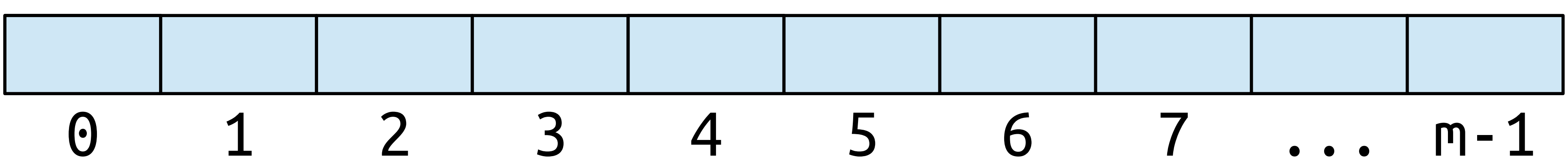
0	1	2	3	4	5	6	7	...	m-1

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 $[m]$.

Find an "obviously bad" family of hash functions that satisfies the distribution property.

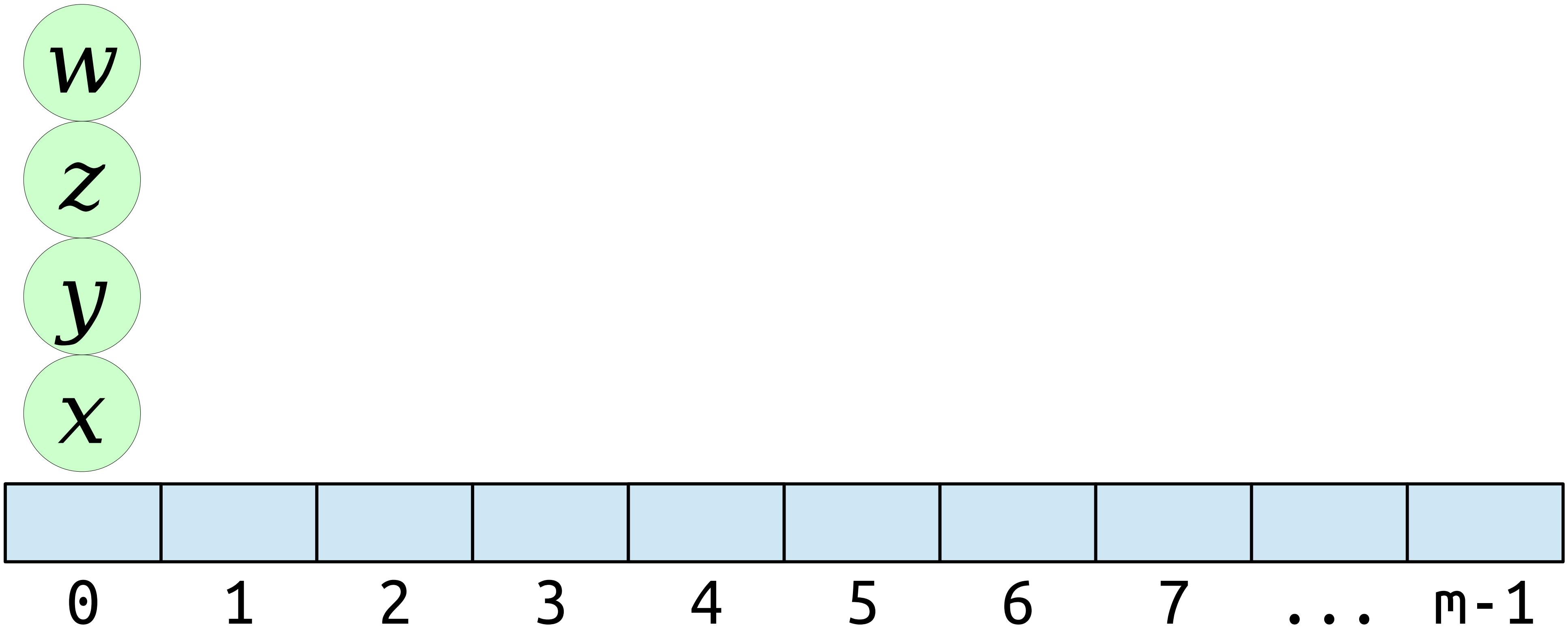
Now, ***private chat me your best guess***. Not sure? Just answer "??".



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 $[m]$.

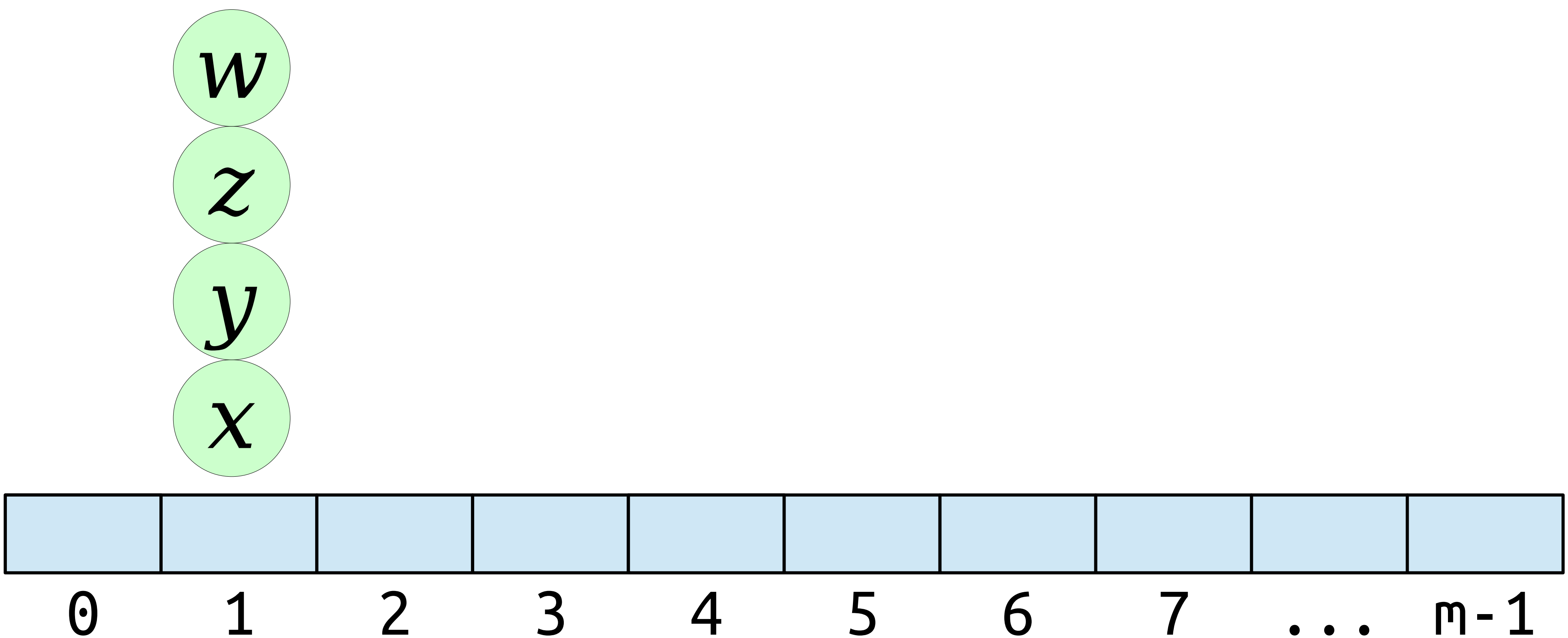
Problem: This rule doesn't guarantee that elements are spread out.



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 $[m]$.

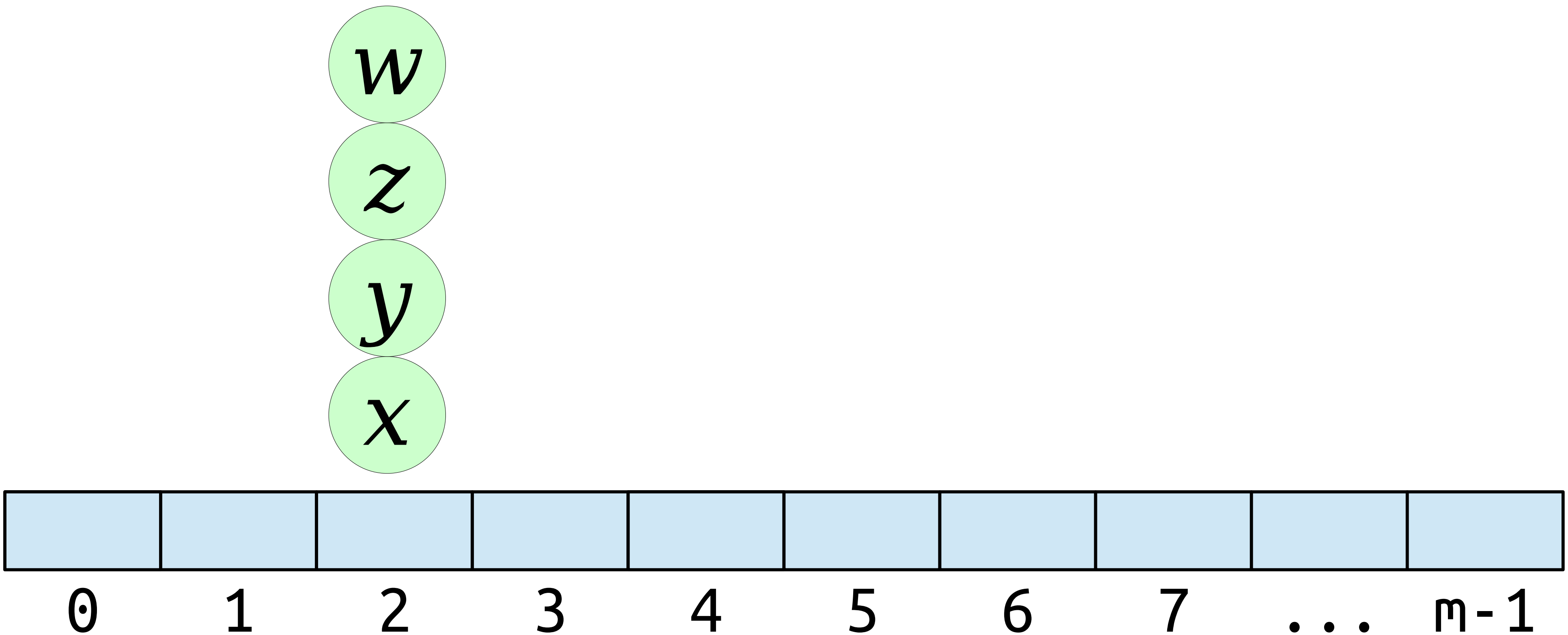
Problem: This rule doesn't guarantee that elements are spread out.



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

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

Problem: This rule doesn't guarantee that elements are spread out.

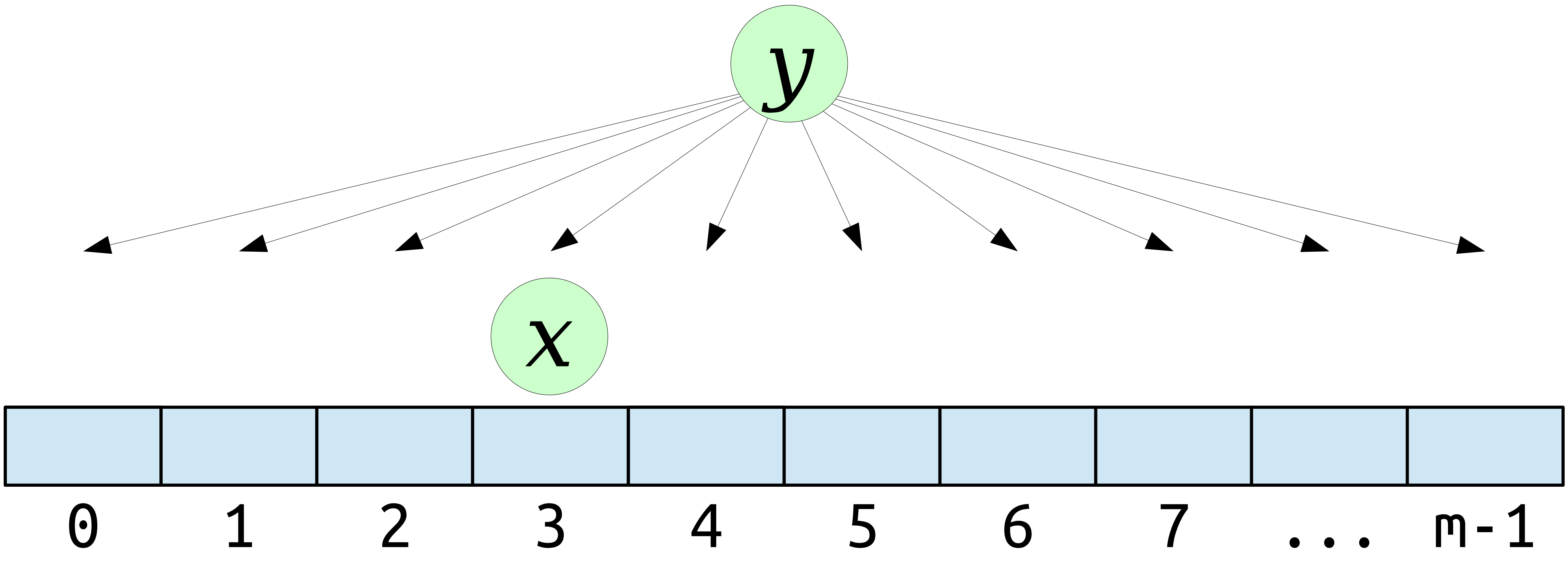


Distribution Property:
Each element should have an equal probability of being placed in each slot.

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



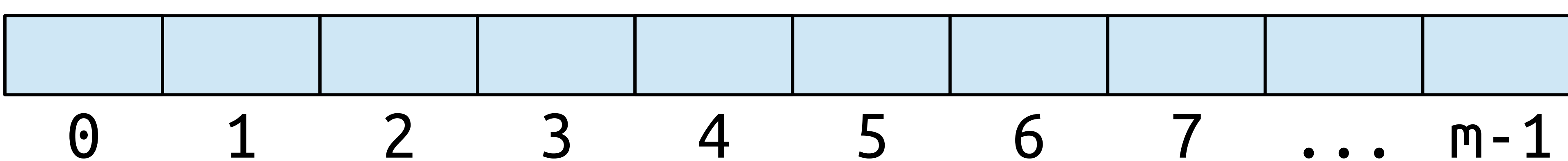
Distribution Property: Each element should have an equal probability of being placed in each slot.

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.

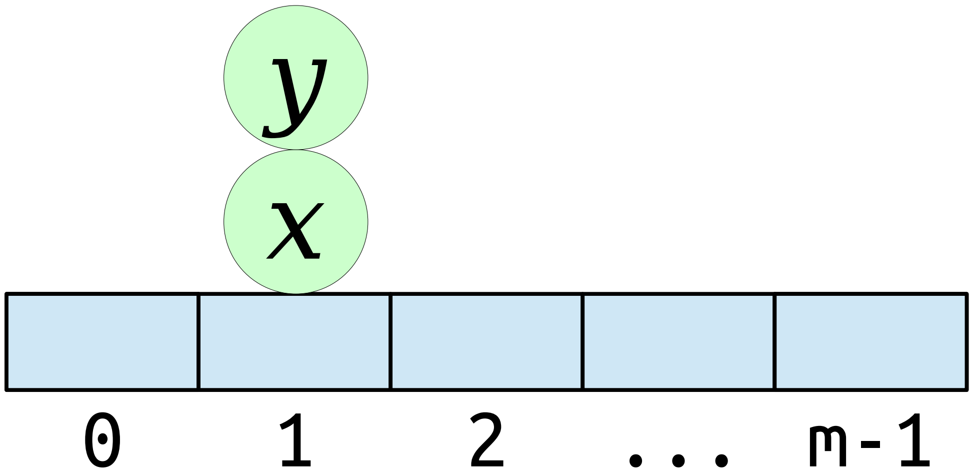
A family of hash functions $\mathscr{H}$ is called ***2-independent*** (or ***pairwise independent***) if it satisfies the distribution and independence properties.



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

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

Intuition:
2-independence means any pair of elements is unlikely to collide.

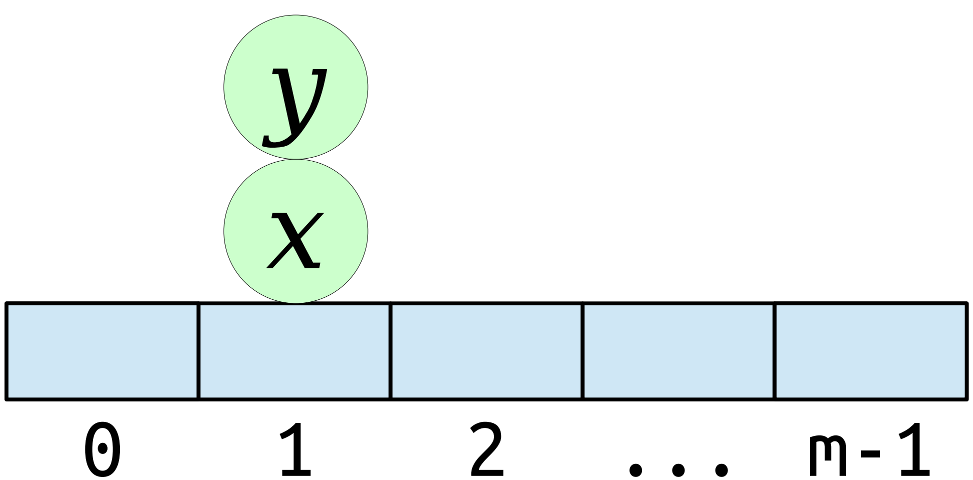


For any $x \in \mathscr{U}$ and random $h \in \mathscr{H}$, the value of $h(x)$ is uniform over $[m]$.

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

Intuition:
2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$



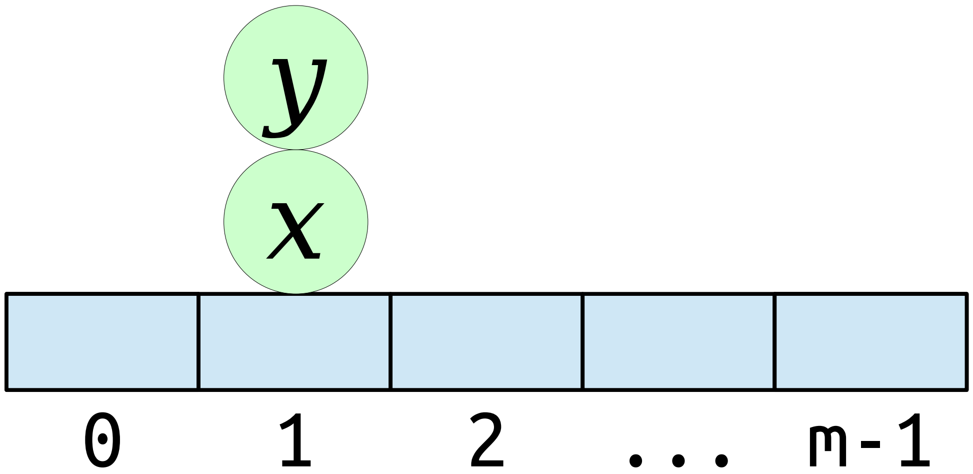
For any $x \in \mathscr{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over $[m]$.

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

Intuition: 2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$

Question: Where did these elements collide with one another?

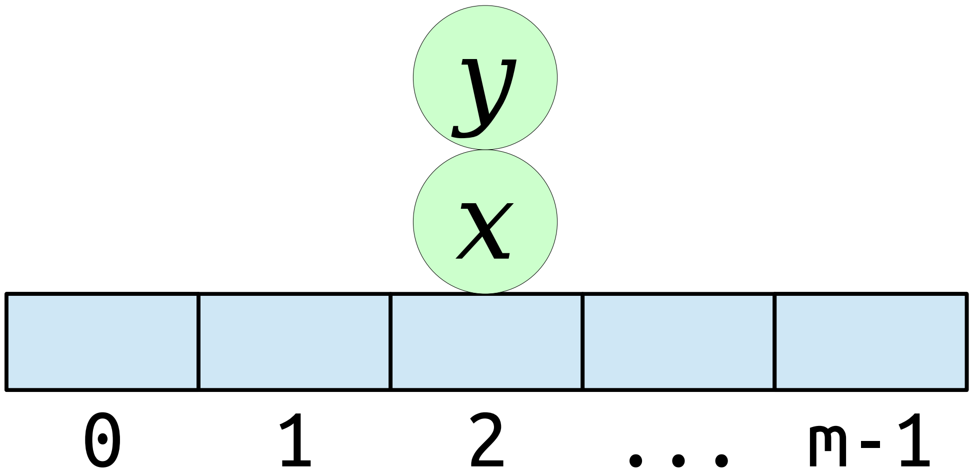


For any $x \in \mathscr{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over $[m]$.

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

Intuition: 2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$



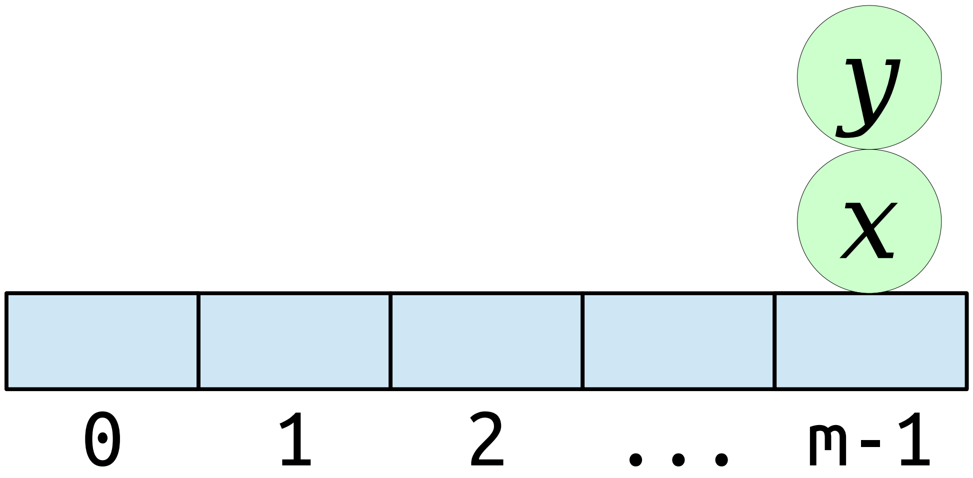
Question: Where did these elements collide with one another?

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

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

Intuition: 2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$



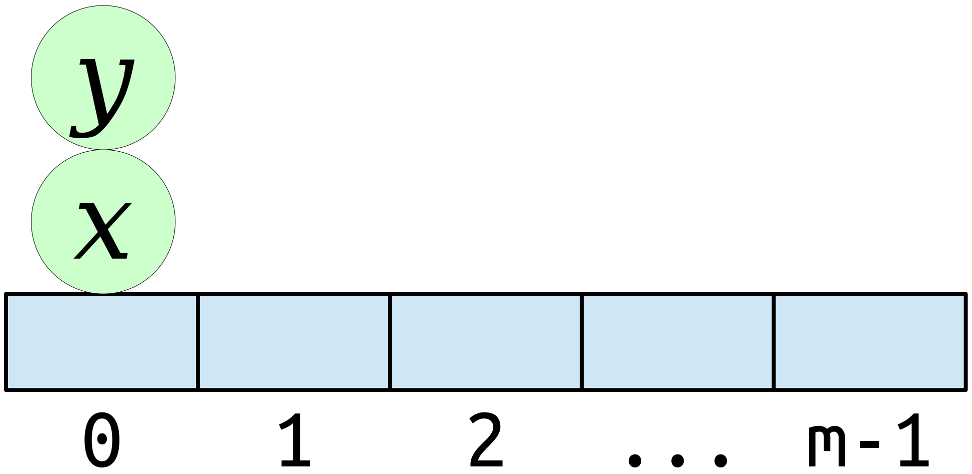
Question: Where did these elements collide with one another?

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

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

Intuition: 2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$



Question: Where did these elements collide with one another?

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

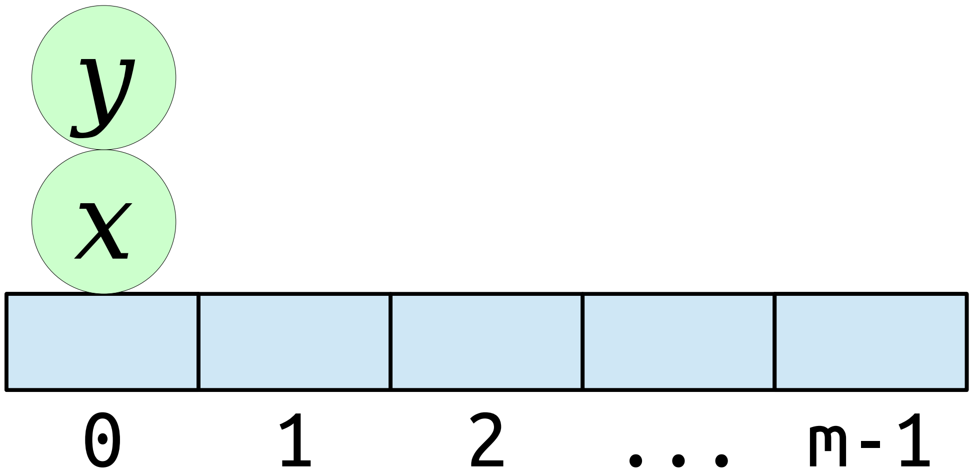
For any distinct $x, y \in \mathscr{U}$ and random $h \in \mathcal{H}$, $h(x)$ and $h(y)$ are independent random variables.

Intuition: 2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i \wedge h(y) = i]$$

Question: Where did these elements collide with one another?

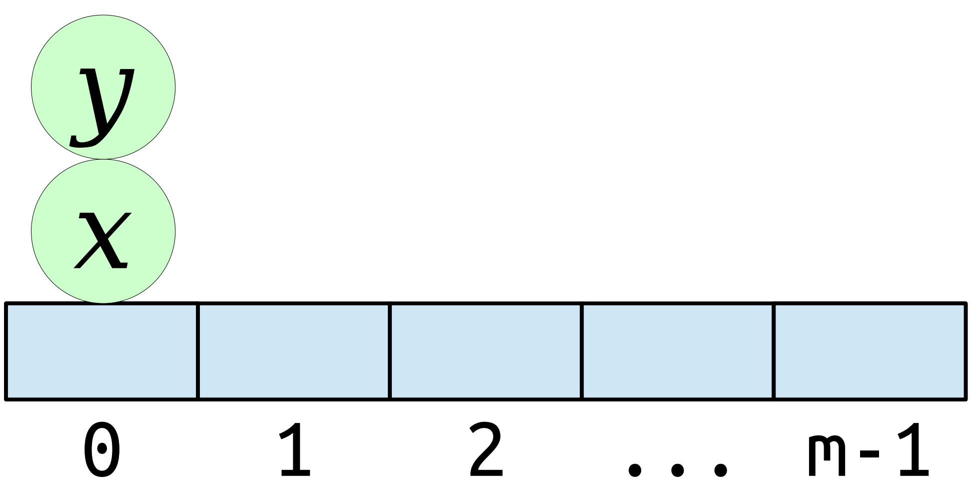


For any $x \in \mathscr{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over $[m]$.

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

Intuition:
2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$
$$= \sum_{i=0}^{m-1} \Pr[h(x) = i \wedge h(y) = i]$$



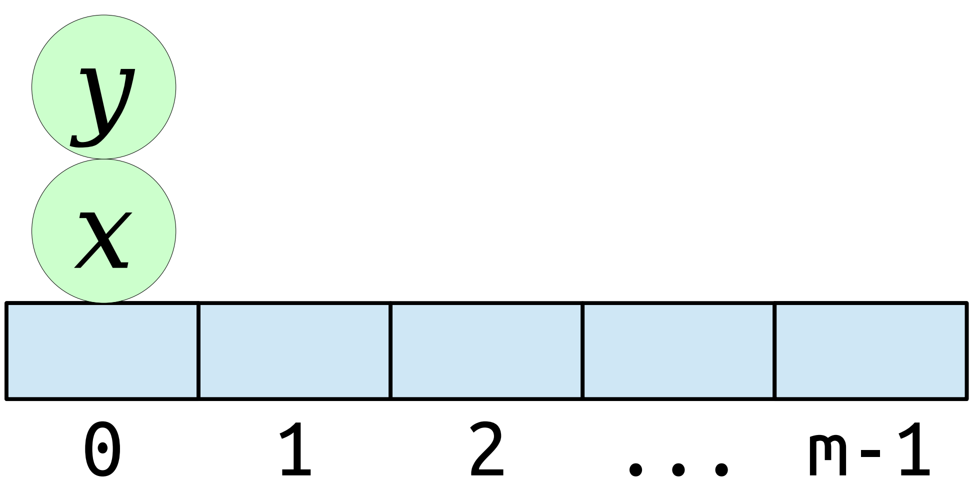
For any $x \in \mathscr{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over $[m]$.

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

Intuition:
2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i \wedge h(y) = i]$$



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

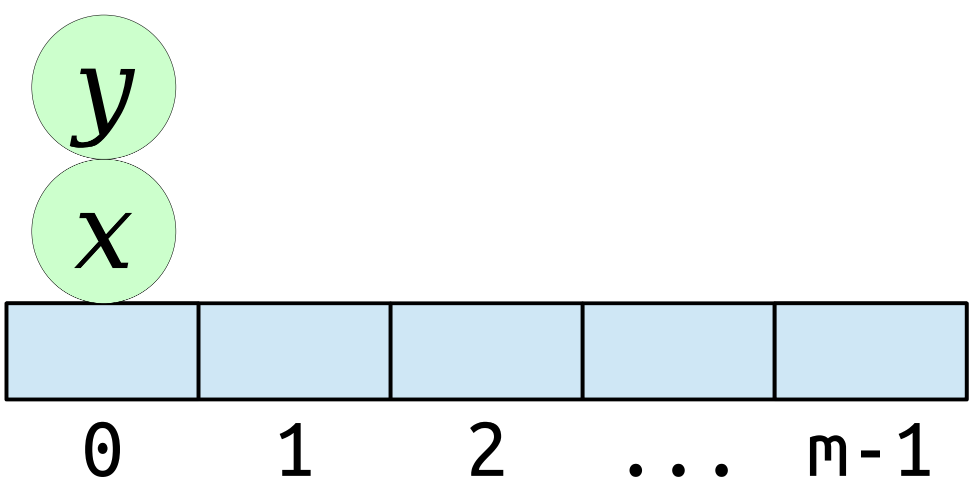
For any distinct $x, y \in \mathscr{U}$ and random $h \in \mathcal{H}$, $h(x)$ and $h(y)$ are independent random variables.

Intuition:
2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i \wedge h(y) = i]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i]$$



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

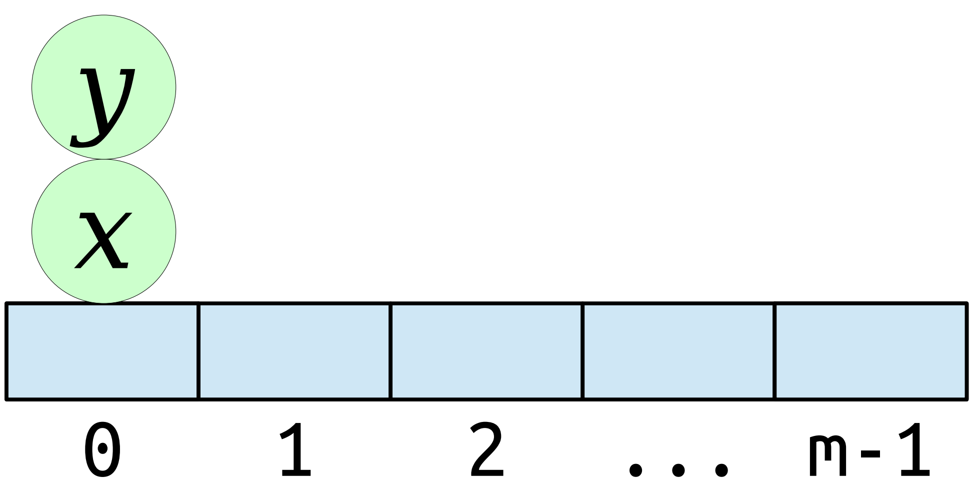
For any distinct $x, y \in \mathscr{U}$ and random $h \in \mathcal{H}$, $h(x)$ and $h(y)$ are independent random variables.

Intuition:
2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i \wedge h(y) = i]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i]$$



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

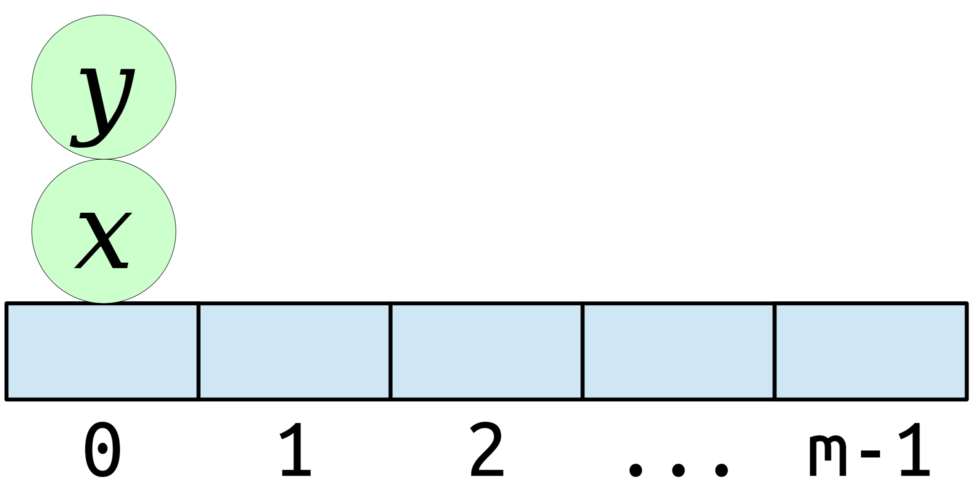
For any distinct $x, y \in \mathscr{U}$ and random $h \in \mathcal{H}$, $h(x)$ and $h(y)$ are independent random variables.

Intuition:
2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i \wedge h(y) = i]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i]$$



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

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

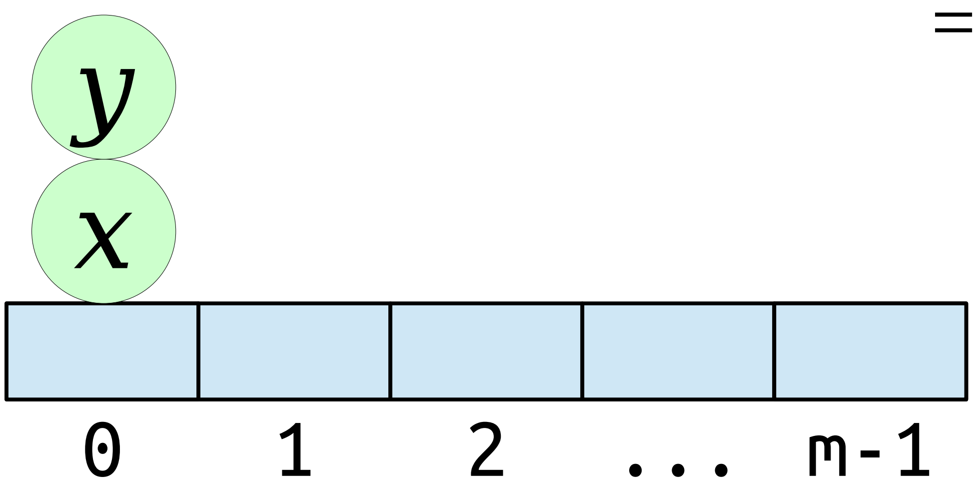
Intuition:
2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i \wedge h(y) = i]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i]$$

$$= \sum_{i=0}^{m-1} \frac{1}{m^2}$$



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

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

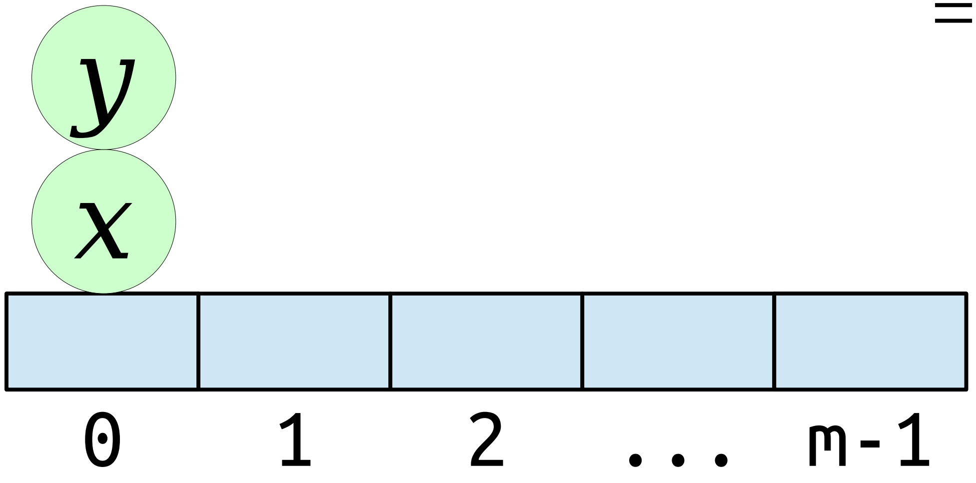
Intuition:
2-independence means any pair of elements is unlikely to collide.

$$\Pr[h(x) = h(y)]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i \wedge h(y) = i]$$

$$= \sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i]$$

$$= \sum_{i=0}^{m-1} \frac{1}{m^2}$$

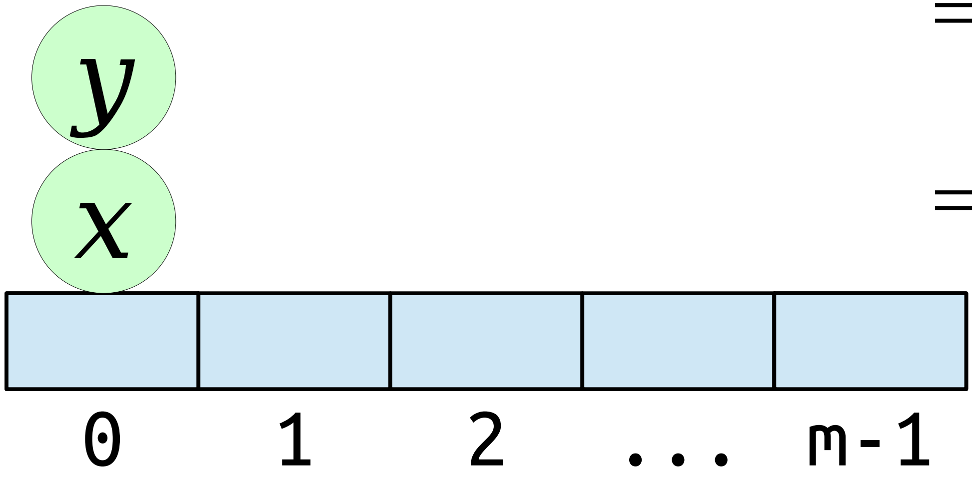


For any $x \in \mathscr{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over $[m]$.

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

Intuition:
2-independence means any pair of elements is unlikely to collide.

$$\begin{aligned} & \Pr[h(x) = h(y)] \\ = & \sum_{i=0}^{m-1} \Pr[h(x) = i \wedge h(y) = i] \\ = & \sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i] \\ = & \sum_{i=0}^{m-1} \frac{1}{m^2} \\ = & \frac{1}{m} \end{aligned}$$



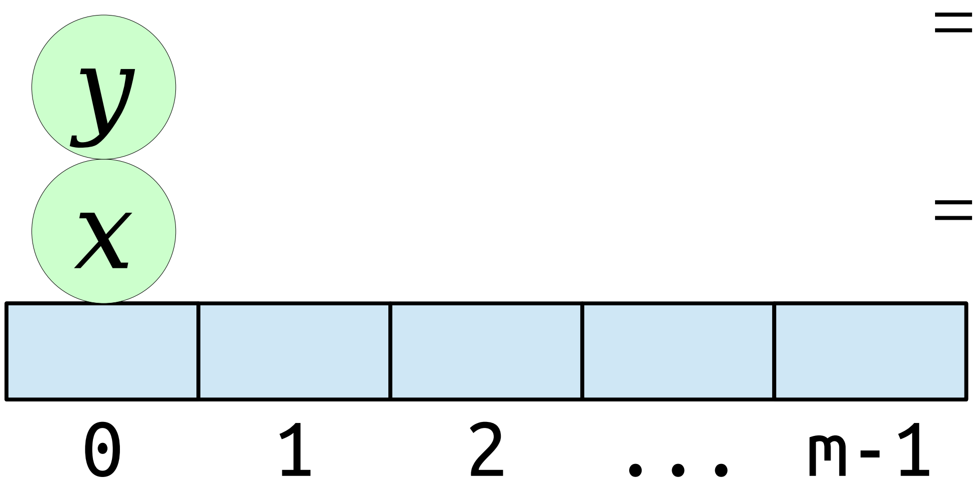
For any $x \in \mathscr{U}$ and random $h \in \mathcal{H}$, the value of $h(x)$ is uniform over $[m]$.

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

Intuition:
2-independence means any pair of elements is unlikely to collide.

$$
\begin{aligned}
&\Pr[h(x) = h(y)] \\
=\ &\sum_{i=0}^{m-1} \Pr[h(x) = i \wedge h(y) = i] \\
=\ &\sum_{i=0}^{m-1} \Pr[h(x) = i] \cdot \Pr[h(y) = i] \\
=\ &\sum_{i=0}^{m-1} \frac{1}{m^2} \\
=\ &\frac{1}{m}
\end{aligned}
$$

This is the same as if h were a truly random function.



For more on hashing outside of Theoryland, check out ***this Stack Exchange post***.

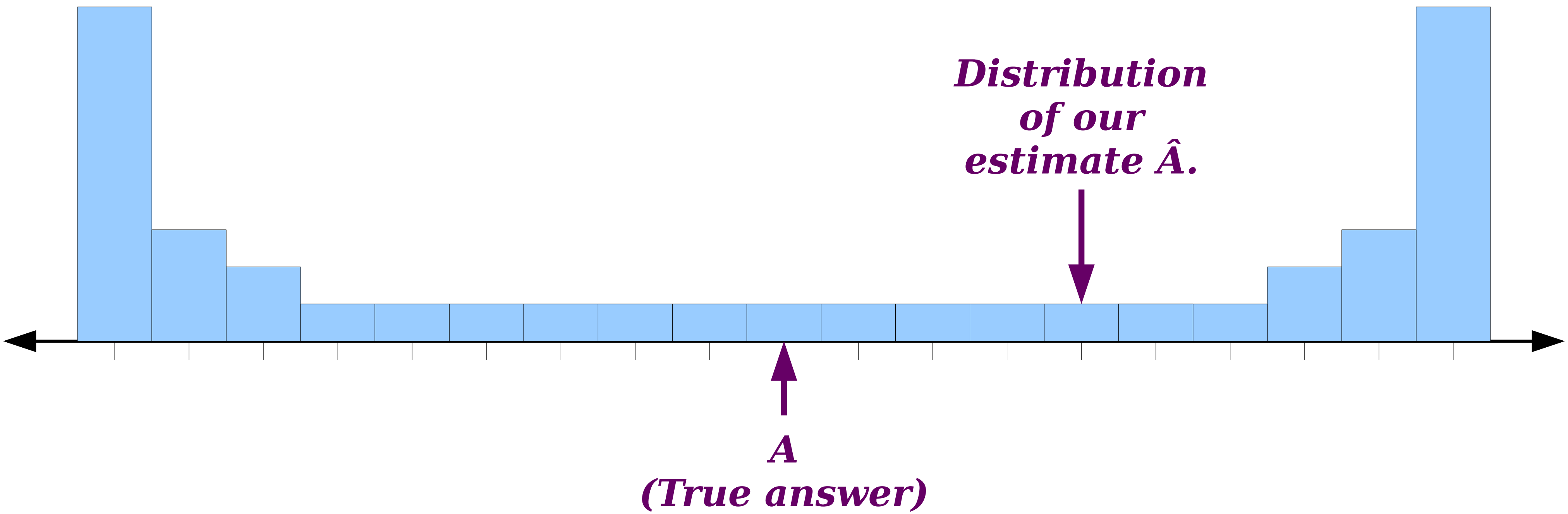
Approximating Quantities

What makes for a good “approximate” solution?

Let $\boldsymbol{A}$ be the true answer. Let $\boldsymbol{\hat{A}}$ be a random variable denoting our estimate.

This would not make for a good estimate. However, we have $E[\hat{A}] = A$.

Observation 1: Being correct in expectation isn't sufficient.

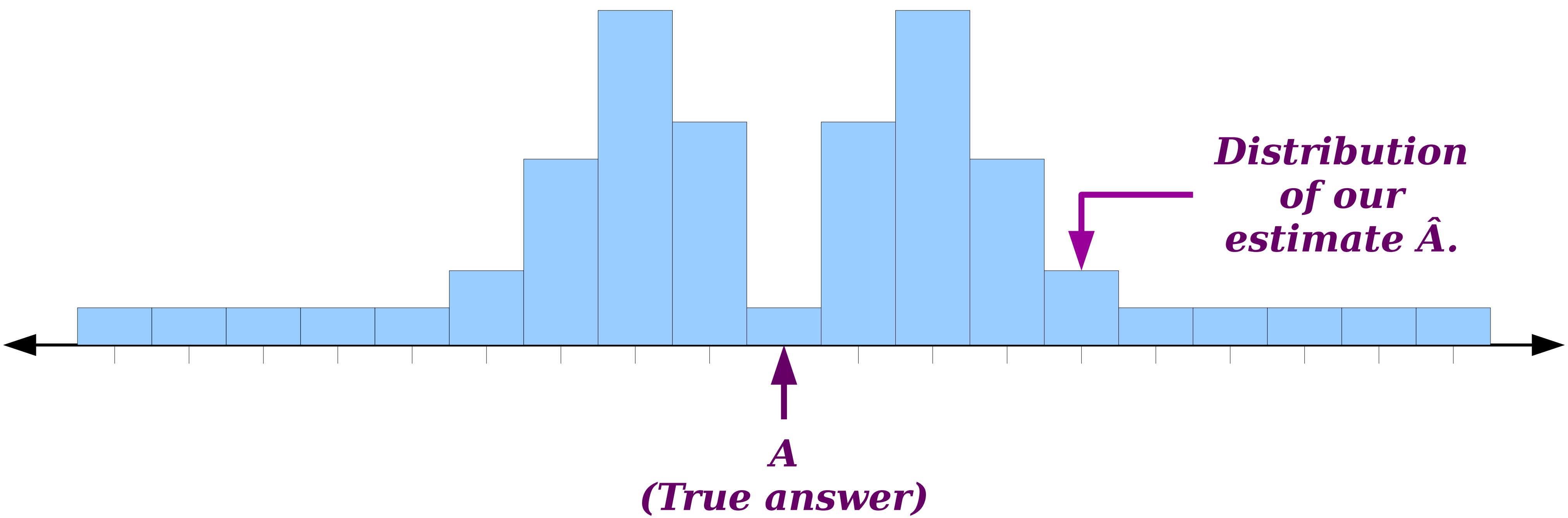


What does it mean for an approximation to be "good"?

Let $\boldsymbol{A}$ be the true answer. Let $\boldsymbol{\hat{A}}$ be a random variable denoting our estimate.

It's unlikely that we'll get the right answer, but we're probably going to be close.

Observation 2: The difference $|\hat{A} - A|$ between our estimate and the truth should ideally be small.

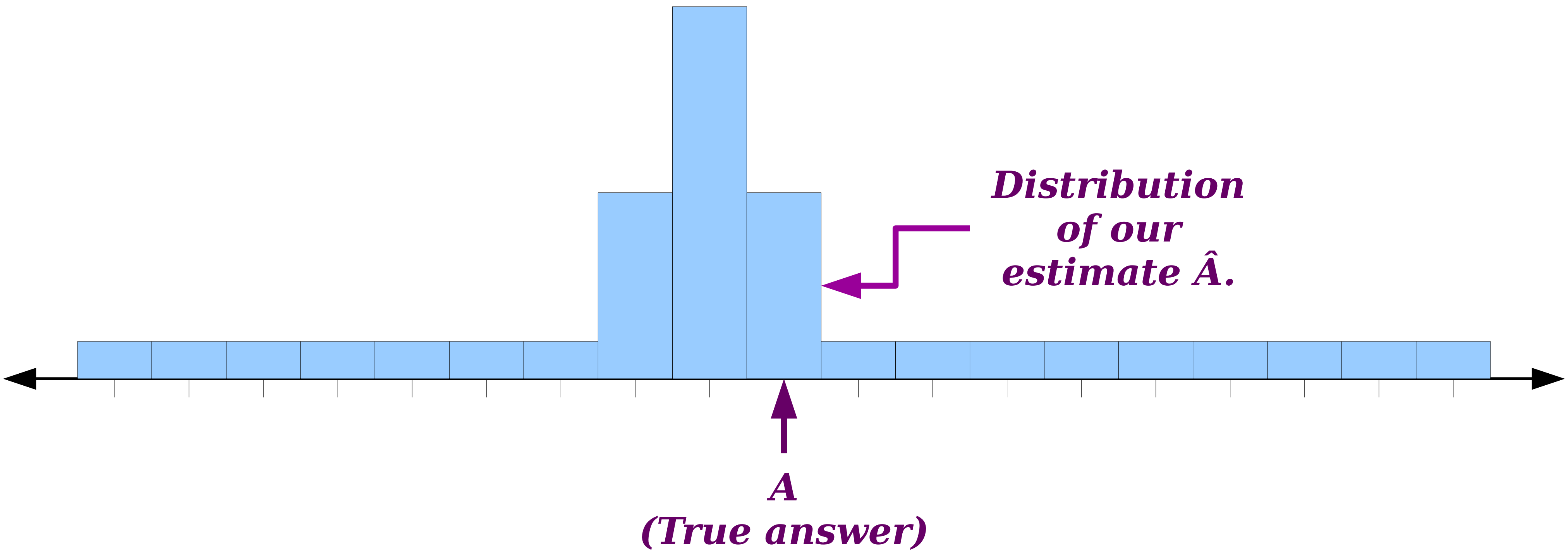


What does it mean for an approximation to be "good"?

Let $\boldsymbol{A}$ be the true answer. Let $\hat{\boldsymbol{A}}$ be a random variable denoting our estimate.

This estimate skews low, but it's very close to the true value.

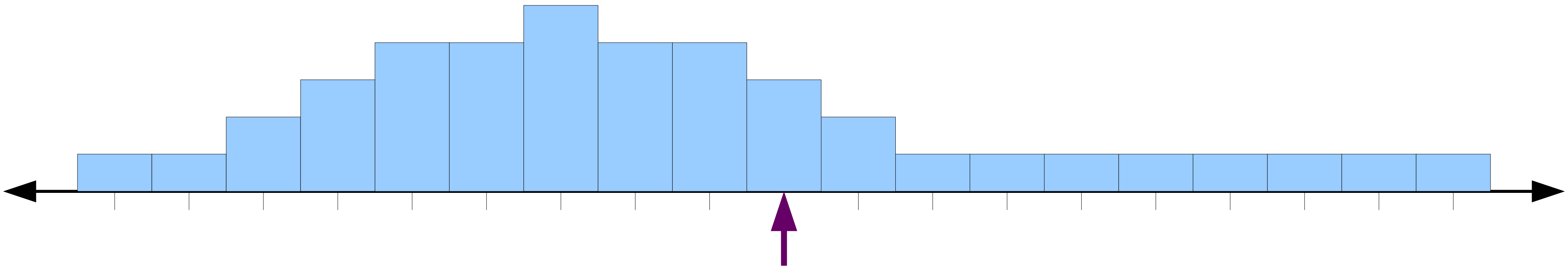
Observation 3: An estimate doesn't have to be unbiased to be useful.



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

Let A be the true answer. Let $\hat{A}$ be a random variable denoting our estimate.

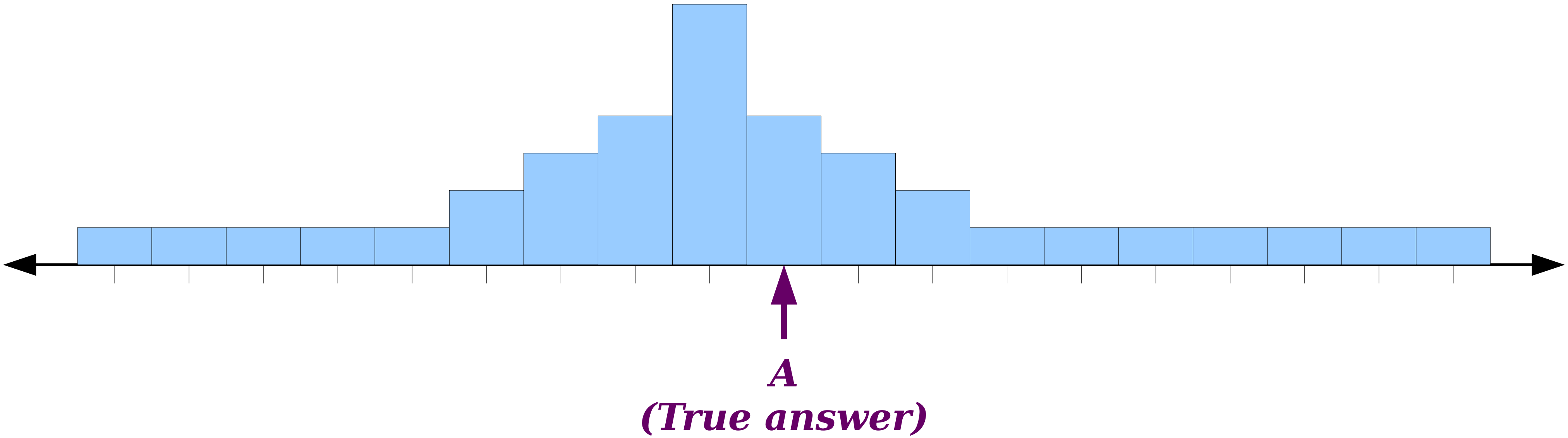
Memory used: 16MB



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

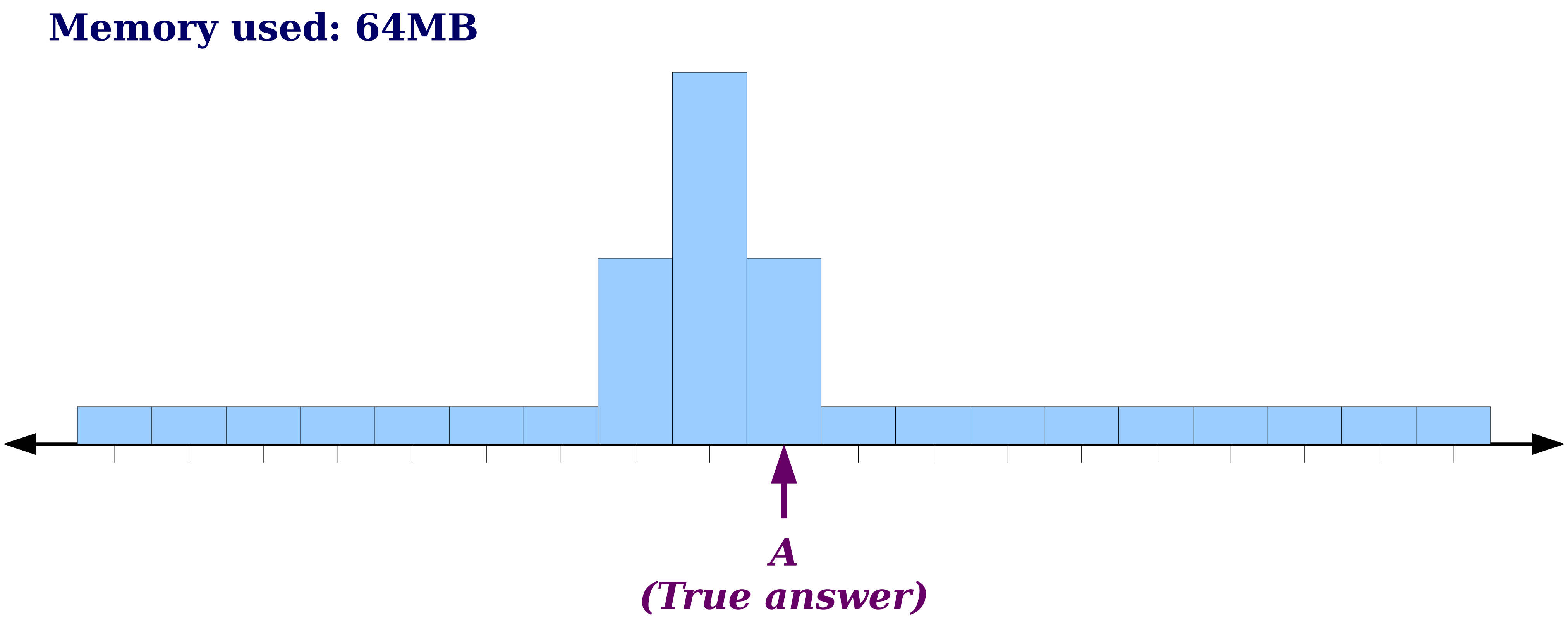
Let $\boldsymbol{A}$ be the true answer. Let $\boldsymbol{\hat{A}}$ be a random variable denoting our estimate.

Memory used: 32MB



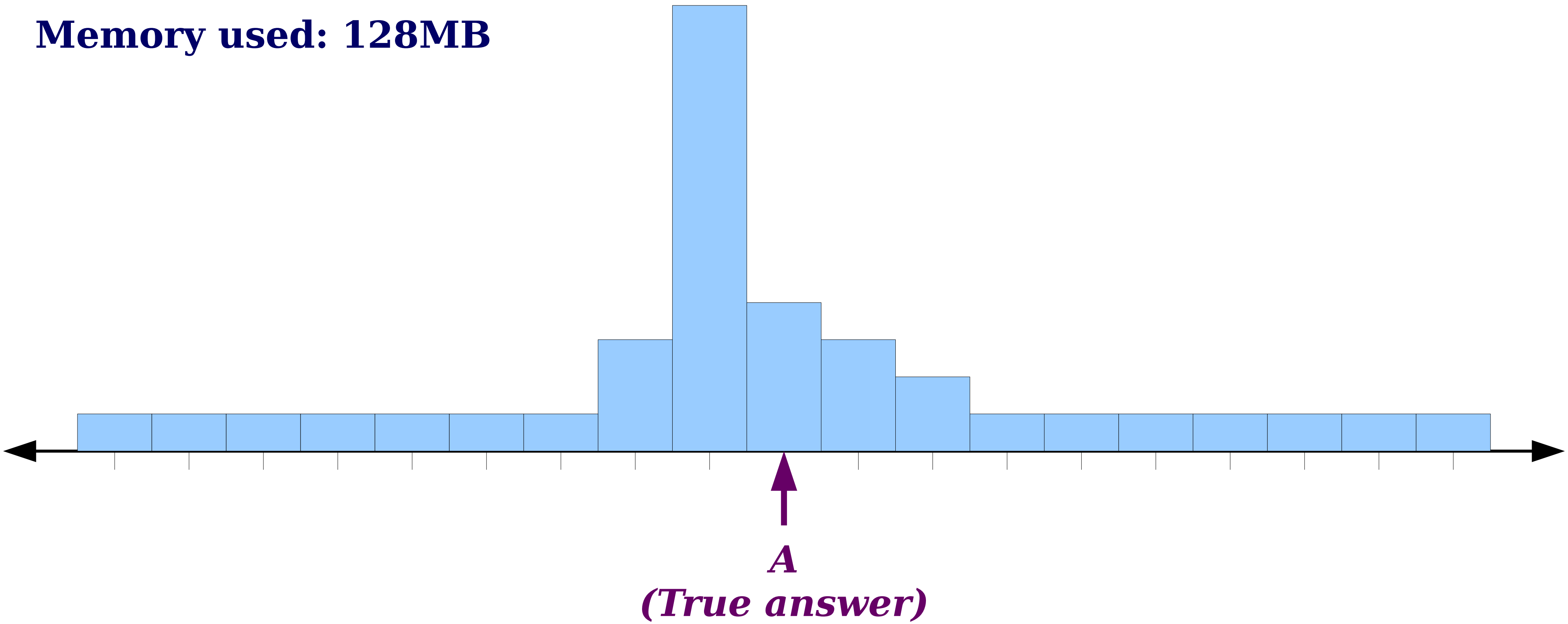
What does it mean for an approximation to be "good"?

Let A be the true answer. Let $\hat{A}$ be a random variable denoting our estimate.



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

Let A be the true answer. Let $\hat{A}$ be a random variable denoting our estimate.

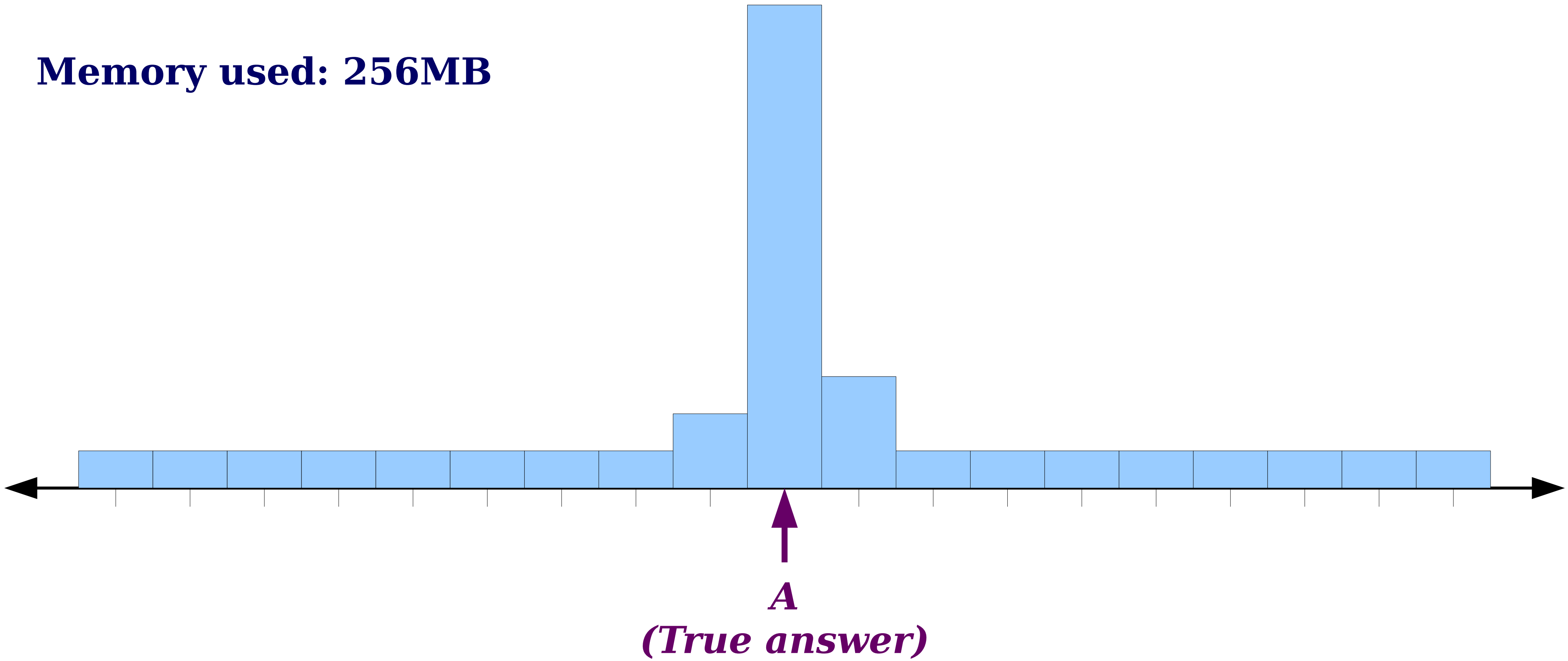


What does it mean for an approximation to be “good”?

Let A be the true answer. Let $\hat{A}$ be a random variable denoting our estimate.

The more resources we allocate, the better our estimate should be.

Observation 4: A good approximation should be tunable.



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

We have two user-provided 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 $1 - \delta$,

$$|\hat{A} - A| \leq \varepsilon \cdot size(input)$$

for some measure of the size of the input.

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

We have two user-provided 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***

$$|\hat{A} - A| \leq \varepsilon \cdot \mathit{size}(\mathit{input})$$

for some measure of the size of the input.

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

We have two user-provided 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"?

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

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

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

for some measure of the size of the input.

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

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

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

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

for some measure of the size of the input.

$\delta = \frac{1}{2}$

ε small

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

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

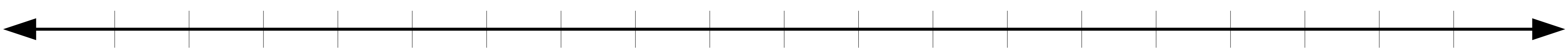
With probability at least $1 - \delta$, ***Probably***

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

for some measure of the size of the input.

$\delta = ½$

ε small



What does it mean for an approximation to be “good”?

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

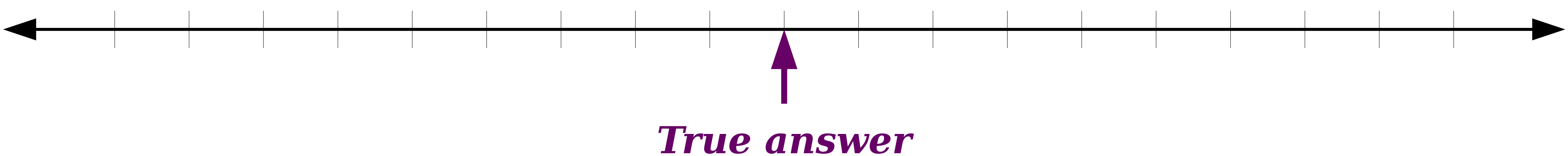
With probability at least $1 - \delta$, — ***Probably***

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

for some measure of the size of the input.

$\delta = ½$

ε small



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

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

With probability at least $1 - \delta$,

$|A - \hat{A}| \leq \varepsilon \cdot size(input)$

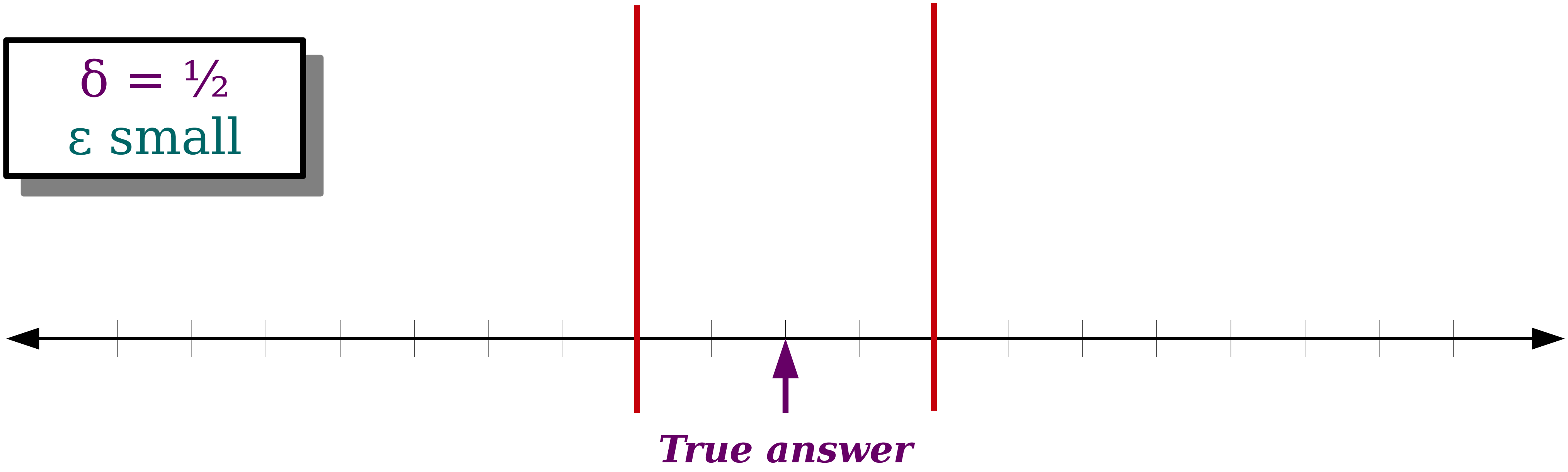
for some measure of the size of the input.

Probably

Approximately Correct

$\delta = ½$

ε small



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

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

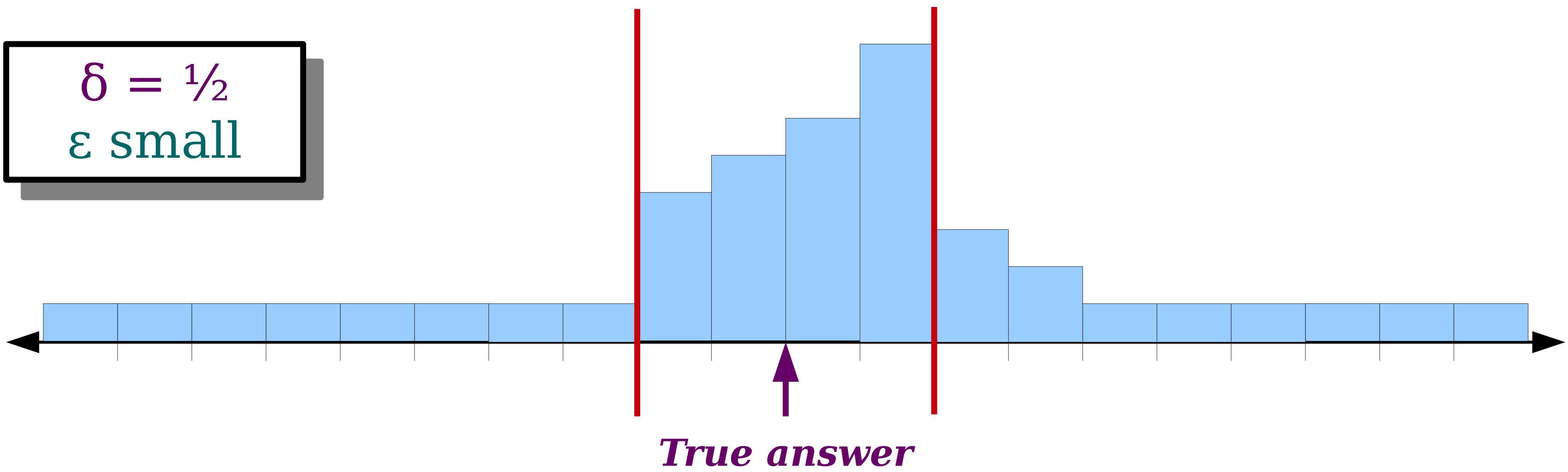
With probability at least $1 - \delta$, — ***Probably***

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

for some measure of the size of the input.

$\delta = ½$

ε small



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

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

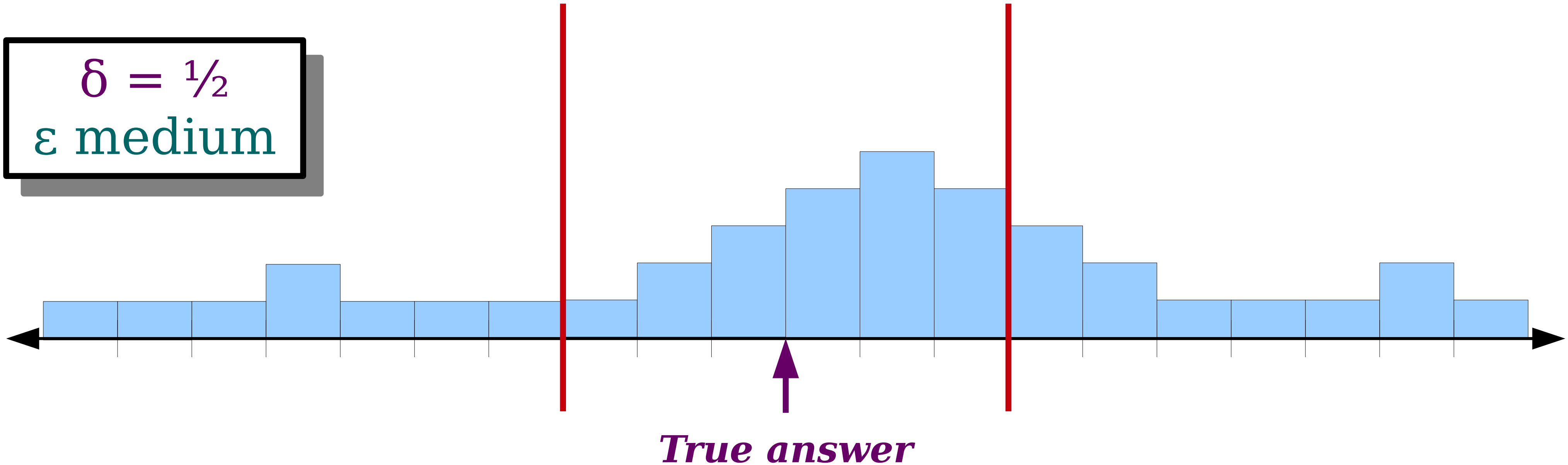
With probability at least $1 - \delta$, — ***Probably***

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

for some measure of the size of the input.

$\delta = ½$

ε medium



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

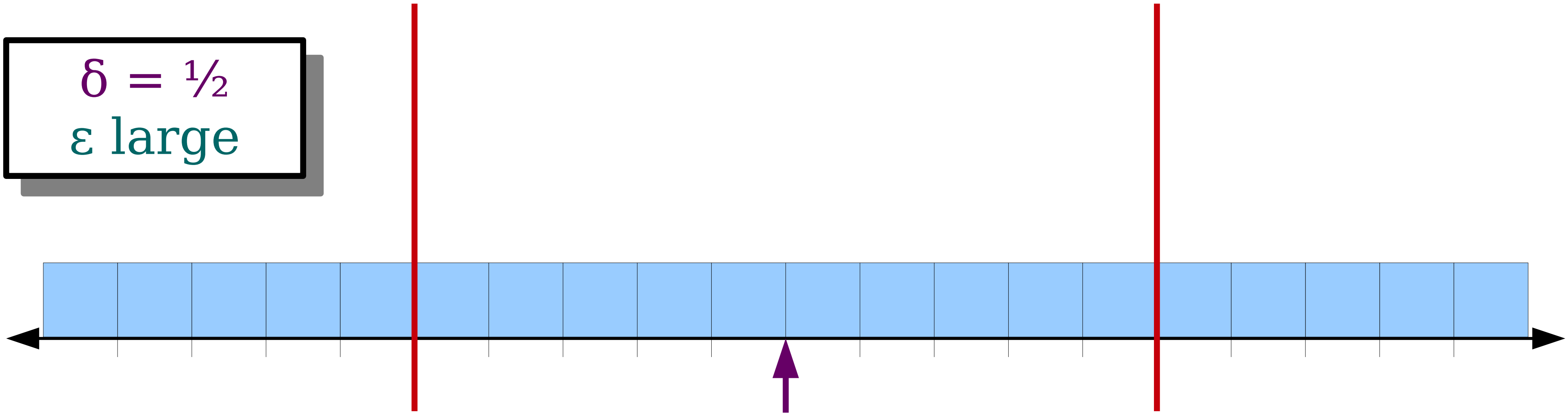
Goal: Make an estimator $\hat{A}$ for some quantity A where

With probability at least 1 – δ, — ***Probably***

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

for some measure of the size of the input.

δ = ½
ε large



What does it mean for an approximation to be “good”?

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

With probability at least $1 - \delta$,

$|A - \hat{A}| \leq \varepsilon \cdot size(input)$

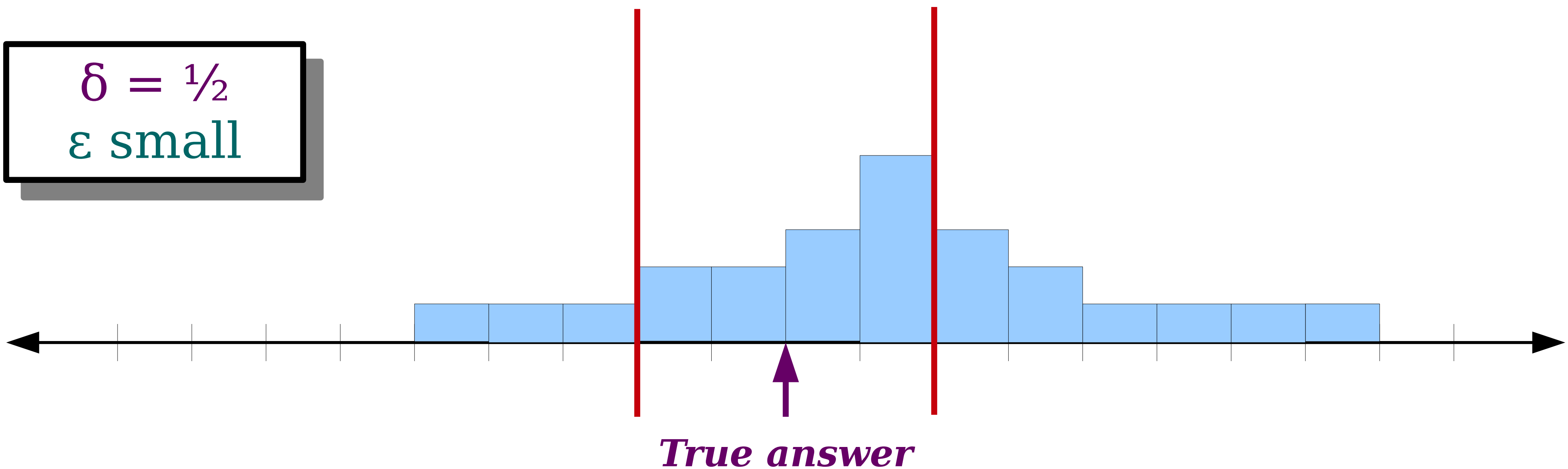
for some measure of the size of the input.

Probably

Approximately Correct

$\delta = ½$

ε small



What does it mean for an approximation to be “good”?

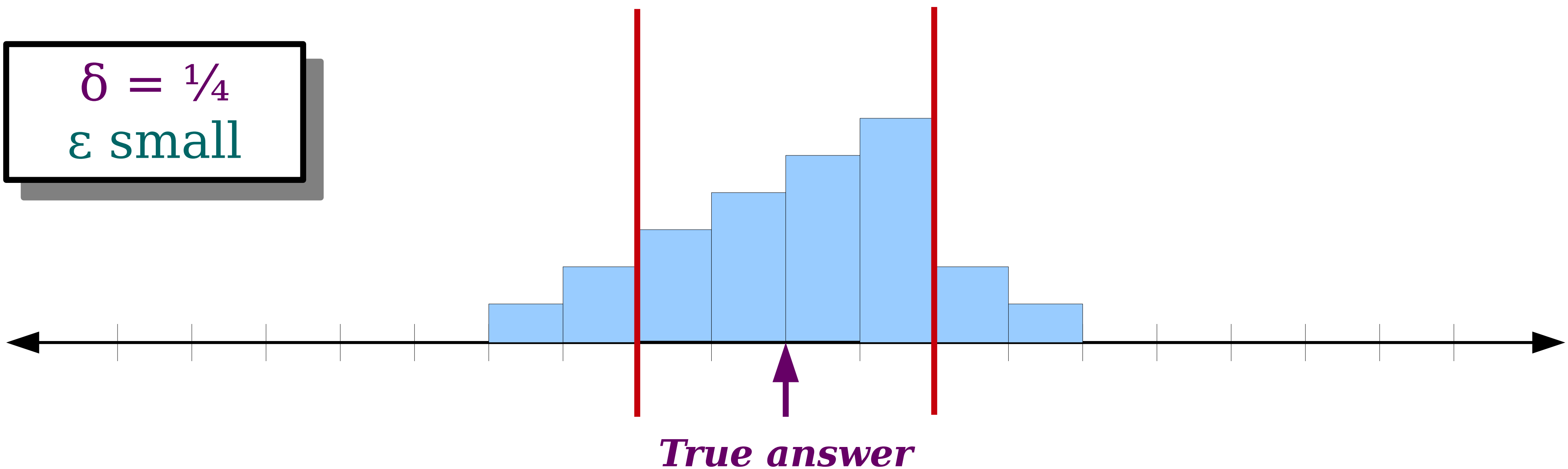
Goal: Make an estimator $\hat{A}$ for some quantity A where

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

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

for some measure of the size of the input.

$\delta = ¼$
ε small



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

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

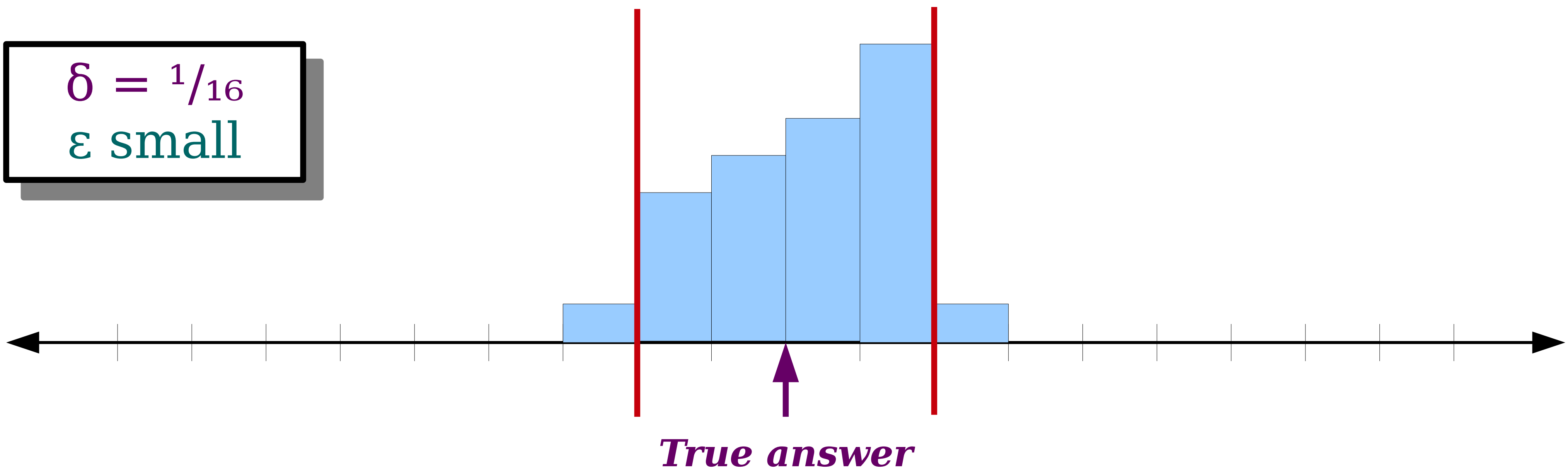
With probability at least $1 - \delta$, — ***Probably***

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

for some measure of the size of the input.

$\delta = 1/16$

ε small



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

Frequency Estimation

Frequency Estimators

- A ***frequency estimator*** is a data structure supporting the following operations:
  - ***increment***(x), which increments the number of times that x has been seen, and
  - ***estimate***(x), which returns an estimate of the frequency of x.
- Using BSTs, we can solve this in space $\Theta(n)$ with worst-case $O(\log n)$ costs on the operations.
- Using hash tables, we can solve this in space $\Theta(n)$ with expected $O(1)$ costs on the operations.

Frequency Estimators

- Frequency estimation has many applications:
  - Search engines: Finding frequent search queries.
  - Network routing: Finding common source and destination addresses.
- In these applications, $\Theta(n)$ memory can be impractical.
- ***Goal:*** Get *approximate* answers to these queries in sublinear space.

The Count-Min Sketch

How to Build an Estimator

1. Design a simple data structure that, intuitively, gives you a good estimate.

2. Use a ***sum of indicator variables*** and ***linearity of expectation*** to prove that, on expectation, the data structure is pretty close to correct.

3. Use a ***concentration inequality*** to show that, with decent probability, the data structure's output is close to its expectation.

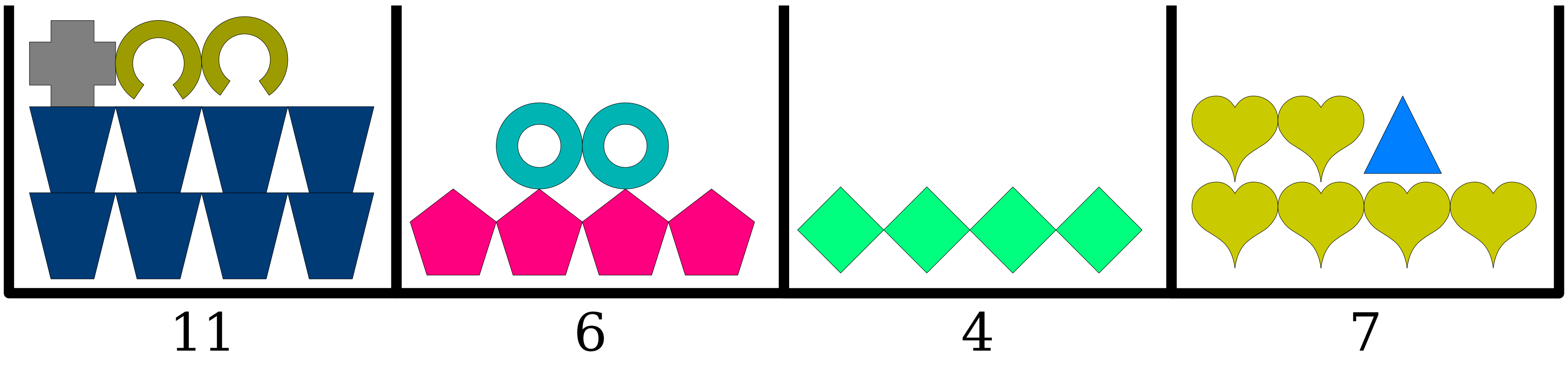
4. Run multiple copies of the data structure in parallel to amplify the success probability.

How to Build an Estimator

1. Design a simple data structure that, intuitively, gives you a good estimate.
2. Use a ***sum of indicator variables*** and ***linearity of expectation*** to prove that, on expectation, the data structure is pretty close to correct.
3. Use a ***concentration inequality*** to show that, with decent probability, the data structure's output is close to its expectation.
4. Run multiple copies of the data structure in parallel to amplify the success probability.

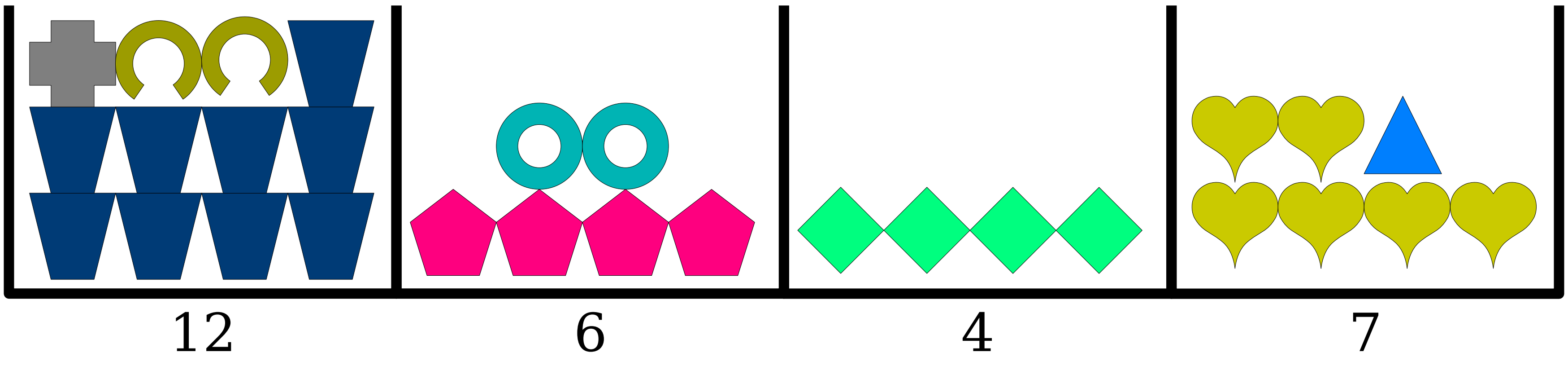
Revisiting the Exact Solution

- In the exact solution to the frequency estimation problem, we maintained a single counter for each distinct element. This is too space-inefficient.
- ***Idea:*** Store a fixed number of counters and assign a counter to each $x_i \in \mathscr{U}$. Multiple x_i's might be assigned to the same counter.
- To ***increment***(x), increment the counter for x.
- To ***estimate***(x), read the value of the counter for x.



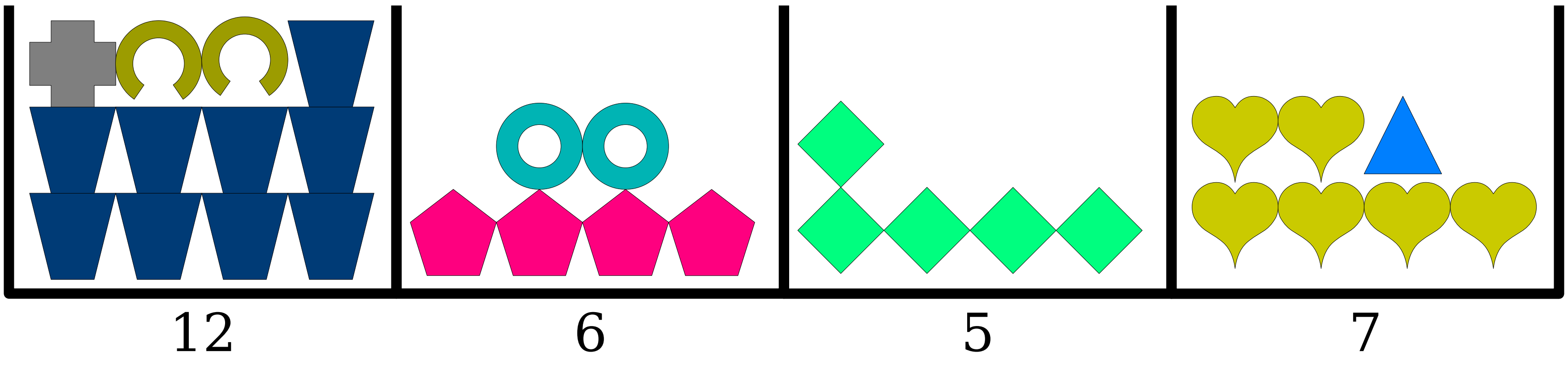
Revisiting the Exact Solution

- In the exact solution to the frequency estimation problem, we maintained a single counter for each distinct element. This is too space-inefficient.
- ***Idea:*** Store a fixed number of counters and assign a counter to each $x_i \in \mathscr{U}$. Multiple x_i's might be assigned to the same counter.
- To ***increment***(x), increment the counter for x.
- To ***estimate***(x), read the value of the counter for x.



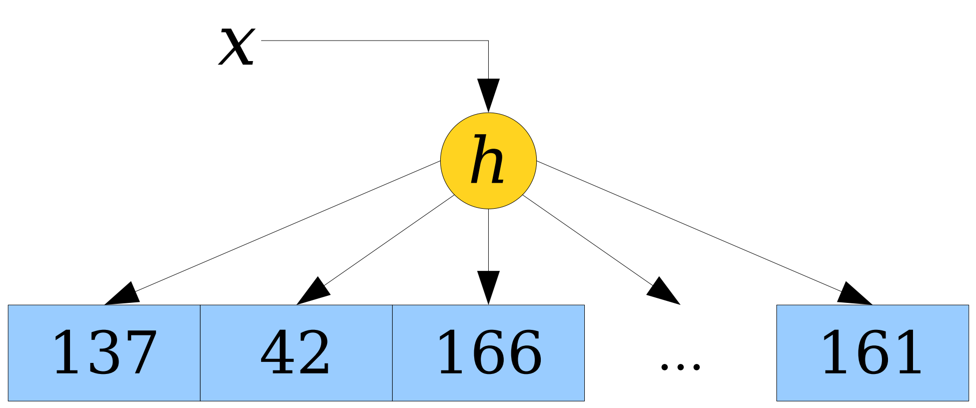
Revisiting the Exact Solution

- In the exact solution to the frequency estimation problem, we maintained a single counter for each distinct element. This is too space-inefficient.
- ***Idea:*** Store a fixed number of counters and assign a counter to each $x_i \in \mathscr{U}$. Multiple x_i's might be assigned to the same counter.
- To ***increment***(x), increment the counter for x.
- To ***estimate***(x), read the value of the counter for x.



Our Initial Structure

- We can model "assigning each x_i to a counter" by using hash functions.
- Choose, from a family of 2-independent hash functions $\mathscr{H}$, a uniformly-random hash function $h : \mathscr{U} \to [w]$.
- Create an array **count** of w counters, each initially zero.
  - We'll choose w later on.
- To ***increment***(x), increment **count**$[h(x)]$.
- To ***estimate***(x), return **count**$[h(x)]$.



Analyzing our Structure

For each $x_i \in \mathscr{U}$, let $\boldsymbol{a}_i$ denote the number of times we've seen x_i.

Similarly, let $\hat{\boldsymbol{a}}_i$ denote our estimated value of the frequency of x_i.

Goal: Bound the probability that the error $(\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i)$ is too high.

Idea: Think of our element frequencies $\boldsymbol{a}_1$, $\boldsymbol{a}_2$, $\boldsymbol{a}_3$, … as a vector

$$\boldsymbol{a} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3, \ldots].$$

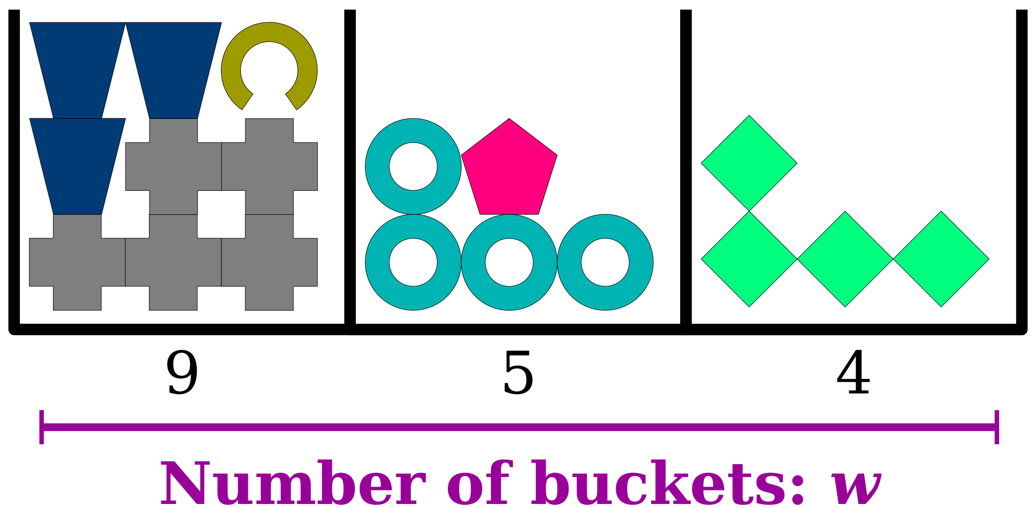
The total number of objects is the sum of the vector entries.

This is called the ***L_1 norm*** of $\boldsymbol{a}$, and is denoted $\|\boldsymbol{a}\|_1$:

$$\|\boldsymbol{a}\|_1 = \sum_i |\boldsymbol{a}_i|$$

There are $\|\boldsymbol{a}\|_1$ total elements distributed across w buckets. We're using a 2-independent hash family.

Reasonable guess: each bin has $\|\boldsymbol{a}\|_1 / w$ elements in it, so $\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i \le \|\boldsymbol{a}\|_1 / w$



Question: Intuitively, what should we expect our approximation error to be?

How to Build an Estimator

1. Design a simple data structure that, intuitively, gives you a good estimate.
2. Use a ***sum of indicator variables*** and ***linearity of expectation*** to prove that, on expectation, the data structure is pretty close to correct.
3. Use a ***concentration inequality*** to show that, with decent probability, the data structure's output is close to its expectation.
4. Run multiple copies of the data structure in parallel to amplify the success probability.

How to Build an Estimator

1. Design a simple data structure that, intuitively, gives you a good estimate.
2. Use a ***sum of indicator variables*** and ***linearity of expectation*** to prove that, on expectation, the data structure is pretty close to correct.
3. Use a ***concentration inequality*** to show that, with decent probability, the data structure's output is close to its expectation.
4. Run multiple copies of the data structure in parallel to amplify the success probability.

Analyzing this Structure

- Let's look at $\hat{\boldsymbol{a}}_i$ = `count`[$h(x_i)$] for some choice of x_i.
- For each element x_j:
  - If $h(x_i) = h(x_j)$, then x_j contributes $\boldsymbol{a}_j$ to `count`[$h(x_i)$].
  - If $h(x_i) \neq h(x_j)$, then x_j contributes 0 to `count`[$h(x_i)$].

Analyzing this Structure

- Let's look at $\hat{\boldsymbol{a}}_i$ = `count`[$h(x_i)$] for some choice of x_i.
- For each element x_j:
  - If $h(x_i) = h(x_j)$, then x_j contributes $\boldsymbol{a}_j$ to `count`[$h(x_i)$].
  - If $h(x_i) \neq h(x_j)$, then x_j contributes 0 to `count`[$h(x_i)$].
- To pin this down precisely, let's define a set of random variables $X_1, X_2, \ldots,$ as follows:

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

Analyzing this Structure

- Let's look at $\hat{\boldsymbol{a}}_i$ = `count`$[h(x_i)]$ for some choice of x_i.
- For each element x_j:
  - If $h(x_i) = h(x_j)$, then x_j contributes $\boldsymbol{a}_j$ to `count`$[h(x_i)]$.
  - If $h(x_i) \neq h(x_j)$, then x_j contributes 0 to `count`$[h(x_i)]$.
- To pin this down precisely, let's define a set of random variables X_1, X_2, …, as follows:

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

Each of these variables is called an ***indicator random variable***, since it "indicates" whether some event occurs.

Analyzing this Structure

- Let's look at $\hat{\boldsymbol{a}}_i$ = `count`[$h(x_i)$] for some choice of x_i.
- For each element x_j:
  - If $h(x_i) = h(x_j)$, then x_j contributes $\boldsymbol{a}_j$ to `count`[$h(x_i)$].
  - If $h(x_i) \neq h(x_j)$, then x_j contributes 0 to `count`[$h(x_i)$].
- To pin this down precisely, let's define a set of random variables $X_1, X_2, \ldots,$ as follows:

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

- The value of $\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i$ is then given by

$$\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i = \sum_{j \neq i} \boldsymbol{a}_j X_j$$

$$\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] = \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j X_j]$$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j]
\end{aligned}
$$

This follows from ***linearity of expectation***. We'll use this property extensively over the next few days.

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j]
\end{aligned}
$$

The values of $\boldsymbol{a}_j$ are not random. ***The randomness comes from our choice of hash function.***

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j]
\end{aligned}
$$

$$
\begin{aligned}
E[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= E[\sum_{j \neq i} \boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} E[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j E[X_j]
\end{aligned}
$$

$$E[X_j] \quad =$$

$$\begin{aligned} \mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\ &= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\ &= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j] \end{aligned}$$

$$\mathrm{E}[X_j] =$$

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

$$
\begin{aligned}
E[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= E\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\
&= \sum_{j \neq i} E[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j E[X_j]
\end{aligned}
$$

$$E[X_j] = 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)]$$

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

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j]
\end{aligned}
$$

$$
\mathrm{E}[X_j] = 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)]
$$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[X_j] &= 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \\
&= \Pr[h(x_i) = h(x_j)]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[X_j] &= 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \\
&= \Pr[h(x_i) = h(x_j)]
\end{aligned}
$$

If X is an indicator variable for some event $\mathcal{E}$, then **E[X] = Pr[$\mathcal{E}$]**. This is really useful when using linearity of expectation!

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[X_j] &= 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \\
&= \Pr[h(x_i) = h(x_j)]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[X_j] &= 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \\
&= \Pr[h(x_i) = h(x_j)]
\end{aligned}
$$

Hey, we saw this earlier!

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j]
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[X_j] &= 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \\
&= \Pr[h(x_i) = h(x_j)] \\
&= \frac{1}{w}
\end{aligned}
$$

Hey, we saw this earlier!

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j] \\
&= \sum_{j \neq i} \frac{\boldsymbol{a}_j}{w}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[X_j] &= 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \\
&= \Pr[h(x_i) = h(x_j)] \\
&= \frac{1}{w}
\end{aligned}
$$

Hey, we saw this earlier!

$$
\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}\Big[\sum_{j \neq i} \boldsymbol{a}_j X_j\Big] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j] \\
&= \sum_{j \neq i} \frac{\boldsymbol{a}_j}{w}
\end{aligned}
$$

$$
\begin{aligned}
\mathrm{E}[X_j] &= 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \\
&= \Pr[h(x_i) = h(x_j)] \\
&= \frac{1}{w}
\end{aligned}
$$

$$\begin{aligned}
\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] &= \mathrm{E}[\sum_{j \neq i} \boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \mathrm{E}[\boldsymbol{a}_j X_j] \\
&= \sum_{j \neq i} \boldsymbol{a}_j \mathrm{E}[X_j] \\
&= \sum_{j \neq i} \frac{\boldsymbol{a}_j}{w} \\
&\leq \frac{\|\boldsymbol{a}\|_1}{w}
\end{aligned}$$

$$\begin{aligned}
\mathrm{E}[X_j] &= 1 \cdot \Pr[h(x_i) = h(x_j)] + 0 \cdot \Pr[h(x_i) \neq h(x_j)] \\
&= \Pr[h(x_i) = h(x_j)] \\
&= \frac{1}{w}
\end{aligned}$$

How to Build an Estimator

1. Design a simple data structure that, intuitively, gives you a good estimate.
2. Use a ***sum of indicator variables*** and ***linearity of expectation*** to prove that, on expectation, the data structure is pretty close to correct.
3. Use a ***concentration inequality*** to show that, with decent probability, the data structure's output is close to its expectation.
4. Run multiple copies of the data structure in parallel to amplify the success probability.

How to Build an Estimator

1. Design a simple data structure that, intuitively, gives you a good estimate.

2. Use a ***sum of indicator variables*** and ***linearity of expectation*** to prove that, on expectation, the data structure is pretty close to correct.

3. Use a ***concentration inequality*** to show that, with decent probability, the data structure's output is close to its expectation.

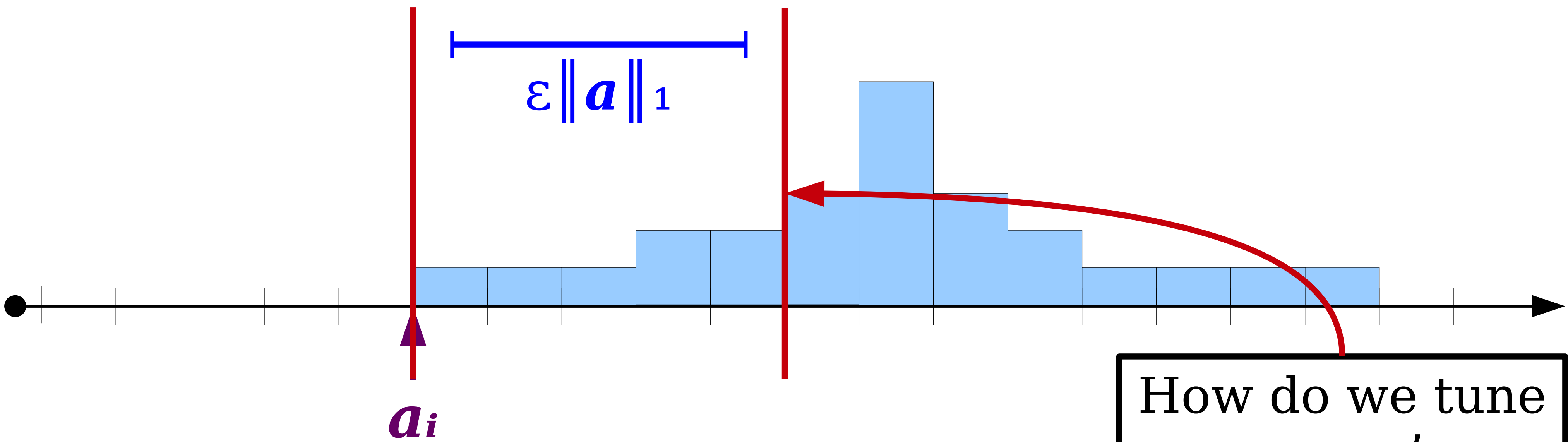
4. Run multiple copies of the data structure in parallel to amplify the success probability.

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

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

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

for some measure of the size of the input.



How do we tune w so we're likely to fall in this range?

$$\mathrm{E}[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i] \le \frac{\|\boldsymbol{a}\|_1}{w}$$

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \left\|\boldsymbol{a}\right\|_1\right]$$

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \left\|\boldsymbol{a}\right\|_1\right]$$

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \left\|\boldsymbol{a}\right\|_1\right]$$

We don't know the exact distribution of this random variable.

However, we have a ***one-sided error***: our estimate can never be lower than the true value. This means that $\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i \geq 0$.

Markov's inequality says that if X is a nonnegative random variable, then

$$\Pr[X \geq c] \leq \frac{\mathrm{E}[X]}{c}.$$

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

$$\leq \frac{\mathrm{E}\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right]}{\varepsilon \|\boldsymbol{a}\|_1}$$

We don't know the exact distribution of this random variable.

However, we have a ***one-sided error***: our estimate can never be lower than the true value. This means that $\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i \geq 0$.

Markov's inequality says that if X is a nonnegative random variable, then

$$\Pr[X \geq c] \leq \frac{\mathrm{E}[X]}{c}.$$

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \left\|\boldsymbol{a}\right\|_1\right]$$

$$\leq \frac{\mathrm{E}\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right]}{\varepsilon \left\|\boldsymbol{a}\right\|_1}$$

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \left\|\boldsymbol{a}\right\|_1\right]$$

$$\leq \frac{\mathrm{E}\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right]}{\varepsilon \left\|\boldsymbol{a}\right\|_1}$$

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \left\|\boldsymbol{a}\right\|_1\right]$$

$$\leq \frac{\mathrm{E}\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right]}{\varepsilon \left\|\boldsymbol{a}\right\|_1}$$

$$\boxed{\mathrm{E}\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right] \leq \frac{\left\|\boldsymbol{a}\right\|_1}{w}}$$

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \left\|\boldsymbol{a}\right\|_1\right]$$

$$\leq \frac{\mathrm{E}\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right]}{\varepsilon \left\|\boldsymbol{a}\right\|_1}$$

$$\leq \frac{\left\|\boldsymbol{a}\right\|_1}{w} \cdot \frac{1}{\varepsilon \left\|\boldsymbol{a}\right\|_1}$$

$$\boxed{\mathrm{E}\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right] \leq \frac{\left\|\boldsymbol{a}\right\|_1}{w}}$$

$$
\begin{aligned}
& \Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i \;>\; \varepsilon \left\|\boldsymbol{a}\right\|_1\right] \\
\leq\quad & \frac{\mathrm{E}\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right]}{\varepsilon \left\|\boldsymbol{a}\right\|_1} \\
\leq\quad & \frac{\left\|\boldsymbol{a}\right\|_1}{w} \cdot \frac{1}{\varepsilon \left\|\boldsymbol{a}\right\|_1}
\end{aligned}
$$

$$
\begin{aligned}
& \Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i \;\; > \;\; \varepsilon \left\|\boldsymbol{a}\right\|_1\right] \\
\leq \quad & \frac{\mathrm{E}\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i\right]}{\varepsilon \left\|\boldsymbol{a}\right\|_1} \\
\leq \quad & \frac{\left\|\boldsymbol{a}\right\|_1}{w} \cdot \frac{1}{\varepsilon \left\|\boldsymbol{a}\right\|_1} \\
= \quad & \frac{1}{\varepsilon\, w}
\end{aligned}
$$

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

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

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

for some measure of input size.

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right] \leq \frac{1}{\varepsilon w}$$

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

With probability at least 1 – δ, **Probably**

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

for some measure of input size.

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right] \leq \frac{1}{\varepsilon w}$$

Initial Idea:

Pick $w = \varepsilon^{-1} \cdot \delta^{-1}$. Then

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right] \leq \delta$$

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

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

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

for some measure of input size.

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \left\|\boldsymbol{a}\right\|_1\right] \leq \frac{1}{\varepsilon w}$$

Initial Idea:

Pick $w = \varepsilon^{-1} \cdot \delta^{-1}$. Then

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \left\|\boldsymbol{a}\right\|_1\right] \leq \delta$$

Suppose we're counting 1,000 distinct items.

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

With probability at least 1 – δ, — ***Probably***

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

for some measure of input size.

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right] \leq \frac{1}{\varepsilon w}$$

Initial Idea:

Pick $w = \varepsilon^{-1} \cdot \delta^{-1}$. Then

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right] \leq \delta$$

Suppose we're counting 1,000 distinct items.

If we want our estimate to be within $\varepsilon\|\boldsymbol{a}\|_1$ of the true value with 99.9% probability, how much memory do we need?

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

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

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

for some measure of input size.

$$\Pr[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1] \leq \frac{1}{\varepsilon w}$$

Initial Idea:

Pick $w = \varepsilon^{-1} \cdot \delta^{-1}$. Then

$$\Pr[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1] \leq \delta$$

Suppose we're counting 1,000 distinct items.

If we want our estimate to be within $\varepsilon\|\boldsymbol{a}\|_1$ of the true value with 99.9% probability, how much memory do we need?

Answer: **$1{,}000 \cdot \varepsilon^{-1}$**.

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

$$\left.\text{With probability at least } 1 - \delta,\right\} \leftarrow \textbf{\textit{Probably}}$$

$$\left.|\boldsymbol{\hat{a}} - \boldsymbol{a}| \leq \varepsilon \cdot \mathit{size}(\mathit{input})\right\} \leftarrow \textbf{\textit{Approximately Correct}}$$

for some measure of input size.

$$\Pr[\boldsymbol{\hat{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1] \leq \frac{1}{\varepsilon w}$$

Initial Idea:

Pick $w = \varepsilon^{-1} \cdot \delta^{-1}$. Then

$$\Pr[\boldsymbol{\hat{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1] \leq \delta$$

Suppose we're counting 1,000 distinct items.

If we want our estimate to be within $\varepsilon\|\boldsymbol{a}\|_1$ of the true value with 99.9% probability, how much memory do we need?

Answer: **$1{,}000 \cdot \varepsilon^{-1}$**.

Can we do better?

How to Build an Estimator

1. Design a simple data structure that, intuitively, gives you a good estimate.
2. Use a ***sum of indicator variables*** and ***linearity of expectation*** to prove that, on expectation, the data structure is pretty close to correct.
3. Use a ***concentration inequality*** to show that the data structure's output is close to its expectation.
4. Run multiple copies of the data structure in parallel to amplify the success probability.

How to Build an Estimator

1. Design a simple data structure that, intuitively, gives you a good estimate.
2. Use a ***sum of indicator variables*** and ***linearity of expectation*** to prove that, on expectation, the data structure is pretty close to correct.
3. Use a ***concentration inequality*** to show that the data structure's output is close to its expectation.
4. Run multiple copies of the data structure in parallel to amplify the success probability.

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

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

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

for some measure of input size.

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right] \leq \frac{1}{\varepsilon w}$$

We could choose $w = k \cdot \varepsilon^{-1}$ for any constant k to get a failure probability of at most k^{-1}. The choice of e is (mostly) arbitrary.

Revised Idea: Pick $w = e \cdot \varepsilon^{-1}$. Then

$$\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right] < e^{-1}$$

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

With probability at least 1 – δ, $\big\}$ ***Probably***

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

for some measure of input size.

$$\Pr[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1] \leq \frac{1}{\varepsilon w}$$

Revised Idea: Pick $w = e \cdot \varepsilon^{-1}$. Then

$$\Pr[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1] < e^{-1}$$

This simple data structure, by itself, is likely to be wrong.

What happens if we run a bunch of copies of this approach in parallel?

Running in Parallel

- Let's run ***d*** copies of our data structure in parallel with one another.
- Each row has its hash function sampled uniformly at random from our hash family.
- Each time we ***increment*** an item, we perform the corresponding ***increment*** operation on each row.

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

h_1	31	41	59	26	53	…	58
h_2	27	18	28	18	28	…	45
h_3	16	18	3	39	88	…	75
…				…			
h_d	69	31	47	18	5	…	59

$d = ??$

Running in Parallel

- Let's run ***d*** copies of our data structure in parallel with one another.
- Each row has its hash function sampled uniformly at random from our hash family.
- Each time we ***increment*** an item, we perform the corresponding ***increment*** operation on each row.

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

h_1	31	41	59	26	53	…	58
h_2	27	18	28	18	28	…	45
h_3	16	18	3	39	88	…	75
…				…			
h_d	69	31	47	18	5	…	59

$d = ??$

Running in Parallel

- Let's run ***d*** copies of our data structure in parallel with one another.
- Each row has its hash function sampled uniformly at random from our hash family.
- Each time we ***increment*** an item, we perform the corresponding ***increment*** operation on each row.

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

$d = ??$

h_1	32	41	59	26	53	…	58
h_2	27	18	29	18	28	…	45
h_3	16	18	3	40	88	…	75
…				…			
h_d	69	31	47	18	5	…	60

Running in Parallel

- Let's run ***d*** copies of our data structure in parallel with one another.
- Each row has its hash function sampled uniformly at random from our hash family.
- Each time we ***increment*** an item, we perform the corresponding ***increment*** operation on each row.

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

h_1	32	41	59	26	53	…	58
h_2	27	18	29	18	28	…	45
h_3	16	18	3	40	88	…	75
…				…			
h_d	69	31	47	18	5	…	60

d = ??

Running in Parallel

- Imagine we call ***estimate***(x) on each of our estimators and get back these estimates.
- We need to give back a single number.
- ***Question:*** How should we aggregate these numbers into a single estimate?

Formulate a hypothesis, but ***don't post anything in chat just yet***.

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

Running in Parallel

- Imagine we call ***estimate***(x) on each of our estimators and get back these estimates.
- We need to give back a single number.
- ***Question:*** How should we aggregate these numbers into a single estimate?

Now, ***private chat me your best guess***. Not sure? Just answer “??”.

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

Running in Parallel

- Imagine we call ***estimate***(x) on each of our estimators and get back these estimates.
- We need to give back a single number.
- ***Question:*** How should we aggregate these numbers into a single estimate?

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

Running in Parallel

- Imagine we call ***estimate***(x) on each of our estimators and get back these estimates.
- We need to give back a single number.
- ***Question:*** How should we aggregate these numbers into a single estimate?

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

Running in Parallel

- Imagine we call ***estimate***(x) on each of our estimators and get back these estimates.
- We need to give back a single number.
- ***Question:*** How should we aggr into a single estimate?

Intuition: The smallest estimate returned has the least "noise," and that's the best guess for the frequency.

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

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is $\min \{\hat{\boldsymbol{a}}_{ij}\}$

$$\Pr\left[\min\{\hat{\boldsymbol{a}}_{ij}\} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is $\min\{\hat{\boldsymbol{a}}_{ij}\}$

$$\Pr\left[\min\left\{\hat{\boldsymbol{a}}_{ij}\right\} - \boldsymbol{a}_i > \varepsilon \left\|\boldsymbol{a}\right\|_1\right]$$

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is $\min\{\hat{\boldsymbol{a}}_{ij}\}$

$$\Pr\left[\min\left\{\hat{\boldsymbol{a}}_{ij}\right\} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

The only way the minimum estimate is inaccurate is if *every* estimate is inaccurate.

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is $\min\{\hat{\boldsymbol{a}}_{ij}\}$

$$\Pr\left[\min\{\hat{\boldsymbol{a}}_{ij}\} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

$$= \Pr\left[\bigwedge_{j=1}^{d} \left(\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right)\right]$$

The only way the minimum estimate is inaccurate is if *every* estimate is inaccurate.

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is $\min\{\hat{\boldsymbol{a}}_{ij}\}$

$$\Pr\left[\min\left\{\,\hat{\boldsymbol{a}}_{ij}\,\right\} - \boldsymbol{a}_i \;>\; \varepsilon\,\|\boldsymbol{a}\|_1\right]$$

$$= \quad \Pr\left[\bigwedge_{j=1}^{d}\left(\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i \;>\; \varepsilon\,\|\boldsymbol{a}\|_1\right)\right]$$

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is $\min\,\{\hat{\boldsymbol{a}}_{ij}\}$

$$\Pr\left[\min\{\hat{\boldsymbol{a}}_{ij}\} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

$$= \Pr\left[\bigwedge_{j=1}^{d} \left(\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right)\right]$$

Each copy of the data structure is independent of the others.

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is $\min\{\hat{\boldsymbol{a}}_{ij}\}$

$$\Pr\left[\min\{\hat{\boldsymbol{a}}_{ij}\} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

$$= \Pr\left[\bigwedge_{j=1}^{d} \left(\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right)\right]$$

$$= \prod_{j=1}^{d} \Pr\left[\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

Each copy of the data structure is independent of the others.

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is min $\{\hat{\boldsymbol{a}}_{ij}\}$

$$\Pr\left[\min\{\hat{\boldsymbol{a}}_{ij}\} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

$$= \Pr\left[\bigwedge_{j=1}^{d} \left(\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right)\right]$$

$$= \prod_{j=1}^{d} \Pr\left[\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is $\min\{\hat{\boldsymbol{a}}_{ij}\}$

$$\Pr\left[\min\{\hat{\boldsymbol{a}}_{ij}\} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

$$= \Pr\left[\bigwedge_{j=1}^{d} \left(\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right)\right]$$

$$= \prod_{j=1}^{d} \Pr\left[\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

$$\Pr[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i \geq \varepsilon \|\boldsymbol{a}\|_1] \leq e^{-1}$$

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is $\min\{\hat{\boldsymbol{a}}_{ij}\}$

$$\Pr\left[\min\left\{\hat{\boldsymbol{a}}_{ij}\right\} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

$$= \Pr\left[\bigwedge_{j=1}^{d}\left(\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right)\right]$$

$$= \prod_{j=1}^{d} \Pr\left[\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right]$$

$$\le \prod_{j=1}^{d} e^{-1}$$

$$\boxed{\Pr\left[\hat{\boldsymbol{a}}_i - \boldsymbol{a}_i \ge \varepsilon \|\boldsymbol{a}\|_1\right] \le e^{-1}}$$

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is min $\{\hat{\boldsymbol{a}}_{ij}\}$

$$
\begin{aligned}
& \Pr\left[\min\{\hat{\boldsymbol{a}}_{ij}\} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right] \\
= \; & \Pr\left[\bigwedge_{j=1}^{d} \left(\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right)\right] \\
= \; & \prod_{j=1}^{d} \Pr\left[\hat{\boldsymbol{a}}_{ij} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1\right] \\
\leq \; & \prod_{j=1}^{d} e^{-1} \\
= \; & e^{-d}
\end{aligned}
$$

Let $\hat{\boldsymbol{a}}_{ij}$ be the estimate from the jth copy of the data structure.

Our final estimate is $\min\{\hat{\boldsymbol{a}}_{ij}\}$

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

With probability at least 1 – δ, ***Probably***

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

for some measure of input size.

$$\Pr[\min\{\hat{\boldsymbol{a}}_{ij}\} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1] \leq e^{-d}$$

Idea: Choose d = -ln δ.
(Equivalently: $d = \ln \delta^{-1}$.) Then

$$\Pr[\min\{\hat{\boldsymbol{a}}_{ij}\} - \boldsymbol{a}_i > \varepsilon \|\boldsymbol{a}\|_1] \leq \delta$$

The Count-Min Sketch

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

h_1	31	41	59	26	53	...	58
h_2	27	18	28	18	28	...	45
h_3	16	18	3	39	88	...	75
...	...						
h_d	69	31	47	18	5	...	59

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

Sampled uniformly and independently from a 2-independent family of hash functions

The Count-Min Sketch

h_1	31	41	59	26	53	…	58
h_2	27	18	28	18	28	…	45
h_3	16	18	3	39	88	…	75
…				…			
h_d	69	31	47	18	5	…	59

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

The Count-Min Sketch

h_1	31	41	59	26	53	...	58
h_2	27	18	28	18	28	...	45
h_3	16	18	3	39	88	...	75
...				...			
h_d	69	31	47	18	5	...	59

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

The Count-Min Sketch

h_1	32	41	59	26	53	…	58
h_2	27	18	28	19	28	…	45
h_3	16	19	3	39	88	…	75
…				…			
h_d	69	31	47	18	5	…	60

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

The Count-Min Sketch

h_1	32	41	59	26	53	…	58
h_2	27	18	28	19	28	…	45
h_3	16	19	3	39	88	…	75
…				…			
h_d	69	31	47	18	5	…	60

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

The Count-Min Sketch

h_1	32	41	59	26	53	…	58
h_2	27	18	28	19	28	…	45
h_3	16	19	3	39	88	…	75
…	…						
h_d	69	31	47	18	5	…	60

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

```
estimate(x):
   result = ∞
   for i = 1 … d:
      result = min(result, count[i][hᵢ(x)])
   return result
```

The Count-Min Sketch

h_1	32	41	59	26	53	…	58
h_2	27	18	28	19	28	…	45
h_3	16	19	3	39	88	…	75
…				…			
h_d	69	31	47	18	5	…	60

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

```
estimate(x):
   result = ∞
   for i = 1 … d:
      result = min(result, count[i][hᵢ(x)])
   return result
```

The Count-Min Sketch

- Update and query times are $\Theta(d)$, which is $\Theta(\log \delta^{-1})$.
- Space usage: $\Theta(\varepsilon^{-1} \cdot \log \delta^{-1})$ counters.
  - This is a *major* improvement over our earlier approach that used $\Theta(\varepsilon^{-1} \cdot \delta^{-1})$ counters.
  - This can be *significantly* better than just storing a raw frequency count!
- Provides an estimate to within $\varepsilon\|\boldsymbol{a}\|_1$ with probability at least $1 - \delta$.

Major Ideas From Today

- ***2-independent hash families*** are useful when we want to keep collisions low.
- A “good” approximation of some quantity should have tunable ***confidence*** and ***accuracy*** parameters.
- ***Sums of indicator variables*** are useful for deriving expected values of estimators.
- ***Concentration inequalities*** like ***Markov’s inequality*** are useful for showing estimators don’t stay too much from their expected values.
- Good estimators can be built from multiple parallel copies of weaker estimators.

Next Time

- ***Count Sketches***
  - An alternative frequency estimator with different time/space bounds.
- ***Cardinality Estimation***
  - Estimating how many different items you’ve seen in a data stream.