

# 面向数据流的数据概要

Data Stream Synopses

**Too much input, too little space - how to deal with big data stream**

# Our Topic: “**Data Stream** Synopses”

- By “data stream”, we mean...
  - **One-Pass**
    - data source → our algorithm → black hole
  - **Limited Memory**
    - unable to store the whole stream
  - **High Speed**
    - must process each element in a time-efficient way
- How does data stream relate to **database systems**?
  - Sequential scan → ***a stream of records***
  - Append-only data, e.g., time series → ***a stream of insertion***
  - Insertion & updates & deletion to a table → ***a stream of deltas***

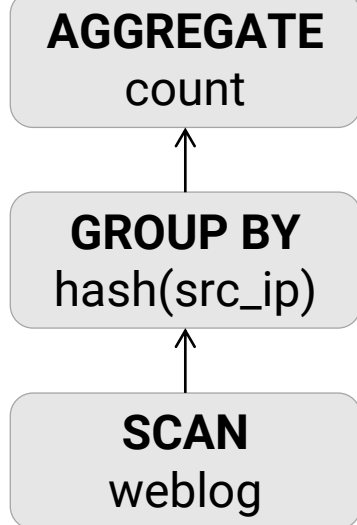
# Our Topic: “Data Stream **Synopses**”

- Synopses: 概要
- Also called ***sketches, probabilistic data structures***
- A compact representation of original data
- Provide approximate answers for particular queries

# How to Answer Following Queries?

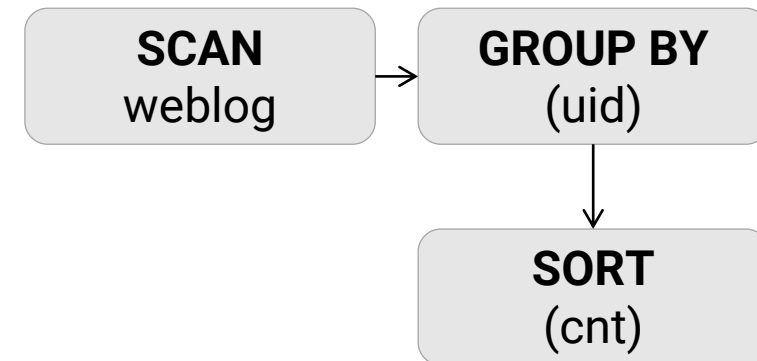
## Counting Distinct Values

```
SELECT COUNT(DISTINCT src_ip)
FROM   weblog
```



## TOP-K

```
SELECT  uid, COUNT(*) AS cnt
FROM    weblog
GROUP BY uid
ORDER BY cnt DESC
LIMIT   100
```



# Answering Them on **Data Streams**?

## Counting Distinct Values

```
SELECT COUNT(DISTINCT src_ip)
FROM   weblog
```

## TOP-K

```
SELECT  uid, COUNT(*) AS cnt
FROM    weblog
GROUP BY uid
ORDER BY cnt  DESC
LIMIT   100
```

- One-Pass
- Limited Memory
- High Speed

(uid, src\_ip, api, latency)

..., (u1, 12.34.56.78, /v1/query, 0.1s), (u2, 12.34.56.89, /v2/buy, 0.2s), ...

# Idea(doesn't work): **Incremental Update**

## Counting Distinct Values

```
SELECT COUNT(DISTINCT src_ip)
FROM   weblog
```

**AGGREGATE**  
count

**GROUP BY**  
hash(src\_ip)

**SCAN**  
weblog

must materialize these intermediate results

## TOP-K

```
SELECT  uid, COUNT(*) AS cnt
FROM    weblog
GROUP BY uid
ORDER BY cnt  DESC
LIMIT   100
```

**SCAN**  
weblog

**GROUP BY**  
(uid)

**SORT**  
(cnt)

# A Compromise: **Approximate Answers**

## Counting Distinct Values

```
SELECT COUNT(DISTINCT src_ip)  
FROM weblog
```

## TOP-K

```
SELECT uid, COUNT(*) AS cnt
```

We want a set of algorithms such that:

- One
- Limited
- High
- **space-efficient & time-efficient**
- report **approximate** answers

..., (u1, 12.56.89, /v2/buy, 0.2s), ...

...

# Probabilistic Data Structures

- As a database guy or a programmer, we may have heard about:
- **Bloom Filter**: for membership queries
- **HyperLogLog**: for counting distinct elements
- **Count-Min Sketch**: for finding heavy hitters or top-k
- Because of...



## Distributed count(distinct) with HyperLogLog on Postgres

BURAK YUCESQY - Apr 4, 2017

Running `SELECT COUNT(DISTINCT)` on your database is all too common. In applications it's typical to have some analytics dashboard highlighting the number of unique items such as unique users, unique products, unique visits. While traditional `SELECT COUNT(DISTINCT)` queries works well in single machine setups, it is a difficult problem to solve in distributed systems. When you have that type of query, you can't just push query to the workers and add up results, because most likely there will be overlapping records in different workers. Instead you can do:

- Pull all distinct data to one machine and count there. (Doesn't scale)
- Do a map/reduce. (Scales but it's very slow)

This is where approximation algorithms or sketches come in. Sketches are probabilistic algorithms which can generate approximate results efficiently within mathematically provable error bounds. There are a many of them out there, but today we're just going to focus on one, HyperLogLog or HLL. HLL is very successful for estimating unique number of elements in a list. First we'll look some at the internals of the HLL to help us understand why HLL algorithm is useful to solve distinct count problem in a scalable way, then how it can be applied in a distributed fashion. Then we will see some examples of HLL usage.

## HIGHLY SCALABLE BLOG

ARTICLES ON BIG DATA, NOSQL, AND HIGHLY SCALABLE SOFTWARE ENGINEERING

≡ MENU

... EXTRAS

POSTS

### PROBABILISTIC DATA STRUCTURES FOR WEB ANALYTICS AND DATA MINING

Statistical analysis and mining of huge multi-terabyte data sets is a common task nowadays, especially in the areas like web analytics and Internet advertising. Analysis of such large data sets often requires powerful distributed data stores like Hadoop and heavy data processing with techniques like MapReduce. This approach often leads to heavyweight high-latency analytical processes and poor applicability to realtime use cases. On the other hand, when one is interested only in simple additive metrics like total page views or average price of conversion, it is obvious that raw data can be efficiently summarized, for example, on a daily basis or using simple in-stream counters. Computation of more advanced metrics like a number of unique visitor or most frequent items is more challenging and requires a lot of resources if implemented straightforwardly. In this article, I provide an overview of probabilistic data structures that allow one to estimate these and many other metrics and trade precision of the estimations for the memory consumption. These data structures can be used both as temporary data accumulators in query

## GOOGLE CLOUD BIG DATA AND MACHINE LEARNING BLOG

Innovation in data processing and machine learning technology



### Counting uniques faster in BigQuery with HyperLogLog++

Wednesday, July 5, 2017

By *Felipe Hoffa*, Developer Advocate

As a data exploration task, counting unique users is usually slow and resource intensive because your database needs to keep track of every unique id it has ever seen and that can consume a lot of RAM.

Let's try it with BigQuery: How many unique users did GitHub have in 2016?

## Count Min Sketch: The Art and of Estimating Stuff

Aug 18, 2016 by *Itamar Haber*

**countminsketch**  
An approximate items counter



# A Bunch of Blog Posts!

# Some Questions...

- Of course we can...
  - pull an open-source implementation from Github
  - plug it in our projects with default parameters
- But do we know:
  - How they works
  - Why they works
  - What error guarantees they provide
  - Where they can be applied
  - Whether they can be improved

# Some Questions...

- Of course we can...
  - pull an open-source implementation from Github
  - plug it in our projects with default parameters

- But do we know:

- How they works
- Why they works
- What error guarantees they provide
- Where they can be applied
- Whether they can be improved

Goals for this talk



# In this talk, we will...

- Give an overview of data stream synopses
- Show the basic theory behind them
- Explain the error guarantees provided by them

# Outline

- Data Stream Model ← a unified view of queries considered by us
- Sketch 101: AMS Sketch ← see how a classic sketch is designed
- The Magic of AMS Sketch ← adapt AMS sketch to answer almost all queries
- Counting Distinct Elements
- Finding Heavy Hitters

# Outline

- Data Stream Model
- Sketch 101: AMS Sketch
- The Magic of AMS Sketch
- Counting Distinct Elements
- Finding Heavy Hitters

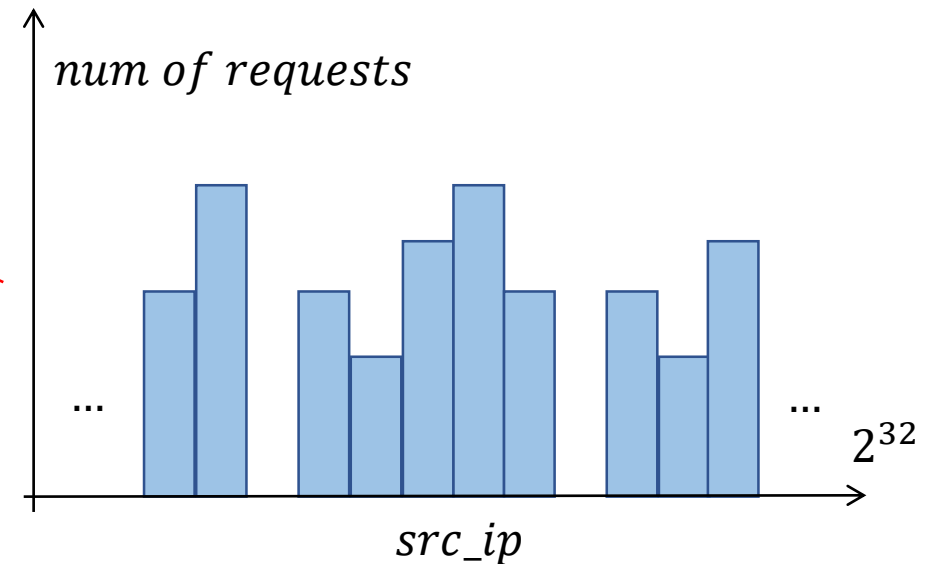
# A not-so-obvious Fact

- Many queries are equivalent to **histogram queries**

# A not-so-obvious Fact

- Many queries are equivalent to histogram queries

TOP-K	
SELECT	<b>src_ip</b> , COUNT(*) cnt
FROM	weblog
GROUP BY	<b>src_ip</b>
ORDER BY	cnt DESC
LIMIT	100





# A not-so-obvious Fact

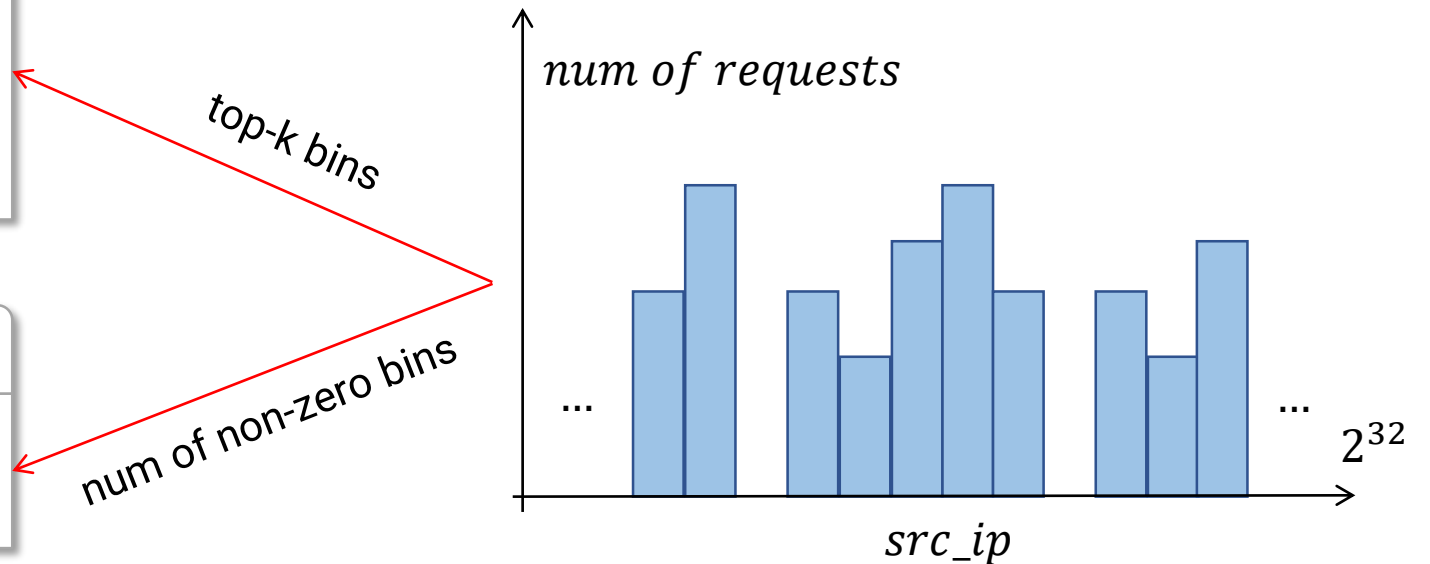
- Many queries are equivalent to histogram queries

## TOP-K

```
SELECT  src_ip, COUNT(*) cnt
FROM    weblog
GROUP BY src_ip
ORDER BY cnt DESC
LIMIT   100
```

## Counting Distinct Values

```
SELECT COUNT(DISTINCT src_ip)
FROM    weblog
```



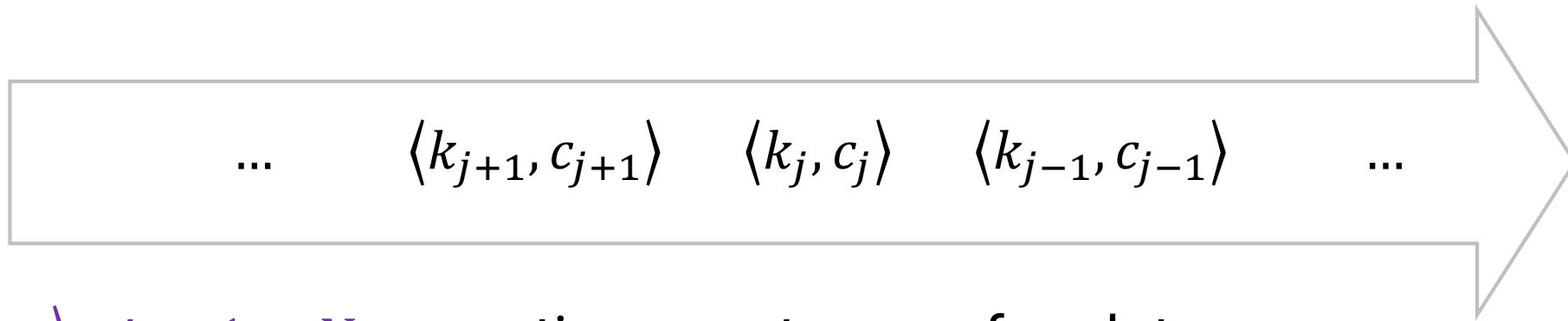
# Data Stream: A Histogram Model

- $\mathbb{H}[1 \dots U]$ : a **logical**, massive, dynamic, 1-dim vector of counters



# Data Stream: A Histogram Model

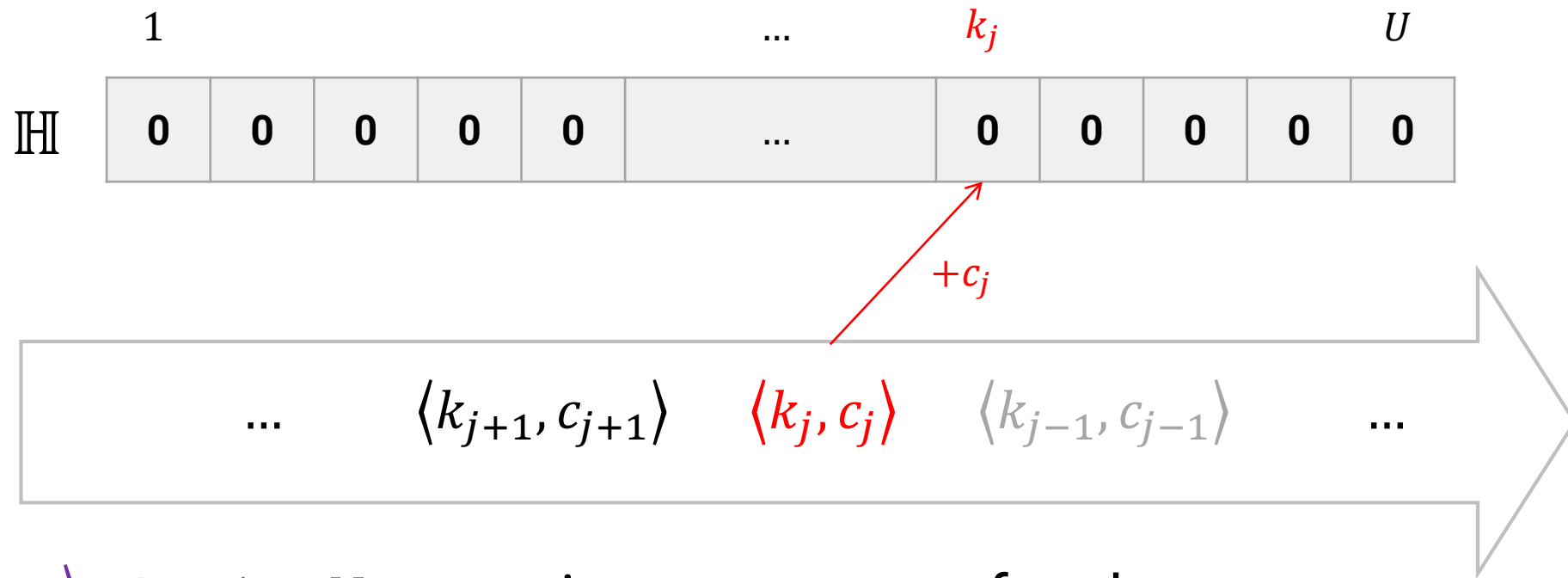
- $\mathbb{H}[1 \dots U]$ : a **logical**, massive, dynamic, 1-dim vector of counters



- $\langle k_j, c_j \rangle, j = 1 \dots N$ : a continuous stream of updates
  - Modify the vector as  $\mathbb{H}[k_j] \leftarrow \mathbb{H}[k_j] + c_j$

# Data Stream: A Histogram Model

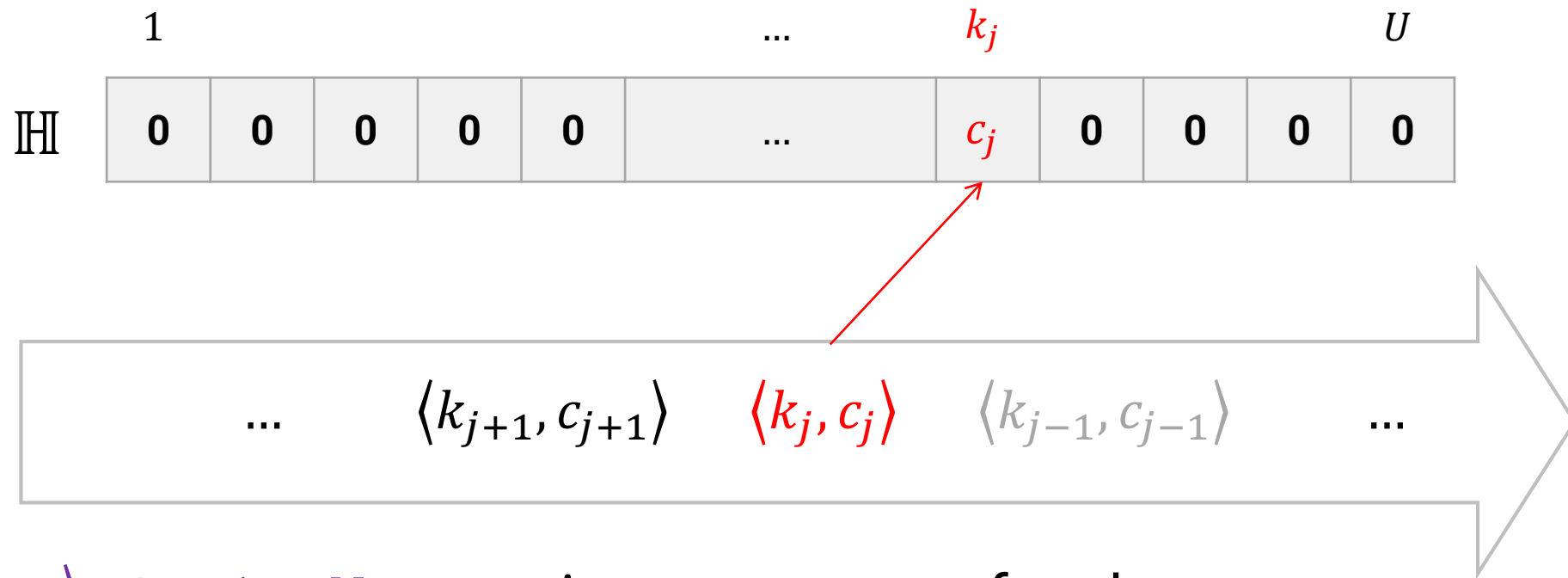
- $\mathbb{H}[1 \dots U]$ : a **logical**, massive, dynamic, 1-dim vector of counters



- $\langle k_j, c_j \rangle, j = 1 \dots N$ : a continuous stream of updates
  - Modify the vector as  $\mathbb{H}[k_j] \leftarrow \mathbb{H}[k_j] + c_j$

# Data Stream: A Histogram Model

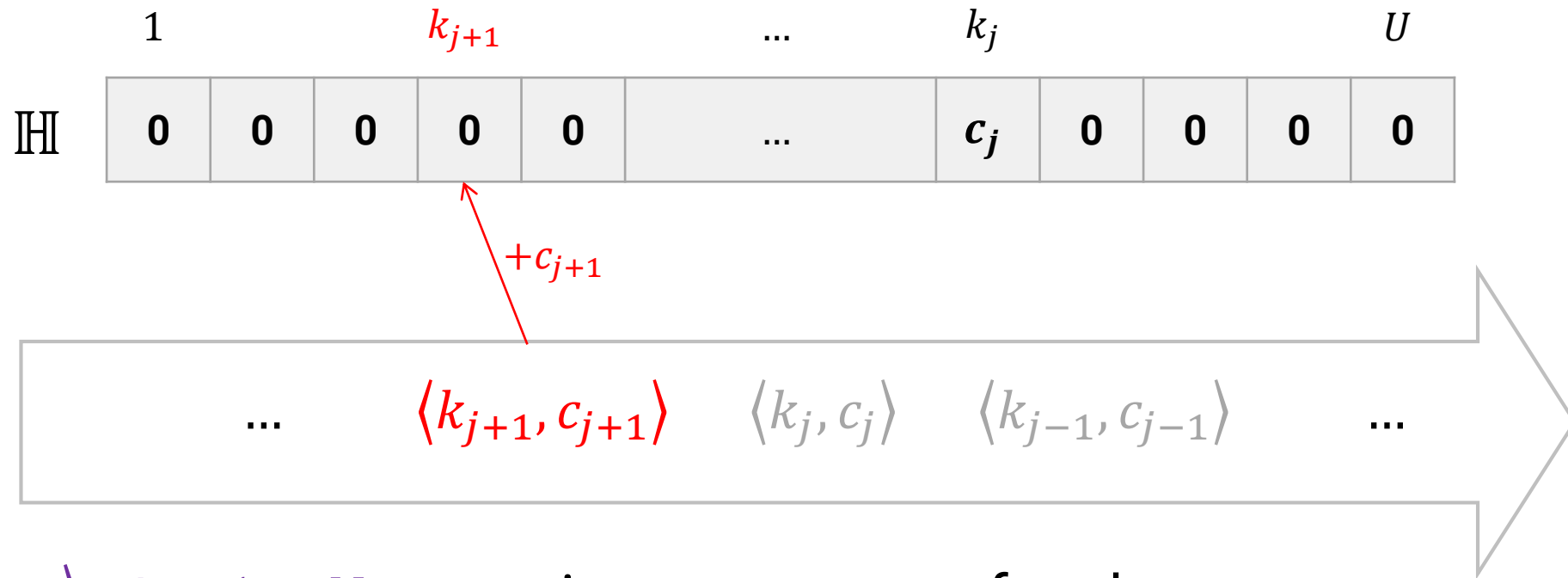
- $\mathbb{H}[1 \dots U]$ : a **logical**, massive, dynamic, 1-dim vector of counters



- $\langle k_j, c_j \rangle, j = 1 \dots N$ : a continuous stream of updates
  - Modify the vector as  $\mathbb{H}[k_j] \leftarrow \mathbb{H}[k_j] + c_j$

# Data Stream: A Histogram Model

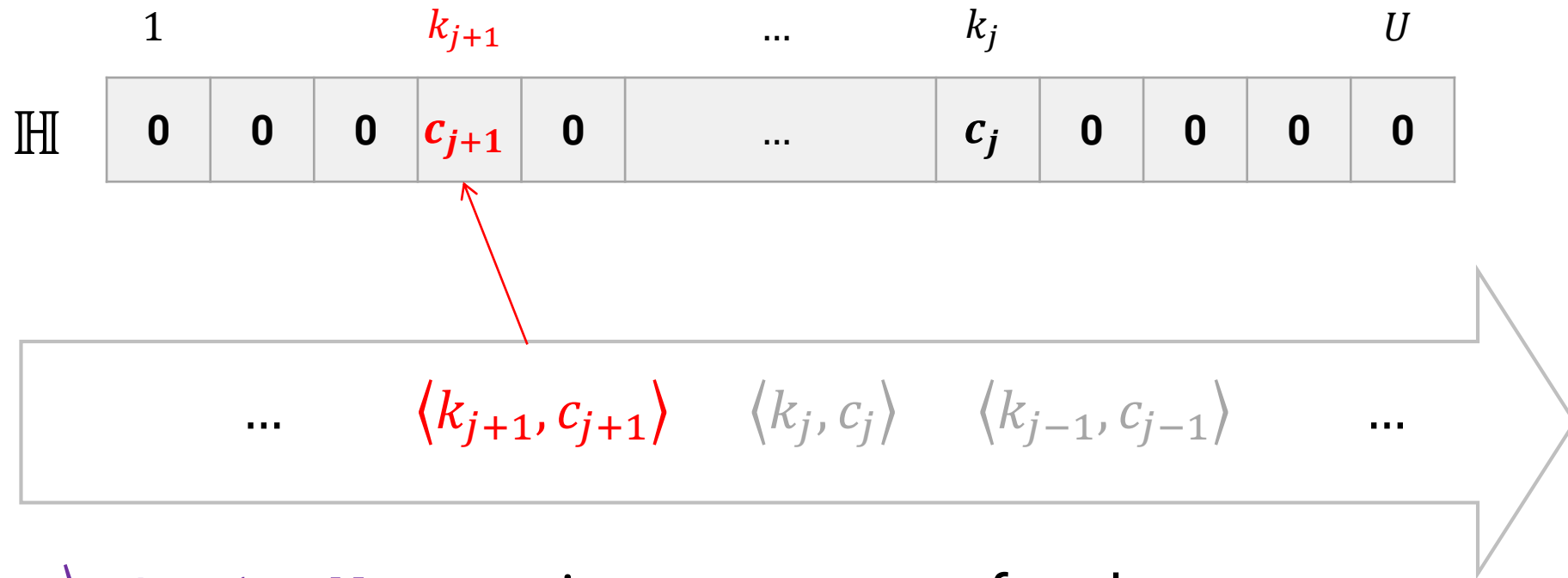
- $\mathbb{H}[1 \dots U]$ : a **logical**, massive, dynamic, 1-dim vector of counters



- $\langle k_j, c_j \rangle, j = 1 \dots N$ : a continuous stream of updates
  - Modify the vector as  $\mathbb{H}[k_j] \leftarrow \mathbb{H}[k_j] + c_j$

# Data Stream: A Histogram Model

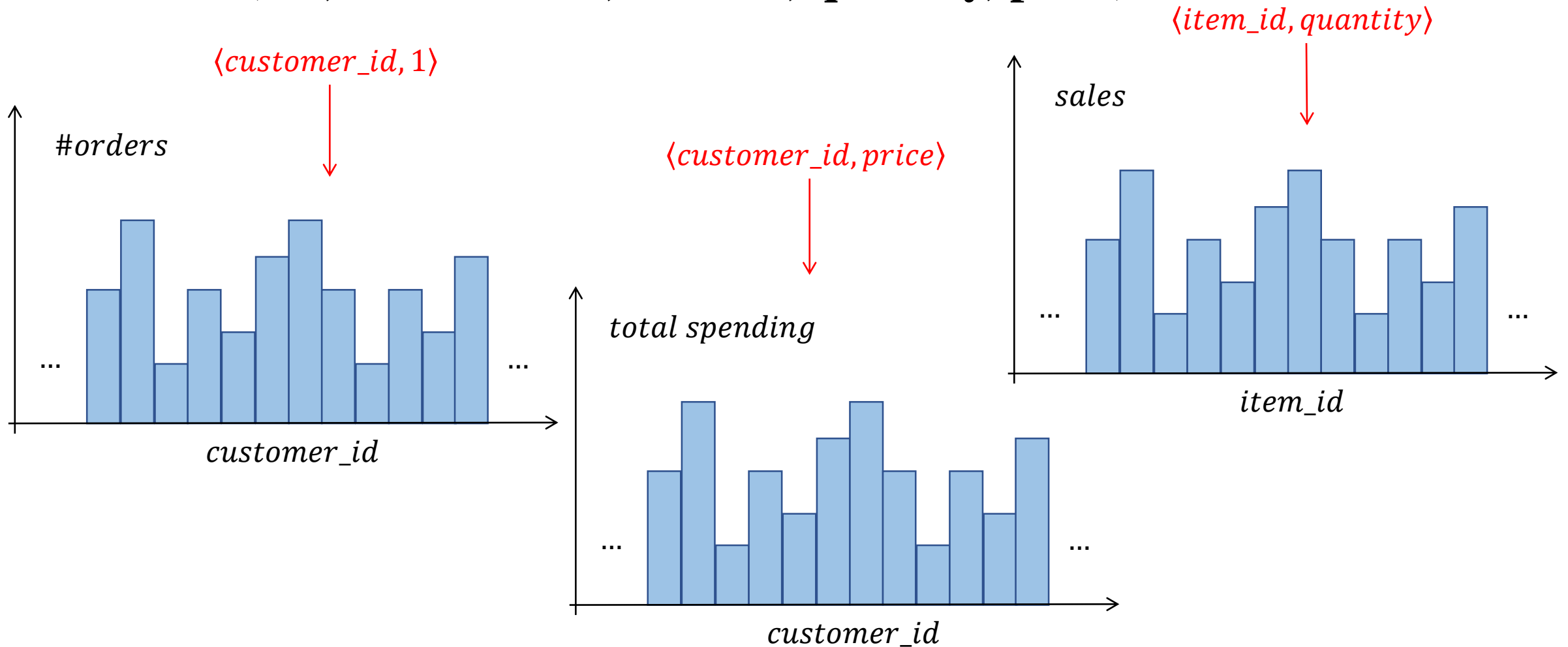
- $\mathbb{H}[1 \dots U]$ : a **logical**, massive, dynamic, 1-dim vector of counters



- $\langle k_j, c_j \rangle, j = 1 \dots N$ : a continuous stream of updates
  - Modify the vector as  $\mathbb{H}[k_j] \leftarrow \mathbb{H}[k_j] + c_j$

# Example: Append-Only Fact Table

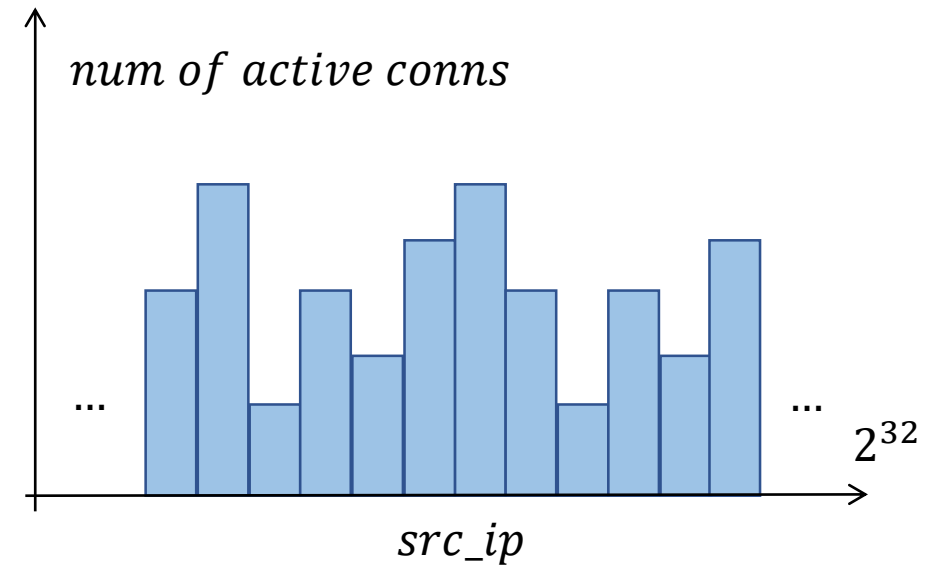
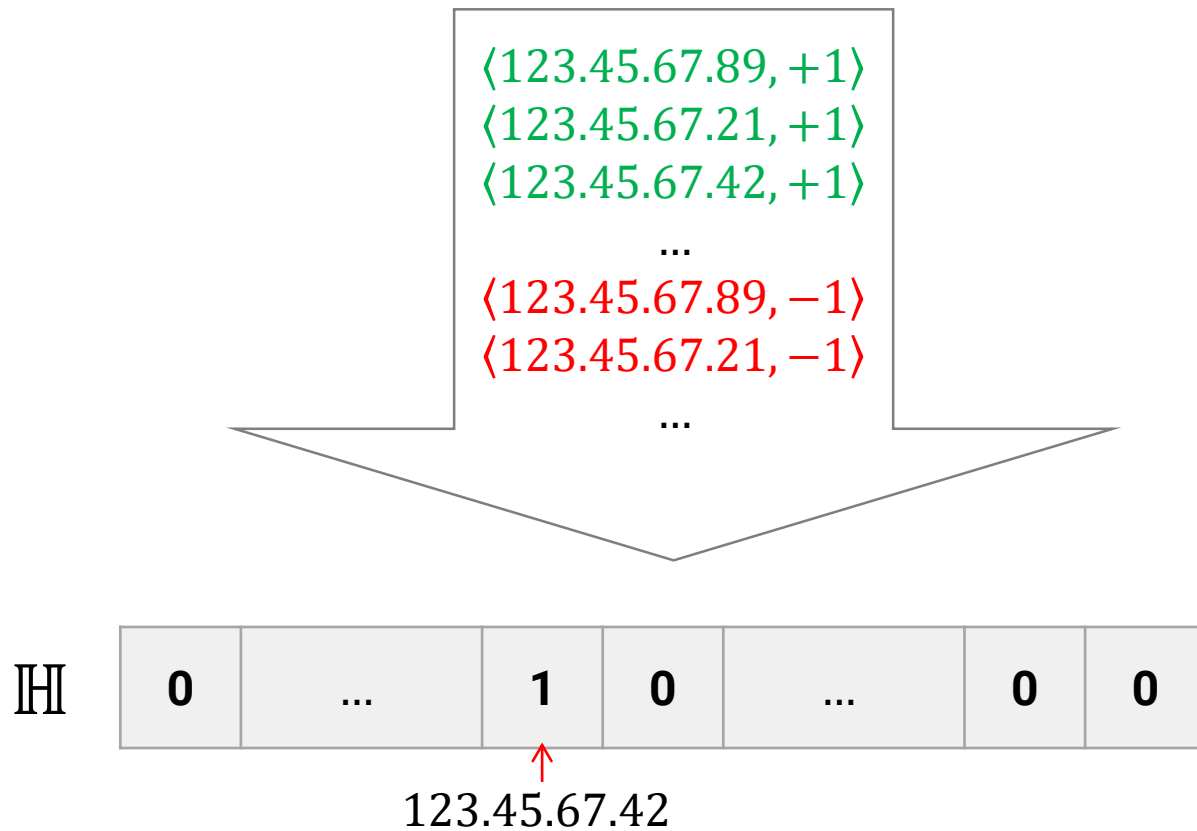
- **orders(oid, customer\_id, item\_id, quantity, price)**





# Example: TCP Connection Counters

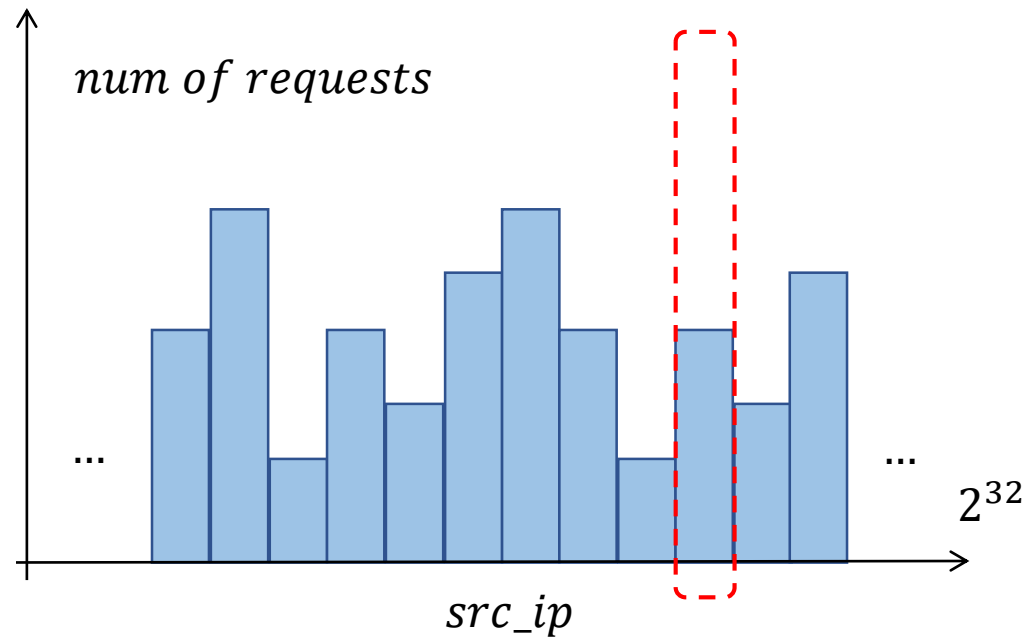
- Connection **established**/**terminated**  $\rightarrow \langle src\_ip, \pm 1 \rangle$



# Queries over the Histogram

$\mathbb{H}$	0	1	4	...	0	2	1
	1			...			$U$

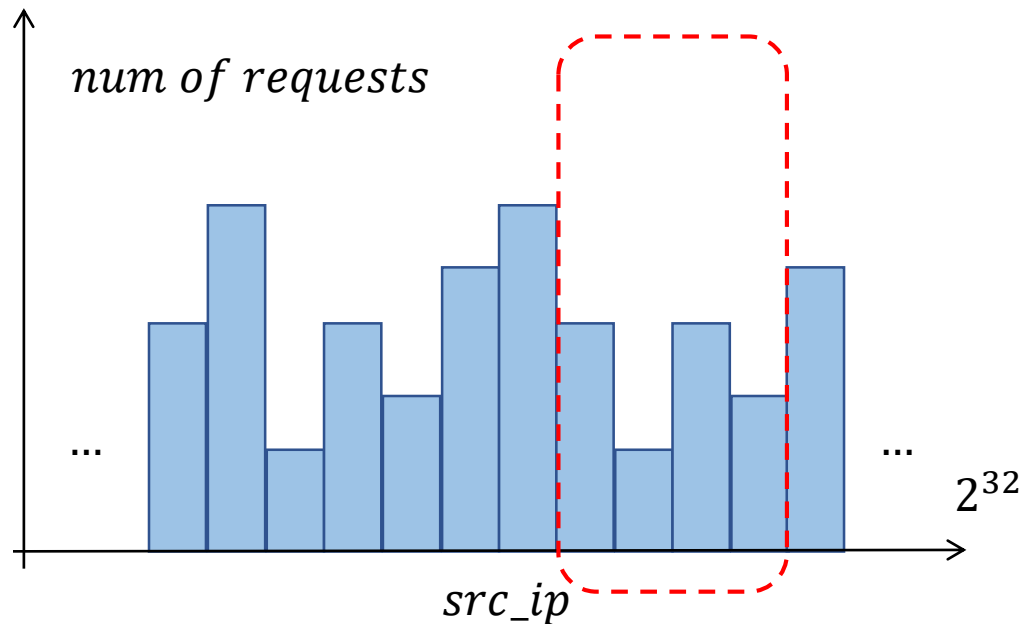
Point Query	$\mathbb{H}[i]$
-------------	-----------------



# Queries over the Histogram

$\mathbb{H}$	0	1	4	...	0	2	1
	1			...			$U$

Point Query	$\mathbb{H}[i]$
Range Query	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

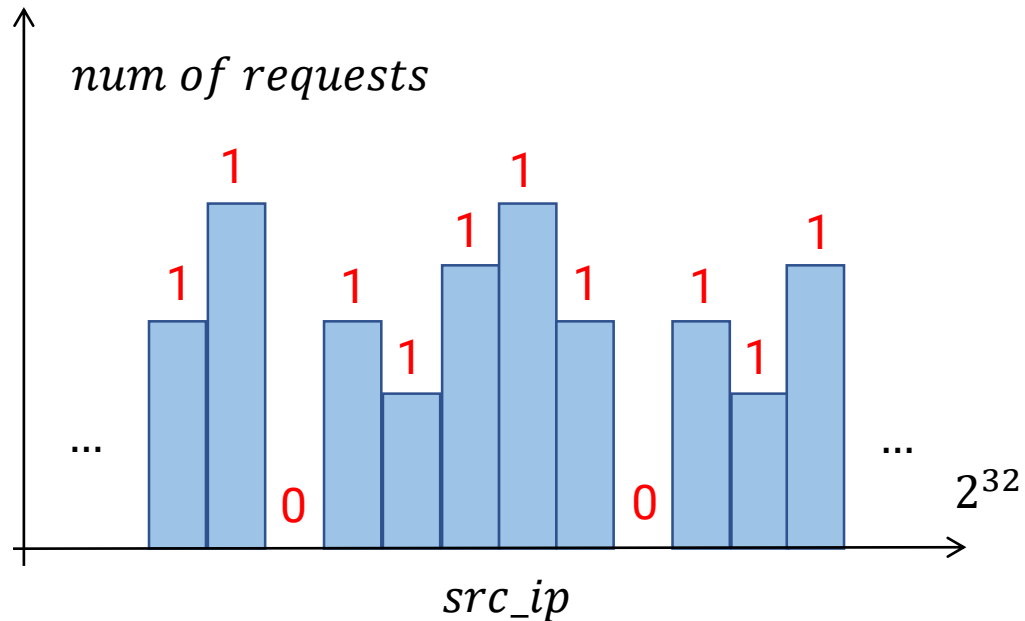


# Queries over the Histogram

$\mathbb{H}$	0	1	4	...	0	2	1
	1			...			$U$

<b>Point Query</b>	$\mathbb{H}[i]$
<b>Range Query</b>	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

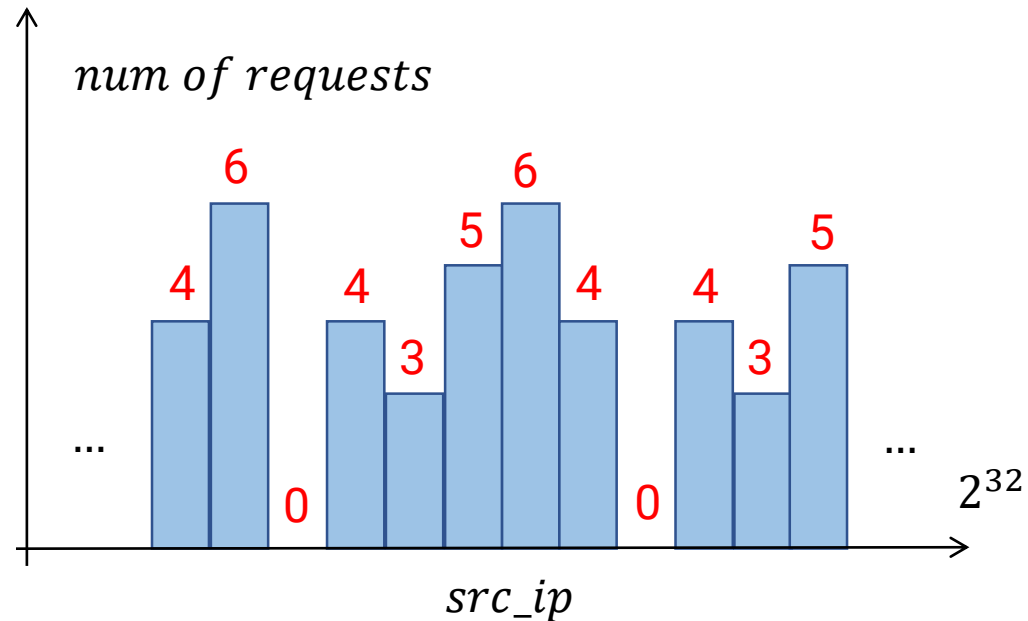
<b>Distinct Elements</b>	$\sum_{i=1}^U \mathbb{H}[i]^0$
--------------------------	--------------------------------



# Queries over the Histogram

$\mathbb{H}$	0	1	4	...	0	2	1
	1			...			$U$

<b>Point Query</b>	$\mathbb{H}[i]$
<b>Range Query</b>	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

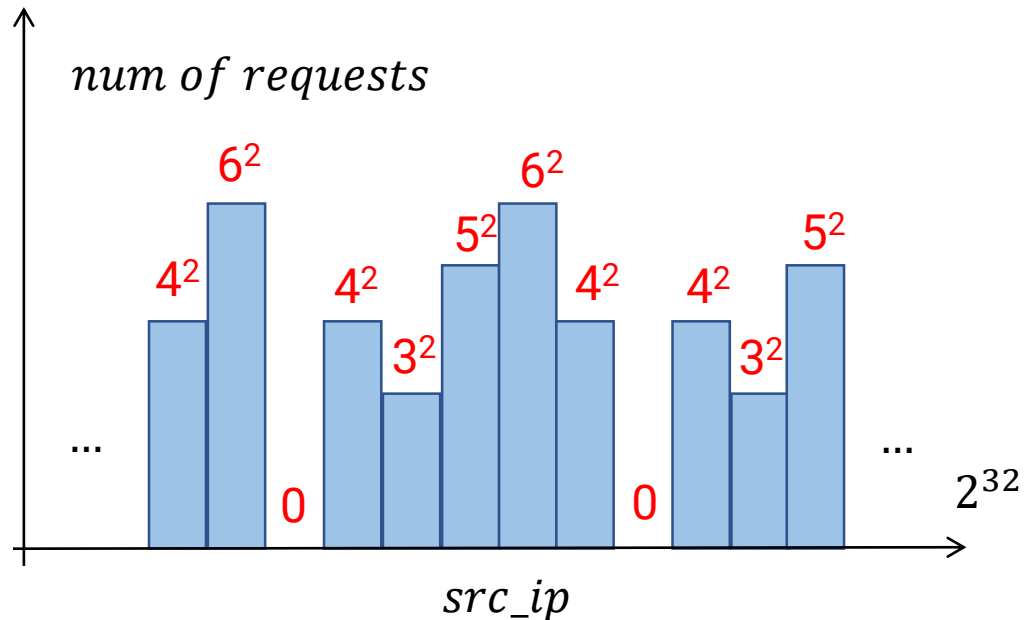


<b>Distinct Elements</b>	$\sum_{i=1}^U \mathbb{H}[i]^0$
<b>Sum</b>	$\sum_{i=1}^U \mathbb{H}[i]$

# Queries over the Histogram

$\mathbb{H}$	0	1	4	...	0	2	1
	1			...			$U$

<b>Point Query</b>	$\mathbb{H}[i]$
<b>Range Query</b>	$\sum_{i=t}^{t+w} \mathbb{H}[i]$



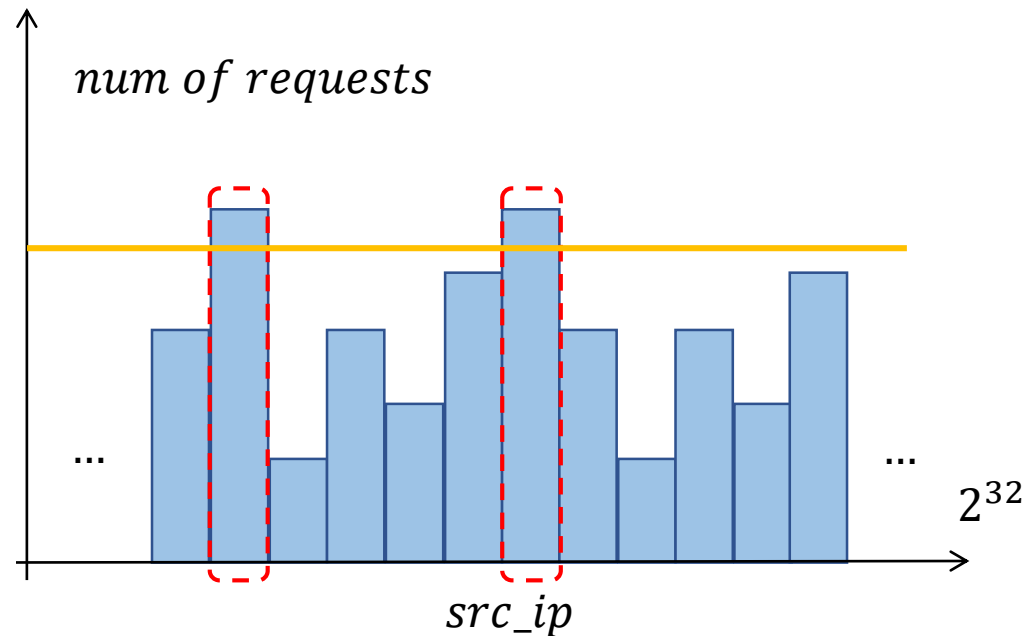
<b>Distinct Elements</b>	$\sum_{i=1}^U \mathbb{H}[i]^0$
<b>Sum</b>	$\sum_{i=1}^U \mathbb{H}[i]$
<b>Sum of Squares</b>	$\sum_{i=1}^U \mathbb{H}[i]^2$

Seem like useless, but we will see its usage later

# Queries over the Histogram

$\mathbb{H}$	0	1	4	...	0	2	1
	1			...			$U$

<b>Point Query</b>	$\mathbb{H}[i]$
<b>Range Query</b>	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

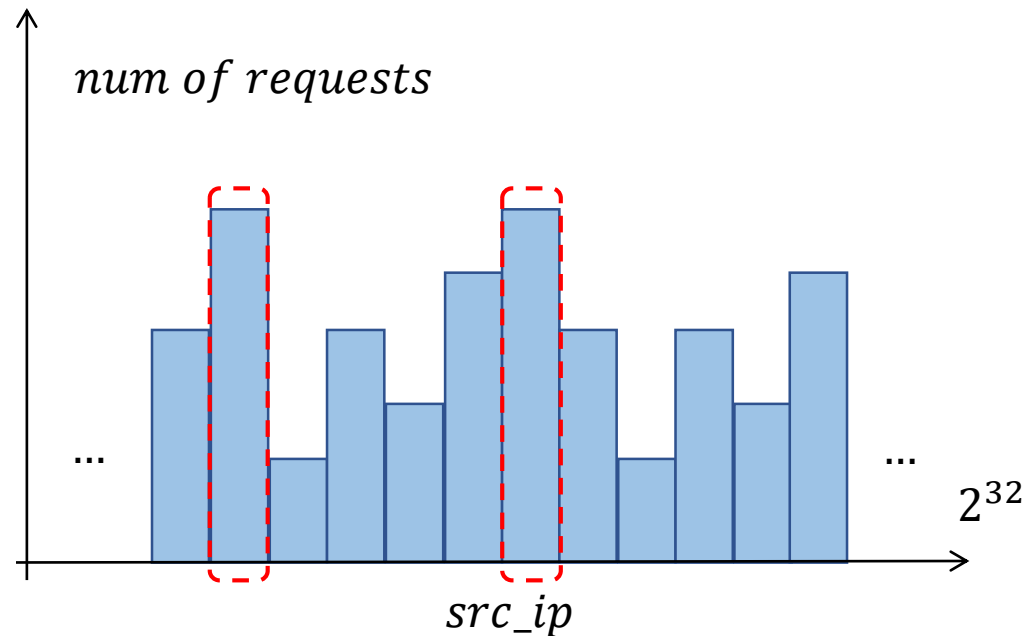


<b>Distinct Elements</b>	$\sum_{i=1}^U \mathbb{H}[i]^0$
<b>Sum</b>	$\sum_{i=1}^U \mathbb{H}[i]$
<b>Sum of Squares</b>	$\sum_{i=1}^U \mathbb{H}[i]^2$
<b>Heavy Hitters</b>	$\left\{ i: \mathbb{H}[i] \geq \varepsilon \sum_{k=1}^U \mathbb{H}[k] \right\}$

# Queries over the Histogram

$\mathbb{H}$	0	1	4	...	0	2	1
	1			...			$U$

<b>Point Query</b>	$\mathbb{H}[i]$
<b>Range Query</b>	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

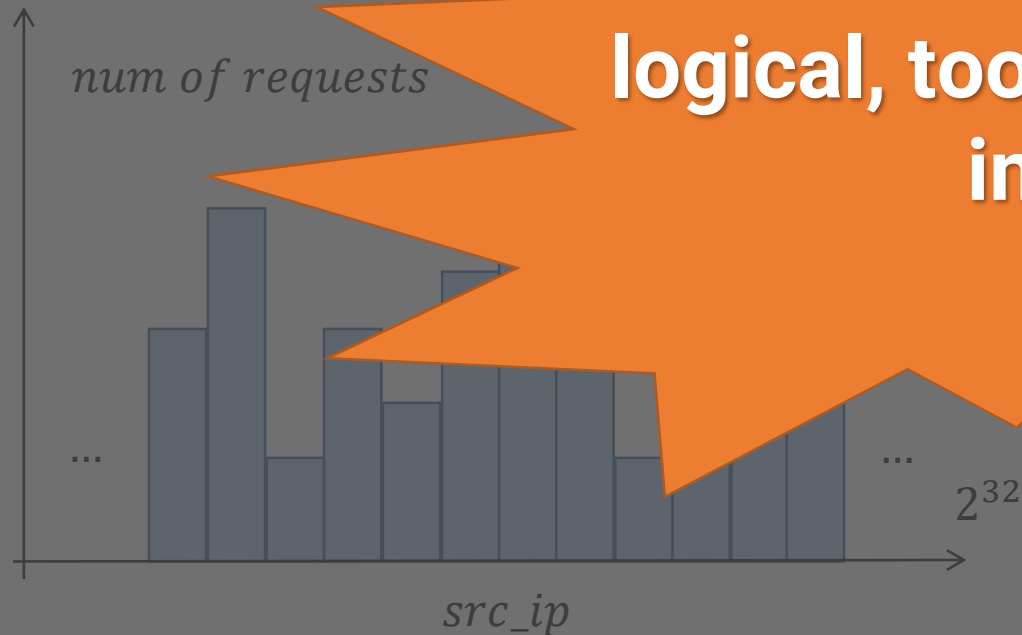


<b>Distinct Elements</b>	$\sum_{i=1}^U \mathbb{H}[i]^0$
<b>Sum</b>	$\sum_{i=1}^U \mathbb{H}[i]$
<b>Sum of Squares</b>	$\sum_{i=1}^U \mathbb{H}[i]^2$
<b>Heavy Hitters</b>	$\left\{ i: \mathbb{H}[i] \geq \varepsilon \sum_{k=1}^U \mathbb{H}[k] \right\}$
<b>Top-K</b>	



# Queries over the Histogram

$\mathbb{H}$	0	1	4	...	0	2	1
	1			...			$U$



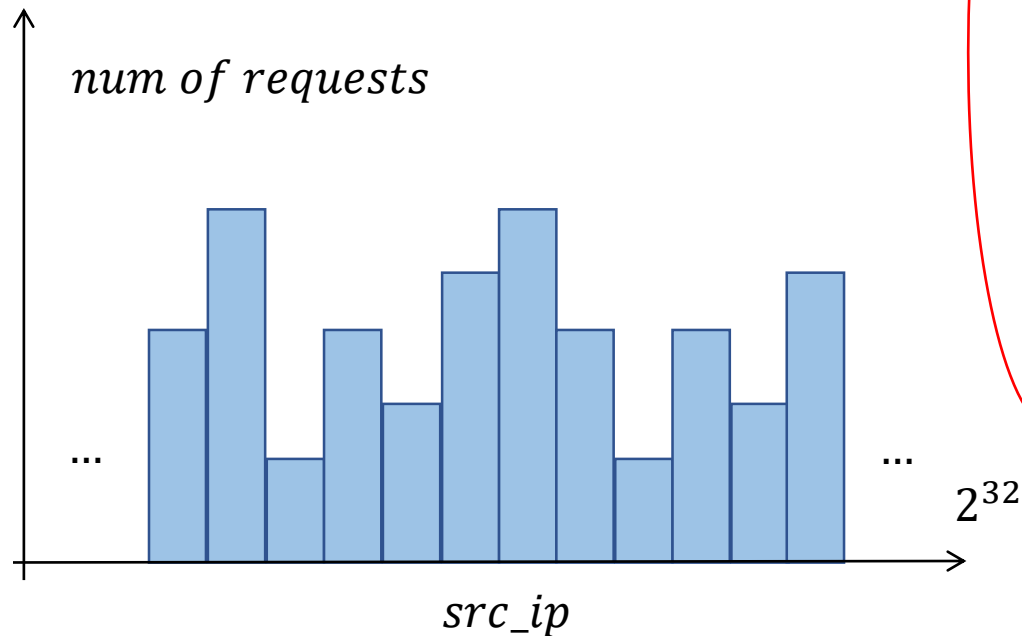
logical, too large to be stored  
in memory

Point Query	$\mathbb{H}[i]$
Range Query	$\sum_{i=t}^{t+w} \mathbb{H}[i]$
Count	$\sum_{i=1}^U \mathbb{H}[i]$
Sum of Squares	$\sum_{i=1}^U \mathbb{H}[i]^2$
Heavy Hitters	$\left\{ i: \mathbb{H}[i] \geq \varepsilon \sum_{k=1}^U \mathbb{H}[k] \right\}$
Top-K	

# Queries over the Histogram

**hard to approximate**  
using very small space

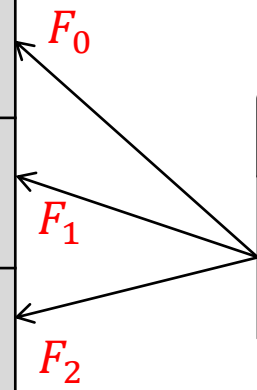
$\mathbb{H}$	0	1	4	...	0	2	1
	1			...			$U$



<b>Point Query</b>	$\mathbb{H}[i]$
<b>Range Query</b>	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

<b>Distinct Elements</b>	$\sum_{i=1}^U \mathbb{H}[i]^0$
<b>Sum</b>	$\sum_{i=1}^U \mathbb{H}[i]$
<b>Sum of Squares</b>	$\sum_{i=1}^U \mathbb{H}[i]^2$
<b>Heavy Hitters</b>	$\left\{ i: \mathbb{H}[i] \geq \varepsilon \sum_{k=1}^U \mathbb{H}[k] \right\}$
<b>Top-K</b>	

# Queries over the Histogram

<b>Distinct Elements</b>	$F_0(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^0$	 $F_0$
<b>Sum</b>	$F_1(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]$	
<b>Sum of Squares</b>	$F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2$	

**Frequency Moments**  
 $F_p(\mathbb{H}) = \sum_i \mathbb{H}[i]^p, p \in \mathbb{Z}$

$\mathbb{H}$	<b>0</b>	<b>1</b>	<b>4</b>	...	<b>0</b>	<b>2</b>	<b>1</b>
	1			...			$U$

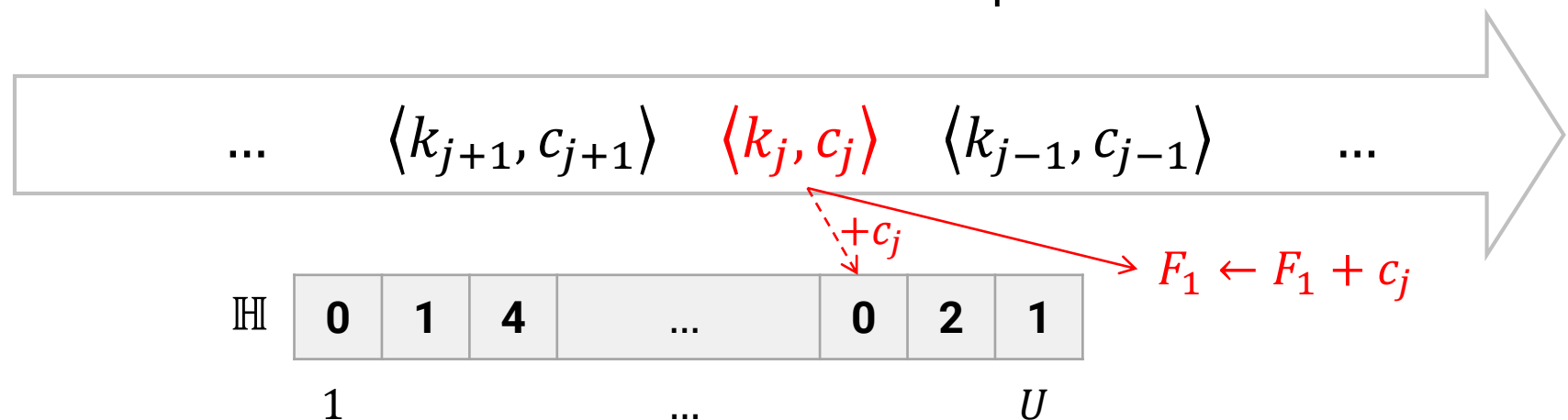
# Queries over the Histogram

<b>Distinct Elements</b>	$F_0(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^0$	$F_0$
<b>Sum</b>	$F_1(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]$	$F_1$
<b>Sum of Squares</b>	$F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2$	$F_2$

**Frequency Moments**

$$F_p(\mathbb{H}) = \sum_i \mathbb{H}[i]^p, p \in \mathbb{Z}$$

Only  $F_1$  can be compute exactly on data streams, using 1-word space



# Queries over the Histogram

<b>Point Query</b>	$\mathbb{H}[i]$
<b>Range Query</b>	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

<b>Distinct Elements</b>	$F_0(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^0$
<del><b>Sum</b></del>	<del><math>F_1(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]</math></del>
<b>Sum of Squares</b>	$F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2$
<b>Heavy Hitters</b>	$\left\{ i: \mathbb{H}[i] \geq \varepsilon \sum_{k=1}^U \mathbb{H}[k] \right\}$
<b>Top-K</b>	

$\mathbb{H}$	<b>0</b>	<b>1</b>	<b>4</b>	...	<b>0</b>	<b>2</b>	<b>1</b>
	1			...			$U$

Our goal is to provide approximate answers for them on data streams

# Outline

- Data Stream Model
- Sketch 101: AMS Sketch
- The Magic of AMS Sketch
- Counting Distinct Elements
- Finding Heavy Hitters

how a classic sketch is designed

Point Query	$\mathbb{H}[i]$
Range Query	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

Distinct Elements	$F_0(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^0$
Sum	$F_1(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]$
Sum of Squares	$F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2$
Heavy Hitters	$\left\{ i: \mathbb{H}[i] \geq \varepsilon \sum_{k=1}^U \mathbb{H}[k] \right\}$
Top-K	

# Sketches: The Common Part

- To estimate a target variable **X**, we maintain a variable **Y**
  - Call them “variable” because they are dependent on the data stream

# Sketches: The Common Part

- To estimate a target variable **X**, we maintain a variable **Y**
  - Call them “variable” because they are dependent on the data stream
- **Y** is called “**atomic sketch**”
  - **Y** can be maintained on data streams efficiently
  - **Y** can be used to estimate **X**, i.e.,  $\mathbf{X} \approx f(\mathbf{Y})$
  - In general, some randomness is introduced to make **Y** a random variable
    - If one **Y** is not enough, we maintain multiply copies of **Y**



# Sketches: The Common Part

- To estimate a target variable **X**, we maintain a variable **Y**
  - Call them “variable” because they are dependent on the data stream
- **Y** is called “**atomic sketch**”
  - **Y** can be maintained on data streams efficiently
  - **Y** can be used to estimate **X**, i.e.,  $\mathbf{X} \approx f(\mathbf{Y})$
  - In general, some randomness is introduced to make **Y** a random variable
    - If one **Y** is not enough, we maintain multiply copies of **Y**

# Sketches: The Common Part

- To estimate a target variable  $X$ , we maintain a variable  $Y$ 
  - Call them “variable” because they are dependent on the data stream
- $Y$  is called “**atomic sketch**”
  - $Y$  can be maintained on data streams efficiently
  - $Y$  can be used to estimate  $X$ , i.e.,  $X \approx f(Y)$
  - In general, some randomness is introduced to make  $Y$  a random variable
    - If one  $Y$  is not enough, we maintain multiply copies of  $Y$

Designing such a  $Y$  is the hardest part - let's take  $F_2$  as an example

# $F_2$ as the **Self-Join** Size

```
SELECT  A.id, B.id, A.salary - B.salary
FROM    employee A JOIN employee B
        USING (department)
WHERE   A.id < B.id
```

*employee*

id	salary	department
1	10000	A
2	9000	A
3	12000	B
4	9999	C
5	12345	C
6	11111	C

$\mathbb{H}$

2	1	3
A	B	C

2 \* 2

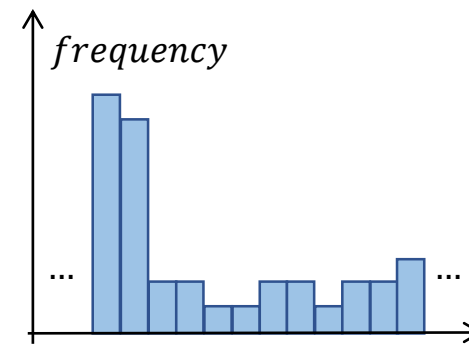
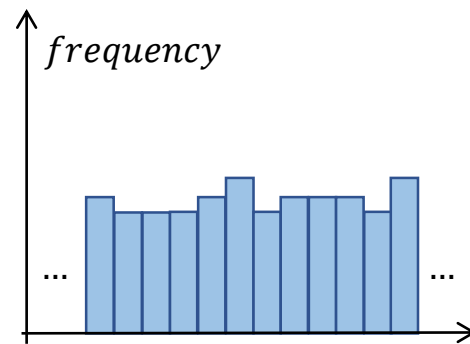
1 \* 1

3 \* 3

$$F_2(\mathbb{H}) = \sum_i \mathbb{H}[i]^2$$

# $F_2$ as a Skewness Measure

- Also called **surprise number**
  - Given  $F_0$  and  $F_1$ , surprise number reflects the skewness of the distribution



- 100 people in total, 11 teams

10	9	9	9	9	9	9	9	9	9	9
----	---	---	---	---	---	---	---	---	---	---

→  $F_2 = 910$

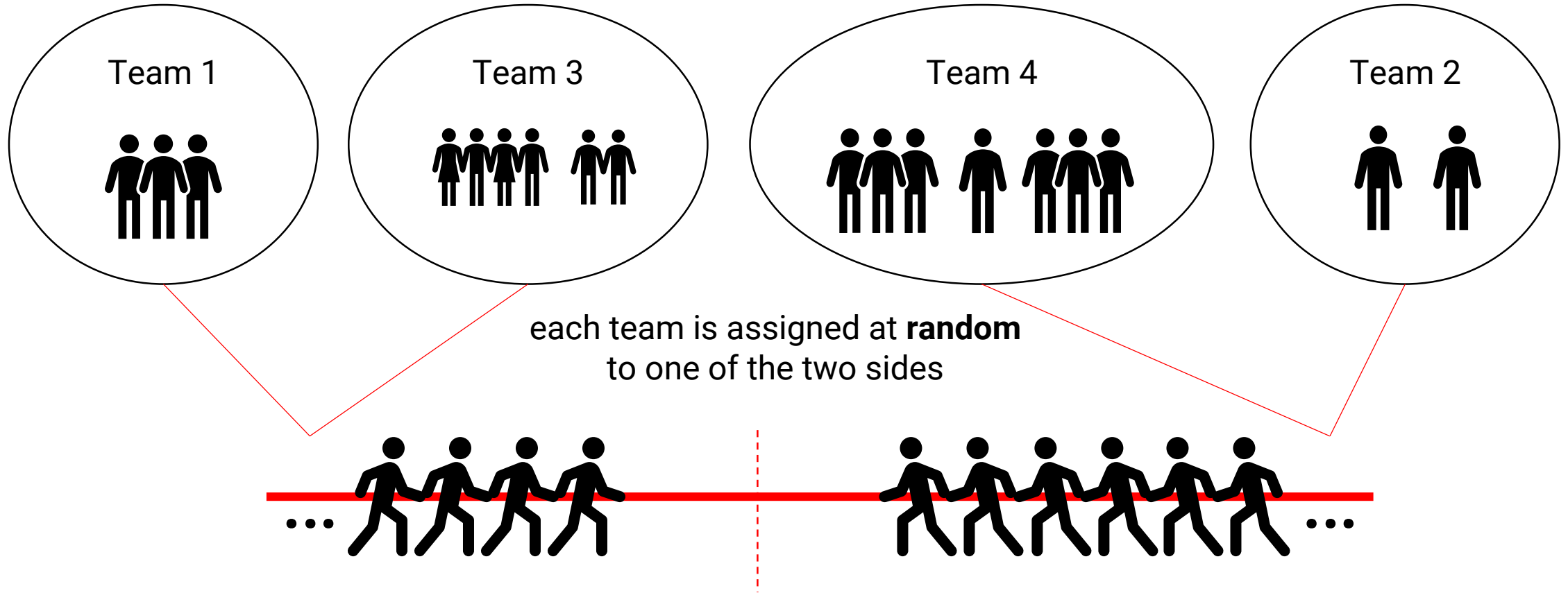
90	1	1	1	1	1	1	1	1	1	1
----	---	---	---	---	---	---	---	---	---	---

→  $F_2 = 8110$

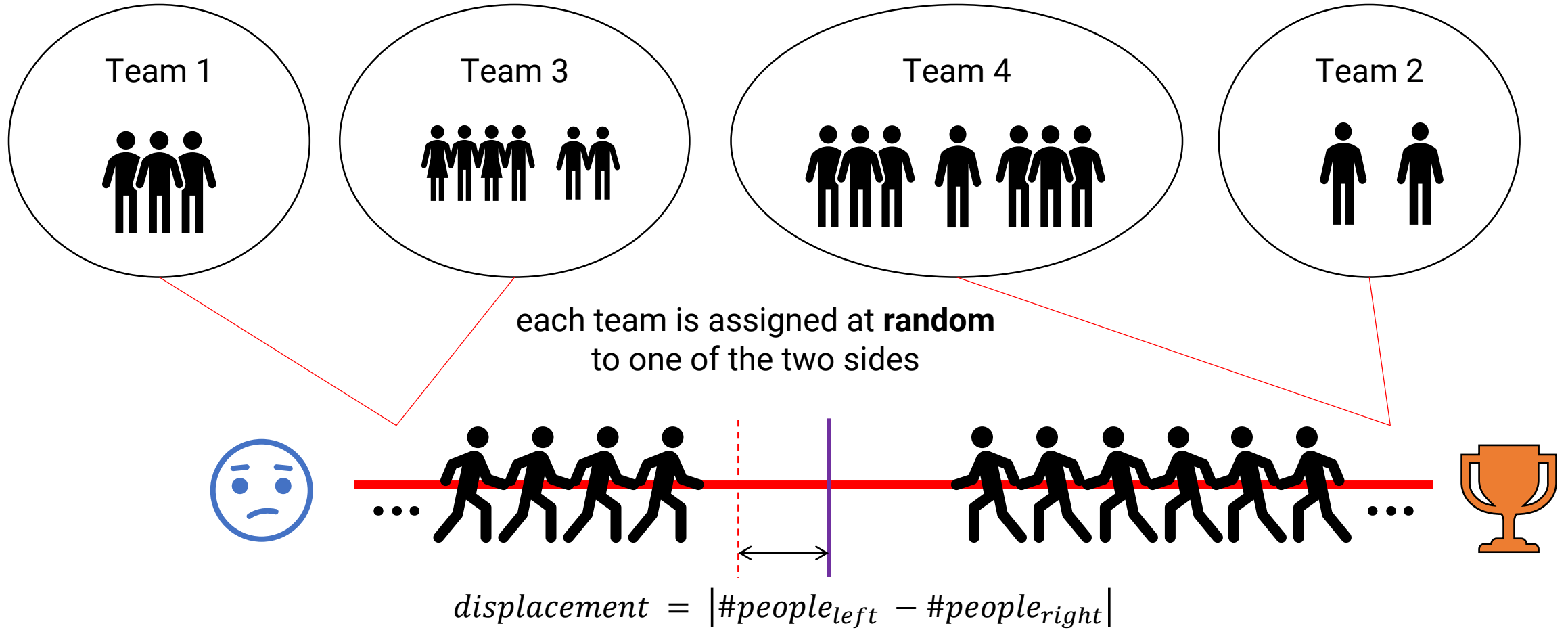
# Estimating $F_2$ : AMS Sketch

- What is the atomic sketch for  $F_2$ ?
- Recall that  $F_2$  reflects the skewness of the distribution
- Suppose:
  - A company consists several teams of varying number of people
  - Our goal is to estimate the skew in the distribution of people to teams
  - How? We arrange a **tug-of-war!** 😊

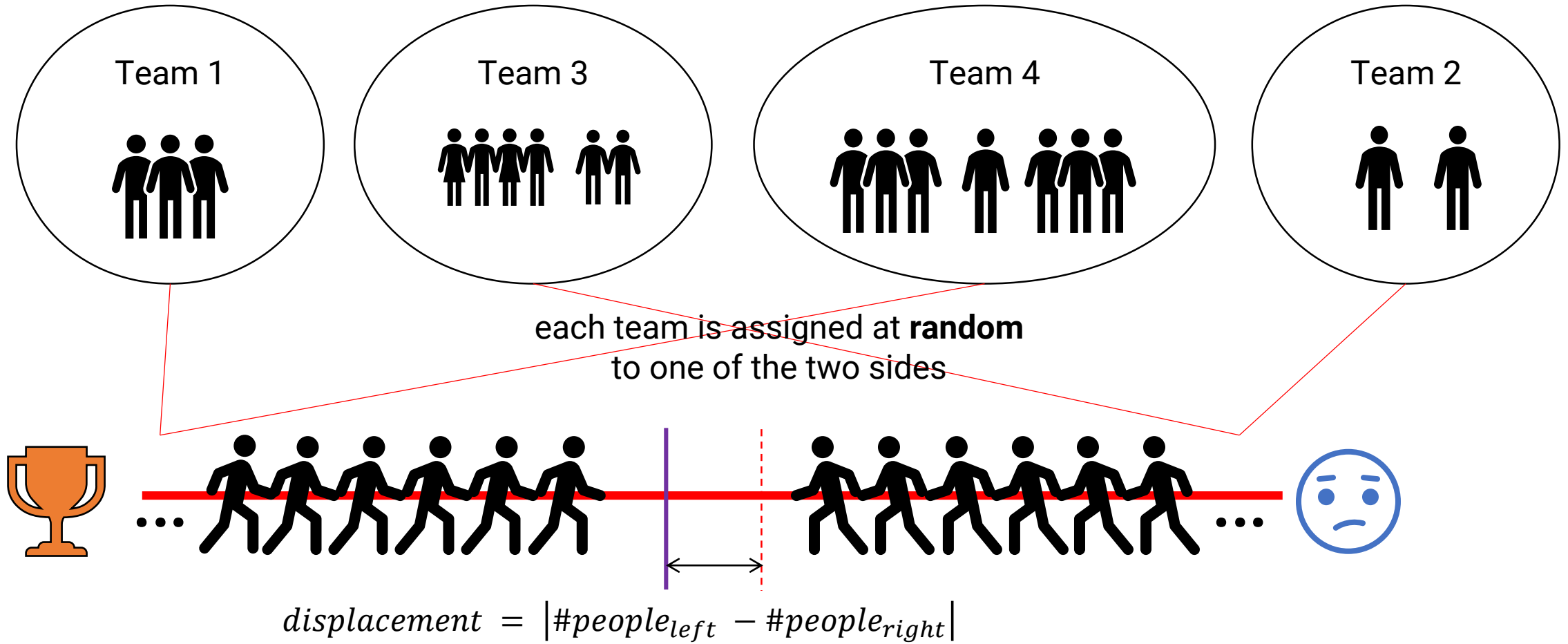
# Tug-of-War



# Tug-of-War

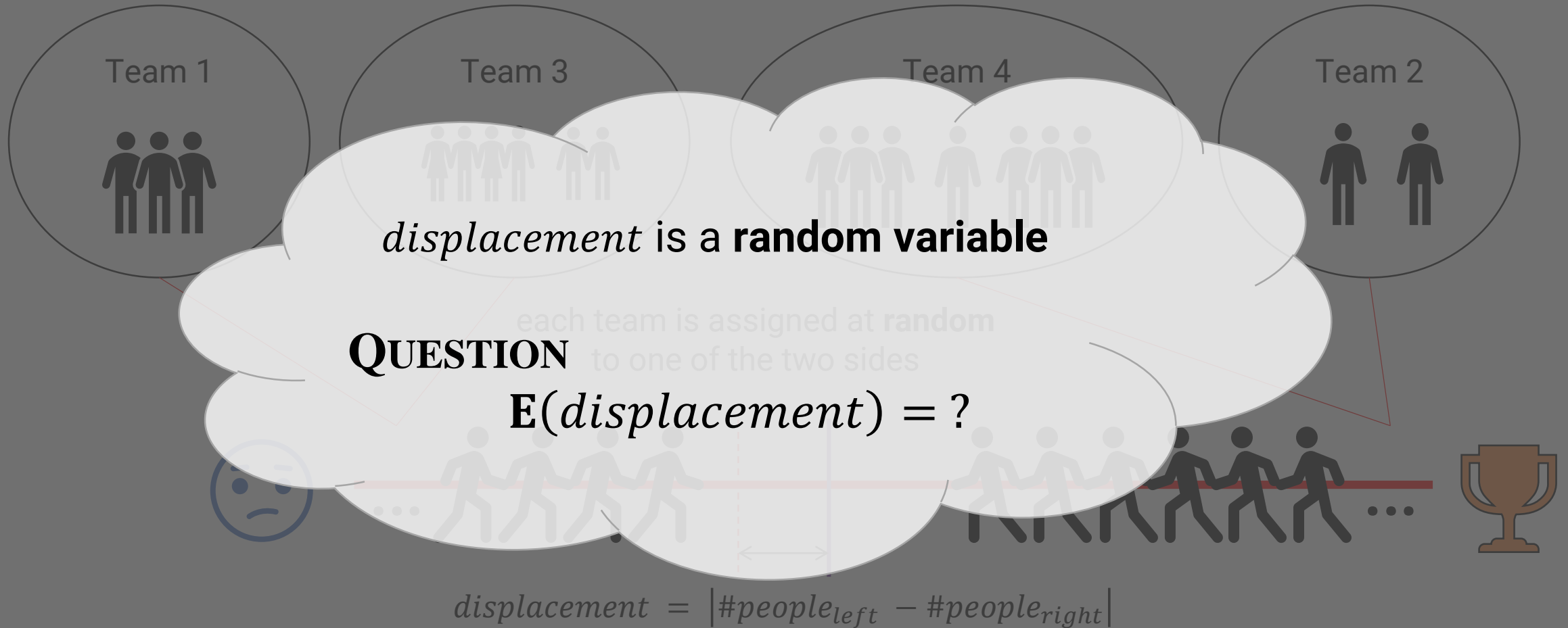


# Tug-of-War





# Tug-of-War



# Tug-of-War

Team 1



Team 3



Team 4



Team 2

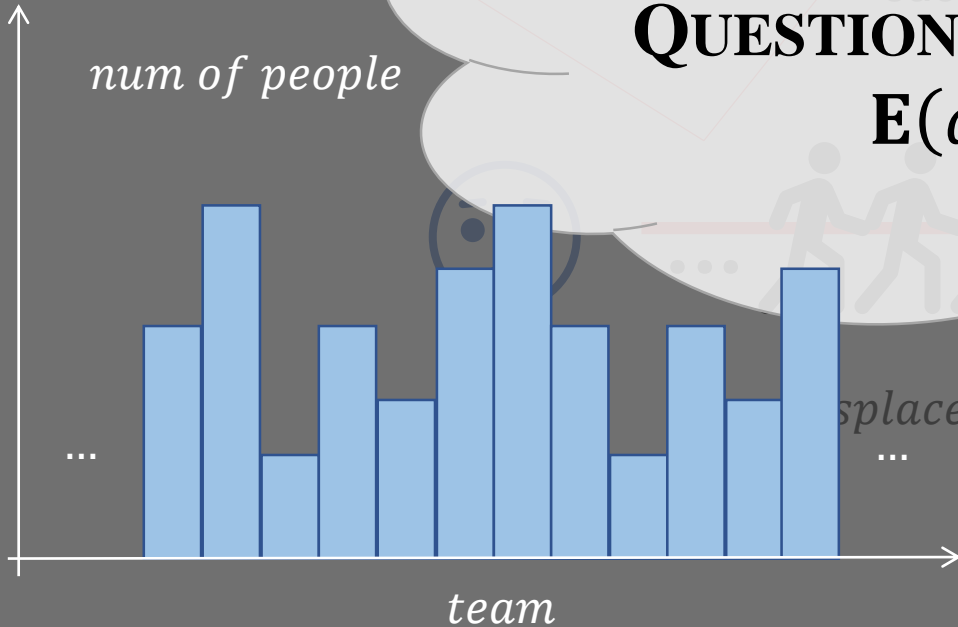


*displacement* is a **random variable**

**QUESTION**

$$E(\text{displacement}) = ? \rightarrow \sqrt{F_2} !!!$$

*num of people*

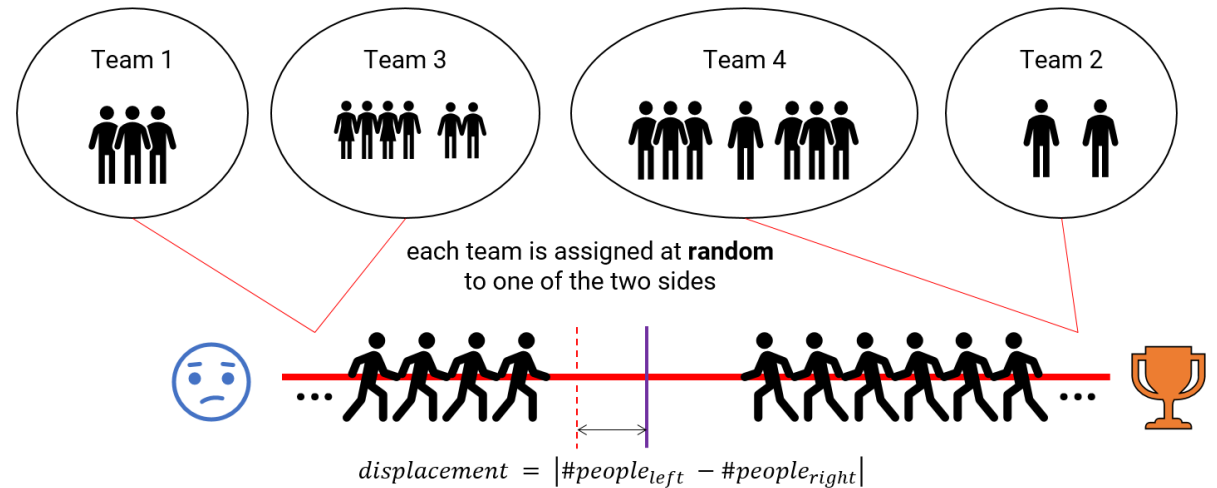


$$\text{displacement} = |\#people_{\text{left}} - \#people_{\text{right}}|$$



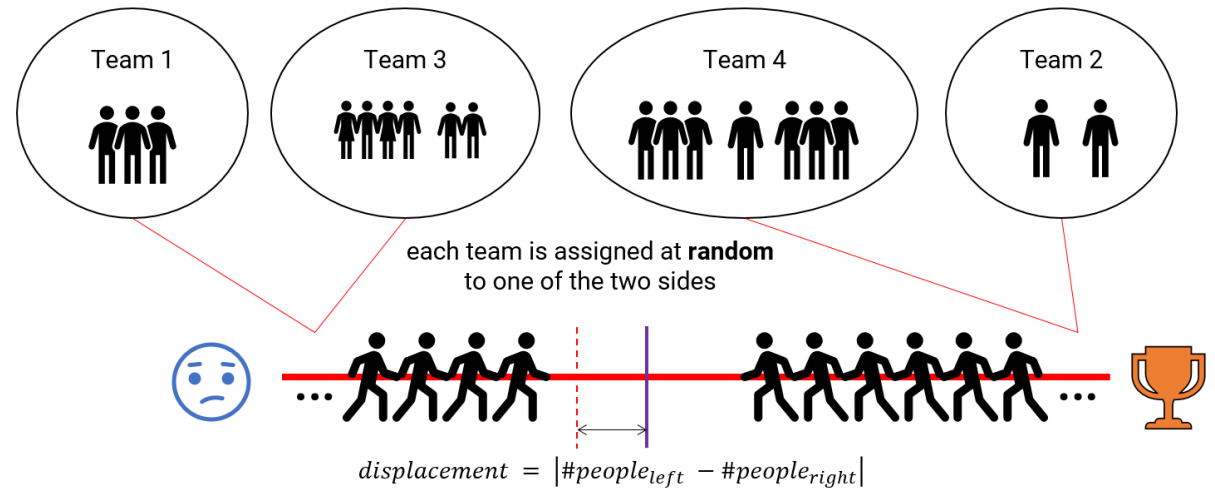
# A Never-Ending Tug-of-War...

- Take a random\* hash function  $h: [1 \dots U] \rightarrow \{-1, +1\}$ 
  - Hash  $TEAM_1, TEAM_2, \dots, TEAM_U$  to  $\{-1, +1\}$



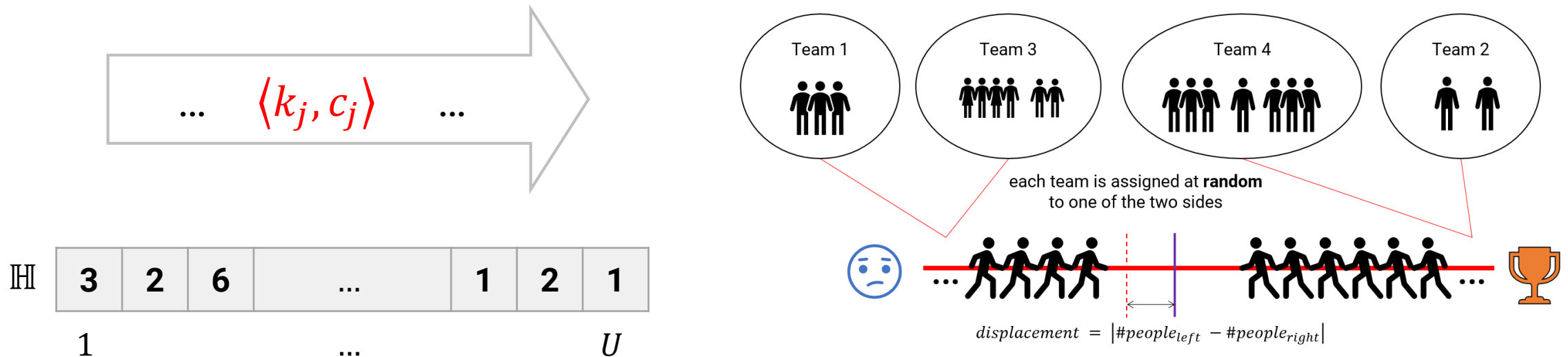
# A Never-Ending Tug-of-War...

- Take a random\* hash function  $h: [1 \dots U] \rightarrow \{-1, +1\}$ 
  - Hash  $TEAM_1, TEAM_2, \dots, TEAM_U$  to  $\{-1, +1\}$
- Maintain  $\mathbf{D} = \sum_{i=1}^U h(i) * \mathbb{H}[i]$

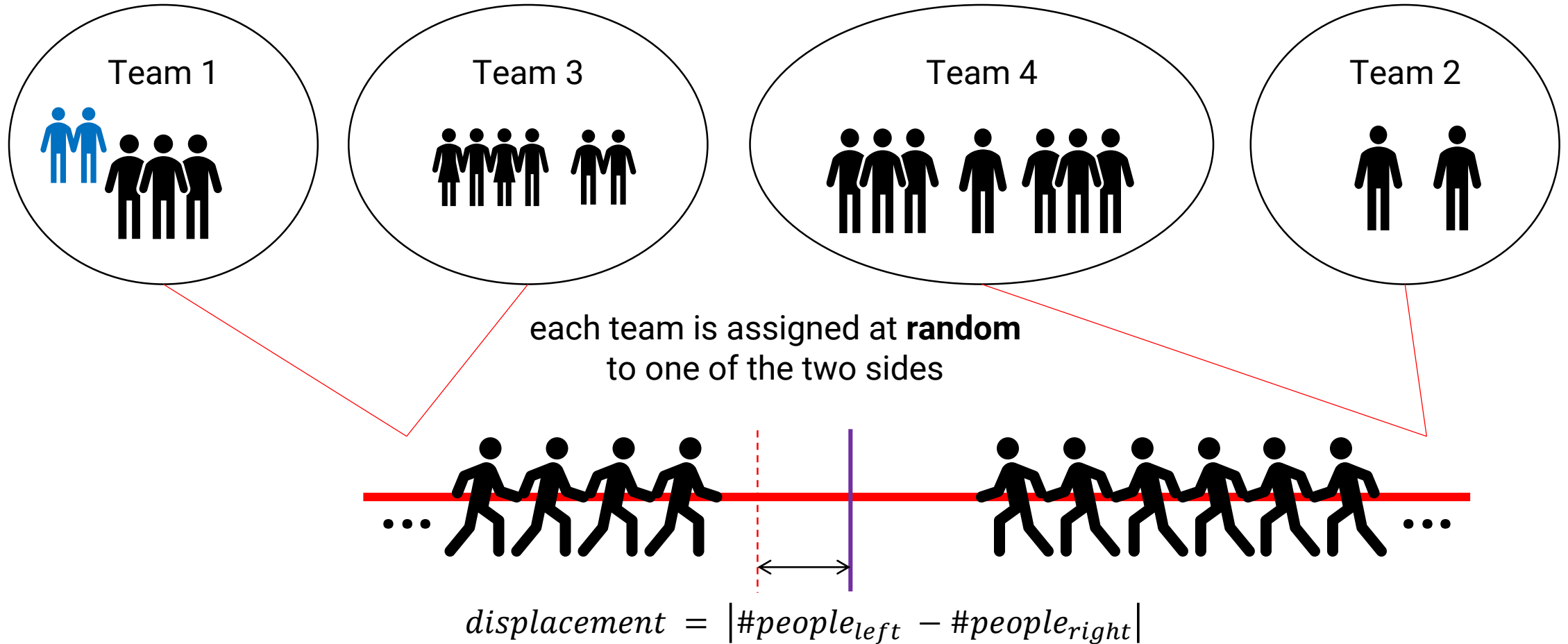


# A Never-Ending Tug-of-War...

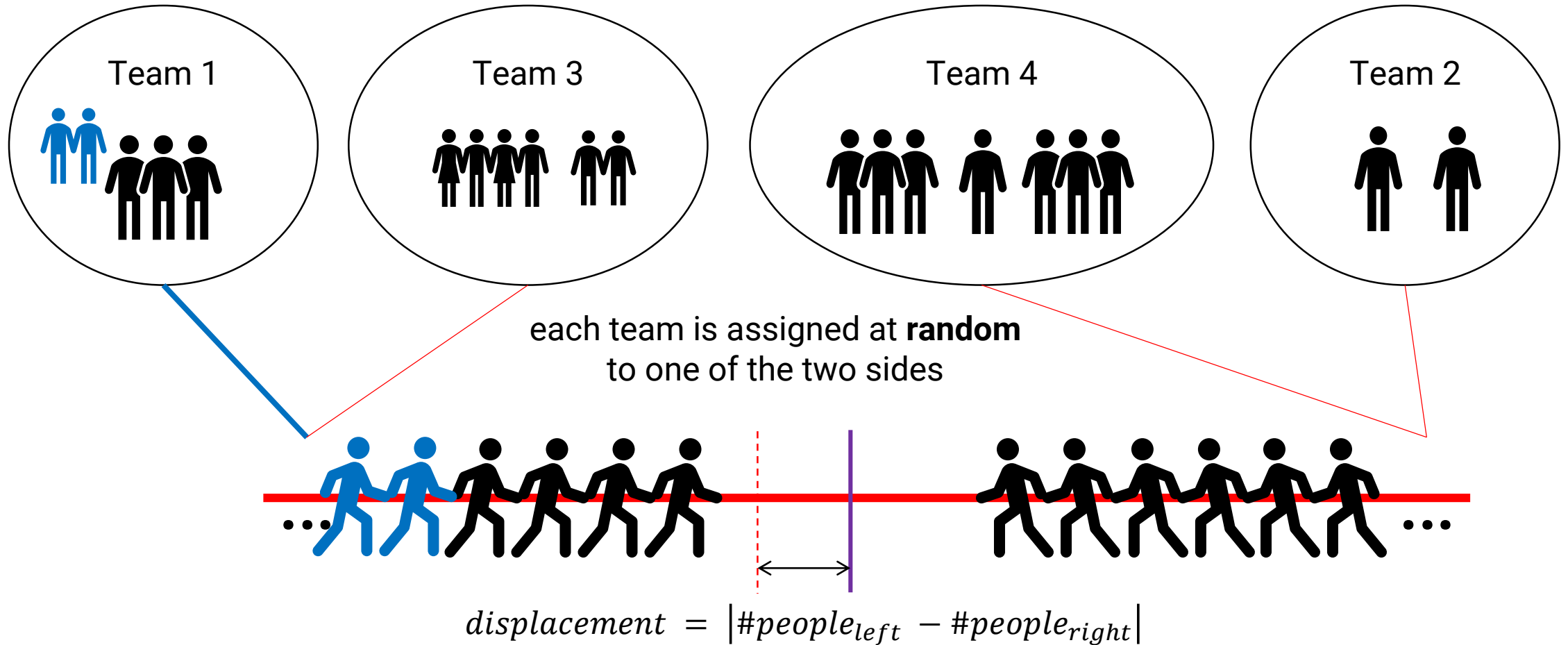
- Take a random\* hash function  $h: [1 \dots U] \rightarrow \{-1, +1\}$ 
  - Hash  $\text{TEAM}_1, \text{TEAM}_2, \dots, \text{TEAM}_U$  to  $\{-1, +1\}$
- Maintain  $\mathbf{D} = \sum_{i=1}^U h(i) * \mathbb{H}[i]$ 
  - Once  $c_j$  new people join team  $k_j$ , we set  $\mathbf{D} \leftarrow \mathbf{D} + h(k_j) * c_j$



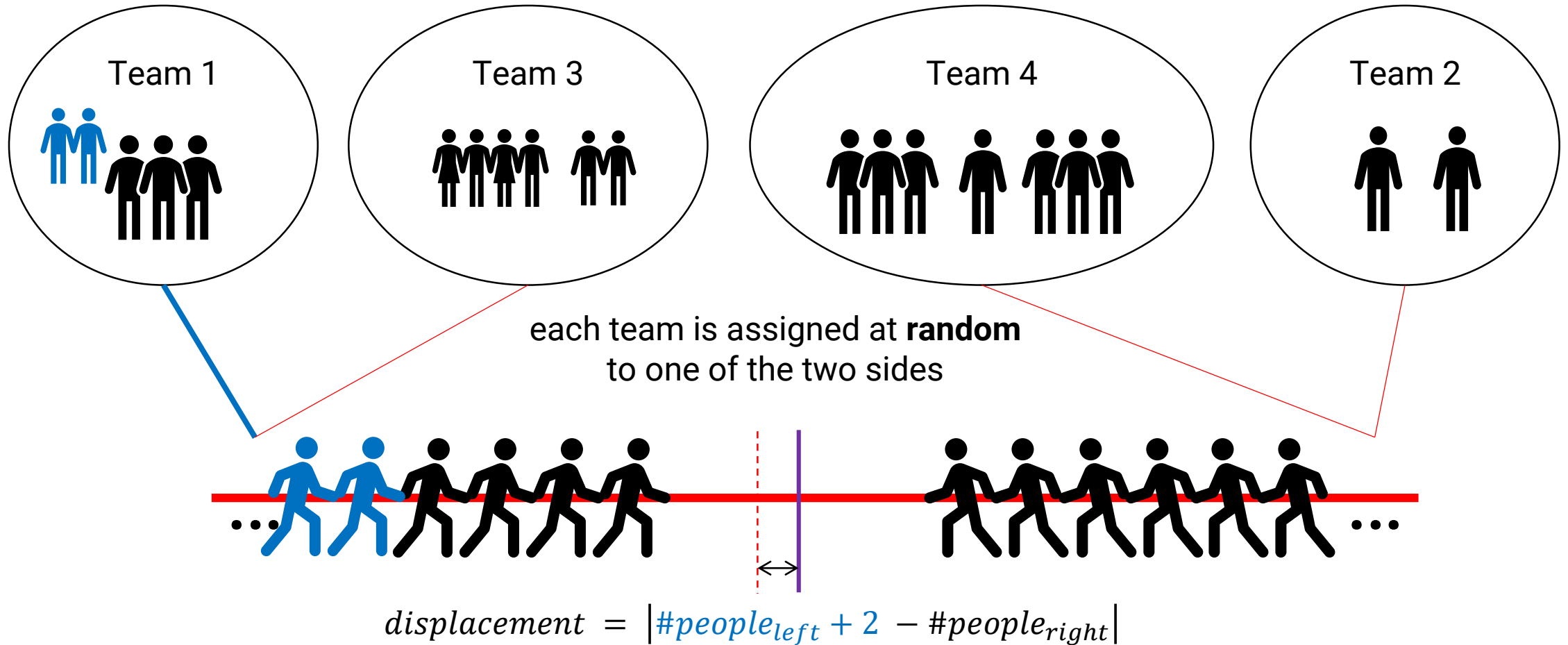
# A Never-Ending Tug-of-War...



# A Never-Ending Tug-of-War...



# A Never-Ending Tug-of-War...





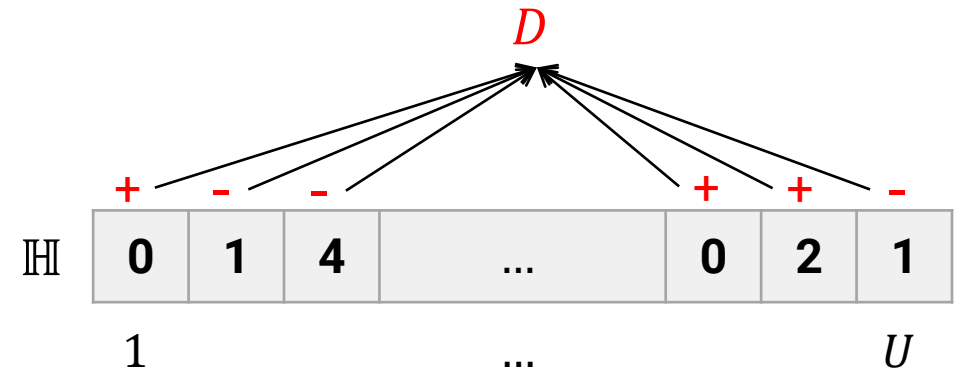
# AMS Sketch

- Take a **4-wise independent** hash function  $h: [1 \dots U] \rightarrow \{-1, +1\}$ 
  - For every  $s_{1\dots 4} \in \{-1, +1\}$  and every distinct  $x_{1\dots 4} \in [1 \dots U]$ :
    - $P[h(x_1) = s_1 \wedge h(x_2) = s_2 \wedge h(x_3) = s_3 \wedge h(x_4) = s_4] = (1/2)^4$

	+	-	-		+	+	-
II	0	1	4	...	0	2	1
	1			...			$U$

# AMS Sketch

- Take a **4-wise independent** hash function  $h: [1 \dots U] \rightarrow \{-1, +1\}$ 
  - For every  $s_1 \dots s_4 \in \{-1, +1\}$  and every distinct  $x_1 \dots x_4 \in [1 \dots U]$ :
    - $P[h(x_1) = s_1 \wedge h(x_2) = s_2 \wedge h(x_3) = s_3 \wedge h(x_4) = s_4] = (1/2)^4$
- Maintain  $D = \sum_{i=1}^U h(i) * \mathbb{H}[i]$ 
  - For each stream update  $\langle k_j, c_j \rangle$ , set  $D \leftarrow D + h(k_j) * c_j$

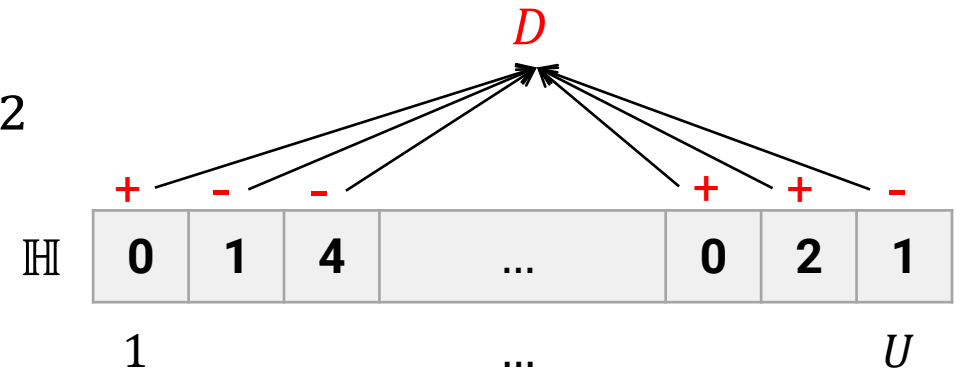


# AMS Sketch

- Take a **4-wise independent** hash function  $h: [1 \dots U] \rightarrow \{-1, +1\}$ 
  - For every  $s_1 \dots s_4 \in \{-1, +1\}$  and every distinct  $x_1 \dots x_4 \in [1 \dots U]$ :
    - $P[h(x_1) = s_1 \wedge h(x_2) = s_2 \wedge h(x_3) = s_3 \wedge h(x_4) = s_4] = (1/2)^4$
- Maintain  $D = \sum_{i=1}^U h(i) * \mathbb{H}[i]$ 
  - For each stream update  $\langle k_j, c_j \rangle$ , set  $D \leftarrow D + h(k_j) * c_j$

- LEMMA  $\mathbf{E}(D^2) = F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2$

- So, we estimate  $\widehat{F_2}(\mathbb{H}) = D^2$



# Proof of Tug-of-War Lemma

- $D = \sum_{i=1}^U h(i) \mathbb{H}[i]$

*2-wise independent*

$$\mathbb{E}(h(i)h(j)) = \mathbb{E}(h(i))\mathbb{E}(h(j)) = 0$$

for  $i \neq j$

$$\begin{aligned}\mathbb{E}(D^2) &= \mathbb{E}\left(\left(\sum_{i=1}^U h(i) \mathbb{H}[i]\right)^2\right) \\ &= \mathbb{E}\left(\sum_{i=1}^U h(i)^2 \mathbb{H}[i]^2\right) + \mathbb{E}\left(\sum_{i \neq j} 2h(i)h(j) \mathbb{H}[i] \mathbb{H}[j]\right) \\ &= \sum_{i=1}^U \mathbb{H}[i]^2 \mathbb{E}(h(i)^2) = \sum_{i=1}^U \mathbb{H}[i]^2 = F_2(\mathbb{H})\end{aligned}$$

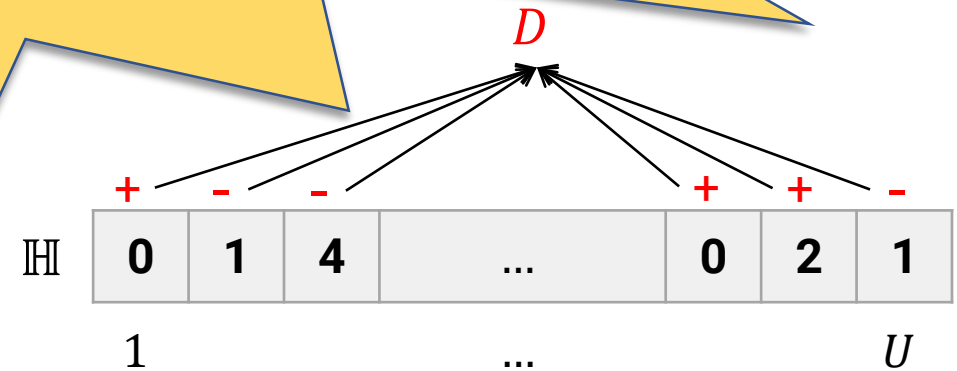
$h(i)^2 = 1$

# AMS Sketch

- Take a **4-wise independent** hash function  $h: [1 \dots U] \rightarrow \{-1, +1\}$ 
  - For every  $x \in [1 \dots U]$ ,  $h(x) \in \{-1, +1\}$
  - $P[h(x_1) = \dots = h(x_k)] = 2^{-k}$

- Maintain **Only one number,  $D$ , need to be stored to estimate  $F_2(\mathbb{H})$ !!!**

- So, we estimate  $\widehat{F}_2(\mathbb{H}) = D^2$



# Wait! **One Random Variable** is Not Enough

- Although  $E(D^2) = F_2(\mathbb{H})$ , maintaining one ***D*** is not enough
- The **variance** of the random variable can be large
  - May observe values varying significantly from the expected value
- Expected values may never be observed in reality
  - Suppose we flip a fair coin and gain \$1 if it's heads, \$0 otherwise
  - $E(\textit{gain}) = 0.5$ , but we never gain \$0.5

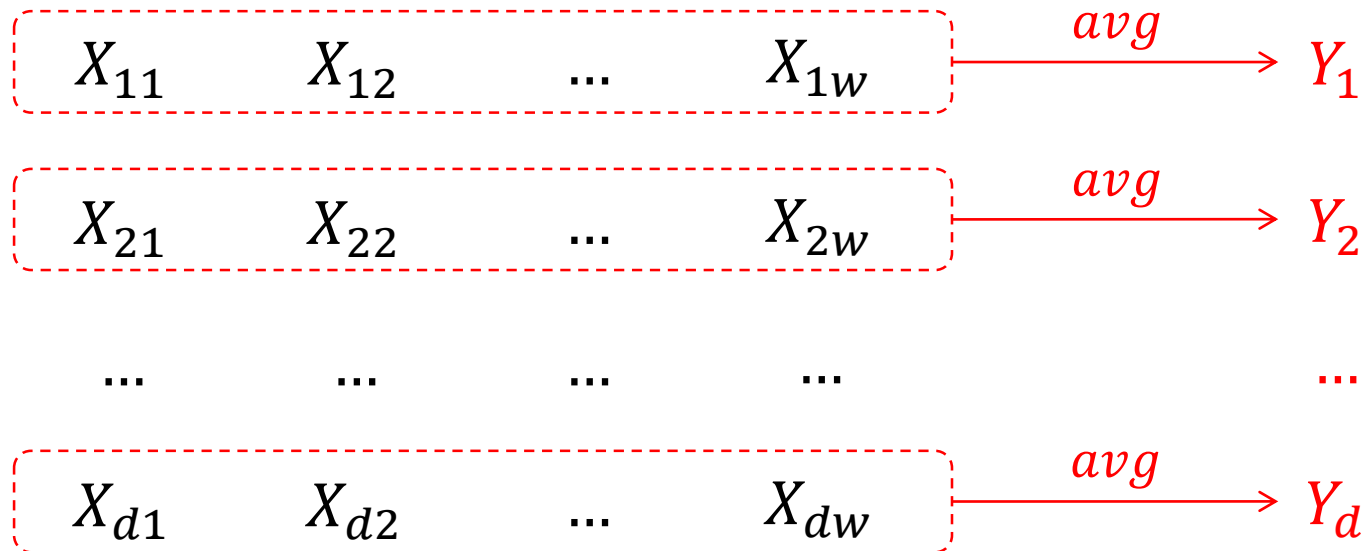
# The Standard Statistical Solution

- Using a matrix of **independent** copies of the random variable

$$\begin{array}{cccc} X_{11} & X_{12} & \dots & X_{1w} \\ X_{21} & X_{22} & \dots & X_{2w} \\ \dots & \dots & \dots & \dots \\ X_{d1} & X_{d2} & \dots & X_{dw} \end{array}$$

# The Standard Statistical Solution

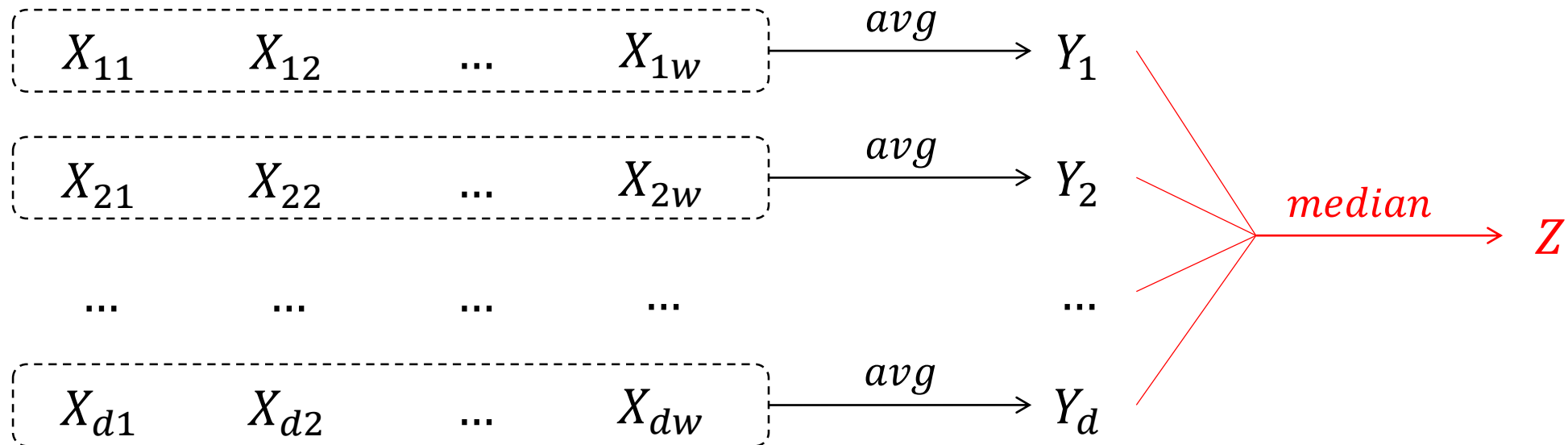
- Using a matrix of **independent** copies of the random variable
- Take the average of each row





# The Standard Statistical Solution

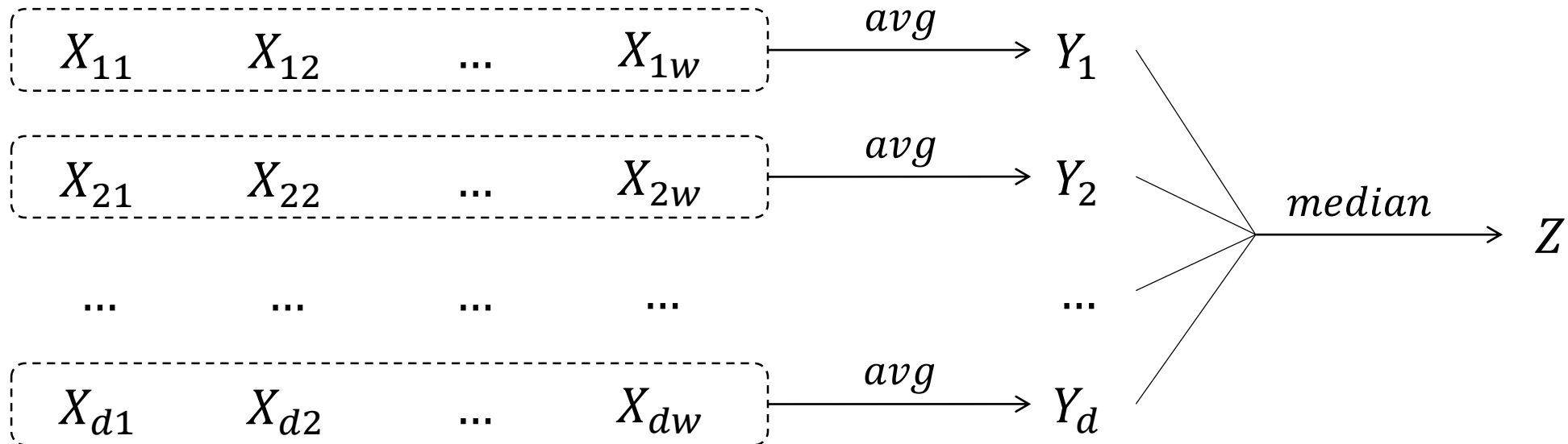
- Using a matrix of **independent** copies of the random variable
- Take the average of each row
- Take the median of averages



# The Standard Statistical Solution

- Using a matrix of **independent** copies of the random variable
- Take the average of each row
- Take the median of averages

why?



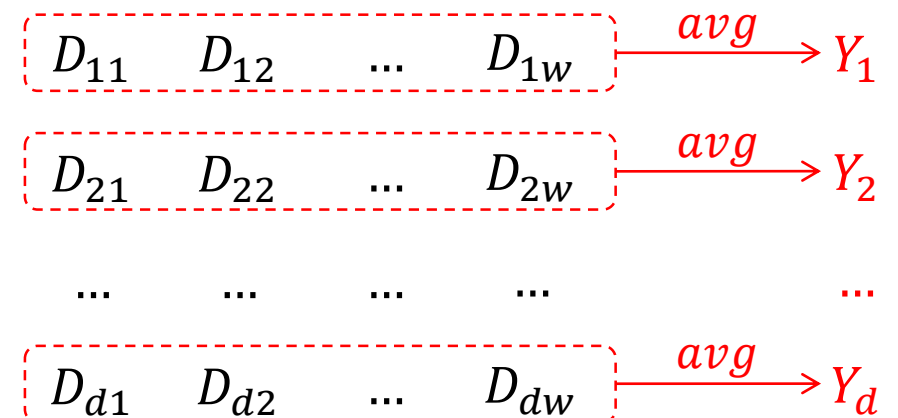
# Averaging: Reducing the Variance

## THEOREM

Let  $X_{i=1\dots n}$  be independent and identically distributed random variables:

$$\text{Var}\left(\frac{1}{n}\sum_{i=1}^n X_i\right) = \frac{1}{n}\text{Var}(X_i)$$

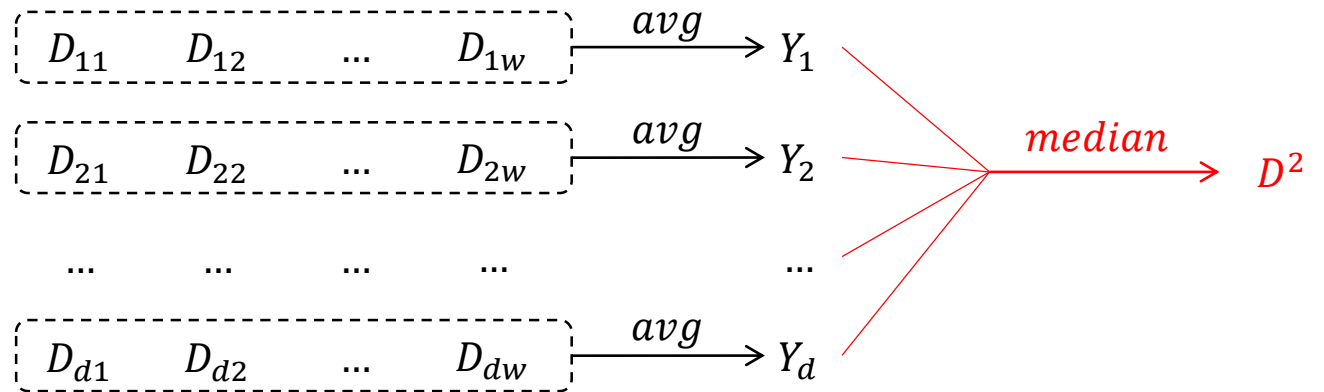
- LEMMA  $\text{Var}(\mathbf{D}^2) \leq 2F_2(\mathbb{H})^2$ 
  - $D^2$  has bounded variance
- Maintain  $w$  independent  $D$ s per row
  - Take their average at query time:
  - $Y_i = \frac{1}{w}\sum_{j=1}^w D_{ij}^2$



# Median: Getting Rid Of Outliers

- An average can be easily skewed by outliers
- Median is more robust to outliers
  - But it only takes discrete values (recall our coin-flipping example)
- So, we take the **median of multiply averages**

- $D^2 = \text{median}_{i=1}^d Y_i$ 
  - where  $Y_i = \left(\frac{1}{w} \sum_{j=1}^w D_{ij}^2\right)$



# The Last Step: Providing An Error Guarantee

- **$(\epsilon, \delta)$ -approximation**: with probability at least  $1 - \delta$ , return an answer  $\widehat{F}_2$  such that

$$(1 - \epsilon)F_2 \leq \widehat{F}_2 \leq (1 + \epsilon)F_2$$

- **#cols**: control the **error bound**
- **#rows**: control the probability that **the error is in this bound**
- Size of the matrix:  $\text{\#rows} = 4 \log(1/\delta)$ ,  $\text{\#cols} = 16/\epsilon^2$ 
  - $(0.1, 0.01)$ -approximation:  $27 \times 1600$

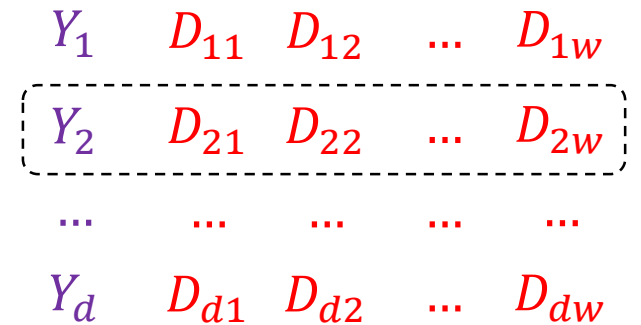
# Bounding the Error

THEOREM (The Chebyshev Inequality)

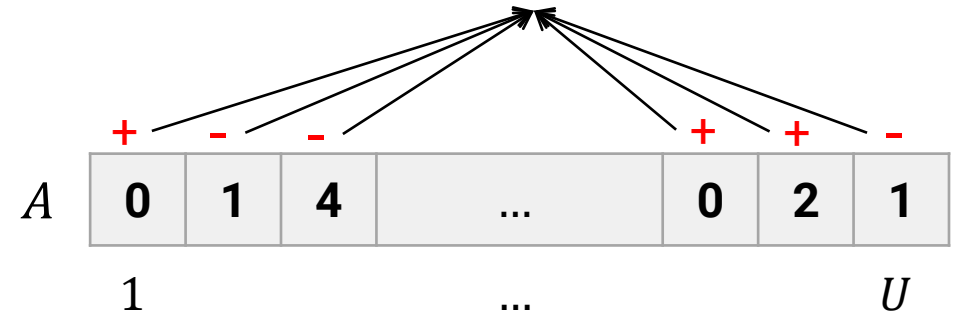
$$\text{Given a random variable } X, \Pr[|X - E(X)| \geq k] \leq \frac{\text{Var}(X)}{k^2}$$

- First, let's look at  $Y_i = \left(\frac{1}{w} \sum_{j=1}^w D_{ij}\right)^2$

$$\Pr[|Y_i - F_2| \leq \epsilon F_2] \leq \frac{\text{Var}(Y_i)}{\epsilon^2 F_2^2} = \frac{\text{Var}(D_{ij}^2)}{w \epsilon^2 F_2^2} \leq \frac{2 F_2^2}{w \epsilon^2 F_2^2} = \frac{2}{w \epsilon^2}$$



- Let  $w = 16/\epsilon^2$ , then prob.  $\leq 1/8$



# Bounding the Error

THEOREM (The Chebyshev Inequality)

Short Version:

For each row of length  $w$ , we can prove that:

$$\Pr[|Y_i - F_2| \leq \epsilon F_2] \leq \frac{2}{w\epsilon^2}$$

For ease of analysis, set  $\frac{2}{w\epsilon^2} = \frac{1}{8}$ , we get  $w = 16/\epsilon^2$

• Let  $w =$



# Bounding the Error (cont.)

THEOREM (Chernoff Bound)

Let  $X_1, X_2, \dots, X_n$  be independent random variables taking values in  $\{0, 1\}$  and  $X = \sum_{i=1}^n X_i$ . Then:

$$\Pr[X \geq (1 + \delta)E(X)] \leq e^{-\delta^2 E(X)/(2+\delta)}, \quad \delta > 0$$

Let indicator variables  $\mathbb{I}_i = \mathbf{1}$  if  $|Y_i - F_2| > \epsilon F_2$ ,  $\mathbb{I}_i = 0$  otherwise.

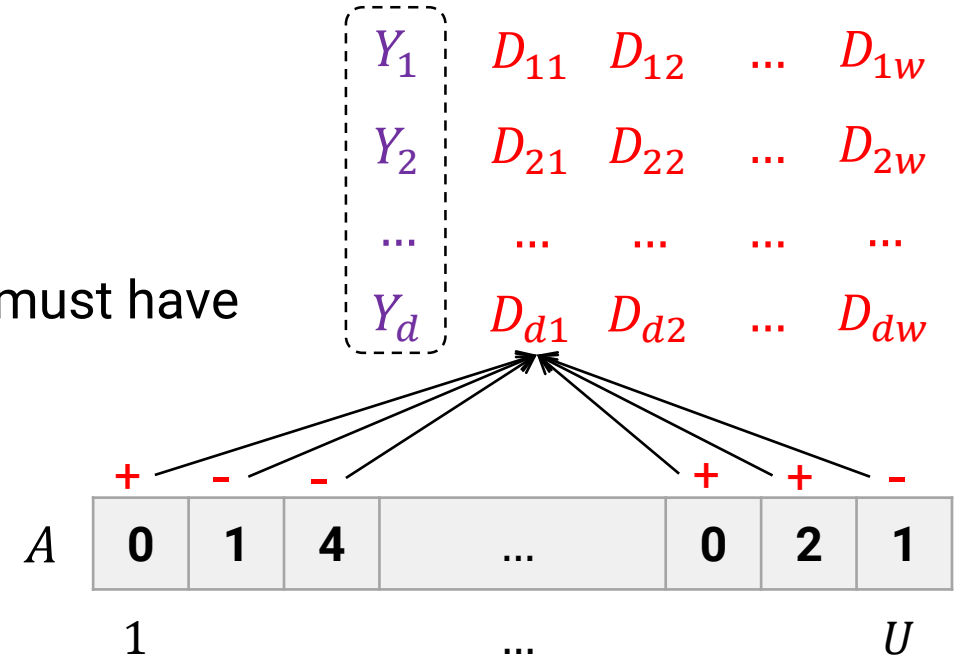
Then  $\Pr[\mathbb{I}_i = 1] \leq 1/8$ ,  $E(\mathbb{I}_i) \leq 1/8$ .

Let  $\mathbb{I} = \sum_{i=1}^d \mathbb{I}_i$ . Then  $E(\mathbb{I}) \leq d/8$

If the median is not within the range, at least half of  $Y_i$ s must have fallen outside the range.

$$\begin{aligned} \Pr[\mathbb{I} \geq d/2] &\leq \Pr[\mathbb{I} \geq (1 + 3)E(\mathbb{I})] \leq e^{-9E(\mathbb{I})/5} \\ &\leq e^{-9d/40} < 2^{-d/4} \end{aligned}$$

Let  $d = 4 \log(1/\delta)$ , then prob.  $< \delta$





# Bounding the Error (cont.)

THEOREM (Chernoff Bound)

Let  $X_1, X_2, \dots, X_n$  be independent random variables, each with  $0 \leq X_i \leq 1$ , and let  $X = X_1 + X_2 + \dots + X_n$ .

Short Version:

For  $d$  rows, we can prove that:

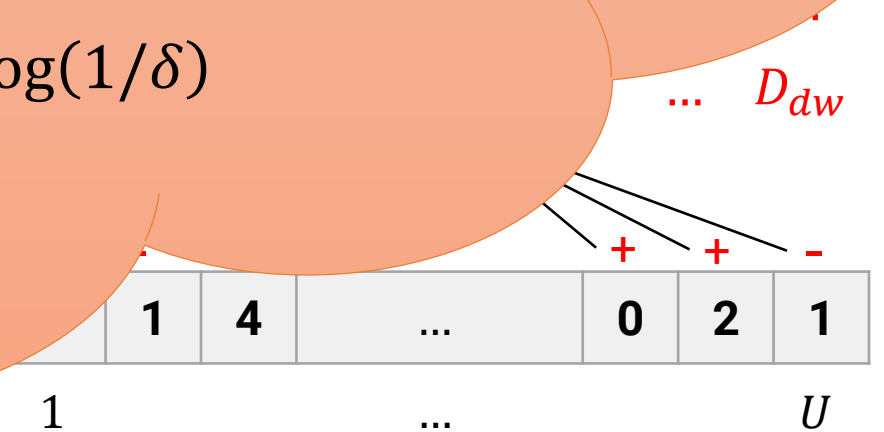
$$\Pr[\text{median is not in the error bound}] < 2^{-d/4}$$

$$\text{set } 2^{-d/4} \text{ to } \delta, \text{ we get } d = 4 \log(1/\delta)$$

$$\Pr[\| \geq d/4]$$

$$\leq \epsilon$$

Let  $d = 4 \log(1/\delta)$ , then prob.  $< \delta$



# The Last Step: Providing An Error Guarantee

- $(\epsilon, \delta)$ -approximation: with probability at least  $1 - \delta$ , return an answer  $\widehat{F}_2$  such that

$$(1 - \epsilon)F_2 \leq \widehat{F}_2 \leq (1 + \epsilon)F_2$$

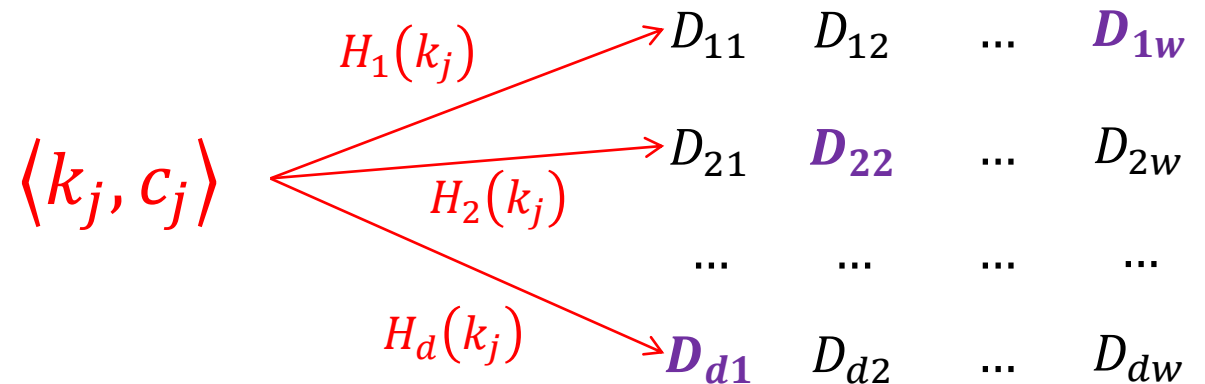
- Size of the matrix:  $\#rows = 4 \log(1/\delta)$ ,  $\#cols = 16/\epsilon^2$ 
  - $(0.1, 0.01)$ -approximation:  $27 \times 1600$

- **High maintenance cost** 😞
  - **each stream element needs to update  $\#rows \times \#cols$  entries**

# Faster Implementation

- For each row of the matrix, instead of updating every variable, we use a hash function to select one variable and update it.

- $H_{1\dots d}: [1 \dots U] \rightarrow [1 \dots w]$



- Take the **sum** of a row (instead of avg)

- $D^2 = \text{median}_{i=1}^d Y_i$ 
  - where  $Y_i = \sum_{j=1}^w D_{ij}^2$

expected value and variance are not changed!!!

# The Complete AMS Sketch Algorithm

```
int          D[d][w];  
HashFunc h[d][w];  
HashFunc H[d];
```

**update**(Z, k, c):

1. for i = 1 to d do
2. pos = H[i](k);
3. D[i][pos] += h[i][pos](k)\*c;

**estimate**(Z):

double sum[d];

1. for i = 1 to d do
2. for j = 1 to w do
3. sum[i] += D[i][j];

4. return median(sum);

# Some Remarks...

- The “AMS” paper has been viewed as one of the foundational works on data streams
  - Led to a sequence of papers in theoretical computer science and data management community
- The pre-mentioned statistical methods are widely used
  - Many sketches have a similar matrix structure

# What Can We Learn from AMS Sketch

Two steps for designing a sketch:

- Design an estimator for the target
  - e.g., tug-of-war for  $F_2$
  - the estimator should be maintainable on data streams
  - THE HARDEST PART
- Maintain multiple independent copies of the estimator

# Outline

- Data Stream Model
- Sketch 101: AMS Sketch
- **The Magic of AMS Sketch**
- Counting Distinct Elements
- Finding Heavy Hitters

<b>Point Query</b>	$\mathbb{H}[i]$
<b>Range Query</b>	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

<b>Distinct Elements</b>	$F_0(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^0$
<b>Sum</b>	$F_1(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]$
<b>Sum of Squares</b>	$F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2$
<b>Heavy Hitters</b>	$\left\{ i: \mathbb{H}[i] \geq \varepsilon \sum_{k=1}^U \mathbb{H}[k] \right\}$
<b>Top-K</b>	

# Dot Product of Histograms

- $F_2$  is actually is the dot product of the histogram with itself
  - i.e.,  $F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2 = \mathbb{H} \cdot \mathbb{H}$

$\mathbb{H}$	0	1	4	...	0	2	1
$\mathbb{H}$	0	1	4	...	0	2	1
	1			...			$U$

The dot product of two vectors  $\mathbf{a} = [a_1, a_2, \dots, a_n]$ , vector  $\mathbf{b} = [b_1, b_2, \dots, b_n]$  is defined as:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$$



# Dot Product of Histograms

- $F_2$  is actually is the dot product of the histogram with itself
  - i.e.,  $F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2 = \mathbb{H} \cdot \mathbb{H}$
  - can be viewed as the **self-join size**

$\mathbb{H}$	0	1	4	...	0	2	1
$\mathbb{H}$	0	1	4	...	0	2	1
	1			...			$U$

```
SELECT  A.id, B.id, A.salary - B.salary
FROM    employee A JOIN employee B
        USING (department)
WHERE   A.id < B.id
```

employee		
id	salary	department
1	10000	A
2	9000	A
3	12000	B
4	9999	C
5	12345	C
6	11111	C

$\mathbb{H}$	2	1	3
	A	B	C

$2 * 2$   
 $1 * 1$   
 $3 * 3$

$F_2(\mathbb{H}) = \sum_i \mathbb{H}[i]^2$

# Dot Product of Histograms

- $F_2$  is actually is the dot product of the histogram with itself
  - i.e.,  $F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2 = \mathbb{H} \cdot \mathbb{H}$
  - can be viewed as the **self-join size**
- What is the dot product of two different histograms?
  - $\mathbb{H}_1 \cdot \mathbb{H}_2 = \sum_{i=1}^U \mathbb{H}_1[i] \mathbb{H}_2[i]$

$\mathbb{H}_1$	1	3	1	...	2	1	1
$\mathbb{H}_2$	0	1	4	...	0	2	1
	1			...			$U$

# Dot Product of Histograms

- $F_2$  is actually is the dot product of the histogram with itself
  - i.e.,  $F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2 = \mathbb{H} \cdot \mathbb{H}$
  - can be viewed as the **self-join size**

- What is the dot product of two different histograms?

- $\mathbb{H}_1 \cdot \mathbb{H}_2 = \sum_{i=1}^U \mathbb{H}_1[i] \mathbb{H}_2[i]$

- **join size !!!**

$\mathbb{H}_1$	1	3	1	...	2	1	1
$\mathbb{H}_2$	0	1	4	...	0	2	1
	1			...			$U$

# Join Size

$$|R \bowtie S| = \mathbb{H}_1 \cdot \mathbb{H}_2$$

 $\mathbb{H}_1$ 

1	2	1	1	1
---	---	---	---	---

 $\mathbb{H}_2$ 

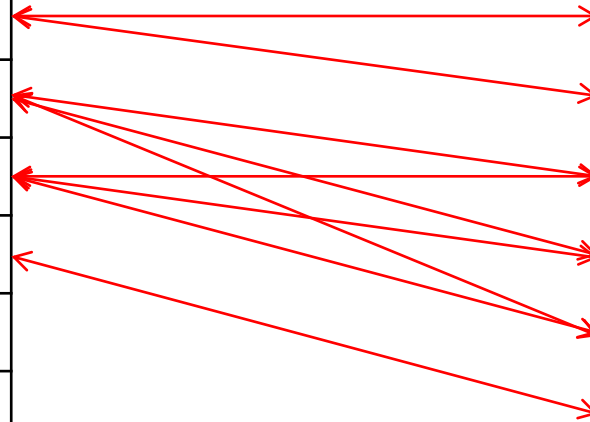
2	3	1	0	0
---	---	---	---	---

Price	GuestID
100.00	1
90.00	2
120.00	2
120.00	3
123.50	4
111.00	5

*payments*

GuestID	Room
1	A
1	B
2	C
2	D
2	E
3	F

*reservations*



# Estimating the Dot Product

- Recall our AMS sketch is  $D = \sum_{i=1}^U h(i)\mathbb{H}(i)$  and:

$$\mathbb{E}(D^2) = \mathbb{E}(D * D) = F_2(\mathbb{H}) = \mathbb{H} \cdot \mathbb{H}$$

# Estimating the Dot Product

- Recall our AMS sketch is  $D = \sum_{i=1}^U h(i)\mathbb{H}(i)$  and:

$$\mathbb{E}(D^2) = \mathbb{E}(D * D) = F_2(\mathbb{H}) = \mathbb{H} \cdot \mathbb{H}$$

- Now assume we have one sketch for each of  $\mathbb{H}_1$  and  $\mathbb{H}_2$

$\mathbb{H}_1$	1	2	1	1	1
----------------	---	---	---	---	---

$$D_1 = \sum_{i=1}^U h(i)\mathbb{H}_1(i)$$

$\mathbb{H}_2$	2	3	1	0	0
----------------	---	---	---	---	---

$$D_2 = \sum_{i=1}^U h(i)\mathbb{H}_2(i)$$

# Estimating the Dot Product

- Recall our AMS sketch is  $D = \sum_{i=1}^U h(i)\mathbb{H}(i)$  and:

$$\mathbb{E}(D^2) = \mathbb{E}(D * D) = F_2(\mathbb{H}) = \mathbb{H} \cdot \mathbb{H}$$

- Now assume we have one sketch for each of  $\mathbb{H}_1$  and  $\mathbb{H}_2$

$\mathbb{H}_1$	1	2	1	1	1
----------------	---	---	---	---	---

LEMMA

$\mathbb{H}_2$	2	3	1	0	0
----------------	---	---	---	---	---

$$D_1 = \sum_{i=1}^U h(i)\mathbb{H}_1(i)$$

$$\mathbb{E}(D_1 * D_2) = \mathbb{H}_1 \cdot \mathbb{H}_2$$

$$D_2 = \sum_{i=1}^U h(i)\mathbb{H}_2(i)$$

# Application: Join Size Estimation

- One of the fundamental problems in query optimization
- $\mathbf{E}(D_R * D_S) = \mathbb{H}_R \cdot \mathbb{H}_S = |R \bowtie S|$
- Thus, we maintain one AMS sketch for each table
  - Take the median of averages of **products of atomic sketches**
    - i.e.,  $\widehat{\mathbb{H}}[\mathbf{k}] = \text{median}_i \sum_{j=1}^w h_{ij}(k) * D_{ij}$

$D_{11}$	$D_{12}$	...	$D_{1w}$
$D_{21}$	$D_{22}$	...	$D_{2w}$
...	...	...	...
$D_{d1}$	$D_{d2}$	...	$D_{dw}$

sketch for  $R$

$D_{11}$	$D_{12}$	...	$D_{1w}$
$D_{21}$	$D_{22}$	...	$D_{2w}$
...	...	...	...
$D_{d1}$	$D_{d2}$	...	$D_{dw}$

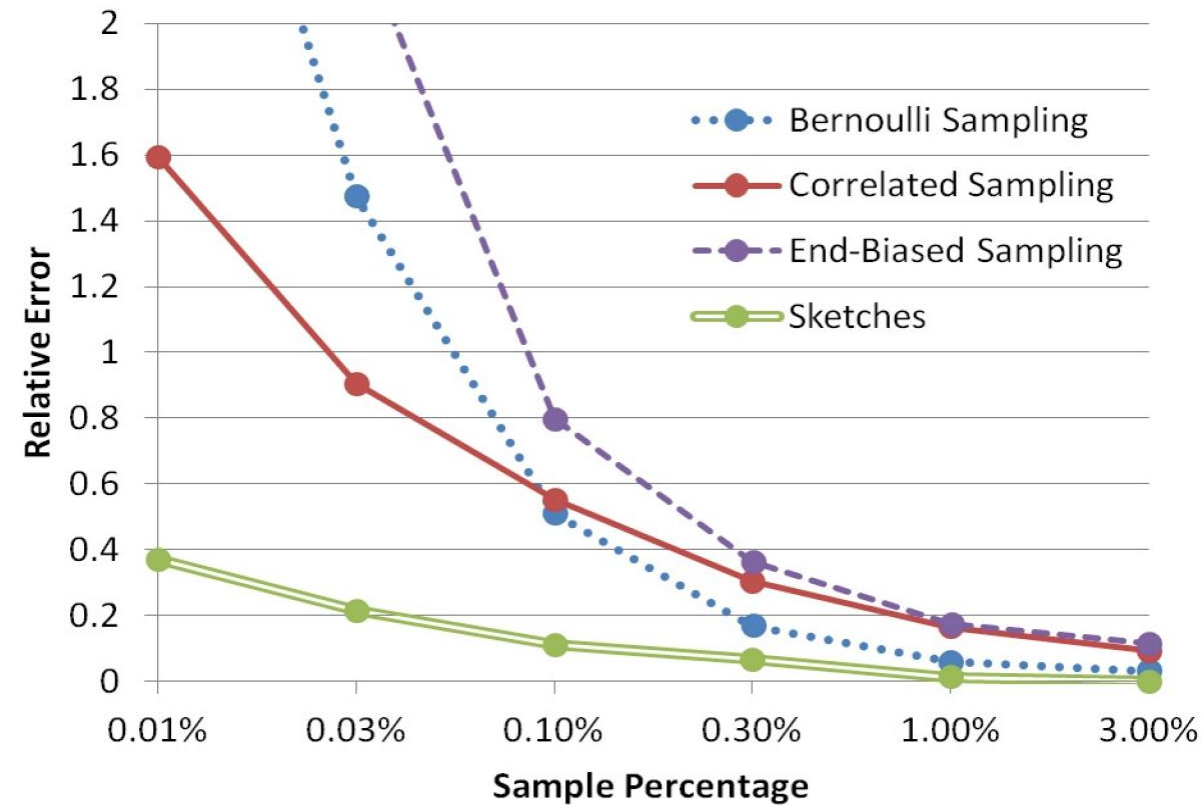
sketch for  $S$

$$\mathbf{E}(D_1 * D_2) = \mathbb{H}_1 \cdot \mathbb{H}_2$$



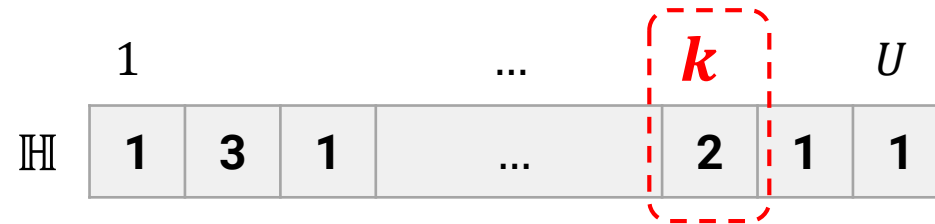
# Application: Join Size Estimation

- Can produce very good estimations!



# Point Queries

- Recall point queries:  $\mathbb{H}[k]$



# Point Queries

- Recall point queries:  $\mathbb{H}[k]$
- Note that point queries can be expressed as **dot products**
  - $\mathbb{H}[k] = \mathbb{H} \cdot \mathbb{H}_k$

	1		...		<i><b>k</b></i>		U
$\mathbb{H}$	1	3	1	...	2	1	1
$\mathbb{H}_k$	0	0	0	...	1	0	0

# Point Queries

- Recall point queries:  $\mathbb{H}[k]$
- Note that point queries can be expressed as **dot products**
  - $\mathbb{H}[k] = \mathbb{H} \cdot \mathbb{H}_k$

	1		...		<i><b>k</b></i>		U
$\mathbb{H}$	1	3	1	...	2	1	1
$\mathbb{H}_k$	0	0	0	...	1	0	0

$$D = \sum_{i=1}^U h(i) \mathbb{H}_k(i)$$

$$D_k = \sum_{i=1}^U h(i) \mathbb{H}_k(i) = h(k)$$

$$E(D_1 * D_2) = \mathbb{H}_1 \cdot \mathbb{H}_2$$

# Point Queries

- Recall point queries:  $\mathbb{H}[k]$
- Note that point queries can be expressed as **dot products**
  - $\mathbb{H}[k] = \mathbb{H} \cdot \mathbb{H}_k$

	1		...		<i><b>k</b></i>		U
$\mathbb{H}$	1	3	1	...	2	1	1
$\mathbb{H}_k$	0	0	0	...	1	0	0

$$D = \sum_{i=1}^U h(i) \mathbb{H}_k(i)$$

$$D_k = \sum_{i=1}^U h(i) \mathbb{H}_k(i) = h(k)$$

$$\mathbb{E}(D * D_k) = \mathbf{E}(D * h(k)) = \mathbb{H} \cdot \mathbb{H}_k = \mathbb{H}[k]$$

$$\mathbb{E}(D_1 * D_2) = \mathbb{H}_1 \cdot \mathbb{H}_2$$

# Point Queries: Details

- Maintain one AMS sketch for a histogram
- Take the median of averages of  $h_{ij}(k) * D_{ij}$ 
  - i.e.,  $\widehat{\mathbb{H}}[\mathbf{k}] = \text{median}_i \sum_{j=1}^w h_{ij}(k) * D_{ij}$

	1		...		<b><i>k</i></b>		<i>U</i>
$\mathbb{H}$	1	3	1	...	2	1	1

$D_{11}$	$D_{12}$	...	$D_{1w}$
$D_{21}$	$D_{22}$	...	$D_{2w}$
...	...	...	...
$D_{d1}$	$D_{d2}$	...	$D_{dw}$

sketch for  $\mathbb{H}$

$$\mathbf{E}(h(\mathbf{k}) * \mathbf{D}) = \mathbb{H}[\mathbf{k}]$$

# Point Queries: Error Guarantee

- **$(\epsilon, \delta)$ -approximation**: using the same AMS sketch structure, with probability at least  $1 - \delta$ , return an answer  $\widehat{\mathbb{H}}[k]$  such that

$$|\widehat{\mathbb{H}}[k] - \mathbb{H}[k]| \leq \epsilon \sqrt{F_2(\mathbb{H})}$$

# Point Queries: Error Guarantee

- **$(\epsilon, \delta)$ -approximation**: using the same AMS sketch structure, with probability at least  $1 - \delta$ , return an answer  $\widehat{\mathbb{H}}[k]$  such that

$$|\widehat{\mathbb{H}}[k] - \mathbb{H}[k]| \leq \epsilon \sqrt{F_2(\mathbb{H})}$$

- This error guarantee is only meaningful for **heavy hitters**
  - Because  $\sqrt{F_2(\mathbb{H})}$  can be large



# Point Queries: Error Guarantee

- **$(\epsilon, \delta)$ -approximation**: using the same AMS sketch structure, with probability at least  $1 - \delta$ , return an answer  $\widehat{\mathbb{H}}[k]$  such that

$$|\widehat{\mathbb{H}}[k] - \mathbb{H}[k]| \leq \epsilon \sqrt{F_2(\mathbb{H})}$$

- This error guarantee is only meaningful for **heavy hitters**

- Because  $\sqrt{F_2(\mathbb{H})}$  can be large

- But this is a quite tight error guarantee

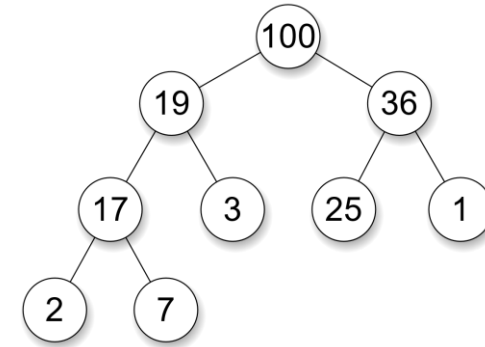
- Because other algorithms provide a  $\epsilon F_1(\mathbb{H})$  guarantee,  $F_1(\mathbb{H}) \geq \sqrt{F_2(\mathbb{H})}$   
count-min sketch

$$(\mathbb{H}[1] + \mathbb{H}[2])^2 \geq \mathbb{H}[1]^2 + \mathbb{H}[2]^2$$

# Finding Heavy Hitters/Top-K

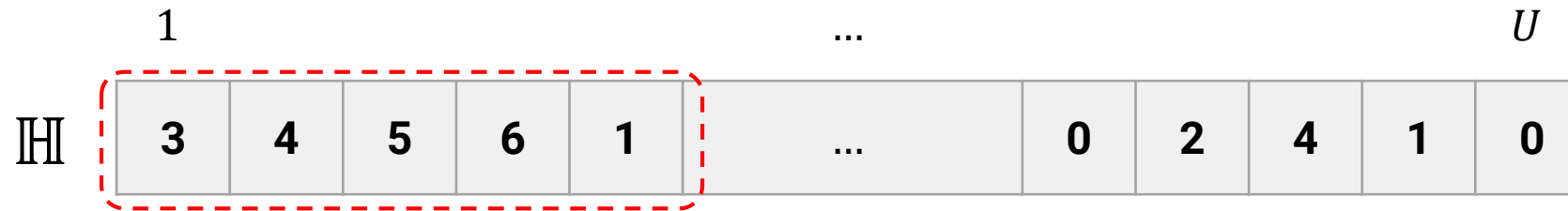
- Maintain a **heap** containing the top-k elements seen so far
  - And their estimated counts

- When a new element is encountered:
  - Update the sketch
  - If the element is already in the heap, its count is incremented
  - Otherwise, obtain an estimated count from the sketch and try to add the element to the heap



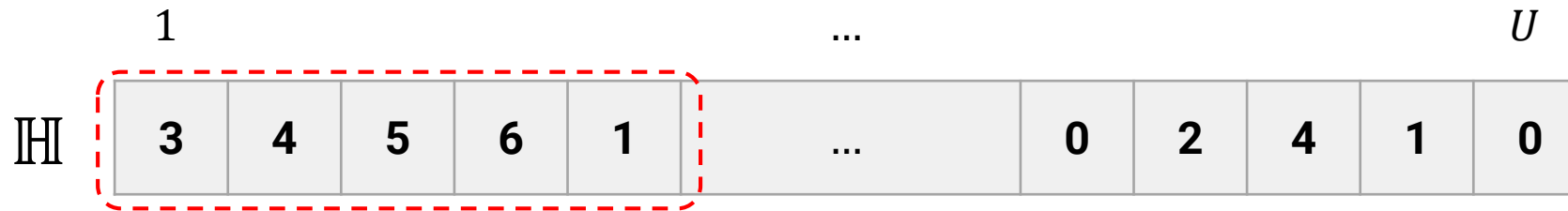
# Range Queries

- Recall our range queries are  $\sum_{i=t}^{t+w} \mathbb{H}[i]$



# Range Queries

- Recall our range queries are  $\sum_{i=t}^{t+w} \mathbb{H}[i]$



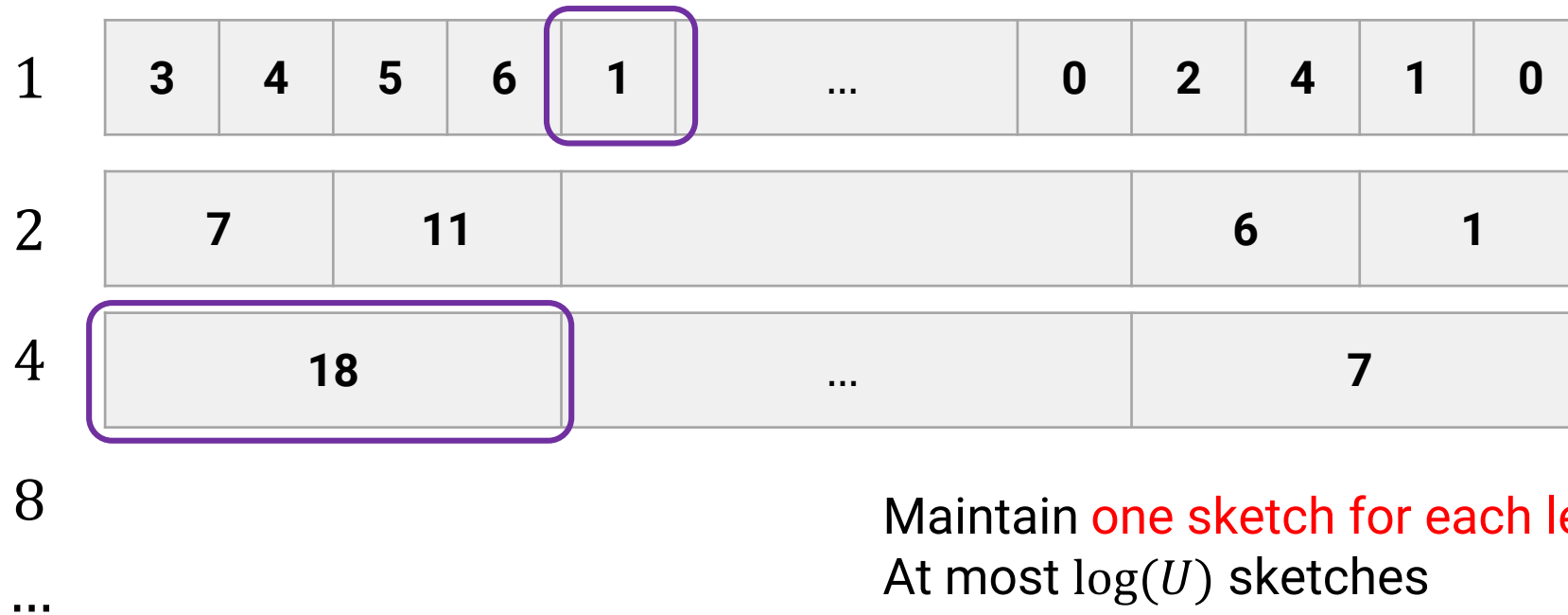
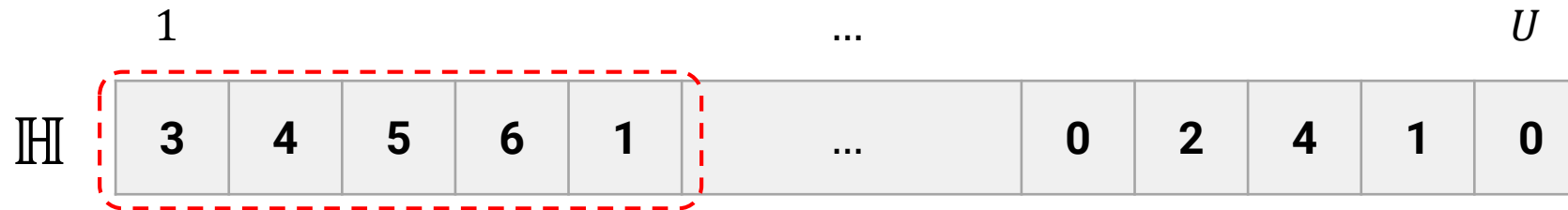
- Naïve Idea: take the sum of point queries



the error increases linearly with the length of the range!!!

# Range Queries

- Standard approach: decomposing to **dyadic ranges**



# Outline

- Data Stream Model
- Sketch 101: AMS Sketch
- The Magic of AMS Sketch
- Counting Distinct Elements
- Finding Heavy Hitters

Point Query	$\mathbb{H}[i]$
Range Query	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

Distinct Elements	$F_0(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^0$
Sum	$F_1(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]$
Sum of Squares	$F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2$
Heavy Hitters	$\left\{ i: \mathbb{H}[i] \geq \varepsilon \sum_{k=1}^U \mathbb{H}[k] \right\}$
Top-K	

# Counting Distinct Values **Exactly**

- Idea: use a hash table to record all observed unique values

## ALGORITHM

```
1. H = new HashTable();  
2. while(v = stream.next())  
3.     H.set(v);  
4. return H.size();
```

- Assume we know the number of distinct values,  $U$ 
  - $U$ -bit bitmap with ideal hash function

# Adding Some Approximation: **Linear Counting**

- Bitmap  $M$  + hash function  $h$ 
  - $\#bits(M) = m < U$
  - $h$  maps each value  $v$  uniformly at random to an  $i \in [1, m]$

ALGORITHM

```
1. M = new Bitmap(m);
2. h = new HashFunc(m);

3. while(v = stream.next())
4.     M.set(h(v));

5. z = M.num_of_zero();
6. return  $m * \ln(m/z)$ ;
```

$$E(z) = m \left(1 - \frac{1}{m}\right)^{F_0} \\ \approx m e^{-F_0/m}$$

⇓

$$\widehat{F_0} = -m \ln \frac{z}{m}$$



# Comments on Linear Counting

- Small estimation error, but...
  - Require linear space
  - Require prior knowledge on  $F_0$ 
    - When the load factor is high, the error can be large
- Used in HYPERLOGLOG and Google's HYPERLOGLOG++
  - For estimating small cardinalities
  - HLL: 40960, HLL++: 11500
- Not Attractive for large cardinalities

# Break the Linearity: Flajolet-Martin Method

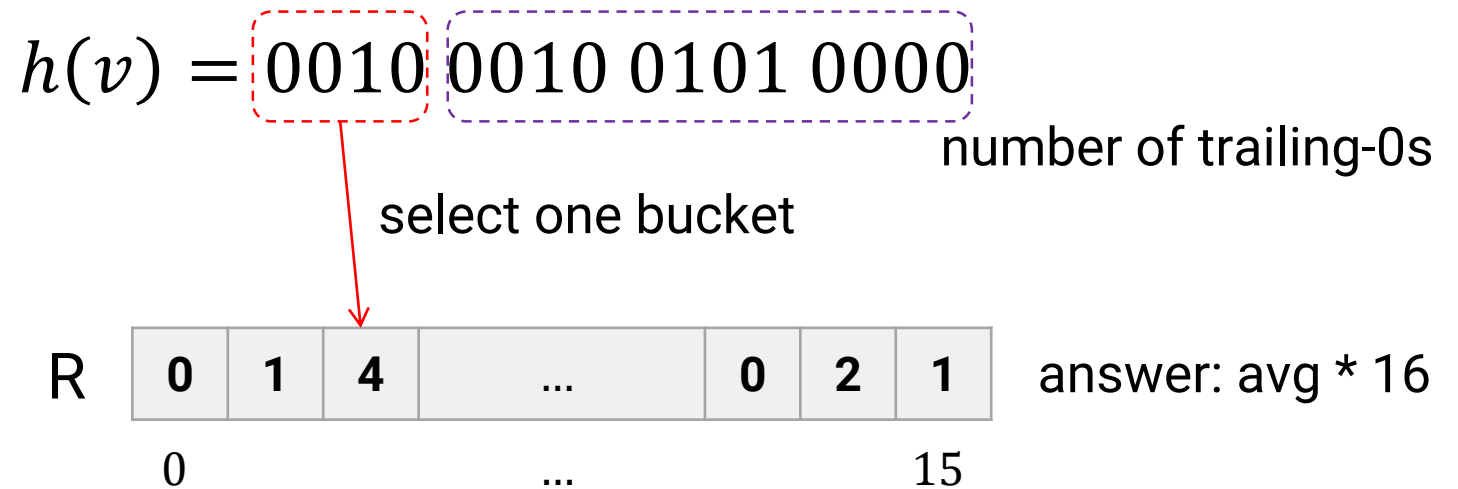
- Hash function  $h$ : maps each element to  $\log_2 N$  bits (e.g., 64-bit)
  - $N$  is the length of the stream
- For each stream element  $v$ 
  - Let  $z(h(v)) = \text{number of trailing 0s}$  in the binary representation of  $h(v)$
  - Record  $R =$  the maximum  $z(h(v))$  seen so far
- Estimated number of distinct elements  $= 2^R / 0.77351$

# Why FM Algorithm Works?

- $h$  hashes  $v$  with equal probability
  - About 50% of  $v$ s hash to \*\*\*\*\*0 (1/2)
  - About 25% of  $v$ s hash to \*\*\*\*\*00 (1/4)
  - About 12.5% of  $v$ s hash to \*\*\*\*\*000 (1/8)
  - About 6.25% of  $v$ s hash to \*\*\*\*\*0000 (1/16)
  - $\Pr[\text{a random } h(v) \text{ ends in at least } z \text{ zeros}] = 1/2^z$
- When we observed  $z$  trailing-0s, we expect there are at least  $2^z$  distinct elements
  - $R$  but not  $R + 1$ :  $[2^R, 2^{R+1}]$ 
    - analysis shows  $2^R / 0.77351$  is a good choice (the proof is hard)

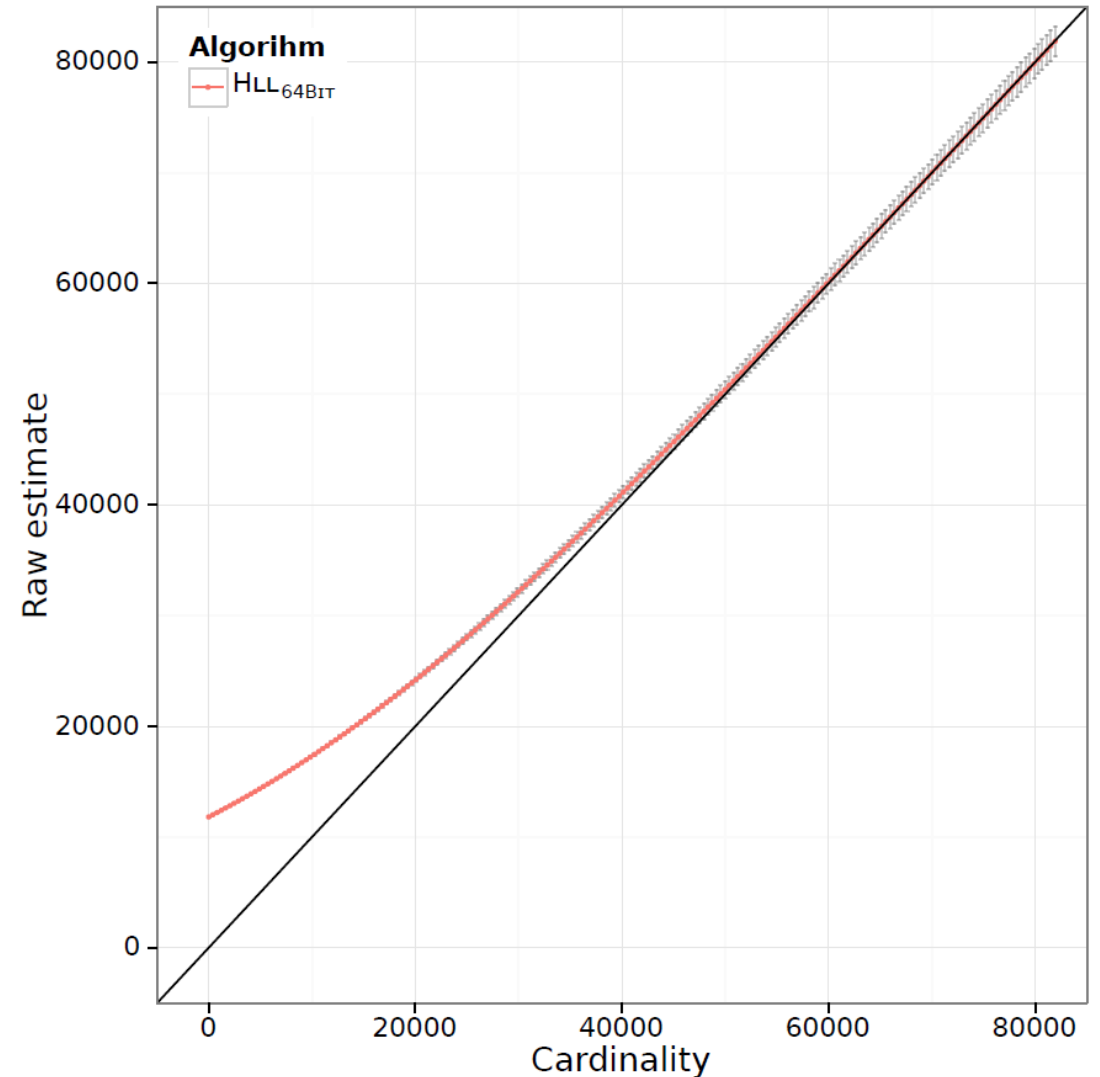
# Again, One Random Variable is Not Enough...

- To reduce the variance, we can take the average over multiply independent copies of  $2^R$ 
  - $m$  copies:  $O(m)$  update cost
- To restrict the update cost to  $O(1)$ , *stochastic averaging* can be used



# HyperLogLog: What are the Differences?

- For small cardinalities, Linear Counting is used
- For large cardinalities, the FM method is used
- number of leading zeros
- Harmonic mean instead of arithmetic mean
- Some tricky correction factors



# Outline

- Data Stream Model
- Sketch 101: AMS Sketch
- The Magic of AMS Sketch
- Counting Distinct Elements
- Finding Heavy Hitters

Point Query	$\mathbb{H}[i]$
Range Query	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

Distinct Elements	$F_0(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^0$
Sum	$F_1(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]$
Sum of Squares	$F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2$
Heavy Hitters	$\left\{ i: \mathbb{H}[i] \geq \varepsilon \sum_{k=1}^U \mathbb{H}[k] \right\}$
Top-K	

# A Deterministic Algorithm

- Find all items of frequency greater than  $1/k$  in one pass
- Initially, an empty table of maximum size  $k$

1		
2		
3		
4		
5		
...		
k		

# A Deterministic Algorithm

- Find all items of frequency greater than  $1/k$  in one pass
- When a new element  $X$  is coming, increase the counter of  $X$ 
  - Or add  $(X, 1)$  if  $X$  is not presented in the table

1		
2		
3		
4		
5		
...		
k		

← A



# A Deterministic Algorithm

- Find all items of frequency greater than  $1/k$  in one pass
- When a new element  $X$  is coming, increase the counter of  $X$ 
  - Or add  $(X, 1)$  if  $X$  is not presented in the table

1	A	1
2		
3		
4		
5		
...		
k		

# A Deterministic Algorithm

- Find all items of frequency greater than  $1/k$  in one pass
- When a new element  $X$  is coming, increase the counter of  $X$ 
  - Or add  $(X, 1)$  if  $X$  is not presented in the table

1	A	1
2		
3		
4		
5		
...		
k		

← A

# A Deterministic Algorithm

- Find all items of frequency greater than  $1/k$  in one pass
- When a new element  $X$  is coming, increase the counter of  $X$ 
  - Or add  $(X, 1)$  if  $X$  is not presented in the table

1	A	2
2		
3		
4		
5		
...		
k		

# A Deterministic Algorithm

- Find all items of frequency greater than  $1/k$  in one pass
- When a new element  $X$  is coming, increase the counter of  $X$ 
  - Or add  $(X, 1)$  if  $X$  is not presented in the table

1	A	2
2	B	1
3	C	10
4	D	3
5	E	1
...	...	...
k		

# A Deterministic Algorithm

- Find all items of frequency greater than  $1/k$  in one pass
- When a new element  $X$  is coming, increase the counter of  $X$ 
  - Or add  $(X, 1)$  if  $X$  is not presented in the table

1	A	2
2	B	1
3	C	10
4	D	3
5	E	1
...	...	...
k		

← Z

# A Deterministic Algorithm

- Find all items of frequency greater than  $1/k$  in one pass
- If the table is full, **subtract 1 from each counter and remove elements with counter value 0**

1	A	2
2	B	1
3	C	10
4	D	3
5	E	1
...	...	...
k	Z	1

# A Deterministic Algorithm

- Find all items of frequency greater than  $1/k$  in one pass
- If the table is full, **subtract 1 from each counter and remove elements with counter value 0**

1	A	2
2	B	1
3	C	10
4	D	3
5	E	1
...	...	...
k	Z	1

-1

1	A	1
2	B	0
3	C	9
4	D	2
5	E	0
...	...	...
k	Z	0

remove

1	A	1
2		
3	C	9
4	D	2
5		
...	...	...
k		

# A Deterministic Algorithm

- It was proved that any element with frequency strictly larger than  $1/k$  will be present in the table at the end of the data stream

1	A	1
2		
3	C	9
4	D	2
5		
...	...	...
k		

Proof:

- Assume an item  $X$  has  $> N/k$  copies.
- One copy is deleted at a time and this is accompanied by the deletion of  $k - 1$  elements other than  $X$ .
- At most  $N/k$  copies of  $X$  can be deleted.
- Thus, at least one copy of  $X$  survives.



# A Deterministic Algorithm

- It was proved that any element with frequency strictly larger than  $1/k$  will be present in the table at the end of the data stream

1	A	1
2		
3	C	9
4	D	2
5		
...	...	...
k		

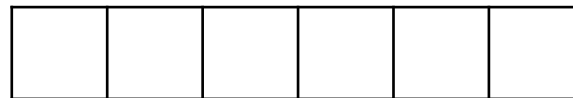
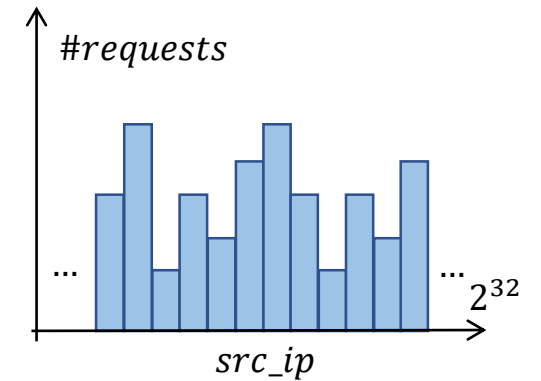
the frequency of elements is lost...

Solutions:

AMS sketch or **Count-Min sketch**

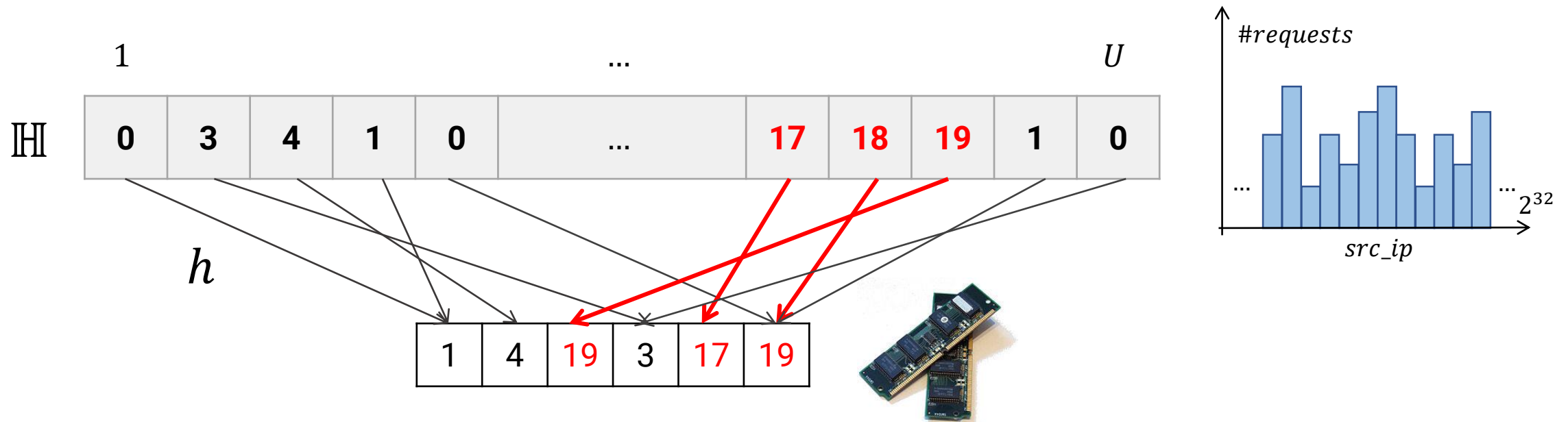
# Counting the Occurrence **Exactly**

- Exact counting requires  $\Omega(U)$  space
  - **one counter for each distinct element**
  - i.e., materializing the histogram



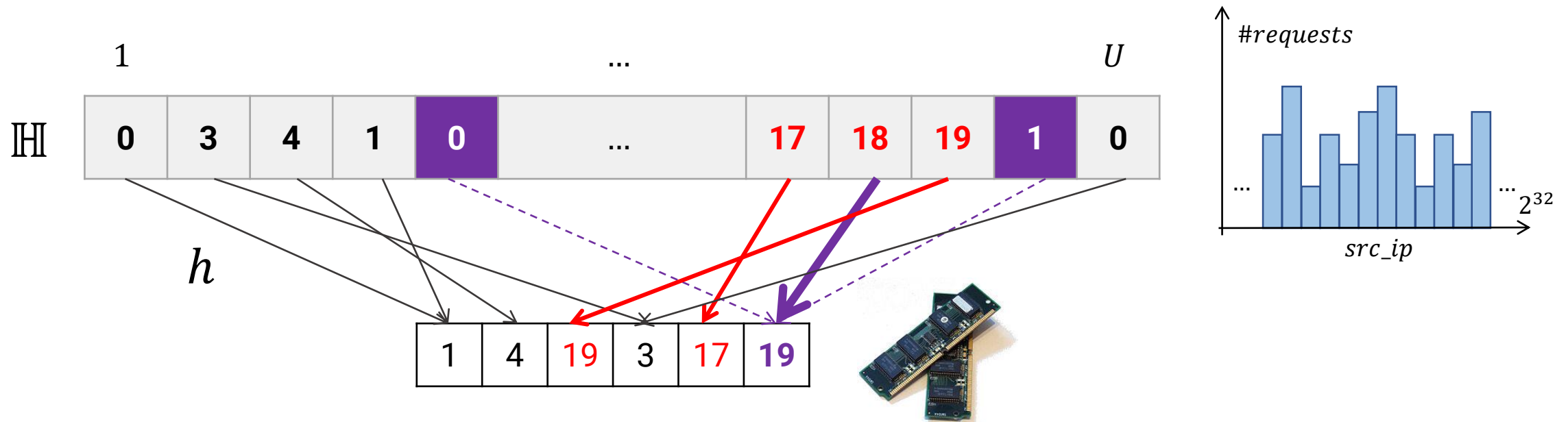
# Approximate Counting with **Hash**

- We are only interested in the frequency of **heavy hitters**
- There are only a small number of heavy hitters
  - They will be hashed into different buckets **with high probability**



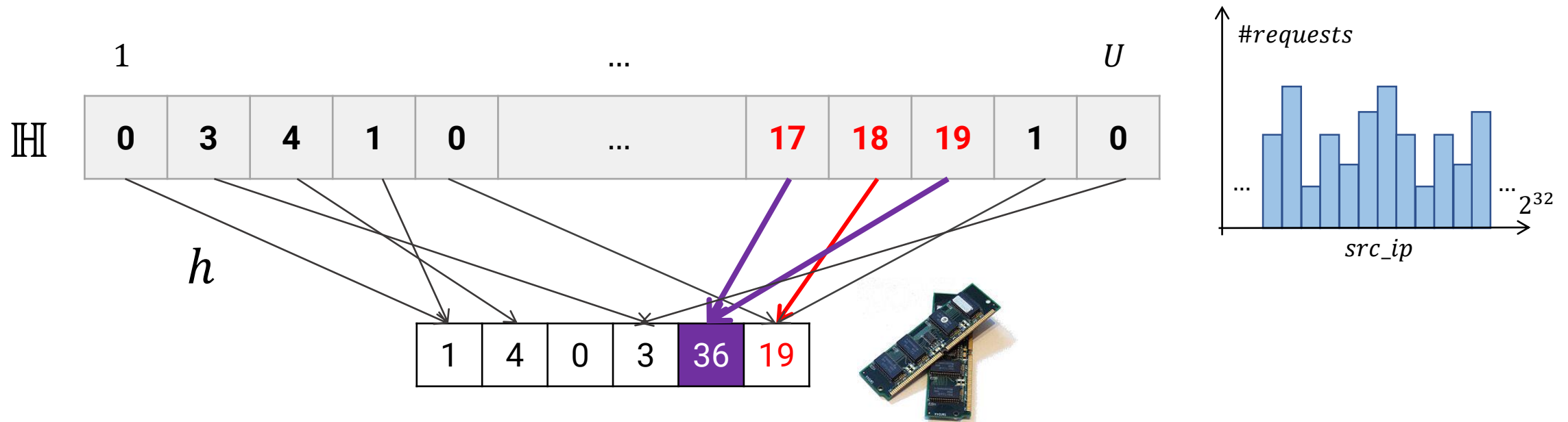
# Problem #1: False Positives

- All elements hashed into the same bucket with a heavy hitter are recognized as heavy hitters



# Problem #2: Estimation Error

- When a collision between heavy hitters really happen, the estimation error can be large

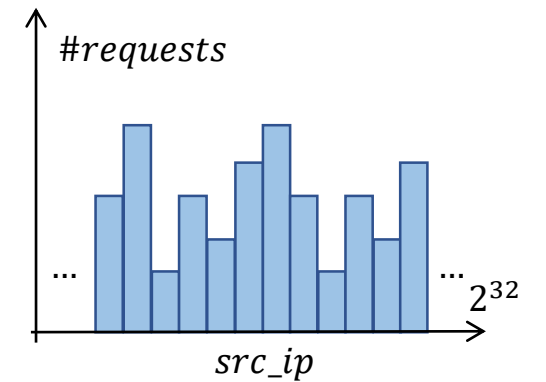


# Count-Min Sketch

- Use multiply independent hash functions

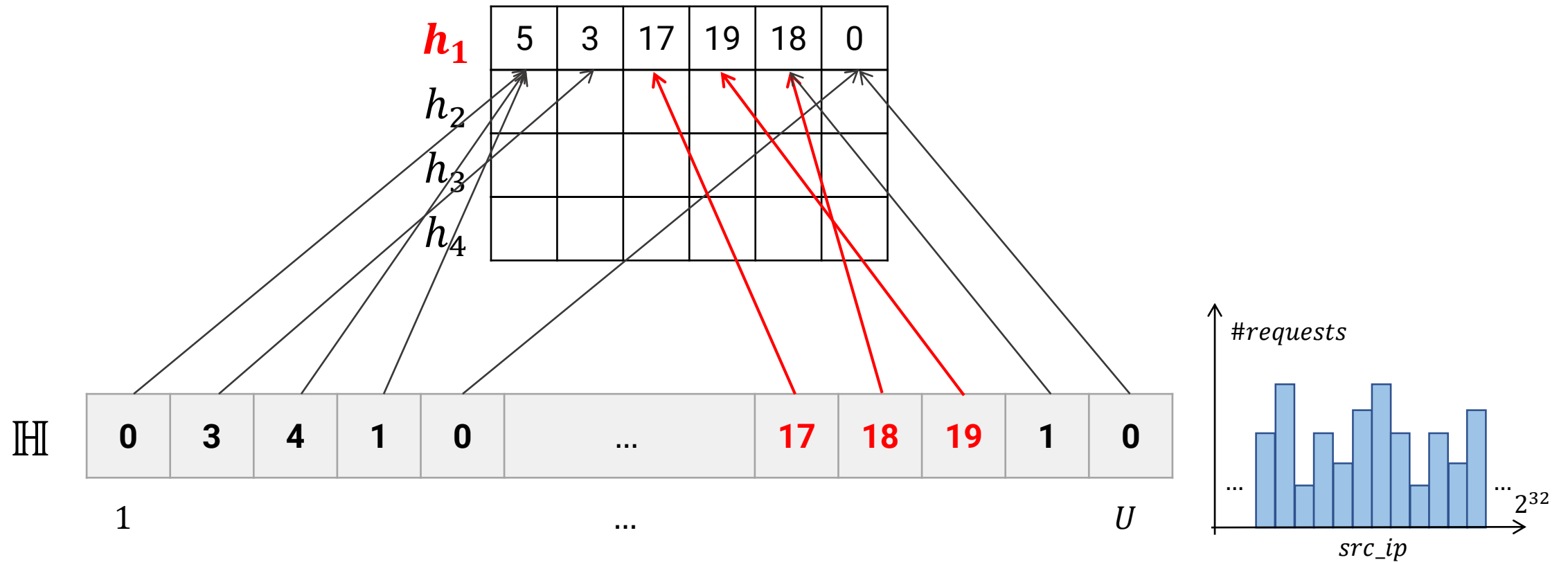
$h_1$						
$h_2$						
$h_3$						
$h_4$						

$\mathbb{H}$	0	3	4	1	0	...	17	18	19	1	0
	1					...					$U$



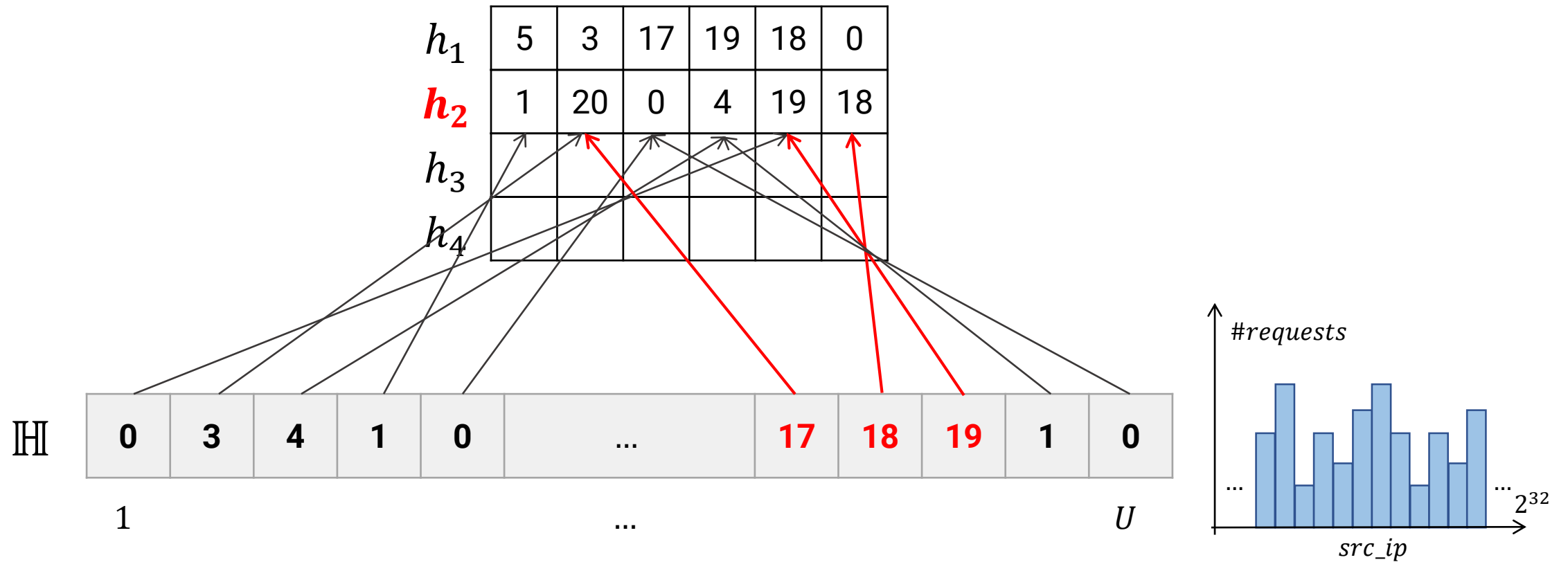
# Count-Min Sketch

- Use multiply independent hash functions



# Count-Min Sketch

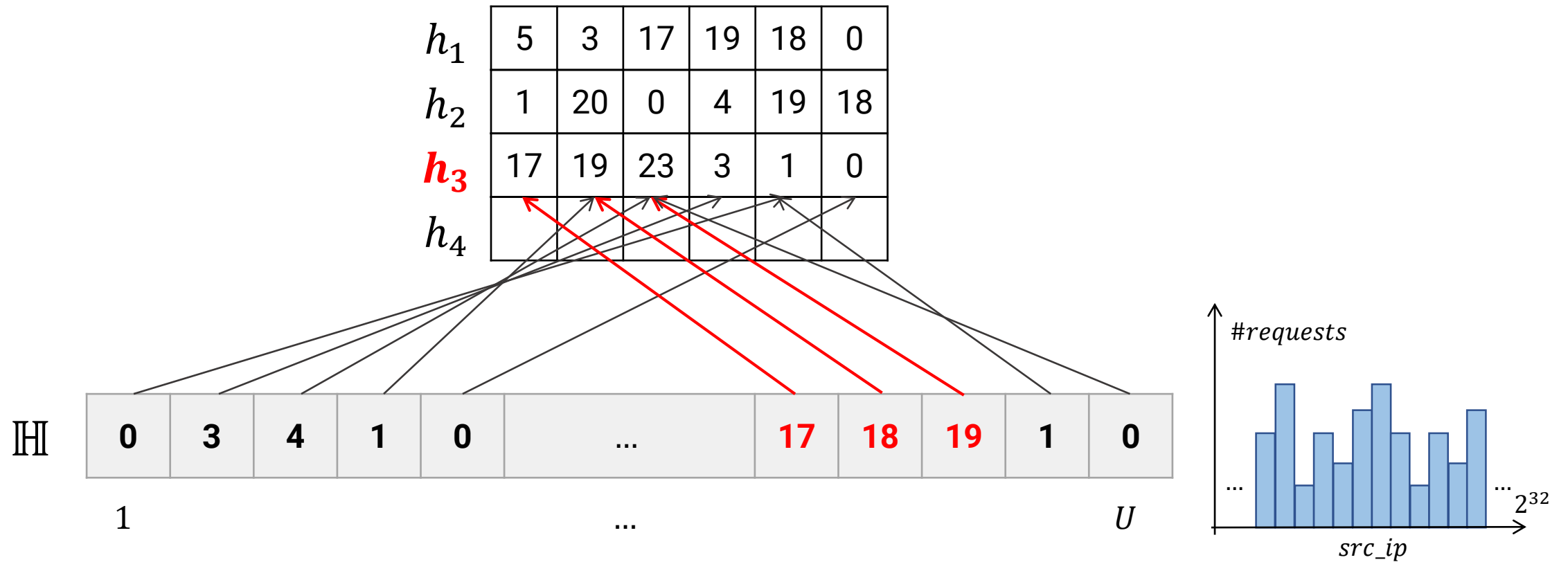
- Use multiply independent hash functions





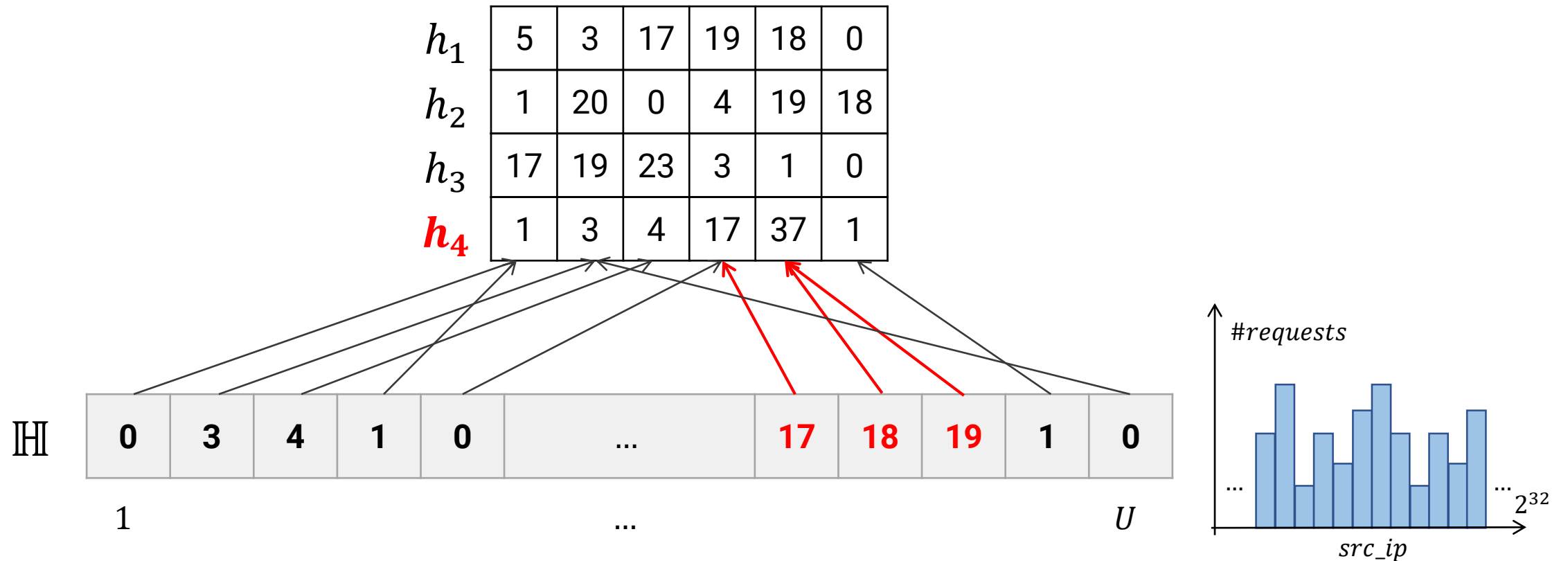
# Count-Min Sketch

- Use multiply independent hash functions



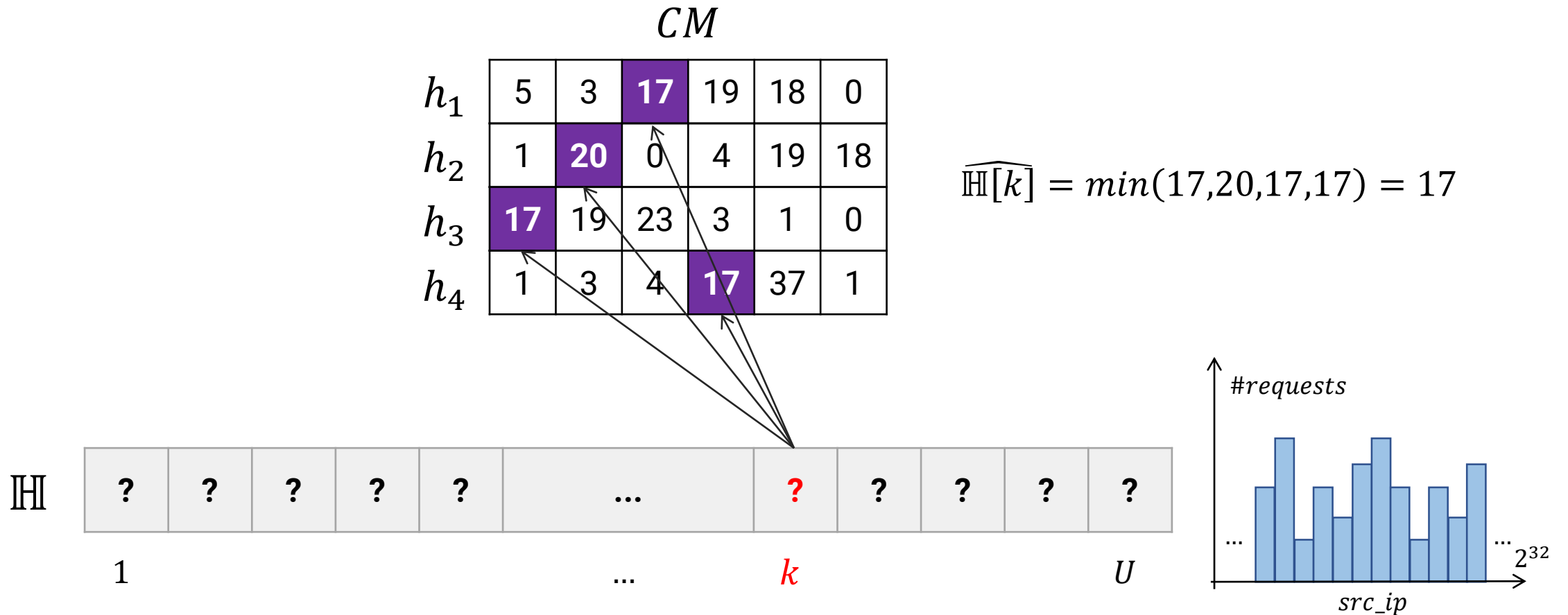
# Count-Min Sketch

- Use multiply independent hash functions



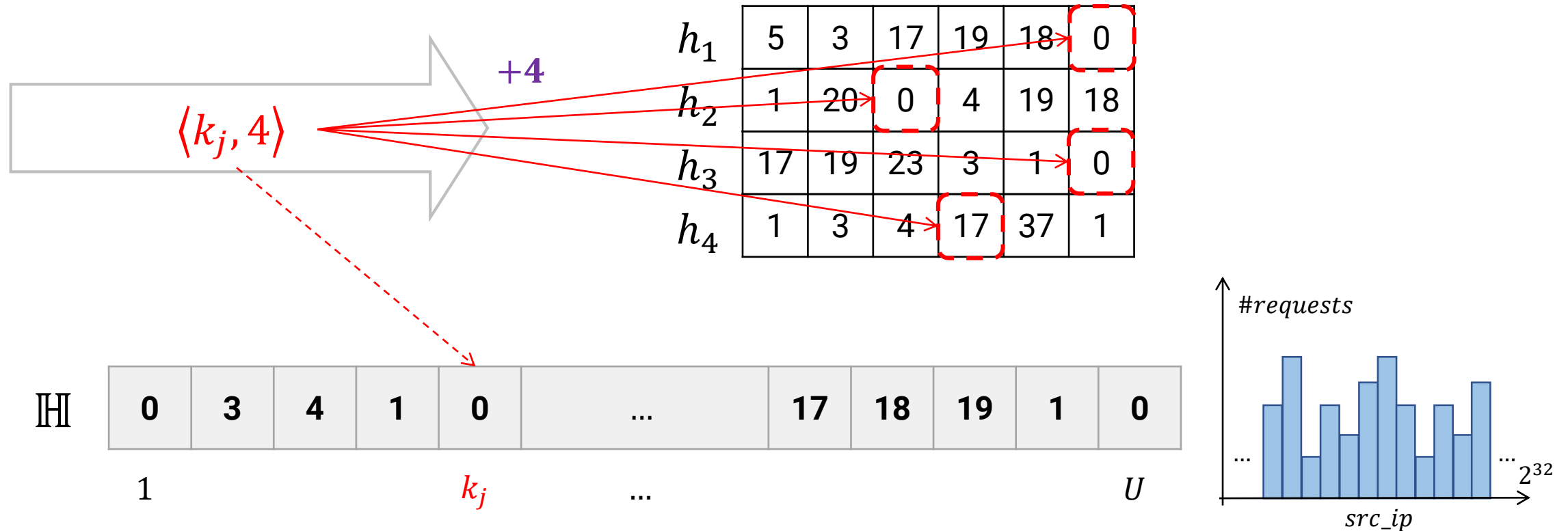
# Count-**Min** Sketch: Handling Queries

- Read the counter in every row and take the **minimum** value



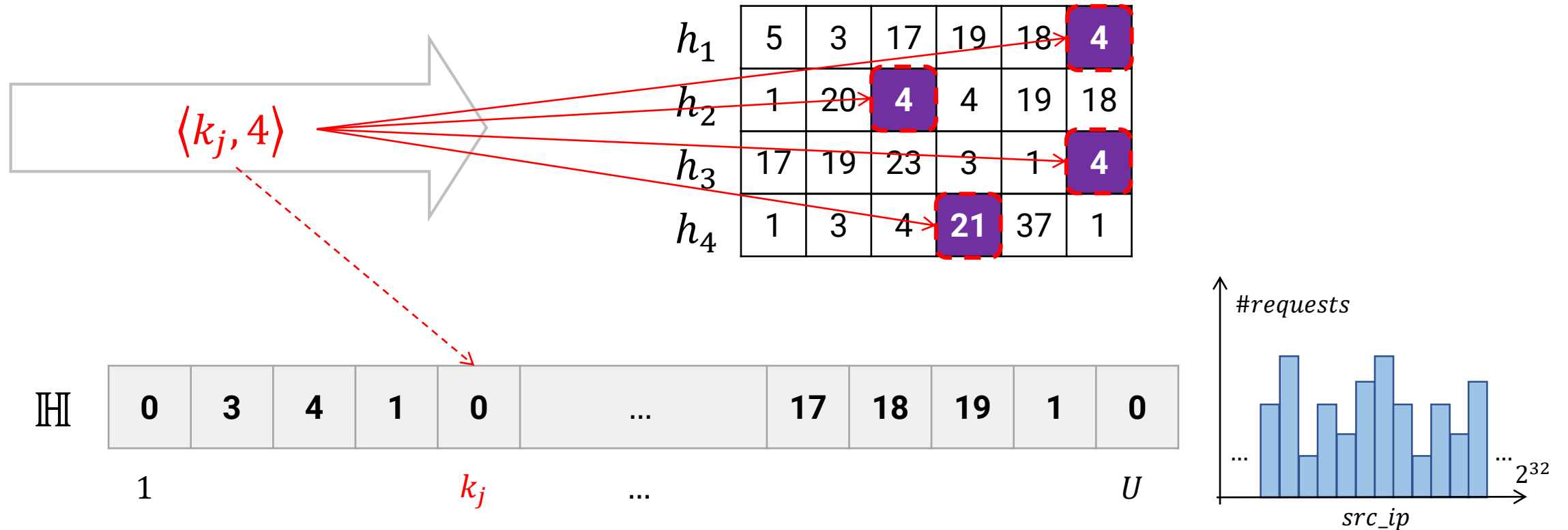
# Count-Min Sketch: Handling Updates

- Use multiply independent hash functions



# Count-Min Sketch: Handling Updates

- Use multiply independent hash functions



# Error Guarantee & Matrix Size

- **$(\epsilon, \delta)$ -approximation**:  $\widehat{\mathbb{H}}[i] \geq \mathbb{H}[i]$  and with probability at least  $1 - \delta$ :

$$\widehat{\mathbb{H}}[i] \leq \mathbb{H}[i] + \epsilon \sum_{k=1}^U \mathbb{H}[k]$$

(0.1, 0.01)-approximation:

with probability  $\geq 99\%$ , for ip 12.34.56.78,  
 (estimated #requests – true #requests)  
 $\leq 1/10$  of total #requests

	1	2	...		$w$
1					
2					
...					
$d$					

$$d = \left\lceil \log \frac{1}{\delta} \right\rceil$$

$$w = \left\lceil \frac{2}{\epsilon} \right\rceil$$

# Outline

- Data Stream Model
- Sketch 101: AMS Sketch
- The Magic of AMS Sketch
- Counting Distinct Elements
- Finding Heavy Hitters

Point Query	$\mathbb{H}[i]$
Range Query	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

Distinct Elements	$F_0(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^0$
Sum	$F_1(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]$
Sum of Squares	$F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2$
Heavy Hitters	$\left\{ i: \mathbb{H}[i] \geq \varepsilon \sum_{k=1}^U \mathbb{H}[k] \right\}$
Top-K	

# Sketches

<b>Point Query</b>	$\mathbb{H}[i]$
<b>Range Query</b>	$\sum_{i=t}^{t+w} \mathbb{H}[i]$

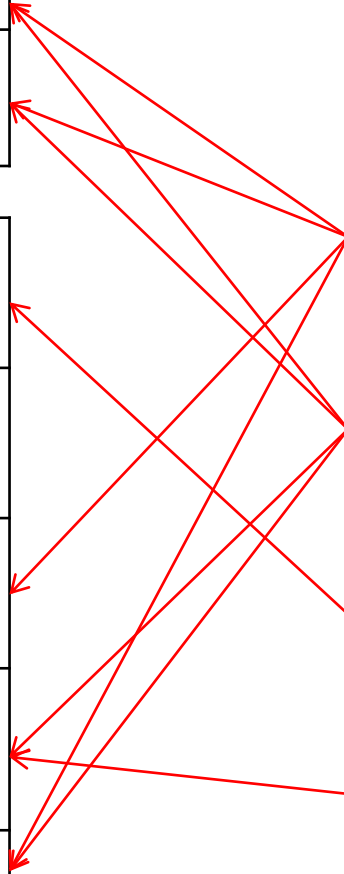
<b>Distinct Elements</b>	$F_0(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^0$
<b>Sum</b>	$F_1(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]$
<b>Sum of Squares</b>	$F_2(\mathbb{H}) = \sum_{i=1}^U \mathbb{H}[i]^2$
<b>Heavy Hitters</b>	$\left\{ i: \mathbb{H}[i] \geq \varepsilon \sum_{k=1}^U \mathbb{H}[k] \right\}$
<b>Top-K</b>	

AMS Sketch

Count-Min Sketch

FM Method

The Deterministic  
Heavy Hitter Algorithm





# Summary

- Data stream: one-pass, limited memory, high speed
- A histogram model for data stream
- Sketches: estimators, error reduction, error guarantees

# Applications to DBMS Internals

- query optimization: index or full table scan? selectivity? join size?
  - Number of distinct values, (self-)join size, point&range queries
- partitioning key selection in shared-nothing parallel DB
  - Number of distinct values, Skewness
- choosing encoding schemes in column stores
  - Use less bits to encode heavy hitters
- ...

# Topics not Covered

- Quantile queries
  - Median, 0.9999...
- Arbitrary point/range queries
  - Histogram, wavelets
- Sampling

Q&A

Backup Slides

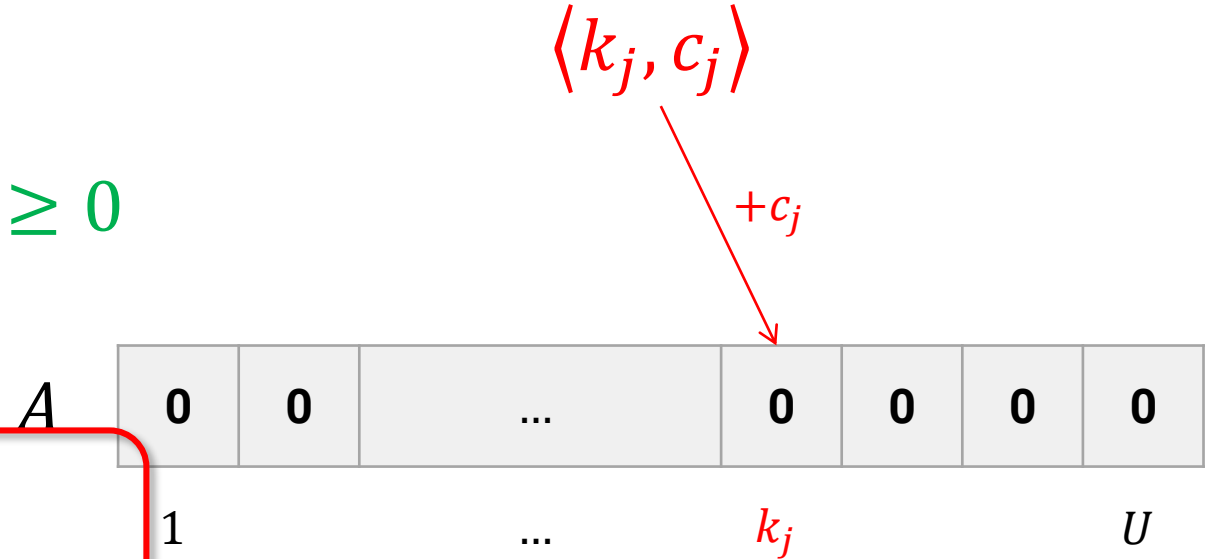
# Approximate Solutions

- **Approximate** functions provided by Oracle/BigQuery/Presto/...
  - `APPROX_COUNT_DISTINCT`
  - `APPROX_QUANTILES/APPROX_PERCENTILE`
  - `APPROX_TOP_COUNT`
  - `NUMERIC_HISTOGRAM`
- How are these functions implemented internally?

# Data Streaming Models

- **Time Series Model:**  $\langle k_j, c_j \rangle, k_j = j$ 
  - Updates arrive in increasing order of  $j$
  - i.e., append-only
- **Cash Register Model:**  $\langle k_j, c_j \rangle, c_j \geq 0$ 
  - i.e., no deletion

- **Turnstile Model:**  $\langle k_j, c_j \rangle$ 
  - $c_j$  can be either positive or negative
  - *The most general model !!!*



# Analysis of Linear Counting

- standard error metric (**load factor**  $t = F_0/m$ )

$$StdErr\left(\frac{\widehat{F_0}}{F_0}\right) = \frac{\sqrt{m(e^t - t - 1)}}{F_0}$$

- To control  $StdErr < \epsilon$  and avoid the bitmap to be filled up:

$$m > \max(a^2, 1/(\epsilon t)^2)(e^t - t - 1)$$

when  $a = \sqrt{5}$ , the fill-up probability is 0.7%



# Why FM Algorithm Works? (cont.)

- Prob. of **NOT** seeing a tail of length  $r$  among  $F_0$  elements:

$$P(R < r) = (1 - 1/2^r)^{F_0} \approx e^{-F_0/2^r}$$

- If  $2^r \gg F_0$ , then  $P(R < r)$  tends to 1
  - In reverse,  $P(R \geq r)$  tends to 0
- If  $2^r \ll F_0$ , then  $P(R < r)$  tends to 0
  - In reverse,  $P(R \geq r)$  tends to 1
- Thus,  $2^R$  will almost always be around  $F_0$  !!!

# Hash Functions

- $h: [1 \dots U] \rightarrow [0 \dots U - 1]$

- 2-wise independent:

$$\mathcal{H}_2 = \{ax + b \bmod P \bmod U\}$$

- $a, b$ : pick randomly from  $[0 \dots P - 1]$

- k-wise independent:

$$\mathcal{H}_k = \left\{ \sum_{i=0}^{k-1} c_i x^i \bmod P \bmod U \right\}$$