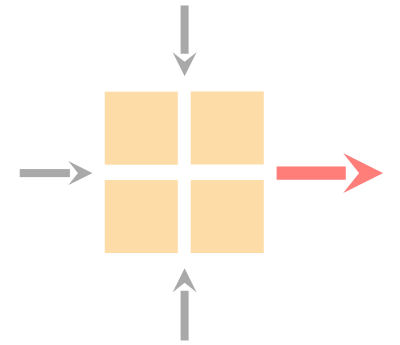


# Advanced Topics in Communication Networks

## Programming Network Data Planes



Alexander Dietmüller

[nsg.ee.ethz.ch](http://nsg.ee.ethz.ch)

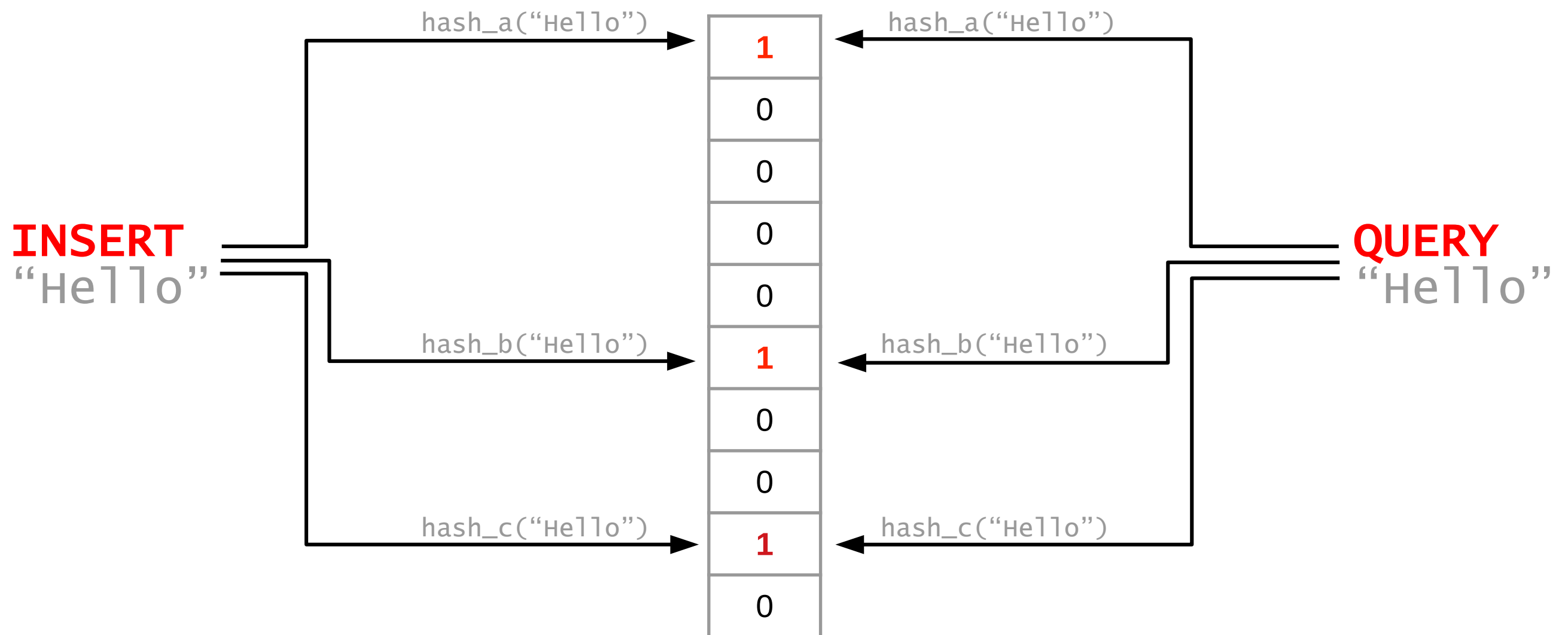
ETH Zürich

Oct. 11 2018

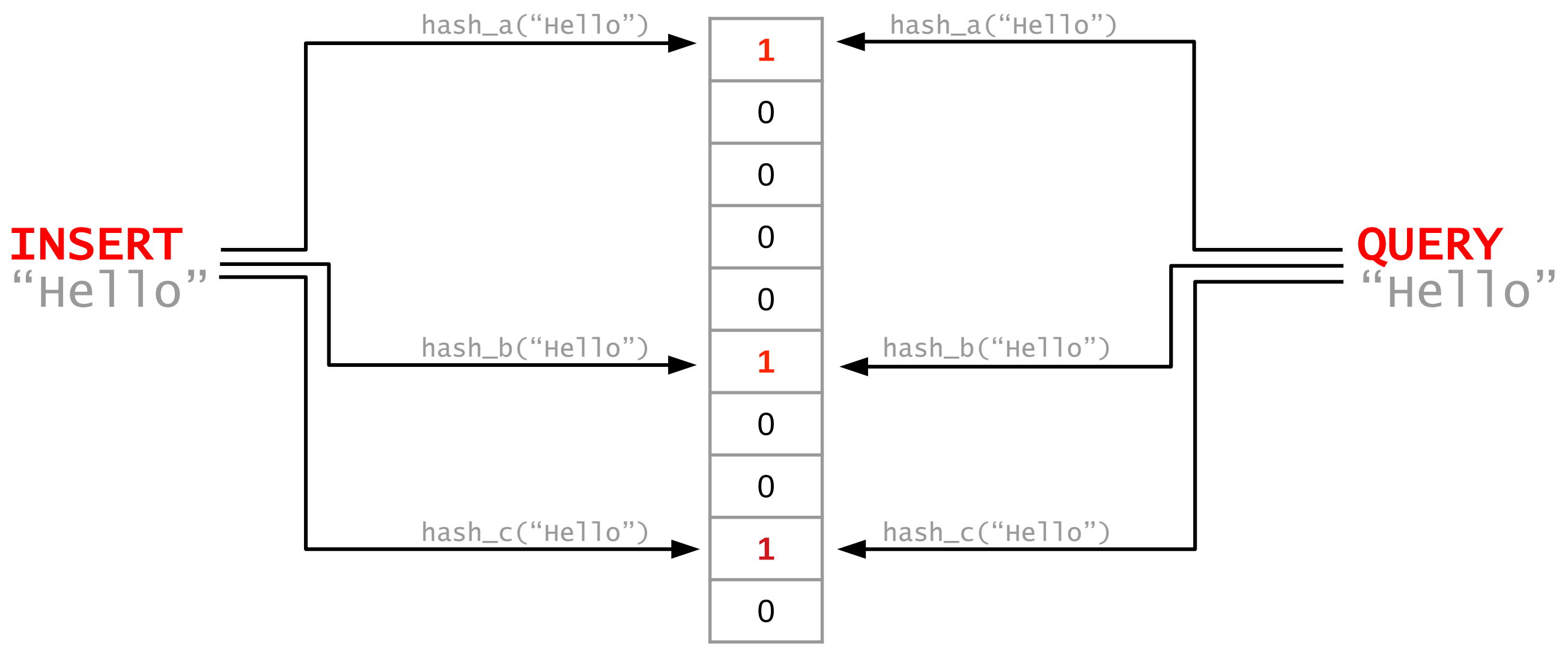
Last week on

# Advanced Topics in Communication Networks

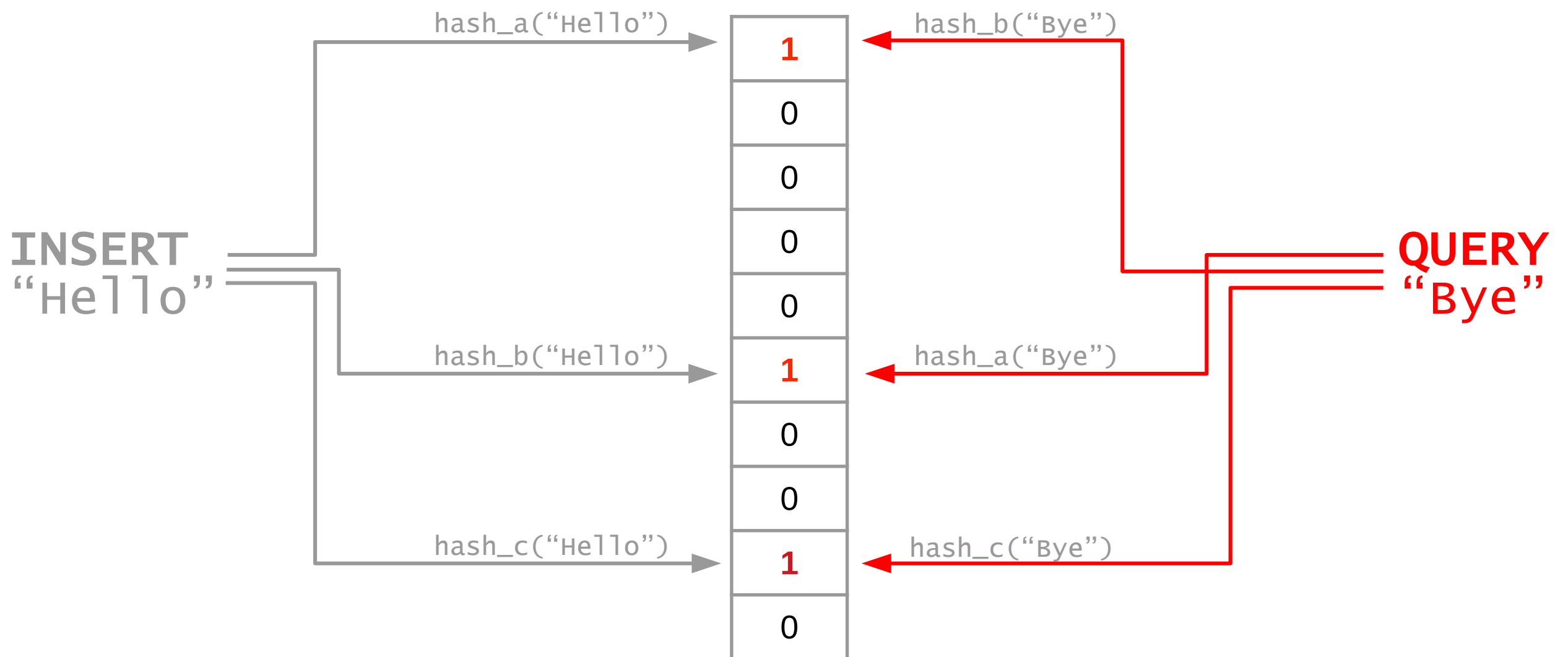
Probabilistic data structures like Bloom Filters help to trade resources with accuracy



Bloom Filters take a fixed number of operations, but hash collisions can cause false positives.



Bloom Filters take a fixed number of operations,  
but hash collisions can cause **false positives**



*A bloom filter is a streaming algorithm*

***answering specific questions approximately.***

*A bloom filter is a streaming algorithm*

*answering **specific questions approximately.***

Is X in the stream?

What is in the stream?

Invertible Bloom Filter

*A bloom filter is a streaming algorithm  
**answering specific questions approximately.***

|  
Is X in the stream?

What is in the stream? Invertible Bloom Filter

**What about other questions?**



**This week** on

**Advanced Topics in Communication Networks**

Today we'll talk about: **important questions,**  
how 'sketches' answer them,  
and limitations of 'sketches'



*Is a certain element in the stream?*

*Bloom Filter*

*How many distinct elements are in the stream?*

*HyperLogLog Sketch, ...*

*How frequently does an element appear?*

*Count Sketch, CountMin Sketch, ...*

*What are the most frequent elements?*

*Count/CountMin + Heap, ...*

*How many elements belong to a certain subnet?*

*SketchLearn* SigComm '18

*In networking, we talk about **packet flows**,  
but these questions apply to other domains as well,  
e.g. **search engines and databases**.*



*Is a certain element in the stream?*

*Bloom Filter*

*How many distinct elements are in the stream?*

*HyperLogLog Sketch, ...*

***How frequently does an element appear?***

*Count Sketch, CountMin Sketch, ...*

*What are the most frequent elements?*

*Count/CountMin + Heap, ...*

*How many elements belong to a certain subnet?*

*SketchLearn* SigComm '18

We are going to look at **frequencies**,  
i.e. **how often** an element occurs in a data stream.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix} \quad \begin{array}{l} \text{vector of frequencies (counts)} \\ \text{of all } \mathbf{distinct\ elements } x_i \end{array}$$

We are going to look at **frequencies**,  
i.e. **how often** an element occurs in a data stream.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$$

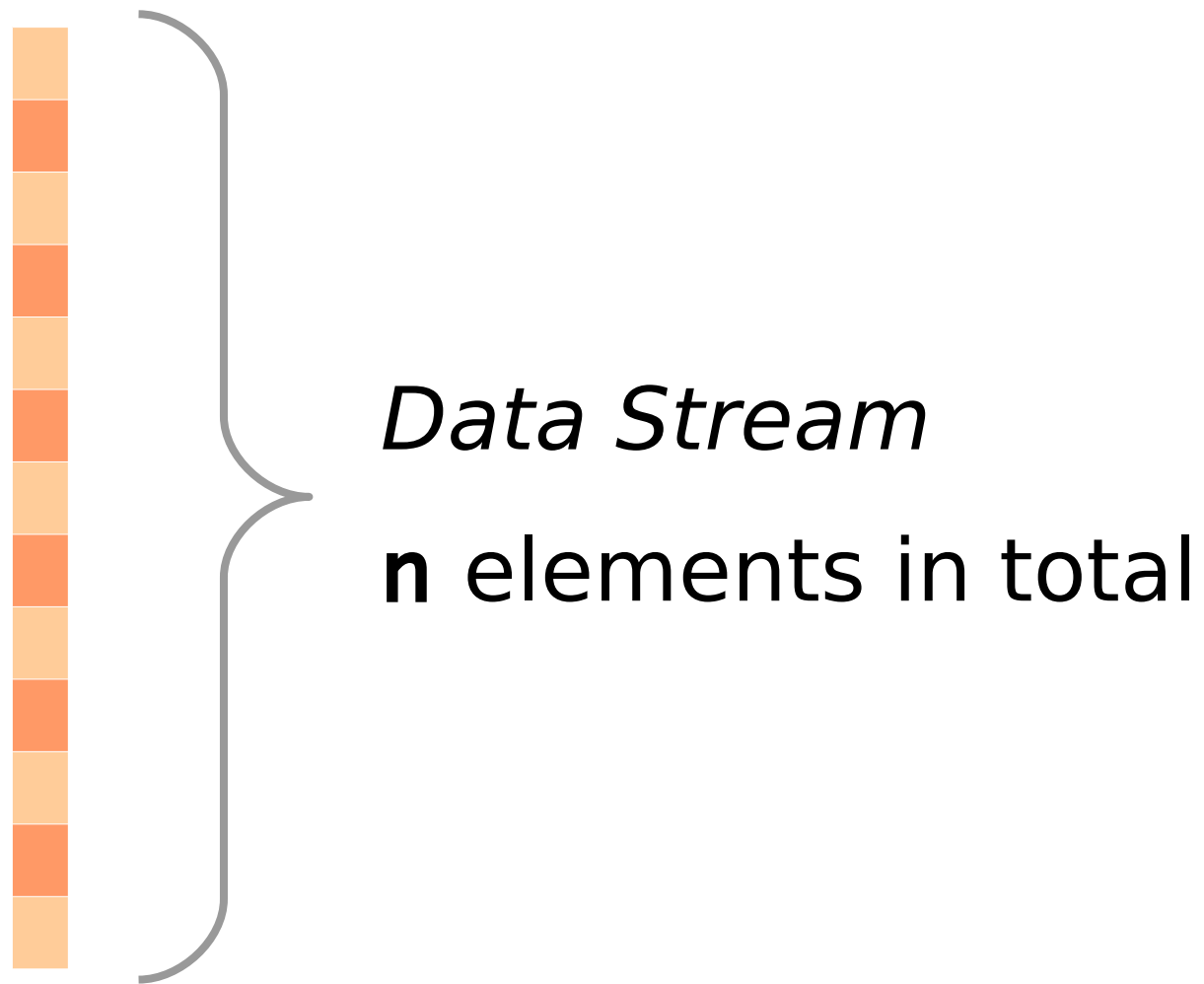
*vector of frequencies (counts)*  
of all **distinct elements**  $x_i$

**distinct flows**

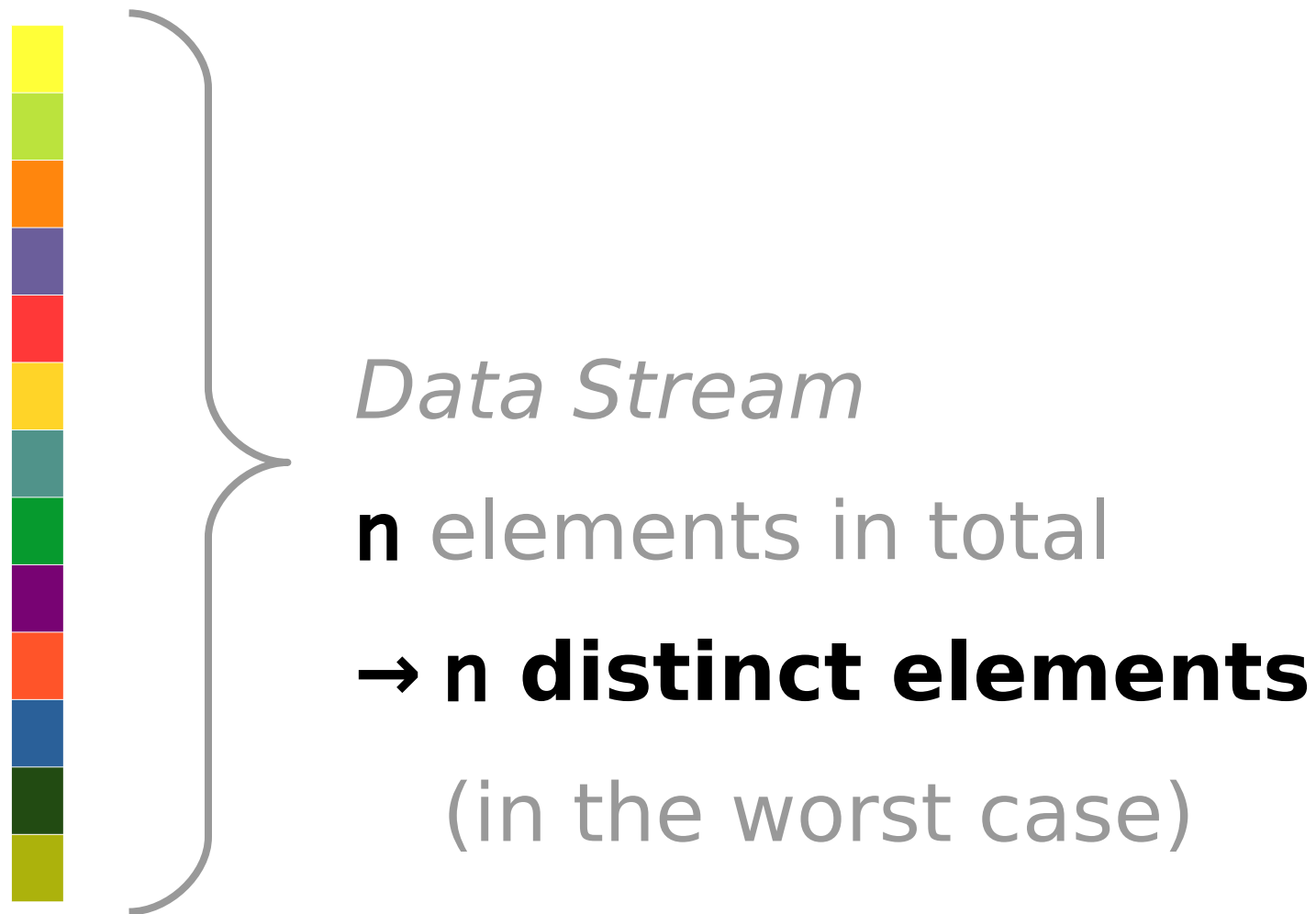
In the worst case, an algorithm providing **exact frequencies** requires **linear space**.



In the worst case, an algorithm providing **exact frequencies** requires **linear space**.



In the worst case, an algorithm providing **exact frequencies** requires **linear space**.



In the worst case, an algorithm providing **exact frequencies** requires **linear space**.



*Data Stream*

**n** elements in total

→ **n distinct elements**

(in the worst case)

→ **n counters** required? :(

# Probabilistic datastructures can help again!

## **Bloom Filters**

*quickly “filter” only those elements that might be in the set*

*More efficient by allowing false positives.*

# Probabilistic datastructures can help again!

## **Bloom Filters**

*quickly “filter” only those elements that might be in the set*

*More efficient by allowing false positives.*

## **Sketches**

*provide a approximate frequencies of elements in a data stream.*

*More efficient by allowing mis-counting.*

Today we'll talk about: important questions,

**how 'sketches' answer them,**

limitations of 'sketches'

*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

***Notation reminder:***

*vector of frequencies (counts)*

*of all **distinct elements**  $x_i$*

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \end{bmatrix}$$



*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

$$Pr \left[ \underset{\substack{\text{estimated} \\ \text{frequency}}}{\hat{x}_i} - \underset{\substack{\text{true} \\ \text{frequency}}}{x_i} \geq \underset{\substack{\text{sum of} \\ \text{frequencies}}}{\varepsilon \|\mathbf{x}\|_1} \right] \leq \delta$$

The estimation error **exceeds**  $\varepsilon \|\mathbf{x}\|_1$   
with a **probability smaller than**  $\delta$

$$Pr \left[ \underset{\substack{\text{estimated} \\ \text{frequency}}}{\hat{x}_i} - \underset{\substack{\text{true} \\ \text{frequency}}}{x_i} \geq \underset{\substack{\text{sum of} \\ \text{frequencies}}}{\varepsilon \|\mathbf{x}\|_1} \right] \leq \delta$$

*relative to L1 norm*

The estimation error **exceeds**  $\varepsilon \|\mathbf{x}\|_1$   
 with a **probability smaller than**  $\delta$

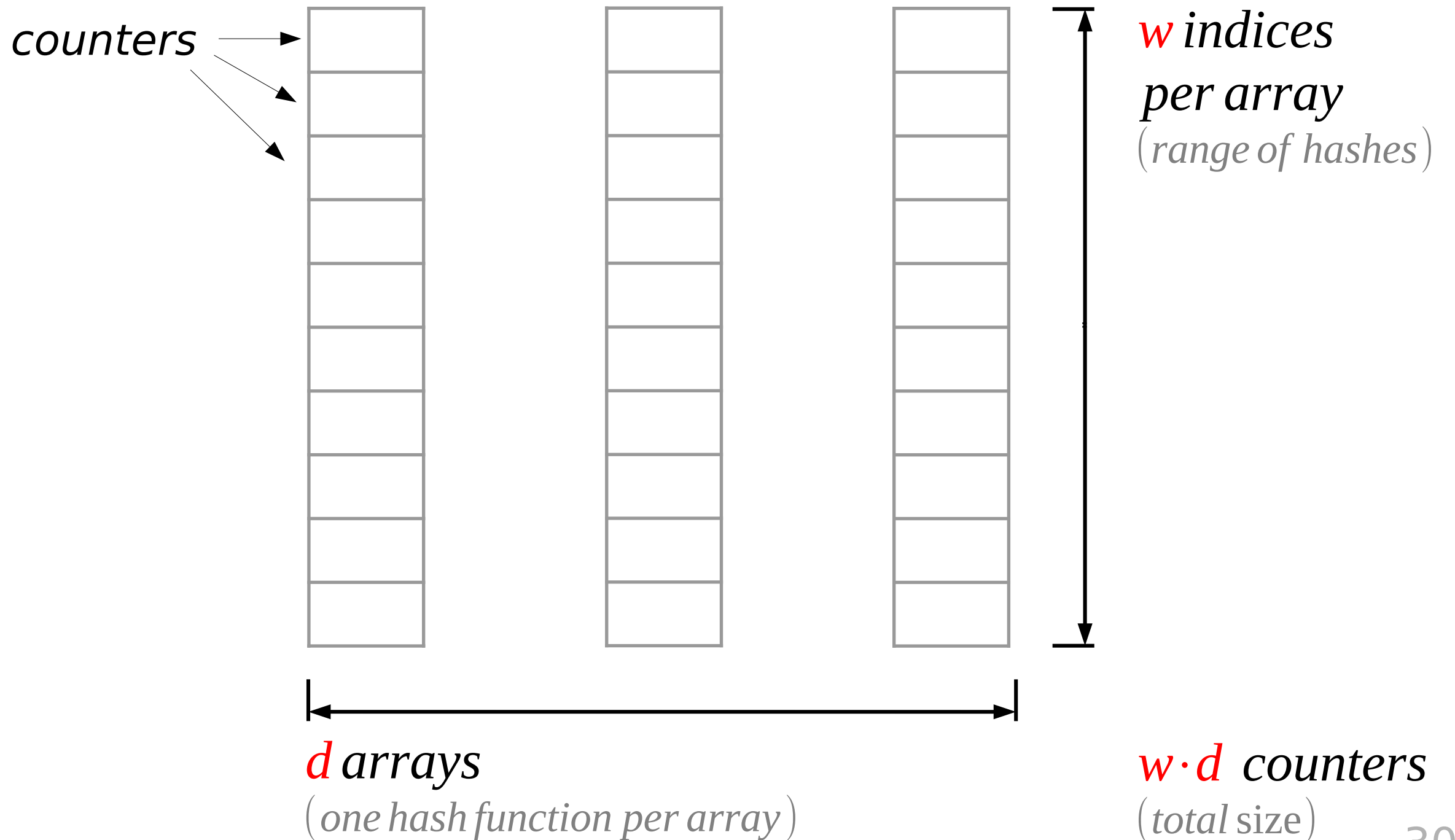
$$Pr \left[ \underset{\substack{\text{estimated} \\ \text{frequency}}}{\hat{x}_i} - \underset{\substack{\text{true} \\ \text{frequency}}}{x_i} \geq \underset{\substack{\text{sum of} \\ \text{frequencies}}}{\varepsilon \|\mathbf{x}\|_1} \right] \leq \delta$$

Let  $\|\mathbf{x}\|_1 = 10000$ ,  $\varepsilon = 0.01$ ,  $\delta = 0.05$

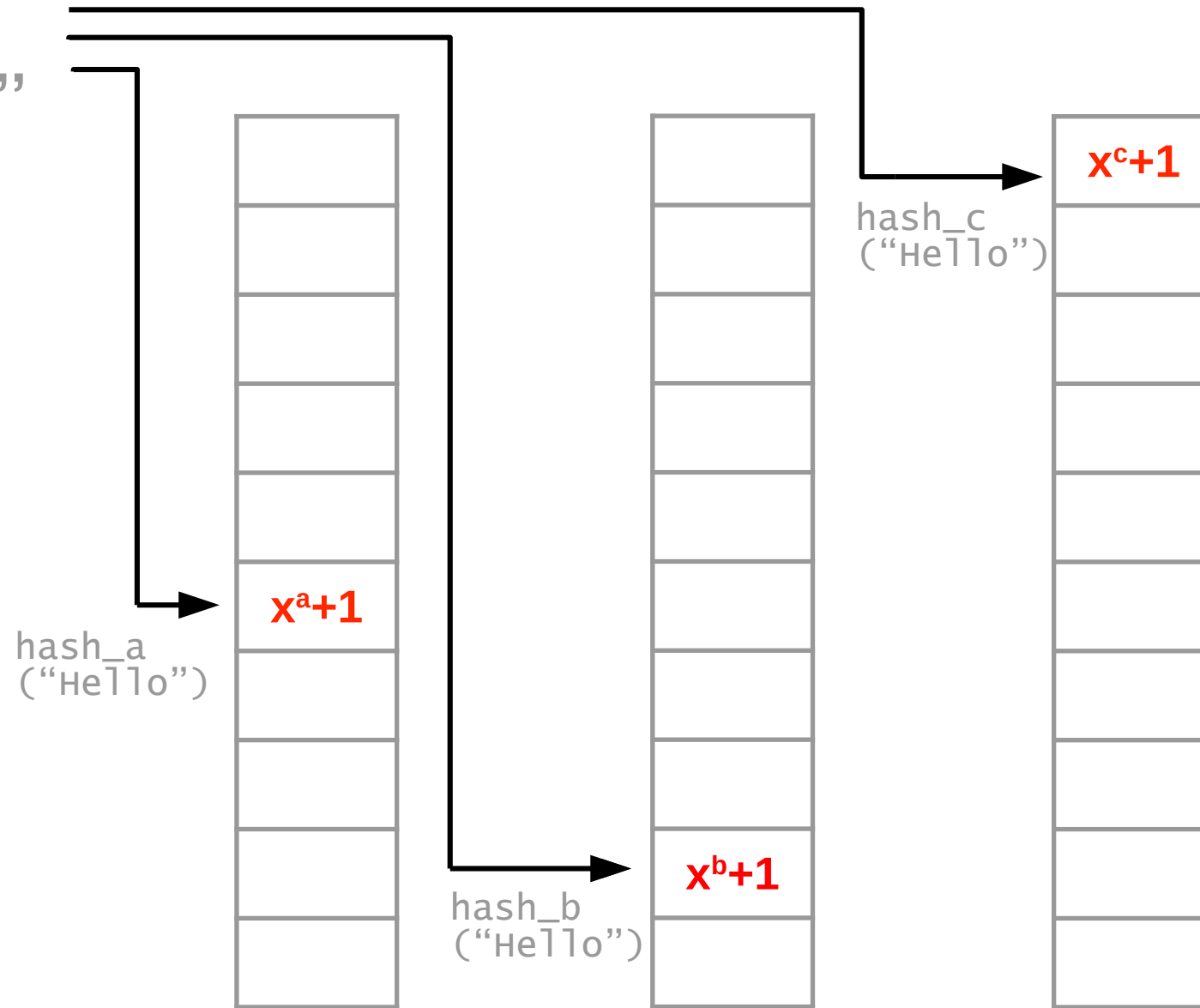
The probability for **any estimate** to be off by **more than 100** is **less than 5%**  
(after counting 10000 elements)

*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

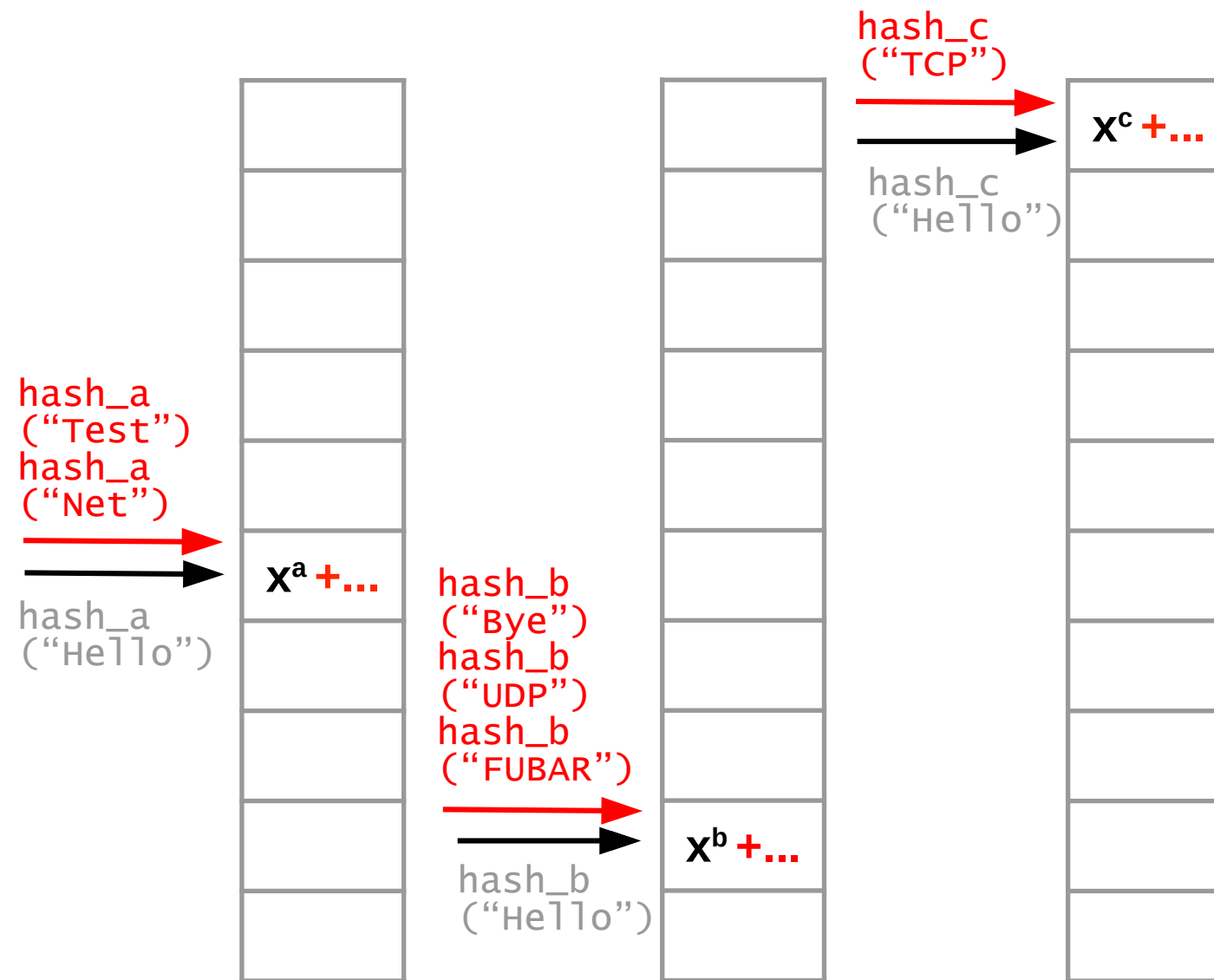
A **CountMin** Sketch uses multiple arrays and hashes.



**COUNT**  
"Hello"

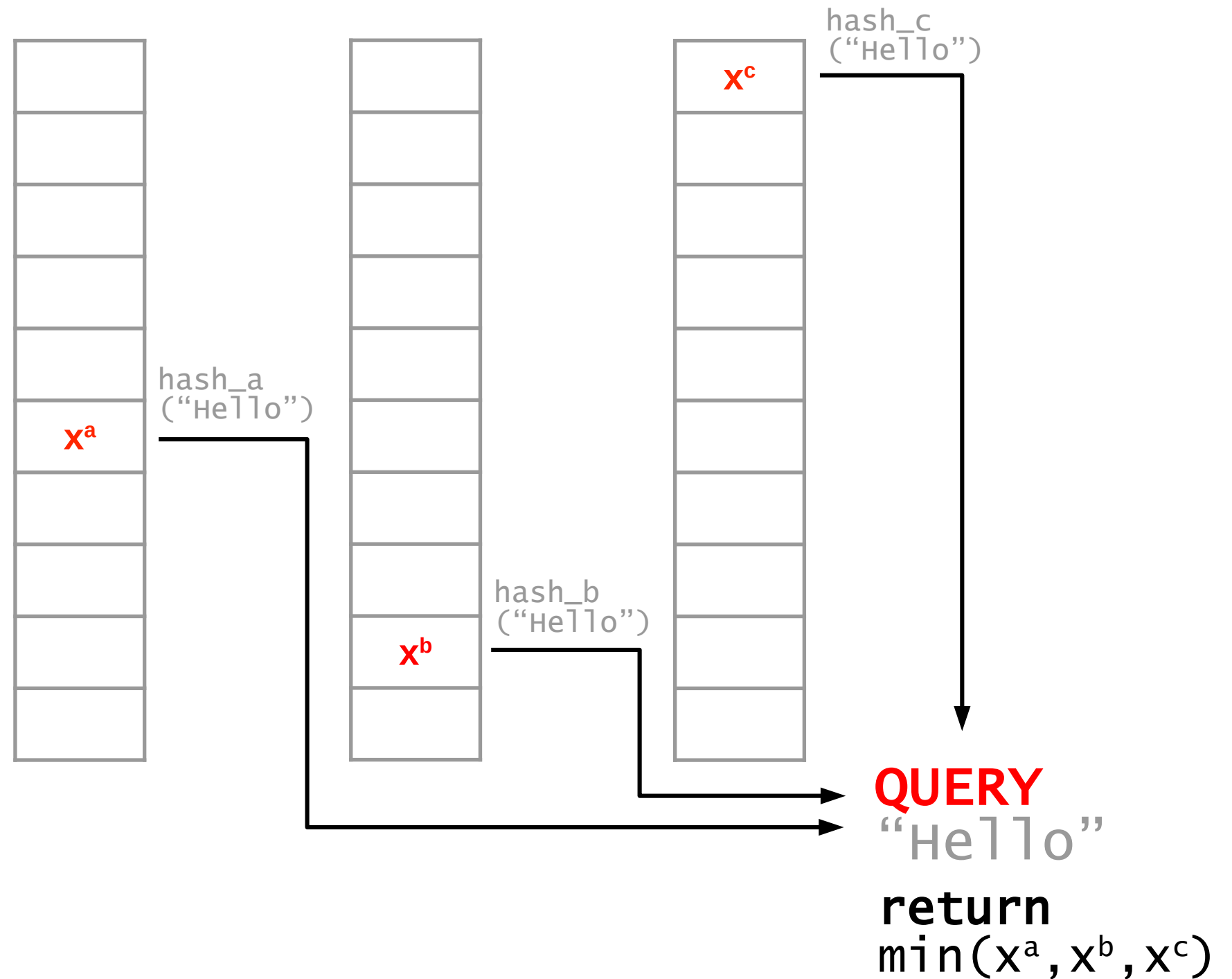


# Hash collisions cause over-counting.





Returning the **minimum value** minimizes the error.



A **CountMin sketch** uses the same principles as a counting bloom filter, but is designed to have **provable L1 error bounds** for frequency queries.

$$\Pr \left[ \underbrace{\hat{x}_i}_{\text{estimated frequency}} - \underbrace{x_i}_{\text{true frequency}} \geq \varepsilon \underbrace{\|\mathbf{x}\|_1}_{\text{sum of frequencies}} \right] \leq \delta$$

Understanding the error bounds allows **dimensioning** the sketch optimally.

**Error Bounds**  
per hash/array

**Error Bounds**  
for the minimum

**Optimal Size**

**Error Bounds**  
per hash/array

$$\hat{x}_i = \min_{h \in h_1 \dots h_d} \hat{x}_i^h$$

*estimated frequency*                      *estimate for specific hash*

**Error Bounds**  
for the minimum

**Optimal Size**

# The error bounds can be derived with **Markov's Inequality**

**Error Bounds**  
per hash/array

$$\Pr [ X \geq c \cdot E [ X ] ] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

**Optimal Size**

# The error bounds can be derived with **Markov's Inequality**

**Error Bounds**  
per hash/array

$$\Pr [\hat{x}_i^h - x_i \geq c \cdot E [\hat{x}_i^h - x_i]] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

**Optimal Size**

**Error Bounds**  
per hash/array

$$\Pr [\hat{x}_i^h - x_i \geq c \cdot E [\hat{x}_i^h - x_i]] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

$$\hat{x}_i^h = x_i + \sum_{x_j \neq x_i} x_j \mathbf{1}_h(x_i, x_j)$$

*true  
frequency*

*over-counting  
from hash collisions*

**Optimal Size**

**Error Bounds**  
per hash/array

$$\Pr [\hat{x}_i^h - x_i \geq c \cdot E [\hat{x}_i^h - x_i]] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

$$\hat{x}_i^h = x_i + \sum_{x_j \neq x_i} x_j \mathbf{1}_h(x_i, x_j)$$

*hash collision*

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

**Optimal Size**



**Error Bounds**  
per hash/array

$$\Pr [\hat{x}_i^h - x_i \geq c \cdot E [\hat{x}_i^h - x_i]] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \mathbf{1}_h(x_i, x_j)$$

*estimation  
error*

*over-counting  
from hash collisions*

**Optimal Size**

**Error Bounds**  
per hash/array

**Error Bounds**  
for the minimum

**Optimal Size**

$$\Pr \left[ \hat{x}_i^h - x_i \geq c \cdot E \left[ \hat{x}_i^h - x_i \right] \right] \leq \frac{1}{c}$$

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \mathbf{1}_h(x_i, x_j)$$

$$E \left[ \hat{x}_i^h - x_i \right] = E \left[ \sum_{x_j \neq x_i} x_j \mathbf{1}_h(x_i, x_j) \right]$$

We treat the **data as a constant** and the **hash as a random function** with certain properties.

**Error Bounds**  
per hash/array

$$\Pr \left[ \hat{x}_i^h - x_i \geq c \cdot E \left[ \hat{x}_i^h - x_i \right] \right] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \mathbf{1}_h(x_i, x_j)$$

$$E \left[ \hat{x}_i^h - x_i \right] = E \left[ \sum_{x_j \neq x_i} \underbrace{x_j}_{\text{random}} \underbrace{\mathbf{1}_h(x_i, x_j)}_{\text{constant}} \right]$$

**Optimal Size**

We treat the **data as a constant** and the **hash as a random function** with certain properties.

**Error Bounds**  
per hash/array

$$\Pr \left[ \hat{x}_i^h - x_i \geq c \cdot E \left[ \hat{x}_i^h - x_i \right] \right] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \mathbf{1}_h(x_i, x_j)$$

$$E \left[ \hat{x}_i^h - x_i \right] = \sum_{x_j \neq x_i} x_j E \left[ \mathbf{1}_h(x_i, x_j) \right]$$

**Optimal Size**

We treat the **data as a constant** and the **hash as a random function** with certain properties.

**Error Bounds**  
per hash/array

$$\Pr \left[ \hat{x}_i^h - x_i \geq c \cdot E \left[ \hat{x}_i^h - x_i \right] \right] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \mathbf{1}_h(x_i, x_j)$$

$$E \left[ \hat{x}_i^h - x_i \right] = \sum_{x_j \neq x_i} x_j \underbrace{E \left[ \mathbf{1}_h(x_i, x_j) \right]}_{\leq \frac{1}{w}}$$

**Optimal Size**

We treat the **data as a constant** and the **hash as a random function** with certain properties.

**Error Bounds**  
per hash/array

$$\Pr \left[ \hat{x}_i^h - x_i \geq c \cdot E \left[ \hat{x}_i^h - x_i \right] \right] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \mathbf{1}_h(x_i, x_j)$$

$$E \left[ \hat{x}_i^h - x_i \right] \leq \sum_{x_j \neq x_i} x_j \frac{1}{w}$$

**Optimal Size**

**Error Bounds**  
per hash/array

**Error Bounds**  
for the minimum

**Optimal Size**

$$\Pr \left[ \hat{x}_i^h - x_i \geq c \cdot E \left[ \hat{x}_i^h - x_i \right] \right] \leq \frac{1}{c}$$

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \mathbf{1}_h(x_i, x_j)$$

$$E \left[ \hat{x}_i^h - x_i \right] \leq \sum_{x_j \neq x_i} x_j \frac{1}{w} \leq \sum_{x_j} x_j \frac{1}{w}$$

**Error Bounds**  
per hash/array

$$\Pr \left[ \hat{x}_i^h - x_i \geq c \cdot E \left[ \hat{x}_i^h - x_i \right] \right] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

$$\hat{x}_i^h - x_i = \sum_{x_j \neq x_i} x_j \mathbf{1}_h(x_i, x_j)$$

$$E \left[ \hat{x}_i^h - x_i \right] \leq \sum_{x_j \neq x_i} x_j \frac{1}{w} \leq \|\mathbf{x}\|_1 \frac{1}{w}$$

**Optimal Size**



**Error Bounds**  
per hash/array

$$\Pr \left[ \hat{x}_i^h - x_i \geq c \cdot \underbrace{E \left[ \hat{x}_i^h - x_i \right]}_{\leq \frac{1}{w} \|\mathbf{x}\|_1} \right] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

**Optimal Size**

**Error Bounds**  
per hash/array

$$\Pr \left[ \hat{x}_i^h - x_i \geq \frac{c}{w} \|\mathbf{x}\|_1 \right] \leq \frac{1}{c}$$

**Error Bounds**  
for the minimum

**Optimal Size**

**Error Bounds**  
per hash/array

$$\Pr \left[ \hat{x}_i^h - x_i \geq \underbrace{\varepsilon^h}_{\frac{c}{w}} \|\mathbf{x}\|_1 \right] \leq \underbrace{\delta^h}_{\frac{1}{c}}$$

**Error Bounds**  
for the minimum

**Optimal Size**

**Error Bounds**  
per hash/array

$$\Pr \left[ \hat{x}_i^h - x_i \geq \underbrace{\varepsilon^h}_{\frac{c}{w}} \|\mathbf{x}\|_1 \right] \leq \underbrace{\delta^h}_{\frac{1}{c}}$$

**Error Bounds**  
for the minimum

*The **estimate for each hash** has a well defined **L1 error bound**.*

**Optimal Size**

**Error Bounds**  
per hash/array

$$\Pr \left[ \hat{x}_i^h - x_i \geq \underbrace{\varepsilon^h}_{\frac{c}{w}} \|\mathbf{x}\|_1 \right] \leq \underbrace{\delta^h}_{\frac{1}{c}}$$

**Error Bounds**  
for the minimum

*The **estimate** for each hash has a well defined **L1 error bound**.*

**Optimal Size**

***What about the minimum?***

**Error Bounds**  
per hash/array

$$\Pr \left[ \hat{x}_i - x_i \geq \frac{c}{w} \|\mathbf{x}\|_1 \right] \leq ?$$

**Error Bounds**  
for the minimum

**Optimal Size**

**Error Bounds**  
per hash/array

**Error Bounds**  
for the minimum

**Optimal Size**

$$\Pr \left[ \underbrace{\min_{h \in h_1 \dots h_d} \hat{x}_i^h}_{\hat{x}_i} - x_i \geq \frac{c}{w} \|\mathbf{x}\|_1 \right] \leq ?$$

Multiple hash functions work like **independent trials**.

**Error Bounds**  
per hash/array

$$\Pr \left[ \underbrace{\min_{h \in h_1 \dots h_d} \hat{x}_i^h}_{\hat{x}_i} - x_i \geq \frac{c}{w} \|\mathbf{x}\|_1 \right] \leq ?$$

**Error Bounds**  
for the minimum

$\Leftrightarrow$

$$\prod_{h \in h_1 \dots h_d} \Pr \left[ \hat{x}_i^h - x_i \geq \frac{c}{w} \|\mathbf{x}\|_1 \right] \leq ?$$

**Optimal Size**



**Error Bounds**  
per hash/array

**Error Bounds**  
for the minimum

**Optimal Size**

$$\Pr \left[ \underbrace{\min_{h \in h_1 \dots h_d} \hat{x}_i^h}_{\hat{x}_i} - x_i \geq \frac{c}{w} \|\mathbf{x}\|_1 \right] \leq ?$$

$\Leftrightarrow$

$$\prod_{h \in h_1 \dots h_d} \underbrace{\Pr \left[ \hat{x}_i^h - x_i \geq \frac{c}{w} \|\mathbf{x}\|_1 \right]}_{\leq \frac{1}{c}} \leq ?$$

*error bound per hash*

**Error Bounds**  
per hash/array

**Error Bounds**  
for the minimum

**Optimal Size**

$$\Pr \left[ \underbrace{\min_{h \in h_1 \dots h_d} \hat{x}_i^h}_{\hat{x}_i} - x_i \geq \frac{c}{w} \|\mathbf{x}\|_1 \right] \leq ?$$

$\Leftrightarrow$

$$\prod_{h \in h_1 \dots h_d} \underbrace{\Pr \left[ \hat{x}_i^h - x_i \geq \frac{c}{w} \|\mathbf{x}\|_1 \right]}_{\leq \frac{1}{c}} \leq \frac{1}{c^d}$$

**Error Bounds**  
per hash/array

**Error Bounds**  
for the minimum

**Optimal Size**

$$\Pr \left[ \underbrace{\min_{h \in h_1 \dots h_d} \hat{x}_i^h}_{\hat{x}_i} - x_i \geq \frac{c}{w} \|\mathbf{x}\|_1 \right] \leq \frac{1}{c^d}$$

**Error Bounds**  
per hash/array

**Error Bounds**  
for the minimum

**Optimal Size**

$$\Pr \left[ \hat{x}_i - x_i \geq \frac{c}{w} \|\mathbf{x}\|_1 \right] \leq \frac{1}{c^d}$$

**Error Bounds**  
per hash/array

**Error Bounds**  
for the minimum

**Optimal Size**

$$\Pr \left[ \hat{x}_i - x_i \geq \underbrace{\varepsilon}_{\frac{c}{w}} \|\mathbf{x}\|_1 \right] \leq \underbrace{\delta}_{\frac{1}{c^d}}$$

*We have proven the error bounds!*

***But what about the constant  $c$ ?***

For **every**  $c$ , there is a pair  $(d, w)$  achieving the error bound and confidence  $(\varepsilon, \delta)$ .

**Error Bounds**  
per hash/array

$$\varepsilon = \frac{c}{w} \Rightarrow w = \left\lceil \frac{c}{\varepsilon} \right\rceil \quad (\text{hash range})$$

$$\delta = \frac{1}{c^d} \Rightarrow d = \left\lceil \log_c \frac{1}{\delta} \right\rceil \quad (\text{\#hashes})$$

**Error Bounds**  
for the minimum

**Optimal Size**

Choosing  $c=e$  **minimizes** the total **number of counters**.

**Error Bounds**  
per hash/array

$$\varepsilon = \frac{e}{w} \Rightarrow w = \left\lceil \frac{e}{\varepsilon} \right\rceil \quad (\text{hash range})$$

$$\delta = \frac{1}{e^d} \Rightarrow d = \left\lceil \ln \frac{1}{\delta} \right\rceil \quad (\text{\#hashes})$$

**Error Bounds**  
for the minimum

$$d \cdot w = \frac{c}{\varepsilon} \log_c \frac{1}{\delta} \stackrel{\text{minimize}}{=} \frac{e}{\varepsilon} \ln \frac{1}{\delta}$$

**Optimal Size**

# A **CountMin** sketch recipe

**Error Bounds**  
per hash/array

**Error Bounds**  
for the minimum

**Optimal Size**

**Given**  $\varepsilon, \delta$ , **choosing**

$$w = \left\lceil \frac{e}{\varepsilon} \right\rceil \quad (\text{hash range})$$

$$d = \left\lceil \ln \frac{1}{\delta} \right\rceil \quad (\text{\#hashes})$$

requires the **minimum number of counters** s.t. the CountMin Sketch can guarantee that

$$\hat{x}_i - x_i \geq \varepsilon \|\mathbf{x}\|_1$$

with a probability less than  $\delta$



*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.

### **CountMin sketch recipe**

**Choose**  $d = \left\lceil \ln \frac{1}{\delta} \right\rceil$ ,  $w = \left\lceil \frac{e}{\epsilon} \right\rceil$

**Then**  $\hat{x}_i - x_i \geq \epsilon \|\mathbf{x}\|_1$  with a probability less than  $\delta$

*A **CountMin sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L1 error bounds** for frequency queries.*

*→ only one design out of many!*

A **Count sketch** uses the same principles as a counting bloom filter, but is **designed** to have **provable L2 error bounds** for frequency queries.

The Count sketch uses **additional hashing** to give **L2 error bounds**, but requires more **resources**.

### CountMin sketch

$h_1, \dots, h_d: U \rightarrow \{1, \dots, w\}$

**COUNT**  $x_i$ :

for  $h$  in  $h_1, \dots, h_d$ :

$\text{Reg}_h[h(x_i)] + 1$

**QUERY**  $x_i$ :

return  $\min_{h \text{ in } h_1, \dots, h_d} ($

$\text{Reg}_h[h(x_i)]$

)

The Count sketch uses **additional hashing** to give **L2 error bounds**, but requires more **resources**.

### CountMin sketch

$h_1, \dots, h_d: U \rightarrow \{1, \dots, w\}$

**COUNT**  $x_i$ :

for  $h$  in  $h_1, \dots, h_d$ :

$\text{Reg}_h[h(x_i)] + 1$

**QUERY**  $x_i$ :

return  $\min_{h \text{ in } h_1, \dots, h_d} ($

$\text{Reg}_h[h(x_i)]$

)

### Count sketch

$h_1, \dots, h_d: U \rightarrow \{1, \dots, w\}$

$g: U \rightarrow \{+1, -1\}$

**COUNT**  $x_i$ :

for  $h$  in  $h_1, \dots, h_d$ :

$\text{Reg}_h[h(x_i)] + g(x_i)$

**QUERY**  $x_i$ :

return  $\text{median}_{h \text{ in } h_1, \dots, h_d} ($

$\text{Reg}_h[h(x_i)] * g(x_i)$

)

The Count sketch uses **additional hashing** to give **L2 error bounds**, but requires more **resources**.

***CountMin sketch recipe***

**Choose**  $d = \left\lceil \ln \frac{1}{\delta} \right\rceil, w = \left\lceil \frac{e}{\epsilon} \right\rceil$

**Then**  $\hat{x}_i - x_i \geq \epsilon \|\mathbf{x}\|_1$  with a probability less than  $\delta$

The Count sketch uses **additional hashing** to give **L2 error bounds**, but requires more **resources**.

### *CountMin sketch recipe*

**Choose**  $d = \left\lceil \ln \frac{1}{\delta} \right\rceil, w = \left\lceil \frac{e}{\epsilon} \right\rceil$

**Then**  $\hat{x}_i - x_i \geq \epsilon \|\mathbf{x}\|_1$  with a probability less than  $\delta$

### *Count sketch recipe*

**Choose**  $d = \left\lceil \ln \frac{1}{\delta} \right\rceil, w = \left\lceil \frac{e}{\epsilon^2} \right\rceil$

**Then**  $\hat{x}_i - x_i \geq \epsilon \|\mathbf{x}\|_2$  with a probability less than  $\delta$



# Sketches are the new black

...and many more!

## OpenSketch

NSDI '13

[source]

**Software Defined Traffic Measurement with OpenSketch**

Minlan Yu<sup>†</sup> Lavanya Jose\* Rui Miao<sup>†</sup>  
<sup>†</sup>University of Southern California \*Princeton University

**Abstract**

Most network management tasks in software-defined networks (SDN) involve two stages: measurement and control. While many efforts have been focused on network control APIs for SDN, little attention goes into measurement. The key challenge of designing a new measurement API is to strike a careful balance between generality (supporting a wide variety of measurement tasks) and efficiency (enabling high link speed and low cost). We propose a software defined traffic measurement architecture OpenSketch, which separates the measurement data plane from the control plane. In the data plane, OpenSketch provides a simple three-stage pipeline (hashing, filtering, and counting), which can be implemented with commodity switch components and support many measurement tasks. In the control plane, OpenSketch provides a measurement library that automatically configures the pipeline and allocates resources for different measurement tasks. Our evaluations of real-world packet traces, our prototype on NetFPGA, and the implementation of five measurement tasks on top of OpenSketch, demonstrate that OpenSketch is general, efficient and easily programmable.

**1 Introduction**

Recent advances in software-defined networking (SDN) have significantly improved network management. Network management involves two important stages: (1) measuring the network in real time (e.g., identifying traffic anomalies or large traffic aggregates) and then (2) adjusting the control of the network accordingly (e.g., routing, access control, and rate limiting). While there have been many efforts on designing the right APIs for network control (e.g., OpenFlow [29], ForCES [1], rule-based forwarding [33], etc.), little thought has gone into designing the right APIs for measurement. Since con-

work management, it is important to design and build a new software-defined measurement architecture. The key challenge is to strike a careful balance between generality (supporting a wide variety of measurement tasks) and efficiency (enabling high link speed and low cost).

Flow-based measurements such as NetFlow [2] and sFlow [42] provide generic support for different measurement tasks, but consume too resources (e.g., CPU, memory, bandwidth) [28, 18, 19]. For example, to identify the big flows whose byte volumes are above a threshold (i.e., heavy hitter detection which is important for traffic engineering in data centers [6]), NetFlow collects flow-level counts for sampled packets in the data plane. A high sampling rate would lead to too many counters, while a lower sampling rate may miss flows. While there are many NetFlow improvements for specific measurement tasks (e.g., [48, 19]), a different measurement task may need to focus on small flows (e.g., anomaly detection) and thus requiring another way of changing NetFlow. Instead, we should provide more customized and dynamic measurement data collection defined by the software written by operators based on the measurement requirements; and provide guarantees on the measurement accuracy.

As an alternative, many sketch-based streaming algorithms have been proposed in the theoretical research community [7, 12, 46, 8, 20, 47], which provide efficient measurement support for individual management tasks. However, these algorithms are not deployed in practice because of their lack of generality: Each of these algorithms answers just one question or produces just one statistic (e.g., the unique number of destinations), so it is too expensive for vendors to build new hardware to support each function. For example, the Space-Saving heavy hitter detection algorithm [8] maintains a hash table of items and counts, and requires customized operations such as keeping a pointer to the item with minimum counts and replacing the minimum-count entry with a

## UnivMon

SIGCOMM '16

[source]

**One Sketch to Rule Them All:  
Rethinking Network Flow Monitoring with UnivMon**

Zaoxing Liu<sup>1</sup>, Antonis Manousis<sup>1</sup>, Gregory Vorsanger<sup>1</sup>, Vyas Sekar<sup>1</sup>, Vladimir Braverman<sup>1</sup>  
<sup>1</sup>Johns Hopkins University · Carnegie Mellon University

**ABSTRACT**

Network management requires accurate estimates of metrics for many applications including traffic engineering (e.g., heavy hitters), anomaly detection (e.g., entropy of source addresses), and security (e.g., DDoS detection). Obtaining accurate estimates given router CPU and memory constraints is a challenging problem. Existing approaches fall in one of two undesirable extremes: (1) low fidelity general-purpose approaches such as sampling, or (2) high fidelity but complex algorithms customized to specific application-level metrics. Ideally, a solution should be both general (i.e., supports many applications) and provide accuracy comparable to custom algorithms. This paper presents UnivMon, a framework for flow monitoring which leverages recent theoretical advances and demonstrates that it is possible to achieve both generality and high accuracy. UnivMon uses an application-agnostic data plane monitoring primitive; different (and possibly unforeseen) estimation algorithms run in the control plane, and use the statistics from the data plane to compute application-level metrics. We present a proof-of-concept implementation of UnivMon using P4 and develop simple coordination techniques to provide a “one-big-switch” abstraction for network-wide monitoring. We evaluate the effectiveness of UnivMon using a range of trace-driven evaluations and show that it offers comparable (and sometimes better) accuracy relative to custom sketching solutions across a range of monitoring tasks.

**CCS Concepts**

•Networks → Network monitoring; Network measurement;

**Keywords**

Flow Monitoring, Sketching, Streaming Algorithms

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission from permissions@acm.org.

SIGCOMM '16, August 22–26, 2016, Florianoópolis, Brazil  
© 2016 ACM 978-1-4503-4160-8...\$15.00  
DOI: <http://dx.doi.org/10.1145/2934872.2934906>

**1 Introduction**

Network management is multi-faceted and encompasses a range of tasks including traffic engineering [11, 32], attack and anomaly detection [49], and forensic analysis [46]. Each such management task requires accurate and timely statistics on different application-level metrics of interest; e.g., the flow size distribution [37], heavy hitters [10], entropy measures [38, 50], or detecting changes in traffic patterns [44].

At a high level, there are two classes of techniques to estimate these metrics of interest. The first class of approaches relies on *generic flow monitoring*, typically with some form of packet sampling (e.g., NetFlow [25]). While generic flow monitoring is good for coarse-grained visibility, prior work has shown that it provides low accuracy for more fine-grained metrics [30, 31, 43]. These well-known limitations of sampling motivated an alternative class of techniques based on *sketching or streaming algorithms*. Here, custom online algorithms and data structures are designed for specific metrics of interest that can yield provable resource-accuracy trade-offs (e.g., [17, 18, 20, 31, 36, 38, 43]).

While the body of work in data streaming and sketching has made significant contributions, we argue that this trajectory of crafting special-purpose algorithms is untenable in the long term. As the number of monitoring tasks grows, this entails significant investment in algorithm design and hardware support for new metrics of interest. While recent tools like OpenSketch [47] and SCREAM [41] provide libraries to reduce the implementation effort and offer efficient resource allocation, they do not address the fundamental need to design and operate new custom sketches for each task. Furthermore, at any given point in time the data plane resources have to be committed (a priori) to a specific set of metrics to monitor and will have fundamental blind spots for other metrics that are not currently being tracked.

Ideally, we want a monitoring framework that offers both generality by delaying the binding to specific applications of interest but at the same time provides the required fidelity for estimating these metrics. Achieving generality and high fidelity simultaneously has been an elusive goal both in theory [33] (Question 24) as well as in practice [45].

In this paper, we present the UnivMon (short for Universal Monitoring) framework that can simultaneously achieve both generality and high fidelity across a broad spectrum of monitoring tasks [31, 36, 38, 51]. UnivMon builds on and

## SketchLearn

SIGCOMM '18

[source]

**SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference**

Qun Huang<sup>†</sup>, Patrick P. C. Lee<sup>‡</sup>, and Yungang Bao<sup>†</sup>  
<sup>†</sup>State Key Lab of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences  
<sup>‡</sup>Department of Computer Science and Engineering, The Chinese University of Hong Kong

**ABSTRACT**

Network measurement is challenged to fulfill stringent resource requirements in the face of massive network traffic. While approximate measurement can trade accuracy for resource savings, it demands intensive manual efforts to configure the right resource-accuracy trade-offs in real deployment. Such user burdens are caused by how existing approximate measurement approaches inherently deal with resource conflicts when tracking massive network traffic with limited resources. In particular, they tightly couple resource configurations with accuracy parameters, so as to provision sufficient resources to bound the measurement errors. We design SketchLearn, a novel sketch-based measurement framework that resolves resource conflicts by learning their statistical properties to eliminate conflicting traffic components. We prototype SketchLearn on OpenVSwitch and P4, and our testbed experiments and stress-test simulation show that SketchLearn accurately and automatically monitors various traffic statistics and effectively supports network-wide measurement with limited resources.

**CCS CONCEPTS**

•Networks → Network measurement;

**KEYWORDS**

Sketch; Network measurement

**ACM Reference Format:**

Qun Huang, Patrick P. C. Lee, and Yungang Bao. 2018. SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference. In SIGCOMM '18: ACM SIGCOMM 2018 Conference, August 20–25, 2018, Budapest, Hungary. ACM, New York, NY, USA, 17 pages. <https://doi.org/10.1145/3230543.3230559>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission from permissions@acm.org.

SIGCOMM '18, August 20–25, 2018, Budapest, Hungary  
© 2018 Association for Computing Machinery.  
ACM ISBN 978-1-4503-5160-8...\$15.00  
<https://doi.org/10.1145/3230543.3230559>

**1 INTRODUCTION**

Network measurement is indispensable to modern network management in clouds and data centers. Administrators measure a variety of traffic statistics, such as per-flow frequency, to infer the key behaviors or any unexpected patterns in operational networks. They use the measured traffic statistics to form the basis of management operations such as traffic engineering, performance diagnosis, and intrusion prevention. Unfortunately, measuring traffic statistics is non-trivial in the face of massive network traffic and large-scale network deployment. Error-free measurement requires per-flow tracking [15], yet today's data center networks can have thousands of concurrent flows in a very small period from 50ms [2] down to even 5ns [56]. This would require tremendous resources for performing per-flow tracking.

In view of the resource constraints, many approaches in the literature leverage approximation techniques to trade between resource usage and measurement accuracy. Examples include sampling [9, 37, 64], top-*k* counting [5, 43, 44, 46], and sketch-based approaches [18, 33, 40, 42, 58], which we collectively refer to as *approximate measurement approaches*. Their idea is to construct compact sub-linear data structures to record traffic statistics, backed by theoretical guarantees on how to achieve accurate measurement with limited resources. Approximate measurement has formed building blocks in many state-of-the-art network-wide measurement systems (e.g., [32, 48, 55, 66, 62, 67]), and is also adopted in production data centers [31, 68].

Although theoretically sound, existing approximate measurement approaches are inconvenient for use. In such approaches, massive network traffic competes for the limited resources, thereby introducing measurement errors due to resource conflicts (e.g., multiple flows are mapped to the same counter in sketch-based measurement). To mitigate errors, sufficient resources must be provisioned in approximate measurement based on its theoretical guarantees. Thus, there exists a *tight binding between resource configurations and accuracy parameters*. Such tight binding leads to several practical limitations (see §2.2 for details): (i) administrators need

# Sketches are the new black

## OpenSketch NSDI '13

[source]

**Software Defined Traffic Measurement with OpenSketch**

Minlan Yu<sup>†</sup> Lavanya Jose\* Rui Miao<sup>†</sup>  
<sup>†</sup>University of Southern California \*Princeton University

**Abstract**

Most network management tasks in software-defined networks (SDN) involve two stages: measurement and control. While many efforts have been focused on network control APIs for SDN, little attention goes into measurement. The key challenge of designing a new measurement API is to strike a careful balance between generality (supporting a wide variety of measurement tasks) and efficiency (enabling high link speed and low cost). We propose a software defined traffic measurement architecture OpenSketch, which separates the measurement data plane from the control plane. In the data plane, OpenSketch provides a simple three-stage pipeline (hashing, filtering, and counting), which can be implemented with commodity switch components and support many measurement tasks. In the control plane, OpenSketch provides a measurement library that automatically configures the pipeline and allocates resources for different measurement tasks. Our evaluations of real-world packet traces, our prototype on NetFGA, and the implementation of five measurement tasks on top of OpenSketch, demonstrate that OpenSketch is general, efficient and easily programmable.

**1 Introduction**

Recent advances in software-defined networking (SDN) have significantly improved network management. Network management involves two important stages: (1) measuring the network in real time (e.g., identifying traffic anomalies or large traffic aggregates) and then (2) adjusting the control of the network accordingly (e.g., routing, access control, and rate limiting). While there have been many efforts on designing the right APIs for network control (e.g., OpenFlow [29], ForCES [1], rule-based forwarding [33], etc.), little thought has gone into designing the right APIs for measurement. Since con-

work management, it is important to design and build a new software-defined measurement architecture. The key challenge is to strike a careful balance between generality (supporting a wide variety of measurement tasks) and efficiency (enabling high link speed and low cost).

Flow-based measurements such as NetFlow [2] and sFlow [42] provide generic support for different measurement tasks, but consume too resources (e.g., CPU, memory, bandwidth) [28, 18, 19]. For example, to identify the big flows whose byte volumes are above a threshold (i.e., heavy hitter detection which is important for traffic engineering in data centers [6]), NetFlow collects flow-level counts for sampled packets in the data plane. A high sampling rate would lead to too many counters, while a lower sampling rate may miss flows. While there are many NetFlow improvements for specific measurement tasks (e.g., [48, 19]), a different measurement task may need to focus on small flows (e.g., anomaly detection) and thus requiring another way of changing NetFlow. Instead, we should provide more customized and dynamic measurement data collection defined by the software written by operators based on the measurement requirements, and provide guarantees on the measurement accuracy.

As an alternative, many sketch-based streaming algorithms have been proposed in the theoretical research community [7, 12, 46, 8, 20, 47], which provide efficient measurement support for individual management tasks. However, these algorithms are not deployed in practice because of their lack of generality. Each of these algorithms answers just one question or produces just one statistic (e.g., the unique number of destinations), so it is too expensive for vendors to build new hardware to support each function. For example, the Space-Saving heavy hitter detection algorithm [8] maintains a hash table of items and counts, and requires customized operations such as keeping a pointer to the item with minimum counts and replacing the minimum-count entry with a

## UnivMon SIGCOMM '16

[source]

**One Sketch to Rule Them All:  
Rethinking Network Flow Monitoring with UnivMon**

Zaoxing Liu<sup>1</sup>, Antonis Manousis<sup>1</sup>, Gregory Vorsanger<sup>1</sup>, Vyas Sekar<sup>1</sup>, Vladimir Braverman<sup>1</sup>  
<sup>1</sup>Johns Hopkins University · Carnegie Mellon University

**ABSTRACT**

Network management requires accurate estimates of metrics for many applications including traffic engineering (e.g., heavy hitters), anomaly detection (e.g., entropy of source addresses), and security (e.g., DDoS detection). Obtaining accurate estimates given router CPU and memory constraints is a challenging problem. Existing approaches fall in one of two undesirable extremes: (1) low fidelity general-purpose approaches such as sampling, or (2) high fidelity but complex algorithms customized to specific application-level metrics. Ideally, a solution should be both general (i.e., supports many applications) and provide accuracy comparable to custom algorithms. This paper presents UnivMon, a framework for flow monitoring which leverages recent theoretical advances and demonstrates that it is possible to achieve both generality and high accuracy. UnivMon uses an application-agnostic data plane monitoring primitive, different (and possibly unforeseen) estimation algorithms run in the control plane, and use the statistics from the data plane to compute application-level metrics. We present a proof-of-concept implementation of UnivMon using P4 and develop simple coordination techniques to provide a "one-big-switch" abstraction for network-wide monitoring. We evaluate the effectiveness of UnivMon using a range of trace-driven evaluations and show that it offers comparable (and sometimes better) accuracy relative to custom sketching solutions across a range of monitoring tasks.

**CCS Concepts**

•Networks → Network monitoring; Network measurement;

**Keywords**

Flow Monitoring, Sketching, Streaming Algorithms

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permission from permissions@acm.org.

SIGCOMM '16, August 22–26, 2016, Florianopolis, Brazil  
© 2016 ACM. ISBN 978-1-4503-4194-6/16/08...\$15.00  
DOI: <http://dx.doi.org/10.1145/2934872.2934906>

**1 Introduction**

Network management is multi-faceted and encompasses a range of tasks including traffic engineering [11, 32], attack and anomaly detection [49], and forensic analysis [46]. Each such management task requires accurate and timely statistics on different application-level metrics of interest; e.g., the flow size distribution [37], heavy hitters [10], entropy measures [38, 50], or detecting changes in traffic patterns [44].

At a high level, there are two classes of techniques to estimate these metrics of interest. The first class of approaches relies on *generic flow monitoring*, typically with some form of packet sampling (e.g., NetFlow [25]). While generic flow monitoring is good for coarse-grained visibility, prior work has shown that it provides low accuracy for more fine-grained metrics [30, 31, 43]. These well-known limitations of sampling motivated an alternative class of techniques based on *sketching or streaming algorithms*. Here, custom online algorithms and data structures are designed for specific metrics of interest that can yield provable resource-accuracy trade-offs (e.g., [17, 18, 20, 31, 36, 38, 43]).

While the body of work in data streaming and sketching has made significant contributions, we argue that this trajectory of crafting special-purpose algorithms is untenable in the long term. As the number of monitoring tasks grows, this entails significant investment in algorithm design and hardware support for new metrics of interest. While recent tools like OpenSketch [47] and SCREAM [41] provide libraries to reduce the implementation effort and offer efficient resource allocation, they do not address the fundamental need to design and operate new custom sketches for each task. Furthermore, at any given point in time the data plane resources have to be committed (a priori) to a specific set of metrics to monitor and will have fundamental blind spots for other metrics that are not currently being tracked.

Ideally, we want a monitoring framework that offers both *generality* by delaying the binding to specific applications of interest but at the same time provides the required *fidelity* for estimating these metrics. Achieving generality and high fidelity simultaneously has been an elusive goal both in theory [33] (Question 24) as well as in practice [45].

In this paper, we present the UnivMon (short for Universal Monitoring) framework that can simultaneously achieve both generality and high fidelity across a broad spectrum of monitoring tasks [31, 36, 38, 51]. UnivMon builds on and

## SketchLearn SIGCOMM '18

[source]

**SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference**

Quan Huang<sup>†</sup>, Patrick P. C. Lee<sup>‡</sup>, and Yungang Bao<sup>†</sup>  
<sup>†</sup>State Key Lab of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences  
<sup>‡</sup>Department of Computer Science and Engineering, The Chinese University of Hong Kong

**ABSTRACT**

Network measurement is challenged to fulfill stringent resource requirements in the face of massive network traffic. While approximate measurement can trade accuracy for resource savings, it demands intensive manual efforts to configure the right resource-accuracy trade-offs in real deployment. Such user burdens are caused by how existing approximate measurement approaches inherently deal with resource conflicts when tracking massive network traffic with limited resources. In particular, they tightly couple resource configurations with accuracy parameters, so as to provision sufficient resources to bound the measurement errors. We design SketchLearn, a novel sketch-based measurement framework that resolves resource conflicts by learning their statistical properties to eliminate conflicting traffic components. We prototype SketchLearn on OpenVswitch and P4, and our tested experiments and stress-test simulation show that SketchLearn accurately and automatically monitors various traffic statistics and effectively supports network-wide measurement with limited resources.

**CCS CONCEPTS**

•Networks → Network measurement;

**KEYWORDS**

Sketch; Network measurement

**ACM Reference Format:**  
Quan Huang, Patrick P. C. Lee, and Yungang Bao. 2018. SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference. In SIGCOMM '18. ACM SIGCOMM 2018, August 20–25, 2018, Budapest, Hungary. ACM, New York, NY, USA, 17 pages. <https://doi.org/10.1145/3230543.3230559>

**1 INTRODUCTION**

Network measurement is indispensable to modern network management in clouds and data centers. Administrators measure a variety of traffic statistics, such as per-flow frequency, to infer the key behaviors or any unexpected patterns in operational networks. They use the measured traffic statistics to form the basis of management operations such as traffic engineering, performance diagnosis, and intrusion prevention. Unfortunately, measuring traffic statistics is non-trivial in the face of massive network traffic and large-scale network deployment. Error-free measurement requires per-flow tracking [15], yet today's data center networks can have thousands of concurrent flows in a very small period from 50ms [2] down to even 5ms [56]. This would require tremendous resources for performing per-flow tracking.

In view of the resource constraints, many approaches in the literature leverage approximation techniques to trade between resource usage and measurement accuracy. Examples include sampling [9, 37, 64], top-k counting [5, 43, 44, 46], and sketch-based approaches [18, 33, 40, 42, 58], which we collectively refer to as *approximate measurement* approaches. Their idea is to construct compact sub-linear data structures to record traffic statistics, backed by theoretical guarantees on how to achieve accurate measurement with limited resources. Approximate measurement has formed building blocks in many state-of-the-art network-wide measurement systems (e.g., [32, 48, 55, 60, 62, 67]), and is also adopted in production data centers [31, 68].

Although theoretically sound, existing approximate measurement approaches are inconvenient for use. In such approaches, massive network traffic competes for the limited resources, thereby introducing measurement errors due to resource conflicts (e.g., multiple flows are mapped to the same counter in sketch-based measurement). To mitigate errors, sufficient resources must be provisioned in approximate measurement based on its theoretical guarantees. Thus, there exists a *tight binding* between resource configurations and accuracy parameters. Such tight binding leads to several practical limitations (see §2.2 for details): (i) administrators need



# SketchLearn combines multiple sketches with elaborate **post-processing for flexibility**

## SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference

Qun Huang<sup>†</sup>, Patrick P. C. Lee<sup>‡</sup>, and Yungang Bao<sup>†</sup>

<sup>†</sup>State Key Lab of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences

<sup>‡</sup>Department of Computer Science and Engineering, The Chinese University of Hong Kong

### ABSTRACT

Network measurement is challenged to fulfill stringent resource requirements in the face of massive network traffic. While approximate measurement can trade accuracy for resource savings, it demands intensive manual efforts to configure the right resource-accuracy trade-offs in real deployment. Such user burdens are caused by how existing approximate measurement approaches inherently deal with resource conflicts when tracking massive network traffic with limited resources. In particular, they tightly couple resource configurations with accuracy parameters, so as to provision sufficient resources to bound the measurement errors. We design SketchLearn, a novel sketch-based measurement framework that resolves resource conflicts by learning their statistical properties to eliminate conflicting traffic components. We prototype SketchLearn on OpenVSwitch and P4, and our testbed experiments and stress-test simulation show that SketchLearn accurately and automatically monitors various traffic statistics and effectively supports network-wide measurement with limited resources.

### CCS CONCEPTS

• Networks → Network measurement;

### KEYWORDS

Sketch; Network measurement

### ACM Reference Format:

Qun Huang, Patrick P. C. Lee, and Yungang Bao. 2018. SketchLearn: Relieving User Burdens in Approximate Measurement with Au-

2018 Conference, August 20–25, 2018, Budapest, Hungary. ACM, New York, NY, USA, 17 pages. <https://doi.org/10.1145/3230543.3230559>

### 1 INTRODUCTION

Network measurement is indispensable to modern network management in clouds and data centers. Administrators measure a variety of traffic statistics, such as per-flow frequency, to infer the key behaviors or any unexpected patterns in operational networks. They use the measured traffic statistics to form the basis of management operations such as traffic engineering, performance diagnosis, and intrusion prevention. Unfortunately, measuring traffic statistics is non-trivial in the face of massive network traffic and large-scale network deployment. Error-free measurement requires per-flow tracking [15], yet today's data center networks can have thousands of concurrent flows in a very small period from 50ms [2] down to even 5ms [56]. This would require tremendous resources for performing per-flow tracking.

In view of the resource constraints, many approaches in the literature leverage approximation techniques to trade between resource usage and measurement accuracy. Examples include sampling [9, 37, 64], top- $k$  counting [5, 43, 44, 46], and sketch-based approaches [18, 33, 40, 42, 58], which we collectively refer to as *approximate measurement* approaches. Their idea is to construct compact sub-linear data structures to record traffic statistics, backed by theoretical guarantees on how to achieve accurate measurement with limited resources. Approximate measurement has formed building blocks in many state-of-the-art network-wide measurement systems (e.g., [32, 48, 55, 60, 62, 67]), and is also adopted in

Today we'll talk about: important questions,  
how 'sketches' answer them,  
**limitations of 'sketches'**

Sketches **compute statistical summaries**, favoring elements with **high frequency**.

$$\Pr \left[ \hat{x}_i - x_i \geq \varepsilon \|\mathbf{x}\|_1 \right] \leq \delta$$

*estimation error*                      *relative to sum of all elements*

Sketches **compute statistical summaries**, favoring elements with **high frequency**.

*Let*  $\varepsilon = 0.01$ ,  $\|\mathbf{x}\|_1 = 10000$  ( $\Rightarrow \varepsilon \cdot \|\mathbf{x}\|_1 = 100$ )

*Assume two flows*  $x_a$ ,  $x_b$ ,

*with*  $\|x_a\|_1 = 1000$ ,  $\|x_b\|_1 = 50$

*high frequency*

*low frequency*

Sketches **compute statistical summaries**, favoring elements with **high frequency**.

*Let*  $\varepsilon = 0.01$ ,  $\|\mathbf{x}\|_1 = 10000$  ( $\Rightarrow \varepsilon \cdot \|\mathbf{x}\|_1 = 100$ )

*Assume two flows*  $x_a$ ,  $x_b$ ,

*with*  $\|x_a\|_1 = 1000$ ,  $\|x_b\|_1 = 50$

Error relative to **stream size**: 1%

Sketches **compute statistical summaries**, favoring elements with **high frequency**.

*Let*  $\varepsilon = 0.01$ ,  $\|\mathbf{x}\|_1 = 10000$  ( $\Rightarrow \varepsilon \cdot \|\mathbf{x}\|_1 = 100$ )

*Assume two flows*  $x_a$ ,  $x_b$ ,

*with*  $\|x_a\|_1 = 1000$ ,  $\|x_b\|_1 = 50$

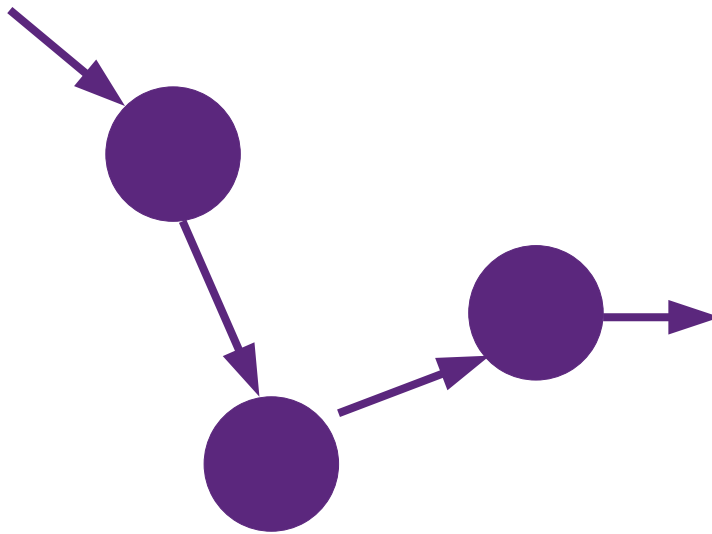
Error relative to **stream size**: 1%

**flow size**:  $x_a$ : 10%,  $x_b$ : **200%**

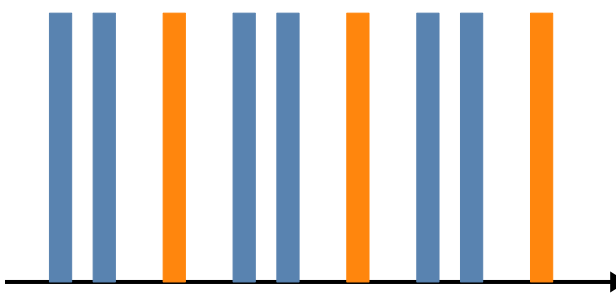


# Other Problems a Sketch **can't** handle

*causality*



*patterns*



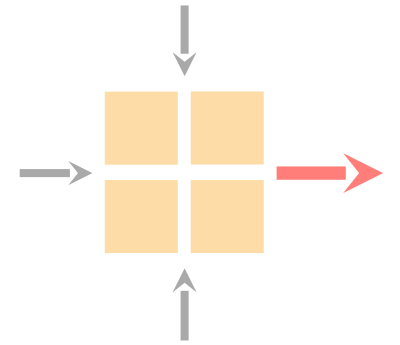
*rare things*



*Regardless of their limitations, sketches provide **trade-offs between resources and error, and provable guarantees to rely on.***

# Advanced Topics in Communication Networks

## Programming Network Data Planes



Alexander Dietmüller

[nsg.ee.ethz.ch](http://nsg.ee.ethz.ch)

ETH Zürich

Oct. 11 2018