

# Tight Bounds for Adversarially Robust Streams and Sliding Windows via Difference Estimators



David P. Woodruff  
Samson Zhou



**Carnegie  
Mellon  
University**

# Model #1: Streaming Model

- ❖ **Input:** Elements of an underlying data set  $S$ , which arrives sequentially
- ❖ **Output:** Evaluation (or approximation) of a given function
- ❖ **Goal:** Use space *sublinear* in the size  $m$  of the input  $S$

1 0 1 1 1 0 0 1

# Heavy-Hitters

- ❖ Given a set  $S$  of  $m$  elements from  $[n]$ , let  $f_i$  be the frequency of element  $i$ . (How often it appears)

$$1\ 1\ 2\ 1\ 2\ 1\ 1\ 2\ 3 \rightarrow [5, 3, 1, 0] := f$$

# Heavy-Hitters

- ❖ Given a set  $S$  of  $m$  elements from  $[n]$ , let  $f_i$  be the frequency of element  $i$ . (How often it appears)
- ❖ Let  $L_2$  be the norm of the frequency vector:

$$L_2 = \sqrt{f_1^2 + f_2^2 + \dots + f_n^2}$$

- ❖ **Goal:** Given a set  $S$  of  $m$  elements from  $[n]$  and a threshold  $\epsilon$ , output the elements  $i$  such that  $f_i > \epsilon L_2$ ...and no elements  $j$  such that  $f_j < \frac{\epsilon}{16} L_2$
- ❖ **Motivation:** DDoS prevention, iceberg queries

# Frequency Moments

- ❖ Given a set  $S$  of  $m$  elements from  $[n]$ , let  $f_i$  be the frequency of element  $i$ . (How often it appears)
- ❖ Let  $F_p$  be the frequency moment of the vector:

$$F_p = f_1^p + f_2^p + \cdots + f_n^p$$

- ❖ **Goal:** Given a set  $S$  of  $m$  elements from  $[n]$  and an accuracy parameter  $\epsilon$ , output a  $(1 + \epsilon)$ -approximation to  $F_p$
- ❖ **Motivation:** Entropy estimation, linear regression

# Distinct Elements

- ❖ Given a set  $S$  of  $m$  elements from  $[n]$ , let  $f_i$  be the frequency of element  $i$ . (How often it appears)
- ❖ Let  $F_0$  be the frequency moment of the vector:

$$F_0 = |\{i : f_i \neq 0\}|$$

- ❖ **Goal:** Given a set  $S$  of  $m$  elements from  $[n]$  and an accuracy parameter  $\epsilon$ , output a  $(1 + \epsilon)$ -approximation to  $F_0$
- ❖ **Motivation:** Traffic monitoring

# $(1 + \epsilon)$ -Approximation Streaming Algorithms

- ❖ Space  $O\left(\frac{1}{\epsilon^2} + \log n\right)$  algorithm for  $F_0$  [KaneNelsonWoodruff10], [Blasiok20]
- ❖ Space  $O\left(\frac{1}{\epsilon^2} \log n\right)$  algorithm for  $F_p$  with  $p \in (0, 2]$  [BlasiokDingNelson17]
- ❖ Space  $O\left(\frac{1}{\epsilon^2} n^{1-2/p} \log^2 n\right)$  algorithm for  $F_p$  with  $p > 2$  [Ganguly11, GangulyWoodruff18]
- ❖ Space  $O\left(\frac{1}{\epsilon^2} \log n\right)$  algorithm for  $L_2$ -heavy hitters [BravermanChestnutIvkinNelsonWangWoodruff17]

# Model #2: Adversarially Robust Streaming

- ❖ **Input:** Elements of an underlying data set  $S$ , which arrives sequentially and *adversarially*
- ❖ **Output:** Evaluation (or approximation) of a given function
- ❖ **Goal:** Use space *sublinear* in the size  $m$  of the input  $S$



1

1





# Model #2: Adversarially Robust Streaming

- ❖ **Input:** Elements of an underlying data set  $S$ , which arrives sequentially and *adversarially*
- ❖ **Output:** Evaluation (or approximation) of a given function
- ❖ **Goal:** Use space *sublinear* in the size  $m$  of the input  $S$



10

1



# Model #2: Adversarially Robust Streaming

- ❖ **Input:** Elements of an underlying data set  $S$ , which arrives sequentially and *adversarially*
- ❖ **Output:** Evaluation (or approximation) of a given function
- ❖ **Goal:** Use space *sublinear* in the size  $m$  of the input  $S$



101

2



# Model #2: Adversarially Robust Streaming

- ❖ **Input:** Elements of an underlying data set  $S$ , which arrives sequentially and *adversarially*
- ❖ **Output:** Evaluation (or approximation) of a given function
- ❖ **Goal:** Use space *sublinear* in the size  $m$  of the input  $S$



1010

3



# Model #2: Adversarially Robust Streaming

- ❖ **Input:** Elements of an underlying data set  $S$ , which arrives sequentially and *adversarially*
- ❖ **Output:** Evaluation (or approximation) of a given function
- ❖ **Goal:** Use space *sublinear* in the size  $m$  of the input  $S$



1010

3



# Model #2: Adversarially Robust Streaming

- ❖ **Input:** Elements of an underlying data set  $S$ , which arrives sequentially and *adversarially*
- ❖ **Output:** Evaluation (or approximation) of a given function
- ❖ **Goal:** Use space *sublinear* in the size  $m$  of the input  $S$
  
- ❖ **Adversarially Robust:** “Future queries may depend on previous queries”
- ❖ **Motivation:** Database queries, adversarial ML

# $(1 + \epsilon)$ -Robust Algorithms [Ben-EliezerJayaramWoodruffYogev20]

- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^3} \log n\right)$  algorithm for  $F_0$
- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^3} \log n\right)$  algorithm for  $F_p$  with  $p \in (0, 2]$
- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^3} n^{1-2/p}\right)$  algorithm for  $F_p$  with  $p > 2$
- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^3} \log n\right)$  algorithm for  $L_2$ -heavy hitters

“A general framework that loses\* nothing in  $n$  and only  $\frac{1}{\epsilon}$ ”

# “What’s an epsilon between friends?”

- ❖ Unsigned integer range:  $n = 2^{32} \sim 4B$
- ❖ Statista:  $\sim 300B$  e-mails sent per day
- ❖ Accuracy:  $\epsilon = 0.01$
- ❖ Since  $\frac{1}{\epsilon} > \log n$ , we should care about  $\frac{1}{\epsilon}$  factors!

# $(1 + \epsilon)$ -Robust Algorithms

[HassidimKaplanMansourMatiasStemmer20]

- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^{2.5}} \log^4 n\right)$  algorithm for  $F_0$
- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^{2.5}} \log^4 n\right)$  algorithm for  $F_p$  with  $p \in (0, 2]$
- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^{2.5}} n^{1-2/p}\right)$  algorithm for  $F_p$  with  $p > 2$
- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^{2.5}} \log^4 n\right)$  algorithm for  $L_2$ -heavy hitters

“ $\frac{1}{\epsilon}$  losses are not necessary”



# Our Results: $(1 + \epsilon)$ -Robust Algorithms

- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^2} \log n\right)$  algorithm for  $F_0$
- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^2} \log n\right)$  algorithm for  $F_p$  with  $p \in (0, 2]$
- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^2} n^{1-2/p}\right)$  algorithm for  $F_p$  with integer  $p > 2$
- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^2} \log n\right)$  algorithm for  $L_2$ -heavy hitters

“No losses\* are necessary!”

# Summary: $(1 + \epsilon)$ -Robust Algorithms

Problem	[BJWY20] Space	[HKM <sup>+</sup> 20] Space	Our Result
Distinct Elements	$\tilde{O}\left(\frac{\log n}{\epsilon^3}\right)$	$\tilde{O}\left(\frac{\log^4 n}{\epsilon^{2.5}}\right)$	$\tilde{O}\left(\frac{\log n}{\epsilon^2}\right)$
$F_p$ Estimation, $p \in (0, 2]$	$\tilde{O}\left(\frac{\log n}{\epsilon^3}\right)$	$\tilde{O}\left(\frac{\log^4 n}{\epsilon^{2.5}}\right)$	$\tilde{O}\left(\frac{\log n}{\epsilon^2}\right)$
$L_2$ -Heavy Hitters	$\tilde{O}\left(\frac{\log n}{\epsilon^3}\right)$	$\tilde{O}\left(\frac{\log^4 n}{\epsilon^{2.5}}\right)$	$\tilde{O}\left(\frac{\log n}{\epsilon^2}\right)$
$F_p$ Estimation, integer $p > 2$	$\tilde{O}\left(\frac{n^{1-2/p}}{\epsilon^3}\right)$	$\tilde{O}\left(\frac{n^{1-2/p}}{\epsilon^{2.5}}\right)$	$\tilde{O}\left(\frac{n^{1-2/p}}{\epsilon^2}\right)$
$F_p$ Estimation, $p \in (0, 2]$ , flip number $\lambda$	$\tilde{O}\left(\frac{\lambda \log^2 n}{\epsilon^2}\right)$	$\tilde{O}\left(\frac{\log^3 n \sqrt{\lambda \log n}}{\epsilon^2}\right)$	$\tilde{O}\left(\frac{\lambda \log^2 n}{\epsilon}\right)$

# Model #3: Sliding Window Model

- ❖ **Input:** Elements of an underlying data set  $S$ , which arrives sequentially
- ❖ **Output:** Evaluation (or approximation) of a given function
- ❖ **Goal:** Use space *sublinear* in the size  $m$  of the input  $S$
  
- ❖ **Sliding Window:** “Only the  $m$  most recent updates form the underlying data set  $S$ ”

1 0 1 1 1 0 0 1

# Model #3: Sliding Window Model

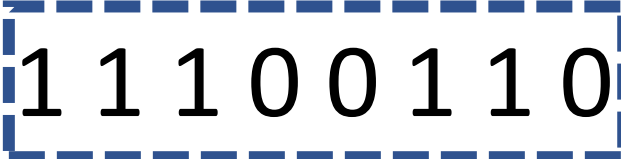
- ❖ **Input:** Elements of an underlying data set  $S$ , which arrives sequentially
- ❖ **Output:** Evaluation (or approximation) of a given function
- ❖ **Goal:** Use space *sublinear* in the size  $m$  of the input  $S$
  
- ❖ **Sliding Window:** “Only the  $m$  most recent updates form the underlying data set  $S$ ”

1 0 1 1 1 0 0 1 1

# Model #3: Sliding Window Model

- ❖ **Input:** Elements of an underlying data set  $S$ , which arrives sequentially
- ❖ **Output:** Evaluation (or approximation) of a given function
- ❖ **Goal:** Use space *sublinear* in the size  $m$  of the input  $S$
  
- ❖ **Sliding Window:** “Only the  $m$  most recent updates form the underlying data set  $S$ ”


1 0 1 1 1 0 0 1 1 0



# Model #3: Sliding Window Model

- ❖ **Input:** Elements of an underlying data set  $S$ , which arrives sequentially
- ❖ **Output:** Evaluation (or approximation) of a given function
- ❖ **Goal:** Use space *sublinear* in the size  $m$  of the input  $S$
- ❖ **Sliding Window:** “Only the  $m$  most recent updates form the underlying data set  $S$ ”
  - ❖ Emphasizes recent interactions, appropriate for time sensitive settings

1 0 1 1 1 0 0 1 1 0 1



# $(1 + \epsilon)$ -Approximation Sliding Window Algorithms

- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^2} \log n\right)$  algorithm for  $F_0$   
[BravermanGrigorescuLangWoodruffZhou18]
- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^{2+p}} n^{1-2/p}\right)$  algorithm for  $F_p$  with  $p > 2$   
[BravermanOstrovsky07]
- ❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^2} \log n\right)$  algorithm for  $L_2$ -heavy hitters  
[BravermanGrigorescuLangWoodruffZhou18]

# $(1 + \epsilon)$ -Approximation Sliding Window Algorithms

- ❖ Space  $O\left(\frac{1}{\epsilon^3} \log^3 n\right)$  algorithm for  $F_p$  with  $p \in (0,1)$   
[BravermanOstrovsky07]
- ❖ Space  $O\left(\frac{1}{\epsilon^{2+p}} \log^3 n\right)$  algorithm for  $F_p$  with  $p \in (1,2]$   
[BravermanOstrovsky07]

“A general framework that loses\* nothing in  $n$  and only  $\frac{1}{\epsilon}$ ”



# Our Results: $(1 + \epsilon)$ -Approximation Sliding Window Algorithms

❖ Space  $\tilde{O}\left(\frac{1}{\epsilon^2} \log n\right)$  algorithm for  $F_p$  with  $p \in (0, 2]$

Problem	[BO07] Space	Our Result
$L_p$ Estimation, $p \in (0, 1)$	$\tilde{O}\left(\frac{\log^3 n}{\epsilon^3}\right)$	$\tilde{O}\left(\frac{\log^2 n}{\epsilon^2}\right)$
$L_p$ Estimation, $p \in (1, 2]$	$\tilde{O}\left(\frac{\log^3 n}{\epsilon^{2+p}}\right)$	$\tilde{O}\left(\frac{\log^2 n}{\epsilon^2}\right)$

“No losses\* are necessary!”

# Format

- ❖ Part 1: Background
- ❖ Part 2: Frameworks
- ❖ Part 3: Difference Estimators

Questions?



# AMS $F_2$ Algorithm

- ❖ Let  $s \in \{-1, +1\}^n$  be a sign vector of length  $n$
- ❖ Let  $Z = \langle s, f \rangle = s_1 f_1 + \dots + s_n f_n$  and consider  $Z^2$

$$E[Z^2] = \sum_{i,j} E[s_i s_j f_i f_j] = f_1^2 + \dots + f_n^2$$

$$\text{Var}[Z^2] \leq \sum_{i,j} E[s_i s_j s_k s_l f_i f_j f_k f_l] \leq 2F_2^2$$

- ❖ Take the mean of  $O\left(\frac{1}{\epsilon^2}\right)$  inner products for  $(1 + \epsilon)$ -approximation  
[AlonMatiasSzegedy99]

# “Attack” on AMS

- ❖ Can learn whether  $s_i = s_j$  from  $\langle s, e_i + e_j \rangle$
- ❖ Let  $f_i = 1$  if  $s_i = s_1$  and  $f_i = -1$  if  $s_i \neq s_1$
- ❖  $Z = \langle s, f \rangle = s_1 f_1 + \dots + s_n f_n = m$  and  $Z^2 = m^2$  deterministically
  
- ❖ What happened? Randomness of algorithm not independent of input

# Reconstruction Attack on Linear Sketches

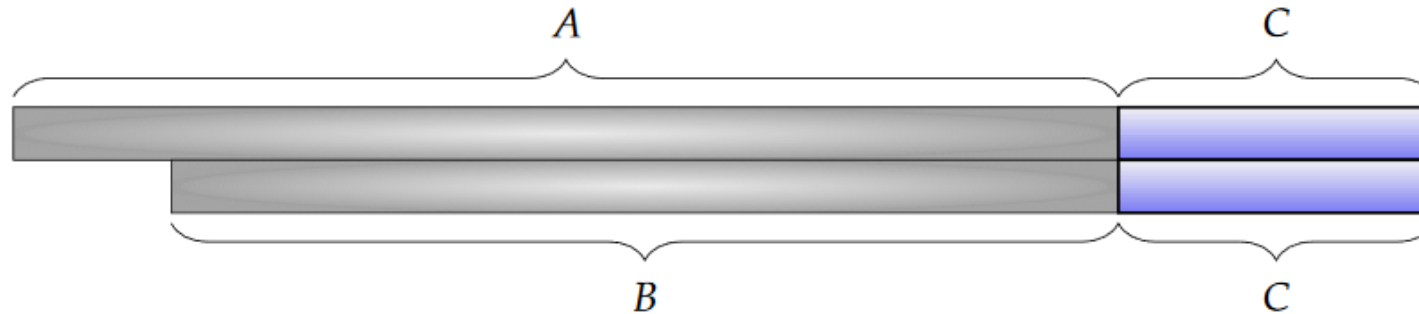
- ❖ Linear sketches are “not robust” to adversarial attacks, must use  $\Omega(n)$  space [HardtWoodruff13]
- ❖ Approximately learn sketch matrix  $U$ , query something in the kernel of  $U$
- ❖ Iterative process, start with  $V_1, \dots, V_r$
- ❖ **Correlation finding**: Find vectors weakly correlated with  $U$  orthogonal to  $V_{i-1}$
- ❖ **Boosting**: Use these vectors to find strongly correlated vector  $v$
- ❖ **Progress**: Set  $V_i = \text{span}(V_{i-1}, v)$

# Insertion-Only Streams

- ❖ **Key:** Deletions are needed to perform this attack
- ❖ Similar lower bounds for the sliding window model  
[\[DatarGionisIndykMotwani02\]](#)
- ❖ Assume insertion-only updates
- ❖ How do the previous results work?

# Sliding Window Algorithms

- ❖ Suppose we are trying to approximate some given function
  1. Suppose we have a streaming algorithm for this function
  2. Suppose this function is “smooth”: If  $f(B)$  is a “good” approximation to  $f(A)$ , then  $f(B \cup C)$  will always be a “good” approximation to  $f(A \cup C)$ .



- ❖ Smooth histogram framework [\[BravermanOstrovsky07\]](#) gives a sliding window algorithm for this function

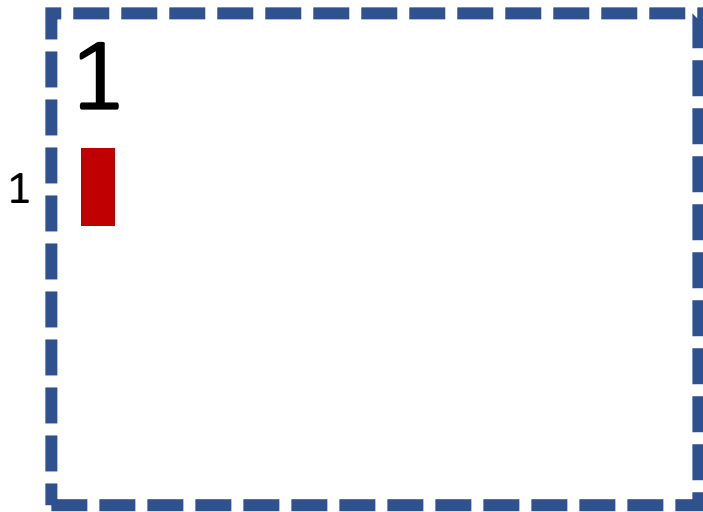
# Smooth Histogram

- ❖ Suppose we are trying to approximate some given function
- ❖ Smooth histogram framework [BO07] gives a sliding window algorithm for this function
- ❖ Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- ❖ Each time there are three instances that report “close” values, delete the middle one
- ❖ Use different checkpoints to “sandwich” the sliding window



# Smooth Histogram

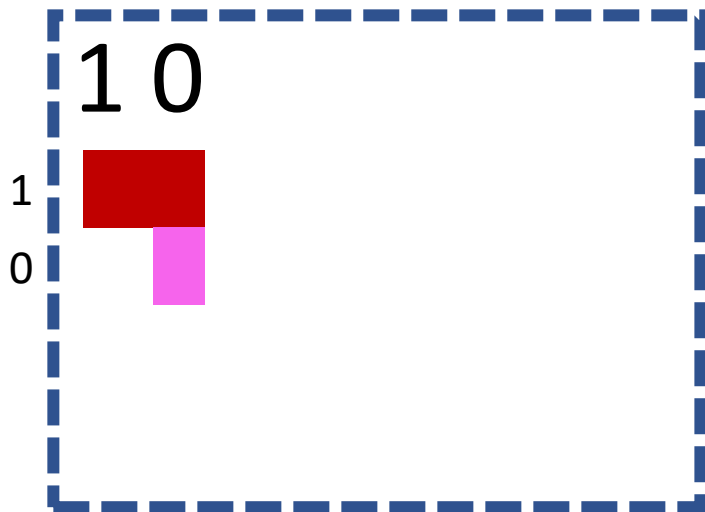
- ❖ Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- ❖ Each time there are three instances that report “close” values, delete the middle one
- ❖ Use different checkpoints to “sandwich” the sliding window



- ❖ Example: Number of ones in sliding window (2-approximation)

# Smooth Histogram

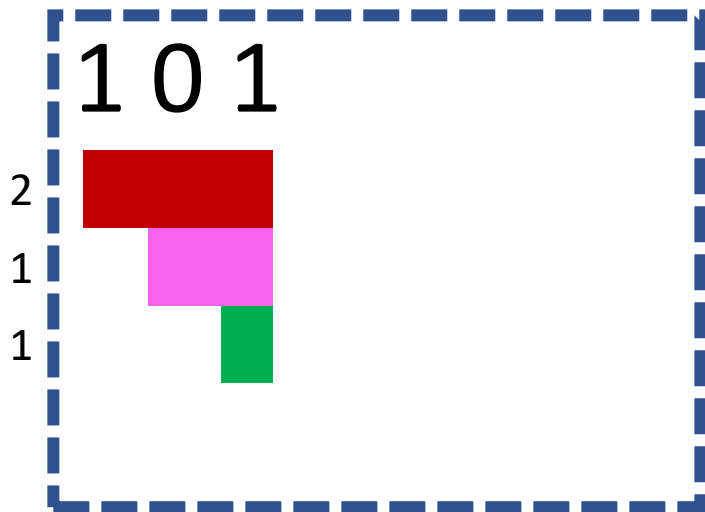
- ❖ Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- ❖ Each time there are three instances that report “close” values, delete the middle one
- ❖ Use different checkpoints to “sandwich” the sliding window



- ❖ Example: Number of ones in sliding window (2-approximation)

# Smooth Histogram

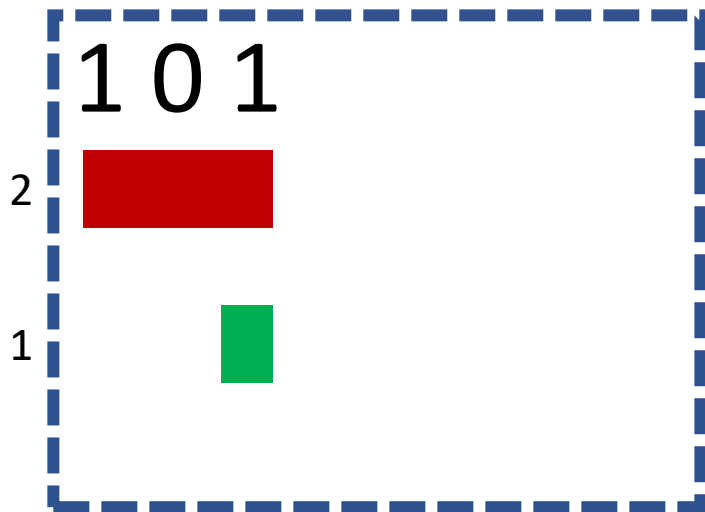
- ❖ Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- ❖ Each time there are three instances that report “close” values, delete the middle one
- ❖ Use different checkpoints to “sandwich” the sliding window



- ❖ Example: Number of ones in sliding window (2-approximation)

# Smooth Histogram

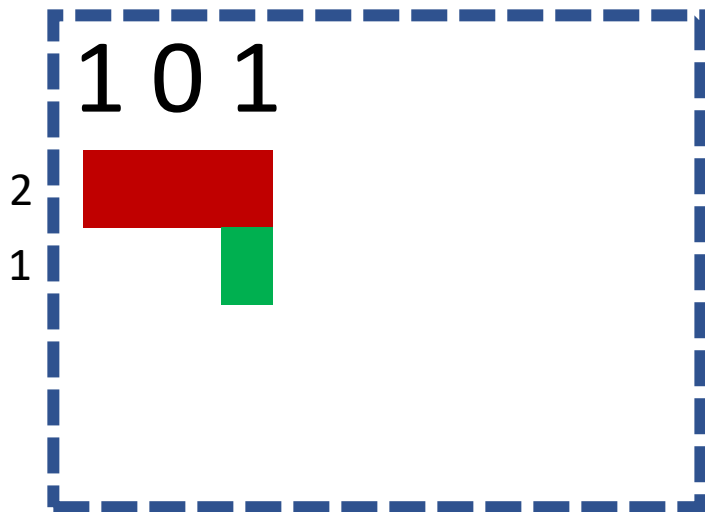
- ❖ Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- ❖ Each time there are three instances that report “close” values, delete the middle one
- ❖ Use different checkpoints to “sandwich” the sliding window



- ❖ Example: Number of ones in sliding window (2-approximation)

# Smooth Histogram

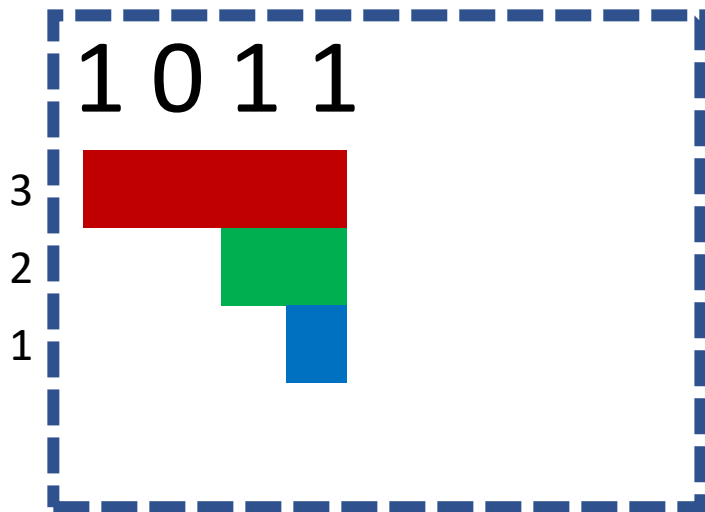
- ❖ Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- ❖ Each time there are three instances that report “close” values, delete the middle one
- ❖ Use different checkpoints to “sandwich” the sliding window



- ❖ Example: Number of ones in sliding window (2-approximation)

# Smooth Histogram

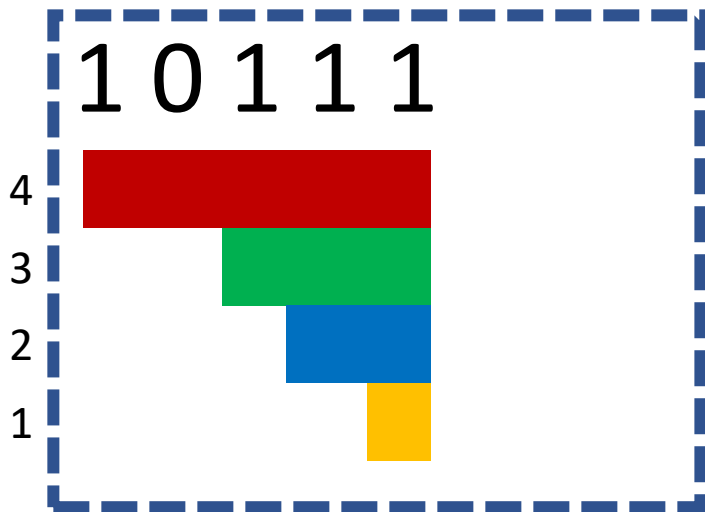
- ❖ Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- ❖ Each time there are three instances that report “close” values, delete the middle one
- ❖ Use different checkpoints to “sandwich” the sliding window



- ❖ Example: Number of ones in sliding window (2-approximation)

# Smooth Histogram

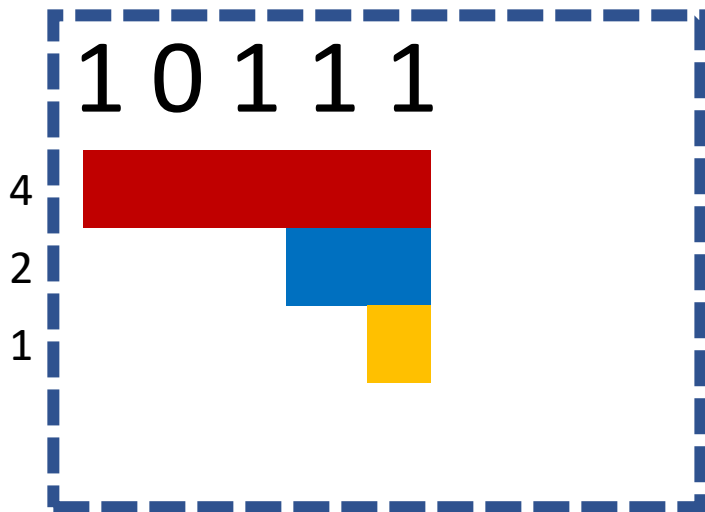
- ❖ Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- ❖ Each time there are three instances that report “close” values, delete the middle one
- ❖ Use different checkpoints to “sandwich” the sliding window



- ❖ Example: Number of ones in sliding window (2-approximation)

# Smooth Histogram

- ❖ Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- ❖ Each time there are three instances that report “close” values, delete the middle one
- ❖ Use different checkpoints to “sandwich” the sliding window

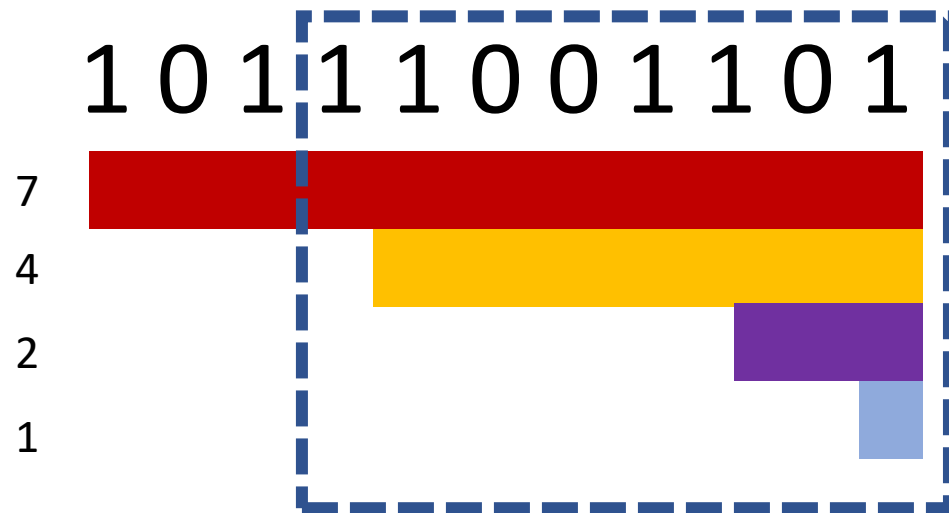


- ❖ Example: Number of ones in sliding window (2-approximation)



# Smooth Histogram

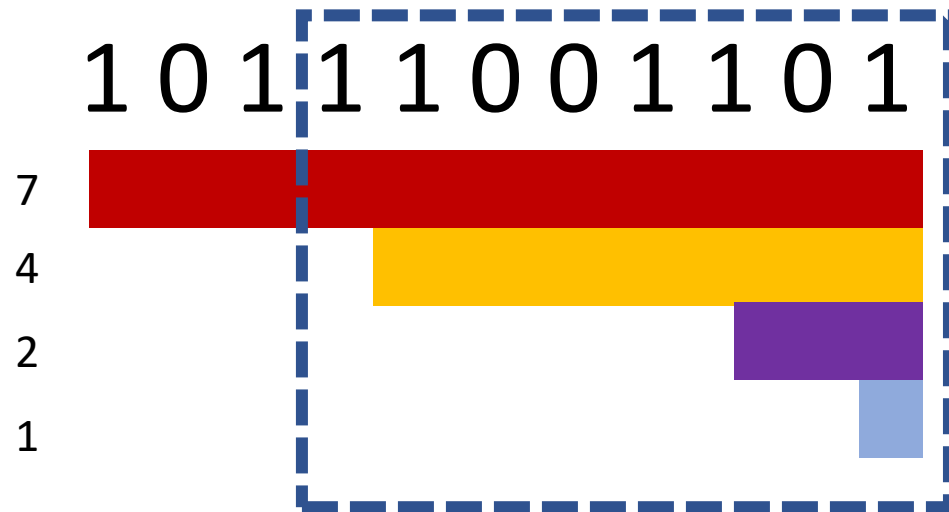
- ❖ Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- ❖ Each time there are three instances that report “close” values, delete the middle one
- ❖ Use different checkpoints to “sandwich” the sliding window



- ❖ Example: Number of ones in sliding window (2-approximation)

# Smooth Histogram

- ❖ Start a new instance of the streaming algorithm (along with existing instances) each time a new element arrives
- ❖ Each time there are three instances that report “close” values, delete the middle one
- ❖ Use different checkpoints to “sandwich” the sliding window



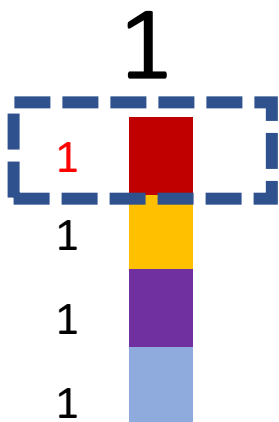
- ❖ Example: Number of ones in sliding window (2-approximation)
- ❖ Number of ones in sliding window is at least 4 and at most 7
- ❖ 4 is a good approximation

# Robust Algorithms

- ❖ Suppose we are trying to approximate some given function
  1. Suppose we have a streaming algorithm for this function
  2. Suppose this function is monotonic and the stream is insertion-only
- ❖ Sketch switching framework [[Ben-EliezerJayaramWoodruffYogev20](#)] gives a robust for this function
- ❖ Start many instances of the streaming algorithm at the beginning
- ❖ Use an instance of the algorithm but “freeze” the output
- ❖ Each time the next instance has value  $(1 + O(\epsilon))$  more than the “frozen” output, use the next instance and “freeze” its output

# Robust Algorithms

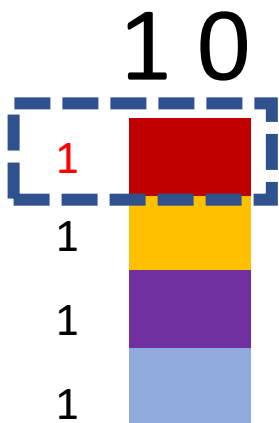
- ❖ Start many instances of the streaming algorithm at the beginning
- ❖ Use an instance of the algorithm but “freeze” the output
- ❖ Each time the next instance has value  $(1 + O(\epsilon))$  more than the “frozen” output, use the next instance and “freeze” its output



- ❖ Example: Number of ones in the stream (2-approximation)

# Robust Algorithms

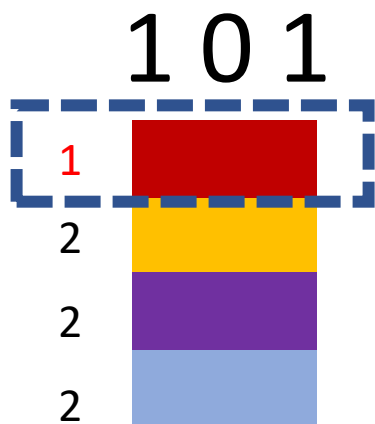
- ❖ Start many instances of the streaming algorithm at the beginning
- ❖ Use an instance of the algorithm but “freeze” the output
- ❖ Each time the next instance has value  $(1 + O(\epsilon))$  more than the “frozen” output, use the next instance and “freeze” its output



- ❖ Example: Number of ones in the stream (2-approximation)

# Robust Algorithms

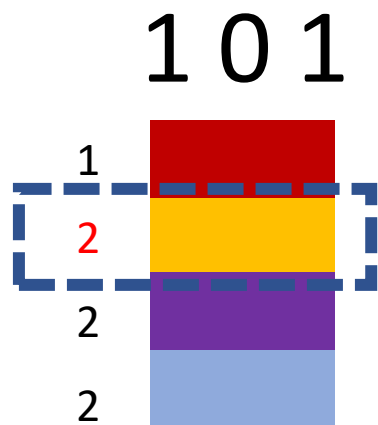
- ❖ Start many instances of the streaming algorithm at the beginning
- ❖ Use an instance of the algorithm but “freeze” the output
- ❖ Each time the next instance has value  $(1 + O(\epsilon))$  more than the “frozen” output, use the next instance and “freeze” its output



- ❖ Example: Number of ones in the stream (2-approximation)

# Robust Algorithms

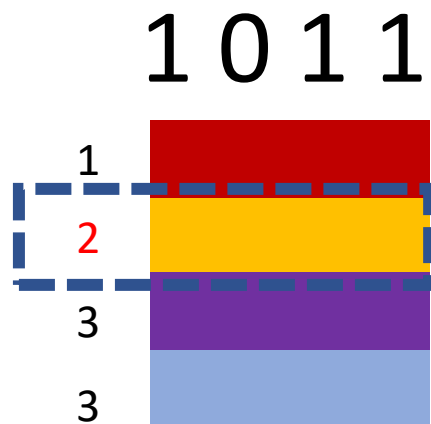
- ❖ Start many instances of the streaming algorithm at the beginning
- ❖ Use an instance of the algorithm but “freeze” the output
- ❖ Each time the next instance has value  $(1 + O(\epsilon))$  more than the “frozen” output, use the next instance and “freeze” its output



- ❖ Example: Number of ones in the stream (2-approximation)

# Robust Algorithms

- ❖ Start many instances of the streaming algorithm at the beginning
- ❖ Use an instance of the algorithm but “freeze” the output
- ❖ Each time the next instance has value  $(1 + O(\epsilon))$  more than the “frozen” output, use the next instance and “freeze” its output

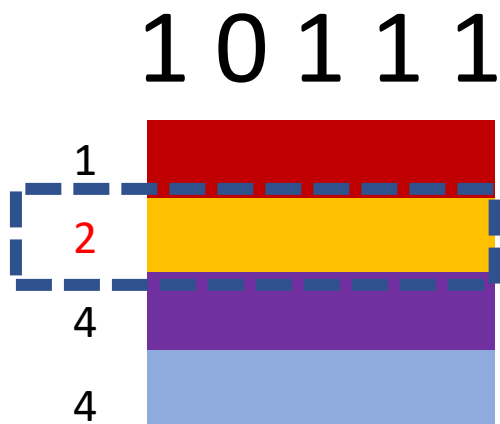


- ❖ Example: Number of ones in the stream (2-approximation)



# Robust Algorithms

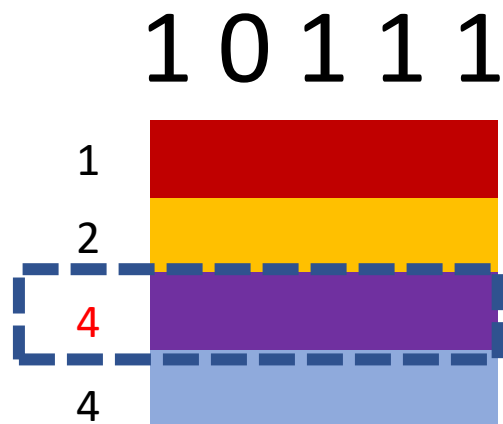
- ❖ Start many instances of the streaming algorithm at the beginning
- ❖ Use an instance of the algorithm but “freeze” the output
- ❖ Each time the next instance has value  $(1 + O(\epsilon))$  more than the “frozen” output, use the next instance and “freeze” its output



- ❖ Example: Number of ones in the stream (2-approximation)

# Robust Algorithms

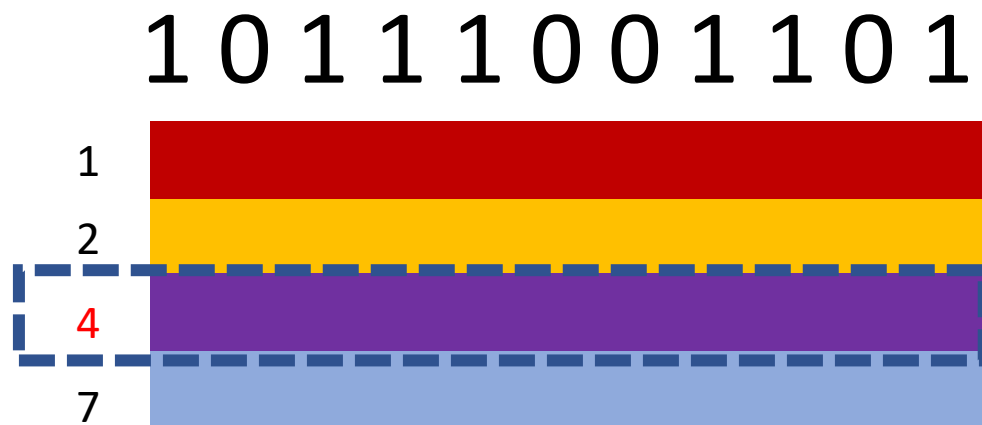
- ❖ Start many instances of the streaming algorithm at the beginning
- ❖ Use an instance of the algorithm but “freeze” the output
- ❖ Each time the next instance has value  $(1 + O(\epsilon))$  more than the “frozen” output, use the next instance and “freeze” its output



- ❖ Example: Number of ones in the stream (2-approximation)

# Robust Algorithms

- ❖ Start many instances of the streaming algorithm at the beginning
- ❖ Use an instance of the algorithm but “freeze” the output
- ❖ Each time the next instance has value  $(1 + O(\epsilon))$  more than the “frozen” output, use the next instance and “freeze” its output



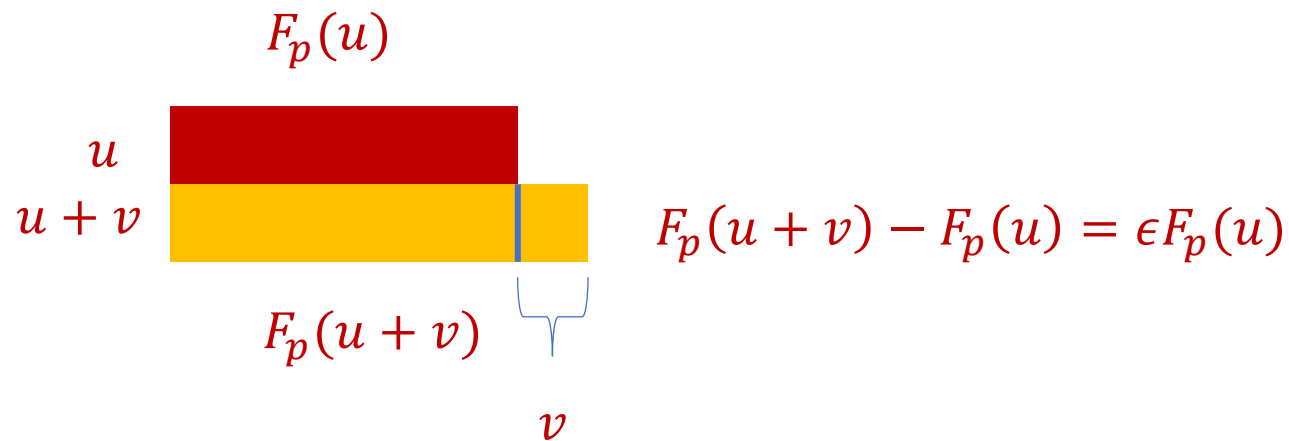
- ❖ Example: Number of ones in the stream (2-approximation)
- ❖ Number of ones stream is at least 4 and at most 8
- ❖ 4 is a good approximation

# Summary

- ❖ Sketch switching for robust algorithms uses  $\frac{1}{\epsilon^2}$  space each time  $F_p$  increases by  $(1 + \epsilon)$  and function increases  $\frac{1}{\epsilon}$  times
- ❖ Smooth histogram for sliding window algorithms uses  $\frac{1}{\epsilon^2}$  space each time  $F_p$  increases by  $(1 + \epsilon)$  and function increases  $\frac{1}{\epsilon}$  times for  $p \in (0,1)$
- ❖ Smooth histogram for sliding window algorithms uses  $\frac{1}{\epsilon^2}$  space each time  $F_p$  increases by  $(1 + \epsilon^p)$  and function increases  $\frac{1}{\epsilon^p}$  times for  $p \in (1,2)$

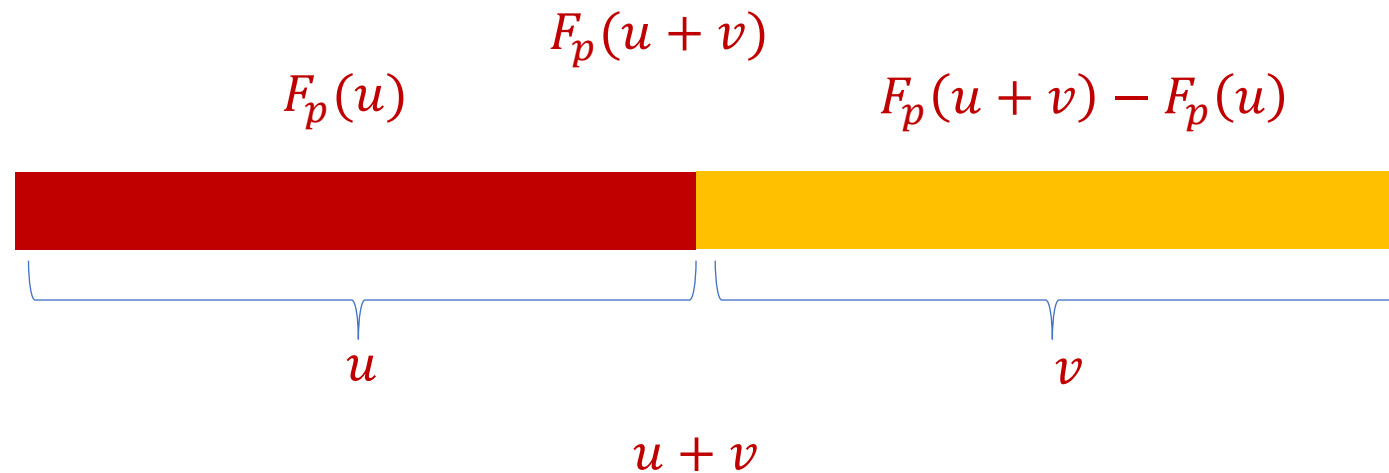
# Intuition

- ❖ Do we really need to pay  $\frac{1}{\epsilon^2}$  space each time  $F_p$  increases by  $(1 + \epsilon)$ ?
- ❖ Only need constant factor approximation to  $\epsilon F_p(u)$
- ❖ Only need constant factor approximation to  $F_p(u + v) - F_p(u)$



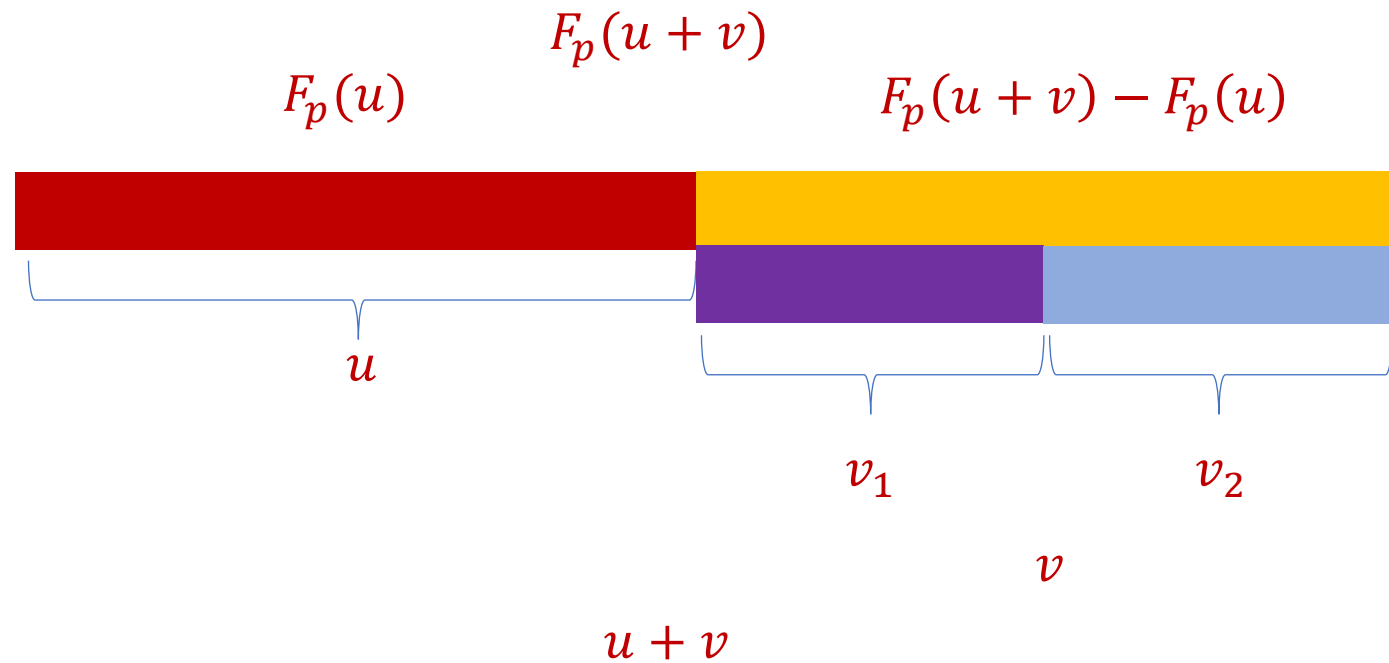
# Sketch Stitching

- ❖ Suppose we want  $F_p(u + v)$
- ❖  $F_p(u + v) = (F_p(u + v) - F_p(u)) + F_p(u)$



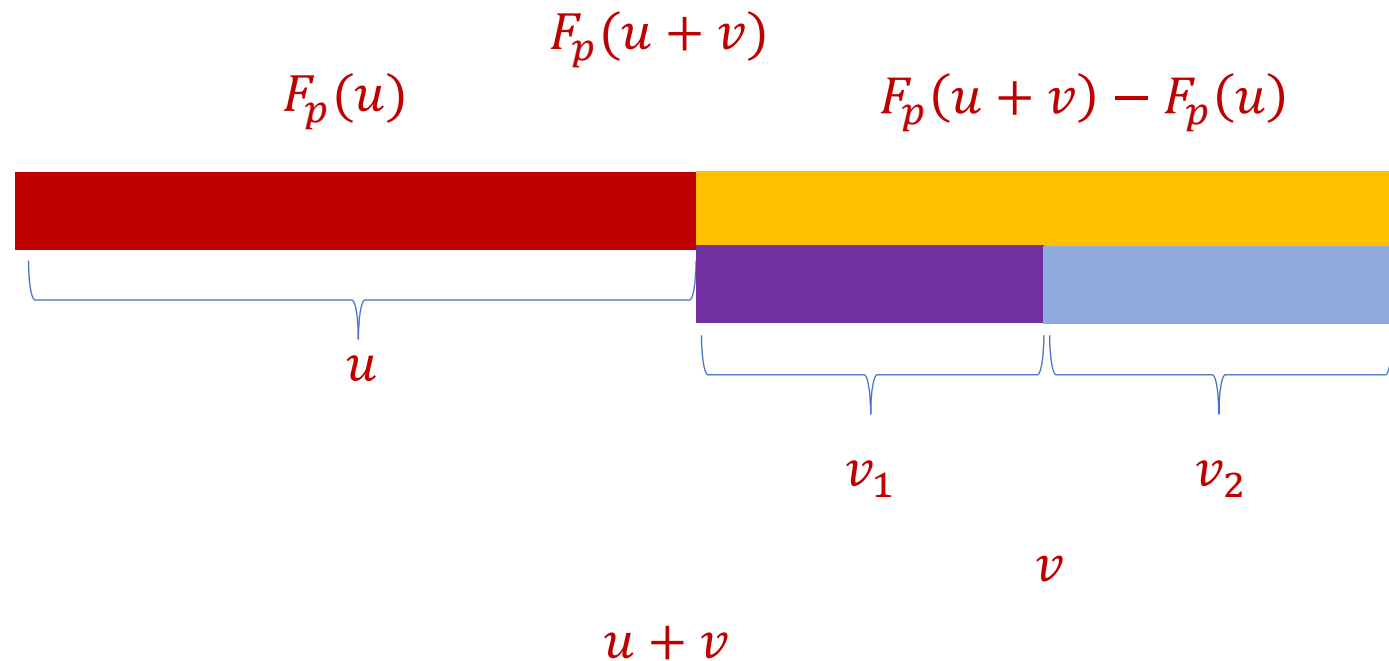
# Sketch Stitching

❖ Suppose we want  $F_p(u + v)$  and  $v = v_1 + v_2 + \dots + v_b$



# Sketch Stitching

- ❖ Suppose we want  $F_p(u + v)$  and  $v = v_1 + v_2 + \dots + v_b$
- ❖  $F_p(u + v) = (F_p(u + v) - F_p(u)) + F_p(u)$





# Sketch Stitching

- ❖ Suppose we want  $F_p(u + v)$  and  $v = v_1 + v_2 + \cdots + v_b$
- ❖  $F_p(u + v) = (F_p(u + v) - F_p(u)) + F_p(u)$
- ❖  $F_p(u + v) = \left( F_p(u + v_1 + \cdots + v_b) - F_p(u + v_1 + \cdots + v_{b-1}) \right) + \left( F_p(u + v_1 + \cdots + v_{b-1}) - F_p(u + v_1 + \cdots + v_{b-2}) \right) + \cdots + \left( F_p(u + v_1) - F_p(u) \right) + F_p(u)$

# Granularity Change

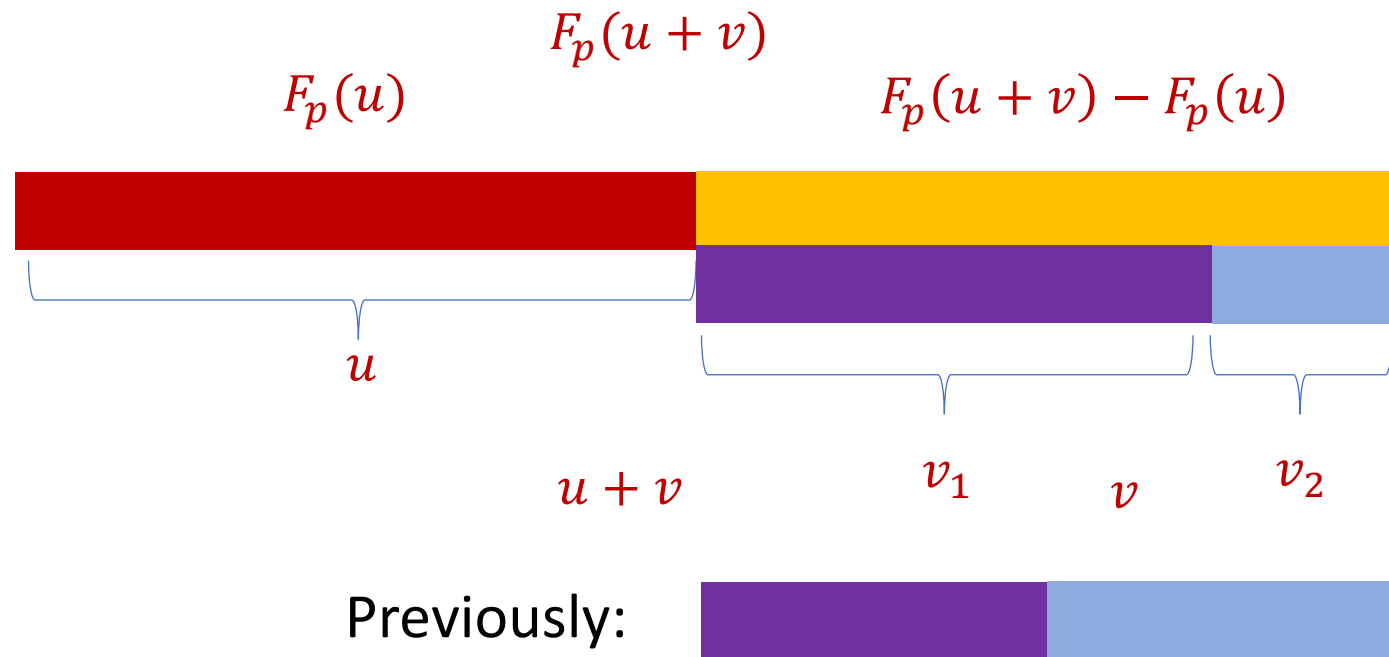
❖ Set each difference to be exponentially decreasing

$$\begin{aligned} \text{❖ } F_p(u + v) = & \left( F_p(u + v_1 + \cdots + v_b) - F_p(u + v_1 + \cdots + v_{b-1}) \right) + \\ & \left( F_p(u + v_1 + \cdots + v_{b-1}) - F_p(u + v_1 + \cdots + v_{b-2}) \right) + \cdots + \\ & \left( F_p(u + v_1) - F_p(u) \right) + F_p(u) \end{aligned}$$

$$\text{❖ } F_p(u + v_1 + \cdots + v_b) - F_p(u + v_1 + \cdots + v_{b-1}) = \frac{1}{2^b} F_p(u)$$

# Granularity Change

$$\blacklozenge F_p(u + v_1 + \dots + v_b) - F_p(u + v_1 + \dots + v_{b-1}) = \frac{1}{2^b} F_p(u)$$

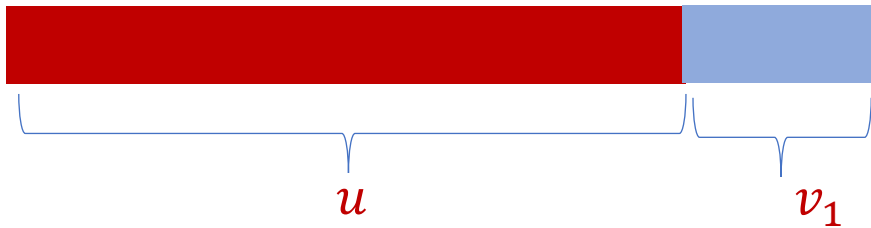


# Granularity Change

- ❖ Set each difference to be exponentially decreasing
- ❖  $F_p(u + v) = \left( F_p(u + v_1 + \dots + v_b) - F_p(u + v_1 + \dots + v_{b-1}) \right) + \left( F_p(u + v_1 + \dots + v_{b-1}) - F_p(u + v_1 + \dots + v_{b-2}) \right) + \dots + \left( F_p(u + v_1) - F_p(u) \right) + F_p(u)$
- ❖  $F_p(u + v_1 + \dots + v_b) - F_p(u + v_1 + \dots + v_{b-1}) = \frac{1}{2^b} F_p(u)$
- ❖ Just need  $2^b \epsilon$ -approximation to  $F_p(u + v_1 + \dots + v_b) - F_p(u + v_1 + \dots + v_{b-1})$
- ❖ Hope is to use space  $\frac{1}{2^{2b} \epsilon^2}$

# Framework

- ❖ Algorithms simultaneously running for each granularity
- ❖ Want space  $\frac{1}{2^{2b}\epsilon^2}$  for granularity  $\frac{1}{2^b} F_p(u)$
- ❖ Need  $2^b$  instances for granularity  $\frac{1}{2^b} F_p(u)$



- ❖ Total space  $\sum \frac{1}{2^b \epsilon^2} = \frac{1}{\epsilon^2}$

# Format

- ❖ Part 1: Background
- ❖ Part 2: Framework
- ❖ Part 3: Difference Estimators

# Questions?



# Difference Estimator

- ❖ If  $F_p(u + v) - F_p(u) = 2^b \epsilon F_p(u)$ , does there exist algorithm that approximates the difference with space  $\frac{1}{2^{2b} \epsilon^2}$ ?



- ❖ **Definition:**  $F_p(u + v) - F_p(u) = \gamma F_p(u)$ , output an estimate to the difference with additive approximation  $\epsilon F_p(u)$



# Difference Estimator

- ❖ **Definition:**  $F_p(u + v) - F_p(u) = \gamma F_p(u)$ , output an estimate to the difference with additive approximation  $\epsilon F_p(u)$
- ❖  $F$  is generally non-linear
- ❖ Ex:  $F_p(u + v) = \frac{1}{\epsilon^4}$ ,  $F_p(u + v) - F_p(u) = 1$
- ❖  $(1 + \epsilon)$  approximations to  $F_p(u + v)$  and  $F_p(u)$  give multiplicative approximation to the difference but use space  $\frac{1}{\epsilon^2}$
- ❖ Constant factor approximations to  $F_p(u + v)$  and  $F_p(u)$  do not give additive approximation  $\epsilon F_p(u)$  to the difference





# Our Results: Difference Estimators

- ❖ Space  $\tilde{O}\left(\frac{\gamma}{\epsilon^2} \log n\right)$  algorithm for  $F_0$
- ❖ Space  $\tilde{O}\left(\frac{\gamma^{2/p}}{\epsilon^2} \log n\right)$  algorithm for  $F_p$  with  $p \in (0, 2]$
- ❖ Space  $\tilde{O}\left(\frac{\gamma}{\epsilon^2} n^{1-2/p}\right)$  algorithm for  $F_p$  with integer  $p > 2$

# $F_2$ Difference Estimator

- ❖ **Definition:**  $F_2(u + v) - F_2(u) = \gamma F_2(u)$ , output an estimate to the difference with additive approximation  $\epsilon F_2(u)$
- ❖  $F_2(u + v) - F_2(u) = \langle u + v, u + v \rangle - \langle u, u \rangle = 2\langle u, v \rangle + \langle v, v \rangle^2$
- ❖ **Inner product property:**  $(1 + \epsilon)$ -approximations to  $\|u\|_2$  and  $\|v\|_2$  gives an  $\epsilon\|u\|_2\|v\|_2$  additive approximation to  $\langle u, v \rangle$
- ❖  $\gamma F_2(u) \geq \langle v, v \rangle^2$  implies  $2\langle u, v \rangle \leq 2\|u\|_2\|v\|_2 \leq 2\sqrt{\gamma} F_2(u)$
- ❖ Just need  $\frac{\epsilon}{\sqrt{\gamma}}$  multiplicative approximation:  $\tilde{O}\left(\frac{\gamma}{\epsilon^2} \log n\right)$  space!

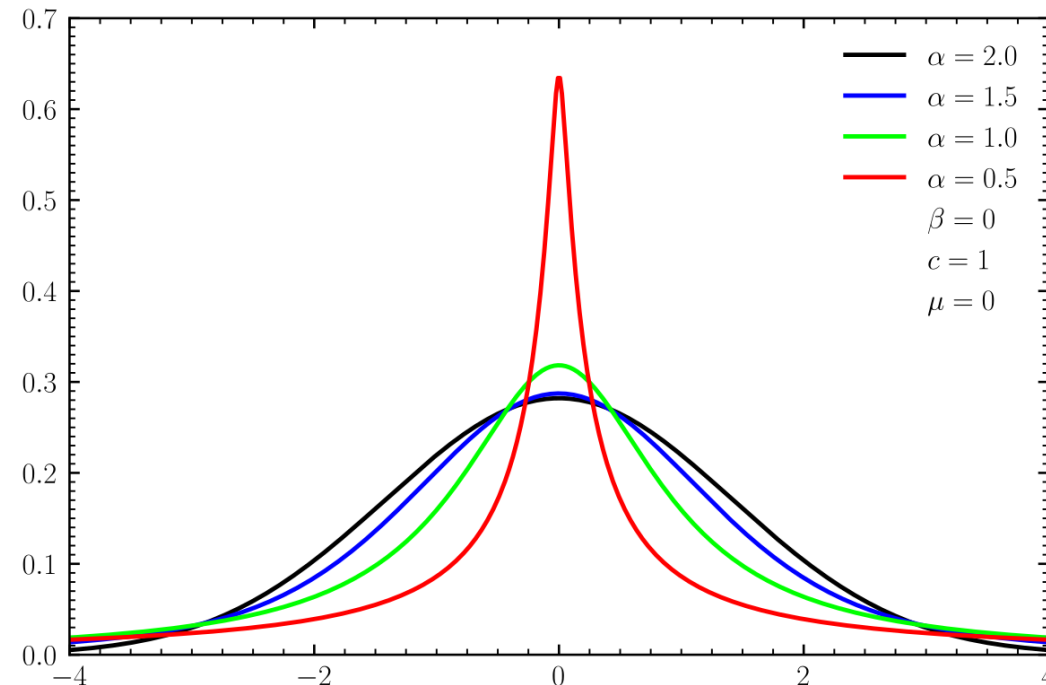
# $F_2$ Difference Estimator

- ❖ **Difference estimator:** Maintain  $\left(1 + \frac{\epsilon}{\sqrt{\gamma}}\right)$ -approximations to  $F_2(u + v)$  and  $F_2(u)$  using AMS sketch
- ❖ For  $F_2(u + v) - F_2(u) = \gamma F_2(u)$ , difference of the outputs is an additive approximation  $\epsilon F_2(u)$  to  $F_2(u + v) - F_2(u)$

# Challenges for Difference Estimators

- ❖  $F_p$  difference estimator: Use  $p$ -stable random variables for  $p \leq 2$ ?
- ❖ How to use approach of [BlasiokDingNelson17]?
- ❖  $\langle Z_p, f \rangle$  where  $Z_p$  has entries drawn from  $p$ -stable distribution

$$p(x) \sim \frac{1}{1 + |x|^{p+1}}$$



# Challenges for Difference Estimators

- ❖  $Z = \text{median}\langle Z_p, f \rangle$
- ❖ How to analyze median of each estimate of  $F_p(u + v) - F_p(u)$ ?
- ❖ Use Li's geometric mean algorithm [Li08]
- ❖ Take the geometric mean of 3 inner products  $\langle Z_p, f \rangle$
- ❖ Take the average of  $O\left(\frac{1}{\epsilon^2}\right)$  geometric means

# Challenges for Difference Estimators

- ❖ **Difference estimator:** Maintain  $\left(1 + \frac{\epsilon}{\gamma^{1/p}}\right)$ -approximations to  $F_p(u + v)$  and  $F_p(u)$  using Li's geometric mean estimator
- ❖  $A(u + v) - A(u) \sim \langle p_1, v \rangle^{p/3} \langle p_2, v \rangle^{p/3} \langle p_3, v \rangle^{p/3} +$   
 $\langle p_1, u \rangle^{p/3} \langle p_2, v \rangle^{p/3} \langle p_1, v \rangle^{p/3} +$   
 $\langle p_1, u \rangle^{p/3} \langle p_2, u \rangle^{p/3} \langle p_1, v \rangle^{p/3} + \dots$
- ❖ Each summand has  $\langle p_1, v \rangle^{p/3}$  term, which has much smaller variance

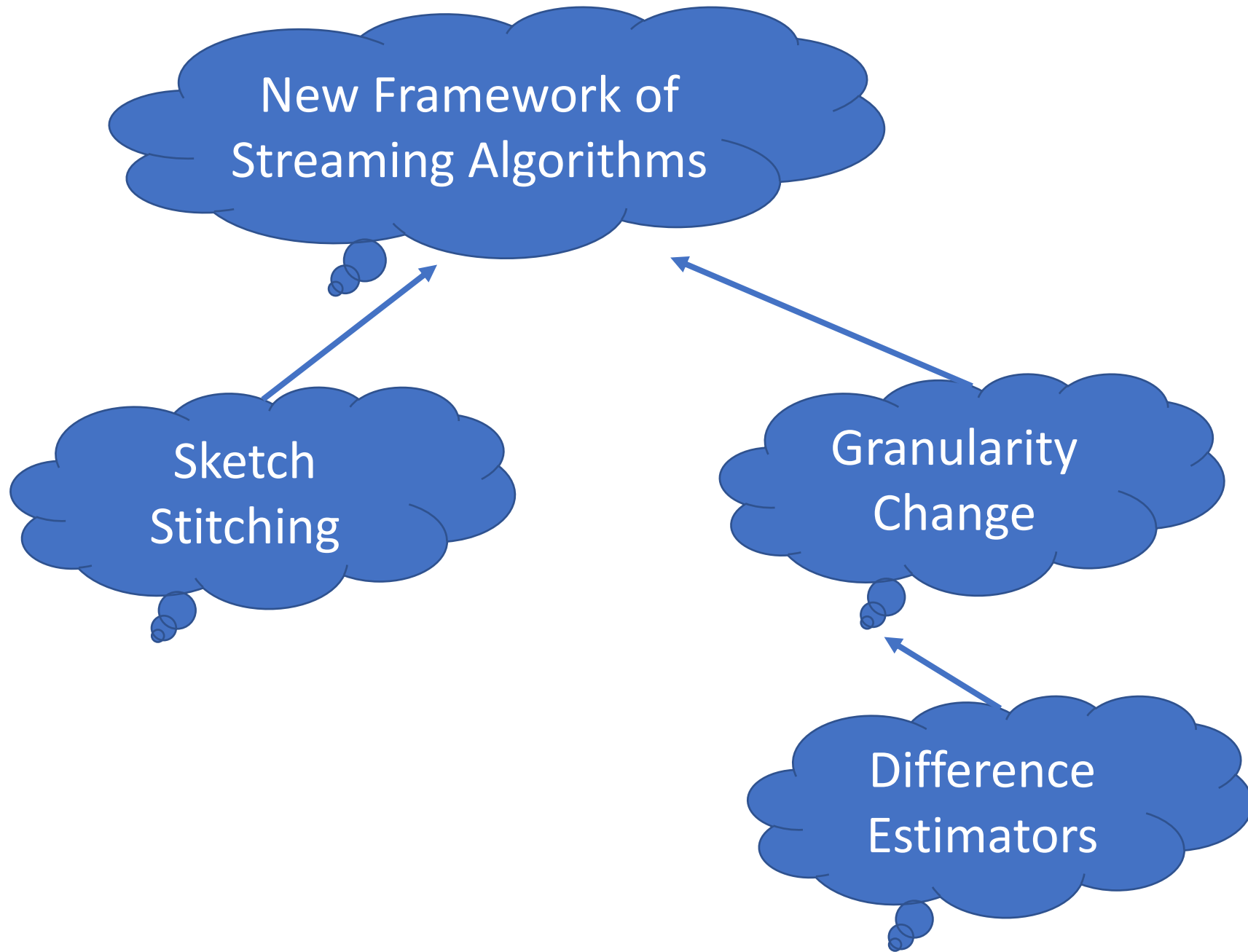
# Challenges for Difference Estimators

- ❖  $F_p$  difference estimator: Generalization of inner products for  $p > 2$ ?
- ❖ Variance can be much larger!
- ❖ Use heavy-hitter algorithm to explicitly track “heavy” elements
- ❖ Use  $L_2$  sampling algorithm with  $n^{1-2/p}$  buckets to sample “light” elements

# Challenges for Difference Estimators

- ❖ Use chaining argument to remove  $\log n$  factor in difference estimator
- ❖ Avoids typical Chernoff + union bound argument by considering the expected supremum of a process, “strong tracking”
- ❖ Use suffix argument to remove  $\log n$  factor in framework
- ❖ Adaptation to sliding window model





# WALDO 2021: Save the Date!



- ❖ Workshop on Algorithms for Large Data Online
- ❖ Streaming, sketching, linear algebra, property testing, learning
- ❖ Virtual format, August 23-25



