On Efficient Sketching Algorithms

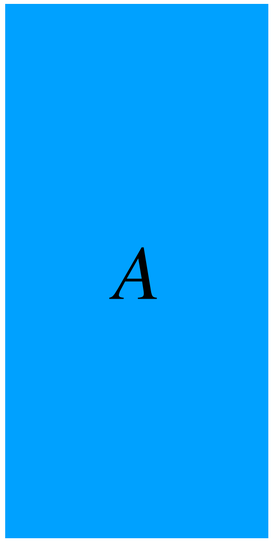
Praneeth Kacham

Thesis Committee

David Woodruff (chair)
Pravesh Kothari
Richard Peng
Rasmus Pagh (University of Copenhagen)

Introduction

- Conventional algorithms are bad at handling large datasets
  - Do not utilize sparsity
  - Assume arbitrary efficient access
- How to construct algorithms that work on large datasets?



Three Settings

- **Classic Setting**
  - Inputs fit in memory
  - Algorithms that run as fast as possible
- **Distributed Setting**
  - Inputs are distributed over multiple servers
  - Coordinator model and arbitrary topologies
  - Protocols that use a low amount of communication

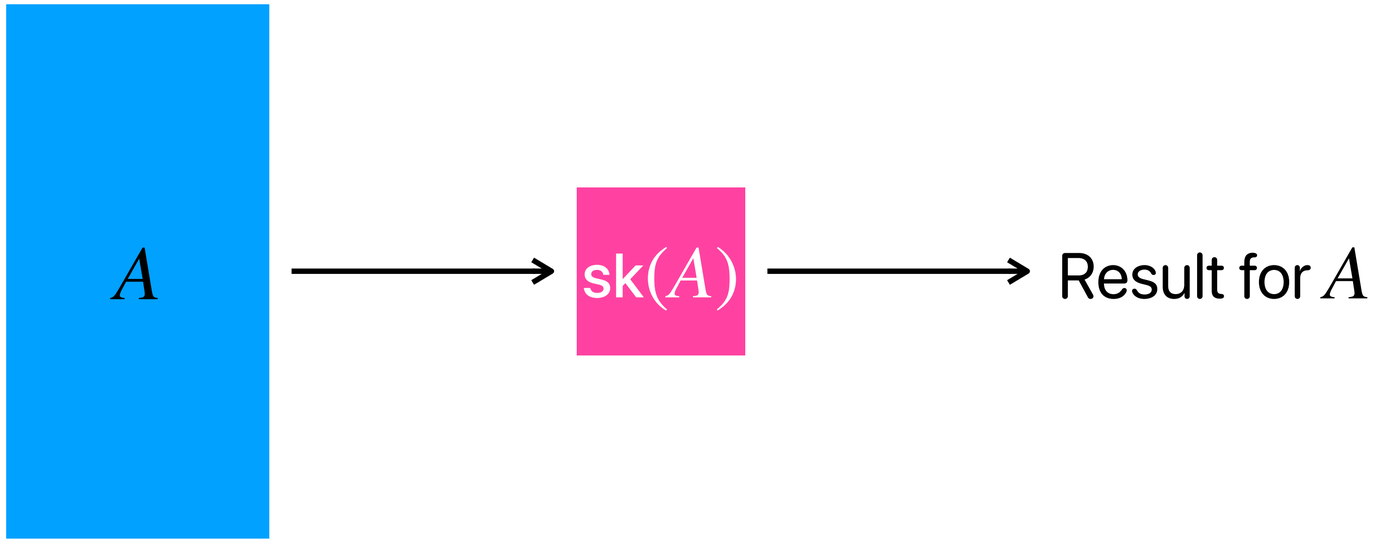
Three Settings

- **Streaming Setting**
    - Inputs are long streams defining large inputs
    - Additive updates or row arrival models
    - Small space algorithms that update their state quickly

A mix of streaming and distributed settings is also studied e.g., Distributed Functional Monitoring

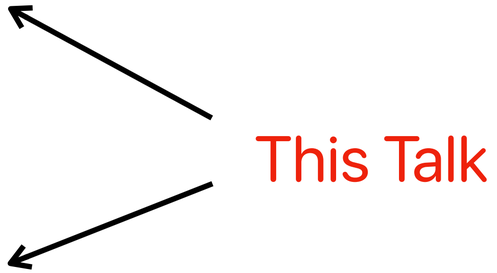
Sketching

- Techniques developed over last 20 years to handle large datasets
- Shrink the large dataset into a small dataset
- Efficiently perform computation on the small dataset

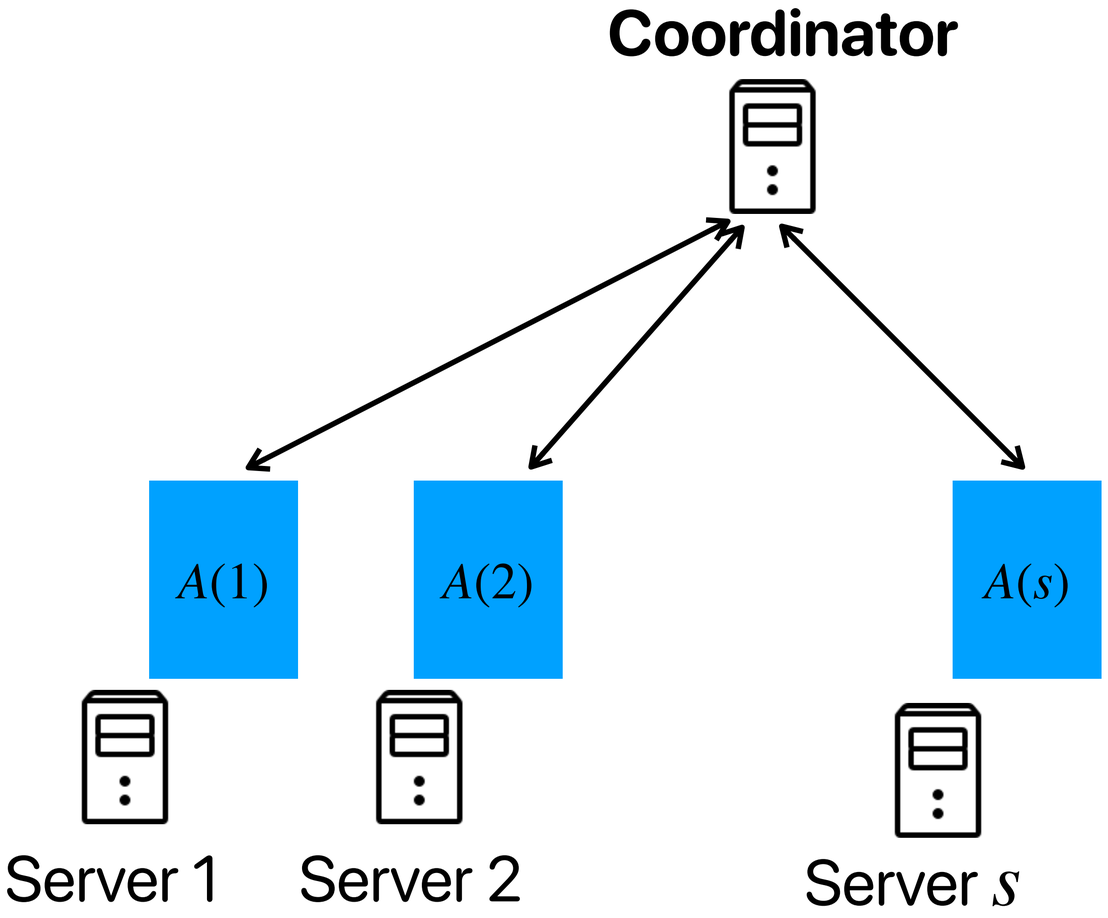


How is Sketching Useful?

- **Classic Setting**
  - Compute a sketch of the data quickly
  - Run time-intensive algorithms on much smaller sketch instead of on the full dataset
- **Distributed Setting**
  - Compute a sketch of the local dataset and send it to the coordinator
  - Smaller the sketch, lower the communication required
- **Streaming Setting**
  - If the sketch construction is efficiently **updateable**, modify the sketch on each update to the underlying dataset



Distributed Setting



$A = A(1) + \cdots + A(s)$

- General point-to-point communication with twice the communication + $\log_2 s$ bits
- We also study arbitrary graph topologies in the thesis

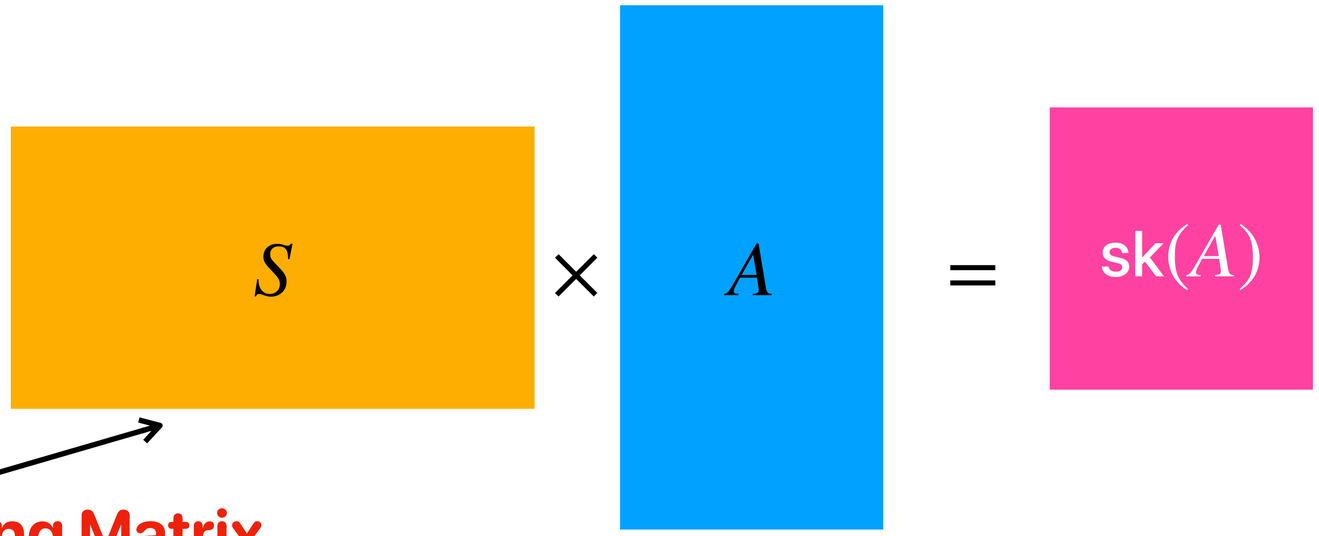
Turnstile Streaming

- Initialize $x \leftarrow 0 \in \mathbb{R}^n$
- On update (i, Δ) :
  - Set $x_i \leftarrow x_i + \Delta$
- Answer queries about x using **small space**
  - $\max_i |x_i| = \|x\|_\infty$
  - $\sum_i |x_i|^p$ (F_p moments)
    - Useful to characterize the distribution
    - $p \in (0,2)$ to approximate the entropy of the distirbution

Updates: (i_1, Δ_1) (i_2, Δ_2) … (i_m, Δ_m)

A Major Technique: Linear Sketching

- Apply a randomized linear transform
    - CountSketch [Clarkson and Woodruff '13], OSNAP [Nelson and Nguyen '13], SRHT [Tropp '10, Ailon and Chazelle '06], …



Sketching Matrix

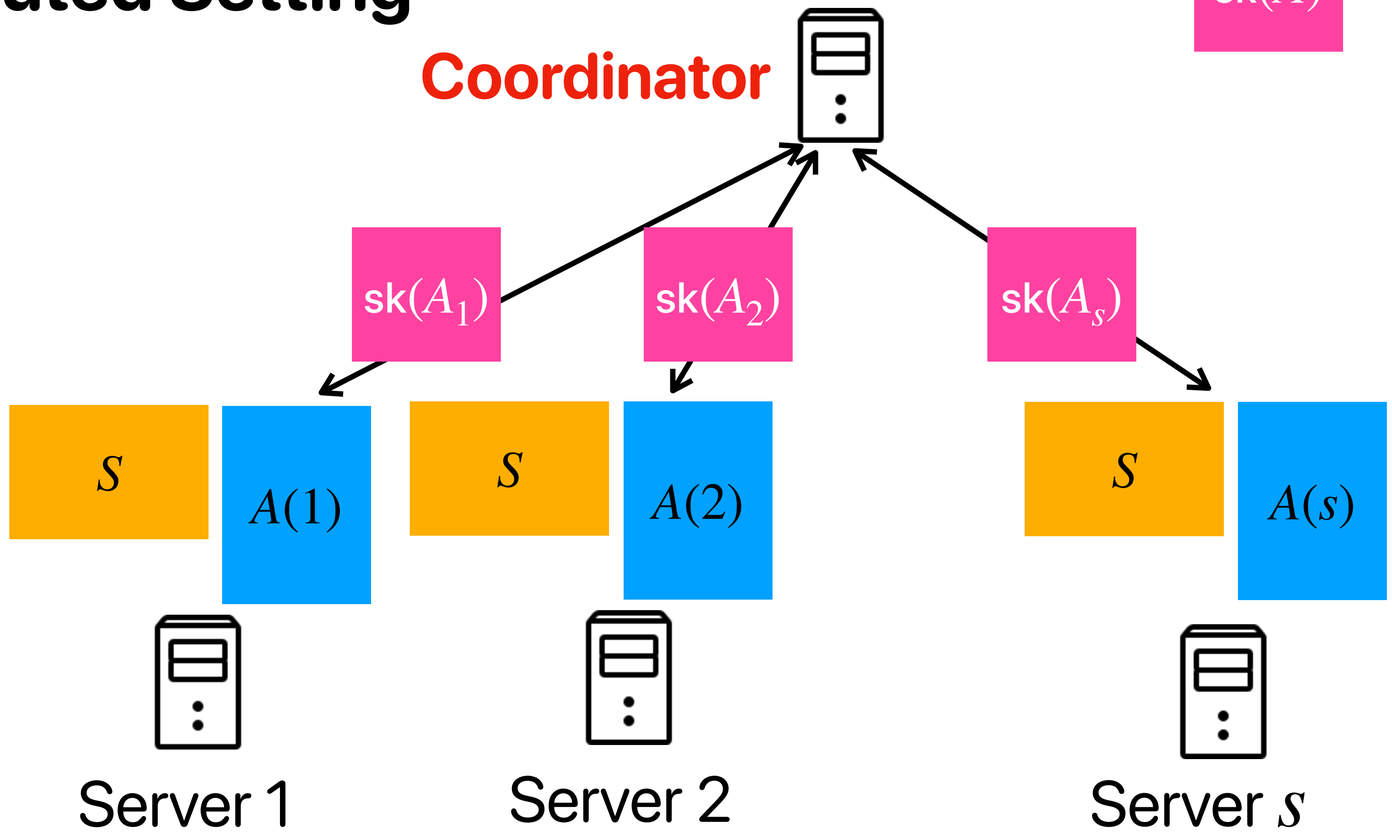
- S consolidates information into the "sketch"

How is Sketching Useful?

Distributed Setting

$\mathsf{sk}(A) = \mathsf{sk}(A_1) + \mathsf{sk}(A_2) + \cdots + \mathsf{sk}(A_s)$

$A = A(1) + \cdots + A(s)$

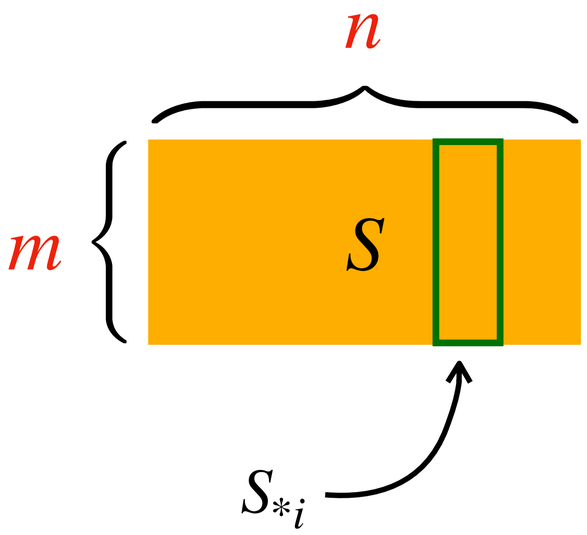


(Can typically be handled with low communication using pseudorandom generators)

How is Sketching Useful?

Turnstile Streaming

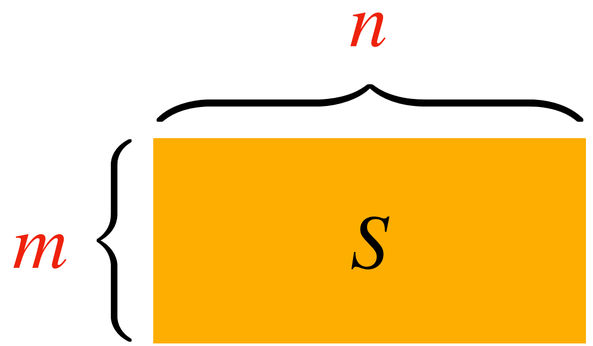
- **Implicitly** initialize a sketching matrix S
  - Maintain $S \cdot x$ in the stream
- Initialize $\text{sk}(x) \leftarrow 0$
- When $x[i] \leftarrow x[i] + \Delta$
  - Retrieve the i-th column S_{*i}
  - Update $\text{sk}(x) \leftarrow \text{sk}(x) + (S_{*i}) \cdot \Delta$
- At all times: $\text{sk}(x) = S \cdot x$
- Extract "information" about x from $\text{sk}(x)$ at the end



$$x[1]\, S_{*1} + \cdots + x[n]\, S_{*n}$$

How is Sketching Useful?

Turnstile Streaming



- S can be stored using small space and m small ⇒ Small space streaming algorithms!
- S_{*i} can be retrieved quickly ⇒ Fast **update** times
    - We will discuss how to obtain space-optimal algorithms with fast update times

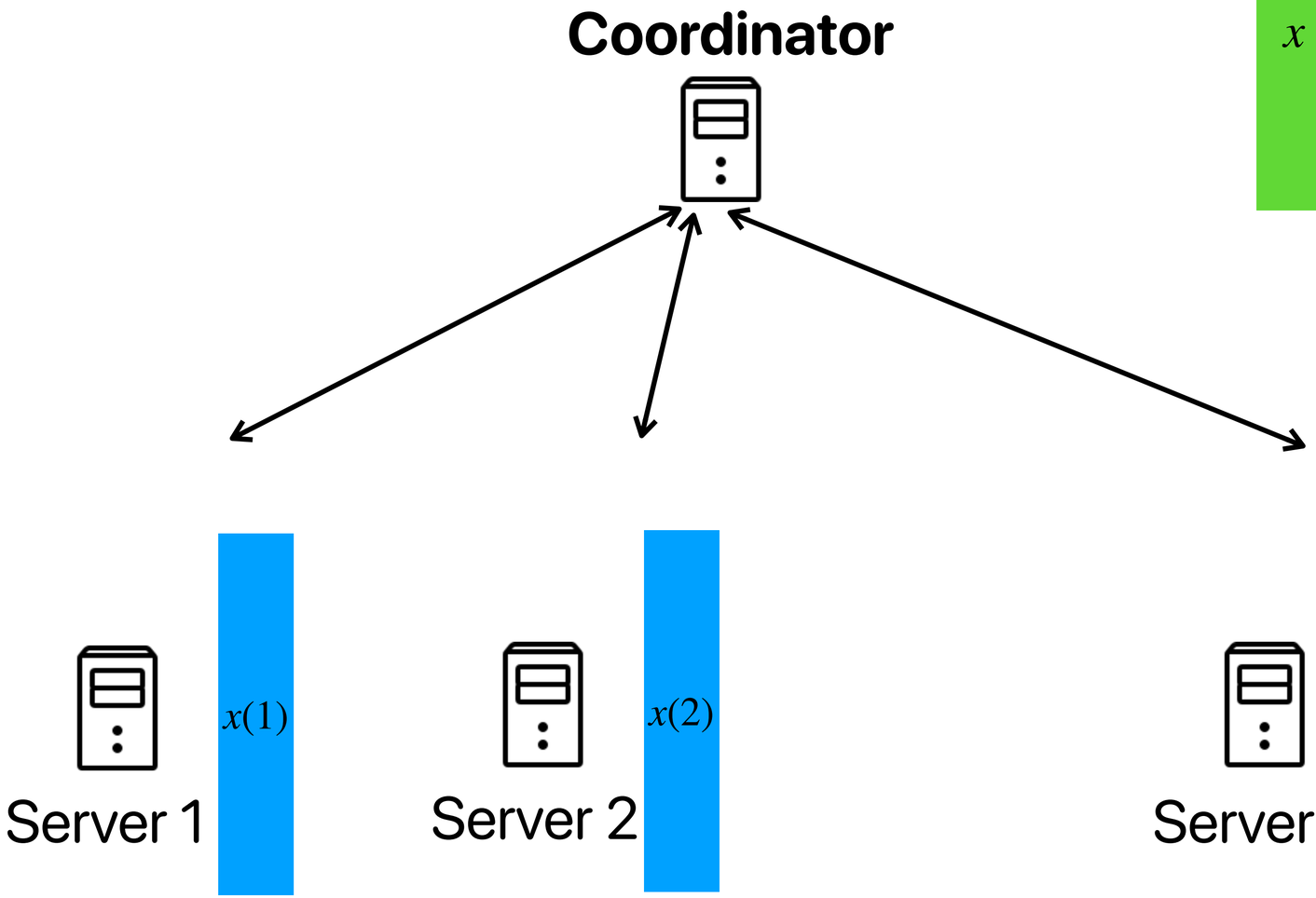
Complexity Measures

- Time complexity is not the only thing!
- Broadly, we care about the following:
    - Time Complexity : Total time, Update time, ...
    - Space Complexity
    - Communication Complexity
    - Randomness Complexity

Optimal Communication Bounds for Classic Functions in the Coordinator Model and Beyond

with Hossein Esfandiari, Vahab Mirrokni, David Woodruff and Peilin Zhong [STOC'24]

Moment Estimation



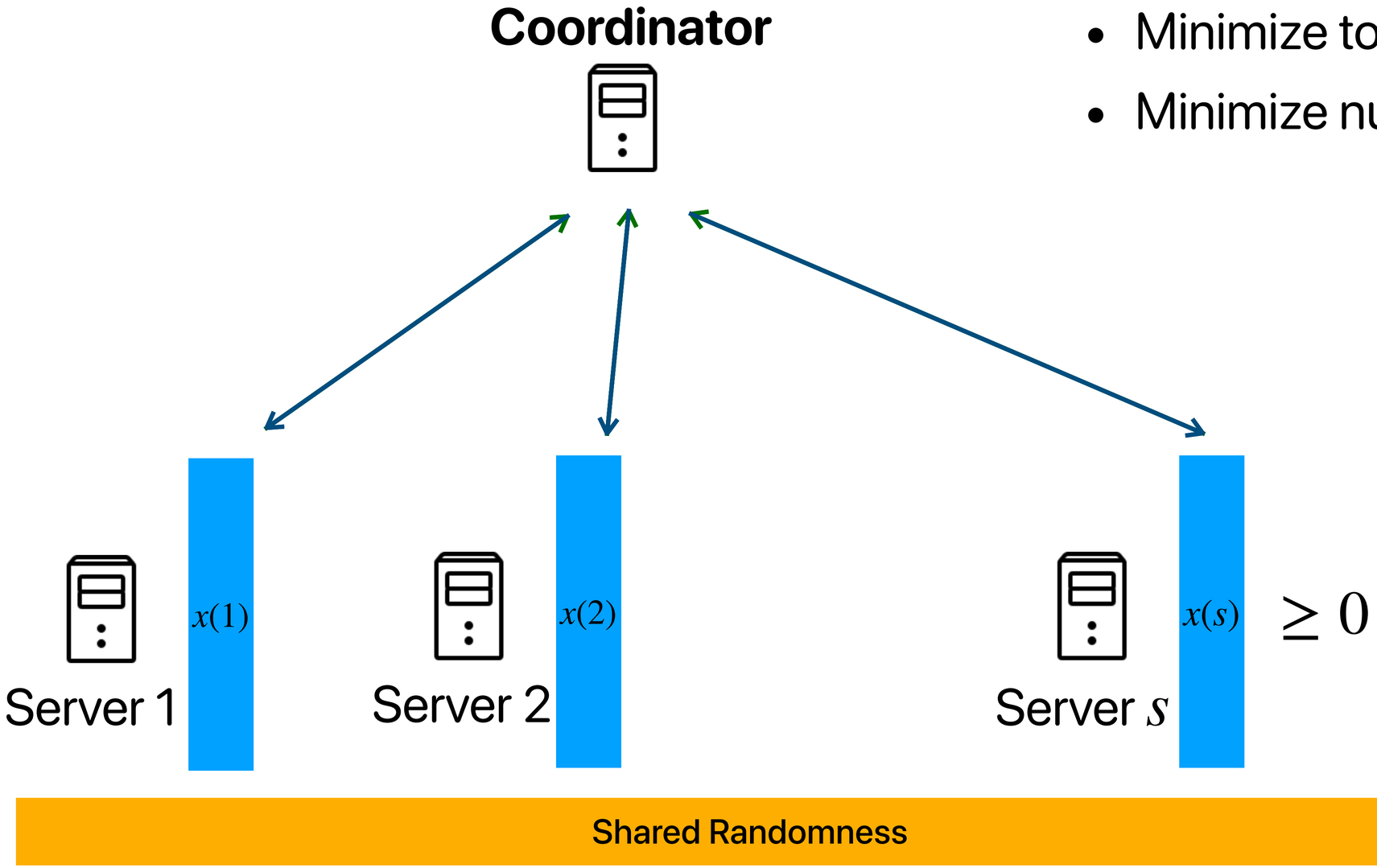
$$x = x(1) + x(2) + \cdots + x(s)$$

Estimate $\sum_i x_i^p$

More generally, $\sum_i f(x_i)$

≥ 0

Communication Model



- Minimize total communication
- Minimize number of rounds

Previous Work

Algorithms:

- Cormode, Muthukrishnan and Yi '11 : $\tilde{O}(n^{1-2/p} \cdot \text{poly}(s/\varepsilon))$ bits
- Woodruff and Zhang '12 : $\tilde{O}(s^{p-1}/\varepsilon^{\Theta(p)} \cdot \text{poly}(\log n))$
- Kannan, Vempala and Woodruff '14 : $\tilde{O}(s^p/\varepsilon^2 \cdot \text{poly}(\log n))$
    - For general functions $O(s^2 \cdot c_{f,s}/\varepsilon^2 \cdot \text{poly}(\log n))$

Distributed functional monitoring

Lower Bounds:

- Woodruff and Zhang '12 : $\Omega(s^{p-1}/\varepsilon^2)$ (s-BTX problem)
- Kannan, Vempala and Woodruff '14 : $\Omega(c_{f,s}/\varepsilon)$ (s-Player Promise Set-Disjointness)

The Parameter $c_{f,s}$

$$f(y_1 + \cdots + y_s) \leq c_{f,s}(f(y_1) + \cdots + f(y_s)) \text{ for all } y_i$$

- Individual function values clue in about the function value of the sum
- As $c_{f,s}$ grows, we expect the protocols to require more communication
    - The lowerbound $\Omega(c_{f,s} / \varepsilon)$ shows it is indeed the case

A New Parameter and Our Result

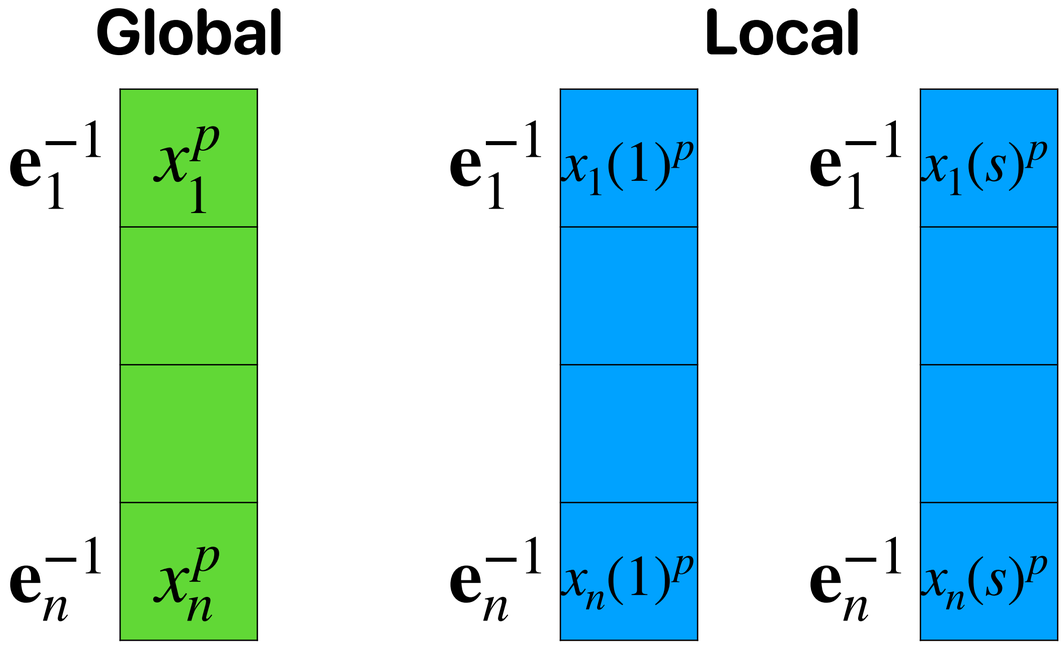
$$f(y_1 + \cdots + y_s) \leq \frac{c_f[s]}{s}\left(\sqrt{f(y_1)} + \cdots + \sqrt{f(y_s)}\right)^2$$

- $c_{f,s} \leq c_f[s] \leq s \cdot c_{f,s}$

$$f(y_1 + \cdots + y_s) \leq c_{f,s}(f(y_1) + \cdots + f(y_s)) \text{ for all } y_i$$

- For $f(y) = y^p$, $c_f[s] = s^{p-1} = c_{f,s}$
- **Theorem:** Given a super-additive function f that is "approximately invertible", there is a **two round** protocol using $O(c_f[s] \cdot \text{polylog}(n)/\varepsilon^2)$ bits of communication to approximate $\sum_i f(x_i)$ up to $1 \pm \varepsilon$
- $\Omega(c_f[s]/\varepsilon^2)$ **lower bound** for a restricted class of functions
  - Suggests $c_f[s]$ captures the complexity better

Key Observations



- $\max_i \mathbf{e}_i^{-1}\lambda_i \equiv \mathbf{e}^{-1}(\sum_i \lambda_i)$
  - Define $\lambda_i = x_i^p = (\sum_{j\in[s]} x_i(j))^p$
- median$(\mathbf{e}^{-1} \cdot \sum_i x_i^p) = \sum_i x_i^p/(\ln 2)$
- Can we compute $i^* = \text{argmax}_i\, \mathbf{e}_i^{-1} \cdot x_i^p$ in one round?
  - Can then compute $\mathbf{e}_{i^*}^{-1} x_{i^*}^p = \mathbf{e}_{i^*}^{-1} \left(\sum_j x_{i^*}(j) \right)^p$ in the second round
- Useful property:
  - $\sum_i \mathbf{e}_i^{-1}\lambda_i \le O(\log^2 n) \max_i \mathbf{e}_i^{-1}\lambda_i$ -- the largest value is significant

High Level Ideas

- All servers sample the **same** exponential random variables $\mathbf{e}_1, \ldots, \mathbf{e}_n$
  - Can be derandomized using pseudorandom generators
- We want to find $i^* = \text{argmax}_i \, \mathbf{e}_i^{-1} (\sum_j x_i(j))^p$
- Server j computes the vector $(\mathbf{e}_1^{-1} x_1(j)^p, \ldots, \mathbf{e}_n^{-1} x_n(j)^p)$

$$i \propto \mathbf{e}_i^{-1} x_i(j)^p$$

- Then server j samples $M = O(s^{p-2} \cdot \log^3 n)$ coordinates independently
- Send all the sampled coordinates to the central server -- Does it receive i^*?

Receiving the Top Coordinate

$$\Pr[\text{Not receiving } i^*] = \prod_j \left(1 - \frac{\mathbf{e}_{i^*}^{-1} x_{i^*}(j)^p}{\sum_i \mathbf{e}_i^{-1} x_i(j)^p}\right)^{O(s^{p-2}\cdot \log^3 n)}$$

$$\leq \exp\left(-s^{p-2}\log^3 n \sum_j \frac{\mathbf{e}_{i^*}^{-1} x_{i^*}(j)^p}{\sum_i \mathbf{e}_i^{-1} x_i(j)^p}\right) \qquad (1 - x \leq \exp(-x))$$

$$\leq \exp\left(-s^{p-2}\log^3 n \cdot \frac{\mathbf{e}_{i^*}^{-1}(\sum_j x_{i^*}(j)^{p/2})^2}{\sum_j \sum_i \mathbf{e}_i^{-1} x_i(j)^p}\right) \qquad \left(\sum_i \frac{a_i}{b_i} \geq \frac{(\sum_i \sqrt{a_i})^2}{\sum_i b_i}\right)$$

Receiving the Top Coordinate (Contd.)

$$\leq \exp\left(-s^{p-2}\log^3 n \cdot \frac{\mathbf{e}_{i*}^{-1}(\sum_j x_{i*}(j)^{p/2})^2}{\sum_j \sum_i \mathbf{e}_i^{-1} x_i(j)^p}\right)$$

$$\left(\sum_j x_{i*}(j)^{p/2}\right)^2 \geq \frac{1}{s^{p-2}}\left(\sum_j x_{i*}(j)\right)^p = \frac{1}{s^{p-2}}\left(x_{i*}\right)^p \qquad (c_f[s] = s^{p-1})$$

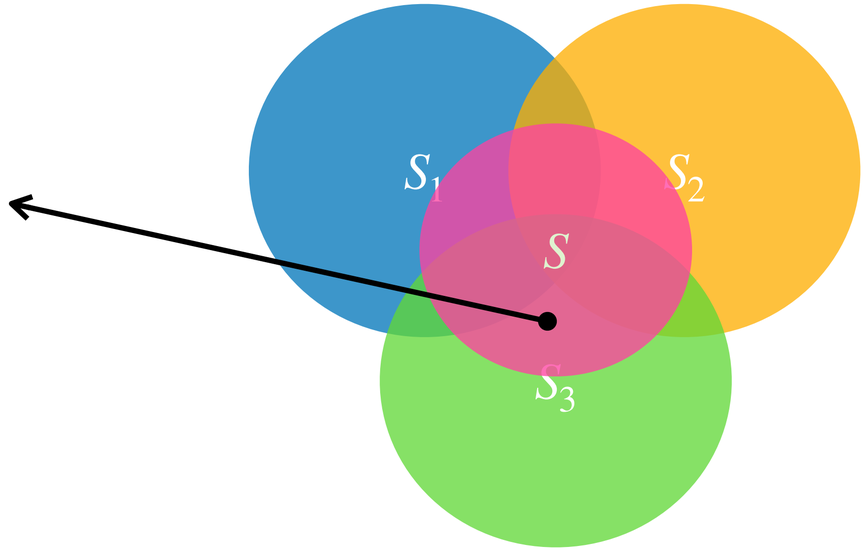
$$\sum_j \sum_i \mathbf{e}_i^{-1} x_i(j)^p \leq \sum_i \mathbf{e}_i^{-1} x_i^p \leq (C\log^2 n)\cdot \mathbf{e}_{i*}^{-1} x_{i*}^p$$

Super-additivity of $f(y) = y^p$

largeness of $\mathbf{e}_{i*}^{-1} x_{i*}^p$

Pr[Not receiving i^*] $\leq 1/\text{poly}(n)$

Computing i^*



- The argument shows that the central server receives i^*
- Can send all $O(s^{p-1} \cdot \log^3 n)$ coordinates to all servers and ask for $x_i(j)$
  - Requires a total of $O(s^p \cdot \log^3 n)$ communication
  - Coordinator needs to find a small set S such that $i^* \in S$
  - We show such S with $|S| \leq$ polylog(n) can be computed by computing approximations to $\mathbf{e}_i^{-1} x_i^p$ for all i by using the sampled coordinates and their values at the sampled servers
    - Critically uses the properties that individual contribution of i^* is quite large and that $\mathbf{e}_{i^*}^{-1} x_{i^*}^p$ is significant fraction of $\sum_i \mathbf{e}_i^{-1} x_i^k$
  - Ask the servers for $x_i(j)$ for only $i \in S$ -- $O(s \cdot$ polylog$(n))$ communication

Extending to general functions f

- Requirements for f
  - Super additivity : $f(x) + f(y) \leq f(x + y)$
  - A multiplicative approximation for $f(x)$ can be used to obtain a multiplicative approximation for f
- The procedure extends and gives a protocol with $O(c_f[s] \cdot \text{polylog}(n)/\varepsilon^2)$ communication

Results

$$f(y_1 + \cdots + y_s) \leq \frac{c_f[s]}{s}\left(\sqrt{f(y_1)} + \cdots + \sqrt{f(y_s)}\right)^2$$

- A protocol using a $O(c_f[s] \cdot \text{polylog}(n)/\varepsilon^2)$ bits of communication
  - For $p \geq 2$, protocol using $O(s^{p-1} \cdot \text{polylog}(n)/\varepsilon^2)$ bits -- optimal up to polylog factors
- The $\Omega(s^{p-1}/\varepsilon^2)$ lower bound can be extended to general functions to show $\Omega(c_f[s]/\varepsilon^2)$ lower bound
  - Requires that $c_f[s]$ is realized for $y_1 = y_2 = \cdots = y_s$

Fast and Space Optimal Streaming Algorithms

with Mikkel Thorup, Rasmus Pagh and David Woodruff [FOCS '23]

Turnstile Streaming

- Initialize $x \leftarrow 0 \in \mathbb{R}^n$
- On update (i, Δ) :
  - Set $x_i \leftarrow x_i + \Delta$
- Answer queries about x using **small space**
  - $\max_i |x_i|$
  - $\sum_i |x_i|^p$ (F_p moments)

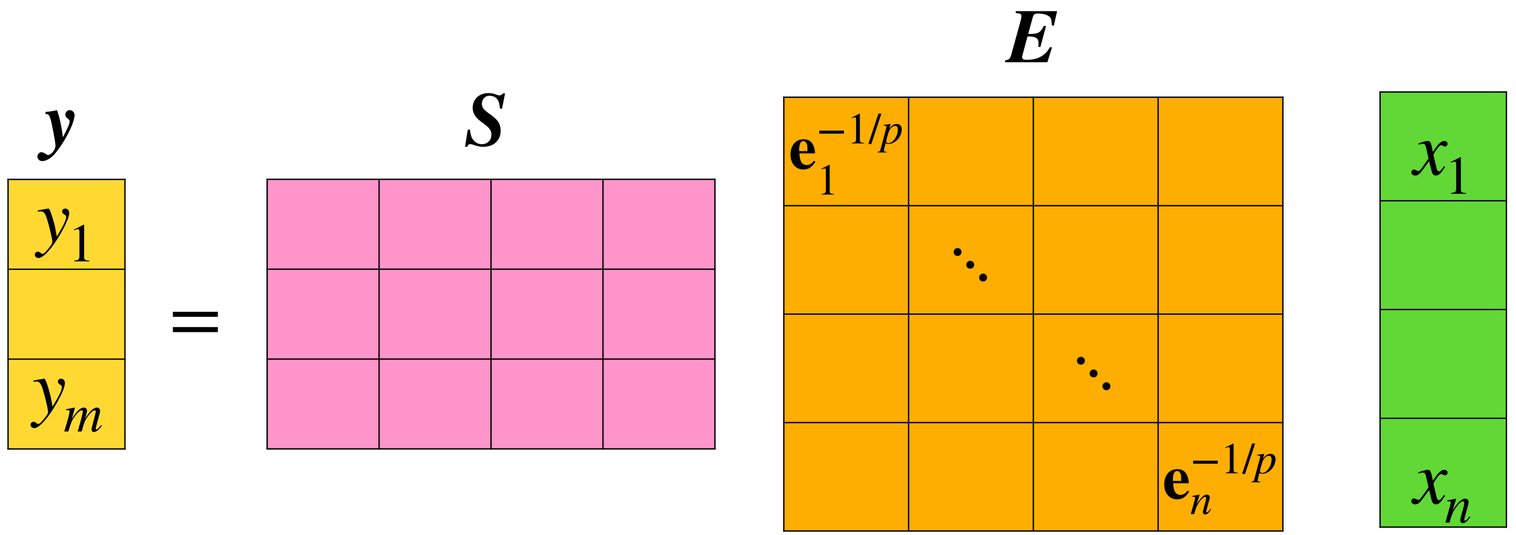
Updates: (i_1, Δ_1) (i_2, Δ_2) … (i_m, Δ_m)

Our Results

- Can we obtain space optimal streaming algorithms with fast update times?
  - **Yes! For many problems.**
- **Theorem**: For $p > 2$, there is an algorithm using **optimal** $\tilde{O}(n^{1-2/p})$ bits of space and an **update time of** $O(1)$ to approximate $F_p(x)$ up to constant factors
  - Improves on poly$(\log n)$ update time of earlier works such as [Andoni, Krauthgamer, Onak '10]

Andoni's Algorithm

- Max stability $\Longrightarrow \max(\mathbf{e}_1^{-1/p}\,|x_1|, \ldots, \mathbf{e}_n^{-1/p}\,|x_n|) \equiv \mathbf{e}^{-1/p} F_p(x)^{1/p}$



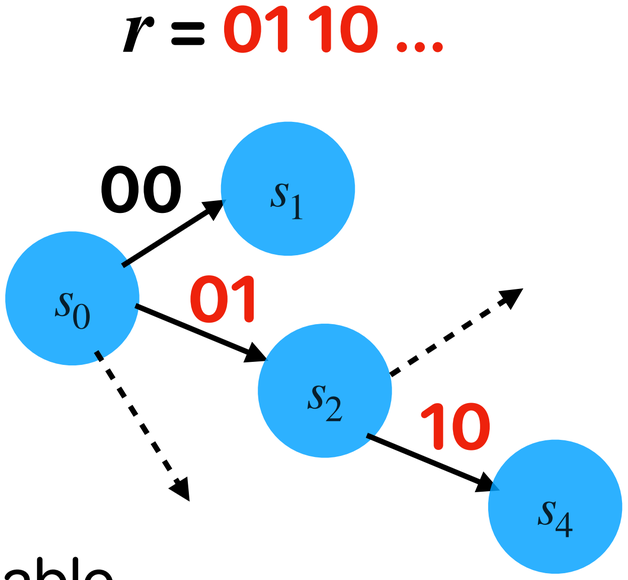
$$\|\boldsymbol{E}x\|_\infty \approx F_p(x)^{1/p}$$

If $m = \Theta(n^{1-2/p} \log n)$, then $\|\boldsymbol{SE}x\|_\infty \approx F_p(x)^{1/p}$

Implementing in a Stream

- S has a concise description using $O(\log n)$-wise independent hash functions
- We need a way to define the matrix E that preserves the properties we want
  - We cannot use independent $\mathbf{e}_1, \ldots, \mathbf{e}_n$ -- $\Omega(n)$ space is required
  - **Idea:** Use a pseudorandom string to generate $\mathbf{e}_1, \ldots, \mathbf{e}_n$
  - Andoni uses the PRG of Nisan and Zuckerman
    - Slow to retrieve an arbitrary $\mathbf{e}_i$

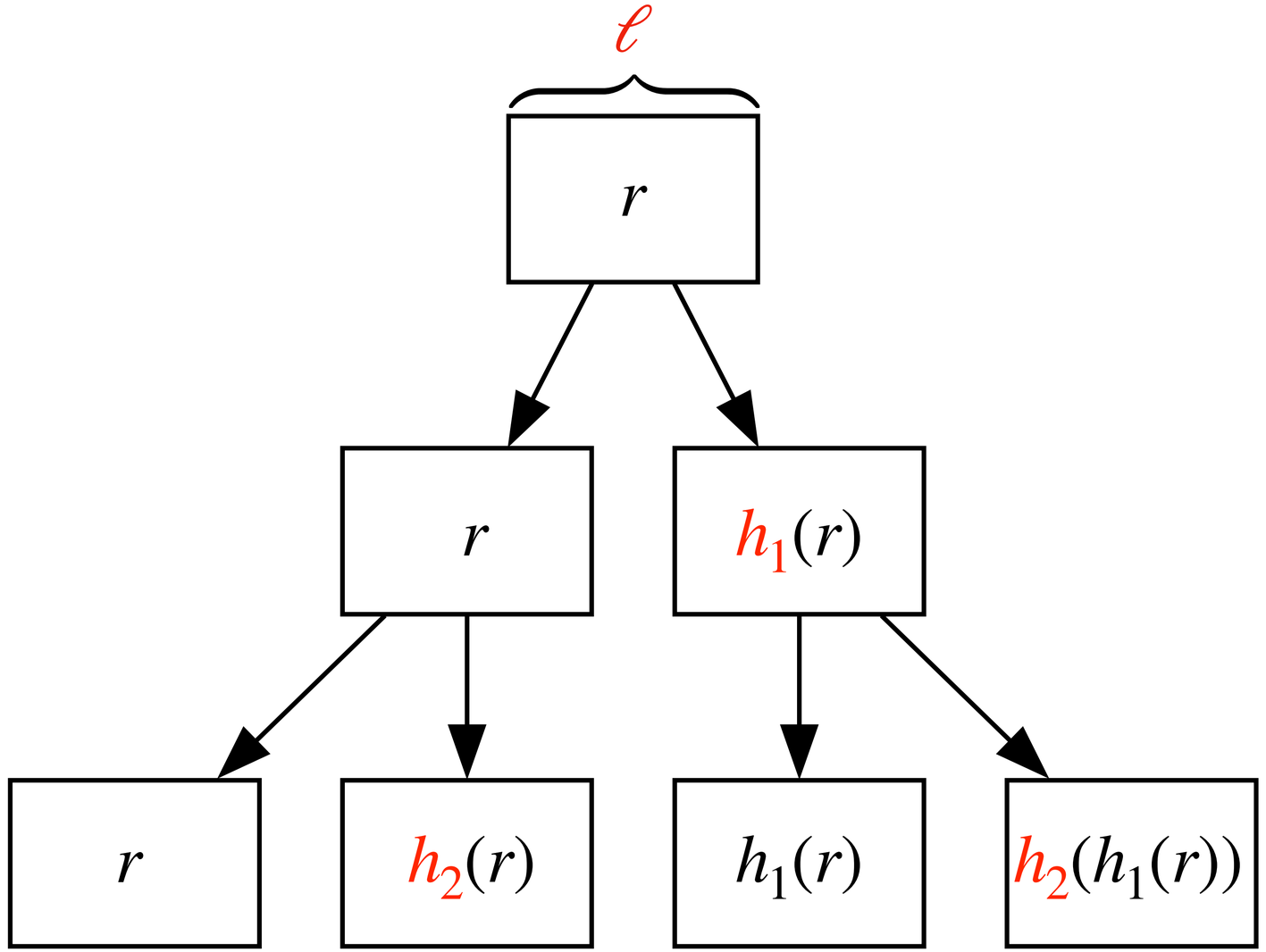
Preserving Properties via Nisan's PRG



- Consider the following algorithm:
  - Uses w bits of space to store its state ($\leq 2^w$ states)
  - Makes a single-pass on the uniform random string
  - Updates its state according to an arbitrary state transition table
- Nisan gave a PRG to "fool" such algorithms by replacing a fully random string with pseudorandom string generated using a **short uniform random seed**
- Indyk gave a recipe to fool turnstile streaming algorithms using such PRGs

Nisan's PRG

- $r \sim \{0,1\}^{\ell}$ and $h_1, \ldots, h_t : \{0,1\}^{\ell} \to \{0,1\}^{\ell}$ 2-wise independent
- Seed length of $O(t \cdot \ell)$
- $2^t \cdot \ell$ length string at the bottom
- If $t, w \leq c \cdot \ell$, the PRG fools an algorithm which uses w bits of space
- Compute block with t hash evaluations
  - Keep t and ℓ small
- Direct derandomization would mean slow update times



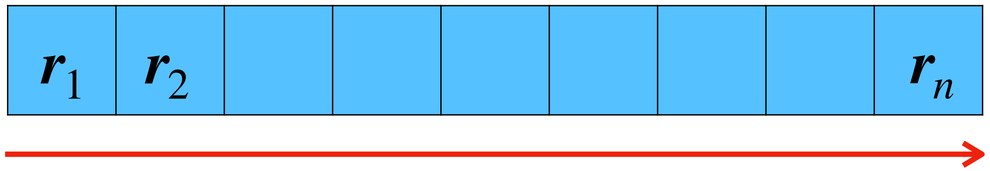
Preserving Properties via Fooling Analysis Algorithms

- We want to sample random variables $\mathbf{e}_1, \ldots, \mathbf{e}_n$ using a small amount of randomness while preserving

$$\max(\mathbf{e}_1^{-1/p}|x_1|, \ldots, \mathbf{e}_n^{-1/p}|x_n|) \equiv \mathbf{e}^{-1/p} F_p^{1/p}$$

Close in TV distance suffices

- Consider the simple algorithm parameterized by x
  - $s \leftarrow 0$
  - For $i = 1, \ldots, n$:
    - $s \leftarrow \max(s, |x_i| \cdot g(\boldsymbol{r}_i)^{-1/p})$



If $\boldsymbol{r}_i$ is a block of uniform random bits, $g(\boldsymbol{r}_i)$ has exponential distribution.

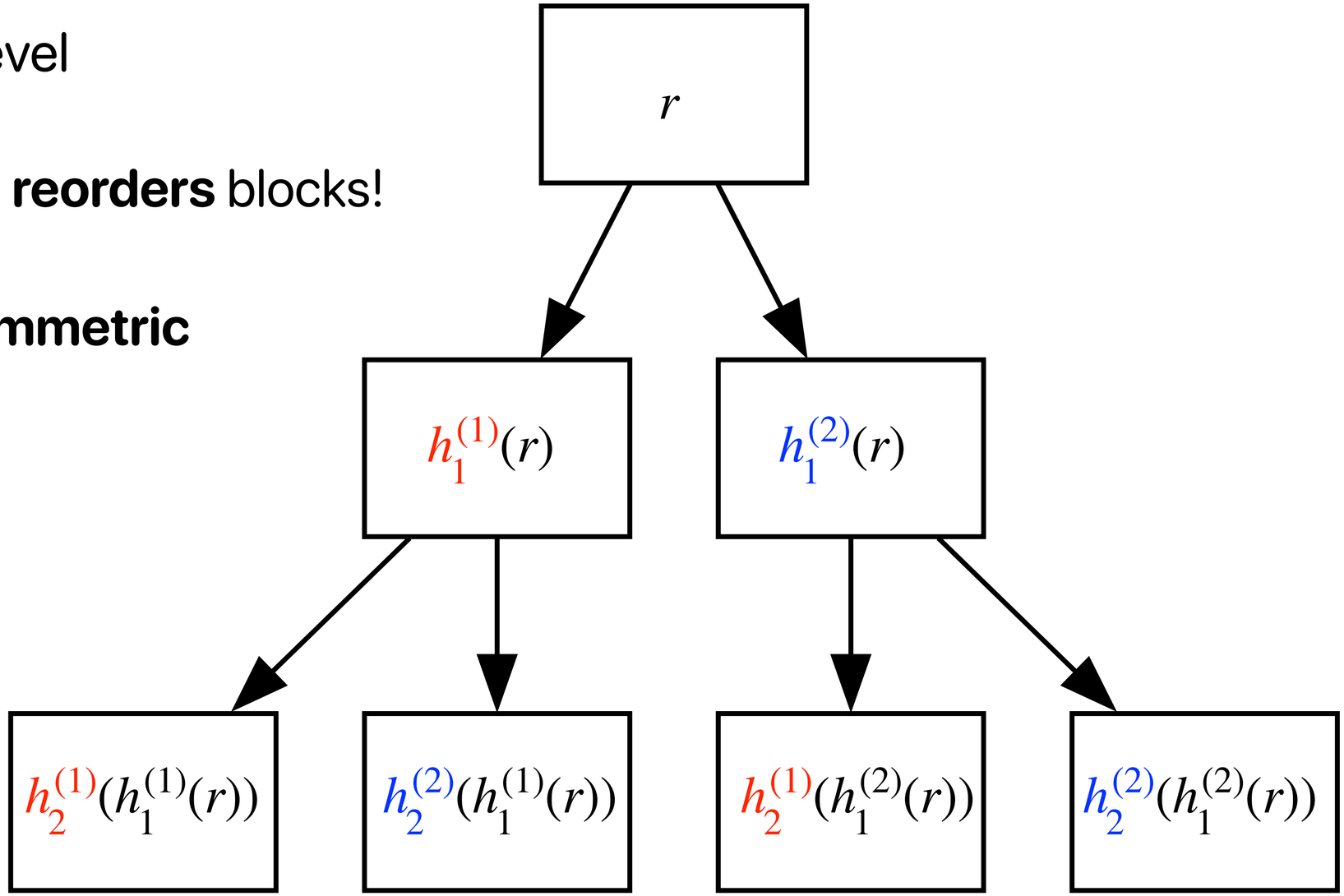
- Preserve the distribution of the output of this small space algorithm

Derandomizing using Nisan's PRG

- Need to fool $O(\log n)$ space algorithm and require poly(n) length bitstring
  - Require $l, t = \Omega(\log n)$
  - Seed of size $O(\log^2 n)$
  - Need to evaluate $O(\log n)$ hash functions to retrieve a block
- We use $\Omega(n^{1-2/p})$ bits to store the sketch
  - Can we use a larger seed for the PRG to decrease $O(\log n)$ hash evaluations?

HashPRG : Our Construction

- **Two** hash functions per level
- Switching $h_1^{(1)}$ and $h_1^{(2)}$ just **reorders** blocks!
- Distribution of strings is **symmetric**



HashPRG

- Why stop at two children per node?
- We show any branching factor works
- Getting to any block of the string is quicker
    - Need to spend more space to store hash functions
    - Time vs space tradeoff

Key Ideas

- We don't need to preserve the distribution of the sketch
    - By more carefully preserving only necessary properties of random variables, we can get by with much weaker PRGs
    - Fast hash function evaluation
- Increase branching factor
    - Fewer hash functions to evaluate for retrieval
- Symmetry of the distribution of pseudorandom strings allows us to derandomize more algorithms

Other Results

- **Theorem**: For $0 < p < 2$, can approximate $F_p(x)$ up to $1 \pm \varepsilon$ using **optimal** $O(\varepsilon^{-2} \log n)$ bits of space and $O(\log n)$ **update time**
  - Valid only for $\varepsilon < 1/n^c$
  - Improves on $O(\log^2 n \log \log n)$ update time of [KNPW '11]
- **CountSketch:** Given t and r, a streaming algorithm which can compute $\hat{x}[i]$ such that for $\alpha \leq 1$

$$\Pr[\,|x[i] - \hat{x}[i]| > \alpha \frac{\|x\|_2}{\sqrt{t}}] \leq 2\exp(-\alpha^2 r) + 1/\text{poly}(n)$$

  - Obtained by derandomizing [Minton and Price '14]
  - The algorithm uses $O(tr \log(n) + \log^2 n)$ bits of space
    - $O(r \log n)$ update time

Other Applications of Sketching

- **Classic**
  - Ridge Regression [KW, AISTATS '20], [KW, ICML '22]
  - Dimensionality Reduction for Sum-of-Distances [FKW, ICML '21]
  - Reduced Rank Regression [KW, COLT '21]
  - Fast and Small Subspace Embeddings [CCKW, SODA '22]
  - PolySketchFormer : Linear Time Transformers obtained via sketching Polynomial Kernels [KMZ, ICML '24]
  - Lower Bounds for Adaptive Matrix Recovery [KW , NeurIPS '23]
  - Fast algorithms for Schatten-p Low Rank Approximation [KW, '24]

Other Applications of Sketching

- **Streaming**
  - Geometric Streaming Algorithms for almost Low Rank Data [EKMWZ, ICML '24]
  - Approximating the Top Eigenvector in Random Order Streams [KW '24]
- **Distributed**
  - Communication Efficient Algorithms in the Personalized CONGEST Model [EKMWZ, STOC '24]

Thank You!