Streaming Graph Computations with a Helpful Advisor

Justin Thaler
Graham Cormode and
Michael Mitzenmacher

Thanks to Andrew McGregor

- A few slides borrowed from IITK Workshop on Algorithms for Processing Massive Data Sets.

Data Streaming Model

- Stream: m elements from universe of size n
  - e.g., $S = \langle x_1, x_2, \ldots, x_m \rangle = 3,5,3,7,5,4,8,7,5,4,8,6,3,2, \ldots$

- Goal: Compute a function of stream, e.g., median, number of distinct elements, frequency moments, heavy hitters.

- Challenge:

  (i) Limited working memory, i.e., sublinear(n,m).

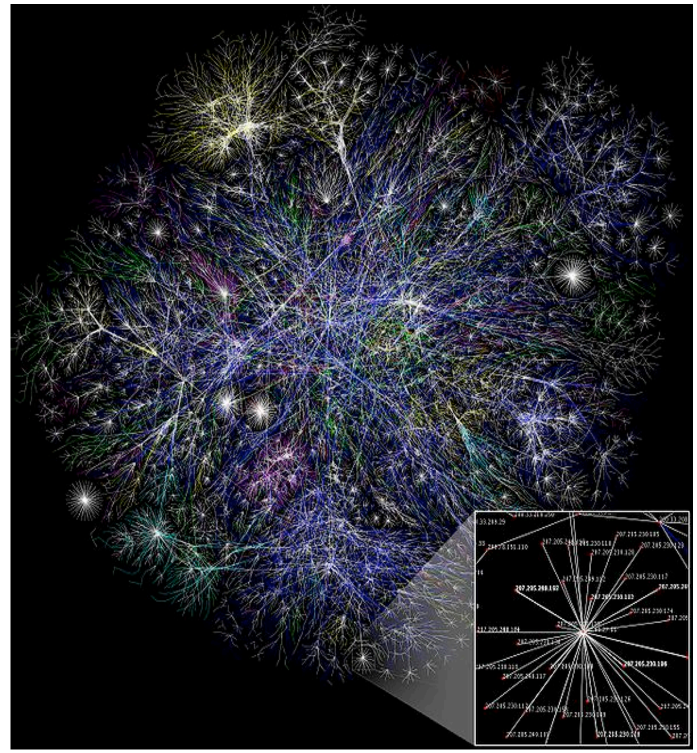
  (ii) Sequential access to adversarially ordered data.

  (iii) Process each update quickly.

Slide derived from [McGregor 10]

Graph Streams

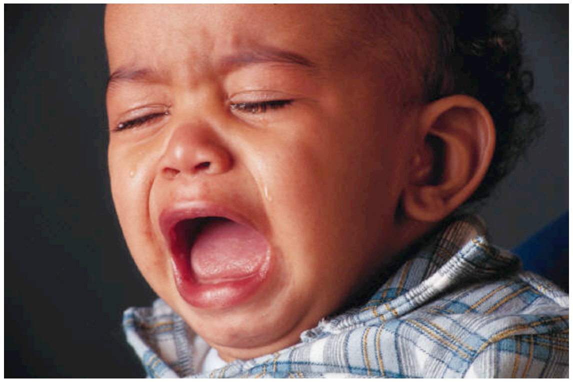
- $S = \langle x_1, x_2, \ldots, x_m \rangle$; $x_i \in [n] \times [n]$
- *A* defines a graph G on n vertices.
- Goal: compute properties of G.
- Challenge: subject to usual streaming constraints.



Snapshot of Internet Graph
Source: Wikipedia

Bad News

- Many graph problems are impossible in standard streaming model (require linear space or many passes over data).
- E.g. $\Omega(n)$ space needed for connectivity, bipartiteness. $\Omega(n^2)$ space needed for counting triangles, diameter, perfect matching.
- Often hard even to approximate.
- Graph problems ripe for outsourcing.



Outsourcing Models

- Stream Punctuation [Tucker et al. 05], Proof Infused Streams [Li et al. 07], Stream Outsourcing [Yi et al. 08], Best-Order Model [Das Sarma et al. 09] (is a special case of our model)

Outsourcing Models

- Stream Punctuation [Tucker et al. 05], Proof Infused Streams [Li et al. 07], Stream Outsourcing [Yi et al. 08], Best-Order Model [Das Sarma et al. 09] (is a special case of our model)

- [Chakrabarti et al. 09] Online Annotation Model: Give streaming algorithm access to powerful *helper* H who can annotate the stream.
    - Main motivation: Commercial cloud computing services such as Amazon EC2. Helper is untrusted.
    - Also, Volunteer Computing (SETI@home. Great Internet Mersenne Prime Search, etc.)
    - Weak peripheral devices.

Online Annotation Model

- **Problem**: Given stream S, want to compute f(S):

$S = \langle x_1, x_2, x_3, x_4, x_5, x_6, \ldots, x_m \rangle$

- Helper H: augments stream with *h*-word annotation:

$(S,a) = \langle x_1, x_2, x_3, x_4, x_5, x_6, \ldots, x_m, a_1, a_2, \ldots, a_h \rangle$

- Verifier V: using *v* words of space and random string *r*, run verification algorithm to compute g(S,a,r) such that for all a either:

  a) $\Pr_r[g(S,a,r) = f(S)] = 1$ (we say a is valid for S) or

  b) $\Pr_r[g(S,a,r) = \perp] \geq 1 - \delta$ (we say a is δ-invalid for S)

  c) And at least one a is valid for S.

Note: this model differs slightly from [Chakrabarti et al. 09].

Online Annotation Model

- Two costs: words of annotation h and working memory v.
  - We refer to (h, v)-protocols.
  - Primarily interested in minimizing v.
  - But strive for optimal tradeoffs between h and v.
  - Proves more challenging for graph streams than numerical streams. Algebraic structure seems critical.

Fingerprinting

- Need a way to test multiset equality (e.g. to see if two streams have the same frequency distribution).
  - But need to do so in a streaming fashion.
  - We often use this to make sure H is "consistent".
- Solution: fingerprints.
  - Hash functions that can be computed by a streaming verifier.
  - If $S \neq S'$ as frequency distributions, then $f(S) \neq f(S')$ w.h.p.
- We choose a fingerprint function f that is linear. $f(S \circ S') = f(S) + f(S')$ where $\circ$ denotes concatenation. Will need this for matrix-vector multiplication.

Two Approaches To Designing Protocols

1. Prove matching upper and lower bounds on a quantity.
   - One bound often easy: just give feasible solution.
   - Proving optimality more difficult. Usually requires problem structure.

2. Use H to "verify" execution of a non-streaming algorithm.

Max-Matching

- [Chakrabarti et al. 09]: (m, 1)-protocol for bipartite Perfect Matching. Also $hv = \Omega(n^2)$ lower bound.

Max-Matching

- [Chakrabarti et al. 09]: (m, 1)-protocol for bipartite Perfect Matching. Also $hv = \Omega(n^2)$ lower bound.
- We give (m, 1)-protocol for general max-cardinality matching.

Max-Matching

- [Chakrabarti et al. 09]: (m, 1)-protocol for bipartite Perfect Matching. Also $hv = \Omega(n^2)$ lower bound.
- We give (m, 1)-protocol for general max-cardinality matching.
- **(Tutte-Berge Formula):** The size of a maximum matching of a graph $G = (V, E)$ equals

$$\tfrac{1}{2} \min_{U \subset V} (|U| - occ(G-U) + |V|)$$

where $occ(H)$ is the number of connected components in the graph H with an odd number of vertices.

Max-Matching

- [Chakrabarti et al. 09]: (m, 1)-protocol for bipartite Perfect Matching. Also $hv = \Omega(n^2)$ lower bound.
- We give (m, 1)-protocol for general max-cardinality matching.
- **(Tutte-Berge Formula):** The size of a maximum matching of a graph $G = (V, E)$ equals

$$\frac{1}{2} \min_{U \subset V} (|U| - occ(G-U) + |V|)$$

where $occ(H)$ is the number of connected components in the graph H with an odd number of vertices.

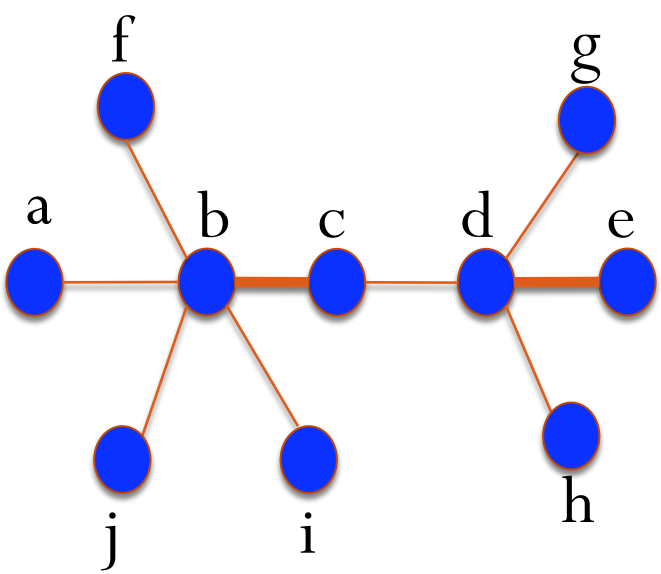
- So for any $U \subset V$, $\frac{1}{2} (|U| - occ(G-U) + |V|)$ is an upper bound on size of max-matching.

Max-Matching

- **(Tutte-Berge Formula):** The size of a maximum matching of a graph $G = (V, E)$ equals

$$\tfrac{1}{2}\, min_{U \subset V}(|U| - occ(G-U) + |V|)$$

where *occ(H)* is the number of components in the graph *H* with an odd number of vertices.

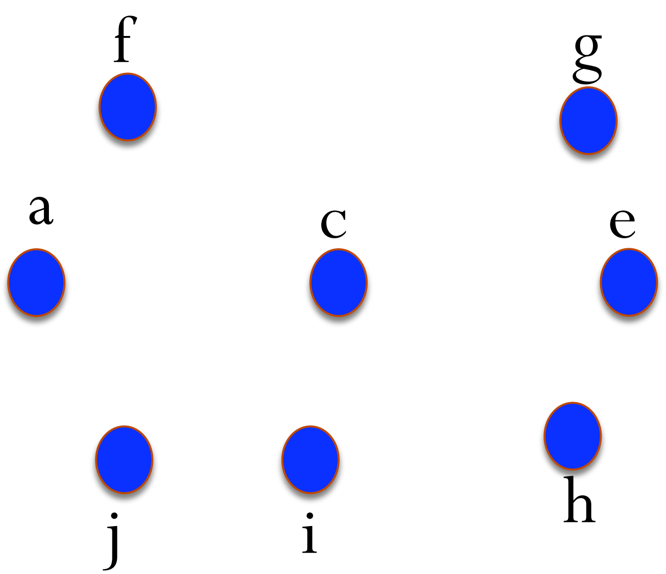


Max-Matching

- **(Tutte-Berge Formula):** The size of a maximum matching of a graph $G = (V, E)$ equals

$$\tfrac{1}{2} \min_{U \subset V} (|U| - occ(G-U) + |V|)$$

where *occ(H)* is the number of components in the graph *H* with an odd number of vertices.



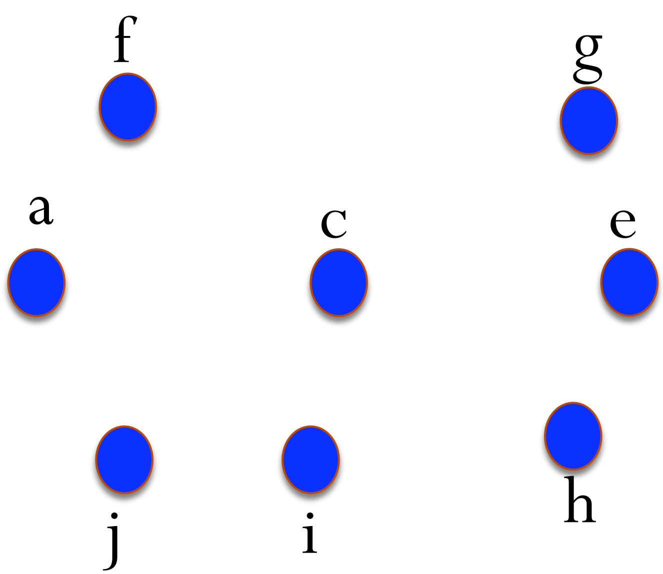
Let U = {b, d}. Then
$\tfrac{1}{2} (|U| - occ(G-U) + |V|) =$
$\tfrac{1}{2} (2 - 8 + 10) = 2.$

Max-Matching

- **(Tutte-Berge Formula):** The size of a maximum matching of a graph $G = (V, E)$ equals

$$\tfrac{1}{2} \min_{U \subset V} (|U| - occ(G-U) + |V|)$$

where $occ(H)$ is the number of components in the graph H with an odd number of vertices.



Let U = {b, d}. Then

$\tfrac{1}{2} (|U| - occ(G-U) + |V|) =$

$\tfrac{1}{2} (2 - 8 + 10) = 2.$

For all other U,

$\tfrac{1}{2} (|U| - occ(G-U) + |V|) \geq 2.$

Max-Matching Protocol

1. H provides a feasible matching of size k. V checks feasibility with fingerprints.
2. H provides $U \subset V$ and claims ½ ($|U|$ -*occ*(*G-U*) + $|V|$)=k. If so, V accepts answer k. Else, V rejects.

- Caveat: H must provide proof of the value of *occ*(*G-U*), because V cannot do this on her own.

Streaming LP problem

- Suppose stream A contains (only the non-zero) entries of matrix $\mathbf{A}$, vectors $\mathbf{b}$ and $\mathbf{c}$, interleaved in any order (updates are of the form e.g. "add y to entry (i,j) of $\mathbf{A}$"). The LP streaming problem on A is to determine $\max \{\mathbf{c}^T \mathbf{x} \mid \mathbf{A}\mathbf{x} \le \mathbf{b}\}$.

Streaming LP problem

- Suppose stream A contains (only the non-zero) entries of matrix **A**, vectors **b** and **c**, interleaved in any order (updates are of the form e.g. "add y to entry (i,j) of **A**"). The LP streaming problem on A is to determine max $\{\mathbf{c}^T \mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$.

- Theorem: There is a $(|\mathbf{A}|, 1)$ protocol for the LP streaming problem, where $|\mathbf{A}|$ is number of non-zero entries in **A**.

Streaming LP problem

- Suppose stream A contains (only the non-zero) entries of matrix $\mathbf{A}$, vectors $\mathbf{b}$ and $\mathbf{c}$, interleaved in any order (updates are of the form e.g. "add y to entry (i,j) of $\mathbf{A}$"). The LP streaming problem on A is to determine max $\{\mathbf{c}^T \mathbf{x} \mid \mathbf{A}\mathbf{x} \leq \mathbf{b}\}$.

- Theorem: There is a $(|\mathbf{A}|, 1)$ protocol for the LP streaming problem, where $|\mathbf{A}|$ is number of non-zero entries in $\mathbf{A}$.
  - Protocol ("naïve" matrix-vector multiplication):
    1. H provides primal-feasible solution $\mathbf{x}$.
    2. For each row i of $\mathbf{A}$:

       Repeat entries of $\mathbf{x}$ and row i of $\mathbf{A}$ in order to prove feasibility. Fingerprints ensure consistency.
    3. Repeat for dual-feasible solution $\mathbf{y}$. Accept if value($\mathbf{x}$)=value($\mathbf{y}$).

Application to Graph Streams

- Corollary: Protocol for TUM IPs, since optimality can be proven via a solution to the dual of its LP relaxation.

Application to Graph Streams

- Corollary: Protocol for TUM IPs, since optimality can be proven via a solution to the dual of its LP relaxation.
- Corollary: (m, 1) protocols for max-flow, min-cut, minimum-weight bipartite perfect matching, and shortest *s*-*t* path. Lower bound of $hv = \Omega(n^2)$ for all four.

Application to Graph Streams

- Corollary: Protocol for TUM IPs, since optimality can be proven via a solution to the dual of its LP relaxation.
- Corollary: (m, 1) protocols for max-flow, min-cut, minimum-weight bipartite perfect matching, and shortest *s*-*t* path. Lower bound of $hv = \Omega(n^2)$ for all four.
- **A** is sparse for the problems above, which suits the naïve protocol. For denser **A**, can get optimal tradeoffs between *h* and *v*.

Dense Matrix-Vector Multiplication

- We will get optimal $(n^{1+\alpha}, n^{1-\alpha})$ protocol. Lower bound: $hv = \Omega(n^2)$.
  - Corollary I: Protocols for dense LPs, effective resistances, verifying eigenvalues of Laplacian.

Dense Matrix-Vector Multiplication

- We will get optimal $(n^{1+\alpha}, n^{1-\alpha})$ protocol. Lower bound: $hv = \Omega(n^2)$.
  - Corollary I: Protocols for dense LPs, effective resistances, verifying eigenvalues of Laplacian.
  - Corollary II: Optimal tradeoffs for Quadratic Programs, Second-Order Cone Programs. $(n^2, 1)$ protocol for Semi-definite Programs.

Dense Matrix-Vector Multiplication

- First idea: Treat as n separate inner-product queries, one for each row of A.
  - Worse than "naïve" solution.
  - Multiplies *both* h and v by n, as compared to a single inner-product query.

Dense Matrix-Vector Multiplication

- First idea: Treat as n separate inner-product queries, one for each row of A.
  - Worse than "naïve" solution.
  - Multiplies *both* *h* and *v* by n, as compared to a single inner-product query.
- Key observation: one vector, **x**, in each inner-product query is constant.
  - This plus linear fingerprints lets us just multiply *h* by n.
  - *v* will be the same as for a *single* inner product query.

Approach 2: Simulate an Algorithm

- Main tool: Offline memory checker [Blum et al. '94]. Allows efficient verification of a sequence of accesses to a large memory.

- Lets us convert any deterministic algorithm into a protocol in our model.

- Running time of the algorithm in the RAM model becomes annotation size h.

Memory Checker [Blum et al. '94]

- Consider a *memory transcript* of a sequence of reads and writes to memory.
- A transcript is *valid* if each read of address i returns the last value written to that address.
- Memory checker requires transcript be provided in a carefully chosen ("augmented") format.
  - Augmentation blows up transcript size only by constant factor.
- *V* checks validity by keeping a constant number of fingerprints and performing simple local checks on the transcript.

Simulation Theorem

- Any graph algorithm *M* in RAM model requiring time t can be (verifiably) simulated by an (m+t, 1)-protocol.

- *Proof sketch:*
  - Step 1: H first plays a valid adjacency-list representation of *G* to "initialize memory".
  - Step 2: H provides a valid augmented transcript T of the read and write operations performed by algorithm.
  - V checks validity using memory-checker. V also checks all read/write operations are as prescribed by *M*.

Simulation Theorem

- Corollary: (m, 1)-protocol for MST; (m + n log n, 1)-protocol to verify single-source shortest paths; $(n^3,1)$-protocol for all-pairs shortest paths.

Simulation Theorem

- Corollary: (m, 1)-protocol for MST; (m + n log n, 1)-protocol to verify single-source shortest paths; (n^3,1)-protocol for all-pairs shortest paths.
- Proof for MST: Given a spanning tree T, there exists a linear-time algorithm *M* for verifying that T is minimum e.g. [King '97].

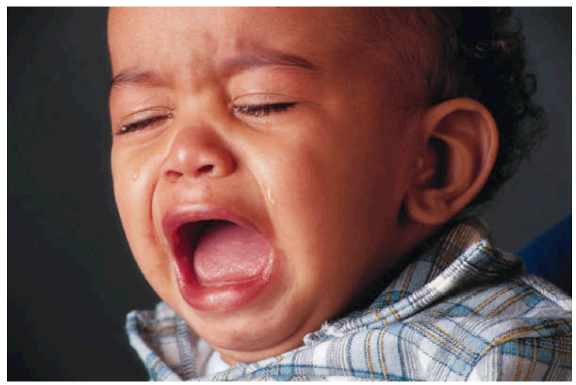
Simulation Theorem

- Corollary: (m, 1)-protocol for MST; (m + n log n, 1)-protocol to verify single-source shortest paths; $(n^3,1)$-protocol for all-pairs shortest paths.
- Proof for MST: Given a spanning tree T, there exists a linear-time algorithm *M* for verifying that T is minimum e.g. [King '97].

- Lower bounds: hv= $\Omega(n^2)$ for single source and all-pairs shortest paths. *hv*= $\Omega(n^2)$ for MST if edge weights specified incrementally.

Pitfall of Memory-Checking

Cannot simulate *randomized* algorithms



Diameter

- Theorem: $(n^2 \log n, 1)$ protocol. Lower bound: $hv = \Omega(n^2)$.

Diameter

- Theorem: ($n^2 \log n$, 1) protocol. Lower bound: $hv = \Omega(n^2)$.
- [Chakrabarti et al. 09]: (n^2, 1) protocol for matrix-matrix multiplication.

Diameter

- Theorem: ($n^2 \log n$, 1) protocol. Lower bound: $hv = \Omega(n^2)$.
- [Chakrabarti et al. 09]: (n^2, 1) protocol for matrix-matrix multiplication.
- Let A be adjacency matrix of *G.*

Diameter

- Theorem: (n^2 log n, 1) protocol. Lower bound: *hv*= Ω (n^2).
- [Chakrabarti et al. 09]: (n^2, 1) protocol for matrix-matrix multiplication.
- Let A be adjacency matrix of *G.*
- $(I + A)^l_{ij} > 0$ if and only if there is a path of length at most l from i to j.

Diameter

- Theorem: ($n^2 \log n$, 1) protocol. Lower bound: $hv = \Omega(n^2)$.
- [Chakrabarti et al. 09]: (n^2, 1) protocol for matrix-matrix multiplication.
- Let A be adjacency matrix of <u>*G.*</u>
- $(I + A)^l_{ij} > 0$ if and only if there is a path of length at most l from i to j.
- Protocol:

1. H claims diameter is l
2. Use repeated squaring to prove $(I+A)^l$ has an entry that is 0, and $(I+A)^{l+1} \neq 0$ for all i,j).

Summary

- (m, 1)-protocol for max-matching. $hv = \Omega(n^2)$ lower bound for dense graphs, so we can't do better.
- (m, 1)-protocols for LPs TUM IPs. $hv = \Omega(n^2)$ lower bound for several TUM IPs.
- Optimal $(n^{1+\alpha}, n^{1-\alpha})$-protocol for dense matrix-vector multiplication. $(n^{1+\alpha}, n^{1-\alpha})$-protocols for effective resistance, verifying eigenvalues of Laplacian or Adjacency matrix, LPs, QPs, SOCPs.
- General simulation theorem; applications to MST, shortest paths.
- $(n^2 \log n, 1)$ protocol for Diameter. $hv = \Omega(n^2)$ lower bound.

Open questions

- Tradeoffs between h, v for matching, MST, diameter?
- Distributed computation: Protocols that work with Map-Reduce.
- What if we allow multiple rounds of interaction between H and V? Can we get exponentially better protocols?

Verifying Computations with Streaming Interactive Proofs

With Graham Cormode and Ke Yi

A General Result

- Universal Arguments [Kilian 92] and Interactive Proofs for Muggles [Goldwasser, Kalai, Rothblum 08] can work with streaming verifier!

A General Result

- Universal Arguments [Kilian 92] and Interactive Proofs for Muggles [Goldwasser, Kalai, Rothblum 08] can work with streaming verifier!

- Therefore: (polylog u, polylog u) computationally sound protocols for NP. (polylog u, polylog u) statistically sound protocols for all of log-space uniform NC. u is input size.

A General Result

- Universal Arguments [Kilian 92] and Interactive Proofs for Muggles [Goldwasser, Kalai, Rothblum 08] can work with streaming verifier!
- Therefore: (polylog u, polylog u) computationally sound protocols for NP. (polylog u, polylog u) statistically sound protocols for all of log-space uniform NC. u is input size.
- Efficient protocols even for problems hard in *non*-streaming setting.
- Exponential improvement over best-possible one-round protocols.

How to Make V Streaming

- Arithmetization: Given function *f'*, extend domain of *f'* to field and replace *f'* with its low-degree extension (LDE) f as a polynomial over the field.

- Can view *f* as a high-distance encoding of *f'*. The error correcting properties of *f* give V considerable power over H.

How to Make V Streaming

- Three observations:
  - 1. In many proof systems, V only accesses the input in order to compute $f(\mathbf{r})$ for small number of $\mathbf{r}$, where f is LDE of input.
  - 2. Moreover, locations $\mathbf{r}$ only depend on V's random coins.
  - 3. V can evaluate $f(\mathbf{r})$ in streaming fashion.

How to Make V Streaming

- Three observations:
  - 1. In many proof systems, V only accesses the input in order to compute $f(\mathbf{r})$ for small number of $\mathbf{r}$, where f is LDE of input.
  - 2. Moreover, locations $\mathbf{r}$ only depend on V's random coins.
  - 3. V can evaluate $f(\mathbf{r})$ in streaming fashion.

- So streaming V tosses all coins in advance; remembers them and keeps them private from H; and computes $f(\mathbf{r})$ during "input observation" phase.

Streaming V can evaluate f(r)

- E.g. Let **a** be the u-dimensional frequency vector of a stream. and view the universe [u] as $[\ell]^d$ where $\ell^d = u$ ("frequency hypercube").
  - Then $f(\mathbf{x}) = \sum_{\mathbf{v} \in [\ell]^d} a_{\mathbf{v}} \chi_{\mathbf{v}}(\mathbf{x})$.
    - Where $\chi_{\mathbf{v}}(\mathbf{v}) = \mathbf{1}$ and $\chi_{\mathbf{v}}(\mathbf{v'}) = 0$ for all other $\mathbf{v'} \in [\ell]^d$.

Streaming V can evaluate f(r)

- E.g. Let **a** be the u-dimensional frequency vector of a stream. and view the universe [u] as $[\ell]^d$ where $\ell^d = u$ ("frequency hypercube").
  - Then $f(\mathbf{x}) = \sum_{\mathbf{v} \in [\ell]^d} a_{\mathbf{v}} \ \chi_{\mathbf{v}}(\mathbf{x})$.
    - Where $\chi_{\mathbf{v}}(\mathbf{v}) = \mathbf{1}$ and $\chi_{\mathbf{v}}(\mathbf{v'}) = 0$ for all other $\mathbf{v'} \in [\ell]^d$.

- V makes one pass over the data stream. If V observes a new entry $a_{\mathbf{v}}$ of the input, V may update

  $$f(\mathbf{r}) \leftarrow f(\mathbf{r}) + a_{\mathbf{v}} \cdot \chi_{\mathbf{v}}(\mathrm{r}).$$

Some comments

- Despite powerful generality, [Goldwasser, Kalai, Rothblum 08] is not optimal for many low-complexity functions of high interest in streaming, database processing.

- E.g. Frequency Moments, Reporting Queries.

- We give improved protocols for these problems.
  - And argue that they are practical.

Tool: Sum-Check Protocol

- Let g be a polynomial over $\mathbf{F}_p$.

Tool: Sum-Check Protocol

- Let g be a polynomial over $\mathbf{F}_p$.
- Say we want to compute $\sum_{\mathbf{z} \in H^d} g(\mathbf{z})$ for some $H \subseteq \mathbf{F}_p$.

Tool: Sum-Check Protocol

- Let g be a polynomial over $\mathbf{F}_p$.
- Say we want to compute $\sum_{\mathbf{z} \in H^d} g(\mathbf{z})$ for some $H \subseteq \mathbf{F}_p$.
- A Sum-Check Protocol lets V do this as long as V can evaluate g at a randomly-chosen location $\mathbf{r}$.

Tool: Sum-Check Protocol

- Let g be a polynomial over $\mathbf{F}_p$.
- Say we want to compute $\sum_{\mathbf{z} \in H^d} g(\mathbf{z})$ for some $H \subseteq \mathbf{F}_p$.
- A Sum-Check Protocol lets V do this as long as V can evaluate g at a randomly-chosen location $\mathbf{r}$.
- Requires d rounds, communication cost in round i is $\deg_i(g)$, the degree of g in variable i.

F_2 protocol

- Goal: Compute $\sum_i a_i^2$

F_2 protocol

- Goal: Compute $\sum_i a_i^2$
- First attempt: Let $\mathbf{a}^2$ denote the entry-wise square of $\mathbf{a}$. Try to apply a sum-check protocol to the LDE g of $\mathbf{a}^2$.
  - i.e. $g = \sum_{v \in [\ell]^d} a^2_{\mathbf{v}} \ \chi_{\mathbf{v}}$.
  - But a streaming verifier cannot evaluate g at a random location.

F_2 protocol

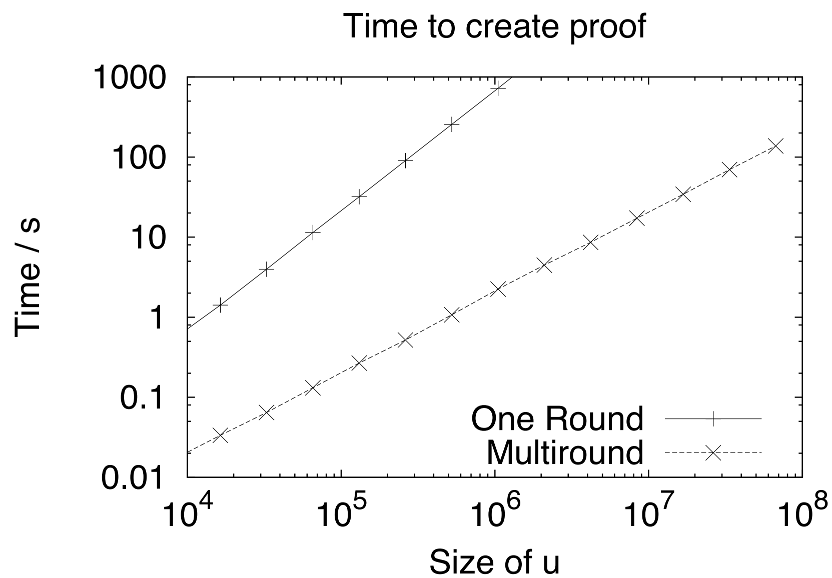
- Goal: Compute $\sum_i a_i^2$
- First attempt: Let $\mathbf{a}^2$ denote the entry-wise square of $\mathbf{a}$. Try to apply a sum-check protocol to the LDE g of $\mathbf{a}^2$.
  - i.e. $g = \sum_{v \in [\ell]^d} a^2_{\mathbf{v}} \ \chi_{\mathbf{v}}$.
  - But a streaming verifier cannot evaluate g at a random location.
- But V can use a slightly higher-degree extension of $\mathbf{a}^2$ instead.
  - i.e. $f^2 = (\sum_{\mathbf{v} \in [\ell]^d} a_{\mathbf{v}} \ \chi_{\mathbf{v}})^2$
  - We know V can evaluate $f(\mathbf{r})$, and $f^2(\mathbf{r}) = f(\mathbf{r})^2$.

Experiments

- Implemented one-round F_2 protocol from [Chakrabarti et al. 09] and multiround F_2 protocol.
  - Single-round space and communication cost grows like $\sqrt{u}$. Still under a megabyte for u=100 million.
  - Multiround space and communication always under 1 KB even when handling GBs of data.

Experiments

- V takes about the same time in both cases (millions of updates per second). But H much more efficient in multiround case.
  - E.g. Multiround H requires less than a second to process streams with millions of updates and u=[250K]. Single-round H requires minutes on same data.
  - Multi-round H's time grows linearly, single-round H's time grows like $u^{3/2}$.



Extension to Frequency-Based Functions

- Frequency based function $F(\mathbf{a})$ is of the form $F(\mathbf{a}) = \sum_i h(\mathbf{a}_i)$ for some $h: \mathbf{N_0} \rightarrow \mathbf{N_0}$.

- e.g. F_k, F_0 (DISTINCT), "How many items have frequency at most i?", verifying F_{max} (highest-frequency).

Extension to Frequency-Based Functions

- First idea: extend h to a polynomial h over $\mathbf{F}_p$ and apply a sum-check protocol to the polynomial $h \circ f$.
  - Streaming V can evaluate $h \circ f(\mathbf{r})$ by computing $f(\mathbf{r})$ and then $h(f(\mathbf{r}))$.
  - Problem: h might have degree u. Resulting communication cost is du, worse than trivial protocol.

Extension to Frequency-Based Functions

- First idea: extend h to a polynomial h over $\mathbf{F}_p$ and apply a sum-check protocol to the polynomial $h \circ f$.
  - Streaming V can evaluate $h \circ f(\boldsymbol{r})$ by computing $f(\boldsymbol{r})$ and then $h(f(\boldsymbol{r}))$.
  - Problem: h might have degree u. Resulting communication cost is du, worse than trivial protocol.

- Solution: We give a ($1/\phi \log u$, $1/\phi \log u$) protocol to identify all items of frequency at least ϕm (the "ϕ-heavy hitters"). Use this protocol to "remove" the heavy items, which allows to control degree of h.

Extension to Frequency-Based Functions

- Result: a ($\sqrt{u} \log u$, $\log u$)-protocol for any frequency-based function that takes log u rounds.

- [Goldwasser, Kalai, Rothblum 08] yields ($\log^2 u$, $\log^2 u$) protocol.

- For 1 TB of data, $\sqrt{u}$ is on the order of 1 MB, $\log^2 u$ is on the order of thousands, $\log u \approx 40$.

- Might prefer to communicate 1 MB of data over 40 rounds than 1 KB over thousands of rounds due to network latency.

Reporting Queries

- Sub-vector query: Given q_L and q_R, determine the non-zero entries of $(a_{q_L}, \ldots, a_{q_R})$.

- We give a $(k + \log u, \log u)$-protocol for Sub-vector requiring $\log u$ rounds, where k is number of non-zero entries in $(a_{q_L}, \ldots, a_{q_R})$.

- In comparison, [Goldwasser, Kalai, Rothblum 08] yields $(k' \log u, k' \log u)$-protocol, where $k' = O(q_R - q_L)$.
  - Improvement is significant when k' is large or the subvector is sparse.

- Protocol is reminiscent of Merkle trees, but we achieve statistical soundness.

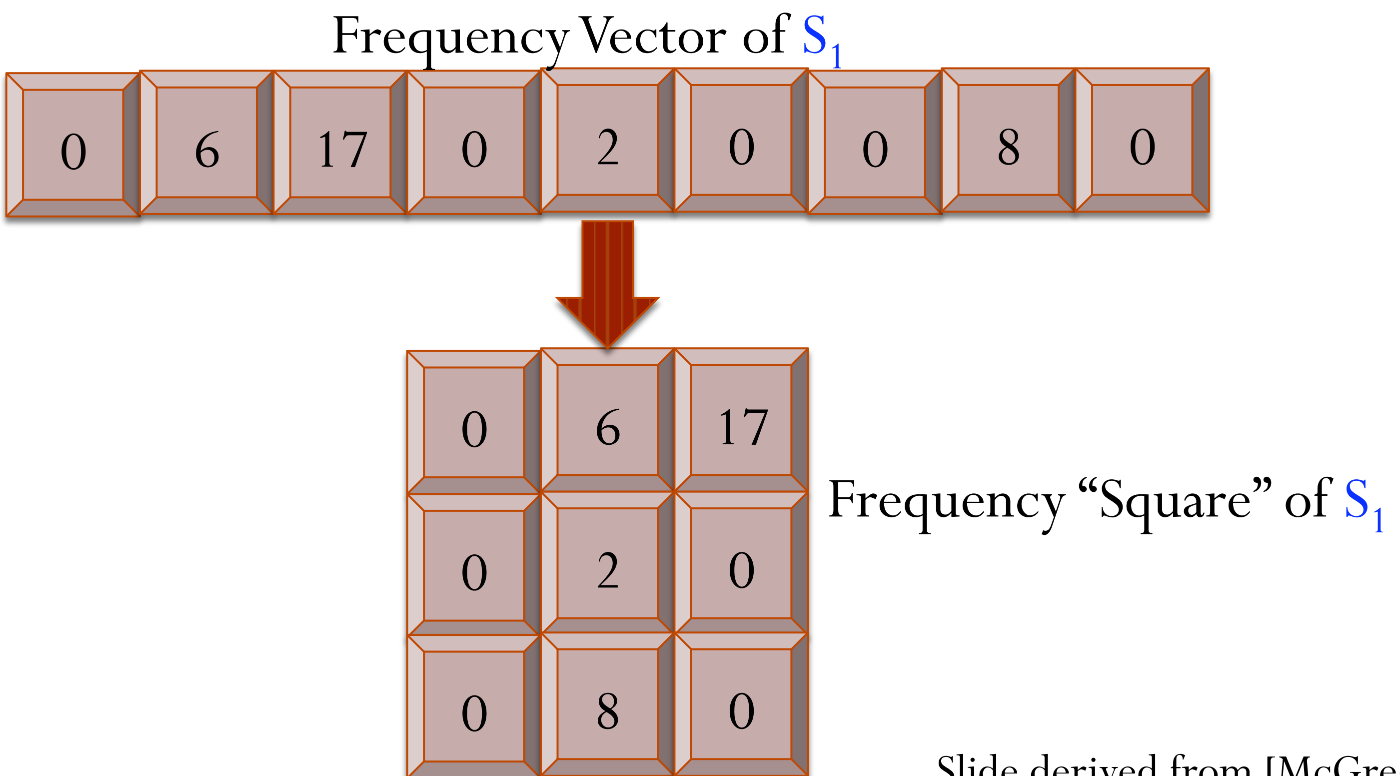
Open Questions

- Reusability?
- Do problems outside of NC possess streaming interactive proofs?
- Better protocols for specific candidates? Prime candidates: F0, Fmax.
- Distributed Computation: Our prover’s messages naturally lend themselves to Map-Reduce setting. Remains to demonstrate this empirically.

Thank you!

Matrix-Vector Multiplication

- Background: [Chakrabarti et al. 09] ($\sqrt{n}$, $\sqrt{n}$)-protocol for inner product of frequency vectors of two streams S_1, S_2.
- View universe [n] as [$\sqrt{n}$] x [$\sqrt{n}$].



Slide derived from [McGregor 10]

Inner-Product Protocol (1/4)

- Want to compute inner product of frequency vectors of S_1, S_2.

Frequency Square of S_1

0	6	17
0	2	0
0	8	1

Frequency Square of S_2

8	2	0
19	21	0
4	8	3

Slide derived from [McGregor 10]

Inner-Product Protocol (2/4)

- First idea: Have H send the inner product "in pieces":
  - row 1 · row 1, row 2 · row 2, etc. Requires $\sqrt{n}$ communication.
- V exactly tracks a piece at random (denoted in yellow) so if H lies about any piece, V has a chance of catching her. Requires space $\sqrt{n}$.

Frequency Square of S_1

0	6	17
0	2	0
0	8	1

Frequency Square of S_2

8	2	0
19	21	0
4	8	3

H sends
12
42
67

Slide derived from [McGregor 10]

Inner-Product Protocol (3/4)

- Problem: If H lies in only one place, V has small chance of catching her.
- Solution: Have H commit (succinctly) to inner products of pieces of a high-distance encoding of the input. If H lies about one piece, she will have to lie about many.
- Need V to evaluate any piece of the encoding in a streaming fashion. Can do this for "low-degree extension" code.

Inner-Product Protocol (4/4)

High-Distance Encoding g
of Frequency Square of S_1

0	6	17
0	2	0
0	8	1
2	8	1
3	7	3
1	2	2

Input is embedded in encoding (low-degree extension)

High-Distance Encoding h
of Frequency Square of S_2

8	2	0
19	21	0
4	8	3
2	3	5
7	8	1
0	2	0

These values will all lie on low-degree polynomial s(x)

H sends

12

42

67

33

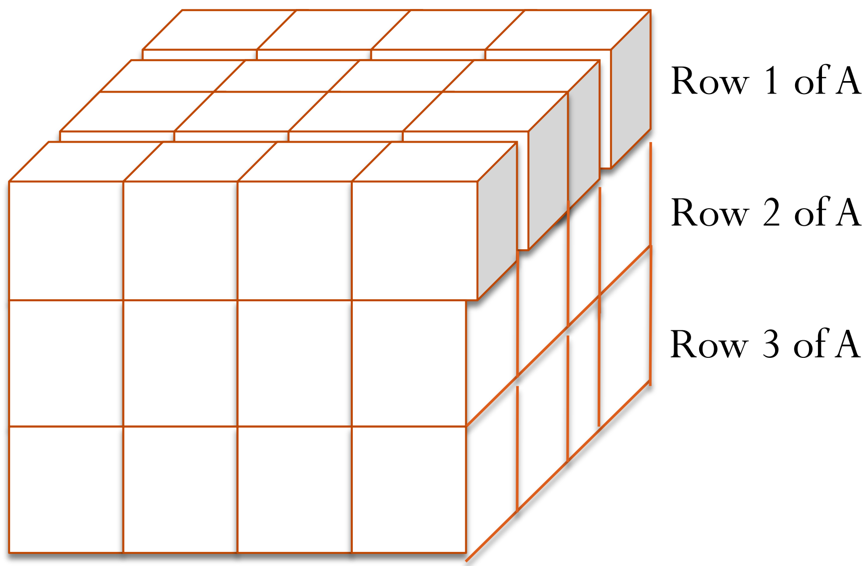
80

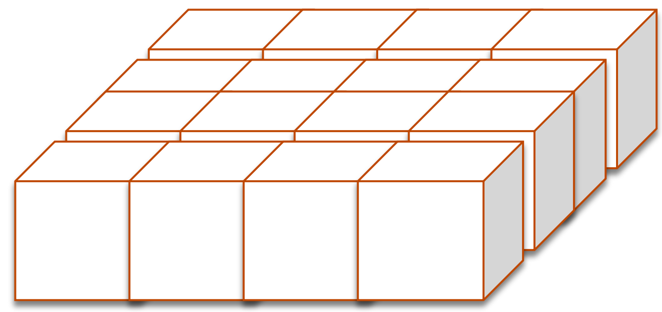
4

Matrix-Vector Multiplication (1/7)

- First idea: Treat as n separate inner-product queries, one for each row of A.
  - Worse than "naïve" solution.
  - Multiplies *both* h and v by n, as compared to a single inner-product query.
- Key insight: one vector, **x**, in each inner-product query is constant.
  - This plus linear fingerprints lets us just multiply h by n.
  - v will be the same as for a *single* inner product query.

Matrix-Vector Multiplication (2/7)





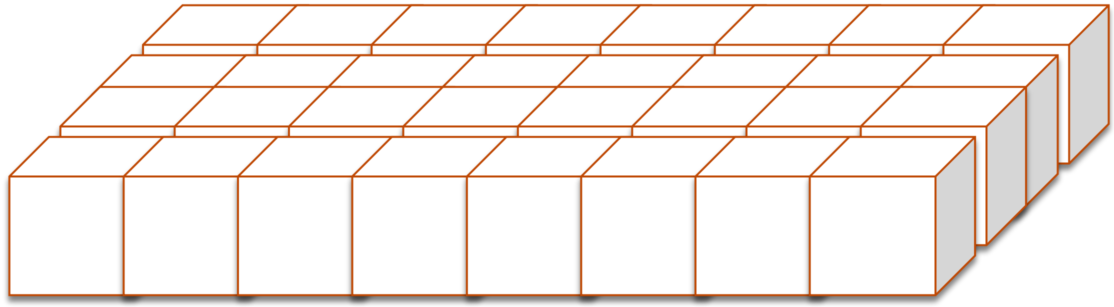
x

A

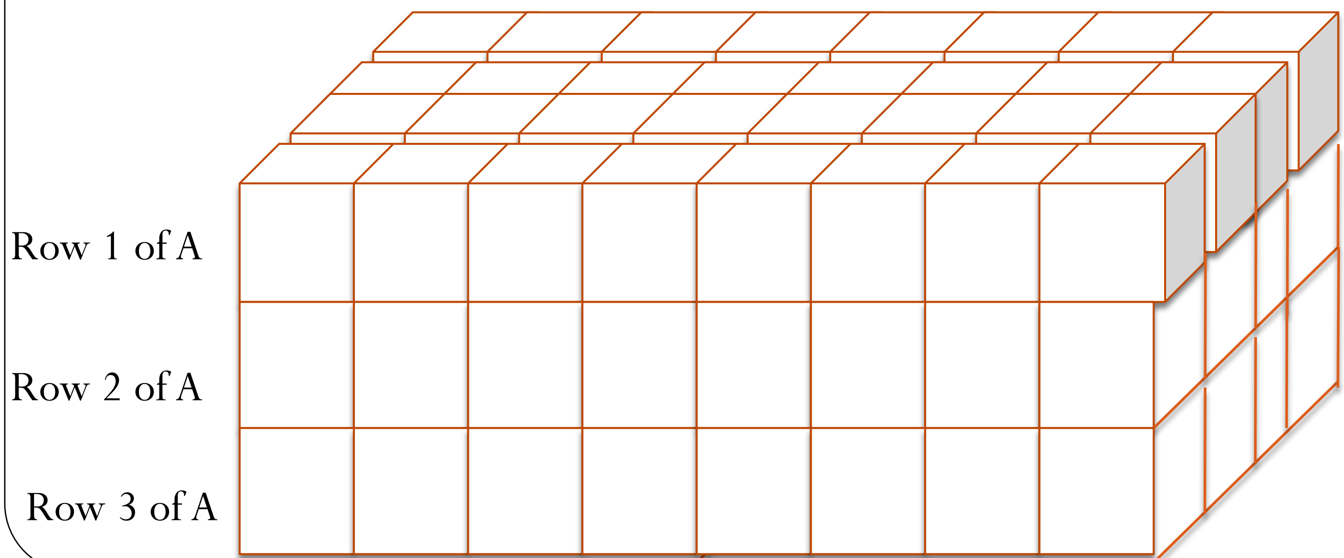
Even though this is drawn as a cube, suppose box has dimensions n x $\sqrt{n}$ x $\sqrt{n}$

Matrix-Vector Multiplication (3/7)

Low-degree extension of each row of A and of **x**



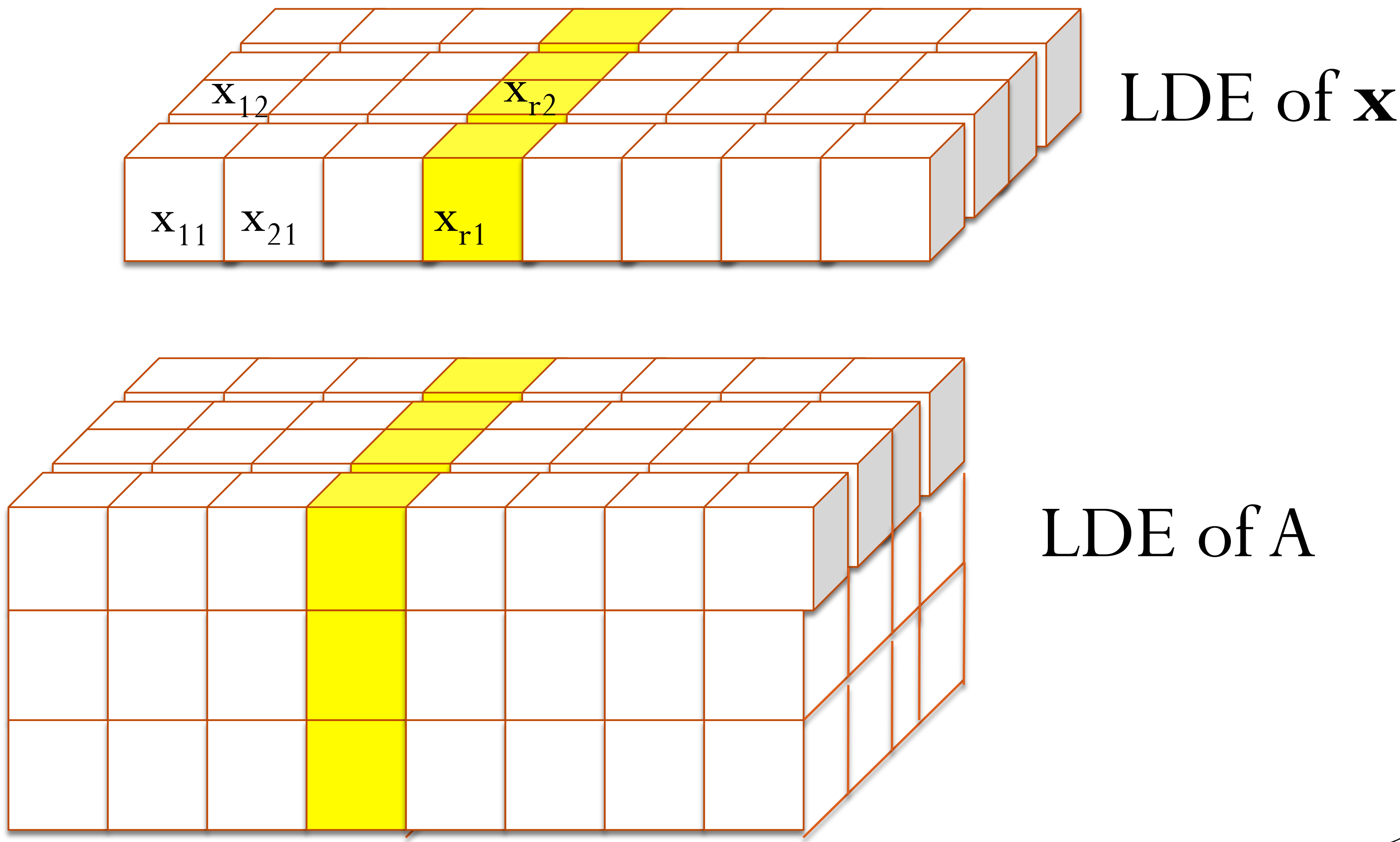
LDE of **x**



LDE of A

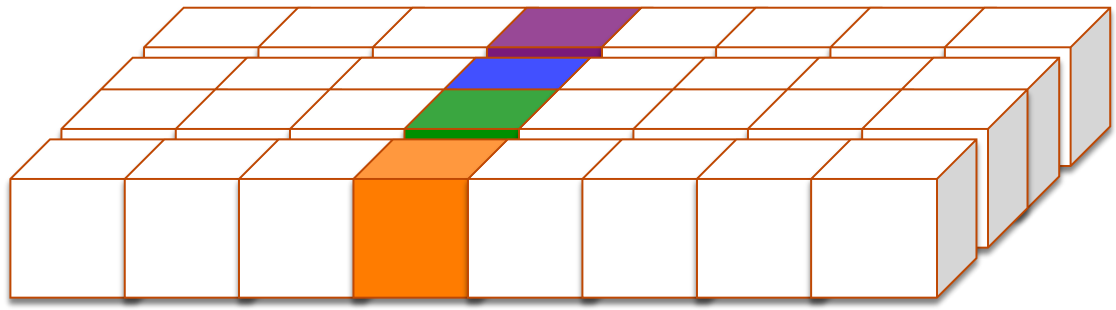
Matrix-Vector Multiplication (4/7)

V evaluates each row of A and **x** at random "piece" r (same r for all rows)

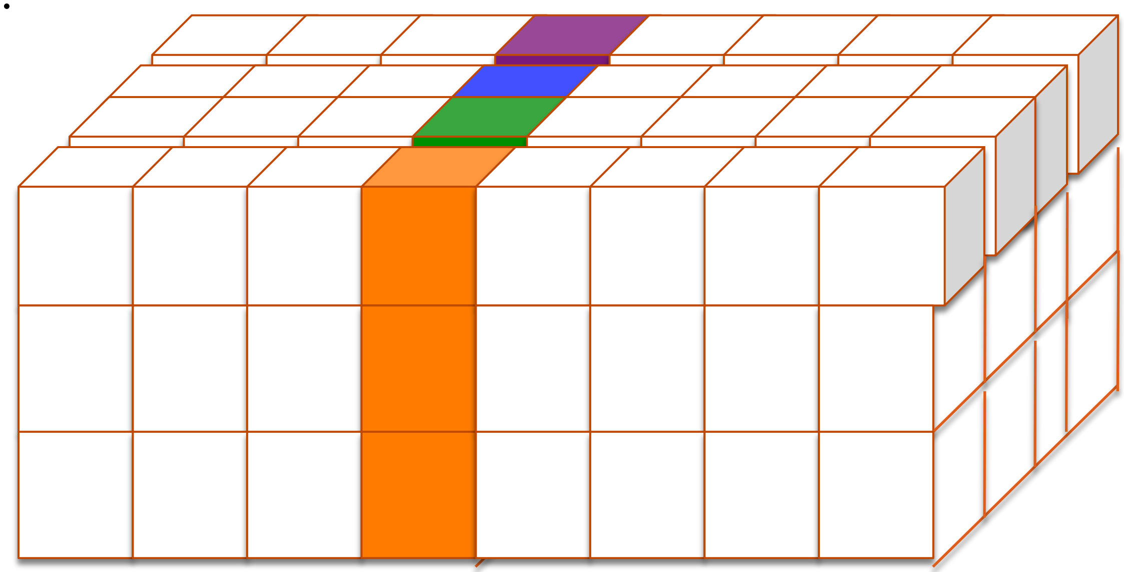


Matrix-Vector Multiplication (5/7)

Each orange entry of A gets multiplied by orange entry of **x** when computing inner product of its “piece”.

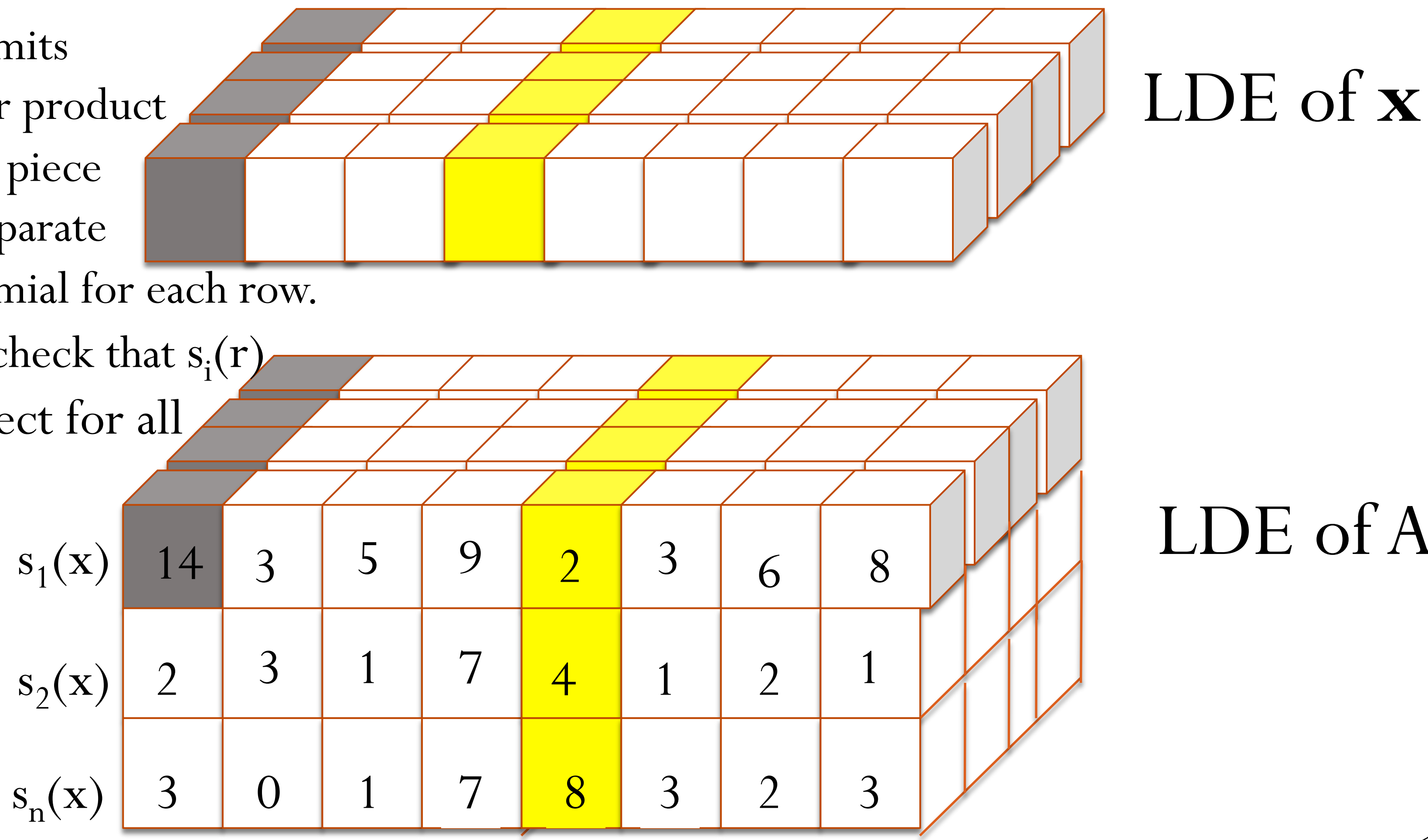


Only need to keep one fingerprint for each color.



Matrix-Vector Multiplication (6/7)

H commits to inner product of each piece via a separate polynomial for each row. V will check that $s_i(r)$ is correct for all rows i.



Matrix-Vector Multiplication (7/7)

- Summary:
  - H sends the inner product of each piece of each row.
  - Conceptually, V will track a random piece of each row (the yellow entries) to catch H in any lies w.h.p.
  - But V need not store all $n * \sqrt{n}$ yellow entries!
    - Can store just $\sqrt{n}$ fingerprints $f_1, \ldots, f_{\sqrt{n}}$
    - Each fingerprint aggregates over n rows, can be computed incrementally by streaming verifier.
    - Works because vector **x** is fixed.