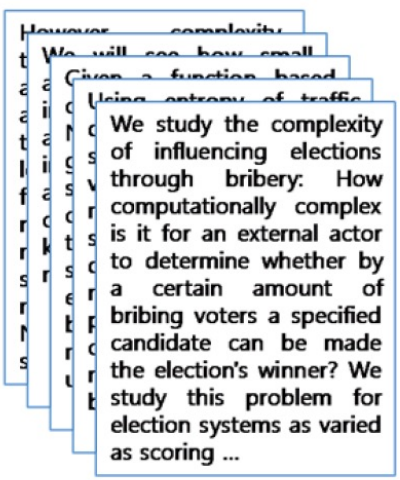
Matrix Sketching in Data Streams

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Mina Ghashami, Amazon
http://cs246.stanford.edu

Data as a Matrix

- In many applications, we can represent data as a matrix: e.g. text analysis, recommendation

Documents



Vector-space representation

	D1	D2	D3	D4	D5
complexity	2		3	2	3
algorithm	3			4	4
entropy	1			2	
traffic		2	3		
network		1	4		

Term-document matrix

Data as a Matrix

- Think of data as $\boldsymbol{A} \in \mathbb{R}^{n\times d}$ containing n row vectors in $\boldsymbol{R}^d$, and typically $n \gg d$

- Some examples of typical web-scale data:

Data	Rows	Columns	n	d	sparse
Textual	Documents	Words	$> 10^{10}$	$10^5 - 10^7$	yes
Visual	Images	Pixels, SIFT	$> 10^8$	$10^5 - 10^6$	no
Audio	Songs	Frequencies	$> 10^8$	$10^5 - 10^6$	no
Machine Learning	Examples	Features	$> 10^6$	$10^2 - 10^4$	yes/no
Financial	Prices	Items, Stocks	$> 10^6$	$10^3 - 10^5$	no

Review: rank-k approximation

- Rank-k approximation to A computes a smaller matrix B of rank k such that B approximates A

Rank-k Approximation

Given $A \in R^{n \times d}$ with $rank(A) = r$, compute a concise matrix B with rank $\mathbf{k} \ll r$ such that it approximates A "accurately".

Review: rank-k approximation

- Rank-k approximation to A computes a smaller matrix B of rank k such that B approximates A

Rank-k Approximation

Given $A \in R^{n \times d}$ with $rank(A) = r$, compute a concise matrix B with rank $\mathbf{k} \ll r$ such that it approximates A "accurately".

- B is much smaller than A that it fits in memory
- Rank(B) << rank(A)
  - If A is a document-term matrix with 10 billion documents and 1 million words $\boldsymbol{A} \in \mathbb{R}^{\mathbf{10^{10}} \times \mathbf{10^{6}}}$ then B would probably be $\boldsymbol{B} \in \mathbb{R}^{\mathbf{1000} \times \mathbf{106}}$

Review: rank-k approximation

- Rank-k approximation to A computes a smaller matrix B of rank k such that B approximates A

Rank-k Approximation

Given $A \in R^{n \times d}$ with $rank(A) = r$, compute a concise matrix B with rank $\mathbf{k} \ll r$ such that it approximates A "accurately".

Review: rank-k approximation

- Rank-k approximation to A computes a smaller matrix B of rank k such that B approximates A

Rank-k Approximation

Given $A \in R^{n \times d}$ with $rank(A) = r$, compute a concise matrix B with rank $\mathbf{k} \ll r$ such that it approximates A "accurately".

- Error difference between A and B is small:

Review: rank-k approximation

- Rank-k approximation to A computes a smaller matrix B of rank k such that B approximates A

Rank-k Approximation

Given $A \in R^{n \times d}$ with $rank(A) = r$, compute a concise matrix B with rank $\mathbf{k} \ll r$ such that it approximates A "accurately".

- Error difference between A and B is small:
  - The covariance error $||A^T A - BTB||_{2,F}$ is small

Review: rank-k approximation

- Rank-k approximation to A computes a smaller matrix B of rank k such that B approximates A

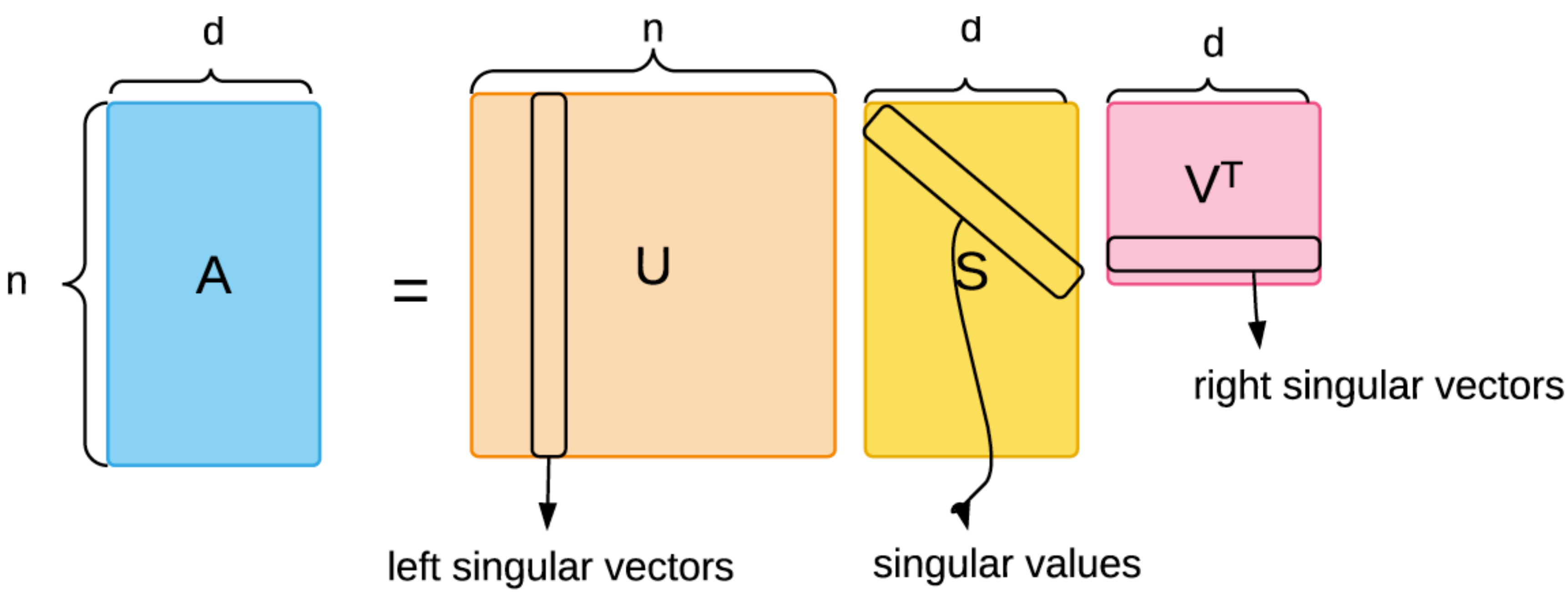
Rank-k Approximation

Given $A \in R^{n \times d}$ with $rank(A) = r$, compute a concise matrix B with rank $\mathbf{k} \ll r$ such that it approximates A "accurately".

- Error difference between A and B is small:
  - The covariance error $\|A^T A - BTB\|_{2,F}$ is small
  - The projection error $\|A - \Pi_B(A)\|_{2,F}$ is small
    - $\Pi_B A$:= projecting rows of A onto the subspace of B
    - If B = USV^T then, the subspace of B is VV^T
    - Therefore $\Pi_B A$ = AVV^T

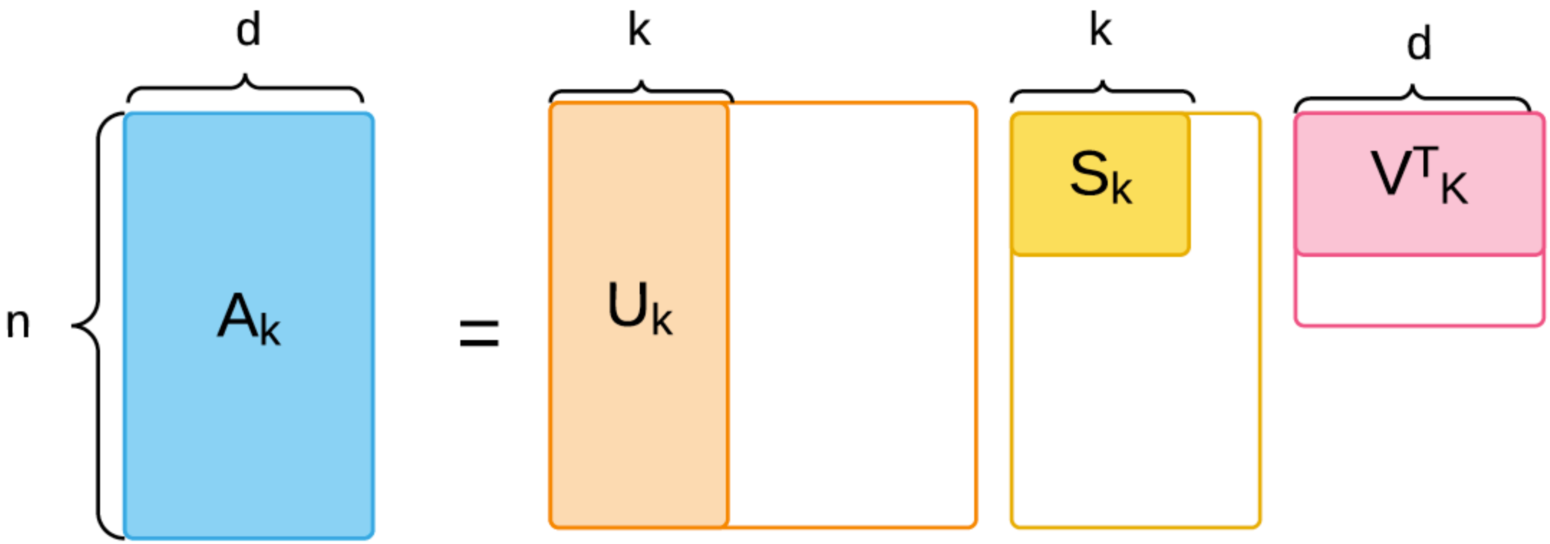
Best Rank-k Approximation

- We saw that SVD computes the **best** rank-k approximation to A



Best Rank-k Approximation

- SVD computes the **best** rank-k approximation



$$A_k = \arg \min_{\text{rank}(B) \leq k} ||A - B||_{F,2}$$

- So the desirable approximation error is

$$||A - \Pi_B(A)||_{2,F} \leq c\,||A - A_k||_{2,F}$$ or $$||A^T A - BTB||_{2,F} \leq c||A - A_k||_{2,F}$$

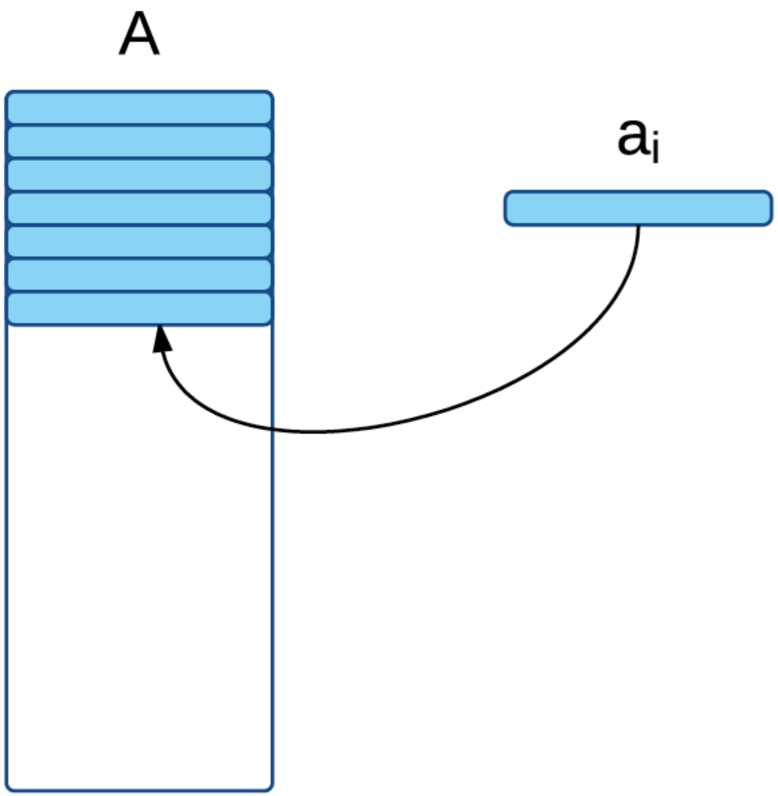
Best Rank-k Approximation

- SVD computes the **best** rank-k approximation to A

- SVD requires $O(nd^2)$ time and $O(nd)$ space
- Not applicable in streaming, or distributed settings
- Not efficient for sparse matrices

Rank-k approximation in stream

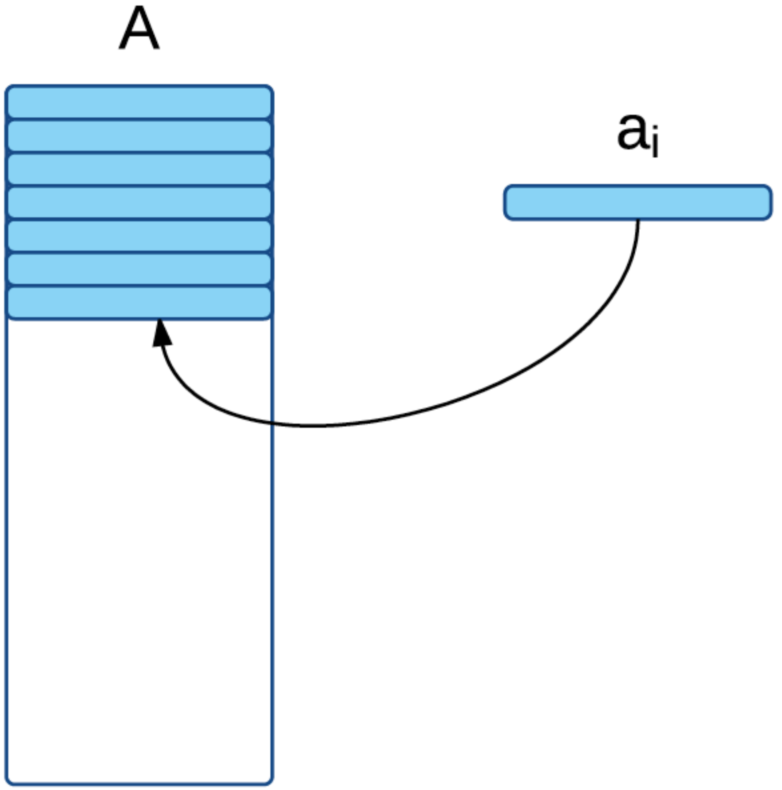
- Can we compute **rank-k approximation** in streaming setting?



Streaming matrix sketching

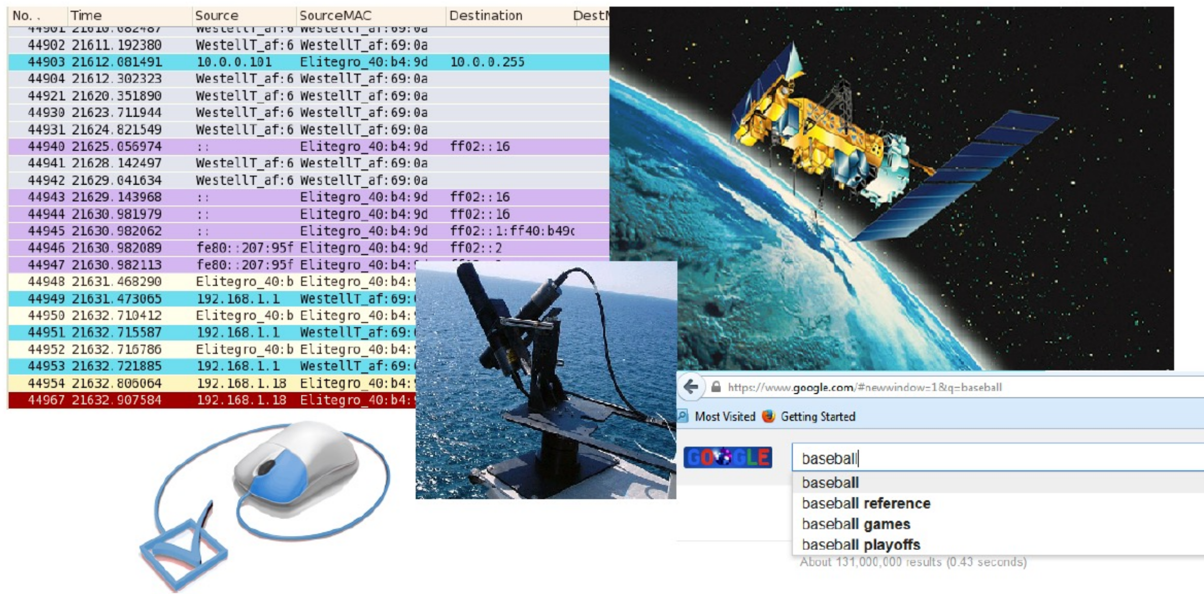
Streaming data matrix

- Every element of the stream is a row vector of fixed ***d***-dimension.
  - We'd like to process A in one pass and using a small amount of memory (sublinear in n)



Streaming data matrix

- Streaming data such as any time series data:
  - ecommerce purchases
  - Traffic sensors
  - Activity logs



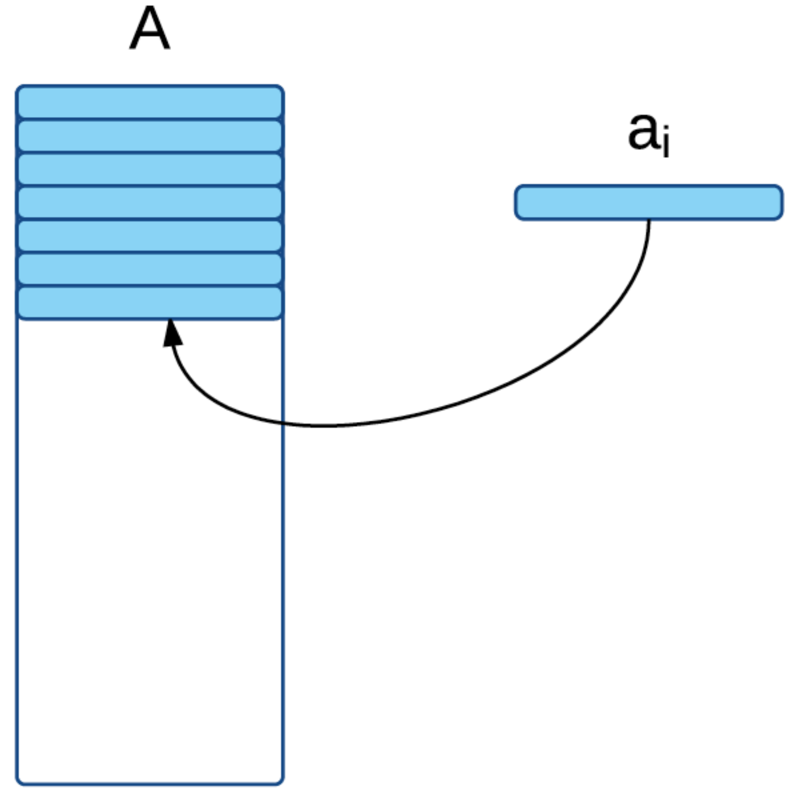
- We can not store the entire data

Application of rank-k approximations

- A large set of data analysis tasks rely on obtaining a **low rank approximation:**
  - Dimension reduction
  - Anomaly detection
  - Data denoising
  - Clustering
  - Recommendation systems

Sketch of a Streaming Matrix

- B is a **sketch** of a streaming matrix A iff
  - B is of a fixed small size that fts in memory
  - At any point in stream, B approximates A



Matrix Sketching Methods

- Almost any matrix sketching methods in streaming setting falls into one of these categories:

1. Row sampling based
2. Random projection based and Hashing
3. Iterative sketching

Row Sampling Methods

Row Sampling Methods

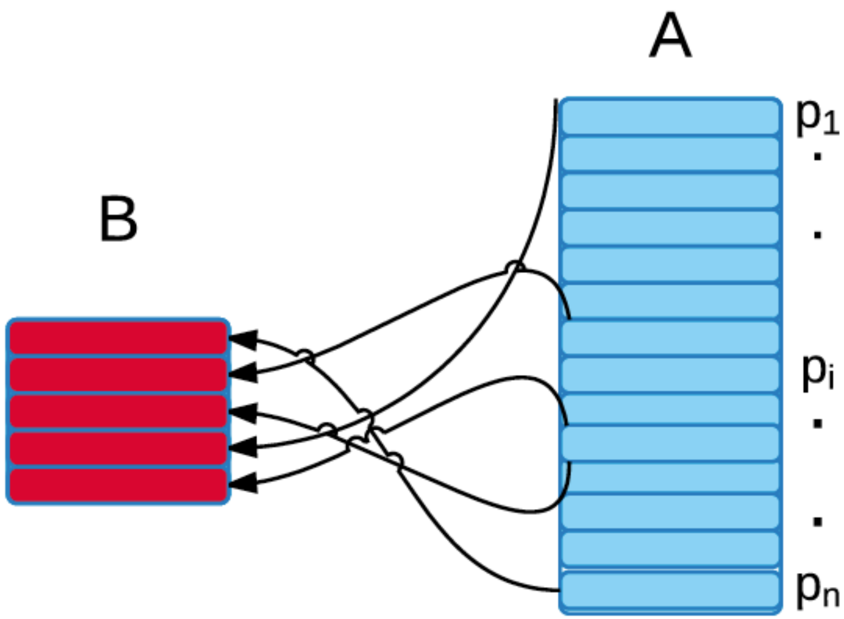
- They select a subset of "important" rows
  - Sample w.r.t a well-defined probability distribution
  - Often sampling is done with replacement

- Methods differ in how they define "importance"

Row Sampling Methods

They construct sketch B by:

- assign a probability p_i to each row a_i
- sample l rows from A to construct B
- rescale B appropriately to make it unbiased



Intuition: Row Sampling Methods

- An Intuitive way to define "importance" of an item:
  - the weight associated to the item, e.g.
    - file records → weights as size of the file,
    - IP addresses → weights as number of times the IP address makes a request
- **why it is necessary to sample important items?**
  - Consider a set of weighted items $S = \{(a_1, w_1), (a_2, w_2), \cdots, (a_n, w_n)\}$ that we want to summarize with a *small* & *representative* sample.
  - We define a *representative* sample as the one estimates total weight of S (i.e. $W_s = \sum_{i=1} w_i$) in expectation.

Intuition: Row Sampling Methods

- This is achievable with a sample set of size one!
  - Sample any item ***(a_j , w_j)*** with an arbitrary fixed probability ***p***, and rescale its weight to ***W_s/p***.
  - Then ***E[weight of the sample] = p. W_s/p = W_s***

- High variance issue:
  - To lower down the variance, (1) sample heavy items (i.e. important items) with higher prob., and (2) sample more items
  - So sample item ***a_j*** with prob. ***p=w_j/W_s*** and rescale it to ***W_s/p***
  - If we sample l items, then rescale items to rescale it to $W_s/(lp)$

FAST MONTE CARLO ALGORITHMS FOR MATRICES I: APPROXIMATING MATRIX MULTIPLICATION, P. Drineas, etal, 2006

Row Sampling algorithms

- In matrices,
  - Each item a_j is a row vector
  - Each weight $w_j = \|a_j\|^2$
  - And $\sum_{j=1}^{n} \|a_j\|^2 = \|A\|_F^2$

- Row sampling algorithm based on L2 norm:
  - Let sample size = l, i.e. the sketch B is $l \times d$
  - For every row a_i arriving in the stream,
    - Update $\|A\|_F^2$ by adding $\|a_j\|_F^2$
    - Compute its sampling probability $p_i = \|a_i\|^2 / \|A\|_F^2$
    - Sample it l times (one for each row of B. If it is sampled, replace the corresponding row in B with a_i)
    - Rescale a_i where it is sampled by $1/\sqrt{l\, p_i}$

This is the Frobenius norm of all rows seen so far

Row Sampling algorithms

- We can show that

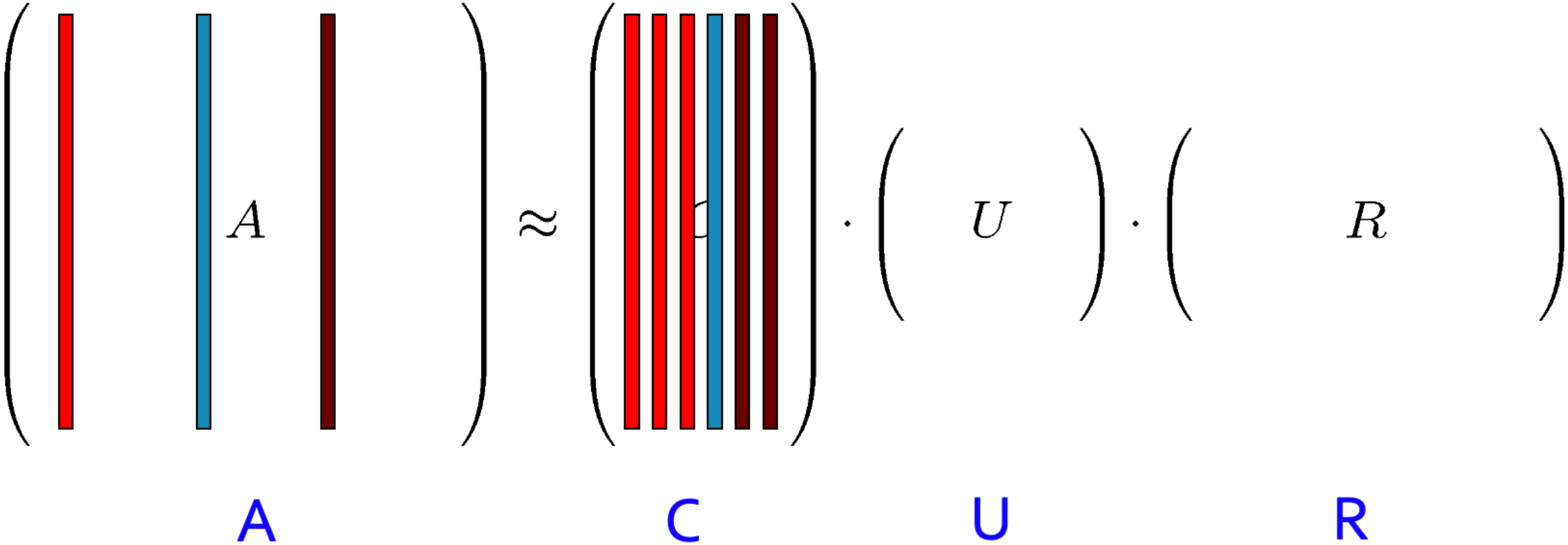
$$E[\|B\|_F] = \|A\|_F$$

- If we sample $\ell = O(k/\varepsilon^2)$ rows, then:

$$\|A - \pi_B(A)\|_F^2 \leq \|A - A_k\|_F^2 + \varepsilon\|A\|_F^2$$

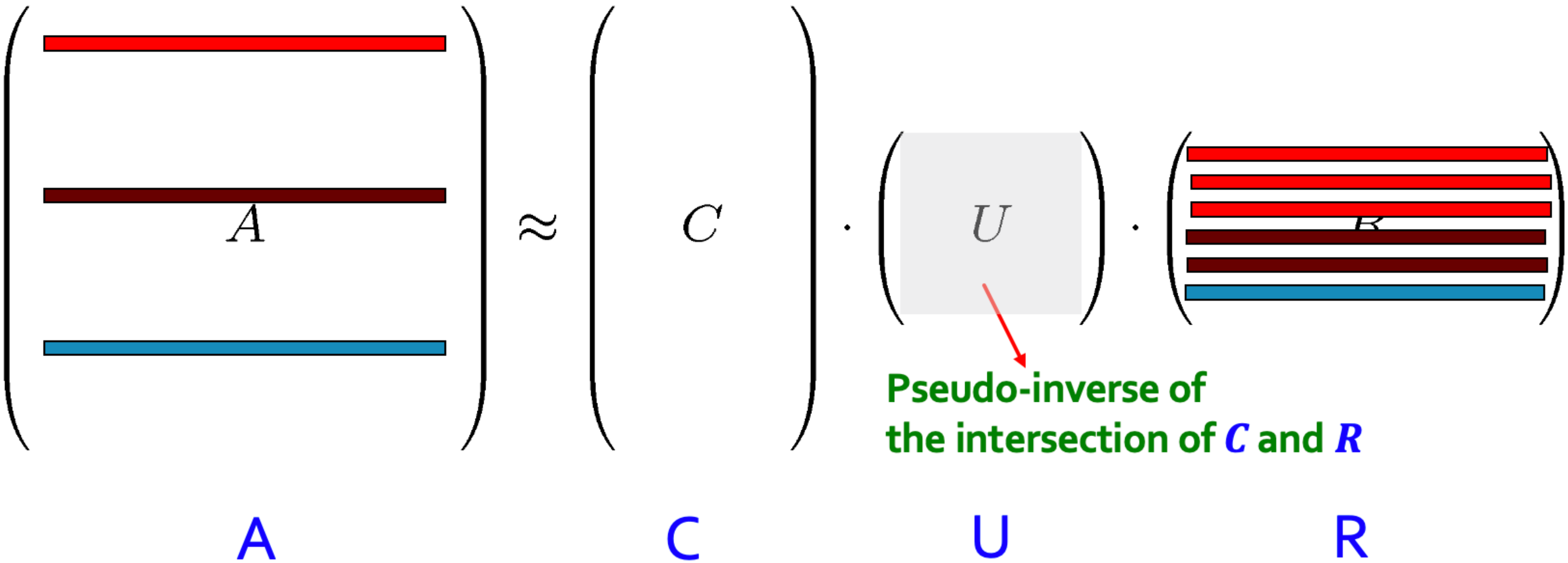
CUR: Row/column sampling

- Row sampling based on L2 norm:
  - CUR method: samples rows/columns with probability = squared norm of rows/columns



CUR: Row/column sampling

- Row sampling based on L2 norm:
  - CUR method: samples rows/columns with probability = squared norm of rows/columns



CUR: Row/column sampling

- Row sampling based on L2 norm:
  - CUR method: samples rows/columns with probability = squared norm of rows/columns
  - Error guarantee: If we sample $\boldsymbol{c} = \ \boldsymbol{O}\left(\frac{\boldsymbol{k\ log\ k}}{\boldsymbol{\varepsilon^2}}\right)$ columns and $\boldsymbol{r}= \ \boldsymbol{O}\left(\frac{\boldsymbol{k\ log\ k}}{\boldsymbol{\varepsilon^2}}\right)$ rows, then

CUR error SVD error

$$||A - CUR||_F \leq (2 + \varepsilon)||A - A_K||_F$$

With probability >= 98%

Row Sampling Methods

+ **Easy interpretation of basis**
  - Since the basis vectors are actual rows/columns

+ **Suitable for Sparse data**
  - Since the basis vectors are actual rows/columns

− **Duplicate columns and rows**
  - Columns of large norms will be sampled multiple times

Random Projection Methods

Random Projection Methods

- **Key idea**: if points in a vector space are projected onto a randomly selected subspace of suitably high dimension, then the distances between points are approximately preserved

- **Johnson-Lindenstrauss Transform (JLT): *d*** datapoints in any dimension ($\mathbb{R}^n$ for $n \gg d$)can get embedded into roughly ***log d*** dimensional space, such that their pair-wise distances are preserved to some extent

Johnson-Lindenstrauss Transform

We define JLT more precisely:

- A random matrix $S \in \mathbb{R}^{r \times n}$ has **JLT** property if for all vectors $v, v' \in \mathbb{R}^n$,

  $$\|\boldsymbol{Sv} - \boldsymbol{Sv'}\|^2 = (\boldsymbol{1} \pm \boldsymbol{\epsilon})\|\boldsymbol{v} - \boldsymbol{v'})\|^2$$

  with probability at least $1 - \delta$

- There are many ways to construct a matrix S that preserve pair-wise distances.
  - All such matrices are called to have the Johnson-Lindenstrauss Transform (JLT) property

How to construct a JLT matrix

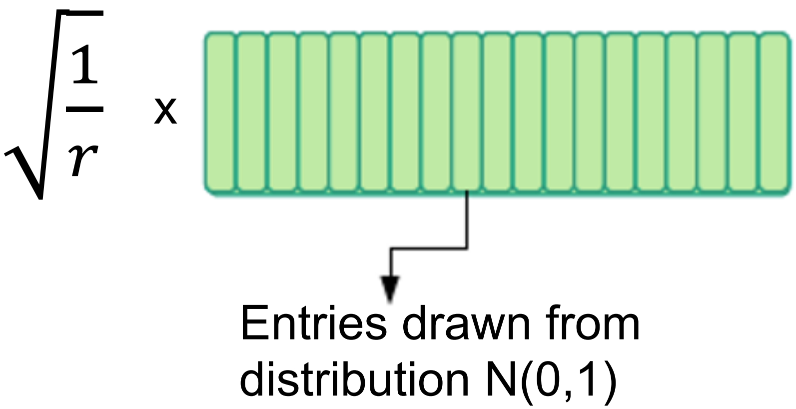
One simple construction of S:

- Pick matrix $S \in \mathbb{R}^{r \times n}$ as an orthogonal projection on a random r-dimensional subspace of $\mathbb{R}^n$ with $r = O(\epsilon^{-2} \log d)$
  - Rows of S are orthogonal vectors
- Then for any matrix $A \in \mathbb{R}^{n \times d}$, SA preserves pair-wise distances between d datapoints in A

How to construct a JLT matrix

- A simpler construction for $S \in \mathbb{R}^{r \times n}$ is:
  - to have entries as independent random variables with the standard normal distribution

$$S = \sqrt{\frac{1}{r}}\ [\text{matrix with entries draw from } N(0,1)]$$

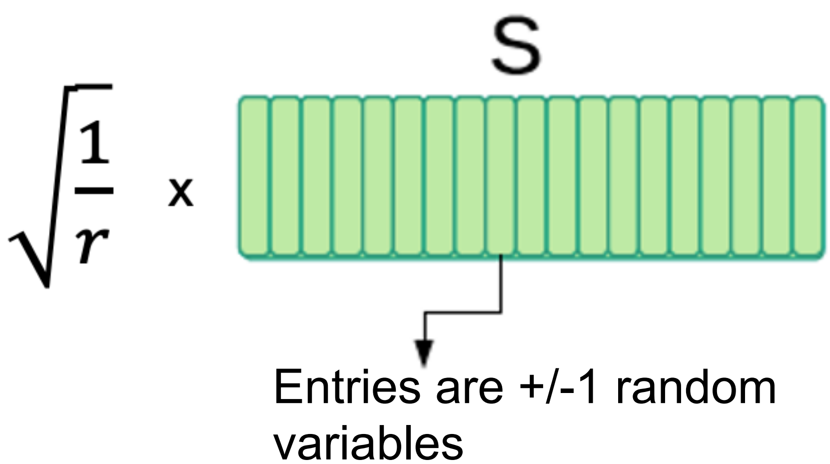


How to construct a JLT matrix

- Another construction for $S \in \mathbb{R}^{r \times n}$ is:

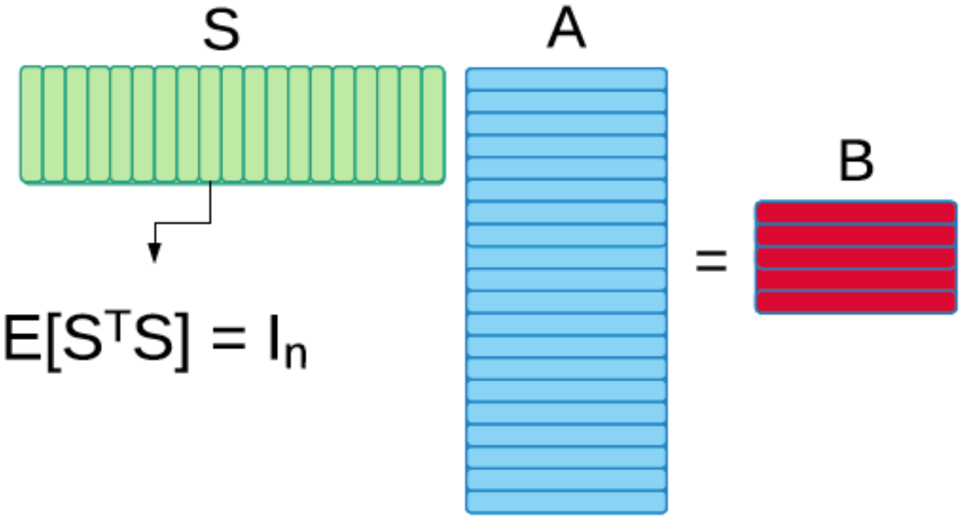
$$S = \sqrt{\frac{1}{r}}\ [\text{entries as independent +/-1 random var}]$$

This is computationally simpler to construct



Random Projection Methods

- They use a JLT matrix $S \in \mathbb{R}^{r \times n}$
- Construct the sketch as $B = SA \in \mathbb{R}^{r \times d}$
  - this projects datapoints from a high-dim space $\mathbb{R}^n$ onto a lower-dim subspace $\mathbb{R}^r$
- They show $\mathbb{E}[B^T B] = A^T \mathbb{E}[S^T S] A = A^T A$



Random Projection Methods

- Depending on JLT construction, we achieve different error bounds:

  - If $S \in \mathbb{R}^{r \times n}$ has has iid zero-mean ± 1 entries and $r = O(\frac{k}{\varepsilon} + k \log k)$ and, then

$$\|A - \pi_{SA}(A)\|_F \leq (1 + \varepsilon)\|A - A_k\|_F$$

Random Projection Methods

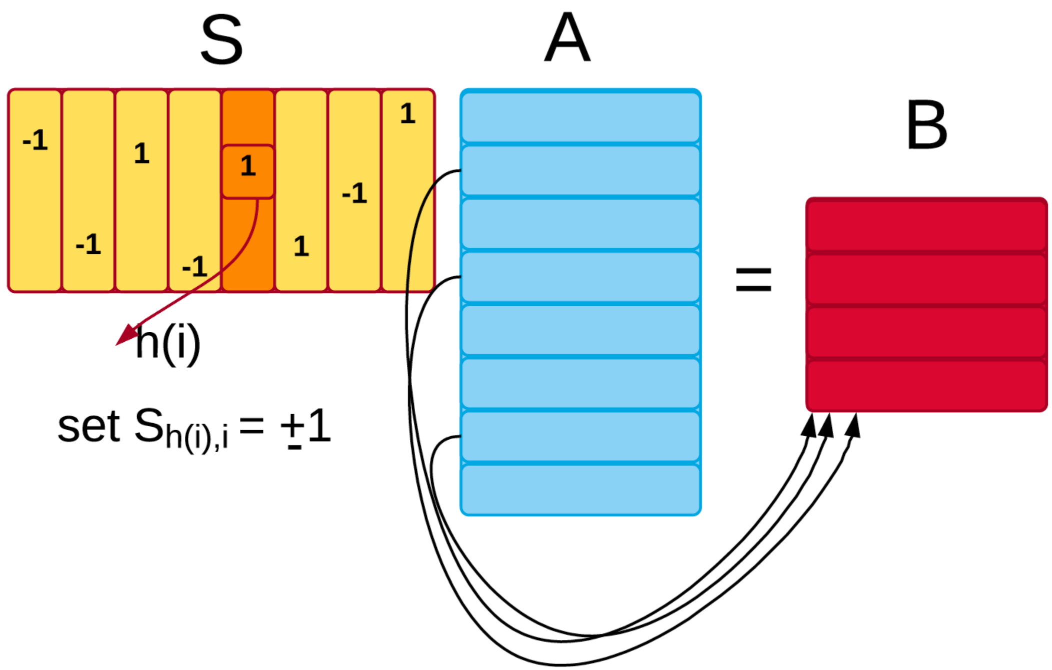
- Computationally efficient
- Sufficiently accurate in practice
- A great pre-processing step in applications

- **Data-oblivious** as their computation involves only a random matrix S
  - Compare to row sampling methods that need to access data to form a sketch

Matrix Hashing Techniques

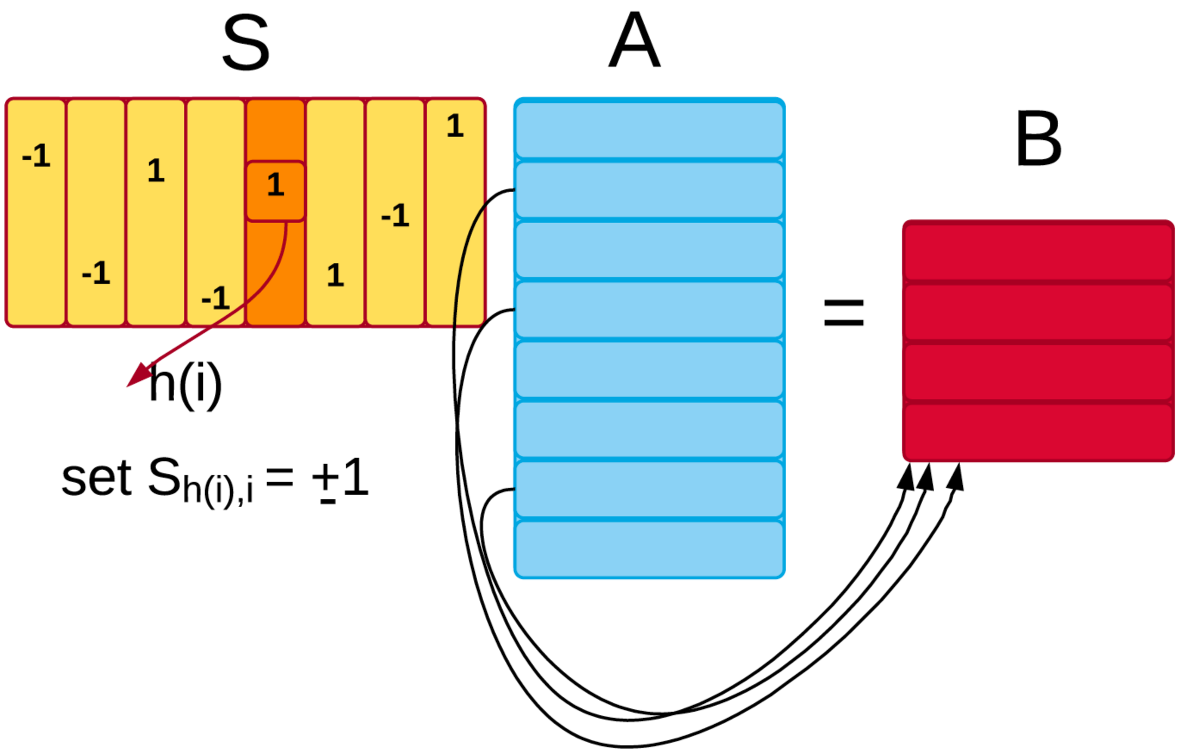
- Use matrix S that contains one ± 1 per column

Only one non-zero entry in each column of S.
The rest of entries are zero



- To build S, use two hash functions:
  - h: [n] → [r] , and g:[n] → {-1, +1}

Matrix Hashing Techniques

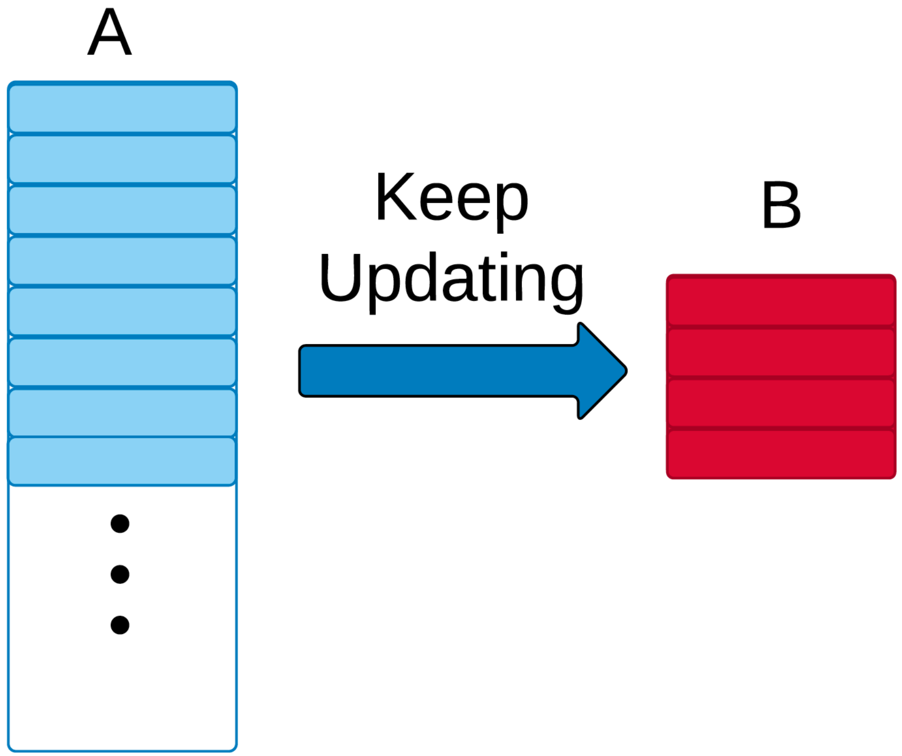


- Very efficient for sparse matrices A
  - can be applied in O(nnz(A)) operations
  - nnz(A) = number of non-zeros of A

Iterative Sketching

Iterative Sketching

- They work over a stream $A =< a_1, a_2, \ldots, a_n >$
- each a_i is read once, get processed quickly and not read again
- with only a small amount of memory available



Iterative Sketching

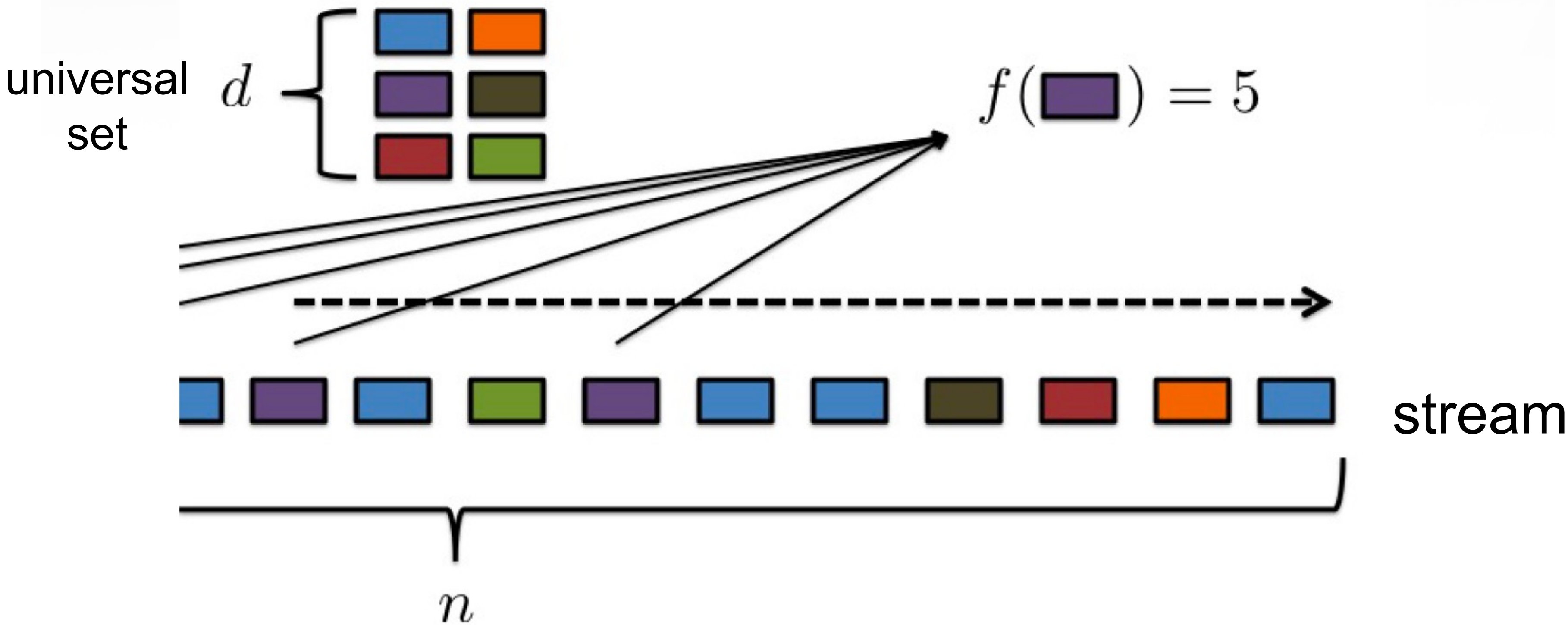
- State of the art method in this group is called "Frequent Directions"

- It is based on Misra-Gries algorithm for finding frequent items in a data stream

- We first see how Misra-Gries algorithm for finding frequent items work
  - Then we extend it to matrices

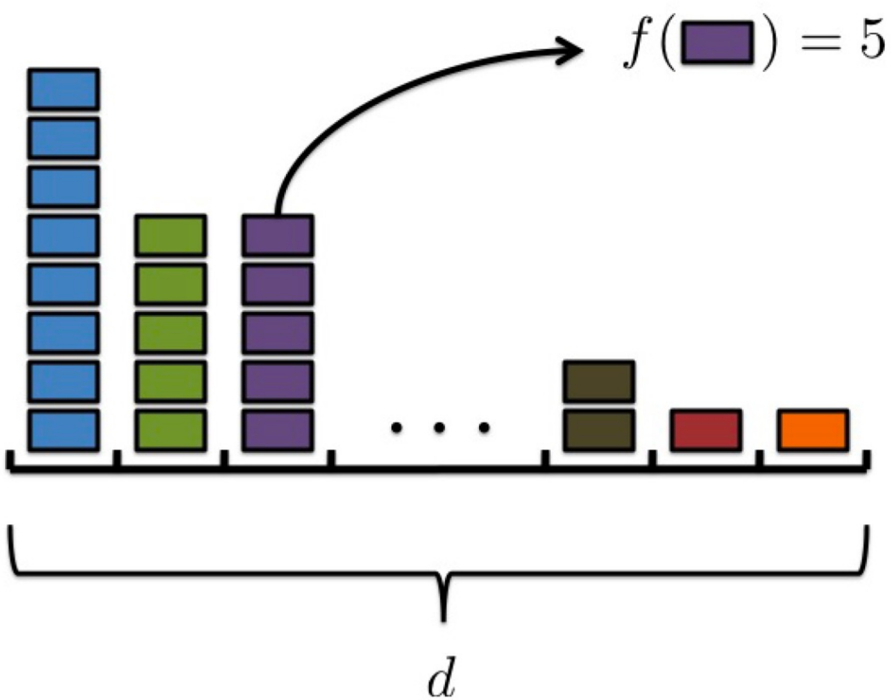
Frequent Items: Misra-Gries

- Suppose there is a stream of items, and we want to find frequency f(i) of each item



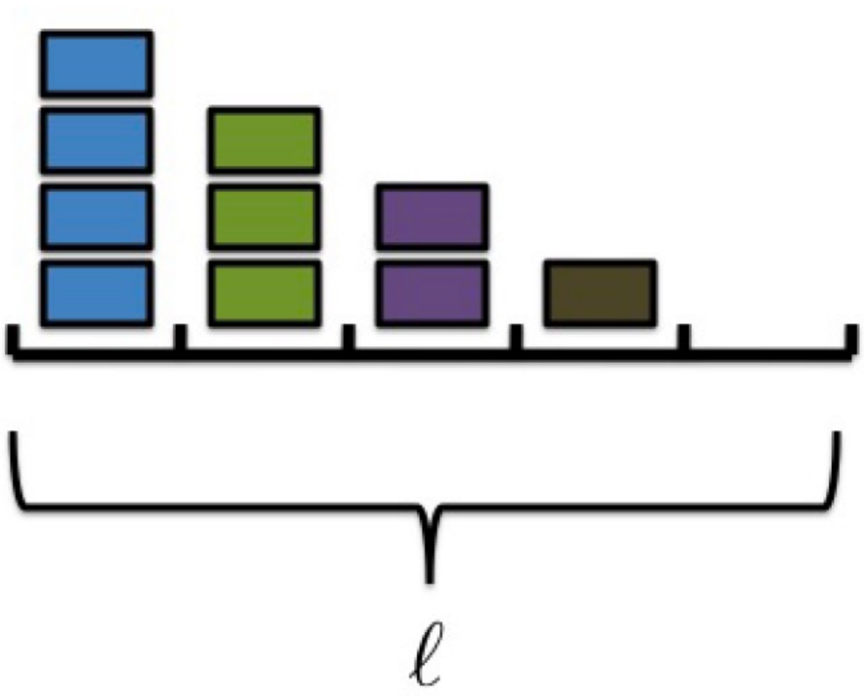
Frequent Items: Misra-Gries

- If we keep ***d*** counters, we can count frequency of every item...
  - But it's not good enough (IP addresses, queries,...)



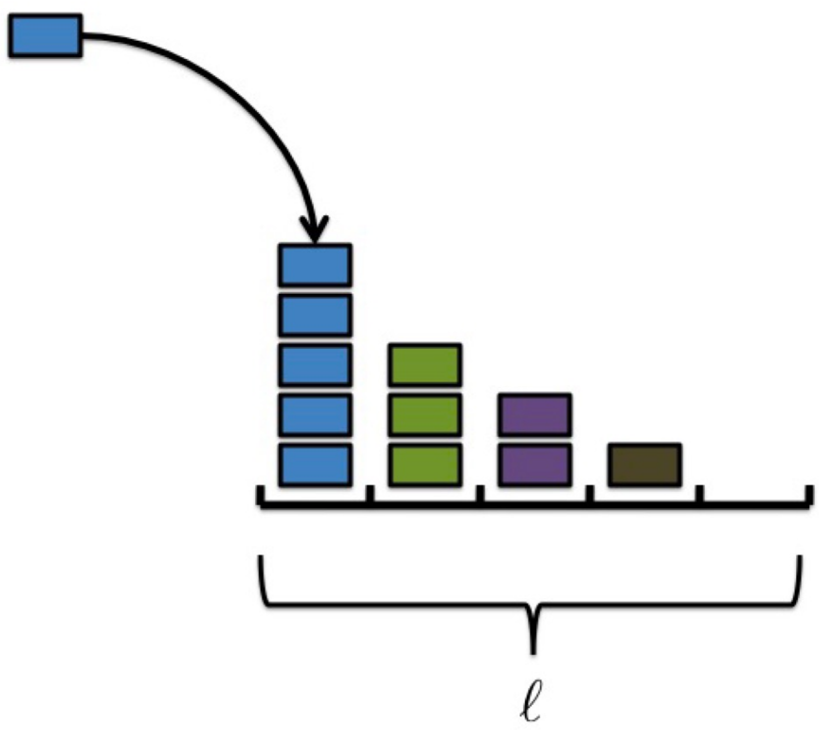
Frequent Items: Misra-Gries

- Let's keep l counters where $l \ll d$



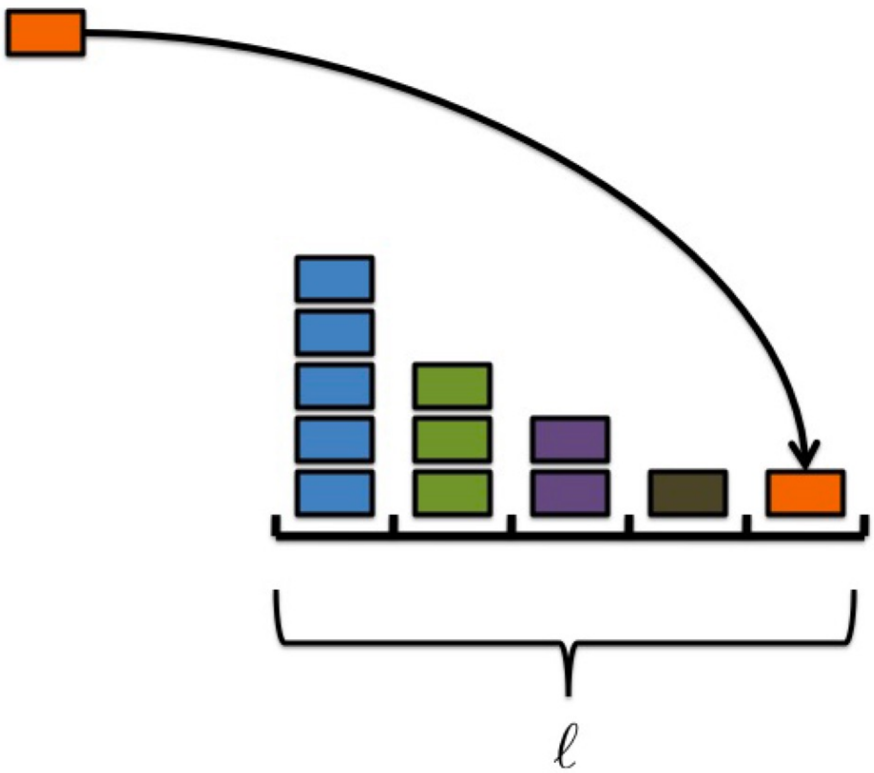
Frequent Items: Misra-Gries

- If a new item arrives in the stream that is already in the counters, we add 1 to its count



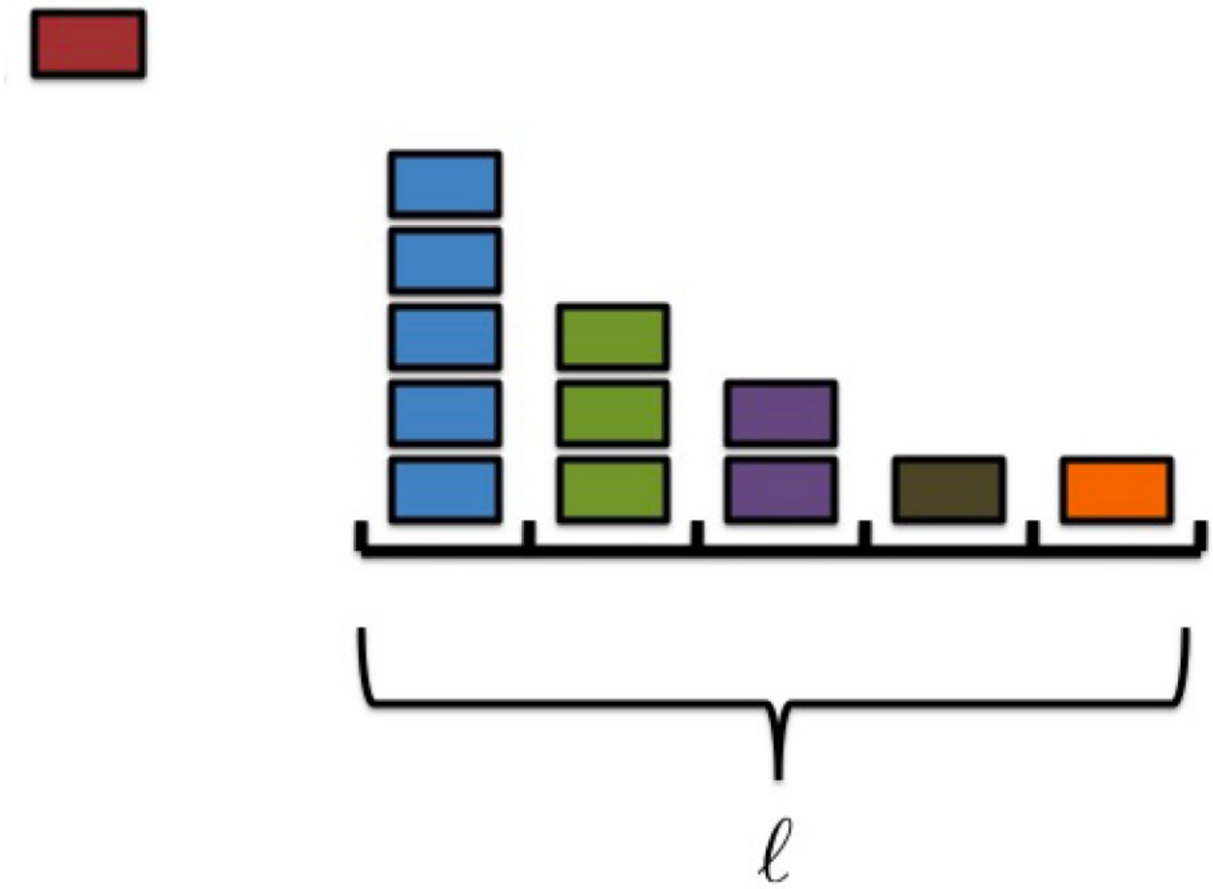
Frequent Items: Misra-Gries

- If the new item is not in the counters and we have space, we create a counter for it and set it to 1



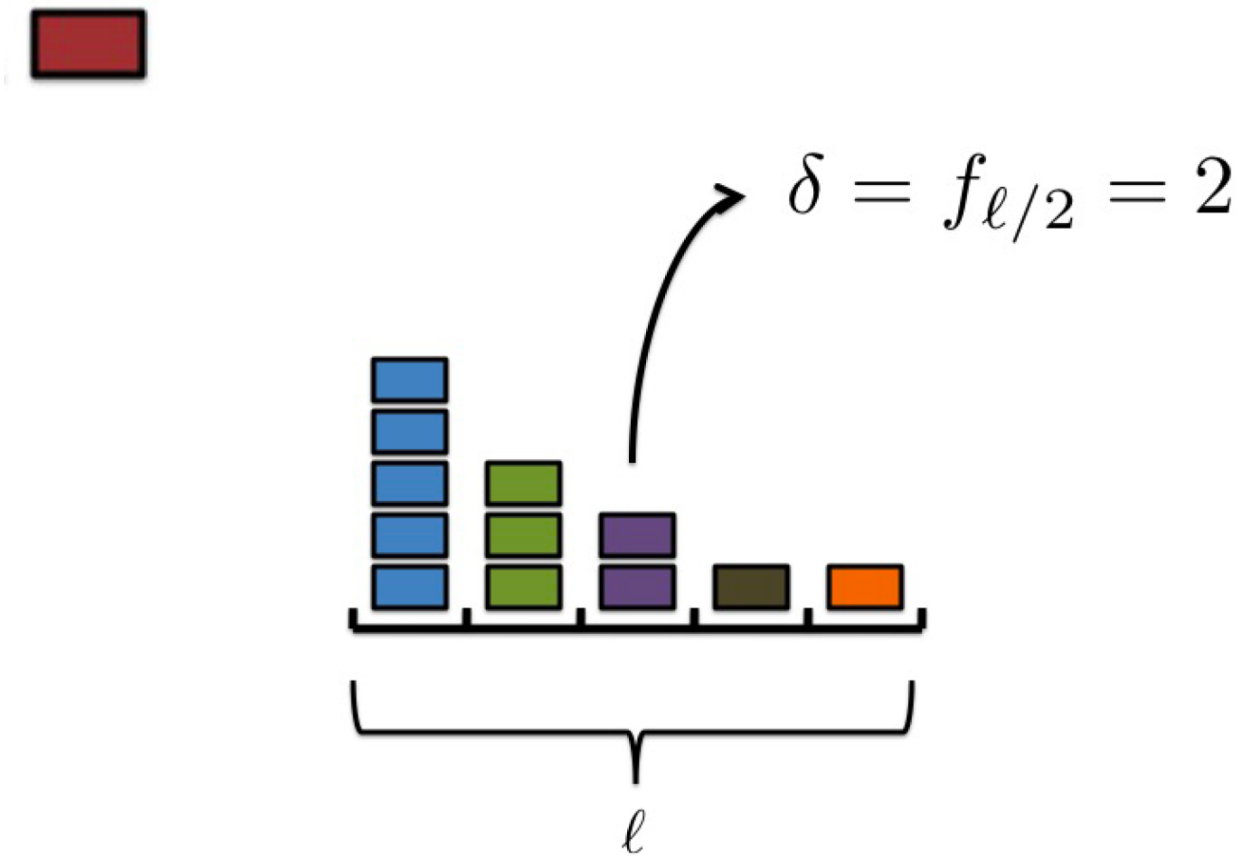
Frequent Items: Misra-Gries

- But what if we don't have space for it?



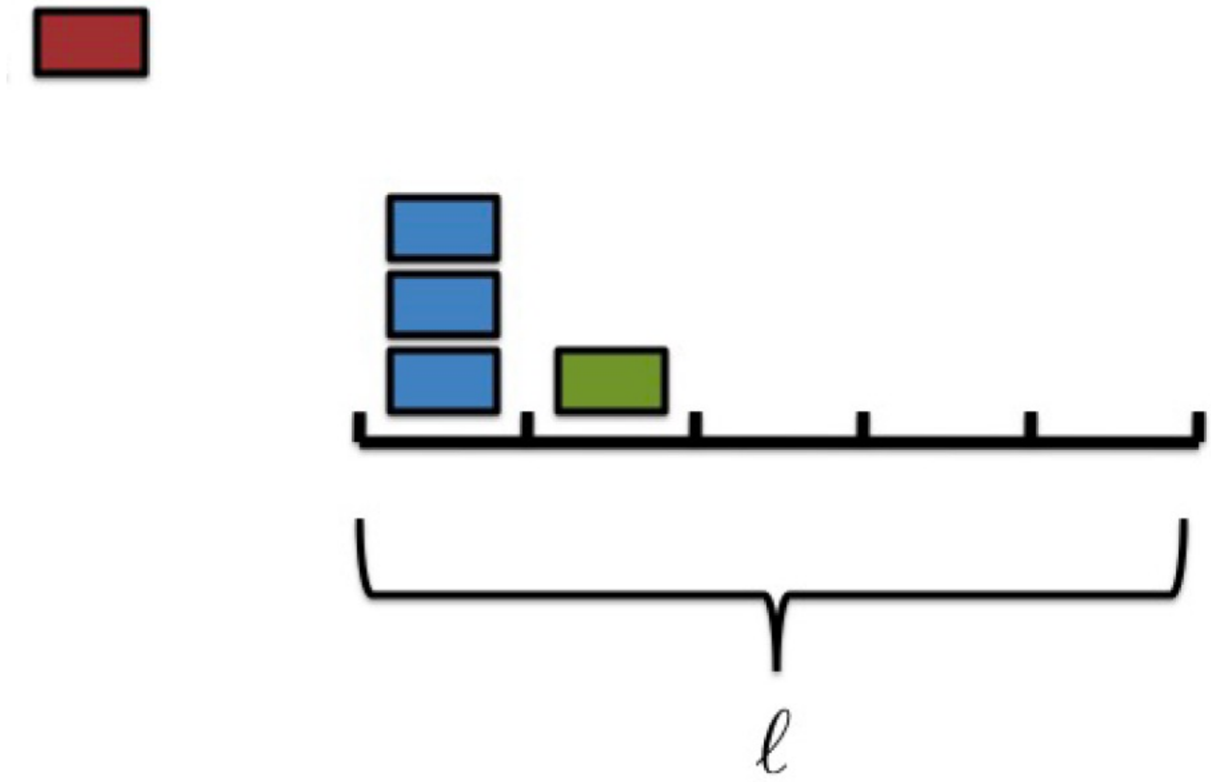
Frequent Items: Misra-Gries

- Let δ be the median counter at time t



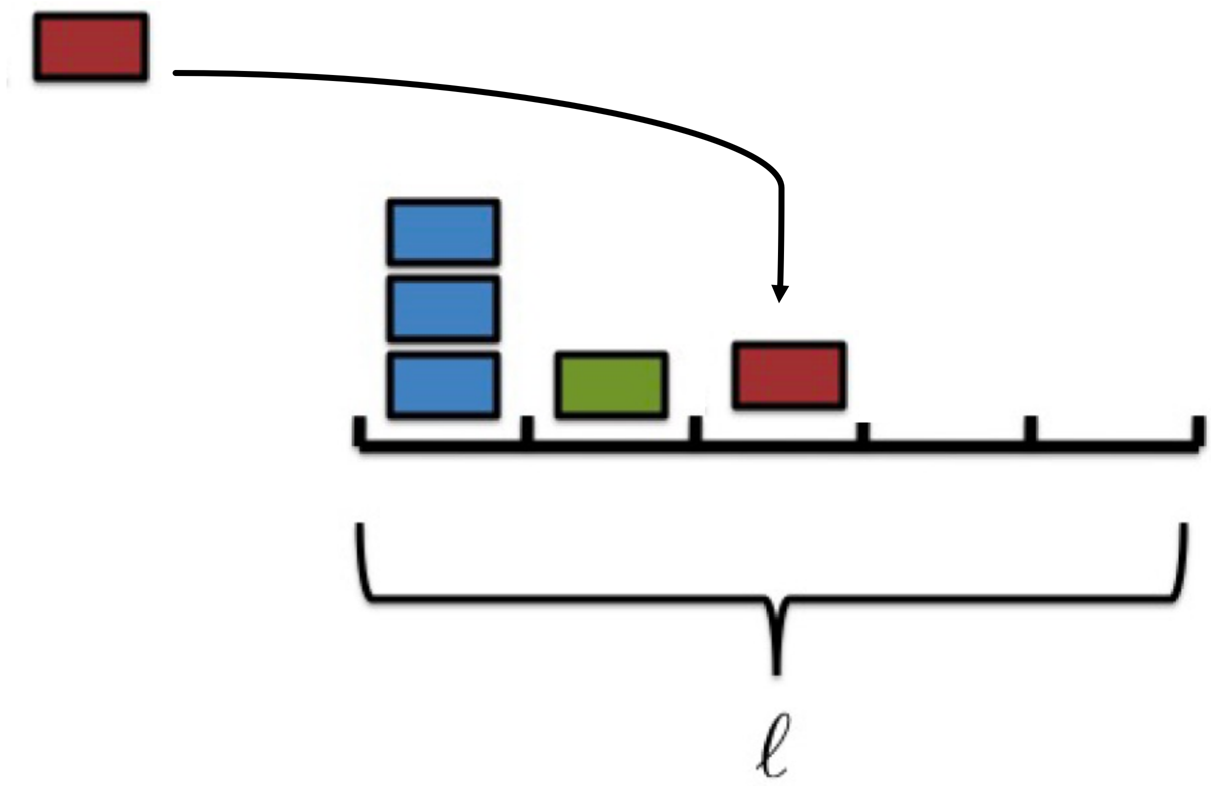
Frequent Items: Misra-Gries

- Decrease all counts by δ (set it to 0 if less than δ)



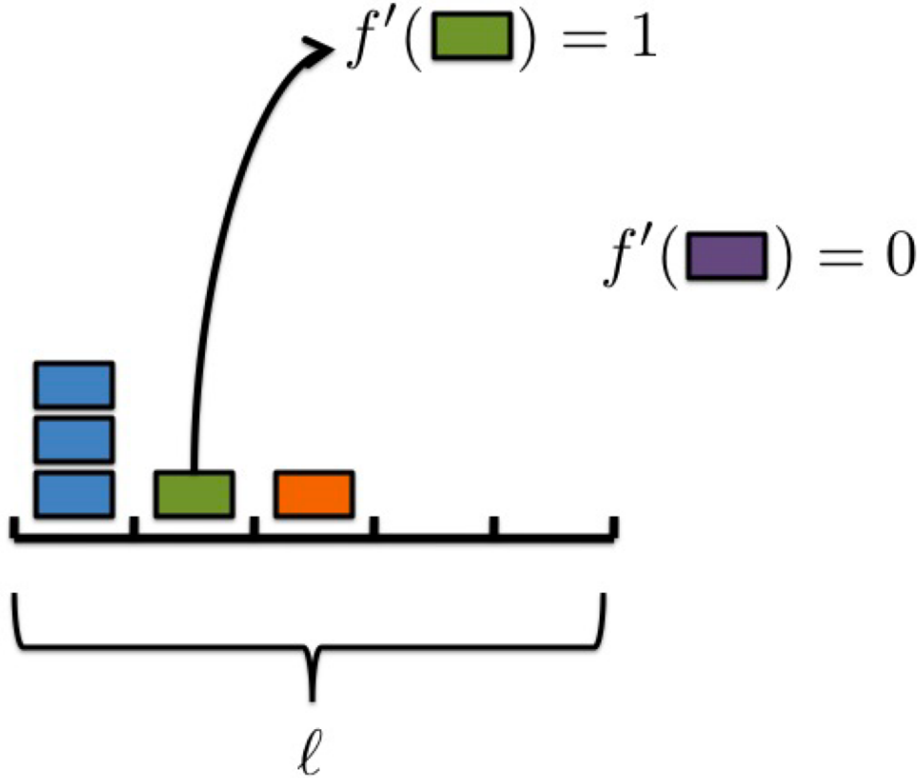
Frequent Items: Misra-Gries

- Now we have space for new item, so we continue...



Frequent Items: Misra-Gries

- At any time in the stream, the approximated counts for items are what we have kept so far



Frequent Items: Misra-Gries

- This method undercounts so for any item i
$$0 \leq f'(i) \leq f(i)$$
- We decrease each count by at most δ_t
$$f'(i) \geq f(i) - \sum \delta_t$$
- At any point that we have seen ***n*** elements in stream: $$\frac{l}{2} \sum \delta_t \leq n$$
- The error guarantee: $\boldsymbol{0 \leq f(i) - f'(i) \leq 2n/l}$

Frequent Items: Misra-Gries

- Misra-Gries produces a non-zero approximated frequency $f'(i)$ for all items that their true frequency $f(i) > 2n/l$

$$f(i) - 2n/l \leq f'(i)$$

- To find items that appear more than 20% of the time i.e. $f(i) > n/5$, take $l = 10$ counters and run Misra-Gries algorithm

Frequent Directions

- Let’s extend it to vectors and matrices
- Stream items are **row vectors** in ***d*** dimension
- At any time ***n*** in the stream, they form a tall matrix $A \in \mathbb{R}^{n \times d}$
- The goal is to find the **most frequent directions** of A

Frequent Directions

Frequent Directions (Lib'13)

Input: $A \in \mathbb{R}^{n \times d}$, and an integer ℓ

$B \leftarrow$ empty matrix $\in \mathbb{R}^{\ell \times d}$

for$(a_i \in A)$

  Insert a_i into B

  if $(B$ is full$)$

    $[U, S, V] \leftarrow \text{svd}(B)$

    $\tilde{S} \leftarrow [\sqrt{S_1^2 - S_{l/2}^2}, \sqrt{S_2^2 - S_{l/2}^2}, \dots 0, \dots, 0]$

  $B \leftarrow \tilde{S} V^T$

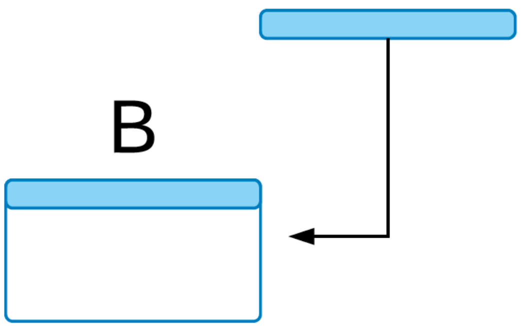
return B

B

Frequent Directions

Frequent Directions (Lib'13)

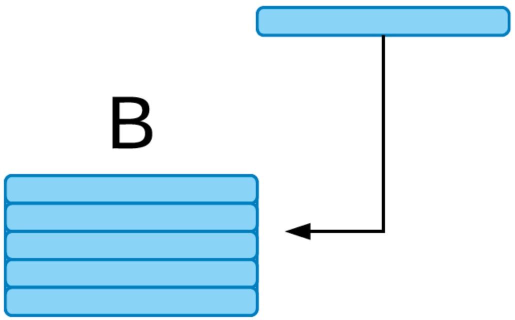
Input: $A \in \mathbb{R}^{n \times d}$, and an integer ℓ
$B \leftarrow$ empty matrix $\in \mathbb{R}^{\ell \times d}$
for$(a_i \in A)$
  Insert a_i into B
  if (B is full)
    $[U, S, V] \leftarrow \mathrm{svd}(B)$
    $\tilde{S} \leftarrow [\sqrt{S_1^2 - S_{l/2}^2}, \sqrt{S_2^2 - S_{l/2}^2} \dots 0, \dots, 0]$
  $B \leftarrow \tilde{S} V^T$
return B



Frequent Directions

Frequent Directions (Lib'13)

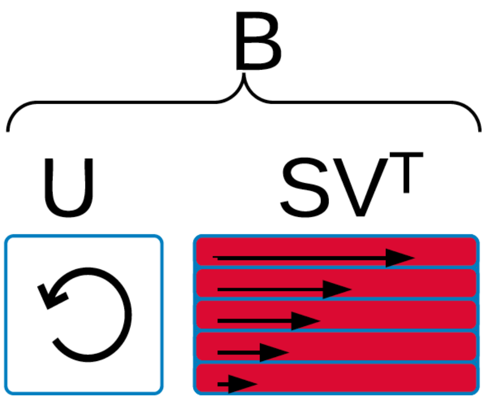
Input: $A \in \mathbb{R}^{n \times d}$, and an integer ℓ
$B \leftarrow$ empty matrix $\in \mathbb{R}^{\ell \times d}$
for$(a_i \in A)$
    Insert a_i into B
    if (B is full)
        $[U, S, V] \leftarrow \mathrm{svd}(B)$
        $\tilde{S} \leftarrow [\sqrt{S_1^2 - S_{l/2}^2}, \sqrt{S_2^2 - S_{l/2}^2} \ldots 0, \ldots, 0]$
    $B \leftarrow \tilde{S} V^T$
return B



Frequent Directions

Frequent Directions (Lib'13)

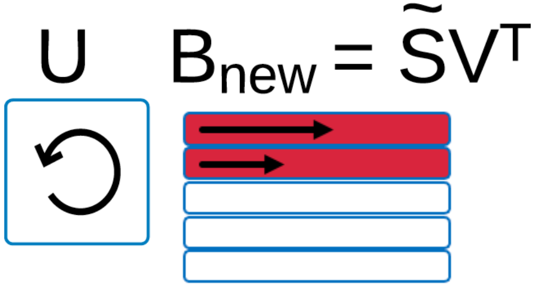
Input: $A \in \mathbb{R}^{n \times d}$, and an integer ℓ
$B \leftarrow$ empty matrix $\in \mathbb{R}^{\ell \times d}$
for$(a_i \in A)$
    Insert a_i into B
    if (B is full)
        $[U, S, V] \leftarrow \text{svd}(B)$
        $\tilde{S} \leftarrow [\sqrt{S_1^2 - S_{l/2}^2}, \sqrt{S_2^2 - S_{l/2}^2} \ldots 0, \ldots, 0]$
    $B \leftarrow \tilde{S} V^T$
return B



Frequent Directions

Frequent Directions (Lib'13)

Input: $A \in \mathbb{R}^{n \times d}$, and an integer ℓ
$B \leftarrow$ empty matrix $\in \mathbb{R}^{\ell \times d}$
for$(a_i \in A)$
  Insert a_i into B
  if (B is full)
    $[U, S, V] \leftarrow \text{svd}(B)$
    $\tilde{S} \leftarrow [\sqrt{S_1^2 - S_{l/2}^2}, \sqrt{S_2^2 - S_{l/2}^2} \ldots 0, \ldots, 0]$
  $B \leftarrow \tilde{S} V^T$
return B



Frequent Directions

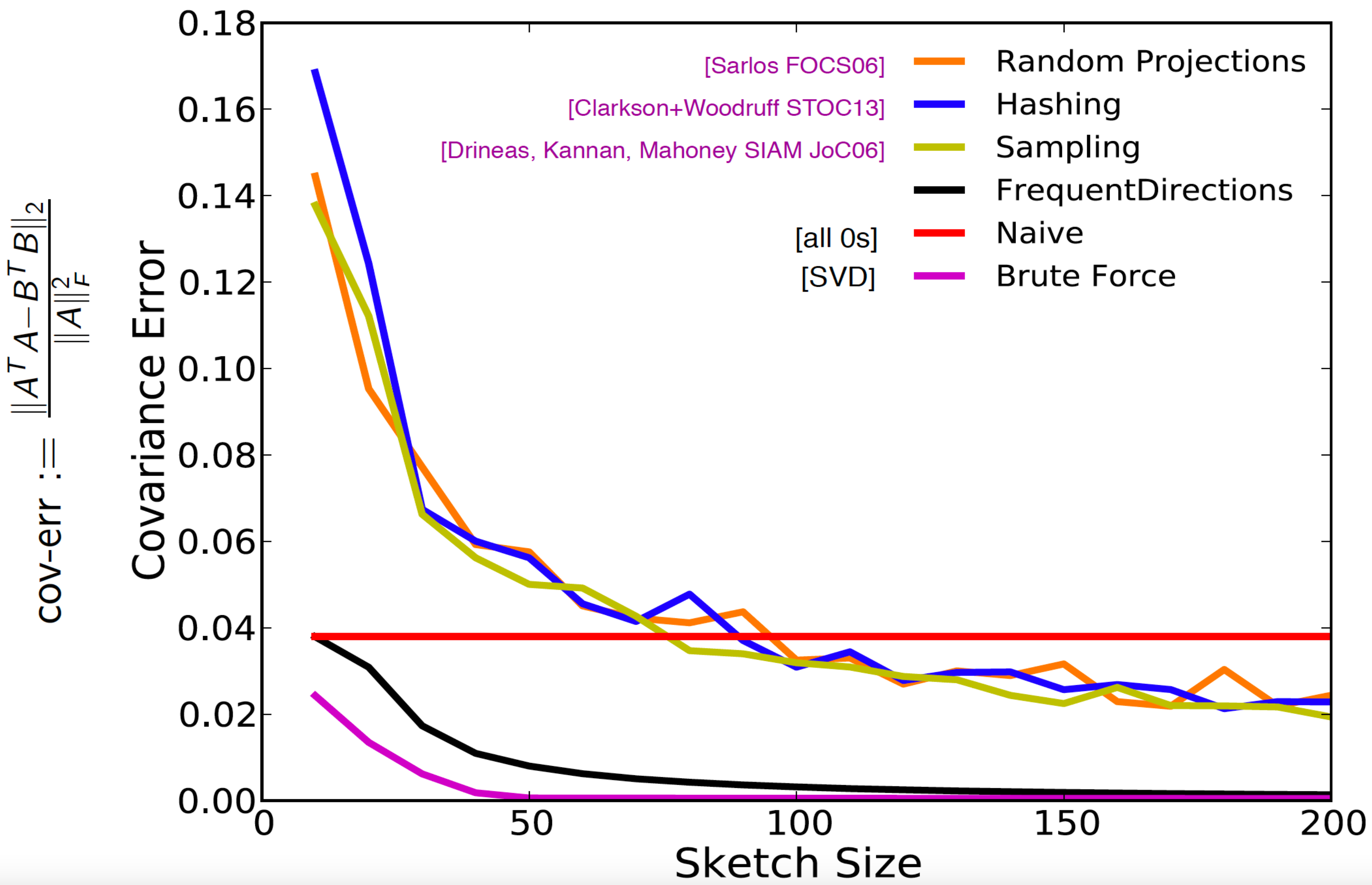
- Similar to the frequent items case, this method has the following error guarantee:

$$\|A^T A - BTB\| \ll \frac{2}{l} \|A\|_F^2$$

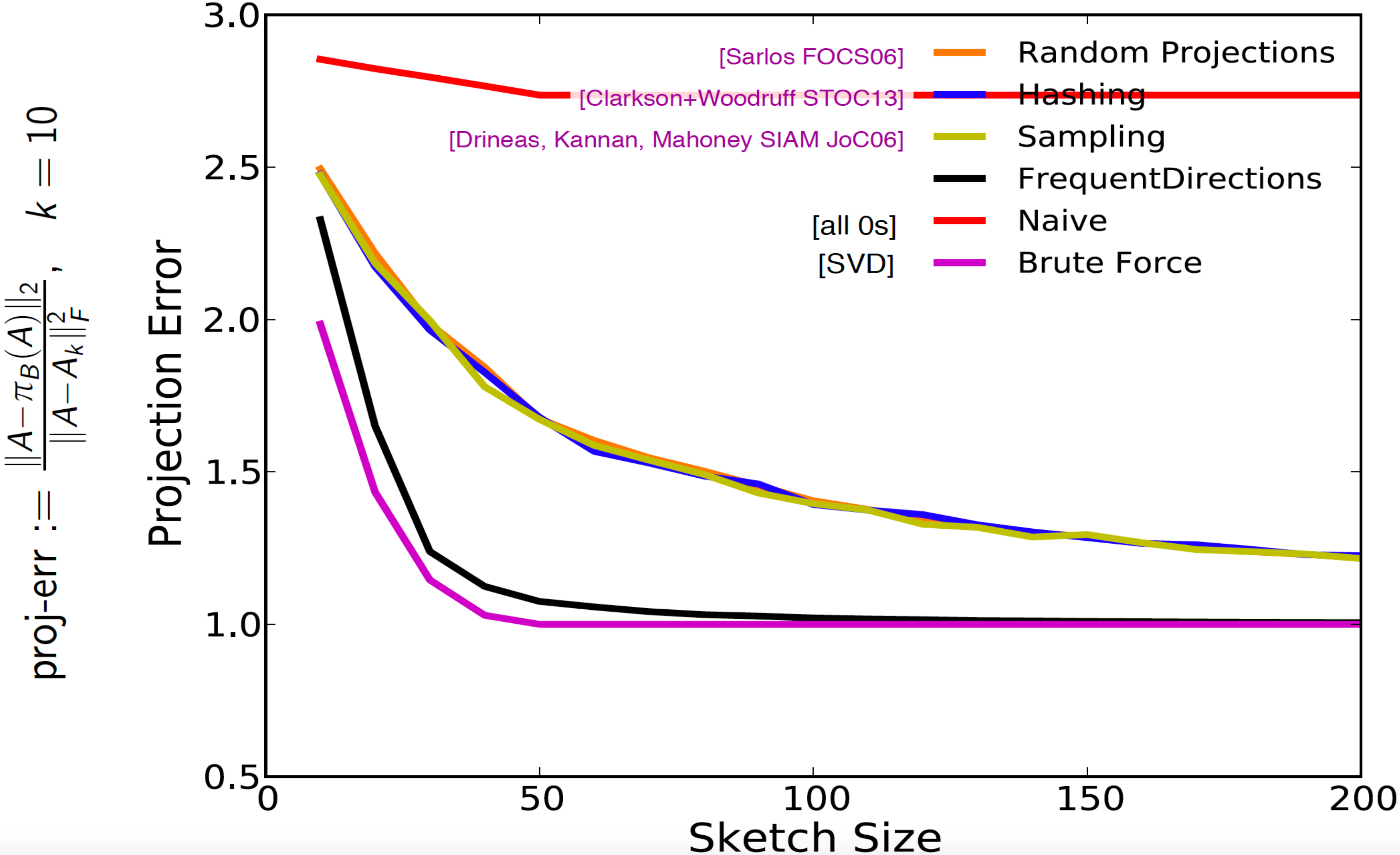
- And if using $l = k + k/\epsilon$

$$\|A - \Pi_B(A)\|_F^2 \ll (1 + \epsilon)\|A - A_k\|_F^2$$

Sketching in Experiment



Sketching in Experiment



Summary

- Matrix Sketching in Streams:
  - Row sampling methods
    - CUR
    - L2 norm based sampling
  - Random projection methods
    - Johnson Lindenstrauss Transform (JLT)
    - Different ways to construct a JLT matrix
  - Iterative sketching methods
    - Misra-Gries algorithm for frequent items
    - Frequent Directions method (state of the art)