

Randomization methods for big-data (other than SGD)

Stephen Becker, CU Applied Math dept

CU Computer Science Department Colloquium, Oct 26 2023

Joint work with several author, mostly my former PhD students:

(and colleagues at CU & the Colorado School of Mines)



Farhad Pourkamali-Anaraki
(Asst. Prof at UC Denver)



David Kozak
(private sector)

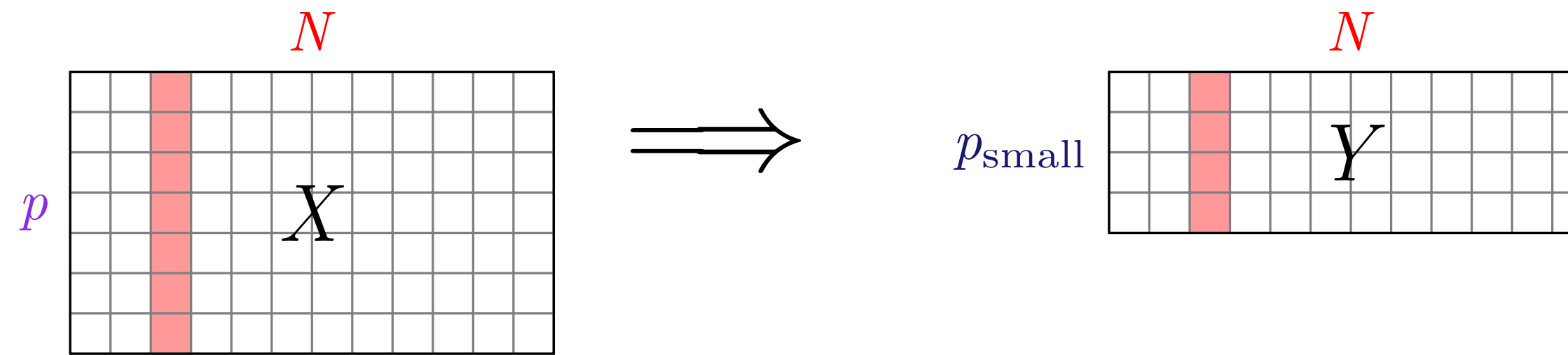


Osman Asif Malik
(Alvarez postdoc at Berkeley Labs)

OUTLINE

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

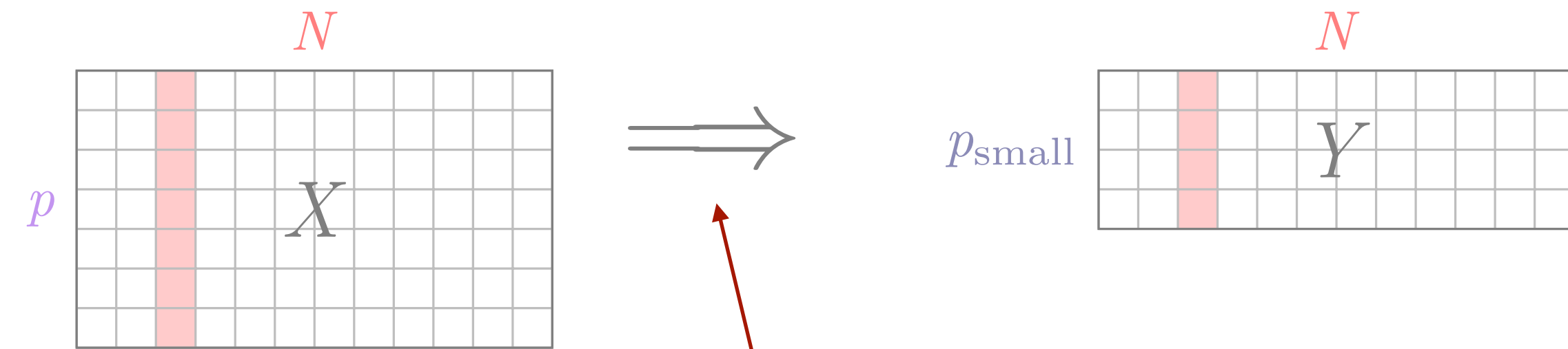
Reducing the dimensionality of data



$x_i \in \mathbb{R}^p$, $i = 1, \dots, N$ is converted to $y_i \in \mathbb{R}^{p_{\text{small}}}$, $i = 1, \dots, N$

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Reducing the dimensionality of data



$x_i \in \mathbb{R}^p$, $i = 1, \dots, N$ is converted to $y_i \in \mathbb{R}^{p_{\text{small}}}$, $i = 1, \dots, N$

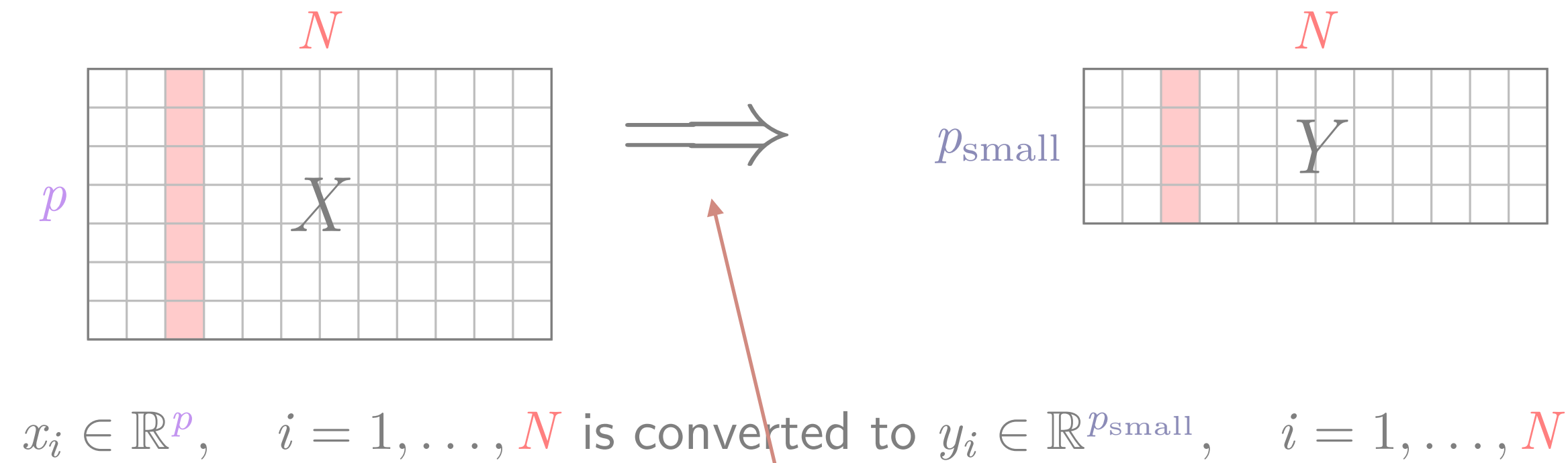
this reduction is often **linear**, in which case can write as

$$p_{\text{small}} \begin{matrix} N \\ Y \end{matrix} = p_{\text{small}} \begin{matrix} p \\ \Phi \end{matrix} \begin{matrix} p \\ N \\ X \end{matrix}$$

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Reducing the dimensionality of data

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization



this reduction is often **linear**, in which case can write as

$$p_{\text{small}} \times N \text{ matrix } Y = p_{\text{small}} \times p \text{ matrix } \Phi \times p \times N \text{ matrix } X$$

Why?

- ▶ Faster computation
 - ▶ Especially if the complexity of subsequent processing is $\mathcal{O}(p^2 N)$
 - ▶ If $p_{\text{small}} = 0.05p$ then speedup is $400\times$
- ▶ Denoising (remove irrelevant components)
- ▶ Fewer degrees-of-freedom reduces chance of overfitting models
- ▶ Visual interpretation (e.g., $p_{\text{small}} = \{2, 3\}$) ← This is “multidimensional scaling”. Specialized techniques (tSNE, UMAP) are best

Method 1: PCA (non-linear)

1. Randomized "sketches"
 - a. **Warmup: PCA**
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Principal Component Analysis (aka Hotelling or Karhunen-Loeve transform)

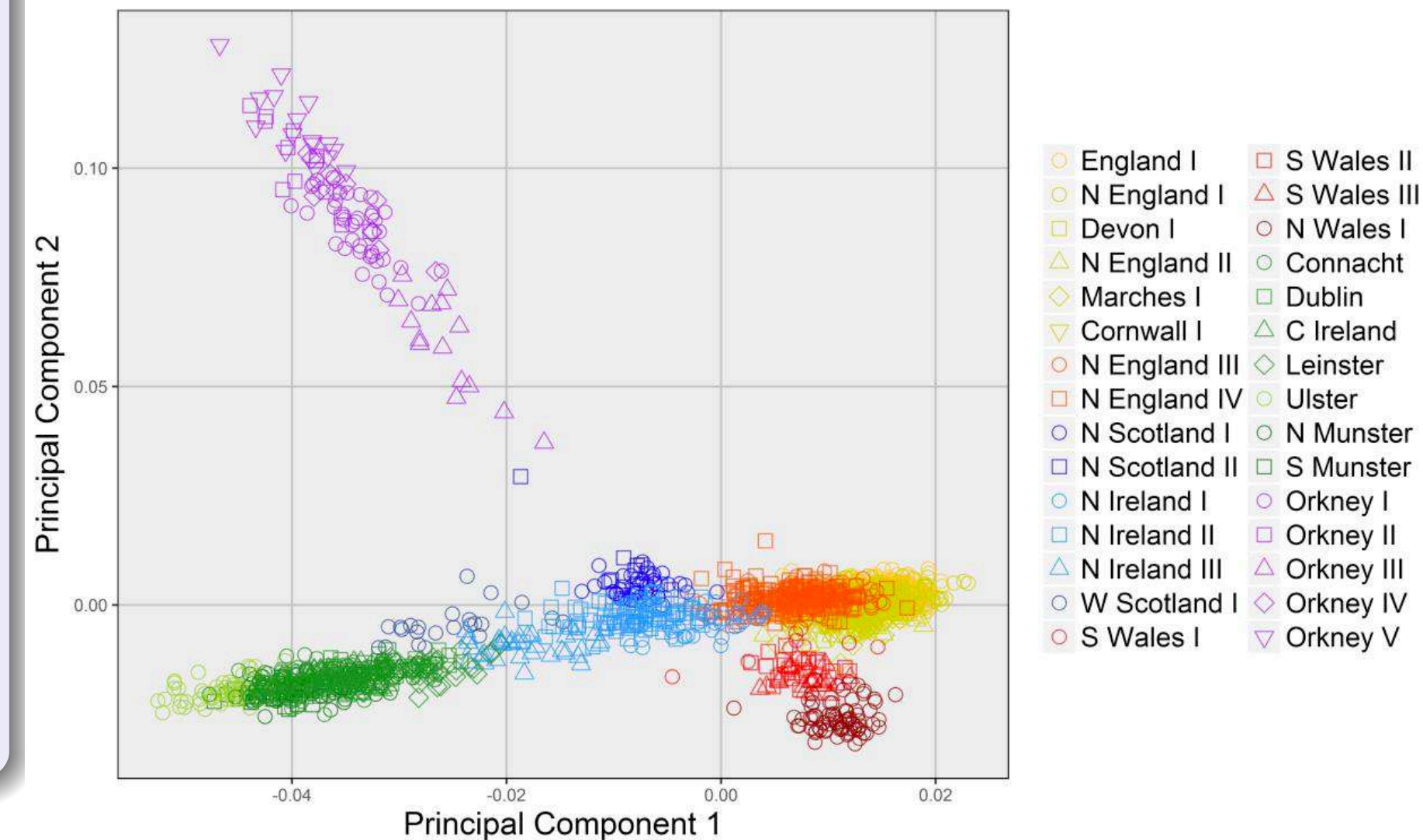
Take SVD

$$X = [U_1 \ U_2] \begin{bmatrix} \Sigma_1 & 0 \\ 0 & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}$$

where $\|\Sigma_2\| \approx \text{small}$. Then PCA takes the form of

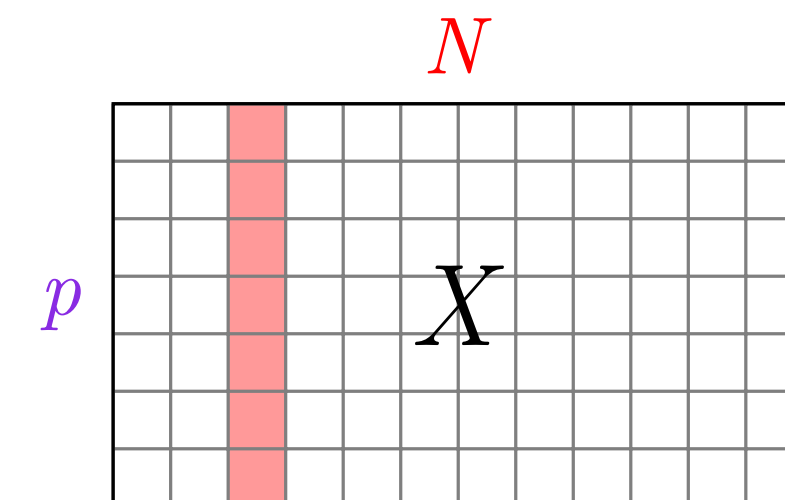
$$Y \leftarrow \Phi_{\text{PCA}} \cdot X$$

where $\Phi_{\text{PCA}} = U_1^T$.



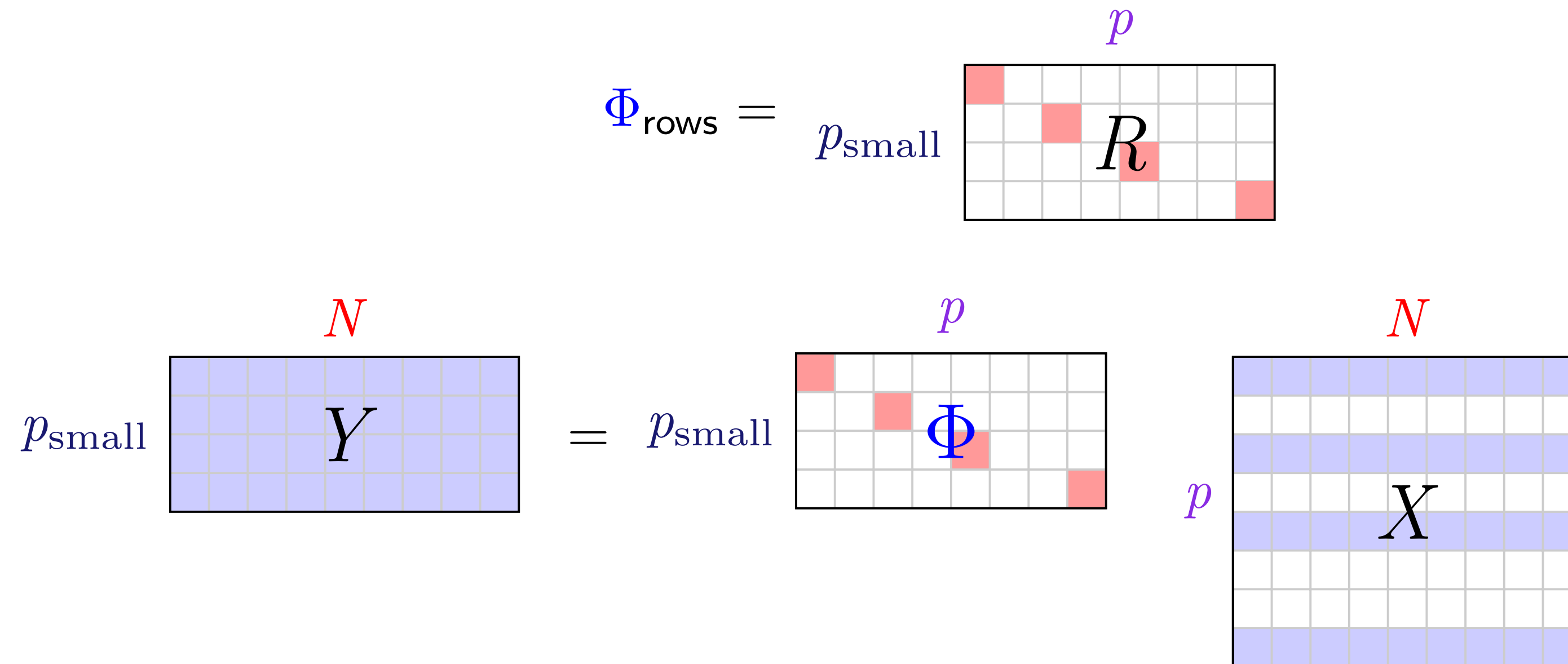
Note:

- ▶ Costly but linear in N : ($\mathcal{O}(p^2 N)$ direct, or about $\mathcal{O}(p_{\text{small}} p N)$ Krylov)
- ▶ Non-linear, since Φ_{PCA} is determined by X
- ▶ Need to process all of X before we can apply $y_i \leftarrow \Phi_{\text{PCA}} \cdot x_i$



Method 2: sub-sample dimensions (i.e., rows)

R randomly samples rows

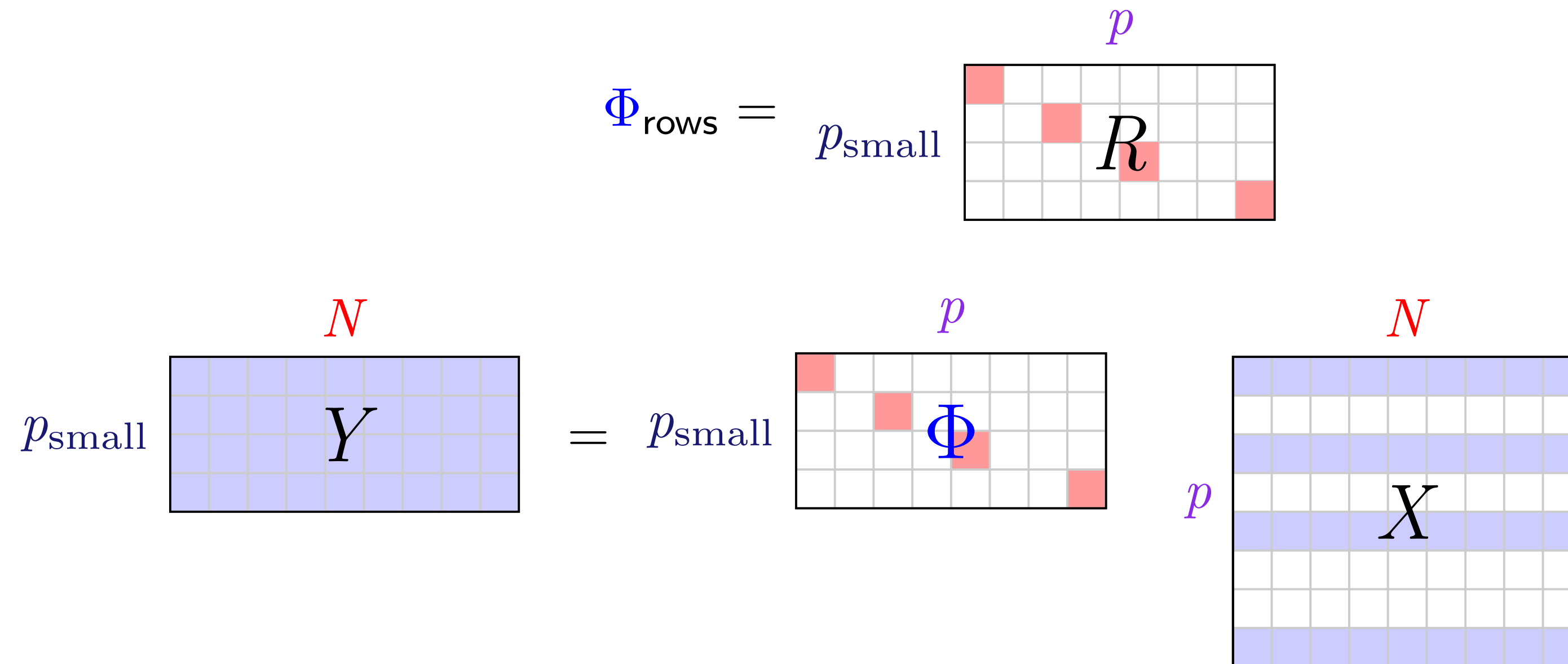


1. Randomized "sketches"
 - a. Warmup: PCA
 - b. **Classical sketches**
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Method 2: sub-sample dimensions (i.e., rows)

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. **Classical sketches**
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

R randomly samples rows



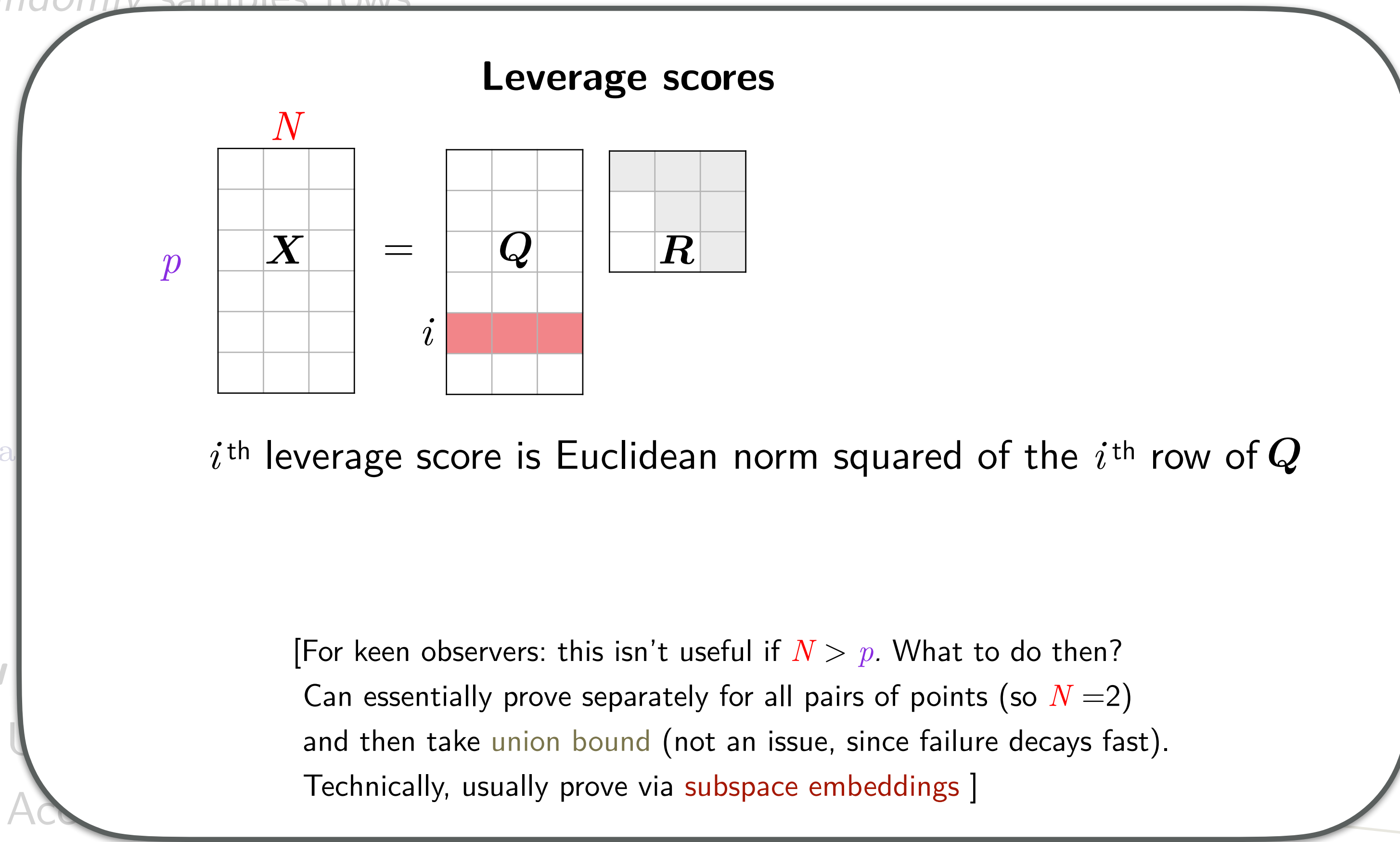
How to choose R ?

- ▶ Uniformly at random (... poor performance) ← “oblivious”
- ▶ According to row-norm (... better, not great, and slower) ← “non-oblivious”
- ▶ According to *leverage scores* based on partial SVD (... better, slow) ← “non-oblivious”

Method 2: sub-sample dimensions (i.e., rows)

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. **Classical sketches**
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

R randomly samples rows



How

[For keen observers: this isn't useful if $N > p$. What to do then?
Can essentially prove separately for all pairs of points (so $N=2$)
and then take **union bound** (not an issue, since failure decays fast).
Technically, usually prove via **subspace embeddings**]

“oblivious”

“non-oblivious”

- ▶ U
- ▶ Acc
- ▶ According to *leverage scores* based on partial SVD (... better, slow)

Method 3: random matrix

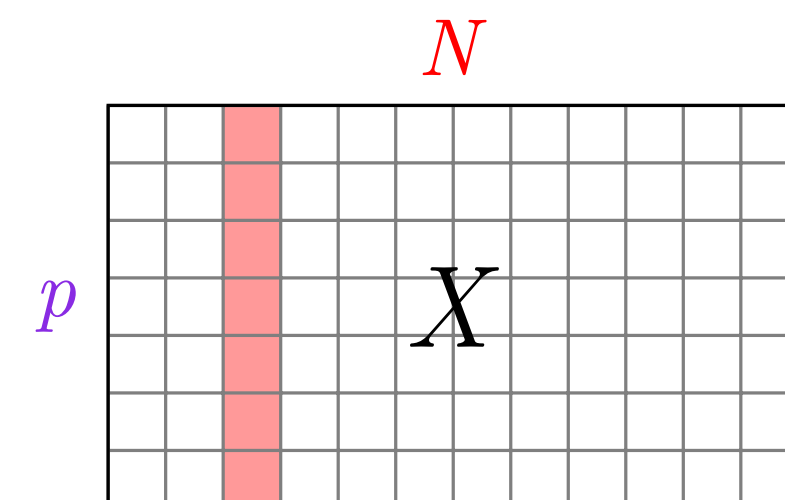
1. Randomized "sketches"
 - a. Warmup: PCA
 - b. **Classical sketches**
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Theorem (Johnson-Lindenstrauss, 1984 (and Indyk-Motwani, 1998))

Choose $\Phi = \Phi_{randn}$ with $p_{small} \propto \varepsilon^{-2} \log N$ iid rows each $\mathcal{N}(0, p/p_{small})$, then for all $x_i, x_j \in \{x_1, \dots, x_N\} \subset \mathbb{R}^p$,

$$1 - \varepsilon \leq \frac{\|\Phi x_i - \Phi x_j\|_2}{\|x_i - x_j\|_2} \leq 1 + \varepsilon$$

with constant probability.



Method 3: random matrix

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. **Classical sketches**
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Theorem (Johnson-Lindenstrauss, 1984 (and Indyk-Motwani, 1998))

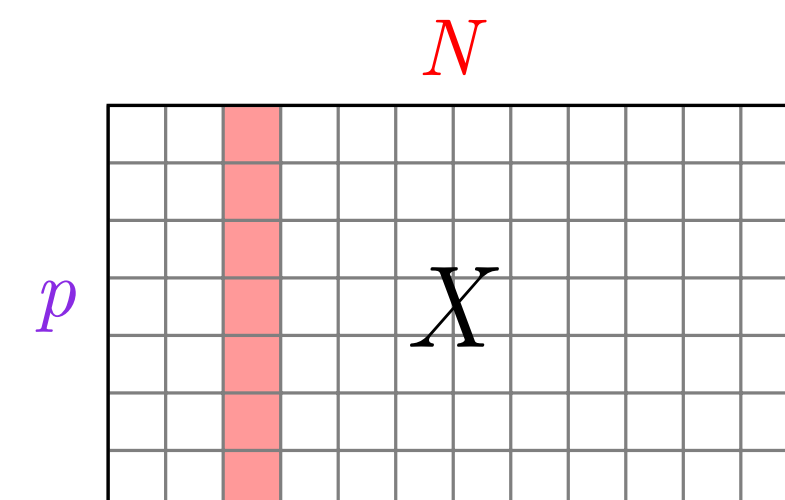
Choose $\Phi = \Phi_{randn}$ with $p_{small} \propto \varepsilon^{-2} \log N$ iid rows each $\mathcal{N}(0, p/p_{small})$, then for all $x_i, x_j \in \{x_1, \dots, x_N\} \subset \mathbb{R}^p$,

$$1 - \varepsilon \leq \frac{\|\Phi x_i - \Phi x_j\|_2}{\|x_i - x_j\|_2} \leq 1 + \varepsilon$$

with constant probability.

- ▶ Φ is linear and data-independent (unlike PCA)
- ▶ but Φ is not orthogonal (unlike PCA)
- ▶ Independent of original dimension p
- ▶ Independent of conditioning of X (unlike PCA)
- ▶ Probabilistic (unlike PCA)

Usually not practical since too costly since computing ΦX is $\mathcal{O}(pp_{small}N)$



Method 3: random matrix

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. **Classical sketches**
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Theorem (Johnson-Lindenstrauss, 1984 (and Indyk-Motwani, 1998))

Choose $\Phi = \Phi_{randn}$ with $p_{small} \propto \varepsilon^{-2} \log N$ iid rows each $\mathcal{N}(0, p/p_{small})$, then for all $x_i, x_j \in \{x_1, \dots, x_N\} \subset \mathbb{R}^p$,

$$1 - \varepsilon \leq \frac{\|\Phi x_i - \Phi x_j\|_2}{\|x_i - x_j\|_2} \leq 1 + \varepsilon$$

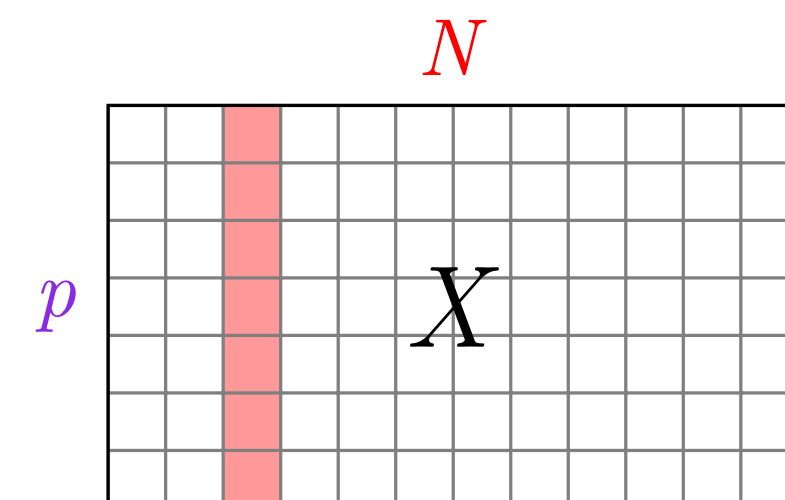
with constant probability.

- ▶ Φ is linear and data-independent (unlike PCA)
- ▶ but Φ is not orthogonal (unlike PCA)
- ▶ Independent of original dimension p
- ▶ Independent of conditioning of X (unlike PCA)
- ▶ Probabilistic (unlike PCA)

Usually not practical since too costly since computing ΦX is $\mathcal{O}(pp_{small}N)$

Many **extensions**, for example:

- sub-Gaussian entries, and/or dependent rows (e.g., columns uniform on sphere)
- dependent columns, e.g., Haar measure on orthogonal matrices
- sparse matrix (worse performance though)

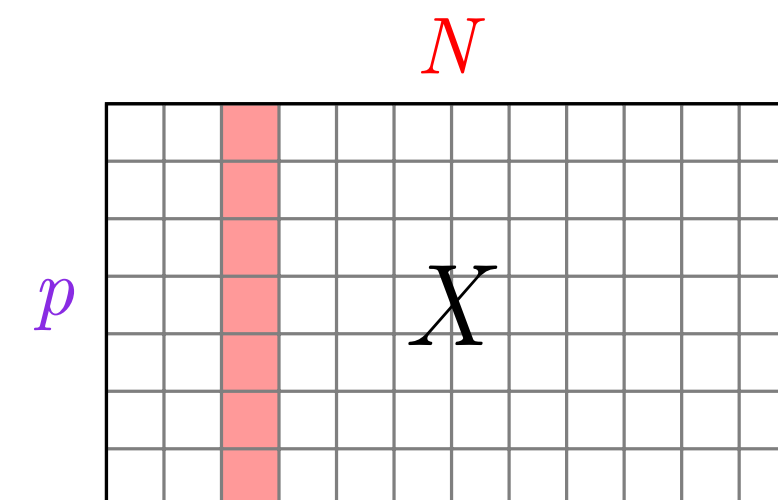


Method 4: Fast Johnson-Lindenstrauss Transforms

R samples rows; \mathcal{F} Fourier-like; D diag. w/ random ± 1 entries (“Rademacher”)
(uniformly)

$$\Phi_{\text{FFT}} = p_{\text{small}} \begin{matrix} p \\ \begin{matrix} \text{grid with } R \end{matrix} \end{matrix} \begin{matrix} p \\ \begin{matrix} \text{grid with } \mathcal{F} \end{matrix} \end{matrix} \begin{matrix} p \\ \begin{matrix} \text{grid with } D \end{matrix} \end{matrix}$$

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization



Method 4: Fast Johnson-Lindenstrauss Transforms

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

R samples rows; \mathcal{F} Fourier-like; D diag. w/ random ± 1 entries (“Rademacher”) (uniformly)

$$\Phi_{\text{FFT}} = p_{\text{small}} \begin{matrix} p \\ \begin{array}{|c|} \hline \begin{array}{c} \begin{array}{|c|} \hline \begin{array}{c} R \end{array} \\ \hline \end{array} \\ \hline \end{array} \\ p \end{matrix} \begin{matrix} p \\ \begin{array}{|c|} \hline \mathcal{F} \\ \hline \end{array} \\ p \end{matrix} \begin{matrix} p \\ \begin{array}{|c|} \hline \begin{array}{c} \begin{array}{|c|} \hline \begin{array}{c} D \end{array} \\ \hline \end{array} \\ \hline \end{array} \\ p \end{matrix}$$

Cost:

- ▶ $\mathcal{F}(x)$ costs $p \log p$ (vs p^2 naively)
- ▶ hence $\mathcal{O}(pN \log p)$ to compute ΦX , so good if $\log p < N$

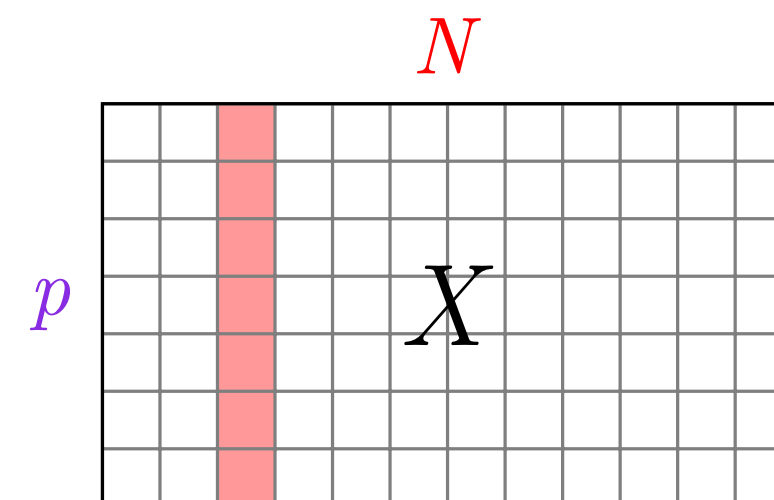
Guarantees: almost as good as classical Johnson-Lindenstrauss

First guarantees in the **Fast Johnson-Lindenstrauss** paper

Ailon and Chazelle, “Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform”, STOC 2006

Think of \mathcal{F} as FFT or DCT or Hadamard transform

also known under many names, e.g., Random Orthogonal System, FJLT



Method 4: Fast Johnson-Lindenstrauss Transforms

R samples rows; \mathcal{F} Fourier-like; D diag. w/ random ± 1 entries (“Rademacher”)

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

$$\Phi_{\text{FFT}} = \begin{matrix} & p & & p & & p \\ p_{\text{small}} & \begin{matrix} \begin{matrix} \color{red}{\square} & & & & \\ & \color{red}{\square} & & & \\ & & \color{red}{\square} & & \\ & & & \color{red}{\square} & \\ & & & & \color{red}{\square} \end{matrix} \\ R \end{matrix} & \begin{matrix} \begin{matrix} \square & & & & \\ & \square & & & \\ & & \square & & \\ & & & \square & \\ & & & & \square \end{matrix} \\ \mathcal{F} \end{matrix} & \begin{matrix} \begin{matrix} \color{red}{\square} & & & & \\ & \color{red}{\square} & & & \\ & & \color{red}{\square} & & \\ & & & \color{red}{\square} & \\ & & & & \color{red}{\square} \end{matrix} \\ D \end{matrix} \end{matrix}$$

How to get guarantees?

Basic idea is that after applying the first two steps, resulting matrix has (almost) uniform leverage scores...

... so **uniform subsampling** is (almost) **leverage score sampling**.

Cost:

- ▶ $\mathcal{F}(x)$ costs $p \log p$
- ▶ hence $\mathcal{O}(pN \log p)$

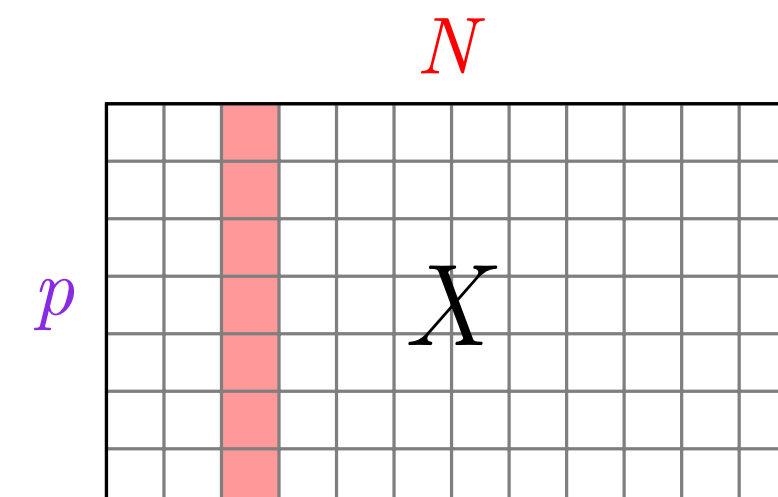
Guarantees: almost as good as...

First guarantees in the **Fast Johnson-Lindenstrauss** paper

Ailon and Chazelle, “Approximate nearest neighbors and the fast Johnson-Lindenstrauss transform”, STOC 2006

Think of \mathcal{F} as FFT or DCT or Hadamard transform

also known under many names, e.g., Random Orthogonal System, FJLT



Method 5: subsample entries of a matrix

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Keep $x'_i \in \mathbb{R}^p$ but with only p_{small} nonzeros

Idea goes back to landmark Achlioptas/McSherry paper (2001)

$$X' = \begin{matrix} & & & & & N \\ & & & & & \\ & & & & & \\ & & & & & \\ p & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \\ & & & & & \end{matrix} X$$

This is the **odd-one-out** so far, because it's not the same linear operator applied to every column

Method 6: “precondition,” then subsample entries of a matrix

- 1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
- 2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Our new twist: first precondition with Random Orthogonal System (ROS)

(i.e., first 2 matrices from the Fast JLT)

$$Y = \begin{matrix} & \begin{matrix} p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \end{matrix} \\ \begin{matrix} p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \end{matrix} & \begin{matrix} \mathcal{F} \\ \mathcal{F} \\ \mathcal{F} \\ \mathcal{F} \\ \mathcal{F} \\ \mathcal{F} \\ \mathcal{F} \\ \mathcal{F} \\ \mathcal{F} \\ \mathcal{F} \end{matrix} \end{matrix} \begin{matrix} & \begin{matrix} p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \end{matrix} \\ \begin{matrix} p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \end{matrix} & \begin{matrix} D \\ D \\ D \\ D \\ D \\ D \\ D \\ D \\ D \\ D \end{matrix} \end{matrix} \begin{matrix} & \begin{matrix} N \\ N \\ N \\ N \\ N \\ N \\ N \\ N \\ N \\ N \end{matrix} \\ \begin{matrix} p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \end{matrix} & \begin{matrix} X \\ X \\ X \\ X \\ X \\ X \\ X \\ X \\ X \\ X \end{matrix} \end{matrix}$$
$$Y' = \begin{matrix} & \begin{matrix} N \\ N \\ N \\ N \\ N \\ N \\ N \\ N \\ N \\ N \end{matrix} \\ \begin{matrix} p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \\ p \end{matrix} & \begin{matrix} Y \\ Y \\ Y \\ Y \\ Y \\ Y \\ Y \\ Y \\ Y \\ Y \end{matrix} \end{matrix}$$

Why precondition?

Easy to see that better than uniform sampling of X is **non-uniform** sampling,
with probability proportional to magnitude of entry

cf. Achlioptas, Z. Karnin, and E. Liberty 2013

Disadvantage of weighted sampling is the extra pass through data

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Why precondition?

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Easy to see that better than uniform sampling of X is **non-uniform** sampling, with probability proportional to magnitude of entry

cf. Achlioptas, Z. Karnin, and E. Liberty 2013

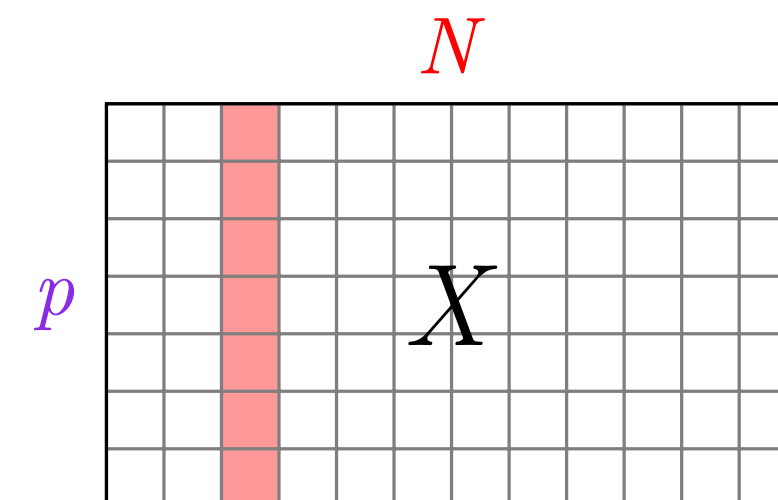
Disadvantage of weighted sampling is the extra pass through data

In our analysis, we have terms like (want this small)

$$\|X\|_{\text{max-entry}} = \max_{\substack{i=1,\dots,N \\ j=1,\dots,p}} |X_{ij}|$$

Assuming $\|x_i\|_2 = 1$,

- ▶ it is possible for $\|X\|_{\text{max-entry}} = 1$ (BAD)
- ▶ best case is $\|X\|_{\text{max-entry}} = 1/\sqrt{p}$



Why precondition?

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Easy to see that better than uniform sampling of X is **non-uniform** sampling, with probability proportional to magnitude of entry

cf. Achlioptas, Z. Karnin, and E. Liberty 2013

Disadvantage of weighted sampling is the extra pass through data

In our analysis, we have terms like (want this small)

$$\|X\|_{\max\text{-entry}} = \max_{\substack{i=1,\dots,N \\ j=1,\dots,p}} |X_{ij}|$$

Assuming $\|x_i\|_2 = 1$,

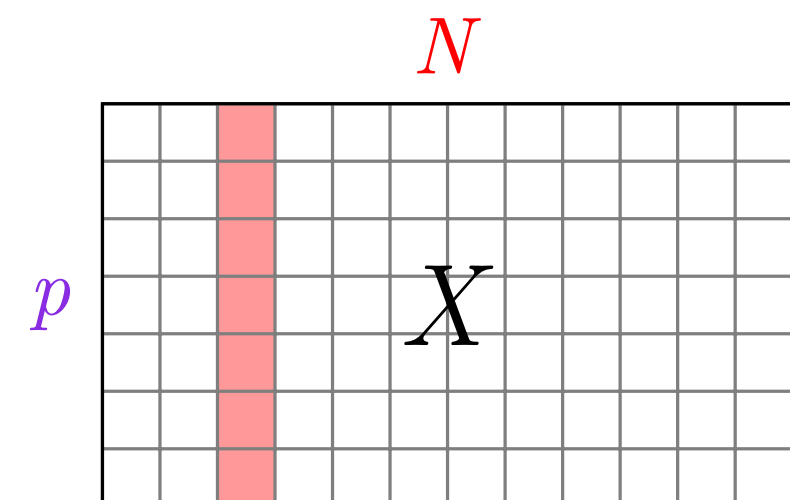
- ▶ it is possible for $\|X\|_{\max\text{-entry}} = 1$ (BAD)
- ▶ best case is $\|X\|_{\max\text{-entry}} = 1/\sqrt{p}$

Benefit of preconditioning

After applying ROS, exponentially small chance that

$$\|Y\|_{\max\text{-entry}} > \sqrt{\log(Np)}/\sqrt{p}$$

N.B. Since FD is unitary, bounds in spectral/Frobenius norm are unchanged



Theory: estimating the mean

Given a true mean $\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$, and our estimate of it \hat{x} from sampled data,



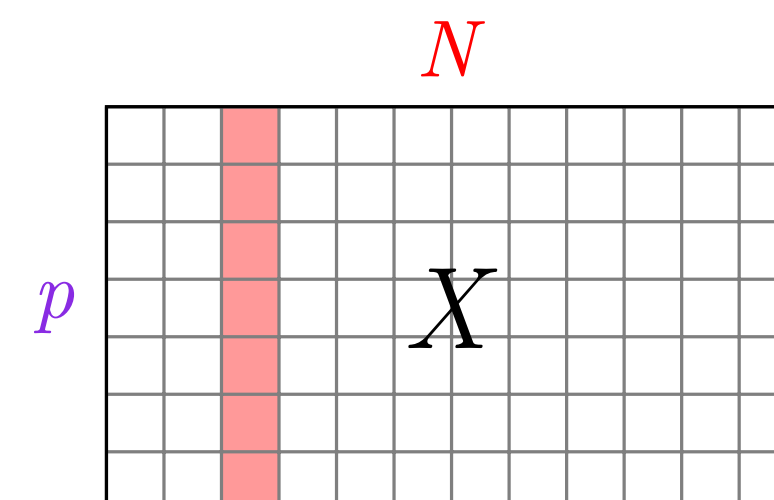
Theorem (Pourkamali-Anaraki & B., *IEEE Trans. Info Theory* 2017)

$\mathbb{E} \hat{x} = \bar{x}$ and $\|\hat{x} - \bar{x}\|_{\infty} \leq t$ with probability greater than

$$1 - 2p \exp\left(\frac{-N\gamma t^2/2}{\|X\|_{\max\text{-row}}^2 + t/3\|X\|_{\max\text{-entry}}}\right)$$

where $\gamma = p_{\text{small}}/p$ (simplifying to $p_{\text{small}} \ll p \ll N$)

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization



Theory: estimating the mean

Given a true mean $\bar{x} = \frac{1}{N} \sum_{i=1}^N x_i$, and our estimate of it \hat{x} from sampled data,

Theorem (Pourkamali-Anaraki & B., *IEEE Trans. Info Theory* 2017)

$\mathbb{E} \hat{x} = \bar{x}$ and $\|\hat{x} - \bar{x}\|_\infty \leq t$ with probability greater than

$$1 - 2p \exp\left(\frac{-N\gamma t^2/2}{\|X\|_{\max\text{-row}}^2 + t/3\|X\|_{\max\text{-entry}}}\right)$$

where $\gamma = p_{\text{small}}/p$ (simplifying to $p_{\text{small}} \ll p \ll N$)

If X has normalized columns, then $\|X\|_{\max\text{-entry}} = 1$ and $\|X\|_{\max\text{-row}} = \sqrt{N}$ are possible, which is bad.

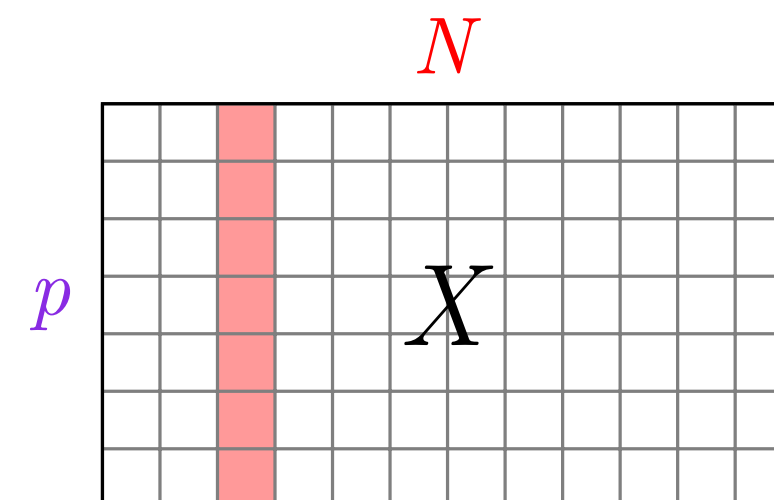
Lemma

If X is preconditioned, then (for $Np = 10^{10}$, $\sqrt{\log(2Np) + 1000} = 32$)

$$\mathbb{P}\left\{\|X\|_{\max\text{-entry}} \geq \frac{\sqrt{2}}{\sqrt{p}} \cdot \sqrt{\log(2Np) + 1000}\right\} \leq .001$$

$$\mathbb{P}\left\{\|X\|_{\max\text{-row}} \geq \frac{\sqrt{2N}}{\sqrt{p}} \cdot \sqrt{\log(2Np) + 1000}\right\} \leq .001$$

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization



Theory: estimating the mean

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Given a true

Probability theory aside

Theor

$$\mathbb{E} \hat{x} =$$

Let $y_i = \pm 1$ be iid Bernoulli, $\hat{x} = \frac{1}{N} \sum_{i=1}^N y_i$

- ▶ $\mathbb{E}(\hat{x}) = 0$
- ▶ $\text{Var}(\hat{x}) = 1/N$

where

Chebyshev, Markov inequalities

$$\mathbb{P}(|\hat{x}| \geq t) \leq \frac{1}{Nt^2}$$

Example: $N = 10^4$ and $t = 0.1$, $\mathbb{P}(|\hat{x}| \geq t) \leq 0.01$

If X has possible

Lemm

If X is

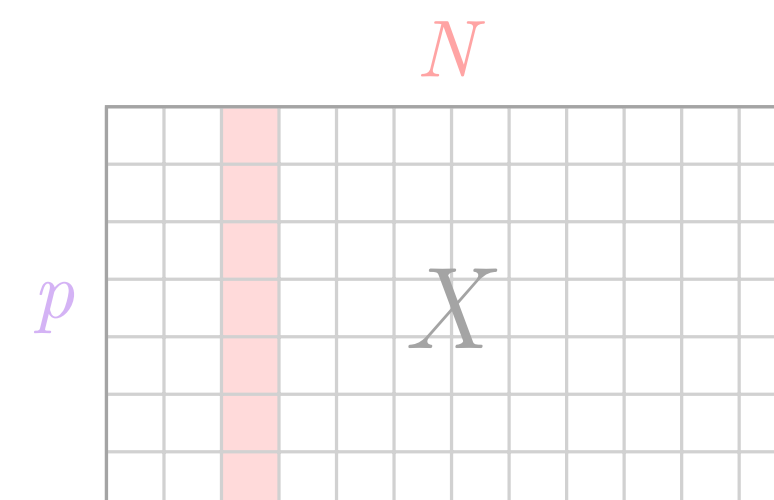
Concentration ineq.

Bernstein, Hoeffding, Chernoff

$$\mathbb{P}(|\hat{x}| \geq t) \leq 2 \exp\left(\frac{-Nt^2/2}{1 + t/3}\right)$$

Example: $N = 10^4$ and $t = 0.1$, $\mathbb{P}(|\hat{x}| \geq t) \leq 2 \cdot 10^{-21}$

intuition: central limit theorem says random sums of iid rv should look Gaussian, and for a Gaussian, we have exponential concentration



Method 7: CountSketch

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

p

p_{small}

	±1				±1	
±1						
			Φ	±1		
		±1	±1			
						±1

Introduced in [Charikar et al. \(2004\)](#), more analysis in, e.g., [Clarkson and Woodruff \(2017\)](#)

Based on **hash functions**

(as a side-effect, doesn't need full iid random variables for analysis to work)

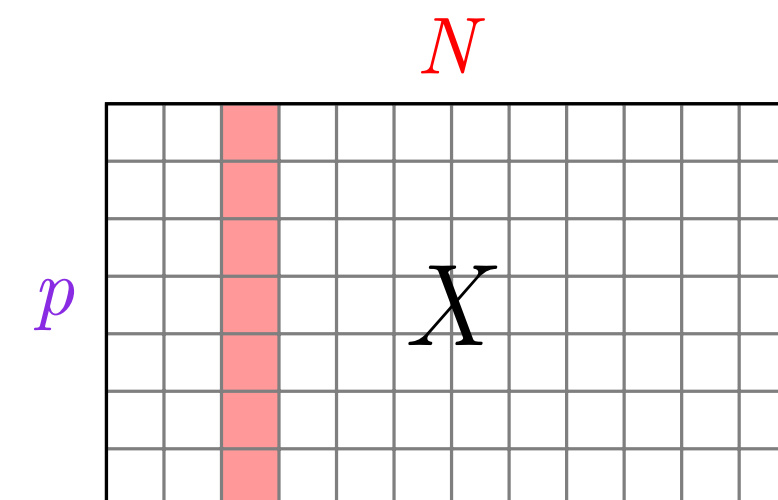
See "[appendix](#)" of these slides for more details

Every column has exactly 1 nonzero entry, location chosen uniformly at random (and value is a Rademacher r.v.)

Has some guarantees, though not as good as a Gaussian (and not a JLT)

... but it's very **fast** to apply

cost: $\text{nnz}(X)$, so no more than pN (vs $p_{\text{small}}pN$ with a Gaussian)



new linear algebra application in: [Malik, B.](#) "Fast randomized matrix and tensor interpolative decomposition using CountSketch", *Adv. Comp. Math* (2020)

Method 8: TensorSketch

Introduced in Pagh (2013), more analysis
in, e.g., Diao, Zong, Sun, Woodruff (2018)

TensorSketch is just CountSketch when the input can be written as a tensor product
(for a special choice of the hash and sign functions)

$$\Phi_{\mathcal{T}} : \mathbb{R}^p \rightarrow \mathbb{R}^{p_{\text{small}}}$$

$$\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \quad \text{where size is } p = p^{(1)} \cdot p^{(2)}$$

Not (yet!) related to tensors

Method 8: TensorSketch

Introduced in Pagh (2013), more analysis
in, e.g., Diao, Zong, Sun, Woodruff (2018)

TensorSketch is just CountSketch when the input can be written as a tensor product
(for a special choice of the hash and sign functions)

$$\Phi_{\mathcal{T}} : \mathbb{R}^p \rightarrow \mathbb{R}^{p_{\text{small}}}$$

$$\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \quad \text{where size is } p = p^{(1)} \cdot p^{(2)}$$

Kronecker/tensor product of vectors

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \mathbf{a} \otimes \mathbf{b} = \begin{bmatrix} a_1 \mathbf{b} \\ a_2 \mathbf{b} \\ a_3 \mathbf{b} \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_1 b_2 \\ a_1 b_3 \\ a_2 b_1 \\ a_2 b_2 \\ a_2 b_3 \\ a_3 b_1 \\ a_3 b_2 \\ a_3 b_3 \end{bmatrix}$$

Method 8: TensorSketch

Introduced in Pagh (2013), more analysis in, e.g., Diao, Zong, Sun, Woodruff (2018)

TensorSketch is just CountSketch when the input can be written as a tensor product (for a special choice of the hash and sign functions)

$$\Phi_{\mathcal{T}} : \mathbb{R}^p \rightarrow \mathbb{R}^{p_{\text{small}}}$$

$$\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)}$$

where size is $p = p^{(1)} \cdot p^{(2)}$

Kronecker/tensor product of vectors

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \mathbf{a} \otimes \mathbf{b} = \begin{bmatrix} a_1 \mathbf{b} \\ a_2 \mathbf{b} \\ a_3 \mathbf{b} \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_1 b_2 \\ a_1 b_3 \\ a_2 b_1 \\ a_2 b_2 \\ a_2 b_3 \\ a_3 b_1 \\ a_3 b_2 \\ a_3 b_3 \end{bmatrix}$$

another equivalent definition:

$$\mathbf{a} \otimes \mathbf{b} = \text{vec}_{\text{col}}(\mathbf{b}\mathbf{a}^T)$$

$$\mathbf{b}\mathbf{a}^T = \begin{bmatrix} b_1 a_1 & b_1 a_2 & b_1 a_3 \\ b_2 a_1 & b_2 a_2 & b_2 a_3 \\ b_3 a_1 & b_3 a_2 & b_3 a_3 \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_2 b_1 & a_3 b_1 \\ a_1 b_2 & a_2 b_2 & a_3 b_2 \\ a_1 b_3 & a_2 b_3 & a_3 b_3 \end{bmatrix}$$

$$\text{vec} : \mathbb{R}^{p^{(1)} \times p^{(2)}} \rightarrow \mathbb{R}^{p^{(1)} p^{(2)}}$$



$$\text{mat} = \text{vec}^{-1}$$

more generally,

$$(\mathbf{A} \otimes \mathbf{B}) \text{vec}_{\text{col}}(\mathbf{X}) = \text{vec}_{\text{col}}(\mathbf{B}\mathbf{X}\mathbf{A}^T)$$

TensorSketch: applying to matrices

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

From vector case $\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)}$ to matrix case $\mathbf{A} = \mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \dots \otimes \mathbf{A}^{(N)}$

Recall the **Kronecker** product:

$$\begin{array}{c} \mathbf{A} \otimes \mathbf{B} \\ \begin{array}{cc} \nearrow & \nwarrow \\ I \times J & K \times L \end{array} \end{array} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{IK \times JL}$$

$$= [\mathbf{a}_1 \otimes \mathbf{b}_1 \quad \mathbf{a}_1 \otimes \mathbf{b}_2 \quad \mathbf{a}_1 \otimes \mathbf{b}_3 \quad \cdots \quad \mathbf{a}_J \otimes \mathbf{b}_{L-1} \quad \mathbf{a}_J \otimes \mathbf{b}_L]$$

Khatri-Rao product

$$\mathbf{A} \odot \mathbf{B} = [\mathbf{a}_1 \otimes \mathbf{b}_1 \quad \mathbf{a}_2 \otimes \mathbf{b}_2 \quad \cdots \quad \mathbf{a}_K \otimes \mathbf{b}_K] \in \mathbb{R}^{IJ \times K}$$

pointwise multiplication (aka Hadamard product)

Observe

$$\mathbf{a} \circ \mathbf{b} = (\mathbf{a}^\top \odot \mathbf{b}^\top)^\top$$

and with some work, $\Phi_{\mathcal{T}}(\mathbf{A}) = \text{FFT}^{-1} \left(\left(\bigodot_{n=1}^N \left(\text{FFT} \left(\mathbf{S}^{(n)} \mathbf{A}^{(n)} \right) \right)^\top \right)^\top \right)$

small countSketch

FFTs are doing circular convolution, i.e., multiplying polynomials modulo a fixed modulus. See appendix

TensorSketch: complexity analysis

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

$$\Phi_{\mathcal{T}} : \mathbb{R}^p \rightarrow \mathbb{R}^{p_{\text{small}}}$$

$$p = p^{(1)} \cdot p^{(2)}$$

Complexity (per mat-vec):

CountSketch applied as TensorSketch vs CountSketch applied naively vs generic dense matrix multiply

$$\mathcal{O}(p^{(1)} + p^{(2)} + p_{\text{small}} \log p_{\text{small}}) \quad \text{vs} \quad \mathcal{O}(p^{(1)} \cdot p^{(2)}) \quad \text{vs} \quad \mathcal{O}(p^{(1)} \cdot p^{(2)} \cdot p_{\text{small}})$$

small CountSketches polynomial multiplication via FFTs

Savings grow as we have more tensor products

If $p = p^{(1)} \cdot p^{(2)} \cdot p^{(q)}$

$$\mathcal{O}(p^{(1)} + p^{(2)} + \dots + p^{(q)} + p_{\text{small}} \log p_{\text{small}}) \quad \text{vs} \quad \mathcal{O}(p^{(1)} p^{(2)} \dots p^{(q)})$$

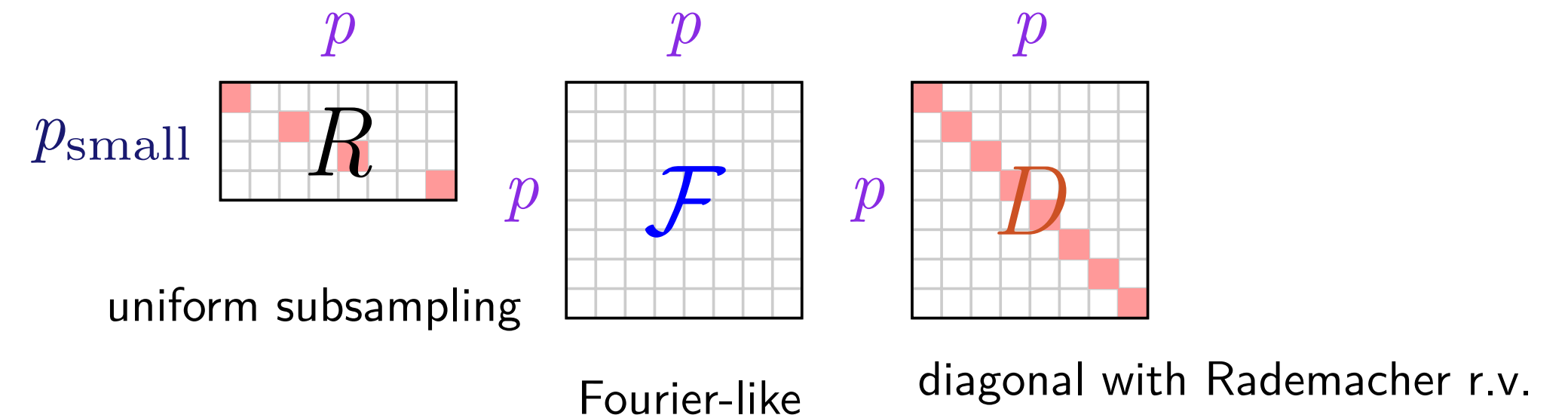
Method 9: Kronecker Fast Johnson-Lindenstrauss sketch

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

As for TensorSketch, suppose each column of data looks like $\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)}$

sizes: $p = p^{(1)} \cdot p^{(2)}$

Recall the Fast Johnson-Lindenstrauss (FJLT): $\Phi = R\mathcal{F}D$



The **Kronecker** FJLT has a similar structure:

$$\Phi_{\text{KFJLT}} = R \left(\mathcal{F}^{(1)} D^{(1)} \otimes \mathcal{F}^{(2)} D^{(2)} \right)$$

which is efficient to apply since

$$R \left(\mathcal{F}^{(1)} D^{(1)} \otimes \mathcal{F}^{(2)} D^{(2)} \right) \cdot \left(\mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \right) = R \left(\left(\mathcal{F}^{(1)} D^{(1)} \mathbf{v}^{(1)} \right) \otimes \left(\mathcal{F}^{(2)} D^{(2)} \mathbf{v}^{(2)} \right) \right)$$



- Battaglino, Ballard, Kolda, “A practical randomized CP tensor decomposition”, *SIAM J. Matrix Anal. Appl.* (2018)
- Jin, Kolda, Ward. “Faster Johnson–Lindenstrauss transforms via Kronecker products,” *Information and Inference: A Journal of the IMA* (2021)
- Malik, B. “Guarantees for the Kronecker fast Johnson–Lindenstrauss transform using a coherence and sampling argument,” *Lin. Alg. & its Applications* (2020)

Method 9: Kronecker Fast Johnson-Lindenstrauss sketch

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

$$\Phi_{\text{KFJLT}} = R \left(\mathcal{F}^{(1)} D^{(1)} \otimes \mathcal{F}^{(2)} D^{(2)} \otimes \dots \otimes \mathcal{F}^{(q)} D^{(q)} \right) \quad p = p^{(1)} p^{(2)} \dots p^{(q)}$$

Theorem (Thm 4.2 in Malik, B. 2020)

If all datapoints have a Kronecker structure, $p_{\text{small}} > \frac{16}{2} \frac{4^q}{\varepsilon^2} \log \left(\frac{4N^2(q+1)}{\delta} \right) \log \left(\frac{4p^{(1)}N^2(q+1)}{\delta} \right) \dots \log \left(\frac{4p^{(q)}N^2(q+1)}{\delta} \right)$ and $\Phi = \Phi_{\text{KFJLT}}$ then with probability at least $1 - \delta$

$$\forall x_i, x_j \in \{x_1, \dots, x_N\} \subset \mathbb{R}^p \quad 1 - \varepsilon \leq \frac{\|\Phi x_i - \Phi x_j\|_2^2}{\|x_i - x_j\|_2^2} \leq 1 + \varepsilon$$

proof idea: FJLT keeps leverage scores almost uniform. With Kronecker product structure, this is (almost) true also, due to properties of **leverage scores of Kronecker products**

in fact, can extend. See: Malik, Xu, Cheng, B., Doostan, Narayan. “Fast Algorithms for Monotone Lower Subsets of Kronecker Least Squares Problems”, <https://arxiv.org/abs/2209.05662>



o Malik, B. “Guarantees for the Kronecker fast Johnson–Lindenstrauss transform using a coherence and sampling argument,” *Lin. Alg. & its Applications* (2020)

Method 9: Kronecker Fast Johnson-Lindenstrauss sketch

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

$$\Phi_{\text{KFJLT}} = R \left(\mathcal{F}^{(1)} D^{(1)} \otimes \mathcal{F}^{(2)} D^{(2)} \otimes \dots \otimes \mathcal{F}^{(q)} D^{(q)} \right) \quad p = p^{(1)} p^{(2)} \dots p^{(q)}$$

Theorem (Thm 4.2 in Malik, B. 2020)

If all datapoints have a Kronecker structure, $p_{\text{small}} > \frac{16}{2} \frac{4^q}{\varepsilon^2} \log \left(\frac{4N^2(q+1)}{\delta} \right) \log \left(\frac{4p^{(1)}N^2(q+1)}{\delta} \right) \dots \log \left(\frac{4p^{(q)}N^2(q+1)}{\delta} \right)$ and $\Phi = \Phi_{\text{KFJLT}}$ then with probability at least $1 - \delta$

$$\forall x_i, x_j \in \{x_1, \dots, x_N\} \subset \mathbb{R}^p \quad 1 - \varepsilon \leq \frac{\|\Phi x_i - \Phi x_j\|_2^2}{\|x_i - x_j\|_2^2} \leq 1 + \varepsilon$$

Theorem (Johnson-Lindenstrauss, 1984 (and Indyk-Motwani, 1998))

Choose $\Phi = \Phi_{\text{randn}}$ with $p_{\text{small}} \propto \varepsilon^{-2} \log N$ iid rows each $\mathcal{N}(0, p/p_{\text{small}})$, then for all $x_i, x_j \in \{x_1, \dots, x_N\} \subset \mathbb{R}^p$,

for reference:

$$1 - \varepsilon \leq \frac{\|\Phi x_i - \Phi x_j\|_2}{\|x_i - x_j\|_2} \leq 1 + \varepsilon$$

with constant probability.

no dependence on p

OUTLINE

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

One application of sketching: least-squares

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. **Warmup: linear algebra**
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Least-squares

Solve $x_{\text{LS}} = \operatorname{argmin} \|Ax - b\|$, A is $p \times N$ with $p \gg N$

Approach 1: randomization to quickly find preconditioner

BLENDENPIK, LSRN

- ▶ If $A = QR$ is a QR-decomposition, then AR^{-1} is well-conditioned
- ▶ Idea: do QR-decomp on reduced-dimension matrix ΦA
- ▶ On large, very ill-conditioned matrices (and tall), about $4\times$ faster than LAPACK

One application of sketching: least-squares

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. **Warmup: linear algebra**
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Least-squares

Solve $x_{\text{LS}} = \operatorname{argmin} \|Ax - b\|$, A is $p \times N$ with $p \gg N$

Approach 1: randomization to quickly find preconditioner

BLENDENPIK, LSRN

- ▶ If $A = QR$ is a QR-decomposition, then AR^{-1} is well-conditioned
- ▶ Idea: do QR-decomp on reduced-dimension matrix ΦA
- ▶ On large, very ill-conditioned matrices (and tall), about $4\times$ faster than LAPACK

Approach 2: directly solve sketched problem

Sarlos, Woodruff, Mahoney, etc.

- ▶ Directly solve $\min_x \|\Phi(Ax - b)\|$
- ▶ Theoretical bounds on objective error if Φ has $p_{\text{small}} = N \operatorname{polylog}(N)$ rows
 - ▶ i.e., p_{small} is independent of p

One application of sketching: least-squares

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. **Warmup: linear algebra**
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Least-squares

Solve $x_{\text{LS}} = \operatorname{argmin} \|Ax - b\|$, A is $p \times N$ with $p \gg N$

Approach 1: randomization to quickly find preconditioner

BLENDENPIK, LSRN

- ▶ If $A = QR$ is a QR-decomposition, then AR^{-1} is well-conditioned
- ▶ Idea: do QR-decomp on reduced-dimension matrix ΦA
- ▶ On large, very ill-conditioned matrices (and tall), about $4\times$ faster than LAPACK

Approach 2: directly solve sketched problem

Sarlos, Woodruff, Mahoney, etc.

- ▶ Directly solve $\min_x \|\Phi(Ax - b)\|$
- ▶ Theoretical bounds on objective error if Φ has $p_{\text{small}} = N \operatorname{polylog}(N)$ rows
 - ▶ i.e., p_{small} is independent of p

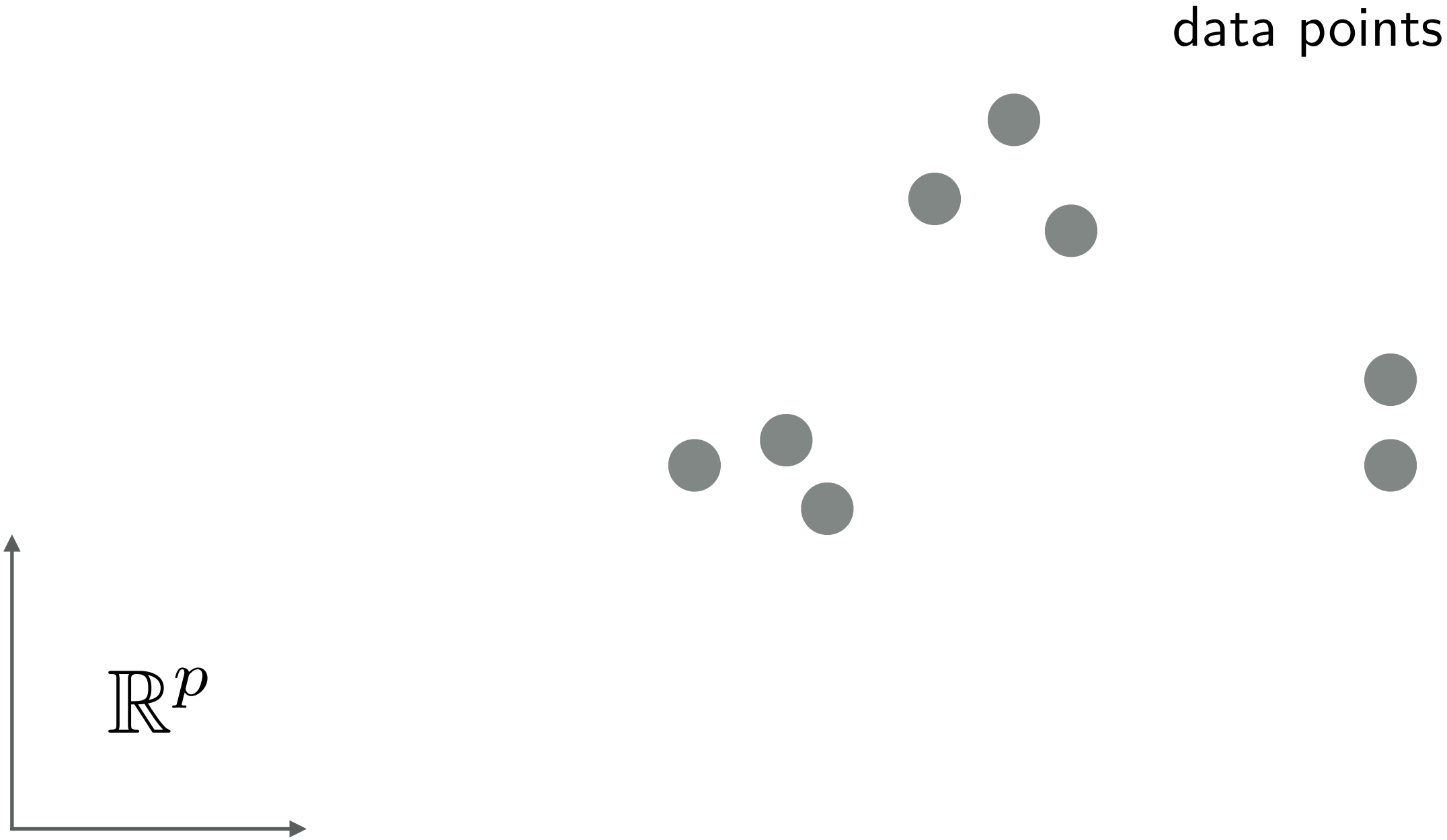
Other applications: SVDs, matrix multiplies, Nystrom/CUR/Interpolative Decompositions, QR without pivoting

OUTLINE

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Machine Learning 101: K-means clustering

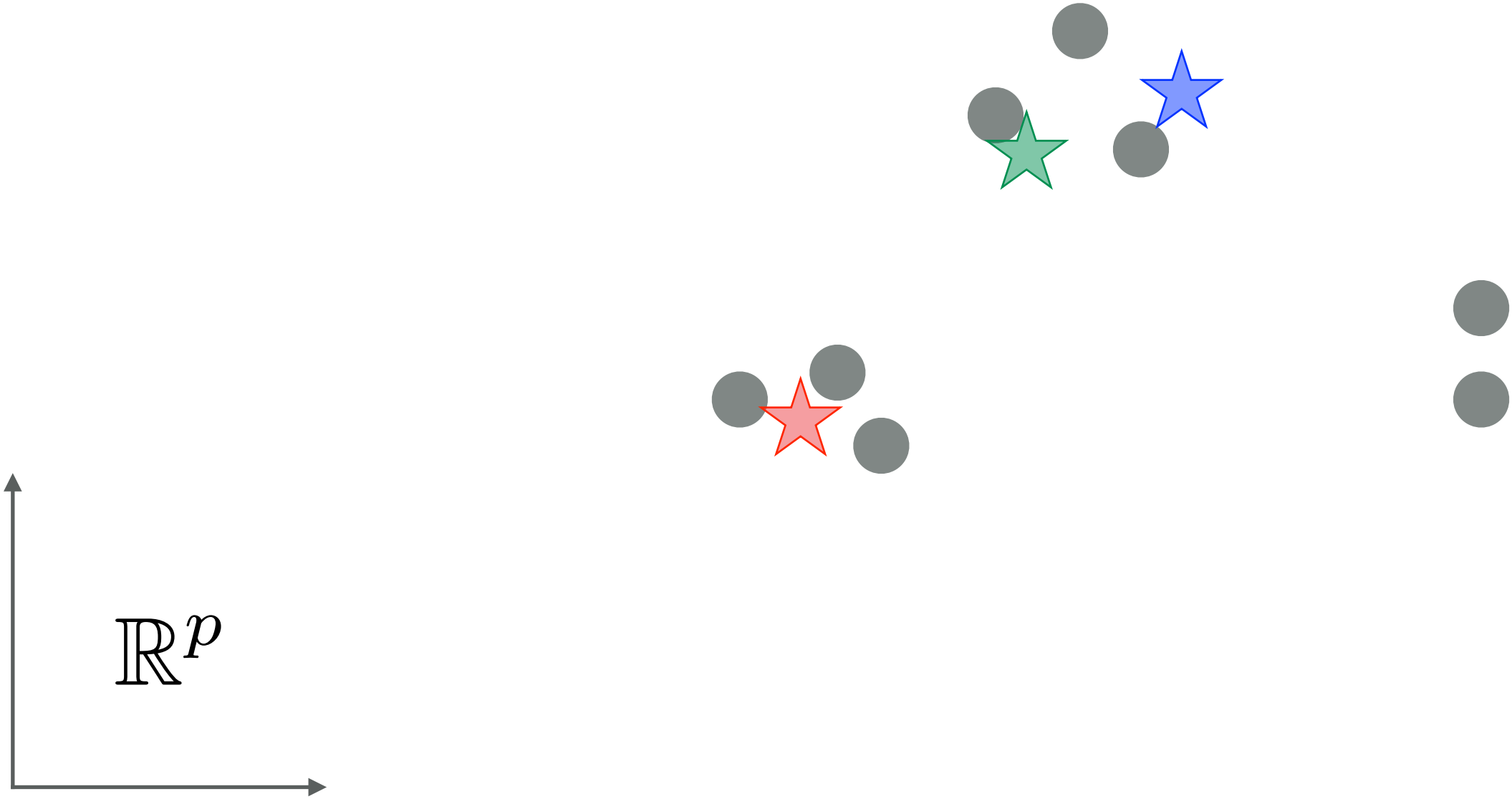
- 1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
- 2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization



Machine Learning 101: K-means clustering

- 1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
- 2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

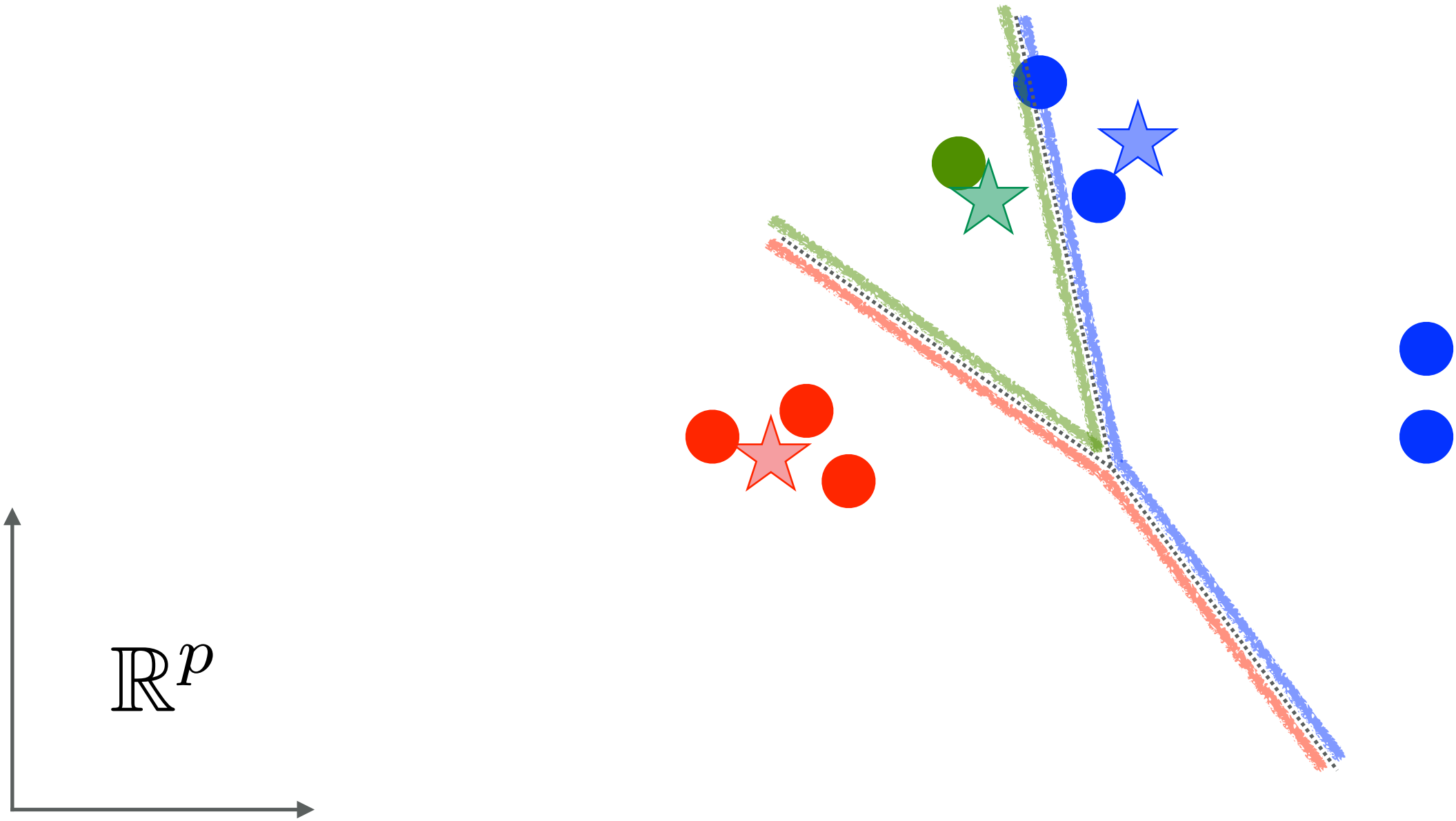
Step: initialize with guesses for cluster **centers**



Machine Learning 101: K-means clustering

- 1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
- 2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

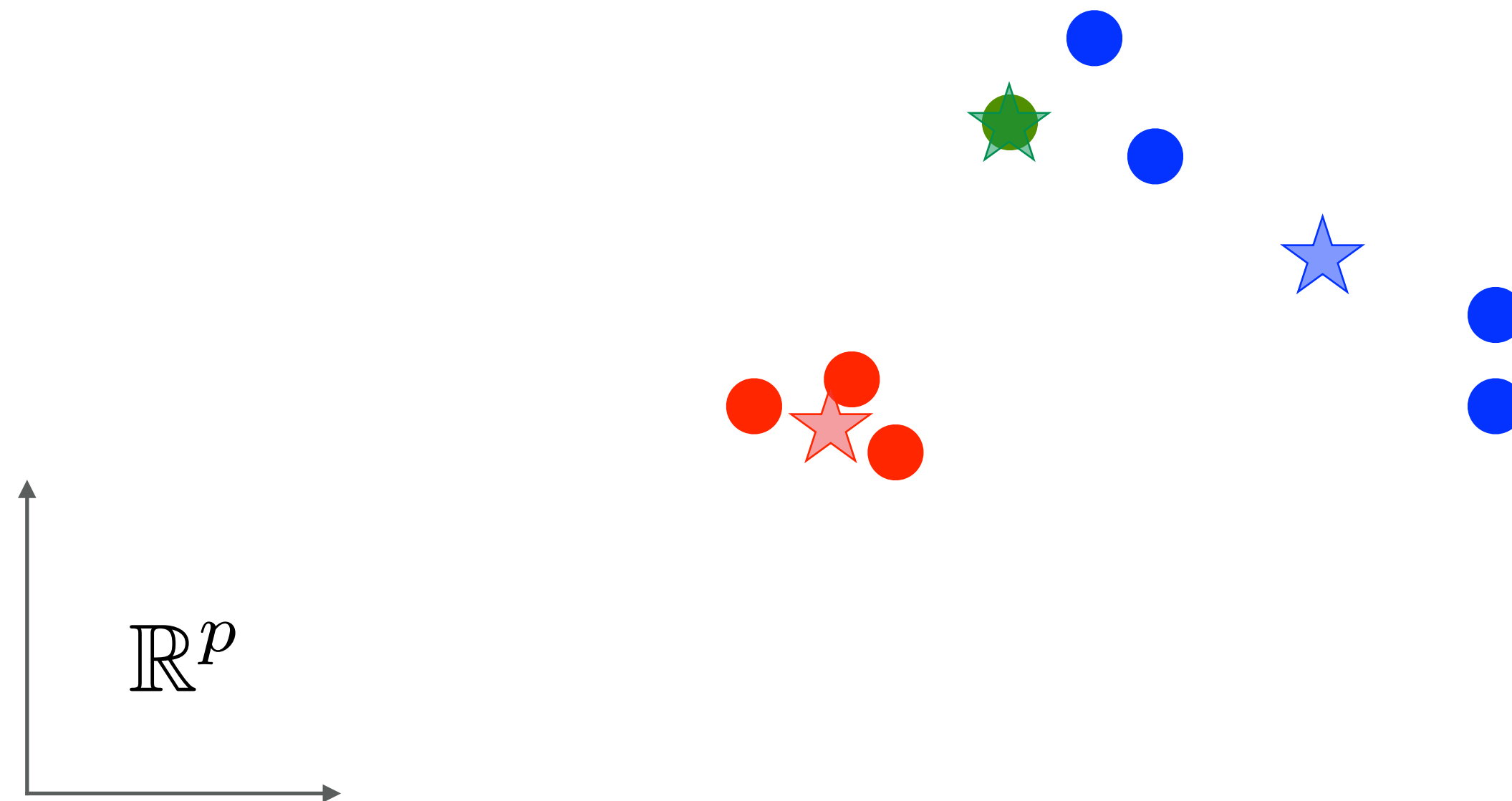
Step: update **assignments**



Machine Learning 101: K-means clustering

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

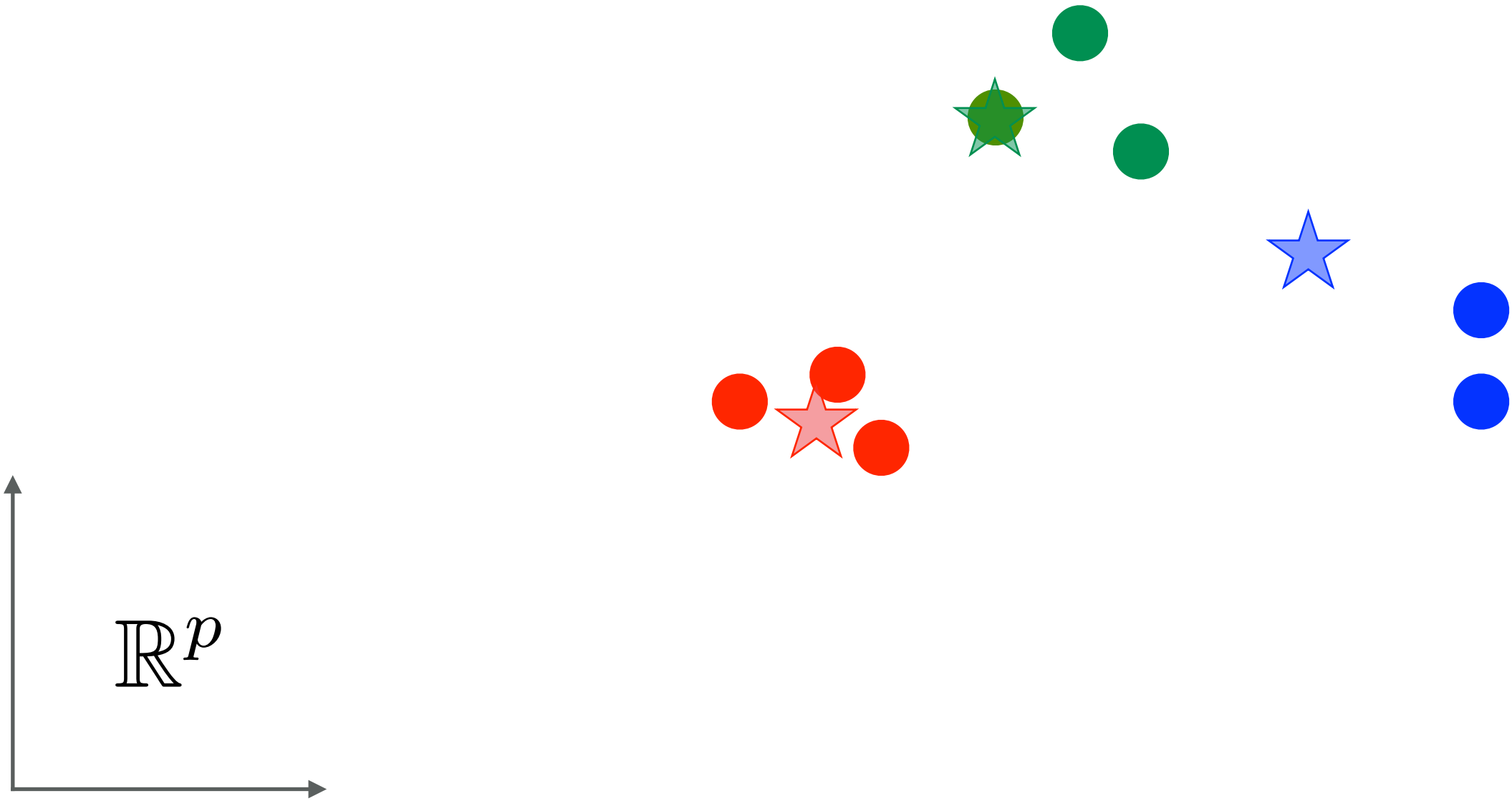
Step: recompute **cluster means**



Machine Learning 101: K-means clustering

- 1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
- 2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

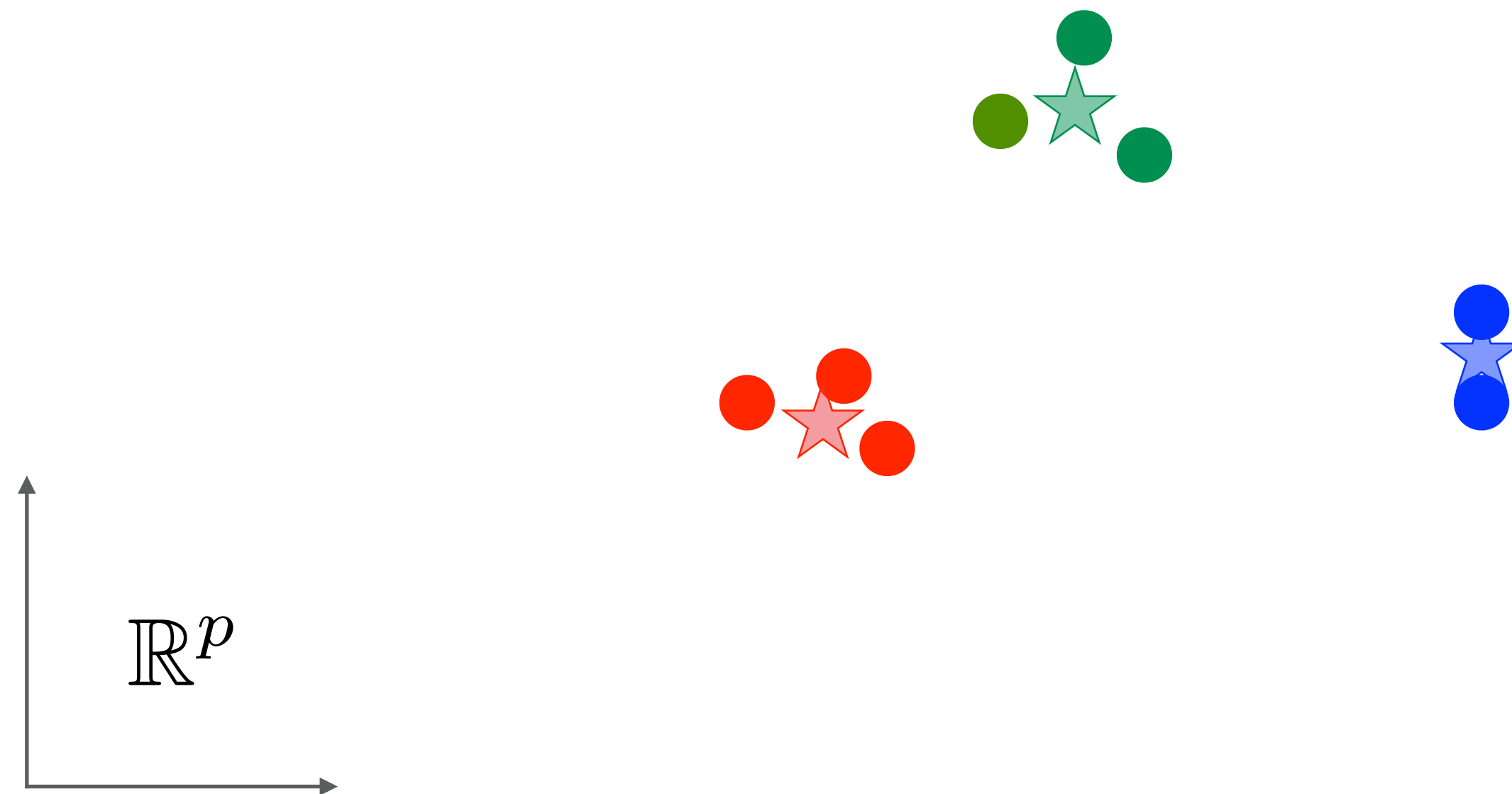
Step: update **assignments**



Machine Learning 101: K-means clustering

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

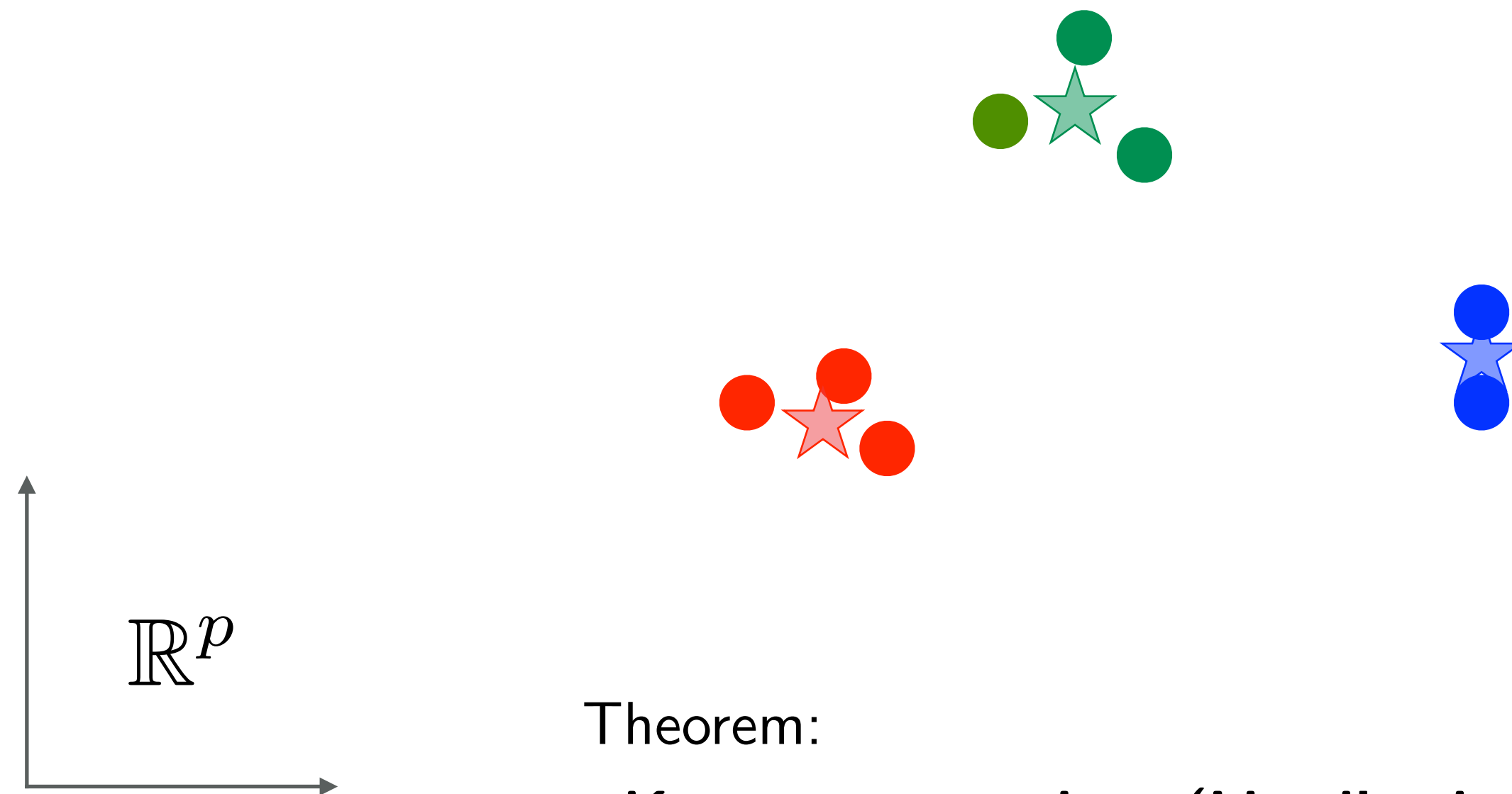
Step: recompute **cluster means**



Machine Learning 101: K-means clustering

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

Step: recompute **cluster means**



Theorem:

K-means procedure (Lloyd's algorithm) converges

Proof:

There are only a finite number of assignments, and we can never cycle since each iteration is an improvement

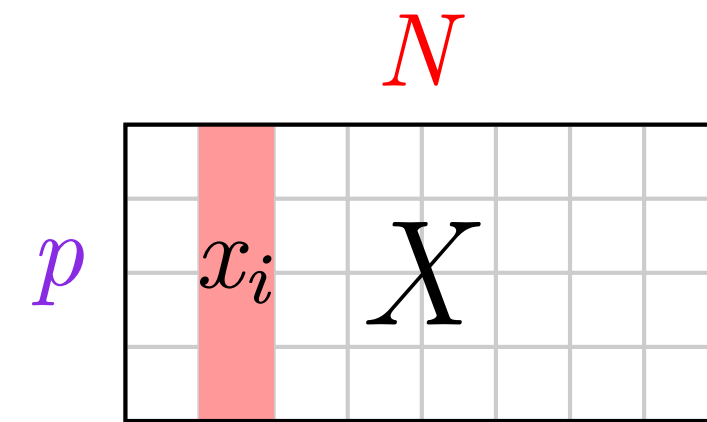
K-means clustering: complexity

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

Hard K-means / Lloyd's algorithm:

$$\{x_i\}_{i=1,\dots,N} \subset \mathbb{R}^p$$

1. Update cluster centers $\mu_k \in \mathbb{R}^p$ for $k = 1, \dots, K$
2. Update assignments $c_i \in \{1, \dots, K\}$ for $i = 1, \dots, N$



Let $X = (x_i)_{i=1,\dots,N}$

To update assignment of x_i :

- ▶ Compute $\|x_i - \mu_k\|$ (cost: p flops).
- ▶ For all k , and all i , this means $\mathcal{O}(KNp)$ cost.

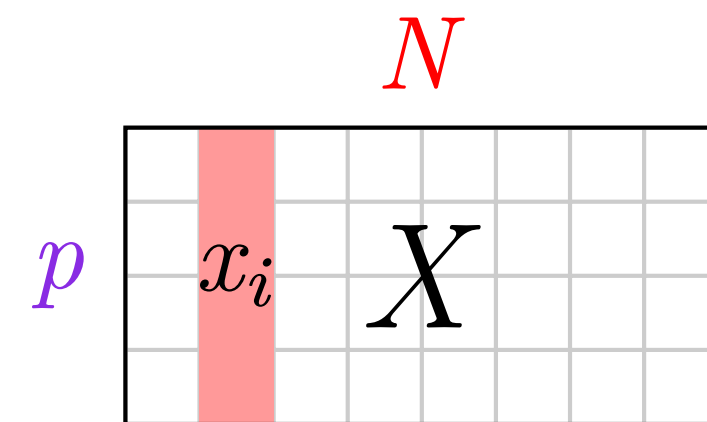
K-means clustering: complexity

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

Hard K-means / Lloyd’s algorithm:

$$\{x_i\}_{i=1,\dots,N} \subset \mathbb{R}^p$$

1. Update cluster centers $\mu_k \in \mathbb{R}^p$ for $k = 1, \dots, K$
2. Update assignments $c_i \in \{1, \dots, K\}$ for $i = 1, \dots, N$

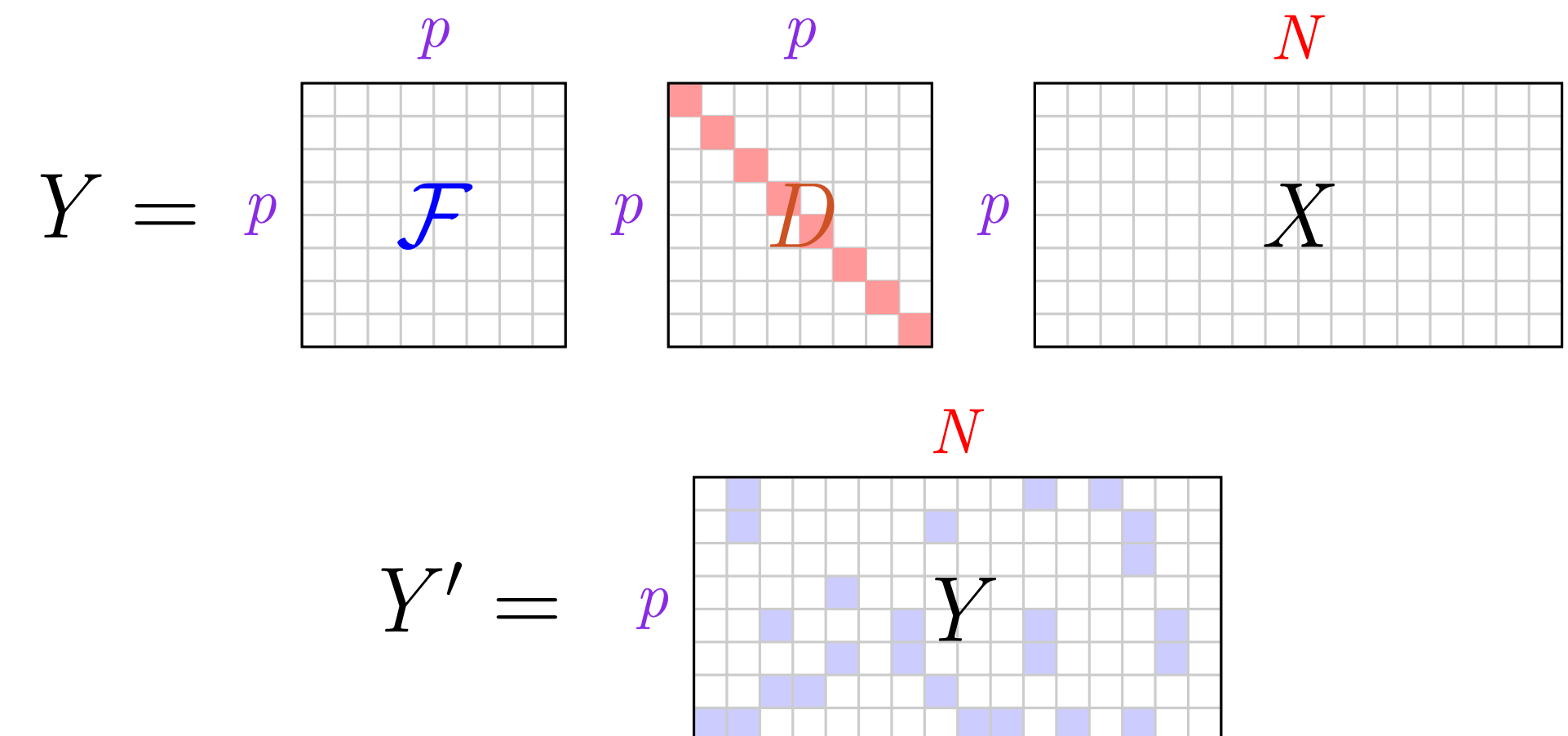


Let $X = (x_i)_{i=1,\dots,N}$

To update assignment of x_i :

- ▶ Compute $\|x_i - \mu_k\|$ (cost: p flops).
- ▶ For all k , and all i , this means $\mathcal{O}(KNp)$ cost.

Our idea: apply “preconditioned” entry-wise subsampling



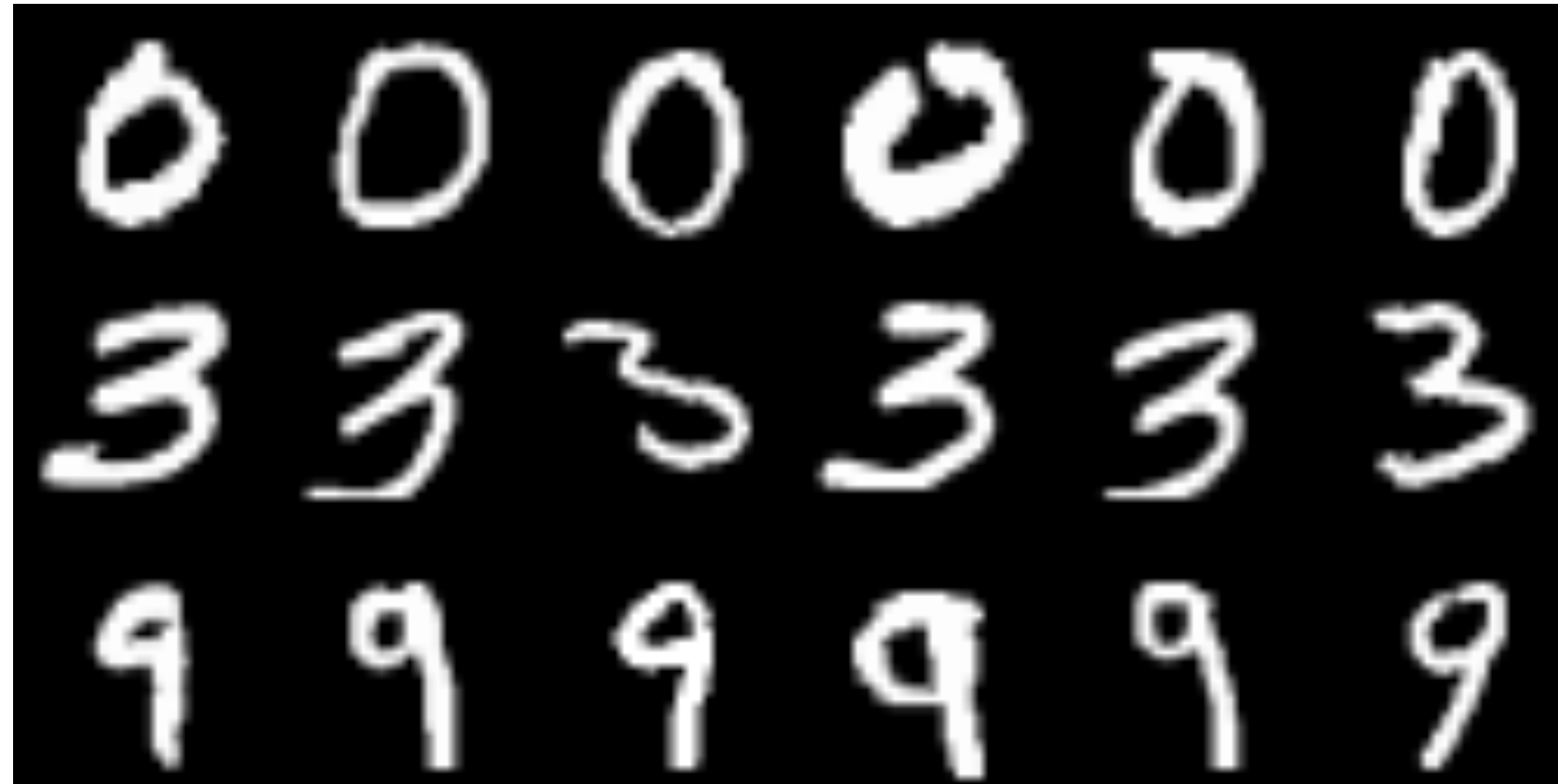
Our sampling is suitable for **streaming** data

MNIST handwritten digits

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

Goal: recover cluster centers, using K-means clustering algorithm

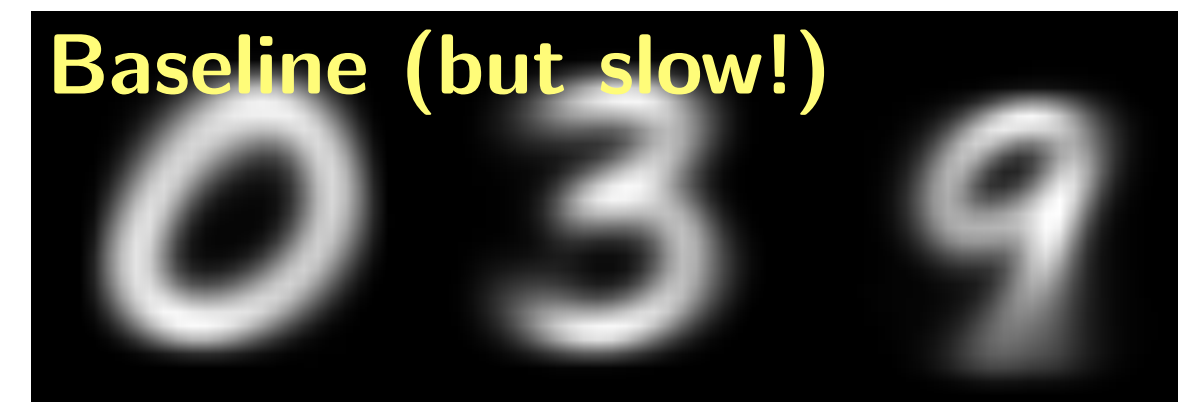
Data: $N = 21,002$ examples of 28×28 pixel images ($p = 784$)



Experiments: MNIST handwritten digits



(a) true cluster centers



(b) K-means, many passes

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

Experiments: MNIST handwritten digits

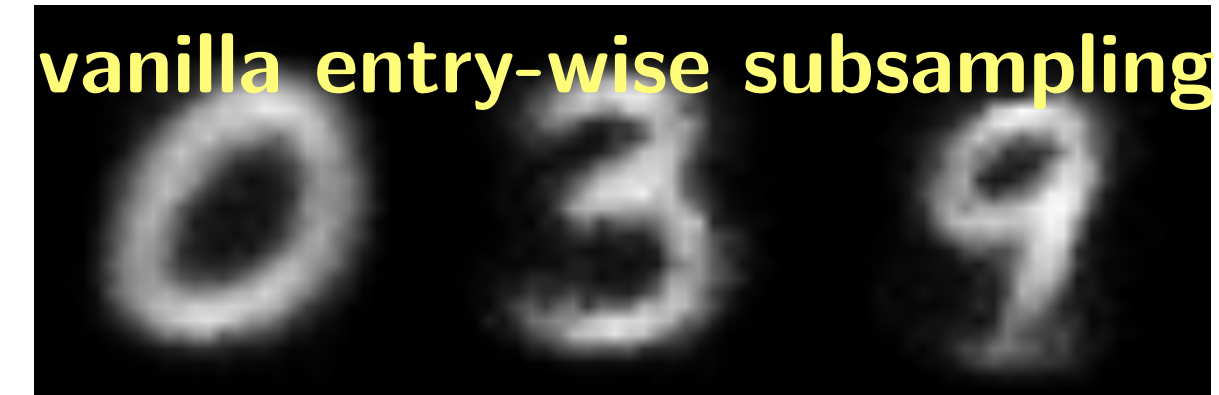
1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization



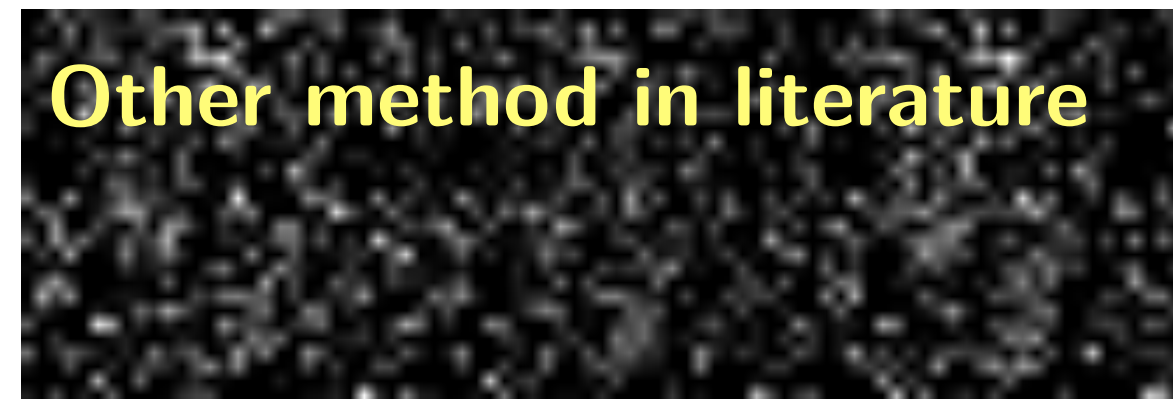
(a) true cluster centers



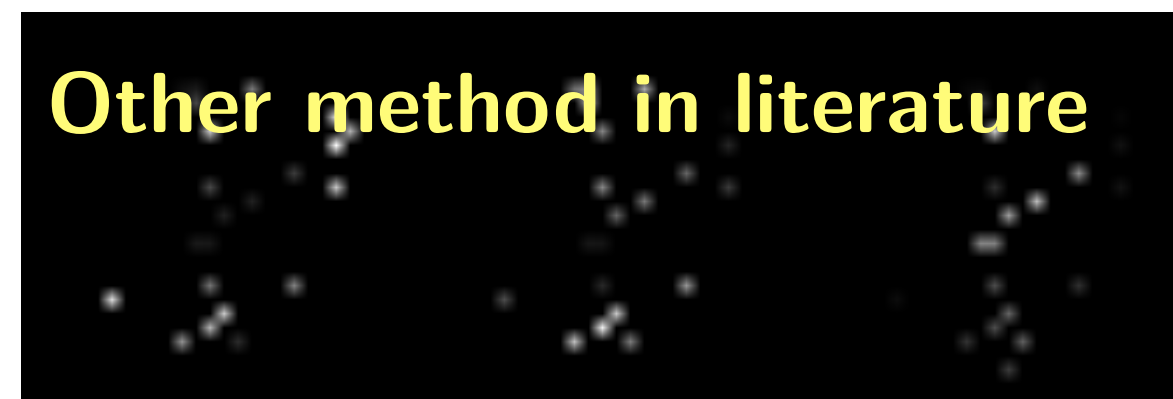
(b) K-means, many passes



(c) sparsified K-means, 1 pass, no preconditioning



(g) feature extraction, 1 pass



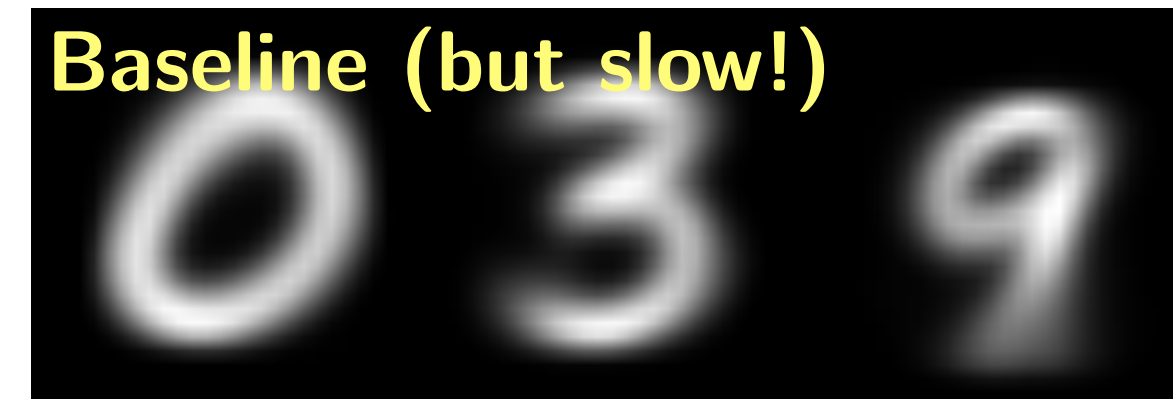
(h) feature selection, 3 passes

Experiments: MNIST handwritten digits

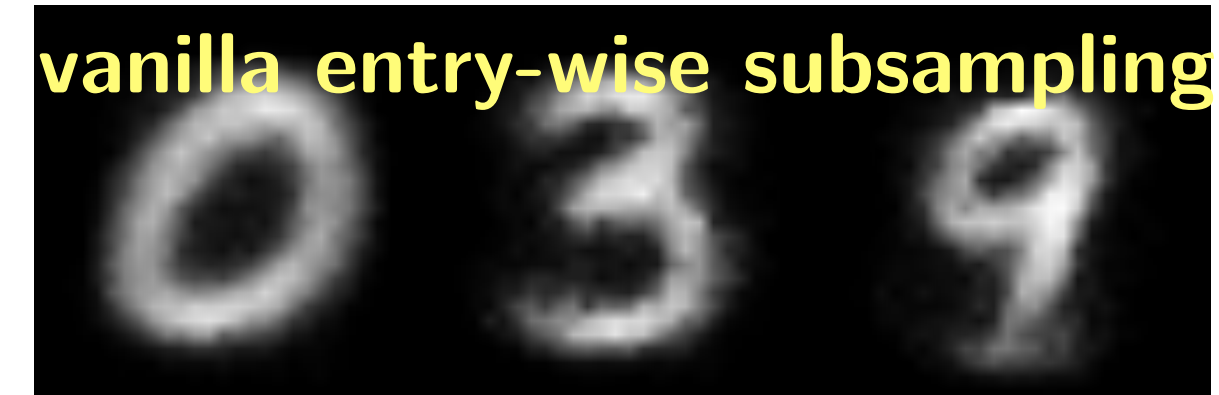
1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization



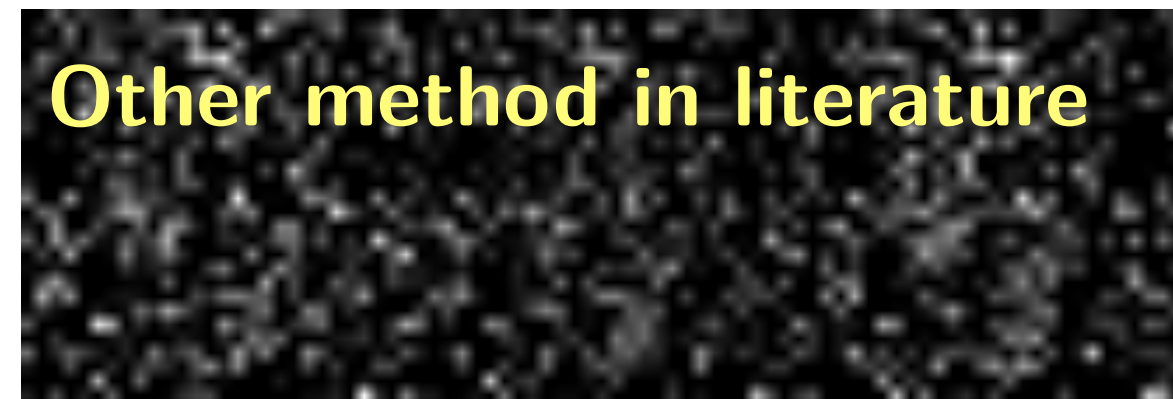
(a) true cluster centers



(b) K-means, many passes



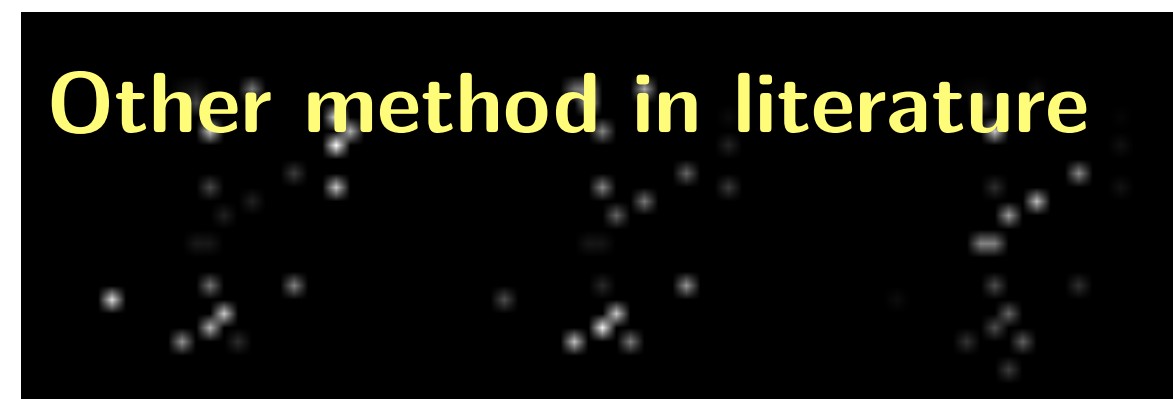
(c) sparsified K-means, 1 pass, no preconditioning



(g) feature extraction, 1 pass



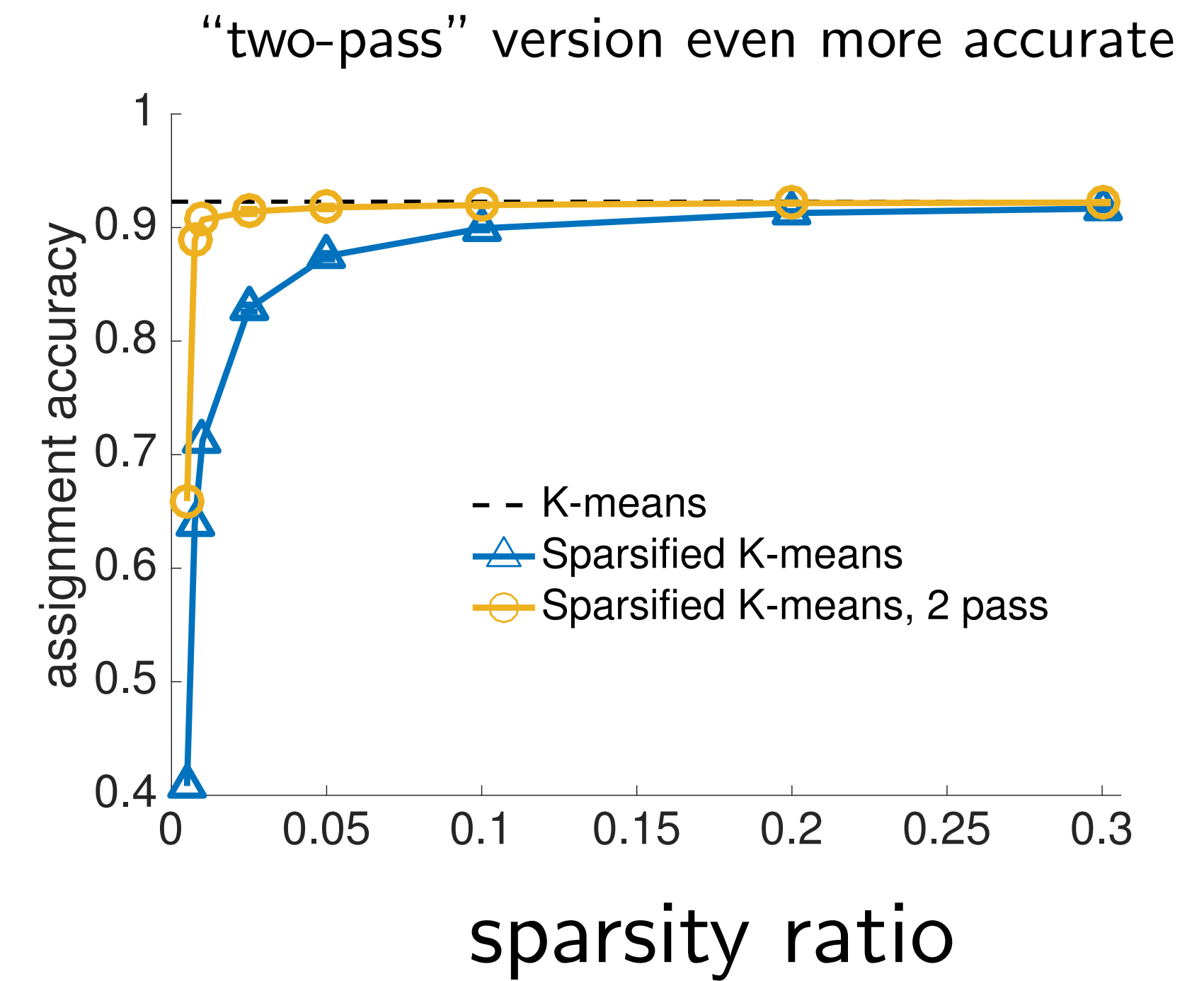
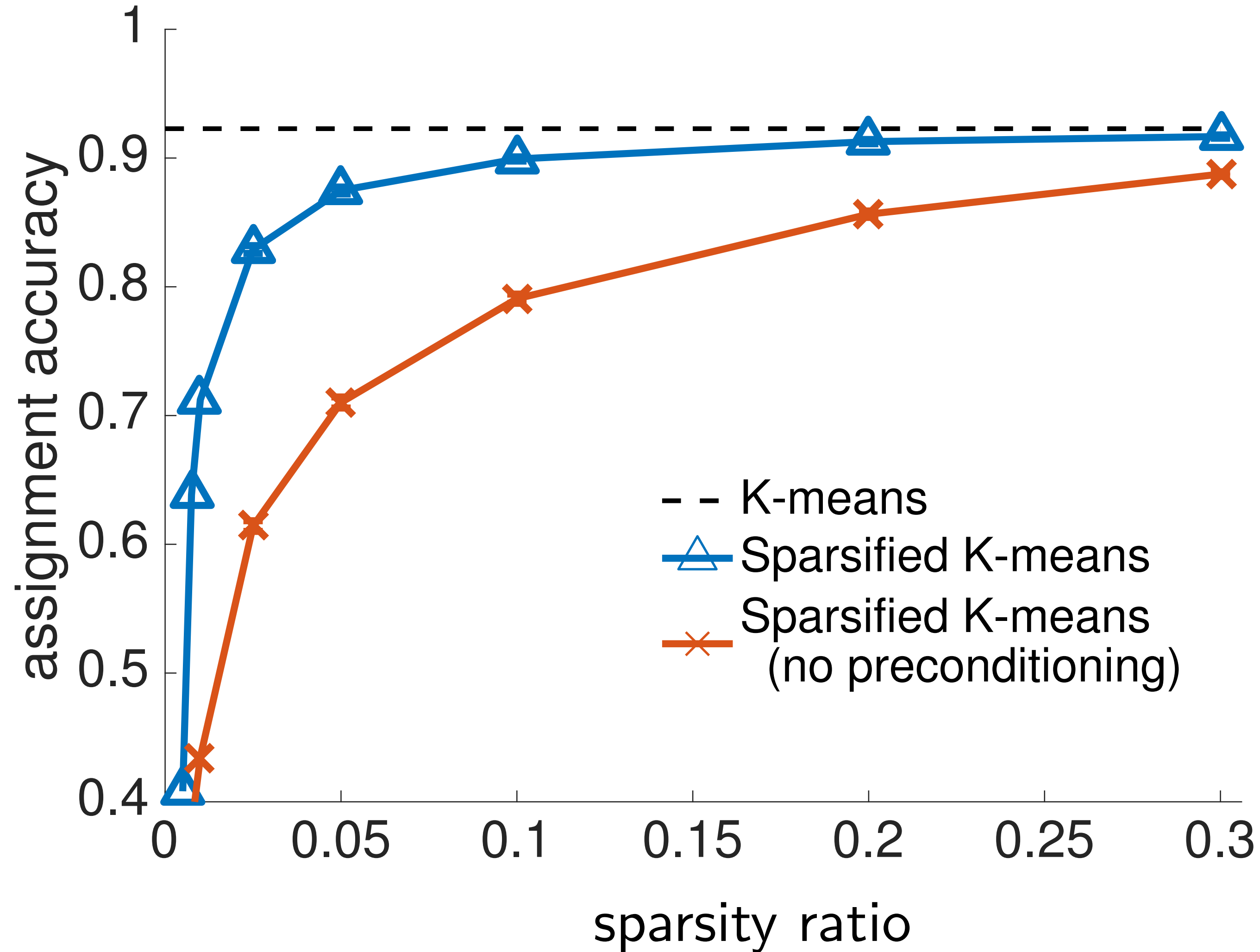
(d) sparsified K-means, 1 pass, preconditioned



(h) feature selection, 3 passes

Accuracy on MNIST

Now look at **classification** accuracy (averaged over 50 trials)



1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

Speed on “Infinite MNIST”

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering**
 - c. Tensor factorizations
 - d. Gradient-free optimization

“Infinite MNIST”, $p = 784$, but now $N = 9,631,605$ instead of 21,002

56 GB data, split into 58 chunks to keep RAM usage around 1 GB

Algorithm	Time to find assignments		Time to update all centers		Combined time	
	Absolute	Speedup	Absolute	Speedup	Absolute	Speedup
K-means	130.0s	1×	150.8s	1×	280.8s	1×
Sparsified	1.3s	100×	5.7s	26.4×	7.0s	40.1×

at 20x undersampling; accuracy is 89% (vs 92% for K-means)

OUTLINE

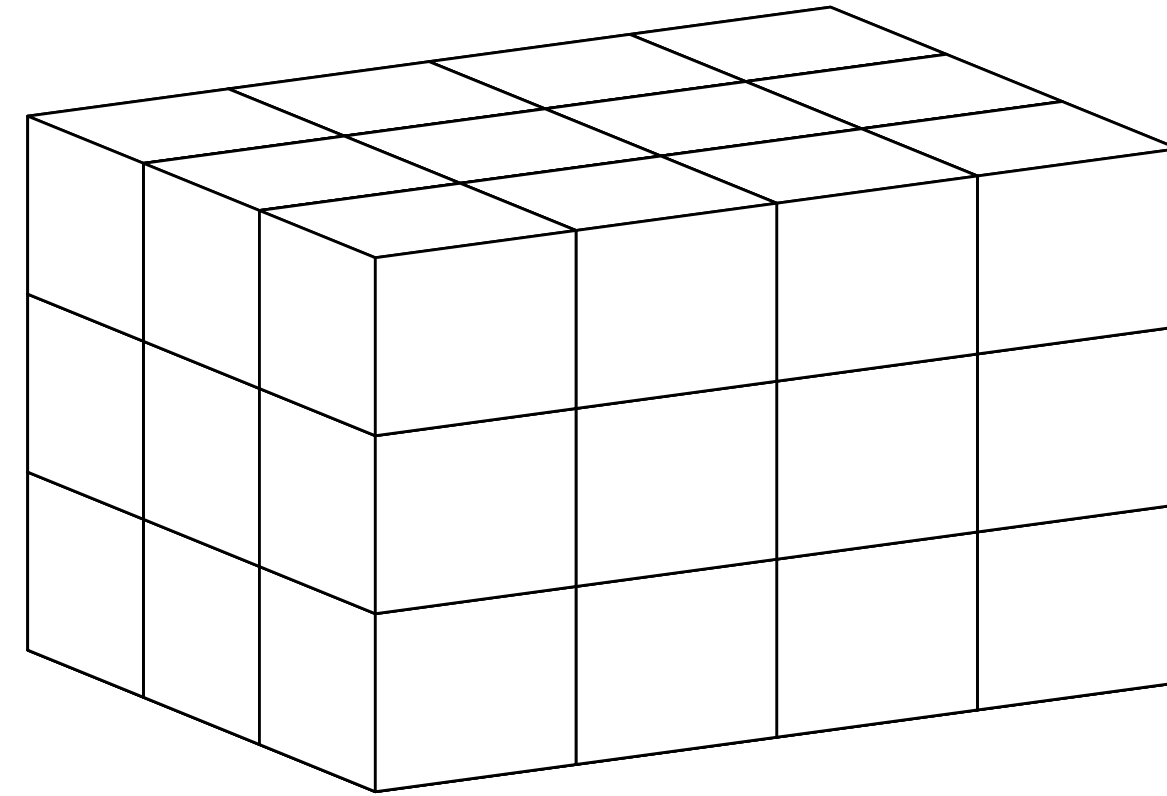
1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Tensor Background: what is a tensor?

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization



Change in notation temporarily



- ▶ A *tensor* $\mathcal{X} \in \mathbb{R}^{l_1 \times l_2 \times \dots \times l_N}$ is an array of dimension N , also called an N -way tensor or order N tensor.
- ▶ A matrix $\mathbf{X} \in \mathbb{R}^{l_1 \times l_2}$ is a 2-way tensor.
- ▶ A vector $\mathbf{x} \in \mathbb{R}^{l_1}$ is a 1-way tensor.
- ▶ A scalar $x \in \mathbb{R}$ is a 0-way tensor.

Examples of tensors

- 1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
- 2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

For example, [Kolda & Sun, '08] consider these datasets:

- ▶ **The Enron dataset** has size $1K \times 1K \times 1.1K \times 200$ and consists of elements of the form (user,user,keyword,day). Element (i, j, k, l) is 1 if user i sent an email to user j with keyword k on day l , 0 otherwise.
- ▶ **The DBLP dataset** has size $5K \times 1K \times 1K$ and consists of elements of the form (author,conference,keyword). Element (i, j, k) is 1 if author i published a paper at conference j containing the keyword k in the title, 0 otherwise.

These are examples of **sparse tensors**

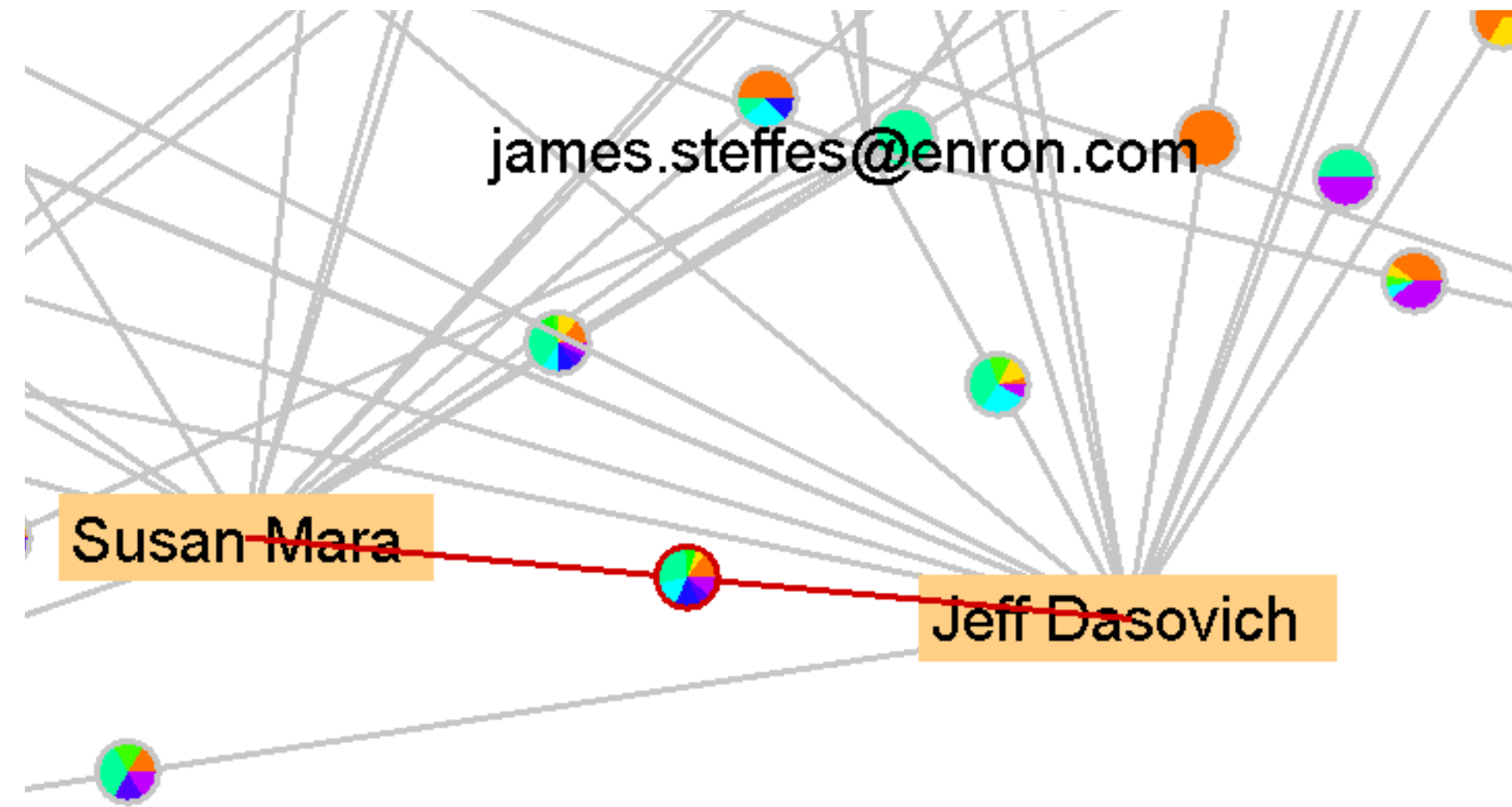


Image: <https://homes.cs.washington.edu/~jheer/projects/enron/>

Examples of tensors

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

For example, [Kolda & Sun, '08] consider these datasets:

- ▶ **The Enron dataset** has size $1K \times 1K \times 1.1K \times 1K$ and consists of elements of the form (user,user,keyword). Element (i,j,k,l) is 1 if user i sent an email to user j with keyword k on day l , 0 otherwise.
- ▶ **The DBLP dataset** has size $5K \times 1K \times 1K \times 1K$ and consists of elements of the form (author,conference,keyword). Element (i,j,k,l) is 1 if author i published a paper at conference j containing the keyword k in the title, 0 otherwise.

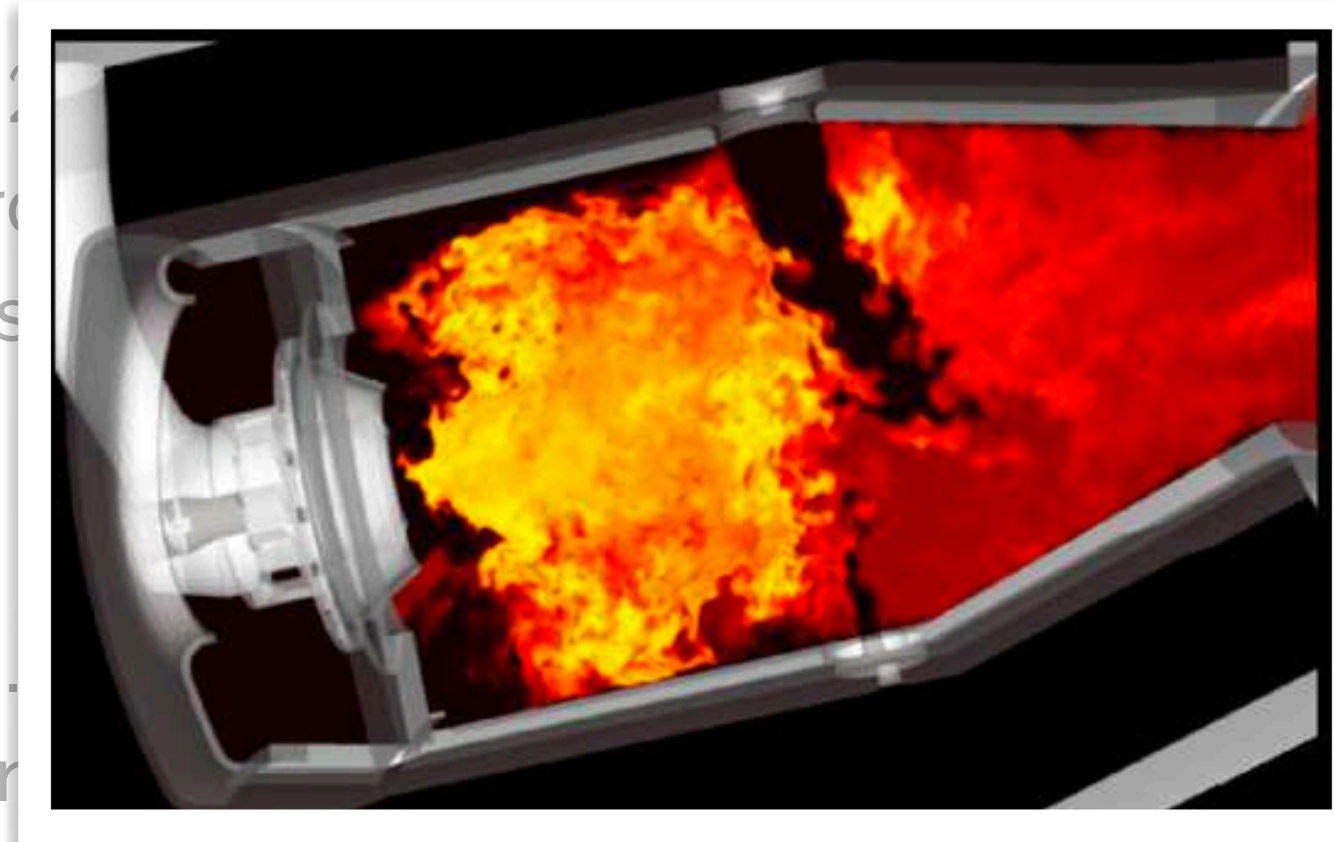
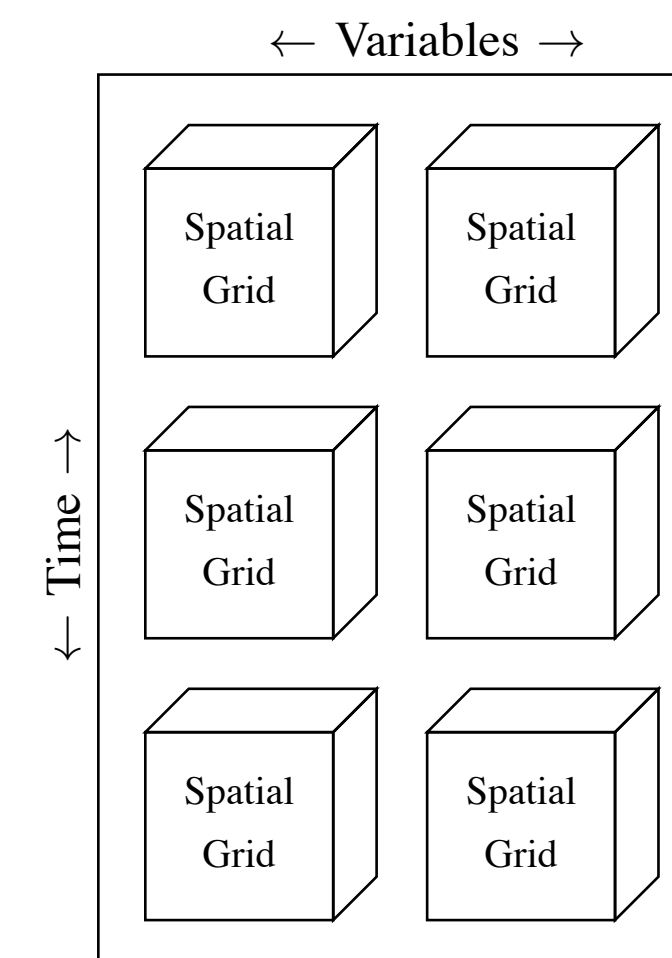


Image: Center for Turbulence Research (Stanford)

[Austin et al., '16] consider tensor data produced by high-fidelity combustion simulations with the following properties:

- ▶ 3-dimensional spatial grid with 512 points per dimension
 - ▶ 64 variables are tracked per grid point
 - ▶ 128 time steps
- $512 \times 512 \times 512 \times 64 \times 128$ (8 TB)



This is a **dense** tensor

... we'll want methods that work with both sparse and dense tensors

Examples of tensors

For example, [Kolda & Sun, '08] consider these datasets:

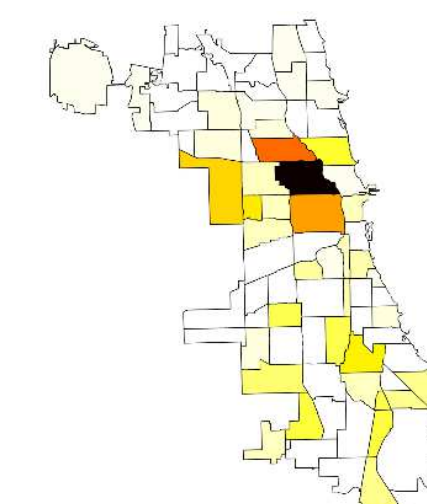
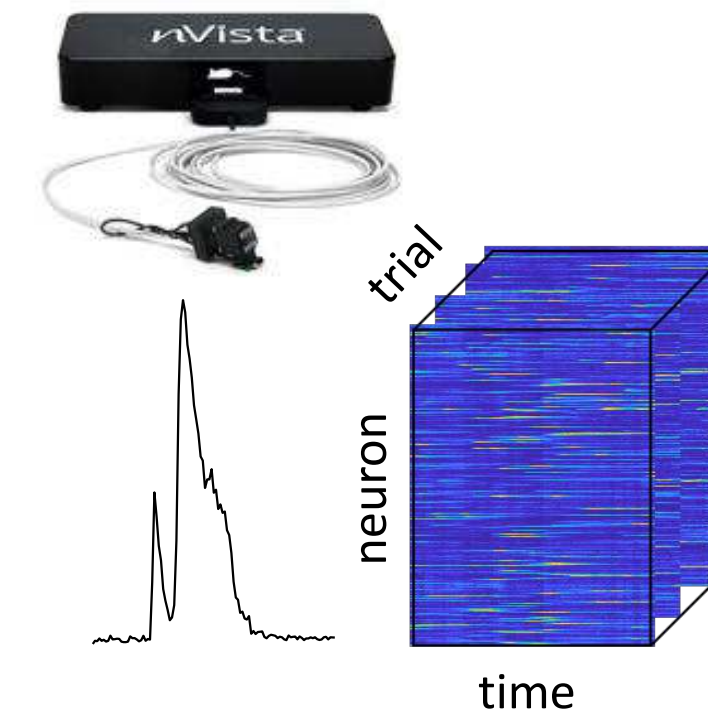
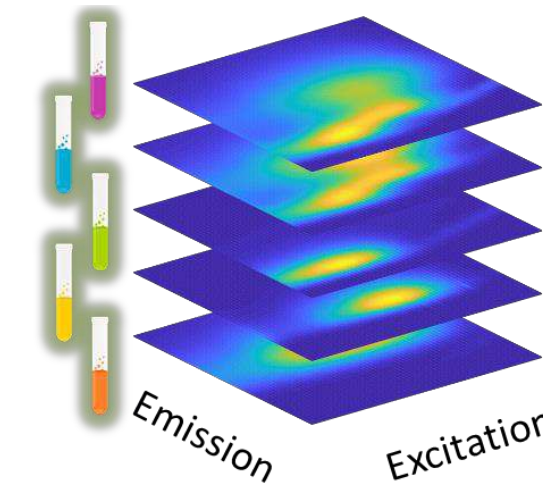
- ▶ **The Enron dataset** has size $1K \times 1K \times 1.1K \times 200$ and consists of elements of the form (user,user,keyword,day). Element (i, j, k, l) is 1 if user i sent an email to user j with keyword k on day l , 0 otherwise.
- ▶ **The DBLP dataset** has size $5K \times 1K \times 1K$ and consists of elements of the form (author,conference,keyword). Element (i, j, k) is 1 if author i published a paper at conference j containing the keyword k in the title, 0 otherwise.

[Austin et al., '16] consider tensor data produced by high-fidelity combustion simulations with the following properties:

- ▶ 3-dimensional spatial grid with 512 points per dimension
 - ▶ 64 variables are tracked per grid point
 - ▶ 128 time steps
- $512 \times 512 \times 512 \times 64 \times 128$ (8 TB)

Others:

- ▶ Chemometrics (emission x excitation x samples, for fluorescence spectroscopy)
- ▶ Neuroscience (neuron x time x trial/stimulus, for calcium imaging)
- ▶ Criminology (day x hour x location x crime, e.g., Chicago crime dataset)



1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

Images: Tamara Kolda

Examples of tensors

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

For example, [Kolda & Sun, '08] consider these datasets:

- ▶ **The Enron dataset** consists of Element (keyword k)
- ▶ **The DBLP dataset** elements (i, j, k) is containing

One-pass methods

All these tensors have a **time** or **sample** index

As time/samples increase, we don't want to store old ones

i.e., want a **streaming/one-pass method**

THIS IS A BIG MOTIVATION FOR SKETCHING

[Austin et al.,
combustion sim

- ▶ 3-dimensional
- ▶ 64 variables
- ▶ 128 time steps

$512 \times 512 \times 512 \times 64 \times 128$ (8 TB)

Others:

- ▶ Chemometrics (emission \times excitation \times samples, for fluorescence spectroscopy)
- ▶ Neuroscience (neuron \times time \times trial/stimulus, for calcium imaging)
- ▶ Criminology (day \times hour \times location \times crime, e.g., Chicago crime dataset)

Tensor operations 1: matricization

Matricization (i.e., flattening) turns a tensor \mathcal{X} into a matrix $\mathbf{X}_{(n)}$

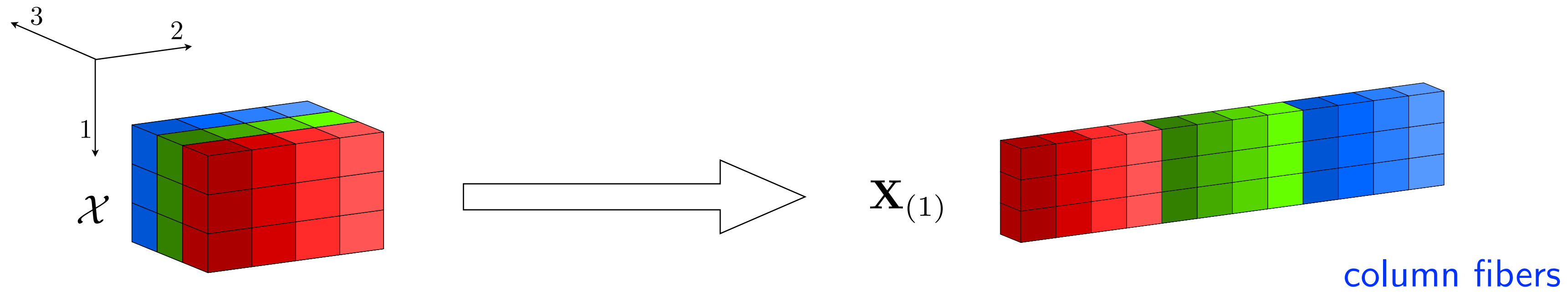
Same idea as turning a matrix (e.g., an image) into a vector

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

Tensor operations 1: matricization

Matricization (i.e., flattening) turns a tensor \mathcal{X} into a matrix $\mathbf{X}_{(n)}$

Same idea as turning a matrix (e.g., an image) into a vector

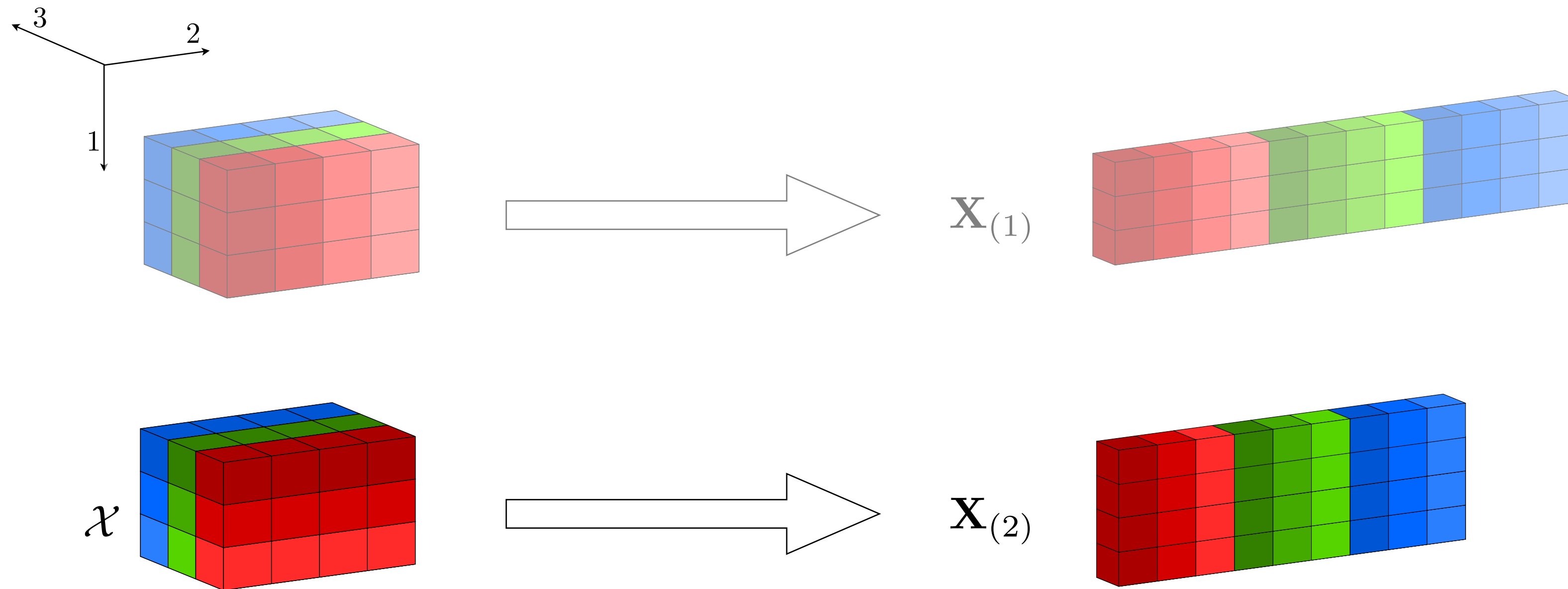


1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

Tensor operations 1: matricization

Matricization (i.e., flattening) turns a tensor \mathcal{X} into a matrix $\mathbf{X}_{(n)}$

Same idea as turning a matrix (e.g., an image) into a vector

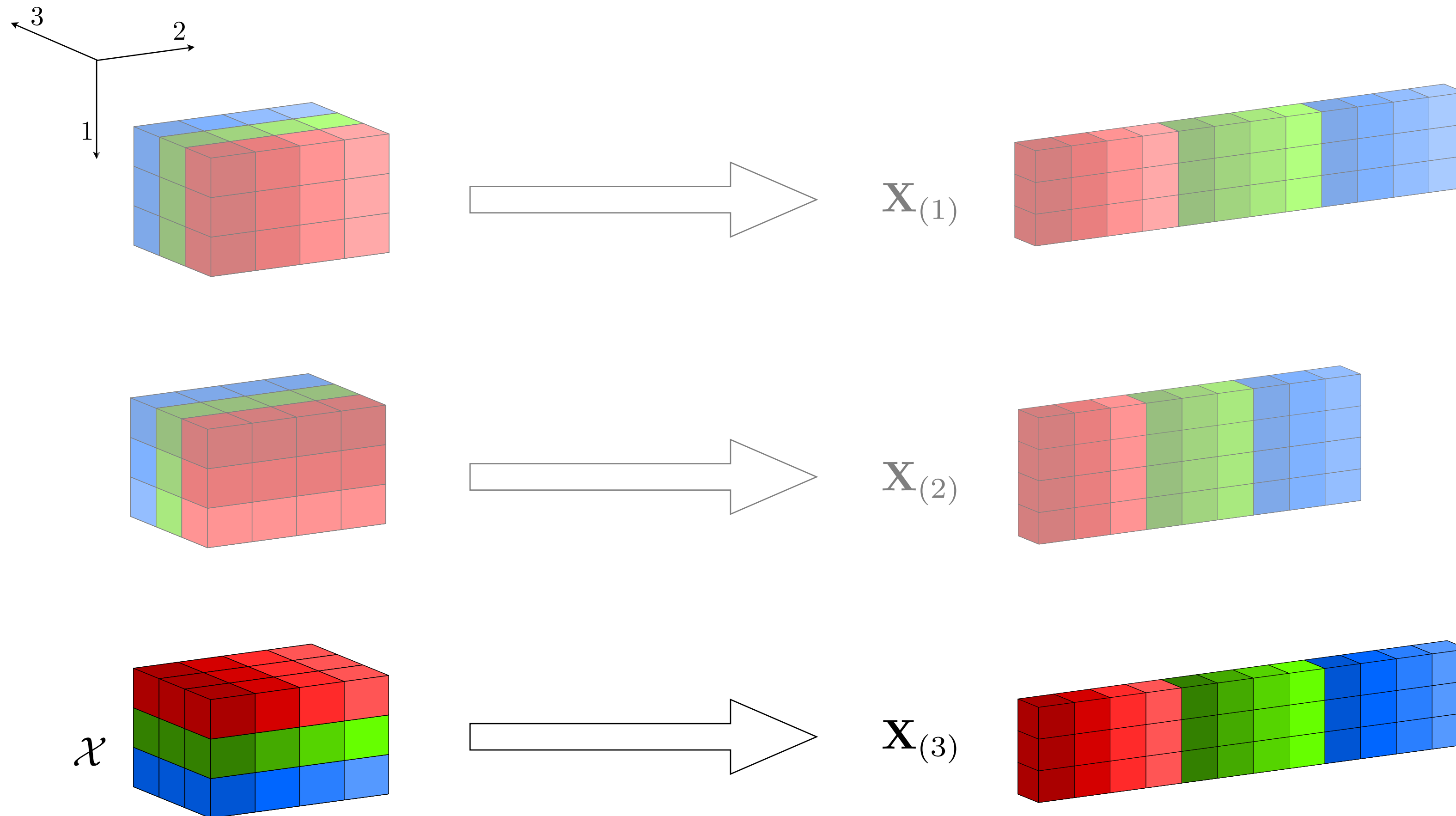


1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

Tensor operations 1: matricization

Matricization (i.e., flattening) turns a tensor \mathcal{X} into a matrix $\mathbf{X}_{(n)}$

Same idea as turning a matrix (e.g., an image) into a vector



“tube” fibers

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

Tensor operations 2: tensor-times-matrix

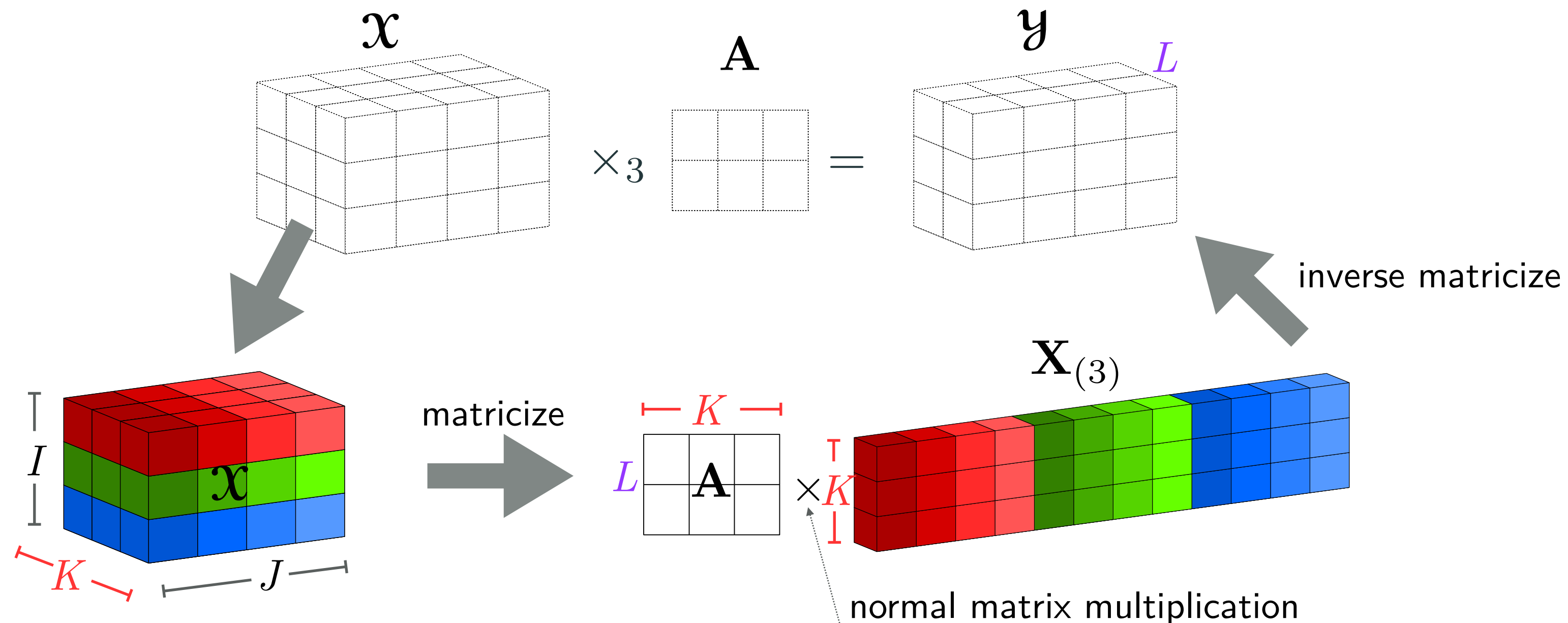
Example:

- Let $\mathcal{X} \in \mathbb{R}^{I \times J \times K}$ and $\mathbf{A} \in \mathbb{R}^{L \times K}$.
- We define $\mathcal{Y} \stackrel{\text{def}}{=} \mathcal{X} \times_3 \mathbf{A} \in \mathbb{R}^{I \times J \times L}$ elementwise by

$$y_{ijl} = \sum_{k=1}^K x_{ijk} a_{kl}.$$

- In matrix terms, $\mathbf{Y}_{(3)} = \mathbf{A} \mathbf{X}_{(3)}$.

Conveniently, order doesn't matter*
 $\mathcal{X} \times_1 \mathbf{A}_1 \times_2 \mathbf{A}_2 = \mathcal{X} \times_2 \mathbf{A}_2 \times_1 \mathbf{A}_1$



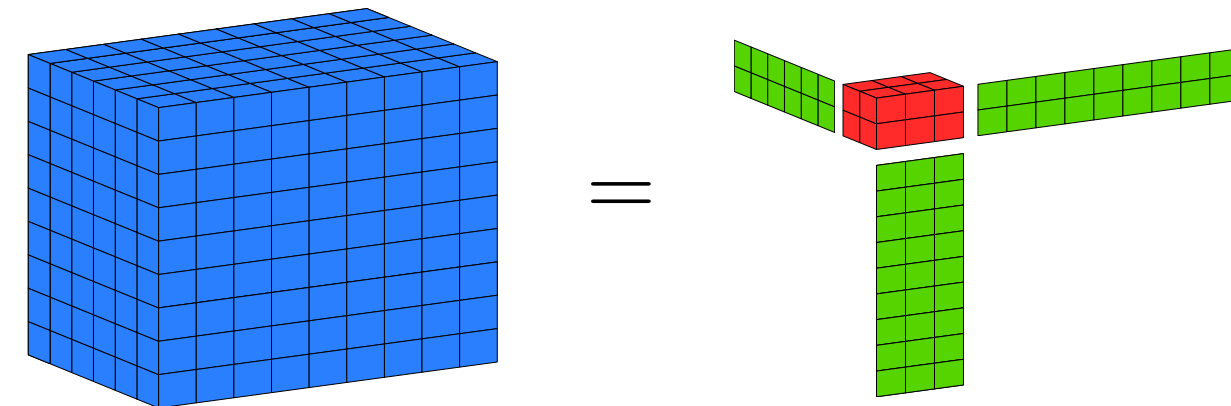
1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

The Tucker decomposition

A Tucker decomposition of $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$ is of the form

$$\mathcal{X} = \mathcal{G} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \cdots \times_N \mathbf{A}^{(N)} =: \llbracket \mathcal{G}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)} \rrbracket \text{ (shorthand notation)}$$

where $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \dots \times R_N}$ and each $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$.

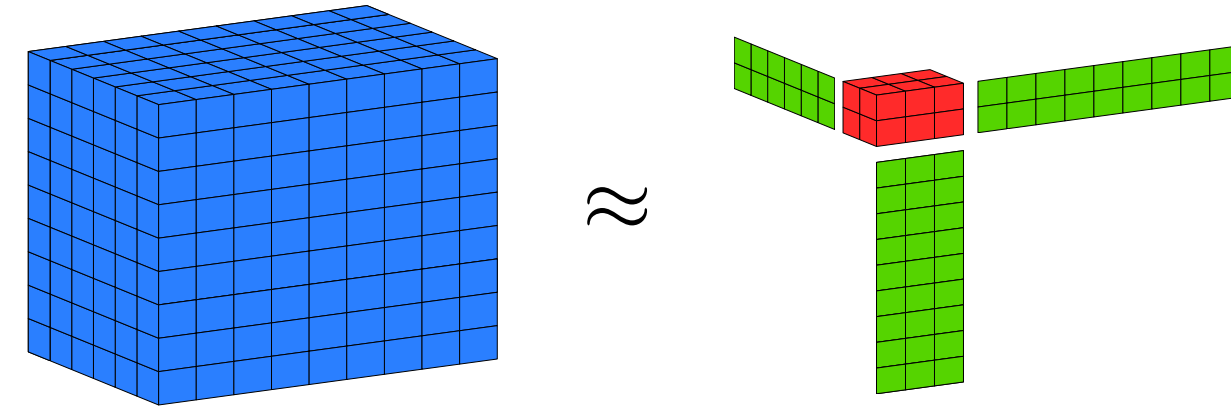


\mathcal{G} is the *core tensor* and $\mathbf{A}^{(n)}$, $n = 1, 2, \dots, N$, are *factor matrices*.

We say that \mathcal{X} is a rank- (R_1, R_2, \dots, R_N) tensor.

Applications of the Tucker decomposition

- **Data compression.** Storing the **red** and **green** objects require less storage than the full **blue** tensor.
- **Data analysis.** Use rows of factor matrices as feature vectors—*unsupervised learning*.



1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

Computing the Tucker decomposition

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

Given a data tensor $\mathcal{X} \in \mathbb{R}^{I_1 \times I_2 \times \dots \times I_N}$, we can find a decomposition $\mathcal{X} \approx \llbracket \mathcal{G}; \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)} \rrbracket$ by solving the optimization problem

$$\min_{\mathcal{G}, \mathbf{A}^{(1)}, \dots, \mathbf{A}^{(N)}} \|\mathcal{X} - \llbracket \mathcal{G}; \mathbf{A}^{(1)}, \mathbf{A}^{(2)}, \dots, \mathbf{A}^{(N)} \rrbracket\|^2, \quad (4)$$

where $\mathcal{G} \in \mathbb{R}^{R_1 \times R_2 \times \dots \times R_N}$ and $\mathbf{A}^{(n)} \in \mathbb{R}^{I_n \times R_n}$ for $n = 1, 2, \dots, N$.

- Nonlinear!
- A common approach to this issue is to use *alternating least squares (ALS)*: Minimize with respect to each $\mathbf{A}^{(n)}$ and \mathcal{G} one at a time. Repeat!
- With ALS, each subproblem is linear and easily solved.

i.e., Gauss-Siedel style

ALS same as *higher-order orthogonal iteration (HOOI)*
and if you do just one iteration, it's *higher-order SVD (HOSVD)*

Computing the Tucker decomposition

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

The algorithm looks like this. Repeat the following until convergence:

Repeat until convergence

1. For $n = 1, \dots, N$, update (10)

$$\mathbf{A}^{(n)} = \arg \min_{\mathbf{A} \in \mathbb{R}^{I_n \times R_n}} \left\| \left(\bigotimes_{\substack{i=1 \\ i \neq n}}^N \mathbf{A}^{(i)} \right) \mathbf{G}_{(n)}^\top \mathbf{A}^\top - \mathbf{X}_{(n)}^\top \right\|_F^2. \quad (11)$$

2. Update $\mathcal{G} = \mathbf{X} \times_1 \mathbf{A}^{(1)\top} \times_2 \mathbf{A}^{(2)\top} \dots \times_N \mathbf{A}^{(N)\top}$. (12)

$$= \arg \min_{\mathbf{z} \in \mathbb{R}^{R_1 \times \dots \times R_N}} \left\| \left(\bigotimes_{i=1}^N \mathbf{A}^{(i)} \right) \mathbf{z}_{(\cdot)} - \mathbf{x}_{(\cdot)} \right\|_2^2.$$

(Line 12 follows if the factor matrices are orthogonalized)

Make extensive use of this identity:

$$\mathbf{y} = \mathbf{X} \times_1 \mathbf{A}^{(1)} \times_2 \mathbf{A}^{(2)} \dots \times_N \mathbf{A}^{(N)} \Leftrightarrow$$

$$\mathbf{Y}_{(n)} = \mathbf{A}^{(n)} \mathbf{X}_{(n)} \left(\mathbf{A}^{(N)} \otimes \dots \otimes \mathbf{A}^{(n+1)} \otimes \mathbf{A}^{(n-1)} \otimes \dots \otimes \mathbf{A}^{(1)} \right)^\top$$

Computing the Tucker decomposition

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

The algorithm looks like this. Repeat the following until convergence:

1. For $n = 1, \dots, N$, update (10)

$$\mathbf{A}^{(n)} = \arg \min_{\mathbf{A} \in \mathbb{R}^{I_n \times R_n}} \left\| \left(\bigotimes_{\substack{i=1 \\ i \neq n}}^N \mathbf{A}^{(i)} \right) \mathbf{G}_{(n)}^\top \mathbf{A}^\top - \mathbf{X}_{(n)}^\top \right\|_F^2. \quad (11)$$

2. Update $\mathcal{G} = \mathbf{X} \times_1 \mathbf{A}^{(1)\top} \times_2 \mathbf{A}^{(2)\top} \cdots \times_N \mathbf{A}^{(N)\top}$. (12)

Issues when dealing with large, sparse tensors:

- The **Kronecker product matrix** is huge.
- The **series of tensor-times-matrix products** is very costly, and can require lots of additional memory when \mathcal{X} is sparse.
- We address this problem by using *TensorSketch*.

Aside: we initially thought that **solving** the least-squares problem would be an issue, and something to fix with sketching, but this is not exactly the case

(partly because of many clever tricks due to orthogonality of \mathbf{A})

Our modified algorithm: Tucker-TS

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

We simply sketch each of these least-squares problems with TensorSketch operators $\mathbf{S}^{(n)}$, $n = 1, \dots, N + 1$, which are defined at the start of the algorithm:

Repeat until convergence

1. For $n = 1, \dots, N$, update

$$\mathbf{A}^{(n)} = \arg \min_{\mathbf{A} \in \mathbb{R}^{I_n \times R_n}} \left\| \left(\mathbf{S}^{(n)} \otimes_{\substack{i=N \\ i \neq n}}^1 \mathbf{A}^{(i)} \right) \mathbf{G}_{(n)}^\top \mathbf{A}^\top - \mathbf{S}^{(n)} \mathbf{X}_{(n)}^\top \right\|_F^2. \quad (18)$$

(and orthogonalize)

2. Update

$$\mathcal{G} = \arg \min_{\mathbf{z} \in \mathbb{R}^{R_1 \times \dots \times R_N}} \left\| \left(\mathbf{S}^{(N+1)} \otimes_{i=N}^1 \mathbf{A}^{(i)} \right) \mathbf{z}_{(\cdot)} - \mathbf{S}^{(N+1)} \mathbf{x}_{(\cdot)} \right\|_2^2. \quad (19)$$

sketched once at very beginning

Tricks:

- don't draw new sketches each iteration
- the TensorSketches have CountSketches/ hashes in common $\mathbf{S}^{(n)}, \mathbf{S}^{(n')}$
- G update solved via conjugate gradient (CG) algo, and can prove it is well-conditioned
- advanced tricks and variants in the paper

Numerical experiment: sparse 3D tensor

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

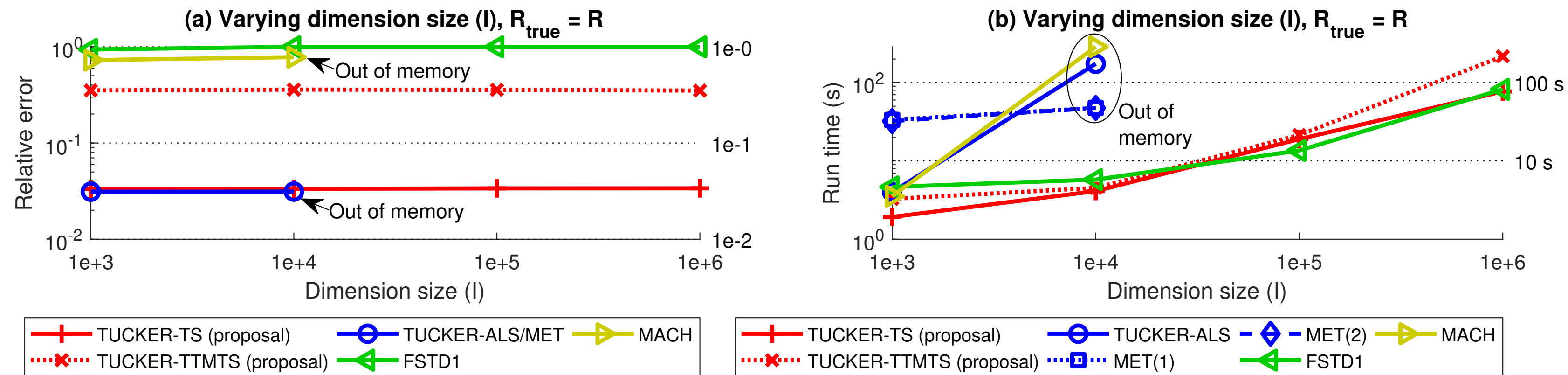


Figure 3: Relative error and run time for random sparse 3-way tensors with varying dimension size I and $\text{nnz}(\mathcal{Y}) \approx 1e+6$. Both the true and target ranks are $(10, 10, 10)$.

Tucker-TS approximates least-squares like this:

$$\begin{aligned} \operatorname{argmin}_x \frac{1}{2} \|Ax - b\|_2^2 &\approx \operatorname{argmin}_x \frac{1}{2} \|\Phi(Ax - b)\|_2^2 \\ &= (A^\top \Phi^\top \Phi A)^{-1} A^\top \Phi^\top \Phi b \end{aligned}$$

Tucker-TTMTS approximates least-squares like this:

$$\begin{aligned} \operatorname{argmin}_x \frac{1}{2} \|Ax - b\|_2^2 &= (A^\top A)^{-1} A^\top b \\ &\approx (A^\top A)^{-1} A^\top \Phi^\top \Phi b \end{aligned}$$

Video

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. **Tensor factorizations**
 - d. Gradient-free optimization

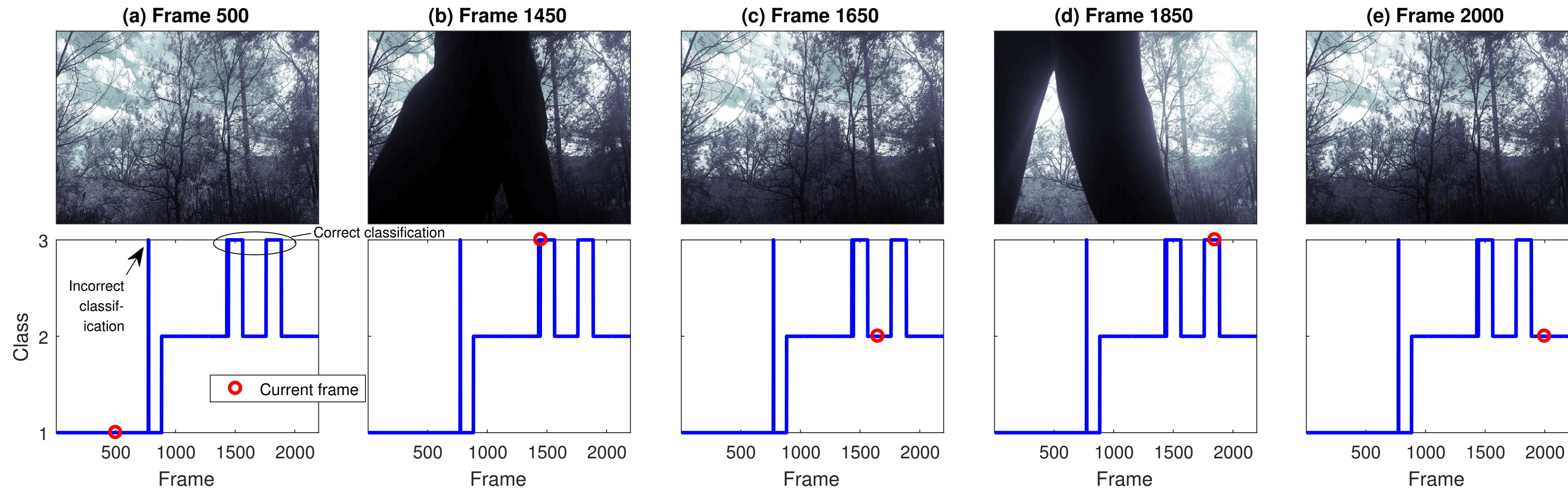


Figure 6: Five sample frames with their assigned classes. The frames (b) and (d) contain a disturbance.

38 GB video (can't all fit into RAM): 2,200 frames, each of size 1,080 by 1,980 pixels

Compute rank (10,10,10) factorization, 30 iterations max

Find 3 factor matrices — take the one corresponding to time and run k-means clustering

(used Tucker-TTMTS variant)

OUTLINE

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

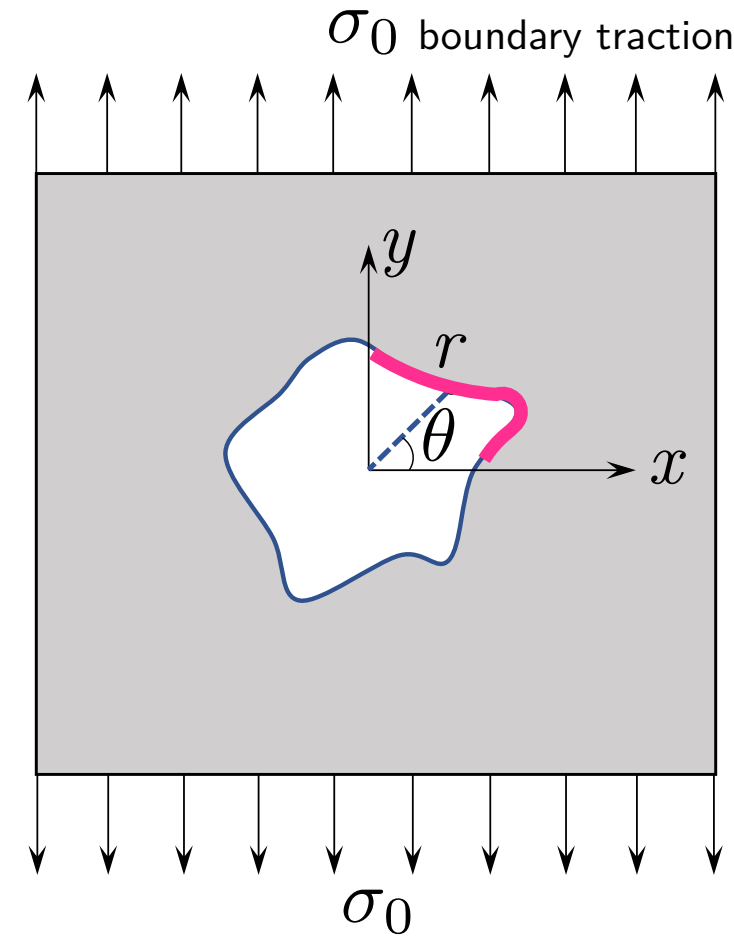
Motivating Application: shape optimization

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

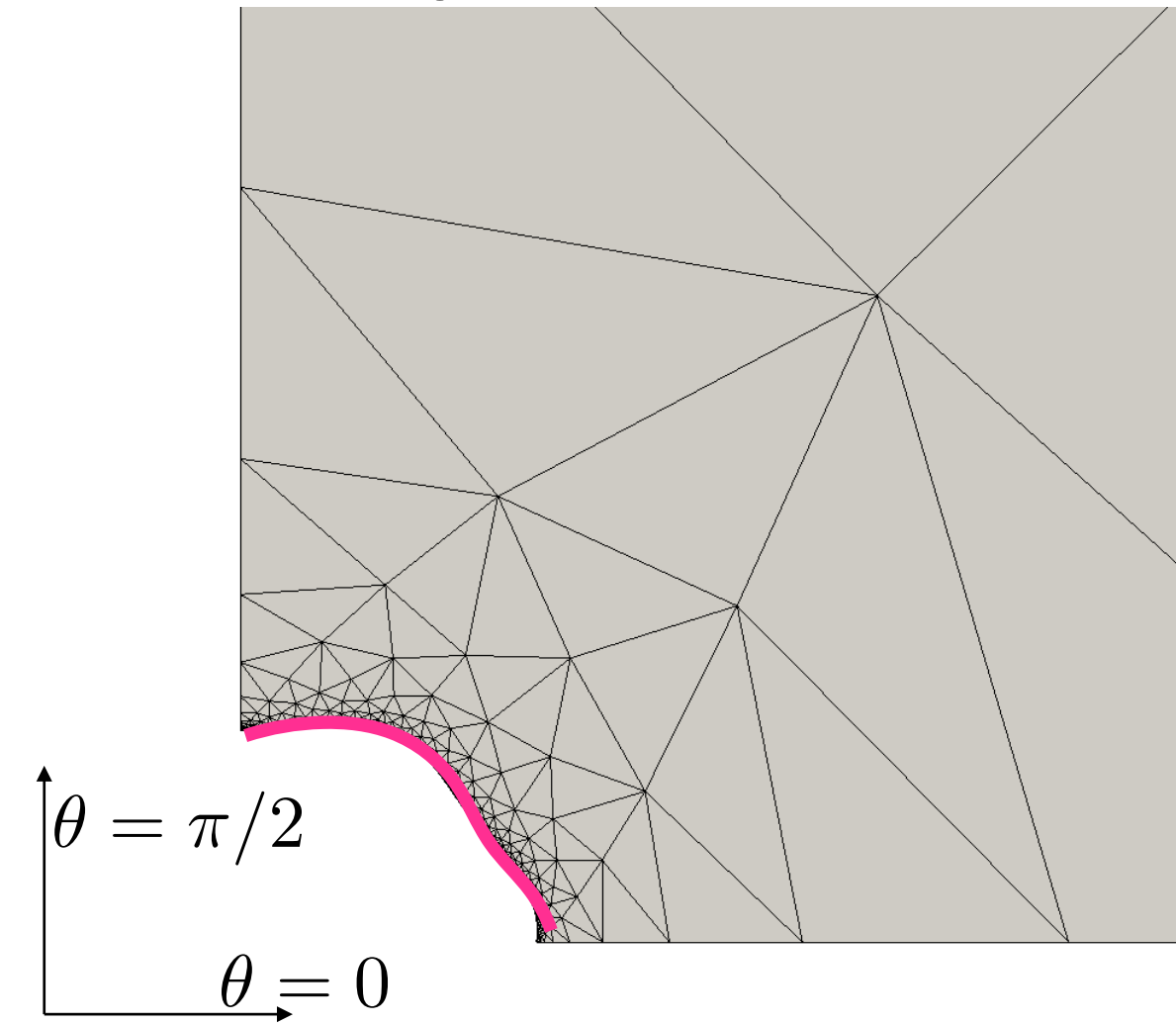
Forward problem: find vertical stress σ_y

linear elasticity PDE

(boundary conditions depend on shape of the hole)



conforming finite element mesh, used in FEniCS



Change in notation temporarily

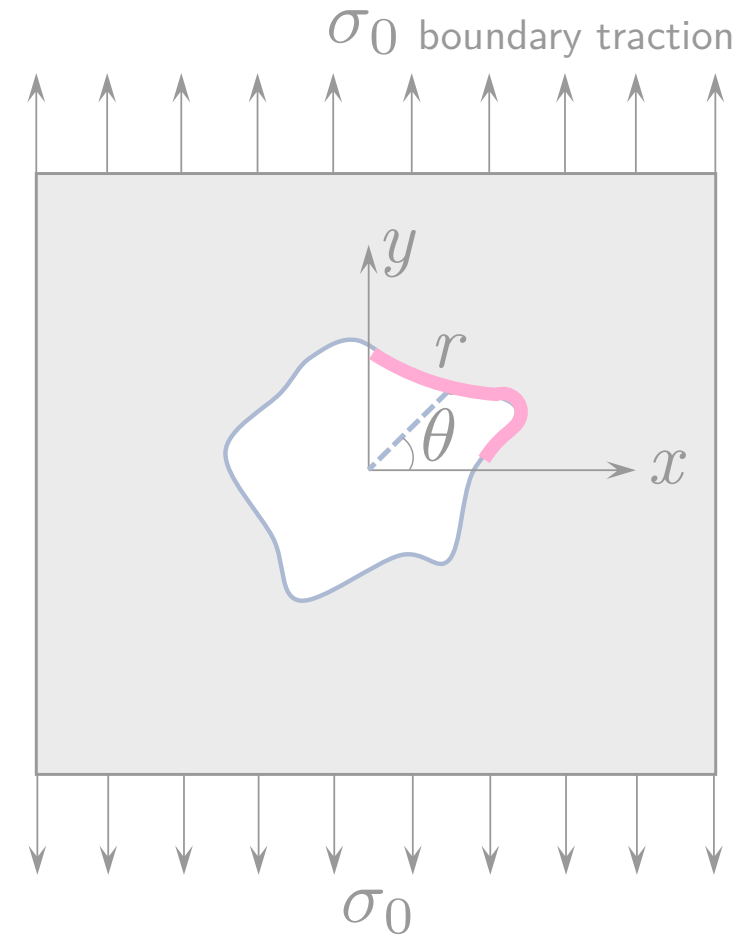
Motivating Application: shape optimization

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

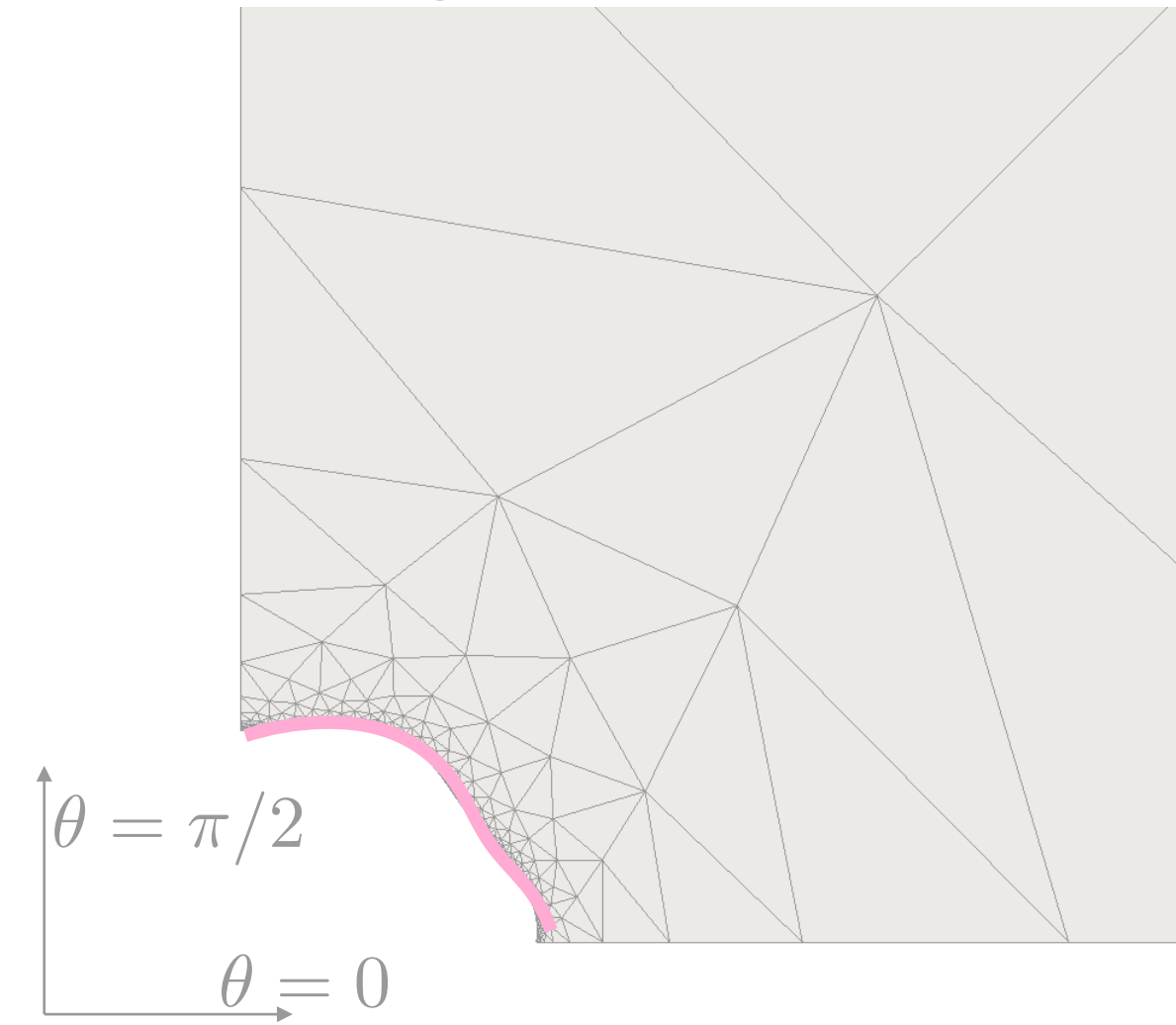
Forward problem: find vertical stress σ_y

linear elasticity PDE

(boundary conditions depend on shape of the hole)



conforming finite element mesh, used in FEniCS



Inverse problem: what shape minimizes the vertical stress?

parameterize the shape of the hole as follows, which automatically enforces a constant area constraint

$$r(\theta) = \frac{1}{2\pi} + \delta \sum_{k=0}^{d/2-1} \frac{1}{\sqrt{2k+1}} (x_{2k+1} \sin((2k+1) \cdot \theta) + x_{2k+2} \cos((2k+1) \cdot \theta))$$

optimization variable:

$$x \in \mathbb{R}^d$$

Generic PDE-constrained optimization

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

implicitly saying that u solves the PDE

$$\min_{u,x} \mathcal{L}(u) \quad \text{subject to} \quad \phi(u,x) = 0$$

Examples:

$$\dot{u} = \Delta u, \quad u(0) = h \quad \text{“}x\text{” is the initial condition}$$

$$\ddot{u} = c^2 \Delta u, \quad u(0) = h \quad \text{“}x\text{” is a parameter}$$

$$\Delta u = 0, \quad u(\Gamma) = h \quad \text{“}x\text{” is the boundary condition}$$

$\mathcal{L}(u)$ is the loss which penalizes something like:

- deviation from observations
 - drag
 - mass
 - cost of materials
 - compliance
- etc.

Generic PDE-constrained optimization

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

implicitly saying that u solves the PDE

$$\min_{u,x} \mathcal{L}(u) \quad \text{subject to} \quad \phi(u,x) = 0$$

$$\phi(u,x) = 0 \quad \implies \quad u = u(x)$$

Rewrite:

$$\min_x f(x) \stackrel{\text{def}}{=} \mathcal{L}(u(x))$$

... but finding the gradient is tricky:

$$\nabla f(x) = \frac{\partial \mathcal{L}}{\partial u} \cdot \frac{\partial u}{\partial x}$$

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time ($\sim 4x$) as a function evaluation

- ... so if we can evaluate $f(x)$ numerically, we can find the gradient
- (this applies if $f : \mathbb{R}^d \rightarrow \mathbb{R}$; $f : \mathbb{R}^d \rightarrow \mathbb{R}^q$ with $q \gg 1$ is another story)

Note: we're assuming derivative exists, just hard to actually calculate
This is *not* non-smooth optimization

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time ($\sim 4x$) as a function evaluation

✗ Requires specialized/restricted libraries/code (`dolfin-adjoint/FEniCS`, `autograd`)

`Jax`, `PyTorch`, `Tensorflow`, ...

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time ($\sim 4x$) as a function evaluation

✗ Requires specialized/restricted libraries/code (`dolfin-adjoint/FEniCS`, `autograd`)
✗ Adjoint state method requires a method to solve adjoint PDE

(and have to parallelize for HPC)

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time ($\sim 4x$) as a function evaluation

✗ Requires specialized/restricted libraries/code (`dolfin-adjoint`/`FEniCS`, `autograd`)

✗ Adjoint state method requires a method to solve adjoint PDE

- difficult to maintain in large code bases, e.g., 4D-var for weather codes

✗ Slow if used for intermediate calculations involving some

- e.g., seismic inversion with many observations

$$f : \mathbb{R}^d \rightarrow \mathbb{R}^q$$

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time ($\sim 4x$) as a function evaluation

✗ Requires specialized/restricted libraries/code (`dolfin-adjoint`/`FEniCS`, `autograd`)

✗ Adjoint state method requires a method to solve adjoint PDE

- difficult to maintain in large code bases, e.g., 4D-var for weather codes

✗ Slow if used for intermediate calculations involving some

- e.g., seismic inversion with many observations

$$f : \mathbb{R}^d \rightarrow \mathbb{R}^q$$

✗ Possible memory explosion

- e.g., time-dependent problems. Check-pointing schemes somewhat helpful

Example: hyper-parameter optimization in deep learning

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time ($\sim 4x$) as a function evaluation

✗ Requires specialized/restricted libraries/code (`dolfin-adjoint`/`FEniCS`, `autograd`)

✗ Adjoint state method requires a method to solve adjoint PDE

- difficult to maintain in large code bases, e.g., 4D-var for weather codes

✗ Slow if used for intermediate calculations involving some

- e.g., seismic inversion with many observations

$$f : \mathbb{R}^d \rightarrow \mathbb{R}^q$$

✗ Possible memory explosion

- e.g., time-dependent problems. Check-pointing schemes somewhat helpful

✗ Requires access to original source code

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Why not just find gradients automatically?

✓ The **adjoint state method** and **reverse-mode automatic differentiation** can automatically calculate gradients in about the same time ($\sim 4x$) as a function evaluation

✗ Requires specialized/restricted libraries/code (`dolfin-adjoint`/`FEniCS`, `autograd`)

✗ Adjoint state method requires a method to solve adjoint PDE

- difficult to maintain in large code bases, e.g., 4D-var for weather codes

✗ Slow if used for intermediate calculations involving some

- e.g., seismic inversion with many observations

$$f : \mathbb{R}^d \rightarrow \mathbb{R}^q$$

✗ Possible memory explosion

- e.g., time-dependent problems. Check-pointing schemes somewhat helpful

✗ Requires access to original source code

✗ Assumes a computational structure

- inapplicable for physical observations (wind farms; *rollout* in AI)

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Baseline Algorithms (for comparison)

$$f : \mathbb{R}^d \rightarrow \mathbb{R}$$

Algorithm Gradient Descent via Finite Differences

- 1: **for** $k = 1, 2, \dots$ **do**
 - 2: Estimate $g_k \approx \nabla f(x_k)$ ▷ Use finite differences
 - 3: $x_{k+1} \leftarrow x_k - \eta_k g_k$ ▷ For appropriate step-size η_k
-

✓ ignoring finite-difference error, enjoys well-understood convergence

✗ requires $d+1$ function evaluations per iter.

Why not use traditional Derivative Free Optimization (DFO) methods?

Answer: most classical DFO methods don't scale well with dimension

Note: there are many other DFO methods... but we won't discuss in this talk

From heuristics to theoretically based, from local to global

- Heuristics: Nelder-Mead, genetic algorithms, particle swarm optimization
 - e.g., CMA-ES, Covariance matrix adaptation evolution strategy
- Simulated Annealing (typical as heuristic)
- Bayesian Optimization
 - Model is Gaussian Process, with acquisition function (exploration/exploitation tradeoff)
- DFO-TR (Trust-region)
 - Model is polynomial

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Baseline Algorithms (for comparison)

$$f : \mathbb{R}^d \rightarrow \mathbb{R}$$

Algorithm Gradient Descent via Finite Differences

- 1: **for** $k = 1, 2, \dots$ **do**
 - 2: Estimate $g_k \approx \nabla f(x_k)$ ▷ Use finite differences
 - 3: $x_{k+1} \leftarrow x_k - \eta_k g_k$ ▷ For appropriate step-size η_k
-

✓ ignoring finite-difference error, enjoys well-understood convergence

✗ requires $d+1$ function evaluations per iter.

Algorithm Randomized Coordinate Descent (CD)

- 1: **for** $k = 1, 2, \dots$ **do**
 - 2: Choose $j \in \{1, 2, \dots, d\}$ at random
 - 3: $g_k = e_j e_j^T \nabla f(x_k)$
 - 4: $x_{k+1} \leftarrow x_k - \eta_k g_k$ ▷ For appropriate step-size η_k (or exact minimization... depends on structure)
-

✓ just 1 function evaluation per iteration

✗ poor convergence properties, slow rates

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Stochastic Subspace Descent

directional derivative $qq^T \nabla f(x_k) = \left(\lim_{h \rightarrow 0} \frac{f(x_k + h \cdot q) - f(x_k)}{h} \right) q$

Assume we can compute this!

e.g.,

- 1) forward finite diff
- 2) forward-mode AD

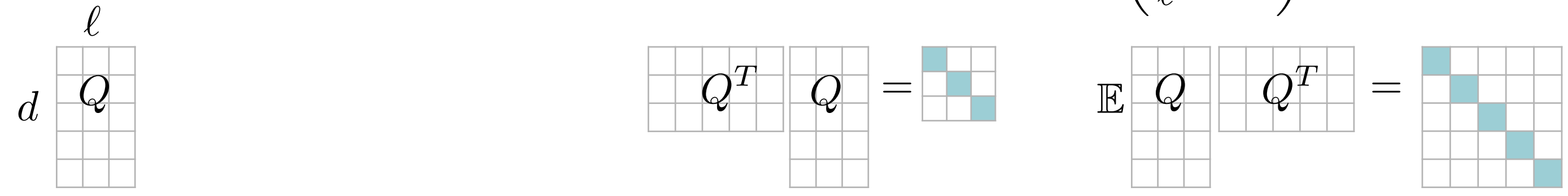
1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Stochastic Subspace Descent

- 1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
- 2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

directional derivative $qq^T \nabla f(x_k) = \left(\lim_{h \rightarrow 0} \frac{f(x_k + h \cdot q) - f(x_k)}{h} \right) q$

$Q = [q_1, q_2, \dots, q_\ell] \sim \text{Haar}(d \times \ell)$ $Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E} \left(\frac{d}{\ell} Q Q^T \right) = I_{d \times d}$



One benefit: in the limit $\ell = d$, $Q Q^T = I_{d \times d}$, and so we'll recover the full gradient (for Gaussians, this is only true in expectation)

Stochastic Subspace Descent

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

directional derivative $qq^T \nabla f(x_k) = \left(\lim_{h \rightarrow 0} \frac{f(x_k + h \cdot q) - f(x_k)}{h} \right) q$

$$Q = [q_1, q_2, \dots, q_\ell] \sim \text{Haar}(d \times \ell) \quad Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E} \left(\frac{d}{\ell} Q Q^T \right) = I_{d \times d}$$

Algorithm “Stochastic Subspace Descent” (SSD)

- 1: **for** $k = 1, 2, \dots$ **do**
 - 2: Draw $Q \sim \text{Haar}(d \times \ell)$ **or any generic SSD**
 - 3: $x_{k+1} \leftarrow x_k - \eta_k \frac{d}{\ell} Q Q^T \nabla f(x_k)$
-

Generic SSD $Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E} \left(\frac{d}{\ell} Q Q^T \right) = I_{d \times d}$

Both **Haar** and **Coordinate Descent** methods are valid generic SSD

Stochastic Subspace Descent

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

directional derivative $qq^T \nabla f(x_k) = \left(\lim_{h \rightarrow 0} \frac{f(x_k + h \cdot q) - f(x_k)}{h} \right) q$

$$Q = [q_1, q_2, \dots, q_\ell] \sim \text{Haar}(d \times \ell) \quad Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E} \left(\frac{d}{\ell} Q Q^T \right) = I_{d \times d}$$

Algorithm “Stochastic Subspace Descent” (SSD)

- 1: **for** $k = 1, 2, \dots$ **do**
 - 2: Draw $Q \sim \text{Haar}(d \times \ell)$
 - 3: $x_{k+1} \leftarrow x_k - \eta_k \frac{d}{\ell} Q Q^T \nabla f(x_k)$
-

We call Q a “Haar” distributed r.v. (i.e., the Haar measure over orthogonal matrices), but really care about $Q Q^T$ which is a projection matrix (onto $\text{col}(Q)$).

We get Q via Gram-Schmidt (or appropriately modified QR) on a Gaussian G , and note $\text{col}(Q) = \text{col}(G)$ w.p. 1, so our update is equivalent to

$$x_{k+1} \leftarrow x_k - \eta_k \frac{d}{\ell} \mathcal{P}_{\text{col}(G)} (\nabla f(x_k))$$

and hence the term “stochastic subspace”.

First theory results (for *generic SSD*)

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**



Theorem (Kozak, Becker, Tenorio, Doostan '20)

Assume: minimizer attained, gradient Lipschitz, stepsize η_k chosen appropriately.

$$\eta = \frac{\ell}{d} \frac{1}{L}$$

constant L constant μ

1. If f is **convex**,

$$\mathbb{E}f(x_k) - f^* \leq 2 \frac{d}{\ell} \frac{L}{k} R^2 = \mathcal{O}(k^{-1})$$

2. If f is **not convex** but satisfies the **Polyak-Lojasiewicz** inequality,

$$\mathbb{E}f(x_k) - f^* \leq \rho^k (f(x_0) - f^*) = \mathcal{O}(\rho^k) \quad \text{and} \quad f(x_k) \xrightarrow{\text{a.s.}} f^*$$

3. If f is **strongly convex**, statements of 2 above hold, and also

$$x_k \xrightarrow{\text{a.s.}} \operatorname{argmin}_x f(x)$$

4. If f is **not convex** (nor PL),

$$\min_{k' \in \{0, \dots, k\}} \mathbb{E} \|\nabla f(x_{k'})\|^2 \leq \frac{d}{\ell} \frac{2L(f(x_0) - f^*)}{k+1}$$

SSD is a special type of SGD, but results are much better than generic SGD analysis

$$f^* \stackrel{\text{def}}{=} \min_x f(x)$$

$$\rho = 1 - \frac{\mu \ell}{L d}$$

d = ambient dimension

ℓ = # directional derivs

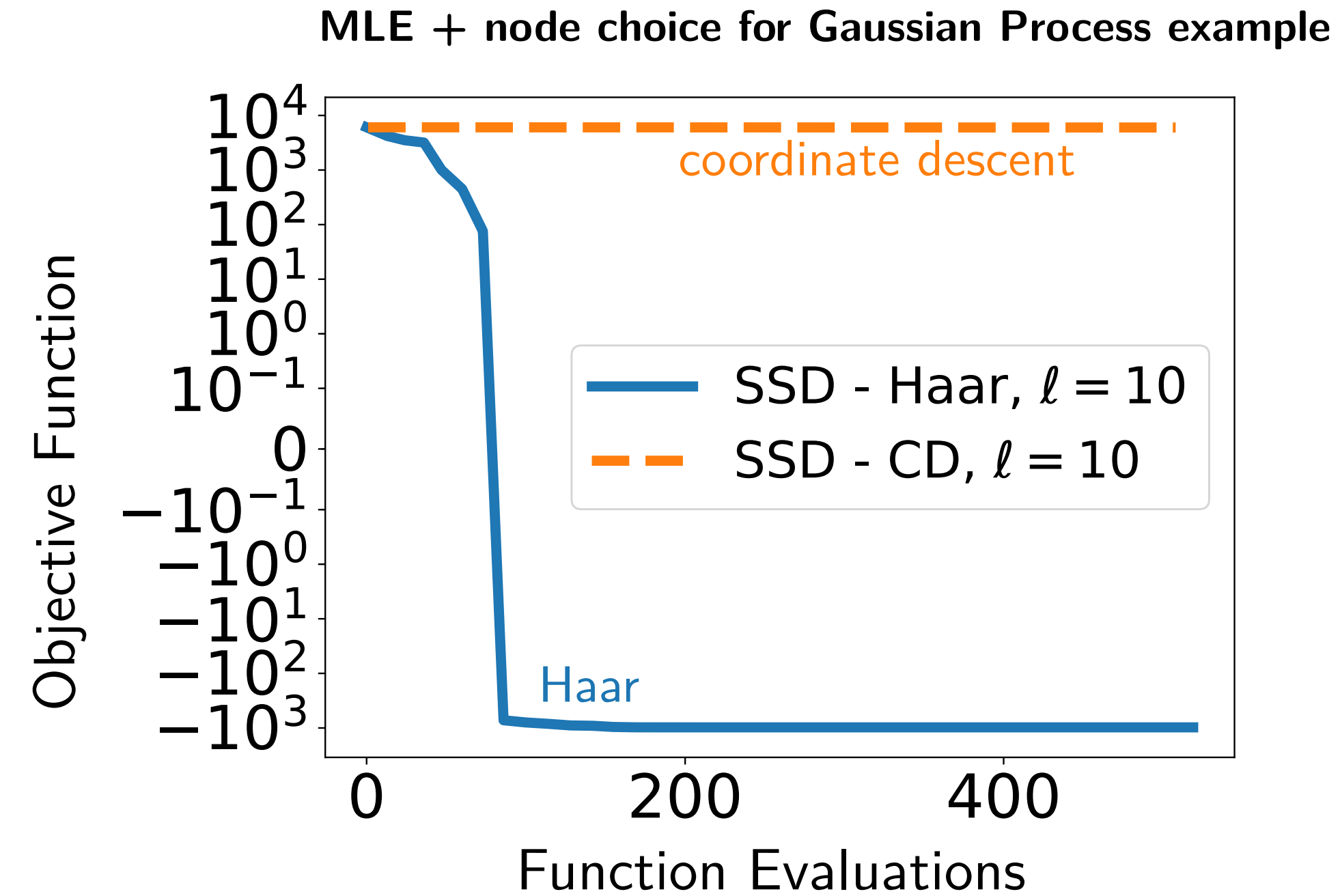
$\frac{d}{\ell} = 1$ is gradient descent

Generic SSD

$$Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E} \left(\frac{d}{\ell} Q Q^T \right) = I_{d \times d}$$

Numerical Results: better than expected

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**



Observation: sometimes SSD (with Haar) drastically outperforms randomized coordinate descent (CD)

Generic SSD

$$Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E} \left(\frac{d}{\ell} Q Q^T \right) = I_{d \times d}$$

Numerical Results: better than expected

Haar SSD drastically outperforms randomized coordinate descent (CD)

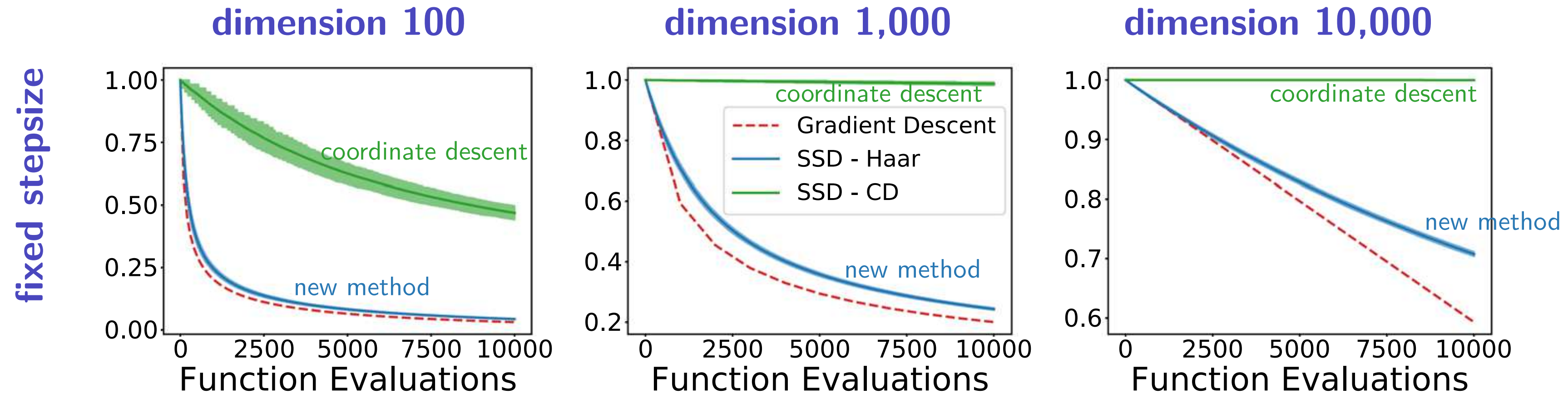


We can force it to happen by making a problem with low “intrinsic” dimension, e.g., Nesterov’s “worst function in the world”

$$f_{\lambda,r}(\mathbf{x}) = \lambda \left((x_1^2 + \sum_{i=1}^{r-1} (x_i - x_{i+1})^2 + x_r^2) / 2 - x_1 \right) / 4,$$

This has **intrinsic dimension** of r

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**



Numerical Results: better than expected

Haar SSD drastically outperforms randomized coordinate descent (CD)

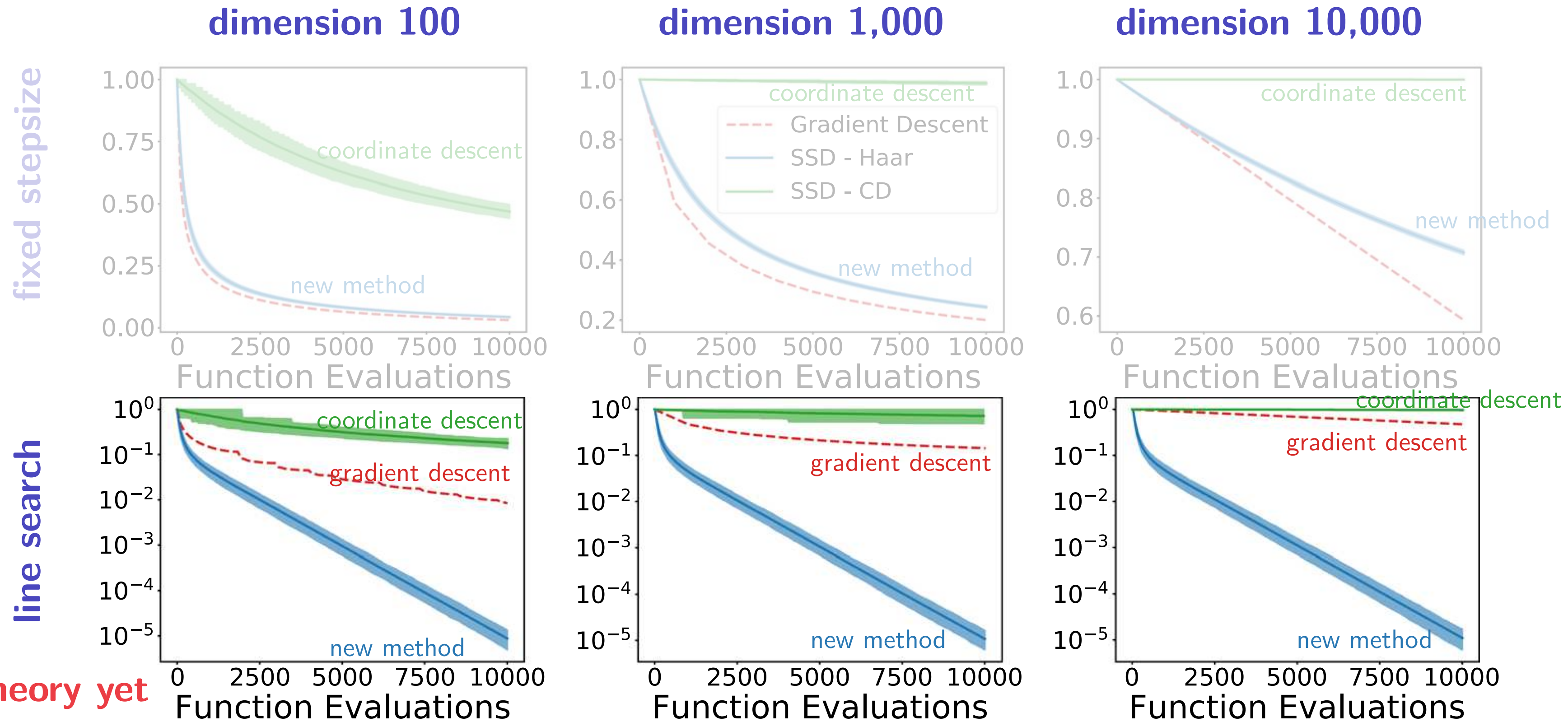


We can force it to happen by making a problem with low “intrinsic” dimension, e.g., Nesterov’s “worst function in the world”

$$f_{\lambda,r}(\mathbf{x}) = \lambda \left((x_1^2 + \sum_{i=1}^{r-1} (x_i - x_{i+1})^2 + x_r^2) / 2 - x_1 \right) / 4,$$

This has **intrinsic dimension** of r

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**



Theory: explain better-than-expected results

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Previous theorem didn't actually rely on properties of Haar distribution, just generic Q :

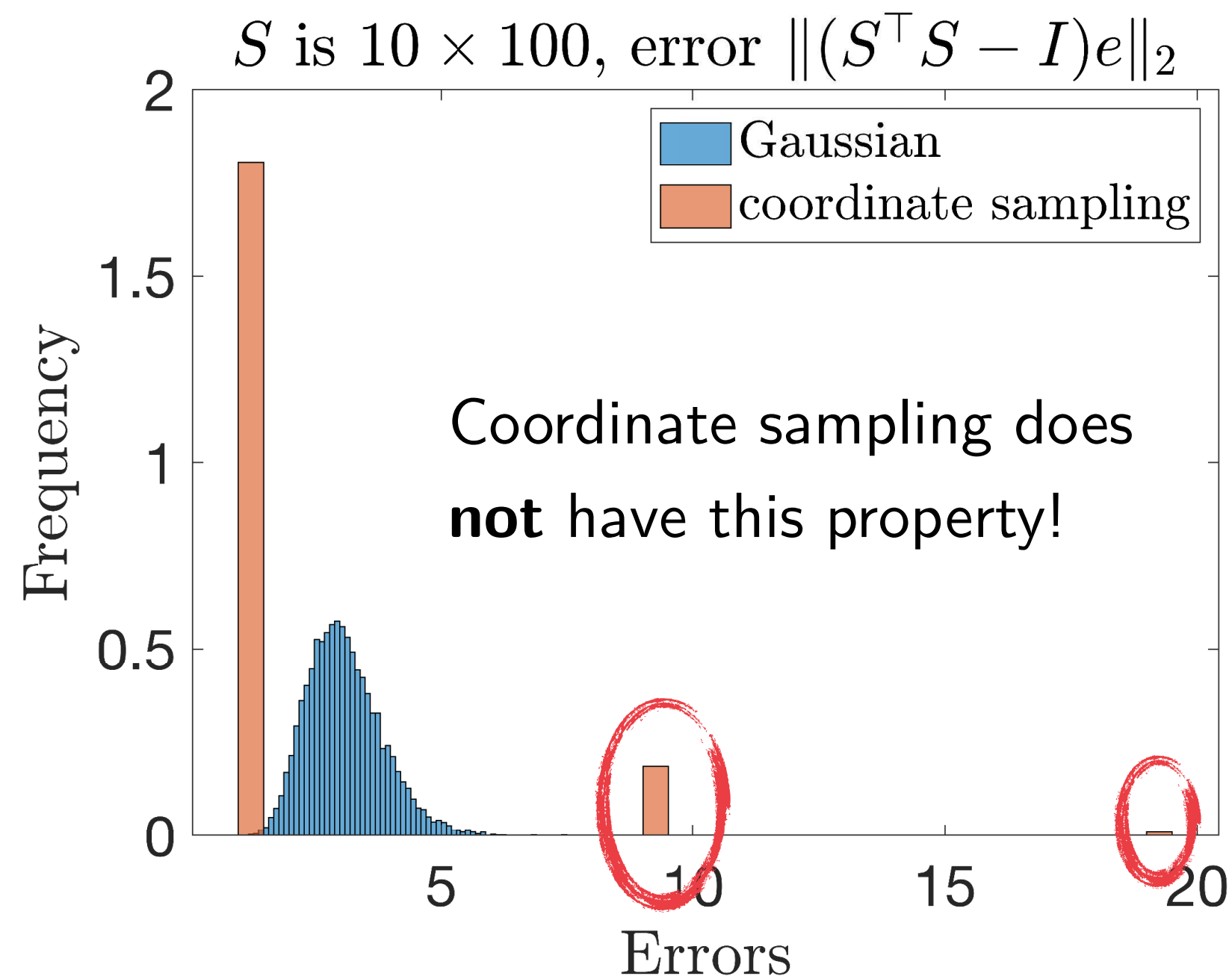
$$Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E} \left(\frac{d}{\ell} Q Q^T \right) = I_{d \times d}$$

Tighter analysis using concentration-of-measure:

Lemma 2 (Johnson-Lindenstrauss style embedding, from Kozak, Becker, Tenorio '19, Lemma 1).

$\forall \epsilon \in (0, 1)$, if $\ell \gtrsim \epsilon^{-2}$, $Q \sim \text{Haar}(d \times \ell)$, then $\forall 0 \neq g \in \mathbb{R}^d$,

$$1 - \epsilon \leq \frac{d}{\ell} \frac{\|Q^T g\|^2}{\|g\|^2} \leq 1 + \epsilon \quad w/ \text{prob. } \delta \geq 0.8$$



Recall...

d = ambient dimension
 ℓ = # directional derivs

Theory: explain better-than-expected results

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. Structured sketches
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. **Gradient-free optimization**

Previous theorem didn't actually rely on properties of Haar distribution, just

$$Q^T Q = I_{\ell \times \ell}, \quad \mathbb{E} \left(\frac{d}{\ell} Q Q^T \right) = I_{d \times d}$$

Tighter analysis using concentration-of-measure:

Lemma 2 (Johnson-Lindenstrauss style embedding, from Kozak, Becker, Tenorio '19, Lemma 1).
 $\forall \epsilon \in (0, 1)$, if $\ell \gtrsim \epsilon^{-2}$, $Q \sim \text{Haar}(d \times \ell)$, then $\forall 0 \neq g \in \mathbb{R}^d$,

$$1 - \epsilon \leq \frac{d \|Q^T g\|^2}{\ell \|g\|^2} \leq 1 + \epsilon \quad w/ \text{prob. } \delta \geq 0.8$$

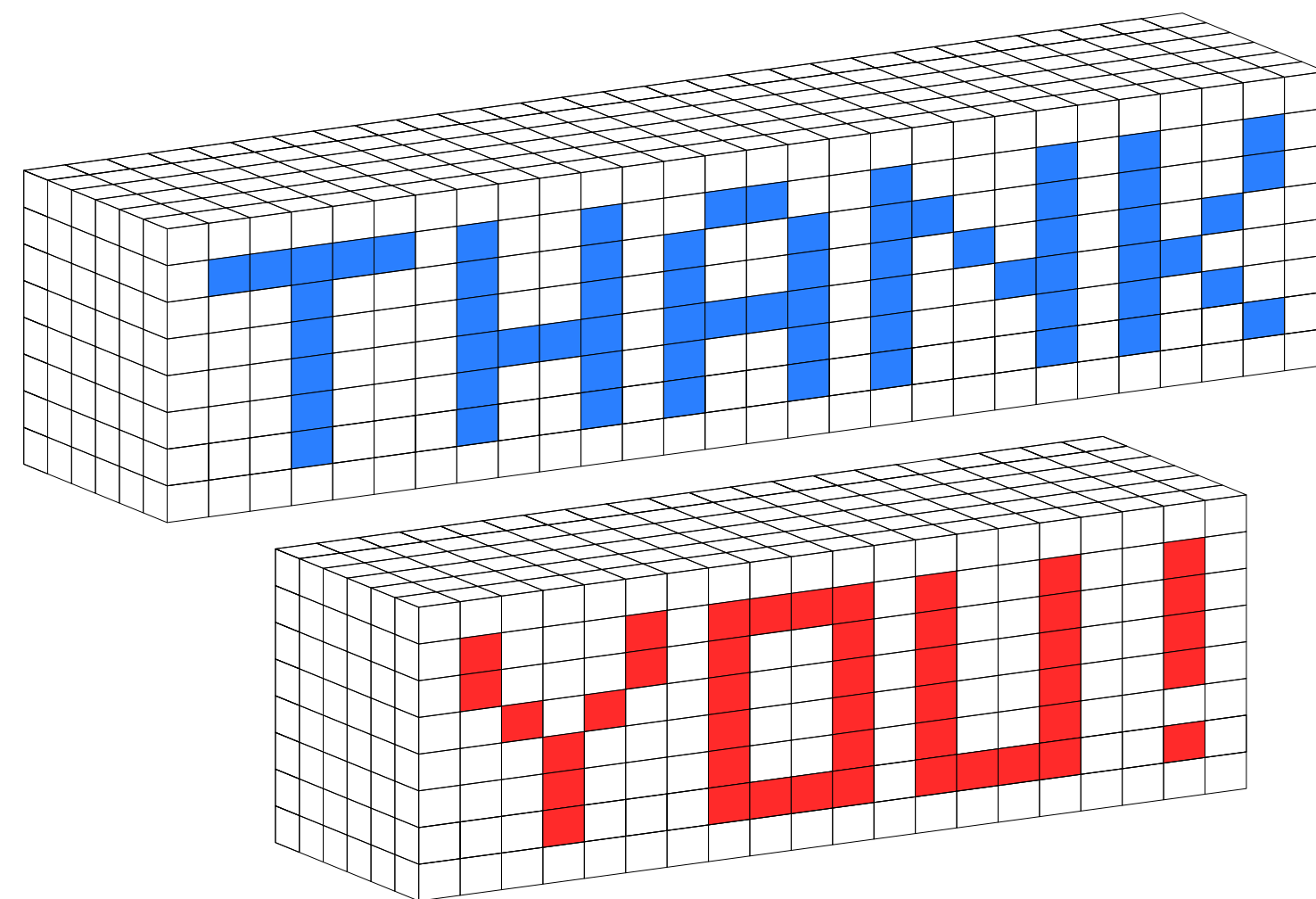
Theorem 3 (Kozak, Becker, Tenorio '19, Thm. 1). *If f is strongly convex and ∇f is Lipschitz continuous, then for an appropriate stepsize η_k , the sequence (x_k) generated by SSD (with $Q \sim \text{Haar}$), for $k > 100$, satisfies*

$$f(x_k) - f^* \leq (1 + (1 - \epsilon)\rho)^{k/2} (f(x_0) - f^*) \quad \text{with probability } \geq 0.998,$$

where $\rho < 1$ depends on ℓ , d and the Lipschitz and strong convexity parameters.

error in JL embedding

due to possibility of failure of JL




Papers and code available at <https://amath.colorado.edu/faculty/becker/>

Method 7: CountSketch

Introduced in Charikar et al. (2004), more analysis in, e.g., Clarkson and Woodruff (2017)

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

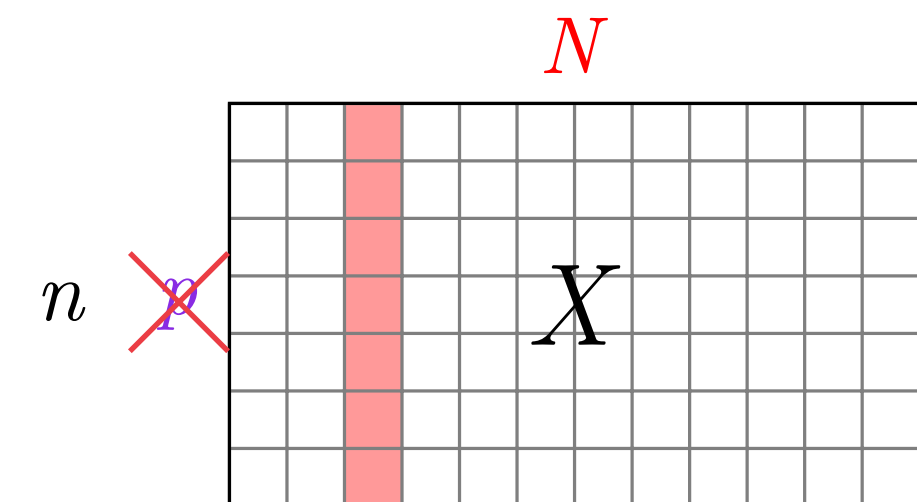
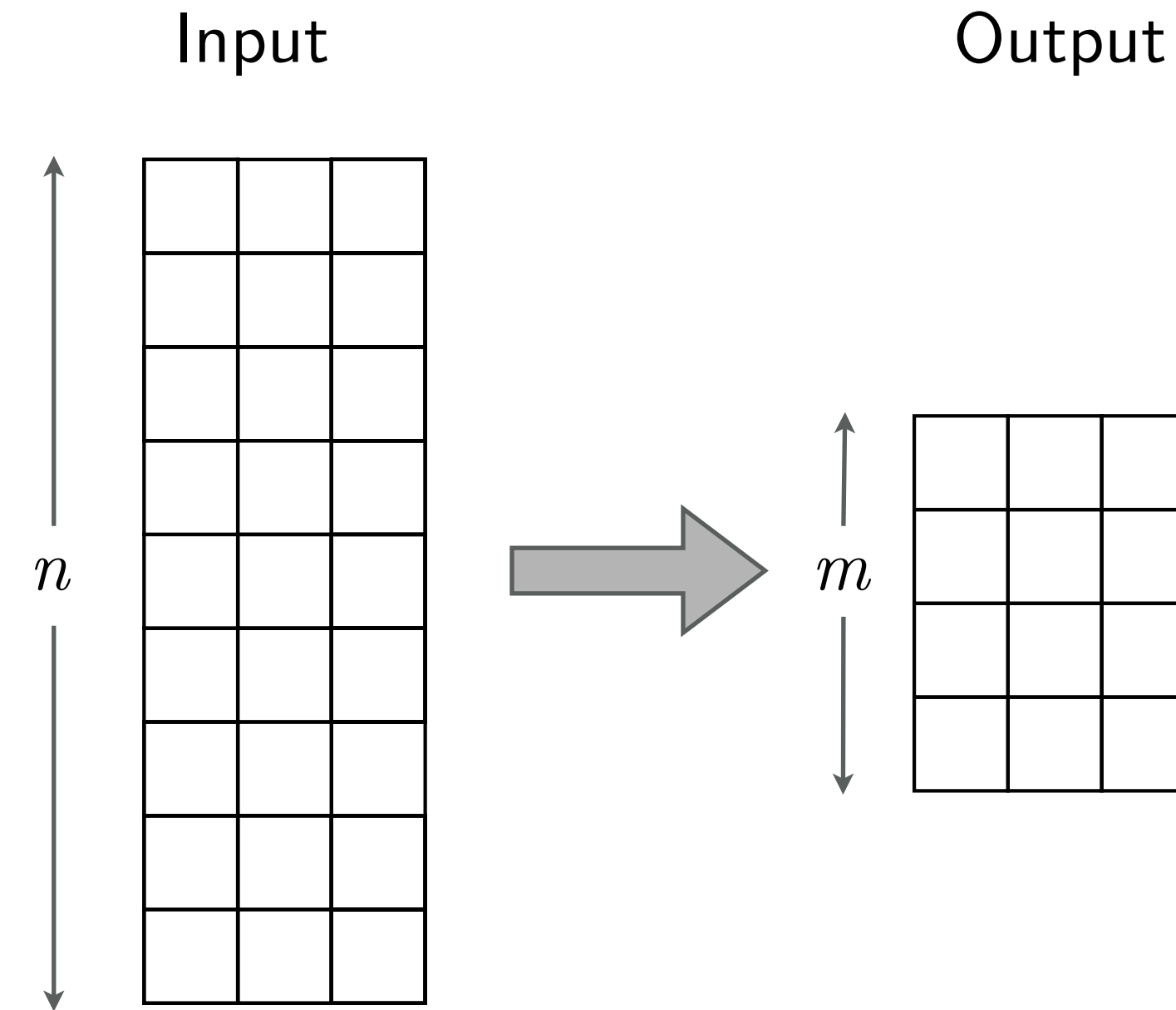
$\mathcal{S} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ linear operator

 Change in notation temporarily

$\Phi \rightarrow \mathcal{S}$

$p \rightarrow n$

$p_{\text{small}} \rightarrow m$



Method 7: CountSketch

- 1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
- 2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

$\mathcal{S} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ linear operator

Notation: $[n] = (0, 1, \dots, n - 1)$

(use 0-based indexing)

Step 1:

Multiply by random sign*

$s : [n] \rightarrow \{\pm 1\}$
(uniform)

+1	+1	+1
+1	+1	+1
-1	-1	-1
+1	+1	+1
-1	-1	-1
-1	-1	-1
-1	-1	-1
+1	+1	+1
-1	-1	-1

* technically doesn't have to be fully random, but must be 2-wise independent

$$s(i) \perp s(j) \quad \text{if } i \neq j$$

Method 7: CountSketch

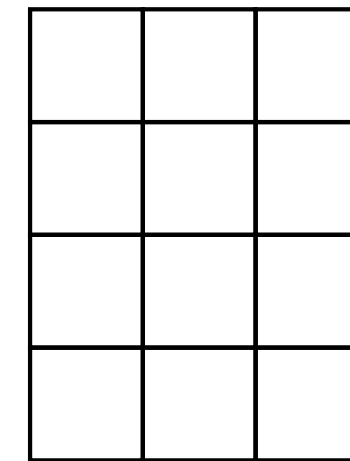
1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

$\mathcal{S} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ linear operator

Step 2:

Assign an output row to every input row (randomly or with a hash function)

$h : [n] \rightarrow [m]$
(uniform)



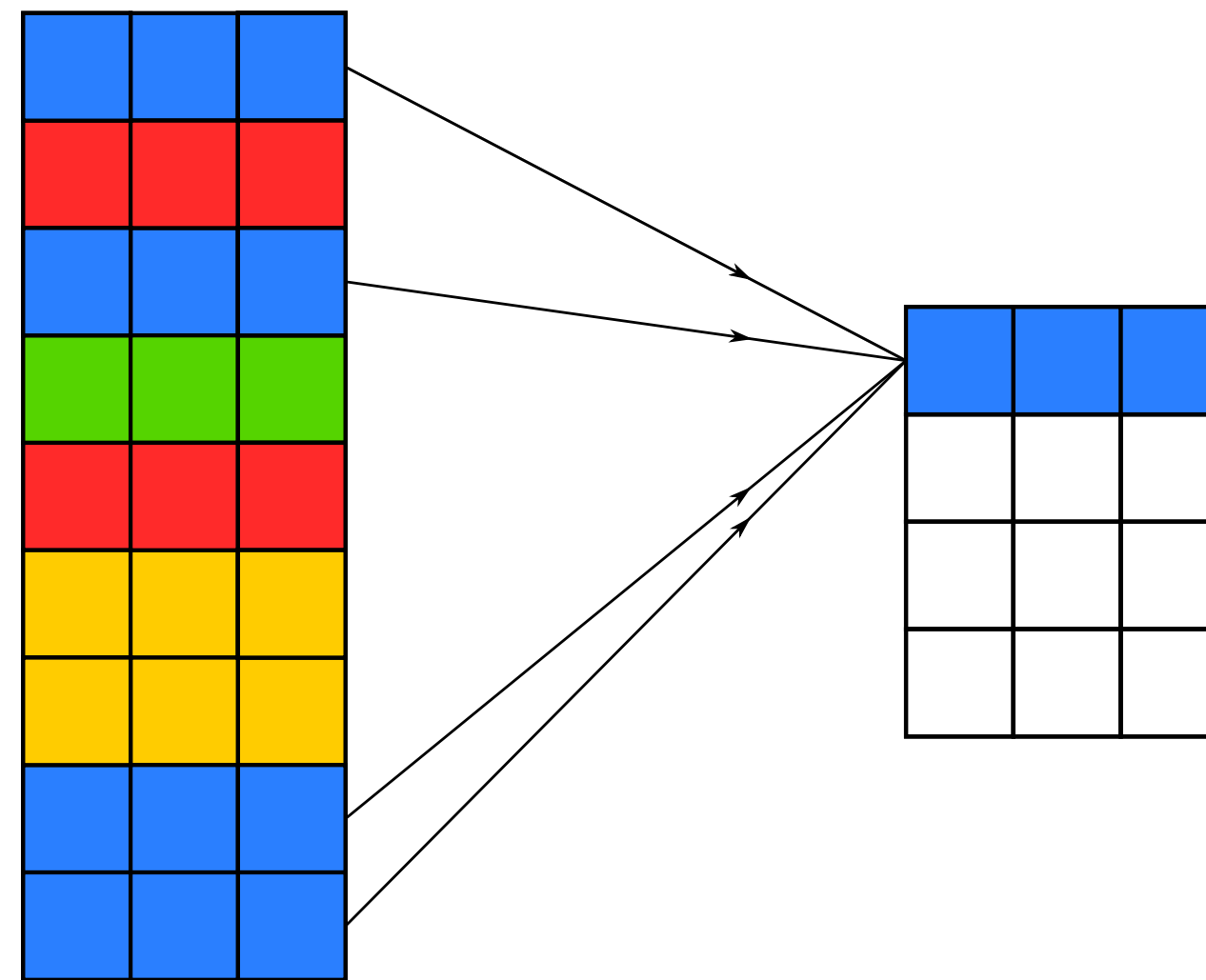
Again, don't need *all* of $h[i]$ to be independent, only pairwise independent
(but easy enough to make them all independent)

Method 7: CountSketch

$\mathcal{S} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ linear operator

Step 3:

Each output row is the sum of corresponding input rows



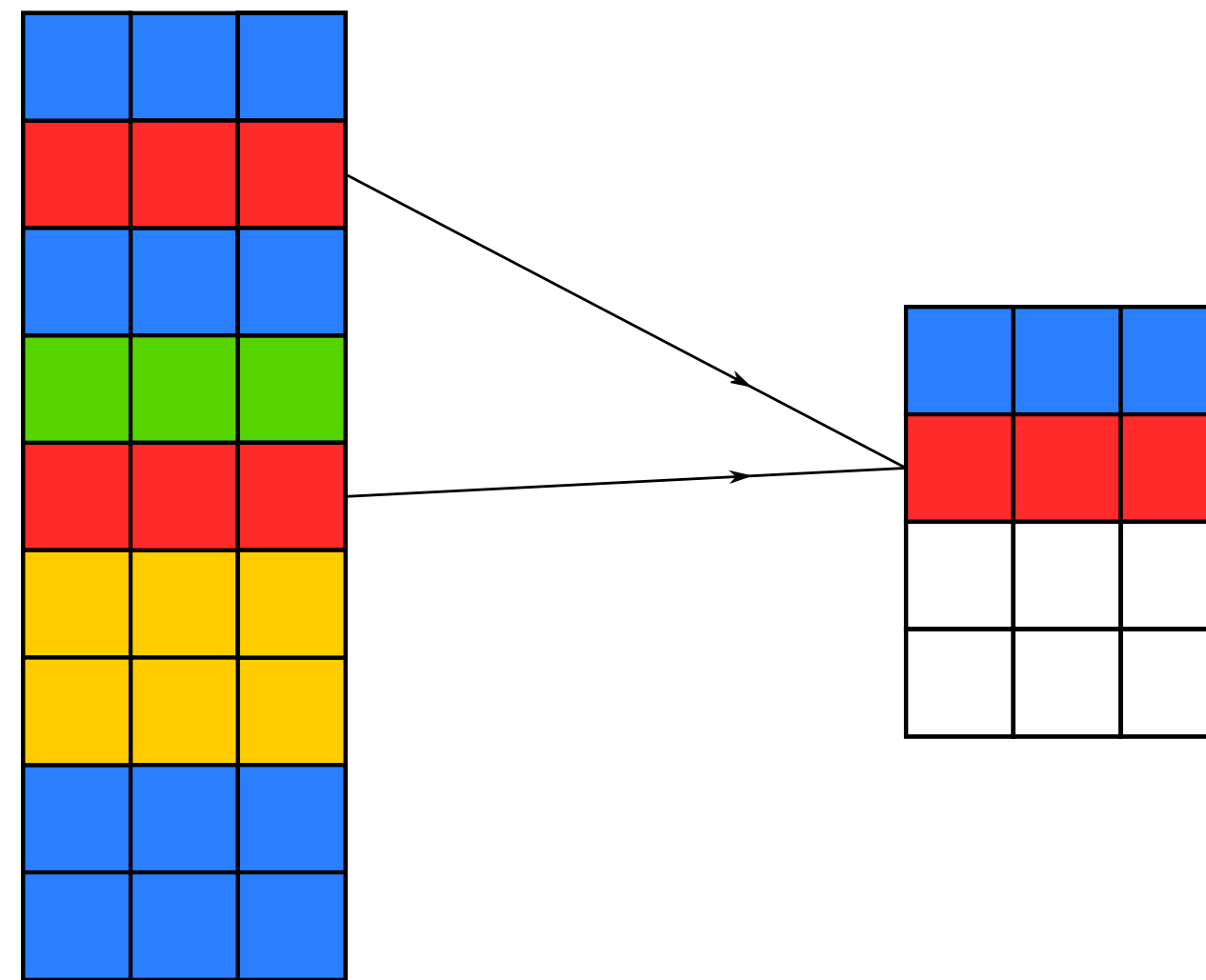
1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Method 7: CountSketch

$\mathcal{S} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ linear operator

Step 3:

Each output row is the sum of corresponding input rows



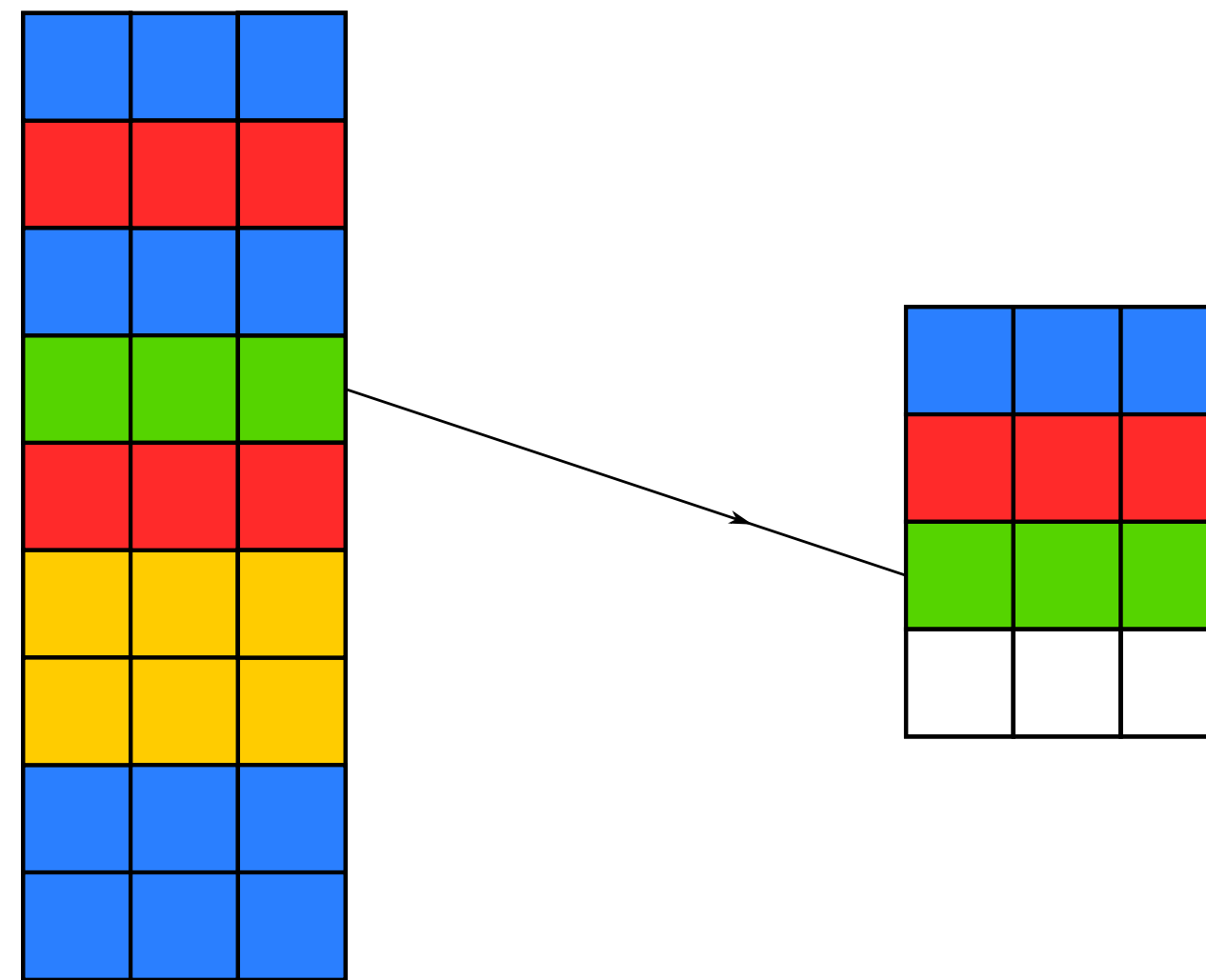
1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Method 7: CountSketch

$\mathcal{S} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ linear operator

Step 3:

Each output row is the sum of corresponding input rows



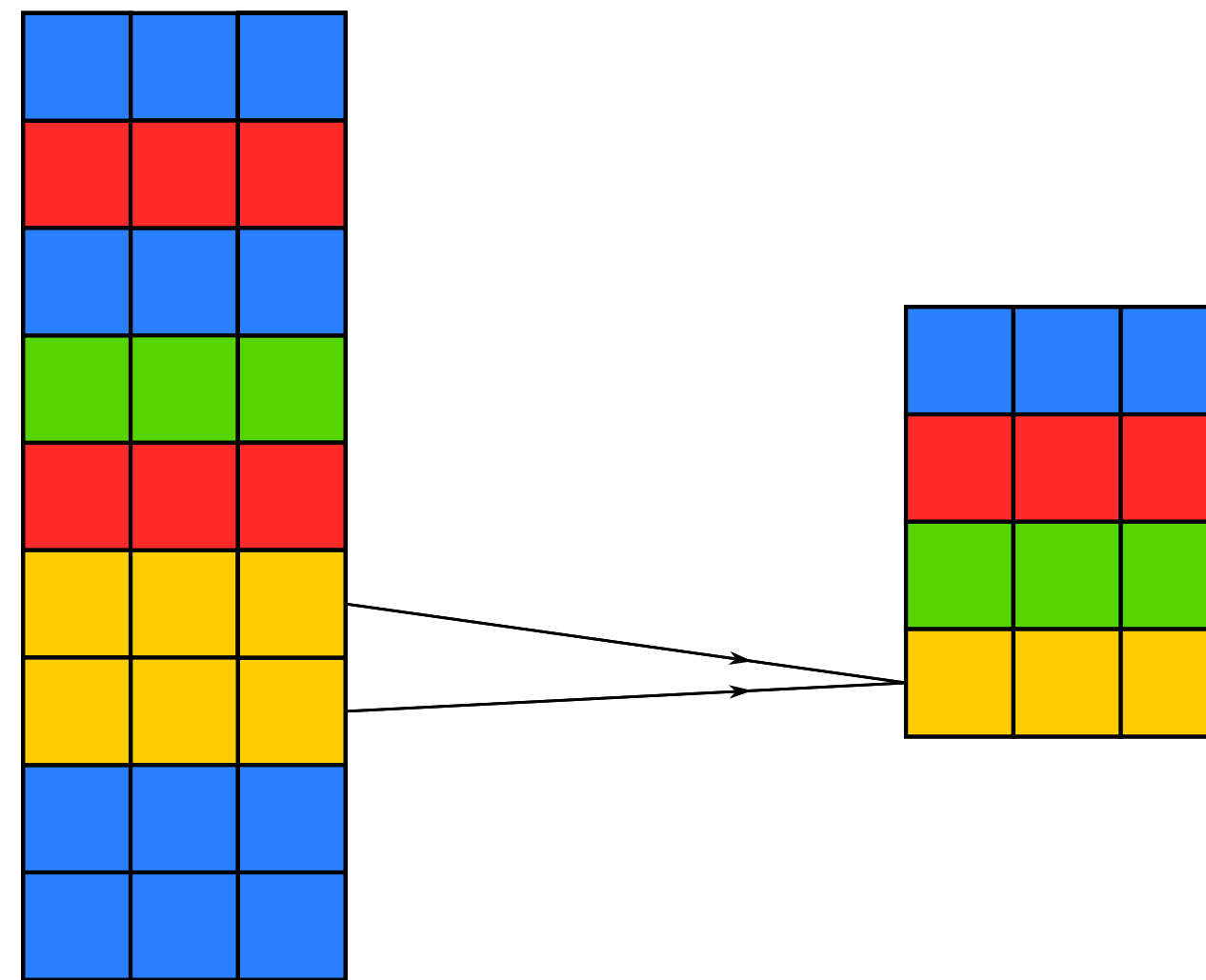
1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Method 7: CountSketch

$\mathcal{S} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ linear operator

Step 3:

Each output row is the sum of corresponding input rows



1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Method 7: CountSketch

$\mathcal{S} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ linear operator

Complexity analysis: every input element is touched once
... so linear complexity
(and can exploit sparsity)

1. Randomized “sketches”
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Method 7: CountSketch

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

$\mathcal{S} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ linear operator

A formula for the output:

$$\mathbf{u} = \mathcal{S}(\mathbf{v}), \quad u_i = \sum_{j|h(j)=i} s(j)v_j$$
$$s : [n] \rightarrow \{\pm 1\}$$
$$h : [n] \rightarrow [m]$$

... and note that the following polynomial has the output as its coefficients:

$$p(\mathbf{x}) \stackrel{\text{def}}{=} \sum_{j \in [n]} s(j) \cdot v_j \cdot \mathbf{x}^{h(j)} = \sum_{i \in [m]} \mathbf{x}^i \left(\sum_{j|h(j)=i} s(j)v_j \right) = \sum_{i \in [m]} u_i \cdot \mathbf{x}^i$$

Method 8: TensorSketch

Introduced in Pagh (2013), more analysis
in, e.g., Diao, Zong, Sun, Woodruff (2018)

TensorSketch is just CountSketch when the input can be written as a tensor product
(for a special choice of the hash and sign functions)



Change in notation temporarily

$\Phi \rightarrow \mathcal{S}$
 $p \rightarrow n$
 $p_{\text{small}} \rightarrow m$

$$\mathcal{T} : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \quad \text{where size is} \quad n = n^{(1)} \cdot n^{(2)}$$

Not (yet!) related to tensors

TensorSketch

Introduced in Pagh (2013), more analysis
in, e.g., Diao, Zong, Sun, Woodruff (2018)

TensorSketch is just CountSketch when the input can be written as a tensor product
(for a special choice of the hash and sign functions)

$$\mathcal{T} : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \quad \text{where size is} \quad n = n^{(1)} \cdot n^{(2)}$$

Kronecker/tensor product

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \mathbf{a} \otimes \mathbf{b} = \begin{bmatrix} a_1 \mathbf{b} \\ a_2 \mathbf{b} \\ a_3 \mathbf{b} \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_1 b_2 \\ a_1 b_3 \\ a_2 b_1 \\ a_2 b_2 \\ a_2 b_3 \\ a_3 b_1 \\ a_3 b_2 \\ a_3 b_3 \end{bmatrix}$$

TensorSketch is just CountSketch when the input can be written as a tensor product
(for a special choice of the hash and sign functions)

$$\mathcal{T} : \mathbb{R}^n \rightarrow \mathbb{R}^m$$

$$\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)} \quad \text{where size is} \quad n = n^{(1)} \cdot n^{(2)}$$

Kronecker/tensor product

$$\mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \mathbf{a} \otimes \mathbf{b} = \begin{bmatrix} a_1 \mathbf{b} \\ a_2 \mathbf{b} \\ a_3 \mathbf{b} \end{bmatrix} = \begin{bmatrix} a_1 b_1 \\ a_1 b_2 \\ a_1 b_3 \\ a_2 b_1 \\ a_2 b_2 \\ a_2 b_3 \\ a_3 b_1 \\ a_3 b_2 \\ a_3 b_3 \end{bmatrix}$$

another equivalent definition:

$$\mathbf{a} \otimes \mathbf{b} = \text{vec}_{\text{col}}(\mathbf{b}\mathbf{a}^T)$$

$$\mathbf{b}\mathbf{a}^T = \begin{bmatrix} b_1 a_1 & b_1 a_2 & b_1 a_3 \\ b_2 a_1 & b_2 a_2 & b_2 a_3 \\ b_3 a_1 & b_3 a_2 & b_3 a_3 \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_2 b_1 & a_3 b_1 \\ a_1 b_2 & a_2 b_2 & a_3 b_2 \\ a_1 b_3 & a_2 b_3 & a_3 b_3 \end{bmatrix}$$

$$\begin{aligned} \text{vec} : [n^{(1)}] \times [n^{(2)}] &\rightarrow [n^{(1)} \cdot n^{(2)}] \\ \text{vec}^{-1} = \text{mat} : [n^{(1)} \cdot n^{(2)}] &\rightarrow [n^{(1)}] \times [n^{(2)}] \end{aligned}$$

more generally,

$$(\mathbf{A} \otimes \mathbf{B}) \text{vec}_{\text{col}}(\mathbf{X}) = \text{vec}_{\text{col}}(\mathbf{B}\mathbf{X}\mathbf{A}^T)$$

TensorSketch

Pick hash functions in a decomposable way:

Input has structure $\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)}$ with sizes $n = n^{(1)} \cdot n^{(2)}$

$$\begin{array}{l} \text{build small} \quad h^{(1)} : [n^{(1)}] \rightarrow [m] \\ \text{hash functions: } h^{(2)} : [n^{(2)}] \rightarrow [m] \end{array} \quad \longrightarrow \quad \begin{array}{l} h : [n] \rightarrow [m] \\ h(j_1, j_2) = h^{(1)}(j_1) + h^{(2)}(j_2) \pmod m \end{array}$$

(we're being loose about how we write the input, since we can use the mat/vec bijection as needed)


Fact: as long as $h^{(i)}$ is fully independent (or at least 3-wise independent), then h is 2-wise independent

TensorSketch

Pick hash functions in a decomposable way:

Input has structure $\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)}$ with sizes $n = n^{(1)} \cdot n^{(2)}$

build small hash functions: $h^{(1)} : [n^{(1)}] \rightarrow [m]$
 $h^{(2)} : [n^{(2)}] \rightarrow [m]$




$h : [n] \rightarrow [m]$
 $h(j_1, j_2) = h^{(1)}(j_1) + h^{(2)}(j_2) \pmod m$

Fact: h is 2-wise independent

(we're being loose about how we write the input, since we can use the mat/vec bijection as needed)

same trick for sign functions: $s^{(1)} : [n^{(1)}] \rightarrow \{\pm 1\}$
 $s^{(2)} : [n^{(2)}] \rightarrow \{\pm 1\}$



$s : [n] \rightarrow \{\pm 1\}$
 $s(j_1, j_2) = s^{(1)}(j_1) \cdot s^{(2)}(j_2)$

still pairwise independent, as needed for theory!

TensorSketch

what's the point? Huge computational speedup.

$$h^{(1)} : [n^{(1)}] \rightarrow [m]$$

$$h^{(2)} : [n^{(2)}] \rightarrow [m]$$

$$p^{(1)}(\mathbf{x}) \stackrel{\text{def}}{=} \sum_{j \in [n^{(1)}]} s^{(1)}(j) \cdot v_j^{(1)} \cdot \mathbf{x}^{h^{(1)}(j)} = \sum_{i \in [m]} u_i^{(1)} \cdot \mathbf{x}^i$$

$$p^{(2)}(\mathbf{x}) \stackrel{\text{def}}{=} \sum_{j \in [n^{(2)}]} s^{(2)}(j) \cdot v_j^{(2)} \cdot \mathbf{x}^{h^{(2)}(j)} = \sum_{i \in [m]} u_i^{(2)} \cdot \mathbf{x}^i$$

Let's compute this: $p^{(1)}(\mathbf{x}) \cdot p^{(2)}(\mathbf{x}) \pmod{\mathbf{x}^m - 1}$

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

TensorSketch

what's the point? Huge computational speedup.

$$p^{(1)}(\mathbf{x}) \cdot p^{(2)}(\mathbf{x}) = \left(\sum_{j_1 \in [n^{(1)}]} s^{(1)}(j_1) \cdot v_{j_1}^{(1)} \cdot \mathbf{x}^{h^{(1)}(j_1)} \right) \cdot \left(\sum_{j_2 \in [n^{(2)}]} s^{(2)}(j_2) \cdot v_{j_2}^{(2)} \cdot \mathbf{x}^{h^{(2)}(j_2)} \right)$$

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

TensorSketch

what's the point? **Huge computational speedup.**

$$\begin{aligned} p^{(1)}(\mathbf{x}) \cdot p^{(2)}(\mathbf{x}) &= \left(\sum_{j_1 \in [n^{(1)}]} s^{(1)}(j_1) \cdot v_{j_1}^{(1)} \cdot \mathbf{x}^{h^{(1)}(j_1)} \right) \cdot \left(\sum_{j_2 \in [n^{(2)}]} s^{(2)}(j_2) \cdot v_{j_2}^{(2)} \cdot \mathbf{x}^{h^{(2)}(j_2)} \right) \\ &= \sum_{j_1 \in [n^{(1)}], j_2 \in [n^{(2)}]} s^{(1)}(j_1) \cdot s^{(2)}(j_2) \cdot v_{j_1}^{(1)} \cdot v_{j_2}^{(2)} \cdot \mathbf{x}^{h^{(1)}(j_1)} \cdot \mathbf{x}^{h^{(2)}(j_2)} \end{aligned}$$

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

TensorSketch

what's the point? Huge computational speedup.

$$\begin{aligned} p^{(1)}(\mathbf{x}) \cdot p^{(2)}(\mathbf{x}) &= \left(\sum_{j_1 \in [n^{(1)}]} s^{(1)}(j_1) \cdot v_{j_1}^{(1)} \cdot \mathbf{x}^{h^{(1)}(j_1)} \right) \cdot \left(\sum_{j_2 \in [n^{(2)}]} s^{(2)}(j_2) \cdot v_{j_2}^{(2)} \cdot \mathbf{x}^{h^{(2)}(j_2)} \right) \\ &= \sum_{j_1 \in [n^{(1)}], j_2 \in [n^{(2)}]} s^{(1)}(j_1) \cdot s^{(2)}(j_2) \cdot v_{j_1}^{(1)} \cdot v_{j_2}^{(2)} \cdot \mathbf{x}^{h^{(1)}(j_1)} \cdot \mathbf{x}^{h^{(2)}(j_2)} \\ &= \sum_{j_1 \in [n^{(1)}], j_2 \in [n^{(2)}]} \mathcal{S}(j_1, j_2) \cdot v_{j_1 j_2} \cdot \mathbf{x}^{h^{(1)}(j_1) h^{(2)}(j_2)} \end{aligned}$$

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

TensorSketch

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

what's the point? Huge computational speedup.

$$\begin{aligned}
 p^{(1)}(\mathbf{x}) \cdot p^{(2)}(\mathbf{x}) &= \left(\sum_{j_1 \in [n^{(1)}]} s^{(1)}(j_1) \cdot v_{j_1}^{(1)} \cdot \mathbf{x}^{h^{(1)}(j_1)} \right) \cdot \left(\sum_{j_2 \in [n^{(2)}]} s^{(2)}(j_2) \cdot v_{j_2}^{(2)} \cdot \mathbf{x}^{h^{(2)}(j_2)} \right) \\
 &= \sum_{j_1 \in [n^{(1)}], j_2 \in [n^{(2)}]} s^{(1)}(j_1) \cdot s^{(2)}(j_2) \cdot v_{j_1}^{(1)} \cdot v_{j_2}^{(2)} \cdot \mathbf{x}^{h^{(1)}(j_1)} \cdot \mathbf{x}^{h^{(2)}(j_2)} \\
 &= \sum_{j_1 \in [n^{(1)}], j_2 \in [n^{(2)}]} \mathcal{S}(j_1, j_2) \cdot v_{j_1 j_2} \cdot \mathbf{x}^{h^{(1)}(j_1) h^{(2)}(j_2)} \\
 &\equiv \sum_{j_1 \in [n^{(1)}], j_2 \in [n^{(2)}]} \mathcal{S}(j_1, j_2) \cdot v_{j_1 j_2} \cdot \mathbf{x}^{h(j_1, j_2)} \pmod{\mathbf{x}^m - 1}
 \end{aligned}$$

$$\mathbf{x}^{kl} = \mathbf{x}^{dm + (kl \pmod m)} = \mathbf{x}^{dm + (kl \pmod m)} = (\mathbf{x}^m)^d \mathbf{x}^{kl \pmod m} \equiv \mathbf{x}^{kl \pmod m} \pmod{\mathbf{x}^m - 1}$$

Recall $\mathbf{x}^m \equiv 1 \pmod{\mathbf{x}^m - 1}$

TensorSketch

what's the point? Huge computational speedup.

$$\begin{aligned} p^{(1)}(\mathbf{x}) \cdot p^{(2)}(\mathbf{x}) &= \left(\sum_{j_1 \in [n^{(1)}]} s^{(1)}(j_1) \cdot v_{j_1}^{(1)} \cdot \mathbf{x}^{h^{(1)}(j_1)} \right) \cdot \left(\sum_{j_2 \in [n^{(2)}]} s^{(2)}(j_2) \cdot v_{j_2}^{(2)} \cdot \mathbf{x}^{h^{(2)}(j_2)} \right) \\ &= \sum_{j_1 \in [n^{(1)}], j_2 \in [n^{(2)}]} s^{(1)}(j_1) \cdot s^{(2)}(j_2) \cdot v_{j_1}^{(1)} \cdot v_{j_2}^{(2)} \cdot \mathbf{x}^{h^{(1)}(j_1)} \cdot \mathbf{x}^{h^{(2)}(j_2)} \\ &= \sum_{j_1 \in [n^{(1)}], j_2 \in [n^{(2)}]} \mathcal{S}(j_1, j_2) \cdot v_{j_1 j_2} \cdot \mathbf{x}^{h^{(1)}(j_1) h^{(2)}(j_2)} \\ &\equiv \sum_{j_1 \in [n^{(1)}], j_2 \in [n^{(2)}]} \mathcal{S}(j_1, j_2) \cdot v_{j_1 j_2} \cdot \mathbf{x}^{h(j_1, j_2)} \pmod{\mathbf{x}^m - 1} \\ &= p(\mathbf{x}) \end{aligned}$$

coefficients are the output
of the CountSketch!

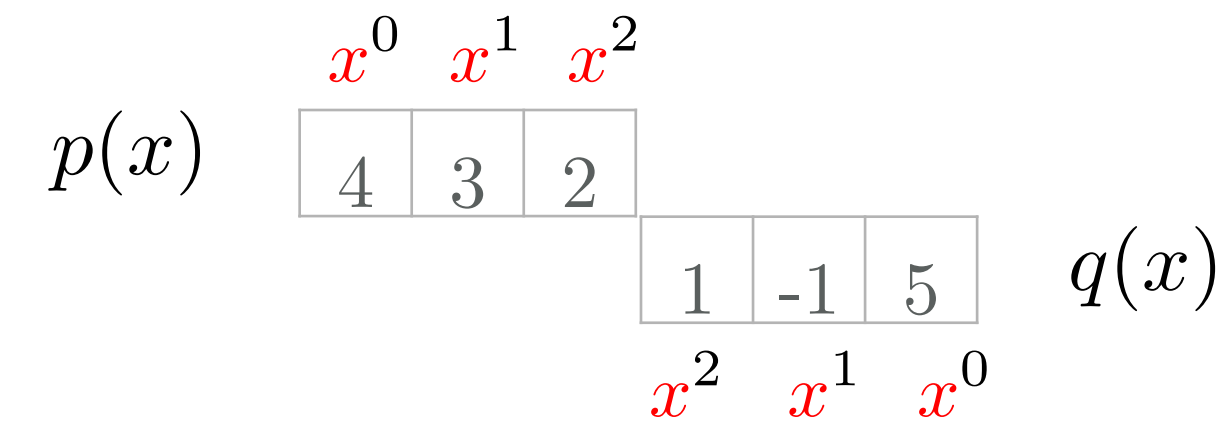
1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

TensorSketch

How to multiply polynomials?

$$(2x^2 + 3x + 4) \cdot (x^2 - x + 5)$$

$p(x)$ $q(x)$

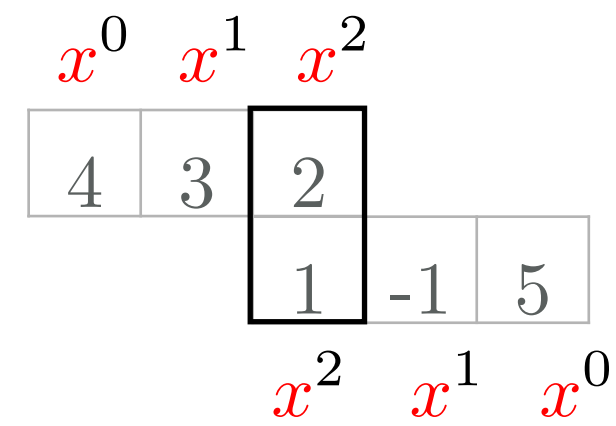


1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

TensorSketch

How to multiply polynomials?

$$(2x^2 + 3x + 4) \cdot (x^2 - x + 5)$$



$$= 2x^4$$

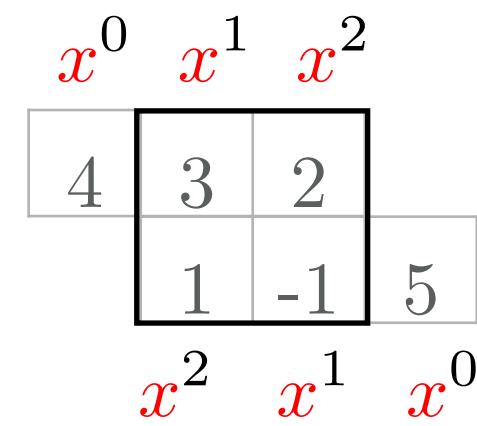
$$2 \cdot 1$$

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

TensorSketch

How to multiply polynomials?

$$(2x^2 + 3x + 4) \cdot (x^2 - x + 5)$$



$$= 2x^4 + x^3$$

$$3 \cdot 1 + 2 \cdot (-1) = 1$$

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

How to multiply polynomials?

$$(2x^2 + 3x + 4) \cdot (x^2 - x + 5)$$

x^0	x^1	x^2
4	3	2
1	-1	5
x^2	x^1	x^0

$$= 2x^4 + x^3 + 11x^2$$

$$4 \cdot 1 + 3 \cdot (-1) + 2 \cdot 5 = 11$$

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

TensorSketch

How to multiply polynomials?

$$(2x^2 + 3x + 4) \cdot (x^2 - x + 5)$$

	x^0	x^1	x^2	
	4	3	2	
1	-1	5		
x^2	x^1	x^0		

$$= 2x^4 + x^3 + 11x^2 + 11x$$

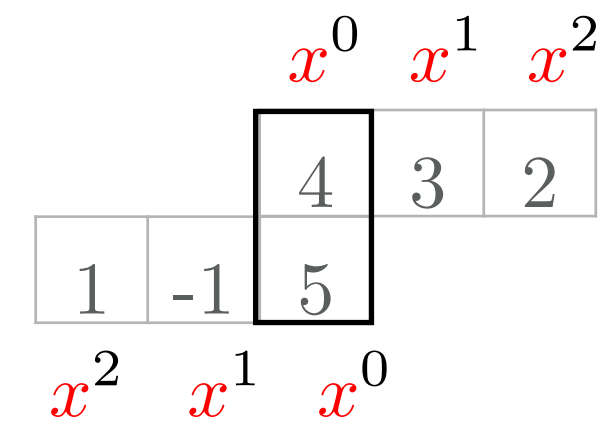
$$4 \cdot (-1) + 3 \cdot 5 = 11$$

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

TensorSketch

How to multiply polynomials?

$$(2x^2 + 3x + 4) \cdot (x^2 - x + 5)$$



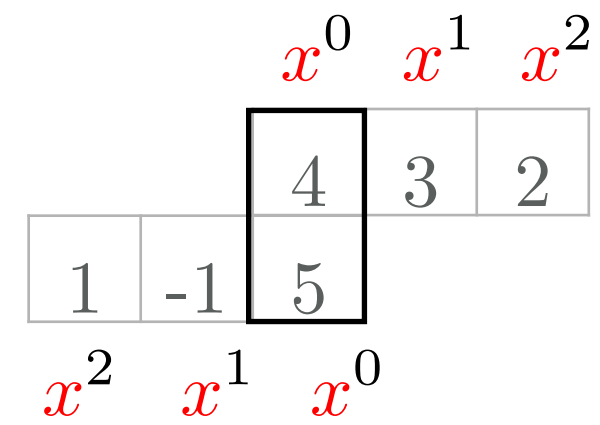
$$= 2x^4 + x^3 + 11x^2 + 11x + 20$$

$$4 \cdot 5 = 20$$

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

How to multiply polynomials?

$$(2x^2 + 3x + 4) \cdot (x^2 - x + 5)$$



$$= 2x^4 + x^3 + 11x^2 + 11x + 20$$

$$\equiv 11x^2 + 13x + 21 \pmod{x^3 - 1}$$

... this is circular convolution!

TensorSketch: complexity analysis

Convolution Theorem (for circular convolutions $*$)

pointwise multiplication

$$\mathcal{F}(g * h) = \mathcal{F}(g)\mathcal{F}(h) \quad \text{i.e.} \quad g * h = \mathcal{F}^{-1}(\mathcal{F}(g)\mathcal{F}(h))$$

Discrete Fourier Transform (implemented via FFT)

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

TensorSketch: complexity analysis

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

Convolution Theorem (for circular convolutions $*$)

$$\mathcal{F}(g * h) = \mathcal{F}(g)\mathcal{F}(h) \quad \text{i.e.} \quad g * h = \mathcal{F}^{-1}\left(\mathcal{F}(g)\mathcal{F}(h)\right)$$

Complexity:

$$\mathcal{T} : \mathbb{R}^n \rightarrow \mathbb{R}^m$$
$$n = n^{(1)} \cdot n^{(2)}$$

CountSketch applied as TensorSketch CountSketch applied naively generic dense matrix multiply

$$\mathcal{O}(n^{(1)} + n^{(2)} + m \log m) \quad \text{vs} \quad \mathcal{O}(n^{(1)}n^{(2)}) \quad \text{vs} \quad \mathcal{O}(n^{(1)}n^{(2)}m)$$

small CountSketches polynomial multiplication via FFTs

Savings grow as we have more tensor products

$$\text{If } n = n^{(1)}n^{(2)} \dots n^{(q)}$$

$$\mathcal{O}(n^{(1)} + n^{(2)} + \dots + n^{(q)} + m \log m) \quad \text{vs} \quad \mathcal{O}(n^{(1)}n^{(2)} \dots n^{(q)})$$

TensorSketch: applying to matrices

1. Randomized "sketches"
 - a. Warmup: PCA
 - b. Classical sketches
 - c. **Structured sketches**
2. Applications
 - a. Warmup: linear algebra
 - b. K-means clustering
 - c. Tensor factorizations
 - d. Gradient-free optimization

From vector case $\mathbf{v} = \mathbf{v}^{(1)} \otimes \mathbf{v}^{(2)}$ to matrix case $\mathbf{A} = \mathbf{A}^{(1)} \otimes \mathbf{A}^{(2)} \otimes \dots \otimes \mathbf{A}^{(N)}$

Recall the **Kronecker** product:

$$\begin{array}{c} \mathbf{A} \otimes \mathbf{B} \\ \begin{array}{cc} \nearrow & \nwarrow \\ I \times J & K \times L \end{array} \end{array} = \begin{bmatrix} a_{11}\mathbf{B} & a_{12}\mathbf{B} & \cdots & a_{1J}\mathbf{B} \\ a_{21}\mathbf{B} & a_{22}\mathbf{B} & \cdots & a_{2J}\mathbf{B} \\ \vdots & \vdots & \ddots & \vdots \\ a_{I1}\mathbf{B} & a_{I2}\mathbf{B} & \cdots & a_{IJ}\mathbf{B} \end{bmatrix}$$

$$= [\mathbf{a}_1 \otimes \mathbf{b}_1 \quad \mathbf{a}_1 \otimes \mathbf{b}_2 \quad \mathbf{a}_1 \otimes \mathbf{b}_3 \quad \cdots \quad \mathbf{a}_J \otimes \mathbf{b}_{L-1} \quad \mathbf{a}_J \otimes \mathbf{b}_L]$$

Khatri-Rao product

$$\mathbf{A} \odot \mathbf{B} = [\mathbf{a}_1 \otimes \mathbf{b}_1 \quad \mathbf{a}_2 \otimes \mathbf{b}_2 \quad \cdots \quad \mathbf{a}_K \otimes \mathbf{b}_K]$$

pointwise multiplication (aka Hadamard product)

Observe

$$\mathbf{a} \circ \mathbf{b} = (\mathbf{a}^\top \odot \mathbf{b}^\top)^\top$$

and with some work,

$$\mathbf{TA} = \text{FFT}^{-1} \left(\left(\bigodot_{n=1}^N \left(\text{FFT} \left(\mathbf{S}^{(n)} \mathbf{A}^{(n)} \right) \right)^\top \right)^\top \right)$$