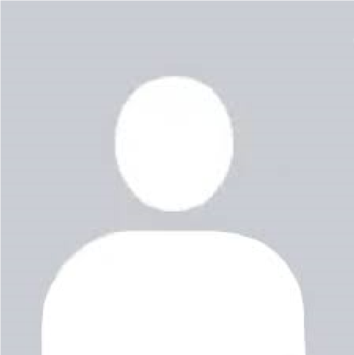
Fast and Space-Optimal Streaming Algorithms for Euclidean Clustering

Vincent Cohen-Addad



Liudeng Wang

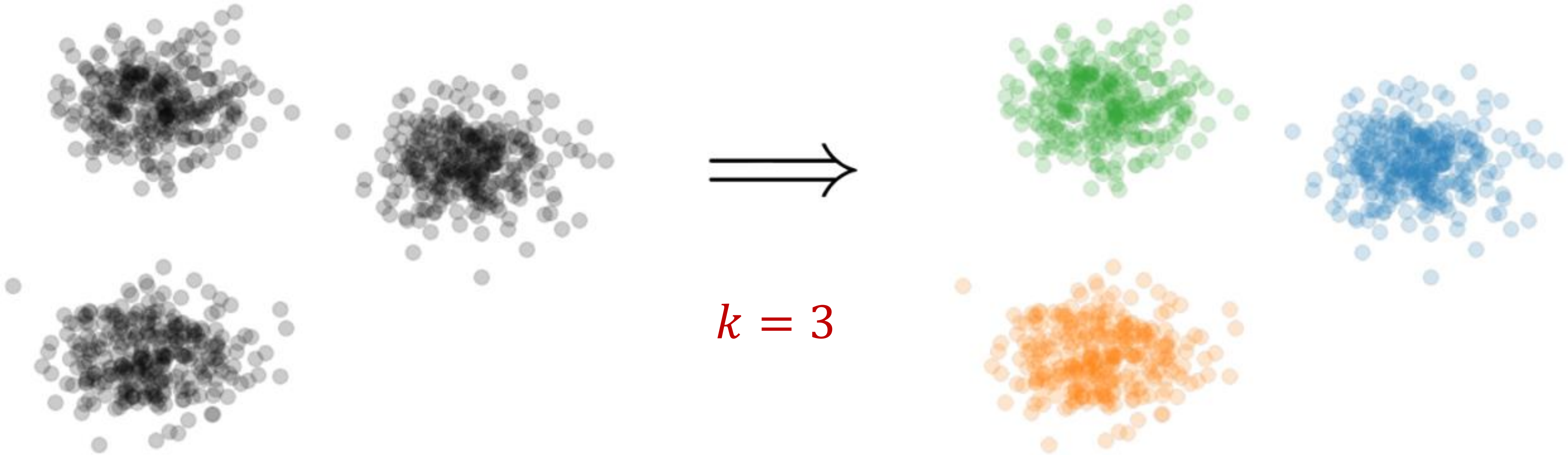
David P. Woodruff

Samson Zhou

Goal: Quickly cluster a stream of n points using $O(1)$ space

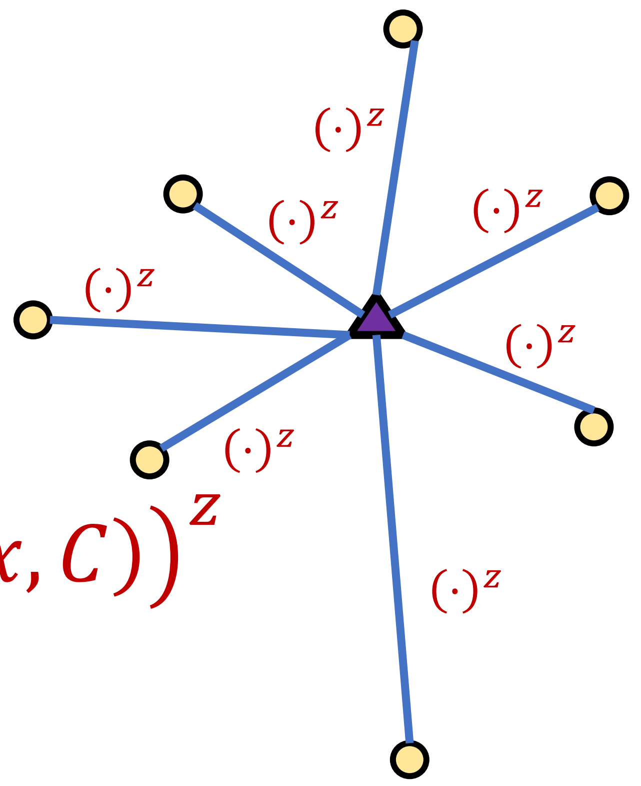
k-Clustering

- Goal: Given input dataset X, find a set C of k centers that implicitly partition X into at most k different clusters, while minimizing some associated cost function of the clustering



k-Clustering

- Define clustering cost $\mathrm{Cost}(X, C)$ to be a function of $\{\mathrm{dist}(x, C)\}_{x \in X}$
- k-median: $\mathrm{Cost}(X, C) = \sum_{x \in X} \mathrm{dist}(x, C)$
- k-means: $\mathrm{Cost}(X, C) = \sum_{x \in X} \left(\mathrm{dist}(x, C)\right)^2$
- (k, z)-clustering: $\mathrm{Cost}(X, C) = \sum_{x \in X} \left(\mathrm{dist}(x, C)\right)^z$



Euclidean k-Clustering

- For Euclidean k-clustering, input points $X = x_1, \ldots, x_n$ are in $\mathbb{R}^d$ (for us, they will be in $[\Delta]^d := \{1,2, \ldots, \Delta\}^d$)

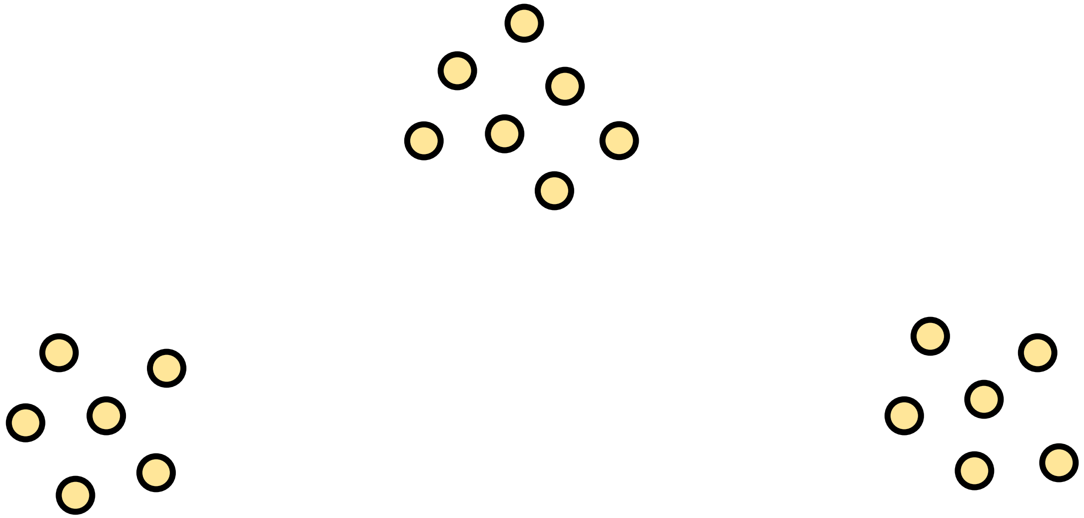
- $\mathrm{dist}(x, y) = \sqrt{(x_1 - y_1)^2 + \cdots + (x_d - y_d)^2}$ is the Euclidean distance

- (k, z)-clustering problem:

$$\min_{C:|C|\leq k} \mathrm{Cost}(X, C) = \min_{C:|C|\leq k} \Sigma_{x\in X} \left(\mathrm{dist}(x, C)\right)^z$$

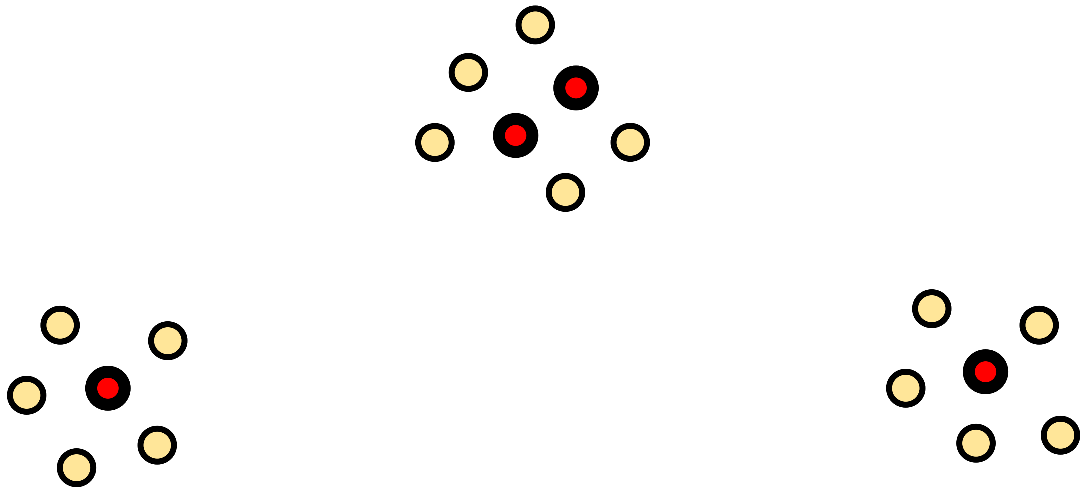
A Streaming Model

- Input: Underlying data set X arrives sequentially
- Output: Output a "good" set C_t of k centers at each time t
- Goal: Use space *sublinear* in the size n of the input X, with fast update time



Coreset

- Weighted (w) subset X' of representative points of X for a specific clustering objective



- For all sets C with $|C| = k$,

$$(1 - \varepsilon)\mathrm{Cost}(X, C) \leq \mathrm{Cost}(X', C, w) \leq (1 + \varepsilon)\mathrm{Cost}(X, C)$$

Coreset Constructions

- For (k, z)-clustering, there exist coreset constructions that only require $\tilde{O}\left(\frac{k}{\min(\varepsilon^4, \varepsilon^{2+z})}\right)$ weighted points of the input [CSS21, CLSS22, CLSSS22, BCJKSTW22, BCPSS24]

- *Independent* of input size n

Goal #1: Cluster a stream of n points using $O(1)$ space, i.e., independent of size n

Goal #2: Cluster a stream of n points using $o(k)$ amortized update time

Our Results (I)

- There exists a one-pass algorithm on insertion-only streams that maintains $(1+\varepsilon)$-coreset for (k,z)-clustering at *all times in the stream* and uses:

    - $\tilde{O}\left(\frac{dk}{\min(\varepsilon^4,\varepsilon^{2+z})}\right)$ words of space

    - $d\log(k)\cdot\mathrm{polylog}(\log(n\Delta))$ amortized update time

Corollary

- There exists a one-pass algorithm on insertion-only streams that maintains $O(z)$-approximation for (k, z)-clustering at *all times in the stream* and uses:
  - $\tilde{O}\left(\frac{dk}{\min(\varepsilon^4, \varepsilon^{2+z})}\right)$ words of space
  - $d \log(k) \cdot \operatorname{polylog}(\log(n\Delta))$ amortized update time

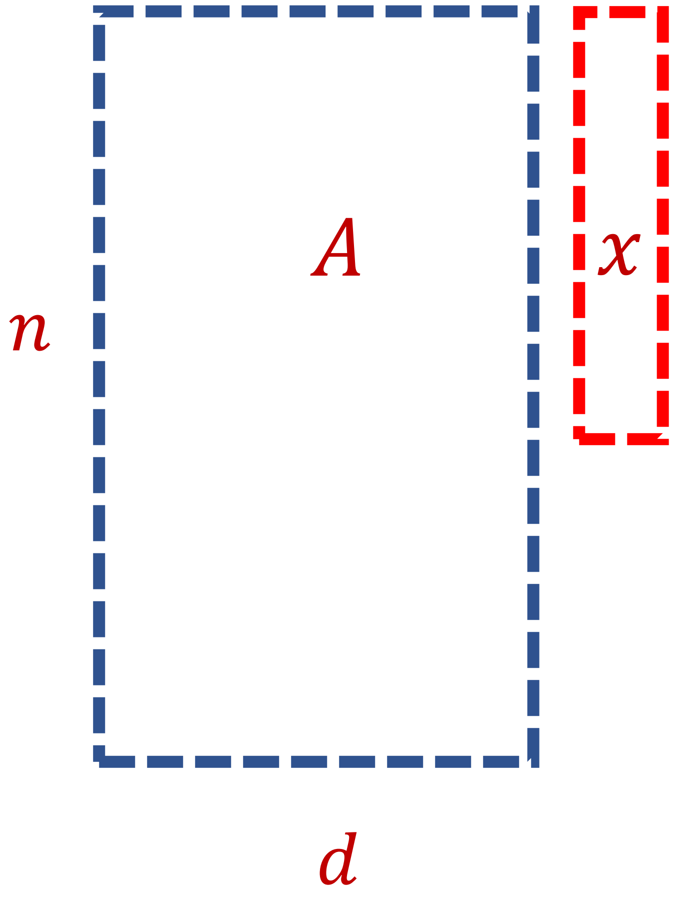
Streaming algorithm	Words of Memory
[HK07], $z \in \{1,2\}$	$\tilde{\mathcal{O}}\left(\frac{dk^{1+z}}{\varepsilon^{\mathcal{O}(d)}} \log^{d+z} n\right)$
[HM04], $z \in \{1,2\}$	$\tilde{\mathcal{O}}\left(\frac{dk}{\varepsilon^{d}} \log^{2d+2} n\right)$
[Che09], $z \in \{1,2\}$	$\tilde{\mathcal{O}}\left(\frac{d^2k^2}{\varepsilon^{2}} \log^{8} n\right)$
[FL11], $z \in \{1,2\}$	$\tilde{\mathcal{O}}\left(\frac{d^2k}{\varepsilon^{2z}} \log^{1+2z} n\right)$
Sensitivity and rejection sampling [BFLR19]	$\tilde{\mathcal{O}}\left(\frac{d^2k^2}{\varepsilon^{2}} \log n\right)$
Online sensitivity sampling	$\tilde{\mathcal{O}}\left(\frac{d^2k^2}{\varepsilon^{2}} \log^{2} n\right)$
Merge-and-reduce with coreset of [CLSS22]	$\tilde{\mathcal{O}}\left(\frac{dk}{\min(\varepsilon^4,\varepsilon^{2+z})} \log^{4} n\right)$
[CWZ23]	$\tilde{\mathcal{O}}\left(\frac{dk}{\min(\varepsilon^4,\varepsilon^{2+z})}\right) \cdot \text{polylog}(\log n)$
Theorem 1.1 (this work)	$\tilde{\mathcal{O}}\left(\frac{dk}{\min(\varepsilon^4,\varepsilon^{2+z})}\right)$

Fig. 1: Table of (k, z)-clustering algorithms on insertion-only streams. We summarize existing results with $z = \mathcal{O}(1)$, $\Delta = \text{poly}(n)$, and the assumption that $k > \frac{1}{\varepsilon^z}$ for the purpose of presentation.

Streaming algorithm	Amortized Update Time
[HK20]	$k^2 \cdot \mathrm{polylog}(n\Delta)$
[BCLP23]	$k \cdot \mathrm{polylog}(n\Delta)$
Theorem 1.1 (this work)	$\log(k) \cdot \mathrm{polylog}(\log(n\Delta))$

Fig. 2: Table of (k, z)-clustering algorithms on data streams, omitting linear dependencies in the dimension d. We remark that [HK20, BCLP23] can handle the fully-dynamic setting, whereas ours cannot. On the other hand, our algorithm uses sublinear space while theirs does not.

Subspace Embedding



- Subspace embedding: Given $\varepsilon > 0$ and $A \in R^{n \times d}$, find matrix $M \in R^{m \times d}$ with $m \ll n$, such that for *every* $x \in \mathbb{R}^d$,

$$(1-\varepsilon)\|Ax\|_2 \leq \|Mx\|_2 \leq (1+\varepsilon)\|Ax\|_2$$

- Equivalent to $(1-\varepsilon)A^\top A \preccurlyeq M^\top M \preccurlyeq (1+\varepsilon)A^\top A$
- Can be used to approximate *all* cuts of a graph when rows of A correspond to graph edges

Our Results (II)

- There exists a one-pass algorithm on insertion-only streams with online condition number κ and maximum entry M that maintains $(1+\varepsilon)$-coreset for subspace embeddings at *all times in the stream* and uses:
  - $\tilde{O}(d^2 \log(nM)) + \frac{d^2}{\varepsilon^2} \cdot \mathrm{polylog}\left(d, \frac{1}{\varepsilon}, \log(n\kappa)\right)$ words of space
  - $O(d)$ amortized update time

Our Results (III)

- Structural properties in “efficient encoding” also give improved communication bounds for clustering in distributed models

Upcoming

- (k, z)-Clustering in $O(1)$ Space

Questions?

Goal #1: Cluster a stream of n points using $O(1)$ space, i.e., independent of size n

Goal #2: Cluster a stream of n points using $o(k)$ amortized update time

(k, z)-Clustering in the Streaming Model

- Merge-and-reduce framework

- Example: Suppose there exists a $(1+\varepsilon)$-coreset construction for k-means clustering that uses $f\left(k, \frac{1}{\varepsilon}\right)$ weighted input points

$\boxed{\tilde{O}\left(\frac{k}{\varepsilon^4}\right)}$

(k, z)-Clustering in the Streaming Model

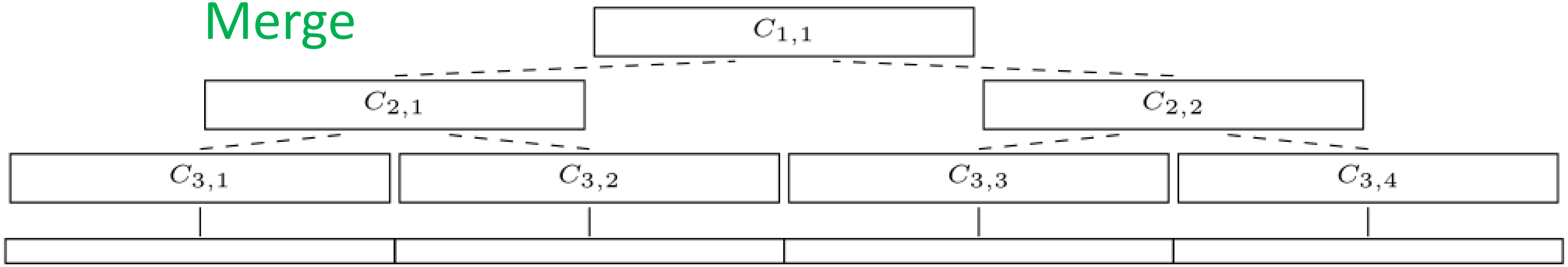
- Partition the stream into blocks containing $f\left(k, \frac{\log n}{\varepsilon}\right)$ points
- Create a $\left(1 + \frac{\varepsilon}{\log n}\right)$-coreset for each block

$\boxed{\tilde{O}\left(\frac{k}{\varepsilon^4}\log^2 n\right)}$

- Create a $\left(1 + \frac{\varepsilon}{\log n}\right)$-coreset for the set of points formed by the union of two coresets for each block

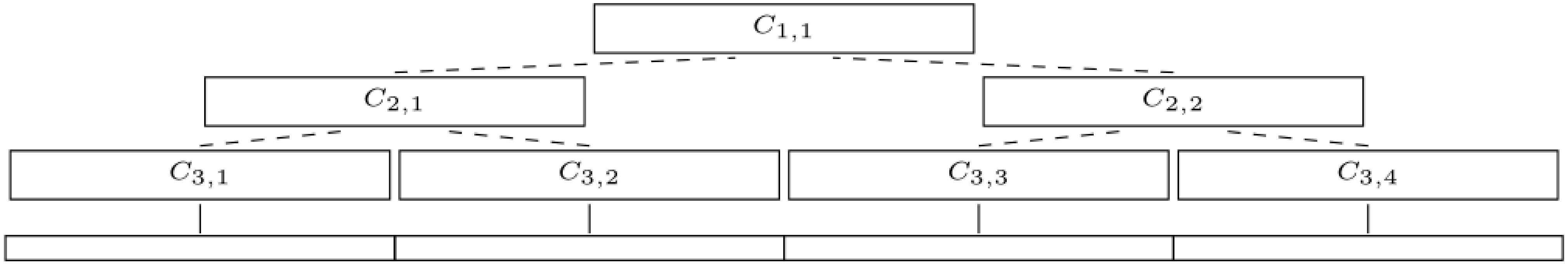
Merge

Reduce



(k, z)-Clustering in the Streaming Model

- There are $O(\log n)$ levels
- Each coreset is a $\left(1 + \frac{\varepsilon}{\log n}\right)$-coreset of two coresets
- Total approximation is $\left(1 + \frac{\varepsilon}{\log n}\right)^{\log n} = (1 + O(\varepsilon))$



(k, z)-Clustering in the Streaming Model

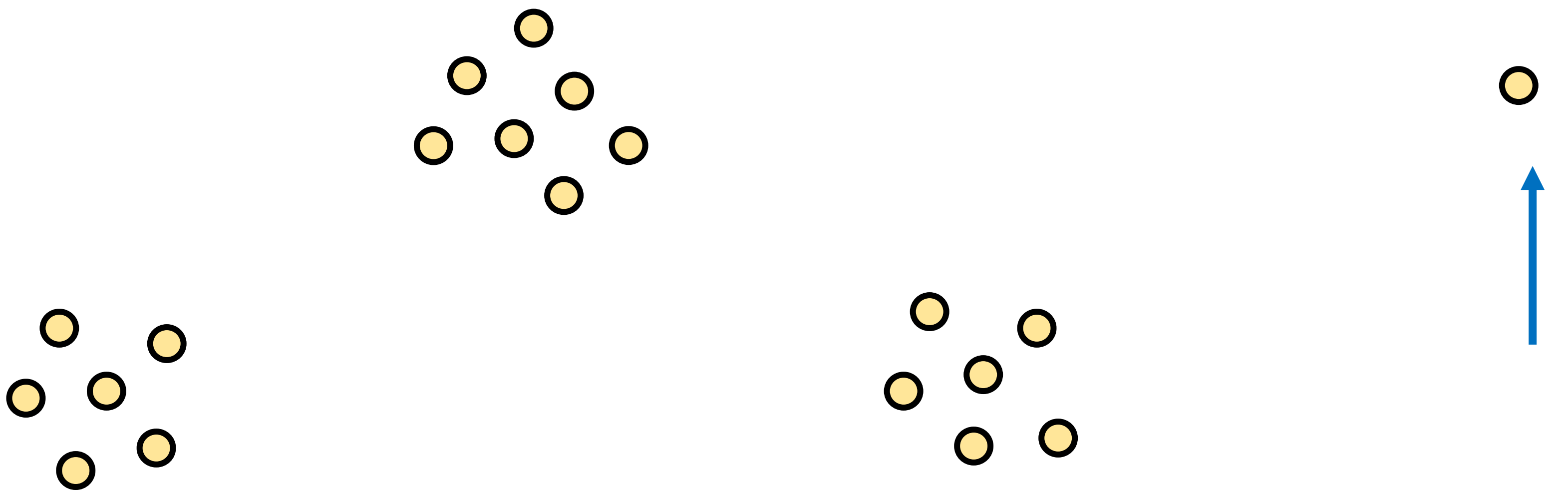
- Total space is $f\left(k, \frac{\log n}{\varepsilon}\right) \cdot O(\log n)$ points

For k-means clustering, this is $\tilde{O}\left(\frac{k}{\varepsilon^4} \cdot \log^3 n\right)$ points

Not independent of size n of the stream length!

Sensitivity Sampling

- The quantity $s(x) = \max_{C} \frac{\mathrm{Cost}(x,C)}{\mathrm{Cost}(X,C)}$ is called the *sensitivity* of x and intuitively measures how "important" the point x is



Online Sensitivity Sampling

- In a data stream, computing/approximating sensitivity $s(x) = \max_C \frac{\text{Cost}(x,C)}{\text{Cost}(X,C)}$ requires seeing the entire dataset X, but then it is too late to sample x

- We define the *online sensitivity* of x_t with respect to a stream $x_1, \dots, x_n$ to be $\varphi(x_t) = \max_C \frac{\text{Cost}(x_t,C)}{\text{Cost}(X_t,C)}$, where $X_t = x_1, \dots, x_t$, which intuitively measures how "important" the point x is *SO FAR*

Online Sensitivity Sampling

- Observation: we can use a $(1+\varepsilon)$-coreset to obtain a $(1+\varepsilon)$-approximation to $\varphi(x_t)$

- Use samples obtained from online sensitivity sampling at each time $t-1$ to obtain a $(1+\varepsilon)$-approximation to $\varphi(x_t)$

- Can then perform online sensitivity sampling at time t and by induction, at all times in the stream

Online Sensitivity Sampling

- The *total online sensitivity* of $X = (x_1, \dots, x_n)$ is $\sum_{t\in[n]} \varphi(x_t)$
- Quantifies how many points will be sampled

- Total online sensitivity is $O(k \log^2(nd\Delta))$ → we get a coreset of size $\sum_t p(x_t) = \frac{k^2 d}{\varepsilon^2} \cdot \mathrm{polylog}(n\Delta)$ (after a union bound)

- Sampling is done online, can view as a new stream X'

Insertion-Only Algorithm [CWZ23]

1. Perform online sensitivity sampling to *implicitly* create new stream X'
2. In parallel, run merge-and-reduce on X'

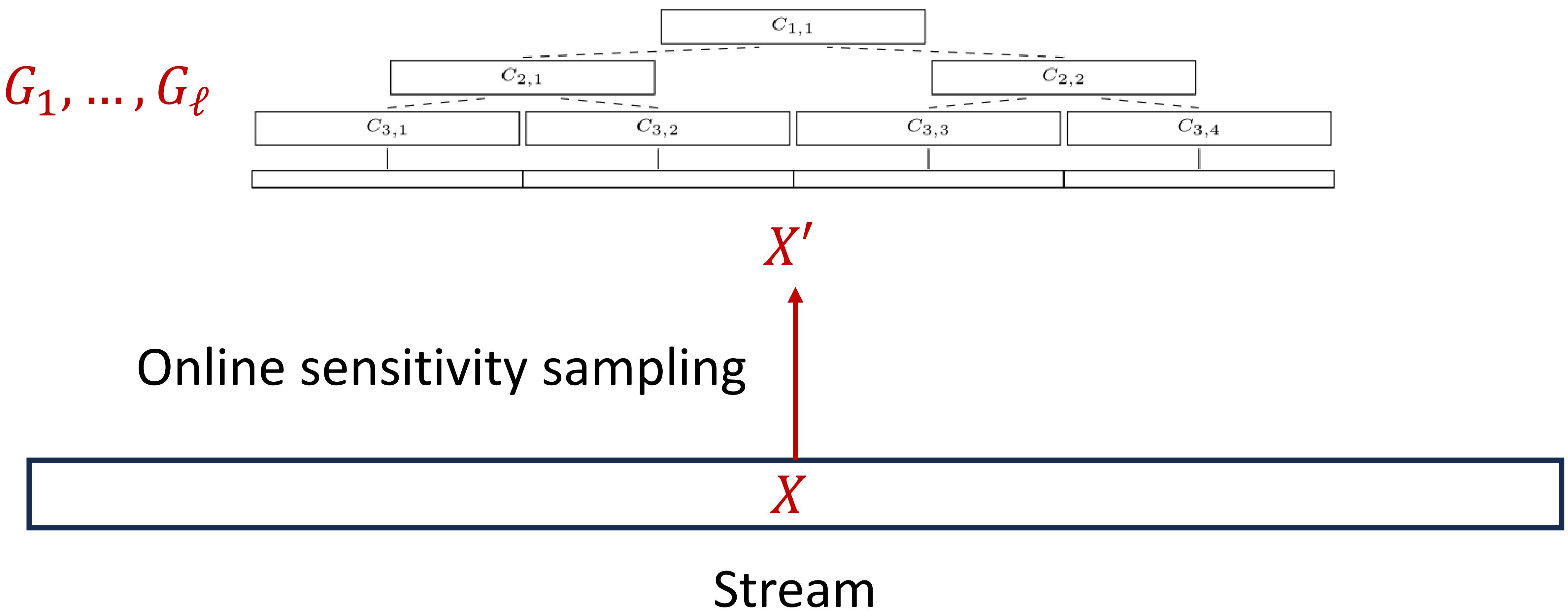
Insertion-Only Algorithm [CWZ23]

- New stream X' has length $\frac{k^2 d}{\varepsilon^2} \cdot \mathrm{polylog}(n\Delta)$
- Run merge-and-reduce on X'
- Recall: merge-and-reduce for k-means stored $\tilde{O}\left(\frac{k}{\varepsilon^4} \cdot \log^3 n\right)$ points, but n was the length of the stream

- Total number of points now is $\tilde{O}\left(\frac{k}{\varepsilon^4}\right) \cdot \mathrm{polylog}\left(k, d, \frac{1}{\varepsilon}, \log(n\Delta)\right)$

Insertion-Only Algorithm

- Total number of points now is $\tilde{O}\left(\frac{k}{\varepsilon^4}\right) \cdot \mathrm{polylog}\left(k, d, \frac{1}{\varepsilon}, \log(n\Delta)\right)$
- Specifically, there are $\mathrm{polylog}\left(k, d, \frac{1}{\varepsilon}, \log(n\Delta)\right)$ groups $G_1, \ldots, G_\ell$ of $\tilde{O}\left(\frac{k}{\varepsilon^4}\right) \cdot \mathrm{polylog}\left(k, d, \frac{1}{\varepsilon}, \log(n\Delta)\right)$ points
- If we want independence of size n of the stream length, cannot afford to store all points explicitly


$G_1, \ldots, G_\ell$
$C_{1,1}$
$C_{2,1}$
$C_{2,2}$
$C_{3,1}$
$C_{3,2}$
$C_{3,3}$
$C_{3,4}$
X'
Online sensitivity sampling
X
Stream

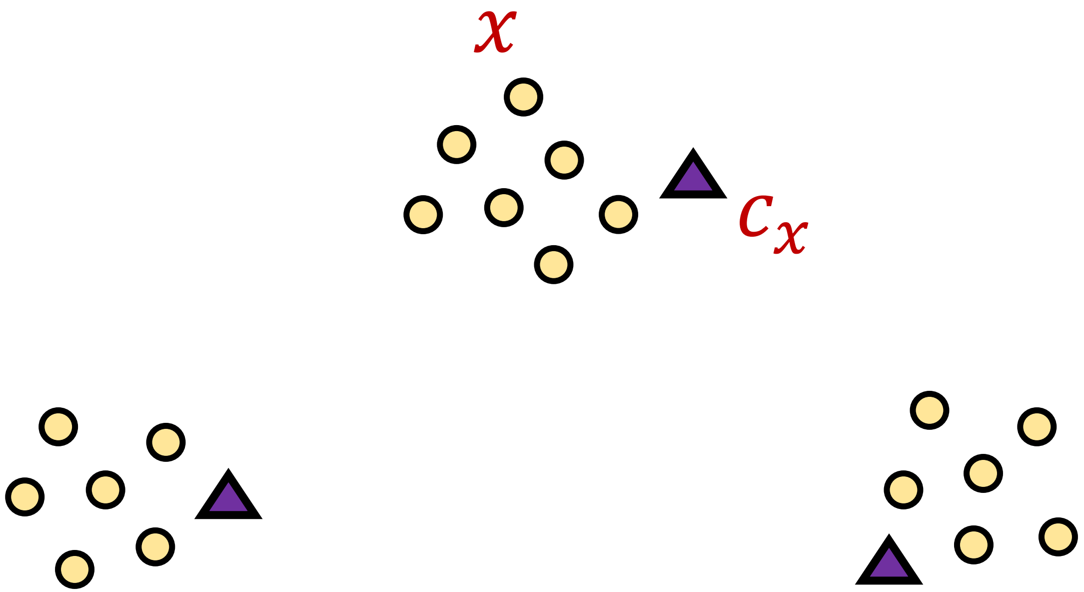
Efficient Local Encoding

- Look at a specific group G_i and compute a near-optimal solution S_i
- Store *offset* of each point from one of the centers of S_i

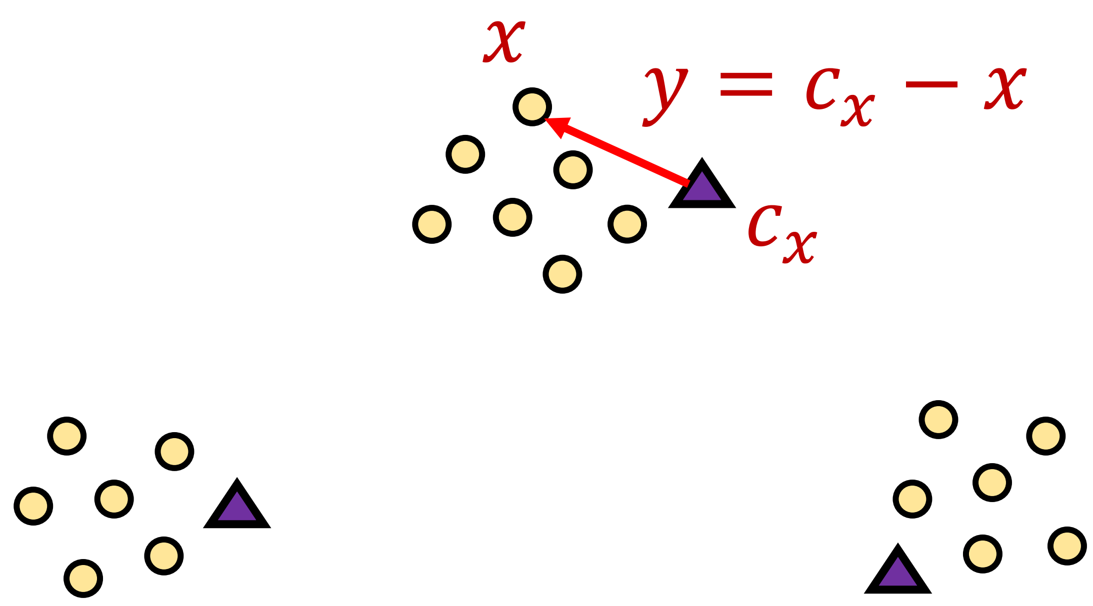
- For each point $x \in G_i$, let c_x be the closest center of S_i and $y = c_x - x$

- Round y coordinate-wise to nearest power of $1 + \varepsilon'$ and store the vector of exponents $\tilde{y}$

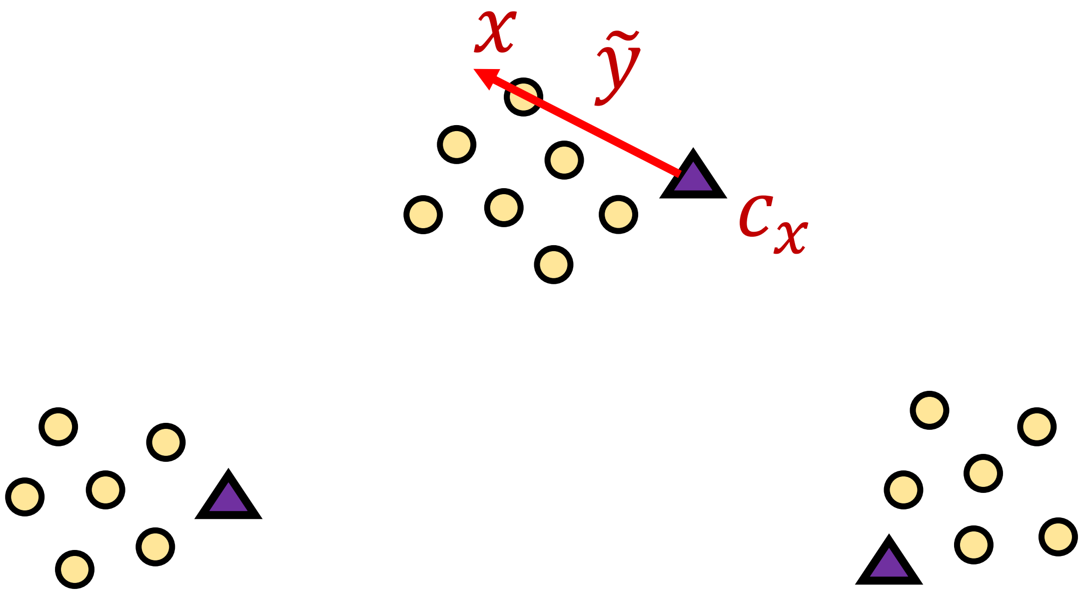
Efficient Local Encoding



Efficient Local Encoding

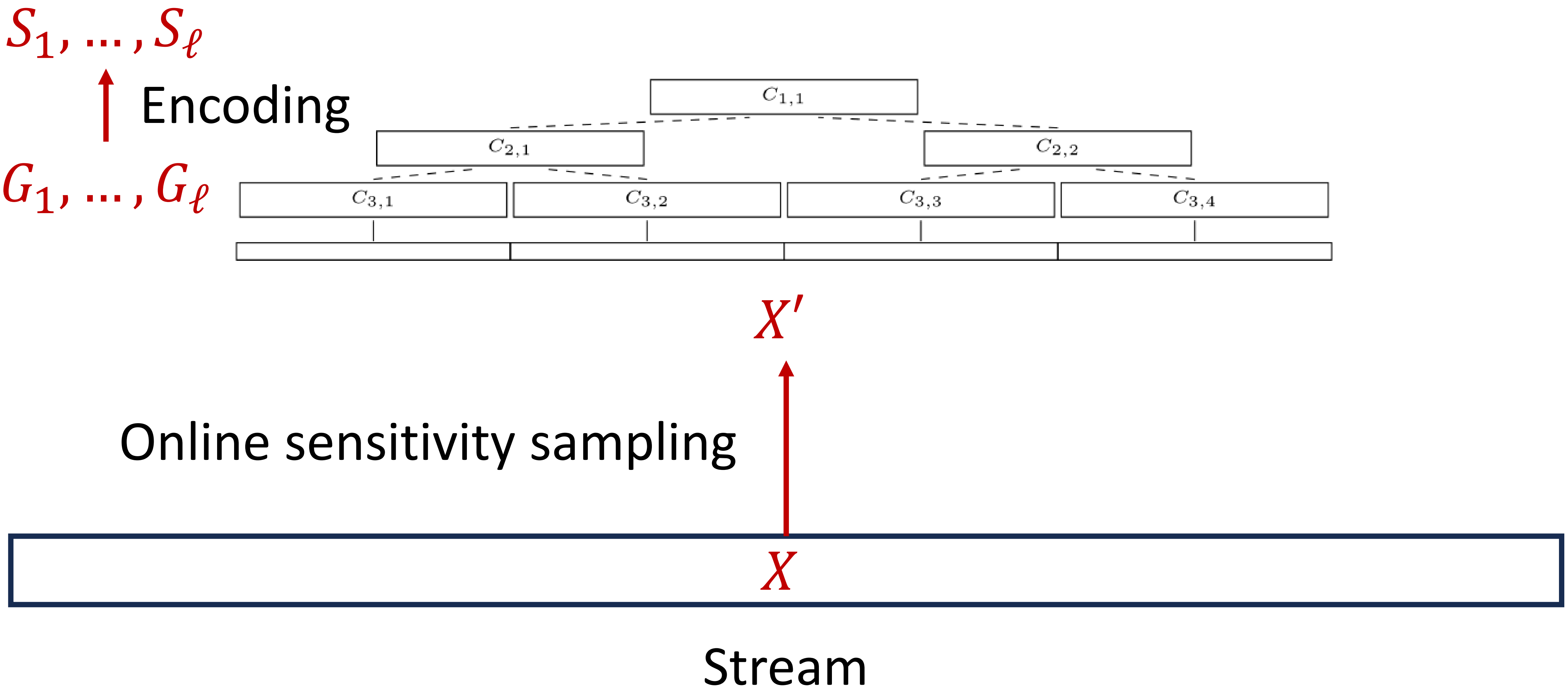


Efficient Local Encoding



Efficient Local Encoding

- Round y coordinate-wise to nearest power of $1 + \varepsilon'$ and store the vector of exponents $\tilde{y}$
- For $\varepsilon' = \mathrm{poly}\left(\frac{\varepsilon}{\log(nd\Delta)}\right)$:
  - Results in a $\left(1 + O(\varepsilon)\right)$-coreset of G_i
  - Encoding each point uses $d \cdot \mathrm{polylog}\left(\frac{1}{\varepsilon}, \log(nd\Delta)\right)$ bits
  - Encoding each group G_i uses $\tilde{O}\left(\frac{dk}{\varepsilon^4}\right)$ words of space, e.g., for S_i
- However, there are $\mathrm{polylog}\left(k, d, \frac{1}{\varepsilon}, \log(n\Delta)\right)$ groups $G_1, \ldots, G_\ell$


$S_1, \dots, S_\ell$
Encoding
$G_1, \dots, G_\ell$
$C_{1,1}$
$C_{2,1}$
$C_{2,2}$
$C_{3,1}$
$C_{3,2}$
$C_{3,3}$
$C_{3,4}$
X'
Online sensitivity sampling
X
Stream

Global Encoding

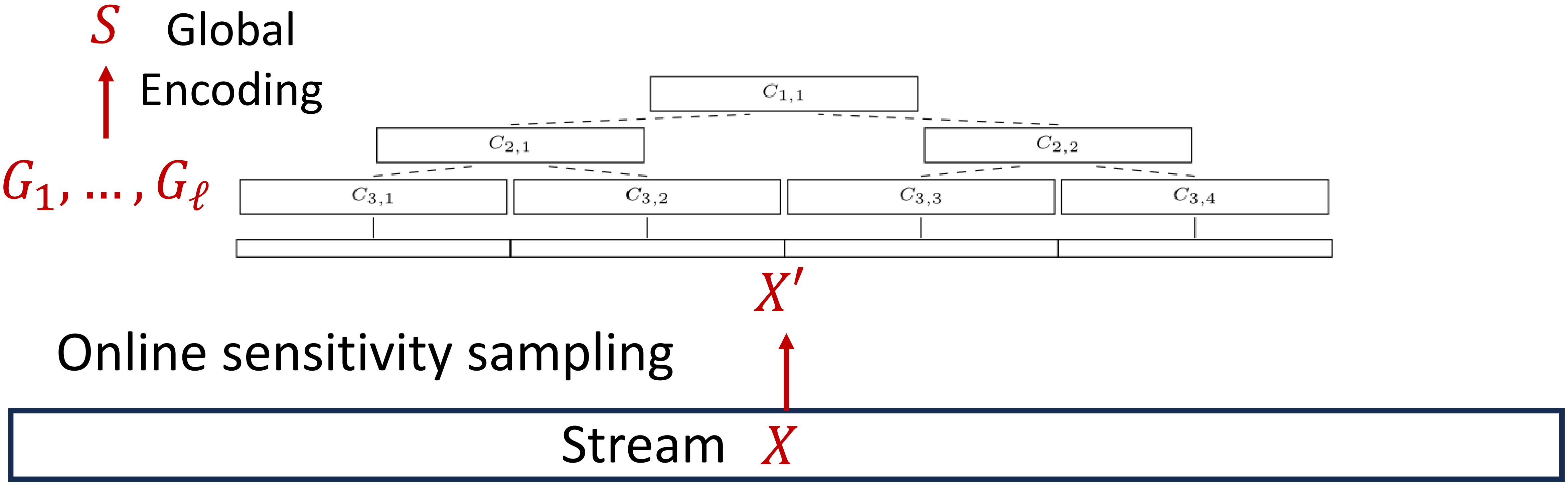
- Instead of storing a near-optimal solution S_i for each group G_i, store a single near-optimal global solution S

- Store *offset* of each point from one of the centers of S
- For each point $x \in G_i$, let c_x be the closest center of S and $y = c_x - x$
- Round y coordinate-wise to nearest power of $1 + \varepsilon'$ and store the vector of exponents $\tilde{y}$

Global Encoding

- For $\varepsilon' = \mathrm{poly}\left(\frac{\varepsilon}{\log(nd\Delta)}\right)$:
  - Rounded points no longer provide a $\left(1 + O(\varepsilon)\right)$-coreset of each G_i, but give $O(\varepsilon) \cdot \mathrm{OPT}$ additive error, so $\left(1 + O(\varepsilon)\right)$-approximation overall

- Global encoding uses $\tilde{O}\left(\frac{dk}{\varepsilon^4}\right)$ total words of space



1. Perform online sensitivity sampling to *implicitly* create new stream X'
2. In parallel, run merge-and-reduce on X'
3. Efficient global encoding on resulting coresets

Upcoming

- (k, z)-Clustering in $o(k)$ Amortized Update Time

Questions?

Fast Clustering

- Algorithm bottleneck: approximation of online sensitivities for the sampling process to form the stream X'

- Computing a "good" approximation to sensitivities is often as hard as computing a "good" approximation to clustering

- Constant-factor approximation in time $O(dn^2)$ [GT08]
- We can view $n = \tilde{O}\left(\frac{k}{\varepsilon^4}\right) \cdot \text{polylog}(d, \log(n\Delta))$

Fast Clustering

- Insight: Previous algorithm utilized a significantly smaller stream X' with length $\frac{k^2 d}{\varepsilon^2} \cdot \mathrm{polylog}(n\Delta)$
- Repeat this idea another level!

- Create stream $\tilde{X}$ with length $n^{1-c} \cdot \frac{k^2 d}{\varepsilon^2} \cdot \mathrm{polylog}(n\Delta)$

Fast Clustering

- Create stream $\tilde{X}$ with length $n^{1-c} \cdot \frac{k^2 d}{\varepsilon^2} \cdot \operatorname{polylog}(n\Delta)$
- Can again use online sensitivity sampling of X to form $\tilde{X}$
- Now just need n^{1-c}-approximations for sensitivities

Fast Clustering

- Theorem: For any constant $c \in (0,1)$, there exists an algorithm that computes n^{1-c}-approximations to the sensitivities of a batch of k points of X using $d \log(k) \cdot \mathrm{polylog}(\log(n\Delta))$ amortized update time

Structural Property

- Both clustering costs and sensitivities are distorted by a $O(1)$ when the cluster centers is among the input points

- $s(x) = \max_{C} \frac{\mathrm{Cost}(x,C)}{\mathrm{Cost}(X,C)}$ is an optimization problem

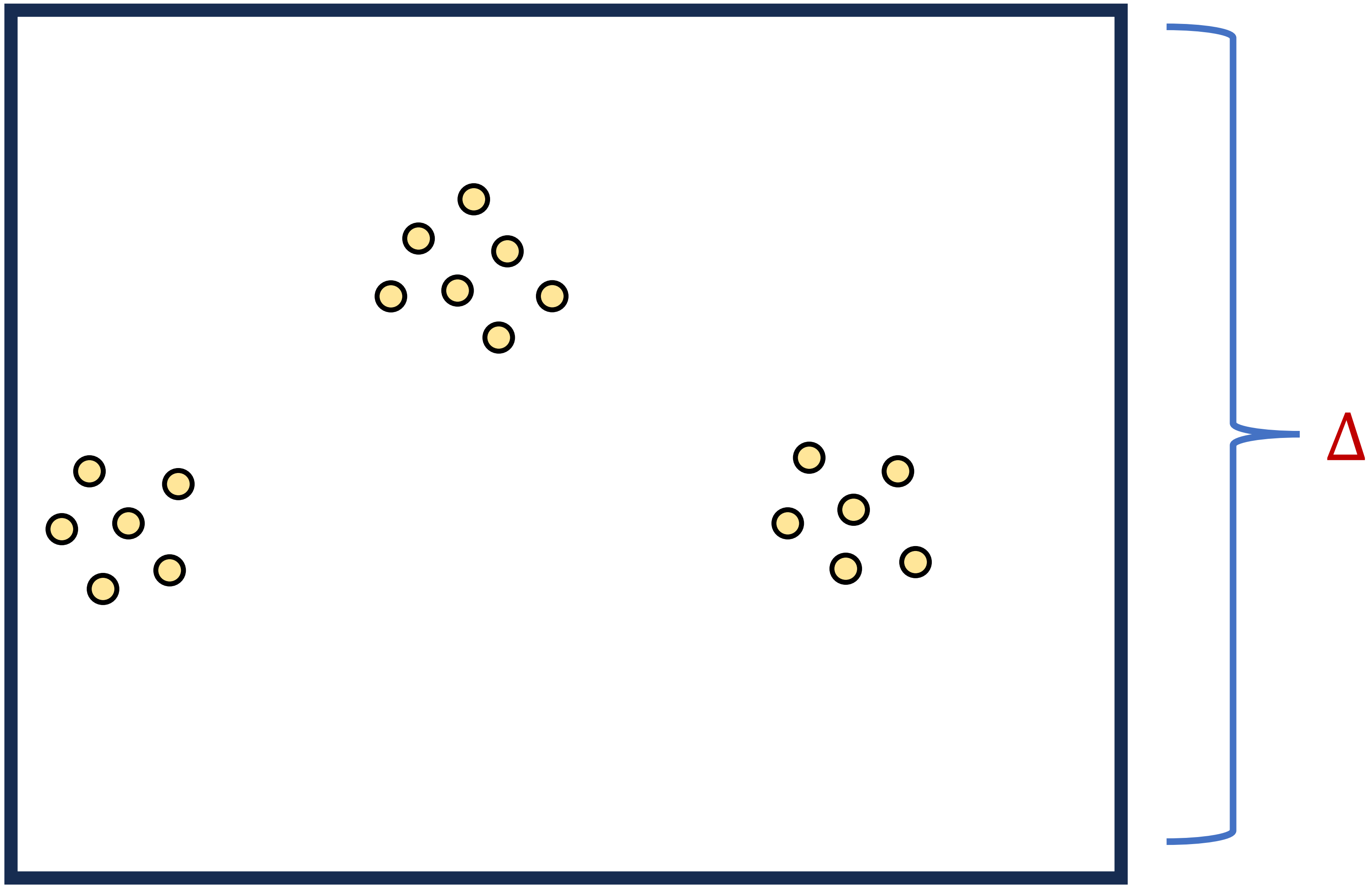
- Fast enumeration over the center serving a point that realizes the sensitivity when $|\mathrm{X}| = \tilde{O}\left(\frac{k}{\varepsilon^4}\right) \cdot \mathrm{polylog}(d, \log(n\Delta))$

Structural Property

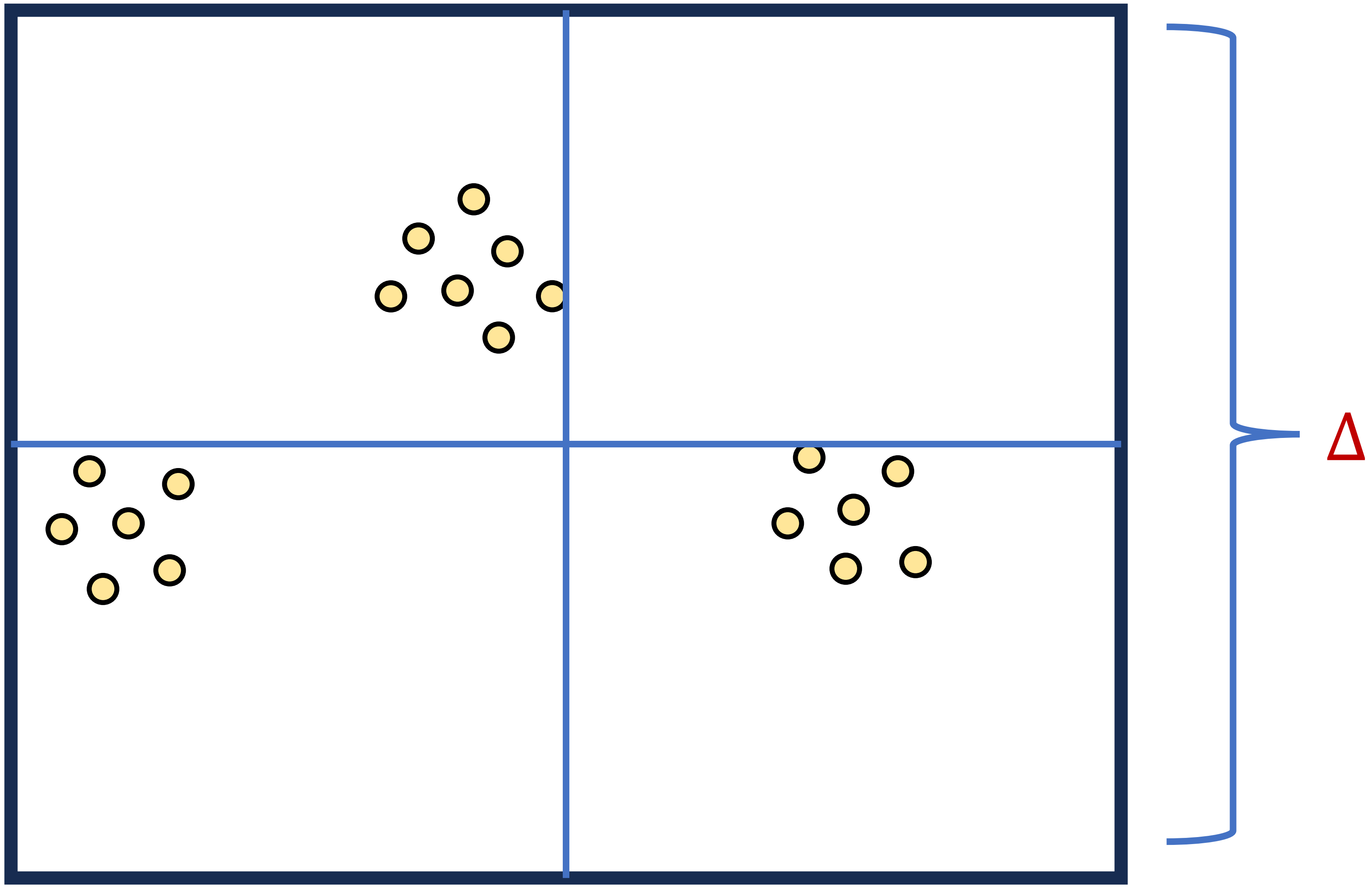
- Fast enumeration over the center serving a point that realizes the sensitivity when $|\mathrm{X}| = \tilde{O}\left(\frac{k}{\varepsilon^4}\right) \cdot \mathrm{polylog}(d, \log(n\Delta))$

- For a fixed center c serving a point x, $\max_{C} \frac{\mathrm{Cost}(x,C)}{\mathrm{Cost}(X,C)}$ is now a constrained optimization problem for minimizing $\mathrm{Cost}(X, C)$, since no center in C can be closer to x than c

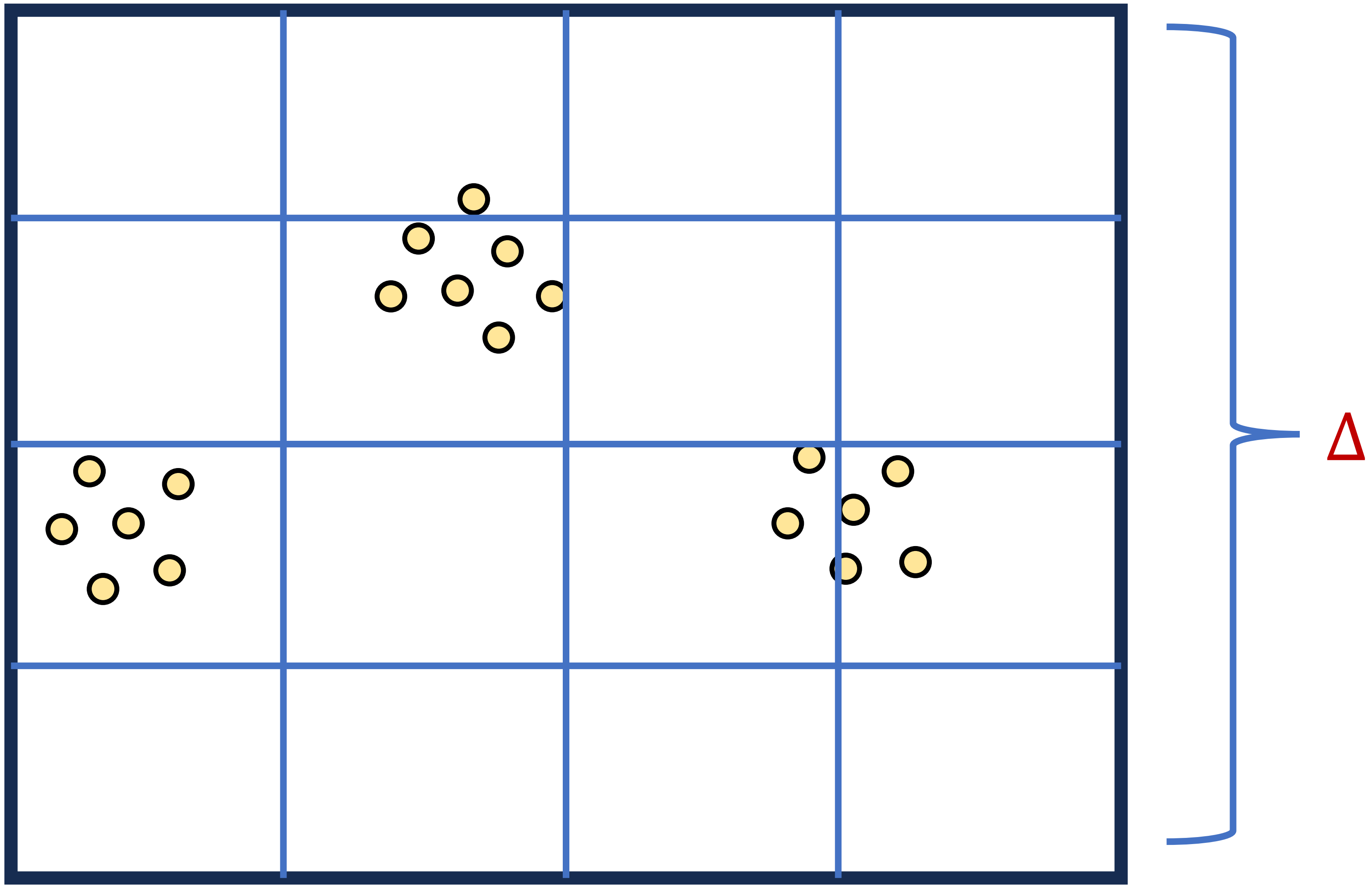
Quadtree Embedding



Quadtree Embedding



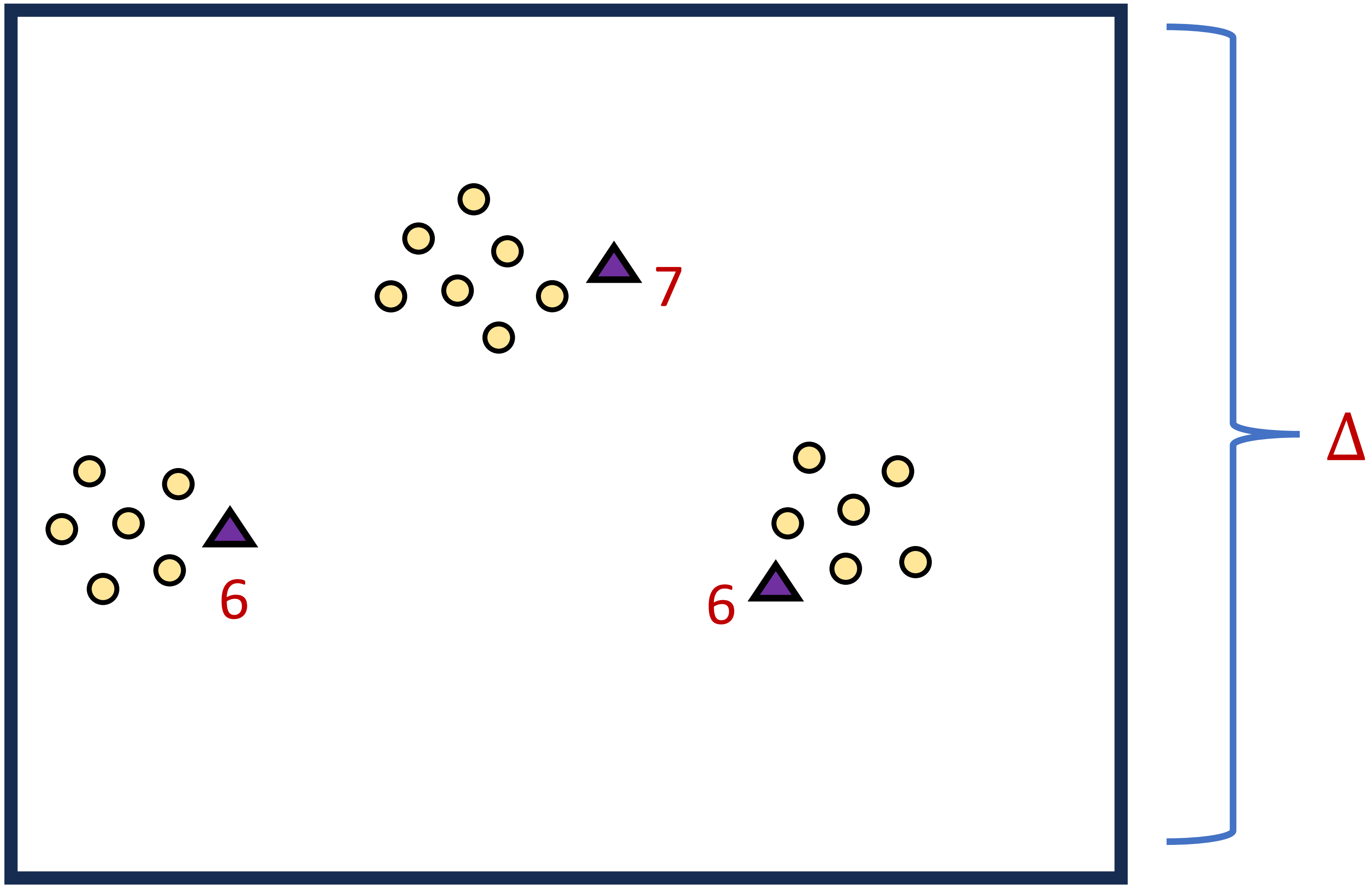
Quadtree Embedding



Quadtree Embedding

Total cost: 0

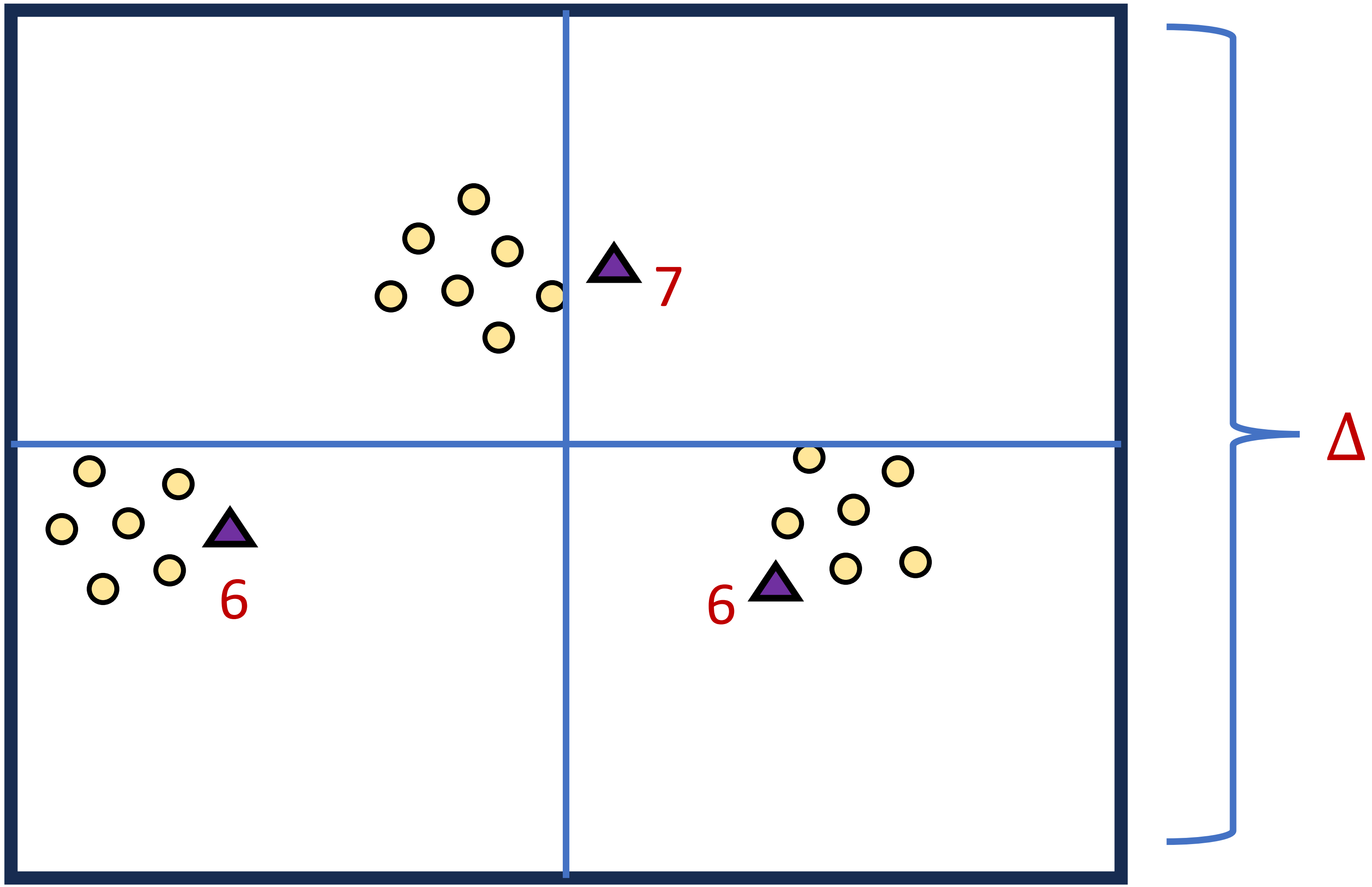
Level cost: 0



Quadtree Embedding

Total cost: $\frac{\Delta}{2} \cdot 7$

Level cost: $\frac{\Delta}{2} \cdot 7$

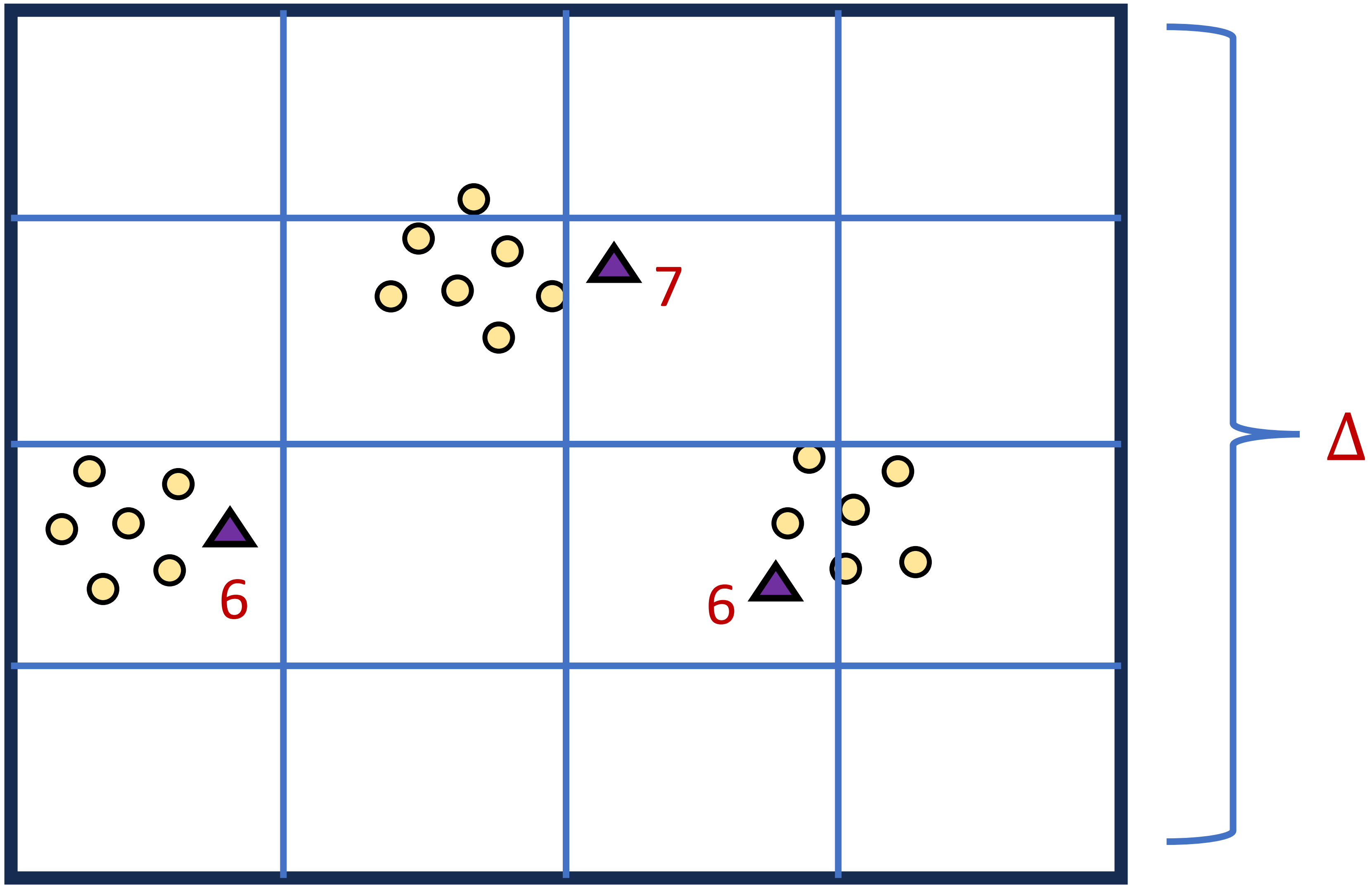


Quadtree Embedding

Total cost:

$\left(\frac{7}{2} + \frac{11}{4}\right)\Delta$

Level cost: $\frac{\Delta}{4} \cdot 11$

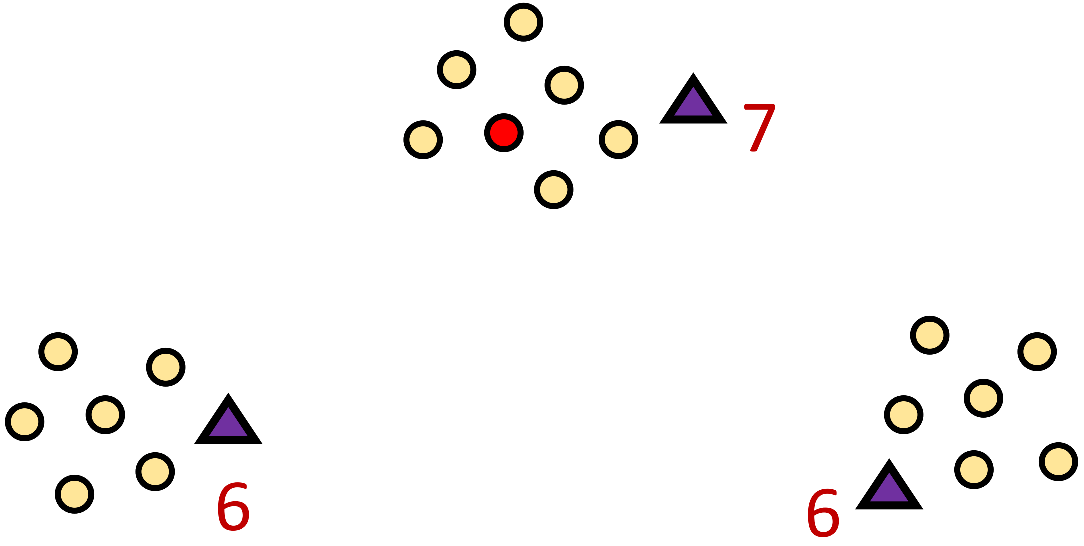


Fast Clustering

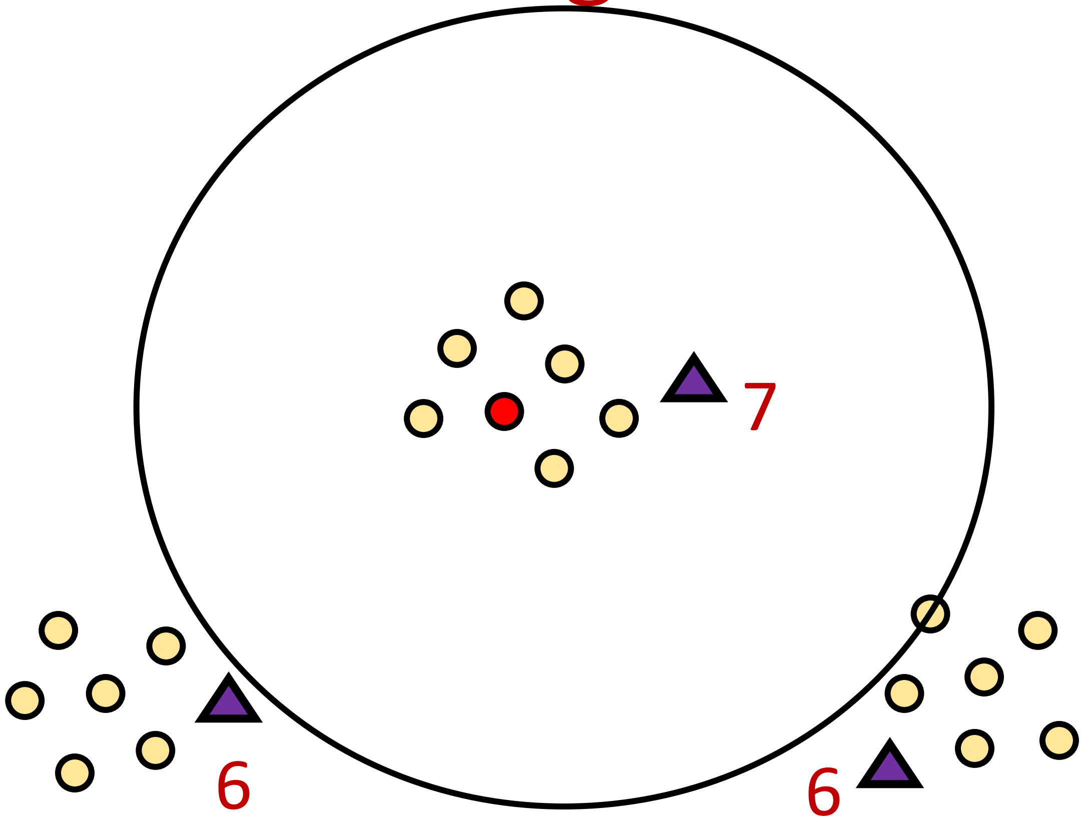
- Choose side length of grid ℓ to be ζ^{ℓ} for $\zeta = O(n^{1-c})$, so there are only $O\left(\frac{1}{c}\right)$ levels in the quadtree
- Makes nearest-neighbor search much faster

- Can quickly solve a near-optimal clustering problem, generalizing an algorithm of [CLNSS20]
- Additional structural result to quickly approximate constrained clustering problem

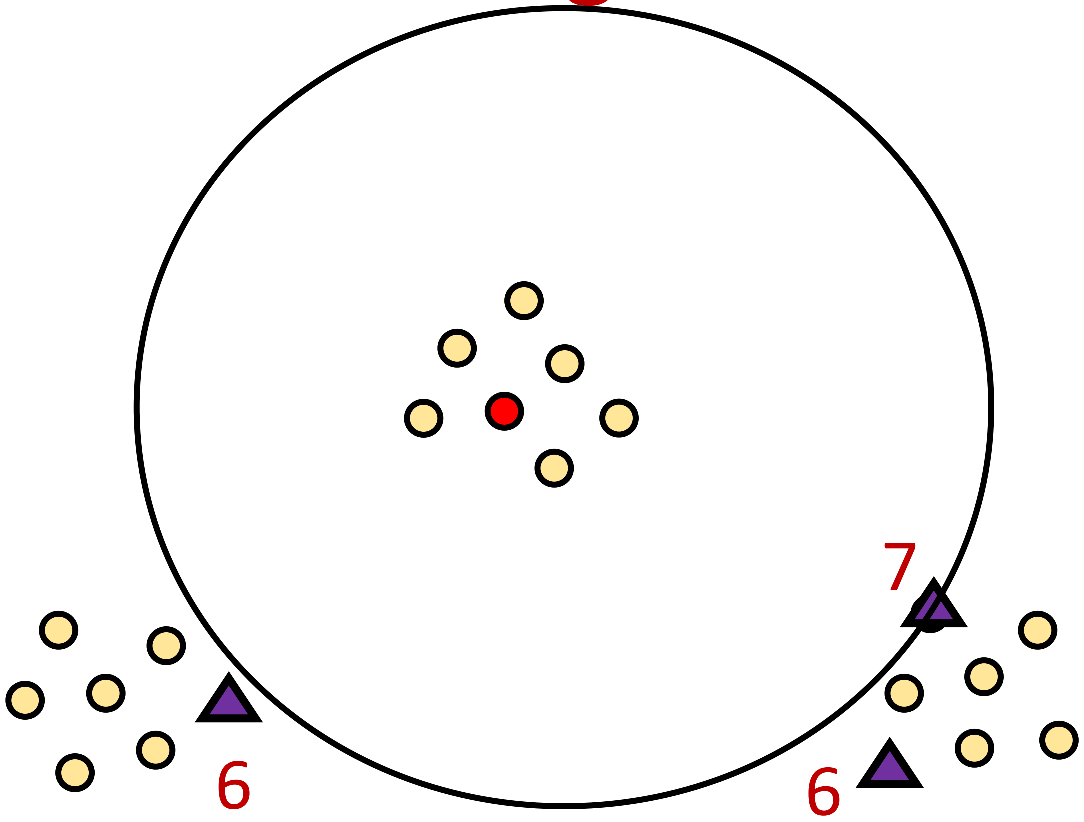
Constrained Clustering



Constrained Clustering



Constrained Clustering



Upcoming

- Subspace embeddings

Questions?

1. Perform online leverage score sampling to *implicitly* create new matrix A'
2. In parallel, run merge-and-reduce on A'
3. Efficient global encoding on resulting coresets

Efficient Local Encoding

- Suppose we have a constant-factor subspace embedding M for the matrix A
- How to achieve $(1+\varepsilon)$-factor subspace embedding?

- Previously: charged each point to closest center

- Round each coordinate in each row to a power of $(1+\varepsilon')$ after multiplying by a deterministic preconditioner of M

Crude Leverage Score Approximation

- Suppose we have a constant-factor subspace embedding M for the matrix A

- Let $Z = (M^\top M)^{-1/2}$ so that $\|Za_t\|_2^2$ is a constant-factor approximation to the leverage score

- $\|gZa_t\|_2^2$ is n^{1-c} approximation to $\|Za_t\|_2^2$ for random gaussian g
- Compute in $O(d)$ time since gZ does not change much over the stream

Summary

- We achieve one-pass algorithms on insertion-only streams that maintain $(1+\varepsilon)$-coreset for (k,z)-clustering and subspace embedding that use:
  - Words of space *independent* of stream length n (matching offline coreset constructions)
  - $d\log(k)\cdot\mathrm{polylog}(\log(n\Delta))$ amortized update time for clustering and $O(d)$ amortized update time for subspace embedding

Open Questions

- Does there exist an algorithm for low-rank approximation on insertion-only streams that uses words of space independent of stream length n?

- Does there exist an algorithm for graph sparsification on insertion-only streams that match the offline coreset constructions?