Differential privacy without a central database

Boston Differential Privacy Summer School, 6-10 June 2022

Uri Stemmer

About this course

- The local model ✓
- The shuffle model ✓
- Streaming/online settings ✓
- Differential privacy as a tool

Differential privacy as a tool

Today's Outline

1. DP is the enemy of overfitting
2. Application to answering adaptive queries
3. Application to adaptive streaming

Find the next number of the sequence

1, 3, 5, 7, ?

Find the next number of the sequence

1, 3, 5, 7, ?

Correct solution

217341

Find the next number of the sequence

1, 3, 5, 7, ?

Correct solution

217341

Because when

$$f(x) = \frac{18111}{2}x^4 - 90555\,x^3 + \frac{633885}{2}x^2 - 452773\,x + 217331$$

wow

many maths

very logic

great

$$f(1) = 1$$
$$f(2) = 3$$
$$f(3) = 5$$
$$f(4) = 7$$
$$\boxed{f(5) = 217341}$$

Find the next number of the sequence

1, 3, 5, 7, ?

Correct solution

217341

Because when

$$f(x) = \frac{18111}{2} x^4 - 90555 x^3 \ldots 331$$

Perfect fit, but clearly ridiculous

Overfitting: f behaves good on the data but fails to generalize

$f(2) = 3$

$f(3) = 5$

$f(4) = 7$

$$\boxed{f(5) = 217341}$$

very logic

great

The Generalization Properties of DP (“anti overfitting”)

Warmup 1:

- Let $\mathfrak{D}$ be a distribution over a domain X, and fix a predicate $h: X \to \{0,1\}$
- Let $S \sim \mathfrak{D}^n$. Then by the Hoeffding bound, w.h.p. we have $\frac{1}{|S|} \cdot \sum_{x \in S} h(x) \approx \mathbb{E}_{x \sim \mathfrak{D}}[h(x)]$

The Generalization Properties of DP ("anti overfitting")

Warmup 1:

- Let $\mathfrak{D}$ be a distribution over a domain X, and fix a predicate $h: X \to \{0,1\}$
- Let $S \sim \mathfrak{D}^n$. Then by the Hoeffding bound, w.h.p. we have $\frac{1}{|S|} \cdot \sum_{x \in S} h(x) \approx \mathbb{E}_{x \sim \mathfrak{D}}[h(x)]$

Warmup 2:

- Let $\mathfrak{D}$ be a distribution over a domain X
- Let $\mathcal{A}: X^n \to 2^X$ be an algorithm that takes a sample and outputs a predicate
- Let $S \sim \mathfrak{D}^n$ and let $h \leftarrow \mathcal{A}(S)$
- Can we claim that the empirical average is close to the expectation?
- Not in general. E.g., $\mathcal{A}$ might choose the function $h(x) = \mathbb{1}\{x \in S\}$

The Generalization Properties of DP ("anti overfitting")

Warmup 1:

- Let $\mathfrak{D}$ be a distribution over a domain X, and fix a predicate $h: X \to \{0,1\}$
- Let $S \sim \mathfrak{D}^n$. Then by the Hoeffding bound, w.h.p. we have $\frac{1}{|S|} \cdot \sum_{x \in S} h(x) \approx \mathbb{E}_{x \sim \mathfrak{D}}[h(x)]$

Warmup 2:

- Let $\mathfrak{D}$ be a distribution over a domain X
- Let $\mathcal{A}: X^n \to 2^X$ be an algorithm that takes a sample and outputs a predicate
- Let $S \sim \mathfrak{D}^n$ and let $h \leftarrow \mathcal{A}(S)$
- Can we claim that the empirical average is close to the expectation?
- Not in general. E.g., $\mathcal{A}$ might choose the function $h(x) = \mathbb{1}\{x \in S\}$

Theorem [DFHPRR'15, BNSSSU'16]:

- Let $\mathcal{A}: X^n \to 2^X$ be a differentially private algorithm that outputs a predicate $h: X \to \{0,1\}$
- Let $\mathfrak{D}$ be a distribution over X
- Let $S \sim \mathfrak{D}^n$ and let $h \leftarrow \mathcal{A}(S)$
- Then w.h.p. we have $\frac{1}{|S|} \cdot \sum_{x \in S} h(x) \approx \mathbb{E}_{x \sim \mathfrak{D}}[h(x)]$

The Generalization Properties of DP ("anti overfitting")

Here we explain only why this is true in expectation, i.e., $\underset{\substack{S\sim\mathfrak{D} \\ h\leftarrow\mathcal{A}(S)}}{\mathbb{E}}[h(S)] \approx \underset{\substack{S\sim\mathfrak{D} \\ h\leftarrow\mathcal{A}(S)}}{\mathbb{E}}[h(\mathfrak{D})]$

Theorem [DFHPRR'15, BNSSSU'16]:

- Let $\mathcal{A}: X^n \to 2^X$ be a differentially private algorithm that outputs a predicate $h: X \to \{0,1\}$
- Let $\mathfrak{D}$ be a distribution over X
- Let $S \sim \mathfrak{D}^n$ and let $h \leftarrow \mathcal{A}(S)$
- Then w.h.p. we have $\frac{1}{|S|} \cdot \sum_{x\in S} h(x) \approx \mathbb{E}_{x\sim\mathfrak{D}}[h(x)]$

The Generalization Properties of DP ("anti overfitting")

Here we explain only why this is true in expectation, i.e., $\mathop{\mathbb{E}}_{\substack{S\sim\mathfrak{D} \\ h\leftarrow\mathcal{A}(S)}}[h(S)] \approx \mathop{\mathbb{E}}_{\substack{S\sim\mathfrak{D} \\ h\leftarrow\mathcal{A}(S)}}[h(\mathfrak{D})]$

Consider 2 experiments:

Experiment 1		Experiment 2
• $S = (x_1, \ldots, x_n) \sim \mathfrak{D}$ • $i \in_R \{1, 2, \ldots, n\}$ • $h \leftarrow \mathcal{A}(S)$ • Return $h(x_i)$	$\underset{DP}{\approx}$	• $S = (x_1, \ldots, x_n) \sim \mathfrak{D}$ • $i \in_R \{1, 2, \ldots, n\}$ • $h \leftarrow \mathcal{A}(S \setminus \{x_i\})$ • Return $h(x_i)$

Theorem [DFHPRR'15, BNSSSU'16]:

- Let $\mathcal{A}: X^n \rightarrow 2^X$ be a differentially private algorithm that outputs a predicate $h: X \rightarrow \{0, 1\}$
- Let $\mathfrak{D}$ be a distribution over X
- Let $S \sim \mathfrak{D}^n$ and let $h \leftarrow \mathcal{A}(S)$
- Then w.h.p. we have $\frac{1}{|S|} \cdot \sum_{x\in S} h(x) \approx \mathbb{E}_{x\sim\mathfrak{D}}[h(x)]$

The Generalization Properties of DP ("anti overfitting")

Here we explain only why this is true in expectation, i.e., $\underset{\substack{S\sim\mathfrak{D} \\ h\leftarrow\mathcal{A}(S)}}{\mathbb{E}}[h(S)] \approx \underset{\substack{S\sim\mathfrak{D} \\ h\leftarrow\mathcal{A}(S)}}{\mathbb{E}}[h(\mathfrak{D})]$

Consider 2 experiments:

Experiment 1		Experiment 2
• $S = (x_1, \dots, x_n) \sim \mathfrak{D}$ • $i \in_R \{1, 2, \dots, n\}$ • $h \leftarrow \mathcal{A}(S)$ • **Return** $h(x_i)$	$\underset{DP}{\approx}$	• $S = (x_1, \dots, x_n) \sim \mathfrak{D}$ • $i \in_R \{1, 2, \dots, n\}$ • $h \leftarrow \mathcal{A}(S \setminus \{x_i\})$ • **Return** $h(x_i)$
h on a random element of S		h on a random element of $\mathfrak{D}$

Theorem [DFHPRR'15, BNSSSU'16]:

- Let $\mathcal{A}: X^n \to 2^X$ be a differentially private algorithm that outputs a predicate $h: X \to \{0, 1\}$
- Let $\mathfrak{D}$ be a distribution over X
- Let $S \sim \mathfrak{D}^n$ and let $h \leftarrow \mathcal{A}(S)$
- Then w.h.p. we have $\frac{1}{|S|} \cdot \sum_{x\in S} h(x) \approx \mathbb{E}_{x\sim\mathfrak{D}}[h(x)]$

Differential privacy as a tool

Today's Outline

1. DP is the enemy of overfitting
2. Application to answering adaptive queries
3. Application to adaptive streaming

Recall: The Statistical Queries Model

- Let $\mathfrak{D}$ be an unknown distribution over a domain $\boldsymbol{X}$
- Consider a data analyst who wants to learn properties of $\mathfrak{D}$
- The analyst interacts with $\mathfrak{D}$ via ***statistical queries***:

  In each step, the analyst specifies a predicate $\boldsymbol{h}: \boldsymbol{X} \to \{\mathbf{0}, \mathbf{1}\}$ and obtains an estimate for $\mathbb{E}_{x\sim\mathfrak{D}}[\boldsymbol{h}(\boldsymbol{x})]$

Unknown dist. $\mathfrak{D}$
over domain $\boldsymbol{X}$

$$\xleftarrow{\quad \boldsymbol{h_1} \quad}$$

$$\xrightarrow{\quad \boldsymbol{a_1} \approx \mathbb{E}_{x\sim\mathfrak{D}}[\boldsymbol{h_1}(\boldsymbol{x})] \quad}$$

$$\xleftarrow{\quad \boldsymbol{h_2} \quad}$$

$$\xrightarrow{\quad \boldsymbol{a_2} \approx \mathbb{E}_{x\sim\mathfrak{D}}[\boldsymbol{h_2}(\boldsymbol{x})] \quad}$$

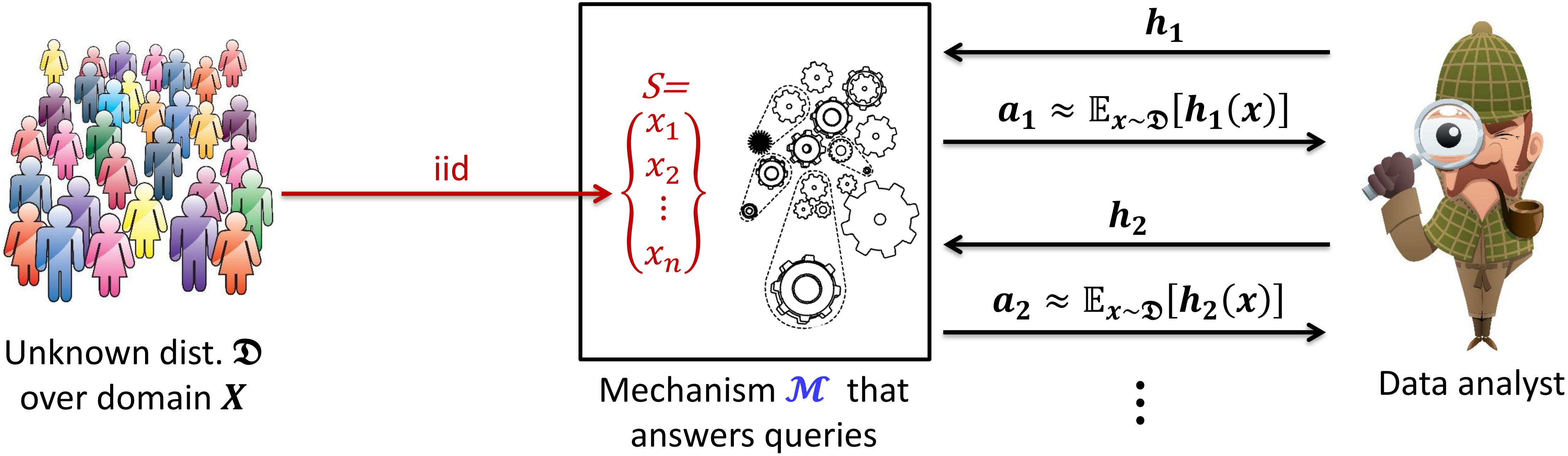
$$\vdots$$

Data analyst

Recall: The Statistical Queries Model

- Let $\mathfrak{D}$ be an unknown distribution over a domain $\boldsymbol{X}$
- Consider a data analyst who wants to learn properties of $\mathfrak{D}$
- The analyst interacts with $\mathfrak{D}$ via ***statistical queries***:

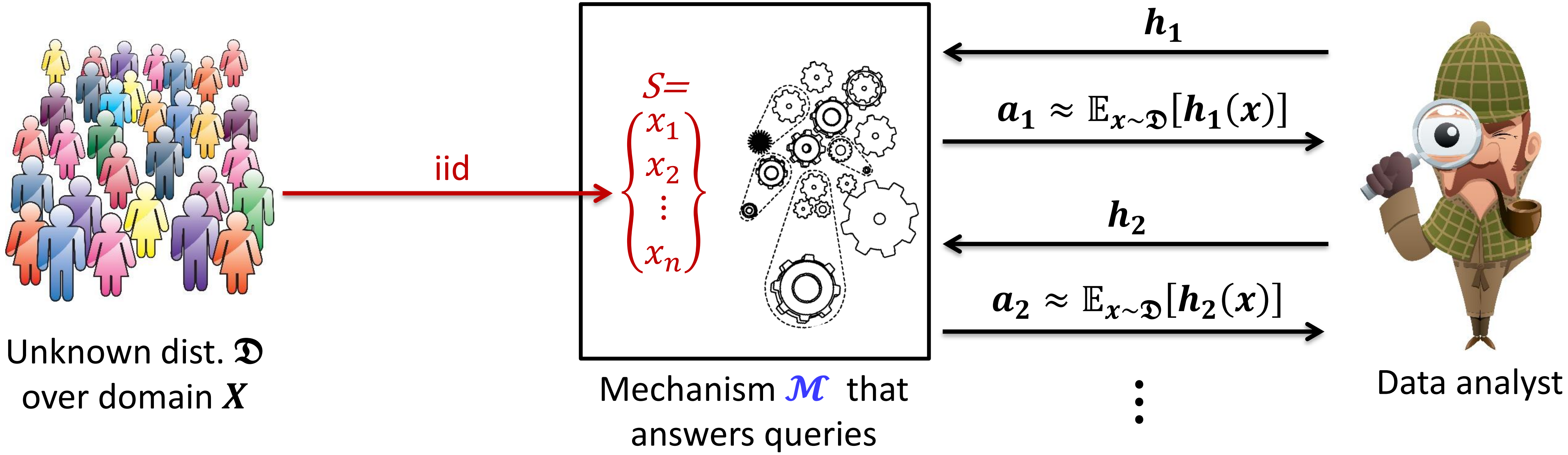
  In each step, the analyst specifies a predicate $\boldsymbol{h}: \boldsymbol{X} \to \{\mathbf{0}, \mathbf{1}\}$ and obtains an estimate for $\mathbb{E}_{x \sim \mathfrak{D}}[\boldsymbol{h}(\boldsymbol{x})]$



Recall: The Statistical Queries Model

- Let $\mathfrak{D}$ be an unknown distribution over a domain $\boldsymbol{X}$
- Consider a data analyst who wants to learn properties of $\mathfrak{D}$
- The analyst interacts with $\mathfrak{D}$ via ***statistical queries***:

  In each step, the analyst specifies a predicate $\boldsymbol{h}: \boldsymbol{X} \to \{\boldsymbol{0}, \boldsymbol{1}\}$ and obtains an estimate for $\mathbb{E}_{x\sim\mathfrak{D}}[\boldsymbol{h}(\boldsymbol{x})]$



We want: w.h.p. $\forall j, \quad \left|\boldsymbol{a_j} - \mathbb{E}_{x\sim\mathfrak{D}}[\boldsymbol{h_j}(\boldsymbol{x})]\right| \le \boldsymbol{\alpha}$

What is the number of samples $\boldsymbol{n}$ that $\mathcal{M}$ needs to ensure this as a function of $\boldsymbol{\alpha}$ and the number of queries $\boldsymbol{k}$?

Recall: The Statistical Queries Model

- Let $\mathfrak{D}$ be
- Consider
- The analy

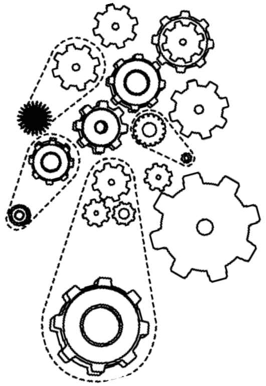
In eac … $\to \{0,1\}$ and obtains an estimate for $\mathbb{E}_{x\sim\mathfrak{D}}[h(x)]$

- The challenge is that the analyst can choose its queries adaptively
- We want to provide accuracy w.r.t. $\mathfrak{D}$
- If we are not careful, we could quickly overfit to S

Unknown dist. $\mathfrak{D}$ over domain X

iid

$S = \{x_1, x_2, \vdots, x_n\}$



Mechanism $\mathcal{M}$ that answers queries

h_1

$a_1 \approx \mathbb{E}_{x\sim\mathfrak{D}}[h_1(x)]$

h_2

$a_2 \approx \mathbb{E}_{x\sim\mathfrak{D}}[h_2(x)]$

$\vdots$

Data analyst

We want: w.h.p. $\forall j,\ \left|a_j - \mathbb{E}_{x\sim\mathfrak{D}}[h_j(x)]\right| \le \alpha$

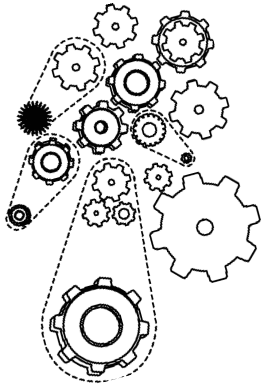
What is the number of samples n that $\mathcal{M}$ needs to ensure this as a function of α and the number of queries k?

Step back: Non adaptive game

Unknown dist. $\mathfrak{D}$ over domain $\boldsymbol{X}$

iid

$S = \{x_1, x_2, \vdots, x_n\}$



Mechanism $\mathcal{M}$ that answers queries

$\boldsymbol{h_1, h_2, \dots, h_k}$

$\boldsymbol{a_1, a_2, \dots, a_k}$

Data analyst

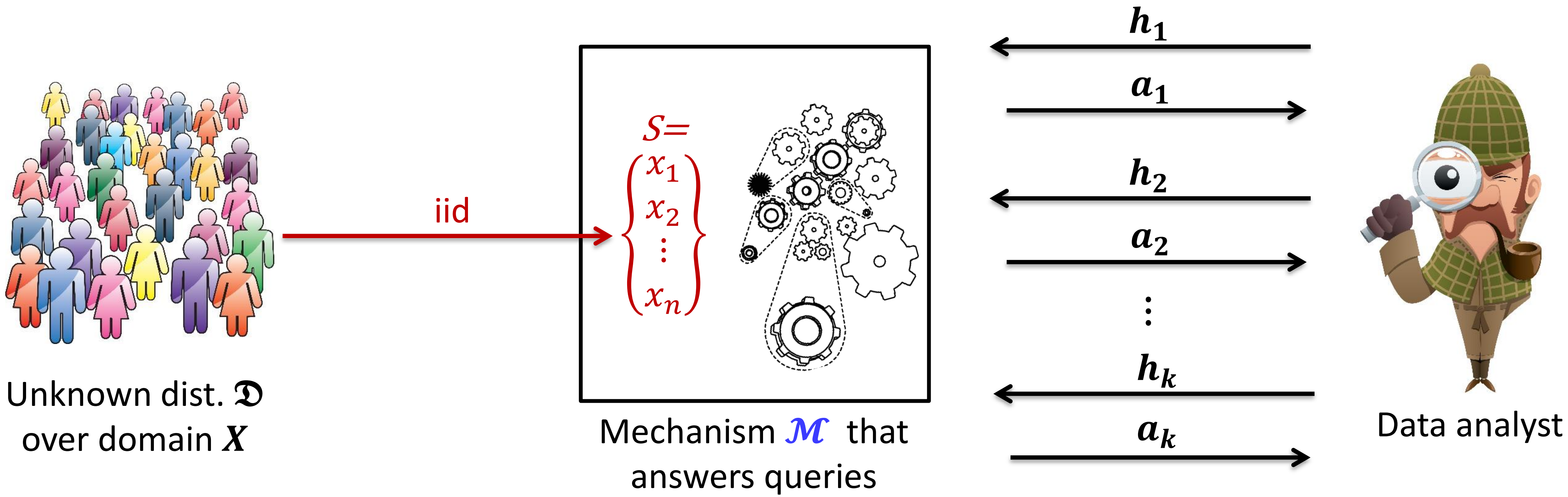
Natural solution: Answer every $\boldsymbol{h_i}$ with its empirical avg $\boldsymbol{a_i = h_i(S)}$

Hoeffding: w.h.p., $\boldsymbol{h_i(S) \approx h_i(\mathfrak{D})}$ for all $\boldsymbol{i}$, provided $\boldsymbol{n \gtrsim \frac{1}{\alpha^2} \log k}$

Can answer exponential number of non-adaptive queries!

Notation: $\boldsymbol{h(\mathfrak{D}) = \mathbb{E}_{x\sim\mathfrak{D}}[h(x)]}$, $\boldsymbol{h(S) = \frac{1}{n}\sum_{i=1}^{n} h(x_i)}$

Recall the adaptive model



Can we answer with empirical average, i.e., answer every $\boldsymbol{h}$ with $\boldsymbol{h(S) = \frac{1}{n}\sum_{i=1}^{n} h(x_i)}$?

Negative result for adaptive case:
Answering every h_i with $h_i(S)$ fails after 1 query

- Domain $X = \{1, 2, \ldots, 2n\}$
- Database S with n iid uniform samples from X

Goal: After 1 query, find h s.t. $h(S) \gg h(\mathfrak{D})$

Negative result for adaptive case: Answering every h_i with $h_i(S)$ fails after 1 query

- Domain $X = \{1, 2, \ldots, 2n\}$
- Database S with n iid uniform samples from X

Goal: After 1 query, find h s.t. $h(S) \gg h(\mathfrak{D})$

Step 1: Recover the database

- Define $h_1(x) = 0.\underbrace{000000\ldots0}_{\#\text{zeroes} = x \cdot \log n}1$, and query $h_1(S) =?$
- **Observe: low-order bits of $h_1(S)$ reveal all entries of S**

Negative result for adaptive case:
Answering every h_i with $h_i(S)$ fails after 1 query

- Domain $X = \{1, 2, \ldots, 2n\}$
- Database S with n iid uniform samples from X

Goal: After 1 query, find h s.t. $h(S) \gg h(\mathfrak{D})$

Step 1: Recover the database

- Define $h_1(x) = 0.\underbrace{000000 \ldots 0}_{\#\text{zeroes} = x \cdot \log n}1$, and query $h_1(S) = ?$
- **Observe: low-order bits of $h_1(S)$ reveal all entries of S**

If $S = (1, 3, 4, 4, 4)$ then $\sum_{x \in S} h_1(x)$ is

$$
\begin{array}{r}
0.0001 \\
+0.0000000001 \\
+0.0000000000001 \\
+0.0000000000001 \\
+0.0000000000001 \\
\hline
0.0001000001011
\end{array}
$$

Negative result for adaptive case:
Answering every h_i with $h_i(S)$ fails after 1 query

- Domain $X = \{1, 2, \ldots, 2n\}$
- Database S with n iid uniform samples from X

Goal: After 1 query, find h s.t. $h(S) \gg h(\mathfrak{D})$

Step 1: Recover the database

- Define $h_1(x) = 0.\underbrace{000000\ldots0}_{\#\text{zeroes} = x \cdot \log n}1$, and query $h_1(S) = ?$
- **Observe: low-order bits of $h_1(S)$ reveal all entries of S**

Step 2: Overfitting

- Define $h_2(x) = \begin{cases} 1 & , \quad x \in S \\ 0 & , \quad x \notin S \end{cases}$
- **Observe: $h_2(S) = 1$ but $h_2(\mathfrak{D}) \leq \frac{1}{2}$**

Negative result for adaptive case:
Answering every h_i with $h_i(S)$ fails after 1 query

- Domain $X = \{1, 2, \dots, 2n\}$
- Database S with n iid uniform samples from X

Goal: After 1 query, find h s.t. $h(S) \gg h(\mathfrak{D})$

Step 1: Recover the database

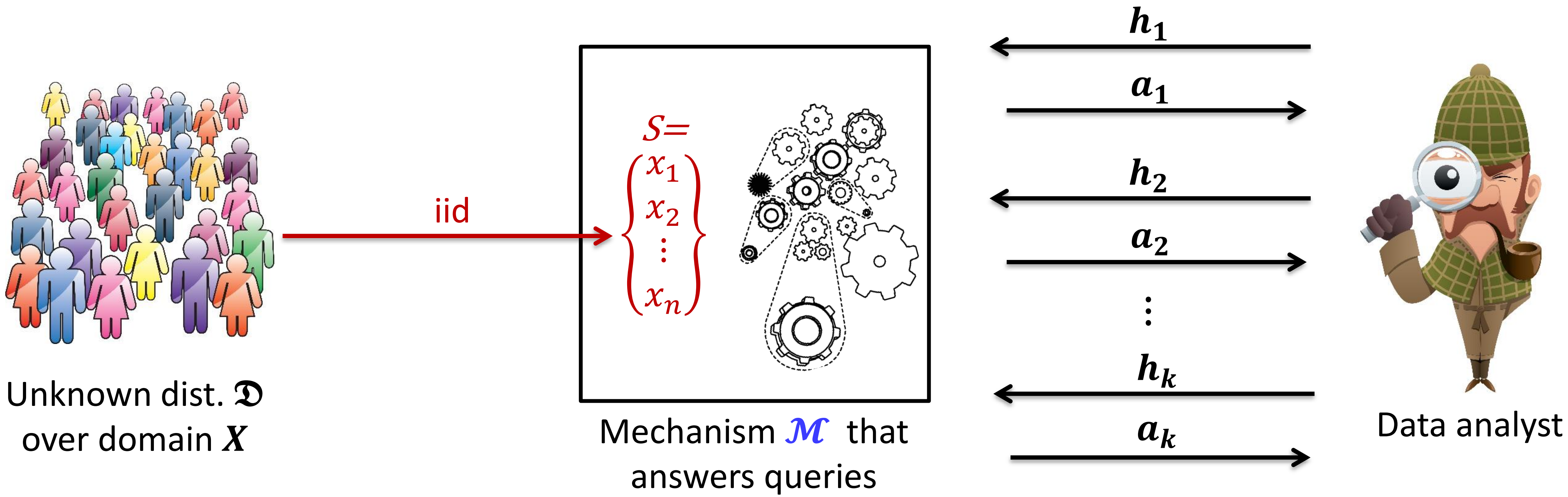
- Define $h_1(x) = 0.\underbrace{000000 \dots 0}_{\#\text{zeroes} = x \cdot \log n}1$, and query $h_1(S) = ?$
- **Observe: low-order bits of $h_1(S)$ reveal all entries of S**

Step 2: Overfitting

- Define $h_2(x) = \begin{cases} 1 & , \quad x \in S \\ 0 & , \quad x \notin S \end{cases}$
- **Observe:** $h_2(S) = 1$ **but** $h_2(\mathfrak{D}) \leq \frac{1}{2}$

Takeaway: Learning info about the data set allows the analyst to overfit

Recall the adaptive model



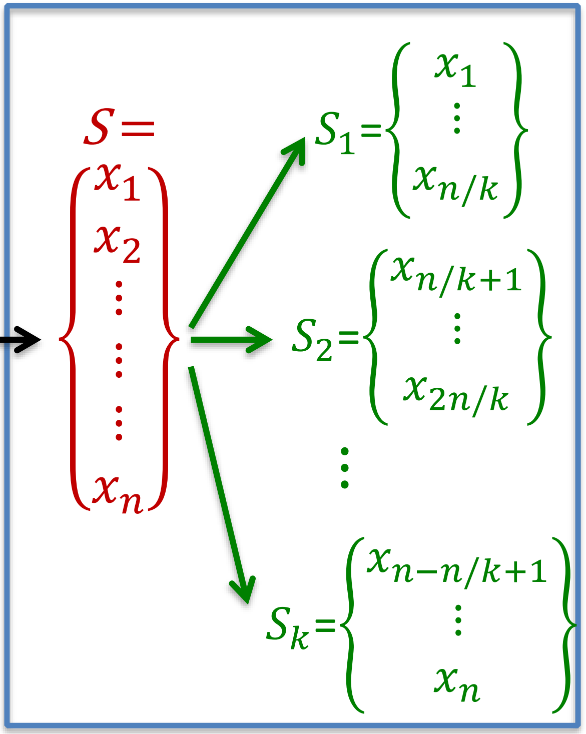
So far: reusing the data and answering with empirical average does not work

Natural approach #2: Data splitting

- Divide the data set into $\boldsymbol{k}$ chunks of size $\boldsymbol{n/k}$ each
- Answer $\boldsymbol{h_i}$ using its empirical mean on chunk $\boldsymbol{i}$

Unknown dist. $\mathfrak{D}$
over domain $\boldsymbol{X}$

$\xrightarrow{iid}$

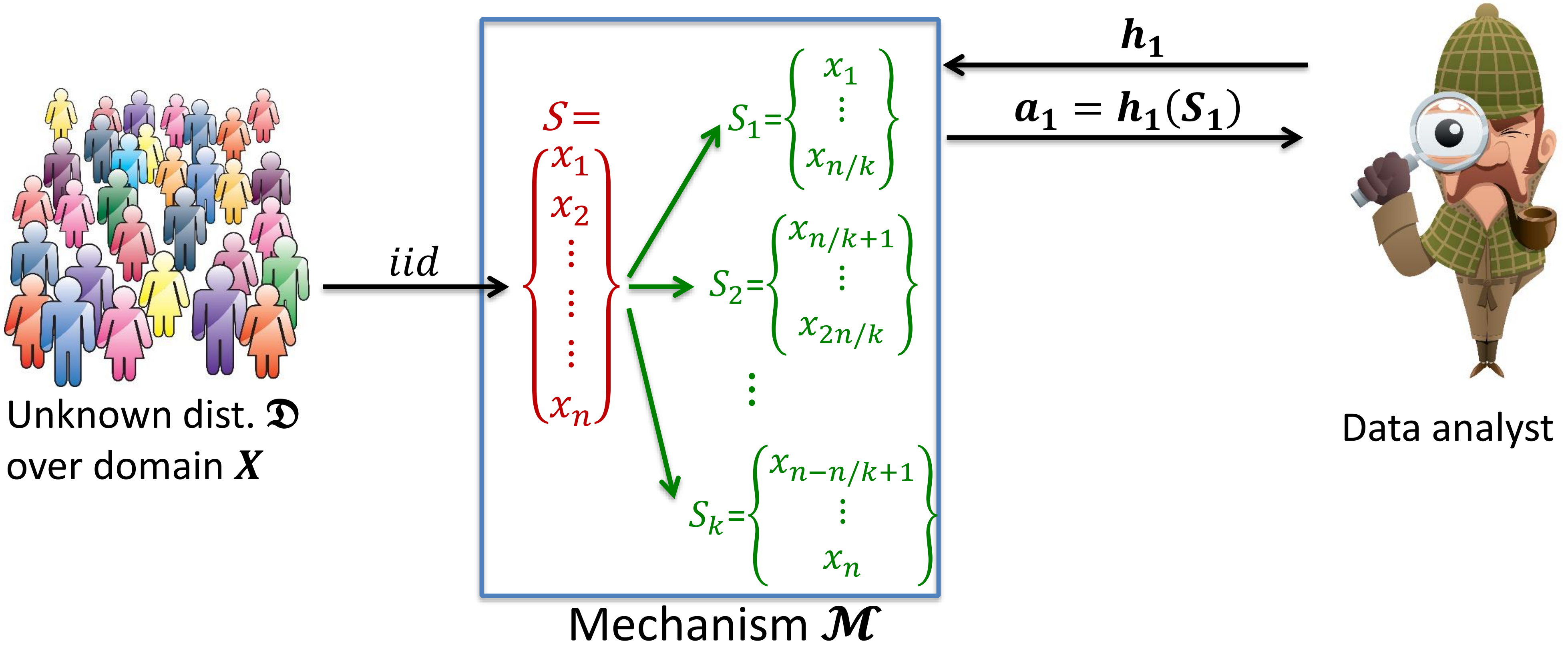


Mechanism $\mathcal{M}$

Data analyst

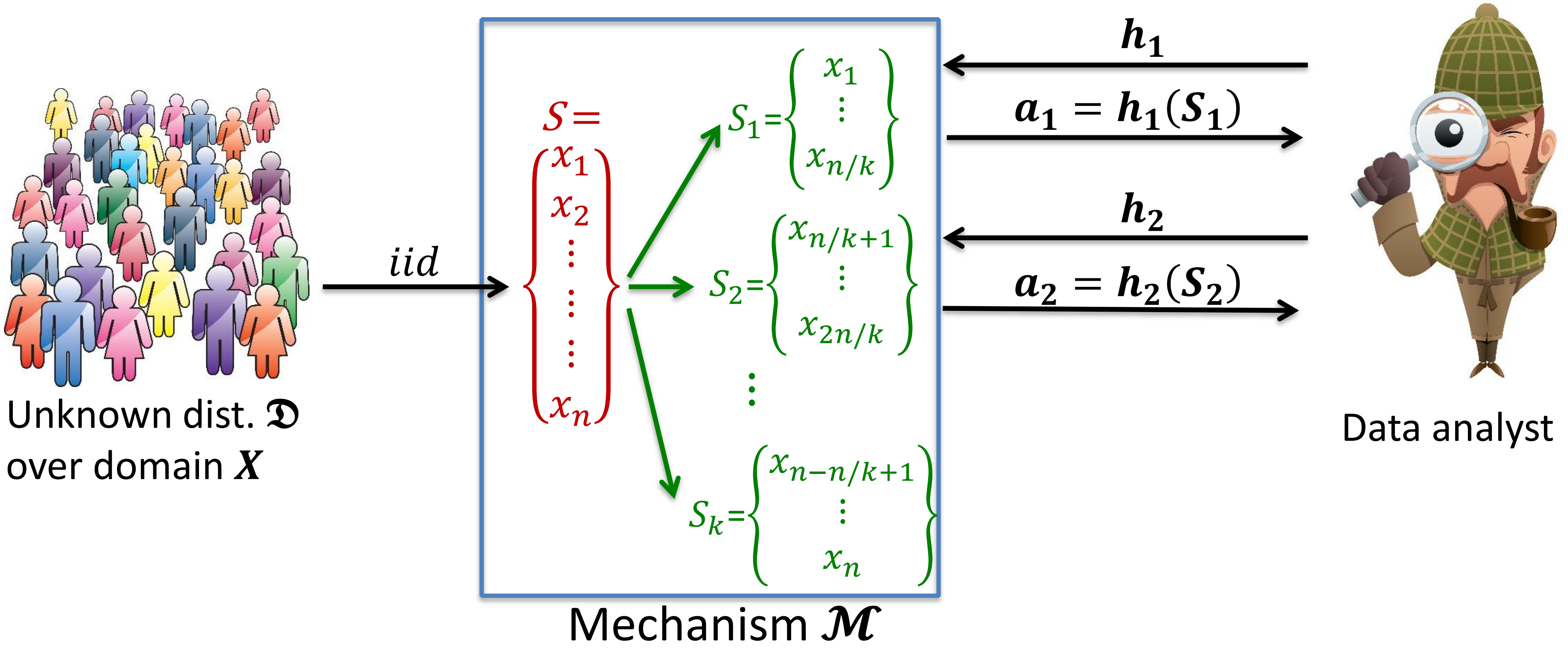
Natural approach #2: Data splitting

- Divide the data set into $\boldsymbol{k}$ chunks of size $\boldsymbol{n/k}$ each
- Answer $\boldsymbol{h_i}$ using its empirical mean on chunk $\boldsymbol{i}$



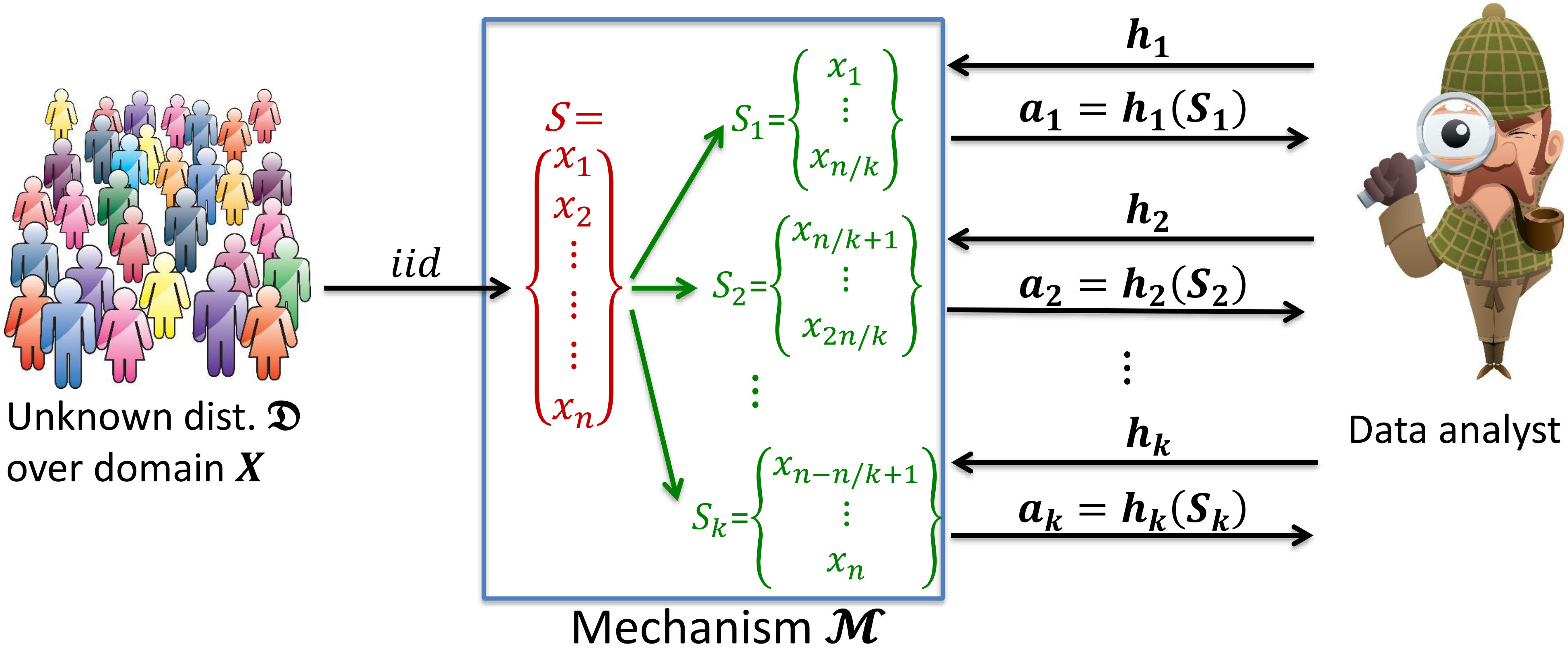
Natural approach #2: Data splitting

- Divide the data set into $\boldsymbol{k}$ chunks of size $\boldsymbol{n/k}$ each
- Answer $\boldsymbol{h_i}$ using its empirical mean on chunk $\boldsymbol{i}$



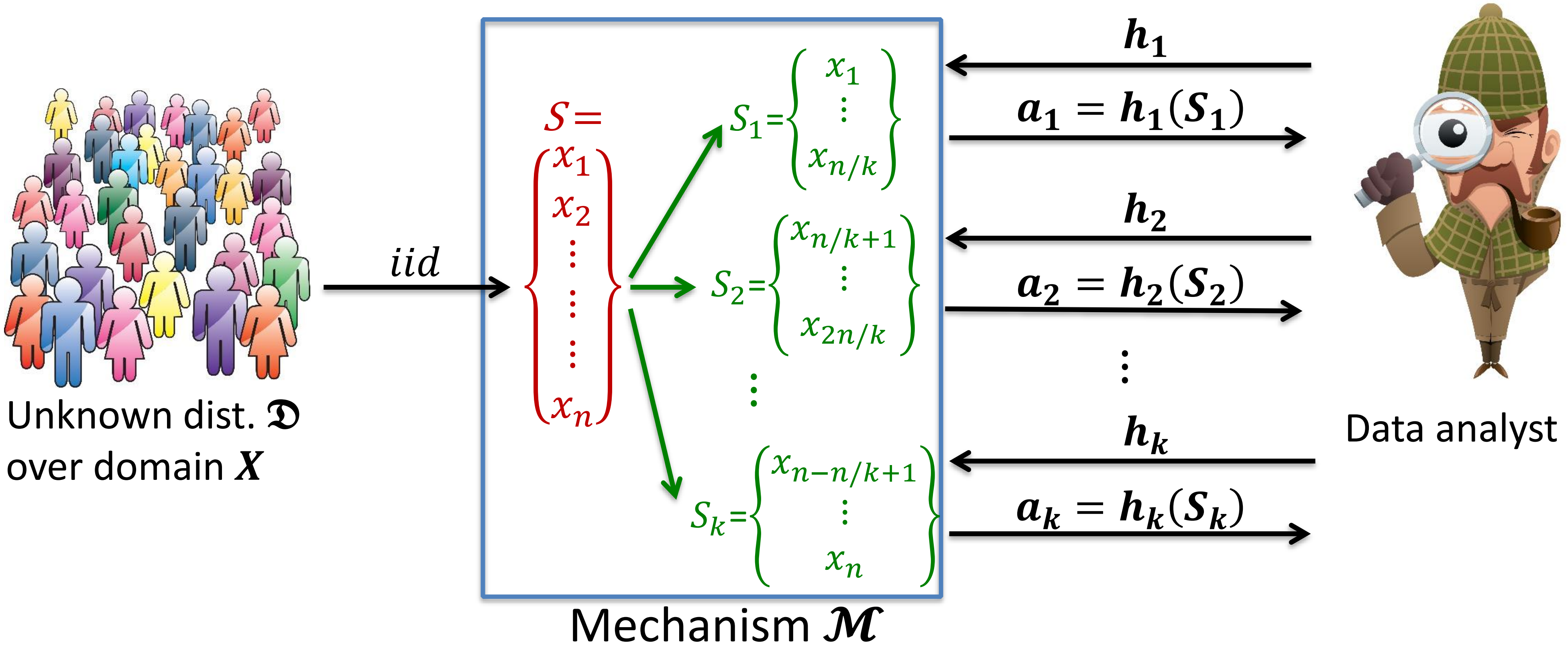
Natural approach #2: Data splitting

- Divide the data set into $\boldsymbol{k}$ chunks of size $\boldsymbol{n/k}$ each
- Answer $\boldsymbol{h_i}$ using its empirical mean on chunk $\boldsymbol{i}$



Natural approach #2: Data splitting

- Divide the data set into $\boldsymbol{k}$ chunks of size $\boldsymbol{n/k}$ each
- Answer $\boldsymbol{h_i}$ using its empirical mean on chunk $\boldsymbol{i}$

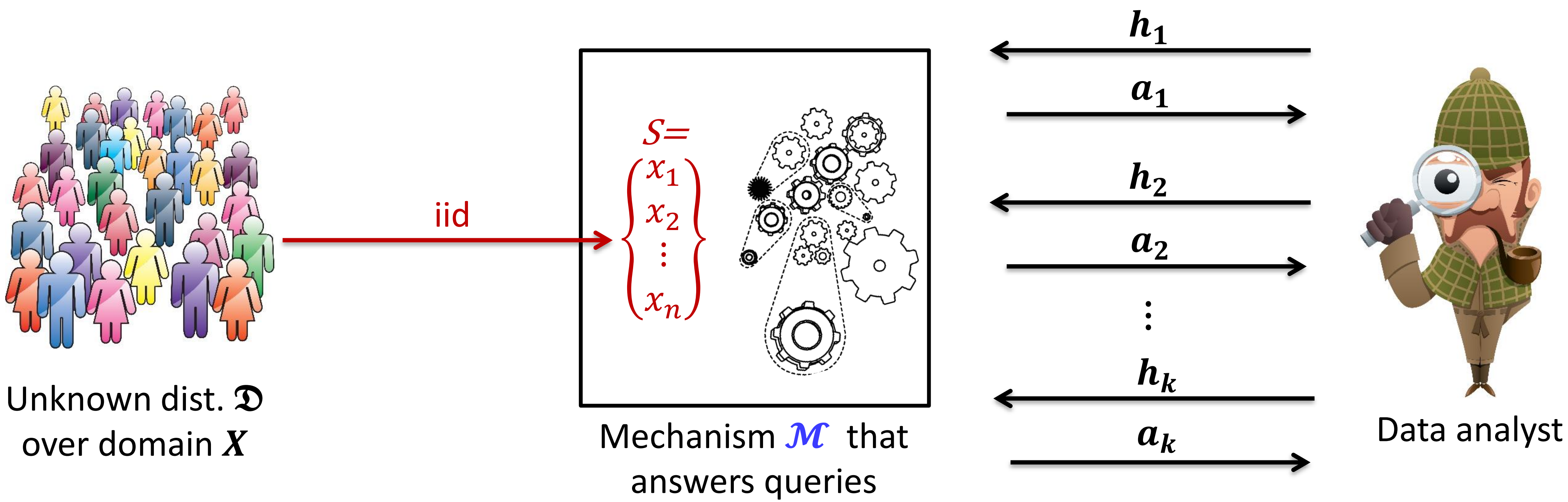


Upside: Can ignore adaptivity and use Hoeffding/Chernoff

Downside: With this approach we need $\boldsymbol{n > k/\alpha^2}$

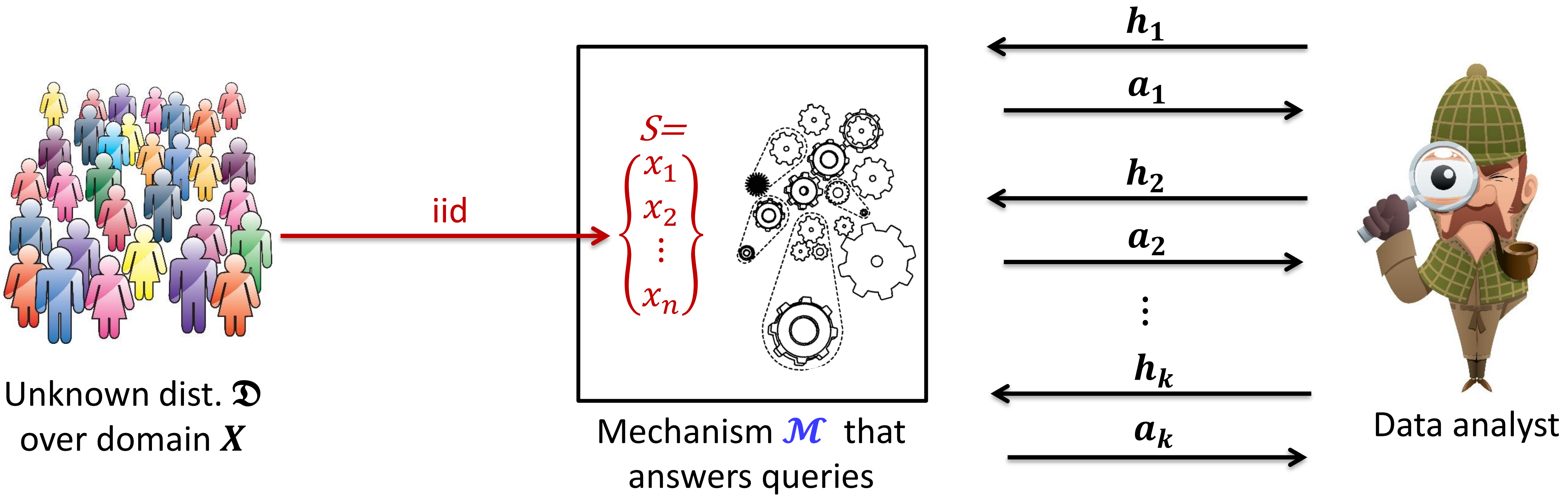
But, we can do better!

DP to the rescue



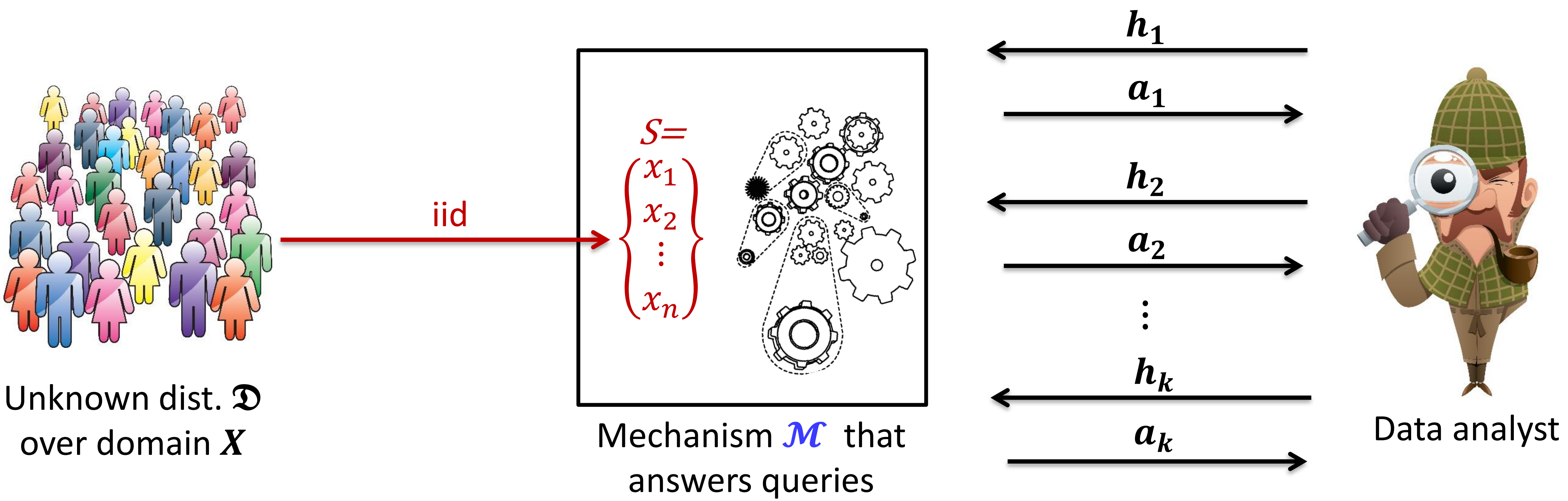
1) Assume $\mathcal{M}$ is (ϵ, δ)-DP mechanism that approximates the empirical average of $\boldsymbol{k}$ adaptively chosen queries

DP to the rescue



1) Assume $\mathcal{M}$ is (ϵ, δ)-DP mechanism that approximates the <u>empirical</u> <u>average</u> of $\boldsymbol{k}$ adaptively chosen queries
2) Then the answers $a_1, \ldots, a_k$ are the result of a private computation on S
3) By post-processing, <u>***the queries***</u> $\boldsymbol{h_1}, \ldots, \boldsymbol{h_k}$ are also the result of a private computation on $\boldsymbol{S}$

DP to the rescue



1) Assume $\mathcal{M}$ is (ϵ, δ)-DP mechanism that approximates the empirical average of $\boldsymbol{k}$ adaptively chosen queries
2) Then the answers $a_1, \ldots, a_k$ are the result of a private computation on S
3) By post-processing, ***the queries*** $\boldsymbol{h_1}, \ldots, \boldsymbol{h_k}$ are also the result of a private computation on $\boldsymbol{S}$
4) But then for every $\boldsymbol{i}$ we have $\boldsymbol{a_i} \underbrace{\approx}_{\textbf{by (1)}} \boldsymbol{h_i(S)} \underbrace{\approx}_{\textbf{DP generalization}} \boldsymbol{h_i(\mathfrak{D})}$

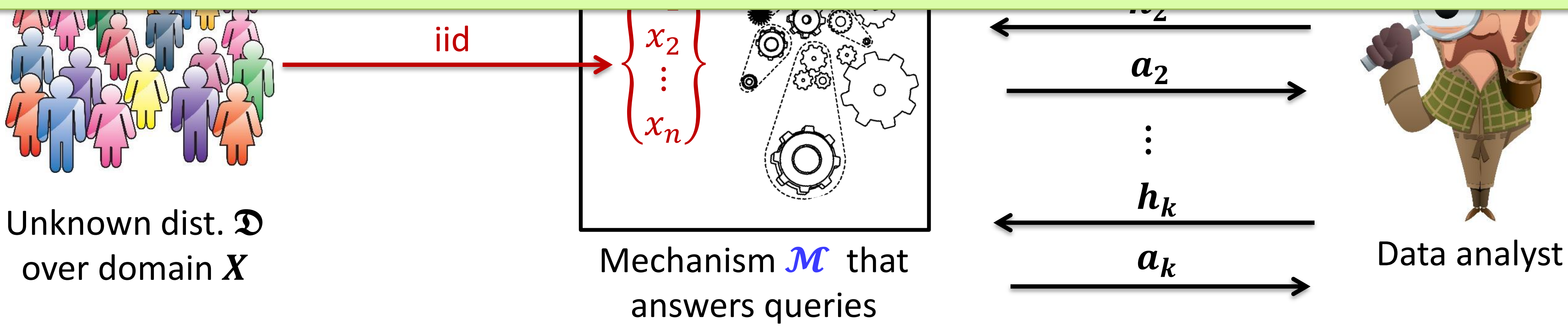
Theorem [DMNS'06]

There is an efficient private alg. estimating the empirical average of $\approx \boldsymbol{n}^2$ adaptive queries using a database of size $\boldsymbol{n}$

$\Longrightarrow$

Theorem [DFHPRP'15, BNSSSU'16]

There is an efficient alg. answering $\approx \boldsymbol{n}^2$ adaptive queries on the distribution using $\boldsymbol{n}$ iid samples



1) Assume $\mathcal{M}$ is (ϵ, δ)-DP mechanism that approximates the empirical average of $\boldsymbol{k}$ adaptively chosen queries
2) Then the answers $a_1, \dots, a_k$ are the result of a private computation on S
3) By post-processing, ***the queries*** $\boldsymbol{h}_1, \dots, \boldsymbol{h}_k$ are also the result of a private computation on $\boldsymbol{S}$
4) But then for every $\boldsymbol{i}$ we have $\boldsymbol{a}_i \underbrace{\approx}_{\text{by (1)}} \boldsymbol{h}_i(\boldsymbol{S}) \underbrace{\approx}_{\text{DP generalization}} \boldsymbol{h}_i(\mathfrak{D})$

Differential privacy as a tool

Today's Outline

1. DP is the enemy of overfitting
2. Application to answering adaptive queries
3. Application to adaptive streaming

Classical vs. Adaptive streaming

- Randomized algorithms are often analyzed under the assumption that their internal randomness is independent of their inputs
- This is a reasonable assumption for offline algorithms, which get all their inputs at once, process it, and spit out the results
- **However, in interactive settings, this assumption is not always reasonable: future inputs may depend on previous outputs, and hence, depend on the internal randomness of the algorithm**

Takeaway: We want to design algorithms providing provable guarantees even for adaptive inputs

Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

Adaptive Streaming

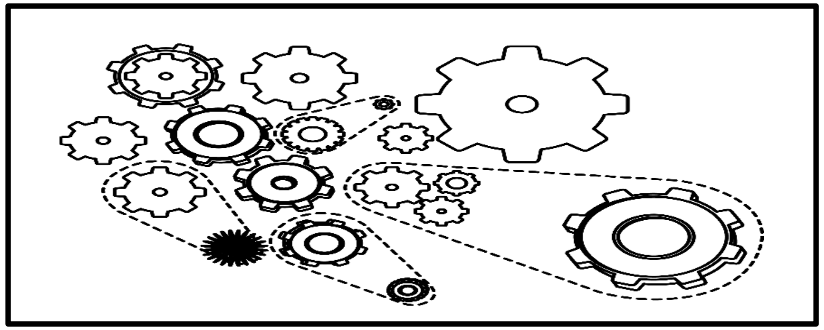
[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

StreamingAlgorithm



Adaptive Streaming

[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

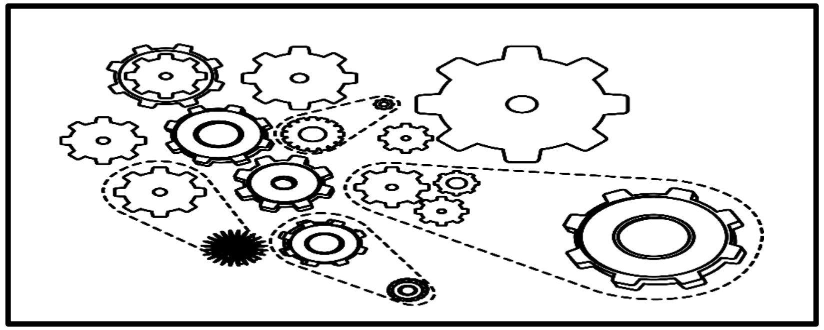
Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

$(u_1, u_2, \ldots, u_m)$ = fixed stream (unknown to the algorithm)

StreamingAlgorithm



Adaptive Streaming

[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

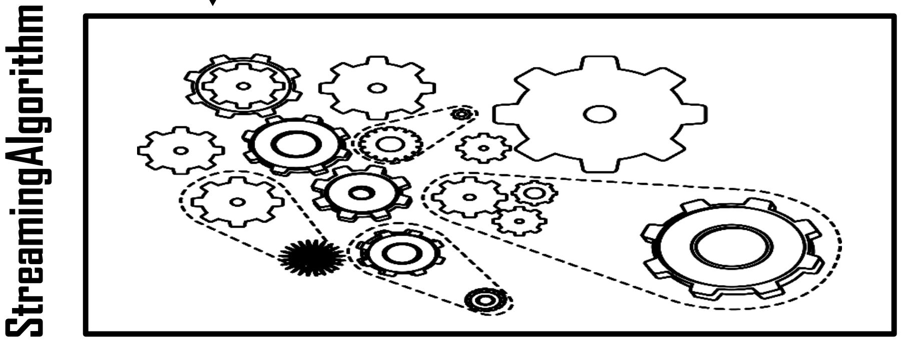
Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

$(u_1, u_2, \ldots, u_m)$ = fixed stream (unknown to the algorithm)

u_1



Adaptive Streaming

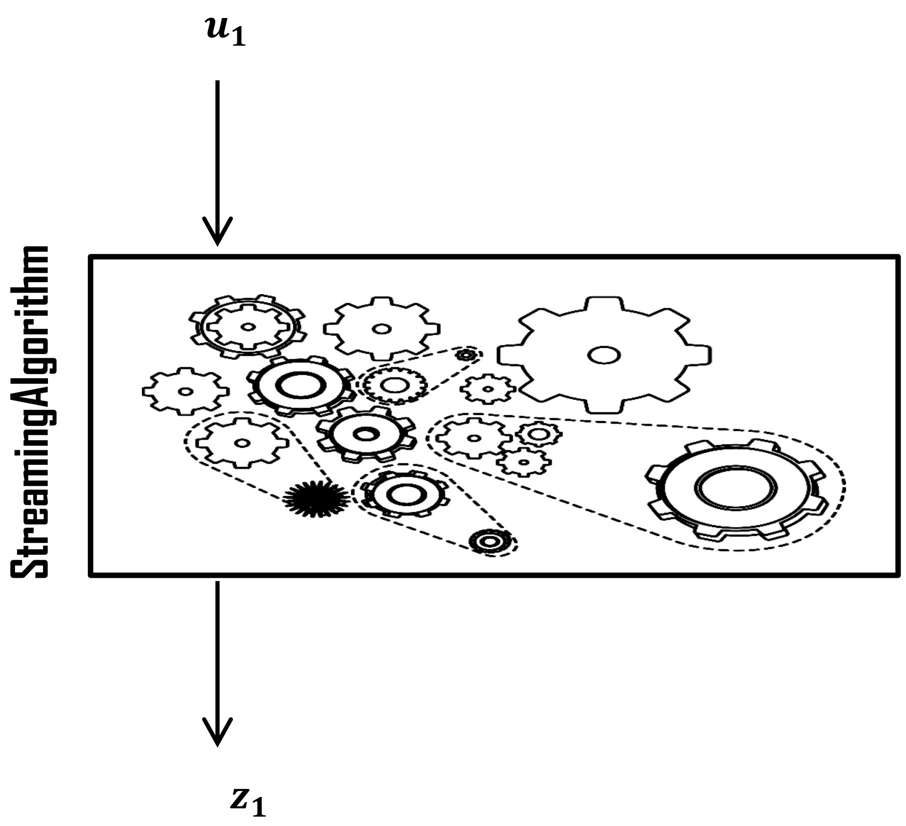
[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

$(u_1, u_2, \ldots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

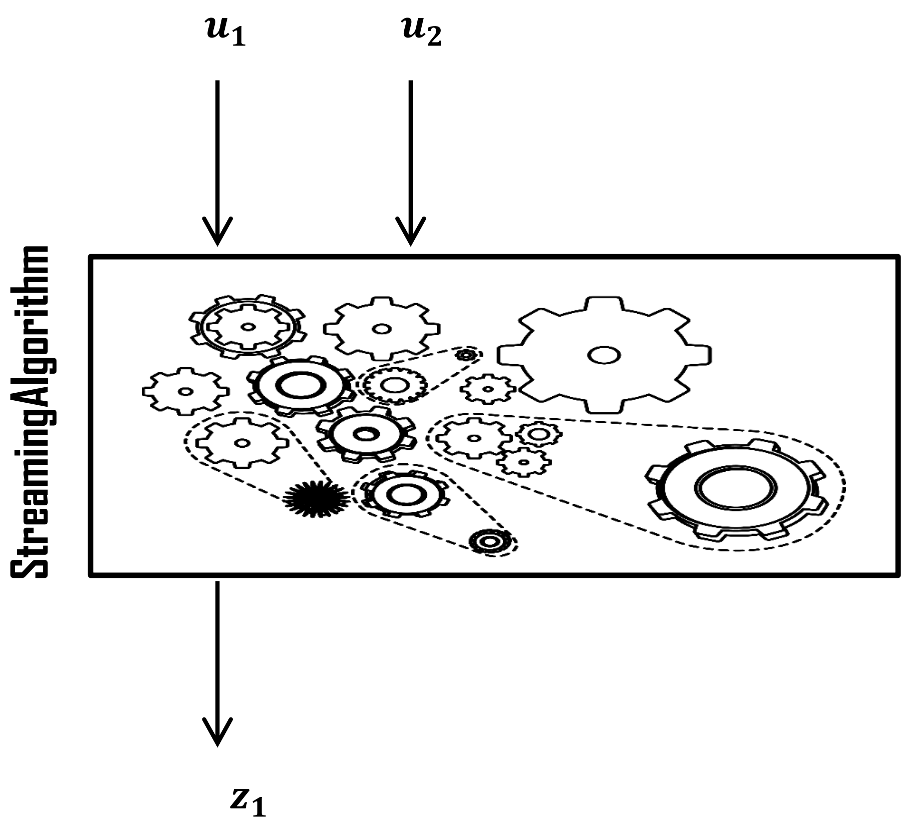
[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

$(u_1, u_2, \ldots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

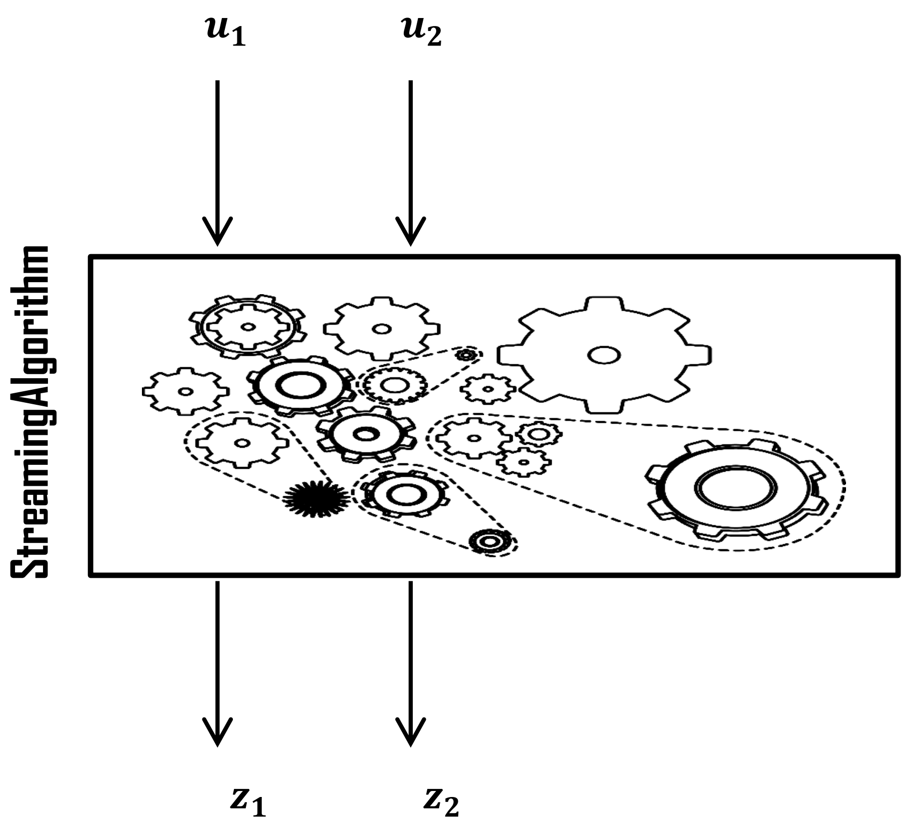
[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

$(u_1, u_2, \dots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

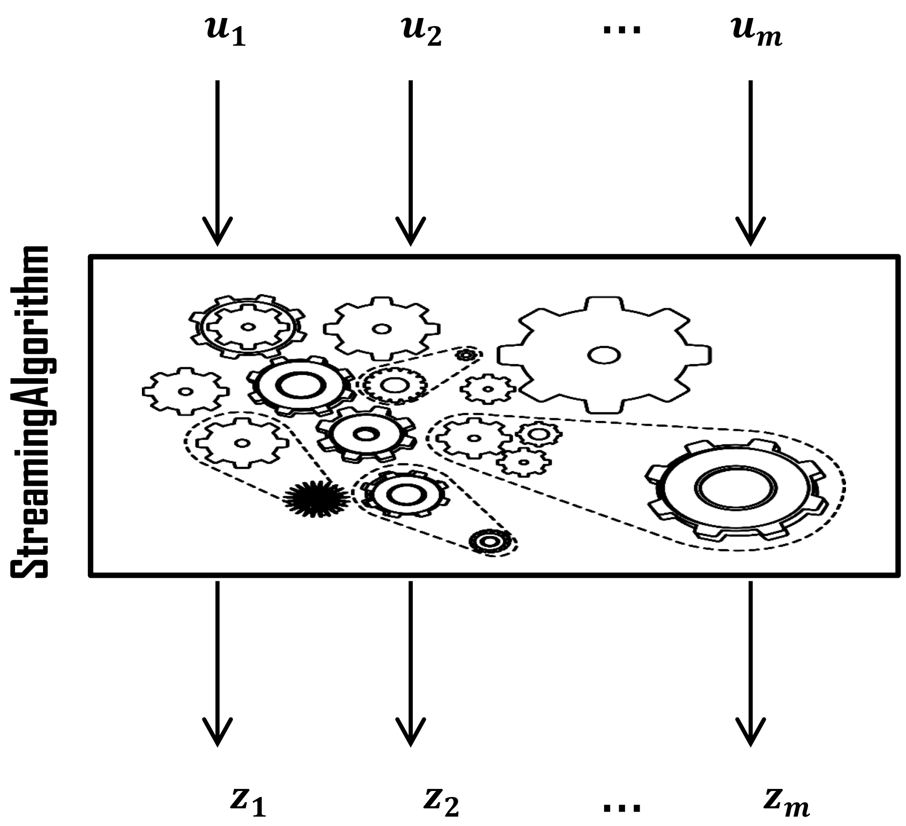
[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

$(u_1, u_2, \dots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

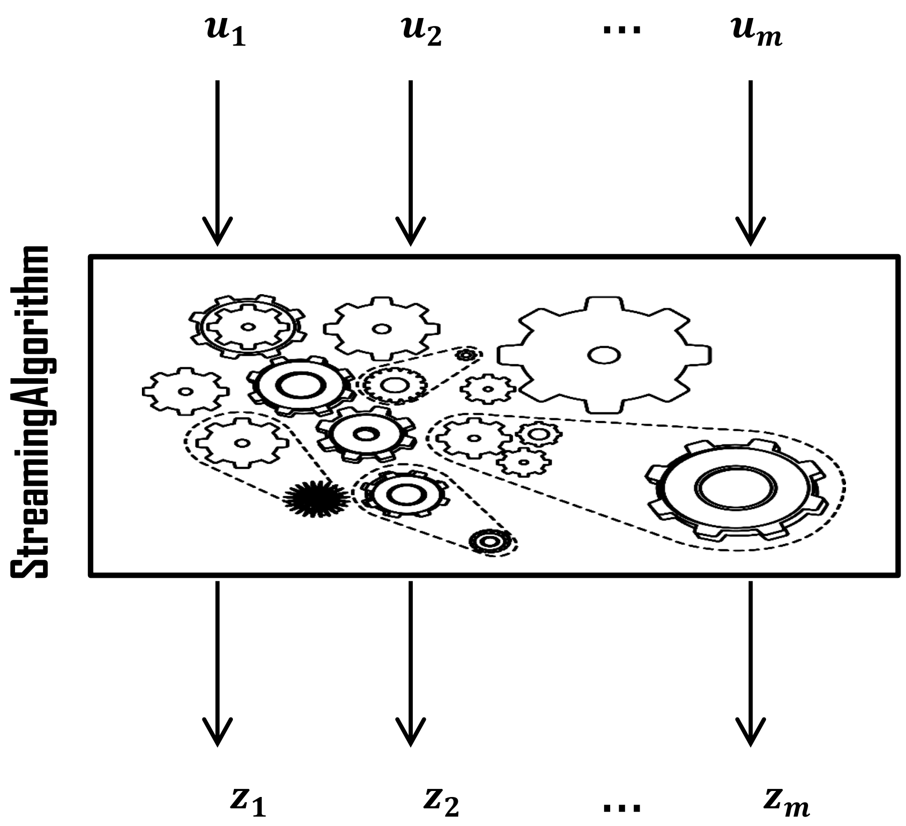
[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

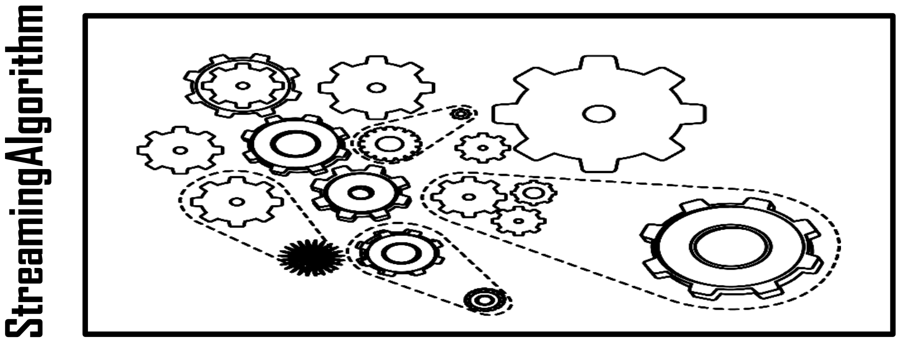
[Alon, Matias, Szegedy '96]

$(u_1, u_2, \dots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

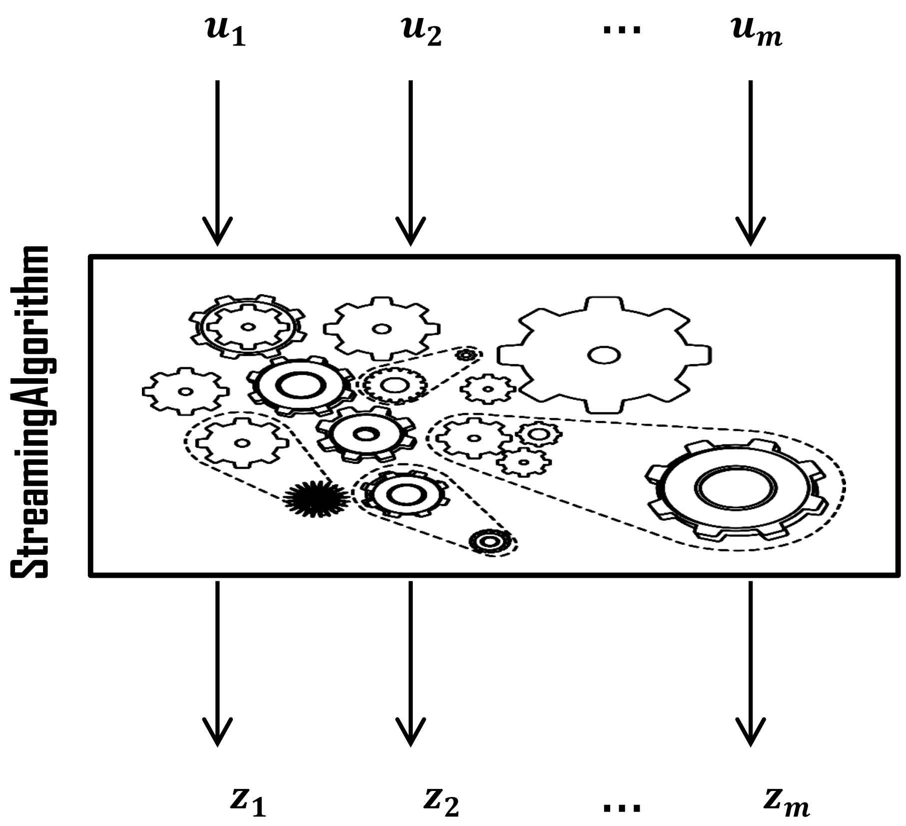


Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

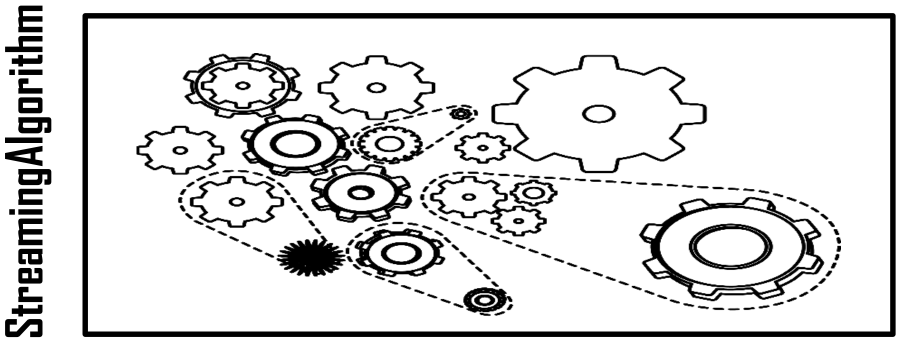
$(u_1, u_2, \dots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Adversary chooses u_i based on previous answers

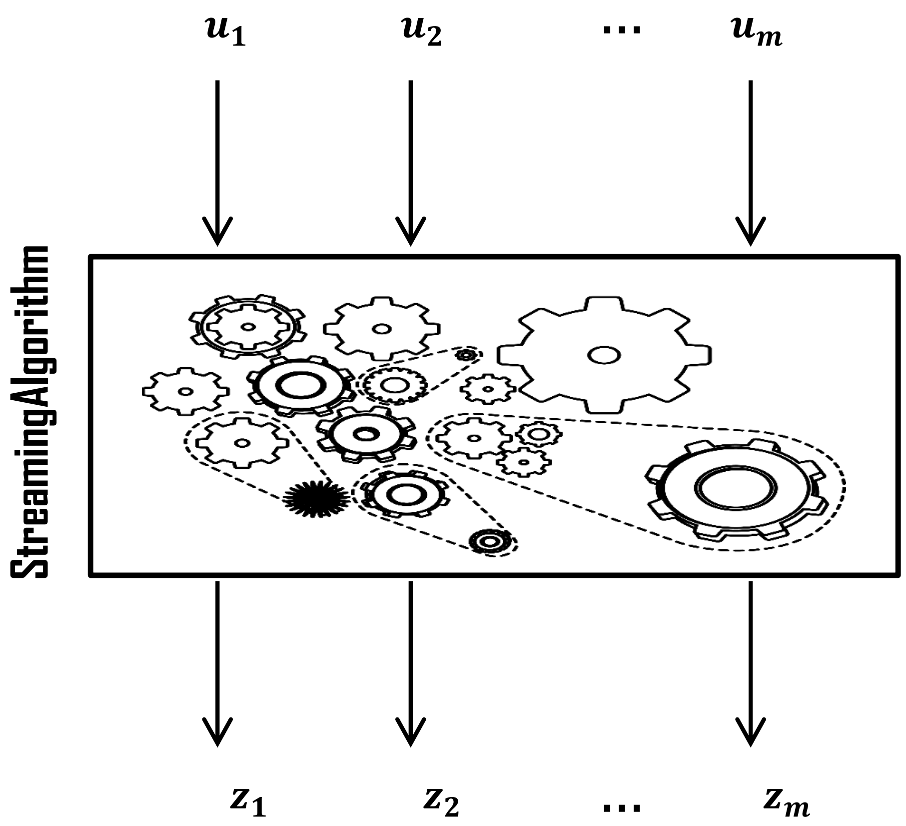


Consider a streaming algorithm that continuously estimates the value of the desired function (The goal is to minimize space requirements)

Oblivious Streaming

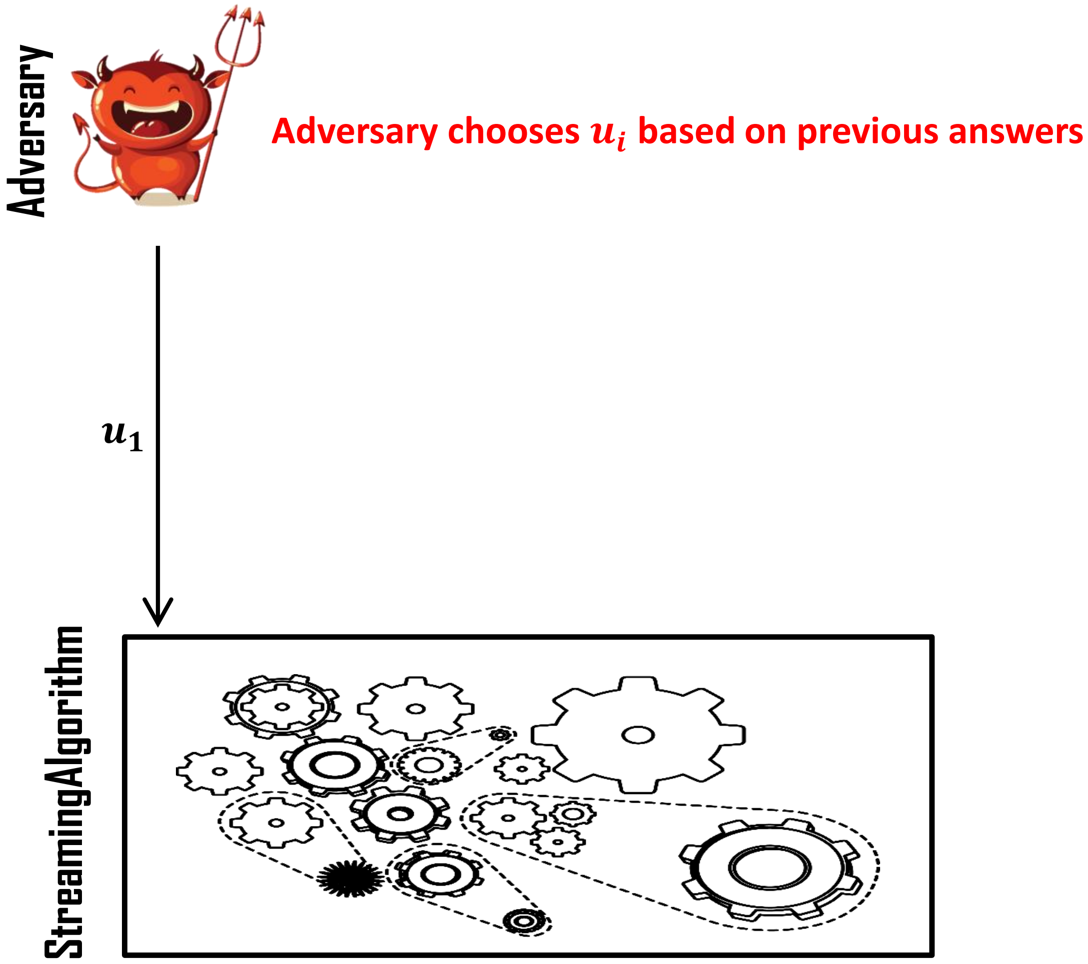
[Alon, Matias, Szegedy '96]

$(u_1, u_2, \dots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

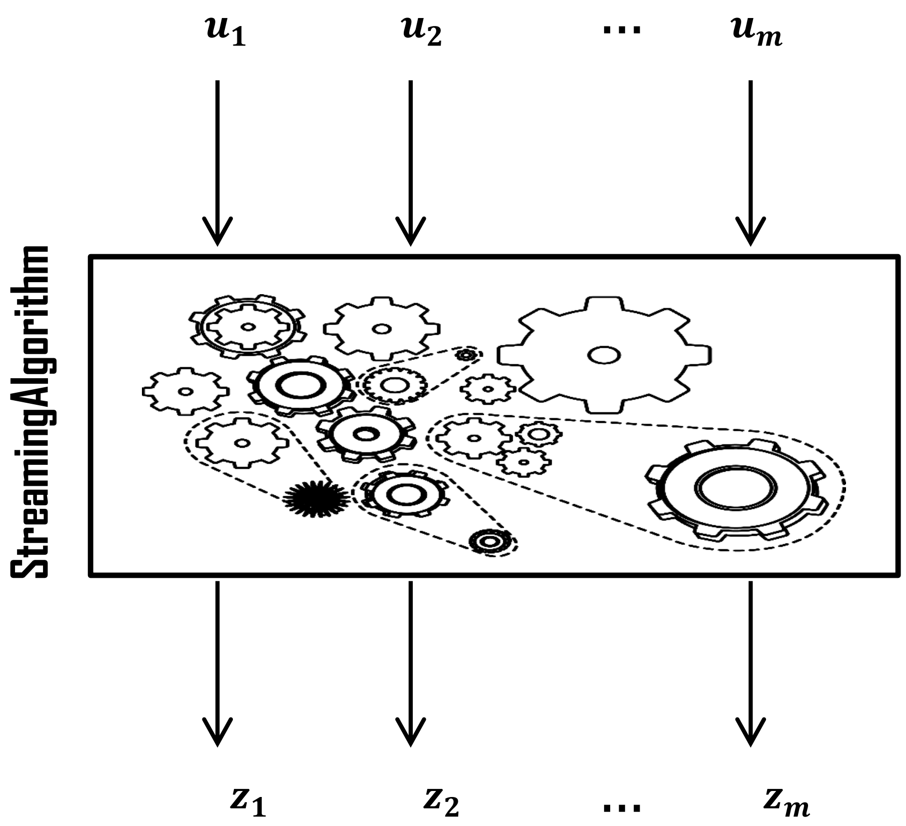


Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

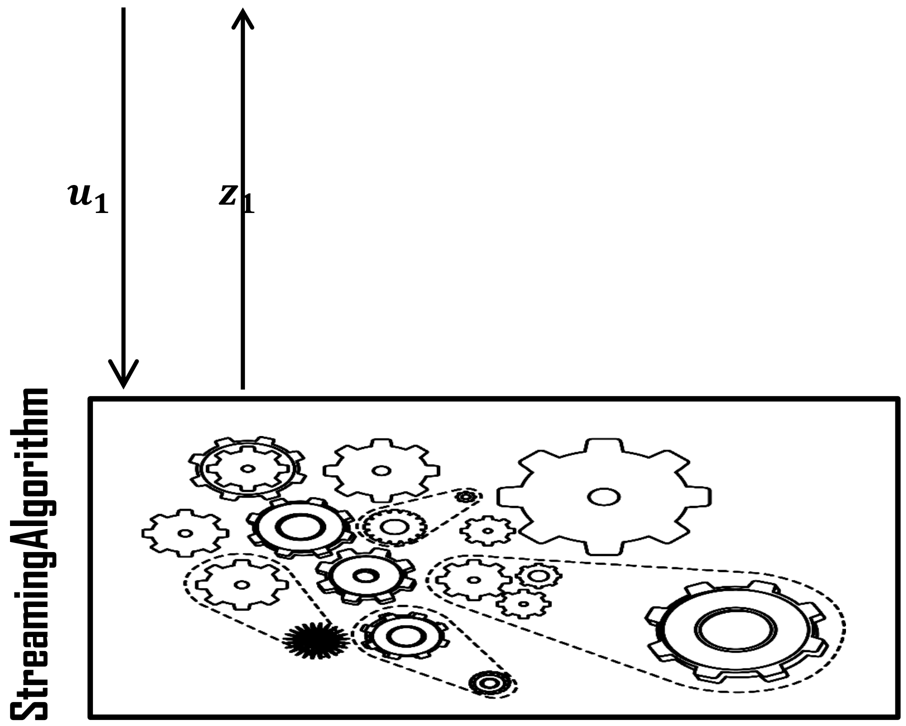
$(u_1, u_2, \ldots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Adversary chooses u_i based on previous answers

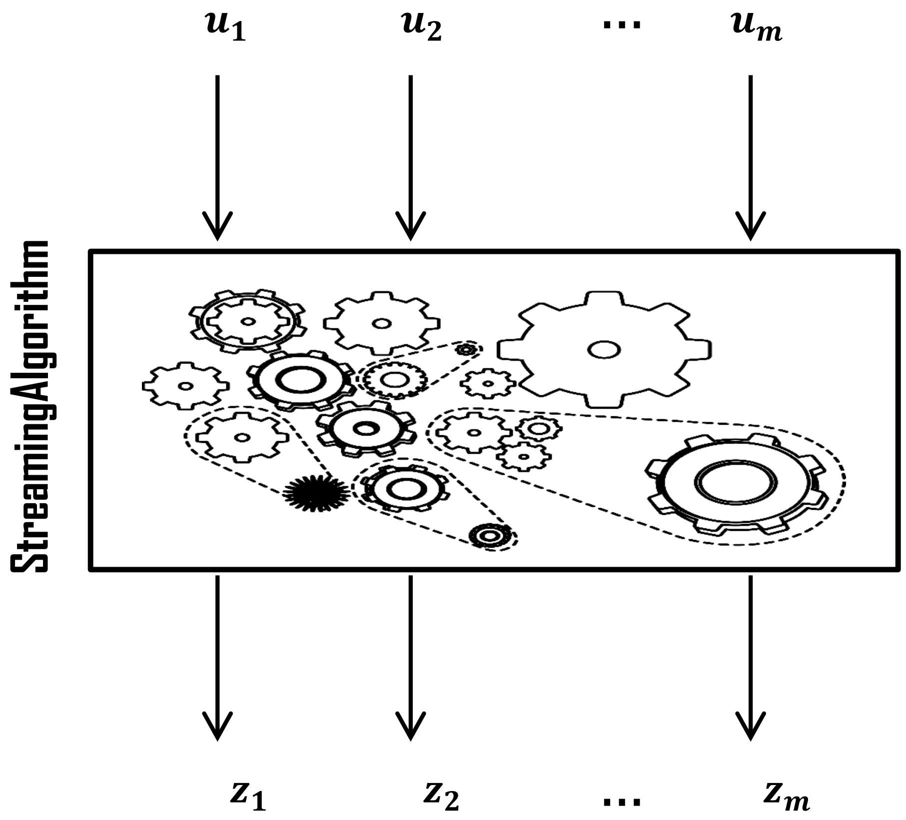


Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

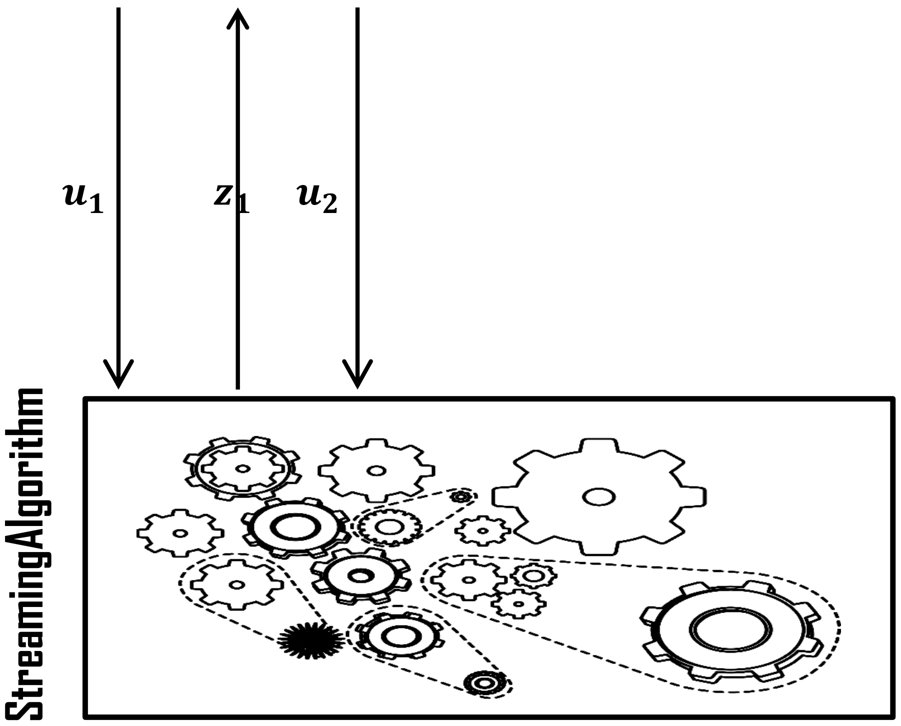
$(u_1, u_2, \dots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Adversary chooses u_i based on previous answers

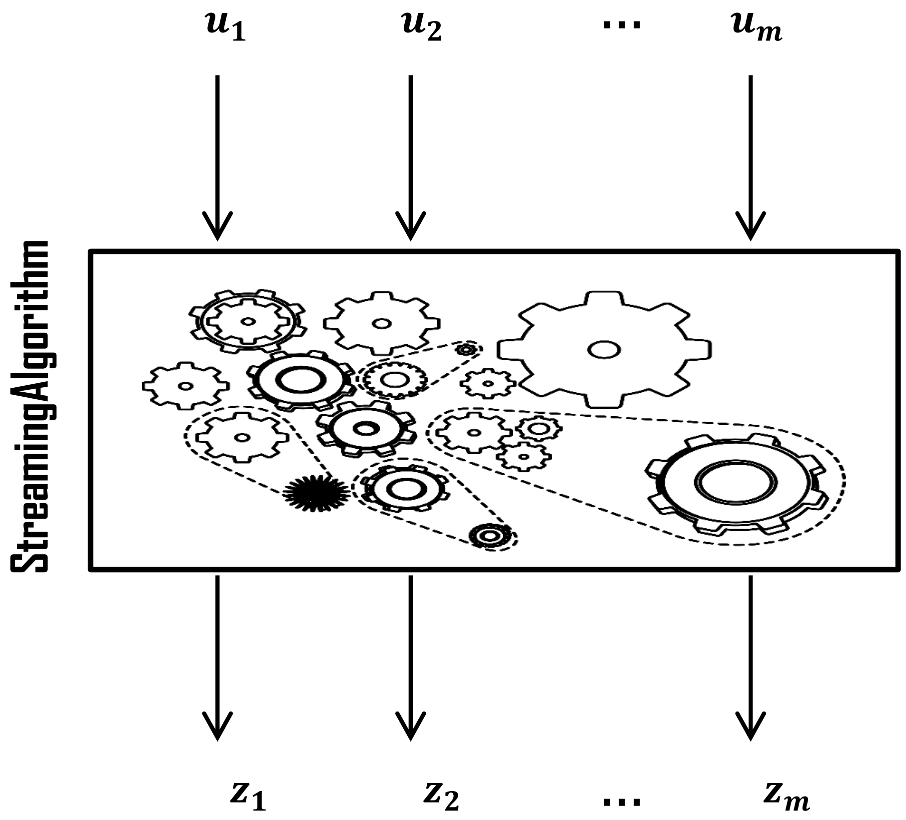


Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

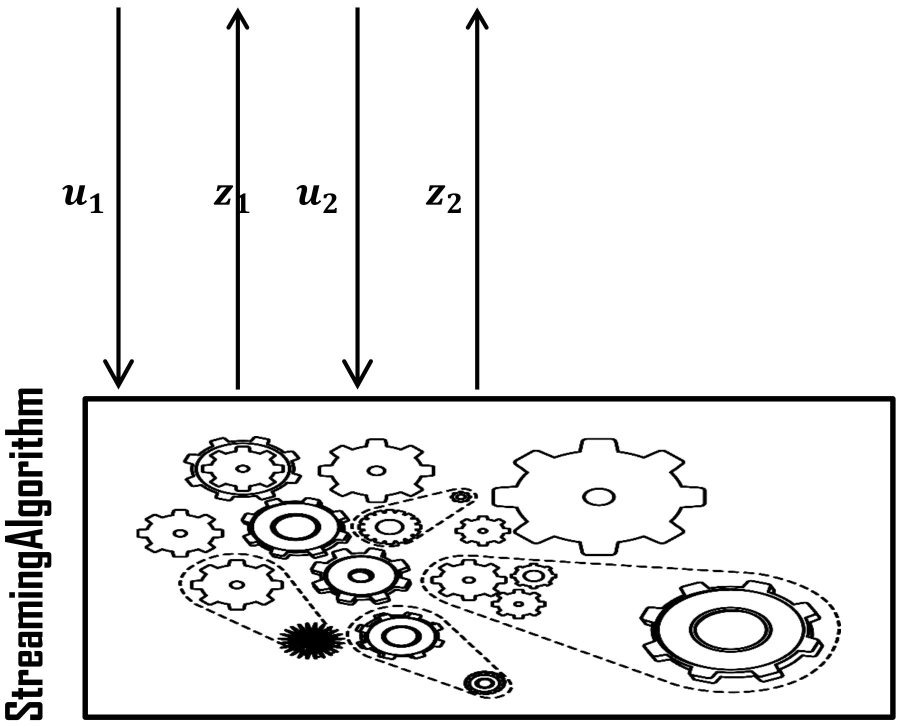
$(u_1, u_2, \dots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Adversary chooses u_i based on previous answers

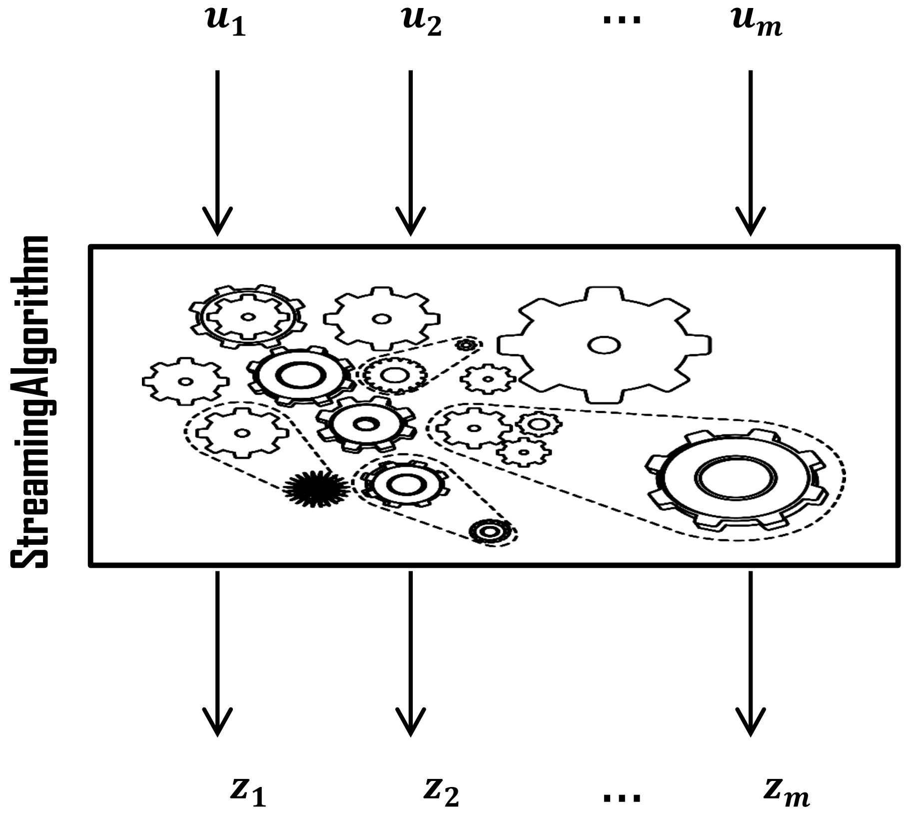


Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

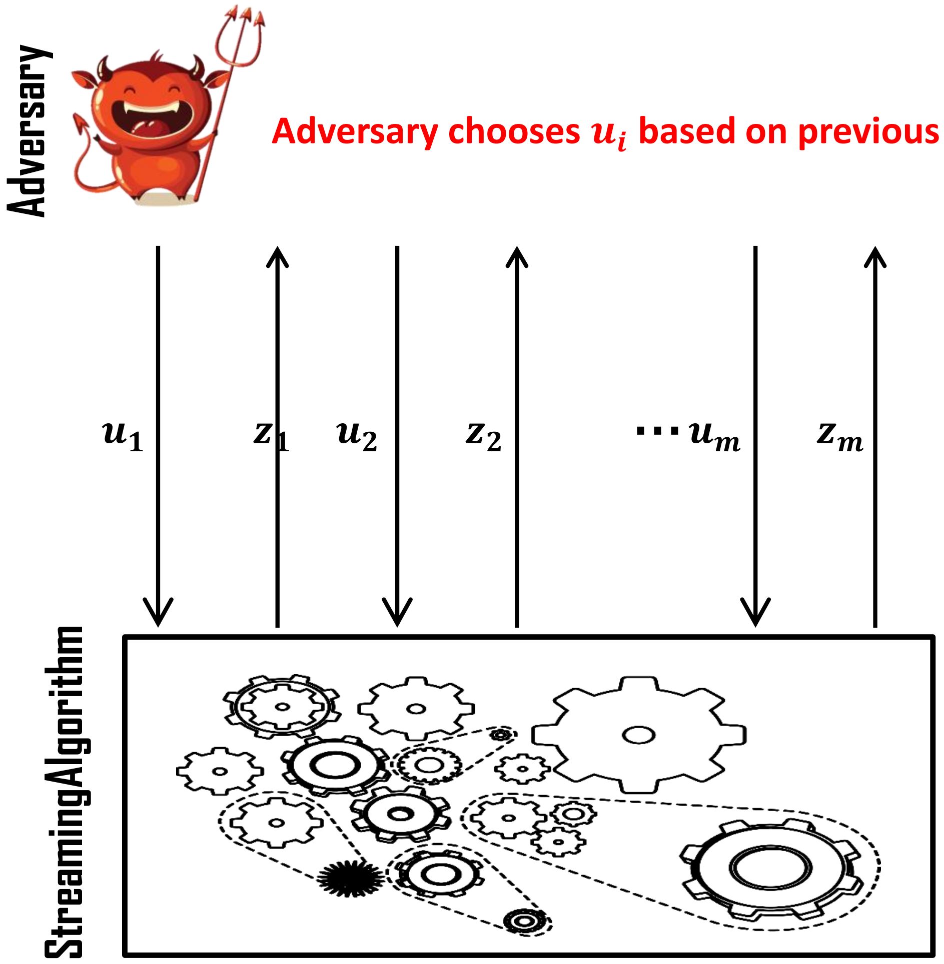
$(u_1, u_2, \dots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Adversary chooses u_i based on previous answers

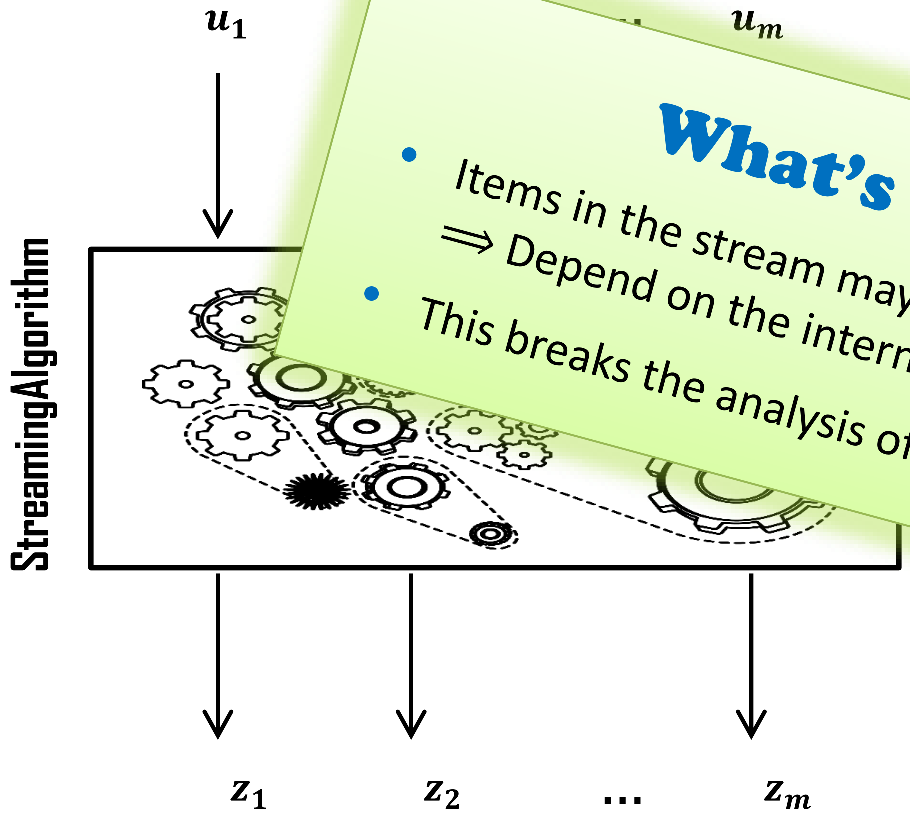


Consider a streaming algorithm that continuously estimates the value of the desired function
(The goal is to minimize space requirements)

Oblivious Streaming

[Alon, Matias, Szegedy '96]

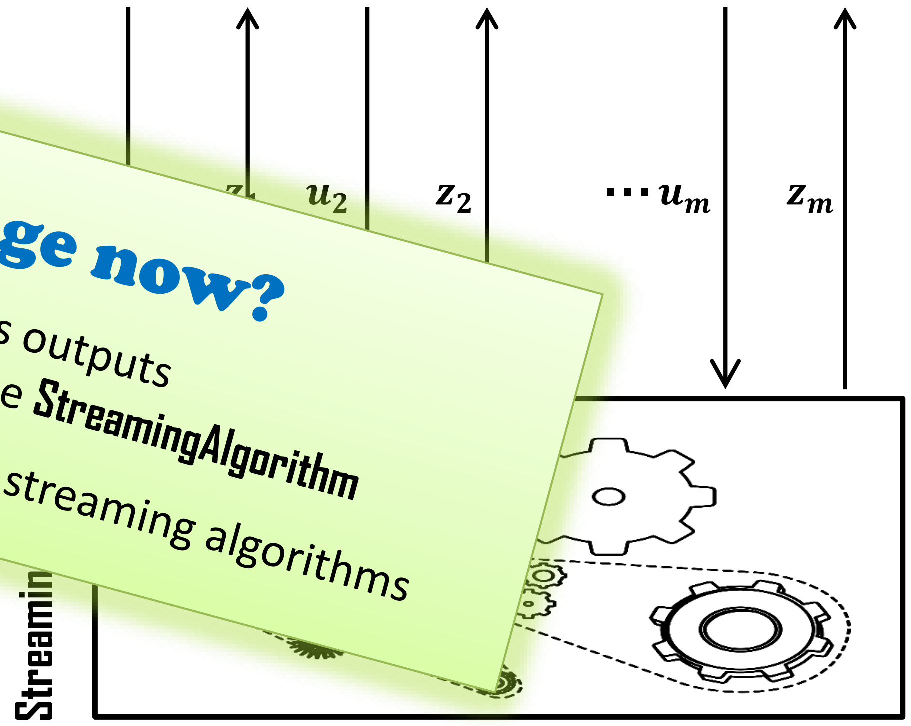
$(u_1, u_2, \dots, u_m)$ = fixed stream (unknown to the algorithm)



Adaptive Streaming

[Hard, Woodruff '13], [Ben-Eliezer, Jayaram, Woodruff, Yogev '20]

Adversary chooses u_i based on previous answers



What's the challenge now?

- Items in the stream may depend on previous outputs
  $\Rightarrow$ Depend on the internal randomness of the **StreamingAlgorithm**
- This breaks the analysis of classical (oblivious) streaming algorithms

The Adversarial Streaming Model

HW13, BJWY20

- Fix a function g mapping a (prefix of the) stream to a real number, and an approximation parameter α
  - E.g., g might count the number of distinct elements in the stream
- Two-player game between a (randomized) **StreamingAlgorithm** and an **Adversary**
- In the ith round:
  1. The **Adversary** chooses an update u_i for the stream, which can depend on all previous stream updates and outputs of **StreamingAlgorithm**
  2. The **StreamingAlgorithm** processes the new update and outputs its current response z_i
- The goal of the **Adversary** is to make the **StreamingAlgorithm** output an incorrect response z_i at some point i

The Adversarial Streaming Model

HW13, BJWY20

- Fix a function g mapping a (prefix of the) stream to a real number, and an approximation parameter α
  - E.g., g might count the number of distinct elements in the stream
- Two-player game between a (randomized) **StreamingAlgorithm** and an **Adversary**
- In the ith round:
  1. The **Adversary** chooses an update u_i for the stream, which can depend on all previous stream updates and outputs of **StreamingAlgorithm**
  2. The **StreamingAlgorithm** processes the new update and outputs its current response z_i
- The goal of the **Adversary** is to make the **StreamingAlgorithm** output an incorrect response z_i at some point i

Do oblivious streaming algorithms work in the adversarial model?

- Deterministic streaming algorithms are adversarially robust
  - However, many streaming algorithms provably **must** be randomized [AMS '96]
- Many of the randomized streaming algorithms are *not* adversarially robust

The Adversarial Streaming Model

HW13, BJWY20

- Fix a function g mapping a (prefix of the) stream to a real number, and an approximation parameter α
  - E.g., g might count the number of distinct elements in the stream
- Two-player game between a (randomized) **StreamingAlgorithm** and an **Adversary**
- In the ith round:
  1. The **Adversary** chooses an update u_i for the stream, which can depend on all previous stream updates and outputs of **StreamingAlgorithm**
  2. The **StreamingAlgorithm** processes the new update and outputs its current response z_i
- The goal of the **Adversary** is to make the **StreamingAlgorithm** output an incorrect response z_i at some point i

Do oblivious streaming algorithms work in the adversarial model?

- Deterministic streaming algorithms are adversarially robust
  - However, many streaming algorithms provably **must** be randomized [AMS '96]
- Many of the randomized streaming algorithms are *not* adversarially robust

Informal takeaway: The difficulty with adversarial streaming is that as time goes by the adversary might learn information about the internal randomness of the algorithm

Example: The AMS sketch for F_2

Alon, Matias,
Szegedy 96

Example: The AMS sketch for F_2

Alon, Matias, Szegedy 96

- Every item in the stream is a pair $(\boldsymbol{u_i}, \boldsymbol{\Delta_i})$ where $\boldsymbol{u_i} \in \mathbb{R}^{\boldsymbol{n}}$ is a standard basis vector and $\boldsymbol{\Delta_i} \in \mathbb{R}$ is its weight
- At every time step $\boldsymbol{i}$, the goal is to estimate $\left\|\boldsymbol{f}^{(i)}\right\|_{\boldsymbol{2}}^{\boldsymbol{2}}$ for $\boldsymbol{f}^{(i)} = \boldsymbol{\Delta_1} \cdot \boldsymbol{u_1} + \cdots + \boldsymbol{\Delta_i} \cdot \boldsymbol{u_i}$

Example: The AMS sketch for F_2

Alon, Matias, Szegedy 96

- Every item in the stream is a pair $(\boldsymbol{u_i}, \boldsymbol{\Delta_i})$ where $\boldsymbol{u_i} \in \mathbb{R}^{\boldsymbol{n}}$ is a standard basis vector and $\boldsymbol{\Delta_i} \in \mathbb{R}$ is its weight
- At every time step $\boldsymbol{i}$, the goal is to estimate $\left\|\boldsymbol{f}^{(i)}\right\|_{\boldsymbol{2}}^{\boldsymbol{2}}$ for $\boldsymbol{f}^{(i)} = \boldsymbol{\Delta_1} \cdot \boldsymbol{u_1} + \cdots + \boldsymbol{\Delta_i} \cdot \boldsymbol{u_i}$

1. Let $\boldsymbol{A}$ be $\boldsymbol{t}x\boldsymbol{n}$ matrix with entries uniformly in $\{\pm\boldsymbol{1}\}$
2. Initiate $\boldsymbol{y} = \vec{\boldsymbol{0}} \in \mathbb{R}^{\boldsymbol{t}}$
3. For $\boldsymbol{i} = \boldsymbol{1}, \boldsymbol{2}, \ldots, \boldsymbol{m}$ do:
   - Obtain the next update vector $\boldsymbol{v_i} = \boldsymbol{\Delta_i} \cdot \boldsymbol{u_i}$
   - Let $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{A} \cdot \boldsymbol{v_i}$
   - Output estimation $\boldsymbol{z_i} = \frac{\boldsymbol{1}}{\boldsymbol{t}} \cdot \|\boldsymbol{y}\|_{\boldsymbol{2}}^{\boldsymbol{2}}$

Example: The AMS sketch for F_2

Alon, Matias, Szegedy 96

- Every item in the stream is a pair $(\boldsymbol{u_i}, \boldsymbol{\Delta_i})$ where $\boldsymbol{u_i} \in \mathbb{R}^{\boldsymbol{n}}$ is a standard basis vector and $\boldsymbol{\Delta_i} \in \mathbb{R}$ is its weight
- At every time step $\boldsymbol{i}$, the goal is to estimate $\left\|\boldsymbol{f}^{(i)}\right\|_2^2$ for $\boldsymbol{f}^{(i)} = \boldsymbol{\Delta_1} \cdot \boldsymbol{u_1} + \cdots + \boldsymbol{\Delta_i} \cdot \boldsymbol{u_i}$

1. Let $\boldsymbol{A}$ be $\boldsymbol{txn}$ matrix with entries uniformly in $\{\pm\boldsymbol{1}\}$
2. Initiate $\boldsymbol{y} = \vec{\boldsymbol{0}} \in \mathbb{R}^{\boldsymbol{t}}$
3. For $\boldsymbol{i} = \boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{m}$ do:
    - Obtain the next update vector $\boldsymbol{v_i} = \boldsymbol{\Delta_i} \cdot \boldsymbol{u_i}$
    - Let $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{A} \cdot \boldsymbol{v_i}$
    - Output estimation $\boldsymbol{z_i} = \frac{\boldsymbol{1}}{\boldsymbol{t}} \cdot \|\boldsymbol{y}\|_2^2$

Analysis:

- Let $\boldsymbol{a_\ell}$ denote the ℓth row of $\boldsymbol{A}$
- Observe: $\boldsymbol{z_i} = \frac{\boldsymbol{1}}{\boldsymbol{t}} \cdot \|\boldsymbol{A} \cdot \boldsymbol{v_1} + \cdots + \boldsymbol{A} \cdot \boldsymbol{v_i}\|_2^2 = \frac{\boldsymbol{1}}{\boldsymbol{t}} \cdot \left\|\boldsymbol{A} \cdot \boldsymbol{f}^{(i)}\right\|_2^2 = \frac{\left(\boldsymbol{a_1} \cdot \boldsymbol{f}^{(i)}\right)^2 + \cdots + \left(\boldsymbol{a_t} \cdot \boldsymbol{f}^{(i)}\right)^2}{\boldsymbol{t}}$

Example: The AMS sketch for F_2

Alon, Matias, Szegedy 96

- Every item in the stream is a pair $(\boldsymbol{u_i}, \boldsymbol{\Delta_i})$ where $\boldsymbol{u_i} \in \mathbb{R}^{\boldsymbol{n}}$ is a standard basis vector and $\boldsymbol{\Delta_i} \in \mathbb{R}$ is its weight
- At every time step $\boldsymbol{i}$, the goal is to estimate $\left\|\boldsymbol{f}^{(i)}\right\|_2^2$ for $\boldsymbol{f}^{(i)} = \boldsymbol{\Delta_1} \cdot \boldsymbol{u_1} + \cdots + \boldsymbol{\Delta_i} \cdot \boldsymbol{u_i}$

1. Let $\boldsymbol{A}$ be $\boldsymbol{txn}$ matrix with entries uniformly in $\{\pm \boldsymbol{1}\}$
2. Initiate $\boldsymbol{y} = \vec{\boldsymbol{0}} \in \mathbb{R}^{\boldsymbol{t}}$
3. For $\boldsymbol{i} = \boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{m}$ do:
   - Obtain the next update vector $\boldsymbol{v_i} = \boldsymbol{\Delta_i} \cdot \boldsymbol{u_i}$
   - Let $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{A} \cdot \boldsymbol{v_i}$
   - Output estimation $\boldsymbol{z_i} = \frac{1}{t} \cdot \|\boldsymbol{y}\|_2^2$

Analysis:

- Let $\boldsymbol{a_\ell}$ denote the ℓth row of $\boldsymbol{A}$
- Observe: $\boldsymbol{z_i} = \frac{1}{t} \cdot \|\boldsymbol{A} \cdot \boldsymbol{v_1} + \cdots + \boldsymbol{A} \cdot \boldsymbol{v_i}\|_2^2 = \frac{1}{t} \cdot \left\|\boldsymbol{A} \cdot \boldsymbol{f}^{(i)}\right\|_2^2 = \frac{\left(\boldsymbol{a_1} \cdot \boldsymbol{f}^{(i)}\right)^2 + \cdots + \left(\boldsymbol{a_t} \cdot \boldsymbol{f}^{(i)}\right)^2}{t}$
- For every (fixed) vector $\boldsymbol{f} \in \mathbb{R}^{\boldsymbol{n}}$ and $\ell \in [\boldsymbol{t}]$ we have

$$\mathbb{E}\left[(\boldsymbol{a_\ell} \cdot \boldsymbol{f})^2\right] = \mathbb{E}\left[\left(\sum_{j \in [n]} \boldsymbol{a_{\ell,j}} \cdot \boldsymbol{f_j}\right)^2\right] \underset{\text{(pairwise)}}{=} \sum_{j \in [n]} \boldsymbol{f_j^2} = \|\boldsymbol{f}\|_2^2$$

Example: The AMS sketch for F_2

Alon, Matias, Szegedy 96

- Every item in the stream is a pair $(\boldsymbol{u_i}, \boldsymbol{\Delta_i})$ where $\boldsymbol{u_i} \in \mathbb{R}^{\boldsymbol{n}}$ is a standard basis vector and $\boldsymbol{\Delta_i} \in \mathbb{R}$ is its weight
- At every time step $\boldsymbol{i}$, the goal is to estimate $\left\|\boldsymbol{f}^{(i)}\right\|_2^2$ for $\boldsymbol{f}^{(i)} = \boldsymbol{\Delta_1} \cdot \boldsymbol{u_1} + \cdots + \boldsymbol{\Delta_i} \cdot \boldsymbol{u_i}$

1. Let $\boldsymbol{A}$ be $\boldsymbol{txn}$ matrix with entries uniformly in $\{\pm\boldsymbol{1}\}$
2. Initiate $\boldsymbol{y} = \vec{\boldsymbol{0}} \in \mathbb{R}^{\boldsymbol{t}}$
3. For $\boldsymbol{i} = \boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{m}$ do:
   - Obtain the next update vector $\boldsymbol{v_i} = \boldsymbol{\Delta_i} \cdot \boldsymbol{u_i}$
   - Let $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{A} \cdot \boldsymbol{v_i}$
   - Output estimation $\boldsymbol{z_i} = \frac{\boldsymbol{1}}{\boldsymbol{t}} \cdot \|\boldsymbol{y}\|_2^2$

Analysis:

- Let $\boldsymbol{a_\ell}$ denote the ℓth row of $\boldsymbol{A}$
- Observe: $\boldsymbol{z_i} = \frac{\boldsymbol{1}}{\boldsymbol{t}} \cdot \|\boldsymbol{A} \cdot \boldsymbol{v_1} + \cdots + \boldsymbol{A} \cdot \boldsymbol{v_i}\|_2^2 = \frac{\boldsymbol{1}}{\boldsymbol{t}} \cdot \left\|\boldsymbol{A} \cdot \boldsymbol{f}^{(i)}\right\|_2^2 = \frac{\left(\boldsymbol{a_1} \cdot \boldsymbol{f}^{(i)}\right)^2 + \cdots + \left(\boldsymbol{a_t} \cdot \boldsymbol{f}^{(i)}\right)^2}{\boldsymbol{t}}$
- For every (fixed) vector $\boldsymbol{f} \in \mathbb{R}^{\boldsymbol{n}}$ and $\ell \in [\boldsymbol{t}]$ we have

$$\mathbb{E}\left[(\boldsymbol{a_\ell} \cdot \boldsymbol{f})^2\right] = \mathbb{E}\left[\left(\sum_{j \in [n]} \boldsymbol{a_{\ell,j}} \cdot \boldsymbol{f_j}\right)^2\right] \underset{\text{(pairwise)}}{=} \sum_{j \in [n]} \boldsymbol{f_j^2} = \|\boldsymbol{f}\|_2^2$$

$\Longrightarrow$ Every $(\boldsymbol{a_\ell} \cdot \boldsymbol{f})^2$ is an unbiased estimator for $\|\boldsymbol{f}\|_2^2$

- Averaging over $\boldsymbol{t}$ reduces variance and improves estimation

Example: The AMS sketch for F_2

Alon, Matias, Szegedy 96

- Every item in the stream is a pair $(\boldsymbol{u_i}, \boldsymbol{\Delta_i})$ where $\boldsymbol{u_i} \in \mathbb{R}^n$ is a standard basis vector and $\boldsymbol{\Delta_i} \in \mathbb{R}$ is its weight
- At every time s ... is to estimate $\|\boldsymbol{f}^{(i)}\|_2^2$ for $\boldsymbol{f}^{(}$ ...

1. Let $\boldsymbol{A}$ be $\boldsymbol{txn}$ matrix with entries uniformly in $\{\pm\boldsymbol{1}\}$
2. Initiate $\boldsymbol{y} = \vec{\boldsymbol{0}} \in \mathbb{R}^{\boldsymbol{t}}$
3. For $\boldsymbol{i} = \boldsymbol{1}, \boldsymbol{2}, \dots, \boldsymbol{m}$ do:
   - Obtain the next update vector $\boldsymbol{v_i} = \boldsymbol{\Delta_i} \cdot \boldsymbol{u_i}$
   - Let $\boldsymbol{y} \leftarrow \boldsymbol{y} + \boldsymbol{A} \cdot \boldsymbol{v_i}$
   - Output estimation $\boldsymbol{z_i} = \frac{\boldsymbol{1}}{\boldsymbol{t}} \cdot \|\boldsymbol{y}\|_2^2$

For the analysis we assumed that the stream is fixed in advance

Analysis:

- Let $\boldsymbol{a_\ell}$ denote the ℓth row of $\boldsymbol{A}$
- Observe: $\boldsymbol{z_i} = \frac{\boldsymbol{1}}{\boldsymbol{t}} \cdot \|\boldsymbol{A} \cdot \boldsymbol{v_1} + \dots + \boldsymbol{A} \cdot \boldsymbol{v_i}\|_2^2 = \frac{\boldsymbol{1}}{\boldsymbol{t}} \cdot \|\boldsymbol{A} \cdot \boldsymbol{f}^{(i)}\|_2$ ... $\frac{\dots \boldsymbol{f}^{(i)}\big)^2 + \dots + \big(\boldsymbol{a_t} \cdot \boldsymbol{f}^{(i)}\big)^2}{\boldsymbol{t}}$
- For every (fixed) vector $\boldsymbol{f} \in \mathbb{R}^{\boldsymbol{n}}$ and $\ell \in [\boldsymbol{t}]$ we have

$$\mathbb{E}\left[(\boldsymbol{a_\ell} \cdot \boldsymbol{f})^2\right] = \mathbb{E}\left[\left(\sum_{\boldsymbol{j} \in [\boldsymbol{n}]} \boldsymbol{a_{\ell,j}} \cdot \boldsymbol{f_j}\right)^2\right] \underset{\text{(pairwise)}}{=} \sum_{\boldsymbol{j} \in [\boldsymbol{n}]} \boldsymbol{f_j^2} = \|\boldsymbol{f}\|_2^2$$

$\Longrightarrow$ Every $(\boldsymbol{a_\ell} \cdot \boldsymbol{f})^2$ is an unbiased estimator for $\|\boldsymbol{f}\|_2^2$

- Averaging over $\boldsymbol{t}$ reduces variance and improves estimation

Adversary for the AMS Sketch

HW13, BJWY20

Adversary for the AMS Sketch

HW13, BJWY20

Recall AMS sketch

- Random matrix $\boldsymbol{A} \in \{\pm\mathbf{1}\}^{\boldsymbol{t}\times\boldsymbol{n}}$
- After the $\boldsymbol{i}$th update, respond with $\frac{\mathbf{1}}{\boldsymbol{t}}\left\|\boldsymbol{A}\cdot\boldsymbol{f}^{(\boldsymbol{i})}\right\|_{\mathbf{2}}^{\mathbf{2}} = \left\|\frac{\mathbf{1}}{\sqrt{\boldsymbol{t}}}\boldsymbol{A}\cdot\boldsymbol{f}^{(\boldsymbol{i})}\right\|_{\mathbf{2}}^{\mathbf{2}}$ where $\boldsymbol{f}^{(\boldsymbol{i})} = \boldsymbol{\Delta}_{\mathbf{1}}\cdot\boldsymbol{u}_{\mathbf{1}} + \cdots + \boldsymbol{\Delta}_{\boldsymbol{i}}\cdot\boldsymbol{u}_{\boldsymbol{i}}$

Adversary for the AMS Sketch

HW13, BJWY20

Recall AMS sketch

- Random matrix $\boldsymbol{A} \in \{\pm\boldsymbol{1}\}^{\boldsymbol{t}\times\boldsymbol{n}}$
- After the $\boldsymbol{i}$th update, respond with $\frac{1}{t}\left\|\boldsymbol{A}\cdot\boldsymbol{f}^{(i)}\right\|_2^2 = \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{f}^{(i)}\right\|_2^2$ where $\boldsymbol{f}^{(i)} = \boldsymbol{\Delta}_1\cdot\boldsymbol{u}_1 + \cdots + \boldsymbol{\Delta}_i\cdot\boldsymbol{u}_i$

The attack

- Set $\boldsymbol{w} \leftarrow \boldsymbol{C}\cdot\sqrt{\boldsymbol{t}}\cdot\boldsymbol{e}_1$
- For $\boldsymbol{i} = \boldsymbol{2}, \boldsymbol{3}, \dots, \boldsymbol{m} = \boldsymbol{O}(\boldsymbol{t})$ do
  1. $\mathbf{old} \leftarrow \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{w}\right\|_2^2$
  2. $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{e}_i$
  3. $\mathbf{new} \leftarrow \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{w}\right\|_2^2$
  4. If $\mathbf{new} > \mathbf{old}$ then $\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{e}_i$

Adversary for the AMS Sketch

HW13, BJWY20

Recall AMS sketch

- Random matrix $\boldsymbol{A} \in \{\pm\mathbf{1}\}^{t\times n}$
- After the ith update, respond with $\frac{1}{t}\left\|A \cdot f^{(i)}\right\|_2^2 = \left\|\frac{1}{\sqrt{t}} A \cdot f^{(i)}\right\|_2^2$ where $f^{(i)} = \Delta_1 \cdot u_1 + \cdots + \Delta_i \cdot u_i$

The attack

- Set $w \leftarrow C \cdot \sqrt{t} \cdot e_1$
- For $i = 2, 3, \dots, m = O(t)$ do
  1. $\mathbf{old} \leftarrow \left\|\frac{1}{\sqrt{t}} A \cdot w\right\|_2^2$
  2. $w \leftarrow w + e_i$
  3. $\mathbf{new} \leftarrow \left\|\frac{1}{\sqrt{t}} A \cdot w\right\|_2^2$
  4. If $\mathbf{new} > \mathbf{old}$ then $w \leftarrow w - e_i$

Analysis

- At all times $\|w\|_2^2 \geq C^2 \cdot t$ by init

  $\Longrightarrow$ Suffices to show that $\left\|\frac{1}{\sqrt{t}} A \cdot w\right\|_2^2$ drops below $C^2/2 \cdot t$

Adversary for the AMS Sketch

HW13, BJWY20

Recall AMS sketch

- Random matrix $\boldsymbol{A} \in \{\pm\mathbf{1}\}^{t\times n}$
- After the $\boldsymbol{i}$th update, respond with $\frac{1}{t}\left\|\boldsymbol{A}\cdot \boldsymbol{f}^{(i)}\right\|_2^2 = \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot \boldsymbol{f}^{(i)}\right\|_2^2$ where $\boldsymbol{f}^{(i)} = \boldsymbol{\Delta}_1\cdot \boldsymbol{u}_1 + \cdots + \boldsymbol{\Delta}_i \cdot \boldsymbol{u}_i$

The attack

- Set $\boldsymbol{w} \leftarrow \boldsymbol{C}\cdot\sqrt{\boldsymbol{t}}\cdot \boldsymbol{e}_1$
- For $\boldsymbol{i} = \mathbf{2}, \mathbf{3}, \dots, \boldsymbol{m} = \boldsymbol{O}(\boldsymbol{t})$ do
  1. $\mathbf{old} \leftarrow \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{w}\right\|_2^2$
  2. $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{e}_i$
  3. $\mathbf{new} \leftarrow \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{w}\right\|_2^2$
  4. If $\mathbf{new} > \mathbf{old}$ then $\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{e}_i$

Analysis

- At all times $\|\boldsymbol{w}\|_2^2 \ge \boldsymbol{C}^2\cdot \boldsymbol{t}$ by init

  $\Longrightarrow$ Suffices to show that $\left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{w}\right\|_2^2$ drops below $\boldsymbol{C}^2/\mathbf{2}\cdot \boldsymbol{t}$

- $\mathbf{new}_i = \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot(\boldsymbol{w}+\boldsymbol{e}_i)\right\|_2^2$

  $= \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{w}\right\|_2^2 + \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{e}_i\right\|_2^2 + 2\left\langle\frac{1}{\sqrt{t}}\boldsymbol{A}\boldsymbol{w},\ \frac{1}{\sqrt{t}}\boldsymbol{A}\boldsymbol{e}_i\right\rangle$

  $= \mathbf{old}_i + \mathbf{1} + \mathbf{2}\left\langle\frac{1}{\sqrt{t}}\boldsymbol{A}\boldsymbol{w},\ \frac{1}{\sqrt{t}}\boldsymbol{A}\boldsymbol{e}_i\right\rangle$

Adversary for the AMS Sketch

HW13, BJWY20

Recall AMS sketch

- Random matrix $\boldsymbol{A} \in \{\pm\mathbf{1}\}^{t\times n}$
- After the ith update, respond with $\frac{1}{t}\left\|\boldsymbol{A}\cdot \boldsymbol{f}^{(i)}\right\|_2^2 = \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot \boldsymbol{f}^{(i)}\right\|_2^2$ where $\boldsymbol{f}^{(i)} = \boldsymbol{\Delta}_1\cdot \boldsymbol{u}_1 + \cdots + \boldsymbol{\Delta}_i \cdot \boldsymbol{u}_i$

The attack

- Set $\boldsymbol{w} \leftarrow \boldsymbol{C}\cdot\sqrt{\boldsymbol{t}}\cdot \boldsymbol{e}_1$
- For $\boldsymbol{i} = \mathbf{2}, \mathbf{3}, \ldots, \boldsymbol{m} = \boldsymbol{O}(\boldsymbol{t})$ do
  1. $\mathbf{old} \leftarrow \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{w}\right\|_2^2$
  2. $\boldsymbol{w} \leftarrow \boldsymbol{w} + \boldsymbol{e}_i$
  3. $\mathbf{new} \leftarrow \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{w}\right\|_2^2$
  4. If $\mathbf{new} > \mathbf{old}$ then $\boldsymbol{w} \leftarrow \boldsymbol{w} - \boldsymbol{e}_i$

Analysis

- At all times $\|\boldsymbol{w}\|_2^2 \geq \boldsymbol{C}^2\cdot \boldsymbol{t}$ by init

  $\Longrightarrow$ Suffices to show that $\left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{w}\right\|_2^2$ drops below $\boldsymbol{C}^2/\mathbf{2}\cdot \boldsymbol{t}$

- $$\mathbf{new}_i = \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot(\boldsymbol{w}+\boldsymbol{e}_i)\right\|_2^2$$
  $$= \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{w}\right\|_2^2 + \left\|\frac{1}{\sqrt{t}}\boldsymbol{A}\cdot\boldsymbol{e}_i\right\|_2^2 + 2\left\langle\frac{1}{\sqrt{t}}\boldsymbol{A}\boldsymbol{w},\ \frac{1}{\sqrt{t}}\boldsymbol{A}\boldsymbol{e}_i\right\rangle$$
  $$= \mathbf{old}_i + 1 + 2\left\langle\frac{1}{\sqrt{t}}\boldsymbol{A}\boldsymbol{w},\ \frac{1}{\sqrt{t}}\boldsymbol{A}\boldsymbol{e}_i\right\rangle$$

- So, $\mathbf{new}_i - \mathbf{old}_i \approx 2\left\langle\frac{1}{\sqrt{t}}\boldsymbol{A}\boldsymbol{w},\ \frac{1}{\sqrt{t}}\boldsymbol{A}\boldsymbol{e}_i\right\rangle$

- This inner product is symmetric, and is "negative enough" with constant prob.

Adversarial Streaming via Differential Privacy

Thm (proven in the next slide):

Oblivious alg $\mathcal{A}$ $\Longrightarrow$ Adversarially robust alg using space $\tilde{O}\left(\sqrt{m}\cdot \mathrm{Space}(\mathcal{A})\right)$

- The idea is to protect the ***internal randomness*** of the algorithm using differential privacy
- This limits (in a precise way) the dependency between the internal randomness of the algorithm and the choices of the adversary
- Notice that differential privacy is ***not*** used here for data privacy. We are ***not*** protecting the privacy of the data items in the stream; only the secrecy of the internal randomness.

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\widetilde{O}\left(\sqrt{m} \cdot \text{Space}(\mathcal{A})\right)$

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \mathrm{Space}(\mathcal{A})\right)$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \dots, r_k) \in (\{0,1\}^*)^k$

1. Initiate k independent instances $\mathcal{A}_1, \dots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \dots, r_k$
2. For $i = 1, 2, \dots, m$:
   a) Receive next update u_i
   b) Insert update u_i into each of $\mathcal{A}_1, \dots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \dots, y_{i,k}$
   c) Output $z_i = \mathbf{PrivateMedian}\left(y_{i,1}, \dots, y_{i,k}\right)$

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \mathrm{Space}(\mathcal{A})\right)$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \ldots, r_k) \in (\{0,1\}^*)^k$

1. Initiate k independent instances $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \ldots, r_k$
2. For $i = 1, 2, \ldots, m$:
   a) Receive next update u_i
   b) Insert update u_i into each of $\mathcal{A}_1, \ldots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \ldots, y_{i,k}$
   c) Output $z_i = \mathbf{PrivateMedian}\left(y_{i,1}, \ldots, y_{i,k}\right)$

Analysis idea:

- $\mathcal{B}$ is differentially private w.r.t. the collection of strings R

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \mathrm{Space}(\mathcal{A})\right)$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \dots, r_k) \in (\{0,1\}^*)^k$

1. Initiate k independent instances $\mathcal{A}_1, \dots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \dots, r_k$
2. For $i = 1, 2, \dots, m$:
   a) Receive next update u_i
   b) Insert update u_i into each of $\mathcal{A}_1, \dots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \dots, y_{i,k}$
   c) Output $z_i = \mathbf{PrivateMedian}\left(y_{i,1}, \dots, y_{i,k}\right)$

Analysis idea:

- $\mathcal{B}$ is differentially private w.r.t. the collection of strings R
- Fix $i \in [m]$ and let $\vec{u}_i = (u_1, \dots, u_i)$ denote the first i updates
- Let $\mathcal{A}(r, \vec{u}_i)$ denote the output of $\mathcal{A}$ after the ith update when it is executed with randomness r on stream $\vec{u}_i$

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \mathrm{Space}(\mathcal{A})\right)$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \ldots, r_k) \in (\{0,1\}^*)^k$

1. Initiate k independent instances $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \ldots, r_k$
2. For $i = 1, 2, \ldots, m$:
   a) Receive next update u_i
   b) Insert update u_i into each of $\mathcal{A}_1, \ldots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \ldots, y_{i,k}$
   c) Output $z_i = \mathbf{PrivateMedian}\left(y_{i,1}, \ldots, y_{i,k}\right)$

Analysis idea:

- $\mathcal{B}$ is differentially private w.r.t. the collection of strings R
- Fix $i \in [m]$ and let $\vec{u}_i = (u_1, \ldots, u_i)$ denote the first i updates
- Let $\mathcal{A}(r, \vec{u}_i)$ denote the output of $\mathcal{A}$ after the ith update when it is executed with randomness r on stream $\vec{u}_i$
- Consider the function $f_{\vec{u}_i}(r) = \mathbb{1}\{\mathcal{A}(r, \vec{u}_i) \in (1 \pm \alpha) \cdot g(\vec{u}_i)\}$

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \text{Space}(\mathcal{A})\right)$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \ldots, r_k) \in (\{0,1\}^*)^k$

1. Initiate k independent instances $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \ldots, r_k$
2. For $i = 1, 2, \ldots, m$:
   a) Receive next update u_i
   b) Insert update u_i into each of $\mathcal{A}_1, \ldots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \ldots, y_{i,k}$
   c) Output $z_i = \mathbf{PrivateMedian}\left(y_{i,1}, \ldots, y_{i,k}\right)$

Analysis idea:

- $\mathcal{B}$ is differentially private w.r.t. the collection of strings R
- Fix $i \in [m]$ and let $\vec{u}_i = (u_1, \ldots, u_i)$ denote the first i updates
- Let $\mathcal{A}\left(r, \vec{u}_i\right)$ denote the output of $\mathcal{A}$ after the ith update when it is executed with randomness r on stream $\vec{u}_i$
- Consider the function $f_{\vec{u}_i}(r) = \mathbb{1}\left\{\mathcal{A}\left(r, \vec{u}_i\right) \in (1 \pm \alpha) \cdot g\left(\vec{u}_i\right)\right\}$
- Observe that $\vec{u}_i$ is the result of a private computation on R (post-processing $\mathcal{B}$'s answers), and hence, so is $f_{\vec{u}_i}$

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \mathrm{Space}(\mathcal{A})\right)$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \dots, r_k) \in (\{0,1\}^*)^k$

1. Initiate k independent instances $\mathcal{A}_1, \dots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \dots, r_k$
2. For $i = 1, 2, \dots, m$:
   a) Receive next update u_i
   b) Insert update u_i into each of $\mathcal{A}_1, \dots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \dots, y_{i,k}$
   c) Output $z_i = \mathbf{PrivateMedian}\left(y_{i,1}, \dots, y_{i,k}\right)$

Analysis idea:

- $\mathcal{B}$ is differentially private w.r.t. the collection of strings R
- Fix $i \in [m]$ and let $\vec{u}_i = (u_1, \dots, u_i)$ denote the first i updates
- Let $\mathcal{A}(r, \vec{u}_i)$ denote the output of $\mathcal{A}$ after the ith update when it is executed with randomness r on stream $\vec{u}_i$
- Consider the function $f_{\vec{u}_i}(r) = \mathbb{1}\{\mathcal{A}(r, \vec{u}_i) \in (1 \pm \alpha) \cdot g(\vec{u}_i)\}$
- Observe that $\vec{u}_i$ is the result of a private computation on R (post-processing $\mathcal{B}$'s answers), and hence, so is $f_{\vec{u}_i}$
- By the generalization properties of DP we have $\frac{1}{k} \cdot \sum_{j=1}^{k} f_{\vec{u}_i}(r_j) \approx \mathbb{E}_r[f_{\vec{u}_i}(r)]$

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \text{Space}(\mathcal{A})\right)$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \ldots, r_k) \in (\{0,1\}^*)^k$

1. Initiate k independent instances $\mathcal{A}_1, \ldots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \ldots, r_k$
2. For $i = 1, 2, \ldots, m$:
   a) Receive next update u_i
   b) Insert update u_i into each of $\mathcal{A}_1, \ldots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \ldots, y_{i,k}$
   c) Output $z_i = \textbf{PrivateMedian}\left(y_{i,1}, \ldots, y_{i,k}\right)$

Analysis idea:

- $\mathcal{B}$ is differentially private w.r.t. the collection of strings R
- Fix $i \in [m]$ and let $\vec{u}_i = (u_1, \ldots, u_i)$ denote the first i updates
- Let $\mathcal{A}(r, \vec{u}_i)$ denote the output of $\mathcal{A}$ after the ith update when it is executed with randomness r on stream $\vec{u}_i$
- Consider the function $f_{\vec{u}_i}(r) = \mathbb{1}\{\mathcal{A}(r, \vec{u}_i) \in (1 \pm \alpha) \cdot g(\vec{u}_i)\}$
- Observe that $\vec{u}_i$ is the result of a private computation on R (post-processing $\mathcal{B}$'s answers), and hence, so is $f_{\vec{u}_i}$
- By the generalization properties of DP we have $\frac{1}{k} \cdot \sum_{j=1}^{k} f_{\vec{u}_i}(r_j) \approx \mathbb{E}_r\left[f_{\vec{u}_i}(r)\right] \approx 1$

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \mathrm{Space}(\mathcal{A})\right)$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \dots, r_k) \in (\{0,1\}^*)^k$

1. Initiate k independent instances $\mathcal{A}_1, \dots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \dots, r_k$
2. For $i = 1, 2, \dots, m$:
   a) Receive next update u_i
   b) Insert update u_i into each of $\mathcal{A}_1, \dots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \dots, y_{i,k}$
   c) Output $z_i = \mathbf{PrivateMedian}\left(y_{i,1}, \dots, y_{i,k}\right)$

Analysis idea:

- $\mathcal{B}$ is differentially private w.r.t. the collection of strings R
- Fix $i \in [m]$ and let $\vec{u}_i = (u_1, \dots, u_i)$ denote the first i updates
- Let $\mathcal{A}\left(r, \vec{u}_i\right)$ denote the output of $\mathcal{A}$ after the ith update when it is executed with randomness r on stream $\vec{u}_i$
- Consider the function $f_{\vec{u}_i}(r) = \mathbb{1}\left\{\mathcal{A}\left(r, \vec{u}_i\right) \in (1 \pm \alpha) \cdot g\left(\vec{u}_i\right)\right\}$
- Observe that $\vec{u}_i$ is the result of a private computation on R (post-processing $\mathcal{B}$'s answers), and hence, so is $f_{\vec{u}_i}$
- By the generalization properties of DP we have $\frac{1}{k} \cdot \sum_{j=1}^{k} \mathbb{1}\left\{y_{i,j} \text{ is accurate}\right\} \approx \frac{1}{k} \cdot \sum_{j=1}^{k} f_{\vec{u}_i}\left(r_j\right) \approx \mathbb{E}_r\left[f_{\vec{u}_i}(r)\right] \approx 1$

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \text{Space}(\mathcal{A})\right)$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \dots, r_k) \in (\{0,1\}^*)^k$

1. Initiate k independent instances $\mathcal{A}_1, \dots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \dots, r_k$
2. For $i = 1, 2, \dots, m$:
   a) Receive next update u_i
   b) Insert update u_i into each of $\mathcal{A}_1, \dots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \dots, y_{i,k}$
   c) Output $z_i = \mathbf{PrivateMedian}\left(y_{i,1}, \dots, y_{i,k}\right)$

Analysis idea:

- $\mathcal{B}$ is differentially private w.r.t. the collection of strings R
- Fix $i \in [m]$ and let $\vec{u}_i = (u_1, \dots, u_i)$ denote the first i updates
- Let $\mathcal{A}(r, \vec{u}_i)$ denote the output of $\mathcal{A}$ after the ith update when it is executed with randomness r on stream $\vec{u}_i$
- Consider the function $f_{\vec{u}_i}(r) = \mathbb{1}\{\mathcal{A}(r, \vec{u}_i) \in (1 \pm \alpha) \cdot g(\vec{u}_i)\}$
- Observe that $\vec{u}_i$ is the result of a private computation on R (post-processing $\mathcal{B}$'s answers), and hence, so is $f_{\vec{u}_i}$
- By the generalization properties of DP we have $\frac{1}{k} \cdot \sum_{j=1}^{k} \mathbb{1}\{y_{i,j} \text{ is accurate}\} \approx \frac{1}{k} \cdot \sum_{j=1}^{k} f_{\vec{u}_i}(r_j) \approx \mathbb{E}_r[f_{\vec{u}_i}(r)] \approx 1$
- So, most of the $y_{i,j}$'s are accurate, and hence, any approximate median is also accurate

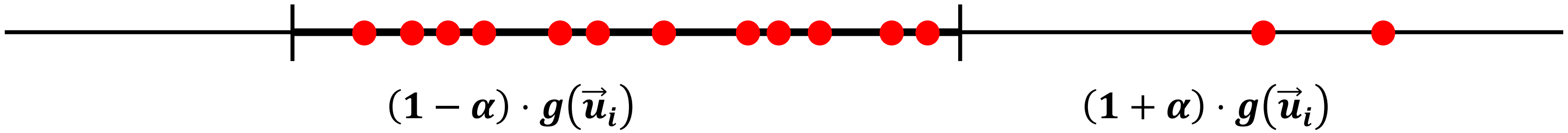
Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \mathrm{Space}(\mathcal{A})\right)$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \dots, r_k) \in (\{0,1\}^*)^k$

1. Initiate k independent instances $\mathcal{A}_1, \dots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \dots, r_k$
2. For $i = 1, 2, \dots, m$:
   a) Receive next update u_i
   b) Insert update u_i into each of $\mathcal{A}_1, \dots, \mathcal{A}_k$ and obtain answers $y_{i,1}, \dots, y_{i,k}$
   c) Output $z_i = \mathbf{PrivateMedian}\left(y_{i,1}, \dots, y_{i,k}\right)$

Analysis idea:

- $\mathcal{B}$ is differentially private w.r.t. the collection of strings R
- Fix $i \in [m]$ and let $\vec{u}_i = (u_1, \dots, u_i)$ denote the first i updates
- Let $\mathcal{A}(r, \vec{u}_i)$ denote the output of $\mathcal{A}$ after the ith update when it is executed with randomness r on stream $\vec{u}_i$
- Consider the function $f_{\vec{u}_i}(r) = \mathbb{1}\{\mathcal{A}(r, \vec{u}_i) \in (1 \pm \alpha) \cdot g(\vec{u}_i)\}$
- Observe that $\vec{u}_i$ is the result of a private computation on R (post-processing $\mathcal{B}$'s answers), and hence, so is $f_{\vec{u}_i}$
- By the generalization properties of DP we have $\frac{1}{k} \cdot \sum_{j=1}^{k} \mathbb{1}\{y_{i,j} \text{ is accurate}\} \approx \frac{1}{k} \cdot \sum_{j=1}^{k} f_{\vec{u}_i}(r_j) \approx \mathbb{E}_r\left[f_{\vec{u}_i}(r)\right] \approx 1$
- So, most of the $y_{i,j}$'s are accurate, and hence, any approximate median is also accurate



Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \text{Space}(\mathcal{A})\right)$

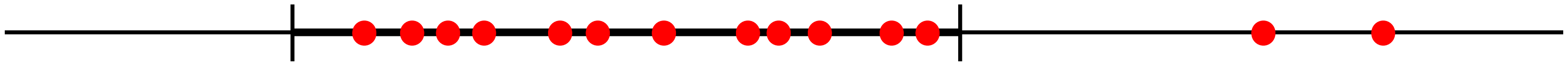
Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \dots, r_k) \in (\{0,1\}^*)^k$

1. Initiate k independent instances $\mathcal{A}_1, \dots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \dots, r_k$
2. For $i = 1, 2, \dots, m$:

These ideas can be formalized to show the following theorem:

Let $\mathcal{A}$ be an oblivious alg for g. There is an adversarially robust alg $\mathcal{B}$ for g using space $\tilde{O}\left(\sqrt{m} \cdot \mathbf{Space}(\mathcal{A})\right)$

- $\mathcal{B}$ is differentially private w.r.t. the collection of strings R
- Fix $i \in [m]$ and let $\vec{u}_i = (u_1, \dots, u_i)$ denote the first i updates
- Let $\mathcal{A}(r, \vec{u}_i)$ denote the output of $\mathcal{A}$ after the ith update when it is executed with randomness r on stream $\vec{u}_i$
- Consider the function $f_{\vec{u}_i}(r) = \mathbb{1}\{\mathcal{A}(r, \vec{u}_i) \in (1 \pm \alpha) \cdot g(\vec{u}_i)\}$
- Observe that $\vec{u}_i$ is the result of a private computation on R (post-processing $\mathcal{B}$'s answers), and hence, so is $f_{\vec{u}_i}$
- By the generalization properties of DP we have $\frac{1}{k} \cdot \sum_{j=1}^{k} \mathbb{1}\{y_{i,j} \text{ is accurate}\} \approx \frac{1}{k} \cdot \sum_{j=1}^{k} f_{\vec{u}_i}(r_j) \approx \mathbb{E}_r[f_{\vec{u}_i}(r)] \approx 1$
- So, most of the $y_{i,j}$'s are accurate, and hence, any approximate median is also accurate



$(1-\alpha) \cdot g(\vec{u}_i)$ $\qquad$ $(1+\alpha) \cdot g(\vec{u}_i)$

Oblivious Alg $\mathcal{A} \Longrightarrow$ Adversarially Robust Alg $\mathcal{B}$ with Space $\tilde{O}\left(\sqrt{m} \cdot \text{Space}(\mathcal{A})\right)$

Input: Collection of $k \approx \sqrt{m}$ random strings $R = (r_1, \dots, r_k) \in (\{0,1\}^*)^k$

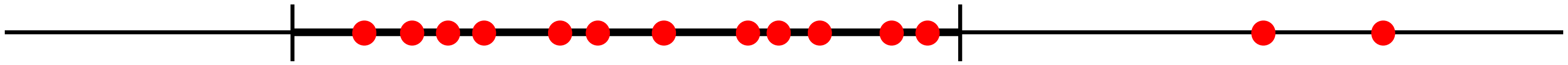
1. Initiate k independent instances $\mathcal{A}_1, \dots, \mathcal{A}_k$ of the oblivious algorithm $\mathcal{A}$ with random strings $r_1, \dots, r_k$
2. For $i = 1, 2, \dots, m$:

These ideas can be formalized to show the following theorem:

Let $\mathcal{A}$ be an oblivious alg for g. There is an adversarially robust alg $\mathcal{B}$ for g using space $\tilde{O}\left(\sqrt{m} \cdot \mathbf{Space}(\mathcal{A})\right)$

Main Takeaway:

- Differential privacy can be used to "hide" the internal randomness of the streaming algorithm from the adversary
- Intuitively, this brings us back to the oblivious setting, where guaranteeing accuracy is significantly easier



$(1-\alpha) \cdot g(\vec{u}_i)$ $(1+\alpha) \cdot g(\vec{u}_i)$

Another application:
Dynamic algorithms with adaptive adversaries

- Similar to adversarial streaming, except that the focus is on runtime instead of space

Another application: Dynamic algorithms with adaptive adversaries

- Similar to adversarial streaming, except that the focus is on runtime instead of space

- Example: Consider a dynamic graph problem where on every time step:
  - The current input specify one edge modification to the graph (either add or remove an edge)
  - We process this input and output a modified approximation for the size of the global min-cut in the graph

Another application: Dynamic algorithms with adaptive adversaries

- Similar to adversarial streaming, except that the focus is on runtime instead of space

- Example: Consider a dynamic graph problem where on every time step:
  - The current input specify one edge modification to the graph (either add or remove an edge)
  - We process this input and output a modified approximation for the size of the global min-cut in the graph

- The hope is that since only one edge was changed, then we won't need to re-compute the size of the global min-cut from scratch. The focus in this line of works is on designing algorithms with fast response time

- Using DP to protect the internal randomness currently results in the fastest algorithms for the adaptive setting

Conclusion

Main Takeaways:

- Strong connection between ability of **adaptive** computations to **remain faithful**, and the **amount of information** that they **leak**
- Differential privacy plays a key role in the state of the art methods

Differential privacy without a central database

Boston Differential Privacy Summer School, 6-10 June 2022

Uri Stemmer

About this course

1) **The local model**
   - What is the model?
   - Computing histograms
   - Computing averages
   - Clustering
   - LDP vs. statistical queries
   - Impossibility result for histograms
   - Interactive LDP protocols
2) **The shuffle model**
   - Secure Multiparty Computation (MPC)
   - What is the shuffle model
   - Counting bits
   - Robustness in the shuffle model
   - Negative result for the shuffle model
   - Interaction
3) **Streaming/online settings**
   - Private streaming algorithms
   - Privacy under continual observation
4) **Differential privacy as a tool**
   - DP is the enemy of overfitting
   - Application to answering adaptive queries
   - Application to adaptive streaming