Differential privacy without a central database

Boston Differential Privacy Summer School, 6-10 June 2022

Uri Stemmer

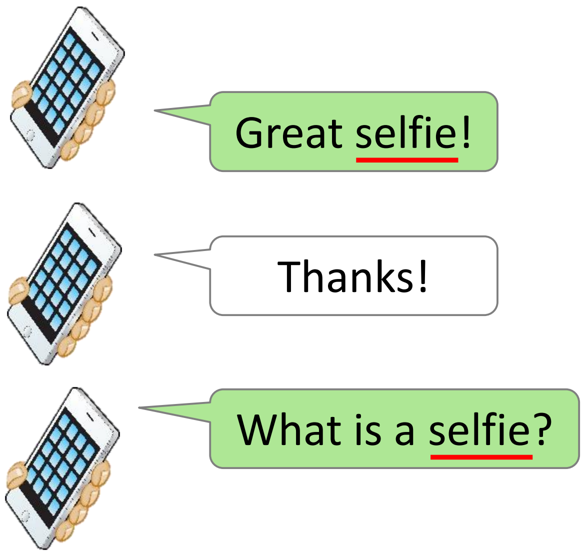
About this course

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

Local Differential Privacy (LDP): Motivation

How can organizations collect high-quality *aggregate* information from their user bases, while guaranteeing that *no individual-specific* information is collected?

Users



Server

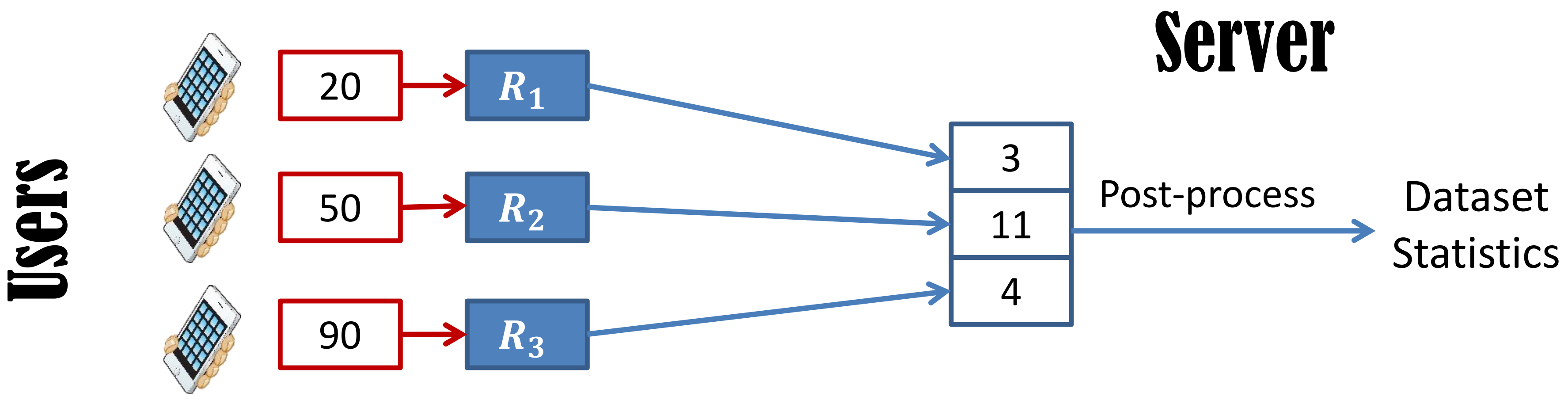
How to learn new words?

- **Identify common "typos" and add them to dictionary!**
- **Privacy concerns?**

Google, **Apple**, **and Microsoft** have been using locally-differentially-private algorithms in the **Chrome browser,** in **iOS-10**, and in **Windows 10**

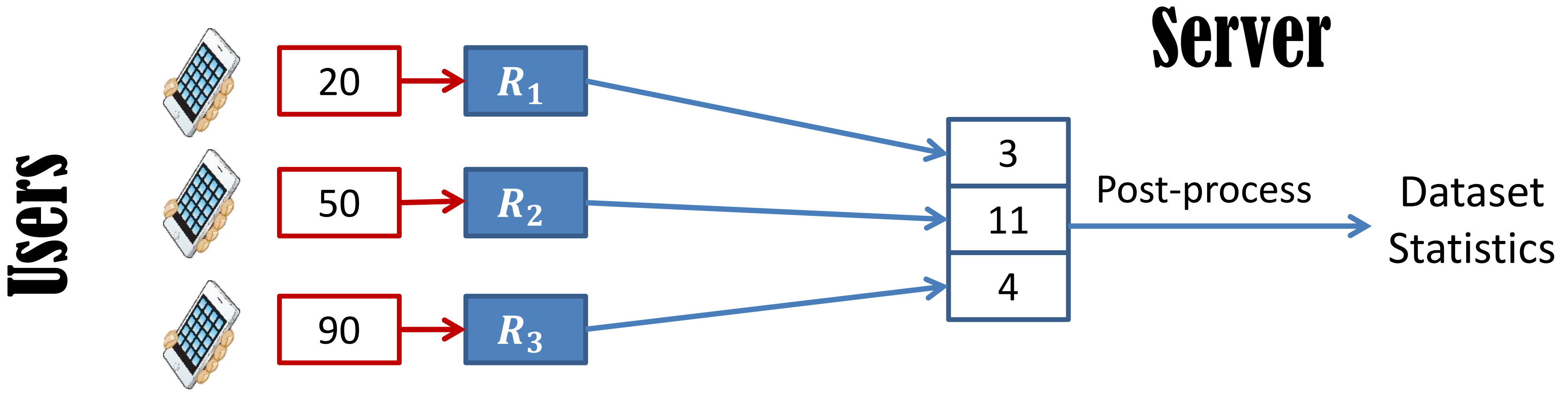
What is Local Differential Privacy?

[Dwork, McSherry, Nissim, Smith 06], [Kasiviswanathan, Lee, Nissim, Raskhodnikova, Smith 08], [Evfimievski, Gehrke, Srikant 03]



What is Local Differential Privacy?

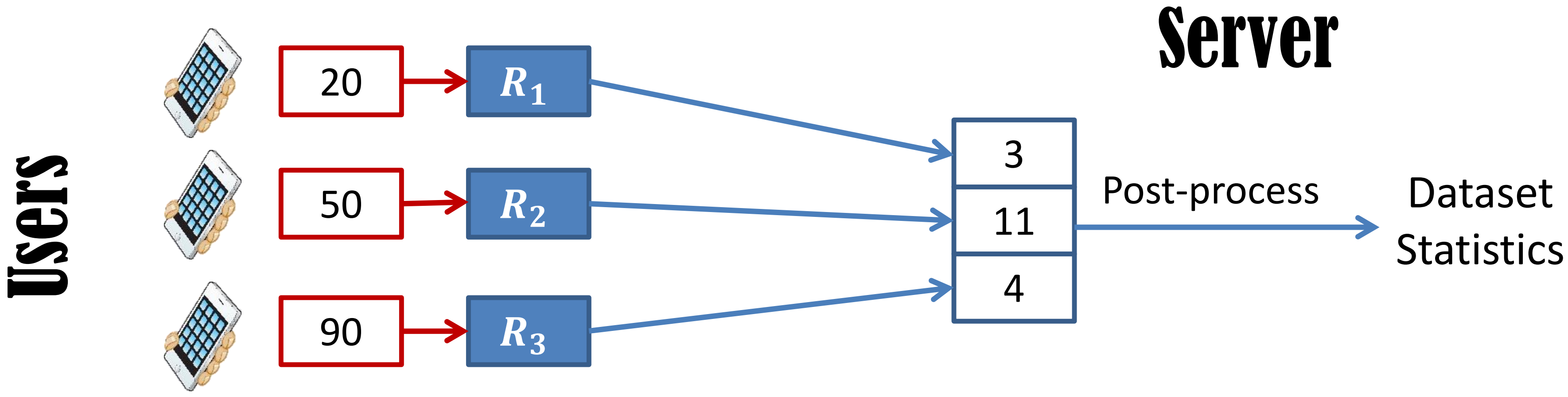
[Dwork, McSherry, Nissim, Smith 06], [Kasiviswanathan, Lee, Nissim, Raskhodnikova, Smith 08], [Evfimievski, Gehrke, Srikant 03]



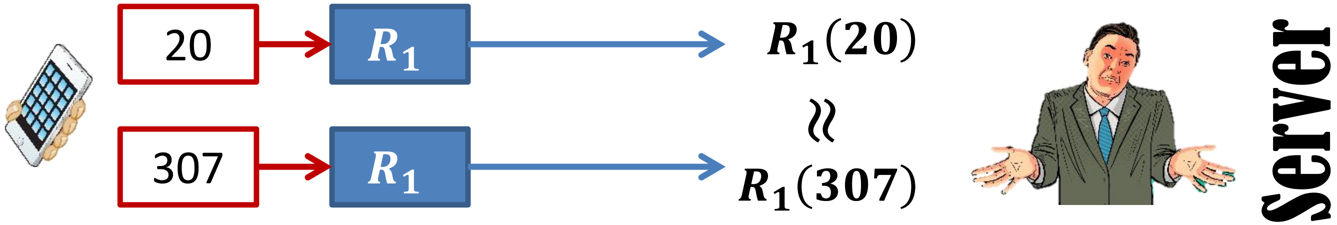
Definition – Local Differential Privacy (simplified)
• ϵ-LDP algorithm accesses every data entry only once, via an ϵ-local randomizer • ϵ-local randomizer is an algorithm $R: X \to Y$ s.t. $\forall x, x' \in X$, $\forall y \in Y$ $$\Pr[R(x) = y] \leq e^{\epsilon} \cdot \Pr[R(x') = y]$$

What is Local Differential Privacy?

[Dwork, McSherry, Nissim, Smith 06], [Kasiviswanathan, Lee, Nissim, Raskhodnikova, Smith 08], [Evfimievski, Gehrke, Srikant 03]

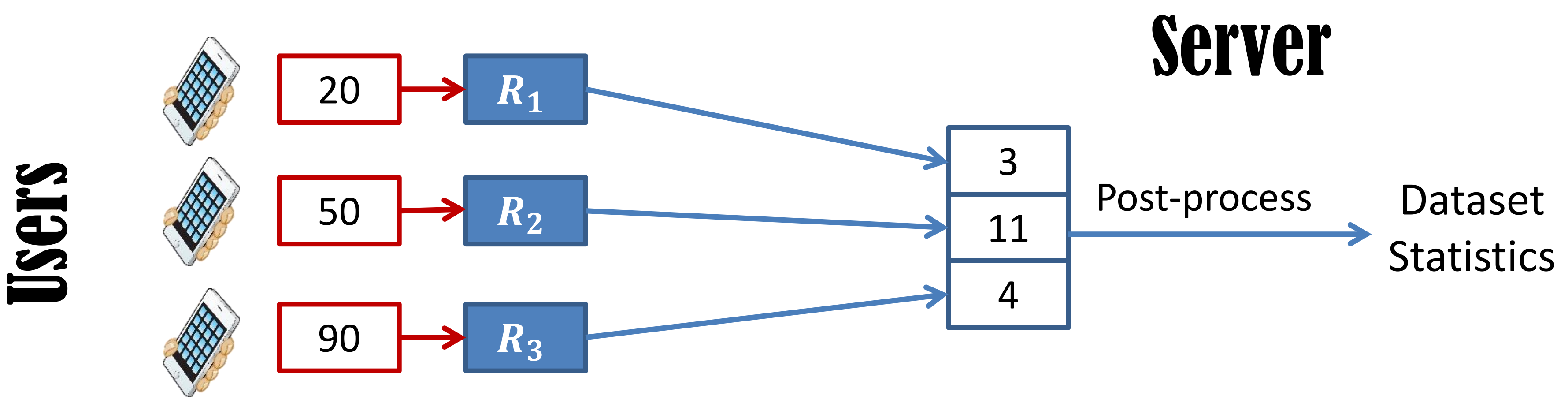


Definition – Local Differential Privacy (simplified)
• ϵ-LDP algorithm accesses every data entry only once, via an ϵ-local randomizer • ϵ-local randomizer is an algorithm $R: X \to Y$ s.t. $\forall x, x' \in X,\ \forall y \in Y$ $$\Pr[R(x) = y] \leq e^{\epsilon} \cdot \Pr[R(x') = y]$$

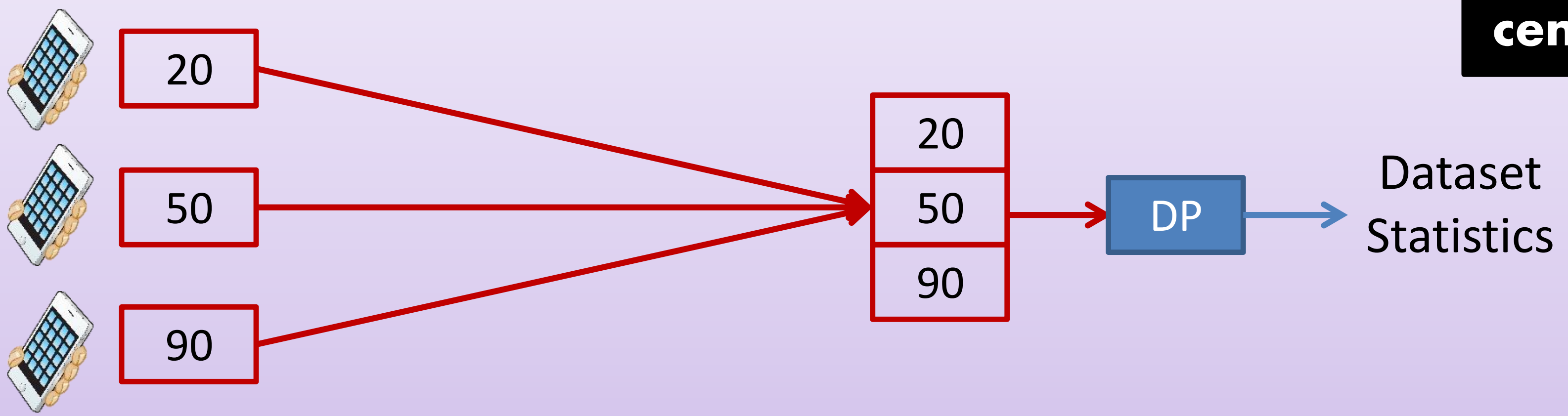


What is Local Differential Privacy?

[Dwork, McSherry, Nissim, Smith 06], [Kasiviswanathan, Lee, Nissim, Raskhodnikova, Smith 08], [Evfimievski, Gehrke, Srikant 03]



Compare to the centralized model



In the standard (centralized) model of DP, we trust the analyzer, and provide privacy against any observer to the <u>outcome</u> of the computation. But the analyzer learns everything

Why use LDP?

- Valuable information about users while providing **strong privacy and trust guarantees**
- Privacy preserved even if the organization is subpoenaed
- Reduces organization liability for securing the data

Challenges

- As every user randomizes her data, accuracy is reduced
- **Number of users might be very large (in the millions)**
- Optimizing runtime and memory usage becomes crucial

The Local Model of Differential Privacy

Today's Outline

1. What is the model?
2. Computing histograms
3. Computing averages
4. Clustering
5. LDP vs. statistical queries
6. Impossibility result for histograms
7. Interactive LDP protocols

Problem Statement: Histograms

- Distributed database $S = (x_1, \dots, x_n) \in X^n$, where user i holds $x_i \in X$
- **Goal:** For every $x \in X$, estimate the multiplicity of x in S, denoted $f_S(x)$

* The error of the protocol is the maximal estimation error

Problem Statement: Histograms

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}) \in \boldsymbol{X^n}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S(x)}$

* The error of the protocol is the maximal estimation error

For example, $\boldsymbol{X}$ could be the set of all (reasonably length) URL domains, and for every user $\boldsymbol{i}$ we have $\boldsymbol{x_i}$ **= homepage address**

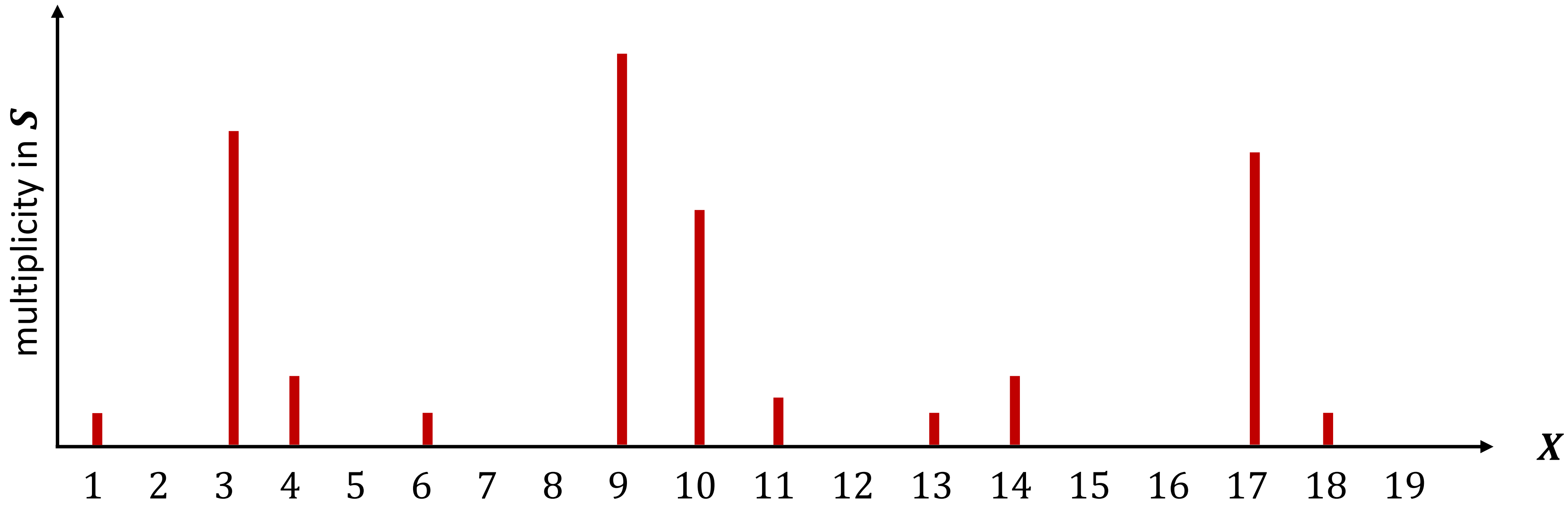
The goal here would be to estimate the popularity of different homepage addresses

Problem Statement: Histograms

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \boldsymbol{X^n}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S(x)}$

* The error of the protocol is the maximal estimation error

In figure:

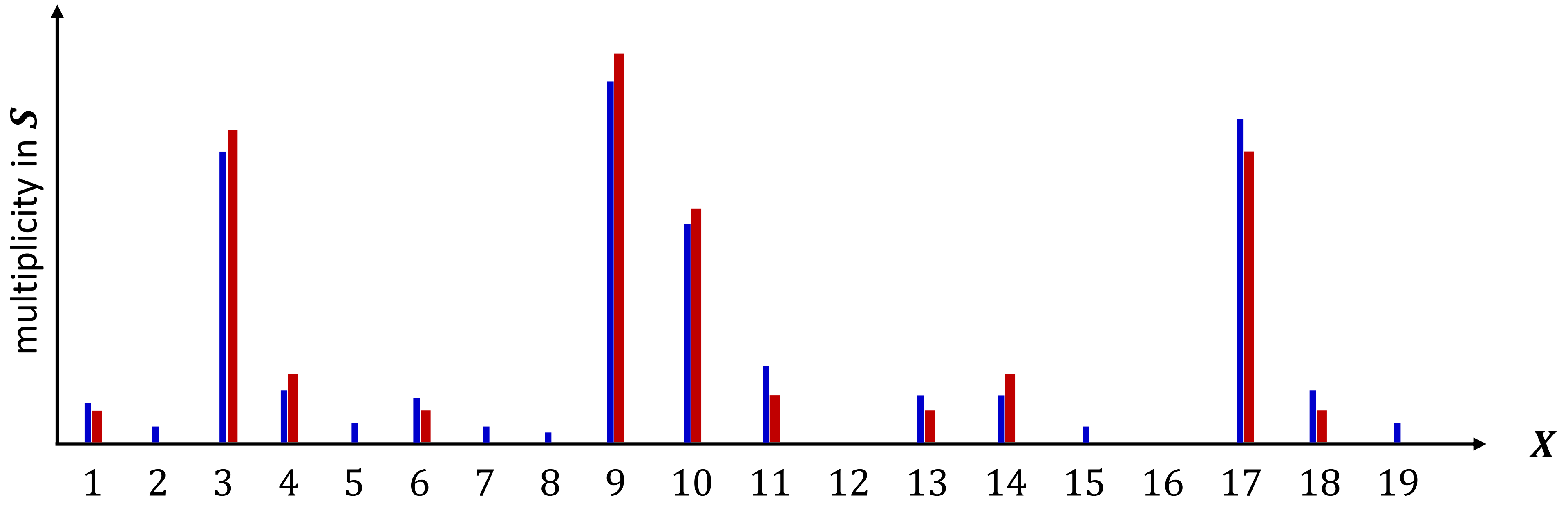


Problem Statement: Histograms

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \boldsymbol{X^n}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S(x)}$

* The error of the protocol is the maximal estimation error

In figure: Want estimations $\hat{f}_S$ s.t. $\max_{x \in X} \left| \hat{f}_S(x) - f_S(x) \right|$ is small

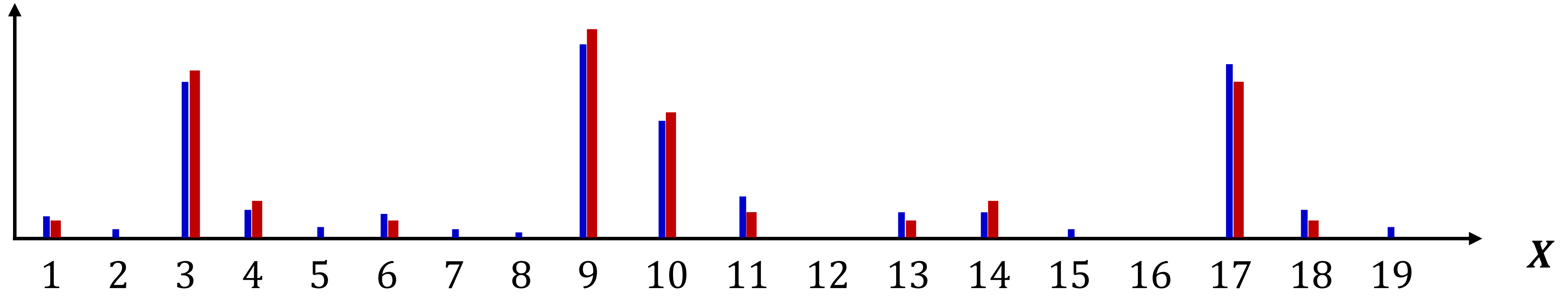


Problem Statement: Histograms

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \boldsymbol{X}^{\boldsymbol{n}}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S(x)}$

* The error of the protocol is the maximal estimation error

In figure: Want estimations $\hat{\boldsymbol{f}}_{\boldsymbol{S}}$ s.t. $\max_{x \in X} \left| \hat{\boldsymbol{f}}_{\boldsymbol{S}}(\boldsymbol{x}) - \boldsymbol{f_S(x)} \right|$ is small

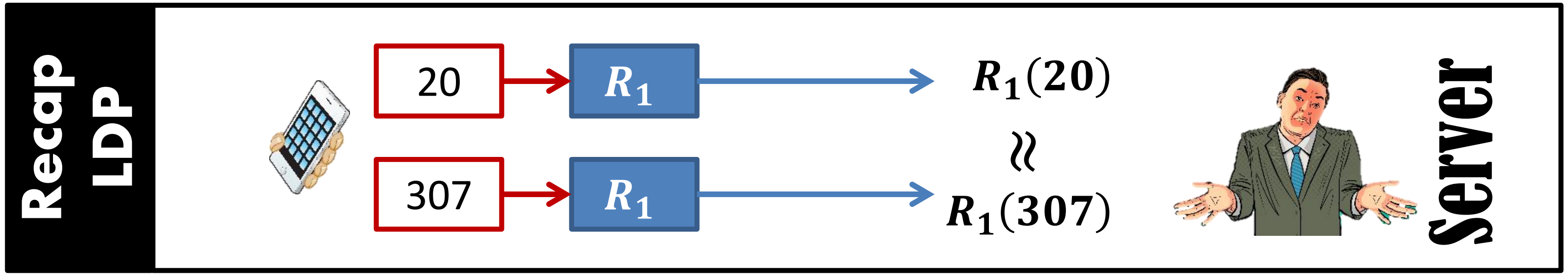


Problem Statement: Histograms

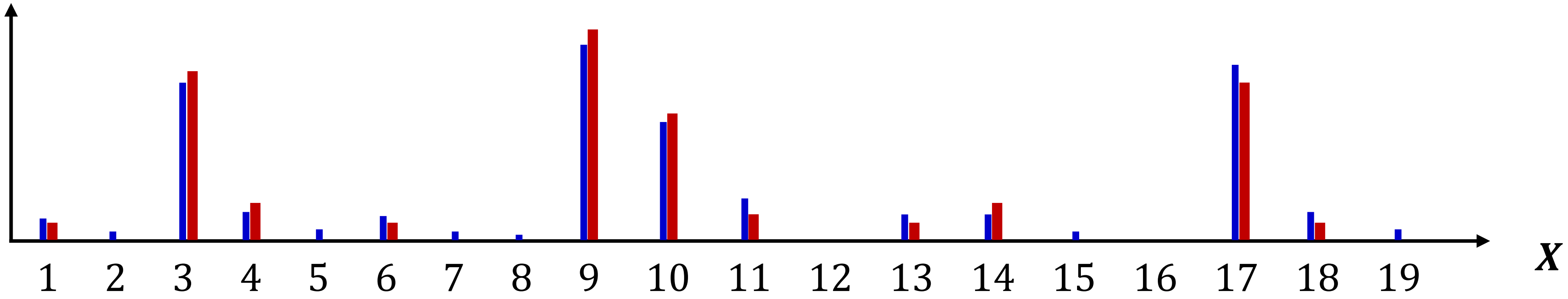
- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \boldsymbol{X^n}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S(x)}$

* The error of the protocol is the maximal estimation error

In figure: Want estimations $\hat{f}_S$ s.t. $\max_{x \in X} \left| \hat{f}_S(x) - f_S(x) \right|$ is small



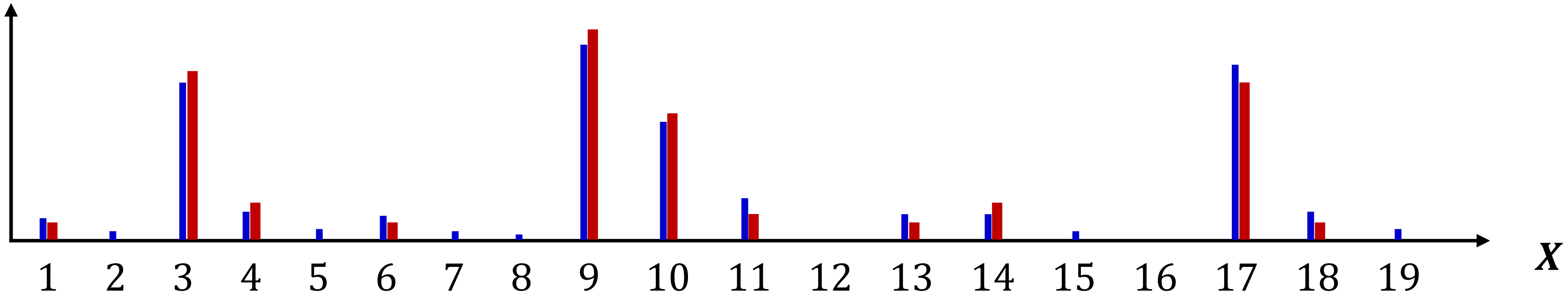
The server learns that many users hold '17', without knowing which are these users!



Why solve under LDP?

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \boldsymbol{X^n}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S(x)}$

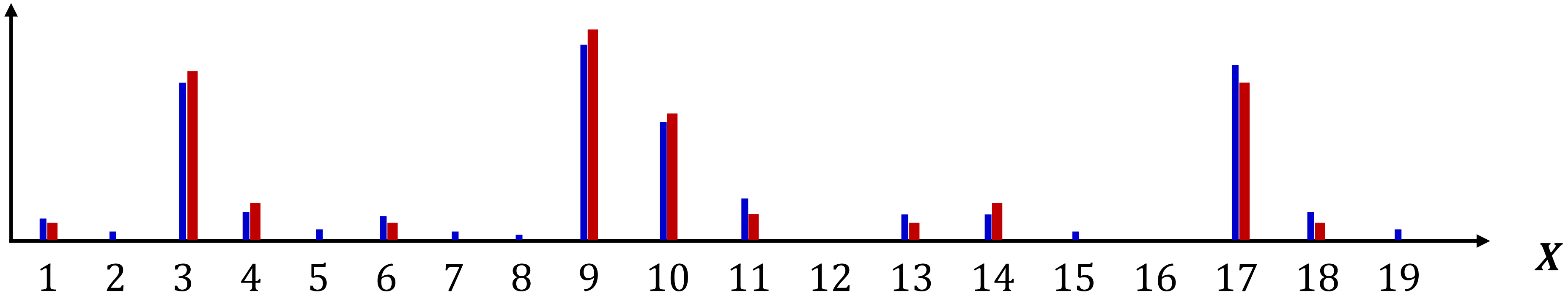
- Arguably the most well-studied problem under LDP, Important subroutine for solving many other problems
  [MS 06], [HKR 12], [EP 14], [BS 15], [QYYKXR 16], [TVVKFSD 17]…
- **Google** and **Apple** have been using using LDP algorithms for this problem in the **Chrome browser** and in **iOS-10:**
  - QuickType suggestions, Emoji suggestions, Lookup Hints, Energy Draining Domains, Autoplay Intent Detection, Crashing Domains, Health Type Usage



Why solve under LDP?

- Distributed database $S = (x_1, \dots, x_n) \in X^n$, where user i holds $x_i \in X$
- **Goal:** For every $x \in X$, estimate the multiplicity of x in S, denoted $f_S(x)$

Observe: If $|X|$ is large, then efficient algorithms cannot output estimations for every $x \in X$ directly



Why solve under LDP?

- Distributed database $S = (x_1, \dots, x_n) \in X^n$, where user i holds $x_i \in X$
- **Goal:** For every $x \in X$, estimate the multiplicity of x in S, denoted $f_S(x)$

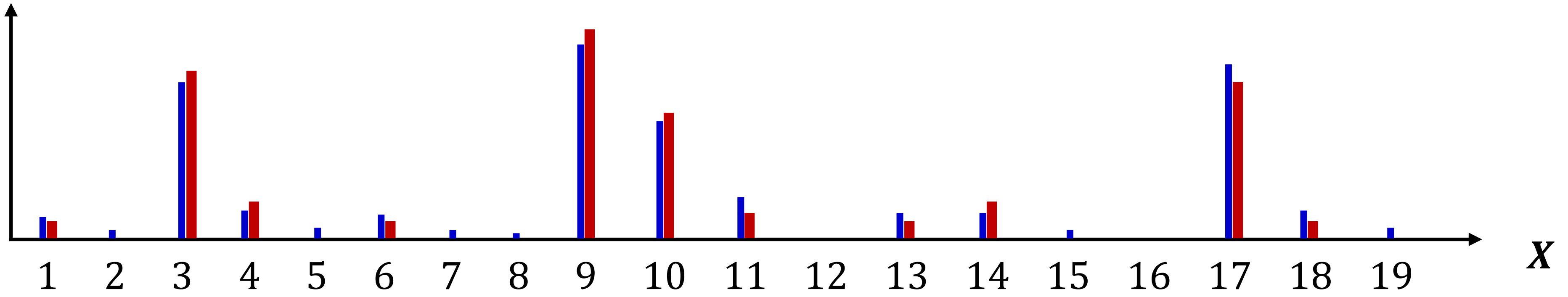
Observe: If $|X|$ is large, then efficient algorithms cannot output estimations for every $x \in X$ directly

Goal 1 – Frequency Oracle:

Frequency oracle is an algorithm that, after communicating with the users, outputs a *data structure* capable of approximating $f_S(x)$ for every $x \in X$

Goal 2 – Heavy Hitters:

Identify a (short) subset $\mathbf{L} \subseteq X$ of "heavy-hitters" with estimates for their frequencies (the frequency of every $x \notin L$ is estimated as $\mathbf{0}$)



Why solve under LDP?

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \boldsymbol{X^n}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S(x)}$

Observe: If $|\boldsymbol{X}|$ is large, then efficient algorithms cannot output estimations for every $\boldsymbol{x} \in \boldsymbol{X}$ directly

Goal 1 – Frequency Oracle:

Frequency oracle is an algorithm that, after communicating with the users, outputs a *data structure* capable of approximating $\boldsymbol{f_S(x)}$ for every $\boldsymbol{x} \in \boldsymbol{X}$

Goal 2 – Heavy Hitters:

Identify a (short) subset $\mathbf{L} \subseteq \boldsymbol{X}$ of "heavy-hitters" with estimates for their frequencies (the frequency of every $\boldsymbol{x} \notin \boldsymbol{L}$ is estimated as $\mathbf{0}$)

- Heavy-hitters is a particular kind of a frequency oracle, so it might be harder to obtain
- Ignoring runtime, the two goals are equivalent

- **What's next?** **(1)** Show a reduction from Goal 2 to Goal 1
  (2) Show how to achieve Goal 1

Part 1: Use Oracle to identify Heavy-Hitters

- Distributed database $S = (x_1, \ldots, x_n) \in X^n$, where user i holds $x_i \in X$
- **Goal:** For every $x \in X$, estimate the multiplicity of x in S, denoted $f_S(x)$

Thm: If there is an ε-LDP frequency oracle with error τ then there is an $O(\varepsilon)$-LDP algorithm for heavy-hitters with error $O(\tau)$ with almost the same runtime, space, and communication complexities

Easier Thm: If there is an efficient ε-LDP frequency oracle with error τ then there is an efficient $\varepsilon \cdot \log|X|$-LDP algorithm for heavy-hitters with error 2τ

Proof of easier theorem

1) Let $\mathbb{O}$ be an $\boldsymbol{\varepsilon}$-LDP frequency oracle with error $\boldsymbol{\tau}$
2) There are $\boldsymbol{n}$ users where every user $\boldsymbol{i} \in [\boldsymbol{n}]$ holds an input $\boldsymbol{x_i} \in \boldsymbol{X}$. Denote $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$

Proof of easier theorem

1) Let $\mathbb{O}$ be an $\boldsymbol{\varepsilon}$-LDP frequency oracle with error $\boldsymbol{\tau}$
2) There are $\boldsymbol{n}$ users where every user $\boldsymbol{i} \in [\boldsymbol{n}]$ holds an input $\boldsymbol{x_i} \in \boldsymbol{X}$. Denote $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$
3) Let $\boldsymbol{h}: \boldsymbol{X} \to [\boldsymbol{T}]$ be a publicly known hash function

   Intuition: if $\boldsymbol{h}$ isolates heavy-hitters then suffices to query $\mathbb{O}$ on hash range. But how?

Simplifying assumption: No collisions in $\boldsymbol{h}$ for elements in $\boldsymbol{S}$

Proof of easier theorem

1) Let $\mathbb{O}$ be an $\boldsymbol{\varepsilon}$-LDP frequency oracle with error $\boldsymbol{\tau}$
2) There are $\boldsymbol{n}$ users where every user $\boldsymbol{i} \in [\boldsymbol{n}]$ holds an input $\boldsymbol{x_i} \in \boldsymbol{X}$. Denote $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$
3) Let $\boldsymbol{h}: \boldsymbol{X} \to [\boldsymbol{T}]$ be a publicly known hash function

   Intuition: if $\boldsymbol{h}$ isolates heavy-hitters then suffices to query $\mathbb{O}$ on hash range. But how?

Simplifying assumption: No collisions in $\boldsymbol{h}$ for elements in $\boldsymbol{S}$

Fix a "heavy-hitter" $\boldsymbol{x}^*$ satisfying $\boldsymbol{f_S}(\boldsymbol{x}^*) > \boldsymbol{2\tau}$, and denote $\boldsymbol{t}^* = \boldsymbol{h}(\boldsymbol{x}^*)$ % We want to identify $\boldsymbol{x}^*$

Proof of easier theorem

1) Let $\mathbb{O}$ be an $\boldsymbol{\varepsilon}$-LDP frequency oracle with error $\boldsymbol{\tau}$
2) There are $\boldsymbol{n}$ users where every user $\boldsymbol{i} \in [\boldsymbol{n}]$ holds an input $\boldsymbol{x_i} \in \boldsymbol{X}$. Denote $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$
3) Let $\boldsymbol{h}\colon \boldsymbol{X} \to [\boldsymbol{T}]$ be a publicly known hash function

   Intuition: if $\boldsymbol{h}$ isolates heavy-hitters then suffices to query $\mathbb{O}$ on hash range. But how?

Simplifying assumption: No collisions in $\boldsymbol{h}$ for elements in $\boldsymbol{S}$

Fix a "heavy-hitter" $\boldsymbol{x}^*$ satisfying $\boldsymbol{f_S}(\boldsymbol{x}^*) > \boldsymbol{2\tau}$, and denote $\boldsymbol{t}^* = \boldsymbol{h}(\boldsymbol{x}^*)$ % We want to identify $\boldsymbol{x}^*$

For $\ell \in [\log|X|]$ define $\boldsymbol{S_\ell} = (\boldsymbol{h}(\boldsymbol{x_i}), \boldsymbol{x_i}[\ell])_{i\in[n]}$, where $\boldsymbol{x_i}[\ell]$= bit ℓ of $\boldsymbol{x_i}$ % Users compute rows locally

Proof of easier theorem

1) Let $\mathbb{O}$ be an $\boldsymbol{\varepsilon}$-LDP frequency oracle with error $\boldsymbol{\tau}$
2) There are $\boldsymbol{n}$ users where every user $\boldsymbol{i} \in [\boldsymbol{n}]$ holds an input $\boldsymbol{x_i} \in \boldsymbol{X}$. Denote $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$
3) Let $\boldsymbol{h}: \boldsymbol{X} \to [\boldsymbol{T}]$ be a publicly known hash function

   Intuition: if $\boldsymbol{h}$ isolates heavy-hitters then suffices to query $\mathbb{O}$ on hash range. But how?

Simplifying assumption: No collisions in $\boldsymbol{h}$ for elements in $\boldsymbol{S}$

Fix a "heavy-hitter" $\boldsymbol{x}^*$ satisfying $\boldsymbol{f_S}(\boldsymbol{x}^*) > \boldsymbol{2\tau}$, and denote $\boldsymbol{t}^* = \boldsymbol{h}(\boldsymbol{x}^*)$ % We want to identify $\boldsymbol{x}^*$

For $\ell \in [\log|X|]$ define $\boldsymbol{S}_\ell = (\boldsymbol{h}(\boldsymbol{x_i}), \boldsymbol{x_i}[\ell])_{\boldsymbol{i} \in [\boldsymbol{n}]}$, where $\boldsymbol{x_i}[\ell]$= bit ℓ of $\boldsymbol{x_i}$ % Users compute rows locally

- $\boldsymbol{x}^*$ is "heavy", hence $(\boldsymbol{t}^*, \boldsymbol{x}^*[\ell])$ appears $> \boldsymbol{2\tau}$ in $\boldsymbol{S}_\ell$ for every ℓ
- No collisions, hence $(\boldsymbol{t}^*, \boldsymbol{1} - \boldsymbol{x}^*[\ell])$ appears **0** times in $\boldsymbol{S}_\ell$
- $\Longrightarrow$ Can identify every bit ℓ of $\boldsymbol{x}^*$ by querying $\mathbb{O}(\boldsymbol{S}_\ell)$ on $(\boldsymbol{t}^*, \boldsymbol{0})$ and $(\boldsymbol{t}^*, \boldsymbol{1})$

Proof of easier theorem

1) Let $\mathbb{O}$ be an $\boldsymbol{\varepsilon}$-LDP frequency oracle with error $\boldsymbol{\tau}$
2) There are $\boldsymbol{n}$ users where every user $\boldsymbol{i} \in [\boldsymbol{n}]$ holds an input $\boldsymbol{x_i} \in \boldsymbol{X}$. Denote $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$
3) Let $\boldsymbol{h}: \boldsymbol{X} \to [\boldsymbol{T}]$ be a publicly known hash function

   Intuition: if $\boldsymbol{h}$ isolates heavy-hitters then suffices to query $\mathbb{O}$ on hash range. But how?

Simplifying assumption: No collisions in $\boldsymbol{h}$ for elements in $\boldsymbol{S}$

Fix a "heavy-hitter" $\boldsymbol{x}^*$ satisfying $\boldsymbol{f_S}(\boldsymbol{x}^*) > \mathbf{2}\boldsymbol{\tau}$, and denote $\boldsymbol{t}^* = \boldsymbol{h}(\boldsymbol{x}^*)$ % We want to identify $\boldsymbol{x}^*$

For $\ell \in [\log|X|]$ define $\boldsymbol{S}_\ell = (\boldsymbol{h}(\boldsymbol{x_i}), \boldsymbol{x_i}[\ell])_{i\in[n]}$, where $\boldsymbol{x_i}[\ell]$= bit ℓ of $\boldsymbol{x_i}$ % Users compute rows locally

- $\boldsymbol{x}^*$ is "heavy", hence $(\boldsymbol{t}^*, \boldsymbol{x}^*[\ell])$ appears $> \mathbf{2}\boldsymbol{\tau}$ in $\boldsymbol{S}_\ell$ for every ℓ
- No collisions, hence $(\boldsymbol{t}^*, \mathbf{1} - \boldsymbol{x}^*[\ell])$ appears $\mathbf{0}$ times in $\boldsymbol{S}_\ell$
- $\Longrightarrow$ Can identify every bit ℓ of $\boldsymbol{x}^*$ by querying $\mathbb{O}(\boldsymbol{S}_\ell)$ on $(\boldsymbol{t}^*, \mathbf{0})$ and $(\boldsymbol{t}^*, \mathbf{1})$

The Protocol: For every $\boldsymbol{t} \in [\boldsymbol{T}]$ construct $\widehat{\boldsymbol{x}}^{(t)}$ as follows:
$\forall \ell \in [\log|X|]$ query $\mathbb{O}(\boldsymbol{S}_\ell)$ on $(\boldsymbol{t}, \mathbf{0})$ and $(\boldsymbol{t}, \mathbf{1})$ and set $\widehat{\boldsymbol{x}}^{(t)}[\ell] \leftarrow \mathbf{argmax}$

By purple, $\widehat{\boldsymbol{x}}^{(t^*)} = \boldsymbol{x}^*$ is identified

- The algorithm returns a list of size T containing all elements x with $f_S(x) \geq 2\tau$
- For our simplifying assumption, suffices to take $T \gtrsim n^2$
- $\Longrightarrow$ Total runtime $\approx n^2$ times the response time of $\mathbb{O}$ (can do better)
- What about privacy? We had $\log|X|$ executions of $\mathbb{O}$
- $\Longrightarrow$ Overall $\varepsilon \cdot \log|X|$-DP by composition

Simplifying assumption: No collisions in h for elements in S

Fix a "heavy-hitter" x^* satisfying $f_S(x^*) > 2\tau$, and denote $t^* = h(x^*)$ % We want to identify x^*

For $\ell \in [\log|X|]$ define $S_\ell = (h(x_i), x_i[\ell])_{i \in [n]}$, where $x_i[\ell]$= bit ℓ of x_i % Users compute rows locally

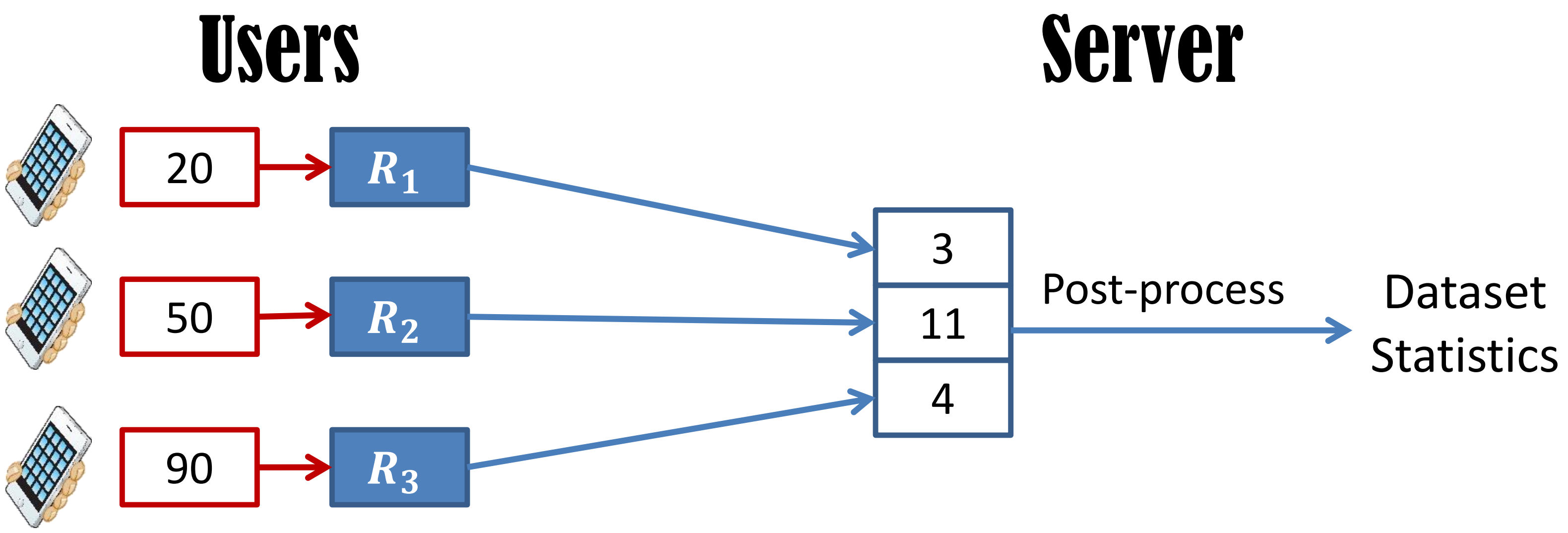
- x^* is "heavy", hence $(t^*, x^*[\ell])$ appears $> 2\tau$ in S_ℓ for every ℓ
- No collisions, hence $(t^*, 1 - x^*[\ell])$ appears 0 times in S_ℓ
- $\Longrightarrow$ Can identify every bit ℓ of x^* by querying $\mathbb{O}(S_\ell)$ on $(t^*, 0)$ and $(t^*, 1)$

The Protocol: For every $t \in [T]$ construct $\hat{x}^{(t)}$ as follows:
$\forall \ell \in [\log|X|]$ query $\mathbb{O}(S_\ell)$ on $(t, 0)$ and $(t, 1)$ and set $\hat{x}^{(t)}[\ell] \leftarrow \mathbf{argmax}$

By purple, $\hat{x}^{(t^*)} = x^*$ is identified

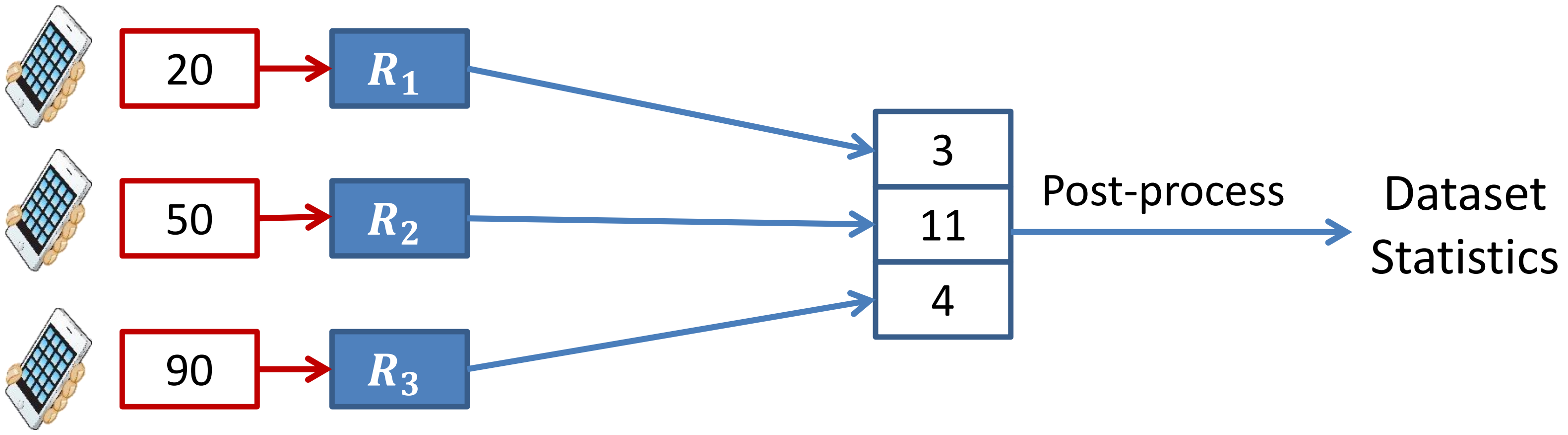
NEXT GOAL: DESIGN A FREQUENCY ORACLE

STEP BACK:
How can LDP be useful at all?



Simple Case: Oracle for X=$\{\pm 1\}$

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \{\pm\mathbf{1}\}^{\boldsymbol{n}}$, each user holds a bit
- **Goal:** Estimate number of ones, denoted $\boldsymbol{f_S}(\mathbf{1})$

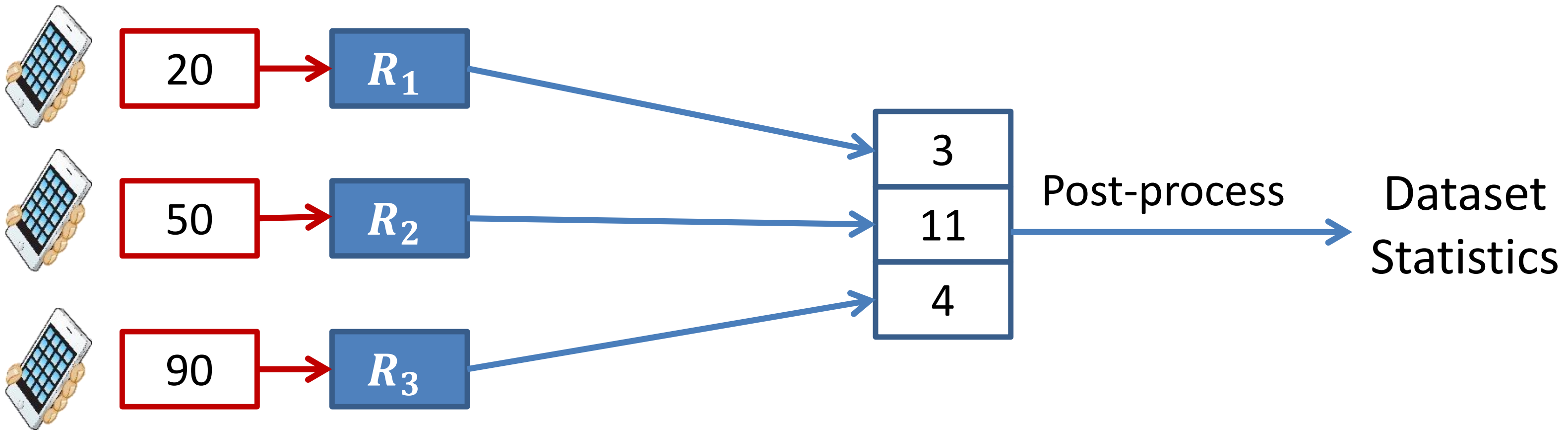


Simple Case: Oracle for X={±1}

- Distributed database $S = (x_1, \dots, x_n) \in \{\pm 1\}^n$, each user holds a bit
- **Goal:** Estimate number of ones, denoted $f_S(1)$

Local randomizer – randomized response $R(x)$: [Warner 1965]

Return x w.p. $\approx \frac{1}{2} + \epsilon$ (and -x otherwise)



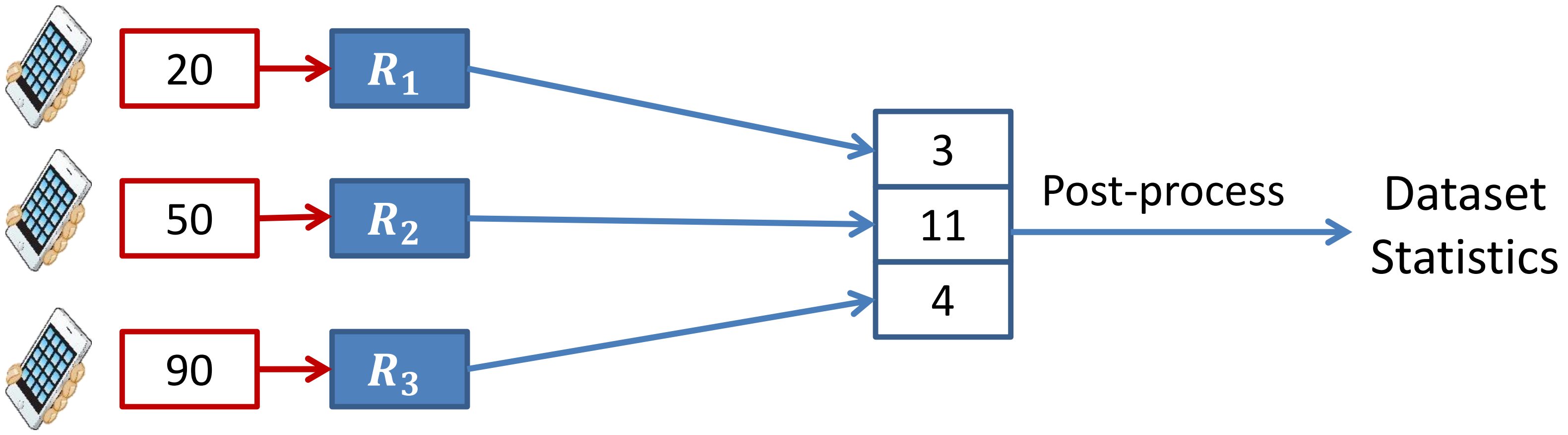
Simple Case: Oracle for X=$\{\pm 1\}$

- Distributed database $S = (x_1, \ldots, x_n) \in \{\pm 1\}^n$, each user holds a bit
- **Goal:** Estimate number of ones, denoted $f_S(1)$

Local randomizer – randomized response $R(x)$: [Warner 1965]

Return x w.p. $\approx \frac{1}{2} + \epsilon$ (and -x otherwise)

Observe: Any output (± 1) is almost as equally likely to result from any input (± 1)



Simple Case: Oracle for X=$\{\pm 1\}$

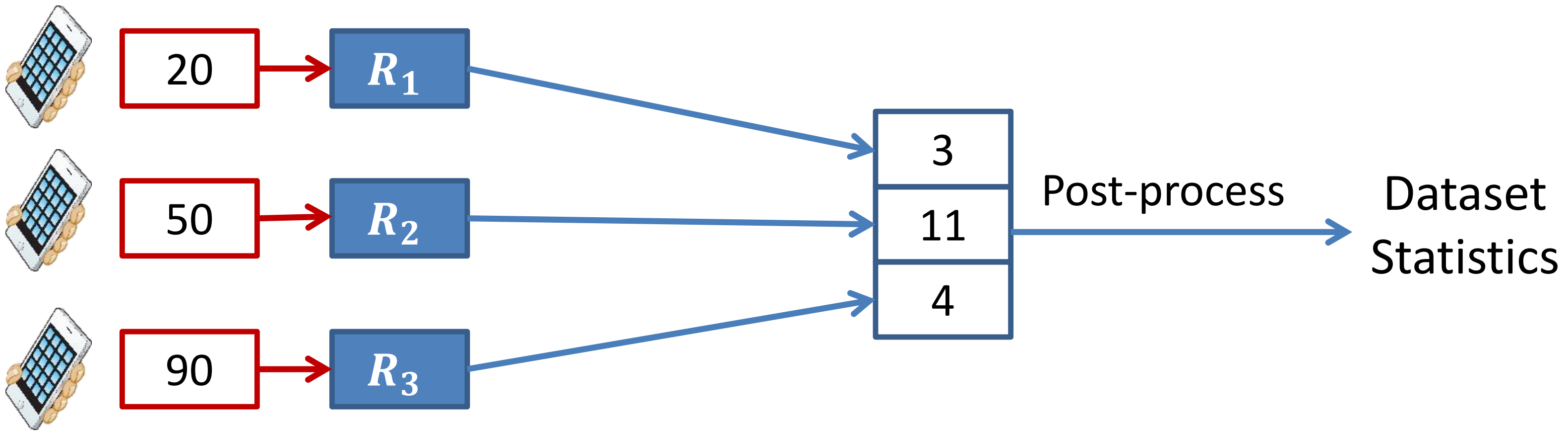
- Distributed database $S = (x_1, \ldots, x_n) \in \{\pm 1\}^n$, each user holds a bit
- **Goal:** Estimate number of ones, denoted $f_S(1)$

Local randomizer – randomized response $R(x)$: [Warner 1965]

Return x w.p. $\approx \frac{1}{2} + \epsilon$ (and -x otherwise)

Observe: Any output (± 1) is almost as equally likely to result from any input (± 1)

Protocol: From every user i obtain $y_i \leftarrow R(x_i)$. Return $\frac{1}{4\epsilon}\left(2\epsilon n + \sum_i y_i\right)$



Simple Case: Oracle for X=$\{\pm 1\}$

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \{\pm \mathbf{1}\}^{\boldsymbol{n}}$, each user holds a bit
- **Goal:** Estimate number of ones, denoted $\boldsymbol{f_S}(\mathbf{1})$

Local randomizer – randomized response $\boldsymbol{R}(\boldsymbol{x})$: [Warner 1965]

Return $\boldsymbol{x}$ w.p. $\approx \frac{\mathbf{1}}{\mathbf{2}} + \boldsymbol{\epsilon}$ (and -$\boldsymbol{x}$ otherwise)

Observe: Any output (± 1) is almost as equally likely to result from any input (± 1)

Protocol: From every user $\boldsymbol{i}$ obtain $\boldsymbol{y_i} \leftarrow \boldsymbol{R}(\boldsymbol{x_i})$. Return $\frac{\mathbf{1}}{\mathbf{4}\boldsymbol{\epsilon}}\left(\mathbf{2}\boldsymbol{\epsilon n} + \sum_i \boldsymbol{y_i}\right)$

Analysis:

Simple Case: Oracle for X=$\{\pm 1\}$

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \{\pm\mathbf{1}\}^{\boldsymbol{n}}$, each user holds a bit
- **Goal:** Estimate number of ones, denoted $\boldsymbol{f_S}(\mathbf{1})$

Local randomizer – randomized response $\boldsymbol{R}(\boldsymbol{x})$: [Warner 1965]

Return $\boldsymbol{x}$ w.p. $\approx \frac{1}{2} + \epsilon$ (and -$\boldsymbol{x}$ otherwise)

Observe: Any output (± 1) is almost as equally likely to result from any input (± 1)

Protocol: From every user $\boldsymbol{i}$ obtain $\boldsymbol{y_i} \leftarrow \boldsymbol{R}(\boldsymbol{x_i})$. Return $\frac{1}{4\epsilon}\left(2\epsilon n + \sum_i y_i\right)$

Analysis: $\mathbb{E}\left[\sum_i y_i\right] = \sum_{i:x_i=1} \mathbb{E}[y_i] + \sum_{i:x_i=-1} \mathbb{E}[y_i]$ $= (-1)\cdot\left(\frac{1}{2}+\epsilon\right) + 1\cdot\left(\frac{1}{2}-\epsilon\right) = -2\epsilon$

$$= f_S(1)\cdot 2\epsilon - f_S(-1)\cdot 2\epsilon = f_S(1)\cdot 4\epsilon - n\cdot 2\epsilon$$

Hoeffding: w.h.p., estimation error at most $\approx \frac{1}{\varepsilon}\sqrt{n}$

Simple Case: Oracle for X=$\{\pm 1\}$

- Distributed database $S = (x_1, \dots, x_n) \in \{\pm 1\}^n$, each user holds a bit
- **Goal:** Estimate number of ones, denoted $f_S(1)$

Local randomizer – randomized response $R(x)$: [Warner 1965]

Return x w.p. $\approx \frac{1}{2} + \epsilon$ (and -x otherwise)

Observe: Any output (± 1) is almost as equally likely to result from any input (± 1)

Protocol: From every user i obtain $y_i \leftarrow R(x_i)$. Return $\frac{1}{4\epsilon}\left(2\epsilon n + \sum_i y_i\right)$

Analysis: $\mathbb{E}\left[\sum_i y_i\right] = \sum_{i:x_i=1} \mathbb{E}[y_i] + \sum_{i:x_i=-1} \mathbb{E}[y_i]$

$= (-1) \cdot \left(\frac{1}{2} + \epsilon\right) + 1 \cdot \left(\frac{1}{2} - \epsilon\right) = -2\epsilon$

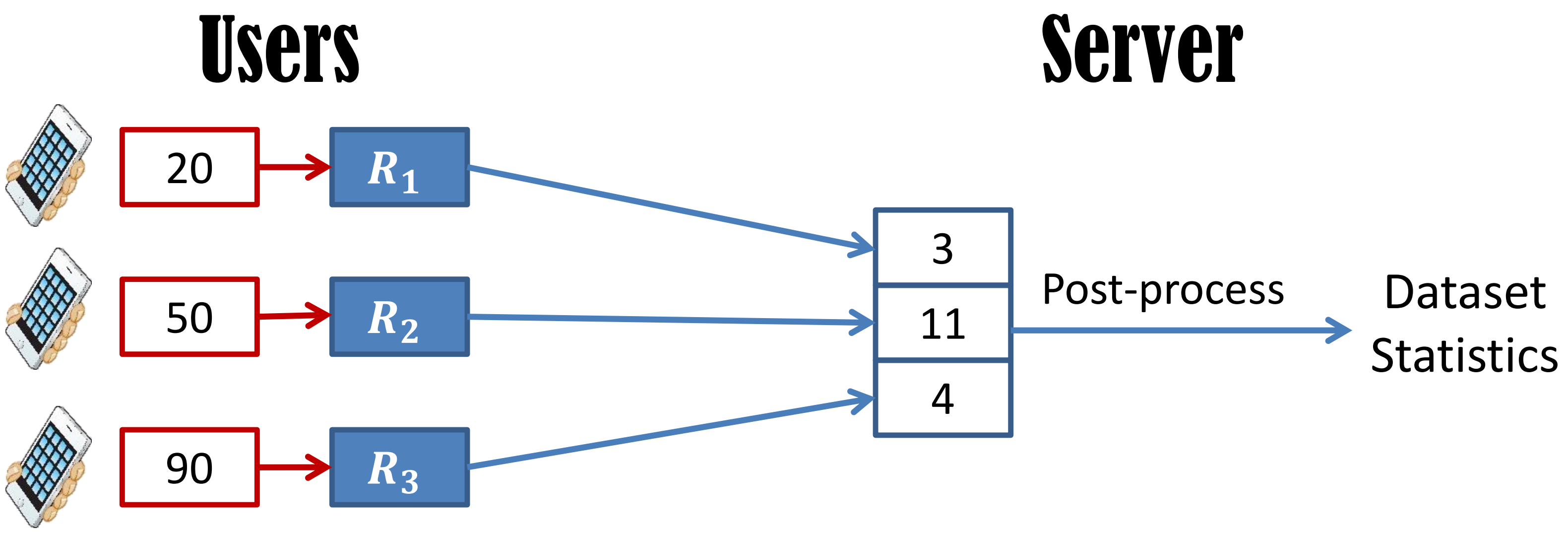
$$= f_S(1) \cdot 2\epsilon - f_S(-1) \cdot 2\epsilon = f_S(1) \cdot 4\epsilon - n \cdot 2\epsilon$$

Hoeffding: w.h.p., estimation error at most $\approx \frac{1}{\varepsilon}\sqrt{n}$

Takeaway: Counting bits under LDP is easy

GENERAL CASE:

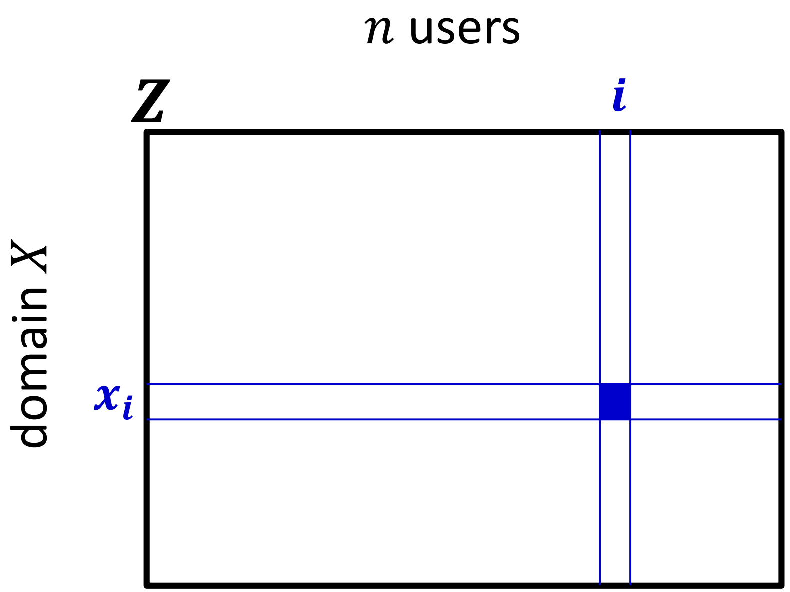
Frequency Oracle for a large domain X



General Case: Oracle for a large domain X

Setting:

- Every user $\boldsymbol{i}$ holds a value $\boldsymbol{x_i} \in \boldsymbol{X}$
- Public uniform matrix $\boldsymbol{Z} \in \{\pm\mathbf{1}\}^{|\boldsymbol{X}|\times \boldsymbol{n}}$
  $\forall \boldsymbol{i} \in [\boldsymbol{n}]$ and $\forall \boldsymbol{x} \in \boldsymbol{X}$ we have a bit $\boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]$
- User $\boldsymbol{i}$ identifies corresponding bit $\boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$



General Case: Oracle for a large domain X

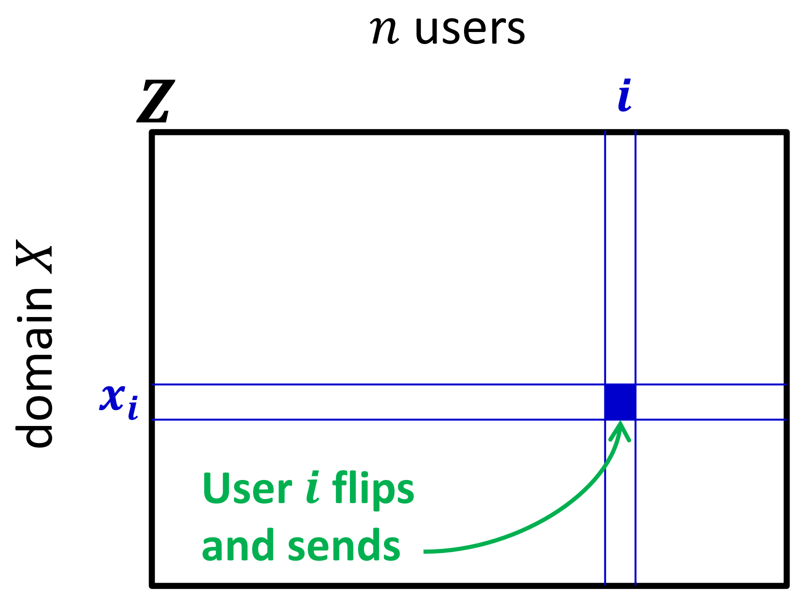
Setting:

- Every user $\boldsymbol{i}$ holds a value $\boldsymbol{x_i} \in \boldsymbol{X}$
- Public uniform matrix $\boldsymbol{Z} \in \{\pm\mathbf{1}\}^{|\boldsymbol{X}| \times \boldsymbol{n}}$
  $\forall \boldsymbol{i} \in [\boldsymbol{n}]$ and $\forall \boldsymbol{x} \in \boldsymbol{X}$ we have a bit $\boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]$
- User $\boldsymbol{i}$ identifies corresponding bit $\boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$

Users randomize their corresponding bits:

User $\boldsymbol{i}$ sends $\boldsymbol{y_i} = \boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$ w.p. $\approx \frac{1}{2} + \epsilon$

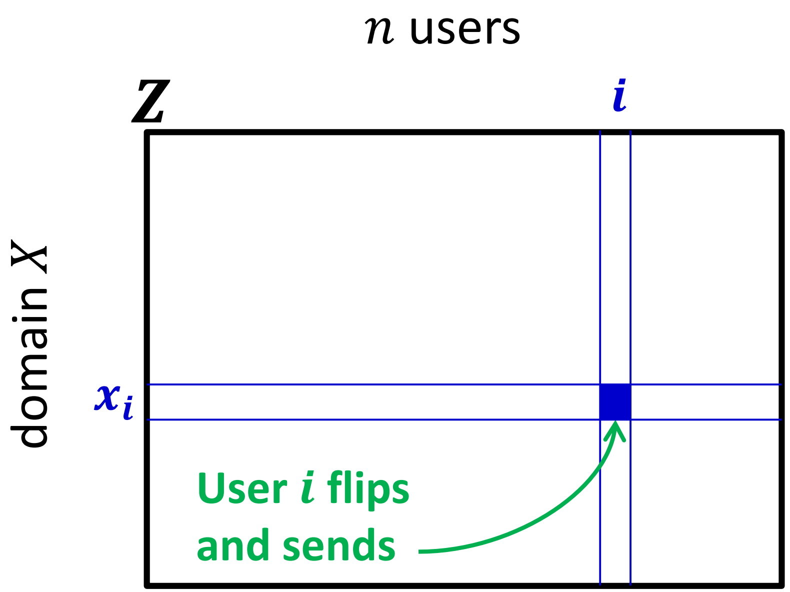
$\boldsymbol{y_i} = -\boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$ w.p. $\approx \frac{1}{2} - \epsilon$



General Case: Oracle for a large domain X

Setting:

- Every user $\boldsymbol{i}$ holds a value $\boldsymbol{x_i} \in \boldsymbol{X}$
- Public uniform matrix $\boldsymbol{Z} \in \{\pm\mathbf{1}\}^{|\boldsymbol{X}|\times \boldsymbol{n}}$
  $\forall \boldsymbol{i} \in [\boldsymbol{n}]$ and $\forall \boldsymbol{x} \in \boldsymbol{X}$ we have a bit $\boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]$
- User $\boldsymbol{i}$ identifies corresponding bit $\boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$



Users randomize their corresponding bits:

User $\boldsymbol{i}$ sends $\boldsymbol{y_i} = \boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$ w.p. $\approx \frac{\mathbf{1}}{\mathbf{2}} + \boldsymbol{\epsilon}$

$\boldsymbol{y_i} = -\boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$ w.p. $\approx \frac{\mathbf{1}}{\mathbf{2}} - \boldsymbol{\epsilon}$

Server: Given a query $\boldsymbol{x} \in \boldsymbol{X}$, return $\hat{\boldsymbol{f}}(\boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{2}\boldsymbol{\epsilon}} \sum_{\boldsymbol{i}\in[\boldsymbol{n}]} \boldsymbol{y_i} \cdot \boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]$

General Case: Oracle for a large domain X

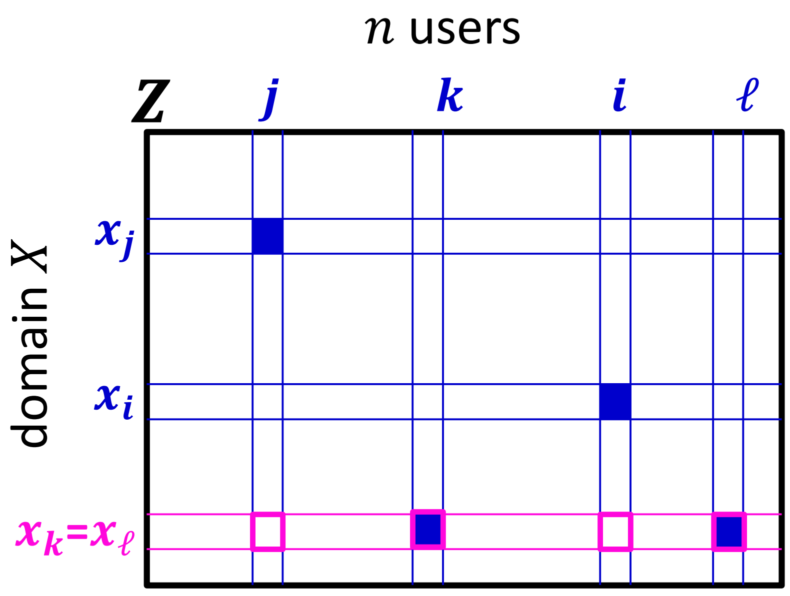
Setting:

- Every user $\boldsymbol{i}$ holds a value $\boldsymbol{x_i} \in \boldsymbol{X}$
- Public uniform matrix $\boldsymbol{Z} \in \{\pm \mathbf{1}\}^{|\boldsymbol{X}| \times \boldsymbol{n}}$
  $\forall \boldsymbol{i} \in [\boldsymbol{n}]$ and $\forall \boldsymbol{x} \in \boldsymbol{X}$ we have a bit $\boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]$
- User $\boldsymbol{i}$ identifies corresponding bit $\boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$

Users randomize their corresponding bits:

User $\boldsymbol{i}$ sends $\boldsymbol{y_i} = \boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$ w.p. $\approx \frac{1}{2} + \boldsymbol{\epsilon}$

$\boldsymbol{y_i} = -\boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$ w.p. $\approx \frac{1}{2} - \boldsymbol{\epsilon}$

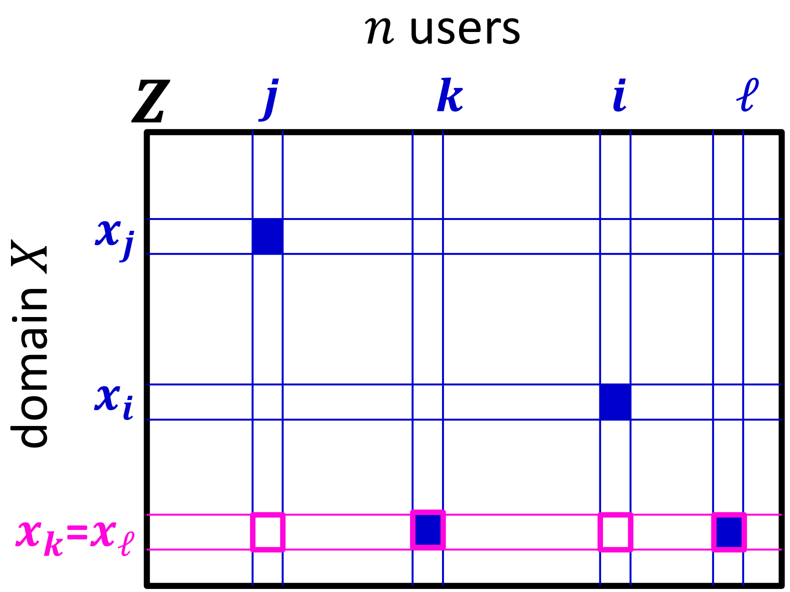


Server: Given a query $\boldsymbol{x} \in \boldsymbol{X}$, return $\hat{\boldsymbol{f}}(\boldsymbol{x}) = \frac{1}{2\epsilon} \sum_{i \in [n]} \boldsymbol{y_i} \cdot \boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]$

General Case: Oracle for a large domain X

Setting:

- Every user $\boldsymbol{i}$ holds a value $\boldsymbol{x_i} \in \boldsymbol{X}$
- Public uniform matrix $\boldsymbol{Z} \in \{\pm\boldsymbol{1}\}^{|\boldsymbol{X}|\times\boldsymbol{n}}$
  $\forall \boldsymbol{i} \in [\boldsymbol{n}]$ and $\forall \boldsymbol{x} \in \boldsymbol{X}$ we have a bit $\boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]$
- User $\boldsymbol{i}$ identifies corresponding bit $\boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$



Users randomize their corresponding bits:

User $\boldsymbol{i}$ sends $\boldsymbol{y_i} = \boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$ w.p. $\approx \frac{1}{2} + \epsilon$

$\boldsymbol{y_i} = -\boldsymbol{Z}[\boldsymbol{x_i}, \boldsymbol{i}]$ w.p. $\approx \frac{1}{2} - \epsilon$

Server: Given a query $\boldsymbol{x} \in \boldsymbol{X}$, return $\hat{\boldsymbol{f}}(\boldsymbol{x}) = \frac{1}{2\epsilon} \sum_{i\in[n]} \boldsymbol{y_i} \cdot \boldsymbol{Z}[\boldsymbol{x}, \boldsymbol{i}]$

$$\mathbb{E}\left[\sum_{i\in[n]} y_i \cdot Z[x, i]\right] = \sum_{i:x_i=x} \mathbb{E}[y_i \cdot Z[x, i]] + \sum_{i:x_i\neq x} \cancel{\mathbb{E}[y_i \cdot Z[x, i]]} = 2\epsilon \cdot f_S(x)$$

Hoeffding bound: w.h.p. our estimation error is at most $\approx \frac{1}{\epsilon}\sqrt{\boldsymbol{n} \cdot \log|\boldsymbol{X}|}$

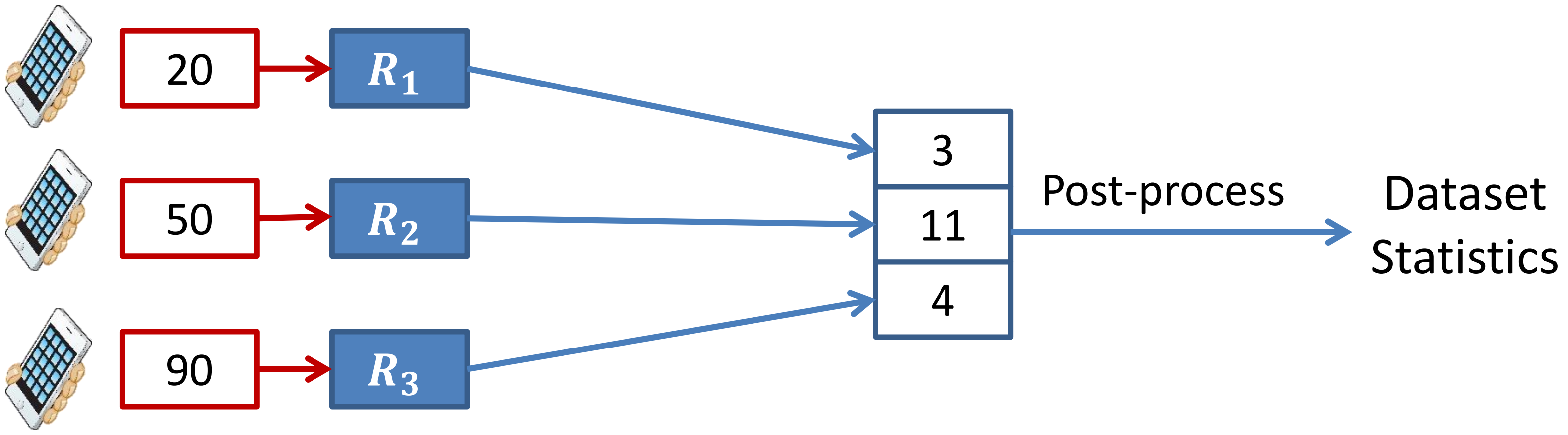
The Local Model of Differential Privacy

Today's Outline

1. What is the model?
2. Computing histograms
3. Computing averages
4. Clustering
5. LDP vs. statistical queries
6. Impossibility result for histograms
7. Interactive LDP protocols

Averages under LDP (informal)

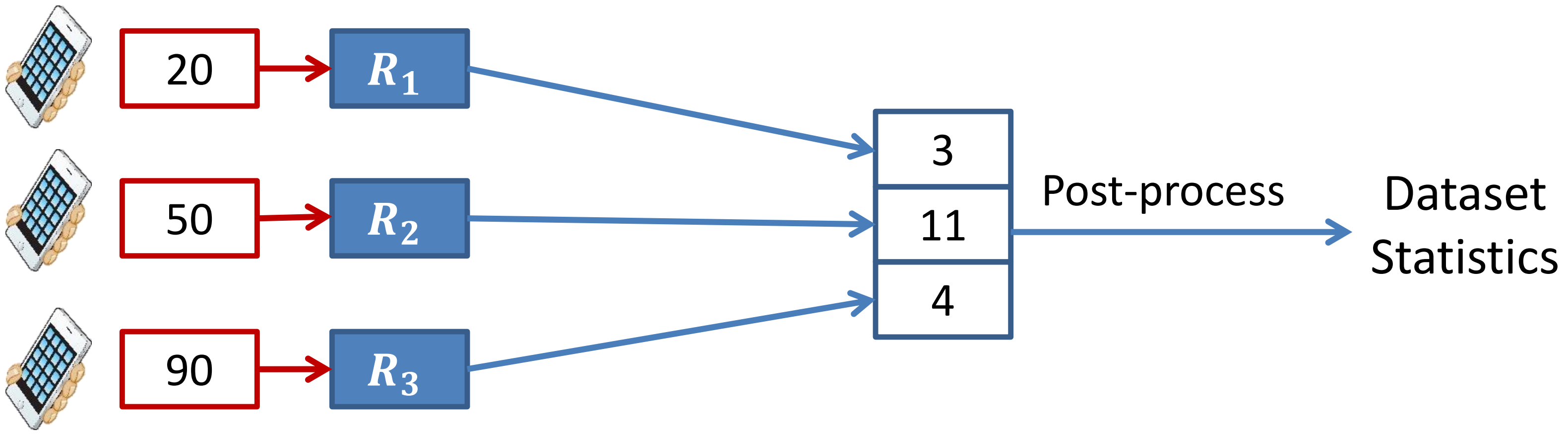
- Distributed database $S = (x_1, \ldots, x_n)$, each user holds a point $x_i \in \mathbb{R}$
- **Goal:** Estimate the average of S



Averages under LDP (informal)

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_n})$, each user holds a point $\boldsymbol{x_i} \in \mathbb{R}$
- **Goal:** Estimate the average of $\boldsymbol{S}$

For example, maybe the inputs are salaries, and our goal is to learn the average salary



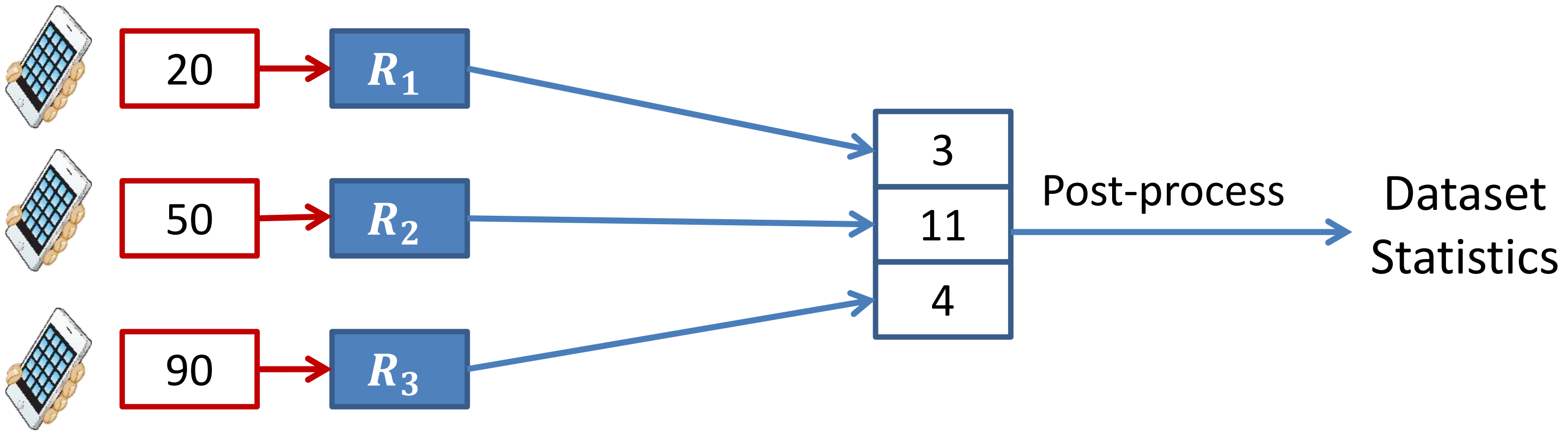
Averages under LDP (informal)

- Distributed database $S = (x_1, \ldots, x_n)$, each user holds a point $x_i \in \mathbb{R}$
- **Goal:** Estimate the average of S

Local randomizer $R(x)$:

Return x + random Gaussian noise (appropriately calibrated)

It can be shown that appropriately calibrated noise "hides" the information of every single individual, and that this randomizer satisfied the definition of LDP



Averages under LDP (informal)

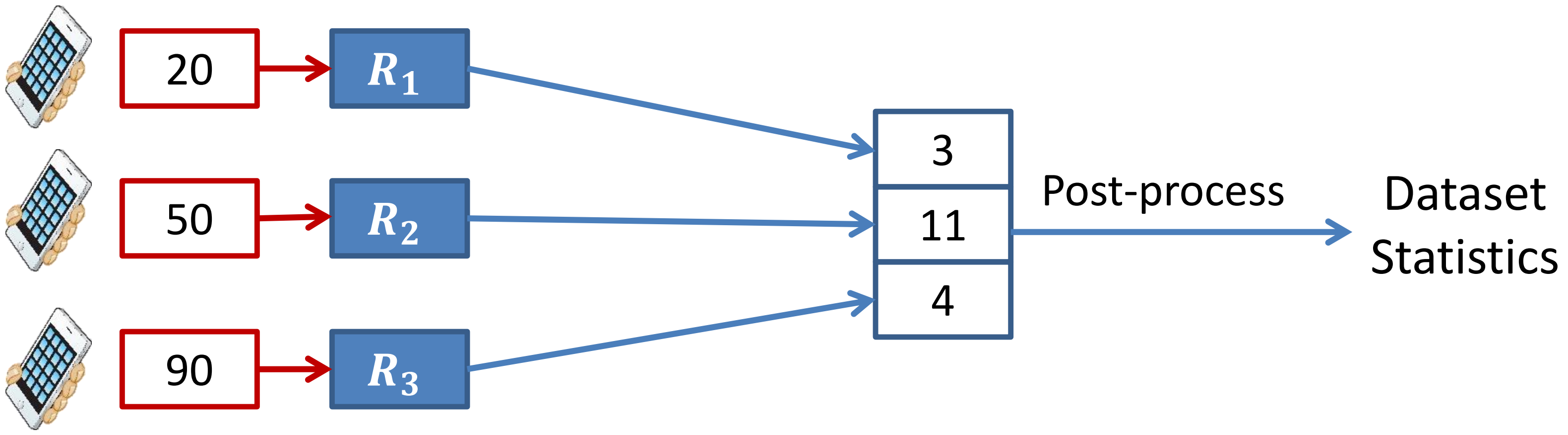
- Distributed database $\boldsymbol{S} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, each user holds a point $\boldsymbol{x}_i \in \mathbb{R}$
- **Goal:** Estimate the average of $\boldsymbol{S}$

Local randomizer $\boldsymbol{R}(\boldsymbol{x})$:

Return $\boldsymbol{x}$ + random Gaussian noise (appropriately calibrated)

It can be shown that appropriately calibrated noise “hides” the information of every single individual, and that this randomizer satisfied the definition of LDP

Protocol: From every user $\boldsymbol{i}$ obtain $\boldsymbol{y}_i \leftarrow \boldsymbol{R}(\boldsymbol{x}_i)$. Return $\frac{\mathbf{1}}{\boldsymbol{n}} \cdot \sum_i \boldsymbol{y}_i$



Averages under LDP (informal)

- Distributed database $S = (x_1, \dots, x_n)$, each user holds a point $x_i \in \mathbb{R}$
- **Goal:** Estimate the average of S

Local randomizer $R(x)$:
Return x + random Gaussian noise (appropriately calibrated)

It can be shown that appropriately calibrated noise "hides" the information of every single individual, and that this randomizer satisfied the definition of LDP

Protocol: From every user i obtain $y_i \leftarrow R(x_i)$. Return $\frac{1}{n} \cdot \sum_i y_i$

Analysis: $$\mathbb{E}\left[\frac{1}{n} \cdot \sum_i y_i\right] = \frac{1}{n} \cdot \sum_i x_i + \mathbb{E}\left[\frac{1}{n} \cdot \sum_i \mathbf{Noise}_i\right] = \frac{1}{n} \cdot \sum_i x_i$$

Averages under LDP (informal)

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$, each user holds a point $\boldsymbol{x_i} \in \mathbb{R}$
- **Goal:** Estimate the average of $\boldsymbol{S}$

Local randomizer $\boldsymbol{R}(\boldsymbol{x})$:

Return $\boldsymbol{x}$ + random Gaussian noise (appropriately calibrated)

It can be shown that appropriately calibrated noise "hides" the information of every single individual, and that this randomizer satisfied the definition of LDP

Protocol: From every user $\boldsymbol{i}$ obtain $\boldsymbol{y_i} \leftarrow \boldsymbol{R}(\boldsymbol{x_i})$. Return $\frac{\boldsymbol{1}}{\boldsymbol{n}} \cdot \sum_i \boldsymbol{y_i}$

Analysis: $\mathbb{E}\left[\frac{\boldsymbol{1}}{\boldsymbol{n}} \cdot \sum_i \boldsymbol{y_i}\right] = \frac{\boldsymbol{1}}{\boldsymbol{n}} \cdot \sum_i \boldsymbol{x_i} + \mathbb{E}\left[\frac{\boldsymbol{1}}{\boldsymbol{n}} \cdot \sum_i \mathbf{Noise}_i\right] = \frac{\boldsymbol{1}}{\boldsymbol{n}} \cdot \sum_i \boldsymbol{x_i}$

- **Error scales with $1/\sqrt{\boldsymbol{n}}$**
- **Can be extended to averages in $\boldsymbol{d}$-dimensions**

Takeaway: We can compute averages under LDP

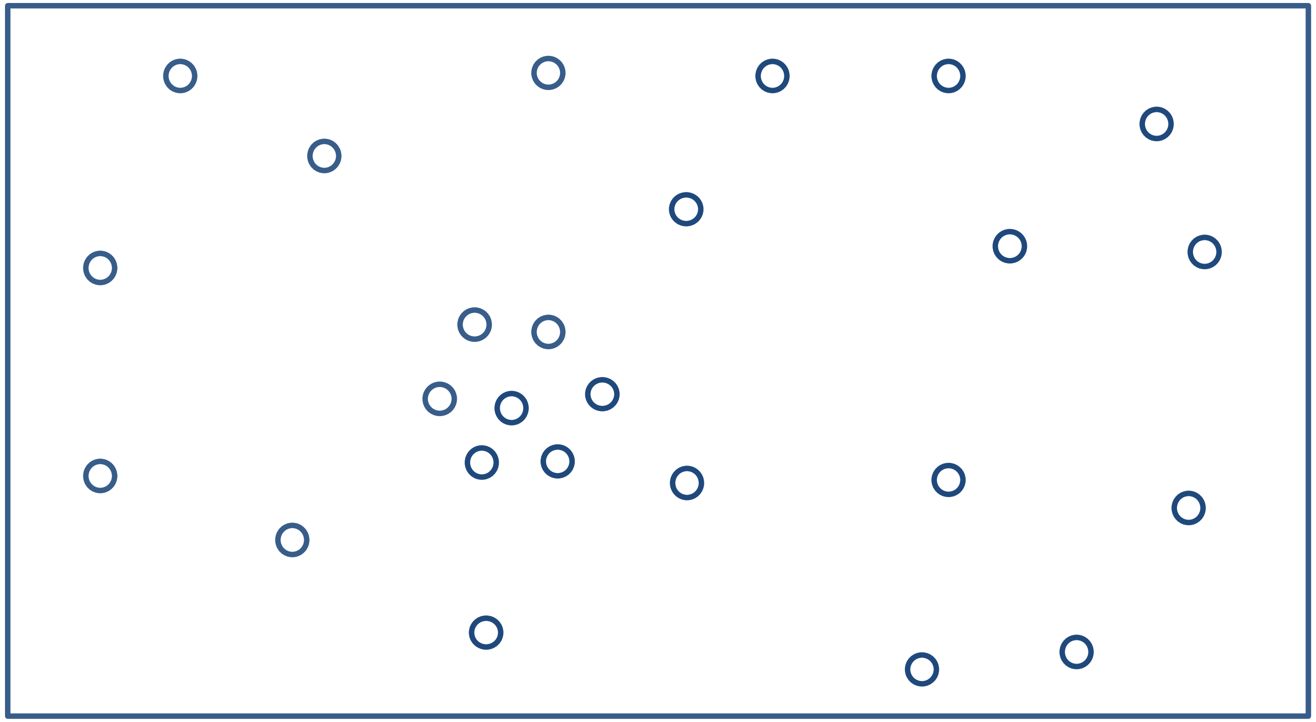
The Local Model of Differential Privacy

Today's Outline

1. What is the model?
2. Computing histograms
3. Computing averages
4. Clustering
5. LDP vs. statistical queries
6. Impossibility result for histograms
7. Interactive LDP protocols
8. A related model

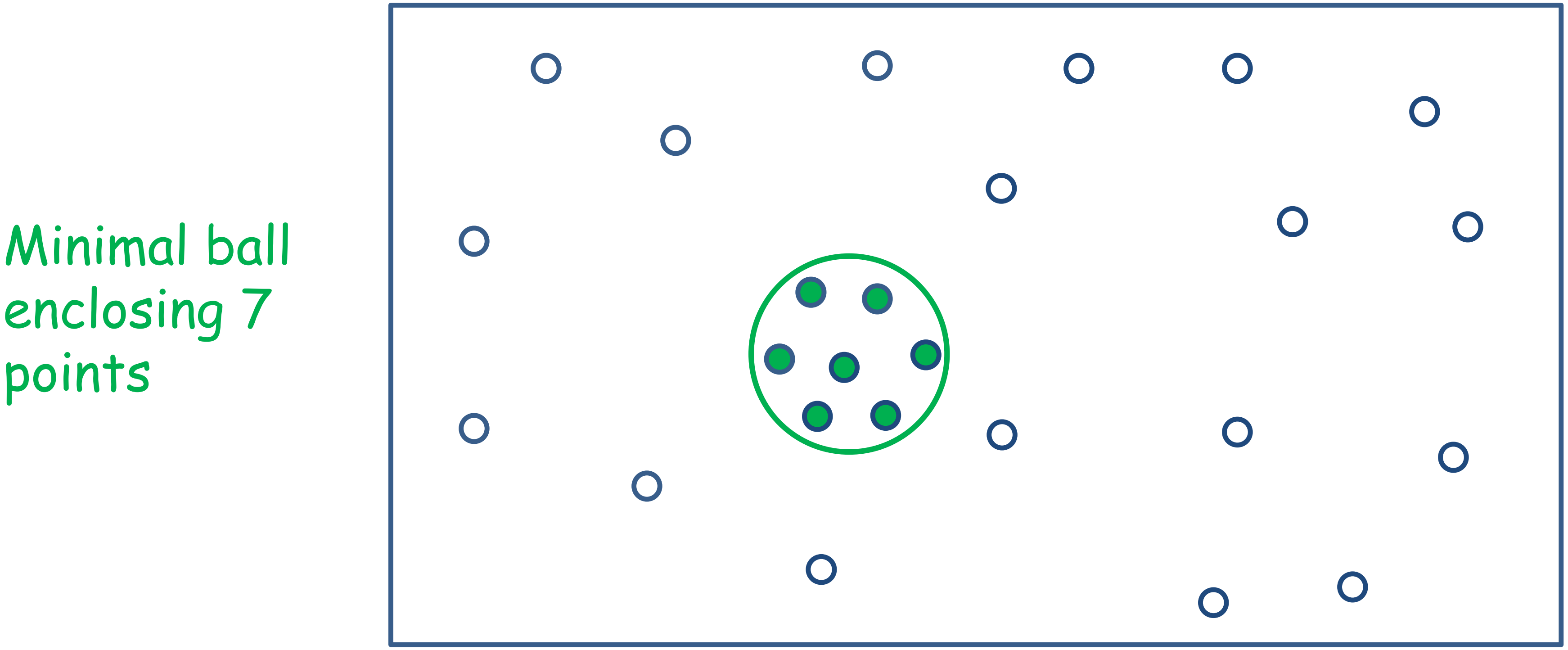
The 1-Cluster Problem

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_n})$, each user holds a point $\boldsymbol{x_i} \in \mathbb{R}^{\boldsymbol{d}}$
- **Find:** Center for a ball of minimal radius enclosing at least $\boldsymbol{t}$ input points



The 1-Cluster Problem

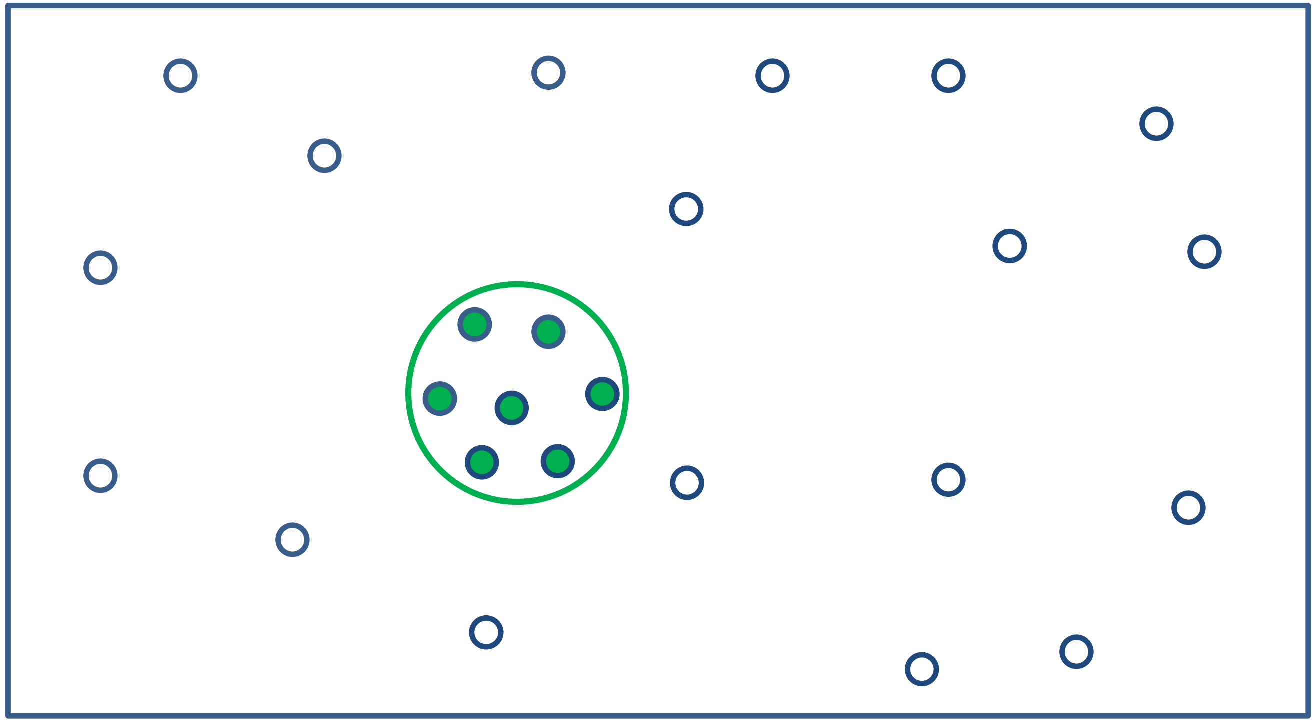
- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$, each user holds a point $\boldsymbol{x_i} \in \mathbb{R}^d$
- **Find:** Center for a ball of minimal radius enclosing at least $\boldsymbol{t}$ input points



The 1-Cluster Problem

- Distributed database $S = (x_1, \dots, x_n)$, each user holds a point $x_i \in \mathbb{R}^d$
- **Find:** Center for a ball of minimal radius enclosing at least t input points

Minimal ball enclosing 7 points



Applications:

✓ **Outlier removal**

✓ **Building block for more complex algorithms**

The 1-Cluster Problem

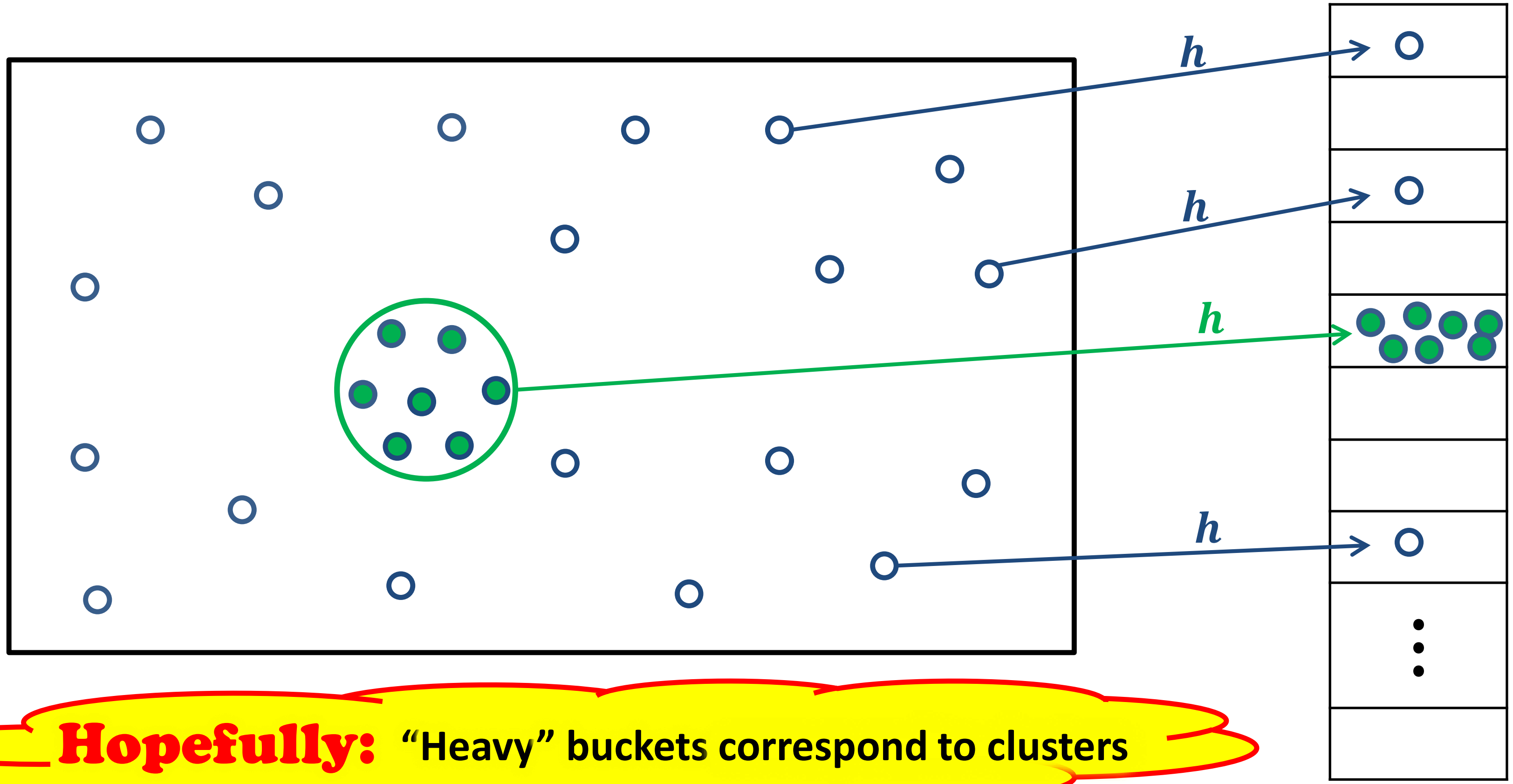
Useful Tool: Locality-Sensitive Hashing (LSH) [Indyk&Motwani]

- **Maximize** the probability of **collision** for **similar** items
- **Minimize** the probability of **collision** for **dissimilar** items

The 1-Cluster Problem

Useful Tool: Locality-Sensitive Hashing (LSH) [Indyk&Motwani]

- **Maximize** the probability of **collision** for **similar** items
- **Minimize** the probability of **collision** for **dissimilar** items



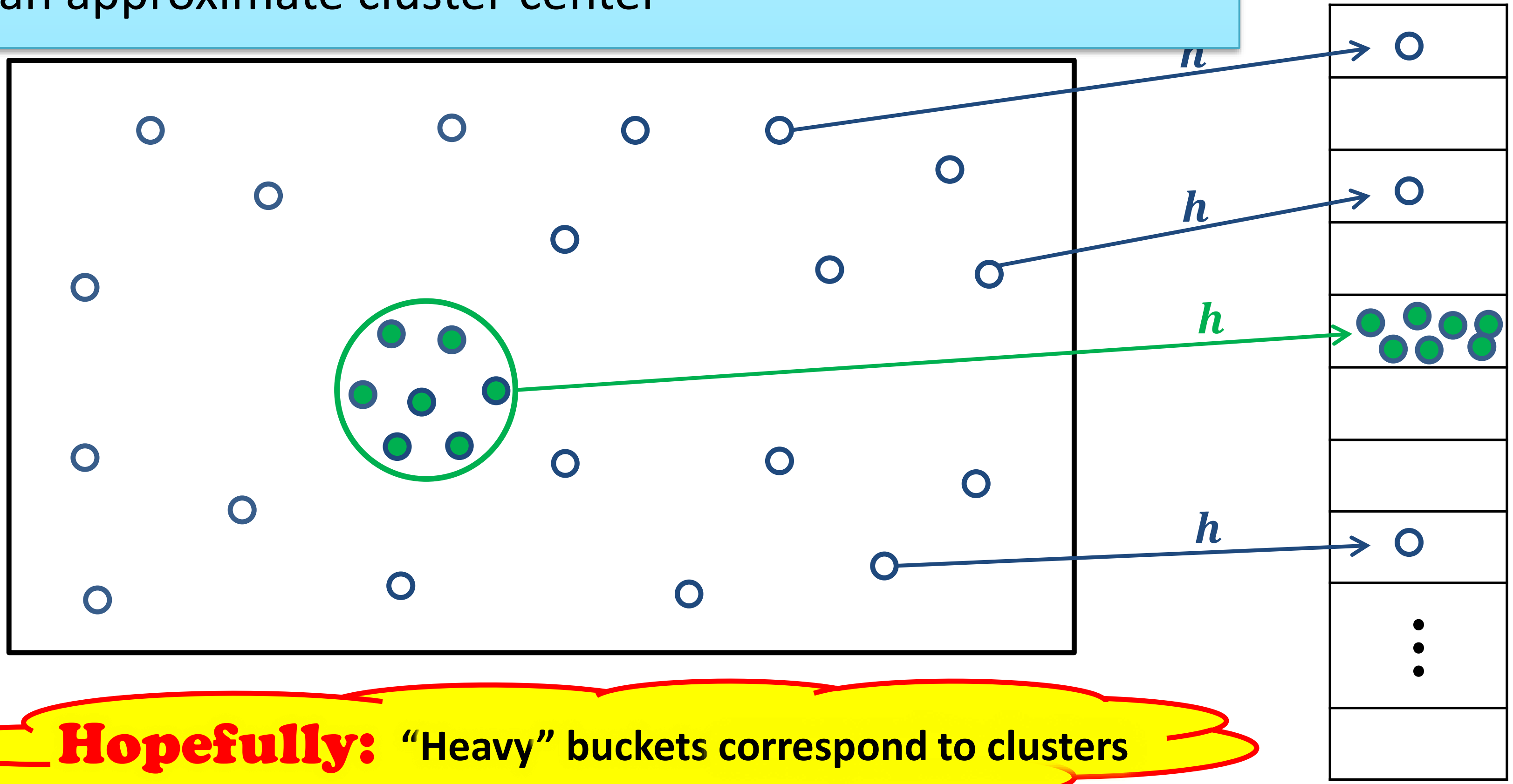
Hopefully: "Heavy" buckets correspond to clusters

Intuitive Overview

1. Identify “heavy” buckets in the hash range, using LDP tool for histograms
2. Identified buckets isolate clustered points
3. Clustered points can be averaged under LDP to obtain an approximate cluster center

[Indyk&Motwani]

ms

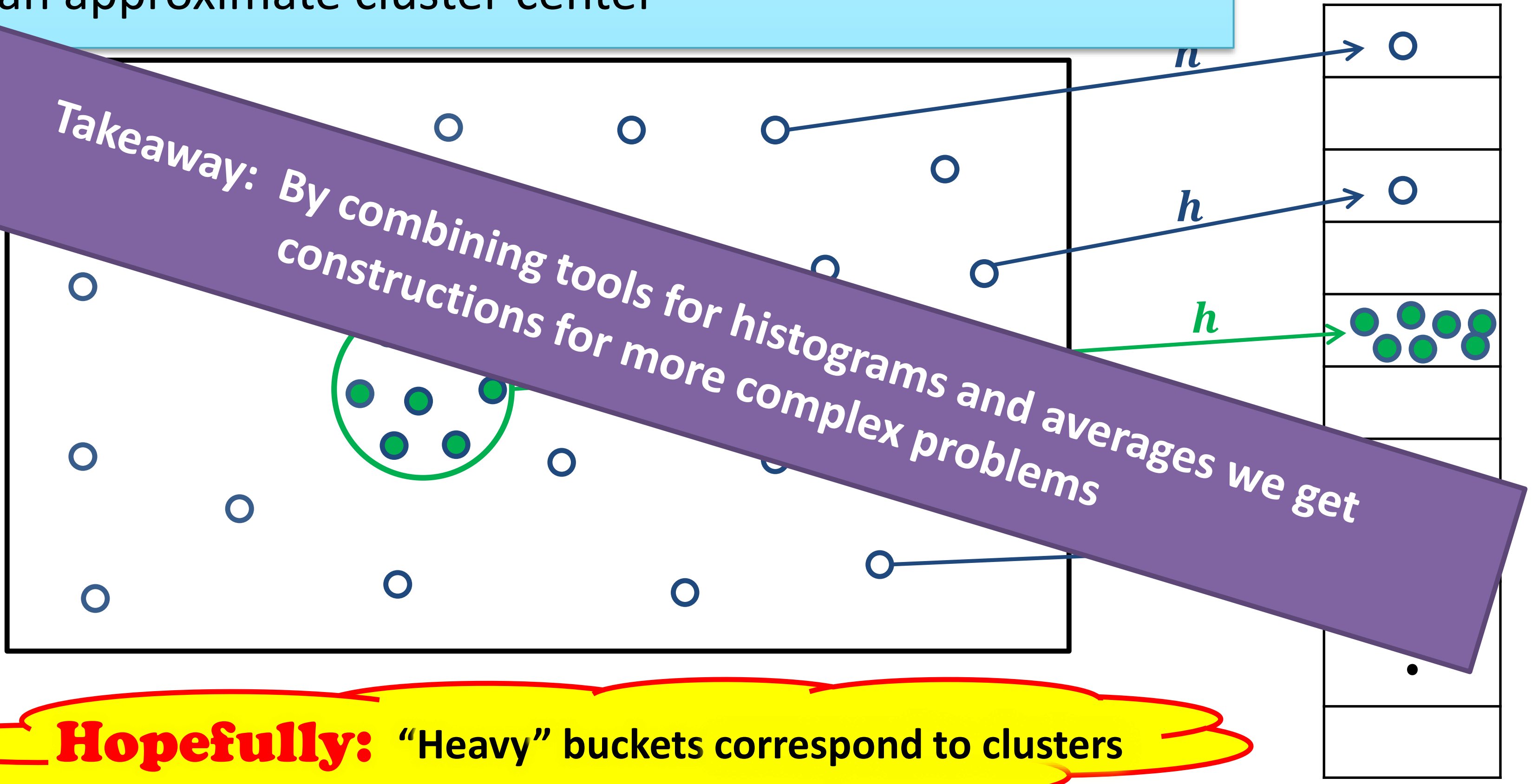


Hopefully: “Heavy” buckets correspond to clusters

Intuitive Overview

1. Identify “heavy” buckets in the hash range, using LDP tool for histograms
2. Identified buckets isolate clustered points
3. Clustered points can be averaged under LDP to obtain an approximate cluster center

[Indyk&Motwani]



Takeaway: By combining tools for histograms and averages we get constructions for more complex problems

Hopefully: “Heavy” buckets correspond to clusters

The Local Model of Differential Privacy

Today's Outline

1. What is the model?
2. Computing histograms
3. Computing averages
4. Clustering
5. LDP vs. statistical queries
6. Impossibility result for histograms
7. Interactive LDP protocols

The Local Model of Differential Privacy

Today's Outline

1. What is the model?
2. Computing histograms
3. Computing averages
4. Clustering

5. LDP vs. statistical queries
6. Impossibility result for histograms
7. Interactive LDP protocols

The Local Model of Differential Privacy

Today's Outline

Other problems that people have looked at:

- Convex optimization [FGV'17], [STU'17], [FMTT'18], [WGSX'20]
- Hypothesis testing [Sheffet'18], [GR'18], [JMNR'19]
- Hypothesis selection [GKKNWZ'20]
- Answering Queries [Bassily'19], [CKS'19]
- Reinforcement Learning [RZLS'20], [ZCHLW'20], [TWZW'21]
- Continual monitoring under LDP [EPK'14], [JRUW'18], [BY'21]

5. **LDP vs. statistical queries**

6. **Impossibility result for histograms**

7. **Interactive LDP protocols**

The Local Model of Differential Privacy

Today's Outline

1. What is the model?
2. Computing histograms
3. Computing averages
4. Clustering
5. LDP vs. statistical queries
6. Impossibility result for histograms
7. Interactive LDP protocols

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 $\boldsymbol{p}: \boldsymbol{X} \rightarrow \{\boldsymbol{0}, \boldsymbol{1}\}$ and obtains an estimate for $\mathbb{E}_{\boldsymbol{x} \sim \mathfrak{D}}[\boldsymbol{p}(\boldsymbol{x})]$

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

Data analyst

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 $\boldsymbol{p}: \boldsymbol{X} \rightarrow \{\mathbf{0}, \mathbf{1}\}$ and obtains an estimate for $\mathbb{E}_{\boldsymbol{x}\sim\mathfrak{D}}[\boldsymbol{p}(\boldsymbol{x})]$

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

$$\xleftarrow{\quad \boldsymbol{p_1} \quad}$$

$$\xrightarrow{\ \boldsymbol{a_1} \approx \mathbb{E}_{\boldsymbol{x}\sim\mathfrak{D}}[\boldsymbol{p_1}(\boldsymbol{x})]\ }$$

Data analyst

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 $\boldsymbol{p}: \boldsymbol{X} \to \{\mathbf{0}, \mathbf{1}\}$ and obtains an estimate for $\mathbb{E}_{x \sim \mathfrak{D}}[\boldsymbol{p}(\boldsymbol{x})]$

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

$\boldsymbol{p_1}$ (←)

$\boldsymbol{a_1} \approx \mathbb{E}_{x \sim \mathfrak{D}}[\boldsymbol{p_1}(\boldsymbol{x})]$ (→)

$\boldsymbol{p_2}$ (←)

$\boldsymbol{a_2} \approx \mathbb{E}_{x \sim \mathfrak{D}}[\boldsymbol{p_2}(\boldsymbol{x})]$ (→)

$\vdots$

Data analyst

The Statistical Queries Model

Theorem [Kasiviswanathan, Lee, Nissim, Raskhodnikova, Smith 08]

What you can learn in the SQ model is exactly what you can learn in the LDP model (where every user holds a point sampled from $\mathfrak{D}$)

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

$\xleftarrow{\boldsymbol{p_1}}$

$\xrightarrow{\boldsymbol{a_1} \approx \mathbb{E}_{x\sim\mathfrak{D}}[\boldsymbol{p_1}(\boldsymbol{x})]}$

$\xleftarrow{\boldsymbol{p_2}}$

$\xrightarrow{\boldsymbol{a_2} \approx \mathbb{E}_{x\sim\mathfrak{D}}[\boldsymbol{p_2}(\boldsymbol{x})]}$

$\vdots$

Data analyst

The Statistical Queries Model

Theorem [Kasiviswanathan, Lee, Nissim, Raskhodnikova, Smith 08]

What you can learn in the SQ model is exactly what you can learn in the LDP model (where every user holds a point sampled from $\mathfrak{D}$)

Easy direction of equivalence:

Every statistical query $\boldsymbol{p}$ can be answered under LDP by estimating the number of users $\boldsymbol{i}$ s.t. $\boldsymbol{p}(\boldsymbol{x}_i) = \mathbf{1}$

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

$\boldsymbol{p}_1$

$\boldsymbol{a}_1 \approx \mathbb{E}_{x\sim\mathfrak{D}}[\boldsymbol{p}_1(\boldsymbol{x})]$

$\boldsymbol{p}_2$

$\boldsymbol{a}_2 \approx \mathbb{E}_{x\sim\mathfrak{D}}[\boldsymbol{p}_2(\boldsymbol{x})]$

$\vdots$

Data analyst

The Statistical Queries Model

Theorem [Kasiviswanathan, Lee, Nissim, Raskhodnikova, Smith 08]

What you can learn in the SQ model is exactly what you can learn in the LDP model (where every user holds a point sampled from $\mathfrak{D}$)

Easy direction of equivalence:

Every statistical query $\boldsymbol{p}$ can be answered under LDP by estimating the number of users $\boldsymbol{i}$ s.t. $\boldsymbol{p(x_i) = 1}$

The great news: The SQ model is well-studied and known to be very expressive. All the existing SQ algorithms can be implemented under LDP!

The great impossibility news*: What cannot be done in the SQ model cannot be done under LDP, e.g., learning PARITY

The Local Model of Differential Privacy

Today's Outline

1. What is the model?
2. Computing histograms
3. Computing averages
4. Clustering
5. LDP vs. statistical queries
6. Impossibility result for histograms
7. Interactive LDP protocols

Histograms under LDP – An impossibility result

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_n}) \in \boldsymbol{X^n}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S}(\boldsymbol{x})$

Theorem: Under LDP, must have error $\boldsymbol{\Omega\left(\frac{1}{\varepsilon}\sqrt{n \cdot \log|X|}\right)}$ [Chan Shi Song] [Bassily Smith]

Histograms under LDP – An impossibility result

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \boldsymbol{X^n}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S}(\boldsymbol{x})$

Theorem: **Under LDP, must have error** $\boldsymbol{\Omega\left(\frac{1}{\varepsilon}\sqrt{n \cdot \log|X|}\right)}$ [Chan Shi Song] [Bassily Smith]

Proof idea: $\boldsymbol{\Omega(\sqrt{n})}$ **for estimating the multiplicity of 1 in the database**

- Let $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \{\boldsymbol{0}, \boldsymbol{1}\}^{\boldsymbol{n}}$ be chosen uniformly at random

Histograms under LDP – An impossibility result

- Distributed database $\boldsymbol{S} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \boldsymbol{X}^n$, where user $\boldsymbol{i}$ holds $\boldsymbol{x}_i \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f}_{\boldsymbol{S}}(\boldsymbol{x})$

Theorem: Under LDP, must have error $\boldsymbol{\Omega}\left(\frac{1}{\varepsilon}\sqrt{\boldsymbol{n}\cdot\log|X|}\right)$ [Chan Shi Song] [Bassily Smith]

Proof idea: $\boldsymbol{\Omega}(\sqrt{\boldsymbol{n}})$ for estimating the multiplicity of 1 in the database

- Let $\boldsymbol{S} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) \in \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{n}}$ be chosen uniformly at random
- Let $\boldsymbol{T}$ denote the transcript. **Main observation: inputs remain roughly uniform given the transcript**
- Specifically, for every $\boldsymbol{t}$ and $\boldsymbol{i}$ we have: $\mathbf{Pr}[\boldsymbol{x}_i = \mathbf{1} | \boldsymbol{T} = \boldsymbol{t}] \approx \frac{\mathbf{1}}{\mathbf{2}} \approx \mathbf{Pr}[\boldsymbol{x}_i = \mathbf{0} | \boldsymbol{T} = \boldsymbol{t}]$

Histograms under LDP – An impossibility result

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \boldsymbol{X^n}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S}(\boldsymbol{x})$

<u>**Theorem:**</u> **Under LDP, must have error** $\boldsymbol{\Omega\left(\frac{1}{\varepsilon}\sqrt{n \cdot \log|X|}\right)}$ [Chan Shi Song] [Bassily Smith]

<u>**Proof idea**</u>**:** $\boldsymbol{\Omega(\sqrt{n})}$ **for estimating the multiplicity of 1 in the database**

- Let $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \{\boldsymbol{0}, \boldsymbol{1}\}^{\boldsymbol{n}}$ be chosen uniformly at random
- Let $\boldsymbol{T}$ denote the transcript. **Main observation: inputs remain roughly uniform given the transcript**
- Specifically, for every $\boldsymbol{t}$ and $\boldsymbol{i}$ we have: $\mathbf{Pr}[\boldsymbol{x_i} = \mathbf{1} | \boldsymbol{T} = \boldsymbol{t}] \approx \frac{\mathbf{1}}{\mathbf{2}} \approx \mathbf{Pr}[\boldsymbol{x_i} = \mathbf{0} | \boldsymbol{T} = \boldsymbol{t}]$

$$\Pr[x_i = 1 | T = t] = \Pr[T = t | x_i = 1] \cdot \frac{\Pr[x_i = 1]}{\Pr[T = t]} \approx \Pr[T = t | x_i = 0] \cdot \frac{\Pr[x_i = 0]}{\Pr[T = t]} = \Pr[x_i = 0 | T = t]$$

Histograms under LDP – An impossibility result

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \boldsymbol{X^n}$, where user $\boldsymbol{i}$ holds $\boldsymbol{x_i} \in \boldsymbol{X}$
- **Goal:** For every $\boldsymbol{x} \in \boldsymbol{X}$, estimate the multiplicity of $\boldsymbol{x}$ in $\boldsymbol{S}$, denoted $\boldsymbol{f_S(x)}$

Theorem: Under LDP, must have error $\boldsymbol{\Omega\left(\frac{1}{\varepsilon}\sqrt{n \cdot \log|X|}\right)}$ [Chan Shi Song] [Bassily Smith]

Proof idea: $\boldsymbol{\Omega(\sqrt{n})}$ for estimating the multiplicity of 1 in the database

- Let $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n}) \in \{\boldsymbol{0}, \boldsymbol{1}\}^{\boldsymbol{n}}$ be chosen uniformly at random
- Let $\boldsymbol{T}$ denote the transcript. **Main observation: inputs remain roughly uniform given the transcript**
- Specifically, for every $\boldsymbol{t}$ and $\boldsymbol{i}$ we have: $\mathbf{Pr}[\boldsymbol{x_i} = \mathbf{1} | \boldsymbol{T} = \boldsymbol{t}] \approx \frac{\mathbf{1}}{\mathbf{2}} \approx \mathbf{Pr}[\boldsymbol{x_i} = \mathbf{0} | \boldsymbol{T} = \boldsymbol{t}]$

$$\Pr[x_i = 1 | T = t] = \Pr[T = t | x_i = 1] \cdot \frac{\Pr[x_i = 1]}{\Pr[T = t]} \approx \Pr[T = t | x_i = 0] \cdot \frac{\Pr[x_i = 0]}{\Pr[T = t]} = \Pr[x_i = 0 | T = t]$$

- So, conditioned on the transcript, $\sum \boldsymbol{x_i}$ is the sum of n nearly uniform bits (and they remain independent). By anti-Chernoff, the error is $\boldsymbol{\Omega(\sqrt{n})}$

Histograms under LDP – An impossibility result

- Distributed database $S = (x_1, \dots, x_n) \in X^n$, where user i holds $x_i \in X$
- **Goal:** For every $x \in X$, estimate the multiplicity of x in S, denoted $f_S(x)$

Theorem: **Under LDP, must have error $\Omega\left(\frac{1}{\varepsilon}\sqrt{n \cdot \log|X|}\right)$** [Chan Shi Song] [Bassily Smith]

Pr f idea: $\Omega(\sqrt{n})$ for estimating the multiplicity of 1 in the database

-) $\in \{0,1\}^n$ be chosen uniformly at random
- Let T deno bservation: inputs remain roughly uniform given the transcript
- Specifically, for every t and i $t] \approx \frac{1}{2} \approx \Pr[x_i = 0 | T = t]$

$$\Pr[x_i = 1|T = t] = \Pr[T = t|x_i = 1] \cdot \frac{\Pr[x_i = 1]}{\Pr[T = t]} \approx \Pr[\ \ \frac{\Pr[x_i = 0]}{\ } = \Pr[x_i = 0|T = t]$$

- So, conditioned on the transcript, $\sum x_i$ is the sum of n nearly uniform bits (and independent). By anti-Chernoff, the error is $\Omega(\sqrt{n})$

This should be contrasted with the centralized model, where the error does not scale with $\sqrt{n}$

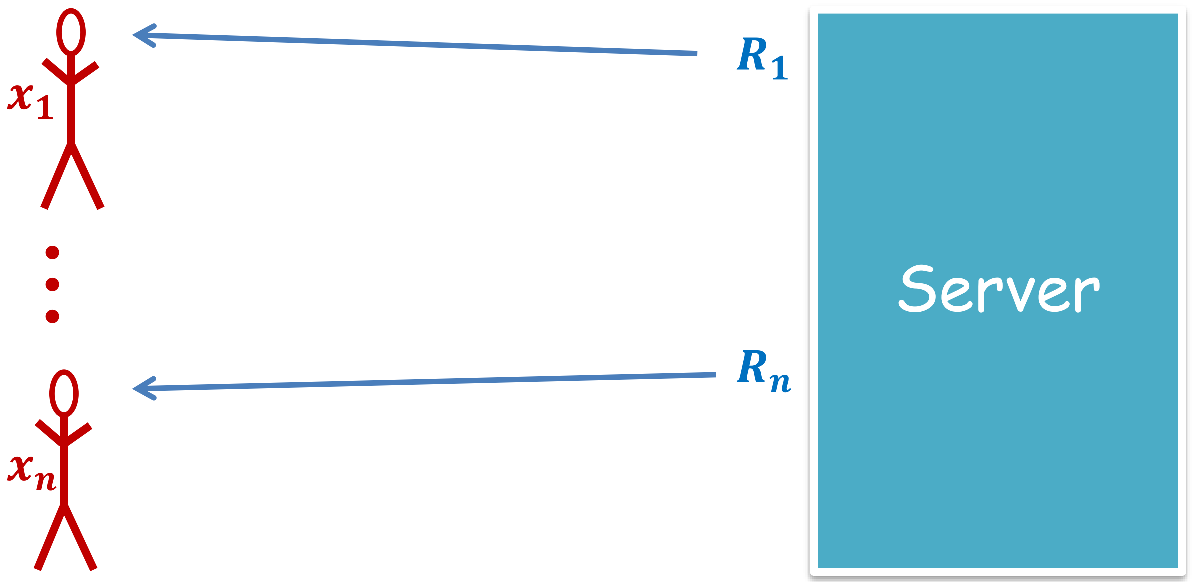
The Local Model of Differential Privacy

Today's Outline

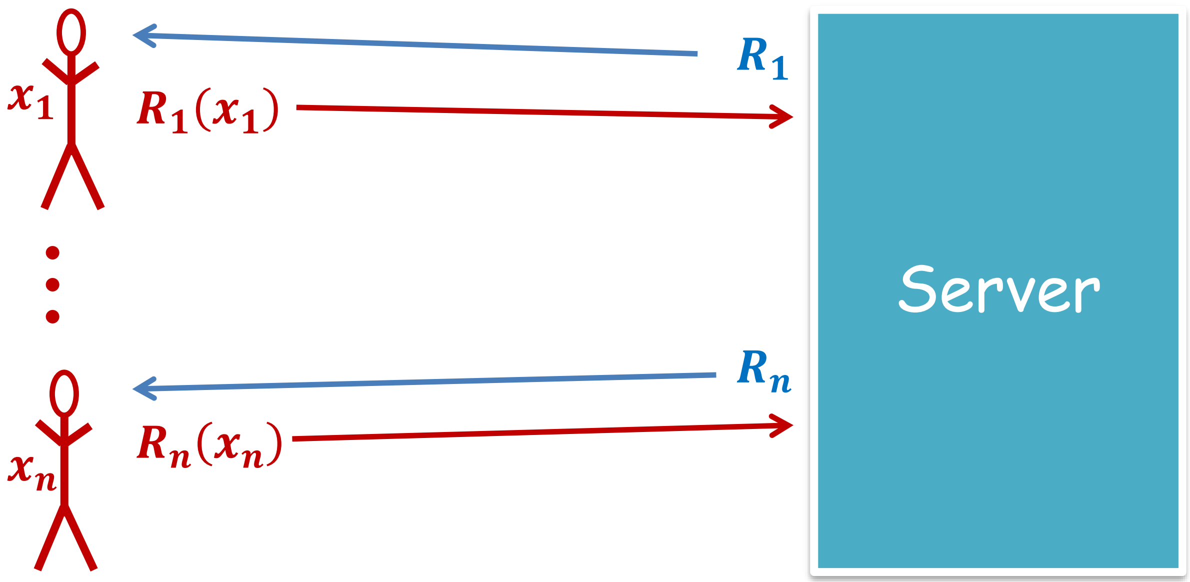
1. What is the model?
2. Computing histograms
3. Computing averages
4. Clustering
5. LDP vs. statistical queries
6. Impossibility result for histograms
7. Interactive LDP protocols

Non- vs. Semi- vs. Fully-interactive LDP

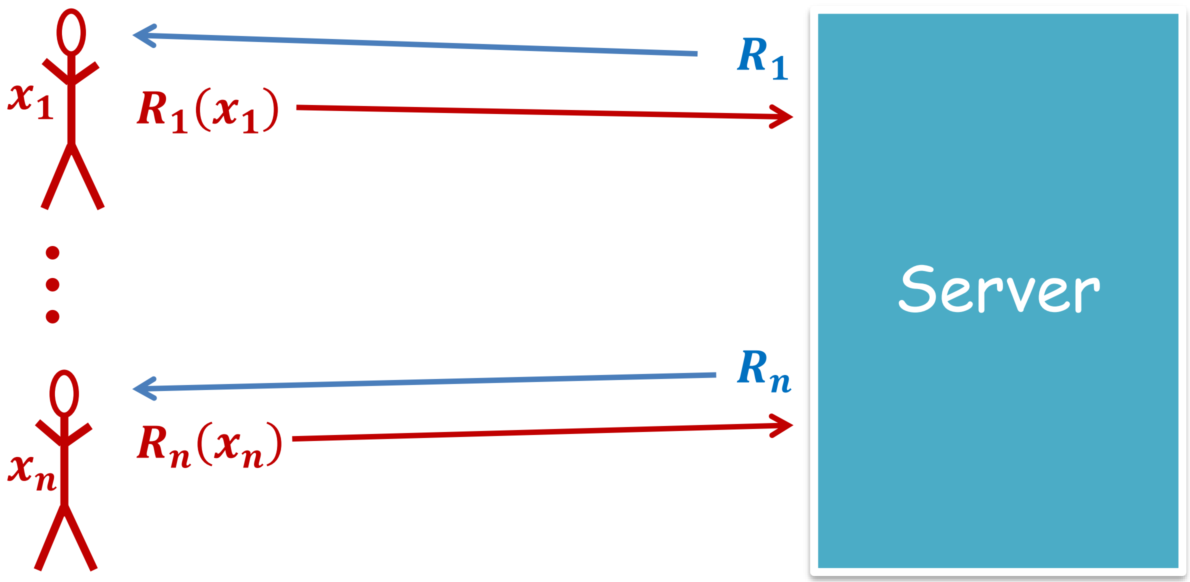
Non- vs. Semi- vs. Fully-interactive LDP



Non- vs. Semi- vs. Fully-interactive LDP



Non- vs. Semi- vs. Fully-interactive LDP

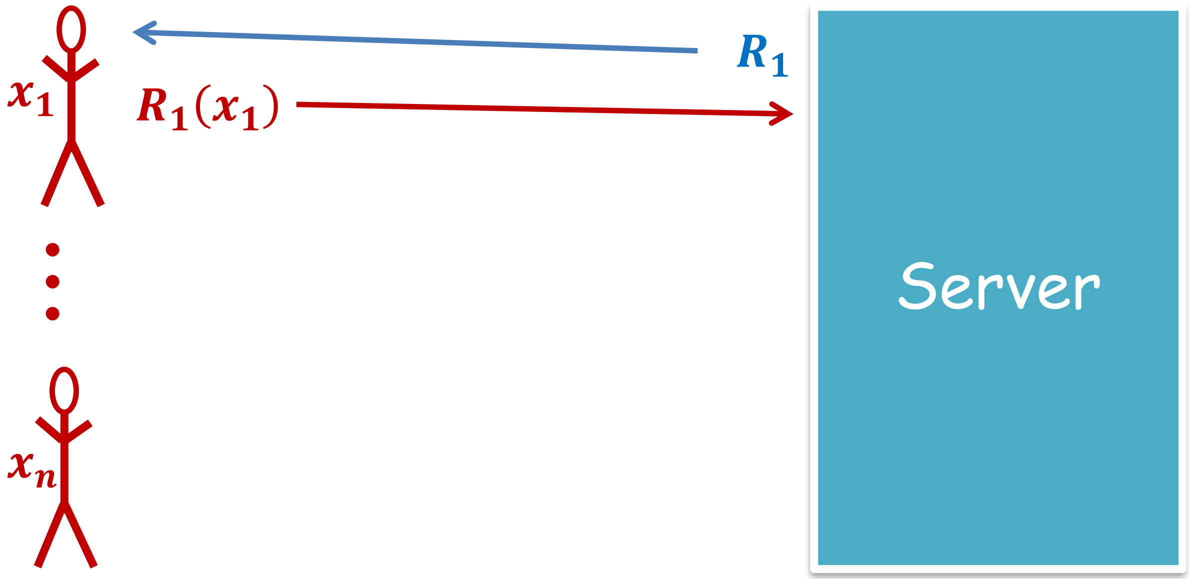


- Non-interactive protocols prepare all the $\mathcal{R}_i$'s before receiving any messages

Non- vs. Semi- vs. Fully-interactive LDP

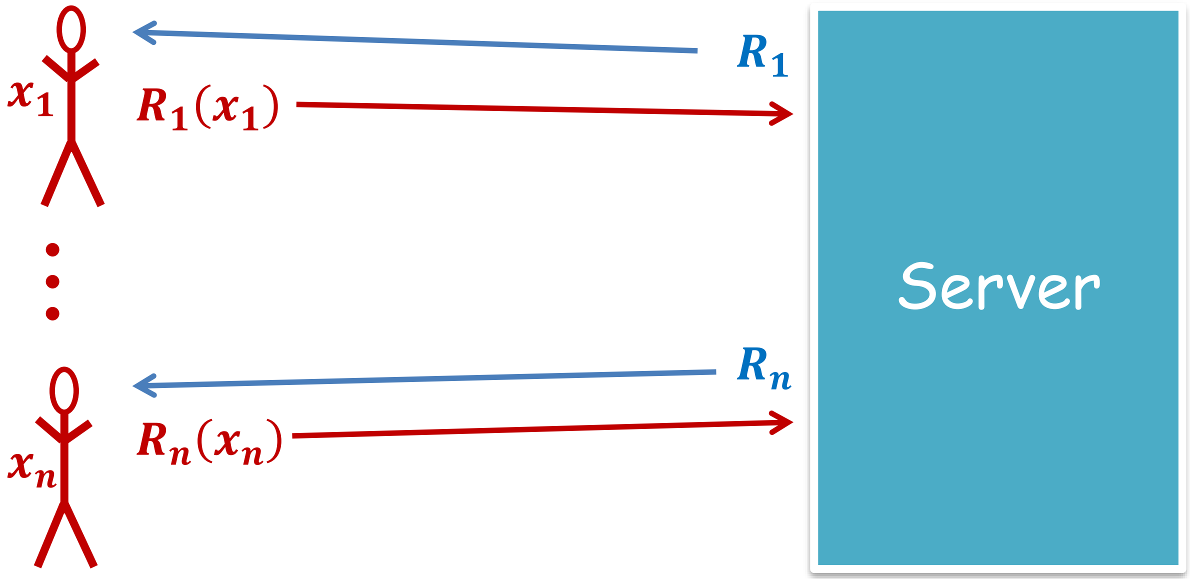
- Non-interactive protocols prepare all the $\mathcal{R}_i$'s before receiving any messages

Non- vs. Semi- vs. Fully-interactive LDP



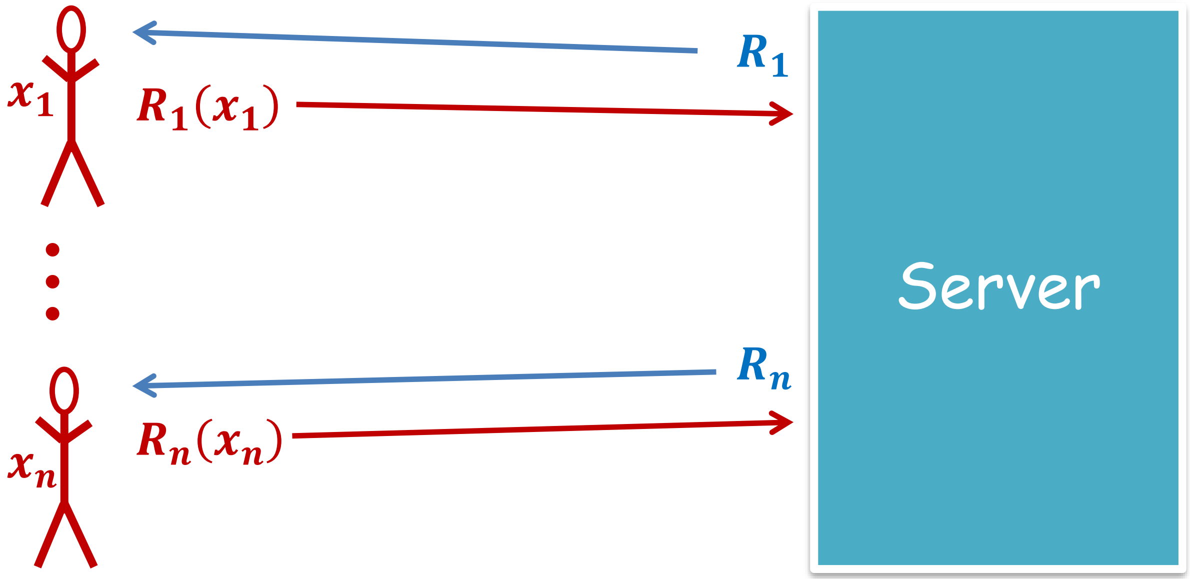
- Non-interactive protocols prepare all the $\mathcal{R}_i$'s before receiving any messages

Non- vs. Semi- vs. Fully-interactive LDP



- Non-interactive protocols prepare all the $\mathcal{R}_i$'s before receiving any messages

Non- vs. Semi- vs. Fully-interactive LDP

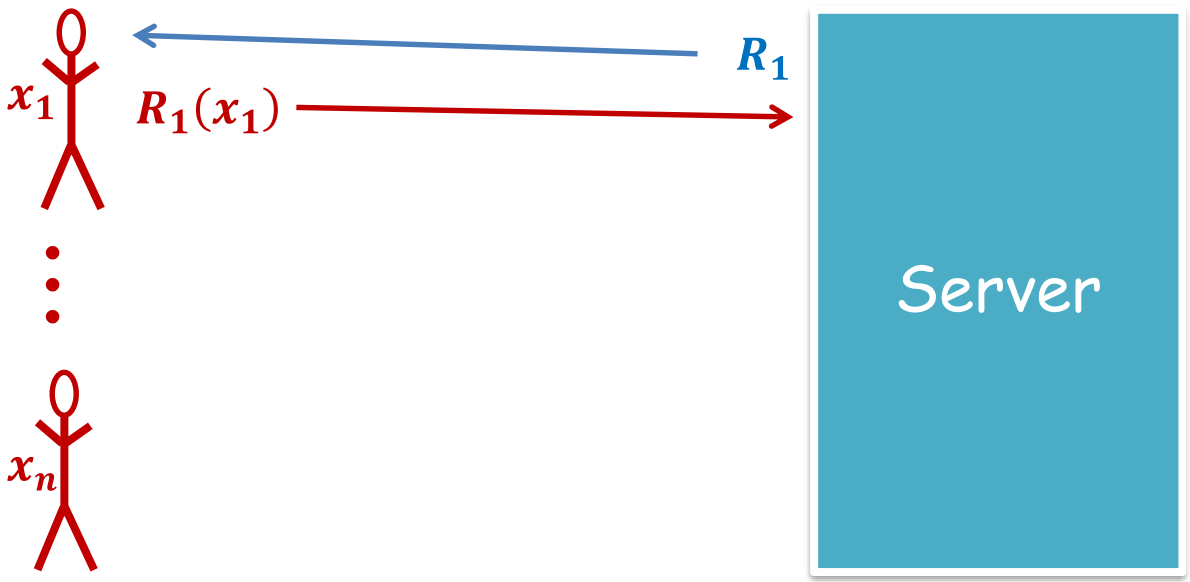


- Non-interactive protocols prepare all the $\mathcal{R}_i$'s before receiving any messages
- Semi-interactive protocols can interact with every user at most once

Non- vs. Semi- vs. Fully-interactive LDP

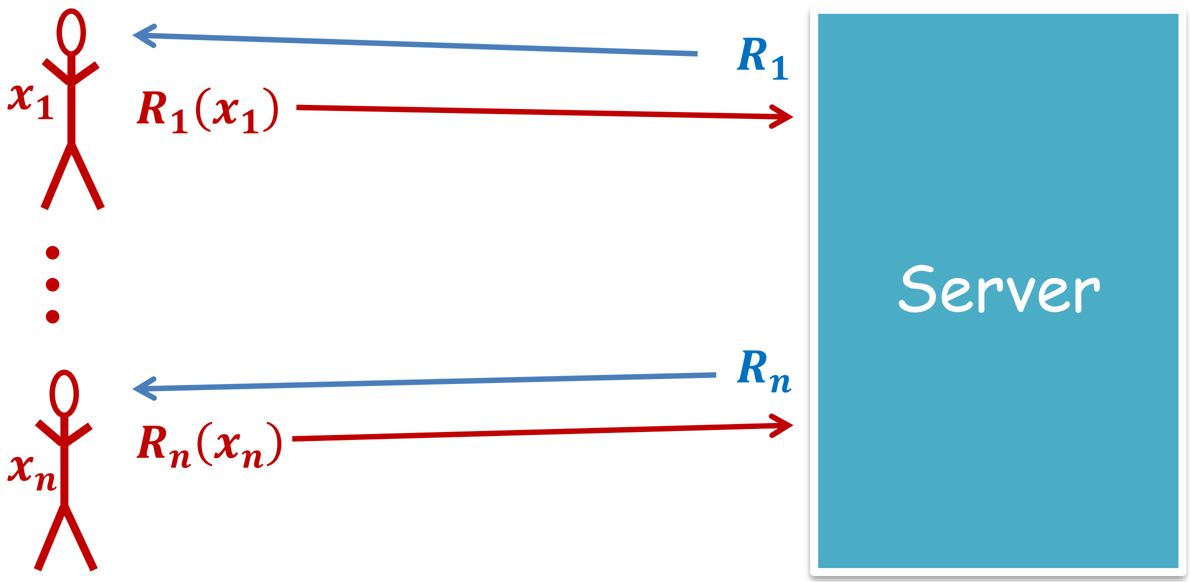
- Non-interactive protocols prepare all the $\mathcal{R}_i$'s before receiving any messages
- Semi-interactive protocols can interact with every user at most once

Non- vs. Semi- vs. Fully-interactive LDP



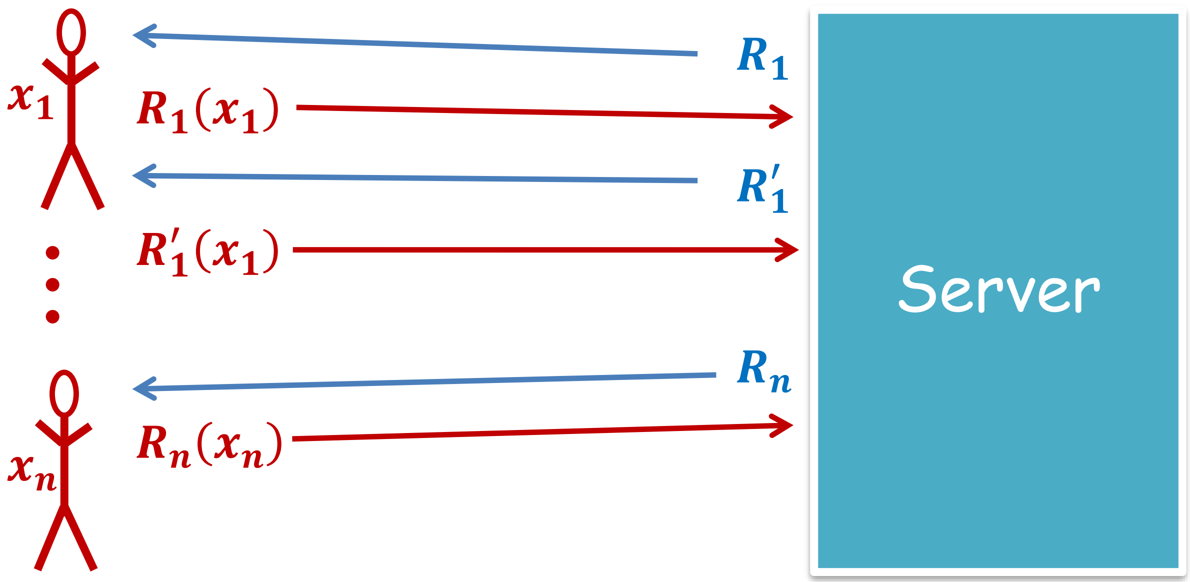
- Non-interactive protocols prepare all the $\mathcal{R}_i$'s before receiving any messages
- Semi-interactive protocols can interact with every user at most once

Non- vs. Semi- vs. Fully-interactive LDP



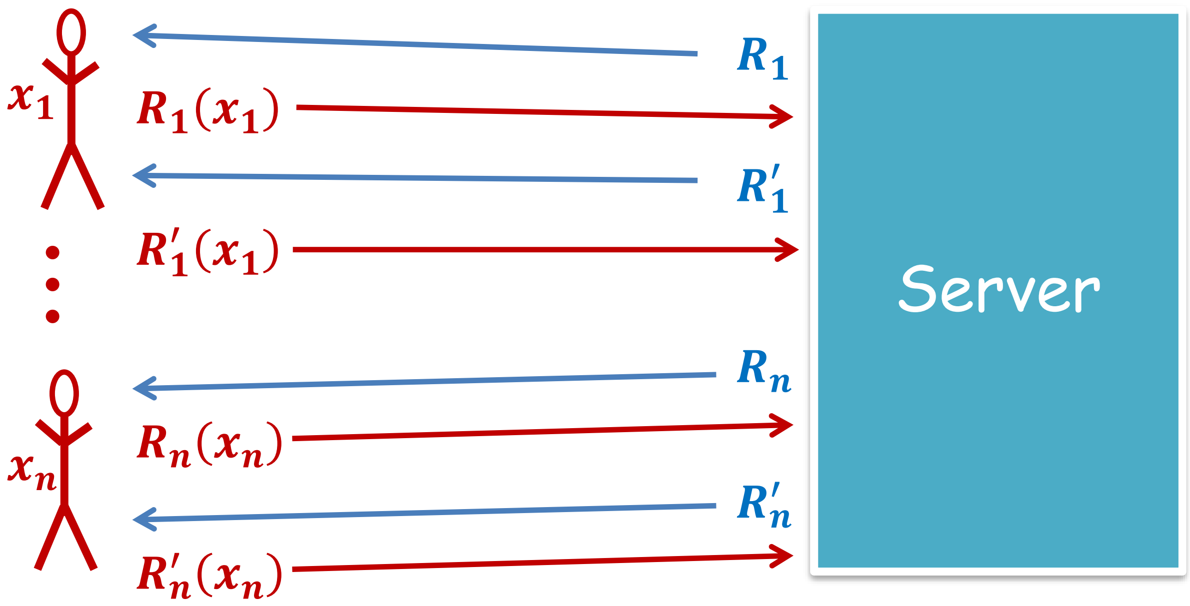
- Non-interactive protocols prepare all the $\mathcal{R}_i$'s before receiving any messages
- Semi-interactive protocols can interact with every user at most once

Non- vs. Semi- vs. Fully-interactive LDP



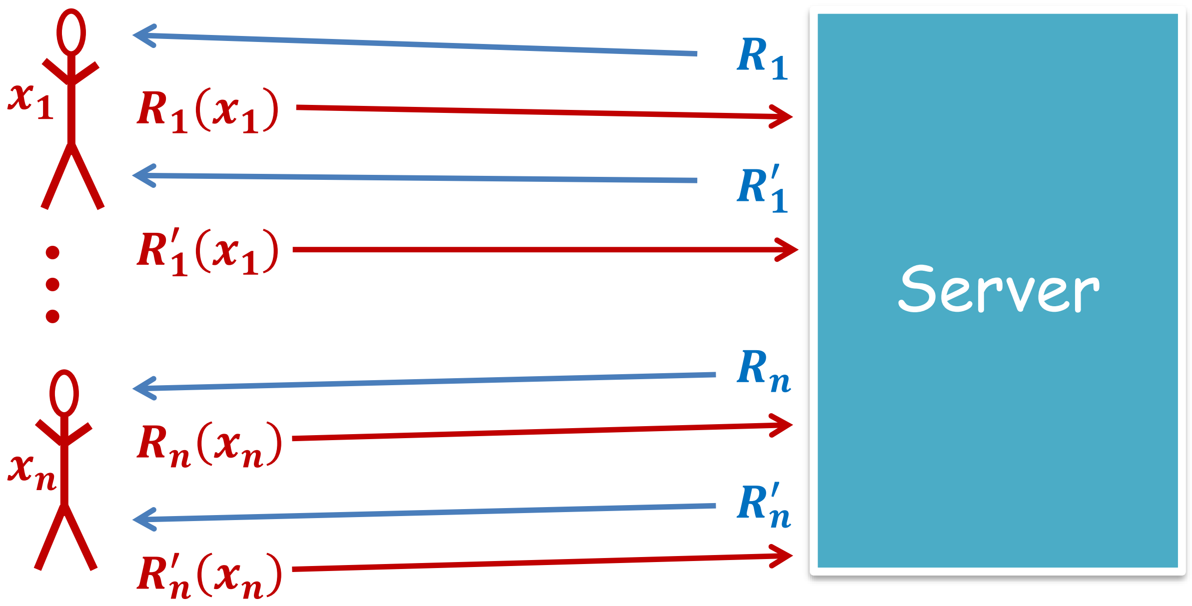
- Non-interactive protocols prepare all the $\mathcal{R}_i$'s before receiving any messages
- Semi-interactive protocols can interact with every user at most once

Non- vs. Semi- vs. Fully-interactive LDP



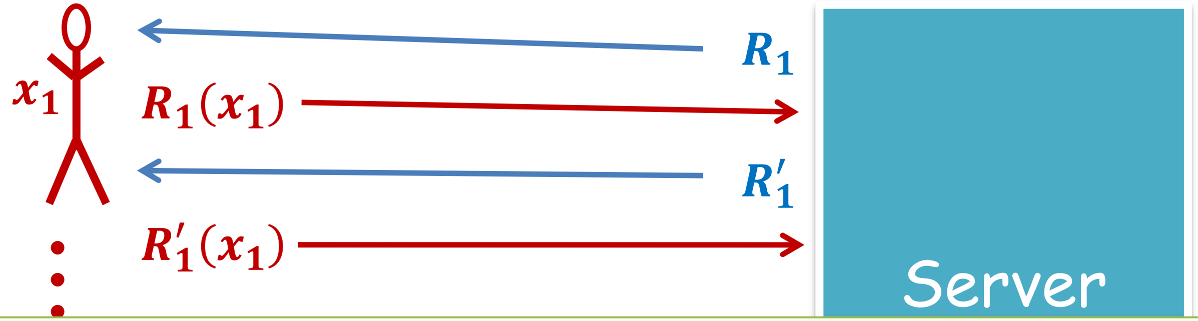
- Non-interactive protocols prepare all the $\mathcal{R}_i$'s before receiving any messages
- Semi-interactive protocols can interact with every user at most once

Non- vs. Semi- vs. Fully-interactive LDP



- Non-interactive protocols prepare all the $\mathcal{R}_i$'s before receiving any messages
- Semi-interactive protocols can interact with every user at most once
- **Fully-interactive protocols are unrestricted.**

Non- vs. Semi- vs. Fully-interactive LDP



<u>Separating non- from semi-interactive LDP:</u>

- Masked PARITY [Kasiviswanathan, Lee, Nissim, Raskhodnikova, Smith 08]
- Learning halfspaces [Daniely, Feldman 19]

<u>Separating semi- from fully-interactive LDP:</u>

- Hidden layers problem [Joseph, Mao, Roth 20]
- Pointer chasing [Joseph, Mao, Roth 20]

A simple pointer chasing problem

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$, where:
  - Every user $\mathbf{1} \le \boldsymbol{i} < \boldsymbol{n}/\mathbf{2}$ holds input $\boldsymbol{x_i} \in \{\mathbf{1}, \mathbf{2}, \dots, \boldsymbol{L}\}$
  - Every user $\boldsymbol{n}/\mathbf{2} \le \boldsymbol{j} \le \boldsymbol{n}$ holds input $\boldsymbol{x_j} = \left(\boldsymbol{x_j}[\mathbf{1}], \dots, \boldsymbol{x_j}[\boldsymbol{L}]\right) \in \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{L}}$
- **Goal: If $x_1 = x_2 = \cdots = x_{n/2} = \ell$, then estimate the average $\frac{2}{n}\sum_{j=n/2}^{n} x_j[\ell]$**

A simple pointer chasing problem

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$, where:
  - Every user $\mathbf{1} \leq \boldsymbol{i} < \boldsymbol{n}/\mathbf{2}$ holds input $\boldsymbol{x_i} \in \{\mathbf{1}, \mathbf{2}, \dots, \boldsymbol{L}\}$
  - Every user $\boldsymbol{n}/\mathbf{2} \leq \boldsymbol{j} \leq \boldsymbol{n}$ holds input $\boldsymbol{x_j} = \left(\boldsymbol{x_j}[\mathbf{1}], \dots, \boldsymbol{x_j}[\boldsymbol{L}]\right) \in \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{L}}$
- **Goal: If $x_1 = x_2 = \cdots = x_{n/2} = \ell$, then estimate the average $\frac{2}{n}\sum_{j=n/2}^{n} x_j[\ell]$**

<u>Observation:</u> Can solve easily under LDP with two rounds (provided that $n \gtrsim \frac{1}{\varepsilon}\log L$):

A simple pointer chasing problem

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$, where:
  - Every user $\mathbf{1} \le \boldsymbol{i} < \boldsymbol{n}/\mathbf{2}$ holds input $\boldsymbol{x_i} \in \{\mathbf{1}, \mathbf{2}, \dots, \boldsymbol{L}\}$
  - Every user $\boldsymbol{n}/\mathbf{2} \le \boldsymbol{j} \le \boldsymbol{n}$ holds input $\boldsymbol{x_j} = \left(\boldsymbol{x_j}[\mathbf{1}], \dots, \boldsymbol{x_j}[\boldsymbol{L}]\right) \in \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{L}}$
- **Goal: If $x_1 = x_2 = \cdots = x_{n/2} = \ell$, then estimate the average $\frac{2}{n}\sum_{j=n/2}^{n} x_j[\ell]$**

<u>Observation:</u> Can solve easily under LDP with two rounds (provided that $n \gtrsim \frac{1}{\varepsilon} \log L$):

- First run an LDP protocol for histograms over users $1 \le i < \frac{n}{2}$ to identify ℓ (if exists)
- Then run an LDP averaging protocol over the ℓth coordinate of users $\frac{n}{2} \le i \le n$

A simple pointer chasing problem

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$, where:
  - Every user $\mathbf{1} \le \boldsymbol{i} < \boldsymbol{n}/\mathbf{2}$ holds input $\boldsymbol{x_i} \in \{\mathbf{1}, \mathbf{2}, \dots, \boldsymbol{L}\}$
  - Every user $\boldsymbol{n}/\mathbf{2} \le \boldsymbol{j} \le \boldsymbol{n}$ holds input $\boldsymbol{x_j} = \left(\boldsymbol{x_j}[\mathbf{1}], \dots, \boldsymbol{x_j}[\boldsymbol{L}]\right) \in \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{L}}$
- **Goal: If $x_1 = x_2 = \cdots = x_{n/2} = \ell$, then estimate the average $\frac{2}{n}\sum_{j=n/2}^{n} x_j[\ell]$**

<u>Observation:</u> Can solve easily under LDP with two rounds (provided that $\boldsymbol{n} \gtrsim \frac{\mathbf{1}}{\boldsymbol{\varepsilon}} \log \boldsymbol{L}$):

- First run an LDP protocol for histograms over users $1 \le i < \frac{n}{2}$ to identify ℓ (if exists)
- Then run an LDP averaging protocol over the ℓth coordinate of users $\frac{n}{2} \le i \le n$

<u>Theorem:</u> Cannot solve under LDP with one round (unless n is MUCH larger)

A simple pointer chasing problem

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$, where:
  - Every user $\mathbf{1} \leq \boldsymbol{i} < \boldsymbol{n}/\mathbf{2}$ holds input $\boldsymbol{x_i} \in \{\mathbf{1}, \mathbf{2}, \dots, \boldsymbol{L}\}$
  - Every user $\boldsymbol{n}/\mathbf{2} \leq \boldsymbol{j} \leq \boldsymbol{n}$ holds input $\boldsymbol{x_j} = \left(\boldsymbol{x_j}[\mathbf{1}], \dots, \boldsymbol{x_j}[\boldsymbol{L}]\right) \in \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{L}}$
- **Goal: If $x_1 = x_2 = \cdots = x_{n/2} = \ell$, then estimate the average $\frac{2}{n} \sum_{j=n/2}^{n} x_j[\ell]$**

<u>Observation:</u> Can solve easily under LDP with two rounds (provided that $\boldsymbol{n} \gtrsim \frac{\mathbf{1}}{\boldsymbol{\varepsilon}} \log \boldsymbol{L}$):

- First run an LDP protocol for histograms over users $1 \leq i < \frac{n}{2}$ to identify ℓ (if exists)
- Then run an LDP averaging protocol over the ℓth coordinate of users $\frac{n}{2} \leq i \leq n$

<u>Theorem:</u> Cannot solve under LDP with one round (unless n is MUCH larger)

<u>Proof idea:</u> If there is a non-interactive LDP protocol Π for this problem, then there is an LDP protocol for computing the averages of **all** $\boldsymbol{L}$ coordinates of the $\boldsymbol{x_j}$'s, which cannot exist.

A simple pointer chasing problem

- Distributed database $\boldsymbol{S} = (\boldsymbol{x_1}, \dots, \boldsymbol{x_n})$, where:
  - Every user $\mathbf{1} \leq \boldsymbol{i} < \boldsymbol{n}/\mathbf{2}$ holds input $\boldsymbol{x_i} \in \{\mathbf{1}, \mathbf{2}, \dots, \boldsymbol{L}\}$
  - Every user $\boldsymbol{n}/\mathbf{2} \leq \boldsymbol{j} \leq \boldsymbol{n}$ holds input $\boldsymbol{x_j} = \left(\boldsymbol{x_j}[\mathbf{1}], \dots, \boldsymbol{x_j}[\boldsymbol{L}]\right) \in \{\mathbf{0}, \mathbf{1}\}^{\boldsymbol{L}}$
- **Goal: If $x_1 = x_2 = \cdots = x_{n/2} = \ell$, then estimate the average $\frac{2}{n}\sum_{j=n/2}^{n} x_j[\ell]$**

Observation: Can solve easily under LDP with two rounds (provided that $n \gtrsim \frac{1}{\varepsilon}\log L$):

- First run an LDP protocol for histograms over users $1 \leq i < \frac{n}{2}$ to identify ℓ (if exists)
- Then run an LDP averaging protocol over the ℓth coordinate of users $\frac{n}{2} \leq i \leq n$

Theorem: Cannot solve under LDP with one round (unless n is MUCH larger)

Proof idea: If there is a non-interactive LDP protocol Π for this problem, then there is an LDP protocol for computing the averages of **all** $\boldsymbol{L}$ coordinates of the $\boldsymbol{x_j}$'s, which cannot exist.

The protocol:
(1) Execute Π on the $\boldsymbol{x_j}$'s and obtain their messages.
(2) For every $1 \leq \ell \leq L$, simulate the x_i users in Π on input $x_i = \ell$, to obtain estimation for the ℓth oordinate

The Local Model of Differential Privacy

Today's Outline

1. What is the model?
2. Computing histograms
3. Computing averages
4. Clustering
5. LDP vs. statistical queries
6. Impossibility result for histograms
7. Interactive LDP protocols

Summary

- LDP provides strong privacy and trust guarantees:
  - ✓ No individual information is being collected
  - ✓ Privacy preserved even if the organization is subpoenaed
- Many tasks are compatible with LDP:
  - ✓ Histograms, Averages, Clustering, …
- Accuracy is generally reduced compared to the centralized model

Questions?