Private Information Retrieval

Dima Kogan

BIU Winter School on Cryptography, January 2022

Private information retrieval [CGKS95]

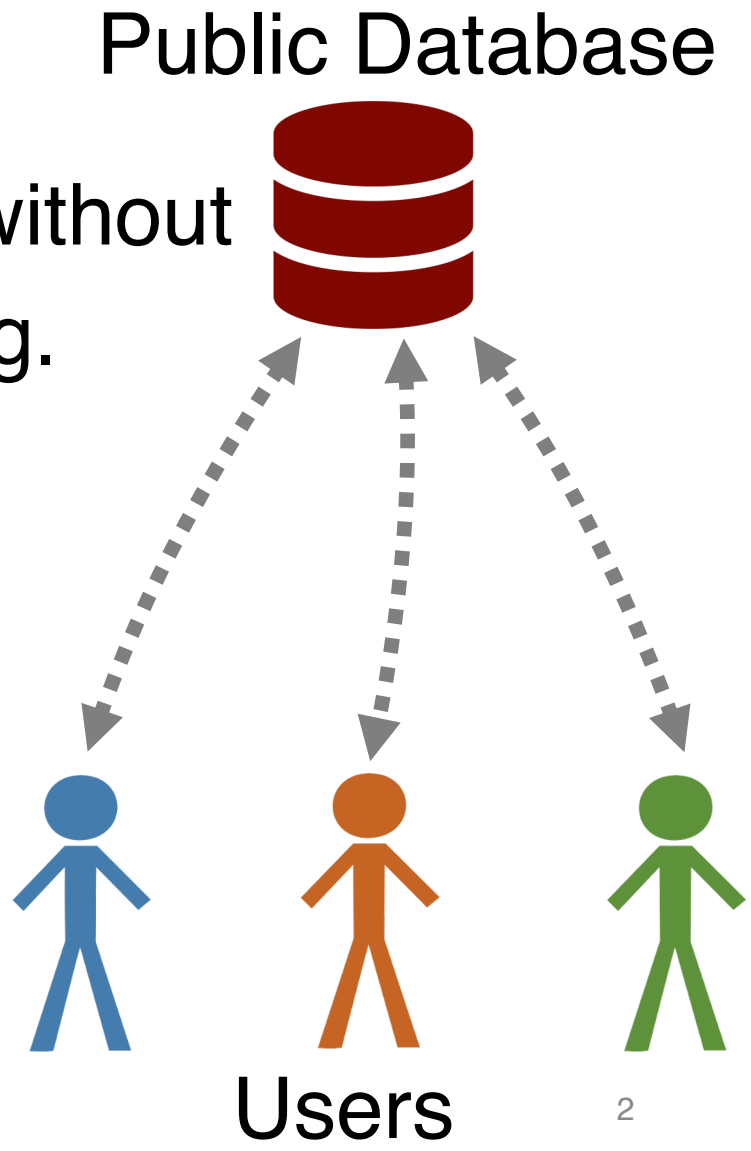
Goal

Users can read records from a public DB without DB learning which records they are reading.

Applications

medical encyclopedia, stocks

privacy-preserving systems



Part I: introduction to PIR

The model

Basic schemes

Extensions

Part II: reducing computation in PIR

PIR requirements

Correctness

Client learns its bit of interest (with overwhelming prob.)

Security

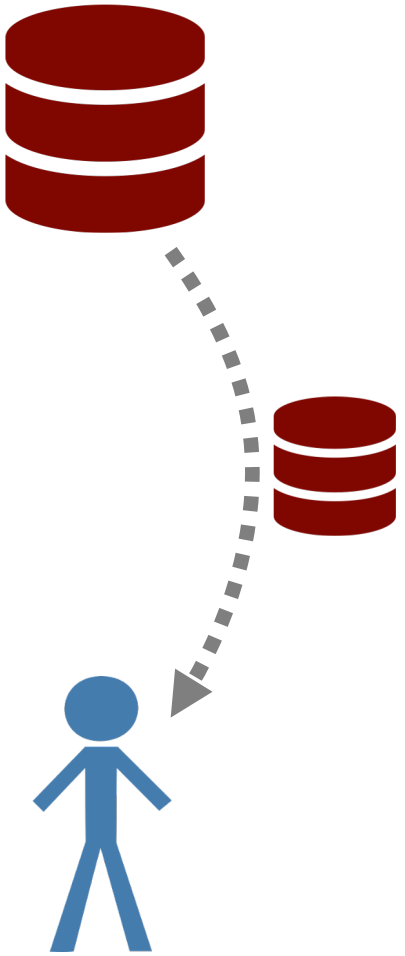
(Malicious) server “learns nothing” about client’s desired bit

Naïve solution

Client downloads entire DB and queries it locally

Perfect correctness and privacy

Downside: large communication cost



PIR requirements

Correctness

Client learns its bit of interest (with overwhelming prob.)

Security

(Malicious) server “learns nothing” about client’s desired bit

Minimize communication

Main approaches to PIR

Multi-server PIR [CGKS95]

Replicate DB on non-colluding servers

Single-server PIR [KO97]

Requires cryptographic assumptions

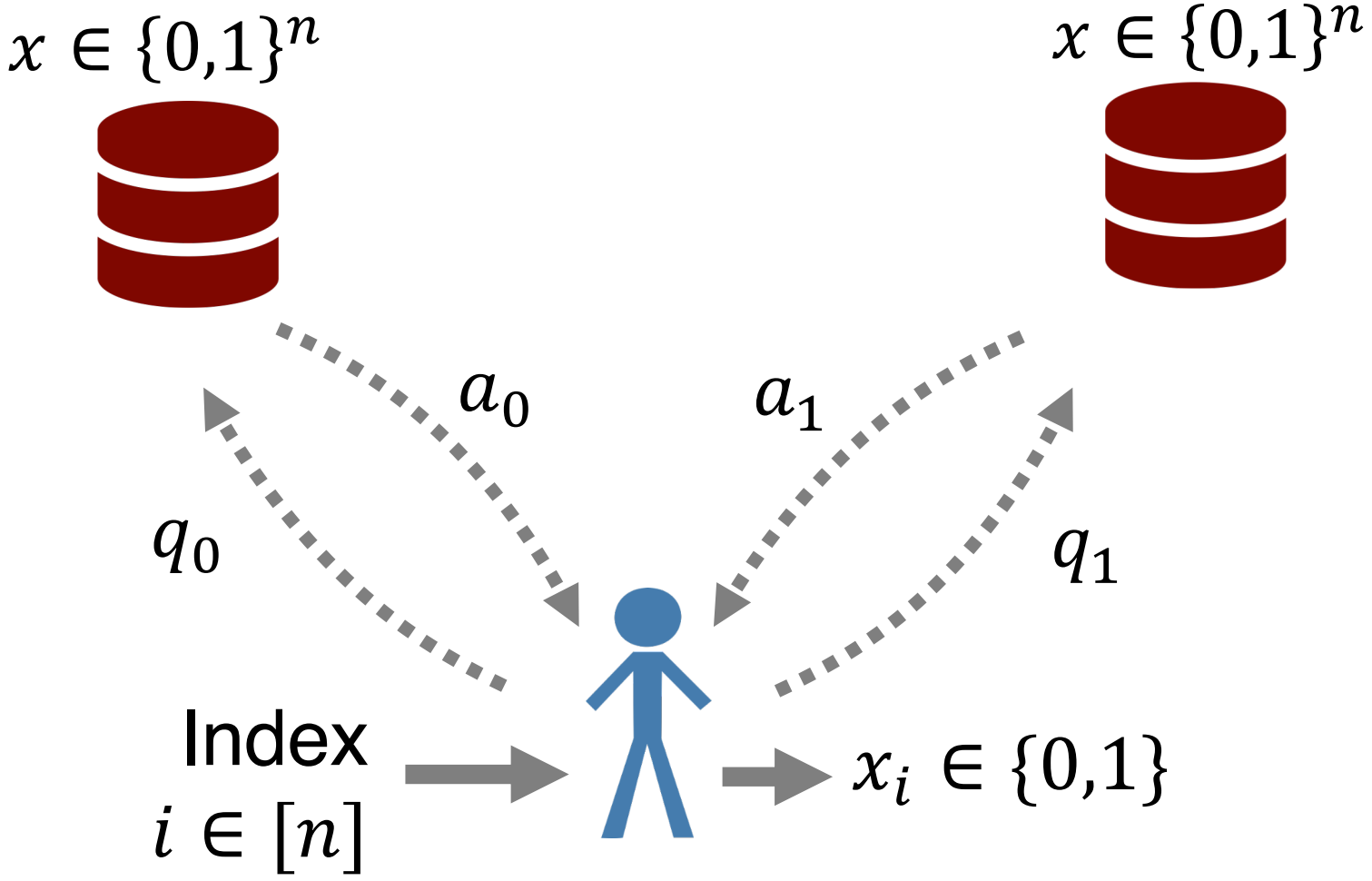
2-server PIR

Three efficient algorithms:

$\text{Query}(1^n, i) \to (q_0, q_1)$

$\text{Answer}(x, q) \to a$

$\text{Reconstruct}(a_0, a_1) \to x_i$



PIR requirements

Correctness

Client learns its bit of interest: $\forall n \in \mathbb{N}, x \in \{0,1\}^n, i \in [n]$:

$$\Pr\left[\text{Reconstruct}(a_0, a_1) = x_i : \begin{array}{r} q_0, q_1 \leftarrow \text{Query}(1^n, i) \\ a_0 \leftarrow \text{Answer}(x, q_0) \\ a_1 \leftarrow \text{Answer}(x, q_1) \end{array}\right] = 1$$

PIR requirements

Security

(Malicious) server "learns nothing" about client's desired bit

$\forall n \in \mathbb{N}, i, j \in [n], s \in \{0,1\},$

$$\{\boldsymbol{q_s}: q_0, q_1 \leftarrow \mathrm{Query}(1^n, i)\} \approx_c \{\boldsymbol{q_s}: q_0, q_1 \leftarrow \mathrm{Query}(1^n, j)\}$$

$$\left\{\begin{array}{c}\text{View of server } s \text{ when}\\ \text{client reads bit } i\end{array}\right\} \approx_c \left\{\begin{array}{c}\text{View of server } s \text{ when}\\ \text{client reads bit } j\end{array}\right\}$$

Non-colluding servers

Can be ≡ for perfect security

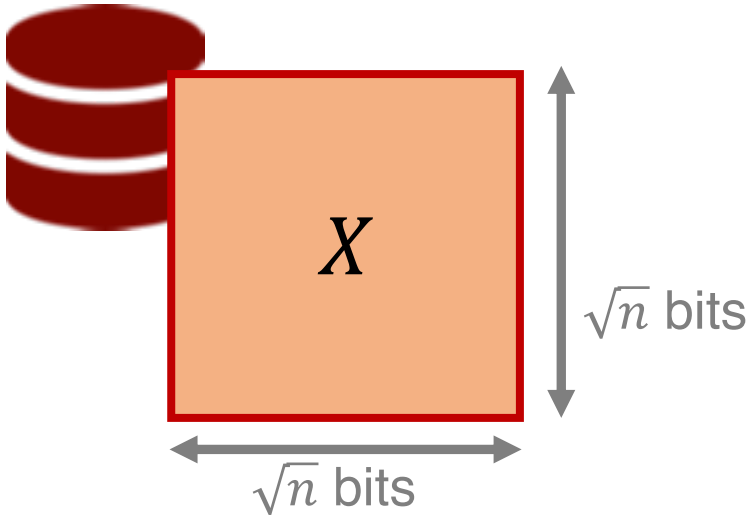
Part I: introduction to PIR

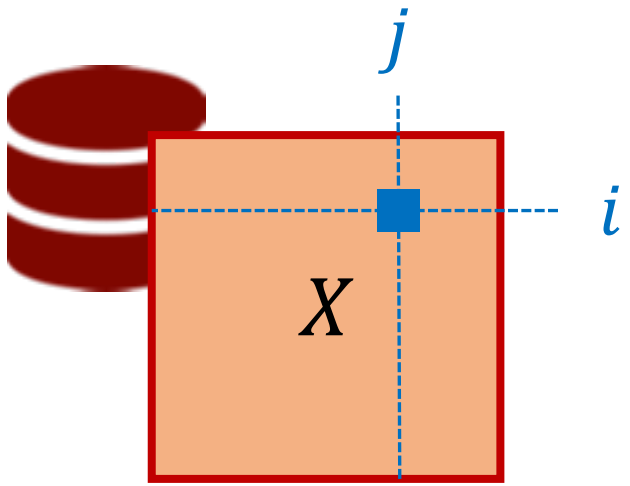
The model

Basic schemes

Extensions

A two-server PIR scheme [CGKS95]

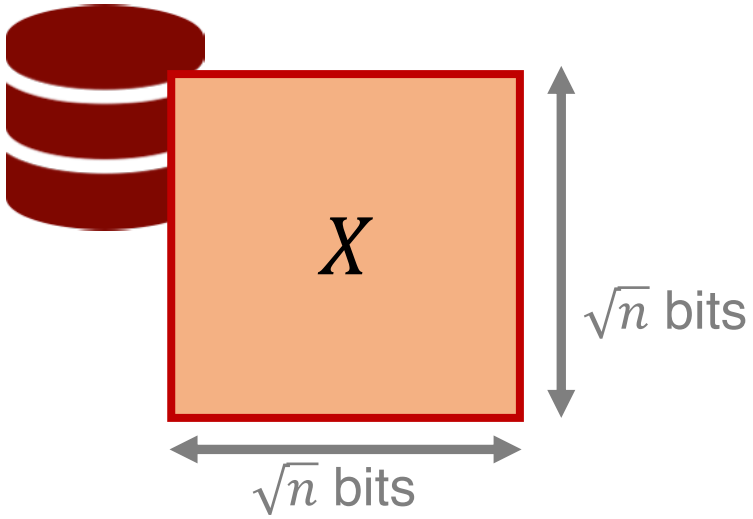


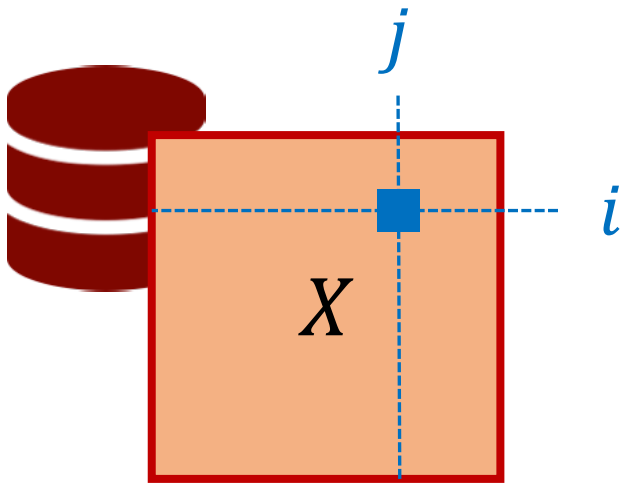


View DB $x \in \{0,1\}^n$ as matrix $X \in \mathbb{Z}_2^{\sqrt{n} \times \sqrt{n}}$

Client wants to read $(i, j) \in [\sqrt{n}] \times [\sqrt{n}]$

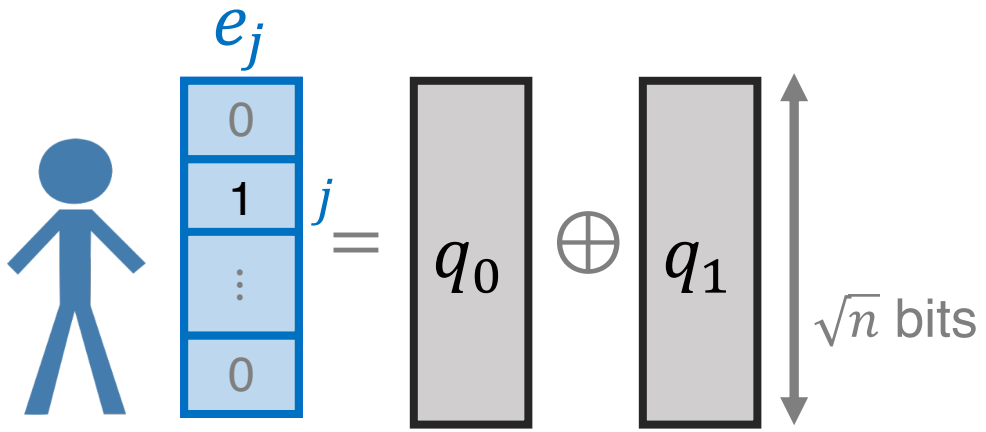
A two-server PIR scheme



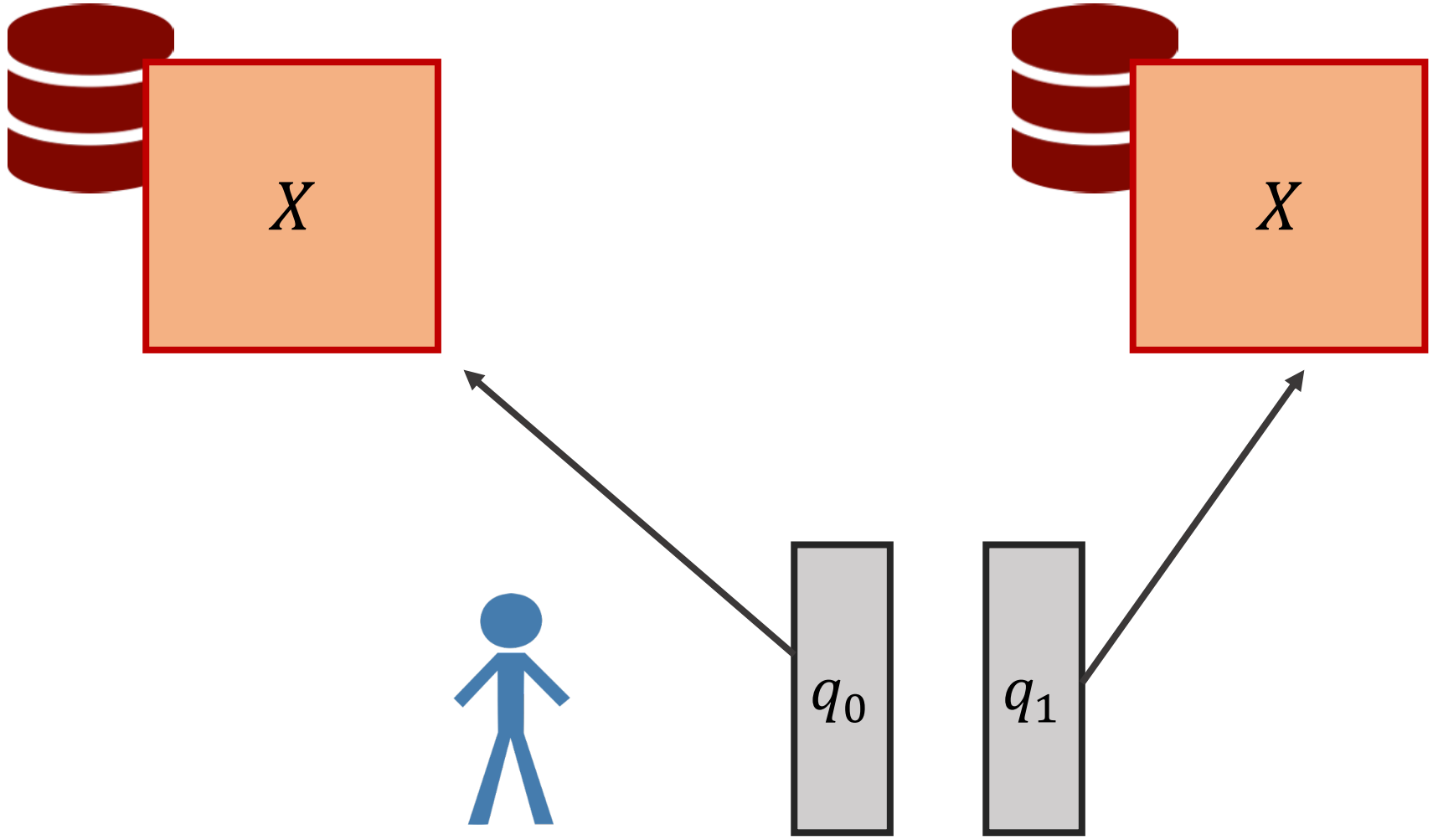


$\text{Query}(1^n, (i, j))$:

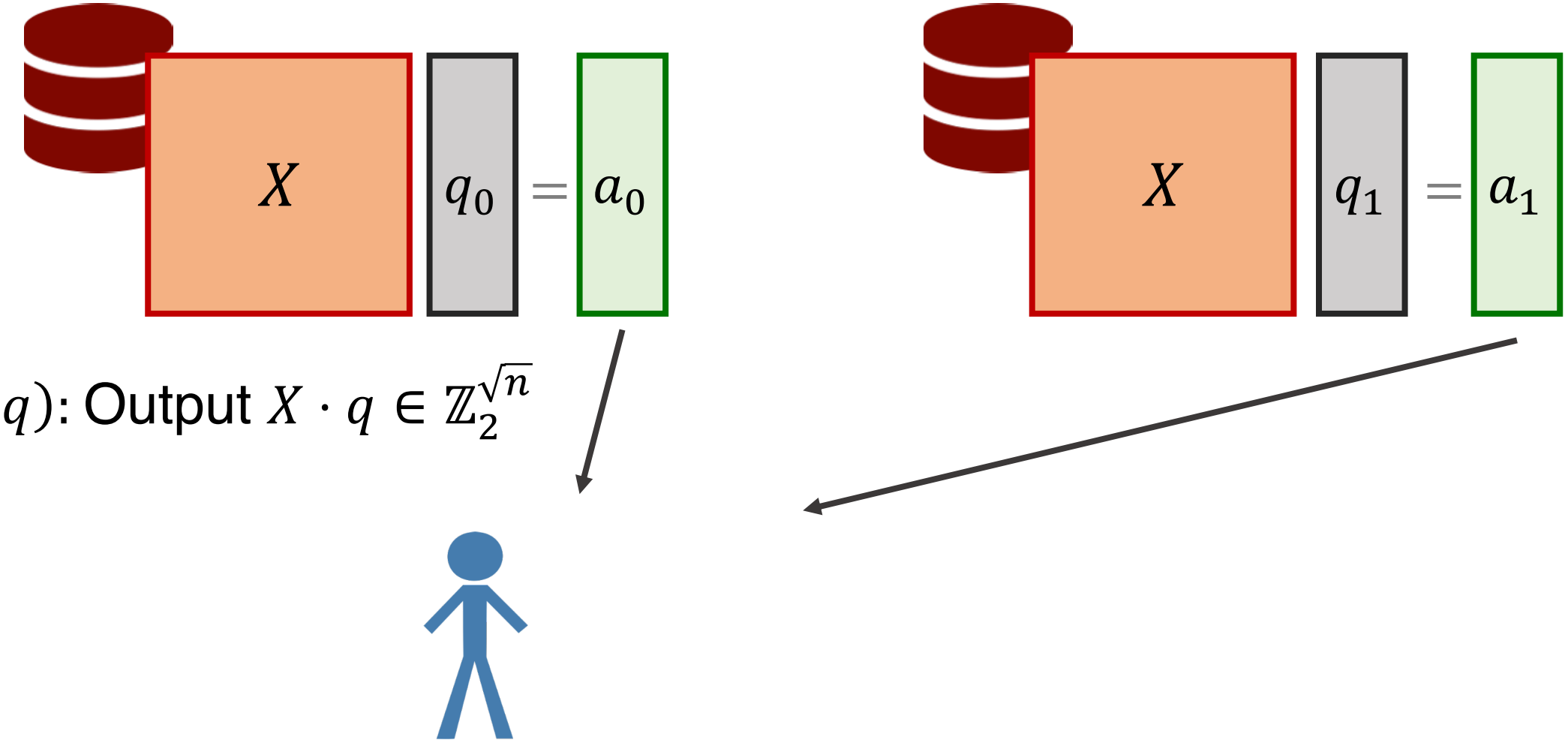
Sample $q_0, q_1 \leftarrow_R \mathbb{Z}_2^{\sqrt{n}}$ s.t. $q_0 + q_1 = e_j$



A two-server PIR scheme

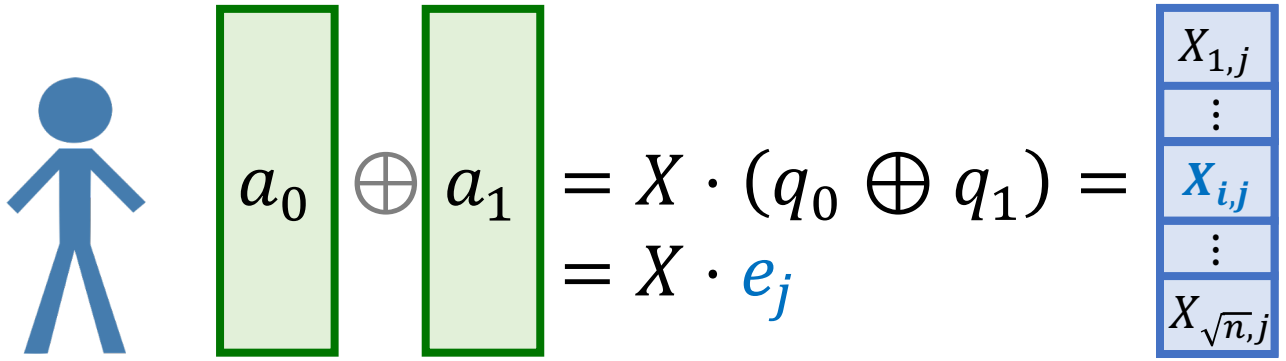


A two-server PIR scheme



A two-server PIR scheme

Reconstruct(a_0, a_1): Output $(a_1 + a_1)_i$



A two-server PIR scheme [CGKS95]

Correctness:

$$(a_0 \oplus a_1)_j = (Xq_0 \oplus Xq_1)_i = \left(X(q_0 \oplus q_1)\right)_i = \left(Xe_j\right)_i = x_{ij}$$

Security:

Each of q_0, q_1 on its own is a random vector, independent of client's input (i, j)

Efficiency: $|q_0| + |q_1| + |a_0| + |a_1| = 4\sqrt{n}$

Can reduce to $O(\log n)$ using Distributed Point Functions.
Security then computational.

The state of PIR

Multi-server PIR

[CGKS95,Amb97,CG97,BI01,BIKR02,Yek08,Efr12,…]):

- Information-theoretic security: $n^{o(1)}$ communication [DG16]
- Computational security: $O(\log n)$ communication [GI14, BGI15]

Single-server PIR [KO97]

Single-server PIR [KO97]

Idea: Encrypt query instead of secret sharing it

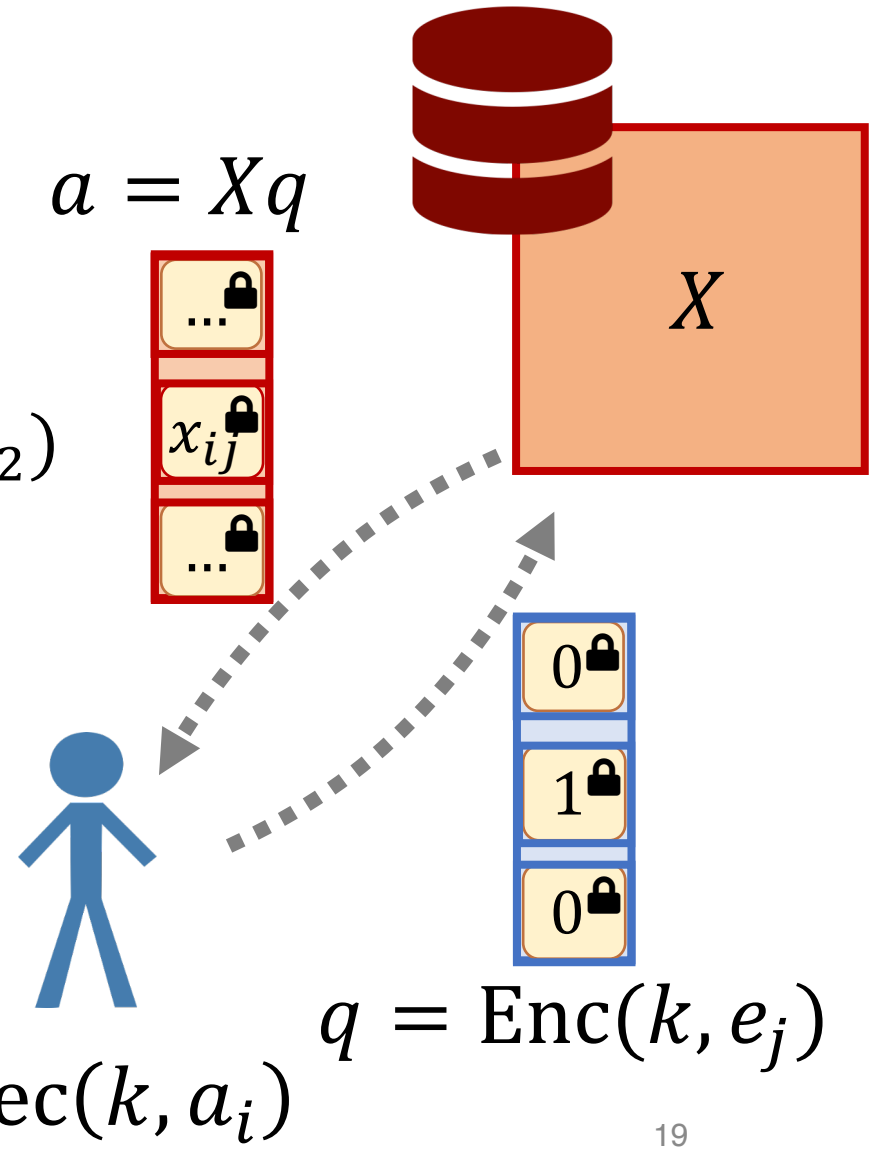
Uses: Linearly Homomorphic Encryption

$$\mathrm{Enc}(k, m_1) + \mathrm{Enc}(k, m_2) = \mathrm{Enc}(k, m_1 + m_2)$$

Can build from DDH, QR, LWE,…

Security: follows from security of LHE

Communication: $2\sqrt{n}$ ciphertexts



The state of PIR

Multi-server PIR [CGKS95]

- Replicate DB on non-colluding servers
- State of the art (following [Amb97,CG97,BI01,BIKR02,Yek08,Efr12,…]):
  - Information-theoretic security: $n^{o(1)}$ communication [DG16]
  - Computational security: $O(\log n)$ communication [GI14, BGI15]

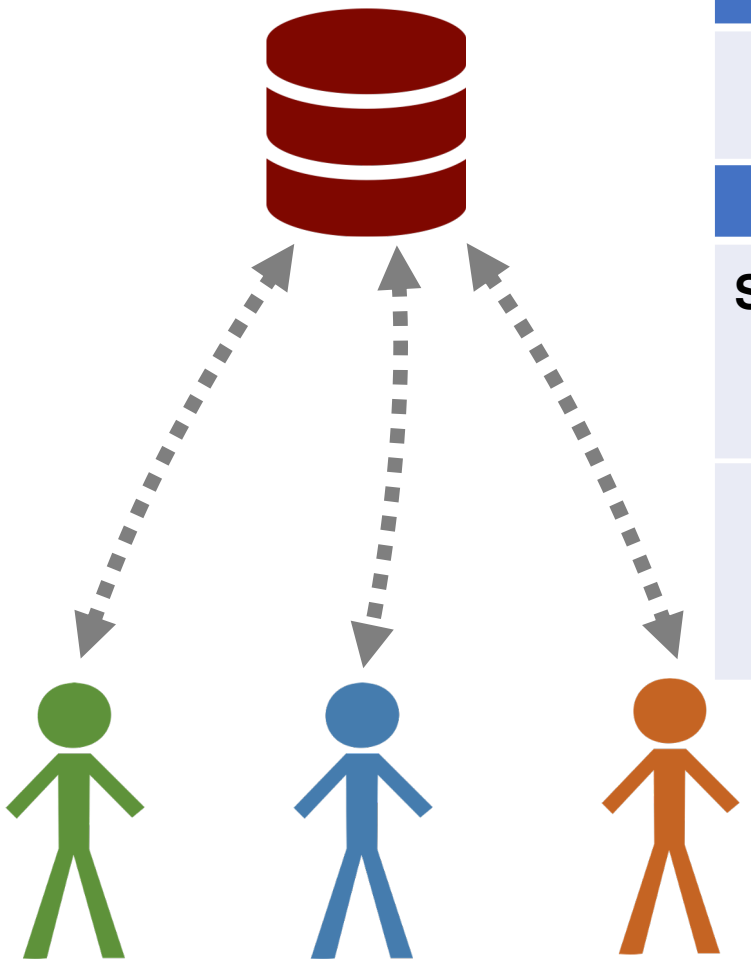
Single-server PIR [KO97]

- Requires cryptographic assumptions
- State of the art:
  - $\mathrm{polylog}(n)$ communication [CMS99, Lip05,…]

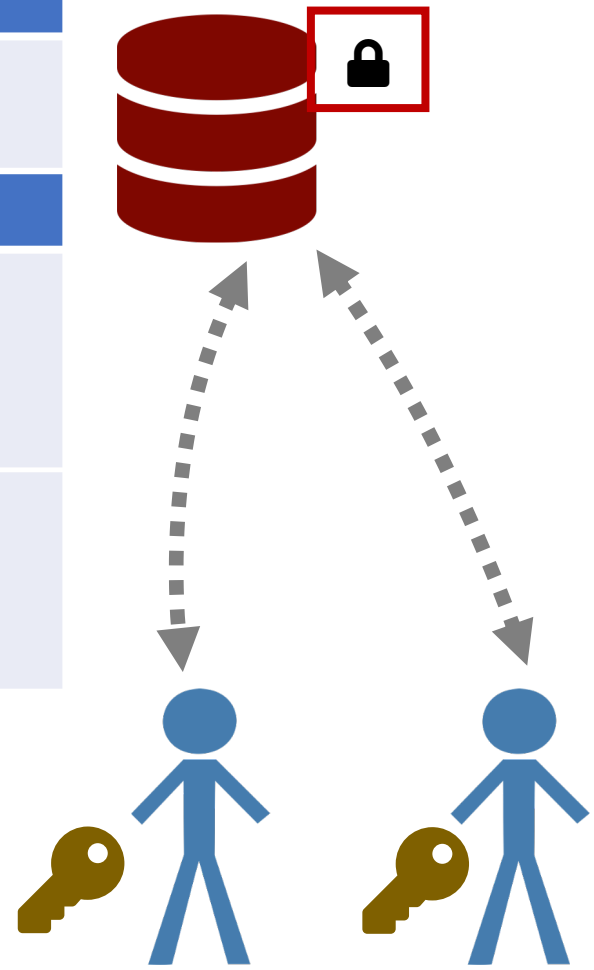
PIR (today) vs ORAM (Sunday)

private access to **public** data

private access to **private** data



Similarity	
both hide the data access pattern	
Differences	
Supports many clients No need in trust or coordination	**Single client** Operations require private state
Read only	**Read & write**



Part I: introduction to PIR

The model

Basic schemes

Extensions

Making PIR more realistic…

Most interesting databases are not n-bit arrays

- Extension 1: longer DB rows
- Extension 2: key-value lookups

Servers also have privacy requirements

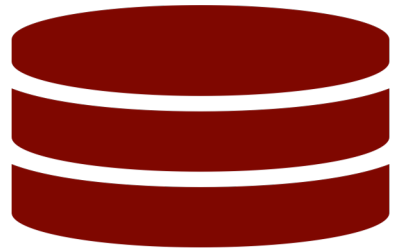
- Extension 3: privacy for the server

Extension 1: longer DB records

On the same server(s)

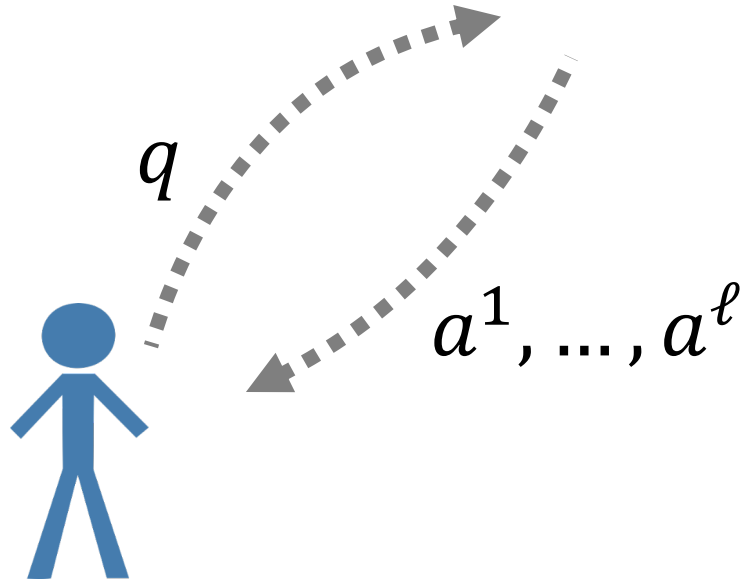
Want: PIR for ℓ-bit records: $\mathrm{X} \in \{0,1\}^{\mathrm{n}\times\ell}$

$X^1, \ldots X^\ell \in \{0,1\}^n$



Solution: query each bit of the record using PIR for 1-bit records

Communication: $\mathrm{U}(\mathrm{n}) + \ell \cdot \mathrm{D}(\mathrm{n})$



Reconstruct bit by bit

Extension 2: PIR by keywords [CGN98]

Standard PIR: DB is array $x \in \{0,1\}^{n \times \ell}$

Many applications want a **set** $S = \{k_1, \ldots, k_n\}$

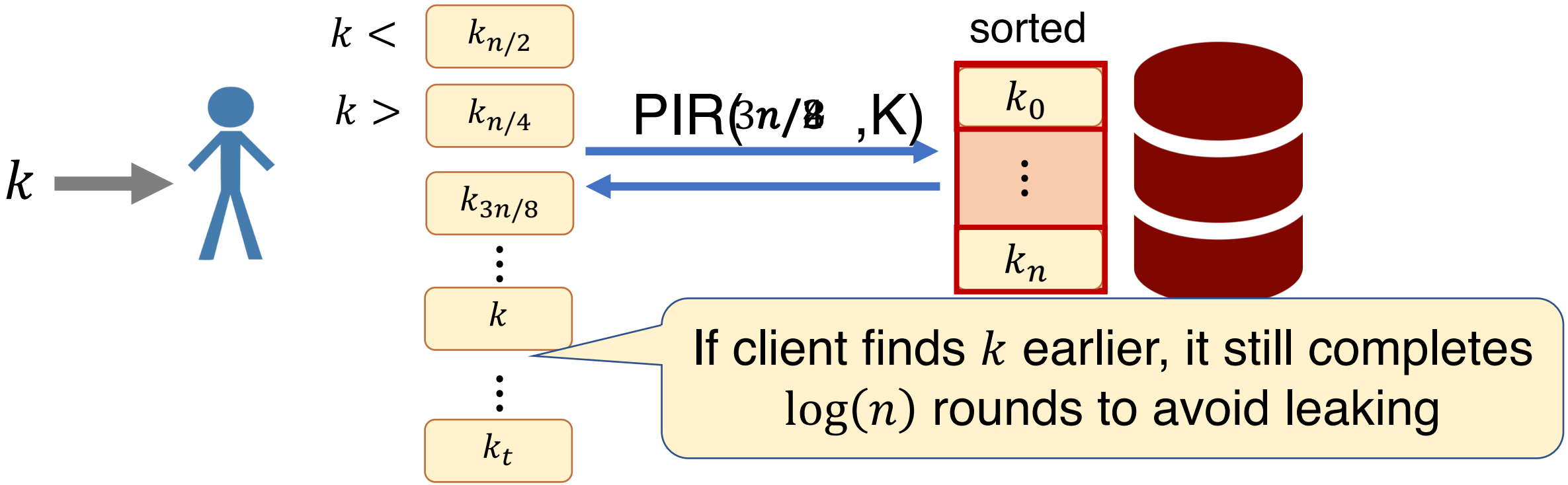
or a **dictionary** $D = \{(k_1, v_1), \ldots, (k_n, v_n)\}$

Idea:

- Build a standard data structure (e.g., binary search tree, hash table,…)
- Wrap each probe into data-structure with PIR

Extensions : PIR by keywords [CGN98]

Simple example: store set S as a sorted array
Client runs binary search for k using PIR



Efficiency: $\log(n)$ PIR queries (can reduce to 2 queries by using perfect hashing)

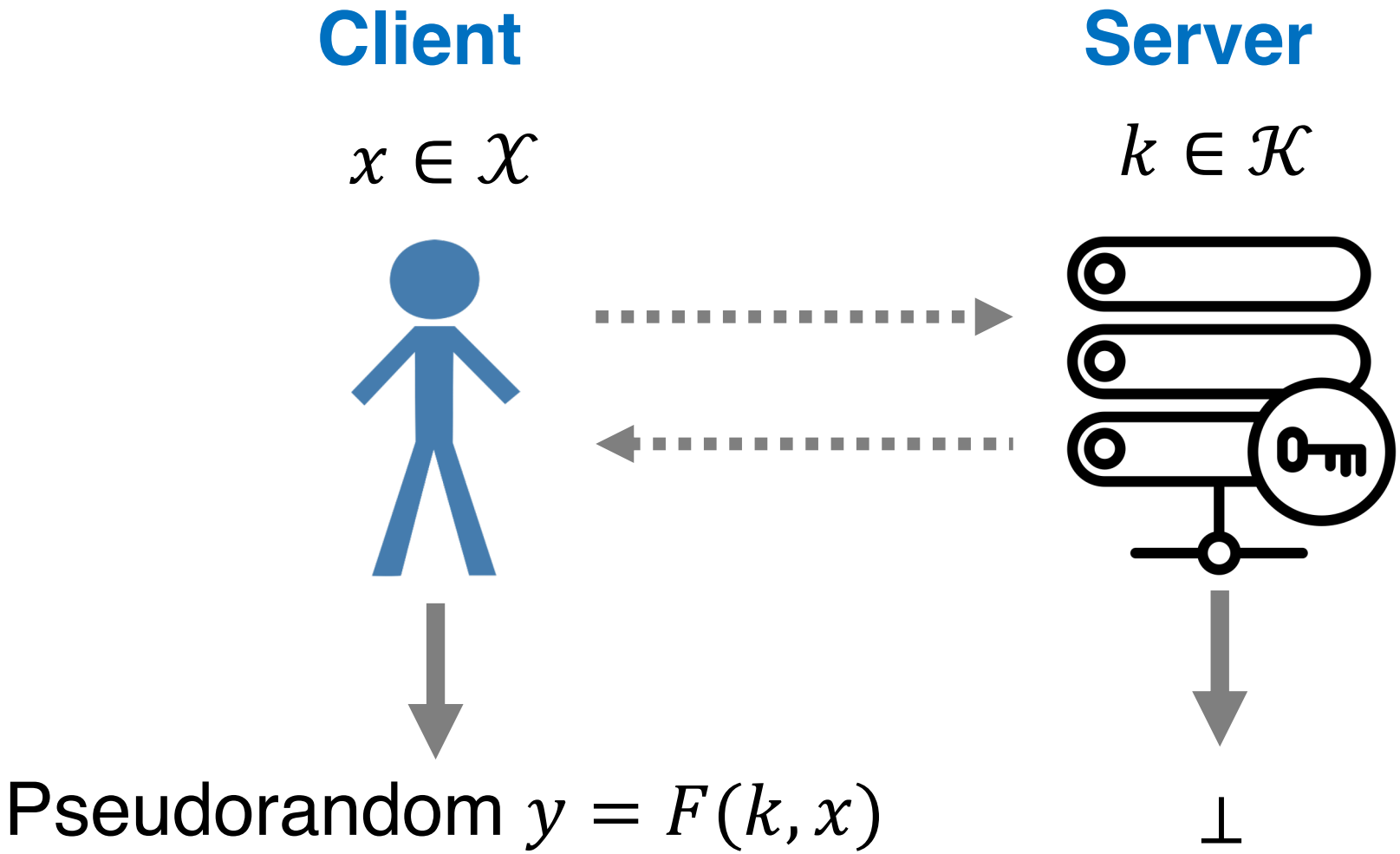
Extension 3: symmetric PIR [GIKM97]

Symmetric PIR – privacy also for the server

Privacy requirement (informal)**:** after each execution of the protocol, the client should learn at most one record in the database

Tool: Oblivious Pseudorandom Function (OPRF)

Tool: Oblivious PRF [NR97,FIPR05]



Symmetric PIR from OPRF [FIPR05]

$E_i = Enc(s_i, x_i)$

$Enc(s_j, x_j)$

$Enc(s_j, x_j)$

s_i

$x_i \leftarrow \mathrm{Dec}(s_i, E_i)$

PIR$(i; E)$

OPRF$(i; k)$

$\mathrm{E_0} = \mathrm{Enc}(s_0, x_0)$

$\mathrm{E_n} = \mathrm{Enc}(s_n, x_n)$

$\mathrm{s_i} \leftarrow \mathrm{F(k, i)}$

$k \leftarrow_R \mathcal{K}$

Client's privacy follows from security of PIR + OPRF

Server's privacy follows from semantic sec. + OPRF

Modern PIR requires lots of **computation**

Server linearly scans the entire DB to respond to a query

⇒a barrier to deployment

Server **must do $\Omega(n)$ work** to respond to a query [BIM04]

- Intuition: If server doesn't touch bit i, client isn't reading bit i
- Holds even if you have **many non-colluding servers**
- Holds irrespective of **cryptographic assumptions**

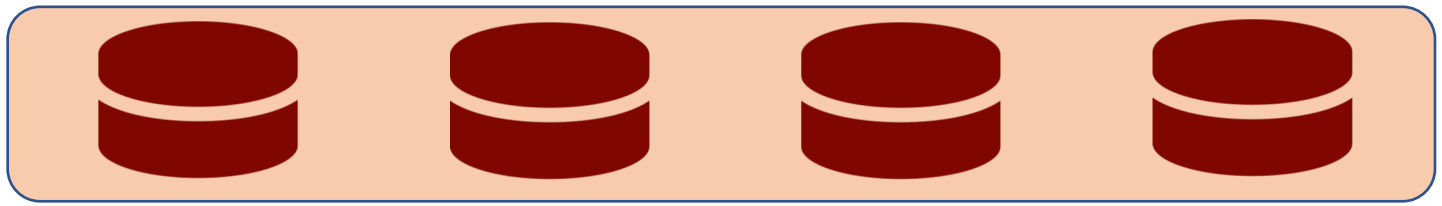
Part II: reducing computation in PIR

Two classic approaches

Recent approach: offline/online PIR

Batch PIR [IKOS04, IKOS06, LG15, Hen16, ACLS18]

$i_1, \ldots, i_Q$



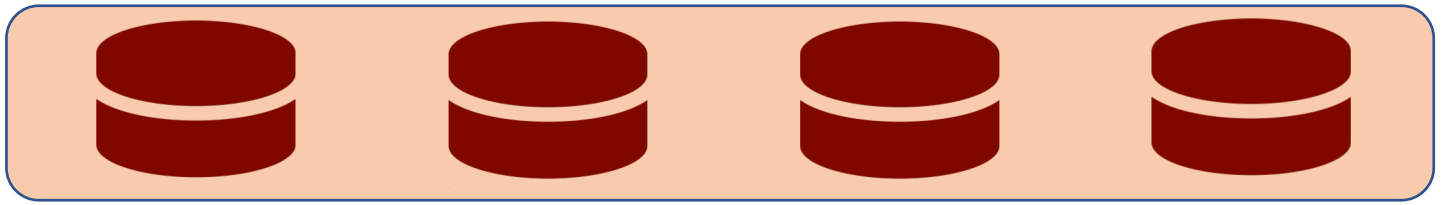
Server & client agree on a random partition of DB into Q buckets

For $i_1, \ldots, i_Q \in [n]$, w.h.p. no bucket contains more than $\lambda \log n$ indices

Client queries each bucket $\lambda \log n$ times

Batch PIR [IKOS04, IKOS06, LG15, Hen16, ACLS18]

$x \in \{0,1\}^{12}$
$Q = 4$



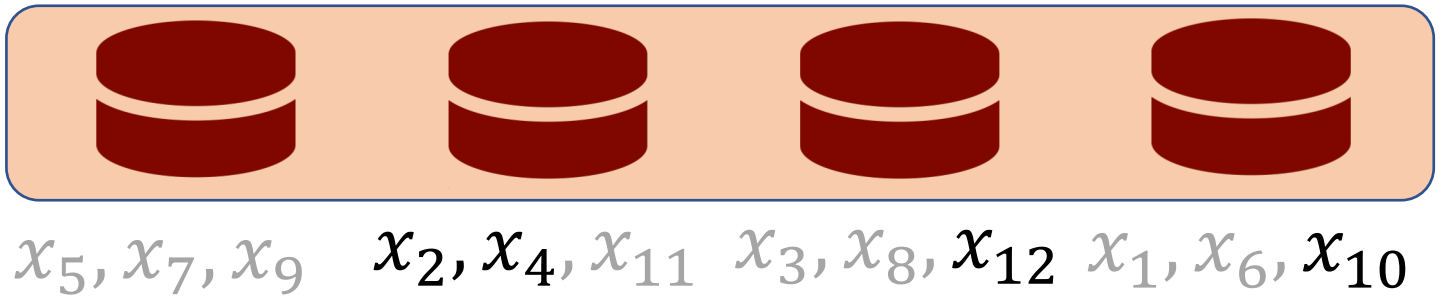
Partition using hash function h

x_5, x_7, x_9 x_2, x_4, x_{11} x_3, x_8, x_{12} x_1, x_6, x_{10}

description of h

Batch PIR [IKOS04, IKOS06, LG15, Hen16, ACLS18]

$x \in \{0,1\}^{12}$
$Q = 4$



Only works if client makes a non-adaptive batch of queries

10,2,4,12

To Bucket 1	To Bucket 2	To Bucket 3	To Bucket 4
$q(1), q(1)$	$q(1), q(2)$	$q(3), q(1)$	$q(3), q(1)$

Batch PIR – analysis

Correctness:

W.h.p each bucket contains $\leq \lambda \log n$ input indices

Correctness follows from correctness of base PIR scheme

Security: follows from security of base PIR scheme (server sees $\lambda \log n$ queries for each bucket, regardless of client's inputs)

Efficiency:

#buckets

time/query

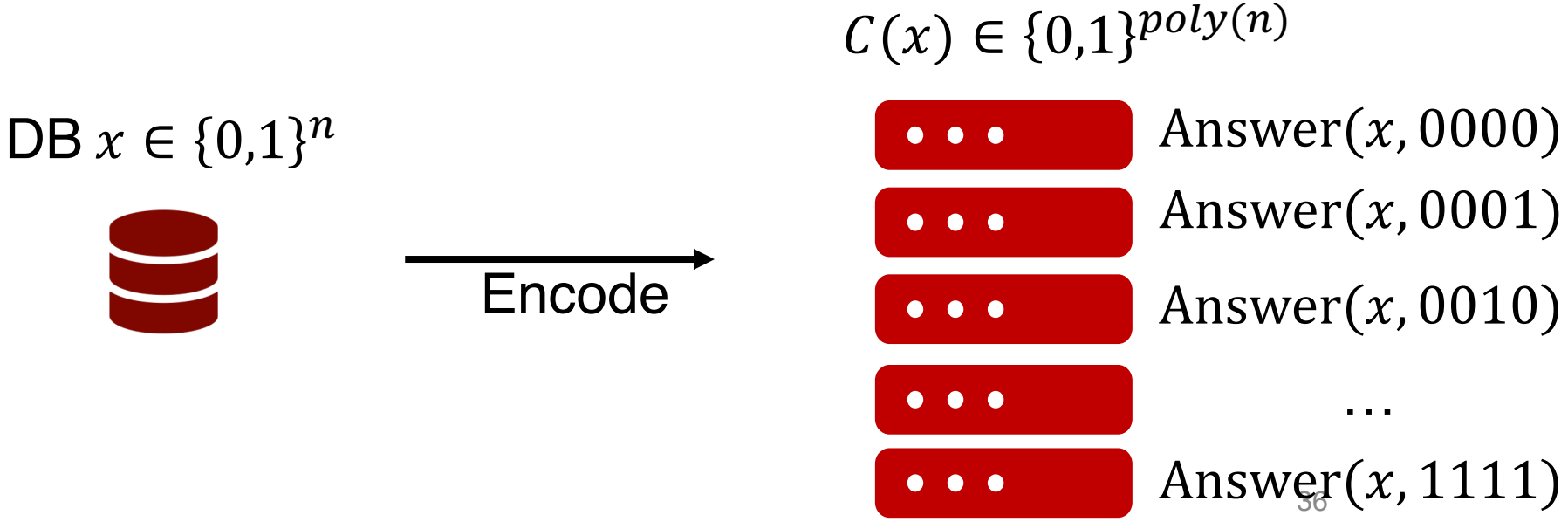
indep. of Q

Server time: $Q \cdot \lambda \log n \cdot T\left(\frac{n}{Q}\right) = Q \cdot \lambda \log n \cdot \frac{n}{Q} = n \cdot \lambda \log n$

queries/bucket

PIR with preprocessing [BIM04]

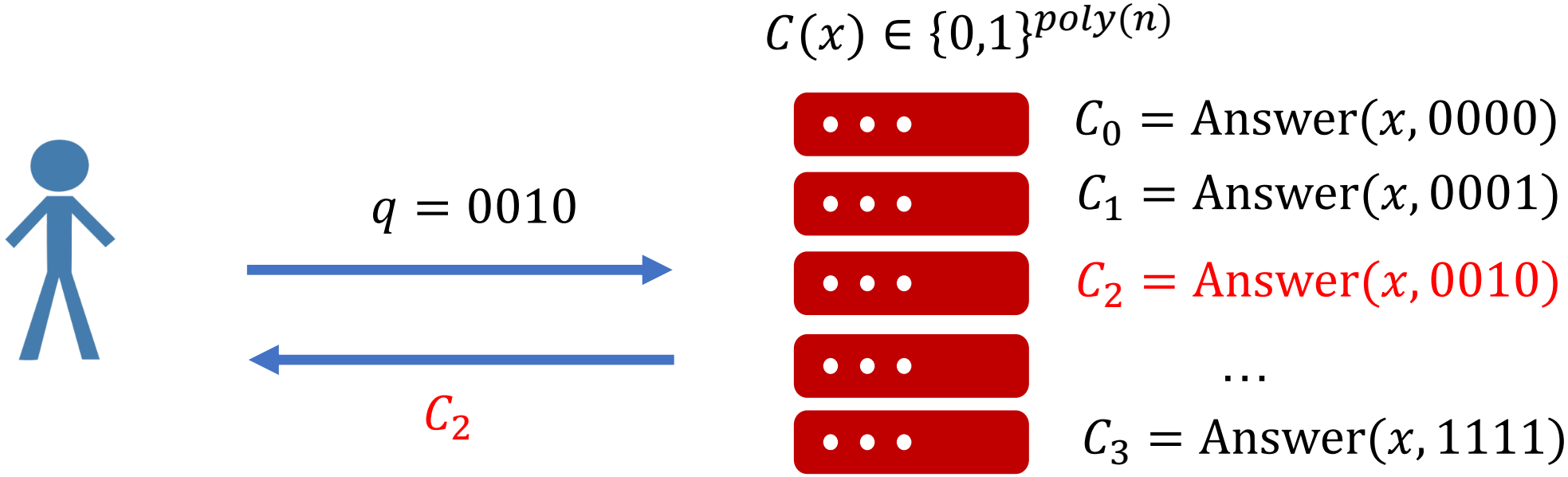
Main idea: servers precompute the responses to all possible queries of a standard PIR scheme.



PIR with preprocessing [BIM04]

Main idea: servers precompute the responses to all possible queries of a standard PIR scheme.

In the online phase: the server looks up the response from a table.



PIR with preprocessing [BIM04]

Base scheme has communication $U(n)$ and $D(n)$

$\Rightarrow$ new scheme with space $S = D(n) \cdot 2^{U(n)}$ and time $T = U(n) + D(n)$

Examples (2-server PIR):

- $\sqrt{n}$ scheme $\Rightarrow$ PIR with preproc. $S = 2^{\sqrt{n}}$ and $T = \sqrt{n}$
- [BI01] $U(n) = O(\log n)$ and $D(n) = n^{1/2+\epsilon}$
  $\Rightarrow$PIR with preproc. $S = \mathrm{poly}(n)$ and $T = n^{1/2+\epsilon}$

Advantage: sublinear server time (e.g., $n^{0.6}$)

Disadvantage: superlinear server storage (e.g., n^3)

Also: DEPIR [BIPW17, CHR17], PANDA [HOWW18] for the 1-server setting

Part II: reducing computation in PIR

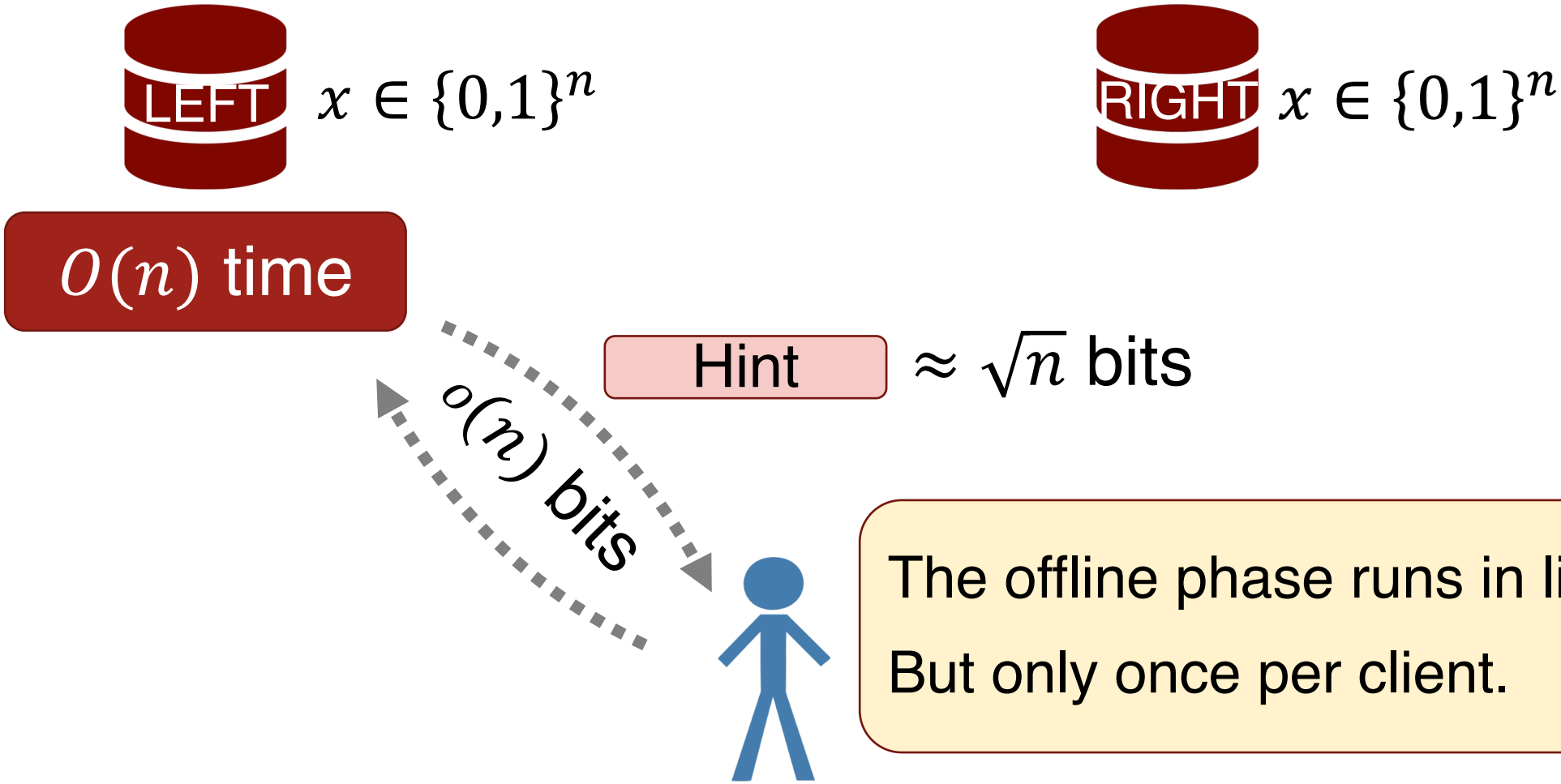
Two classic approaches

Recent approach: offline/online PIR

Based on joint works with Henry Corrigan-Gibbs and Alexandra Henzinger [CK20,KC21,CHK22]

The model

Step 1: Offline phase



The offline phase runs in linear time.

But only once per client.

The model

Step 1: Offline phase

$x \in \{0,1\}^n$

$x \in \{0,1\}^n$

Client stores hint

Hint

The model
Step 2: Online phase – reading x_i

[DIO01, BIM04, BLW17, PPY18]

$x \in \{0,1\}^n$

$x \in \{0,1\}^n$

Sublinear online time $o(n)$ time

Client can repeat online phase to read multiple items **adaptively**

$o(n)$ bits

Index $i \in [n]$

$x_i \in \{0,1\}$

Hint

Offline/online PIR [CK20,KC21,CHK22]

up to $\mathrm{poly}(\lambda, \log n)$ factors for length-n DB and sec. parameter λ

Two-server scheme

- $\sqrt{n}$ communication and online time (from any PRG)
- Client uses $\sqrt{n}$ time and storage (using PRPs)

Offline/online PIR [CK20,KC21,CHK22]

up to $\mathrm{poly}(\lambda, \log n)$ factors for length-n DB and sec. parameter λ

Two-server scheme

- $\sqrt{n}$ communication and online time (from any PRG)
- Client uses $\sqrt{n}$ time and storage (using PRPs)

Single-server scheme

- $n^{3/4}$ online time (from DDH, DCR,…)
- $\sqrt{n}$ from FHE
- No public-key operations in the online phase

Our $\sqrt{n}$ schemes achieve optimal comm–online time tradeoff

Lower bound

- For offline/online schemes that store DB in its original form
- Communication C and online time T must be $C \cdot T \geq n$

Our scheme
Step 1: Offline phase

$x \in \{0,1\}^n$

$x \in \{0,1\}^n$

Computes $h_1, \ldots, h_m \in \{0,1\}$

$$h_j = \sum_{\ell \in S_j} x_\ell \bmod 2$$

$h_1, \ldots, h_m$

$S_1, \ldots, S_m$

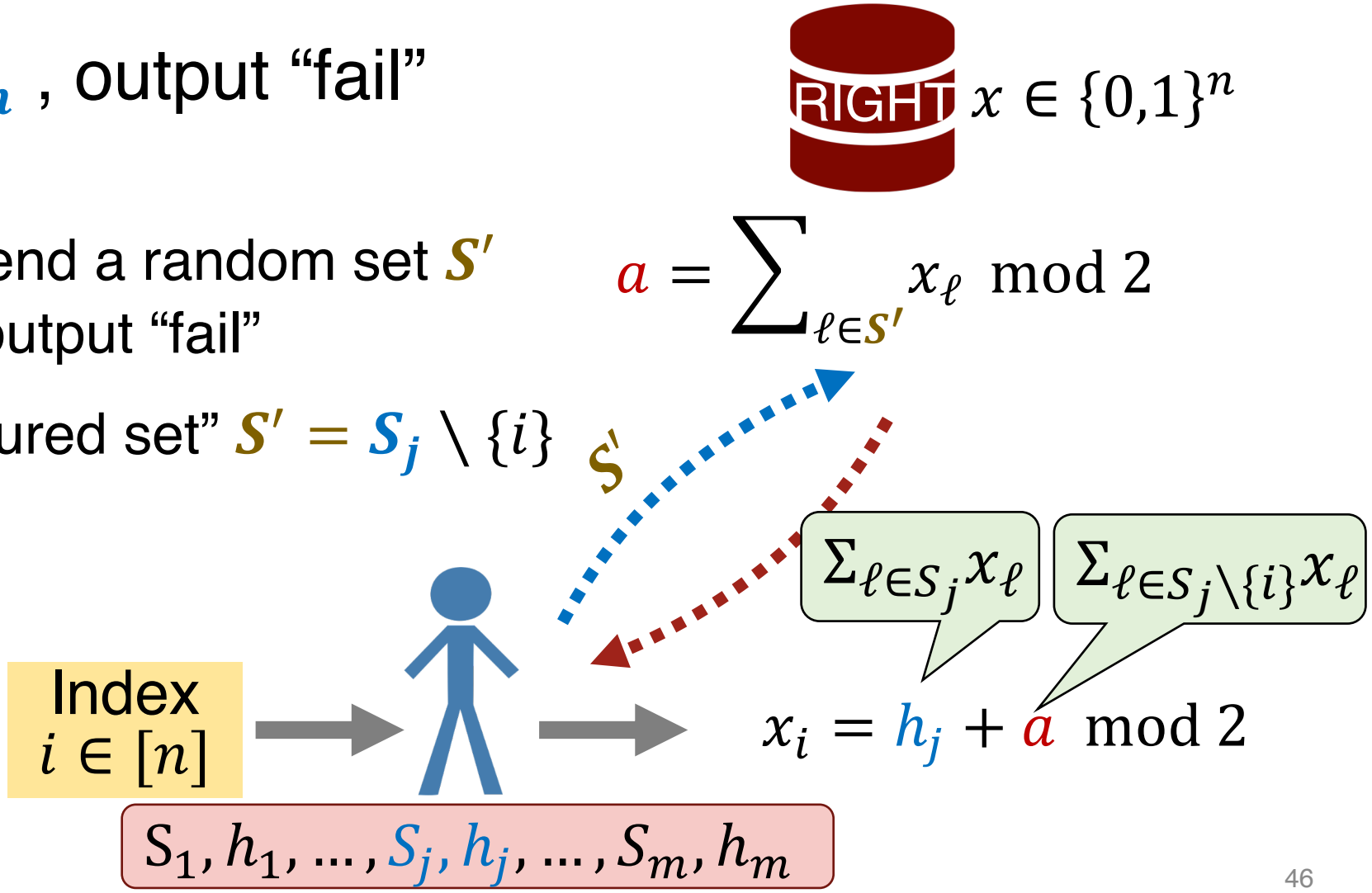
$S_1, h_1, \ldots, S_m, h_m$

Random subsets $S_1, \ldots, S_m \subset [n]$
each of size $|S_j| = \sqrt{n}$

Our scheme
Step 2: Online phase – reading x_i

If $i \notin S_1 \cup \cdots \cup S_m$, output "fail"
Else, $i \in S_j$,

- With prob $\frac{\sqrt{n}-1}{n}$, send a random set S' containing i and output "fail"
- Else, send "punctured set" $S' = S_j \setminus \{i\}$



Our scheme
Correctness

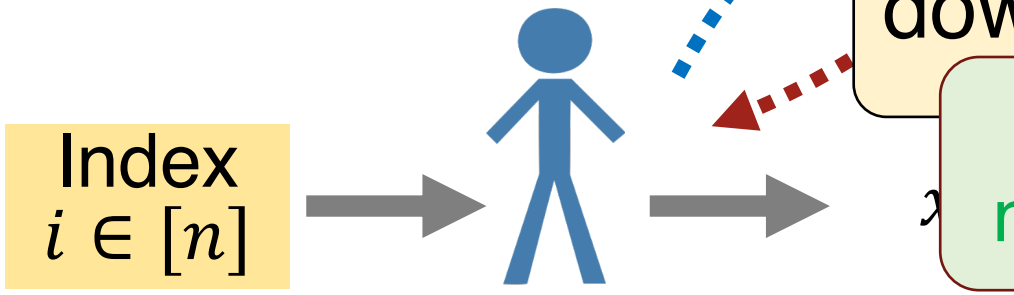
If $i \notin S_1 \cup \cdots \cup S_m$, output "fail"
Else, $i \in S_j$,

- With prob $\frac{\sqrt{n}-1}{n}$, send a random set S' containing i and output "fail"
- Else, send $S' = S_j \setminus \{i\}$

Choose
$$m \approx \lambda\sqrt{n}$$
Then:
$$\Pr[\text{Fail}_1] \leq \text{negl}(\lambda)$$
(λn balls into n bins)

$$\Pr[\text{Fail}_2] \leq 1/\sqrt{n}$$
Repeat λ times to drive down failure prob.

can avoid this using more involved scheme

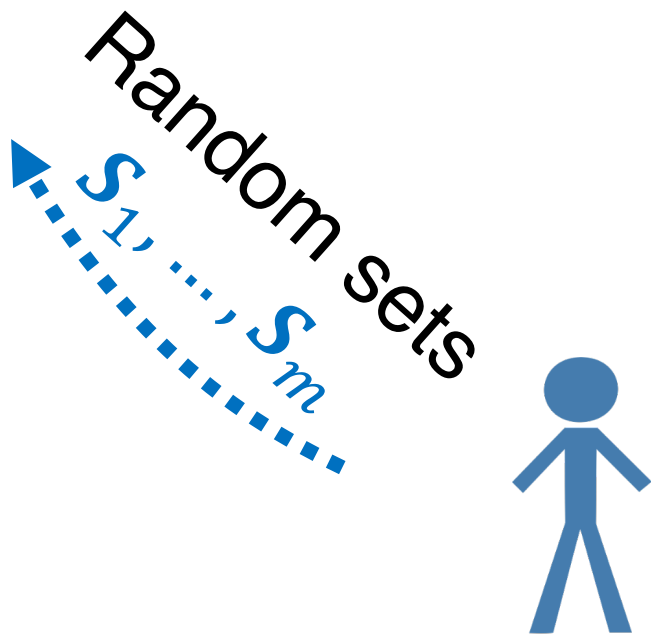


$S_1, h_1, \ldots, S_j, h_j, \ldots, S_m, h_m$

Our scheme
Security – left server

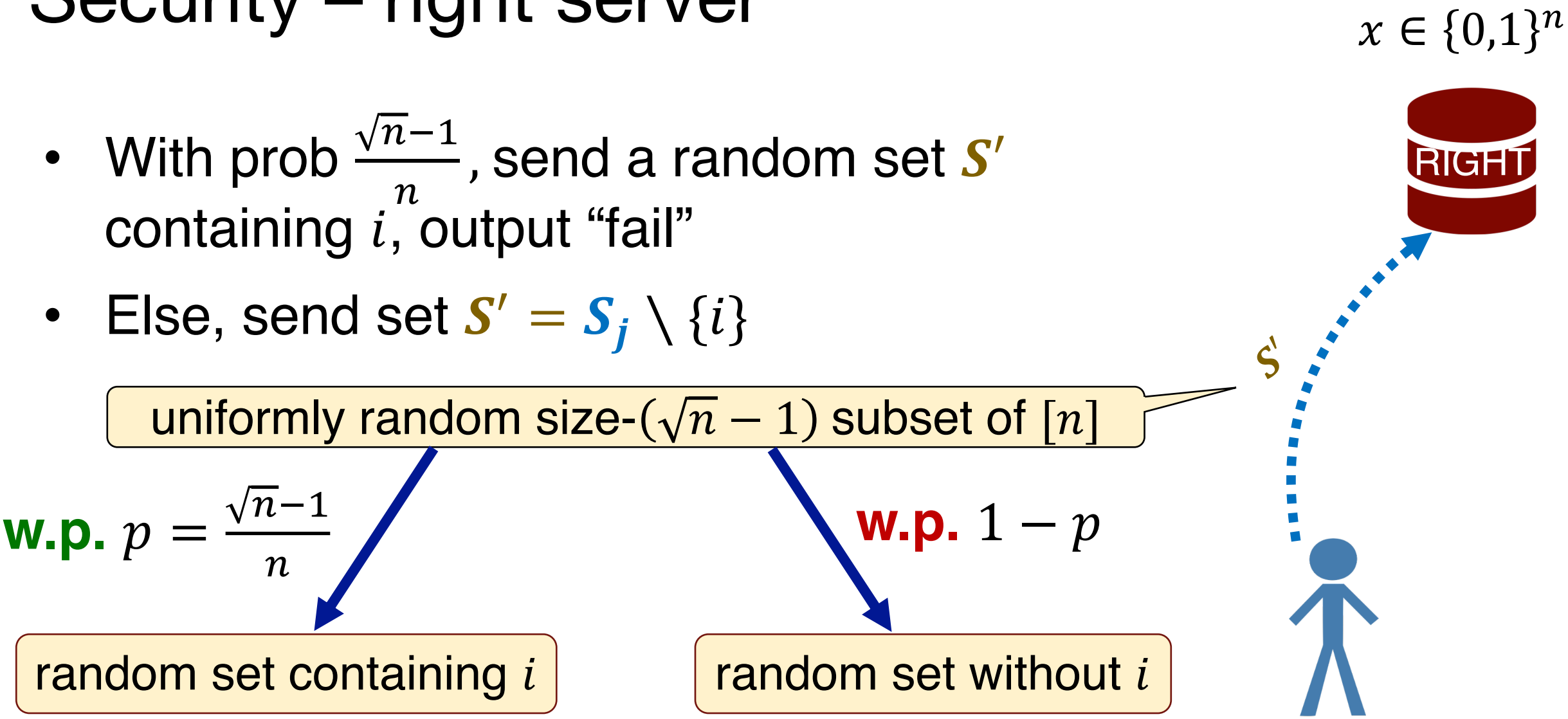
$x \in \{0,1\}^n$

$x \in \{0,1\}^n$

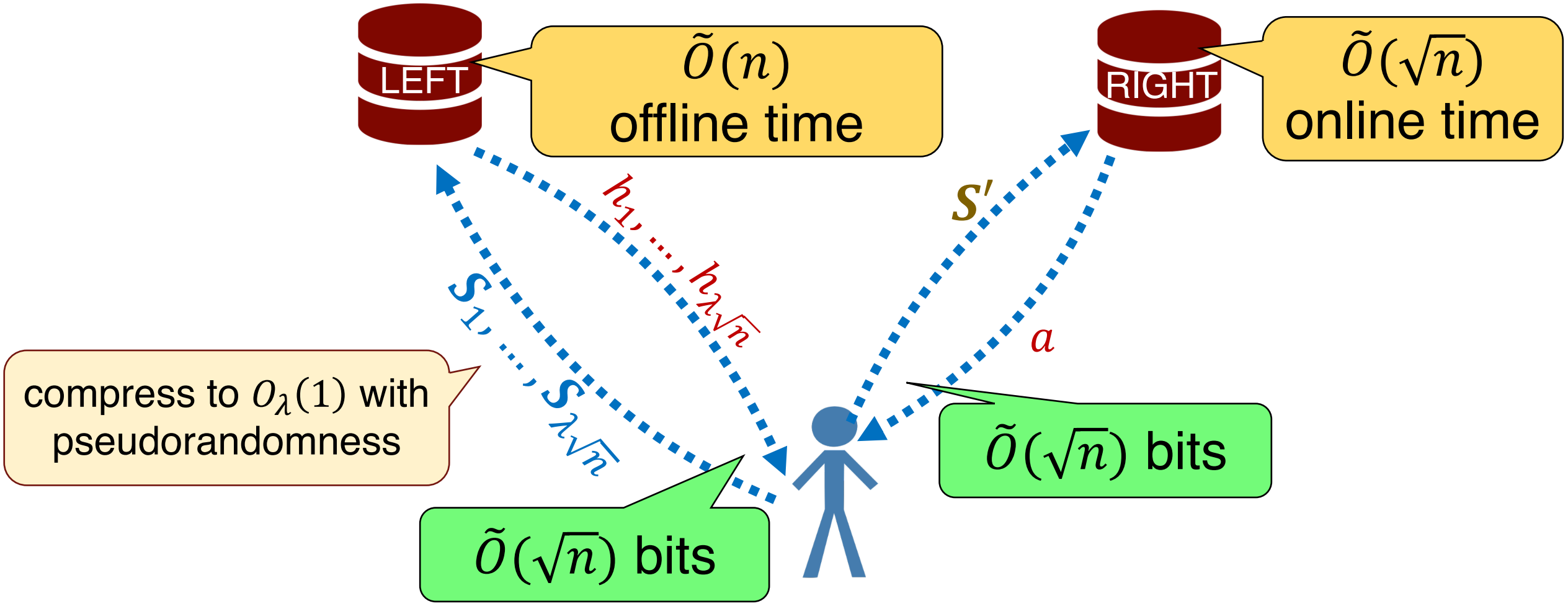


Our scheme
Security – right server

- With prob $\frac{\sqrt{n}-1}{n}$, send a random set S' containing i, output "fail"
- Else, send set $S' = S_j \setminus \{i\}$



Our scheme
Efficiency



Two-server scheme summary

- $\sqrt{n}$ communication, online time, amortized total time per-query
- Client uses $\sqrt{n}$ time and storage (using PRPs)

Extensions (see paper)

- Trade-off communication for online time
- Statistical-security variant: $n^{2/3}$ communication and client time
- Reducing online communication to $\log n$
  - Using short description of 'Puncturable sets'
  - Client storage and time increase to $n^{5/6}$

[SACM20] show how to avoid this blowup using a new construction from privately puncturable PRFs (from LWE)

Offline/online PIR

Two-server scheme

- $\sqrt{n}$ communication and online time (from any PRG)
- Can reuse a single offline interaction for many online queries

Single-server scheme

- $n^{3/4}$ communication and online time (from DDH, DCR,…)
  - $\sqrt{n}$ from FHE
- No public-key operations in the online phase

Lower bound

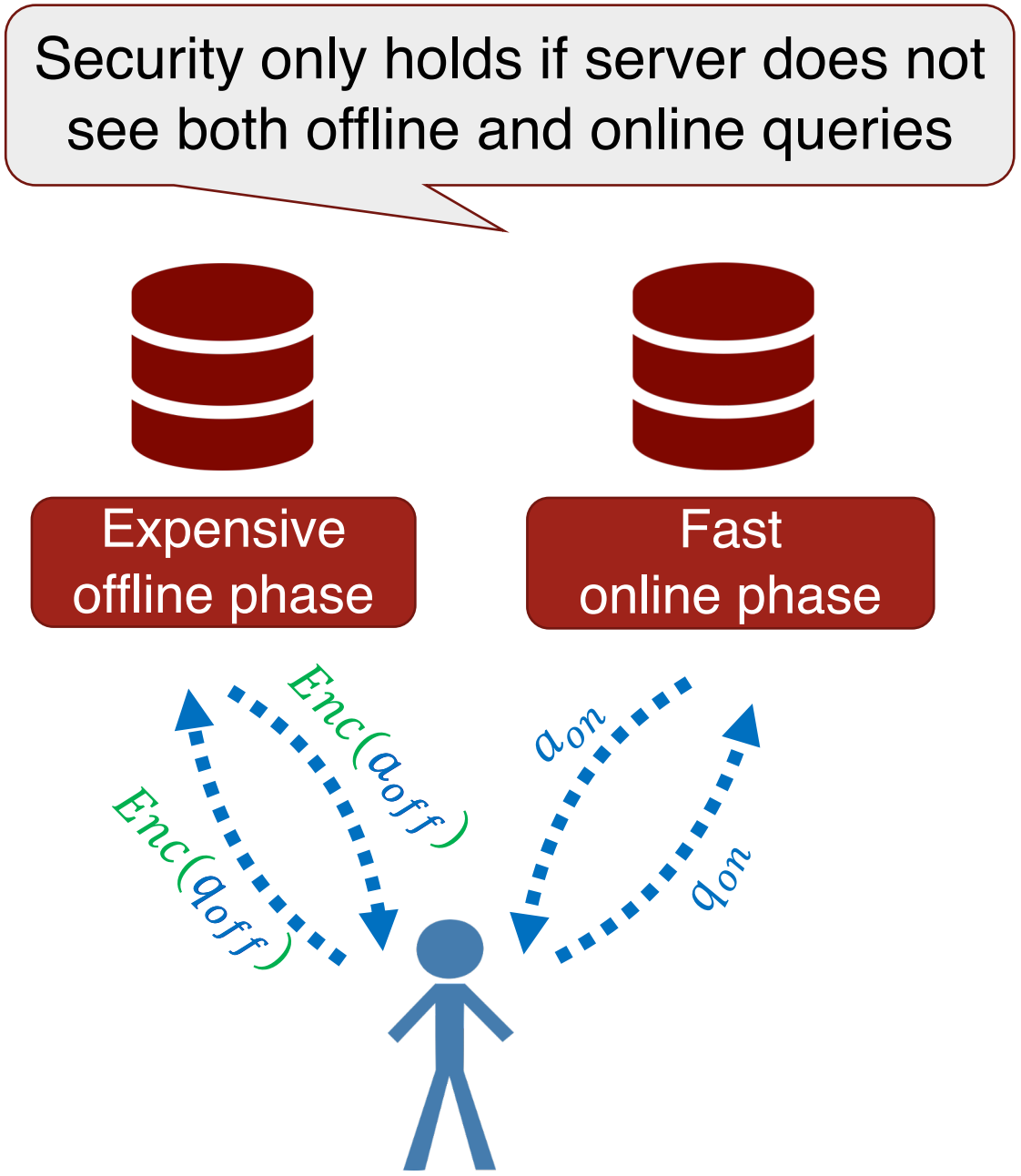
- For offline/online schemes that store DB in its original form
- Communication C and online time T must be $C \cdot T \geq n$

From 2 servers to 1

Run both offline and online phases with the same server

Single server homomorphically evaluates offline query

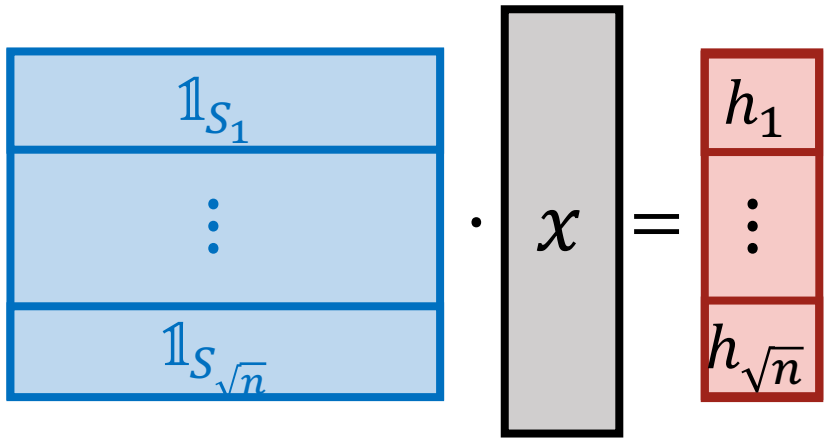
- Option 1: Fully HE
  - $\sqrt{n}$ communication and online time
- Option 2: Additively HE
  - $n^{3/4}$ communication and online time



From 2 servers to 1 – linearize the offline phase

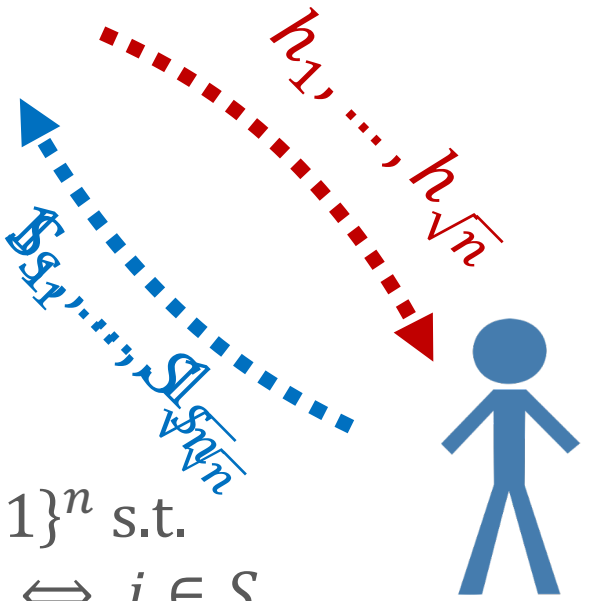
Make server's offline response **linear** in its query:

- Client sends characteristic vectors of sets
- Server computes matrix-vector product



$x \in \{0,1\}^n$

$h_j = \sum_{i \in S_j} x_i \mod 2$



$\mathbb{1}_S \in \{0,1\}^n$ s.t.
$(\mathbb{1}_S)_i = 1 \Leftrightarrow i \in S$

Recall: in offline phase of 2-server scheme:

- Server sends $\sqrt{n}$ sets of size $\sqrt{n}$
- Server responds with parity of each set

From 2 servers to 1 – encrypt the offline phase

Make server's offline response **linear** in its query

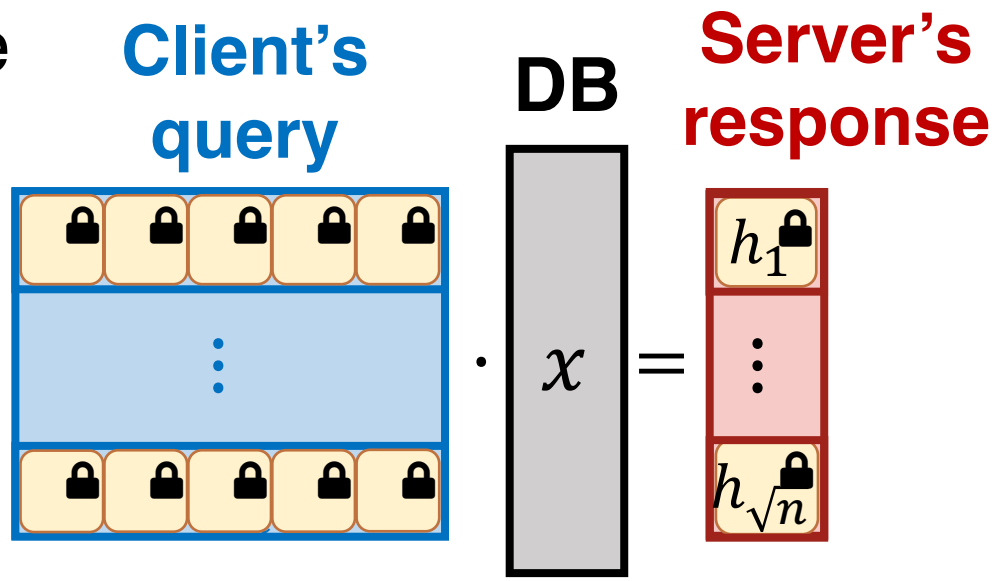
Client encrypts offline query component-wise with **linearly homomorphic encryption**

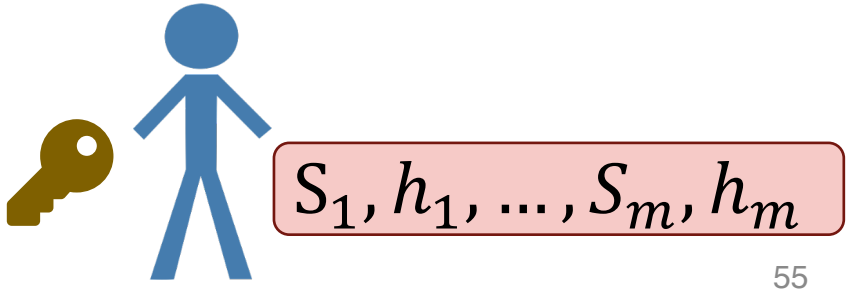
- E.g., Paillier

Server homomorphically computes the hint

Client decrypts hint

Online phase remains unchanged





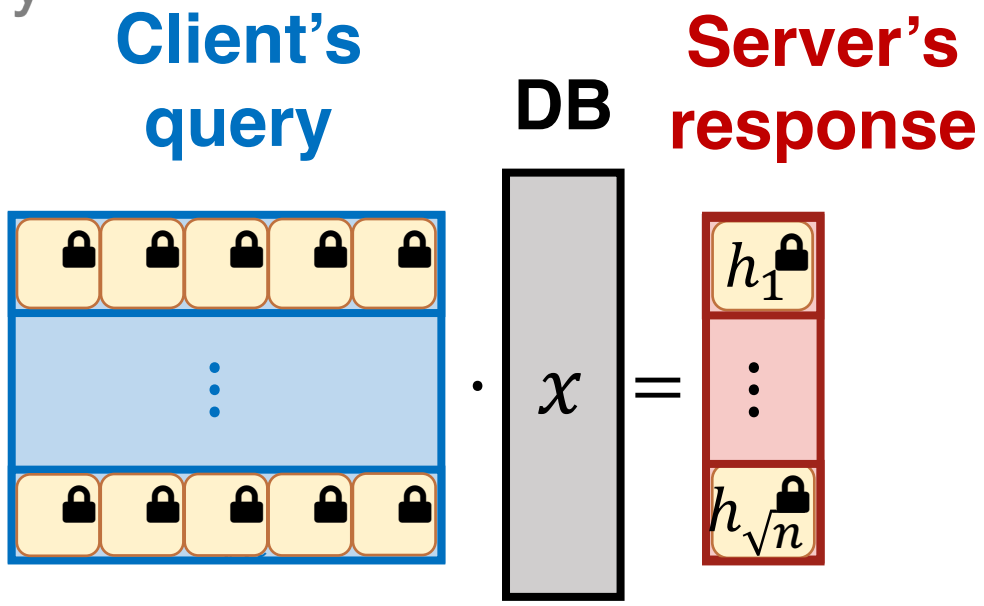
From 2 servers to 1 – efficiency

Make server's offline response **linear** in its query

Client encrypts offline query component-wise with **linearly homomorphic encryption**

Efficiency: as described, the client

- uploads $\sqrt{n} \times n$ matrix → $n^{3/2}$ ciphertexts
- downloads $\sqrt{n}$ ciphertexts



Can use standard PIR rebalancing [CGKS95] to get $n^{3/4}$ **total communication**

Idea: Divide DB into blocks of size $\sqrt{n}$ of instead of single bits

More tricks to get $n^{2/3}$ communication and quasilinear server time

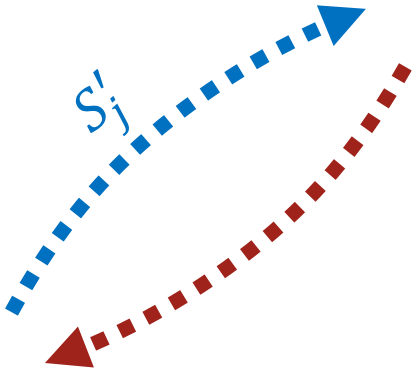
Multiple queries

Goal: amortize cost of offline phase across multiple **adaptive** online queries

$x \in \{0,1\}^n$

Problem: cannot reuse S_j

Given S_j^1 and S_j^2, server knows $i_1 = S_j^2 \setminus S_j^1$



We will see a generic approach (using "Batch PIR" ideas)

Hint $S_1, h_1, \ldots, S_j, h_j, \ldots, S_m, h_m$

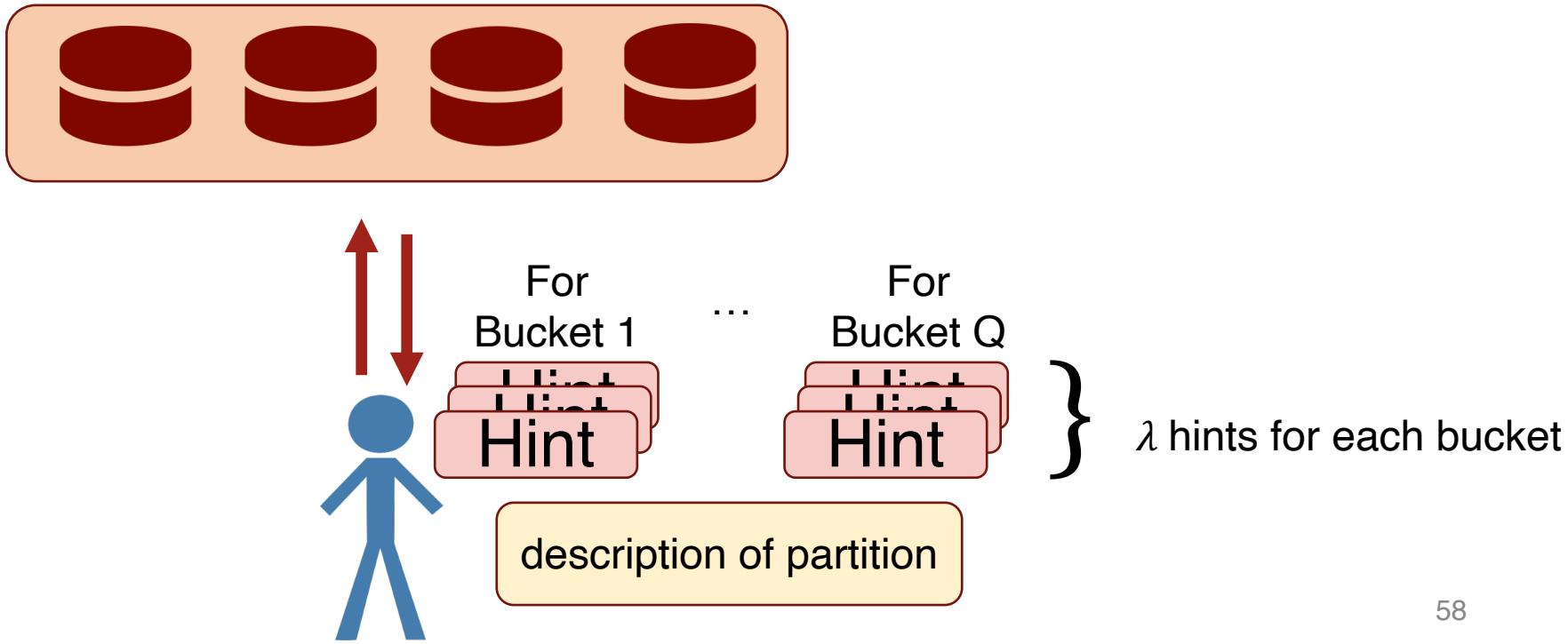
Multiple adaptive queries – offline phase

Partition the DB into Q random buckets

Run the offline phase λ times for each bucket

$x \in \{0,1\}^n$

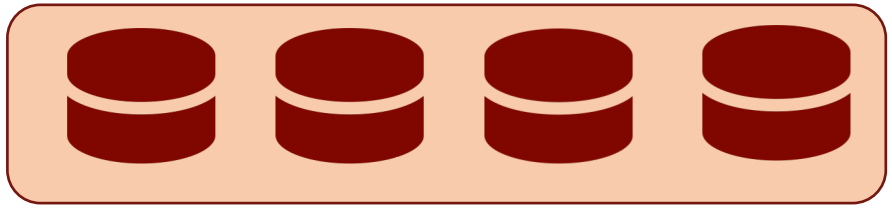
Q buckets

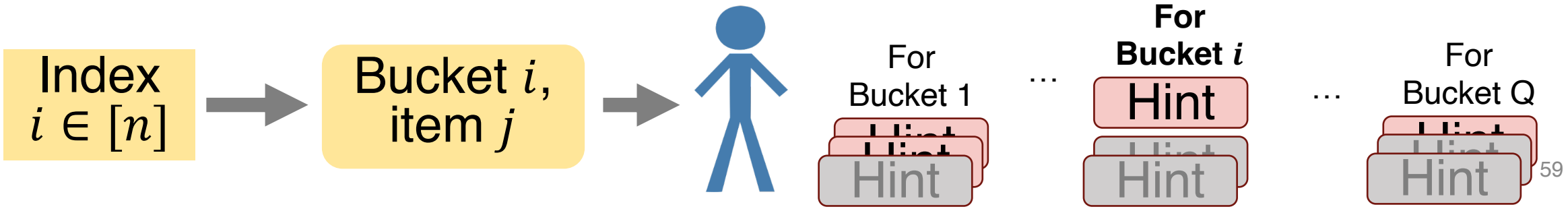


Multiple adaptive queries – online phase

- Find matching bucket and unused hint

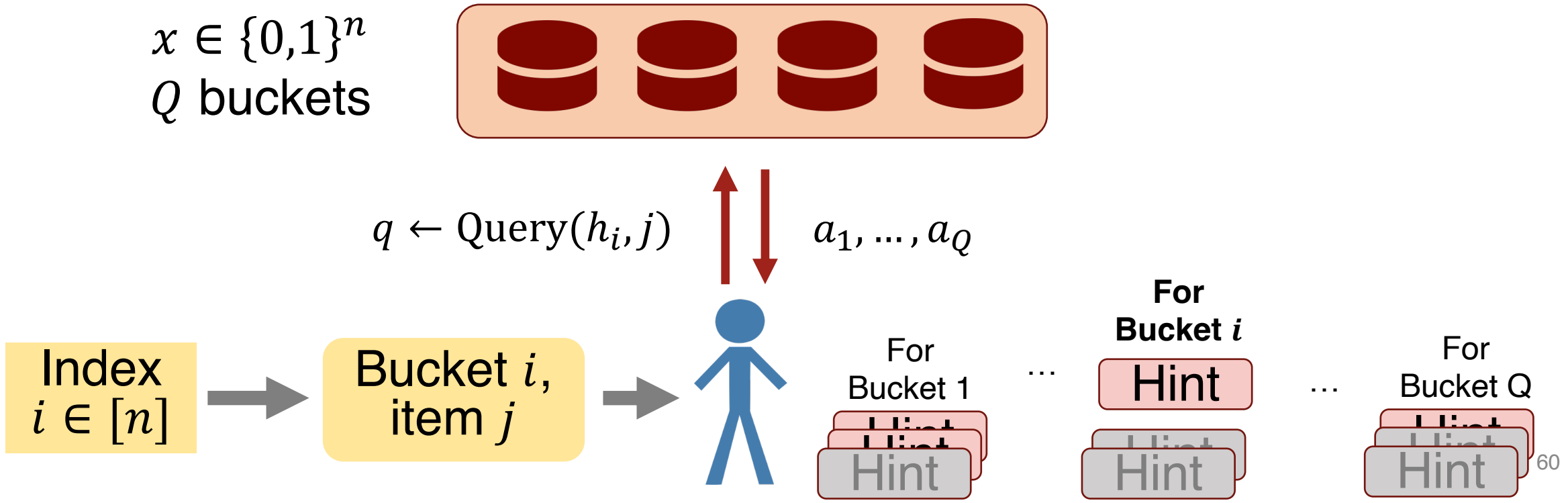
$x \in \{0,1\}^n$
Q buckets





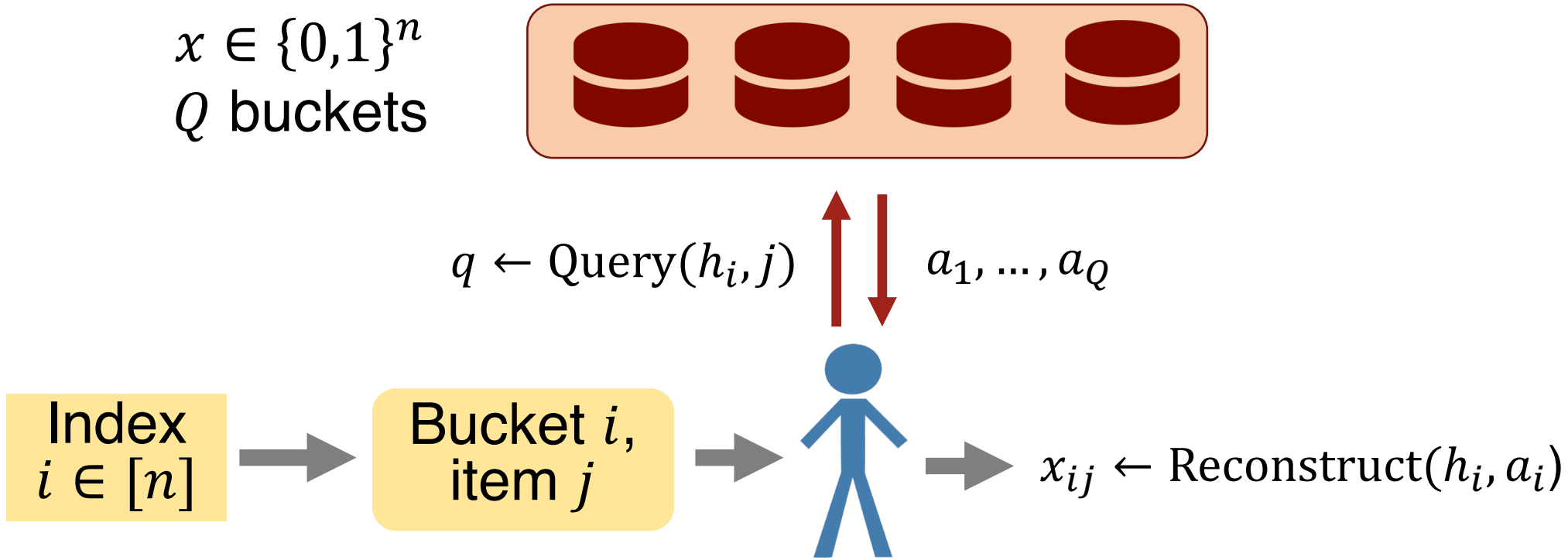
Multiple adaptive queries – online phase

- Find matching bucket and unused hint
- Query using unused hint, server answers w.r.t. each bucket



Multiple adaptive queries – online phase

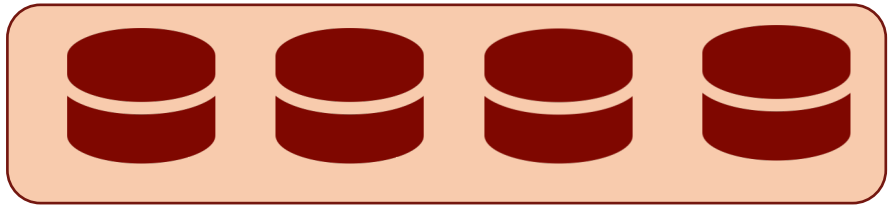
- Find matching bucket
- Query using unused hint, server answers w.r.t. each bucket
- Reconstruct using matching response

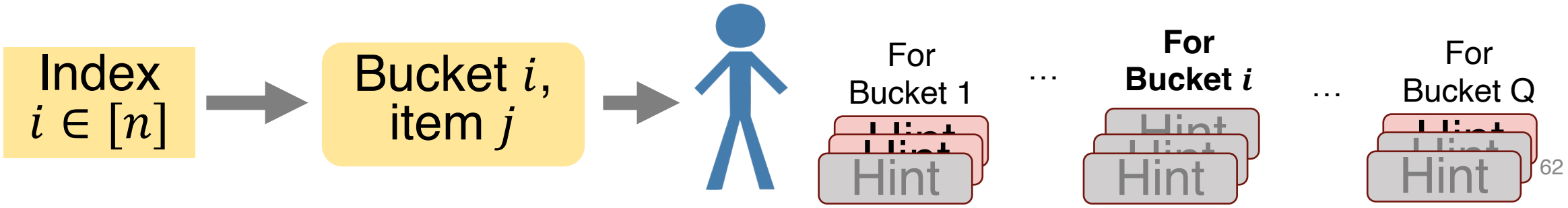


Multiple adaptive queries – online phase

- Find matching bucket
- Query using unused hint, server answers with respect to each bucket
- Reconstruct using matching response
- Mark the hint as used

$x \in \{0,1\}^n$
Q buckets





Multiple adaptive queries – analysis

Correctness:

The probability that a bucket overflows is negligible.

If no overflows, correctness follows from base PIR scheme.

Security:

Server sees a single base query each time,

security follows from base PIR scheme.

Multiple adaptive queries – analysis

Efficiency:

Online server time is $T'(n) = Q \cdot T\left(\frac{n}{Q}\right)$

2-server PIR: our $\mathrm{T} = \sqrt{n}$ scheme and $Q = n^{1/3} \Rightarrow T' = n^{2/3}$

- Different (non-generic) approach gives: $T' = \sqrt{n}$ without additional assumption (only PRGs)

1-server PIR: our $\mathrm{T} = n^{2/3}$ scheme and $Q = n^{1/4} \Rightarrow T' = n^{3/4}$

- Different (non-generic) approach gives: $T' = \sqrt{n}$ from FHE

Offline-online PIR with DB updates

When a database is repeatedly changing:

- standard PIR “just works”
- offline-online PIR requires the client to get an up-to-date hint

Naïve approach: rerun the offline phase after each change

- Linear amount of server work on each change

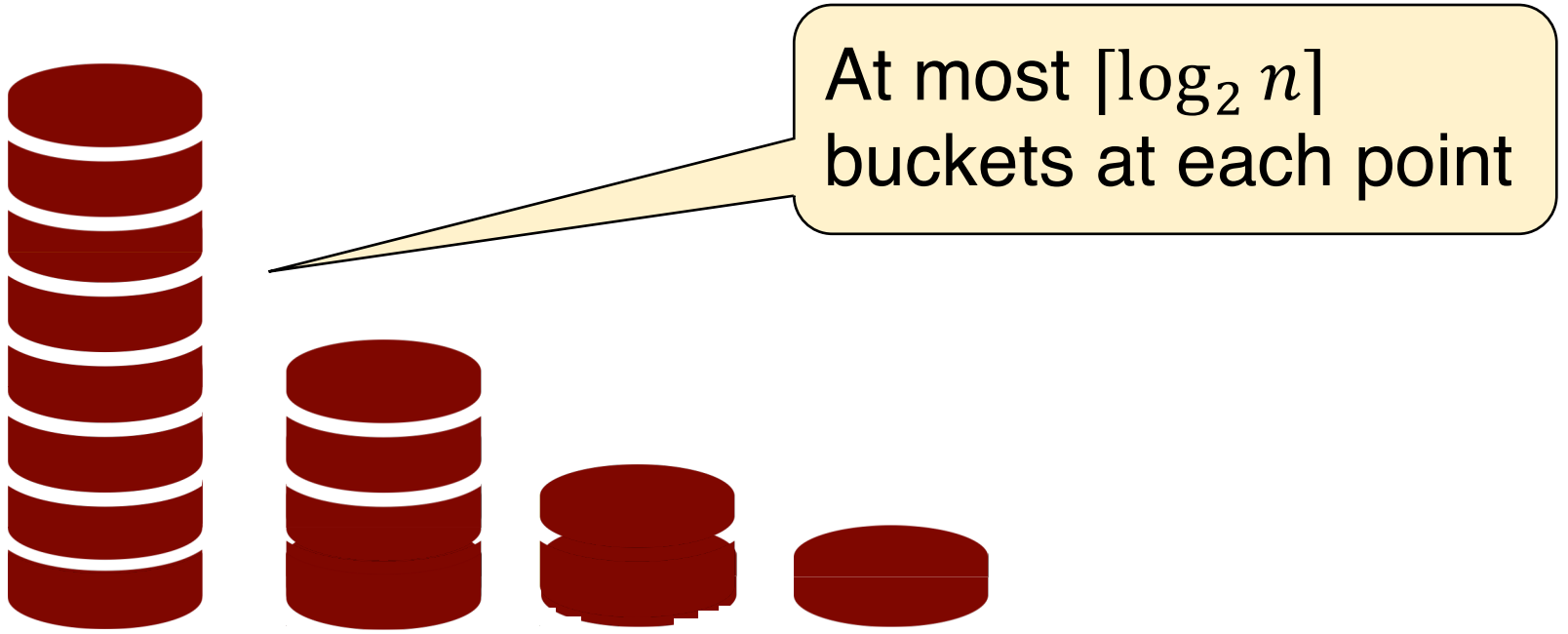
Refined approach: incremental preprocessing

- Logarithmic (amortized) amount of server work on each change

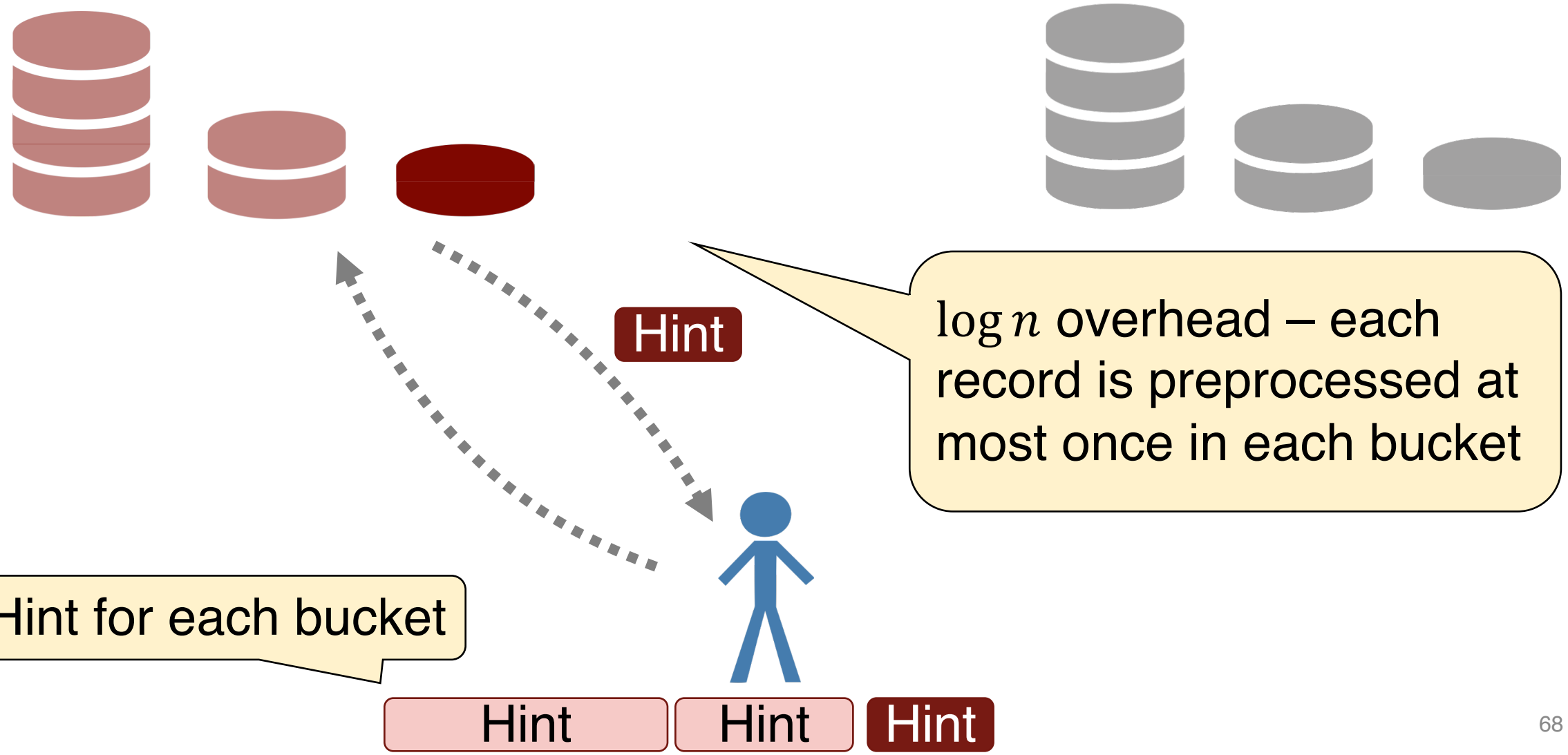
[BS80,GO96,CKO90,SSP13,SPS14,PT16]

Incremental preprocessing

- Database divided into **buckets**
- New record gets a new bucket
- Merge buckets with equal size

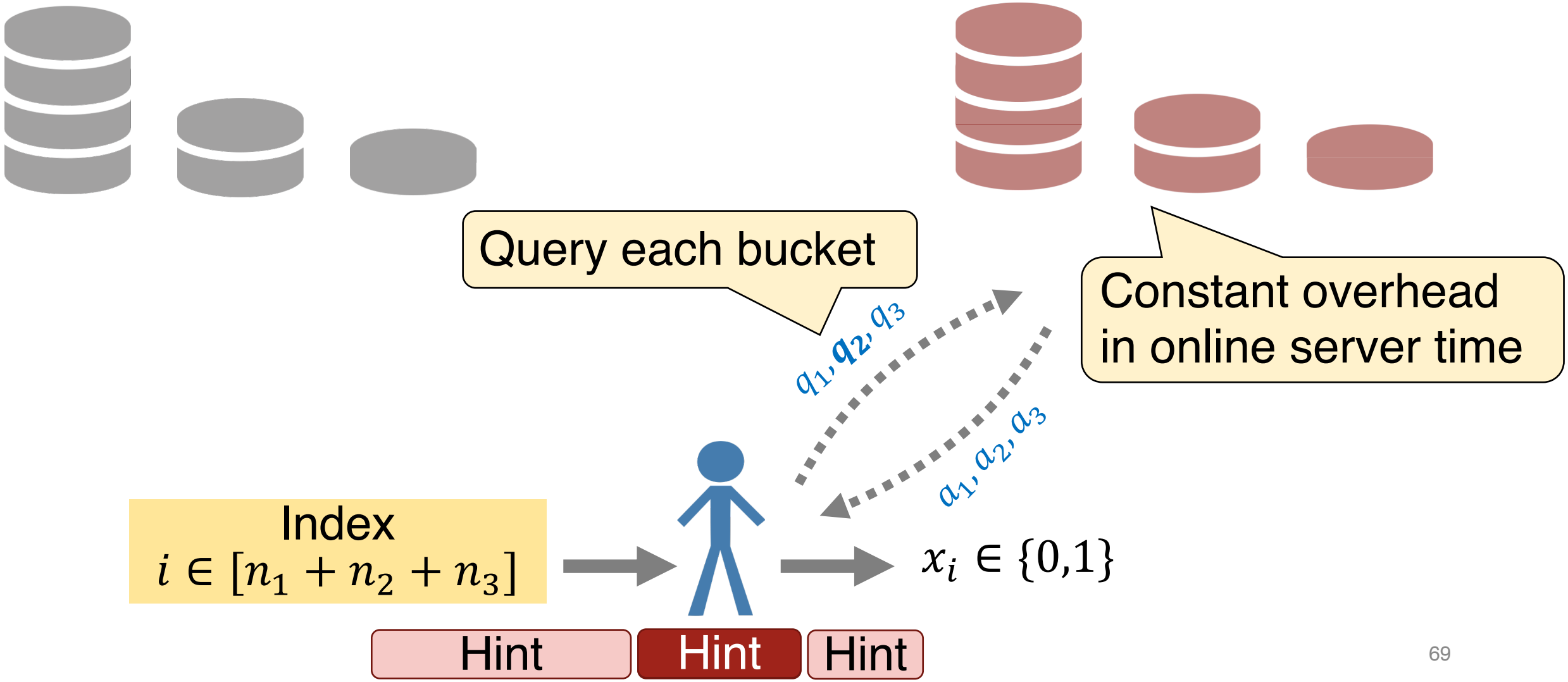


Incremental preprocessing
Offline phase



Incremental preprocessing
Online phase

[MZRA22] More efficient incremental offline/online PIR

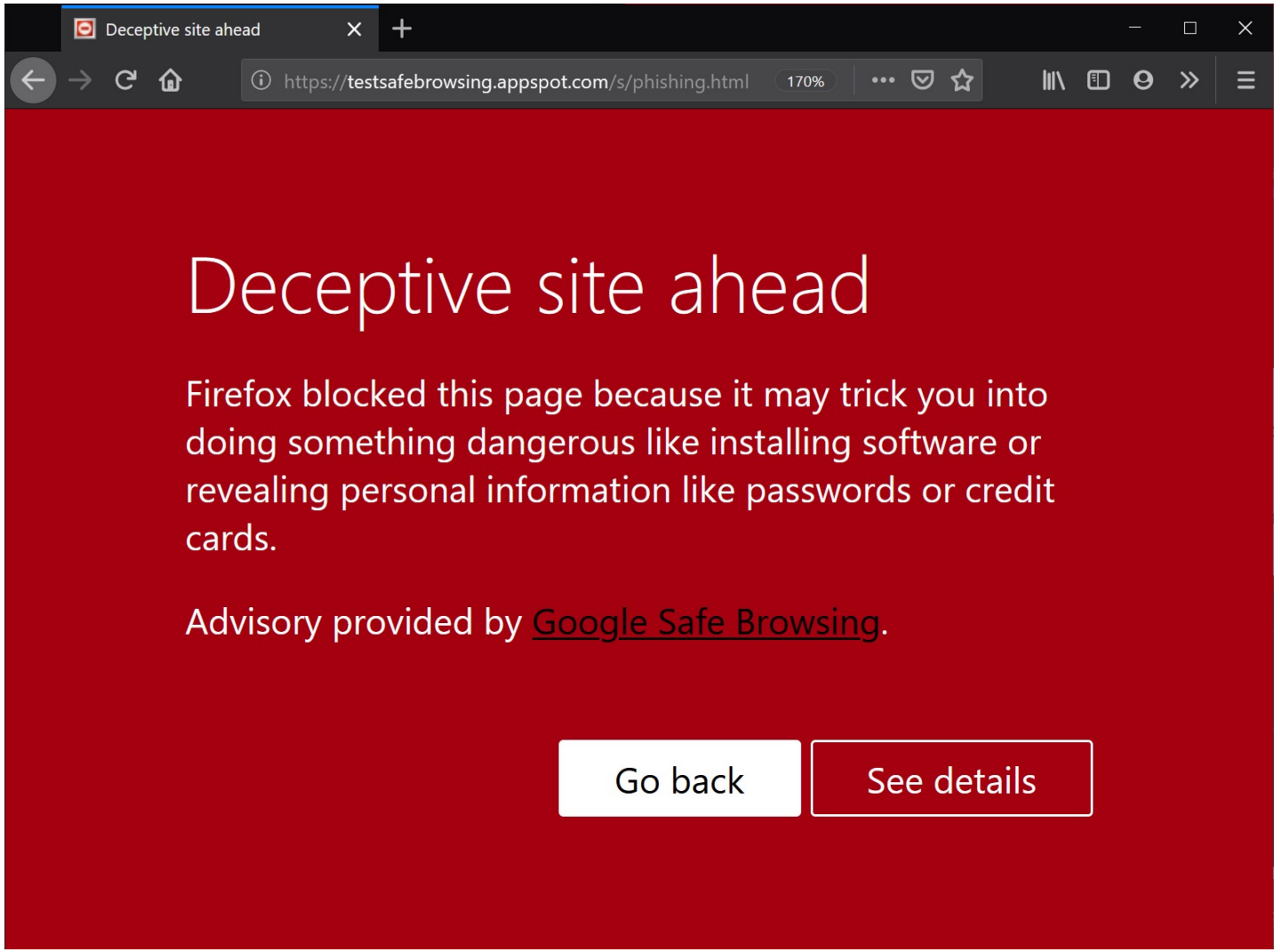


Putting Offline-Online PIR to the test

Built a system for private Safe Browsing queries

Integrated and evaluated with Firefox browser

Results: compare offline-online & DPF-based PIR


Deceptive site ahead
https://testsafebrowsing.appspot.com/s/phishing.html
170%
Deceptive site ahead
Firefox blocked this page because it may trick you into doing something dangerous like installing software or revealing personal information like passwords or credit cards.
Advisory provided by Google Safe Browsing.
Go back
See details

Safe Browsing Today

Leaks information about user's browsing history [GKL16,Per19,BK20]

Firefox browser

evil.com

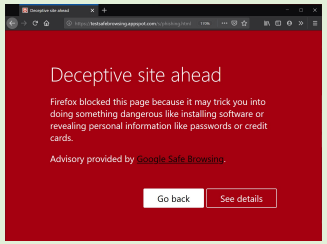
Partial hashes

0x104
0x130
0x1F3
0x1FF
0x24C
0x2B2
...

Lookup
0x24C

Full hash
0x24C1A8…

Warn



Google Safe Browsing

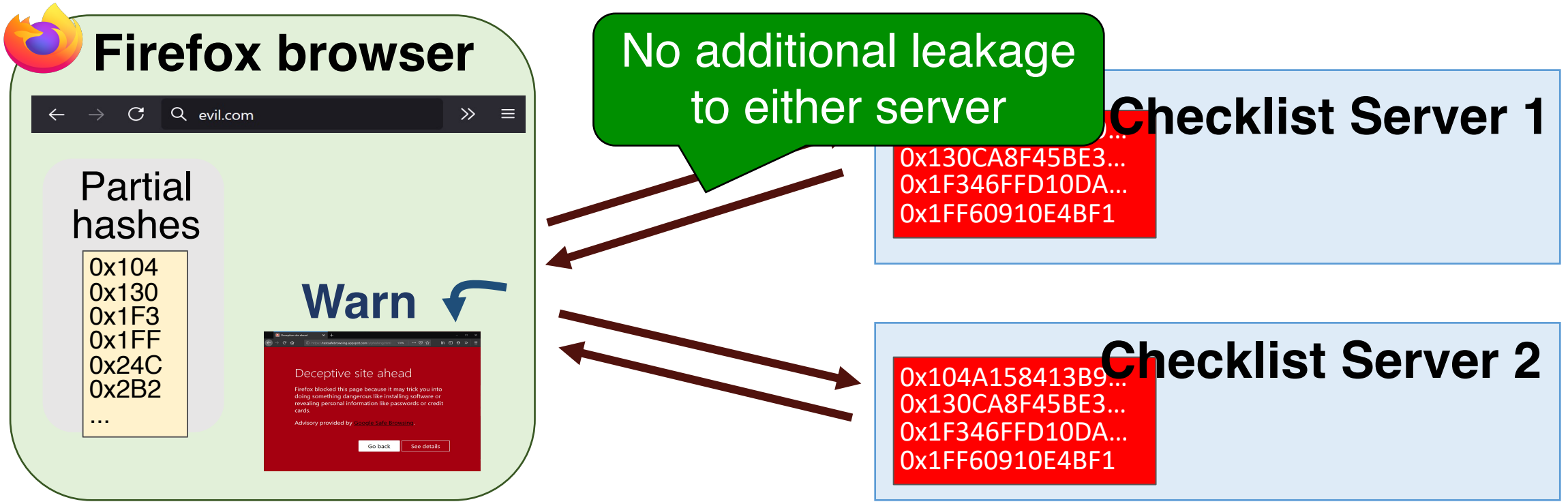
Hashes of ~3M dangerous URLs

0x104A158413B9…
0x130CA8F45BE3…
0x1F346FFD10DA…
0x1FF60910E4BF1…
0x24CC8335003A…
0x2B26ED8D140B…
…

Checklist: A system for private blocklist lookups

(applied to Safe Browsing)

- Roughly 2500 lines of Go + 500 lines of C
- Supports new offline/online and standard (DPF-based) PIR



Estimating the Safe Browsing parameters

Monitor a week of Safe Browsing requests and responses

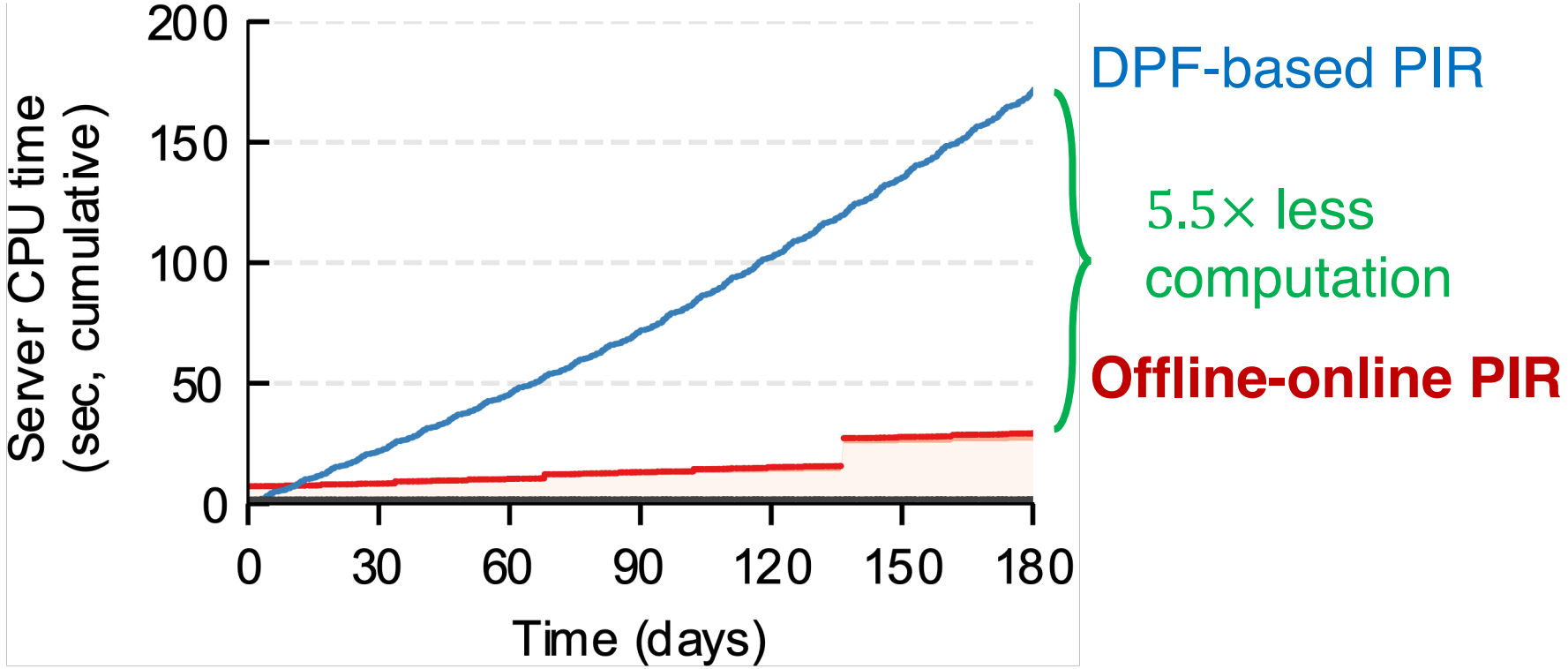
Local proxy forwards requests to the real Safe Browsing API

Deduce:

- Frequency of lookups
- Frequency of updates
- Database growth

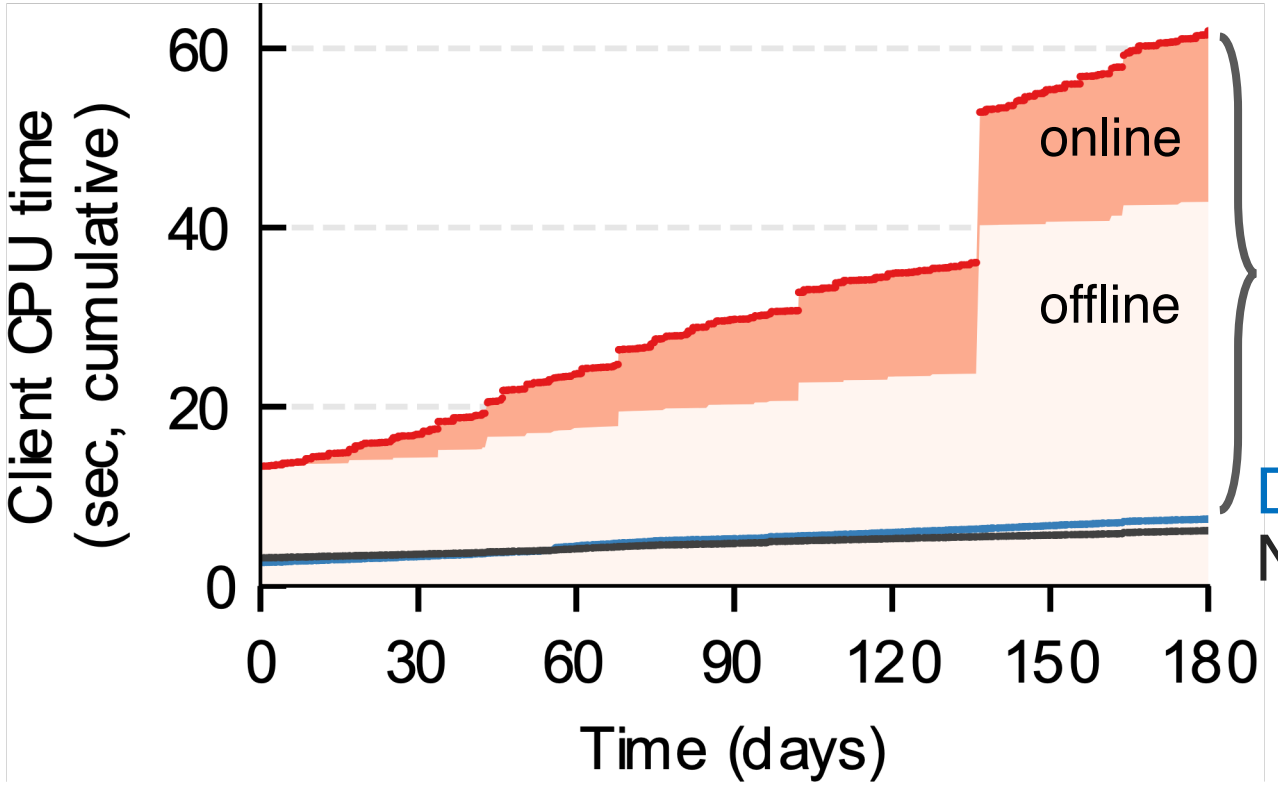
Offline/online PIR is more efficient for the server than traditional PIR

Replay recorded trace of a single user



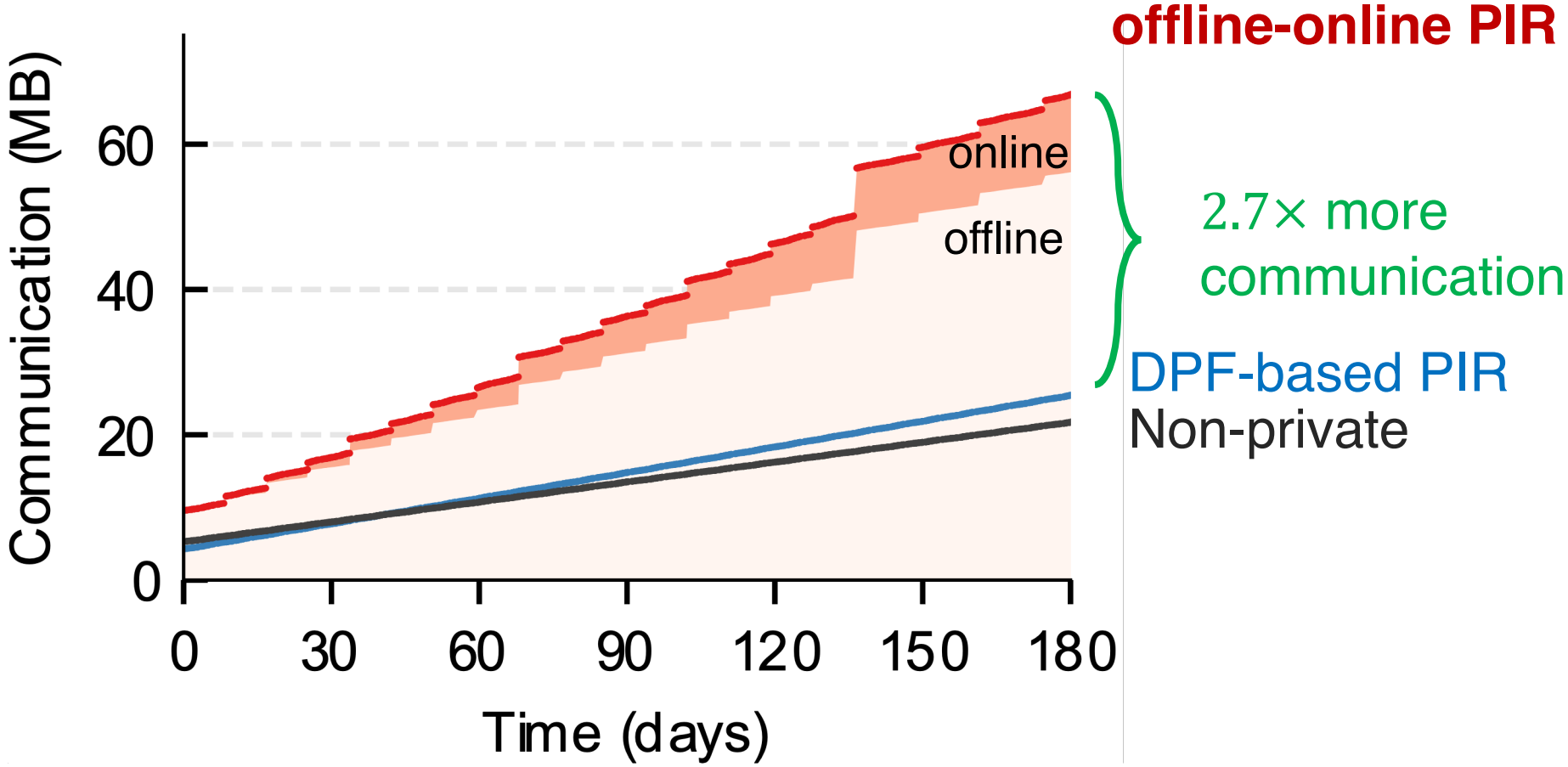
Server running on e2-standard-4 Google Compute Engine machine (4 vCPUs, 16 GB RAM)

Traditional PIR is more efficient for the client than offline/online PIR



Client running on a Pixel 5 mobile phone

Traditional PIR requires less communication than offline/online PIR



Conclusion

- PIR has an array of applications to privacy-preserving systems
- Costs of PIR are now within the realm of practicality
  - Several alternatives that allow for different trade-offs

Next steps: Make it work in the real world

dkogan@cs.stanford.edu

https://cs.stanford.edu/~dkogan/