PRIVATE INFORMATION RETRIEVAL

TODAY:
- Motivation
- Two-server PIR scheme
- Computational single server PIR scheme

Last two lectures : "generic" MPC

← works for any computation (described as a circuit)

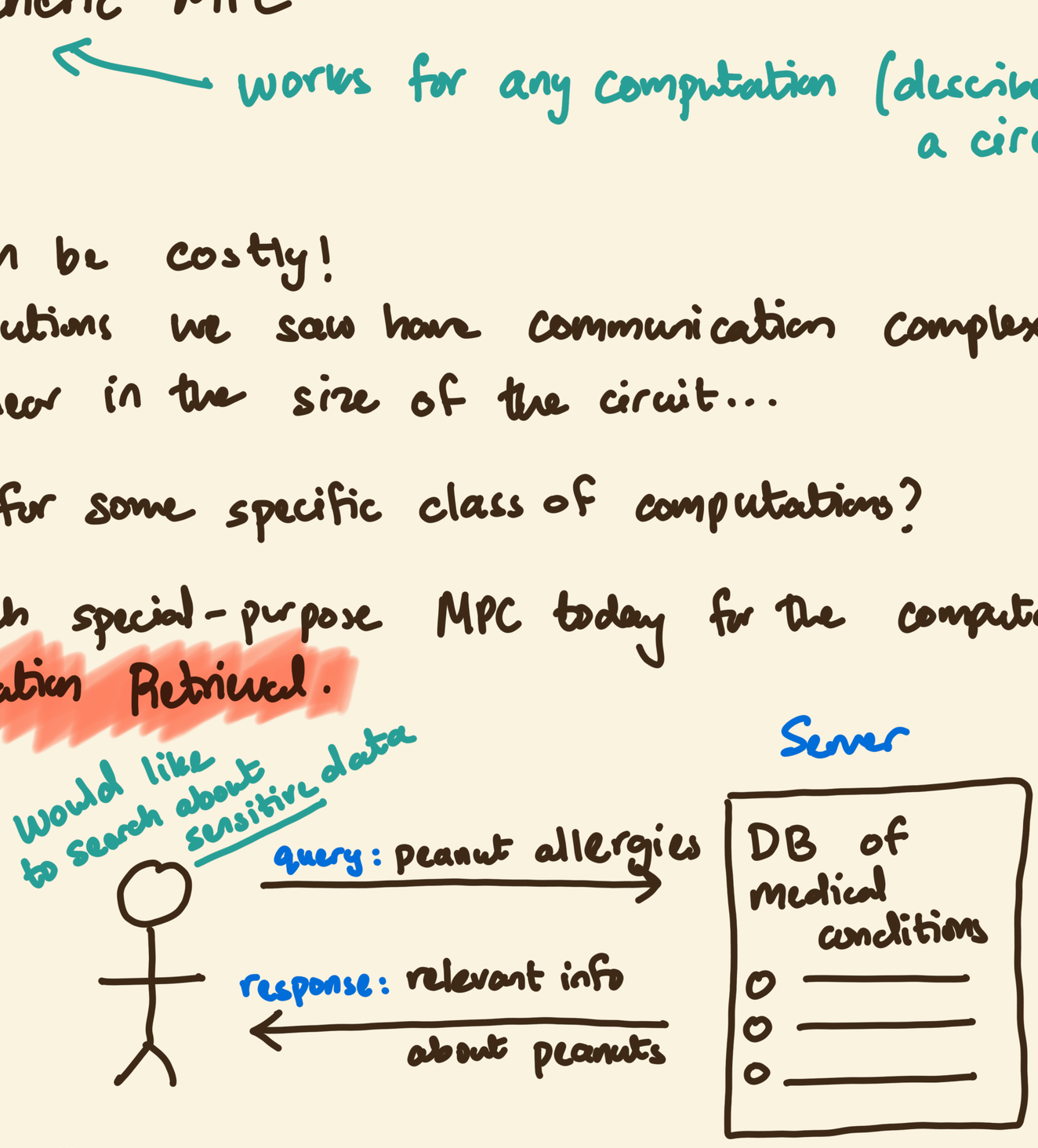
Can be costly!
Solutions we saw have communication complexity linear in the size of the circuit...

Can we do better? Say, for some specific class of computations?

Yes! And we'll see one such special-purpose MPC today for the computation known as **Private Information Retrieval.**

Motivation

E.g. Private browsing (simplified)



Server learns your query... maybe you wish to keep your sensitive data private so info is not sold to ad platforms or insurance companies

We can perform the DB lookup privately using MPC techniques we've already seen.
But circuit for DB lookup is linear in size of DB ← can be really expensive :(

MPC for $f_{lookup}(DB, i) = (DB[i], \perp)$

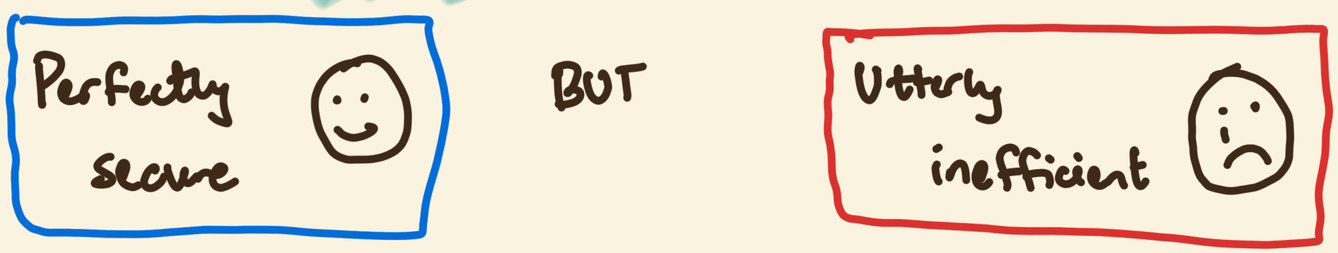
Private Information Retrieval (PIR) is a specialized MPC which will allow us to query the DB without the server learning <u>any info</u> about the query!

← wait a min... is this not just OT?
No! OT is in fact stronger: it required privacy from the user too. In PIR, we don't need to hide DB[j] for $j \neq i$ from the user. It is one-sided privacy!

First (Trivial) Construction

Since, we don't care about DB's privacy for the user...

Just send the entire DB!



We want to reduce communication between server and client.
Asymptotically, we want sublinear in the size of DB!

Is this ever possible with perfect security?

No. if $|comm.| < |DB|$, we must lose some info. about some entry j. $j \neq i$ since $DB[i]$ is what the user wishes to learn. Unbounded server can identify j and now learns $i \neq j$.

We can get around this issue in two ways

we'll cover both today!

Multiple (Non-colluding) Servers [CGKS95]

- Replace single server with 2 or more non-colluding servers
  - ↖ not allowed to communicate
- can still achieve perfect security

Computational Security [K097]

- Security against computationally bounded server
- Relies on cryptographic primitives

Two Server PIR

Two servers: S_0 & S_1

DB: bit-vector of length N : $DB \in \{0,1\}^N$

Goal: For input $i \in [N]$, user learns $DB[i]$ without revealing i to S_0 or S_1.

Def. A **two server PIR scheme** is a tuple of eff. algs.:

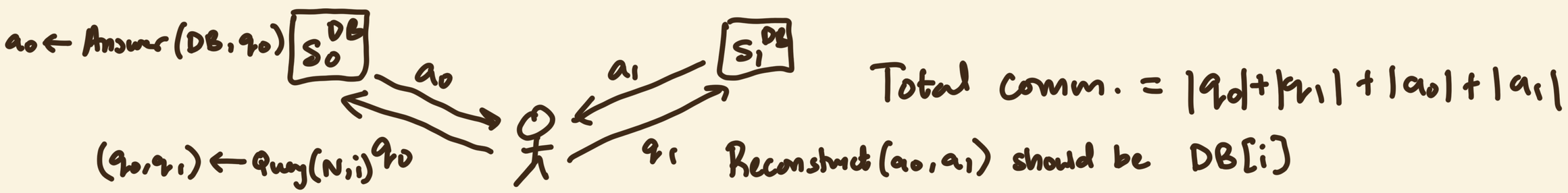
- $\text{Query}(N, i) \to (q_0, q_1)$: $N = |DB|$, i is the query index

query strings: q_0 sent to S_0; q_1 sent to S_1

- $\text{Answer}(DB, q) \to a$: $DB \in \{0,1\}^N$, q is a query string from the client/user, a is the server response
- $\text{Reconstruct}(a_0, a_1) \to b$: a_i are server responses, b is reconstructed entry

S_0 : a_0 → client

S_1 : a_1 → client



Correctness. $\forall N \in \mathbb{N}, i \in [N], DB \in \{0,1\}^N$,

$$\Pr\left[b = DB[i] \;:\; \begin{array}{l} (q_0, q_1) \leftarrow \mathsf{Query}(N, i) \\ a_0 \leftarrow \mathsf{Answer}(DB, q_0) \\ a_1 \leftarrow \mathsf{Answer}(DB, q_1) \\ b \leftarrow \mathsf{Reconstruct}(a_0, a_1) \end{array} \right] = 1$$

Security. $\forall$ poly-bounded $N \in \mathbb{N}$, $DB \in \{0,1\}^N$, pairs $(i,j) \in [N]^2$, $b \in \{0,1\}$,

$$\{ q_b : (q_0, q_1) \leftarrow \mathsf{Query}(N, i) \} \approx \{ q_b : (q_0, q_1) \leftarrow \mathsf{Query}(N, j) \}$$

↑ perfect/statistical/computational

Note. we're looking at query strings sent to each server individually.

A(nother) Trivial Construction

Query(N, i):

- Sample $q_0 \xleftarrow{\$} \{0,1\}^N$ and set $q_1 := q_0 \oplus e_i$ ($e_i := (0, \ldots, 0, \underset{\text{ith position}}{1}, 0, \ldots, 0)^T$)
- Output (q_0, q_1)

Answer(DB, q):

- Output $a \leftarrow \langle DB, q \rangle$ ← recall inner product (mod 2): $\sum_{i=1}^{N} DB[i] \cdot q[i] \pmod 2$

Reconstruct(a_0, a_1):

- Output $a_0 \oplus a_1$

Correctness. $a_0 \oplus a_1 = \langle DB, q_0 \rangle \oplus \langle DB, q_1 \rangle$

$= \langle DB, q_0 \oplus q_1 \rangle$

$= \langle DB, e_i \rangle$

$= DB[i]$

Security. Marginally, q_0 and q_1 are uniformly random N-bit vectors

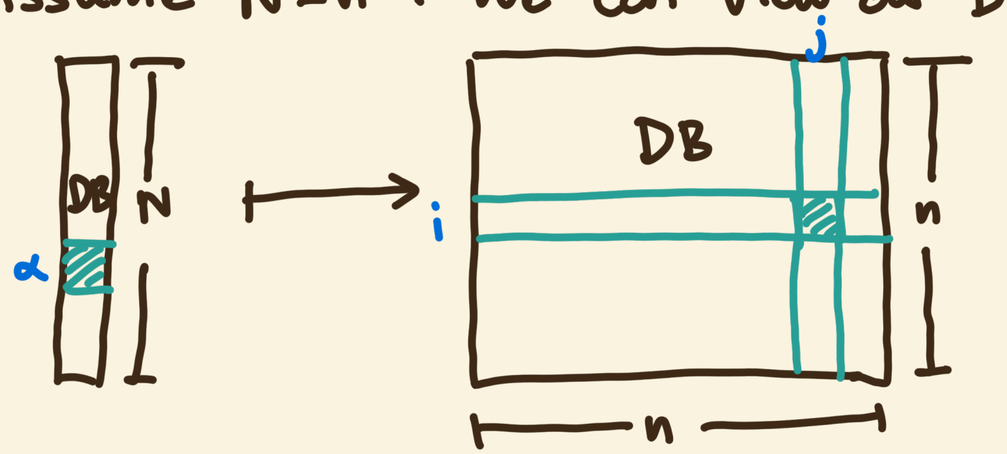
server S_i only sees q_i, and so, the distributions are identical!

But alas, communication is still linear :(← traded download (server ↦ client)

for upload (client ↦ server)

[CGKS95] Construction

Assume $N = n^2$. We can view our DB as a matrix.



$i = \lfloor \alpha / n \rfloor$

$j = \alpha \bmod n$ $(\alpha = n \cdot i + j)$

We will construct

$q_0 \oplus q_1 = e_j =$

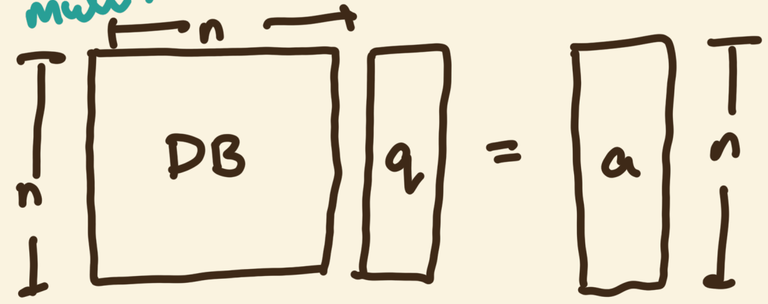
← j^{th} position

Query $(N, (i,j))$: ← interpreted as a pair

- Sample $q_0 \xleftarrow{\$} \{0,1\}^n$ ← vs. N from before, and set $q_1 := q_0 \oplus e_j$
- Output (q_0, q_1)

Answer (DB, q):

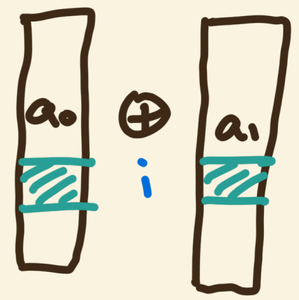
- Compute $DB \cdot q \to a$ ← matrix-vector mult.



- Output a.

Reconstruct (a_0, a_1):

- Output $b \leftarrow (a_0 \oplus a_1)[i]$



Correctness.

$$\begin{aligned} a_0 \oplus a_1 &= DB \cdot q_0 \oplus DB \cdot q_1 \\ &= DB \cdot (q_0 \oplus q_1) \\ &= DB \cdot e_j \\ &= DB_j \leftarrow j^{th} \text{ col of } DB \end{aligned}$$

So, $(a_0 \oplus a_1)[i] = DB_j[i]$
$= DB[i,j]$ ✓

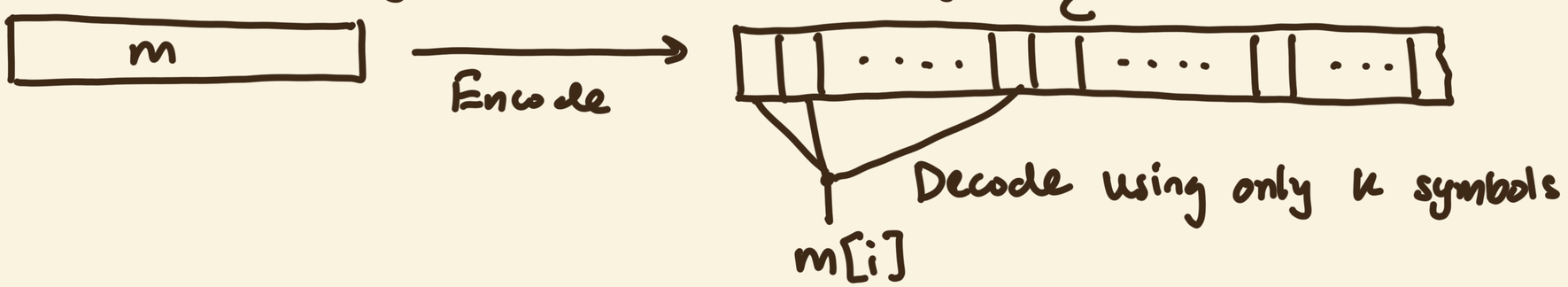
Security. similar to before:
q_0 and q_1 are unif. random (marginally)

Communication Complexity. $O(n) = O(\sqrt{N})$ bits! (upload = download)

(Note that client learns all values in the column but this is fine for PIR!)

Aside. Connection to Locally Decodable Codes (LDCs).

Perfectly secure (or information-theoretic) PIR is closely related to LDCs in coding theory. LDCs are a special type of error-correcting code:



Decoding any bit of the message does not require reading the whole codeword

Using LDCs, we can do even better: $O(N^{\sqrt{\log\log N/\log N}}) = O(N^{o(1)})$ comm. due to [DGIS]

Computational Single Server PIR

Using two servers, we could hide the query from both servers?
Other ways to "hide" data? Encryption! What if we encrypt the query vector?

We wish to compute $DB \cdot q$ but standard encryption does not support this...

→ Enter Linearly Homomorphic Encryption

Briefly, Linearly Homomorphic Encryption (LHE) is an enc. scheme that also satisfies the following property:

Linear Homomorphism.

$$\forall k \in \mathcal{K},\ \forall m_0, m_1 \in \mathcal{M}\ (\text{abelian group}),\ \forall c_0, c_1 \in \mathbb{Z}_r$$

$$c_0 \cdot \mathrm{Enc}(k, m_0) + c_1 \cdot \mathrm{Enc}(k, m_1) = \mathrm{Enc}(k, c_0 m_0 + c_1 m_1)$$

$+$: group op. of ctxt space; $+$: group op. of msg space

[KO97] use LHE to get Comp. Single Server PIR:

Idea. Do CGKS under LHE

E.g. ElGamal is a public-key LHE

Query$(N, (i,j))$:

- Sample $k \xleftarrow{\$} \mathcal{K}$
- Output $q := \left(\mathrm{Enc}(k, e_j[1]) \overset{c_1}{\cdot\cdot} i,\ \mathrm{Enc}(k, e_j[2]) \overset{c_2}{\cdot\cdot} i,\ \ldots,\ \mathrm{Enc}(k, e_j[n]) \overset{c_n}{\cdot\cdot} i\right)$

q: vector of ciphertexts. Encryption of e_j vector component-wise

Answer(DB, q):

- Using the LHE, we can compute $a := DB \cdot q$ (scalars $\cdot$ ctxts)

$$a = \begin{bmatrix} & DB & \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} DB_1[1]\cdot c_1 + DB_2[1]\cdot c_2 + \cdots + DB_n[1]\cdot c_n \\ \vdots \\ DB_1[n]\cdot c_1 + DB_2[n]\cdot c_2 + \cdots + DB_n[n]\cdot c_n \end{bmatrix}$$

$$= \begin{bmatrix} \mathrm{Enc}(k, \sum_{i=1}^{n} DB_i[1] \cdot e_j[i]) \\ \vdots \\ \mathrm{Enc}(k, \sum_{i=1}^{n} DB_i[n] \cdot e_j[i]) \end{bmatrix}$$

by the linear homomorphism property

$$= \begin{bmatrix} \mathrm{Enc}(k, DB_j[1]) \\ \vdots \\ \mathrm{Enc}(k, DB_j[n]) \end{bmatrix}$$

- Output a

Reconstruct(k, a):

- Decrypt all n ciphertexts in a

$$u \leftarrow (\mathrm{Dec}(k, a[1]), \ldots, \mathrm{Dec}(k, a[n]))$$

- Output $u[i]$. ← which will be equal to $DB_j[i] = DB[i,j]$

Correctness. Follows from what we showed above.

Security (informal). Follows from semantic security of LHE.

Enc. of e_j is indist. from Enc of $e_{j'}$

Communication. $|ctxt| = O(\lambda)$ so overall comm. is $O(\lambda n) = O(\lambda \sqrt{N})$.

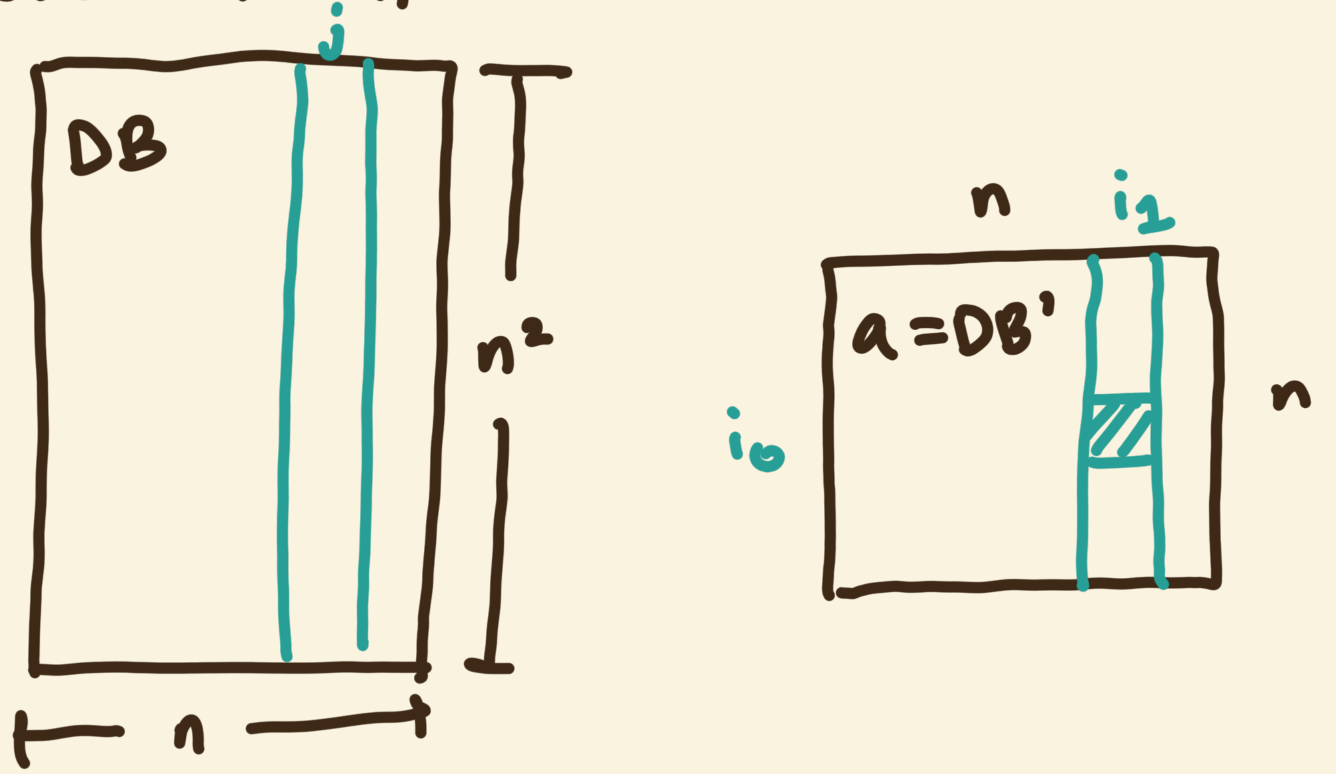
The question we keep asking: **Can we do better??**

Yes! We will use recursion :O.

The client is only interested in $DB[i,j]$ (= $DB_j[i]$) but the server is currently sending the entire column: DB_j.

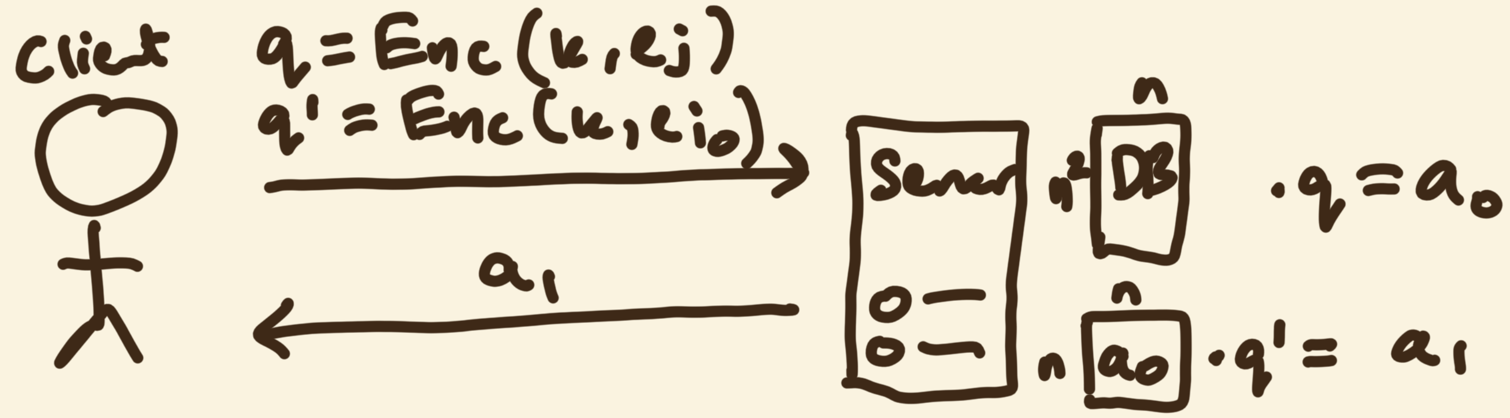
Idea. Think of DB_j as the database we do PIR on!

Suppose $N = n^3$.



$$\alpha = (i_0 \cdot n + i_1) + j \quad i_0, i_1, j \in [n]$$

Client sends two selection vectors:



$$\text{Total comm.} = |q| + |q'| + |a_1|$$
$$= O(\lambda n)$$
$$= O(\lambda N^{1/3})$$

Nice!

Can we keep recursing?

Yes, but there is a tipping point.

Issue. $|\text{entry}| = 1$ bit but $|\text{ctxt}| = \lambda$ bit

Every time we recurse, we save on comm. w.r.t. to DB but blow up comm. by a factor of λ.

Why? Because we recurse on a sub-DB of ciphertexts NOT a DB of 1-bits...

So, ctxt of λ bits $\Rightarrow |a_0| = \lambda n$,

which means $|a_1| = \lambda(\lambda n) = \lambda^2 n$

(λ: ctxt size; λn: sub-DB size)

If you recurse ℓ times, $|a_\ell| = \lambda^\ell n = \lambda^\ell N^{1/\ell}$ ← can blow up very quickly...

Solution. Use LHE with (1) good "rate": $|m|/|c|$ is closer to 1

(2) large message space

E.g. Damgård-Jurik

State of the art. polylogarithmic in $|DB|$. See website for survey.