From Lattices to Tensor Cores: Scaling up Private Information Retrieval

David Wu

August 2026

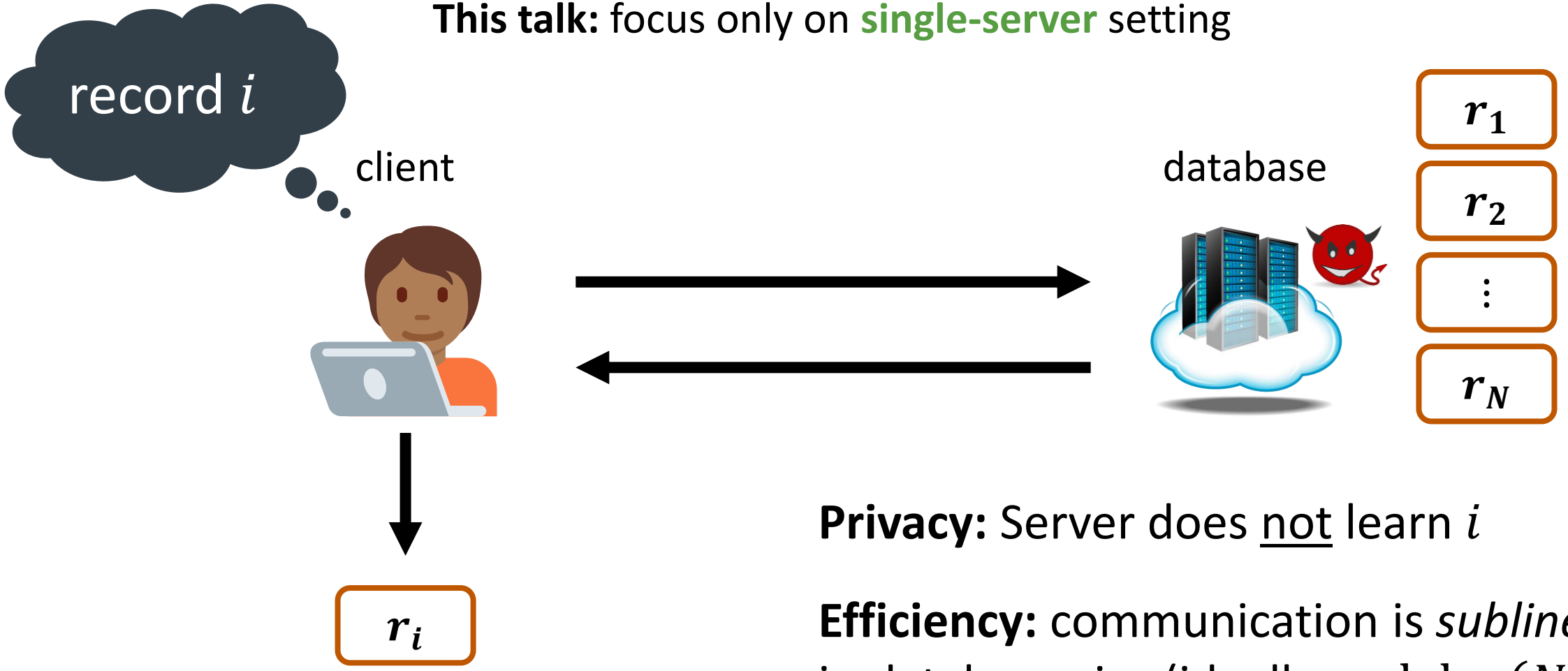
joint work with Sidaarth Sabhnani

Work funded in part by an Amazon Research Award

Private Information Retrieval (PIR)

[CGKS95]

This talk: focus only on **single-server** setting



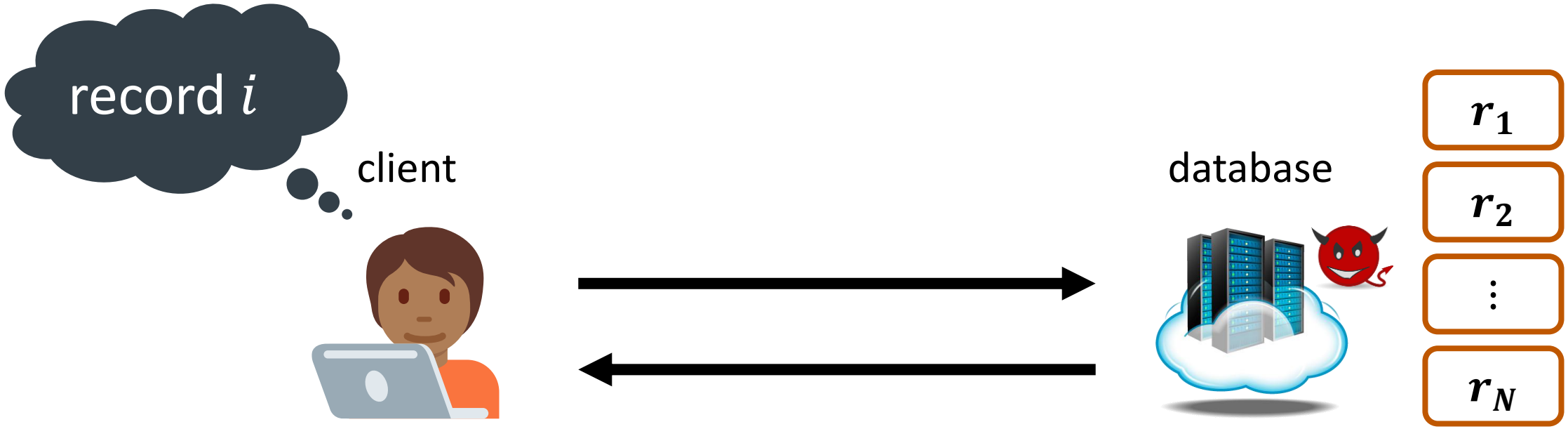
Privacy: Server does not learn i

Efficiency: communication is *sublinear* in database size (ideally: $\text{polylog}(N)$)

Correctness: Client learns r_i

Private Information Retrieval (PIR)

[CGKS95]

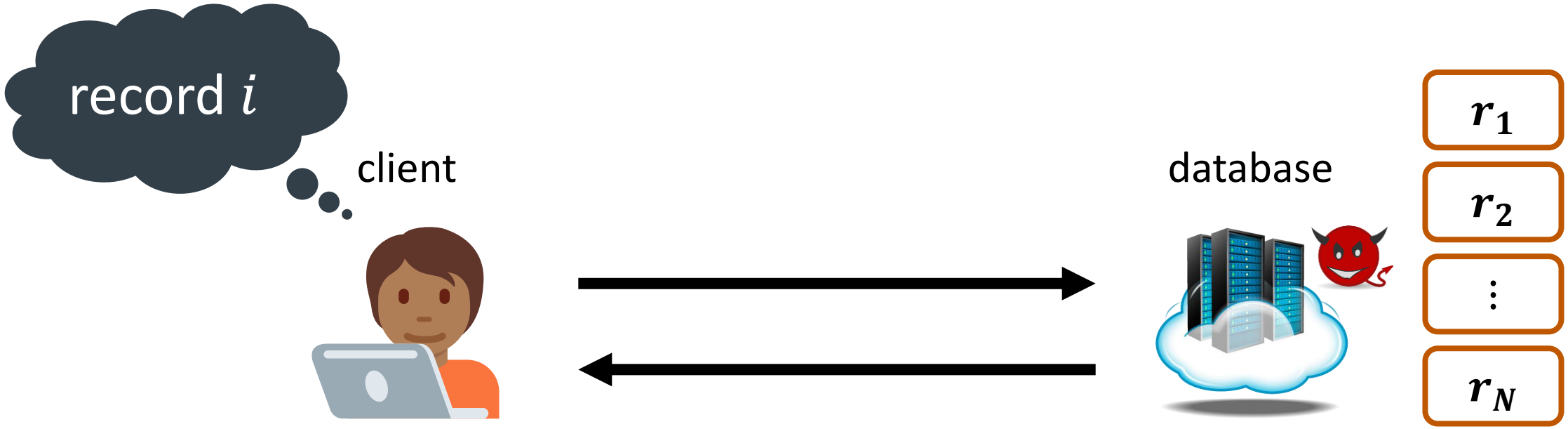


Basic building block in many privacy-preserving protocols

- Metadata-private messaging
- Certificate transparency auditing
- Private content delivery
- Contact discovery
- Private web search
- Private navigation
- Private contact tracing
- Private DNS
- Private blocklist lookup

Private Information Retrieval (PIR)

[CGKS95]

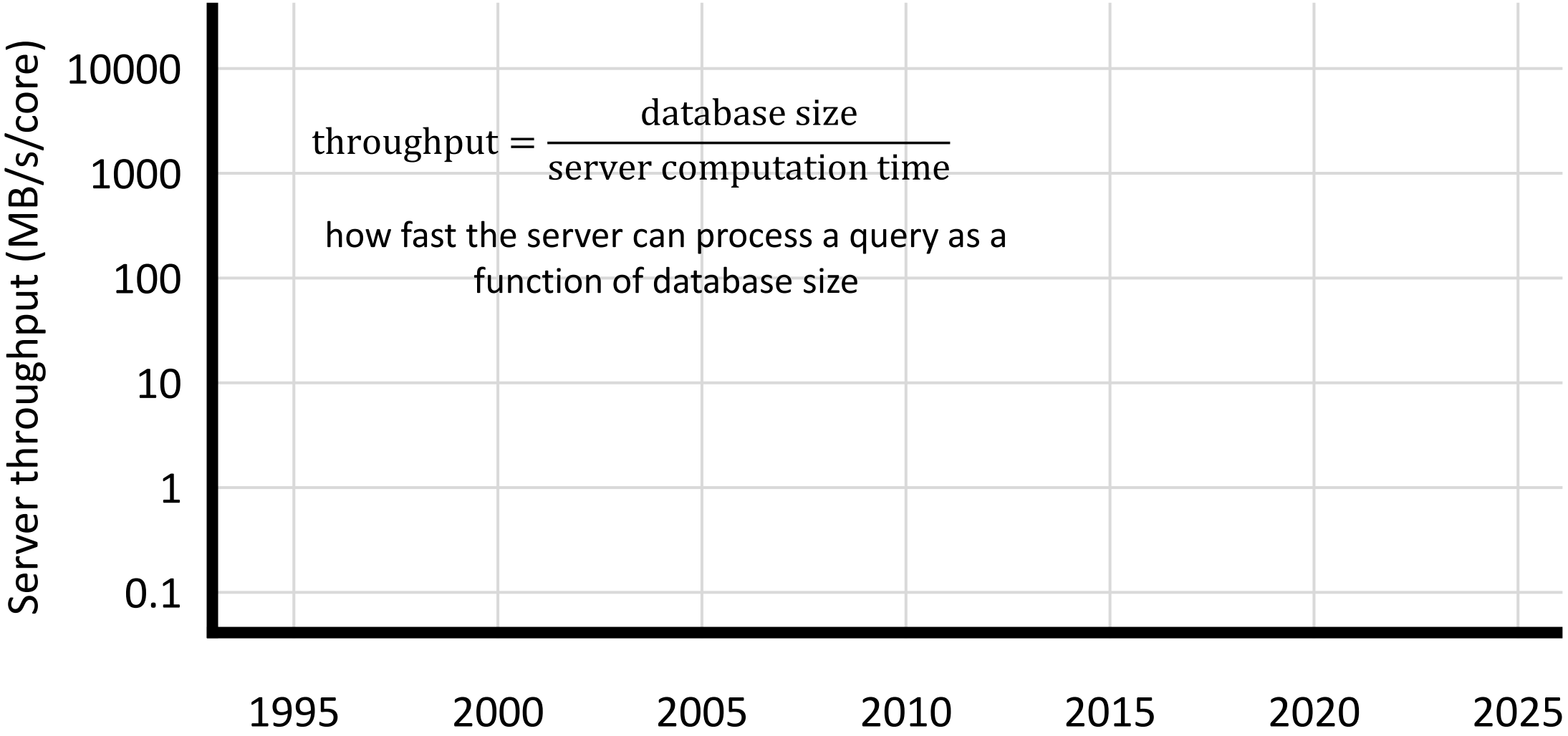


Basic building block in many privacy-preserving protocols

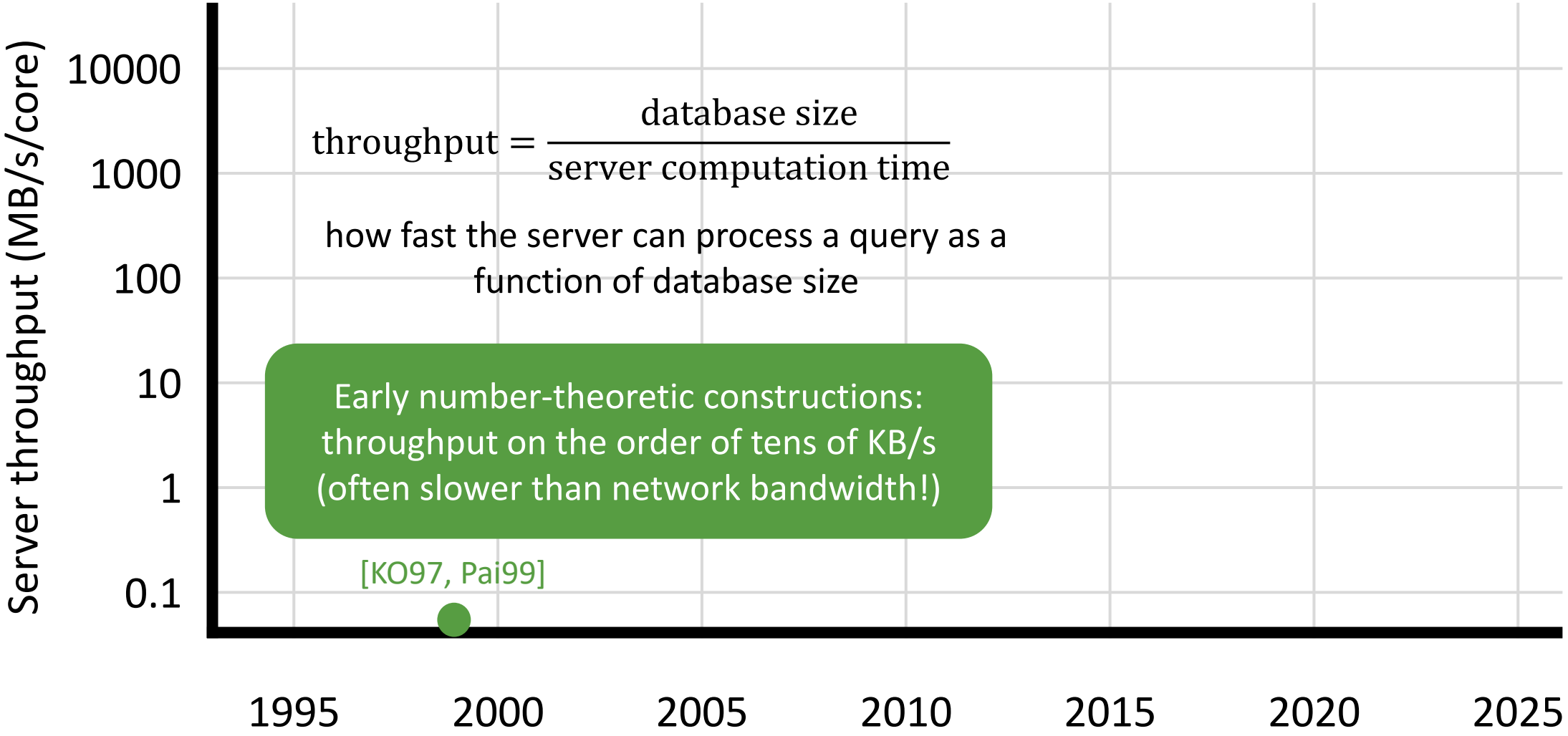
Recent deployments:

- Apple: private caller ID lookup, private image search
- Meta: advanced browsing protection

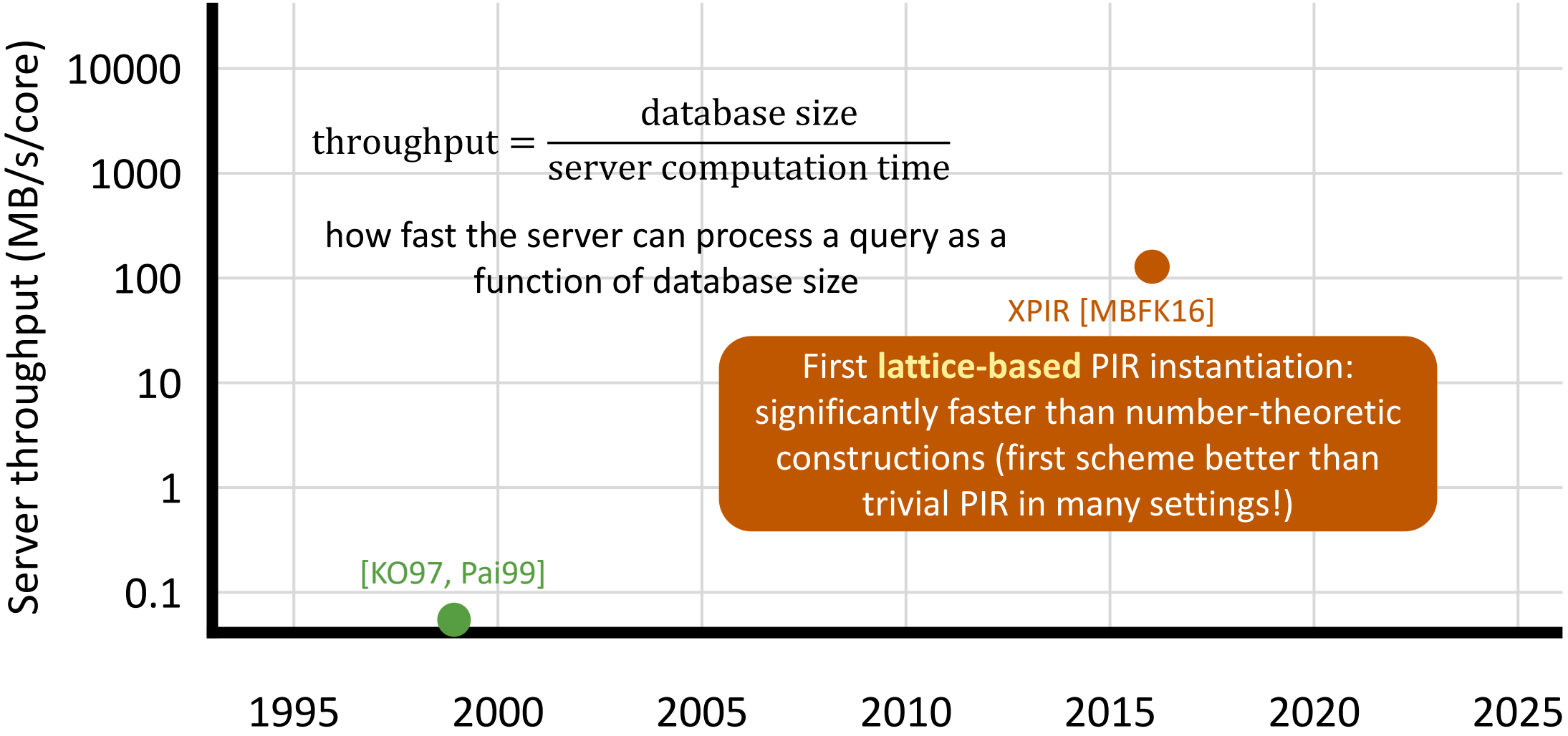
30 Years of PIR Research



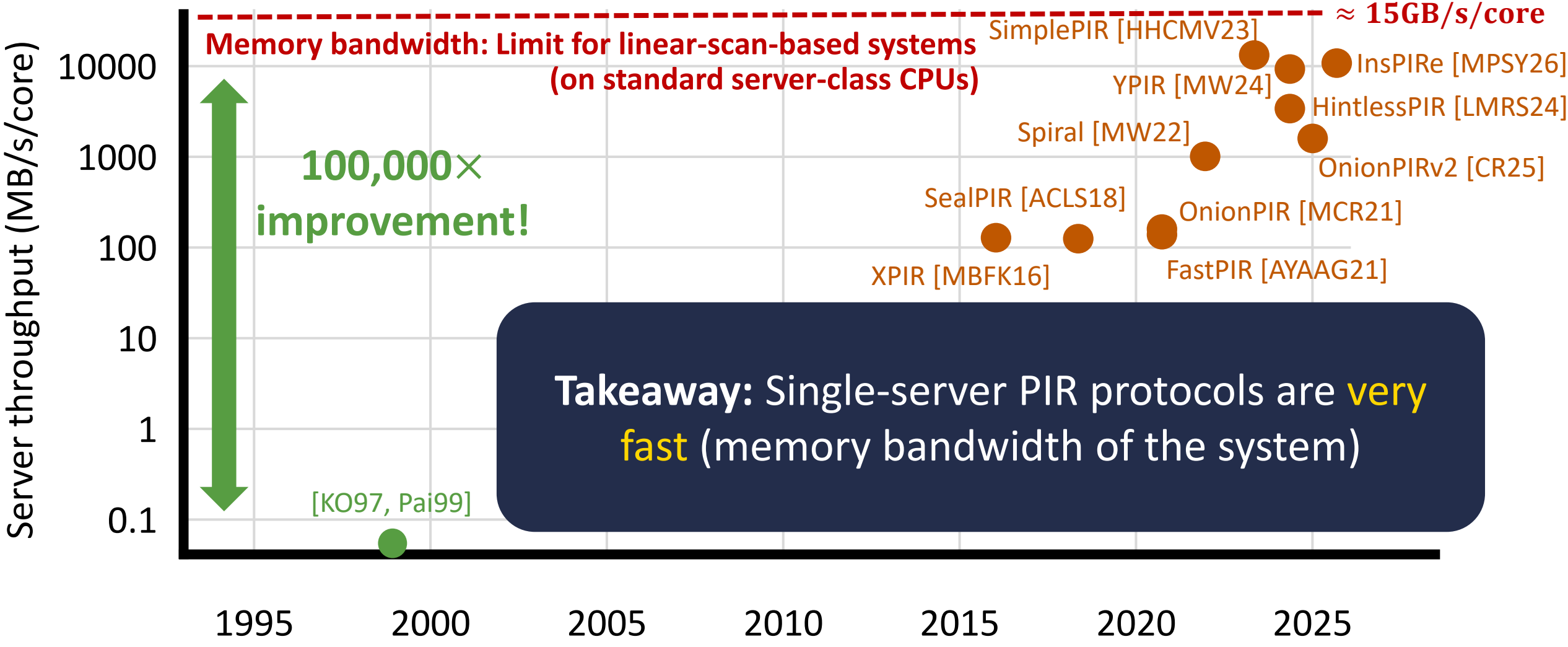
30 Years of PIR Research



30 Years of PIR Research



30 Years of PIR Research



Beyond Memory Bandwidth

[LG15, MW24, LHC25]

Can we go beyond memory bandwidth with a linear-scan PIR?

Yes, but in an ***amortized*** sense

Idea: if reading memory is the bottleneck, then schedule more CPU operations per byte of memory read

Cross-client batching: process a batch of k queries in parallel (from different, *non-coordinating* clients)

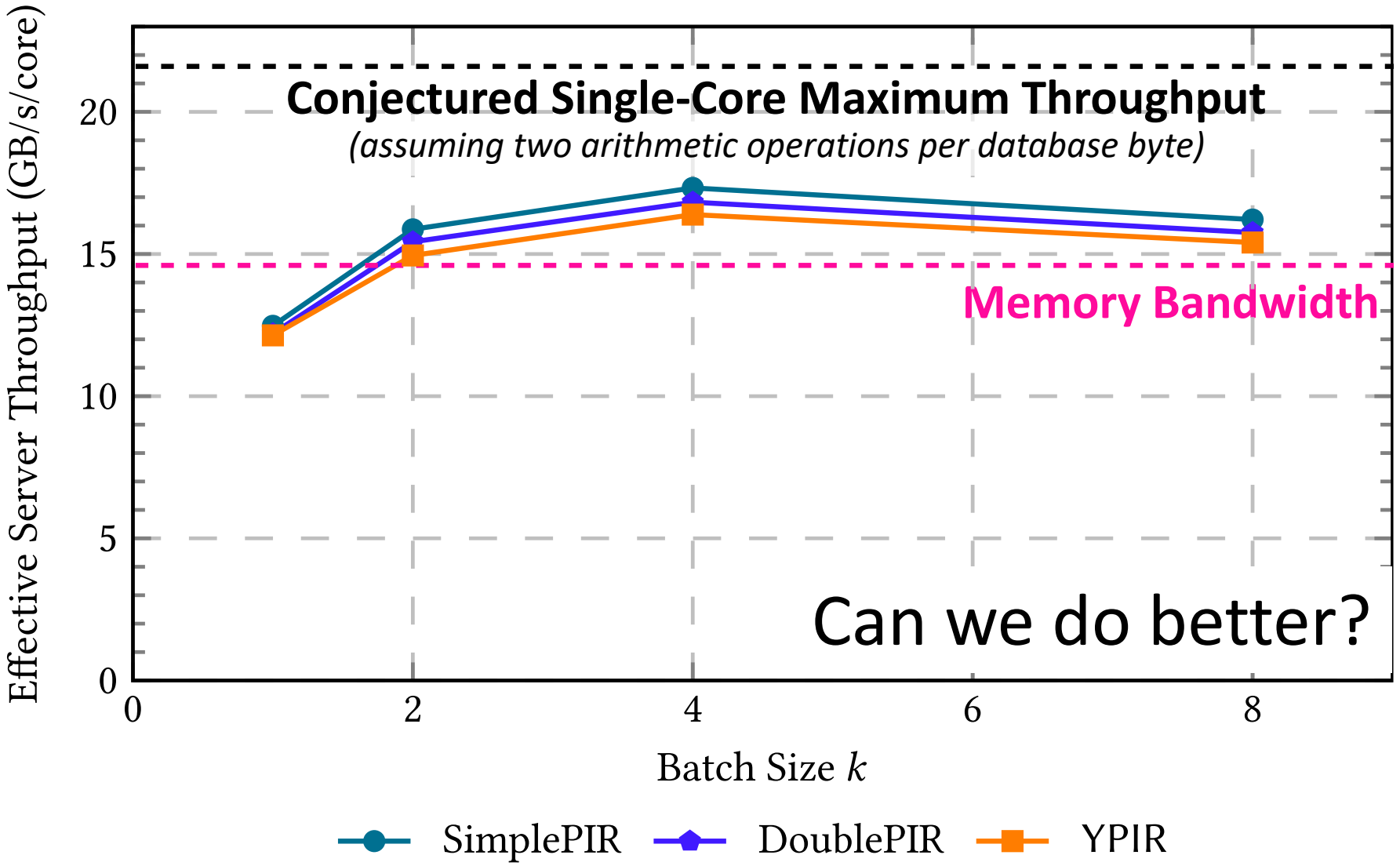
In many recent lattice-based PIR protocols, query processing is a product between a query vector and a database matrix

Cross-client batching: matrix-vector product ⇒ matrix-matrix product

Beyond Memory Bandwidth

[MW24]

$$\frac{k \cdot |\text{DB}|}{\text{Server time}}$$



Hardware Acceleration for PIR

Main computation in PIR: matrix multiplication

NVIDIA L40S

Memory bandwidth: 864 GB/s

64-core CPU: 1.4 trillion FLOPS

Computation: 91.6 trillion 32-bit (floating point) operations/s

Tensor cores: special architecture tailored for **low-precision** matrix multiplication operations for machine learning

INT8-INT32 tensor cores: takes two input matrices $\boldsymbol{A}, \boldsymbol{B}$ over $\mathbb{Z}_{2^8}$ and outputs product $\boldsymbol{AB}$ over $\mathbb{Z}_{2^{32}}$ $\left(\mathbb{Z}_{2^8} \times \mathbb{Z}_{2^8} \to \mathbb{Z}_{2^{32}}\right)$

Throughput: 733 trillion INT-8 operations/s

"tensor core model of computation"

recent Intel CPUs also support it (Intel AMX)

Hardware Acceleration for PIR

Main computation in PIR: matrix multiplication

NVIDIA L40S

Memory bandwidth: 864 GB/s

64-core CPU: 1.4 trillion FLOPS

Computation: 91.6 trillion 32-bit (floating point) operations/s

Tensor cores: special architecture tailored for **low-precision** matrix multiplication operations for machine learning

INT8-INT32 tensor cores: takes two input matrices $\boldsymbol{A}, \boldsymbol{B}$ over $\mathbb{Z}_{2^8}$ and outputs product $\boldsymbol{AB}$ over $\mathbb{Z}_{2^{32}}$ $\left(\mathbb{Z}_{2^8} \times \mathbb{Z}_{2^8} \to \mathbb{Z}_{2^{32}}\right)$

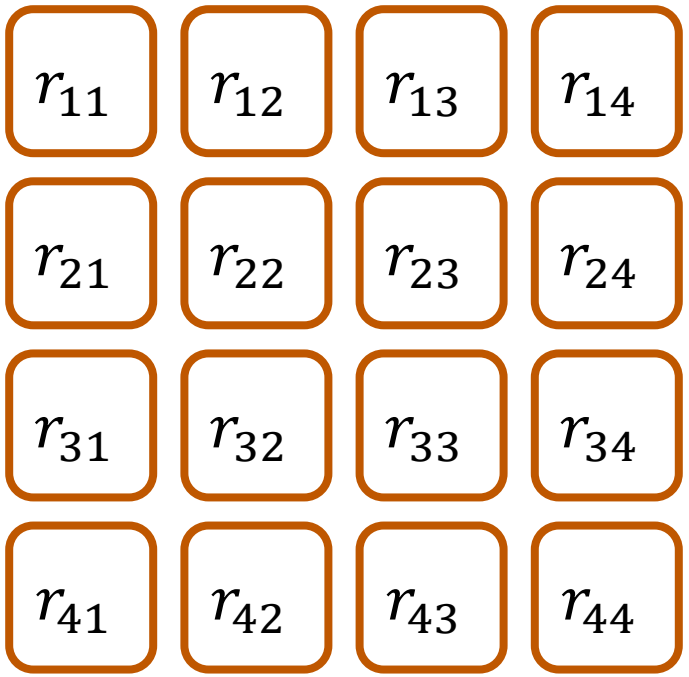
Throughput: 733 trillion INT-8 operations/s

Can we leverage tensor cores to accelerate PIR?

PIR from Homomorphic Encryption

[KO97]

Starting point: a $\sqrt{N}$ construction (N = number of records)

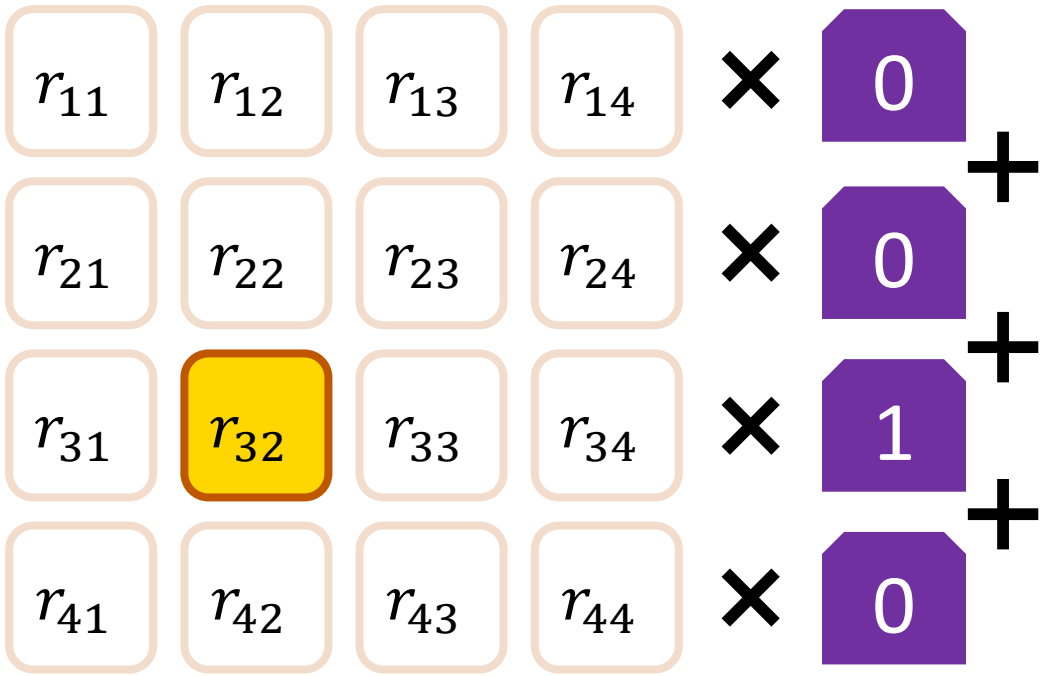


Arrange the database as a $\sqrt{N}$-by-$\sqrt{N}$ matrix

PIR from Homomorphic Encryption

[KO97]

Starting point: a $\sqrt{N}$ construction (N = number of records)



Encrypt a 0/1 vector indicating the row containing the desired record

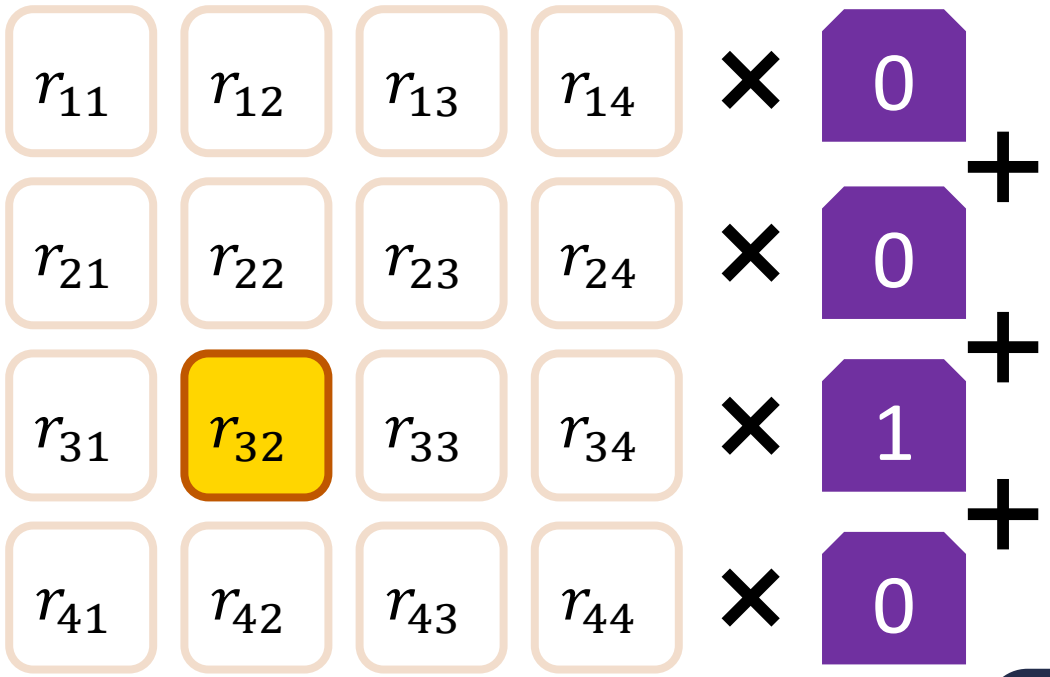
Arrange the database as a $\sqrt{N}$-by-$\sqrt{N}$ matrix

Homomorphically compute product between query vector and database matrix

PIR from Homomorphic Encryption

[KO97]

Starting point: a $\sqrt{N}$ construction (N = number of records)



Encrypt a 0/1 vector indicating the row containing the desired record

Arrange the database as a $\sqrt{N}$-by-$\sqrt{N}$ matrix

Database is in the clear, so *additive* homomorphism suffices

PIR from Homomorphic Encryption

[KO97]

Starting point: a $\sqrt{N}$ construction (N = number of records)

Client decrypts to learn records

Encrypt a 0/1 vector indicating the row containing the desired record

r_{31} r_{32} r_{33} r_{34}

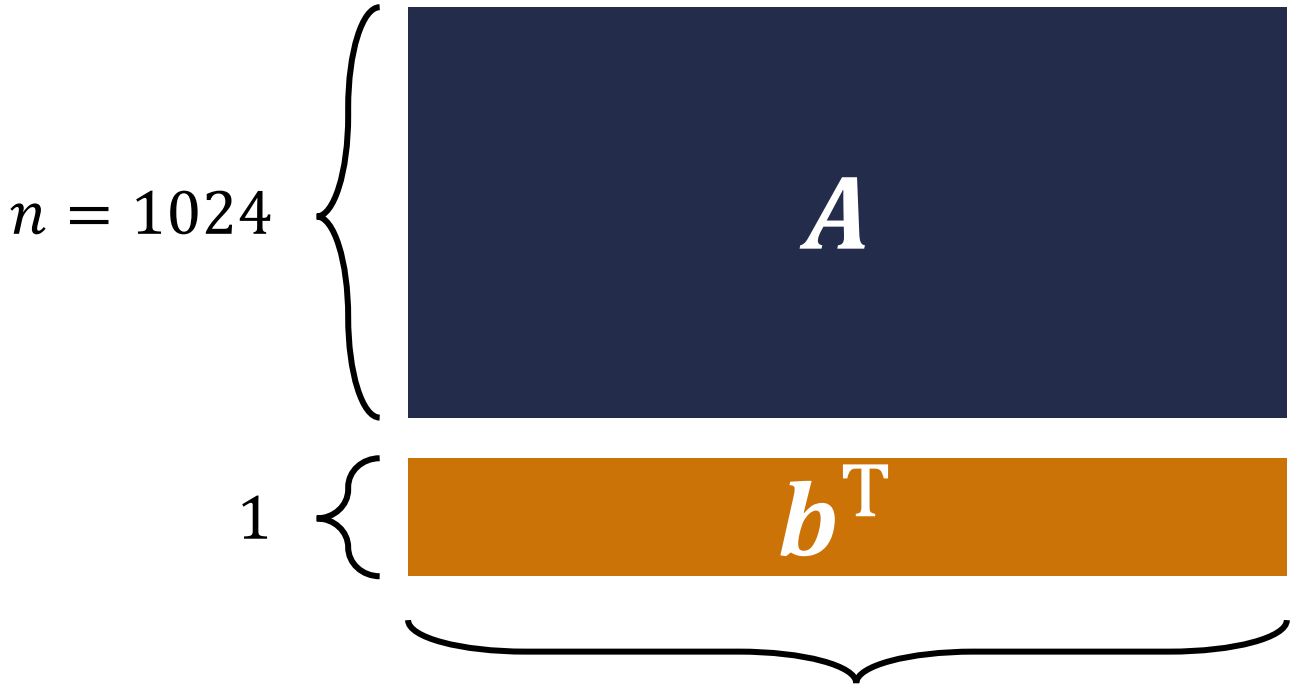
Response size: $O_\lambda(\sqrt{N})$

Homomorphically compute product between query vector and database matrix

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Building block: Regev's linearly homomorphic encryption from LWE [Reg05]



Elements of $\boldsymbol{A}$ are random 32-bit integers

$\boldsymbol{b}$ is a ciphertext (encrypts an indicator vector)

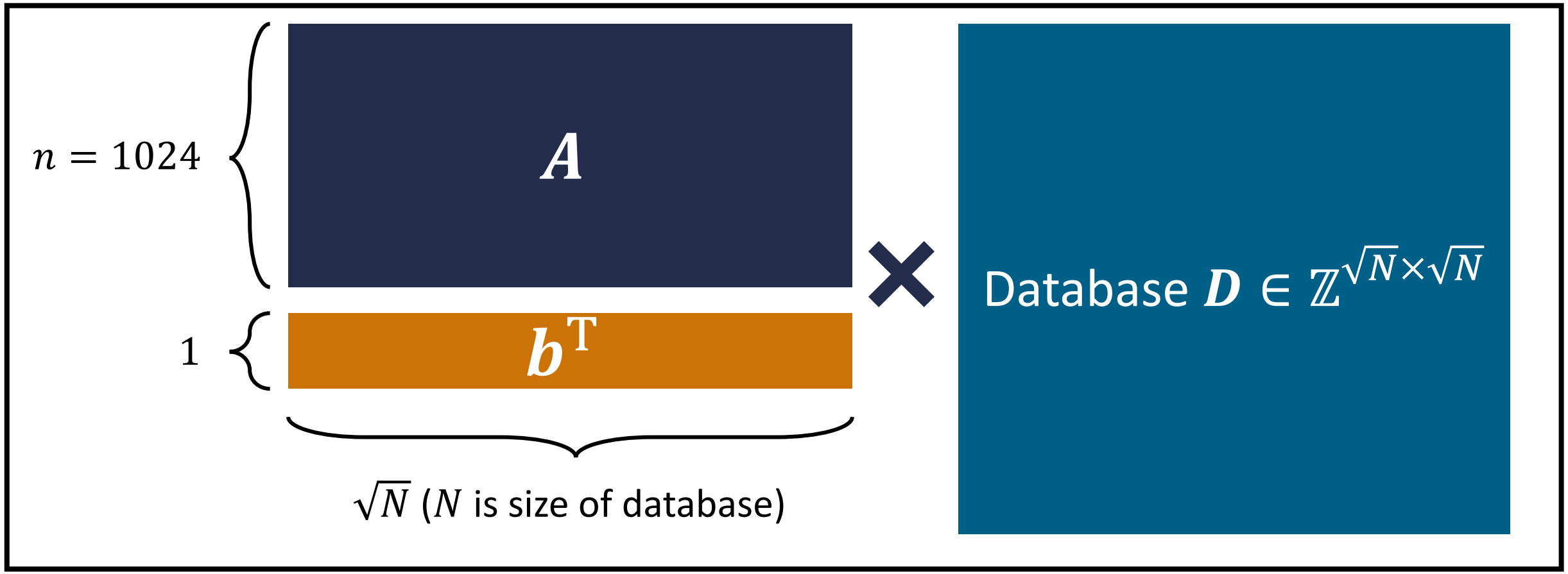
Query structure in SimplePIR

All components are elements of $\mathbb{Z}_q = \mathbb{Z}_{2^{32}}$ (i.e., 32-bit integers)

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Building block: Regev's linearly homomorphic encryption from LWE [Reg05]



All components are elements of $\mathbb{Z}_q = \mathbb{Z}_{2^{32}}$ (i.e., 32-bit integers)

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Key insight in SimplePIR: $\boldsymbol{A}$ is query-independent so move computation of $\boldsymbol{AD}$ offline

Given $\boldsymbol{b}^{\mathrm{T}}\boldsymbol{D}$ and $\boldsymbol{AD}$, client can decrypt and learn the record

$\boldsymbol{AD}$

offline phase

online phase

$\boldsymbol{b}^{\mathrm{T}}$

$\boldsymbol{b}^{\mathrm{T}}\boldsymbol{D}$

Database

$\boldsymbol{D} \in \mathbb{Z}_p^{\sqrt{N}\times\sqrt{N}}$

Query and response size:

$\sqrt{N}$ elements over $\mathbb{Z}_q$

Server computation:

$2N$ operations over $\mathbb{Z}_q$

2 operations per database entry

SimplePIR: Lightweight PIR from LWE

[HHCMV23]

Key insight in SimplePIR: $\boldsymbol{A}$ is query-independent so move computation of $\boldsymbol{AD}$ offline

Limitation: hint is **large** ($n \times$ larger than response) and needs to be updated whenever database changes

$\boldsymbol{AD}$

offline phase

online phase

$\boldsymbol{b}^{\mathbf{T}}$

$\boldsymbol{b}^{\mathbf{T}}\boldsymbol{D}$

Very GPU-friendly: offline phase and online phase consist of matrix multiplications

Database $\boldsymbol{D} \in \mathbb{Z}_p^{\sqrt{N} \times \sqrt{N}}$

On modern processors, can process query at *memory bandwidth*

2 operations per database entry

Query and response size:
$\sqrt{N}$ elements over $\mathbb{Z}_q$

PIR with *Silent* Preprocessing

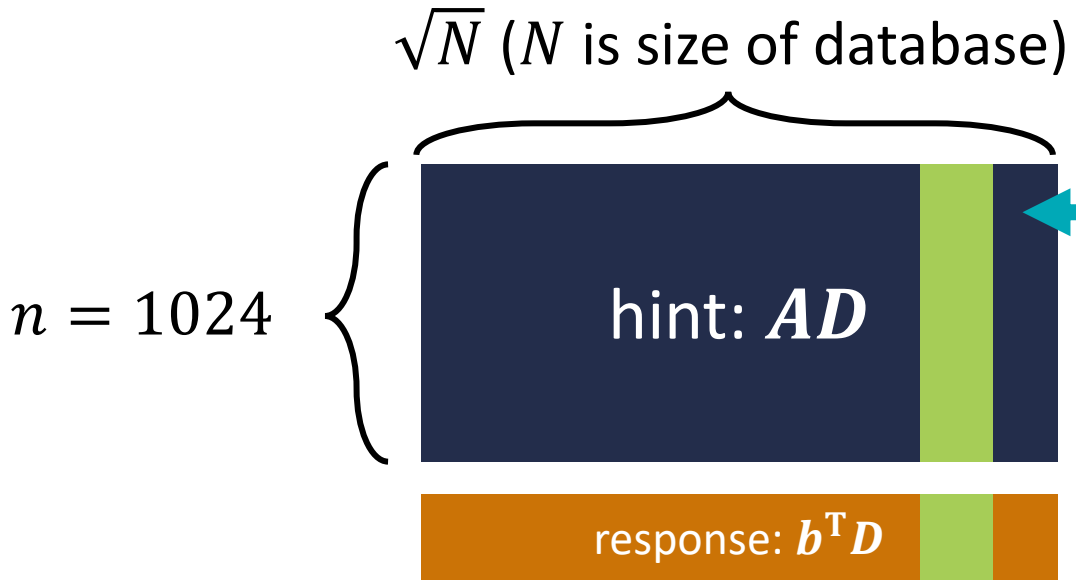
Silent preprocessing: allow **offline** preprocessing, but **no communication**

No need to manage hints; better suited for dynamic databases

Approach: *compress* the hint and include as part of the response

Why is the SimplePIR hint so large?

[Tiptoe; HDCZ23]
[HintlessPIR; LMRS24]
[YPIR; MW24]
[InsPIRe; MPSY26]



j^{th} column of hint is the "randomness" for the ciphertext that encrypts the j^{th} record

Hint is large because Regev encryption has poor rate: encrypting one scalar requires $(n+1)$ $\mathbb{Z}_q$ elements

Higher Rate using Polynomial Rings

Common technique to get better rate: use polynomial rings

Ciphertext space:

$$\mathbb{Z}_q^{n+1} \longrightarrow R_q^2 = \left(\mathbb{Z}_q[x]/(x^n+1)\right)^2$$

degree-n polynomials with coefficients in $\mathbb{Z}_q$

Each ciphertext now encrypts a degree-n polynomial (with coefficients in $\mathbb{Z}_p$)

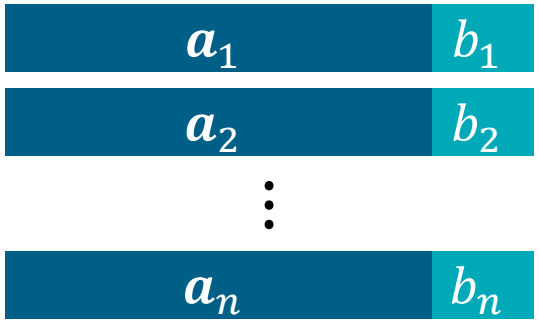
$$\text{rate} = \frac{\text{size of plaintext}}{\text{size of ciphertext}} = \frac{\log p}{(n+1)\log q} \longrightarrow \frac{n \log p}{2n \log q} = \frac{\log p}{2 \log q}$$

*rings allow **packing** more data into each ciphertext*

Recall $n = 2^{10} = 1024$, so using rings gives over 500× reduction in size

YPIR/InsPIRe: Packing LWE Ciphertexts

[MW24, MPSY26]



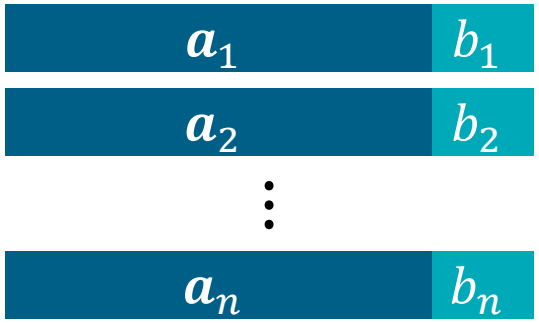
LWE ciphertexts under the secret key $\boldsymbol{s} \in \mathbb{Z}_q^n$

Regev invariant: $b_i - \boldsymbol{s}^{\mathrm{T}}\boldsymbol{a}_i \approx \lfloor q/2 \rceil \cdot \mu_i$

Can round to recover μ_i

YPIR/InsPIRe: Packing LWE Ciphertexts

[MW24, MPSY26]



LWE ciphertexts under the secret key $\boldsymbol{s} \in \mathbb{Z}_q^n$

Reinterpret $\boldsymbol{a}_i \in \mathbb{Z}_q^n$ as coefficients of a polynomial $\tilde{a}_i \in R_q$

$$\tilde{a}_i = a_{i,0} + a_{i,1}x + \cdots + a_{i,n-1}x^{n-1} \in R_q$$

Reinterpret $b_i \in \mathbb{Z}_q$ as a constant polynomial $\tilde{b}_i \in R_q$

Reinterpret $\boldsymbol{s} \in \mathbb{Z}_q^n$ as coefficients of polynomial $\tilde{s} \in R_q$ (in nega-cyclic order)

$$\tilde{s} = s_0 + s_1x^{-1} + \cdots + s_{n-1}x^{-n-1} \in R_q$$

$$R_q = \left(\mathbb{Z}_q[x]\right)/(x^n - 1)$$

Observation: Constant term of $\tilde{b}_i - \tilde{s}\tilde{a}_i = b_i - \boldsymbol{s}^{\mathrm{T}}\boldsymbol{a}_i \approx \lfloor q/2 \rfloor \cdot \mu_i$

$\left(\tilde{a}_i, \tilde{b}_i\right)$ is an RLWE ciphertext that decrypts to a value with the right constant term

Suppose we can homomorphically zero-out the high-order terms

$\left(\tilde{a}_i', \tilde{b}_i'\right)$ is an RLWE ciphertext that decrypts to $b_i - \boldsymbol{s}^{\mathrm{T}}\boldsymbol{a}_i$

Packed ciphertext is then $\left(\sum_{i\in[n]} \tilde{a}_i' x^{i-1}, \sum_{i\in[n]} \tilde{b}_i' x^{i-1}\right)$, which decrypts to $\approx \sum_{i\in[n]} \lfloor q/2 \rfloor \cdot \mu_i x^{i-1}$

Homomorphic Trace Evaluation

Input: $(\tilde{a}_i, \tilde{b}_i)$ is an RLWE ciphertext that encrypts a polynomial $\mu_i \in R_p$

Output: $(\tilde{a}'_i, \tilde{b}'_i)$ is an RLWE ciphertext that encrypts the constant $\mu_i(0) \in \mathbb{Z}_p$

Let $R = (\mathbb{Z}[x])/(x^n + 1)$, where $n > 4$ is a power-of-two

Fact: Trace operator $\mathrm{Tr}: R \to R$ over R maps $r \in R$ to $n \cdot r(0)$

InsPIRe [MPSY26]**:** Trace operation over R can be written as

$$\mathrm{Tr}(r) = \sum_{j=0}^{n/2-1} \left(\kappa_5^j(r) + \kappa_{2n-1}\left(\kappa_5^j(r)\right) \right)$$

κ_i are automorphisms over R

κ_5: $\quad r(x) \mapsto r(x^5)$

κ_{2n-1}: $r(x) \mapsto r(x^{2n-1})$

Technically, the trace is the sum over *all* automorphisms $\kappa_0, \ldots, \kappa_{n-1}$.
Over R, the automorphism group is generated by κ_5 and κ_{2n-1}

Straightforward to homomorphically evaluate automorphisms on RLWE ciphertexts

InsPIRe: query includes keys to perform automorphisms; server sends back packed response

Homomorphic Trace Evaluation

Input: $(\tilde{a}_i, \tilde{b}_i)$ is an RLWE ciphertext that encrypts a polynomial $\mu_i \in R_p$

Output: $(\tilde{a}'_i, \tilde{b}'_i)$ is an RLWE ciphertext that encrypts the constant $\mu_i(0) \in \mathbb{Z}_p$

Let $R = (\mathbb{Z}[x])/(x^n + 1)$, where $n > 4$ is a power-of-two

Fact: Trace operator $\mathrm{Tr}: R \to R$ over R maps $r \in R$ to $n \cdot r(0)$

InsPIRe [MPSY26]**:** Trace operation over R can be written as

$$\mathrm{Tr}(r) = \sum_{j=0}^{n/2-1} \left(\kappa_5^j(r) + \kappa_{2n-1}\left(\kappa_5^j(r)\right)\right)$$

κ_i are automorphisms over R

κ_5: $r(x) \mapsto r(x^5)$

κ_{2n-1}: $r(x) \mapsto r(x^{2n-1})$

Is this a matrix multiplication?

Distributional PIR [LHC25]: perform packing on the CPU (considered HintlessPIR approach)

VIPIR [KJWGSA26]: design alternative GPU-friendly packing (requires offline communication)

Spoiler: It is!

A Deep Dive into the Trace Computation

Suppose $(\tilde{a}, \tilde{b})$ is an RLWE encryption of μ with secret key $\tilde{s}$: $\tilde{b} - \tilde{s}\tilde{a} \approx \mu$

dropping scaling factor here

For any automorphism κ, this means $\kappa(\tilde{b}) - \kappa(\tilde{s}) \cdot \kappa(\tilde{a}) \approx \kappa(\mu)$

$\left(\kappa(\tilde{a}), \kappa(\tilde{b})\right)$ is an encryption of $\kappa(\mu)$ under the transformed key $\kappa(s)$

To support homomorphic evaluation of κ, client provides a key-switching key [GHS12, BGV12]

$$(\widetilde{\boldsymbol{a}}_{\mathrm{ks}}, \widetilde{\boldsymbol{b}}_{\mathrm{ks}}) = (\widetilde{\boldsymbol{a}}_{\mathrm{ks}}, \tilde{s}\widetilde{\boldsymbol{a}}_{\mathrm{ks}} + \boldsymbol{e} - \kappa(s) \cdot \boldsymbol{g})$$

$\boldsymbol{g} = \left[2^0, 2^1, \ldots, 2^{\lceil \log q \rceil - 1}\right]$ is the gadget vector [MP12]

$\boldsymbol{g}^{-1}(\cdot)$ is bit-decomposition operator

Key switching is a **linear** function

New ciphertext under $\tilde{s}$: $\left(\widetilde{\boldsymbol{a}}_{\mathrm{ks}}^{\mathrm{T}} \cdot \boldsymbol{g}^{-1}\left(\kappa(\tilde{a})\right),\ \kappa(\tilde{b}) + \widetilde{\boldsymbol{b}}_{\mathrm{ks}}^{\mathrm{T}} \cdot \boldsymbol{g}^{-1}\left(\kappa(\tilde{a})\right)\right)$

Key observation: some components can be fixed, others depend on the query [LMRS24]

A Deep Dive into the Trace Computation

Suppose $(\tilde{a}, \tilde{b})$ is an RLWE encryption of μ with secret key $\tilde{s}$: $\tilde{b} - \tilde{s}\tilde{a} \approx \mu$

For any automorphism κ, this means $\kappa(\tilde{b}) - \kappa(\tilde{s}) \cdot \kappa(\tilde{a}) \approx \kappa(\mu)$

$\left(\kappa(\tilde{a}), \kappa(\tilde{b})\right)$ is an encryption of $\kappa(\mu)$ under the transformed key $\kappa(s)$

To support homomorphic evaluation of κ, client provides a key-switching key [GHS12, BGV12]

$$(\tilde{\boldsymbol{a}}_{\mathrm{ks}}, \tilde{\boldsymbol{b}}_{\mathrm{ks}}) = (\tilde{\boldsymbol{a}}_{\mathrm{ks}}, \tilde{s}\tilde{\boldsymbol{a}}_{\mathrm{ks}} + \boldsymbol{e} - \kappa(s) \cdot \boldsymbol{g})$$

$\boldsymbol{g} = \left[2^0, 2^1, \ldots, 2^{\lceil \log q \rceil - 1}\right]$ is the gadget vector [MP12]

$\boldsymbol{g}^{-1}(\cdot)$ is bit-decomposition operator

Key switching is a **linear** function

New ciphertext under $\tilde{s}$: $\left(\tilde{\boldsymbol{a}}_{\mathrm{ks}}^{\mathrm{T}} \cdot \boldsymbol{g}^{-1}(\kappa(\tilde{a})),\ \kappa(\tilde{b}) + \tilde{\boldsymbol{b}}_{\mathrm{ks}}^{\mathrm{T}} \cdot \boldsymbol{g}^{-1}(\kappa(\tilde{a}))\right)$

Key observation: some components can be fixed, others depend on the query [LMRS24]

A Deep Dive into the Trace Computation

$$\mathrm{Tr}(r) = \sum_{j=0}^{n/2-1} \left(\kappa_5^j(r) + \kappa_{2n-1}\left(\kappa_5^j(r)\right) \right)$$

What does this look like homomorphically? Say $(\tilde{a}, \tilde{b})$ encrypts r and (a^*, b^*) encrypts $\mathrm{Tr}(r)$

Observation 1: a^* can be computed offline; only b^* needs to be computed online

Observation 2: Each key-switch is an inner product with something that only depends on $\tilde{a}$

Key switching: $\left(\tilde{\boldsymbol{a}}_{\mathrm{ks}}^{\mathrm{T}} \cdot \boldsymbol{g}^{-1}\left(\kappa(\tilde{a})\right),\ \ \kappa\left(\tilde{b}\right) + \tilde{\boldsymbol{b}}_{\mathrm{ks}}^{\mathrm{T}} \cdot \boldsymbol{g}^{-1}\left(\kappa(\tilde{a})\right) \right)$

A Deep Dive into the Trace Computation

$$\mathrm{Tr}(r) = \sum_{j=0}^{n/2-1} \left(\kappa_5^j(r) + \kappa_{2n-1}\left(\kappa_5^j(r)\right) \right)$$

What does this look like homomorphically? Say $(\tilde{a}, \tilde{b})$ encrypts r and (a^*, b^*) encrypts $\mathrm{Tr}(r)$

Observation 1: a^* can be computed offline; only b^* needs to be computed online

Observation 2: Each key-switch is an inner product with something that only depends on $\tilde{a}$

$$\tilde{b}^* = \tilde{b} + \sum_{j\in[n/2-1]} \left(\kappa_5^{j-1}(\widetilde{\boldsymbol{b}}_{\mathrm{ks},5})\right)^{\mathrm{T}} \boldsymbol{g}^{-1}(\tilde{\alpha}_j) + \sum_{j\in[n/2-1]} \left(\kappa_{2n-1}\left(\kappa_5^{j-1}(\widetilde{\boldsymbol{b}}_{\mathrm{ks},5})\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}(\tilde{\alpha}_{n/2+j}) + \widetilde{\boldsymbol{b}}_{\mathrm{ks},2n-1}^{\mathrm{T}}\, \boldsymbol{g}^{-1}(\tilde{\alpha}_{n/2})$$

All of the α_j are fixed functions of $\widetilde{\boldsymbol{a}}, \widetilde{\boldsymbol{a}}_{\mathrm{ks},5}, \widetilde{\boldsymbol{a}}_{\mathrm{ks},2n-1}$

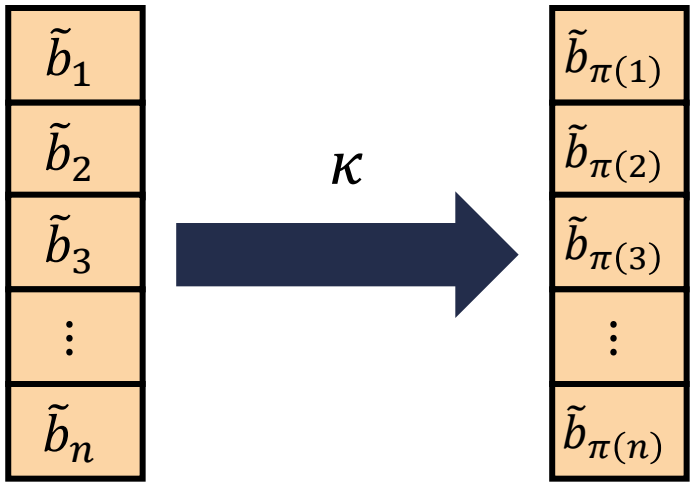
Is this a matrix multiplication?

A Deep Dive into the Trace Computation

$$\tilde{b}^* = \tilde{b} + \sum_{j\in[n/2-1]} \left(\kappa_5^{j-1}(\tilde{\boldsymbol{b}}_{\mathrm{ks},5})\right)^{\mathrm{T}} \boldsymbol{g}^{-1}(\tilde{\alpha}_j) + \sum_{j\in[n/2-1]} \left(\kappa_{2n-1}\left(\kappa_5^{j-1}(\tilde{\boldsymbol{b}}_{\mathrm{ks},5})\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}(\tilde{\alpha}_{n/2+j}) + \tilde{\boldsymbol{b}}_{\mathrm{ks},2n-1}^{\mathrm{T}}\, \boldsymbol{g}^{-1}(\tilde{\alpha}_{n/2})$$

all linear functions in $\tilde{\boldsymbol{b}}_{\mathrm{ks},5}$

View polynomials in their NTT representation (i.e., in evaluation representation)



$\tilde{b} \in R_q$ $\quad$ $\kappa(\tilde{b}) \in R_q$

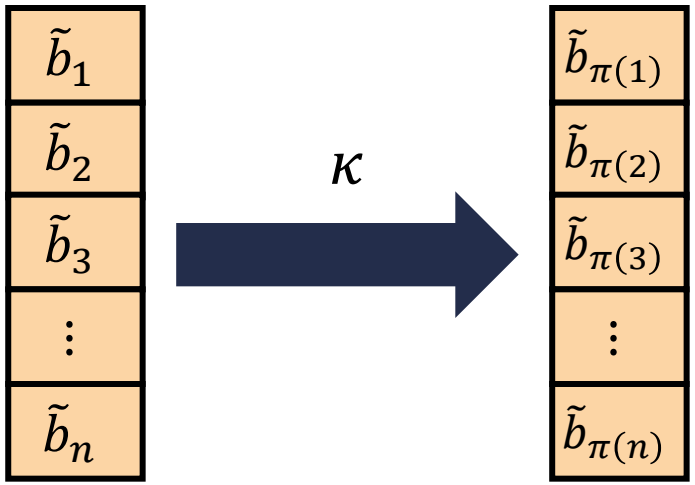
Multiplication is component-wise in NTT representation

Automorphisms permute components

A Deep Dive into the Trace Computation

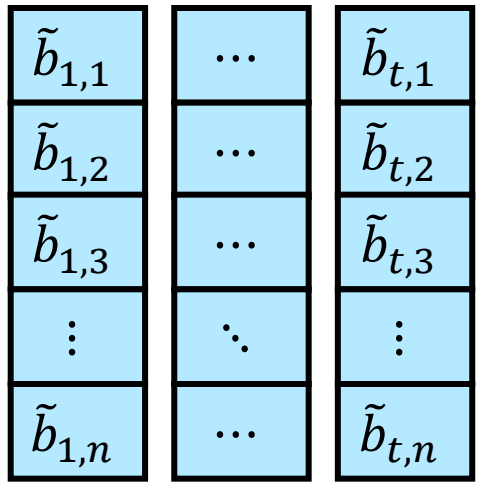
$$\tilde{b}^* = \tilde{b} + \sum_{j\in[n/2-1]} \left(\kappa_5^{j-1}\left(\tilde{\boldsymbol{b}}_{\mathrm{ks},5}\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}\left(\tilde{\alpha}_j\right) + \sum_{j\in[n/2-1]} \left(\kappa_{2n-1}\left(\kappa_5^{j-1}\left(\tilde{\boldsymbol{b}}_{\mathrm{ks},5}\right)\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}\left(\tilde{\alpha}_{n/2+j}\right) + \tilde{\boldsymbol{b}}_{\mathrm{ks},2n-1}^{\mathrm{T}}\, \boldsymbol{g}^{-1}\left(\tilde{\alpha}_{n/2}\right)$$

View polynomials in their NTT representation (i.e., in evaluation representation)

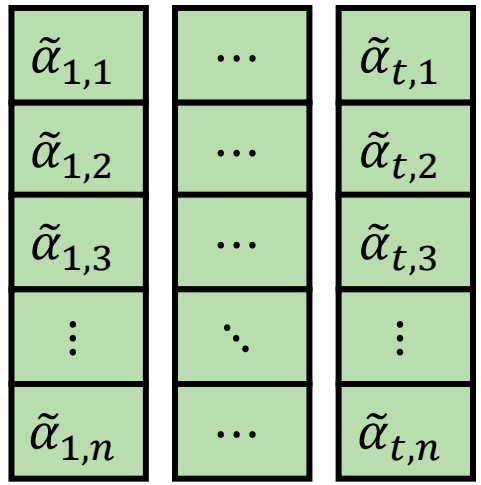


$\tilde{b} \in R_q$ $\kappa(\tilde{b}) \in R_q$

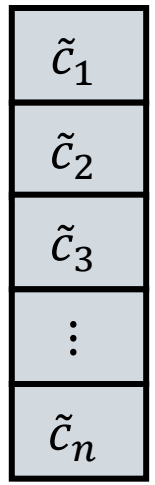
Automorphisms permute components



$\tilde{\boldsymbol{b}}_{\mathrm{ks},5}^{\mathrm{T}} \in R_q^t$



$\boldsymbol{g}^{-1}\left(\tilde{\alpha}_j\right)^{\mathrm{T}} \in R_q^t$



$$\tilde{c} = \left(\kappa\left(\tilde{\boldsymbol{b}}_{\mathrm{ks},5}\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}\left(\tilde{\alpha}_j\right) \in R_q$$

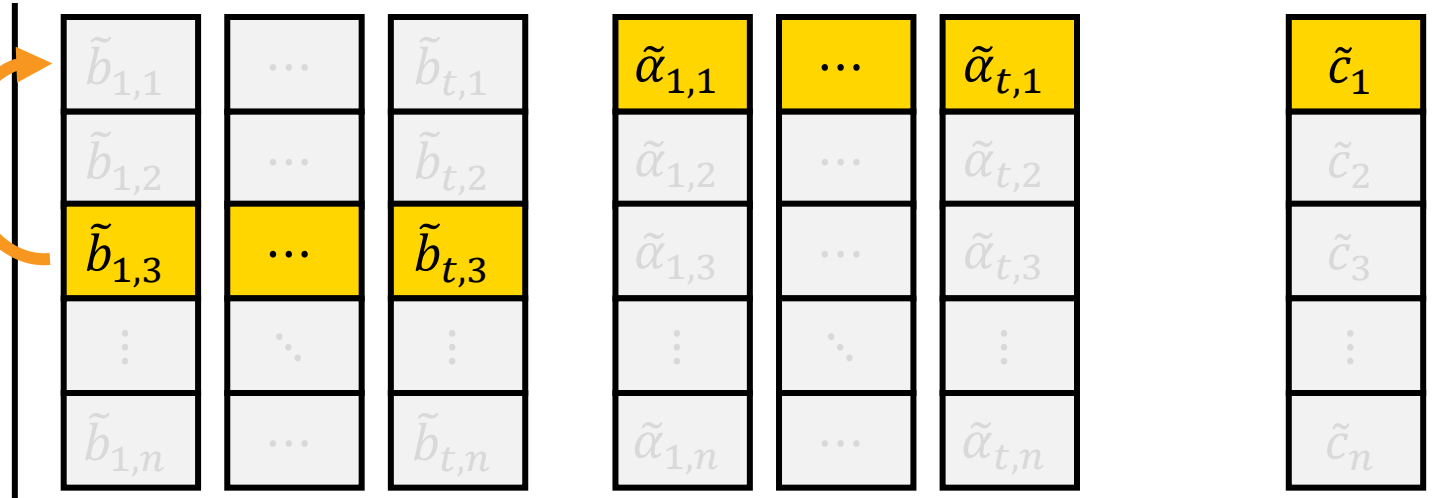
A Deep Dive into the Trace Computation

$$\tilde{b}^* = \tilde{b} + \sum_{j\in[n/2-1]} \left(\kappa_5^{j-1}\left(\widetilde{\boldsymbol{b}}_{\mathrm{ks},5}\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}\left(\tilde{\alpha}_j\right) + \sum_{j\in[n/2-1]} \left(\kappa_{2n-1}\left(\kappa_5^{j-1}\left(\widetilde{\boldsymbol{b}}_{\mathrm{ks},5}\right)\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}\left(\tilde{\alpha}_{n/2+j}\right) + \widetilde{\boldsymbol{b}}_{\mathrm{ks},2n-1}^{\mathrm{T}}\, \boldsymbol{g}^{-1}\left(\tilde{\alpha}_{n/2}\right)$$

View polynomials in their NTT representation (i.e., in evaluation representation)

$\pi(3) = 1$

Observation: each component $\tilde{c}_i$ is an inner product between one "row" of the NTT representation of $\widetilde{\boldsymbol{b}}_{\mathrm{ks},5}$ and a fixed vector



$\widetilde{\boldsymbol{b}}_{\mathrm{ks},5}^{\mathrm{T}} \in R_q^t$ $\quad$ $\boldsymbol{g}^{-1}\left(\tilde{\alpha}_j\right)^{\mathrm{T}} \in R_q^t$

$$\tilde{c} = \left(\kappa\left(\widetilde{\boldsymbol{b}}_{\mathrm{ks},5}\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}\left(\tilde{\alpha}_j\right) \in R_q$$

A Deep Dive into the Trace Computation

$$\tilde{b}^* = \tilde{b} + \sum_{j\in[n/2-1]} \left(\kappa_5^{j-1}\left(\tilde{\boldsymbol{b}}_{\mathrm{ks},5}\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}\left(\tilde{\alpha}_j\right) + \sum_{j\in[n/2-1]} \left(\kappa_{2n-1}\left(\kappa_5^{j-1}\left(\tilde{\boldsymbol{b}}_{\mathrm{ks},5}\right)\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}\left(\tilde{\alpha}_{n/2+j}\right) + \tilde{\boldsymbol{b}}_{\mathrm{ks},2n-1}^{\mathrm{T}}\, \boldsymbol{g}^{-1}\left(\tilde{\alpha}_{n/2}\right)$$

View polynomials in their NTT representation (i.e., in evaluation representation)

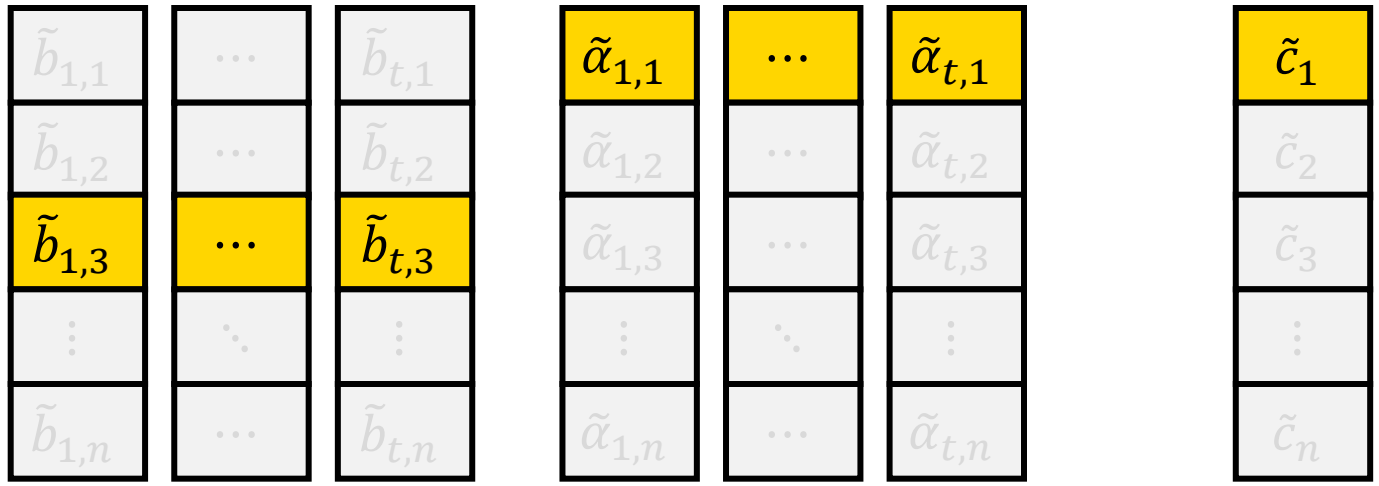
We can write the shaded term as

$$\tilde{b}_i^* = \sum_{k\in[n]} \tilde{\boldsymbol{b}}_k^{\mathrm{T}} \boldsymbol{u}_{i,k}$$

$\tilde{\boldsymbol{b}}_k^{\mathrm{T}} = k^{\mathrm{th}}$ row of NTT representation of $\tilde{\boldsymbol{b}}_{\mathrm{ks},5}^{\mathrm{T}}$

$\boldsymbol{u}_{i,k}$: fixed vector (determined in offline phase)

This is now a matrix multiplication!



$\tilde{\boldsymbol{b}}_{\mathrm{ks},5}^{\mathrm{T}} \in R_q^t$ $\quad$ $\boldsymbol{g}^{-1}\left(\tilde{\alpha}_j\right)^{\mathrm{T}} \in R_q^t$

$$\tilde{c} = \left(\kappa\left(\tilde{\boldsymbol{b}}_{\mathrm{ks},5}\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}\left(\tilde{\alpha}_j\right) \in R_q$$

A Deep Dive into the Trace Computation

$$\tilde{b}^* = \tilde{b} + \sum_{j\in[n/2-1]} \left(\kappa_5^{j-1}(\tilde{\boldsymbol{b}}_{\mathrm{ks},5})\right)^{\mathrm{T}} \boldsymbol{g}^{-1}(\tilde{\alpha}_j) + \sum_{j\in[n/2-1]} \left(\kappa_{2n-1}\left(\kappa_5^{j-1}(\tilde{\boldsymbol{b}}_{\mathrm{ks},5})\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}(\tilde{\alpha}_{n/2+j}) + \tilde{\boldsymbol{b}}_{\mathrm{ks},2n-1}^{\mathrm{T}}\, \boldsymbol{g}^{-1}(\tilde{\alpha}_{n/2})$$

View polynomials in their NTT representation (i.e., in evaluation representation)

We can write the shaded term as

$$\tilde{b}_i^* = \sum_{k\in[n]} \tilde{\boldsymbol{b}}_k^{\mathrm{T}} \boldsymbol{u}_{i,k}$$

$\tilde{\boldsymbol{b}}_k^{\mathrm{T}} = k^{\mathrm{th}}$ row of NTT representation of $\tilde{\boldsymbol{b}}_{\mathrm{ks},5}^{\mathrm{T}}$

$\boldsymbol{u}_{i,k}$: fixed vector (determined in offline phase)

This is now a matrix multiplication!

To compute the NTT representation of the shaded term:

$$\hat{\boldsymbol{b}}^{\mathrm{T}} = \left[\, \tilde{\boldsymbol{b}}_1^{\mathrm{T}} \mid \cdots \mid \tilde{\boldsymbol{b}}_n^{\mathrm{T}} \,\right] \in \mathbb{Z}_q^{nt}$$

$$\hat{\boldsymbol{U}} = \begin{bmatrix} \boldsymbol{u}_{1,1} & \cdots & \boldsymbol{u}_{n,1} \\ \vdots & \ddots & \vdots \\ \boldsymbol{u}_{1,n} & \cdots & \boldsymbol{u}_{n,1} \end{bmatrix} \in \mathbb{Z}_q^{nt\times n}$$

NTT representation is then $\hat{\boldsymbol{b}}^{\mathrm{T}}\hat{\boldsymbol{U}}$

To handle multiple independent queries, stack $\hat{\boldsymbol{b}}_1^{\mathrm{T}}, \ldots, \hat{\boldsymbol{b}}_K^{\mathrm{T}}$ into a matrix $\hat{\boldsymbol{B}}$ and compute $\hat{\boldsymbol{B}}\hat{\boldsymbol{U}}$

Back to PIR

$$\tilde{b}^* = \tilde{b} + \sum_{j \in [n/2-1]} \left(\kappa_5^{j-1}(\tilde{\boldsymbol{b}}_{\mathrm{ks},5})\right)^{\mathrm{T}} \boldsymbol{g}^{-1}(\tilde{\alpha}_j) + \sum_{j \in [n/2-1]} \left(\kappa_{2n-1}\left(\kappa_5^{j-1}(\tilde{\boldsymbol{b}}_{\mathrm{ks},5})\right)\right)^{\mathrm{T}} \boldsymbol{g}^{-1}(\tilde{\alpha}_{n/2+j}) + \tilde{\boldsymbol{b}}_{\mathrm{ks},2n-1}^{\mathrm{T}} \boldsymbol{g}^{-1}(\tilde{\alpha}_{n/2})$$

SimplePIR response

Packing component

Packing component

Both the SimplePIR response and the packing components are matrix-vector product between the client query and either the database or a matrix in the public parameters

Both operations are GPU-friendly (in **parallel** even)

Many independent queries: matrix-vector multiplication → matrix-matrix multiplication

For fast implementation, need to use modulus switching (power-of-two modulus for SimplePIR, NTT-friendly modulus for packing)

Overall Protocol Flow

No communication!

offline phase

online phase

$\boldsymbol{b}^{\mathrm{T}}, \widetilde{\boldsymbol{b}}^{\mathrm{T}}_{\mathrm{ks},5}, \widetilde{\boldsymbol{b}}^{\mathrm{T}}_{\mathrm{ks},2n-1}$

Compute:

Database
$\boldsymbol{D} \in \mathbb{Z}_p^{\sqrt{N}\times\sqrt{N}}$

$\boldsymbol{AD}$

query packing components/matrices

Performance Comparison

Setup: 4 GB database, 32 KB records

Implementation from [LHC25]

		CPU-based		GPU-based	
		HintlessPIR	**InsPIRe**	**SimplePIR**	**This Work**
Offline	Download	–	–	144 MB	–
Online	Upload	722 KB	1.04 MB	512 KB	576 KB
	Download	2.97 MB	96 KB	40 KB	112 KB
	Response Time	1.4 s	718 ms	33 ms	8.2 ms
	Throughput	2.8 GB/s	5.6 GB/s	123 GB/s	488 GB/s
	Cost (per million queries)	$489.89	$114.39	$20.49	$14.56

Excludes hint download costs

Performance Comparison

Setup: 4 GB database, 32 KB records

Implementation from [LHC25]

		CPU-based		GPU-based	
		HintlessPIR	**InsPIRe**	**SimplePIR**	
Offline	Download	–	–	144 MB	
Online	Upload			512 KB	
	Download			40 KB	112 KB
	Response Time			33 ms	8.2 ms
	Throughput	2.8 GB/s	5.6 GB/s	123 GB/s	488 GB/s
	Cost (per million queries)	$489.89	$114.39	$20.49	$14.56

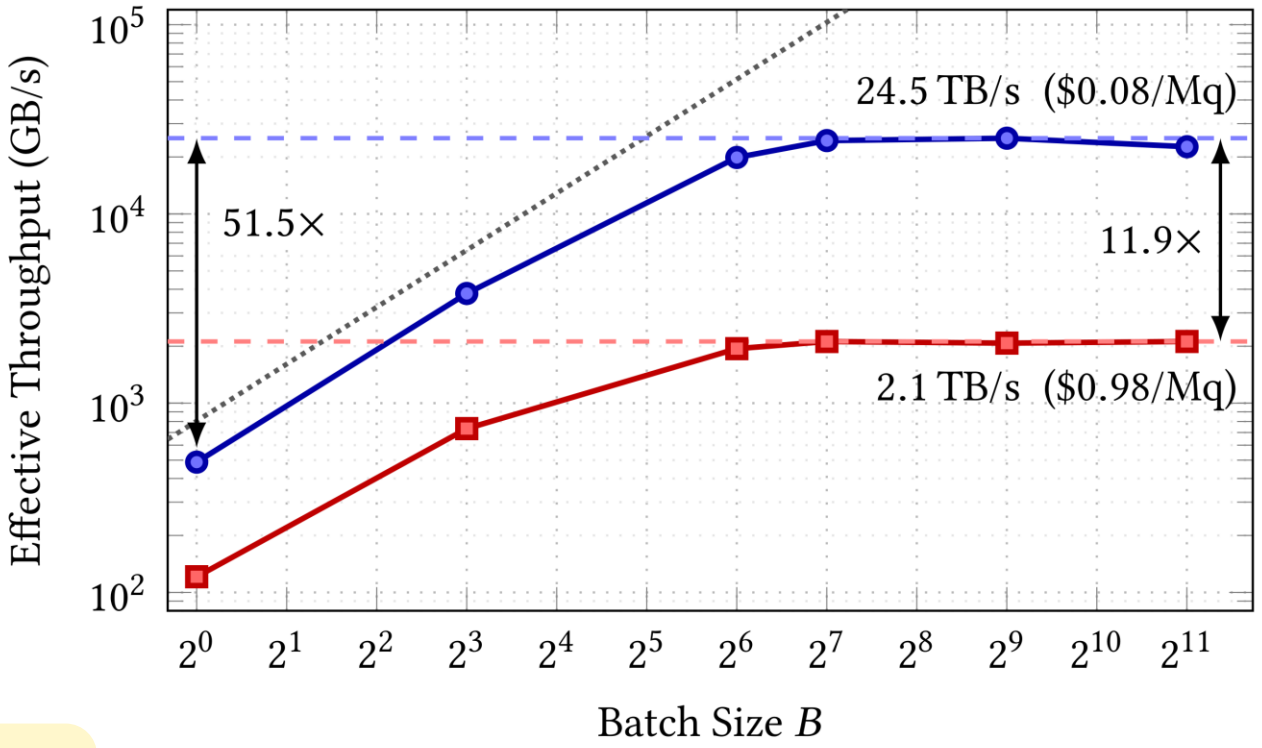
Over 88 × higher throughput than CPU-based implementations for single-query setting (but we are not compute-bound yet)

72 KB is actually fixed (can be reused across queries); reduces per-query cost to $7.93

Excludes hint download costs

Cross-Client Batching

Cross-client batching: process multiple independent queries at once (more compute saturation)



$$\frac{k \cdot |\mathrm{DB}|}{\text{Server time}}$$

Effective throughput peaks around 25 TB/s

On typical CPUs, we expect ≈ 25 GB/s/core

(could be faster with special instruction sets)

Setup: 4 GB database, 32 KB records

Application to Private Wikipedia

Goal: read (text-only) Wikipedia over PIR

As of November 2023, English Wikipedia contains $6.4 \cdot 10^6$ articles
(8 GB after compression)

Record size is 128 KB (compressed articles are all under 128 KB)

Communication: 320 KB upload, 160 KB download
Server processing: $\approx$ 60 ms

https://www.cs.utexas.edu/~dwu4/pir-demo

Hardware Acceleration for Cryptography

Modern hardware is shaped by machine learning applications

For machine learning applications, elementary operation is matrix multiplication

In addition to Boolean circuits, arithmetic circuits, RAM programs, we should also think about "matrix multiplication programs"

Modern hardware designed for "matrix multiplication programs"

For PIR: using GPU tensor cores, we can now obtain (effective) throughputs of 25-275 TB/s (275 TB/s uses fleet of 8 GPUs over a 256 GB database)

Thank You!

(paper coming soon)