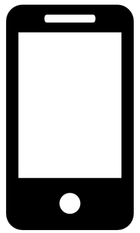
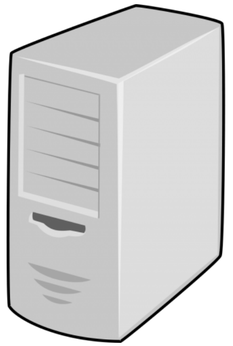
CS 350S: Privacy-Preserving Systems

Private Information Retrieval III

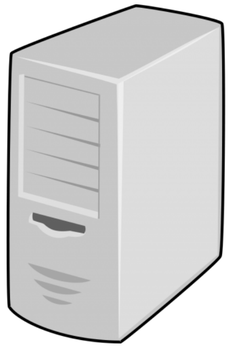
Recap: two-server PIR scheme

Write database as $\sqrt{n} \times \sqrt{n}$ matrix





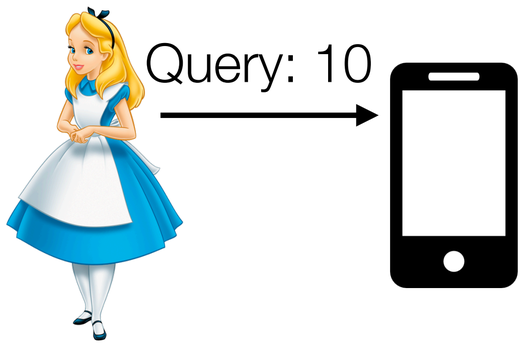
$$\begin{matrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{matrix}$$

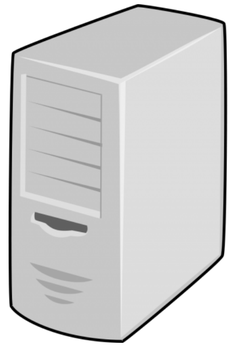


$$\begin{matrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{matrix}$$

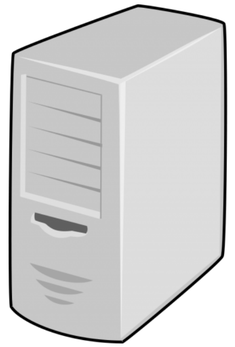
Recap: two-server PIR scheme

Write database as $\sqrt{n} \times \sqrt{n}$ matrix





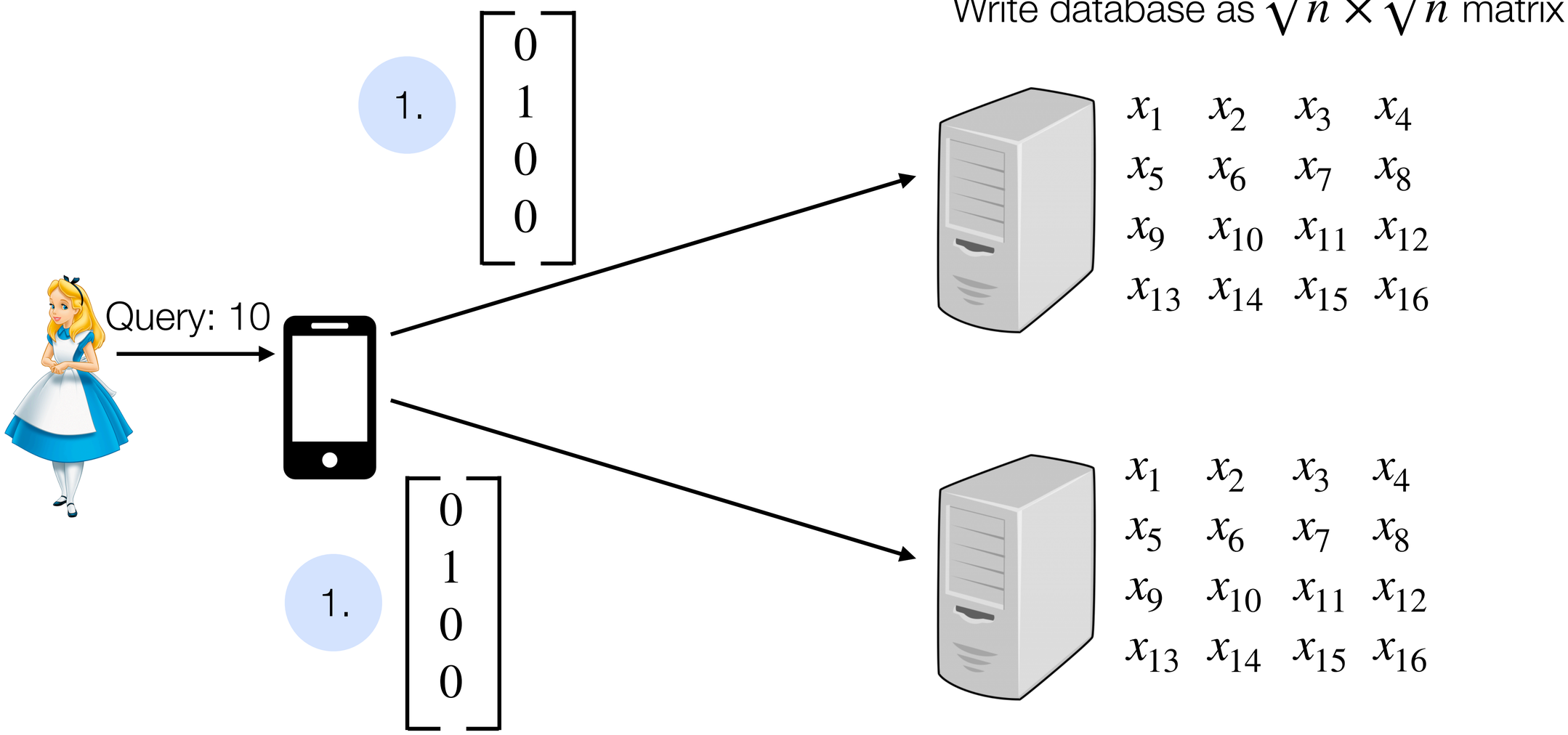
$$\begin{matrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{matrix}$$



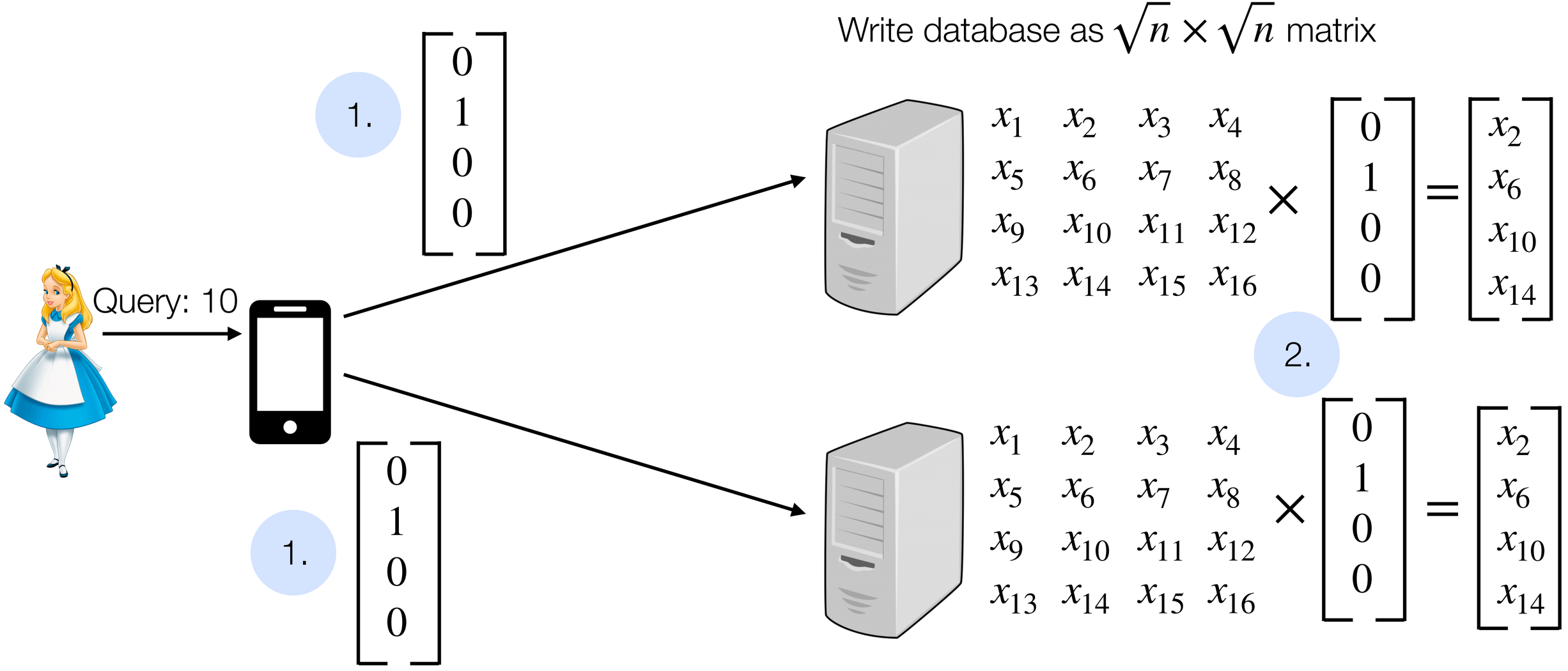
$$\begin{matrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{matrix}$$

Recap: two-server PIR scheme

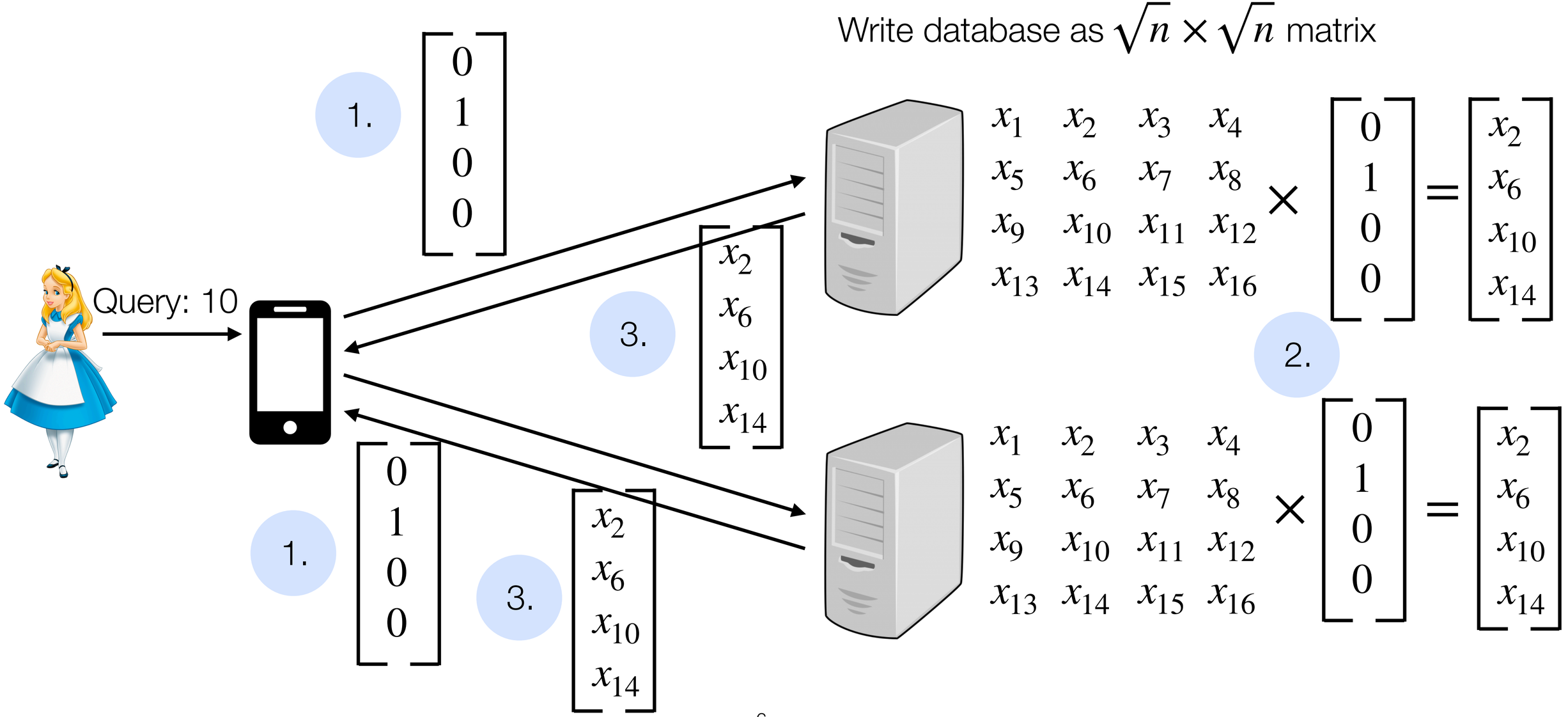
Write database as $\sqrt{n} \times \sqrt{n}$ matrix



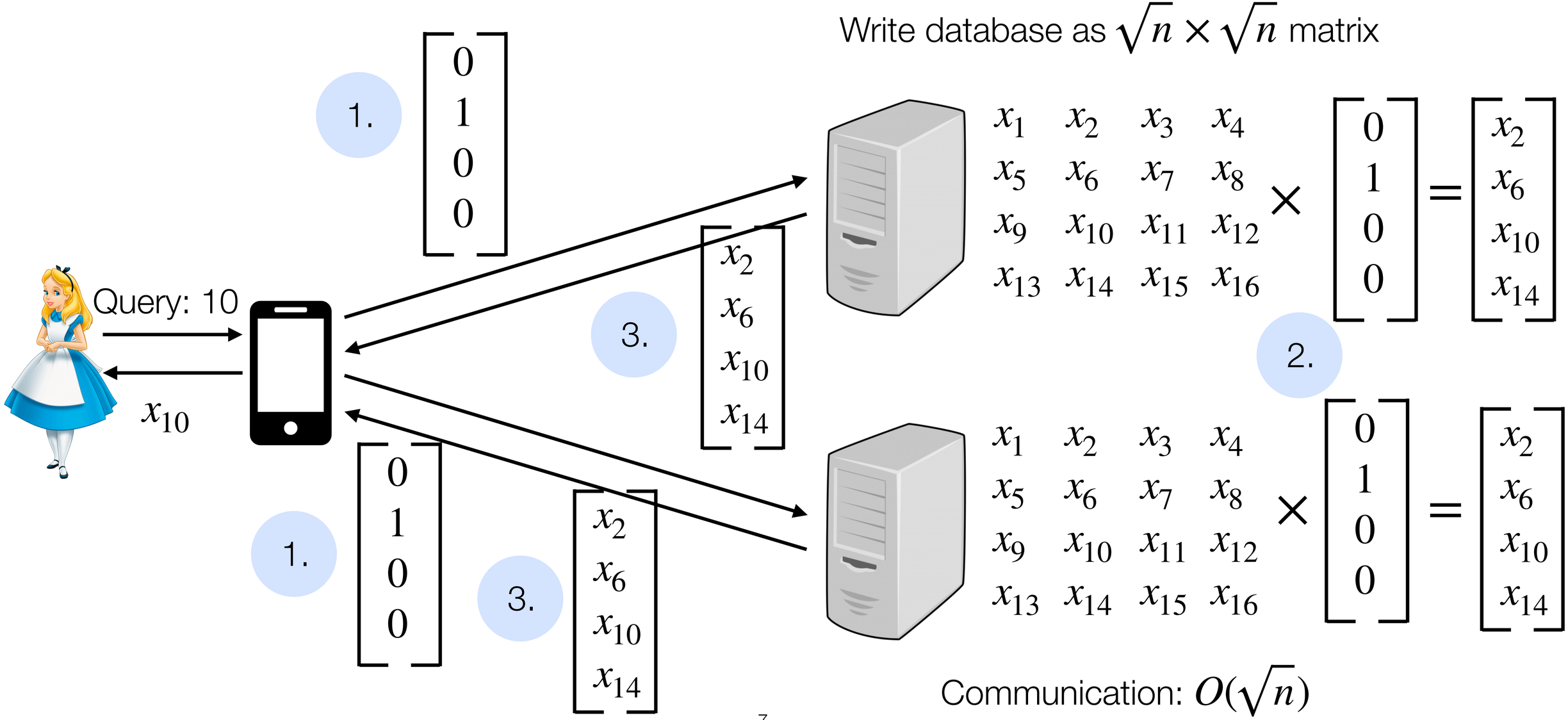
Recap: two-server PIR scheme



Recap: two-server PIR scheme



Recap: two-server PIR scheme

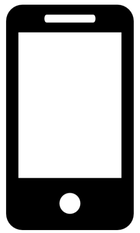


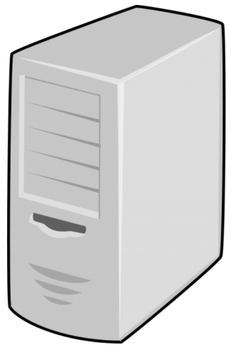
Recap: additively homomorphic encryption

Encryption scheme that supports:

- Adding ciphertexts: $\mathsf{Enc}_k(x) + \mathsf{Enc}_k(y) = \mathsf{Enc}_k(x + y)$
- Multiplication by a constant (by extension): $c \cdot \mathsf{Enc}_k(x) = \mathsf{Enc}_k(c \cdot x)$

Recap: single-server PIR scheme

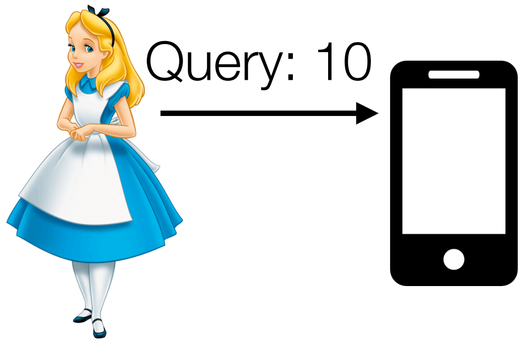


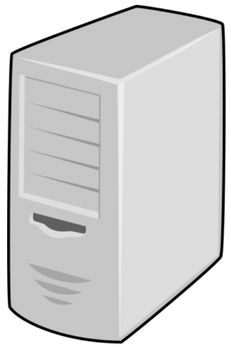


Write database as $\sqrt{n} \times \sqrt{n}$ matrix

$$\begin{matrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{matrix}$$

Recap: single-server PIR scheme

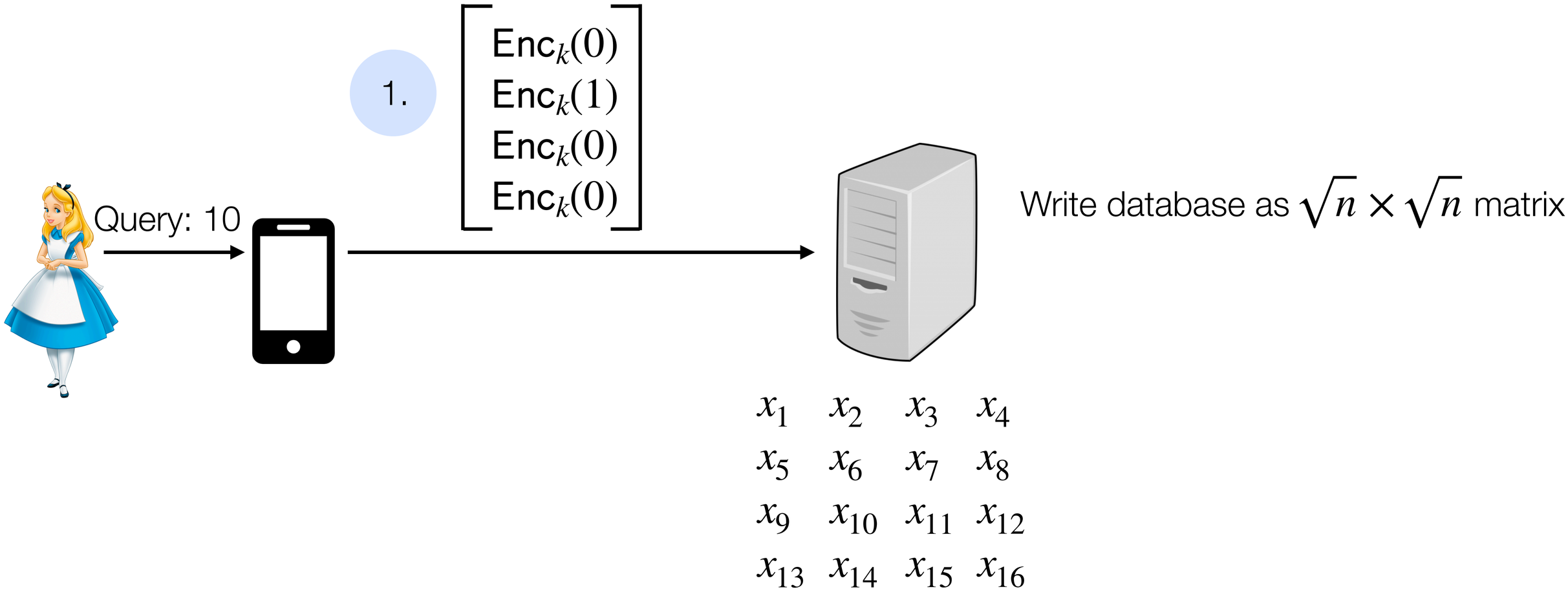




Write database as $\sqrt{n} \times \sqrt{n}$ matrix

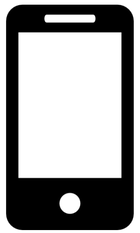
$$\begin{matrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{matrix}$$

Recap: single-server PIR scheme



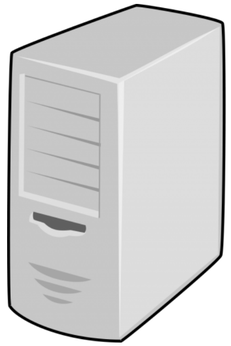
Recap: single-server PIR scheme

Query: 10



1.

$$\begin{bmatrix} \mathsf{Enc}_k(0) \\ \mathsf{Enc}_k(1) \\ \mathsf{Enc}_k(0) \\ \mathsf{Enc}_k(0) \end{bmatrix}$$



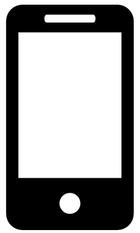
Write database as $\sqrt{n} \times \sqrt{n}$ matrix

2.

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{bmatrix} \times \begin{bmatrix} \mathsf{Enc}_k(0) \\ \mathsf{Enc}_k(1) \\ \mathsf{Enc}_k(0) \\ \mathsf{Enc}_k(0) \end{bmatrix} = \begin{bmatrix} \mathsf{Enc}_k(x_2) \\ \mathsf{Enc}_k(x_6) \\ \mathsf{Enc}_k(x_{10}) \\ \mathsf{Enc}_k(x_{14}) \end{bmatrix}$$

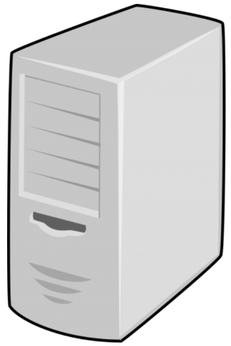
Recap: single-server PIR scheme

Query: 10



1.

$$\begin{bmatrix} \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(1) \\ \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(0) \end{bmatrix}$$



Write database as $\sqrt{n} \times \sqrt{n}$ matrix

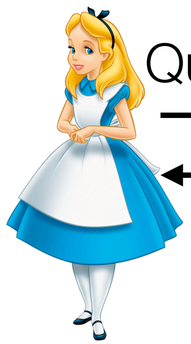
2.

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{bmatrix} \times \begin{bmatrix} \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(1) \\ \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(0) \end{bmatrix} = \begin{bmatrix} \mathrm{Enc}_k(x_2) \\ \mathrm{Enc}_k(x_6) \\ \mathrm{Enc}_k(x_{10}) \\ \mathrm{Enc}_k(x_{14}) \end{bmatrix}$$

3.

$$\begin{bmatrix} \mathrm{Enc}_k(x_2) \\ \mathrm{Enc}_k(x_6) \\ \mathrm{Enc}_k(x_{10}) \\ \mathrm{Enc}_k(x_{14}) \end{bmatrix}$$

Recap: single-server PIR scheme

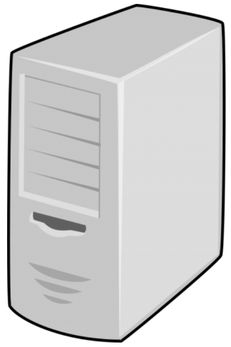


Query: 10

x_{10}

1.

$$\begin{bmatrix} \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(1) \\ \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(0) \end{bmatrix}$$



Write database as $\sqrt{n} \times \sqrt{n}$ matrix

2.

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{bmatrix} \times \begin{bmatrix} \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(1) \\ \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(0) \end{bmatrix} = \begin{bmatrix} \mathrm{Enc}_k(x_2) \\ \mathrm{Enc}_k(x_6) \\ \mathrm{Enc}_k(x_{10}) \\ \mathrm{Enc}_k(x_{14}) \end{bmatrix}$$

3.

$$\begin{bmatrix} \mathrm{Enc}_k(x_2) \\ \mathrm{Enc}_k(x_6) \\ \mathrm{Enc}_k(x_{10}) \\ \mathrm{Enc}_k(x_{14}) \end{bmatrix}$$

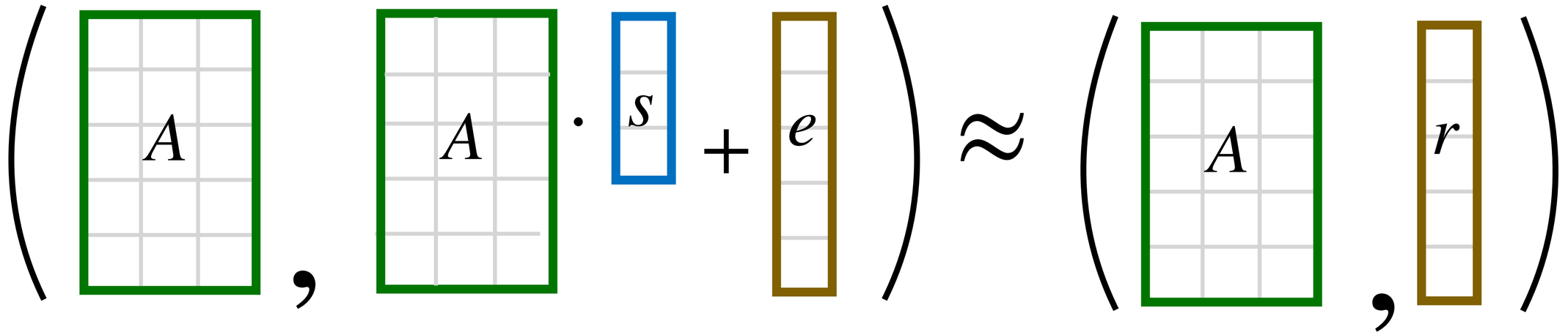
Outline

1. **SimplePIR**
2. Tiptoe
3. Student presentation: Compass

Learning with errors assumption

[Regev09]

Parameterized by n, m, q and error distribution χ



where $A \in \mathbb{Z}_q^{m \times n}$, $s \in \mathbb{Z}_q^n$, $e \in \chi^m$, and $r \in \mathbb{Z}_q^m$ is a random vector

Choosing error distribution: balance between utility and security

- χ is uniform distribution: secure but not useful
- χ always outputs 0: useful but not secure
- Standard setting: χ is discrete Gaussian distribution restricted to some output range

Regev encryption

[Regev09]

Parameterized by $n, m,$ ciphertext modulus q, plaintext modulus p, and error distribution χ

Encrypt message $m \in \mathbb{Z}_p$ with secret key s

$$\left(a \;,\; a^T \cdot s + e + m \lfloor q/p \rfloor \right) = (a, c)$$

LWE assumption

$$\left(A \;,\; A \cdot s + e \right) \approx \left(A \;\; r \right)$$

Regev encryption

[Regev09]

Parameterized by n, m, q, plaintext modulus p, and error distribution χ

Encrypt message $m \in \mathbb{Z}_p$ with secret key s

$$\left(a \;,\; a^T \cdot s + e + m \lfloor q/p \rfloor \right) = (a, c)$$

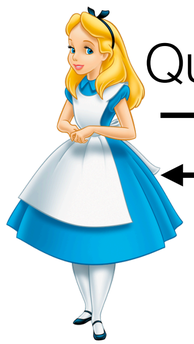
Decrypt ciphertext (a, c) with secret key s

Round $c - a^T \cdot s$ to the nearest multiple of $\lfloor q/p \rfloor$

Succeeds as long as error is “small”, i.e., $< \frac{1}{2} \cdot \lfloor q/p \rfloor$

Feature: additively homomorphic — $\mathsf{Enc}_k(m_1) + \mathsf{Enc}_k(m_2) = \mathsf{Enc}_k(m_1 + m_2)$

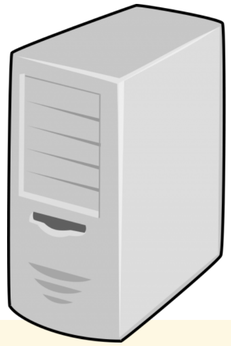
Recall: single-server PIR scheme



Query: 10

x_{10}

1. $\begin{bmatrix} \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(1) \\ \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(0) \end{bmatrix}$



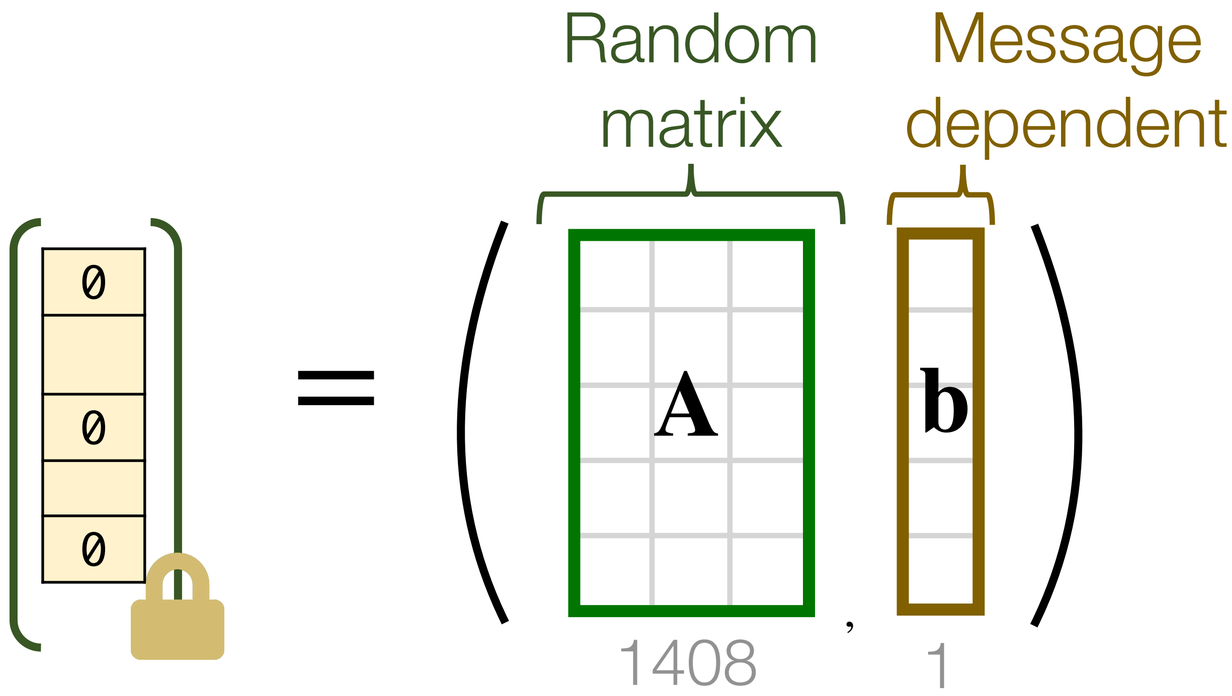
Write database as $\sqrt{n} \times \sqrt{n}$ matrix

2.

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 \\ x_9 & x_{10} & x_{11} & x_{12} \\ x_{13} & x_{14} & x_{15} & x_{16} \end{bmatrix} \times \begin{bmatrix} \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(1) \\ \mathrm{Enc}_k(0) \\ \mathrm{Enc}_k(0) \end{bmatrix} = \begin{bmatrix} \mathrm{Enc}_k(x_2) \\ \mathrm{Enc}_k(x_6) \\ \mathrm{Enc}_k(x_{10}) \\ \mathrm{Enc}_k(x_{14}) \end{bmatrix}$$

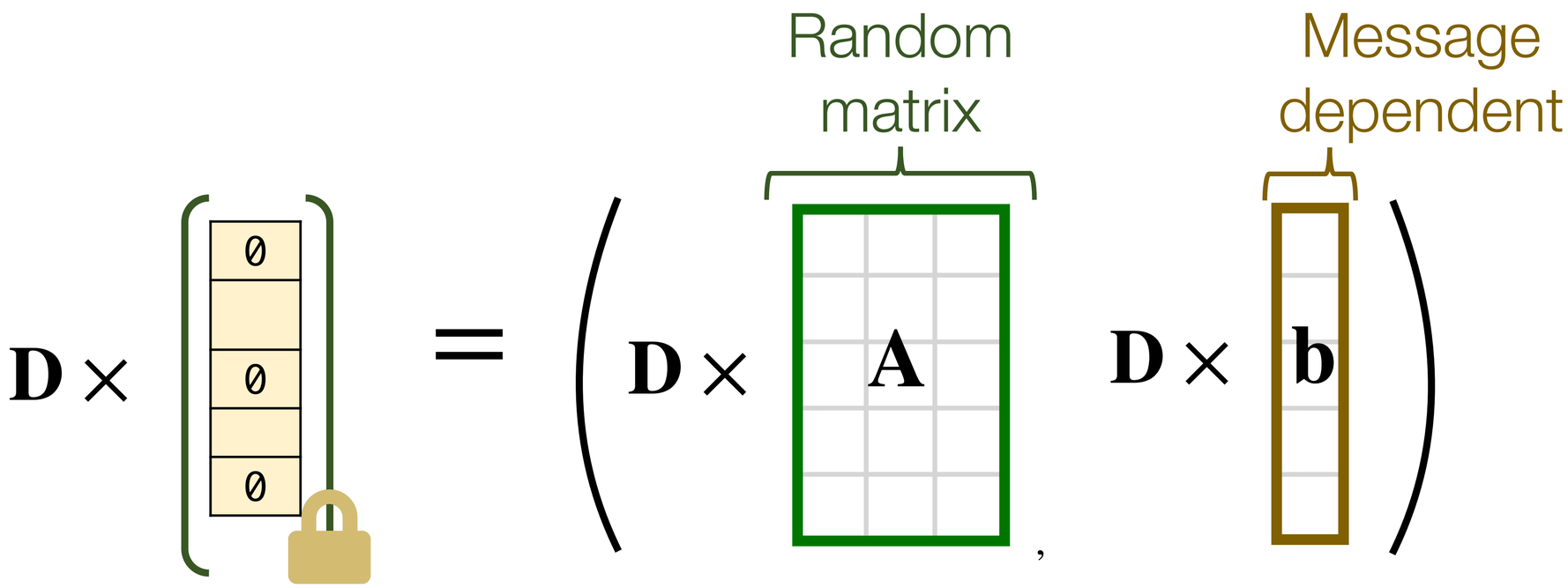
3. $\begin{bmatrix} \mathrm{Enc}_k(x_2) \\ \mathrm{Enc}_k(x_6) \\ \mathrm{Enc}_k(x_{10}) \\ \mathrm{Enc}_k(x_{14}) \end{bmatrix}$

Regev encryption



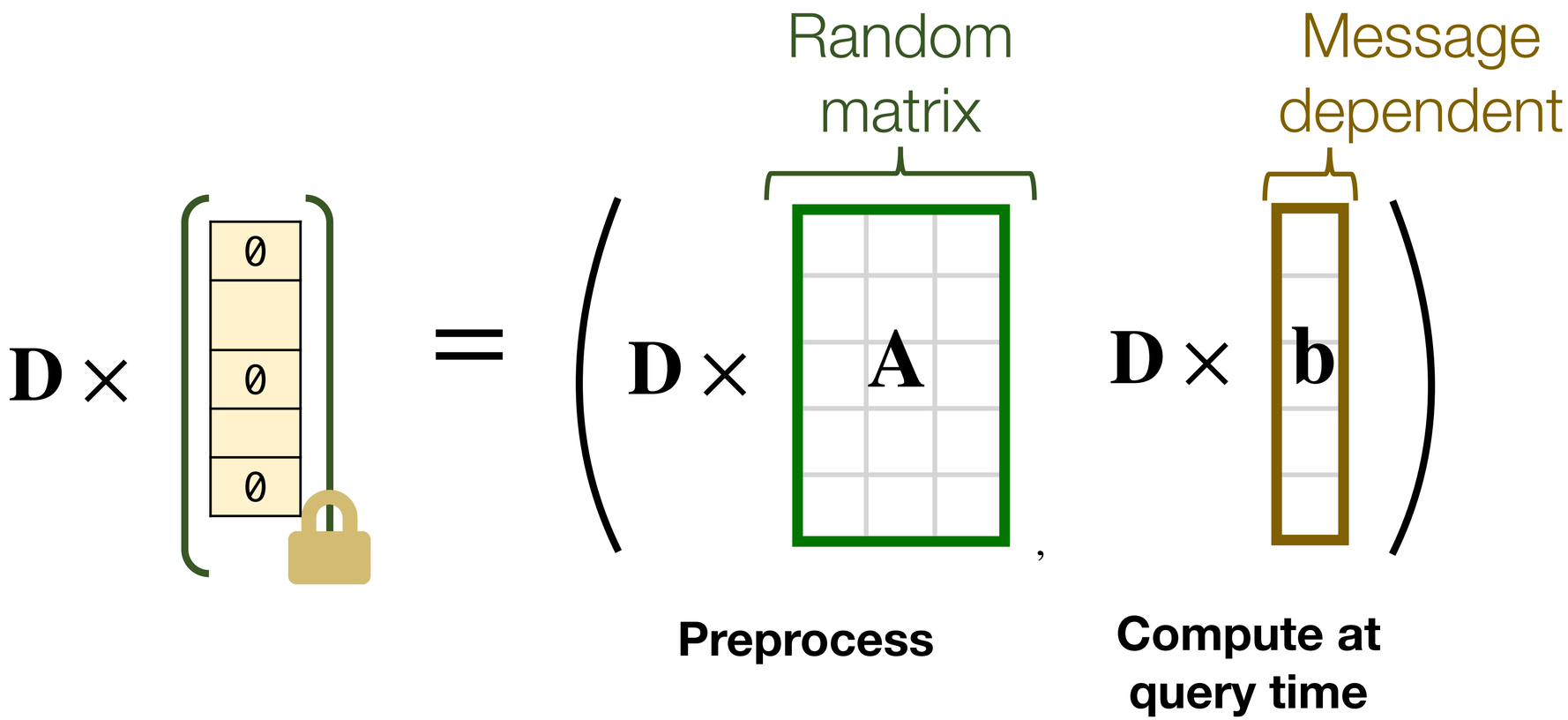
Message is a vector of values

Processing a PIR query



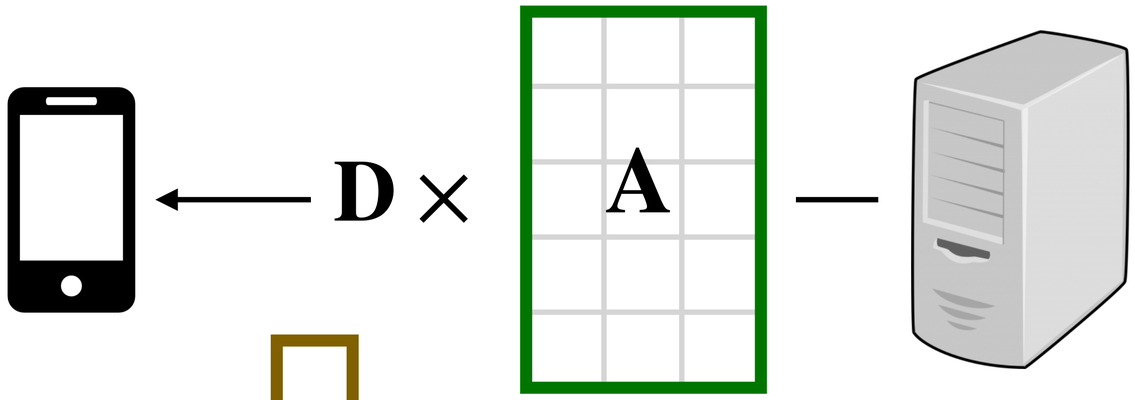
SimplePIR

[Henzinger, Hong, Corrigan-Gibbs, Meiklejohn, Vakuntanathan]



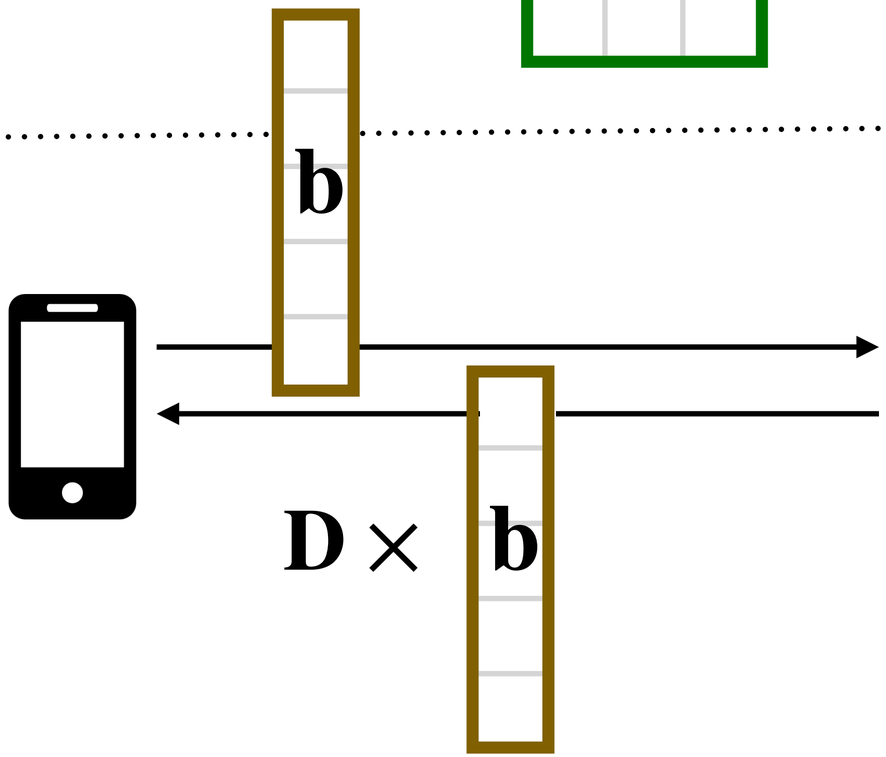
SimplePIR

Preprocessing:
Download hint



Database **D**

Online:
make query



Database **D**

Can re-use hint to
make many queries

High-throughput queries

Outline

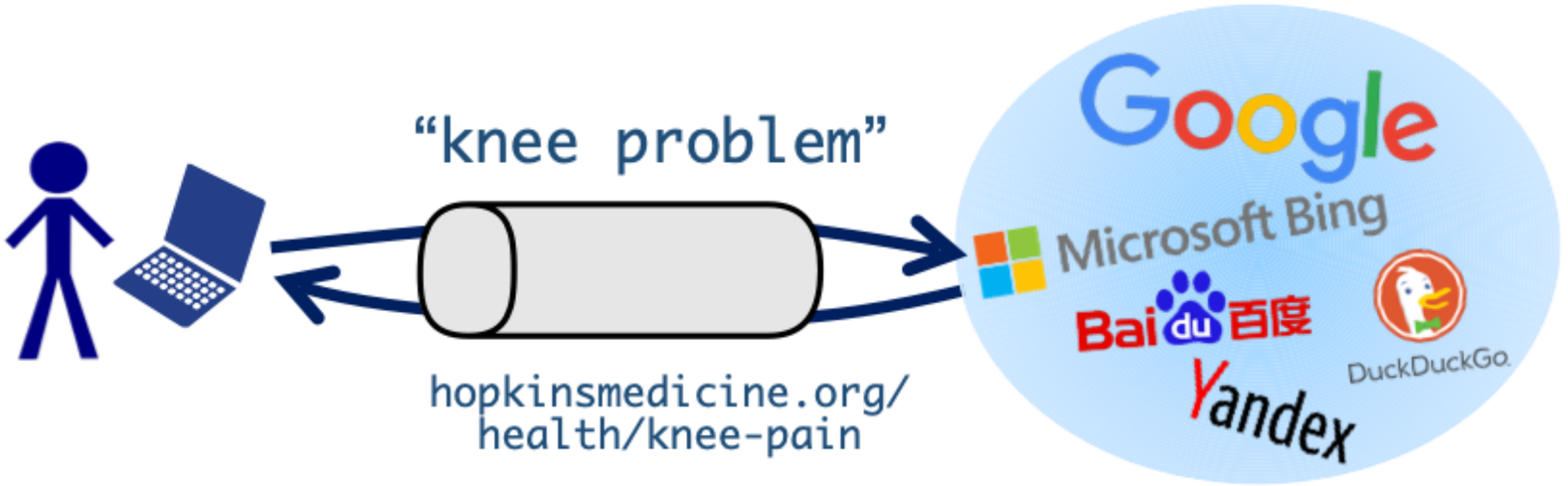
1. SimplePIR
2. **Tiptoe**
3. Student presentation: Compass

Web-search queries reveal your sensitive data

Health	`ballet knee problem`
Finances	`job opportunities in west palm beach`
Religion	`african american churches in norfolk va`
Citizenship	`application forms us citizen`
Relationships	`domestic violence laws in new york city`

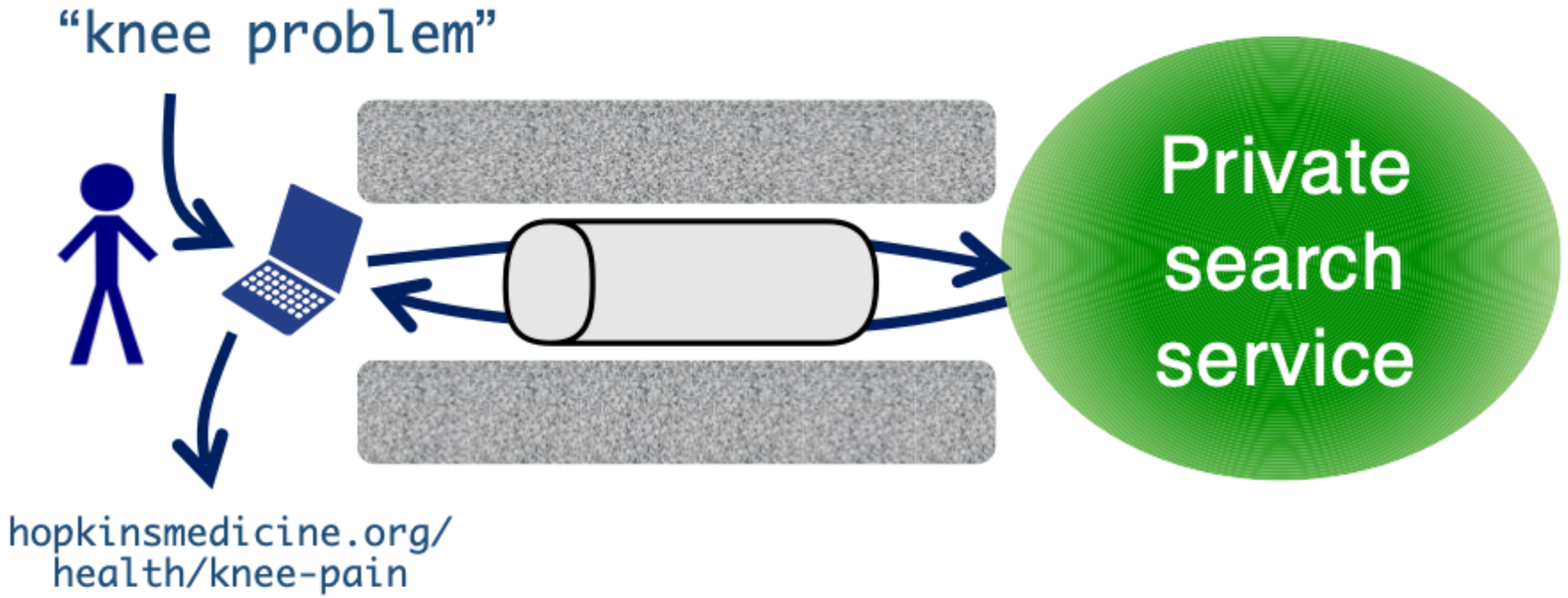
https://trec.nist.gov/data/million.query07.html

Today: Send query to search engine

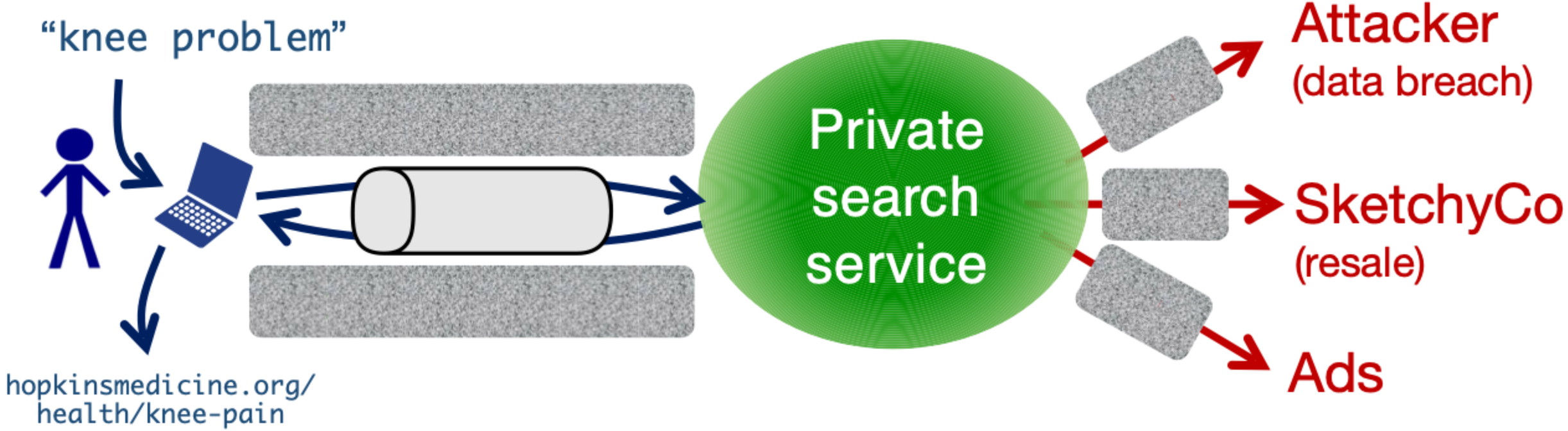


Today: Send query to search engine

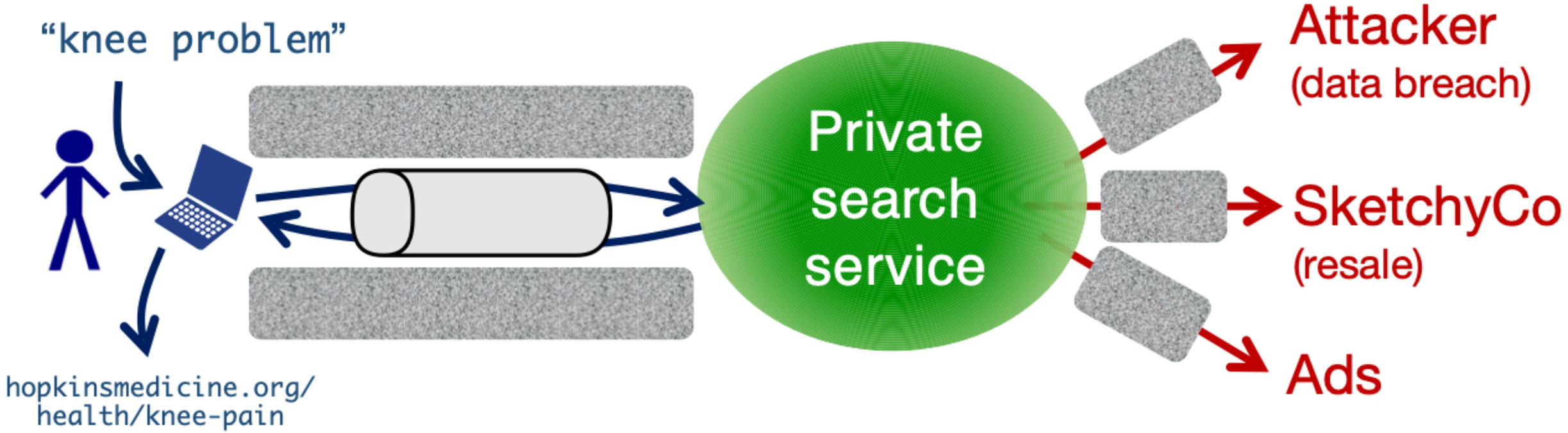
Goal: Search without revealing the query



Goal: Search without revealing the query



Goal: Search without revealing the query



Limitations:
- does not hide subsequent HTTP(S) requests
- does not hide *when* the client makes searches
- does not guarantee correct search results

Challenges

1. Search algorithms are not “crypto friendly”

- Crypto: Boolean circuits
- Search: Random access to terabytes of data ☹

2. Computing on encrypted data is expensive

- Orders of magnitude worse than computation on plaintext

3. The web is large

Tiptoe: A private web-search engine

[Henzinger, Dauterman, Corrigan-Gibbs, Zeldovich]

+ Server learns no information about client's query
  unlike DuckDuckGo, Google over Tor, …

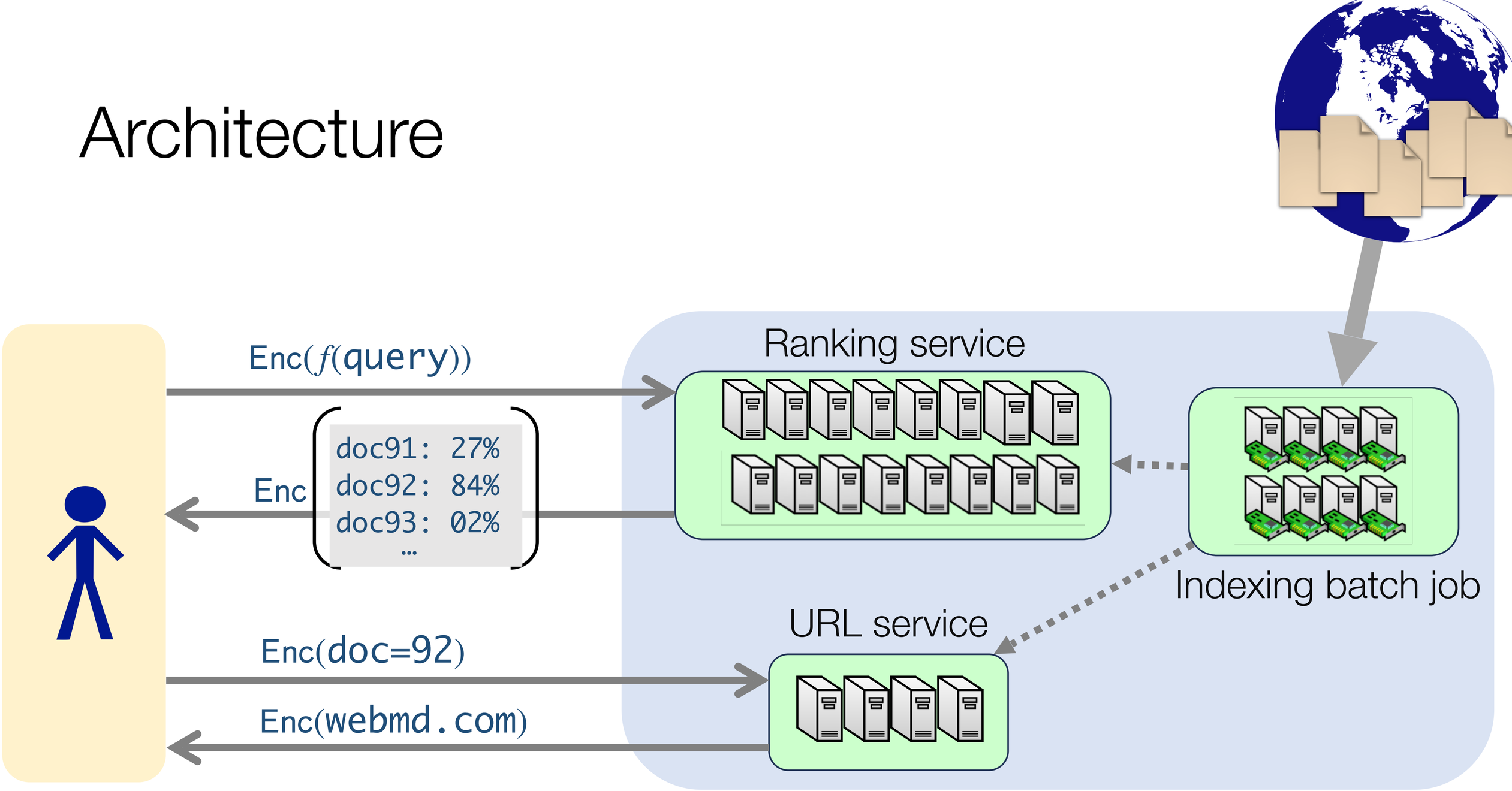
+ Searches 364 million web pages with 2.7 seconds of latency
  with 145 core-s of compute, 57 MiB comm., and 300 MiB of client storage

+ Supports image search; extensible to code, audio, video

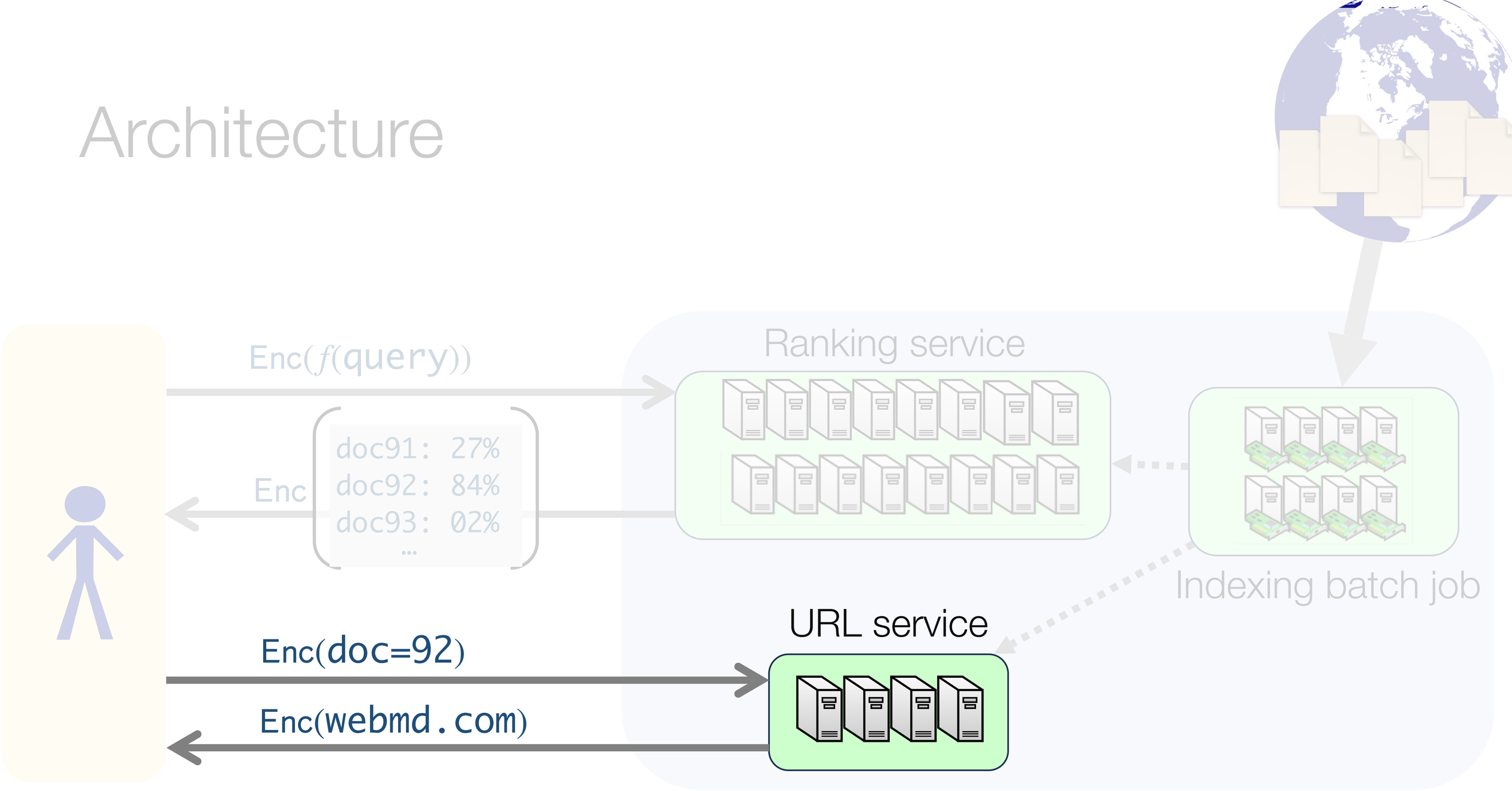
– Search quality cannot compete with non-private search
  evaluated with standard information-retrieval benchmark (MS-MARCO)

Slides made with co-authors

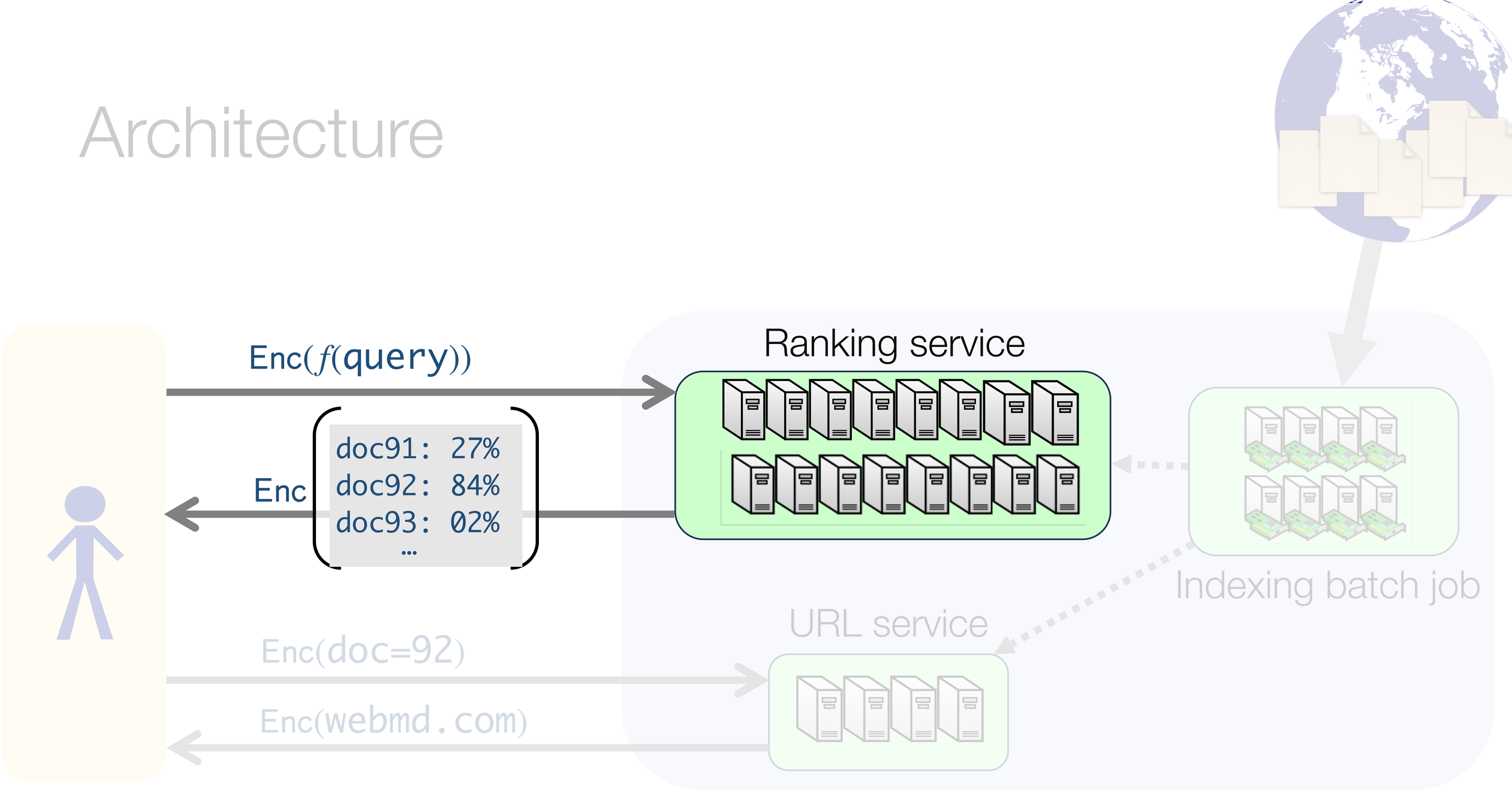
Architecture



Architecture



Architecture



Tiptoe: Design ideas

1. Reduce private text search to private nearest-neighbor search

   Key tool: Semantic embeddings [Osgood57, …]

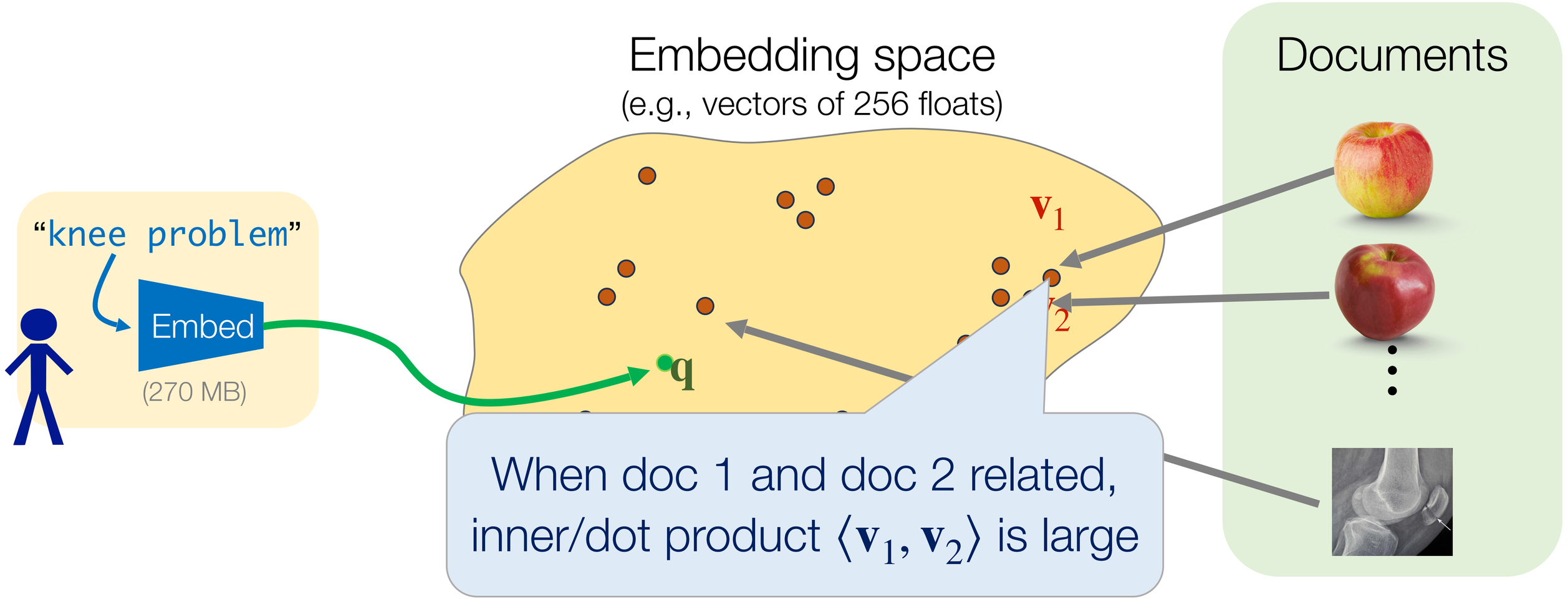
2. Reduce private nearest-neighbor search to private matrix multiplication

   Key tool: Clustering to reduce communication to $\sqrt{N}$ (from $\sqrt{N}$)

3. Use a (new) fast scheme for private matrix multiplication

   Key tool: Fast linearly homomorphic encryption [HHCMV'22]

1. Reduce private text search to private nearest-neighbor search using semantic embeddings [Osgood57, …, MCCD13, BCLT19…]



Why embeddings are useful

- Reduce a messy problem to a clean one (text search) (nearest-neighbor search)
- Can improve/optimize embeddings independently of crypto
- There are open embeddings for text, code, images, videos, audio…
- The embedding has no effect on privacy

Tiptoe: Design ideas

1. Reduce private text search to private nearest-neighbor search

   Key tool: Semantic embeddings [Osgood57, …]

2. Reduce private nearest-neighbor search to private matrix multiplication

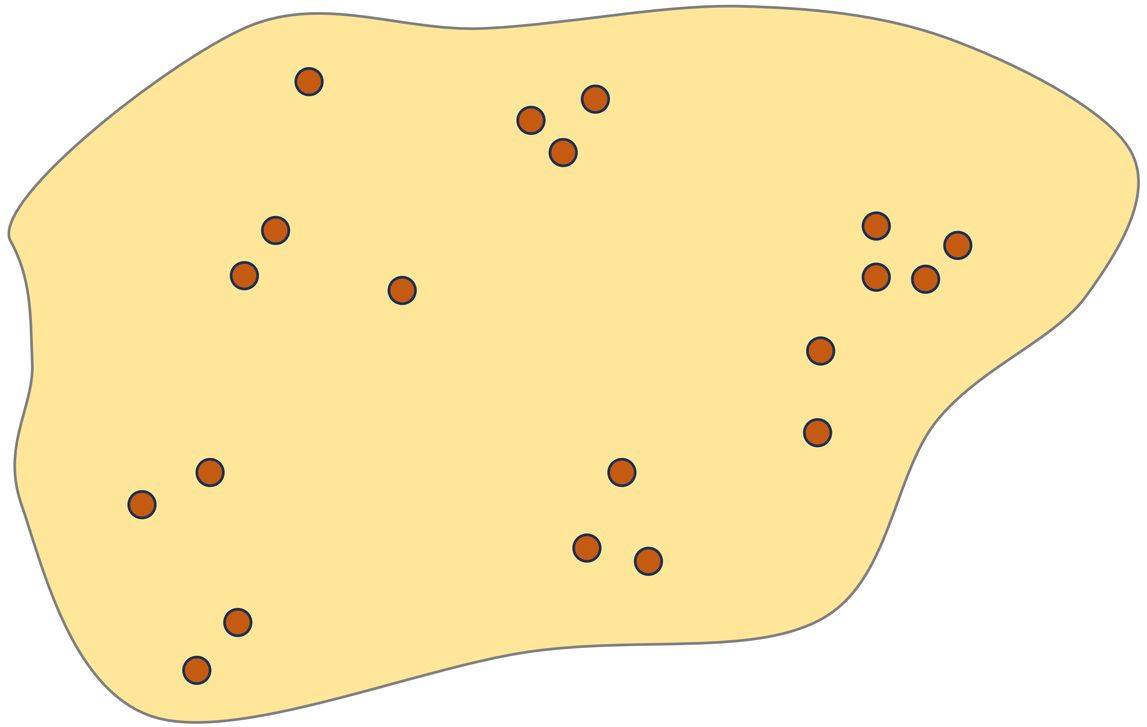
   Key tool: Clustering to reduce communication to $\sqrt{N}$ (from $\sqrt{N}$)

3. Use a (new) fast scheme for private matrix multiplication

   Key tool: Fast linearly homomorphic encryption [HHCMV'22]

2. Reduce nearest-neighbor search to private matrix multiplication

Preprocessing batch job

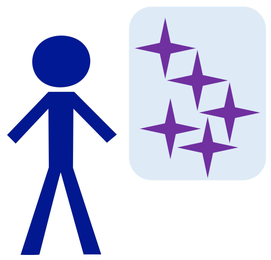


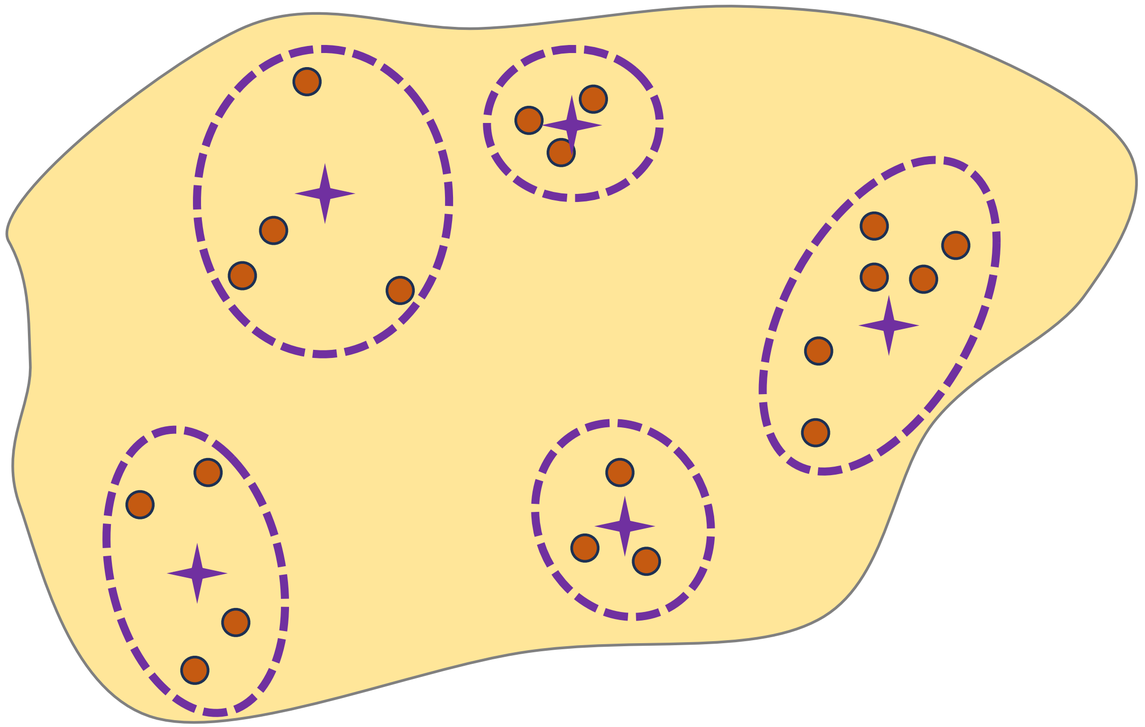
Embedding space

2. Reduce nearest-neighbor search to private matrix multiplication

Preprocessing batch job

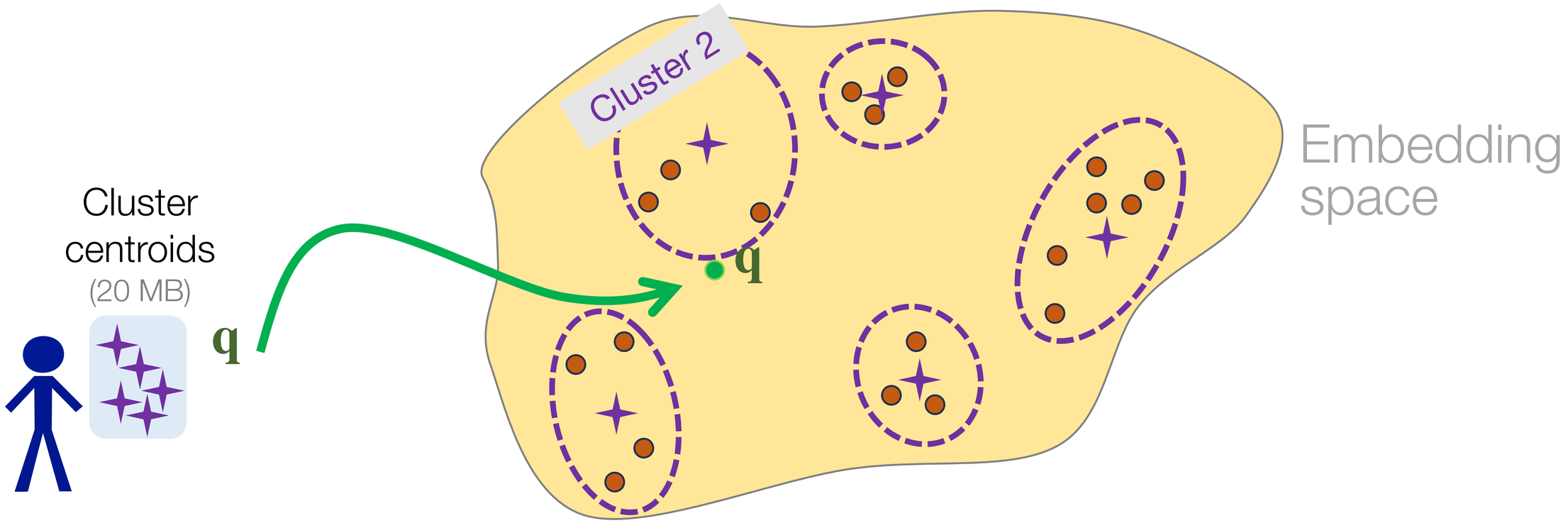
Cluster centroids
(20 MB)





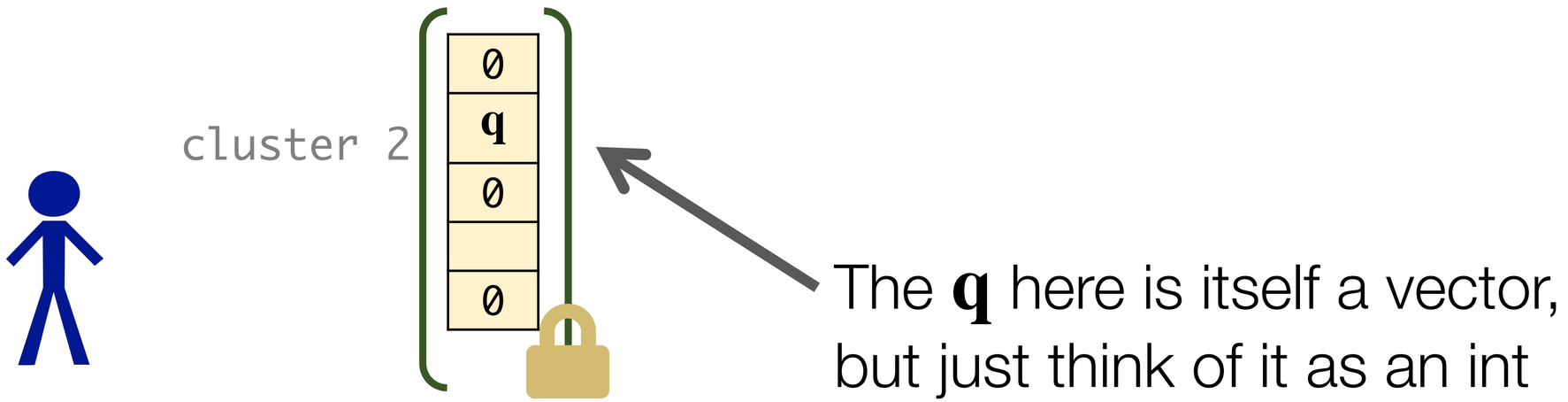
Embedding space

2. Reduce nearest-neighbor search to private matrix multiplication
At query time: Identify “best” cluster



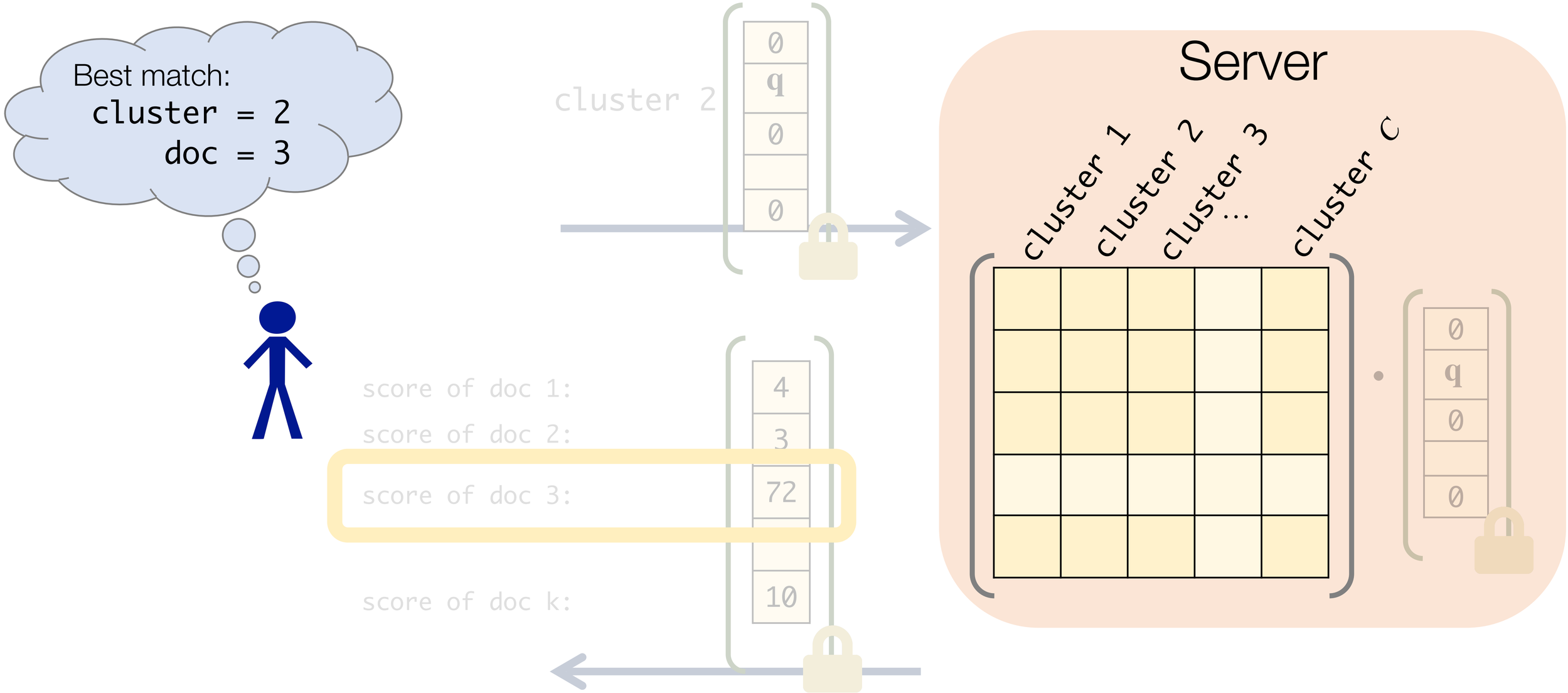
2. Reduce nearest-neighbor search to private matrix multiplication

At query time: Fetch scores for docs in “best” cluster



2. Reduce nearest-neighbor search to private matrix multiplication

At query time: Fetch scores for docs in “best” cluster



2. Reduce nearest-neighbor search to private matrix multiplication
At query time: Fetch scores for docs in “best” cluster

With C clusters and length-L embeddings,

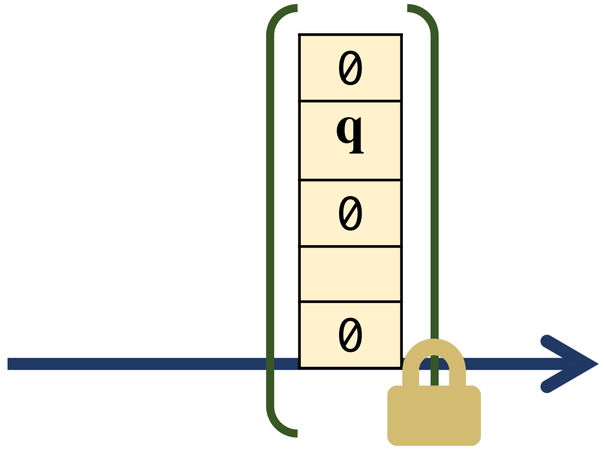
Upload: $O(C \cdot L)$ bytes

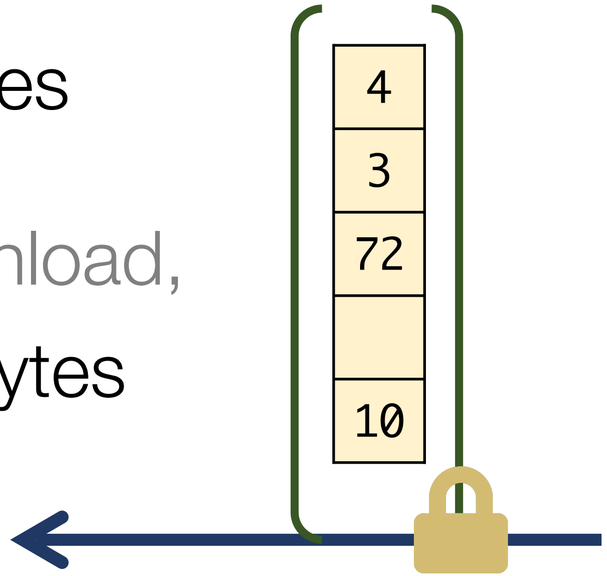
With N total documents,

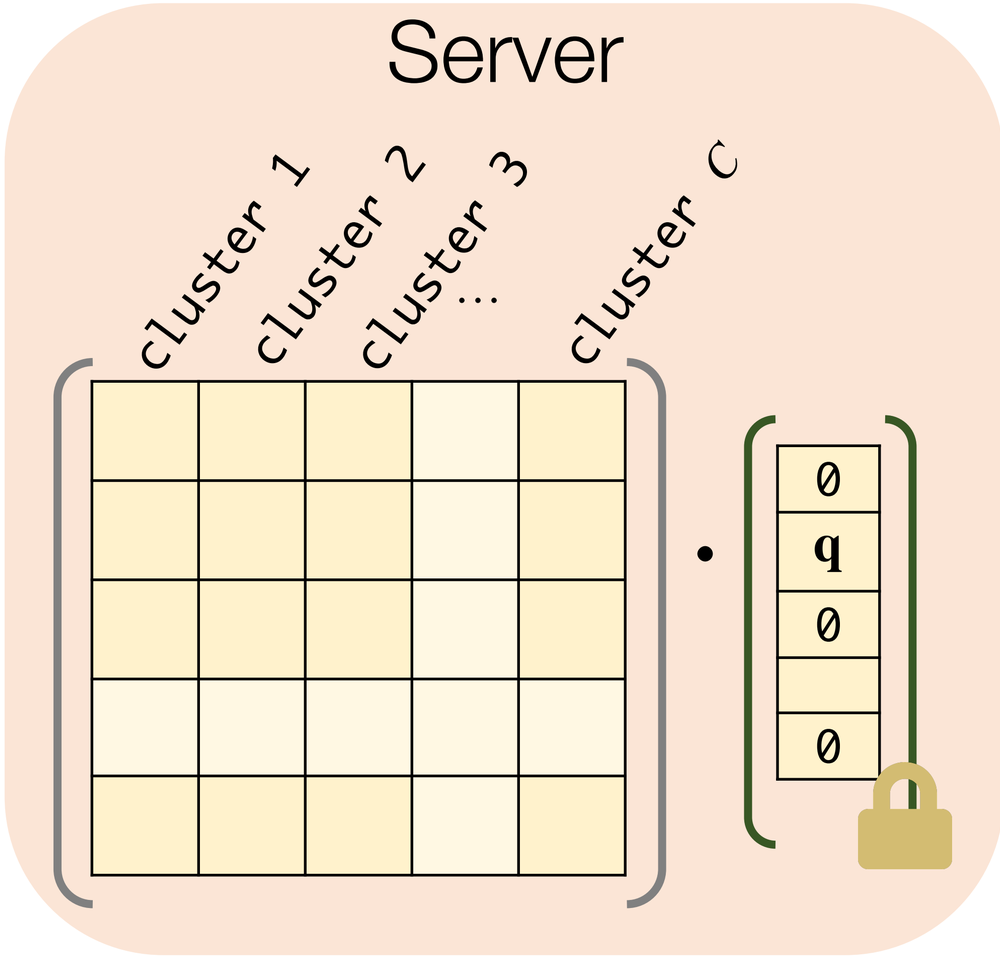
Download: $O(N/C)$ bytes

Balancing upload & download,

Total comm.: $O(\sqrt{NL}$ bytes







Tiptoe: Design ideas

1. Reduce private text search to private nearest-neighbor search

   Key tool: Semantic embeddings [Osgood57, …]

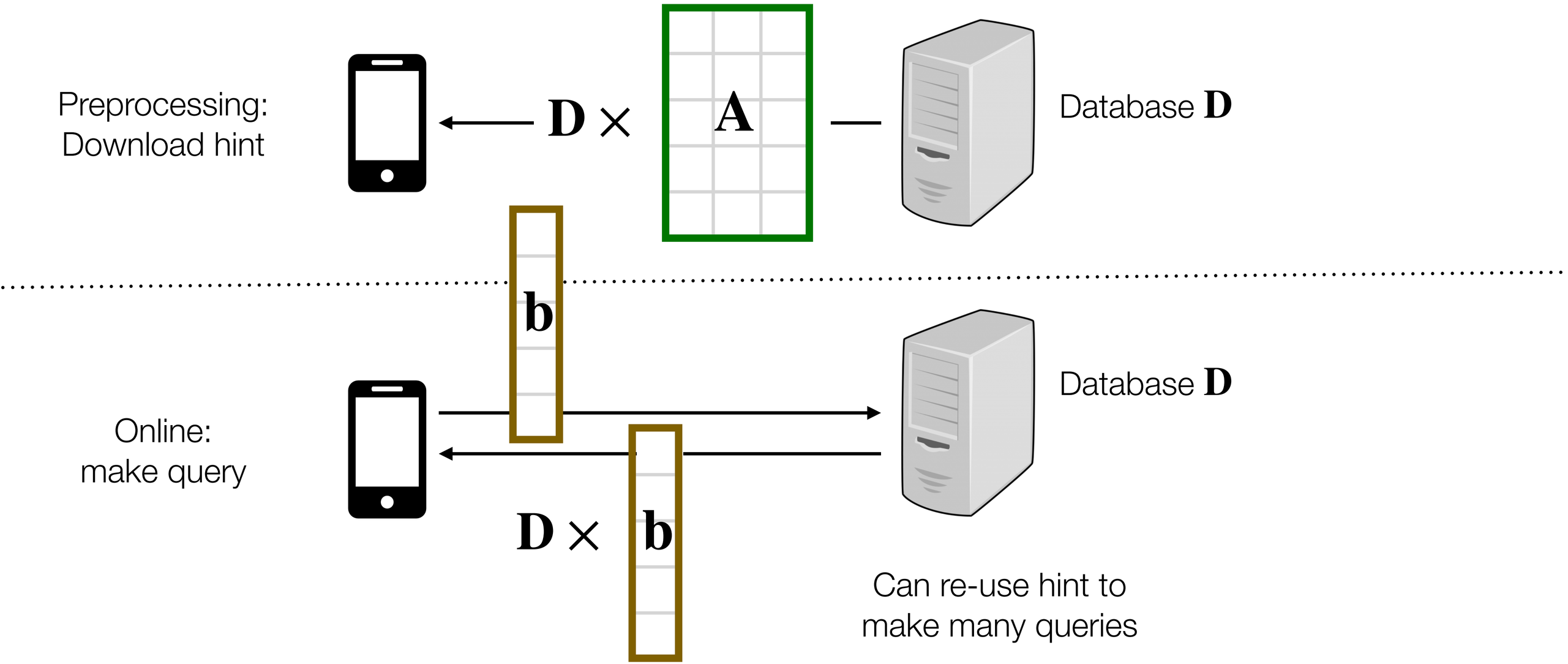
2. Reduce private nearest-neighbor search to private matrix multiplication

   Key tool: Clustering to reduce communication to $\sqrt{N}$ (from $\sqrt{N}$)

3. Use a (new) fast scheme for private matrix multiplication

   Key tool: Fast linearly homomorphic encryption [HHCMV'22]

SimplePIR


Preprocessing:
Download hint
D ×
A
Database D
b
Online:
make query
Database D
D ×
b
Can re-use hint to
make many queries

Avoid storing the hint at the client

Idea: outsource the hint to the server

Most of the decryption algorithm just requires linear operations

- Can perform under encryption at the server

Tiptoe: Design ideas

1. Reduce private text search to private nearest-neighbor search

   Key tool: Semantic embeddings [Osgood57, …]

2. Reduce private nearest-neighbor search to private matrix multiplication

   Key tool: Clustering to reduce communication to $\sqrt{N}$ (from $\sqrt{N}$)

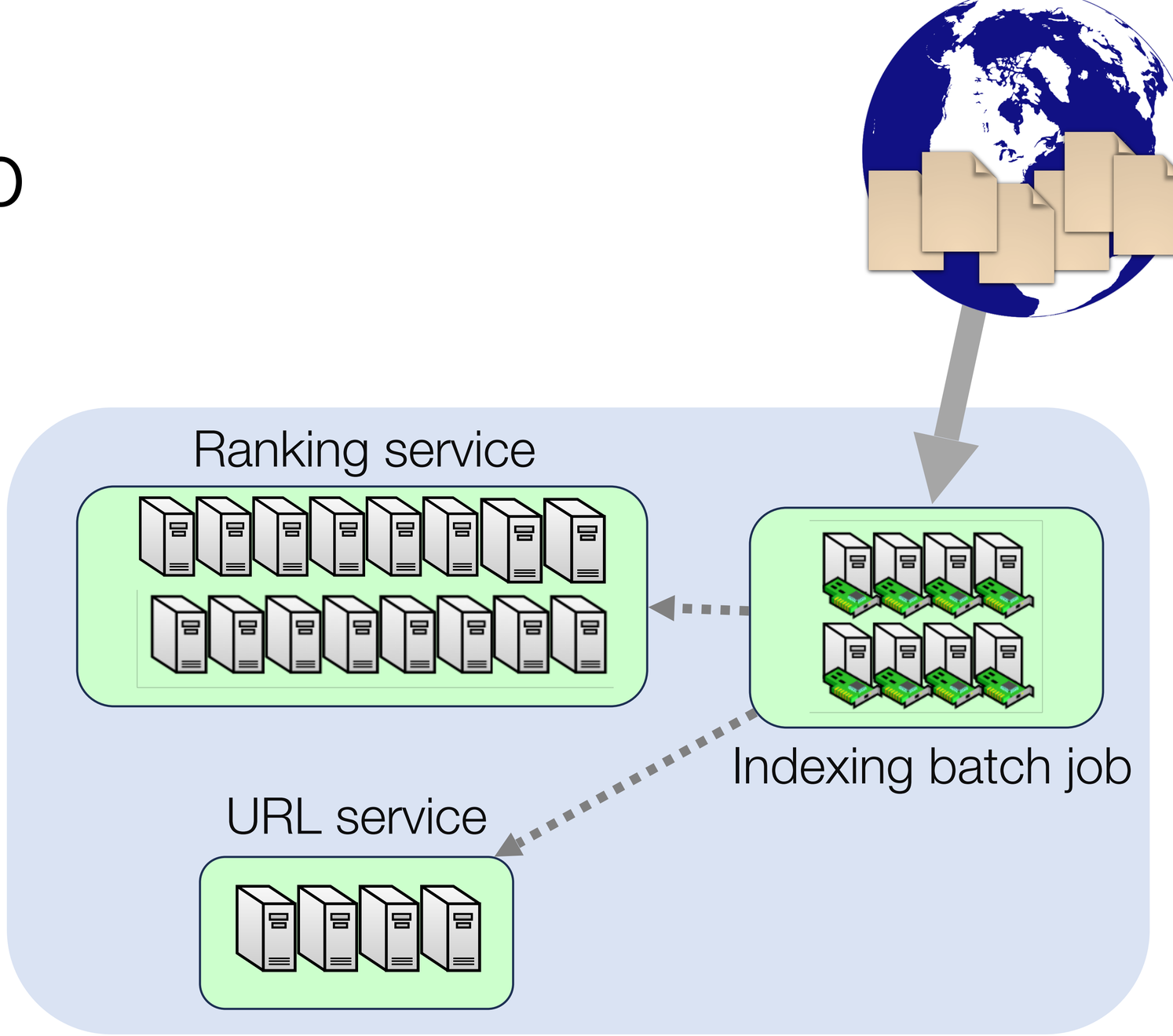
3. Use a (new) fast scheme for private matrix multiplication

   Key tool: Fast <u>linearly</u> homomorphic encryption [HHCMV'22]

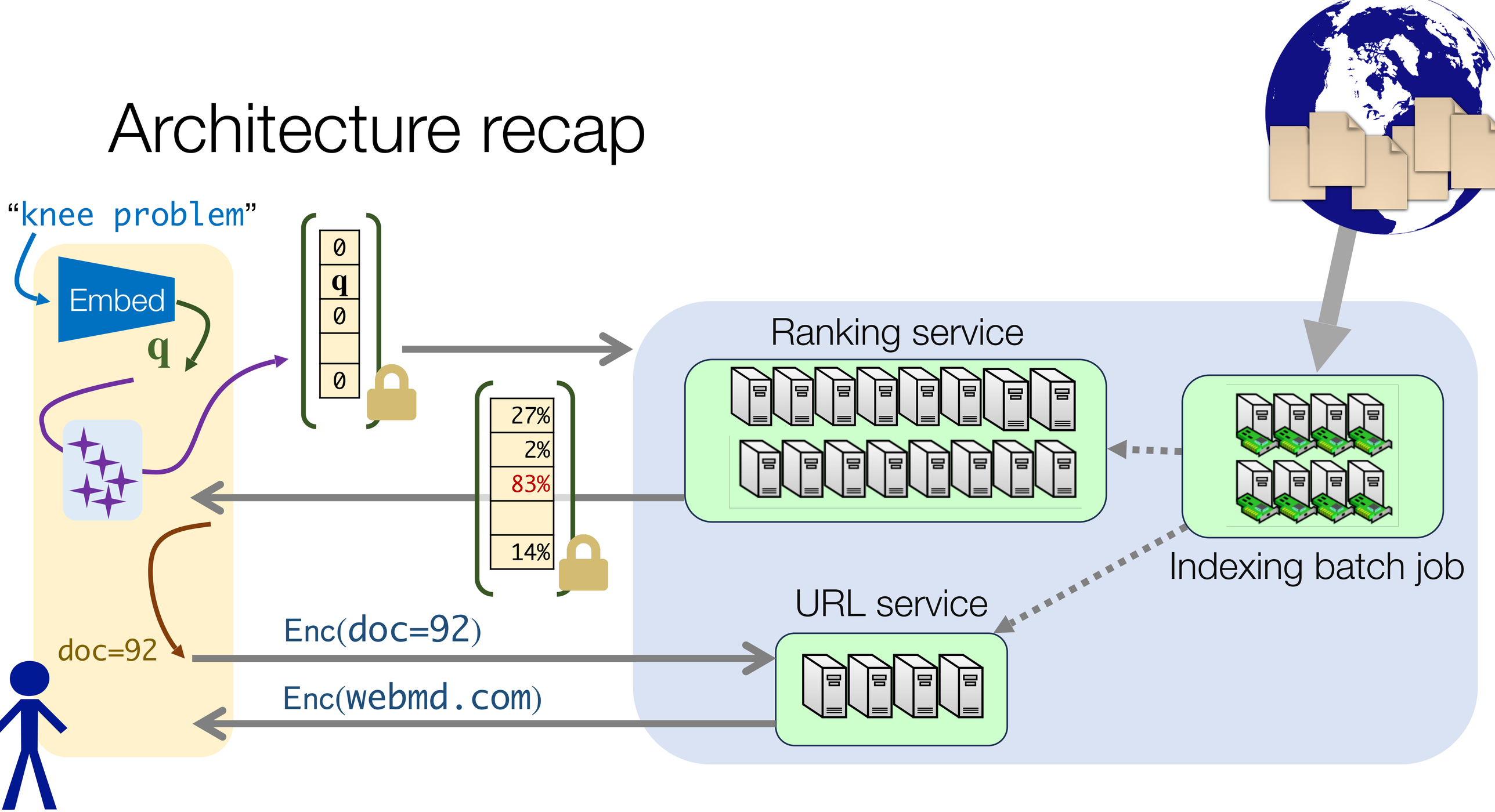
Architecture recap

Batch job

- Embeds documents
- Clusters embeddings
- Computes the encryption-scheme “hint” for both services



Architecture recap



Strengths and weaknesses of Tiptoe from the evaluation

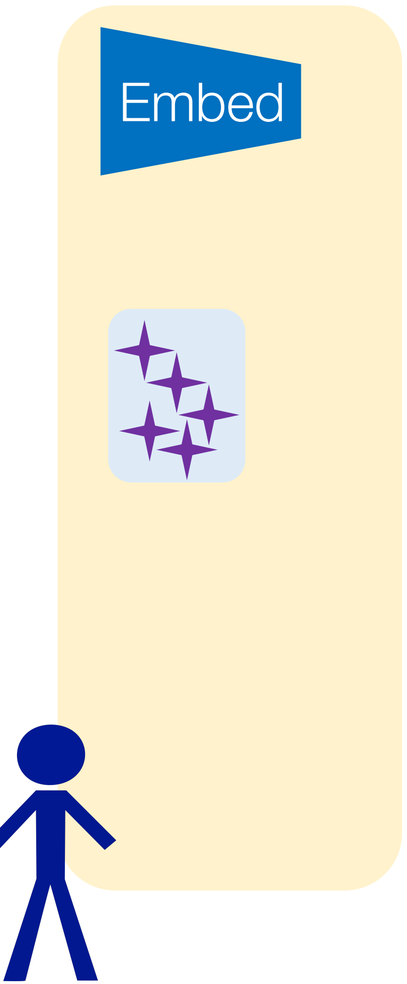
Strengths

- Evaluates on real webpages (364M from common crawl) and images (400M from LAION-400M)
- Similar MRR@100 score to Coeus, but much better performance
- Costs orders of magnitude lower than client-side index or Coeus
- Clustering makes it possible for communication to scale sub linearly with the number of documents with a comparatively small drop in search quality
- Infrastructure is compatible with text and images

Weaknesses

- Search quality worse than that of state-of-the-art plaintext search algorithms
- Querying more clusters could help improve search quality, but would substantially increase costs
- Evaluation over hundreds of millions of documents, but web is much larger (shows analytical scaling to tens of billions of documents)
- Compute costs scale linearly with number of documents
- Requires client to store state (embedding function + cluster centroids)

Architecture recap



Client download

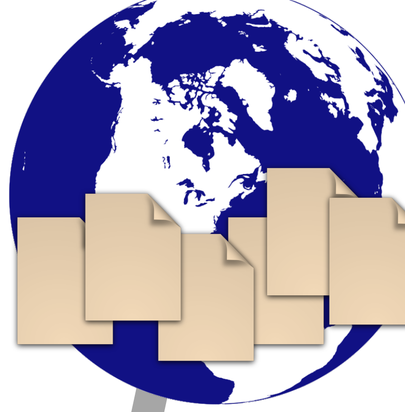
- Embedding function 270 MB
- Cluster centroids 20 MB

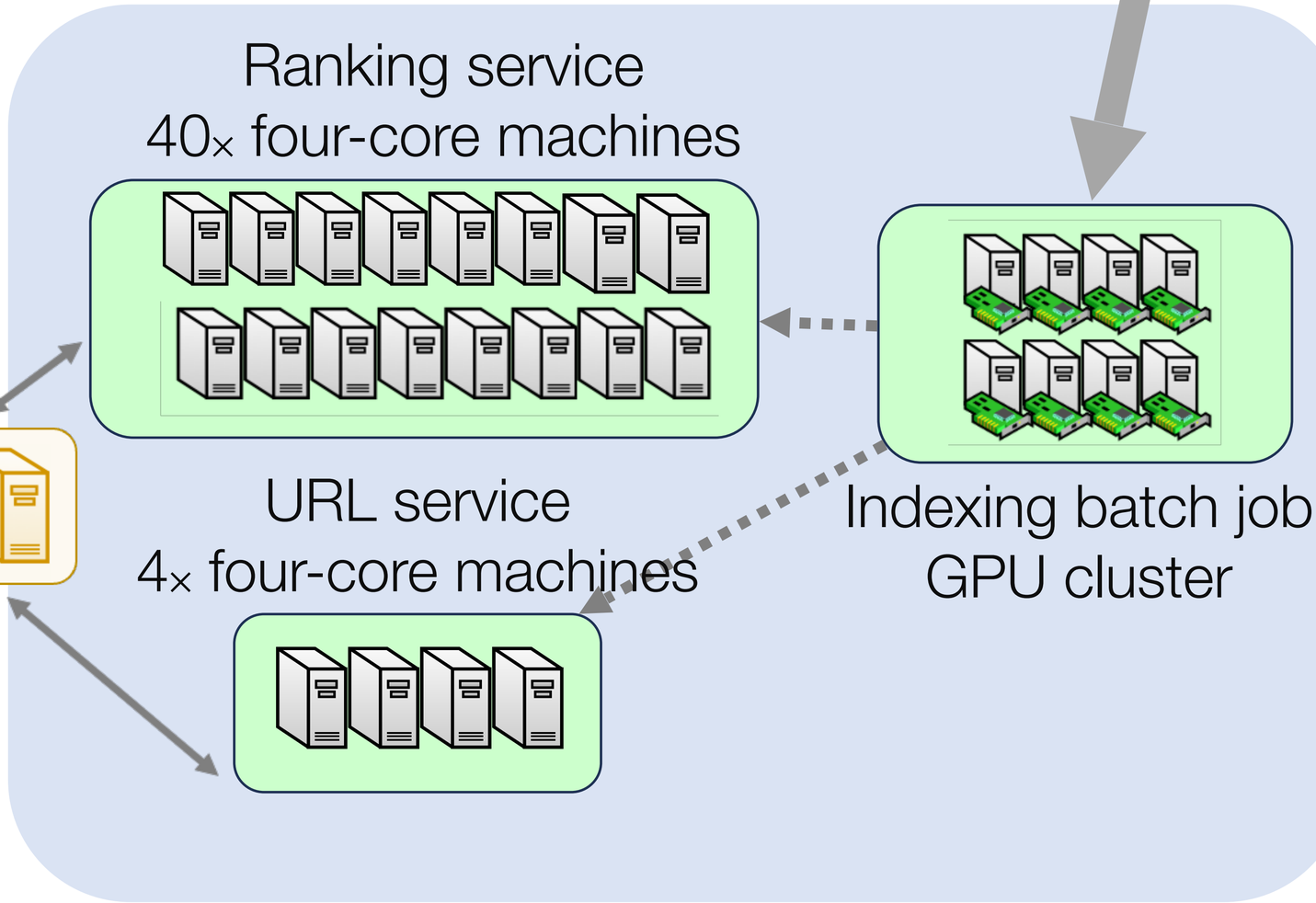
Implementation

6,500 lines of code (Go and Python)
+ our external private information retrieval library (SimplePIR, 2300 lines)

Search quality:	MS Marco doc-rank “dev” dataset, *3.2M pages*
Text search:	Cleaned common crawl, *364M pages* `msmarco-distilbert-base-tas-b` embedding
Image search:	LAION-400M data set, *400M images* `CLIP` embedding (prepackaged)

Implementation





100 Mbps
50 ms RTT

Perceived latency:
2.5 seconds

Tiptoe is cheaper than state-of-the-art private search

	Coeus (SOSP 21)	Tiptoe	Gain
Docs searched	5 million	364 million	72×
Client storage	-	0.3 GiB	– ∞×
Server compute (per million docs)	2,580 core-s	0.4 core-s	6,450×
Communication (per million docs)	10 MiB	0.15 MiB	66×
AWS cost (per million docs)	≤ 1 US cent	≤ 0.0008 US cent	1,250×

Tiptoe is cheaper than state-of-the-art private search

	Coeus (SOSP 21)	Tiptoe	Gain
Docs searched	5 million	364 million	72 ×
Client storage	121 MiB	16 MiB	– ∞ ×
Server compute (per million docs)	2,580 core-s	0.4 core-s	6,450 ×
Communication (per million docs)	10 MiB	0.15 MiB	66 ×
AWS cost (per million docs)	≤ 1 US cent	≤ 0.0008 US cent	1,250 ×

Semantic embeddings are 100 × smaller than a bag-of-words representation.

Tiptoe is cheaper than state-of-the-art private search

	Coeus (SOSP 21)	Tiptoe	Gain
Docs searched	5 million	364 million	72 ×
Client storage	0	364 million	– ∞ ×
Server compute (per million docs)	2.580 core-s	0.4 core-s	6,450 ×
Communication (per million docs)	3.9 MiB	60.2 KiB	66 ×
AWS cost (per million docs)	≤ 1 US cent	≤ 0.0008 US cent	1,250 ×

Semantic embeddings are 100 × smaller than a bag-of-words representation.

Tiptoe's high-throughput crypto tools are over 10 × faster than prior work.

Tiptoe is cheaper than state-of-the-art private search

	Coeus (SOSP 21)	Tiptoe	Gain
Docs searched	5 million	364 million	72 ×
Client storage	0	124 MiB	– ∞ ×
Server compute (per million docs)	2,580 core-s	0.4 core-s	6,450 ×
Communication (per million docs)	3.4 MiB	0.05 MiB	66 ×
AWS cost (per million docs)	$0.19	$0.00015	1,250 ×

Semantic embeddings are 100 × smaller than a bag-of-words representation.

Tiptoe's high-throughput crypto tools are over 10 × faster than prior work.

Clustering lets Tiptoe's communication scale sub-linearly with the corpus size.

Tiptoe's search executes in seconds

	Text search	Image search
Docs searched	364 million	400 million
System config	208 vCPUs	384 vCPUs
End-to-end latency	2.7 s	3.5 s

On the Common crawl + LAION 400M data sets

Tiptoe's search executes in seconds

		Text search	Image search
Docs searched		364 million	400 million
System config		208 vCPUs	384 vCPUs
End-to-end latency		2.7 s	3.5 s
Comm.	ahead of time	42 MiB	50 MiB
	at query time	15 MiB	21 MiB
Client preprocessing		34 s	35 s
Server compute		145 core-s	339 core-s

On the Common crawl + LAION 400M data sets

Tiptoe's search executes in seconds

	Text search	Image search
Docs searched	364 million	400 million
System config	208 vCPUs	384 vCPUs
End-to-end latency	2.7 s	3.5 s
Comm. { ahead of time	42 MiB	50 MiB
Comm. { at query time	15 MiB	21 MiB
Client preprocessing	34 s	35 s
Server compute	145 core-s	339 core-s

Extremely parallelizable, DRAM-bandwidth bound

On the Common crawl + LAION 400M data sets

Tiptoe's search executes in seconds

		Text search	Image search
Docs searched		364 million	400 million
System config		208 vCPUs	384 vCPUs
End-to-end latency		2.7 s	3.5 s
Comm.	ahead of time	42 MiB	50 MiB
	at query time	15 MiB	21 MiB
Client preprocessing		34 s	35 s
Server compute		145 core-s	339 core-s

Off of the critical path

Extremely parallelizable, DRAM-bandwidth bound

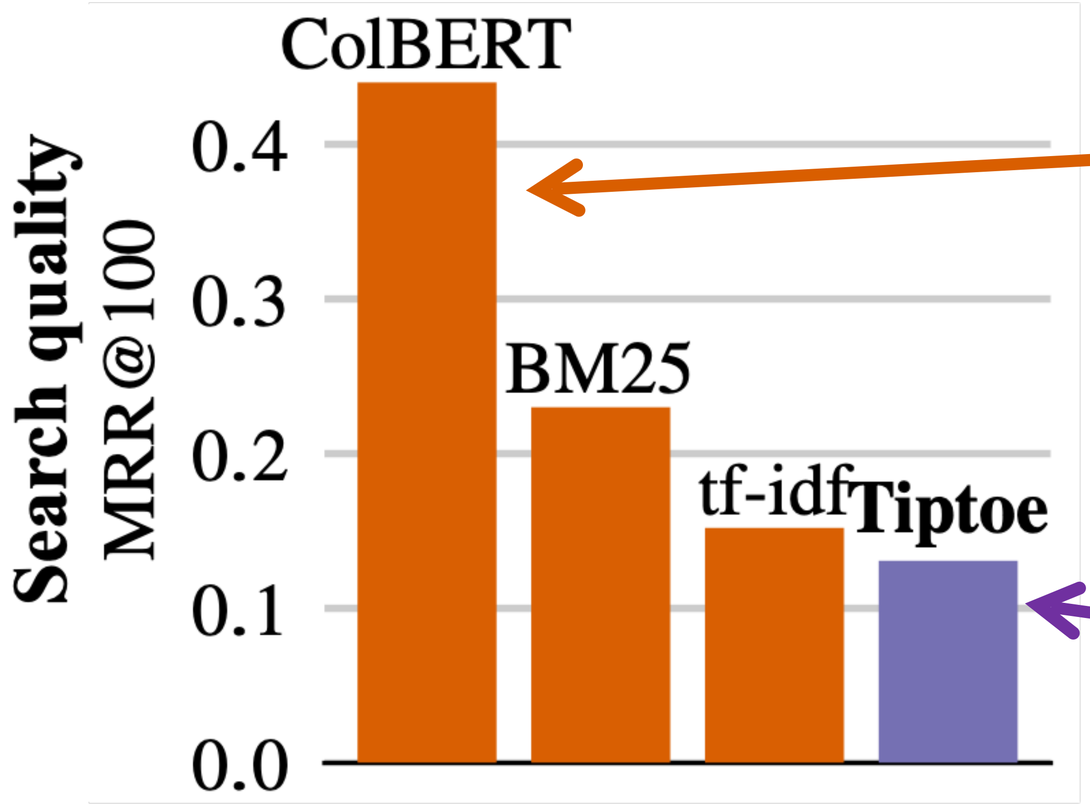
On the Common crawl + LAION 400M data sets

Preprocessing

Run on heterogeneous GPU cluster + 80-core machine

	Text 364M docs	Image 400M docs
Embeddings	92 corehours	583 corehours
Build clusters	927	1,493
Dim. reduction + cluster resizing	312	290
Crypto preproc.	50	120
Total time	0.013 coresec/doc	0.022 coresec/doc

Search quality is “tolerable” (MS MACRO benchmark)



Best result ends up ranked 2 of 100

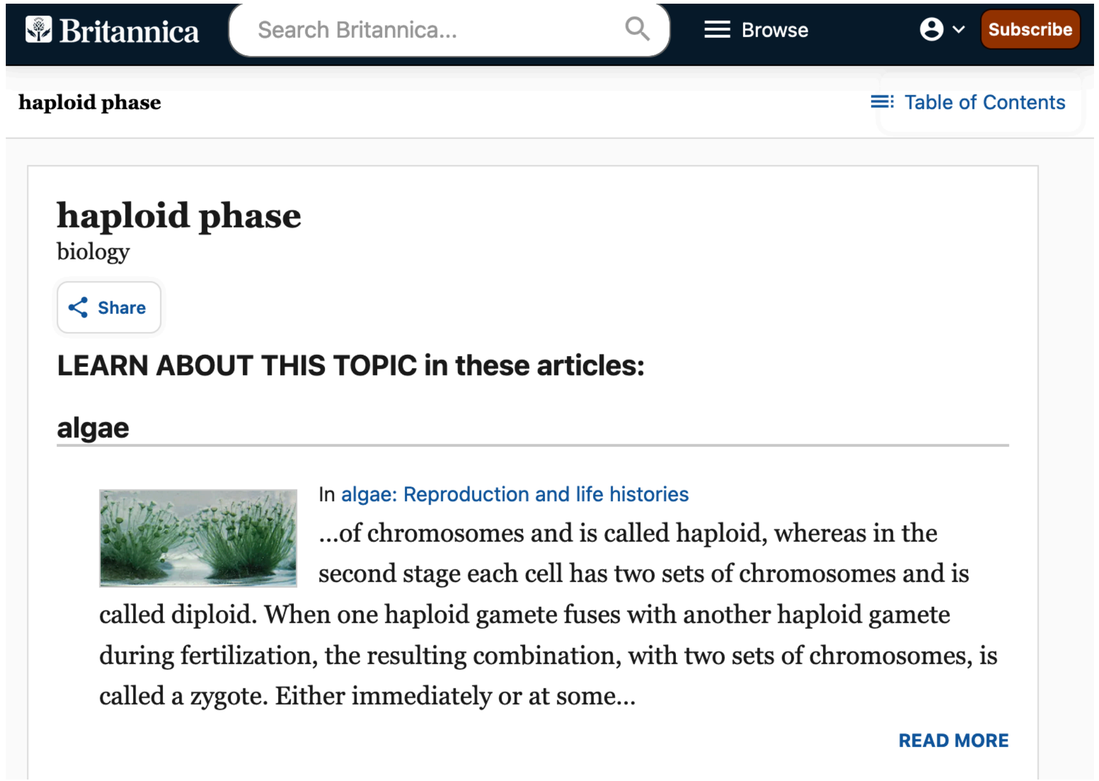
Best result ends up ranked 10 of 100

As embeddings improve, Tiptoe search quality will improve.

Examples: Text search

`how long before eagles get feathers`

`the meaning of haploid cell`



Exact string search is not as good…

Examples: Image search

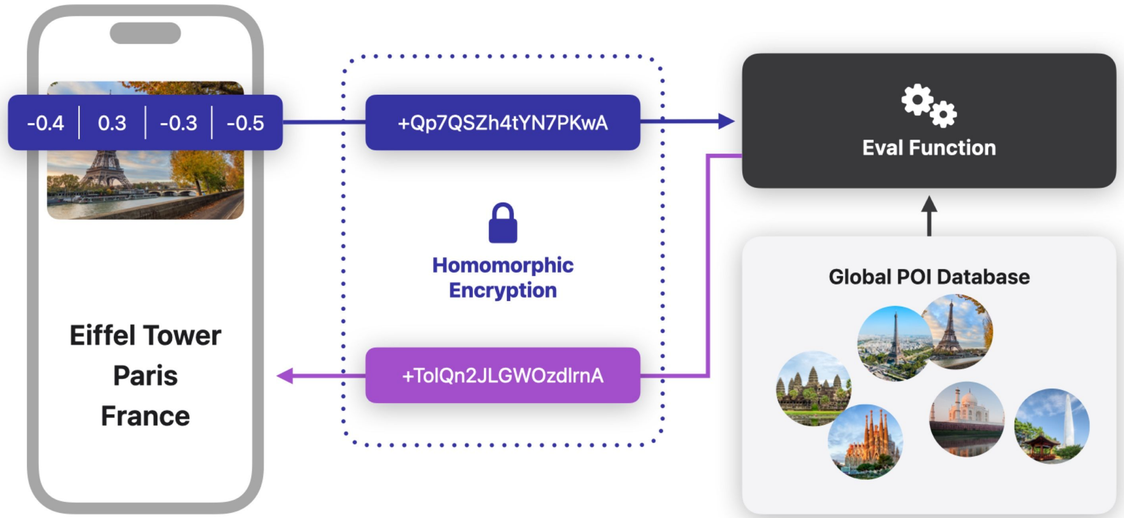
close up of a pizza sitting on top of a plate

zebra standing on top of a lush green field

brown bear standing in front of a tree

Apple’s Enhanced Visual Search

- Apple’s Enhanced Visual Search lets users see if their photos contain particular landmarks
- Uses design techniques drawn from Tiptoe
- For their use case, the cost of scanning over every cluster is too high
- Their solution: add differentially private noise to hide the identity of the queried cluster (can now query a sublinear number of clusters)



https://machinelearning.apple.com/research/homomorphic-encryption

Outline

1. SimplePIR
2. Tiptoe
3. **Student presentation: Compass**

References

Asi, Hilal, Fabian Boemer, Nicholas Genise, Muhammad Haris Mughees, Tabitha Ogilvie, Rehan Rishi, Kunal Talwar et al. "Scalable private search with wally." *arXiv preprint arXiv:2406.06761* (2024).

Henzinger, Alexandra, Matthew M. Hong, Henry Corrigan-Gibbs, Sarah Meiklejohn, and Vinod Vaikuntanathan. "One server for the price of two: Simple and fast {Single-Server} private information retrieval." In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 3889-3905. 2023.

Henzinger, Alexandra, Emma Dauterman, Henry Corrigan-Gibbs, and Nickolai Zeldovich. "Private web search with Tiptoe." In *Proceedings of the 29th symposium on operating systems principles*, pp. 396-416. 2023.

Regev, Oded. "On lattices, learning with errors, random linear codes, and cryptography." *Journal of the ACM (JACM)* 56, no. 6 (2009): 1-40.

https://65610.csail.mit.edu/2025/lec/l05-pke.pdf

https://machinelearning.apple.com/research/homomorphic-encryption