DORY: An Encrypted Search System with Distributed Trust

Emma Dauterman, Eric Feng, Ellen Luo, Raluca Ada Popa, and Ion Stoica
UC Berkeley

OSDI 2020

End-to-end encrypted filesystems

End-to-end encrypted systems are increasingly popular.

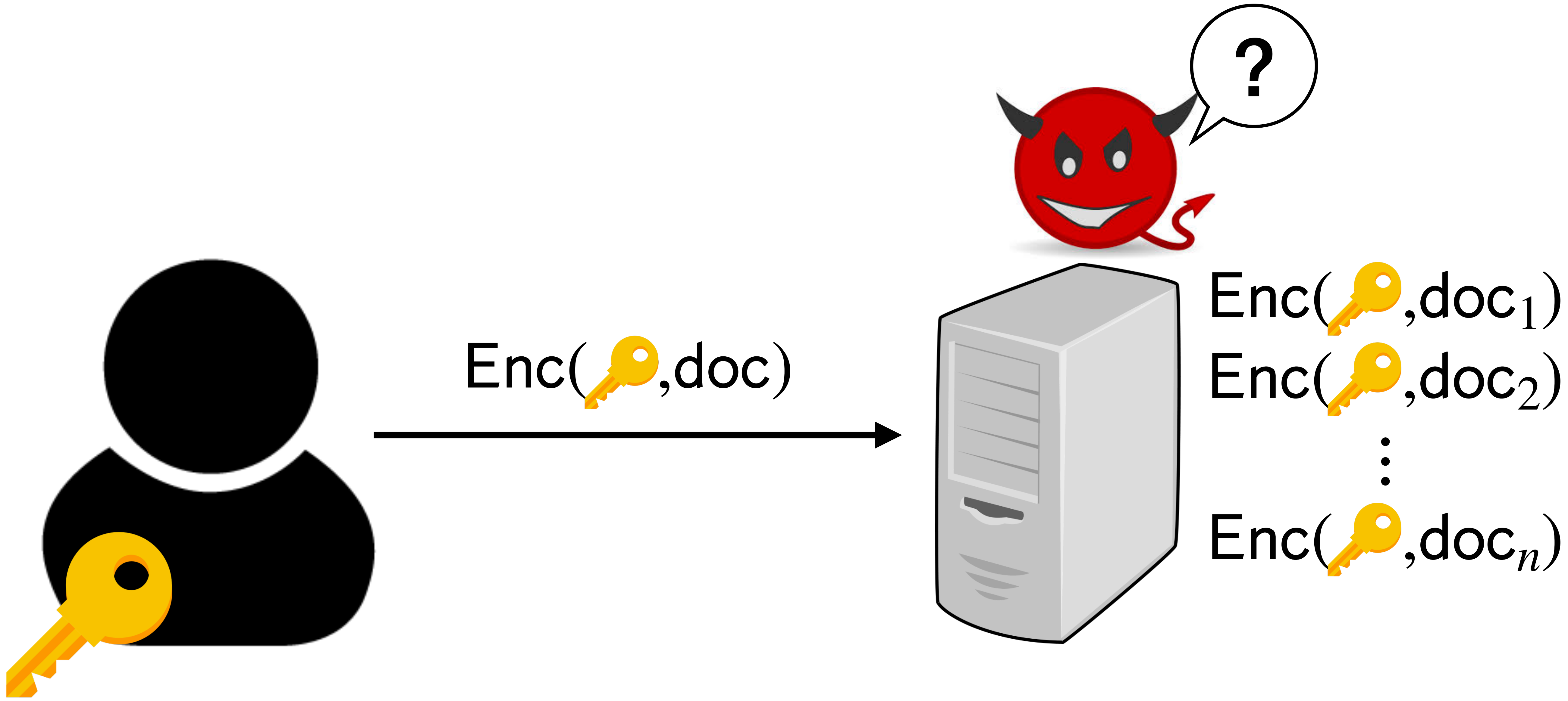
Keybase PREVEIL SpiderOak sync.com tresorit

End-to-end encrypted filesystems

End-to-end encrypted systems are increasingly popular.

Keybase PREVEIL SpiderOak sync.com tresorit

Provide strong security guarantees if attacker compromises server.



Users expect the ability to search

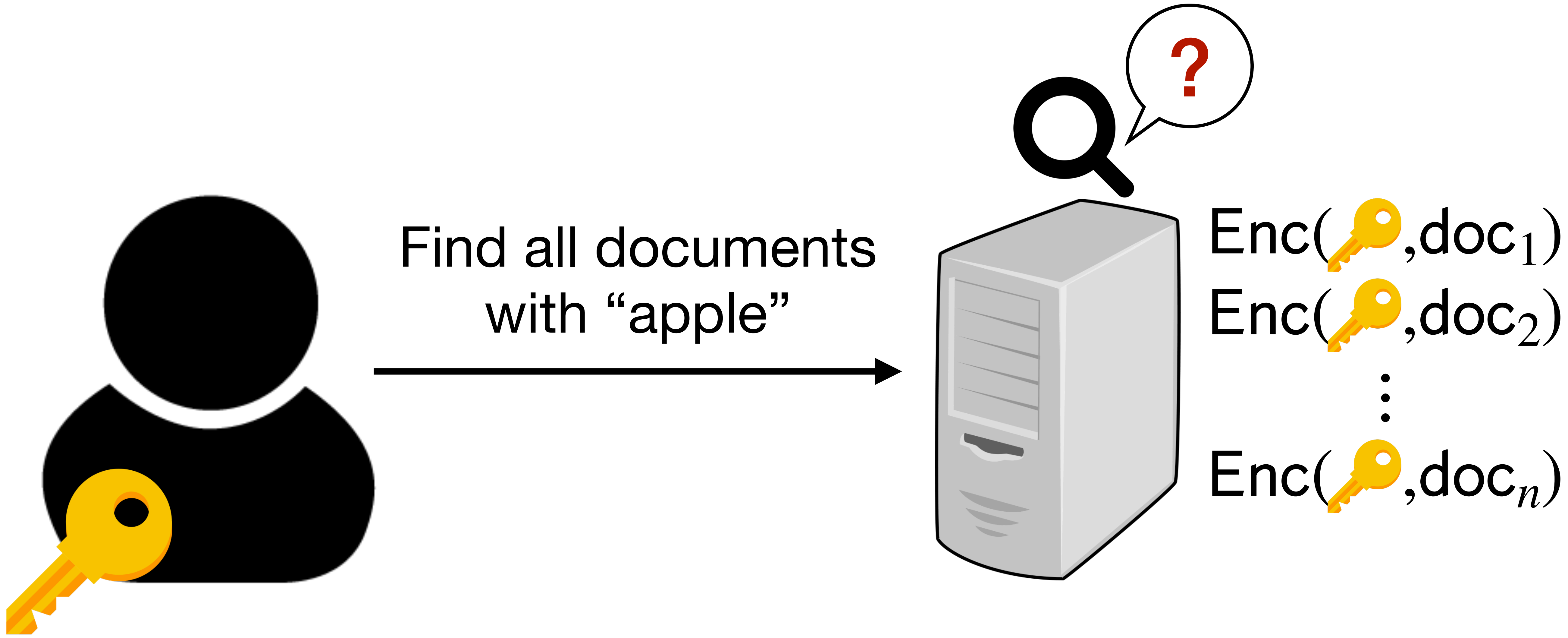
Users expect the ability to search

Users expect the ability to search

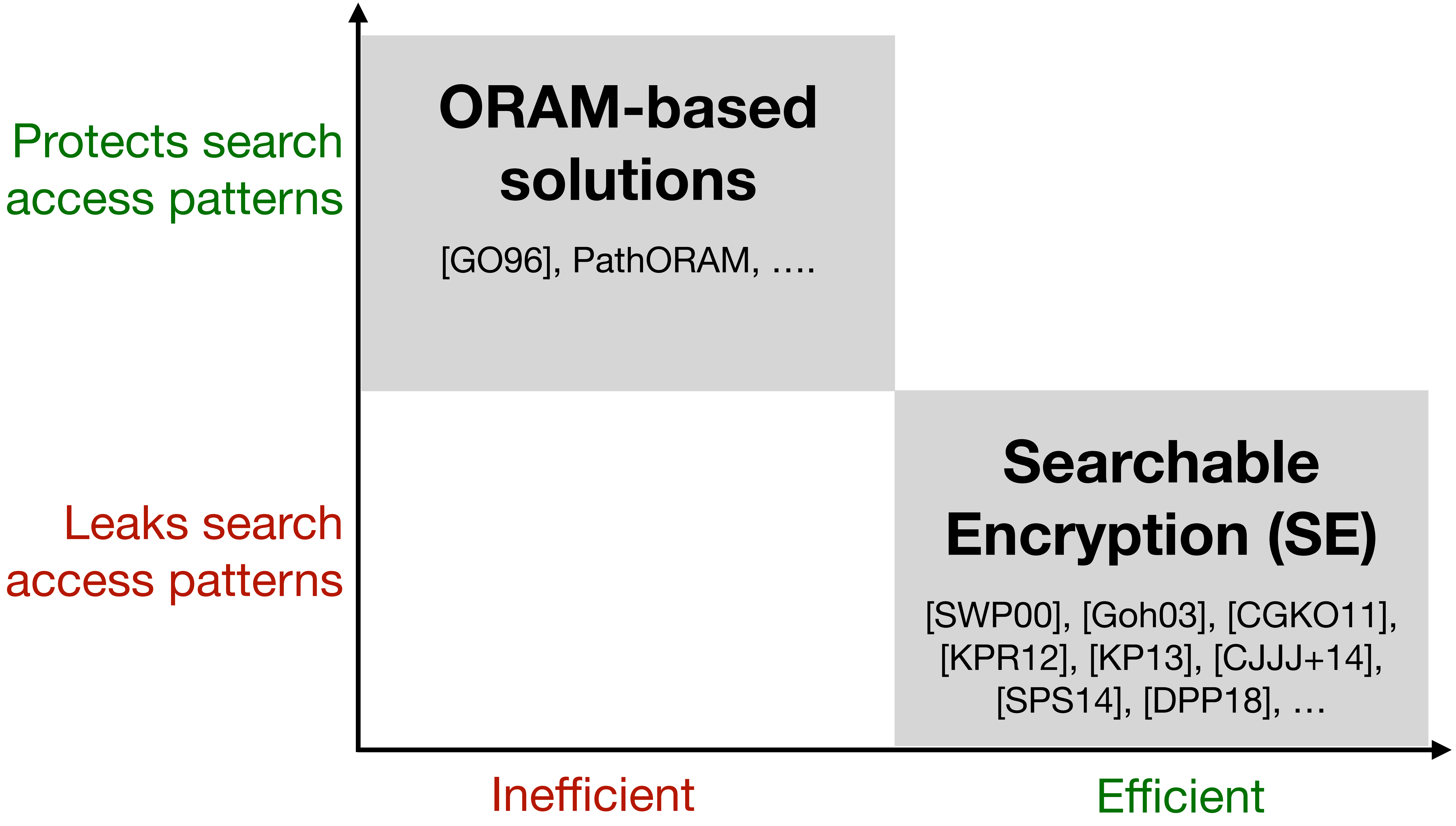
Doc 1
Doc 7
Doc 21
Doc 53

Search for end-to-end encrypted filesystems

Challenge: server cannot decrypt data to search.

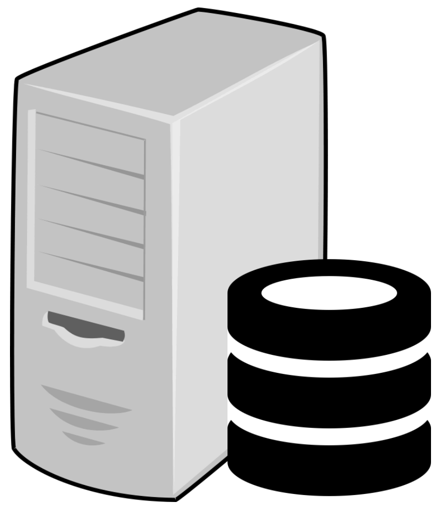


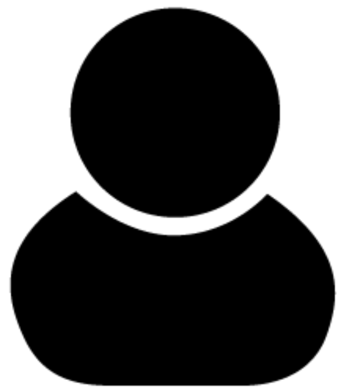
Tradeoff between security and performance



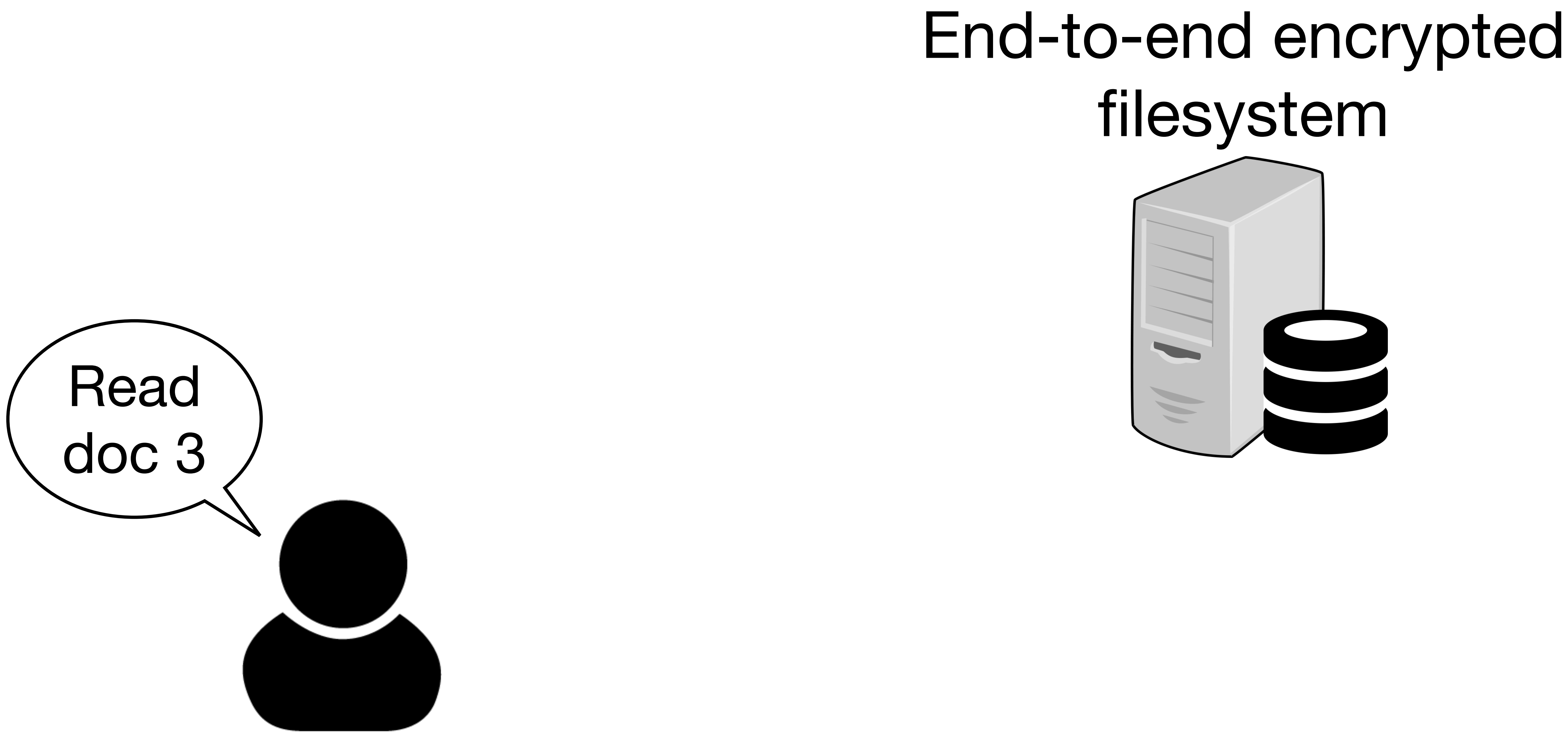
Filesystem leakage is at the document level

End-to-end encrypted
filesystem

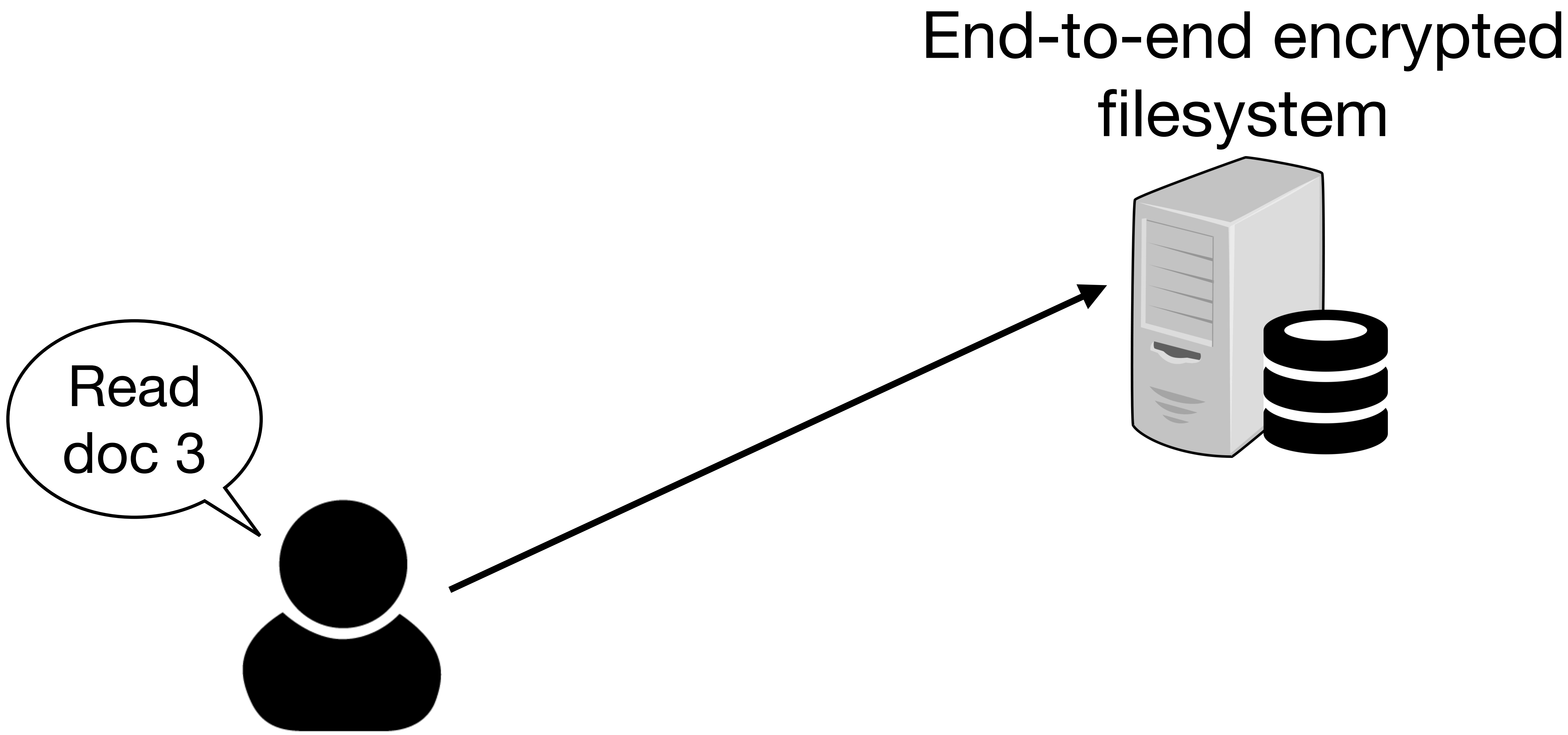




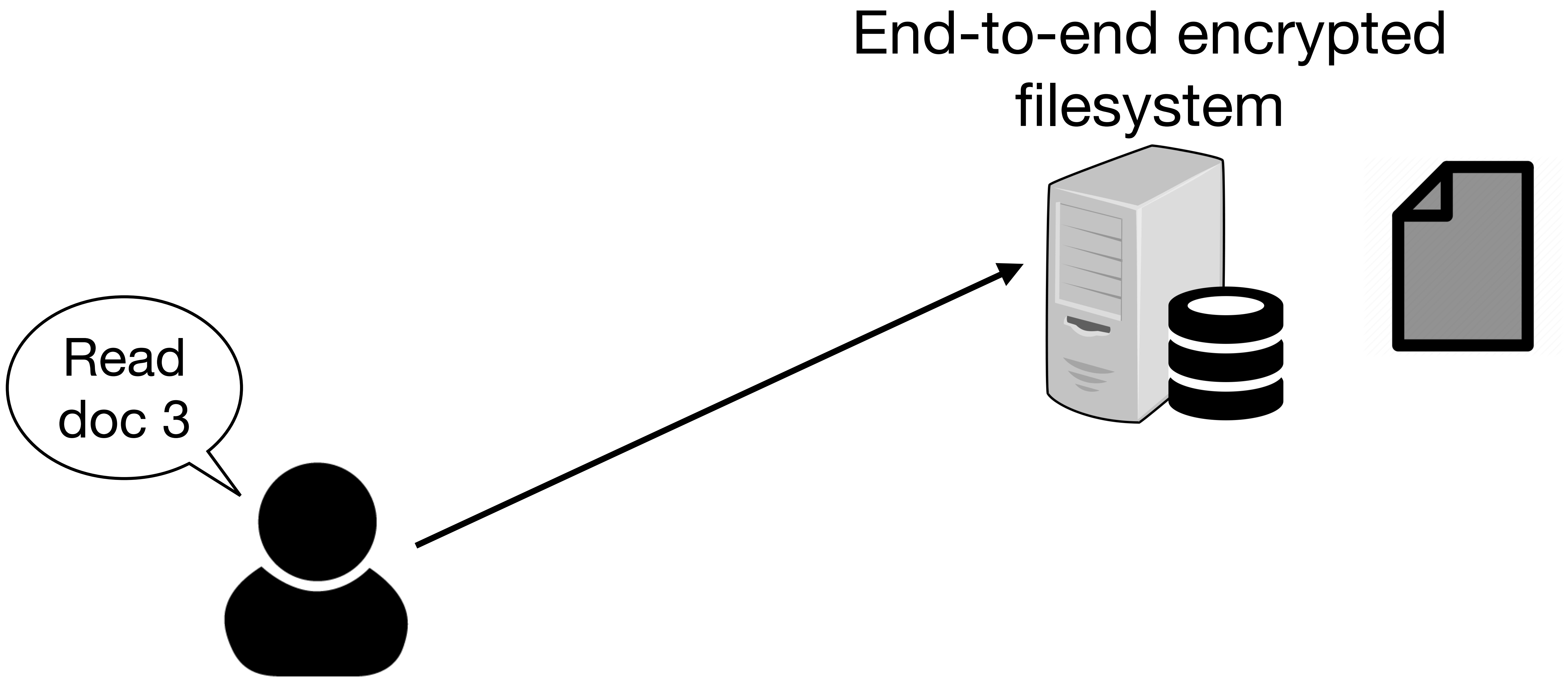
Filesystem leakage is at the document level



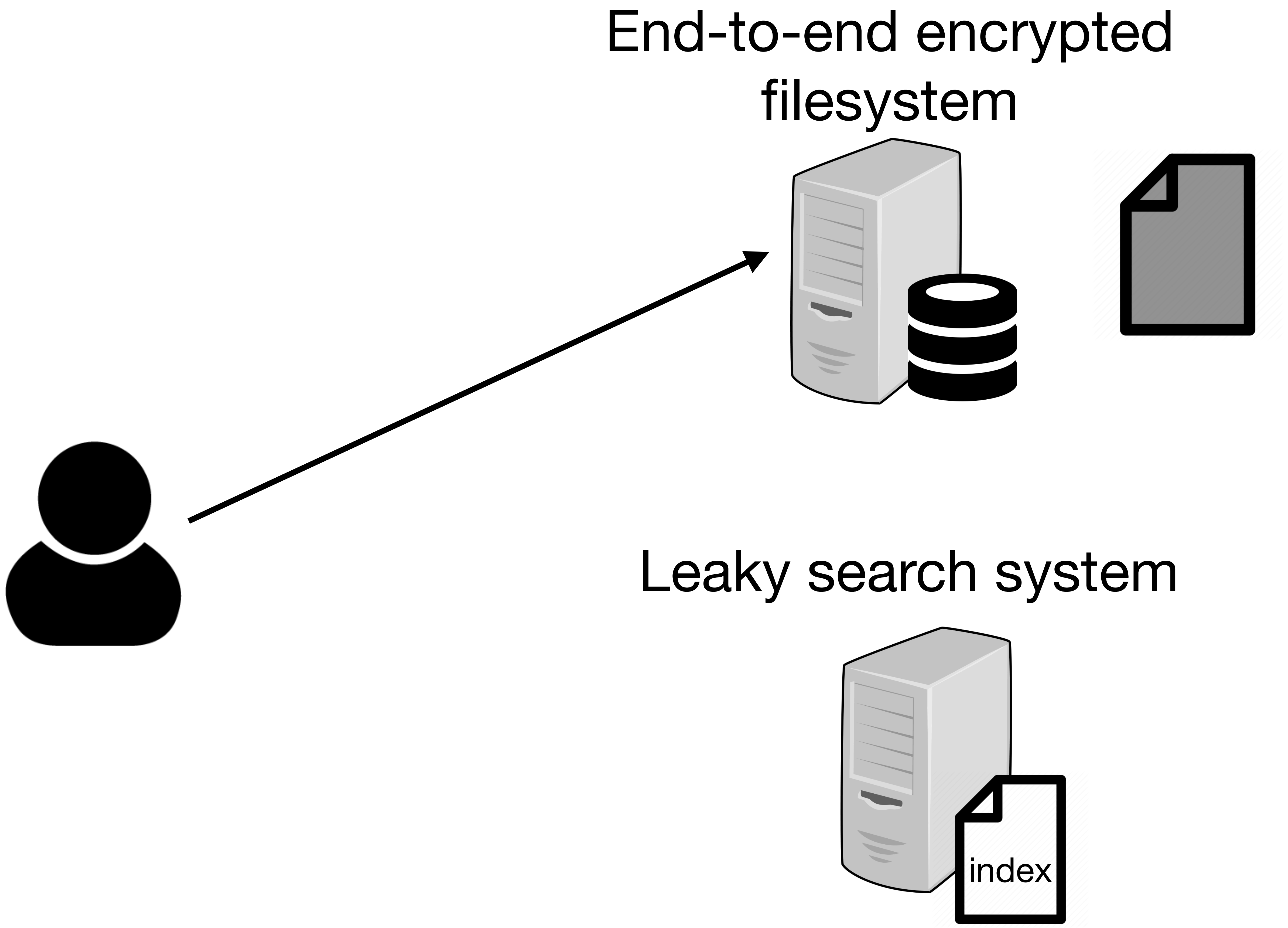
Filesystem leakage is at the document level



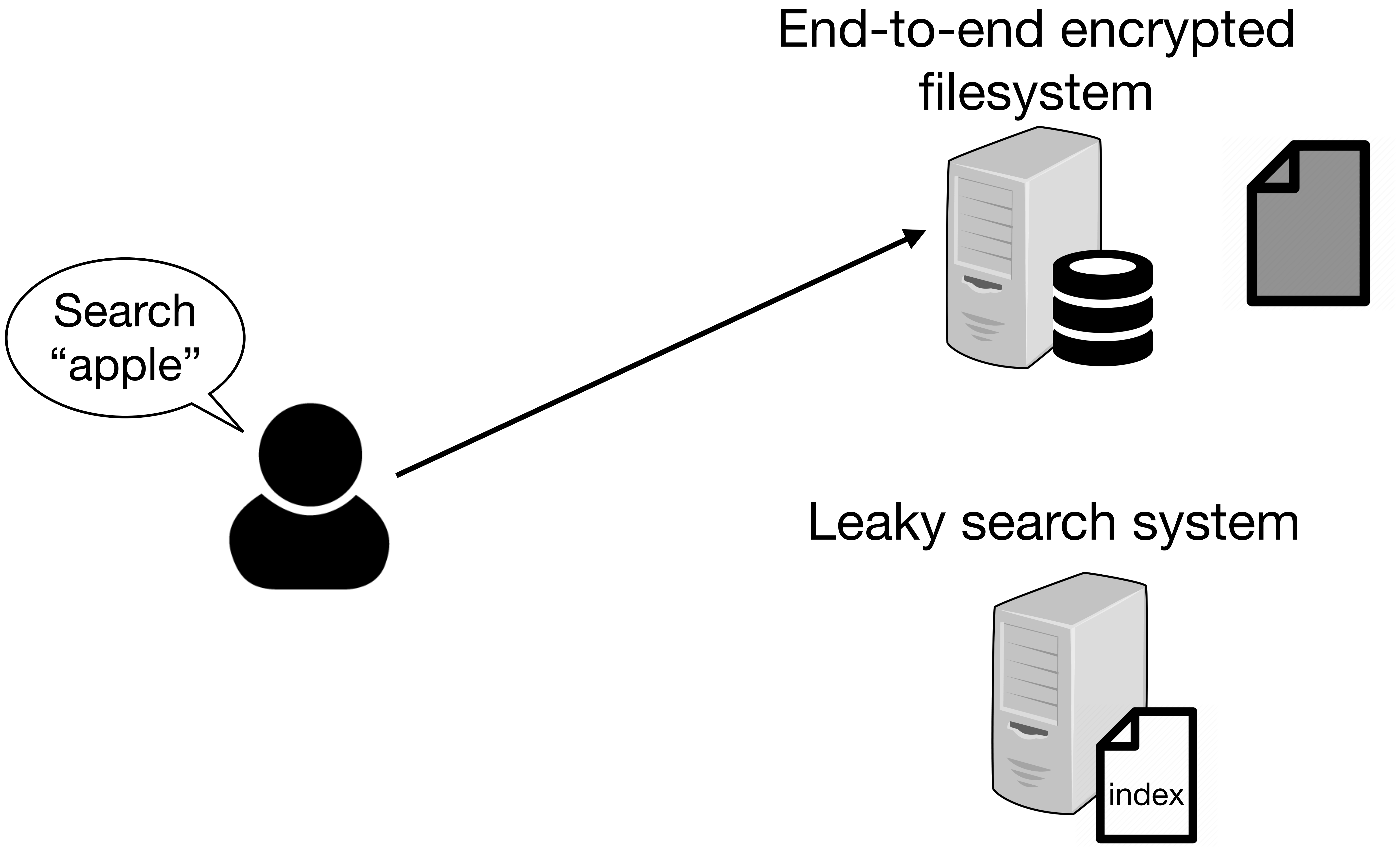
Filesystem leakage is at the document level



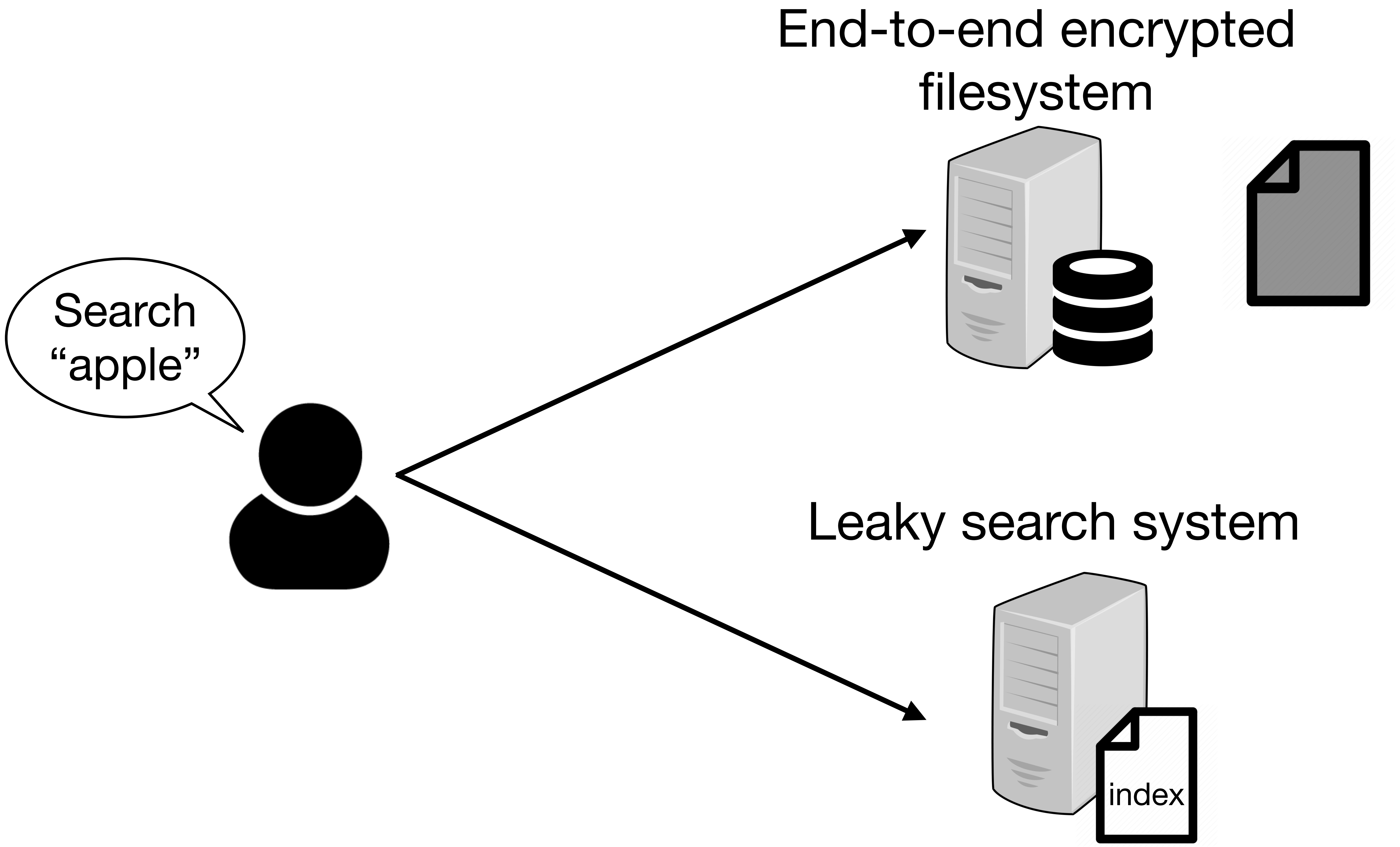
Search access pattern leakage is at the word level



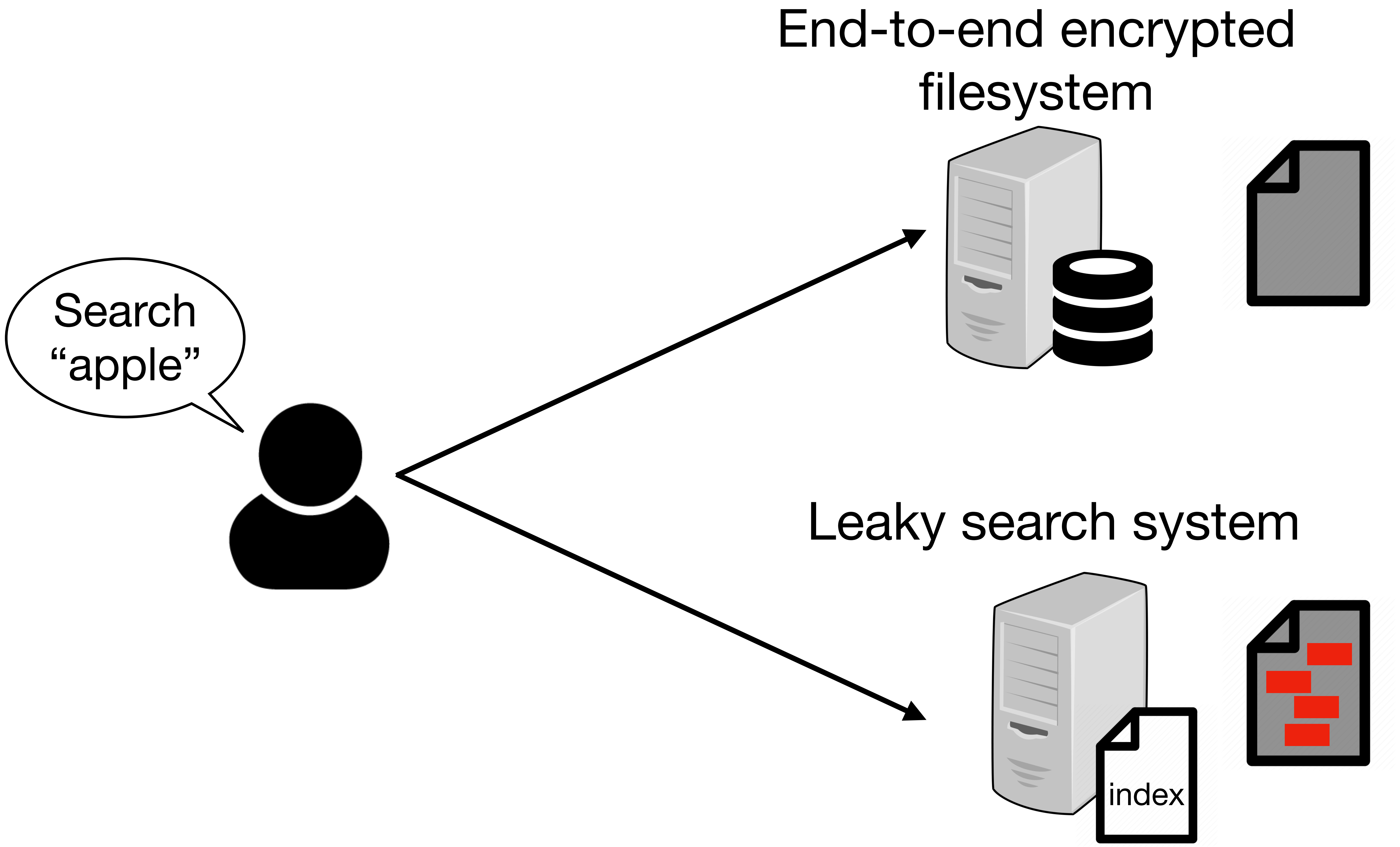
Search access pattern leakage is at the word level



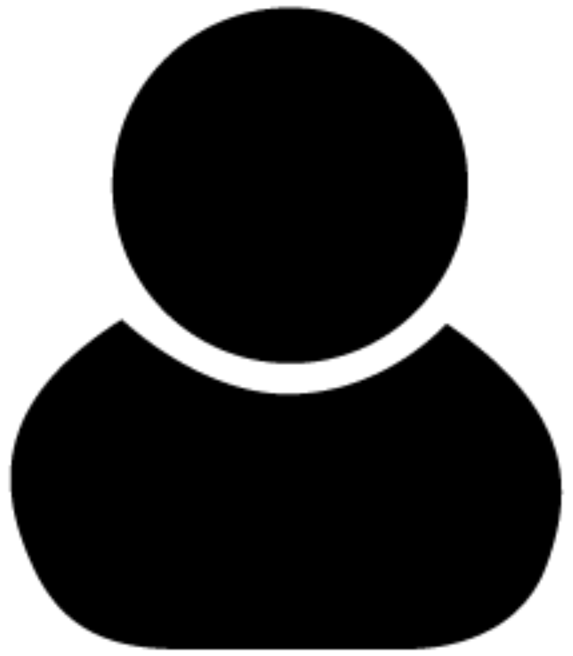
Search access pattern leakage is at the word level



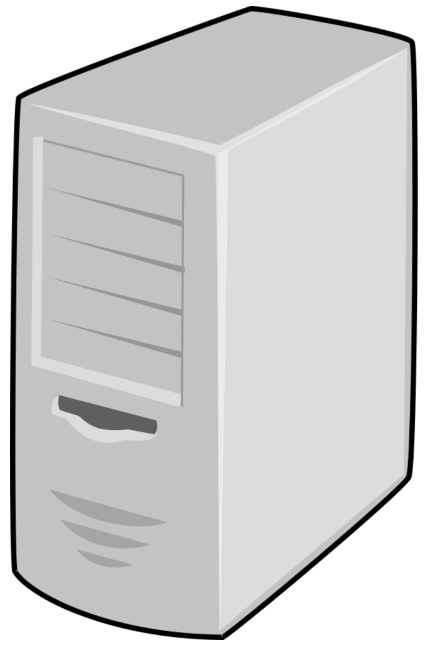
Search access pattern leakage is at the word level



Search access patterns can be used to recover document plaintext



File Injection Attack [ZKP16]



$\mathsf{Enc}(\mathsf{word}_1)$: $\mathsf{Enc}(\mathsf{doc}_1), \ldots$

$\mathsf{Enc}(\mathsf{word}_2)$: $\mathsf{Enc}(\mathsf{doc}_{12}), \ldots$

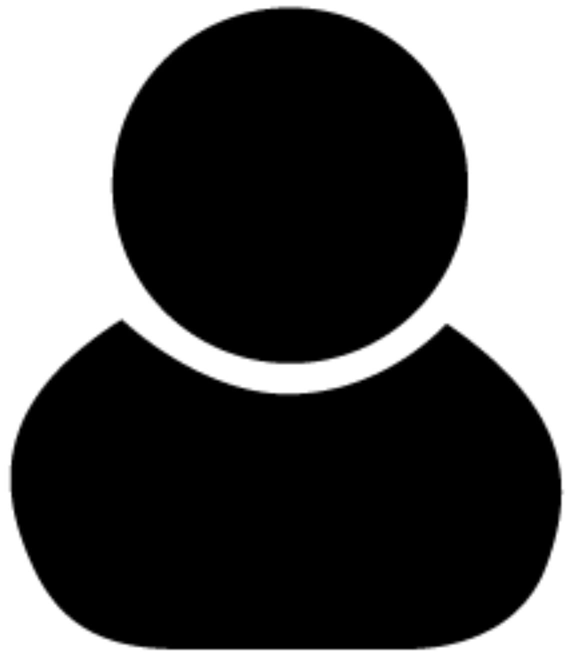
$\mathsf{Enc}(\mathsf{flu})$:

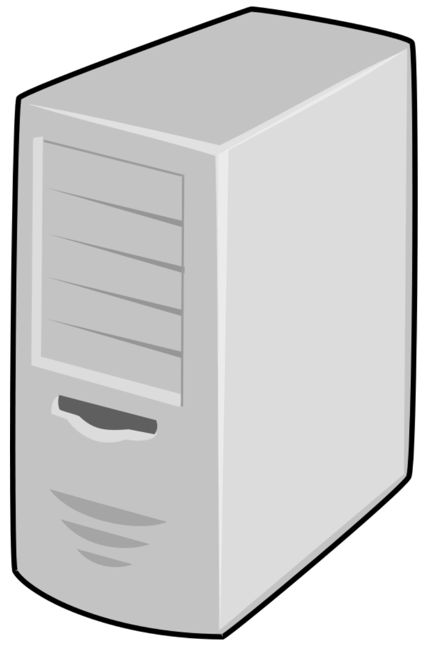
$\vdots$

$\mathsf{Enc}(\mathsf{word}_n)$: $\mathsf{Enc}(\mathsf{doc}_5), \ldots$

Search access patterns can be used to recover document plaintext

File Injection Attack [ZKP16]





$\text{Enc}(\text{word}_1) : \text{Enc}(\text{doc}_1), \ldots$

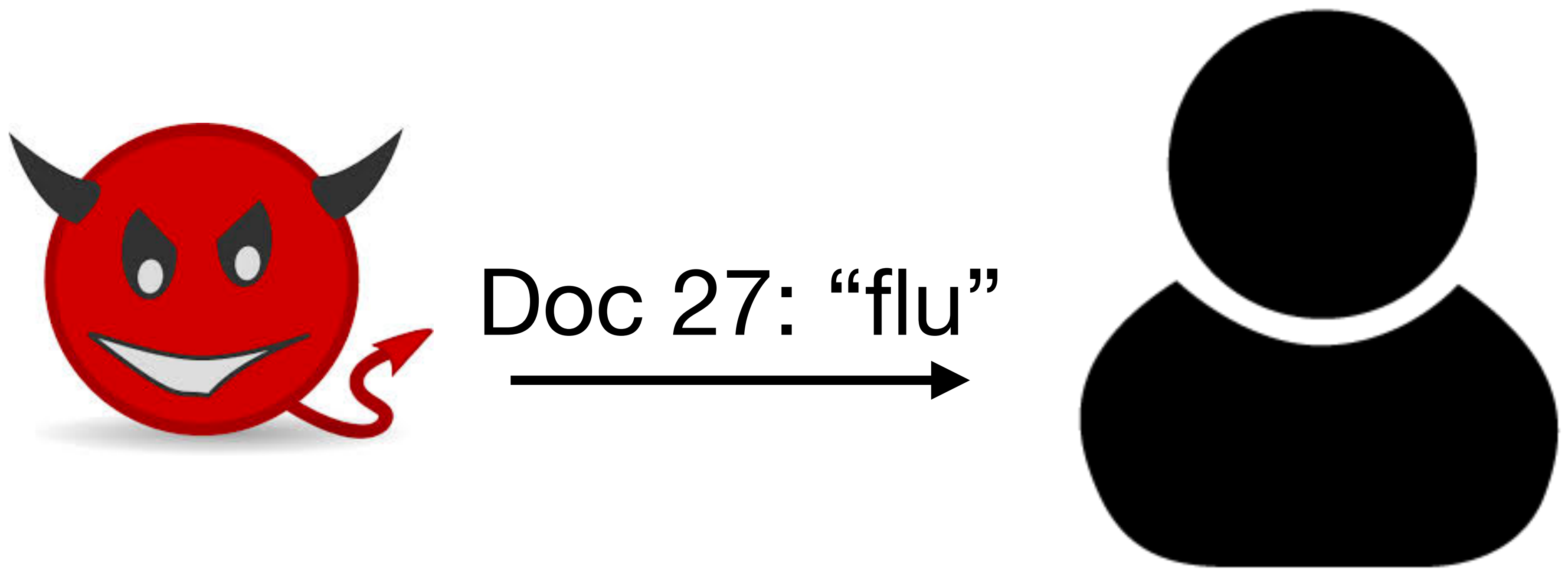
$\text{Enc}(\text{word}_2) : \text{Enc}(\text{doc}_{12}), \ldots$

$\text{Enc}(\text{flu}) :$

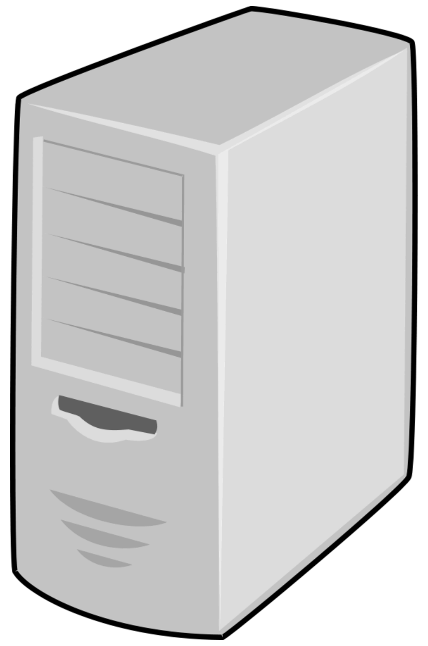
$\vdots$

$\text{Enc}(\text{word}_n) : \text{Enc}(\text{doc}_5), \ldots$

Search access patterns can be used to recover document plaintext

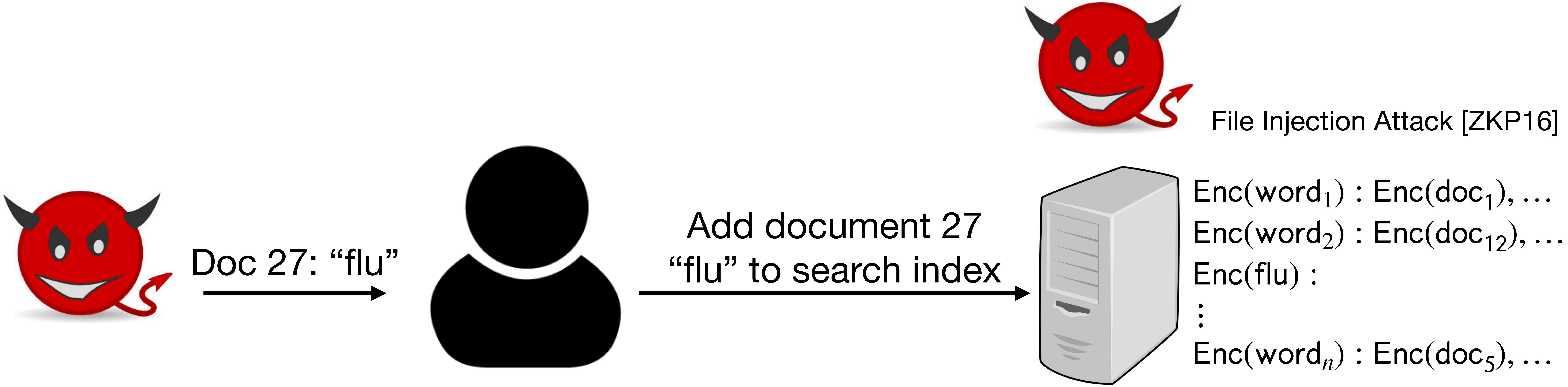


File Injection Attack [ZKP16]

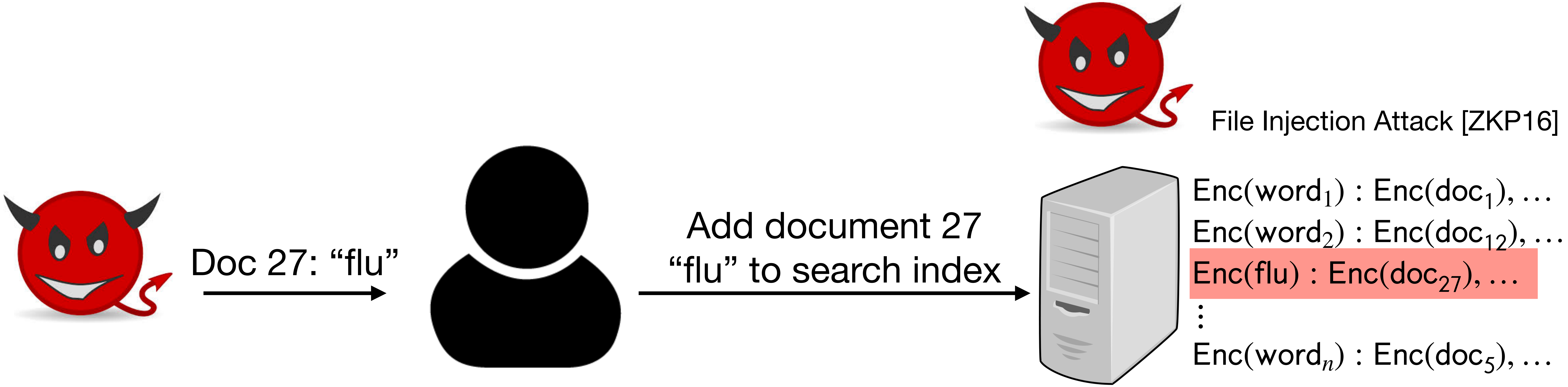


$\text{Enc}(\text{word}_1)$: $\text{Enc}(\text{doc}_1), \ldots$
$\text{Enc}(\text{word}_2)$: $\text{Enc}(\text{doc}_{12}), \ldots$
$\text{Enc}(\text{flu})$:
$\vdots$
$\text{Enc}(\text{word}_n)$: $\text{Enc}(\text{doc}_5), \ldots$

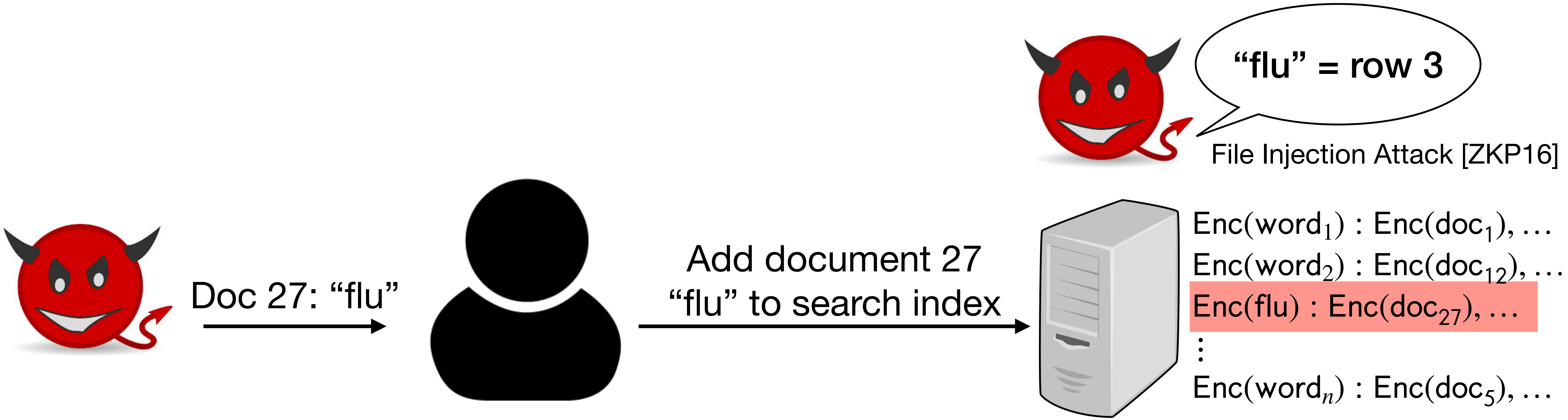
Search access patterns can be used to recover document plaintext



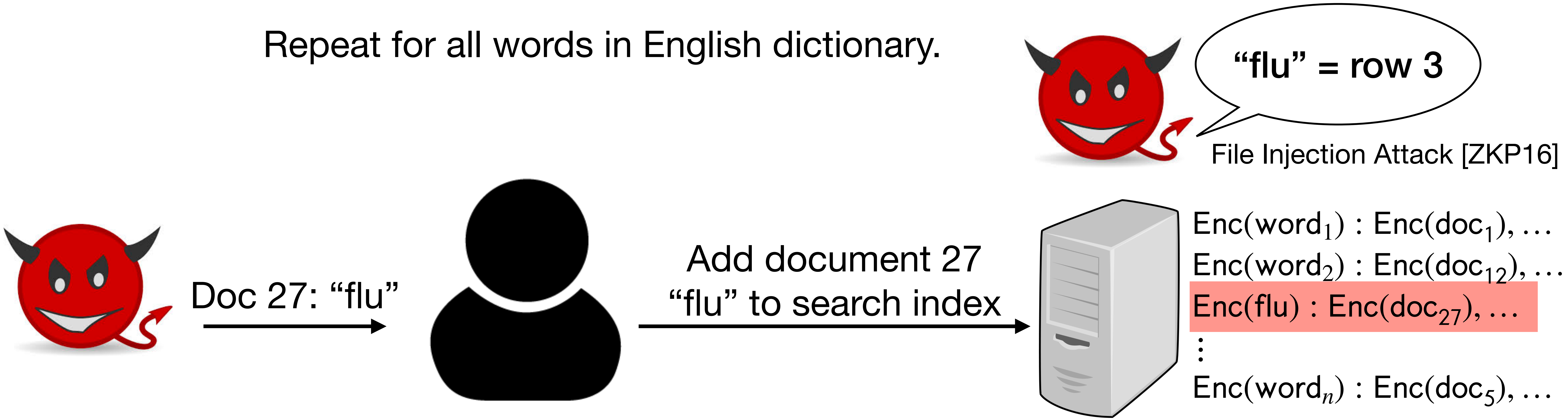
Search access patterns can be used to recover document plaintext



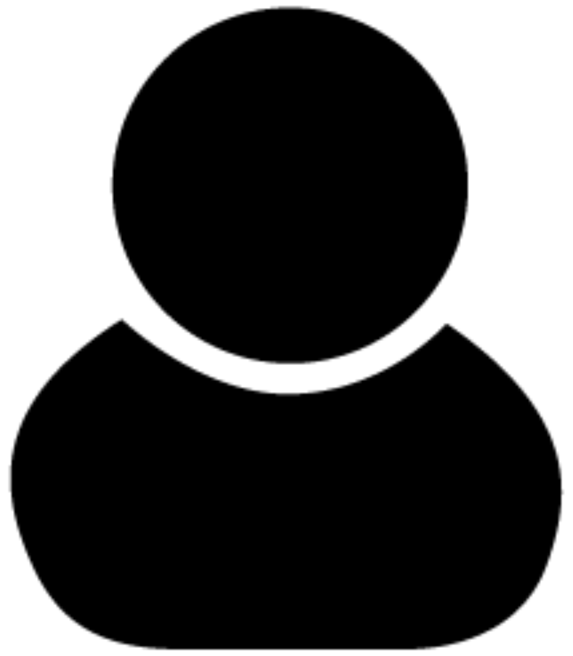
Search access patterns can be used to recover document plaintext



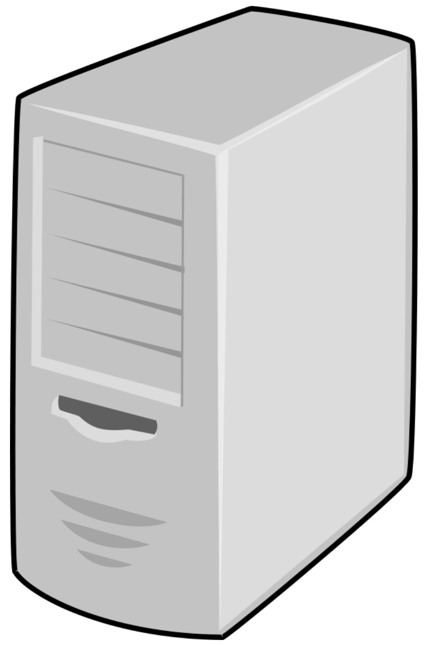
Search access patterns can be used to recover document plaintext



Search access patterns can be used to recover document plaintext



File Injection Attack [ZKP16]



$\mathsf{Enc}(\mathsf{word}_1) : \mathsf{Enc}(\mathsf{doc}_1), \ldots$

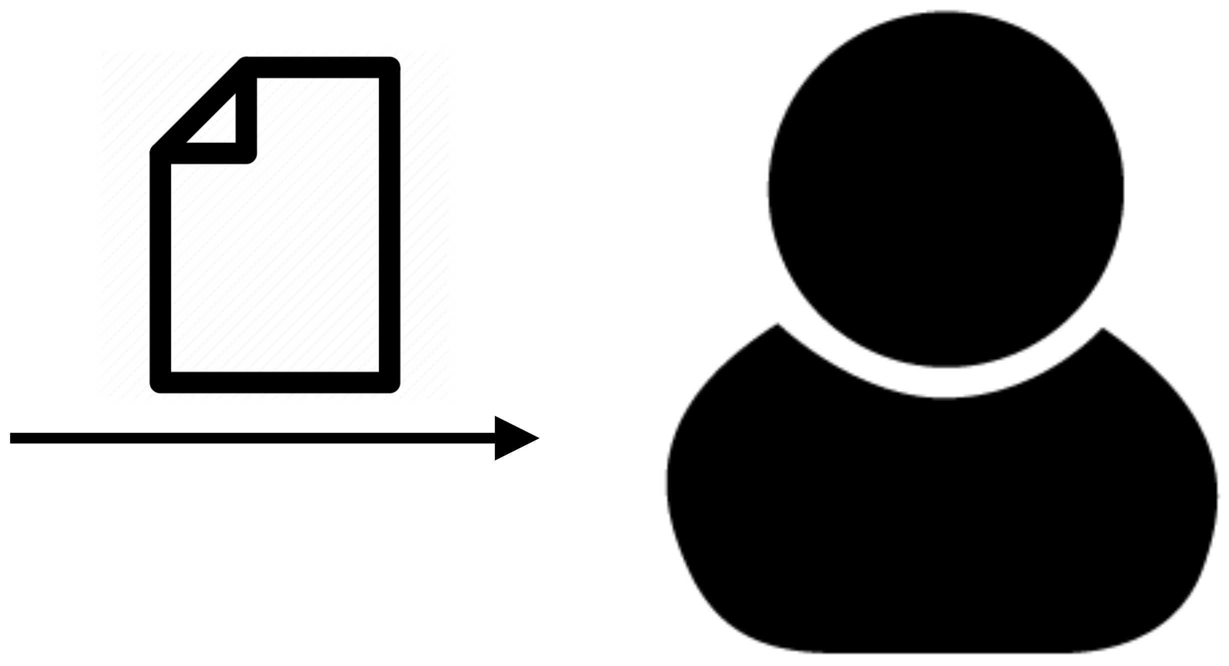
$\mathsf{Enc}(\mathsf{word}_2) : \mathsf{Enc}(\mathsf{doc}_{12}), \ldots$

$\mathsf{Enc}(\mathsf{flu}) : \mathsf{Enc}(\mathsf{doc}_{27}), \ldots$

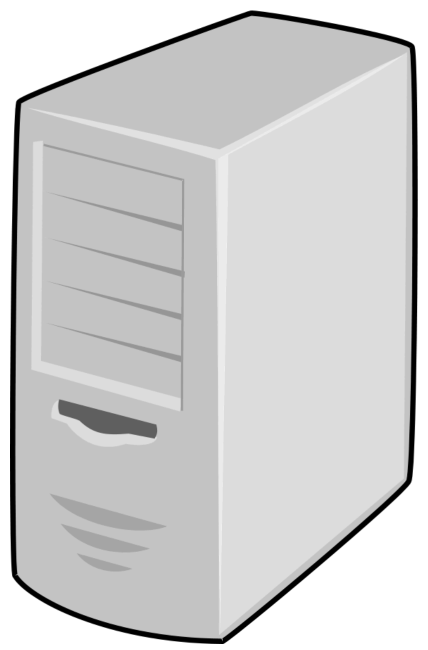
$\vdots$

$\mathsf{Enc}(\mathsf{word}_n) : \mathsf{Enc}(\mathsf{doc}_5), \ldots$

Search access patterns can be used to recover document plaintext

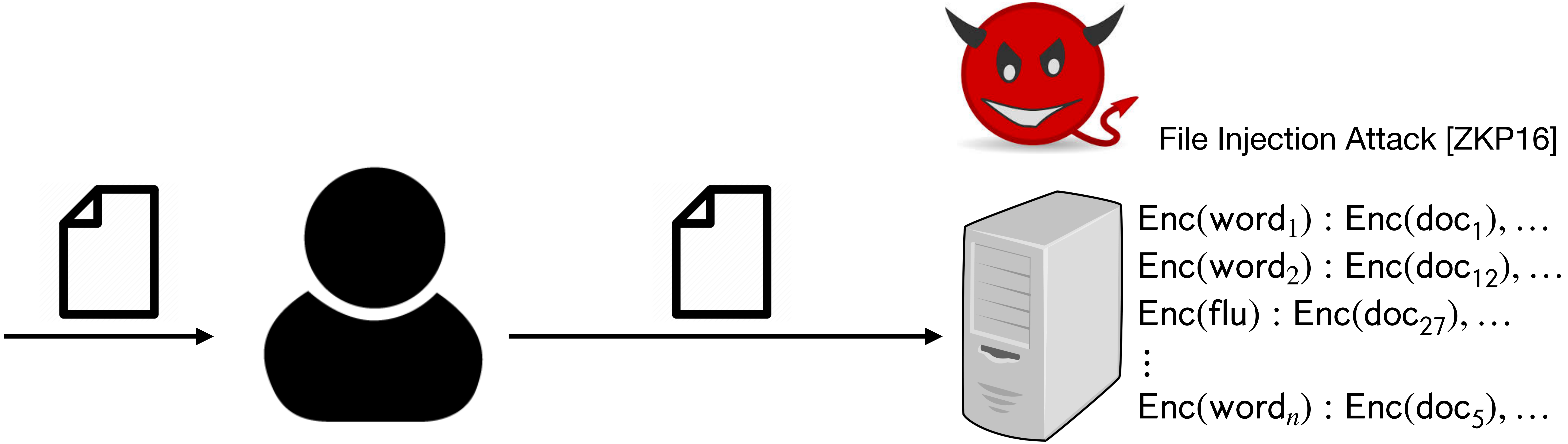


File Injection Attack [ZKP16]

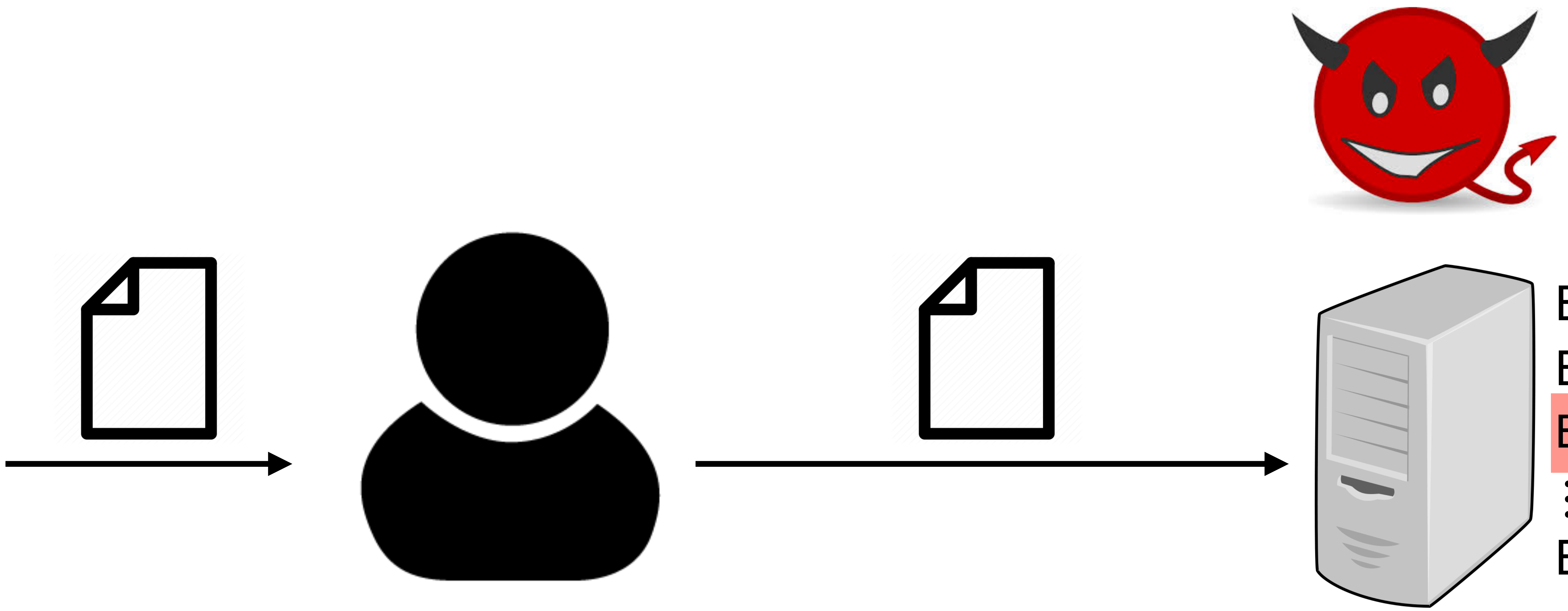


$\text{Enc}(\text{word}_1) : \text{Enc}(\text{doc}_1), \ldots$
$\text{Enc}(\text{word}_2) : \text{Enc}(\text{doc}_{12}), \ldots$
$\text{Enc}(\text{flu}) : \text{Enc}(\text{doc}_{27}), \ldots$
$\vdots$
$\text{Enc}(\text{word}_n) : \text{Enc}(\text{doc}_5), \ldots$

Search access patterns can be used to recover document plaintext



Search access patterns can be used to recover document plaintext



File Injection Attack [ZKP16]

$\mathsf{Enc}(\mathsf{word}_1) : \mathsf{Enc}(\mathsf{doc}_1), \ldots$

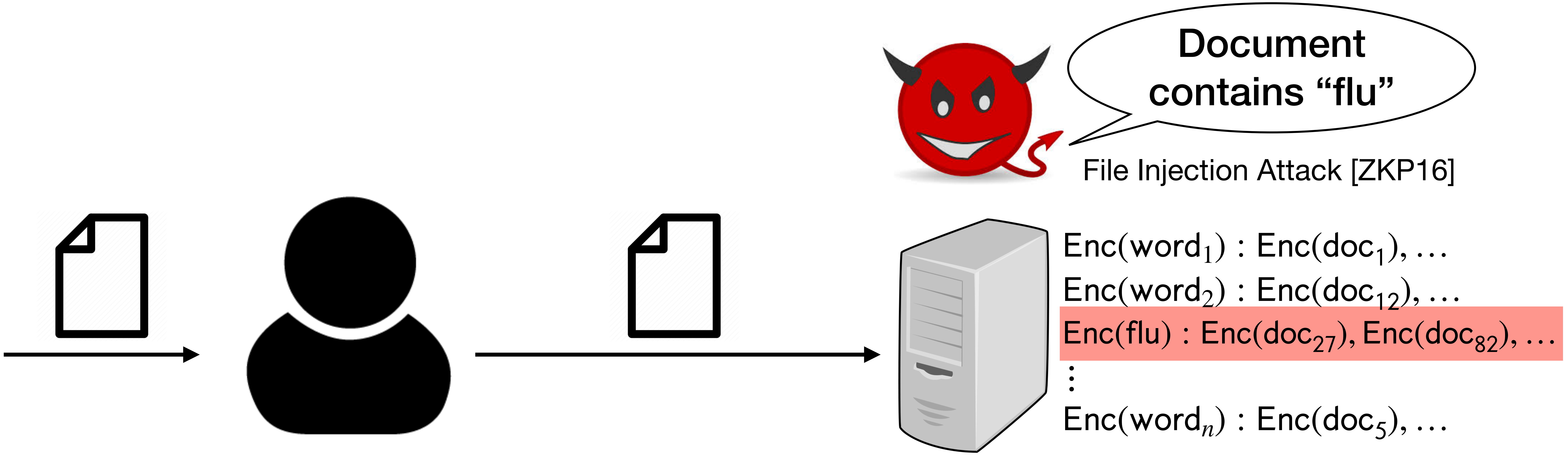
$\mathsf{Enc}(\mathsf{word}_2) : \mathsf{Enc}(\mathsf{doc}_{12}), \ldots$

$\mathsf{Enc}(\mathsf{flu}) : \mathsf{Enc}(\mathsf{doc}_{27}), \mathsf{Enc}(\mathsf{doc}_{82}), \ldots$

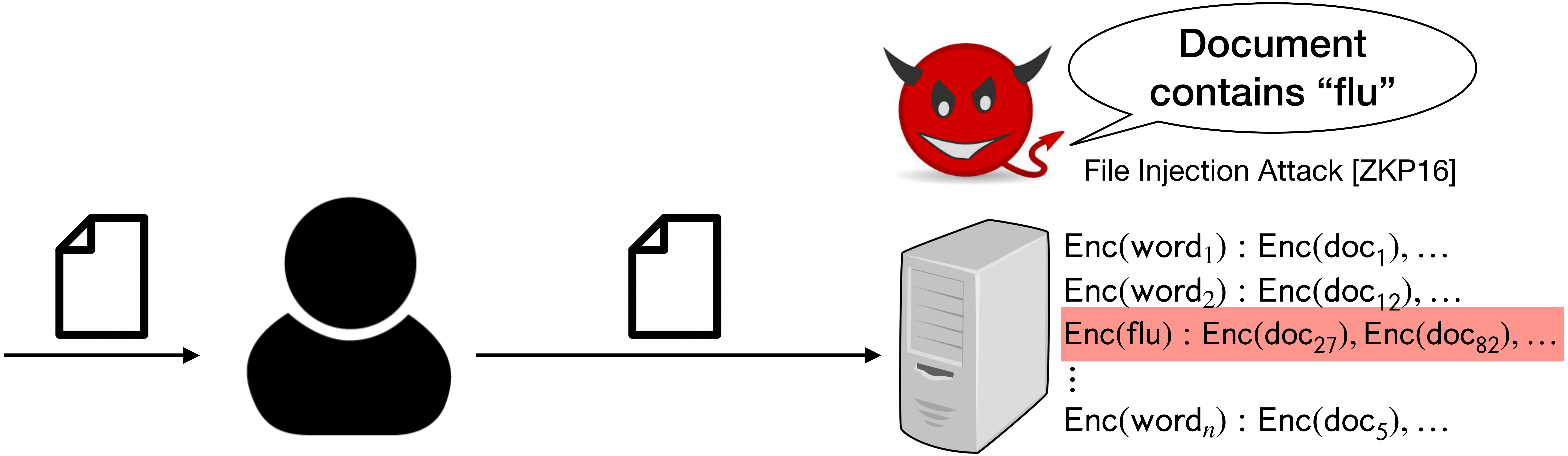
$\vdots$

$\mathsf{Enc}(\mathsf{word}_n) : \mathsf{Enc}(\mathsf{doc}_5), \ldots$

Search access patterns can be used to recover document plaintext



Search access patterns can be used to recover document plaintext



… and many more attacks

[IKK12], [CGPR15], [KKNO16], [LZWT14], [PW16], [GTS17], [PWLP20], ….

Drawbacks of ORAM-based solutions

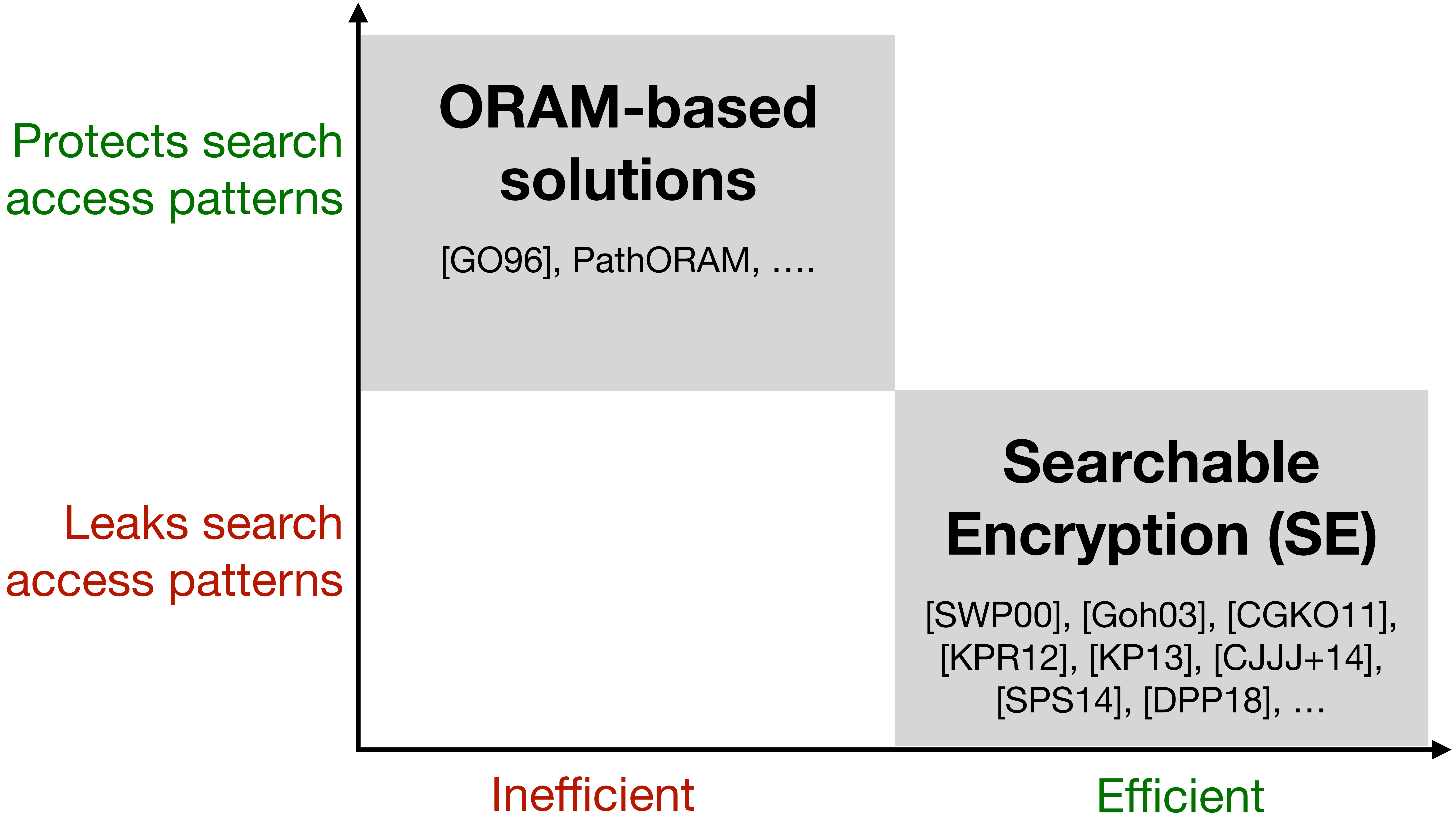
ORAM: client can read/write data at server and hide access patterns [GO96, SVSF+13].

Can implement search by building inverted index in ORAM.

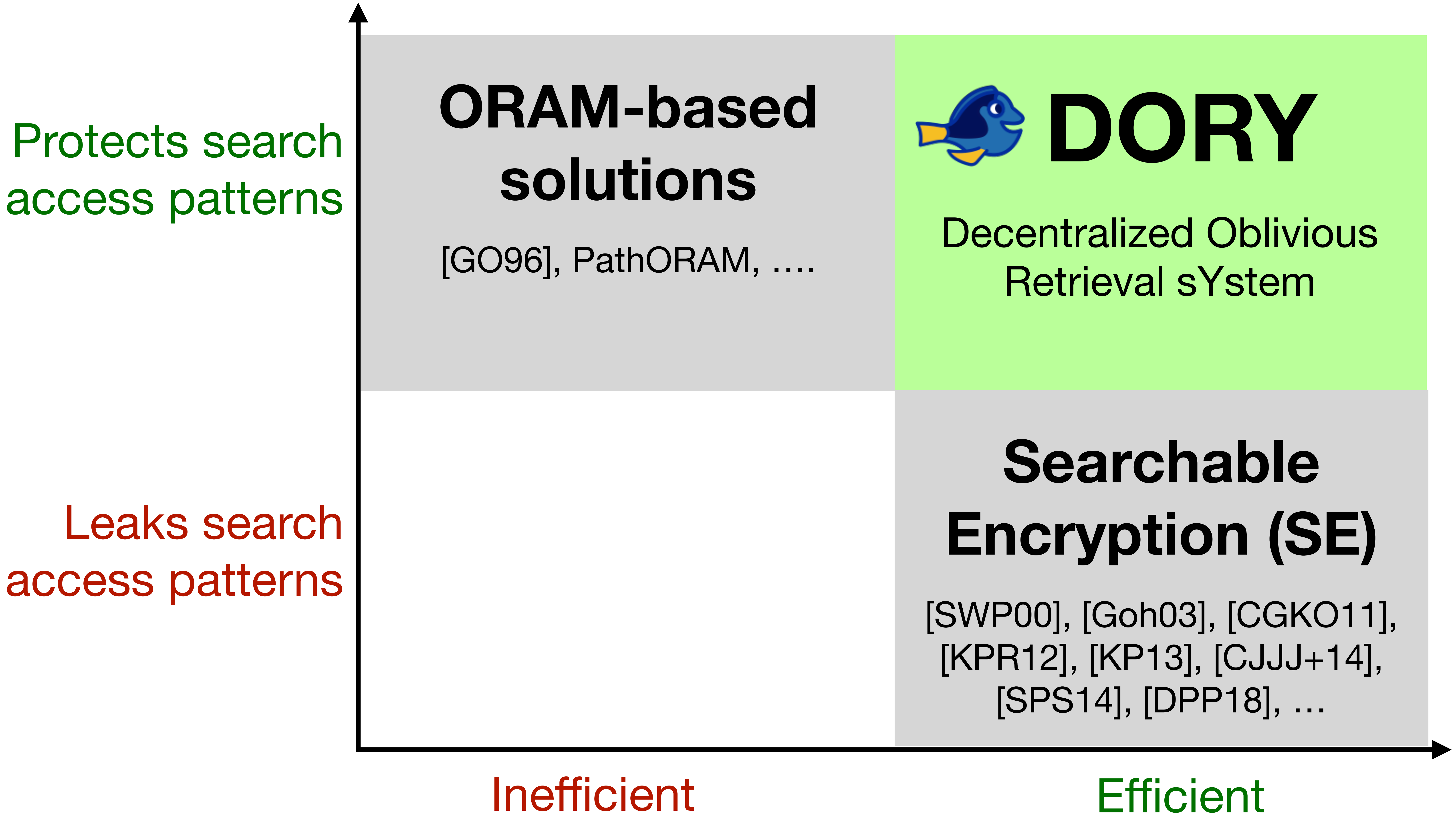
+ Runtime logarithmic in index size.

- Large constants make cost prohibitive for encrypted filesystems.

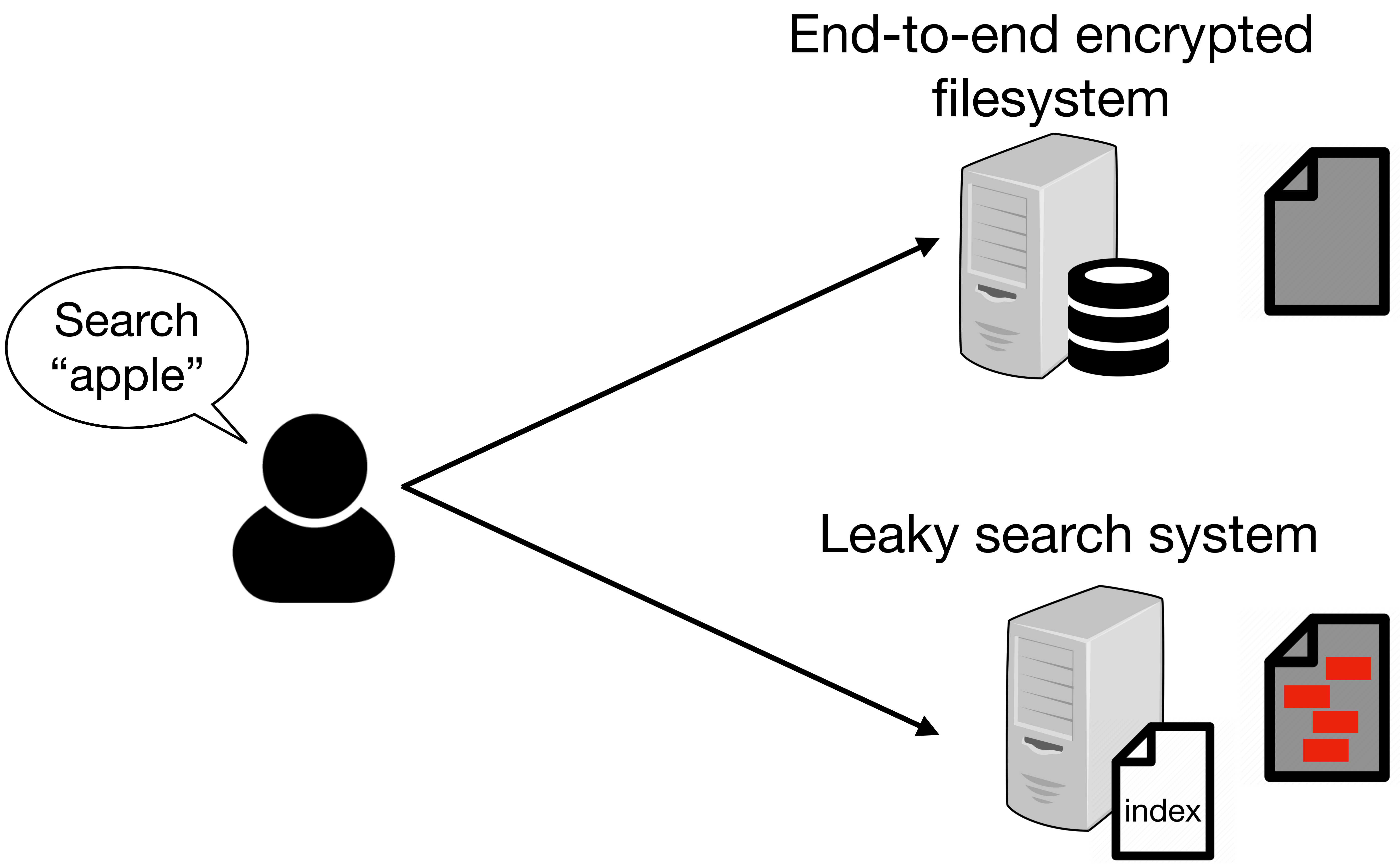
DORY


Protects search access patterns
Leaks search access patterns
ORAM-based solutions
[GO96], PathORAM, ….
Searchable Encryption (SE)
[SWP00], [Goh03], [CGKO11], [KPR12], [KP13], [CJJJ+14], [SPS14], [DPP18], …
Inefficient
Efficient

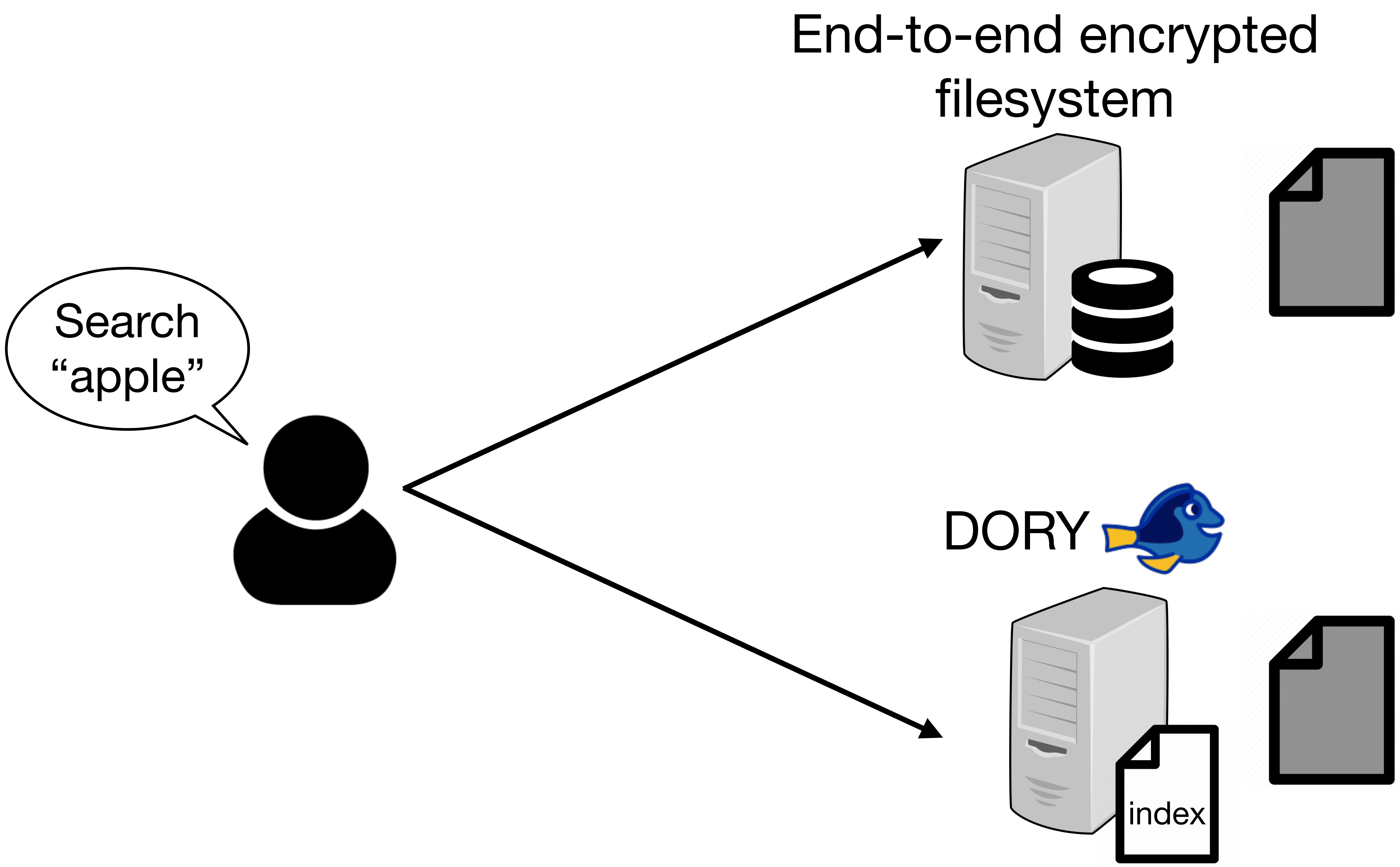
DORY


Protects search access patterns
Leaks search access patterns
ORAM-based solutions
[GO96], PathORAM, ….
DORY
Decentralized Oblivious Retrieval sYstem
Searchable Encryption (SE)
[SWP00], [Goh03], [CGKO11], [KPR12], [KP13], [CJJJ+14], [SPS14], [DPP18], …
Inefficient
Efficient

DORY eliminates search access pattern leakage



DORY eliminates search access pattern leakage



To tackle this problem, we return to the system model:

What do *real* encrypted filesystems require from a search system?

Finding DORY: Identifying a system model 

Surveyed 5 companies providing end-to-end encrypted filesystems.

Keybase PREVEIL SpiderOak sync.com tresorit

Each wanted server-side search, but didn't deploy because concerned about:

- Search access patterns
- Performance

Survey findings

See paper for full quantitative and qualitative findings.

- Requirements for **latency**, **cost**, and **concurrency**.

Survey findings

See paper for full quantitative and qualitative findings.

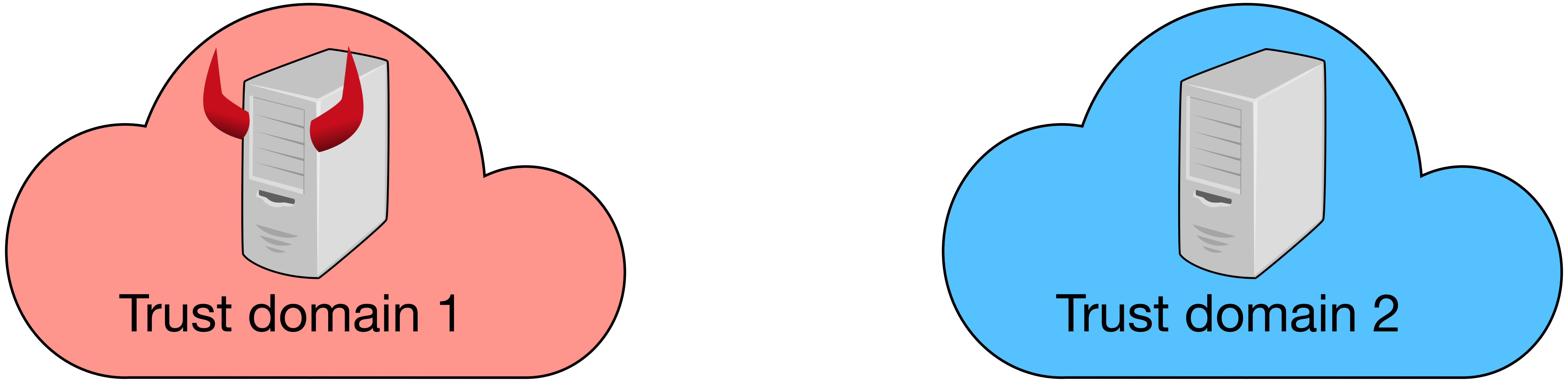
- Requirements for **latency**, **cost**, and **concurrency**.

Two most relevant findings:

1. **Linear scan for search is acceptable** if search latency and cost meet requirements for expected workloads.
2. **Distributing trust is acceptable** if certain security requirements are met.

Distributed trust

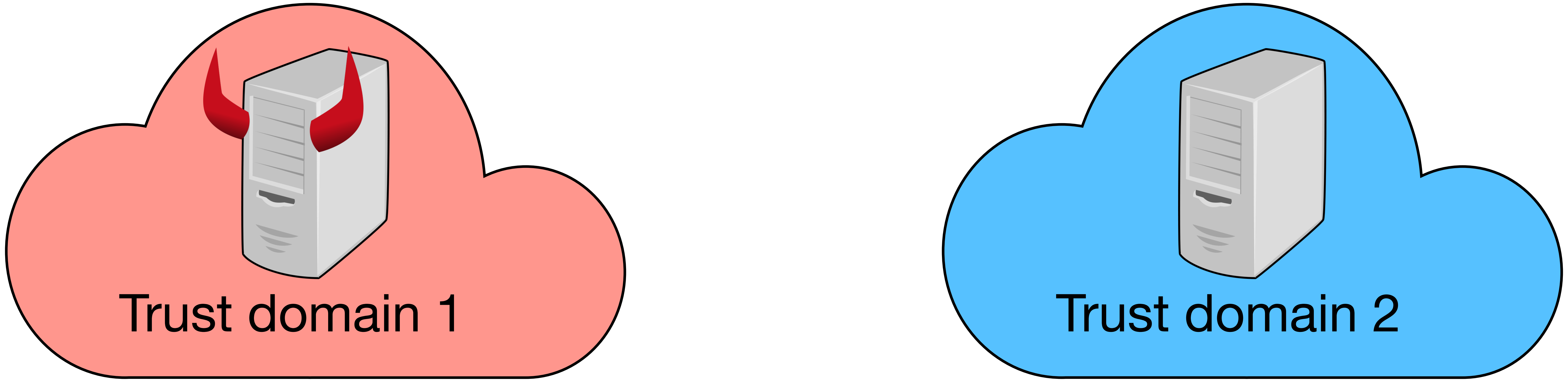
Provide security guarantees if an attacker can compromise some, but not all, trust domains.



Distributed trust requirements

At least one honest trust domain: attacker can't learn search access patterns.

- The other trust domains can be malicious.



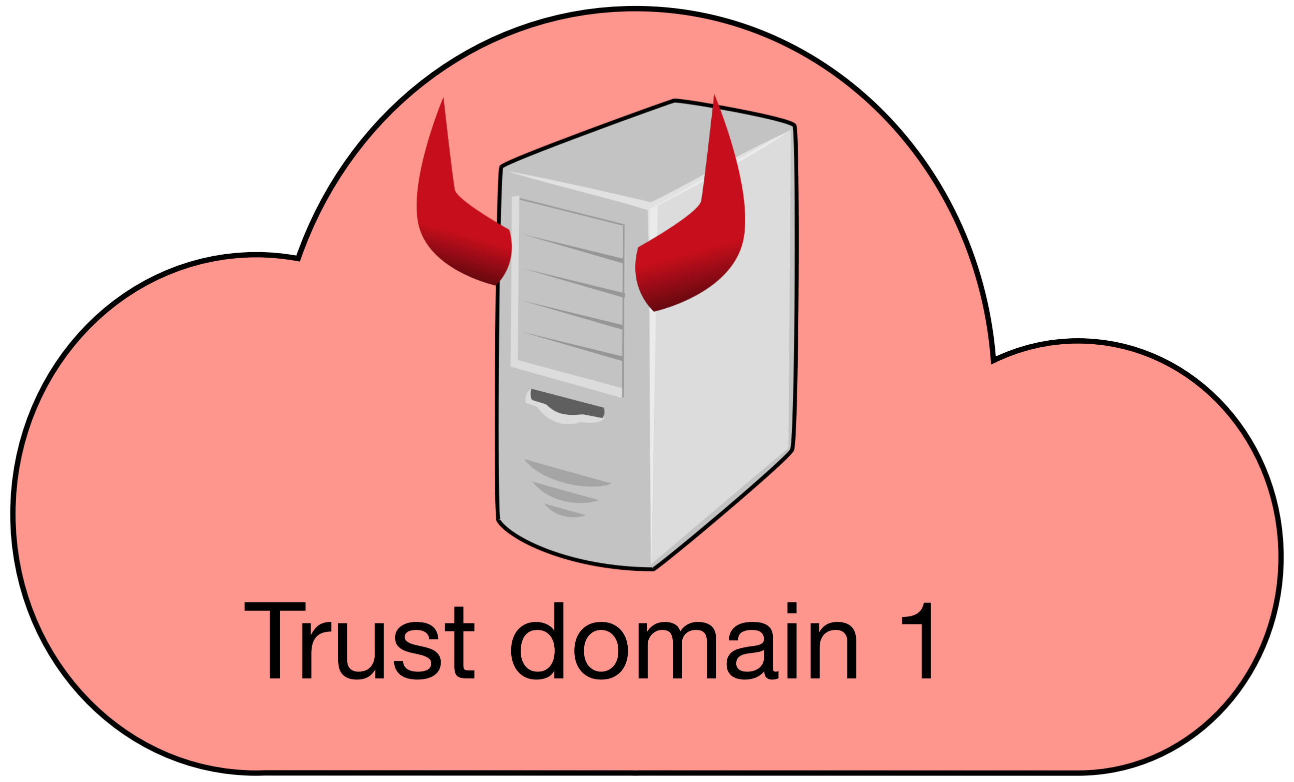
Distributed trust requirements

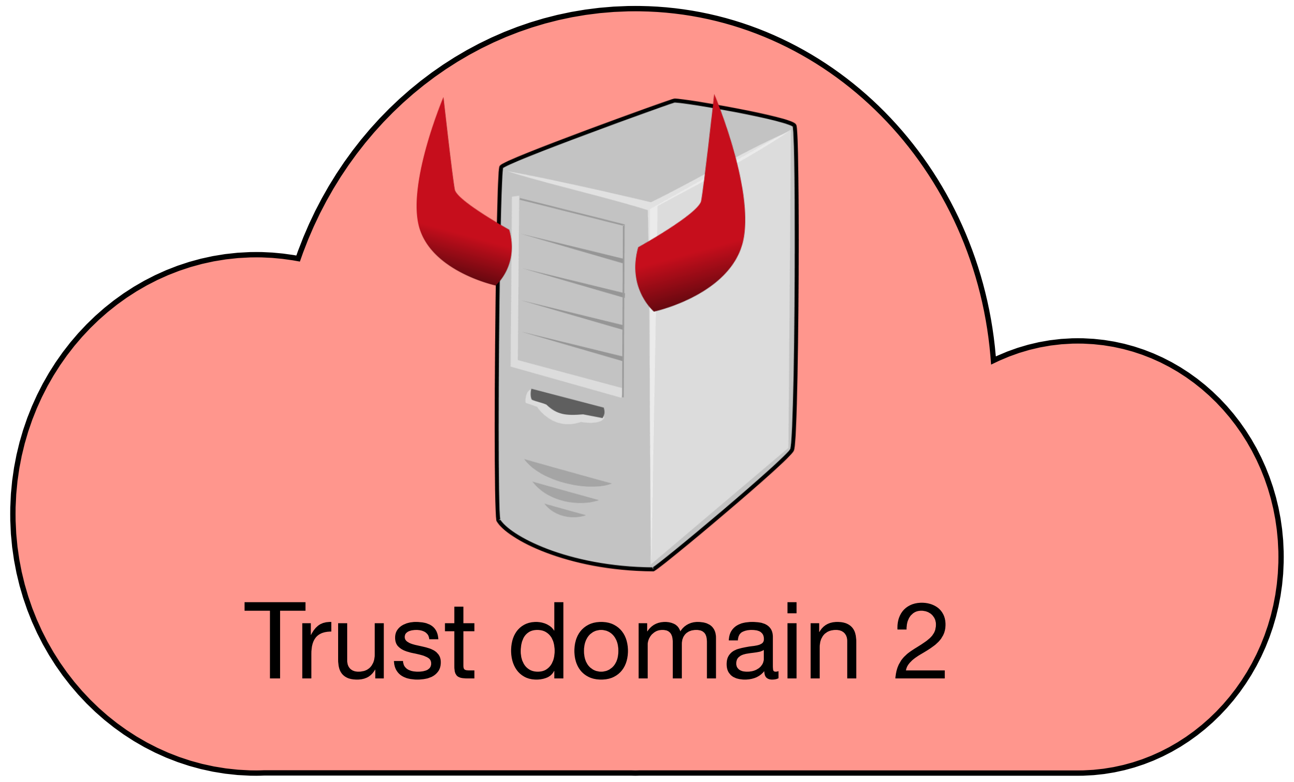
At least one honest trust domain: attacker can't learn search access patterns.

- The other trust domains can be malicious.

No honest trust domains: attacker can’t directly assemble search index.

- Search access patterns are not protected.



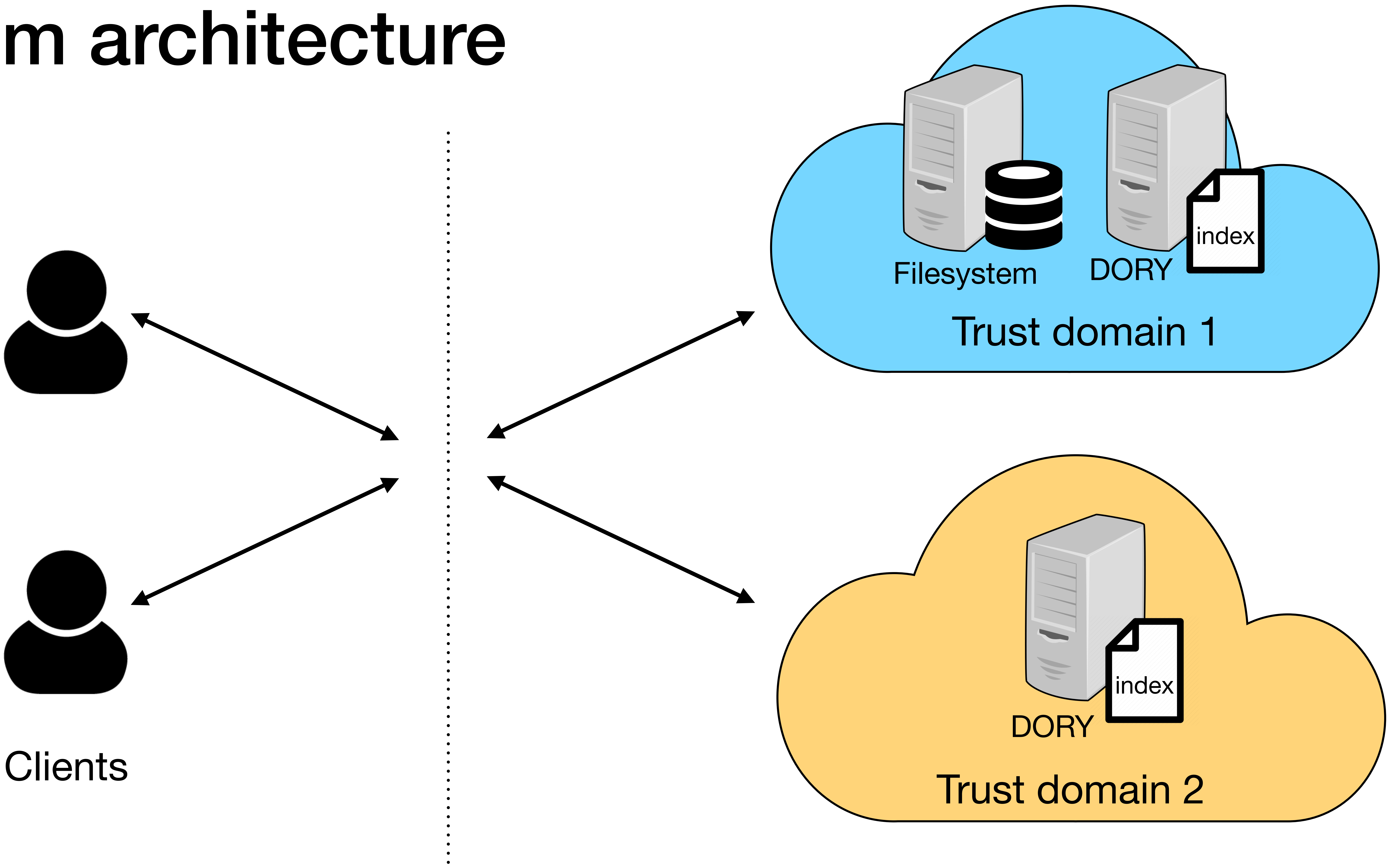


Outline

1. **DORY design**

2. DORY evaluation

System architecture



[Simplified; does not account for replication]

Building DORY

Search index [simplified]

$$
\begin{array}{llllll}
\text{Doc 0} & x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\
\text{Doc 1} & x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\
\text{Doc 2} & x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\
 & \vdots & \vdots & \vdots & & \vdots \\
\text{Doc } n & x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m}
\end{array}
$$

Search index [simplified]

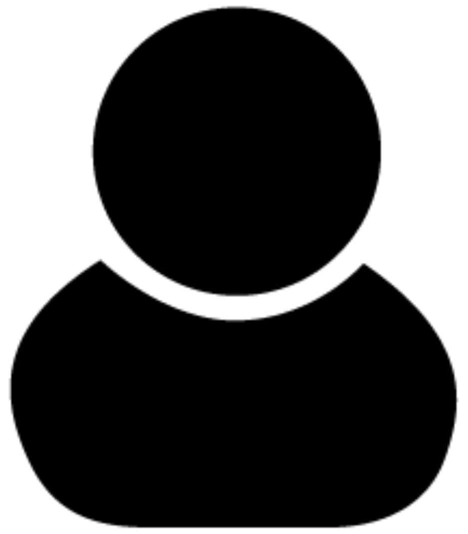
$$
\begin{array}{lccccc}
\text{Doc } 0 & x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\
\text{Doc } 1 & x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\
\text{Doc } 2 & x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\
 & \vdots & \vdots & \vdots & & \vdots \\
\text{Doc } n & x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m}
\end{array}
$$

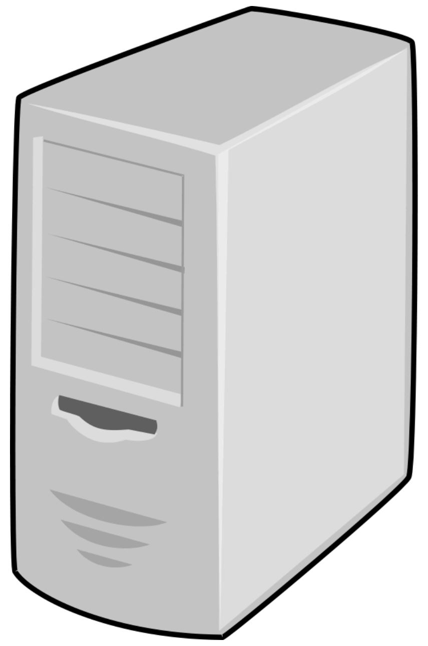
Bitmap for keywords in doc 1

Update [simplified]

`update(docID, keywords)`

- Client creates a bitmap for `keywords`.
- Client sends server the bitmap.
- Server updates the bitmap at row `docID`.



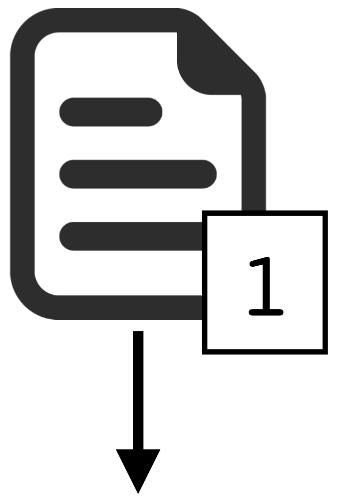


$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

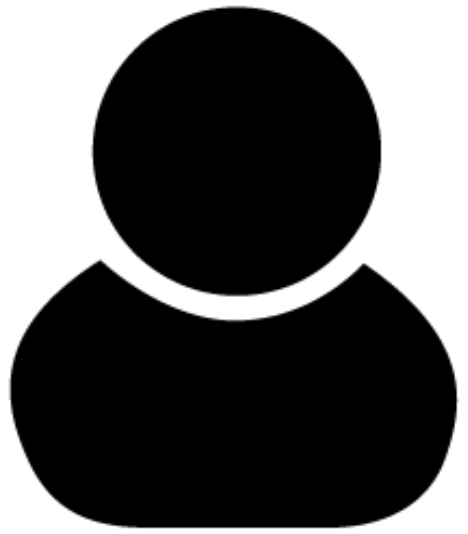
Update [simplified]

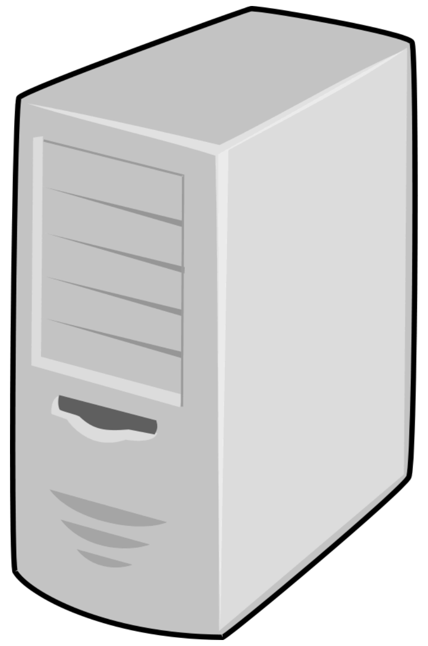
`update(docID, keywords)`

- Client creates a bitmap for `keywords`.
- Client sends server the bitmap.
- Server updates the bitmap at row `docID`.



$$x'_0 \; x'_1 \; x'_2 \; \ldots \; x'_m$$



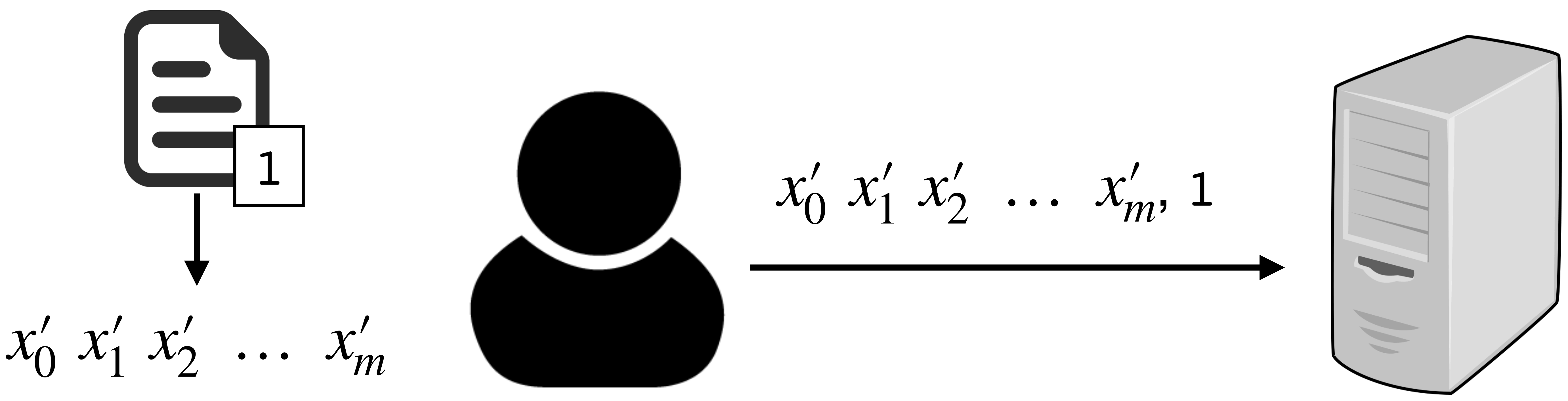


$$\begin{array}{ccccc} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{array}$$

Update [simplified]

`update(docID, keywords)`

- Client creates a bitmap for `keywords`.
- Client sends server the bitmap.
- Server updates the bitmap at row `docID`.

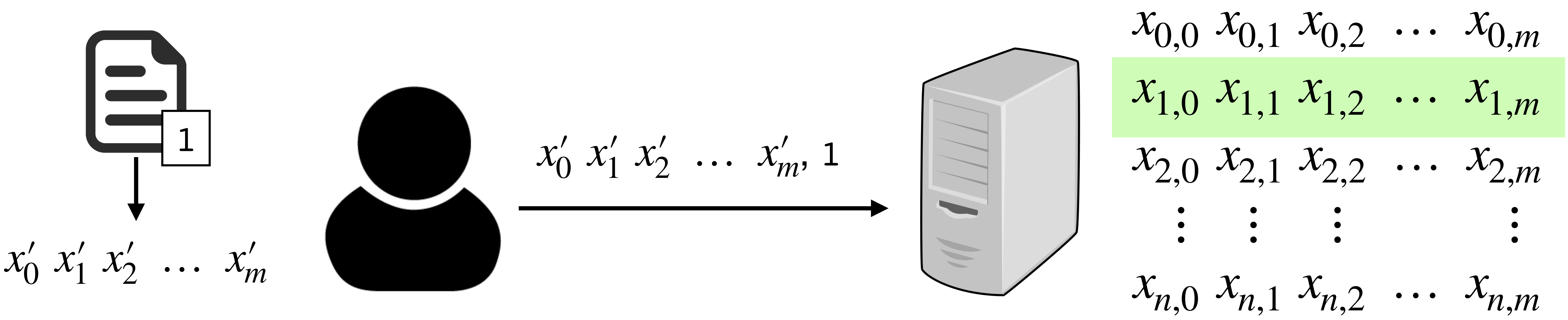


$$
\begin{matrix}
x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\
x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\
x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\
\vdots & \vdots & \vdots & & \vdots \\
x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m}
\end{matrix}
$$

Update [simplified]

`update(docID, keywords)`

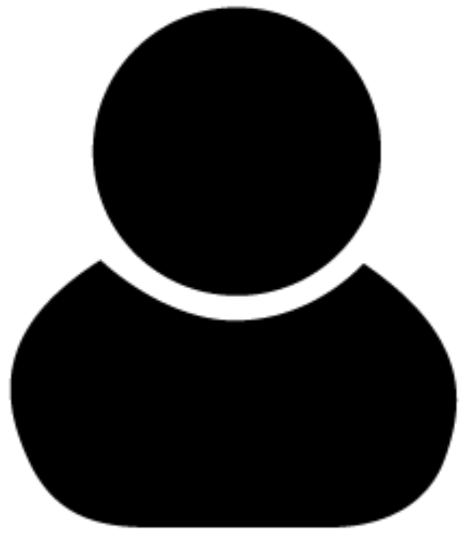
- Client creates a bitmap for `keywords`.
- Client sends server the bitmap.
- Server updates the bitmap at row `docID`.

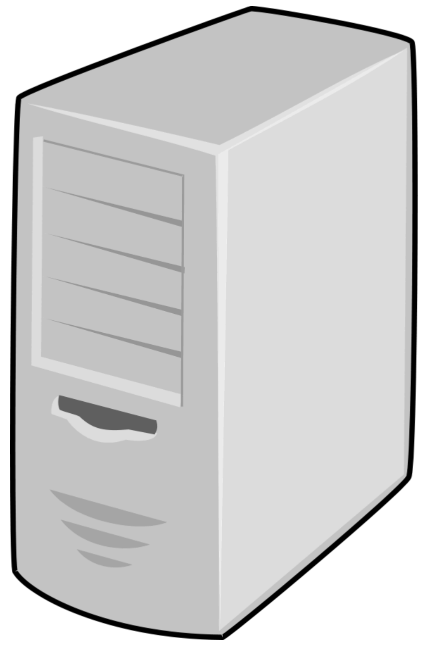


Search [simplified]

`search(keyword):`

- Client computes the index for `keyword` and sends to server.
- Server responds with corresponding column.
- Client outputs row numbers where column value is 1.



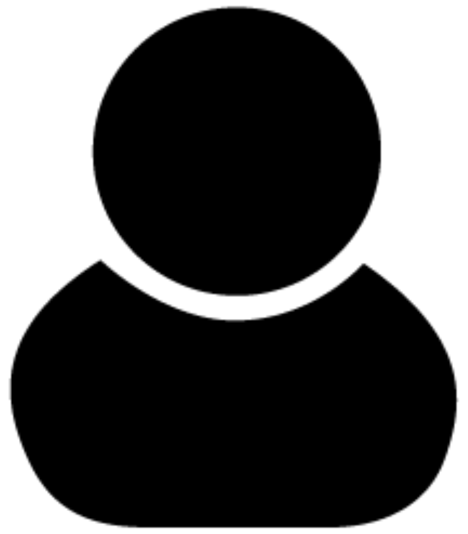


$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

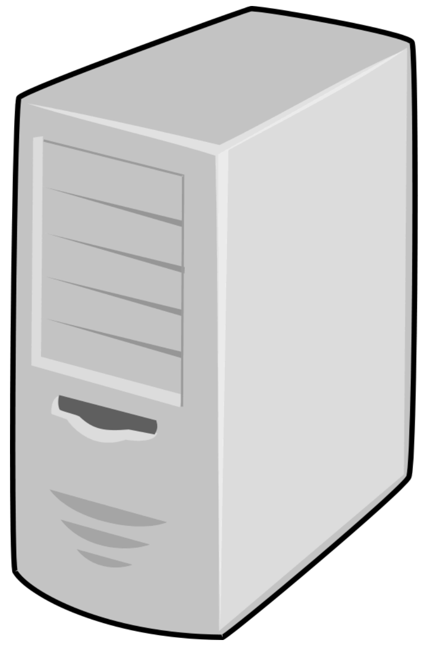
Search [simplified]

`search(keyword):`

- Client computes the index for `keyword` and sends to server.
- Server responds with corresponding column.
- Client outputs row numbers where column value is 1.



`GetIndex(keyword)` $\longrightarrow$ `2`

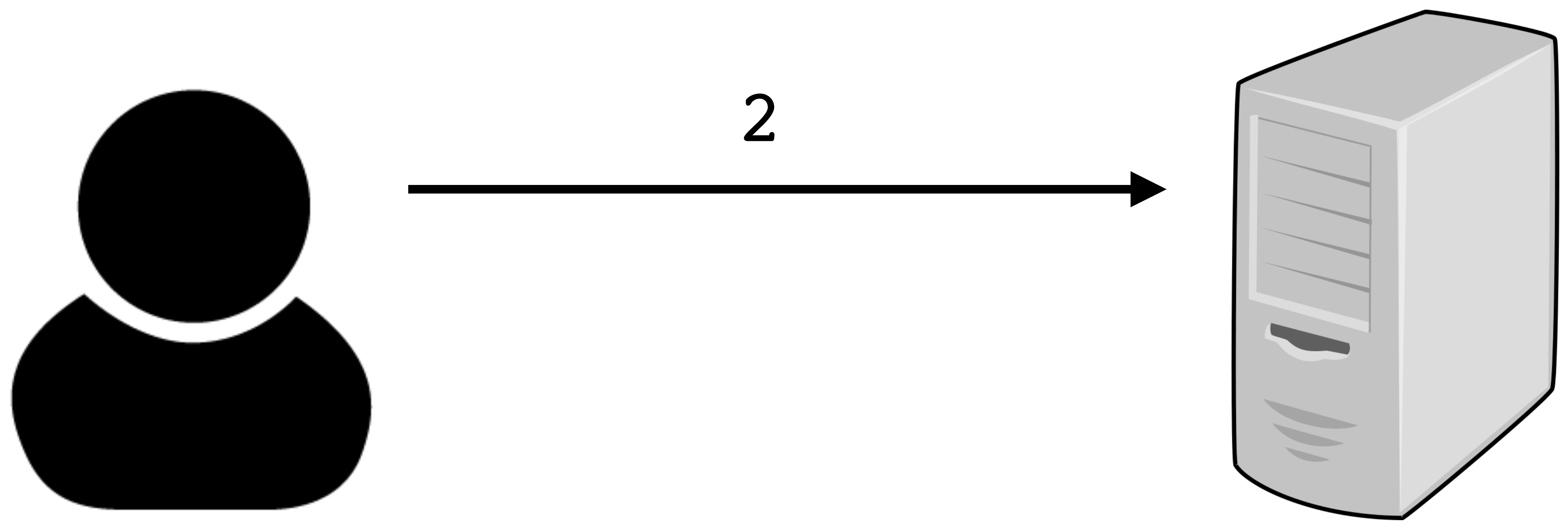


$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

Search [simplified]

`search(keyword):`

- Client computes the index for `keyword` and sends to server.
- Server responds with corresponding column.
- Client outputs row numbers where column value is 1.



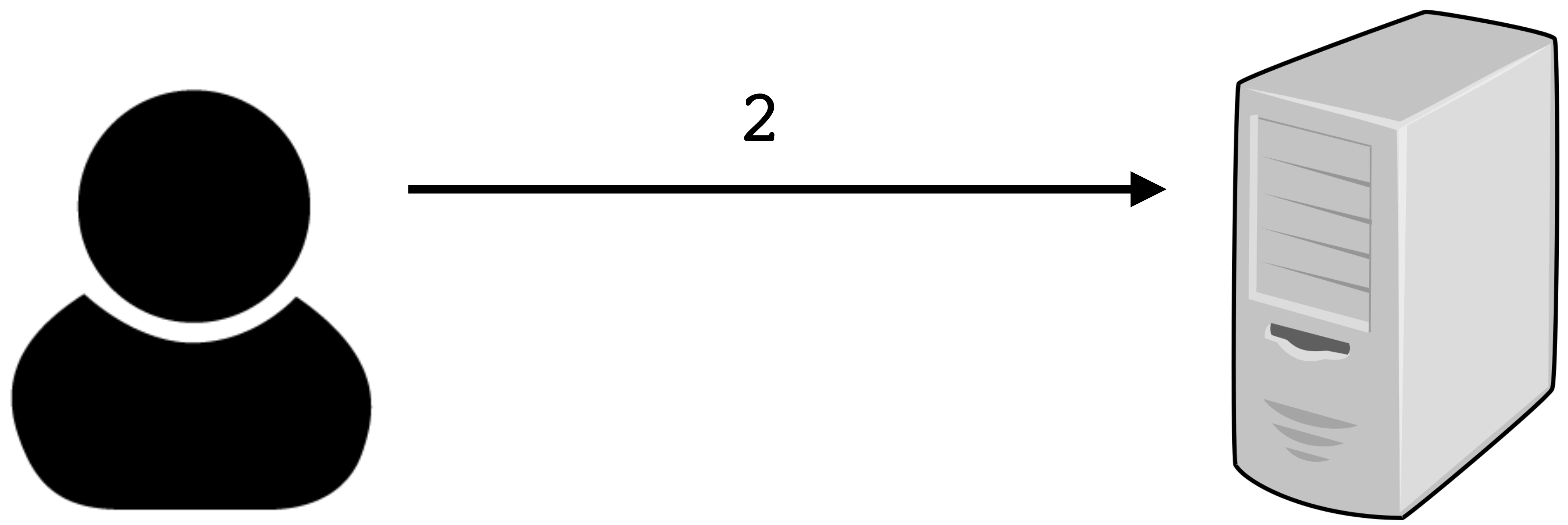
$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

`GetIndex(keyword)` $\longrightarrow$ `2`

Search [simplified]

`search(keyword):`

- Client computes the index for `keyword` and sends to server.
- Server responds with corresponding column.
- Client outputs row numbers where column value is 1.



$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

`GetIndex(keyword)` $\longrightarrow$ `2`

Search [simplified]

`search(keyword):`

- Client computes the index for `keyword` and sends to server.
- Server responds with corresponding column.
- Client outputs row numbers where column value is 1.

$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

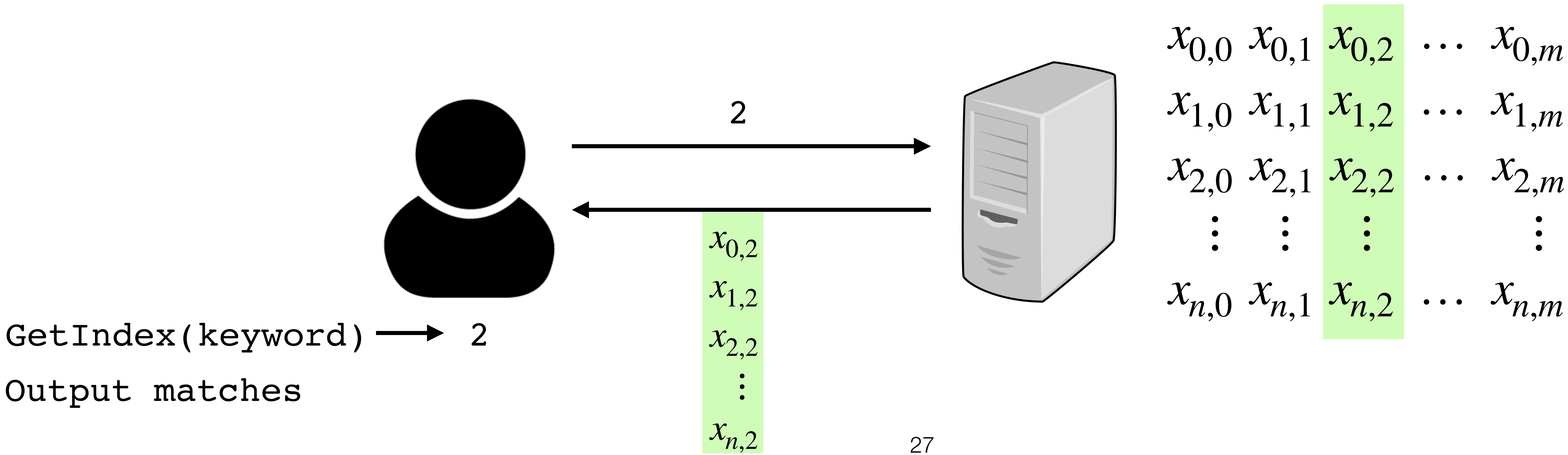
$$\begin{matrix} x_{0,2} \\ x_{1,2} \\ x_{2,2} \\ \vdots \\ x_{n,2} \end{matrix}$$

`GetIndex(keyword)` → `2`

Search [simplified]

`search(keyword):`

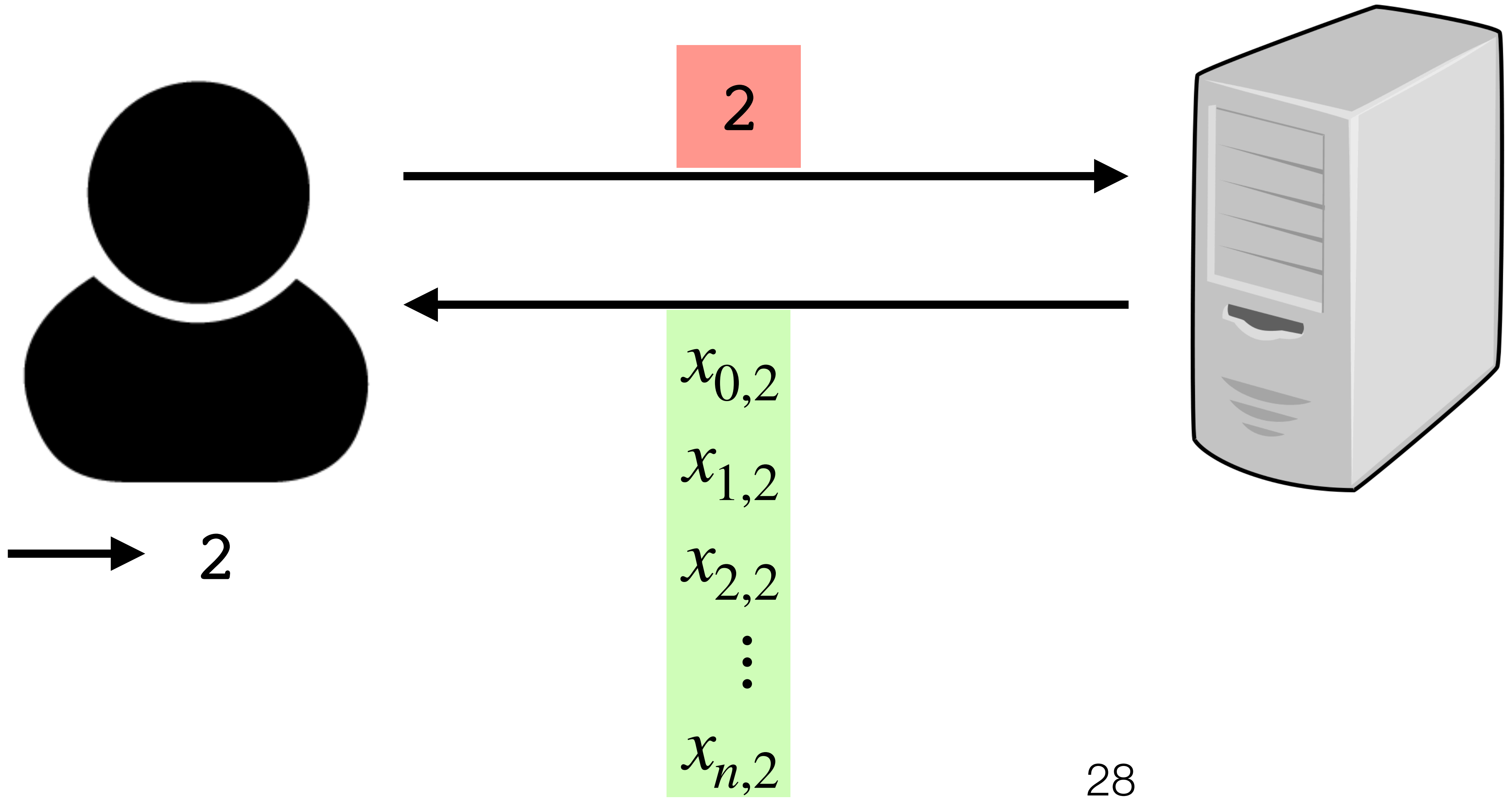
- Client computes the index for `keyword` and sends to server.
- Server responds with corresponding column.
- Client outputs row numbers where column value is 1.



Challenge #1: Hiding search access patterns

Attacker learns search access patterns.

- Column requested leak data about keyword searched for.

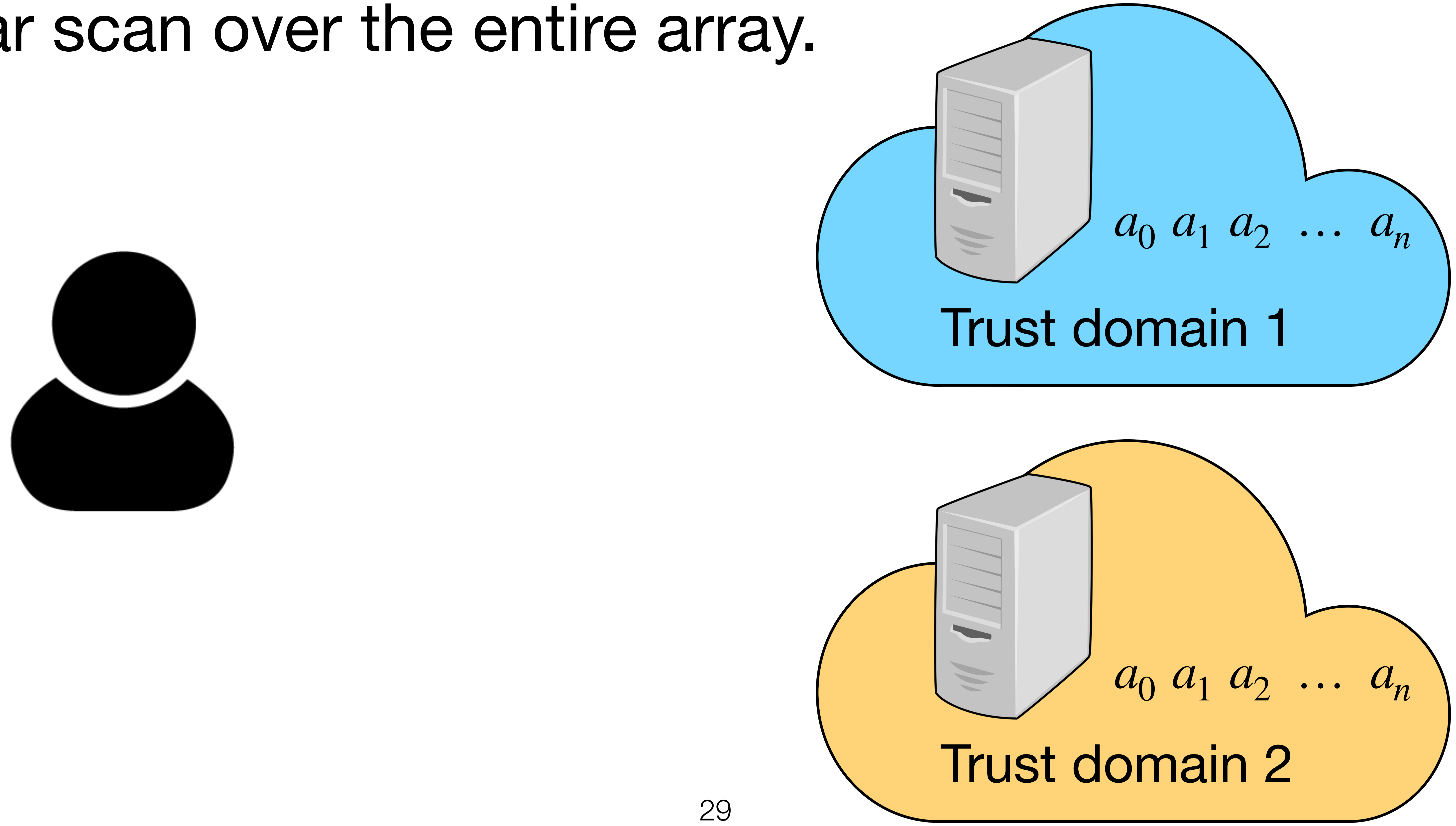


$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

```
GetIndex(keyword) ⟶ 2
Output matches
```

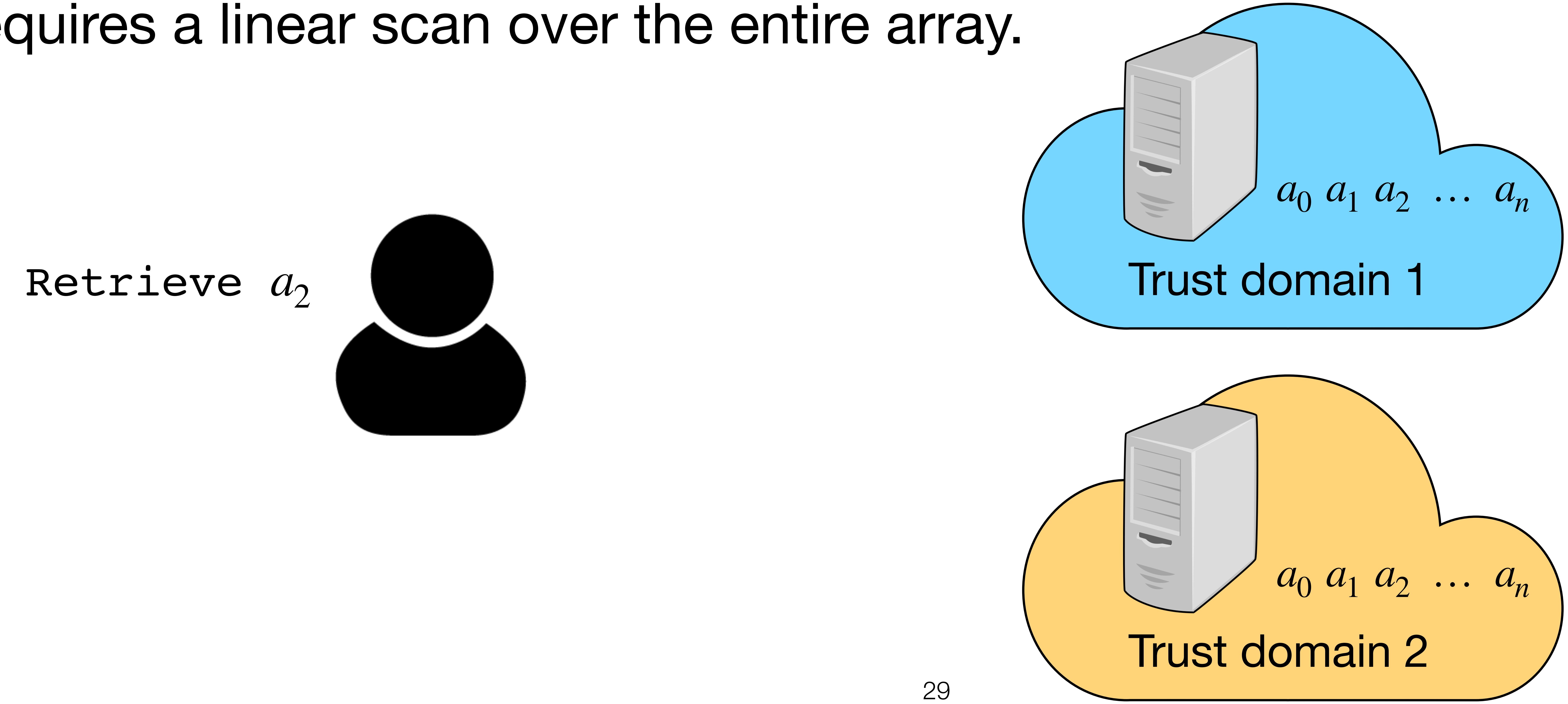
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



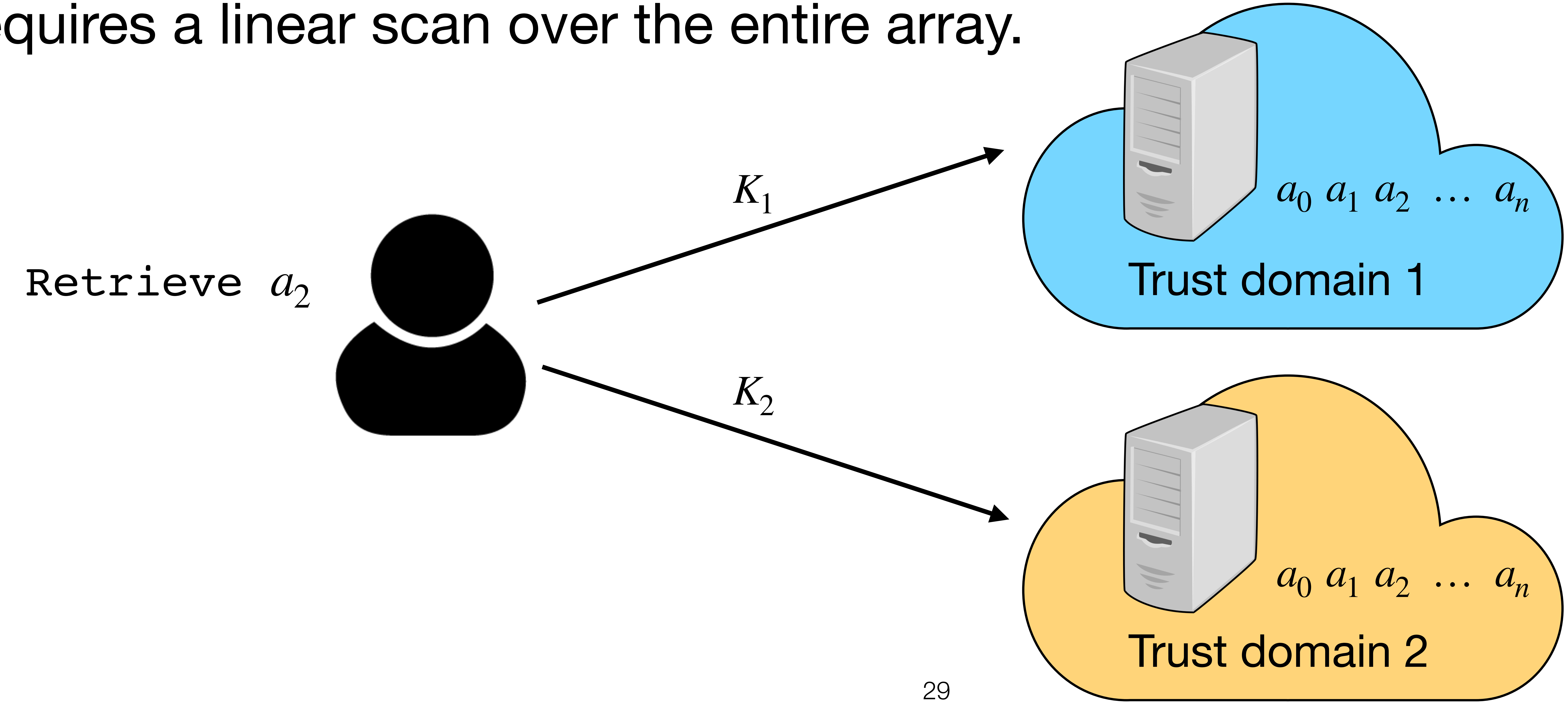
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



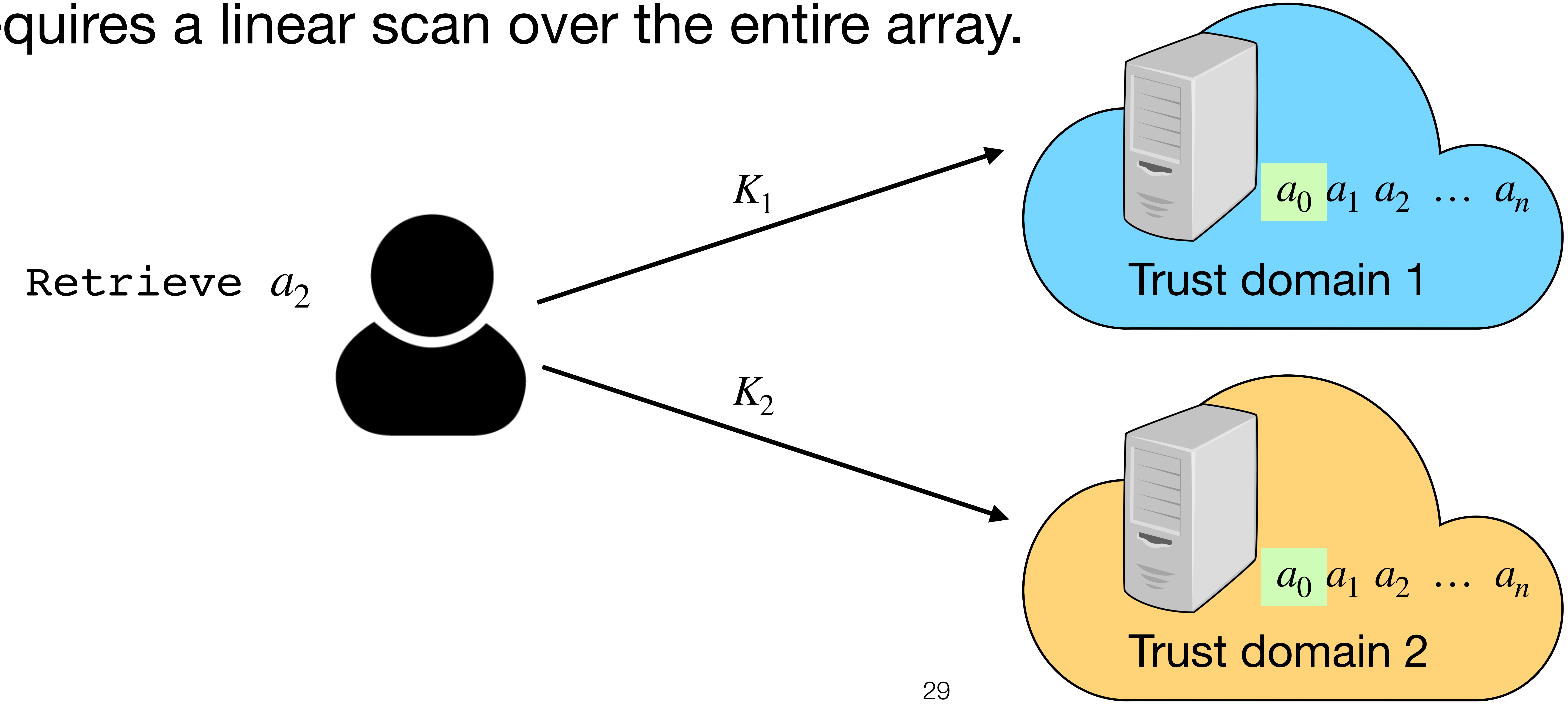
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



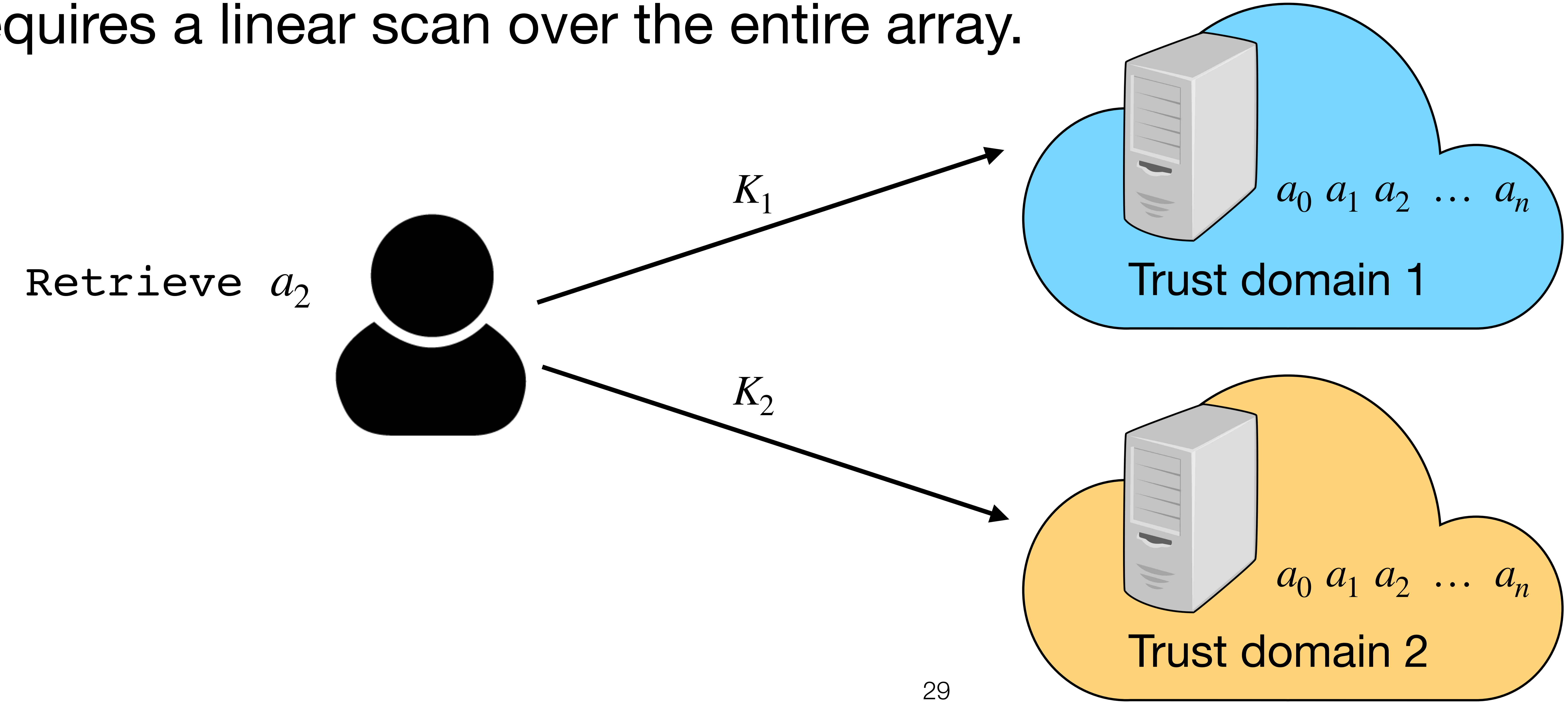
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



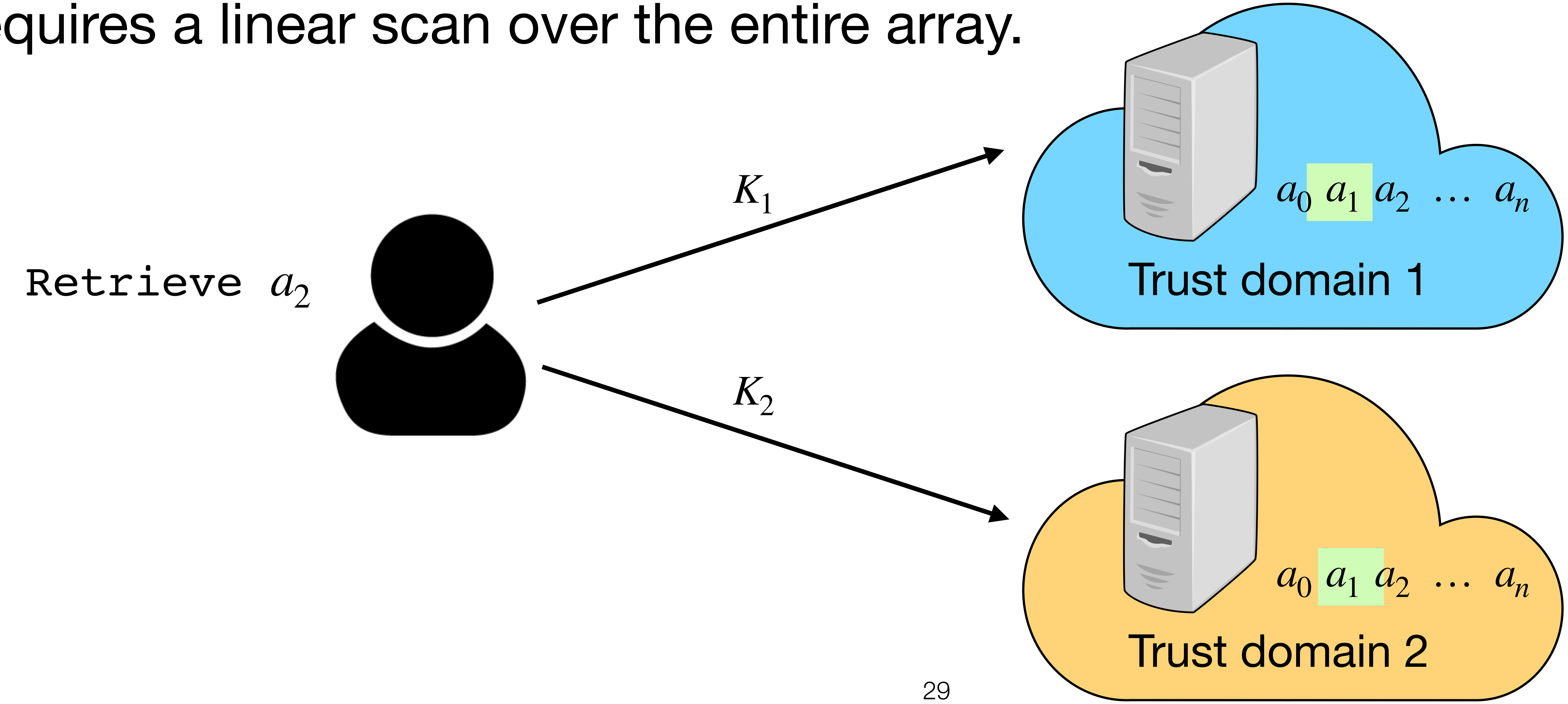
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



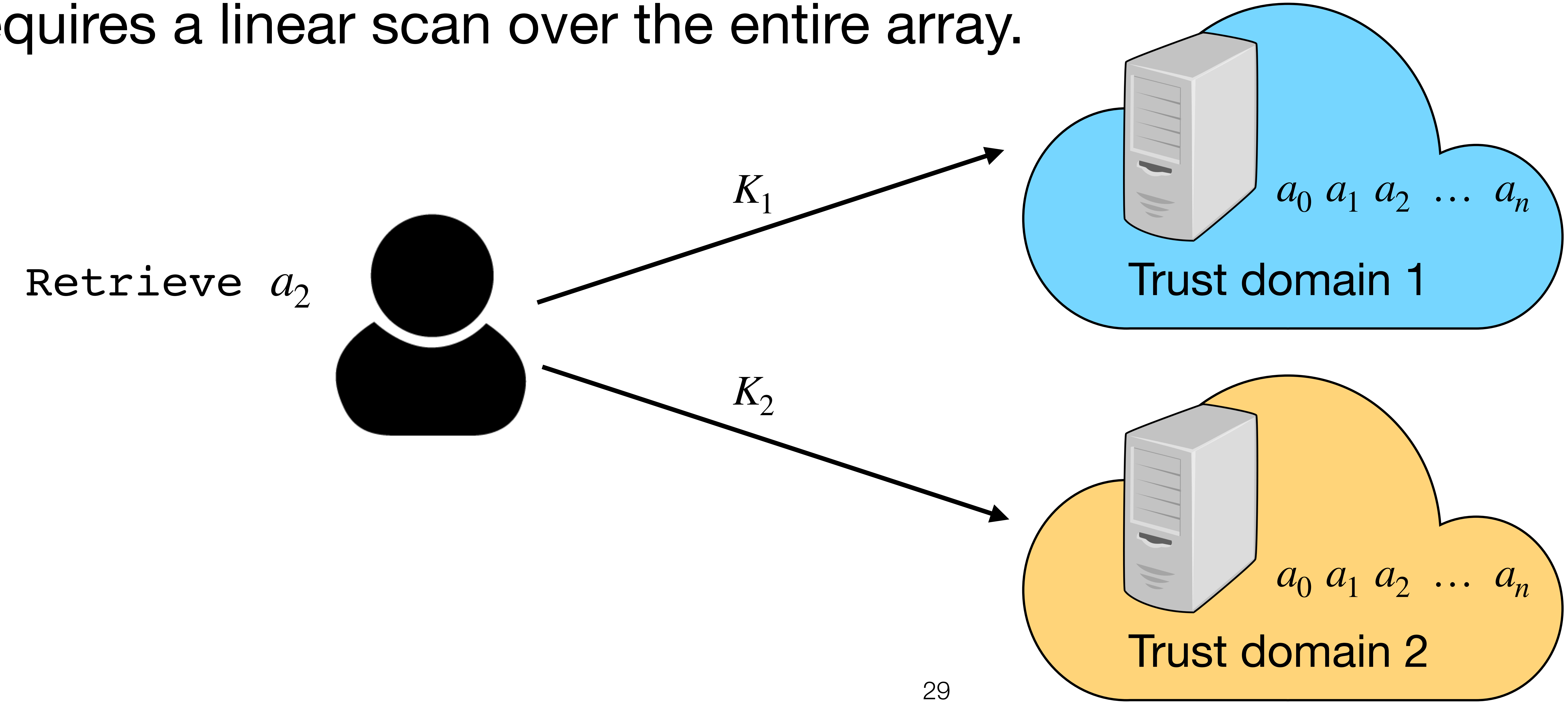
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



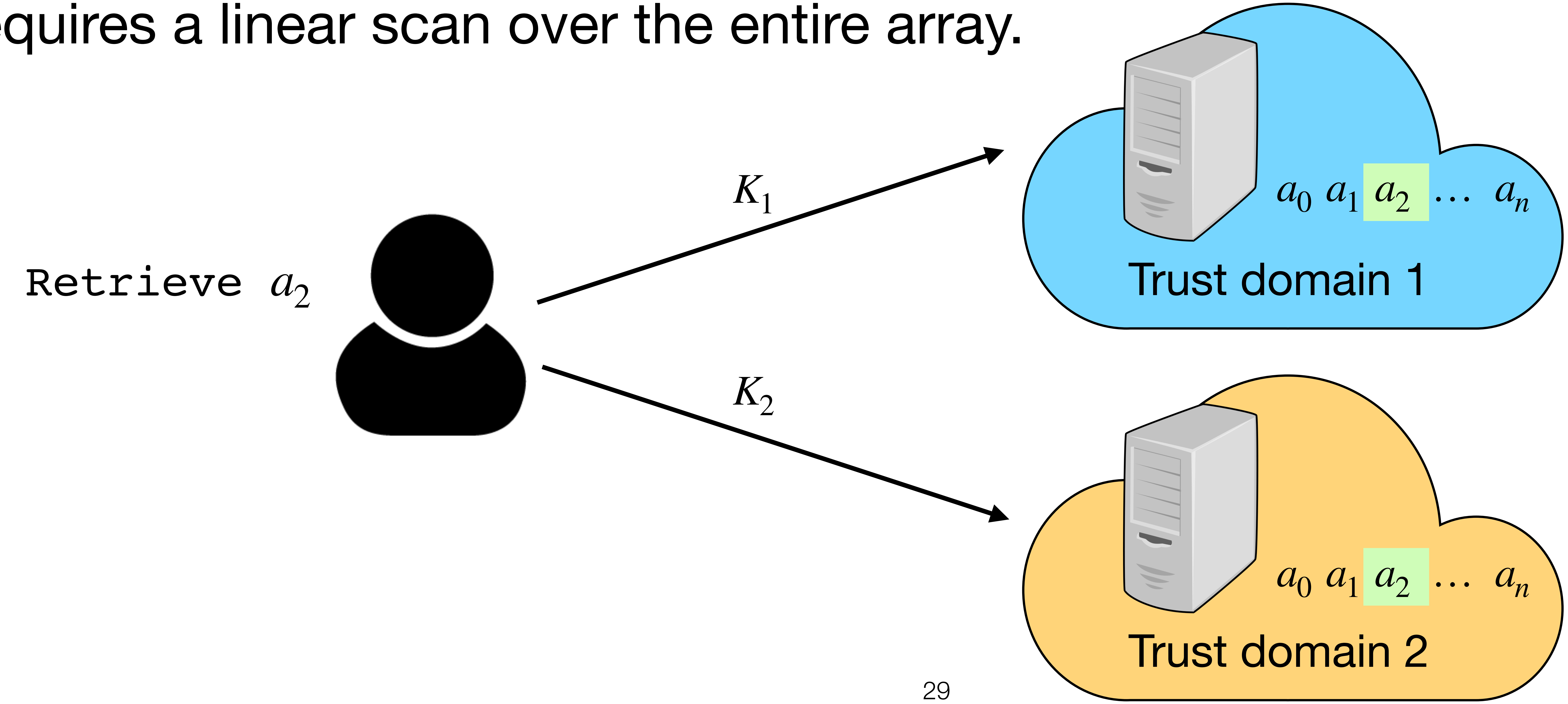
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



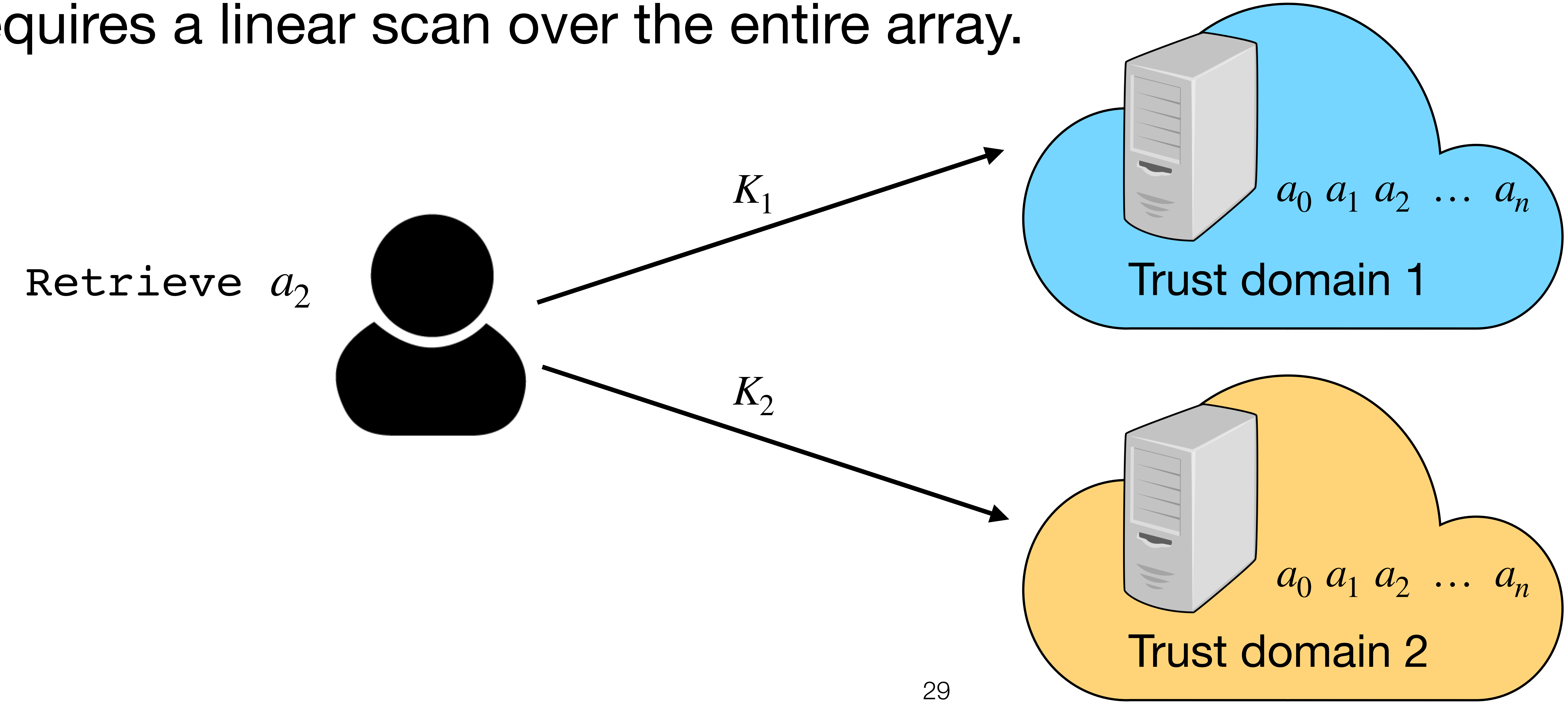
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



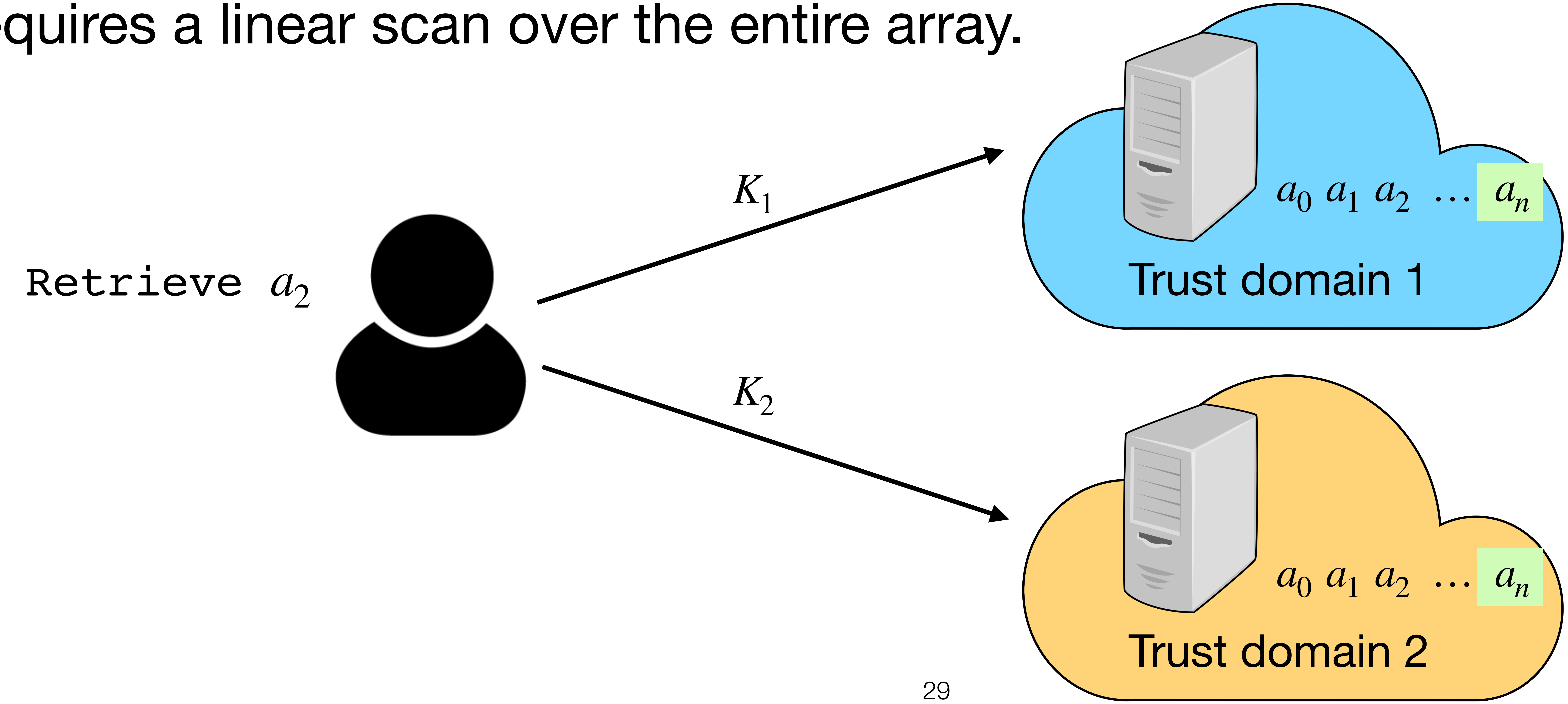
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



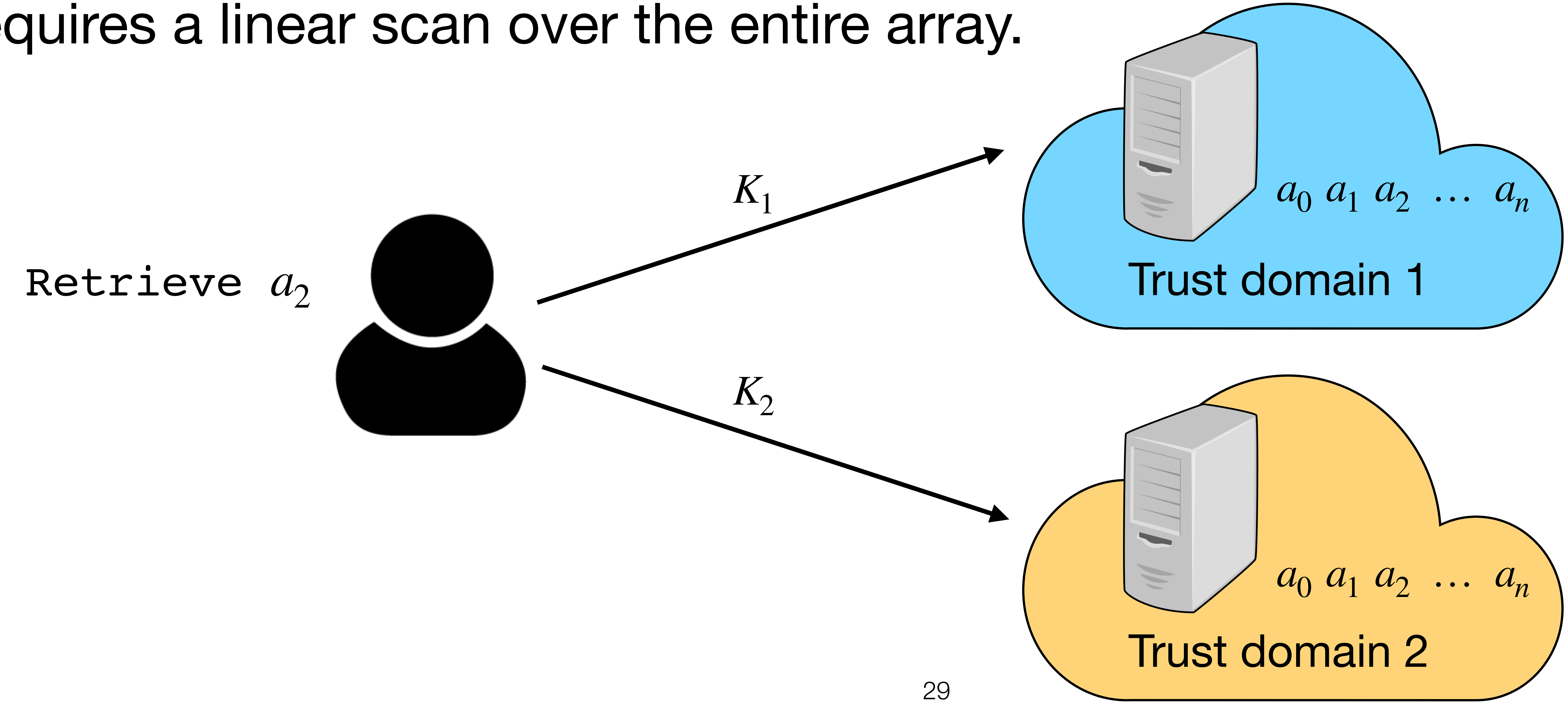
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



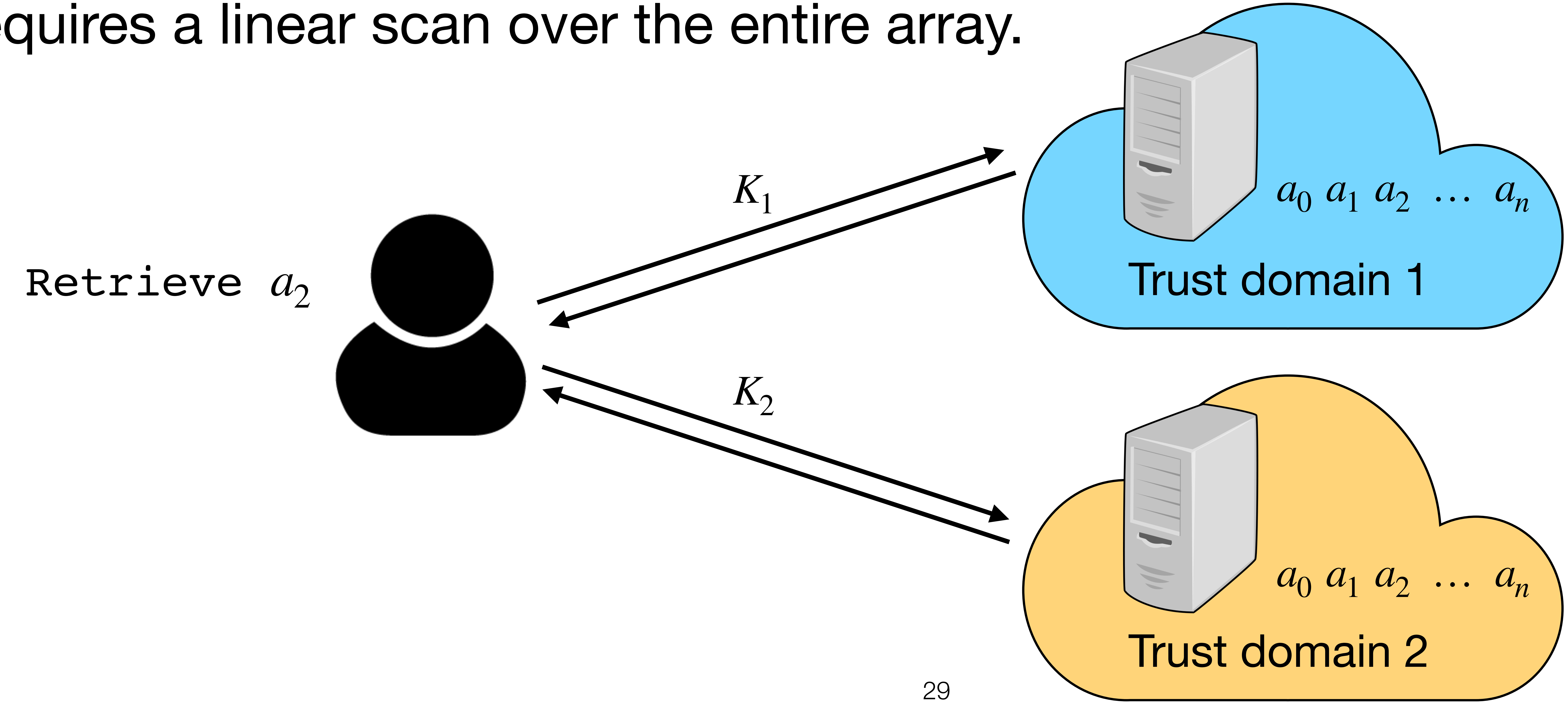
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



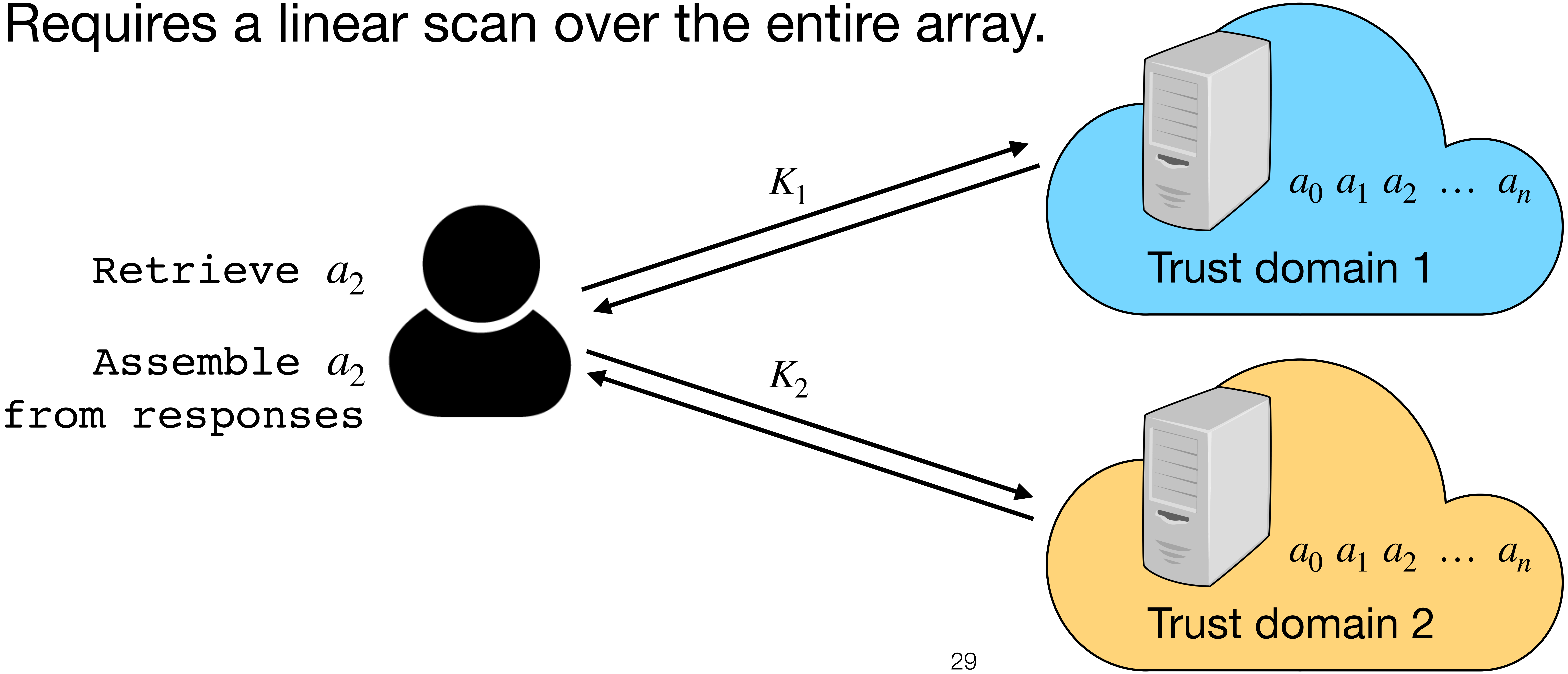
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



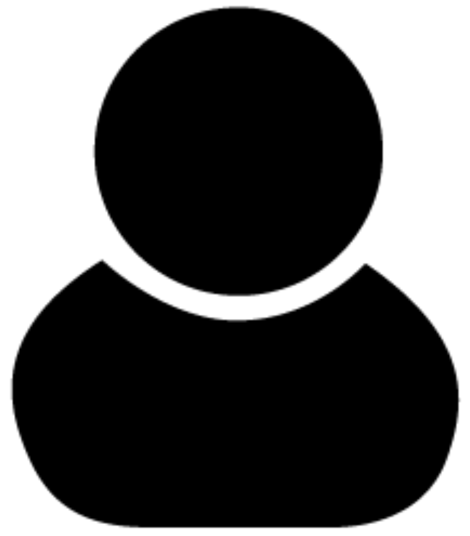
Tool: Distributed Point Functions (DPFs) [GI14, BGI15, BGI16]

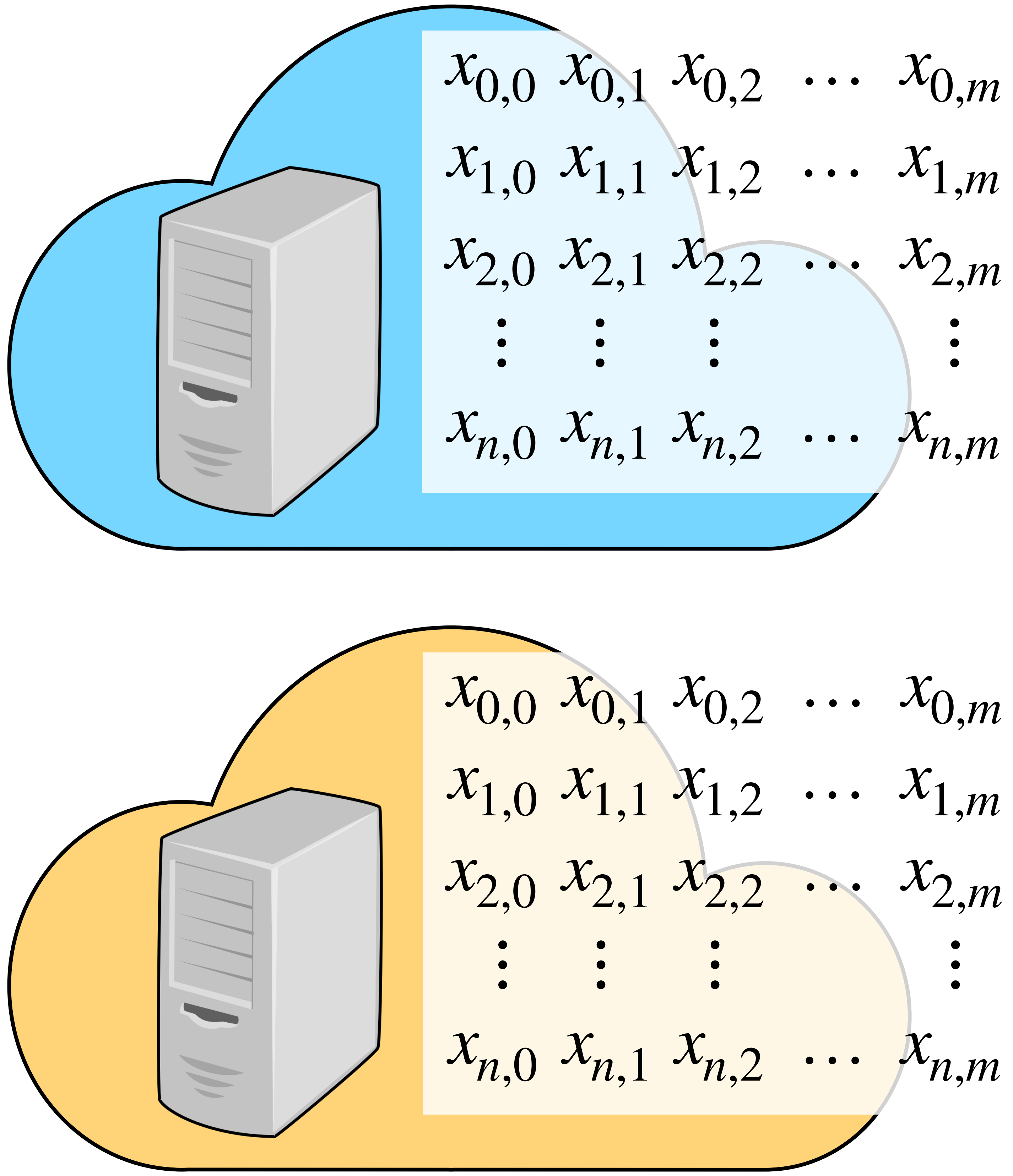
- Uses multiple servers to hide which element the user is retrieving.
- If at least one server is honest, an attacker cannot learn the index requested.
- Requires a linear scan over the entire array.



Leveraging DPFs to search

If at least one trust domain is honest, DORY hides search access patterns

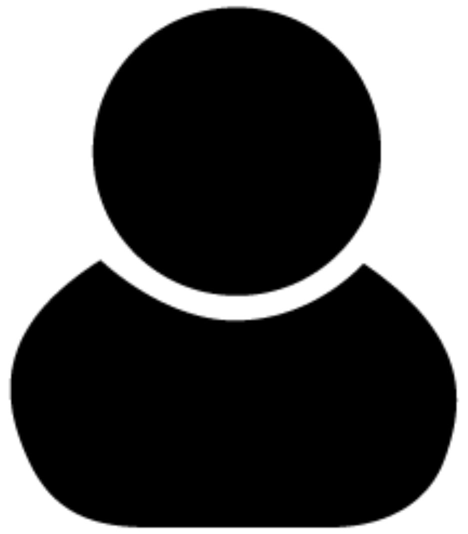


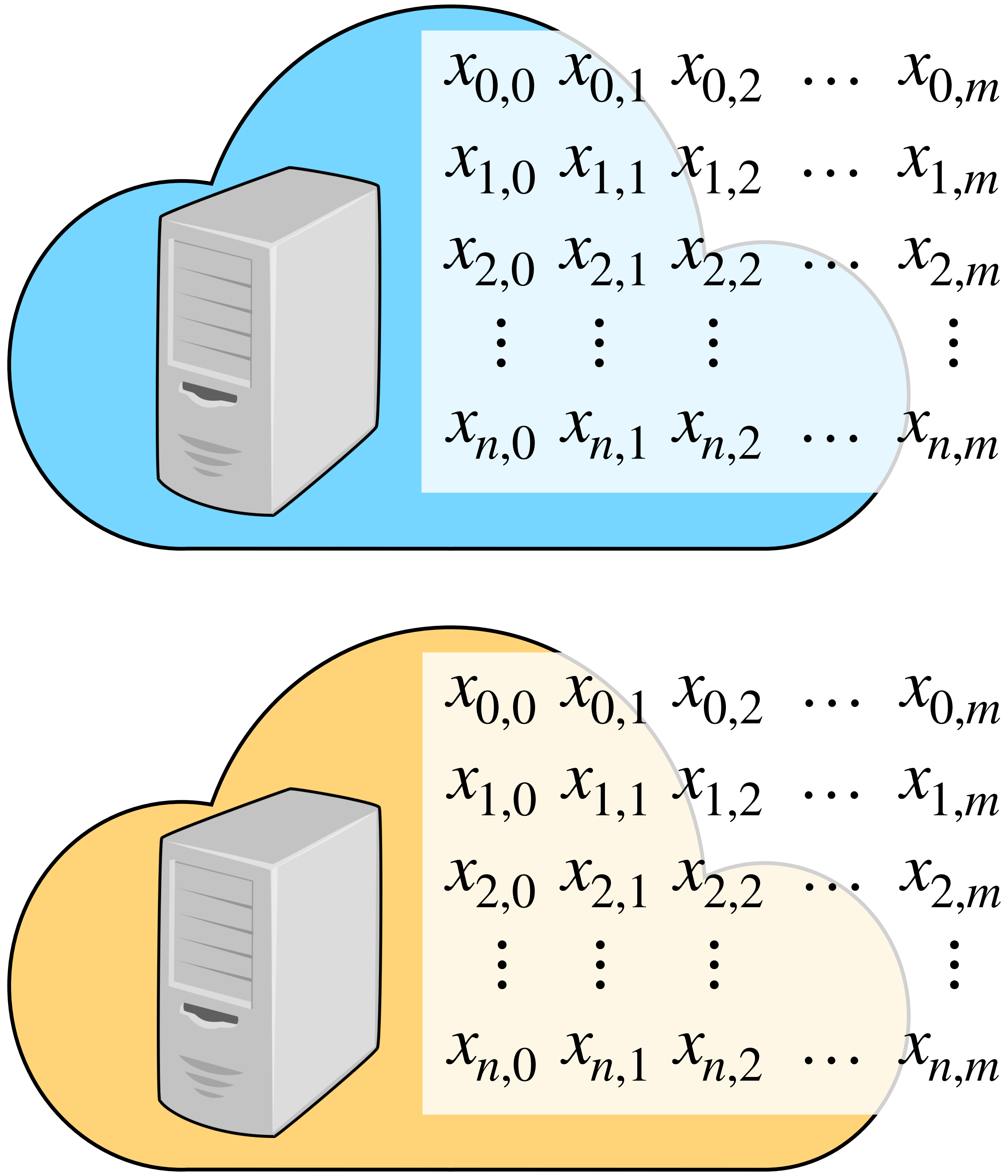


Leveraging DPFs to search

If at least one trust domain is honest, DORY hides search access patterns

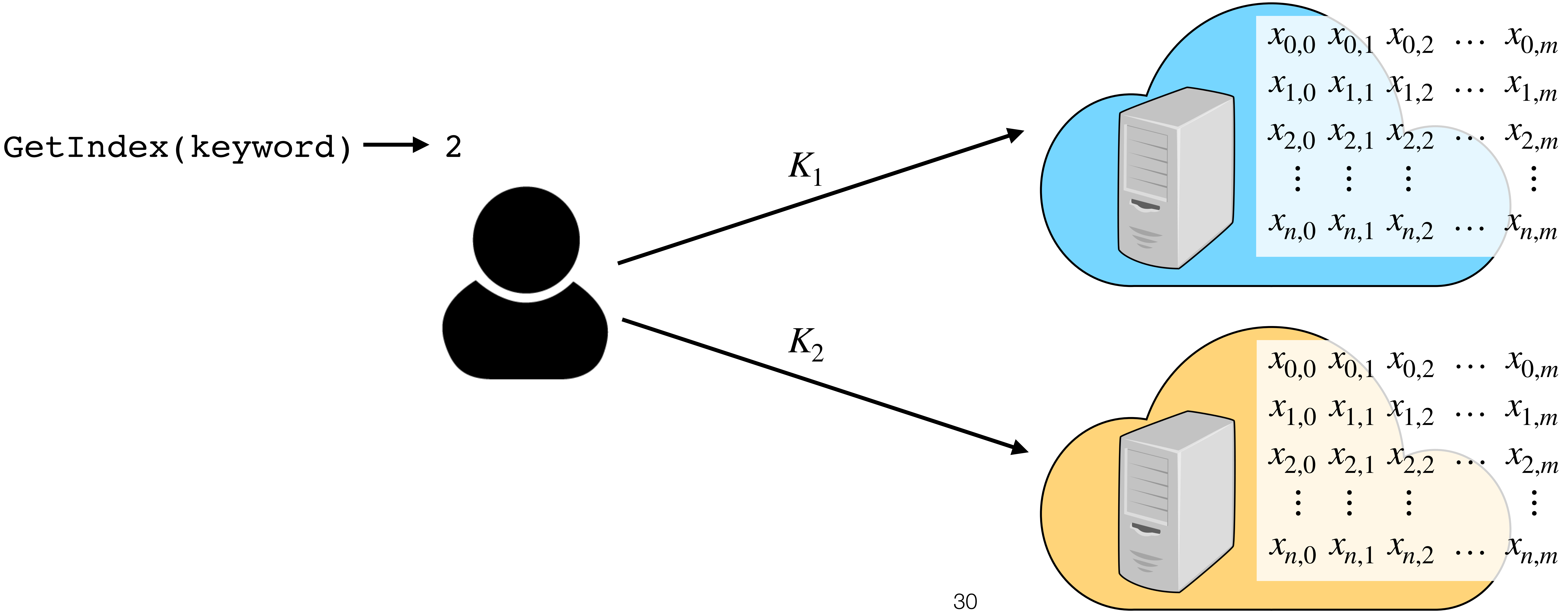
`GetIndex(keyword)` $\longrightarrow$ 2





Leveraging DPFs to search

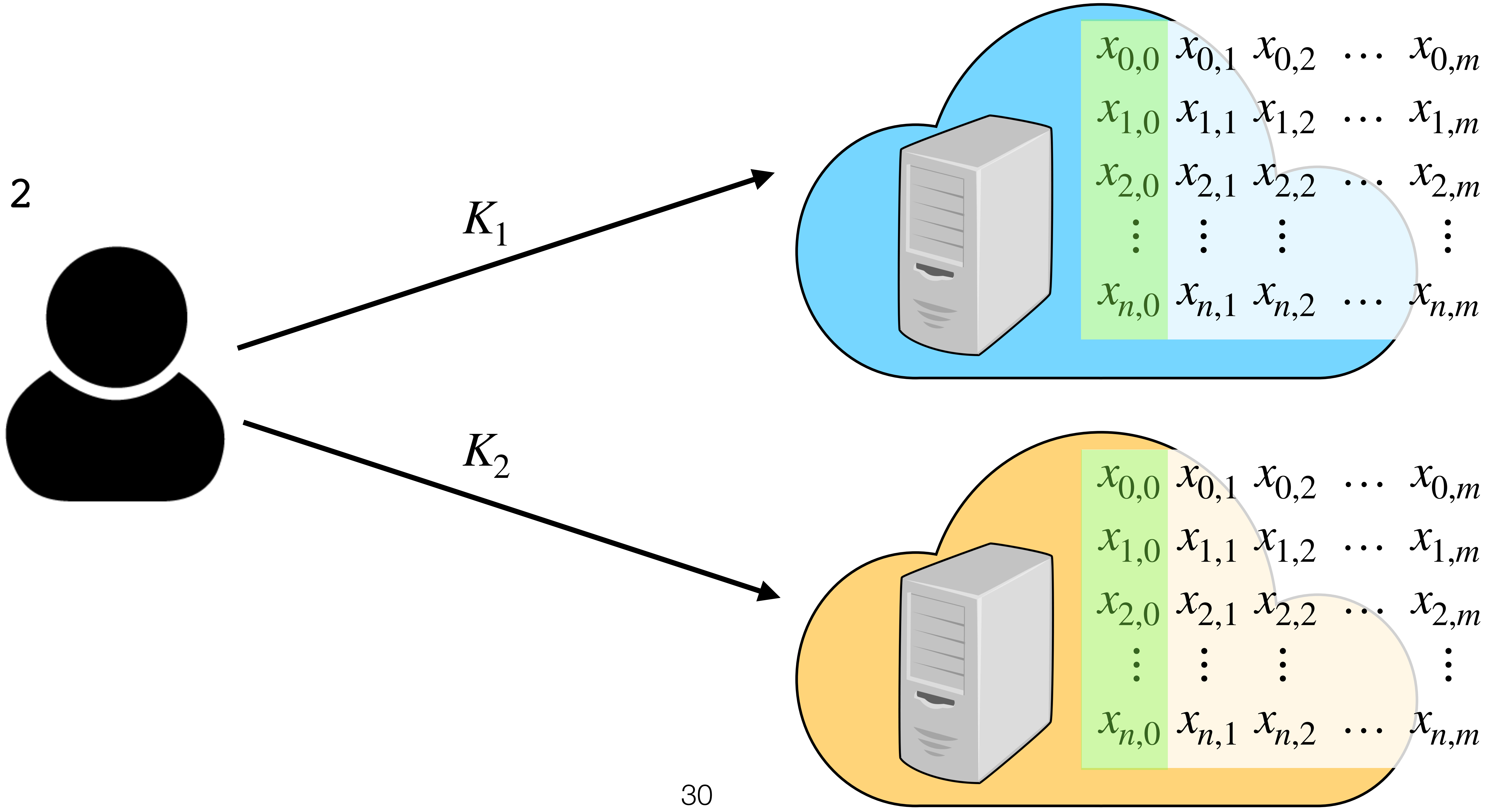
If at least one trust domain is honest, DORY hides search access patterns



Leveraging DPFs to search

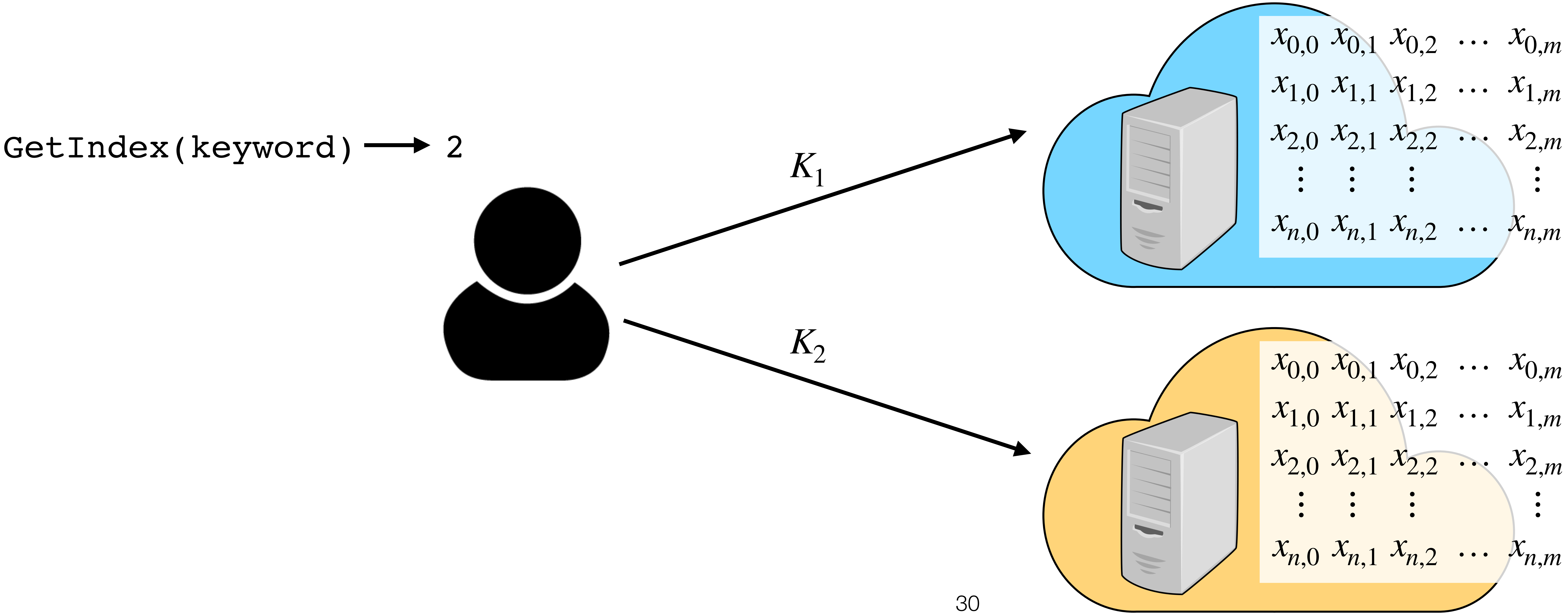
If at least one trust domain is honest, DORY hides search access patterns

`GetIndex(keyword)` $\longrightarrow$ 2



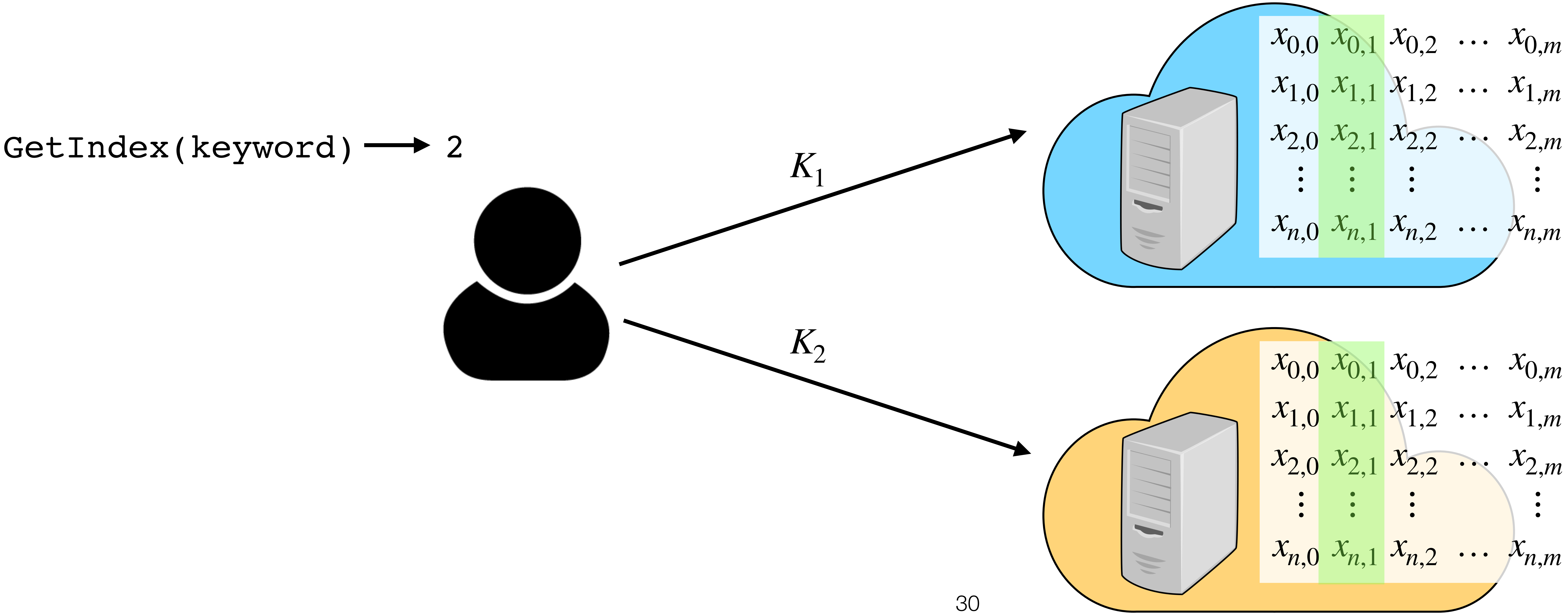
Leveraging DPFs to search

If at least one trust domain is honest, DORY hides search access patterns



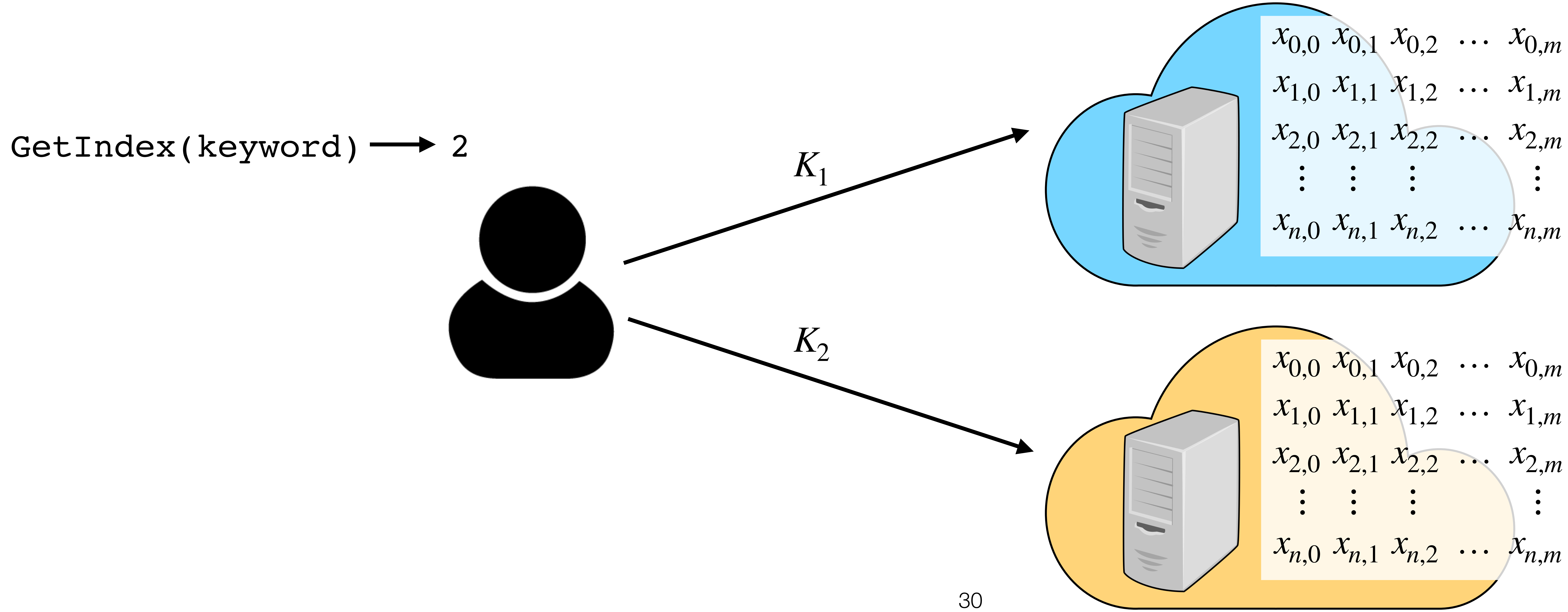
Leveraging DPFs to search

If at least one trust domain is honest, DORY hides search access patterns



Leveraging DPFs to search

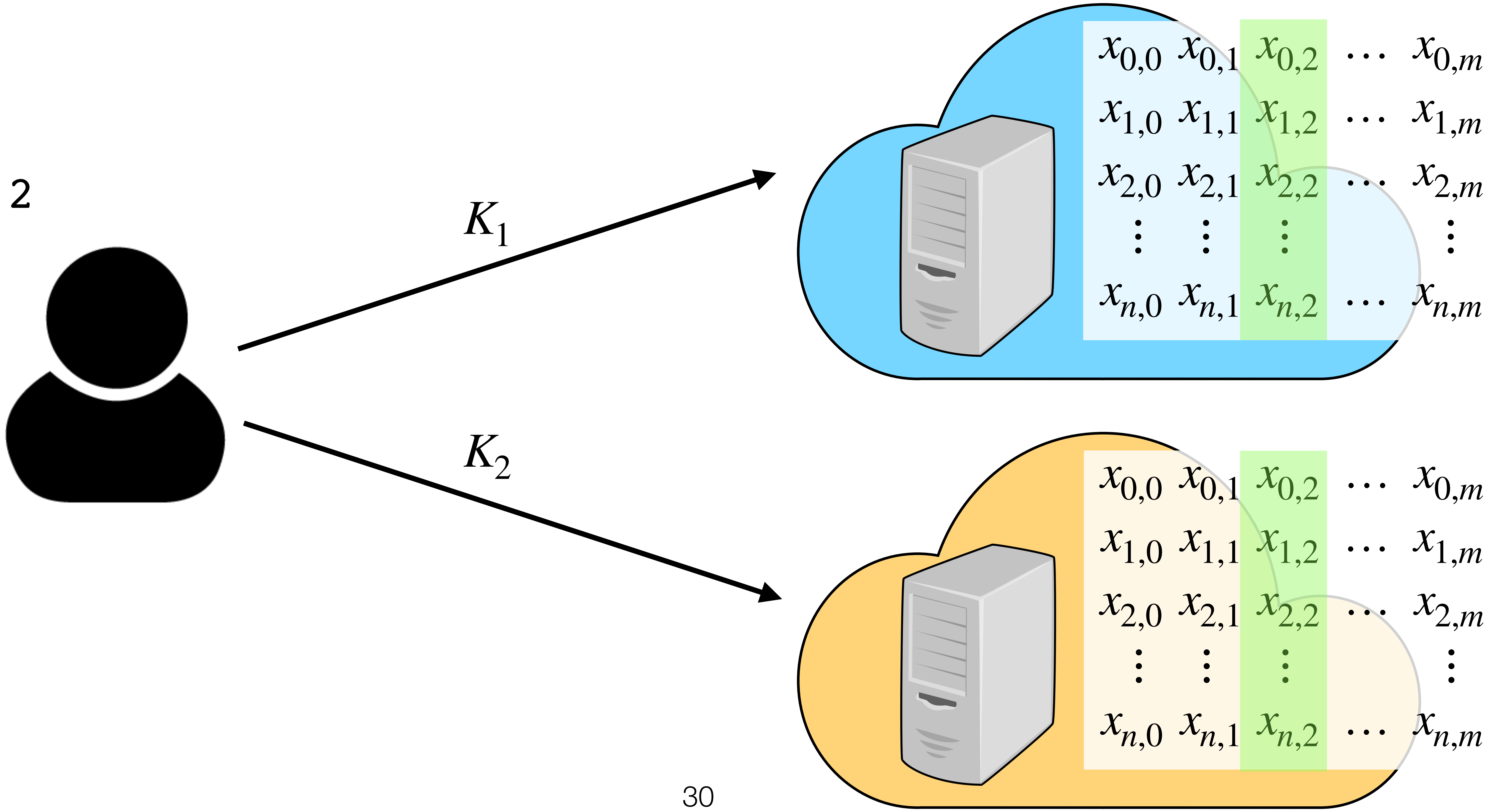
If at least one trust domain is honest, DORY hides search access patterns



Leveraging DPFs to search

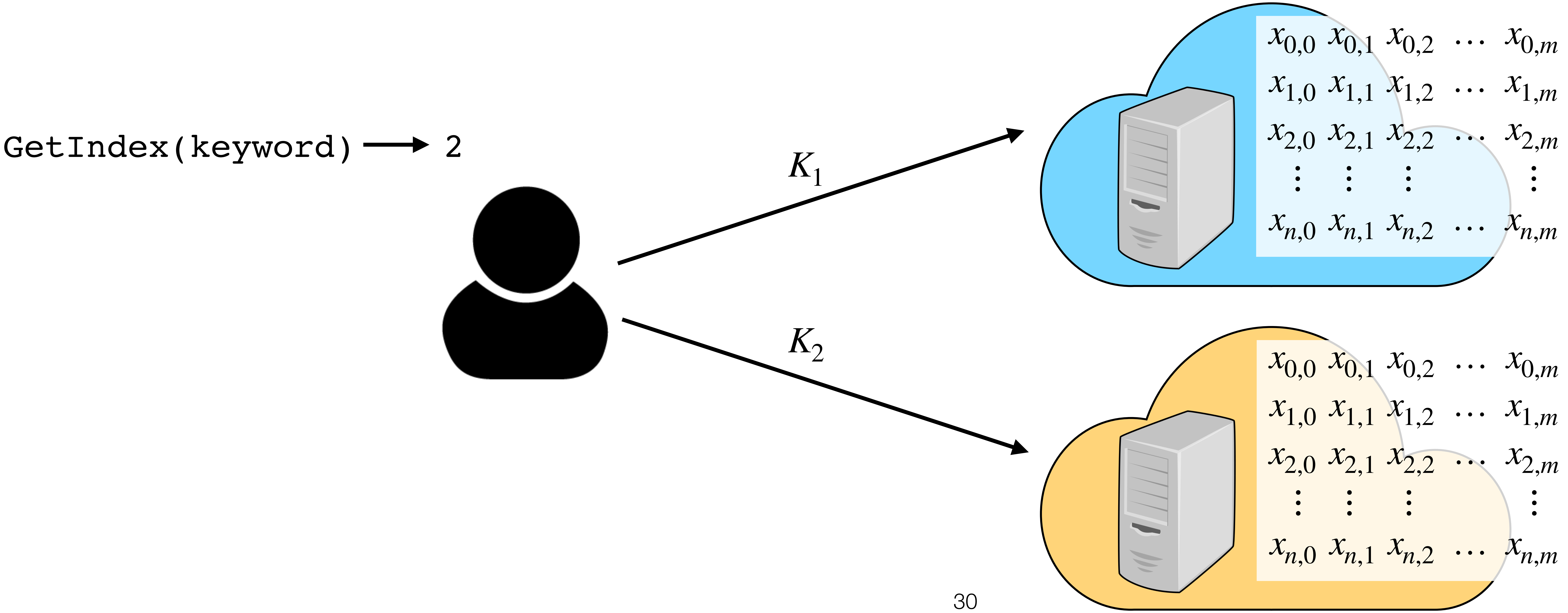
If at least one trust domain is honest, DORY hides search access patterns

`GetIndex(keyword)` ⟶ 2



Leveraging DPFs to search

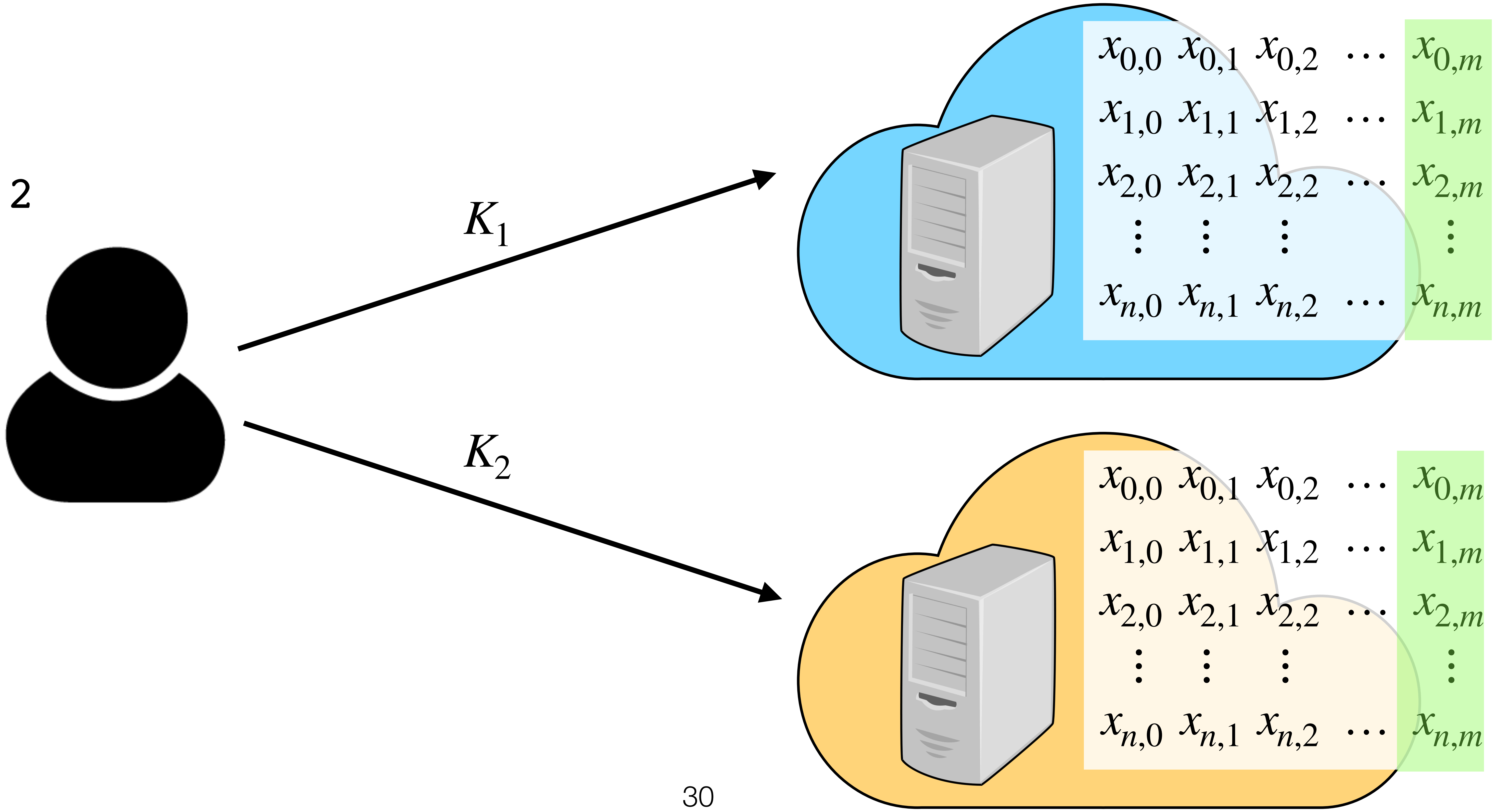
If at least one trust domain is honest, DORY hides search access patterns



Leveraging DPFs to search

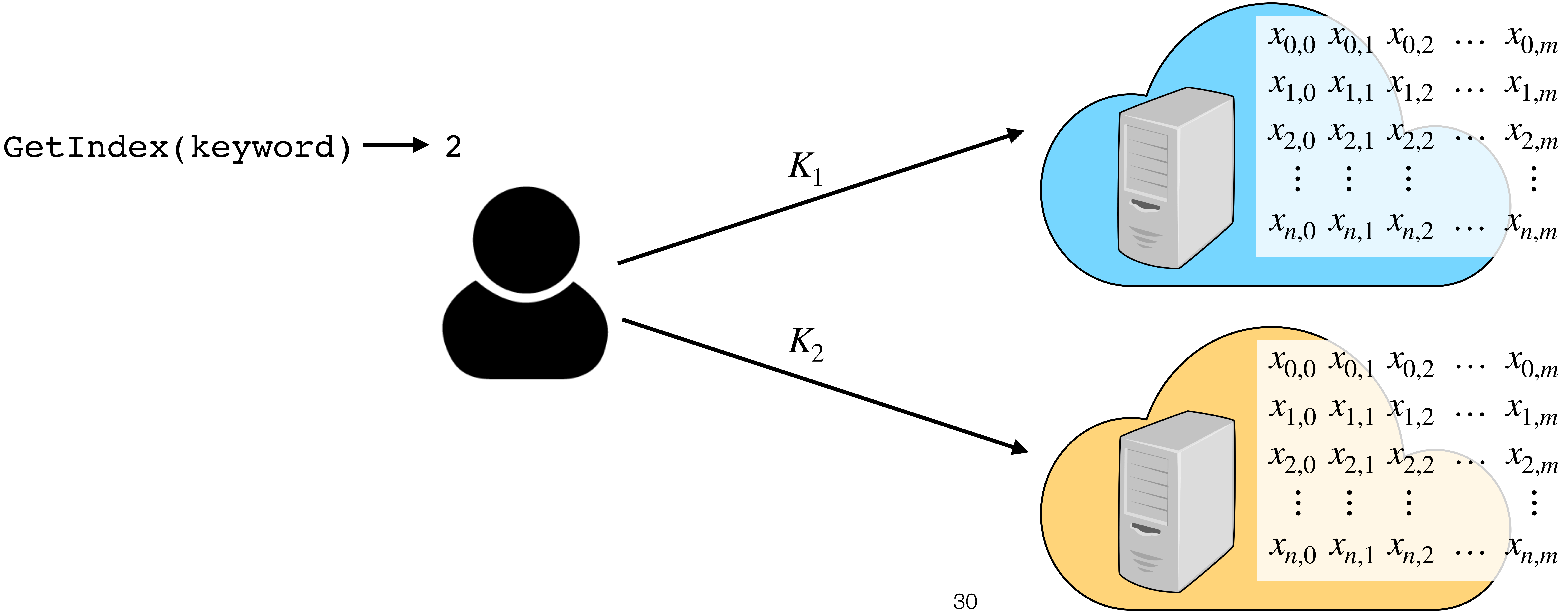
If at least one trust domain is honest, DORY hides search access patterns

`GetIndex(keyword)` ⟶ 2



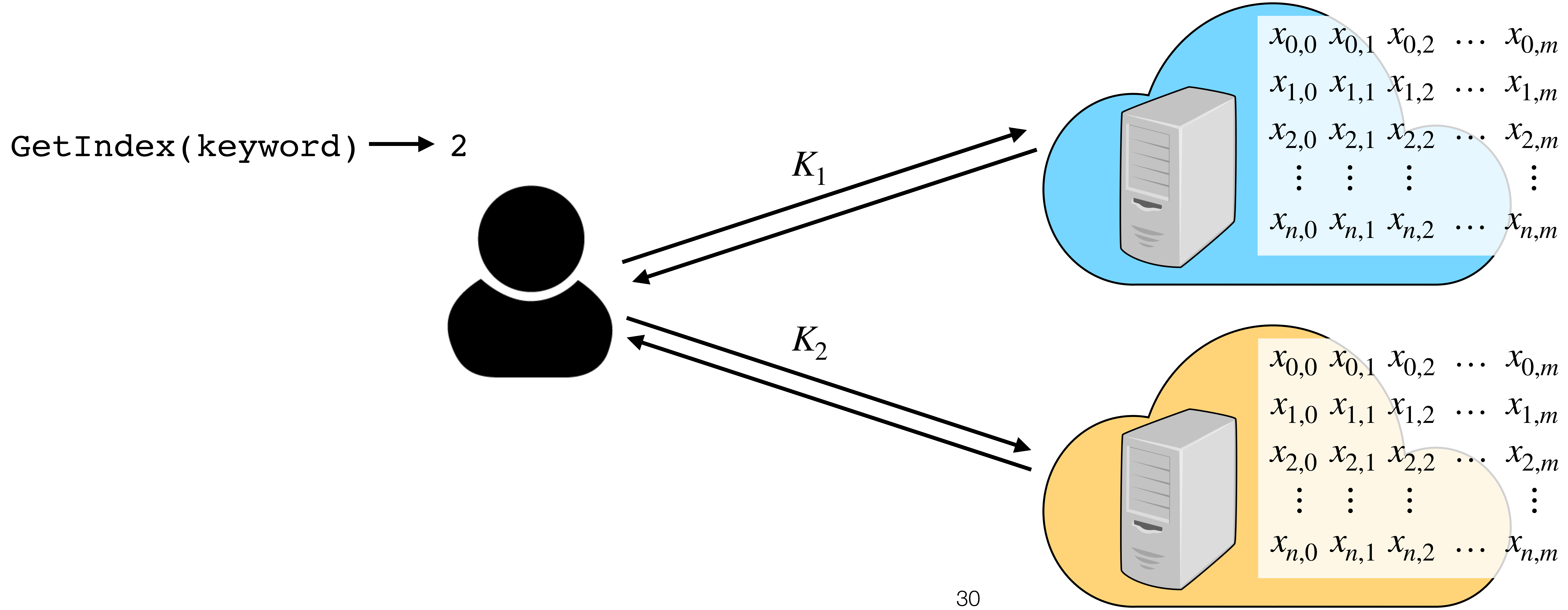
Leveraging DPFs to search

If at least one trust domain is honest, DORY hides search access patterns



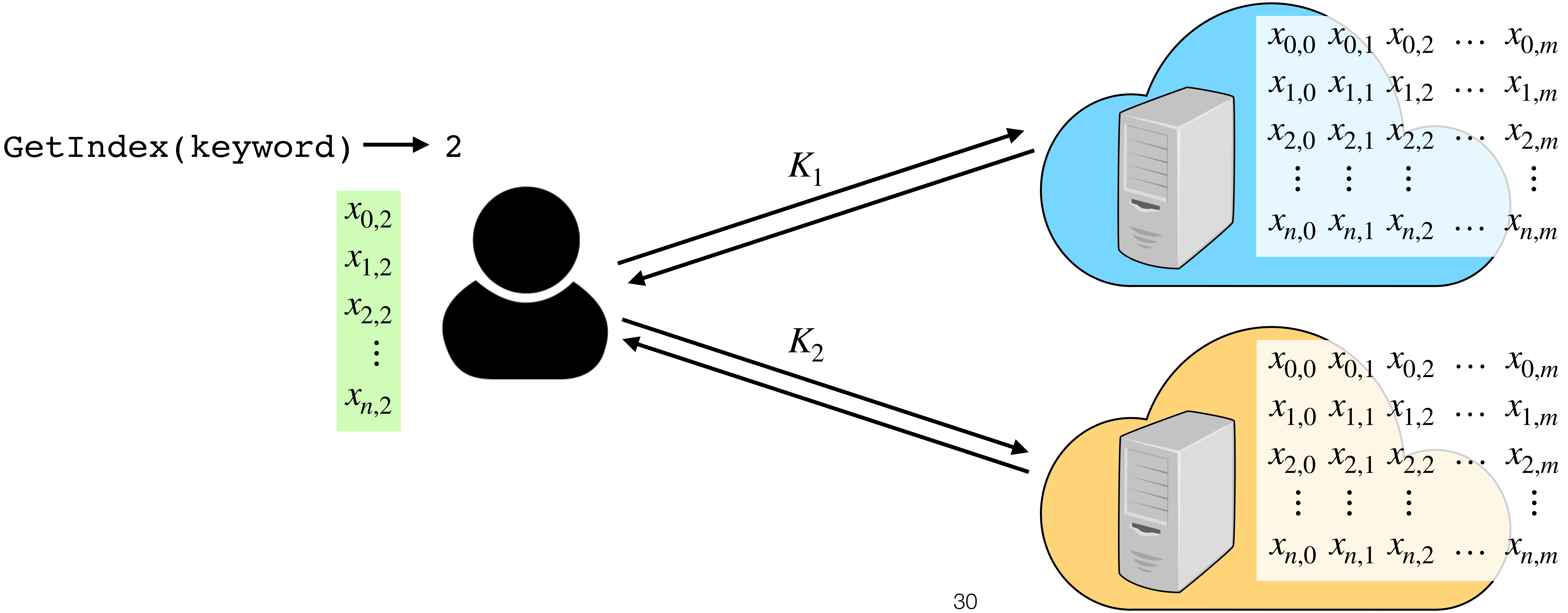
Leveraging DPFs to search

If at least one trust domain is honest, DORY hides search access patterns



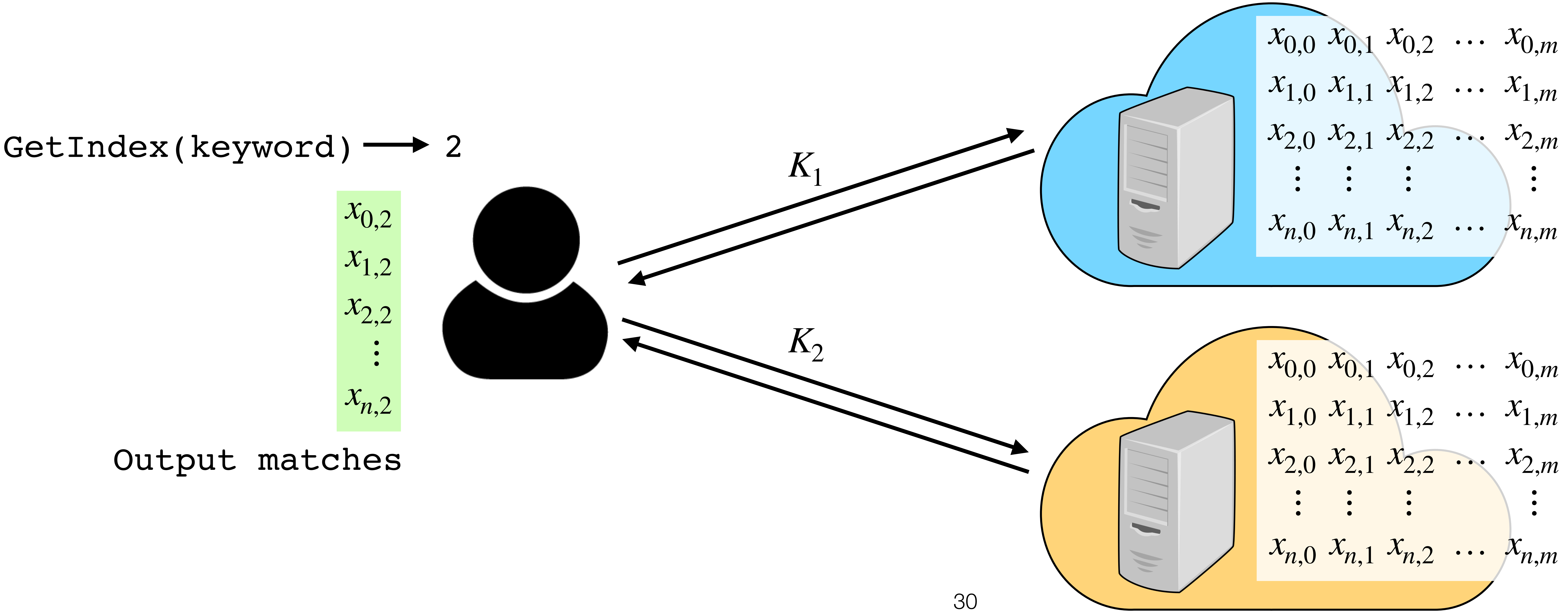
Leveraging DPFs to search

If at least one trust domain is honest, DORY hides search access patterns



Leveraging DPFs to search

If at least one trust domain is honest, DORY hides search access patterns



Challenge #2: Compressing the search index

A bitmap for every word in the English dictionary is long!

- The linear scan for search takes a long time…

$$
\begin{array}{lccccc}
\text{Doc } 0 & x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\
\text{Doc } 1 & x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\
\text{Doc } 2 & x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\
 & \vdots & \vdots & \vdots & & \vdots \\
\text{Doc } n & x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m}
\end{array}
$$

Bitmap for keywords in doc 1

Using Bloom filters to compress the search index

Bloom filters provide efficient membership testing

Apple
Orange

0 0 0 0 0 0 0 0 0 0

$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

Using Bloom filters to compress the search index

Bloom filters provide efficient membership testing

Apple
Orange

Apple 0 0 0 0 0 0 0 0 0 0

$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

Using Bloom filters to compress the search index

Bloom filters provide efficient membership testing

Apple
Orange

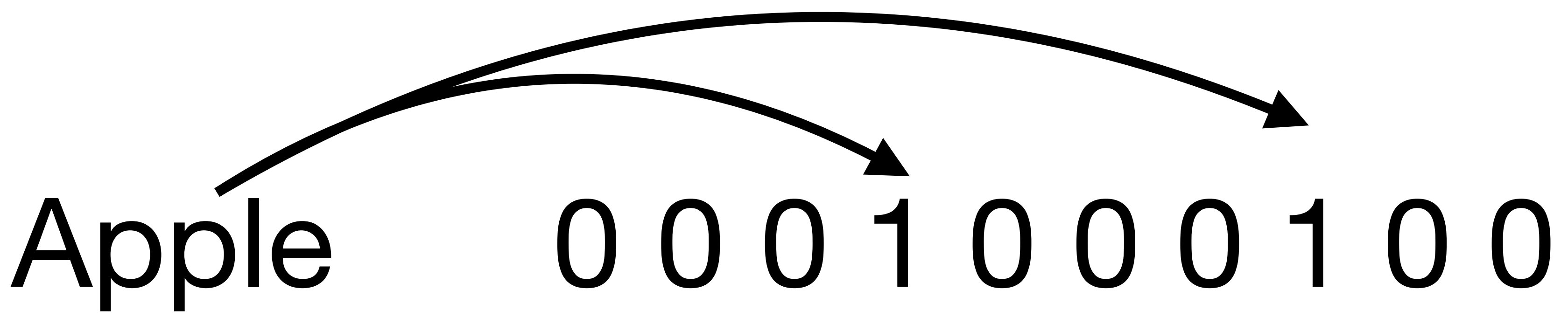
Apple 0 0 0 0 0 0 0 0 0 0

$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

Using Bloom filters to compress the search index

Bloom filters provide efficient membership testing

~~Apple~~
Orange



$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

Using Bloom filters to compress the search index

Bloom filters provide efficient membership testing

~~Apple~~
Orange

Orange 0 0 0 1 0 0 0 1 0 0

$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

Using Bloom filters to compress the search index

Bloom filters provide efficient membership testing

~~Apple~~
Orange

Orange 0 0 0 1 0 0 0 1 0 0

$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

Using Bloom filters to compress the search index

Bloom filters provide efficient membership testing

~~Apple~~
~~Orange~~

Orange 0 0 1 1 0 1 0 1 0 0

$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

Using Bloom filters to compress the search index

Bloom filters provide efficient membership testing

~~Apple~~
~~Orange~~

Orange 0 0 1 1 0 1 0 1 0 0

$$\begin{array}{ccccc} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{array}$$

Using Bloom filters to compress the search index

Bloom filters provide efficient membership testing

~~Apple~~
~~Orange~~

Orange 0 0 1 1 0 1 0 1 0 0

$$
\begin{matrix}
x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\
x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\
x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\
\vdots & \vdots & \vdots & & \vdots \\
x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m}
\end{matrix}
$$

+ Preserves search column alignment

+ Compression

+ No fixed dictionary

Challenge #3: Encrypting the search index

Attacker should not immediately learn the search index contents.

Strawman: Encrypt every bit in Bloom filter.

- Search index size blows up by factor of $\lambda \approx 128$.

$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix}$$

Challenge #3: Encrypting the search index

Attacker should not immediately learn the search index contents.

Strawman: Encrypt every bit in Bloom filter.

- Search index size blows up by factor of $\lambda \approx 128$.

$$
\begin{matrix}
x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\
x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\
x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\
\vdots & \vdots & \vdots & & \vdots \\
x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m}
\end{matrix}
\quad \Longrightarrow \quad
\begin{matrix}
\mathsf{Enc}_k(x_{0,0}) & \mathsf{Enc}_k(x_{0,1}) & \mathsf{Enc}_k(x_{0,2}) & \ldots & \mathsf{Enc}_k(x_{0,m}) \\
\mathsf{Enc}_k(x_{1,0}) & \mathsf{Enc}_k(x_{1,1}) & \mathsf{Enc}_k(x_{1,2}) & \ldots & \mathsf{Enc}_k(x_{1,m}) \\
\mathsf{Enc}_k(x_{2,0}) & \mathsf{Enc}_k(x_{2,1}) & \mathsf{Enc}_k(x_{2,2}) & \ldots & \mathsf{Enc}_k(x_{2,m}) \\
\vdots & \vdots & \vdots & & \vdots \\
\mathsf{Enc}_k(x_{n,0}) & \mathsf{Enc}_k(x_{n,1}) & \mathsf{Enc}_k(x_{n,2}) & \ldots & \mathsf{Enc}_k(x_{n,m})
\end{matrix}
$$

Challenge #3: Encrypting the search index

Solution: generate a unique one-time pad using document version number.

$$
\begin{matrix}
x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\
x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\
x_{2,0} & x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\
\vdots & \vdots & \vdots & & \vdots \\
x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m}
\end{matrix}
\quad \bigoplus \quad
\begin{matrix}
\mathsf{PRF}_k(0 \,||\, \mathsf{version}_0) \\
\mathsf{PRF}_k(1 \,||\, \mathsf{version}_1) \\
\mathsf{PRF}_k(2 \,||\, \mathsf{version}_2) \\
\vdots \\
\mathsf{PRF}_k(n \,||\, \mathsf{version}_n)
\end{matrix}
\quad \Rightarrow \quad
\begin{matrix}
y_{0,0} & y_{0,1} & y_{0,2} & \cdots & y_{0,m} \\
y_{1,0} & y_{1,1} & y_{1,2} & \cdots & y_{1,m} \\
y_{2,0} & y_{2,1} & y_{2,2} & \cdots & y_{2,m} \\
\vdots & \vdots & \vdots & & \vdots \\
y_{n,0} & y_{n,1} & y_{n,2} & \cdots & y_{n,m}
\end{matrix}
$$

Challenge #4: Malicious attackers

Need to defend against attackers that can influence server behavior.

Strawman: MAC every bit

- Search index (and search time) blows up by factor of λ.

$$\begin{matrix} x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{matrix} \qquad \begin{matrix} t_{0,0} & t_{0,1} & t_{0,2} & \cdots & t_{0,m} \\ t_{1,0} & t_{1,1} & t_{1,2} & \cdots & t_{1,m} \\ \vdots & \vdots & \vdots & & \vdots \\ t_{n,0} & t_{n,1} & t_{n,2} & \cdots & t_{n,m} \end{matrix}$$

[KL08]

Challenge #4: Malicious attackers

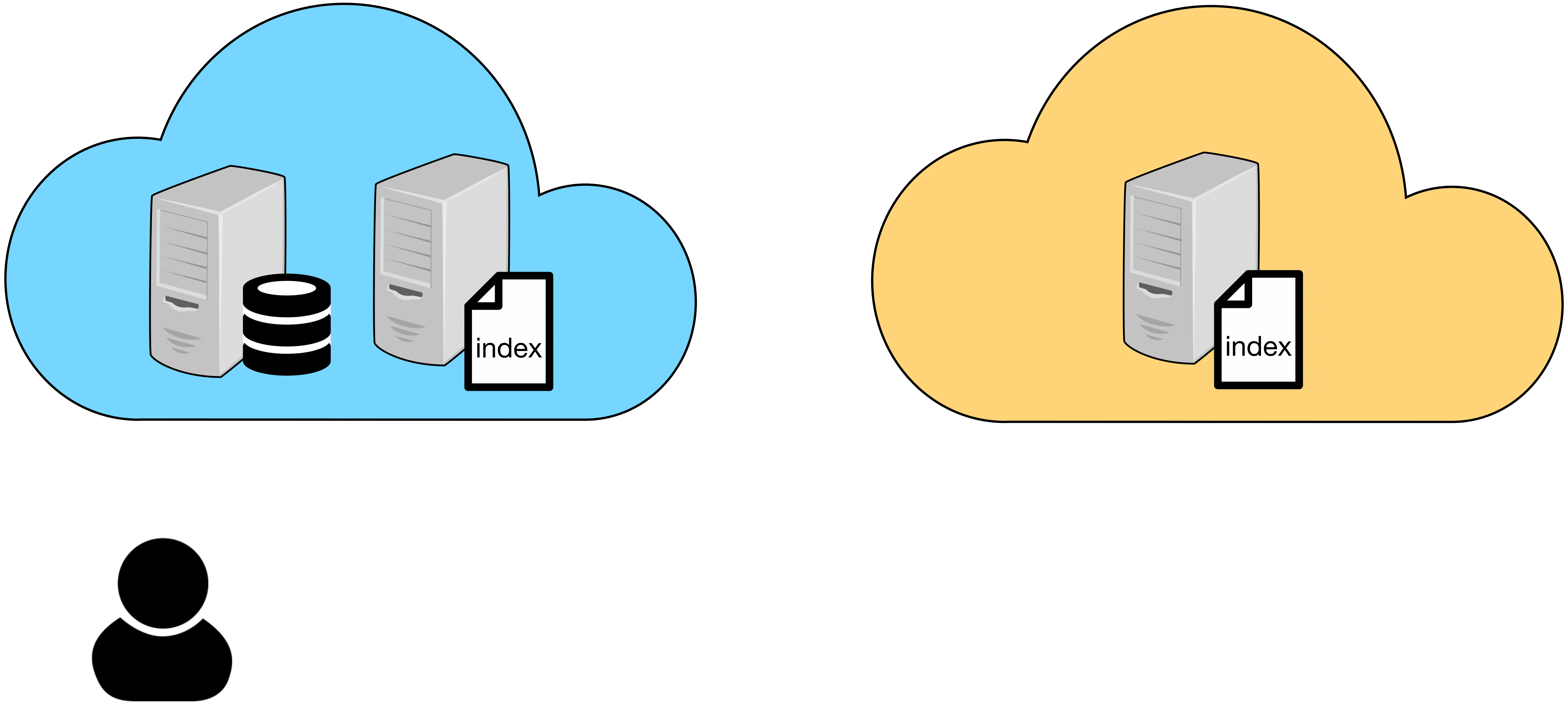
Need to defend against attackers that can influence server behavior.

Solution: use **aggregate MACs** to keep a single MAC per column.

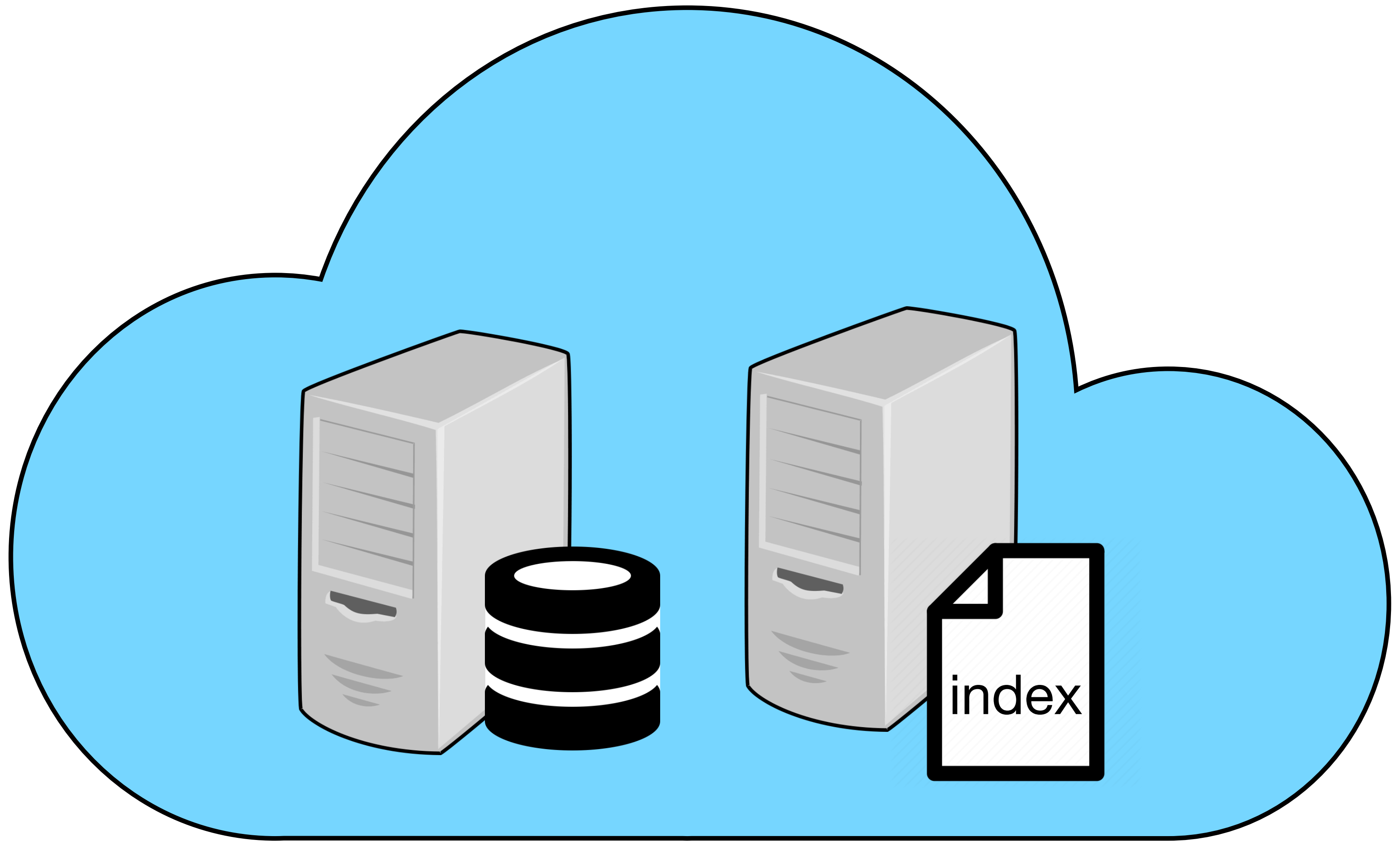
$$\begin{array}{ccccc} t_0 & t_1 & t_2 & \cdots & t_m \\ x_{0,0} & x_{0,1} & x_{0,2} & \cdots & x_{0,m} \\ x_{1,0} & x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ \vdots & \vdots & \vdots & & \vdots \\ x_{n,0} & x_{n,1} & x_{n,2} & \cdots & x_{n,m} \end{array}$$

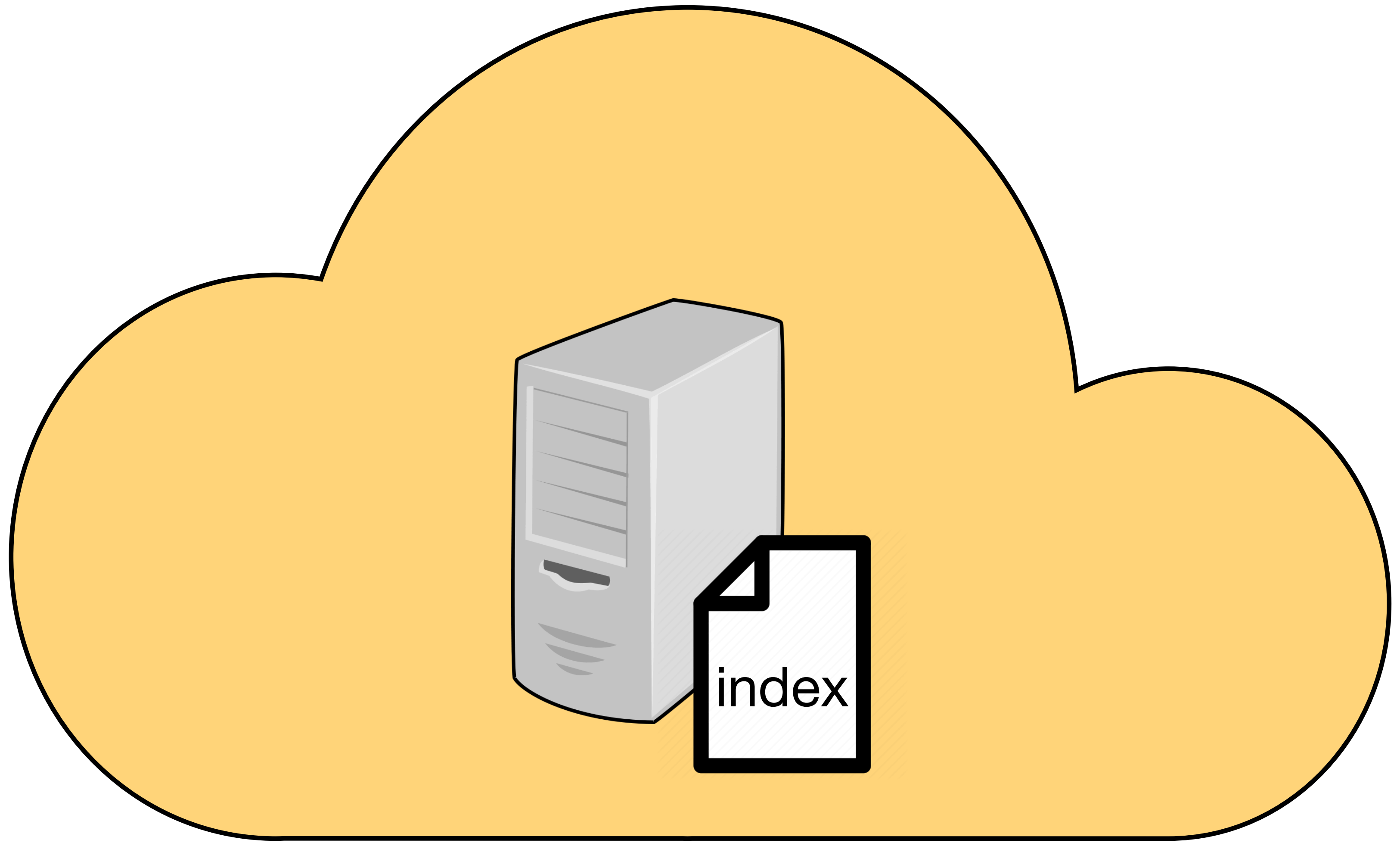
[KL08]

Other contributions (see paper)



Other contributions (see paper)

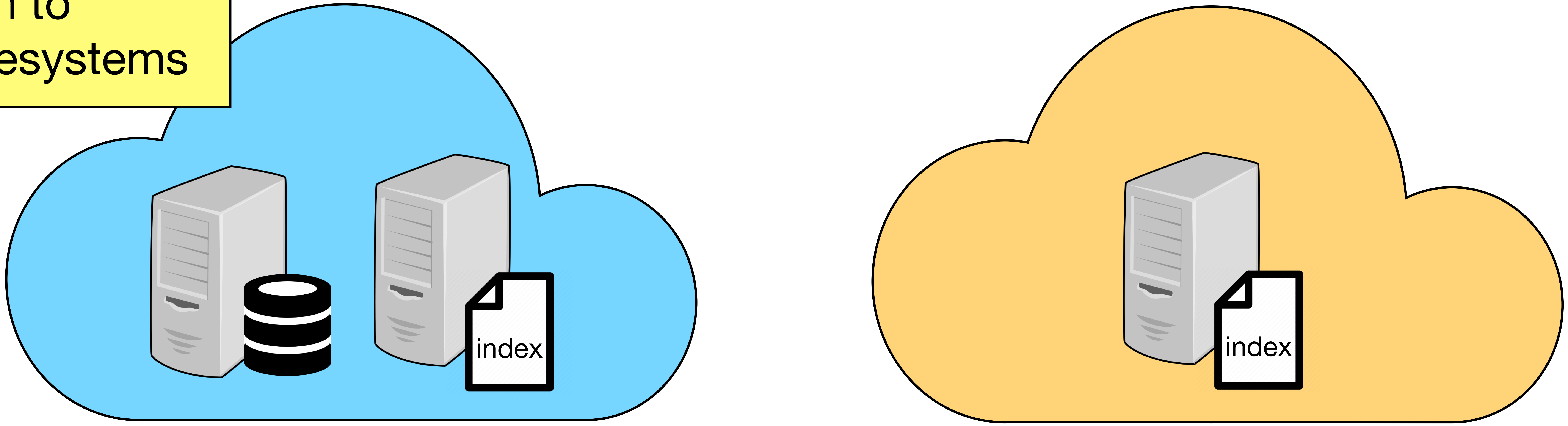




1. Efficient user revocation

Other contributions (see paper)

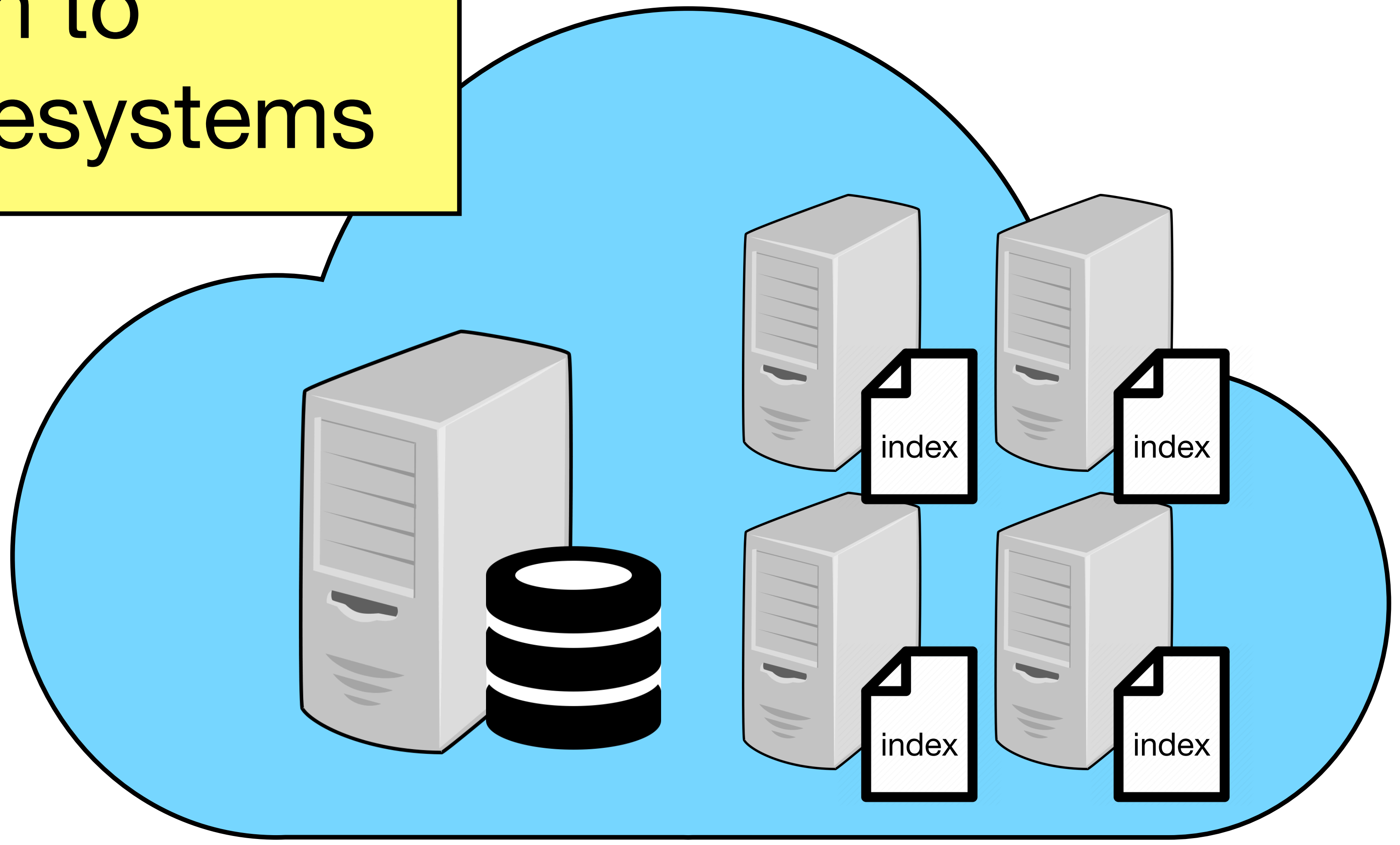
2. Extension to oblivious filesystems



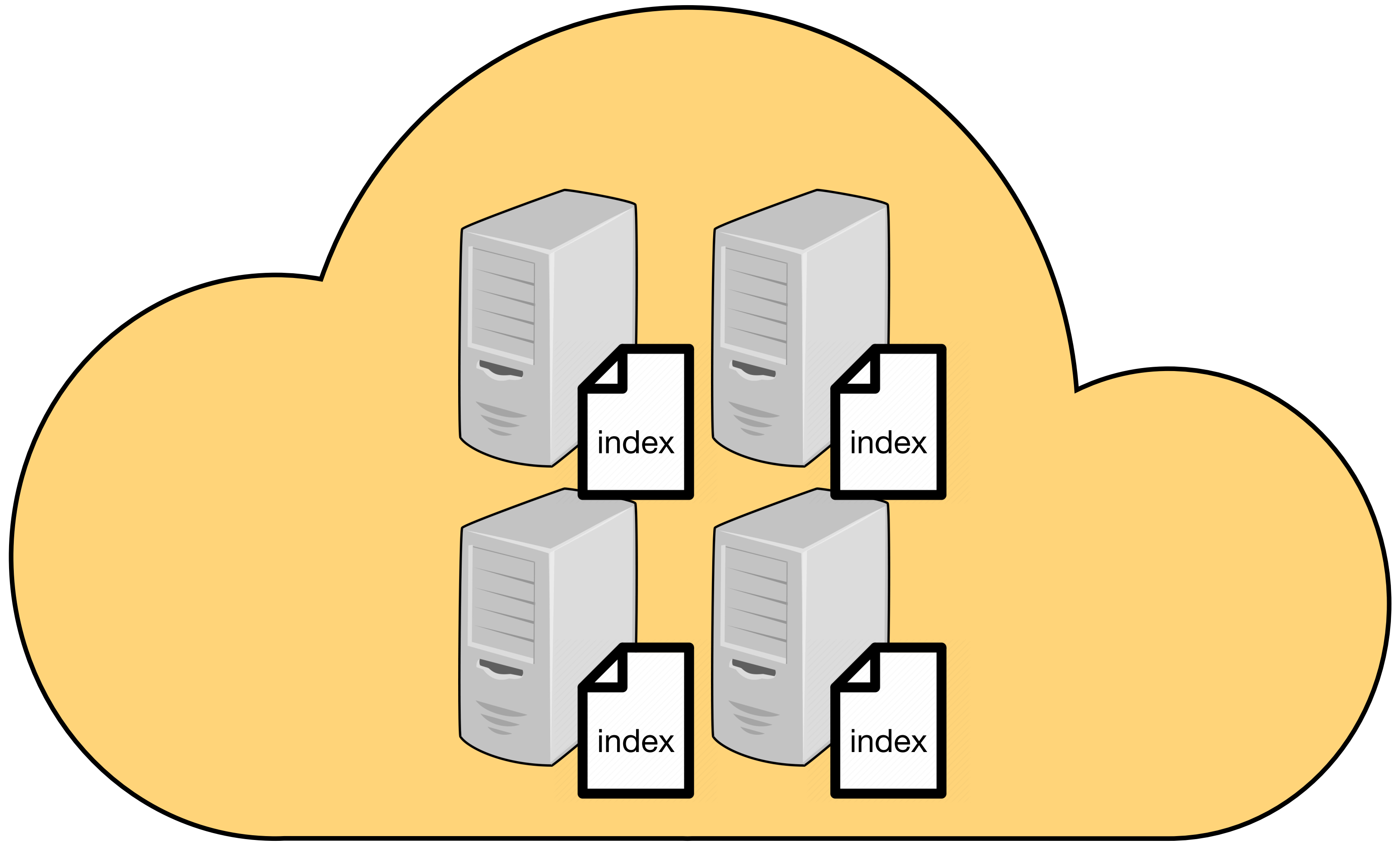
1. Efficient user revocation

Other contributions (see paper)

2. Extension to oblivious filesystems



1. Efficient user revocation



3. Efficient replication leveraging DORY’s cryptographic properties

Outline

~~1. DORY design~~

2. DORY evaluation

Evaluation setup

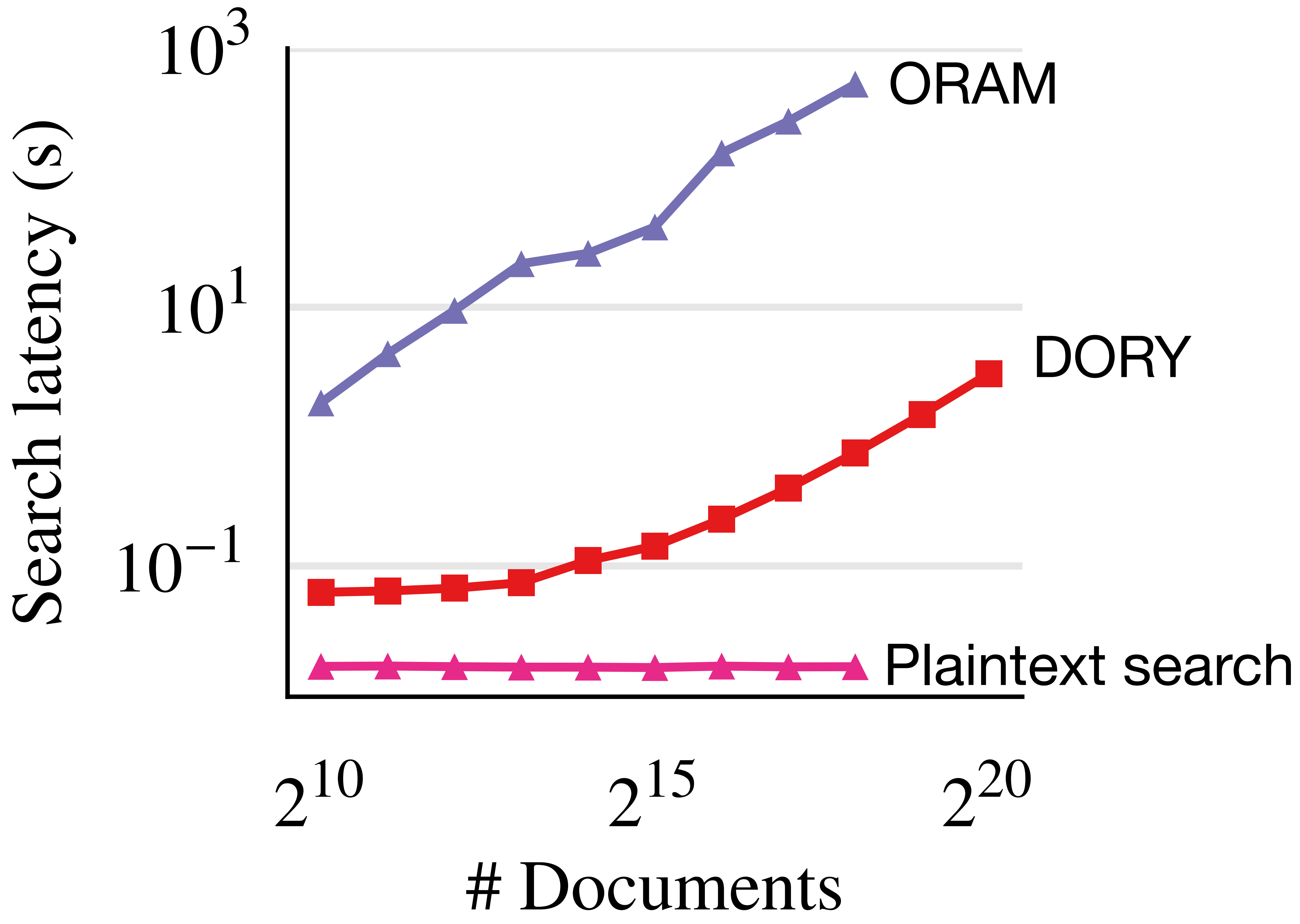
https://github.com/ucbrise/dory

Evaluated performance using Enron email dataset.

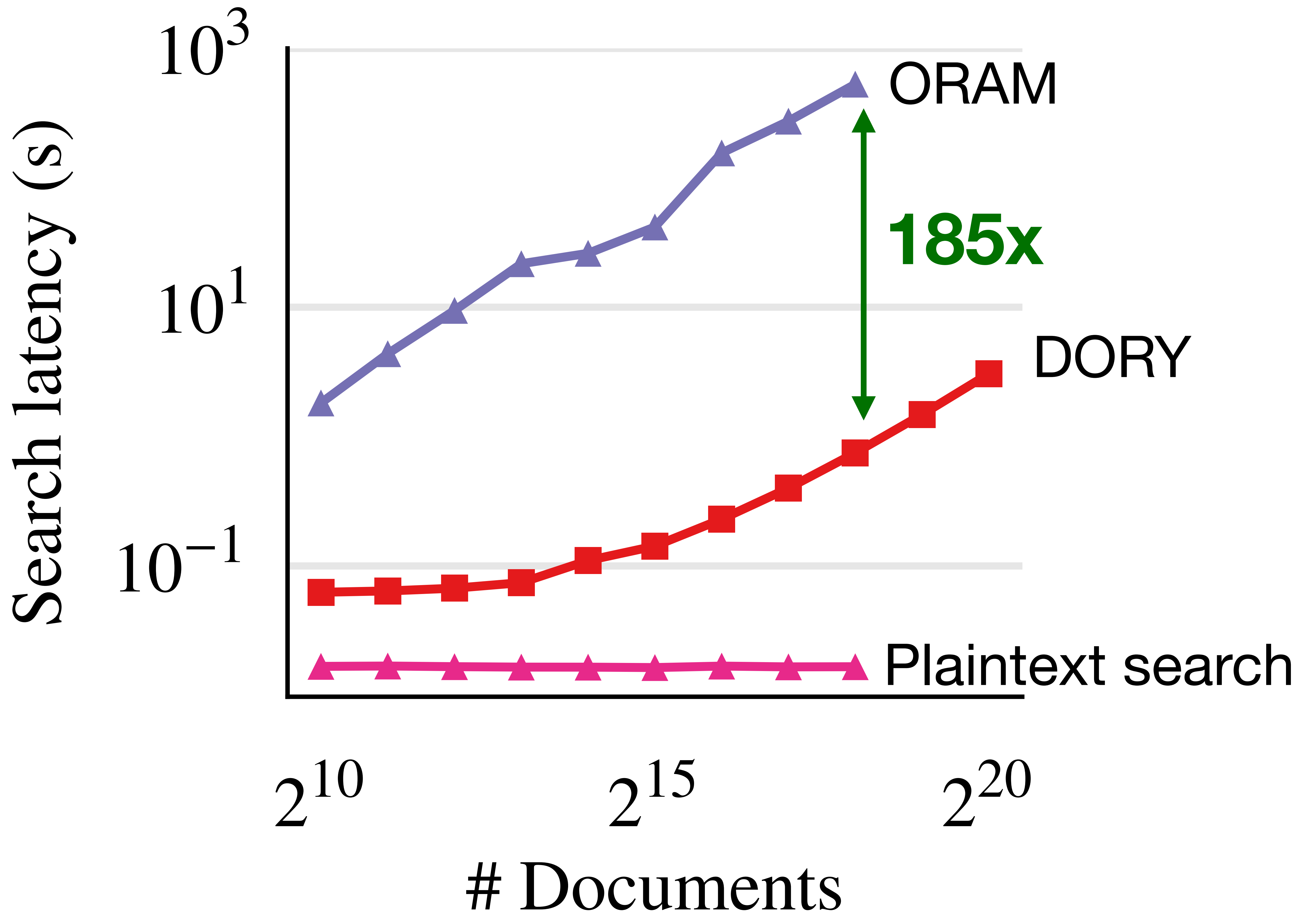
Two baselines:

- Plaintext search: inverted index without encryption
- ORAM baseline: inverted index in PathORAM [SVSF+13] (see paper)

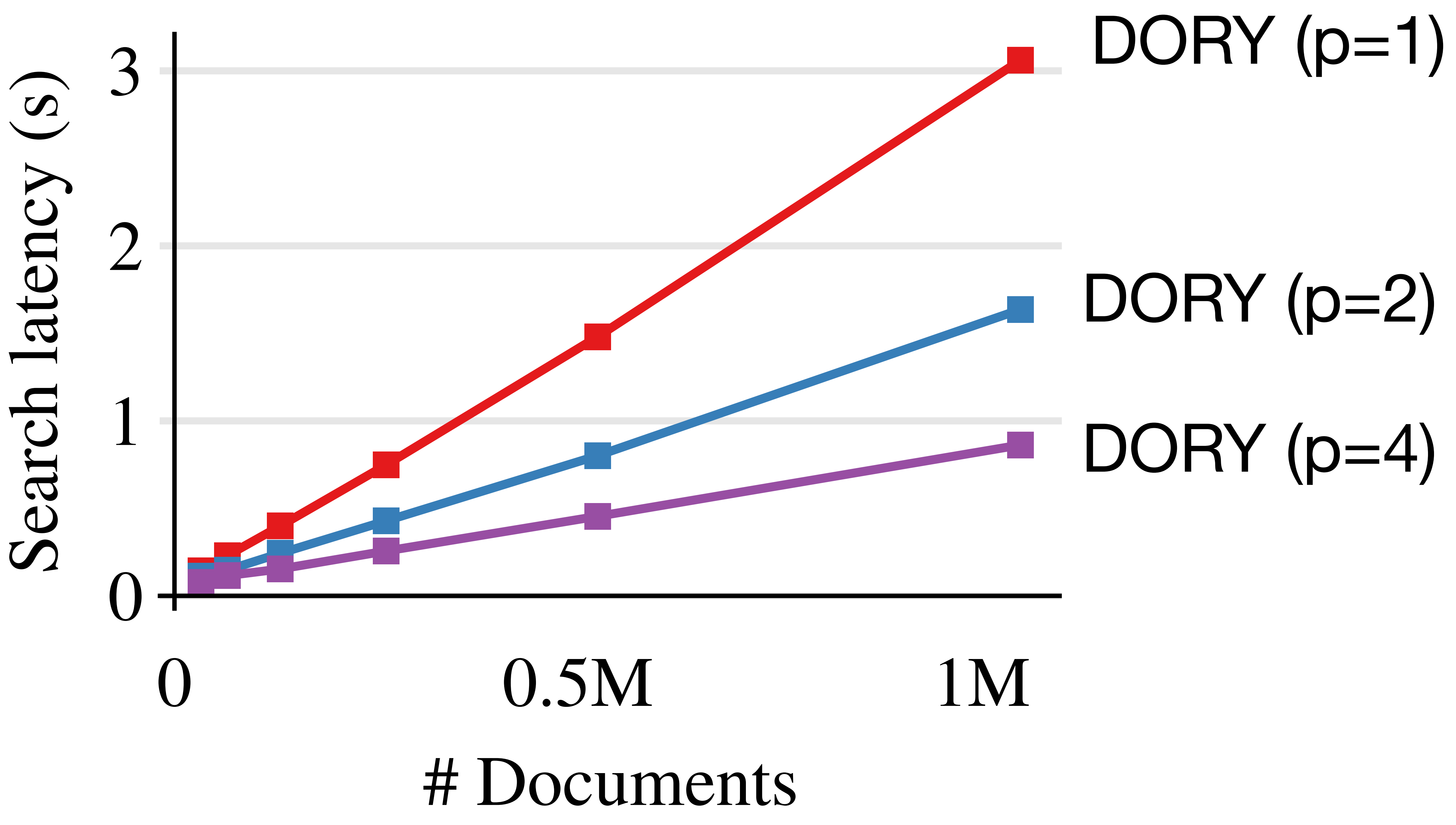
Search latency



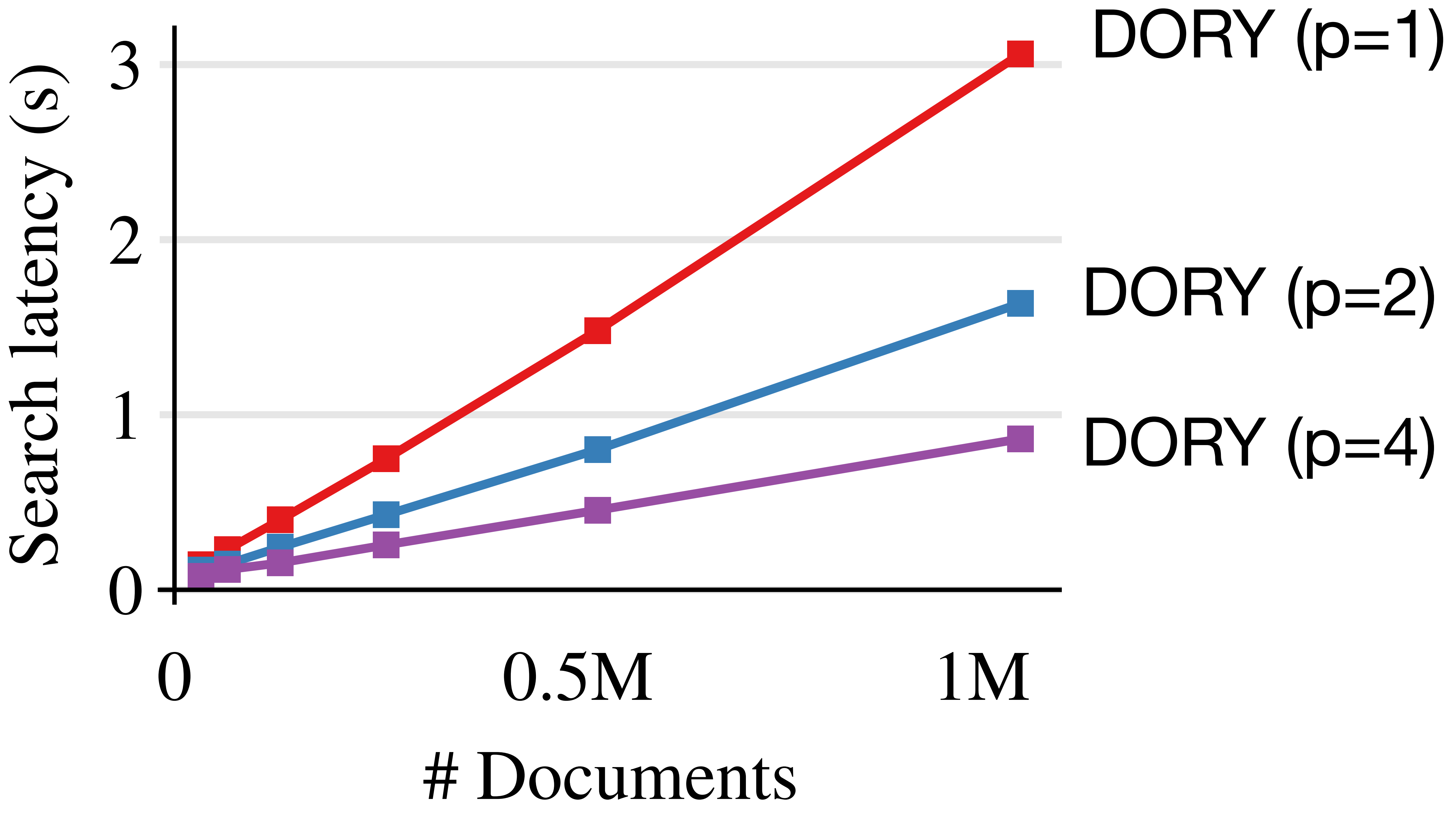
Search latency



Effect of parallelism on search latency

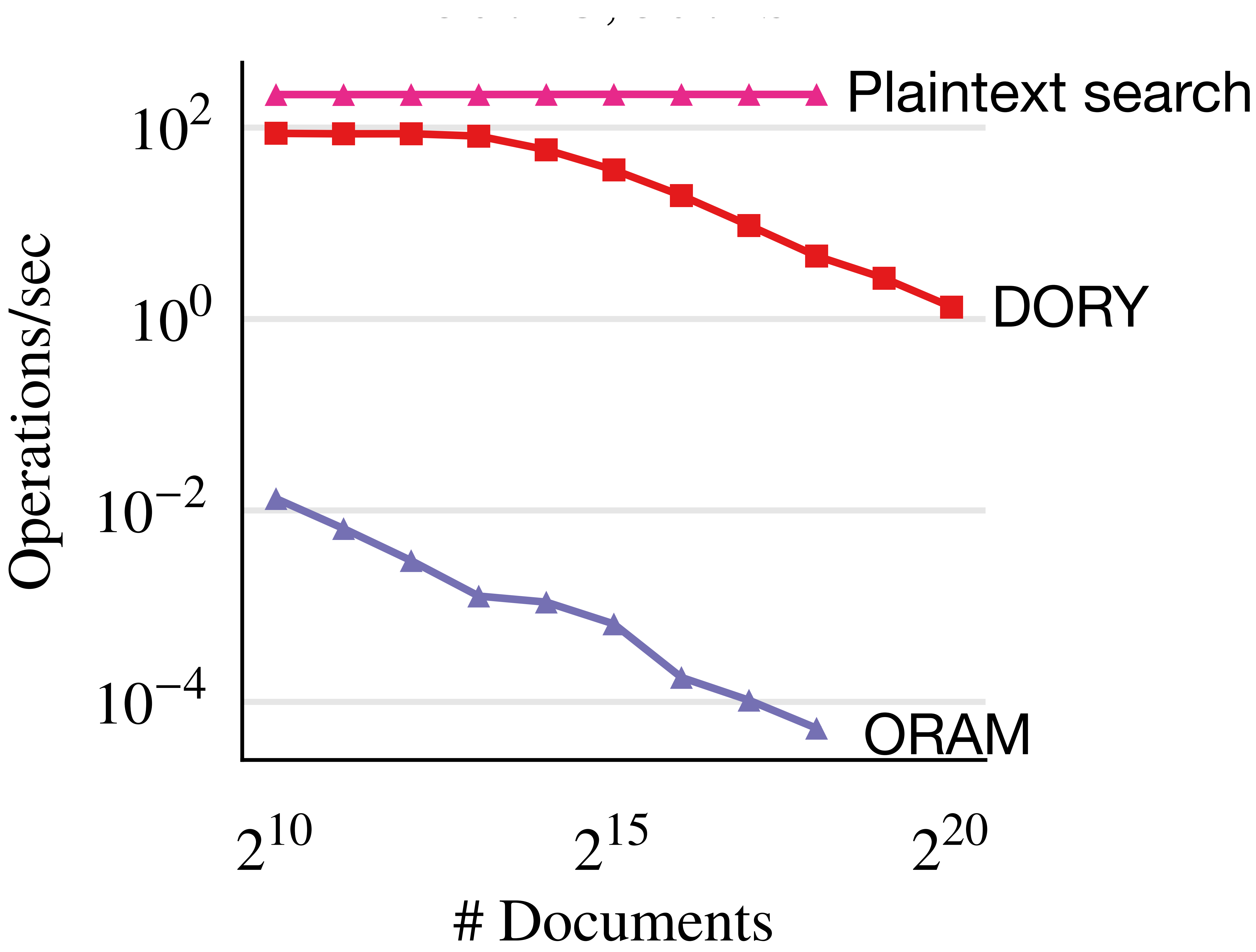


Effect of parallelism on search latency



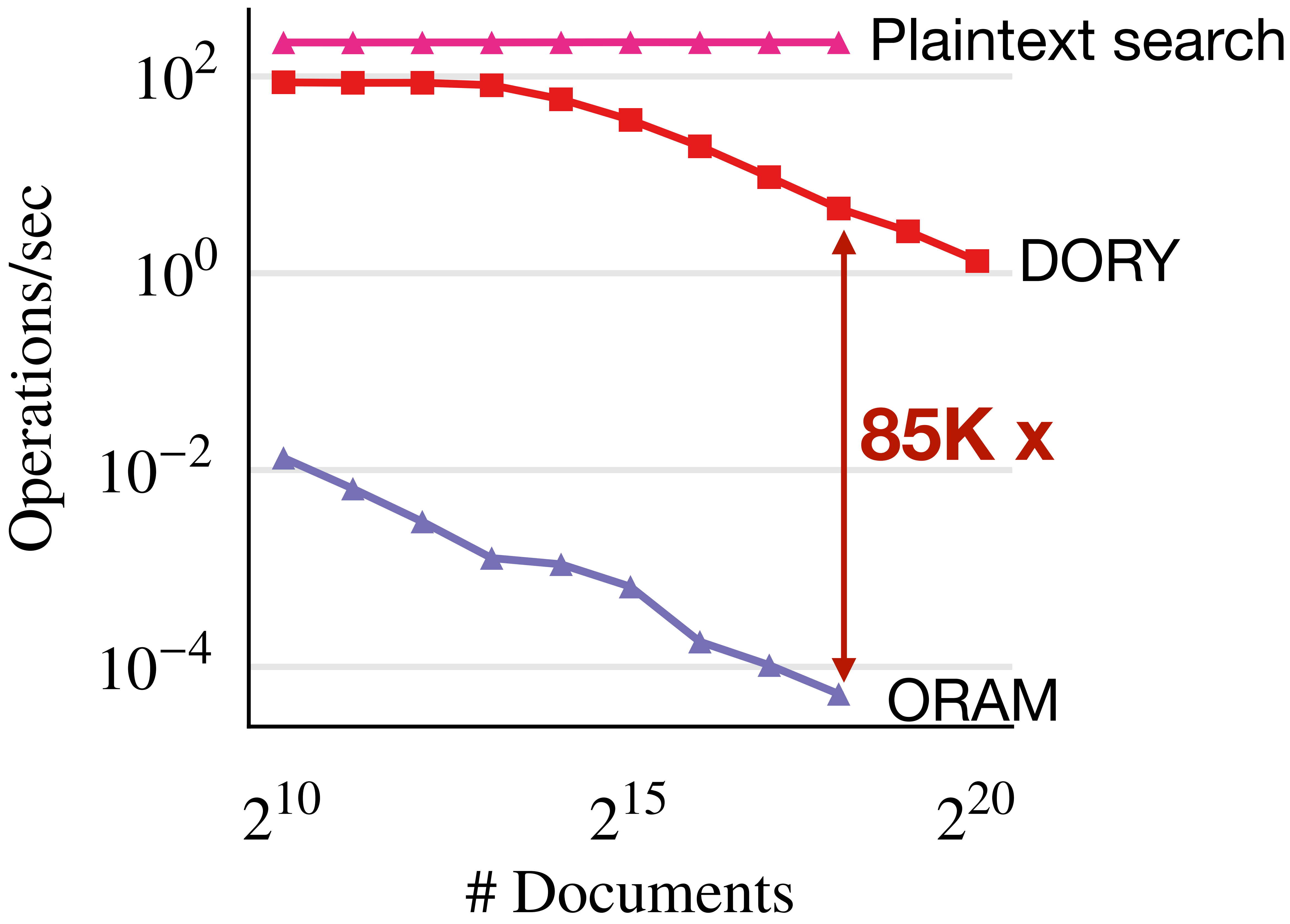
Parallelism improves search latency by roughly a factor of p (degree of parallelism).

Throughput



50% updates, 50% searches

Throughput



50% updates, 50% searches

Conclusion

- DORY is an efficient search system that hides search access patterns.
- By re-examining the system model, DORY reconciles the tension between efficiency and search access patterns.
- Search should not be a barrier to adoption of end-to-end encrypted systems.

Conclusion

- DORY is an efficient search system that hides search access patterns.
- By re-examining the system model, DORY reconciles the tension between efficiency and search access patterns.
- Search should not be a barrier to adoption of end-to-end encrypted systems.

Thanks! 

Emma Dauterman
edauterman@berkeley.edu
https://github.com/ucbrise/dory

References

[GO96] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[SWP00] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Security & Privacy*, pages 44–55. IEEE, 2000.

[Goh03] E.-J. Goh et al. Secure indexes. *IACR Cryptology ePrint Archive*, 2003:216, 2003.

[KL08] J. Katz and A. Y. Lindell. Aggregate message authentication codes. In *Cryptographers' Track at the RSA Conference*, pages 155–169, 2008.

[CGKO11] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[KPR12] S. Kamara, C. Papamanthou, and T. Roeder. Dynamic searchable symmetric encryption. In *CCS*, pages 965–976. ACM, 2012.

[KP13] S.KamaraandC.Papamanthou. Parallel and dynamic searchable symmetric encryption. In *Financial Cryptography and Data Security*, pages 258–274. Springer, 2013.

[SVSF+13] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *CCS*, pages 299–310. ACM, 2013.

[CJJJ+14] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: data structures and implementation. In *NDSS*, volume 14, pages 23–26. Citeseer, 2014.

[SPS14] E. Stefanov, C. Papamanthou, and E. Shi. Practical dynamic searchable encryption with small leakage. In *NDSS*, volume 71, pages 72–75, 2014.

[ZKP16] Y. Zhang, J. Katz, and C. Papamanthou. All your queries are belong to us: The power of file-injection attacks on searchable encryption. In *USENIX Security*, pages 707–720, 2016.

[DPP18] I. Demertzis, D. Papadopoulos, and C. Papamanthou. Searchable encryption with optimal locality: Achieving sublogarithmic read efficiency. In *CRYPTO*, 2018.