Dynamic Searchable Encryption with Small Client Storage

Ioannis Demertzis
University of Maryland
yannis@umd.edu

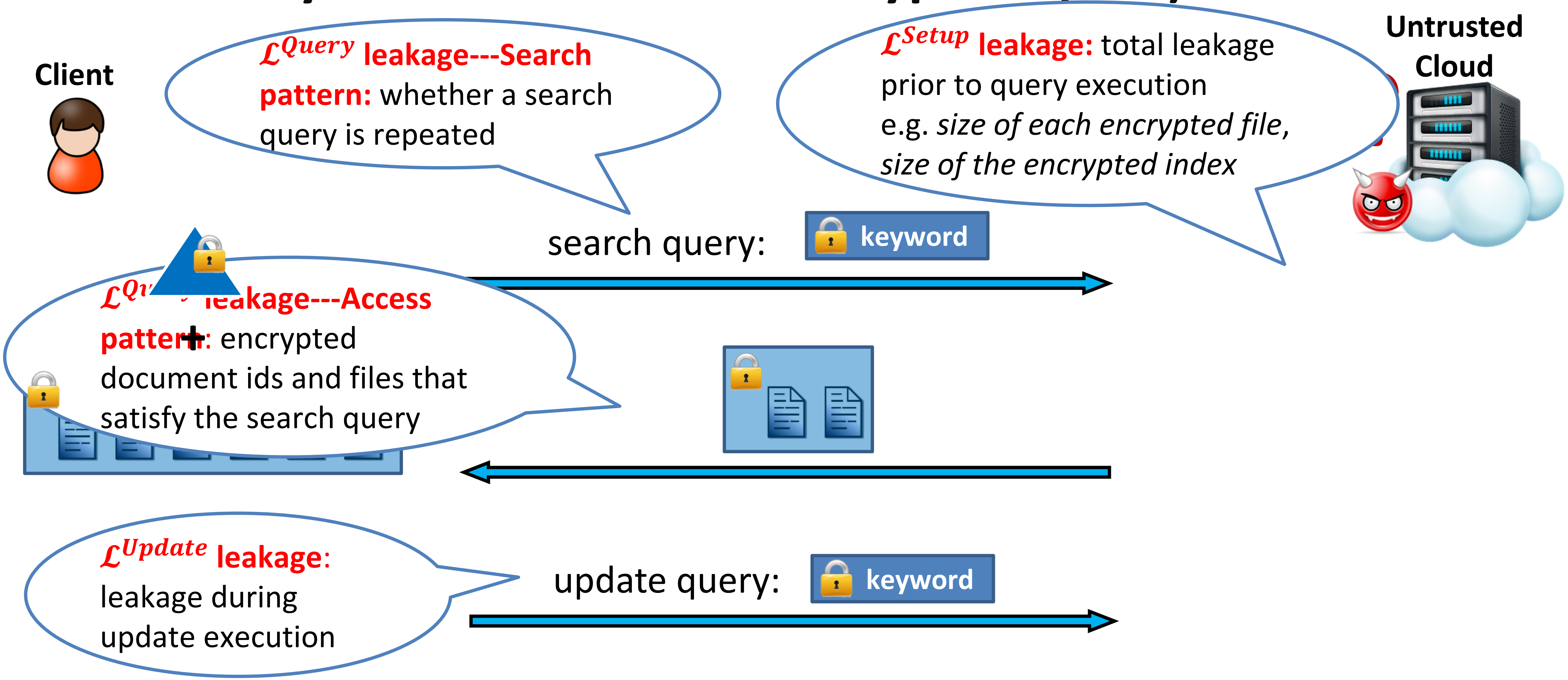
Javad Ghareh Chamani
HKUST and Sharif University of Technology
jgc@cse.ust.hk

Dimitrios Papadopoulos
HKUST
dipapado@cse.ust.hk

Charalampos Papamathou
University of Maryland
cpap@umd.edu

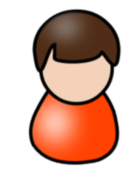
香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

What is Dynamic Searchable Encryption (DSE)?

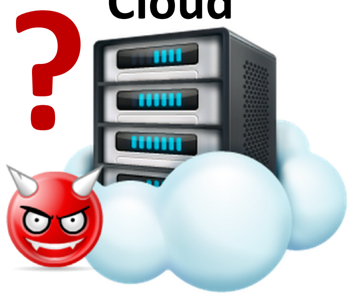


<u>Security (informal):</u> The adversary does not learn anything beyond the above leakages!

Update Leakage --- Forward Privacy



High-level idea: The server should **not** be able to relate an update with a previous operation!

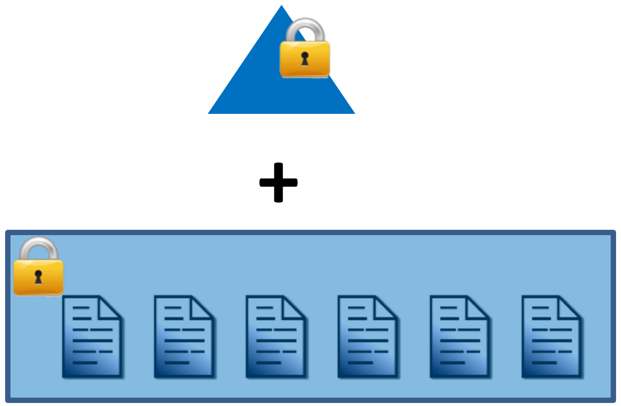


Time

1 — Add (F1, **w1**) →

2 — Query (**w1**) →

3 — Add (F2, **w1**) →

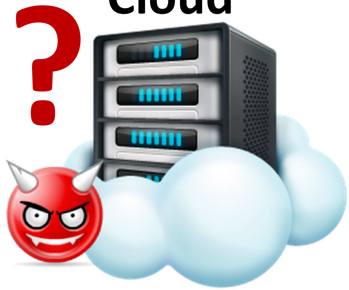


- Server should **not** learn that update in timestamp **3** is for the same keyword!
- ***Definition:*** *A DSE scheme is forward private if the update does not reveal any information about the involved keyword, i.e.,* $\mathcal{L}^{Update}(w) = \mathcal{L}(op, id)$

Update Leakage --- Backward Privacy

Client

High-level idea: The server should reveal controlled information about deleted files during search



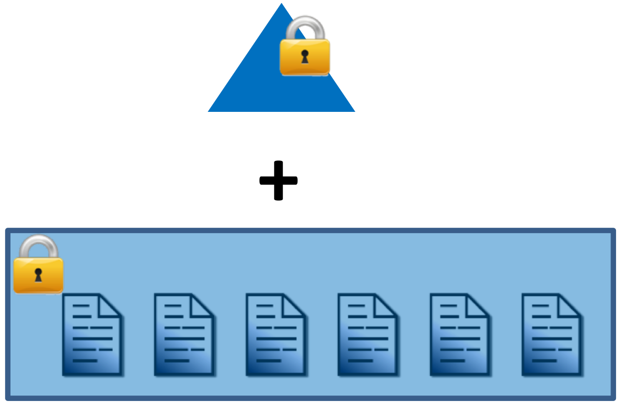
Time

1 Add (F1, **w1**)

2 Add (F2, **w1**)

3 Del (F1, **w1**)

4 Query (**w1**)



$\boldsymbol{TimeDB}(\boldsymbol{w1}) = \{(F2, 2)\}$

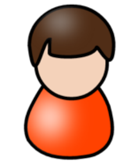
$\boldsymbol{TimeDB}(w) = \{$files **currently** containing w and when they were stored$\}$

$\boldsymbol{Updates}(w) = \{$timestamp of each update for w$\}$ $\boldsymbol{Updates}(\boldsymbol{w1}) = \{1, 2, 3\}$

$\boldsymbol{DelHist}(w) = \{$exactly which deletion cancels which addition$\}$ $\boldsymbol{DelHist}(\boldsymbol{w1}) = \{(1, 3)\}$

Update Leakage --- Backward Privacy

Client



Untrusted Cloud

High-level idea: The server should reveal controlled information about deleted files during search

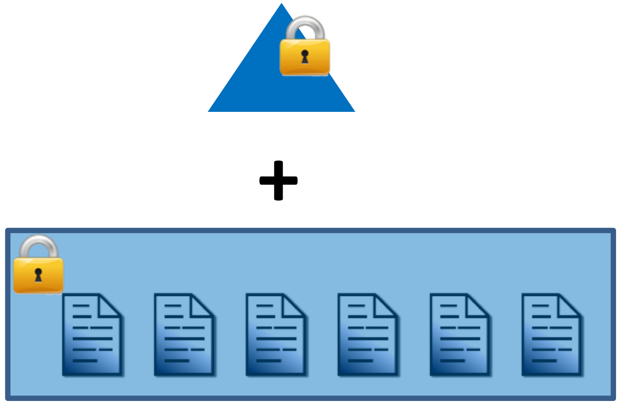
Time

1 Add (F1, **w1**)

2 Add (F2, **w1**)

3 Del (F1, **w1**)

4 Query (**w1**)



More Leakage

Backward Privacy Type – I: $\mathcal{L}^{Search}(w) = \mathcal{L}(\mathbf{TimeDB}(w))$

Backward Privacy Type – II: $\mathcal{L}^{Search}(w) = \mathcal{L}(\mathbf{TimeDB}(w), \mathbf{Updates}(w))$

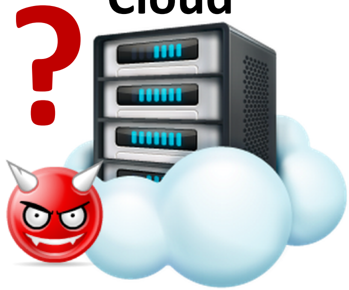
Backward Privacy Type – III: $\mathcal{L}^{Search}(w) = \mathcal{L}(\mathbf{TimeDB}(w), \mathbf{Updates}(w), \mathbf{DelHist}(w))$

Issues with Prior Forward & Backward DSE schemes

Client

Require: The client to store an operation counter for each unique keyword**!!!**

Untrusted Cloud

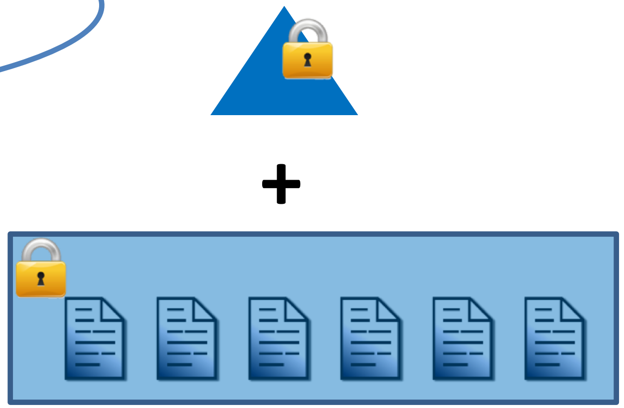


Keyword	Counter
w1	3
w2	2
w3	4
…	
wM	5

O(|W|), where |W| is the dictionary size

O(|W|) can be up to **O(N)**, where N is the total DB size

Synchronization issues: if the client wants to access the encrypted DB from multiple devices

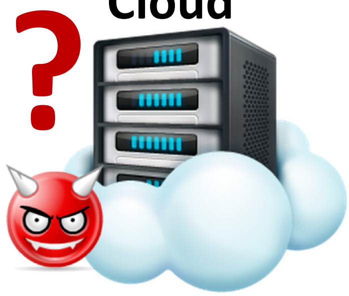


Issues with Prior Forward & Backward DSE schemes

Client

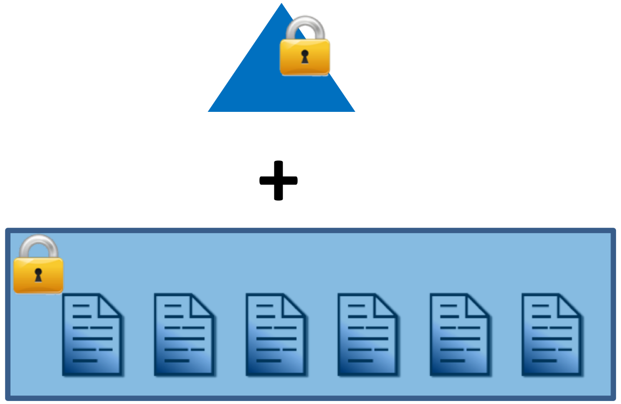
Outsourcing the operational counters to the server requires the use of Oblivious Indexes ~**extra $O(\log^2 N)$** overhead **and** **$O(\log N)$** rounds of interaction!

Untrusted Cloud



Keyword	Counter
w1	3
w2	2
w3	4
…	
wM	5

E.g., an Oblivious Hash Map (**OMAP**)



Prior state-of-the-art Works & Our Contributions

	Search	Update	Search RT	BP-Type
Moneta	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(\log^2 N)$	2	Type-I
OMAP+**Mitra**	$O(a_w + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-II
SDa	$O(a_w + \log N)$	$O(\log N)$ (am.)	1	Type-II
SDd	$O(a_w + \log N)$	$O(\log^3 N)$	1	Type-II
ORION	$O(n_w \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-I
HORUS	$O(n_w \log d_w \log N + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-III
QOS	$O(n_w \log i_w + \log^2 N)$	$O(\log^3 N)$	$O(\log N)$	Type-III

N: total number of (file, keyword) pairs, $\mathbf{a_w}$: #updates for keyword **w**

$\mathbf{n_w}$: #files **currently** containing keyword **w**, $\mathbf{i_w}$: #inserts for keyword **w**

Prior state-of-the-art Works & Our Contributions

	Search	Update	Search RT	BP-Type
Moneta	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(\log^2 N)$	2	Type-I
OMAP+Mitra	$O(a_w + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-II
SDa	$O(a_w + \log N)$	$O(\log N)$ (am.)	1	Type-II
SDd	$O(a_w + \log N)$	$O(\log^3 N)$	1	Type-II
ORION	$O(n_w \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-I
HORUS	$O(n_w \log d_w \log N + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-III
QOS	$O(n_w \log i_w + \log^2 N)$	$O(\log^3 N)$	$O(\log N)$	Type-III

SDa and **SDd** have **20x-85x** faster search time than MITRA

QOS has **14x-16531x** faster search time than HORUS

Prior state-of-the-art Works & Our Contributions

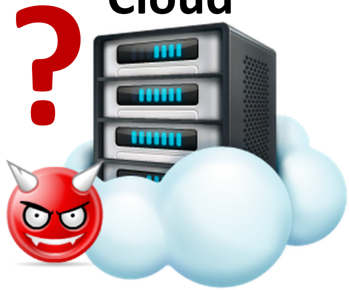
	Search	Update	Search RT	BP-Type
Moneta	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(\log^2 N)$	2	Type-I
OMAP+Mitra	$O(a_w + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-II
SDa	$O(a_w + \log N)$	$O(\log N)$ (am.)	1	Type-II
SDd	$O(a_w + \log N)$	$O(\log^3 N)$	1	Type-II
ORION	$O(n_w \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-I
HORUS	$O(n_w \log d_w \log N + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-III
QOS	$O(n_w \log i_w + \log^2 N)$	$O(\log^3 N)$	$O(\log N)$	Type-III

QOS has better search time for deletion ratios between **40%-80%**

SDa scheme

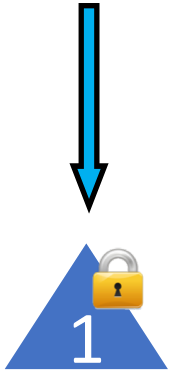
Client

Untrusted Cloud



High-level idea: Organize **N** updates in a collection of at most **$\log_2 N$** independent encrypted indexes

Add (F1, w1)

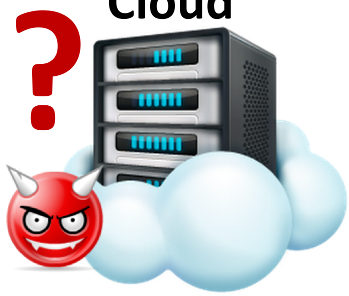


SDa scheme

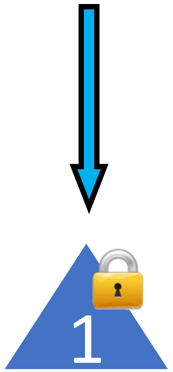
Client

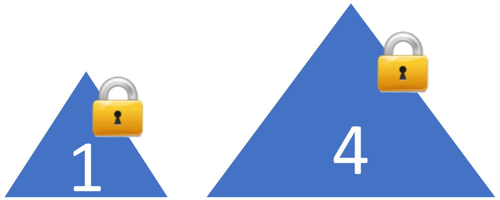
High-level idea: Organize **N** updates in a collection of at most **$\log_2 N$** independent encrypted indexes

Untrusted Cloud



Add (F4, w1)

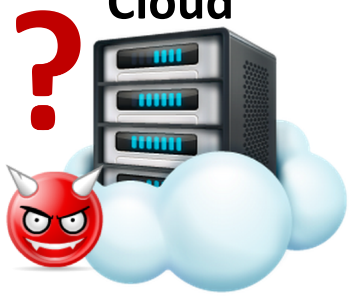




SDa scheme

Client

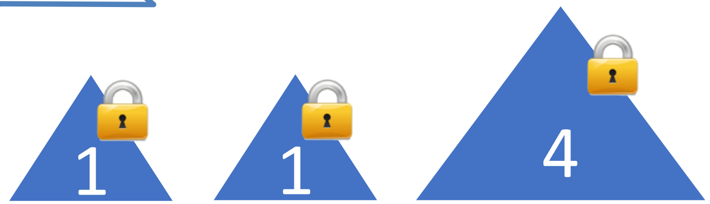
Untrusted Cloud



High-level idea: Organize **N** updates in a collection of at most **$\log_2 N$** independent encrypted indexes

Add (F4, w1)

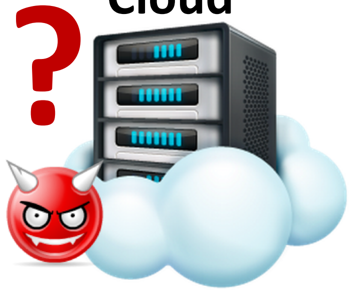
If we keep adding the number of encrypted indexes will be **O(N)**



SDa scheme

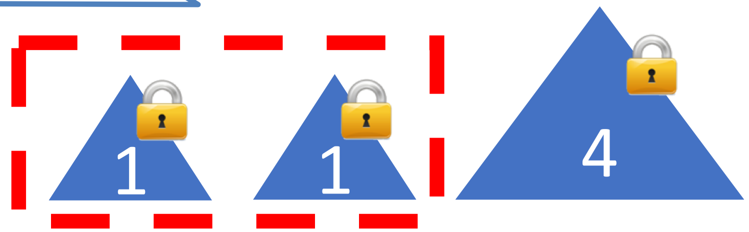
Client

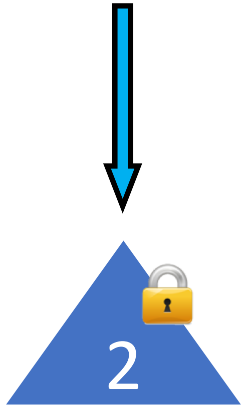
Untrusted Cloud



High-level idea: Organize **N** updates in a collection of at most **$\log_2 N$** independent encrypted indexes

Whenever **two** indexes of the same size exist, **download them and merge** them in a new index





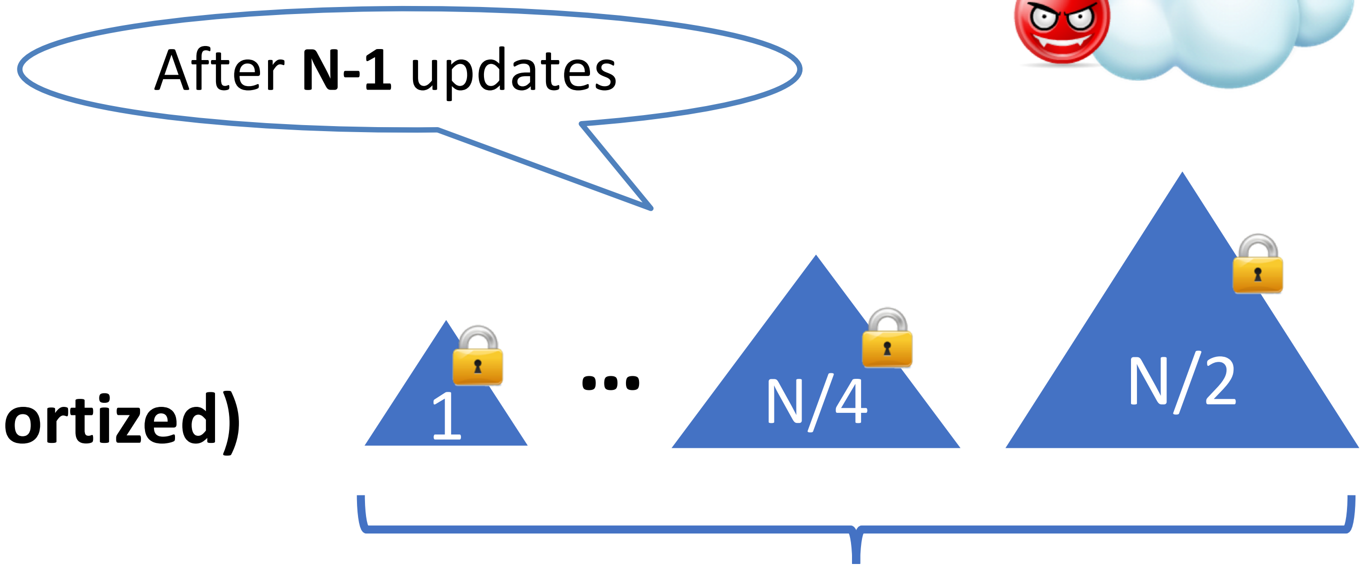
*Assuming that **N** is a power of **2**

SDa scheme

Client

Untrusted Cloud

High-level idea: Organize **N** updates in a collection of at most **$\log_2$N** independent encrypted indexes



Update cost = O($\log_2$N) (amortized)

O($\log_2$N)
encrypted indexes

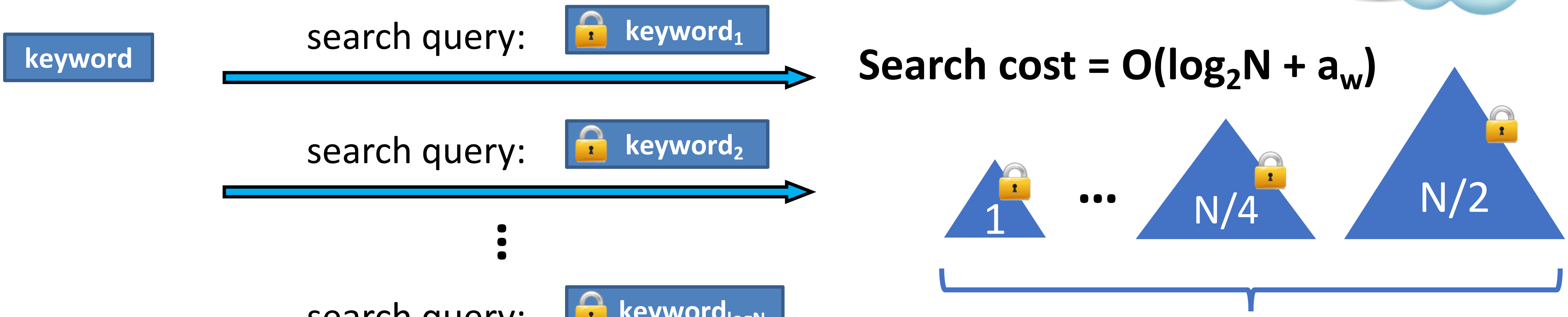
SDa scheme

*Assuming that **N** is a power of **2**

Client

High-level idea: Organize **N** updates in a collection of at most **$\log_2 N$** independent encrypted indexes

Untrusted Cloud

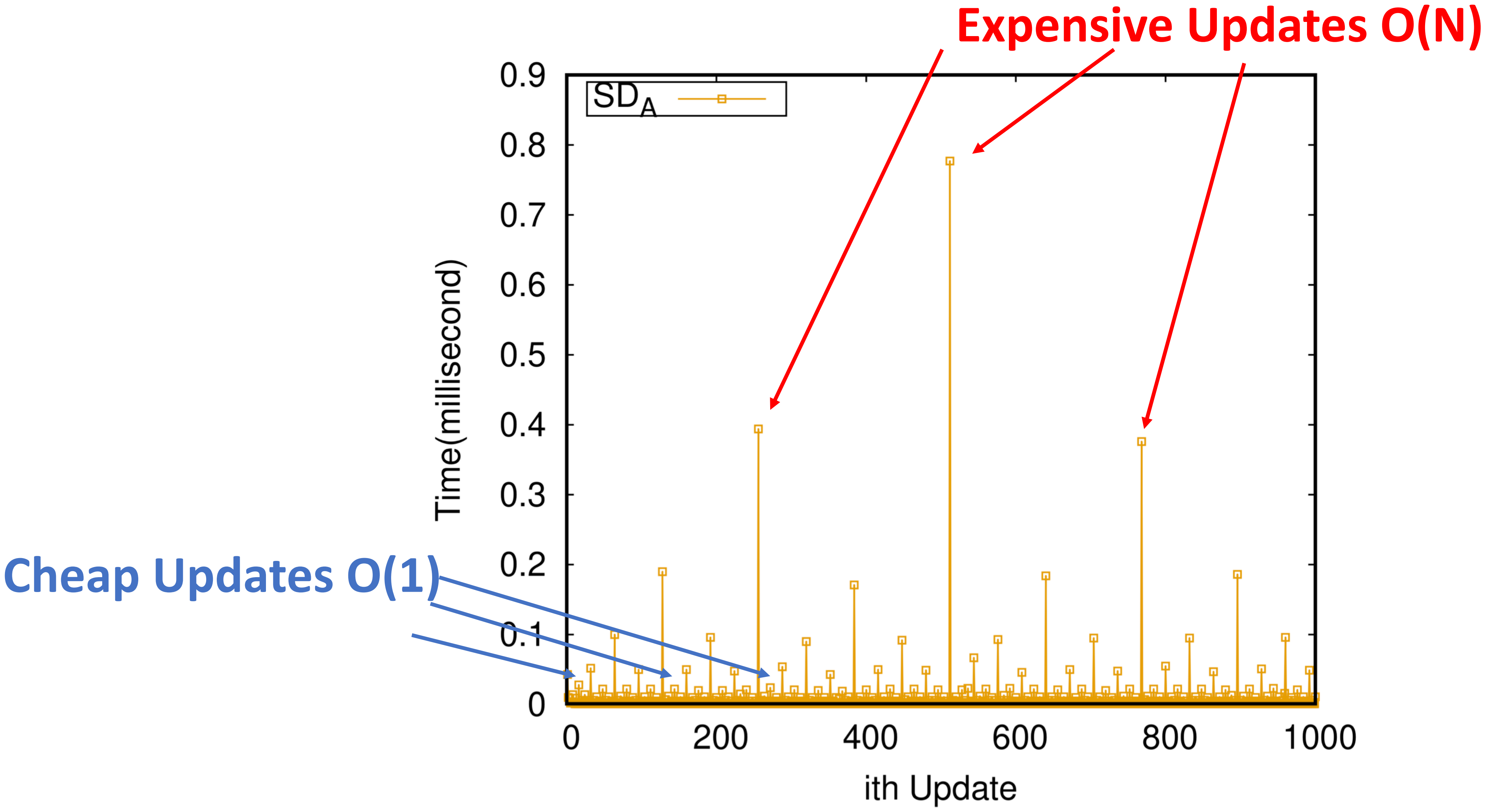


Search cost = $O(\log_2 N + a_w)$

$O(\log_2 N)$ encrypted indexes

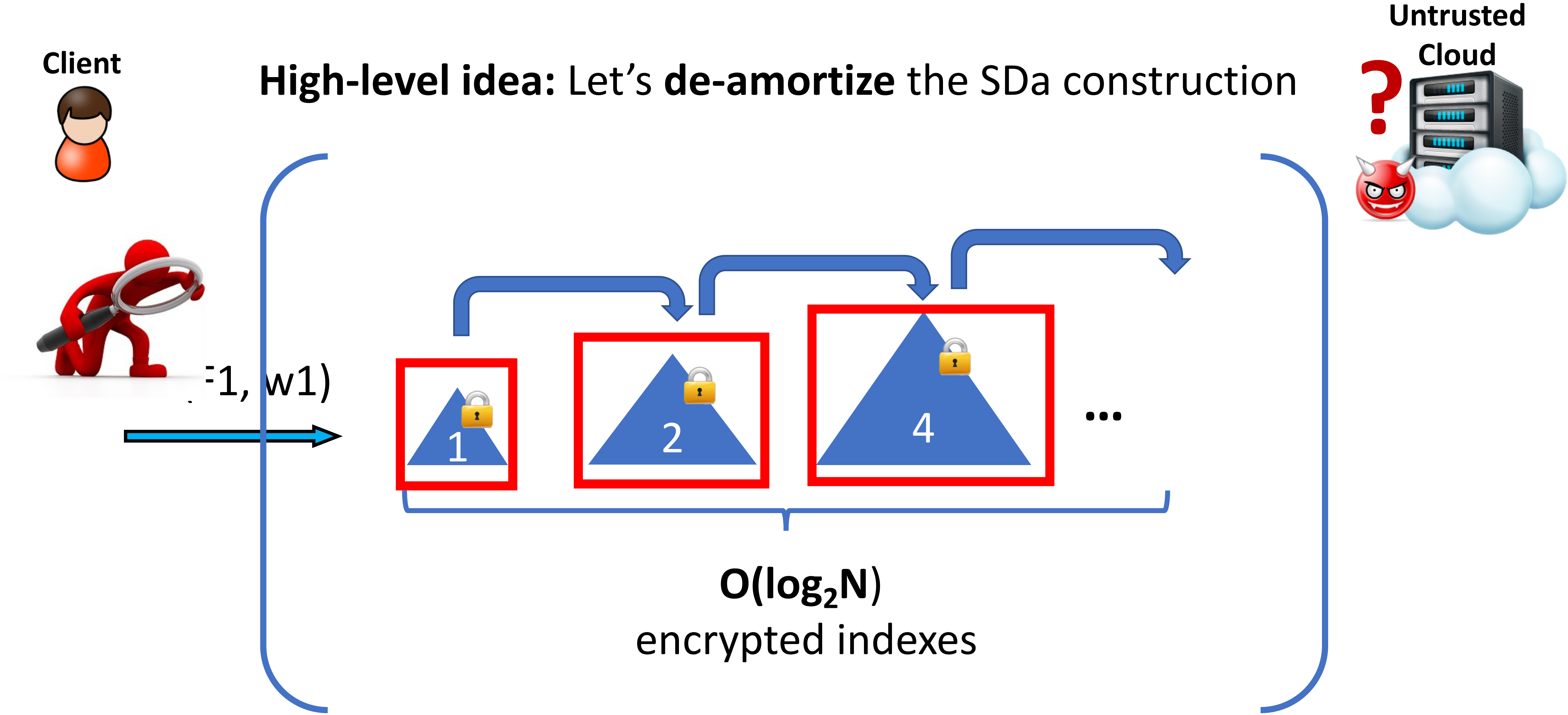
Intuition Forward/Backward privacy: Every index is built with a fresh key and the used static SE is response hiding!!!

SDa --- Amortized Update Cost



SDd scheme

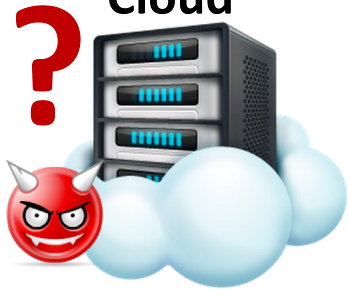
High-level idea: Let's **de-amortize** the SDa construction

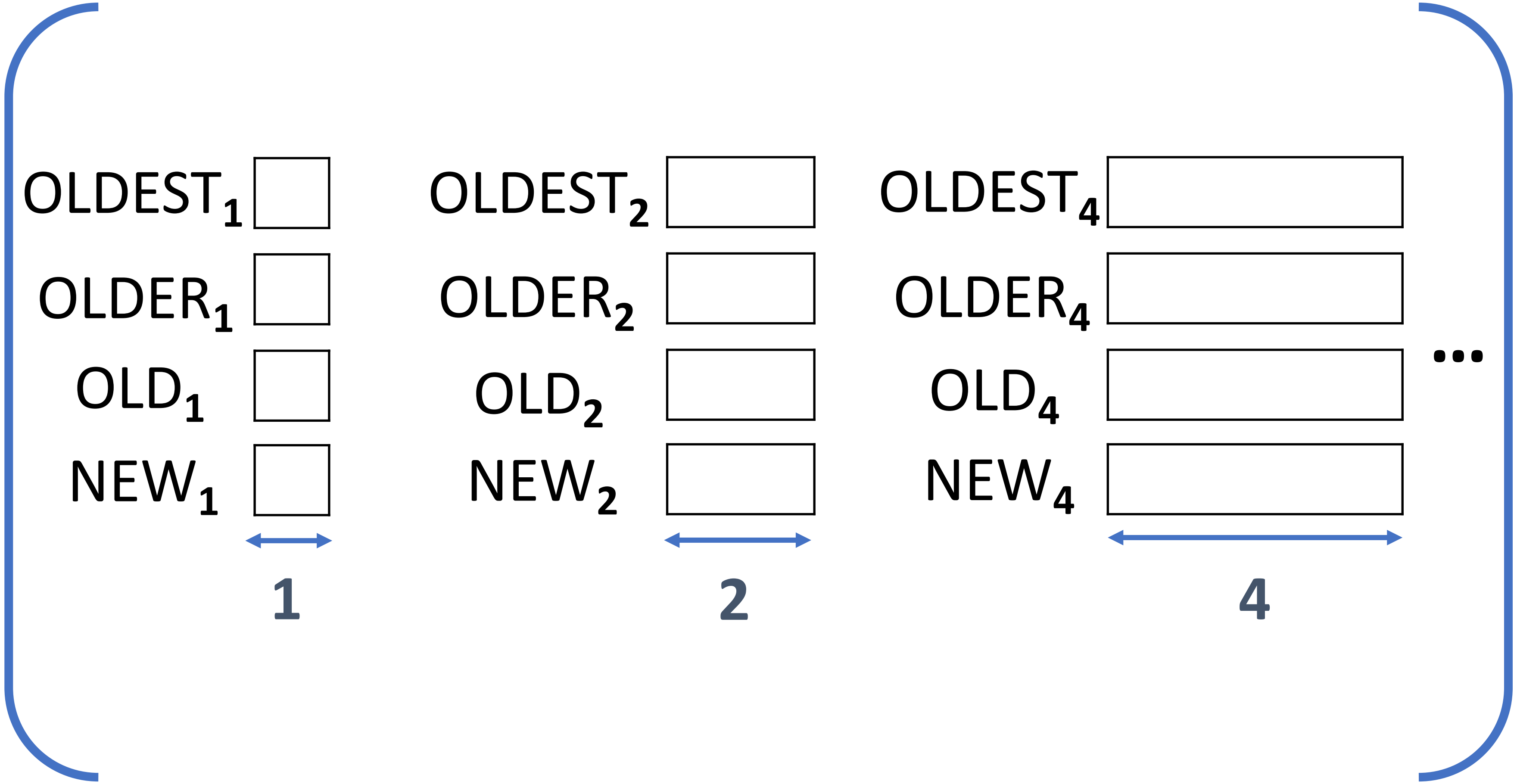


SDd scheme

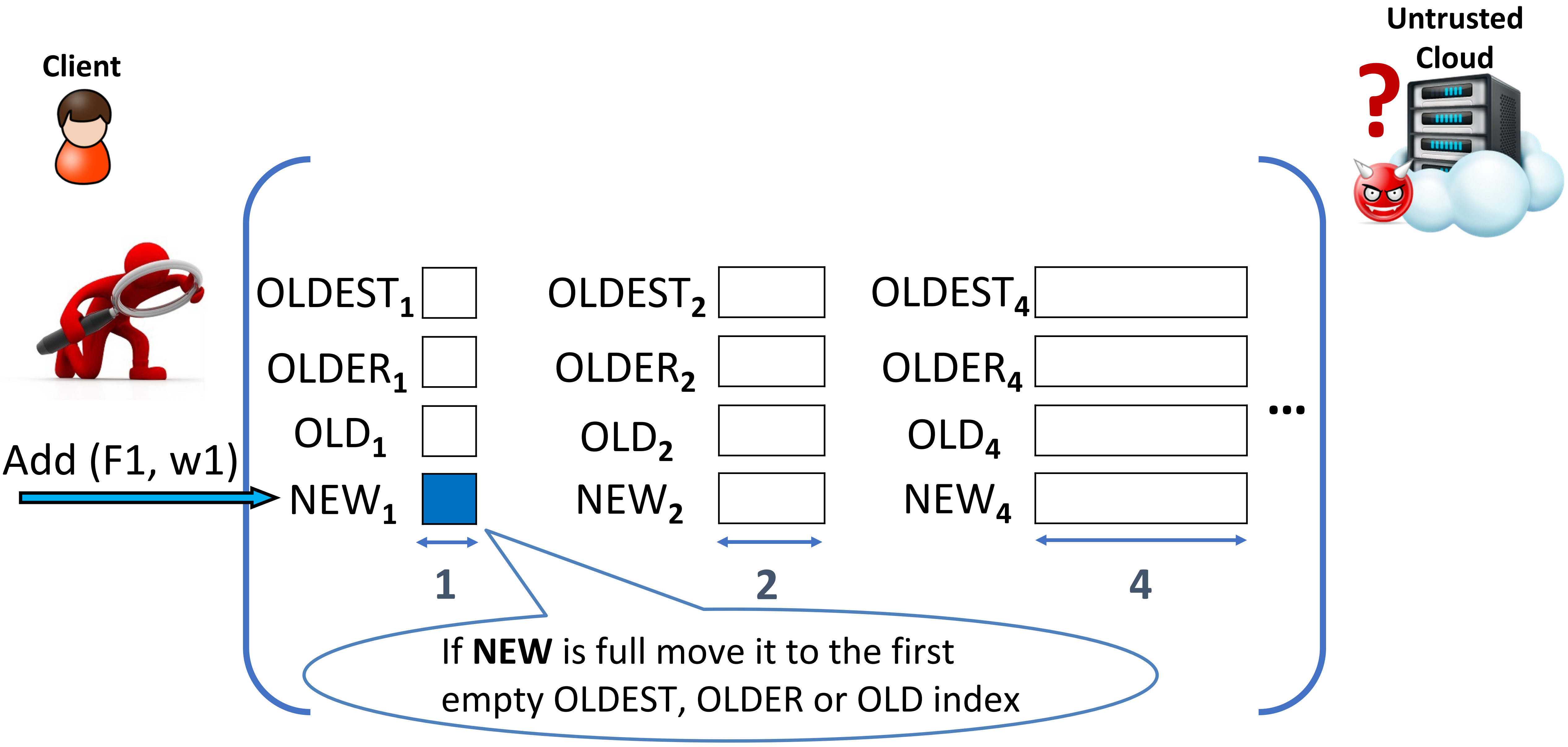
Client

Untrusted Cloud

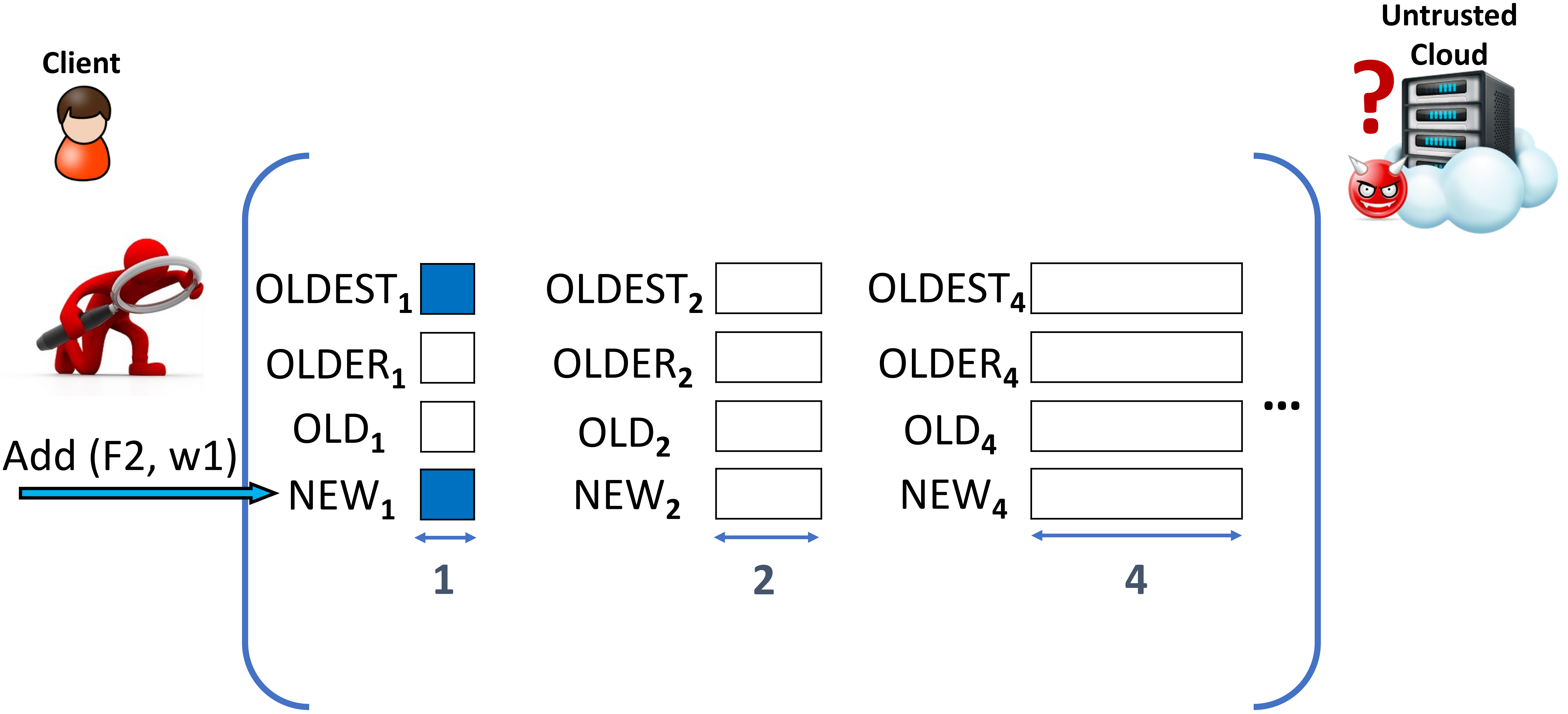




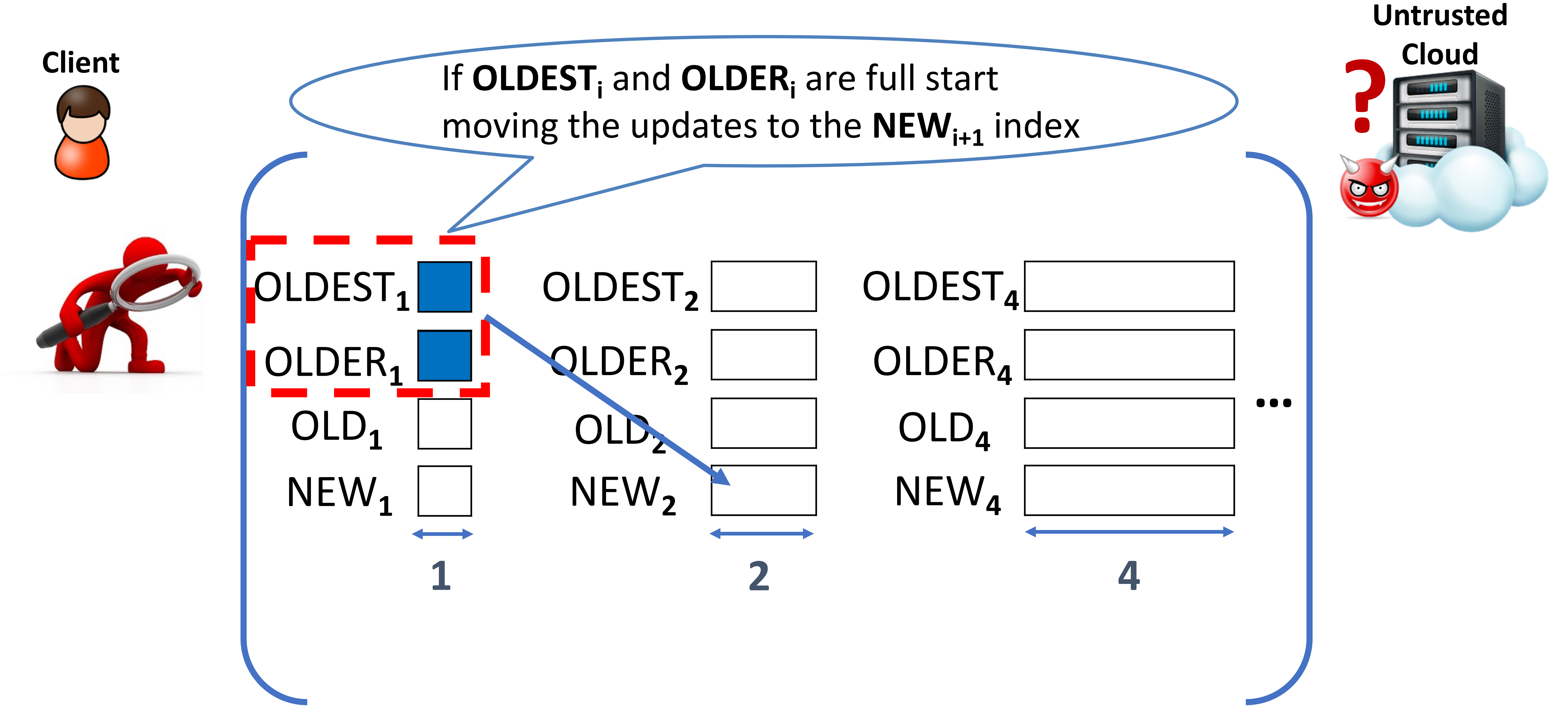
SDd scheme


Client
Untrusted
Cloud
?
Add (F1, w1)
$OLDEST_1$
$OLDER_1$
OLD_1
NEW_1
$OLDEST_2$
$OLDER_2$
OLD_2
NEW_2
$OLDEST_4$
$OLDER_4$
OLD_4
NEW_4
…
1
2
4
If **NEW** is full move it to the first empty OLDEST, OLDER or OLD index

SDd scheme

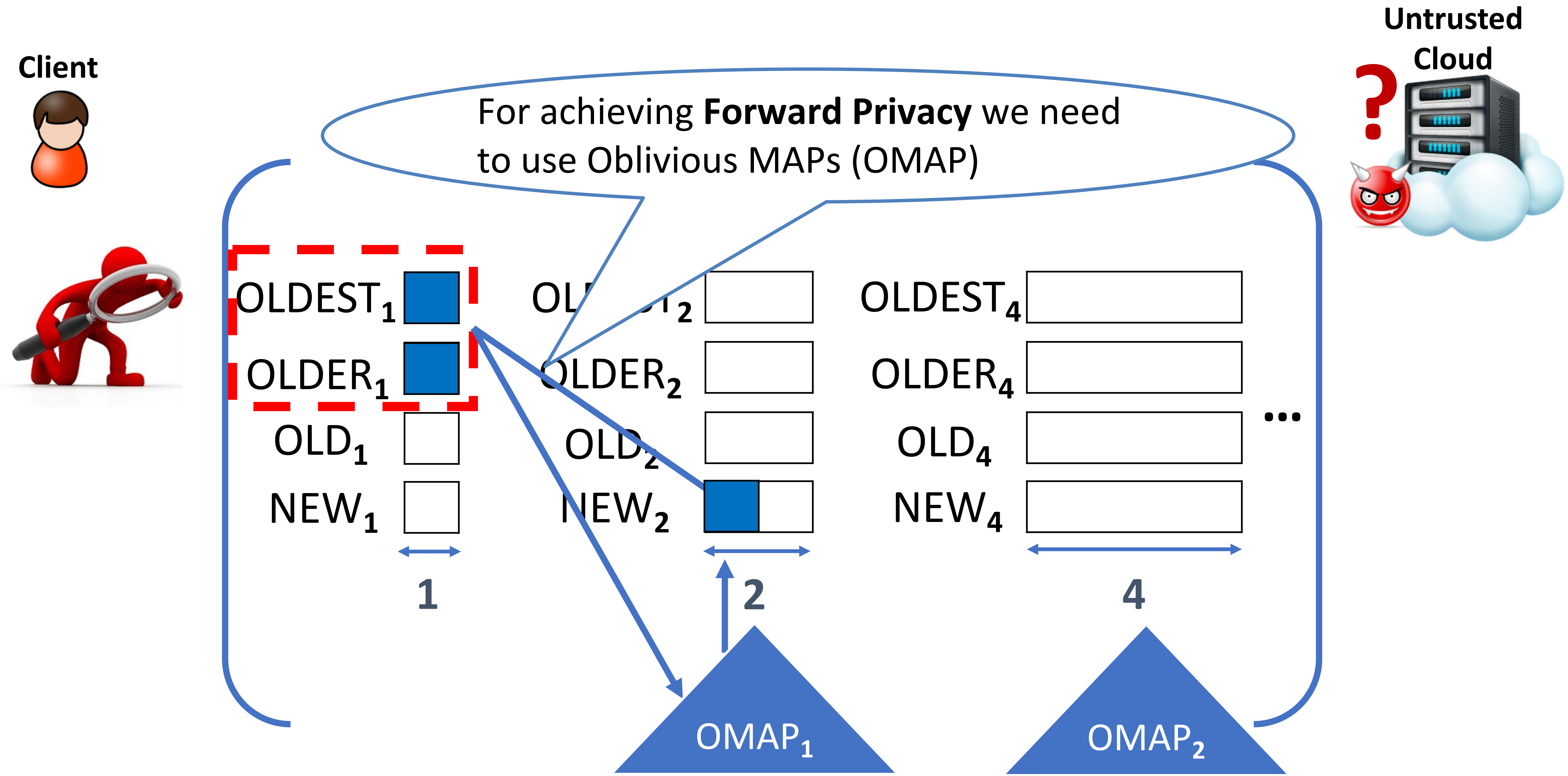

Client
Untrusted Cloud
?
Add (F2, w1)
OLDEST$_1$
OLDER$_1$
OLD$_1$
NEW$_1$
1
OLDEST$_2$
OLDER$_2$
OLD$_2$
NEW$_2$
2
OLDEST$_4$
OLDER$_4$
OLD$_4$
NEW$_4$
4
…

SDd scheme

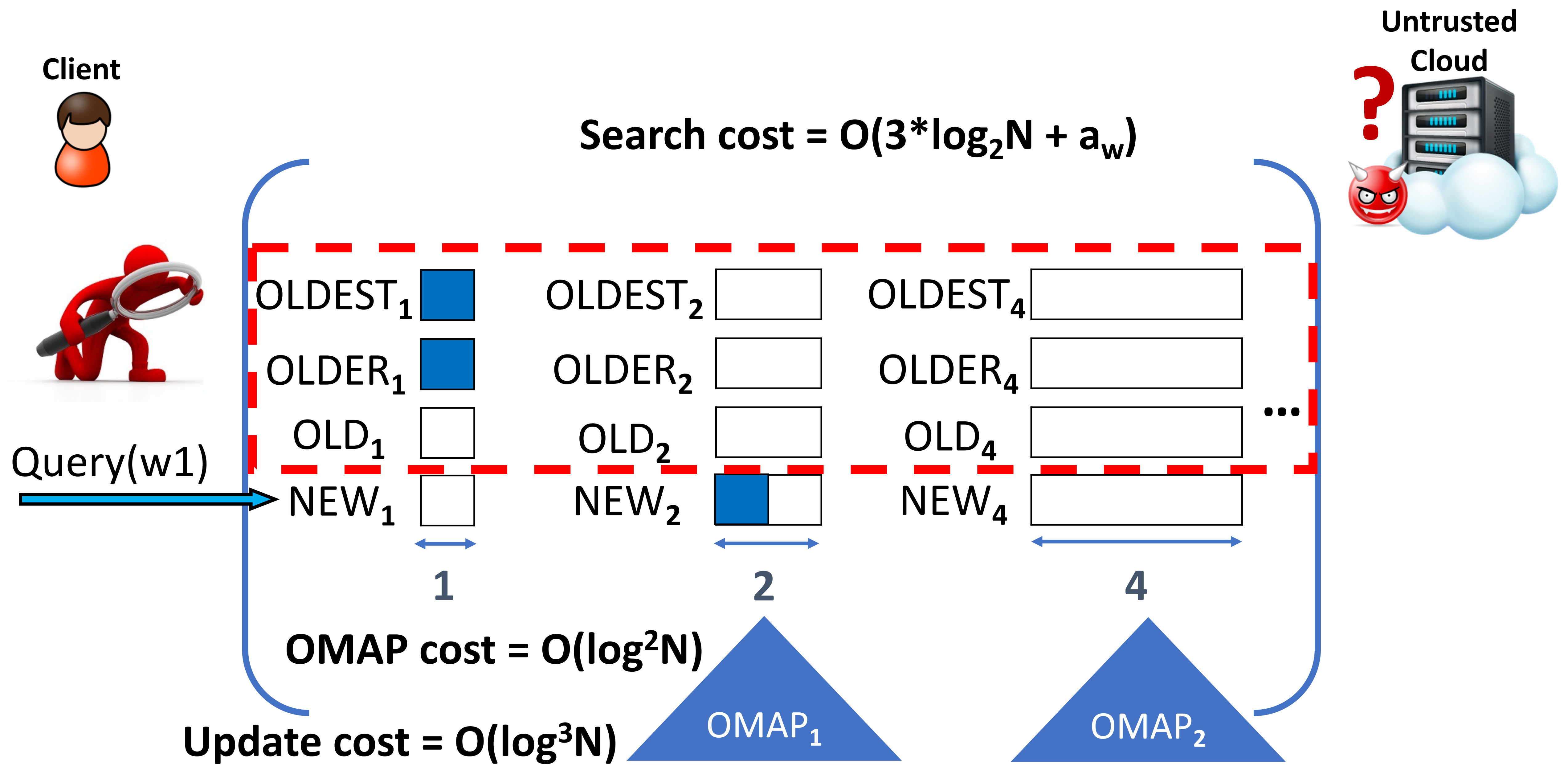


Client
Untrusted Cloud
If OLDEST$_i$ and OLDER$_i$ are full start moving the updates to the NEW$_{i+1}$ index
OLDEST$_1$
OLDER$_1$
OLD$_1$
NEW$_1$
OLDEST$_2$
OLDER$_2$
OLD$_2$
NEW$_2$
OLDEST$_4$
OLDER$_4$
OLD$_4$
NEW$_4$
…
1
2
4

SDd scheme


Client
Untrusted Cloud
?
For achieving **Forward Privacy** we need to use Oblivious MAPs (OMAP)
OLDEST$_1$
OLDER$_1$
OLD$_1$
NEW$_1$
1
OLDER$_2$
NEW$_2$
2
OLDEST$_4$
OLDER$_4$
OLD$_4$
NEW$_4$
4
...
OMAP$_1$
OMAP$_2$

SDd scheme



Untrusted Cloud
Client
Search cost = $O(3*\log_2 N + a_w)$
$OLDEST_1$
$OLDER_1$
OLD_1
NEW_1
$OLDEST_2$
$OLDER_2$
OLD_2
NEW_2
$OLDEST_4$
$OLDER_4$
OLD_4
NEW_4
...
Query(w1)
1
2
4
OMAP cost = $O(\log^2 N)$
$OMAP_1$
$OMAP_2$
Update cost = $O(\log^3 N)$

Prior state-of-the-art Works & Our Contributions

	Search	Update	Search RT	BP-Type
Moneta	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(\log^2 N)$	2	Type-I
OMAP+Mitra	$O(a_w + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-II
SDa	$O(a_w + \log N)$	$O(\log N)$ (am.)	1	Type-II
SDd	$O(a_w + \log N)$	$O(\log^3 N)$	1	Type-II
ORION	$O(n_w \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-I
HORUS	$O(n_w \log d_w \log N + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-III
QOS	$O(n_w \log i_w + \log^2 N)$	$O(\log^3 N)$	$O(\log N)$	Type-III

- ***Definition:*** *A DSE scheme has optimal (resp. quasi-optimal) search time, if the asymptotic complexity of Search is* $\boldsymbol{O(n_w)}$ *(resp.* $\boldsymbol{O(n_w\, \mathrm{polylog}(N))}$*).*

Motivation for Optimal/Quasi-optimal Search

Client

Add (F1, **w1**)

Add (F2, **w1**)

⋮

Add (FN, **w1**)

Del (F1, **w1**)

⋮

Del (FN-1, **w1**)

Query(**w1**)

SDd search cost
= $O(\log_2 N + a_w)$

Untrusted Cloud

+

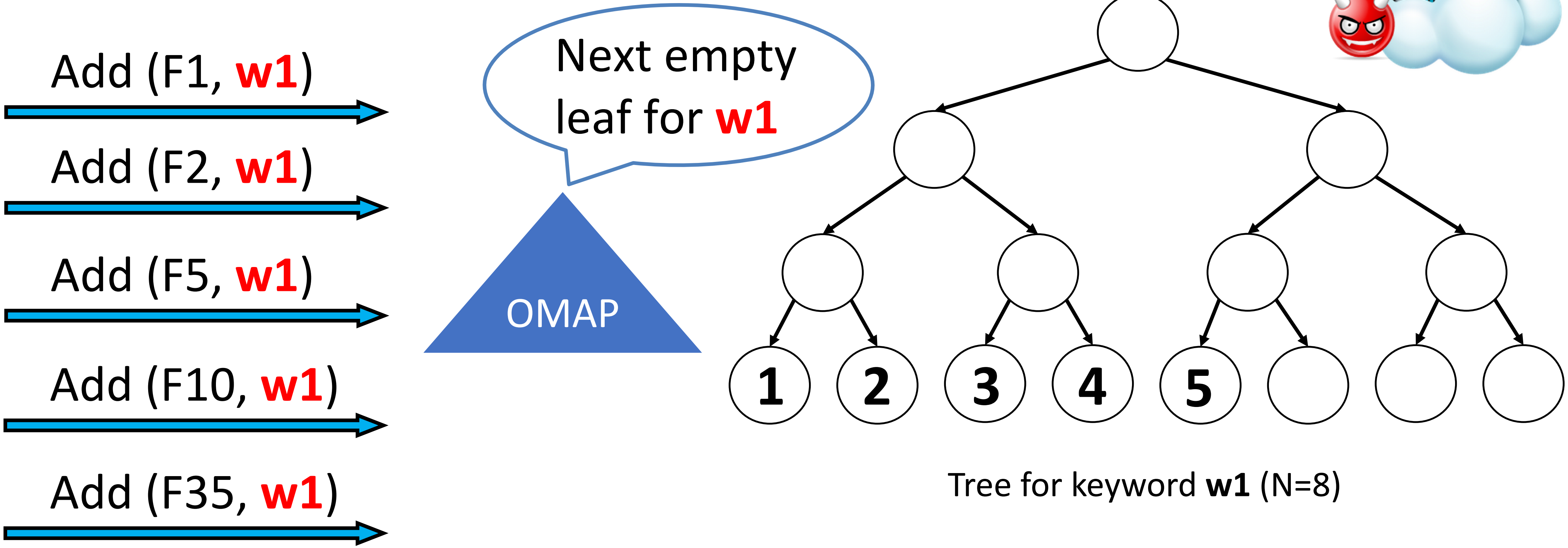
w1 is contained only in **File N**, but the result has size **$O(a_w)$** ~= **O(N)**

QOS --- Main Idea

Client

Untrusted Cloud

Idea: For each keyword **w** create a data structure that helps us avoid the deleted regions



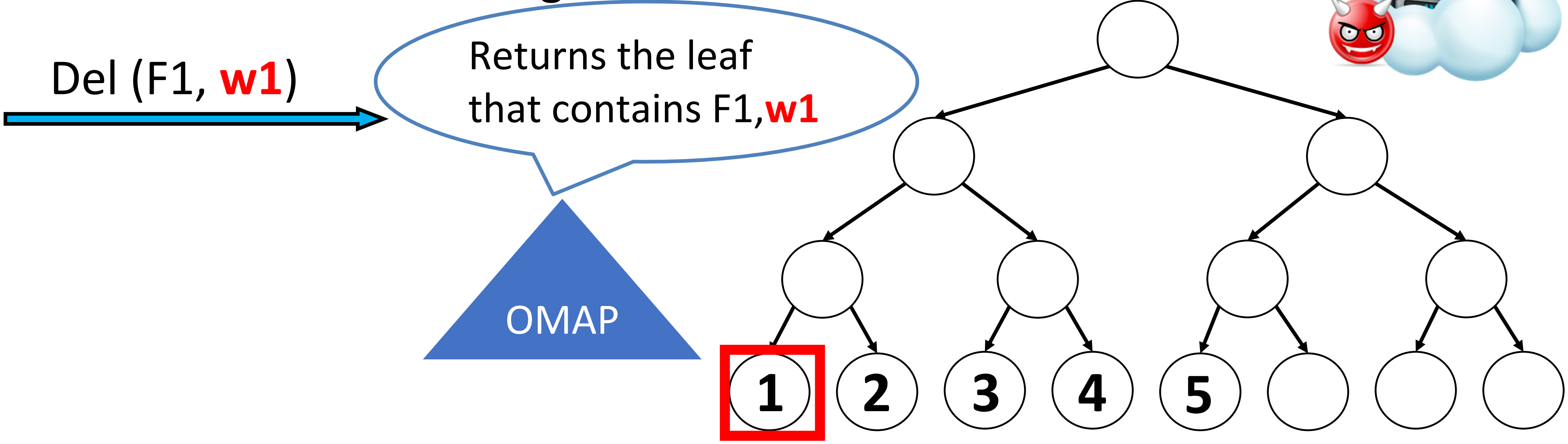
Tree for keyword **w1** (N=8)

QOS --- Main Idea

Client

Untrusted Cloud

Idea: For each keyword **w** create a data structure that helps us avoid the deleted regions



Tree for keyword **w1** (N=8)

QOS --- Main Idea

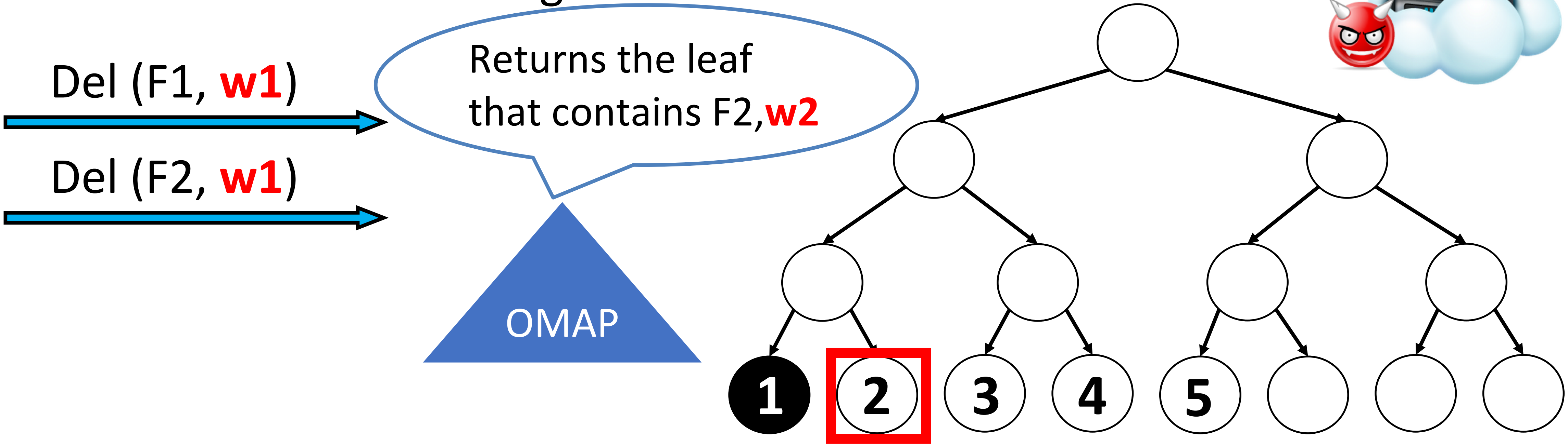
Client

Idea: For each keyword **w** create a data structure that helps us avoid the deleted regions

Untrusted Cloud

Del (F1, **w1**)

Del (F2, **w1**)



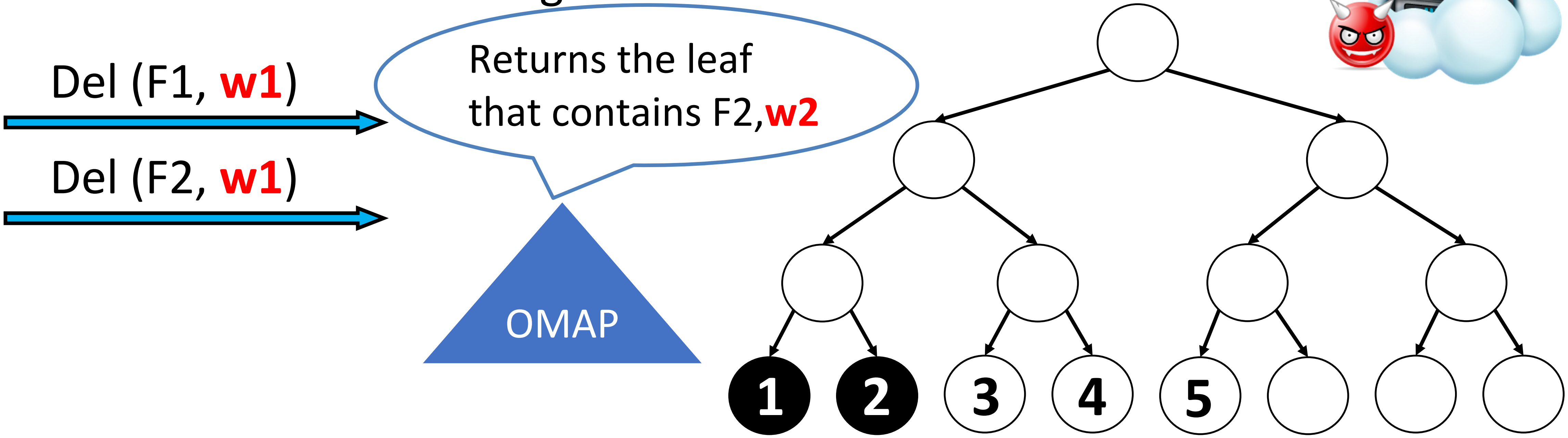
Tree for keyword **w1** (N=8)

QOS --- Main Idea

Client

Untrusted Cloud

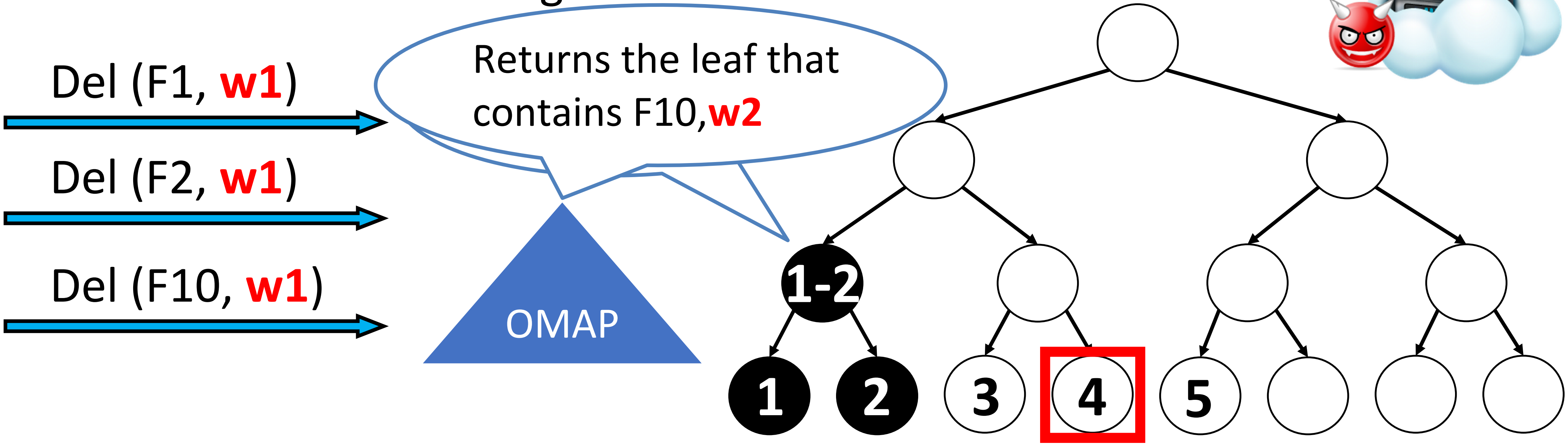
Idea: For each keyword **w** create a data structure that helps us avoid the deleted regions



Tree for keyword **w1** (N=8)

QOS --- Main Idea

<u>Idea:</u> For each keyword **w** create a data structure that helps us avoid the deleted regions

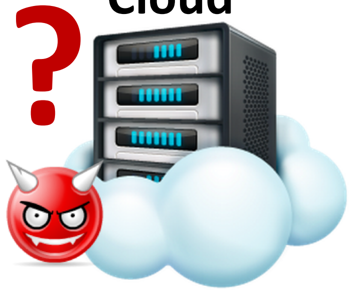


Tree for keyword **w1** (N=8)

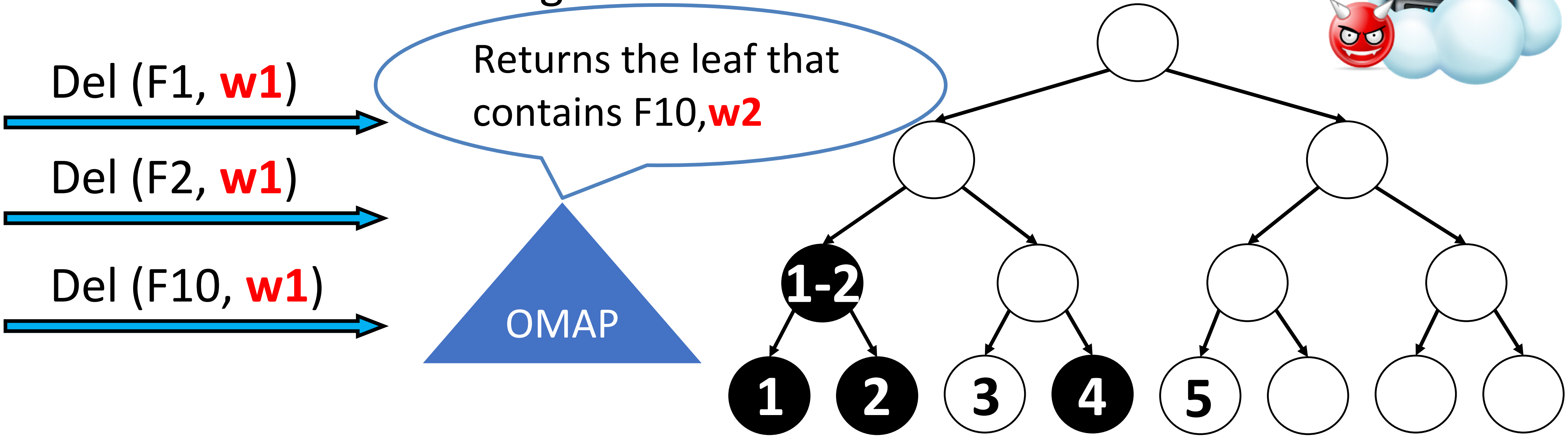
QOS --- Main Idea

Client

Untrusted Cloud



Idea: For each keyword **w** create a data structure that helps us avoid the deleted regions

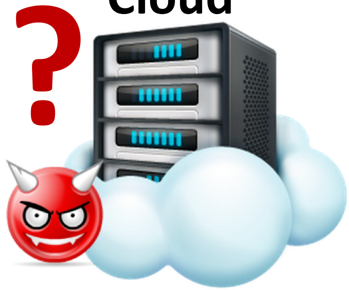


Tree for keyword **w1** (N=8)

QOS --- Main Idea

Client

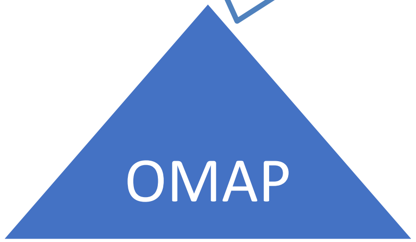
Untrusted Cloud

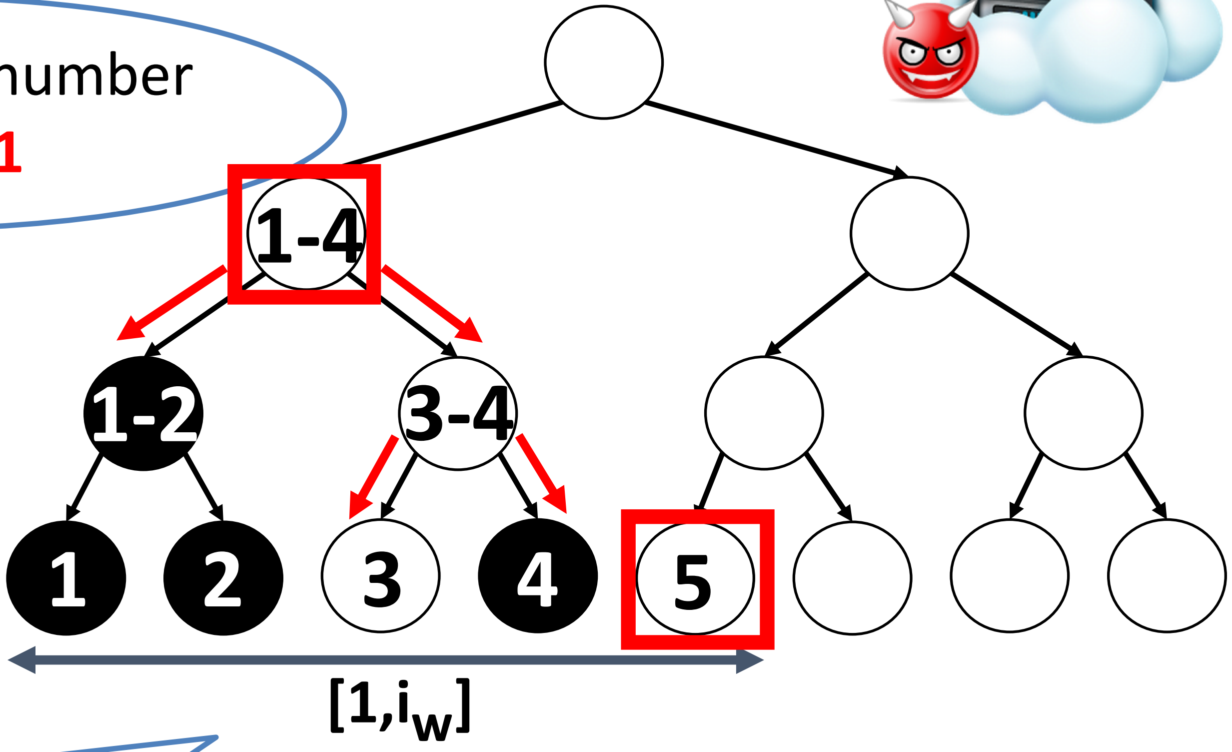


Idea: For each keyword **w** create a data structure that helps us avoid the deleted regions

Query(**w1**)

Returns $\mathbf{i_w}$ the number of inserts for **w1**





Search cost = $O(\log^2 N + n_w \log i_w)$

Computes the **Best Range Cover** of **$[1,i_w]$**

Experimental Evaluation

- We implemented **SDa**, **SDd** and **QOS** in C++
  - OpenSSL for cryptographic operations
  - AES-NI enabled

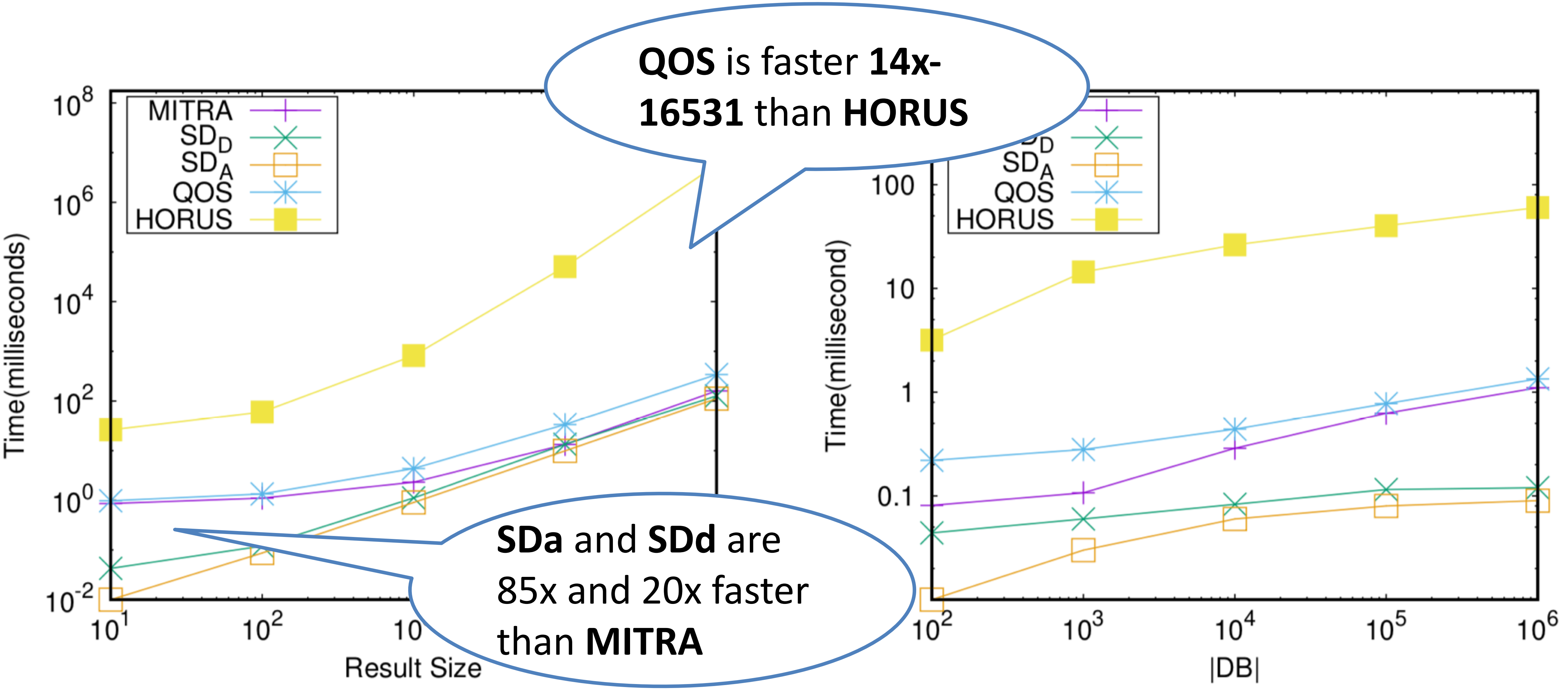
- We compare our schemes with the previous state-of-the-art DSE schemes
  - **HORUS** and **MITRA**

- We measured search time, update time, and communication size
  - Synthetic dataset of **100** to **100M** records
  - Real dataset of **6M** crime incidents in Chicago

- Experiments using r5.8xlarge AWS machines
  - 32-core Intel Xeon 8259CL 2.5GHz processor
  - Running Ubuntu 16.04 LTS, with 256GB RAM, 100GB SSD (GP2), and AES-NI enabled.

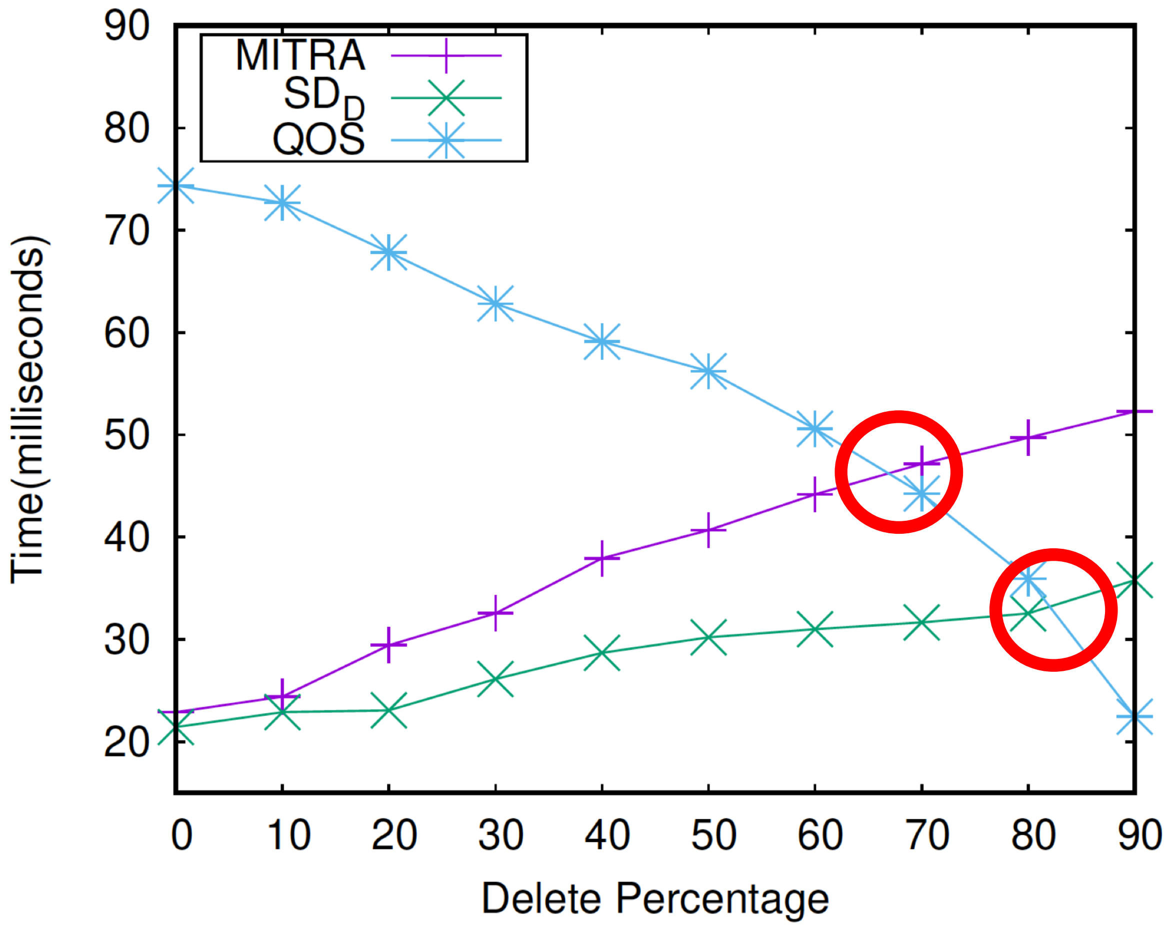
Our code is available here: `https://github.com/jgharehchamani/Small-Client-SSE`

Search Time with 10% Deletions



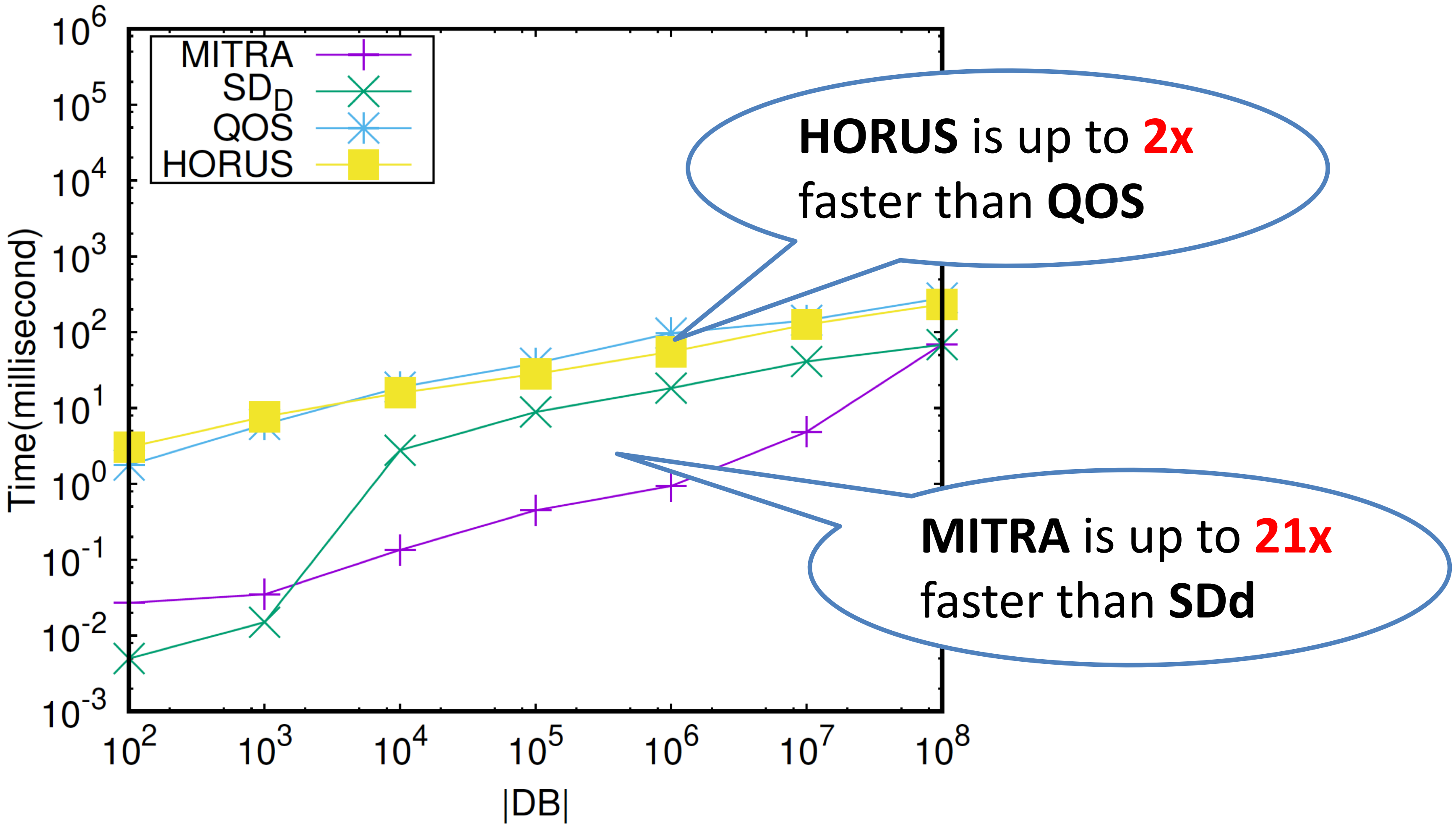
- Synthetic Dataset **1M records**
- Result size **100 records**

Search Time with 0-90% Deletion Percentage



- Synthetic Dataset **1M records** and **i_w=20K**

Update time



- Synthetic Dataset **100-100M records**

Conclusion

	Search	Update	Search RT	BP-Type
Moneta	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(\log^2 N)$	2	Type-I
OMAP+Mitra	$O(a_w + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-II
SDa	$O(a_w + \log N)$	$O(\log N)$ (am.)	1	Type-II
SDd	$O(a_w + \log N)$	$O(\log^3 N)$	1	Type-II
ORION	$O(n_w \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-I
HORUS	$O(n_w \log d_w \log N + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-III
QOS	$O(n_w \log i_w + \log^2 N)$	$O(\log^3 N)$	$O(\log N)$	Type-III

SDa and SDd

- **20x-85x** faster search time than MITRA
- Minimal client storage & non-interactive search

QOS (for delete intensive query workloads)

- is the state-of-the-art DSE with **quasi-optimal** search time
- **14x-16531x** faster search time than HORUS
- better search time than MITRA and SDd after different deletion ratios between **40%-80%**

Thank You! Questions? 

	Search	Update	Search RT	BP-Type
Moneta	$\tilde{O}(a_w \log N + \log^3 N)$	$\tilde{O}(\log^2 N)$	2	Type-I
OMAP+Mitra	$O(a_w + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-II
SDa	$O(a_w + \log N)$	$O(\log N)$ (am.)	1	Type-II
SDd	$O(a_w + \log N)$	$O(\log^3 N)$	1	Type-II
ORION	$O(n_w \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-I
HORUS	$O(n_w \log d_w \log N + \log^2 N)$	$O(\log^2 N)$	$O(\log N)$	Type-III
QOS	$O(n_w \log i_w + \log^2 N)$	$O(\log^3 N)$	$O(\log N)$	Type-III

SDa and SDd

- **20x-85x** faster search time than MITRA
- Minimal client storage & non-interactive search

QOS **(for delete intensive query workloads)**

- is the state-of-the-art DSE with **quasi-optimal** search time
- **14x-16531x** faster search time than HORUS
- better search time than MITRA and SDd after different deletion ratios between **40%-80%**