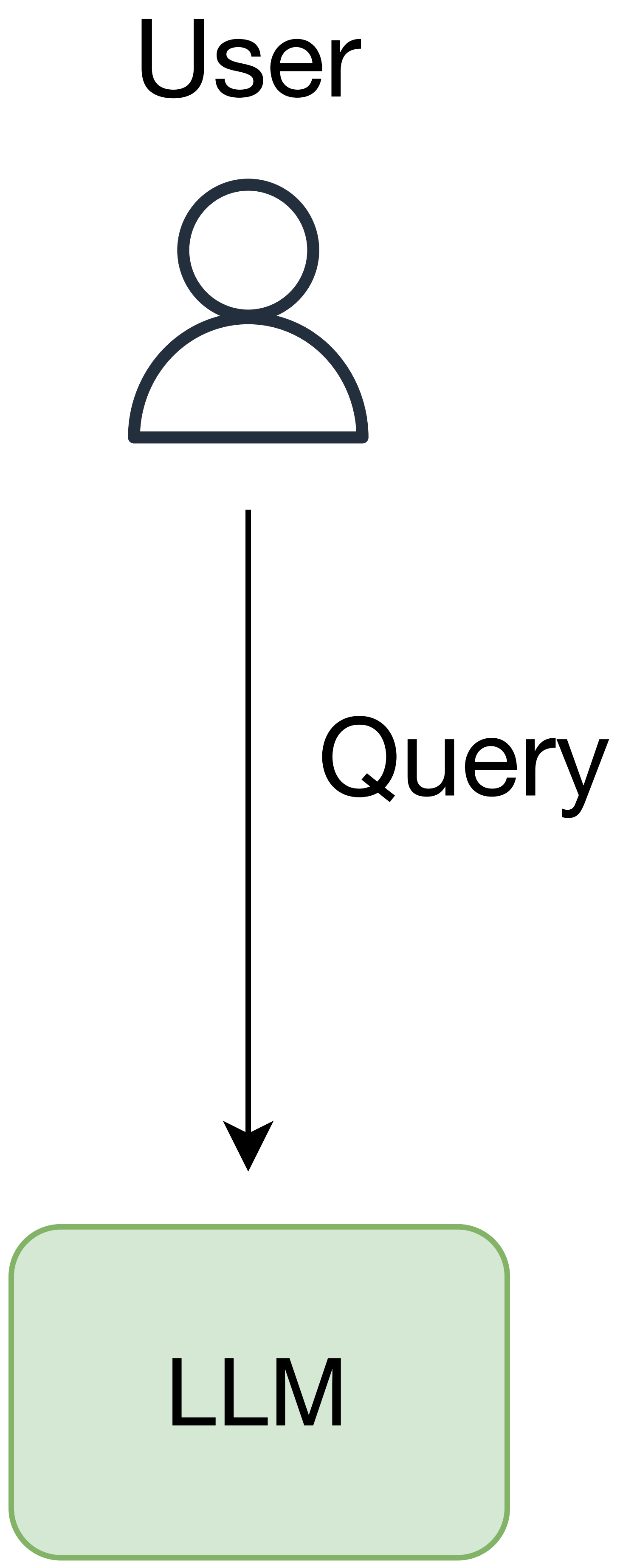
Proximity: Approximate Caching for RAGs

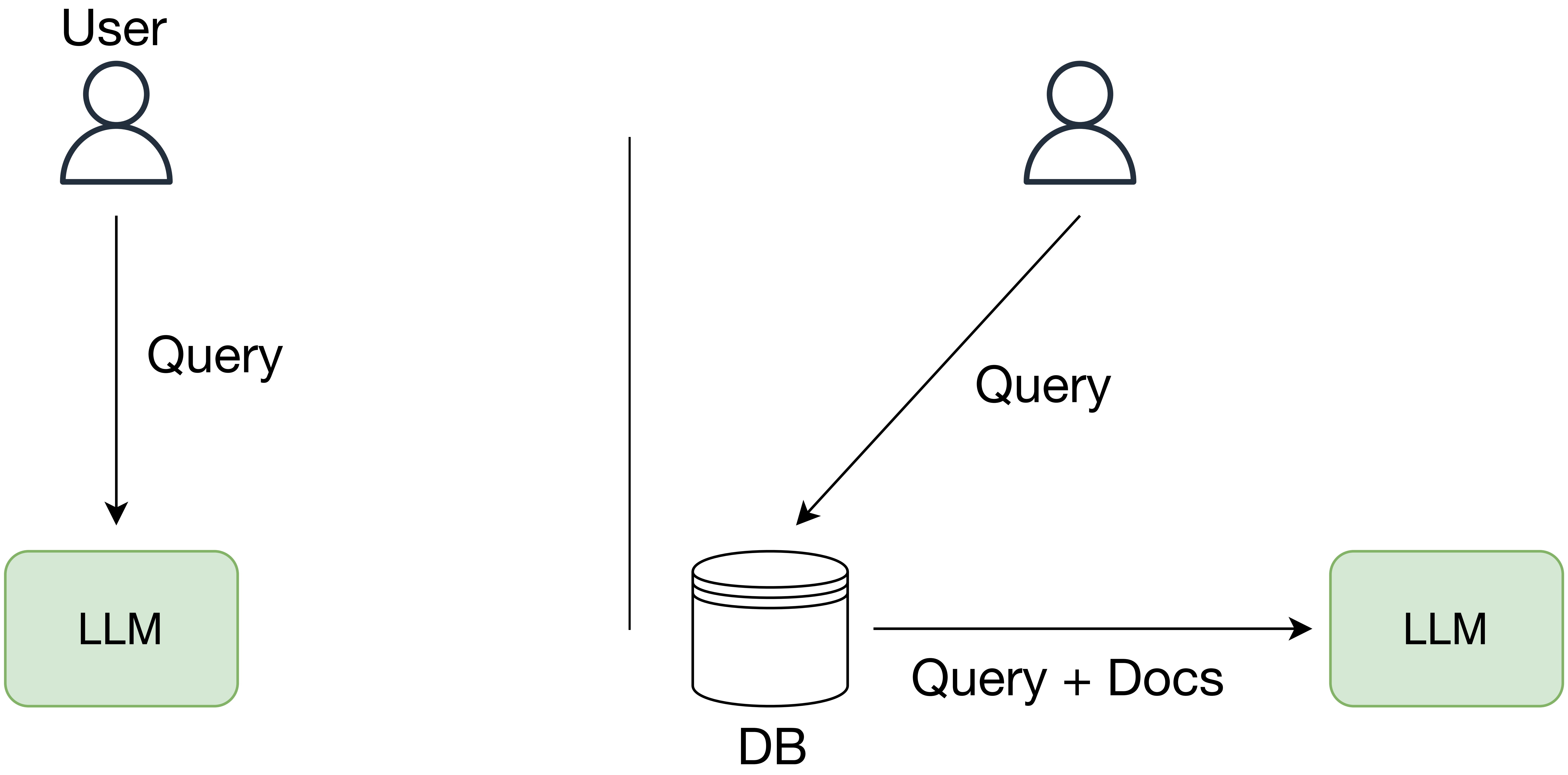
Shai Bergman
Zhang Ji
(Huawei Research)

Anne-Marie Kermarrec
Diana Petrescu
Rafael Pires
Mathis Randl
Martijn de Vos
(EPFL)

Context: RAG pipeline



Context: RAG pipeline

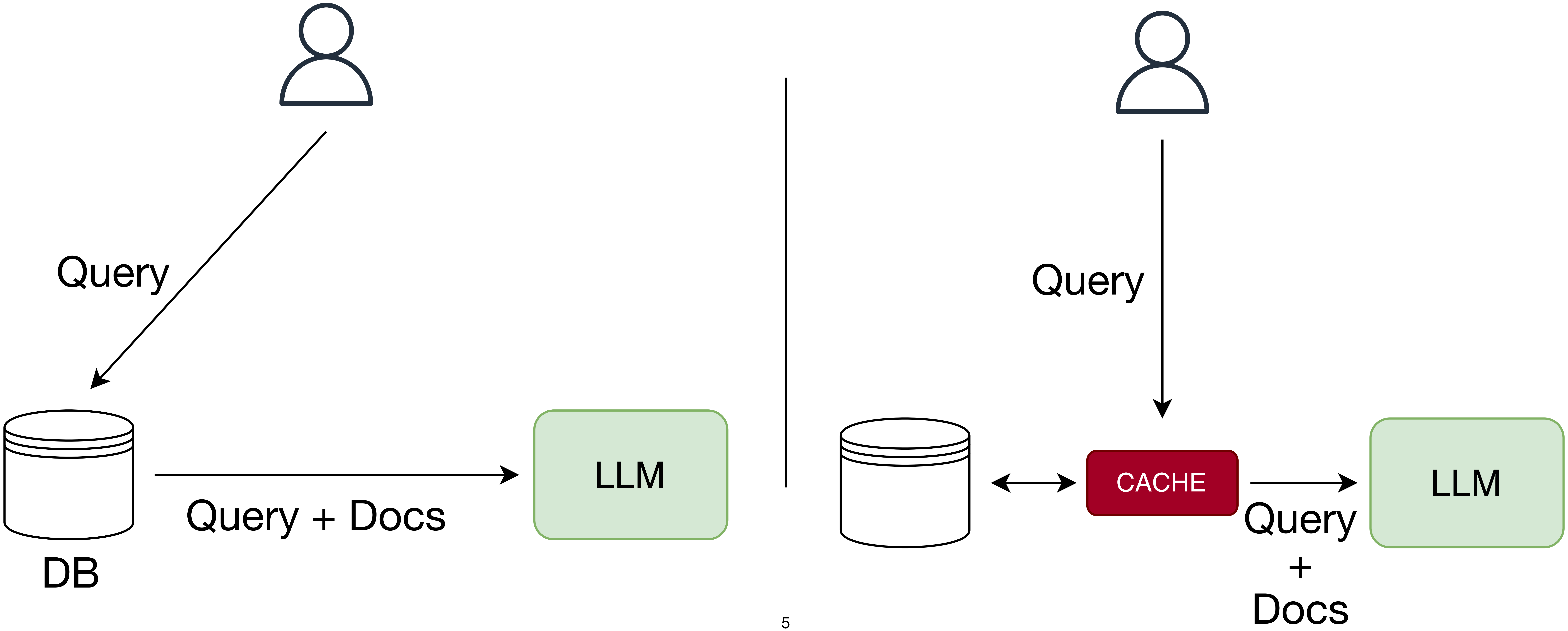


Databases can be very slow

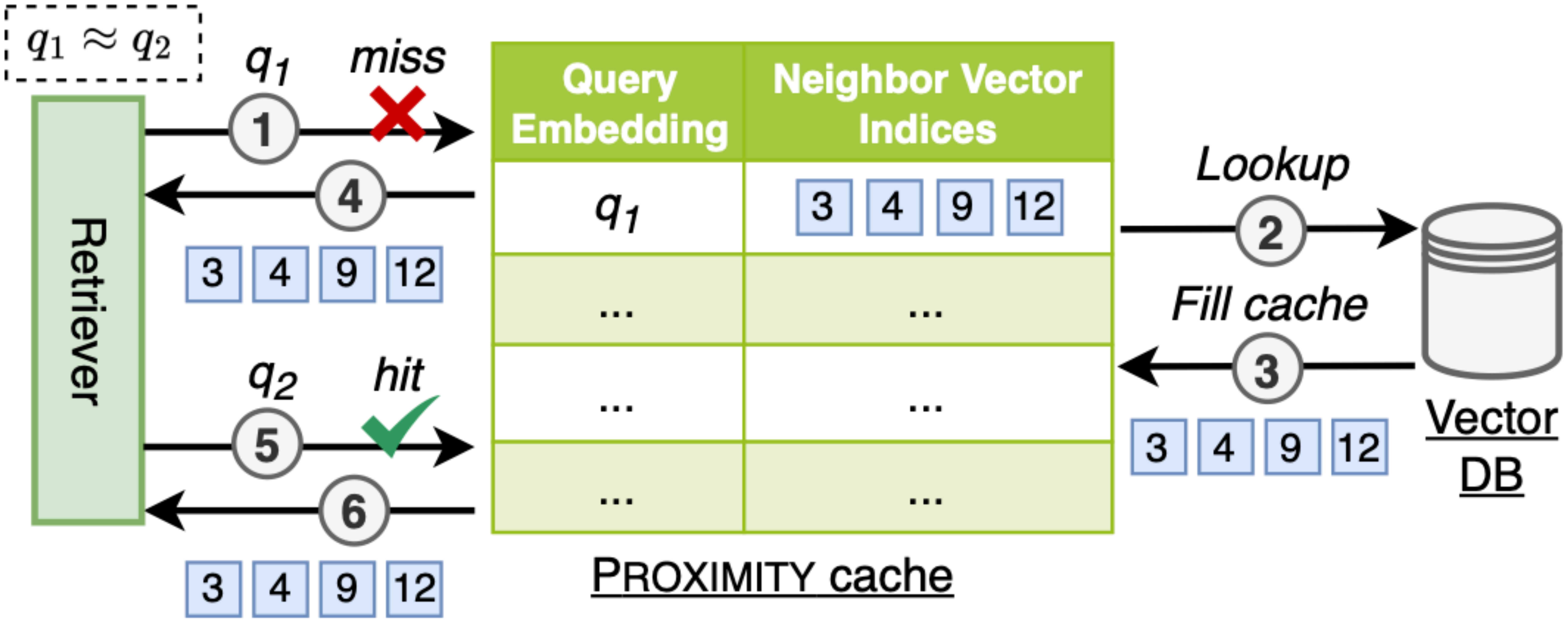
Nearest neighbor problem is hard!

- On a ~20M vector dataset (PubMed)
  - … approximate search is ~20 milliseconds
  - … exact search is ~4s!
- … and query similarities are unexploited

Proposition: Add a cache

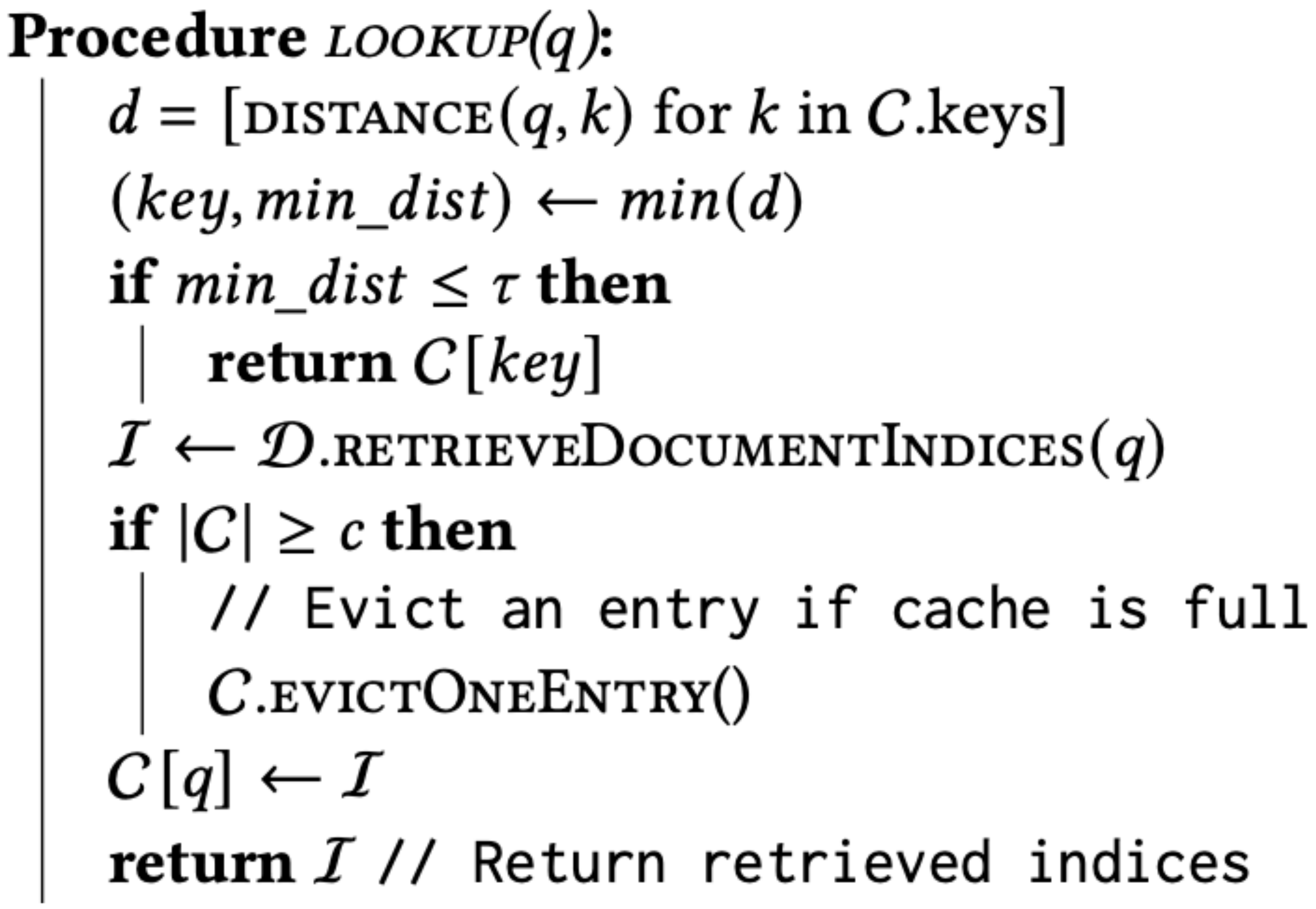


Proposition: Add an approximate cache



Caching algorithm

…in more detail

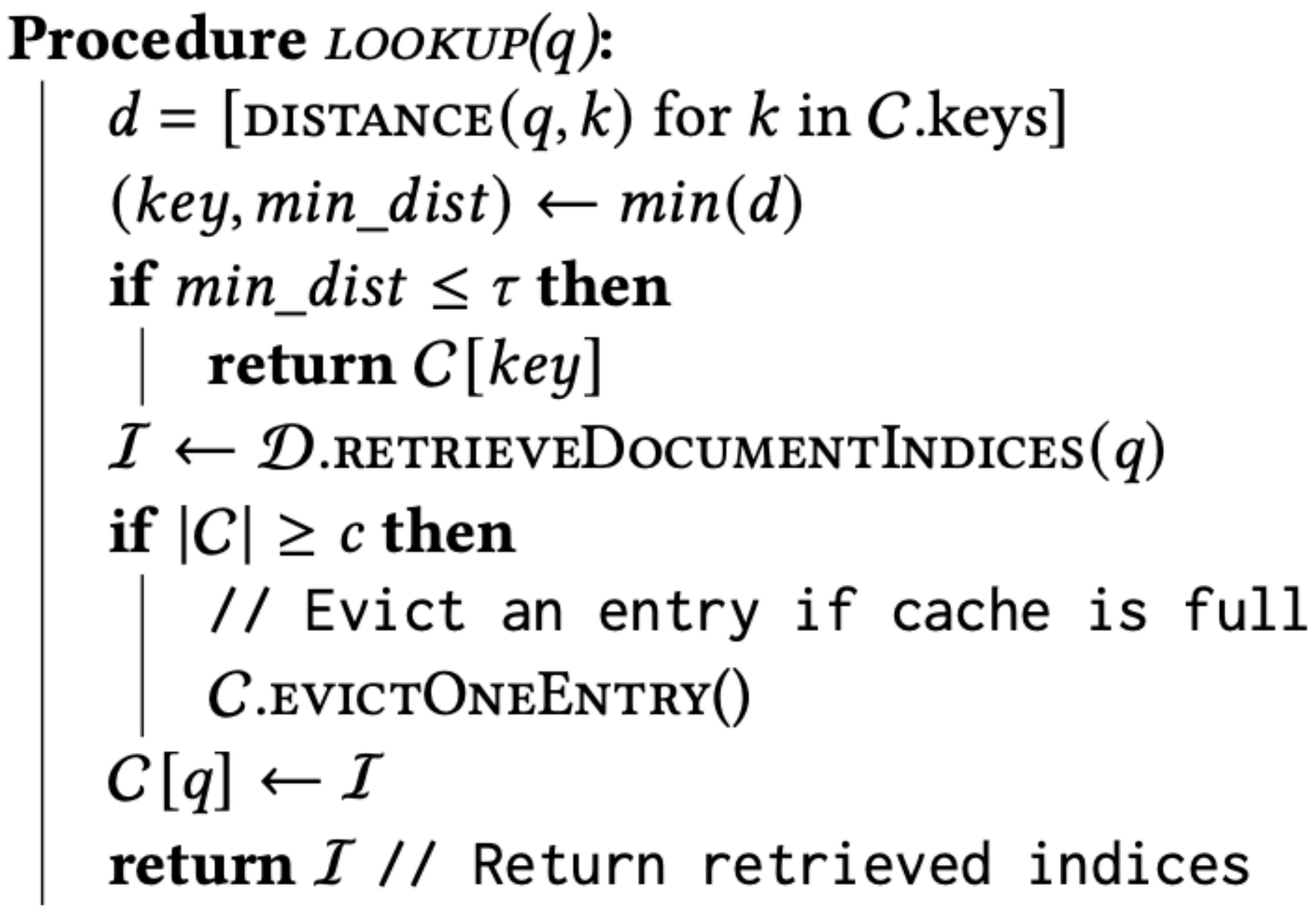


```
Procedure LOOKUP($q$):
    $d = [\text{DISTANCE}(q, k) \text{ for } k \text{ in } C\text{.keys}]$
    $(key, min\_dist) \leftarrow min(d)$
    if $min\_dist \leq \tau$ then
        return $C[key]$
    $\mathcal{I} \leftarrow \mathcal{D}$.RETRIEVEDOCUMENTINDICES($q$)
    if $|C| \geq c$ then
        // Evict an entry if cache is full
        $C$.EVICTONEENTRY()
    $C[q] \leftarrow \mathcal{I}$
    return $\mathcal{I}$ // Return retrieved indices
```

https://www.github.com/sacs-epfl/proximity

Caching algorithm

…in more detail

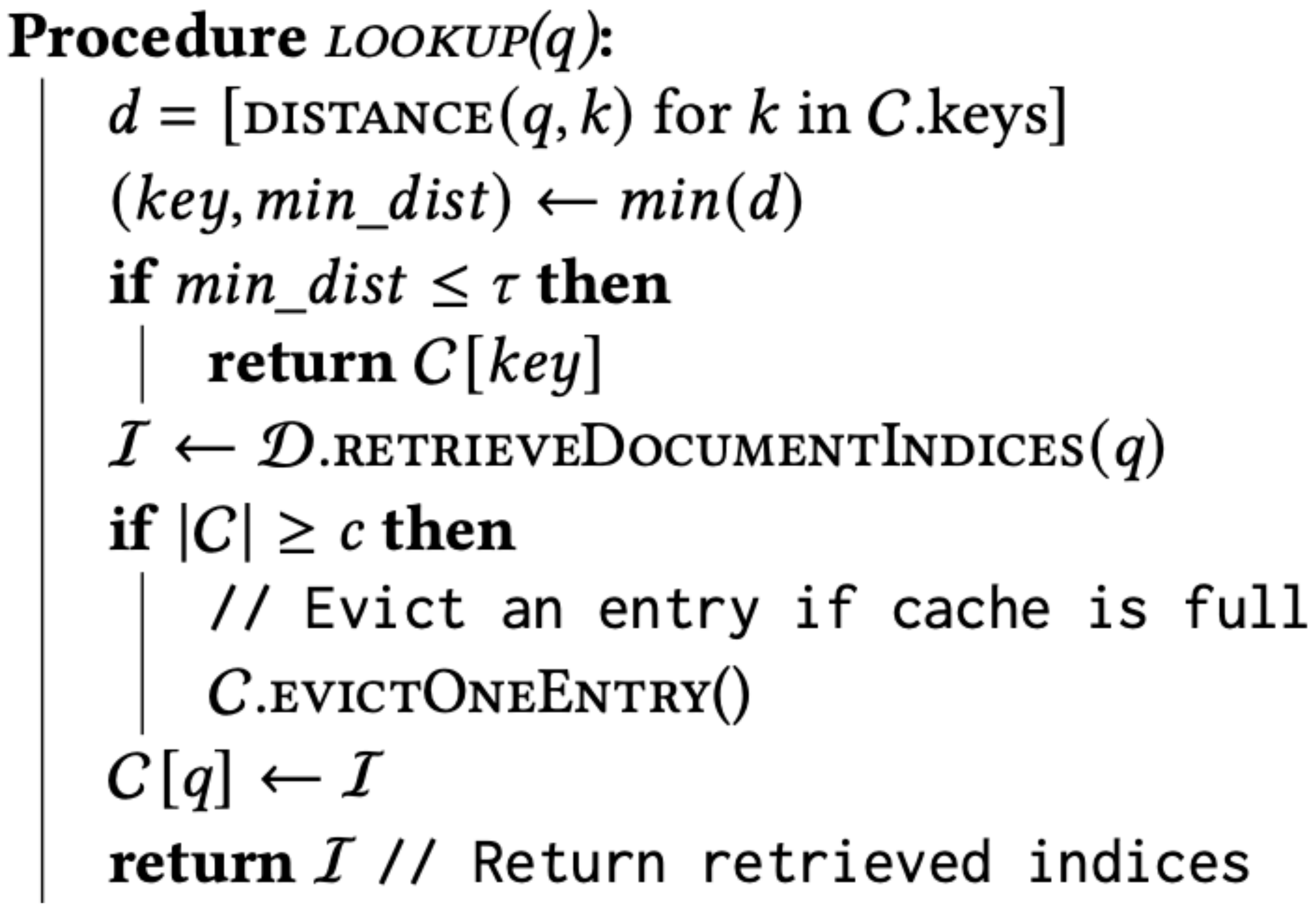


```
Procedure LOOKUP($q$):
    $d$ = [DISTANCE($q, k$) for $k$ in $C$.keys]
    $(key, min\_dist) \leftarrow min(d)$
    if $min\_dist \leq \tau$ then
        return $C[key]$
    $\mathcal{I} \leftarrow \mathcal{D}$.RETRIEVEDOCUMENTINDICES($q$)
    if $|C| \geq c$ then
        // Evict an entry if cache is full
        $C$.EVICTONEENTRY()
    $C[q] \leftarrow \mathcal{I}$
    return $\mathcal{I}$ // Return retrieved indices
```

https://www.github.com/sacs-epfl/proximity

Caching algorithm

…in more detail

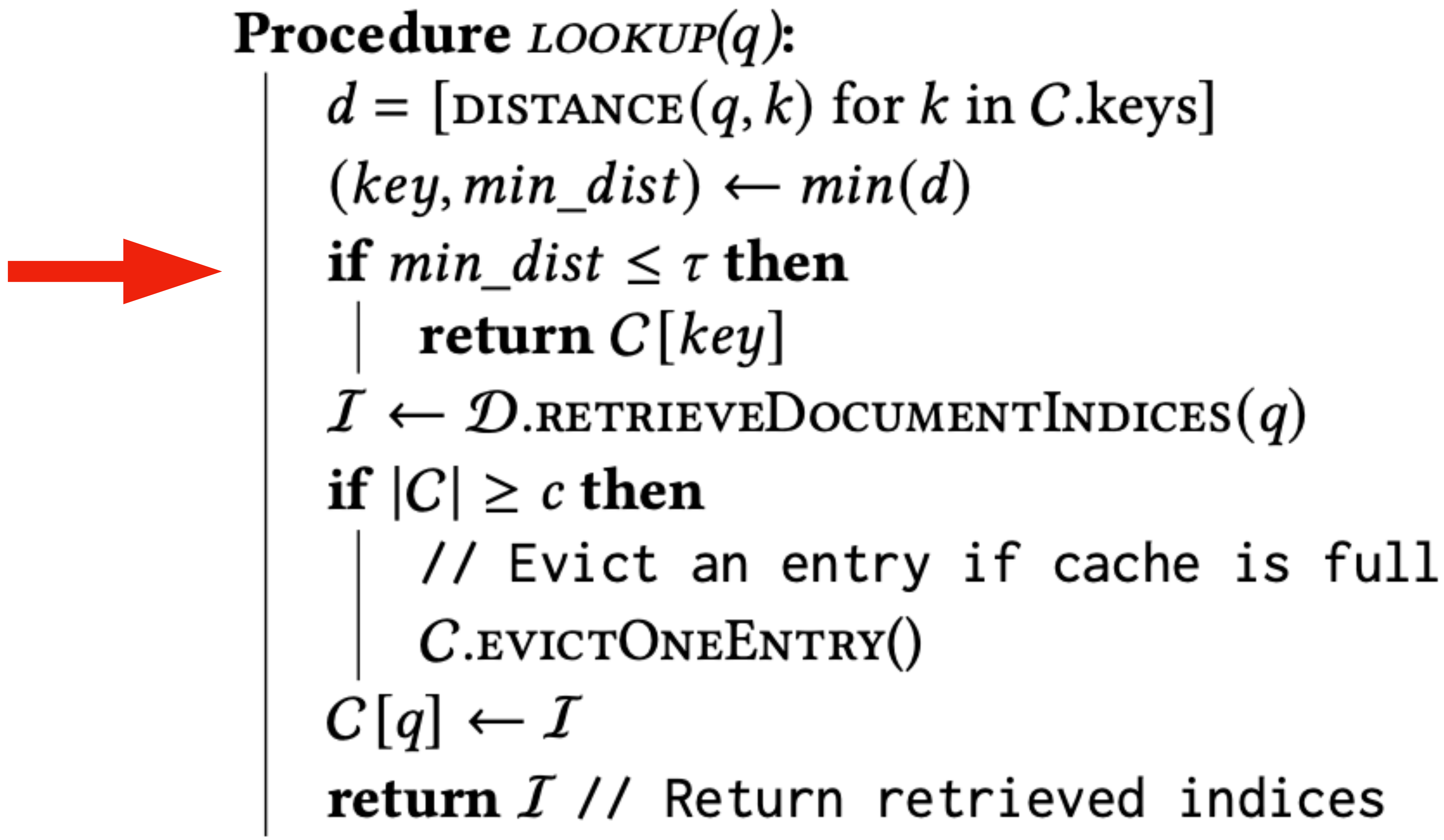


```
Procedure LOOKUP($q$):
    $d$ = [DISTANCE$(q, k)$ for $k$ in $C$.keys]
    $(key, min\_dist) \leftarrow min(d)$
    if $min\_dist \leq \tau$ then
        return $C[key]$
    $\mathcal{I} \leftarrow \mathcal{D}$.RETRIEVEDOCUMENTINDICES$(q)$
    if $|C| \geq c$ then
        // Evict an entry if cache is full
        $C$.EVICTONEENTRY()
    $C[q] \leftarrow \mathcal{I}$
    return $\mathcal{I}$ // Return retrieved indices
```

https://www.github.com/sacs-epfl/proximity

Caching algorithm

…in more detail

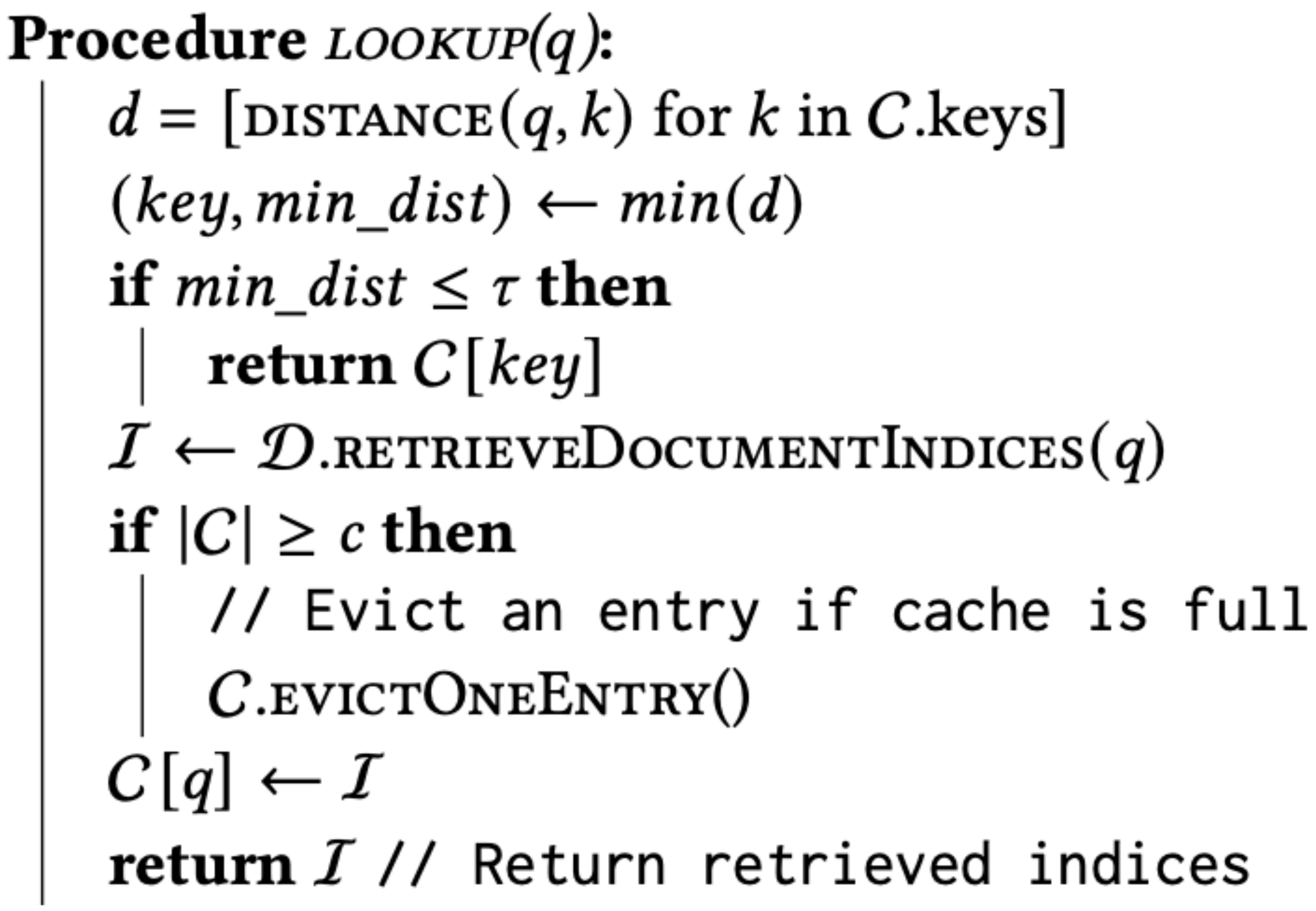


```
Procedure LOOKUP($q$):
    $d$ = [DISTANCE($q, k$) for $k$ in $C$.keys]
    ($key, min\_dist$) ← $min(d)$
    if $min\_dist \leq \tau$ then
        return $C[key]$
    $\mathcal{I} \leftarrow \mathcal{D}$.RETRIEVEDOCUMENTINDICES($q$)
    if $|C| \geq c$ then
        // Evict an entry if cache is full
        $C$.EVICTONEENTRY()
    $C[q] \leftarrow \mathcal{I}$
    return $\mathcal{I}$ // Return retrieved indices
```

https://www.github.com/sacs-epfl/proximity

Caching algorithm

…in more detail

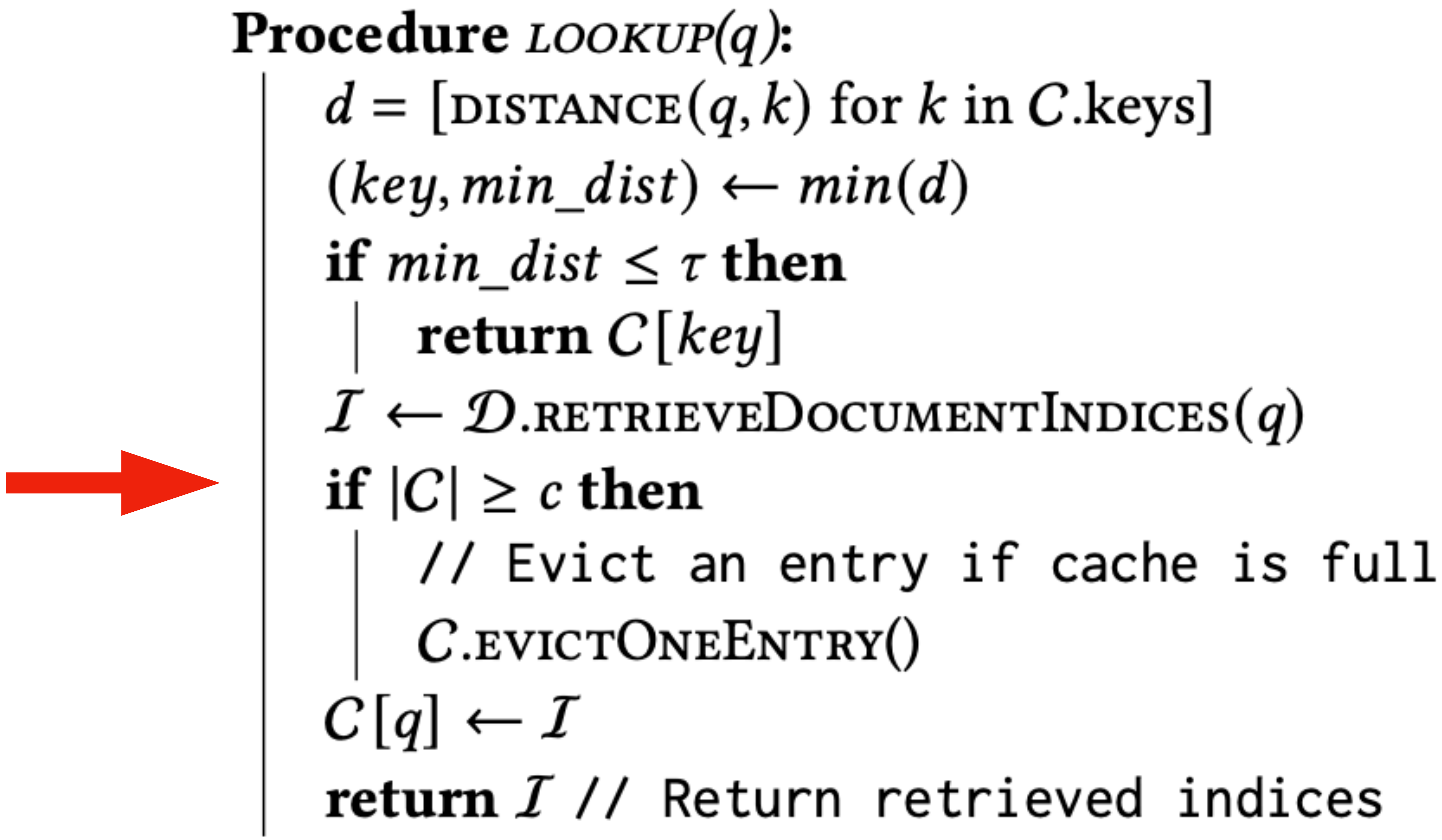


```
Procedure LOOKUP($q$):
    $d$ = [DISTANCE($q, k$) for $k$ in $C$.keys]
    $(key, min\_dist) \leftarrow min(d)$
    if $min\_dist \leq \tau$ then
        return $C[key]$
    $\mathcal{I} \leftarrow \mathcal{D}$.RETRIEVEDOCUMENTINDICES($q$)
    if $|C| \geq c$ then
        // Evict an entry if cache is full
        $C$.EVICTONEENTRY()
    $C[q] \leftarrow \mathcal{I}$
    return $\mathcal{I}$ // Return retrieved indices
```

https://www.github.com/sacs-epfl/proximity

Caching algorithm

…in more detail

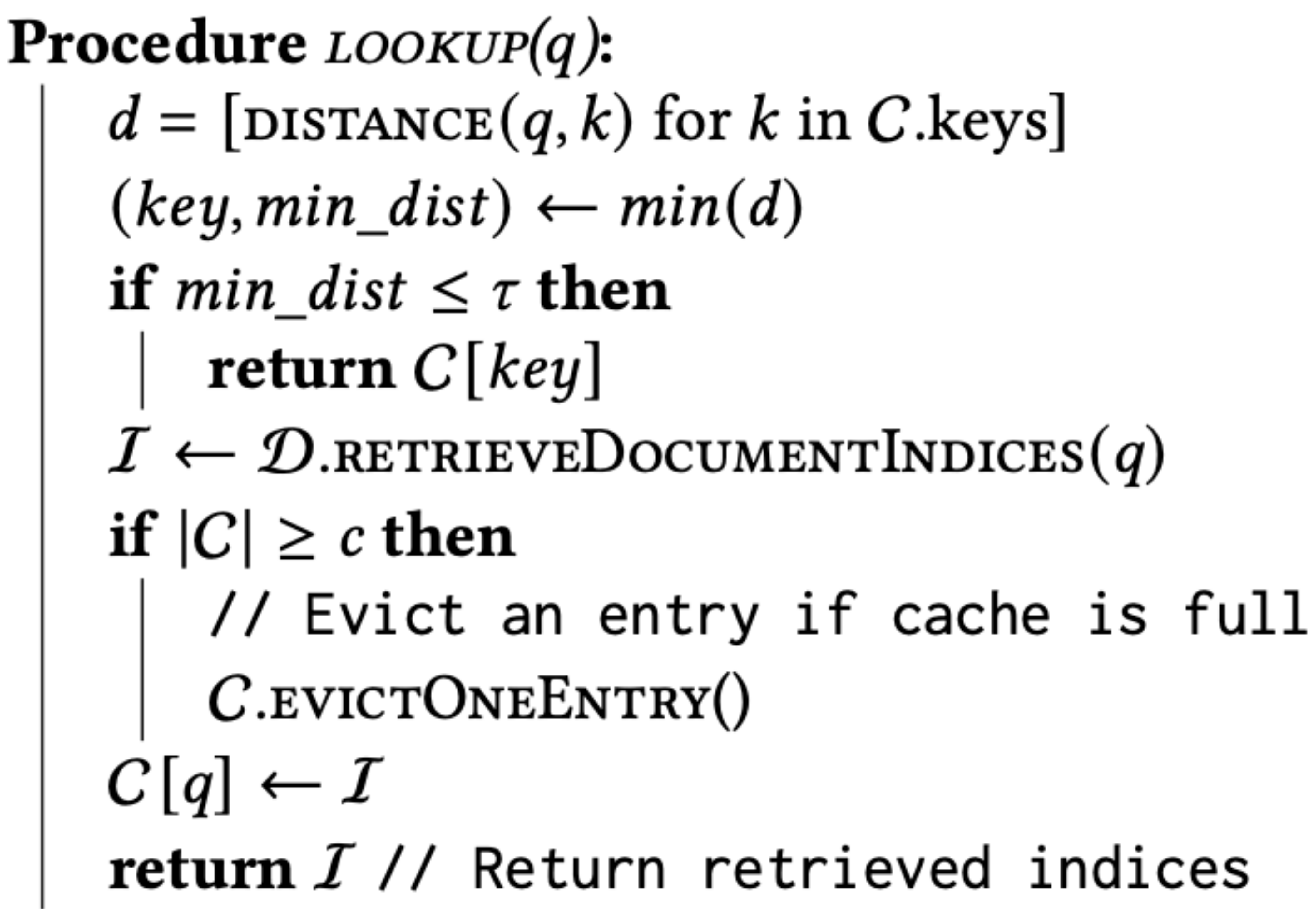


```
Procedure LOOKUP($q$):
    $d$ = [DISTANCE($q, k$) for $k$ in $C$.keys]
    $(key, min\_dist) \leftarrow min(d)$
    if $min\_dist \leq \tau$ then
        return $C[key]$
    $\mathcal{I} \leftarrow \mathcal{D}$.RETRIEVEDOCUMENTINDICES($q$)
    if $|C| \geq c$ then
        // Evict an entry if cache is full
        $C$.EVICTONEENTRY()
    $C[q] \leftarrow \mathcal{I}$
    return $\mathcal{I}$ // Return retrieved indices
```

https://www.github.com/sacs-epfl/proximity

Caching algorithm

…in more detail

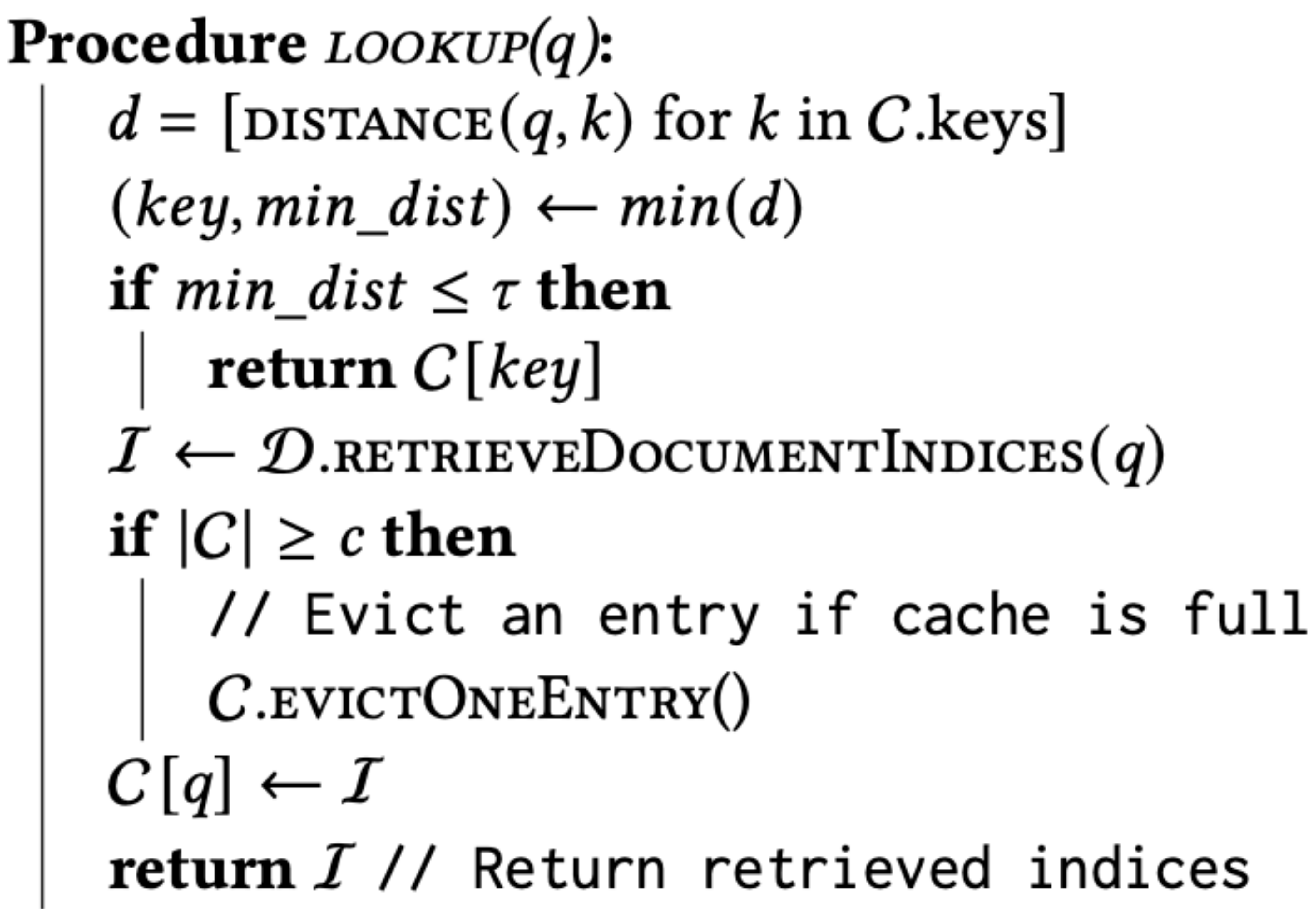


```
Procedure LOOKUP($q$):
    $d$ = [DISTANCE($q, k$) for $k$ in $C$.keys]
    $(key, min\_dist) \leftarrow min(d)$
    if $min\_dist \leq \tau$ then
        return $C[key]$
    $\mathcal{I} \leftarrow \mathcal{D}$.RETRIEVEDOCUMENTINDICES($q$)
    if $|C| \geq c$ then
        // Evict an entry if cache is full
        $C$.EVICTONEENTRY()
    $C[q] \leftarrow \mathcal{I}$
    return $\mathcal{I}$ // Return retrieved indices
```

https://www.github.com/sacs-epfl/proximity

Caching algorithm

…in more detail



```
Procedure LOOKUP($q$):
    $d$ = [DISTANCE($q, k$) for $k$ in $C$.keys]
    $(key, min\_dist) \leftarrow min(d)$
    if $min\_dist \leq \tau$ then
        return $C[key]$
    $\mathcal{I} \leftarrow \mathcal{D}$.RETRIEVEDOCUMENTINDICES($q$)
    if $|C| \geq c$ then
        // Evict an entry if cache is full
        $C$.EVICTONEENTRY()
    $C[q] \leftarrow \mathcal{I}$
    return $\mathcal{I}$ // Return retrieved indices
```

We get a few tweakable parameters:

- Cache capacity
- Cache tolerance
- Eviction policy

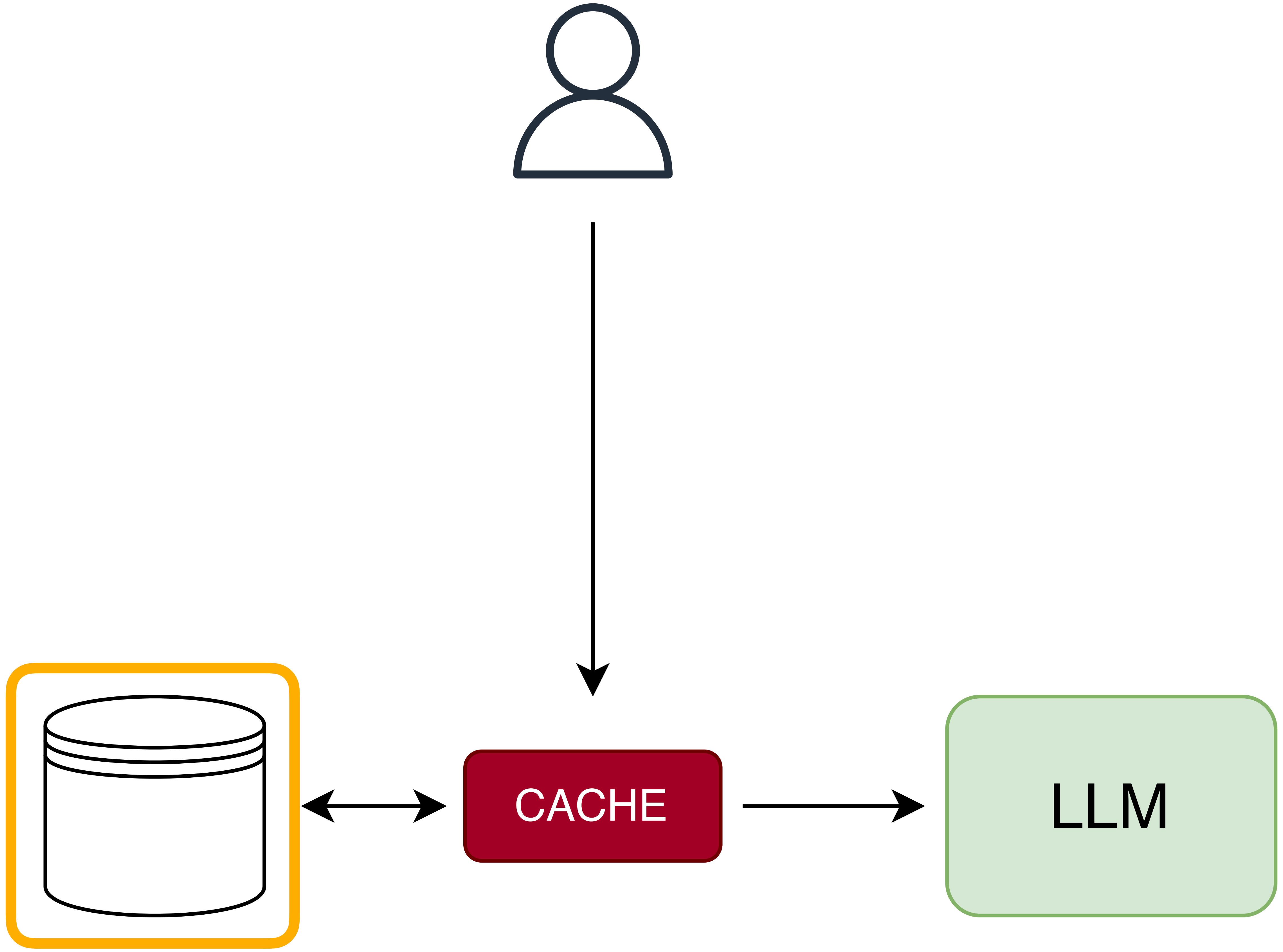
https://www.github.com/sacs-epfl/proximity

Evaluation

- We test on typical RAG dataset: PubMed (23M vectors)

- Database is FAISS-Flat

- LLaMA 3.1 8B Instruct model

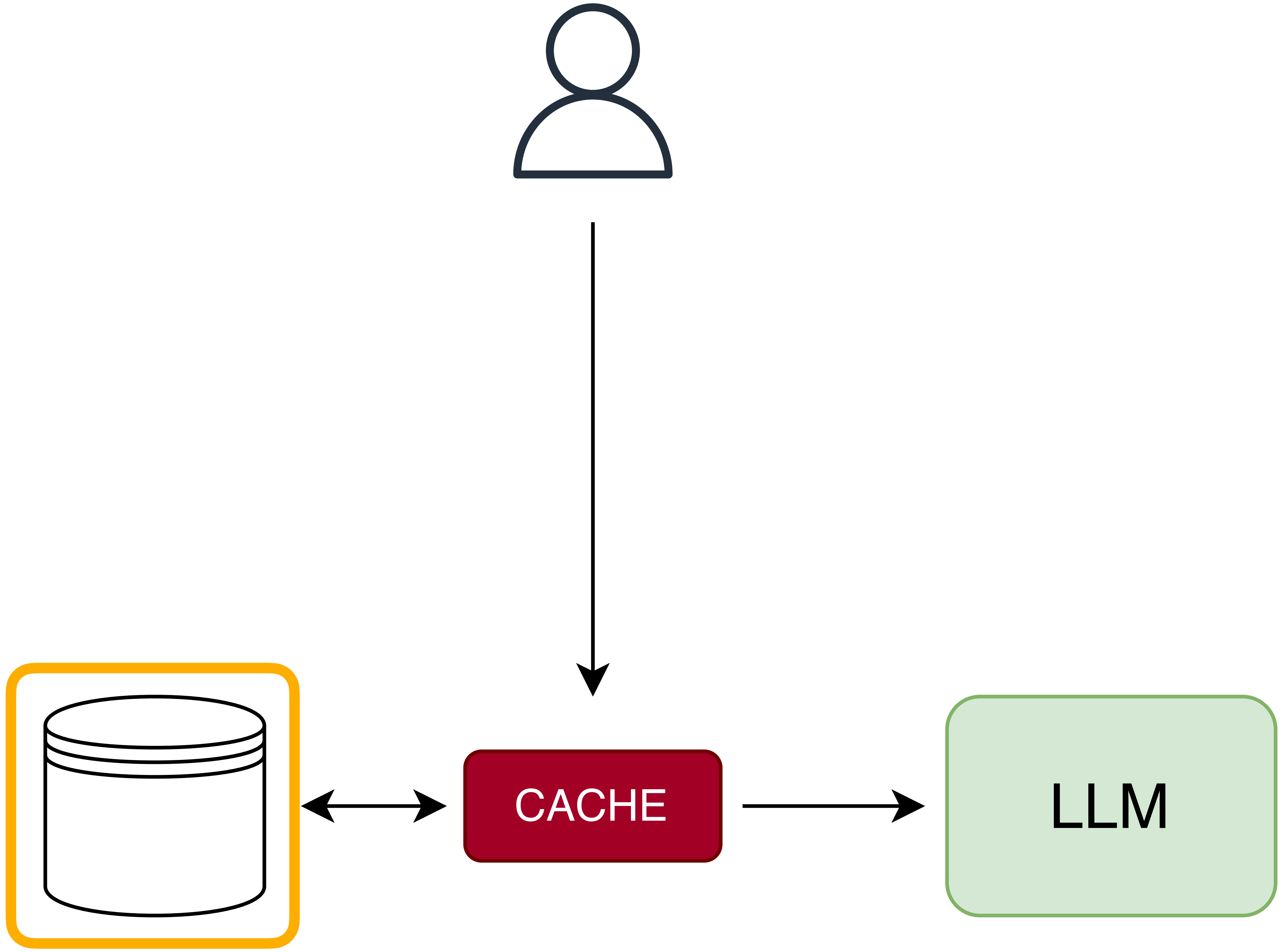


Evaluation

- We test on typical RAG dataset: PubMed (23M vectors)

- Database is FAISS-Flat

- LLaMA 3.1 8B Instruct model

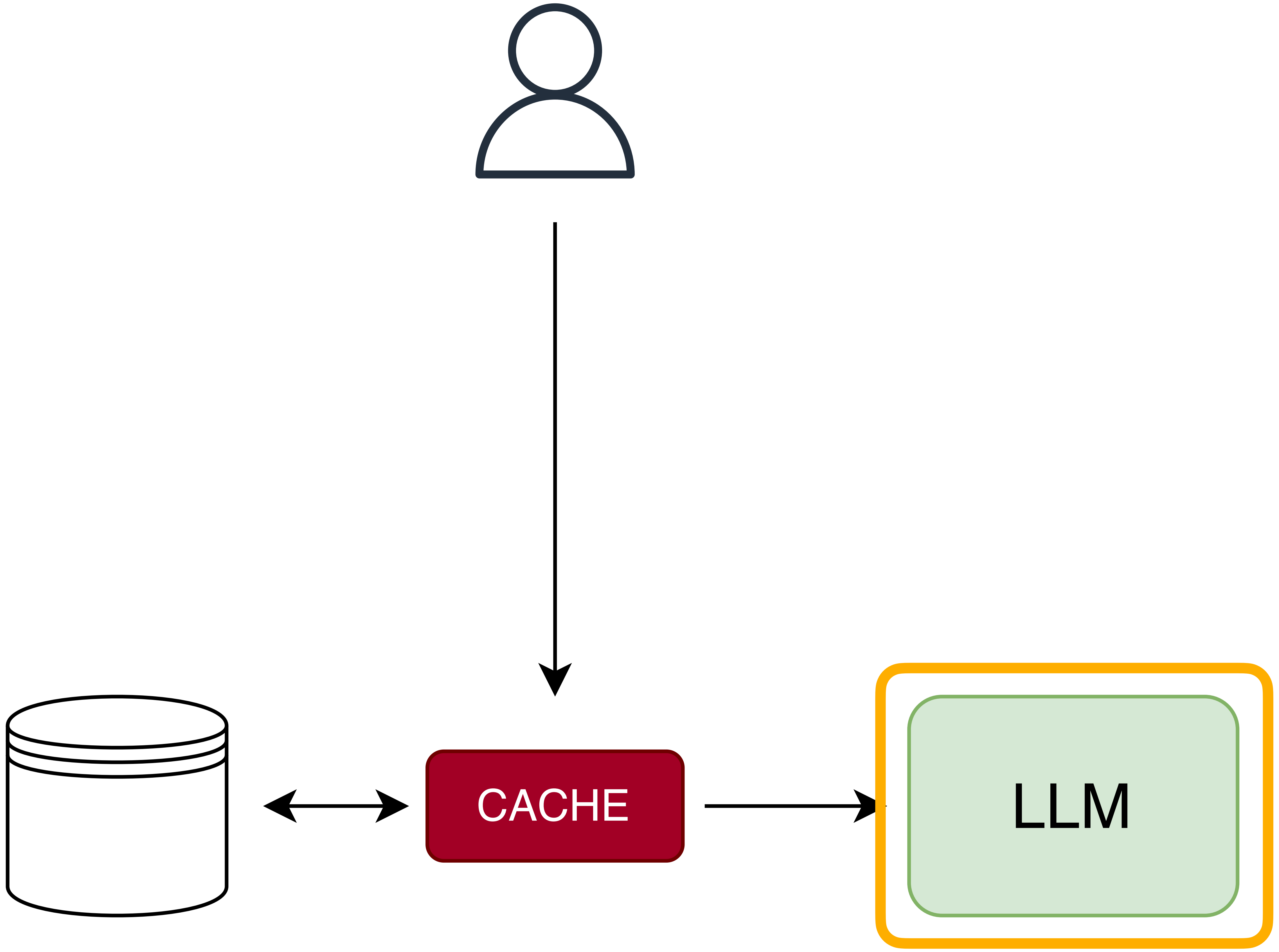


Evaluation

- We test on typical RAG dataset: PubMed (23M vectors)

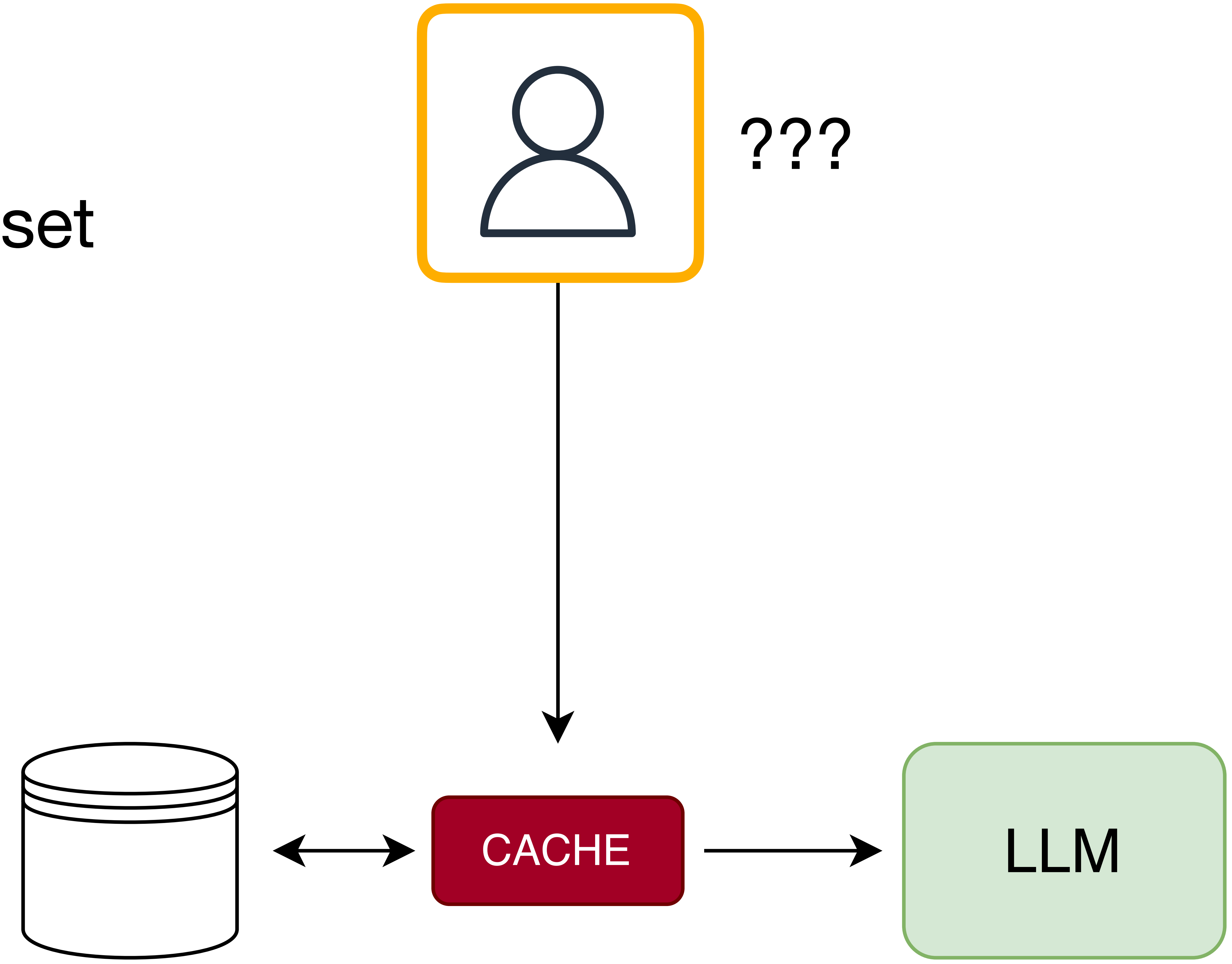
- Database is FAISS-Flat

- LLaMA 3.1 8B Instruct model



Input traces

- Finding traces for execution is difficult, so we generate traces from a real dataset
- PubMedQA (500 medical questions)
- Prepend introduction to the question, randomize order



Evaluation metrics

… comparing against no-cache RAG

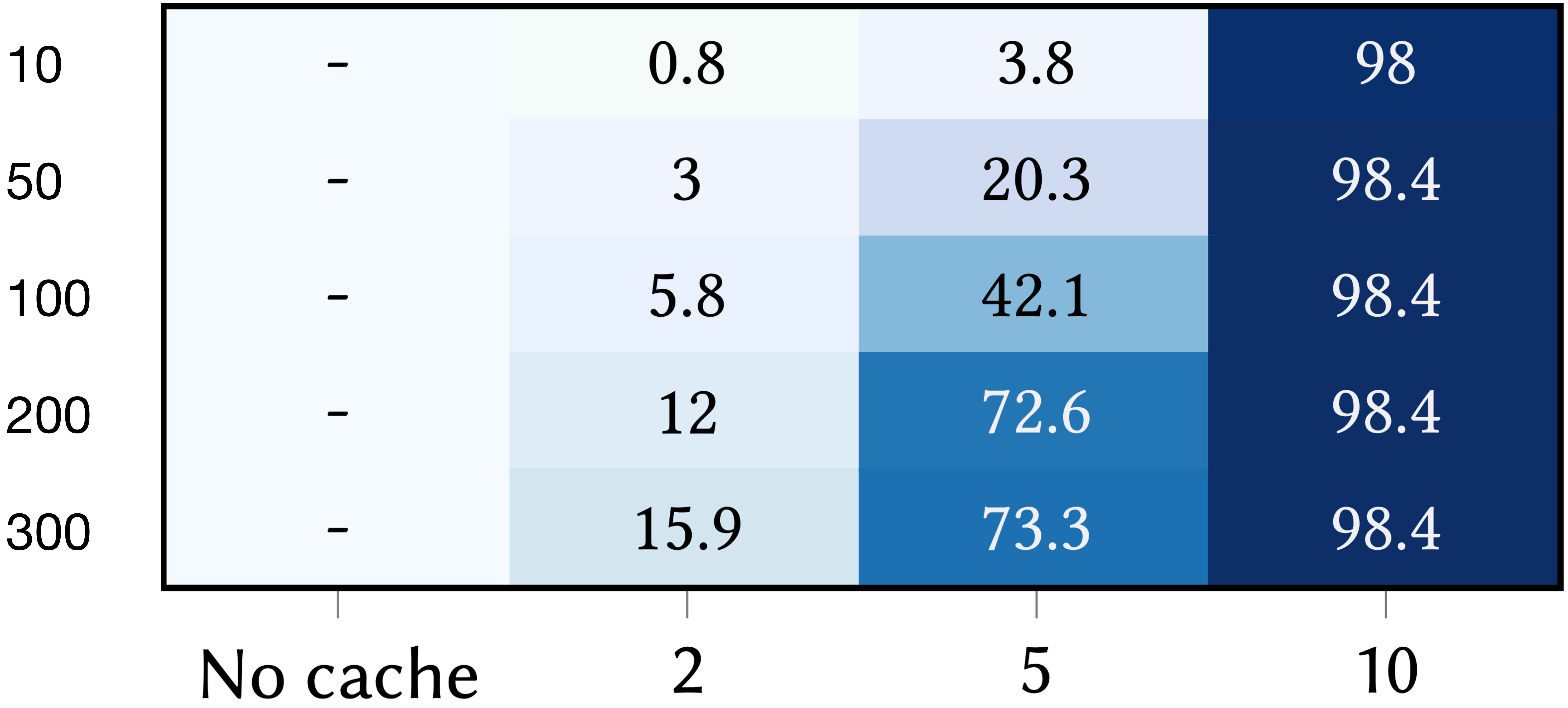
- The cache will influence the **accuracy** of the LLM

- The cache will influence the **retrieval time** of the database

- Retrieval time is a direct function of the **hit rate**

Hit rate (%)

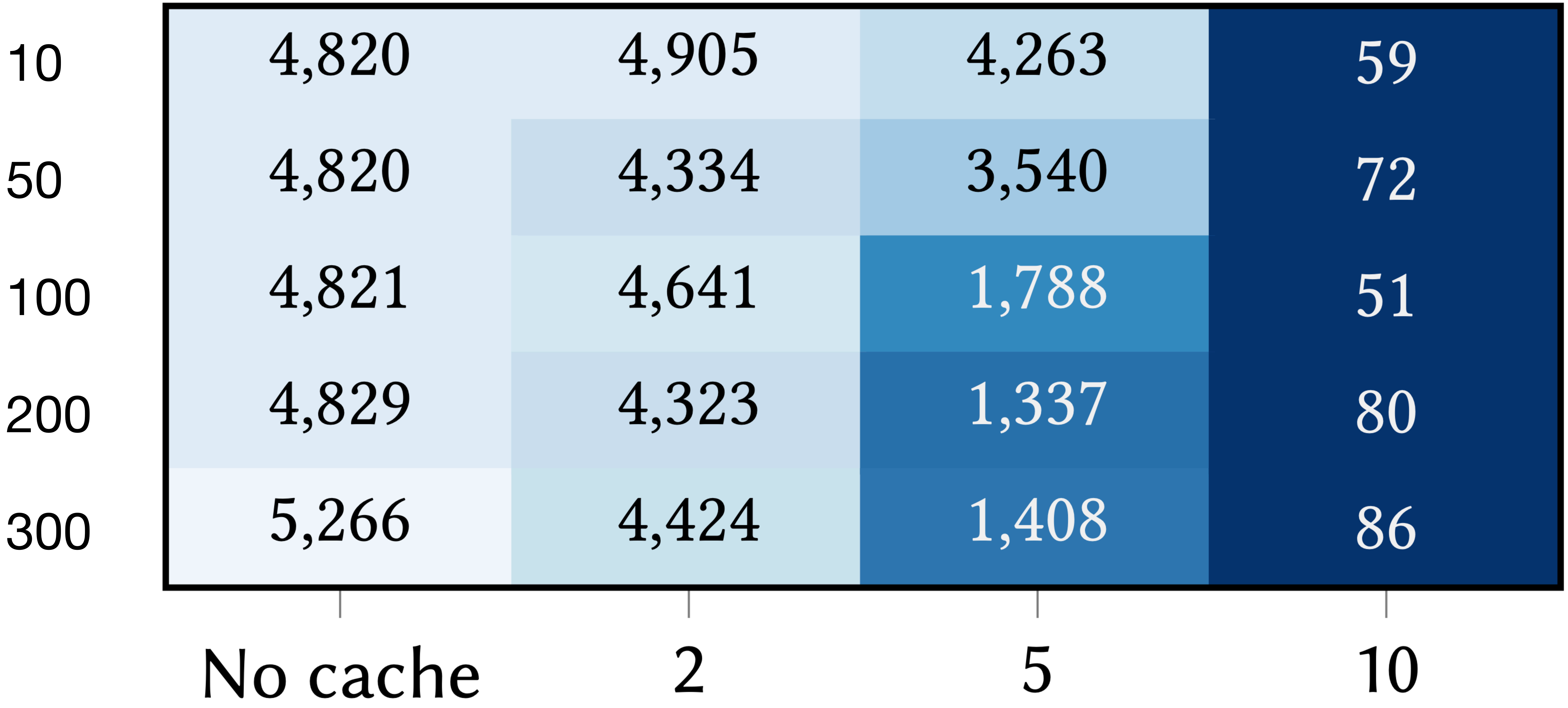
wrt. capacity (y) & tolerance (x) (darker is better)



	No cache	2	5	10
10	-	0.8	3.8	98
50	-	3	20.3	98.4
100	-	5.8	42.1	98.4
200	-	12	72.6	98.4
300	-	15.9	73.3	98.4

Retrieval latency (ms)

wrt. capacity (y) & tolerance (x) (darker is better)

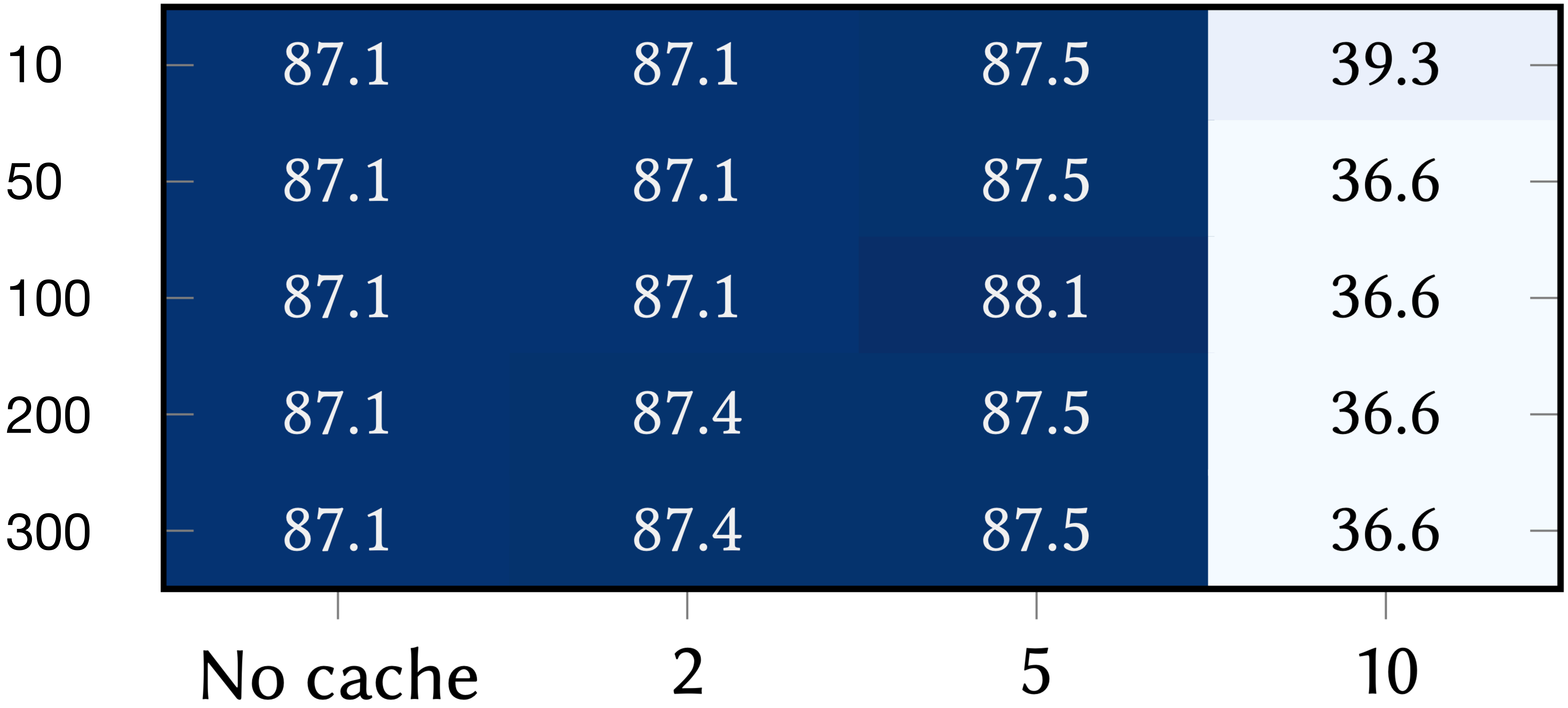


	No cache	2	5	10
10	4,820	4,905	4,263	59
50	4,820	4,334	3,540	72
100	4,821	4,641	1,788	51
200	4,829	4,323	1,337	80
300	5,266	4,424	1,408	86

(this is exact search)

Accuracy (%)

wrt. capacity (y) & tolerance (x) (darker is better)



	No cache	2	5	10
10	87.1	87.1	87.5	39.3
50	87.1	87.1	87.5	36.6
100	87.1	87.1	88.1	36.6
200	87.1	87.4	87.5	36.6
300	87.1	87.4	87.5	36.6

Conclusion

- Approximate caching mitigates the main issue of RAG databases

- Average retrieval latency dramatically improves

- Accuracy does not have to be sacrificed