slides: https://bit.ly/2P0KuW1

Reconfigurable Inverted Index

Yusuke Matsui[1] Ryota Hinami[2] Shin'ichi Satoh[1]

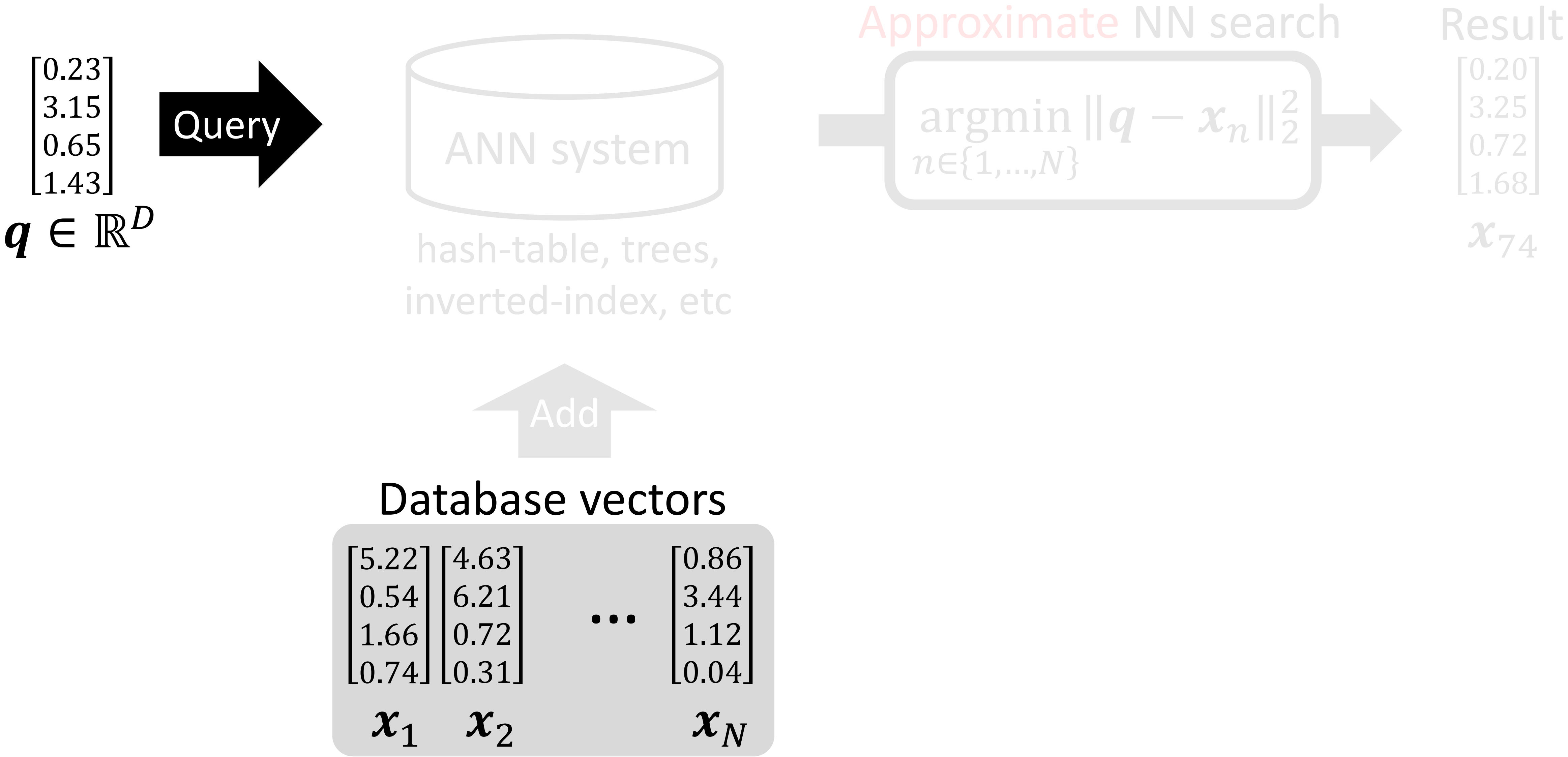
[1]National Institute of Informatics [2]The University of Tokyo

大学共同利用機関法人 情報・システム研究機構
国立情報学研究所
National Institute of Informatics

東京大学
THE UNIVERSITY OF TOKYO

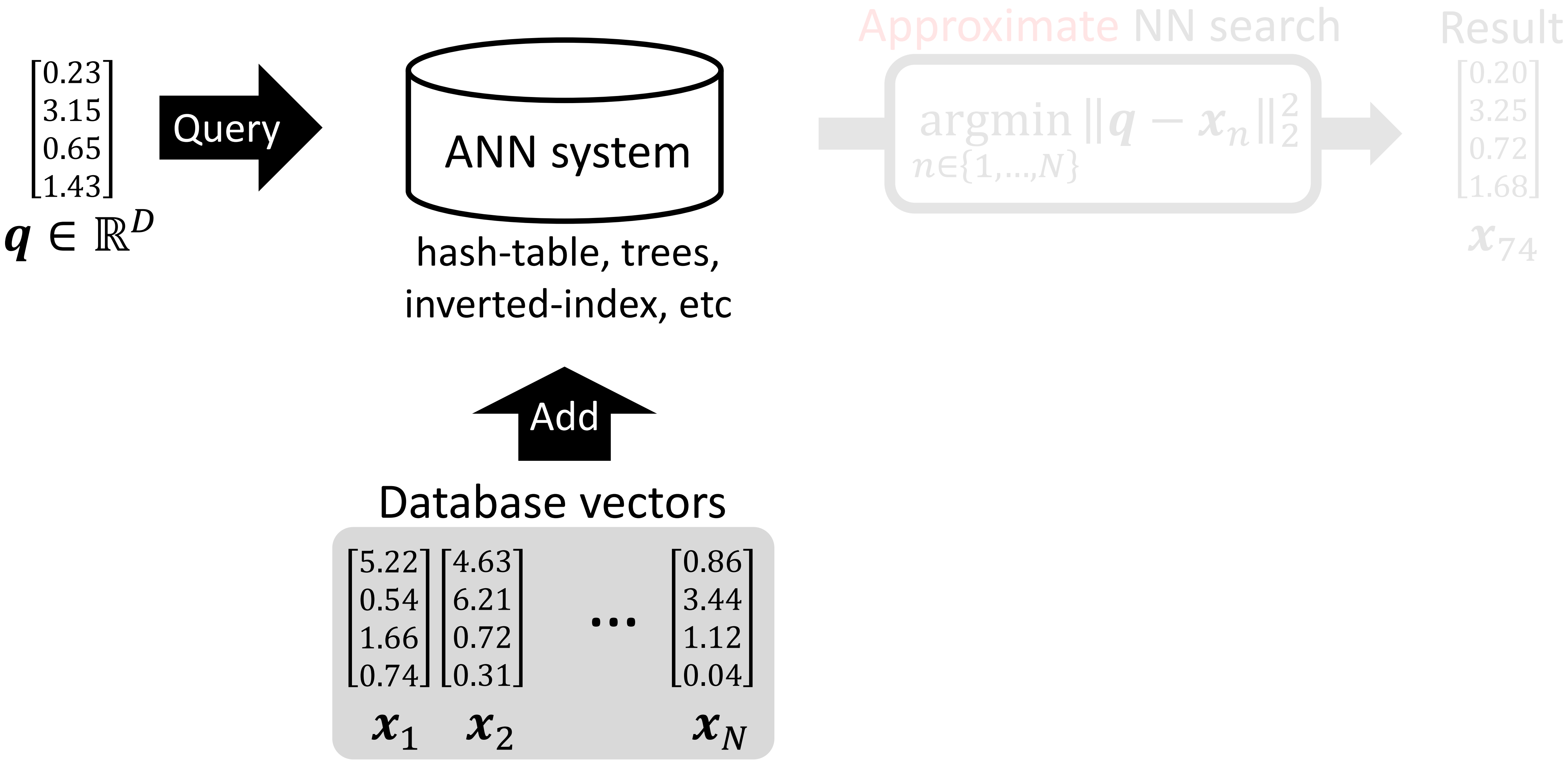
Approximate nearest neighbor search

slides: https://bit.ly/2P0KuW1



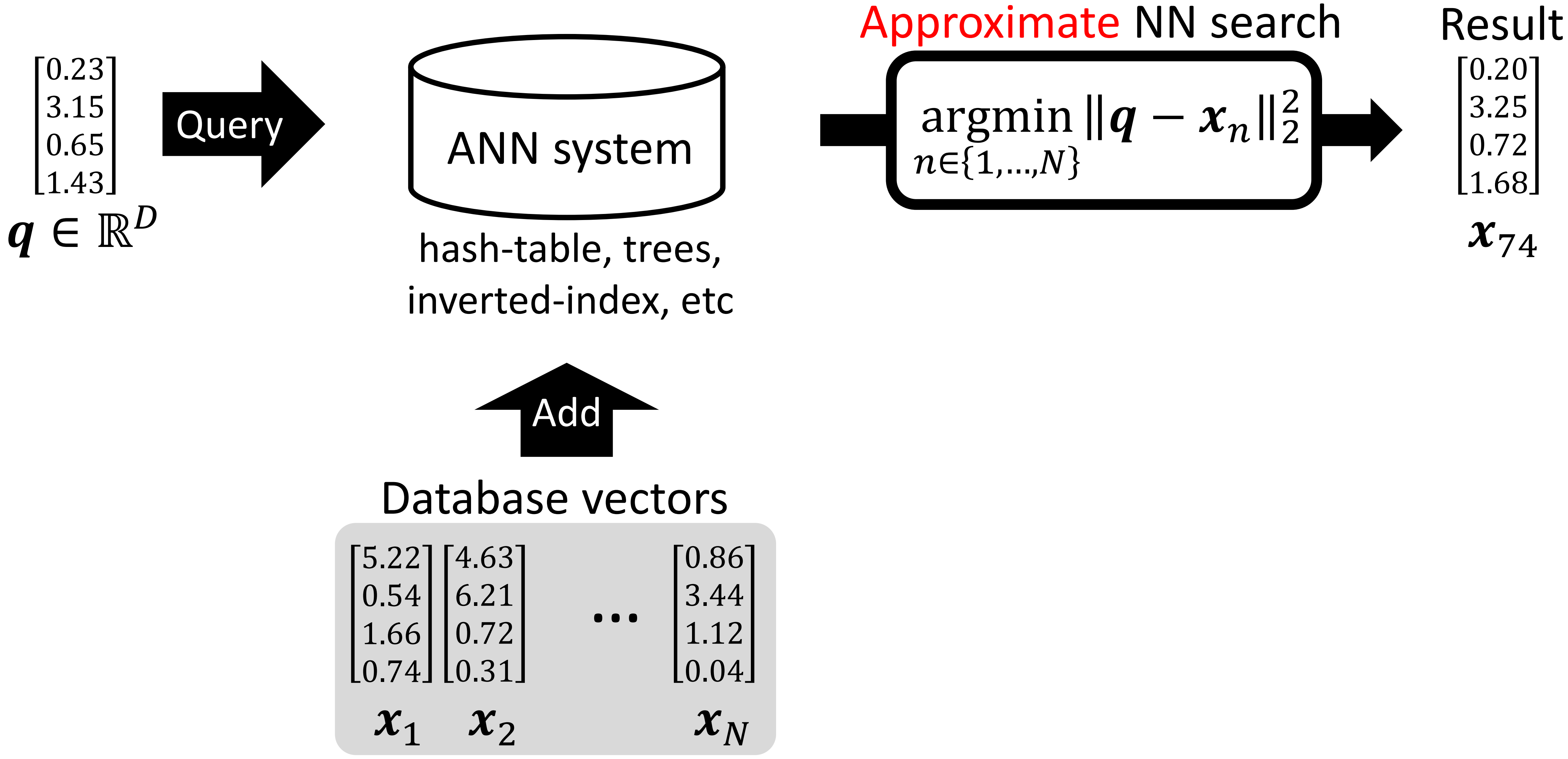
slides: https://bit.ly/2P0KuW1

Approximate nearest neighbor search



slides: https://bit.ly/2P0KuW1

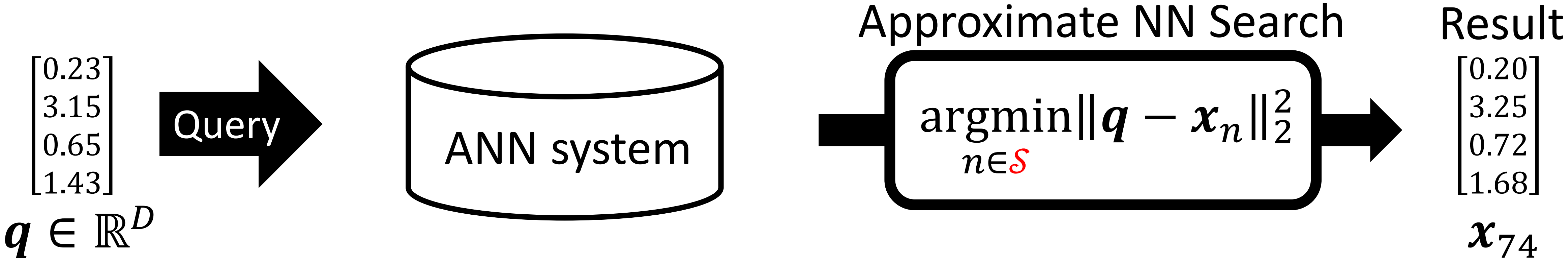
Approximate nearest neighbor search



Related work

- Locality-sensitive-hashing (LSH)
  - FALCONN [Andoni+, 15] [Razenshteyn+, 18]
- Project/tree-based
  - FLANN [Muja+, 14]
  - Annoy [Bernhardsson, 18]
- Graph traversal
  - NSW/HNSW on NMSLIB [Malkov+, 16][Boytsov+, 13]
- Product quantization (PQ)
  - IVFPQ on Faiss [Jégou+, 11][Johnson+, 17] etc.
  - ***Our Reconfigurable Inverted Index***

slides: https://bit.ly/2P0KuW1



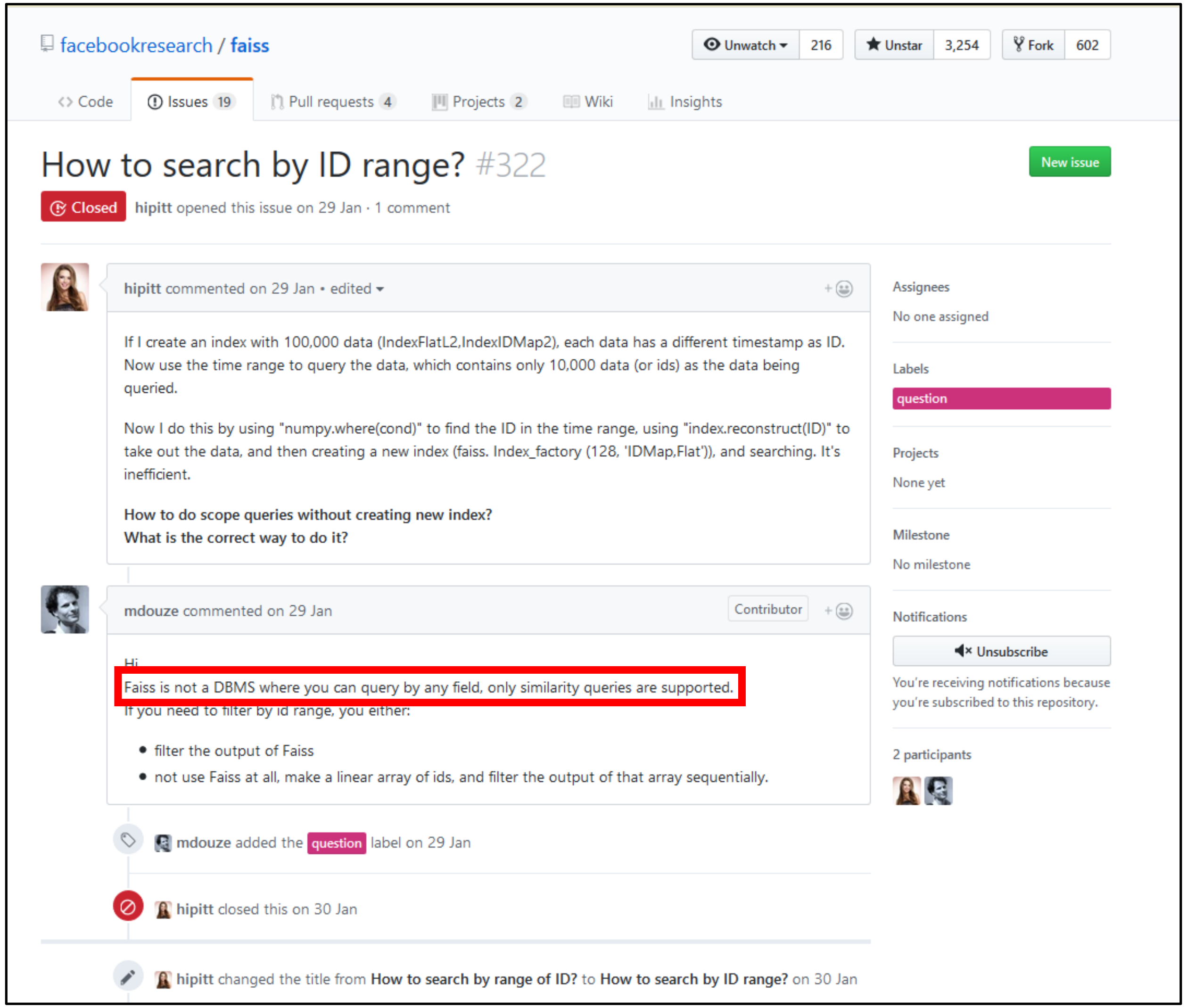
Subset search problem

- Existing ANN systems are fast for the all vectors
  - Search is over $\mathcal{S} = \{1, \dots, N\}$

- However, it is **hard** to run the search for a subset
  - Search is over $\mathcal{S} \subseteq \{1, \dots, N\}$
  - e.g., searching from $\{\boldsymbol{x}_{1000}, \dots, \boldsymbol{x}_{2000}\}$
  - **Why?** Systems are usually optimized for $\mathcal{S} = \{1, \dots, N\}$

slides: https://bit.ly/2P0KuW1

There is a demand for subset search!



facebookresearch / faiss

Unwatch 216 | Unstar 3,254 | Fork 602

Code | Issues 19 | Pull requests 4 | Projects 2 | Wiki | Insights

How to search by ID range? #322

New issue

Closed **hipitt** opened this issue on 29 Jan · 1 comment

hipitt commented on 29 Jan • edited

If I create an index with 100,000 data (IndexFlatL2,IndexIDMap2), each data has a different timestamp as ID. Now use the time range to query the data, which contains only 10,000 data (or ids) as the data being queried.

Now I do this by using "numpy.where(cond)" to find the ID in the time range, using "index.reconstruct(ID)" to take out the data, and then creating a new index (faiss. Index_factory (128, 'IDMap,Flat')), and searching. It's inefficient.

How to do scope queries without creating new index?
What is the correct way to do it?

mdouze commented on 29 Jan — Contributor

Hi,
Faiss is not a DBMS where you can query by any field, only similarity queries are supported.
If you need to filter by id range, you either:

- filter the output of Faiss
- not use Faiss at all, make a linear array of ids, and filter the output of that array sequentially.

mdouze added the question label on 29 Jan

hipitt closed this on 30 Jan

hipitt changed the title from **How to search by range of ID?** to **How to search by ID range?** on 30 Jan

Assignees: No one assigned

Labels: question

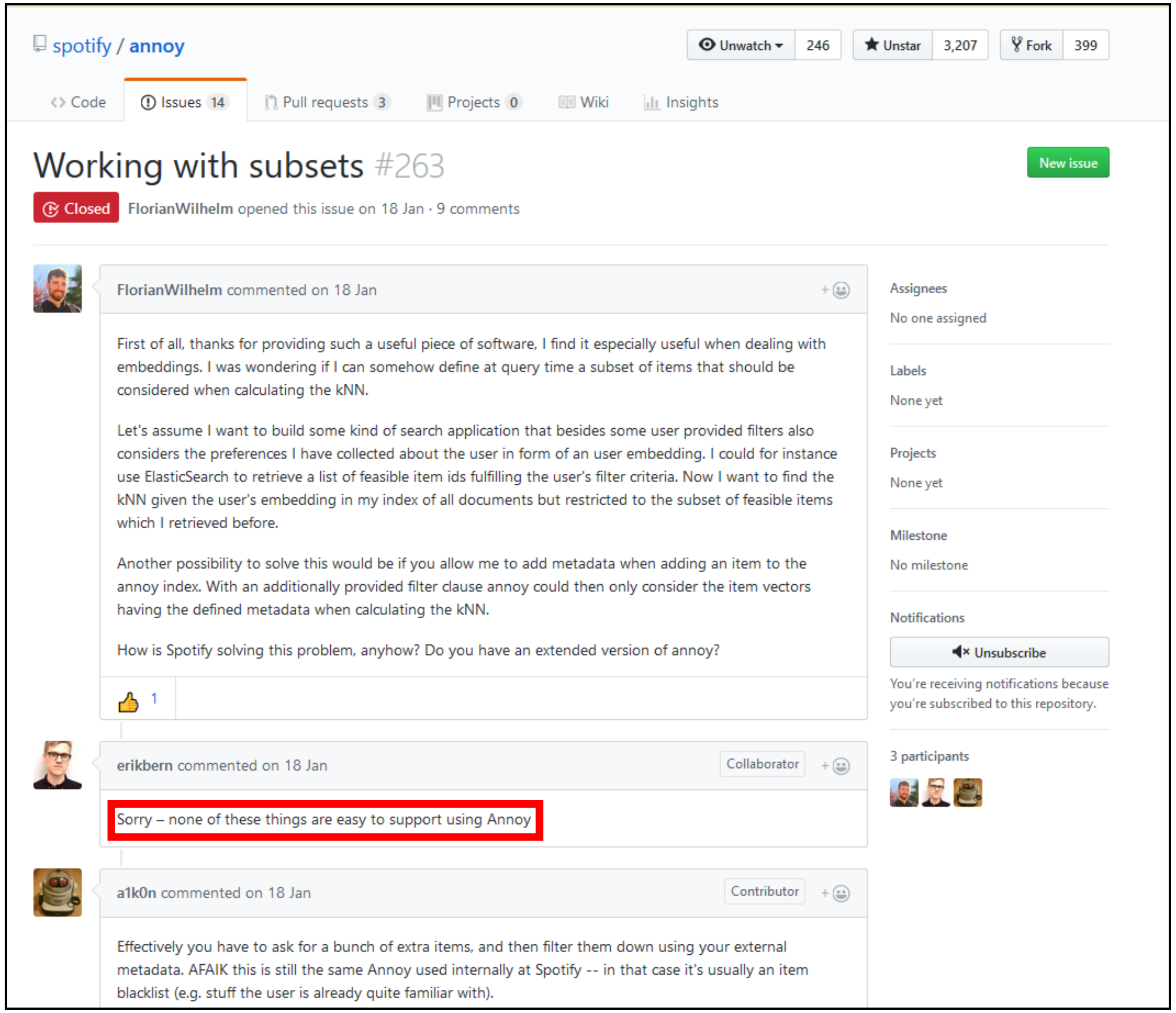
Projects: None yet

Milestone: No milestone

Notifications: Unsubscribe

You're receiving notifications because you're subscribed to this repository.

2 participants



spotify / annoy

Unwatch 246 | Unstar 3,207 | Fork 399

Code | Issues 14 | Pull requests 3 | Projects 0 | Wiki | Insights

Working with subsets #263

New issue

Closed **FlorianWilhelm** opened this issue on 18 Jan · 9 comments

FlorianWilhelm commented on 18 Jan

First of all, thanks for providing such a useful piece of software, I find it especially useful when dealing with embeddings. I was wondering if I can somehow define at query time a subset of items that should be considered when calculating the kNN.

Let's assume I want to build some kind of search application that besides some user provided filters also considers the preferences I have collected about the user in form of an user embedding. I could for instance use ElasticSearch to retrieve a list of feasible item ids fulfilling the user's filter criteria. Now I want to find the kNN given the user's embedding in my index of all documents but restricted to the subset of feasible items which I retrieved before.

Another possibility to solve this would be if you allow me to add metadata when adding an item to the annoy index. With an additionally provided filter clause annoy could then only consider the item vectors having the defined metadata when calculating the kNN.

How is Spotify solving this problem, anyhow? Do you have an extended version of annoy?

👍 1

erikbern commented on 18 Jan — Collaborator

Sorry – none of these things are easy to support using Annoy

a1k0n commented on 18 Jan — Contributor

Effectively you have to ask for a bunch of extra items, and then filter them down using your external metadata. AFAIK this is still the same Annoy used internally at Spotify -- in that case it's usually an item blacklist (e.g. stuff the user is already quite familiar with).

Assignees: No one assigned

Labels: None yet

Projects: None yet

Milestone: No milestone

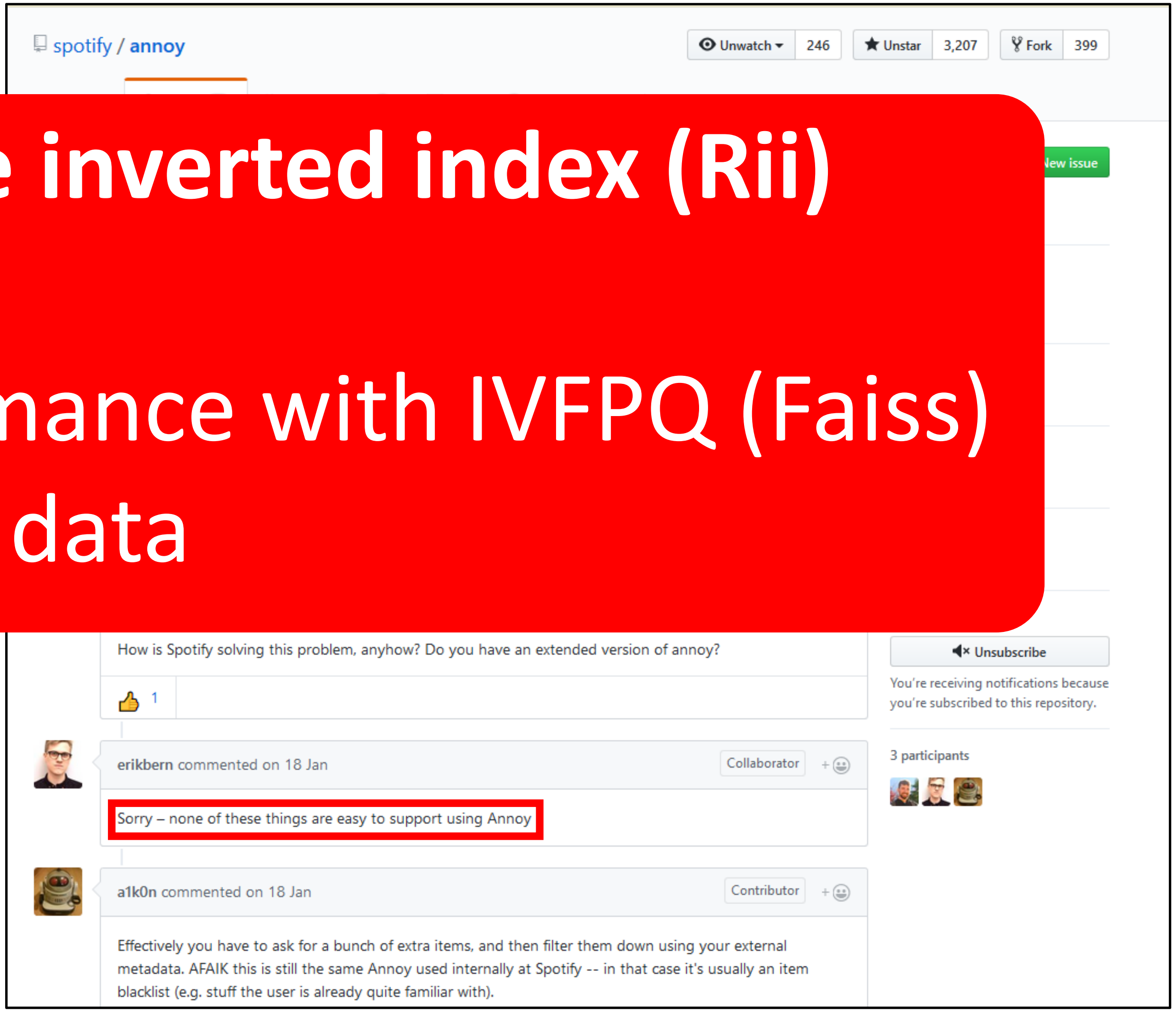
Notifications: Unsubscribe

You're receiving notifications because you're subscribed to this repository.

3 participants

slides: https://bit.ly/2P0KuW1

There is a demand for subset search!



Propose: **Reconfigurable inverted index (Rii)**

✓ Subset search

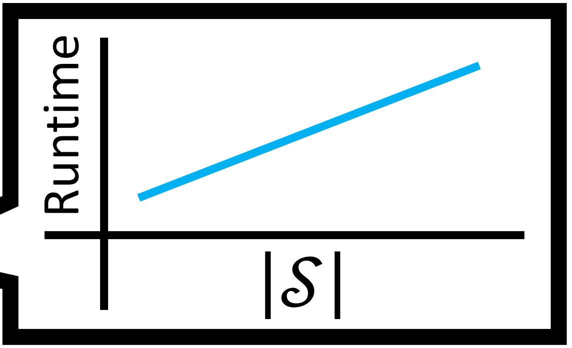
✓ A comparative performance with IVFPQ (Faiss)

✓ 10 ms for billion-scale data

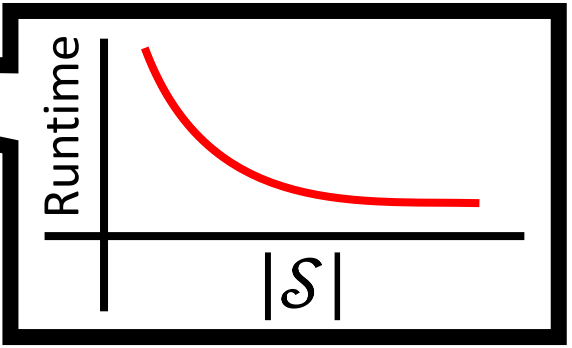
slides: https://bit.ly/2P0KuW1

Reconfigurable inverted index (Rii)

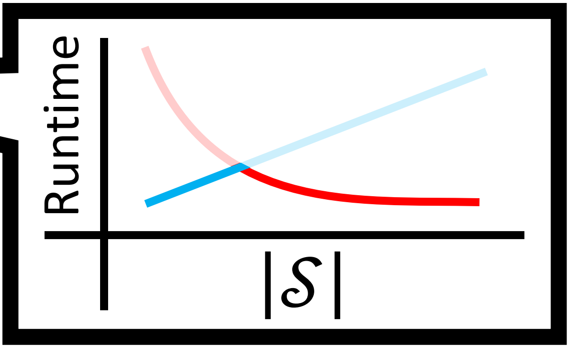
- Preliminary
  - PQ linear scan
  - IVFPQ
- Data structure
- Search



Fast if $|\mathcal{S}|$ is small



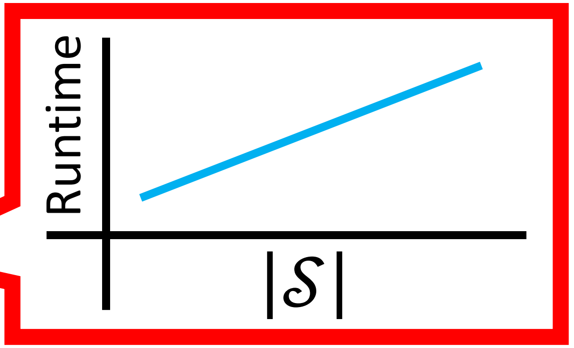
Fast if $|\mathcal{S}|$ is large



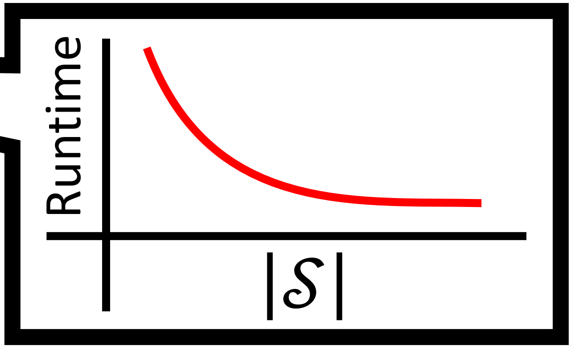
Cherry pick!
Always fast

Reconfigurable inverted index (Rii)

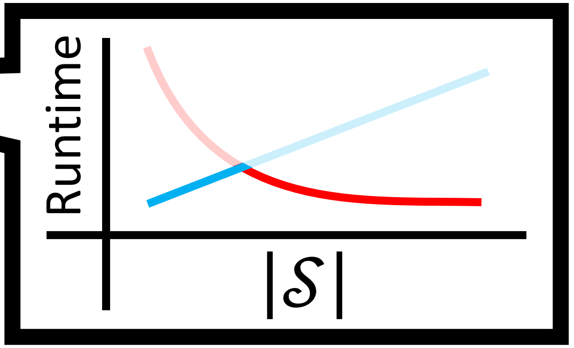
- Preliminary
  - PQ linear scan
  - IVFPQ
- Data structure
- Search



Fast if $|\mathcal{S}|$ is small



Fast if $|\mathcal{S}|$ is large

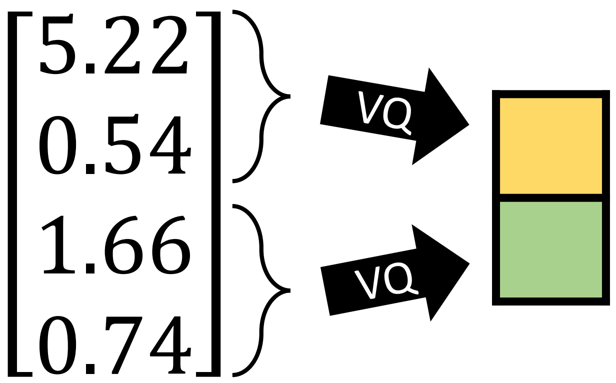


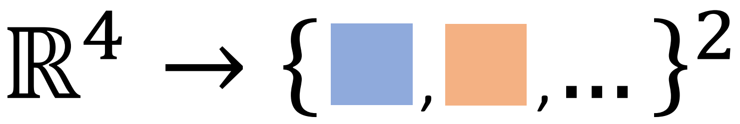
Cherry pick!
Always fast

slides: https://bit.ly/2P0KuW1

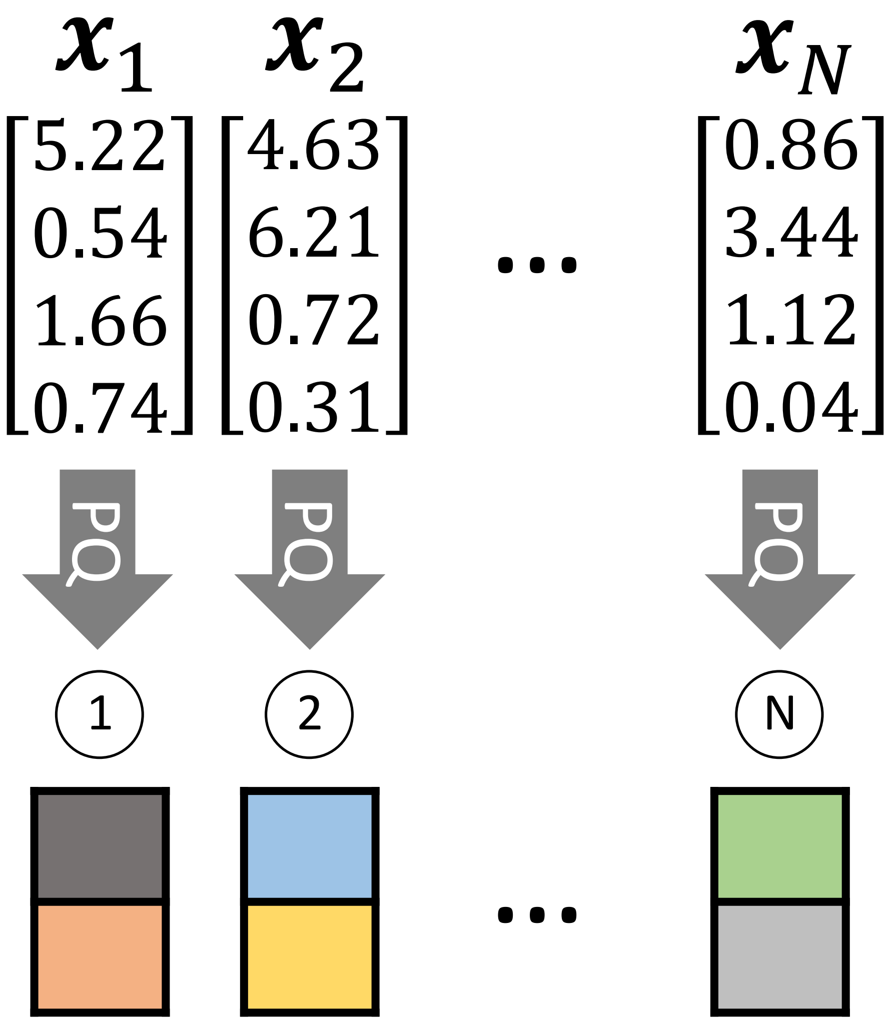
Preliminary: Product quantization (PQ) [Jégou+, TPAMI 11]

PQ: Compress a vector into a short code



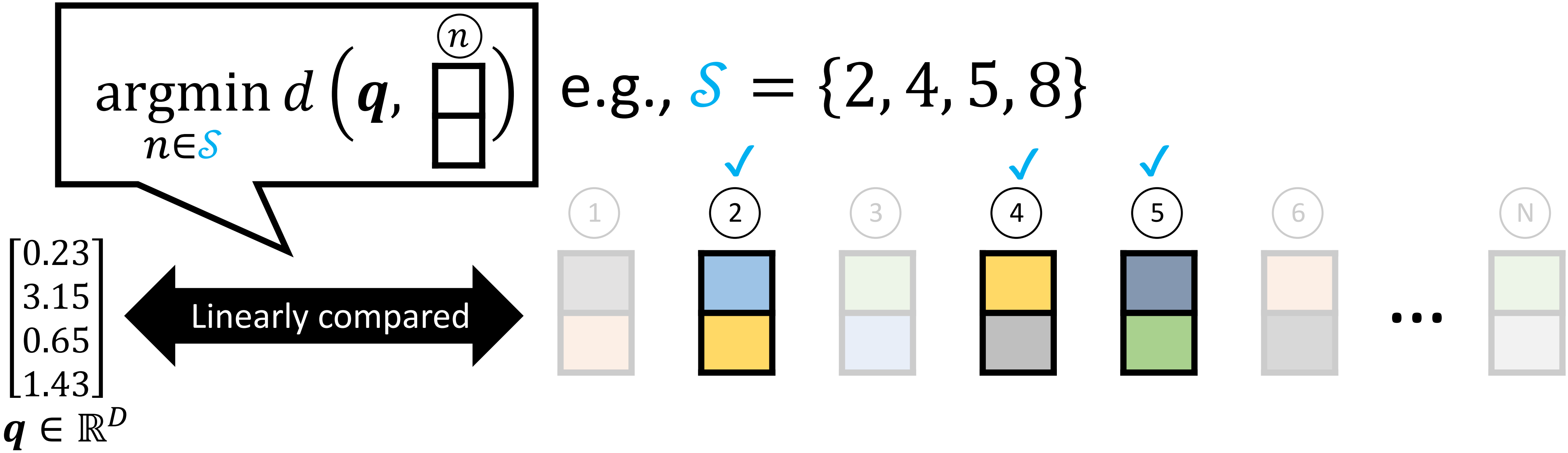


All database vectors are PQ-encoded beforehand

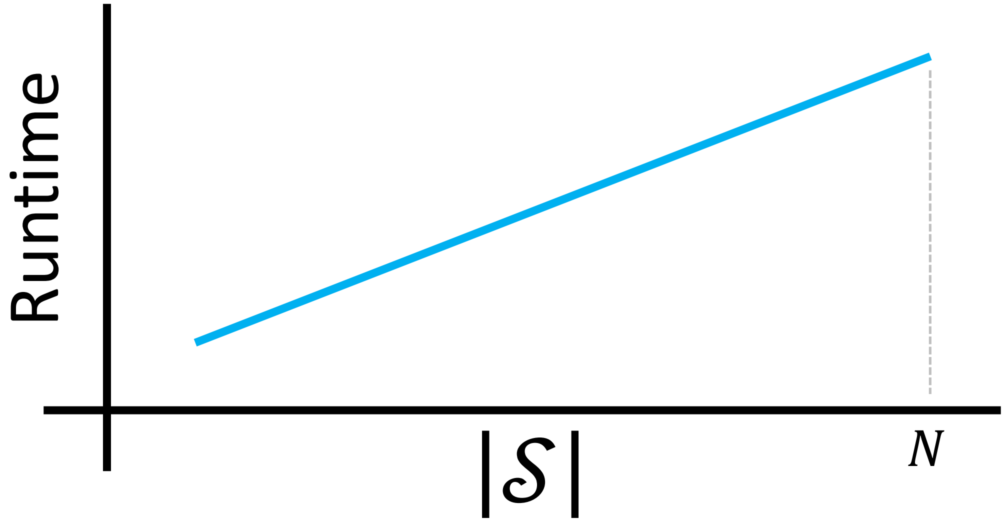


Preliminary: Product quantization (PQ) [Jégou+, TPAMI 11]

- The subset search is possible with a linear cost of $|\mathcal{S}|$

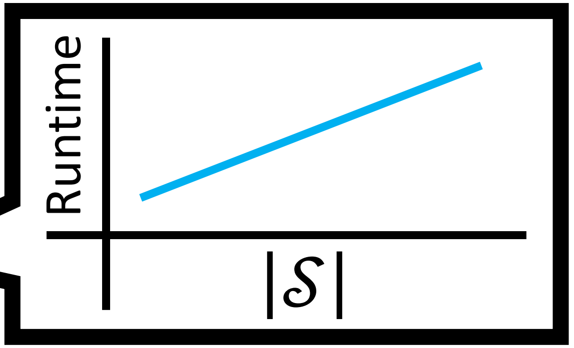


- The search is efficient only if $|\mathcal{S}|$ is small

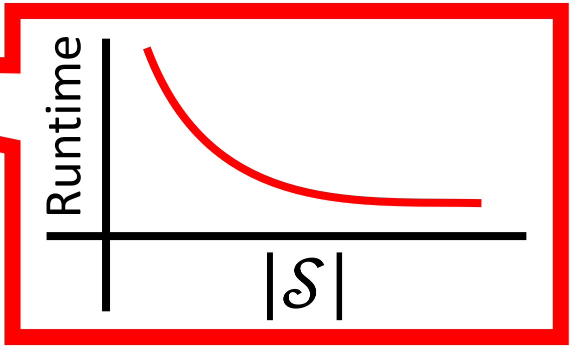


Reconfigurable inverted index (Rii)

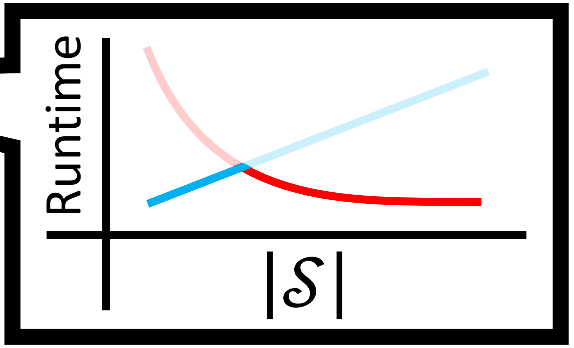
- Preliminary
  - PQ linear scan
  - IVFPQ
- Data structure
- Search
- Evaluation



Fast if $|\mathcal{S}|$ is small



Fast if $|\mathcal{S}|$ is large

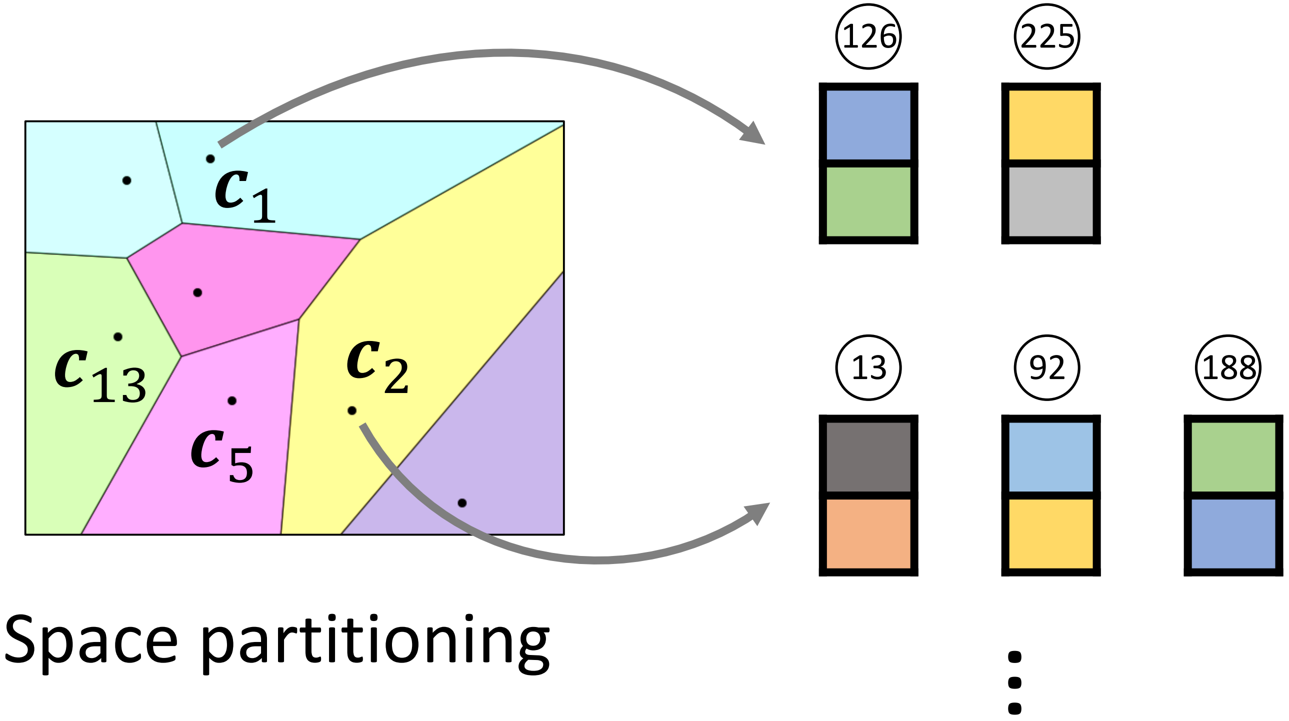


Cherry pick!
Always fast

slides: https://bit.ly/2P0KuW1

Preliminary: Inverted Index + PQ (IVFPQ) [Jégou+, TPAMI 11]

- Current basic data structure for a large-scale search
- Subset-search is possible only if $|\mathcal{S}|$ is large

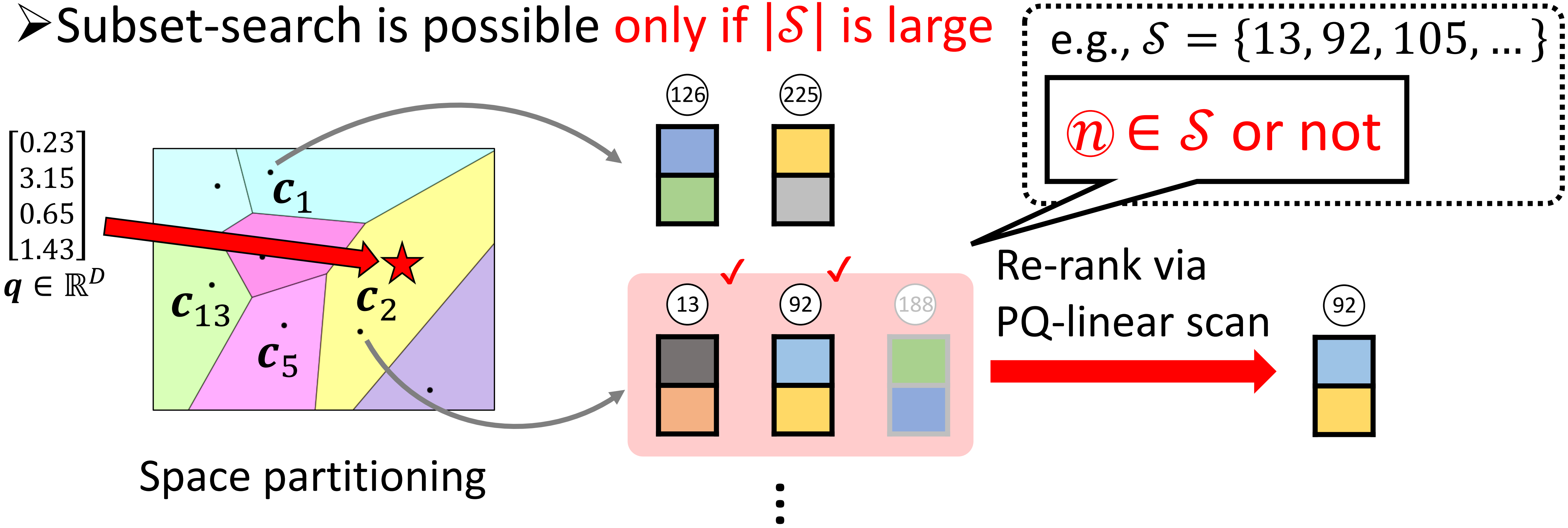


Space partitioning

slides: https://bit.ly/2P0KuW1

Preliminary: Inverted Index + PQ (IVFPQ) [Jégou+, TPAMI 11]

- Current basic data structure for a large-scale search
- Subset-search is possible only if $|\mathcal{S}|$ is large

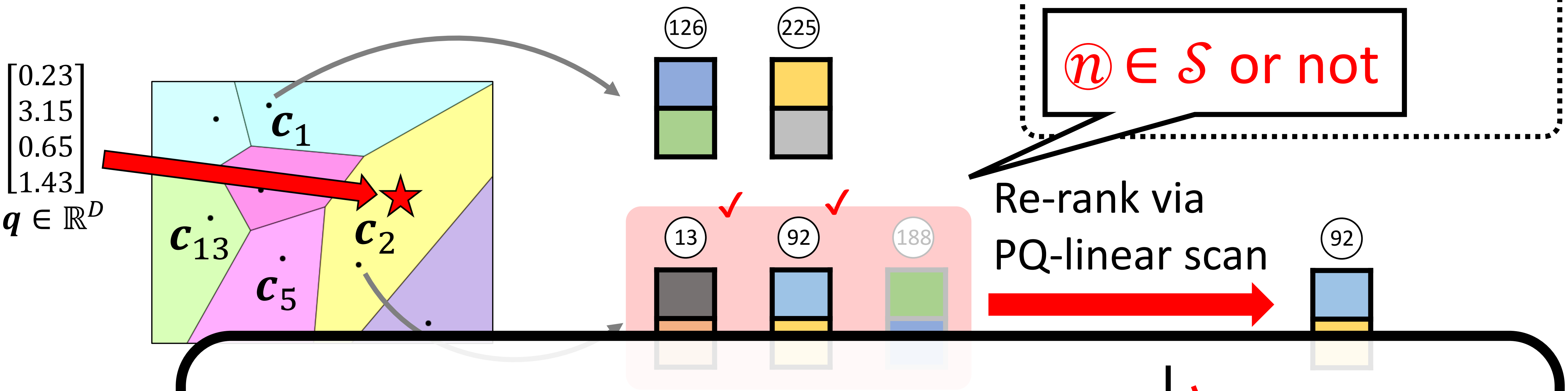


1. Find the closest space: $k^* = \text{argmin}_k \|\boldsymbol{q} - \boldsymbol{c}_k\|_2^2$
2. Focus the k^*th space, accept items $\in \mathcal{S}$
3. Re-rank the items via PQ-linear scan

slides: https://bit.ly/2P0KuW1

Preliminary: Inverted Index + PQ (IVFPQ) [Jégou+, TPAMI 11]

- ➢Current basic data structure for a large-scale search
- ➢Subset-search is possible **only if $|\mathcal{S}|$ is large**

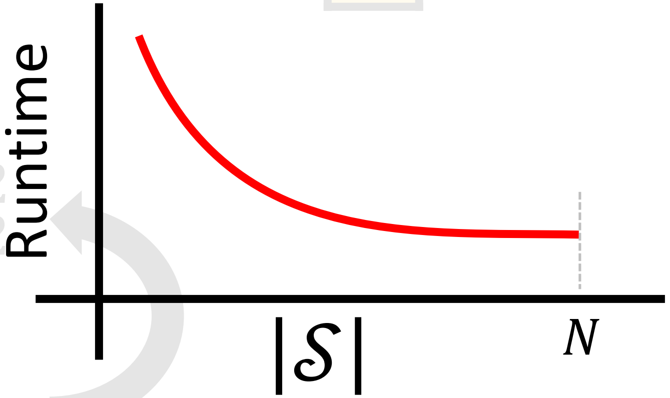


Space partitioning

1. Find the closest space: $k' = \mathrm{argmin}_k \|\boldsymbol{q} - \boldsymbol{c}_k\|_2^2$
2. Focus the k'-th space, accept items $\in \mathcal{S}$
3. Re-rank the items via PQ-linear scan

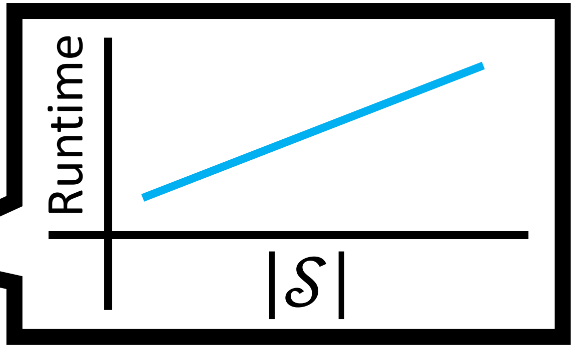
Why is it slow for small $|\mathcal{S}|$?

e.g., if $|\mathcal{S}|$ is small and they are far away from the query, we might need to scan all items

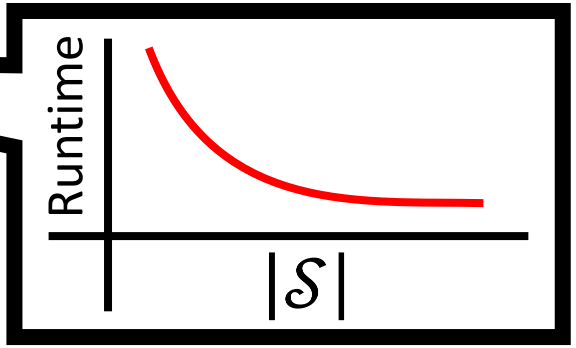


Reconfigurable inverted index (Rii)

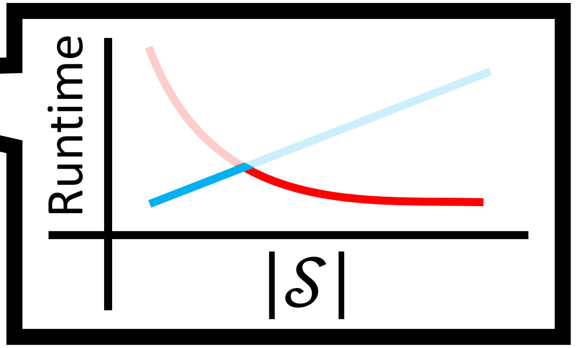
- Preliminary
  - PQ linear scan
  - IVFPQ
- **Data structure**
- Search



Fast if $|\mathcal{S}|$ is small



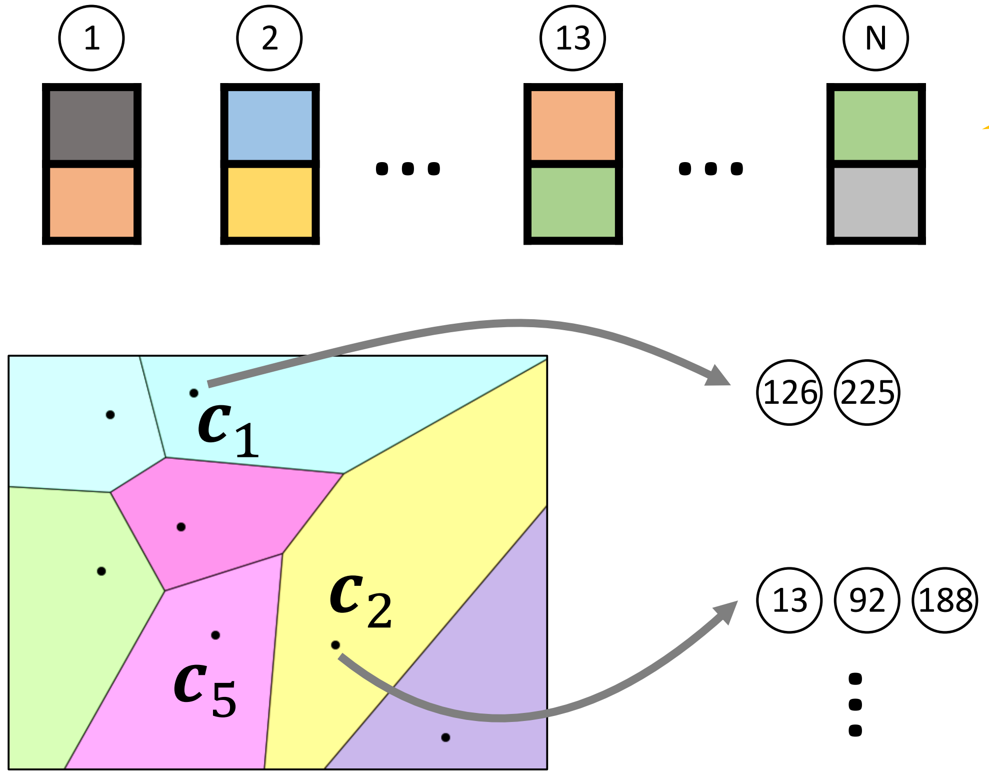
Fast if $|\mathcal{S}|$ is large



Cherry pick!
Always fast

Data structure

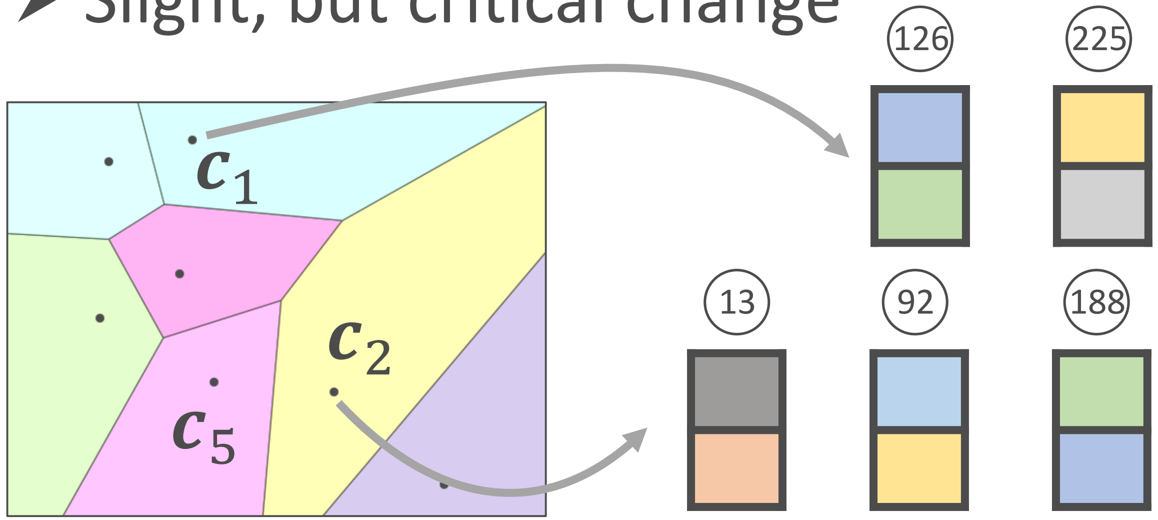
- Store (1) PQ-codes **linearly**, and (2) IDs as an inverted index
- Can run either PQ-linear-scan or IVFPQ with a **single data structure**



Key: store codes linearly

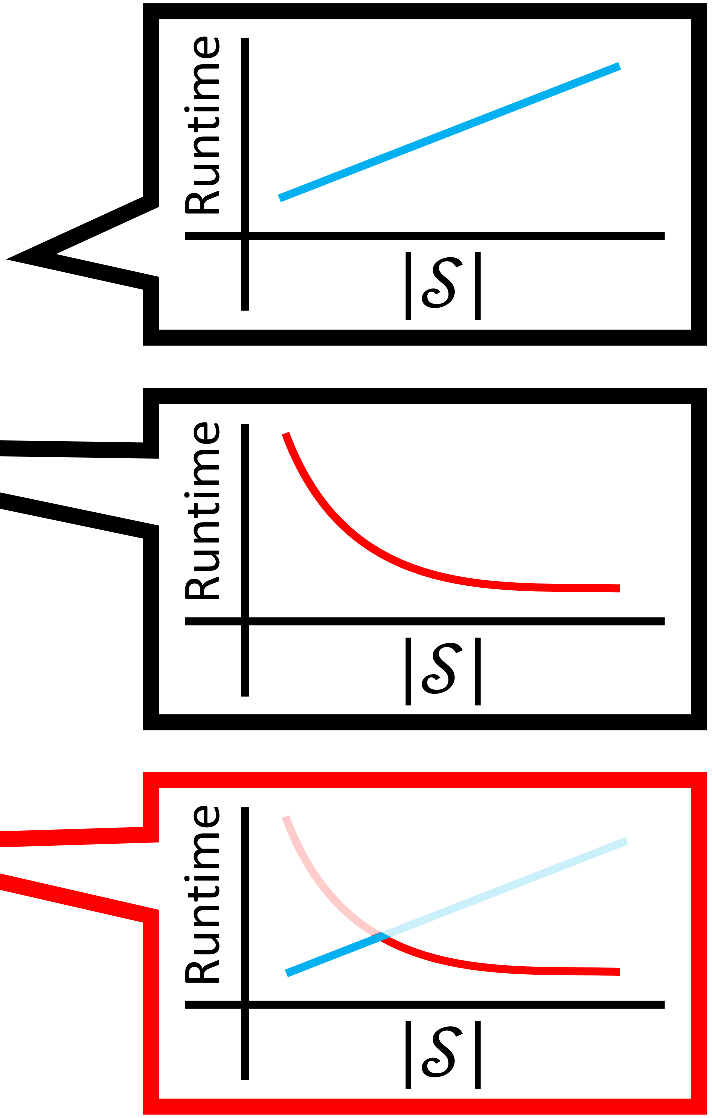
cf. IVFPQ

- PQ-codes are also chunked. Natural
- Slight, but critical change



Reconfigurable inverted index (Rii)

- Preliminary
  - PQ linear scan
  - IVFPQ
- Data structure
- Search



Fast if $|\mathcal{S}|$ is small

Fast if $|\mathcal{S}|$ is large

Cherry pick! Always fast

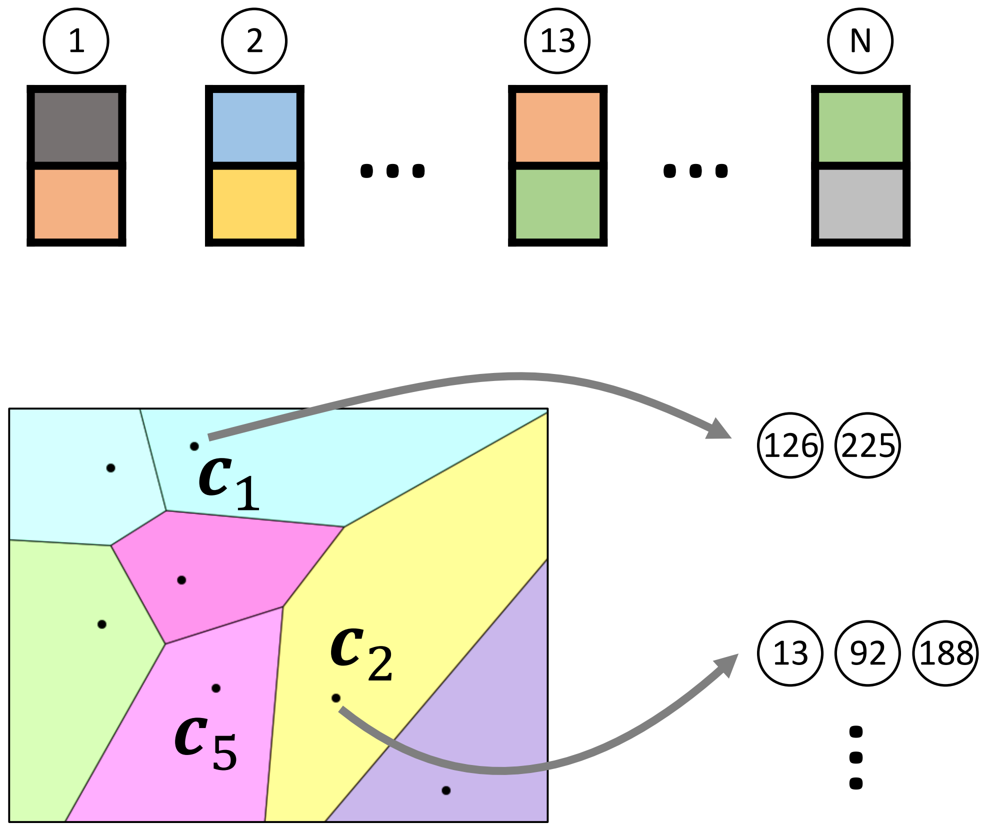
Search

slides: https://bit.ly/2P0KuW1

- If $|\mathcal{S}|$ is small, run PQ-linear scan
- If $|\mathcal{S}|$ is large, run IVFPQ

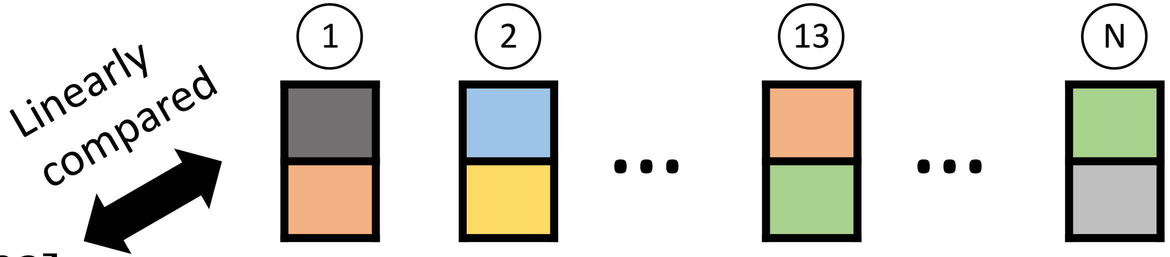
$$\begin{bmatrix}0.23\\3.15\\0.65\\1.43\end{bmatrix}$$

$\boldsymbol{q} \in \mathbb{R}^D$

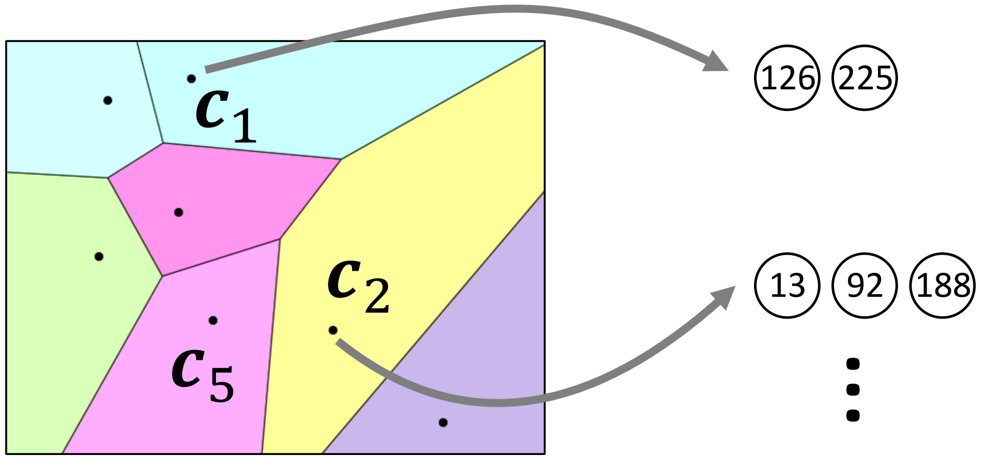


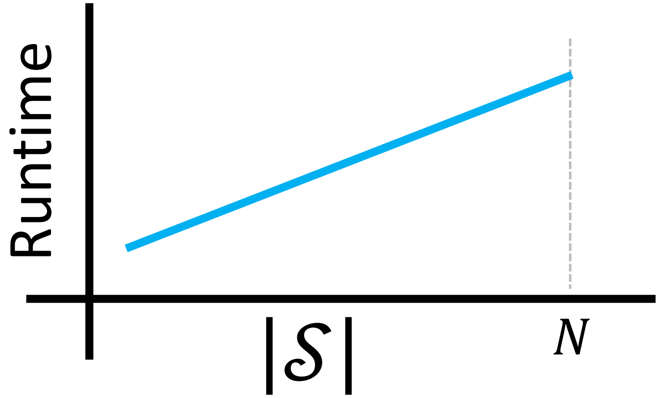
Search

- ➢If $|\mathcal{S}|$ is small, run PQ-linear scan
- ➢If $|\mathcal{S}|$ is large, run IVFPQ



$$\begin{bmatrix}0.23\\3.15\\0.65\\1.43\end{bmatrix}$$
$\boldsymbol{q} \in \mathbb{R}^D$

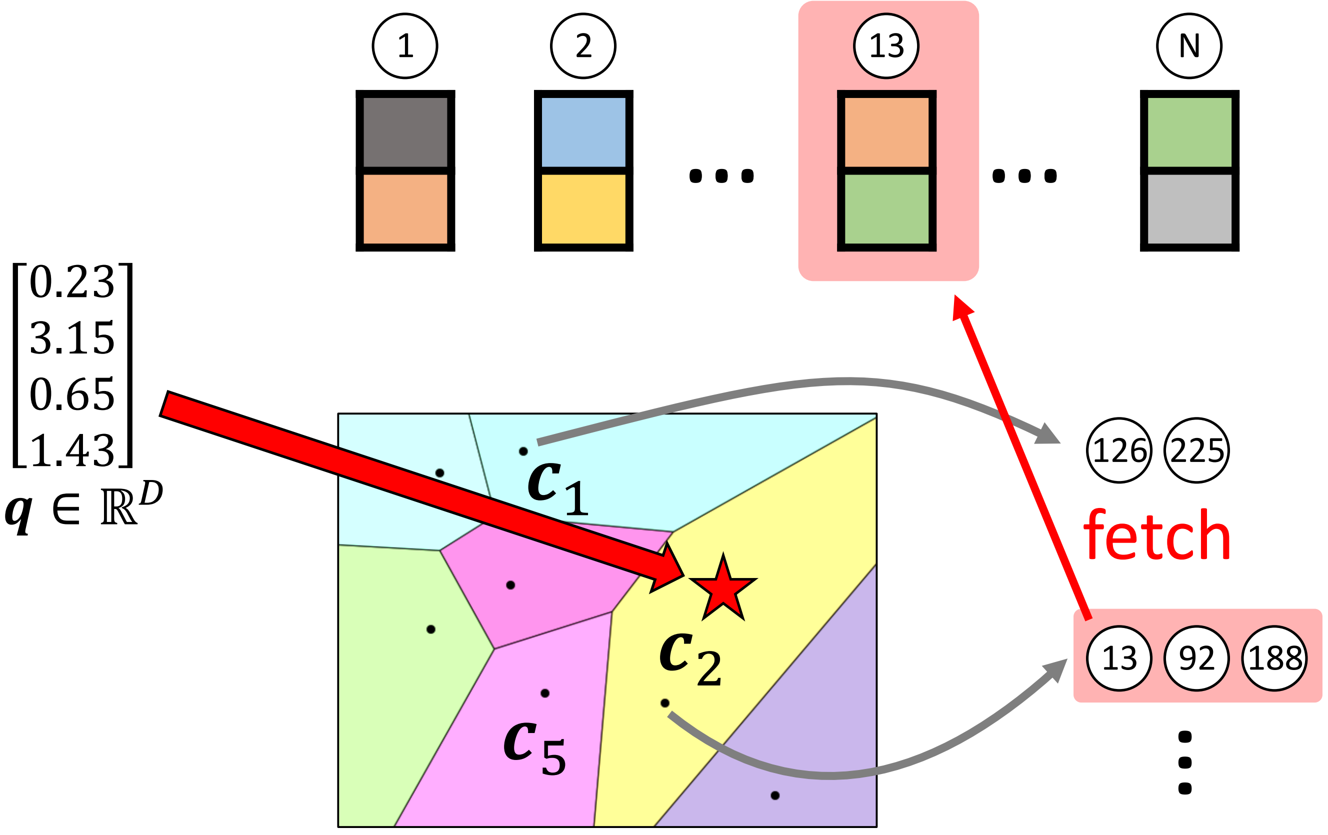


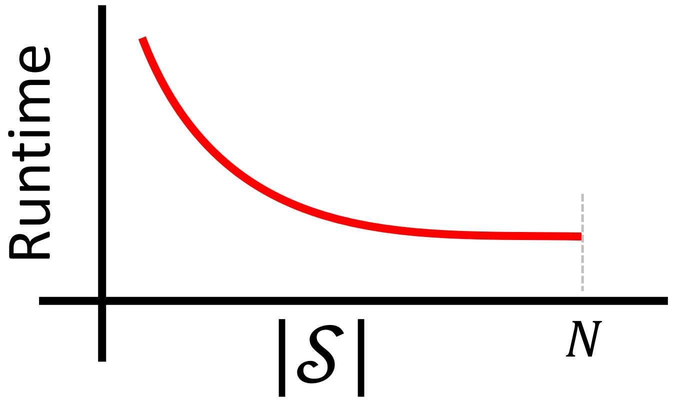


slides: https://bit.ly/2P0KuW1

Search

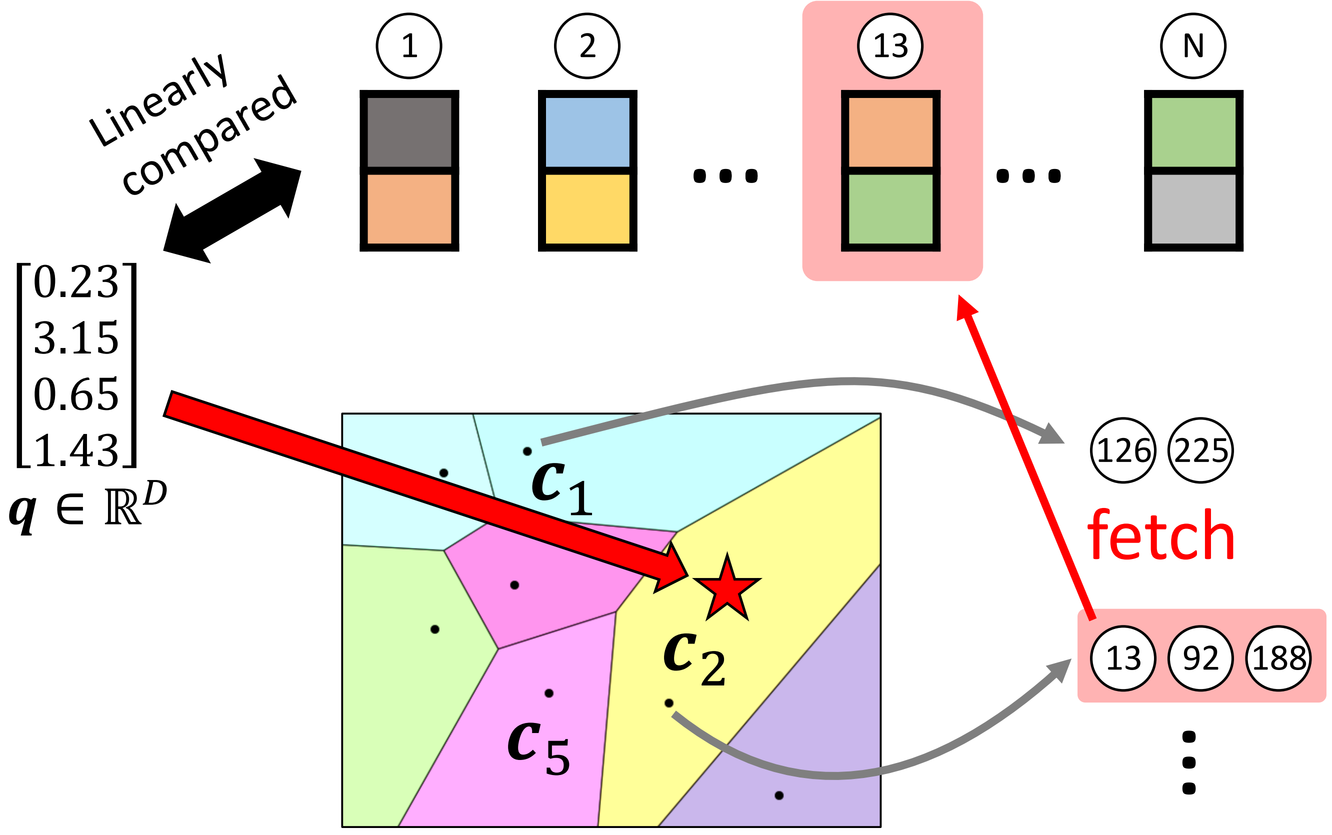
- If $|\mathcal{S}|$ is small, run PQ-linear scan
- If $|\mathcal{S}|$ is large, run IVFPQ

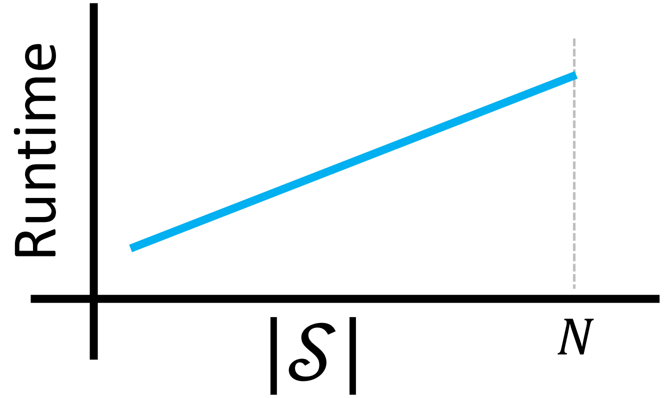


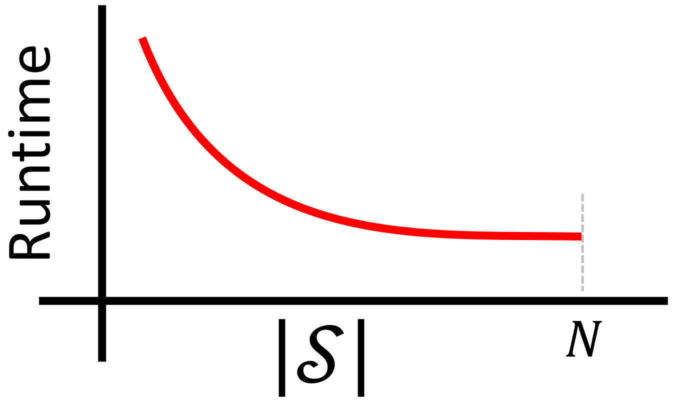


Search

- If $|\mathcal{S}|$ is small, run PQ-linear scan
- If $|\mathcal{S}|$ is large, run IVFPQ





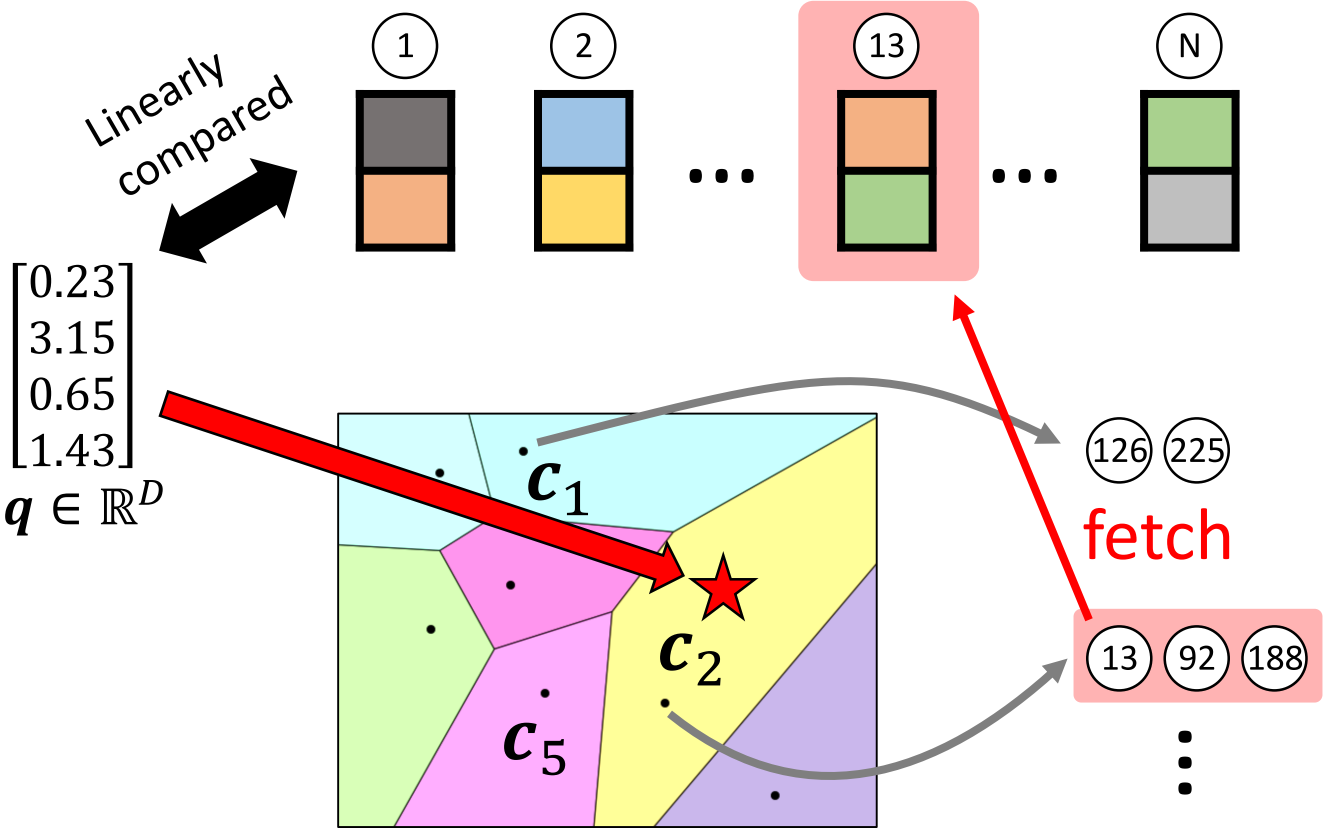


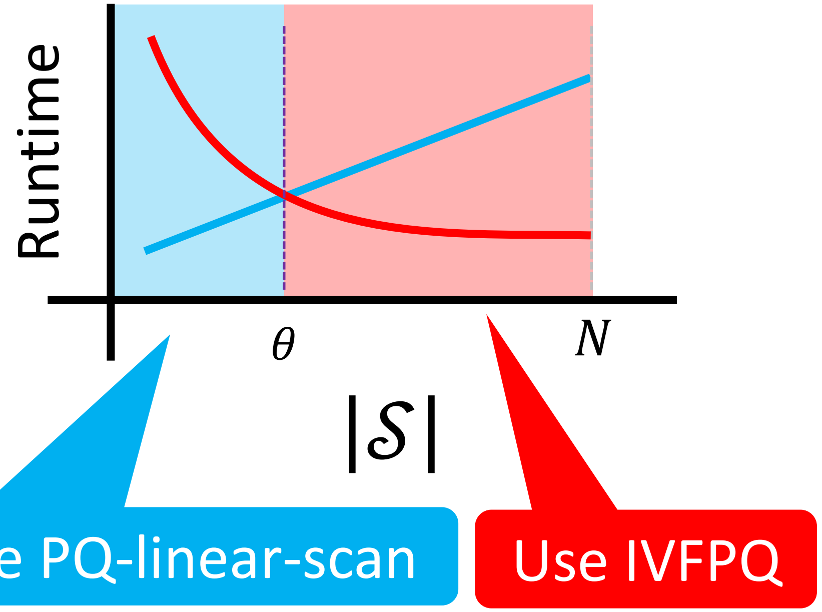
slides: https://bit.ly/2P0KuW1

Search

- If $|\mathcal{S}|$ is small, run PQ-linear scan
- If $|\mathcal{S}|$ is large, run IVFPQ

- Set a threshold θ
- Key: Switch two methods based on $|\mathcal{S}| \lessgtr \theta$



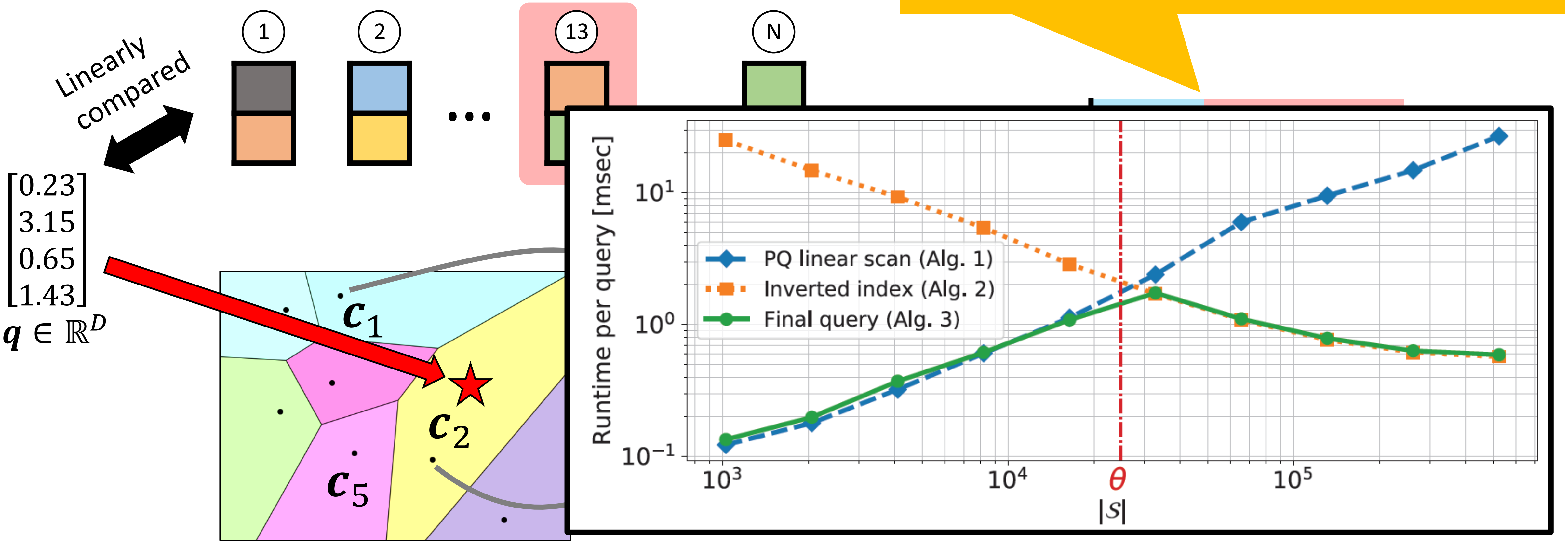


slides: https://bit.ly/2P0KuW1

Search

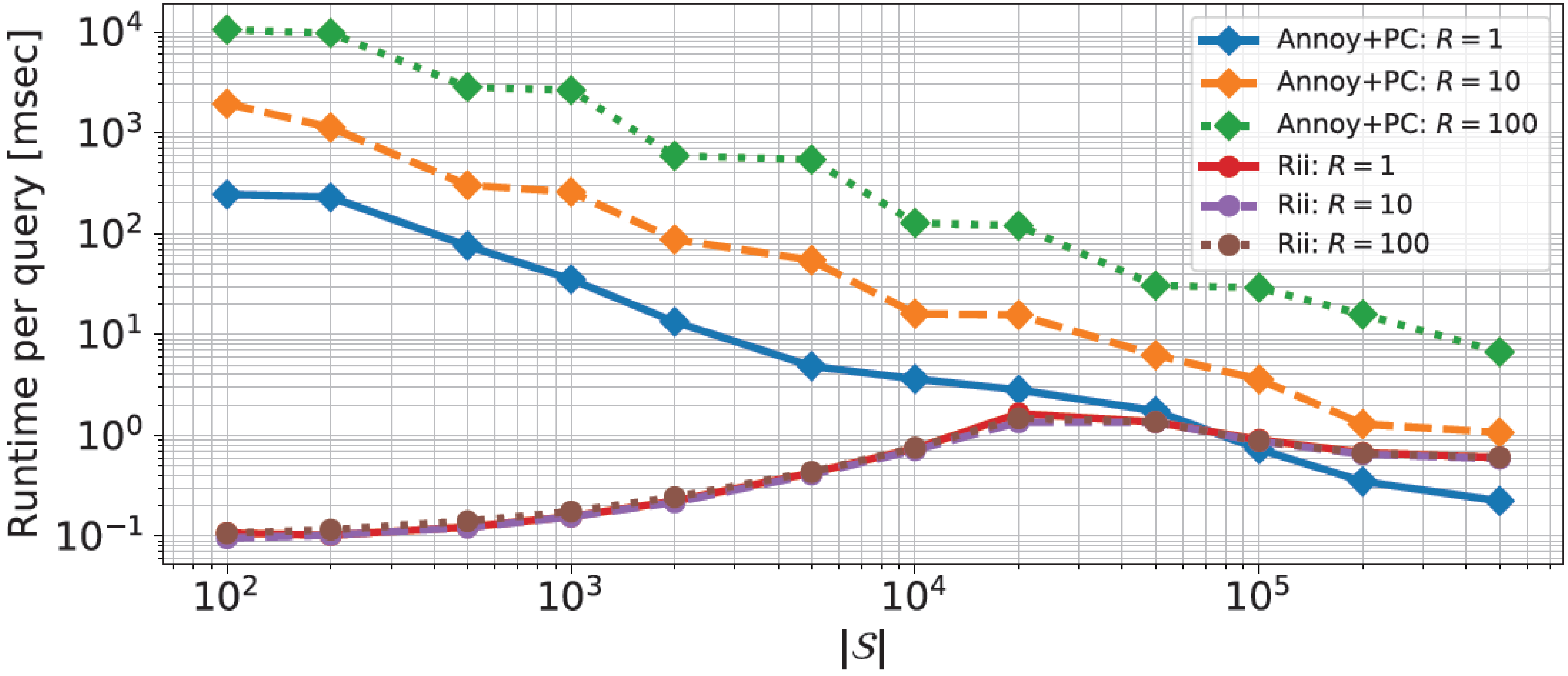
- If $|\mathcal{S}|$ is small, run PQ-linear scan
- If $|\mathcal{S}|$ is large, run IVFPQ

- Set a threshold θ
- Key: Switch two methods based on $|\mathcal{S}| \lessgtr \theta$



Evaluation

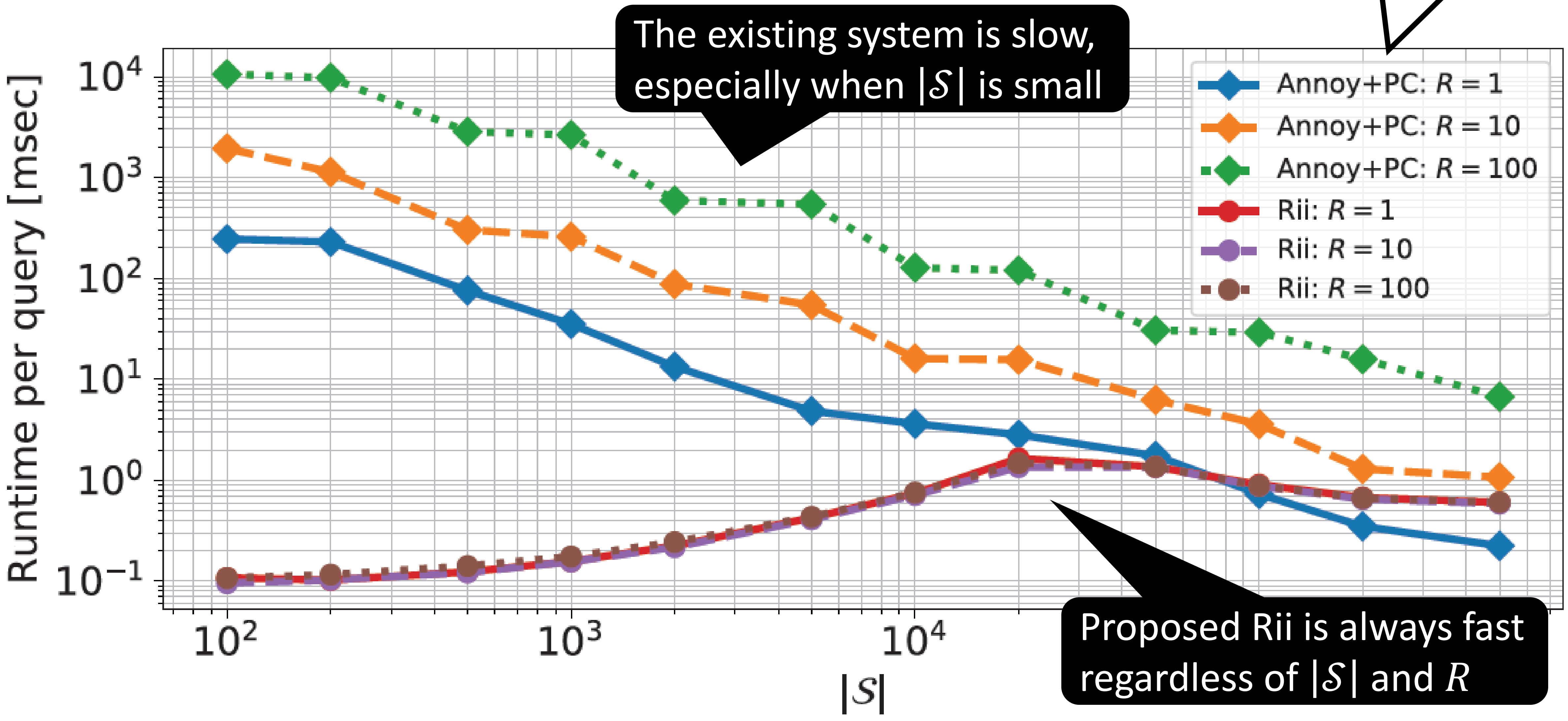
➢SIFT1M ($N = 10^6$, $D = 128$). Results for top-R search



Evaluation

- Existing system: Annoy
- Force to search a subset

➢SIFT1M ($N = 10^6$, $D = 128$). Results for top-R search



slides: https://bit.ly/2P0KuW1

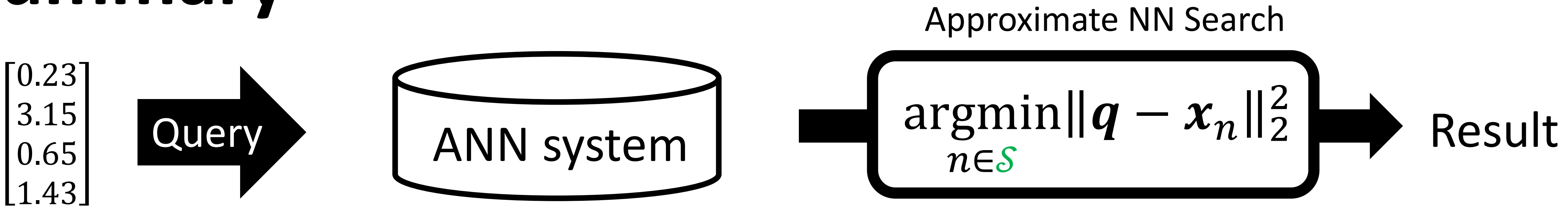
```
$ pip install rii
```

https://github.com/matsui528/rii

```python
import rii
import nanopq

# Prepare a PQ/OPQ codec with M=32 sub spaces
codec = nanopq.PQ(M=32).fit(vecs=Xt)  # Trained using Xt

# Instantiate a Rii class with the codec
e = rii.Rii(fine_quantizer=codec)

# Add vectors
e.add_configure(vecs=X)

# Search
ids, dists = e.query(q=q, topk=3, target_ids=S)
print(ids, dists)  # e.g., [7484 8173 1556] [15.062 15.385 16.169]
```

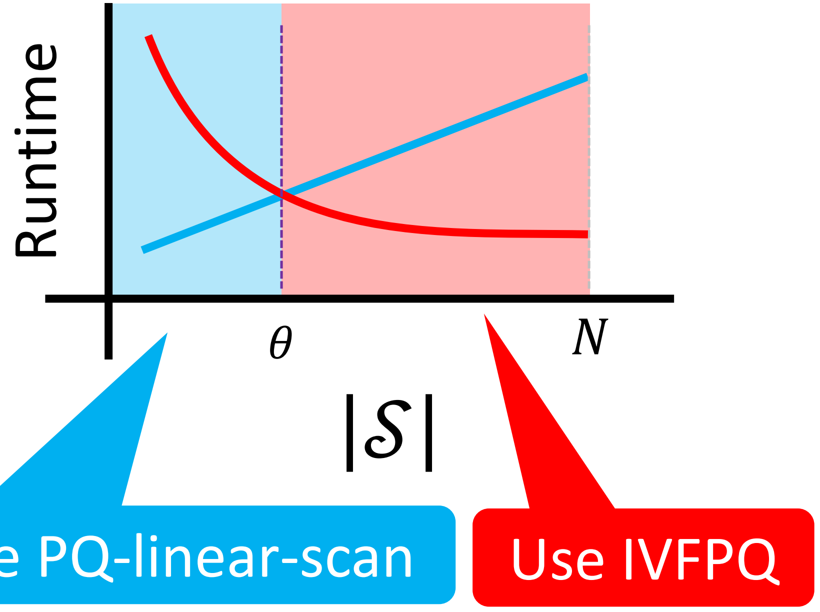
slides: https://bit.ly/2P0KuW1

Summary



Reconfigurable inverted index:

- Store PQ-codes linearly
- Switch method based on $|\mathcal{S}|$



- PyPI:

```
$ pip install rii
```

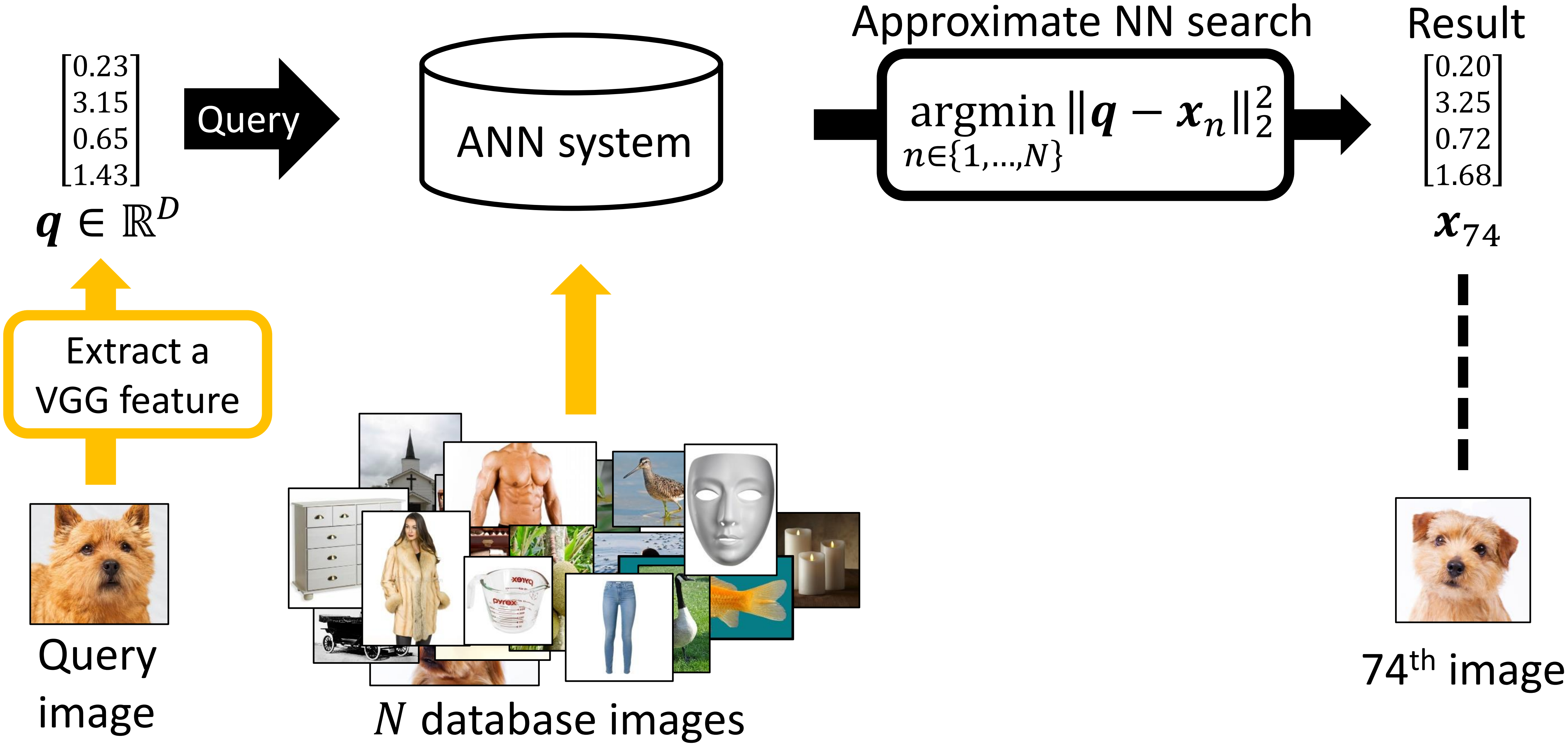
- One thing I couldn’t mention:

<u>Reconfiguration</u>: the system remains fast even after many new items are added

- See our paper, or come to our poster:
  - ✓ <u>Poster session 5 (15:30 – 16:30)</u>

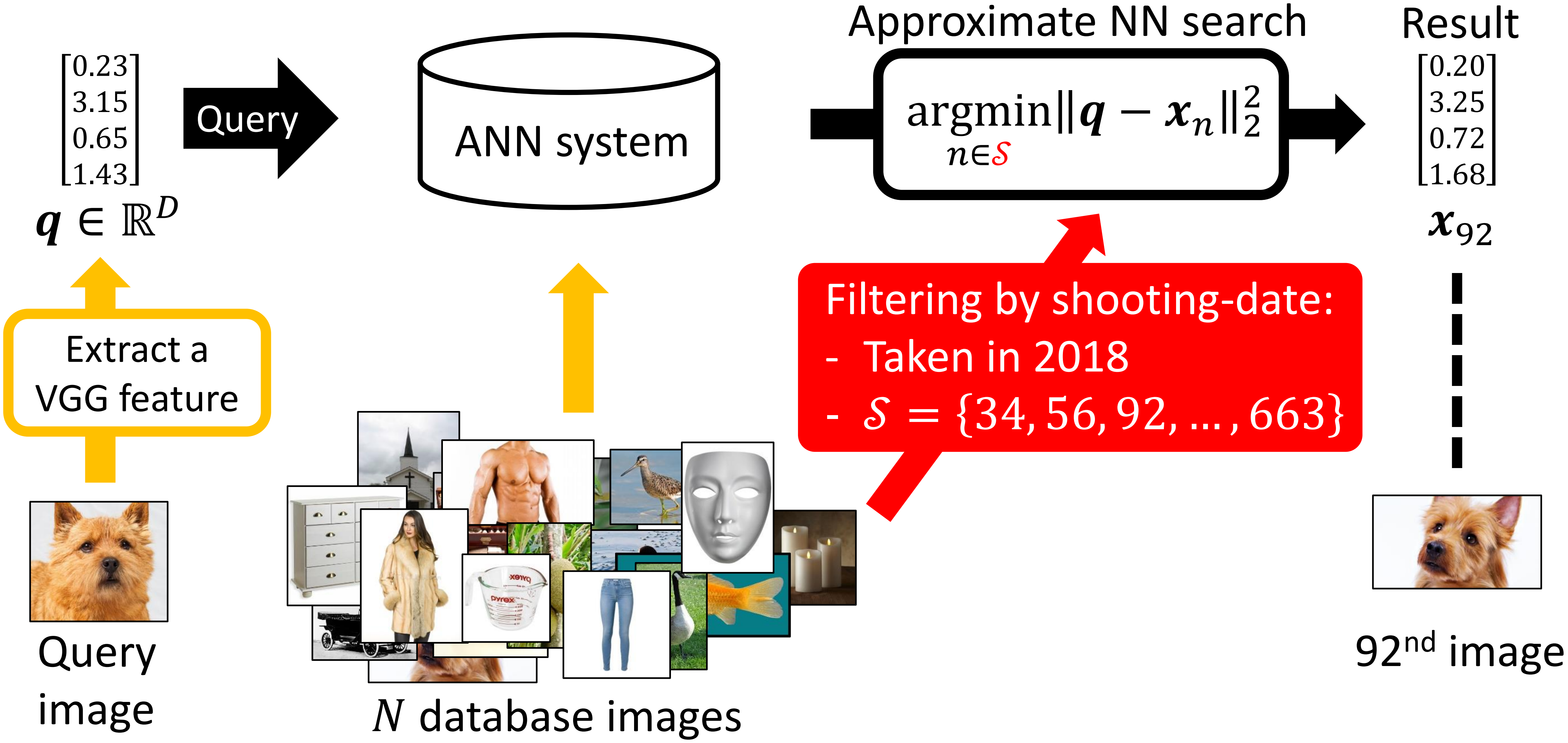
slides: https://bit.ly/2P0KuW1

Example: Image search



slides: https://bit.ly/2P0KuW1

Example of subset search for image



slides: https://bit.ly/2P0KuW1

Evaluation

- ➢Extensive comparison against existing methods
- ➢For a fixed accuracy (Recall@1), check runtime and its disk space

Dataset	Method	Parameters	Recall@1 (fixed)	Runtime/query	Disk space	Build time
SIFT1M	Annoy [11]	$n_{\text{trees}} = 2000, k_{\text{search}} = 400$	0.67	0.18 ms	1703 MB	899 sec
	FALCONN [1, 41]	$n_{\text{probes}} = 16$	0.63	0.87 ms	-	**1.8 sec**
	NMSLIB (HNSW) [14, 33, 39]	efS = 4	0.67	**0.043 ms**	669 MB	436 sec
	Faiss (IVFADC) [25, 26]	$K = 10^3, M = 64, n_{\text{probe}} = 4$	0.67	0.61 ms	73 MB	30 sec
	Rii (proposed)	$K = 10^3, M = 64, L = 5000$	0.64	0.73 ms	**69 MB**	82 sec
	Rii-OPQ (proposed)	$K = 10^3, M = 64, L = 5000$	0.65	0.82 ms	**69 MB**	85 sec
GIST1M	Annoy [11]	$n_{\text{trees}} = 2000, k_{\text{search}} = 2000$	0.49	1.2 ms	5023 MB	2088 sec
	FALCONN [1, 41]	$n_{\text{probes}} = 512$	0.53	8.6 ms	-	**7.2 sec**
	NMSLIB (HNSW) [14, 33, 39]	efS = 8	0.49	**0.19 ms**	3997 MB	1576 sec
	Faiss (IVFADC) [25, 26]	$K = 10^3, M = 240, n_{\text{probe}} = 8$	0.52	3.8 ms	253 MB	51 sec
	Rii (proposed)	$K = 10^3, M = 240, L = 8000$	0.45	3.2 ms	**246 MB**	353 sec
	Rii-OPQ (proposed)	$K = 10^3, M = 240, L = 8000$	0.50	3.8 ms	249 MB	388 sec

slides: https://bit.ly/2P0KuW1

Evaluation

- Extensive comparison against existing methods
- For a fixed accuracy (Recall@1), check runtime and its disk space

Dataset	Method	Parameters	Recall@1 (fixed)	Runtime/query	Disk space	Build time
SIFT1M	Annoy [11]	$n_{\text{trees}} = 2000, k_{\text{search}} = 400$	0.67	0.18 ms	1703 MB	899 sec
	FALCONN [1, 41]	$n_{\text{probes}} = 16$	0.63	0.87 ms	-	**1.8 sec**
	NMSLIB (HNSW) [14, 33, 39]	efS = 4	0.67	**0.043 ms**	669 MB	436 sec
	Faiss (IVFADC) [25, 26]	$K = 10^3, M = 64,$ [illegible] $= 4$	0.67	0.61 ms	73 MB	30 sec
	Rii (proposed)	$K = 10^3, M$ [illegible] $= 5000$	0.64	0.73 ms	**69 MB**	82 sec
	Rii-OPQ (proposed)	$K = 10^3$ [illegible] $= 5000$	0.65	0.82 ms	**69 MB**	85 sec
[illegible]	Annoy [11]	[illegible] $= 2000$	0.49	1.2 ms	5023 MB	2088 sec
	[illegible]	[illegible]	0.53	8.6 ms	-	**7.2 sec**
	[illegible]	[illegible]	0.49	**0.19 ms**	3997 MB	1576 sec
	[illegible]	[illegible]	0.52	3.8 ms	253 MB	51 sec
	[illegible]	[illegible]	0.45	3.2 ms	**246 MB**	353 sec
	Rii-OPQ (proposed)	$K = 10^3, M = 240, L = 8000$	0.50	3.8 ms	249 MB	388 sec

NMSLIB is extremely fast, but consume relatively large disk space (~memory)

Evaluation

- ➢Extensive comparison against existing methods
- ➢For a fixed accuracy (Recall@1), check runtime and its disk space

Dataset	Method	Parameters	Recall@1 (fixed)	Runtime/query	Disk space	Build time
SIFT1M	Annoy [11]	$n_{\text{trees}} = 2000, k_{\text{search}} = 400$	0.67	0.18 ms	1703 MB	899 sec
	FALCONN [1, 41]	$n_{\text{probes}} = 16$	0.63	0.87 ms	-	**1.8 sec**
	NMSLIB (HNSW) [14, 33, 39]	efS = 4	0.67	**0.043 ms**	669 MB	436 sec
	Faiss (IVFADC) [25, 26]	$K = 10^3, M = 64$ [illegible] robe $= 4$	0.67	0.61 ms	73 MB	30 sec
	Rii (proposed)	$K = 10^3, M$ [illegible] $= 5000$	0.64	0.73 ms	**69 MB**	82 sec
	Rii-OPQ (proposed)	$K = 10^3$ [illegible] $= 5000$	0.65	0.82 ms	**69 MB**	85 sec
[illegible]	Annoy [11]	[illegible] $= 2000$	0.49	1.2 ms	[illegible]023 MB	2088 sec
	[illegible]	[illegible]	0.53	8.6 ms	[illegible]	**7.2 sec**
	[illegible]	[illegible]	0.49	**0.19 ms**	[illegible]MB	1576 sec
	[illegible]	[illegible]	0.52	3.8 ms	[illegible]	51 sec
	[illegible]	[illegible]	0.45	3.2 ms	[illegible]	353 sec
	Rii-OPQ (proposed)	$K = 10^{[illegible]}, M = 240, L =$ [illegible]	[illegible]	[illegible]	[illegible]	[illegible]

NMSLIB is extremely fast, but consume relatively large disk space (~memory)

Proposed Rii achieved a comparative performance with Faiss