CVPR 2023 Tutorial on

Neural Search in Action

Yusuke Matsui
The University of Tokyo

Martin Aumüller
IT University of Copenhagen

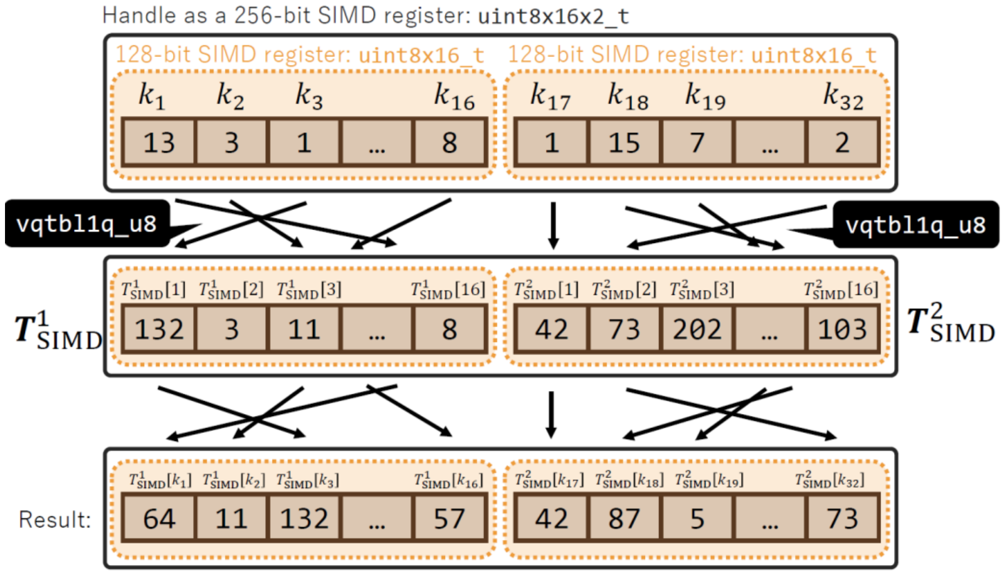
Han Xiao
Jina AI

Yusuke Matsui

Lecturer (Assistant Professor), the University of Tokyo, Japan

http://yusukematsui.me @utokyo_bunny @matsui528

- ✓ Image retrieval
- ✓ Large-scale indexing



ARM 4-bit PQ [Matsui+, ICASSP 22]

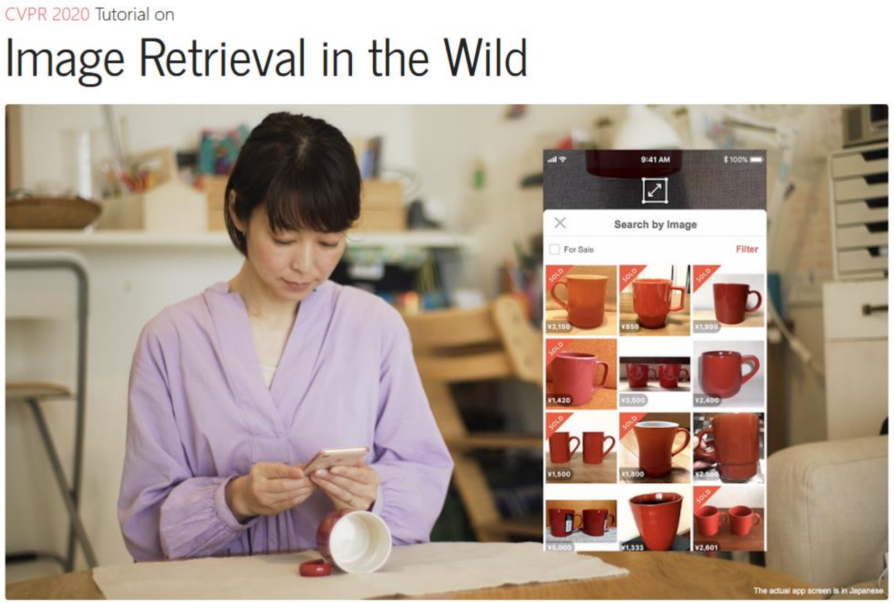


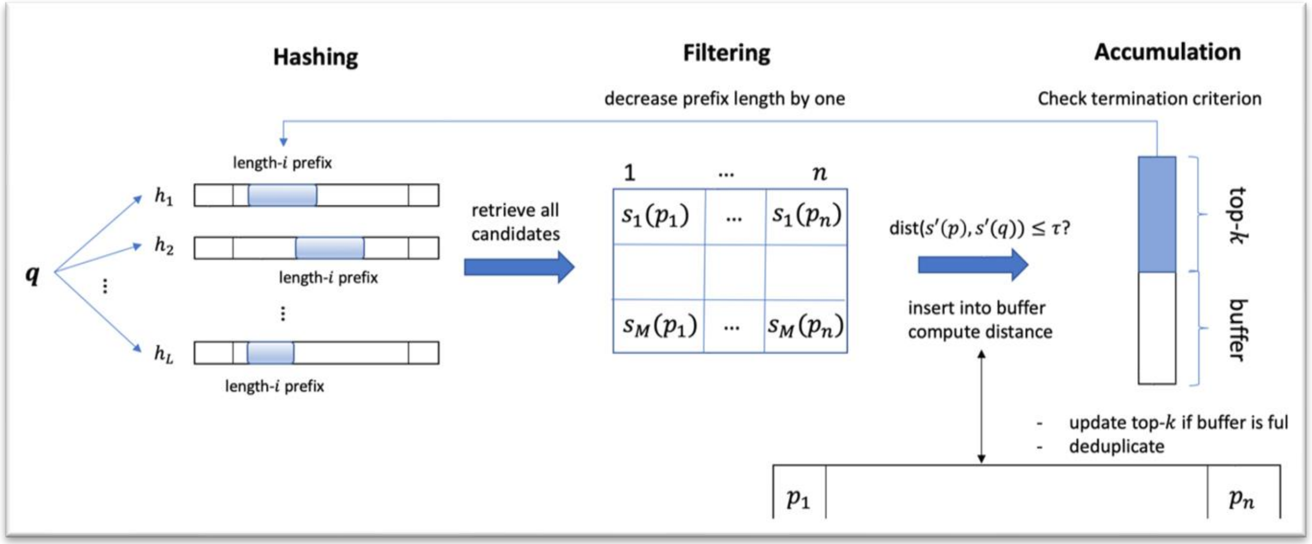
Image Retrieval in the Wild [Matsui+, CVPR 20, tutorial]

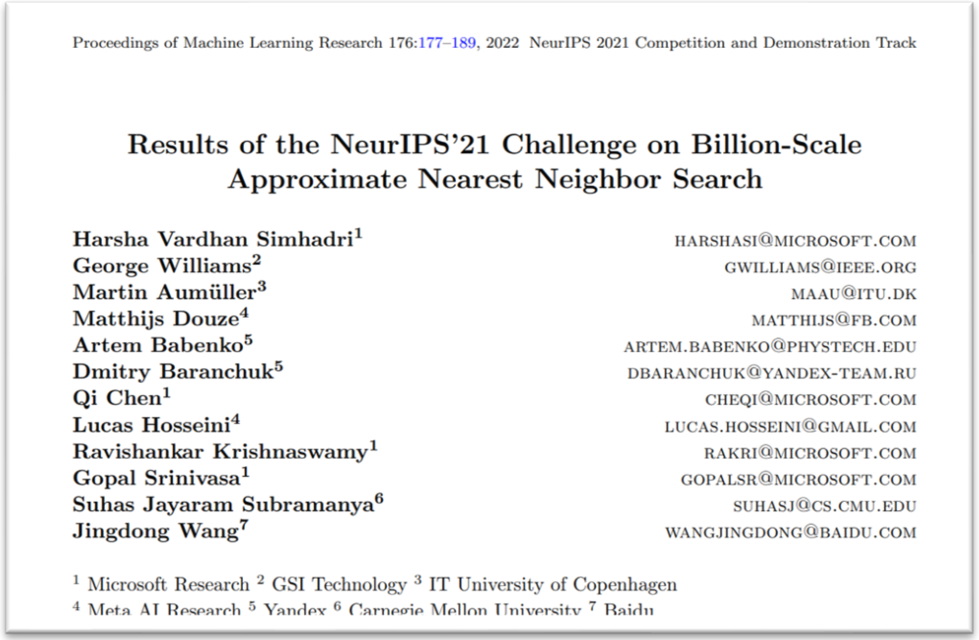
Martin Aumüller

Associate Professor, IT University of Copenhagen, Denmark

http://itu.dk/people/maau @maumueller

- ✓ Similarity search using hashing
- ✓ Benchmarking & workload generation





Proceedings of Machine Learning Research 176:177–189, 2022 NeurIPS 2021 Competition and Demonstration Track

Results of the NeurIPS’21 Challenge on Billion-Scale Approximate Nearest Neighbor Search

Harsha Vardhan Simhadri[1]	HARSHASI@MICROSOFT.COM
George Williams[2]	GWILLIAMS@IEEE.ORG
Martin Aumüller[3]	MAAU@ITU.DK
Matthijs Douze[4]	MATTHIJS@FB.COM
Artem Babenko[5]	ARTEM.BABENKO@PHYSTECH.EDU
Dmitry Baranchuk[5]	DBARANCHUK@YANDEX-TEAM.RU
Qi Chen[1]	CHEQI@MICROSOFT.COM
Lucas Hosseini[4]	LUCAS.HOSSEINI@GMAIL.COM
Ravishankar Krishnaswamy[1]	RAKRI@MICROSOFT.COM
Gopal Srinivasa[1]	GOPALSR@MICROSOFT.COM
Suhas Jayaram Subramanya[6]	SUHASJ@CS.CMU.EDU
Jingdong Wang[7]	WANGJINGDONG@BAIDU.COM

[1] Microsoft Research [2] GSI Technology [3] IT University of Copenhagen
[4] Meta AI Research [5] Yandex [6] Carnegie Mellon University [7] Baidu

PUFFINN
[Aumüller+, ESA 2019]

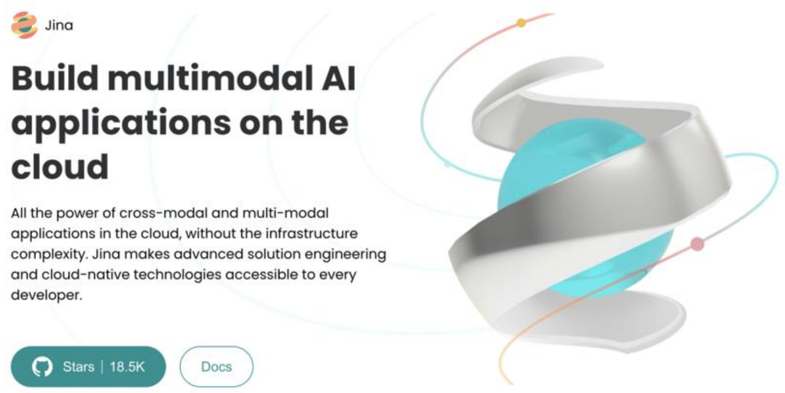
Billion-Scale ANN Challenge
[Aumüller+, NeurIPS 21, Competition]

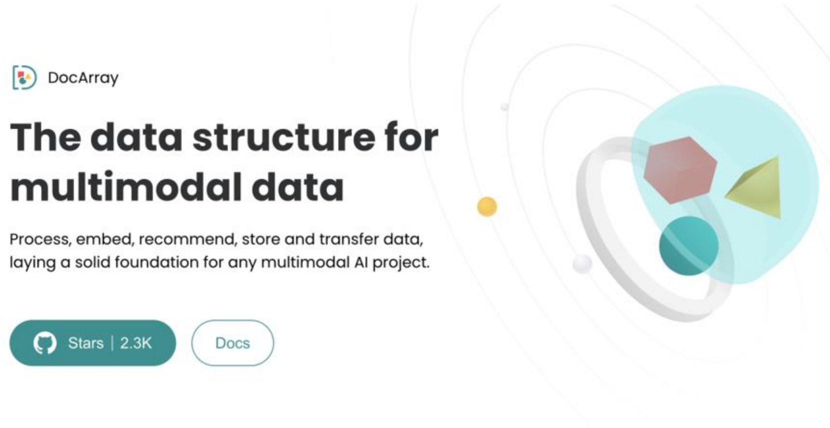
Han Xiao

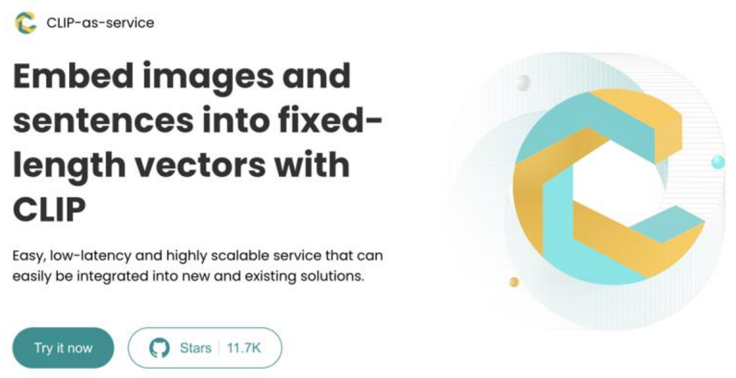
Founder & CEO of Jina AI

https://jina.ai @hxiao

- ✓ Multimodal search & generation
- ✓ Model tuning & serving; prompt tuning & serving





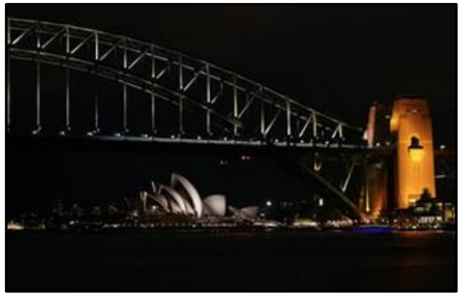


Example: Multimodal Search

“Two dogs playing in the snow”

Search

Images



...

Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Example: LLM + embedding

"Who won curling gold at the 2022 Winter Olympics?"

+

"Chinami Yoshida¥n¥n==Personal…"

"Lviv bid for the 2022 Winter…"

"2022 Olympics medal winners…"

⋮

"Damir Sharipzyanov¥n¥n=Career…"

"Niklas Edin, Oskar Eriksson, …"

Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

Target audiences

- Those who want to try Neural Search
- Those who have tried Neural Search but would like to know more about the algorithm in depth

Our talk

- Million-scale search (Yusuke)
- Billion-scale search (Martin)
- Query language (Han)

Schedule

Time	Session	Presenter
13:30 – 13:40	Opening	Yusuke Matsui
13:40 – 14:30	Theory and Applications of Graph-based Search	Yusuke Matsui
14:30 – 15:20	A Survey on Approximate Nearest Neighbors in a Billion-Scale Settings	Martin Aumüller
15:20 – 15:30	Break	
15:30 – 16:20	Query Language for Neural Search in Practical Applications	Han Xiao

JUNE 18-22, 2023
CVPR
VANCOUVER, CANADA

Theory and Applications of Graph-based Search

Yusuke Matsui
The University of Tokyo

Yusuke Matsui

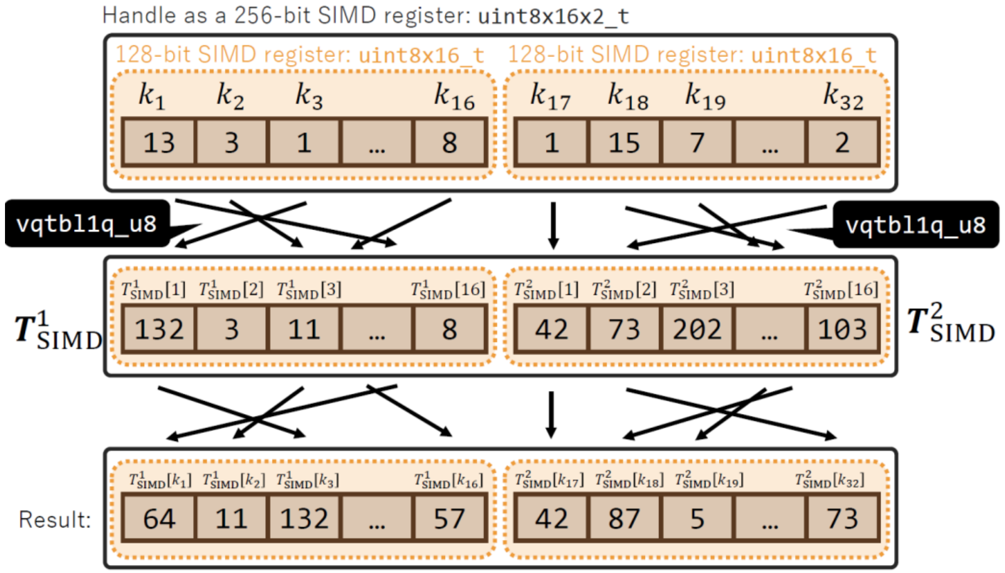
東京大学 THE UNIVERSITY OF TOKYO

Lecturer (Assistant Professor), the University of Tokyo, Japan

http://yusukematsui.me @utokyo_bunny @matsui528

✓ Image retrieval

✓ Large-scale indexing



ARM 4-bit PQ [Matsui+, ICASSP 22]

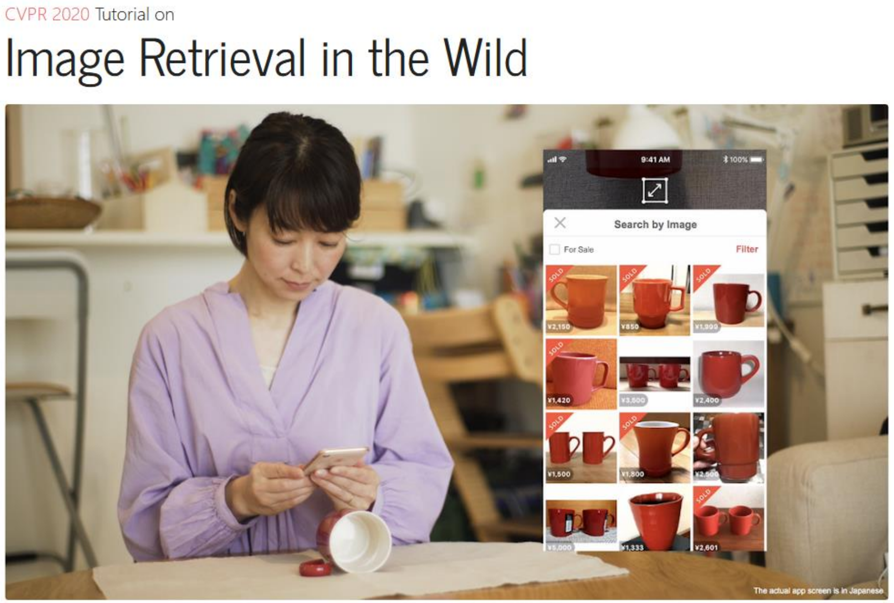
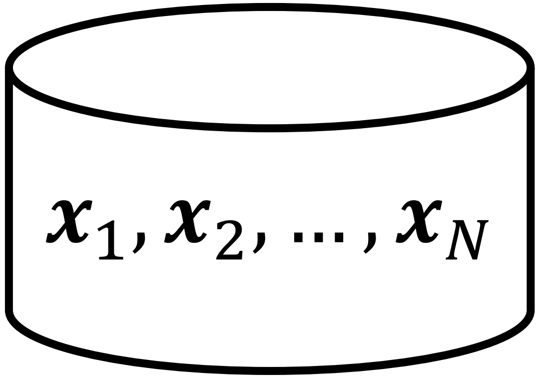


Image Retrieval in the Wild [Matsui+, CVPR 20, tutorial]

- Background
- Graph-based search
  - Basic (construction and search)
  - Observation
  - Properties
- Representative works
  - HNSW, NSG, NGT, Vamana
- Discussion

- **Background**
- Graph-based search
  - ✓ Basic (construction and search)
  - ✓ Observation
  - ✓ Properties
- Representative works
  - ✓ HNSW, NSG, NGT, Vamana
- Discussion

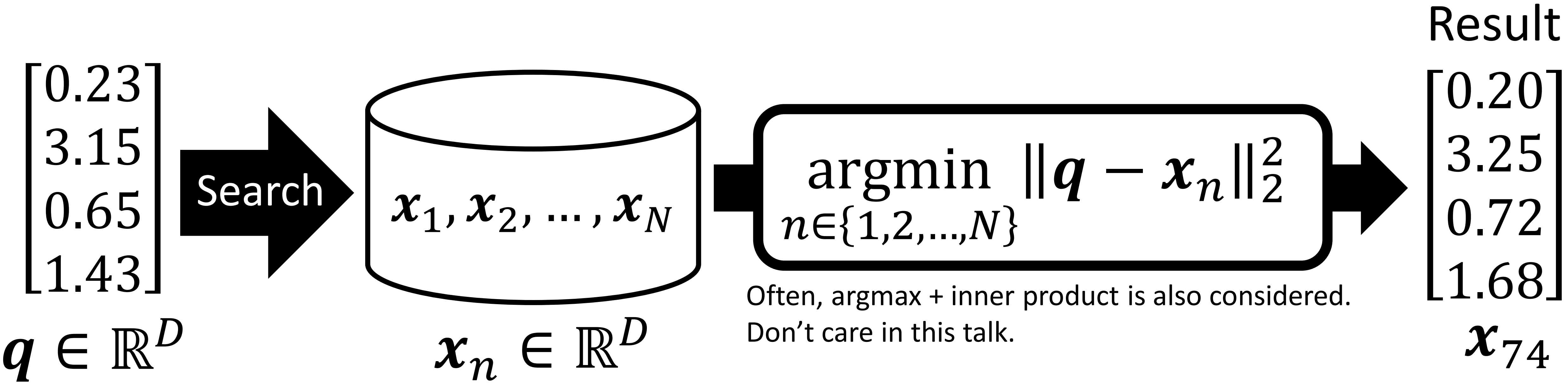
Nearest Neighbor Search; NN



$\boldsymbol{x}_n \in \mathbb{R}^D$

- N D-dim database vectors: $\{\boldsymbol{x}_n\}_{n=1}^N$

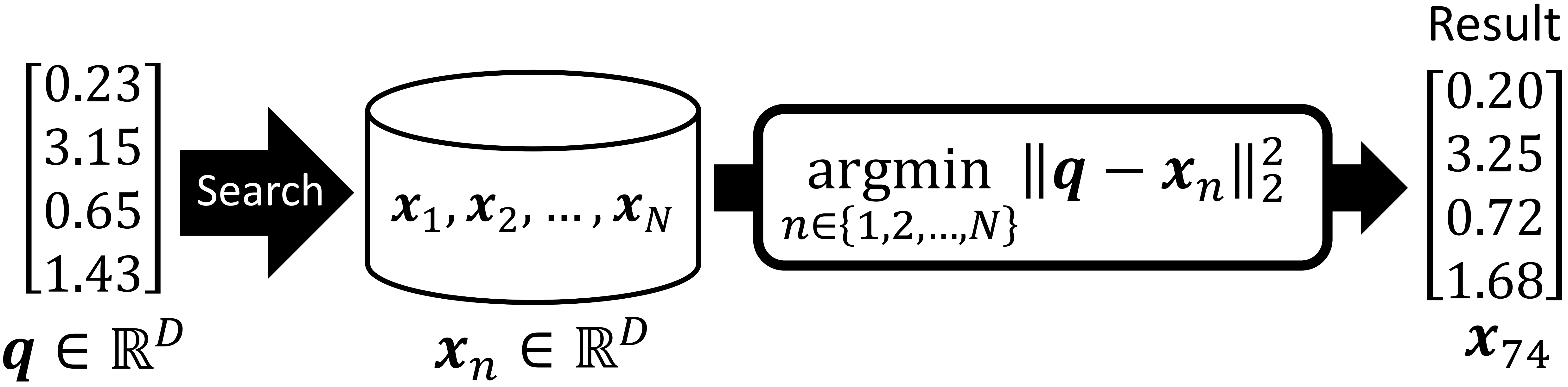
Nearest Neighbor Search; NN



Often, argmax + inner product is also considered.
Don't care in this talk.

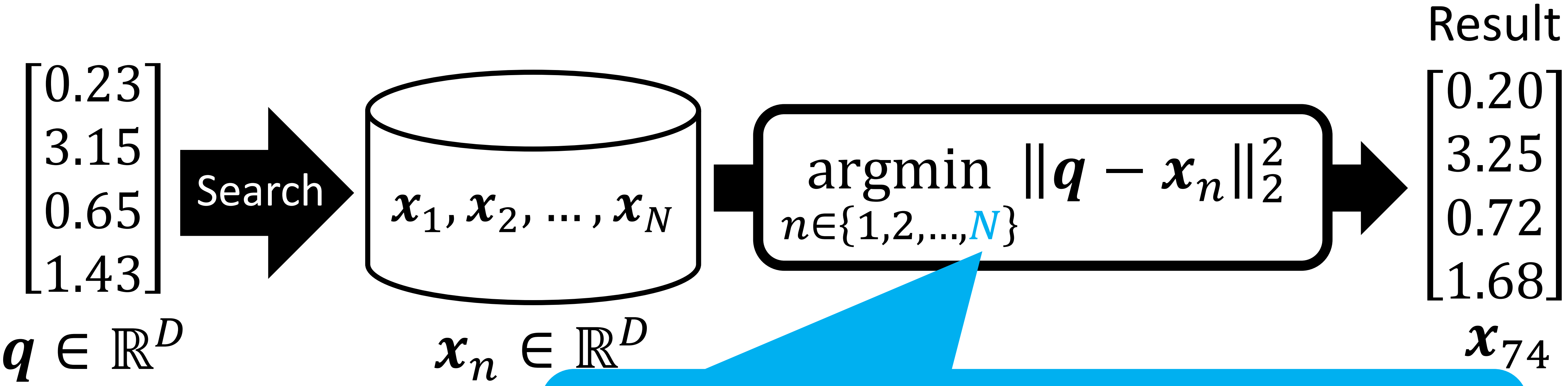
- N D-dim database vectors: $\{\boldsymbol{x}_n\}_{n=1}^{N}$
- Given a query $\boldsymbol{q}$, find the closest vector from the database
- One of the fundamental problems in computer science
- Solution: linear scan, $O(ND)$, slow ☹

Approximate Nearest Neighbor Search; ANN



- Faster search
- Don’t necessarily have to be exact neighbors
- Trade off: runtime, accuracy, and memory-consumption

Approximate Nearest Neighbor Search; ANN



- In this talk, suppose: $N < 10^9$
- All data can be loaded on memory

- Faster search
- Don’t necessarily have to be exact neighbors
- Trade off: runtime, accuracy, and memory-consumption

Real-world use cases 1: multimodal search

Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Real-world use cases 1: multimodal search

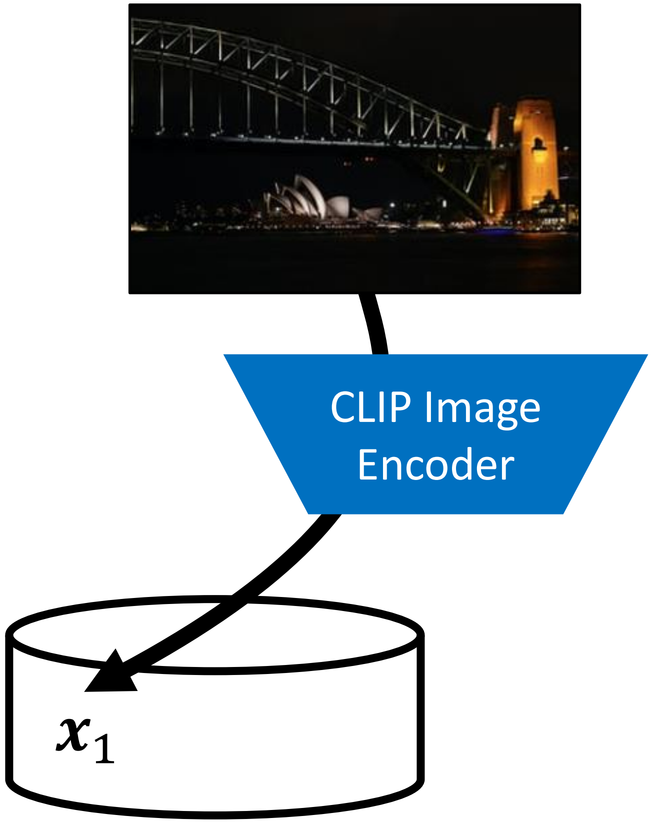


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Real-world use cases 1: multimodal search

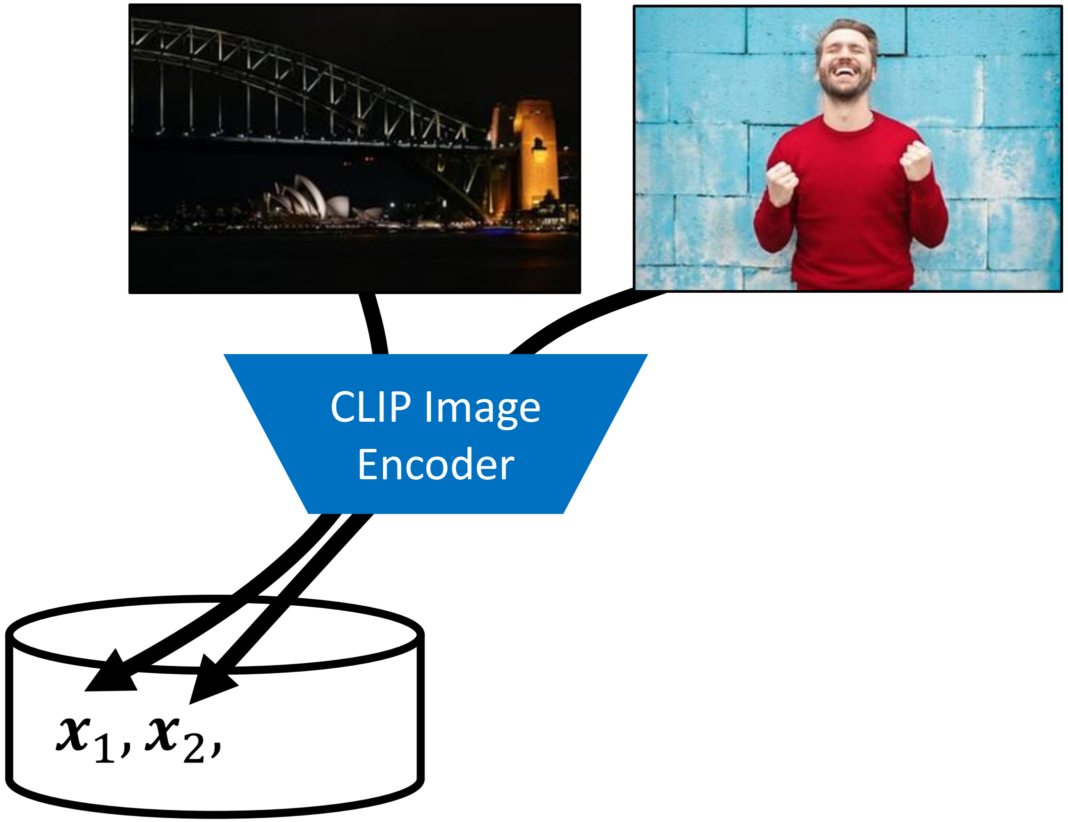


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Real-world use cases 1: multimodal search

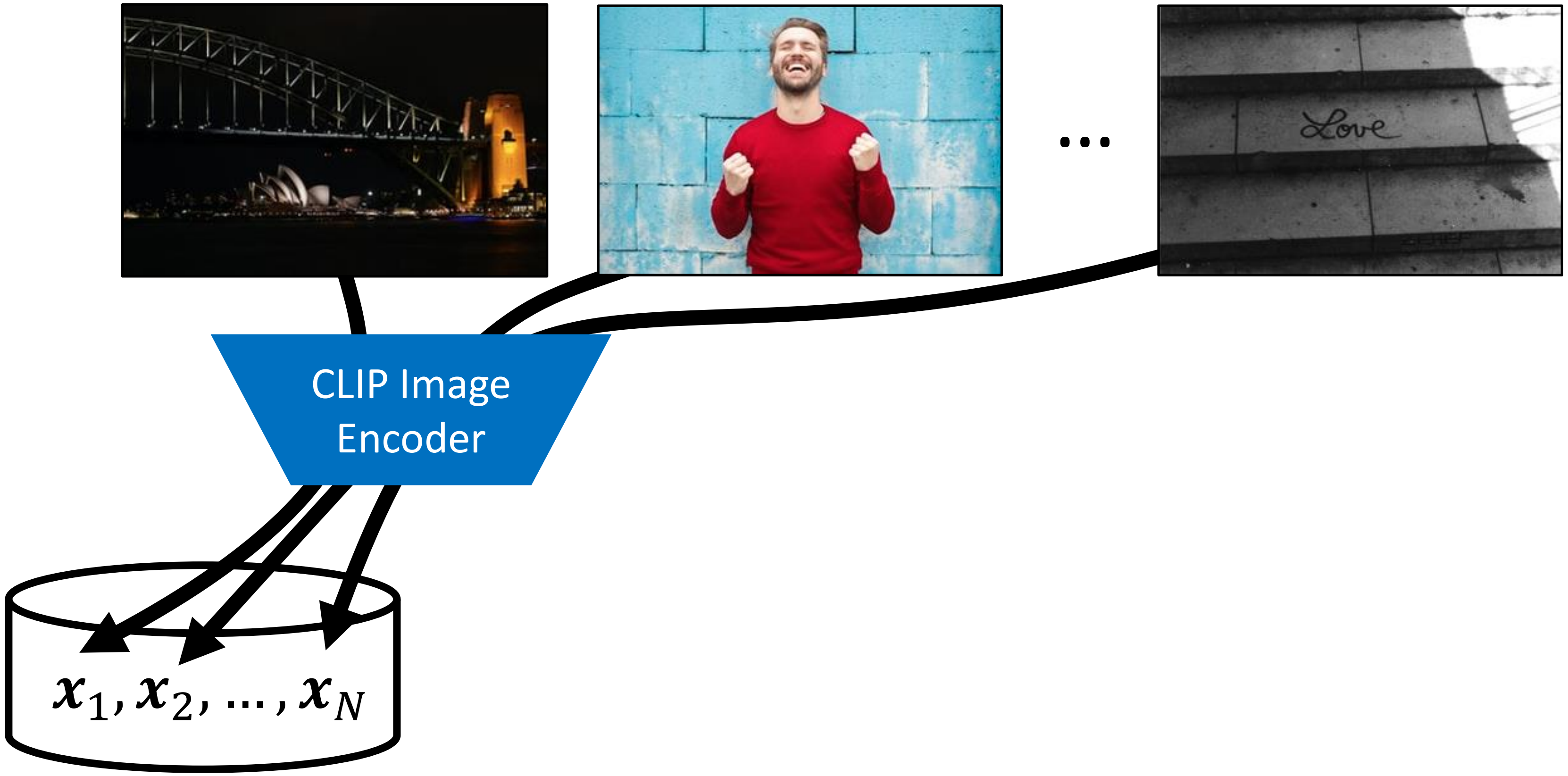


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Real-world use cases 1: multimodal search

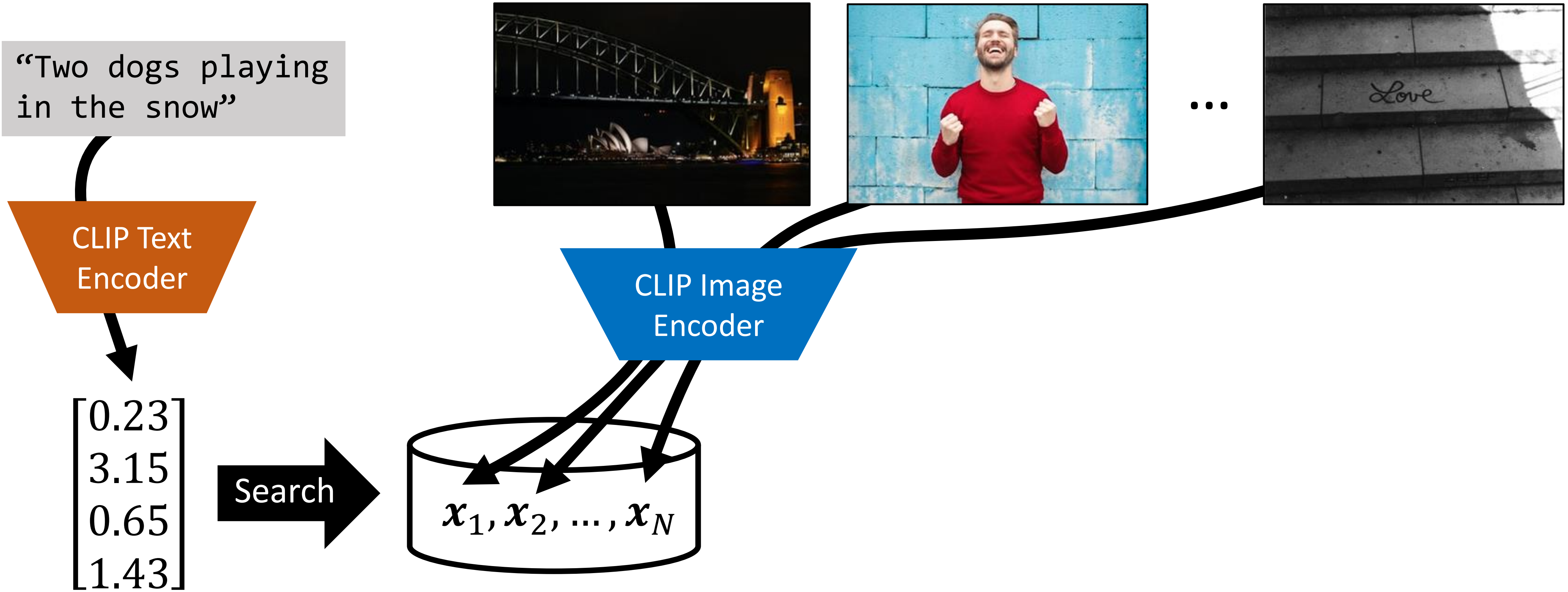


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Real-world use cases 1: multimodal search

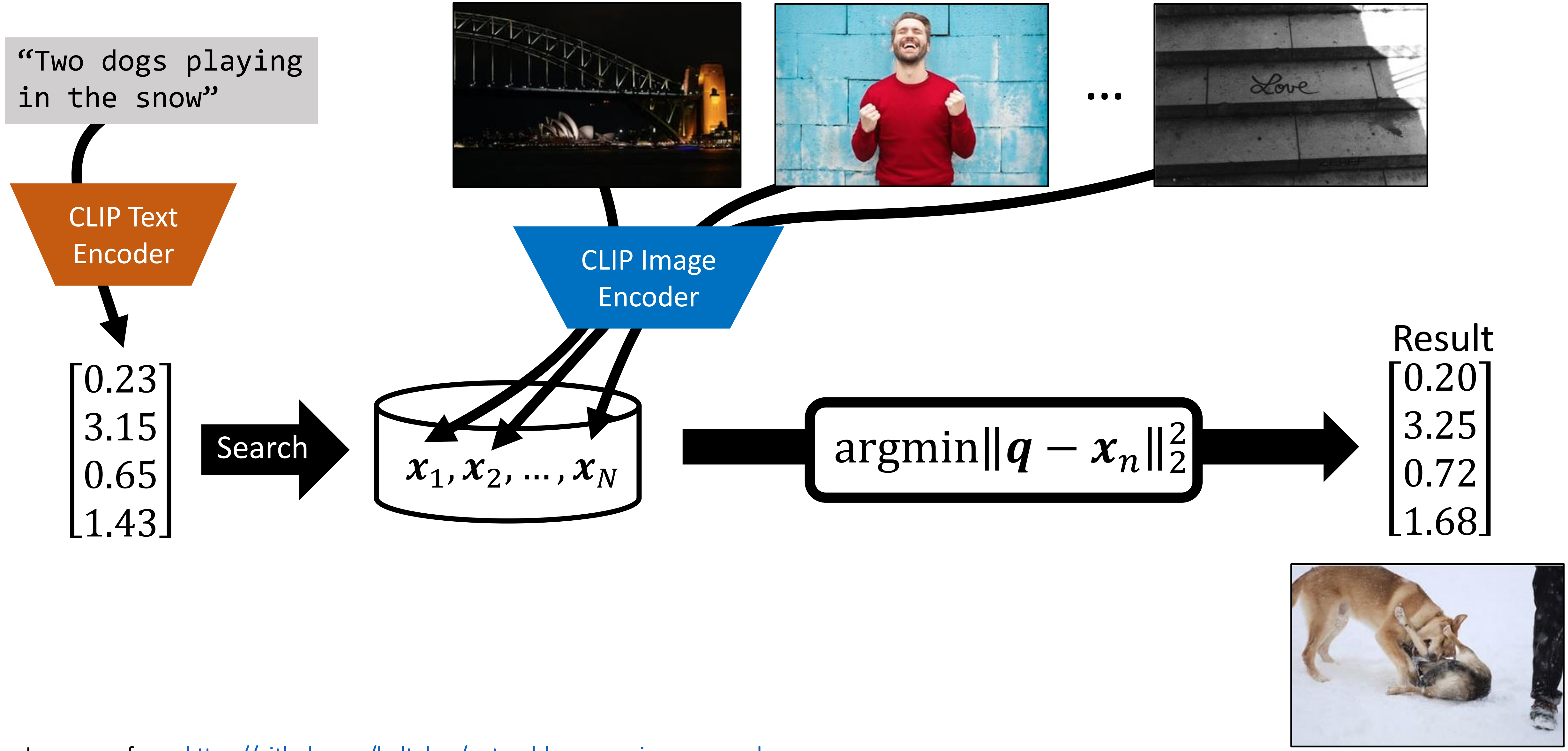
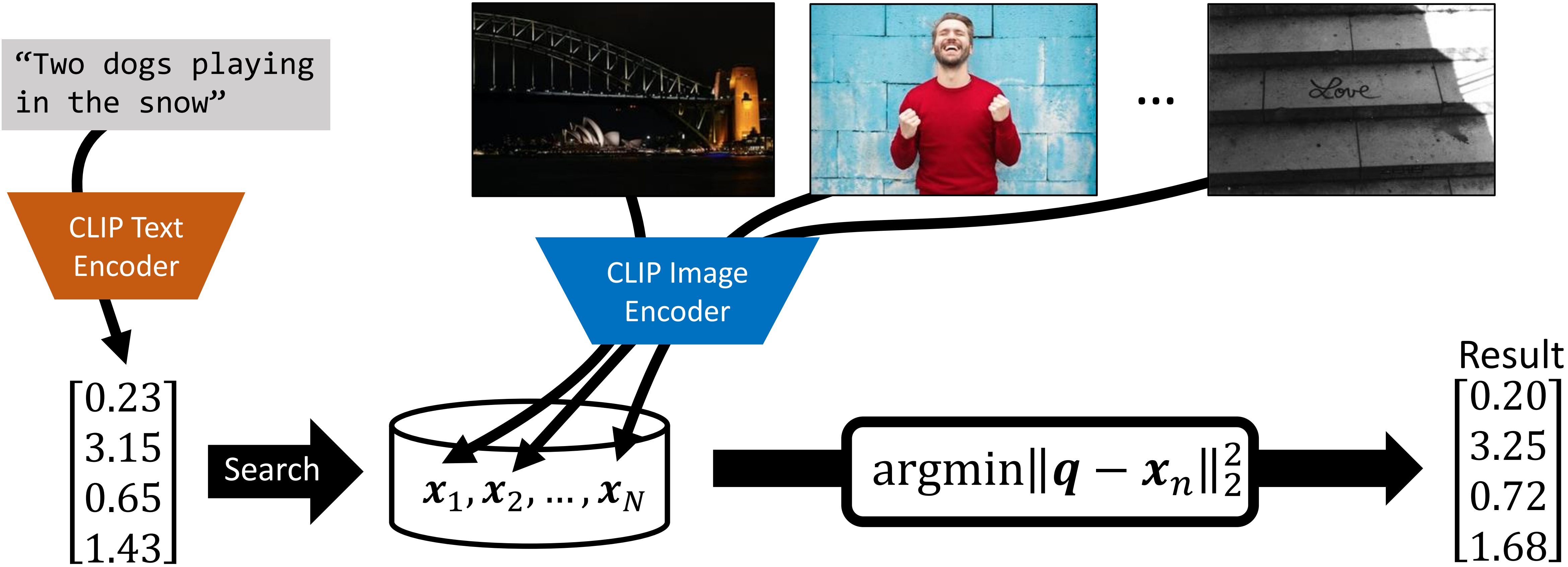


Image are from: https://github.com/haltakov/natural-language-image-search

Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Real-world use cases 1: multimodal search



- Encoder determines **the upper bound** of the accuracy of the system
- ANN determines a **trade-off** between accuracy, runtime, and memory

Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Real-world use cases 2: LLM + embedding

```
"Who won curling
gold at the 2022
Winter Olympics?"
```

ChatGPT 3.5
(trained in 2021)

Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb   Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

Real-world use cases 2: LLM + embedding

"Who won curling gold at the 2022 Winter Olympics?"

Ask

ChatGPT 3.5
(trained in 2021)

“I'm sorry, but as an AI language model, I don't have information about the future events.”

Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb

Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

Real-world use cases 2: LLM + embedding

```
"Who won curling
gold at the 2022
Winter Olympics?"
```

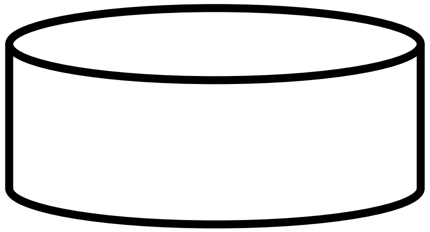
ChatGPT 3.5
(trained in 2021)

Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb   Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

Real-world use cases 2: LLM + embedding

```
"Who won curling
gold at the 2022
Winter Olympics?"
```

ChatGPT 3.5
(trained in 2021)



“Chinami Yoshida¥n¥n==Personal…”

“Lviv bid for the 2022 Winter…”

⋮

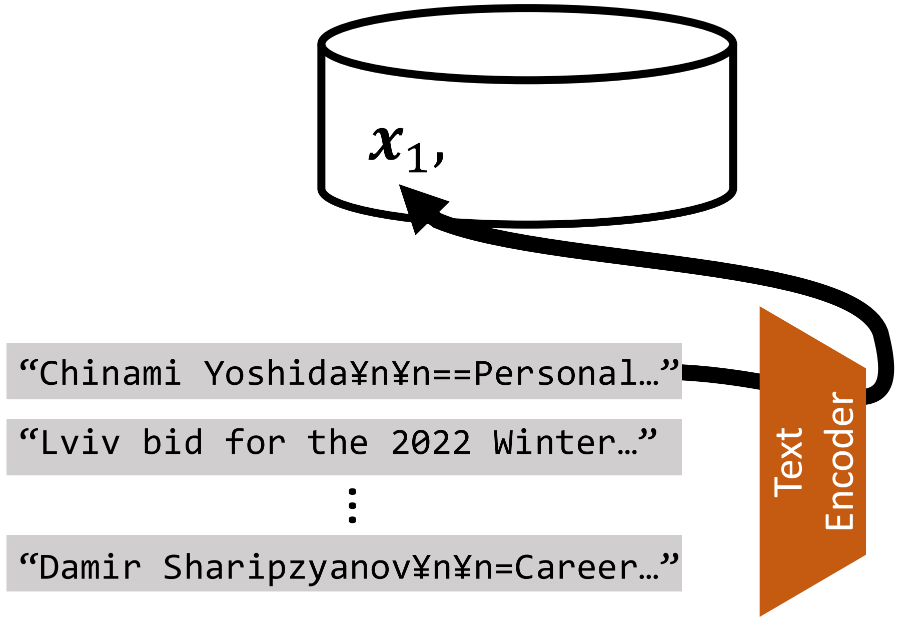
“Damir Sharipzyanov¥n¥n=Career…”

Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

Real-world use cases 2: LLM + embedding

```
"Who won curling
gold at the 2022
Winter Olympics?"
```

ChatGPT 3.5
(trained in 2021)

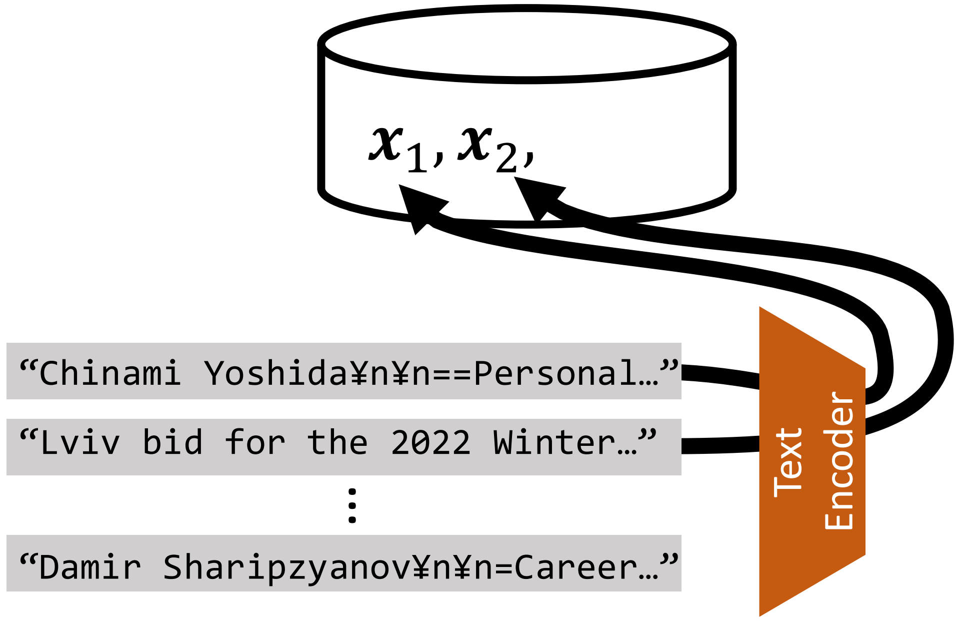


Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

Real-world use cases 2: LLM + embedding

"Who won curling gold at the 2022 Winter Olympics?"

ChatGPT 3.5
(trained in 2021)

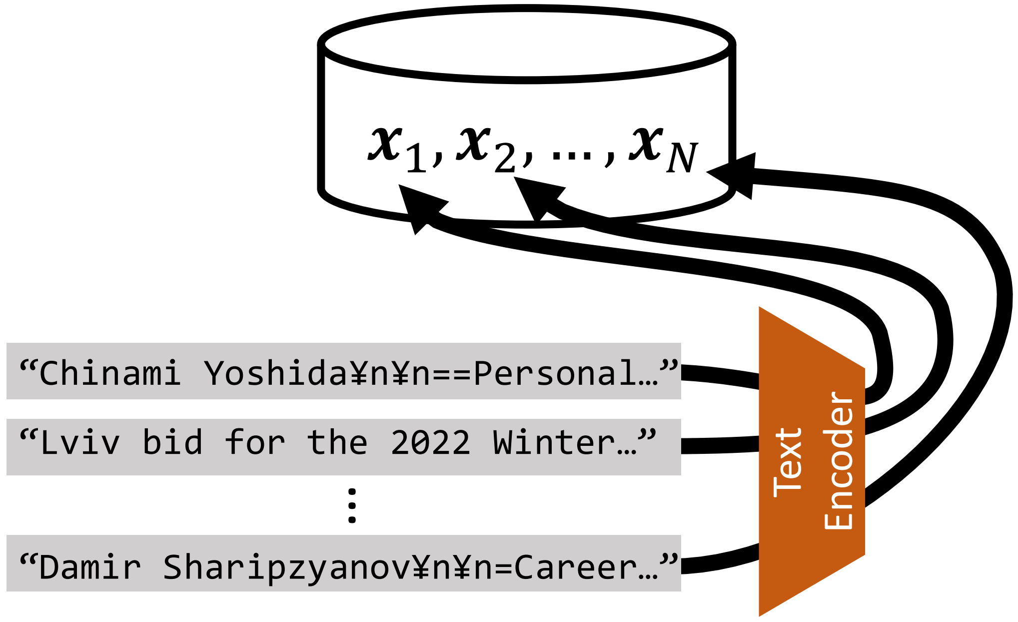


Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

Real-world use cases 2: LLM + embedding

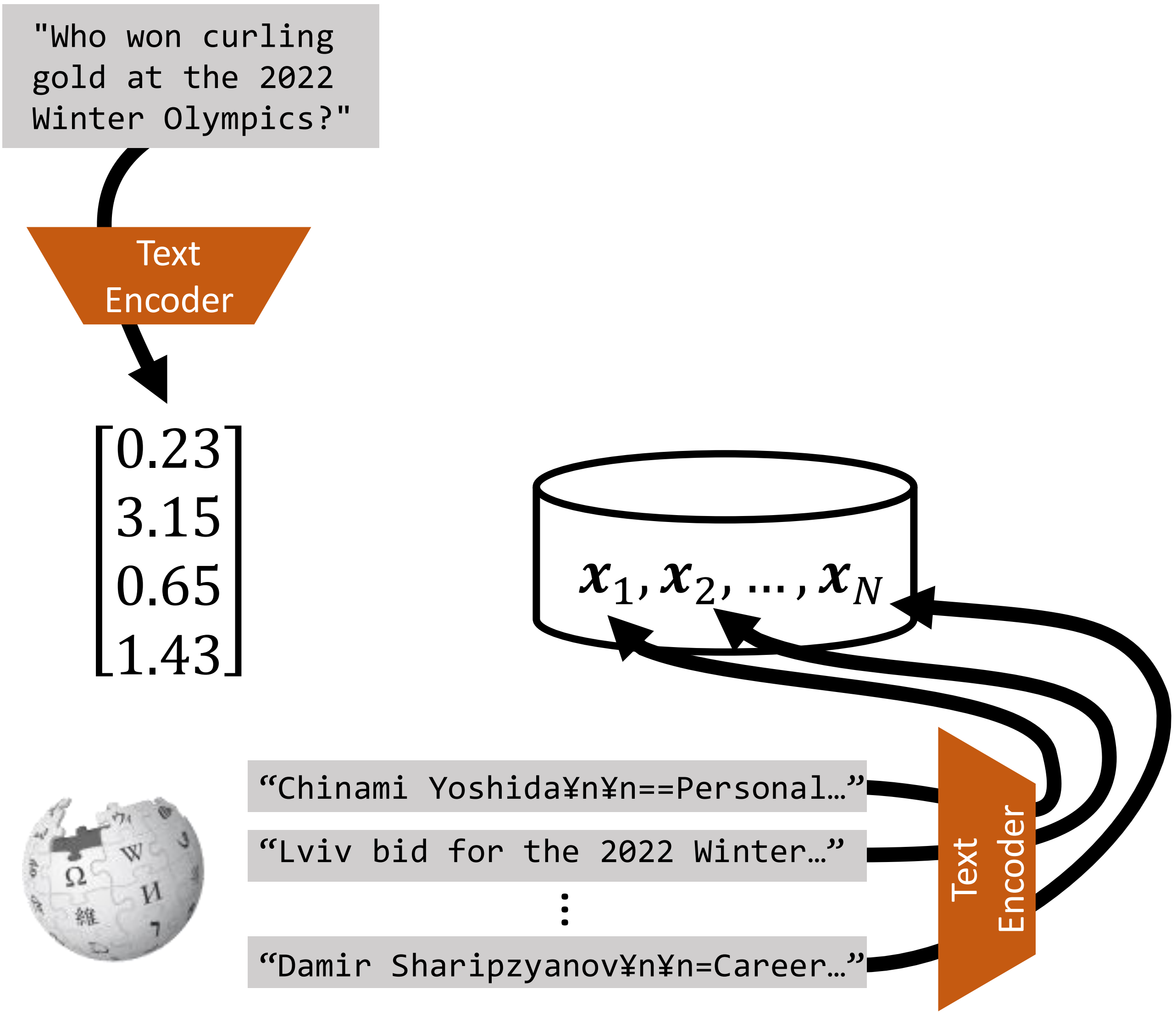
"Who won curling gold at the 2022 Winter Olympics?"

ChatGPT 3.5
(trained in 2021)



Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

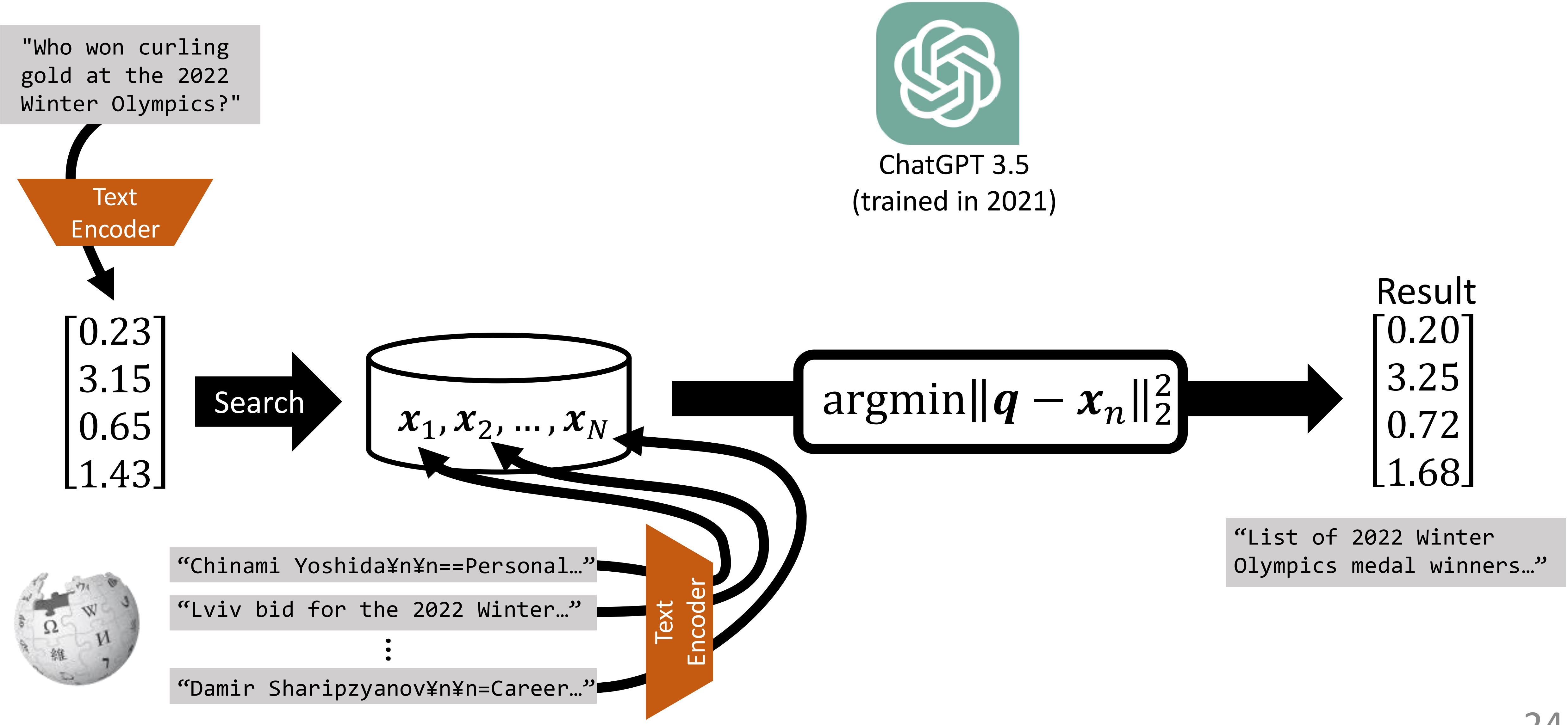
Real-world use cases 2: LLM + embedding



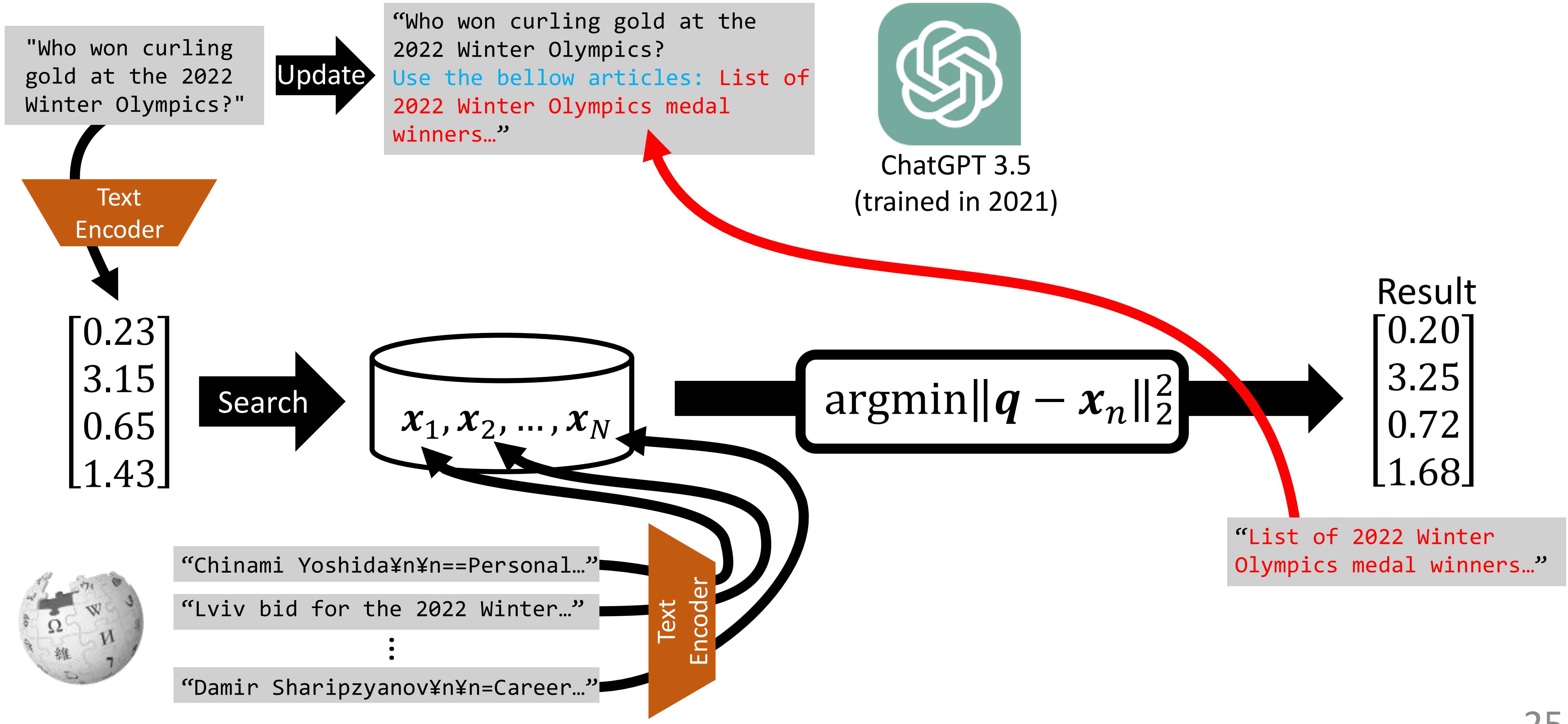
ChatGPT 3.5
(trained in 2021)

Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

Real-world use cases 2: LLM + embedding

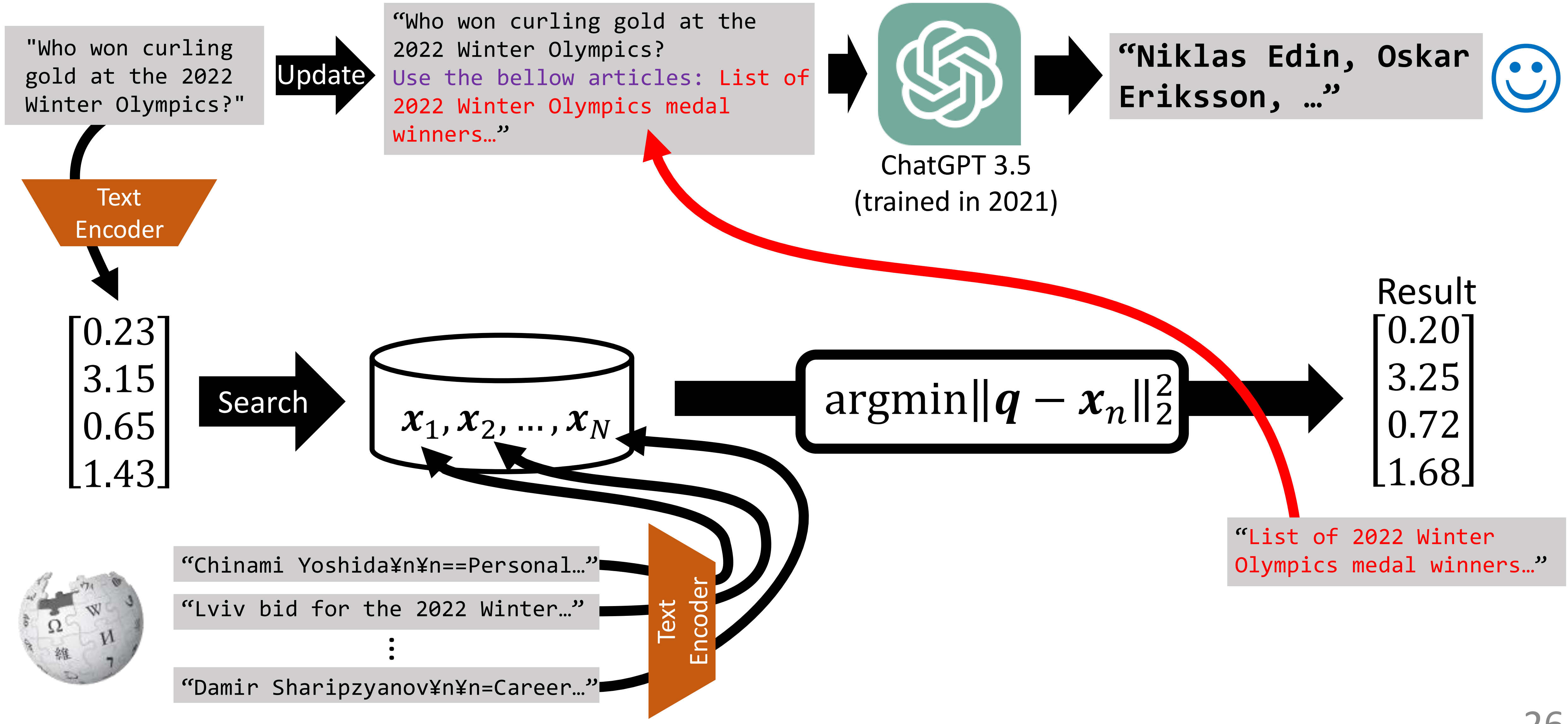


Real-world use cases 2: LLM + embedding



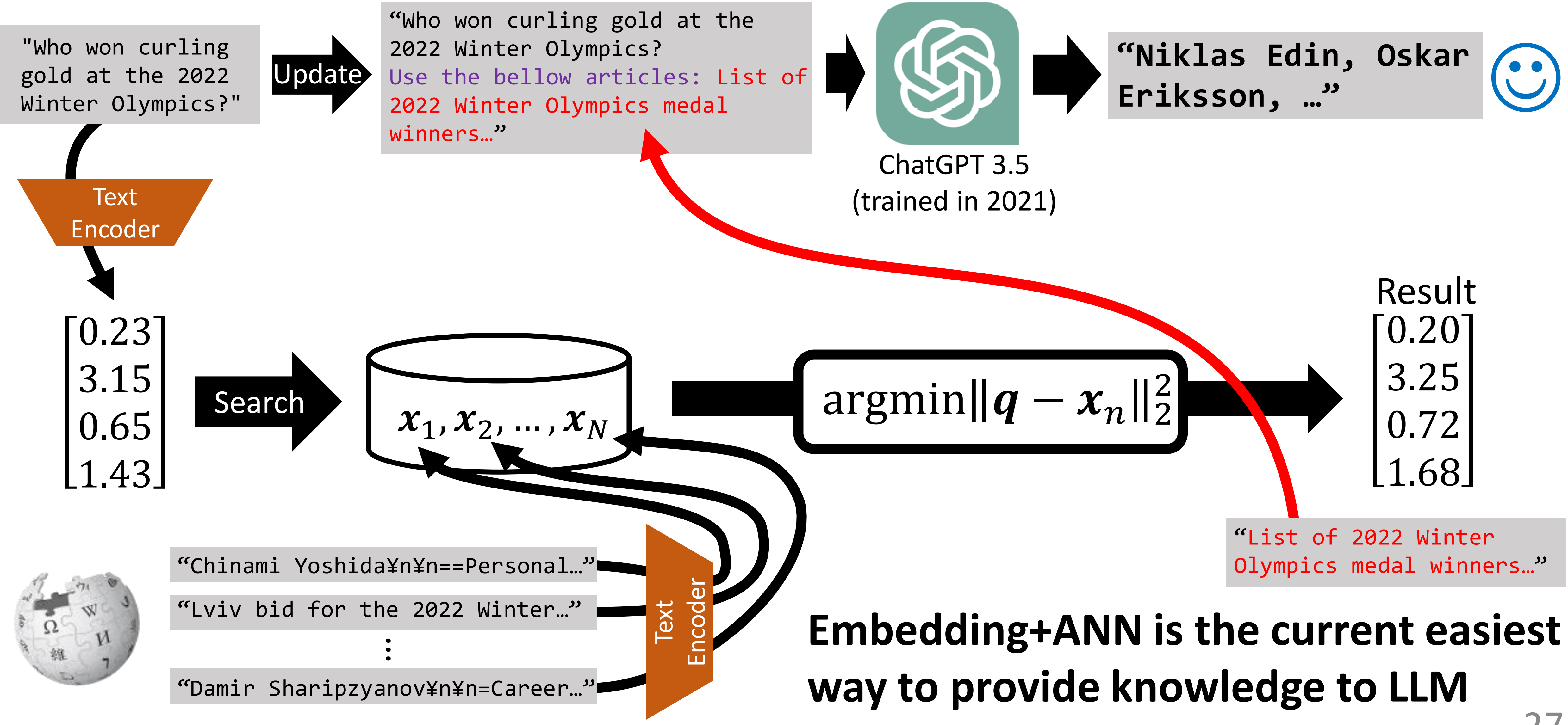
Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

Real-world use cases 2: LLM + embedding



Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

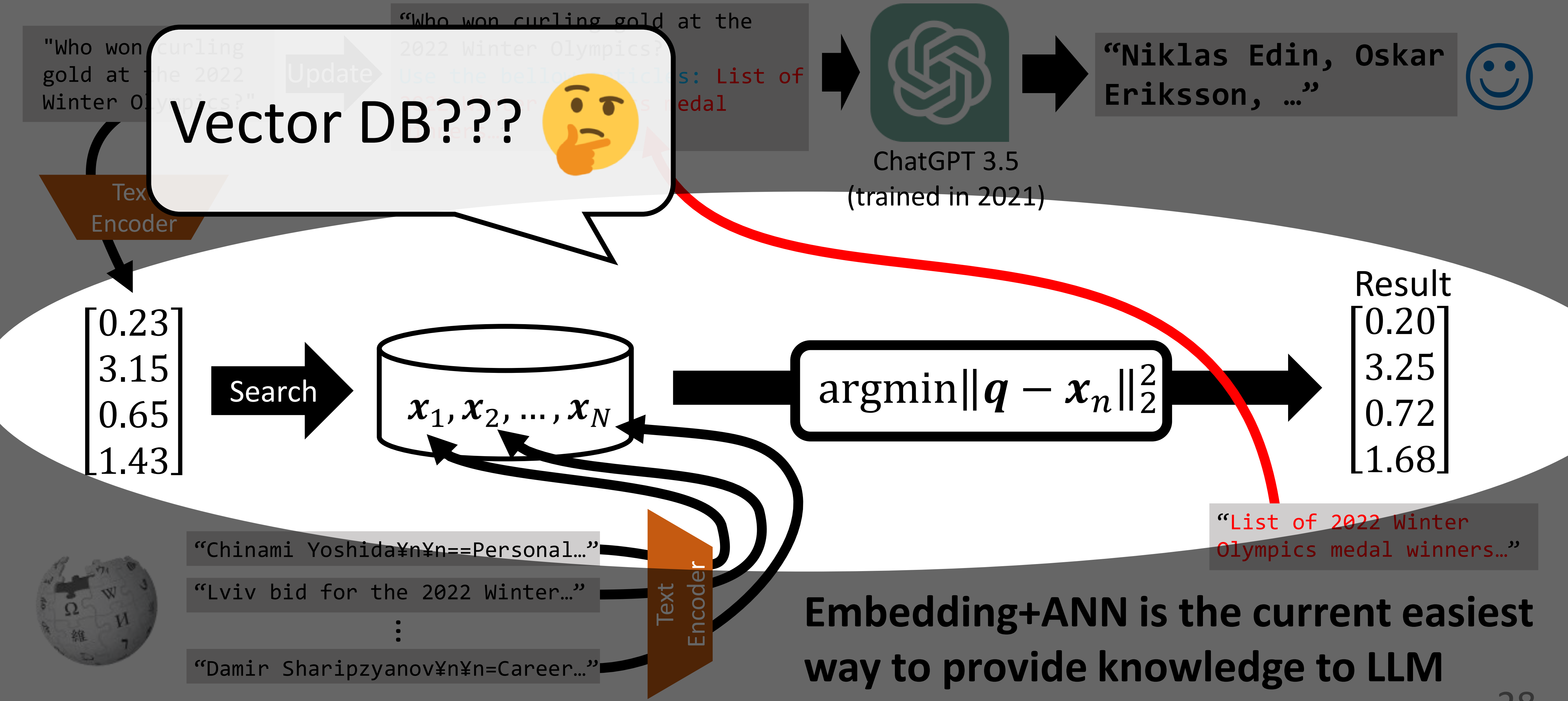
Real-world use cases 2: LLM + embedding



Embedding+ANN is the current easiest way to provide knowledge to LLM

Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

Real-world use cases 2: LLM + embedding



Embedding+ANN is the current easiest way to provide knowledge to LLM

Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb

https://em-content.zobj.net/thumbs/120/twitter/322/thinking-face_1f914.png

Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

Three levels of technology

Algorithm

- Scientific paper
- Math
- Often, by researchers

Product Quantization + Inverted Index (PQ, IVFPQ) [Jégou+, TPAMI 2011]

Hierarchical Navigable Small World (HNSW) [Malkov+, TPAMI 2019]

ScaNN (4-bit PQ) [Guo+, ICML 2020]

Library

- Implementations of algorithms
- Usually, a search function only
- By researchers, developers, etc

faiss

NMSLIB

hnswlib

ScaNN

Service (e.g., vector DB)

- Library + (handling metadata, serving, scaling, IO, CRUD, etc)
- Usually, by companies

Pinecone

Qdrant

Milvus

jina

Vald

Vertex AI Matching Engine

Weaviate

Three levels of technology

Algorithm
- Scientific paper
- Math
- Often, by researchers

Library
- Implementations of algorithms
- Usually, a search function only
- By researchers, developers, etc

Service (e.g., vector DB)
- Library + (handling metadata, serving, scaling, IO, CRUD, etc)
- Usually, by companies

Product Quantization + Inverted Index (PQ, IVFPQ) [Jégou+, TPAMI 2011]

Hierarchical Navigable Small World (HNSW) [Malkov+, TPAMI 2019]

ScaNN (4-bit PQ) [Guo+, ICML 2020]

faiss

hnswlib

Sc

Pinecone

Qdrant

One library may implement multiple algorithms

☹ "I benchmarked faiss"

☺ "I benchmarked PQ in faiss"

Three levels of technology

One algorithm may be implemented in multiple libraries

- Math
- Often, by re

- Usually, a search function only
- By researchers, developers, etc

Service (e.g., vector DB)
- Library + (handling metadata, serving, scaling, IO, CRUD, etc)
- Usually, by companies

Product Quantiza
Inverted Index (PQ, IV
[Jégou+, TPAMI 2011]

Hierarchical Navigable Small World (HNSW) [Malkov+, TPAMI 2019]

ScaNN (4-bit PQ) [Guo+, ICML 2020]

faiss

NMSLIB

hnswlib

ScaNN

Pinecone

Qdrant

Milvus

jina

Vald

Vertex AI Matching Engine

Weaviate

Three levels of technology

Algorithm
- Scientific paper
- Math
- Often, by researchers

Product Quantization + Inverted Index (PQ, IVFPQ) [Jégou+, TPAMI 2011]

Hierarchical Navigable Small World (HNSW) [Malkov+, TPAMI 2019]

ScaNN (4-bit PQ) [Guo+, ICML 2020]

Library
- Implementations of algorithms
- Usually, a search function only
- By researchers, developers, etc

faiss

ScaNN

Service (e.g., vector DB)
- Library + (handling metadata, serving, scaling, IO, CRUD, etc)
- Usually, by companies

Pinecone

Qdrant

Vald

Vertex AI Matching Engine

Weaviate

Often, one library = one algorithm

Three levels of technology

One service may use some libraries

Algorithm
- Scientific paper
- Math
- Often, by researchers

- Usually, a search function
- By researchers, developers, e

, vector DB)
ndling metadata,
rving, scaling, IO, CRUD, etc)
sually, by companies

Product Quantization + Inverted Index (PQ, IVFPQ) [Jégou+, TPAMI 2011]

Hierarchical Navigable Small World (HNSW) [Malkov+, TPAMI 2019]

… or re-implement algorithms from scratch (e.g., by Go)

faiss

NMSLIB

hnswlib

ScaNN

Pinecone

Qdrant

Milvus

jina

Vald

Vertex AI Matching Engine

Weaviate

Three levels of technology

Algorithm
- Scientific paper
- Math
- Often, by researchers

Product Quantization + Inverted Index (PQ, IVFPQ) [Jégou+, TPAMI 2011]

Hierarchical Navigable Small World (HNSW) [Malkov+, TPAMI 2019]

ScaNN (4-bit PQ) [Guo+, ICML 2020]

Library
- Implementations of algorithms
- Usually, a search function only
- By researchers, developers, etc

faiss

hnswlib

ScaNN

Service (e.g., vector DB)
- Library + (handling metadata, serving, scaling, IO, CRUD, etc)
- Usually, by companies

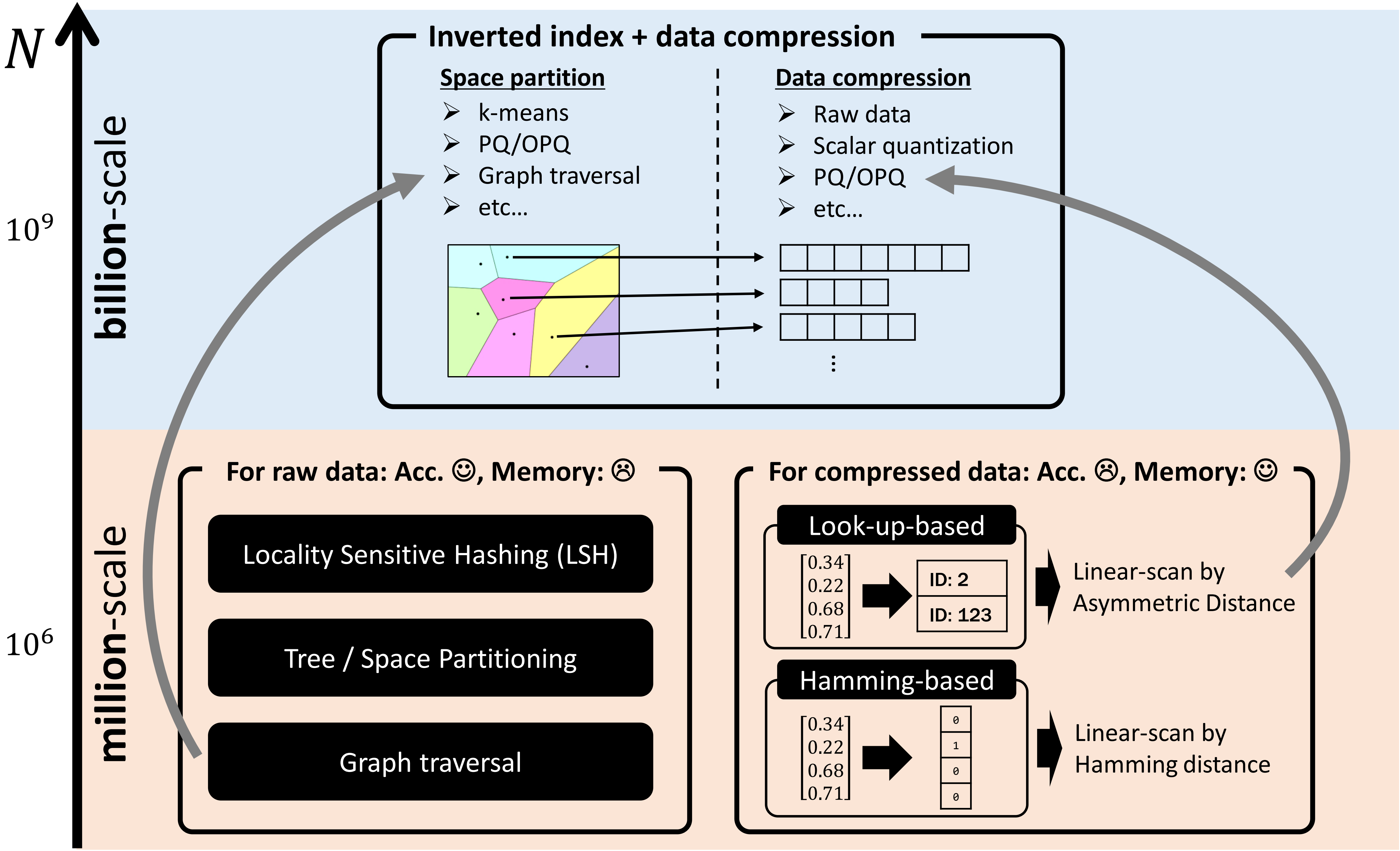
Pinecone

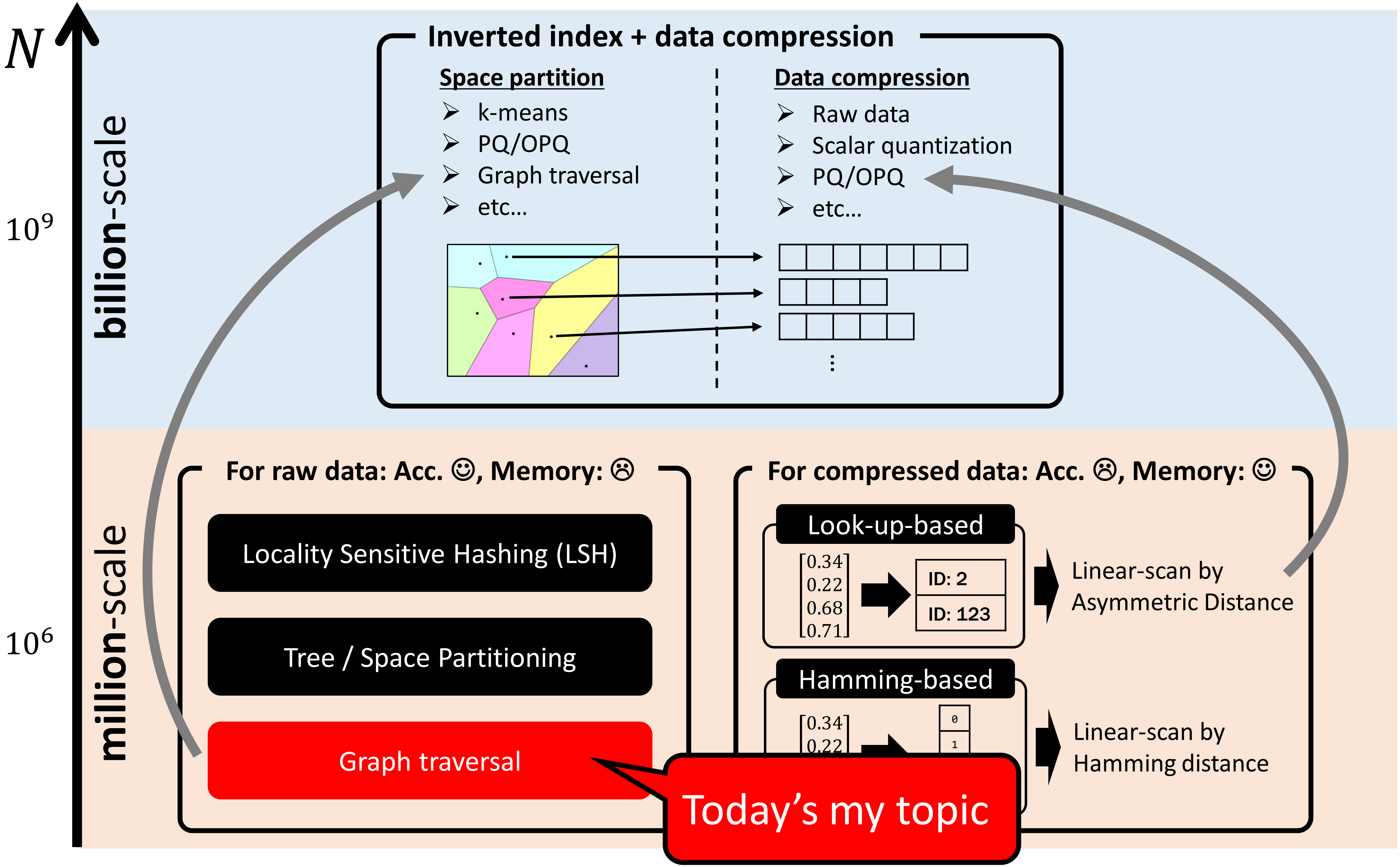
Vald

Vertex AI Matching Engine

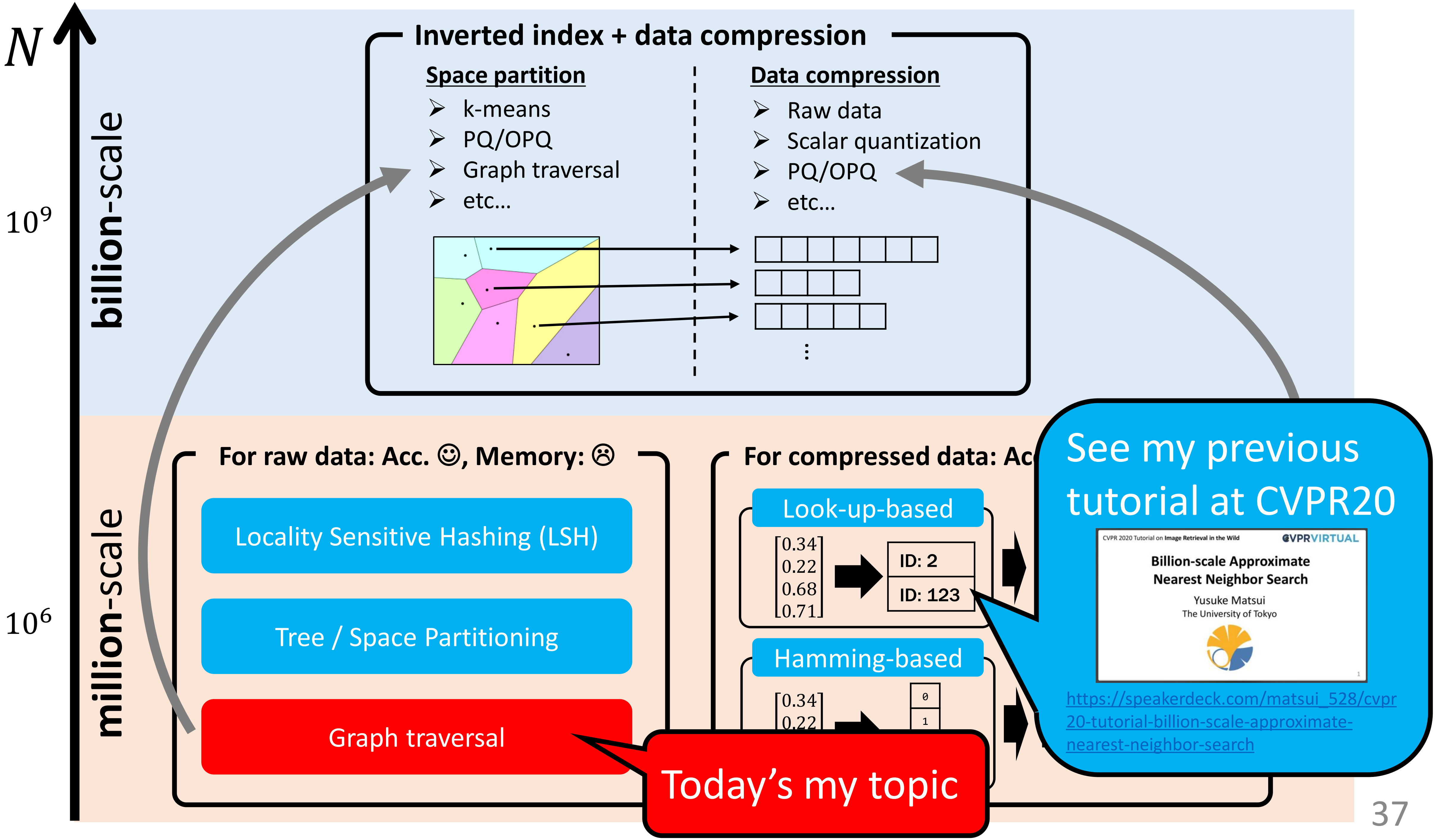
Weaviate

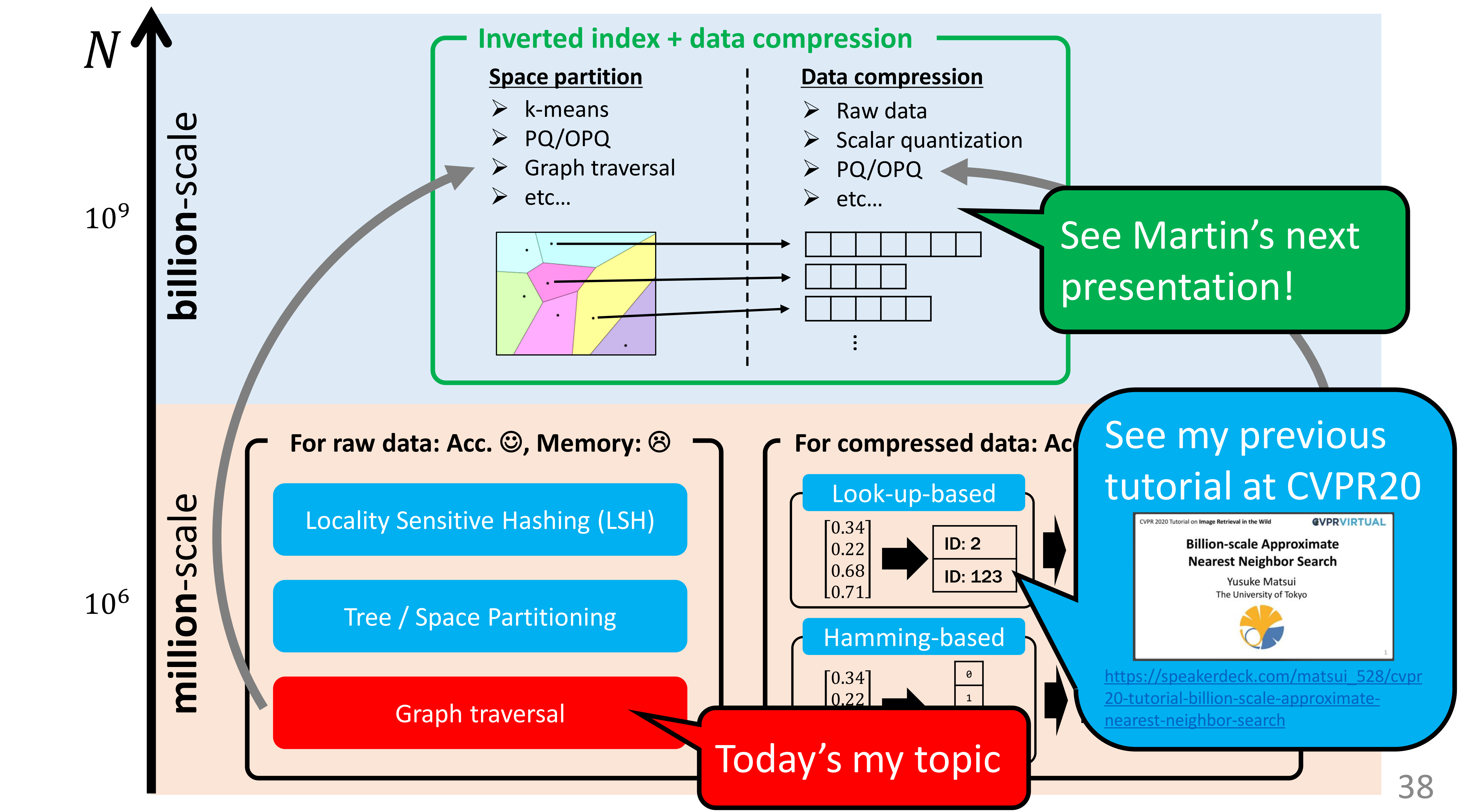
This talk mainly focuses algorithms


N
10^9
10^6
billion-scale
million-scale
Inverted index + data compression
Space partition
k-means
PQ/OPQ
Graph traversal
etc…
Data compression
Raw data
Scalar quantization
PQ/OPQ
etc…
For raw data: Acc. ☺, Memory: ☹
Locality Sensitive Hashing (LSH)
Tree / Space Partitioning
Graph traversal
For compressed data: Acc. ☹, Memory: ☺
Look-up-based
0.34
0.22
0.68
0.71
ID: 2
ID: 123
Linear-scan by
Asymmetric Distance
Hamming-based
0.34
0.22
0.68
0.71
0
1
0
0
Linear-scan by
Hamming distance



N
10^9
10^6
billion-scale
million-scale
Inverted index + data compression
Space partition
k-means
PQ/OPQ
Graph traversal
etc...
Data compression
Raw data
Scalar quantization
PQ/OPQ
etc...
For raw data: Acc. ☺, Memory: ☹
Locality Sensitive Hashing (LSH)
Tree / Space Partitioning
Graph traversal
For compressed data: Acc. ☹, Memory: ☺
Look-up-based
0.34
0.22
0.68
0.71
ID: 2
ID: 123
Linear-scan by Asymmetric Distance
Hamming-based
0.34
0.22
Linear-scan by Hamming distance
Today's my topic


N
10^9
10^6
billion-scale
million-scale
Inverted index + data compression
Space partition
k-means
PQ/OPQ
Graph traversal
etc…
Data compression
Raw data
Scalar quantization
PQ/OPQ
etc…
For raw data: Acc. ☺, Memory: ☹
Locality Sensitive Hashing (LSH)
Tree / Space Partitioning
Graph traversal
For compressed data: Ac
Look-up-based
0.34
0.22
0.68
0.71
ID: 2
ID: 123
Hamming-based
0.34
0.22
0
1
Today's my topic
See my previous tutorial at CVPR20
CVPR 2020 Tutorial on Image Retrieval in the Wild
CVPRVIRTUAL
Billion-scale Approximate Nearest Neighbor Search
Yusuke Matsui
The University of Tokyo
https://speakerdeck.com/matsui_528/cvpr20-tutorial-billion-scale-approximate-nearest-neighbor-search

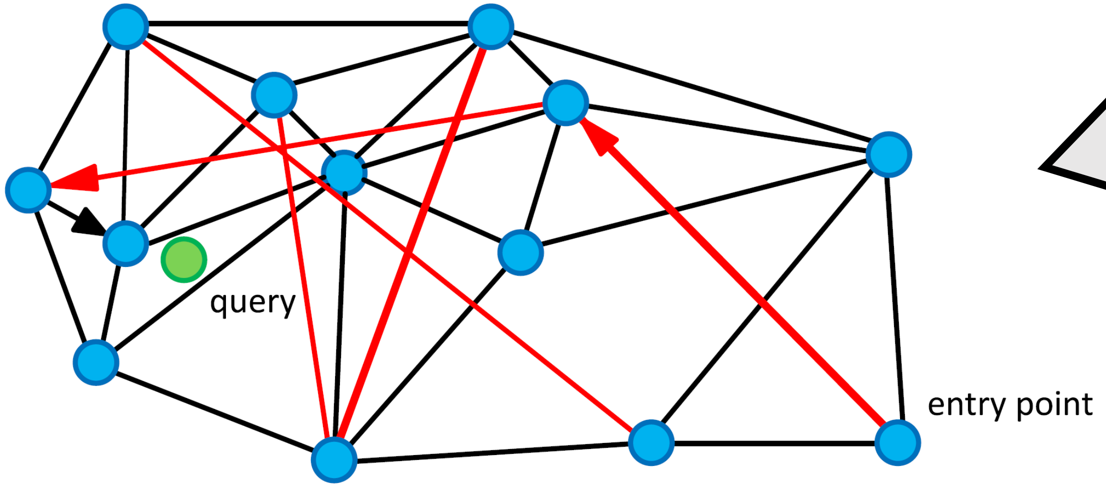


N
10^9
10^6
billion-scale
million-scale
Inverted index + data compression
Space partition
k-means
PQ/OPQ
Graph traversal
etc…
Data compression
Raw data
Scalar quantization
PQ/OPQ
etc…
See Martin's next presentation!
For raw data: Acc. ☺, Memory: ☹
Locality Sensitive Hashing (LSH)
Tree / Space Partitioning
Graph traversal
For compressed data: Ac
Look-up-based
0.34
0.22
0.68
0.71
ID: 2
ID: 123
Hamming-based
0.34
0.22
0
1
Today's my topic
See my previous tutorial at CVPR20
CVPR 2020 Tutorial on Image Retrieval in the Wild
CVPR VIRTUAL
Billion-scale Approximate Nearest Neighbor Search
Yusuke Matsui
The University of Tokyo
https://speakerdeck.com/matsui_528/cvpr20-tutorial-billion-scale-approximate-nearest-neighbor-search

- Background
- Graph-based search
  - Basic (construction and search)
  - Observation
  - Properties
- Representative works
  - HNSW, NSG, NGT, Vamana
- Discussion

Graph search

- ➢ De facto standard if all data can be loaded on memory
- ➢ Fast and accurate for real-world data
- ➢ Important for billion-scale situation as well
  - ✓ Graph-search is a building block for billion-scale systems



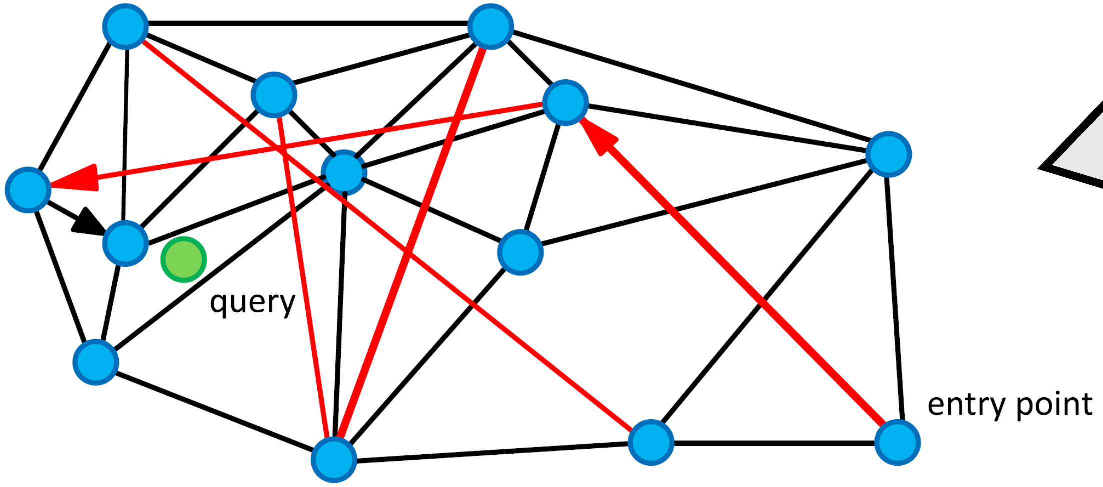
- ➢ Traverse graph towards the query
- ➢ Seems intuitive, but not so much easy to understand
- ➢ Review the algorithm carefully

Images are from [Malkov+, Information Systems, 2013]

Graph search

- De facto standard if all data can be loaded on memory
- Fast and accurate for real-world data
- Important for billion-scale situation as well
  - ✓ Graph-search is a building block for billion-scale systems

The purpose of this tutorial is to make graph search not a black box

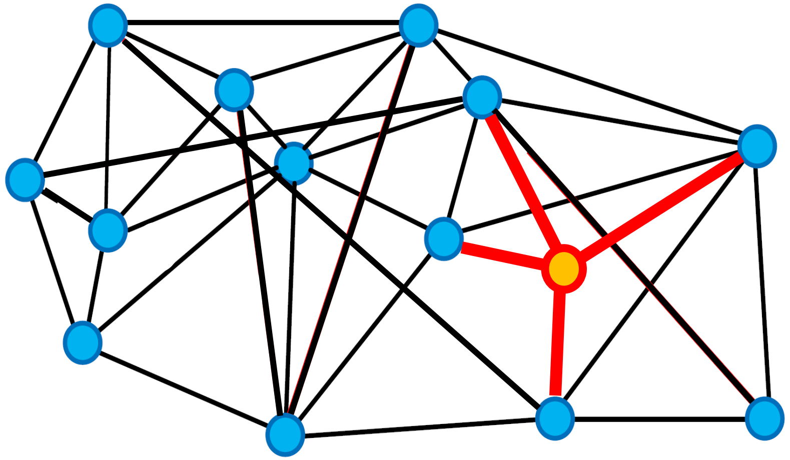


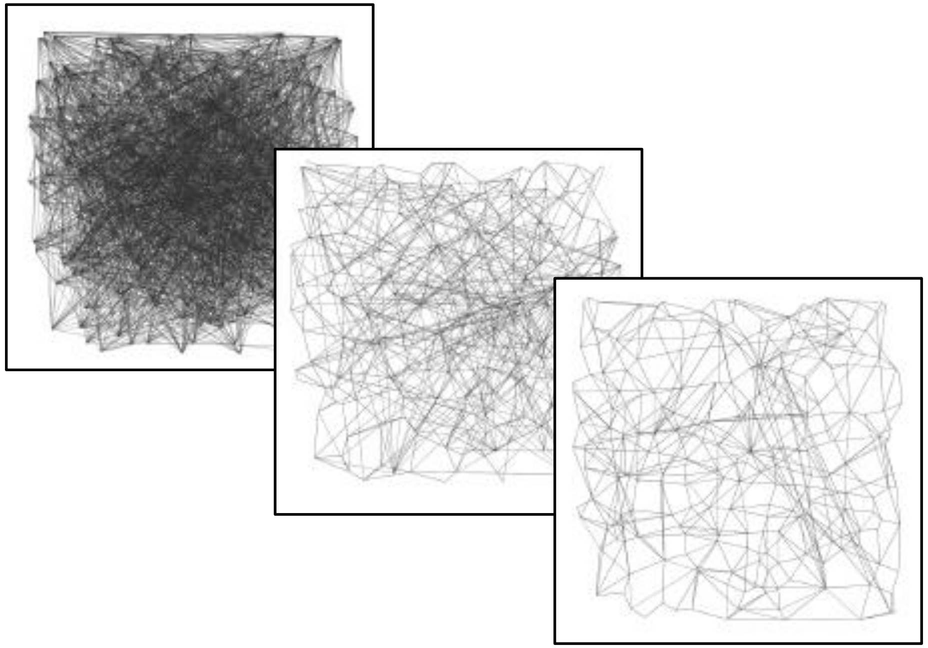
- Traverse graph towards the query
- Seems intuitive, but not so much easy to understand
- Review the algorithm carefully

Images are from [Malkov+, Information Systems, 2013]

Construction

Images are from [Malkov+, Information Systems, 2013] and [Subramanya+, NerulPS 2019]





Increment approach

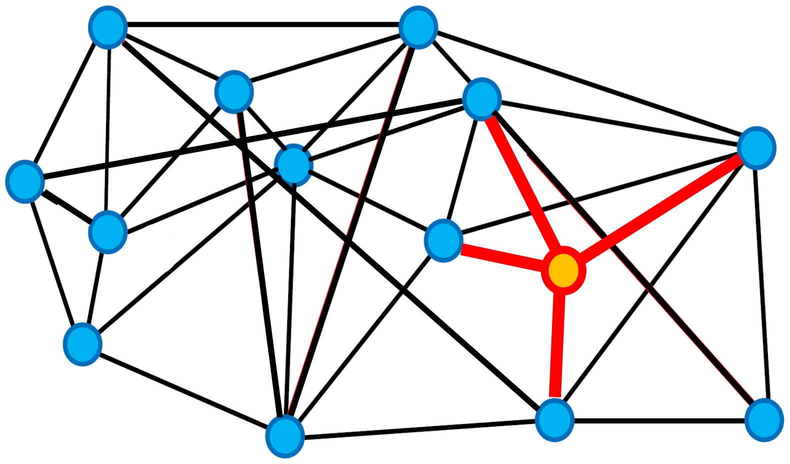
- ➢ Add a new item to the current graph incrementally

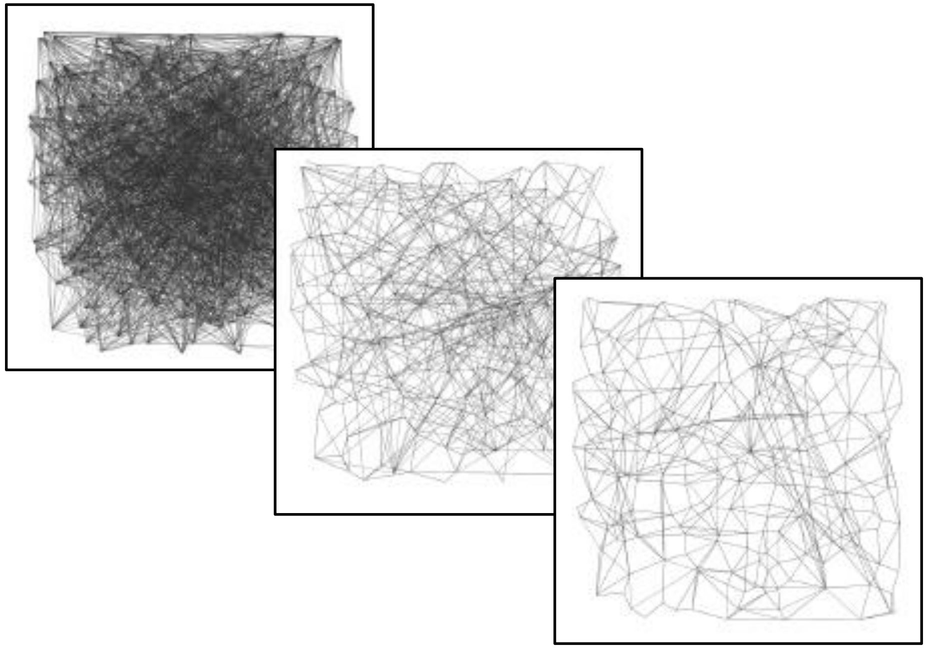
Refinement approach

- ➢ Iteratively refine an initial graph

Construction

Images are from [Malkov+, Information Systems, 2013] and [Subramanya+, NeruIPS 2019]





Increment approach

- Add a new item to the current graph incrementally

Refinement approach

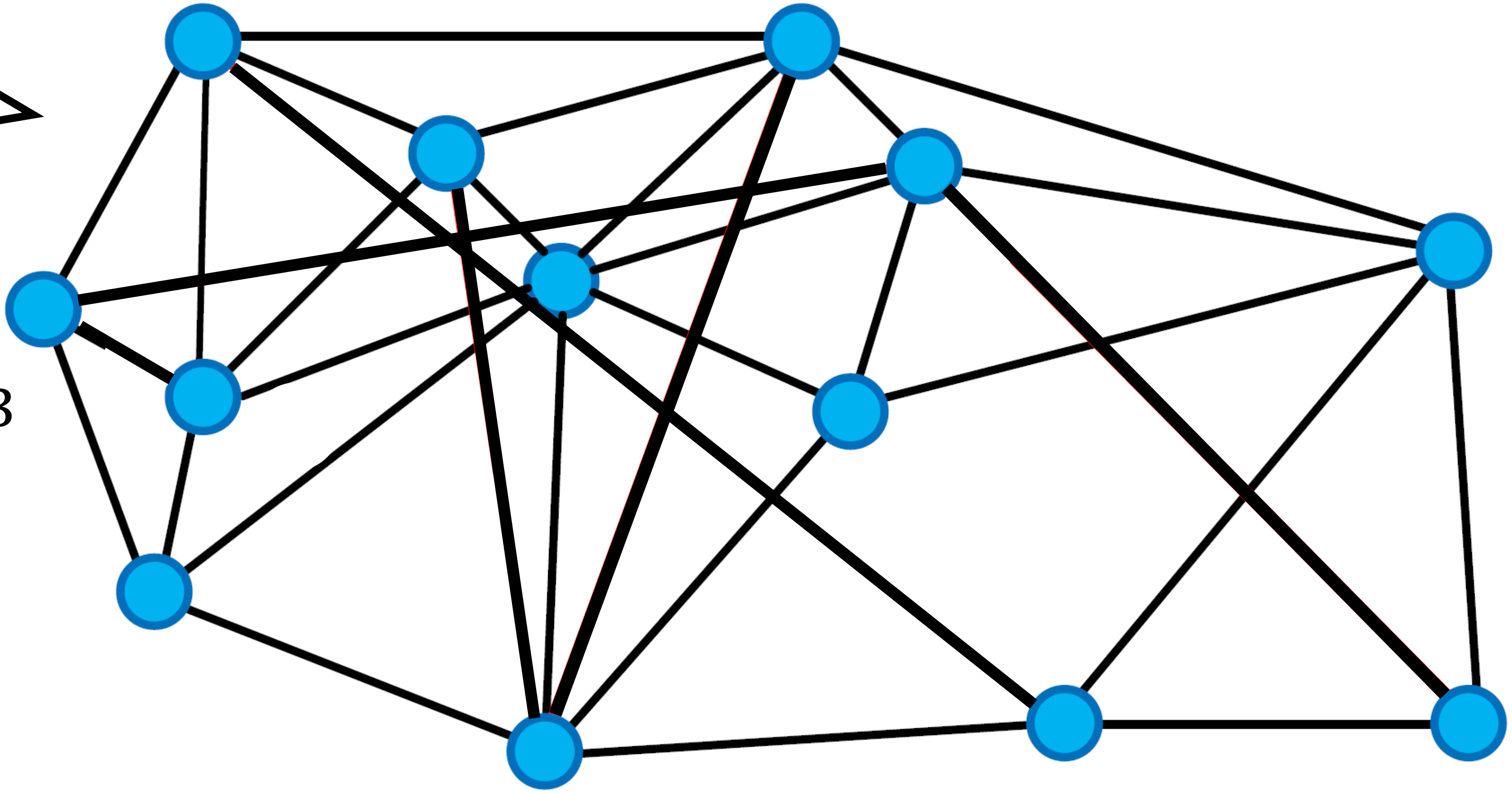
- Iteratively refine an initial graph

Construction: incremental approach

Images are from [Malkov+, Information Systems, 2013]

Graph of $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{90}$

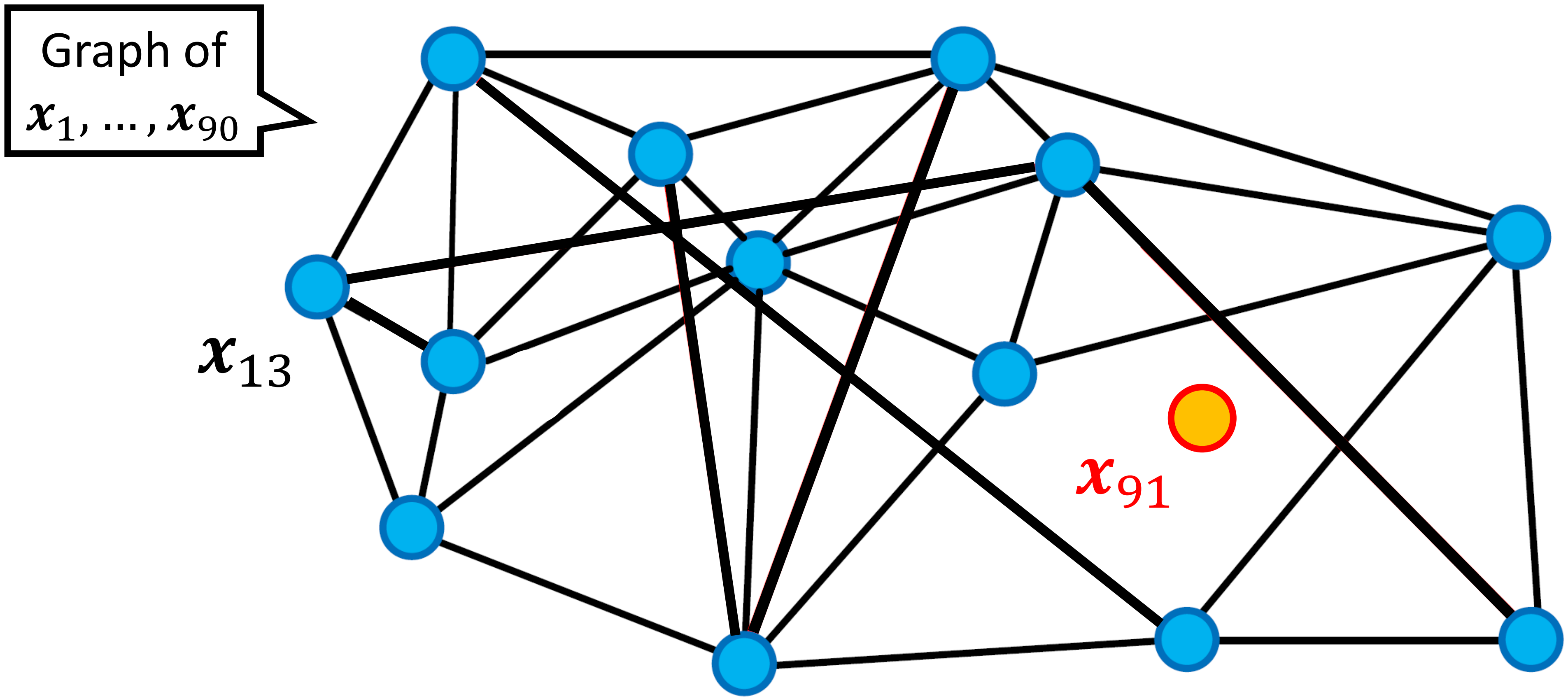
$\boldsymbol{x}_{13}$



➢Each node is a database vector

Construction: incremental approach

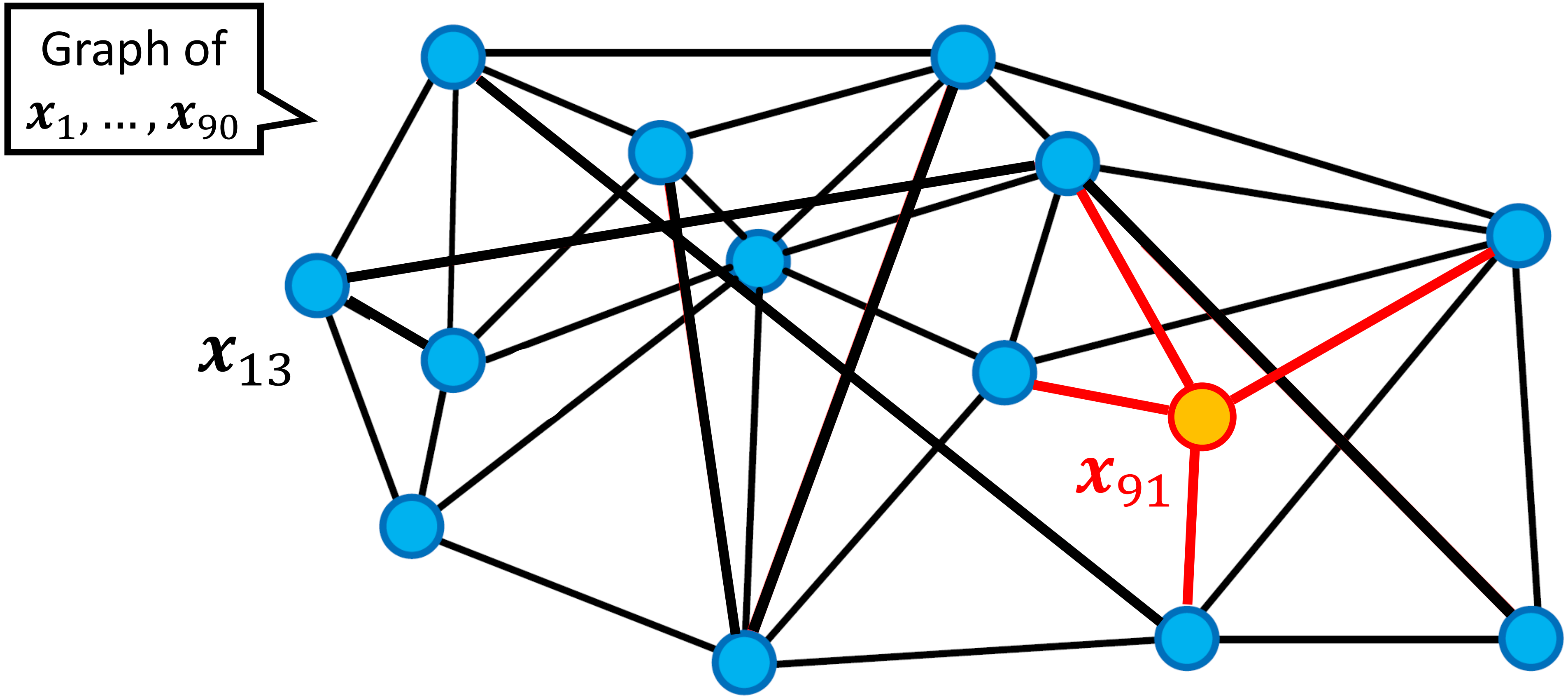
Images are from [Malkov+, Information Systems, 2013]



- Each node is a database vector
- Given a new database vector,

Construction: incremental approach

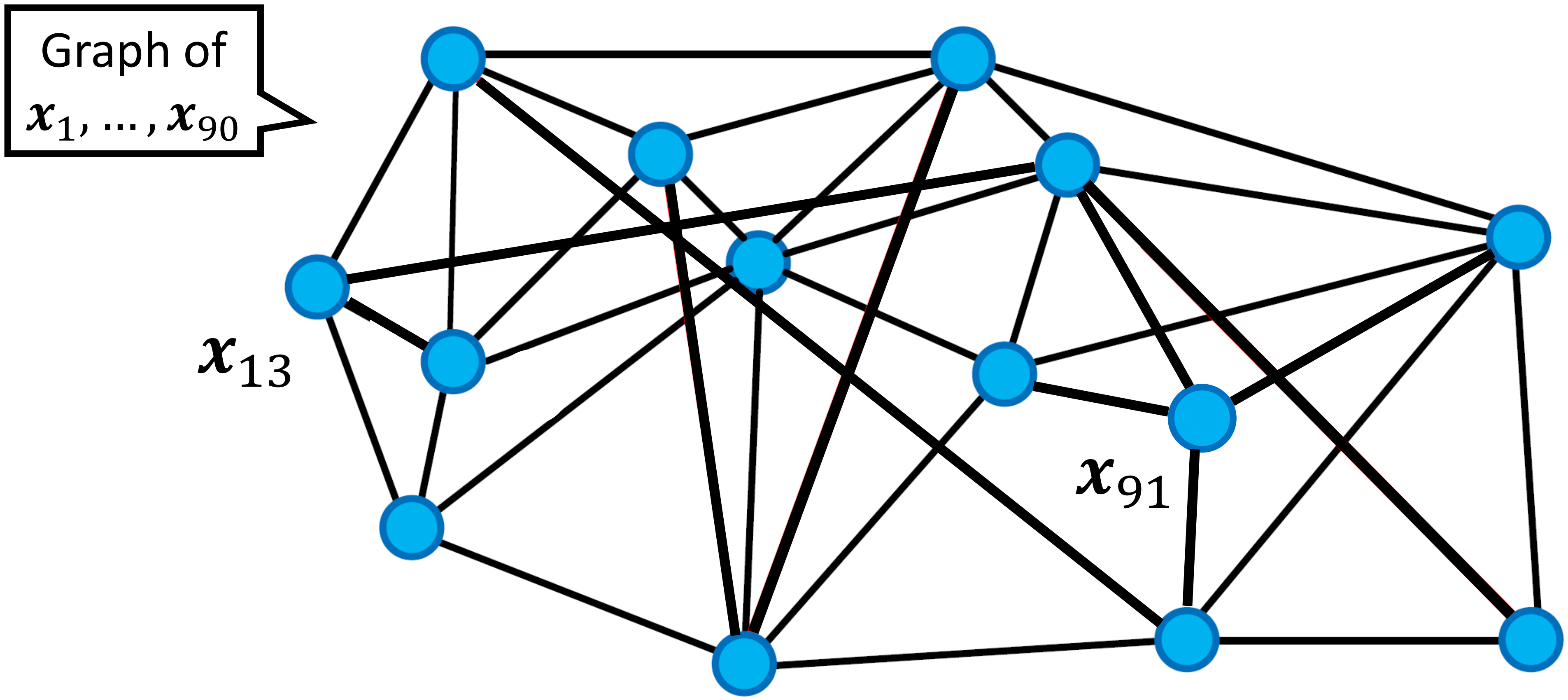
Images are from [Malkov+, Information Systems, 2013]



- Each node is a database vector
- Given a new database vector, create new edges to neighbors

Construction: incremental approach

Images are from [Malkov+, Information Systems, 2013]

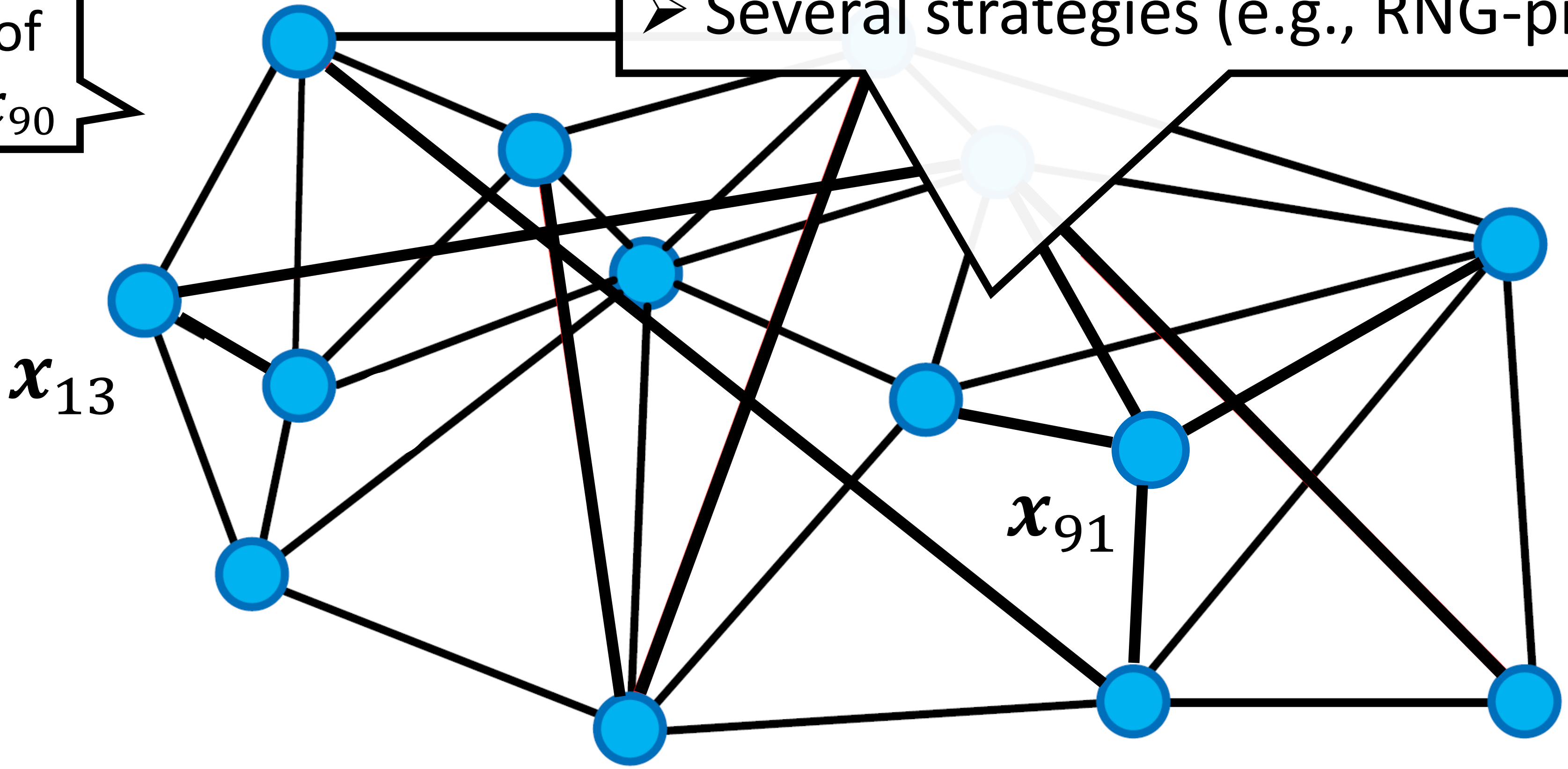


- ➢Each node is a database vector
- ➢Given a new database vector, create new edges to neighbors

Construction: incremental approach

- ➢ Prune edges if some node have too many edges
- ➢ Several strategies (e.g., RNG-pruning)

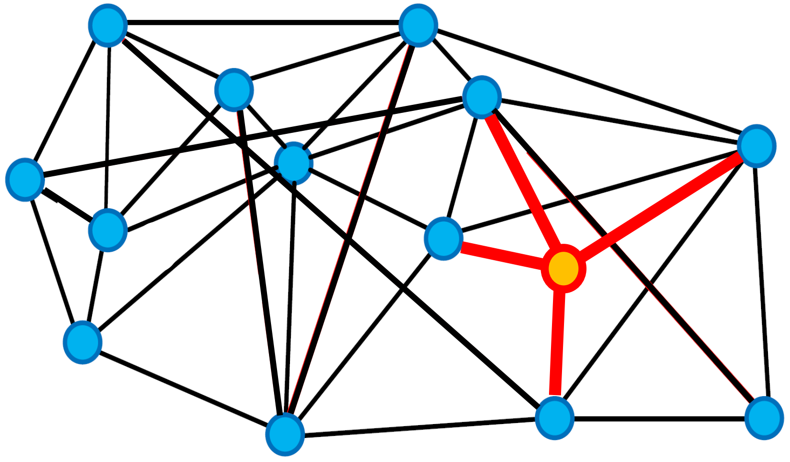
Graph of $\boldsymbol{x}_1, \dots, \boldsymbol{x}_{90}$

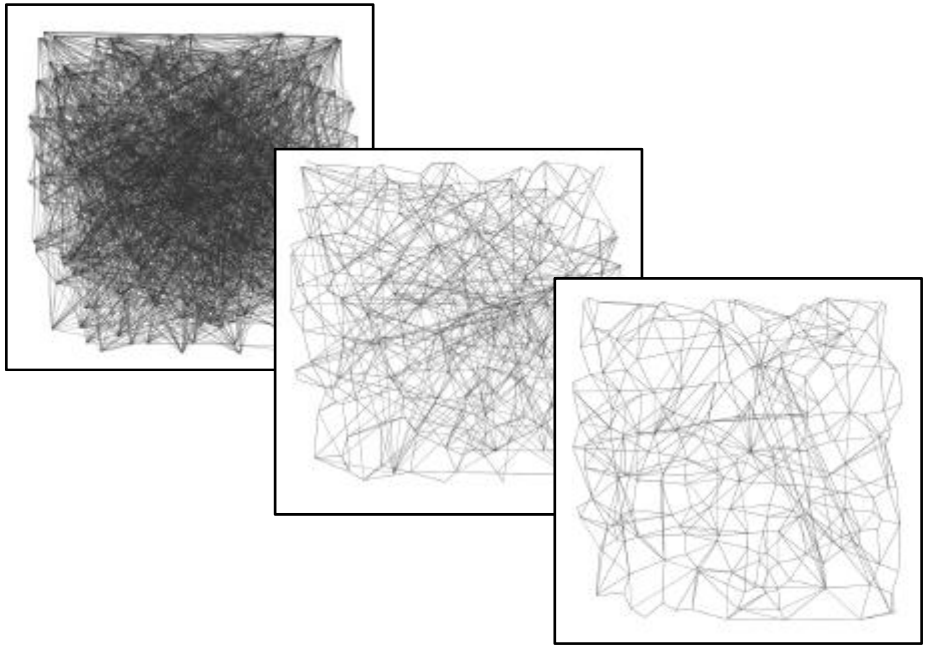


- ➢ Each node is a database vector
- ➢ Given a new database vector, create new edges to neighbors

Construction

Images are from [Malkov+, Information Systems, 2013] and [Subramanya+, NeruIPS 2019]





Increment approach

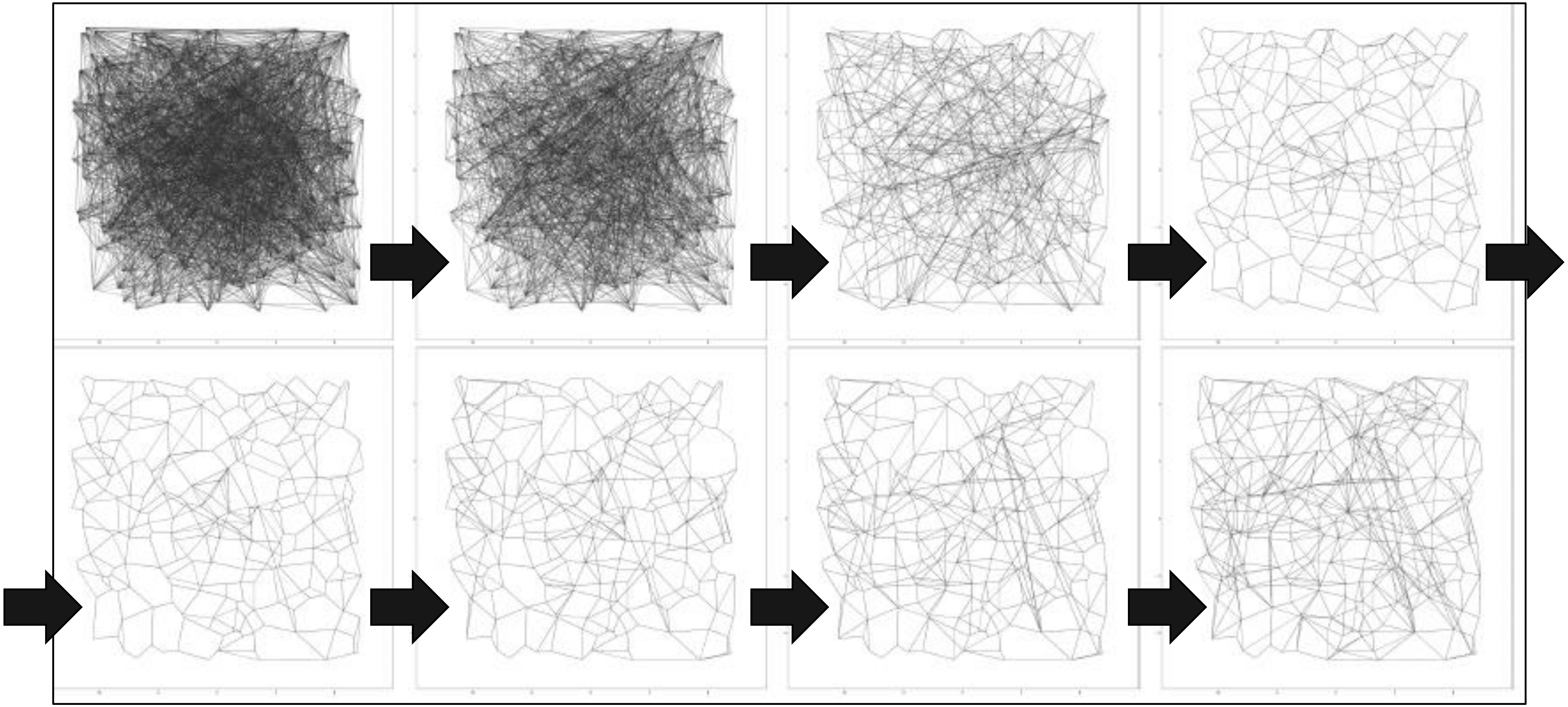
- Add a new item to the current graph incrementally

Refinement approach

- Iteratively refine an initial graph

Construction: refinement approach

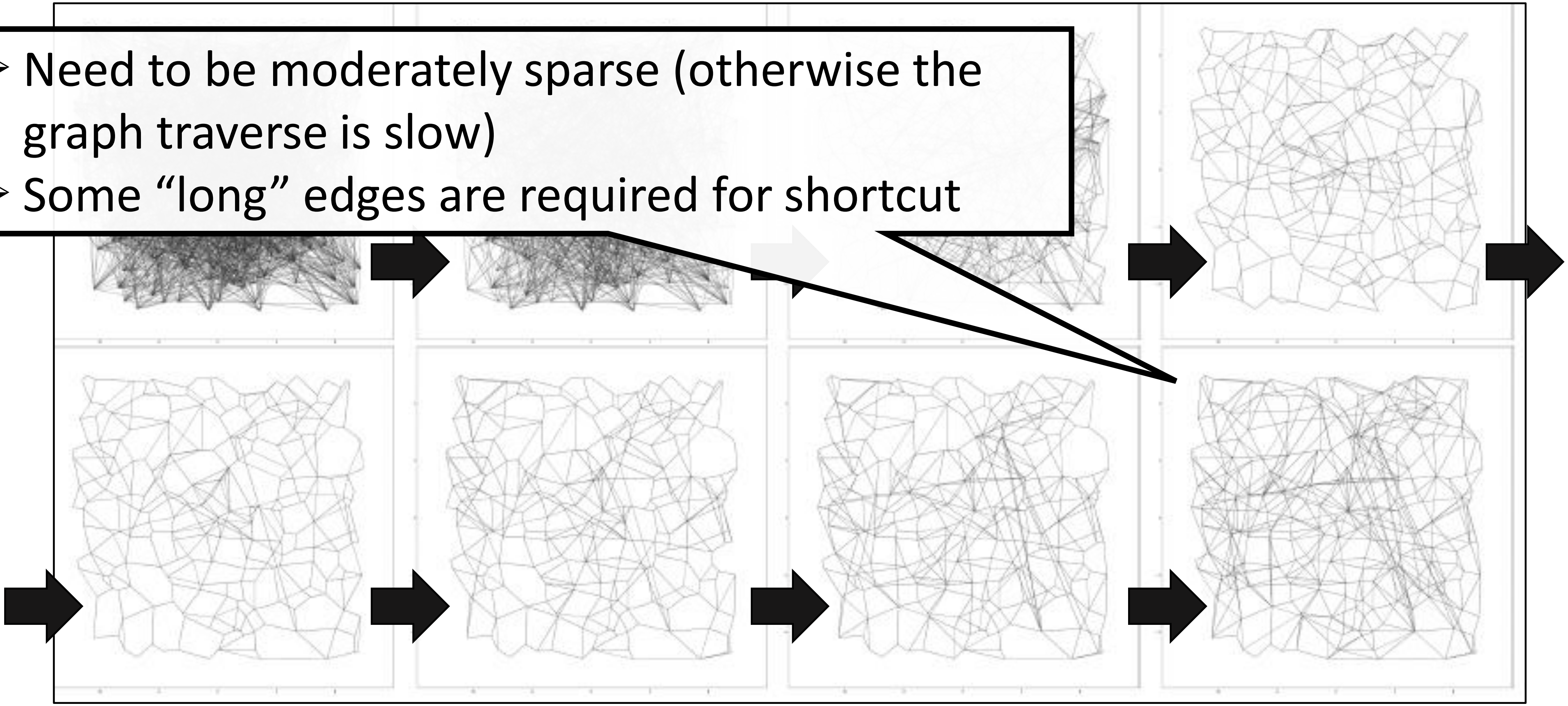
Images are from [Subramanya+, NeruIPS 2019]



- ➢ Create an initial graph (e.g., random graph or approx. kNN graph)
- ➢ Refine it iteratively (pruning/adding edges)

Construction: refinement approach

Images are from [Subramanya+, NeruIPS 2019]

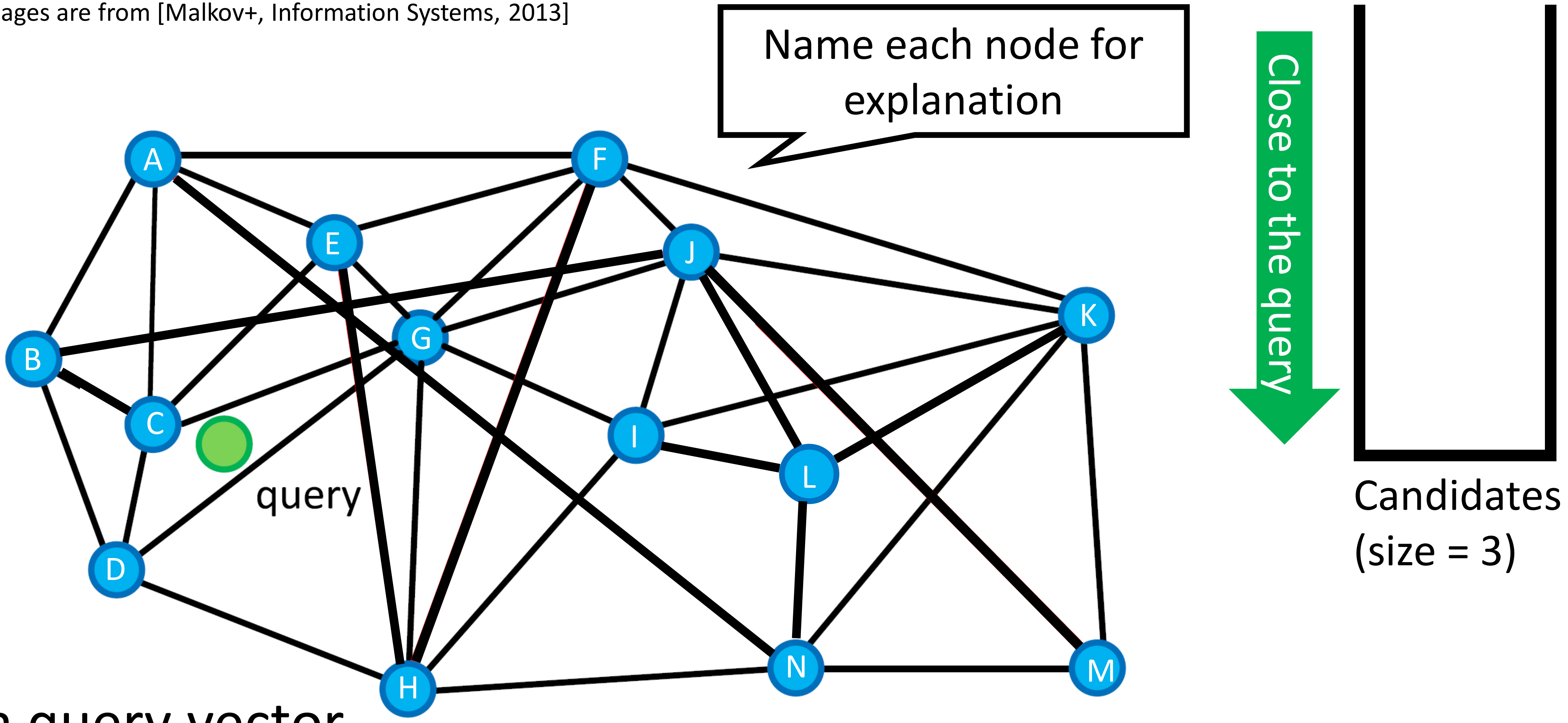


- Need to be moderately sparse (otherwise the graph traverse is slow)
- Some “long” edges are required for shortcut

- Create an initial graph (e.g., random graph or approx. kNN graph)
- Refine it iteratively (pruning/adding edges)

Search

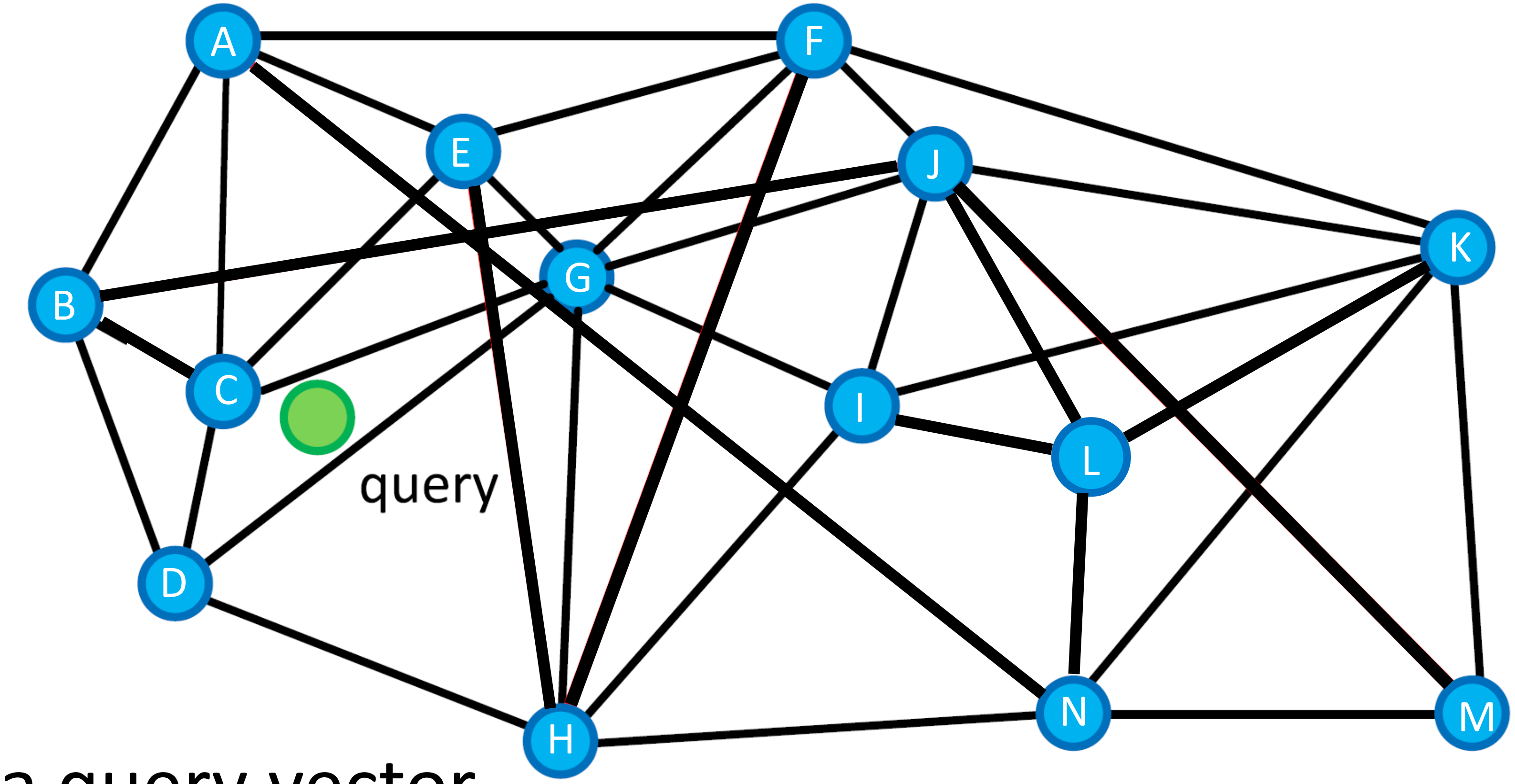
Images are from [Malkov+, Information Systems, 2013]

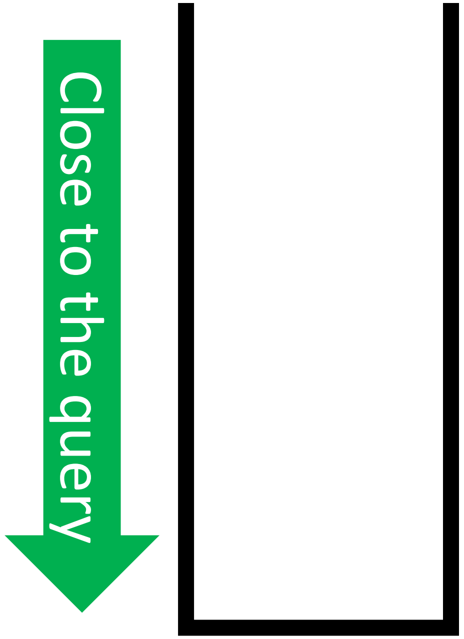


- ➢ Given a query vector

Search

Images are from [Malkov+, Information Systems, 2013]



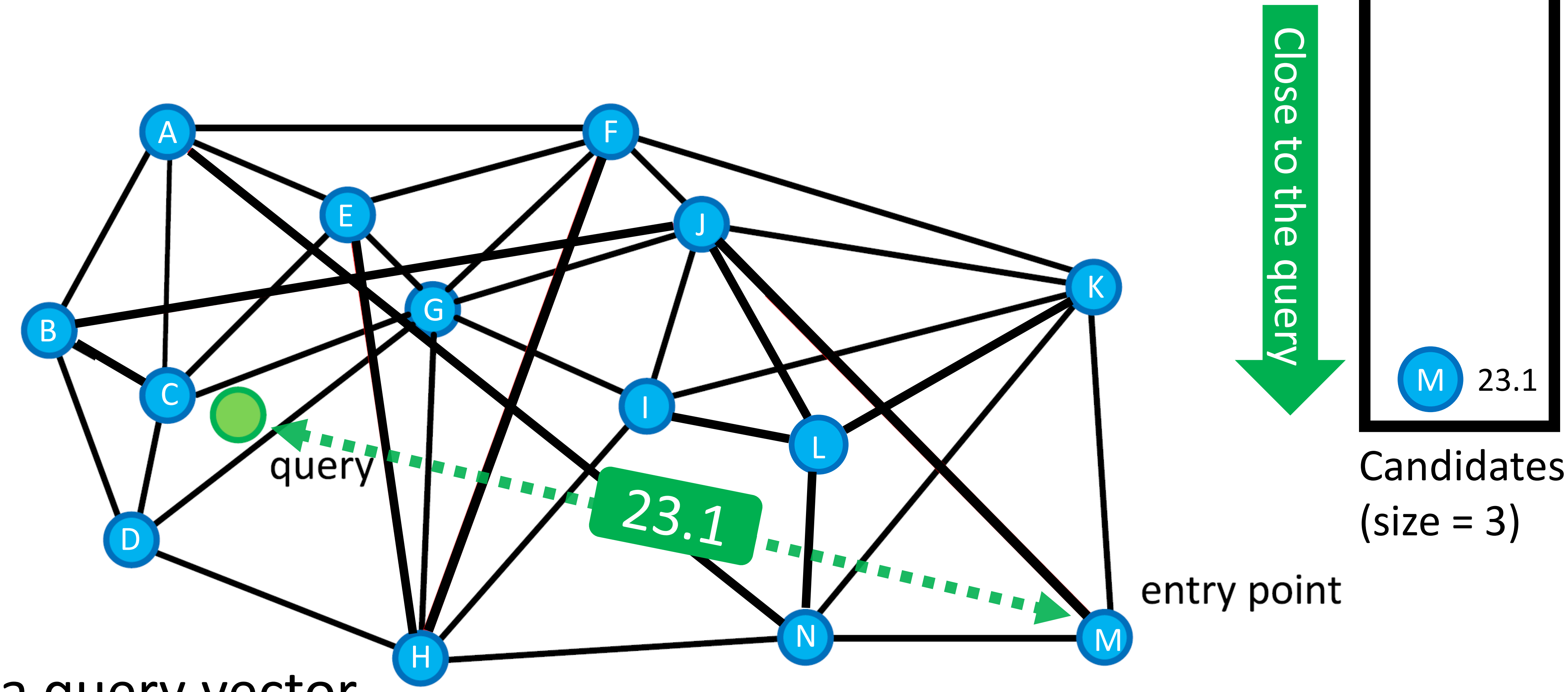


Candidates (size = 3)

- Given a query vector
- Start from an entry point (e.g., M)

Search

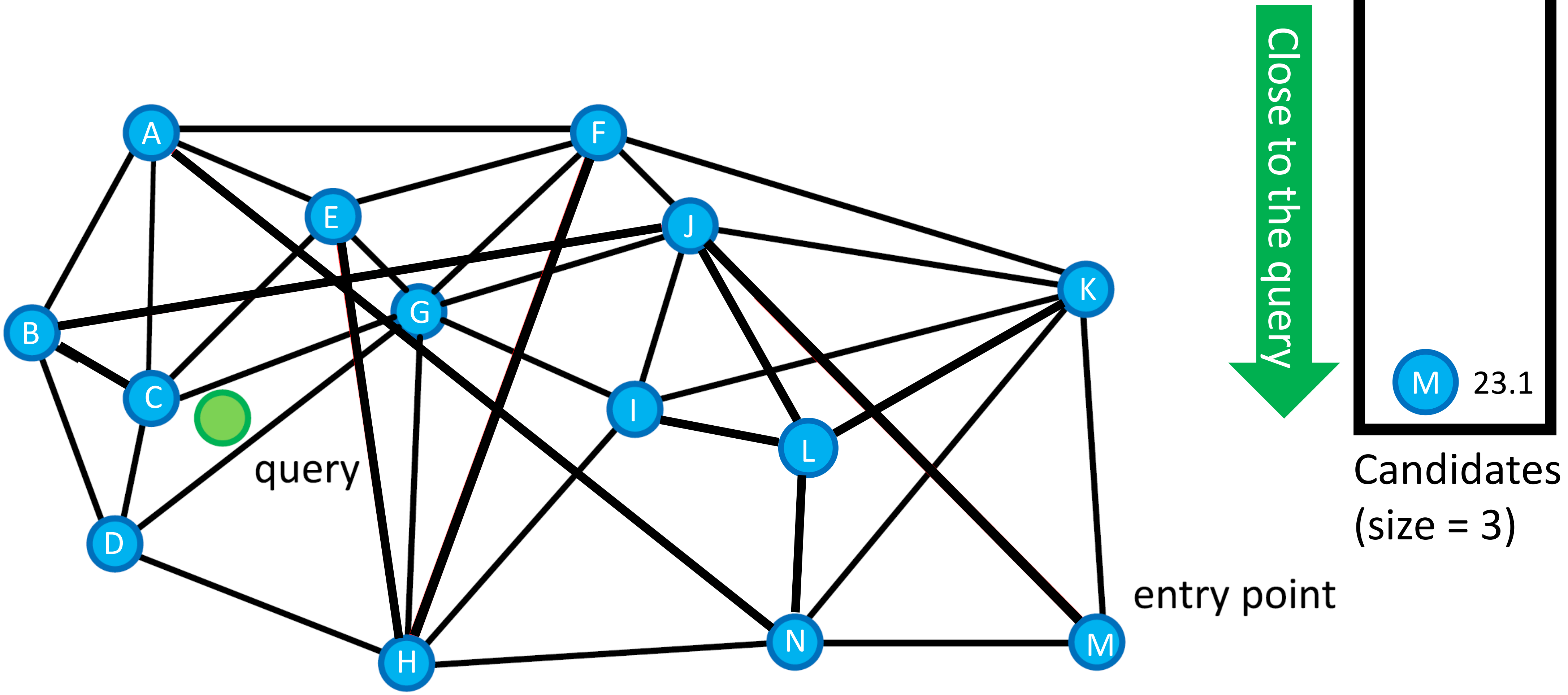
Images are from [Malkov+, Information Systems, 2013]



- Given a query vector
- Start from an entry point (e.g., M). Record the distance to q.

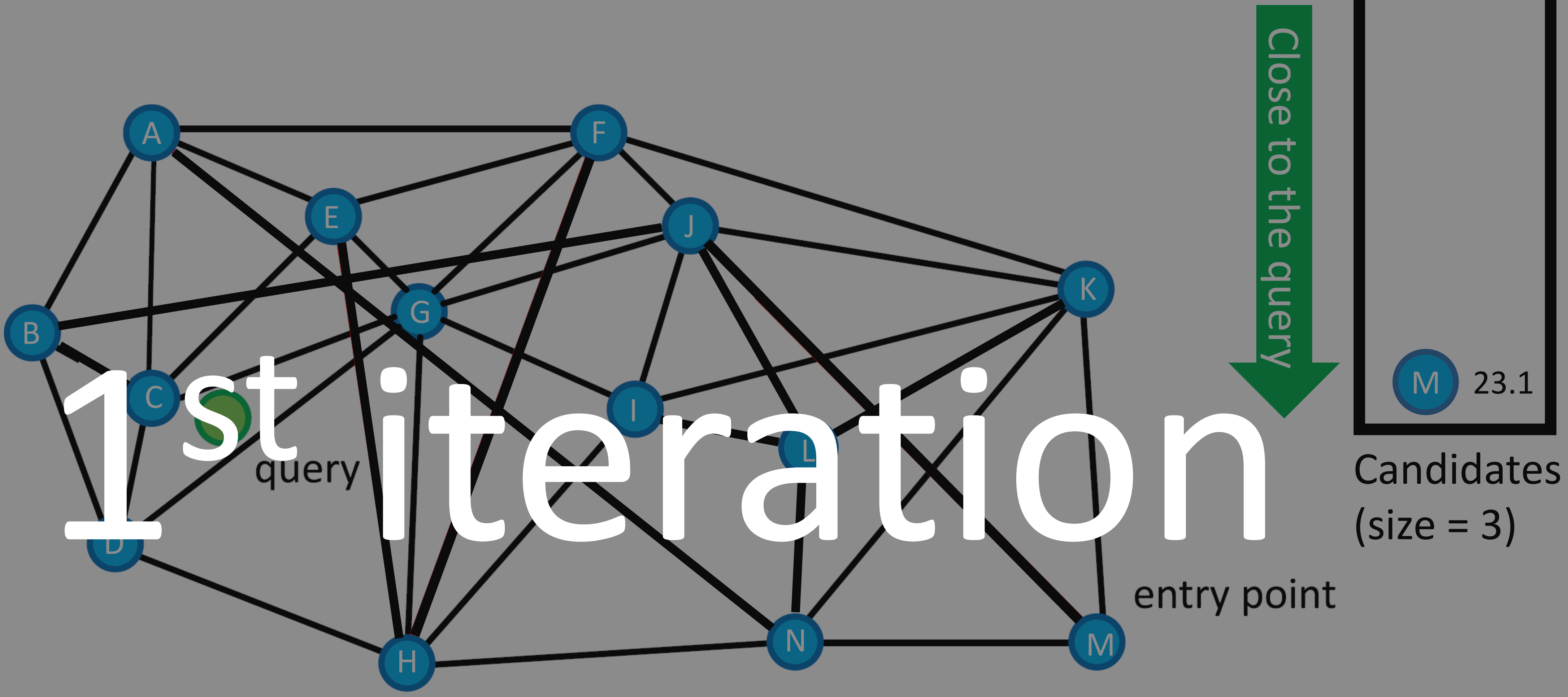
Search

Images are from [Malkov+, Information Systems, 2013]



Search

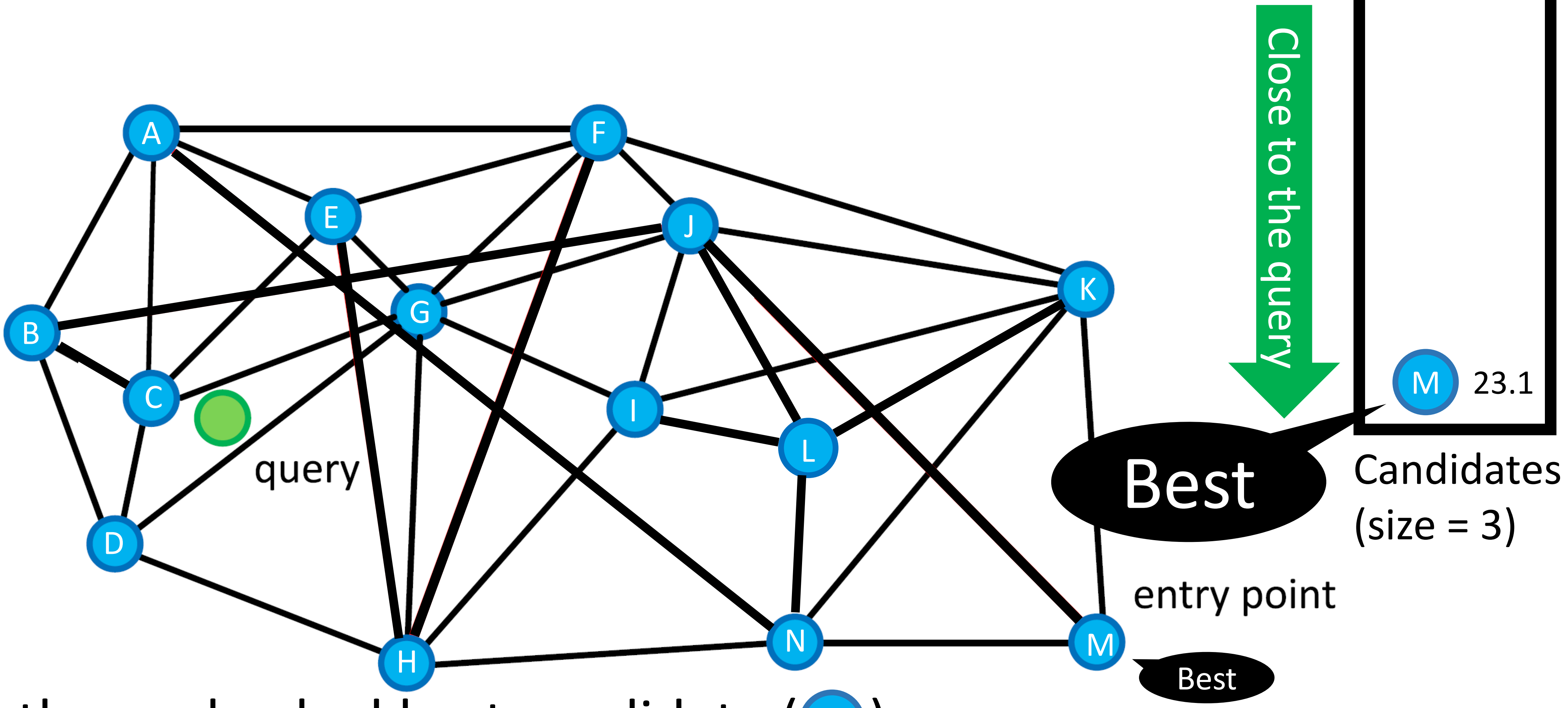
Images are from [Malkov+, Information Systems, 2013]



1st iteration

Search

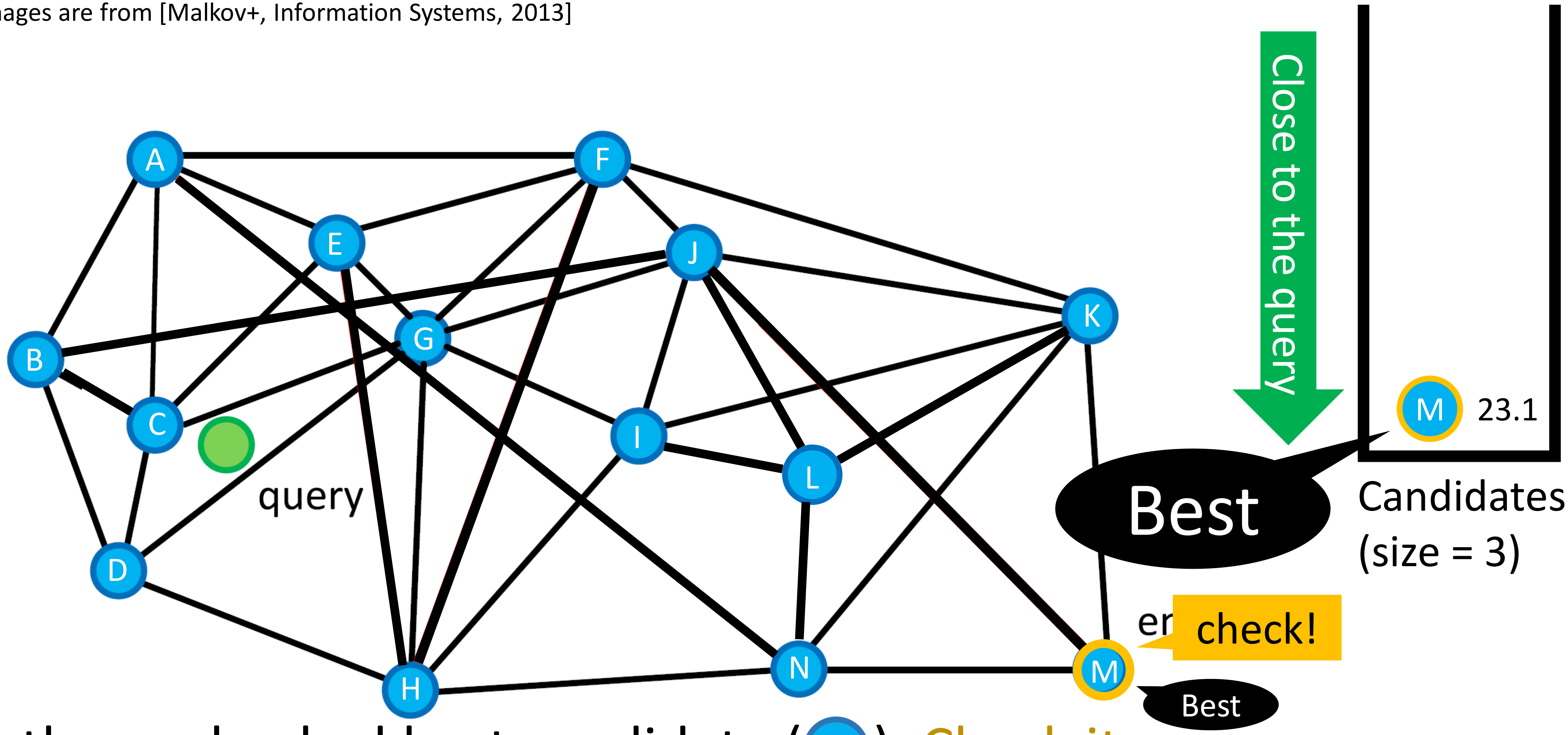
Images are from [Malkov+, Information Systems, 2013]



➢ Pick up the unchecked best candidate (M)

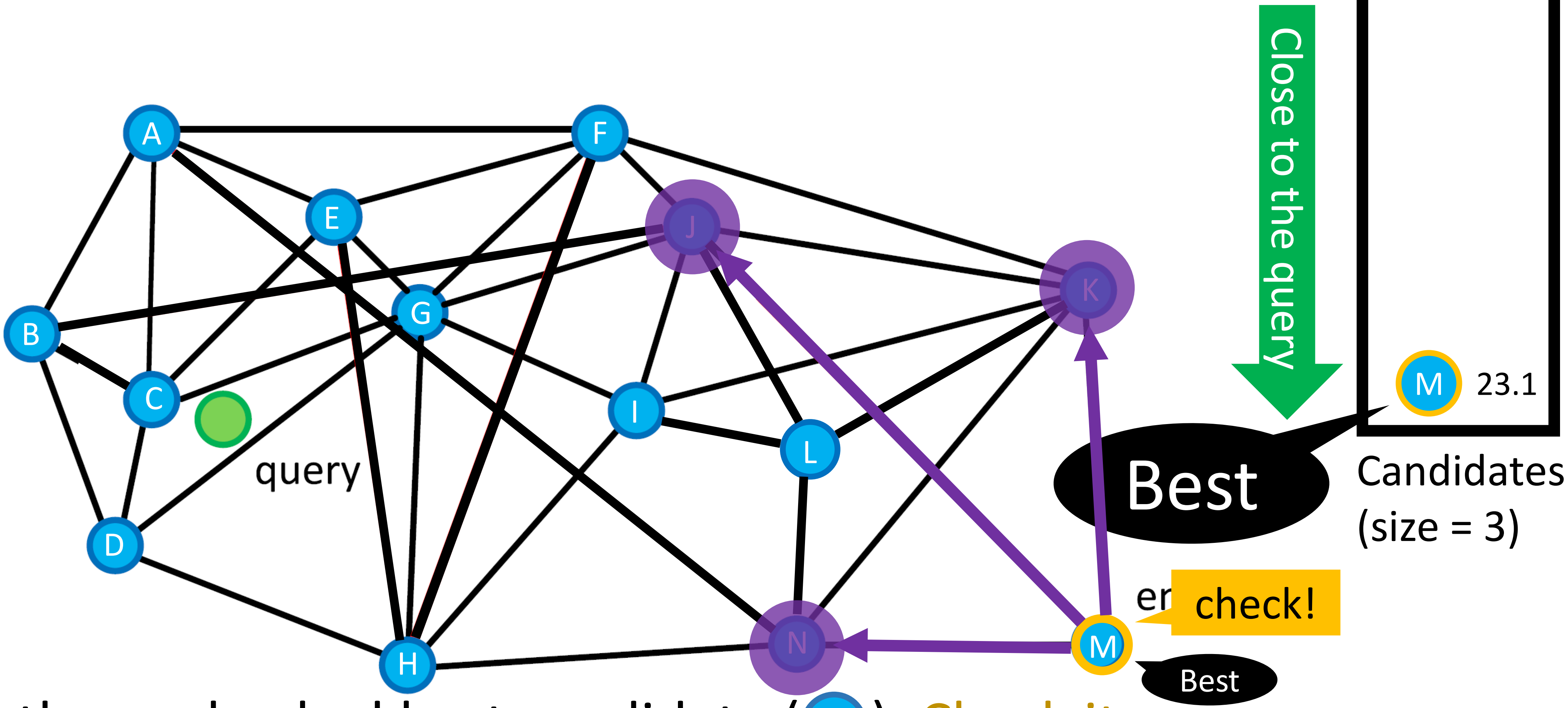
Search

Images are from [Malkov+, Information Systems, 2013]



➢ Pick up the unchecked best candidate (M). Check it.

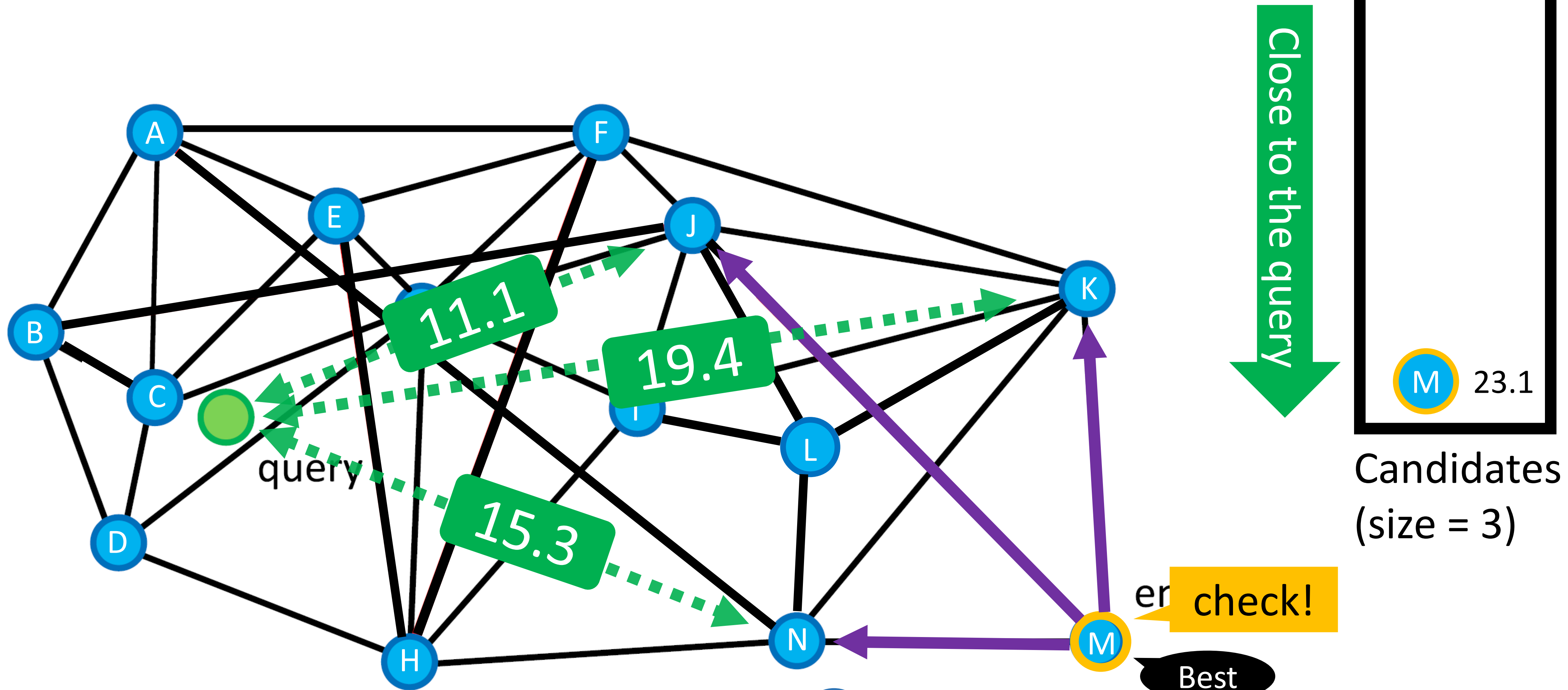
Search Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (M). Check it.
- Find the connected points.

Search

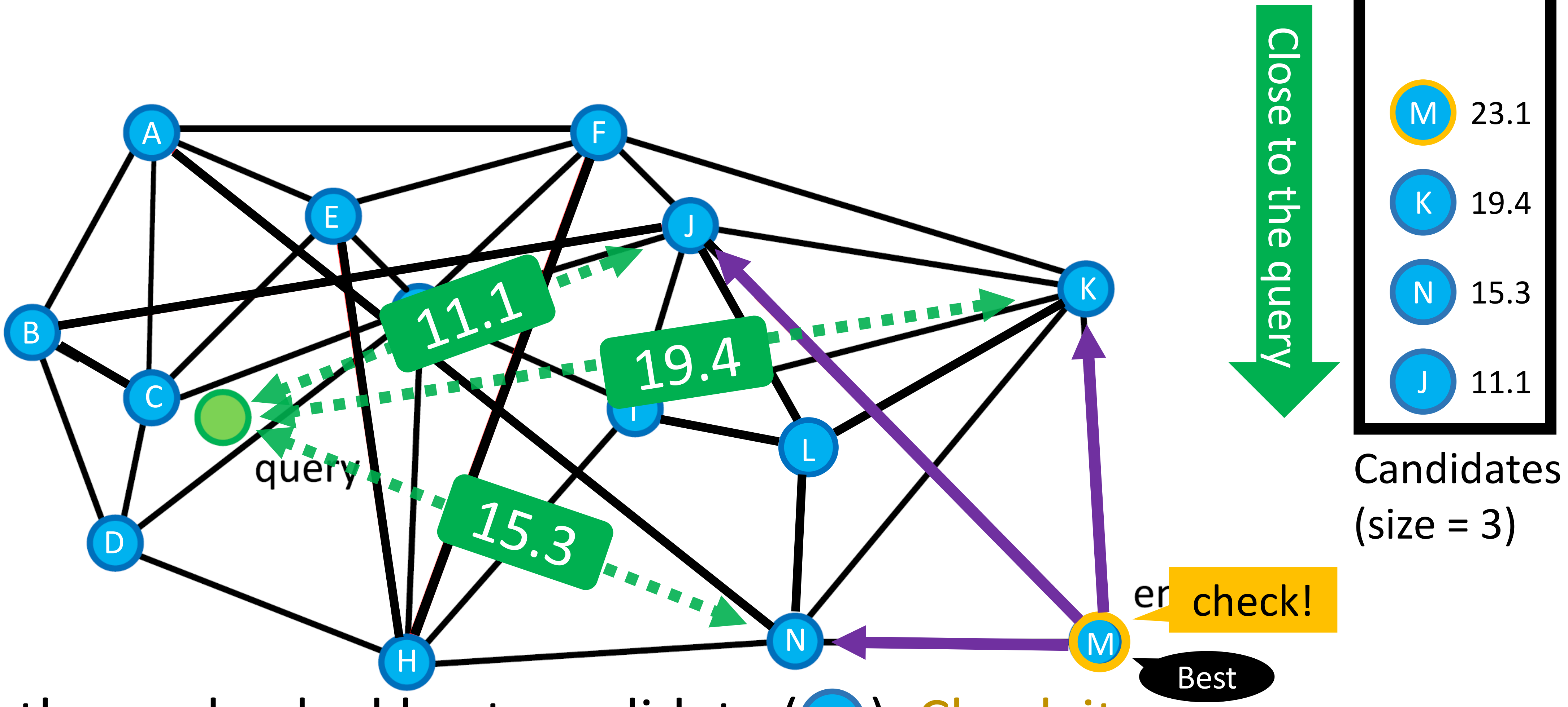
Images are from [Malkov+, Information Systems, 2013]



- ➢ Pick up the unchecked best candidate (M). Check it.
- ➢ Find the connected points.
- ➢ Record the distances to q.

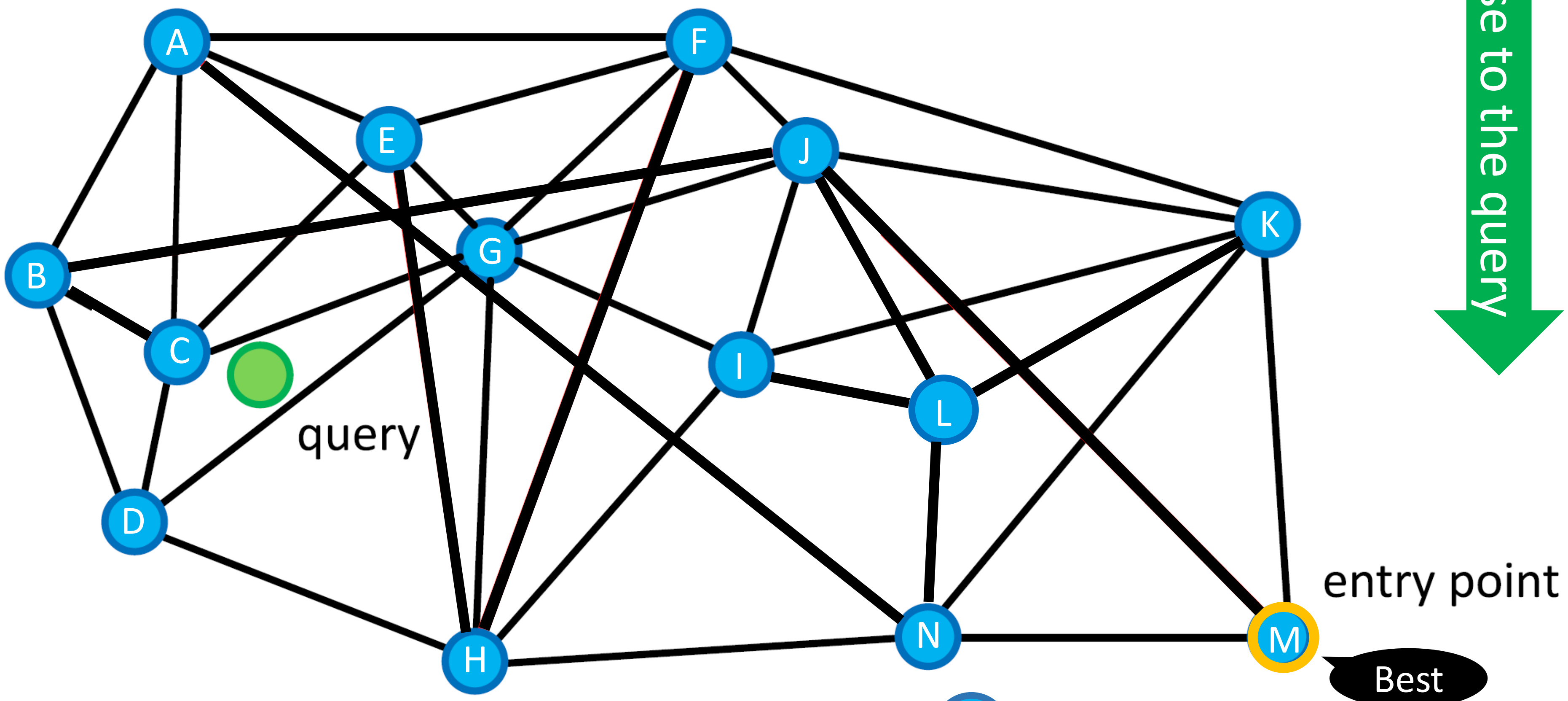
Search

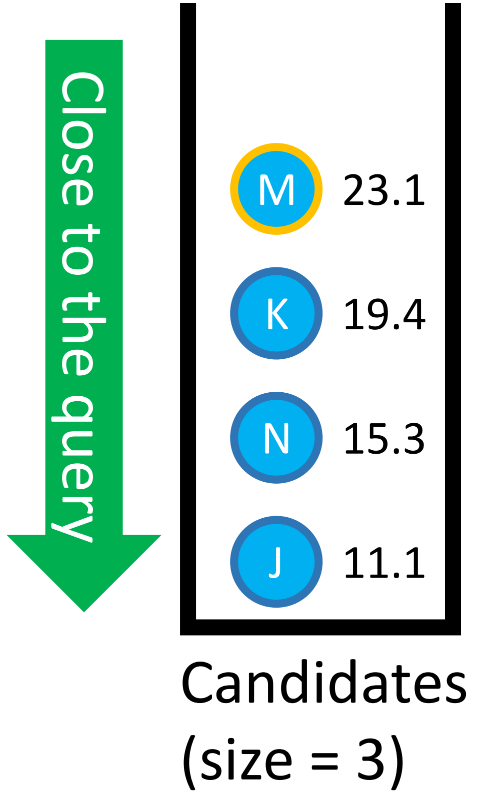
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (M). Check it.
- Find the connected points.
- Record the distances to q.

Search Images are from [Malkov+, Information Systems, 2013]

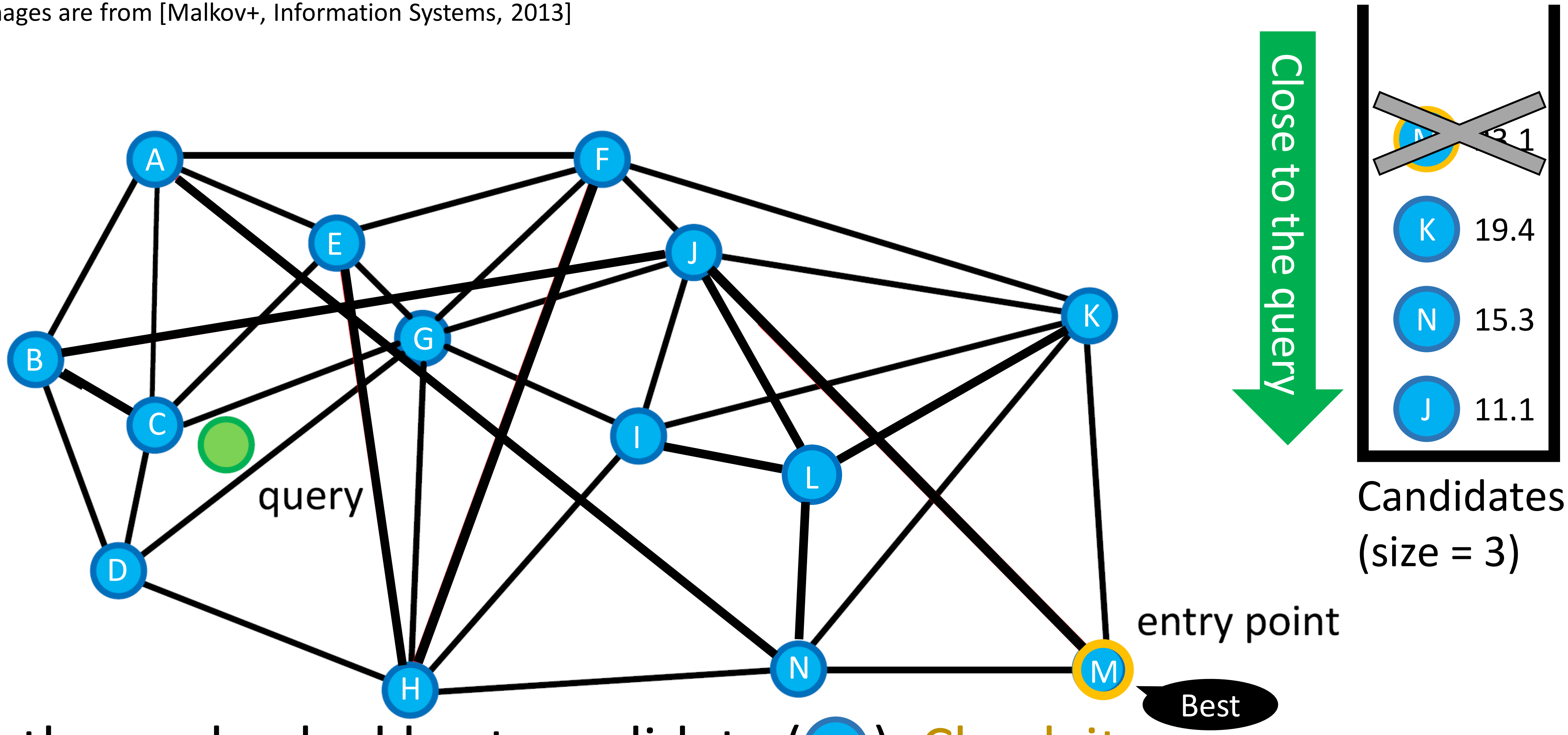




- Pick up the unchecked best candidate (M). Check it.
- Find the connected points.
- Record the distances to q.

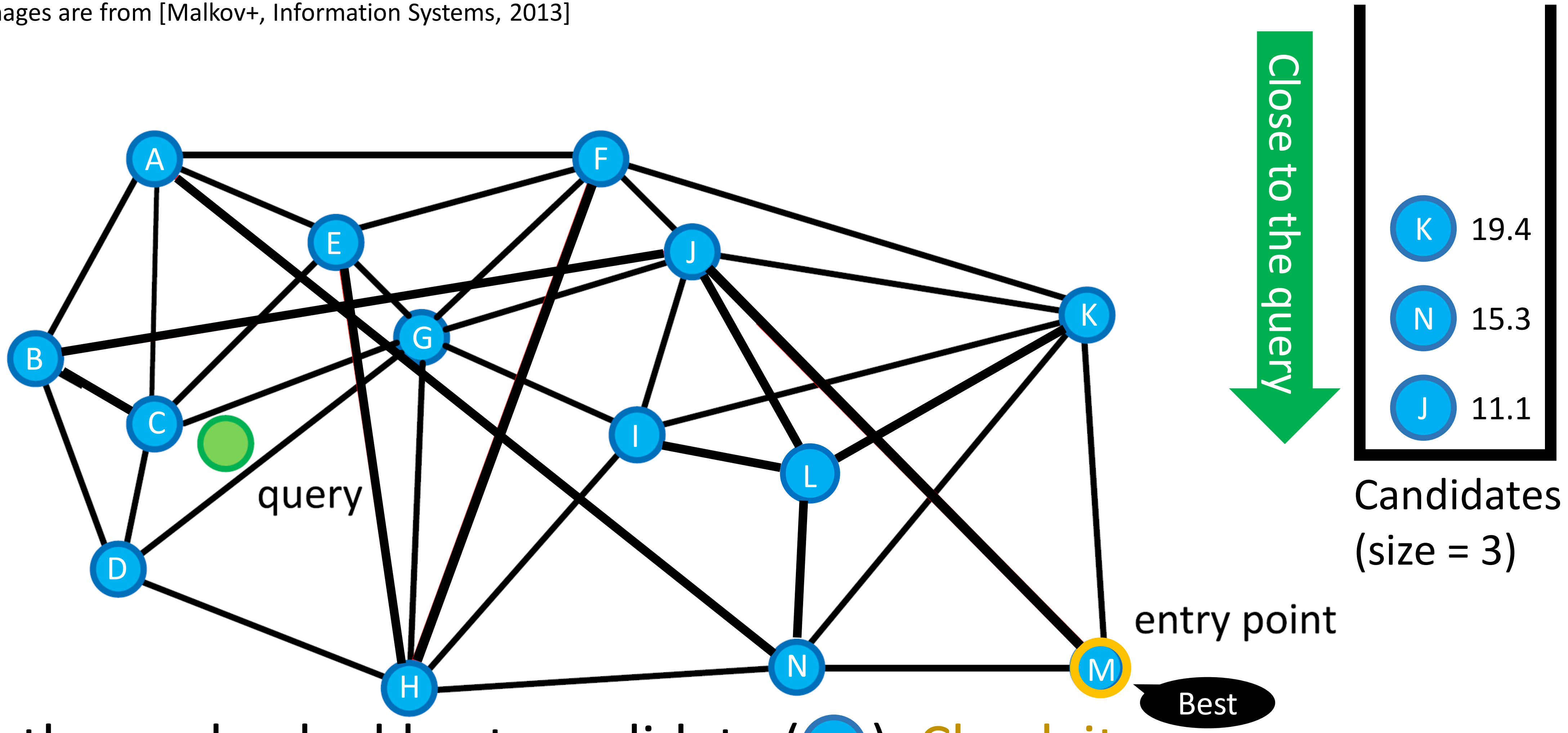
Search

Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (M). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

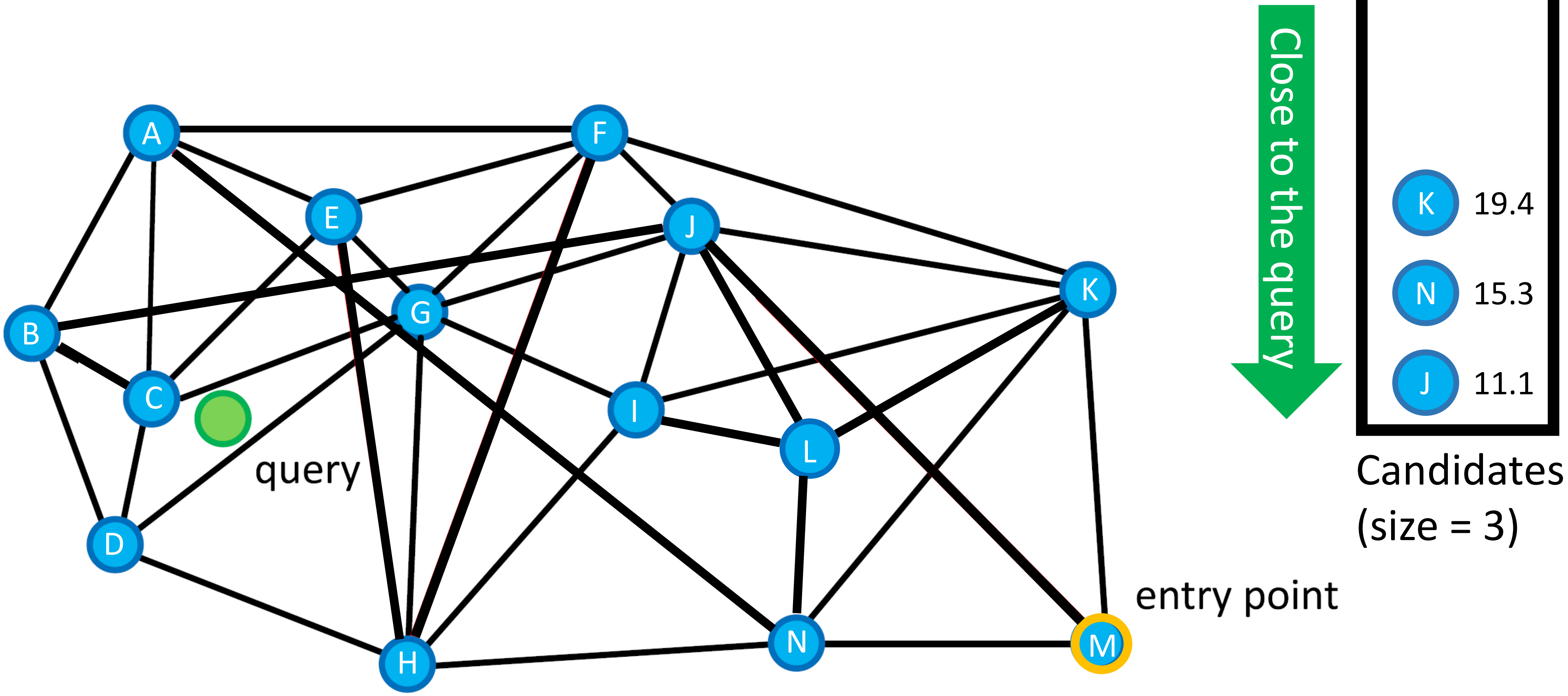
Search Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (M). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

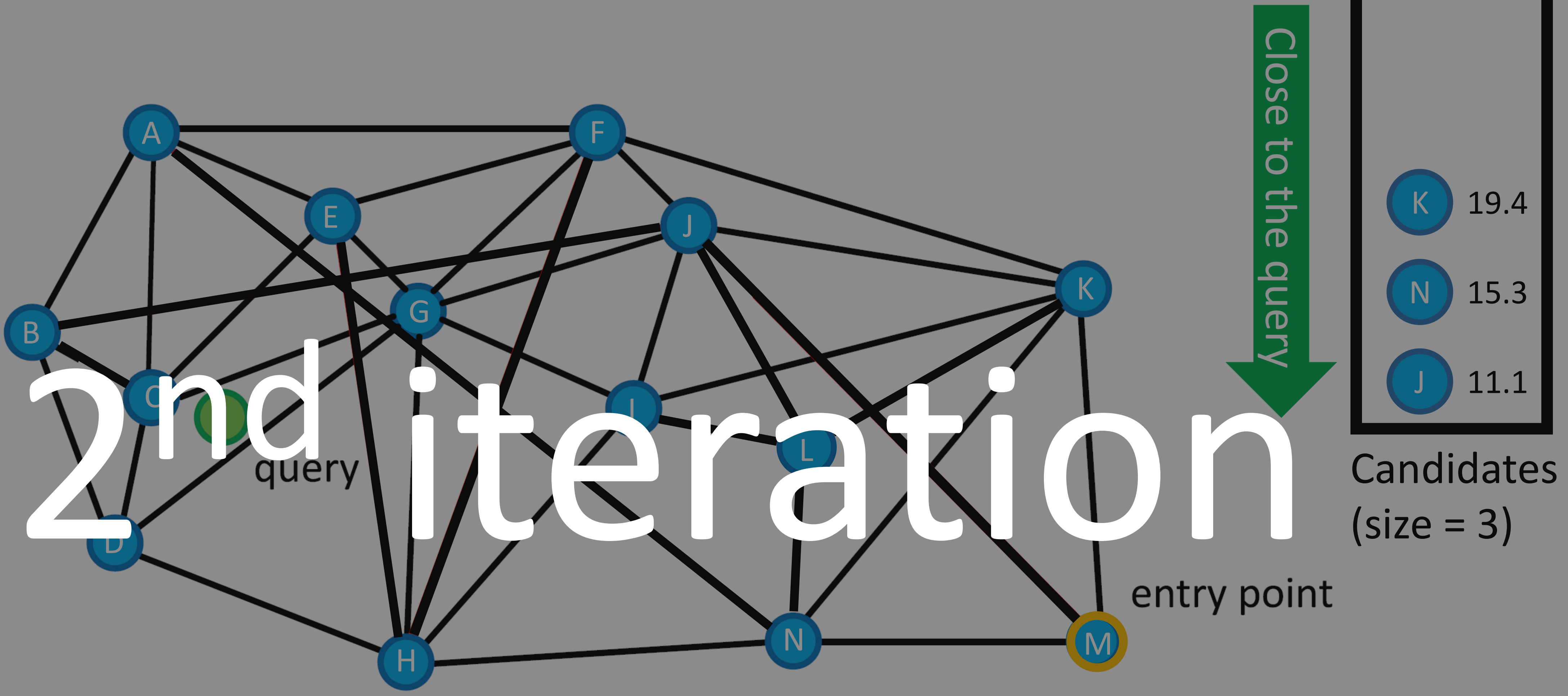
Search

Images are from [Malkov+, Information Systems, 2013]



Search

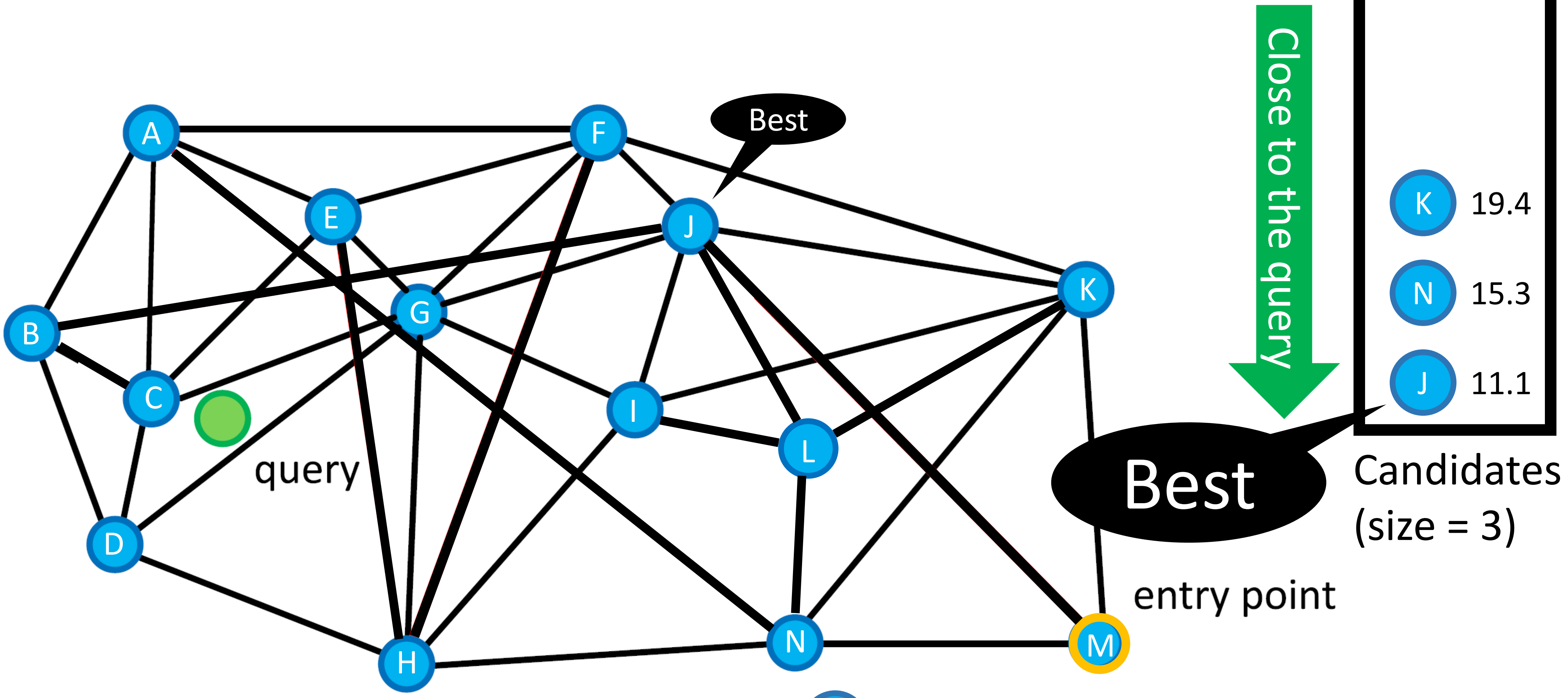
Images are from [Malkov+, Information Systems, 2013]



2nd iteration

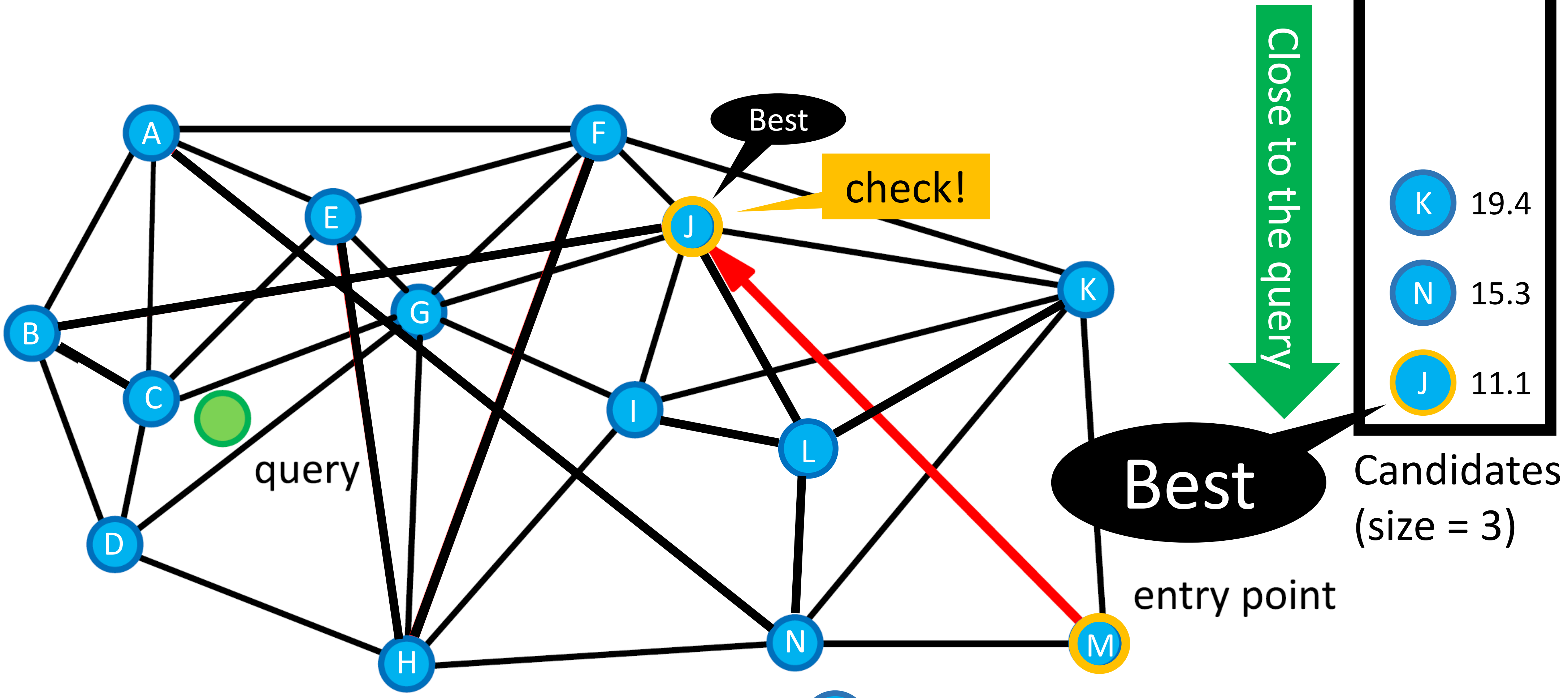
Search

Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (J)

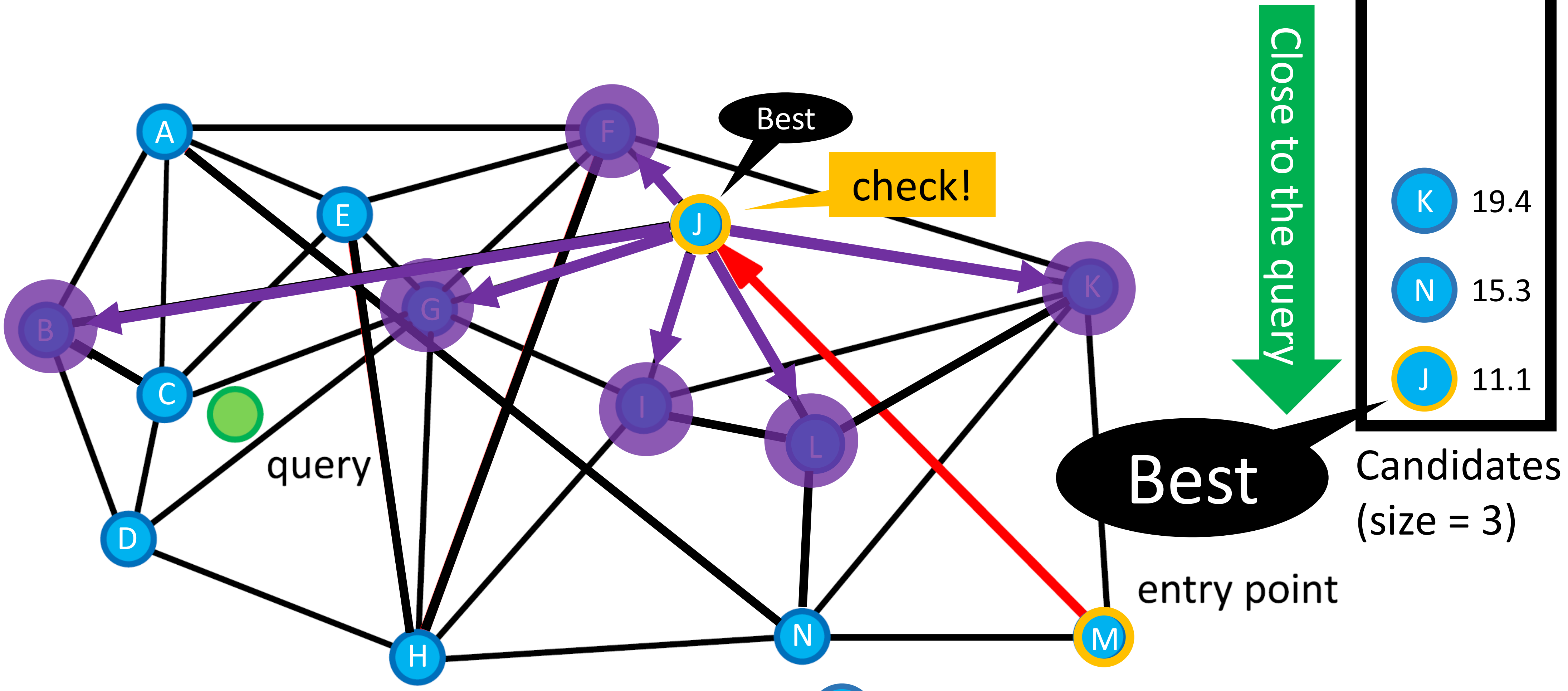
Search Images are from [Malkov+, Information Systems, 2013]



➢ Pick up the unchecked best candidate (J). Check it.

Search

Images are from [Malkov+, Information Systems, 2013]

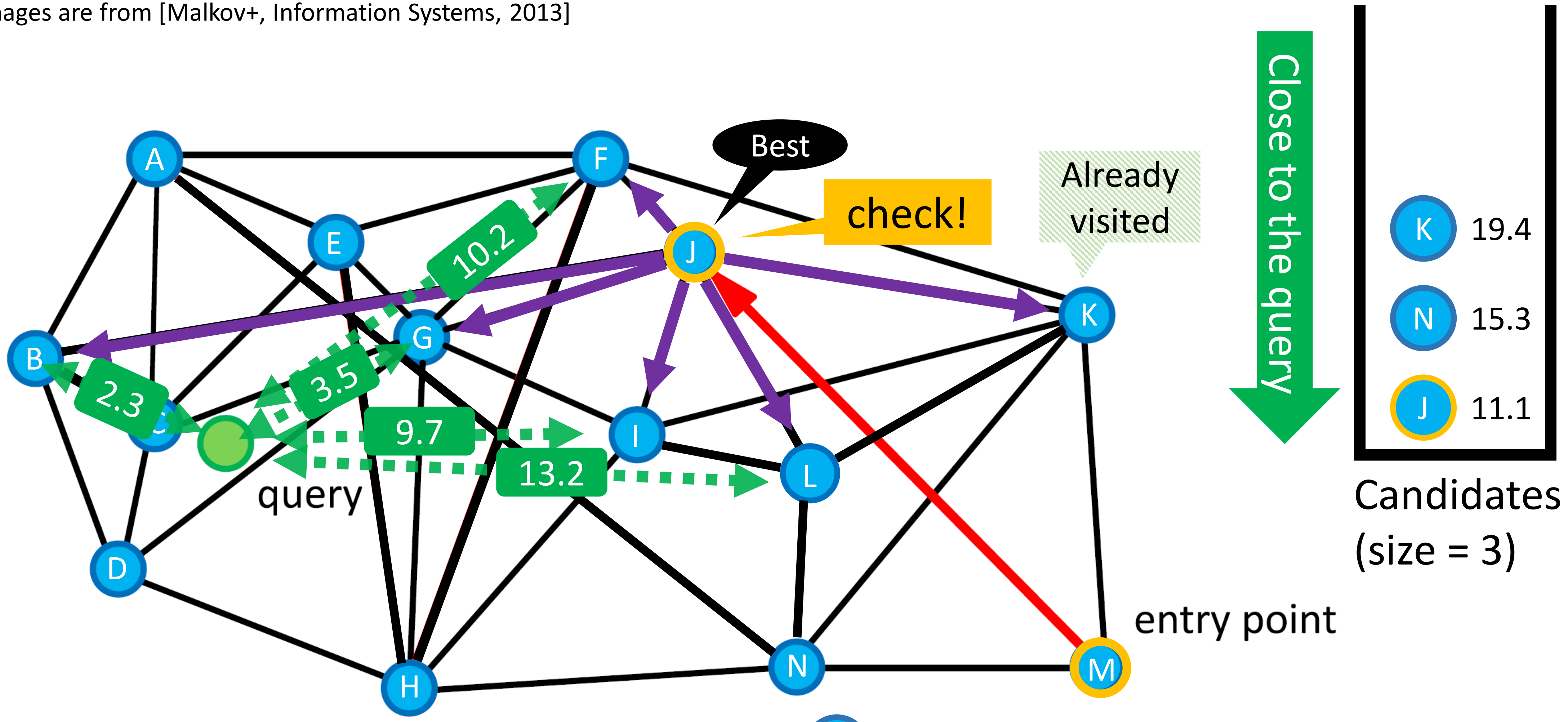


➢ Pick up the unchecked best candidate (J). Check it.

➢ Find the connected points.

Search

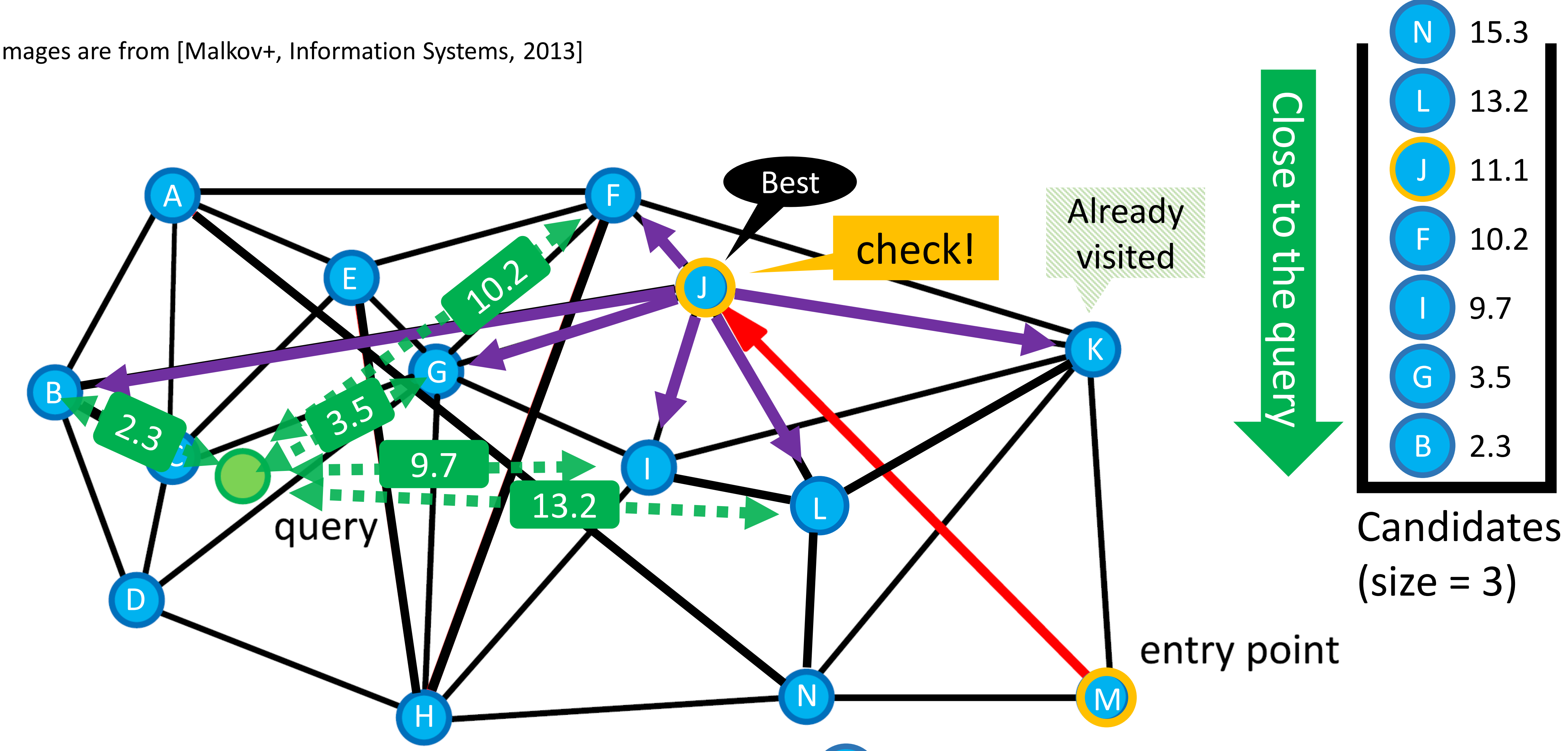
Images are from [Malkov+, Information Systems, 2013]



- ➢ Pick up the unchecked best candidate (J). Check it.
- ➢ Find the connected points.
- ➢ Record the distances to q.

Search

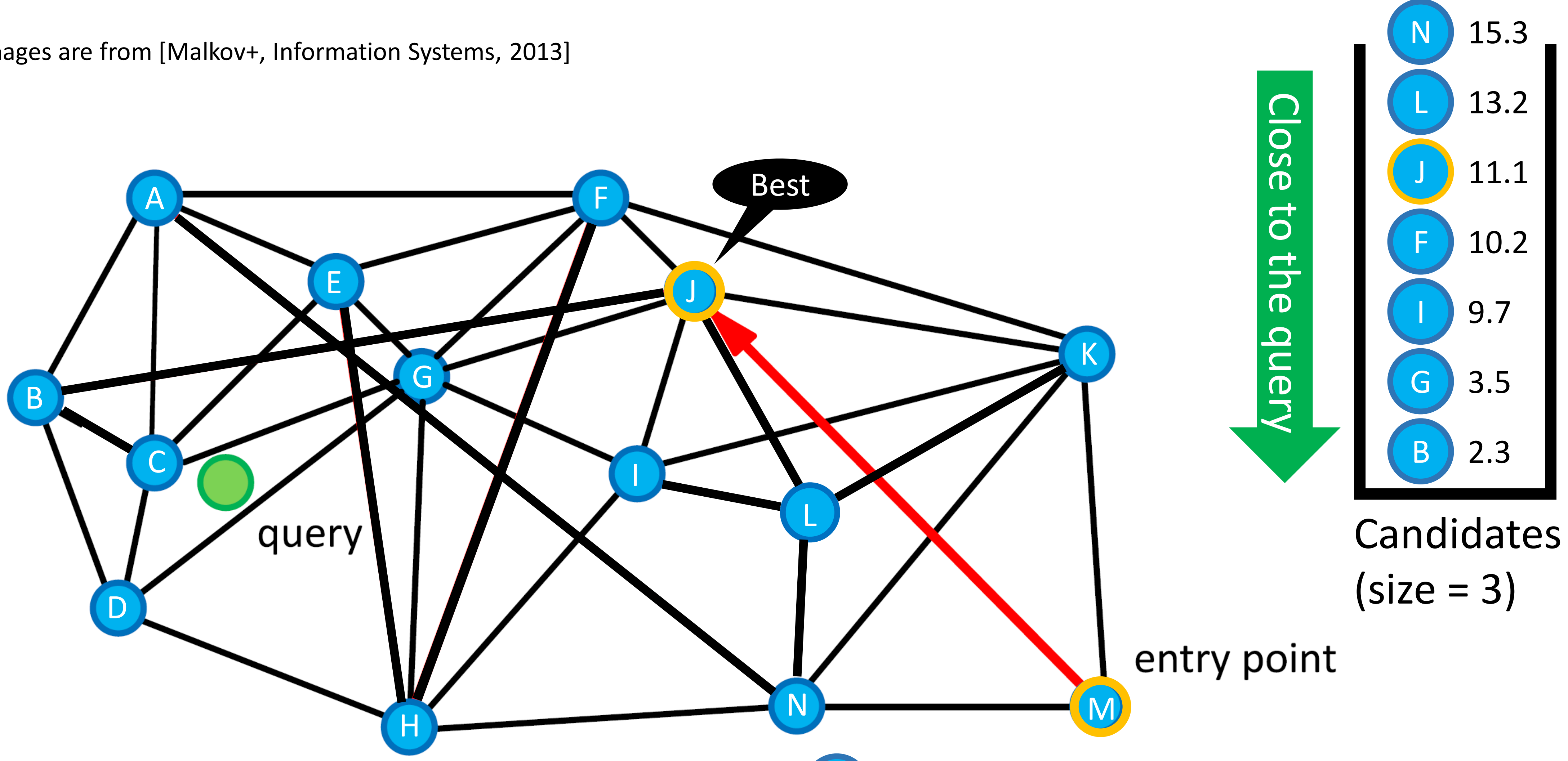
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (J). Check it.
- Find the connected points.
- Record the distances to q.

Search

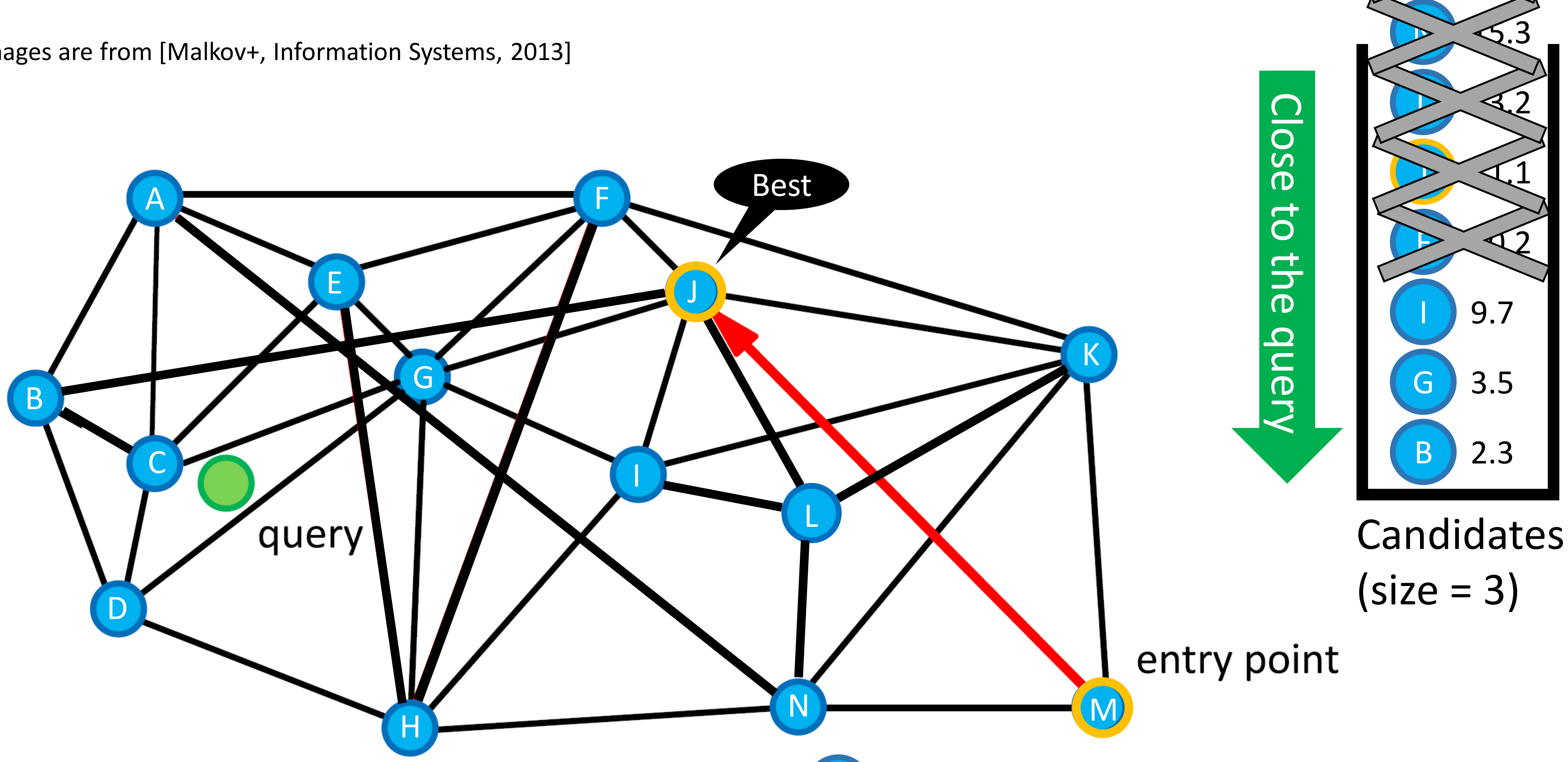
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (J). Check it.
- Find the connected points.
- Record the distances to q.

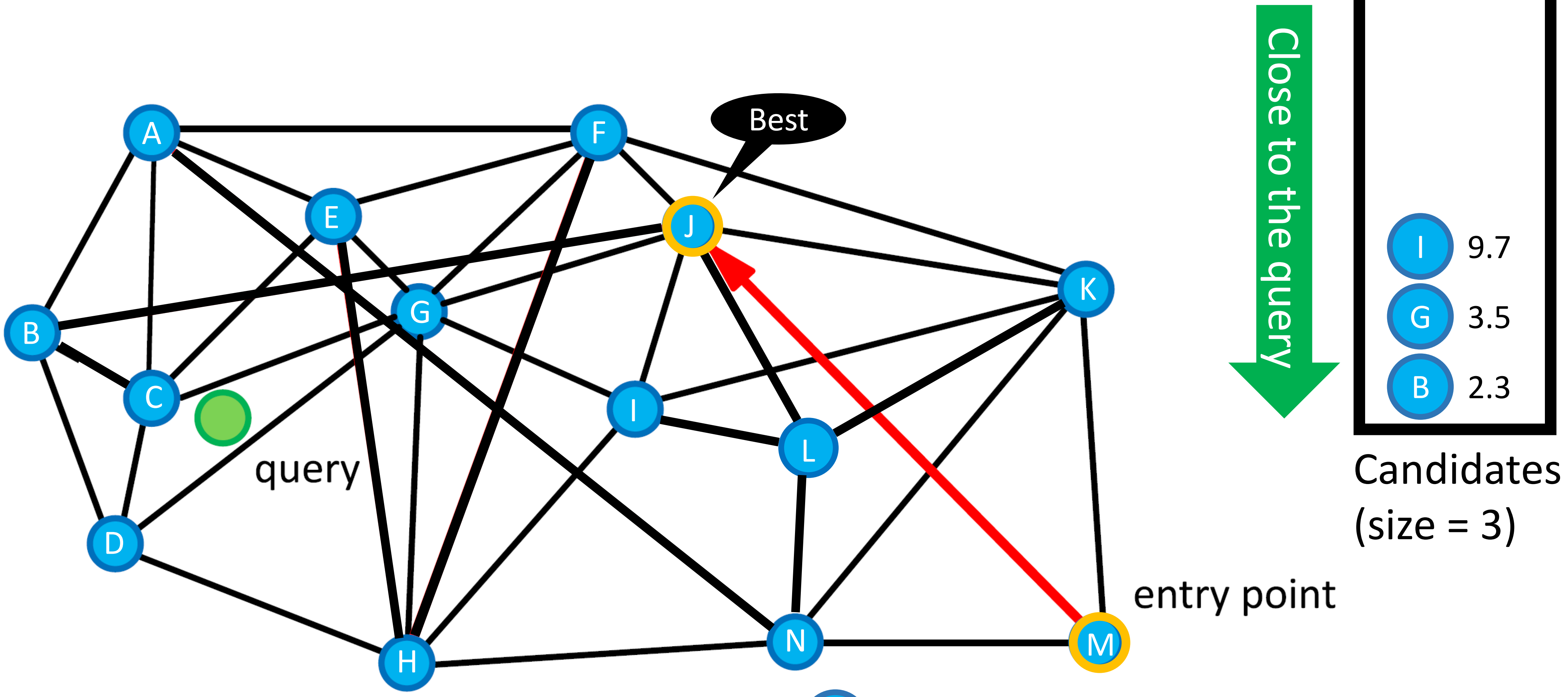
Search

Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (J). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

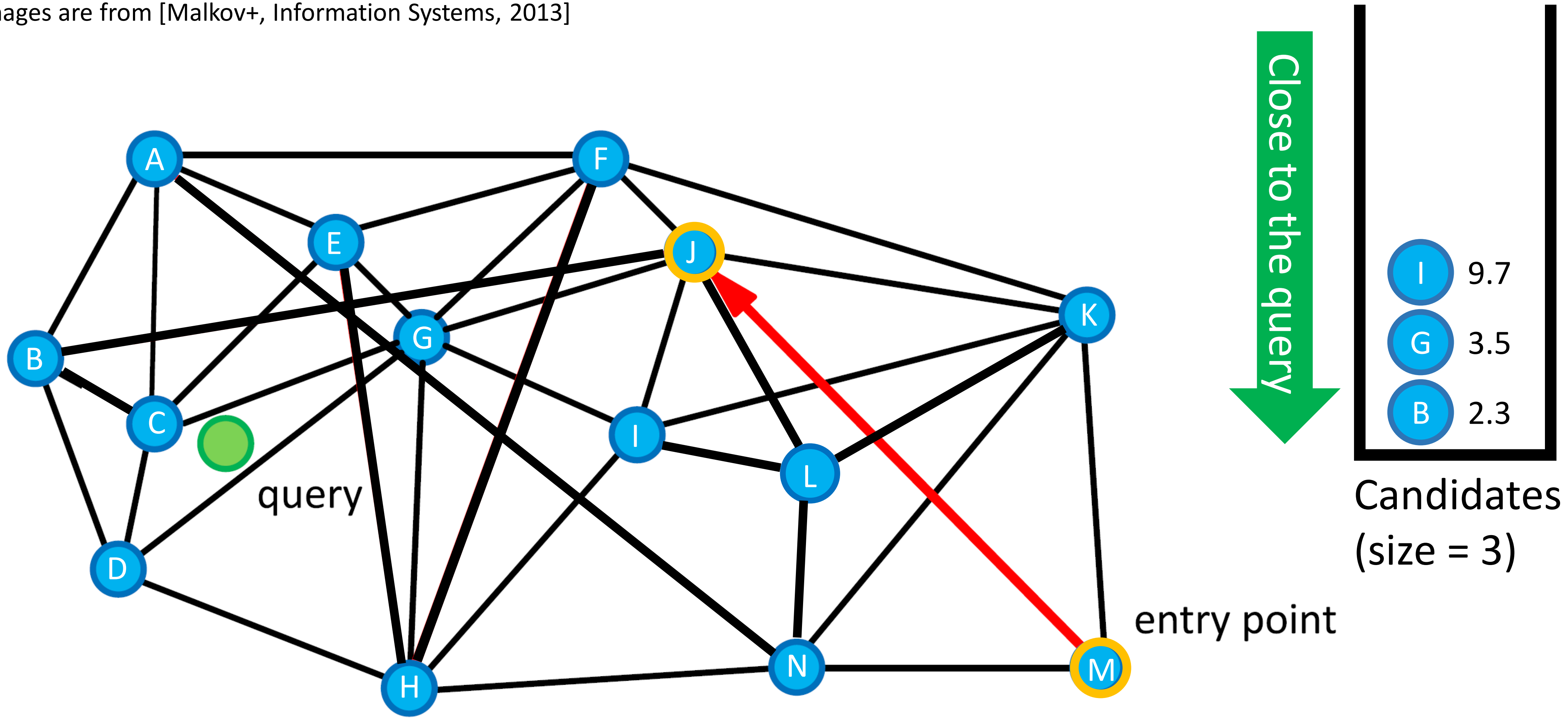
Search Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (J). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

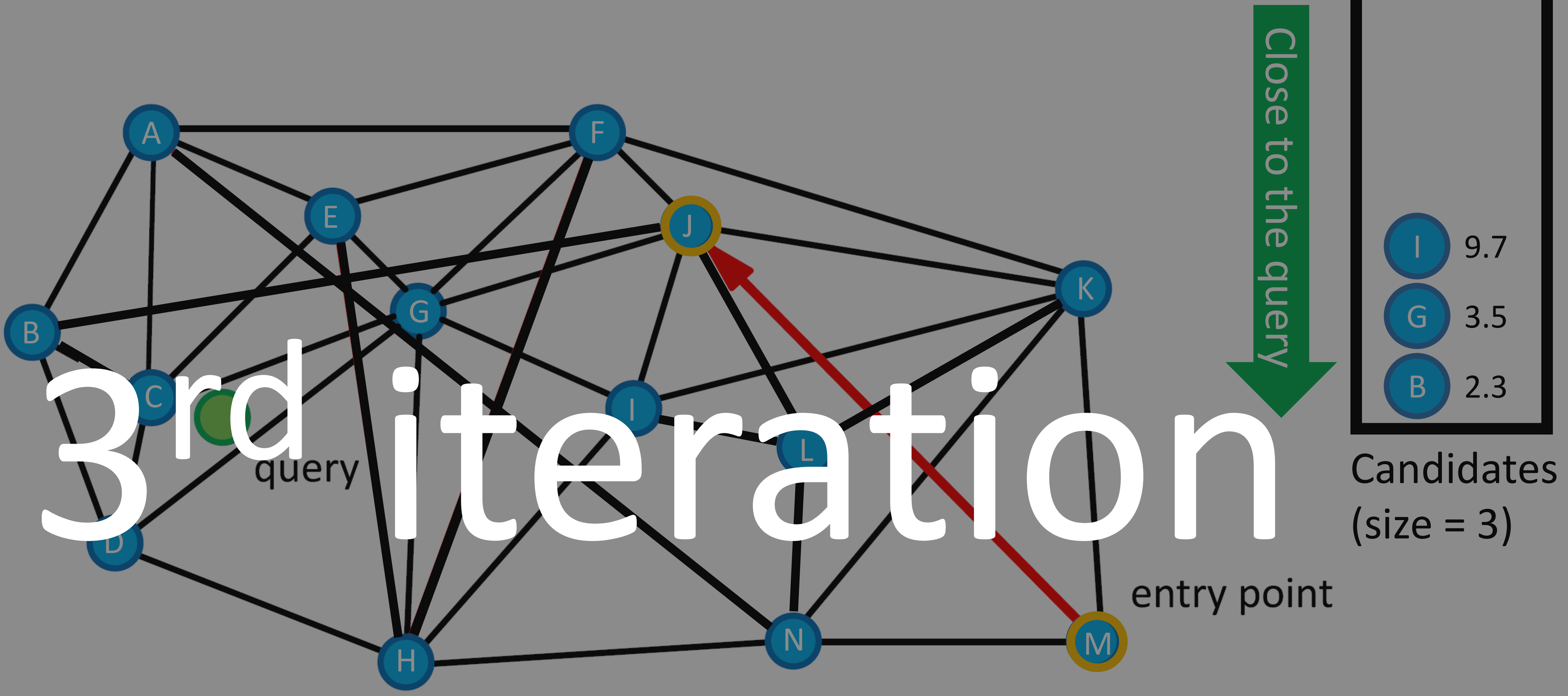
Search

Images are from [Malkov+, Information Systems, 2013]



Search

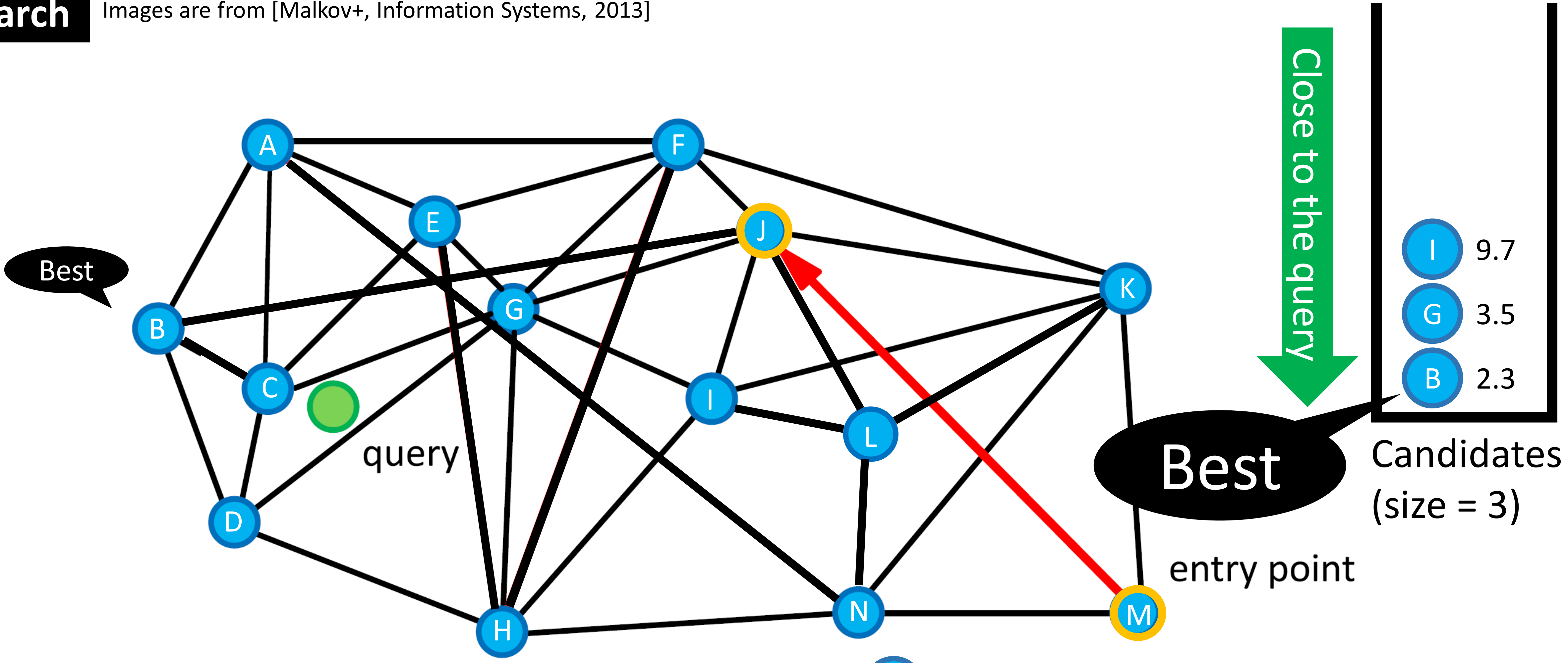
Images are from [Malkov+, Information Systems, 2013]



3rd iteration

Search

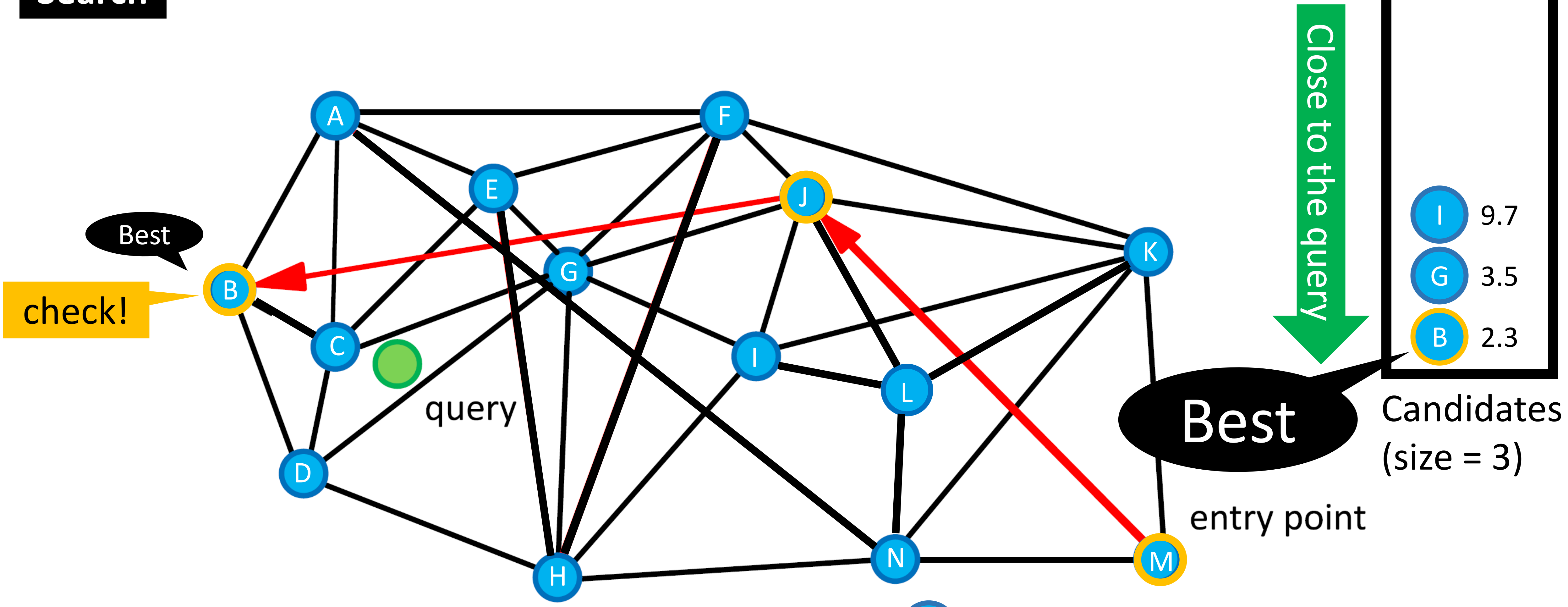
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (B)

Search

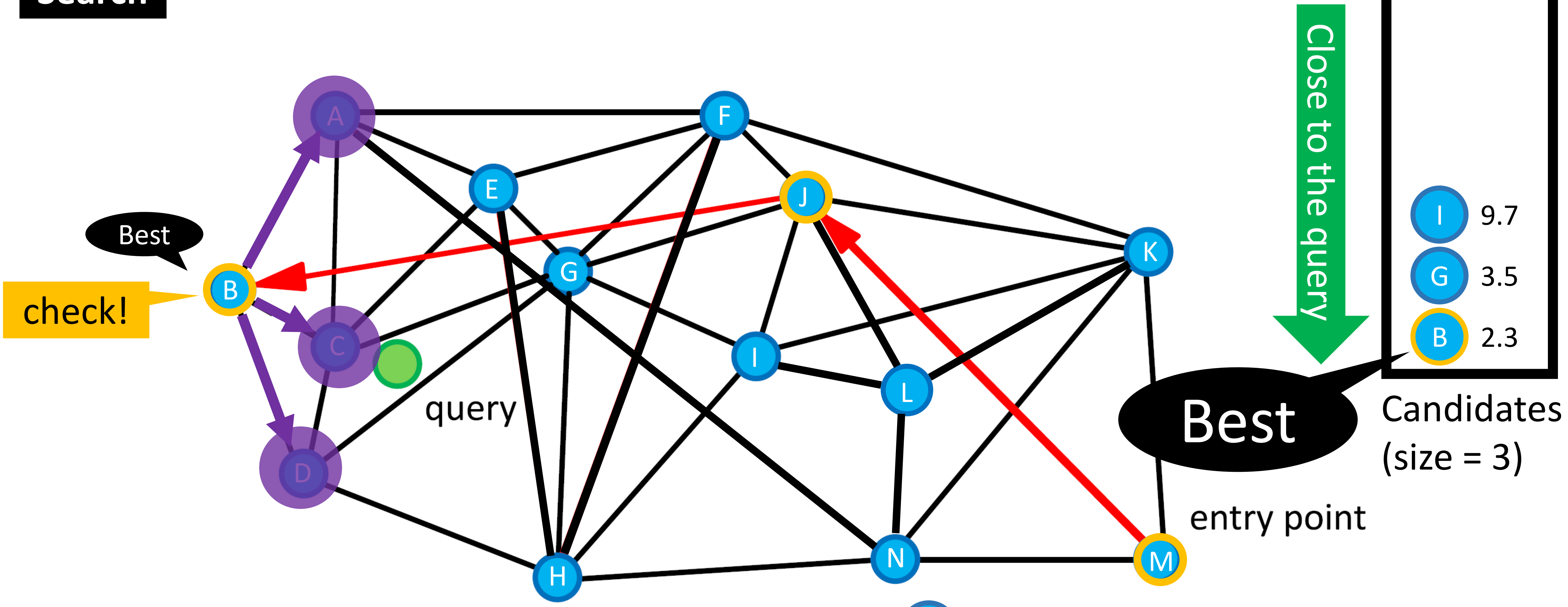
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (B). Check it.

Search

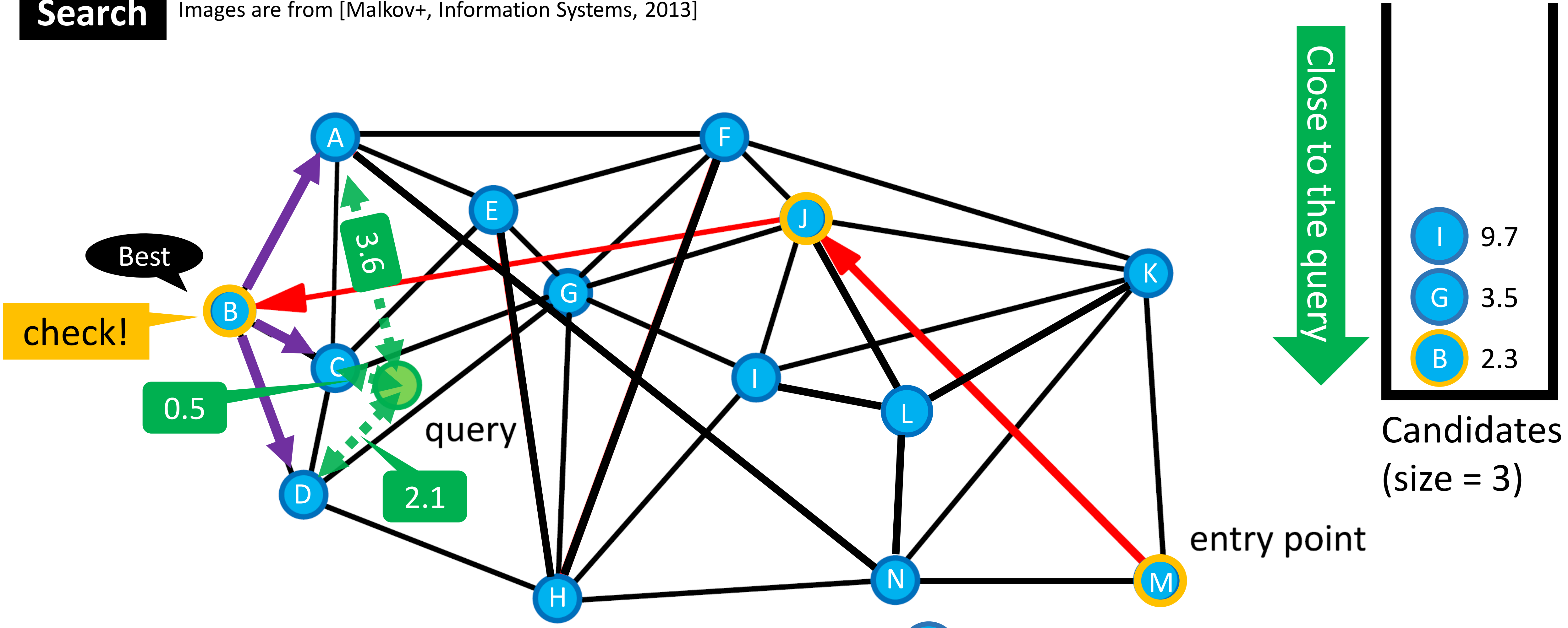
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (B). Check it.
- Find the connected points.

Search

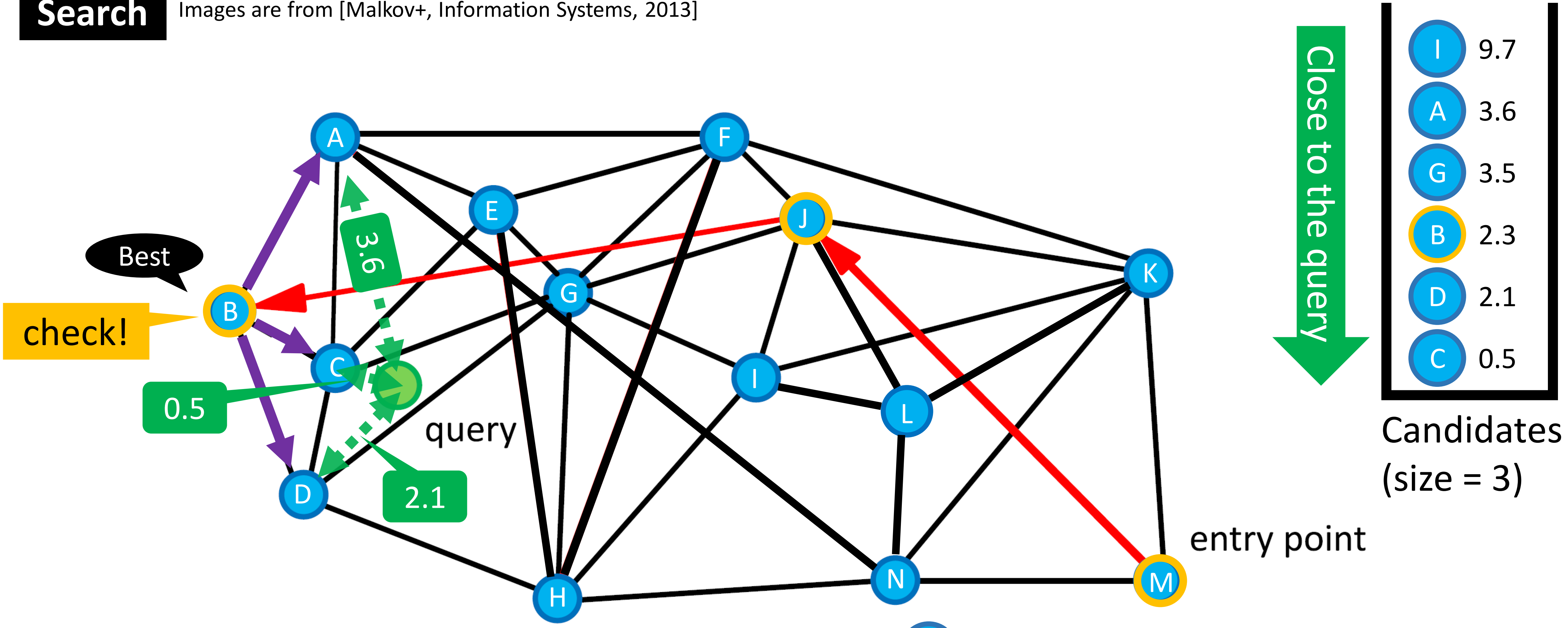
Images are from [Malkov+, Information Systems, 2013]



- ➢ Pick up the unchecked best candidate (B). Check it.
- ➢ Find the connected points.
- ➢ Record the distances to q.

Search

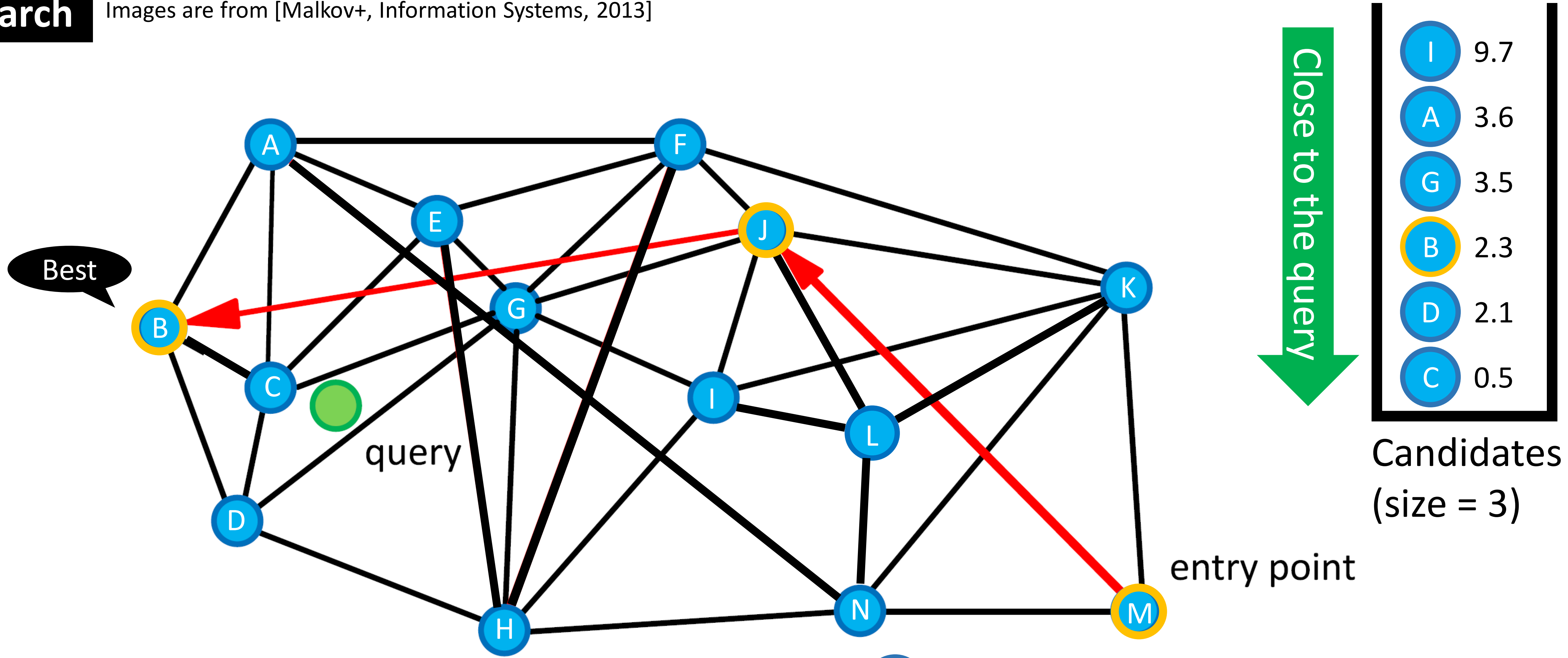
Images are from [Malkov+, Information Systems, 2013]



- ➢ Pick up the unchecked best candidate (B). Check it.
- ➢ Find the connected points.
- ➢ Record the distances to q.

Search

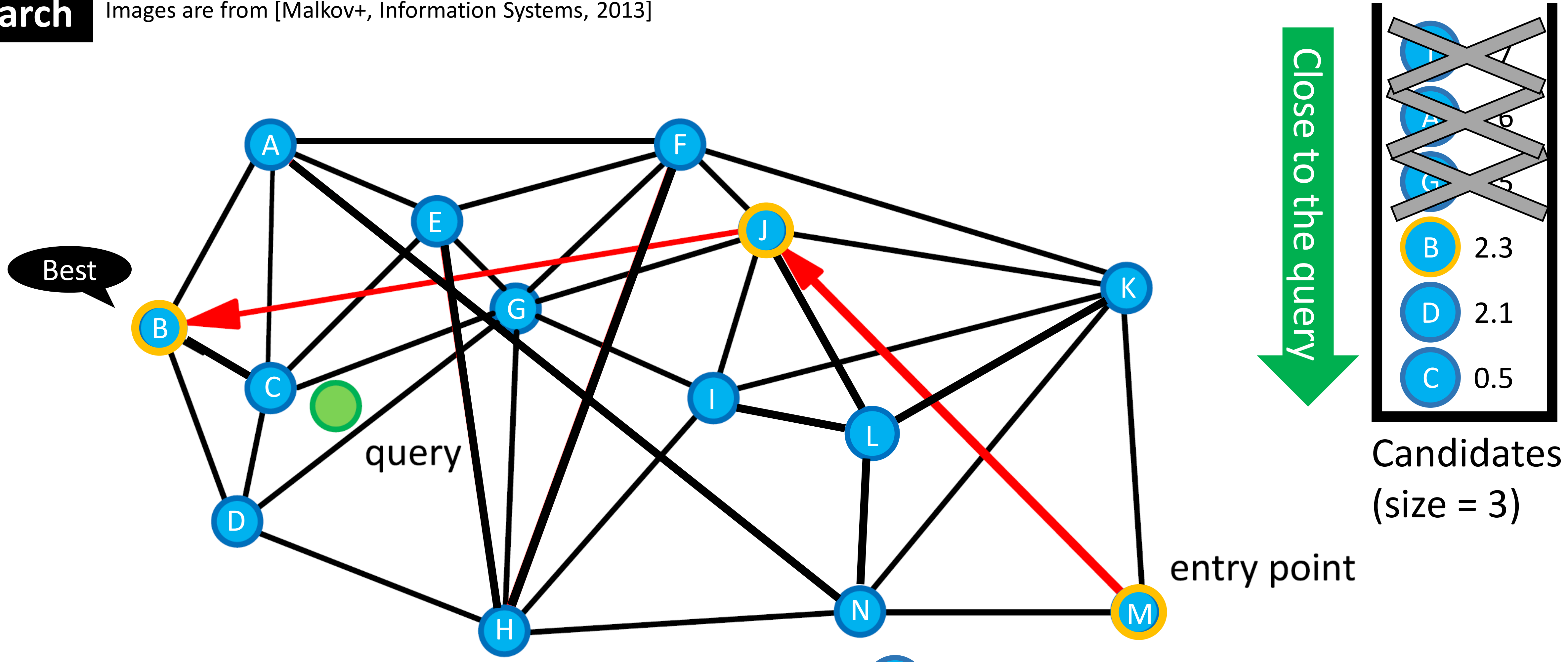
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (B). Check it.
- Find the connected points.
- Record the distances to q.

Search

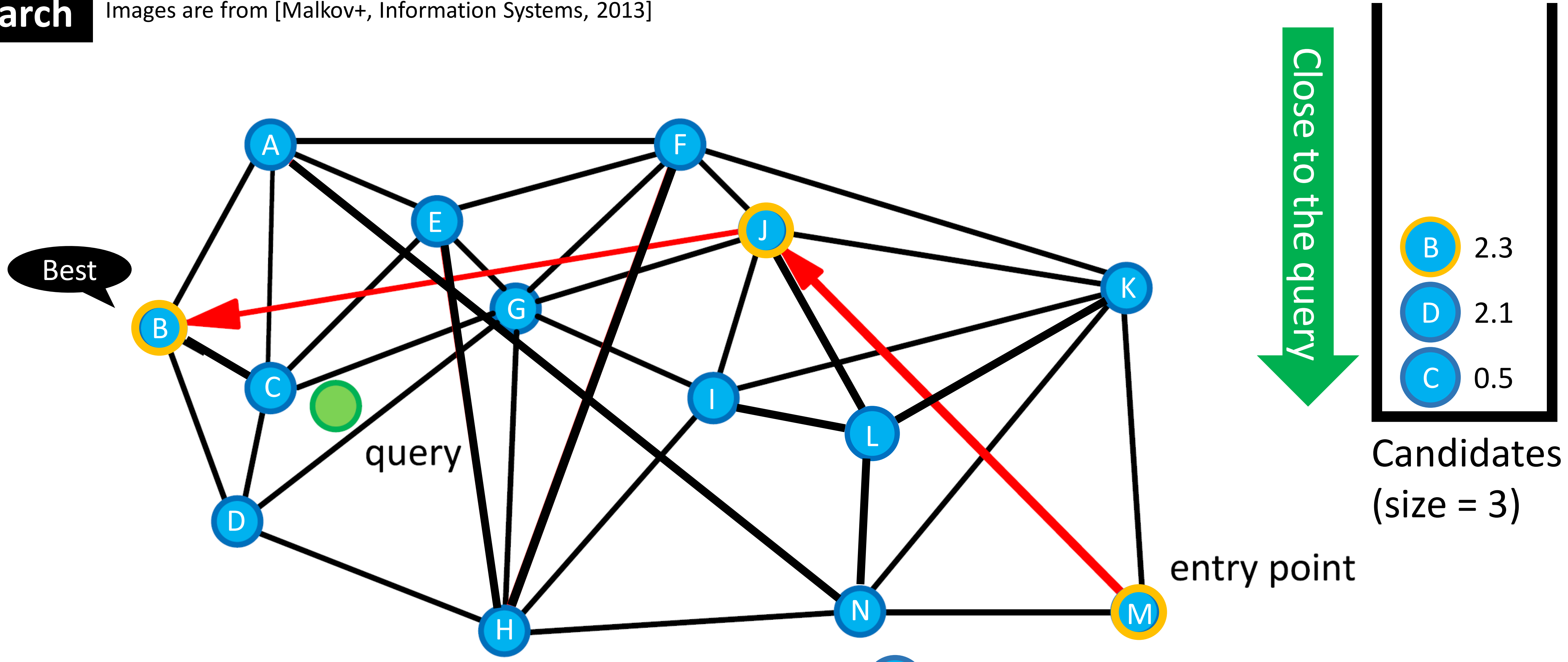
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (B). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

Search

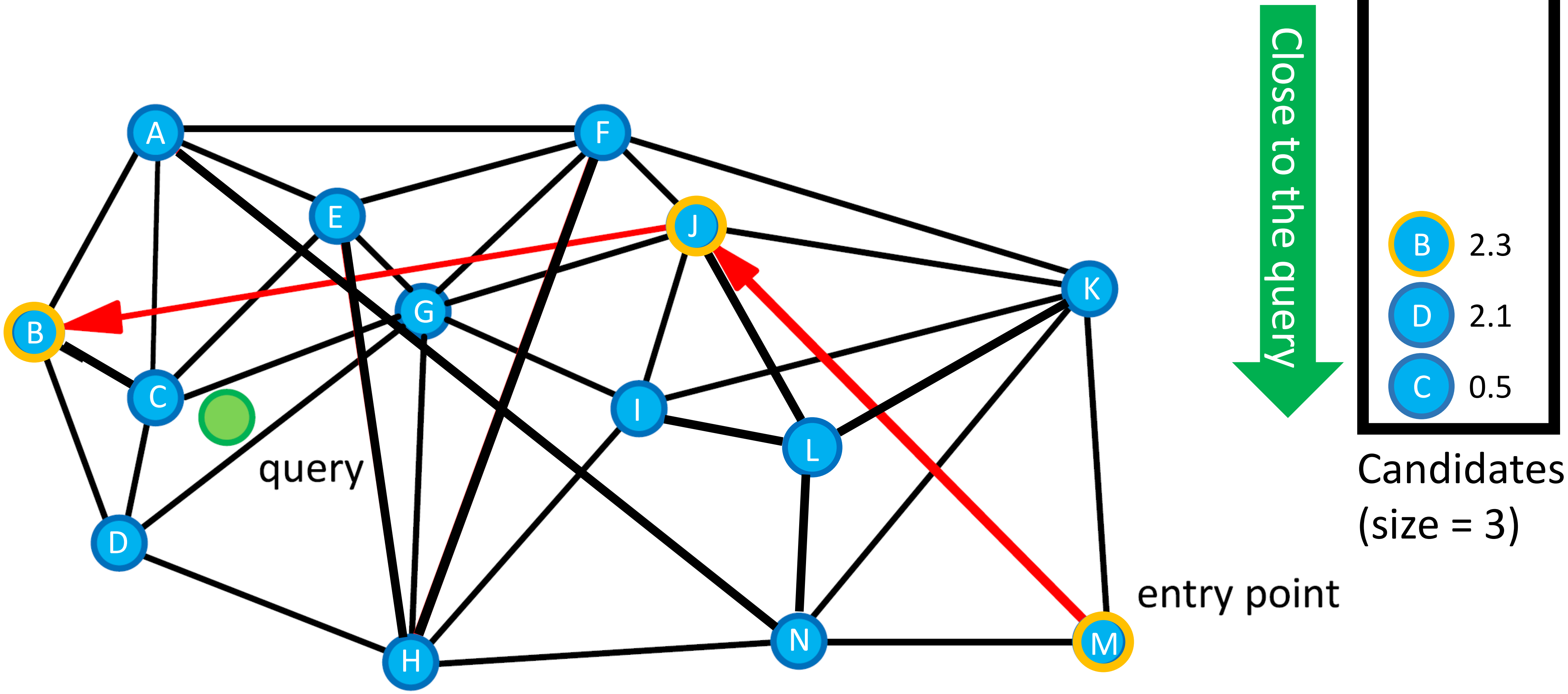
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (B). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

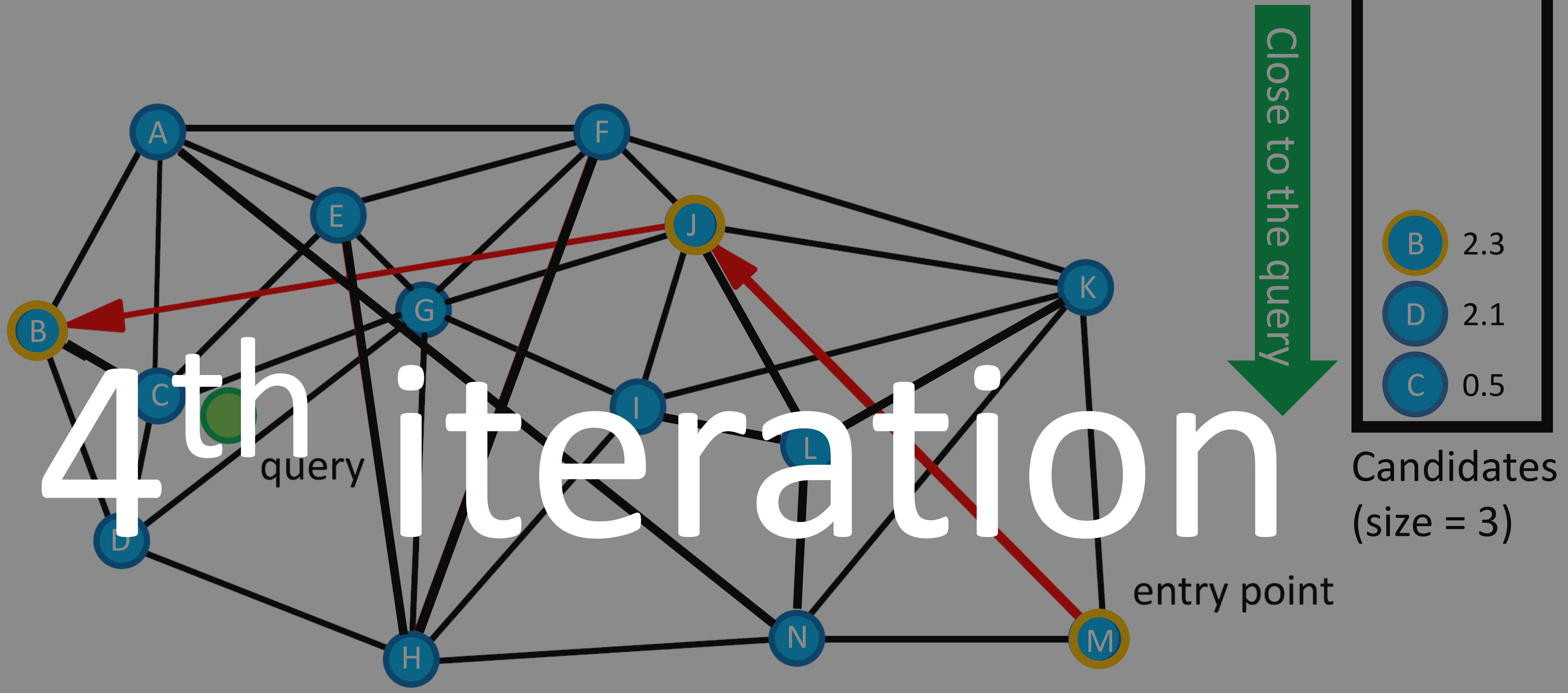
Search

Images are from [Malkov+, Information Systems, 2013]



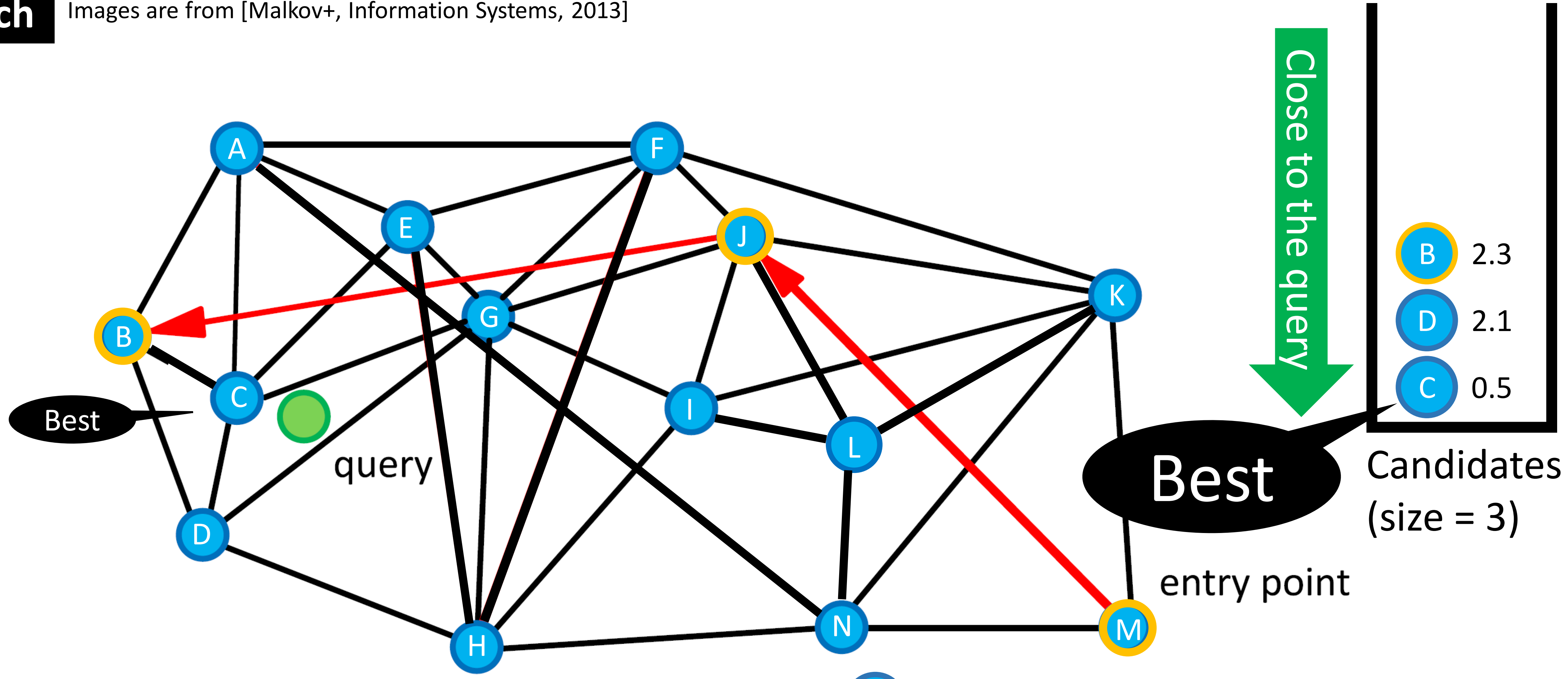
Search

Images are from [Malkov+, Information Systems, 2013]



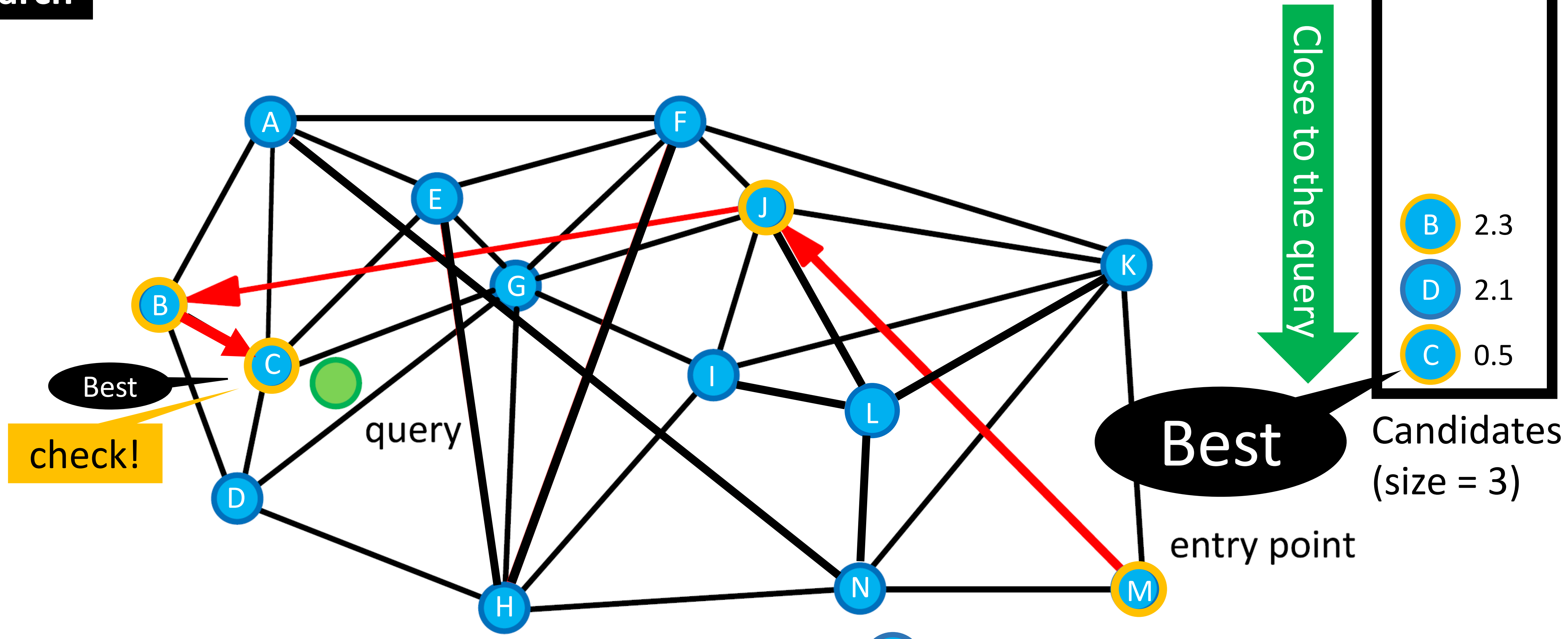
Search

Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (C).

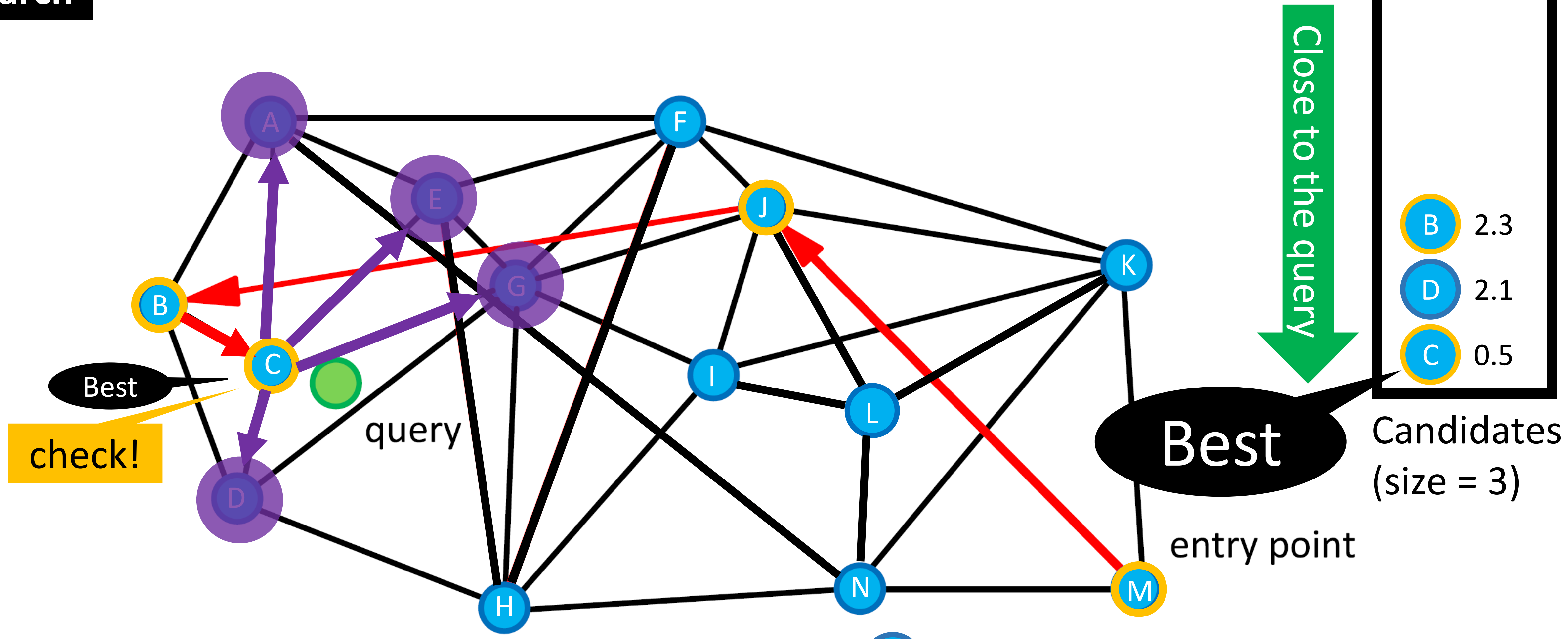
Search Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (C). Check it.

Search

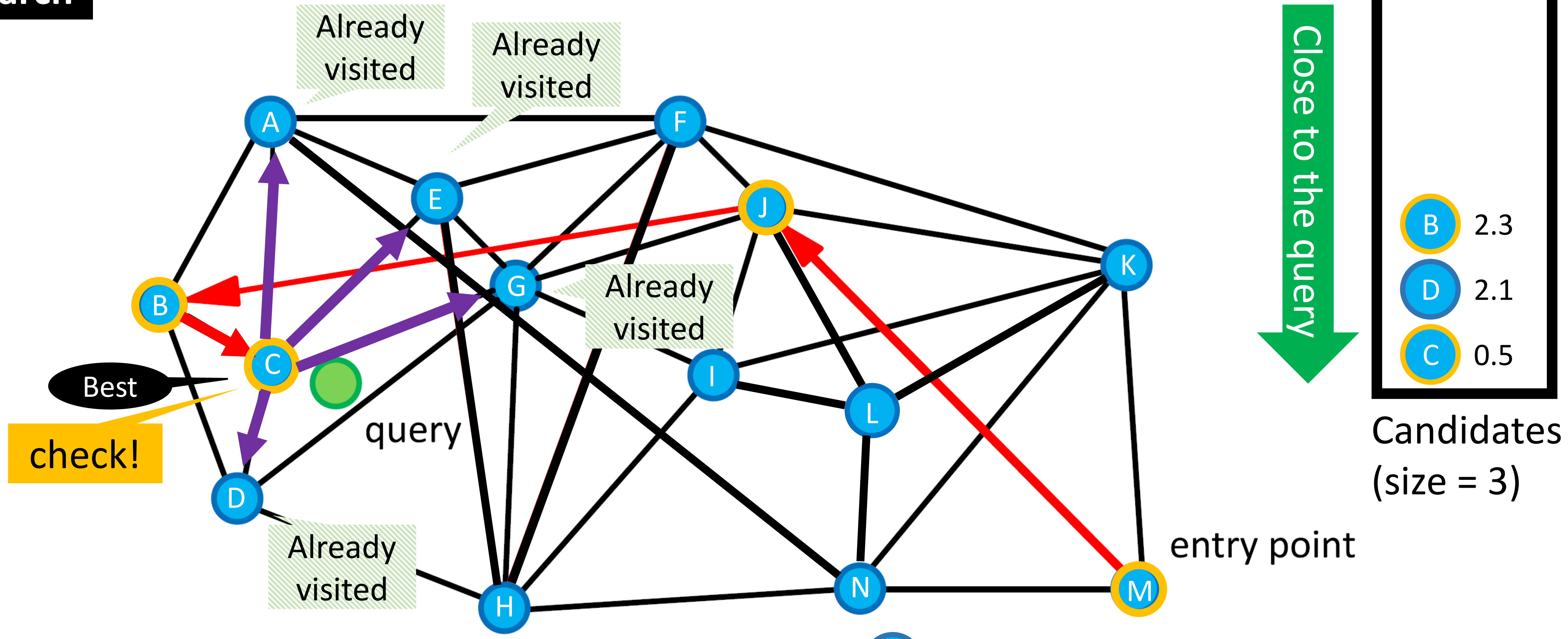
Images are from [Malkov+, Information Systems, 2013]



- ➢ Pick up the unchecked best candidate (C). Check it.
- ➢ Find the connected points.

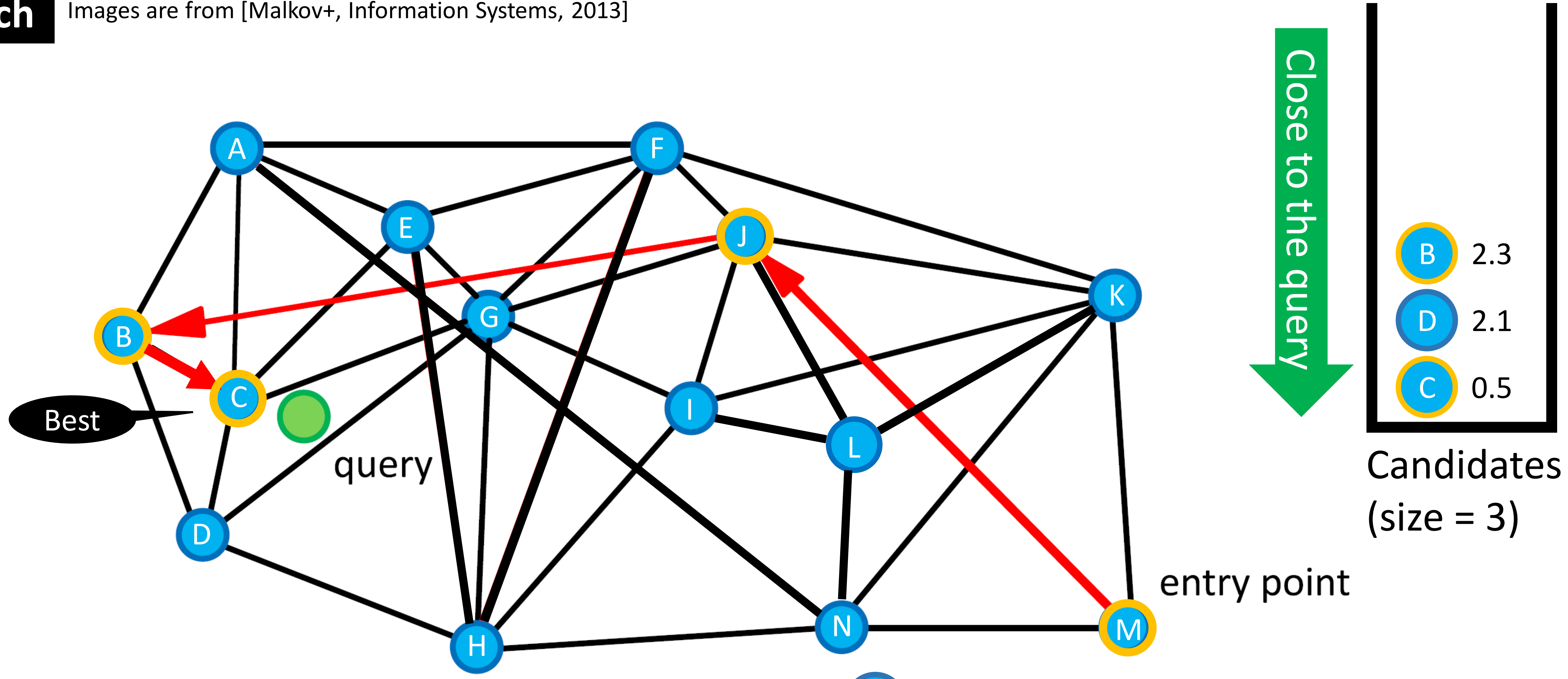
Search

Images are from [Malkov+, Information Systems, 2013]



- ➢ Pick up the unchecked best candidate (C). Check it.
- ➢ Find the connected points.
- ➢ Record the distances to q.

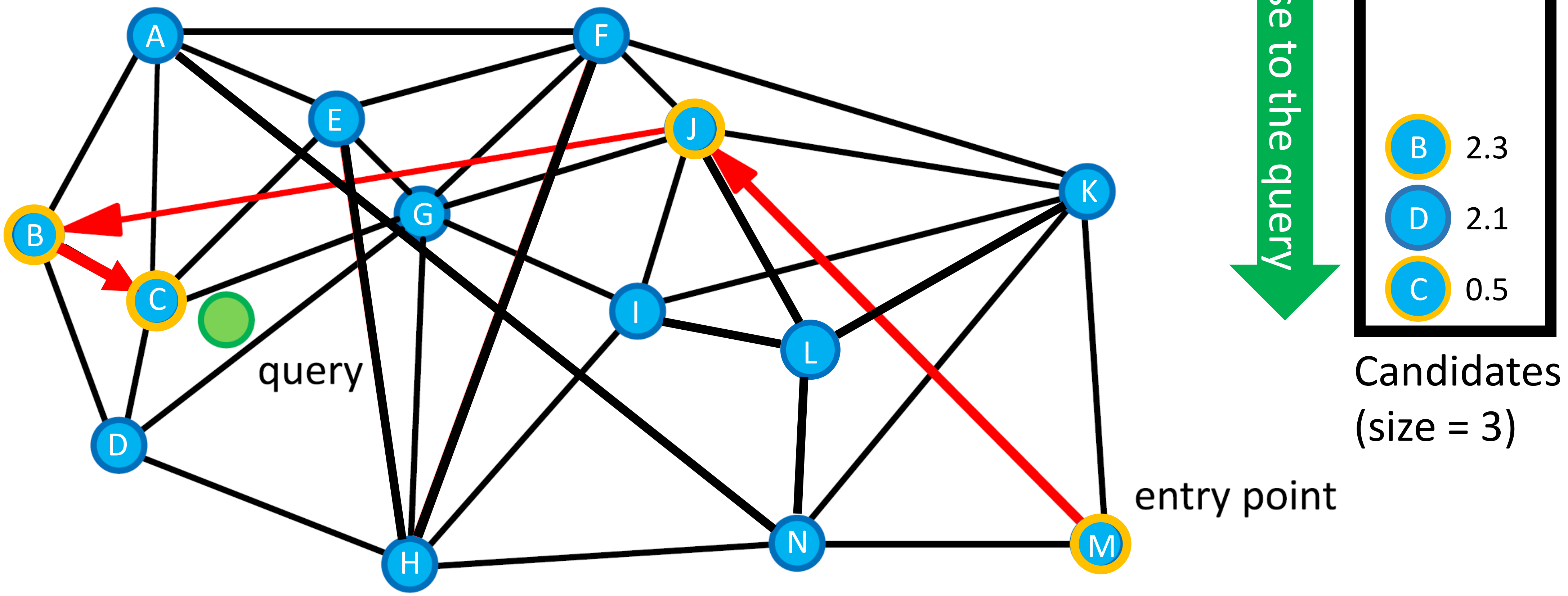
Search Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (C). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

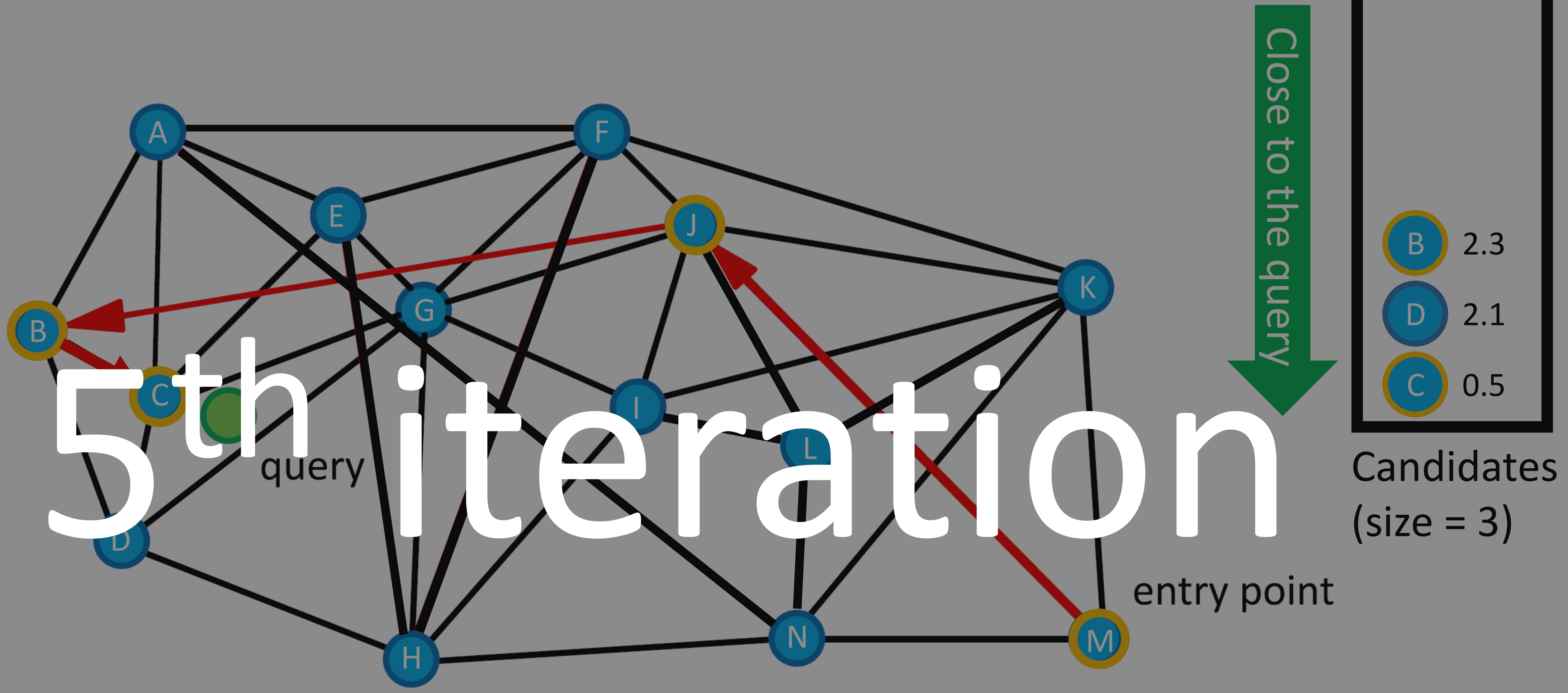
Search

Images are from [Malkov+, Information Systems, 2013]

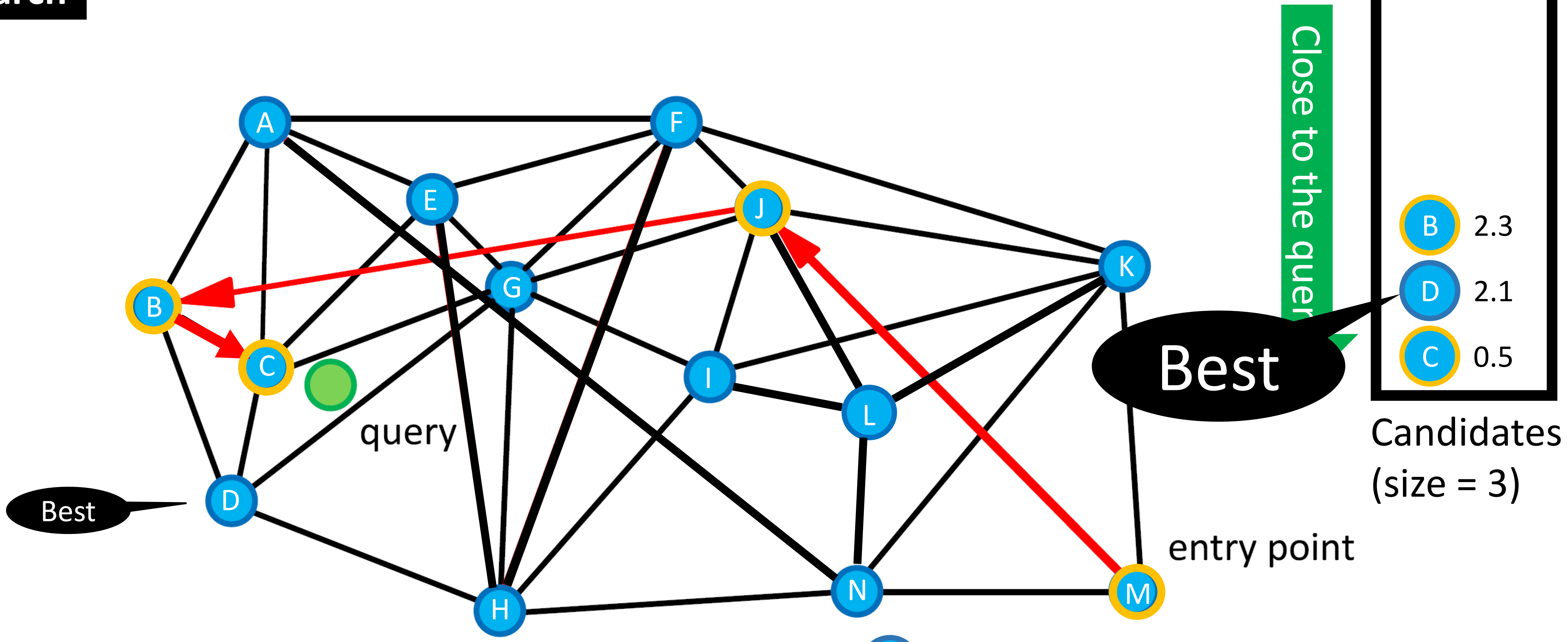


Search

Images are from [Malkov+, Information Systems, 2013]



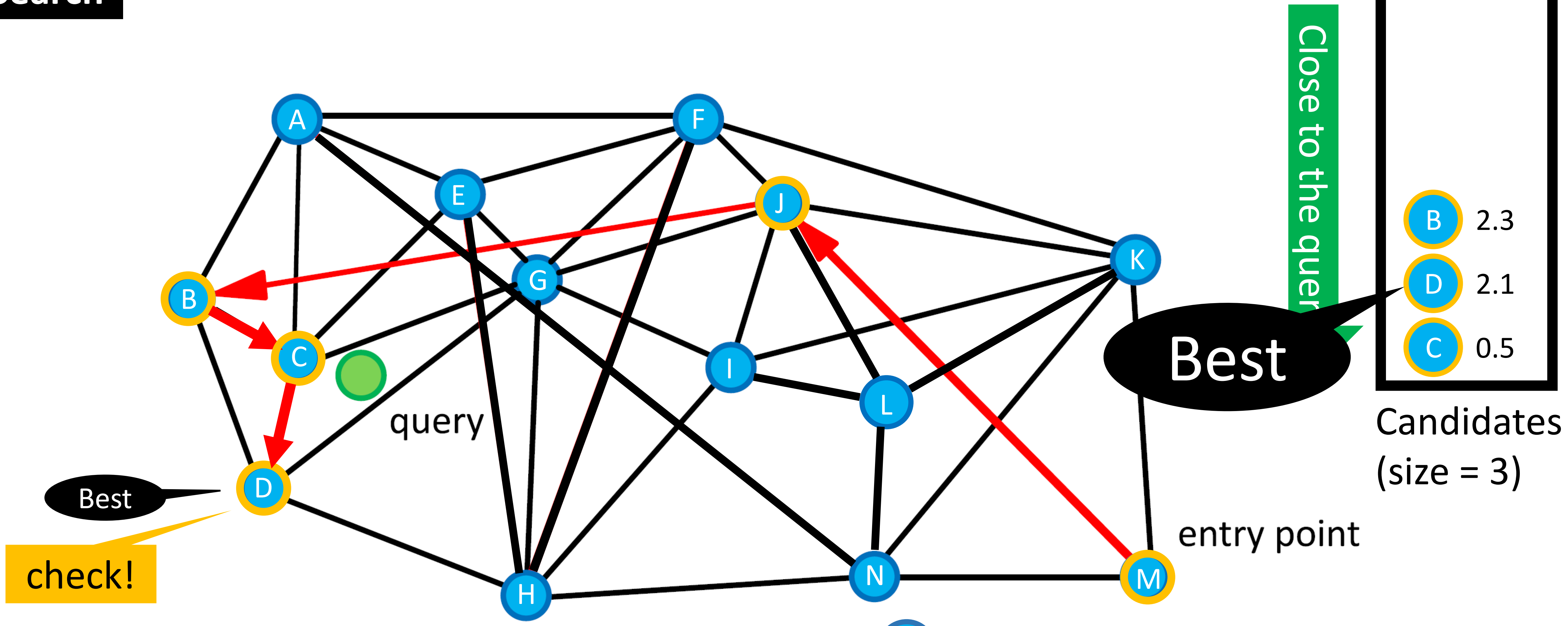
Search Images are from [Malkov+, Information Systems, 2013]



➢ Pick up the unchecked best candidate (D).

Search

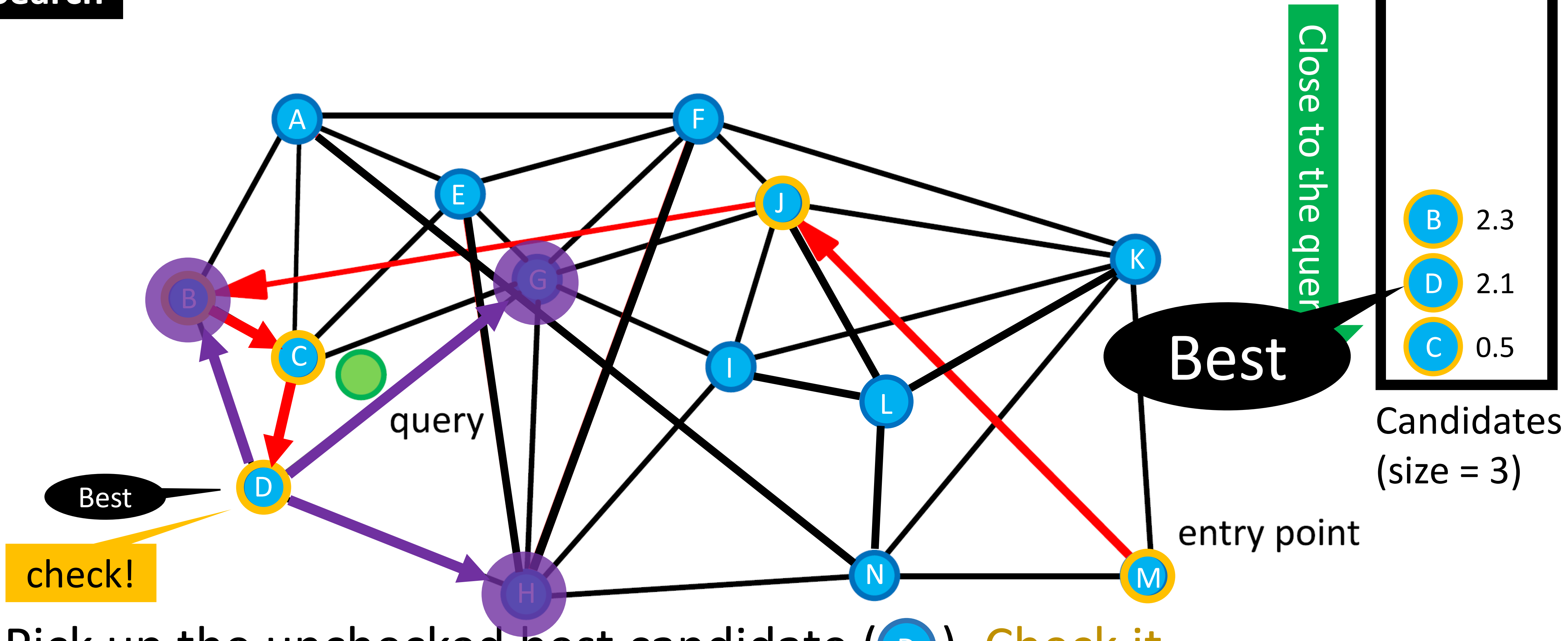
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (D). Check it.

Search

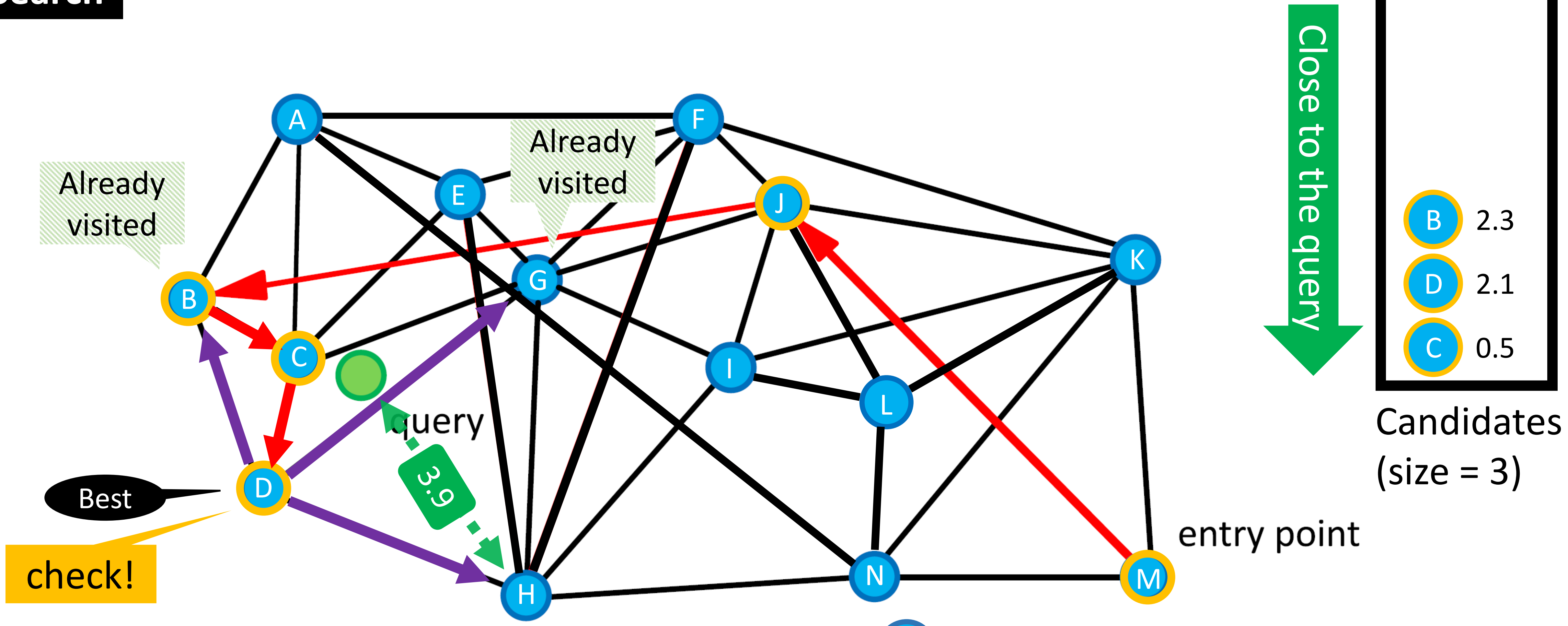
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (D). Check it.
- Find the connected points.

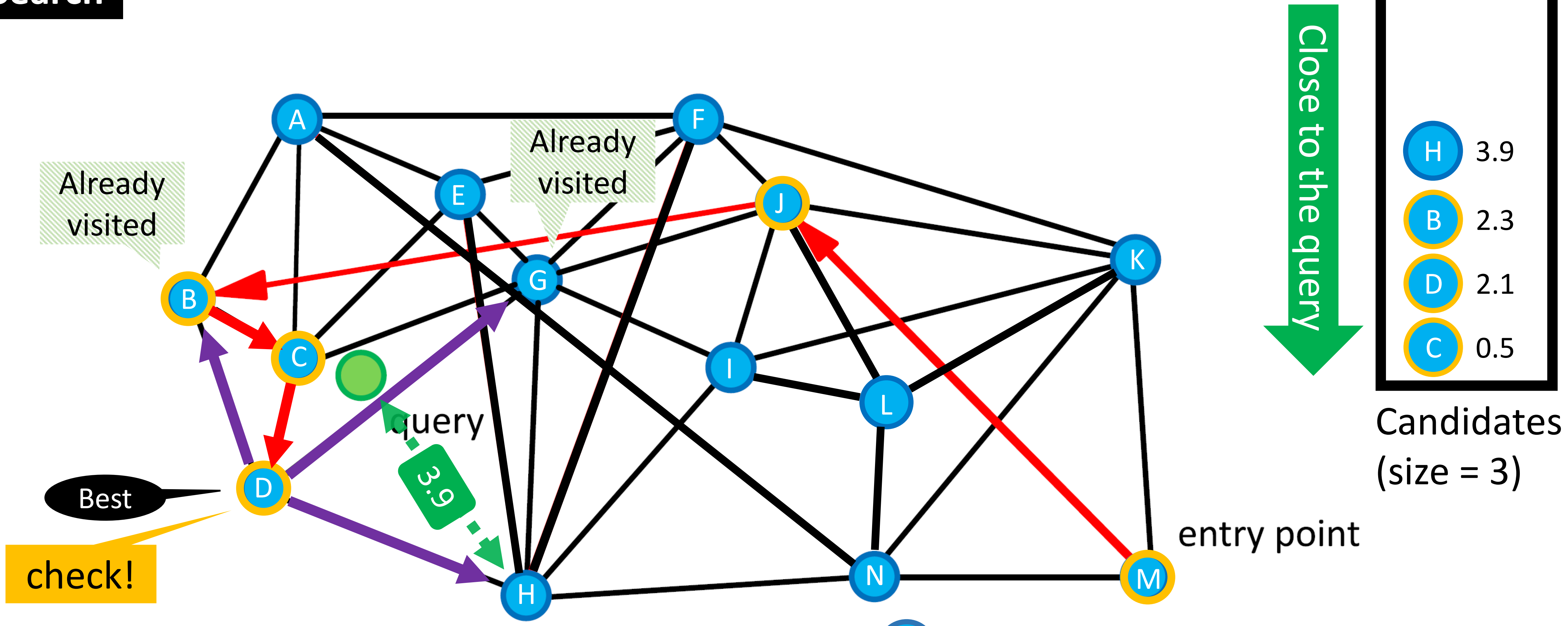
Search

Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (D). Check it.
- Find the connected points.
- Record the distances to q.

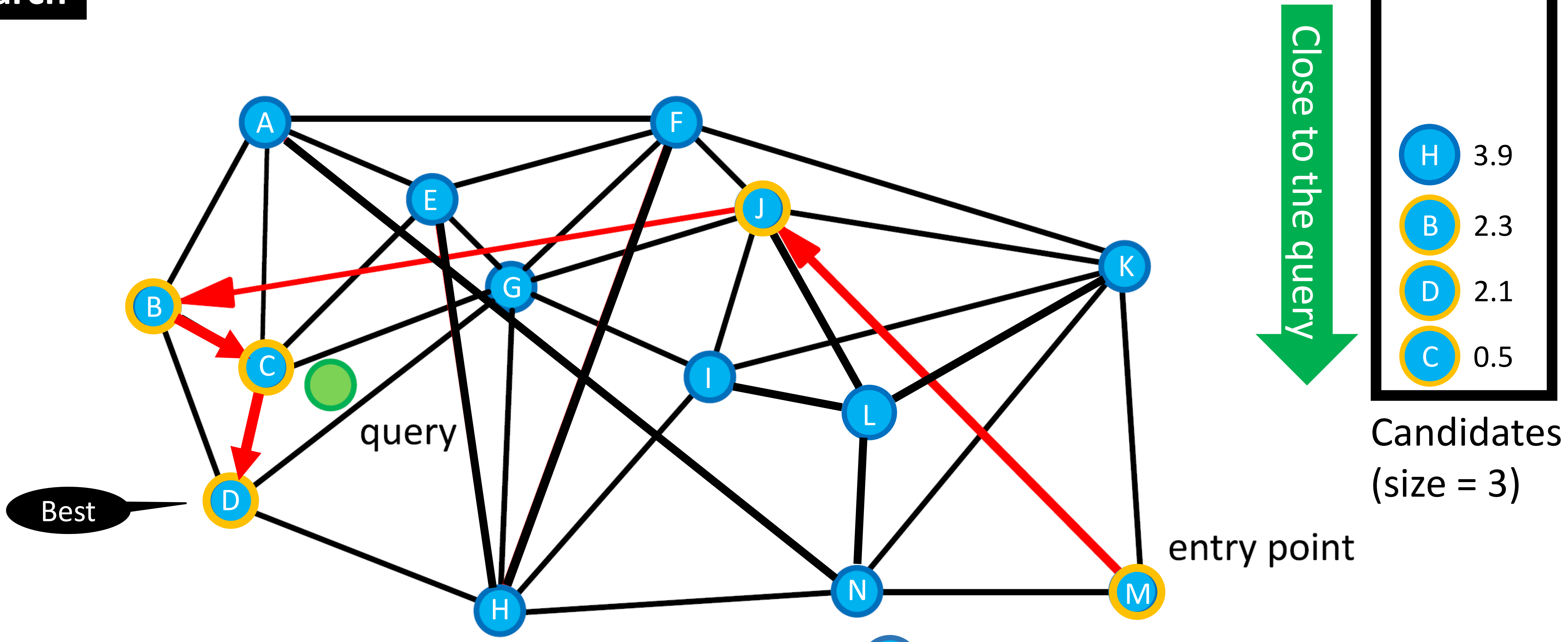
Search Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (D). Check it.
- Find the connected points.
- Record the distances to q.

Search

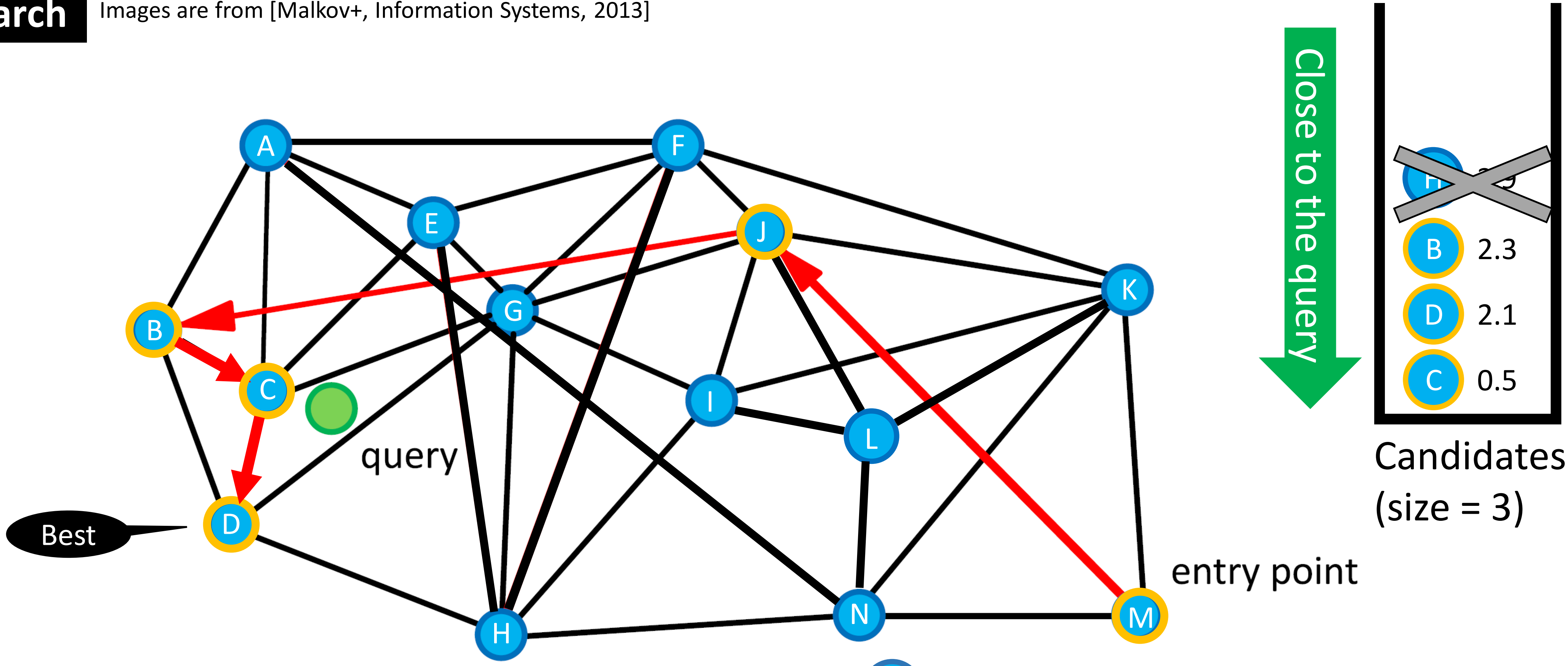
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (D). Check it.
- Find the connected points.
- Record the distances to q.

Search

Images are from [Malkov+, Information Systems, 2013]

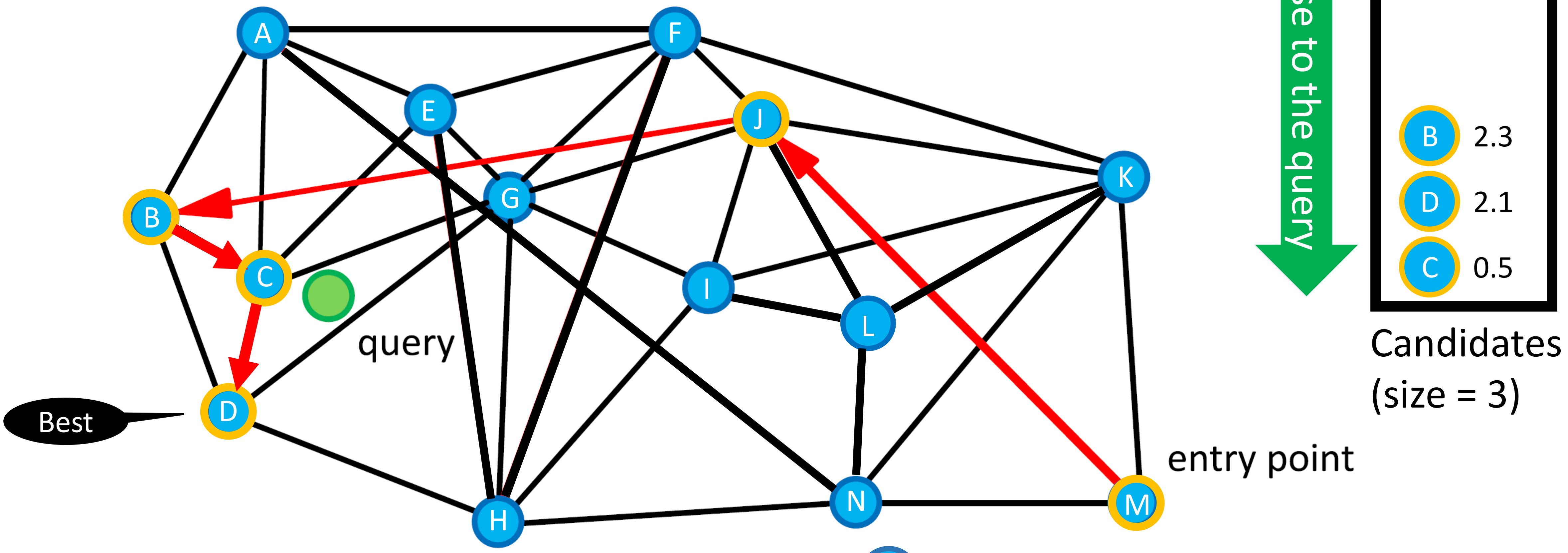


Candidates (size = 3)

- Pick up the unchecked best candidate (D). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

Search

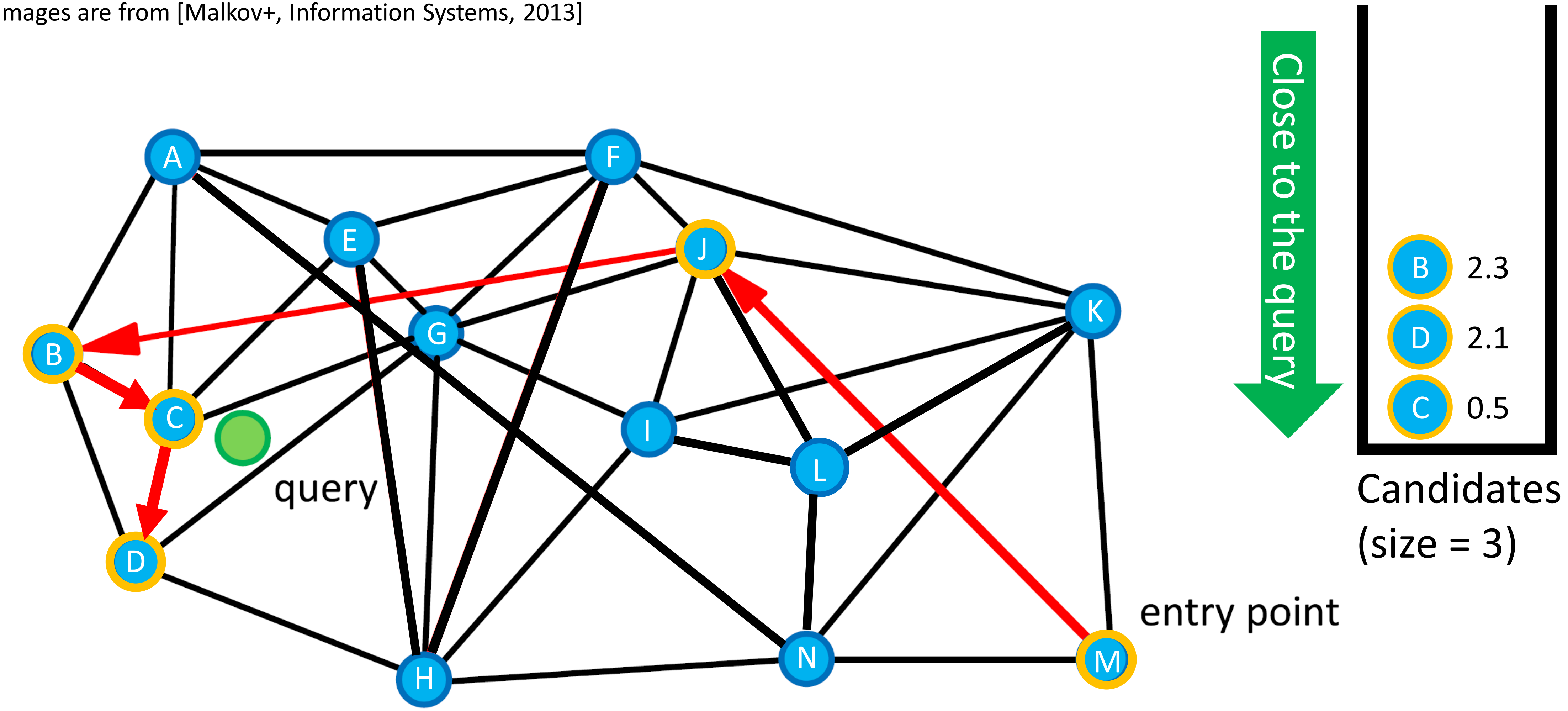
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (D). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

Search

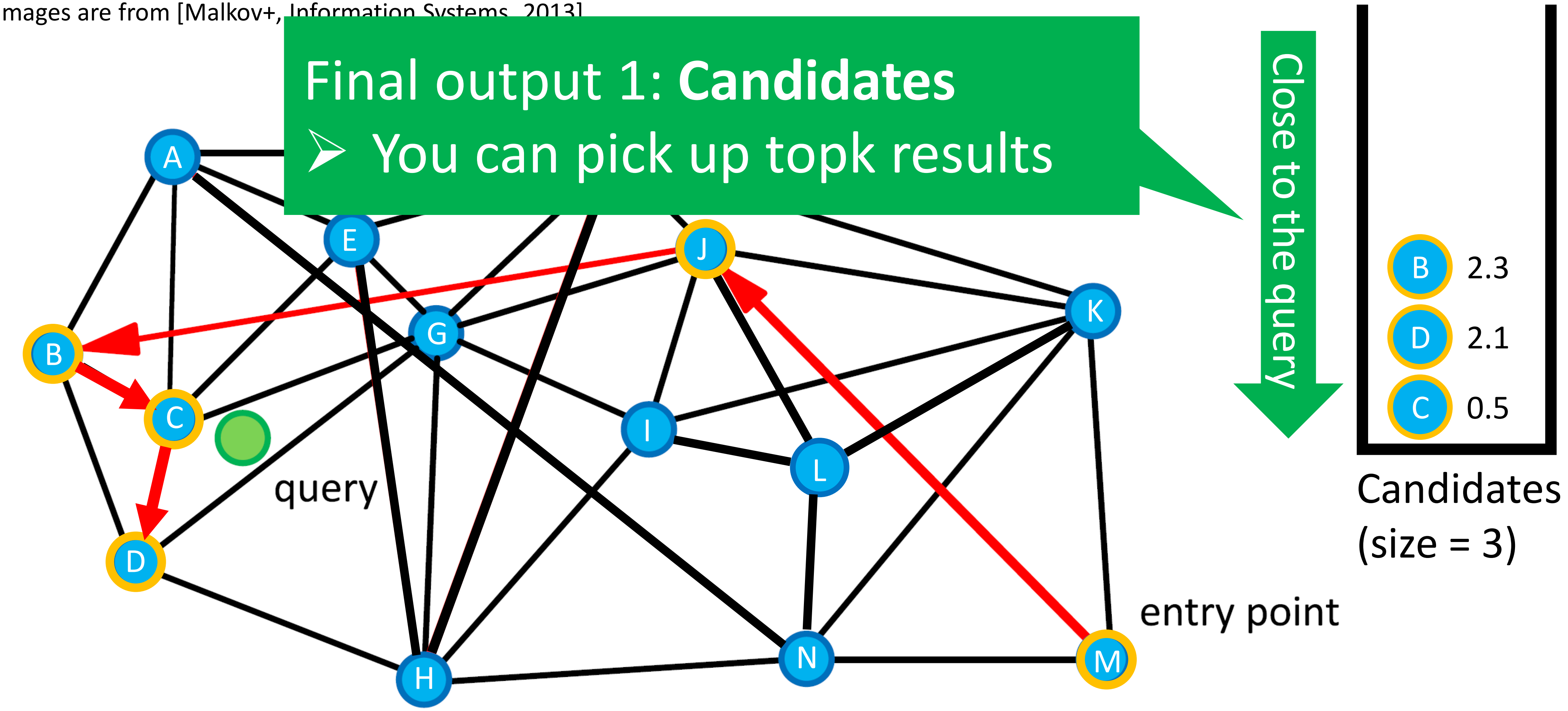
Images are from [Malkov+, Information Systems, 2013]



- All candidates are checked. Finish.
- Here, C is the closet to the query ( )

Search

Images are from [Malkov+, Information Systems, 2013]



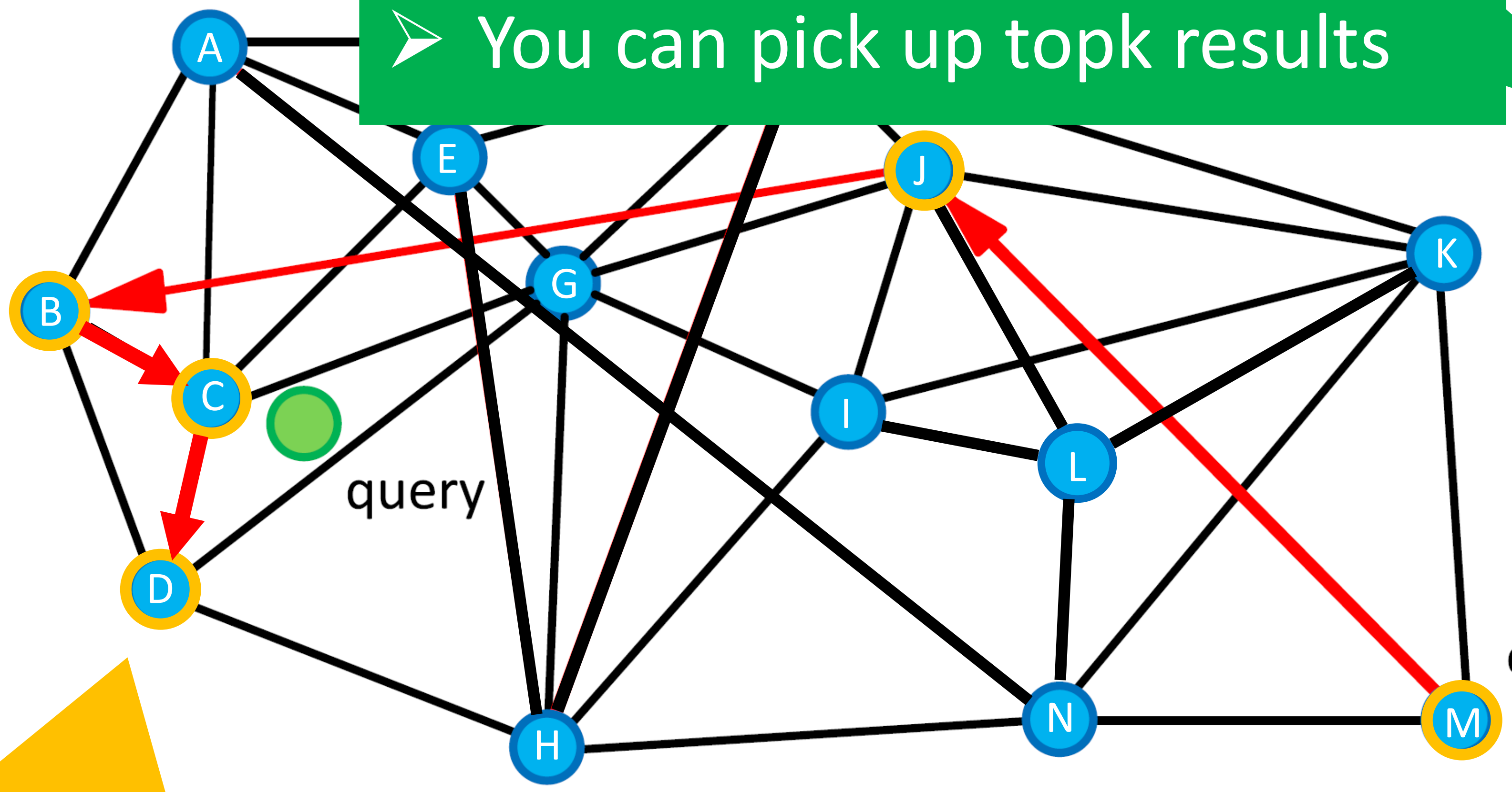
- All candidates are checked. Finish.
- Here, C is the closet to the query ( )

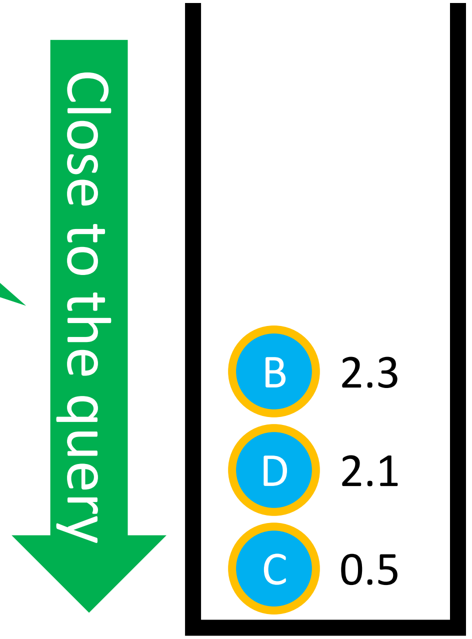
Search

Images are from [Malkov+, Information Systems, 2013]

Final output 1: **Candidates**

- You can pick up topk results





Candidates (size = 3)

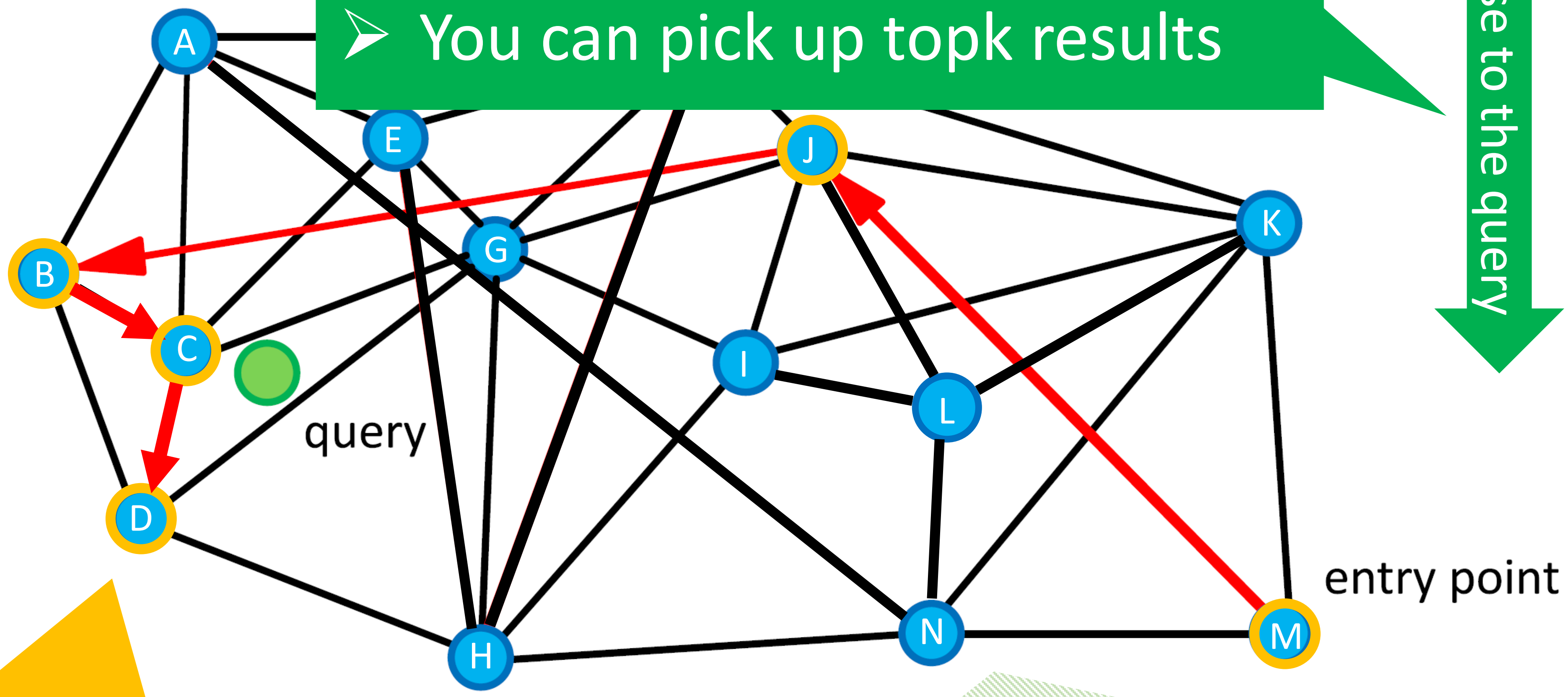
- All ca es are checked. Finish.
- ery ( )

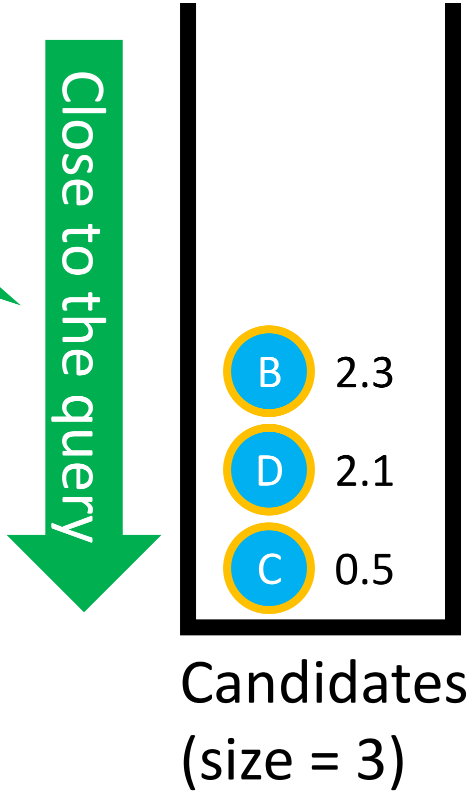
Final output 2: **Checked items**

- i.e., search path

Search Images are from [Malkov+, Information Systems, 2013]

Final output 1: Candidates
- You can pick up topk results





- All ca… es are checked. Finish.
- …ery

Final output 2: Checked items
- i.e., search path

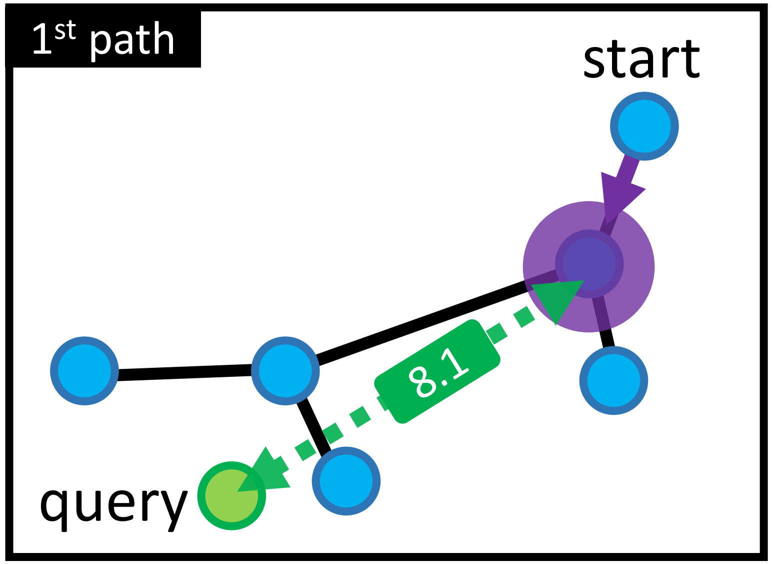
Final output 3: Visit flag
- For each item, visited or not

- Background
- **Graph-based search**
  - Basic (construction and search)
  - **Observation**
  - Properties
- Representative works
  - HNSW, NSG, NGT, Vamana
- Discussion

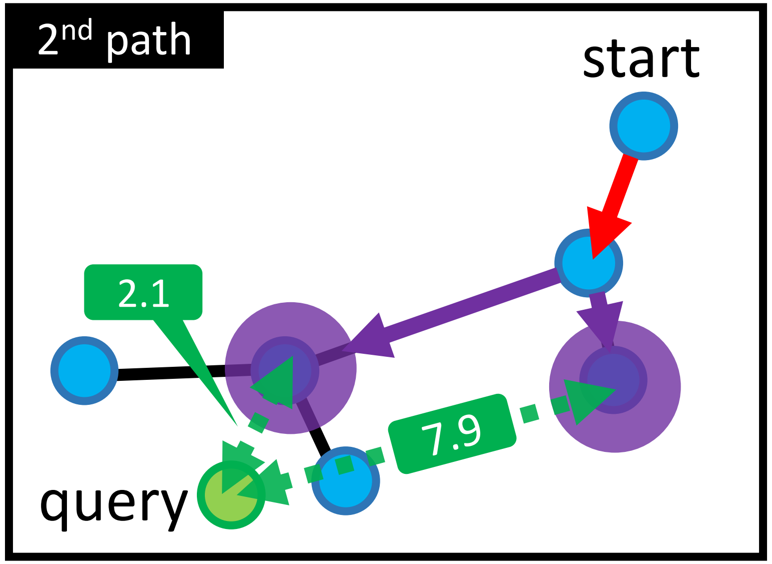
Observation: runtime

➢ Item comparison takes time; $O(D)$

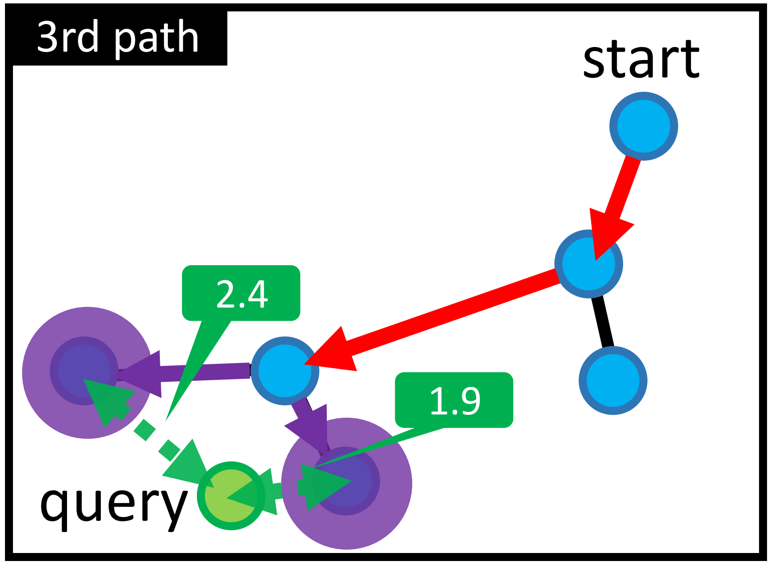
➢ The overall runtime ~ `#item_comparison`
~ `length_of_search_path * average_outdegree`



`outdegree = 1`



`outdegree = 2`



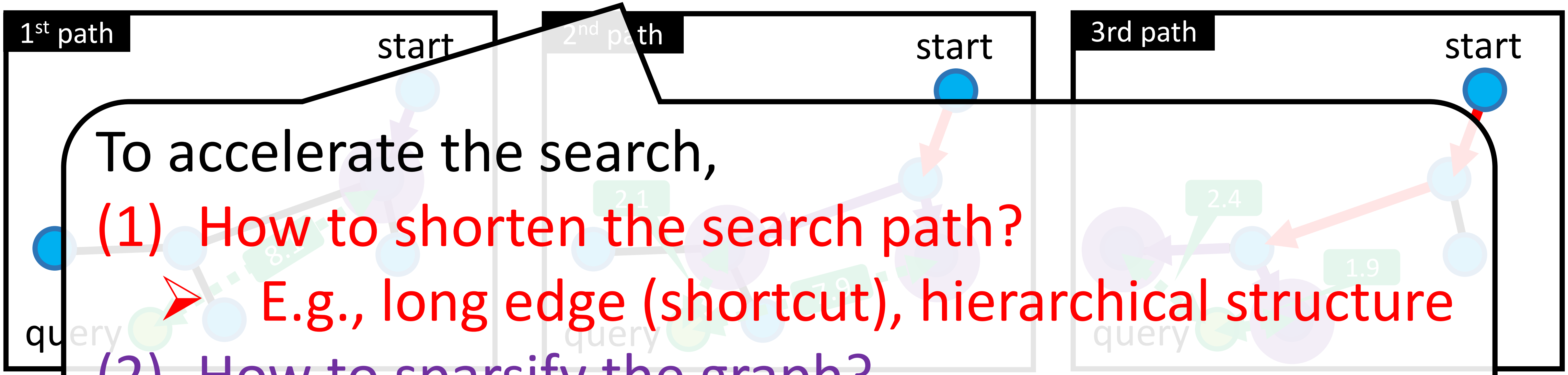
`outdegree = 2`

`#item_comparison = 3 * (1 + 2 + 2)/3 = 5`

Observation: runtime

- Item comparison takes time; $O(D)$

- The overall runtime ~ #item_comparison
  ~ length_of_search_path * average_outdegree



To accelerate the search,

(1) How to shorten the search path?

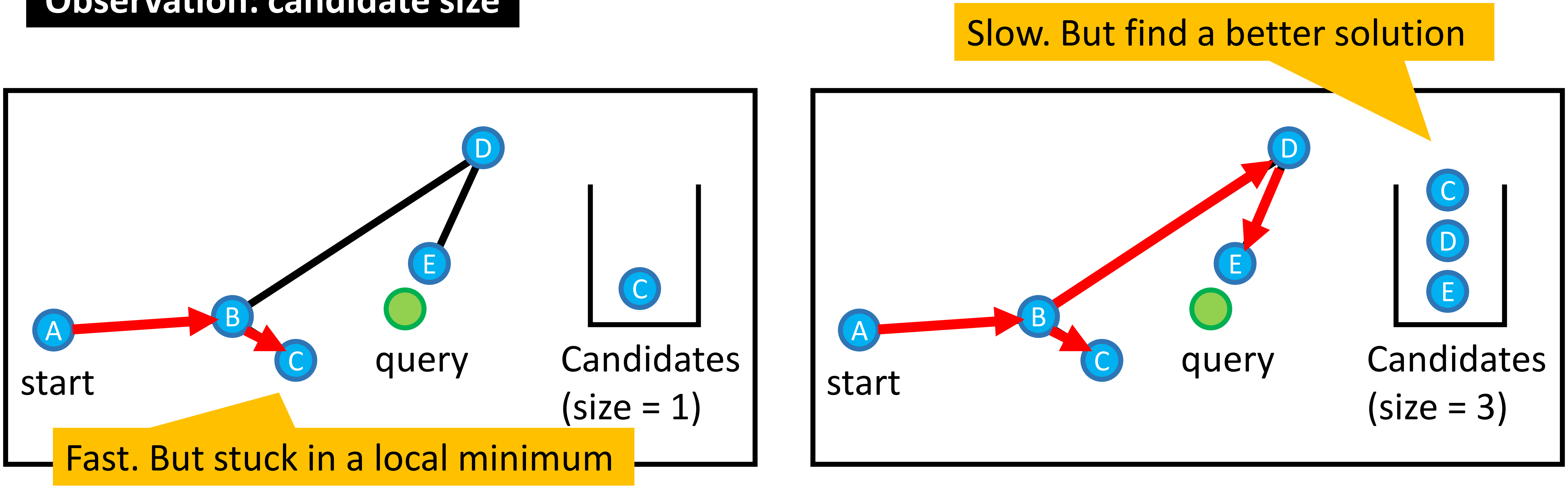
- E.g., long edge (shortcut), hierarchical structure

(2) How to sparsify the graph?

- E.g., deleting redundant edges

#item_comparison = 3 * (1 + 2 + 2)/3 = 5

Observation: candidate size



size = 1: Greedy search

size > 1: Beam search

- Larger candidate size, better but slower results
- Online parameter to control the trade-off
- Called "ef" in HNSW

Pseudo code

Algorithm 1. Search-on-Graph(G, $\mathbf{p}$, $\mathbf{q}$, l)

Require: graph G, start node $\mathbf{p}$, query point $\mathbf{q}$, candidate pool size l
Ensure: k nearest neighbors of $\mathbf{q}$

```
i = 0, candidate pool S = ∅
S.add(p)
while i < l do
    i = the id of the first unchecked node p_i in S
    mark p_i as checked
    for all neighbor n of p_i in G do
      if n has not been visited then
        S.add(n)
      end if
  end for
  sort S in ascending order of the distance to q
  if S.size() > l then
    S.resize(l) // remove nodes from back of S to keep its
    size no larger than l
  end if
end while
return the first k nodes in S
```

NSG [Cong+, VLDB 19]

Algorithm 1: GreedySearch(s, x_q, k, L)

Data: Graph G with start node s, query x_q, result size k, search list size $L \geq k$
Result: Result set $\mathcal{L}$ containing k-approx NNs, and a set $\mathcal{V}$ containing all the visited nodes

begin

```
  initialize sets 𝓛 ← {s} and 𝓥 ← ∅
  while 𝓛 \ 𝓥 ≠ ∅ do
    let p* ← arg min_{p∈𝓛\𝓥} ||x_p − x_q||
    update 𝓛 ← 𝓛 ∪ N_out(p*) and
     𝓥 ← 𝓥 ∪ {p*}
    if |𝓛| > L then
      update 𝓛 to retain closest L
       points to x_q
  return [closest k points from 𝓛; 𝓥]
```

DiskANN [Subramanya+, NeurIPS 19]

Algorithm 1 Beam search

Data: graph G, query q, initial vertex v_0, output size k
Initialization:
V = $\{v_0\}$ // a set of visited vertices
H = $\{v_0 : d(v_0, q)\}$ // a heap of candidates

```
while has runtime budget do
  v_i = SelectNearest(H, q)
  for v̂ ∈ Expand(v_i, G) do
    if v̂ ∉ V then
      V := Add(V, v̂)
      H := Insert(H, v̂, d(v̂, q))
    end
  end
end
return TopK(V, q, k)
```

Learning to route [Baranchuk+, ICML 19]

- All papers have totally different pseudo code ☹
- Principles are the same. But small details are different.
- Hint: Explicitly state the data structure or not

Pseudo code

Candidates are stored in an array

Candidates are stored in a set

```
Algorithm 1. Search-on-Graph(G, p, q, l)
Require: graph G, start node p, query point q, candidate pool size l
Ensure: k nearest neighbors of q
i = 0, candidate pool S = ∅
S.add(p)
while i < l do
    i = the id of the first unchecked node p_i in S
    mark p_i as checked
    for all neighbor n of p_i in G do
        if n has not been visited then
            S.add(n)
        end if
    end for
    sort S in ascending order of the distance to q
    if S.size() > l then
        S.resize(l) // remove nodes from back of S to keep its
        size no larger than l
    end if
end while
return the first k nodes in S
```

NSG [Cong+, VLDB 19]

```
Algorithm 1: GreedySearch(s, x_q, k, L)
Data: Graph G with start node s, query x_q, result
      size k, search list size L ≥ k
Result: Result set ℒ containing k-approx NNs, and
        a set 𝒱 containing all the visited nodes
begin
    initialize sets ℒ ← {s} and 𝒱 ← ∅
    while ℒ \ 𝒱 ≠ ∅ do
        let p* ← arg min_{p∈ℒ\𝒱} ||x_p − x_q||
        update ℒ ← ℒ ∪ N_out(p*) and
          𝒱 ← 𝒱 ∪ {p*}
        if |ℒ| > L then
            update ℒ to retain closest L
            points to x_q
    return [closest k points from ℒ; 𝒱]
```

DiskANN [Subramanya+, NeurIPS 19]

```
Algorithm 1 Beam search
Data: graph G, query q, initial vertex v_0, output size k
Initialization:
V = {v_0} // a set of visited vertices
H = {v_0 : d(v_0, q)} // a heap of candidates
while has runtime budget do
    v_i = SelectNearest(H, q)
    for v̂ ∈ Expand(v_i, G) do
        if v̂ ∉ V then
            V := Add(V, v̂)
            H := Insert(H, v̂, d(v̂, q))
        end
    end
end
return TopK(V, q, k)
```

Learning to route [Baranchuk+, ICML 19]

Sort the array explicitly

When need to sort, say “closest L points”

Candidates are stored in a heap; automatically sorted

- All papers have totally different pseudo code ☹
- Principles are the same. But small details are different.
- Hint: Explicitly state the data structure or not

Pseudo code

Algorithm 1. Search-on-Graph(G, **p**, **q**, l)

Require: graph G, start node **p**, query point **q**, candidate pool size l
Ensure: k nearest neighbors of **q**

```
i = 0, candidate pool S = ∅
S.add(p)
while i < l do
    i = the id of the first unchecked node p_i in S
    mark p_i as checked
    for all neighbor n of p_i in G do
        if n has not been visited then
            S.add(n)
        end if
    end for
    sort S in ascending order of the distance to q
    if S.size() > l then
        S.resize(l) // remove nodes from back of S to keep its
        size no larger than l
    end if
end while
return the first k nodes in S
```

NSG [Cong+, VLDB 19]

Algorithm 1: GreedySearch(s, x_q, k, L)

Data: Graph G with start node s, query x_q, result size k, search list size $L \geq k$
Result: Result set $\mathcal{L}$ containing k-approx NNs, and a set $\mathcal{V}$ containing all the visited nodes

begin

```
initialize sets L ← {s} and V ← ∅
while L \ V ≠ ∅ do
    let p* ← arg min_{p∈L\V} ||x_p − x_q||
    update L ← L ∪ N_out(p*) and
    V ← V ∪ {p*}
    if |L| > L then
        update L to retain closest L
        points to x_q
return [closest k points from L; V]
```

DiskANN [Subramanya+, NeurIPS 19]

Algorithm 1 Beam search

Data: graph G, query q, initial vertex v_0, output size k
Initialization:

```
V = {v_0} // a set of visited vertices
H = {v_0 : d(v_0, q)} // a heap of candidates
while has runtime budget do
    v_i = SelectNearest(H, q)
    for v̂ ∈ Expand(v_i, G) do
        if v̂ ∉ V then
            V := Add(V, v̂)
            H := Insert(H, v̂, d(v̂, q))
        end
    end
end
return TopK(V, q, k)
```

Learning to route [Baranchuk+, ICML 19]

Just "check"

Checked items are stored in a set ("visit" in this code means "check" in our notation)

- All papers have totally different pseudo code ☹
- Principles are the same. But small details are different.
- Hint: Explicitly state the data structure or not

Pseudo code

Algorithm 1. Search-on-Graph(G, **p**, **q**, l)

Require: graph G, start node **p**, query point **q**, candidate pool size l
Ensure: k nearest neighbors of **q**

$i = 0$, candidate pool $S = \emptyset$
S.add(p)
while $i < l$ **do**
  i = the id of the first unchecked node p_i in S
  mark $\mathbf{p_i}$ as checked
  for all neighbor **n** of $\mathbf{p_i}$ in G **do**
    if n has not been visited **then**
      S.add(n)
    end if
  end for
  sort S in ascending order of the distance to **q**
  if S.size() $> l$ **then**
    S.resize(l) // remove nodes from back of S to keep its size no larger than l
  end if
end while
return the first k nodes in S

NSG [Cong+, VLDB 19]

Algorithm 1: GreedySearch(s, x_q, k, L)

Data: Graph G with start node s, query x_q, result size k, search list size $L \geq k$
Result: Result set $\mathcal{L}$ containing k-approx NNs, and a set $\mathcal{V}$ containing all the visited nodes
begin
  initialize sets $\mathcal{L} \leftarrow \{s\}$ and $\mathcal{V} \leftarrow \emptyset$
  while $\mathcal{L} \setminus \mathcal{V} \neq \emptyset$ **do**
    let $p* \leftarrow \arg\min_{p \in \mathcal{L} \setminus \mathcal{V}} ||\mathsf{x}_p - \mathsf{x}_q||$
    update $\mathcal{L} \leftarrow \mathcal{L} \cup N_{\text{out}}(p^*)$ and $\mathcal{V} \leftarrow \mathcal{V} \cup \{p^*\}$
    if $|\mathcal{L}| > L$ **then**
      update $\mathcal{L}$ to retain closest L points to x_q
  return [closest k points from $\mathcal{L}$; $\mathcal{V}$]

DiskANN [Subramanya+, NeurIPS 19]

Algorithm 1 Beam search

Data: graph G, query q, initial vertex v_0, output size k
Initialization:
V = $\{v_0\}$ // a set of visited vertices
H = $\{v_0 : d(v_0, q)\}$ // a heap of candidates
while *has runtime budget* **do**
  v_i = SelectNearest(H, q)
  for $\hat{v} \in Expand(v_i, G)$ **do**
    if $\hat{v} \notin V$ **then**
      $V := Add(V, \hat{v})$
      $H := Insert(H, \hat{v}, d(\hat{v}, q))$
    end
  end
end
return TopK(V, q, k)

Learning to route [Baranchuk+, ICML 19]

Visited item are simply said to be "visited"; implying an additional hidden data structure (array)

Visited items are stored in a set

- All papers have totally different pseudo code ☹
- Principles are the same. But small details are different.
- Hint: Explicitly state the data structure or not

Pseudo code

Algorithm 1. Search-on-Graph(G, **p**, **q**, l)

```
Require: graph G, start node p, query point q, candidate pool
         size l
Ensure: k nearest neighbors of q
i = 0, candidate pool S = ∅
S.add(p)
while i < l do
    i = the id of the first unchecked node p_i in S
    mark p_i as checked
    for all neighbor n of p_i in G do
      if n has not been visited then
        S.add(n)
      end if
  end for
  sort S in ascending order of the distance to q
  if S.size() > l then
    S.resize(l) / / remove nodes from back of S
    size no larger than l
  end if
end while
return the first k nodes in S
```

NSG [Cong+, VLDB 19]

Algorithm 1: GreedySearch(s, x_q, k, L)

```
Data: Graph G with start node s, query x_q, result
      size k, search list size L ≥ k
Result: Result set 𝓛 containing k-approx NNs, and
        a set 𝓥 containing all the visited nodes
begin
  initialize sets 𝓛 ← {s} and 𝓥 ← ∅
  while 𝓛 \ 𝓥 ≠ ∅ do
    let p* ← arg min_{p∈𝓛\𝓥} ||x_p − x_q||
    update 𝓛 ← 𝓛 ∪ N_out(p*) and
     𝓥 ← 𝓥 ∪ {p*}
    if |𝓛| > L then
      update 𝓛 to retain closest L
  ...
  return [...; 𝓥]
```

...PS 19]

Termination condition??

Algorithm 1 Beam search

```
Data: graph G, query q, initial vertex v_0, output size k
Initialization:
V = {v_0} // a set of visited vertices
H = {v_0 : d(v_0, q)} // a heap of candidates
while has runtime budget do
  v_i = SelectNearest(H, q)
  for v̂ ∈ Expand(v_i, G) do
    if v̂ ∉ V then
      V := Add(V, v̂)
      H := Insert(H, v̂, d(v̂, q))
    end
  end
end
return TopK(V, q, k)
```

Learning to route [Baranchuk+, ICML 19]

- All papers have totally different pseudo code ☹
- Principles are the same. But small details are different.
- Hint: Explicitly state the data structure or not

Pseudo code

Algorithm 1. Search-on-Graph(G, $\mathbf{p}$, $\mathbf{q}$, l)

Require: graph G, start node $\mathbf{p}$, query point $\mathbf{q}$, candidate pool size l
Ensure: k nearest neighbors of $\mathbf{q}$

$i = 0$, candidate pool $S = \emptyset$
S.add($\mathbf{p}$)
while $i < l$ **do**
  i = the id of the first unchecked node p_i in S
  mark $\mathbf{p_i}$ as checked
  for all neighbor $\mathbf{n}$ of $\mathbf{p_i}$ in G **do**
    if $\mathbf{n}$ has not been visited **then**
      S.add($\mathbf{n}$)
    end if
  end for
  sort S in ascending order of the distance to $\mathbf{q}$
  if S.size() $> l$ **then**
    S.resize(l) / / remove nodes from back of S to keep its size no larger than l
  end if
end while
return the first k nodes in S

NSG [Cong+, VLDB 19]

Algorithm 1: GreedySearch(s, x_q, k, L)

Data: Graph G with start node s, query x_q, result size k, search list size $L \geq k$
Result: Result set $\mathcal{L}$ containing k-approx NNs, and a set $\mathcal{V}$ containing all the visited nodes
begin
  initialize sets $\mathcal{L} \leftarrow \{s\}$ and $\mathcal{V} \leftarrow \emptyset$
  while $\mathcal{L} \setminus \mathcal{V} \neq \emptyset$ **do**
    let $p* \leftarrow \arg\min_{p \in \mathcal{L} \setminus \mathcal{V}} ||\mathrm{x}_p - \mathrm{x}_q||$
    update $\mathcal{L} \leftarrow \mathcal{L} \cup N_{\mathrm{out}}(p^*)$ and $\mathcal{V} \leftarrow \mathcal{V} \cup \{p^*\}$
    if $|\mathcal{L}| > L$ **then**
      update $\mathcal{L}$ to retain closest L points to x_q
  return [closest k points from $\mathcal{L}$; $\mathcal{V}$]

DiskANN [Subramanya+, NeurIPS 19]

Algorithm 1 Beam search

Data: graph G, query q, initial vertex v_0, output size k
Initialization:
V = $\{v_0\}$ // a set of visited vertices
H = $\{v_0 : d(v_0, q)\}$ // a heap of candidates
while *has runtime budget* **do**
  v_i = SelectNearest(H, q)
  for $\hat{v} \in Expand(v_i, G)$ **do**
    if $\hat{v} \notin V$ **then**
      $V := Add(V, \hat{v})$
      $H := Insert(H, \hat{v}, d(\hat{v}, q))$
    end
  end
end
return TopK(V, q, k)

Learning to route [Baranchuk+, ICML 19]

- All papers have totally different pseudo code ☹

My explanation was based on NSG, but with slight modifications for simplicity:
- Candidates are stored in an automatically-sorted array
- Termination condition is "all candidates are checked"

Pseudo code

Algorithm 1. Search-on-Graph(G, **p**, **q**, l)

```
Require: graph G, start node p, query point q, candidate pool size l
Ensure: k nearest neighbors of q
i = 0, candidate pool S = ∅
S.add(p)
while i < l do
    i = the id of the first unchecked node p_i in S
    mark p_i as checked
    for all neighbor n of p_i in G do
        if n has not been visited then
            S.add(n)
        end if
    end for
    sort S in ascending order of the distance to q
    if S.size() > l then
        S.resize(l) // remove nodes from back of S to keep its size no larger than l
    end if
end while
return the first k nodes in S
```

NSG [Cong+, VLDB 19]

Algorithm 1: GreedySearch(s, x_q, k, L)

```
Data: Graph G with start node s, query x_q, result size k, search list size L ≥ k
Result: Result set ℒ containing k-approx NNs, and a set 𝒱 containing all the visited nodes
begin
    initialize sets ℒ ← {s} and 𝒱 ← ∅
    while ℒ \ 𝒱 ≠ ∅ do
        let p* ← arg min_{p∈ℒ\𝒱} ||x_p − x_q||
        update ℒ ← ℒ ∪ N_out(p*) and 𝒱 ← 𝒱 ∪ {p*}
            update ℒ to retain closest L points to x_q
    return [closest k points from ℒ; 𝒱]
```

DiskANN [Subramanya+, NeurIPS 19]

Algorithm 1 Beam search

```
Data: graph G, query q, initial vertex v_0, output size k
Initialization:
V = {v_0} // a set of visited vertices
H = {v_0 : d(v_0, q)} // a heap of candidates
while has runtime budget do
    v_i = SelectNearest(H, q)
    for v̂ ∈ Expand(v_i, G) do
        if v̂ ∉ V then
            V := Add(V, v̂)
            H := Insert(H, v̂, d(v̂, q))
        end
    end
end
return TopK(V, q, k)
```

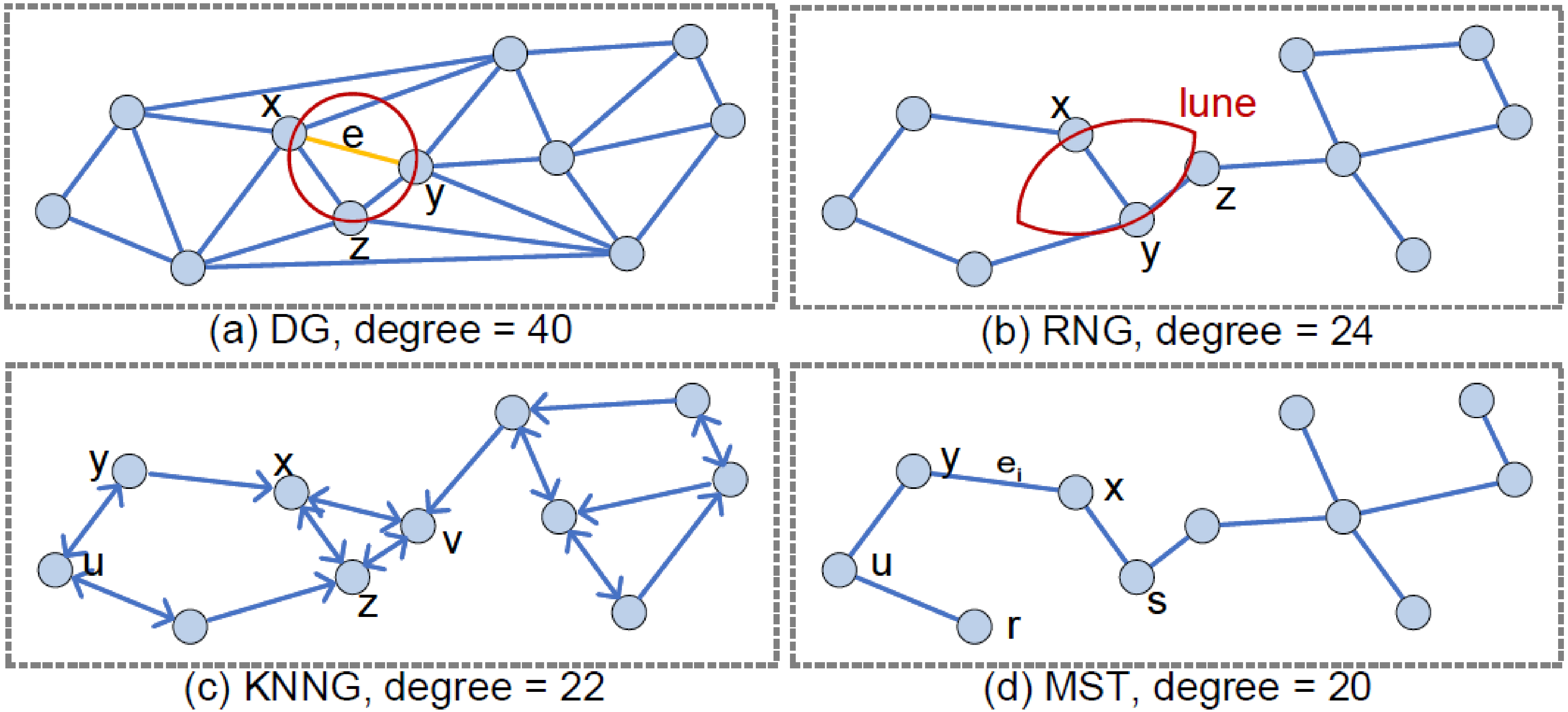
Learning to route [Baranchuk+, ICML 19]

Formal (?) definition would be helpful for everyone

- All papers have totally different pseudo code ☹
- Principles are the same. But small parts are very different
- Hint: Explicitly state the data structure or not

- Background
- Graph-based search
  - Basic (construction and search)
  - Observation
  - Properties
- Representative works
  - HNSW, NSG, NGT, Vamana
- Discussion

Base graph



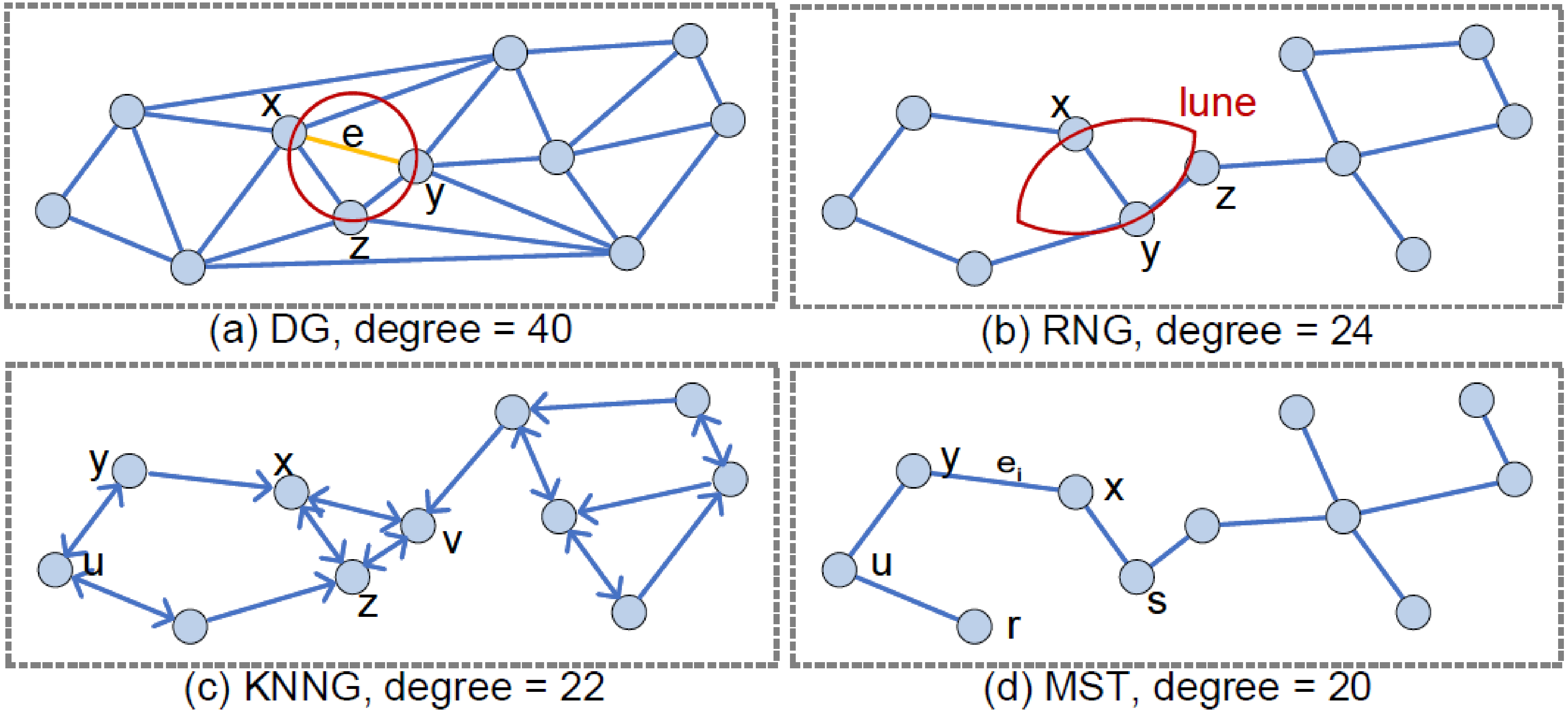
- Although there are many graph algorithms, there exists four base graphs.
- These base graphs are (1) slow to be constructed, and (2) often too dense
- Each algorithm often improves one of the base graphs

Images are from an excellent survey paper [Wang+, VLDB 2021]

Base graph

Principal:

- Not too dense: Search is slow for dense graph
- But moderately dense: Each points should be reachable

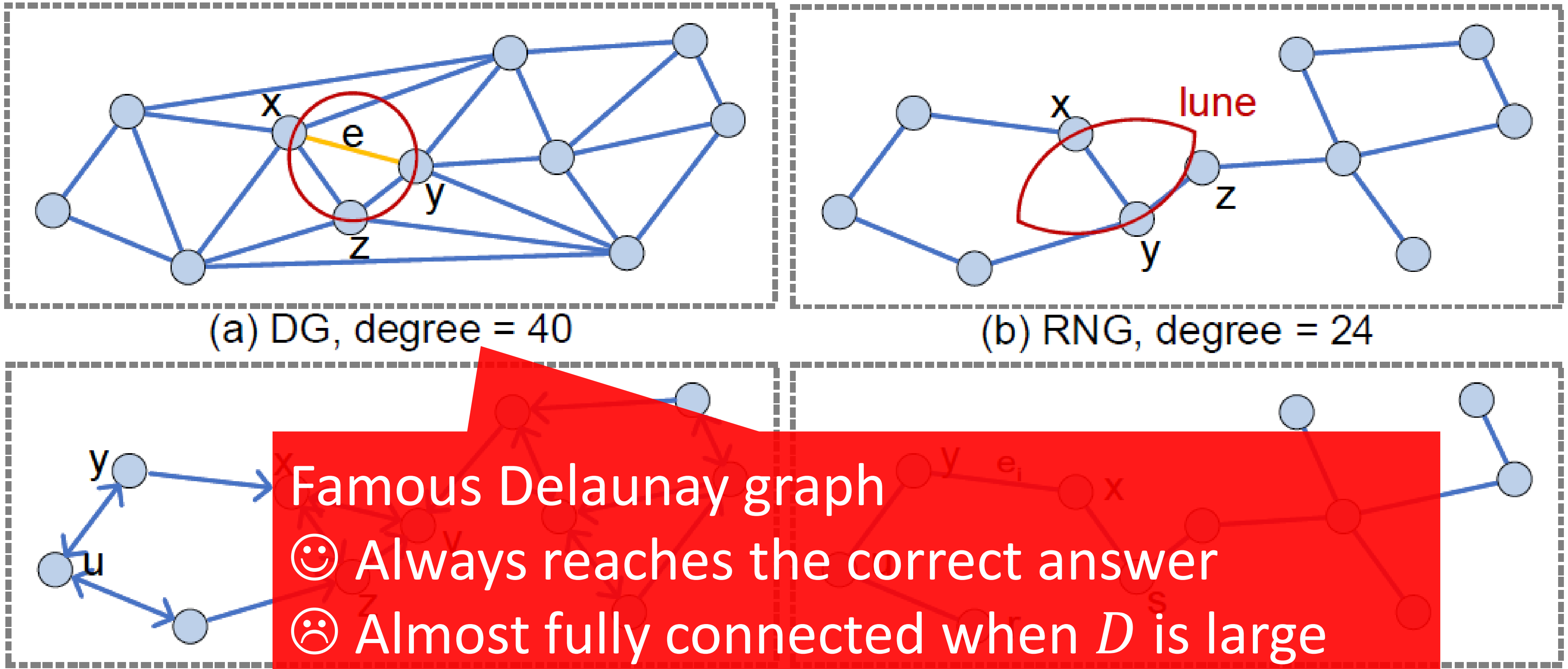


- Although there are many graph algorithms, there exists four base graphs.
- These base graphs are (1) slow to be constructed, and (2) often too dense
- Each algorithm often improves one of the base graphs

Images are from an excellent survey paper [Wang+, VLDB 2021]

Base graph

Principal:
- Not too dense: Search is slow for dense graph
- But moderately dense: Each points should be reachable



(a) DG, degree = 40

(b) RNG, degree = 24

(c) KNNG, degree = 22

(d) MST, degree = 20

Famous Delaunay graph

☺ Always reaches the correct answer

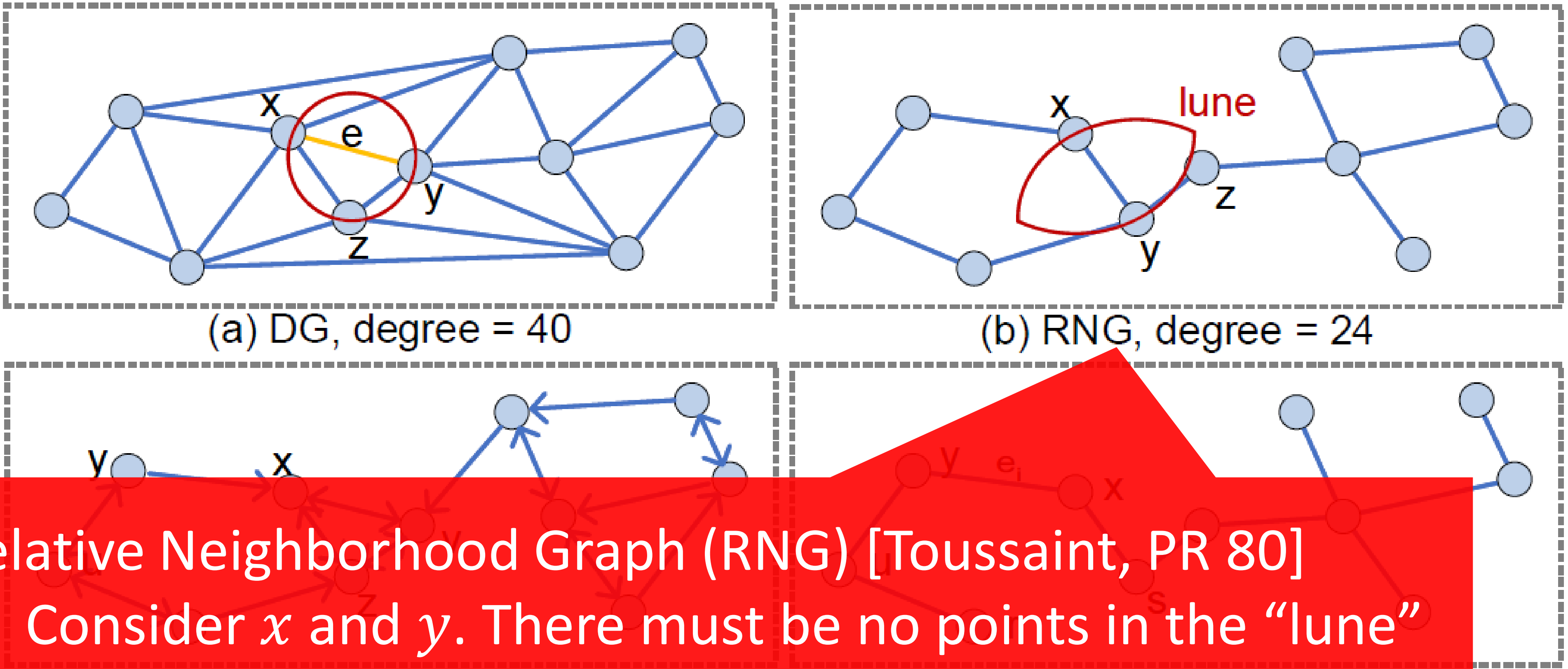
☹ Almost fully connected when D is large

- Although there are many graph algorithms, there exists four base graphs.
- These base graphs are (1) slow to be constructed, and (2) often too dense
- Each algorithm often improves one of the base graphs

Images are from an excellent survey paper [Wang+, VLDB 2021]

Base graph

Principal:
- Not too dense: Search is slow for dense graph
- But moderately dense: Each points should be reachable



(a) DG, degree = 40

(b) RNG, degree = 24

(c) KNNG, degree = 22

(d) MST, degree = 20

Relative Neighborhood Graph (RNG) [Toussaint, PR 80]
- Consider x and y. There must be no points in the “lune”
- Can cut off redundant edges
- Not famous in general, but widely used in ANN
- Will review again later

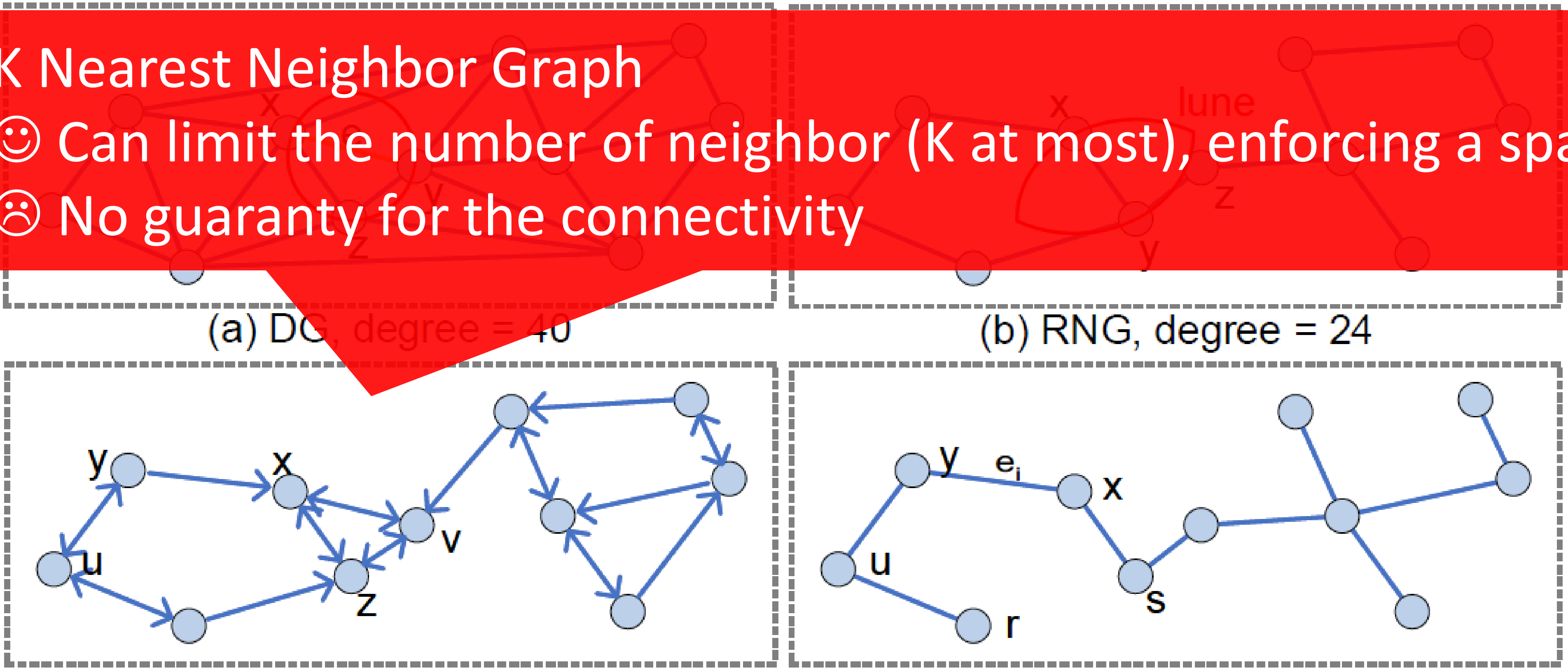
- Although there are many graph algorithms, there exists four base graphs.
- These base graphs are (1) slow to be constructed, and (2) often too dense
- Each algorithm often improves one of the base graphs

Images are from an excellent survey paper [Wang+, VLDB 2021]

Base graph

Principal:
- Not too dense: Search is slow for dense graph
- But moderately dense: Each points should be reachable

K Nearest Neighbor Graph
☺ Can limit the number of neighbor (K at most), enforcing a sparsity
☹ No guaranty for the connectivity



(a) DG, degree = 40 (b) RNG, degree = 24

(c) KNNG, degree = 22 (d) MST, degree = 20

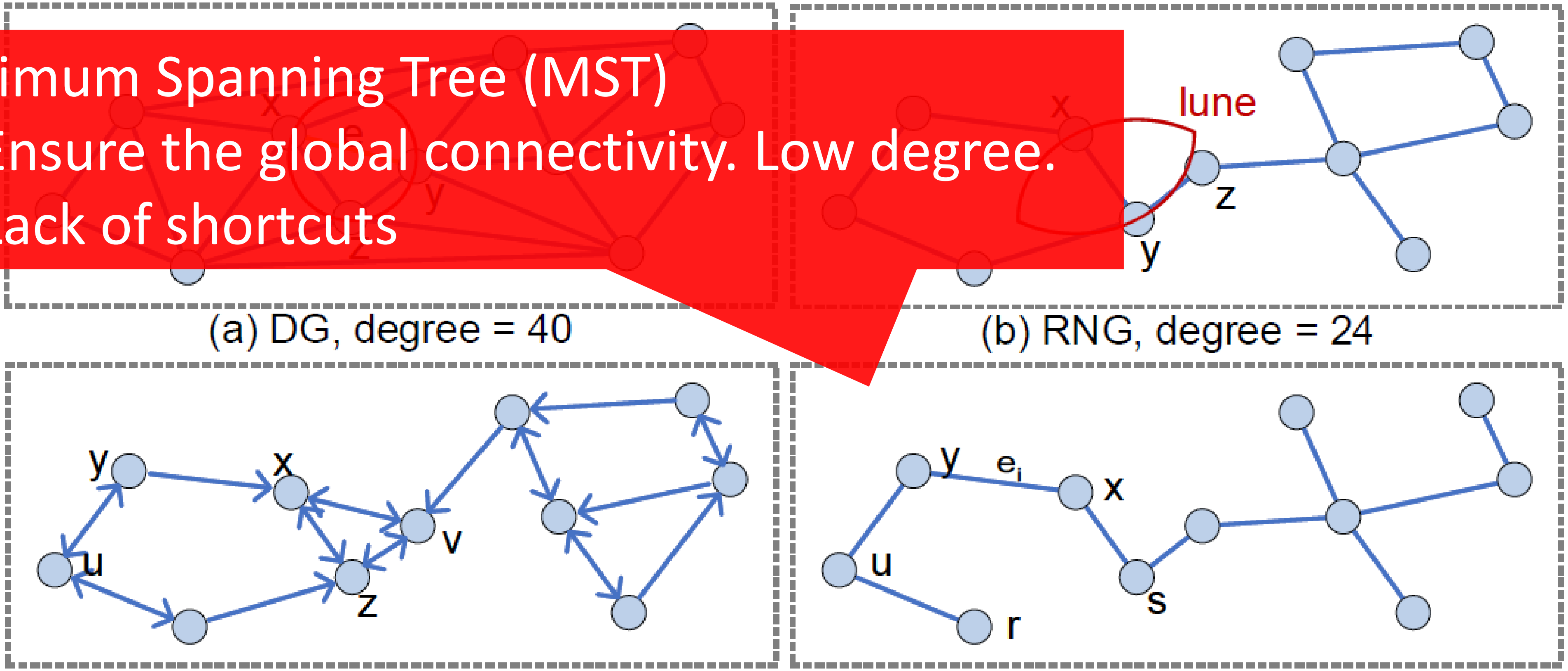
- Although there are many graph algorithms, there exists four base graphs.
- These base graphs are (1) slow to be constructed, and (2) often too dense
- Each algorithm often improves one of the base graphs

Images are from an excellent survey paper [Wang+, VLDB 2021]

Base graph

Principal:
- ➢ Not too dense: Search is slow for dense graph
- ➢ But moderately dense: Each points should be reachable

Minimum Spanning Tree (MST)
☺ Ensure the global connectivity. Low degree.
☹ Lack of shortcuts



(a) DG, degree = 40
(b) RNG, degree = 24
(c) KNNG, degree = 22
(d) MST, degree = 20

- ➢ Although there are many graph algorithms, there exists four base graphs.
- ➢ These base graphs are (1) slow to be constructed, and (2) often too dense
- ➢ Each algorithm often improves one of the base graphs

Images are from an excellent survey paper [Wang+, VLDB 2021]

Graph search algorithms

Table 2: Summary of important representative graph-based ANNS algorithms

Algorithm	Base Graph	Edge	Build Complexity	Search Complexity
KGraph [31]	KNNG	directed	$O(\|S\|^{1.14})$	$O(\|S\|^{0.54})^{\ddagger}$
NGT [46]	KNNG+DG+RNG	directed	$O(\|S\|^{1.14})^{\ddagger}$	$O(\|S\|^{0.59})^{\ddagger}$
SPTAG [27]	KNNG+RNG	directed	$O(\|S\| \cdot \log(\|S\|^{c} + t^{t}))^{\dagger}$	$O(\|S\|^{0.68})^{\ddagger}$
NSW [65]	DG	undirected	$O(\|S\| \cdot \log^{2}(\|S\|))^{\ddagger}$	$O(\log^{2}(\|S\|))^{\dagger}$
IEH [54]	KNNG	directed	$O(\|S\|^{2} \cdot \log(\|S\|) + \|S\|^{2})^{\ddagger}$	$O(\|S\|^{0.52})^{\ddagger}$
FANNG [43]	RNG	directed	$O(\|S\|^{2} \cdot \log(\|S\|))$	$O(\|S\|^{0.2})$
HNSW [67]	DG+RNG	directed	$O(\|S\| \cdot \log(\|S\|))$	$O(\log(\|S\|))$
EFANNA [36]	KNNG	directed	$O(\|S\|^{1.13})^{\ddagger}$	$O(\|S\|^{0.55})^{\ddagger}$
DPG [61]	KNNG+RNG	undirected	$O(\|S\|^{1.14} + \|S\|)^{\ddagger}$	$O(\|S\|^{0.28})^{\ddagger}$
NSG [38]	KNNG+RNG	directed	$O(\|S\|^{\frac{1+c}{c}} \cdot \log(\|S\|) + \|S\|^{1.14})^{\dagger}$	$O(\log(\|S\|))$
HCNNG [72]	MST	directed	$O(\|S\| \cdot \log(\|S\|))$	$O(\|S\|^{0.4})^{\ddagger}$
Vamana [88]	RNG	directed	$O(\|S\|^{1.16})^{\ddagger}$	$O(\|S\|^{0.75})^{\ddagger}$
NSSG [37]	KNNG+RNG	directed	$O(\|S\| + \|S\|^{1.14})$	$O(\log(\|S\|))$

† c, t are the constants. ‡ Complexity is not informed by the authors; we derive it based on the related papers' descriptions and experimental estimates. See Appendix D for deatils.

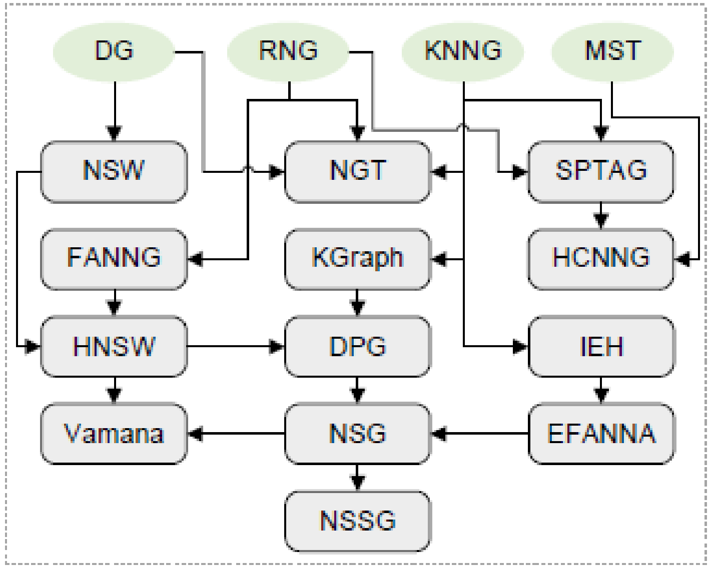
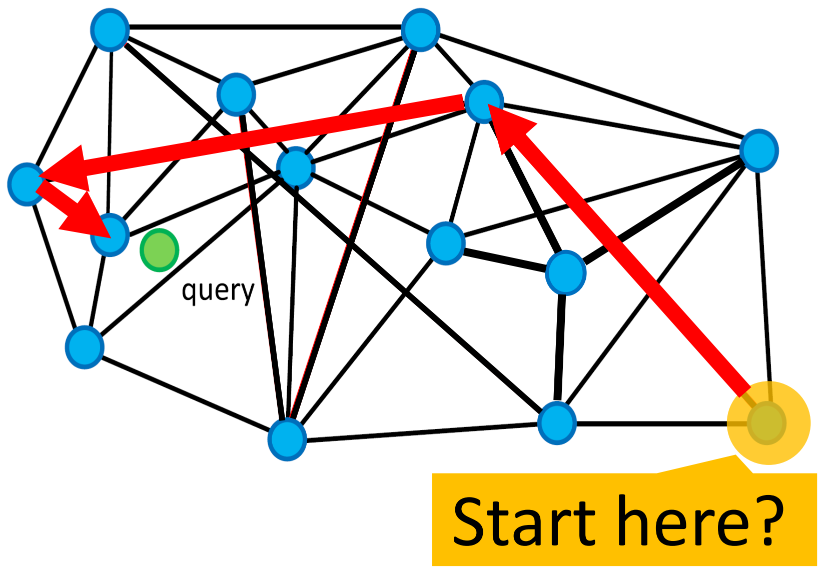


Figure 3: Roadmaps of graph-based ANNS algorithms. The arrows from a base graph (green shading) to an algorithm (gray shading) and from one algorithm to another indicate the dependence and development relationships.

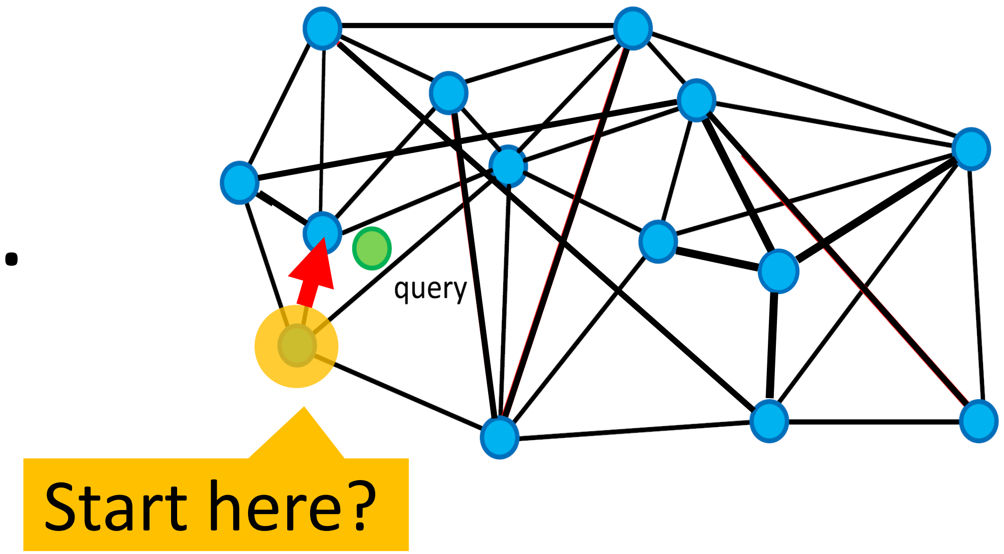
- Lots of algorithms 😂
- The basic structure is same: (1) designing a good graph + (2) beam search

Images are from an excellent survey paper [Wang+, VLDB 2021]

The initial seed matters

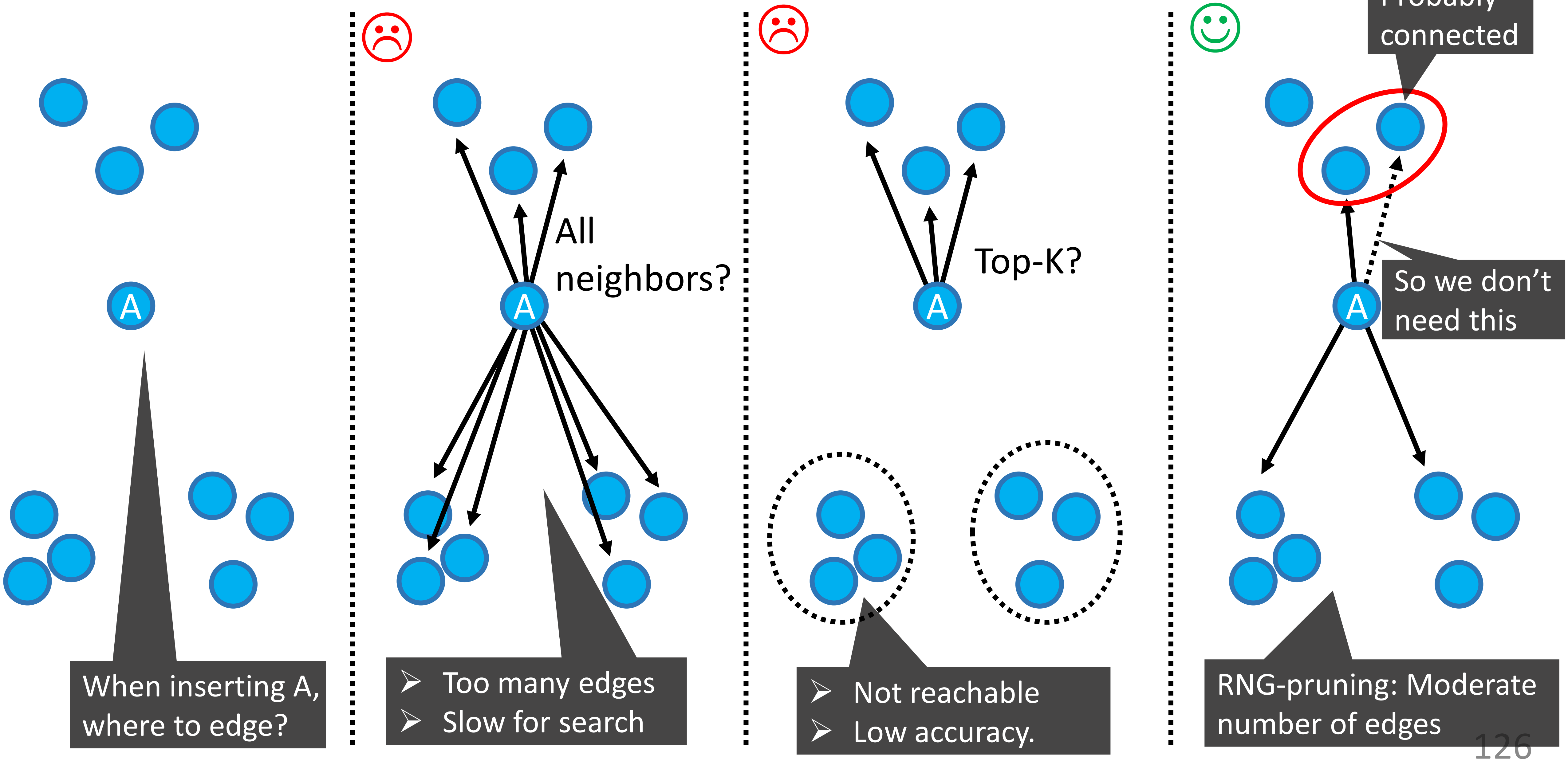


v.s.

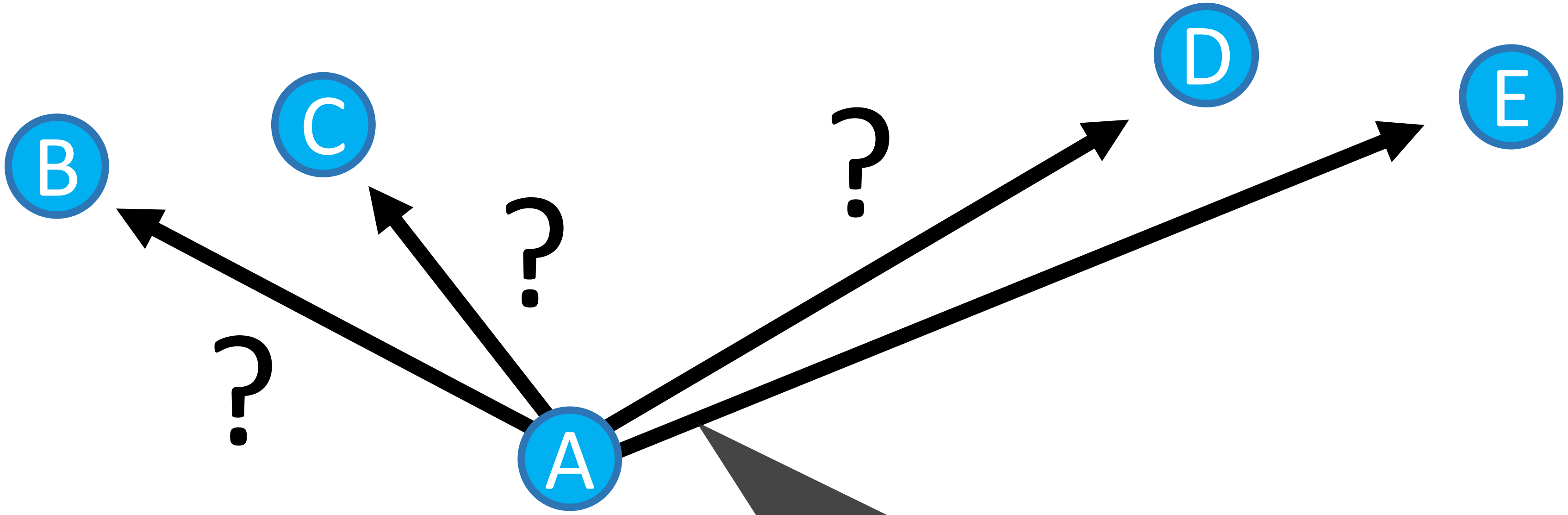


- ➤ Starting from a good seed ➡ Shorter path ➡ Faster search
- ➤ Finding a good seed is also an ANN problem
- ➤ Solve a small ANN problem by tree [NST; Iwasaki+, arXiv 18], hash [Effana; Fu+, arXiv 16] or LSH [LGTM; Arai+, DEXA 21]

Edge selection: RNG-pruning

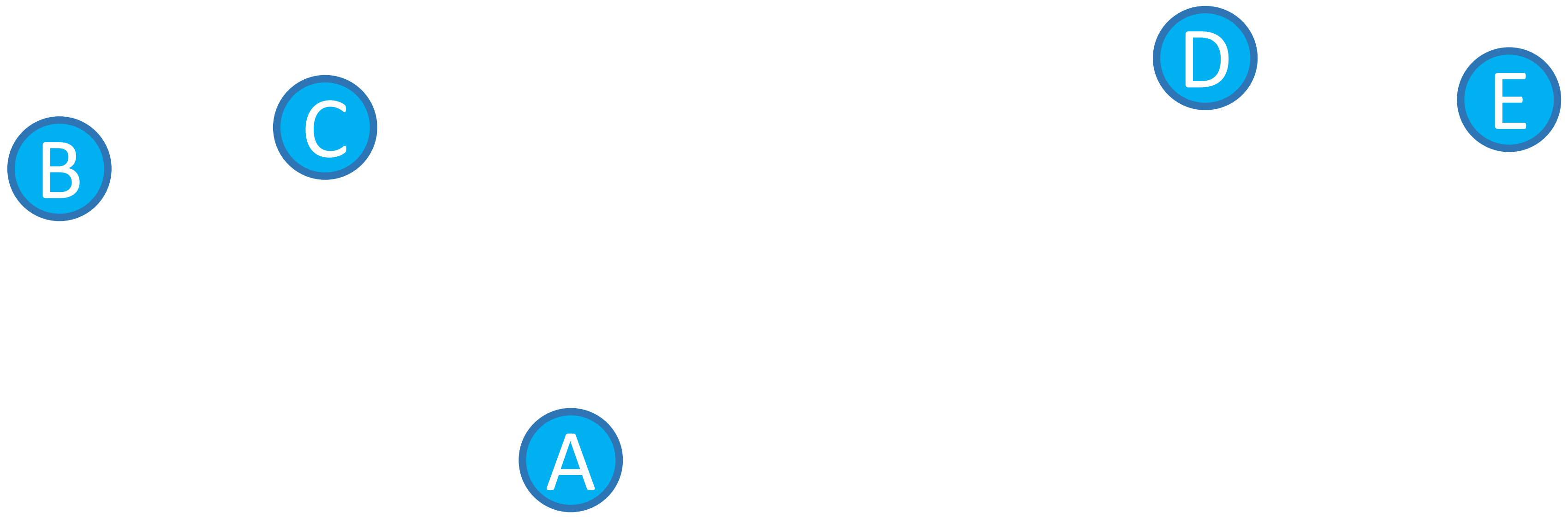


Edge selection: RNG-pruning

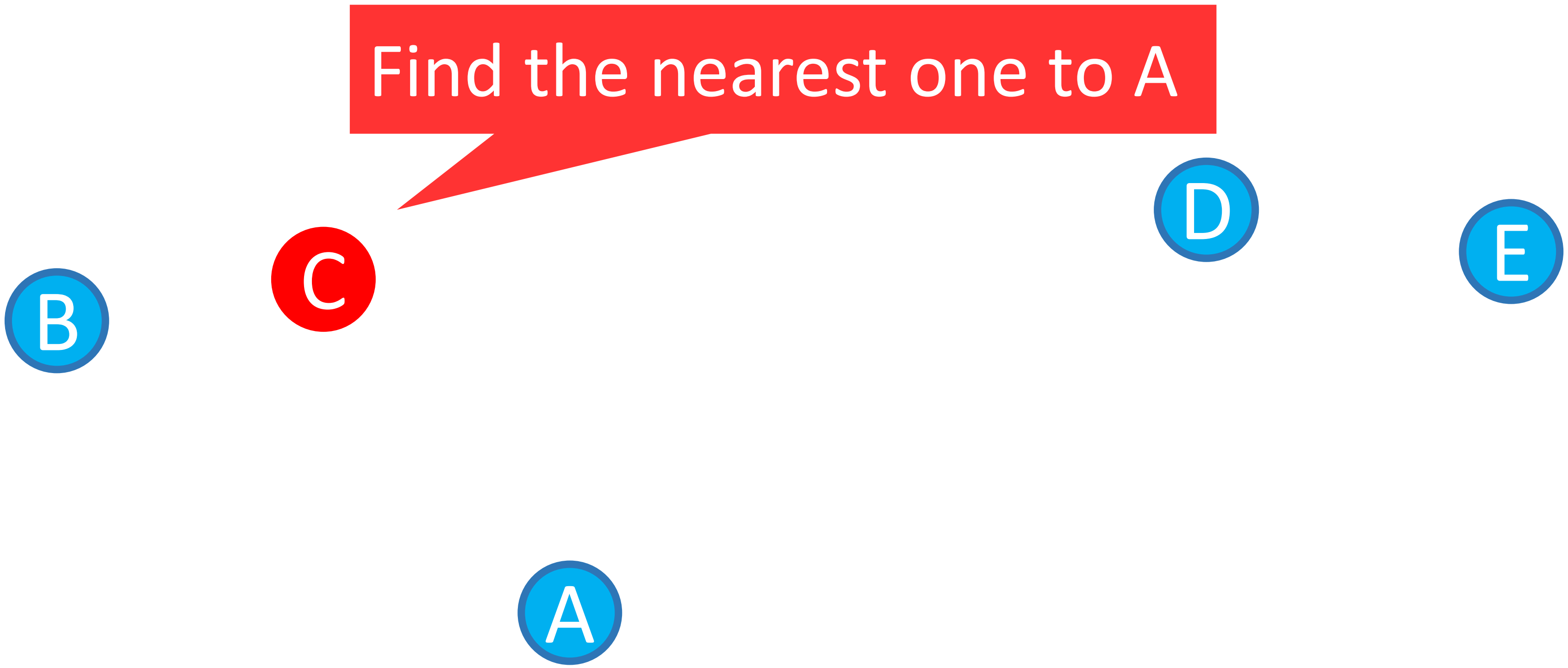


Given A, make edges to B, C, D, and E?

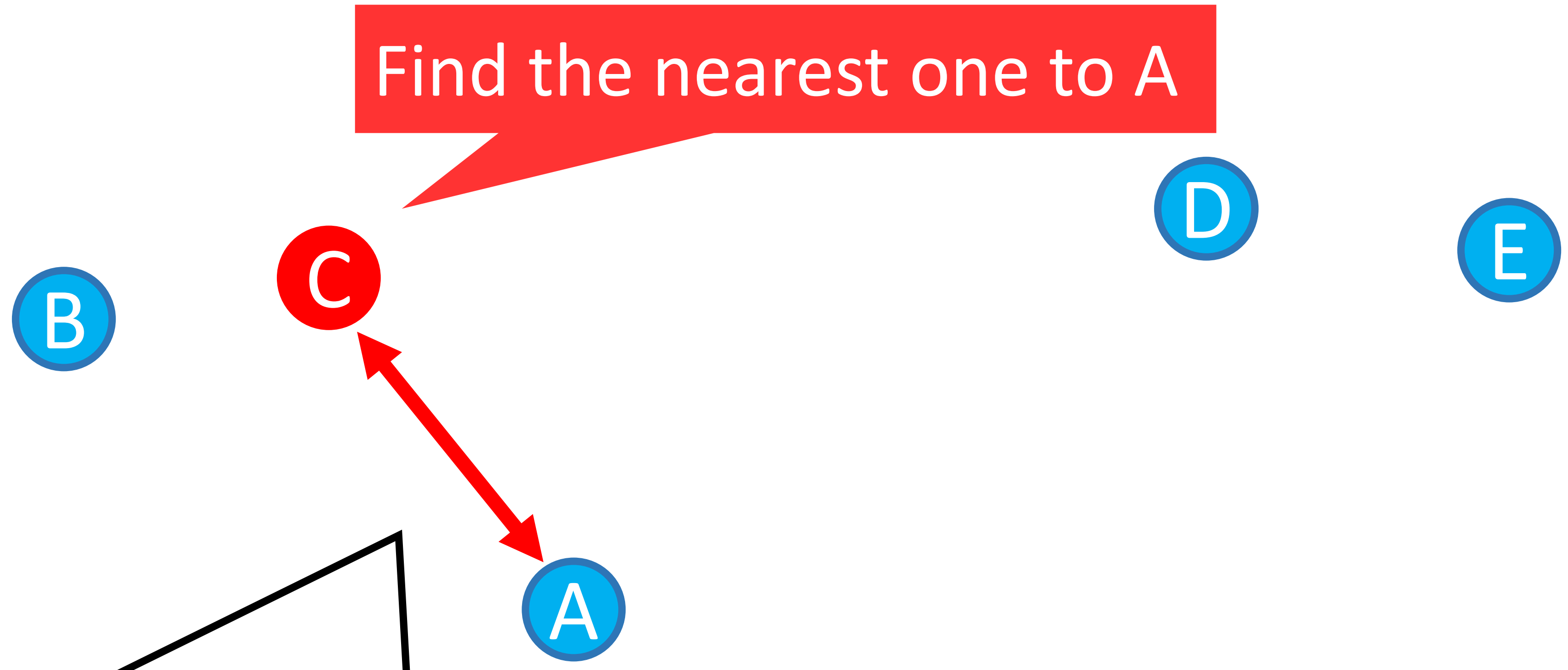
Edge selection: RNG-pruning



Edge selection: RNG-pruning



Edge selection: RNG-pruning



- For all neighbors of A, compare ↔ and ↔
- If ↔ is the shortest, make an edge

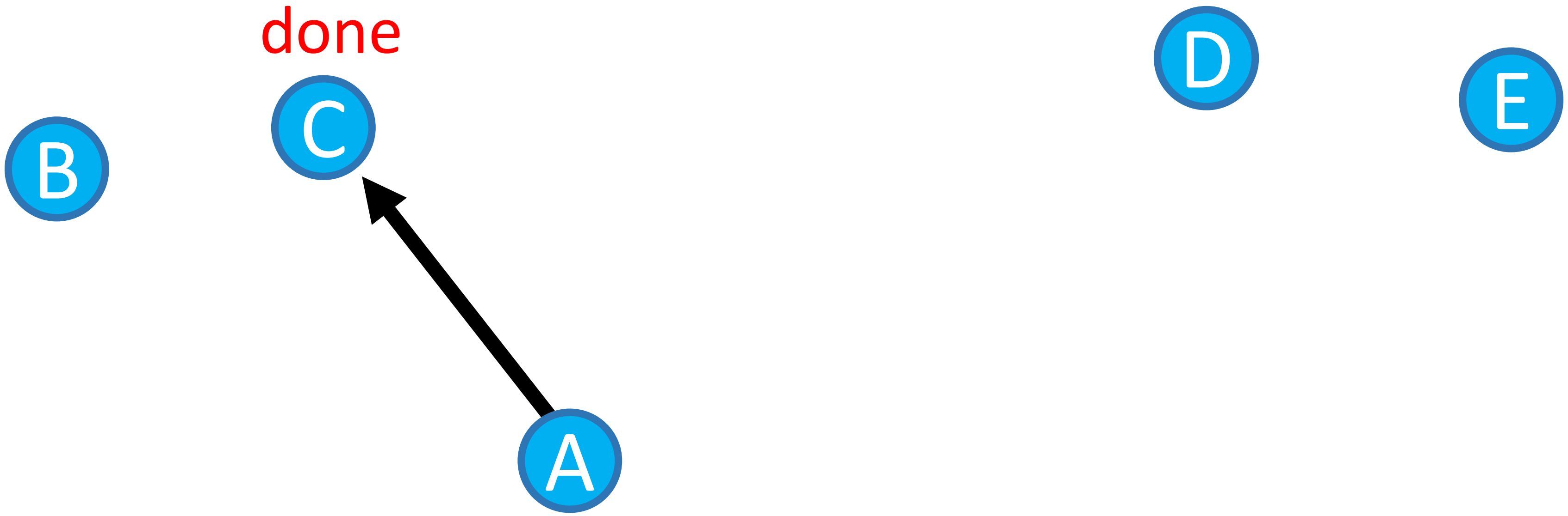
Edge selection: RNG-pruning

Find the nearest one to A

- For all neighbors of A, compare and
- If is the shortest, make an edge

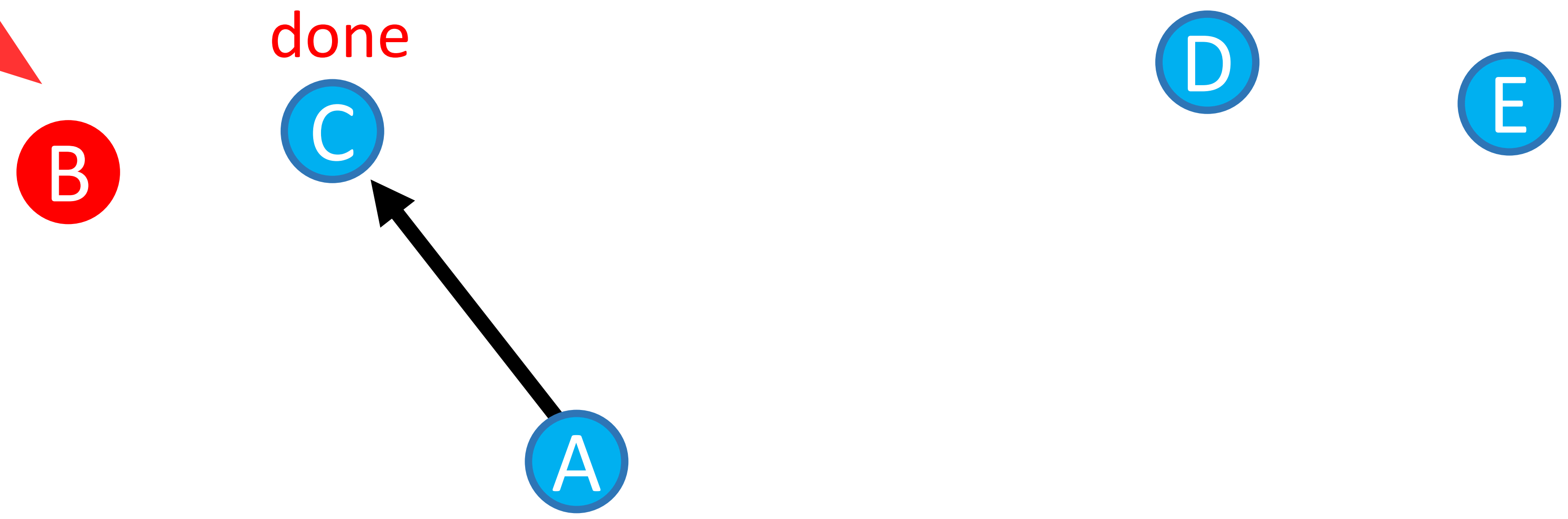
This time, there are no neighbors. So let's make an edge

Edge selection: RNG-pruning



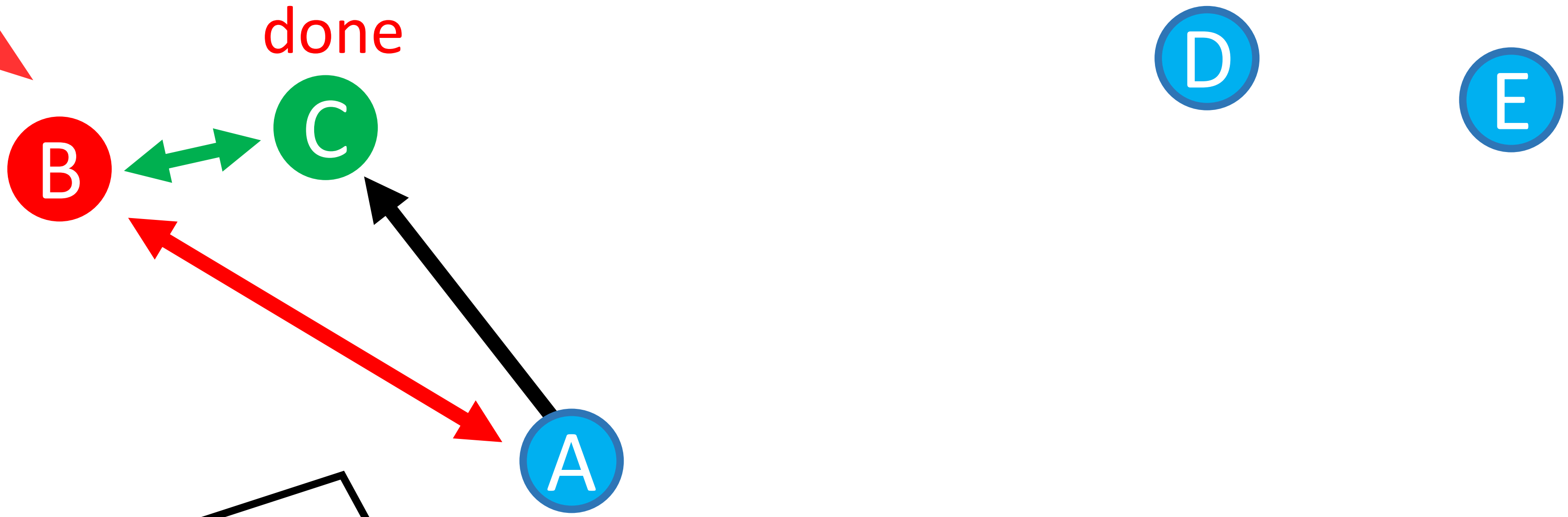
Edge selection: RNG-pruning

Find the 2nd nearest one to A



Edge selection: RNG-pruning

Find the 2nd nearest one to A

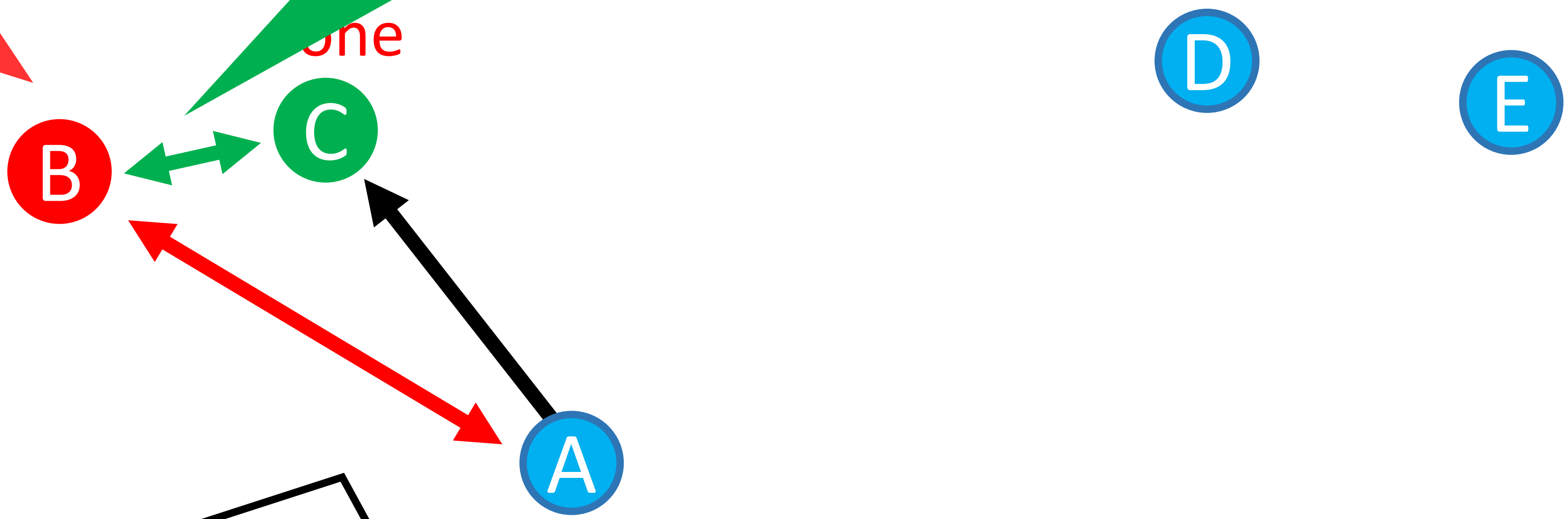


- For all neighbors of A, compare and
- If is the shortest, make an edge

Edge selection: RNG-pruning

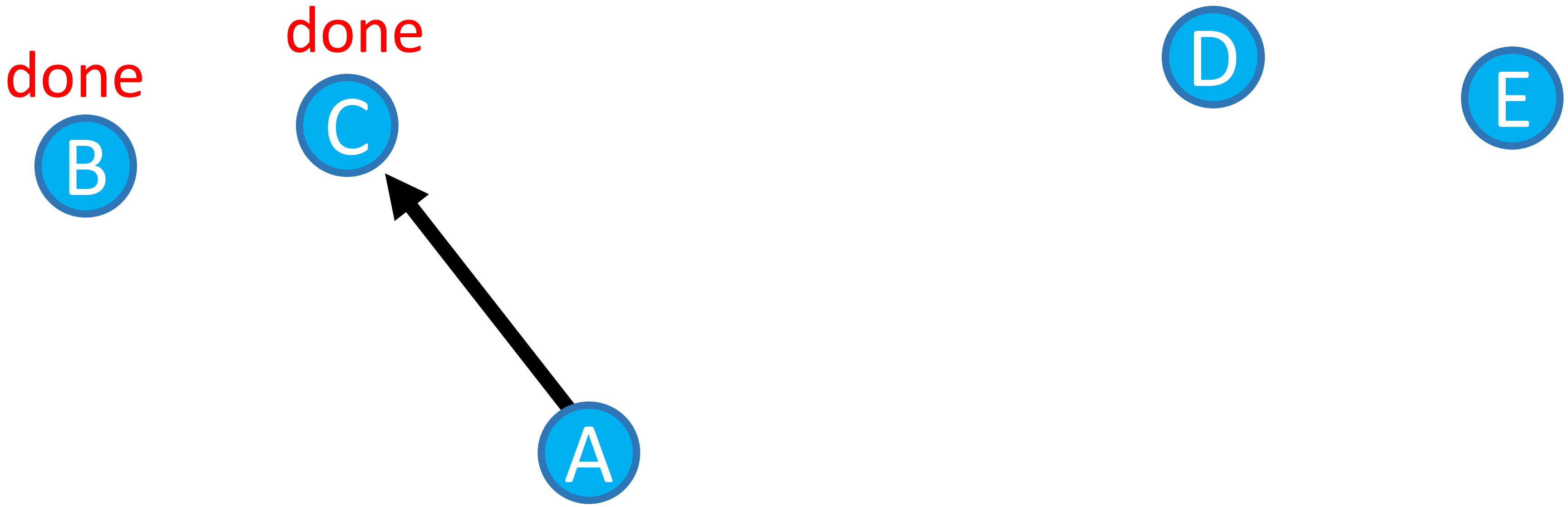
Shortest! Not make an edge

Find the 2nd nearest one

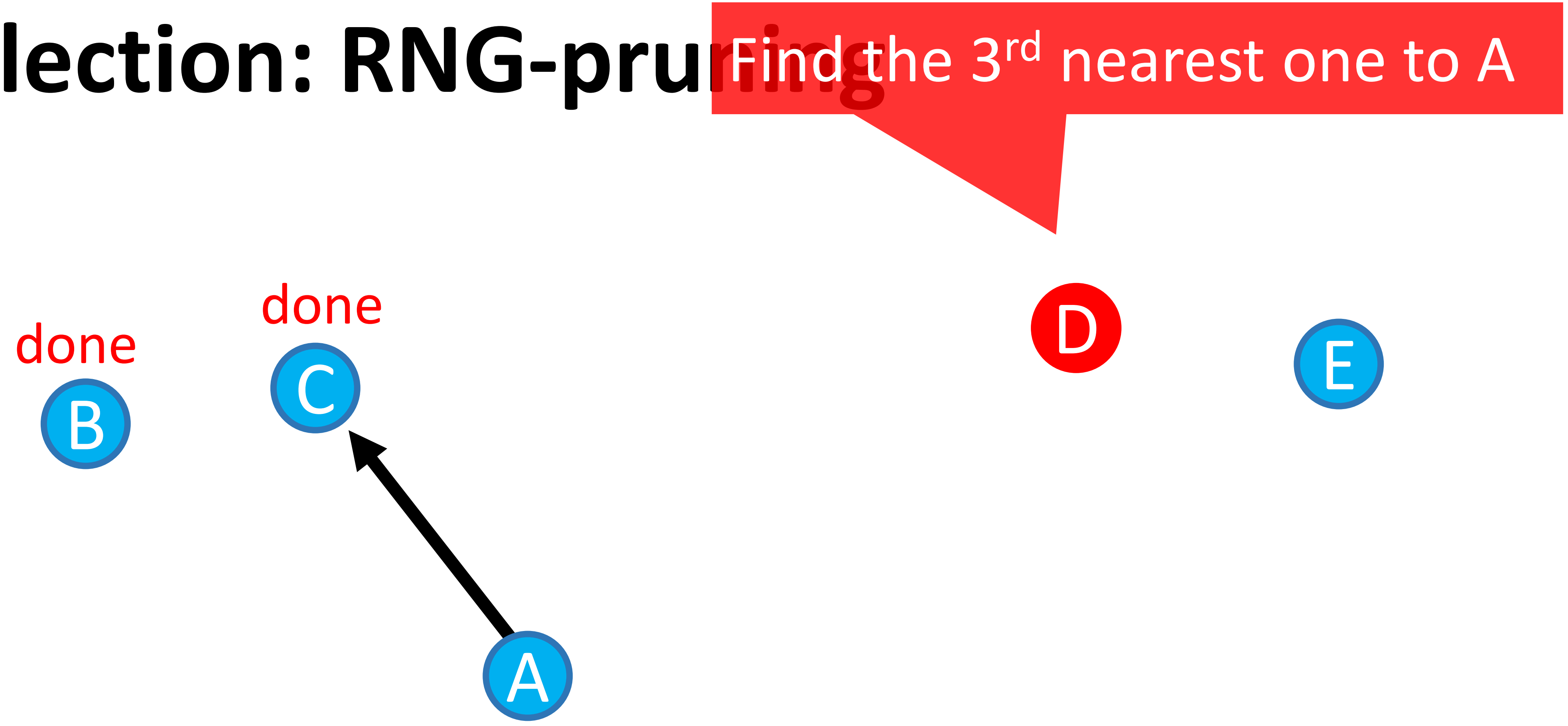


- For all neighbors of A, compare and
- If is the shortest, make an edge

Edge selection: RNG-pruning

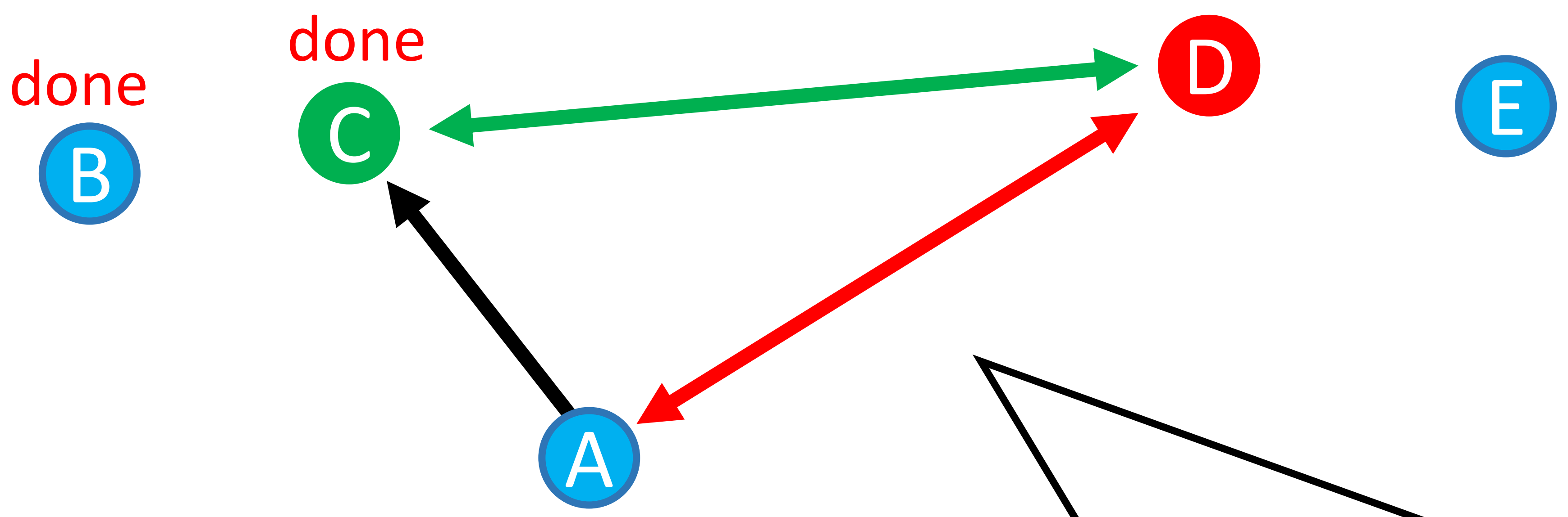


Edge selection: RNG-pruning



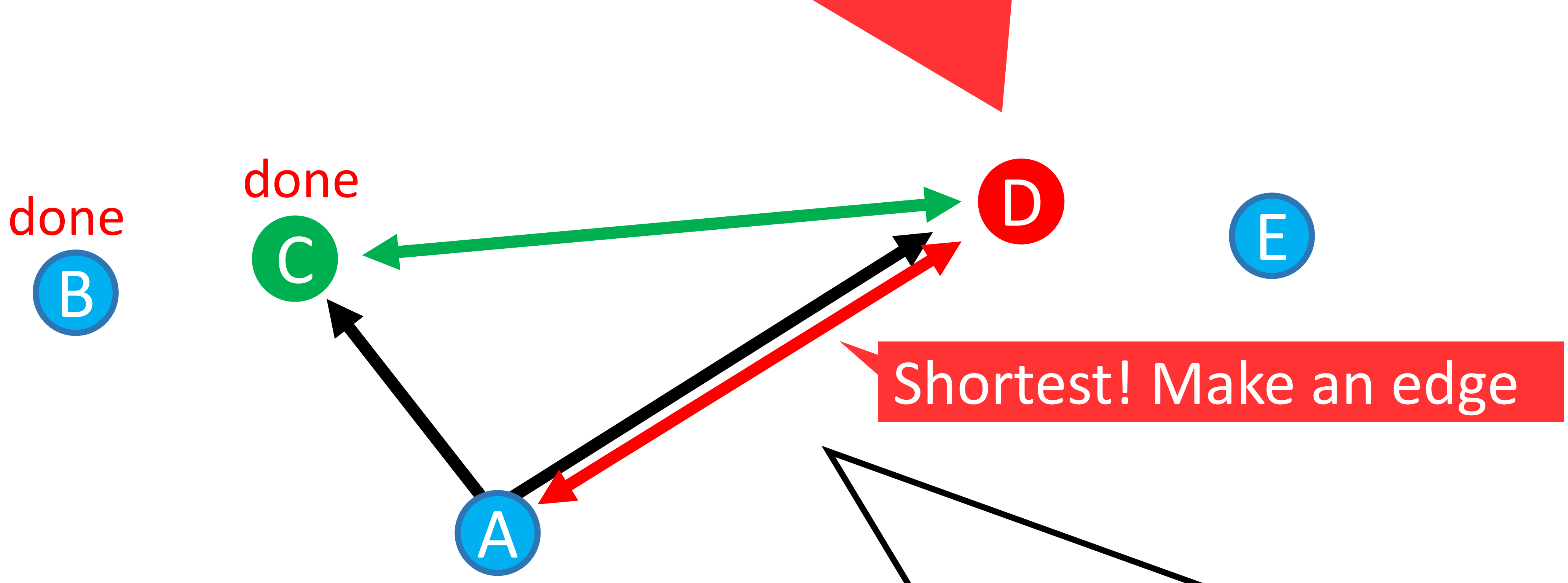
Edge selection: RNG-pruning

Find the 3rd nearest one to A



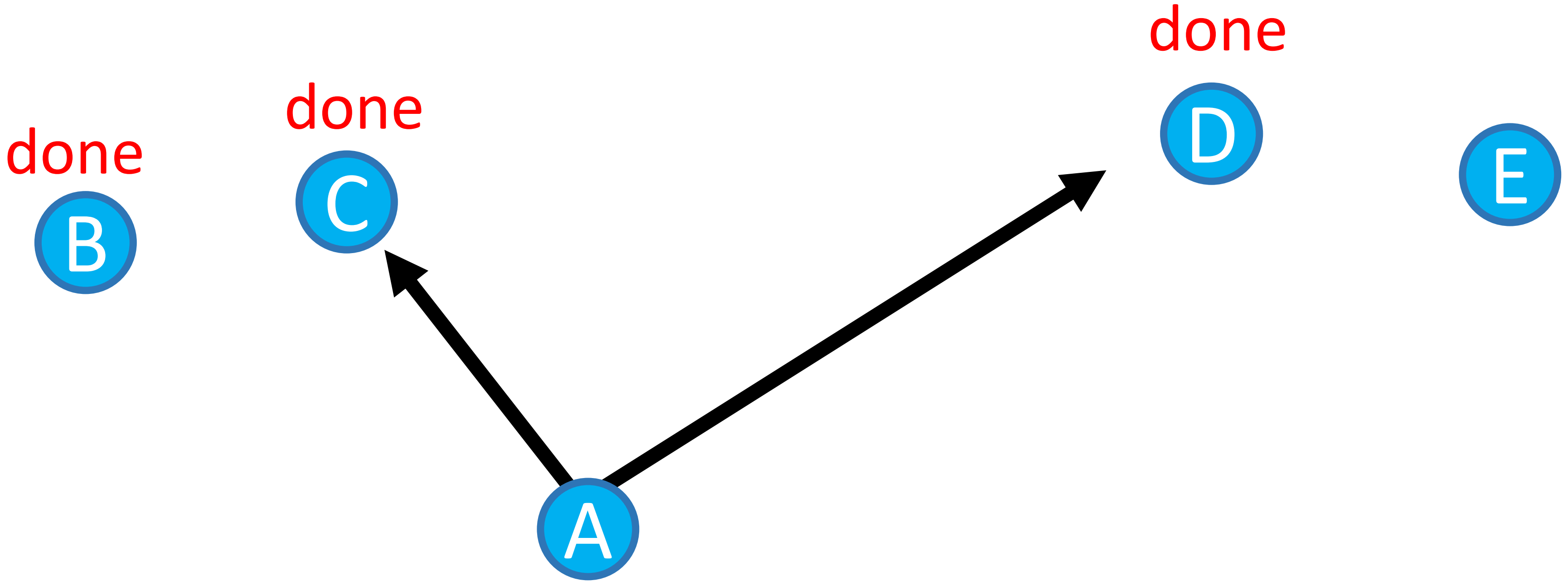
- For all neighbors of A, compare and
- If is the shortest, make an edge

Edge selection: RNG-pruning



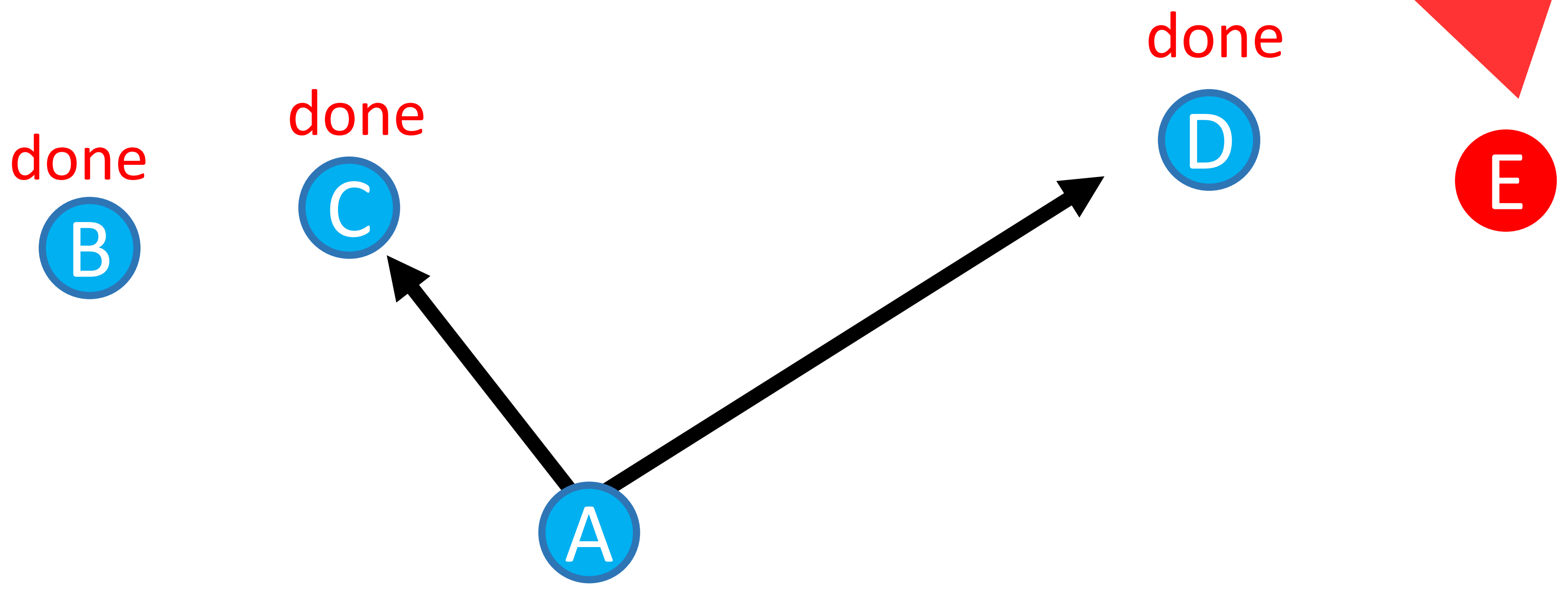
- For all neighbors of A, compare and
- If is the shortest, make an edge

Edge selection: RNG-pruning



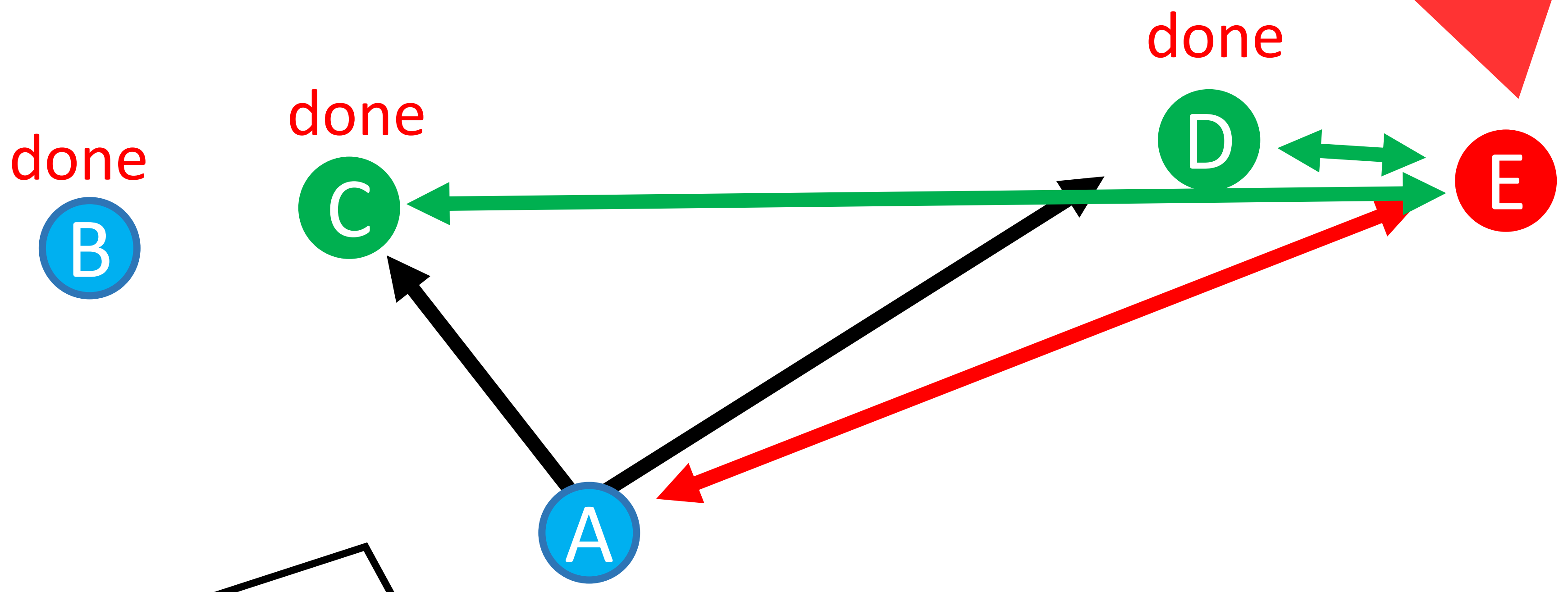
Edge selection: RNG-pruning

Find the 4^{th} nearest one to A



Edge selection: RNG-pruning

Find the 4th nearest one to A

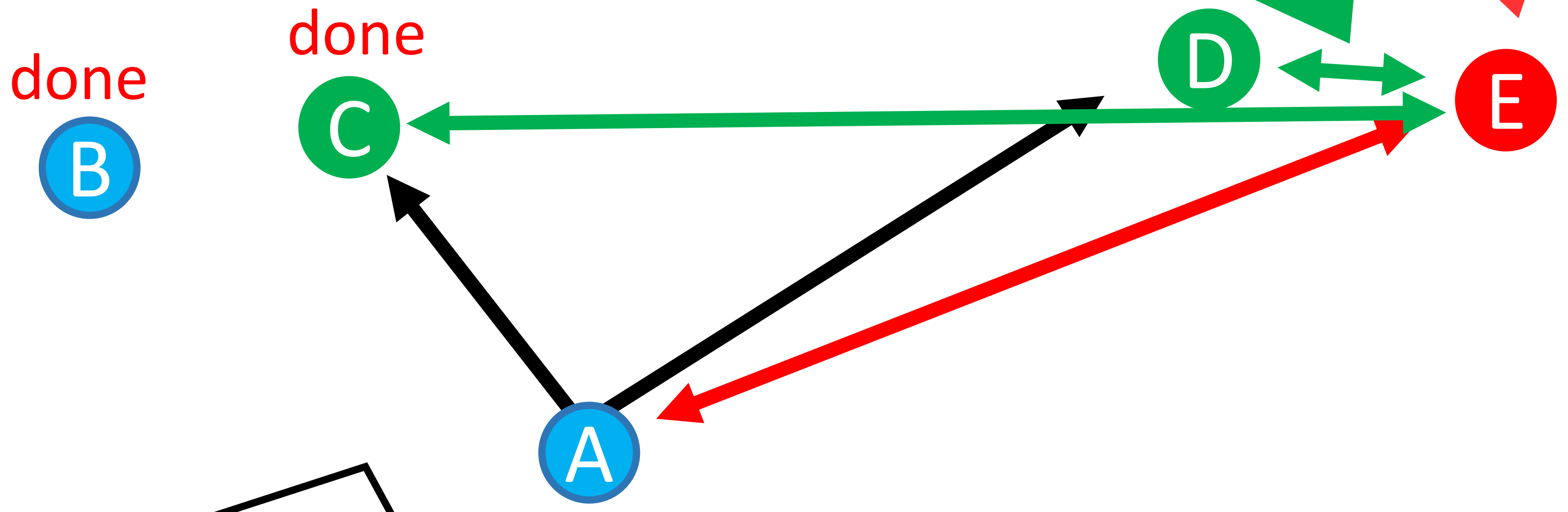


- For all neighbors of A, compare and
- If is the shortest, make an edge

Edge selection: RNG-pruning

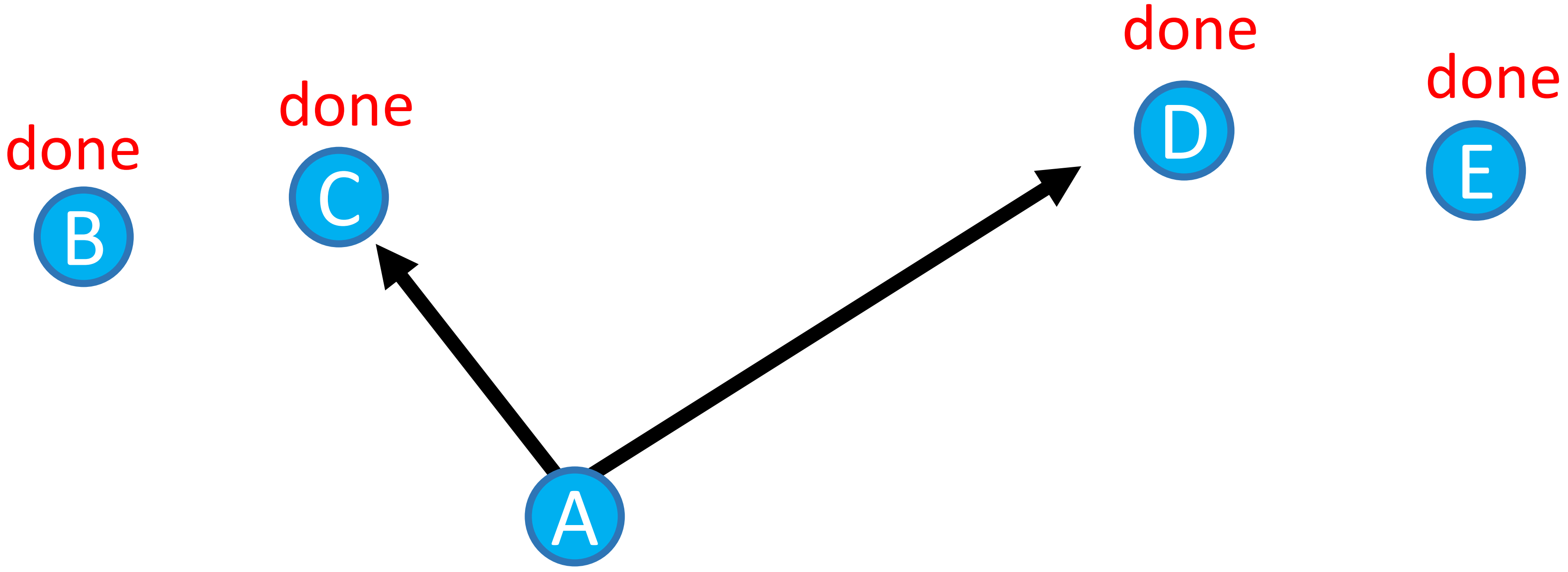
Find the 4^{th} nearest one to A

Shortest! Not make an edge

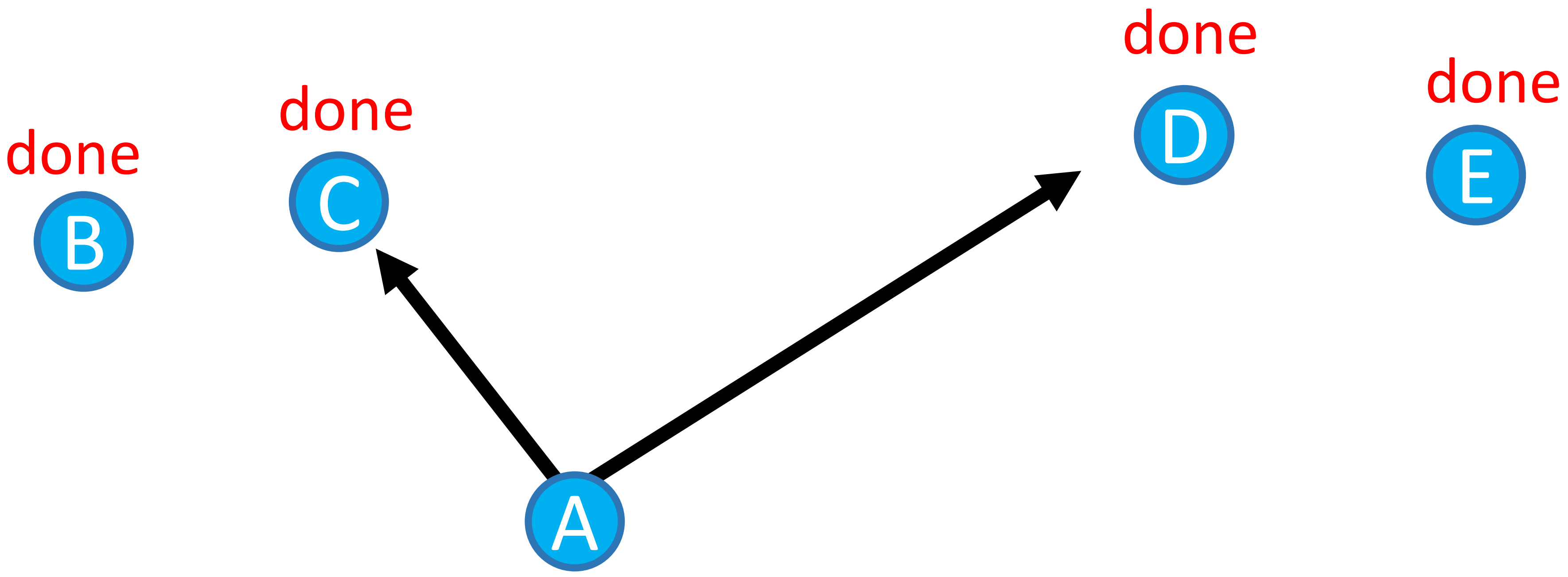


- For all neighbors of A, compare and
- If is the shortest, make an edge

Edge selection: RNG-pruning



Edge selection: RNG-pruning



- RNG-pruning is an effective edge-pruning technique, and used in several algorithms

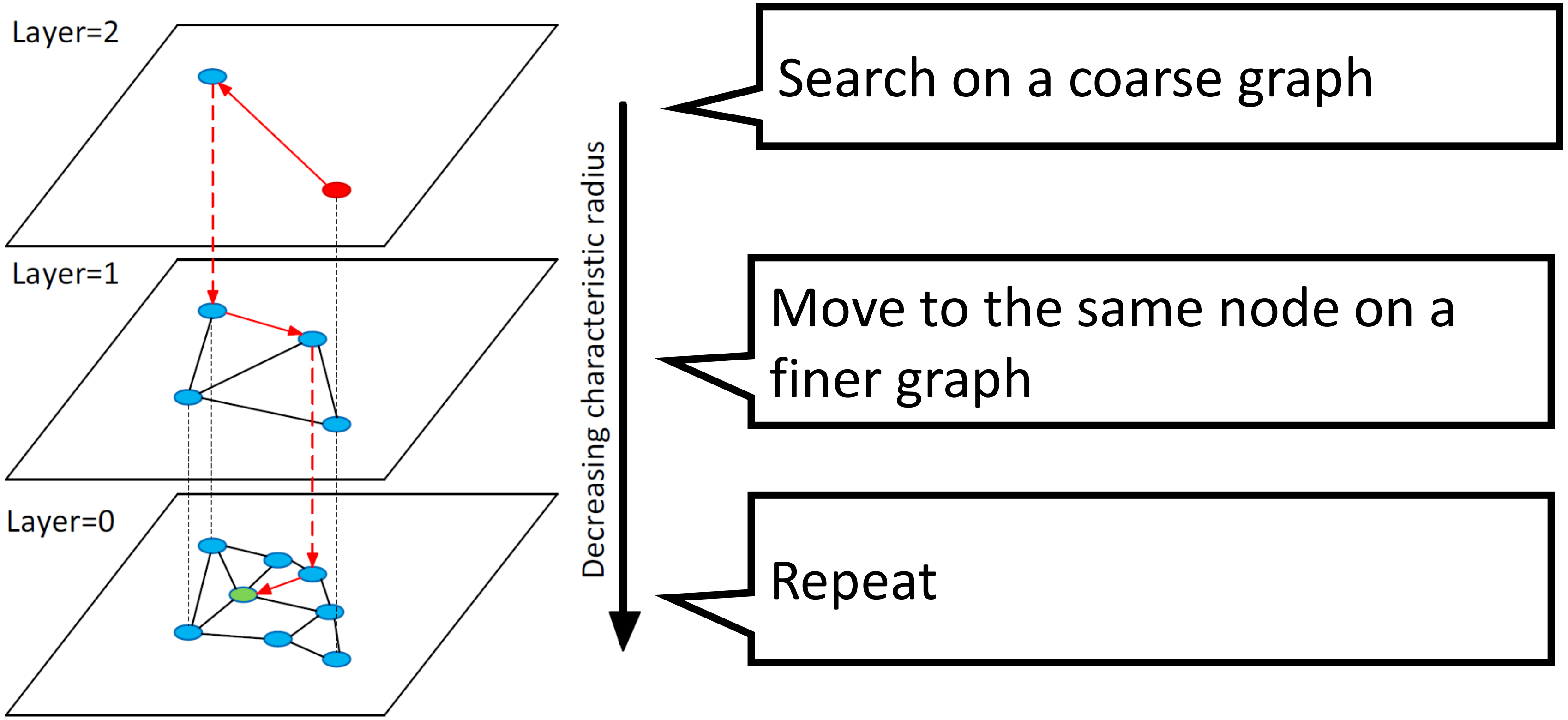
Pros: Implementation is easy

Cons: Require many distance computations

- Background
- Graph-based search
  - Basic (construction and search)
  - Observation
  - Properties
- Representative works
  - HNSW, NSG, NGT, Vamana
- Discussion

Hierarchical Navigable Small World; HNSW

- Construct the graph hierarchically [Malkov and Yashunin, TPAMI, 2019]
- Fix #edge per node by RNG-pruning
- The most famous algorithm; works very well in real world



[Malkov and Yashunin, TPAMI, 2019]

Hierarchical Navigable Small World; HNSW

- Used in various services
  - ✓ milvus, weaviate, qdrant, vearch, elasticsearch, OpenSearch, vespa, redis, Lucene…

- Three famous implementations
  - ✓ NMSLIB (the original implementation)
  - ✓ hnswlib (light-weight implementation from NMSLIB)
  - ✓ Faiss (re-implemented version by the faiss team)

[NMSLIB] https://github.com/nmslib/nmslib
[hnswlib] https://github.com/nmslib/hnswlib
[Faiss] https://github.com/facebookresearch/faiss/blob/main/faiss/IndexHNSW.h

Discussion from Faiss User Forum in FB

Note that this discussion was in 2020 and the libraries have been updated a lot since then

Any implementation difference between NMSLIB, hnswlib, and faiss-hnsw?

Yury Malkov
(the author of
HNSW paper)

My view on the implementation differences (I might forgot something):

1) nmslib’s HNSW requires internal index conversion step (from nmslib’s format to an internal one) to have good performance, and after the conversion the index cannot be updated with new elements. nmslib also has a simple "graph diversification" postprocessing after building the index (controlled by the "post" parameter) and sophisticated queue optimizations which makes it a bit faster compared to other implementations. Another advantage of nmslib is out-of-the box support for large collection of distance functions, including some exotic distances.

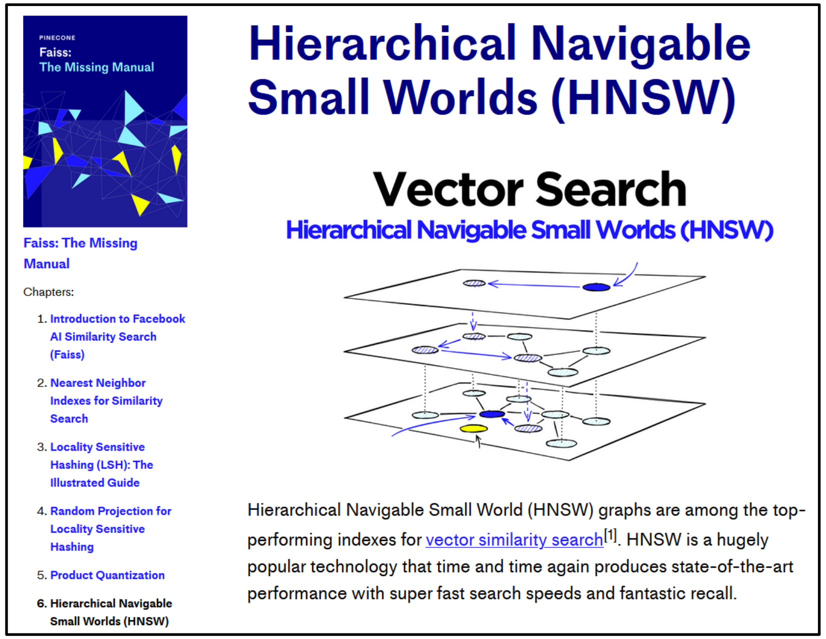
2) hnswlib is a header-only C++ library reimplementation of nmslib's hnsw. It does not have the index conversion step, thus - the Pros (compared to nmslib): much more memory efficient and faster at build time. It also supports index insertions, element updates (with incremental graph rewiring - added recently) and fake deletions (mark elements as deleted to avoid returning them during the graph traversal). Cons (compared to nnmslib): It is a tad slower than nmslib due to lack of graph postprocessing and queue optimization; out-of-the box version supports only 3 distance functions, compared to many distance functions in nmslib. Overall, I've tried to keep hnswlib as close as possible to a distributed index (hence no index postprocessing).

3) Faiss hnsw is a different reimplementation. It has its own algorithmic features, like having the first elements in the upper layers on the structure (opposed to random in other implementations). It is a bit more memory efficient compared to hnswlib with raw vectors and optimized for batch processing. Due to the latter it is noticeably slower at single query processing (opposed to nmslib or hnswlib) and generally a bit slower for batch queries (the last time I’ve tested, but there were exceptions). The implementation also supports incremental insertions (also preferably batched), quantized data and two-level encoding, which makes it much less memory hungry and the overall best when memory is a big concern.

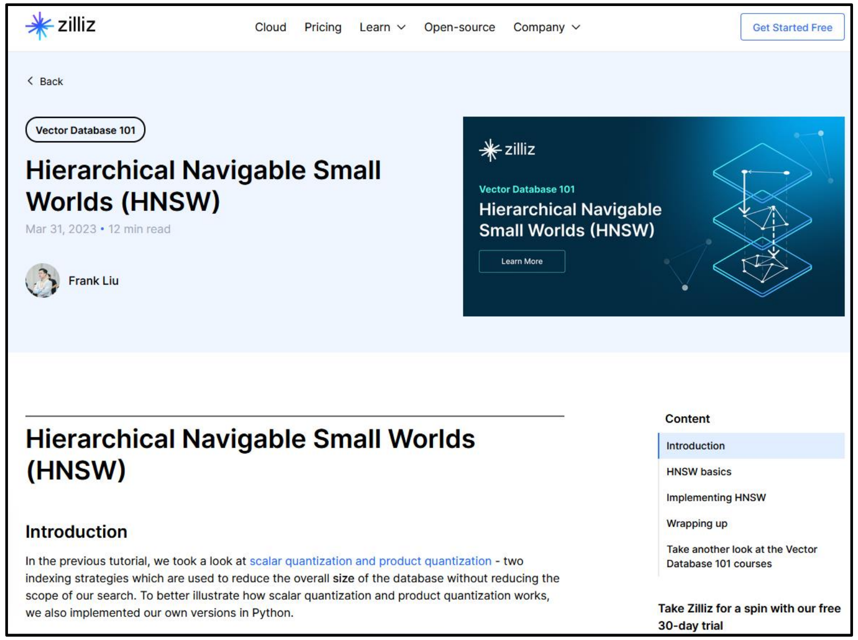
https://www.facebook.com/groups/faissusers/posts/917143142043306/?comment_id=917533385337615&reply_comment_id=920542105036743

Hierarchical Navigable Small World; HNSW

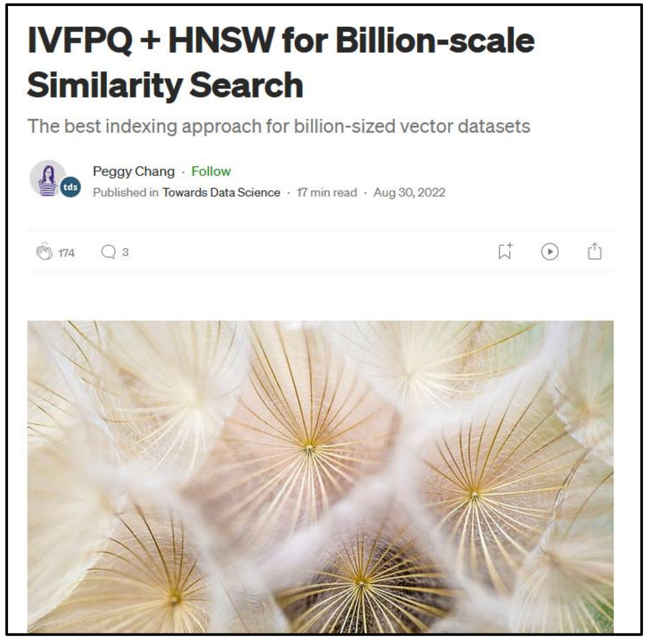
➢ See the following excellent blog posts for more details



https://www.pinecone.io/learn/hnsw/
James Briggs, PINECONE, Faiss: The Missing Manual, 6. Hierarchical Navigable Small Worlds (HNSW)



https://zilliz.com/blog/hierarchical-navigable-small-worlds-HNSW
Frank Liu, zilliz, Vector Database 101, Hierarchical Navigable Small Worlds (HNSW)



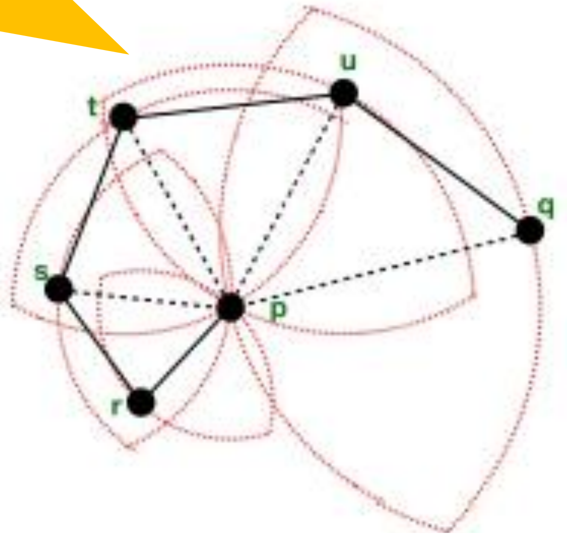
https://towardsdatascience.com/ivfpq-hnsw-for-billion-scale-similarity-search-89ff2f89d90e
Peggy Chang, IVFPQ + HNSW for Billion-scale Similarity Search

Navigating Spreading-out Graph (NSG) [Fu+, VLDB 19]

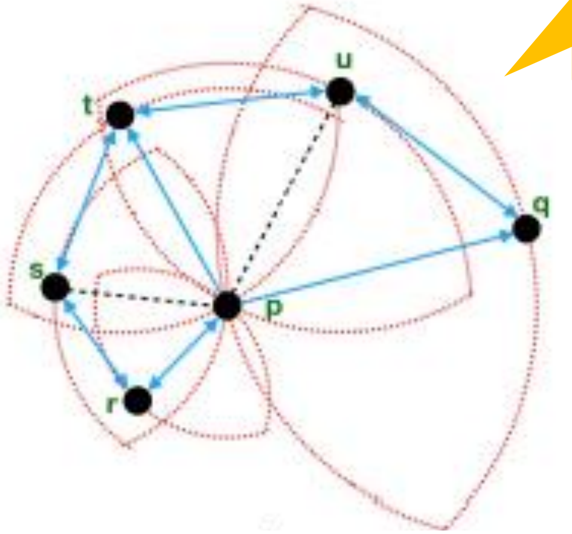
- ➢ Monotonic RNG
- ➢ In some cases, slightly better than HNSW
- ➢ Used in Alibaba’s Taobao

- ➢ Recall the def. of RNG is “no point in a lune”
- ➢ The path “p -> q” is ling

Monotonic RNG can make more edges



RNG



Monotonic RNG

Images are from [Fu+, VLDB 19]

Navigating Spreading-out Graph (NSG) [Fu+, VLDB 19]

- The original implementation: https://github.com/ZJULearning/nsg

- Implemented in faiss as well
- If you're using faiss-hnsw and need a little bit more performance with the same interface, worth trying NSG

```
IndexHNSWFlat(int d, int M, MetricType metric)
IndexNSGFlat(int d, int R, MetricType metric)
```

Neighborhood Graph and Tree (NGT)

[Iwasaki+, arXiv 18]

- Make use of range search for construction
- Obtain a seed via VP-tree

- Current best methods in ann-benchmarks are NGT-based algorithms
- Quantization is natively available

- Repository: https://github.com/yahoojapan/NGT
- From Yahoo Japan
- Used in Vald

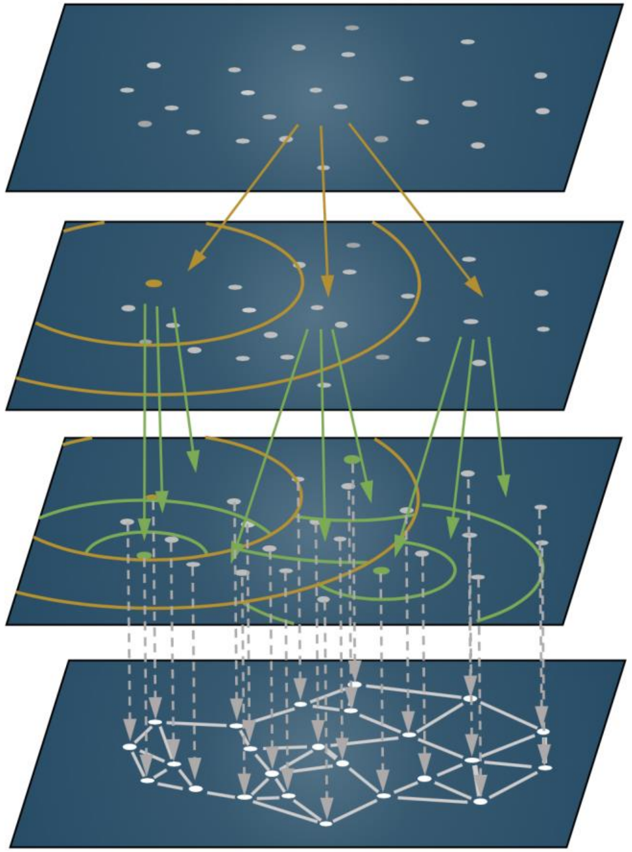
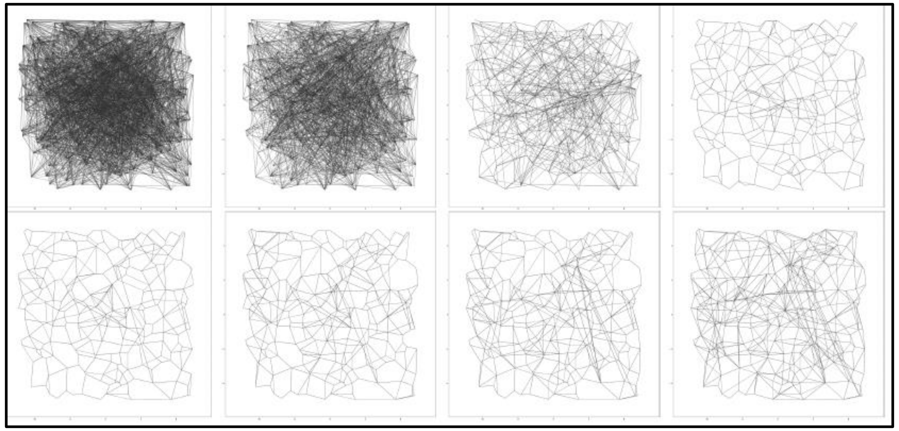


Image are from the original repository

DiskANN (Vamana) [Subramanya+, NeurIPS 19]

- Vamana: Graph-based search algorithm
- DiskANN: Disk-friendly search system using Vamana
- From MSR India https://github.com/microsoft/DiskANN



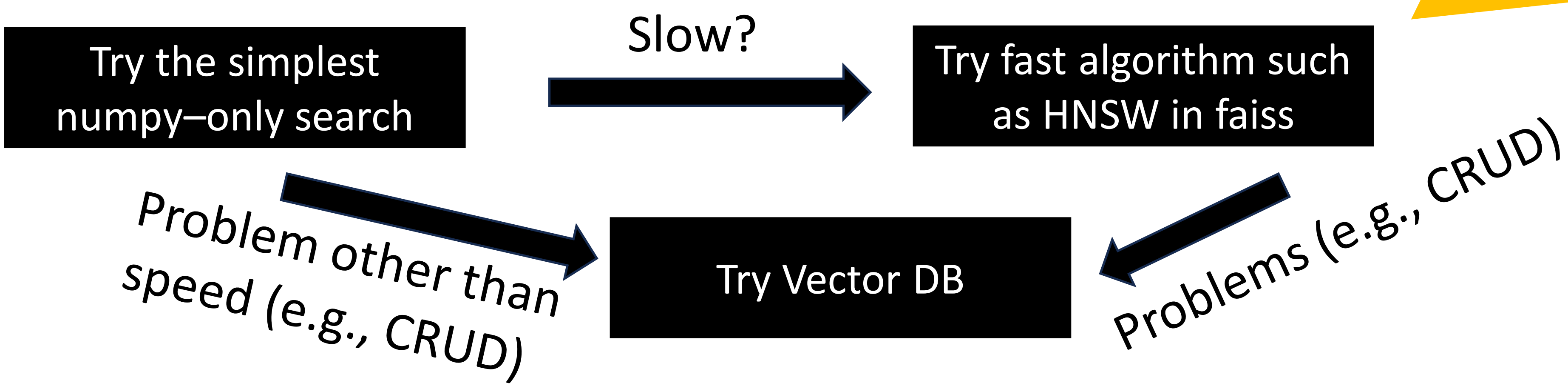
- Good option for huge data (not the main focus of this talk, though)
- The same team is actively developing interesting functionalites
  - ✓ Data update: FreshDiskANN [Singh+, arXiv 21]
  - ✓ Filter: Filtered-DiskANN [Gollapudi+, WWW 23]

- Background
- Graph-based search
  - Basic (construction and search)
  - Observation
  - Properties
- Representative works
  - HNSW, NSG, NGT, Vamana
- Discussion

Just NN? Vector DB?

- Vector DB companies say “Vector DB is cool”
  - ✓ https://weaviate.io/blog/vector-library-vs-vector-database
  - ✓ https://codelabs.milvus.io/vector-database-101-what-is-a-vector-database/index#2
  - ✓ https://zilliz.com/learn/what-is-vector-database
- My own idea:

If speed is the only concern, just use libraries



- Which vector DB? ➡ No conclusions!
- If you need a clean & well designed API, I recommend taking a look at docarray in Jina AI (see Han’s talk today!)

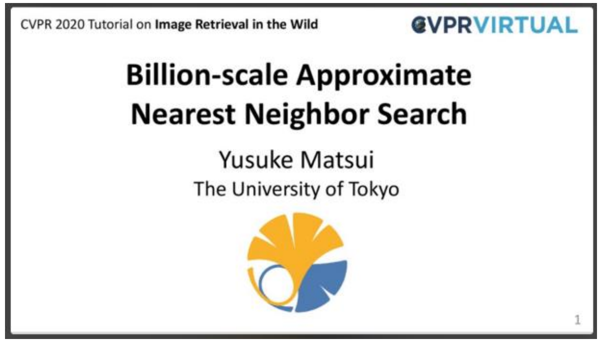
Useful resources

- Several companies have very useful blog series

- Pinecone Blog
  - ✓ https://www.pinecone.io/learn/
- Weaviate Blog
  - ✓ https://weaviate.io/blog
- Jina AI Blog
  - ✓ https://jina.ai/news/
- Zilliz Blog
  - ✓ https://zilliz.com/blog
- Romain Beaumont Blog
  - ✓ https://rom1504.medium.com/

Progress in the last three years

- Three years have passed since my previous tutorial at CVPR 2020



Y. Matasui, “Billion-scale Approximate Nearest Neighbor Search”, CVPR 2020 Tutorial
- Slide: https://speakerdeck.com/matsui_528/cvpr20-tutorial-billion-scale-approximate-nearest-neighbor-search
- Video: https://youtu.be/SKrHs03i08Q

- What progress in the last three years in the ANN field?

Progress in the last three years

- The basic framework is still same (HNSW and IVFPQ!)
- HNSW is still de facto standard; although several papers claim they perform better
- Disk-based systems are getting attention
- **Vector DB** has gained rapid popularity for LLM applications.
- Because of LLM, we should suppose **D as ~1000** (not ~100)
- GPU-ANN is powerful, but less widespread than I expected; CPUs are more convenient for LLM
- Competitions (SISAP and bigann-benchmarks)
- New billion-scale datasets
- A breakthrough algorithm that goes beyond graph-based methods awaits. 🤔

- ➢ Background
- ➢ Graph-based search
  - ✓ Basic (construction and search)
  - ✓ Observation
  - ✓ Properties
- ➢ Representative works
  - ✓ HNSW, NSG, NGT, Vamana
- ➢ Discussion

To make graph search **not a black box**

Reference

- [Jégou+, TPAMI 2011] H. Jégou+, "Product Quantization for Nearest Neighbor Search", IEEE TPAMI 2011
- [Guo+, ICML 2020] R. Guo+, "Accelerating Large-Scale Inference with Anisotropic Vector Quantization", ICML 2020
- [Malkov+, TPAMI 2019] Y. Malkov+, "Efficient and Robust Approximate Nearest Neighbor search using Hierarchical Navigable Small World Graphs," IEEE TPAMI 2019
- [Malkov+, IS 13] Y, Malkov+, "Approximate Nearest Neighbor Algorithm based on Navigable Small World Graphs", Information Systems 2013
- [Fu+, VLDB 19] C. Fu+, "Fast Approximate Nearest Neighbor Search With The Navigating Spreading-out Graphs", 2019
- [Subramanya+, NeurIPS 19] S. J. Subramanya+, "DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node", NeurIPS 2019
- [Baranchuk+, ICML 19] D. Baranchuk+, "Learning to Route in Similarity Graphs"
- [Wang+, VLDB 21] M. Wang+, "A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search", VLDB 2021
- [Toussaint, PR 80] G. T. Toussaint, "The Relative Neighbouhood Graph of A Finite Planar Set", Pattern Recognition 1980
- [Fu+, arXiv 16] C. Fu and D. Cai, "Efanna: An Extremely Fast Approximate Nearest Neighbor Search Algorithm based on knn Graph", arXiv 2016
- [Arai+, DEXA 21] Y. Arai+, "LGTM: A Fast and Accurate kNN Search Algorithm in High-Dimensional Spaces", DEXA 2021
- [Iwasaki+, arXiv 18] M. Iwasaki and D. Miyazaki, "Optimization if Indexing Based on k-Nearest Neighbor Graph for Proximity Search in High-dimensional Data", arXiv 2018
- [Singh+, arXiv 21] A. Singh+, "FreshDiskANN: A Fast and Accurate Graph-Based ANN Index for Streaming Similarity Search", arXiv 2021
- [Gollapudi+, WWW 23] S. Gollapudi+, "Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters", WWW 2023

Reference

- [Pinecone] https://www.pinecone.io/
- [Milvus] https://milvus.io/
- [Qdrant] https://qdrant.tech/
- [Weaviate] https://weaviate.io/
- [Vertex AI Matching Engine] https://cloud.google.com/vertex-ai/docs/matching-engine
- [Vald] https://vald.vdaas.org/
- [Vearch] https://vearch.github.io/
- [Elasticsearch] https://www.elastic.co/jp/blog/introducing-approximate-nearest-neighbor-search-in-elasticsearch-8-0
- [OpenSearch] https://opensearch.org/docs/latest/search-plugins/knn/approximate-knn/
- [Vespa] https://vespa.ai/
- [Redis] https://redis.com/solutions/use-cases/vector-database/
- [Lucene] https://lucene.apache.org/core/9_1_0/core/org/apache/lucene/util/hnsw/HnswGraphSearcher.html
- [SISAP] SISAP 2023 Indexing Challenge https://sisap-challenges.github.io/
- [Bigann-benchmarks] Billion-Scale Approximate Nearest Neighbor Search Challenge: NeurIPS'21 competition track https://big-ann-benchmarks.com/

Thank you!

Time	Session	Presenter
~~13:30 – 13:40~~	~~Opening~~	~~Yusuke Matsui~~
~~13:40 – 14:30~~	~~Theory and Applications of Graph-based Search~~	~~Yusuke Matsui~~
14:30 – 15:20	A Survey on Approximate Nearest Neighbors in a Billion-Scale Settings	Martin Aumüller
15:20 – 15:30	Break	
15:30 – 16:20	Query Language for Neural Search in Practical Applications	Han Xiao

Acknowledgements

- I would like to express my deep gratitude to Prof. Daichi Amagata, Naoki Ono, and Tomohiro Kanaumi for reviewing the contents of this tutorial and providing valuable feedback.
- This work was supported by JST AIP Acceleration Research JPMJCR23U2, Japan.

Billion-Scale Nearest Neighbor Search

CVPR 2023 Tutorial on **Neural Search in Action**, Part 2

Martin Aumüller

IT University of Copenhagen, maau@itu.dk

JUNE 18-22, 2023 CVPR VANCOUVER, CANADA

IT UNIVERSITY OF CPH

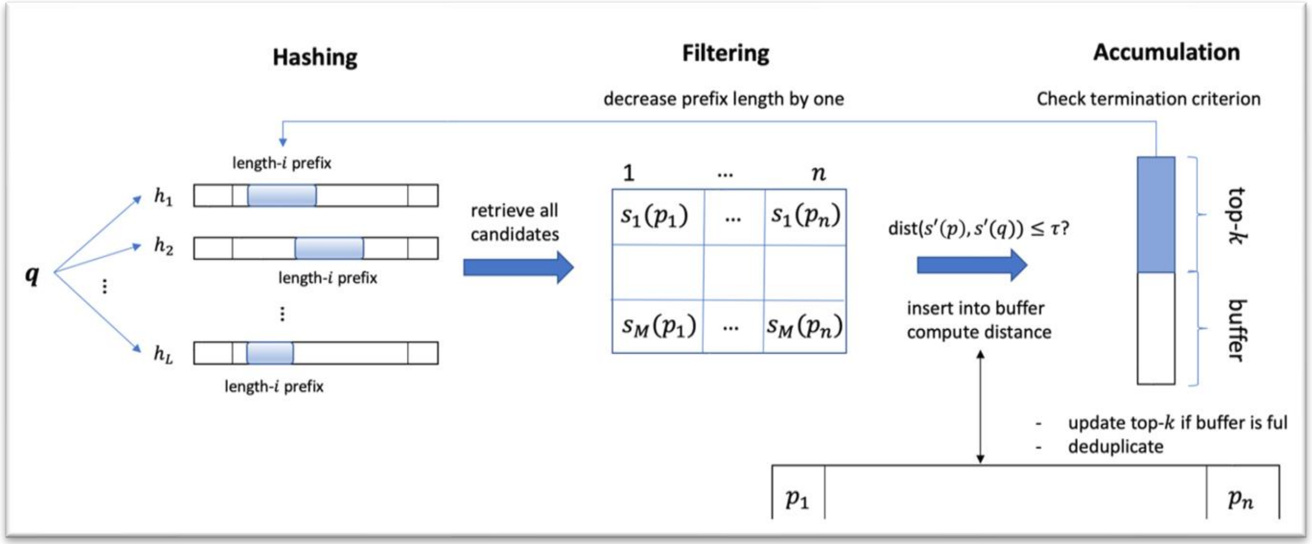
Martin Aumüller

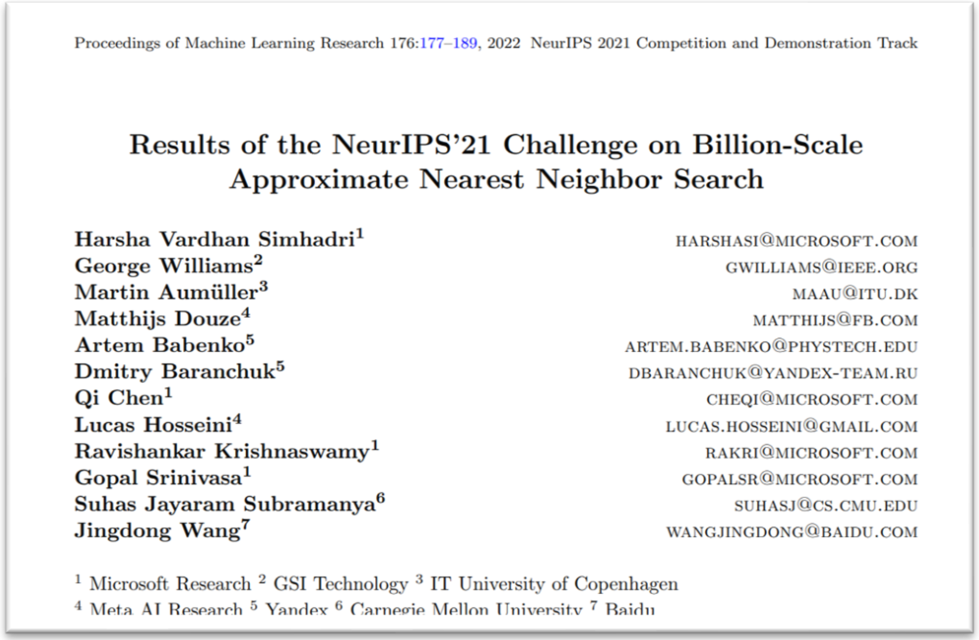
Associate Professor, IT University of Copenhagen, Denmark

http://itu.dk/people/maau @maumueller

✓ Similarity search using hashing

✓ Benchmarking & workload generation





Proceedings of Machine Learning Research 176:177–189, 2022 NeurIPS 2021 Competition and Demonstration Track

Results of the NeurIPS'21 Challenge on Billion-Scale Approximate Nearest Neighbor Search

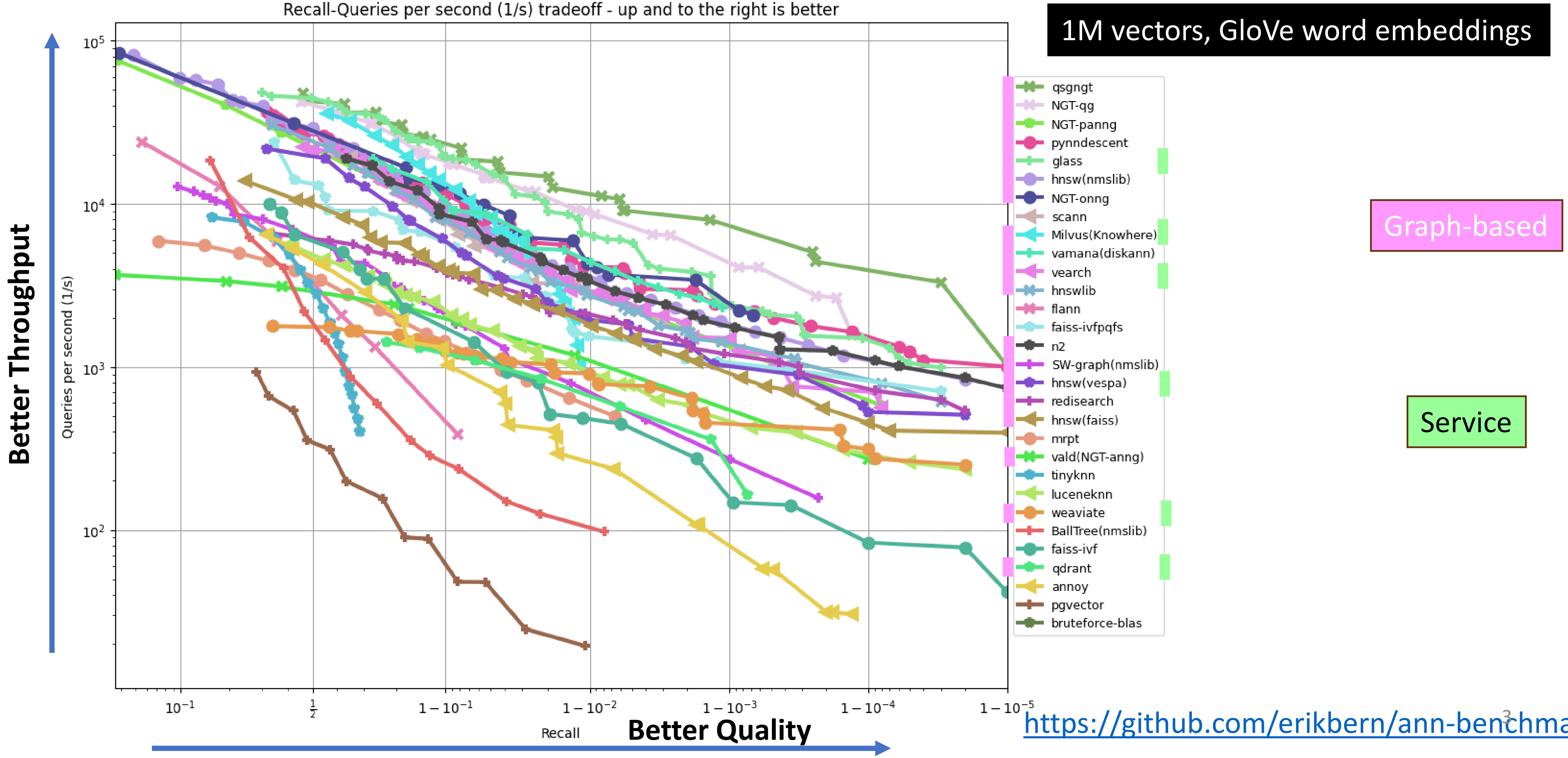
Harsha Vardhan Simhadri[1] HARSHASI@MICROSOFT.COM
George Williams[2] GWILLIAMS@IEEE.ORG
Martin Aumüller[3] MAAU@ITU.DK
Matthijs Douze[4] MATTHIJS@FB.COM
Artem Babenko[5] ARTEM.BABENKO@PHYSTECH.EDU
Dmitry Baranchuk[5] DBARANCHUK@YANDEX-TEAM.RU
Qi Chen[1] CHEQI@MICROSOFT.COM
Lucas Hosseini[4] LUCAS.HOSSEINI@GMAIL.COM
Ravishankar Krishnaswamy[1] RAKRI@MICROSOFT.COM
Gopal Srinivasa[1] GOPALSR@MICROSOFT.COM
Suhas Jayaram Subramanya[6] SUHASJ@CS.CMU.EDU
Jingdong Wang[7] WANGJINGDONG@BAIDU.COM

[1] Microsoft Research [2] GSI Technology [3] IT University of Copenhagen
[4] Meta AI Research [5] Yandex [6] Carnegie Mellon University [7] Baidu

PUFFINN
[Aumüller+, ESA 2019]

Billion-Scale ANN Challenge
[Aumüller+, NeurIPS 21, Competition]

From Million-Scale to Billion-Scale ANN

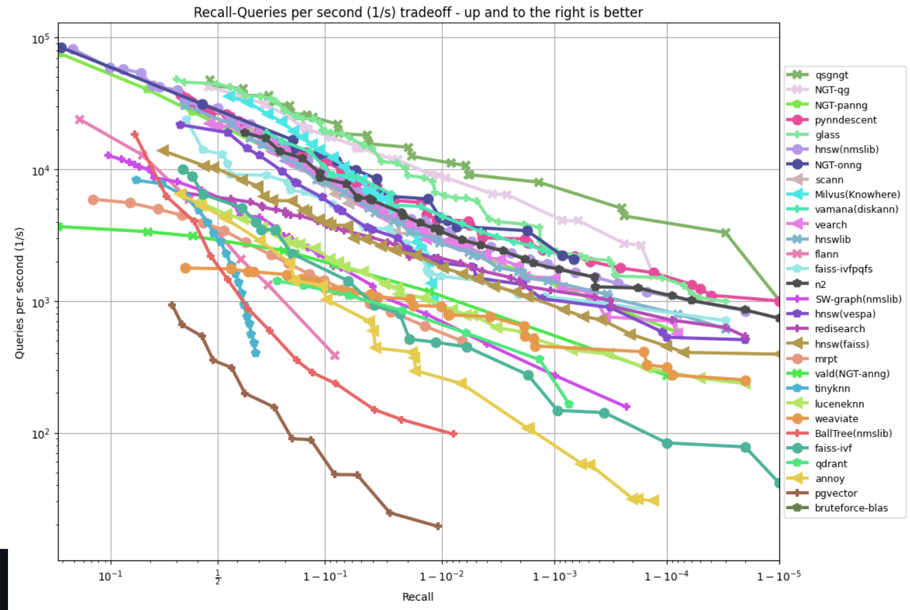


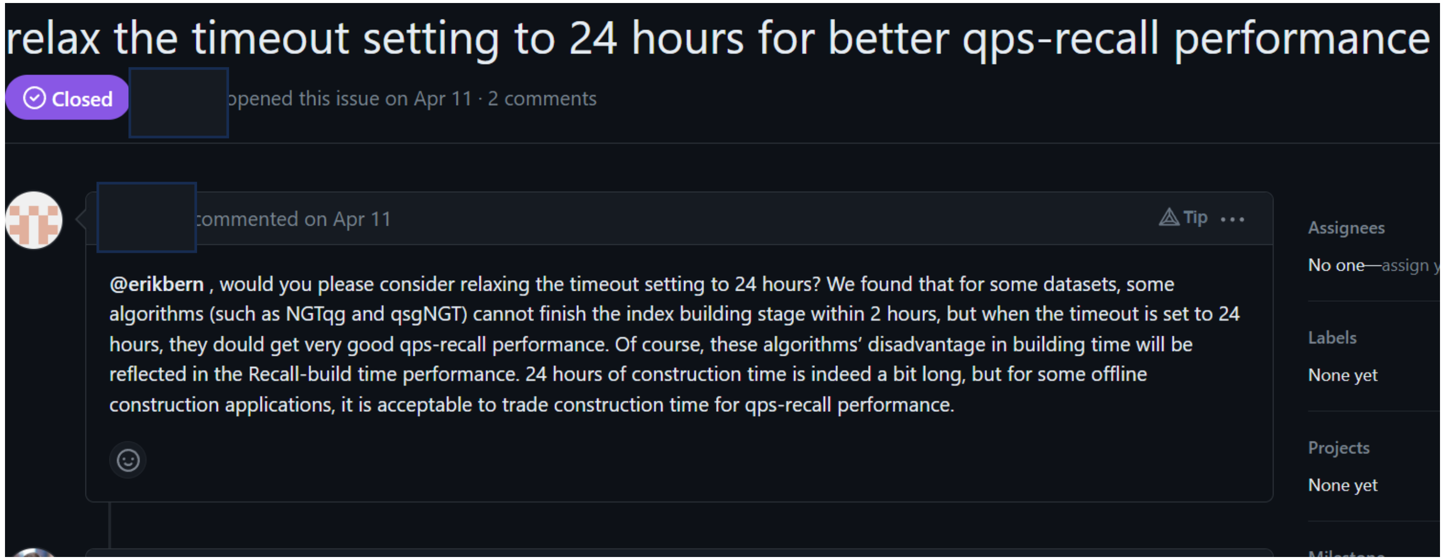
https://github.com/erikbern/ann-benchmarks

From Million-Scale to Billion-Scale ANN

Rules

- Index building + searching **single-threaded**
- **2 hours time limit**, container killed afterwards





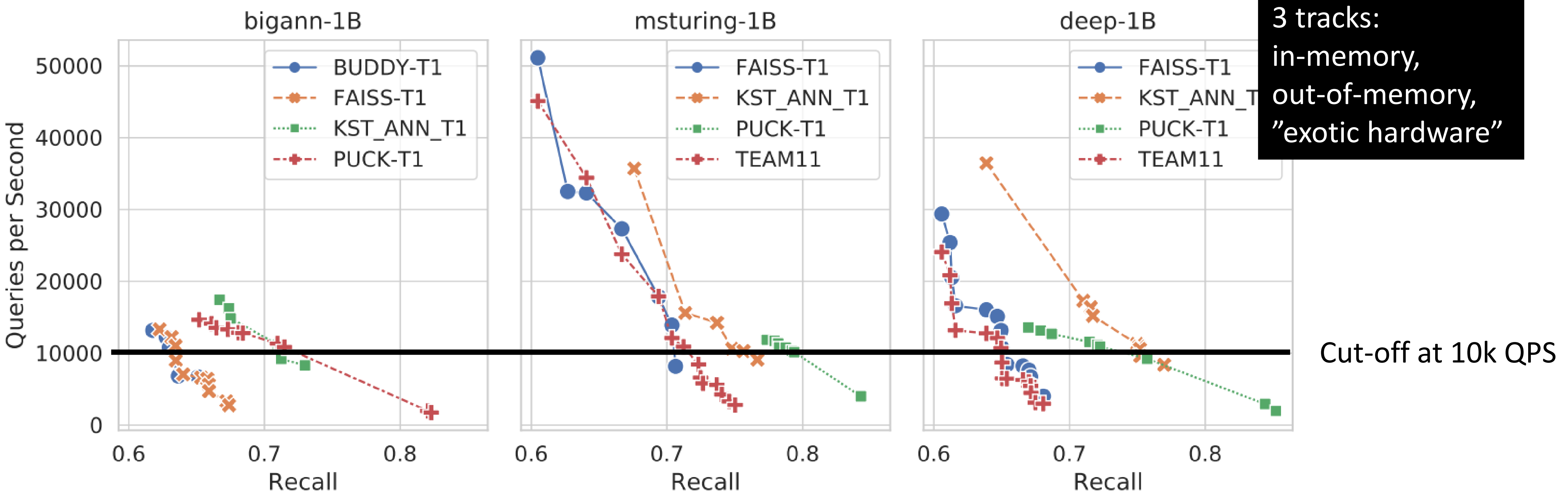
Q: Scaling up by 1000x?

2 hours → 2000 hours ~ 83 days

24 hours → 24000 hours ~ 3 years

(unrealistic scaling)

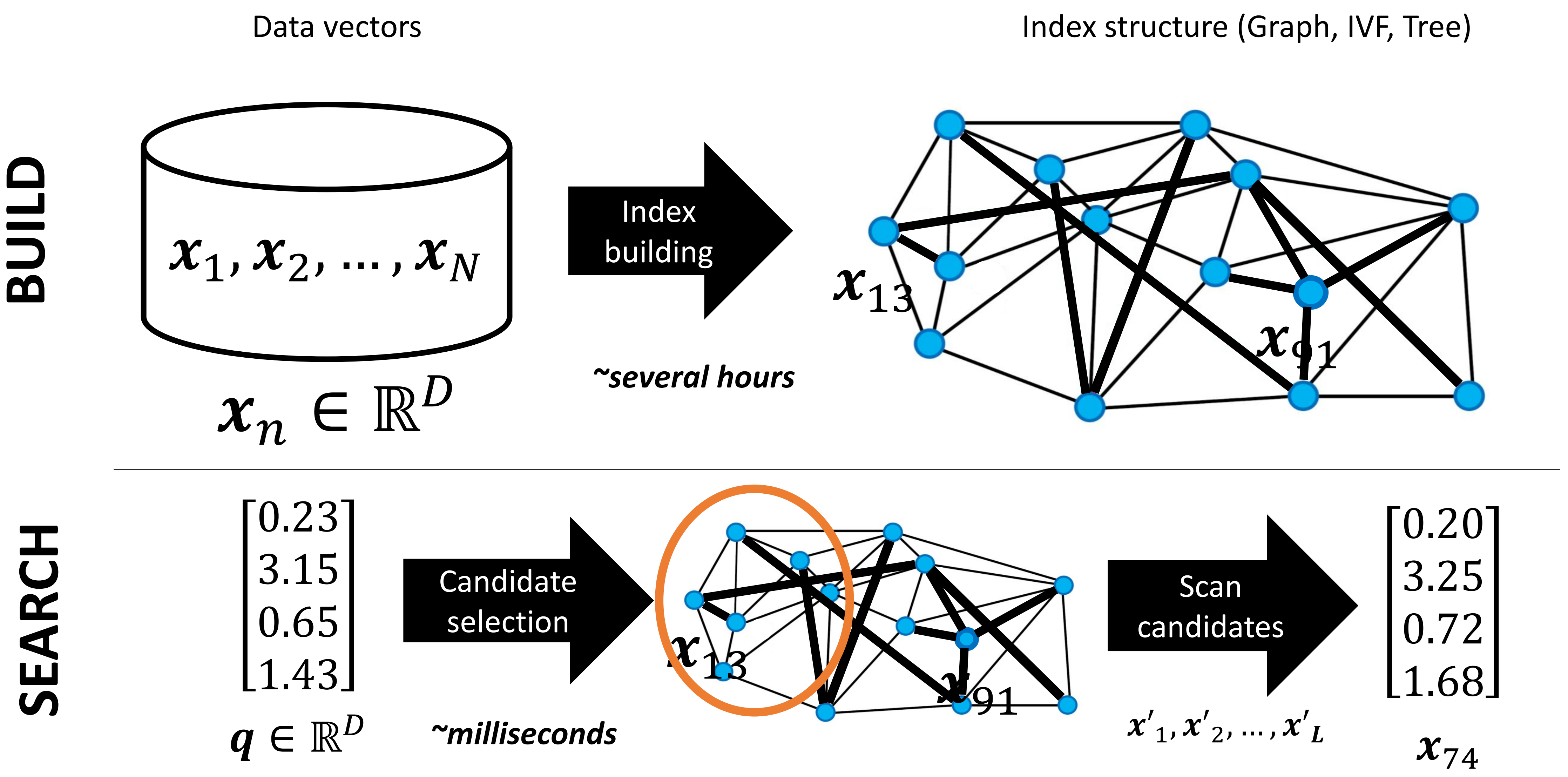
Billion-Scale ANN Challenge [Simhadri+, NeurIPS 2021]



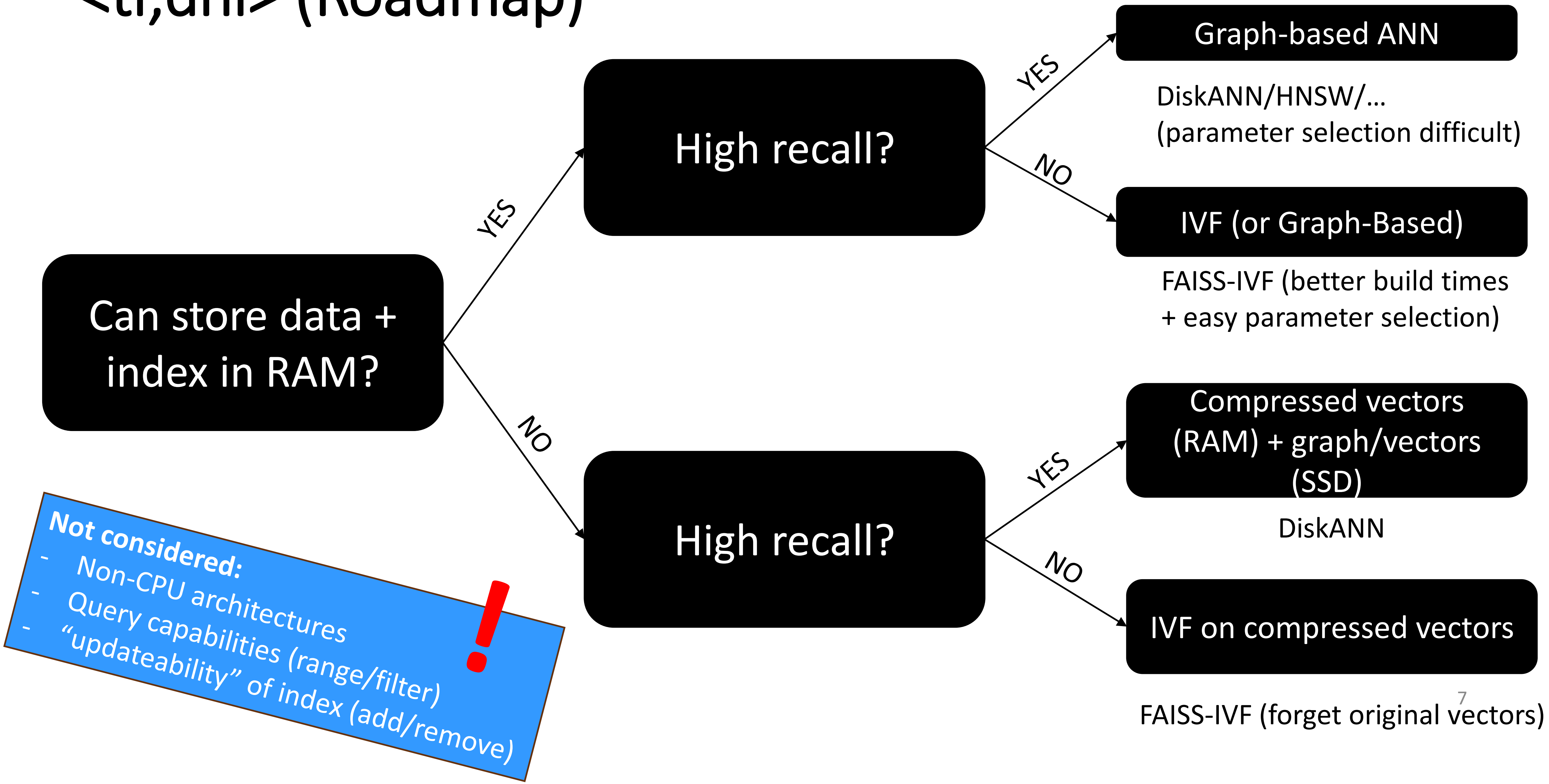
3 tracks:
in-memory,
out-of-memory,
”exotic hardware”

Many entries did not improve on baseline by much.

The ANN search pipeline



<tl;dnl> (Roadmap)



Billion-Scale Datasets

Meta AI: Image descriptors for copy detection

Dataset	Datatype	Dimensions	Distance	Range/k-NN	Base data	Sample data	Query data	Ground truth	Release terms
BIGANN	uint8	128	L2	k-NN	1B points	100M base points	10K queries	link	CC0
Facebook SimSearchNet++*	uint8	256	L2	Range	1B points	N/A	100k queries	link	CC BY-NC
Microsoft Turing-ANNS*	float32	100	L2	k-NN	1B points	N/A	100K queries	link	link to terms
Microsoft SPACEV*	int8	100	L2	k-NN	1B points	100M base points	29.3K queries	link	O-UDA
Yandex DEEP	float32	96	L2	k-NN	1B points	350M base points	10K queries	link	CC BY 4.0
Yandex Text-to-Image*	float32	200	inner-product	k-NN	1B points	50M queries	100K queries	link	CC BY 4.0

256 GB

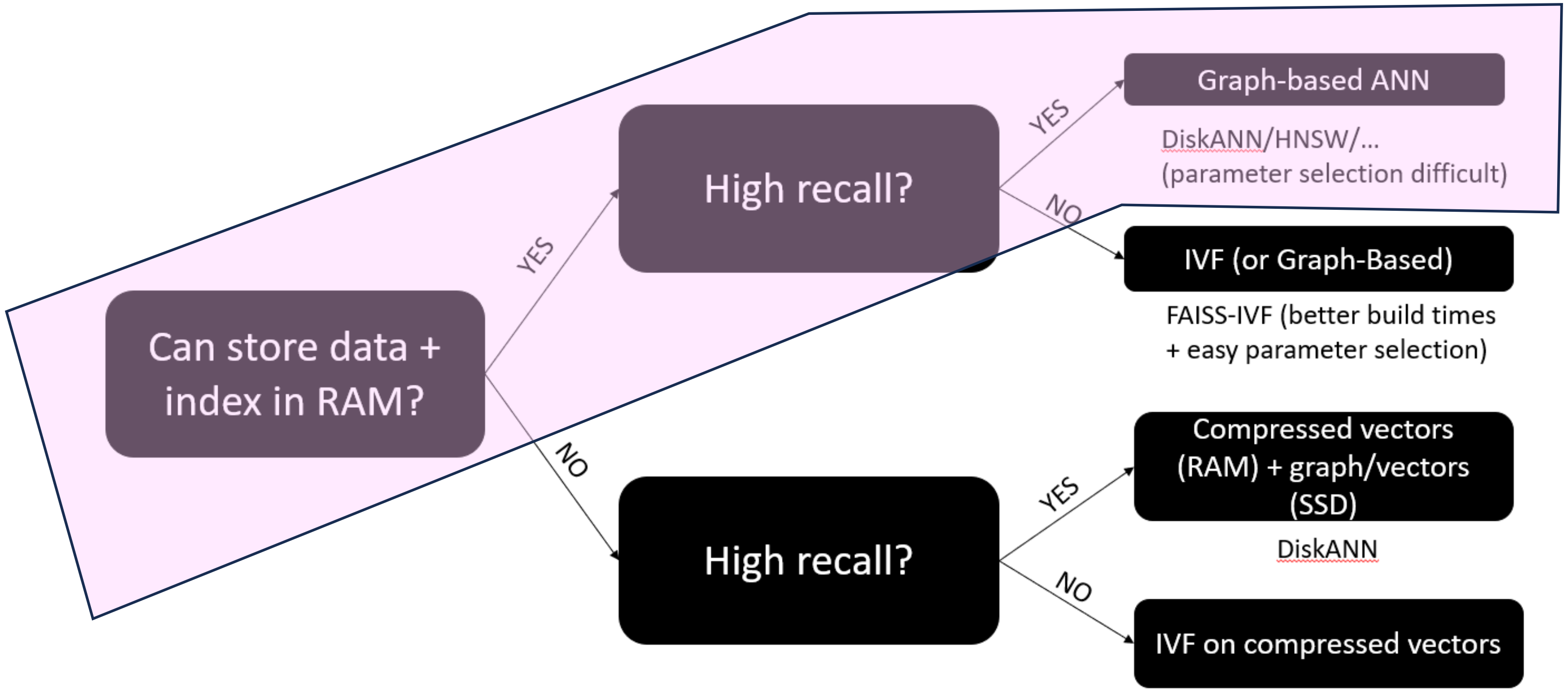
100 GB

800 GB

Microsoft Bing: Search string → Web documents

https://big-ann-benchmarks.com/

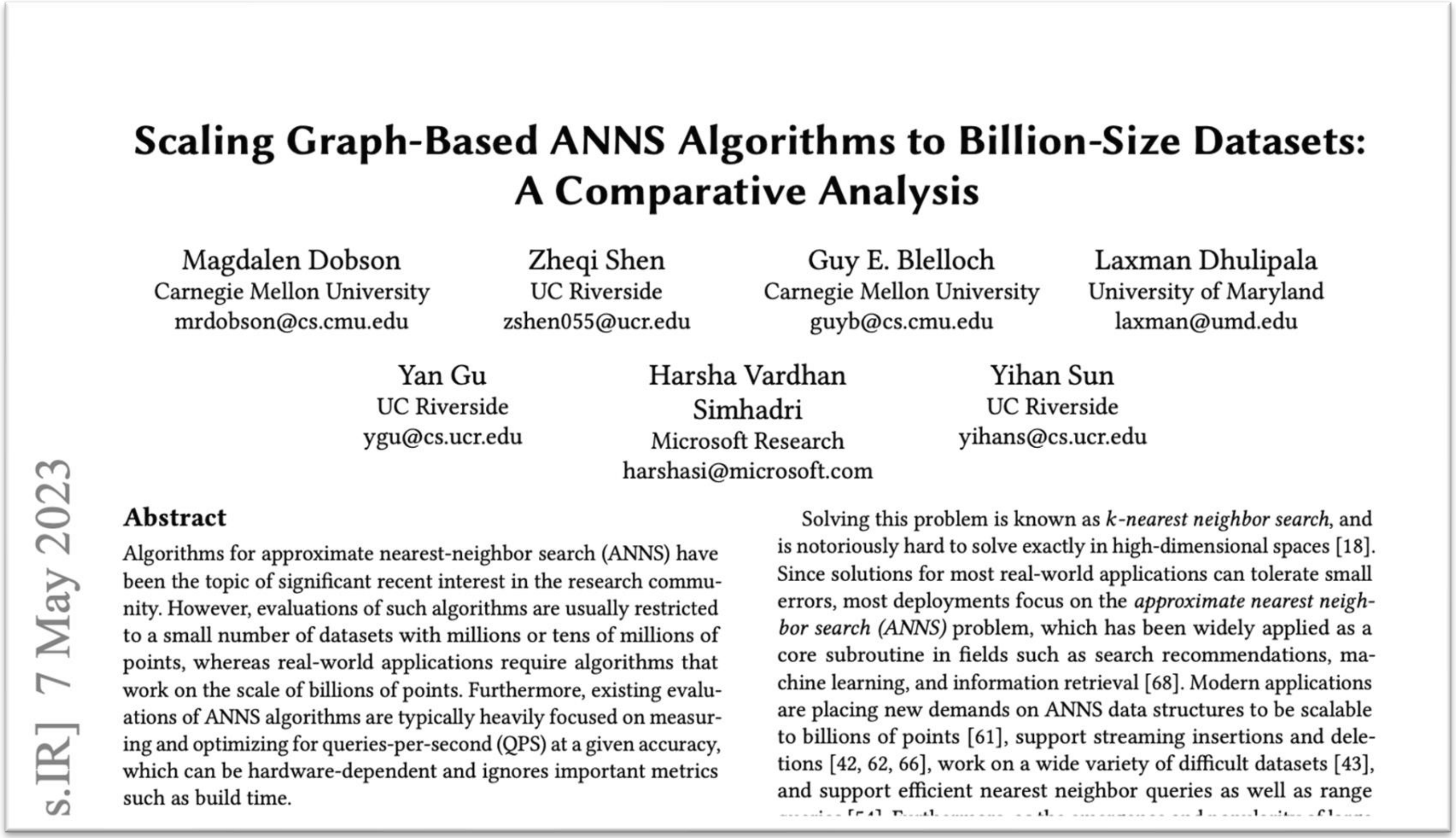
NeurIPS 2021 Challenge



High Resources, High Recall

Possible setup: Multi-Socket Xeon, 256 GB - 2TB of RAM

Scaling Graph-Based Approaches



Scaling Graph-Based ANNS Algorithms to Billion-Size Datasets: A Comparative Analysis

Magdalen Dobson
Carnegie Mellon University
mrdobson@cs.cmu.edu

Zheqi Shen
UC Riverside
zshen055@ucr.edu

Guy E. Blelloch
Carnegie Mellon University
guyb@cs.cmu.edu

Laxman Dhulipala
University of Maryland
laxman@umd.edu

Yan Gu
UC Riverside
ygu@cs.ucr.edu

Harsha Vardhan Simhadri
Microsoft Research
harshasi@microsoft.com

Yihan Sun
UC Riverside
yihans@cs.ucr.edu

Abstract

Algorithms for approximate nearest-neighbor search (ANNS) have been the topic of significant recent interest in the research community. However, evaluations of such algorithms are usually restricted to a small number of datasets with millions or tens of millions of points, whereas real-world applications require algorithms that work on the scale of billions of points. Furthermore, existing evaluations of ANNS algorithms are typically heavily focused on measuring and optimizing for queries-per-second (QPS) at a given accuracy, which can be hardware-dependent and ignores important metrics such as build time.

Solving this problem is known as *k-nearest neighbor search*, and is notoriously hard to solve exactly in high-dimensional spaces [18]. Since solutions for most real-world applications can tolerate small errors, most deployments focus on the *approximate nearest neighbor search (ANNS)* problem, which has been widely applied as a core subroutine in fields such as search recommendations, machine learning, and information retrieval [68]. Modern applications are placing new demands on ANNS data structures to be scalable to billions of points [61], support streaming insertions and deletions [42, 62, 66], work on a wide variety of difficult datasets [43], and support efficient nearest neighbor queries as well as range

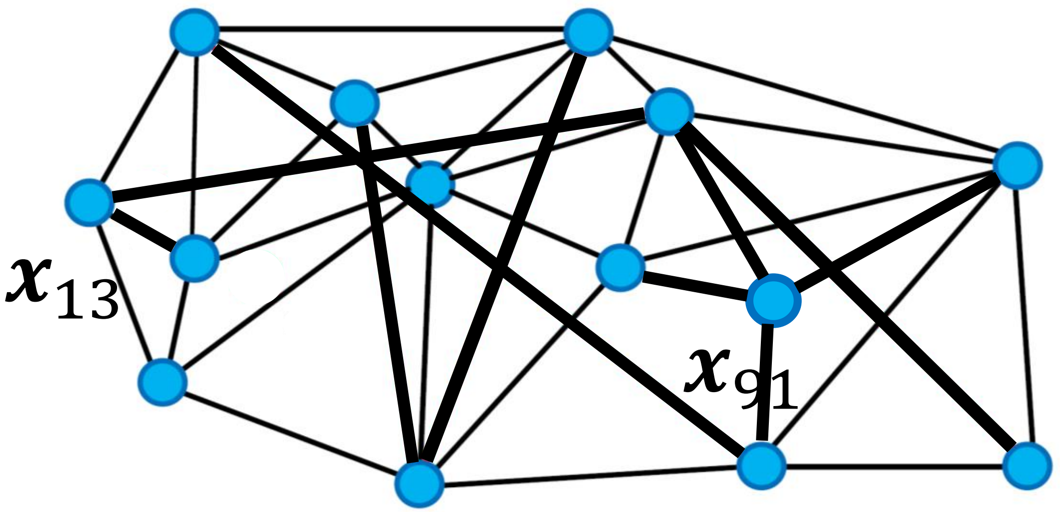
Machines

- Azure Msv2 (4 Xeon, 192 vCPUs, 2 TB RAM), $384 USD/day
- Azure Ev5 (2 Xeon, 96 vCPUs, 672 GB RAM), $144 USD/day

https://arxiv.org/pdf/2305.04359.pdf

Scaling Graph-Based Approaches

- **Recap**
  - Vectors are nodes
  - Connected to "diverse set of similar points" + long range edges
- **Incremental build**
  - Use search algorithm to find potential candidate neighbors
  - Prune these candidates



Index size?

~1B x "avg. degree of node"

Practically all algorithms enforce user-set bound!

Faster build?

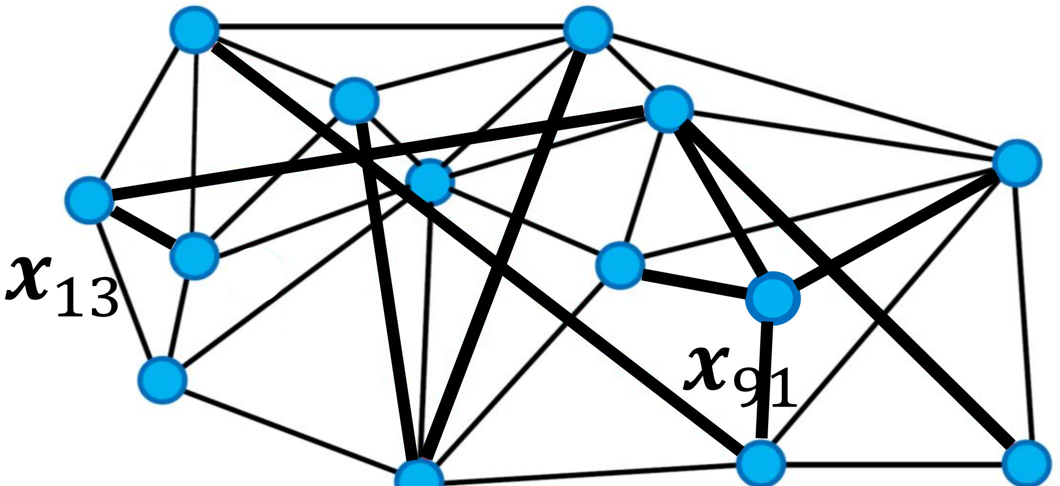
Smaller target degree + smaller beam width

Tradeoffs?

Need larger beam width to compensate for "worse build graph"

Parallelizing insertion

- Order all points arbitrarily
- For each point:
  - Carry out greedy search for nearest neighbor in “current graph”
  - Connect to *pruned* set of vertices found during the NN search



Algorithm 2: insert(p, s, R, L).

Input: Point p, starting point s, beam width L, degree bound R.
Output: Point p is inserted into the nearest neighbor graph.

$\mathcal{V}, \mathcal{K} \leftarrow \text{greedySearch}(p, s, L, 1)$
$\text{N}_{\text{out}}(p) \leftarrow \text{prune}(\mathcal{V})$
for $q \in N_{out}(p)$ **do**
  $\text{N}_{\text{out}}(q) \leftarrow \text{N}_{\text{out}}(q) \cup \{p\}$
  if $|N_{out}(q)| > R$ **then**
    $\text{N}_{\text{out}}(q) \leftarrow \text{prune}(\text{N}_{\text{out}}(q))$

Thread-safety?

Algorithm 3: batchBuild$(\mathcal{P}, s, R, L)$.

Input: Point set $\mathcal{P}$, starting point s, beam width L, degree bound R.
Output: A nearest neighbor graph consisting of all points in $\mathcal{P}$ and start point s.

$i \leftarrow 0$
while $2^i \leq |\mathcal{P}|$ **do**
  parallel for $j \in [2^i, 2^{i+1})$ **do**
    $\mathcal{V}, \mathcal{K} \leftarrow \text{greedySearch}(\mathcal{P}[j], s, L)$
    $\text{N}_{\text{out}}(\mathcal{P}[j]) \leftarrow \text{prune}(\mathcal{V})$
  $\mathcal{B} \leftarrow \bigcup_{j=2^i}^{2^{i+1}-1} \text{N}_{\text{out}}(P[j])$
  parallel for $b \in \mathcal{B}$ **do**
    `// Find` $\mathcal{N}$ `as all points in the current batch that added` b `as their neighbors`
    $\mathcal{N} \leftarrow \{\mathcal{P}[j] \mid j \in [2^i, 2^{i+1}) \wedge b \in \text{N}_{\text{out}}(\mathcal{P}[j])\}$
    $\text{N}_{\text{out}}(b) \leftarrow \text{N}_{\text{out}}(b) \cup \mathcal{N}$
    if $|N_{out}(b)| > R$ **then** $\text{N}_{\text{out}}(b) \leftarrow \text{prune}(\text{N}_{\text{out}}(b))$
  $i \leftarrow i + 1$

“prefix doubling”

Understanding parameters

- **Index building**
  - Degree bound R
    - upper limit on index size
  - Beam width L (building)
    - better neighbors
  - Pruning factor (α)
    - ”diversified neighbors”
- **Searching**
  - Beam width R_{search}

Sensitive to parameter choices & they are difficult to choose!

> ***DiskANN*** The main parameters for the DiskANN index build are (1) the degree bound R, (2) the beam width L used during insertion, and (3) the pruning parameter α. In our experiments, we found that no single parameter setting was optimal for all recall regimes, and that there were significant tradeoffs in other recall values when maximizing for recall above .99; thus we chose to use parameters optimized for the .94-.97 range. Note that for TEXT2IMAGE, which minimizes negative inner product, the α value must be less than one in order to select for a denser graph.

> 1-million experiments. Due to scalability issues, we could not report results on the 25GB experiments for HCNNG (indexing time exceeded 24 hours) and KGRAPH/DPG (could not reach an acceptable accuracy, i.e., recall > 0.8). Due to the low performance on the 25GB experiments of VAMANA and EFANNA (indexing a 25GB dataset required over 300GB RAM and indexing a 100GB dataset needed more than the 1.4TB of available memory) and NSG (since it uses EFANNA as a base graph), we excluded them from experiments with larger datasets.
> **Indexing Time.** Figure 1 shows that on the 25GB dataset, ELPIS can build its index 2x and 5x faster than HNSW and NSG, respectively, and over an order of magnitude faster than the other competitors. On the other dataset sizes, ELPIS is twice faster than its second best competitor, HNSW. Since NSG [50] is built on top of EFANNA [48], we include the time to build both indexing structures. Although VAMANA [111] builds the graph based on a random initial graph, it spends more than 7 hours to create the Deep25GB index. This is

https://www.vldb.org/pvldb/vol16/p1548-azizi.pdf

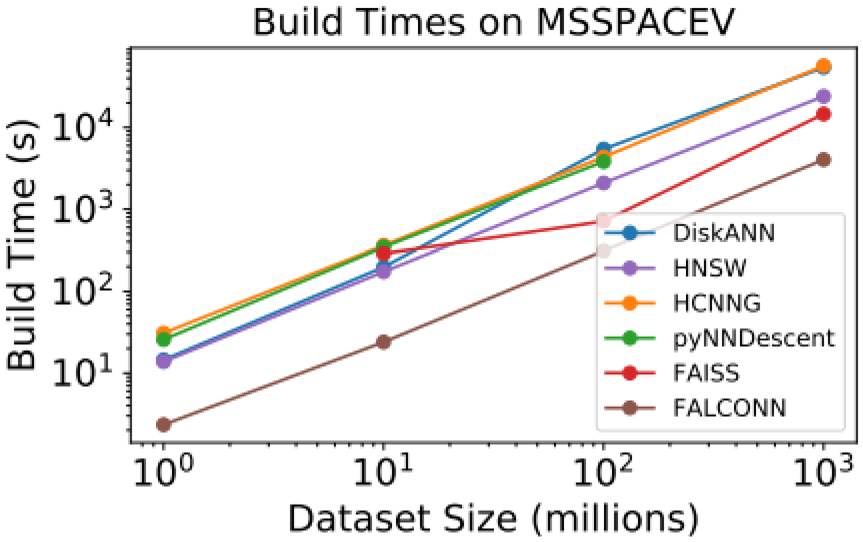
Build times & scaling

	BIGANN	MSSPACEV	TEXT2IMAGE	SSNPP
DiskANN	$R = 64, L = 128, \alpha = 1.2$	$R = 64, L = 128, \alpha = 1.2$	$R = 64, L = 128, \alpha = .9$	$R = 150, L = 400, \alpha = 1.2$

Degree bound

Beam width

Billion-scale: Index size not more than 4R GB (e.g., 256GB, 600GB)



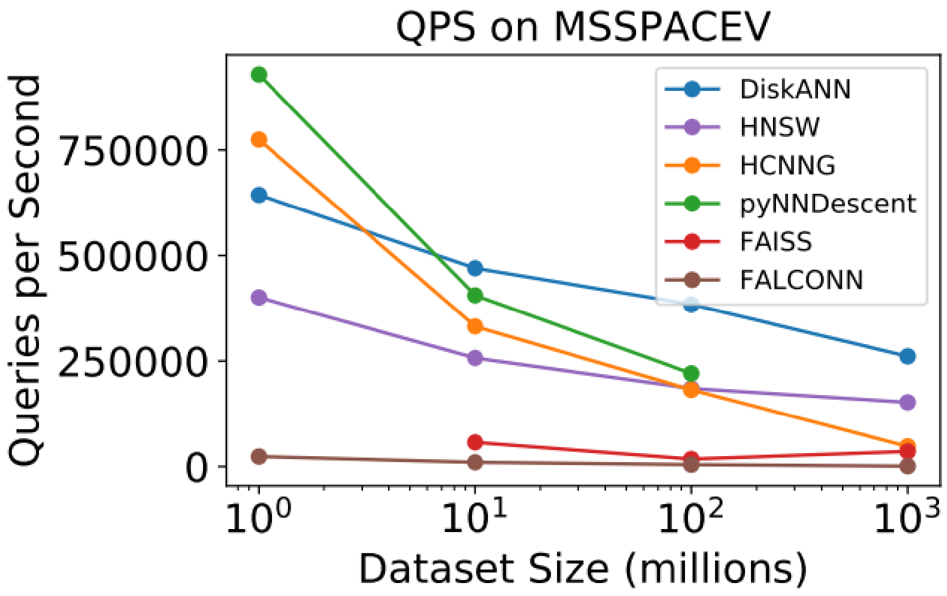
	BIGANN	MSSPACEV	TEXT2IMAGE	SSNPP
DiskANN	11.0	15.1	61.6	83.1
HNSW	9.2	6.7	14.9	91.6
HCNNG	8.6	15.8	21.4	19.0
FAISS	5.2	4.1	4.5	4.5
FALCONN	1.75	1.12	1.45	1.42

Table 1: Build times (hours) on billion-scale datasets.

10x increase → 11-12x build time increase

Parallelizing search

- Usually parallelization over queries (inter-query parallelism)
- Not so much in focus
- Beam width selection: “trial-and-error”

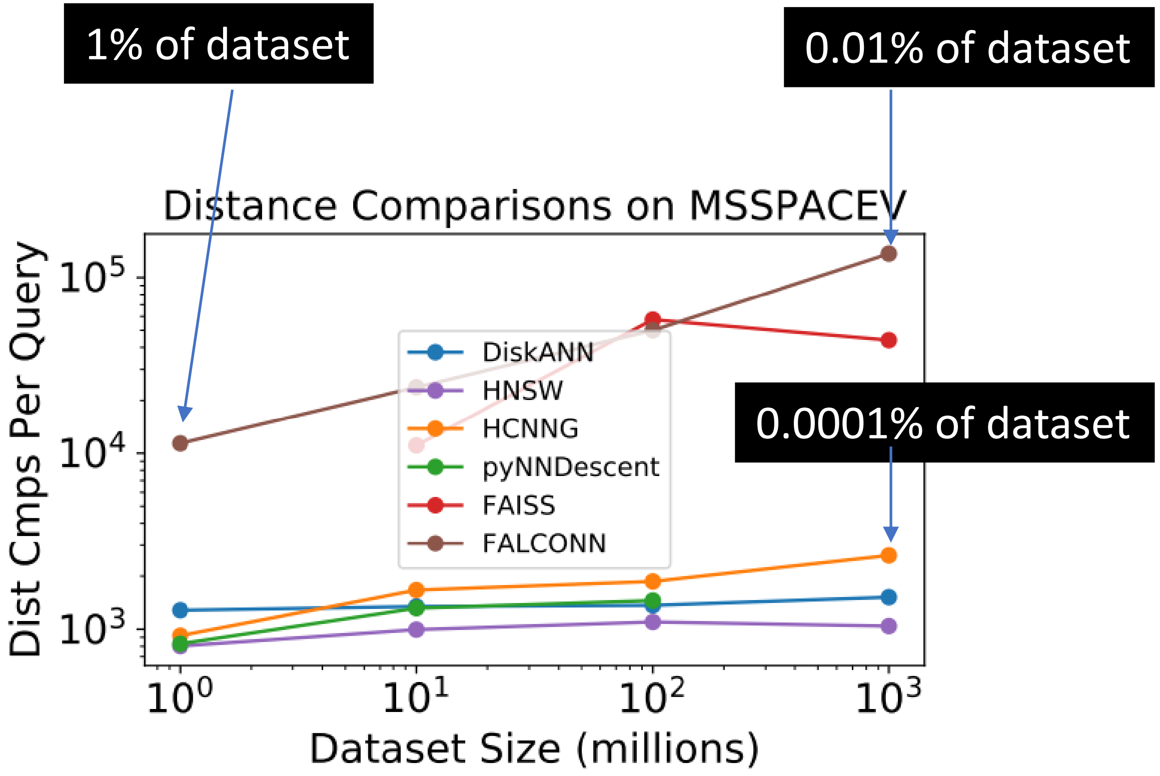


(b) QPS for fixed recall (.8) as dataset size increases.

Scaling: dataset 1000x larger → queries 2x slower

Summary

- **Advantages**
  - Good scaling of *#candidates*
  - Unparalleled performance in high-recall regime
- **Disadvantages**
  - Influence of parameter choices difficult to predict
  - High index building times (but "almost out-of-box")



(c) Distance comparisons per query for fixed recall (.8) as the dataset size increases.

How to get started (DiskANN)

[DiskANN: Simhadri+, NeurIPS19]

```
FROM ubuntu:jammy


RUN apt update
RUN apt install -y software-properties-common
RUN add-apt-repository -y ppa:git-core/ppa
RUN apt update
RUN DEBIAN_FRONTEND=noninteractive apt install -y git
make cmake g++ libaio-dev libgoogle-perftools-dev
libunwind-dev clang-format libboost-dev
libboost-program-options-dev libmkl-full-dev
libcpprest-dev python3.10

RUN git clone https://github.com/microsoft/DiskANN.git
WORKDIR /home/app/DiskANN
RUN pip3 install virtualenv build
RUN python3 -m build
RUN pip install dist/diskannpy-0.5.
0-cp310-cp310-linux_x86_64.whl
WORKDIR /home/app
```

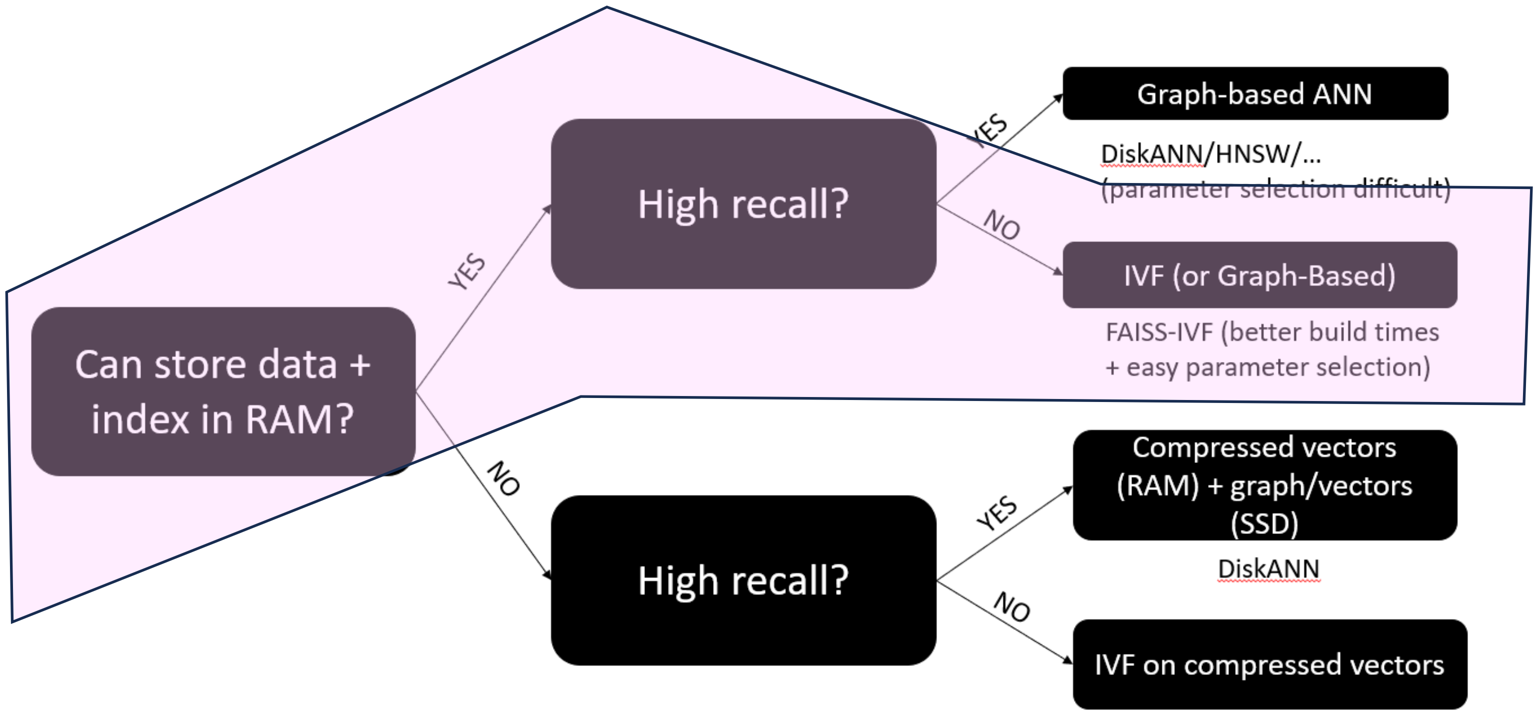
```python
import numpy as np
import diskannpy

class diskann:
    def fit(self, ds, L, R):
        """Build index for dataset `ds` with `R` degree, `L` beam width."""
        diskannpy.build_memory_index(
            data = ds.get_dataset_fn(),
            distance_metric = 'l2',
            vector_dtype = np.int8,
            complexity=L,
            graph_degree=R,
            num_threads = 64,
            alpha=1.2,
            use_pq_build=False,
            num_pq_bytes=0, #irrelevant given use_pq_build=False
            use_opq=False
        )

        print('Loading index..')
        self.index = diskannpy.StaticMemoryIndex(
            distance_metric = 'l2',
            vector_dtype = np.int8,
            num_threads = 64, #to allocate scratch space for up to 64 search threads
            initial_search_complexity = 100
        )
        print('Index ready for search')

    def query(self, X, k, Ls):
        """Carry out a batch query for k-NN of query set X."""
        self.res, self.query_dists = self.index.batch_search(X, k, Ls, 64)
```

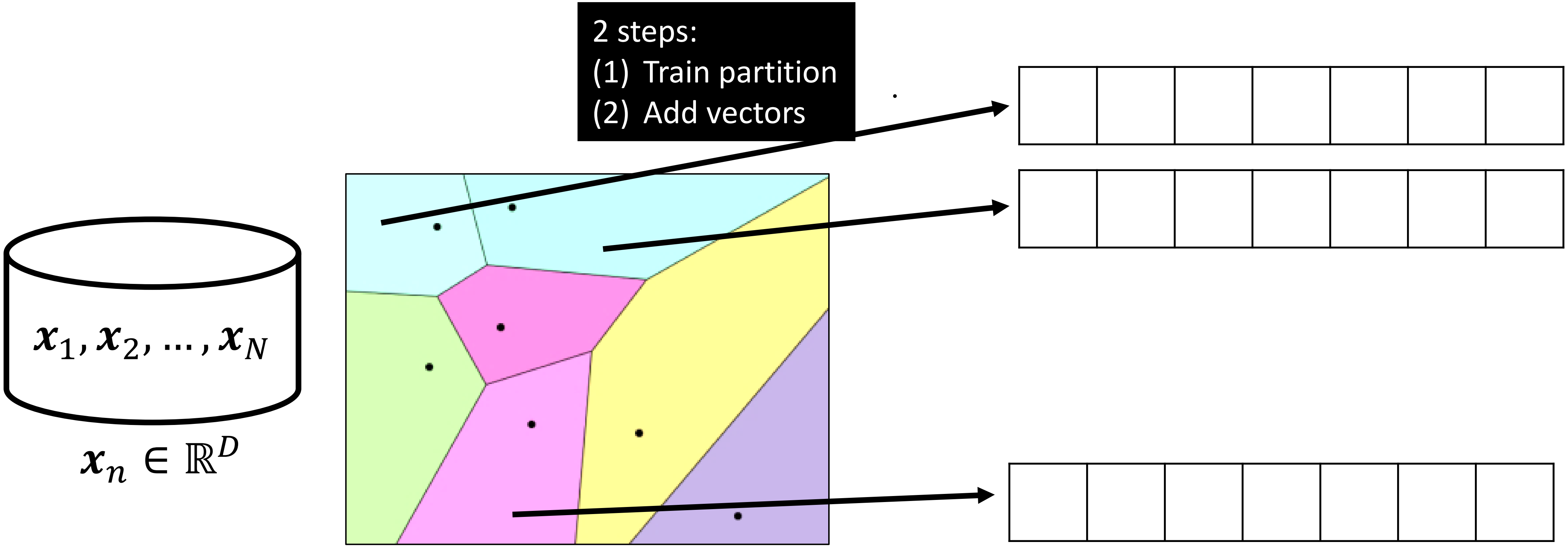
Official Documentation: https://github.com/Microsoft/DiskANN
Python examples: https://github.com/harsha-simhadri/big-ann-benchmarks,



High Resources, Low Recall

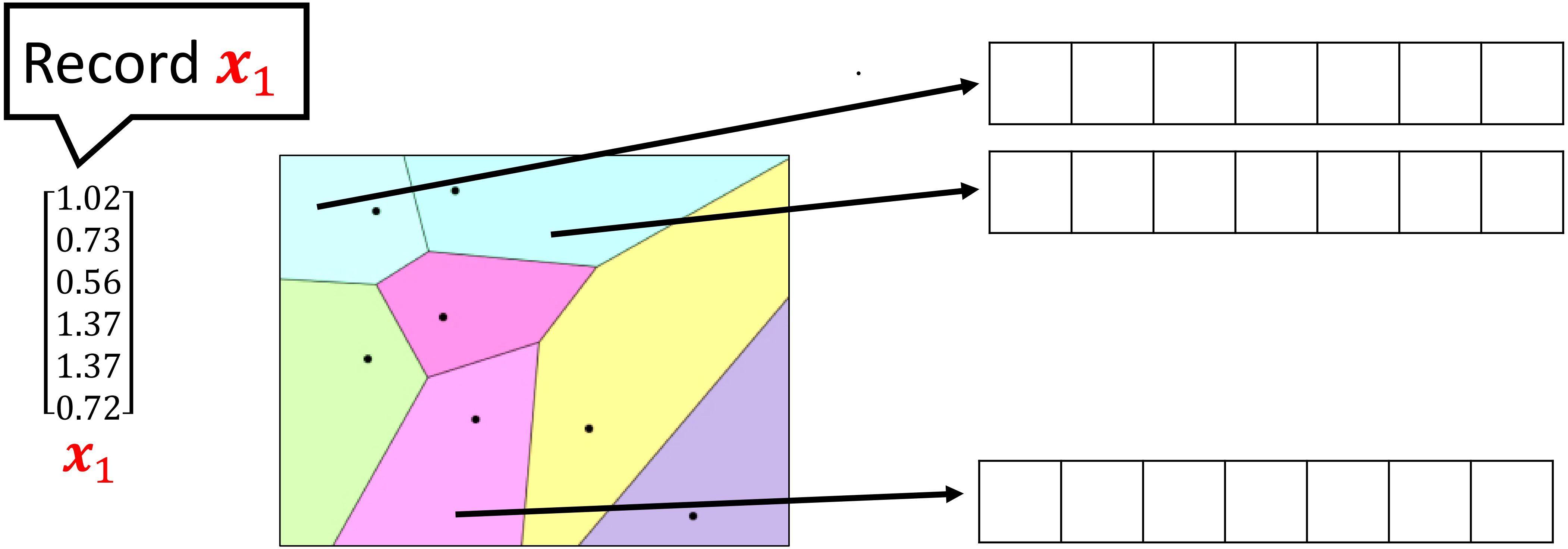
Possible setup: Multi-Socket Xeon, 256 GB - 2TB of RAM

IVF-based solutions (“inverted file index”)



Finding a space partition: <u>Clustering-based (k-means)</u>, LSH-based, …

IVF: insert a vector



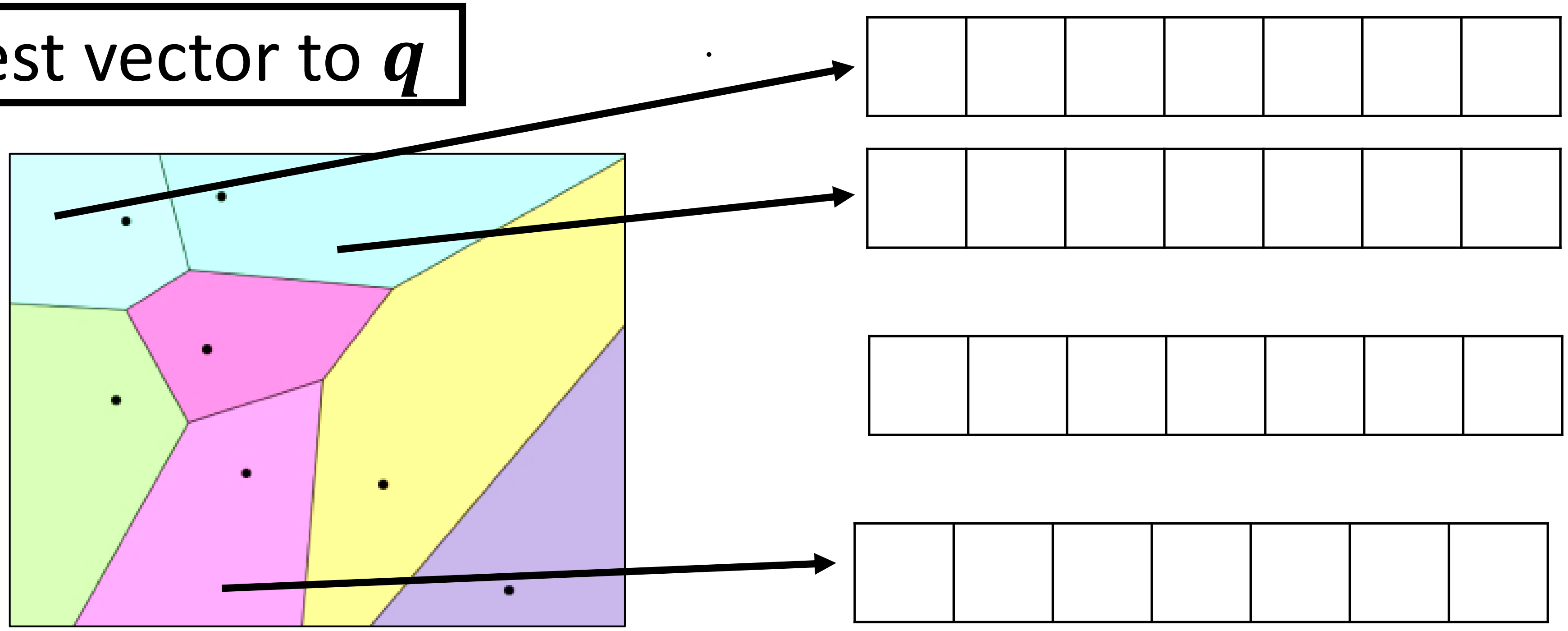
Cells: all points closest to given centroid ("Voronoi cells")
Build parameter: #clusters

IVF: search

Find the nearest vector to $\boldsymbol{q}$

$$\begin{bmatrix}0.54\\2.35\\0.82\\0.42\\0.14\\0.32\end{bmatrix}$$

$\boldsymbol{q}$

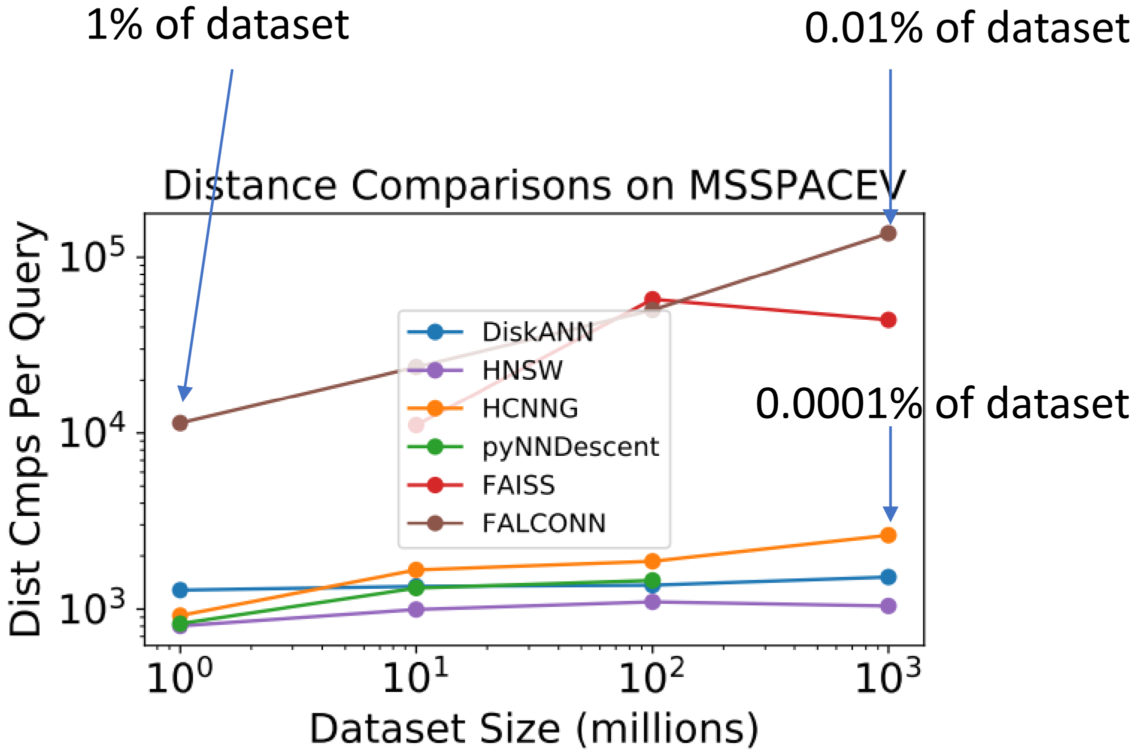


Search parameter: #clusters to inspect

Candidates: #clusters inspected * avg. cluster size

How to choose parameters?

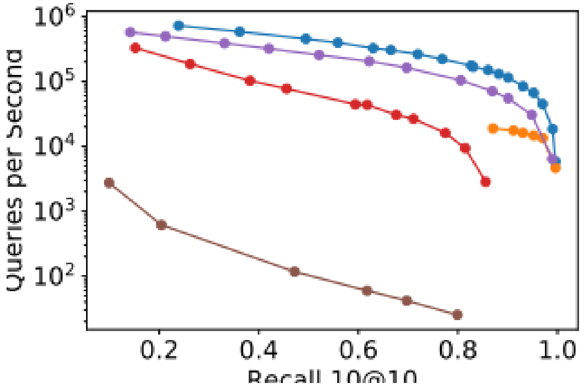
- **Goal**: inspect 0.0001% of dataset for 1B vectors ➔ 1000 points
- **Back-of-the-envelope calculation:**
  - ~1000 points per cluster
  - → need a million clusters
- **Making this practical**
  - Build an index on centroids
- **Standard solution**
  - Build a graph on top of the centroids
  - Alternatives: hierarchical k-means



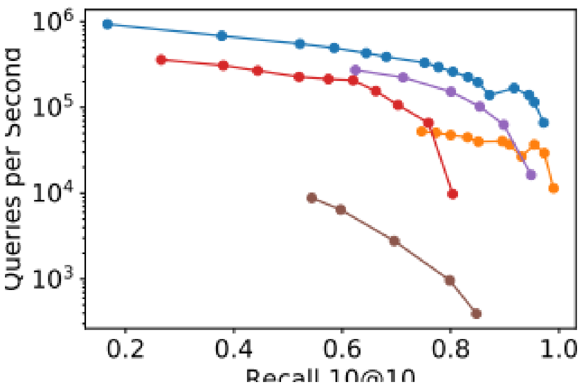
(c) Distance comparisons per query for fixed recall (.8) as the dataset size increases.

IVF-based approaches

- **Advantages**
  - Predictable index size and relatively easy to understand parameters
  - Strong implementations available
  - GPU-based solutions
- **Disadvantages**
  - Many candidates necessary in the high-recall regime
  - Quantization necessary to limit impact of these distance computations



(a) BIGANN-1B



(b) MSSPACEV-1B

Great documentation with code examples!

https://github.com/facebookresearch/faiss/wiki

How to get started?

facebookresearch / **faiss** Public

- Install via `conda install -c pytorch faiss-cpu`

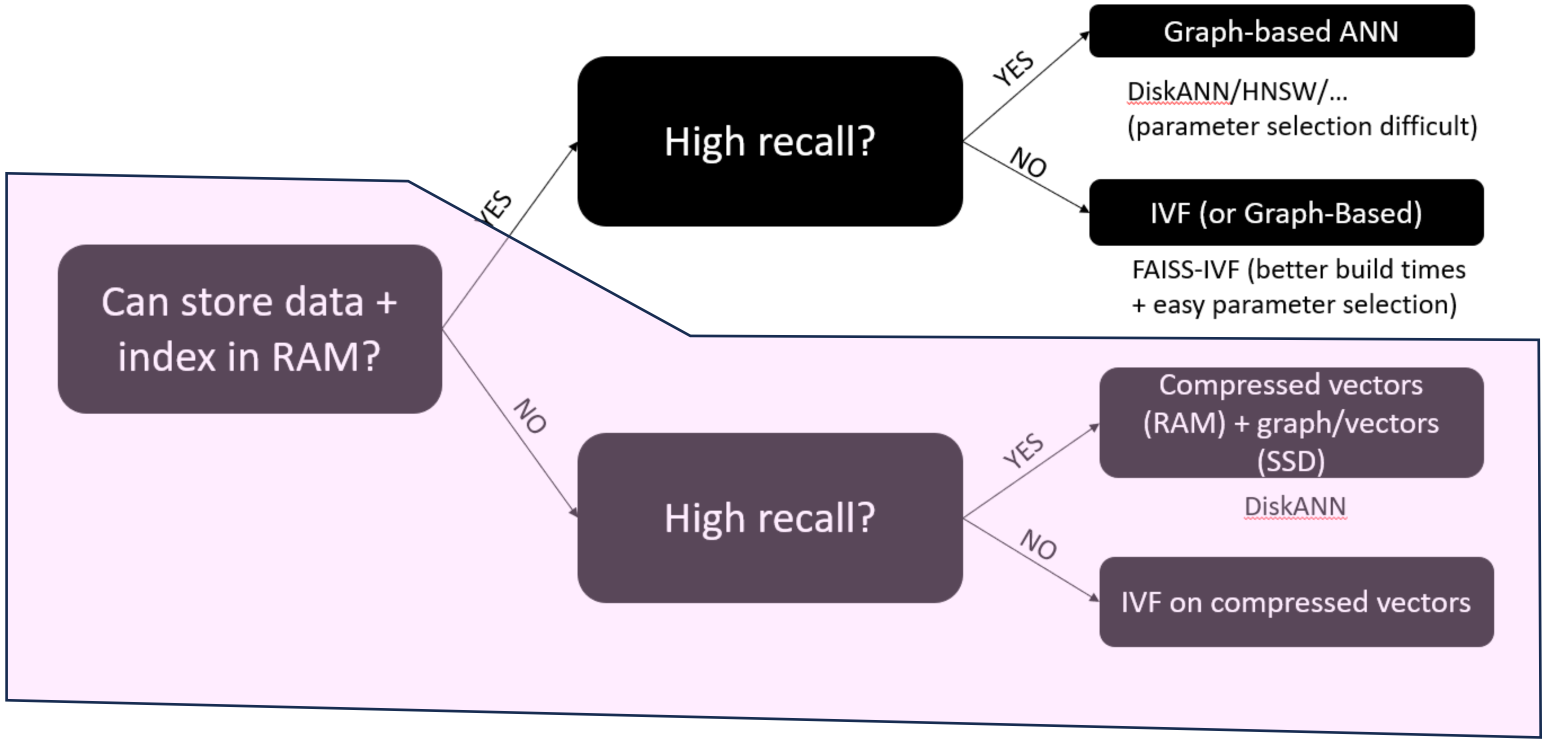
```
nlist = 100
k = 4
quantizer = faiss.IndexFlatL2(d)  # the other index
index = faiss.IndexIVFFlat(quantizer, d, nlist)
assert not index.is_trained
index.train(xb)
assert index.is_trained

index.add(xb)                  # add may be a bit slower as well
D, I = index.search(xq, k)     # actual search
print(I[-5:])                  # neighbors of the 5 last queries
index.nprobe = 10              # default nprobe is 1, try a few more
D, I = index.search(xq, k)
print(I[-5:])                  # neighbors of the 5 last queries
```

index = faiss.index_factory(128, "PCA64,IVF16384_HNSW32,Flat")

Index factories available!

https://github.com/facebookresearch/faiss/wiki/Guidelines-to-choose-an-index



Billion-Scale ANN with limited resources

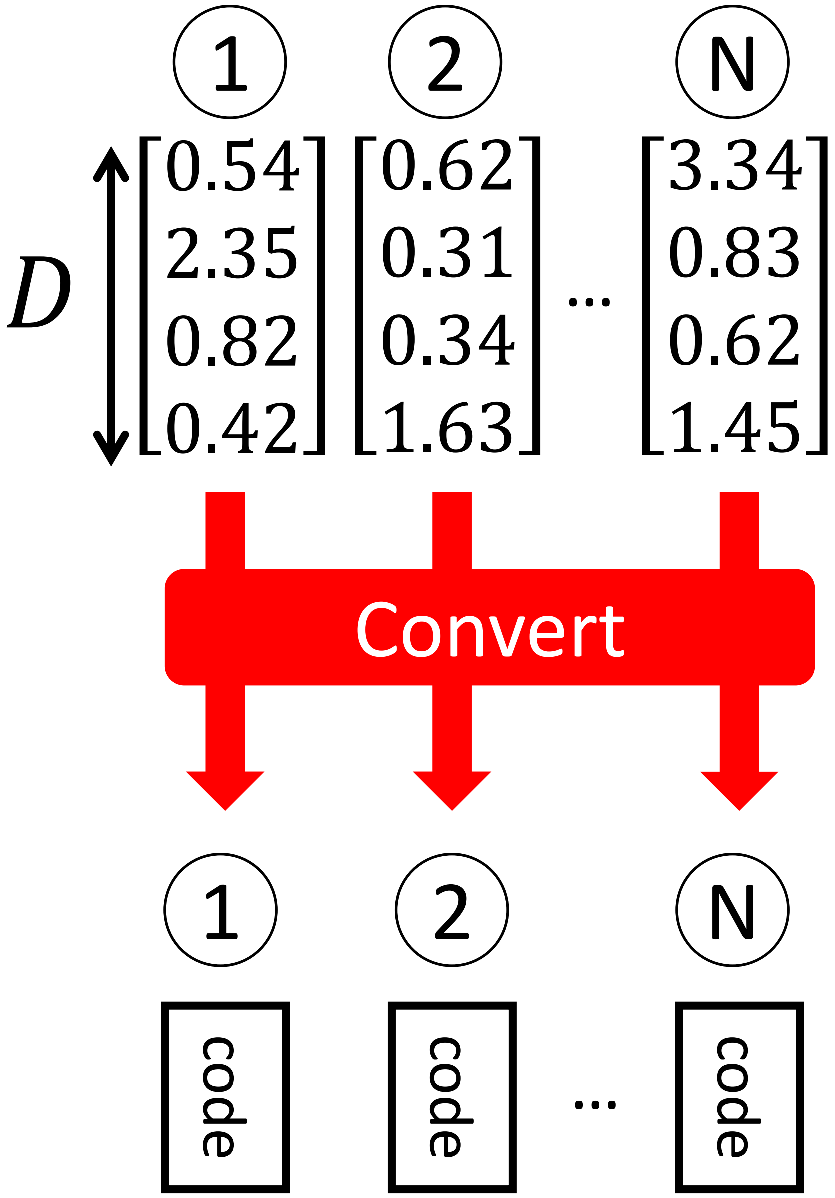
Interlude: Vector Quantization

Quantization techniques

	BIGANN	MSSPACEV	TEXT2IMAGE	SSNPP
DiskANN	$R = 64, L = 128, \alpha = 1.2$	$R = 64, L = 128, \alpha = 1.2$	$R = 64, L = 128, \alpha = .9$	$R = 150, L = 400, \alpha = 1.2$
HNSW	$m = 32, efc = 128, \alpha = .82$	$m = 32, efc = 128, \alpha = .83$	$m = 32, efc = 128, \alpha = 1.1$	$m = 75, efc = 400, \alpha = .82$
HCNNG	$T = 30, Ls = 1000, s = 3$	$T = 50, Ls = 1000, s = 3$	$T = 30, Ls = 1000, s = 3$	$T = 50, Ls = 1000, s = 3$
pyNNDescent	$K = 40, Ls = 100,$ $T = 10, \alpha = 1.2$	$K = 60, Ls = 100,$ $T = 10, \alpha = 1.2$	$K = 60, Ls = 100,$ $T = 10, \alpha = .9$	$K = 60, Ls = 1000,$ $T = 10, \alpha = 1.4$
FAISS	OPQ64_128, IVF1048576_HNSW32, PQ128x4fsr	OPQ64_128, IVF1048576_HNSW32, PQ64x4fsr	OPQ64_128, IVF1048576_HNSW32, PQ128x4fsr	OPQ64_128, IVF1048576_HNSW32, PQ64

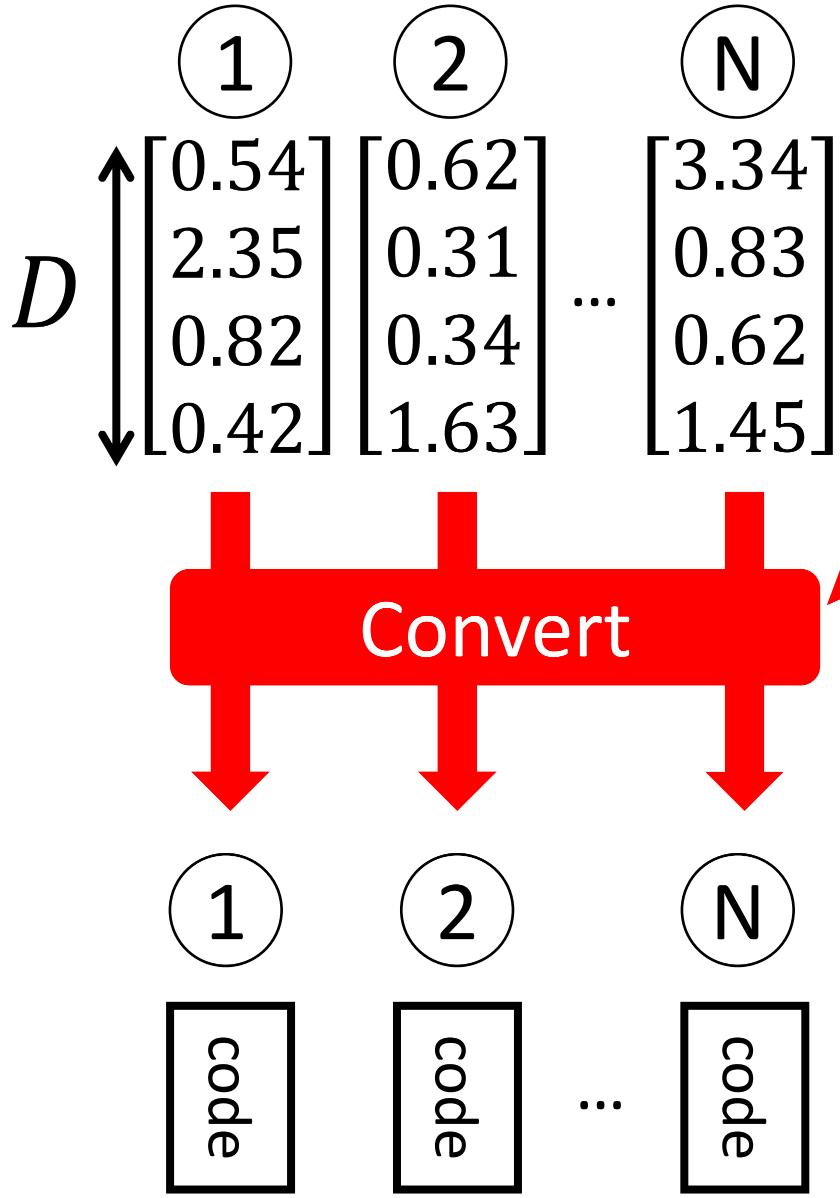
Cluster with 1M centroids, using HNSW to index the centroids

Basic idea



- Need $4ND$ byte to represent N real-valued vectors using floats
- If N or D is too large, we cannot read the data on memory
  - ✓ E.g., 512 GB for $D = 128, N = 10^9$
- Convert each vector to a **short-code**
- Short-code is designed as memory-efficient
  - ✓ E.g., 4 GB for the above example, with 32-bit code
- Run search for short-codes

Basic idea



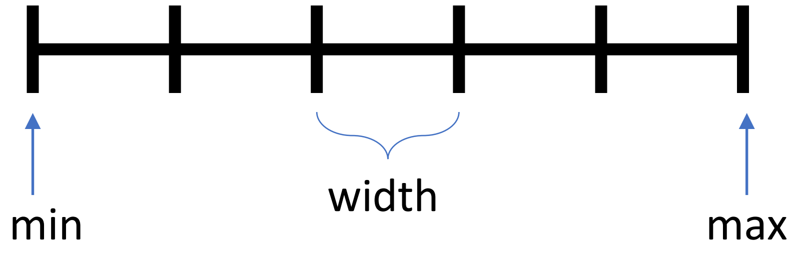
- Need $4ND$ byte to represent N real-valued vectors

What kind of conversion is preferred?

1. The "distance" between two codes can be calculated
2. The distance can be computed quickly
3. That distance approximates the distance between the original vectors (e.g., L_2)
4. Sufficiently small length of codes can achieve the above three criteria

Quantization Techniques

- **Low precision**
  - work with fp16 instead of 32/64 bit floats
- **Scalar quantization**
  - split up $[\min, \max]$ into K equidistant parts



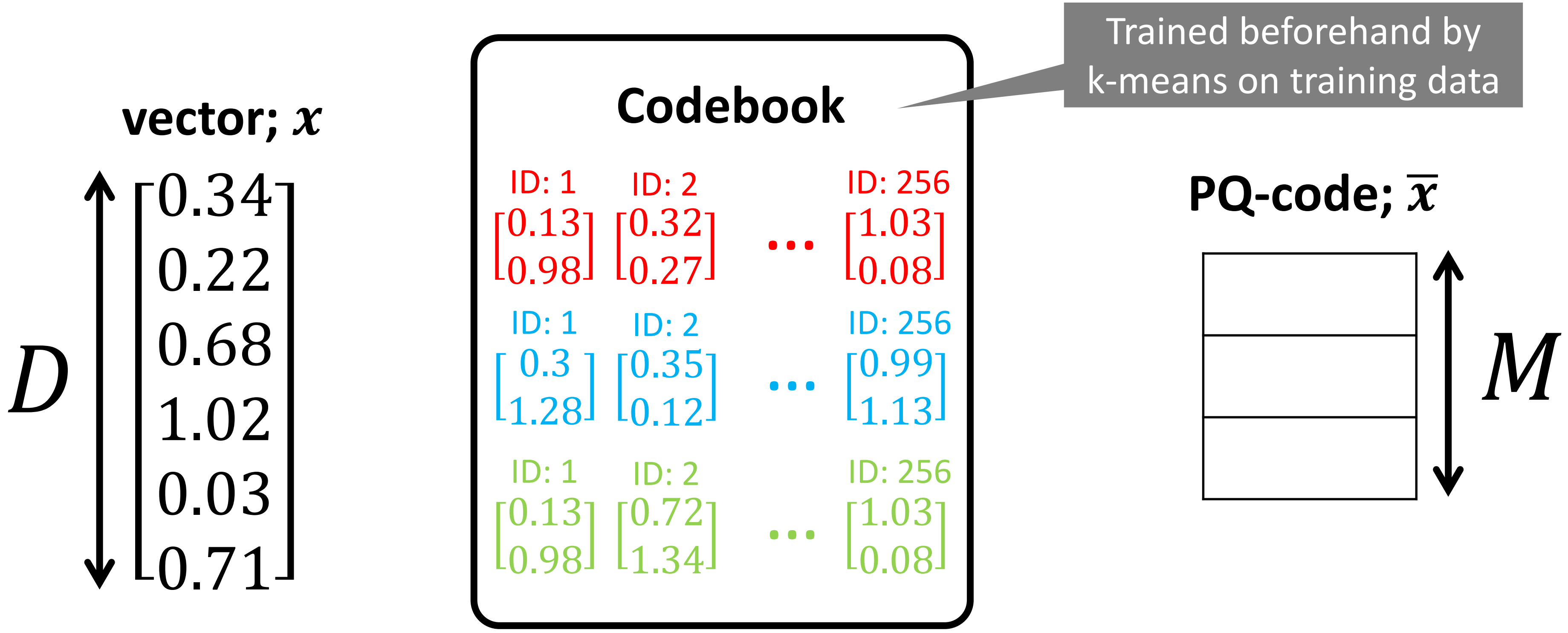
Interval $[0,3]$ split up into 6 parts

$$\begin{bmatrix}0.54\\2.35\\0.82\\0.42\end{bmatrix} \Rightarrow \begin{bmatrix}1\\5\\1\\0\end{bmatrix}$$

- **(binary/locality-sensitive) Hashing**
  - Apply hashing to embed into lower dimensional space
- **Product quantization**

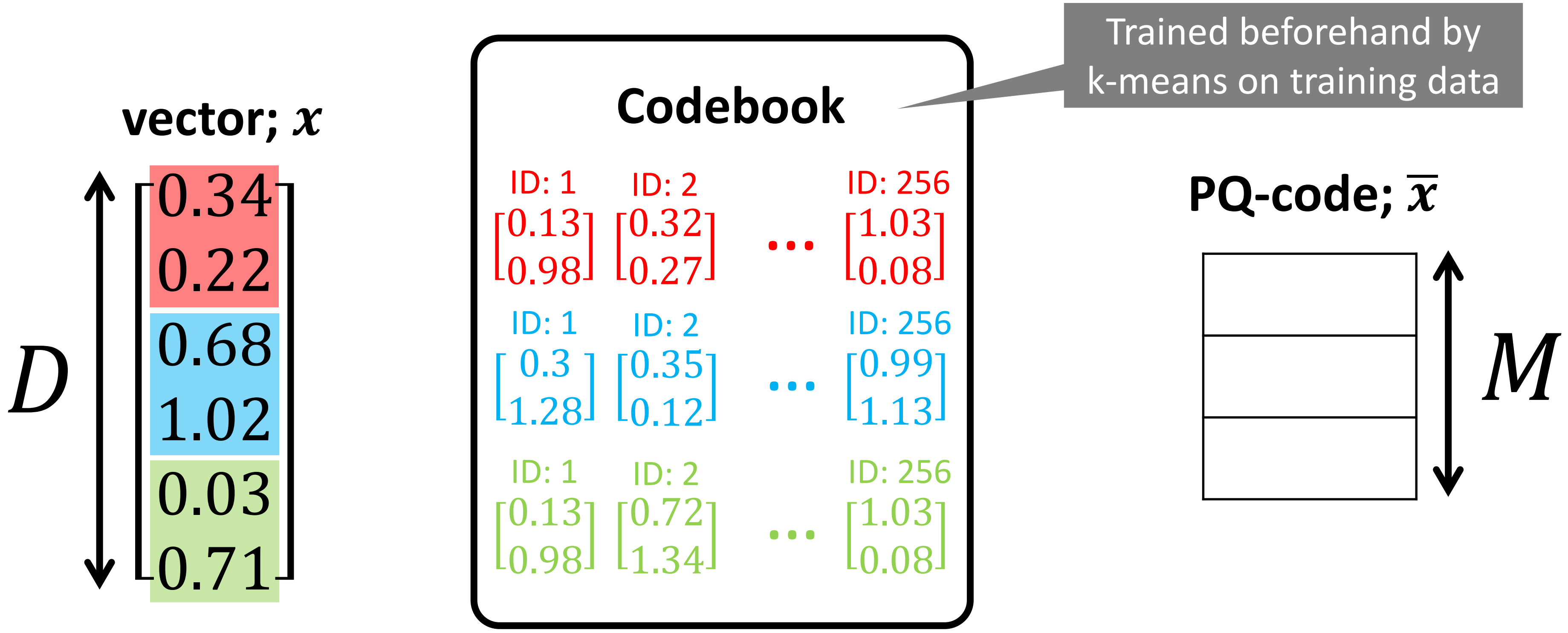
Product Quantization; PQ [Jégou+, TPAMI 2011]

- Split a vector into sub-vectors, and quantize each sub-vector



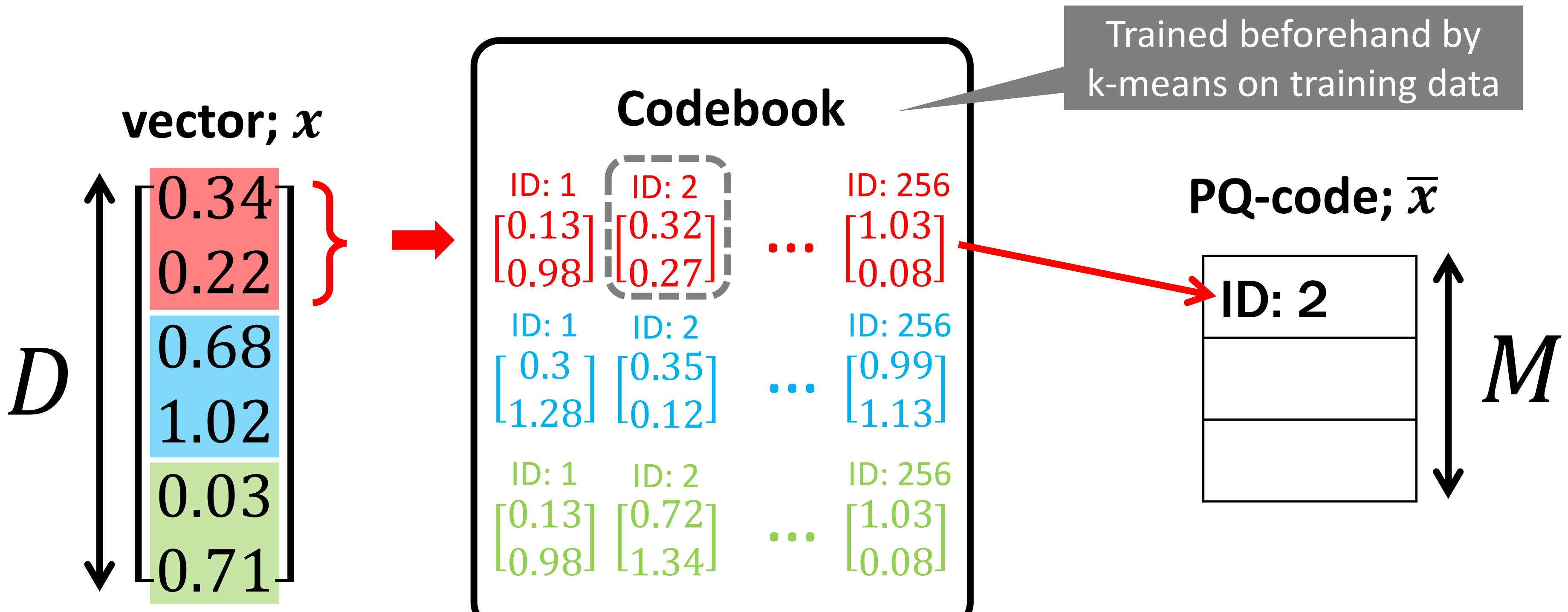
Product Quantization; PQ [Jégou+, TPAMI 2011]

- Split a vector into sub-vectors, and quantize each sub-vector



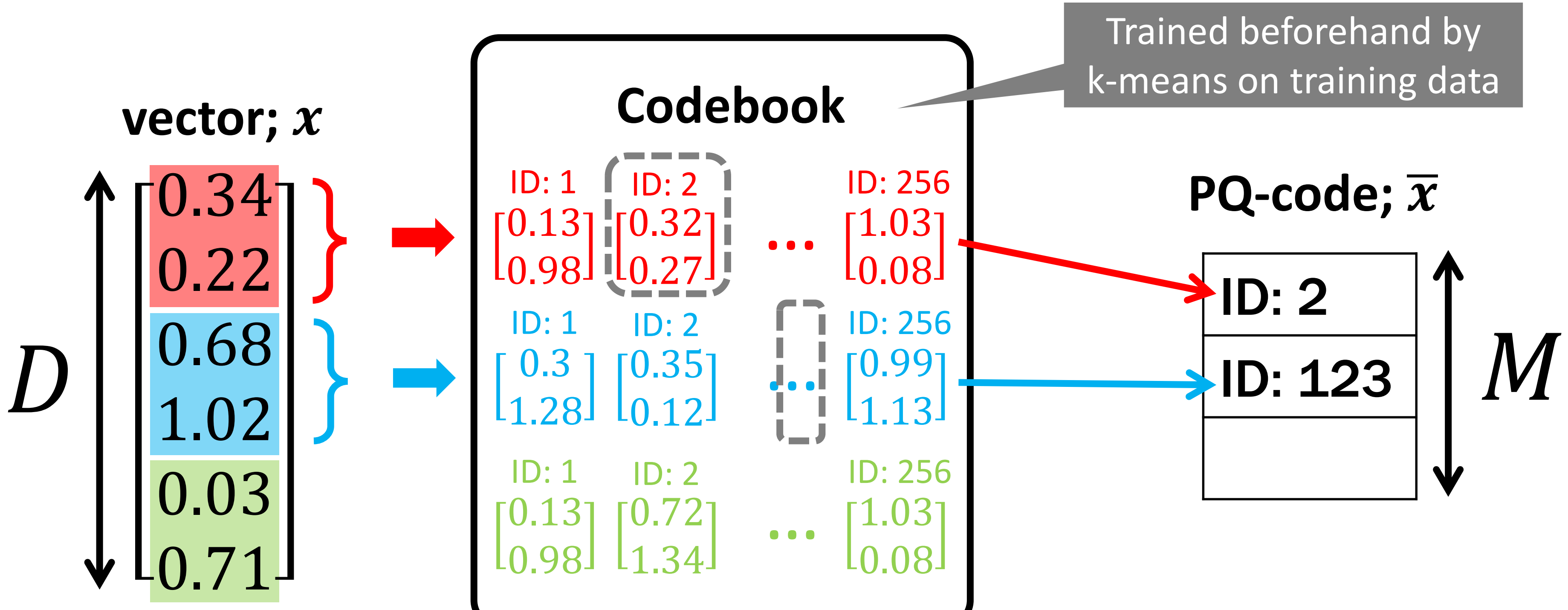
Product Quantization; PQ [Jégou+, TPAMI 2011]

- Split a vector into sub-vectors, and quantize each sub-vector



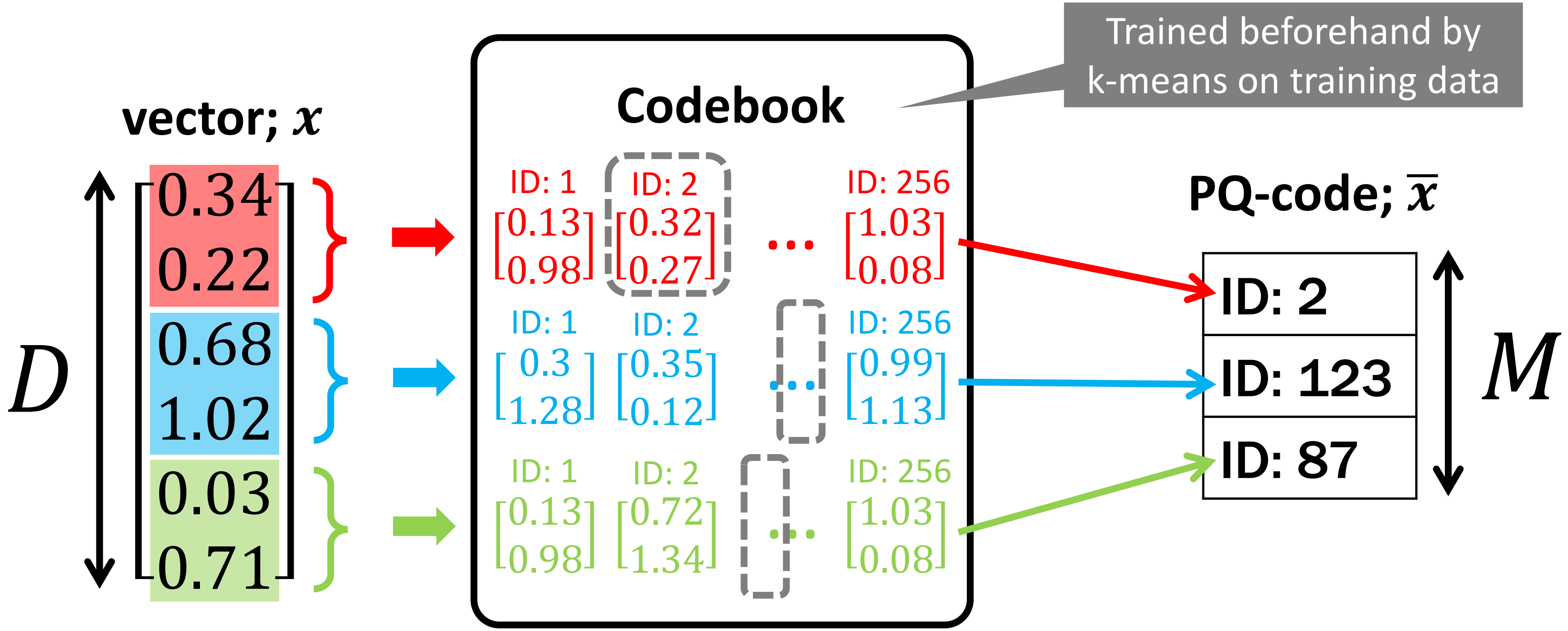
Product Quantization; PQ [Jégou, TPAMI 2011]

- ➢ Split a vector into sub-vectors, and quantize each sub-vector



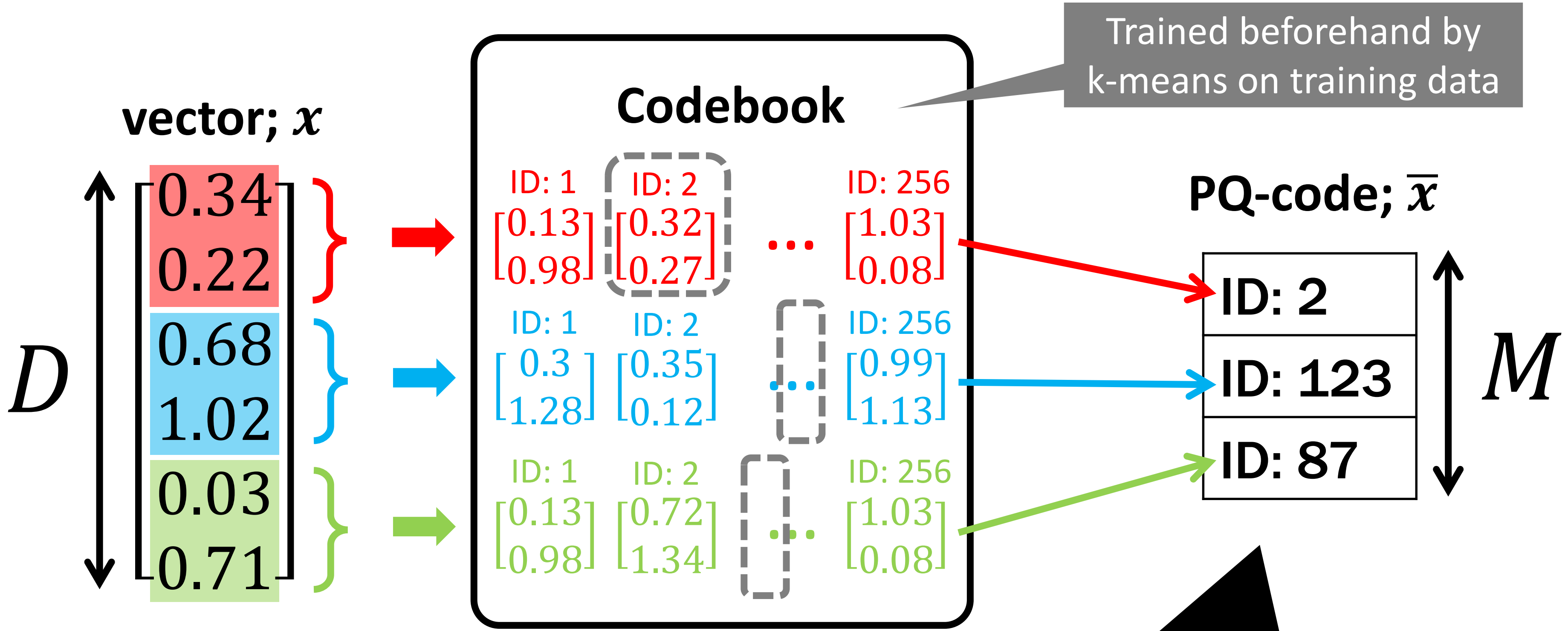
Product Quantization; PQ [Jégou, TPAMI 2011]

➢ Split a vector into sub-vectors, and quantize each sub-vector



Product Quantization; PQ [Jégou, TPAMI 2011]

- Split a vector into sub-vectors, and quantize each sub-vector

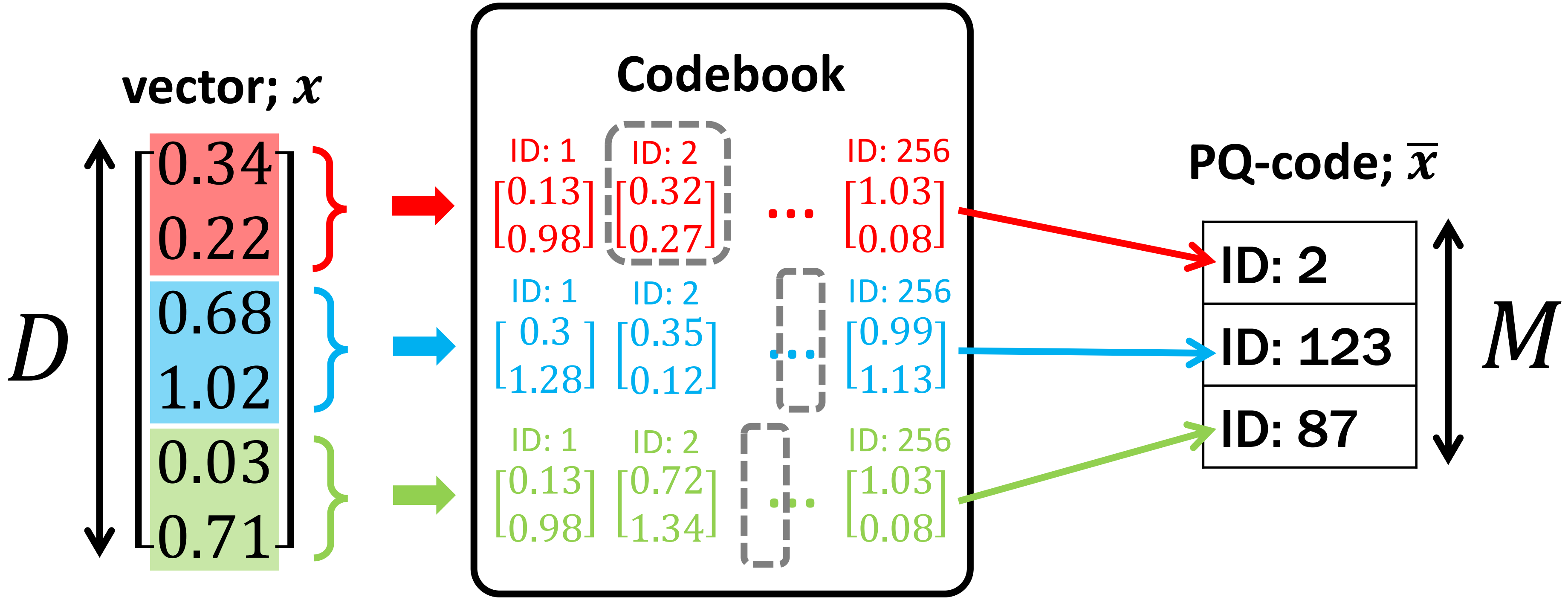


- Simple
- Memory efficient
- Distance can be estimated

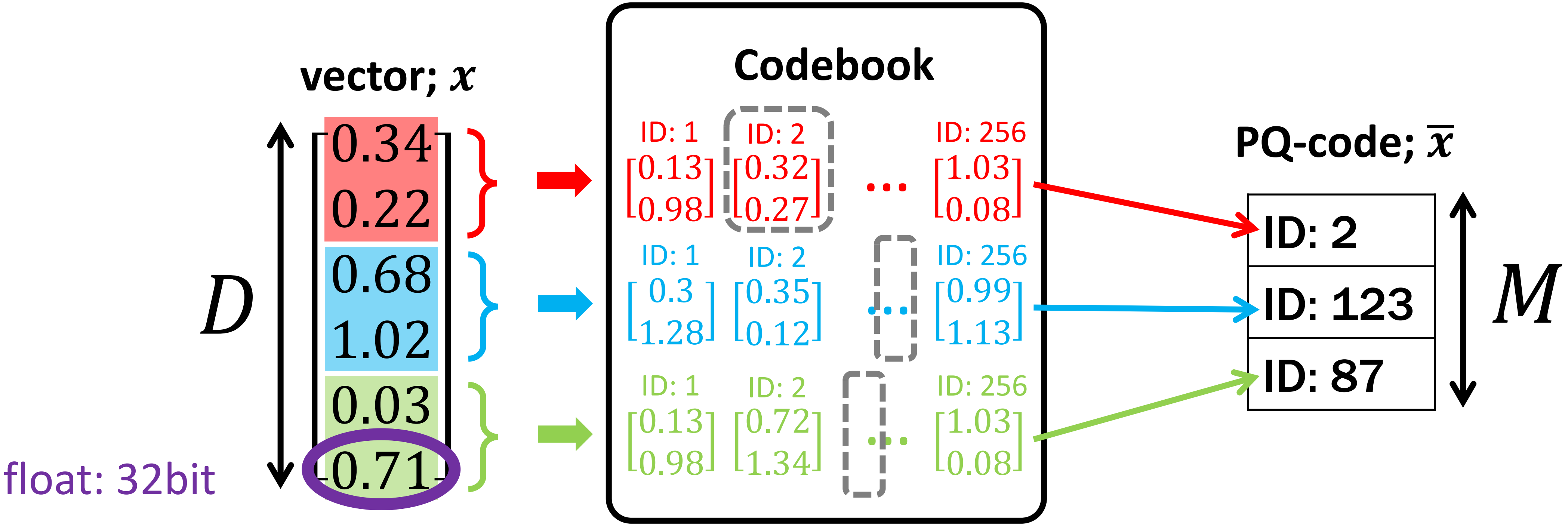
Bar notation for PQ-code:

$$\boldsymbol{x} \in \mathbb{R}^D \mapsto \bar{\boldsymbol{x}} \in \{1, \ldots, 256\}^M$$

Product Quantization: Memory efficient



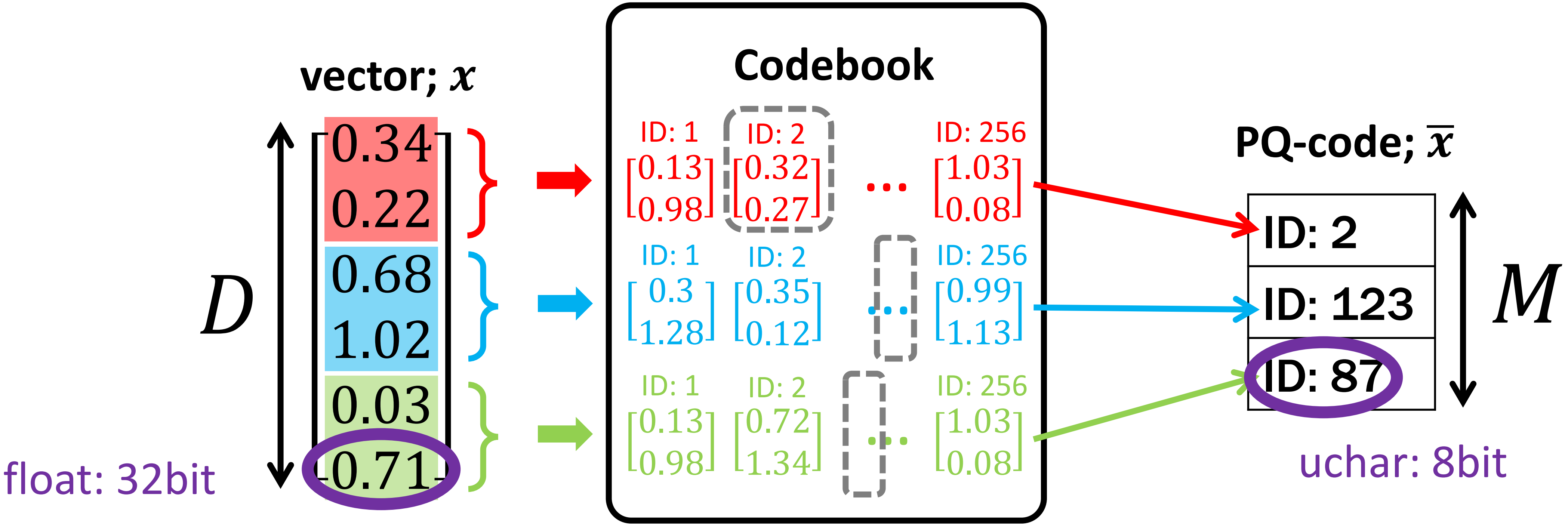
Product Quantization: Memory efficient



e.g., $D = 128$

$128 \times 32 = 4096$ [bit]

Product Quantization: Memory efficient

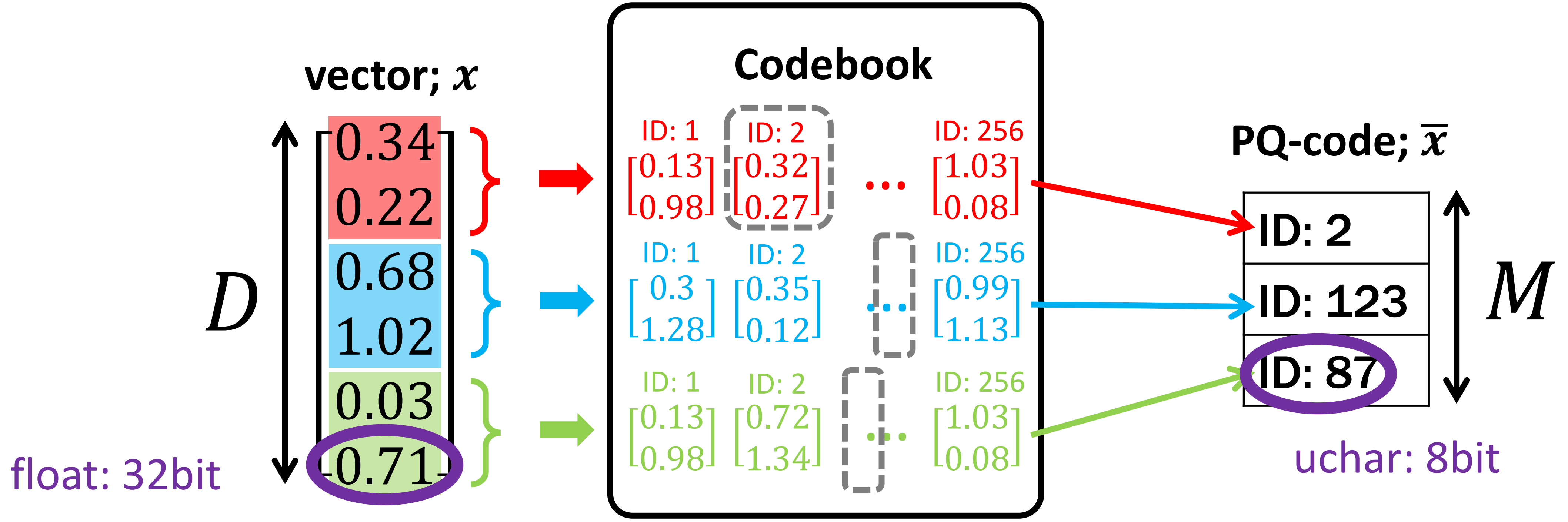


e.g., $D = 128$
$128 \times 32 = 4096$ [bit]

e.g., $M = 8$
$8 \times 8 = 64$ [bit]

Product Quantization: Memory efficient

Can store all 1B vectors with 8 GB of RAM!



e.g., $D = 128$
$128 \times 32 = 4096$ [bit]

e.g., $M = 8$
$8 \times 8 = 64$ [bit]

1/64

Product Quantization: Distance estimation

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

$$\begin{bmatrix} 0.34 \\ 0.22 \\ 0.68 \\ 1.02 \\ 0.03 \\ 0.71 \end{bmatrix}$$

Database vectors

$$\begin{array}{cccc} \boldsymbol{x}_1 & \boldsymbol{x}_2 & & \boldsymbol{x}_N \\ \begin{bmatrix} 0.54 \\ 2.35 \\ 0.82 \\ 0.42 \\ 0.14 \\ 0.32 \end{bmatrix} & \begin{bmatrix} 0.62 \\ 0.31 \\ 0.34 \\ 1.63 \\ 1.43 \\ 0.74 \end{bmatrix} & \cdots & \begin{bmatrix} 3.34 \\ 0.83 \\ 0.62 \\ 1.45 \\ 0.12 \\ 2.32 \end{bmatrix} \end{array}$$

Product Quantization: Distance estimation

Database vectors

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

$$\begin{bmatrix} 0.34 \\ 0.22 \\ 0.68 \\ 1.02 \\ 0.03 \\ 0.71 \end{bmatrix}$$

$$\begin{array}{cccc} \boldsymbol{x}_1 & \boldsymbol{x}_2 & & \boldsymbol{x}_N \\ \begin{bmatrix} 0.54 \\ 2.35 \\ 0.82 \\ 0.42 \\ 0.14 \\ 0.32 \end{bmatrix} & \begin{bmatrix} 0.62 \\ 0.31 \\ 0.34 \\ 1.63 \\ 1.43 \\ 0.74 \end{bmatrix} & \cdots & \begin{bmatrix} 3.34 \\ 0.83 \\ 0.62 \\ 1.45 \\ 0.12 \\ 2.32 \end{bmatrix} \end{array}$$

Product quantization

Product Quantization: Distance estimation

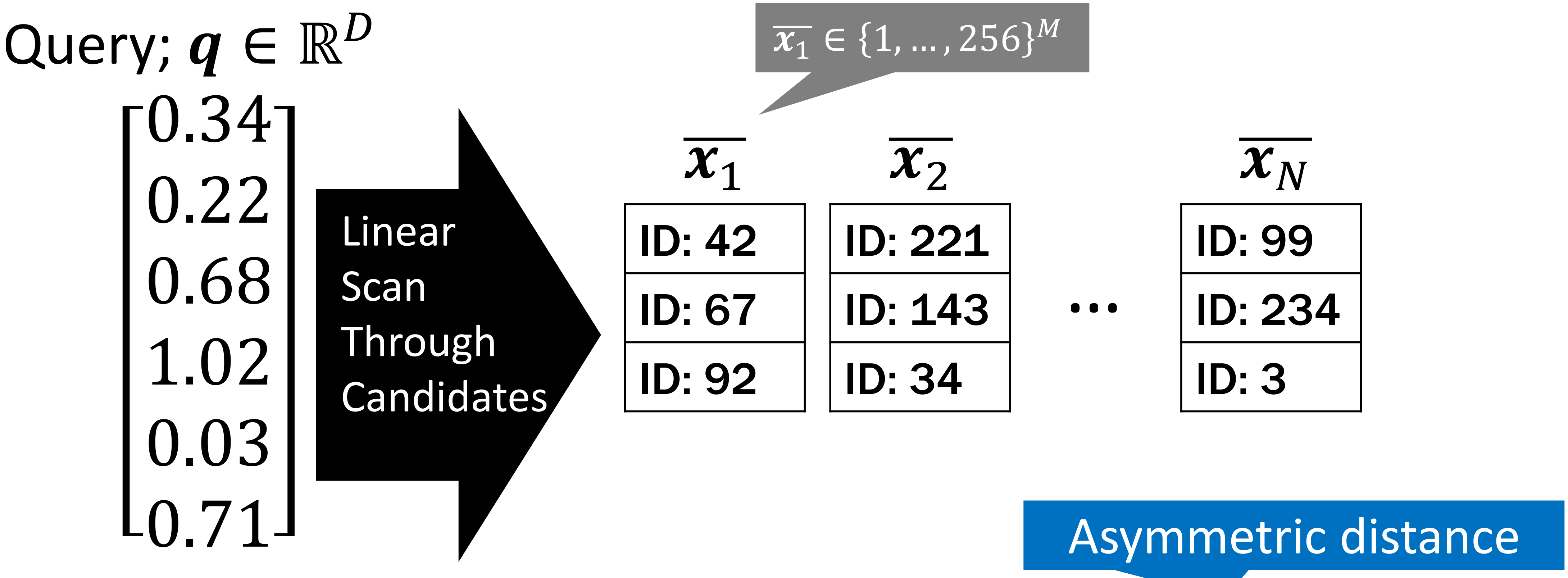
Query; $\boldsymbol{q} \in \mathbb{R}^D$

$$\begin{bmatrix} 0.34 \\ 0.22 \\ 0.68 \\ 1.02 \\ 0.03 \\ 0.71 \end{bmatrix}$$

$\overline{\boldsymbol{x}_1} \in \{1, \dots, 256\}^M$

$\overline{\boldsymbol{x}_1}$	$\overline{\boldsymbol{x}_2}$	…	$\overline{\boldsymbol{x}_N}$
ID: 42	ID: 221		ID: 99
ID: 67	ID: 143	…	ID: 234
ID: 92	ID: 34		ID: 3

Product Quantization: Distance estimation



- $d(\boldsymbol{q}, \boldsymbol{x})^2$ can be efficiently approximated by $d_A(\boldsymbol{q}, \overline{\boldsymbol{x}})^2$
- Lookup-trick: Looking up pre-computed distance-tables
- Candidate selection by d_A

```python
import numpy as np
from scipy.cluster.vq import vq, kmeans2
from scipy.spatial.distance import cdist

def train(vec, M):
    Ds = int(vec.shape[1] / M)  # Ds = D / M
    # codeword[m][k] = c_k^m
    codeword = np.empty((M, 256, Ds), np.float32)

    for m in range(M):
        vec_sub = vec[:, m * Ds : (m + 1) * Ds]
        codeword[m], label = kmeans2(vec_sub, 256)

    return codeword

def encode(codeword, vec):  # vec = {x_n}_{n=1}^N
    M, _K, Ds = codeword.shape
    # pqcode[n] = i(x_n),   pqcode[n][m] = i^m(x_n)
    pqcode = np.empty((vec.shape[0], M), np.uint8)

    for m in range(M):  # Eq. (2) and Eq. (3)
        vec_sub = vec[:, m * Ds: (m + 1) * Ds]
        pqcode[:, m], dist = vq(vec_sub, codeword[m])

    return pqcode
```

Not pseudo codes

```python
def search(codeword, pqcode, query):
    M, _K, Ds = codeword.shape
    # dist_table = D(m, k)
    dist_table = np.empty((M, 256), np.float32)

    for m in range(M):
        query_sub = query[m * Ds: (m + 1) * Ds]
        dist_table[m, :] = cdist([query_sub],
         ↪  codeword[m], 'sqeuclidean')[0]  # Eq. (5)

    # Eq. (6)
    dist = np.sum(dist_table[range(M), pqcode], axis=1)

    return dist

if __name__ == "__main__":
    # Read vec_train, vec ({x_n}_{n=1}^N), and query (y)
    M = 4
    codeword = train(vec_train, M)
    pqcode = encode(codeword, vec)
    dist = search(codeword, pqcode, query)
    print(dist)
```

- Only tens of lines in Python
- Pure Python library: nanopq https://github.com/matsui528/nanopq
- `pip install nanopq`

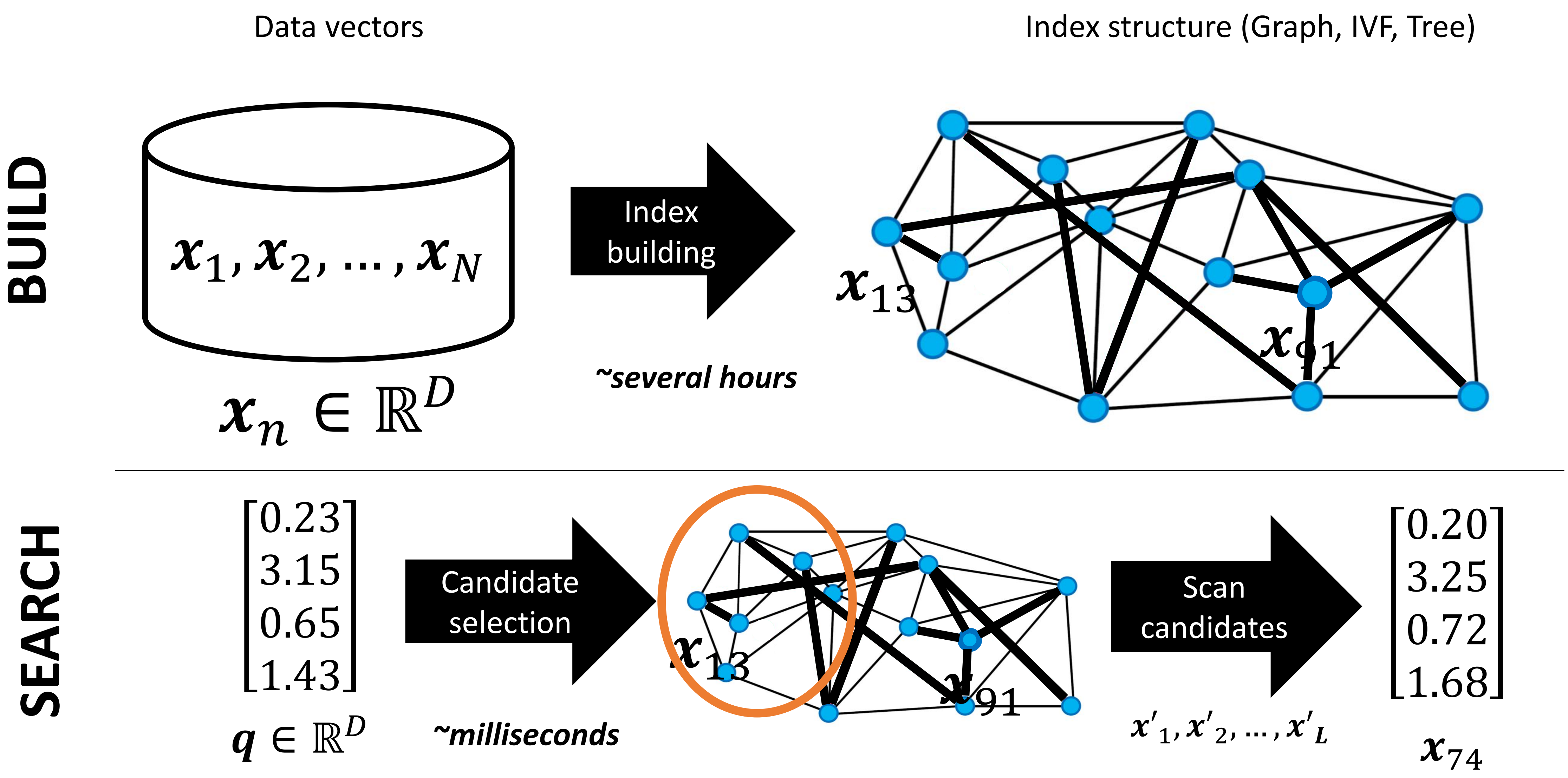
Rotate vectors to allow for better product quantization [Ge+14]

	BIGANN	MSSPACEV	TEXT2IMAGE	SSNPP
DiskANN	$R = 64, L = 128, \alpha = 1.2$	$R = 64, L = 128, \alpha = 1.2$	$R = 64, L = 128, \alpha = .9$	$R = 150, L = 400, \alpha = 1.2$
HNSW	$m = 32, efc = 128, \alpha = .82$	$m = 32, efc = 128, \alpha = .83$	$m = 32, efc = 128, \alpha = 1.1$	$m = 75, efc = 400, \alpha = .82$
HCNNG	$T = 30, Ls = 1000, s = 3$	$T = 50, Ls = 1000, s = 3$	$T = 30, Ls = 1000, s = 3$	$T = 50, Ls = 1000, s = 3$
pyNNDescent	$K = 40, Ls = 100,$ $T = 10, \alpha = 1.2$	$K = 60, Ls = 100,$ $T = 10, \alpha = 1.2$	$K = 60, Ls = 100,$ $T = 10, \alpha = .9$	$K = 60, Ls = 1000,$ $T = 10, \alpha = 1.4$
FAISS	OPQ64_128, IVF1048576_HNSW32, PQ128x4fsr	OPQ64_128, IVF1048576_HNSW32, PQ64x4fsr	OPQ64_128, IVF1048576_HNSW32, PQ128x4fsr	OPQ64_128, IVF1048576_HNSW32, PQ64

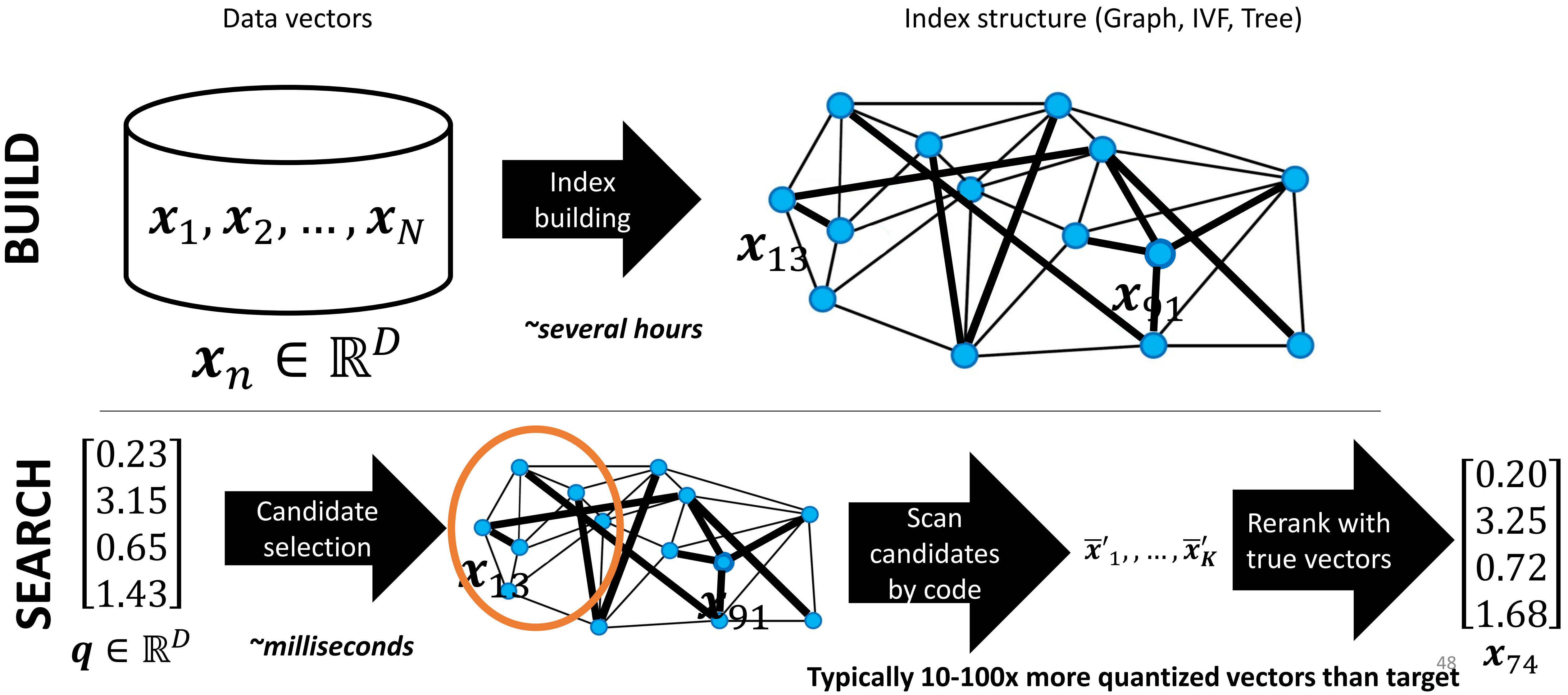
- Compress vector into 128 blocks,
- each with 2^4 = 16 codewords,
- use SIMD-based asymmetric distance computation [Andre+17]

Cluster with 1M centroids, using HNSW to index the centroids

The ANN search pipeline



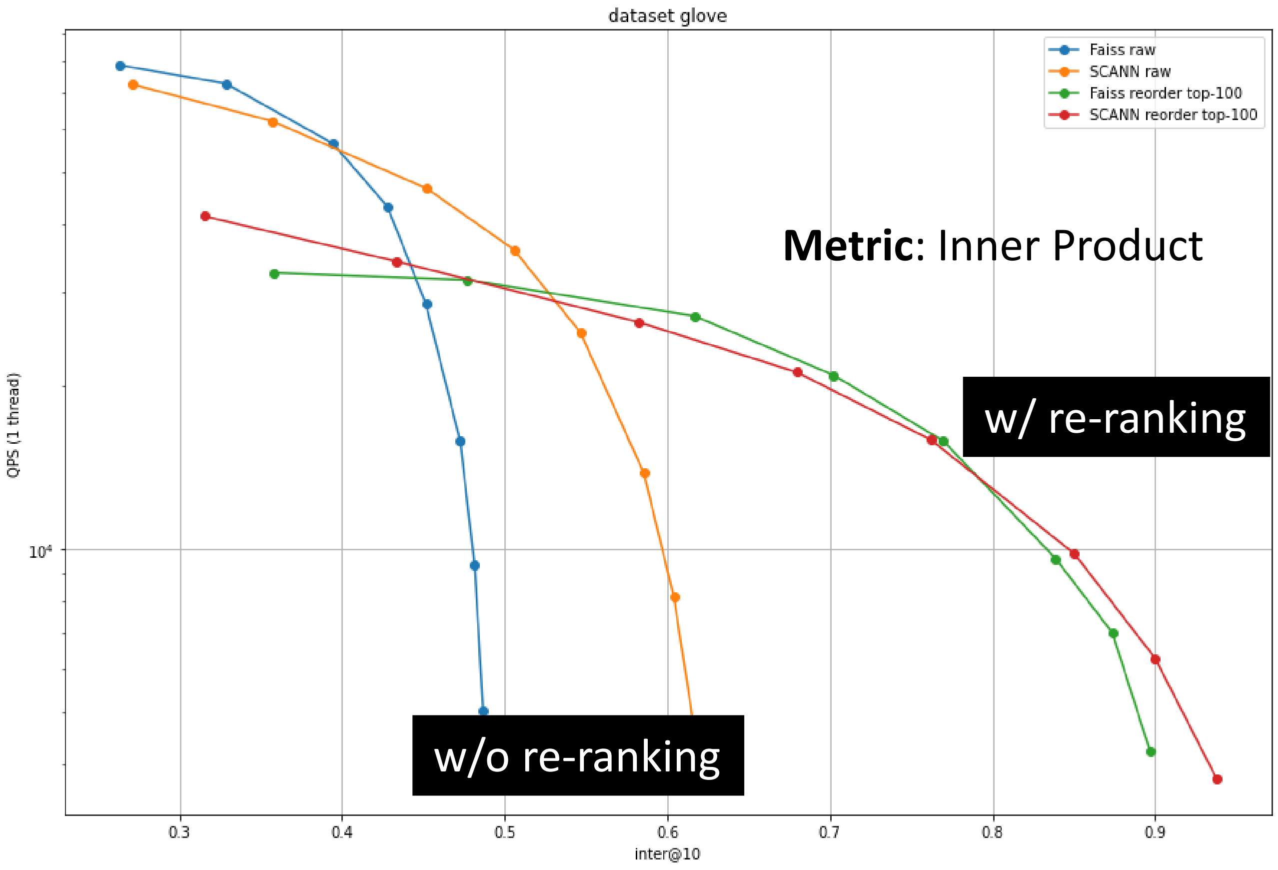
The ANN search pipeline (with quantization)



Index on Quantized Vectors

SCANN: Guo+, ICML 2020.

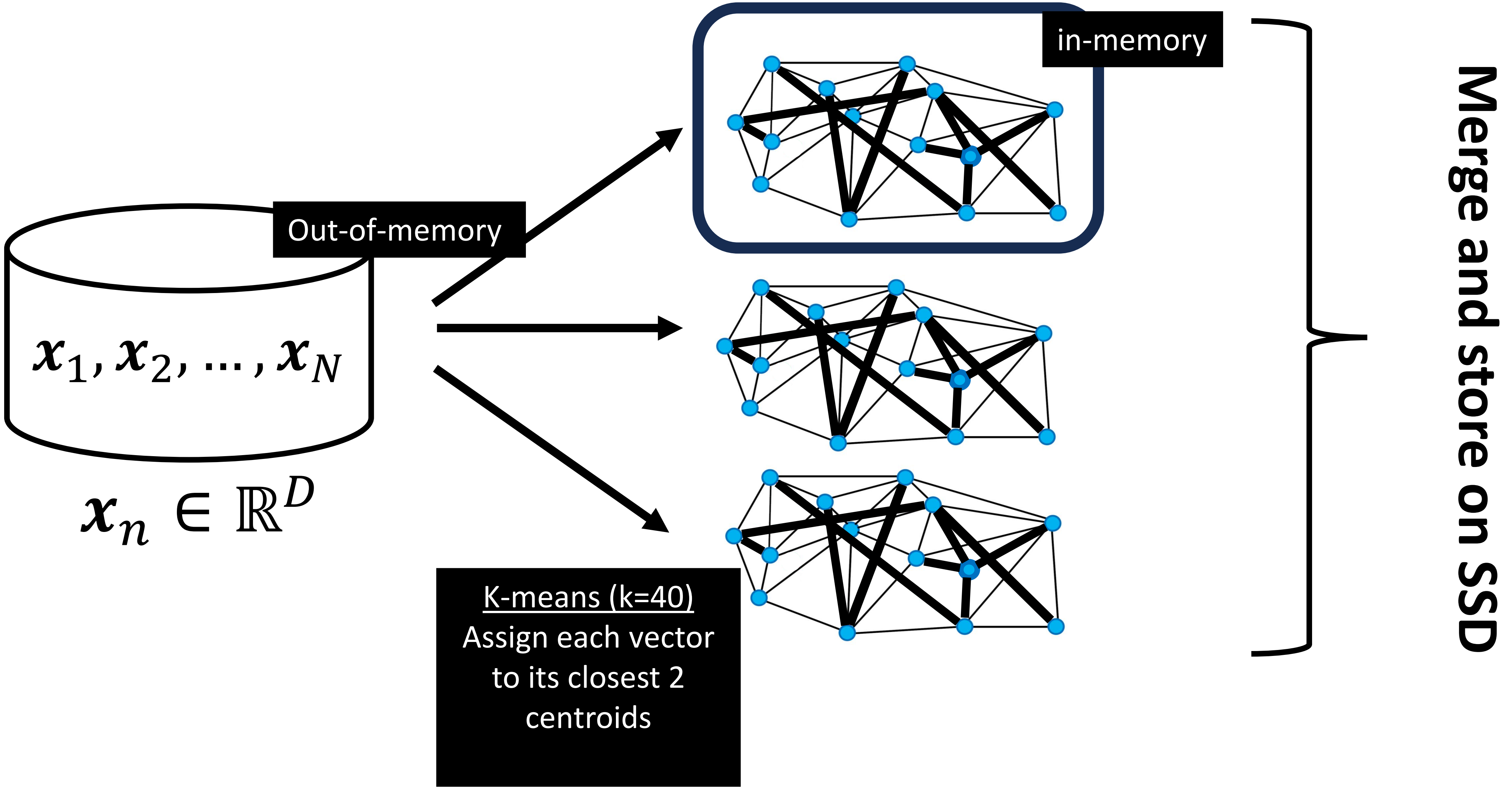
- Learn codes, represent each vector by its PQ code
- Code size: 32-64 byte
  - Can store the compressed vectors in memory
  - Lookup tables in cache/avx registers
- Index cost on top
  - **Graph**: 1G * degree_bound
    - Typically requires small degree_bounds (not well studied?)
  - **IVF**: 1M centroids + index on centroids on top of vectors
    - Usually works well



Recall quality very data dependent!

https://github.com/facebookresearch/faiss/wiki/Indexing-1M-vectors

Out-of-Memory index + High-Recall (DiskANN)



DiskANN out-of-memory

RAM **32+ GB**

$\bar{x}_i$

ID: 42
ID: 67
ID: 92

~32byte /vector

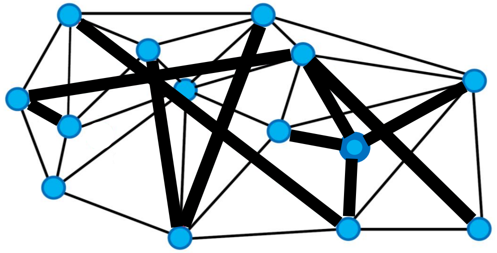
Expanding a node:

1. Read adjacent nodes from SSD (+ fetch original vector "for free")
2. Compute distances of query to neighbors (using PQ codes)

Still serves 1k+ queries per second

SSD **dataset size + 512 GB graph**

$R = 128$ degree



Memory layout for vector x_i

x_i	y_1		y_R

Original vector

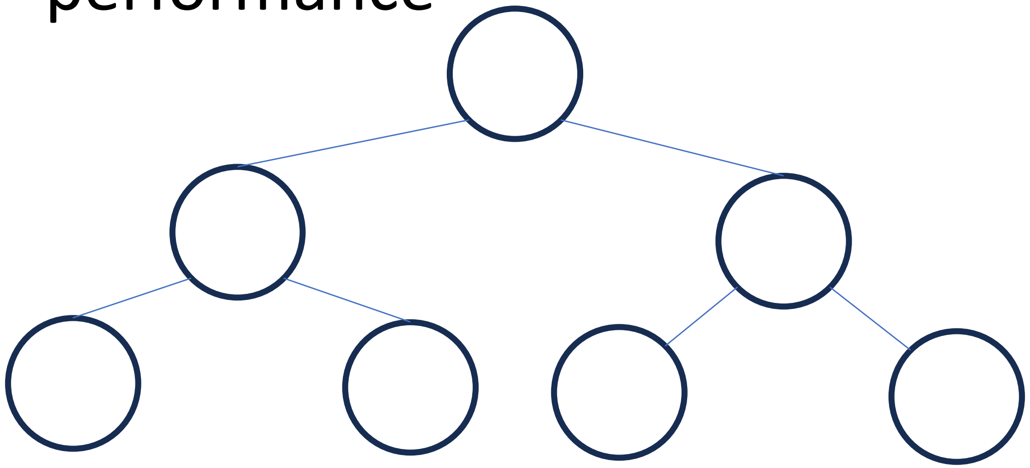
Neighbor indices, Padded if $< R$ neighbors

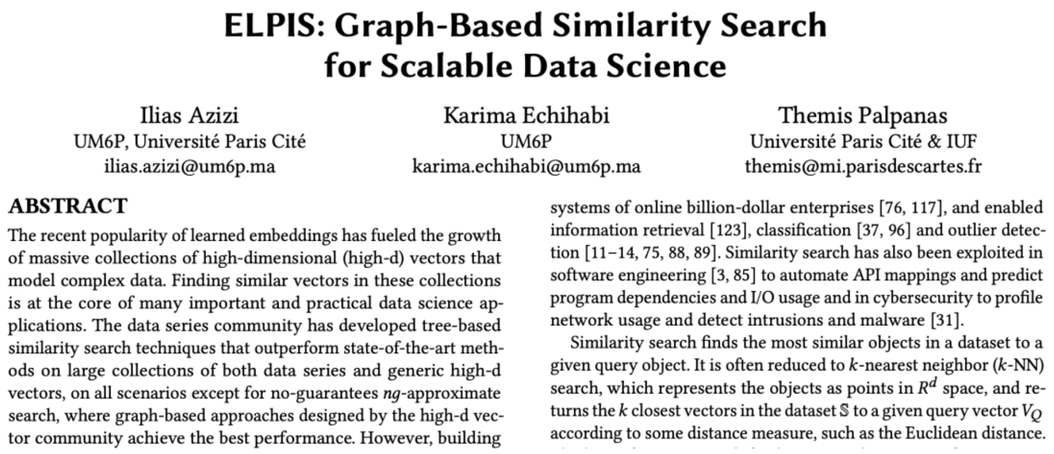
1 block read from SSD

(Very) recent developments

A new graph approach?

- Hierarchical tree, leaves are HNSW graphs
- Interesting quantization technique motivated by time series
- Better build times, good query performance





ELPIS: Graph-Based Similarity Search for Scalable Data Science

Ilias Azizi
UM6P, Université Paris Cité
ilias.azizi@um6p.ma

Karima Echihabi
UM6P
karima.echihabi@um6p.ma

Themis Palpanas
Université Paris Cité & IUF
themis@mi.parisdescartes.fr

ABSTRACT

The recent popularity of learned embeddings has fueled the growth of massive collections of high-dimensional (high-d) vectors that model complex data. Finding similar vectors in these collections is at the core of many important and practical data science applications. The data series community has developed tree-based similarity search techniques that outperform state-of-the-art methods on large collections of both data series and generic high-d vectors, on all scenarios except for no-guarantees *ng*-approximate search, where graph-based approaches designed by the high-d vector community achieve the best performance. However, building

systems of online billion-dollar enterprises [76, 117], and enabled information retrieval [123], classification [37, 96] and outlier detection [11–14, 75, 88, 89]. Similarity search has also been exploited in software engineering [3, 85] to automate API mappings and predict program dependencies and I/O usage and in cybersecurity to profile network usage and detect intrusions and malware [31].

Similarity search finds the most similar objects in a dataset to a given query object. It is often reduced to k-nearest neighbor (k-NN) search, which represents the objects as points in R^d space, and returns the k closest vectors in the dataset $\mathbb{S}$ to a given query vector V_Q according to some distance measure, such as the Euclidean distance.

To appear at VLDB 2023, https://www.vldb.org/pvldb/vol16/p1548-azizi.pdf

Automated Parameter tuning

- Finding build/search parameters by constrained optimization
- Build on top of ScaNN

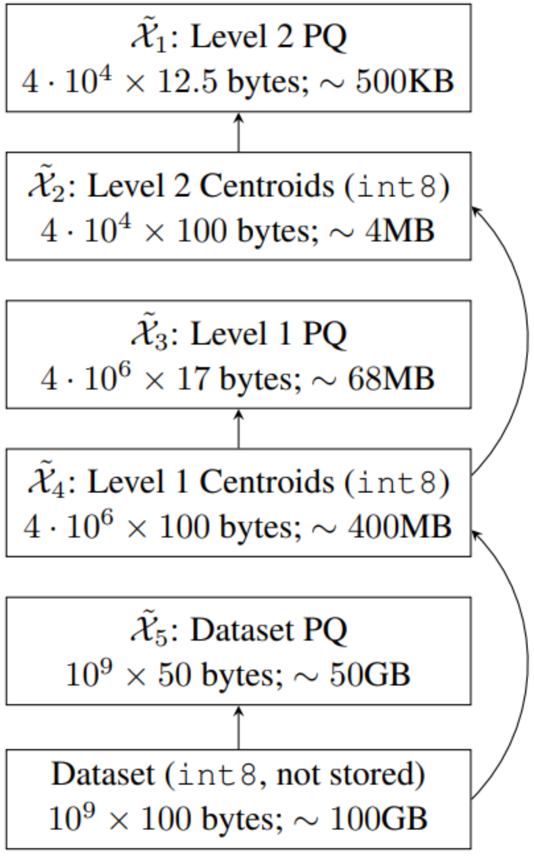
AUTOMATING NEAREST NEIGHBOR SEARCH CONFIGURATION WITH CONSTRAINED OPTIMIZATION

Philip Sun, Ruiqi Guo & Sanjiv Kumar
Google Research
New York, NY
{sunphil,guorq,sanjivk}@google.com

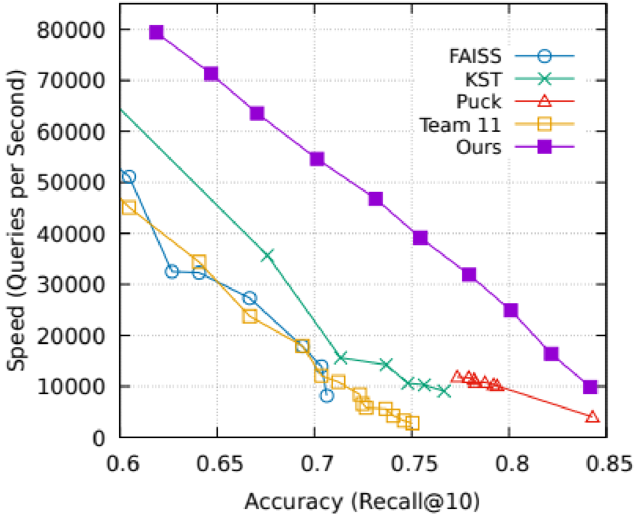
ICLR23

ABSTRACT

The approximate nearest neighbor (ANN) search problem is fundamental to efficiently serving many real-world machine learning applications. A number of techniques have been developed for ANN search that are efficient, accurate, and scalable. However, such techniques typically have a number of parameters that affect the speed-recall tradeoff, and exhibit poor performance when such parameters aren't properly set. Tuning these parameters has traditionally been a manual process, demanding in-depth knowledge of the underlying search algorithm. This is becoming an increasingly unrealistic demand as ANN search grows in popu-



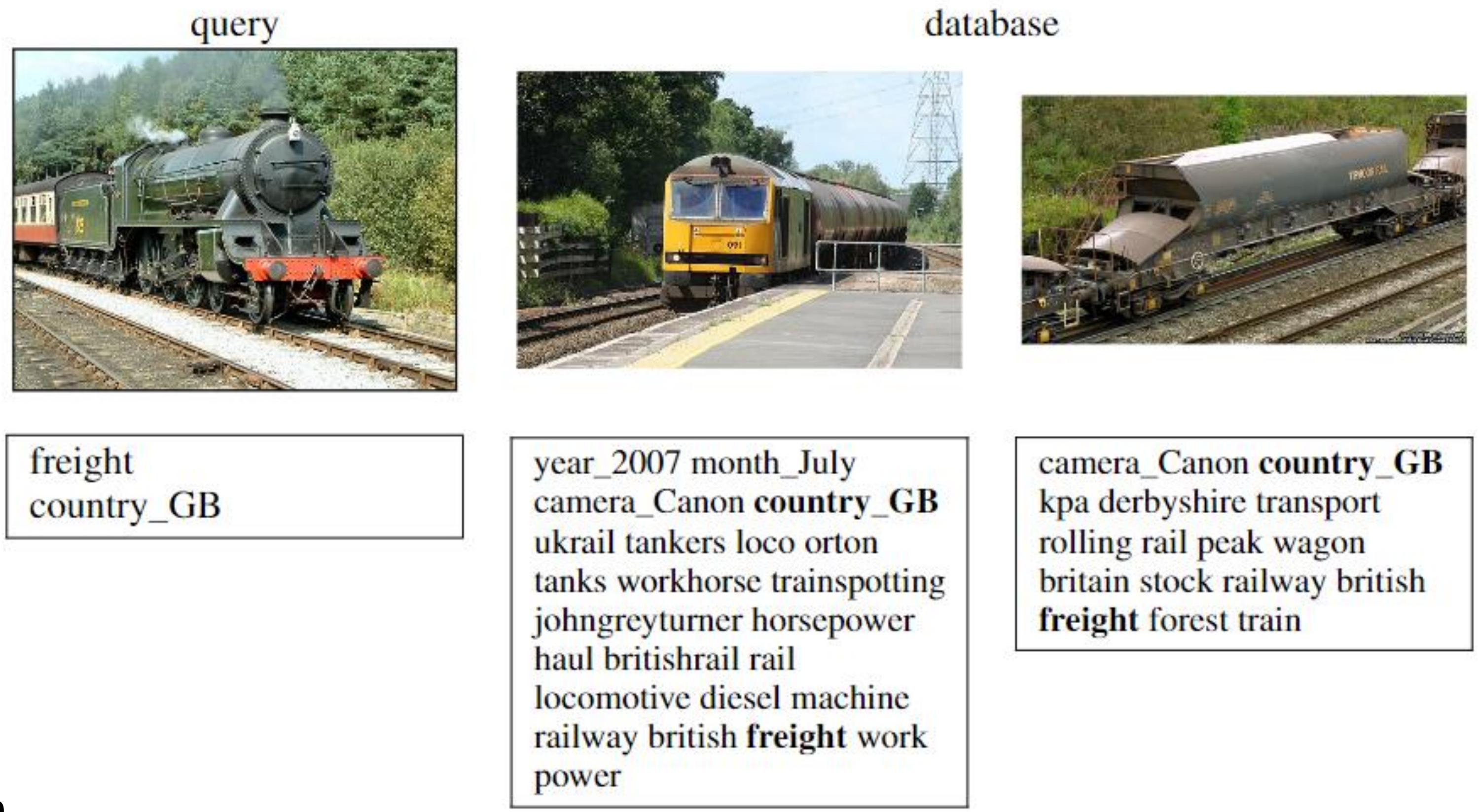
(b) Microsoft Turing-ANNS



(b) Microsoft Turing-ANNS

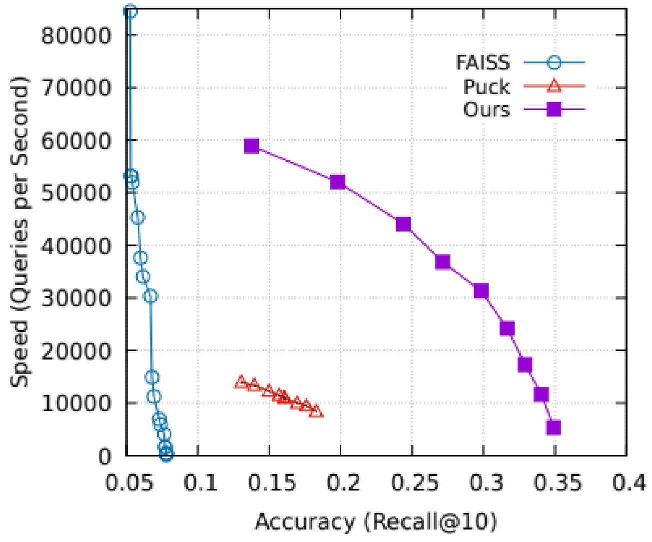
Filtered search

- **Setting**
  - Vectors have associated metadata
  - Example, YFCC: tags, gps, date
- **Query**
  - Find the most similar images to this images that were taken with a Sony Camera in 2017 in Vancouver



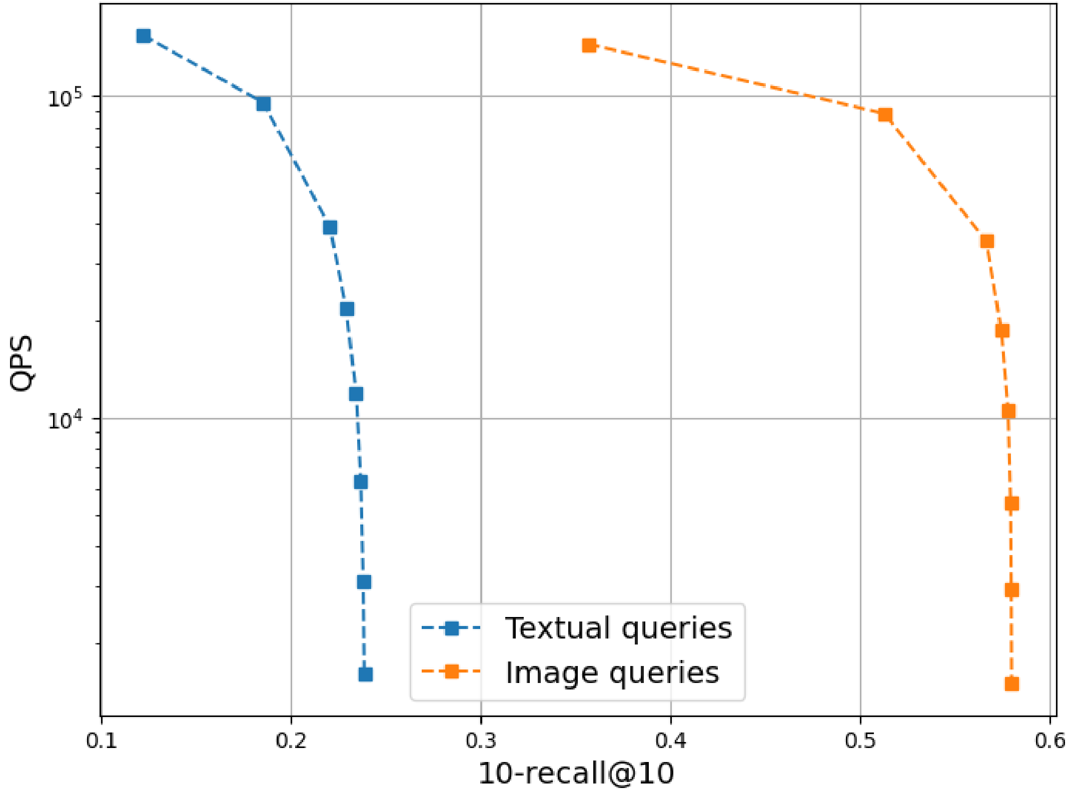
Out-of-distribution queries

- **Setting**
  - Vectors are image embeddings
  - Queries are text embeddings



(c) Yandex Text-to-Image

https://arxiv.org/pdf/2301.01702.pdf



OPQ64_128,IVF16384,PQ64

Yandex, Text-2-Image dataset

Streaming settings

- **Setting**
  - Many applications (search engine, recommender system) need to handle updates
- Daily rebuilds often too expensive
- **Question**: Clever update strategies?

	Web Search & Reco	Email Search	Enterprise search
Index Size	~1 trillion pages	100s of trillions of sentences	Trillions of paragraphs across documents
Update Rate (latency <1s)	Billions of updates/day	Ingest new email, Purge deletes	Handle >1% change/day
Search latency/QPS	<10ms 10-100K+ Queries/sec	100s of ms	10-100ms

https://harsha-simhadri.org/pubs/ANNS-talk-Sep22.pptx

NeurIPS 2023 Challenge: Practical Vector Search

- **4 Tasks (10M vectors)**
  - Filtered ANN
  - Streaming ANN
  - Out-of-distribution ANN
  - ANN on sparse data
- Strong baselines based on IVF (faiss) and graphs (DiskANN)
- Cloud credits available for testing (screening process)

Practical Vector Search Challenge 2023

Harsha Vardhan Simhadri*
Microsoft Research India
harshasi@microsoft.com

Martin Aumüller
IT University of Copenhagen
maau@itu.dk

Dmitry Baranchuk
Yandex
dbaranchuk@yandex-team.ru

Matthijs Douze
Meta AI Research
matthijs@meta.com

Edo Liberty
Pinecone.io
edo@pinecone.io

Amir Ingber
Pinecone.io
ingber@pinecone.io

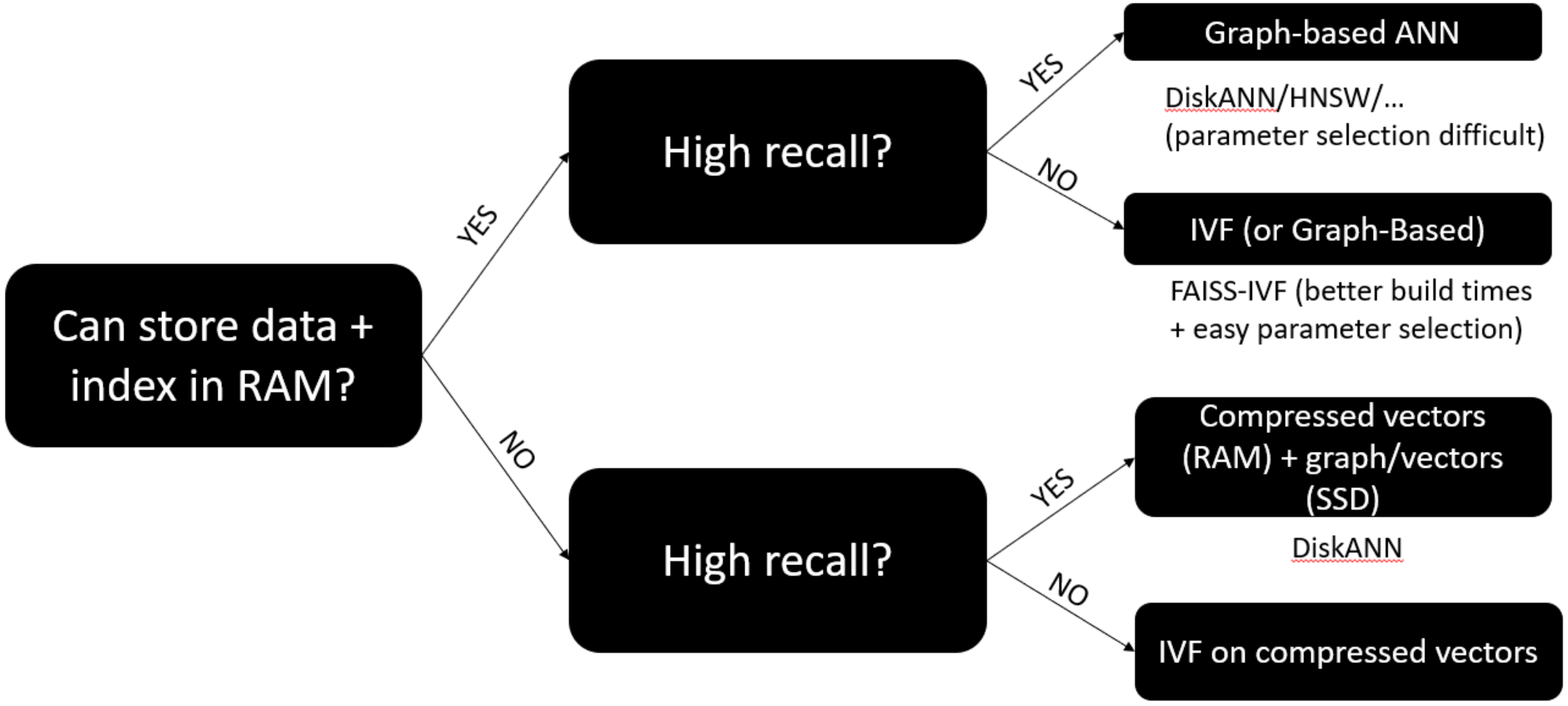
Frank Liu
Zilliz
frank.liu@zilliz.com

George Williams
Independent Researcher
gwilliams@ieee.org

Official announcement soon!

https://big-ann-benchmarks.com

Timeframe: July-November 2023



Thanks!

https://matsui528.github.io/cvpr2023_tutorial_neural_search/

https://big-ann-benchmarks.com

JUNE 18-22, 2023
CVPR
VANCOUVER, CANADA

Neural Search in Action

Representing, transiting & searching **multimodal data**

Han Xiao, Founder of Jina AI
@hxiao @JinaAI_

Berlin · Beijing · Shenzhen

About me & Jina AI

Han Xiao, Founder & CEO of Jina AI. Based in Berlin, Germany.

- ML PhD in 2014 TU Munich; Zalando Research; Tencent AI Lab; Creator of Fashion-MNIST.

Jina AI

- Founded in 2020, focus on multimodal AI search & create
- Opensource contributor: Jina, **DocArray (Linux Foundation)**, CLIP-as-service, …
- 60 people, HQ in Berlin. Offices in Beijing, Shenzhen.

Jina AI Tech Spectrum

Prompt tuning

the process of crafting and refining the input prompts in order to guide its output towards specific, desired responses.

Model serving

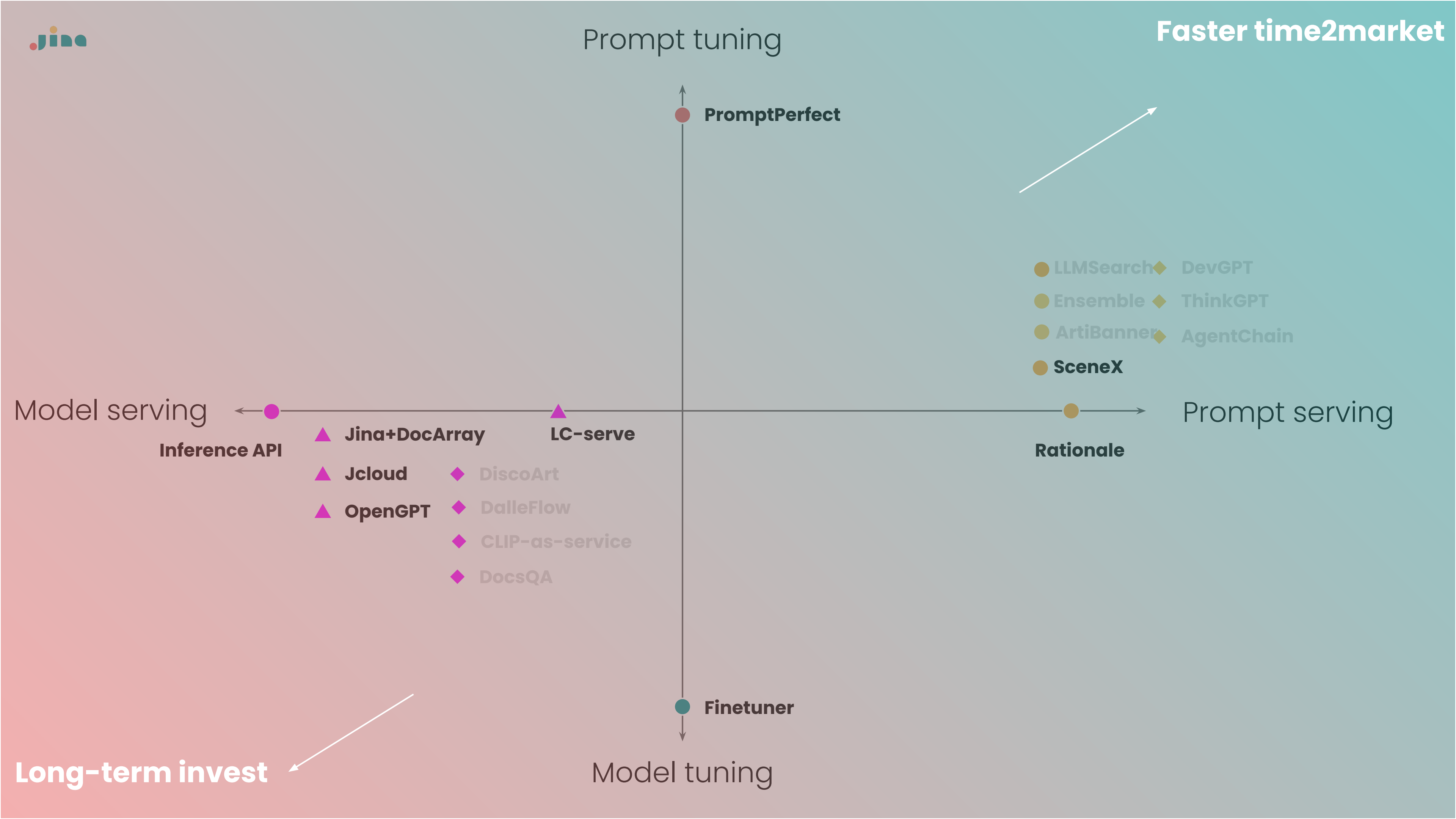
the deployment of fine-tuned models in a production environment, usually requiring substantial resources such as GPU hosting. MLOps, emphasizing the serving of mid-size to large models in a scalable, efficient, and reliable manner.

Prompt serving

wrapping and serving prompts through an API, without hosting heavy models. The API calls a public large language model service and handles the orchestration of inputs and outputs in a chain of operations.

Model tuning

Also known as fine-tuning, involves adjusting the parameters of a pre-trained model on a new, often task-specific dataset to improve its performance and adapt it to a specific application.

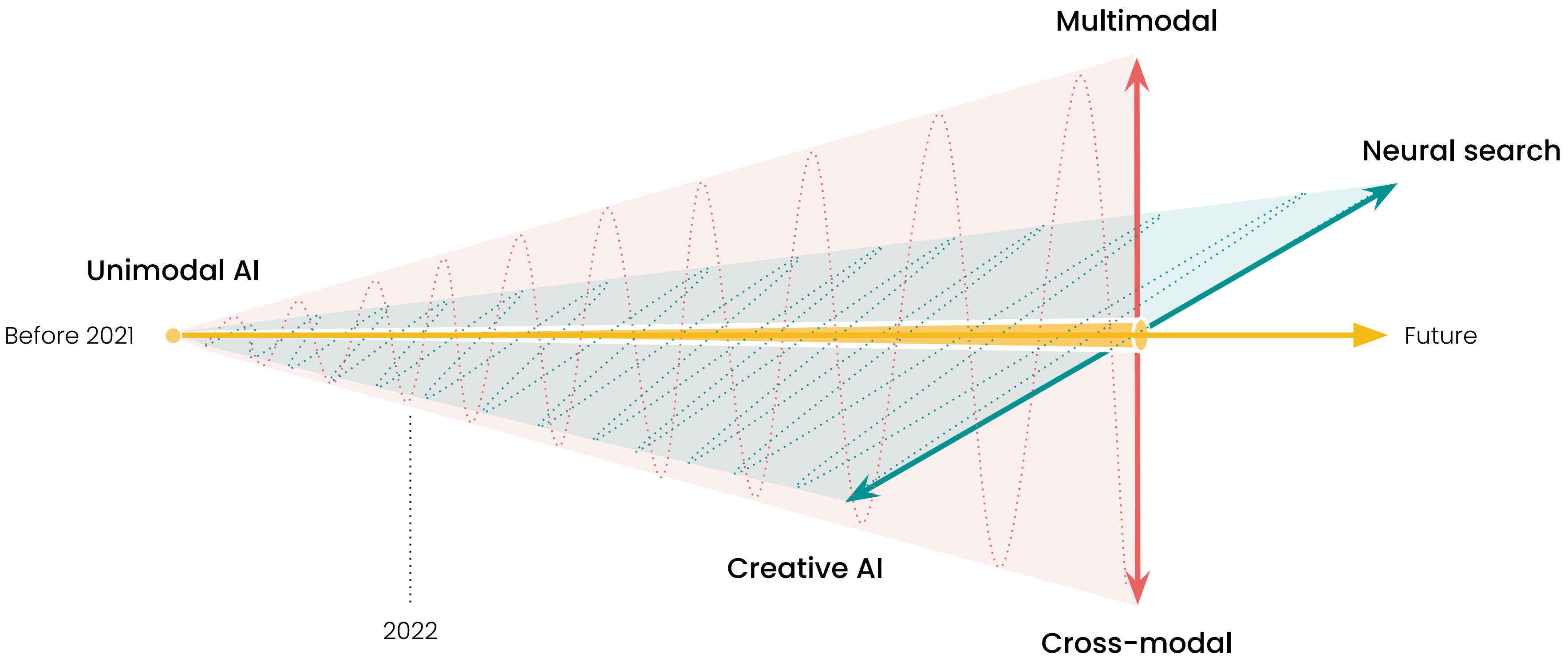

Faster time2market
Prompt tuning
PromptPerfect
LLMSearch
DevGPT
Ensemble
ThinkGPT
ArtiBanner
AgentChain
SceneX
Model serving
Prompt serving
Inference API
Jina+DocArray
LC-serve
Rationale
Jcloud
DiscoArt
OpenGPT
DalleFlow
CLIP-as-service
DocsQA
Finetuner
Model tuning
Long-term invest

Agenda

- **Preliminary: multimodal AI**
- Opensource package: DocArray
  - Motivation
  - Representing data
  - Transiting data
  - Storing data
  - Retrieving data
- Multimodal at scale in production

This tutorial may require technical knowledge. Familiarity with Python 3.7+ concepts like data classes could be helpful.

Preliminary:
from unimodal to multimodal


Multimodal
Neural search
Unimodal AI
Before 2021
Future
Creative AI
2022
Cross-modal

From unimodal to multimodal

"**modality**" roughly means "**data type**".

- Unimodal AI refers to applying AI to one specific type of data.
- Most early machine learning works fall into this category.
- Even today, when you open any machine learning literature, unimodal AI is still the majority of the content.

Unimodal – NLP

LDA was the 2010's transformer

"Arts"	"Budgets"	"Children"	"Education"
NEW	MILLION	CHILDREN	SCHOOL
FILM	TAX	WOMEN	STUDENTS
SHOW	PROGRAM	PEOPLE	SCHOOLS
MUSIC	BUDGET	CHILD	EDUCATION
MOVIE	BILLION	YEARS	TEACHERS
PLAY	FEDERAL	FAMILIES	HIGH
MUSICAL	YEAR	WORK	PUBLIC
BEST	SPENDING	PARENTS	TEACHER
ACTOR	NEW	SAYS	BENNETT
FIRST	STATE	FAMILY	MANIGAT
YORK	PLAN	WELFARE	NAMPHY
OPERA	MONEY	MEN	STATE
THEATER	PROGRAMS	PERCENT	PRESIDENT
ACTRESS	GOVERNMENT	CARE	ELEMENTARY
LOVE	CONGRESS	LIFE	HAITI

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. "Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services," Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center's share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.

JMLR: Workshop and Conference Proceedings 13: 63-78
2nd Asian Conference on Machine Learning (ACML2010), Tokyo, Japan, Nov. 8–10, 2010.

Efficient Collapsed Gibbs Sampling For Latent Dirichlet Allocation

Han Xiao XIAOH@IN.TUM.DE
Thomas Stibor STIBOR@IN.TUM.DE
Department of Informatics
Technical University of Munich, GERMANY

Editor: Masashi Sugiyama and Qiang Yang

Abstract

Collapsed Gibbs sampling is a frequently applied method to approximate intractable integrals in probabilistic generative models such as latent Dirichlet allocation. This sampling method has however the crucial drawback of high computational complexity, which makes it limited applicable on large data sets. We propose a novel *dynamic sampling* strategy to significantly improve the efficiency of collapsed Gibbs sampling. The strategy is explored in terms of efficiency, convergence and perplexity. Besides, we present a straight-forward parallelization to further improve the efficiency. Finally, we underpin our proposed improvements with a comparative study on different scale data sets.

Keywords: Gibbs sampling, Optimization, Latent Dirichlet Allocation

1. Introduction

Latent Dirichlet allocation (LDA) is a generative probabilistic model that was first proposed by Blei et al. (2003) to discover topics in text documents. LDA is based on the

Unimodal tasks in NLP

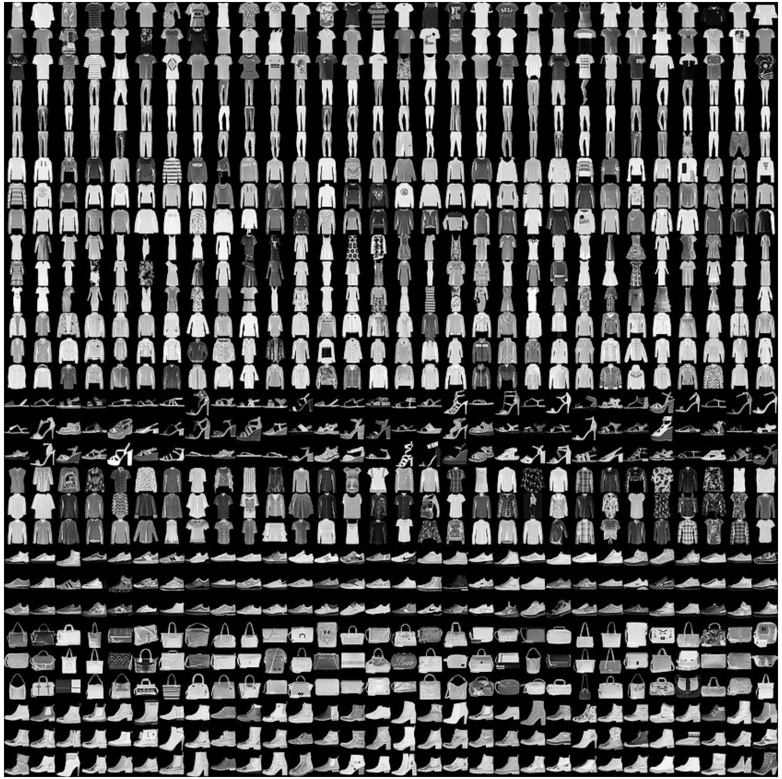
Adhoc methods for NLP problems

Sentiment analysis	Text classification	Topic modeling	Text summarization	Natural language generation
Named entity recognition	Word sense disambiguation	Parts-of-speech tagging	Grammatical parsing	Machine translation
Question answering	Spam filtering	Language modeling	Dialog systems	Information extraction
Semantic role labeling	Part-of-speech induction	Co-reference resolution	Pronoun resolution	Sentence segmentation

Textual Modality

Unimodal - CV

Fashion-MNIST, 2017



)7747v2 [cs.LG] 15 Sep 2017

Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms

Han Xiao
Zalando Research
Mühlenstraße 25, 10243 Berlin
han.xiao@zalando.de

Kashif Rasul
Zalando Research
Mühlenstraße 25, 10243 Berlin
kashif.rasul@zalando.de

Roland Vollgraf
Zalando Research
Mühlenstraße 25, 10243 Berlin
roland.vollgraf@zalando.de

Abstract

We present Fashion-MNIST, a new dataset comprising of 28×28 grayscale images of $70,000$ fashion products from 10 categories, with $7,000$ images per category. The training set has $60,000$ images and the test set has $10,000$ images. Fashion-MNIST is intended to serve as a direct drop-in replacement for the original MNIST dataset for benchmarking machine learning algorithms, as it shares the same image size, data format and the structure of training and testing splits. The dataset is freely available at https://github.com/zalandoresearch/fashion-mnist.

Unimodal tasks in CV

Object classification and detection	Image segmentation	Object tracking	Action recognition	Scene understanding
3D reconstruction	Pose estimation	Depth estimation	Stereo vision	Texture recognition and classification
Material recognition	Object recognition in video	Facial recognition and identification	Human activity recognition	Image super-resolution
Neural style transfer	Image inpainting	Video frame interpolation	Multiple object tracking in 3D	SLAM

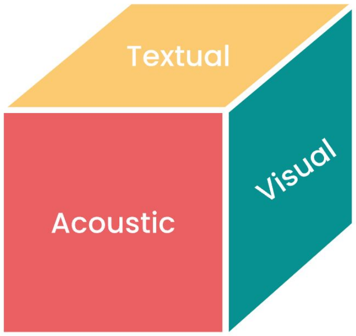
Visual Modality

Unimodal tasks in speech & audio

Automatic Speech Recognition	Text-to-Speech	Speaker Recognition	Speaker Diarization	Speech Enhancement
Music Recommendation	Music Genre Recognition	Music Artist Recognition	Music Structure Segmentation	Music Tempo Estimation
Audio Source Separation	Sound Event Detection	Sound Event Classification	Sound Event Localization	Audio Scene Recognition
Audio Captioning	Emotion Recognition	Speech Translation	Voice Activity Detection	Silence Detection

Acoustic Modality

Unimodal know-how are hardly transferable



- Tasks are specific to just one modality (e.g. textual, visual, acoustic, etc).
- Knowledge is learned from and applied to only one modality (i.e. a visual algorithm can only learn from and be applied to images).

Sentiment analysis	Text classification	Topic modeling	Text summarization	Natural language generation
Named entity recognition	Word sense disambiguation	Parts-of-speech tagging	Grammatical parsing	Machine translation
Question answering	Spam filtering	Language modeling	Dialog systems	Information extraction
Semantic role labeling	Part-of-speech induction	Co-reference resolution	Pronoun resolution	Sentence segmentation

Textual Modality

Object classification and detection	Image segmentation	Object tracking	Action recognition	Scene understanding
3D reconstruction	Pose estimation	Depth estimation	Stereo vision	Texture recognition and classification
Material recognition	Object recognition in video	Facial recognition and identification	Human activity recognition	Image super-resolution
Neural style transfer	Image inpainting	Video frame interpolation	Multiple object tracking in 3D	SLAM

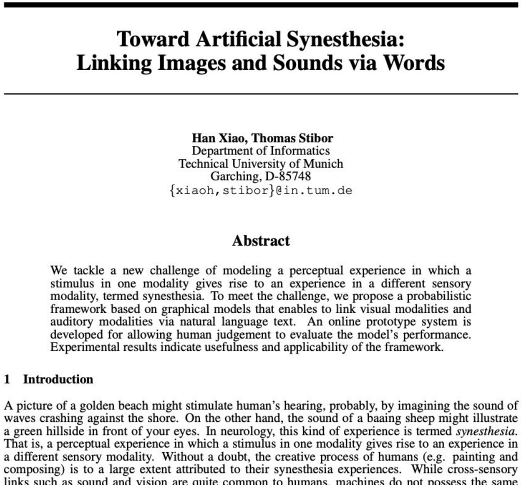
Visual Modality

Automatic Speech Recognition	Text-to-Speech	Speaker Recognition	Speaker Diarization	Speech Enhancement
Music Recommendation	Music Genre Recognition	Music Artist Recognition	Music Structure Segmentation	Music Tempo Estimation
Audio Source Separation	Sound Event Detection	Sound Event Classification	Sound Event Localization	Audio Scene Recognition
Audio Captioning	Emotion Recognition	Speech Translation	Voice Activity Detection	Silence Detection

Acoustic Modality

A detour: cross-modal model

NIPS 2010, Cross-LDA



Toward Artificial Synesthesia: Linking Images and Sounds via Words

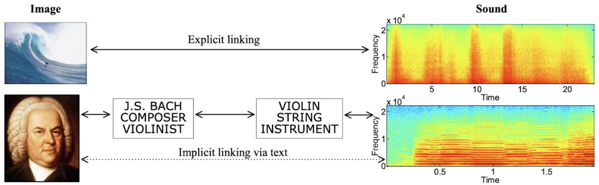
Han Xiao, Thomas Stibor
Department of Informatics
Technical University of Munich
Garching, D-85748
{xiaoh,stibor}@in.tum.de

Abstract

We tackle a new challenge of modeling a perceptual experience in which a stimulus in one modality gives rise to an experience in a different sensory modality, termed synesthesia. To meet the challenge, we propose a probabilistic framework based on graphical models that enables to link visual modalities and auditory modalities via natural language text. An online prototype system is developed for allowing human judgement to evaluate the model's performance. Experimental results indicate usefulness and applicability of the framework.

1 Introduction

A picture of a golden beach might stimulate human's hearing, probably, by imagining the sound of waves crashing against the shore. On the other hand, the sound of a baaing sheep might illustrate a green hillside in front of your eyes. In neurology, this kind of experience is termed *synesthesia*. That is, a perceptual experience in which a stimulus in one modality gives rise to an experience in a different sensory modality. Without a doubt, the creative process of humans (e.g. painting and composing) is to a large extent attributed to their synesthesia experiences. While cross-sensory links such as sound and vision are quite common to humans, machines do not possess the same



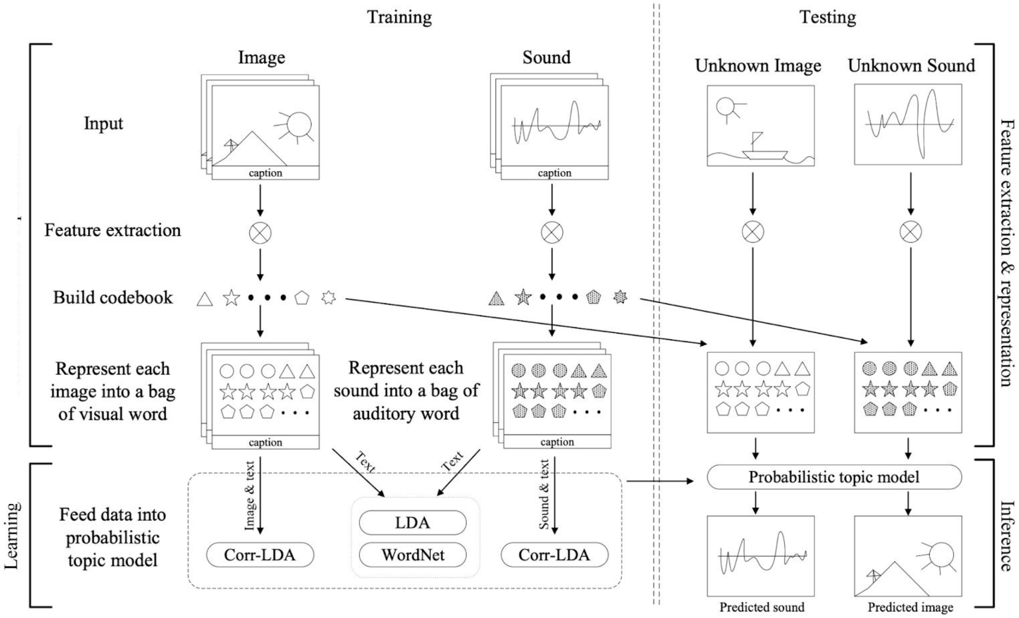
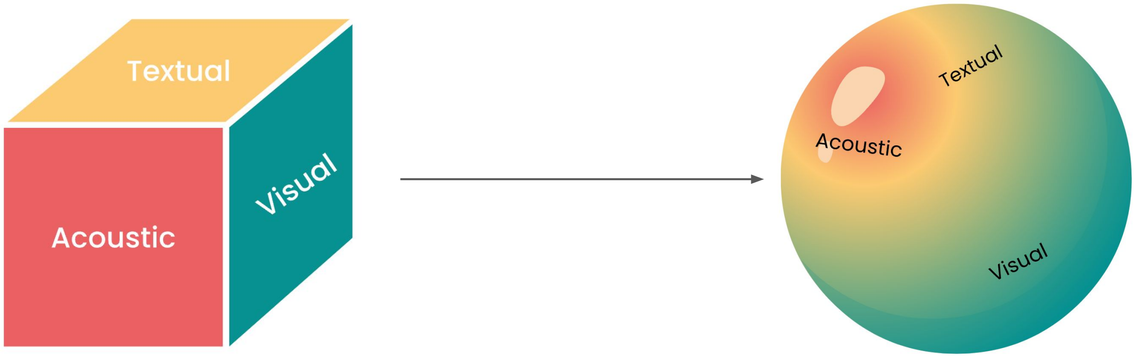


Figure 2: Probabilistic framework for performing the image composition and sound illustration task. The framework is an extension based on the work flow proposed in [8]. Images and sounds are represented in bags-of-words, so that the difference between the two modalities can be omitted. Once we have the algorithm for inferring sounds from an image, we can apply it to infer images from a sound by mirroring the algorithm.

Erase the boundary between modalities



- Tasks are shared and transferred between multiple modalities (so one algorithm can work with images and text and audio).
- Knowledge is learned from and applied to multiple modalities (so an algorithm can learn from textual data and apply that to visual data).

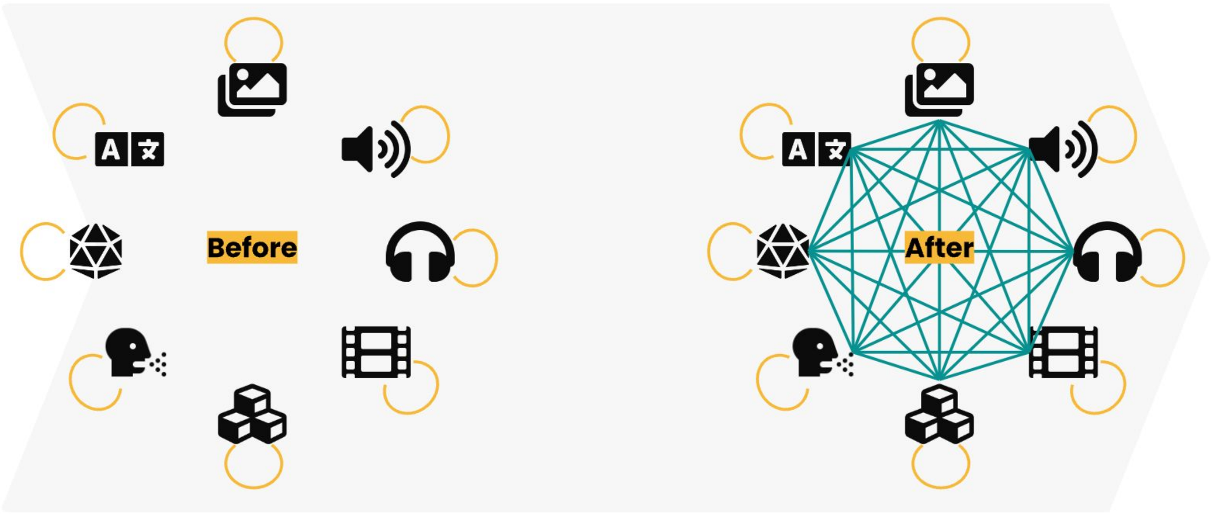
Paradigm shift from unimodal to multimodal

The rise of multimodal AI can be attributed to advances in two machine learning techniques: **Representation learning** and **transfer learning**.

- Representation learning lets models create common representations for all modalities.
- Transfer learning lets models first learn fundamental knowledge, and then fine-tune on specific domains.

CLIP, DALLE, BLIP, Bark, GPT4

We will see more and more AI applications move beyond one data modality and leverage relationships between different modalities



The paradigm shift from single-modal AI to multimodal AI

“An artificial intelligence system trained on words and sentences alone will never approximate human understanding.”

Y. Lecun in 2022 in AI And The Limits Of Language

Multimodal AI is the future,
but the ML ecosystem is not yet
suited for it.

Agenda

- Preliminary: multimodal AI
- **Opensource package: DocArray**
  - Motivation
  - Representing data
  - Transiting data
  - Storing data
  - Retrieving data
- Multimodal at scale in production

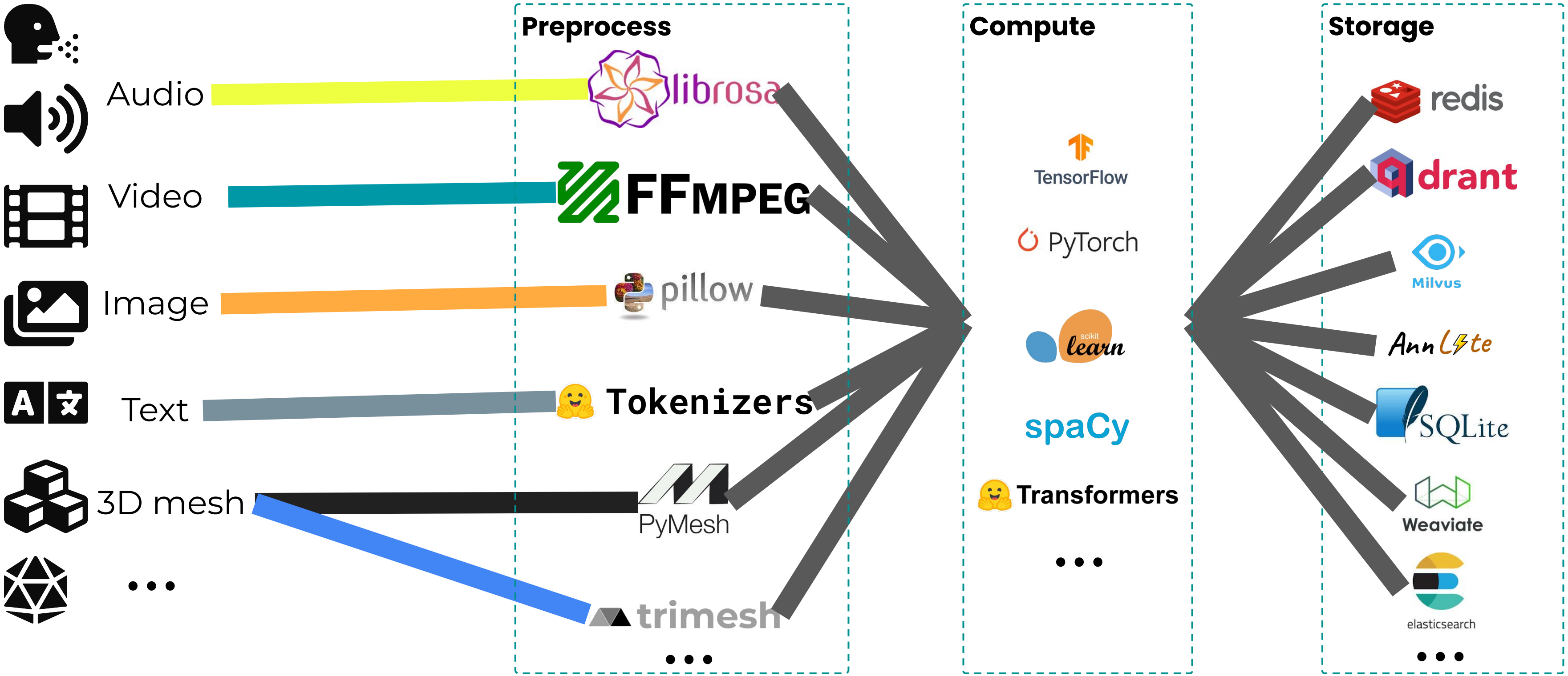
This tutorial may require technical knowledge. Familiarity with Python 3.7+ concepts like data classes could be helpful.

DocArray for representing, transiting, storing, searching multimodal data

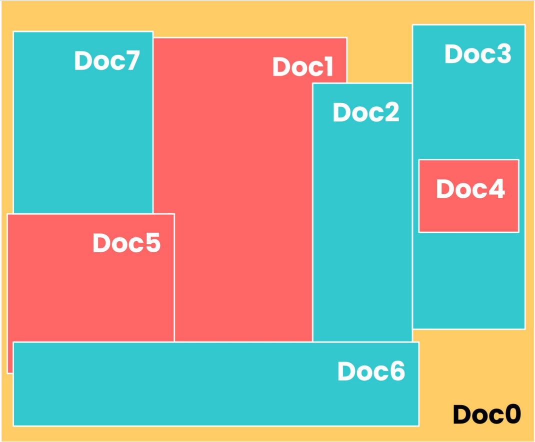
Representing multimodal data is a pain

- Lack of common interface for different modalities makes it difficult to work with multiple modalities at the same time.
- No easy way to represent unstructured and nested multimodal data.

Lack of common interface

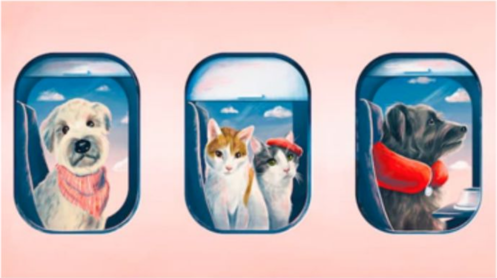


No easy way to represent unstructured nested multimodal data



- Unstructured document
- Nested content
- Different modalities (text, image, …)

DocArray way of representing multimodal data



By the Way A Post Travel Destination

Everything to know about flying with pets, from picking your seat to keeping your animal calm

By Nathan Diller

```python
from docarray import dataclass, Docume
from docarray.typing import Image, Text, .


@dataclass
class WPArticle:
    banner: Image
    headline: Text
    meta: JSON


a = WPArticle(
    banner='dog-cat-flight.png',
    headline='Everything to know about fl
    meta={
        'author': 'Nathan Diller',
        'column': 'By the Way - A Post Tr
    },
)

doc = Document(a)
```

Frequent data transfer over network is expensive

Multimodal data is processed by multiple models and models are usually deployed in a distributed way.

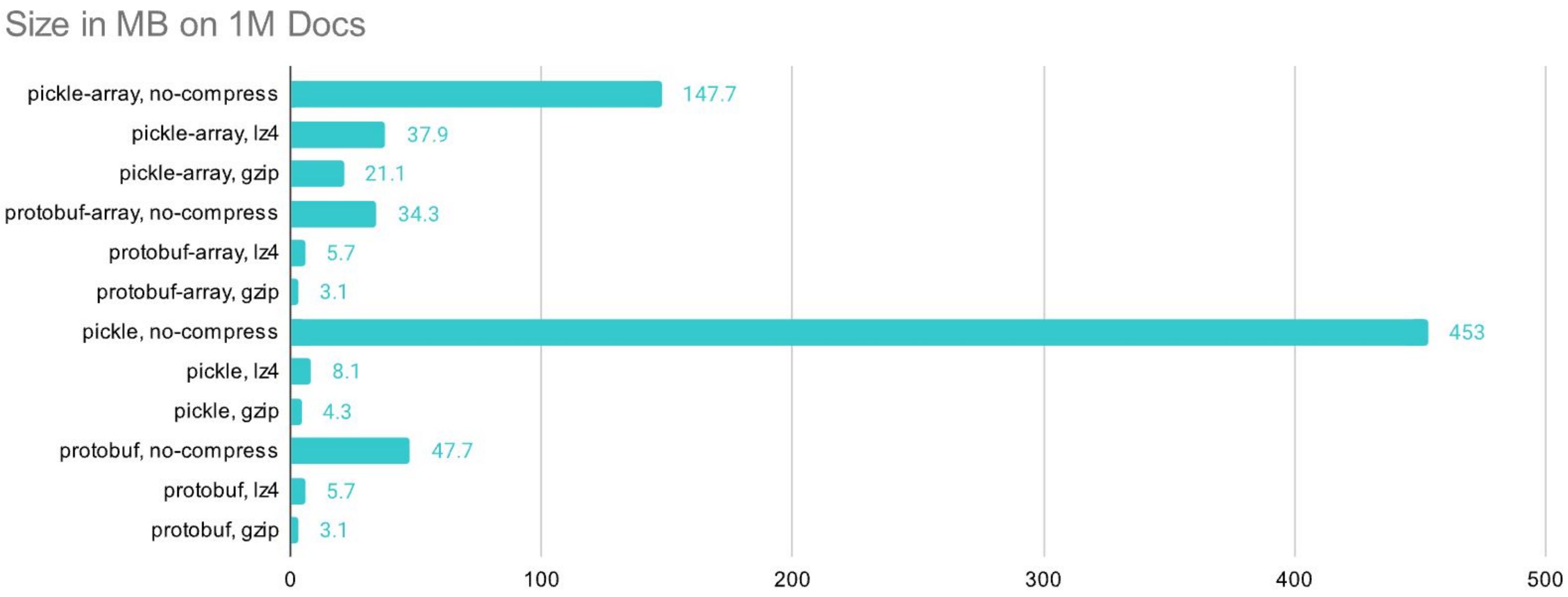
Data at rest	Data in use	Data in transit
Inactive data under very occasional changes, stored physically in database, warehouse, spreadsheet, archives, etc.	Active data under constant change, stored physically in database, warehouse, spreadsheet, etc.	Traversing a network or temporarily residing in computer memory to be read or updated

Performant serialization is important

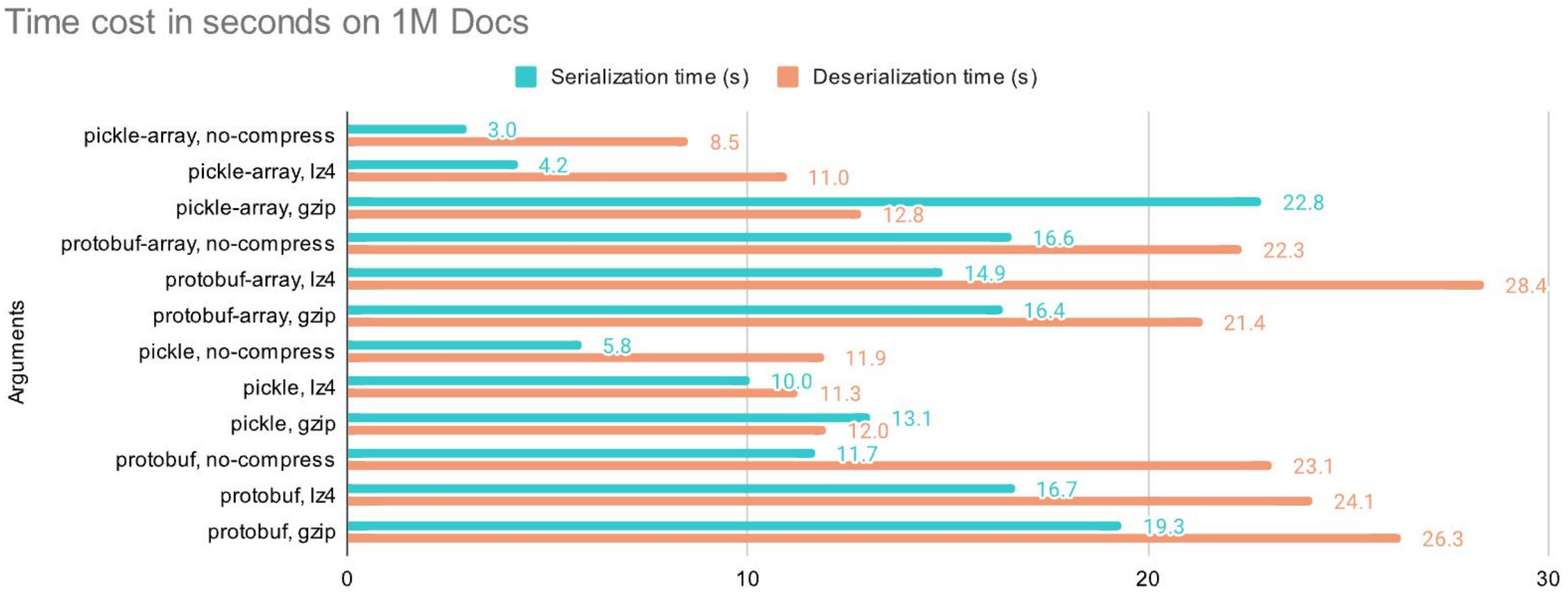
DocArray is designed to be “ready-to-wire” at anytime.

- JSON string: `.from_json()` / `.to_json()`
    - Pydantic model: `.from_pydantic_model()` / `.to_pydantic_model()`
- Bytes (compressed): `.from_bytes()` / `.to_bytes()`
    - Disk serialization: `.save_binary()` / `.load_binary()`
- Base64 (compressed): `.from_base64()` / `.to_base64()`
- Protobuf Message: `.from_protobuf()` / `.to_protobuf()`
- Python List: `.from_list()` / `.to_list()`
- Pandas Dataframe: `.from_dataframe()` / `.to_dataframe()`
- Cloud: `.push()` / `.pull()`

Binary serialization optimized for in-transit & at-rest



Binary serialization optimized for in-transit & at-rest



Storing nested data with databases is complicated

- Complex and nested schema are not directly supported in databases
- Explosion in numbers of vector databases with different APIs but no universal client

DocArray way of storing data

DocArray Storage

```python
from docarray import DocumentArray, Document

da = DocumentArray(storage='milvus',
                   config={'connection': 'example.db'})

with da:
    da.append(Document())
da.summary()
```

DocArray way of storing data

DocArray Storage

```python
from docarray import DocumentArray, Document

da = DocumentArray(storage='milvus',
                   config={'connectio

with da:
    da.append(Document())
da.summary()
```

```
'mivlus'
'qdrant'
'weaviate'
'elasticsearch'
'redis'
'opensearch'
'annlite'
'sqlite'
```

Vector Search via a consistent API

```python
from docarray import Document, DocumentArray
import numpy as np

n_dim = 3
da = DocumentArray(
    storage='annlite',
    config={'n_dim': n_dim, 'metric': 'Euclidean'},
)

with da:
    da.extend([Document(embedding=i * np.ones(n_dim)) for i in range(10)])

result = da.find(np.array([2, 2, 2]), limit=6)
result[:, 'embedding']
```

Vector Search via a consistent API

```python
from docarray import Document, DocumentArray
import numpy as np

n_dim = 3
da = DocumentArray(
    storage='annlite',
    config={'n_dim': n_dim, 'metric': 'Euclid
)

with da:
    da.extend([Document(embedding=i * np.ones

result = da.find(np.array([2, 2, 2]), limit=6
result[:, 'embedding']
```

Name	Construction	Vector search	Vector search + Filter	Filte
In memory	`DocumentArray()`	✅	✅	✅
`SQLite`	`DocumentArray(storage='sqlite')`	❌	❌	✅
`Weaviate`	`DocumentArray(storage='weaviate')`	✅	✅	✅
`Qdrant`	`DocumentArray(storage='qdrant')`	✅	✅	✅
`AnnLite`	`DocumentArray(storage='annlite')`	✅	✅	✅
`ElasticSearch`	`DocumentArray(storage='elasticsearch')`	✅	✅	✅
`Redis`	`DocumentArray(storage='redis')`	✅	✅	✅
`Milvus`	`DocumentArray(storage='milvus')`	✅	✅	✅

Quick Recap

- It’s like JSON, but for intensive computation.
- It’s like numpy.ndarray, but for unstructured data.
- It’s like pandas.DataFrame, but for nested and mixed media **data with embeddings.**
- It’s like Protobuf, but for data scientists and deep learning engineers.

Quick R

- It’s like JSO
- It’s like num
- It’s like pan
- It’s like Prot

	DocArray	`numpy.ndarray`	JSON	`pandas.DataFrame`	Protobuf
Tensor/matrix data	✅	✅	❌	✅	✔
Text data	✅	❌	✅	✅	✅
Media data	✅	❌	❌	❌	❌
Nested data	✅	❌	✅	❌	✅
Mixed data of the above four	✅	❌	❌	❌	❌
Easy to (de)serialize	✅	❌	✅	✅	✅
Data validation (of the output)	✅	❌	❌	❌	✅
Pythonic experience	✅	✅	❌	✔	❌
IO support for filetypes	✅	❌	❌	❌	❌
Deep learning framework support	✅	✅	❌	❌	❌
multi-core/GPU support	✅	✔	❌	❌	❌
Rich functions for data types	✅	❌	❌	✅	❌

ings.

Hands-on DocArray

Install DocArray

To install `DocArray (0.33)`, you can use the following command:

```
pip install "docarray[full]"
```

https://docs.docarray.org/

For old DocArray, more compatibility and features

```
pip install "docarray[full]"==0.21
```

Representing data – Document

At the heart of `DocArray` lies the concept of `BaseDoc`.

The following Python code defines a `BannerDoc` class that can be used to represent the data of a website banner:

```python
from docarray import BaseDoc
from docarray.typing import ImageUrl


class BannerDoc(BaseDoc):
    image_url: ImageUrl
    title: str
    description: str
```

Representing data – Document

You can then instantiate a `BannerDoc` object and access its attributes:

```
banner = BannerDoc(
    image_url='https://example.com/image.png',
    title='Hello World',
    description='This is a banner',
)

assert banner.image_url == 'https://example.com/image.png'
assert banner.title == 'Hello World'
assert banner.description == 'This is a banner'
```

Representing multimodal data with nested structure

Let's say you want to represent a YouTube video in your application, perhaps to build a search system for YouTube videos.

A YouTube video is not only composed of a video, but also has a title, description, thumbnail (and more, but let's keep it simple).

All of these elements are from different modalities:

the title and description are text,

the thumbnail is an image,

and the video itself is, well, a video.

DocArray lets you represent all of this multimodal data in a single object.

Representing multimodal data with nested structure

First for the thumbnail image:

```
from docarray import BaseDoc
from docarray.typing import ImageUrl, ImageBytes


class ImageDoc(BaseDoc):
    url: ImageUrl
    bytes: ImageBytes = (
        None  # bytes are not always loaded in memory, so we make it optional
    )
```

Representing multimodal data with nested structure

Then for the video itself:

```
from docarray import BaseDoc
from docarray.typing import VideoUrl, VideoBytes


class VideoDoc(BaseDoc):
    url: VideoUrl
    bytes: VideoBytes = (
        None  # bytes are not always loaded in memory, so we make it optional
    )
```

Representing multimodal data with nested structure

All the elements that compose a YouTube video are ready:

```python
from docarray import BaseDoc


class YouTubeVideoDoc(BaseDoc):
    title: str
    description: str
    thumbnail: ImageDoc
    video: VideoDoc
```

Representing multimodal data with nested structure

All the elements that compose a YouTube video are ready:

```
from docarray import BaseDoc


class YouTubeVideoDoc(BaseDoc):
    title: str
    description: str
    thumbnail: ImageDoc
    video: VideoDoc
```

You see here that `ImageDoc` and `VideoDoc` are also BaseDoc, and they are later used inside another BaseDoc`. This is what we call nested data representation.

BaseDoc can be nested to represent any kind of data hierarchy.

Representing multimodal data with nested structure

All the elements that compose a YouTube video are ready:

```
from docarray import BaseDoc


class YouTubeVideoDoc(BaseDoc):
    title: str
    description: str
    thumbnail: ImageDoc
    video: VideoDoc
```

You see here that `ImageDoc` and `VideoDoc` are also BaseDoc, and they are later used inside another BaseDoc`. This is what we call nested data representation.

BaseDoc can be nested to represent any kind of data hierarchy.

This representation can be used to send or store data. You can even use it directly to train a machine learning Pytorch model on this representation.

Recap: representing multimodal data

- "Dataclass" look and feel, for defining the structure
- Strong typing, for defining modality
  - Python built-in types
  - Numpy types
  - URI types
    - **Text**
    - **Image**
    - **Audio**
    - **Video**
    - **Mesh3D**
    - **PointCloud3D**
  - Tensor types
    - **ImageTensor**
    - **AudioTensor**
    - **VideoTensor**
    - **Embedding**
  - **Optional[]**

```python
from docarray import BaseDoc
from docarray.typing import ImageUrl, ImageBytes


class ImageDoc(BaseDoc):
    url: ImageUrl
    bytes: ImageBytes = (
        None  # bytes are not always loaded in m
    )
```

Representing an array of multimodal data

The fundamental building block of DocArray is the `BaseDoc` class which represents a *single* document, a *single* datapoint.

However, in machine learning we often need to work with an *array* of documents, and an *array* of data points.

We introduce

- `DocList` which is **a Python list** of `BaseDoc`s
- `DocVec` which is **a column-based representation** of `BaseDoc`s

Example of DocList

First you need to create a `Doc` class, our data schema. Let's say you want to represent a banner with an image, a title and a description:

```
from docarray import BaseDoc, DocList
from docarray.typing import ImageUrl


class BannerDoc(BaseDoc):
    image: ImageUrl
    title: str
    description: str
```

Example of DocList

First you need to create a `Doc` class, our data schema. Let's say you want to represent a banner with an image, a title and a description:

```
from docarray import BaseDoc, DocList
from docarray.typing import ImageUrl


cl
```

Let's instantiate several `BannerDoc` s:

```
banner1 = BannerDoc(
    image='https://example.com/image1.png',
    title='Hello World',
    description='This is a banner',
)

banner2 = BannerDoc(
    image='https://example.com/image2.png',
    title='Bye Bye World',
    description='This is (distopic) banner',
)
```

Example of DocList

`DocList` and `DocVec` are both `AnyDocArray`s. The following section will use `DocList` as an example, but the same applies to `DocVec`.

You can now collect them into a `DocList` of `BannerDoc` s:

```
docs = DocList[BannerDoc]([banner1, banner2])

docs.summary()
```

```
┌────────── DocList Summary ──────────┐
│                                     │
│   Type     DocList[BannerDoc]       │
│   Length   2                        │
│                                     │
└─────────────────────────────────────┘
┌───── Document Schema ─────┐
│                           │
│   BannerDoc               │
│   ├── image: ImageUrl     │
│   ├── title: str          │
│   └── description: str    │
│                           │
└───────────────────────────┘
```

Example of DocList

You can access documents inside it with the usual Python array API:

```
print(docs[0])
```

```
BannerDoc(image='https://example.com/image1.png', title='Hello World', description
```

or iterate over it:

```
for doc in docs:
    print(doc)
```

```
BannerDoc(image='https://example.com/image1.png', title='Hello World', description
BannerDoc(image='https://example.com/image2.png', title='Bye Bye World', descripti
```

Accessing member attribute at array level

At the document level:

```
print(banner1.image)
```

```
https://example.com/image1.png'
```

At the Array level:

```
print(docs.image)
```

```
['https://example.com/image1.png', 'https://example.com/image2.png']
```

Accessing member attribute at array level

At the document level:

```
print(banner1.image)
```

```
https://example.com/image1.png'
```

At the Array level:

```
print(docs.image)
```

```
['https://example.com/image1.png', 'h
```

You can even access the attributes of the nested `BaseDoc` at the Array level:

```
print(docs.banner.image)
```

```
['https://example.com/image1.png', 'https://example.com/image2.png']
```

This is just the same way that you would do it with BaseDoc:

```
print(page1.banner.image)
```

```
'https://example.com/image1.png'
```

DocList[DocType] syntax

`DocList[DocType]` creates a custom `DocList` that can only contain `DocType` Documents.

Non-typing DocList for heterogeneous data

```python
from docarray import BaseDoc, DocList
from docarray.typing import ImageUrl, AudioUrl


class ImageDoc(BaseDoc):
    url: ImageUrl


class AudioDoc(BaseDoc):
    url: AudioUrl


docs = DocList(
    [
        ImageDoc(url='https://example.com/image1.png'),
        AudioDoc(url='https://example.com/audio1.mp3'),
    ]
)
```

Strong-typing DocList for homogeneous data

```python
try:
    docs = DocList[ImageDoc](
        [
            ImageDoc(url='https://example.com/image1.png'),
            AudioDoc(url='https://example.com/audio1.mp3'),
        ]
    )
except ValueError as e:
    print(e)
```

```
ValueError: AudioDoc(
    id='e286b10f58533f48a0928460f0206441',
    url=AudioUrl('https://example.com/audio1.mp3', host_type='domain')
) is not a <class '__main__.ImageDoc'>
```

DocList vs DocVec

`DocList` is based on Python Lists. You can append, extend, insert, pop, and so on. In DocList, data is individually owned by each `BaseDoc` collect just different Document references.

Use `DocList` when you want to be able to rearrange or re-rank your data. One flaw of `DocList` is that none of the data is contiguous in memory, so you cannot leverage functions that require contiguous data without first copying the data in a continuous array.

`DocVec` is a columnar data structure. `DocVec` is always an array of homogeneous Documents. The idea is that every attribute of the `BaseDoc` will be stored in a contiguous array: a column.

DocList vs DocVec

Let's say you want to embed a batch of Images:

```
def embed(image: NdArray['batch_size', 3, 224, 224]):
    ...
```

DocList vs DocVec

```
from docarray import BaseDoc
from docarray.typing import NdArray


class ImageDoc(BaseDoc):
    image: NdArray[
        3, 224, 224
    ] = None  # [3, 224, 224] this just mean we know in advance the shape of the te
```

DocList vs DocVec

```python
from docarray import BaseDoc
from docarray.typing import NdArray


class I
    ima

    ] =
```

```python
from docarray import DocList
import numpy as np

docs = DocList[ImageDoc](
    [ImageDoc(image=np.random.rand(3, 224, 224)) for _ in range(10)]
)

embed (np.stack(docs.image))
...
embed (np.stack(docs.image))
```

DocList vs DocVec

```
from docarray import BaseDoc
from docarray.typing import NdArray


class ImageDoc(BaseDoc):
    image:
        3,
    ] = No
```

```
from docarray import DocVec
import numpy as np

docs = DocVec[ImageDoc](
    [ImageDoc(image=np.random.rand(3, 224, 224)) for _ in range(10)]
)

embed(docs.image)
```

Access the view of Document in DocVec

If you access a document inside a `DocVec` you will get a document view. A document view is a view of the columnar data structure which looks and behaves like a `BaseDoc` instance. It is a `BaseDoc` instance but with a different way to access the data.

```
from docarray import DocVec

docs = DocVec[ImageDoc](
    [ImageDoc(image=np.random.rand(3, 224, 224)) for _ in range(10)]
)

my_doc = docs[0]

assert my_doc.is_view()  # True
```

whereas with DocList:

```
docs = DocList[ImageDoc](
    [ImageDoc(image=np.random.rand(3, 224, 224)) for _ in range(10)]
)

my_doc = docs[0]

assert not my_doc.is_view()  # False
```

Access the view of Document in DocVec

If you access a document inside a `DocVec` you will get a document view. A document view is a view of the columnar data structure which looks and behaves like a `BaseDoc` instance. It is a `BaseDoc` instance but with a different way to access the data.

you should use `DocVec` when you need to work with contiguous data, and you should use `DocList` when you need to rearrange or extend your data.

```python
docs = DocList[ImageDoc](
    [ImageDoc(image=np.random.rand(3, 224, 224)) for _ in range(10)]
)

my_doc = docs[0]

assert not my_doc.is_view()  # False
```

Storing & retrieving via Vector Database

```python
from docarray import DocList, BaseDoc
from docarray.index import HnswDocumentIndex
import numpy as np

from docarray.typing import ImageUrl, ImageTensor, NdArray


class ImageDoc(BaseDoc):
    url: ImageUrl
    tensor: ImageTensor
    embedding: NdArray[128]


# create some data
dl = DocList[ImageDoc](
    [
        ImageDoc(
            url="https://upload.wikimedia.org/wikipedia/commons/2/2f/Alpamayo.jpg",
            tensor=np.zeros((3, 224, 224)),
            embedding=np.random.random((128,)),
        )
        for _ in range(100)
    ]
)

# create a Document Index
index = HnswDocumentIndex[ImageDoc](work_dir='/tmp/test_index2')


# index your data
index.index(dl)

# find similar Documents
query = dl[0]
results, scores = index.find(query, limit=10, search_field='embedding')
```

Storing & retrieving via Vector Database

```python
from docarray import DocList, BaseDoc
from docarray.index import HnswDocumentIndex
import numpy as np

from docarray.typing import ImageUrl, ImageTensor, NdArray


class ImageDoc(BaseDoc):
    url: ImageUrl
    tensor: ImageTensor
    embedding: NdArray[128]


# create some data
dl = DocList[ImageDoc](
    [
        ImageDoc(
            url="https://upload.wikimedia.org/wikipedia/commons/2/2f/Alpamayo.jpg",
            tensor=np.zeros((3, 224, 224)),
            embedding=np.random.random((128,)),
        )
        for _ in range(100)
    ]
)

# create a Document Index
index = HnswDocumentIndex[ImageDoc](work_dir='/tmp/test_index2')


# index your data
index.index(dl)

# find similar Documents
query = dl[0]
results, scores = index.find(query, limit=10, search_field='embedding')
```

Document Index: ORM for vector DBs

Document Index provides a unified interface to a number of vector databases.

You can think of Document Index as an **ORM for vector databases**.

Currently, DocArray supports the following vector databases:

- Weaviate | Docs
- Qdrant | Docs
- Elasticsearch v7 and v8 | Docs
- HNSWlib | Docs

*Old DocArray v0.21 supports Milvus, Redis, Opensearch

Construct a HNSWDocumentIndex

To use HnswDocumentIndex, you need to install extra dependencies with the following command: `pip install "docarray[hnswlib]"`

To create a Document Index, you first need a document that defines the schema of your index:

```
from docarray import BaseDoc
from docarray.index import HnswDocumentIndex
from docarray.typing import NdArray


class MyDoc(BaseDoc):
    embedding: NdArray[128]
    text: str


db = HnswDocumentIndex[MyDoc](work_dir='./my_test_db')
```

Construct a HNSWDocumentIndex

To use HnswDocumentIndex, you need to install extra dependencies with the following command: `pip install "docarray[hnswlib]"`

To create a Document Index, you first need a document th

```
from docarray import BaseDoc
from docarray.index import HnswDocumentIndex
from docarray.typing import NdArray


class MyDoc(BaseDoc):
    embedding: NdArray[128]
    text: str


db = HnswDocumentIndex[MyDoc](work_dir='./my_tes
```

In this code snippet, `HnswDocumentIndex` takes a schema of the form of `MyDoc`. The Document Index then *creates a column for each field in* `MyDoc`.

Construct a HNSWDocumentIndex

To use HnswDocumentIndex, you need to install extra dependencies with the following command: `pip install "docarray[hnswlib]"`

To create a Document Index, you first need a document th

```
from docarray import BaseDoc
from docarray.index import HnswDocumentIndex
from docarray.typing import NdArray


class MyDoc(BaseDoc):
    embedding: NdArray[128]
    text: str


db = HnswDocumentIndex[MyDoc](work_dir='./my_tes
```

In this code snippet, `HnswDocumentIndex` takes a schema of the form of `MyDoc`. The Document Index then *creates a column for each field in* `MyDoc`.

The column types in the backend database are determined by the type hints of the document's fields. Optionally, you can customize the database types for every field.

Construct a HNSWDocumentIndex

To use HnswDocumentIndex, you need to install extra dependencies with the following command: `pip install "docarray[hnswlib]"`

To create a Document Index, you first need a document th

```
from docarray import BaseDoc
from docarray.index import HnswDocumentIndex
from docarray.typing import NdArray


class MyDoc(BaseDoc):
    embedding: NdArray[128]
    text: str


db = HnswDocumentIndex[MyDoc](work_dir='./my_tes
```

In this code snippet, `HnswDocumentIndex` takes a schema of the form of `MyDoc`. The Document Index then *creates a column for each field in* `MyDoc`.

The column types in the backend database are determined by the type hints of the document's fields. Optionally, you can customize the database types for every field.

Most vector databases need to know the dimensionality of the vectors that will be stored. Here, that is automatically inferred from the type hint of the `embedding` field: `NdArray[128]` means that the database will store vectors with 128 dimensions.

Index data

Now that you have a Document Index, you can add data to it, using the index() method:

```
import numpy as np
from docarray import DocList

# create some random data
docs = DocList[MyDoc](
    [MyDoc(embedding=np.random.rand(128), text=f'text {i}') for i in range(100)]
)

# index the data
db.index(docs)
```

Index data

Now that you have a Document Index, you can add data to it, using the index() method:

```
import numpy as np
from docarray import DocList

# create some random data
docs = DocList[MyDoc](
    [MyDoc(embedding=np.random.rand(128), text=f'text {i}'
)

# index the data
db.index(docs)
```

```
from docarray import BaseDoc
from docarray.index import HnswDocume
from docarray.typing import NdArray


class MyDoc(BaseDoc):
    embedding: NdArray[128]
    text: str


db = HnswDocumentIndex[MyDoc](work_dir='./my_test_db')
```

As you can see, `DocList[MyDoc]` and `HnswDocumentIndex[MyDoc]` are both parameterized with `MyDoc`. This means that they share the same schema, and in general, the schema of a Document Index and the data that you want to store need to have compatible schemas

Vector search

Search by Document | Search by raw vector

```python
# create a query Document
query = MyDoc(embedding=np.random.rand(128), text='query')

# find similar Documents
matches, scores = db.find(query, search_field='embedding', limit=5)

print(f'{matches=}')
print(f'{matches.text=}')
print(f'{scores=}')
```

Vector search

Search by Document | Search by raw vector

```python
# create a query Document
query = MyDoc(embedding=np.random.rand(128), text='query')

# find similar Documents
matches, scores = db.find(query, search_field='embedding', limit=5)

print(f'{matches=}')
print(f'{matches.text=}')
print(f'{scores=}')
```

Search by Document | **Search by raw vector**

```python
# create a query vector
query = np.random.rand(128)

# find similar Documents
matches, scores = db.find(query, search_field='embedding', limit=5)

print(f'{matches=}')
print(f'{matches.text=}')
print(f'{scores=}')
```

Vector search

Search by Document | Search by raw vector

```python
# create a query Document
query = MyDoc(embedding=np.random.rand(128), text='query')

# find similar Documents
matches, scores = db.find(query, search_field='embedding', limit=5)

print(f'{matches=}')
print(f'{matches.text=}')
print(f'{scores=}')
```

Search by Document | Search by raw vector

```python
# create a query vector
query = np.random.rand(128)

# find similar Documents
matches, scores = db.find(query, search_field='embedding', limit=5)

print(f'{matches=}')
print(f'{matches.text=}')
print(f'{scores=}')
```

Search by Documents | Search by raw vectors

```python
# create some query Documents
queries = DocList[MyDoc](
    MyDoc(embedding=np.random.rand(128), text=f'query {i}') for i in range(3)
)

# find similar Documents
matches, scores = db.find_batched(queries, search_field='embedding', limit=5)

print(f'{matches=}')
print(f'{matches[0].text=}')
print(f'{scores=}')
```

Vector search

Search by Document | Search by raw vector

```
# create a query Document
query = MyDoc(embedding=np.random.rand(128), text='query')

# find similar Documents
matches, scores = db.find(query, search_field='embedding', limit=5)

print(f'{matches=}')
print(f'{matches.text=}')
print(f'{scores=}')
```

Search by Document | **Search by raw vector**

```
# create a query vector
query = np.random.rand(128)

# find similar Documents
matches, scores = db.find(query, search_field='embedding', limit=5)

print(f'{matches=}')
print(f'{matches.text=}')
print(f'{scores=}')
```

Search by Documents | Search by raw vectors

```
# create some query Documents
queries = DocList[MyDoc](
    MyDoc(embedding=np.random.rand(128), text=f'query {i}') for i in range(3)
)

# find similar Documents
matches, scores = db.find_batched(queries, search_field='embedding', limit=5)

print(f'{matches=}')
print(f'{matches[0].text=}')
print(f'{scores=}')
```

Search by Documents | **Search by raw vectors**

```
# create some query vectors
query = np.random.rand(3, 128)

# find similar Documents
matches, scores = db.find_batched(query, search_field='embedding', limit=5)

print(f'{matches=}')
print(f'{matches[0].text=}')
print(f'{scores=}')
```

Hybrid search through the query builder

Document Index supports atomic operations for vector similarity search, text search and filter search.

To combine these operations into a single, hybrid search query, you can use the query builder that is accessible through build_query():

```
# prepare a query
q_doc = MyDoc(embedding=np.random.rand(128), text='query')

query = (
    db.build_query()  # get empty query object
    .find(query=q_doc, search_field='embedding')  # add vector similarity search
    .filter(filter_query={'text': {'$exists': True}})  # add filter search
    .build()  # build the query
)

# execute the combined query and return the results
results = db.execute_query(query)
print(f'{results=}')
```

Customize vector DB configuration

```
db = HnswDocumentIndex[MyDoc](work_dir='/tmp/my_db')

db.configure(
    default_column_config={
        np.ndarray: {
            'dim': -1,
            'index': True,
            'space': 'ip',
            'max_elements': 2048,
            'ef_construction': 100,
            'ef': 15,
            'M': 8,
            'allow_replace_deleted': True,
            'num_threads': 5,
        },
        None: {},
    }
)
```

Indexing and searching multimodal data

In the following example you can see a complex schema that contains nested Documents. The `YouTubeVideoDoc` contains a `VideoDoc` and an `ImageDoc`, alongside some "basic" fields:

Year in Review: 2021 in Graphic Design

Linus Boman

119K views • 1 year ago

```python
from docarray.typing import ImageUrl, VideoUrl, AnyTensor


# define a nested schema
class ImageDoc(BaseDoc):
    url: ImageUrl
    tensor: AnyTensor = Field(space='cosine', dim=64)


class VideoDoc(BaseDoc):
    url: VideoUrl
    tensor: AnyTensor = Field(space='cosine', dim=128)


class YouTubeVideoDoc(BaseDoc):
    title: str
    description: str
    thumbnail: ImageDoc
    video: VideoDoc
    tensor: AnyTensor = Field(space='cosine', dim=256)


# create a Document Index
doc_index = HnswDocumentIndex[YouTubeVideoDoc](work_dir='/tmp2')

# create some data
index_docs = [
    YouTubeVideoDoc(
        title=f'video {i+1}',
        description=f'this is video from author {10*i}',
        thumbnail=ImageDoc(url=f'http://example.ai/images/{i}', tensor=np.ones(64)),
        video=VideoDoc(url=f'http://example.ai/videos/{i}', tensor=np.ones(128)),
        tensor=np.ones(256),
    )
    for i in range(8)
]

# index the Documents
doc_index.index(index_docs)
```

Indexing and searching multimodal data

You can perform search on any nesting level by using the dunder operator to specify the field defined in the nested data.

```python
# create a query Document
query_doc = YouTubeVideoDoc(
    title=f'video query',
    description=f'this is a query video',
    thumbnail=ImageDoc(url=f'http://example.ai/images/1024', tensor=np.ones(64)),
    video=VideoDoc(url=f'http://example.ai/videos/1024', tensor=np.ones(128)),
    tensor=np.ones(256),
)

# find by the `youtubevideo` tensor; root level
docs, scores = doc_index.find(query_doc, search_field='tensor', limit=3)

# find by the `thumbnail` tensor; nested level
docs, scores = doc_index.find(query_doc, search_field='thumbnail__tensor', limit=3)

# find by the `video` tensor; neseted level
docs, scores = doc_index.find(query_doc, search_field='video__tensor', limit=3)

```

Nested DocList with subindex

Documents can be nested by containing a `DocList` of other documents, which is a slightly more complicated scenario than the previous one.

In this case, the nested DocList will be represented as a **new sub-index** (or table, collection, etc., depending on the database backend), that is linked with **the parent index** (table, collection, ...).

```python
class ImageDoc(BaseDoc):
    url: ImageUrl
    tensor_image: AnyTensor = Field(space='cosine', dim=64)


class VideoDoc(BaseDoc):
    url: VideoUrl
    images: DocList[ImageDoc]
    tensor_video: AnyTensor = Field(space='cosine', dim=128)


class MyDoc(BaseDoc):
    docs: DocList[VideoDoc]
    tensor: AnyTensor = Field(space='cosine', dim=256)


# create a Document Index
doc_index = HnswDocumentIndex[MyDoc](work_dir='/tmp3')

# create some data
index_docs = [
    MyDoc(
        docs=DocList[VideoDoc](
            [
                VideoDoc(
                    url=f'http://example.ai/videos/{i}-{j}',
                    images=DocList[ImageDoc](
                        [
                            ImageDoc(
                                url=f'http://example.ai/images/{i}-{j}-{k}',
                                tensor_image=np.ones(64),
                            )
                            for k in range(10)
                        ]
                    ),
                    tensor_video=np.ones(128),
                )
                for j in range(10)
            ]
        ),
        tensor=np.ones(256),
    )
    for i in range(10)
]

# index the Documents
doc_index.index(index_docs)
```

Search by subindex

```
# find by the `VideoDoc` tensor
root_docs, sub_docs, scores = doc_index.find_subindex(
    np.ones(128), subindex='docs', search_field='tensor_video', limit=3
)

# find by the `ImageDoc` tensor
root_docs, sub_docs, scores = doc_index.find_subindex(
    np.ones(64), subindex='docs__images', search_field='tensor_image', limit=3
)

```

Transiting data over network

Sending via REST API/JSON -> Backend: FastAPI

```python
import numpy as np
from fastapi import FastAPI
from docarray.base_doc import DocArrayResponse
from docarray import BaseDoc
from docarray.documents import ImageDoc
from docarray.typing import NdArray

class InputDoc(BaseDoc):
    img: ImageDoc
    text: str


class OutputDoc(BaseDoc):
    embedding_clip: NdArray
    embedding_bert: NdArray


app = FastAPI()


@app.post("/embed/", response_model=OutputDoc, response_class=DocArrayResponse)
async def create_item(doc: InputDoc) -> OutputDoc:
    ## call my fancy model to generate the embeddings
    doc = OutputDoc(
        embedding_clip=embed(doc.image)), embedding_bert=embed(doc.text))
    )
    return doc

```

Sending via gRPC/ws -> Backend: Jina microservice

Transiting data over network

Sending via REST API/JSON -> Backend: FastAPI

Sending via gRPC/ws -> Backend: Jina microservice

```
import numpy as np
from fastapi import FastAPI
from docarray.base_doc import DocArrayResponse
from docarray import BaseDoc
from docarray.documents import ImageDoc
from docarray.typing import NdArray

class InputDoc(BaseDoc):
    img: ImageDoc
    text: str


class OutputDoc(BaseDoc):
    embedding_clip: NdArray
    embedding_bert: NdArray


app = FastAPI()


@app.post("/e
async def cre
    ## call m
    doc = Out
        embed
    )
    return do
```

```
async with AsyncClient(app=app, base_url="http://test") as ac:
    response = await ac.post("/doc/", data=docs.to_json())  # sending docs as json

assert response.status_code == 200
# You can read FastAPI's response in the following way
docs = DocList[TextDoc].from_json(response.content.decode())
```

Transiting data over network

Sending via gRPC/ws -> Backend: Jina microservice

```python
class WhisperExecutor(Executor):
    def __init__(self, device: str, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.model = whisper.load_model("medium.en", device=device)

    @requests
    def transcribe(self, docs: DocList[AudioURL], **kwargs) -> DocList[Response]:
        response_docs = DocList[Response]()
        for doc in docs:
            transcribed_text = self.model.transcribe(str(doc.audio))['text']
            response_docs.append(Response(text=transcribed_text))

        return response_doc

```

Transiting data over network

Sending via gRPC/ws -> Backend: Jina microservice

```
class WhisperExecutor(Executor):
    def __init__(self, device: str, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.model = whisper.load_model("medium.en", device=device)

    @requests
    def transcribe(self, docs: DocList[AudioURL], **kwargs) -> DocList[Response]:
        response_docs = DocList[Respo
        for doc in docs:
            transcribed_text = self.m
            response_docs.append(Resp

        return response_doc

```

```
dep = Deployment(
    uses=WhisperExecutor, uses_with={'device': "cpu"}, port=12349, timeout_ready=-1
)

with dep:
    docs = d.post(
        on='/transcribe',
        inputs=[AudioURL(audio='resources/audio.mp3')],
        return_type=DocList[Response],
    )

print(docs[0].text)

```

Agenda

- Preliminary: multimodal AI
- Opensource package: DocArray
  - Motivation
  - Representing data
  - Transiting data
  - Storing data
  - Retrieving data
- **Multimodal at scale in production**

This tutorial may require technical knowledge. Familiarity with Python 3.7+ concepts like data classes could be helpful.

An end to end example

https://docs.docarray.org/how_to/multimodal_training_and_serving/

Thanks for your attention

jina.ai
@JinaAI_
han.xiao@jina.ai

Berlin · Beijing · Shenzhen