FOCS 2025

A History of Approximate Nearest Neighbor Search from an Applications Perspective

Workshop of Approximate Nearest Neighbor Search: Bridging Theoretical Foundations and Industrial Frontiers at FOCS 2025
Dec 14, 2025 @Sydney, Australia

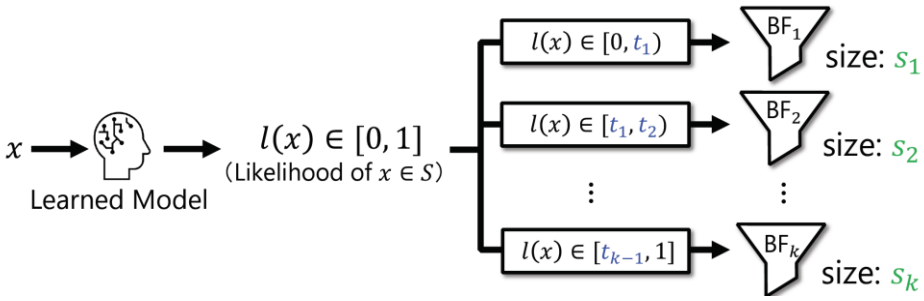
Yusuke Matsui
The University of Tokyo

Yusuke Matsui

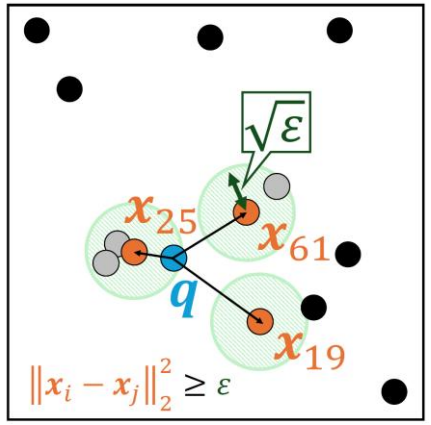
Lecturer (Assistant Professor), the University of Tokyo, Japan

http://yusukematsui.me @utokyo_bunny @matsui528

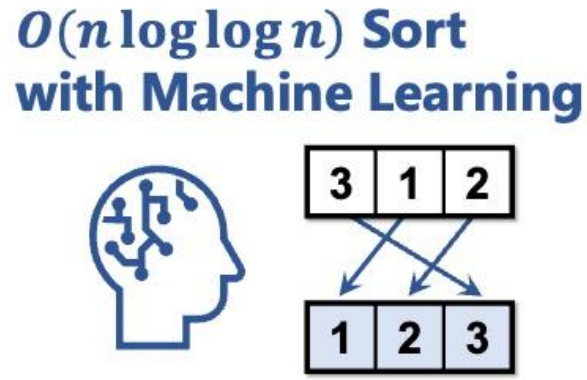
- ✓ Computer vision
- ✓ Data structure + Machine Learning



ML-enhanced Bloom Filter
[Sato & Matsui, NeurIPS 23]



Diverse nearest neighbor search
[Matsui, CVPR 25]



ML-enhanced Sorting
[Sato & Matsui, TMLR 25]

The 1st Workshop on Vector Databases (VecDB)

VecDB@ICML2025, 18 July 2025, Vancouver, Canada

Speakers

Matthijs Douze (Meta)

Faiss

Hiroyuki Ootomo (NVIDIA)

CAGRA

Prashant Pandey (Northeastern University)

Organizers

Yusuke Matsui (The University of Tokyo)

Martin Aumüller (IT University of Copenhagen)

Magdalen Dobson Manohar (Carnegie Mellon University)

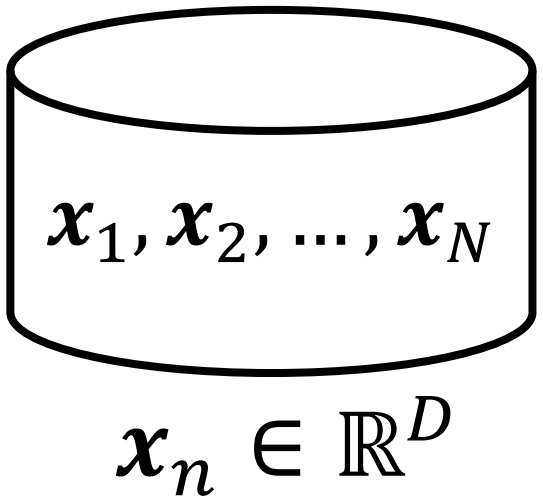
Harsha Vardhan Simhadri (Microsoft)

DiskANN

- Organized the 1st workshop on Vector Databases at ICML 2025
- **Planning the 2nd edition in 2026**
- Forum for NN researchers from various fields
- Welcome your submissions!

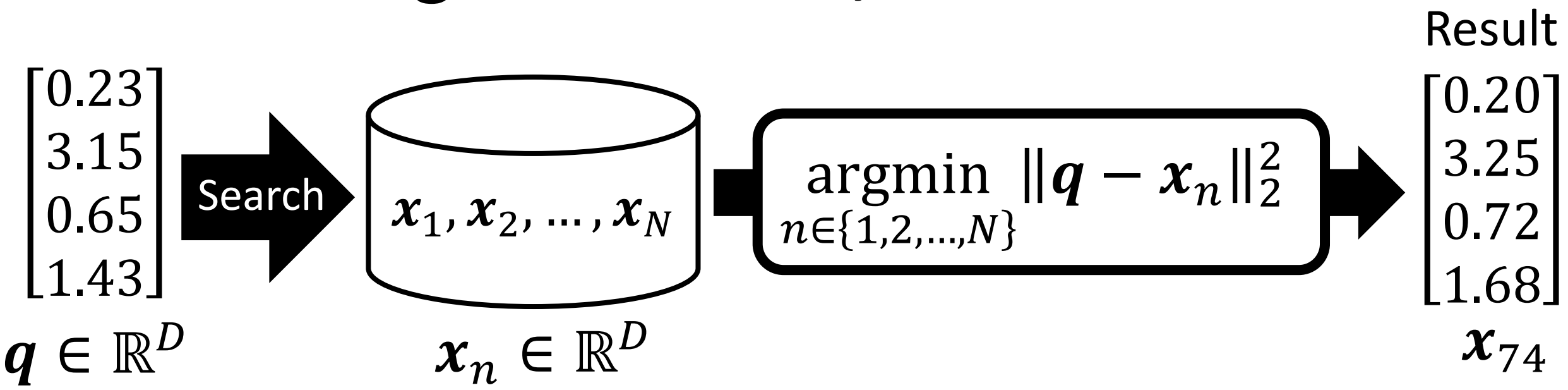
The best paper was a theory paper!
H. Xu, P. Indyk, S. Silwal, "Bi-metric Framework for Efficient Nearest Neighbor Search"

Nearest Neighbor Search; NN



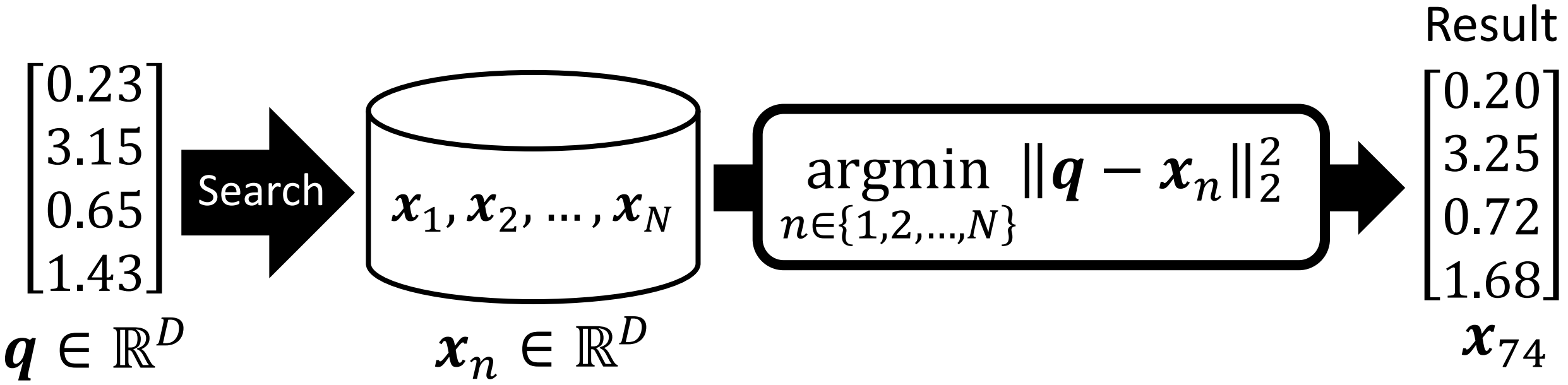
- N D-dim database vectors: $\{\boldsymbol{x}_n\}_{n=1}^{N}$

Nearest Neighbor Search; NN



- 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

Why NN/ANN?

- NN/ANN is an interesting research area because:
  - ✓ it's a **pure theory** (yes, this is FOCS WS!)
  - ✓ at the same time, it is **directly used in applications**, e.g., Vector DBs

- Several research areas (CV, NLP, DB, ...)

- Because of RAG and Vector DB, ANN has become popular more and more

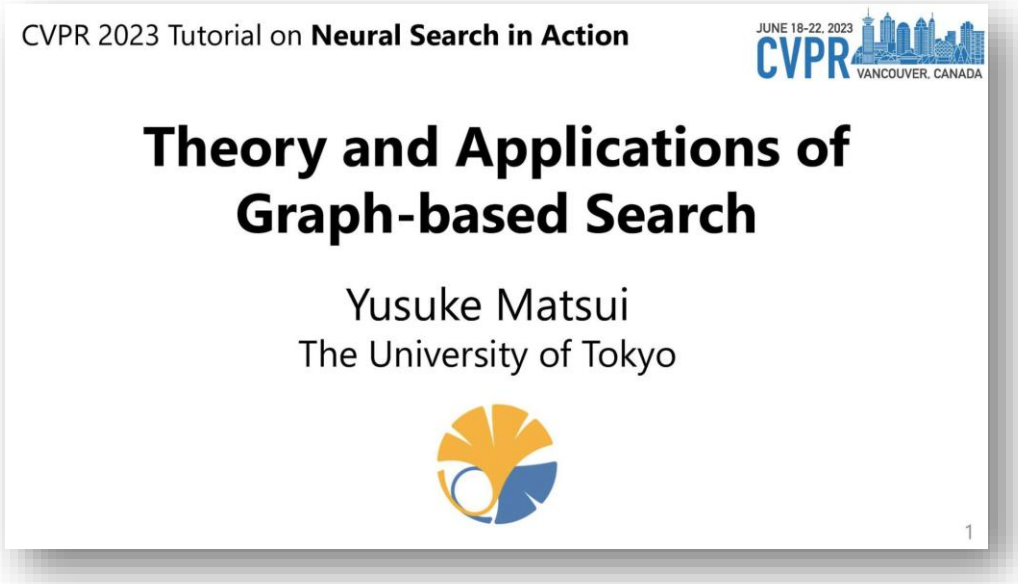
Jie+, "Survey of Vector Database Management Systems", arXiv 2023

Name	Type	Sub-Type	Vector Ex.	Vector Ap
EuclidesDB (2018) [6]	Nat.	Vec.	✓	✓
Vearch (2018) [17,81]	Nat.	Vec.	✗	✓
Pinecone (2019) [9]	Nat.	Vec.	✗	✓
Vald (2020) [8]	Nat.	Vec.	✓	✓
Chroma (2022) [14]	Nat.	Vec.	✗	✓
Weaviate (2019) [5]	Nat.	Mix	✗	✓
Milvus (2021) [12,125]	Nat.	Mix	✓	✓
NucliaDB (2021) [13]	Nat.	Mix	✗	✓
Qdrant (2021) [10]	Nat.	Mix	✓	✓
Manu (2022) [63]	Nat.	Mix	✓	✓
Marqo (2022) [11]	Nat.	Mix	✗	✓
Vespa (2020) [18]	Ext.	NoSQL	✓	✓
Cosmos DB (2023)	Ext.	NoSQL	✗	✓
MongoDB (2023)	Ext.	NoSQL	✗	✓
Neo4j (2023)	Ext.	NoSQL	✗	✓
Redis (2023)	Ext.	NoSQL	✓	✓
AnalyticDB-V (2020) [133]	Ext.	Rel.	✓	✓
PASE+PG (2020) [139]	Ext.	Rel.	✓	✓
pgvector+PG (2021) [7]	Ext.	Rel.	✓	✓
SingleStoreDB (2022) [19,105]	Ext.	Rel.	✓	✓
ClickHouse (2023) [20]	Ext.	Rel.	✓	✓
MyScale (2023) [21]	Ext.	Rel.	✓	✗

This talk

- Reorganized and expanded version of my previous tutorials of CVPR2020 and CVPR2023
- See the original tutorials for more detailed contents

- CVPR 2020 Tutorial on Image Retrieval in the Wild
- Y. Matsui, “Billion-scale Approximate Nearest Neighbor Search”
- https://speakerdeck.com/matsui_528/cvpr20-tutorial-billion-scale-approximate-nearest-neighbor-search



- CVPR 2023 Tutorial on Neural Search in Action
- Y. Matsui, “Theory and Applications of Graph-based Search”
- https://speakerdeck.com/matsui_528/cvpr23-tutorial-theory-and-applications-of-graph-based-search

Outline

1. History from an applications perspective

2. Importance of implementation:
   nearest neighbor search in faiss

3. Basics of modern baseline: graph-based search

Outline

1. **History from an applications perspective**

2. Importance of implementation:
   nearest neighbor search in faiss

3. Basics of modern baseline: graph-based search

What tasks has ANN been used for?

Computer Vision (CV):
SIFT + BoF

CV:
Image search

Natural Language Processing (NLP) &
Information Retrieval:
Text search (Dense? Sparse?)

Machine Learning:
kNN classification, metric learning

CV & NLP:
CLIP multimodal search

Large Language Models:
Retrieval Augmented Generation (RAG)

Database:
VectorDB

What tasks has ANN been used for?

Computer Vision (CV):
SIFT + BoF

CV:
Image search

Natural Language Processing (NLP) &
Information Retrieval:
Text search (Dense? Sparse?)

Machine Learning:
kNN classification, metric learning

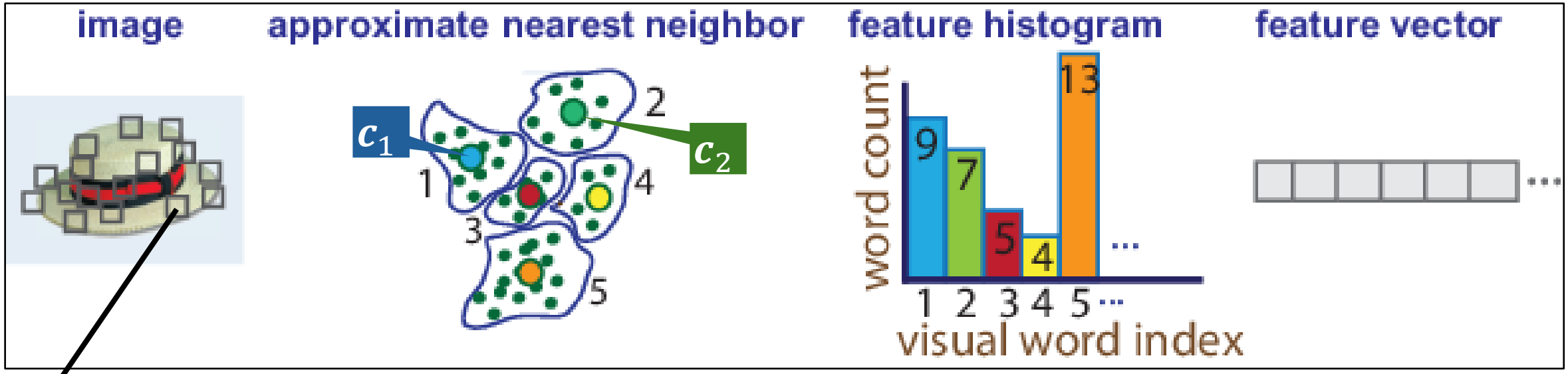
CV & NLP:
CLIP multimodal search

Large Language Models:
Retrieval Augmented Generation (RAG)

Database:
VectorDB

SIFT (local feature) + BoF (Bag-of-features) + SVM

https://jp.mathworks.com/help/vision/ug/image-classification-with-bag-of-visual-words.html



$\boldsymbol{x} \in \mathbb{R}^{128}$

Extract a local patch

Given codewords $\{\boldsymbol{c}_k\}$, find the closest one

$$k^* = \underset{k \in \{1,\ldots,5\}}{\operatorname{argmin}} \|\boldsymbol{x} - \boldsymbol{c}_k\|_2^2$$

Create a histogram, run SVM, recognize an image…

- This is nearest neighbor search!
- K is 10^3 to 10^4
- Must be in memory

- To compute BoF fast, several practical ANN technologies have been invented in the CV area in the 2000s – 2010s… E.g.: Product Quantization
- Good old days…. 😭

What tasks has ANN been used for?

Computer Vision (CV):
SIFT + BoF

CV:
Image search

Natural Language Processing (NLP) &
Information Retrieval:
Text search (Dense? Sparse?)

Machine Learning:
kNN classification, metric learning

CV & NLP:
CLIP multimodal search

Large Language Models:
Retrieval Augmented Generation (RAG)

Database:
VectorDB

Image 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

Image Search

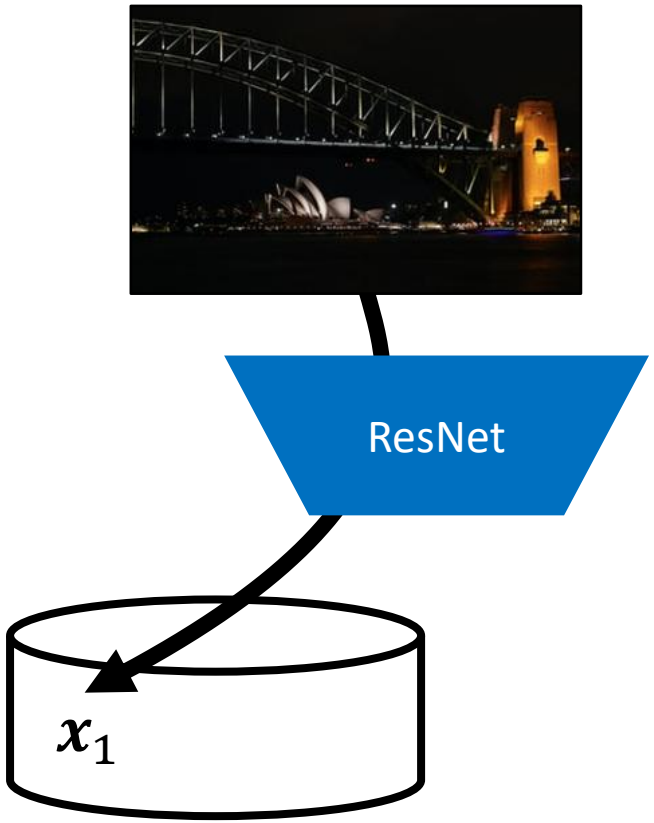


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Image Search

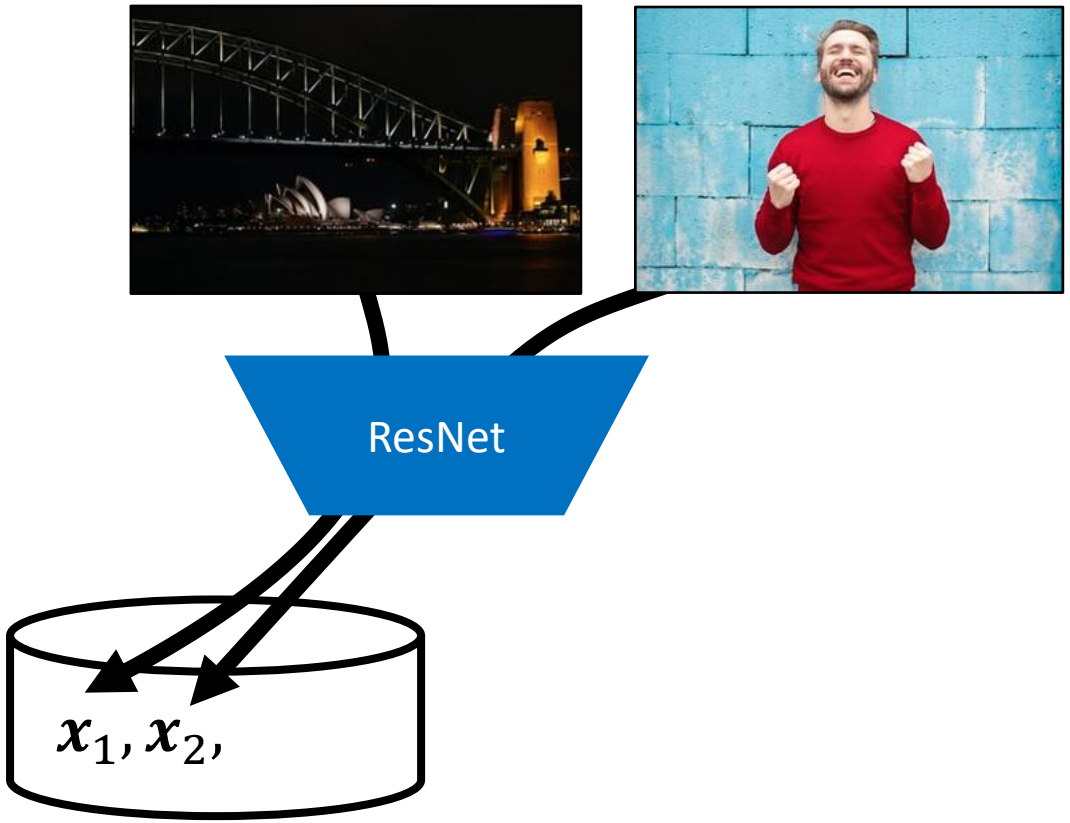


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Image Search

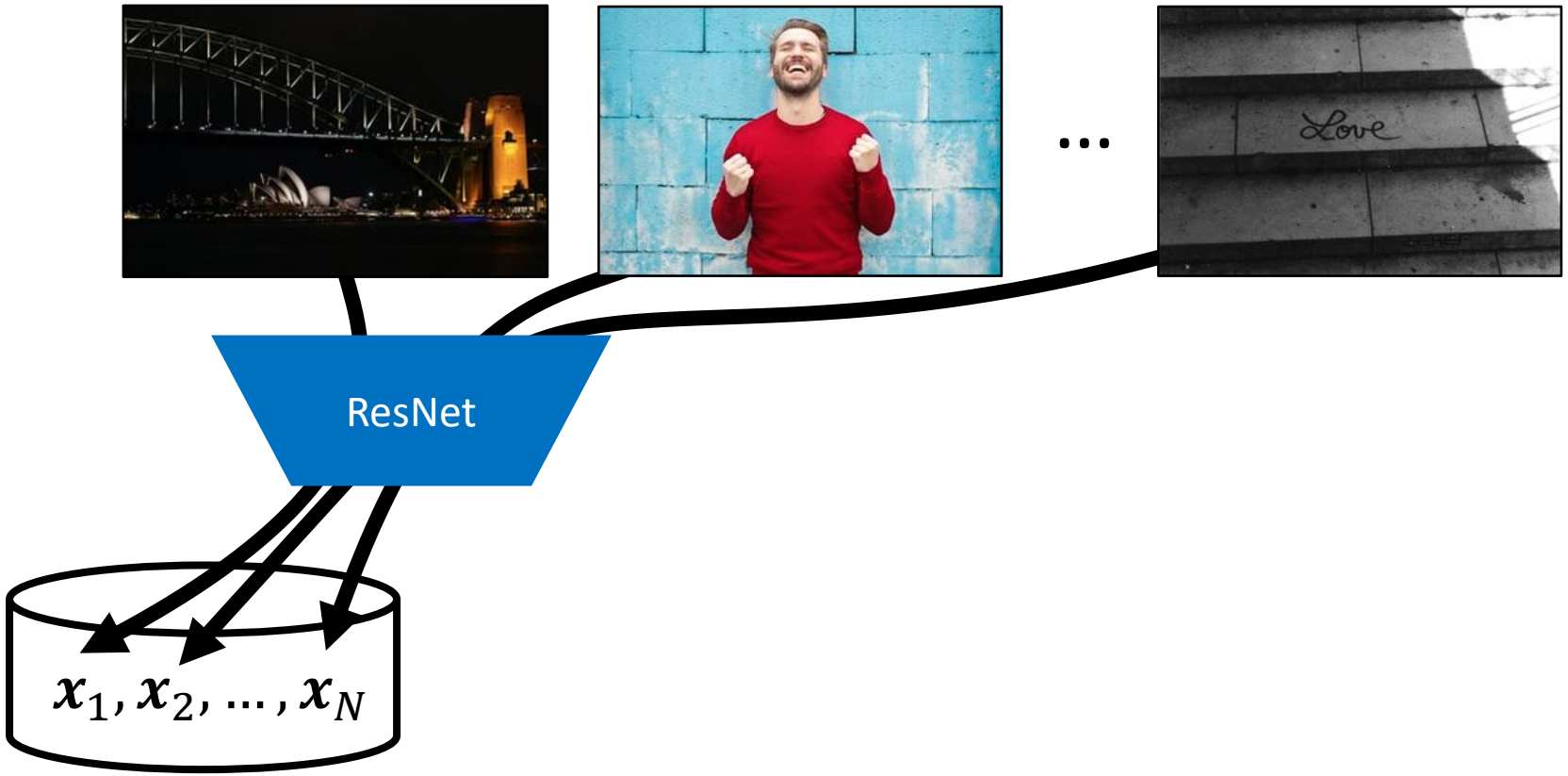


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Image Search

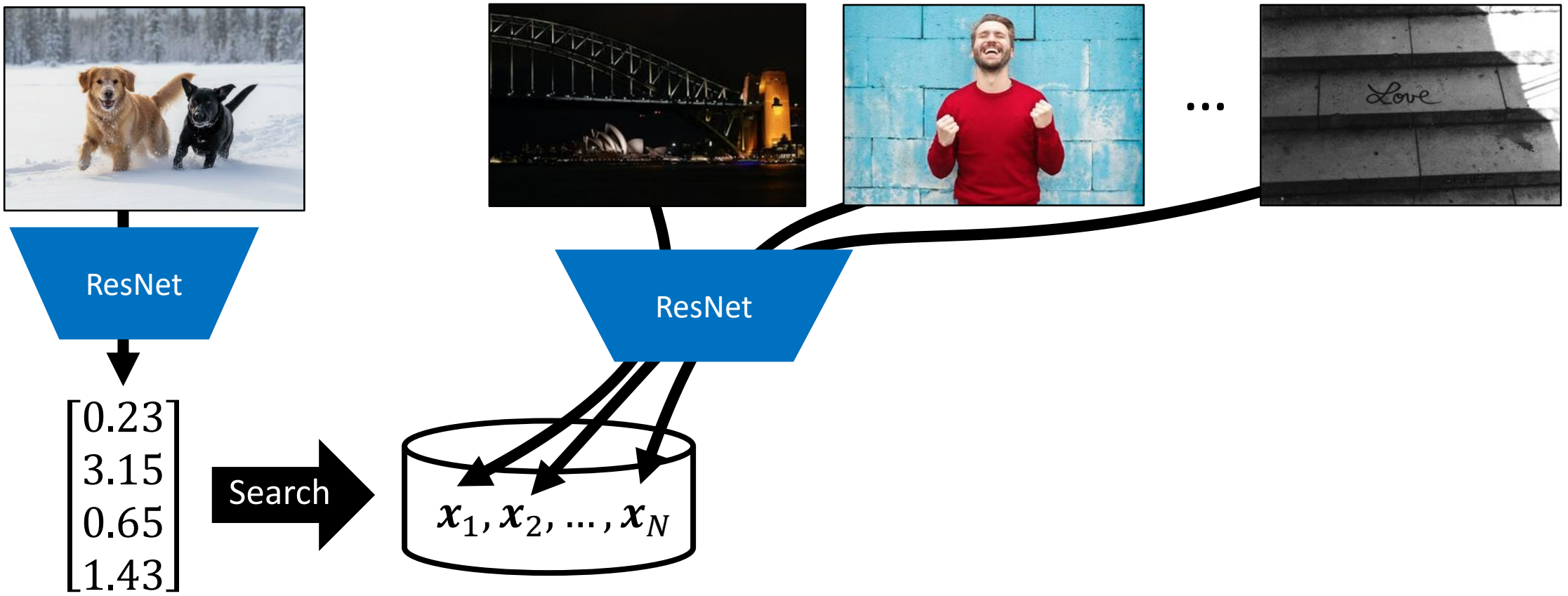
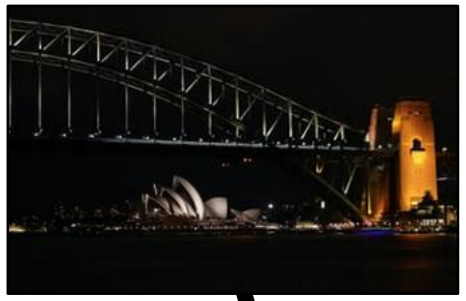


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

Image Search



...

ResNet

ResNet

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

Search

$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N$

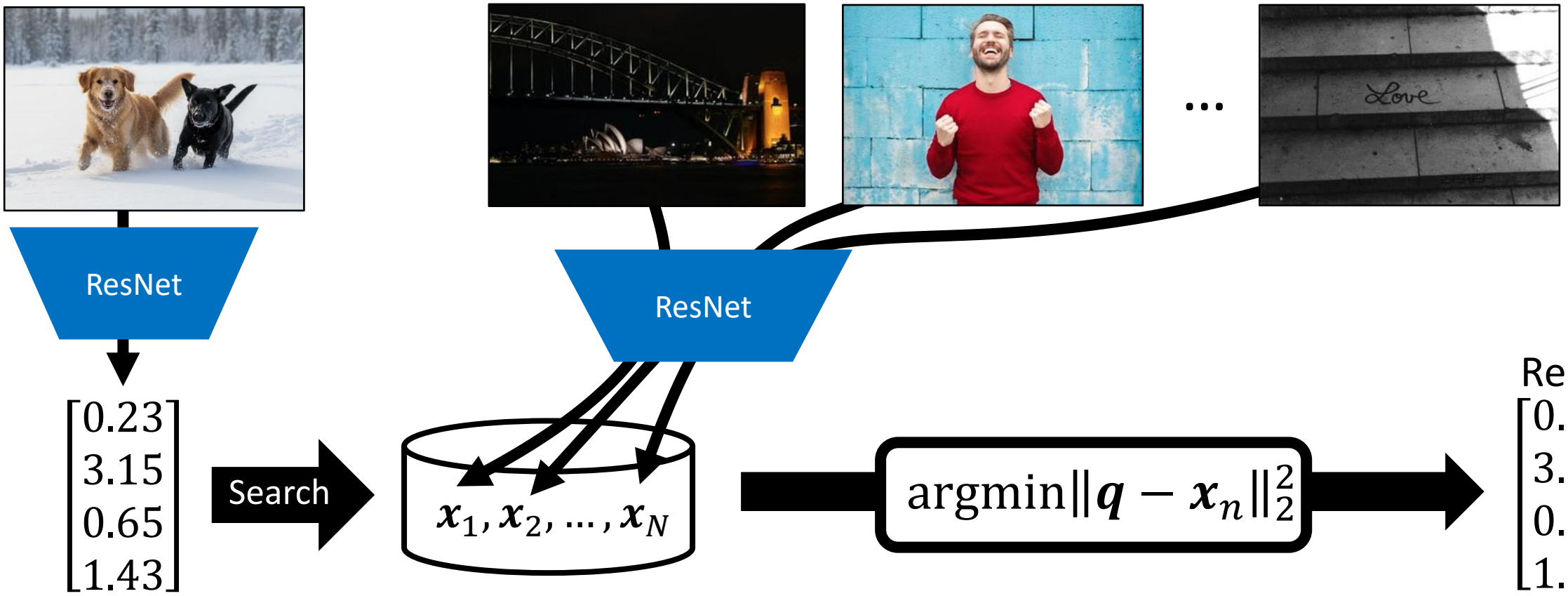
$$\text{argmin} \|\boldsymbol{q} - \boldsymbol{x}_n\|_2^2$$

Result

$$\begin{bmatrix} 0.20 \\ 3.25 \\ 0.72 \\ 1.68 \end{bmatrix}$$

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

Image Search



- Use a pre-trained CNN model (e.g., ResNet) as a feature extractor
- Represent an image as a high-dimensional vector
- Image-retrieval by nearest neighbor 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

What tasks has ANN been used for?

Computer Vision (CV):
SIFT + BoF

CV:
Image search

Natural Language Processing (NLP) &
Information Retrieval:
Text search (Dense? Sparse?)

Machine Learning:
kNN classification, metric learning

CV & NLP:
CLIP multimodal search

Large Language Models:
Retrieval Augmented Generation (RAG)

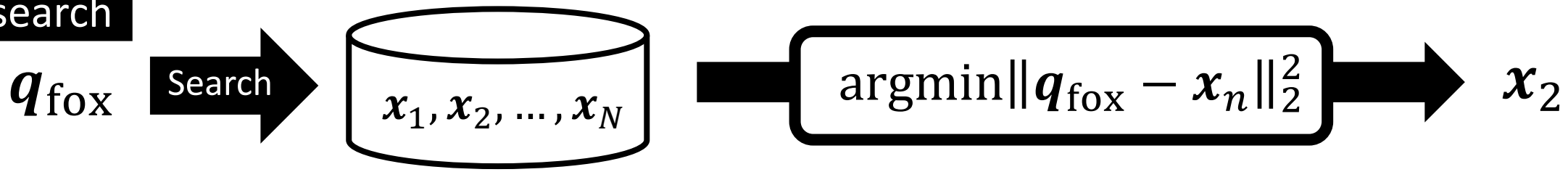
Database:
VectorDB

Text Search (Dense and/or Sparse)

Query: “fox” → search

Doc1: “The cat sleeps on chairs”
Doc2: “A quick blue fox runs”
Doc3: “The cat sits near window”
…

Dense search



- How to design embedding? BERT and its successors…

Sparse search

“fox” →

Term	ID
the	1, 3
fox	2
cat	2, 3
Blue	2
…	…

→ Doc2

- Classical “matching” and its extensions
- TF-IDF, BM25, SPLADE…

- Dense search: accurate
- Sparse search: fast
- Combining two approaches is a hot topic in information retrieval

What tasks has ANN been used for?

Computer Vision (CV):
SIFT + BoF

CV:
Image search

Natural Language Processing (NLP) &
Information Retrieval:
Text search (Dense? Sparse?)

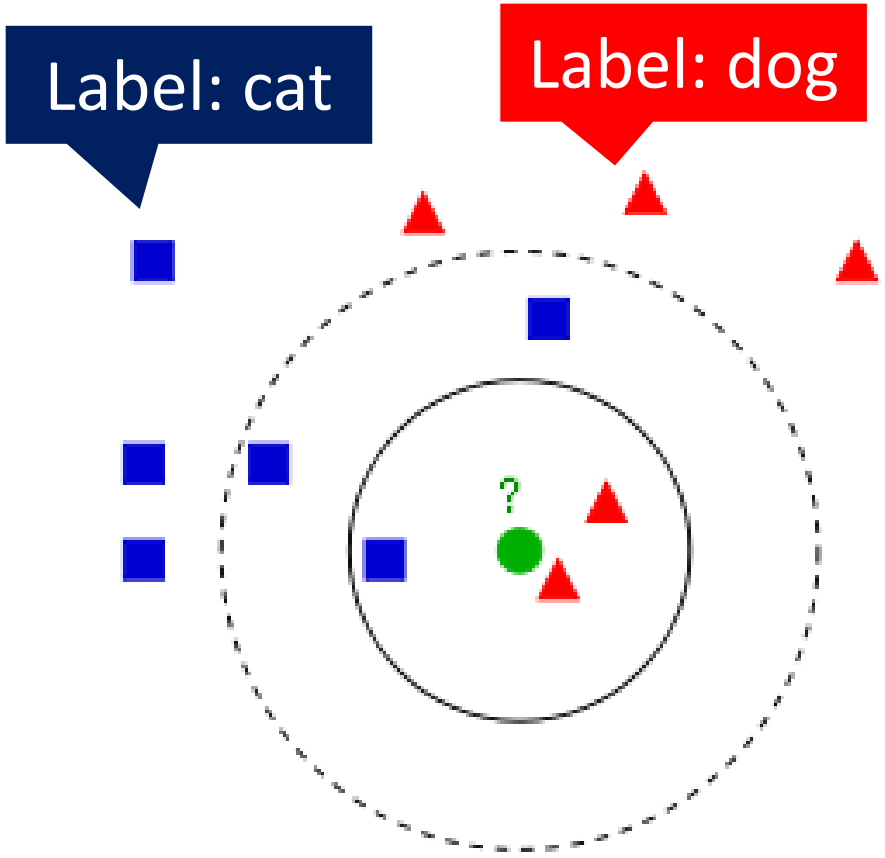
Machine Learning:
kNN classification, metric learning

CV & NLP:
CLIP multimodal search

Large Language Models:
Retrieval Augmented Generation (RAG)

Database:
VectorDB

kNN classification



https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

- The most straightforward approach for classification
- Given a query, find the closest sample from training data, and report its label
- Although super simple, it's actually effective if the embedding is good and #samples are large
- We can just run ANN search
- Complex ML -> Simple but large-scale search

What tasks has ANN been used for?

Computer Vision (CV):
SIFT + BoF

CV:
Image search

Natural Language Processing (NLP) &
Information Retrieval:
Text search (Dense? Sparse?)

Machine Learning:
kNN classification, metric learning

CV & NLP:
CLIP multimodal search

Large Language Models:
Retrieval Augmented Generation (RAG)

Database:
VectorDB

CLIP 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

CLIP multimodal search

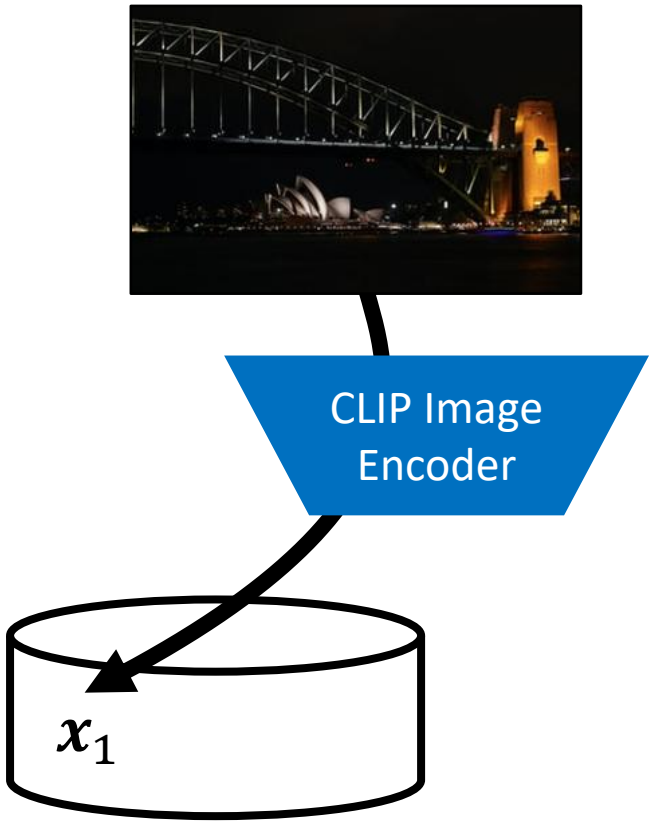


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

CLIP multimodal search

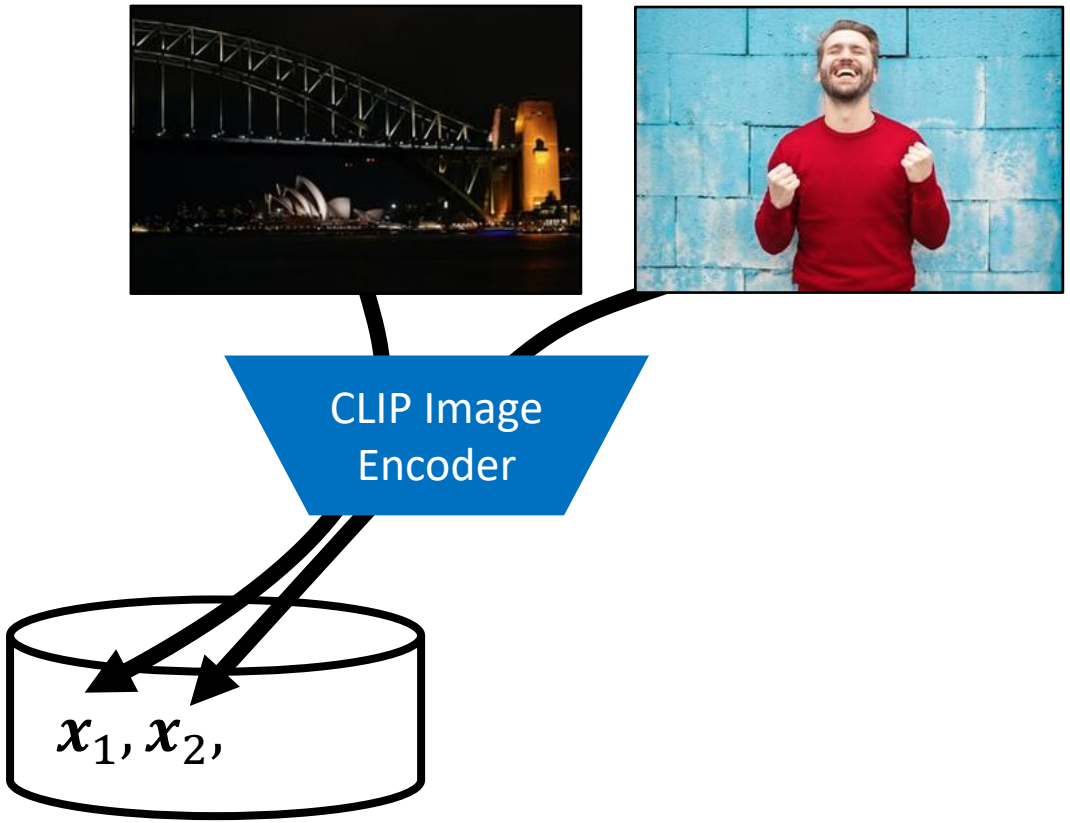


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

CLIP multimodal search

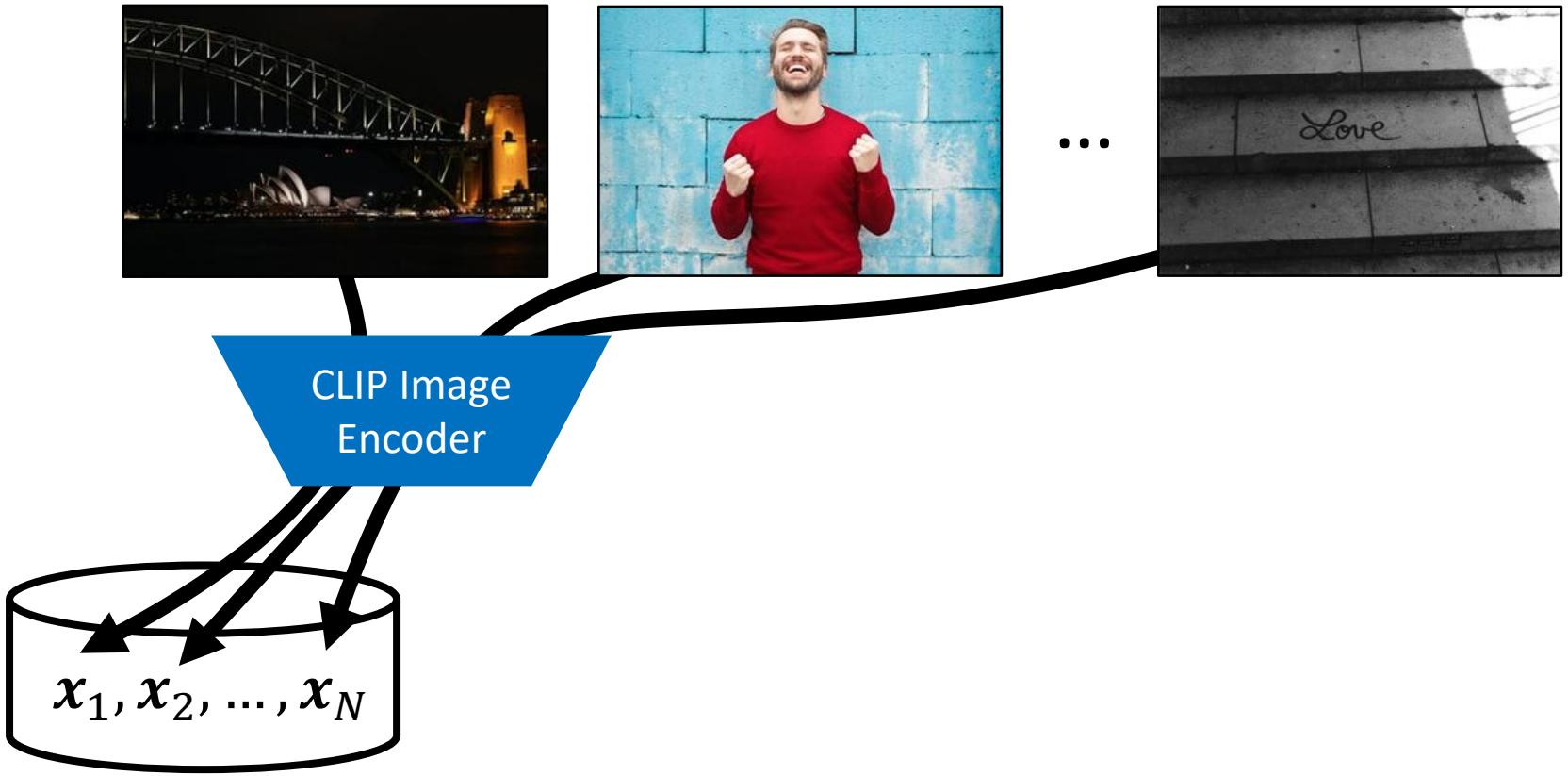


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

CLIP multimodal search

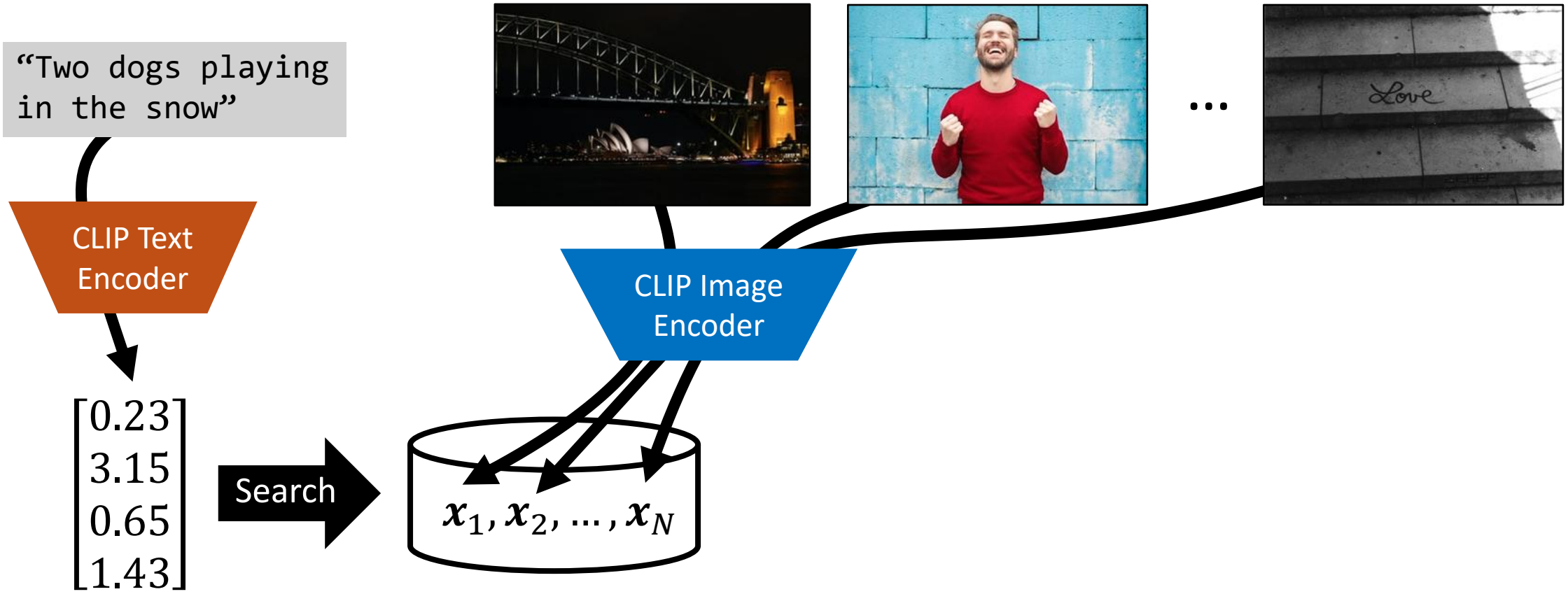


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

CLIP multimodal search

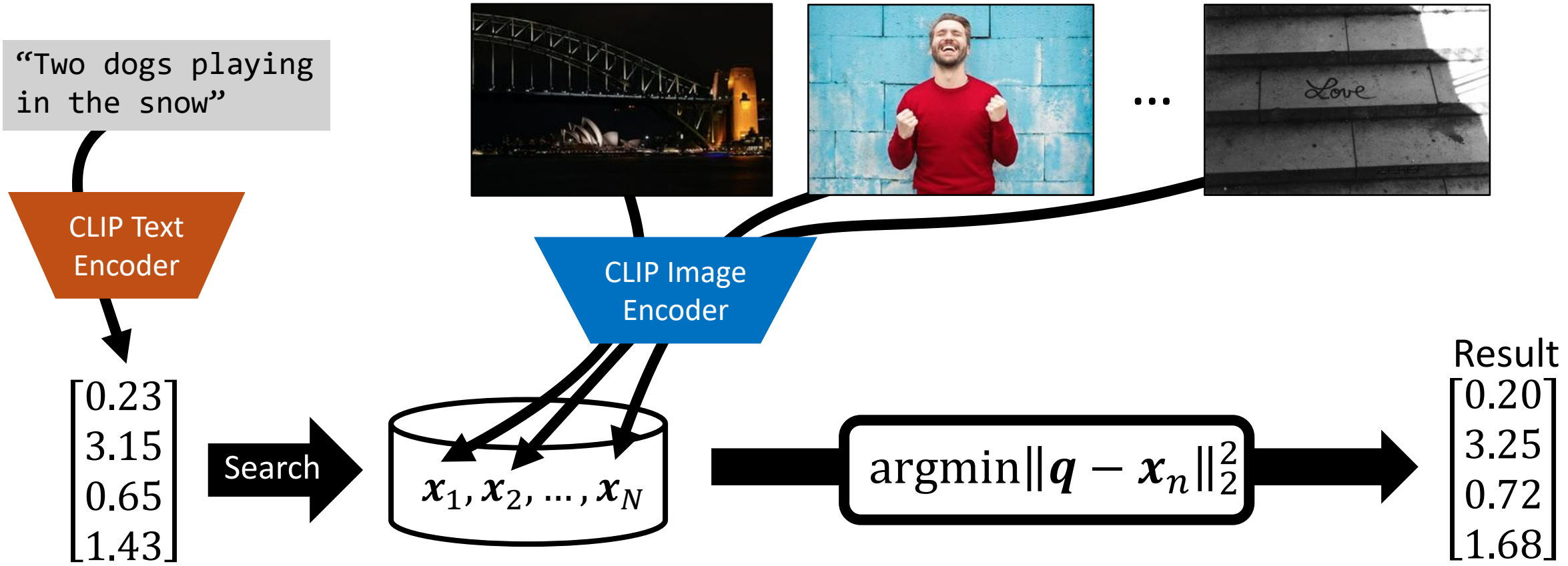
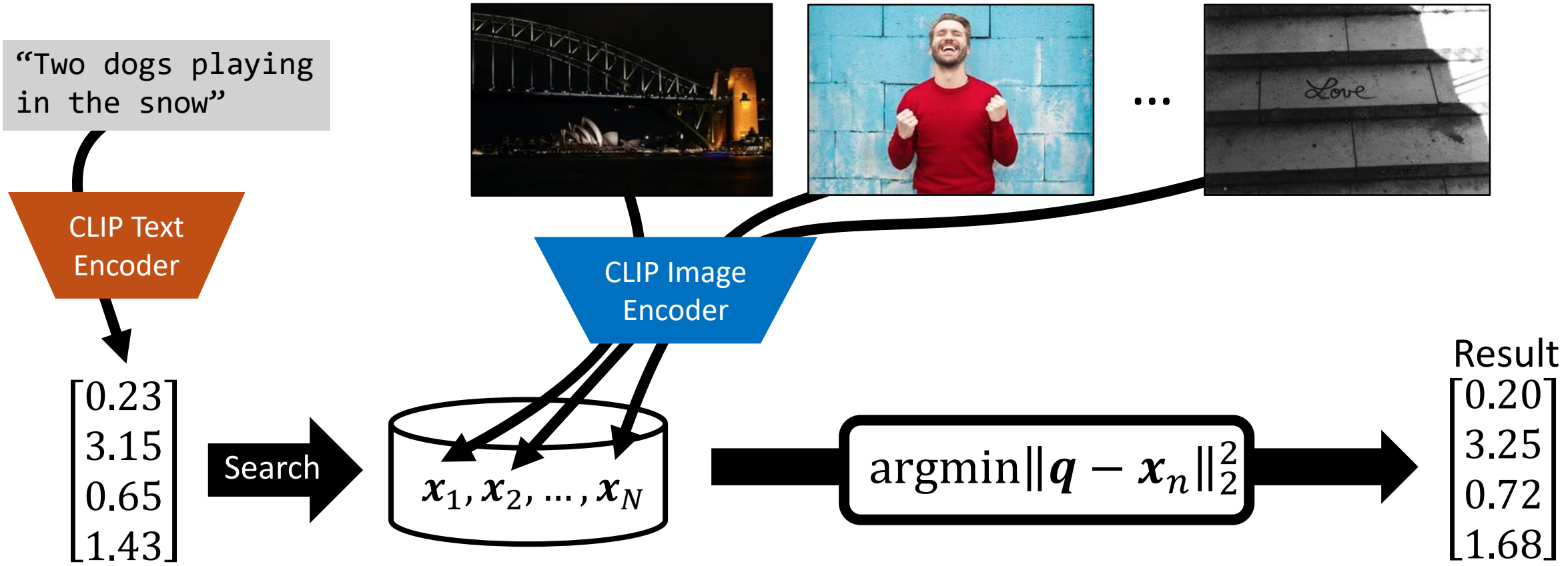


Image are from: https://github.com/haltakov/natural-language-image-search
Credit: Photos by Genton Damian, bruce mars, Dalal Nizam, and Richard Burlton on Unsplash

CLIP multimodal search



- CLIP enables us to compare images and texts
- 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

What tasks has ANN been used for?

Computer Vision (CV):
SIFT + BoF

CV:
Image search

Natural Language Processing (NLP) &
Information Retrieval:
Text search (Dense? Sparse?)

Machine Learning:
kNN classification, metric learning

CV & NLP:
CLIP multimodal search

Large Language Models:
Retrieval Augmented Generation (RAG)

Database:
VectorDB

RAG: LLM + embedding

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

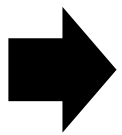
ChatGPT 3.5
(trained in 2021)

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

RAG: LLM + embedding

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

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

RAG: LLM + embedding

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

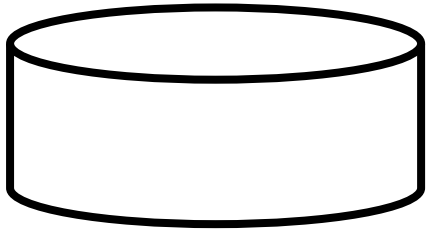
ChatGPT 3.5
(trained in 2021)

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

RAG: 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

RAG: LLM + embedding

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

ChatGPT 3.5
(trained in 2021)

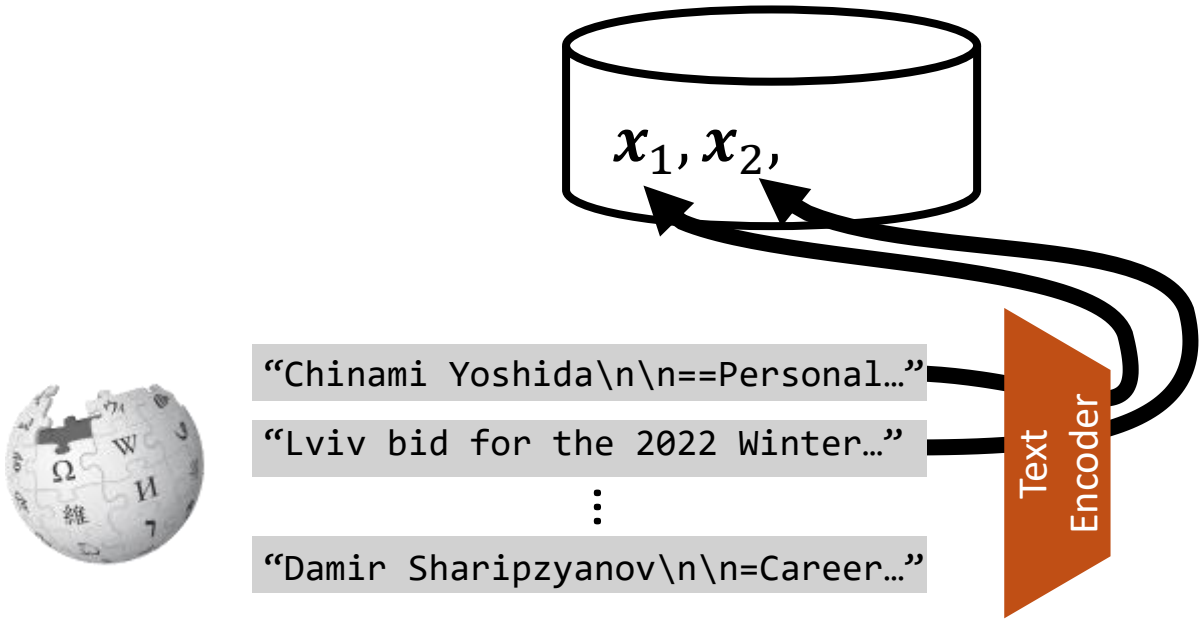
Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb

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

RAG: LLM + embedding

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

ChatGPT 3.5
(trained in 2021)



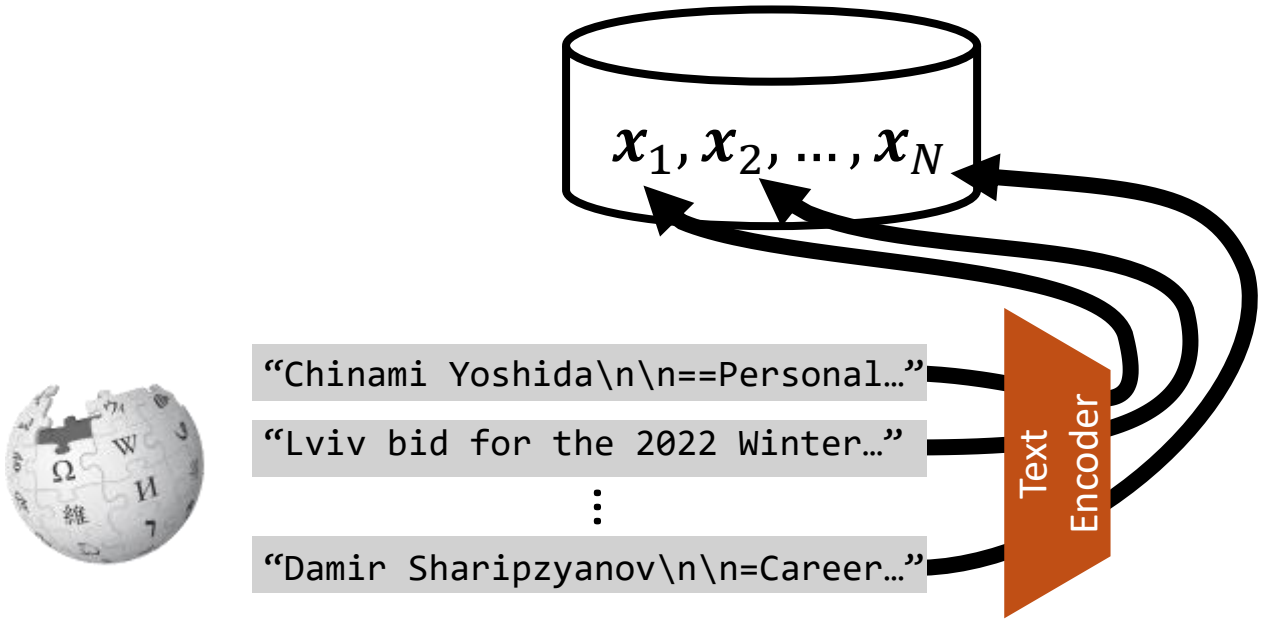
Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb

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

RAG: 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

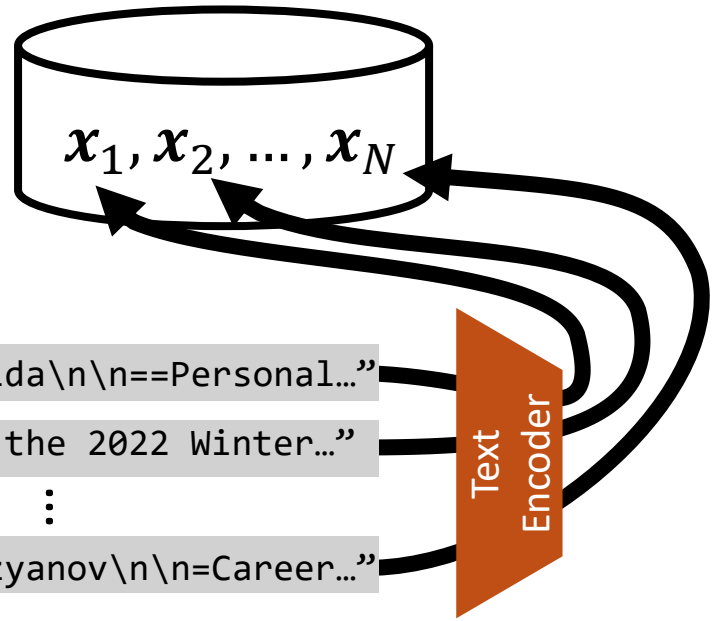
Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

RAG: LLM + embedding

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

Text Encoder

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

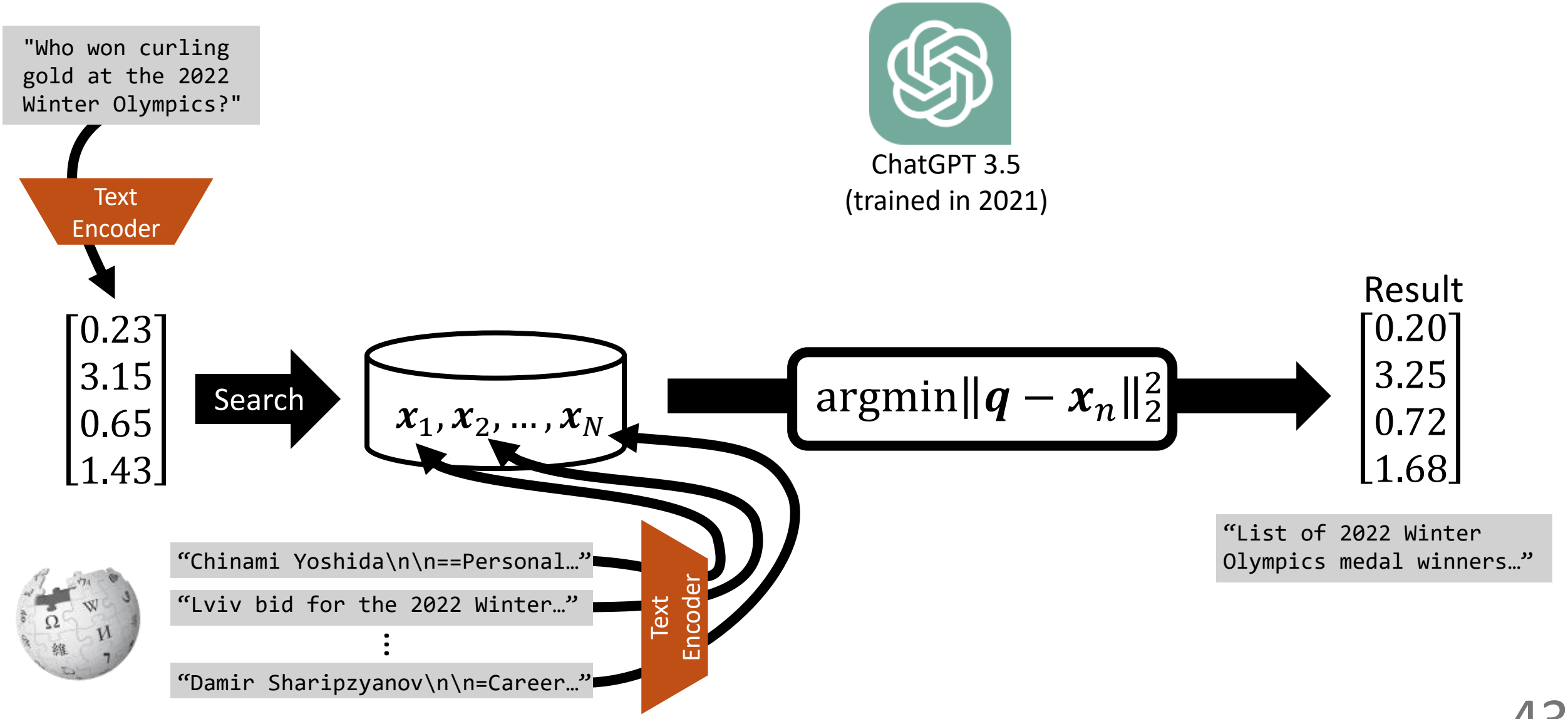


ChatGPT 3.5
(trained in 2021)

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

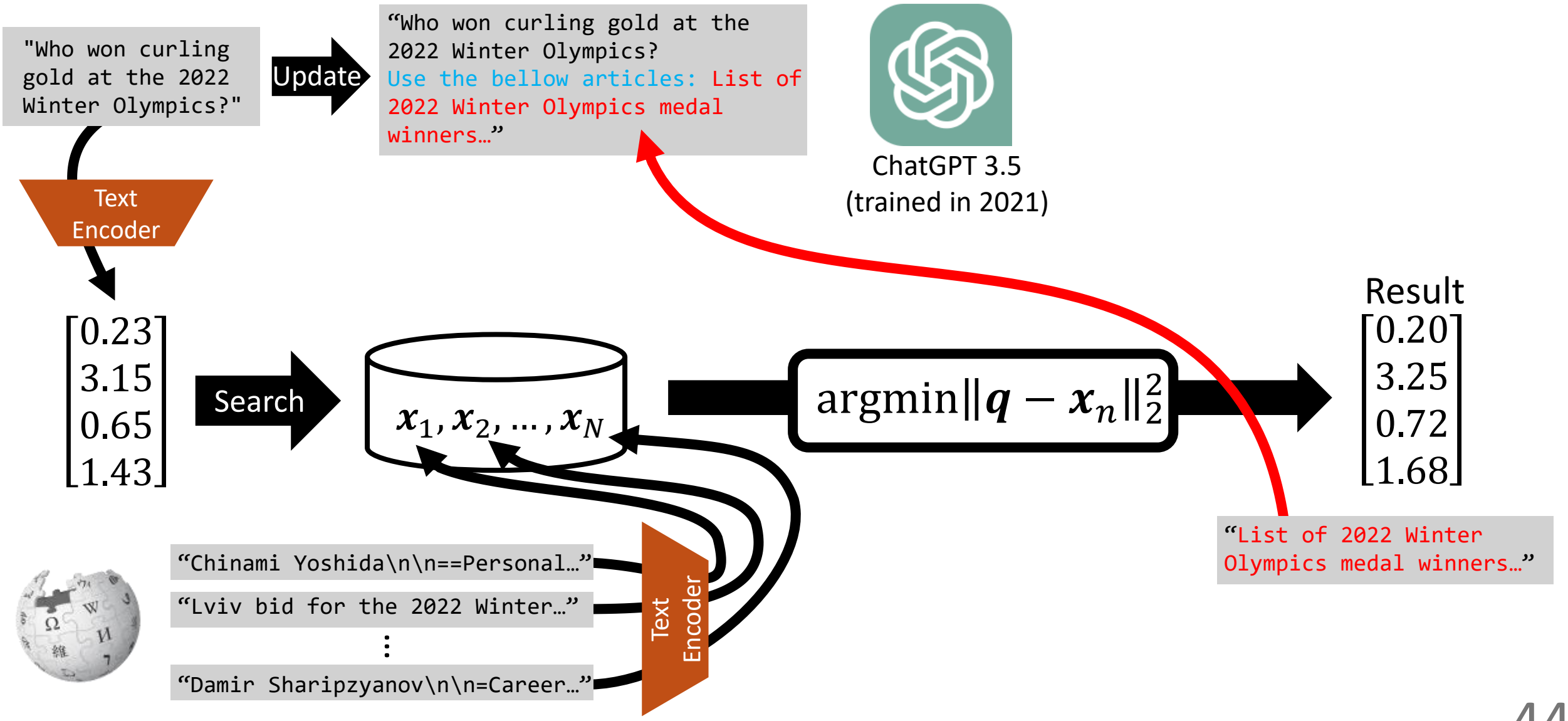
Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

RAG: LLM + embedding



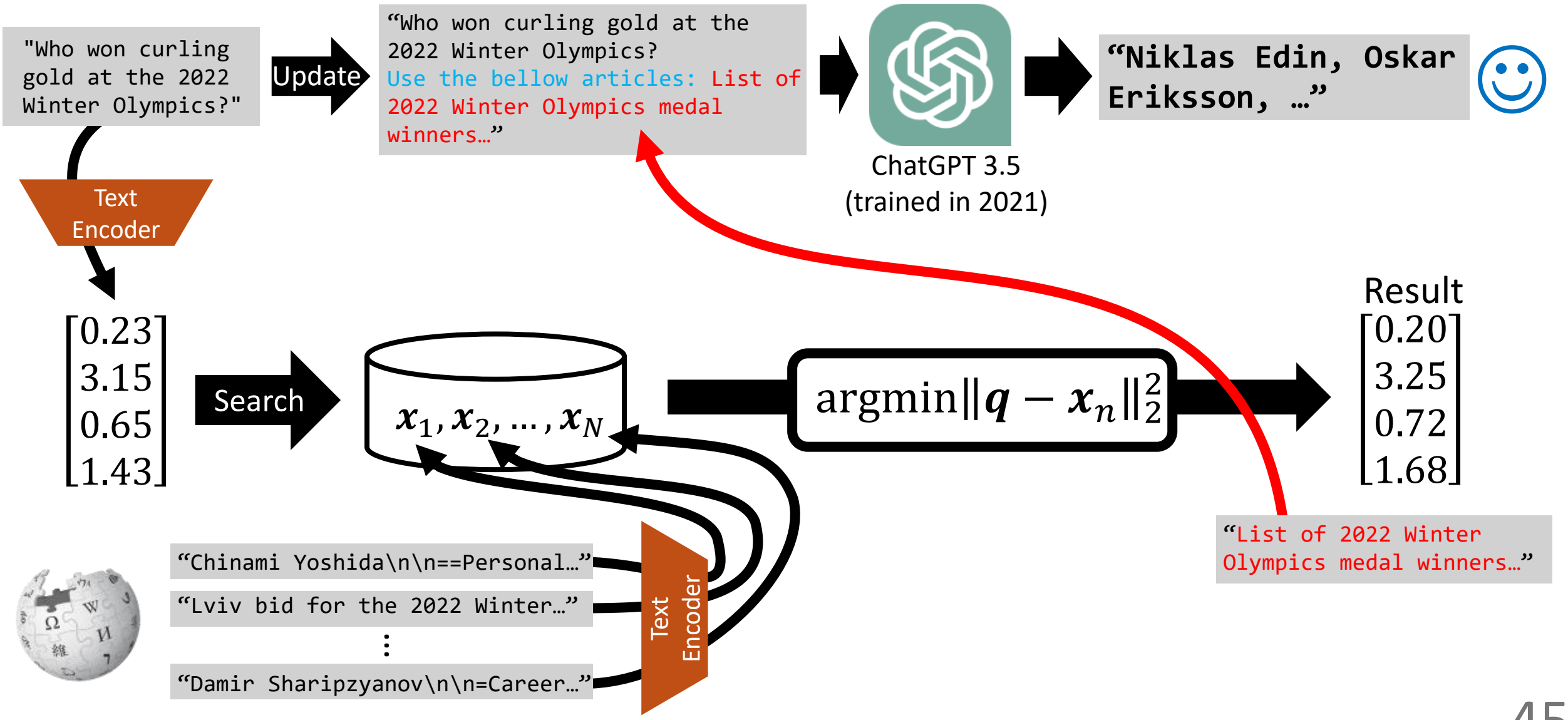
Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

RAG: LLM + embedding



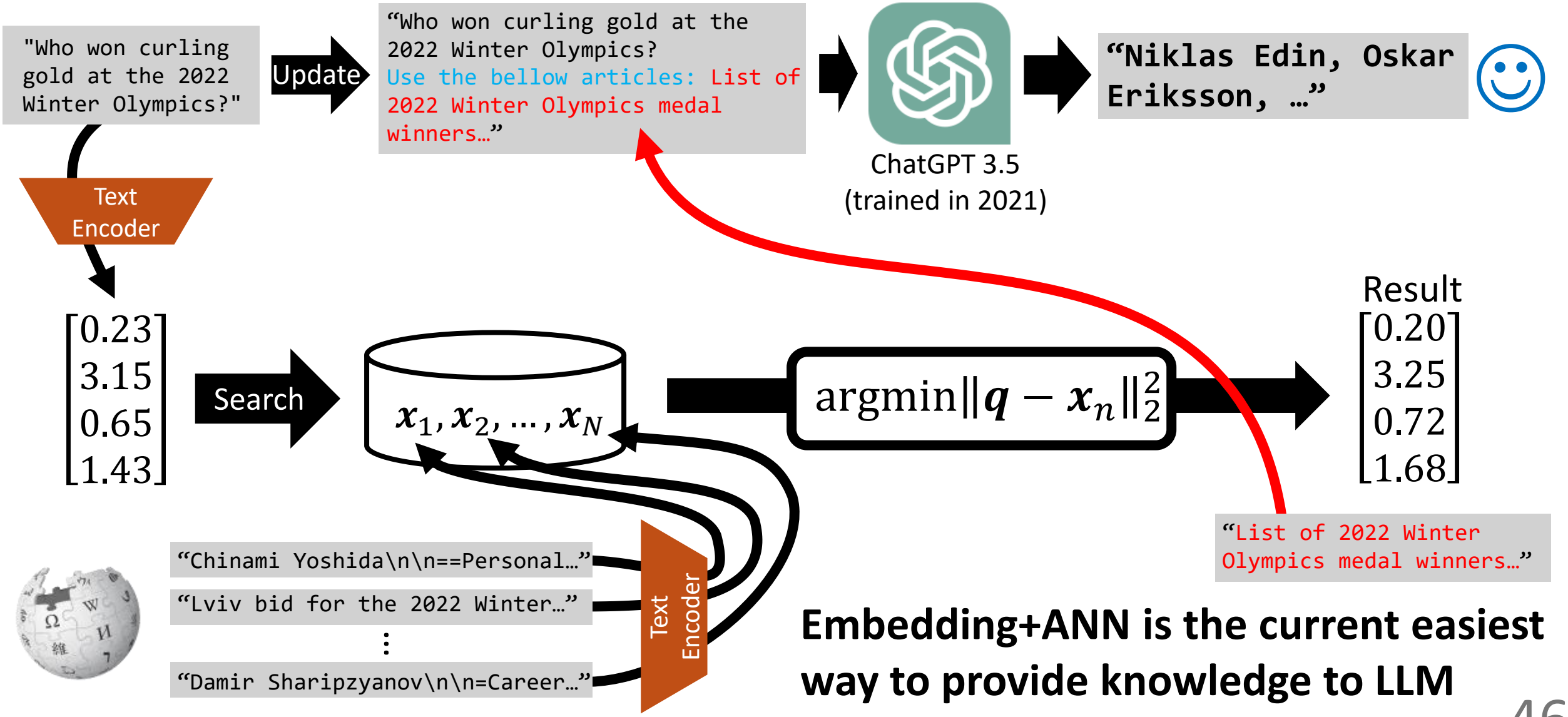
Texts are from: https://github.com/openai/openaicookbook/blob/main/examples/Question_answering_using_embeddings.ipynb Icon credit: https://ja.wikipedia.org/wiki/ChatGPT

RAG: 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

RAG: 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

What tasks has ANN been used for?

Computer Vision (CV):
SIFT + BoF

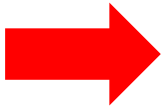
CV:
Image search

Natural Language Processing (NLP) &
Information Retrieval:
Text search (Dense? Sparse?)

Machine Learning:
kNN classification, metric learning

CV & NLP:
CLIP multimodal search

Large Language Models:
Retrieval Augmented Generation (RAG)



Database:
VectorDB

What is Vector DB?

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

What is Vector DB?

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

Pinecone

Qdrant

One library may implement multiple algorithms

☹ "I benchmarked faiss"

☺ "I benchmarked PQ in faiss"

What is Vector DB?

One algorithm may be implemented in multiple libraries

- Often, by re

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

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

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

- 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

What is Vector DB?

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

ScaNN

Pinecone

Qdrant

Vald

Vertex AI Matching Engine

Weaviate

Often, one library = one algorithm

What is Vector DB?

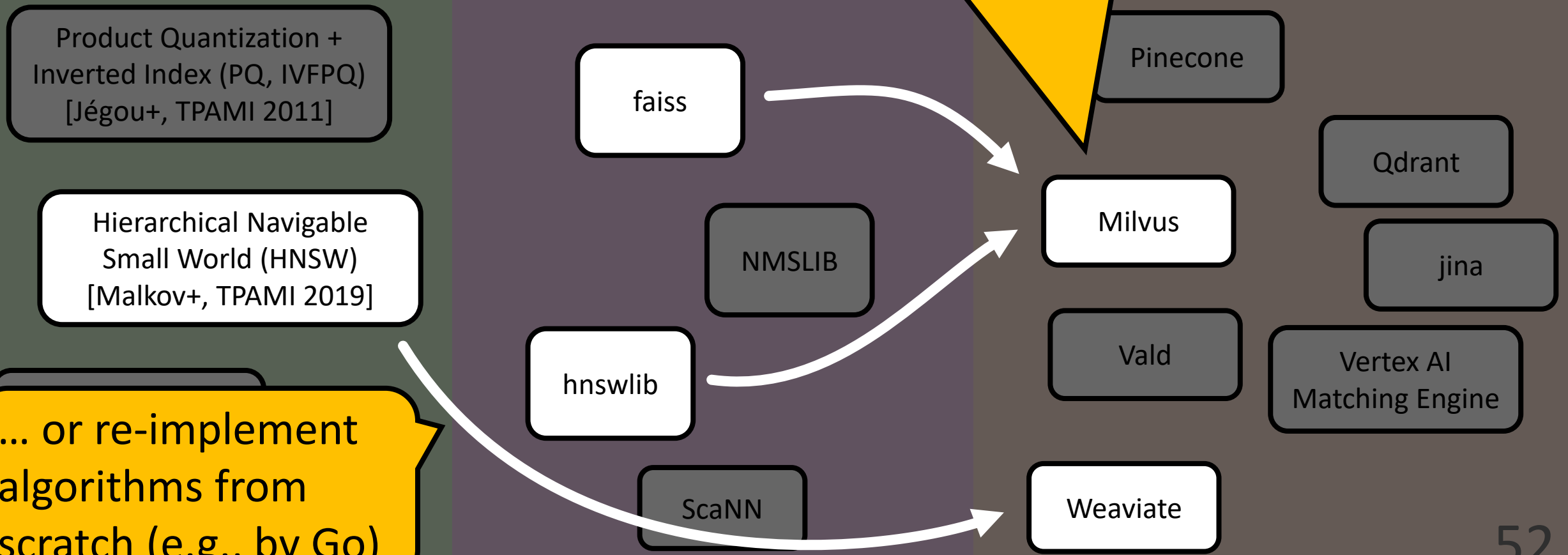
One service may use some libraries

Algorithm

- Scientific paper
- Math
- Often, by researchers

, vector DB)

- ndling metadata,
- rving, scaling, IO, CRUD, etc)
- By researchers, developers, e… sually, by companies



What is Vector DB?

Algorithm
- Scientific paper
- Math
- Often, by researchers

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

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

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

faiss

NMSLIB

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

aviate

- Recently, the DB community has been the most active
- The CV community has settled down a bit 😭

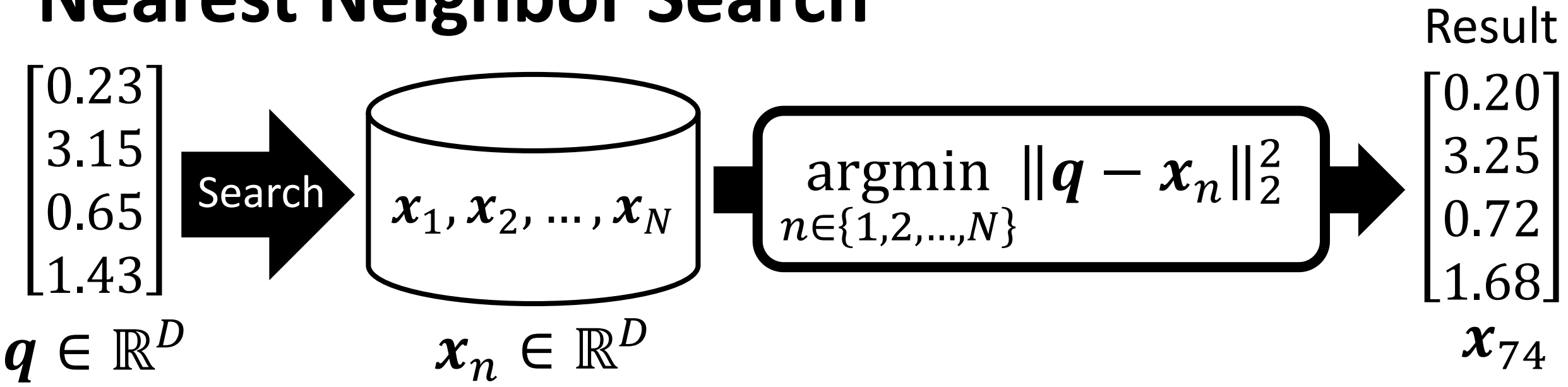
Outline

1. History from an applications perspective

2. **Importance of implementation: nearest neighbor search in faiss**

3. Basics of modern baseline: graph-based search

Nearest Neighbor Search



- Before trying ANN, we should try NN
- Introduce a naïve implementation
- Introduce a fast implementation: **Faiss** library
- Experience the drastic difference between the two implementations

M D-dim query vectors $\mathcal{Q} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \dots, \boldsymbol{q}_M\}$

N D-dim database vectors $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$ $M \ll N$

Task：Given $\boldsymbol{q} \in \mathcal{Q}$ and $\boldsymbol{x} \in \mathcal{X}$, compute $\|\boldsymbol{q} - \boldsymbol{x}\|_2^2$

M D-dim query vectors $\mathcal{Q} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_M\}$

N D-dim database vectors $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ $M \ll N$

Task：Given $\boldsymbol{q} \in \mathcal{Q}$ and $\boldsymbol{x} \in \mathcal{X}$, compute $\|\boldsymbol{q} - \boldsymbol{x}\|_2^2$

Naïve impl.

```
def l2sqr(q, x):
  diff = 0.0
  for (d = 0; d < D; ++d):
    diff += (q[d] - x[d])**2
  return diff
```

```
parfor q in Q:
  for x in X:
    l2sqr(q, x)
```

Parallelize query-side

Select min by heap, but omit it now

M D-dim query vectors $\mathcal{Q} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_M\}$

N D-dim database vectors $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$ $M \ll N$

Task：Given $\boldsymbol{q} \in \mathcal{Q}$ and $\boldsymbol{x} \in \mathcal{X}$, compute $\|\boldsymbol{q} - \boldsymbol{x}\|_2^2$

Naïve impl.

```
def l2sqr(q, x):
  diff = 0.0
  for (d = 0; d < D; ++d):
    diff += (q[d] - x[d])**2
  return diff
```

```
parfor q in Q:
  for x in X:
    l2sqr(q, x)
```

Parallelize query-side

Select min by heap, but omit it now

faiss impl.

Disclaimer: the code used in the explanation is from several years ago and is not up to date.

if $M < 20$：

compute $\|\boldsymbol{q} - \boldsymbol{x}\|_2^2$ by SIMD

else：

compute $\|\boldsymbol{q} - \boldsymbol{x}\|_2^2 = \|\boldsymbol{q}\|_2^2 - 2\boldsymbol{q}^\top\boldsymbol{x} + \|\boldsymbol{x}\|_2^2$ by BLAS

M D-dim query vectors $\mathcal{Q} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \dots, \boldsymbol{q}_M\}$

N D-dim database vectors $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$ $M \ll N$

Task：Given $\boldsymbol{q} \in \mathcal{Q}$ and $\boldsymbol{x} \in \mathcal{X}$, compute $\|\boldsymbol{q} - \boldsymbol{x}\|_2^2$

Naïve impl.

```
def l2sqr(q, x):
  diff = 0.0
  for (d = 0; d < D; ++d):
    diff += (q[d] - x[d])**2
  return diff
```

```
parfor q in Q:
  for x in X:
    l2sqr(q, x)
```

Parallelize query-side

Select min by heap, but omit it now

faiss impl.

Disclaimer: the code used in the explanation is from several years ago and is not up to date.

if $M < 20$:

compute $\|\boldsymbol{q} - \boldsymbol{x}\|_2^2$ by SIMD

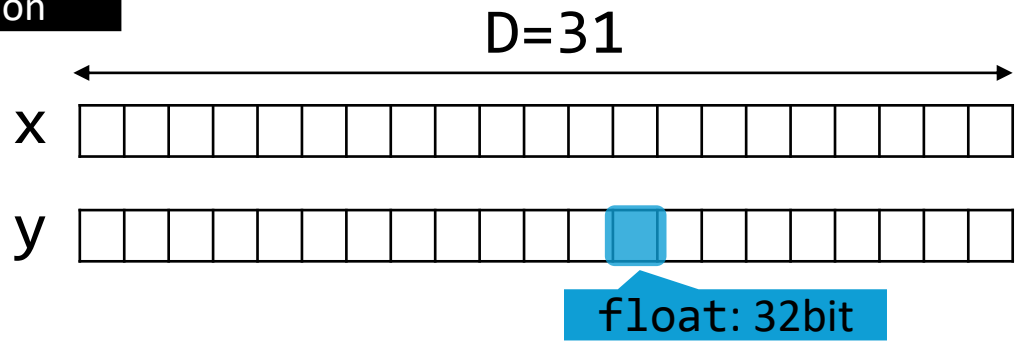
else :

compute $\|\boldsymbol{q} - \boldsymbol{x}\|_2^2 = \|\boldsymbol{q}\|_2^2 - 2\boldsymbol{q}^\top\boldsymbol{x} + \|\boldsymbol{x}\|_2^2$ by BLAS

$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```



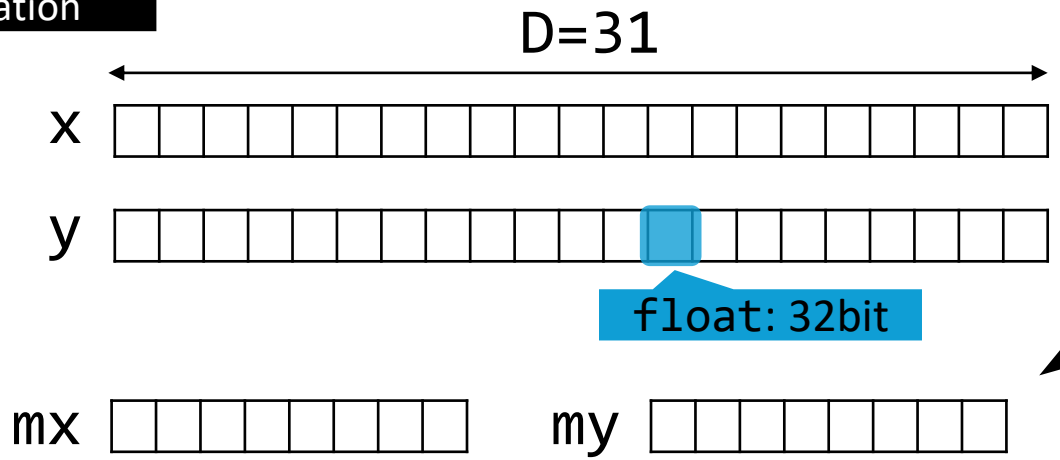
Ref.

```
def l2sqr(x, y):
  diff = 0.0
  for (d = 0; d < D; ++d):
    diff += (x[d] - y[d])**2
  return diff
```

$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```



- 256bit SIMD Register
- Process eight floats at once

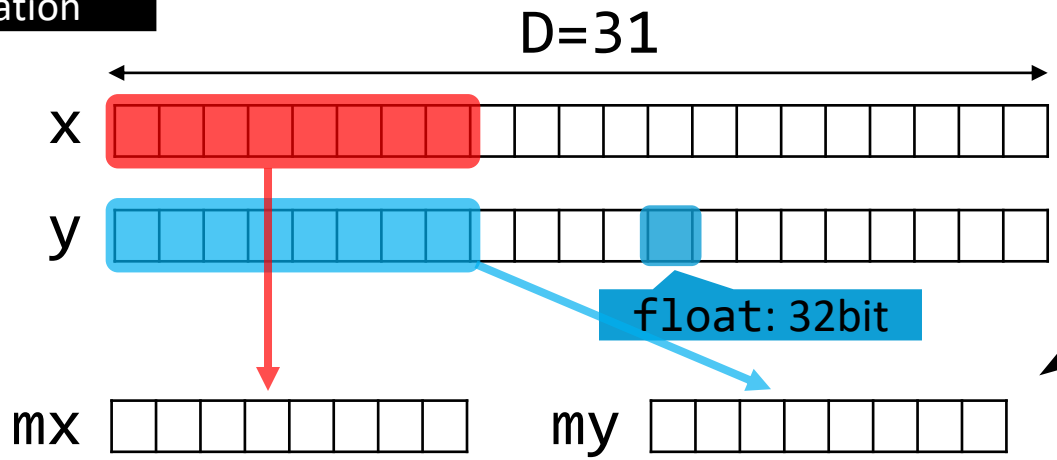
Ref.

```
def l2sqr(x, y):
    diff = 0.0
    for (d = 0; d < D; ++d):
        diff += (x[d] - y[d])**2
    return diff
```

$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```



- 256bit SIMD Register
- Process eight floats at once

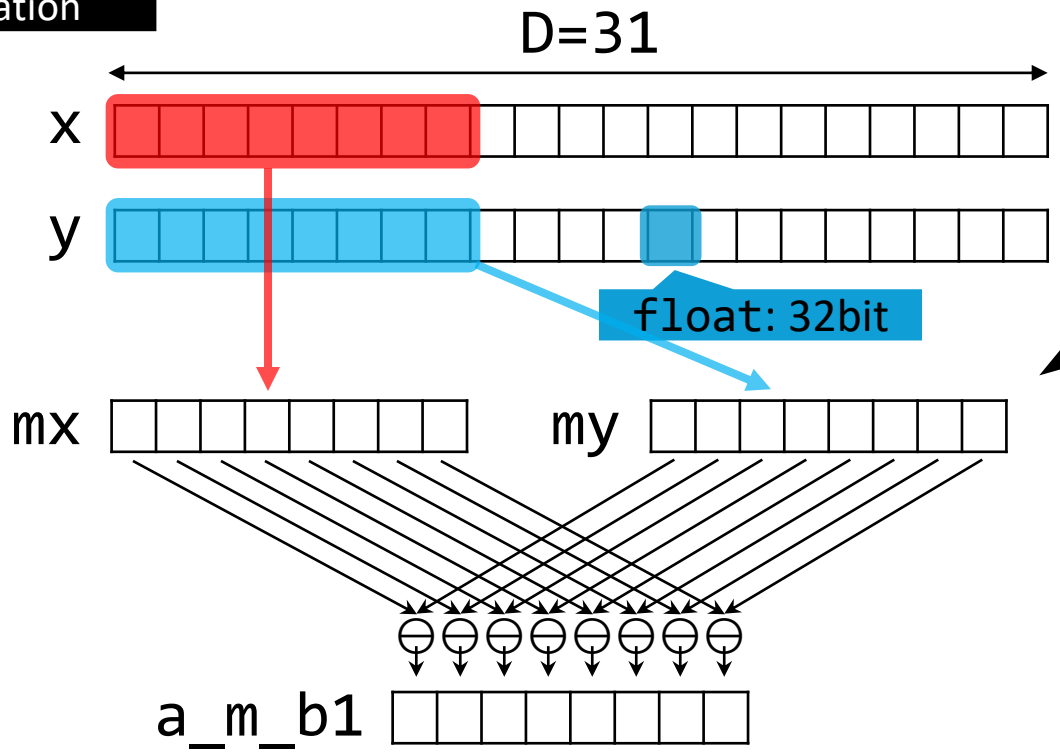
Ref.

```
def l2sqr(x, y):
    diff = 0.0
    for (d = 0; d < D; ++d):
        diff += (x[d] - y[d])**2
    return diff
```

$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```



- 256bit SIMD Register
- Process eight floats at once

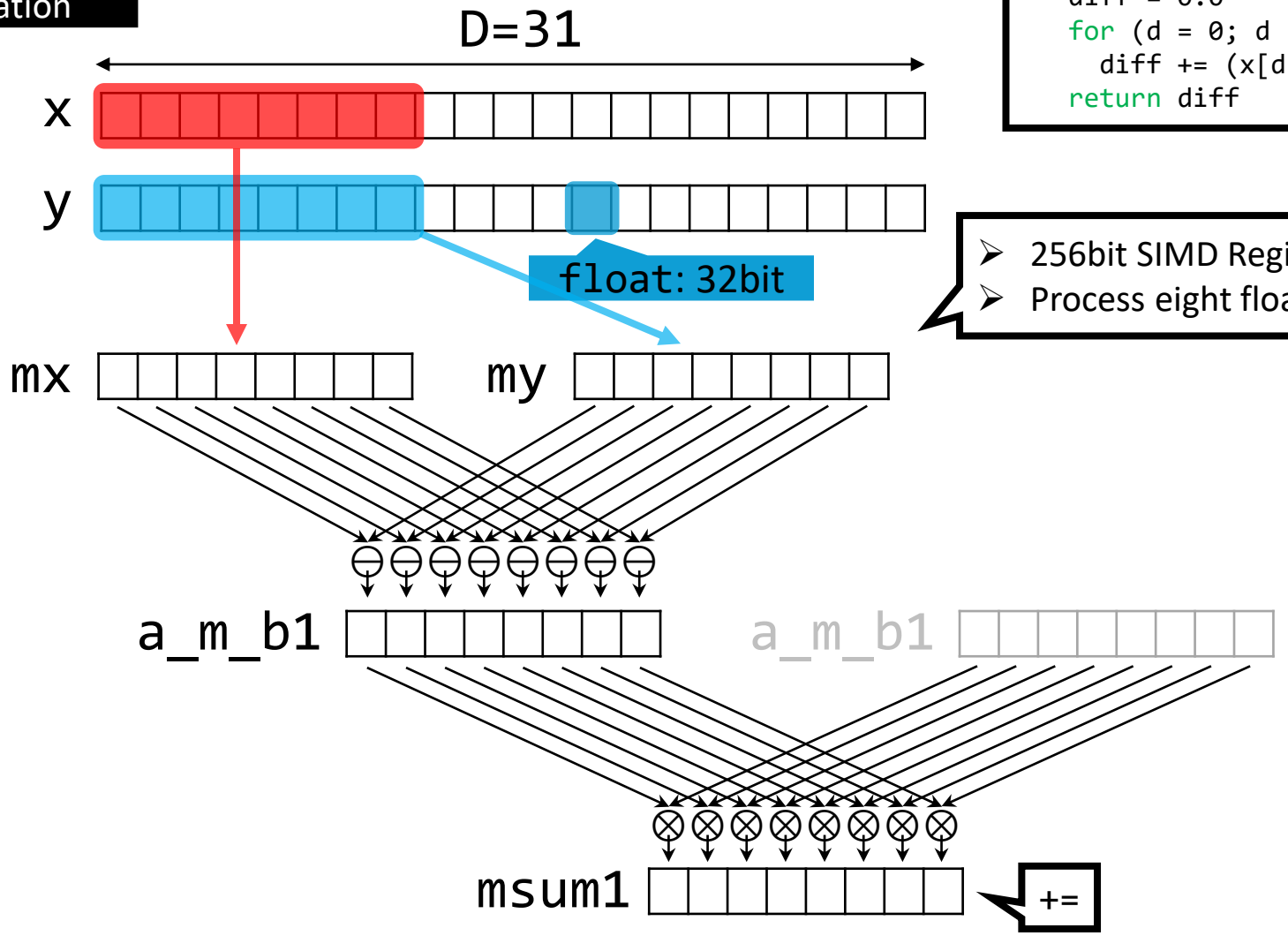
Ref.

```
def l2sqr(x, y):
    diff = 0.0
    for (d = 0; d < D; ++d):
        diff += (x[d] - y[d])**2
    return diff
```

$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```



- 256bit SIMD Register
- Process eight floats at once

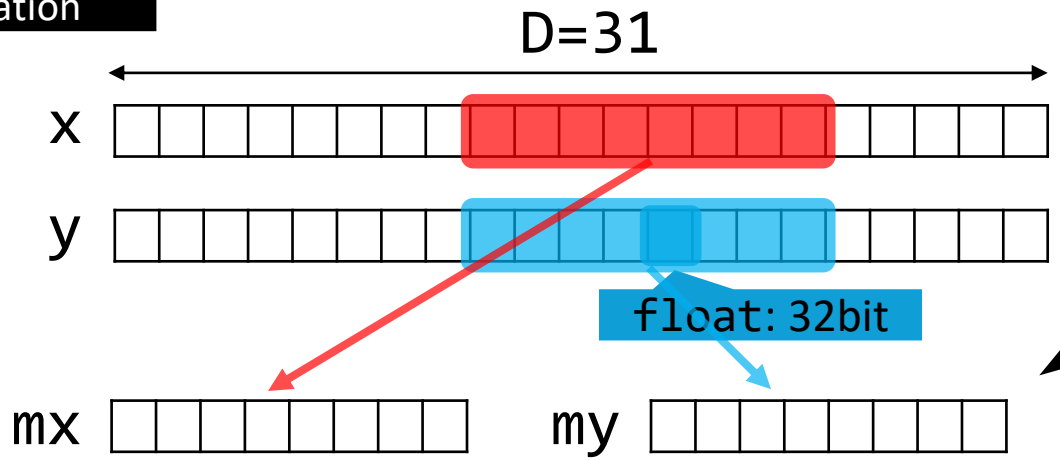
Ref.

```
def l2sqr(x, y):
    diff = 0.0
    for (d = 0; d < D; ++d):
        diff += (x[d] - y[d])**2
    return diff
```

$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

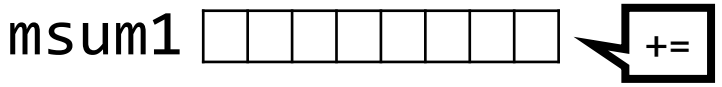
```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```



- 256bit SIMD Register
- Process eight floats at once

Ref.

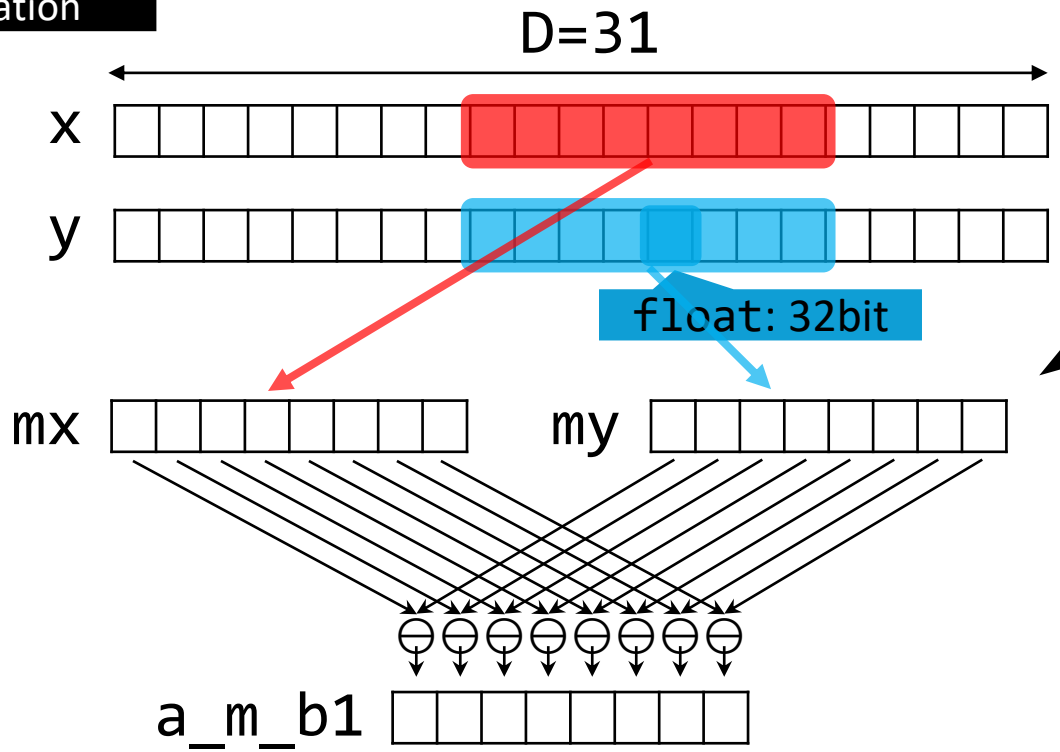
```
def l2sqr(x, y):
    diff = 0.0
    for (d = 0; d < D; ++d):
        diff += (x[d] - y[d])**2
    return diff
```



$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

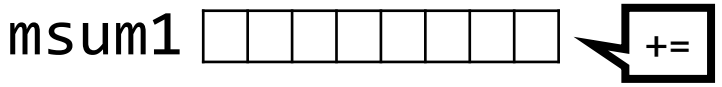
```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```



- 256bit SIMD Register
- Process eight floats at once

Ref.

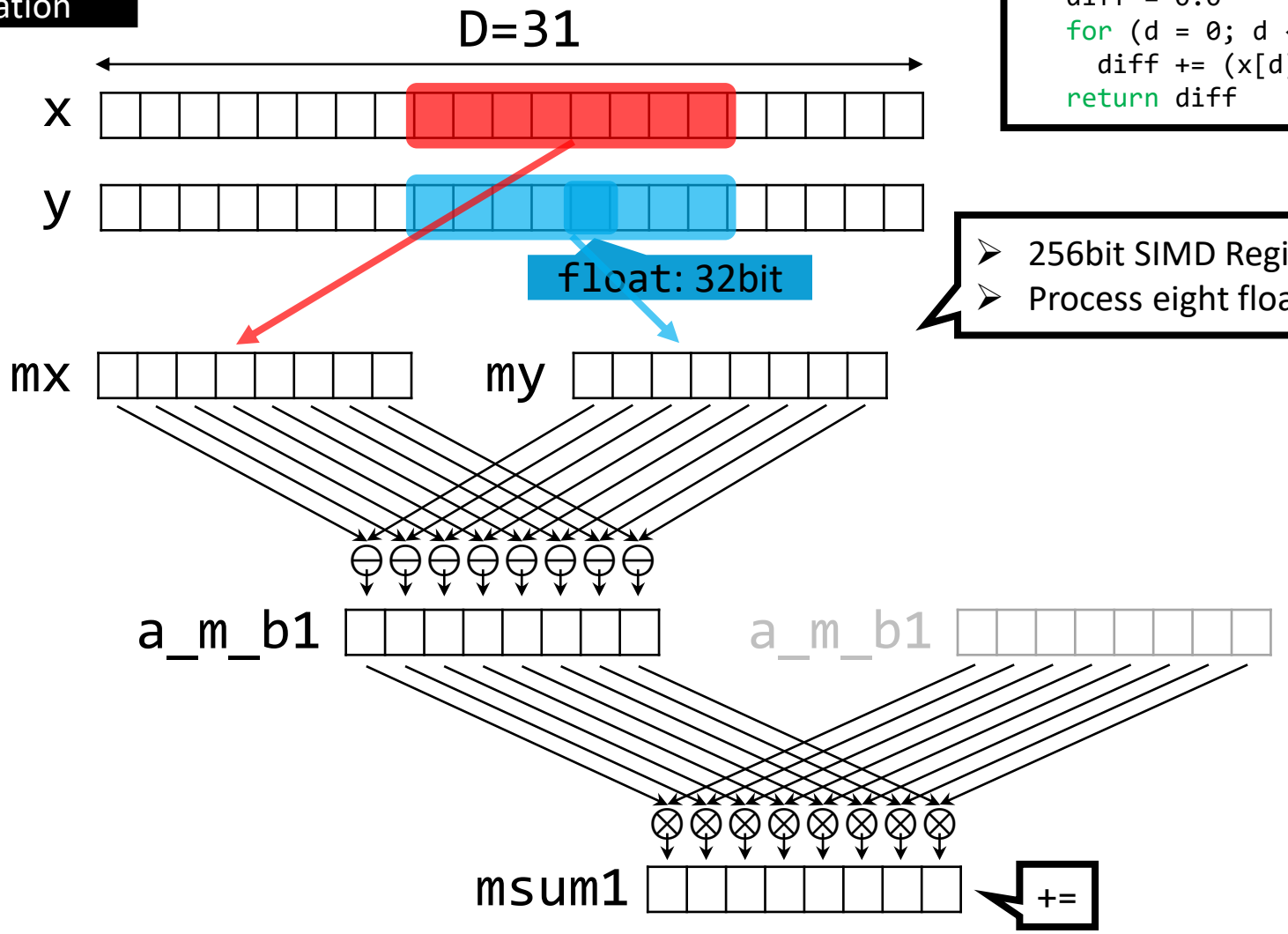
```
def l2sqr(x, y):
    diff = 0.0
    for (d = 0; d < D; ++d):
        diff += (x[d] - y[d])**2
    return diff
```



$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```



- 256bit SIMD Register
- Process eight floats at once

Ref.

```
def l2sqr(x, y):
    diff = 0.0
    for (d = 0; d < D; ++d):
        diff += (x[d] - y[d])**2
    return diff
```

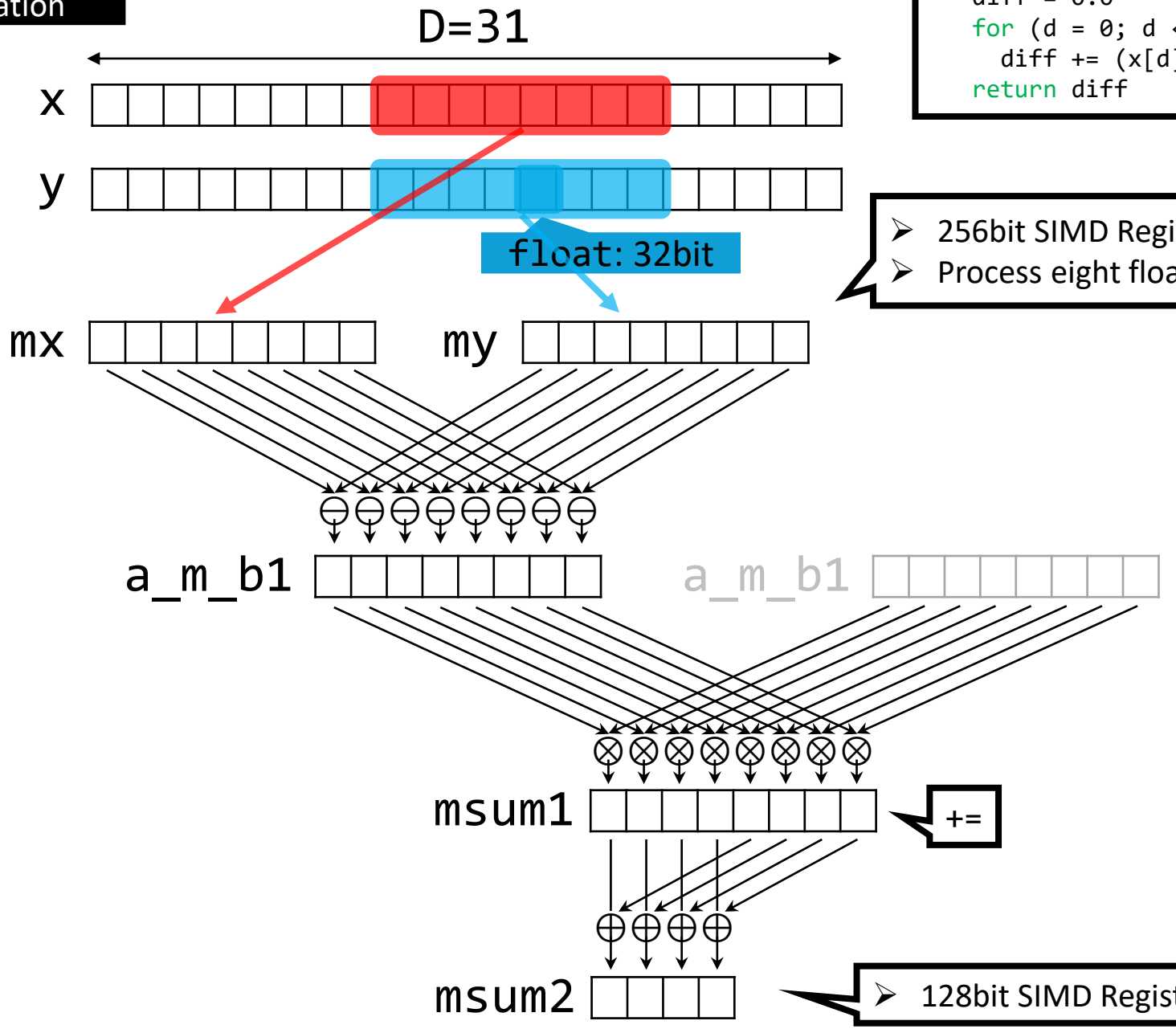
$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```

Ref.

```
def l2sqr(x, y):
    diff = 0.0
    for (d = 0; d < D; ++d):
        diff += (x[d] - y[d])**2
    return diff
```



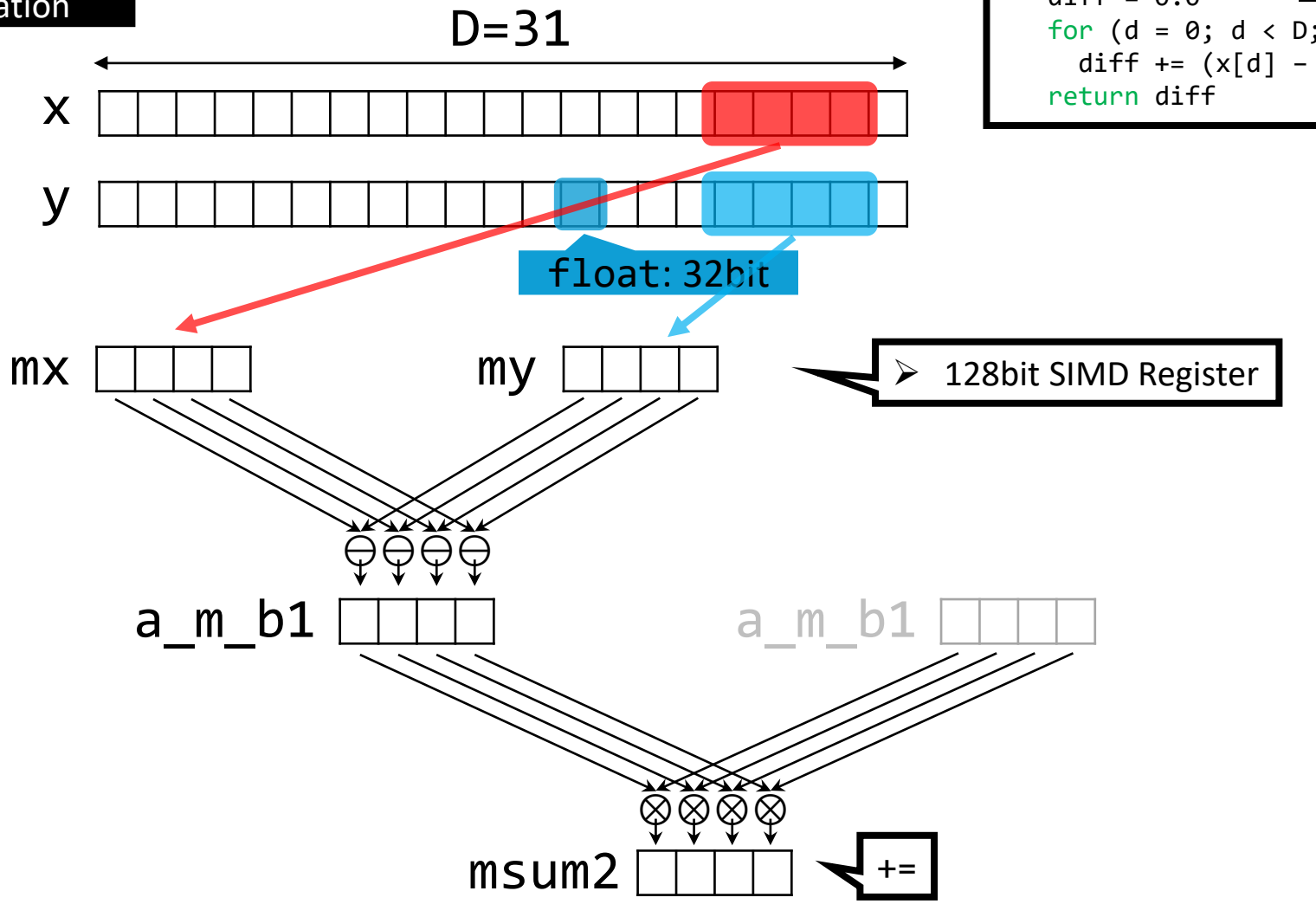
- 256bit SIMD Register
- Process eight floats at once

- 128bit SIMD Register

$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```



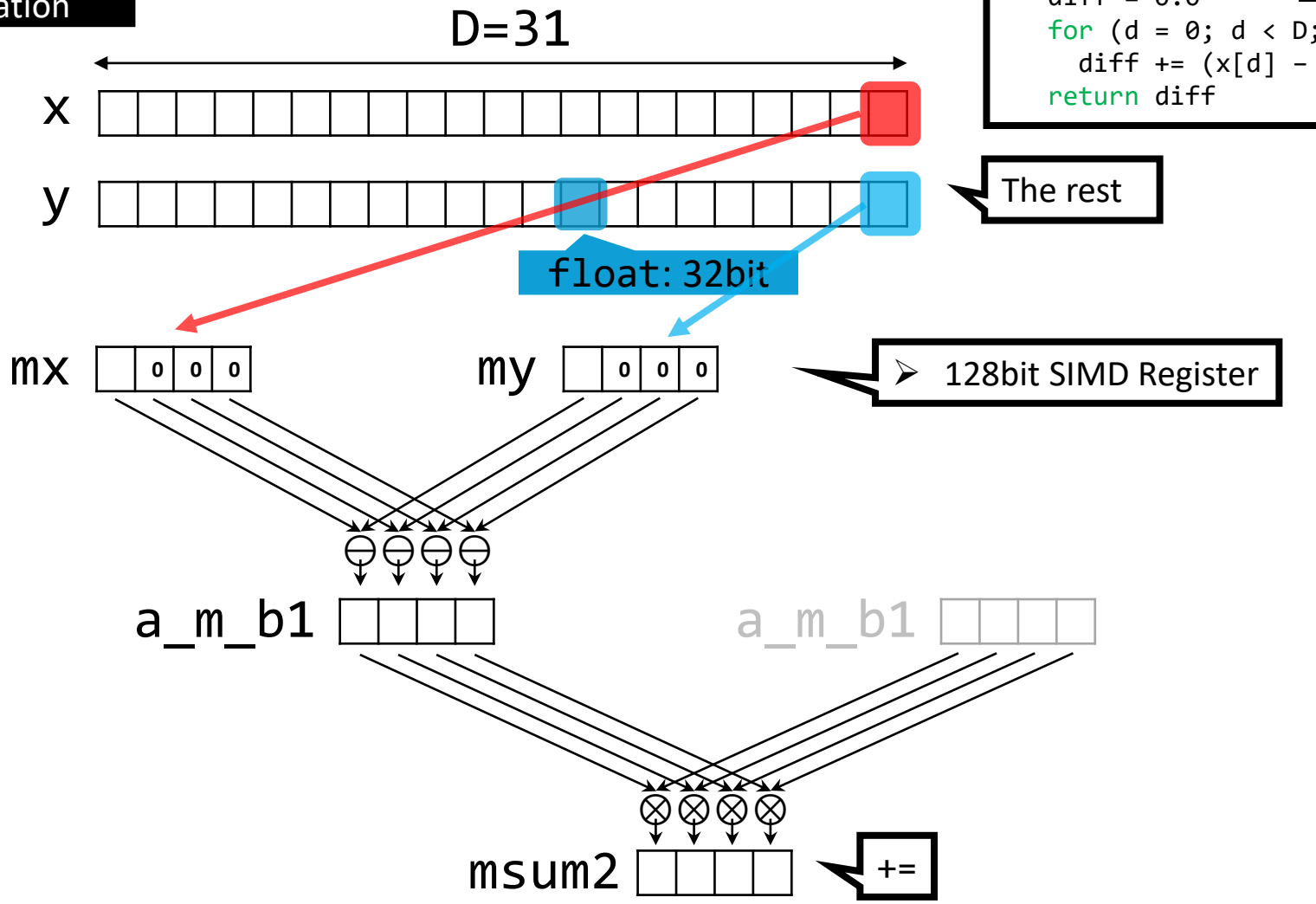
Ref.

```
def l2sqr(x, y):
    diff = 0.0
    for (d = 0; d < D; ++d):
        diff += (x[d] - y[d])**2
    return diff
```

$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```



Ref.

```
def l2sqr(x, y):
    diff = 0.0
    for (d = 0; d < D; ++d):
        diff += (x[d] - y[d])**2
    return diff
```

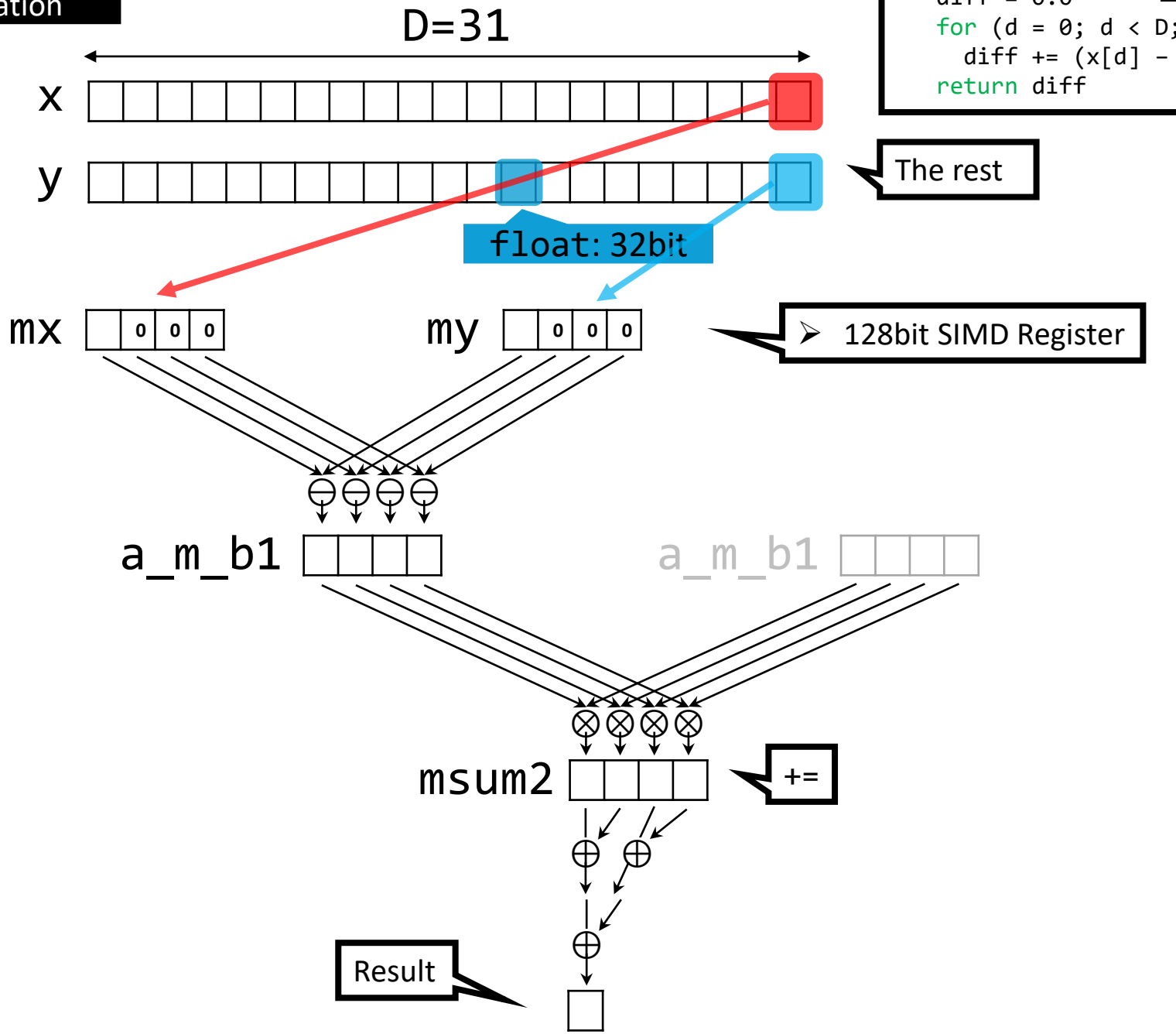
$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 += _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```

Ref.

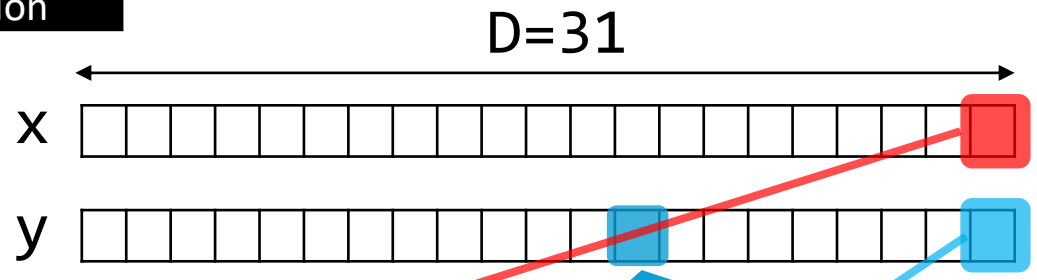
```
def l2sqr(x, y):
    diff = 0.0
    for (d = 0; d < D; ++d):
        diff += (x[d] - y[d])**2
    return diff
```



$\|x - y\|_2^2$ by SIMD

Rename variables for the sake of explanation

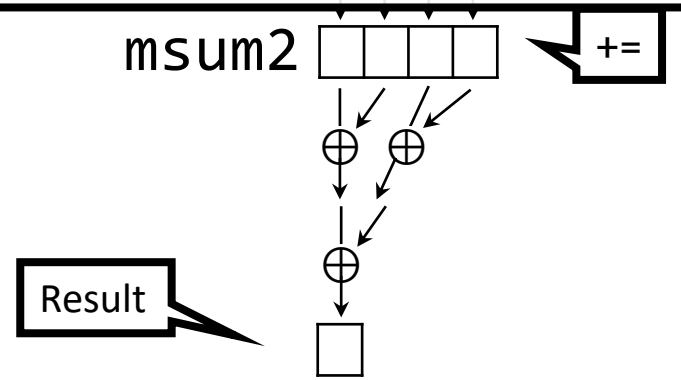
```
float fvec_L2sqr (const float * x,
                  const float * y,
                  size_t d)
{
    __m256 msum1 = _mm256_setzero_ps();

    while (d >= 8) {
        __m256 mx = _mm256_loadu_ps (x); x += 8;
        __m256 my = _mm256_loadu_ps (y); y += 8;
        const __m256 a_m_b1 = mx - my;
        msum1 += a_m_b1 * a_m_b1;
        d -= 8;
    }

    __m128 msum2 = _mm256_extractf128_ps(msum1, 1);
    msum2 +=  _mm256_extractf128_ps(msum1, 0);

    if (d >= 4) {
        __m128 mx = _mm_loadu_ps (x); x += 4;
        __m128 my = _mm_loadu_ps (y); y += 4;
        const __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
        d -= 4;
    }

    if (d > 0) {
        __m128 mx = masked_read (d, x);
        __m128 my = masked_read (d, y);
        __m128 a_m_b1 = mx - my;
        msum2 += a_m_b1 * a_m_b1;
    }

    msum2 = _mm_hadd_ps (msum2, msum2);
    msum2 = _mm_hadd_ps (msum2, msum2);
    return  _mm_cvtss_f32 (msum2);
}
```




```
def l2sqr(x, y):
  diff = 0.0
  for (d = 0; d < D; ++d):
    diff += (x[d] - y[d])**2
  return diff
```

Ref.

- SIMD codes of faiss are simple and easy to read
- Being able to read SIMD codes comes in handy sometimes; *why this impl is super fast*
- Another example of SIMD L2sqr from HNSW:
  https://github.com/nmslib/hnswlib/blob/master/hnswlib/space_l2.h



M D-dim query vectors $\mathcal{Q} = \{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_M\}$

N D-dim database vectors $\mathcal{X} = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N\}$

Task：Given $\boldsymbol{q} \in \mathcal{Q}$ and $\boldsymbol{x} \in \mathcal{X}$, compute $\|\boldsymbol{q} - \boldsymbol{x}\|_2^2$

Naïve impl.

```
def l2sqr(q, x):
  diff = 0.0
  for (d = 0; d < D; ++d):
    diff += (q[d] - x[d])**2
  return diff
```

```
parfor q in Q:
  for x in X:
    l2sqr(q, x)
```

Parallelize query-side

Select min by heap, but omit it now

faiss impl.

if $M < 20$：

compute $\|\boldsymbol{q} - \boldsymbol{x}\|_2^2$ by SIMD

else：

compute $\|\boldsymbol{q} - \boldsymbol{x}\|_2^2 = \|\boldsymbol{q}\|_2^2 - 2\boldsymbol{q}^\top \boldsymbol{x} + \|\boldsymbol{x}\|_2^2$ by BLAS

Compute $\|\boldsymbol{q}-\boldsymbol{x}\|_2^2 = \|\boldsymbol{q}\|_2^2 - 2\boldsymbol{q}^\top\boldsymbol{x} + \|\boldsymbol{x}\|_2^2$ with BLAS

Stack M D-dim query vectors to a $D \times M$ matrix: $Q = [\boldsymbol{q}_1, \boldsymbol{q}_2, \dots, \boldsymbol{q}_M] \in \mathbb{R}^{D\times M}$

Stack N D-dim database vectors to a $D \times N$ matrix: $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N] \in \mathbb{R}^{D\times N}$

SIMD-accelerated function

```
# Compute tables
q_norms = norms(Q)      #  ||q_1||_2^2, ||q_2||_2^2, ..., ||q_M||_2^2
x_norms = norms(X)      #  ||x_1||_2^2, ||x_2||_2^2, ..., ||x_N||_2^2
ip = sgemm_(Q, X, …)    #  Q^T X

# Scan and sum
parfor (m = 0; m < M; ++m):
  for (n = 0; n < N; ++n):
    dist = q_norms[m] + x_norms[n] – 2 * ip[m][n]
```

- Matrix multiplication by BLAS
- Dominant if Q and X are large
- The difference of the background matters:
  - ✓ Intel MKL is 30% faster than OpenBLAS

$\|\boldsymbol{q}_m - \boldsymbol{x}_n\|_2^2$ $\|\boldsymbol{q}_m\|_2^2$ $\|\boldsymbol{x}_n\|_2^2$ $(Q^\top X)_{mn}$

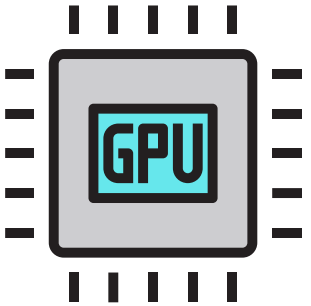
NN in GPU (faiss-gpu) is 10x faster than NN in CPU (faiss-cpu)

Benchmark: https://github.com/facebookresearch/faiss/wiki/Low-level-benchmarks

- NN-GPU always compute $\|\boldsymbol{q}\|_2^2 - 2\boldsymbol{q}^\top \boldsymbol{x} + \|\boldsymbol{x}\|_2^2$

- k-means for 1M vectors (D=256, K=20000)
  - ✓ 11 min on CPU
  - ✓ 55 sec on 1 Pascal-class P100 GPU (float32 math)
  - ✓ 34 sec on 1 Pascal-class P100 GPU (float16 math)
  - ✓ 21 sec on 4 Pascal-class P100 GPUs (float32 math)
  - ✓ 16 sec on 4 Pascal-class P100 GPUs (float16 math)

x10 faster



- If GPU is available and its memory is enough, try GPU-NN
- The behavior is little bit different (e.g., a restriction for top-k)

Outline

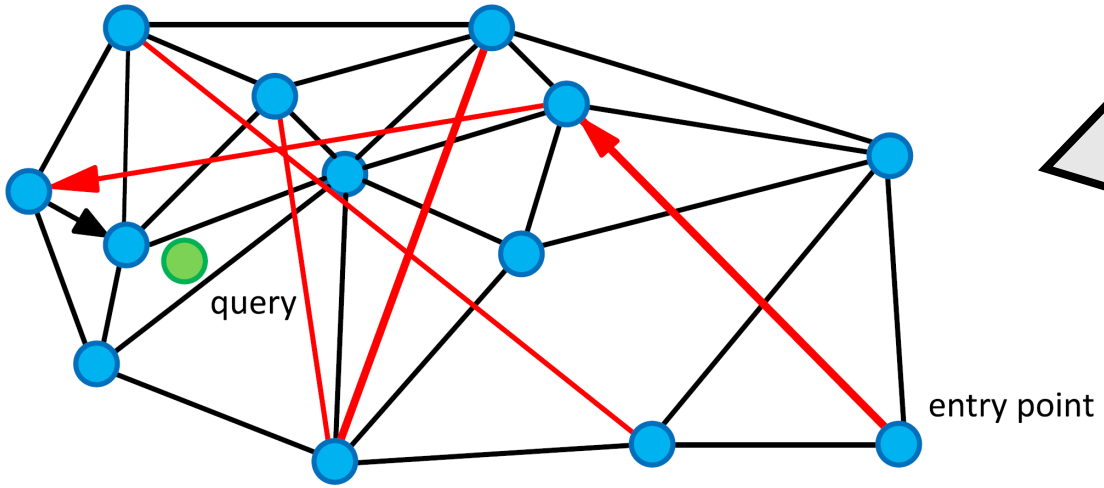
1. History from an applications perspective

2. Importance of implementation:
   nearest neighbor search in faiss

3. **Basics of modern baseline: graph-based search**

Graph search

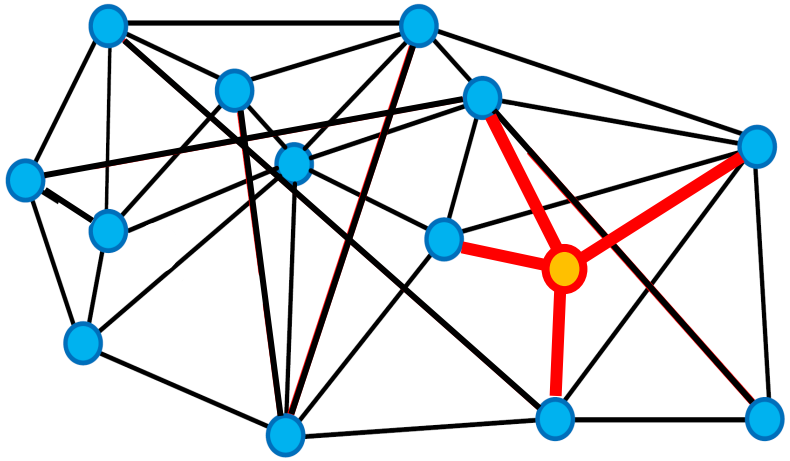
- De facto standard if all data can be loaded on memory
- Fast and accurate for real-world data
- Important for billion-scale situation as well
  - ✓ Graph-search is a building block for billion-scale systems

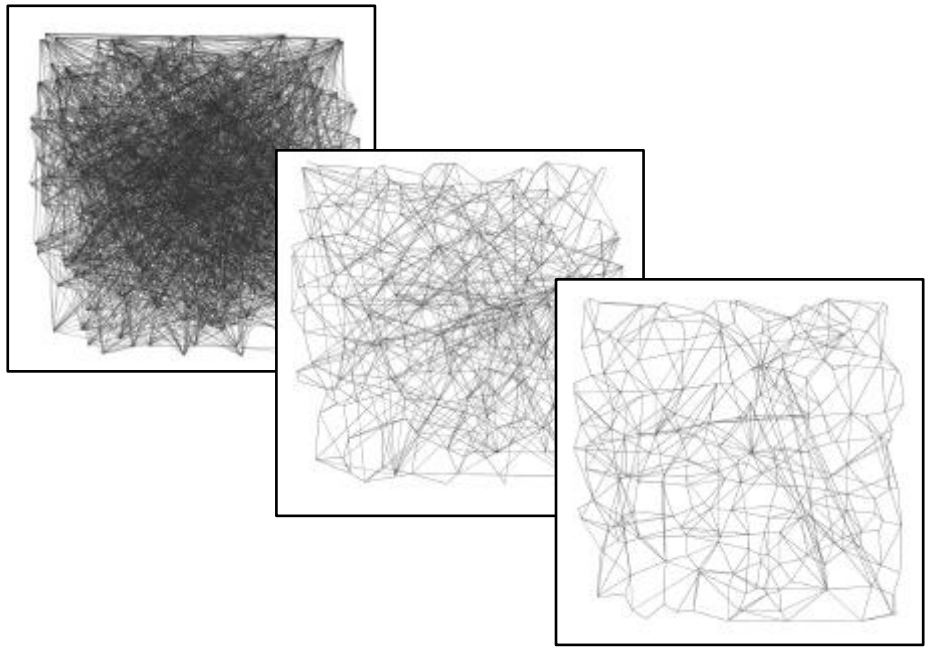


- Traverse graph towards the query
- Seems intuitive, but not so much easy to understand
- Review the algorithm carefully

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

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





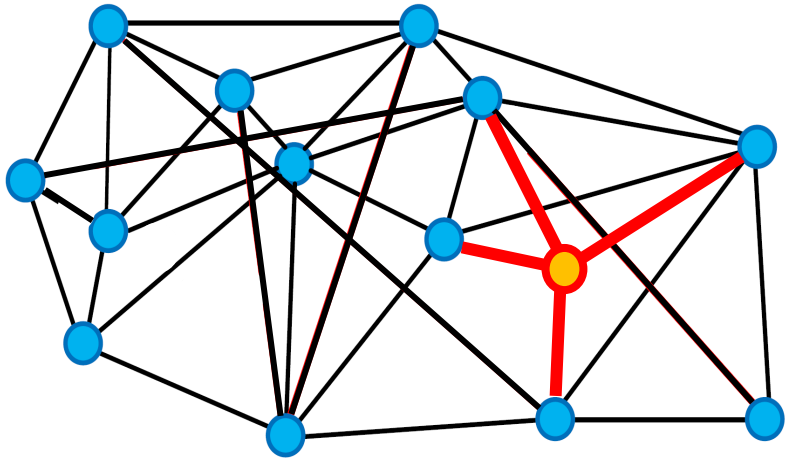
Increment approach

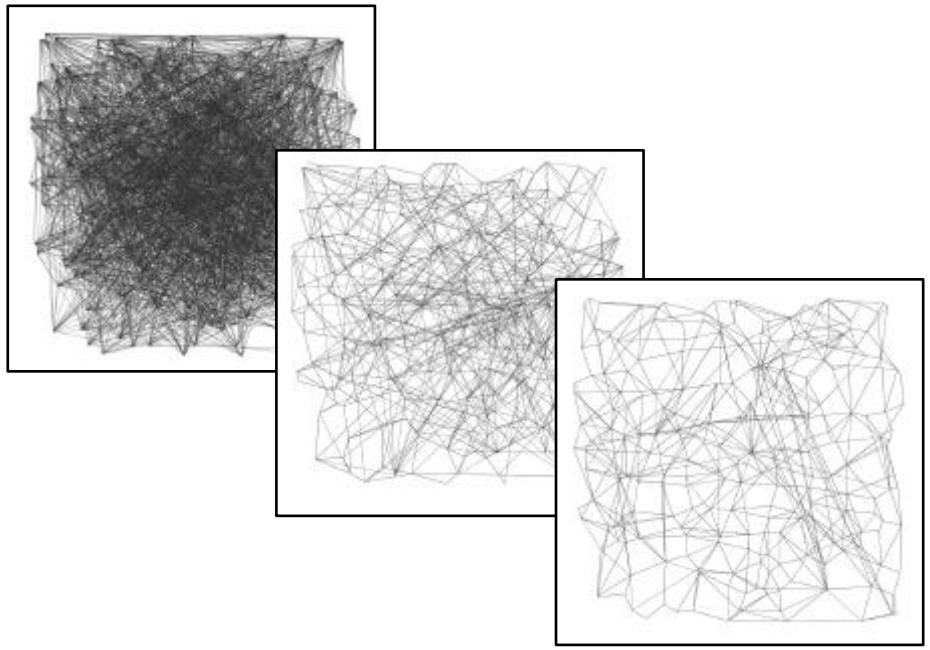
- Add a new item to the current graph incrementally

Refinement approach

- Iteratively refine an initial graph

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





Increment approach

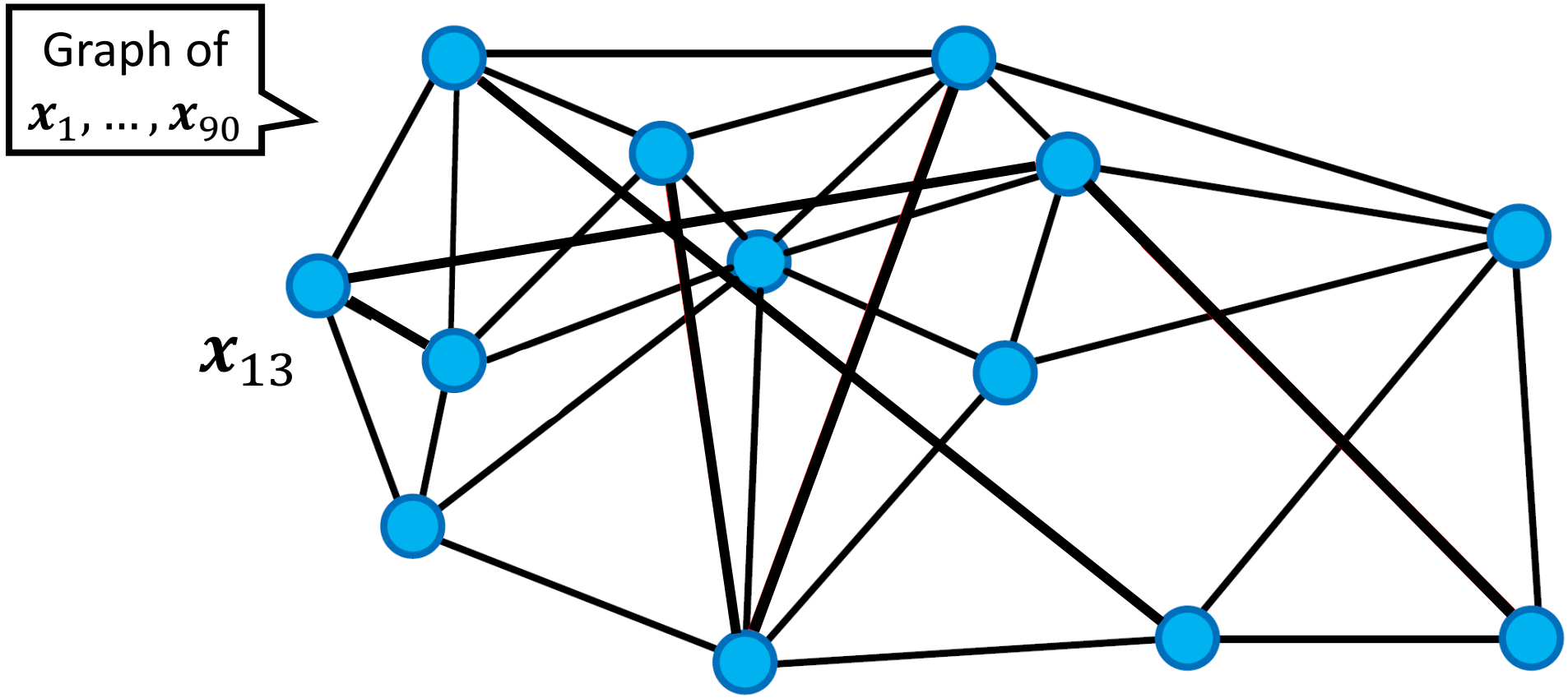
- Add a new item to the current graph incrementally

Refinement approach

- Iteratively refine an initial graph

Construction: incremental approach

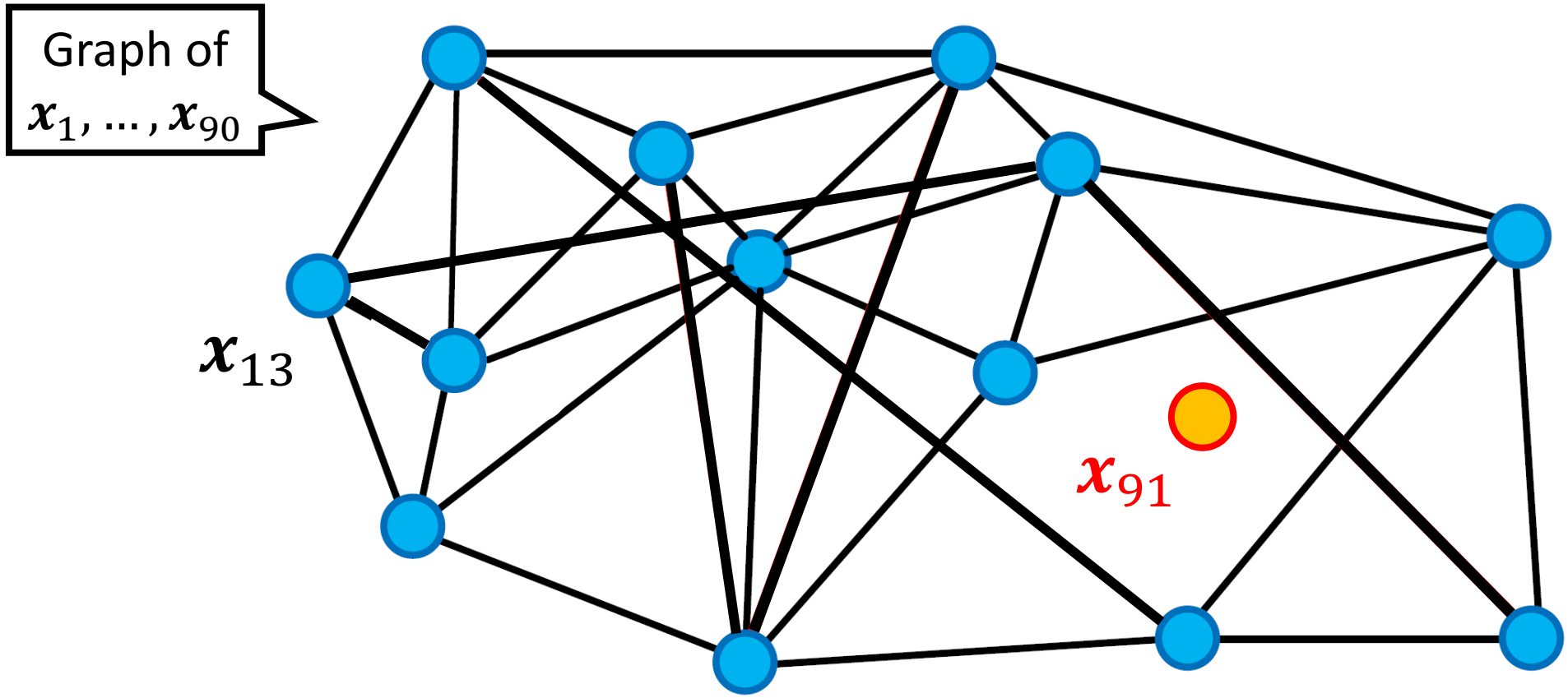
Images are from [Malkov+, Information Systems, 2013]



- Each node is a database vector

Construction: incremental approach

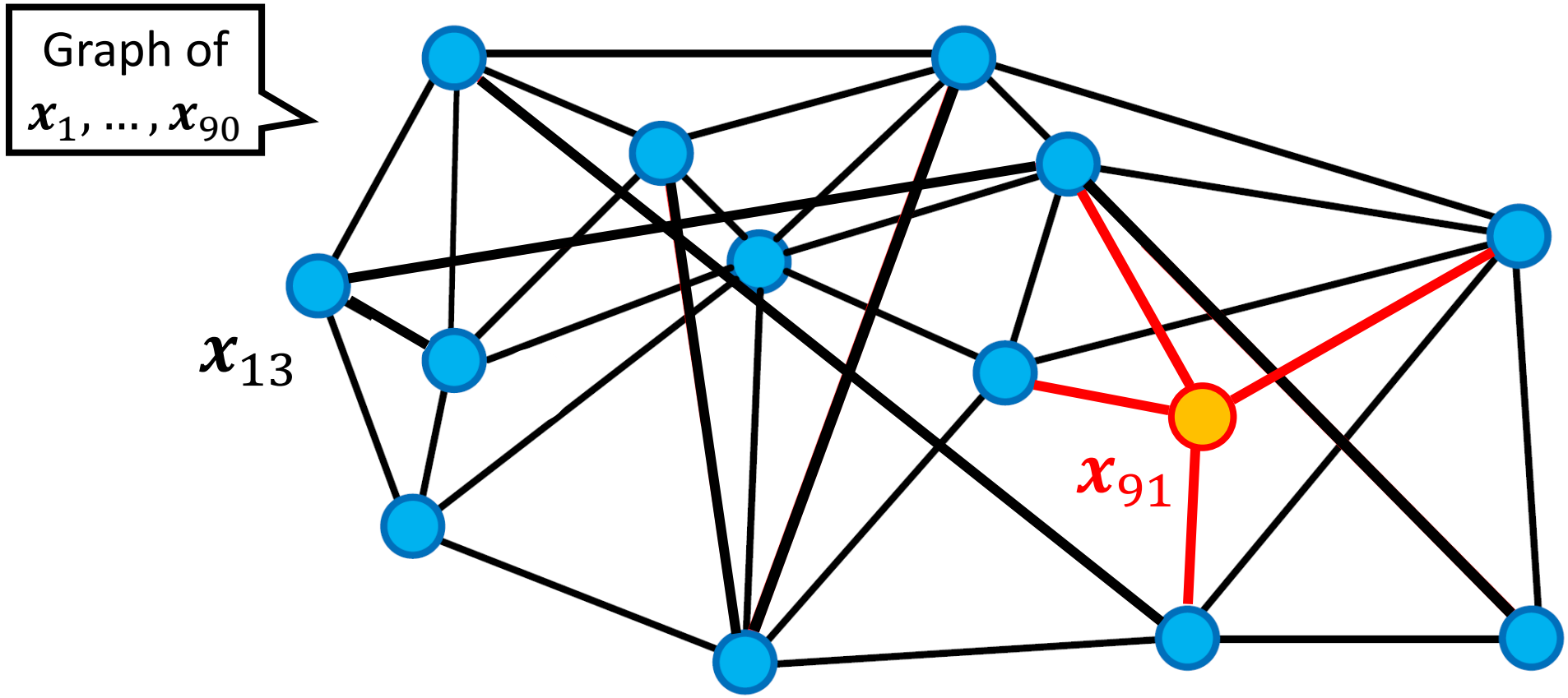
Images are from [Malkov+, Information Systems, 2013]



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

Construction: incremental approach

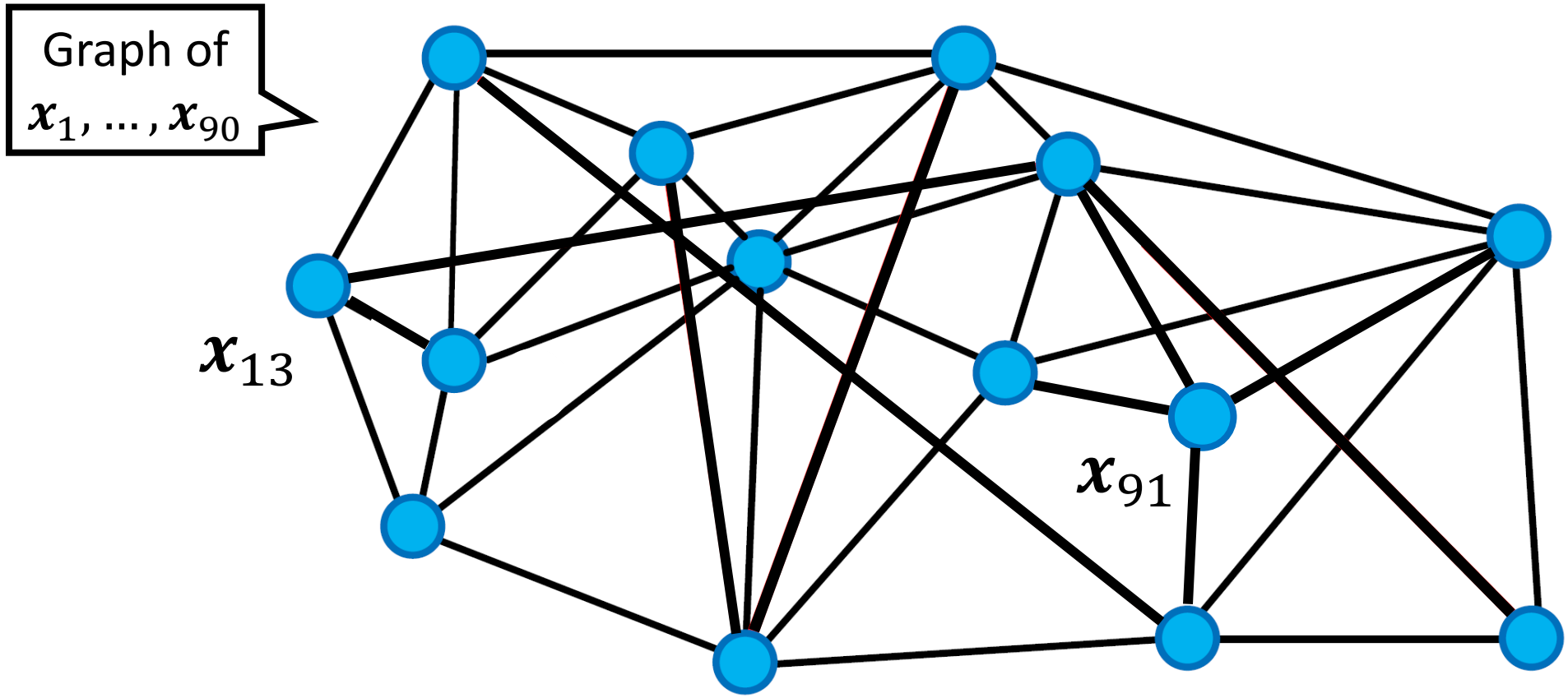
Images are from [Malkov+, Information Systems, 2013]



- 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]

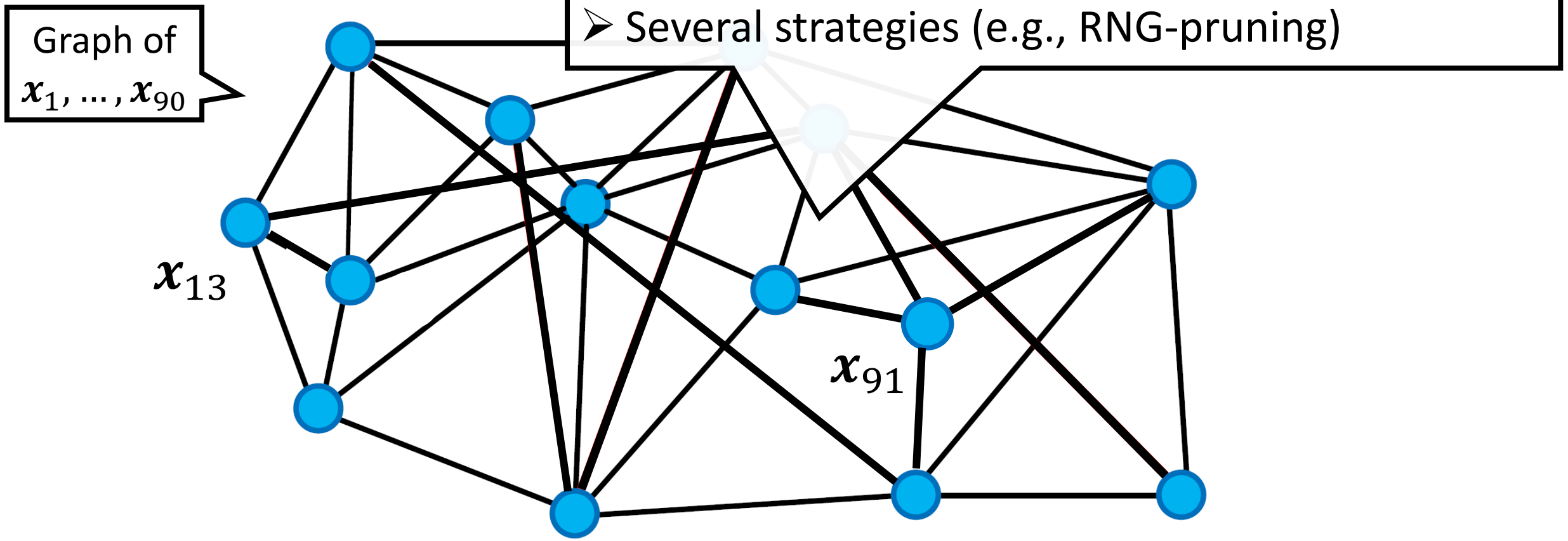


- Each node is a database vector
- Given a new database vector, create new edges to neighbors

Construction: incremental approach

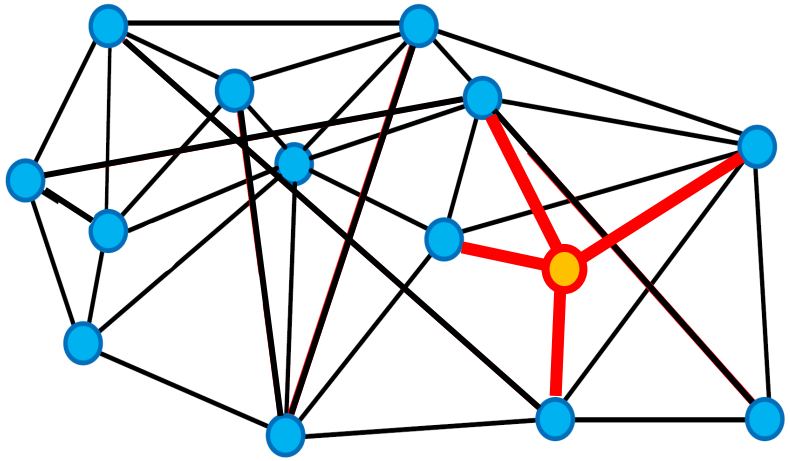
Images are from [Malkov, Information Systems, 2013]

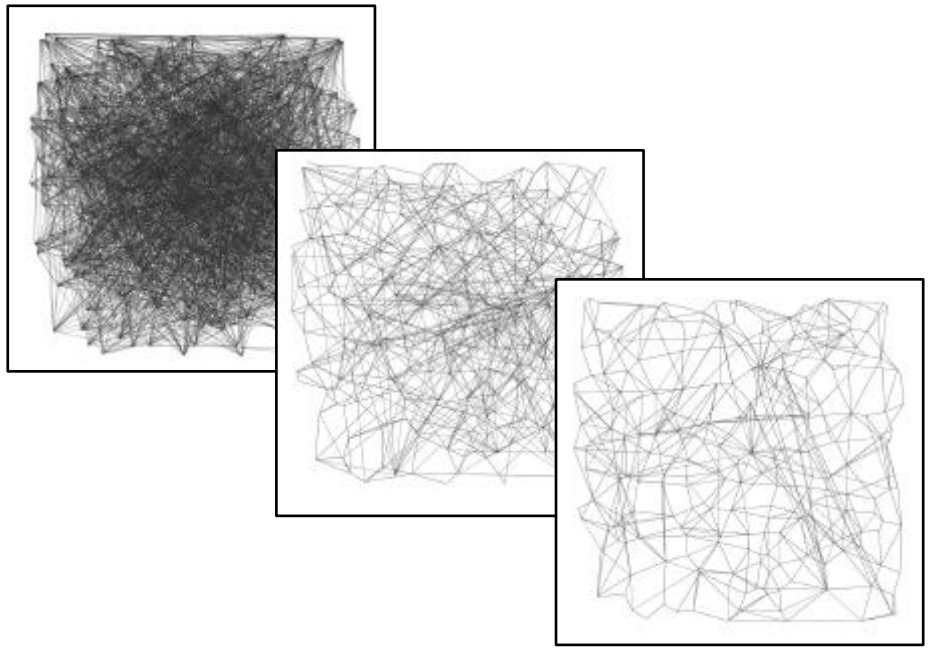
- Prune edges if some node have too many edges
- Several strategies (e.g., RNG-pruning)



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

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





Increment approach

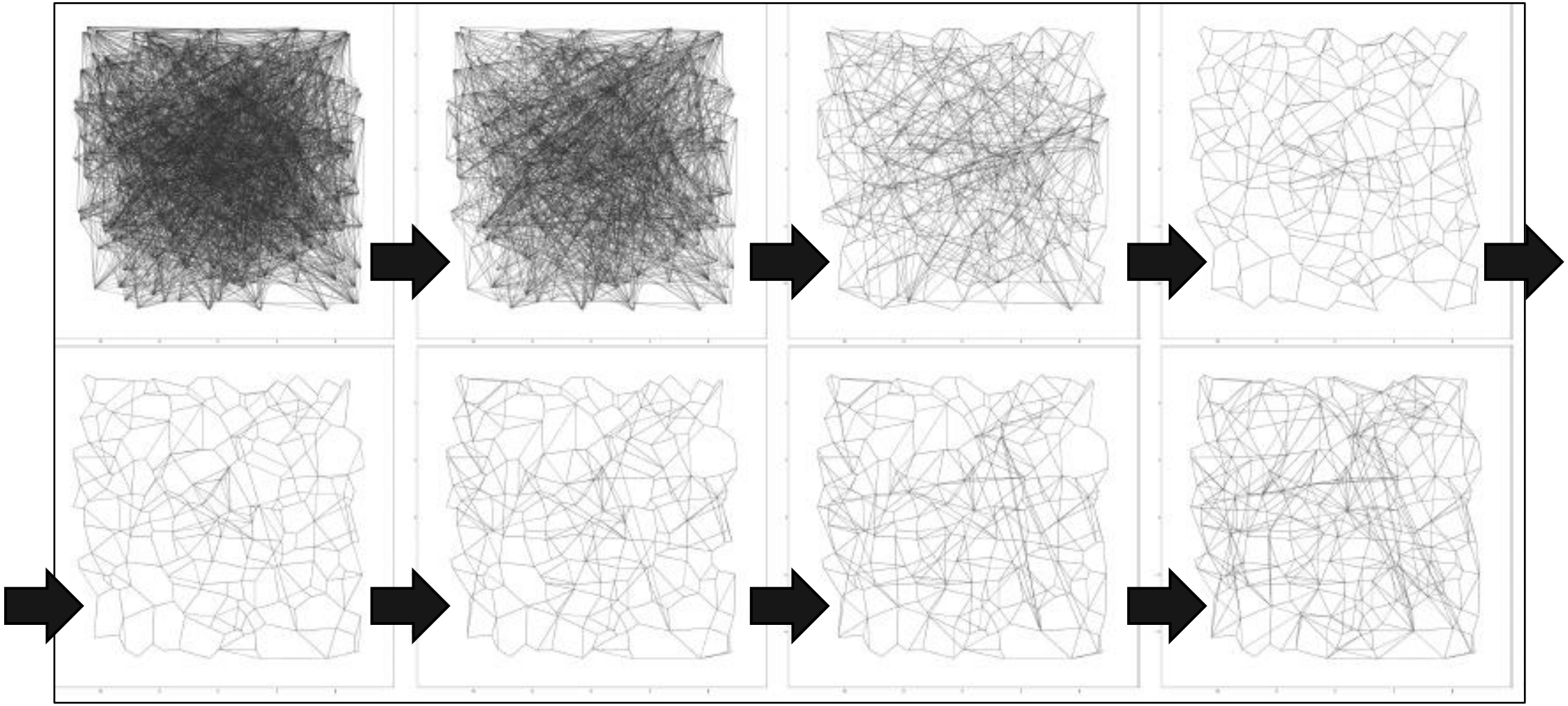
- Add a new item to the current graph incrementally

Refinement approach

- Iteratively refine an initial graph

Construction: refinement approach

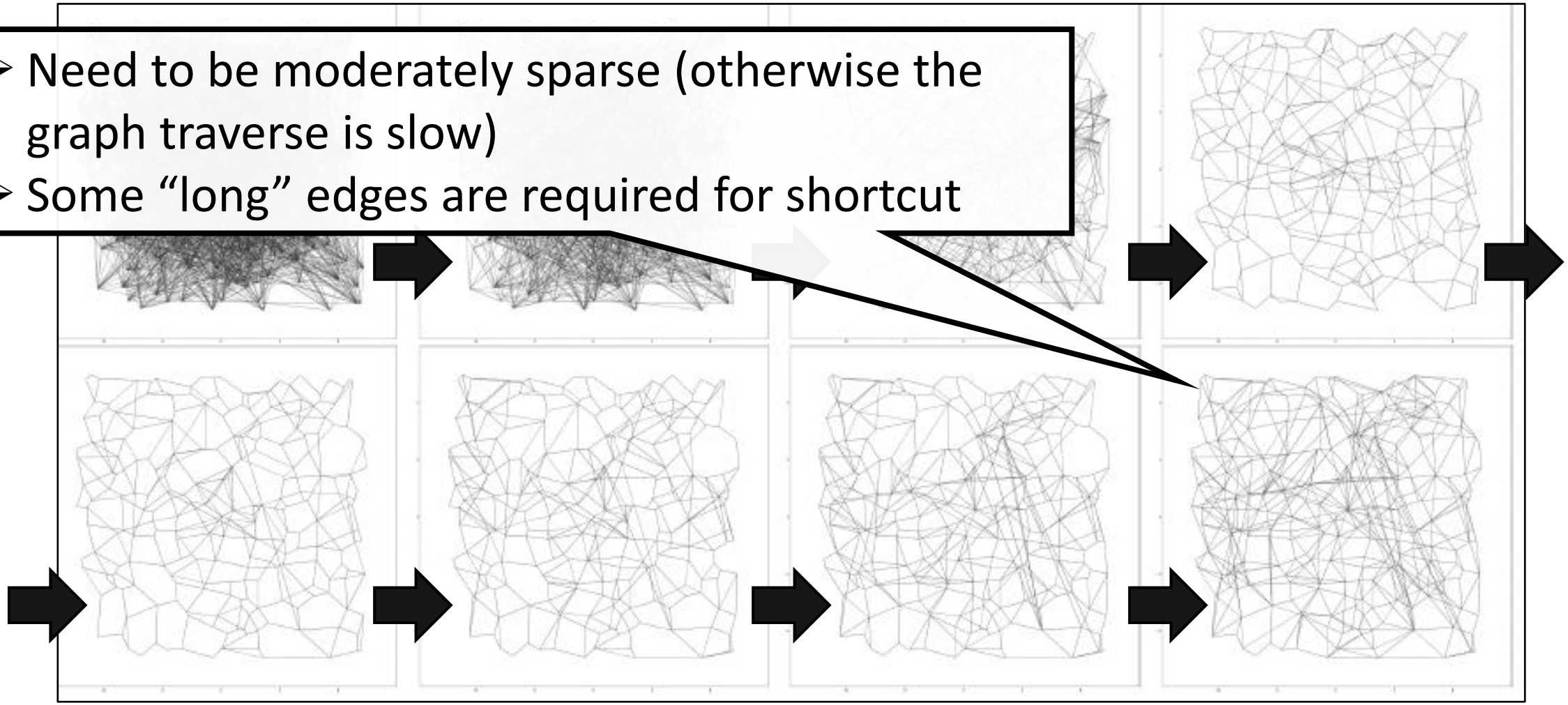
Images are from [Subramanya+, NeruIPS 2019]



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

Construction: refinement approach

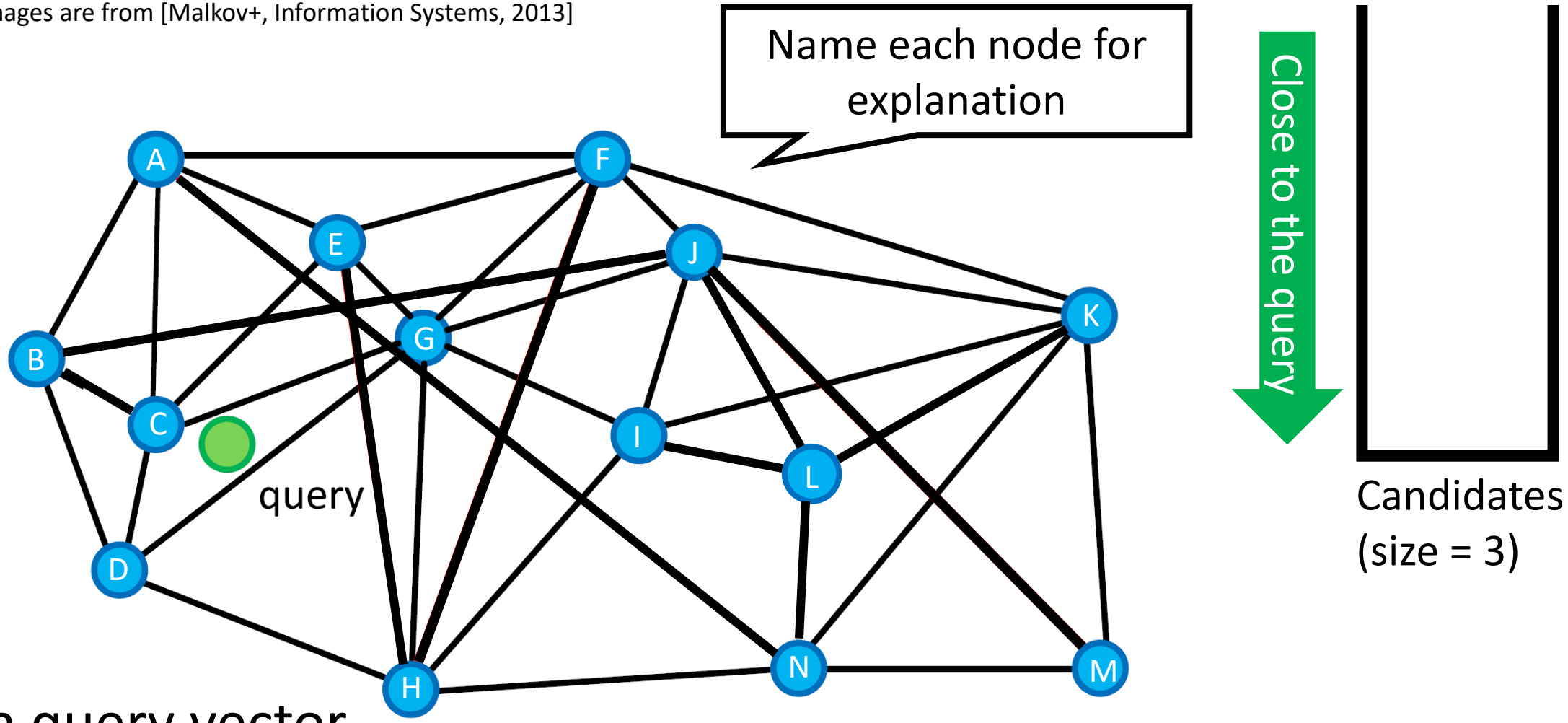
Images are from [Subramanya+, NeruIPS 2019]



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

Search

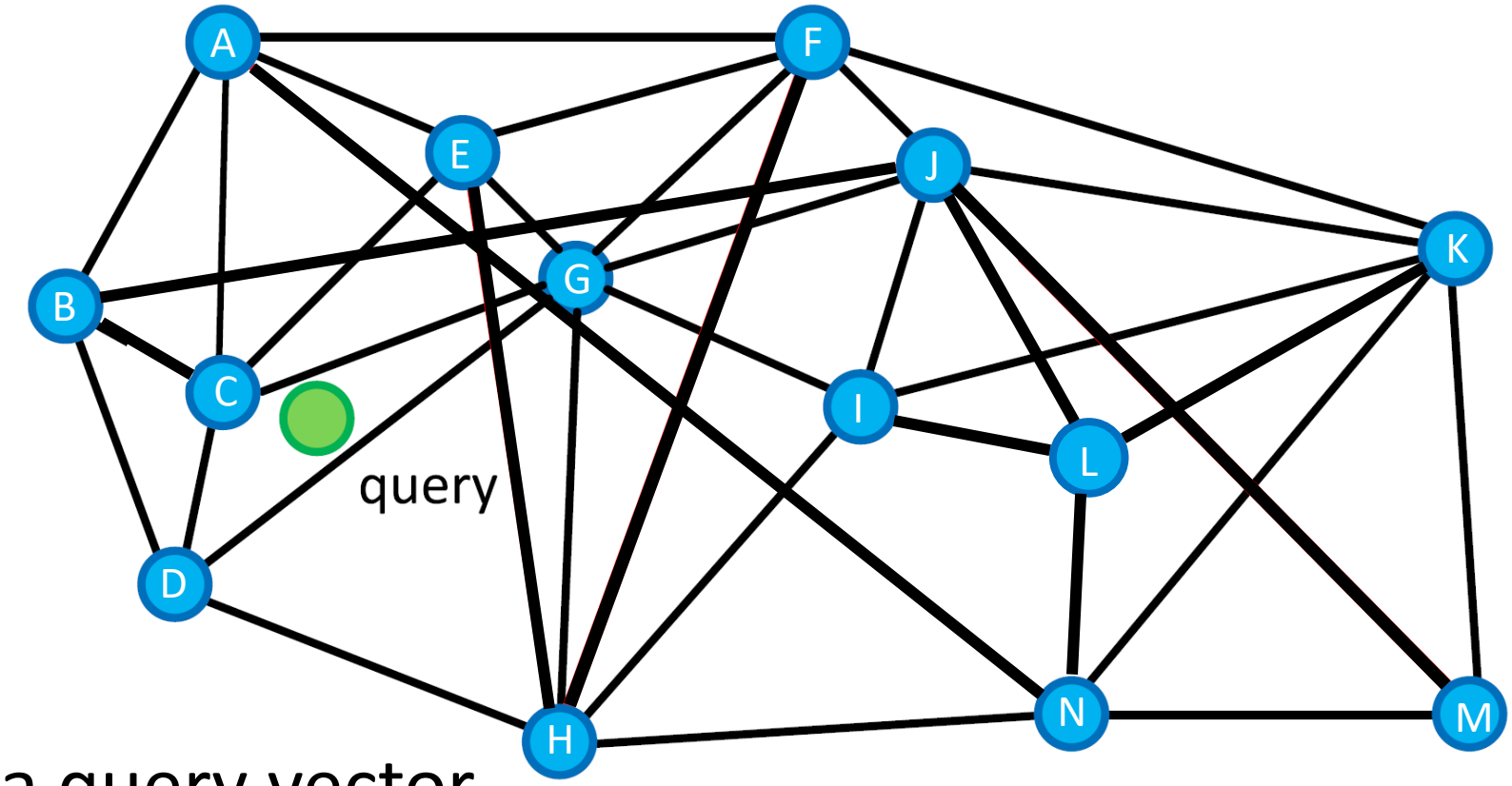
Images are from [Malkov+, Information Systems, 2013]

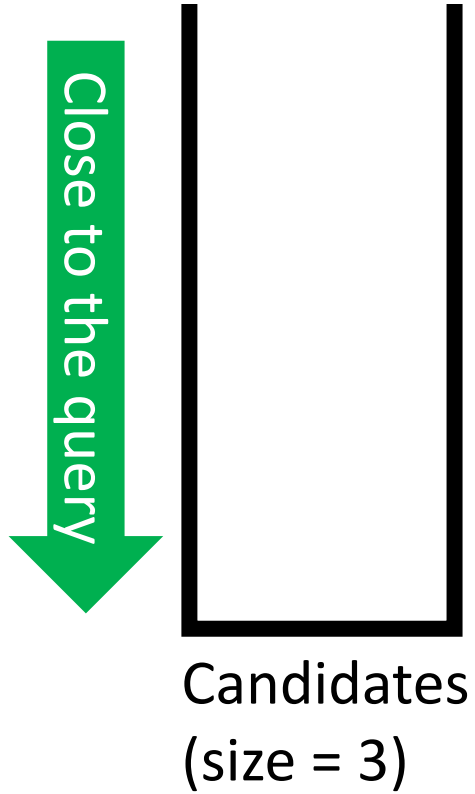


- Given a query vector

Search

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

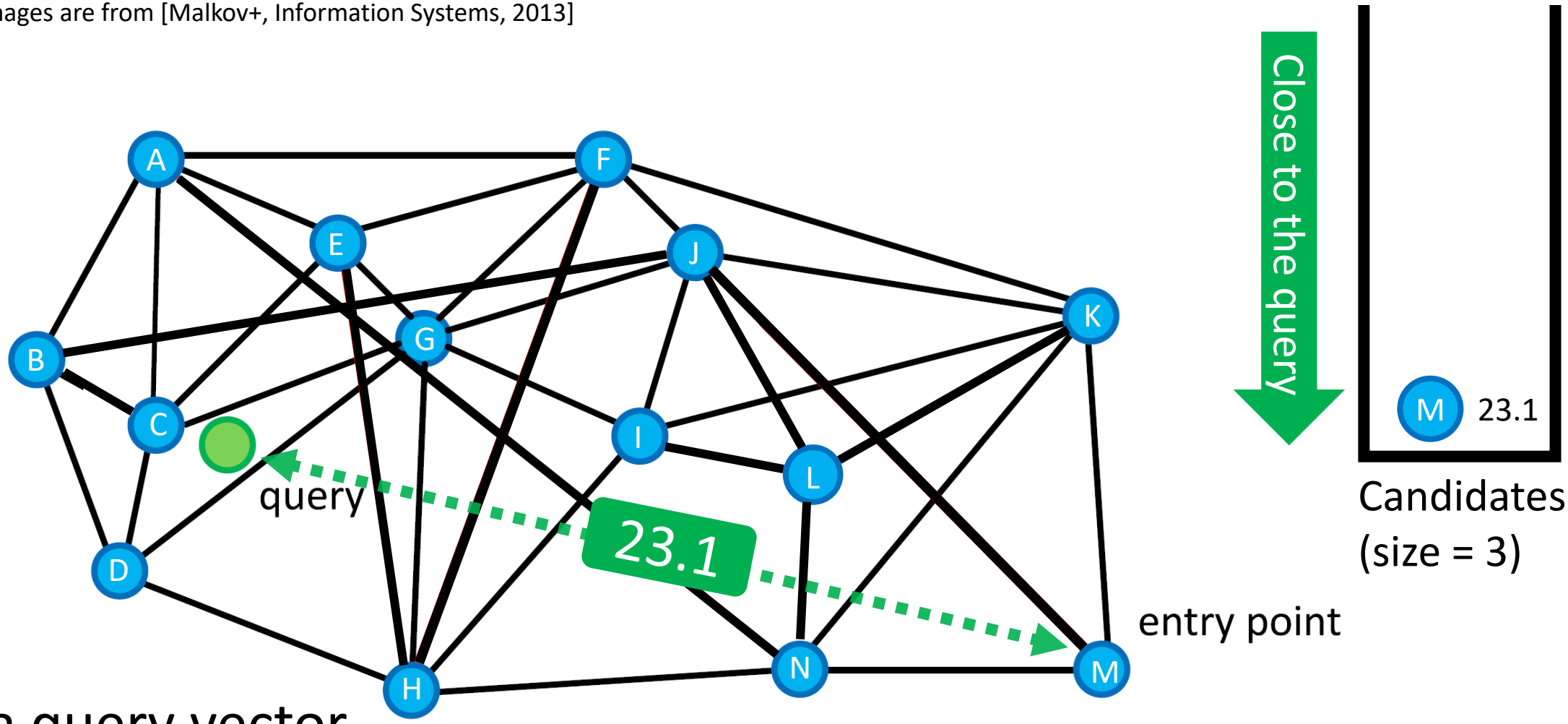




- Given a query vector
- Start from an entry point (e.g., M)

Search

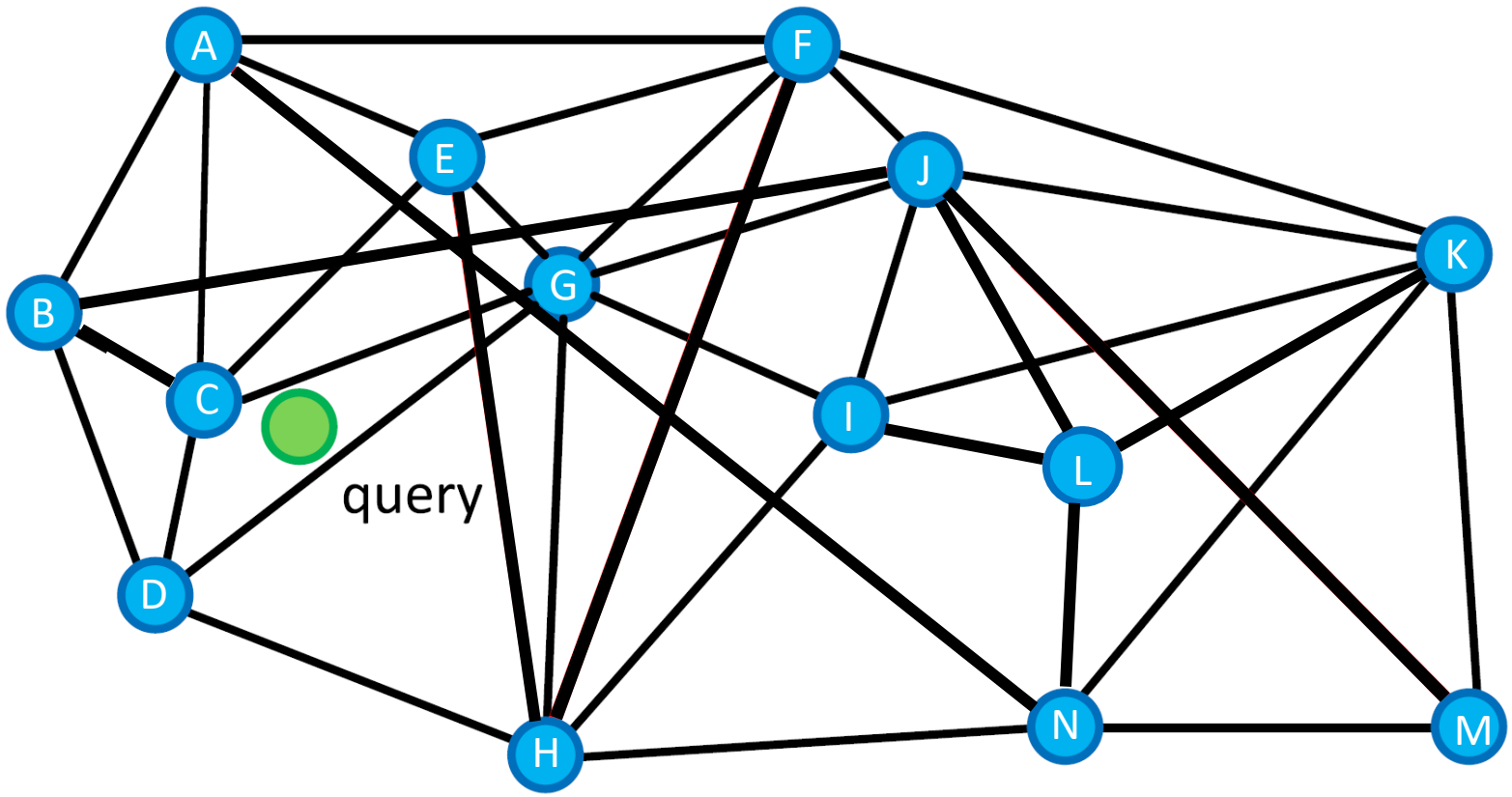
Images are from [Malkov+, Information Systems, 2013]

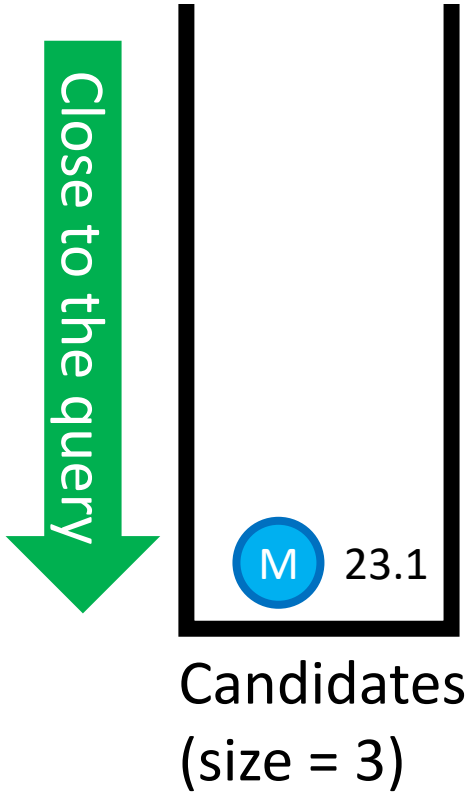


- Given a query vector
- Start from an entry point (e.g., M). Record the distance to q.

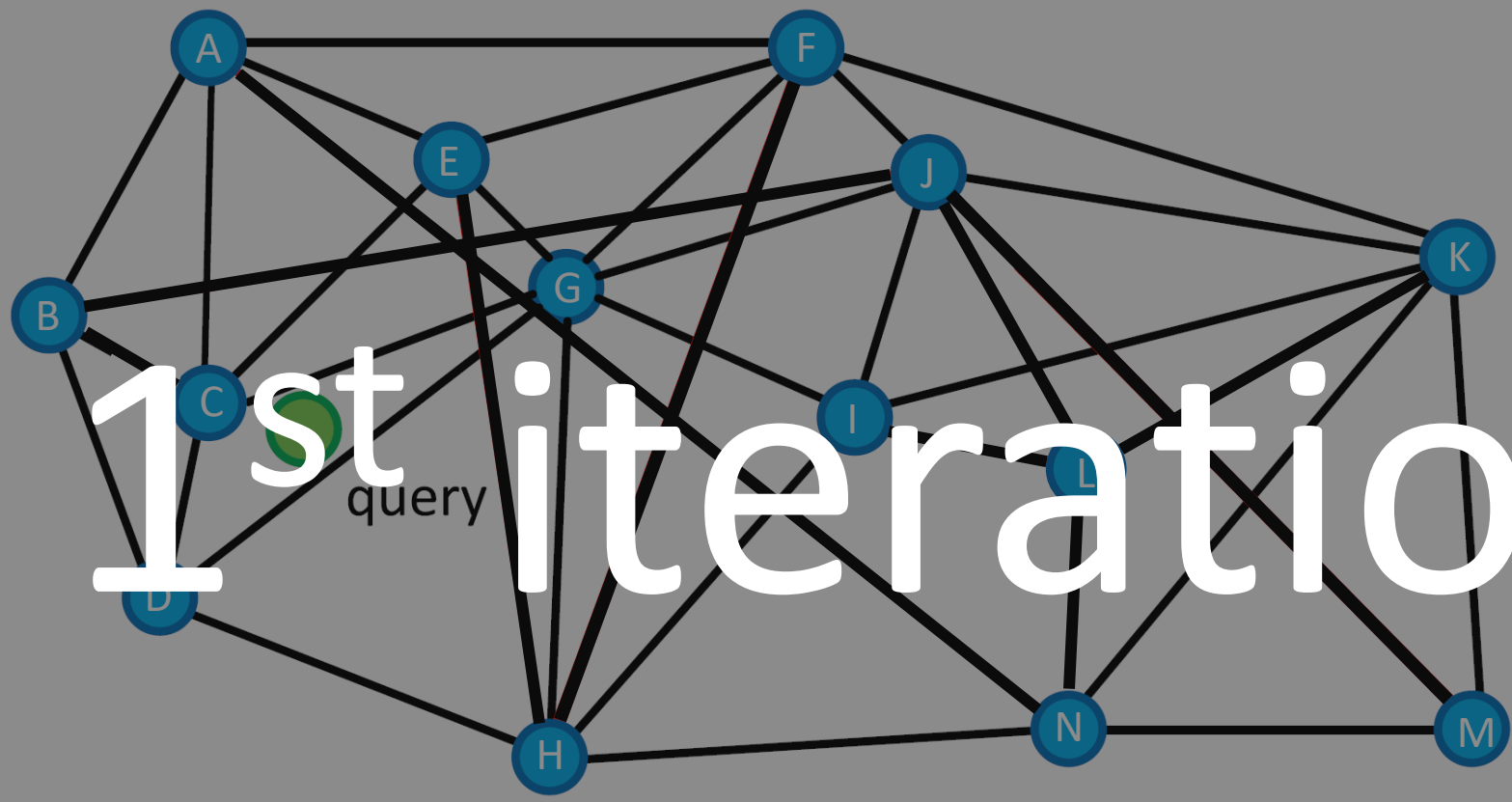
Search

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



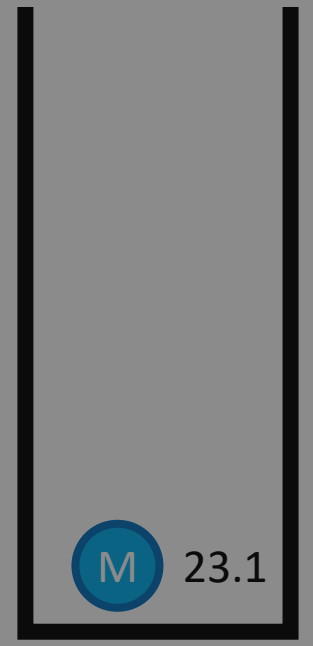


Images are from [Malkov+, Information Systems, 2013]



1st iteration

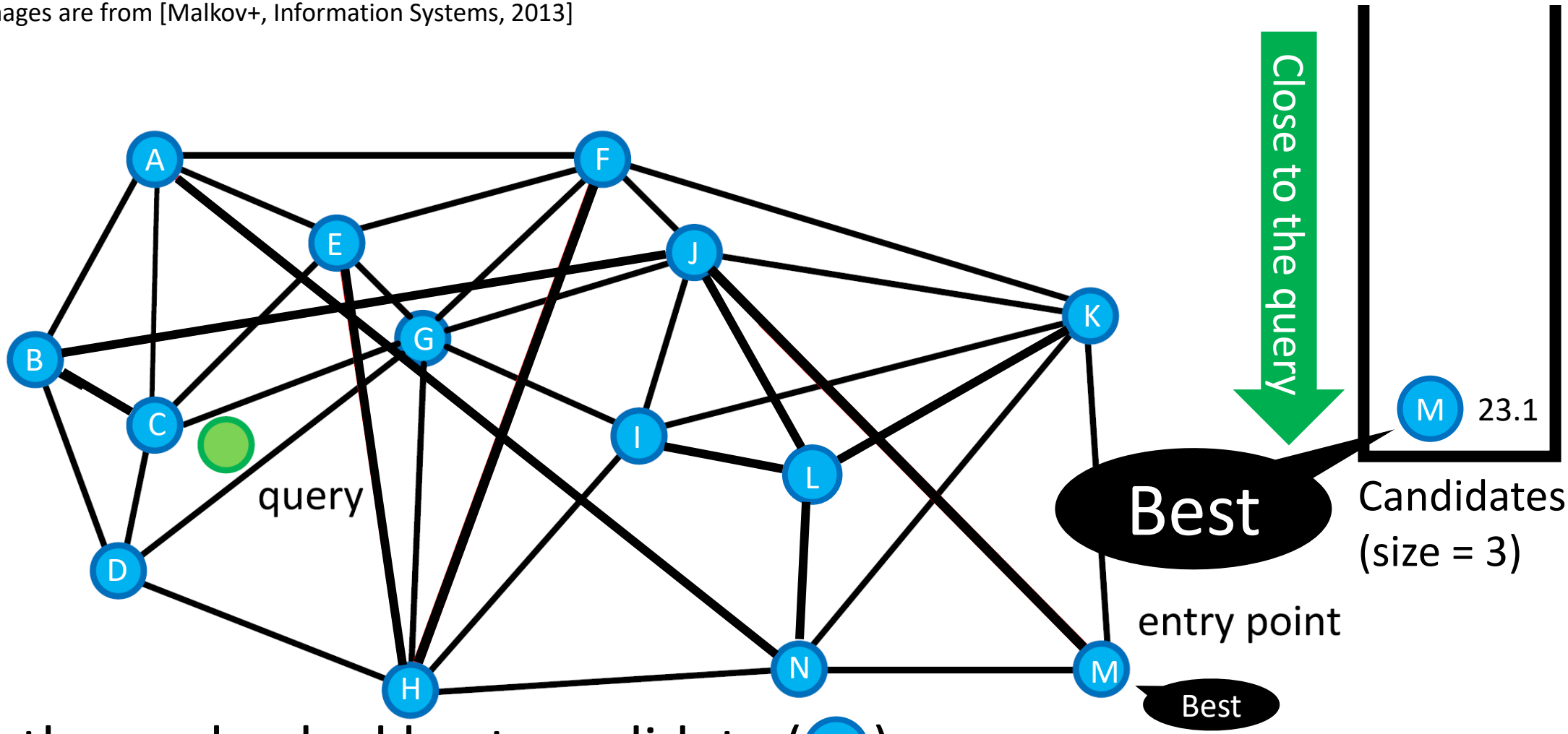
Close to the query



Candidates
(size = 3)

Search

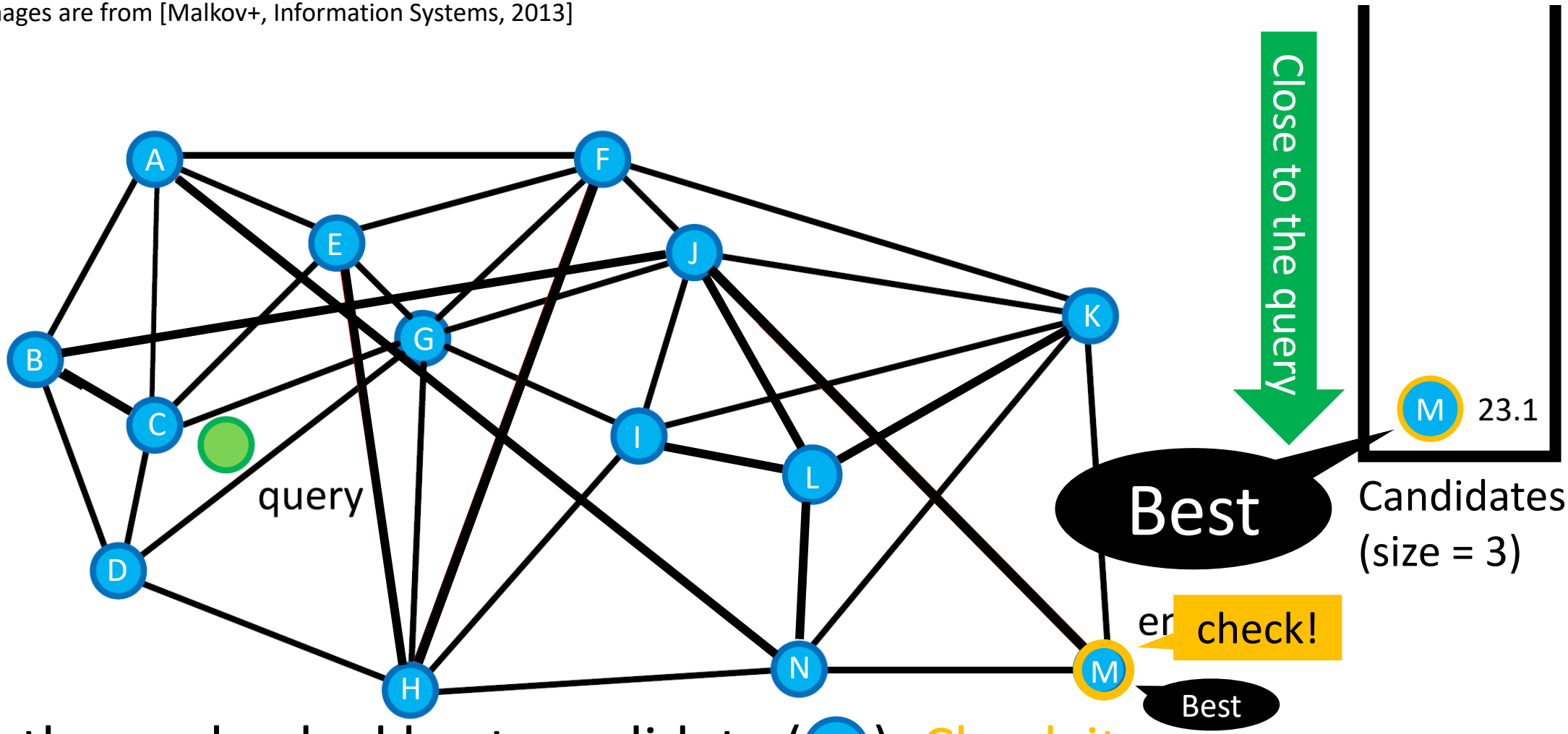
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (M)

Search

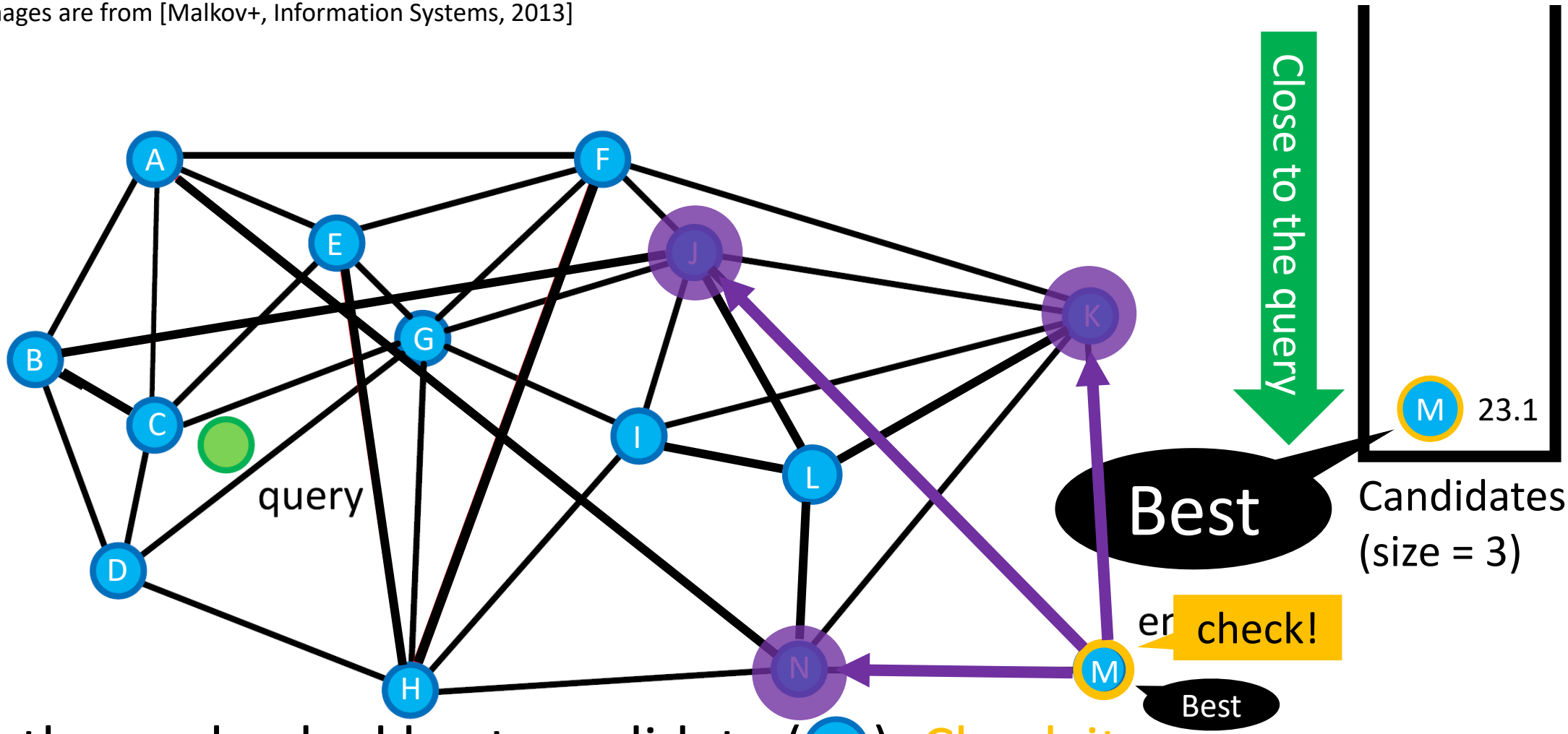
Images are from [Malkov+, Information Systems, 2013]



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

Search

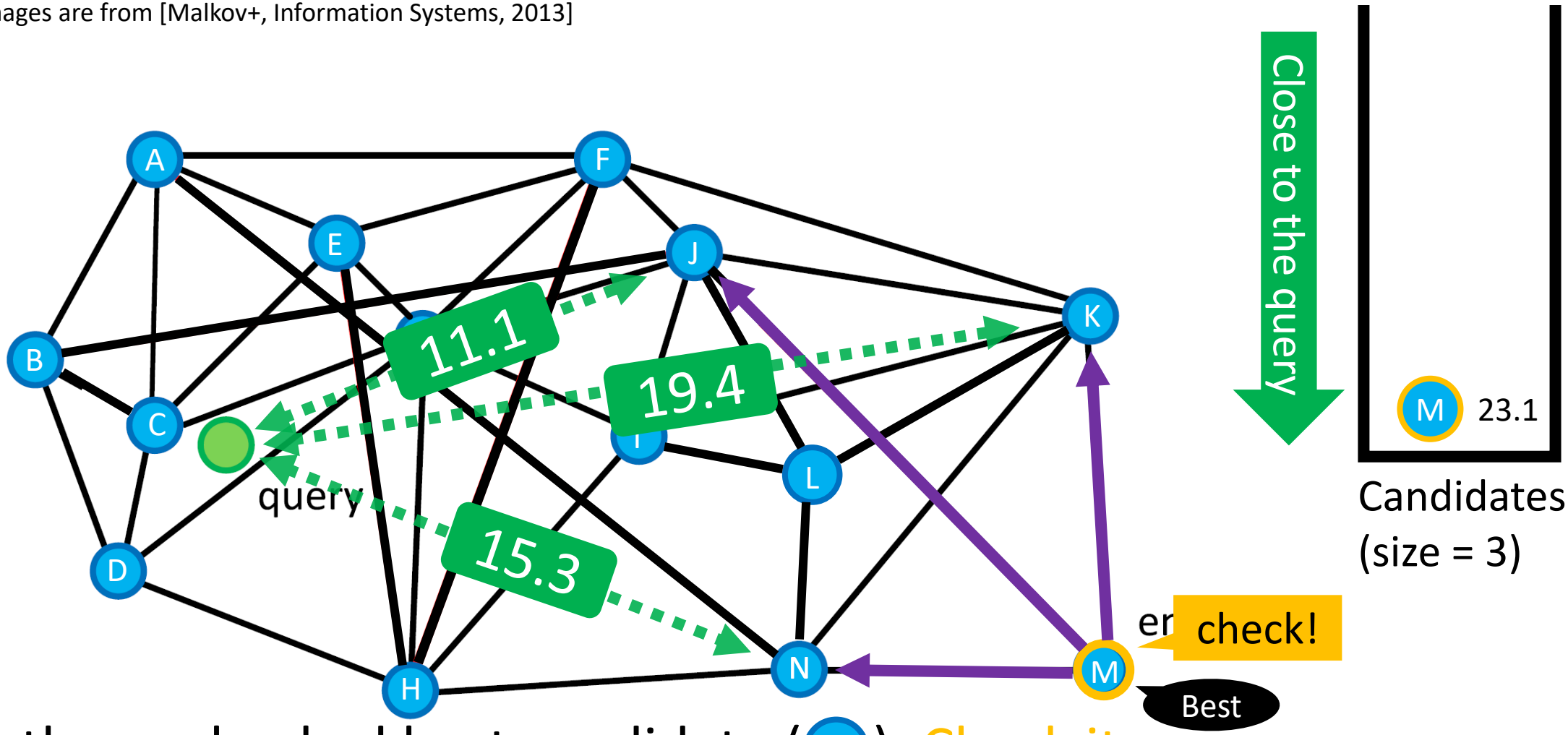
Images are from [Malkov+, Information Systems, 2013]



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

Search

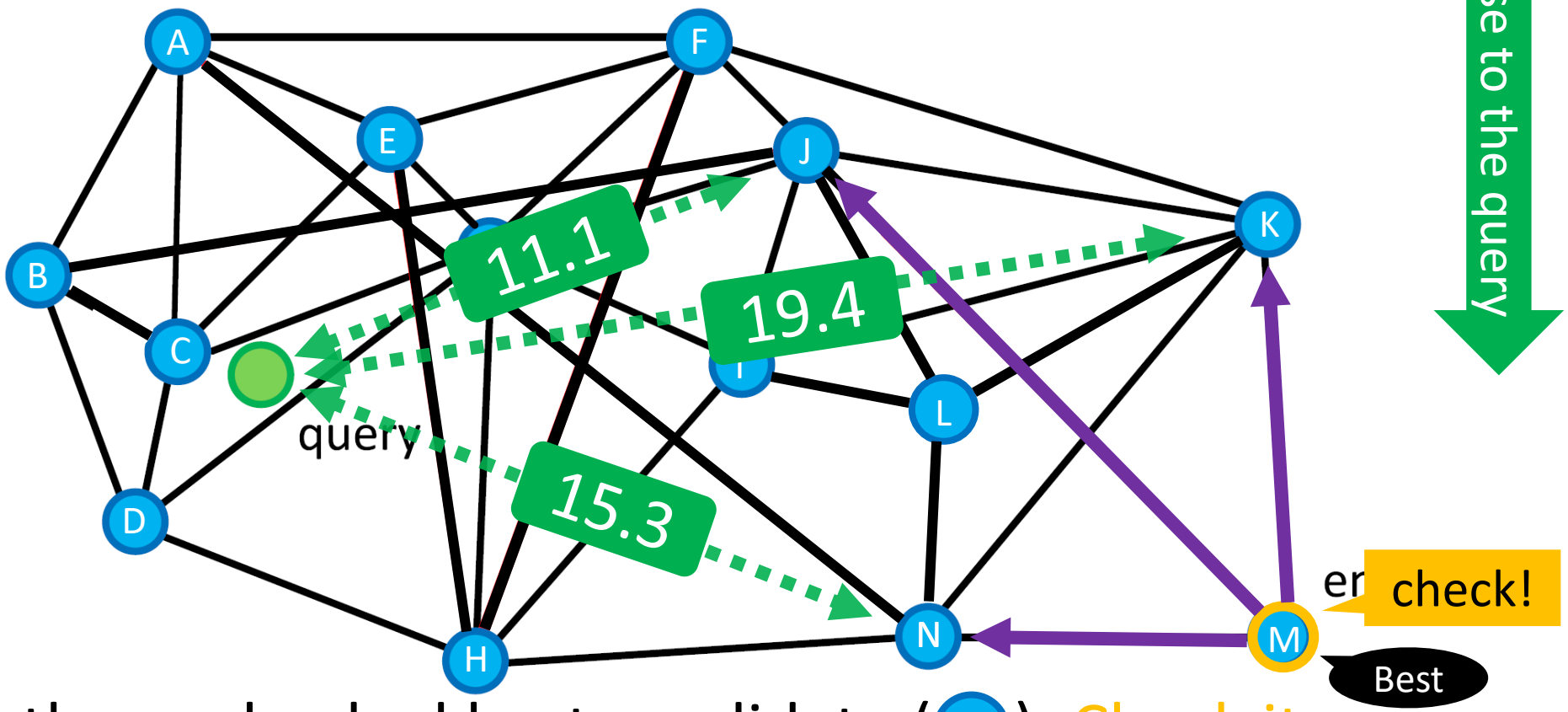
Images are from [Malkov+, Information Systems, 2013]

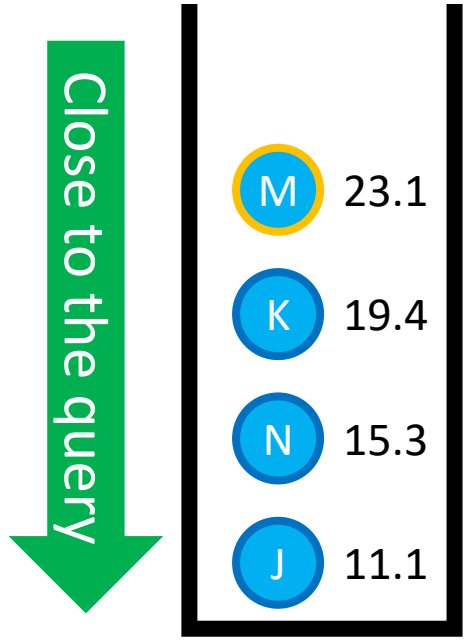


- Pick up the unchecked best candidate (M). Check it.
- Find the connected points.
- Record the distances to q.

Search

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

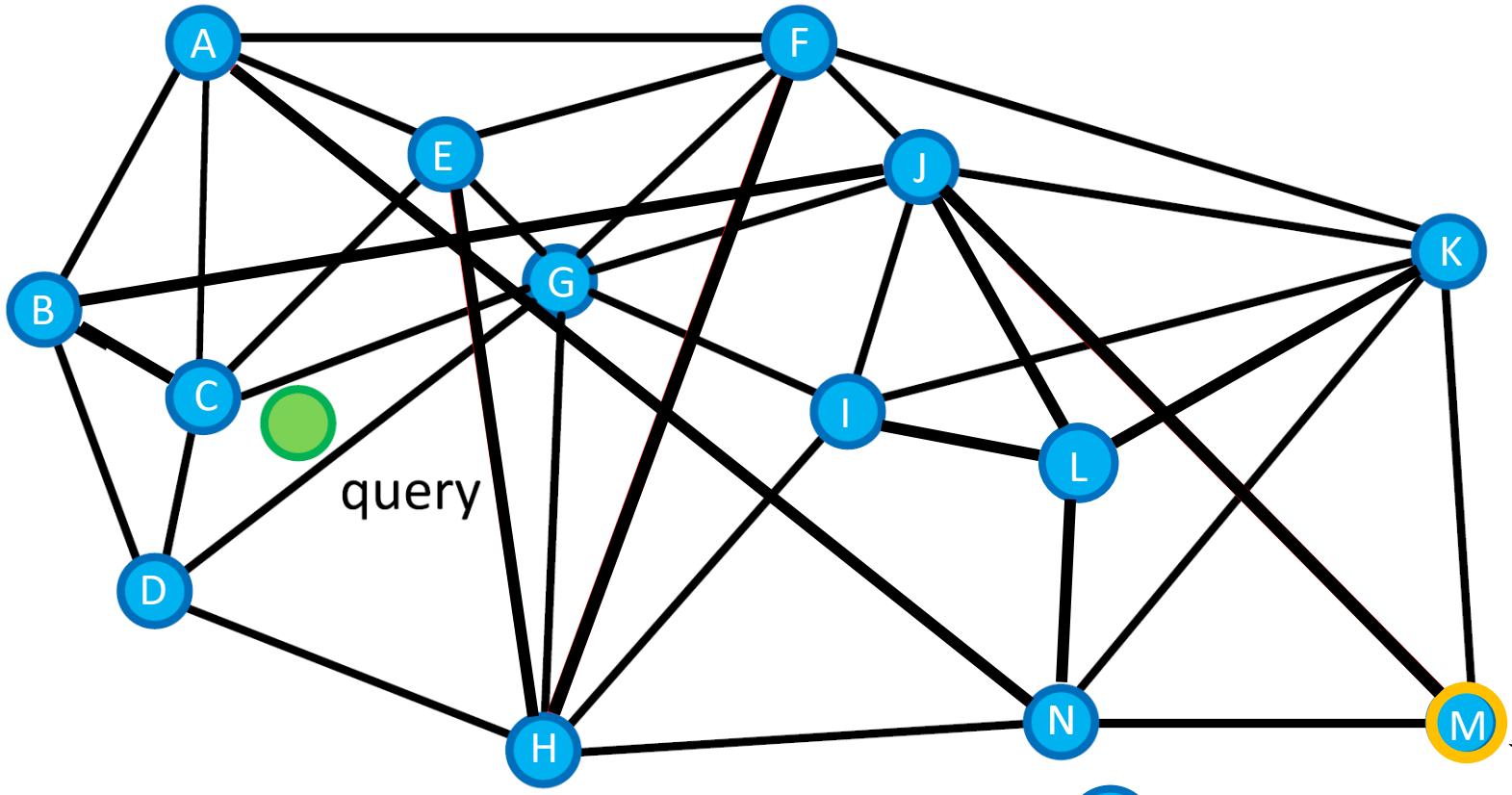


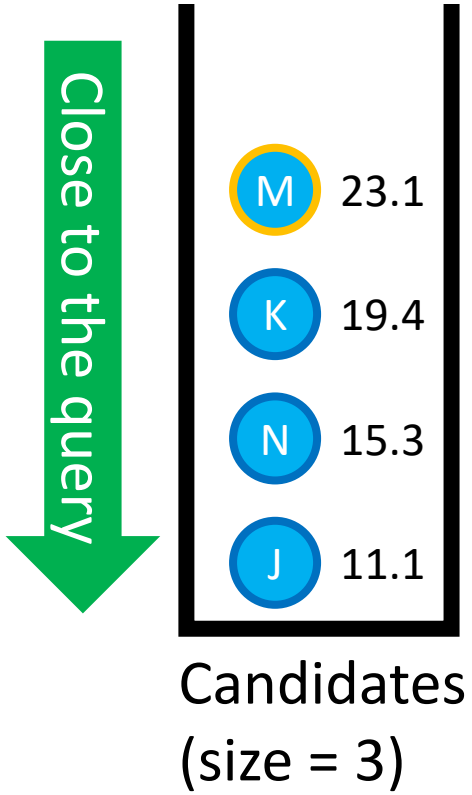


- Pick up the unchecked best candidate (M). Check it.
- Find the connected points.
- Record the distances to q.

Search

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

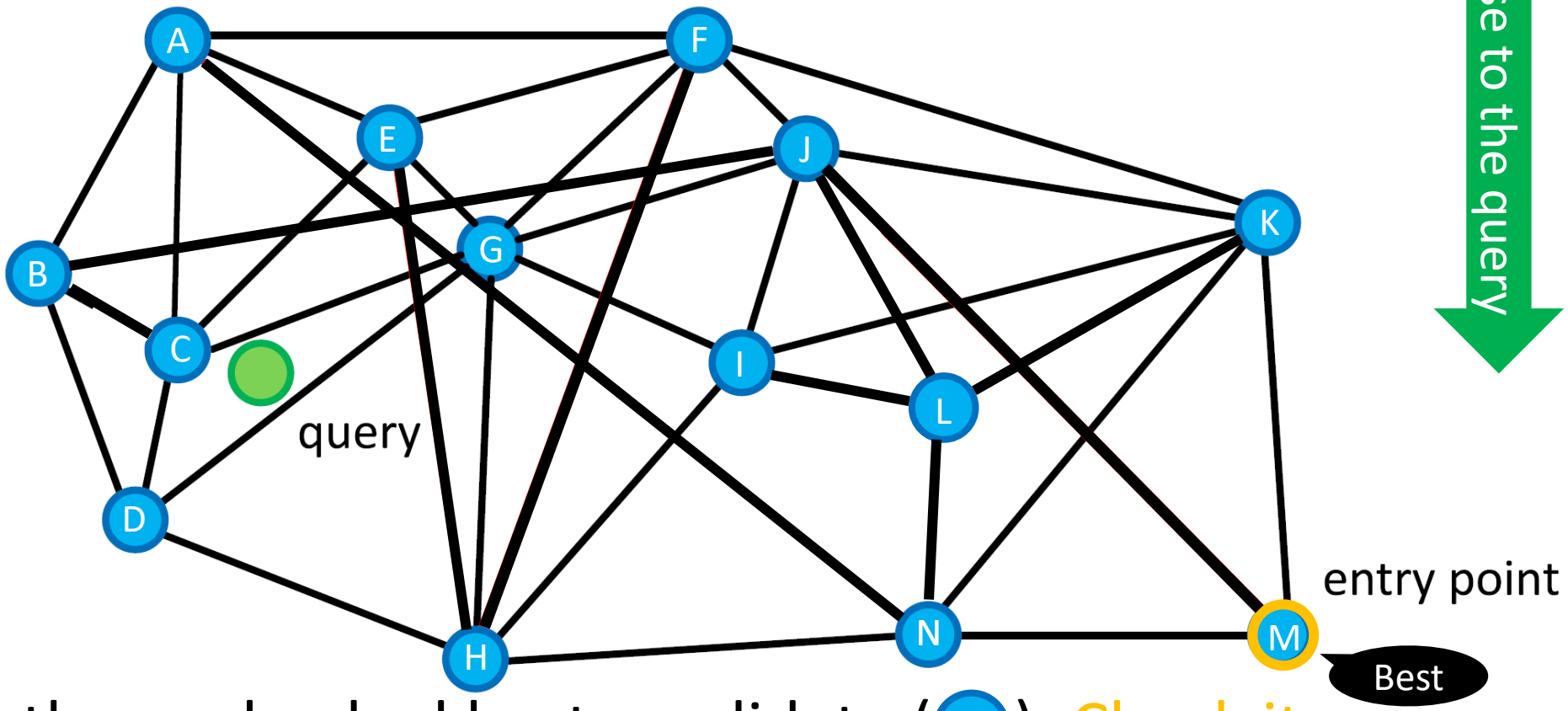




- Pick up the unchecked best candidate (M). Check it.
- Find the connected points.
- Record the distances to q.

Search

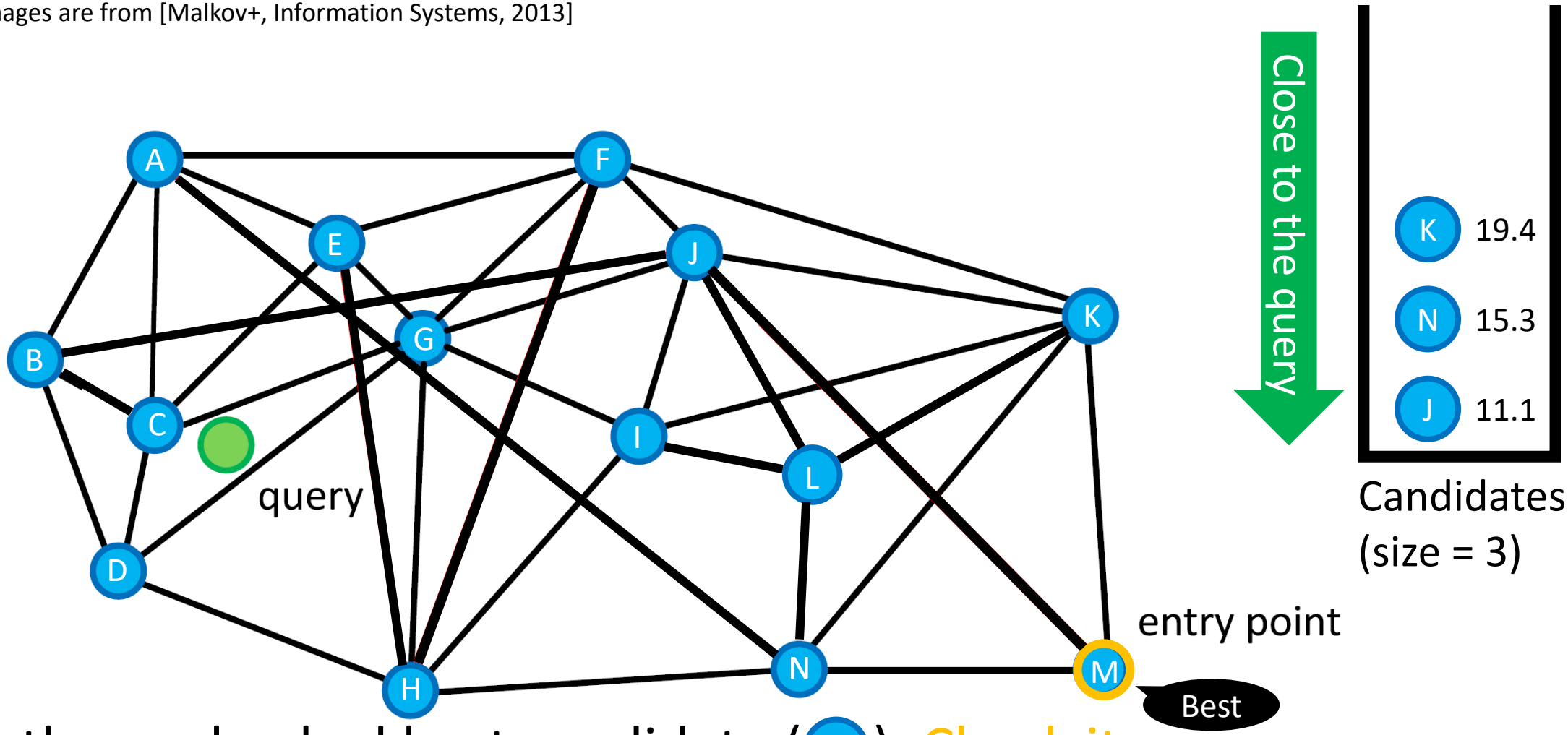
Images are from [Malkov+, Information Systems, 2013]



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

Search

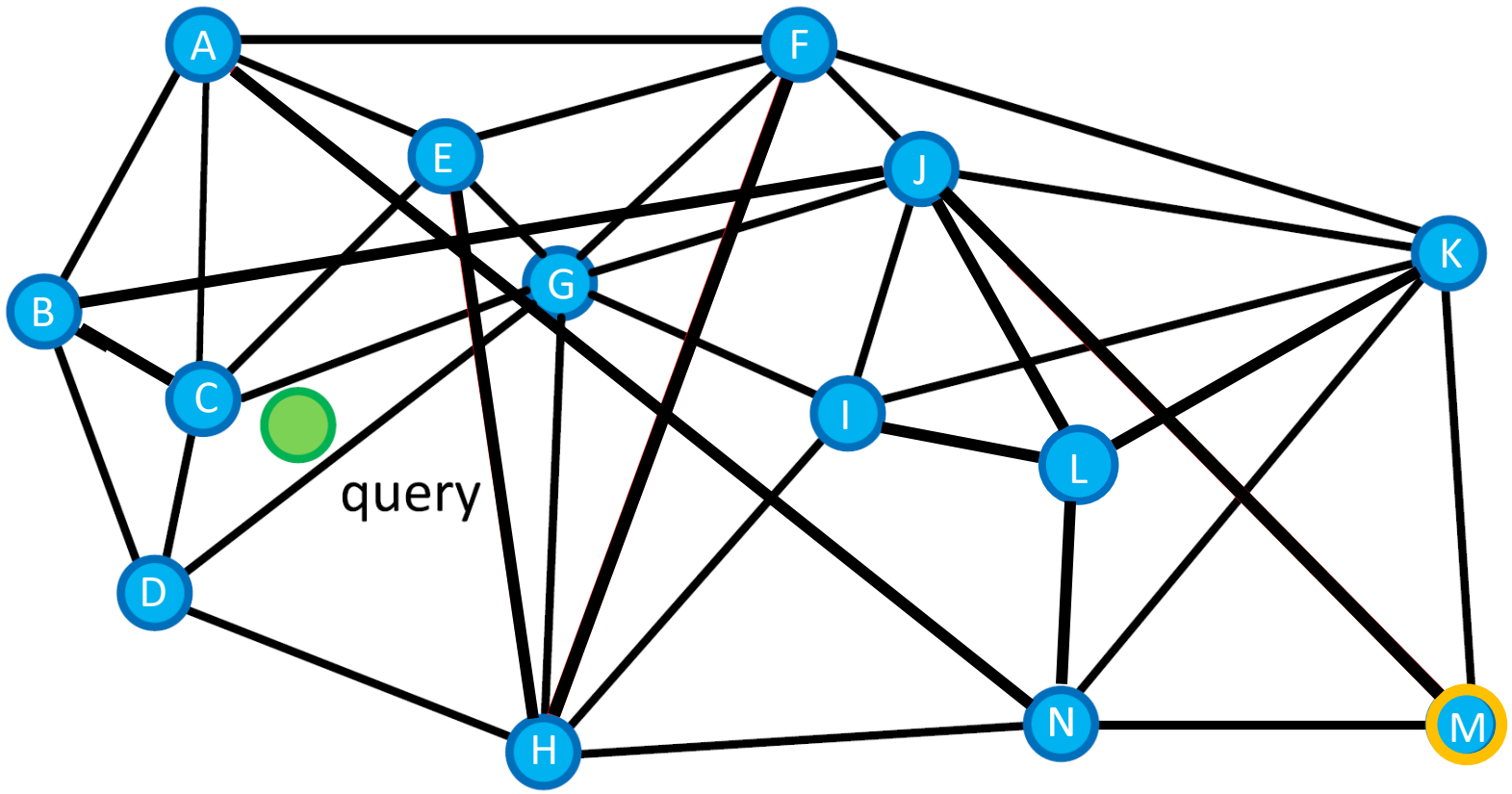
Images are from [Malkov+, Information Systems, 2013]

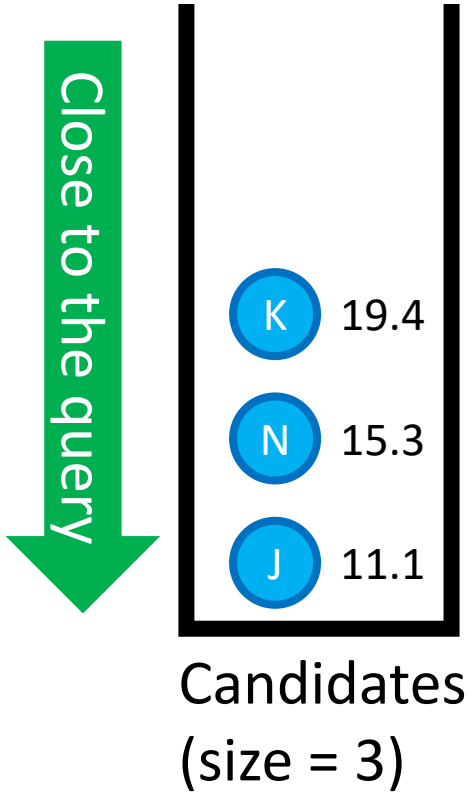


- Pick up the unchecked best candidate (M). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

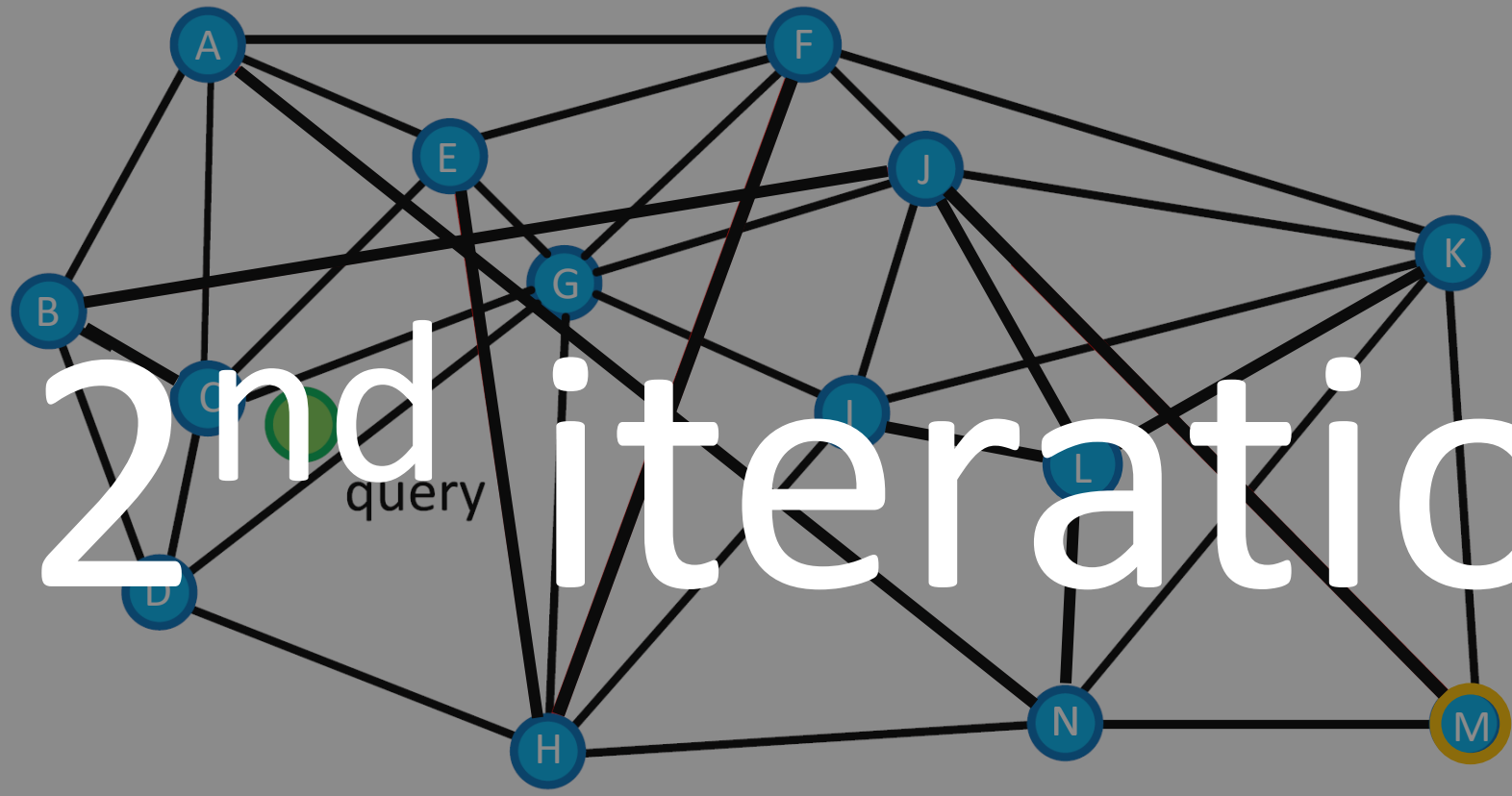
Search

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



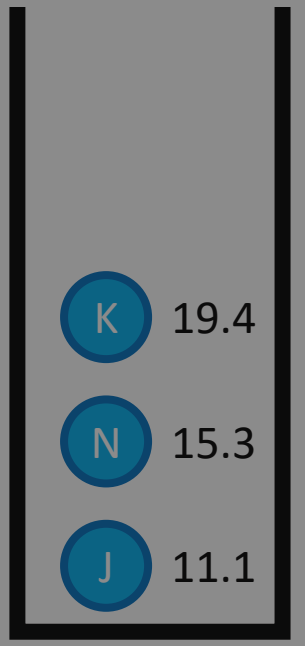


Images are from [Malkov+, Information Systems, 2013]



2nd iteration

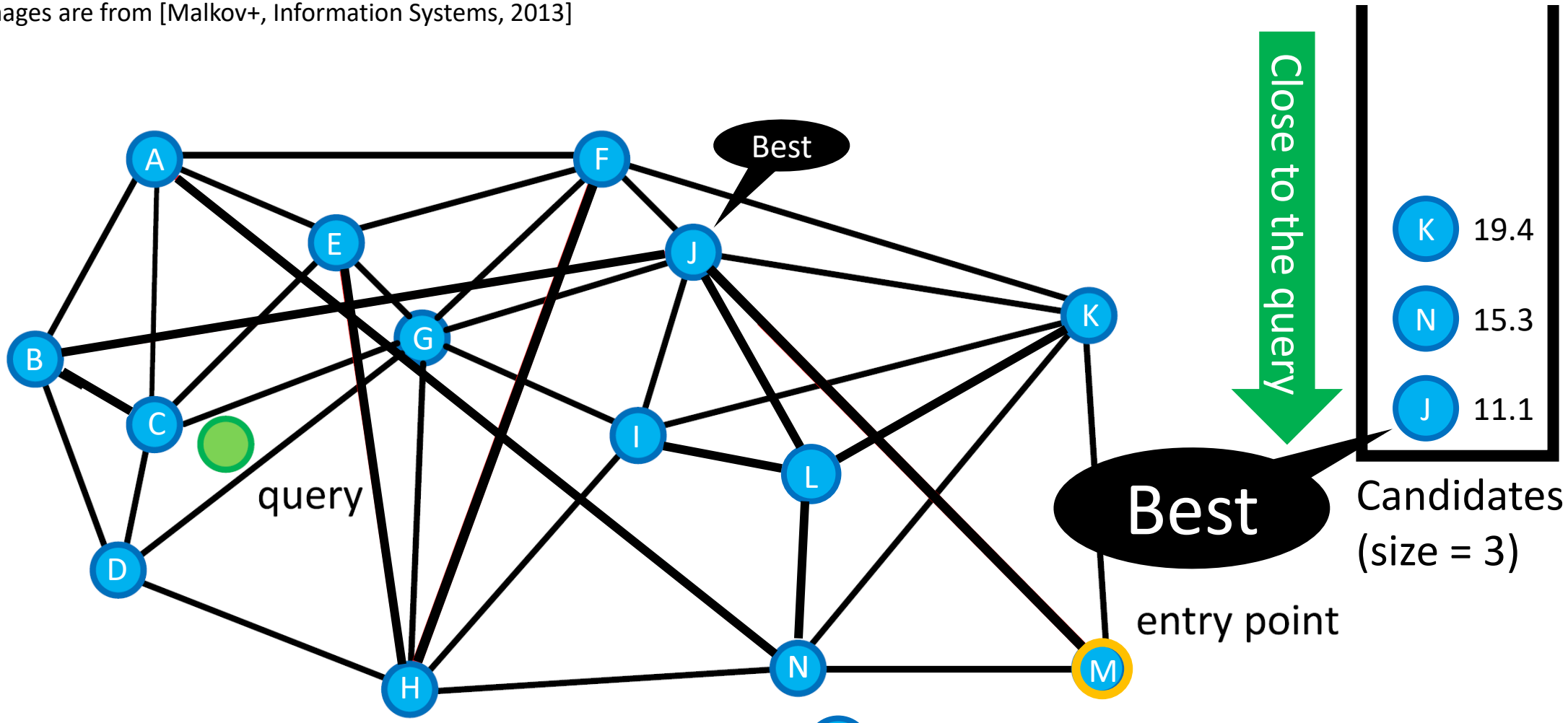
Close to the query



Candidates
(size = 3)

Search

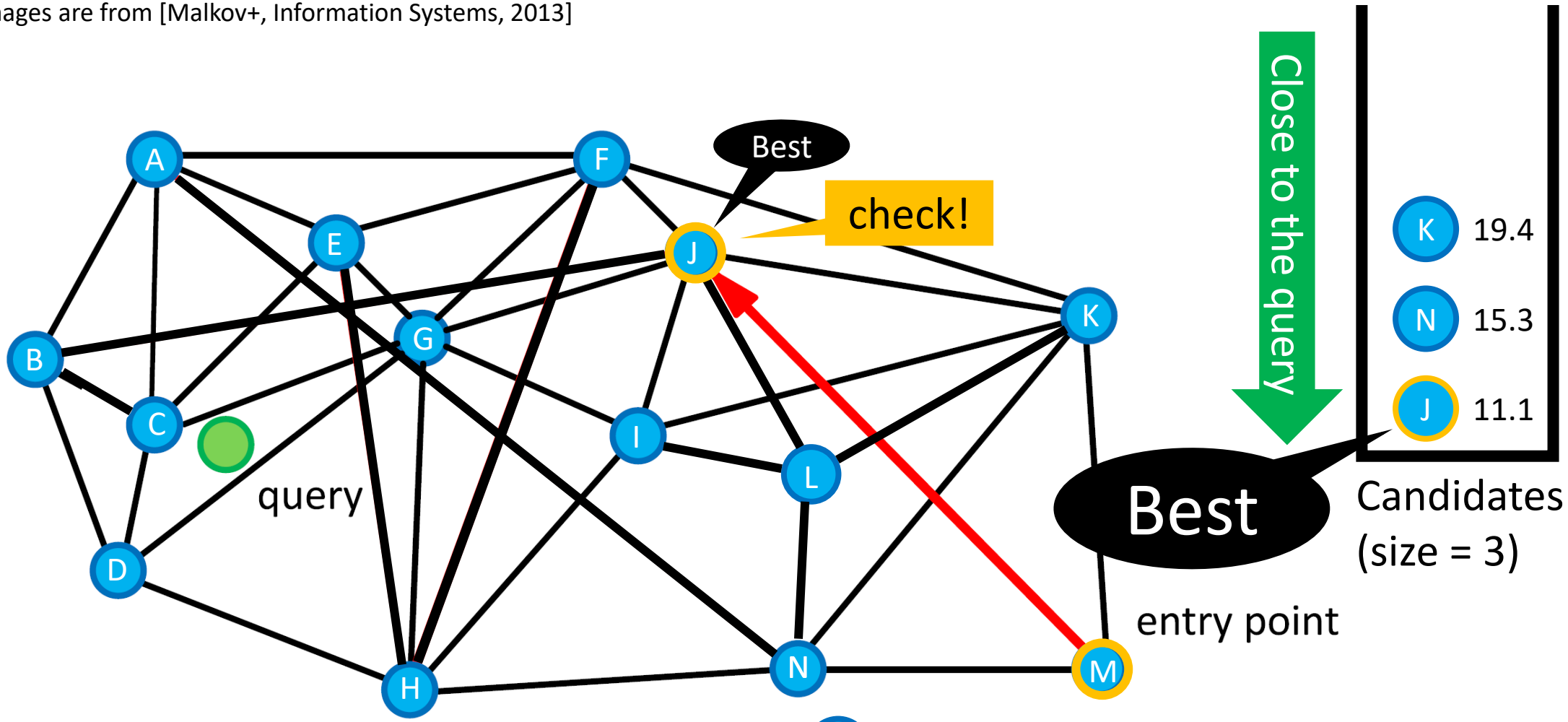
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (J)

Search

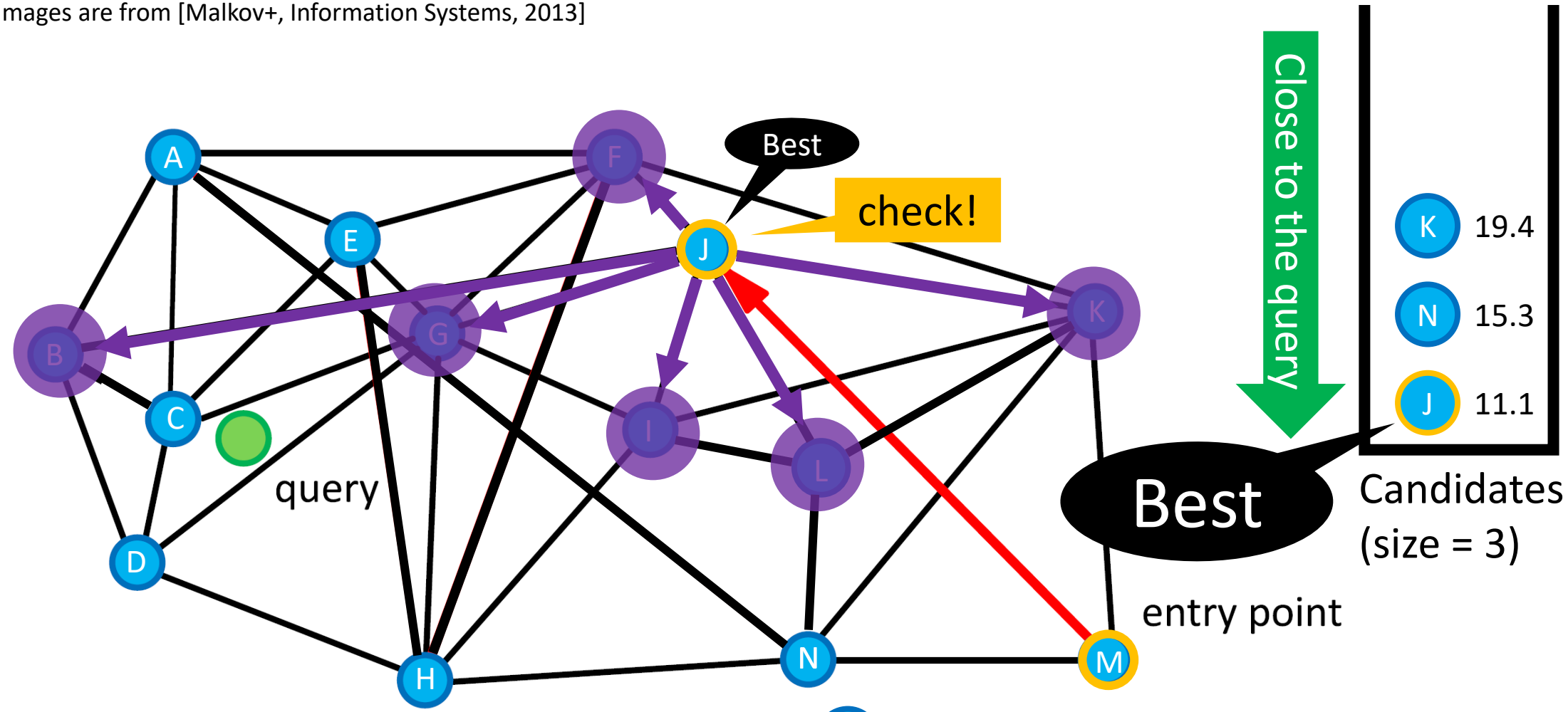
Images are from [Malkov+, Information Systems, 2013]



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

Search

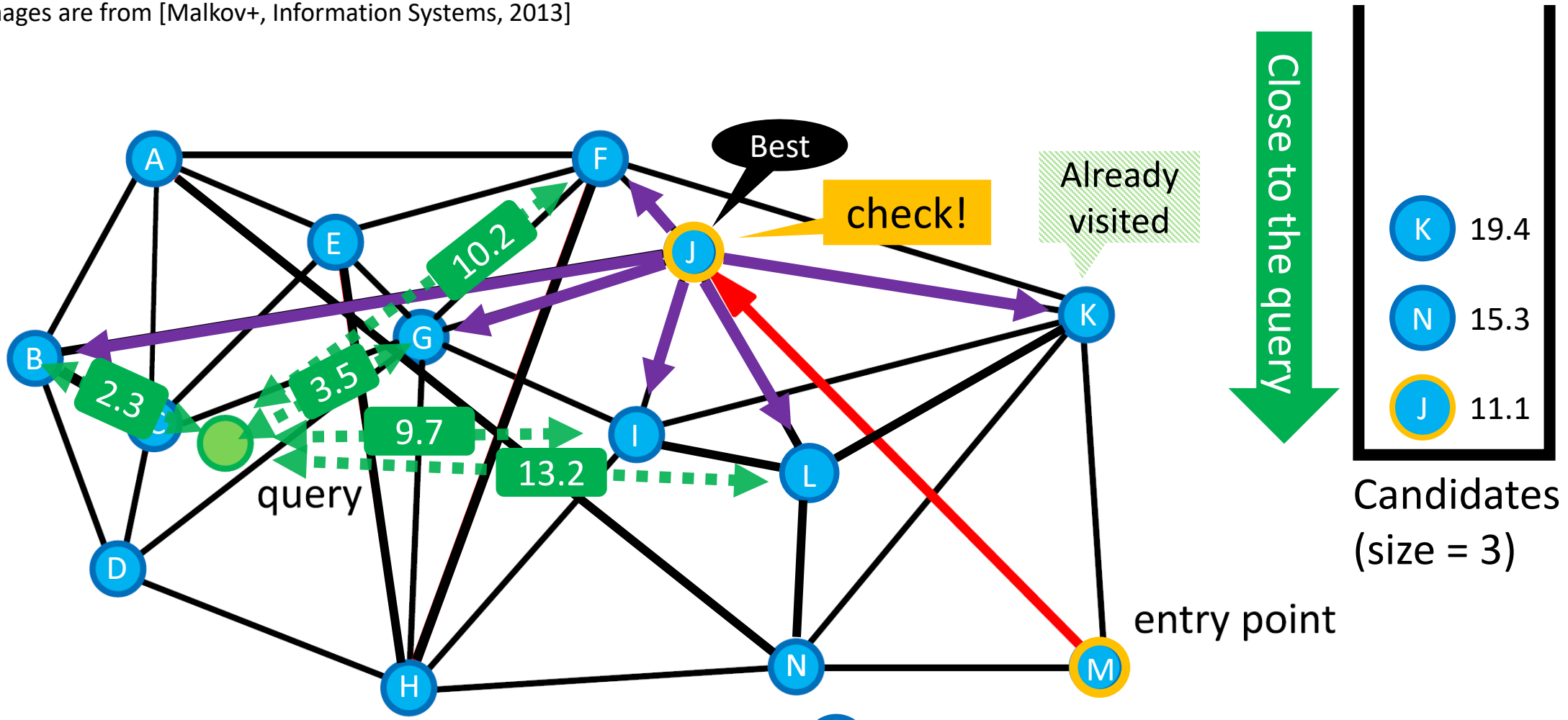
Images are from [Malkov+, Information Systems, 2013]



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

Search

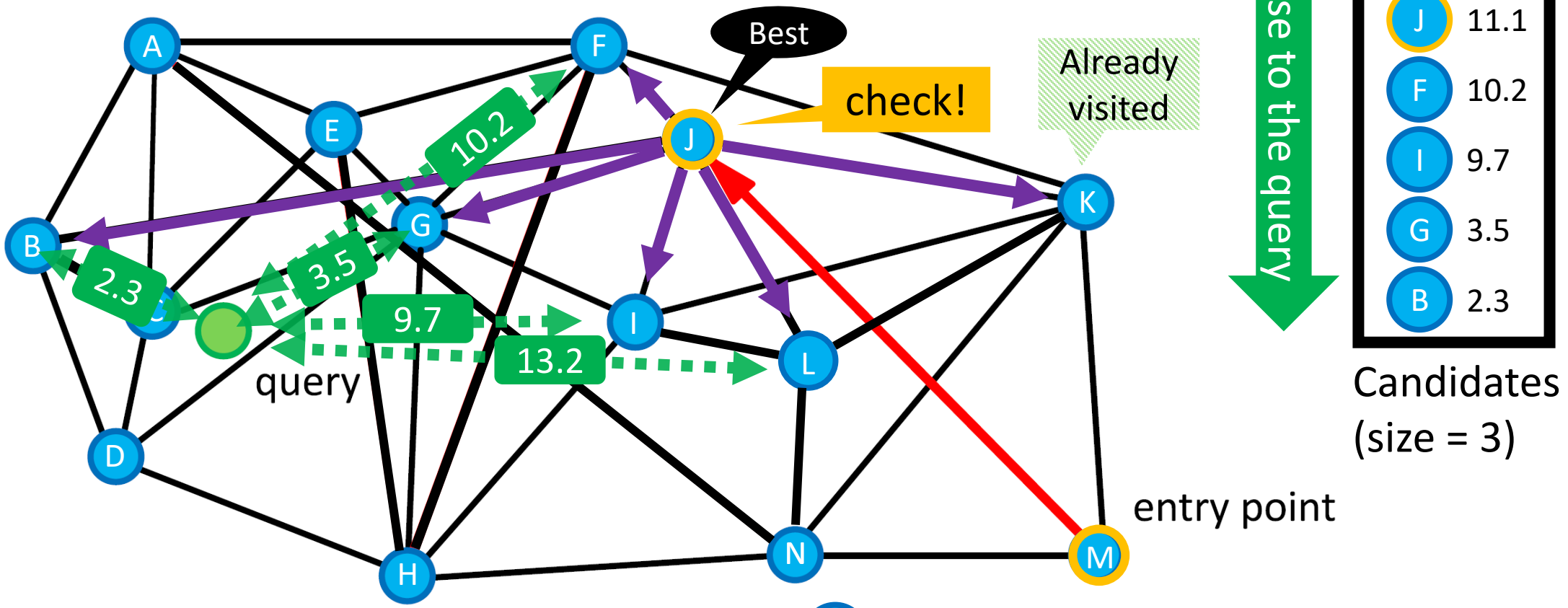
Images are from [Malkov+, Information Systems, 2013]



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

Search

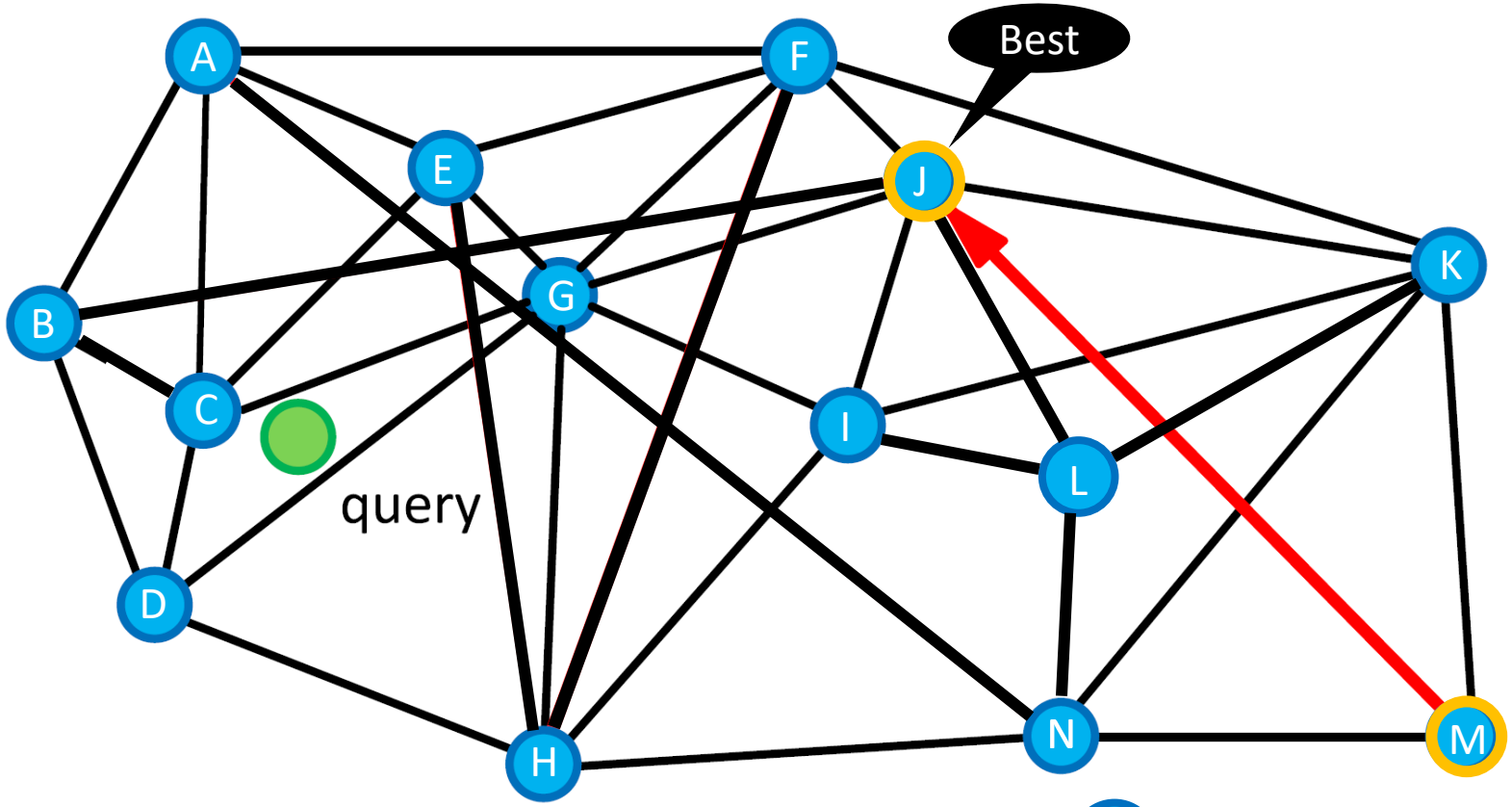
Images are from [Malkov+, Information Systems, 2013]

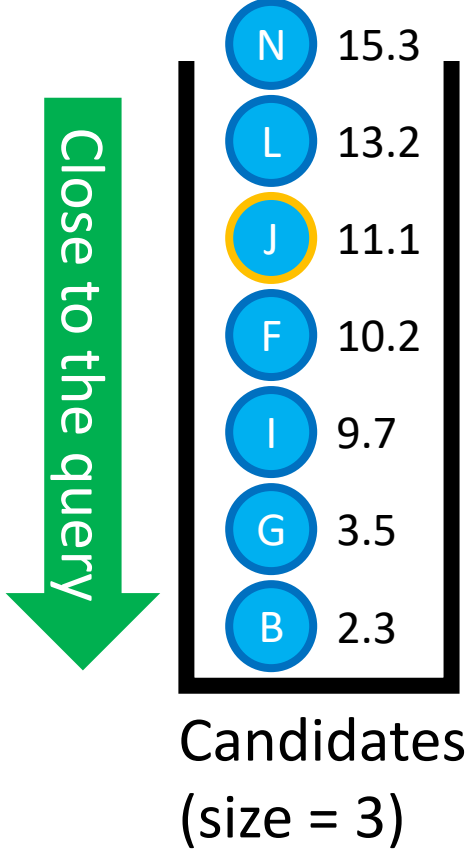


- Pick up the unchecked best candidate (J). Check it.
- Find the connected points.
- Record the distances to q.

Search

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

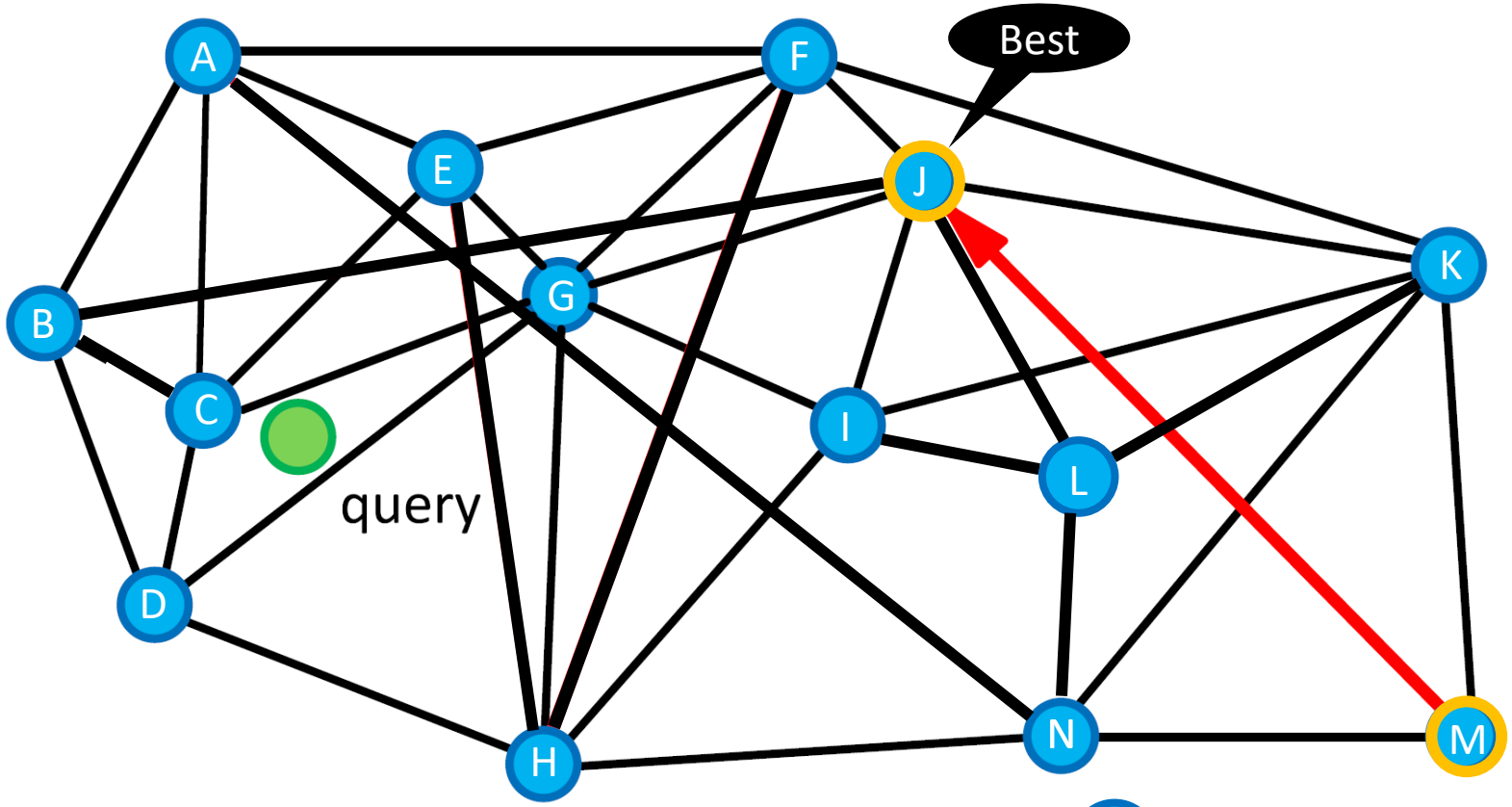


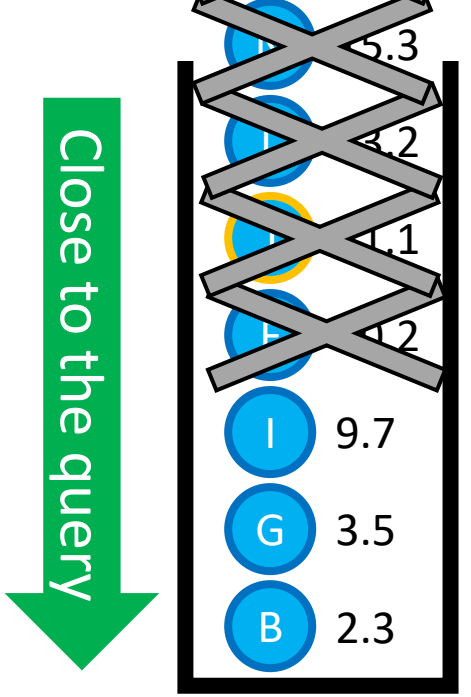


- Pick up the unchecked best candidate (J). Check it.
- Find the connected points.
- Record the distances to q.

Search

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



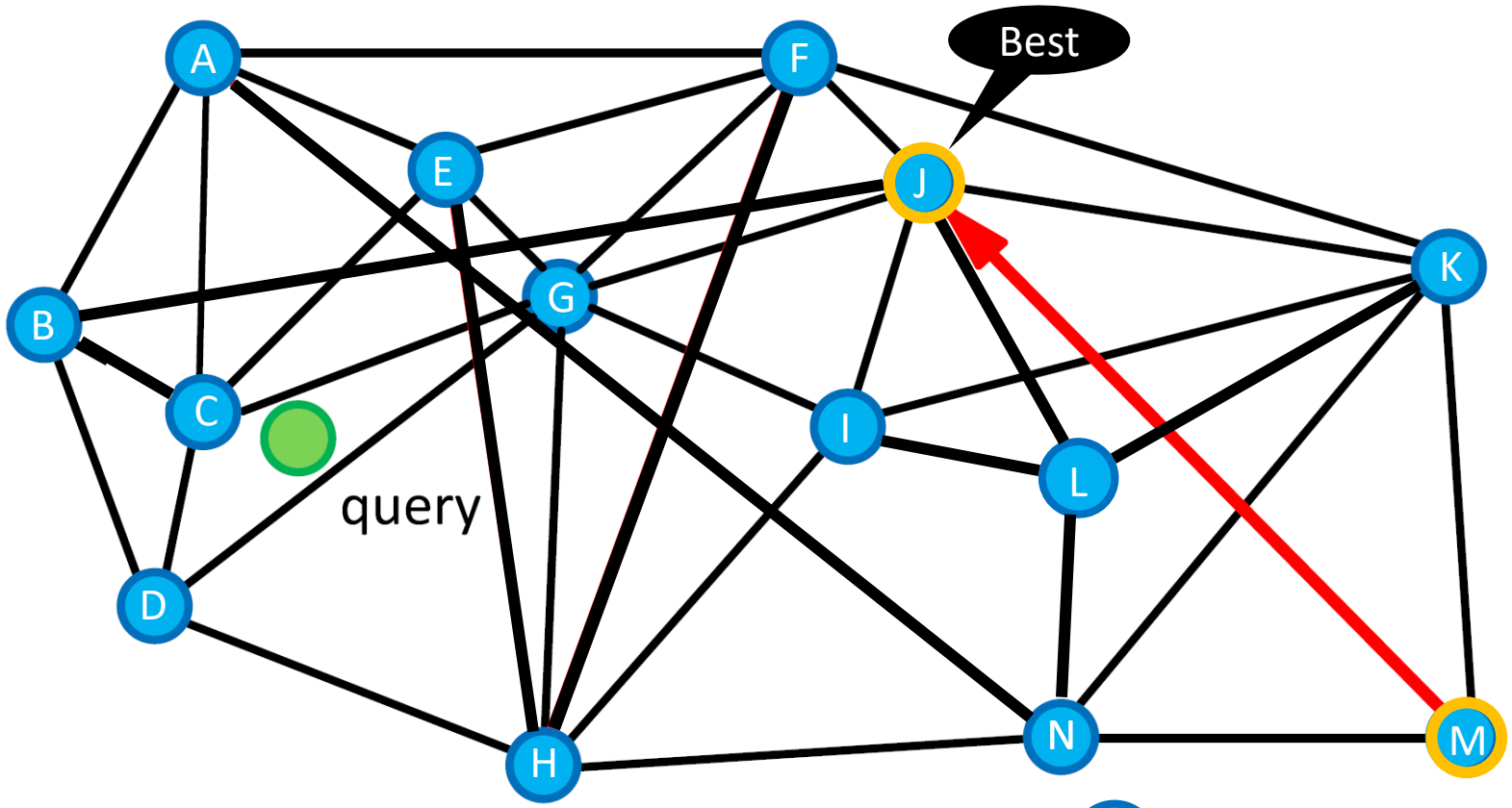


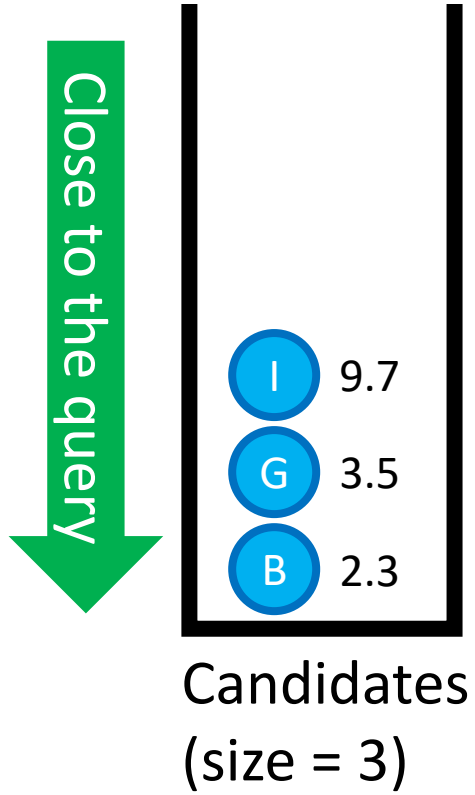
Candidates (size = 3)

- 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]

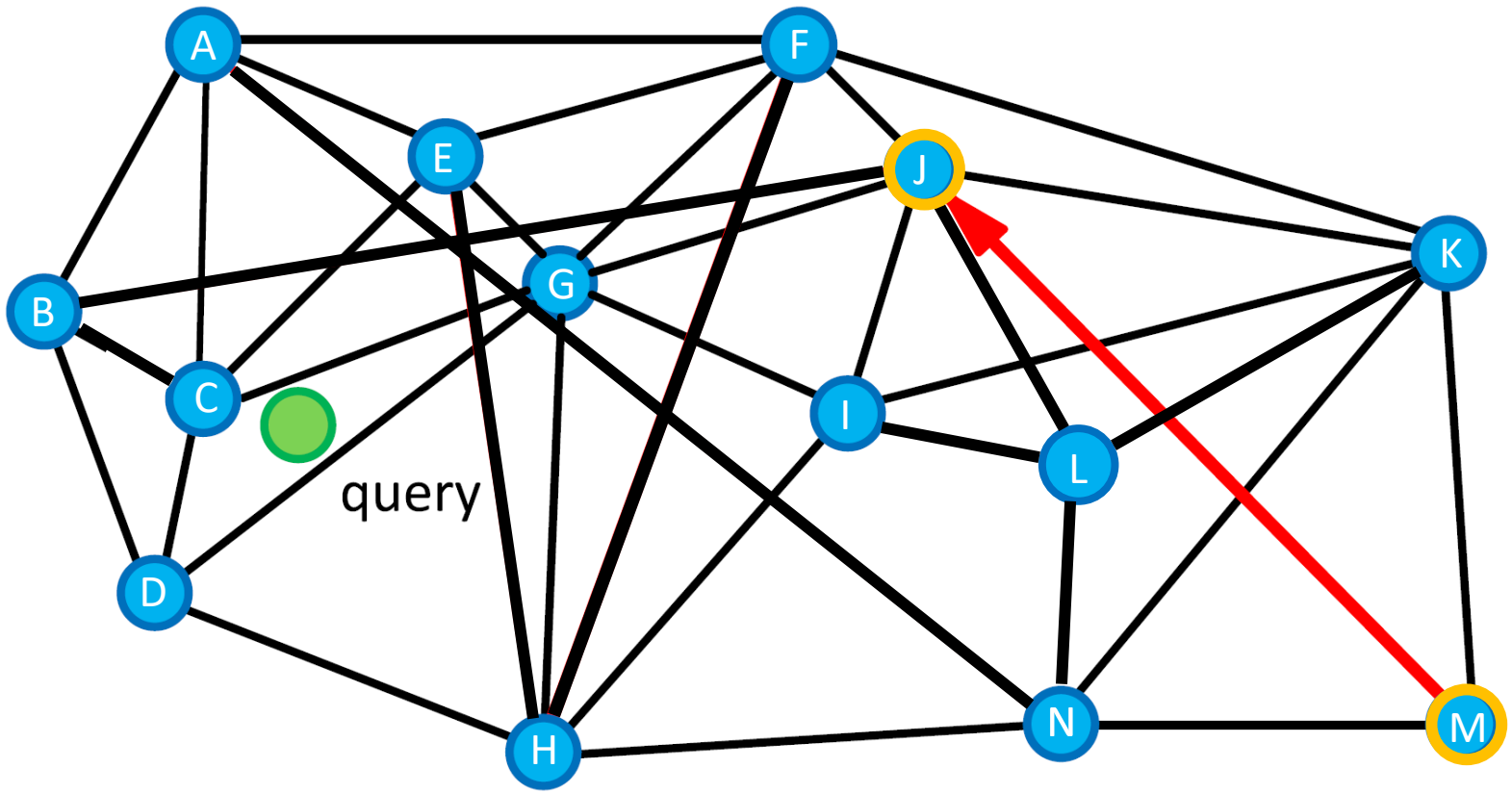


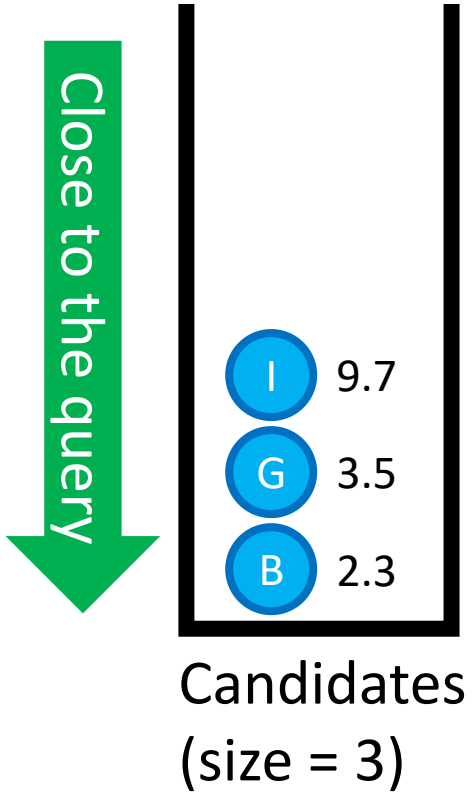


- Pick up the unchecked best candidate (J). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

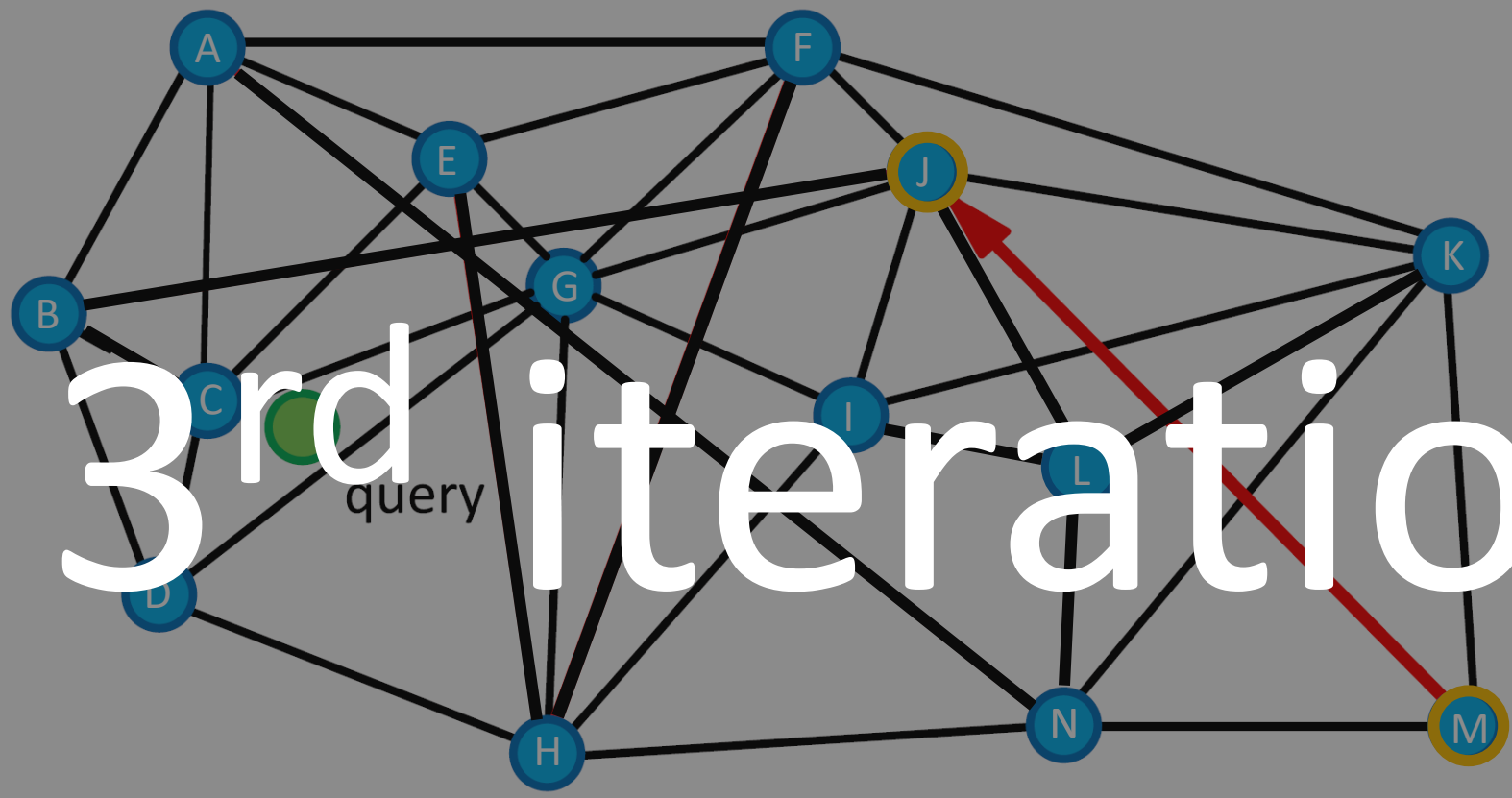
Search

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



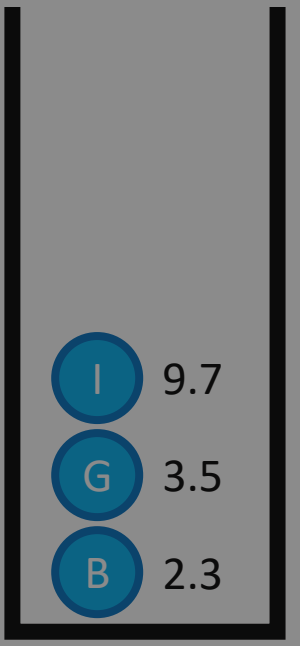


Images are from [Malkov+, Information Systems, 2013]



3rd iteration

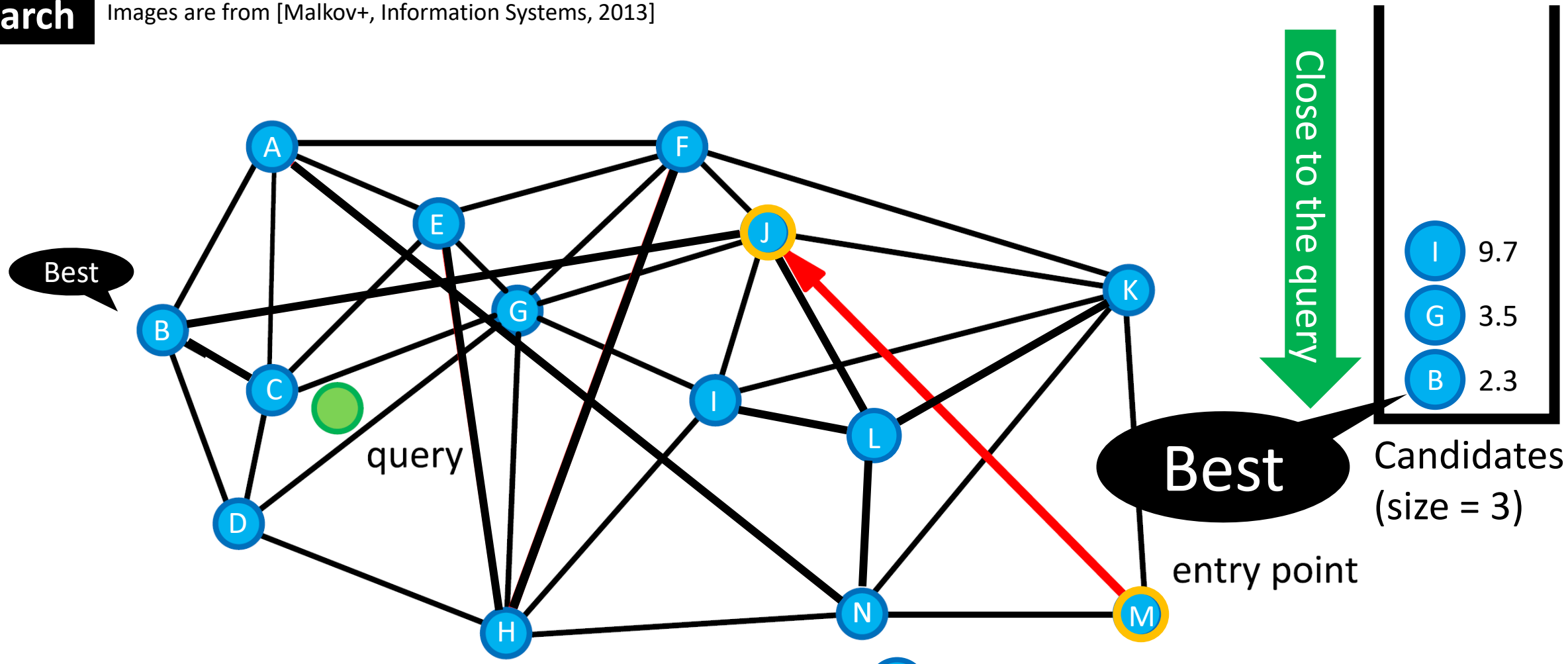
Close to the query



Candidates
(size = 3)

Search

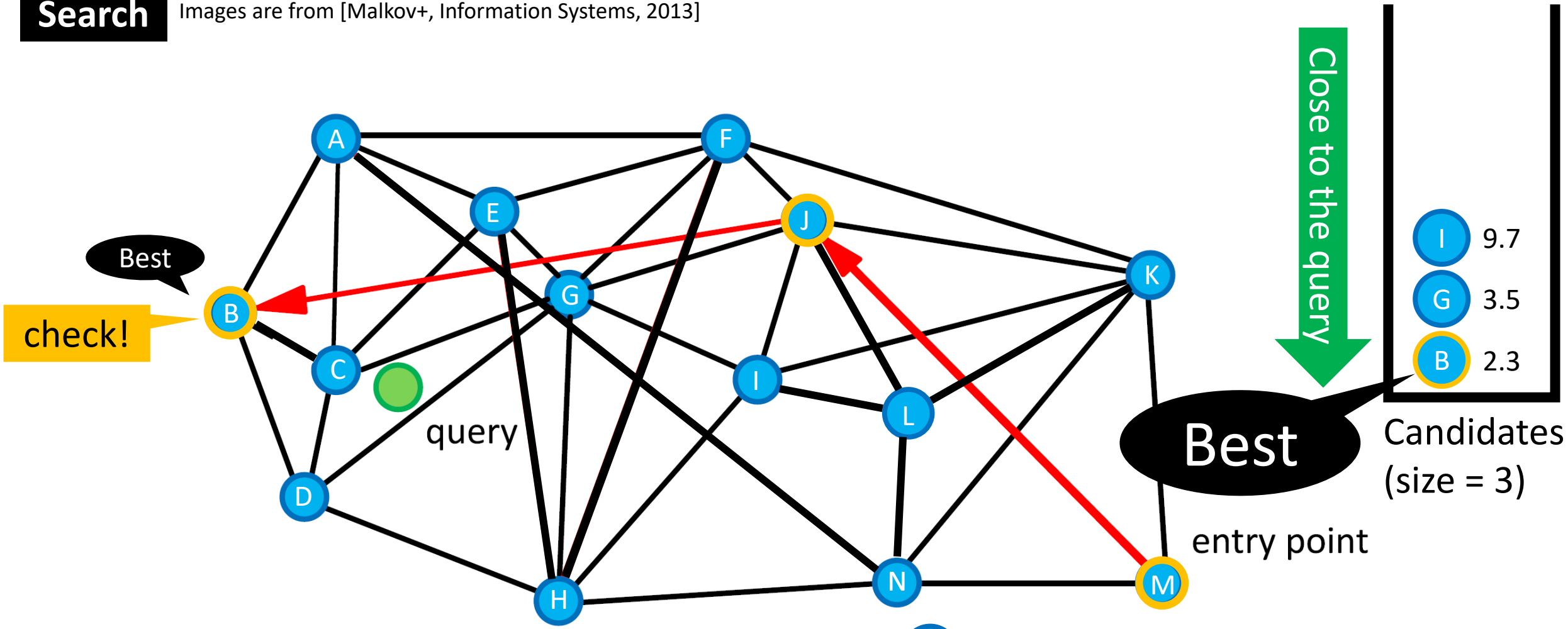
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (B)

Search

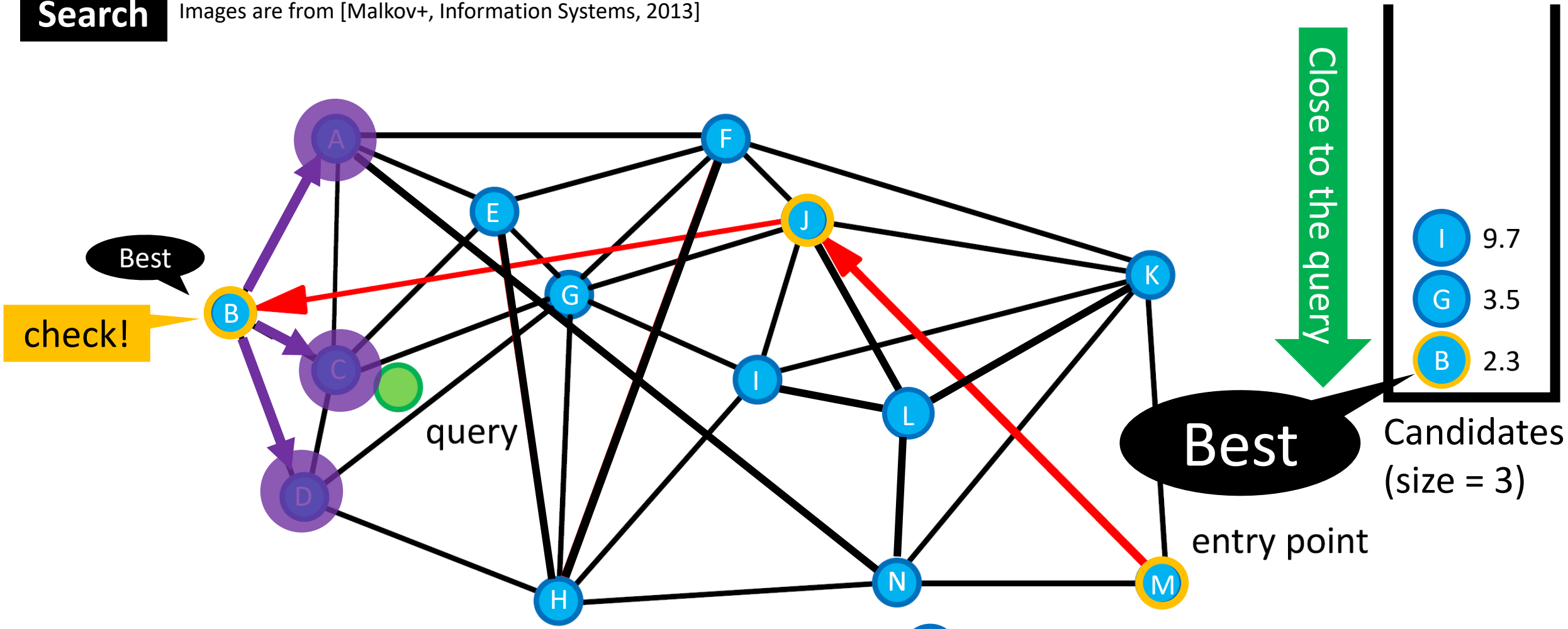
Images are from [Malkov+, Information Systems, 2013]



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

Search

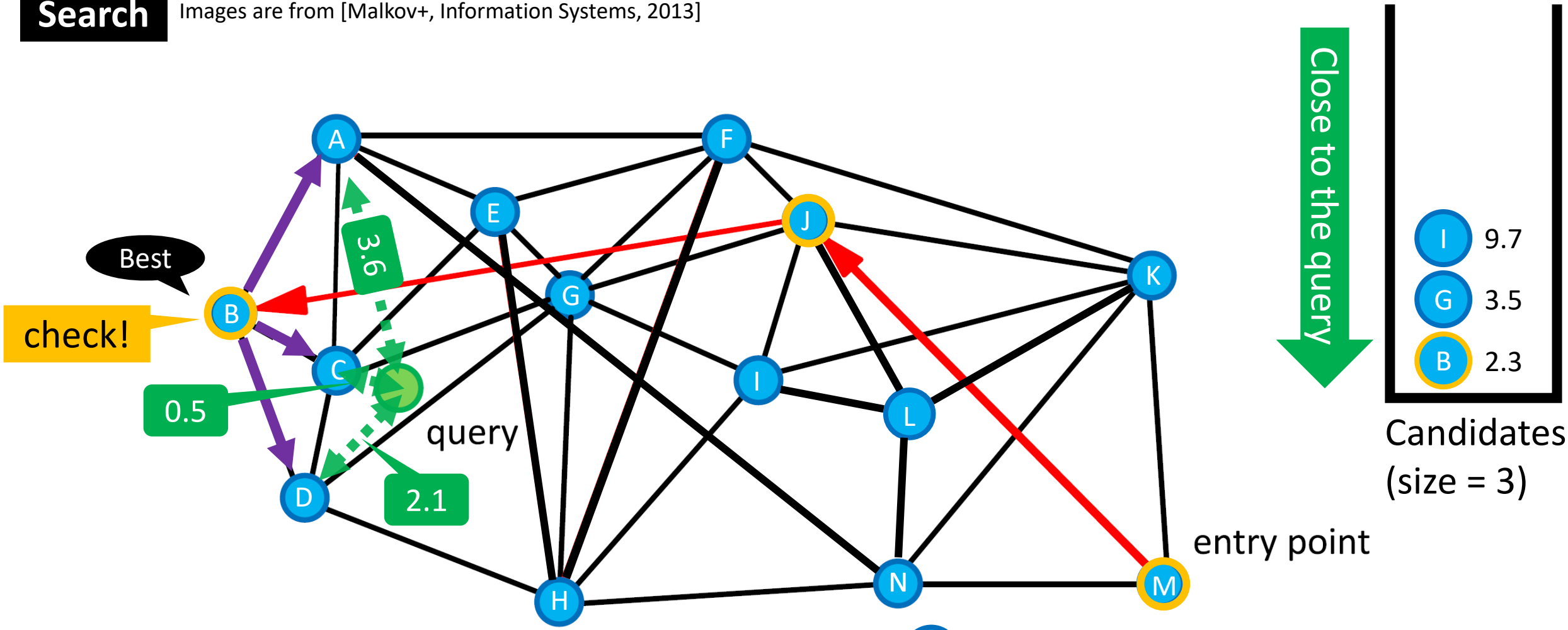
Images are from [Malkov+, Information Systems, 2013]



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

Search

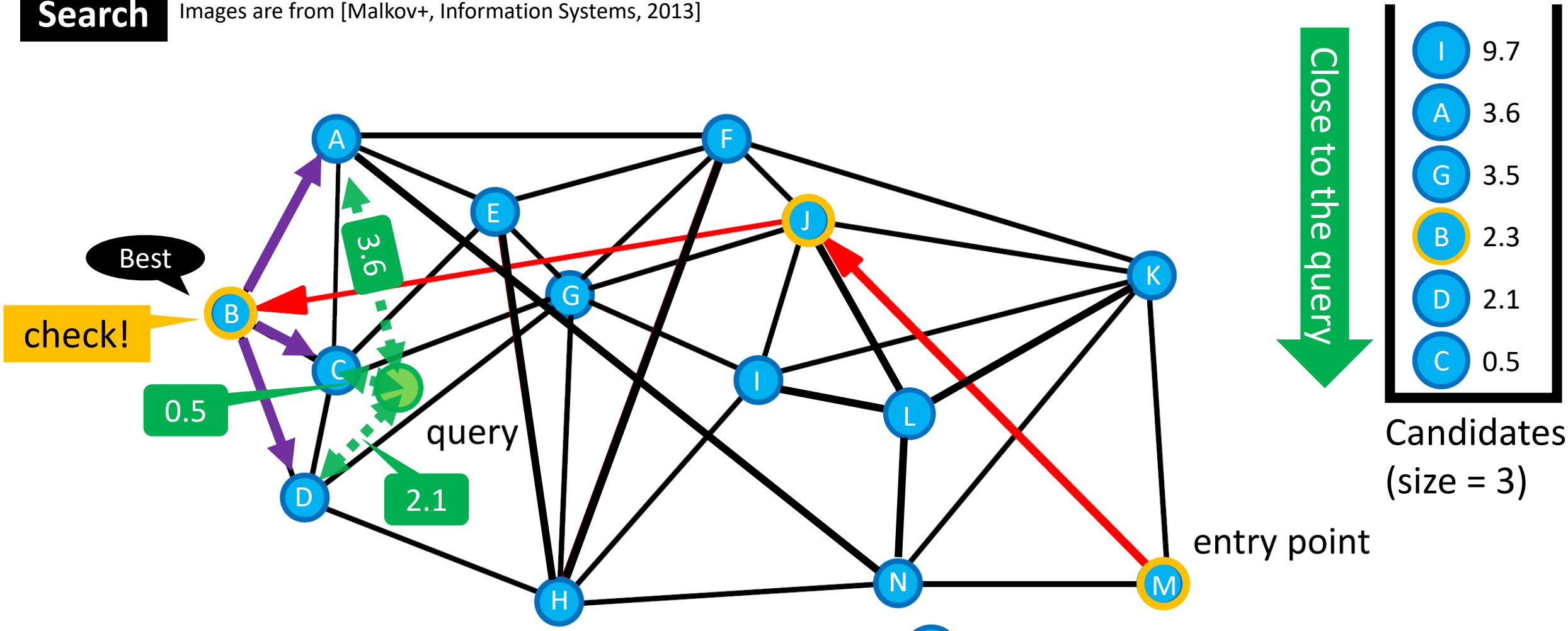
Images are from [Malkov+, Information Systems, 2013]



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

Search

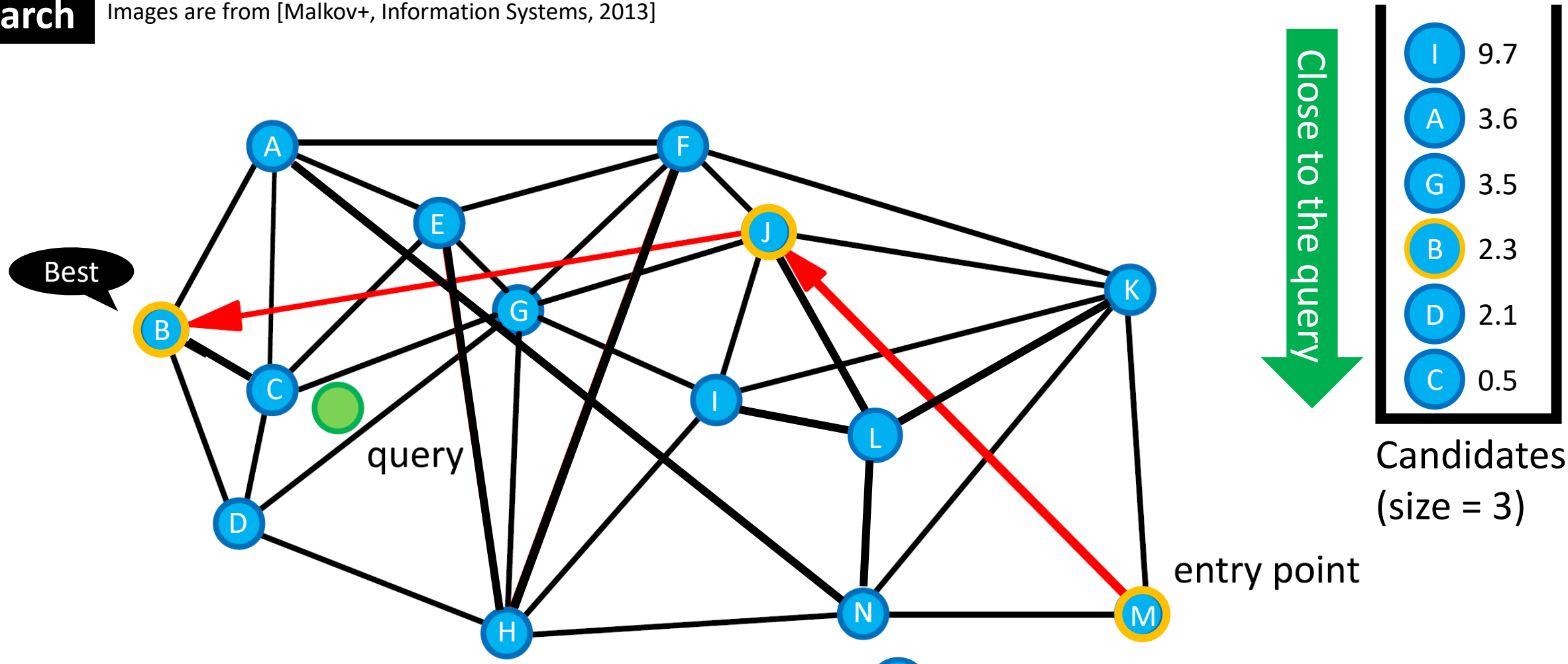
Images are from [Malkov+, Information Systems, 2013]



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

Search

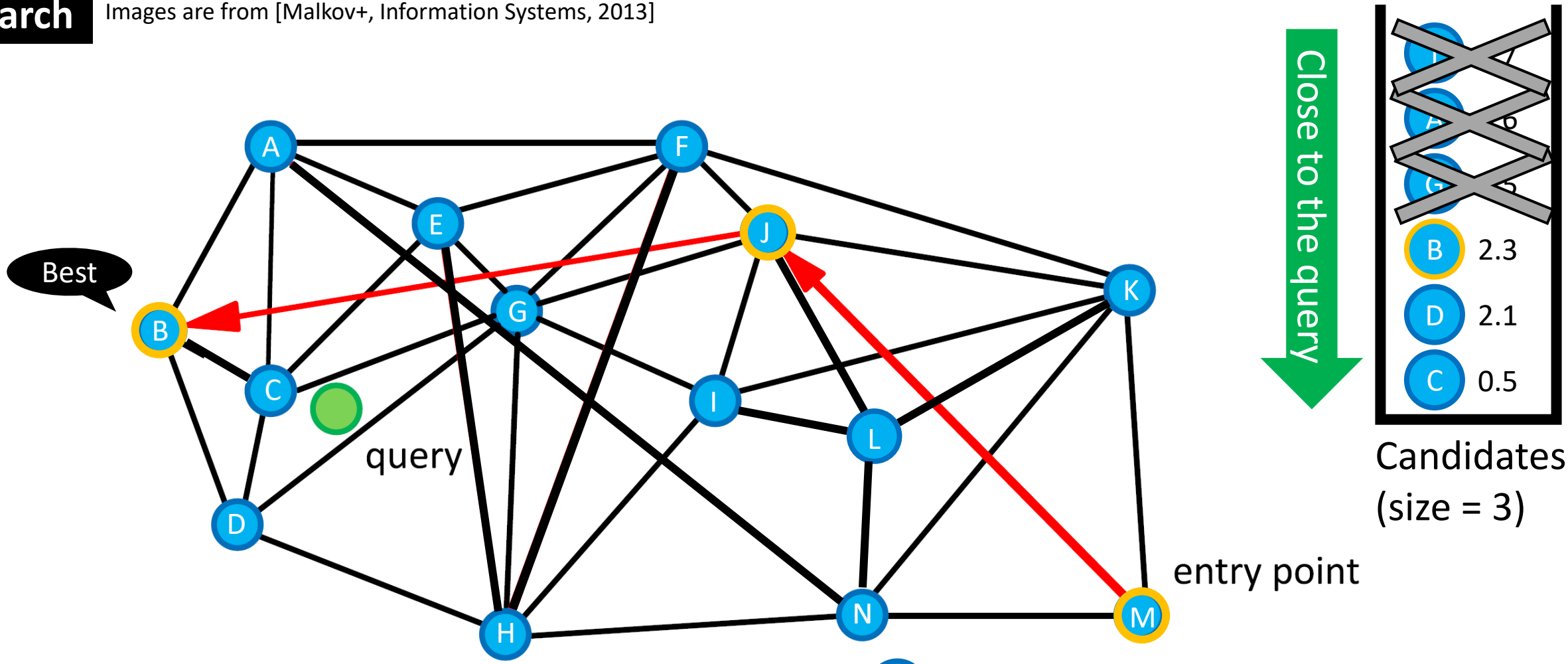
Images are from [Malkov+, Information Systems, 2013]



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

Search

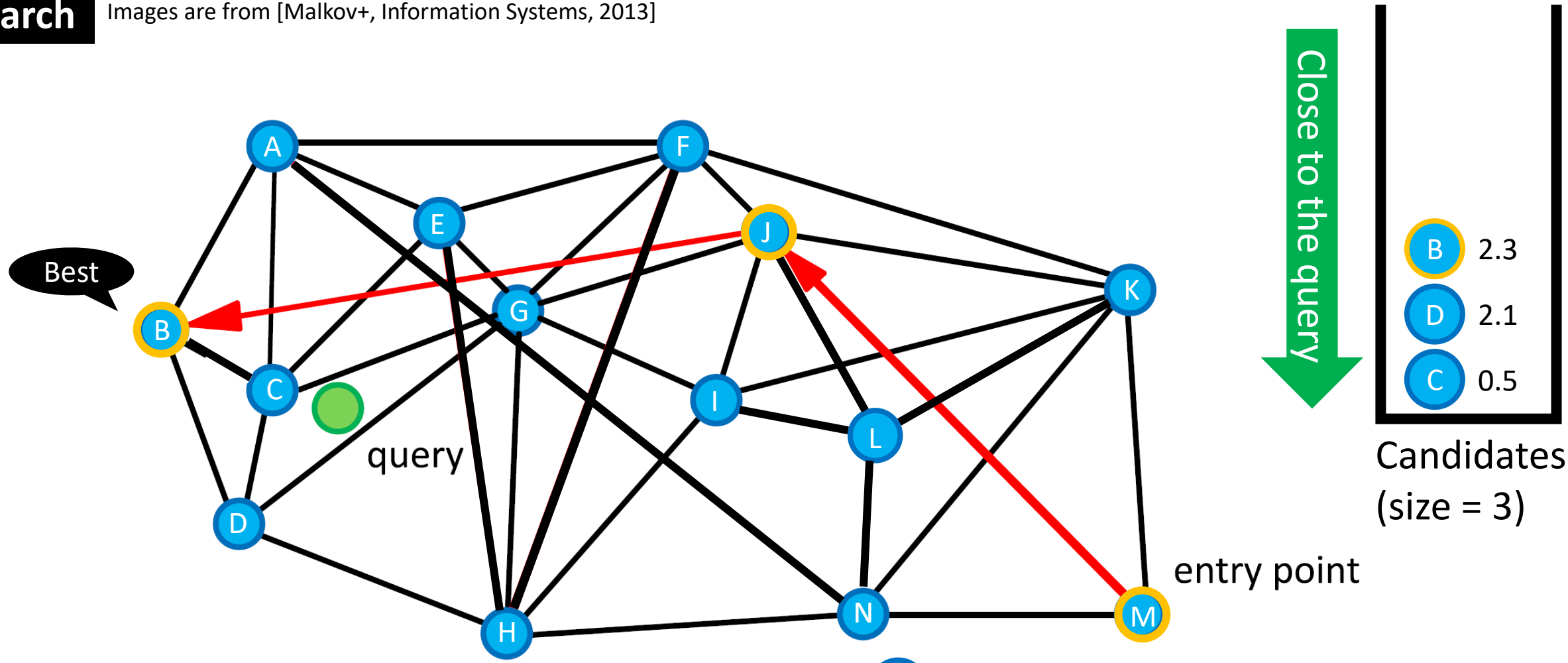
Images are from [Malkov+, Information Systems, 2013]



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

Search

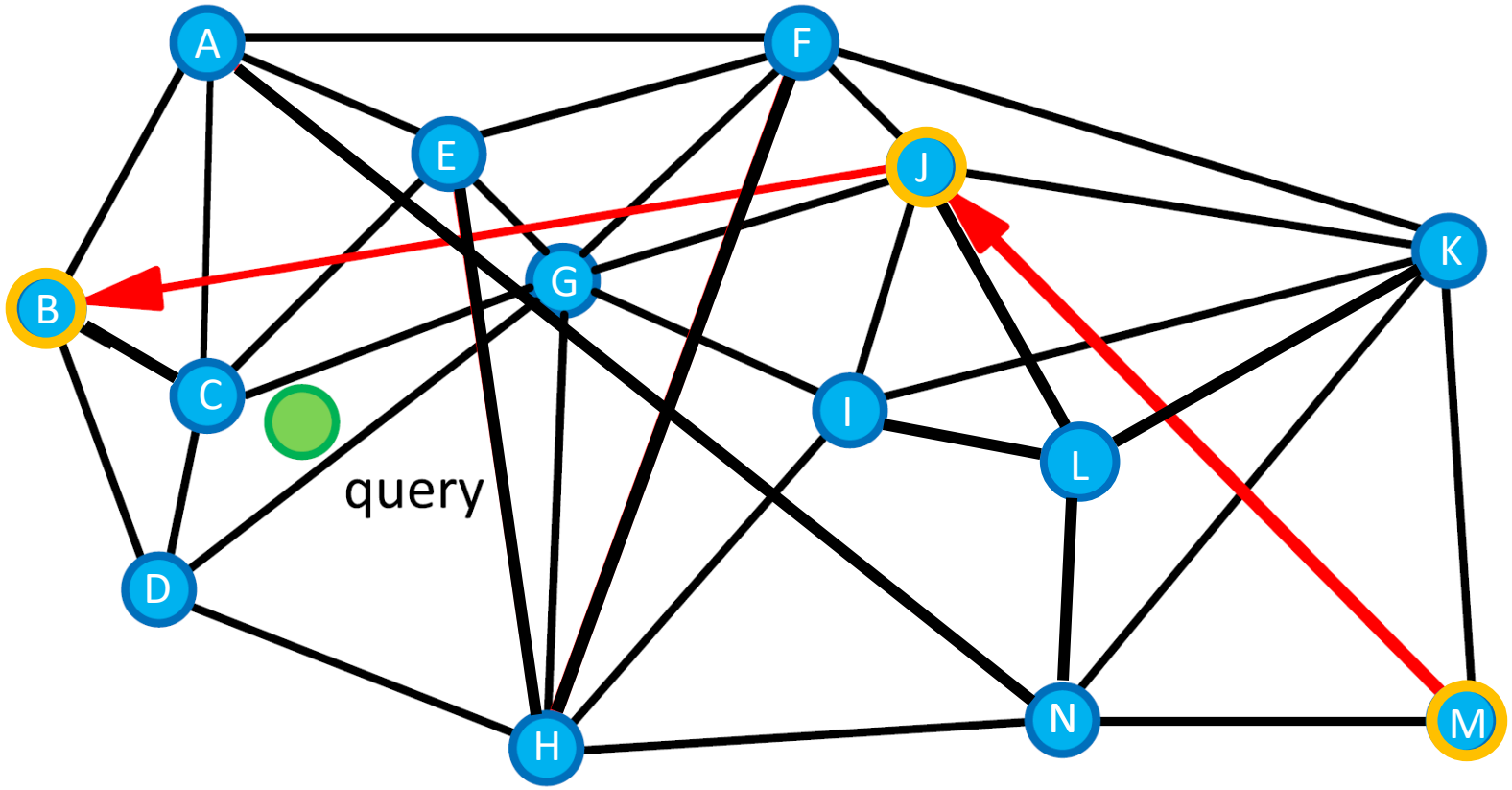
Images are from [Malkov+, Information Systems, 2013]

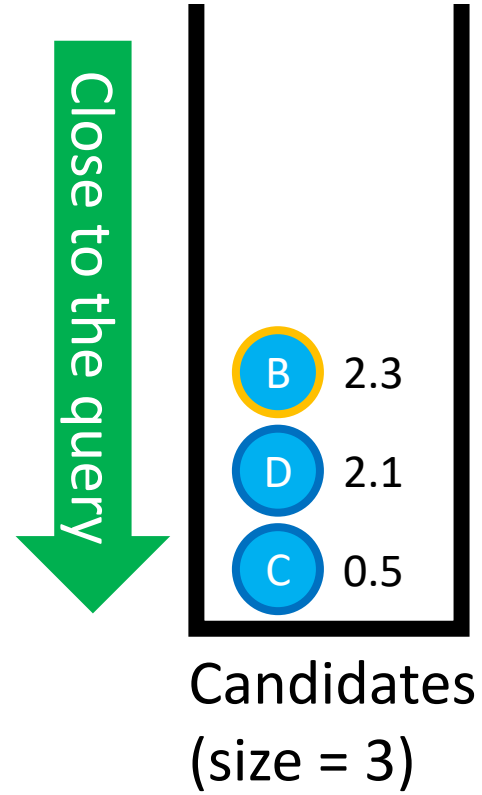


- Pick up the unchecked best candidate (B). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

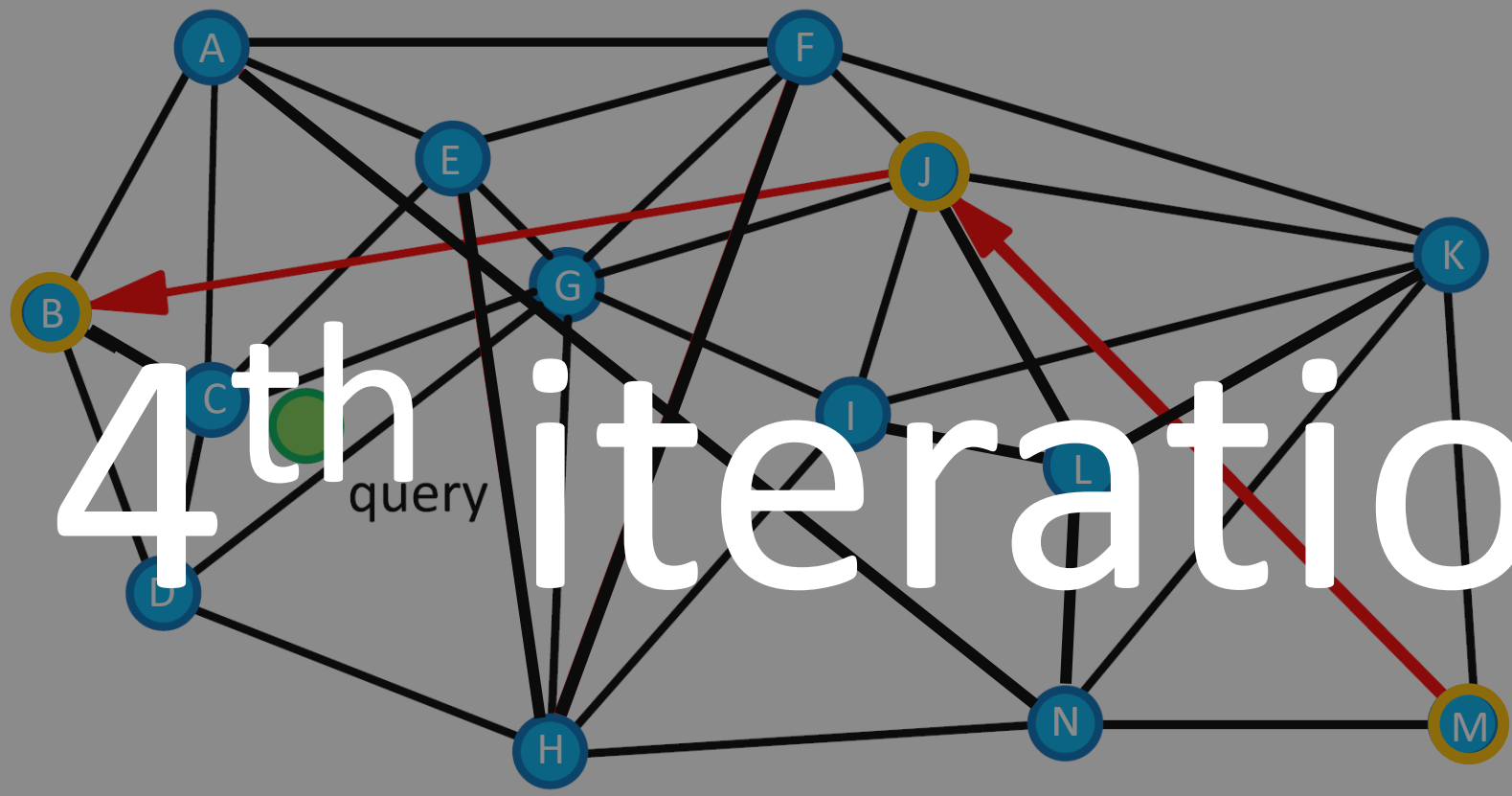
Search

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



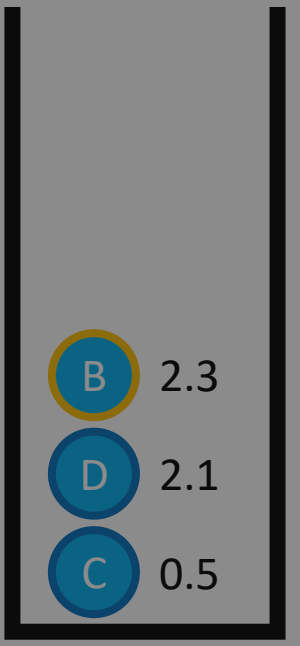


Images are from [Malkov+, Information Systems, 2013]



4th iteration

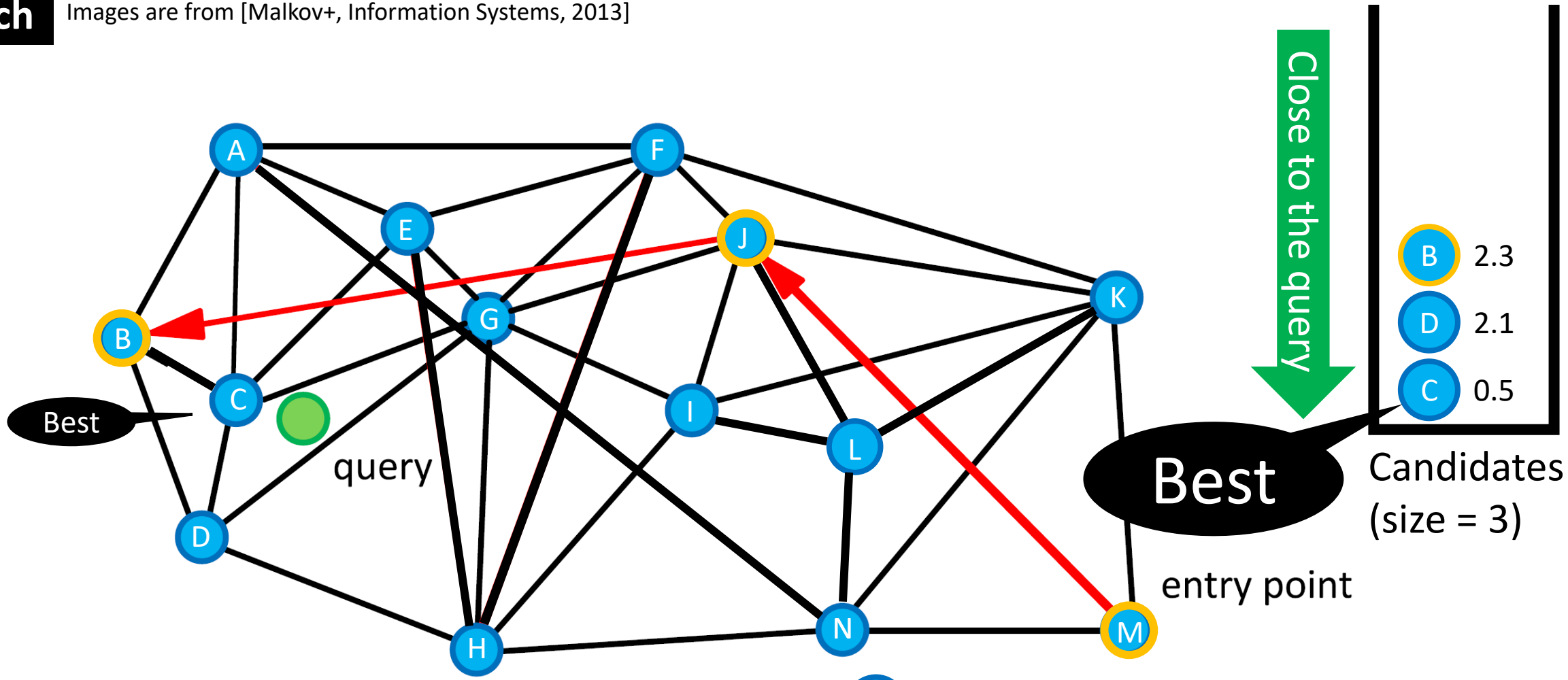
Close to the query



Candidates
(size = 3)

Search

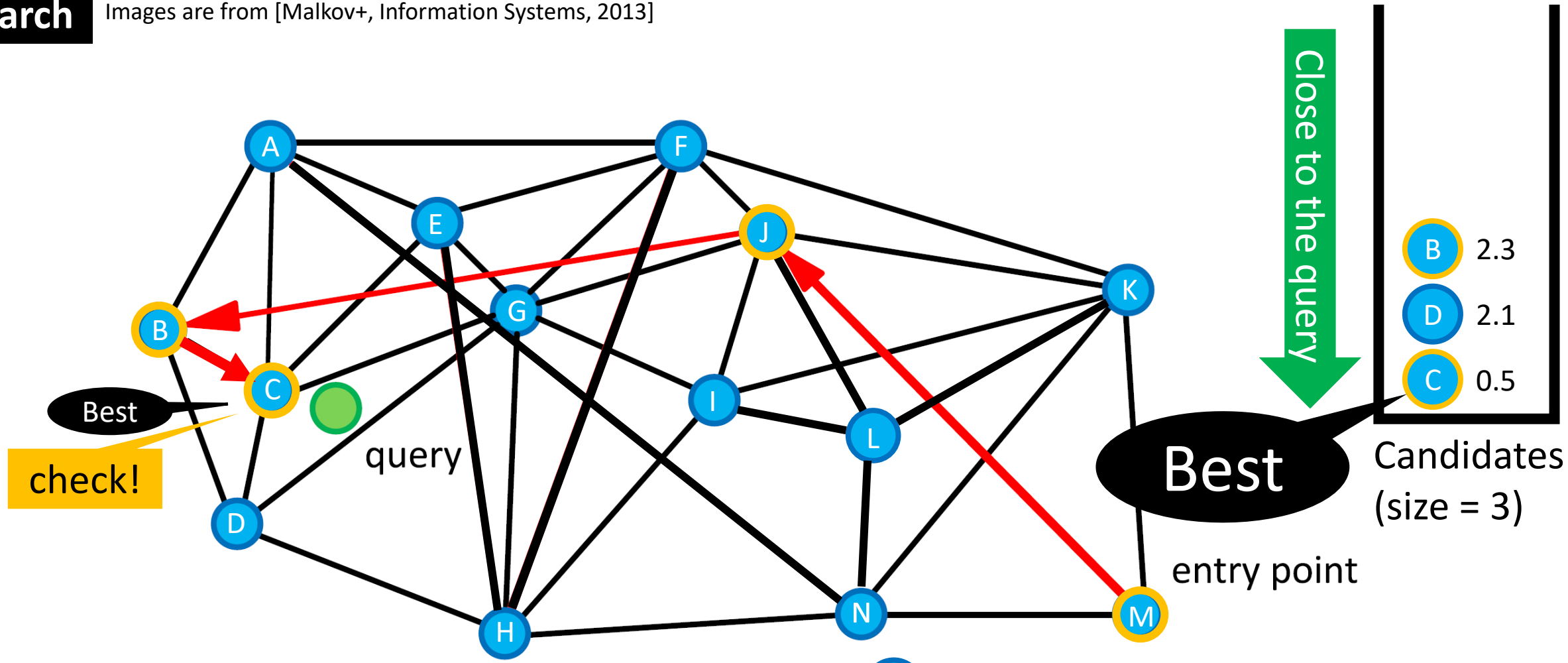
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (C).

Search

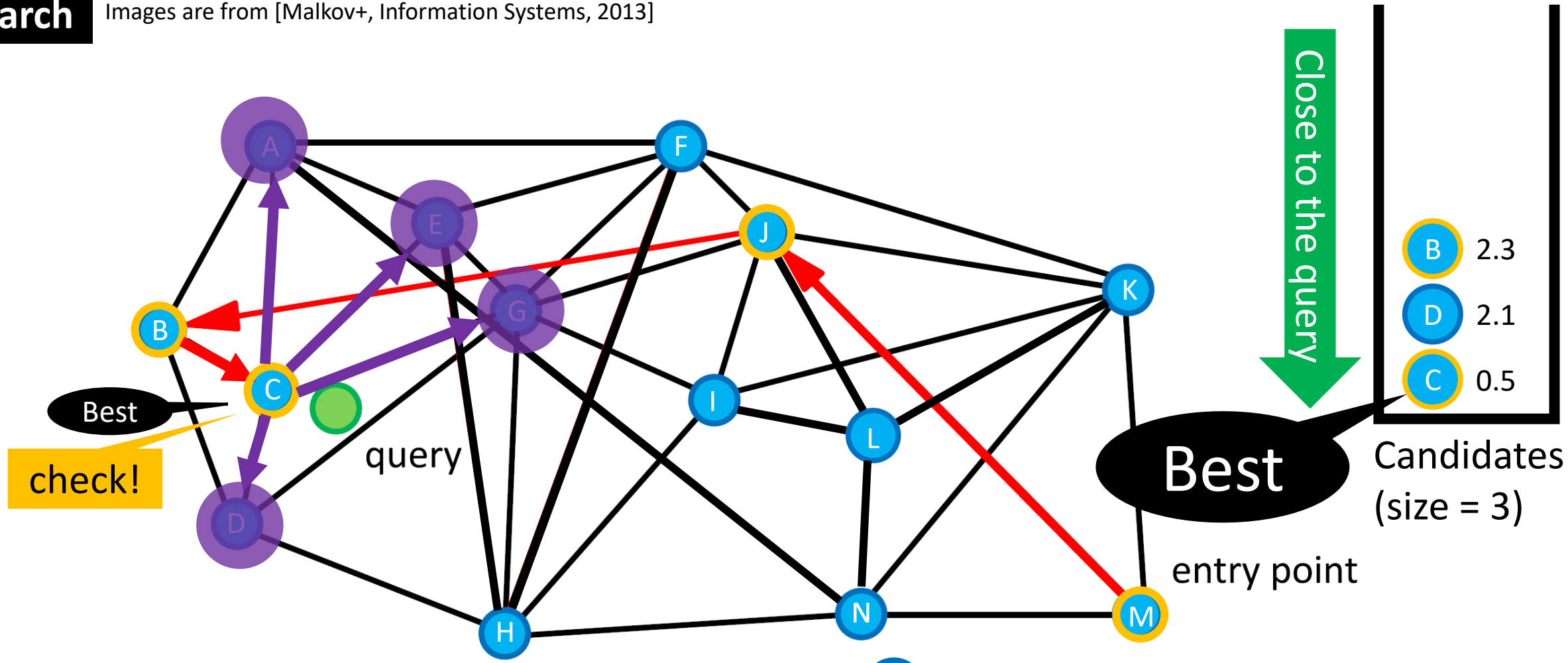
Images are from [Malkov+, Information Systems, 2013]



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

Search

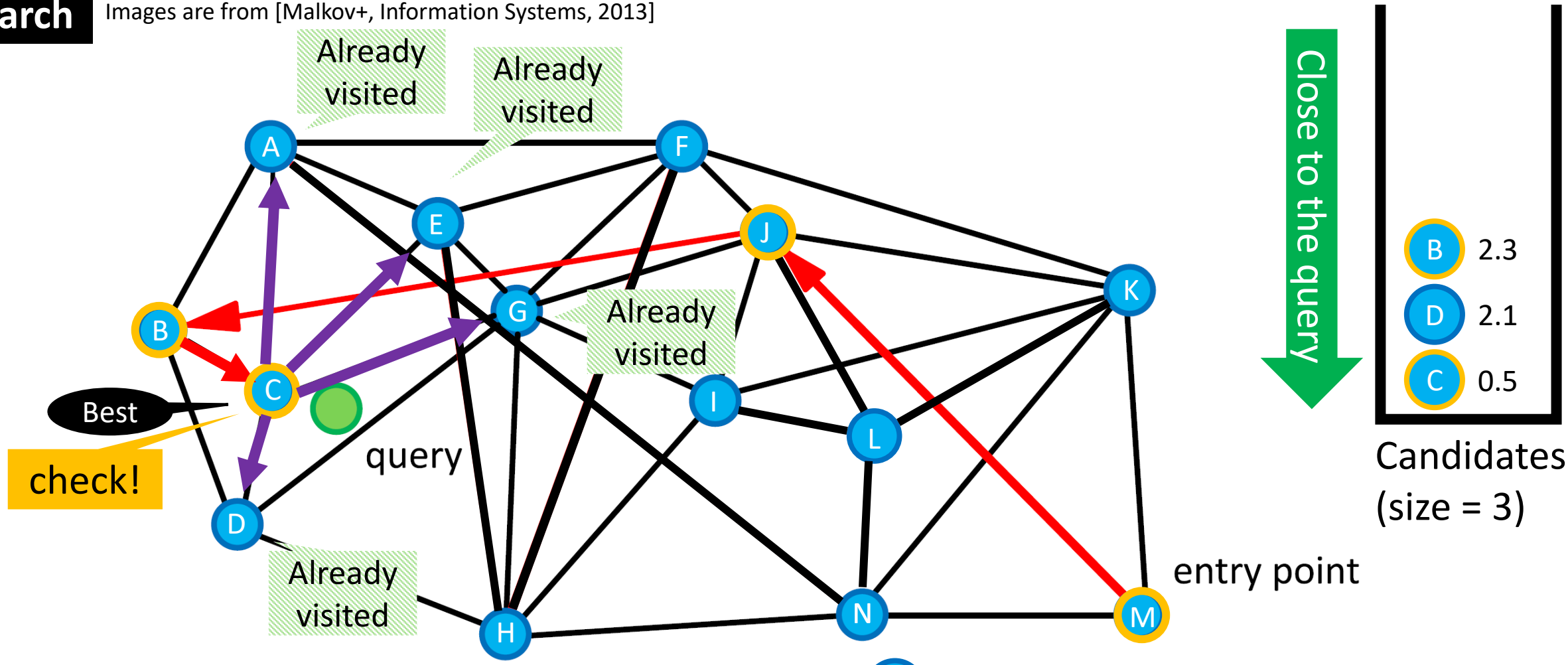
Images are from [Malkov+, Information Systems, 2013]



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

Search

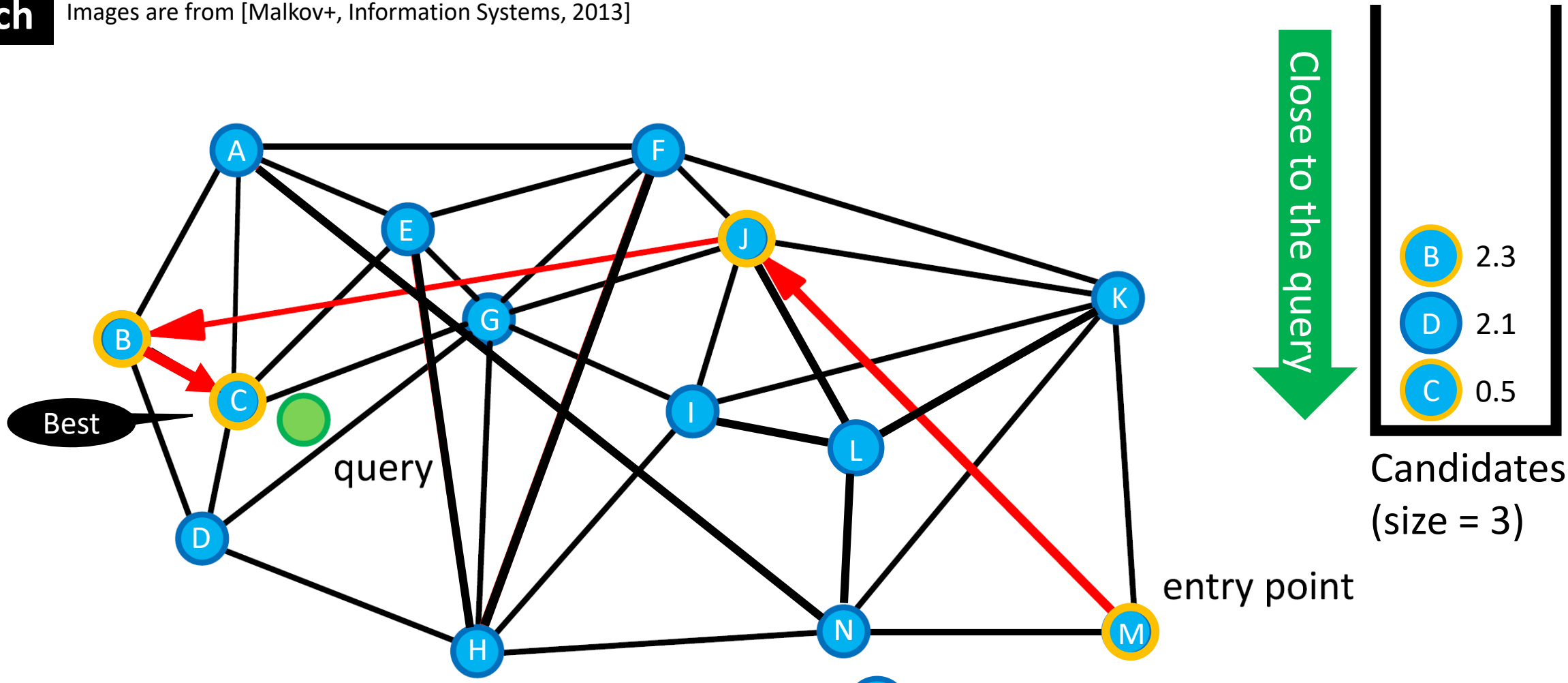
Images are from [Malkov+, Information Systems, 2013]



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

Search

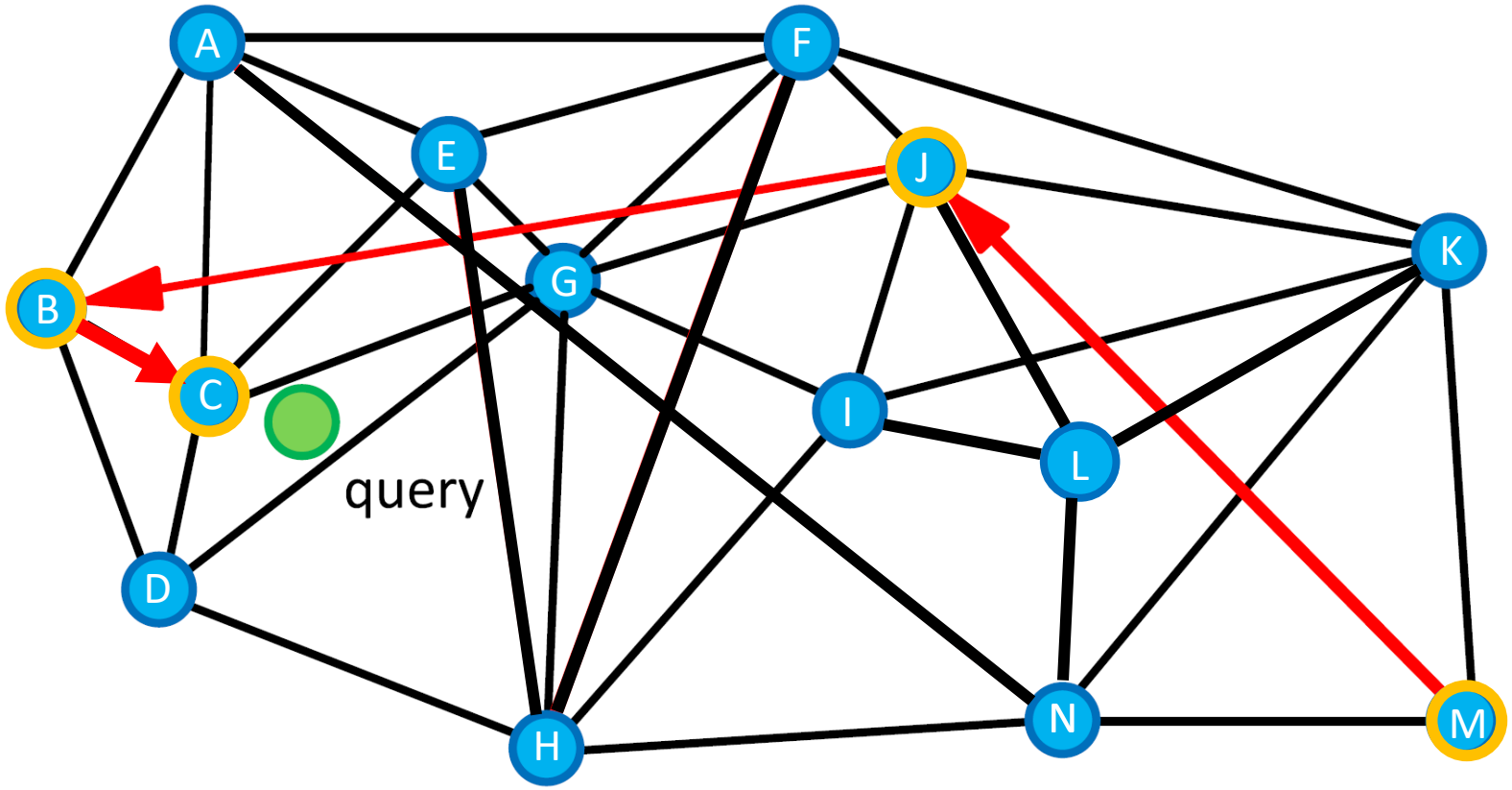
Images are from [Malkov+, Information Systems, 2013]

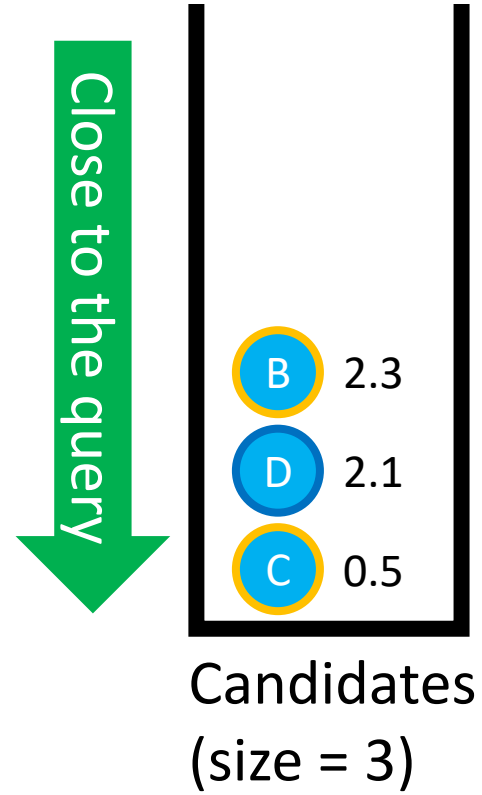


- Pick up the unchecked best candidate (C). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

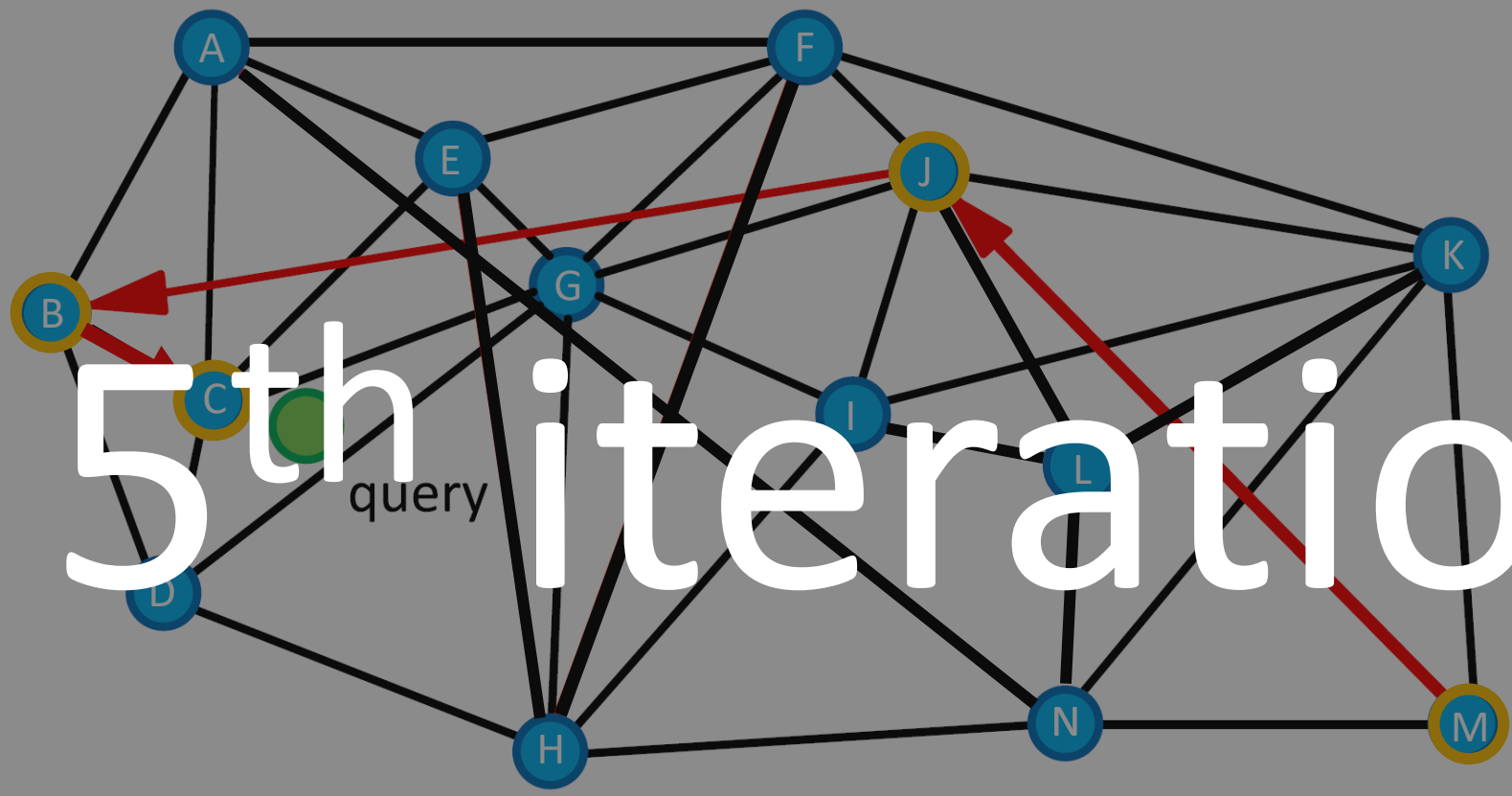
Search

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





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



5th iteration

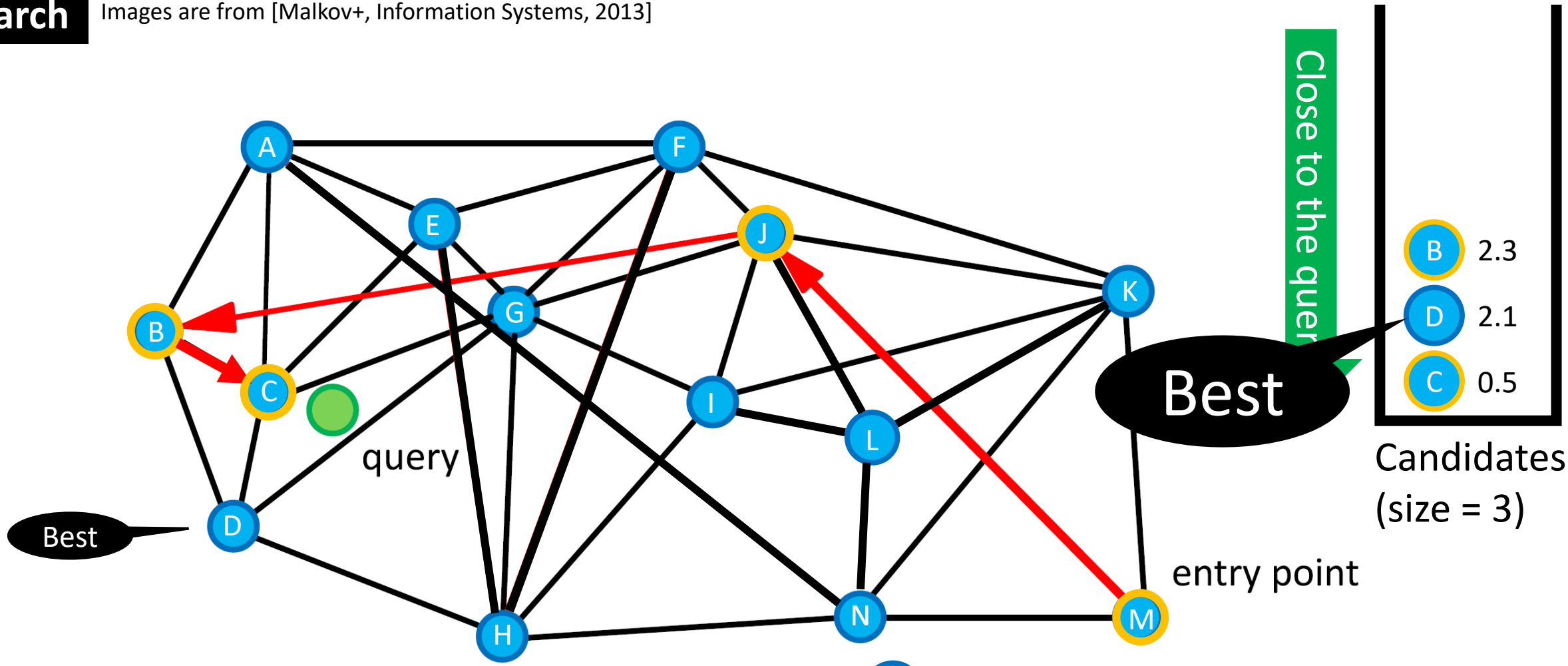
Close to the query

B 2.3
D 2.1
C 0.5

Candidates
(size = 3)

Search

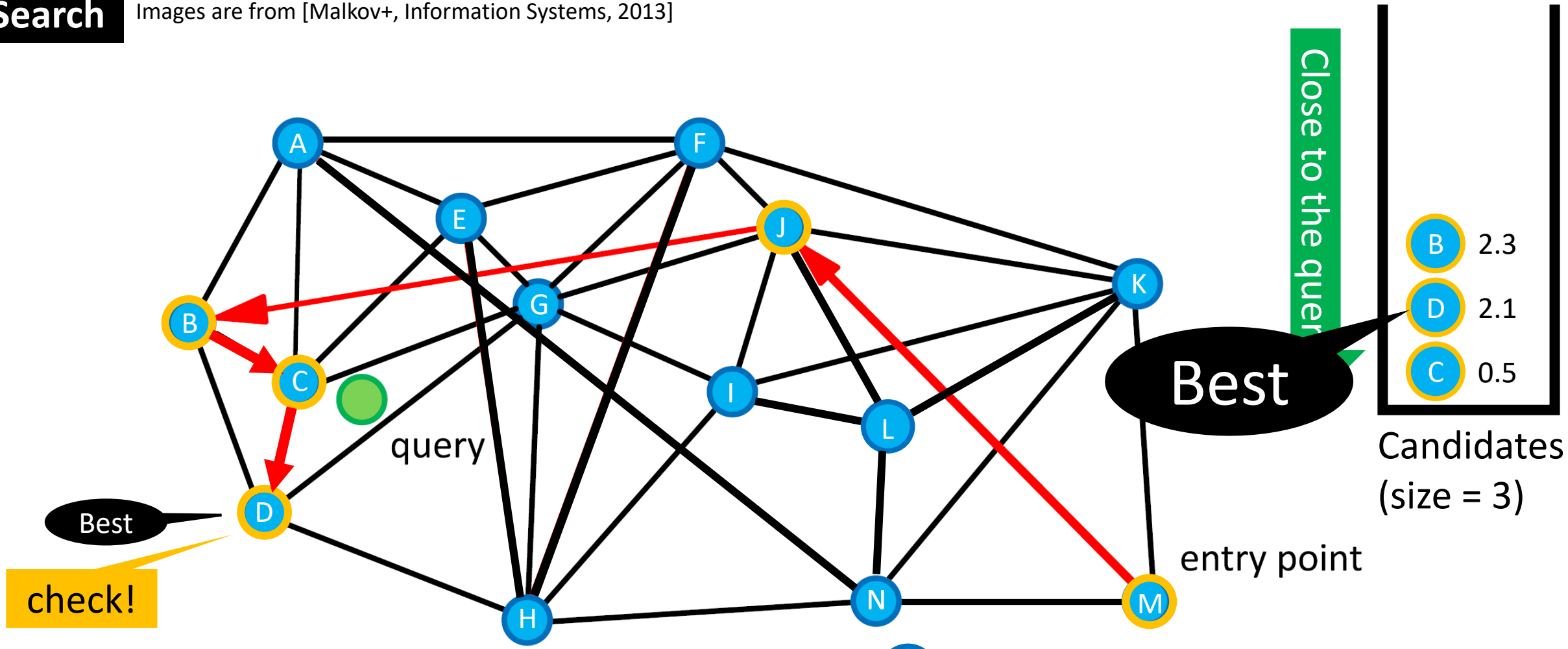
Images are from [Malkov+, Information Systems, 2013]



- Pick up the unchecked best candidate (D).

Search

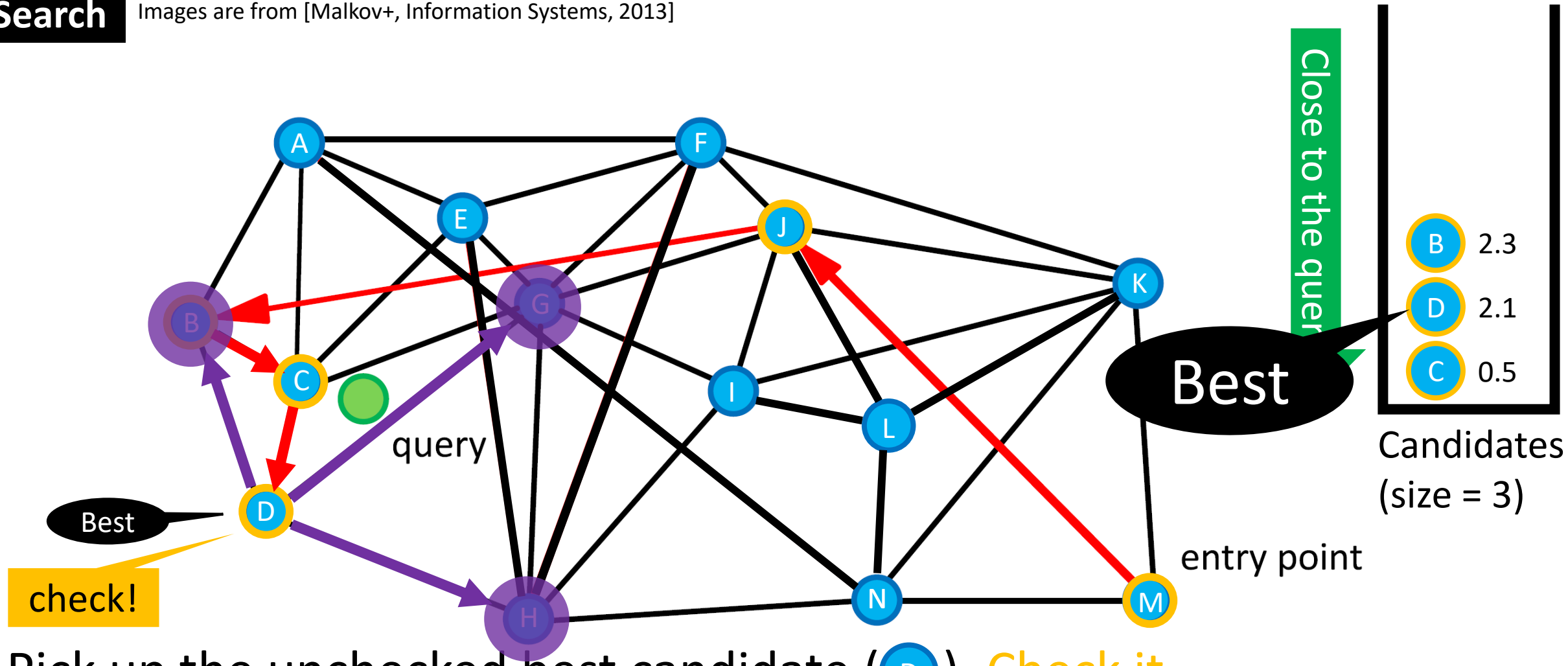
Images are from [Malkov+, Information Systems, 2013]



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

Search

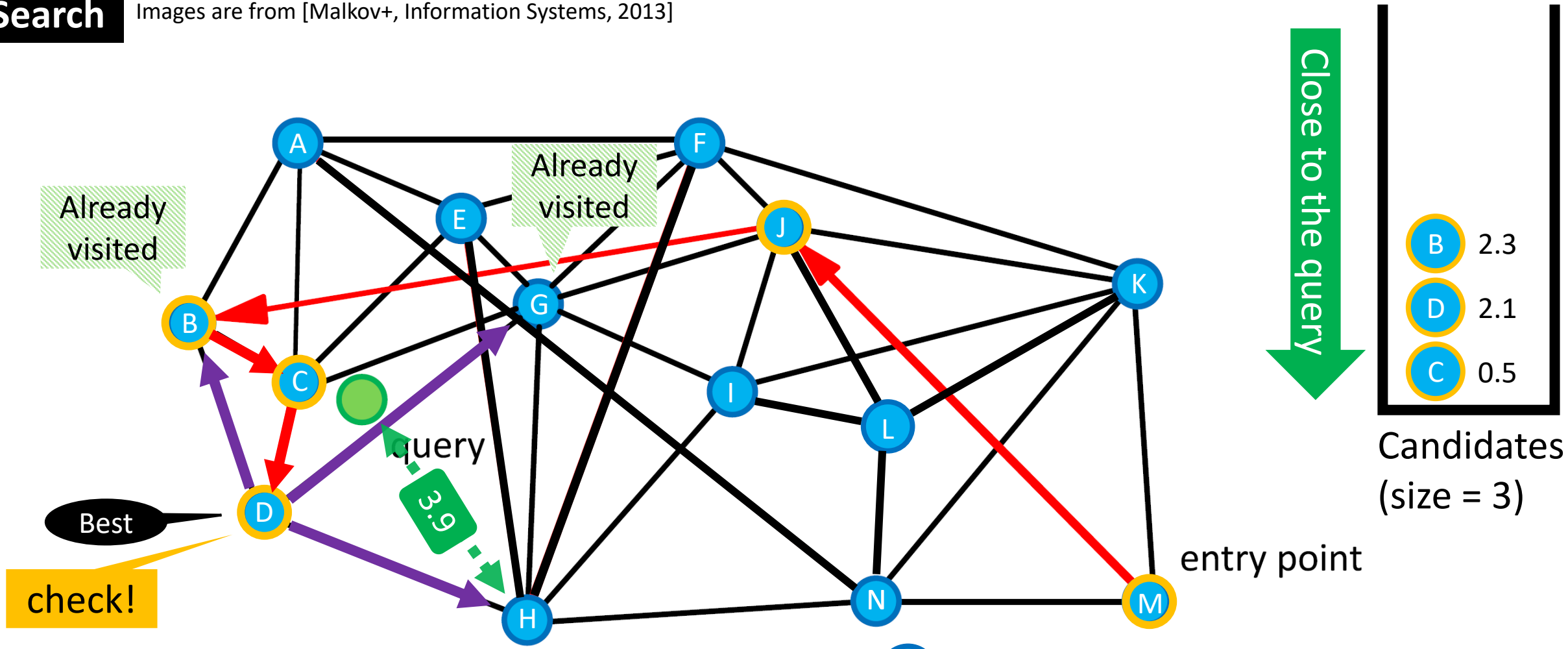
Images are from [Malkov+, Information Systems, 2013]



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

Search

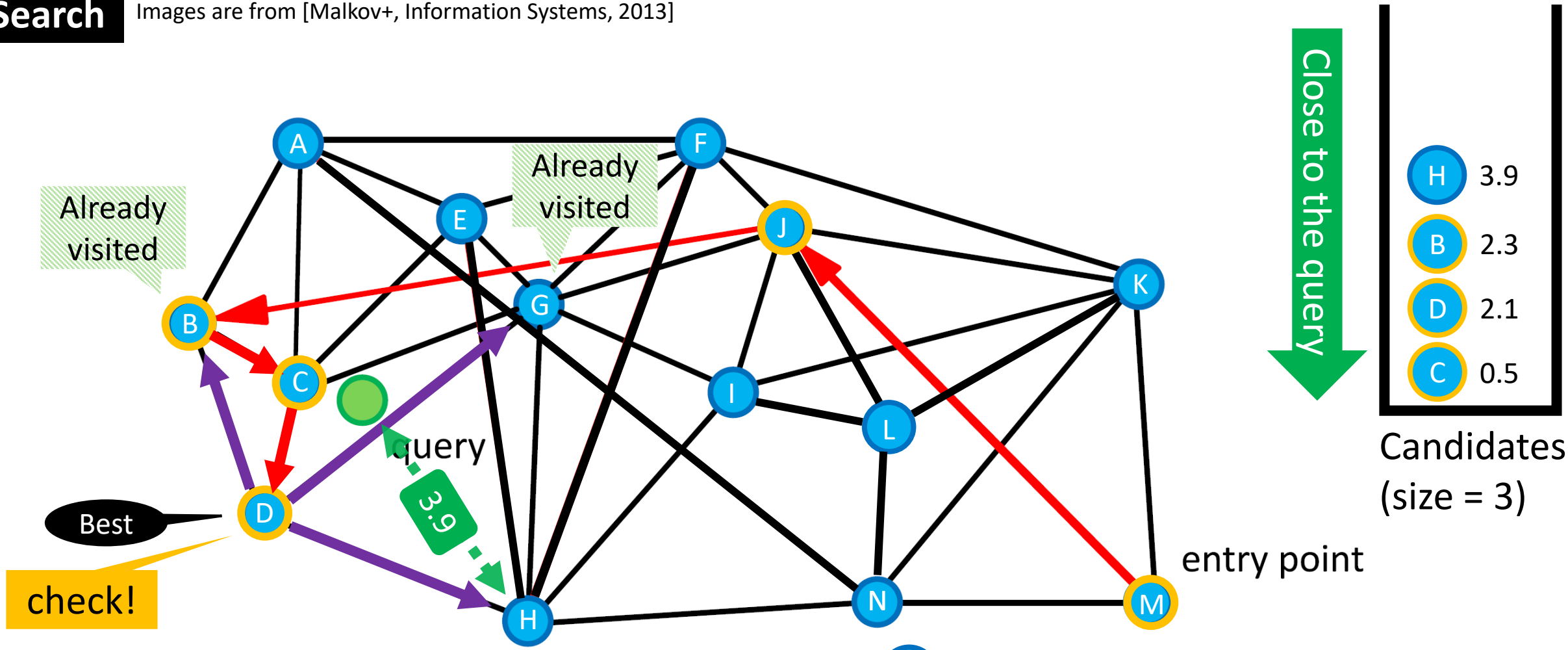
Images are from [Malkov+, Information Systems, 2013]



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

Search

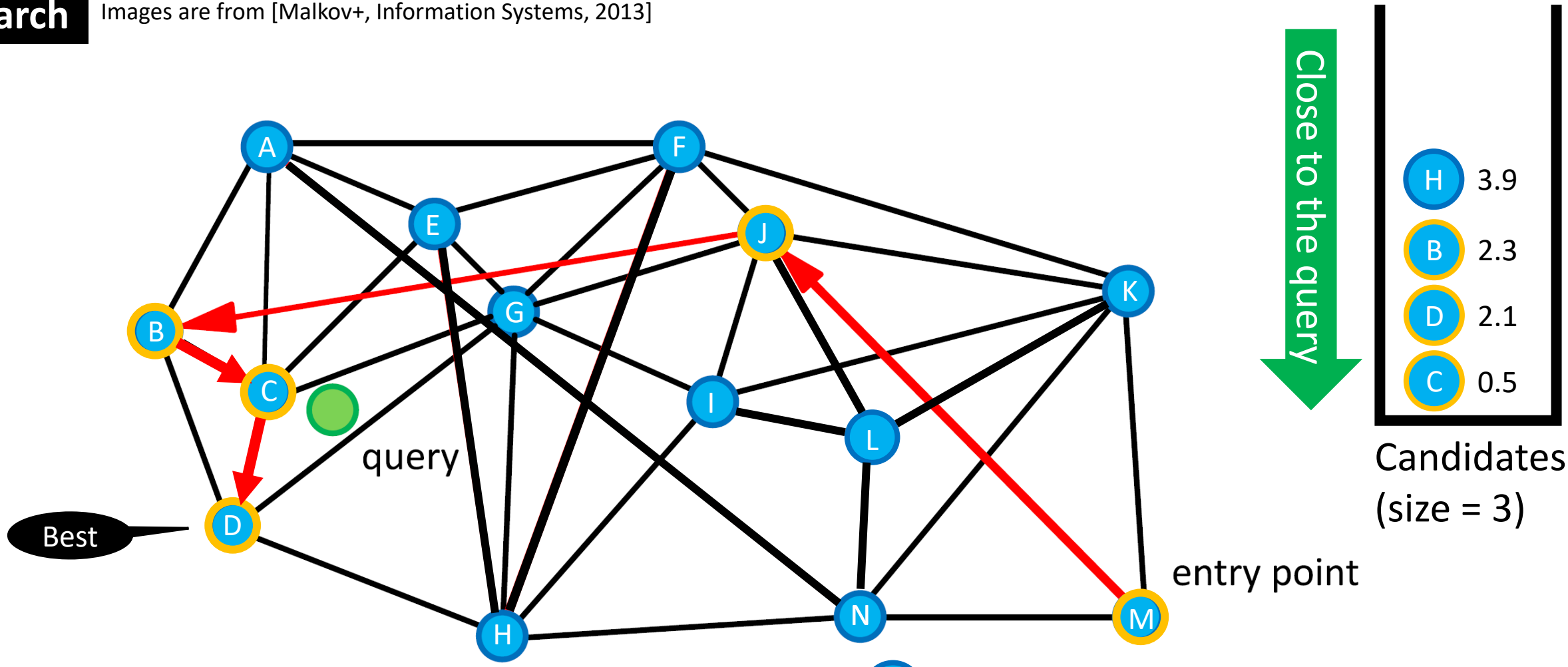
Images are from [Malkov+, Information Systems, 2013]



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

Search

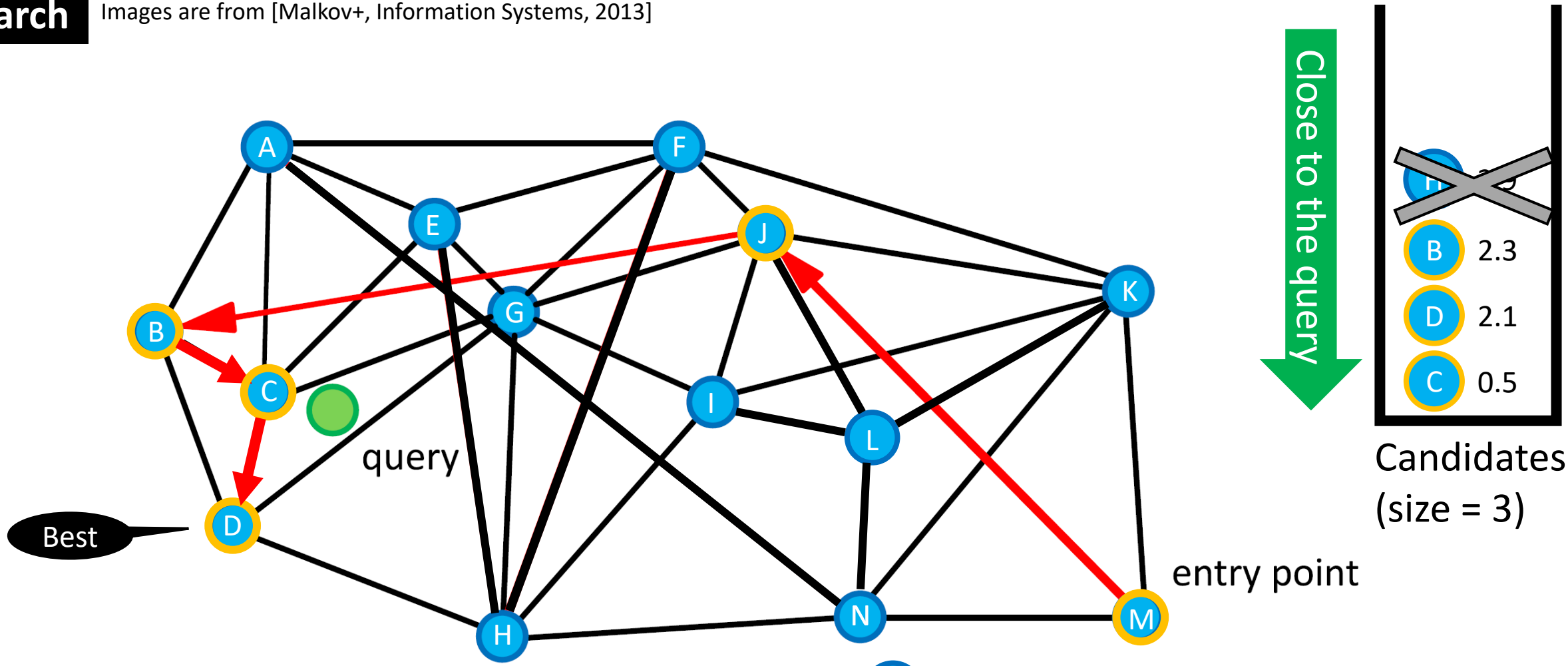
Images are from [Malkov+, Information Systems, 2013]



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

Search

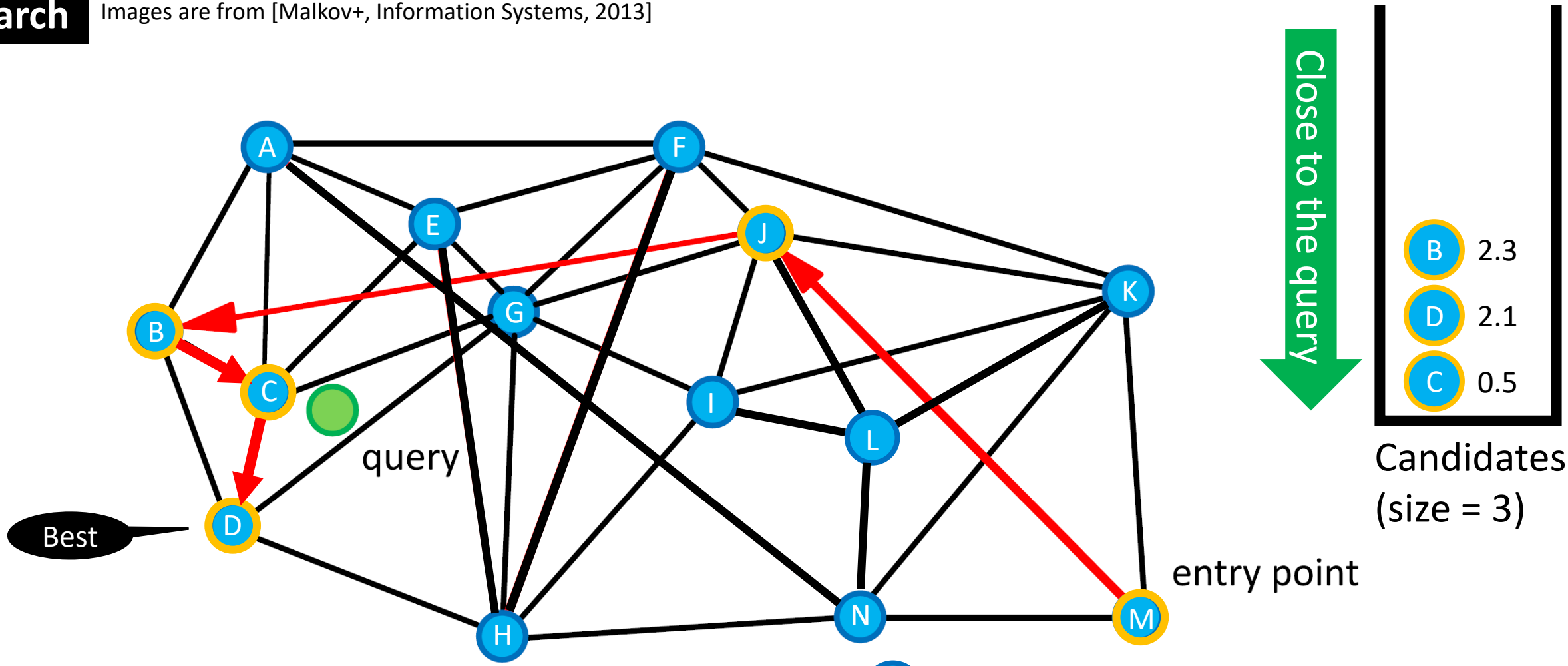
Images are from [Malkov+, Information Systems, 2013]



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

Search

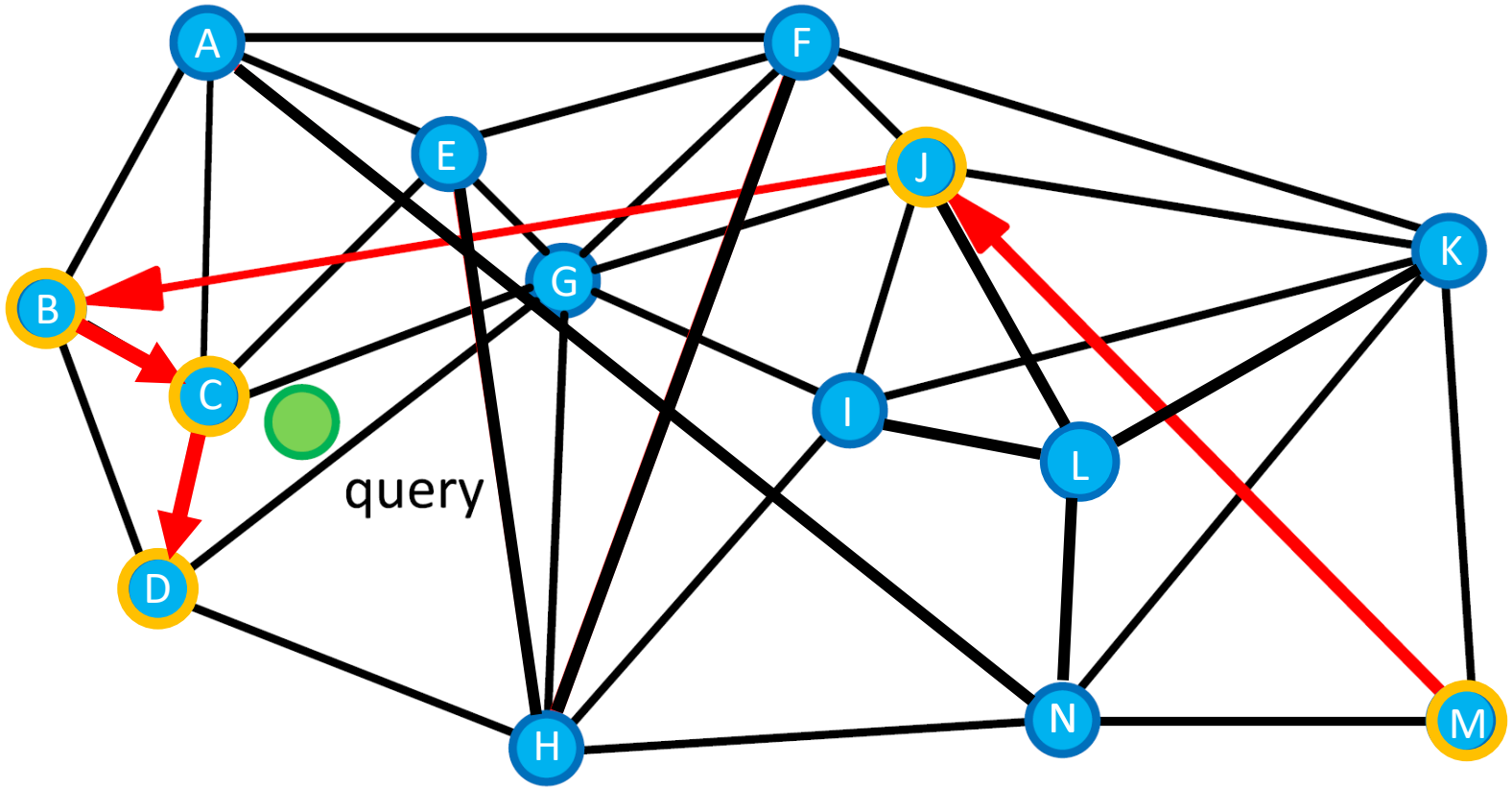
Images are from [Malkov+, Information Systems, 2013]

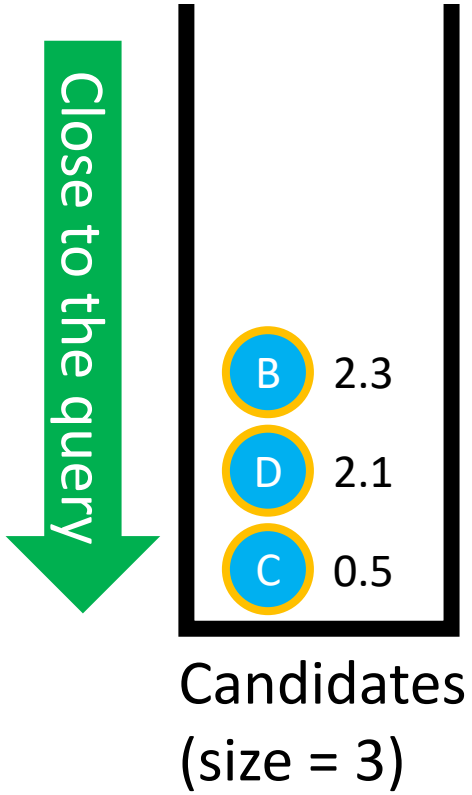


- Pick up the unchecked best candidate (D). Check it.
- Find the connected points.
- Record the distances to q.
- Maintain the candidates (size=3)

Search

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



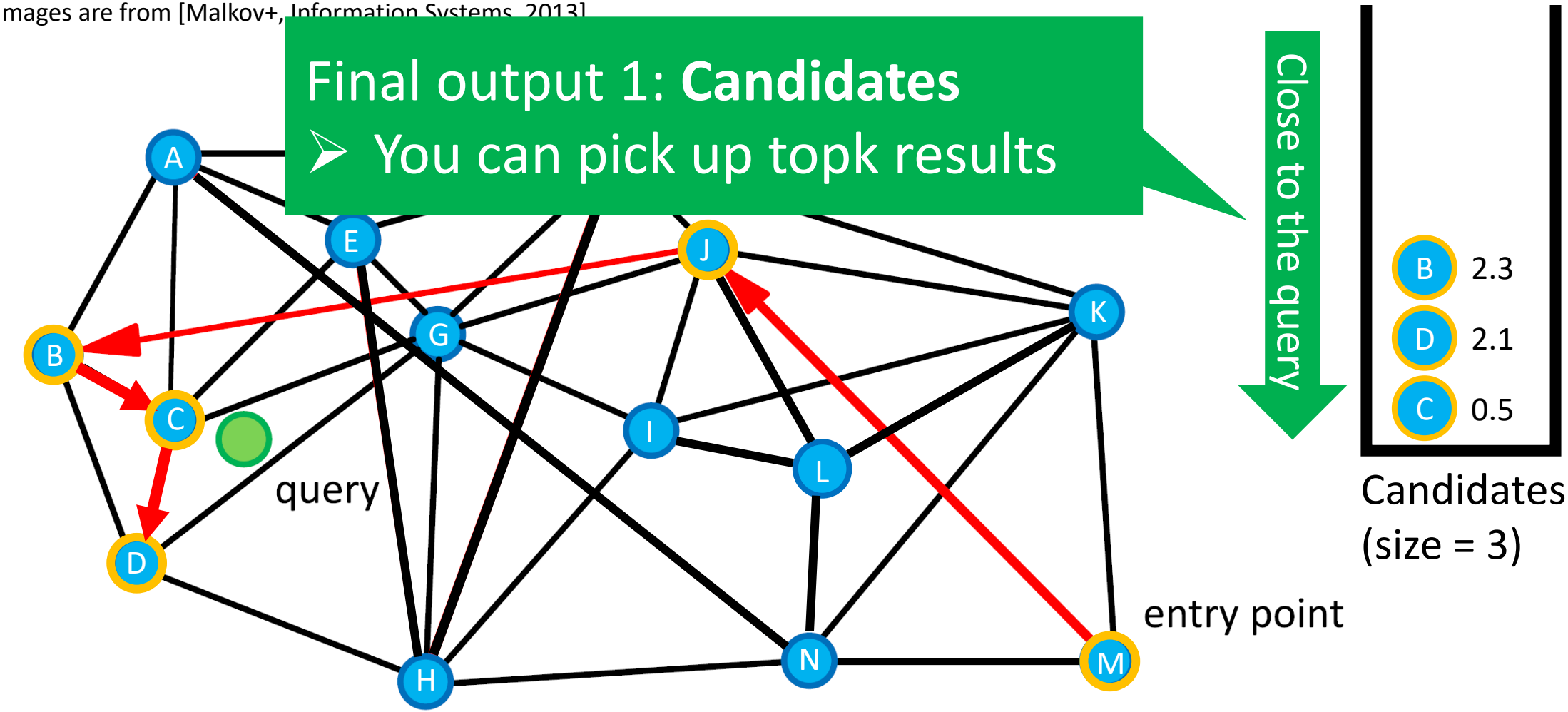


- All candidates are checked. Finish.
- Here, C is the closet to the query ( )

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

Final output 1: **Candidates**

- You can pick up topk results

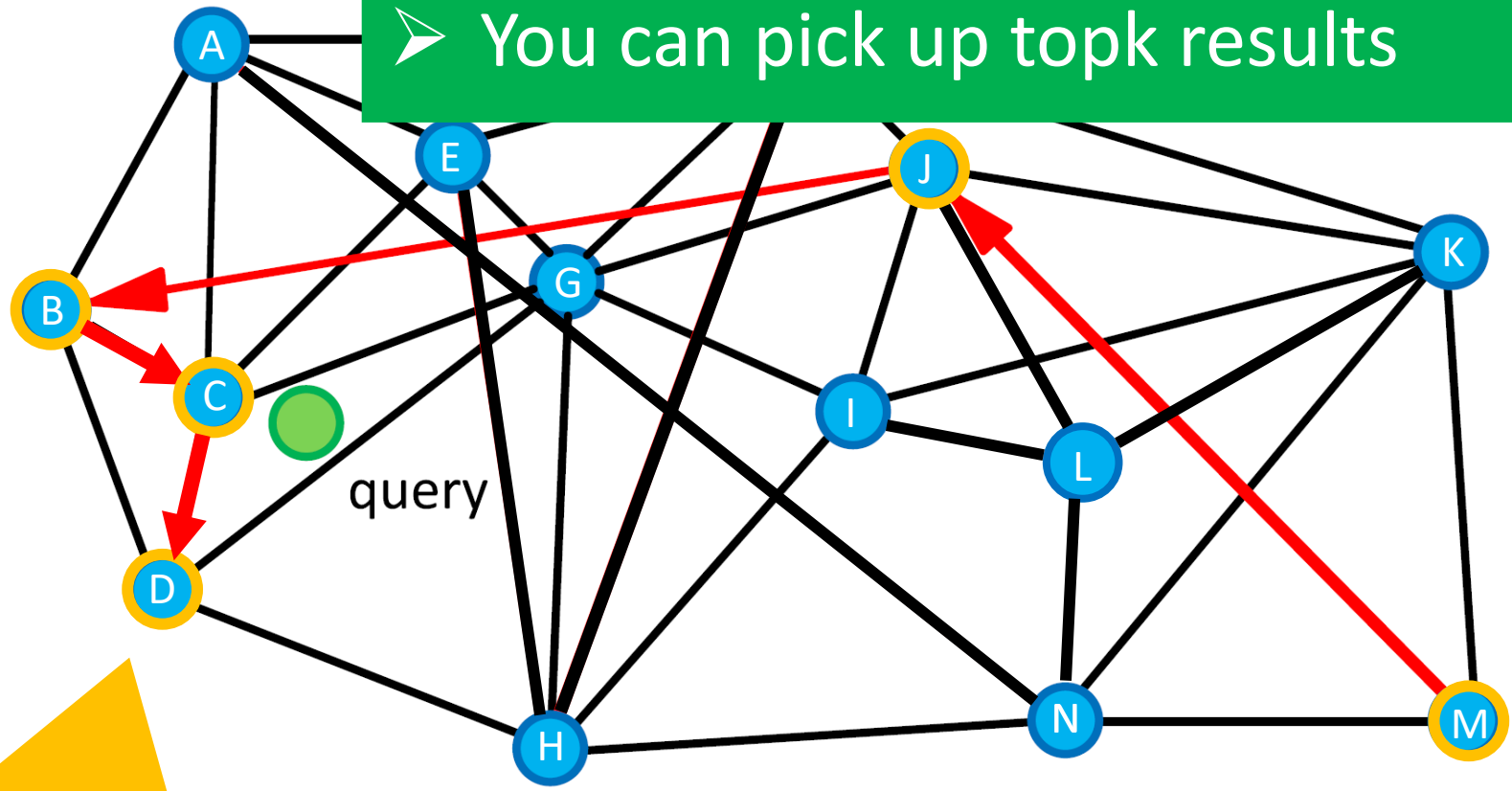


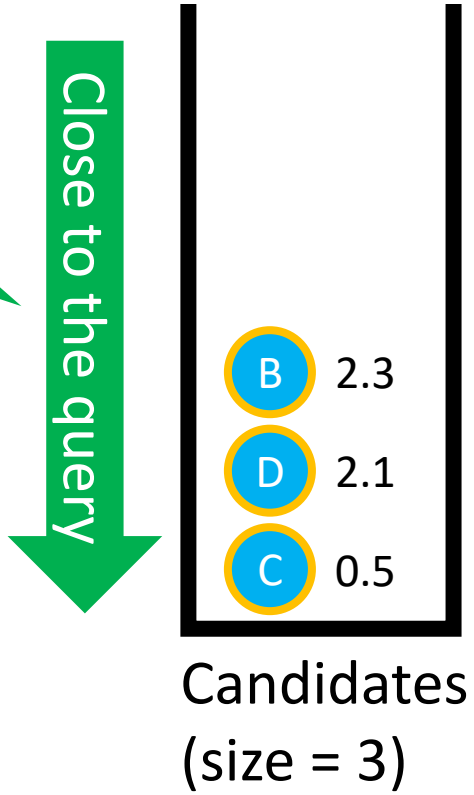
- All candidates are checked. Finish.
- Here, C is the closet to the query ( )

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

Final output 1: **Candidates**

- You can pick up topk results





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

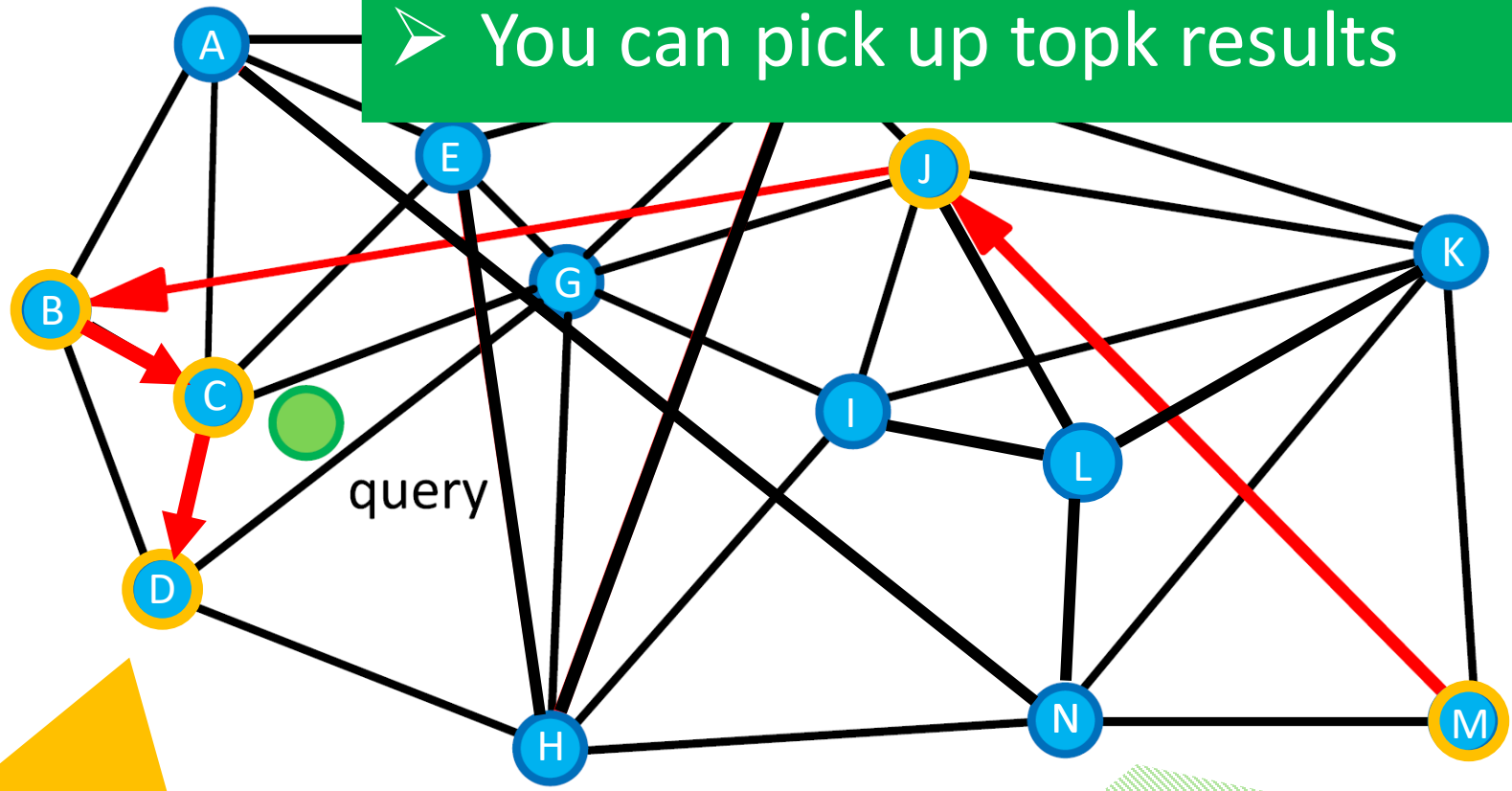
Final output 2: **Checked items**

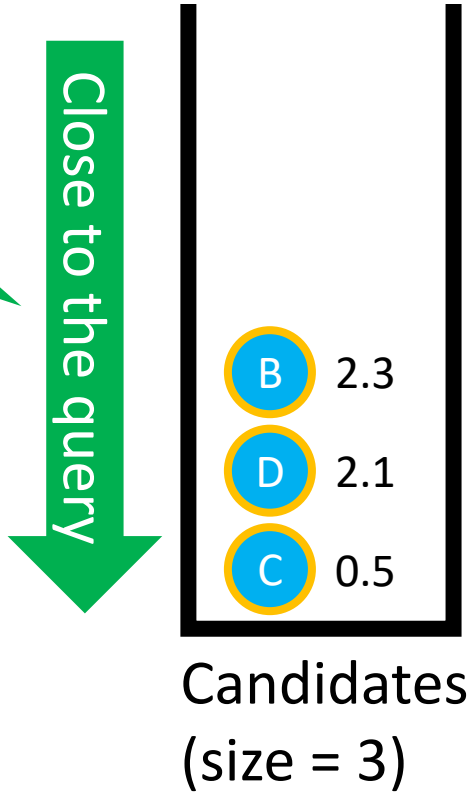
- i.e., search path

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

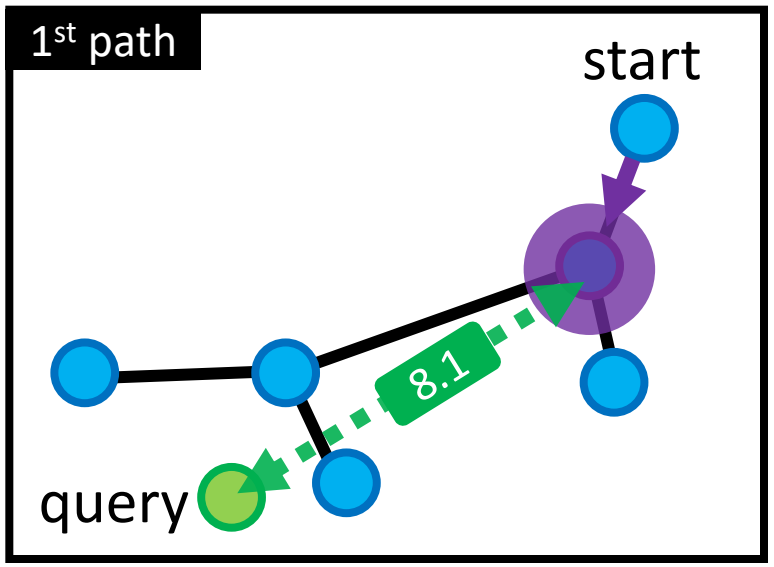
Final output 3: **Visit flag**

- For each item, visited or not

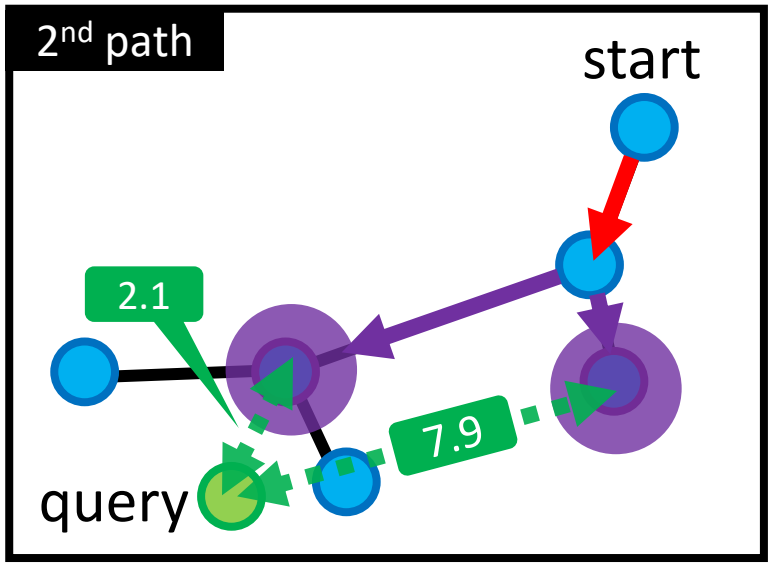
Observation: runtime

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

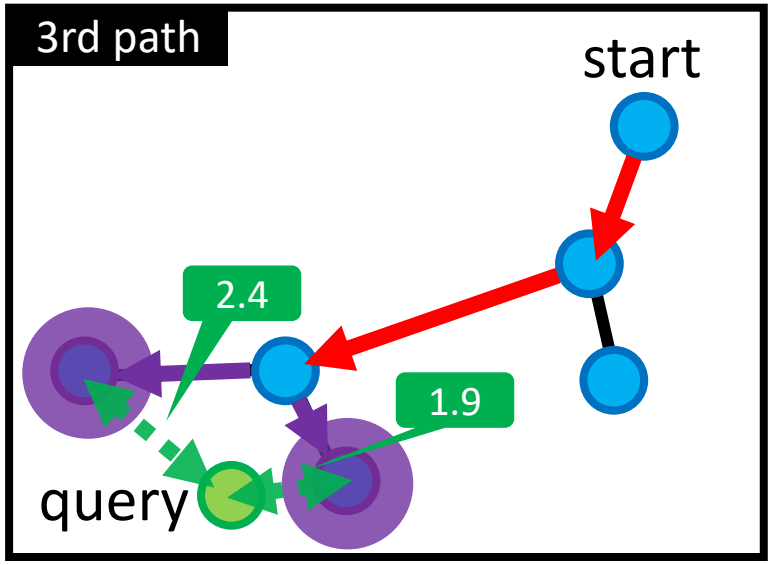
- The overall runtime ~ `#item_comparison`
  ~ `length_of_search_path * average_outdegree`



`outdegree = 1`



`outdegree = 2`



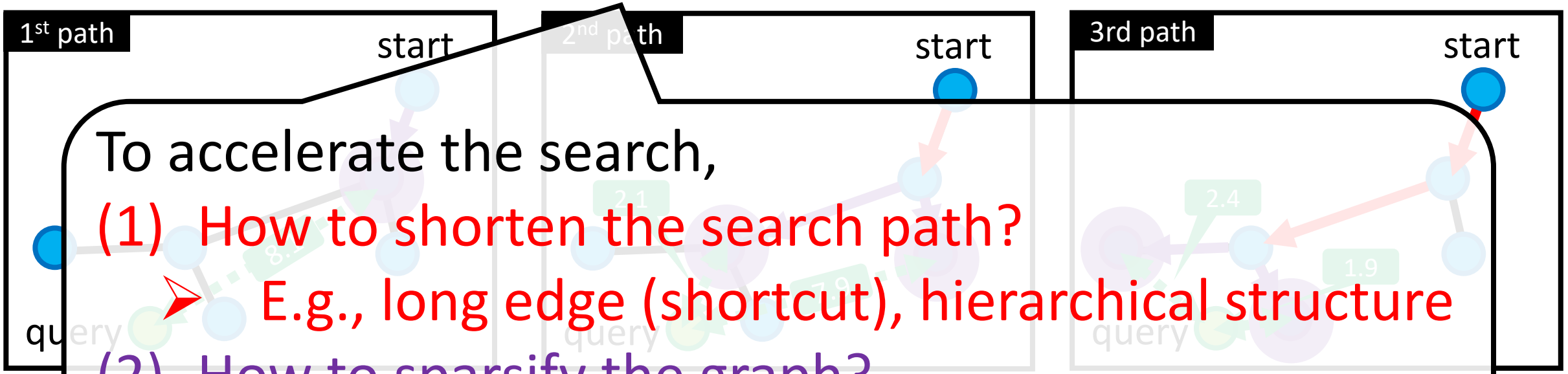
`outdegree = 2`

`#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`

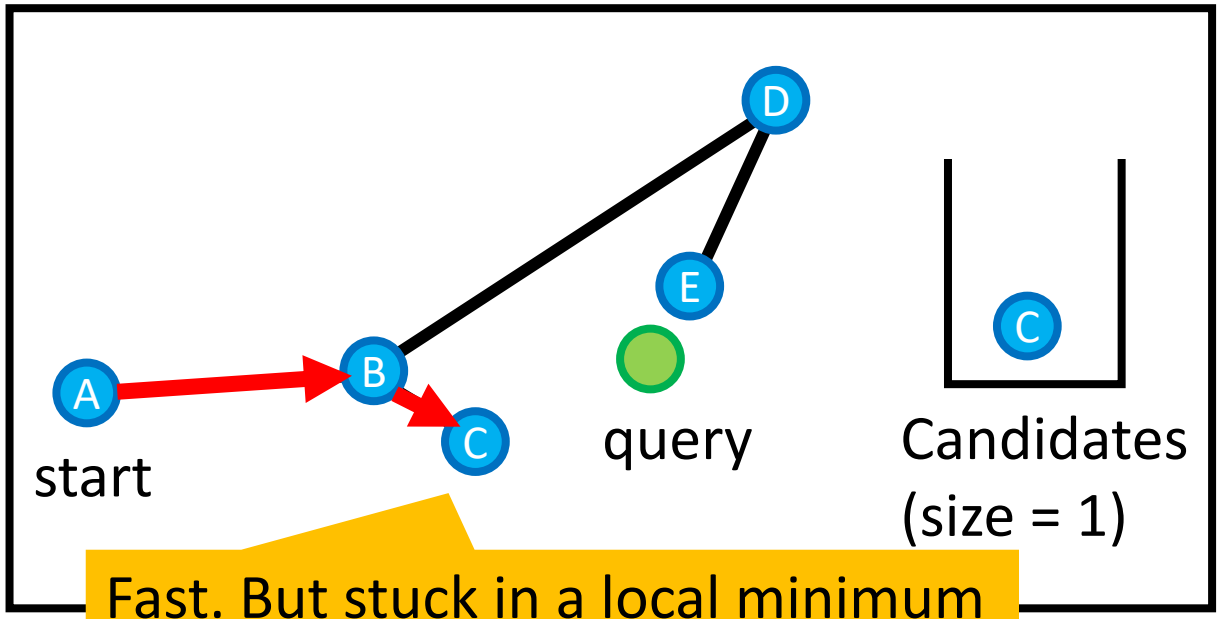


To accelerate the search,

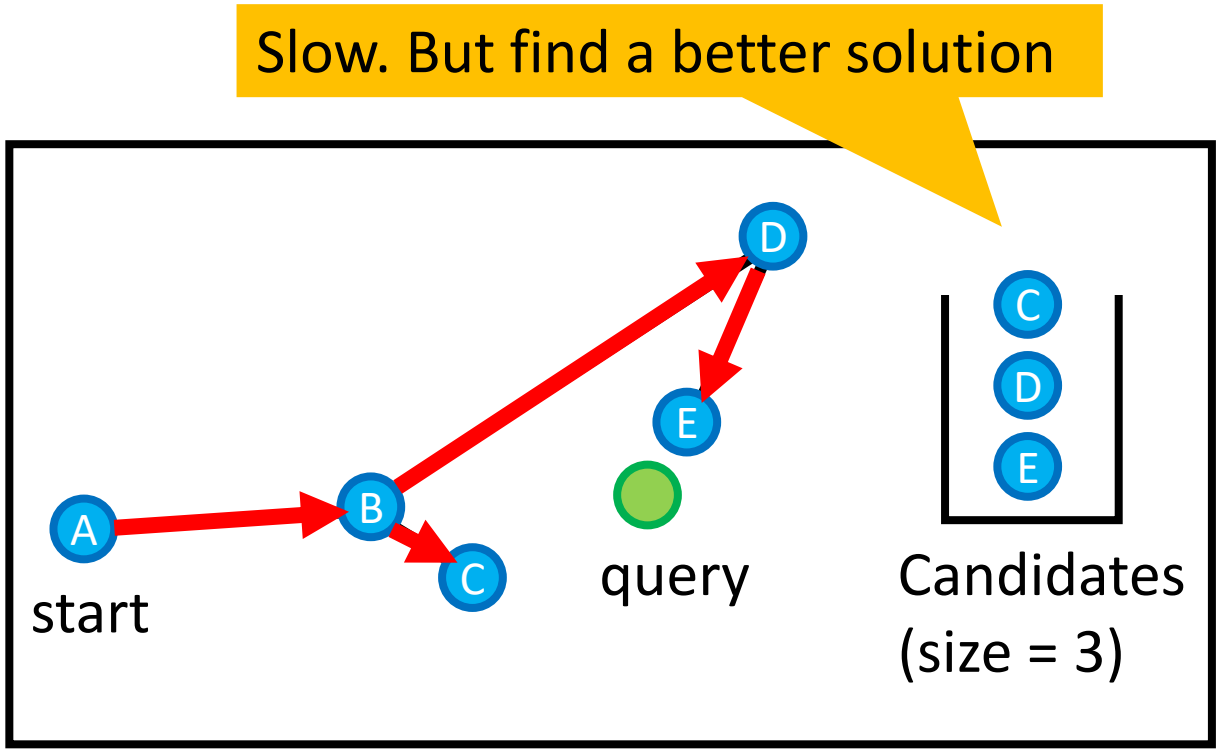
(1) How to shorten the search path?
- E.g., long edge (shortcut), hierarchical structure

(2) How to sparsify the graph?
- E.g., deleting redundant edges

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, **p**, **q**, l)

Require: graph G, start node **p**, query point **q**, candidate pool size l
Ensure: k nearest neighbors of **q**
1: $i = 0$, candidate pool $S = \emptyset$
2: S.add(**p**)
3: **while** $i < l$ **do**
4: i = the id of the first unchecked node p_i in S
5: mark $\mathbf{p}_i$ as checked
6: **for all** neighbor **n** of $\mathbf{p}_i$ in G **do**
7: **if** **n** has not been visited **then**
8: S.add(**n**)
9: **end if**
10: **end for**
11: sort S in ascending order of the distance to **q**
12: **if** S.size() $> l$ **then**
13: S.resize(l) // remove nodes from back of S to keep its size no larger than l
14: **end if**
15: **end while**
16: 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 $p* \leftarrow \arg\min_{p \in \mathcal{L} \setminus \mathcal{V}} ||x_p - x_q||$
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]

- 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

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

1: $i = 0$, candidate pool $S = \emptyset$
2: S.add(p)
3: **while** $i < l$ **do**
4: i = the id of the first unchecked node p_i in S
5: mark p_i as checked
6: **for all** neighbor n of p_i in G **do**
7: **if** n has not been visited **then**
8: S.add(n)
9: **end if**
10: **end for**
11: sort S in ascending order of the distance to q
12: **if** S.size() $> l$ **then**
13: S.resize(l) // remove nodes from back of S to keep its size no larger than l
14: **end if**
15: **end while**
16: return the first k nodes in S

NSG [Cong+, VLDB 19]

Candidates are stored in a set

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 $p* \leftarrow \arg\min_{p \in \mathcal{L} \setminus \mathcal{V}} ||x_p - x_q||$
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]

Sort the array explicitly

When need to sort, say “closest L points”

Candidates are stored in a heap; automatically sorted

- All pseudo code is essentially 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
1: $i = 0$, candidate pool $S = \emptyset$
2: S.add(p)
3: **while** $i < l$ **do**
4: i = the id of the first unchecked node p_i in S
5: mark p_i as checked
6: **for** all neighbor n of p_i in G **do**
7: **if** n has not been visited **then**
8: S.add(n)
9: **end if**
10: **end for**
11: sort S in ascending order of the distance to q
12: **if** S.size() $> l$ **then**
13: S.resize() // remove nodes from back of S to keep its
14: size no larger than l
15: **end if**
16: **end while**
17: 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 $\hat{v} \in Expand(v_i, G)$ **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 pseudo codes have totally different pseudo codes
- Principles are the same. But small details are different.
- Hint: Explicitly state the data structure or not

Just "check"

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

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**
1: $i = 0$, candidate pool $S = \emptyset$
2: S.add(**p**)
3: **while** $i < l$ **do**
4: i = the id of the first unchecked node p_i in S
5: mark $\mathbf{p}_i$ as checked
6: **for all** neighbor **n** of $\mathbf{p}_i$ in G **do**
7: **if** **n** has not been visited **then**
8: S.add(**n**)
9: **end if**
10: **end for**
11: sort S in ascending order of the distance to **q**
12: **if** S.size() $> l$ **then**
13: S.resize(l) // remove nodes from back of S to keep its
14: size no larger than l
15: **end if**
16: **end while**
17: 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 $\hat{v} \in Expand(v_i, G)$ **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**
1: $i = 0$, candidate pool $S = \emptyset$
2: S.add(**p**)
3: **while** $i < l$ **do**
4: i = the id of the first unchecked node p_i in S
5: mark $\mathbf{p}_i$ as checked
6: **for all** neighbor **n** of $\mathbf{p}_i$ in G **do**
7: **if** **n** has not been visited **then**
8: S.add(**n**)
9: **end if**
10: **end for**
11: sort S in ascending order of the distance to **q**
12: **if** S.size() $> l$ **then**
13: S.resize(l) // remove nodes from back of
14: size no larger than l
15: **end if**
16: **end while**
17: 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 $\mathcal{L} \leftarrow \mathcal{L} \cup N_{\text{out}}(p^*)$ and
$\mathcal{V} \leftarrow \mathcal{V} \cup \{p^*\}$
if $|\mathcal{L}| > L$ **then**
update $\mathcal{L}$ to retain closest L
; $\mathcal{V}$]

IPS 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]

Termination condition??

- 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**
1: $i = 0$, candidate pool $S = \emptyset$
2: S.add(**p**)
3: **while** $i < l$ **do**
4: i = the id of the first unchecked node p_i in S
5: mark $\mathbf{p}_i$ as checked
6: **for all** neighbor **n** of $\mathbf{p}_i$ in G **do**
7: **if** **n** has not been visited **then**
8: S.add(**n**)
9: **end if**
10: **end for**
11: sort S in ascending order of the distance to **q**
12: **if** S.size() $> l$ **then**
13: S.resize(l) // remove nodes from back of S to keep its size no larger than l
14: **end if**
15: **end while**
16: **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 $p* \leftarrow \arg\min_{p \in \mathcal{L} \setminus \mathcal{V}} ||x_p - x_q||$
    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]

- All papers have totally different pseudo code 😟
- Principle underlying: Best-first search
- Hint: Explicitly state the data structure or not
- 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**
1: $i = 0$, candidate pool $S = \emptyset$
2: S.add(**p**)
3: **while** $i < l$ **do**
4: i = the id of the first unchecked node p_i in S
5: mark $\mathbf{p}_i$ as checked
6: **for all** neighbor **n** of $\mathbf{p}_i$ in G **do**
7: **if** **n** has not been visited **then**
8: S.add(**n**)
9: **end if**
10: **end for**
11: sort S in ascending order of the distance to **q**
12: **if** S.size() $> l$ **then**
13: S.resize(l) // remove nodes from back of S to keep its size no larger than l
14: **end if**
15: **end while**
16: **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 $p* \leftarrow \arg\min_{p \in \mathcal{L} \setminus \mathcal{V}} ||x_p - x_q||$
update $\mathcal{L} \leftarrow \mathcal{L} \cup N_{out}(p^*)$ and $\mathcal{V} \leftarrow \mathcal{V} \cup \{p^*\}$
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
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 details are different.
- Hint: Explicitly state the data structure or not

Outline

1. History from an applications perspective
2. Importance of implementation
3. Basics of modern baseline