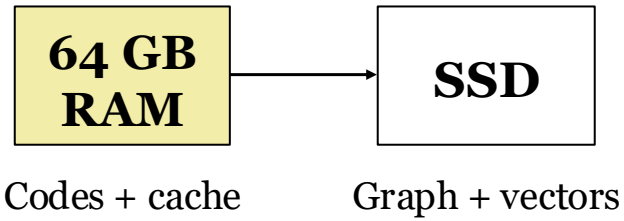
DiskANN:

Fast Accurate Billion-point Nearest Neighbor Search on a Single Node

Suhas Jayaram Subramanya, Devvrit, Rohan Kadekodi,
Ravishankar Krishnaswamy, Harsha Vardhan Simhadri

NeurIPS 2019



Baoming Chang

CS 848 November 5, 2026

UNIVERSITY OF WATERLOO

Why the index does not fit in RAM

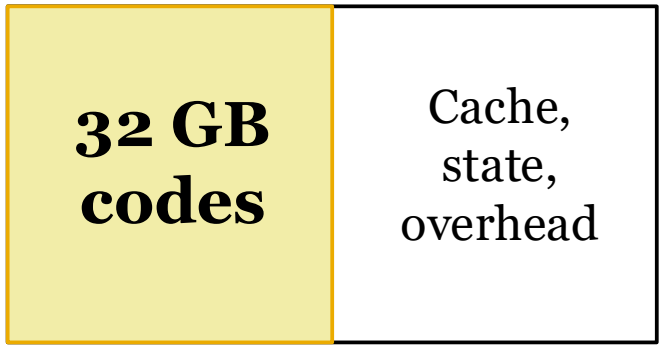
1 billion 128-dimensional vectors. Assume a maximum out-degree of R = 128.

Choose one raw representation

Coordinates	Bytes / point	Payload
float32	128 × 4 = 512	512 GB
or uint8	128 × 1 = 128	128 GB

Graph IDs 128 × 4 = 512 bytes / point
Up to 512 GB

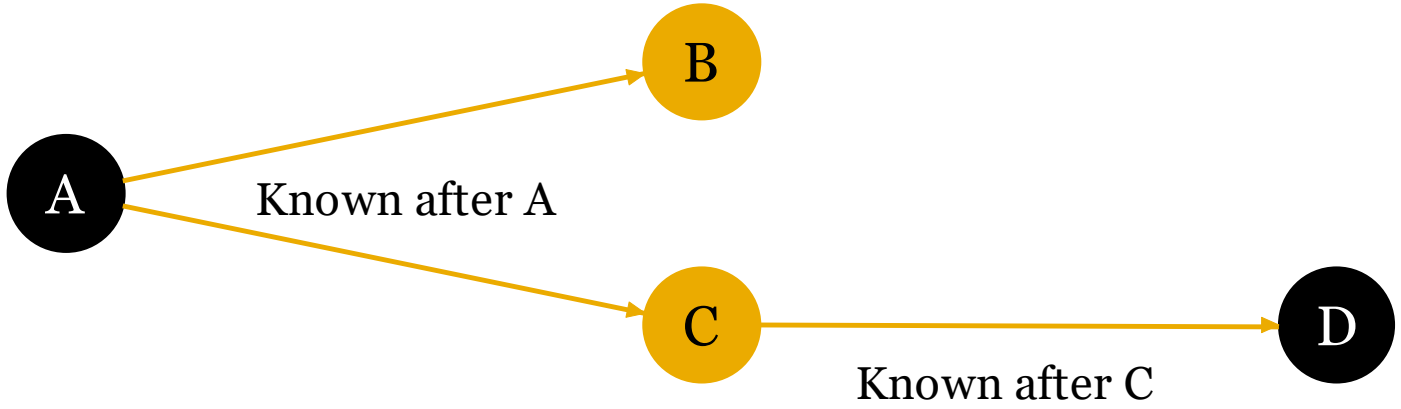
RAM budget: 64 GB



32 bytes per point

SSD reads and dependent rounds

A reveals both B and C. Only C reveals the address of D.



Serial	**A**	**B**	**C**	**D**	**4 reads, 4 rounds**
Parallel	**A**	**B + C**	**D**		**4 reads, 3 rounds**

The design of DiskANN

Graph construction and SSD execution form one system.

1	**Vamana**	Search for candidate neighbors, then retain useful routes.
2	**SSD search**	Use compressed codes in RAM, parallel reads, and cached records.
3	**Building at scale**	Construct overlapping partitions and merge their edges.

GreedySearch maintains two sets

A point can remain in C after it joins the expanded history V.

C: at most L retained IDs



V: every expanded ID

p03, p13, p04, p05

C controls who can be expanded next.

V records what has already expanded.

GreedySearch: pseudocode

The bounded candidate list and the expanded history serve different purposes.

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

Data: Graph G; start s; query x_q; result size k; list size $L \geq k$.
Result: Returned k candidates and the expanded set V.

$C \leftarrow \{s\}, \qquad V \leftarrow \varnothing$
while $C \setminus V \neq \varnothing$ **do**
  $v \leftarrow \arg\min_{u \in C \setminus V} \|x_u - x_q\|$
  $C \leftarrow C \cup N^+(v)$
  $V \leftarrow V \cup \{v\}$
  if $|C| > L$ **then**
    $C \leftarrow$ the closest L points in C to x_q
return [closest k points in C to x_q; V]

C: retained candidates

At most L retained alternatives for future expansion.

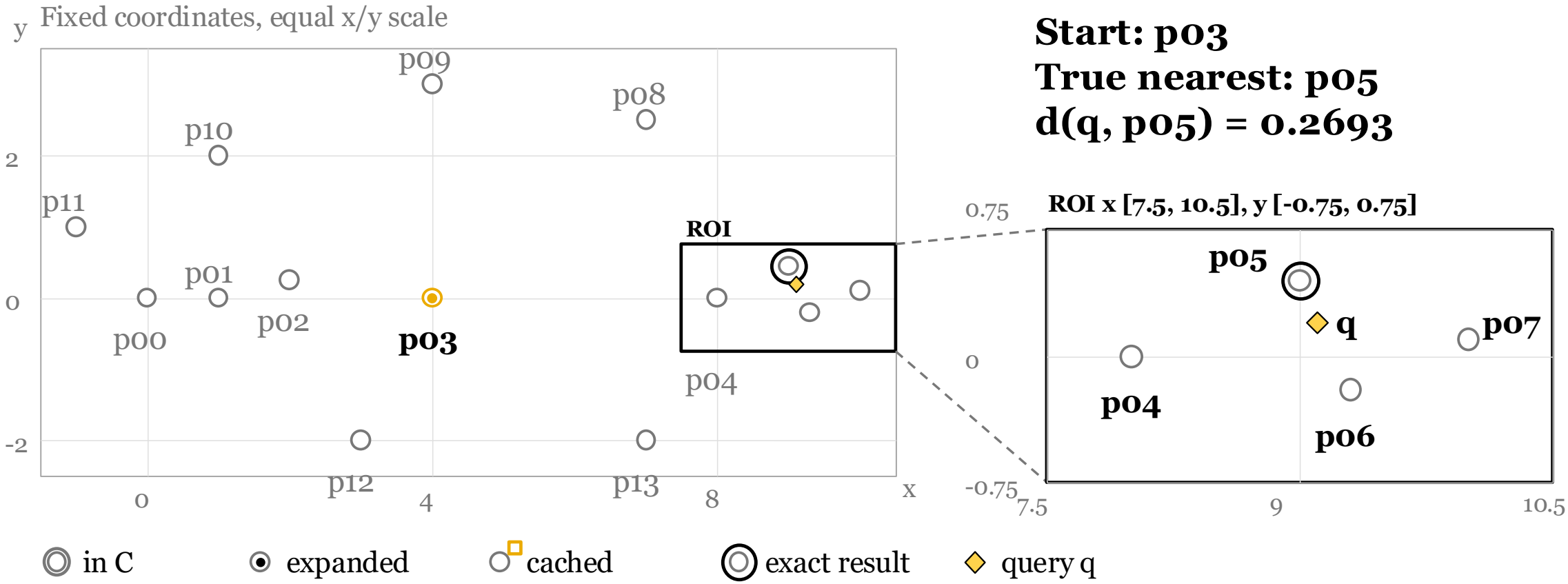
V: expanded history

Keeps every expanded point, including points that later leave C.

UNIVERSITY OF WATERLOO

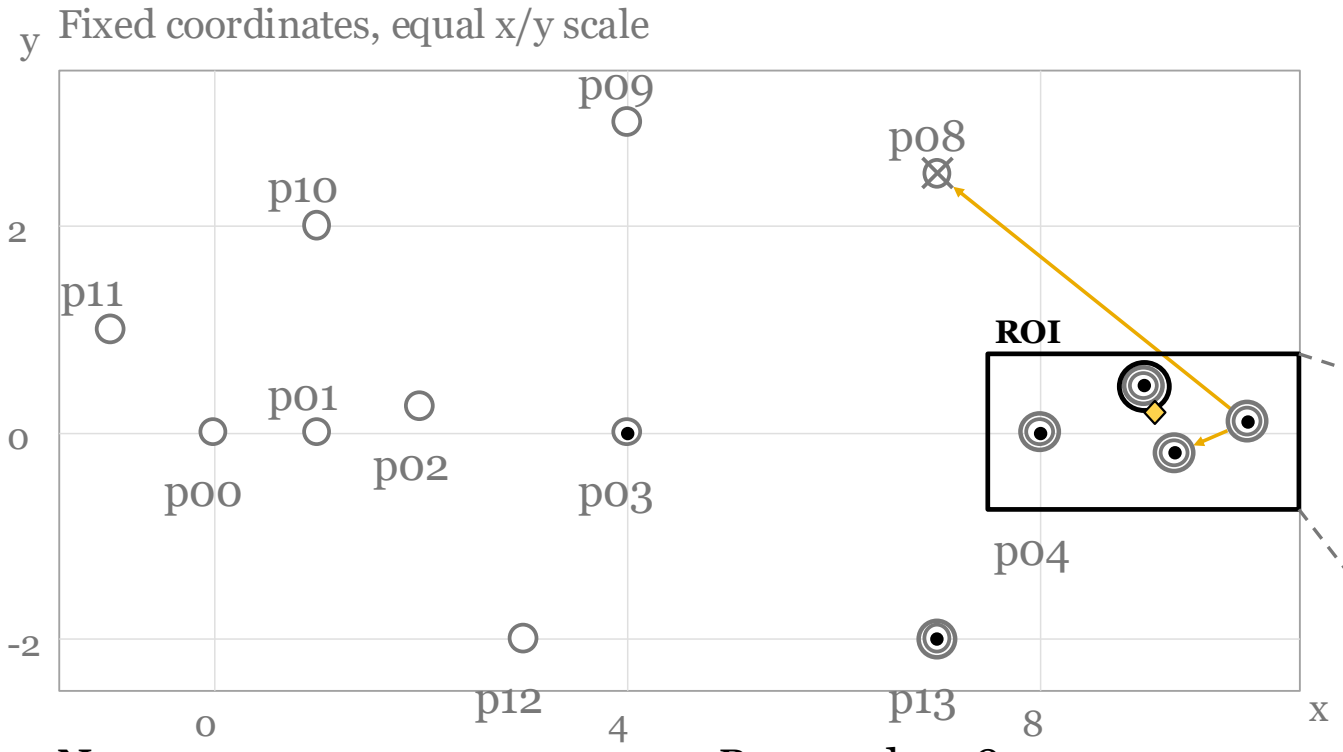
GreedySearch: example

Start p03. q = (9.1, 0.2). The true nearest stored point is p05.



GreedySearch expands the nearest available candidate

Exact distances. L = 5. k = 1. Expanded p07. Stop: every retained ID is expanded.



New: none

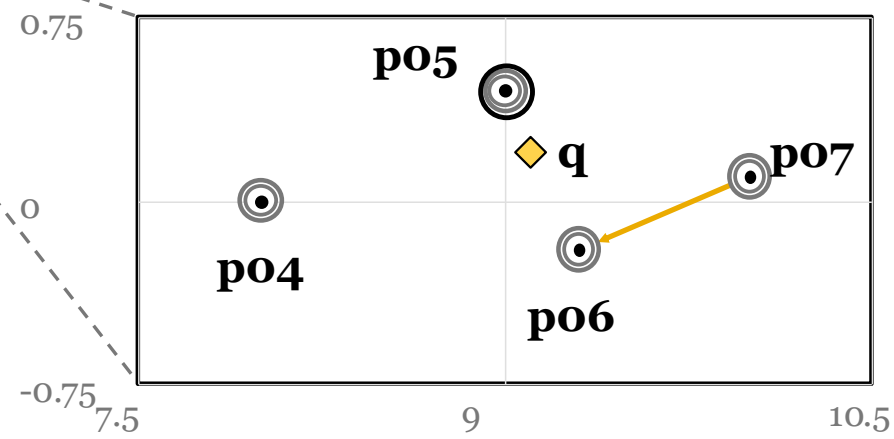
Removed: p08

C has 5 IDs. V has 6. Return p05.

C: exact query distance

p05	0.269	expanded
p06	0.447	expanded
p07	0.906	expanded
p04	1.118	expanded
p13	3.041	expanded

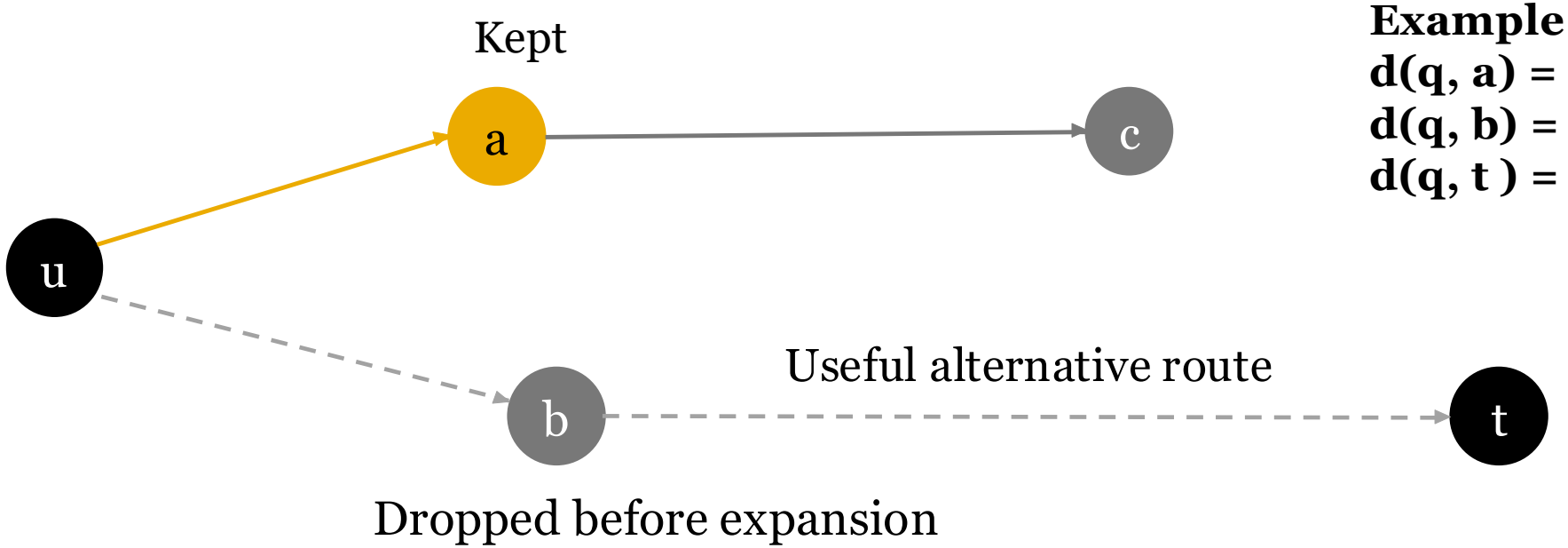
ROI x [7.5, 10.5], y [-0.75, 0.75]



Live demo: GreedySearch

A small candidate list can lose a useful route

A candidate can be dropped before its outgoing route is explored.



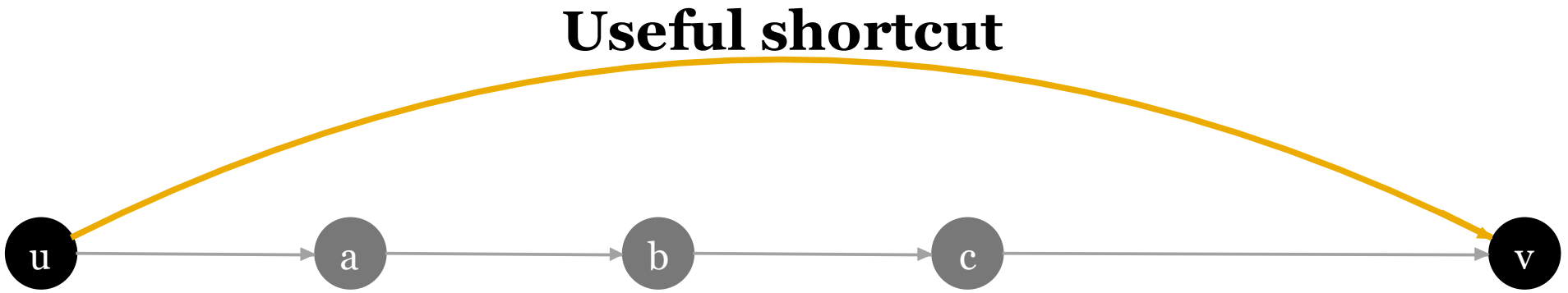
Example query scores
d(q, a) = 1.0
d(q, b) = 1.2
d(q, t) = 0.1

L_{build}: construction search

$L_{query} \geq k$: serving search

Useful shortcuts reduce intermediate expansions

A useful edge bypasses intermediate records on the way to the query region.



Intermediate records need not expand

Reach another region within a bounded outgoing degree

RobustPrune: pseudocode

Retain a useful neighbor, then remove candidates that it covers.

Algorithm 2: RobustPrune(p, V, α, R)

Data: Graph G; pivot p; candidates V; $\alpha \geq 1$; degree bound R.
Result: Rebuilt outgoing neighbors $N^+(p)$, with $|N^+(p)| \leq R$.

```
1  U ← (V ∪ N⁺(p)) \ {p}
2  N⁺(p) ← ∅
3  while U ≠ ∅ do
4      v ← arg min_{u∈U} d(p, u)
5      N⁺(p) ← N⁺(p) ∪ {v}
6      if |N⁺(p)| = R then break
7      U ← U \ {v}
8      for each u ∈ U do
9          if α d(v, u) ≤ d(p, u) then
10             U ← U \ {u}
```

Each selection

Choose nearest v to p.
Test all remaining u.
Repeat with survivors.

Cost per call

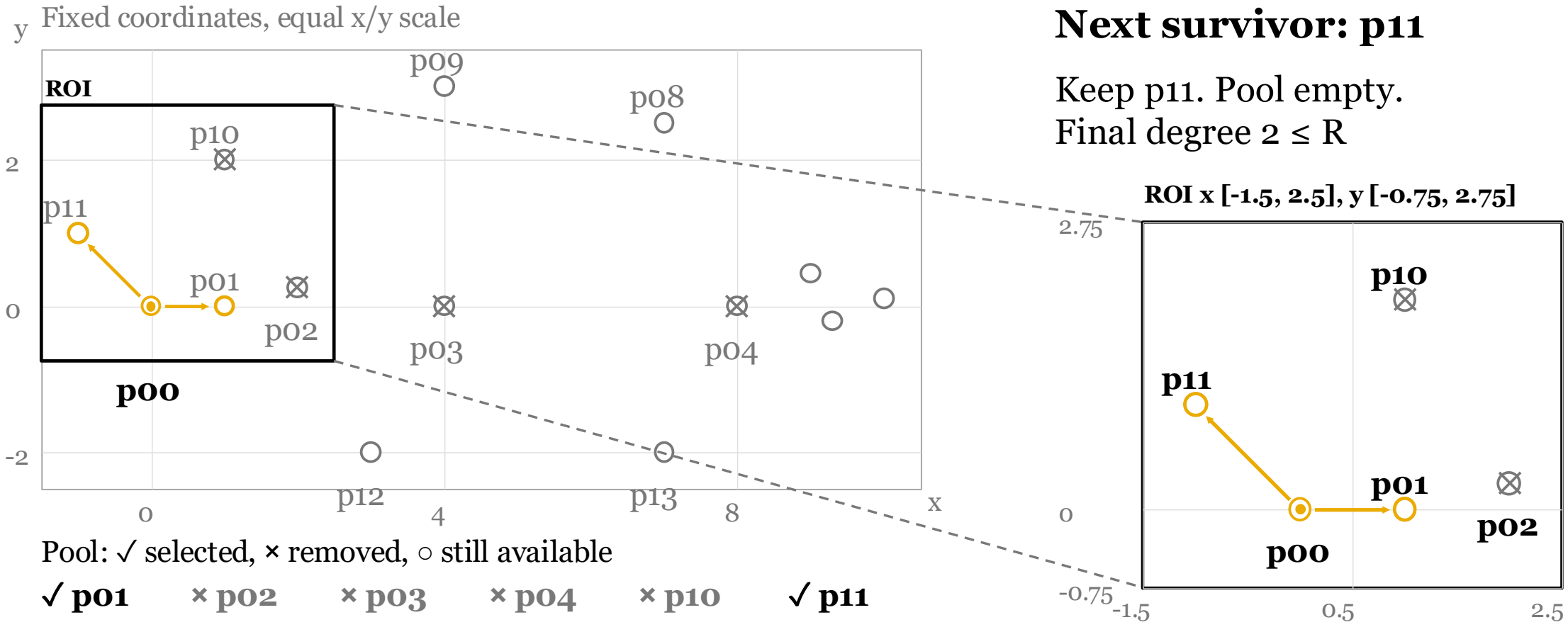
c initial candidates,
at most R selections.

$O(Rc)$ distance tests

UNIVERSITY OF WATERLOO

RobustPrune selects, scans, and repeats

R = 4. Union old neighbors with supplied candidates, then remove self and duplicates.



Pool: ✓ selected, × removed, ○ still available

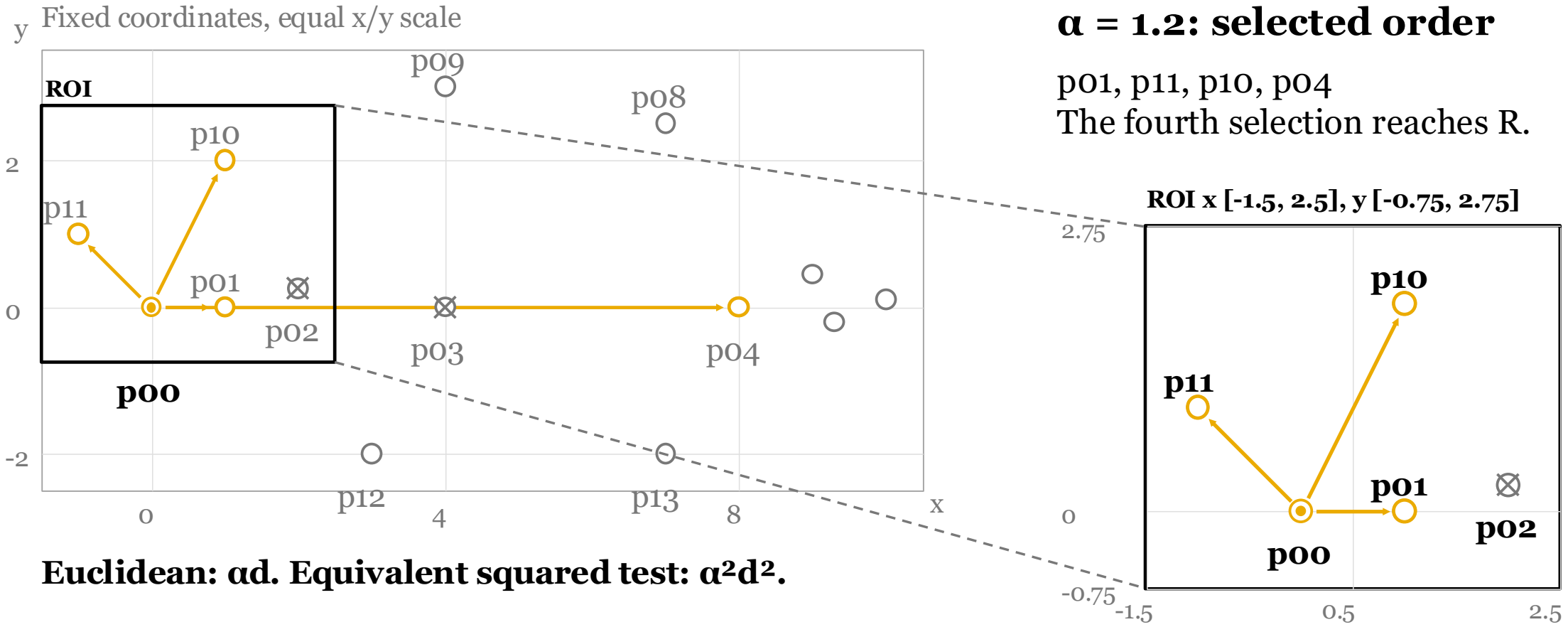
✓ p01 × p02 × p03 × p04 × p10 ✓ p11

Live demo: RobustPrune with α = 1

UNIVERSITY OF WATERLOO

A larger alpha makes pruning less aggressive

Same pivot p00, same candidate pool and R = 4.



α = 1.2: selected order

p01, p11, p10, p04
The fourth selection reaches R.

Euclidean: αd. Equivalent squared test: α²d².

Live demo: compare α = 1 and α = 1.2

Why the ideal graph makes progress

For a stored target t, pruning gives one of two useful outcomes.

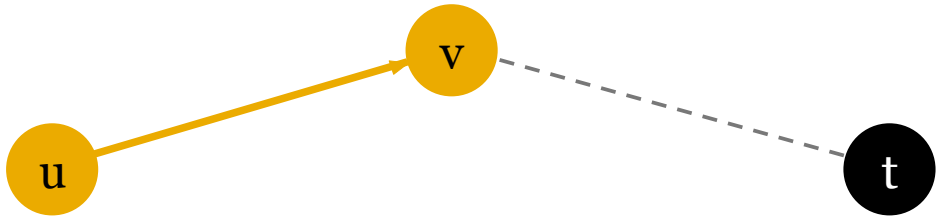
Ideal assumptions Stored target t. Full candidate pool. Exact distances. No degree limit, and $\alpha > 1$.

Case 1: keep t directly



One hop reaches t.

Case 2: keep v that removes t



$$\alpha\, d(v, t) \le d(u, t)$$

$$d(v, t) \le d(u, t) / \alpha$$

A logarithmic bound for the ideal graph

Stored target $t \in P$, exact distances, $k = L = 1$, and $\alpha > 1$

$$\text{For every } v \in P, \quad \text{run RobustPrune}(v, P \setminus \{v\}, \alpha, n-1)\,.$$

A retained neighbor makes progress

$$\alpha\, d(w,t) \le d(v_i,t) \quad \Longrightarrow \quad d(v_{i+1},t) \le \frac{d(v_i,t)}{\alpha}$$

Repeat the contraction

$$d(v_h,t) \le \frac{d(s,t)}{\alpha^h} \le \frac{\Delta}{\alpha^h}$$

A finite distance scale

$$\Delta = \max_{a,b \in P} d(a,b),$$

$$\delta = \min_{a \neq b \in P} d(a,b) > 0.$$

Before arrival, $d(v_h, t) \ge \delta$.

$$H \le 1 + \left\lfloor \log_\alpha \frac{\Delta}{\delta} \right\rfloor$$

For fixed $\alpha > 1$, logarithmic in Δ / δ.

Vamana: pseudocode

Search supplies the candidate pool; pruning controls the outgoing edges.

Algorithm 3: Vamana — two-pass schedule

Data: Dataset P; $\alpha \geq 1$; search-list size L; degree bound R.
Result: Directed graph G with $|N^+(p)| \leq R$ for every $p \in P$.

```
1  G ← a random directed graph with R out-neighbors per point
2  s ← medoid(P);   σ ← a random permutation of P
3  for a ∈ [1, α] in order do
4      for each p ∈ σ do
5          [C; V] ← GreedySearch(s, x_p, 1, L)
6          RobustPrune(p, V, a, R)
7          for each j ∈ N⁺(p) do
8              if |N⁺(j) ∪ {p}| > R then
9                  RobustPrune(j, N⁺(j) ∪ {p}, a, R)
10             else
11                 N⁺(j) ← N⁺(j) ∪ {p}
```

Two passes

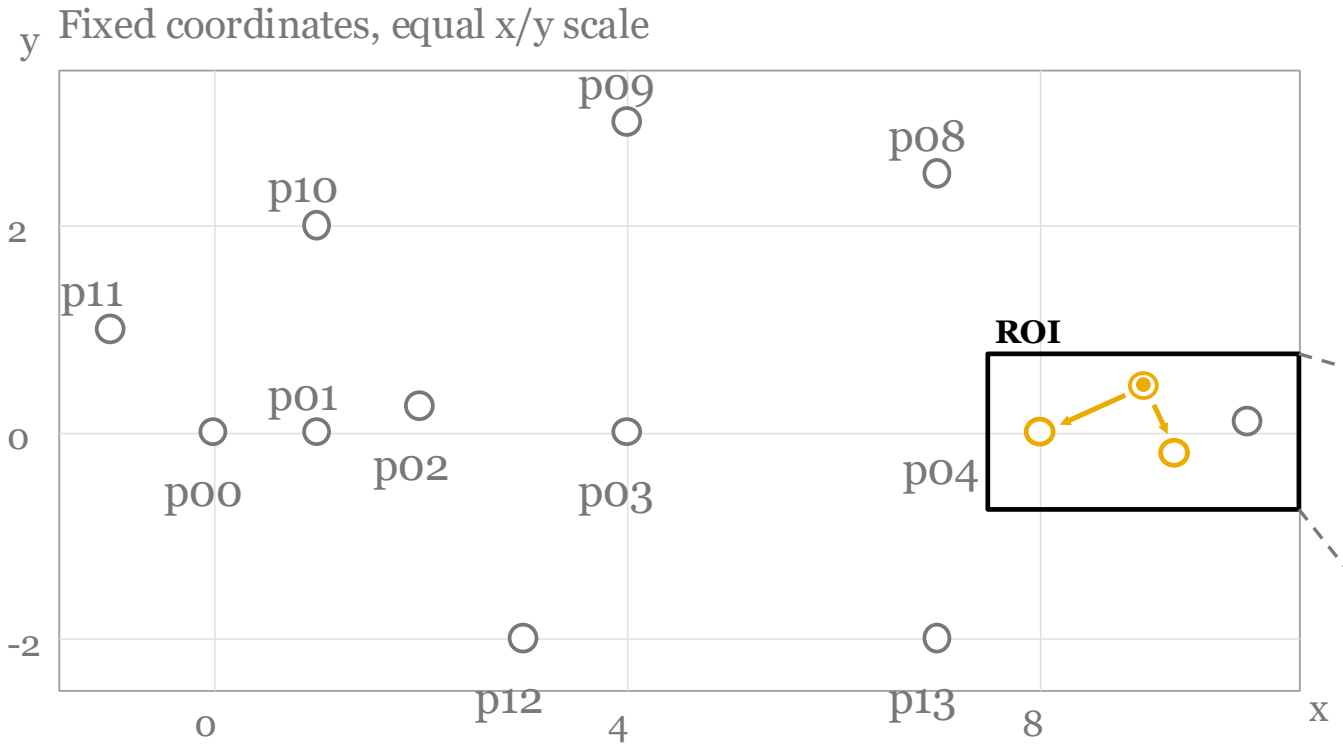
First α = 1,
then the chosen α.
The graph persists.

Reverse proposals

Adding p to a neighbor
may overflow R.
Prune that union.

Refining one stored point

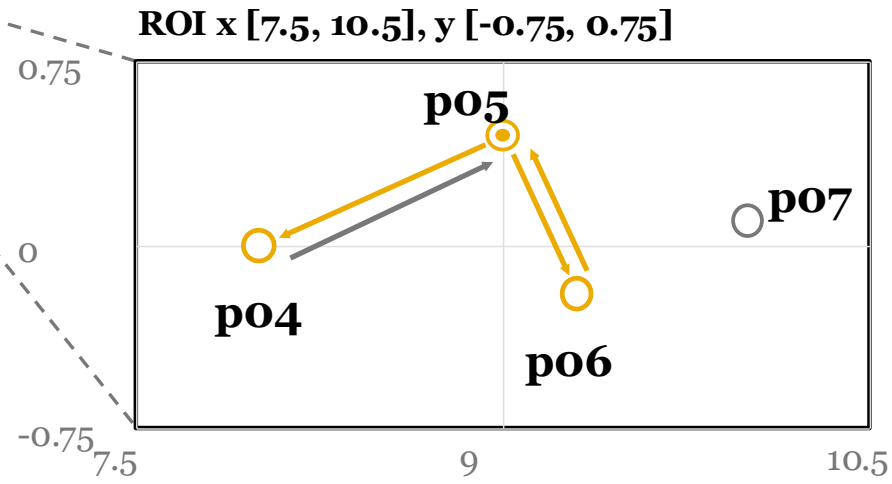
Processing p05. The construction query is its stored vector $x_{p05} = (9, 0.45)$.



Reverse proposal uses set union

4 Propose reverse edges

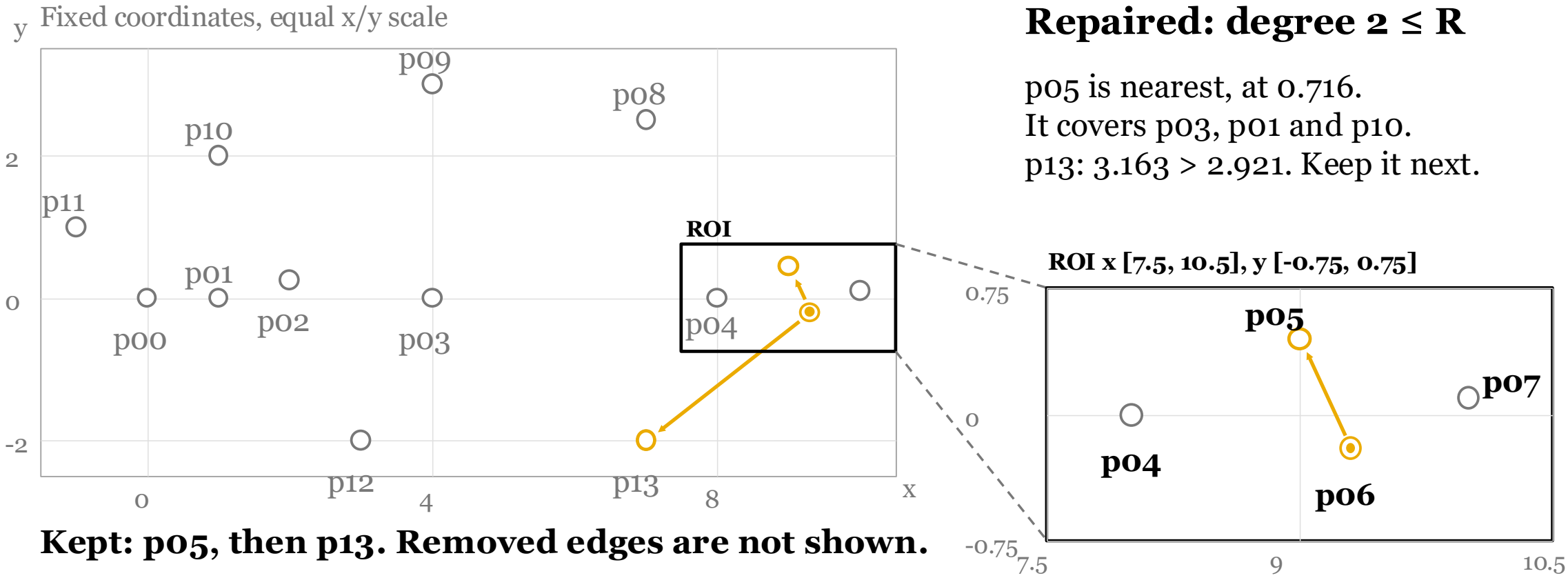
Kept p05 → p06
Proposed p06 → p05
p04 → p05 already exists.



Reverse-edge example

Reverse-edge repair respects the degree bound

Repairing p06 with α = 1 and R = 4. The reverse proposal creates 5 candidates.



Repaired: degree 2 ≤ R

p05 is nearest, at 0.716.
It covers p03, p01 and p10.
p13: 3.163 > 2.921. Keep it next.

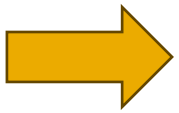
Kept: p05, then p13. Removed edges are not shown.

Two passes refine the same graph

Every stored point is refined again with the user's chosen α.

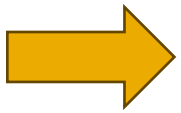
Initial graph

All stored points
Random outgoing edges
Medoid entry



Pass 1

Visit every point
α = 1
Search, prune,
reverse repair



Pass 2

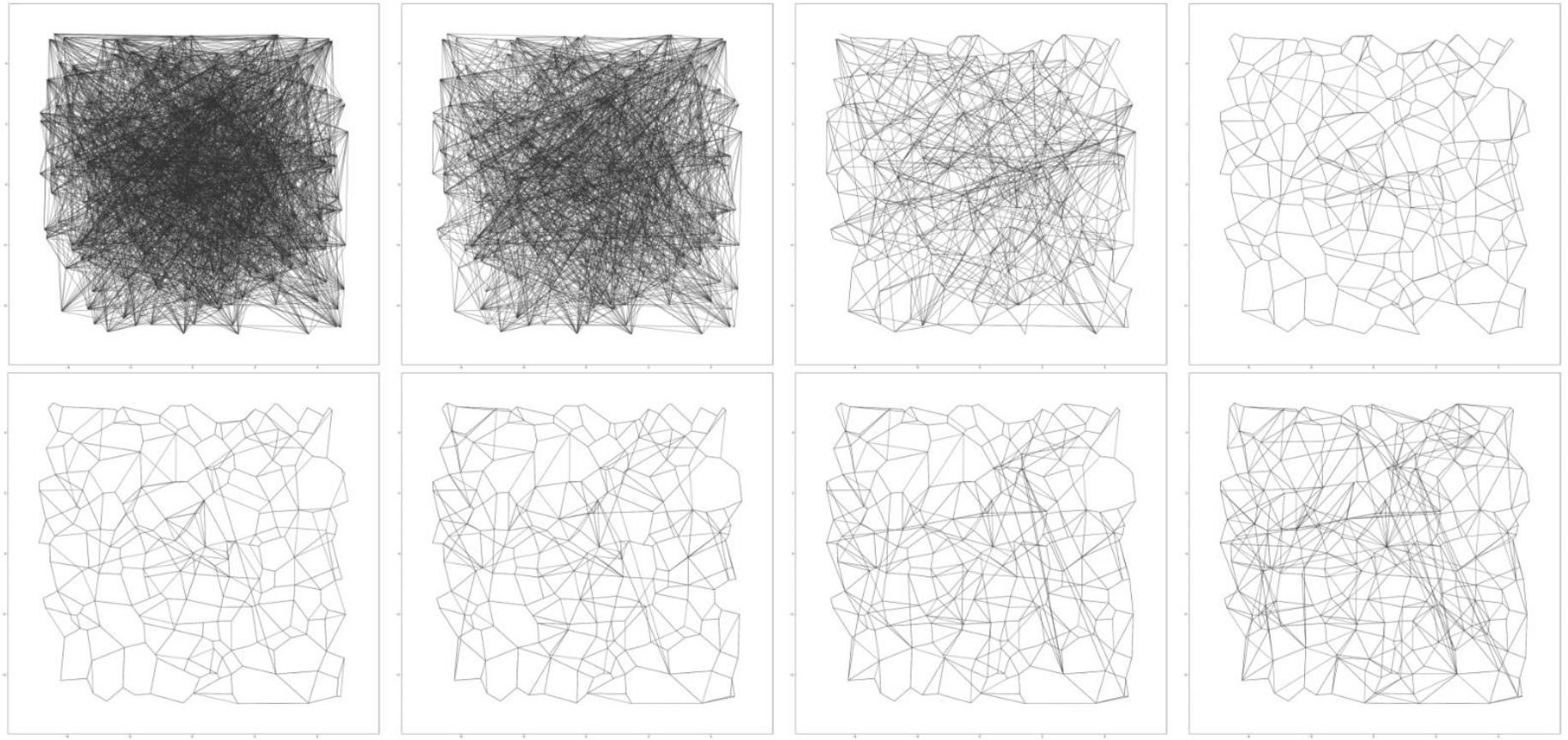
Visit every point again
User's α, here 1.2
Continue the pass 1 graph

The graph persists across passes.

Live demo: two passes

Vamana builds local and longer-range routes

Original Figure 1: construction adds local and longer-range routes.



HNSW, NSG, and Vamana

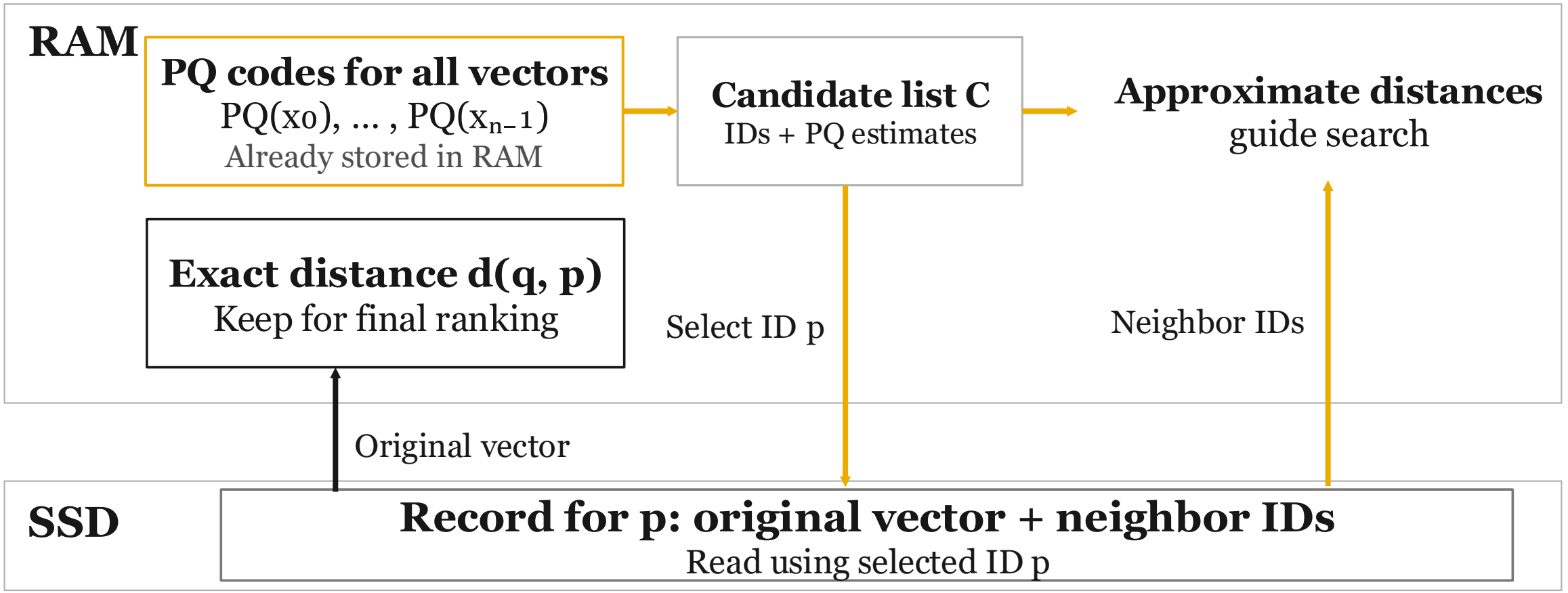
How each method builds a navigable graph and keeps longer routes.

Graph	Structure	Construction and neighbor choice	Long-range navigation
HNSW	Multiple layers (top-down search)	Incremental insertion. Heuristic selection from final search shortlist.	Upper layers supply long-range shortcuts.
NSG	Single graph (approx. medoid)	Approximate k-NN graph. MRNG-like pruning of visited candidates; DFS connectivity repair.	Sparse spreading-out links, no hierarchy.
Vamana	Single graph (fixed medoid)	Random seed and two passes. RobustPrune uses expanded V with tunable α.	$\alpha > 1$ preserves more long-range alternatives.

Vamana keeps useful routes in one graph without a k-NN seed or hierarchy.

RAM guides the next SSD request

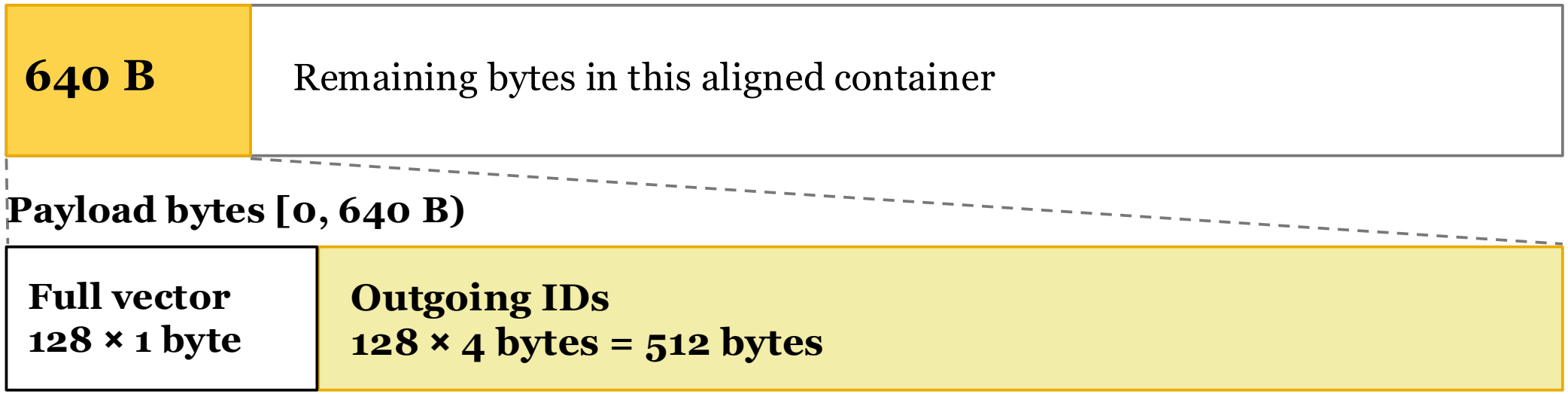
PQ codes select which SSD record to read. Exact scores require the original vector.



A record stores the vector and its neighbors

SIFT example: 128 uint8 coordinates and up to 128 outgoing IDs

One aligned 4 KB container = 4,096 bytes



Co-residence lets the query score p without a separate vector read.

Beam width trades reads for rounds

L = 5, no cache. Predict: does W = 2 reduce both reads and rounds?

	Round 1	Round 2	Round 3	Round 4	Round 5	Round 6	Round 7
W = 1	**p03**	**p13**	**p04**	**p05**	**p06**	**p07**	**p08**

7 reads, 7 rounds

W = 2	**p03**	**p13**	**p04**	**p07**	**p06**
		p09	**p08**	**p05**	

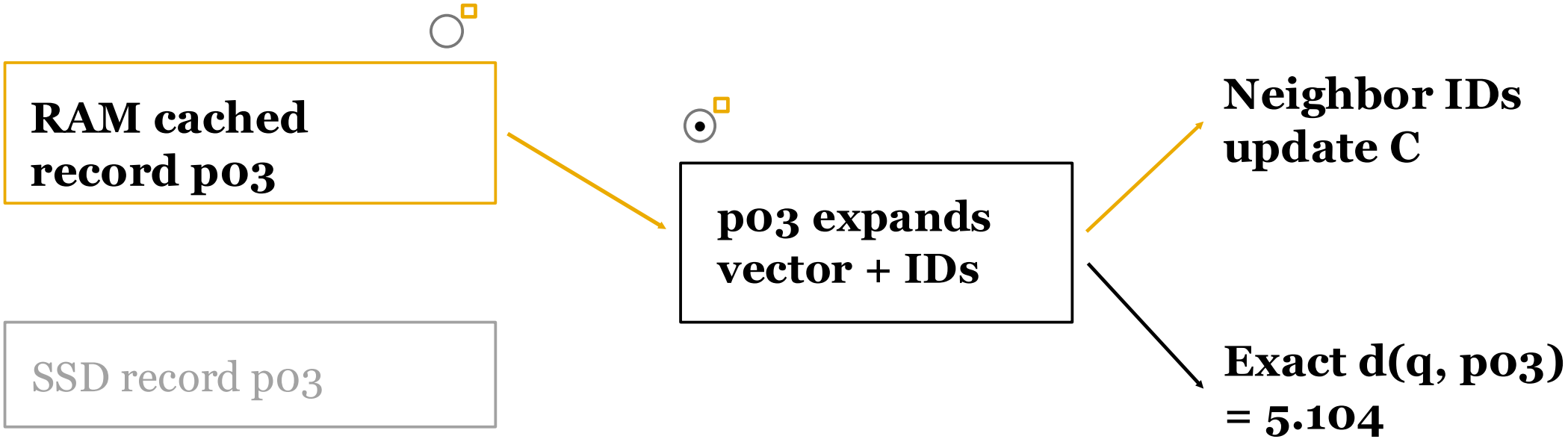
Reads increase, but rounds decrease. p09 is the extra read.

8 reads
5 rounds

+1 read
−2 rounds

Caching removes reads without removing expansion

L = 5, W = 2. The graph, query and parameters stay fixed. Only p03 cache membership changes.



7 SSD reads, 4 I/O rounds, 8 expanded records

in C expanded cached exact result query q

Live demo: serial reads, beam search, and cache

Exact scores survive removal from the candidate list

Original query outside-C rescue. q = (9.1, 0.2). L = 2, W = 1, no cache.

Search list C

p06	0.922
p07	0.922

p05 has left C.

Stored exact score

p05

d(q, p05) = 0.2693

p05 wins among expanded records

in C expanded cached exact result query q

PQ can change the candidate order

q = (9.1, 0.2). Compact-code distances can differ from exact distances.

Exact distance		Compressed guidance	
p05	0.269	**p06**	0.922
p06	0.447	**p07**	0.922
p07	0.906	**p05**	1.118

Answer: p06, p07, then p05.
p06 and p07 tie. Ascending ID resolves it.

An unread vector cannot be reranked

q = (9.1, 0.2). L = 1, W = 1, no cache. Approximate ties use ascending ID.

Expanding p04 discovers p05

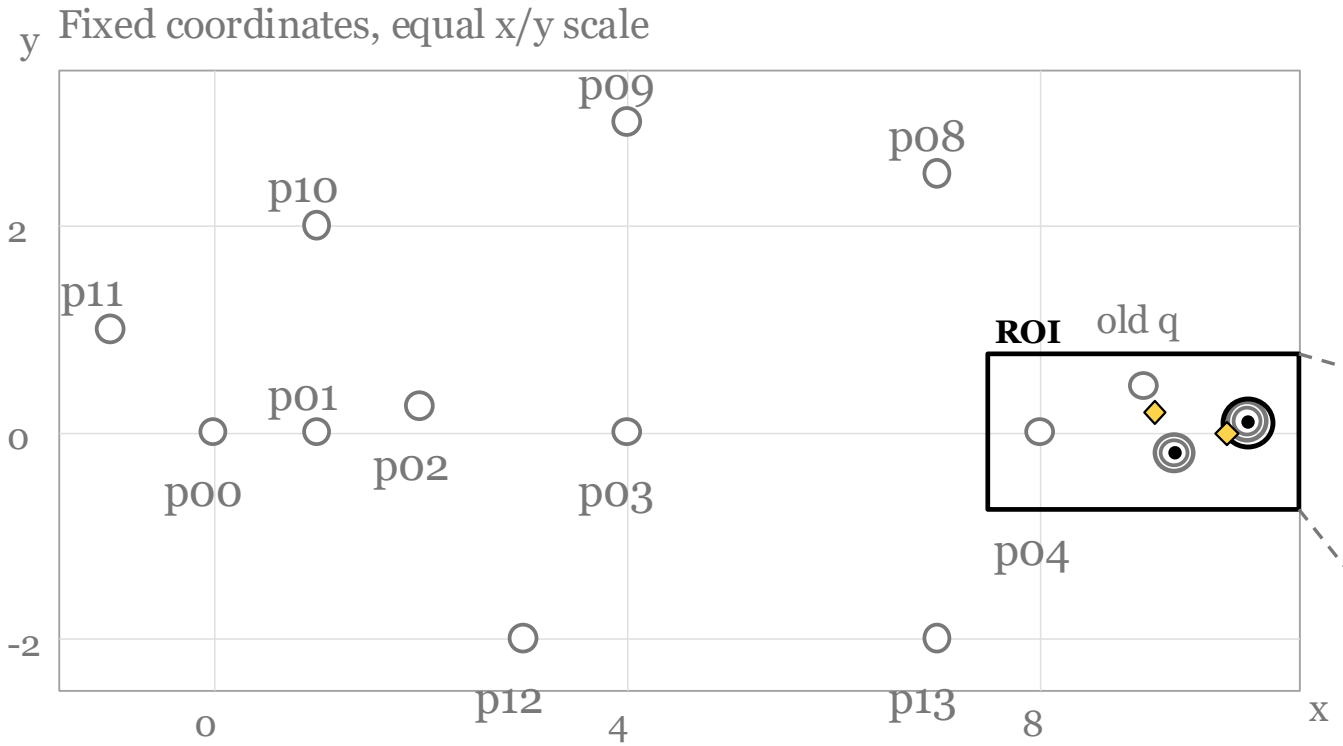
⊙ p04: 1.118 = ⊗ p05: 1.118

The true nearest point is p05: exact d = 0.2693

p04 wins the ID tie. p05 drops before expansion.
No original p05 vector is read, so it cannot be reranked.

Example: search and exact reranking

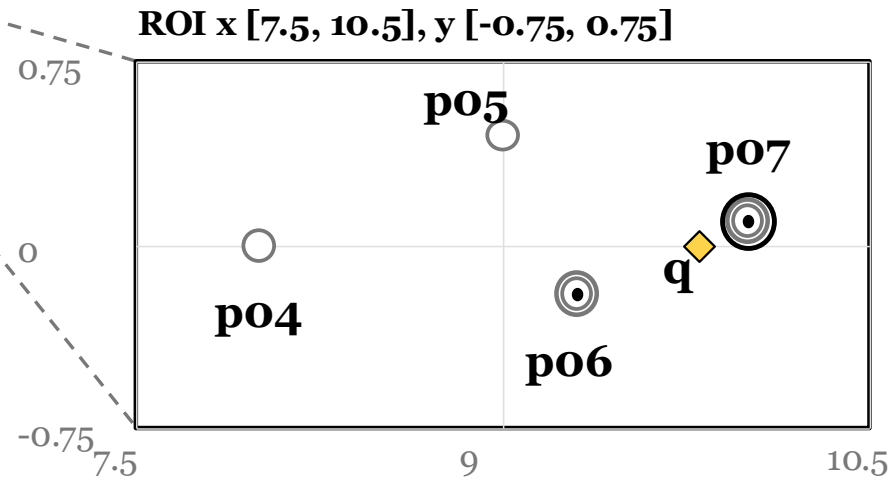
Changed-query tie correction. q = (9.8, 0). L = 2, W = 1, no cache.



p07 wins exact ranking and remains in final C.

New query distances

Point	Exact d	Code d
p06	0.539	0.200
p07	0.224	0.200



Live demo: changed-query tie

Exact-score rescue

Accounting for query work

Let m be expanded records and u be distinct IDs scored with PQ.

Neighbor IDs	At most mR_{final}
PQ scores	O(uM), with M subquantizers
Exact scores	O(md), with d full dimensions
Candidate management	Additional queue or list operations

RAM codes: nB bytes

SSD requests and dependent rounds remain separate costs.

Why a one-shot build needs more RAM

Building changes graph edges using full vectors. Serving navigates a finished graph on SSD.

ONE-SHOT CONSTRUCTION

- **Original vectors**

Exact distances for selecting neighbors

- **Mutable graph**

Outgoing edges are repeatedly added and pruned

- **Build candidate sets**

Workspaces for GreedySearch and RobustPrune

DISKANN QUERY SERVING

- **PQ codes for all vectors**

Approximate distances guide the next read

- **Cached full records**

A subset avoids repeated SSD reads

- **Active query state**

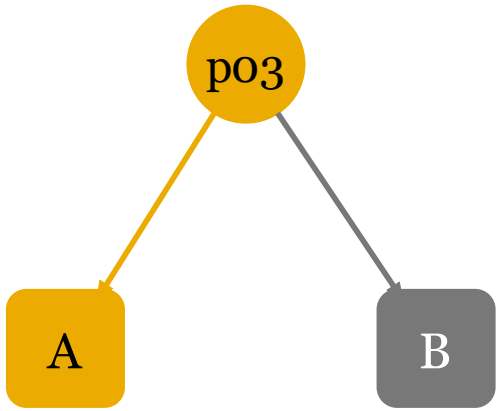
Candidate lists for concurrent searches

One-shot construction needs full data in RAM, but serving does not.

Overlapping partitions produce one graph

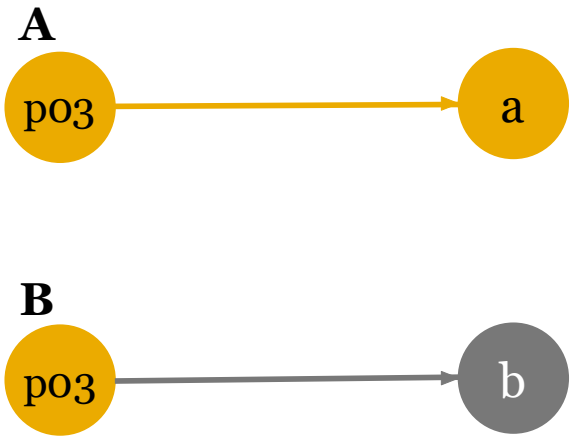
Each point joins ℓ nearby partitions. Local edges are united by original ID.

1 Assignment



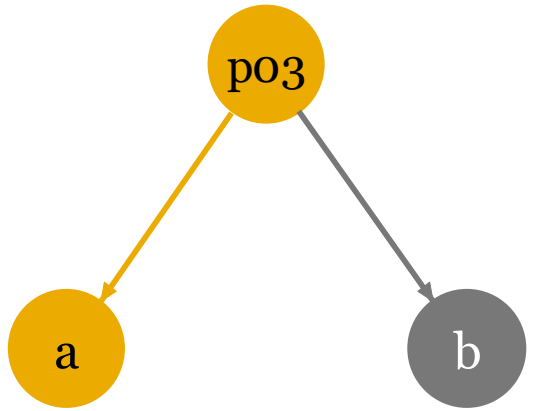
$\ell = 2$ memberships

2 Local graphs



Local cap: R_{local}

3 ID union



p03 connects to a and b

$$R_{final} \leq \ell \, R_{local}$$

Experimental setting and recall measures

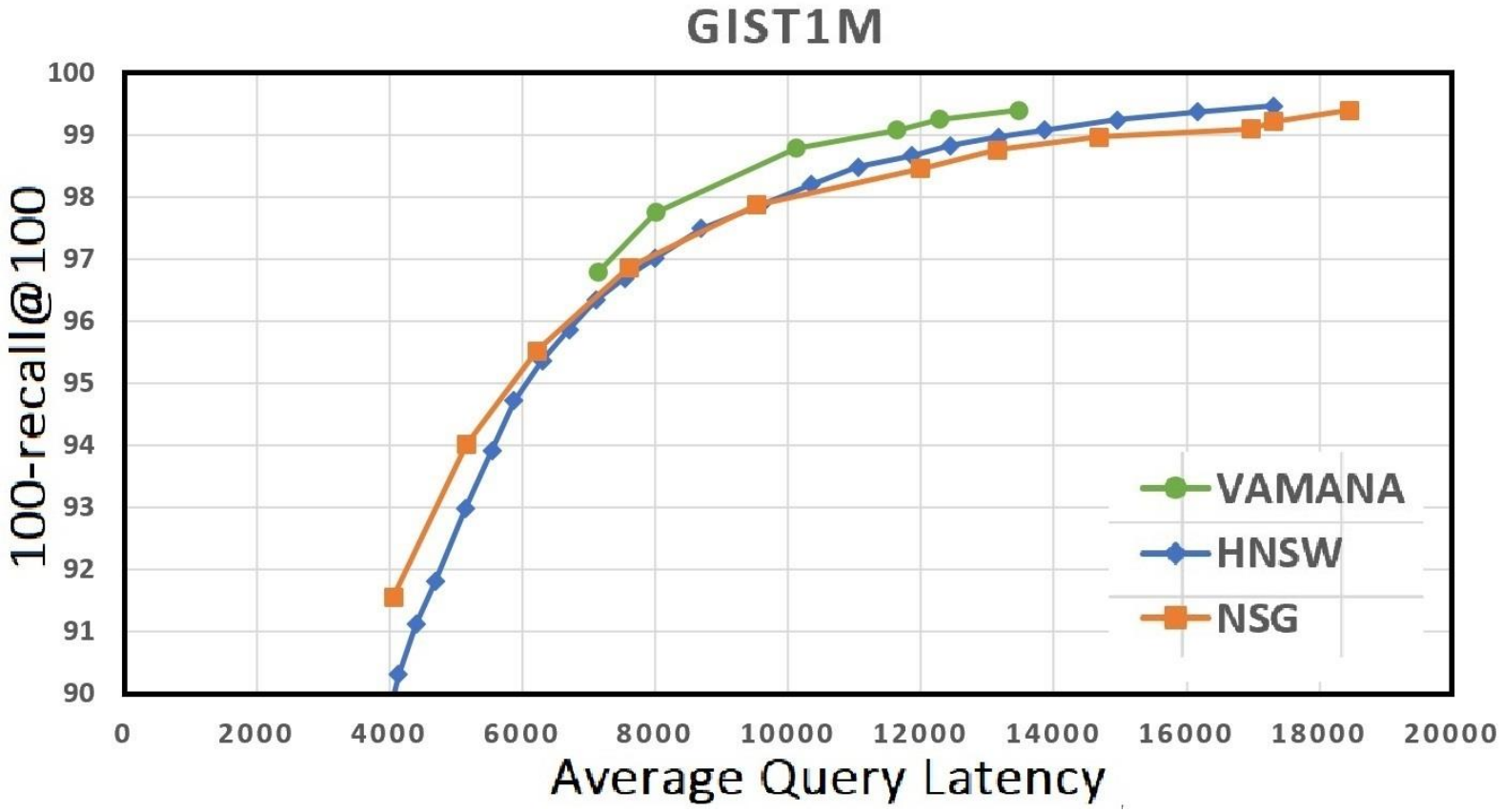
Serving: 16 cores, 64 GB RAM, two 1 TB SSDs in RAID 0

Experiment	Dataset	Recall	Unit
RAM graphs	SIFT1M, GIST1M, DEEP1M	100-recall@100	µs
Navigation rounds	SIFT1M	5-recall@5 = 98%	Rounds
SSD search	SIFT1B, DEEP1B	1-recall@1	ms

1-recall@1: fraction of queries returning the true nearest point first.

Vamana is competitive in RAM

GIST1M, 960 dimensions. 100-recall@100 versus average latency in microseconds.



Near 99% recall

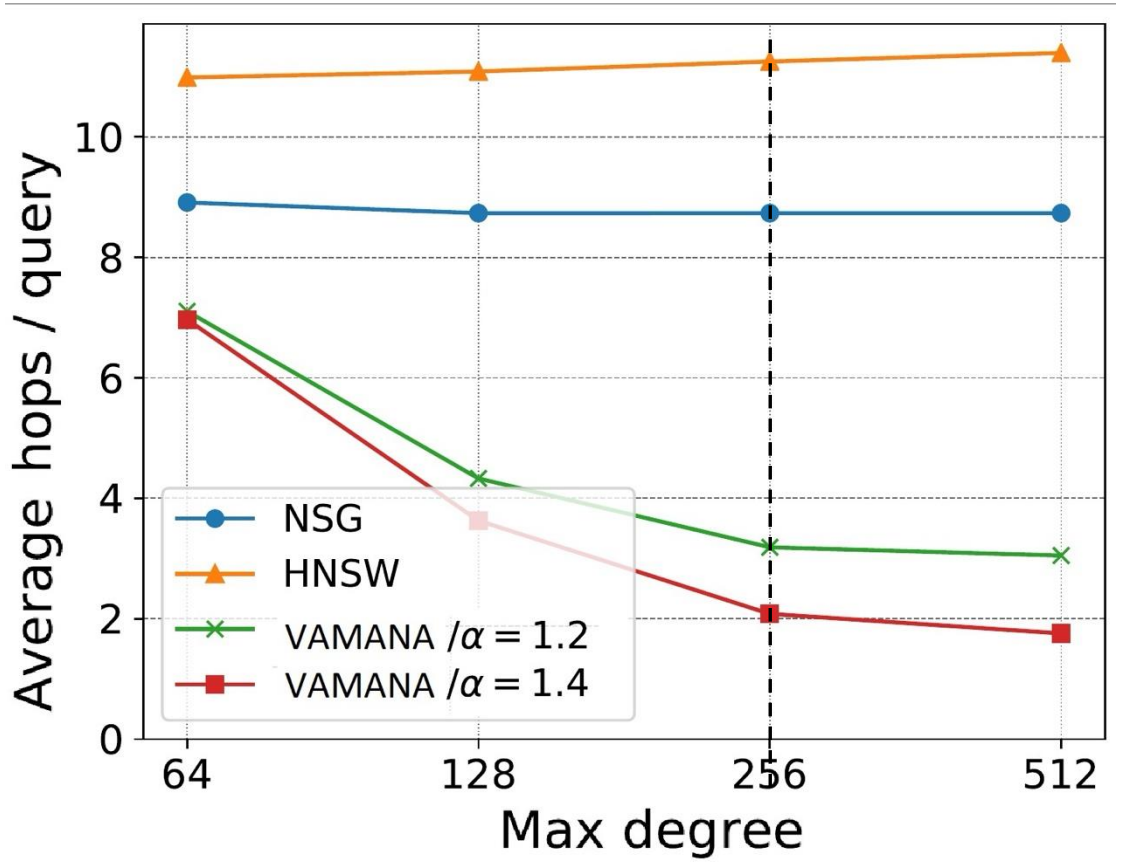
Vamana lies left of the other curves here.

Separate dataset: DEEP1M

Build	Seconds
Vamana	149
HNSW	219
NSG	480

Vamana reduces dependent rounds in this experiment

SIFT1M at 98% 5-recall@5. Compare methods at the same maximum degree.



At R = 256

Vamana has fewer rounds here.
A round can contain several reads.

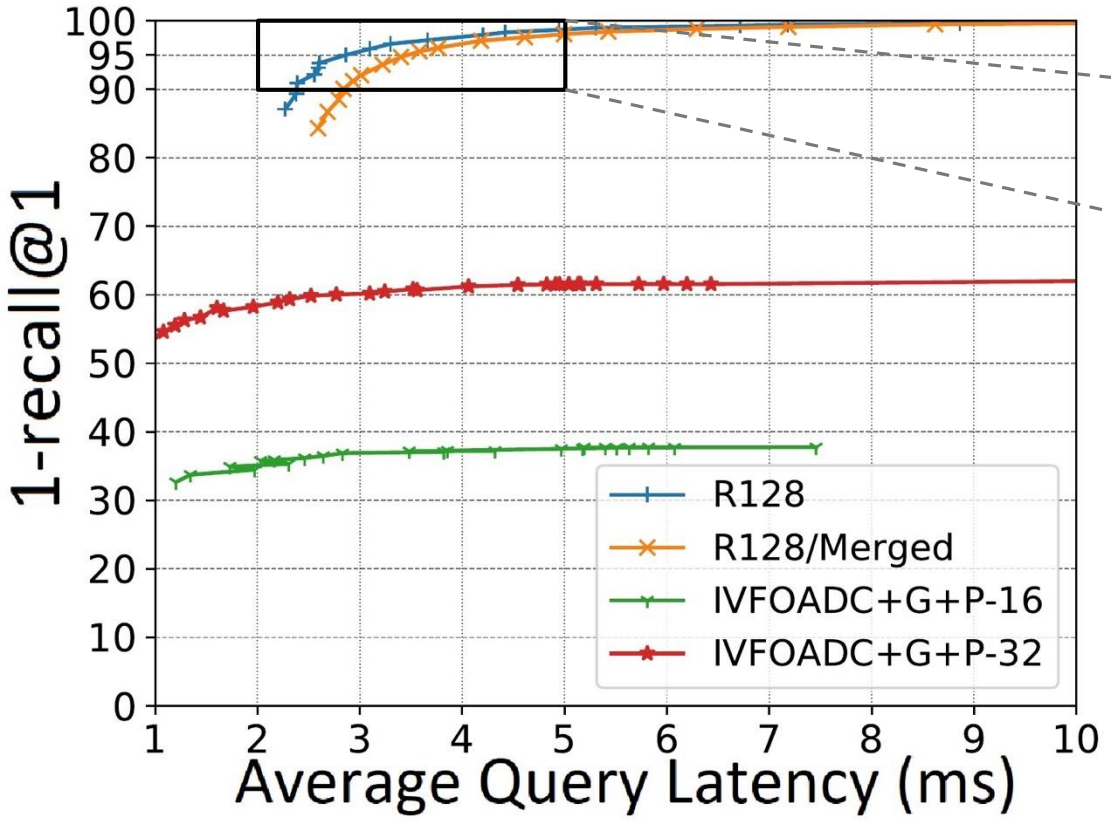
Cache policies in the paper

HNSW: upper layers
NSG / Vamana: first 3 BFS levels

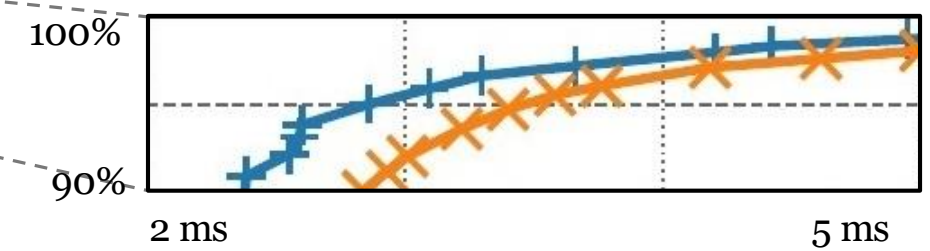
The paper does not establish a matched-byte cache comparison.

SIFT1B: high recall with millisecond latency

SIFT1B. 1-recall@1 versus average latency in milliseconds.



Source ROI: 2–5 ms, 90%–100%



Authors report

98.68% below 5 ms, one shot
Merged: ≤20% extra latency
at matched recall here

The merged build stays below 64 GB

≈1,100 GB

One-shot peak RAM

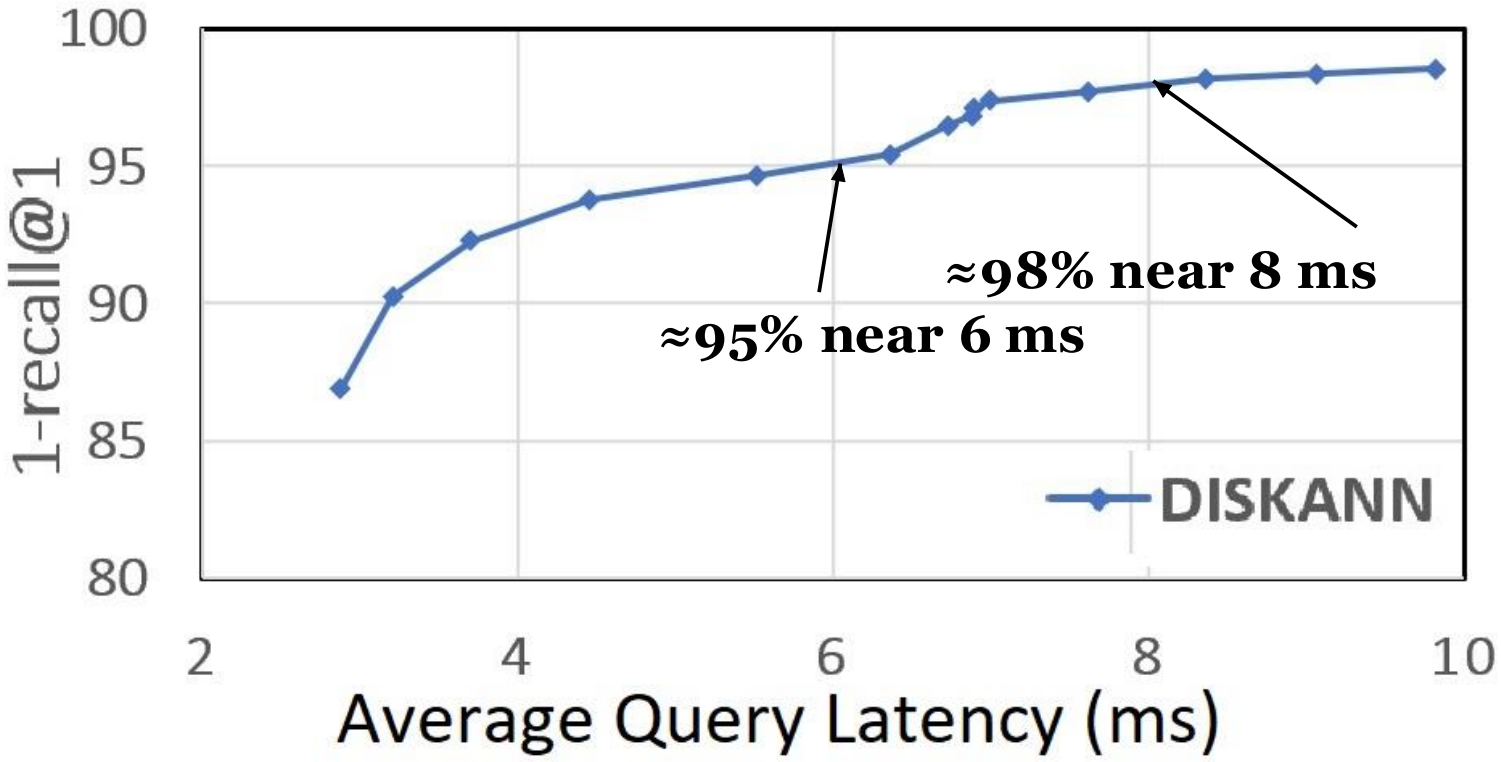
<64 GB

Merged peak RAM

SIFT1B build	One shot	Merged
Machine RAM / time	1,792 GB / ≈2 days	64 GB / ≈5 days
Build settings	R=128, L=50, α=1.2	40 clusters, ℓ=2 local R=64, L=50, α=1.2
Average final degree	113.9	92.1

DEEP1B has a different recall-latency curve

Merged graph, 96 dimensions.



The main contribution

RAM chooses which records to read. Each SSD read supports navigation and scoring.

RAM

PQ codes rank candidates

Beam search overlaps SSD reads.
Cache hits avoid some reads.

SSD

Vamana graph + original vectors

Read only selected records during search.

Offline build

Build overlapping local graphs.
Merge edges into one serving graph.

After an SSD read

Neighbor IDs: new candidates.
Original vector: exact distance.

Limitations of the 2019 DiskANN design

The 2019 system targets static, unfiltered ANN. Search decisions still have costs.

Search quality and filters

- PQ scores can drop useful candidates before SSD reads. Increasing L may recover them, but adds search work.
- The original graph does not model filters. Excluding points may break useful routes.

Build and I/O costs

- Billion-point construction takes substantial time. The 2019 paper does not address streaming updates.
- Higher recall may need more SSD rounds. Wider beams can add reads, while bigger caches use RAM.

Later systems add online maintenance and label-aware search.

Later work extends updates and filtering

These capabilities require algorithms beyond the 2019 static index.

Later paper	Added capability	Remaining work or scope
FreshDiskANN 2021	Support inserts and deletes during search	Keep recent updates in RAM and periodically merge them into the SSD index
IP-DiskANN 2025	Reduce the cost of maintaining a changing graph	Repair edges in place, with occasional cleanup of deleted links
Filtered-DiskANN 2023	Search for nearest neighbors that match a filter	Use label information during graph construction and search

Discussion

1 When should we stop searching?

Stop early if recent reads find no closer points and few promising candidates remain.
This saves time, but the search may still miss a better point.

2 How should we use a limited number of SSD reads?

Read several points together when they all look promising.
If one path keeps finding closer points, spend more reads along it.

3 What if the data changes?

Repair connections around added or removed points first.
Rebuild more of the graph only if searches become slower or less accurate.