When Semantic Search Breaks

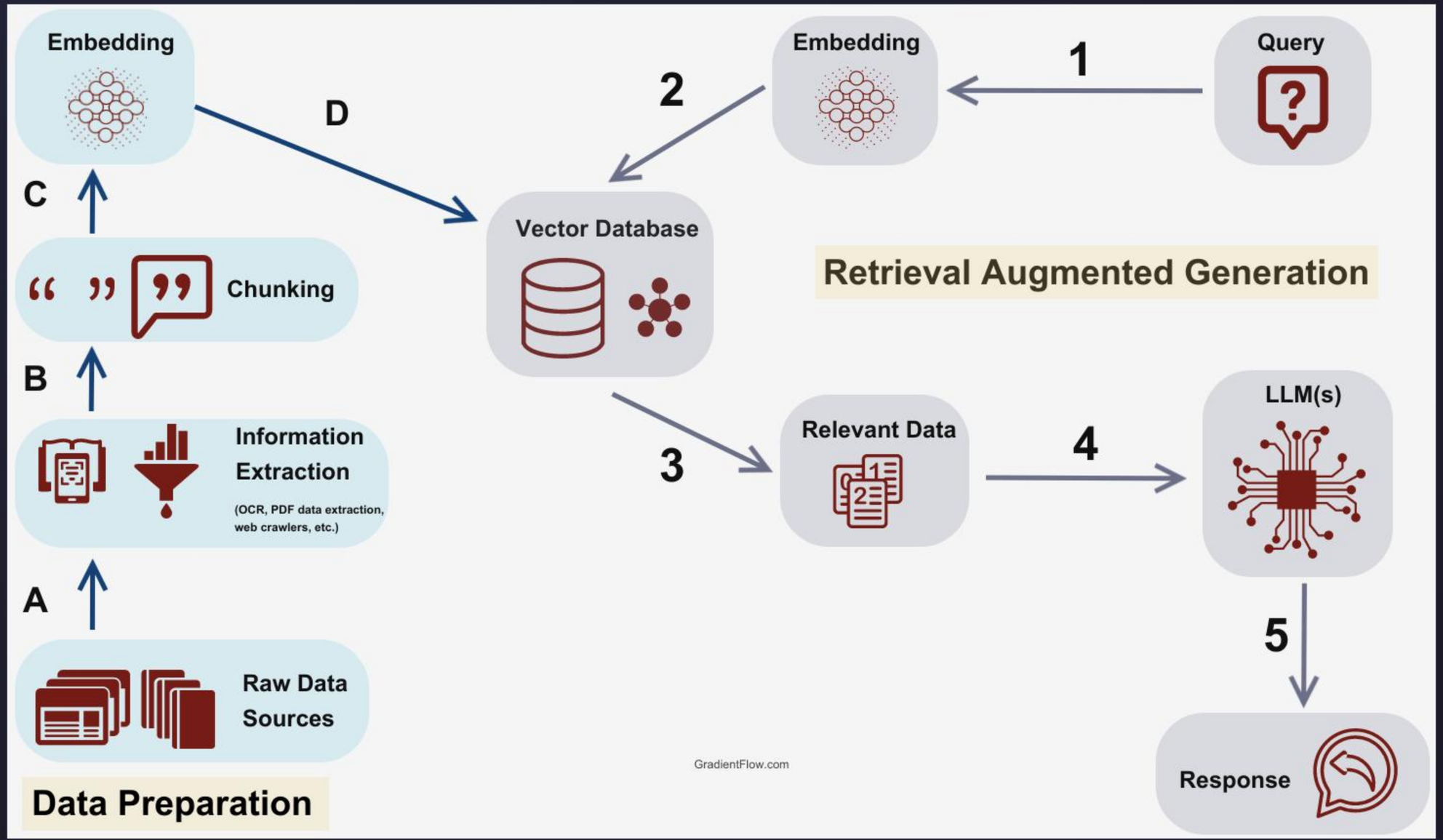
RAM Walls · Silent Failures · Architecture Decisions

Jeevan

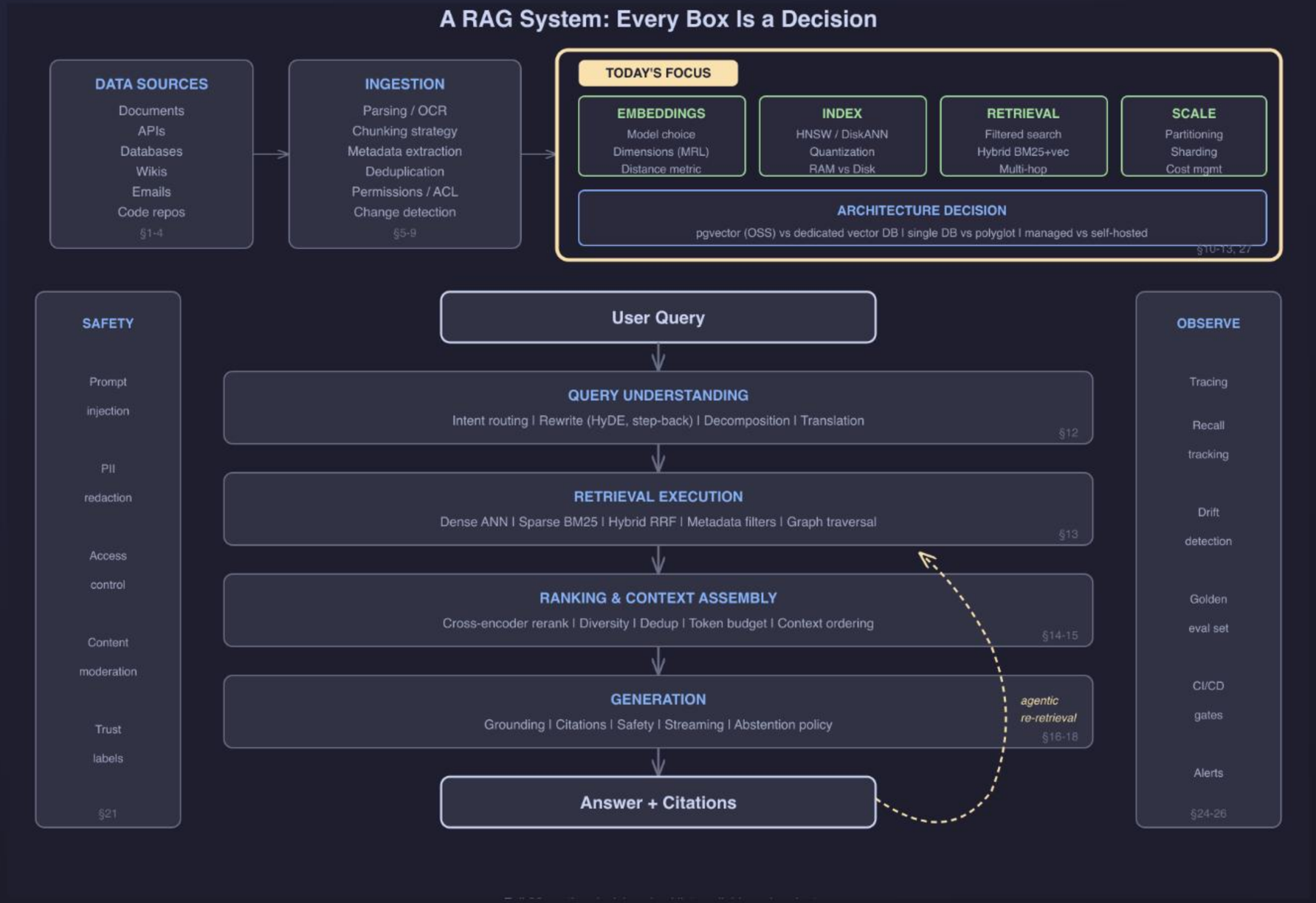
Open Source Summit Korea · August 2026

Entain

What Is RAG?


Embedding
D
Embedding
2
1
Query
C
Vector Database
Retrieval Augmented Generation
Chunking
B
Information Extraction
(OCR, PDF data extraction, web crawlers, etc.)
3
Relevant Data
4
LLM(s)
A
5
Raw Data Sources
Response
GradientFlow.com
Data Preparation

A RAG System: Every Box Is a Decision



The Scaling Path

The Scaling Path

We start from first principles and build up.

100K vectors

Everything works

Don't optimize.
Just ship.

SHIP IT

→

1-10M

Proper indexes

Need filters
Need MRL (256d)

GETTING REAL

→

10-50M

RAM wall hits

Need compression
Need quantization

CFO NOTICED

→

50M+

Disk indexes

Need DiskANN
Need observability

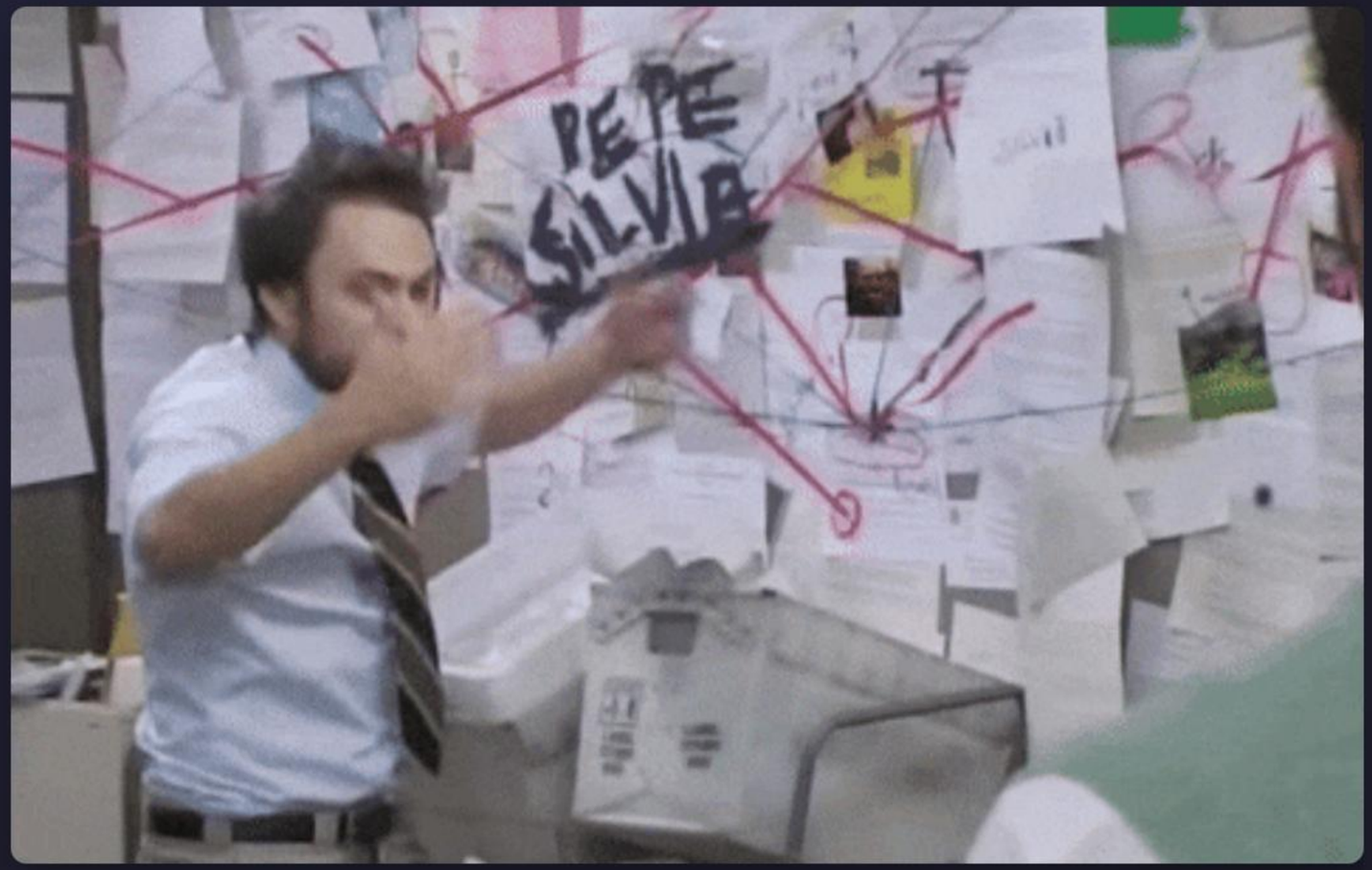
SILENT FAILURES

Why This Talk, Why Now

```
Month 1:  "Let's add semantic search!"
          → 100K vectors, works great ✅

Month 4:  "Scale to all our docs"
          → 10M vectors, still fine ✅

Month 8:  "Enterprise rollout"
          → 100M vectors, RAM bill explodes 💸

Month 9:  "Add per-tenant filtering"
          → recall silently drops to 40% 📉

Month 10: "Maybe we need a separate vector DB?"
          → now syncing two databases forever 🔄
```

Same pattern. Same order. Every team.

This talk: the math, the tools, and the decisions — in plain English.



What We'll Cover

First Principles

Embeddings, distance, why approximate works

The RAM Wall

Where scale breaks and what to do

Three Compression Levers

Dimensions → Bits → Disk

Silent Failures

Filtered search + recall drift

Architecture Decisions

Benchmarks, trade-offs, decision matrix

DEMO

Two Sentences → Numbers → Similarity

✅ High Similarity

Same meaning, no shared words

`"cancel my subscription"`

`"I want to unsubscribe"`

→ ~68%

❌ Low Similarity

Same word, different meaning

`"the bank is closed for the holiday"`

`"the river bank was steep and muddy"`

→ ~26%

384 dimensions per sentence. That's what costs 920 GB at scale.

How Do Computers Compare Text?

Computers don't understand words.

```
"I love fries"   vs   "Fries are great"
```

👴 Human → instantly similar.

💻 Computer → two different strings.

The fix: turn text into numbers that capture meaning.

```
"I love fries"     → [0.2, 0.8, 0.1, ...] 384 numbers
"Fries are great"  → [0.3, 0.7, 0.2, ...] 384 numbers
"The sky is blue"  → [0.9, 0.1, 0.8, ...] 384 numbers
```

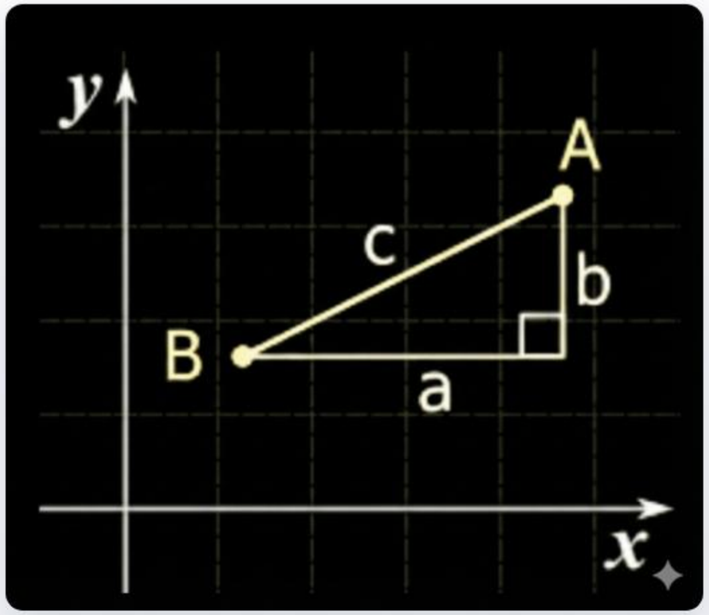
Similar meanings → similar numbers.

That's an embedding. A GPS coordinate for meaning — close meanings, close numbers.

How Do We Measure "Close"?

Distance between two points — already familiar:

```
Point A = (x₁, y₁)    Point B = (x₂, y₂)

Distance = √((x₂−x₁)² + (y₂−y₁)²)
```



Same idea, just more dimensions:

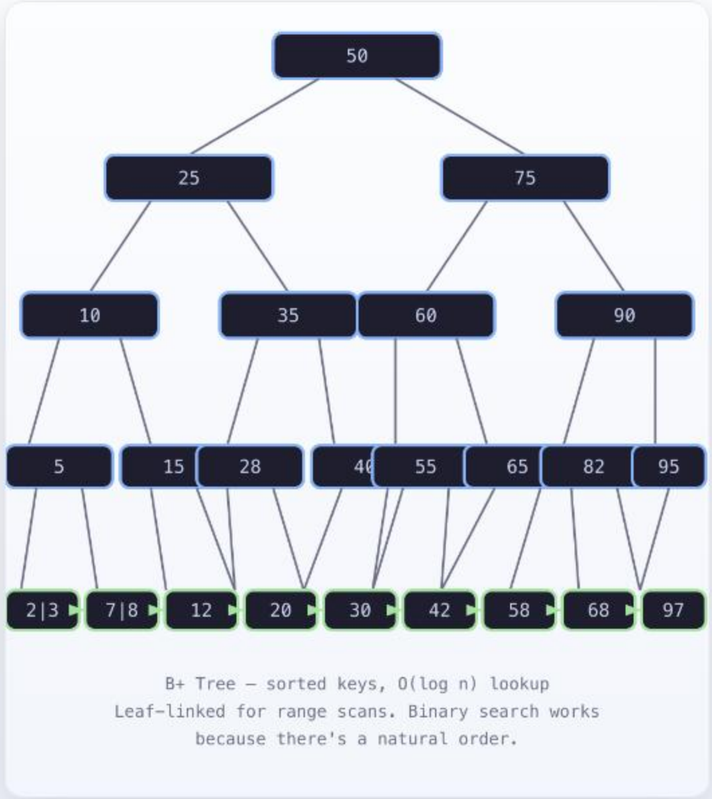
```
Vector A = [0.2, 0.8, 0.1, ... 384 nums]
Vector B = [0.3, 0.7, 0.2, ... 384 nums]

Distance = √((0.3−0.2)² + (0.7−0.8)² + ...)
```

That's Euclidean distance. Works, but...

How do you search when there's no sort order?

Why Vector Indexing Is Different

B-tree: exact ordering, binary search, O(log n)



B+ Tree — sorted keys, O(log n) lookup
Leaf-linked for range scans. Binary search works because there's a natural order.

Vector indexes have no natural sort order.

```
A = [0.21, 0.87, 0.14, ..., 0.53]
B = [0.93, 0.12, 0.38, ..., 0.71]
C = [0.34, 0.76, 0.22, ..., 0.48]
```

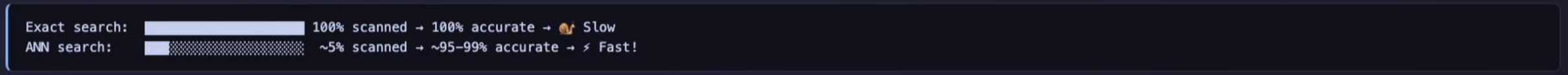
No "less than" for 384 dimensions. **Exact search = check every vector** — too slow at scale.

The Key Insight: Approximate Is Good Enough

What if we don't need the *exact* top 10?

What if finding 9 out of 10 true best matches is acceptable?

This is **Approximate Nearest Neighbor (ANN)** search.



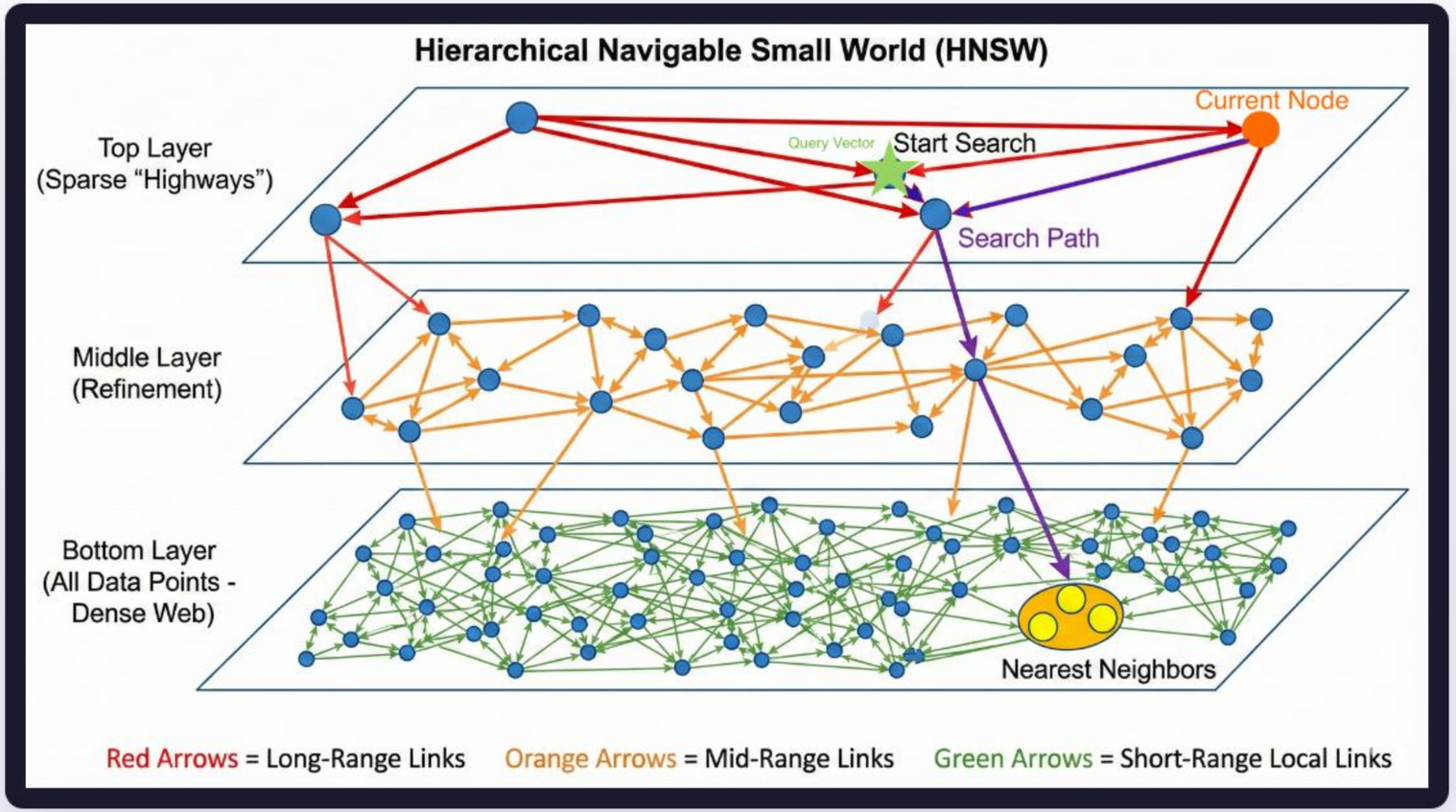
Analogy: 📦 Finding a delivery address in Seoul

- **Exact search:** Check every building in every street → hours 🐌
- **With postal code:** Go to the right 구(gu), check nearby blocks → minutes ⚡
- **The catch:** Might miss a building right on the border

This trade-off — 95% accuracy at 100x speed — is what makes vector search viable.

How ANN Indexes Work

Close enough — without checking every vector.



HNSW — multi-layer graph · ~95-99% recall · 1-5ms · must stay in RAM

Like driving Seoul → Sokcho: expressway → regional road → local street.

What happens when your index outgrows RAM?

The RAM Wall

Everyone wants to build AI on their own data.

Leadership wants to know why the infrastructure bill just tripled.

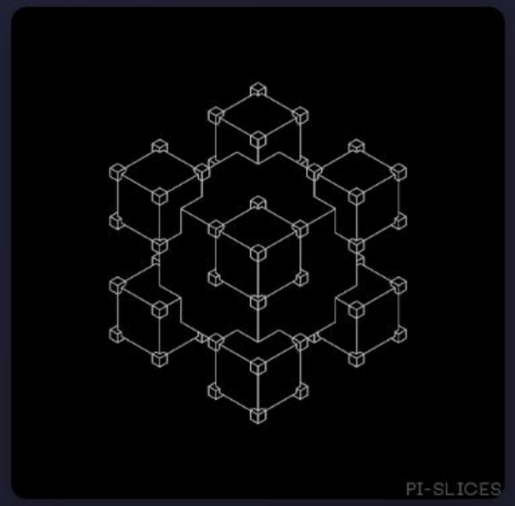
```
Per vector:  1536 dims × 4 bytes = 6,144 bytes ≈ 6 KB
```

Demo used 384d (MiniLM). Production models (OpenAI, Cohere) use 1536d. This is production math.

Scale	Raw Vectors	+ HNSW (50%)	Approx. RAM Cost
1M	6 GB	~9 GB	~$50/mo
10M	61 GB	~92 GB	~$500/mo
100M	614 GB	~920 GB	~$5,000+/mo
1B	6.1 TB	~9.2 TB	💀

64 GB → 920 GB = not 15x cost, it's 30-50x operational complexity.

The Cost of Getting It Wrong



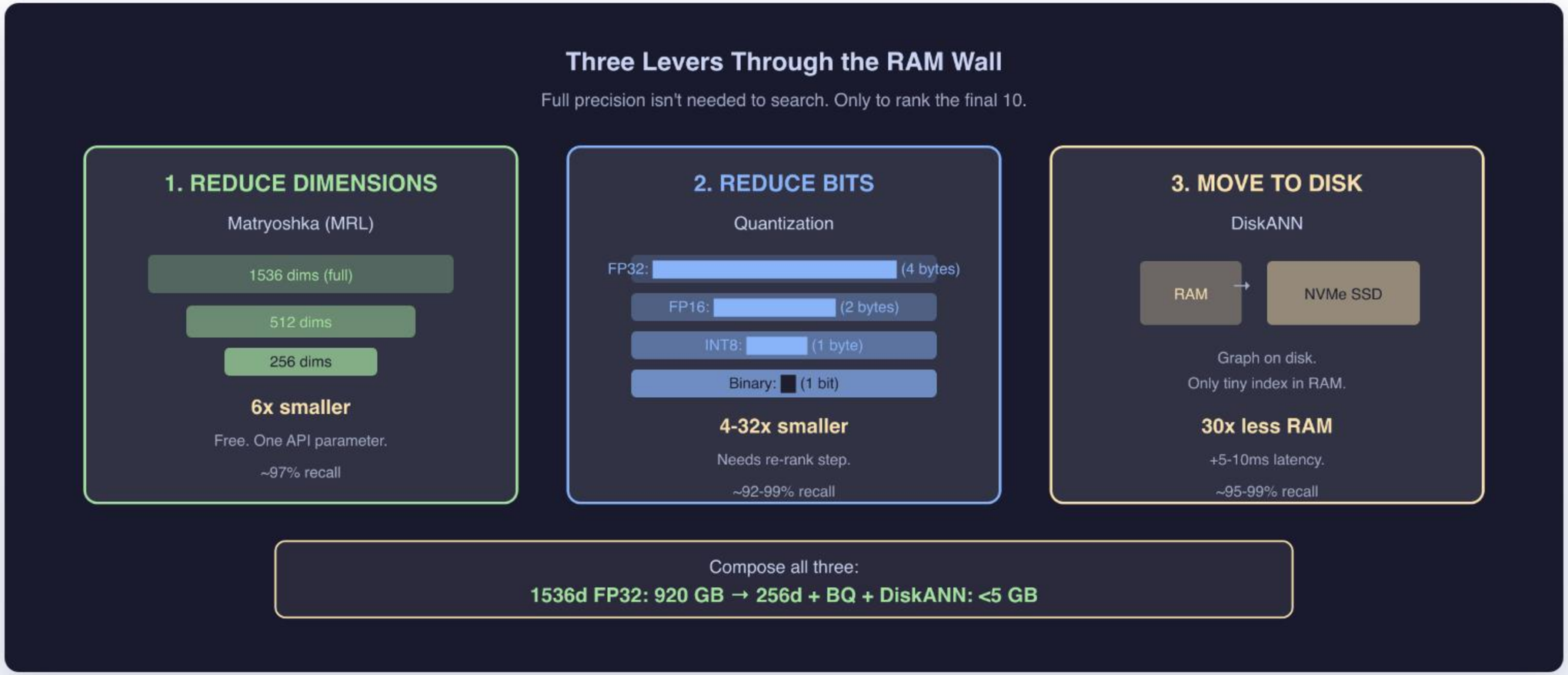
1. "Just use HNSW" → 750 GB RAM → $5K/mo 💸

2. "Add a separate vector DB" → two systems to sync forever 🔄

3. "We'll optimize later" → index rebuild = 3 days, no deploys 🚢

Not bad tools — premature decisions made without doing the math.

Three Ways Through the Wall



920 GB → 5 GB compressed index. Full vectors stay on SSD for re-rank.

(Do them in order: dimensions first, then bits, then disk.)

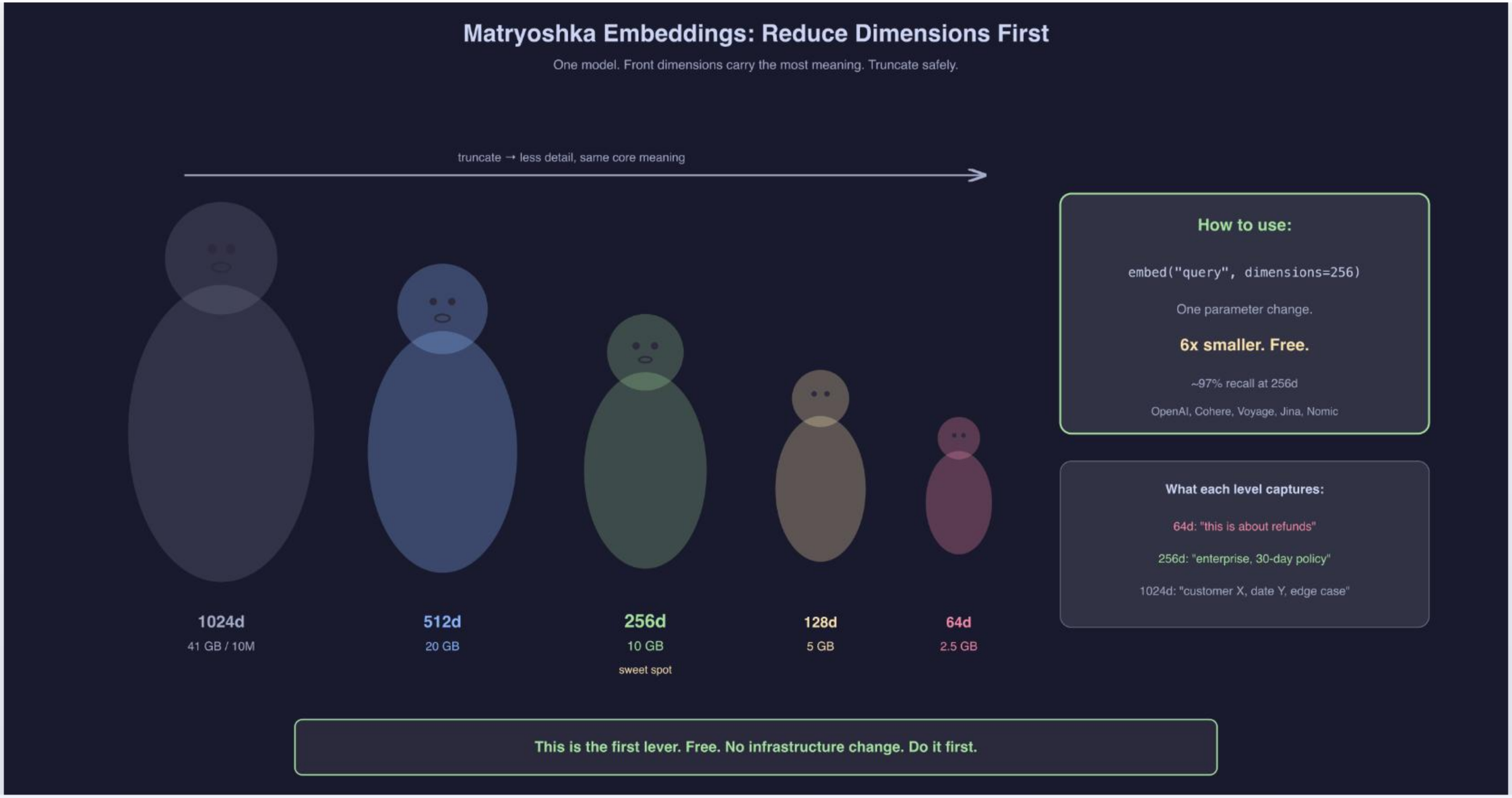
LEVER 1 · REDUCE DIMENSIONS

Matryoshka Embeddings

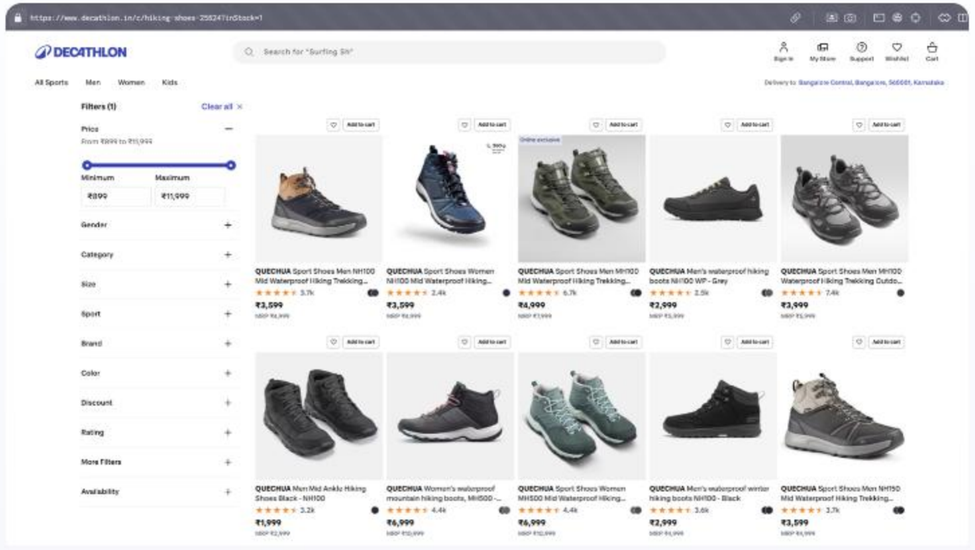
Same doll, fewer layers — still recognizable.

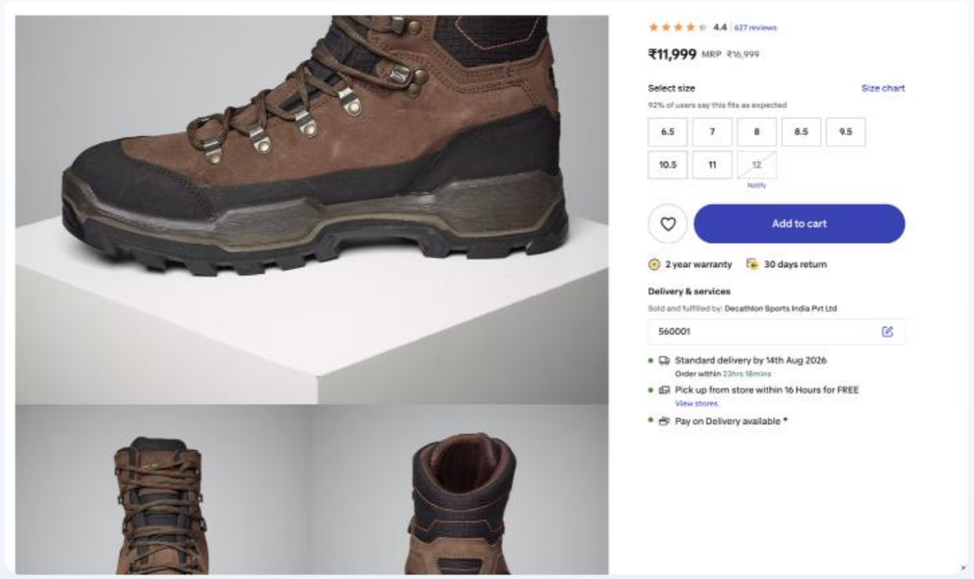
Same embedding, fewer dimensions — still useful.

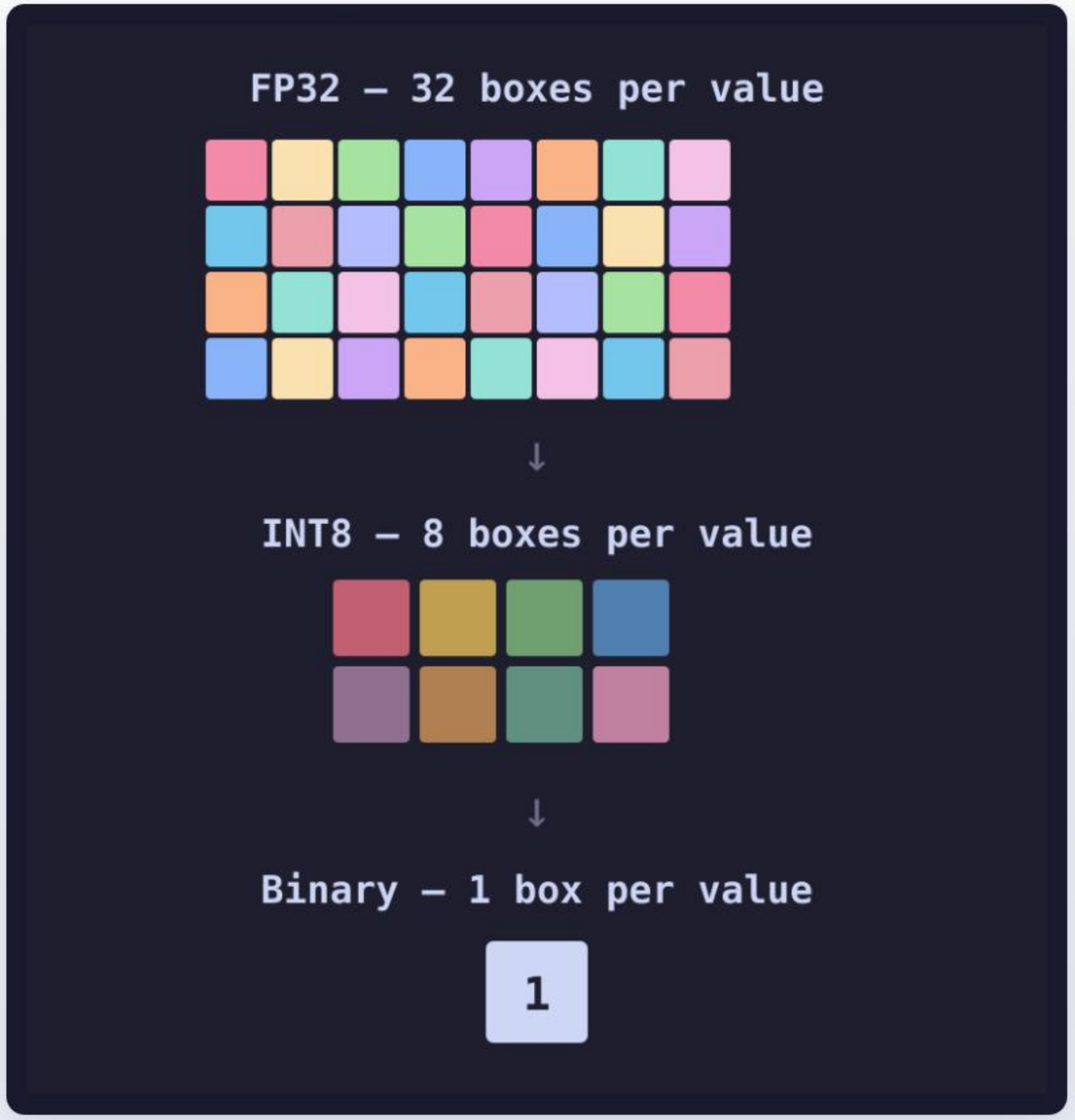
Matryoshka Embeddings (MRL)



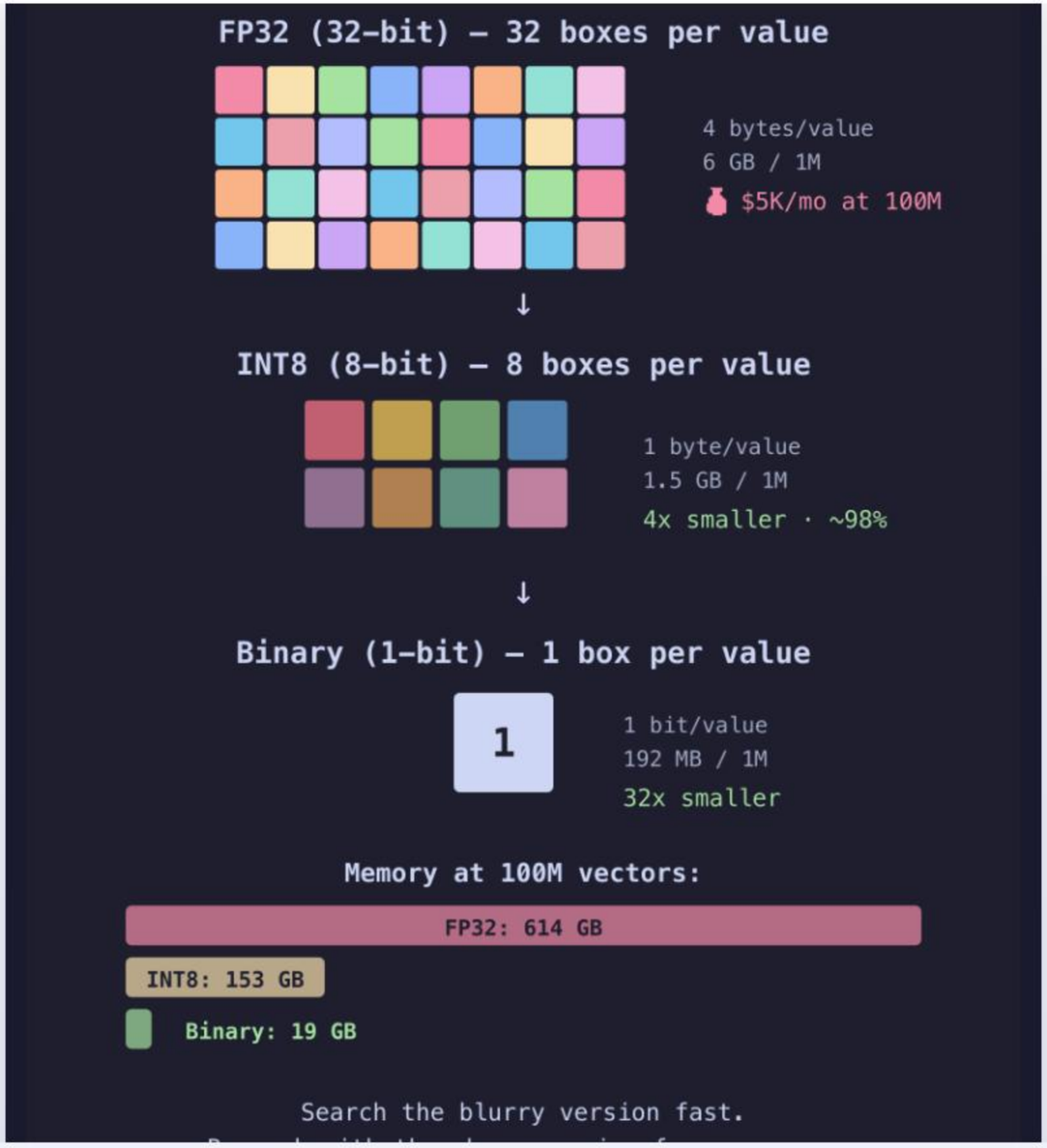
Quantization





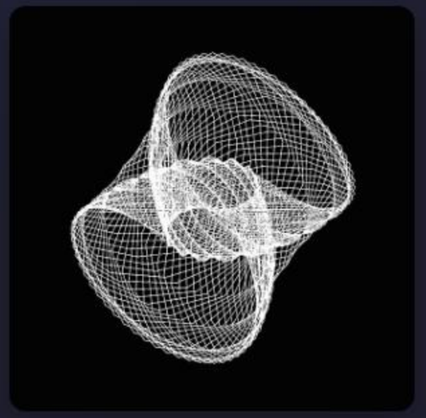


Quantization — The Details



DEMO

Quantization in Action



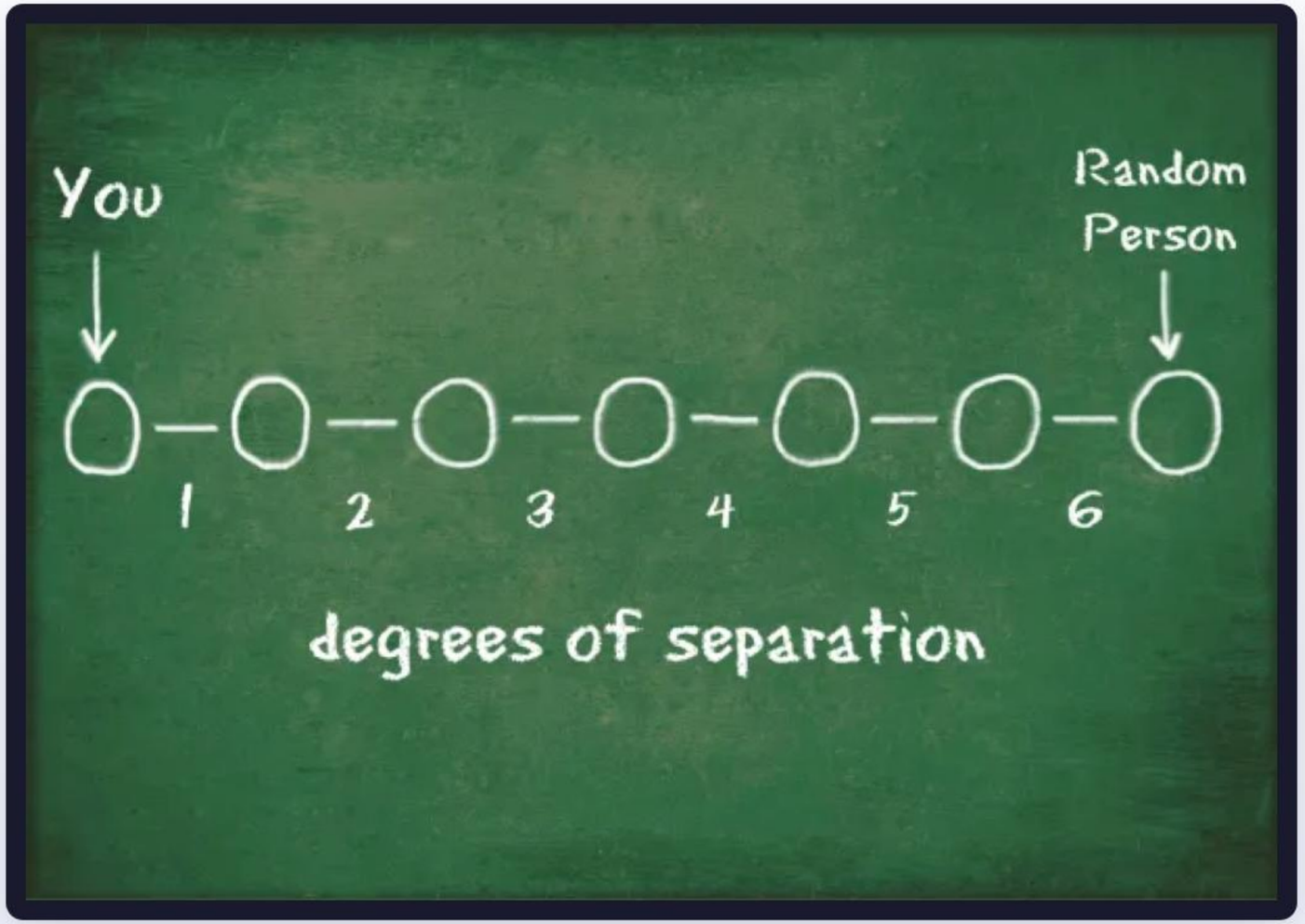
Method	Size (1M)	Recall@10
FP32 (baseline)	6.1 GB	100%
Binary (1-bit)	192 MB	~10%
Binary + rerank	192 MB	~92-96%

32× smaller. Still finds the right answers.

Recall is optimistic at small scale. At 1M+ vectors, expect ~92-96% with rerank.

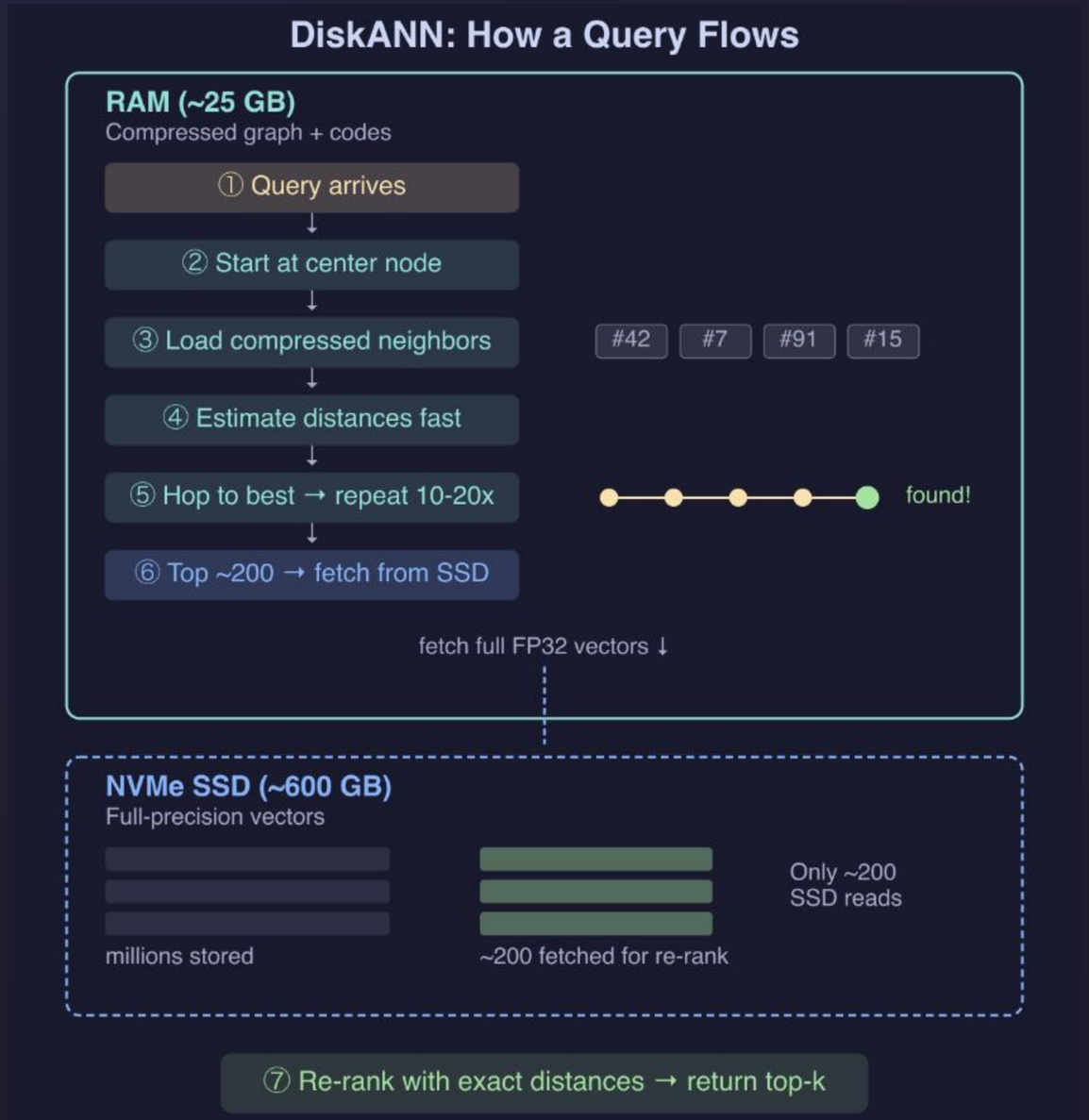
DiskANN — Six Degrees of Separation

8 billion people on Earth.
Any two connected by just 6 hops.



DiskANN uses the same idea: build a graph where any vector is ~6 hops from any other. Keep the graph in RAM, keep the actual vectors on SSD.

DiskANN: The Architecture



What can go wrong after you solve the wall?

SILENT FAILURE #1

Filtered Search

```
SELECT * FROM products
WHERE tenant_id = 42
ORDER BY embedding <=> query LIMIT 10;
```

⚠️ **HNSW returns 10 nearest vectors.**

Filter throws 9 away.

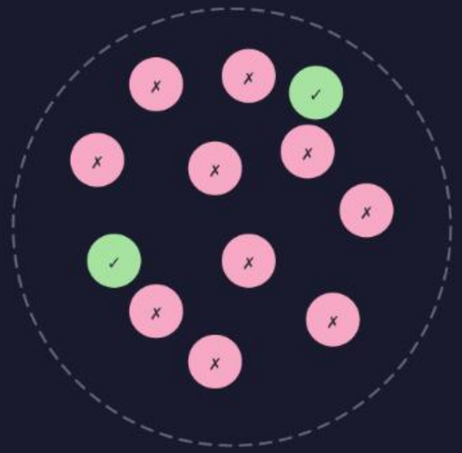
Asked for 10. Got 0.

No error. No warning. No log line.

Pre-Filter vs Post-Filter: Both Fail

Post-Filter (ANN → then filter)

HNSW graph (50K docs)



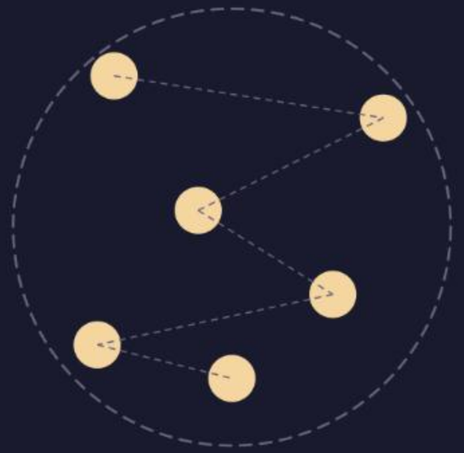
matches filter doesn't match

ANN returns top 10 nearest
→ filter: only 2 match tenant_id=42

✗ Asked for 10, got 2

Pre-Filter (filter → then ANN)

HNSW graph (50K docs)



200 docs match tenant_id=42
→ scattered across graph

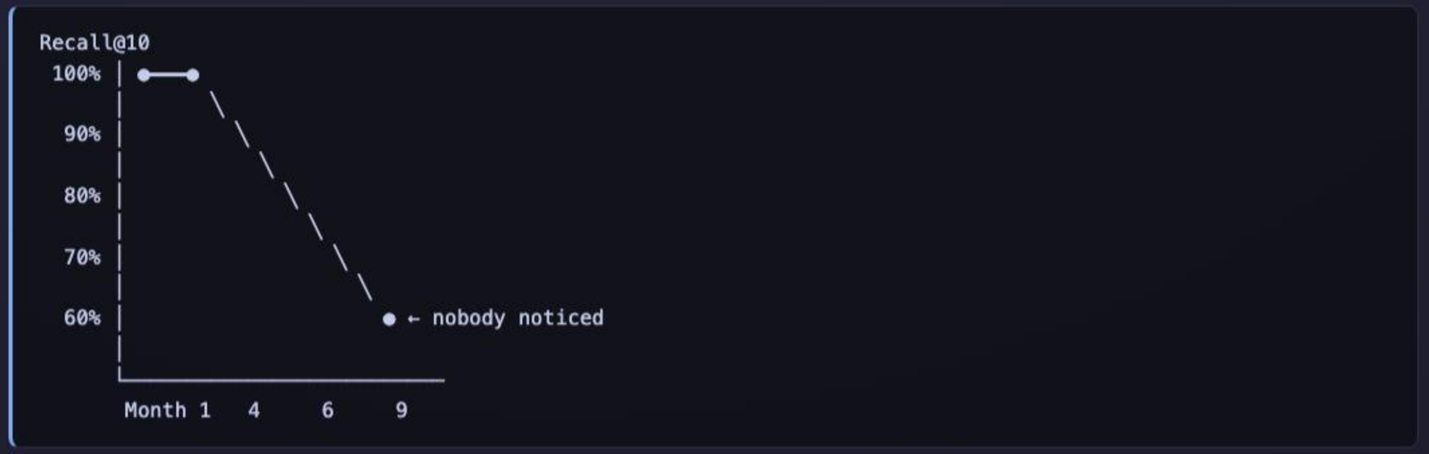
✗ Falls back to sequential scan

Both approaches break. You need adaptive filtered search.

SILENT FAILURE #2

Recall Drift

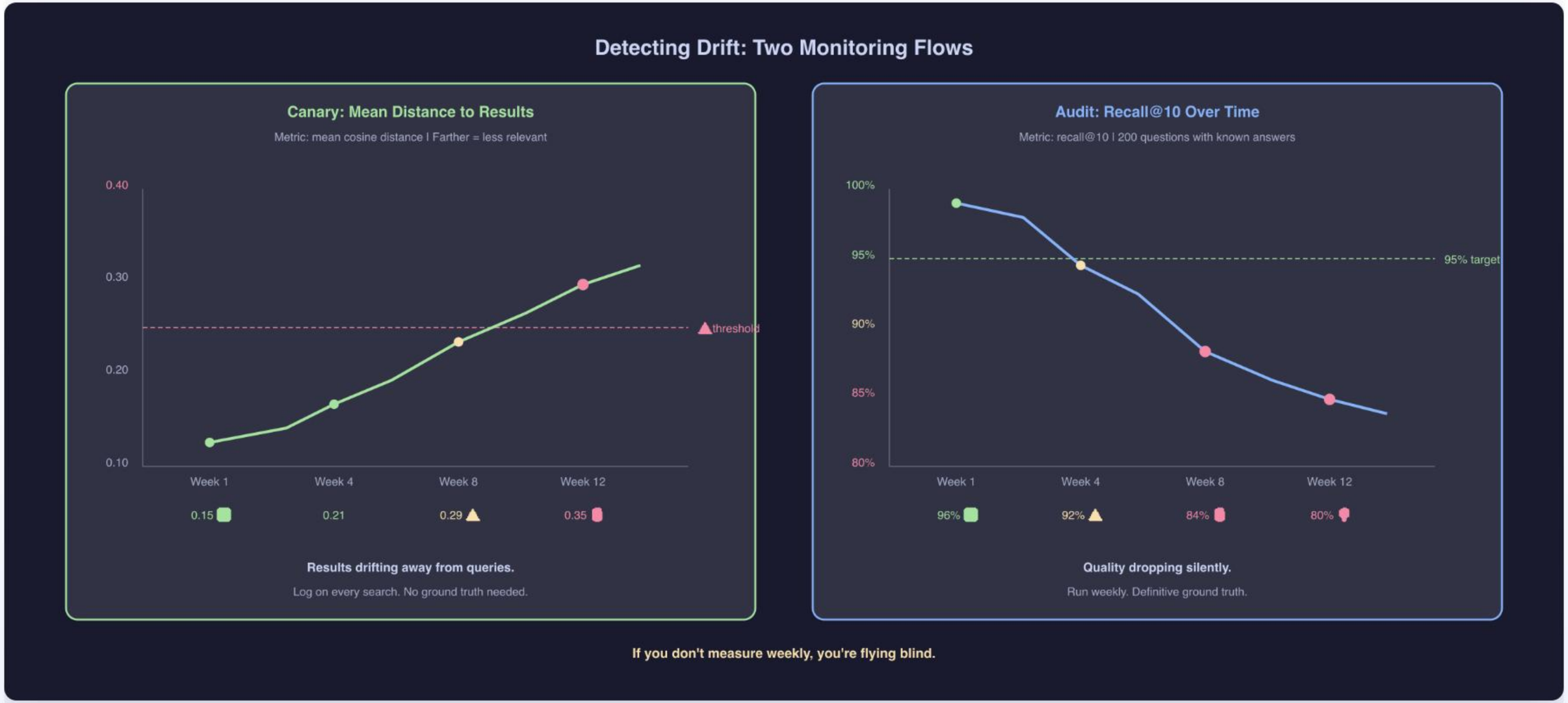
#1 returns zero results. #2 returns the WRONG results.



No error. No alert. No log line.

The system doesn't crash. It just quietly becomes useless.

Detecting Drift: Two Flows



Decision Matrix

SCALE →

< 1M	1M – 10M	10M – 50M	50M+
pgvector + HNSW	+ MRL (256d)	+ Quantization	DiskANN
Just ship.	6× free savings	32× smaller	RAM → SSD

Start with what you have. Stay longer than you think.

Exhaust: dimensions → quantization → DiskANN
before adding infrastructure.

The End

Remember Month 10? You don't have to get there.

Do the math early. Pick compression before it's urgent. Monitor recall always.

Scan for the repo & slides

📫 hello@noobj.me

🌐 noobj.me