Vectors: How to better support a nasty data type

Jonathan Katz

Principal Product Manager – Technical
Amazon RDS

Agenda

Overview: Why do we care about vector search?

Why use PostgreSQL for vector searches?

Year-in-review of pgvector development

Ongoing work and recommendations

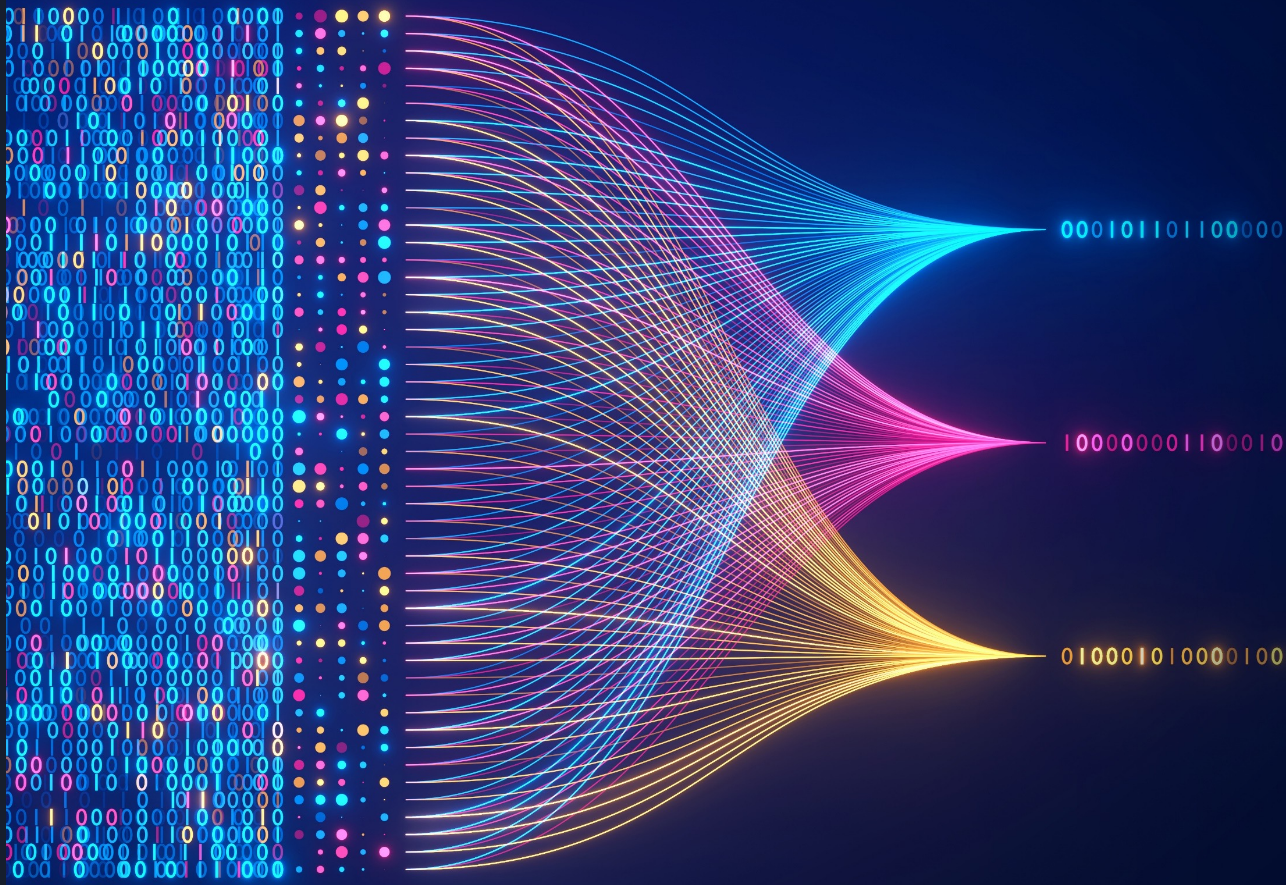
Generative AI is powered by foundation models

Pretrained on vast amounts of unstructured data

Contain a large number of parameters that make them capable of learning complex concepts

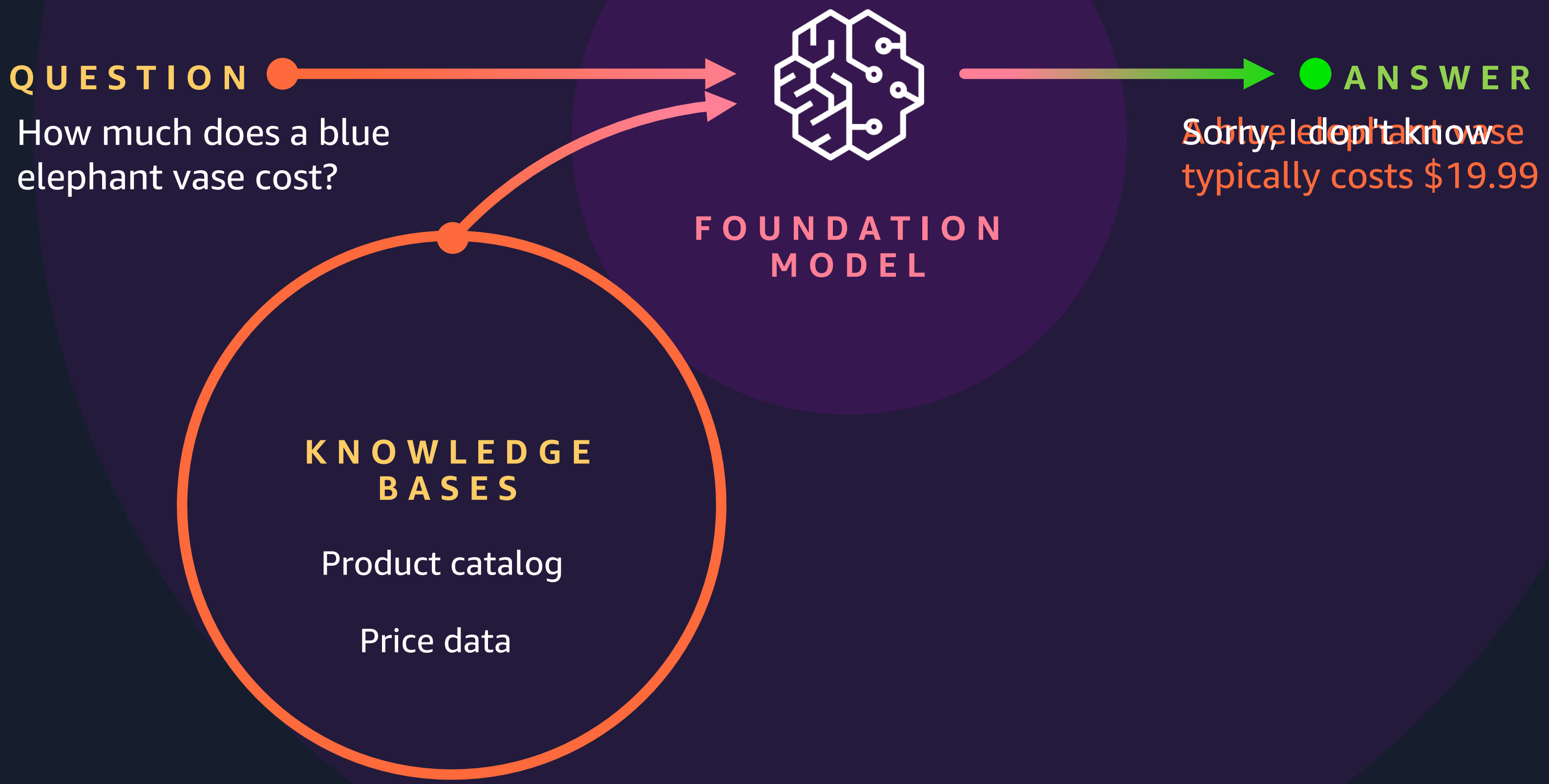
Can be applied in a wide range of contexts

Customize FMs using your data for domain-specific tasks



Retrieval Augmented Generation (RAG)

Configure FM to interact with your data



QUESTION

How much does a blue elephant vase cost?

FOUNDATION MODEL

ANSWER

Sorry, I don't know

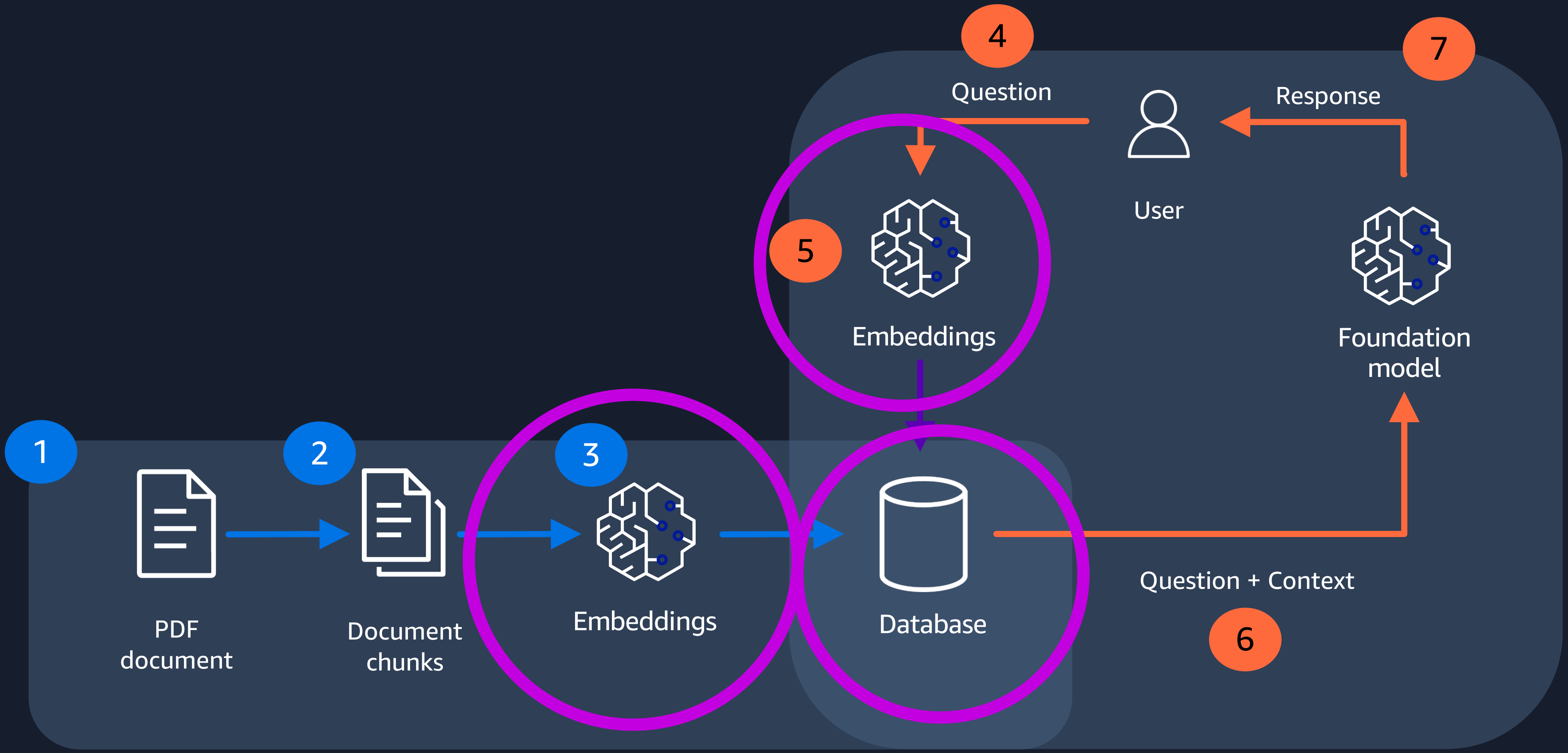
A blue elephant vase typically costs $19.99

KNOWLEDGE BASES

Product catalog

Price data

The role of vectors in RAG



Challenges with vectors

- Time to generate embeddings
- Embedding size
- ~~Compression~~
- Query time

1536 dimensions

Blue elephant vase that can hold up to three plants in its hand painted…

0.12310
0.24234
0.59405
0.23430
0.23432
0.20551
0.70543
0.20559

0.20559
0.70543
0.23432
0.24234
0.23430
0.12310
0.20551
0.59405

0.1234
0.1231
0.1232
0.9005
0.2489

4-byte floats

6152B => 6KiB

1,000,000 => 5.7GB

Approximate nearest neighbor (ANN)

- Find similar vectors without searching all of them

- Faster than exact nearest neighbor

- “Recall” – % of expected results

Recall: 80%

Key metrics to consider

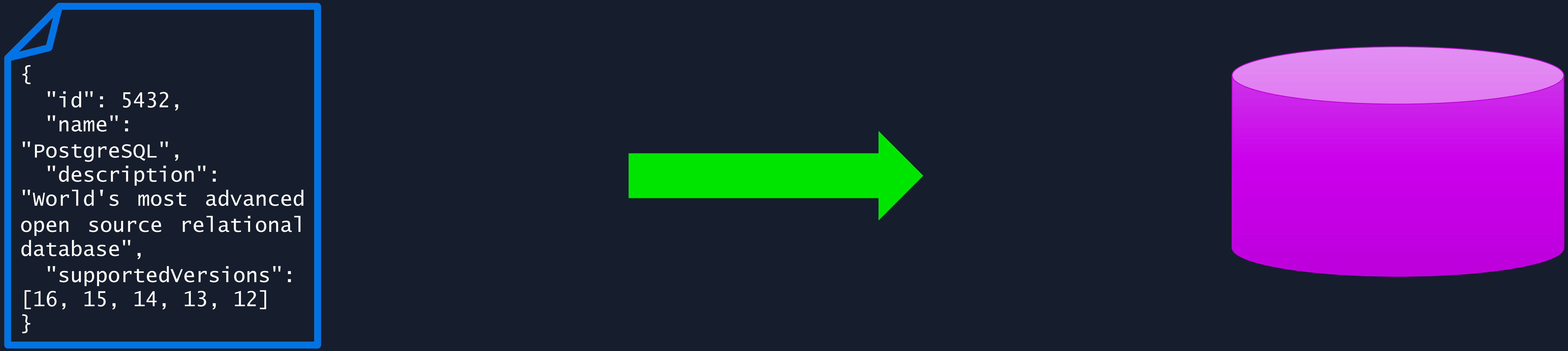
- Index build time
- Index size
- Recall
- Query throughput (queries per second)
- p99 query latency

PostgreSQL as a "vector database"

```json
{
  "id": 5432,
  "name": "PostgreSQL",
  "description": "World's most advanced open source relational database",
  "supportedVersions": [16, 15, 14, 13, 12]
}
```

```
{
  "id": 5432,
  "name":
"PostgreSQL",
  "description":
"World's most advanced
open source relational
database",
  "supportedVersions":
[16, 15, 14, 13, 12]
}
```

id	**5432**
name	PostgreSQL
description	world's most...
supportedVersions	[16,15,14,13,12]



```
{
  "id": 5432,
  "name":
"PostgreSQL",
  "description":
"World's most advanced
open source relational
database",
  "supportedVersions":
[16, 15, 14, 13, 12]
}
```

Timeline of JSON storage

- 2000-2001: JSON invented
- 2004: AJAX model emerges in wider deployments
- 2006: RFC 4627 publishes JSON format
- 2006-2009: JSON-specific data stores emerge
- 2012: PostgreSQL adds support for JSON (text)
- 2013: ECMA-404 standardizes JSON
- 2014: PostgreSQL adds support for JSONB (binary)
- 2017: SQL/JSON standard published
- 2019: PostgreSQL adds SQL/JSON path language
- 2023: PostgreSQL adds SQL/JSON constructors and predicates
- 2024: PostgreSQL adds SQL/JSON query functions and JSON_TABLE

Why use PostgreSQL for vector searches?

- Existing client libraries work without modification
  - May require an upgrade

- Convenient to co-locate app + AI/ML data in same database

- Interfacing with PostgreSQL storage gives ACID transactional storage

Why care about ACID for vectors?

- Atomicity: "All or nothing" stored in transaction (bulk loads)
- Consistency: Follows rules for other data stored in database
- Isolation: Correctness in returned results; committed transactions "immediately available"
- Durability: One committed, vectors are safely stored.

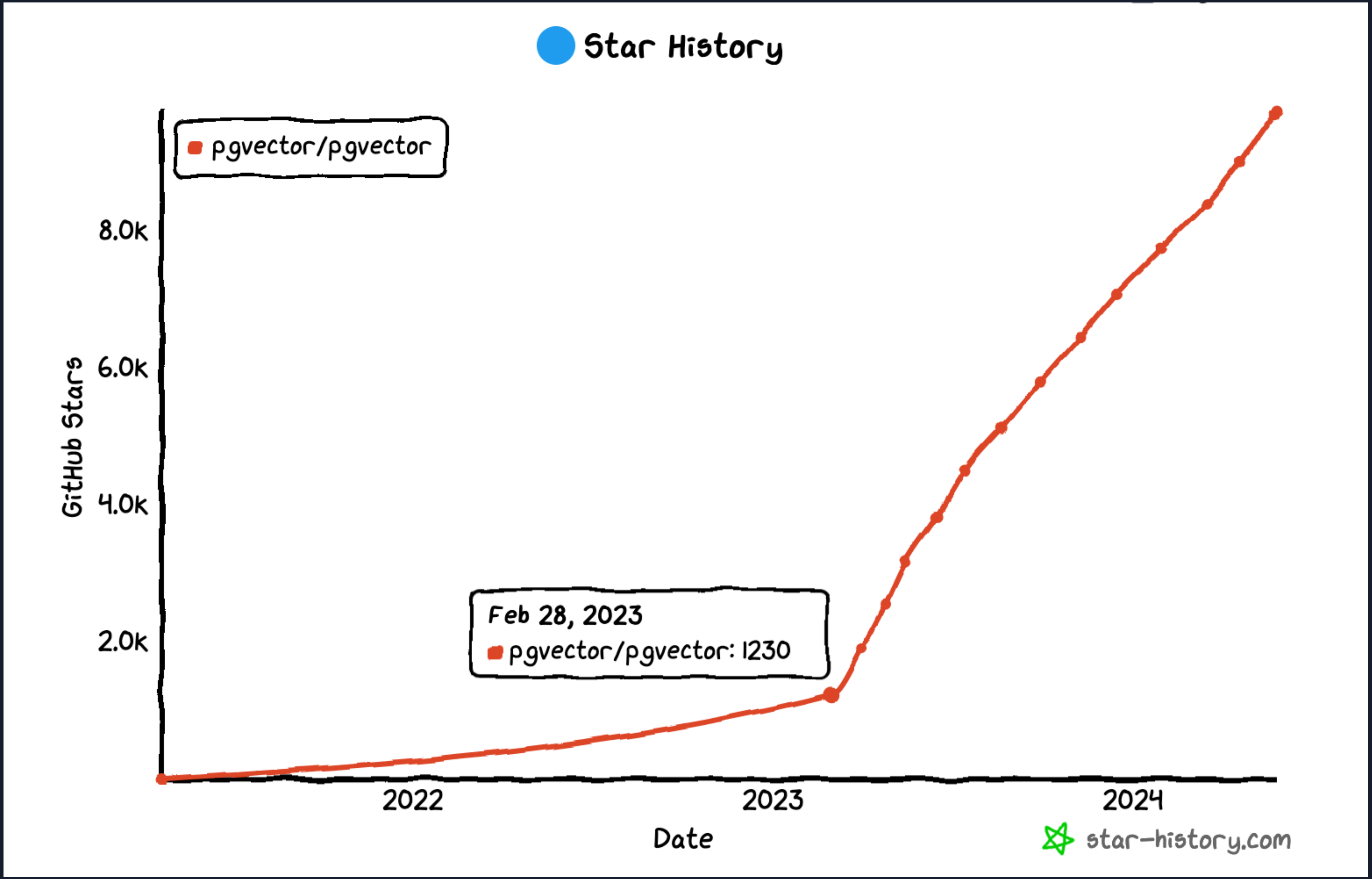
PostgreSQL support for vectors

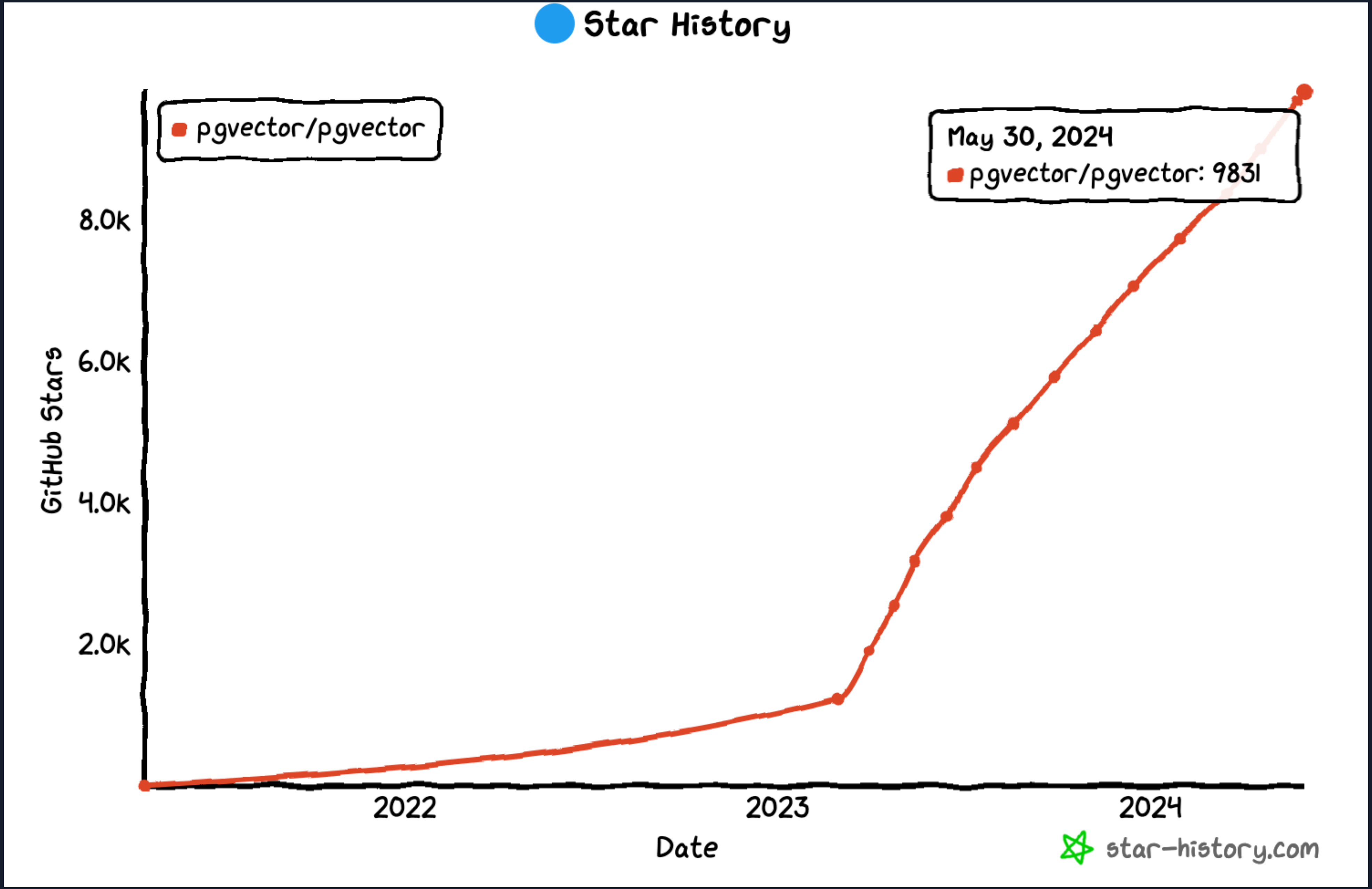
Native

- ARRAY
- cube

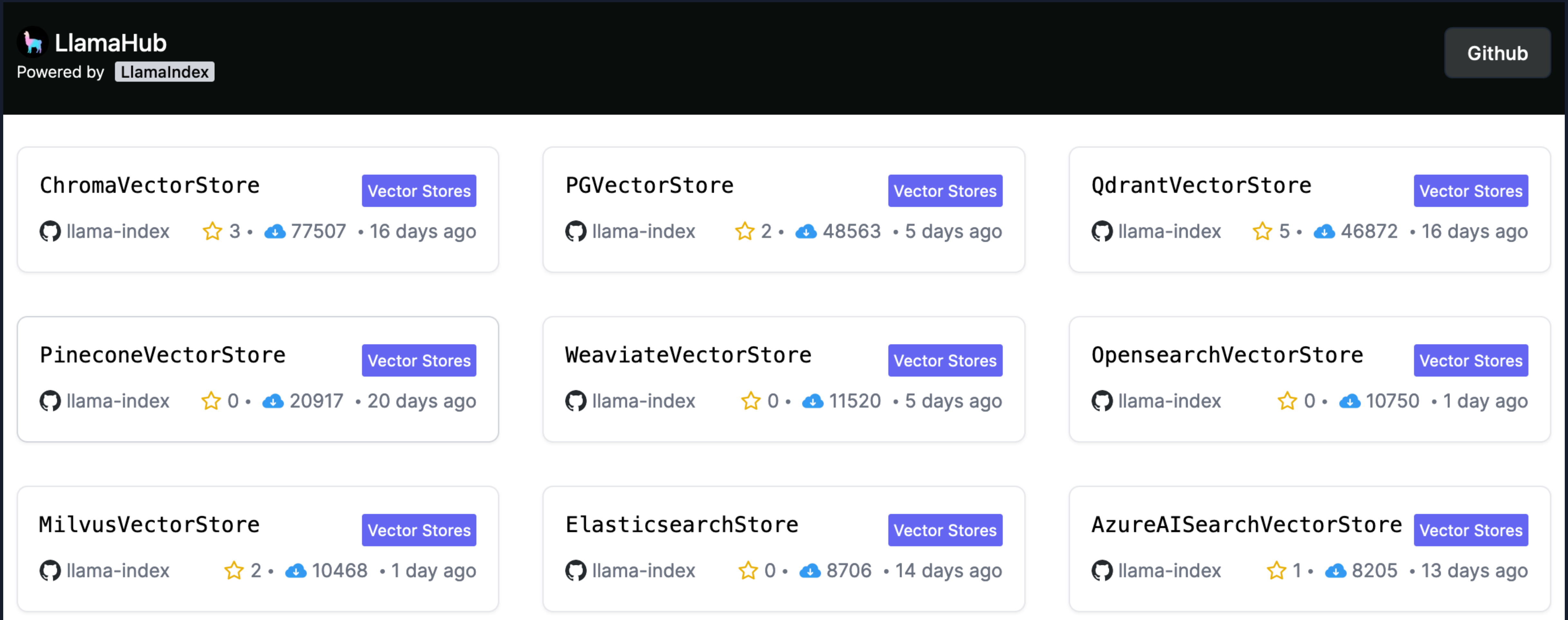
Extensions

- pgvector
- ~~pg_embedding~~
- pgvecto.rs
- Lantern
- Timescale Vector
- pgvector-remote


Star History
pgvector/pgvector
8.0k
6.0k
4.0k
2.0k
GitHub Stars
Feb 28, 2023
pgvector/pgvector: 1230
2022
2023
2024
Date
star-history.com


Star History
pgvector/pgvector
May 30, 2024
pgvector/pgvector: 9831
GitHub Stars
8.0k
6.0k
4.0k
2.0k
2022
2023
2024
Date
star-history.com

pgvector popularity



`ChromaVectorStore` Vector Stores
llama-index ☆ 3 • ☁ 77507 • 16 days ago

`PGVectorStore` Vector Stores
llama-index ☆ 2 • ☁ 48563 • 5 days ago

`QdrantVectorStore` Vector Stores
llama-index ☆ 5 • ☁ 46872 • 16 days ago

`PineconeVectorStore` Vector Stores
llama-index ☆ 0 • ☁ 20917 • 20 days ago

`WeaviateVectorStore` Vector Stores
llama-index ☆ 0 • ☁ 11520 • 5 days ago

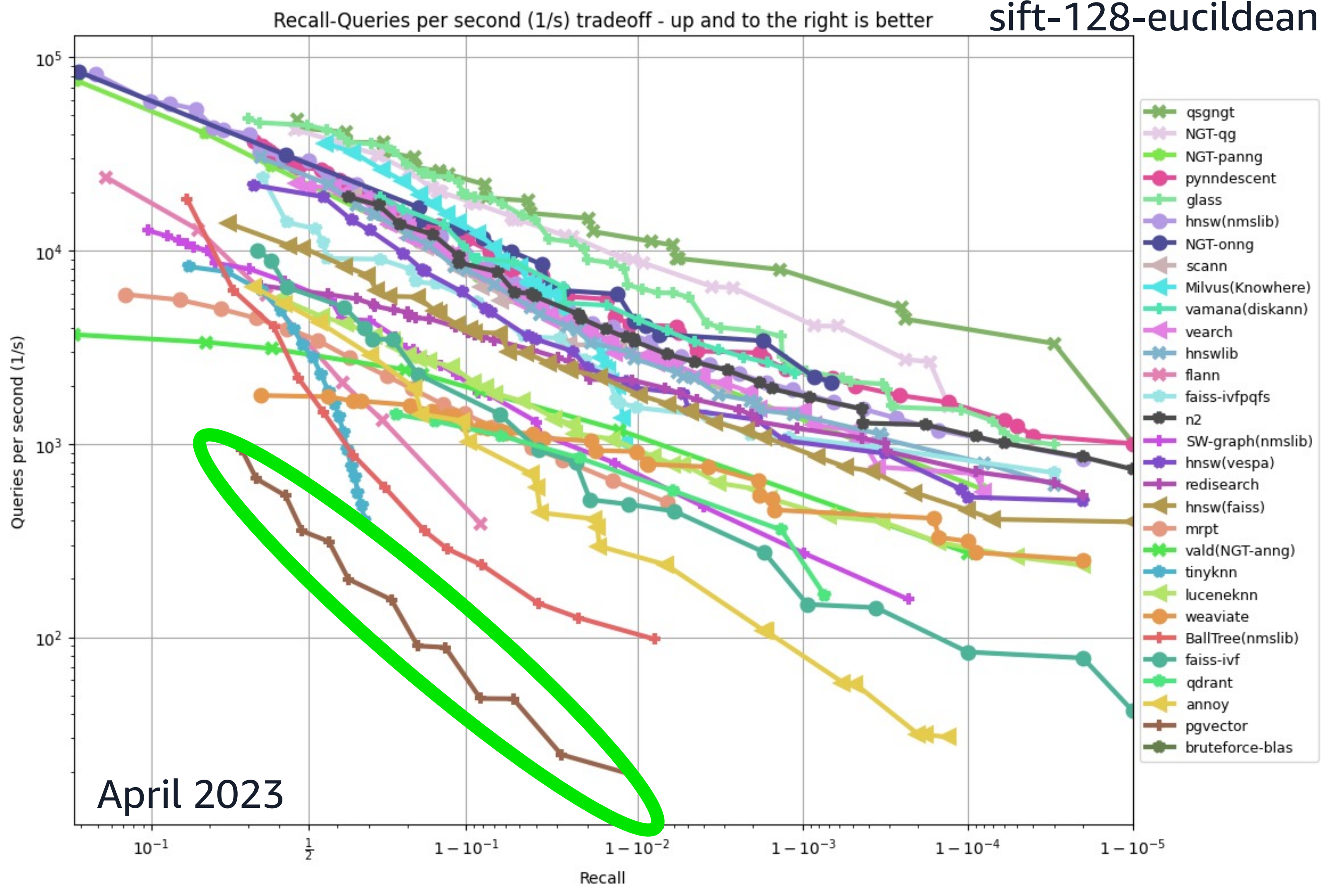
`OpensearchVectorStore` Vector Stores
llama-index ☆ 0 • ☁ 10750 • 1 day ago

`MilvusVectorStore` Vector Stores
llama-index ☆ 2 • ☁ 10468 • 1 day ago

`ElasticsearchStore` Vector Stores
llama-index ☆ 0 • ☁ 8706 • 14 days ago

`AzureAISearchVectorStore` Vector Stores
llama-index ☆ 1 • ☁ 8205 • 13 days ago

Why pgvector?



sift-128-eucildean

April 2023

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

Why pgvector?

2023

- Vector searches in PostgreSQL
  - "It was there"
- Can use existing PostgreSQL drivers
- Open source
- C-based

2024

- High performance vector searches
- Support for larger vectors
- Sustained, rapid improvements
- Better support in developer tools

pgvector: Year-in-review timeline

- **v0.4.x** (1/2023 – 6/2023)
  - Improved IVFFlat plan costs
  - Increasing dimension of vectors stored in table + index
- **v0.5.x** (8/2023 – 10/2023)
  - Add HNSW index + distance function performance improvements
  - Parallel IVFFlat builds
- **v0.6.x** (1/2024 – 3/2024)
  - Parallel HNSW index builds + in-memory build optimizations
- **v0.7.x** (4/2024)
  - halfvec (2-byte float), bit(n) index support, sparsevec (up to 1B dim)
  - Quantization (scalar/binary), Jaccard/hamming distance, explicit SIMD

Indexing in pgvector

How does pgvector index a vector?

0.0234
0.093
-0.9123
0.1055

Valid?

✅ Same dimensions?
✅ Magnitude > 0?

Normalized?

🛠️ If not, normalize

0.0253
0.1007
-0.9880
0.1142

Indexing methods: IVFFlat and HNSW

- IVFFlat
  - K-means based
  - Organize vectors into lists
  - Requires prepopulated data
  - Insert time bounded by # lists

- HNSW
  - Graph based
  - Organize vectors into “neighborhoods”
  - Iterative insertions
  - Insertion time increases as data in graph increases

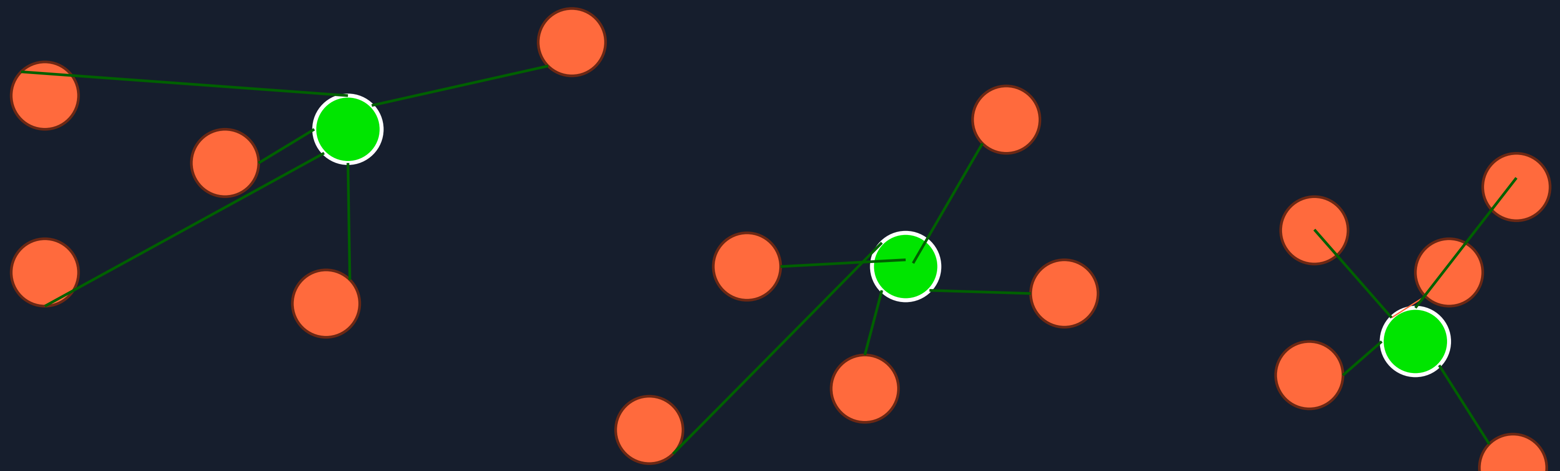
IVFFlat index building parameters

- `lists`
    - Number of “buckets” for organizing vectors
    - Tradeoff between number of vectors in bucket and relevancy

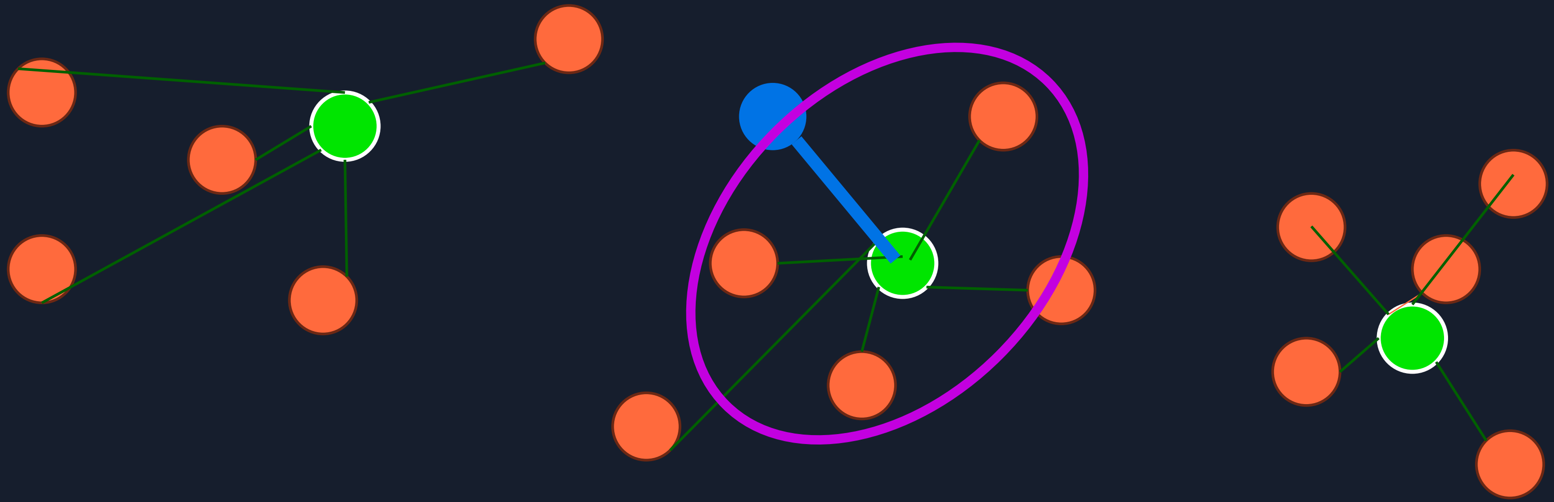
```
        CREATE INDEX ON products
USING ivfflat(embedding) WITH (lists=3);
```

Building an IVFFlat index

Building an IVFFlat index: Assign lists

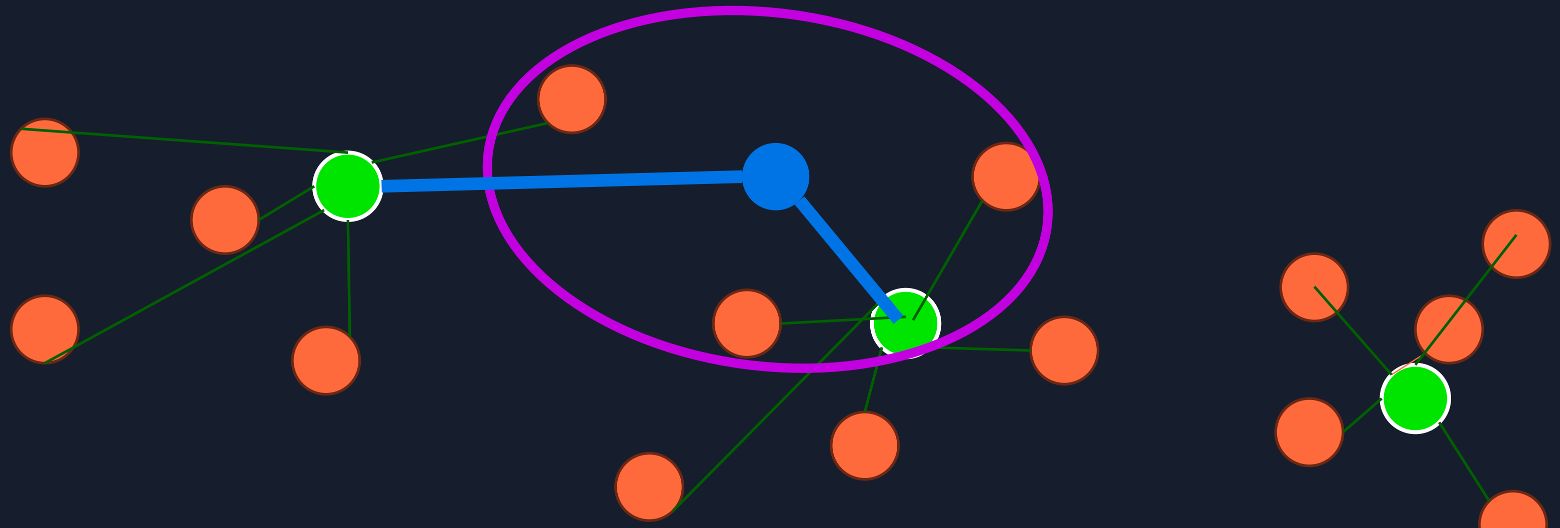


Querying an IVFFlat index



```
SET ivfflat.probes TO 1

SELECT id FROM products ORDER BY $1 <-> embedding LIMIT 3
```

Querying an IVFFlat index



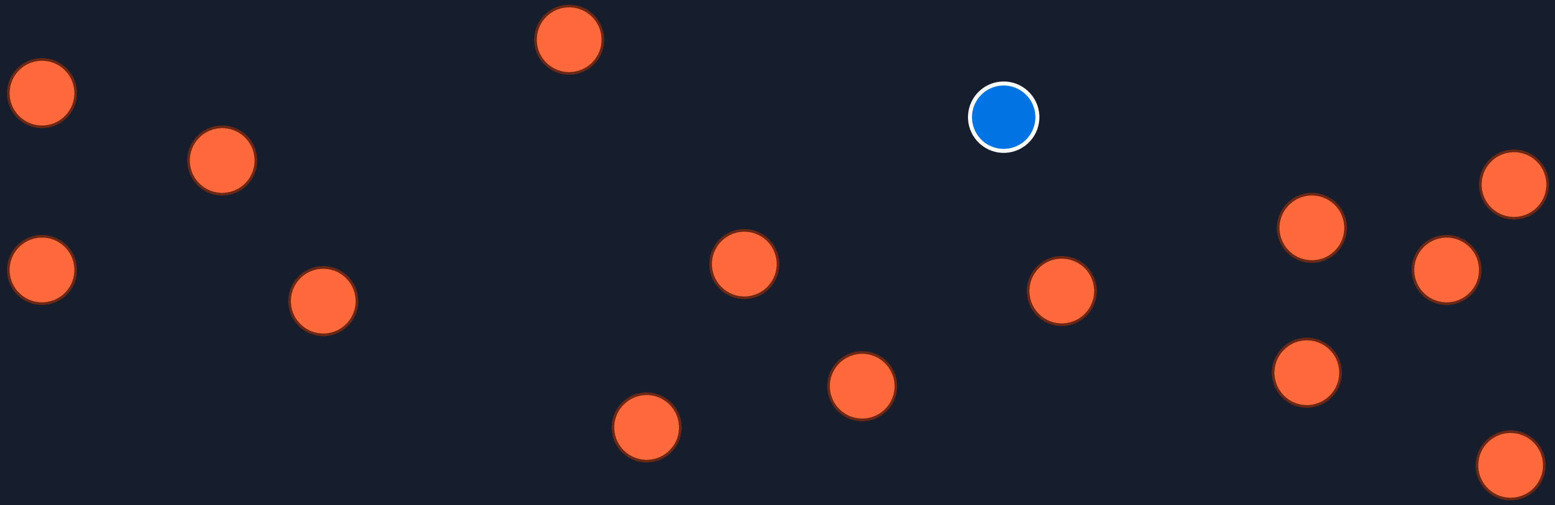
```
SET ivfflat.probes TO 2

SELECT id FROM products ORDER BY $1 <-> embedding LIMIT 3
```

HNSW index building parameters

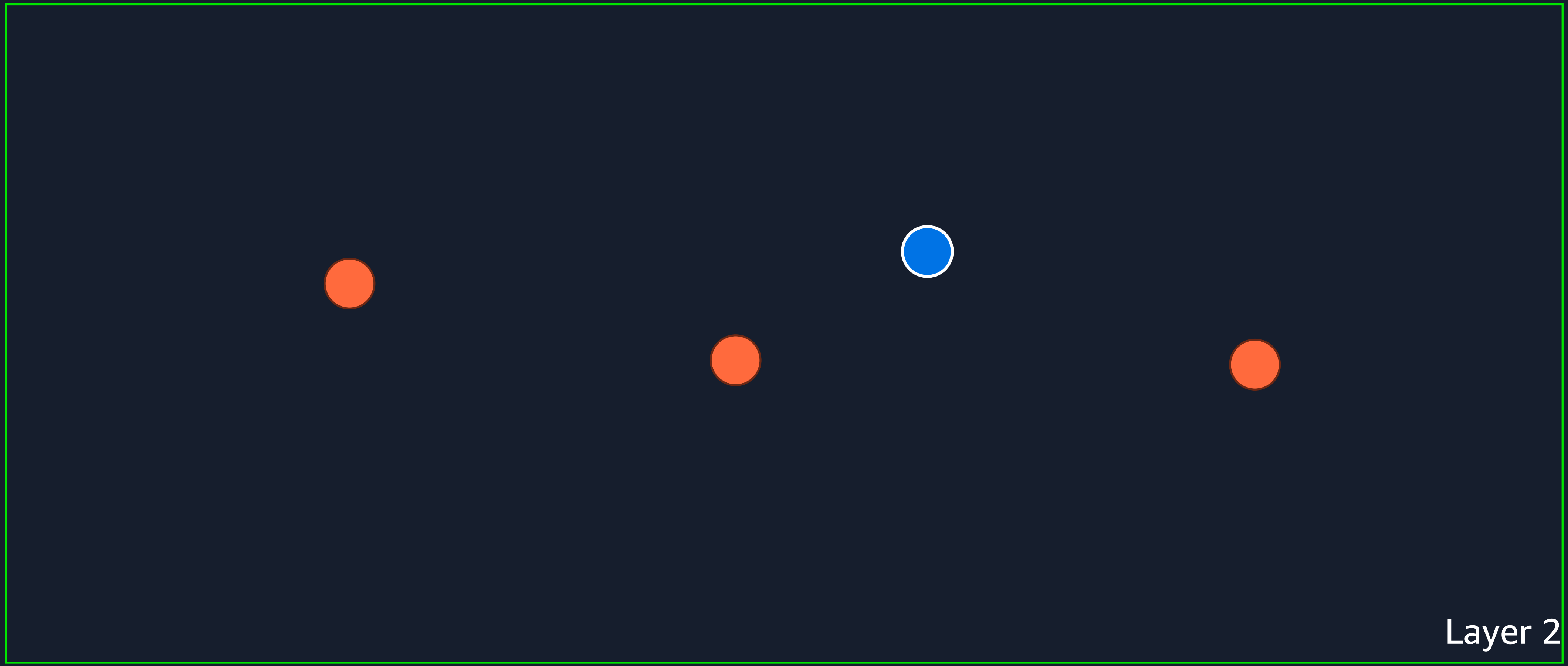
- `m`
  - Maximum number of bidirectional links between indexed vectors
  - Default: 16

- `ef_construction`
  - Number of vectors to maintain in “nearest neighbor” list
  - Default: 64

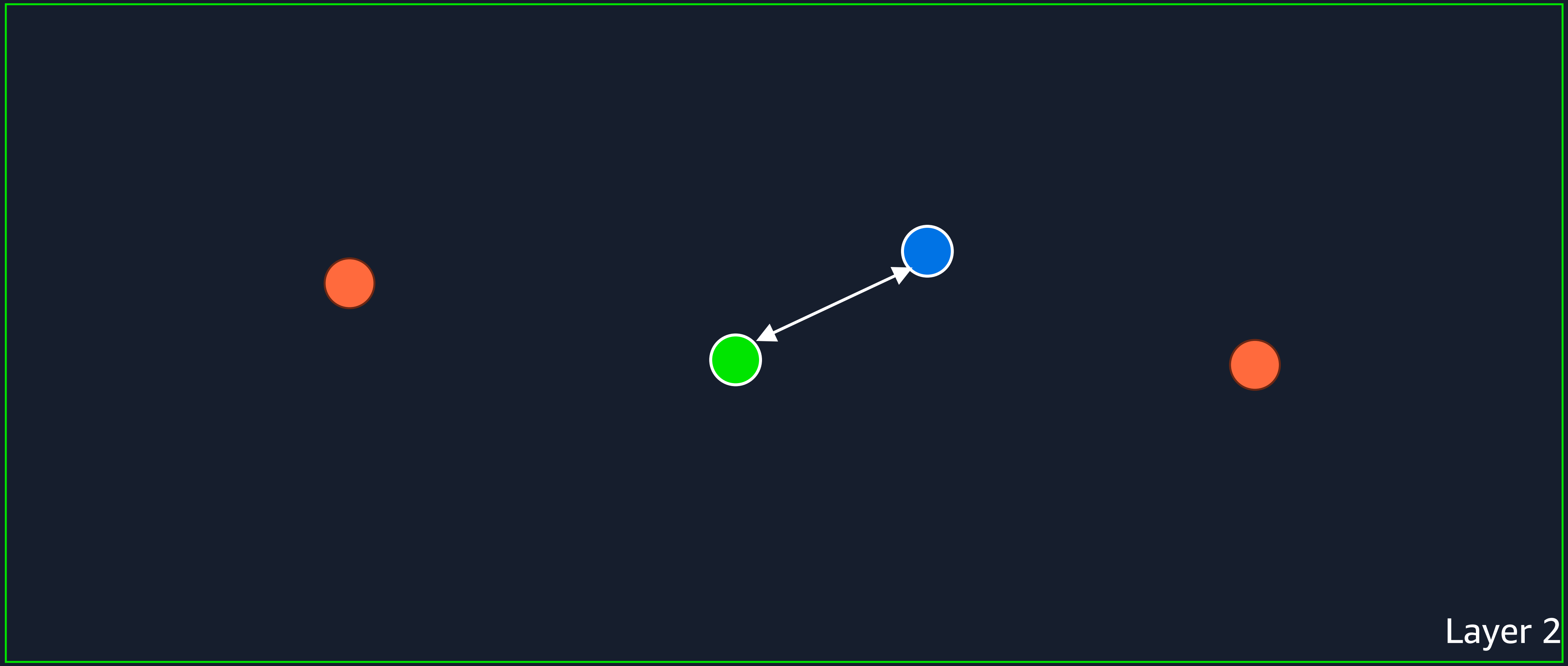
Building an HNSW index



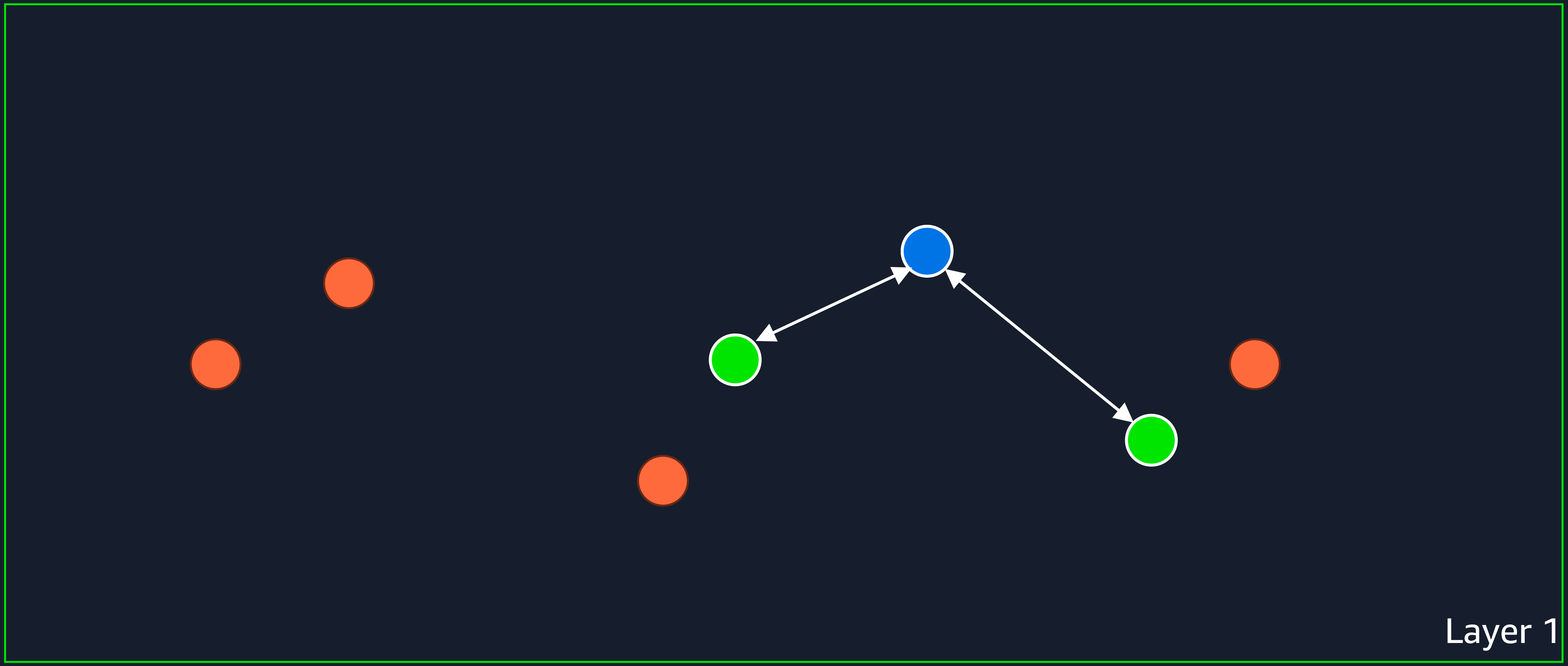
Building an HNSW index



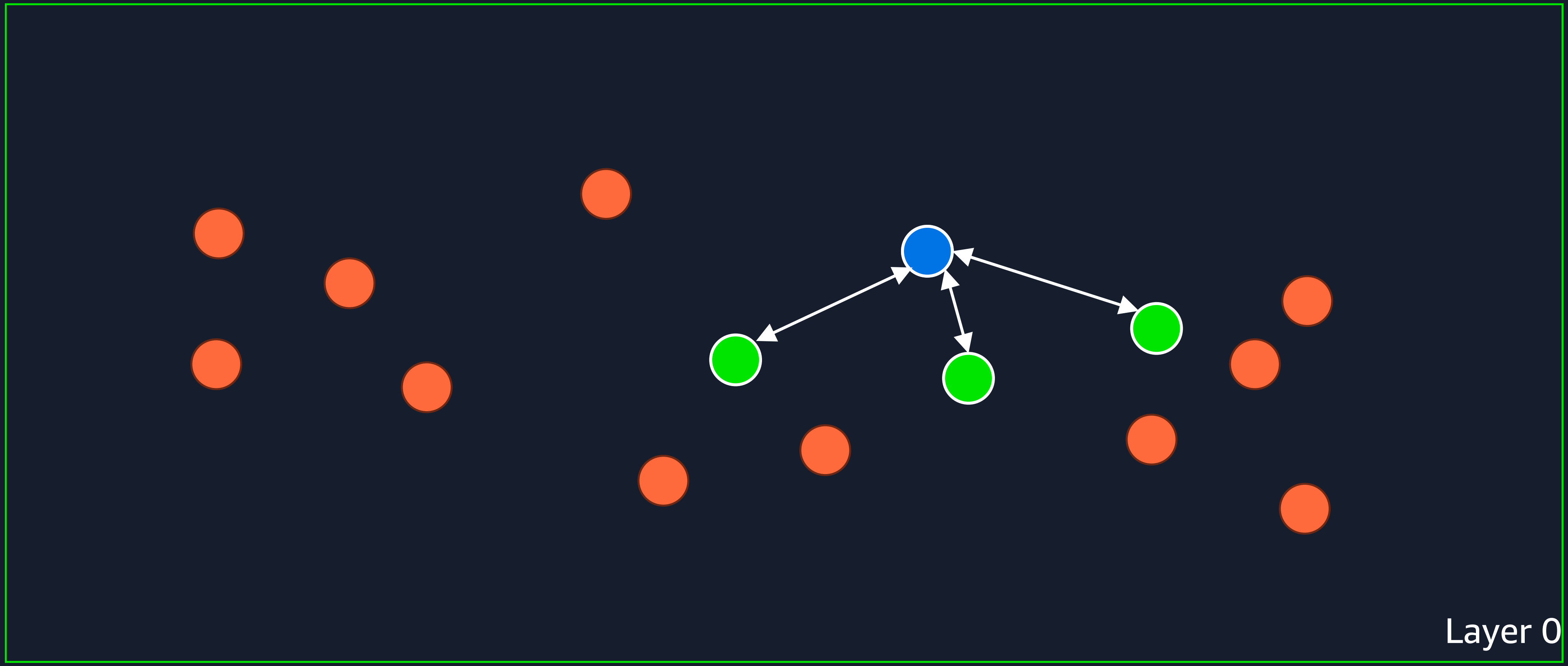
Building an HNSW index



Building an HNSW index



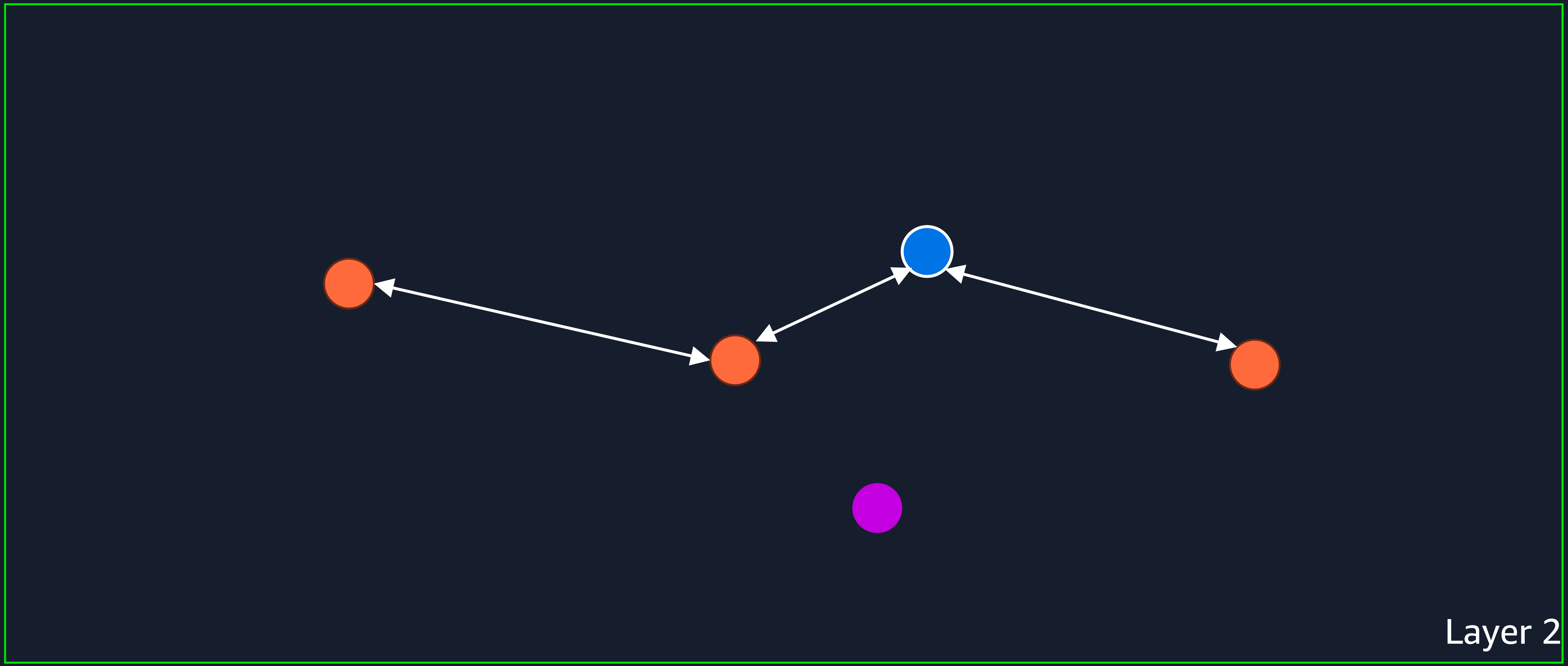
Building an HNSW index



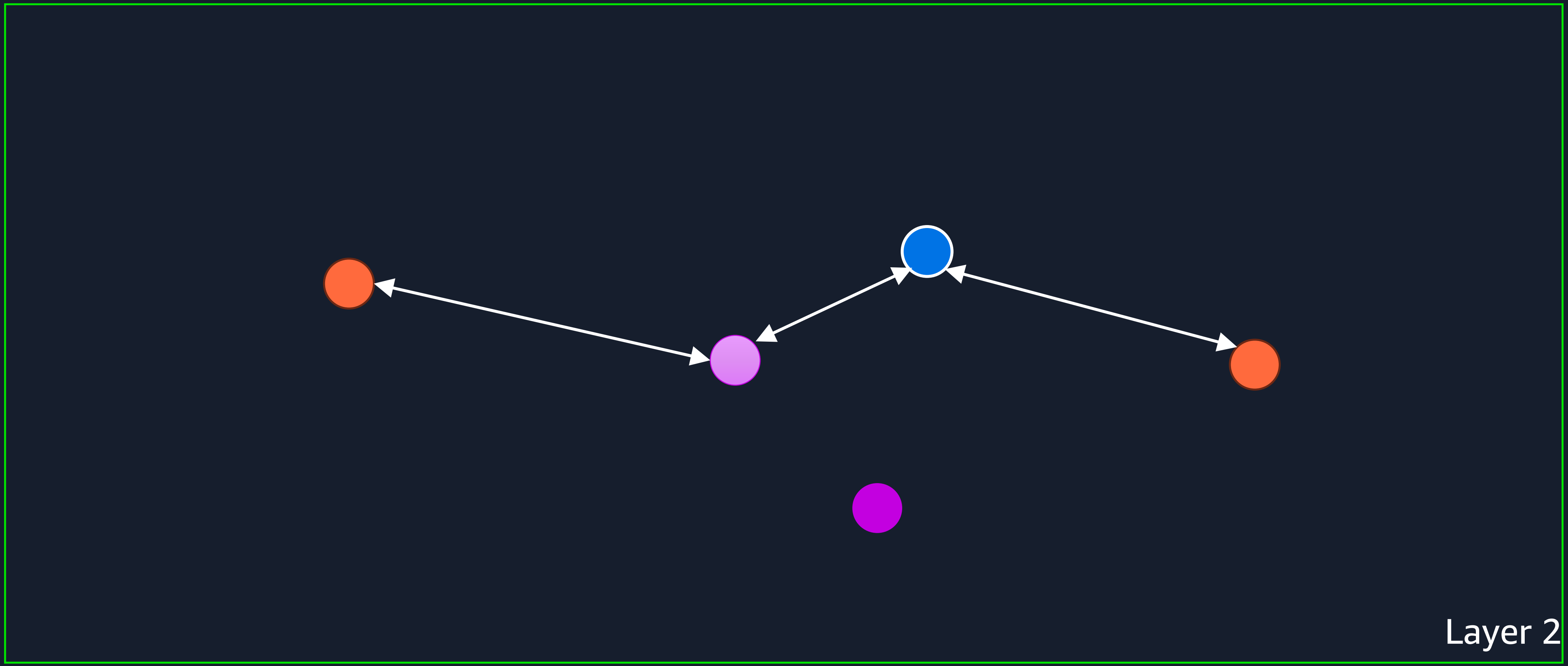
HNSW query parameters

- `hnsw.ef_search`
  - Number of vectors to maintain in “nearest neighbor” list
  - Must be greater than or equal to `LIMIT`

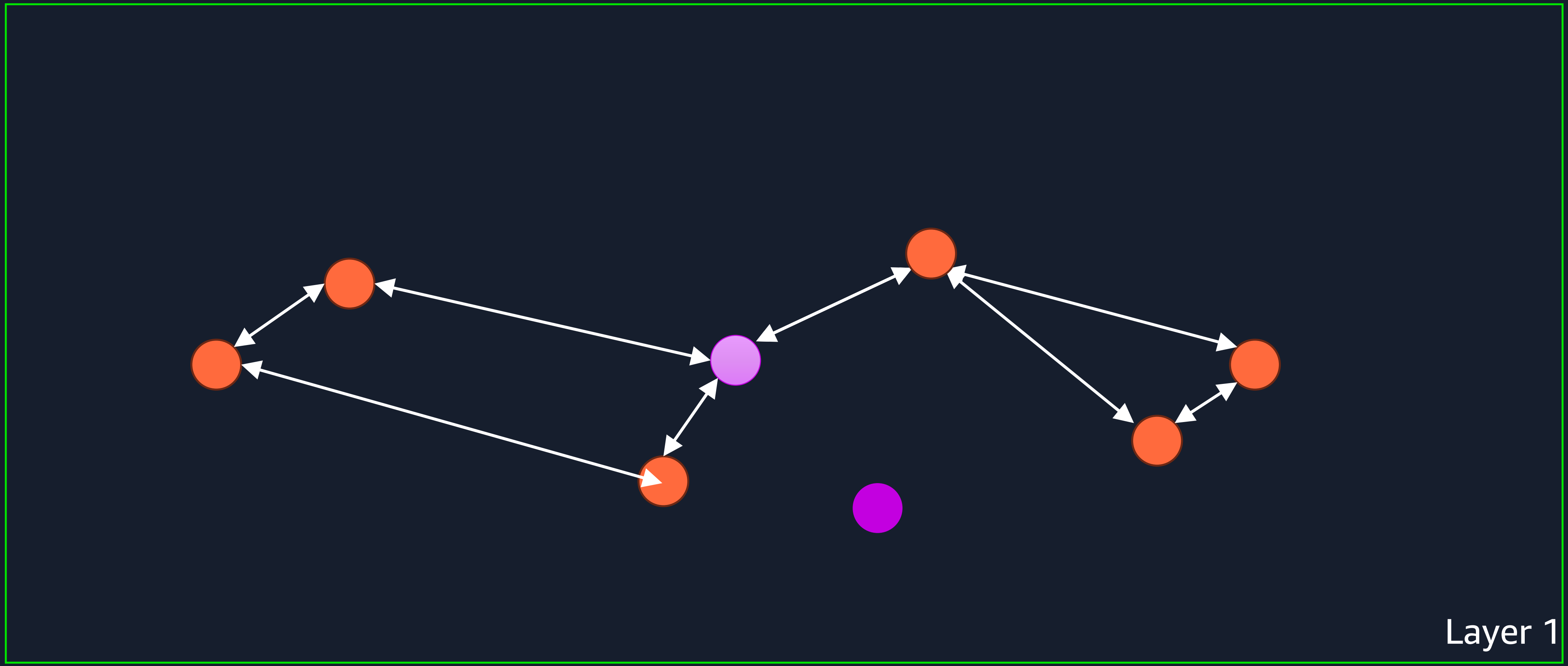
Querying an HNSW index



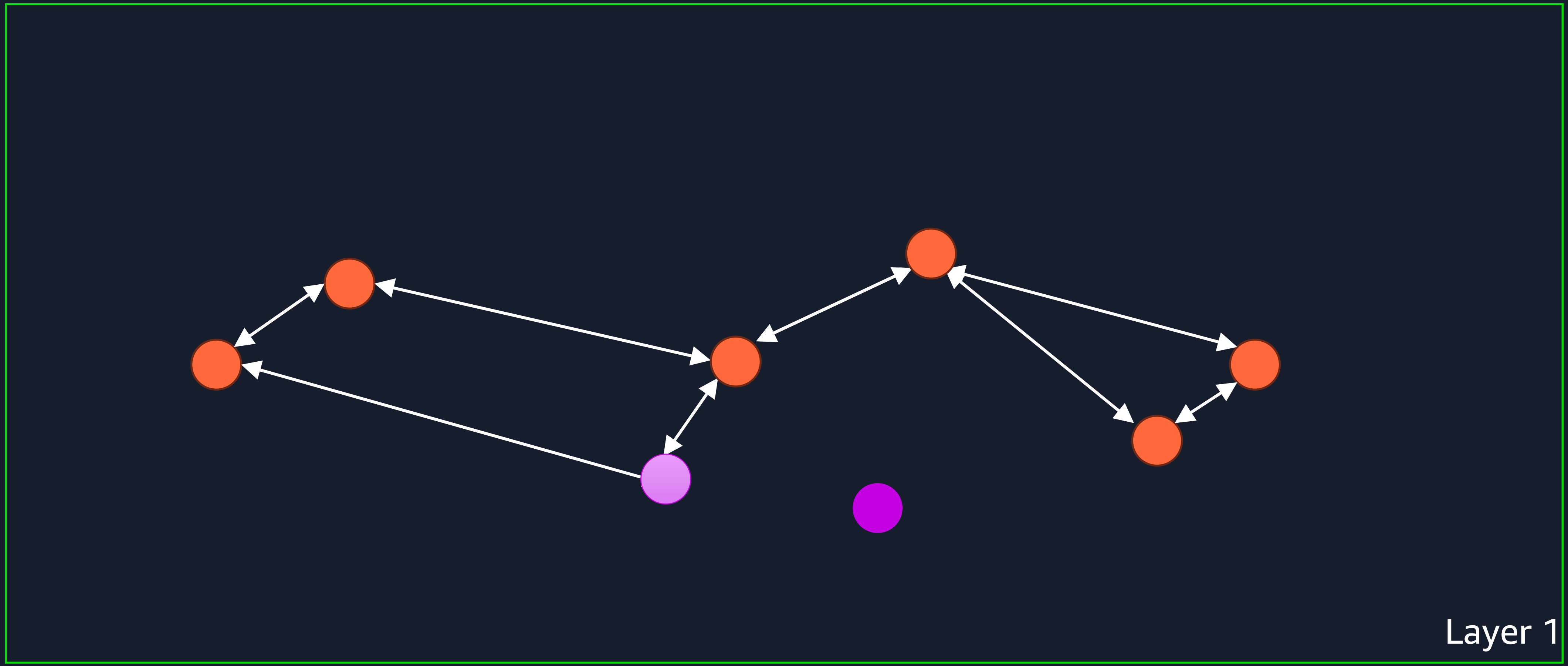
Querying an HNSW index



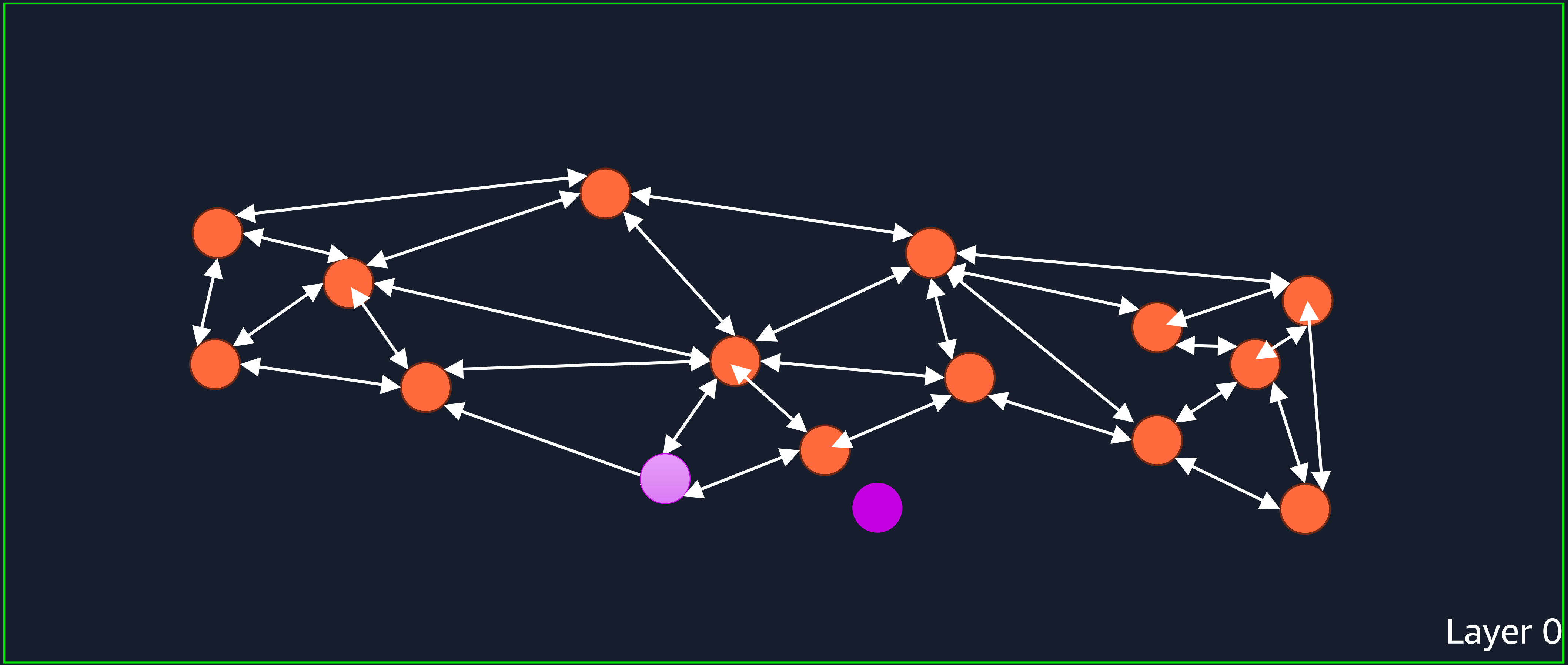
Querying an HNSW index



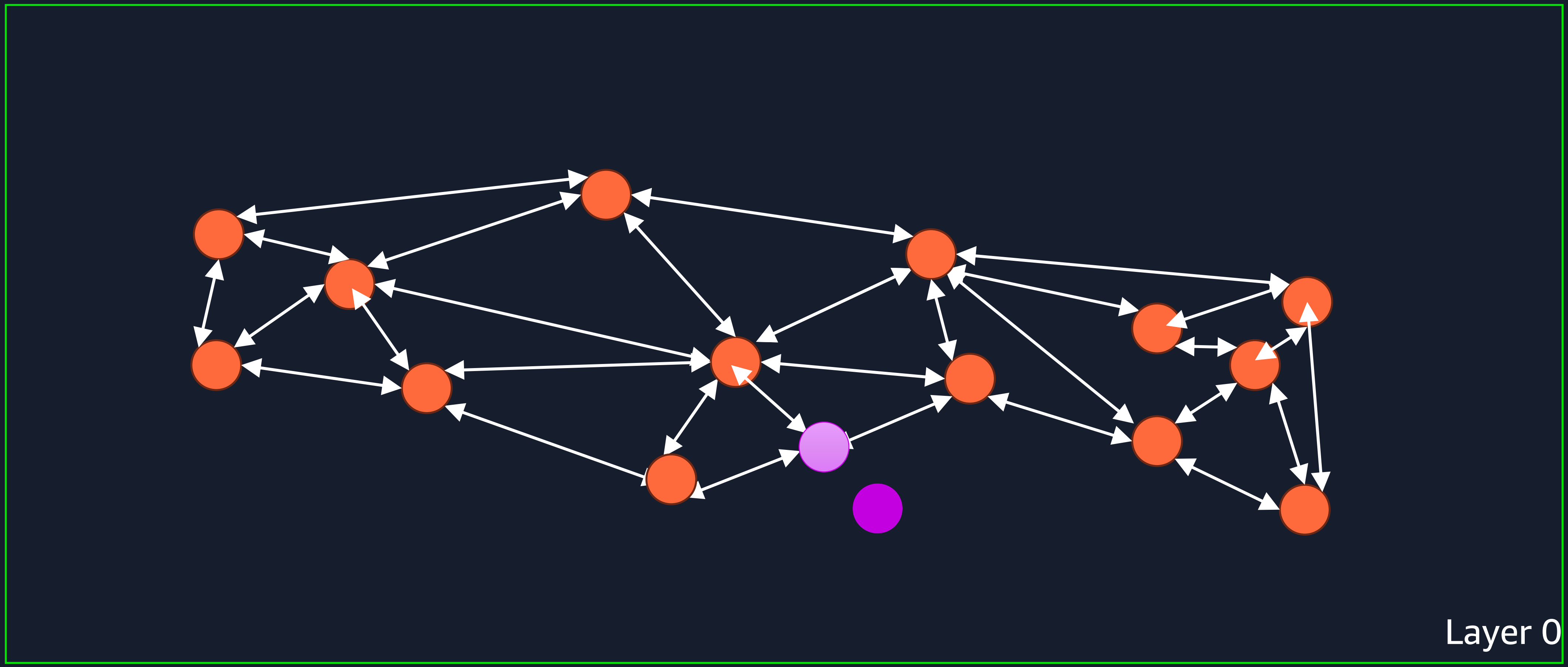
Querying an HNSW index



Querying an HNSW index



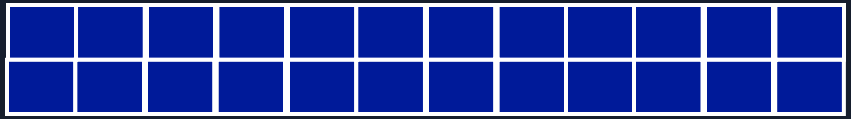
Querying an HNSW index



Quantization

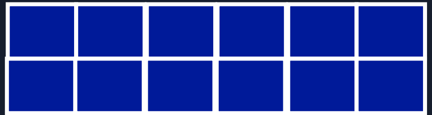
Flat

[0.0435122, -0.2304432, -0.4521324,
0.98652234, -0.1123234, 0.75401234]



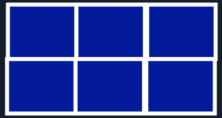
Scalar Quantization (2-byte float)

[0.0432, -0.234, -0.452,0.986,
-0.112, 0.751]



Scalar Quantization (1-byte uint)

[129, 99, 67, 244, 126, 230]



Binary Quantization

[1, 0, 0, 1, 0, 1]

pgvector and Quantization

```
-- 2-byte float (fp16) quantization
CREATE INDEX ON documents USING
      hnsw((embedding::halfvec(3072)) halfvec_cosine_ops);

SELECT id
FROM documents
ORDER BY embedding::halfvec(3072) <=> $1::halfvec(3072)
LIMIT 10;
```

pgvector and Quantization

```
-- Binary quantization
CREATE INDEX ON documents USING
      hnsw ((binary_quantize(embedding)::bit(3072)) bit_hamming_ops);

SELECT id FROM documents
ORDER BY binary_quantize(embedding)::bit(3072) <~> binary_quantize($1)
LIMIT 10;

-- Rerank query for binary quantization
SELECT i.id FROM (
      SELECT id, embedding <=> $1 AS distance
      FROM items
      ORDER BY binary_quantize(embedding)::bit(3072) <~> binary_quantize($1)
      LIMIT 800 -- bound by hnsw.ef_search
) i
ORDER BY i.distance LIMIT 10;
```

Scalar quantization

dbpedia-openai-1m-angular (1MM 1,536-dim); m=16; ef_construction=256

	No Quantization	2-byte float quantization
Index size (MB)	7734	3867
Index build time (s)	250	146
Recall @ ef_search=10	0.851	0.854
QPS @ ef_search=10	1154	1164
Recall @ ef_search=40	0.967	0.968
QPS @ ef_search=40	567	583
Recall @ ef_search=200	0.996	0.996
QPS @ ef_search=200	158	163

Binary quantization

dbpedia-openai-1m-angular (1MM 1,536-dim); m=16; ef_construction=256

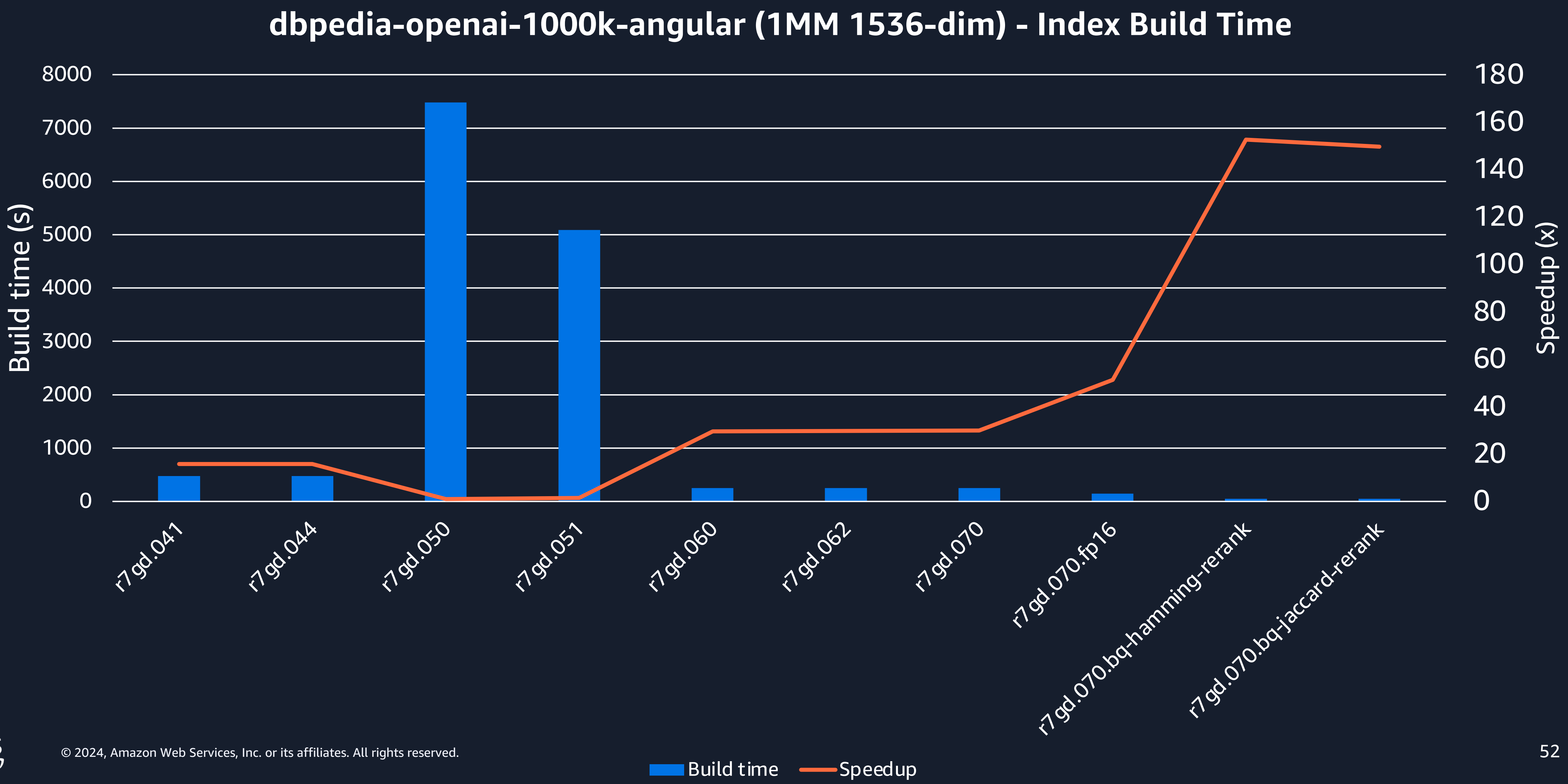
	No Quantization	Binary quantization / rerank
Index size (MB)	7734	473
Index build time (s)	250	49
Recall @ ef_search=10	0.851	0.604
QPS @ ef_search=10	1154	1687
Recall @ ef_search=40	0.967	0.916
QPS @ ef_search=40	567	883
Recall @ ef_search=200	0.996	0.990
QPS @ ef_search=200	158	236

Quantization limitations

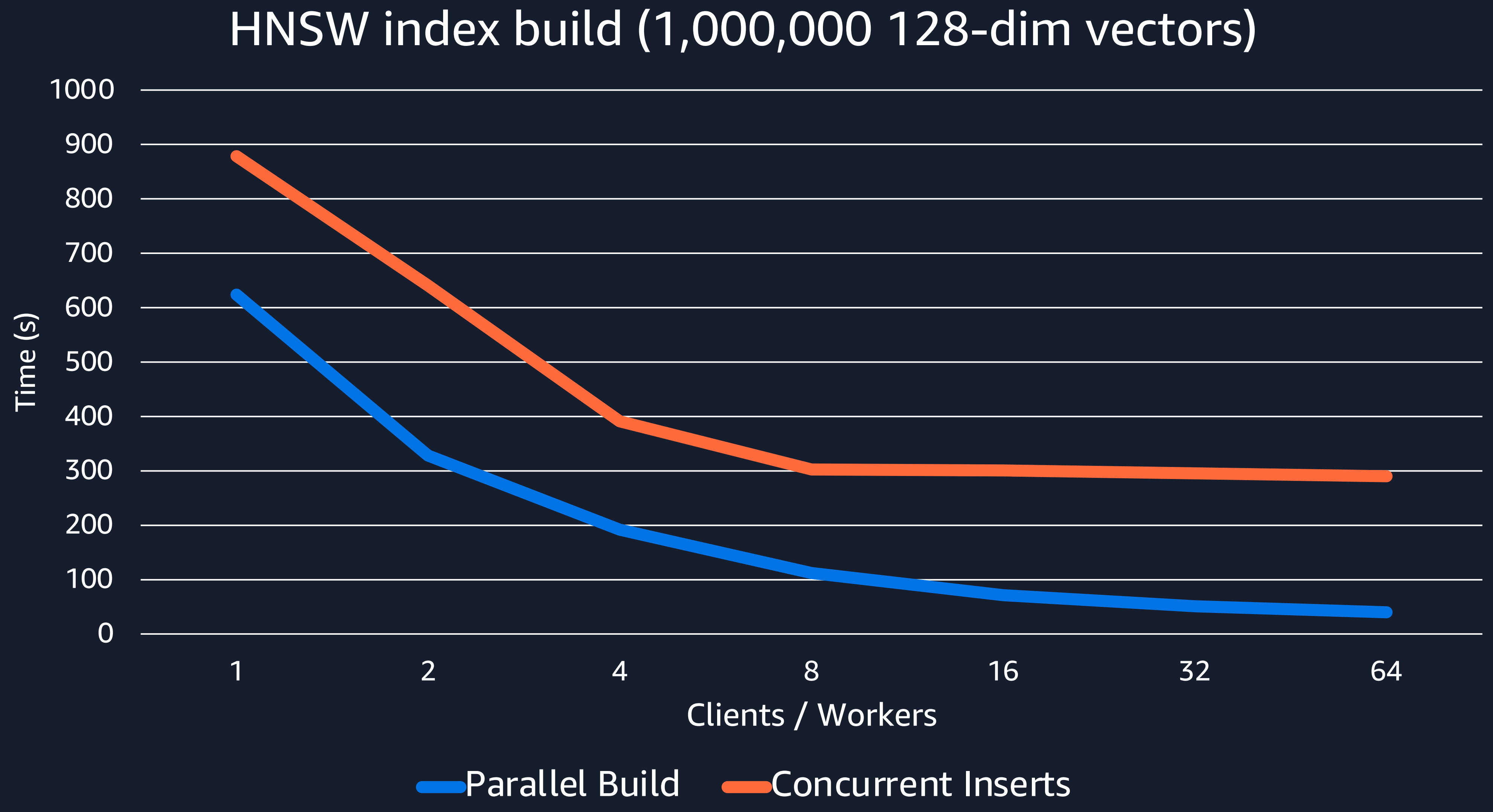
- Reduction of information
- Law of large numbers / curse of dimensionality
- "Double storage" – heap / index

A year of pgvector in charts

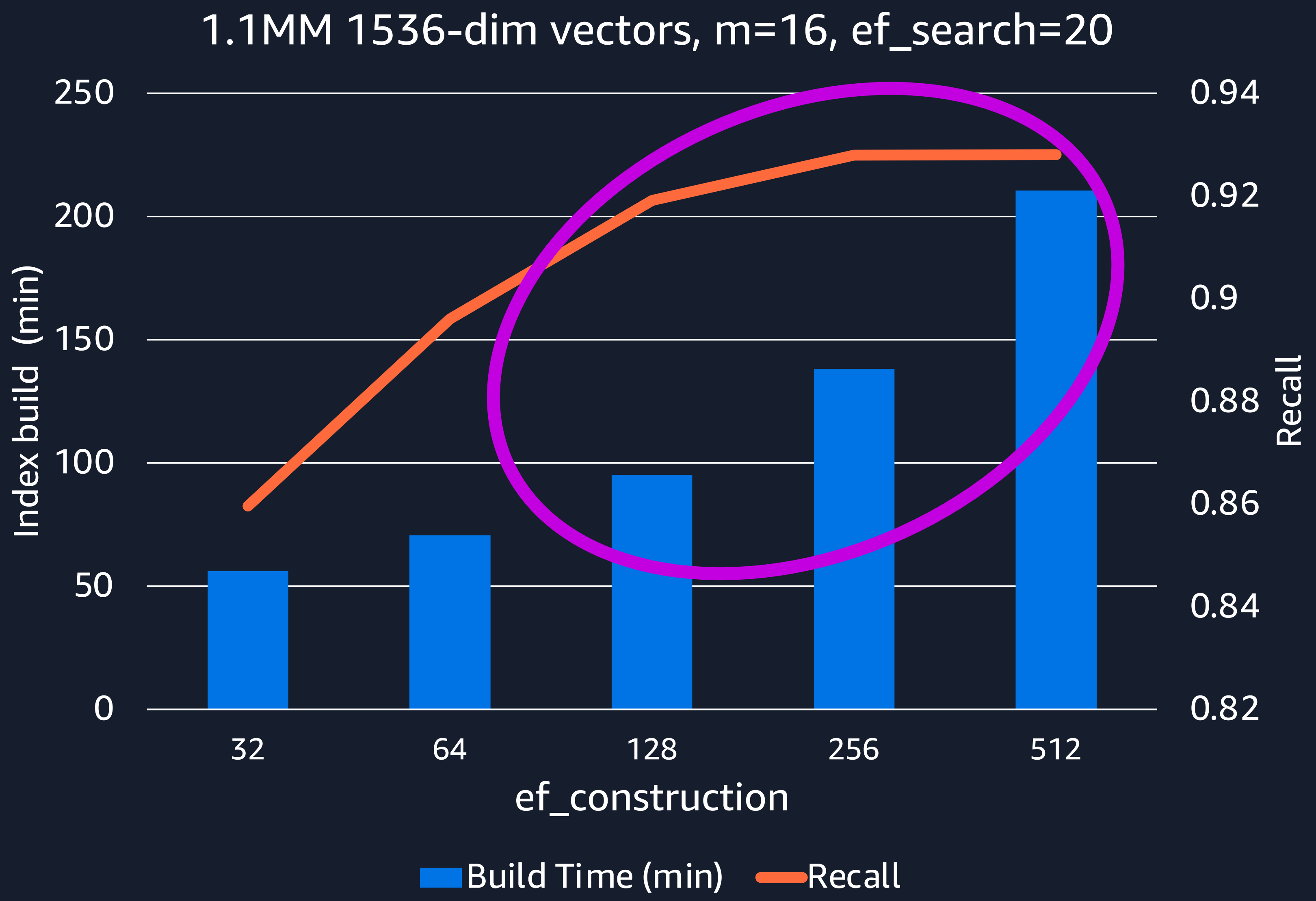
pgvector index build time



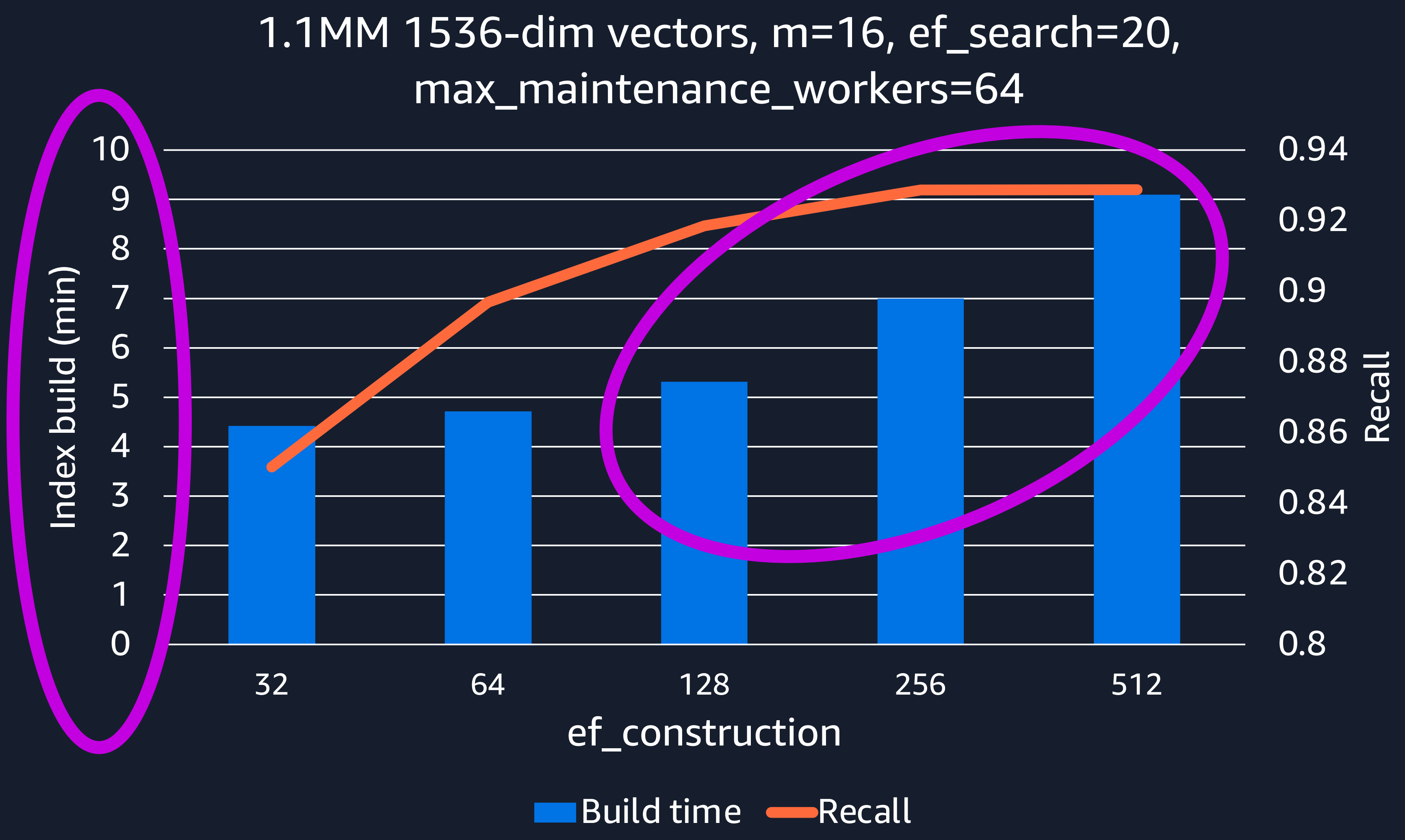
Impact of parallelism on HNSW build time



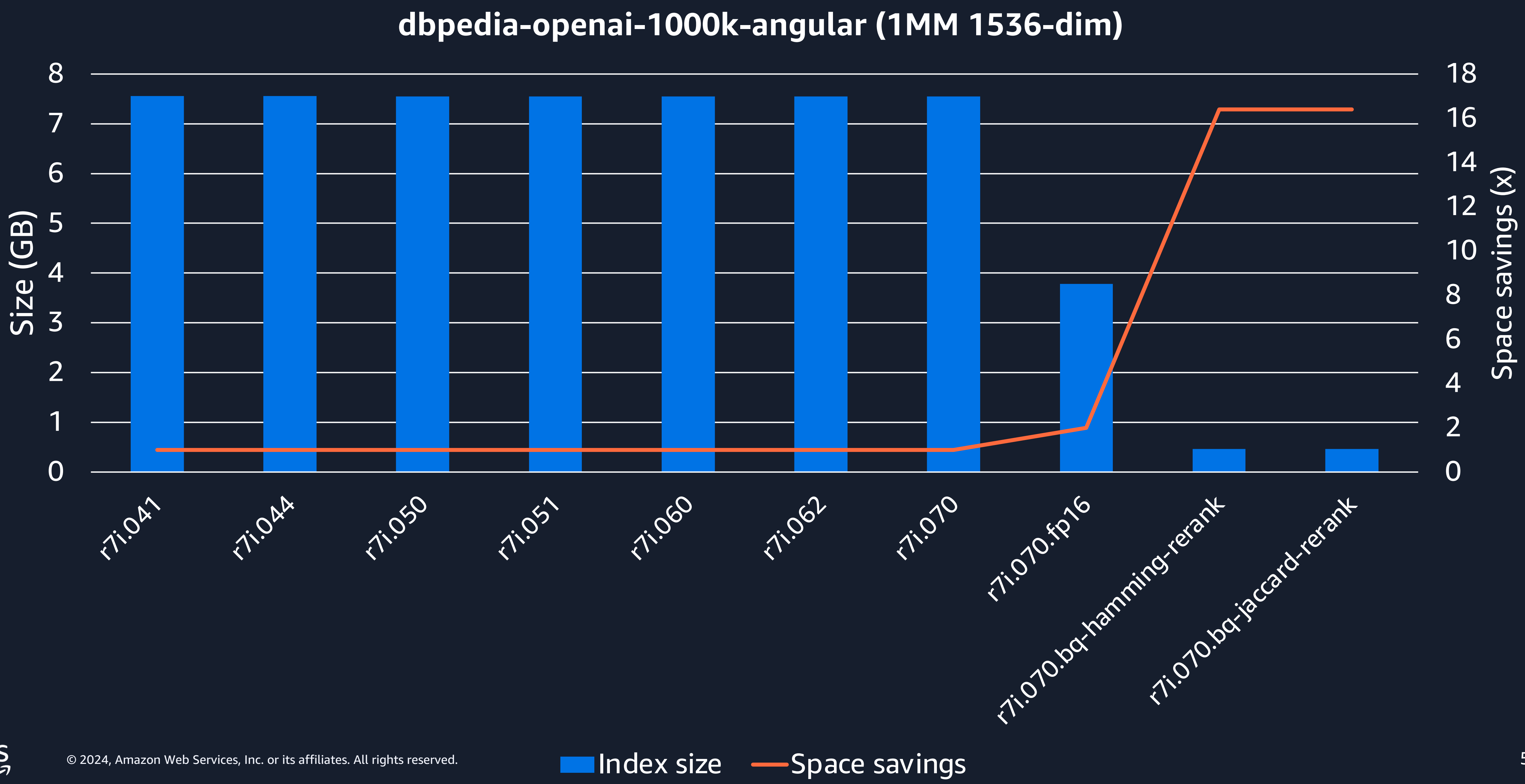
Why index build speed matters (Serial build)



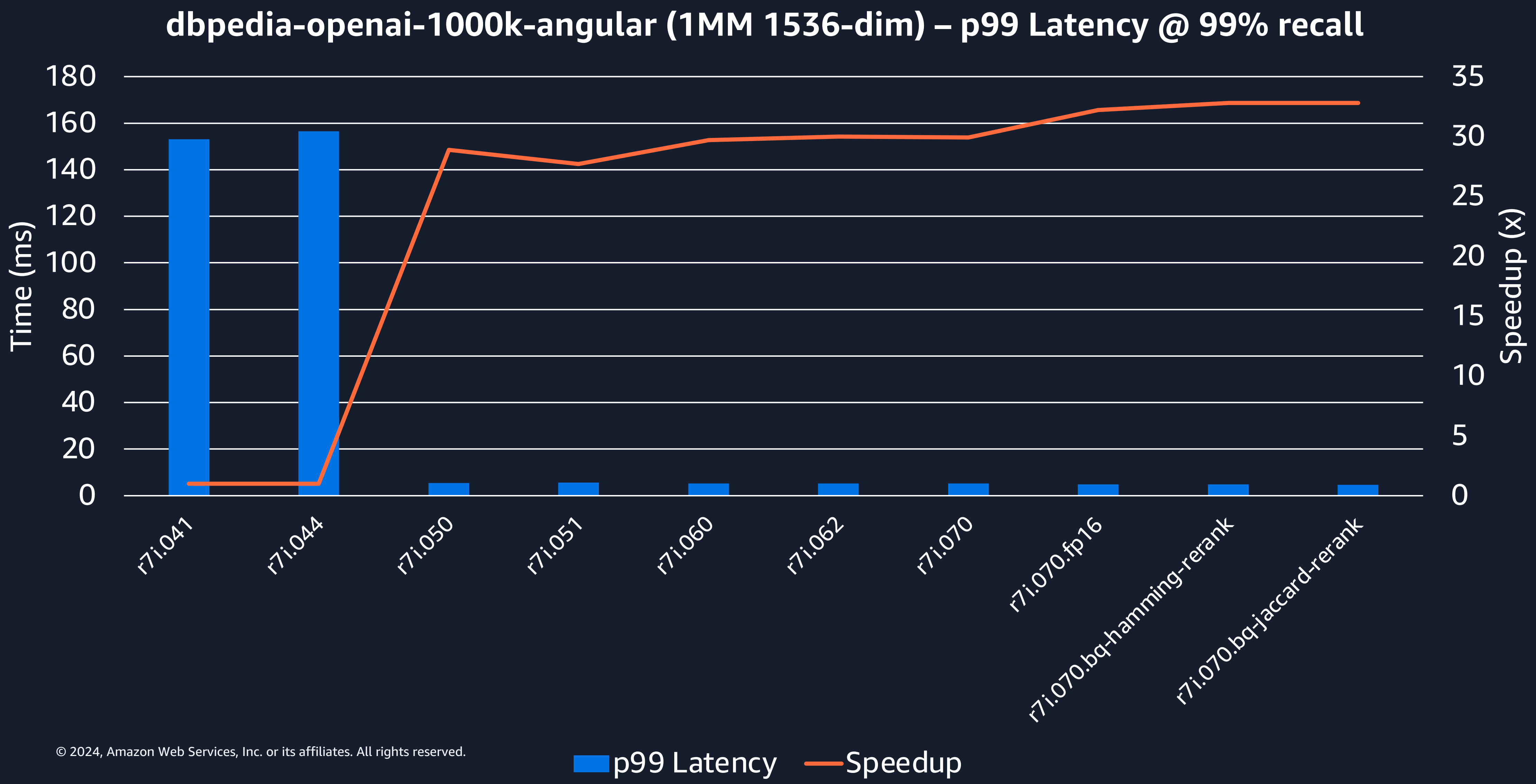
Why index build speed matters (Parallel build)



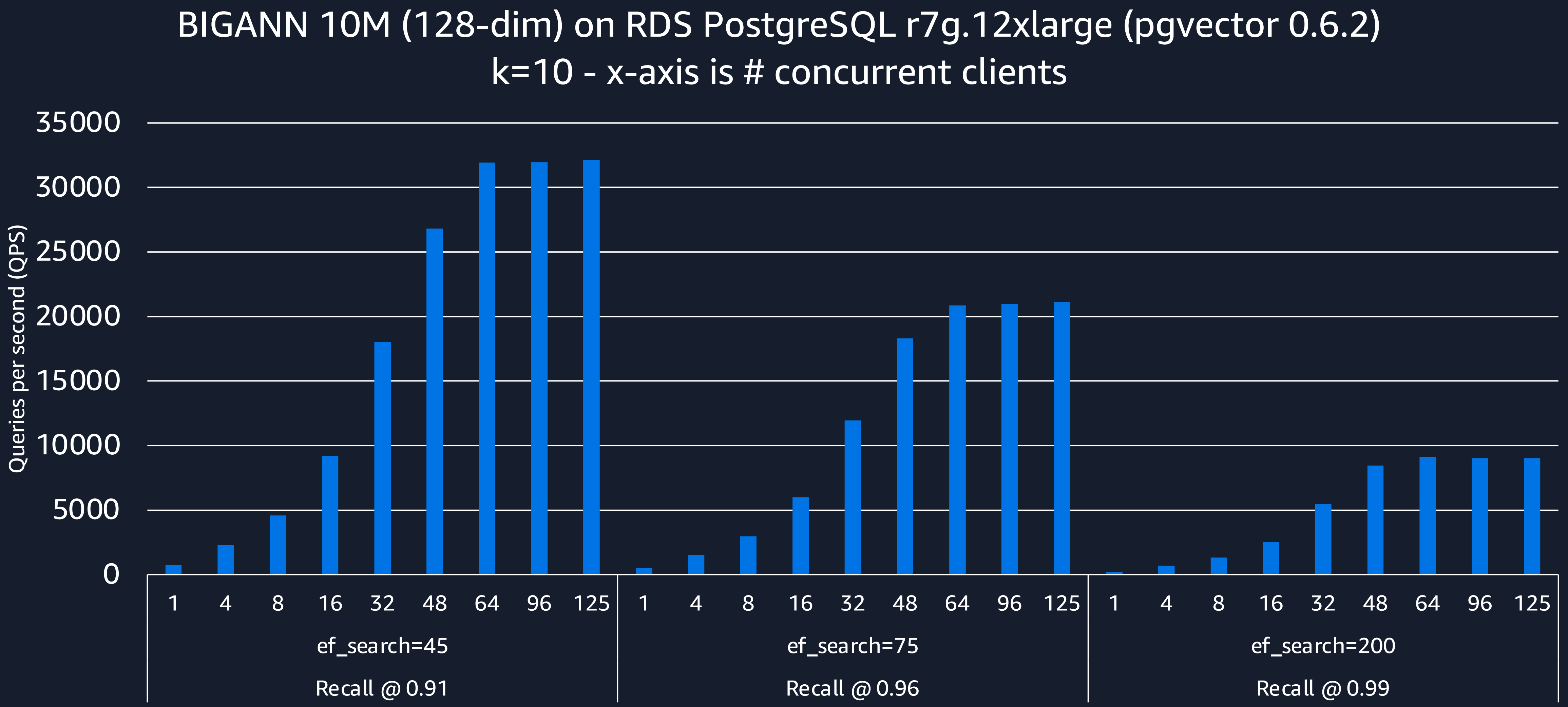
pgvector index size



pgvector query latency (p99)



pgvector throughput



Ongoing challenges and recommendations

The 8K conundrum

- Page: PostgreSQL atomic storage unit
  - 8192 bytes = 8K = 8KiB

- Heap (table) pages are resizable as a compile time flag

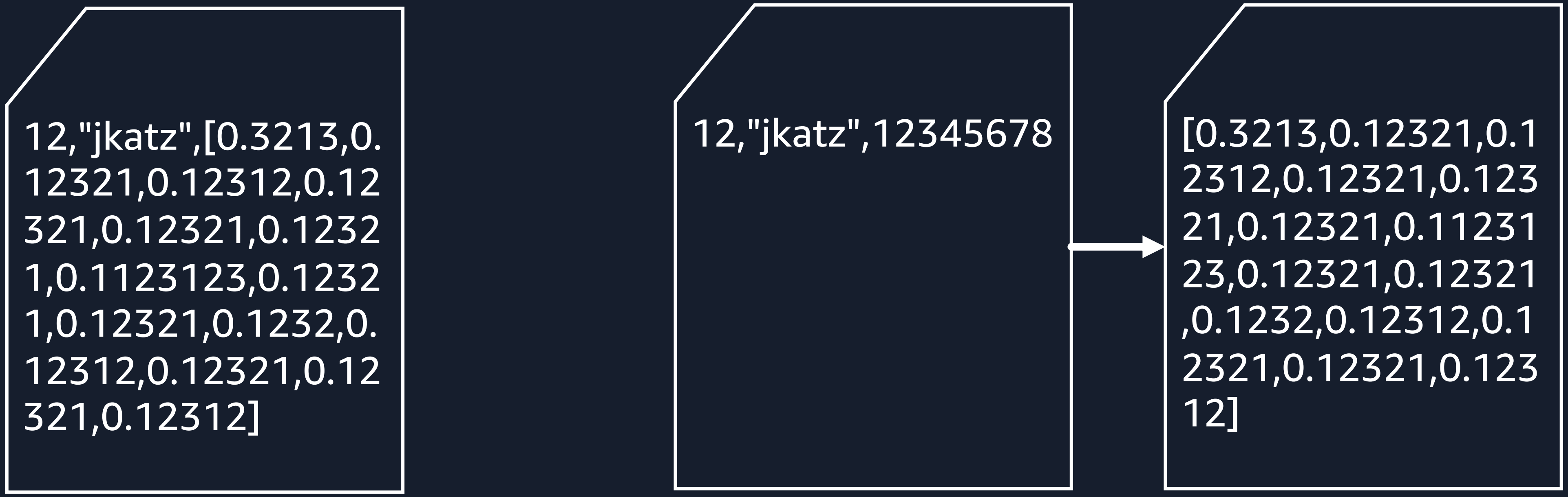
- Index pages are not resizable

- This is a real (😉) problem for vectors
  - 1536-dim 4-byte vector = 6KiB
  - 3072-dim 4-byte vector = 12KiB

🍞 Can't we can TOAST?

- TOAST (The Oversized-Attribute Storage Technique) is a mechanism for storing data larger than 8KB
  - By default, PostgreSQL “TOASTs” values over 2KB (510d 4-byte float)
- Storage types:
  - `PLAIN`: Data stored inline with table
  - EXTENDED: Data stored/compressed in TOAST table when threshold exceeded
    - pgvector default before 0.6.0
  - EXTERNAL: Data stored in TOAST table when threshold exceeded
    - pgvector default 0.6.0+
  - `MAIN`: Data stored compressed inline with table

Visualizing TOAST for pgvector



12,"jkatz",[0.3213,0.12321,0.12312,0.12321,0.12321,0.12321,0.1123123,0.12321,0.12321,0.1232,0.12312,0.12321,0.12321,0.12312]

12,"jkatz",12345678

[0.3213,0.12321,0.12312,0.12321,0.12321,0.12321,0.1123123,0.12321,0.12321,0.1232,0.12312,0.12321,0.12321,0.12312]

PLAIN

EXTENDED / EXTERNAL

Impact of TOAST on vector data

- Traditionally, TOAST data is not on the "hot path"
  - Impacts query plan and maintenance operations

- Compression is ineffective

- Unable to use for index pages

Impact of TOAST on pgvector queries

```
Limit (cost=772135.51..772136.73 rows=10 width=12)
-> Gather Merge (cost=772135.51..1991670.17 rows=10000002 width=12)
    Workers Planned: 6
    -> Sort (cost=771135.42..775302.08 rows=1666667 width=12)
        Sort Key: ((<-> embedding))
        -> Parallel Seq Scan on vecs128 (cost=0.00..735119.34 rows=1666667
width=12)
```

128 dimensions

Impact of TOAST on pgvector queries

```
Limit (cost=149970.15..149971.34 rows=10 width=12)
-> Gather Merge (cost=149970.15..1347330.44 rows=10000116 width=12)
    Workers Planned: 4
    -> Sort (cost=148970.09..155220.16 rows=2500029 width=12)
        Sort Key: (($1 <-> embedding))
        -> Parallel Seq Scan on vecs1536 (cost=0.00..94945.36 rows=2500029
width=12)
```

1,536 dimensions

Impact of TOAST on pgvector queries

```
Limit (cost=95704.33..95705.58 rows=10 width=12)
-> Gather Merge (cost=95704.33..1352239.13 rows=10000111 width=12)
    Workers Planned: 11
    -> Sort (cost=94704.11..96976.86 rows=909101 width=12)
        Sort Key: (($1 <-> embedding))
        -> Parallel Seq Scan on vecs1536 (cost=0.00..75058.77 rows=909101 width=12)
```

1,536 dimensions

```
SET min_parallel_table_scan_size TO 1
```

Improving PostgreSQL storage of vector data

- Continue investing in quantization
- Improve planner to understand when TOAST data is part of the hot path
- TOAST / page chaining system for index pages
- Modifiable size for index pages

Filtering

```
SELECT id
FROM products
WHERE products.category_id = 7
ORDER BY :'q' <-> products.embedding
LIMIT 10;
```

How filtering impacts ANN queries

- PostgreSQL may choose to not use the index
- Uses an index, but does not return enough results
- Filtering occurs after using the index

Current filtering strategies

- Partial index
- Partition

```
CREATE INDEX ON docs
  USING hnsw(embedding vector_l2_ops)
  WHERE category_id = 7;
---
CREATE TABLE docs_cat7
  PARTITION OF docs
  FOR VALUES IN (7);

CREATE INDEX ON docs_cat7
 USING hnsw(embedding vector_l2_ops);
```

Filtering with "hybrid search"

```
SELECT id
FROM products
WHERE
     plainto_tsquery('english', 'elephant vase') @@
          to_tsvector('english', description)
ORDER BY :'q' <=> embedding
LIMIT 10;
```

Improving filtering with vector data in PostgreSQL

- "Multi-column" indexes

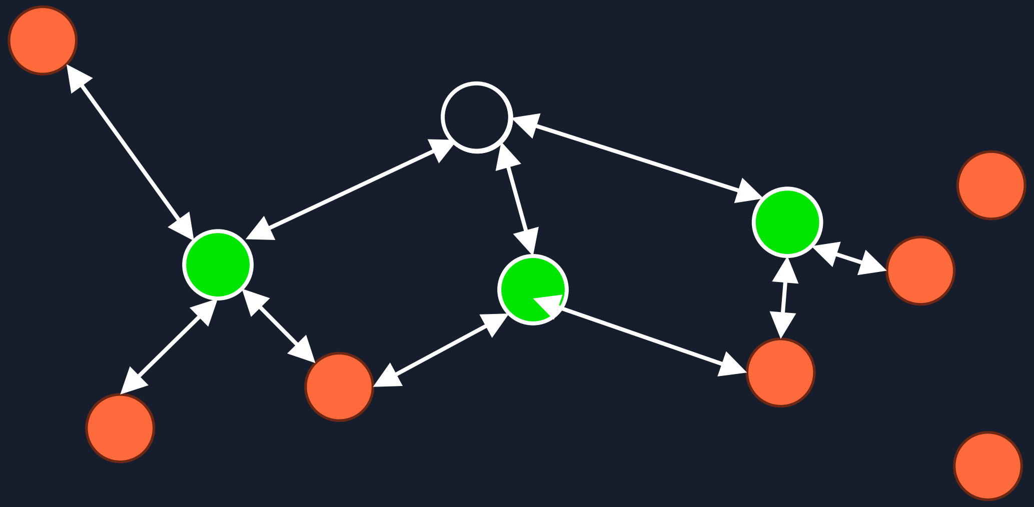
- Remove extra distance calculation executions when filtering junk columns
  - Index-only scans

- Pushdown to covering indexes

- Using other index mechanisms to filter data set

pgvector and VACUUM

- Innovation: pgvector HNSW implementation supports updates and deletes!



Phase 1: Hide

Phase 2: Repair

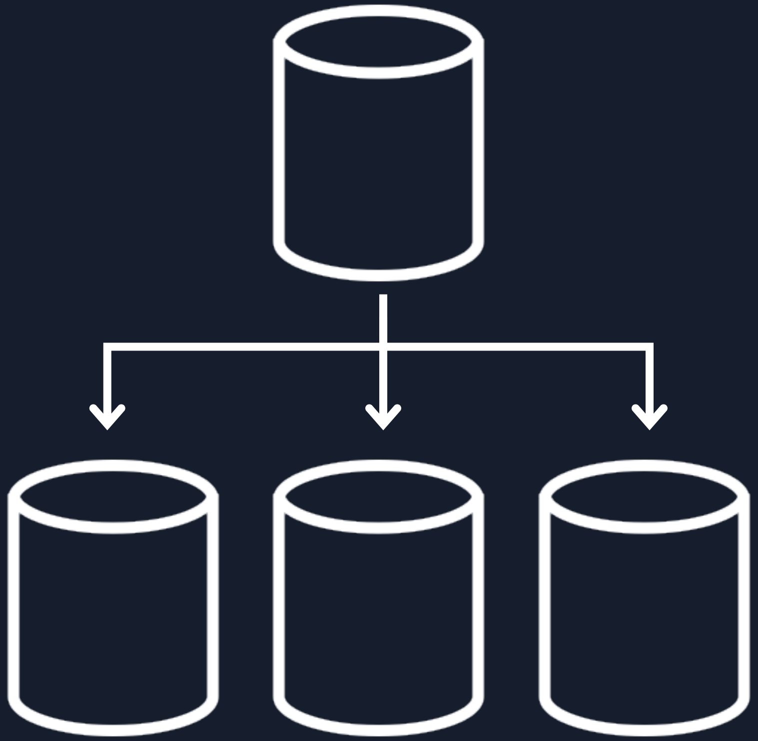
Improving VACUUM for pgvector

- Framework for parallel vacuum of custom index types

- Anything that can simplify implementing VACUUM :-)

Distributed queries for pgvector – why?

- Not enough memory for workload to meet latency target
- Network overhead must be acceptable
- Can manage complexity of multi-node system



Setup foreign data wrapper

```
CREATE EXTENSION IF NOT EXISTS vector;
CREATE EXTENSION IF NOT EXISTS postgres_fdw;

CREATE SERVER vectors1
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (
  async_capable 'true', extensions 'vector', dbname 'vectors', host
'<NODE1>'
);

CREATE SERVER vectors2
FOREIGN DATA WRAPPER postgres_fdw
OPTIONS (
  async_capable 'true', extensions 'vector', dbname 'vectors', host
'<NODE2>'
);
```

Setup foreign tables

```
CREATE TABLE vectors (
  id uuid,
  node_id int,
  embedding vector(768)
) PARTITION BY LIST(node_id);

CREATE FOREIGN TABLE vectors_node1 PARTITION OF vectors
  FOR VALUES IN (1)
  SERVER vectors1
  OPTIONS (schema_name 'public', table_name 'vectors');

CREATE FOREIGN TABLE vectors_node2 PARTITION OF vectors
  FOR VALUES IN (2)
  SERVER vectors2
  OPTIONS (schema_name 'public', table_name 'vectors');
```

Example EXPLAIN output

```
Limit (cost=200.01..206.45 rows=10 width=28) (actual time=18.171..18.182 rows=10
loops=1)
  -> Merge Append (cost=200.01..3222700.01 rows=5000000 width=28) (actual
time=18.169..18.179 rows=10 loops=1)
  Sort Key: (('$1'::vector <=> vectors.embedding))
    -> Foreign Scan on vectors_node1 vectors_1 (cost=100.00..1586350.00
rows=2500000 width=28) (actual time=8.607..8.609 rows=2 loops=1)
    -> Foreign Scan on vectors_node2 vectors_2 (cost=100.00..1586350.00
rows=2500000 width=28) (actual time=9.559..9.566 rows=9 loops=1)

Planning Time: 0.298 ms
Execution Time: 19.355 ms
```

Parallel query for pgvector

- pgvector doesn't support parallel query
    - Benefits IVFFlat more than HNSW

- Index AM won't let PostgreSQL choose parallel plan

Hardware acceleration for pgvector

- Index building (esp. HNSW) uses most computation time
    - Can see increased CPU utilization with higher hnsw.ef_search

- pgvector uses compiler autovectorization, but has started adding explicit dispatching instructions

- Newer CPU architectures contain more instructions for SIMD, but may not be widely available

- GPU – huge penalty to move to GPU memory without GPUDirect
    - Index building could benefit from GPU

"Dogs not barking"

- Matrices / tensors
- Storage type / capacity
- Native vector support for PostgreSQL

Summary and next steps

(Prioritized) summary of areas to improve

- Filters
    - Hybrid search
- Parallelized vacuum
- Better async pushdown for postgres_fdw
- Additional hardware acceleration methods
- Parallel query
- TOAST / storage updates

Community contributions make pgvector better

- Improved locking during HNSW build
- SIMD dispatching for distance functions
- Integration of upstream SIMD support (pg_popcount)
- Memory allocation / usage optimizations during index builds
- Identify common search patterns and help prioritize

Thank you!

Jonathan Katz

jkatz@amazon.com

@jkatz05