FROM TEXT TO MEANING: PGVECTOR REVOLUTIONIZES POSTGRESQL SEARCH

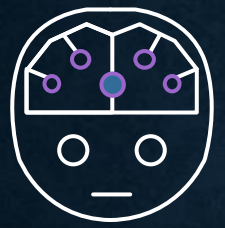
Kiran Janarthan Singh
Sr Database Specialist SA
Amazon Web Services

Vivek Singh
Principal Database Specialist – PostgreSQL
Amazon Web Services

AGENDA

- Introduction
- What is vector and embedding
- Search concept
- PostgreSQL as vector store
- Vector indexes deep dive
- Demo: AI powered similarity search using pgvector
- Questions

GENERATIVE AI



Artificial intelligence (AI)

Any technique that allows computers to mimic human intelligence using logic, if-then statements, and machine learning

Machine learning (ML)

A subset of AI that uses machines to search for patterns in data to build logic models automatically

Deep learning (DL)

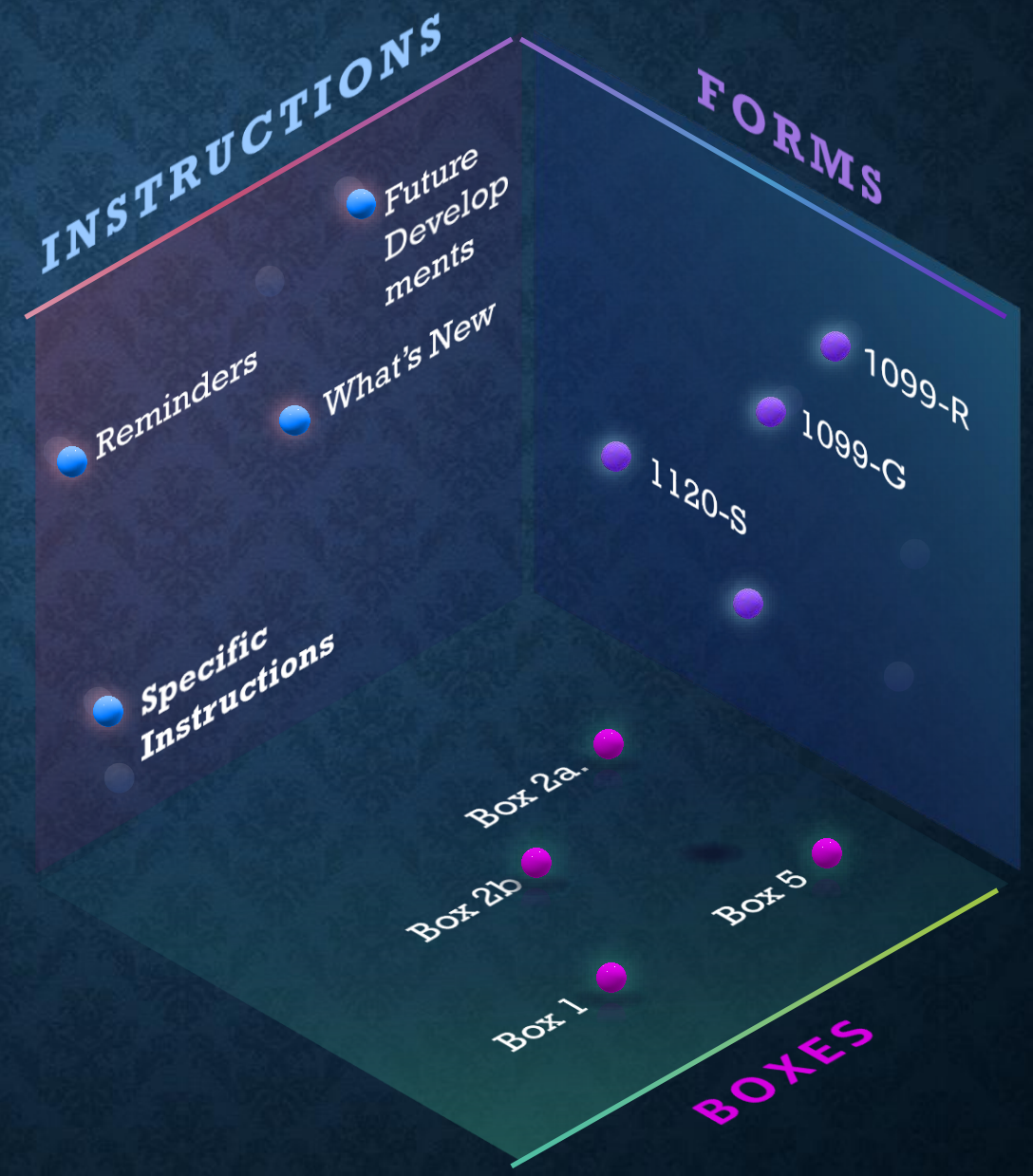
A subset of ML composed of deeply multi-layered neural networks that perform tasks like speech and image recognition

Generative AI

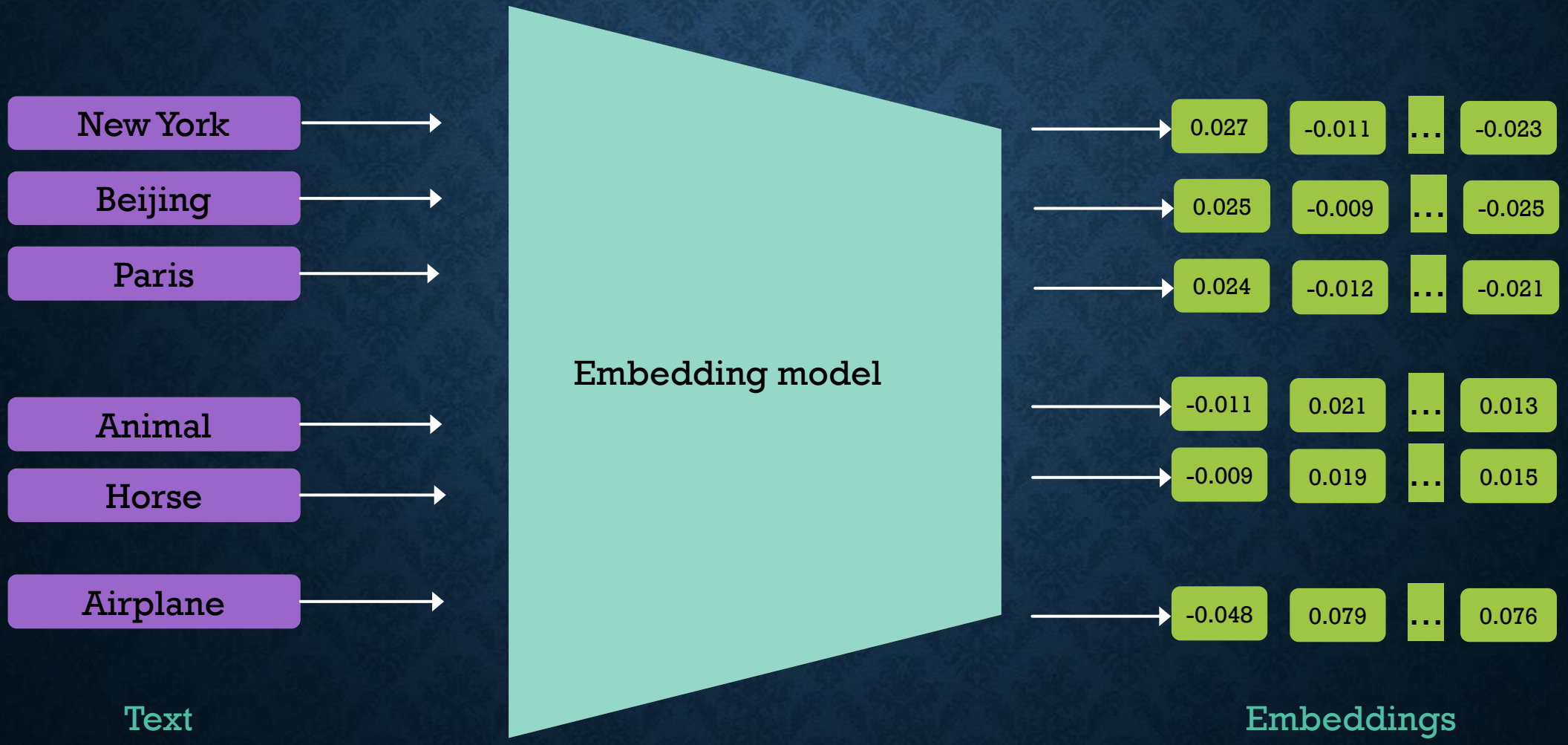
Powered by **large models** that are pre-trained on vast corpora of data and commonly referred to as **foundation models (FMs)**

VECTOR SPACE

EMBEDDINGS REPRESENTING SIMILAR CONTEXT/VECTORS CAN FORM CLUSTERS

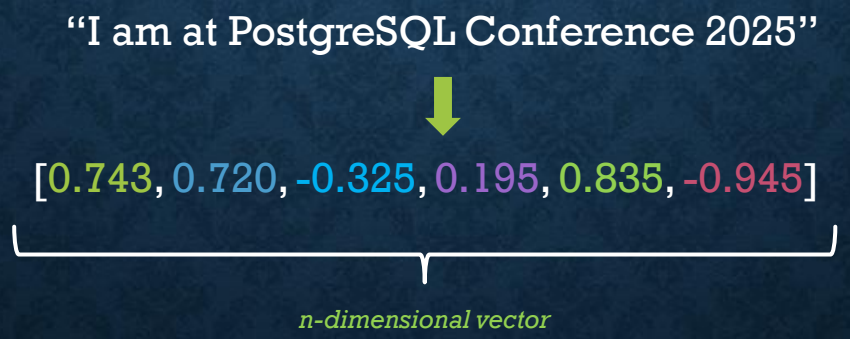


WHAT ARE VECTOR EMBEDDINGS?



WHAT IS A VECTOR EMBEDDING ?

- A numerical representation of words or sentences, used in NLP
- NLP models can easily perform tasks such as querying, classification, and applying machine learning algorithms on textual data



VECTOR DATABASES

- Pinecone
- Milvus
- Weaviate
- Chroma
- Pgvector

POSTGRESQL AS A VECTOR STORE

RISKS OF NOT UPGRADING POSTGRESQL DATABASE

Limitation	Limitations of traditional text search
Semantics	Need for semantic understanding
Demand	Growing demand for context-aware search
Evolution	Evolution from keywords to meaning

WHY USE POSTGRESQL FOR VECTOR SEARCHES?

- Existing client libraries work without modification
- Convenient to co-locate app and AI/ML data in same database
- PostgreSQL acts as persistent transactional store while working with other vector search systems

NATIVE VECTOR SUPPORT AND CHALLENGES

ARRAY data type

- Multiple data types (int4, int8, float4, float8)
- "Unlimited" dimensions
- No native distance operations
  - Can add using Trusted Language Extensions + PL/Rust
- No native indexing

Cube data type

- float8 values
- Euclidean, Manhattan, Chebyshev distances
- K-NN GiST index – exact nearest neighbor search
- Limited to 100 dimensions

WHAT IS PGVECTOR?

Support for storage, indexing, searching, metadata with choice of distance

vector data types

halfvec type to store half-precision vectors (*Added in 0.7.0*)

Supports IVFFlat/HNSW indexing

Exact nearest neighbor (K-NN)
Approximate nearest neighbor (ANN)

Co-locate with embeddings

Distance operators (<->, <=>, <#>, <+>, <~>, <%>)

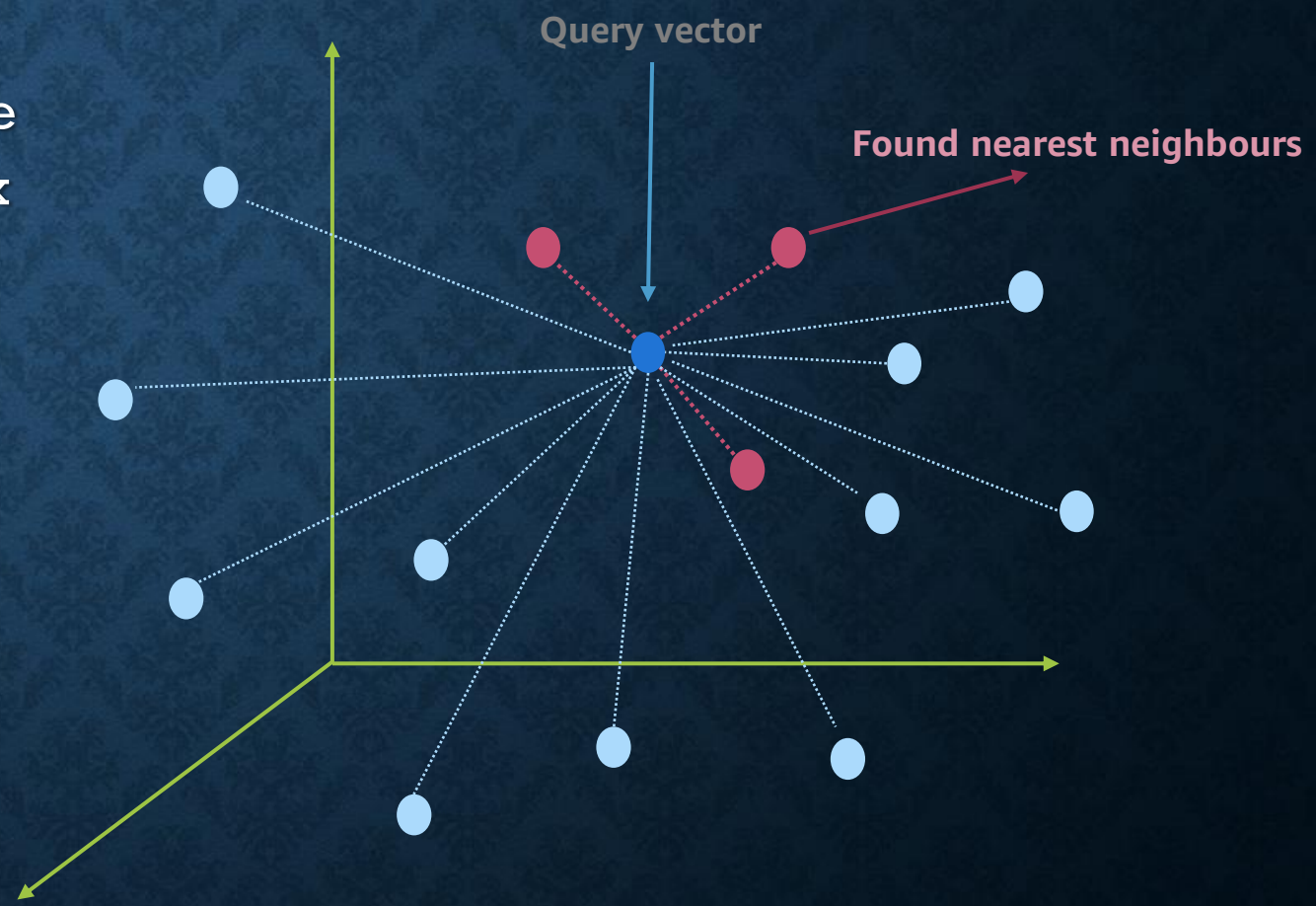
github.com/pgvector/pgvector
Note: <+>, <~>, and <%> operators available only from pgvector version 0.7.0

CONCEPT OF AN EXACT NEAREST NEIGHBOURS SEARCH

K NEAREST NEIGHBOR (K-NN)

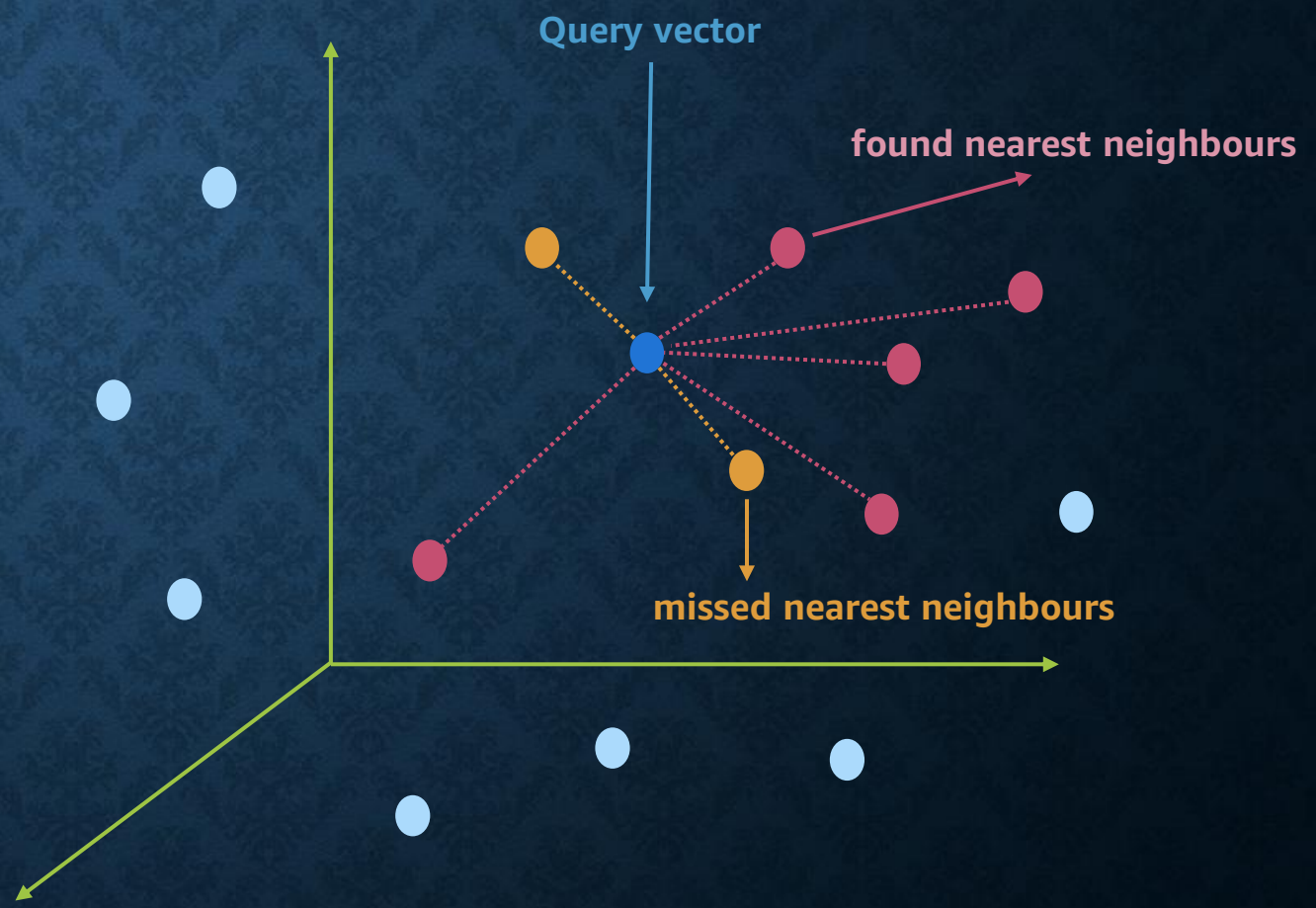
- For exact match (100% recall) search on the PostgreSQL vector column without an index

- K-NN searches find the nearest neighbors for a query by comparing its vector to all stored vectors and returning the k closest ones.



APPROXIMATE NEAREST NEIGHBOR (ANN)

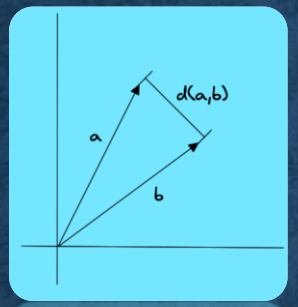
- Find similar vectors without searching all of them
- Faster than exact nearest neighbor
- "Recall" – % of expected results



PGVECTOR: OFFERS DISTANCE OPERATIONS

Euclidean (L2)

Useful for counts / measurements
Recommendation Systems

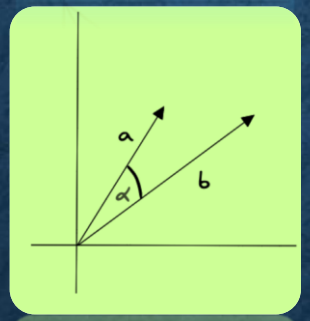


$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

<->

Cosine Similarity

Useful for semantic search and document classification

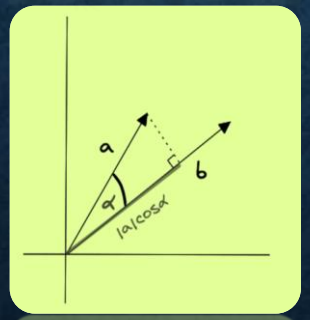


$$sim(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}|| \cdot ||\mathbf{b}||}$$

<=>

Dot Product

Useful for collaborative filtering



$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}||\mathbf{b}| cos\alpha$$

<#>

PGVECTOR EXAMPLE: QUERYING NEAREST NEIGHBOR

- Supports exact and approximate nearest neighbor (ANN) search
  - L2 distance <->
  - Inner product <#>
  - Cosine distance <=>

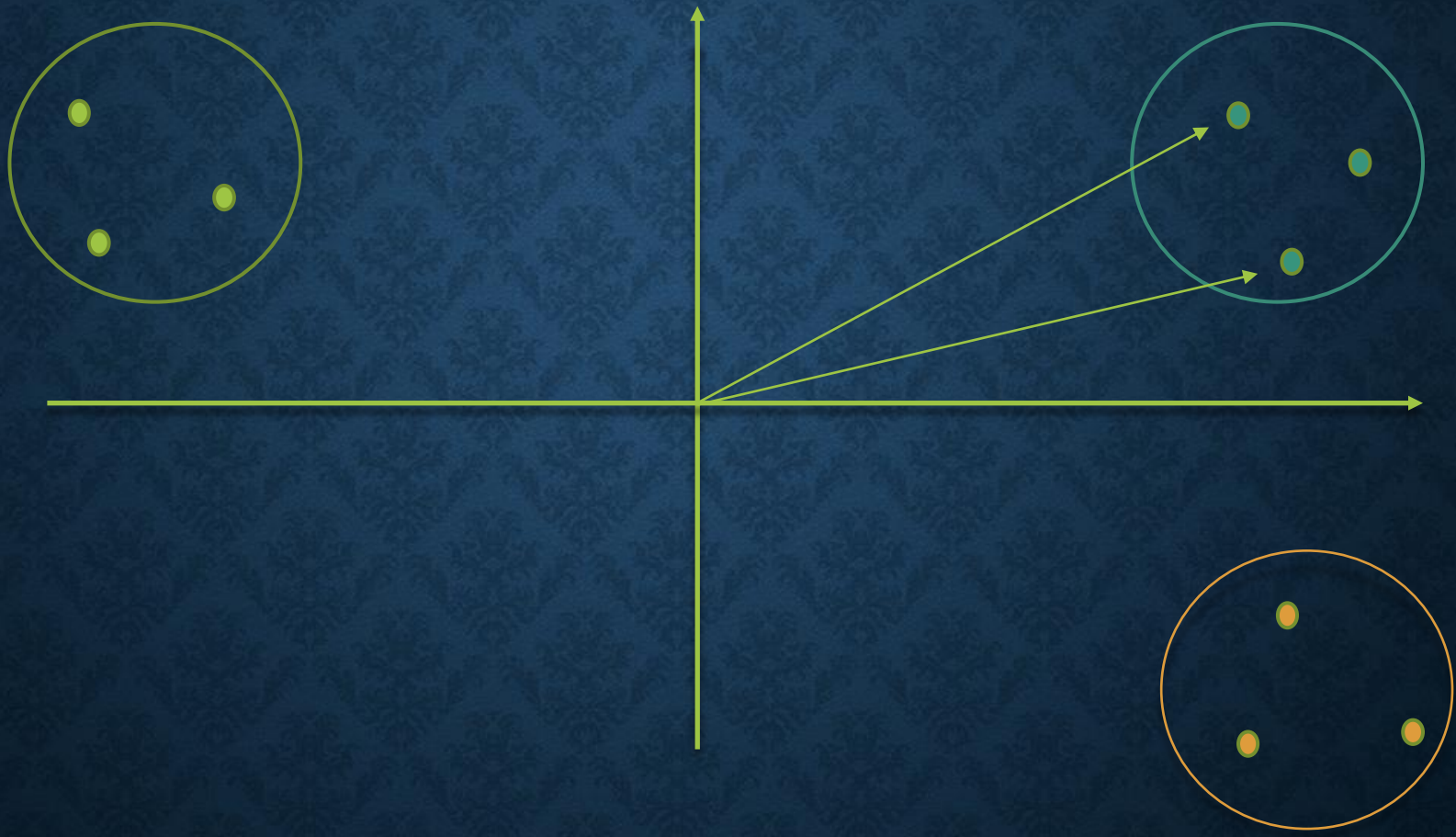
```
CREATE TABLE test_embeddings(product_id bigint, embeddings vector(3) );
INSERT INTO test_embeddings VALUES
(1, '[1, 2, 3]'), (2, '[2, 3, 4]'), (3, '[7, 6, 8]'), (4, '[8, 6, 9]’);

SELECT product_id, embeddings, embeddings <-> '[3,1,2]' AS distance
FROM test_embeddings ORDER BY embeddings <-> '[3,1,2]’ limit 2;

product_id | embeddings |     distance
------------+------------+------------------
          1 | [1,2,3]    | 2.449489742783178
          2 | [2,3,4]    |                 3
(2 rows)
```

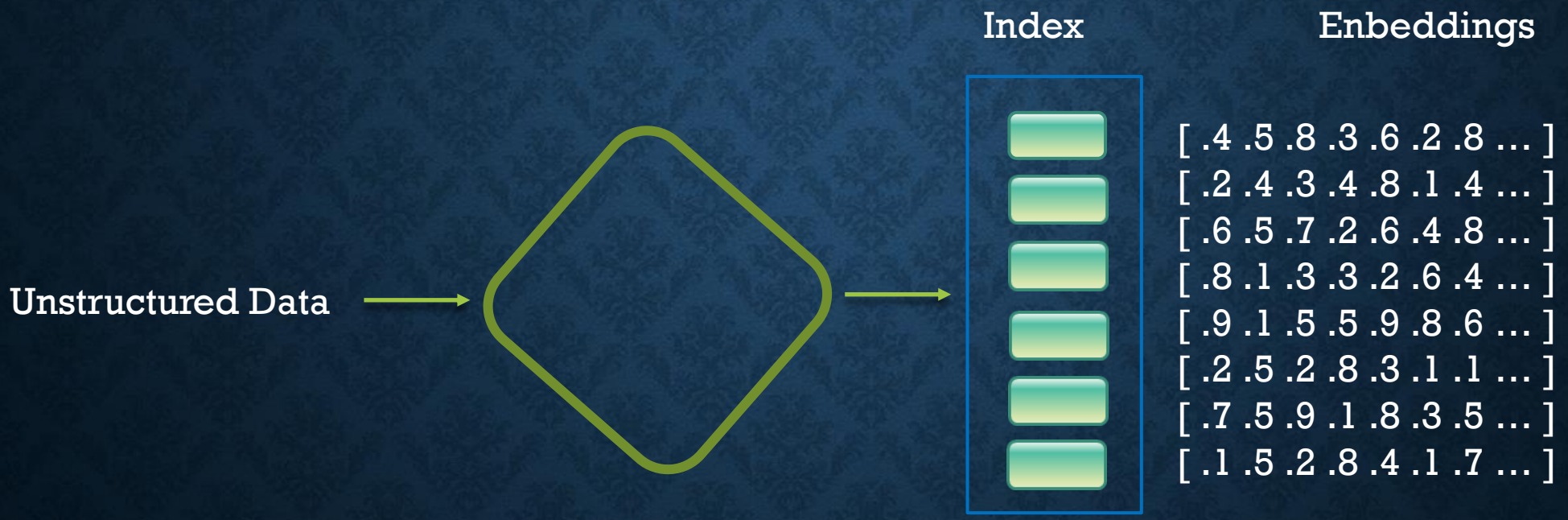
INDEXING PGVECTOR – LISTS & PROBES

2D VECTOR EMBEDDING EXAMPLE



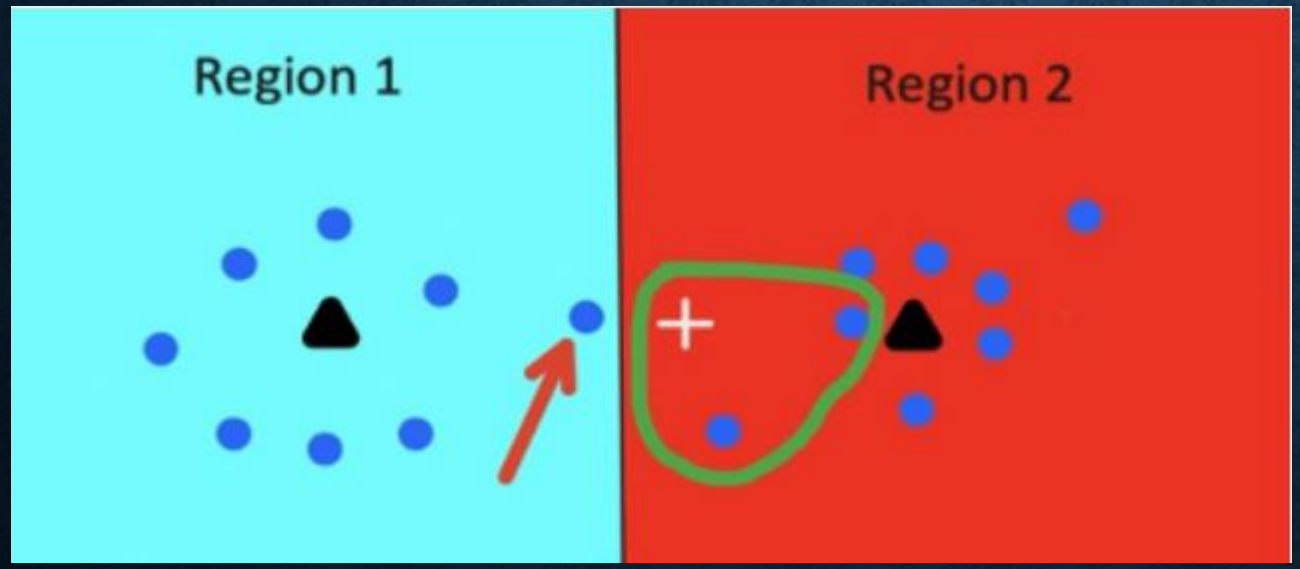
$$d = \sqrt{[(x2 - x1)^2 + (y2 - y1)^2]}$$

VECTOR INDEXING



IVFFLAT INDEX BUILDING PARAMETERS

- `lists`
  - Number of "buckets/regions/clusters" for organizing vectors
  - Tradeoff between number of vectors in bucket and relevancy



BEST PRACTICES FOR BUILDING IVFFLAT INDEXES

- Choose value of lists to maximize recall but minimize effort of search
  - < 1MM vectors: # vectors / 1000
  - > 1MM vectors: √(# vectors)
- May be necessary to rebuild when adding/modifying vectors in index
- Use parallelism to accelerate build times
- Increase maintenance_work_mem for faster index creation

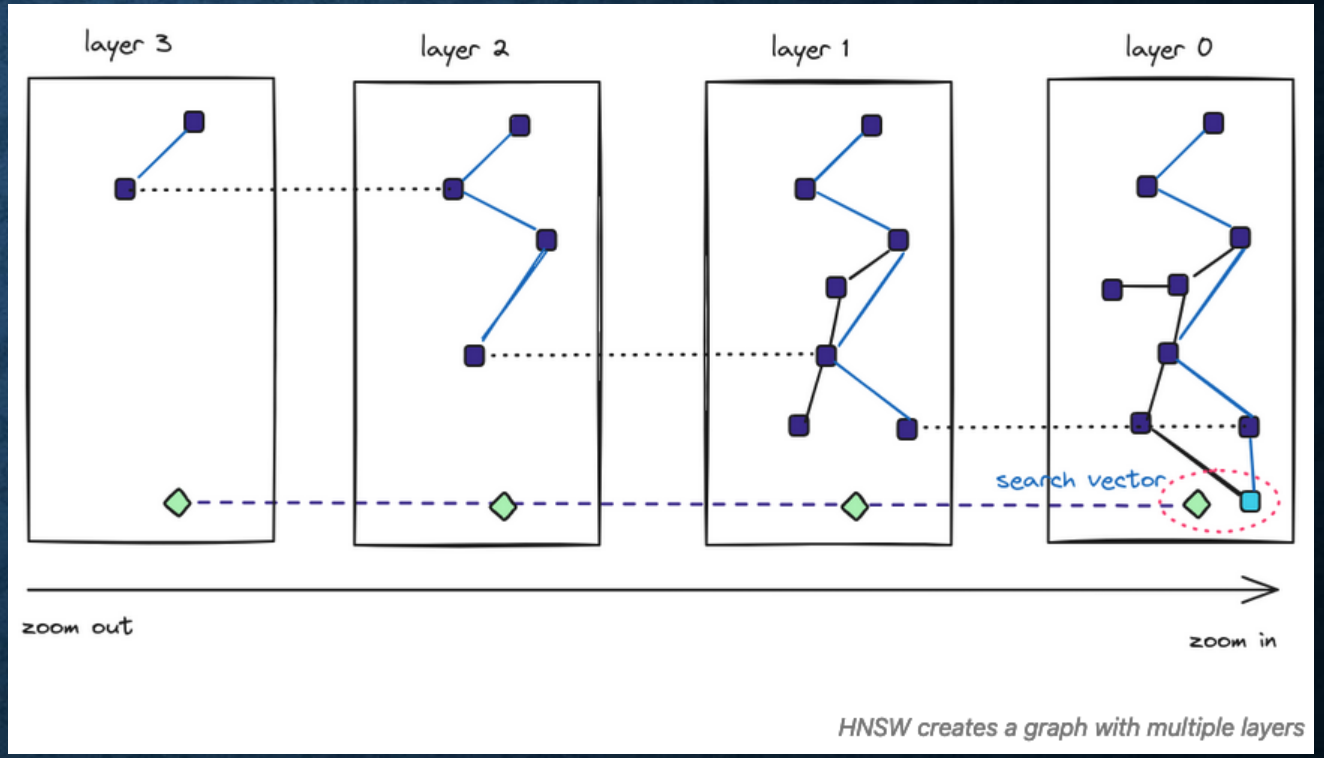
HNSW (Hierarchical Navigable Small Worlds)

- Index can be created on empty table
- Longer time to build and requires more memory, but better recall
- vector - up to 2,000 dimensions

halfvec - up to 4,000 dimensions (added in 0.7.0)

bit - up to 64,000 dimensions (added in 0.7.0)

sparsevec - up to 1,000 non-zero elements (added in 0.7.0)



Source: https://tembo.io/blog/vector-indexes-in-pgvector/#hnsw

https://jkatz05.com/post/postgres/pgvector-hnsw-performance/

BEST PRACTICES FOR HNSW INDEXES

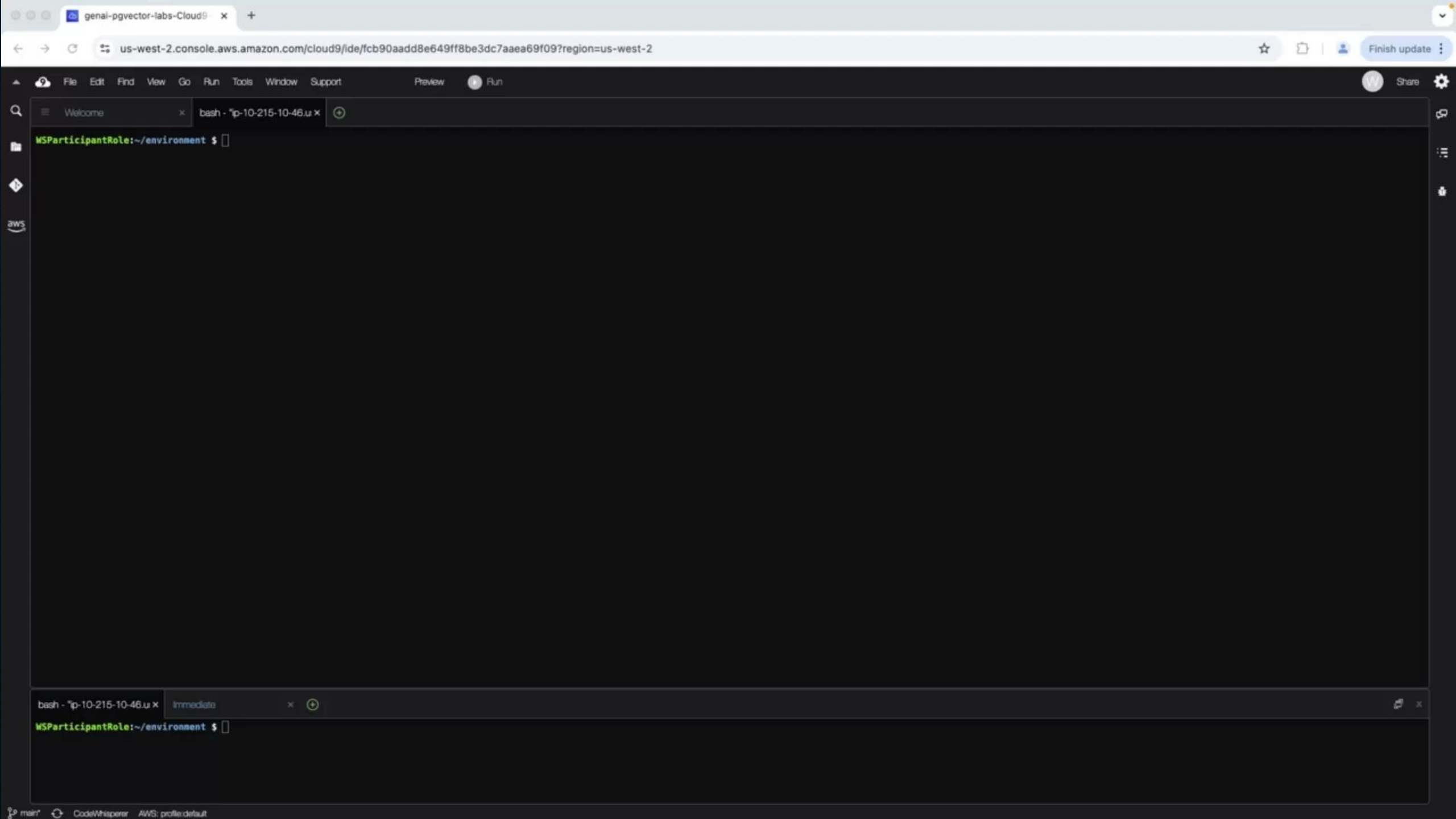
- Building HNSW indexes
- Default values (`M=16,ef_construction=64`) usually work
- (pgvector 0.5.1) Start with empty index and use concurrent writes to accelerate builds
  - `INSERT` or `COPY`

- Performance strategies
- Index building has biggest impact on performance/recall
- Increasing `hnsw.ef_search` increases recall, decreases performance

WHICH INDEX DO I CHOOSE?

1. If you care more about index size, then choose IVFFlat
2. If you care more about index build time, then select IVFFlat
3. If you care more about speed, then choose HNSW
4. If you expect vectors to be added or modified, then select HNSW

DEMO: RECOMMENDATION SEARCH USING PGVECTOR


genai-pgvector-labs-Cloud9
us-west-2.console.aws.amazon.com/cloud9/ide/fcb90aadd8e649ff8be3dc7aaea69f09?region=us-west-2
Finish update
File Edit Find View Go Run Tools Window Support
Preview
Run
Share
Welcome
bash - "ip-10-215-10-46.u
WSParticipantRole:~/environment $
bash - "ip-10-215-10-46.u
Immediate
WSParticipantRole:~/environment $
main*
CodeWhisperer
AWS: profile:default

Q&A

Thank you!

Vivek Singh

Principal Database Specialist at AWS

Kiran Singh

AWS Solutions Architect | AWS Speaker | Driving Innovation and Change in the ISV and SI Sectors