InterSystems IRIS: Achieving Speed and Usability in an Integrated Vector Database

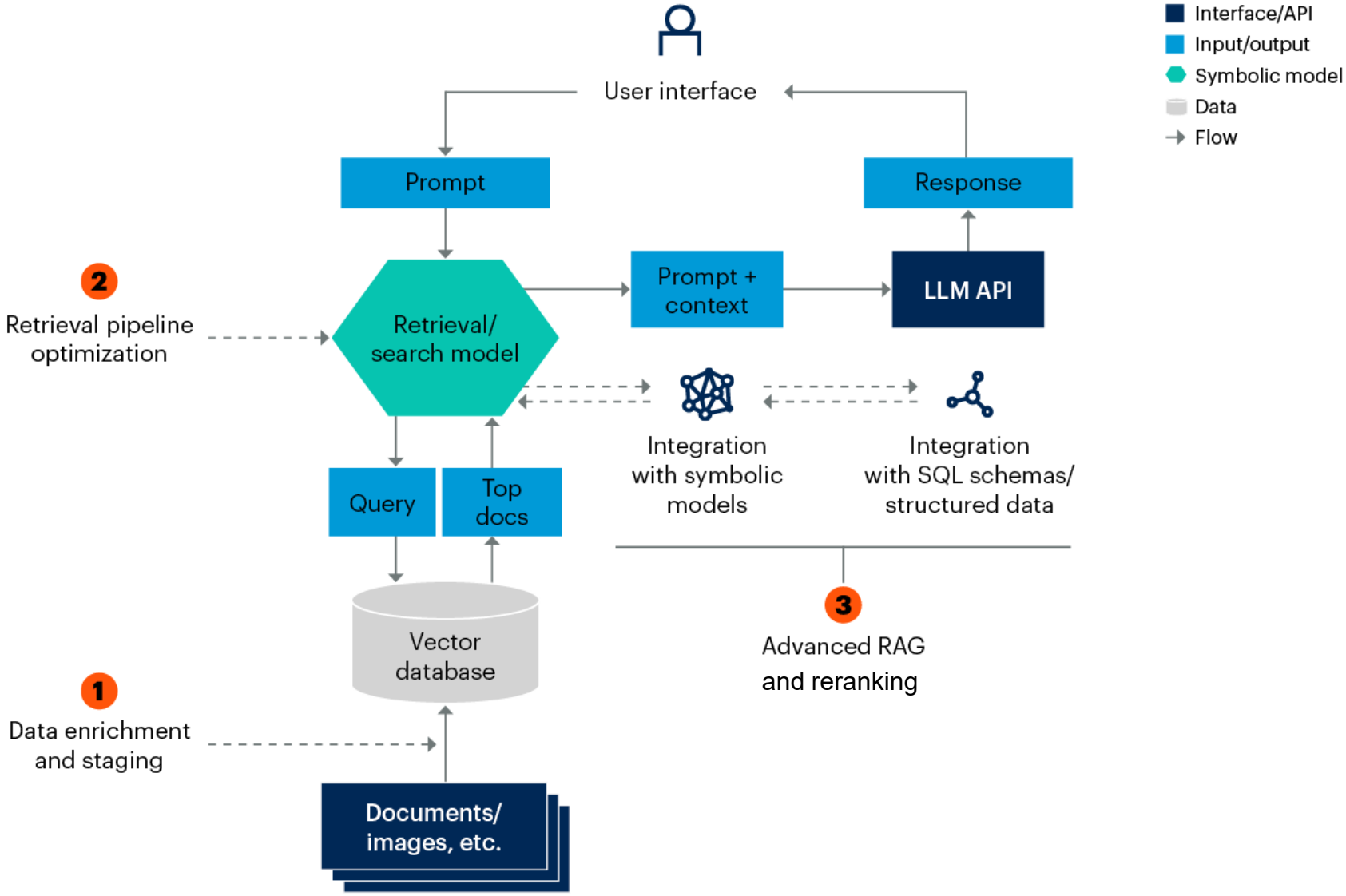
David Van De Griek, Boya Song, Philip Miloslavsky, Yuchen Liu, Yiwen Huang, Mark Hanson, Jeff Fried, Dmitriy Bochkov

NEDB 2025

Overview

- New design for Integrated Vector DB
- Focused on RAG applications
- Based on:
  - multimodel core (“common data plane”)
  - vector data type and vector operations
  - compact storage model
  - vector-aware SQL optimizer
  - embedded Python
- Commercially in use

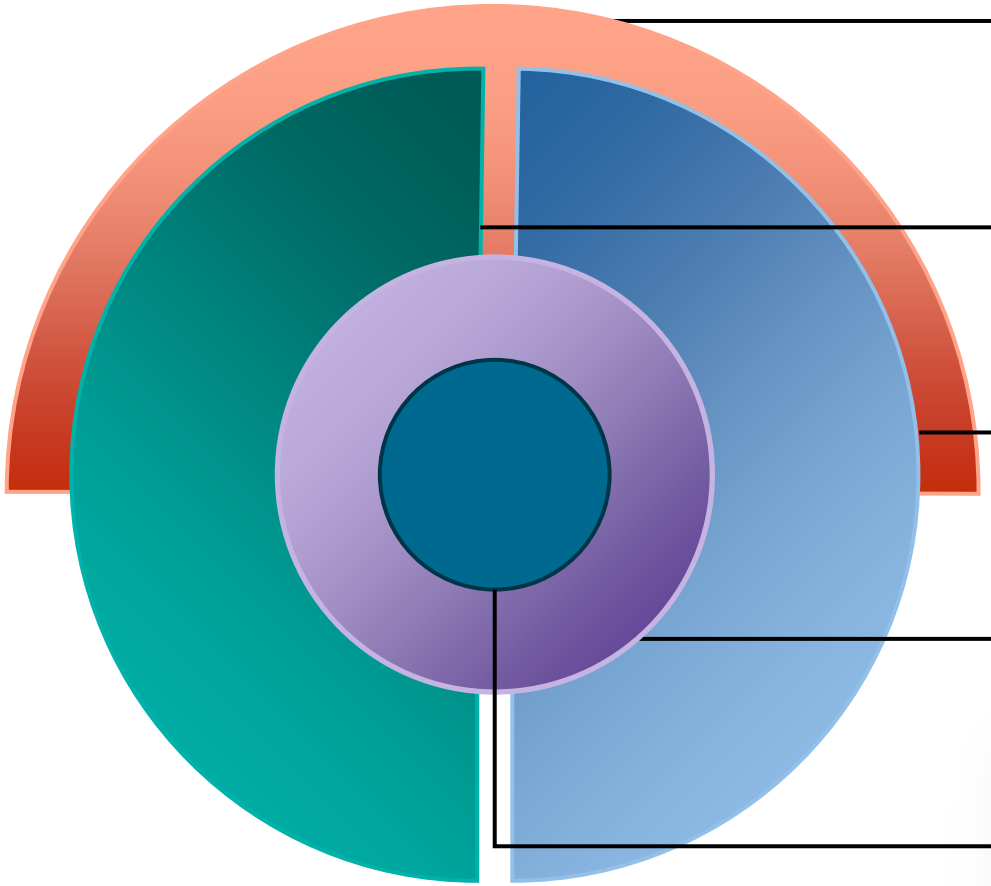
Vector Database as used in RAG applications



Specialized and Integrated Vector Databases

Criteria	Specialized Vector Databases	Integrated Vector Databases
Examples	Pinecone, Milvus, Weviate	PPASE, PostgreSQL+pgvector, ElasticSearch **InterSystems IRIS**
Ease of Use	Designed specifically for vector data	Integrated into existing databases, leveraging familiar interfaces and tools
Footprint	Typically requires additional infrastructure and resources	Utilizes existing database infrastructure, reducing the need for additional resources
Performance	Optimized for high-dimensional vector searches, often with advanced indexing techniques	Performance can vary based on the underlying database but benefits from integrated indexing

InterSystems IRIS Architecture Layers



Smart Data Fabric
Single ecosystem to solve complex data problems

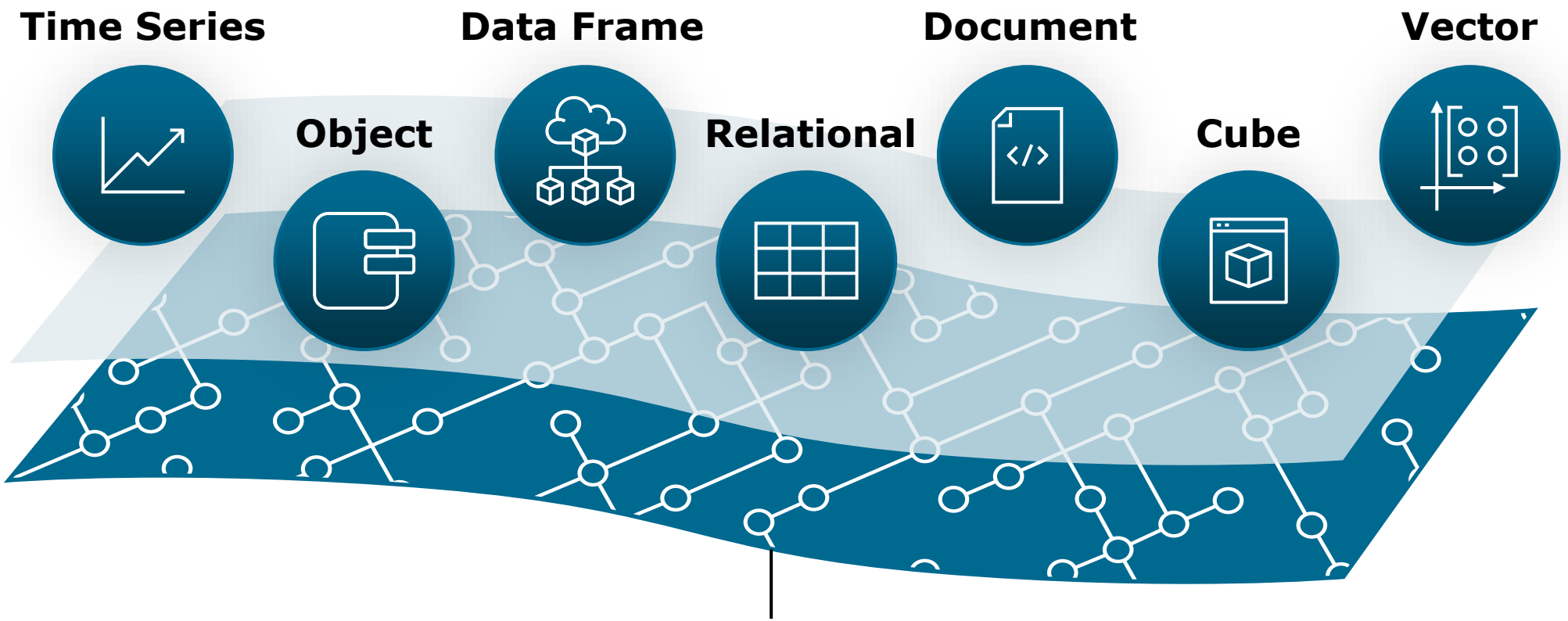
Analytics & AI
Run close to the data, using familiar tools

Interoperability
Low-code integration, workflow & API mgmt

Horizontal Scale-out
Distributed, coherent cache at massive scale

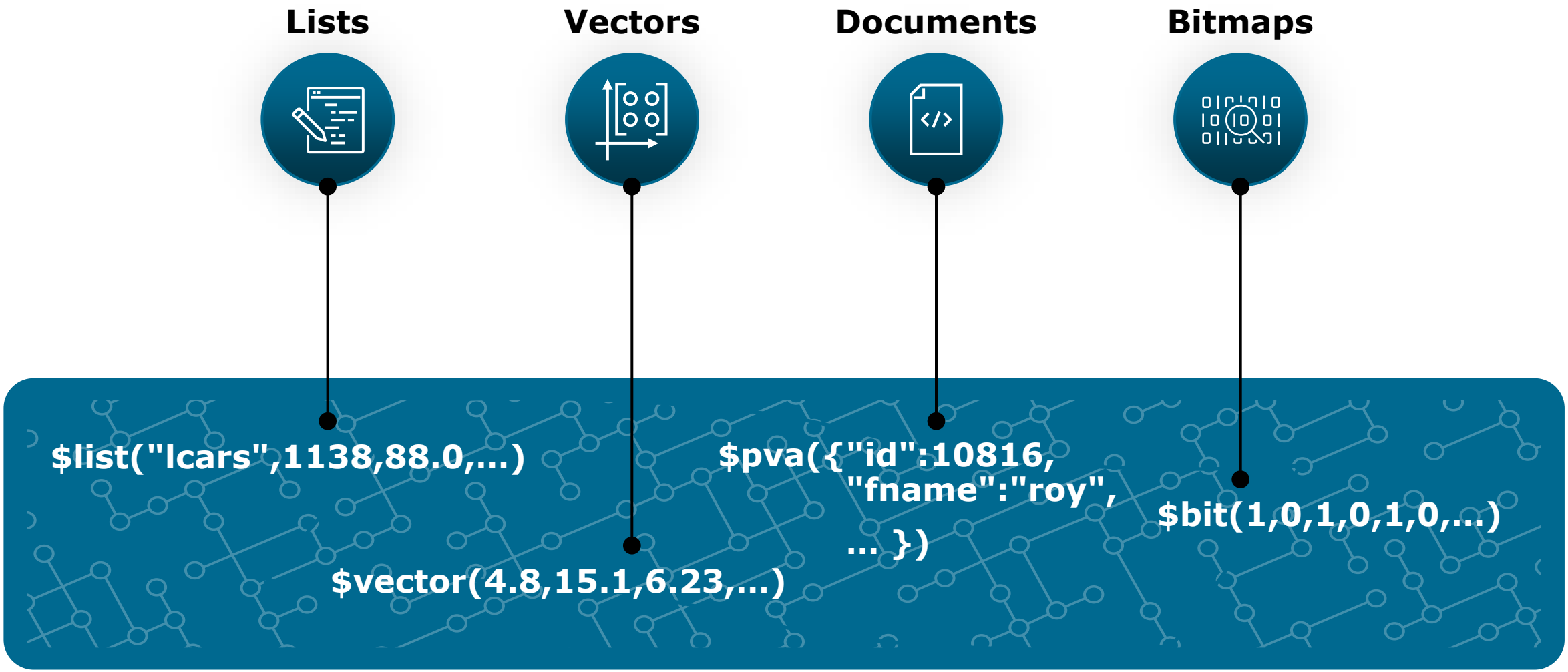
Common Data Plane
High performance, multi-model, multi-lingual

The Common Data Plane



^global(<key1>,<key2>,...) = $encoding(<val1>,<val2>,...)

Fast, Flexible Data Encodings



Projections

Document

```
{
"location": "ICU5",
"collector_name": "Smartlinx5",
"sensor_name": "BP3",
"bed_id": 8605,
"readings": [
  { "start_time": "2024-07-04 10:12:03.642",
    "interval": 3.333,
    "values": [7200, 7300, 7700, 8500, 9100, ...
] }
  , ... ]
}
```

Location	Sensor	Date	Time	Value
ICU5	BP3	2024-07-04	10:12:03.642	7200
ICU5	BP3	2024-07-04	10:12:06.975	7300
ICU5	BP3	2024-07-04	10:12:10.308	7700
ICU5	BP3	2024-07-04	10:12:13.641	8500
ICU5	BP3	2024-07-04	10:12:16.974	9100
...	...	...	...	...

Relational

PUT

GET

SELECT

INSERT

```
^wave(202407, "ICU5", 123) = $list("Smartlinx5", "BP3", 8605)
^wave(202407, "ICU5", 123, 1, "ts") = $list("2024-07-04 10:12:03.642", 3.333)
^wave(202407, "ICU5", 123, 1, "v") = $vector(7200, 7300, 7700, 8500, 9100, ...)
```

Design Goal

Unified, versatile data engine that supports vector fields and vector indices (e.g. HNSW index)

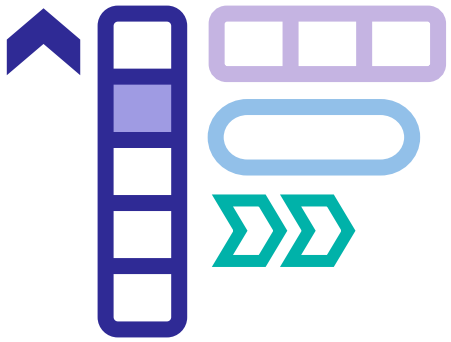
- Unified storage for data and index vector data, columnar data and regular data
  - Minimize data duplication
  - Index/field size not limited by memory size

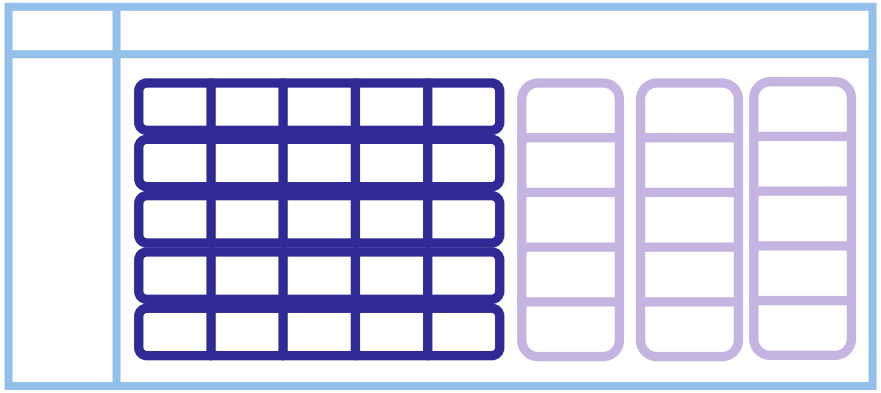
- Unified SQL engine with great query processing capabilities, including
    - Transactions
    - Full SQL Support
    - Filtered Vector Search
    - Vector Range Search
    - Vector Range Join
    - Vector with Fulltext Search
    - Semantic Join

Vector Storage Model evolved from columnar use cases

Storage model for SQL based on native $vector data type to deliver key analytical querying facilities needed for next-generation Data Warehouses, Lakes and Lakehouses

- Aligns physical table layout with typical analytical access patterns
- $vector language feature designed to support **translytical workloads**
- **Order of magnitude speedup** for analytical queries thanks to SIMD use and vectorized execution
- **Schema flexibility** - mixing row & column storage - is a key InterSystems IRIS differentiator
- **Indexing flexibility** – can use a columnar index on row-based storage, etc.





Storage design

Vector fields stored as a collection of chunked vectors, 64K values per chunk.

Metadata (Columnar Index Map and Columnar Data Map) support fast sorting and SIMD operations.

Special %Vector type: %Embedding

Flexible design, including **ability to accommodate VERY long vectors**

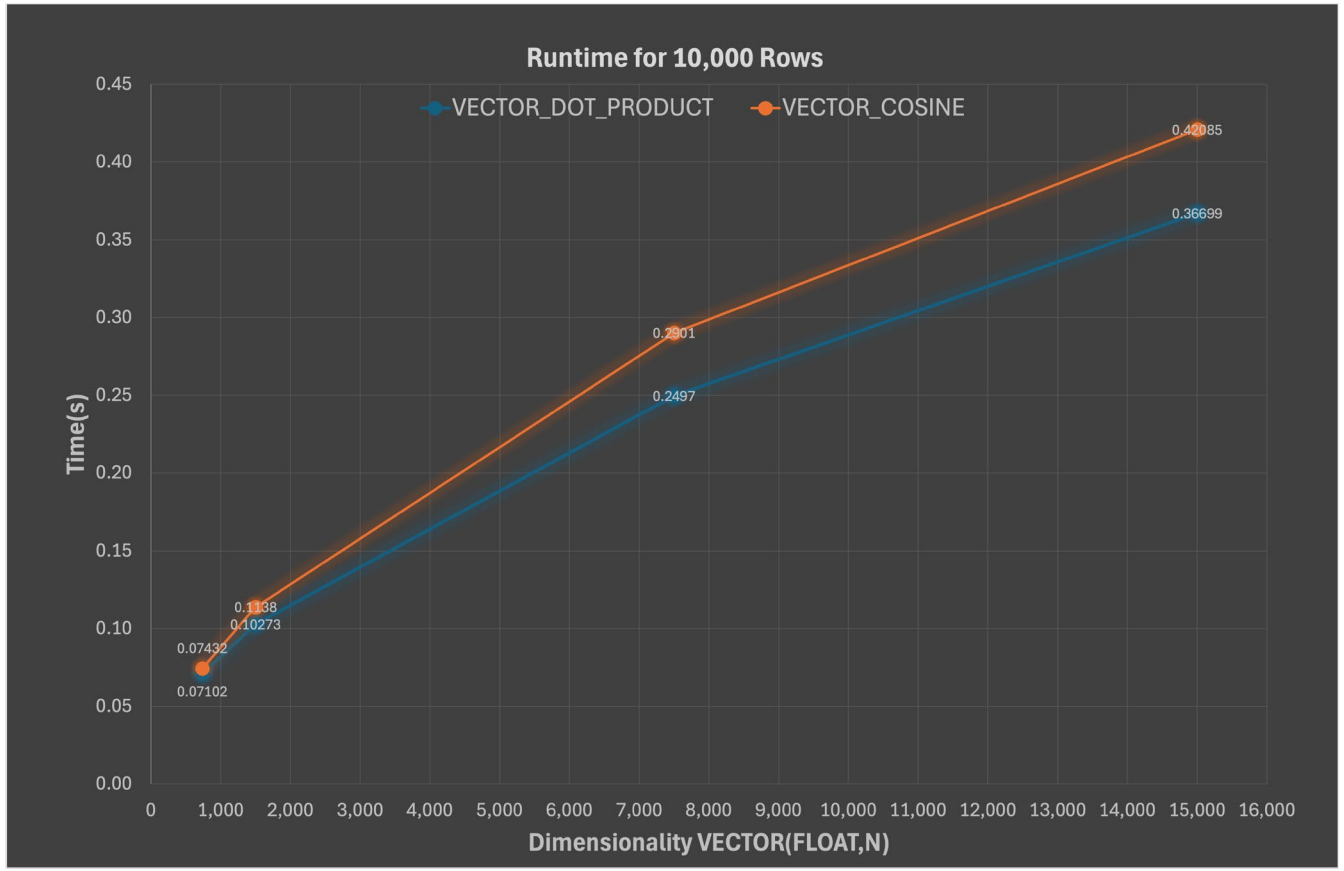
- Store long vectors in their own globals for performance and footprint
- Provision for managing space/precision tradeoffs
  - $vector can be integer, decimal FP; supports sparse encodings and different storage size magnitude for both sparse and dense encodings
- new DataDefinitionLocation property

Fast vector operations

- **Numeric Operations (scalar and vector-wise)**
  - $VECTOROP("+" | "-" | "/" | ... | "cosine" | "dot-product" , vector, vector | scalar, bitmap) returns vector
- **String Operations (scalar and vector-wise)**
  - $VECTOROP("_" | "lower" | "substring" | ... , vector, vector | scalar, bitmap) returns vector
- **Filter Operations (scalar and vector-wise)**
  - $VECTOROP("=" | ">" | "<" | ... | "defined" | "undefined" , vector, vector | scalar, bitmap) returns bitmap
- **Aggregate Operations**
  - $VECTOROP(“count” | “max” | “min” | “sum”, vector, bitmap) returns scalar
- **Grouping Operations**
  - $VECTOROP("group", “count” | “max” | “min” | “sum”, vector, bitmap, list) modifies list
- **Miscellaneous Operations**
  - $VECTOROP("convert", vector)
  - $VECTOROP("mask", vector, scalar)
  - $VECTOROP("positions", vector, bitmap)
  - $VECTOROP("bytesize", vector)
  - ...

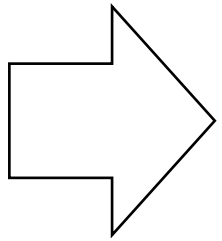
Vector Dimensionality and Performance

- We can support vector storage and operation on high-dimensional vectors thanks to the fast $vectorop
- Runtime of vector operation increases sub-linearly



Query processing implementation: Unified SQL Engine

- Unified Storage Model
  - comparable access cost
    - for any storage type
    - for any index type
- Universal Query Optimizer
  - pre-optimizer query rewrite
  - awareness of vector algorithms
  - multi-index plans
- Adaptive Parallel Execution
  - data format agnostic



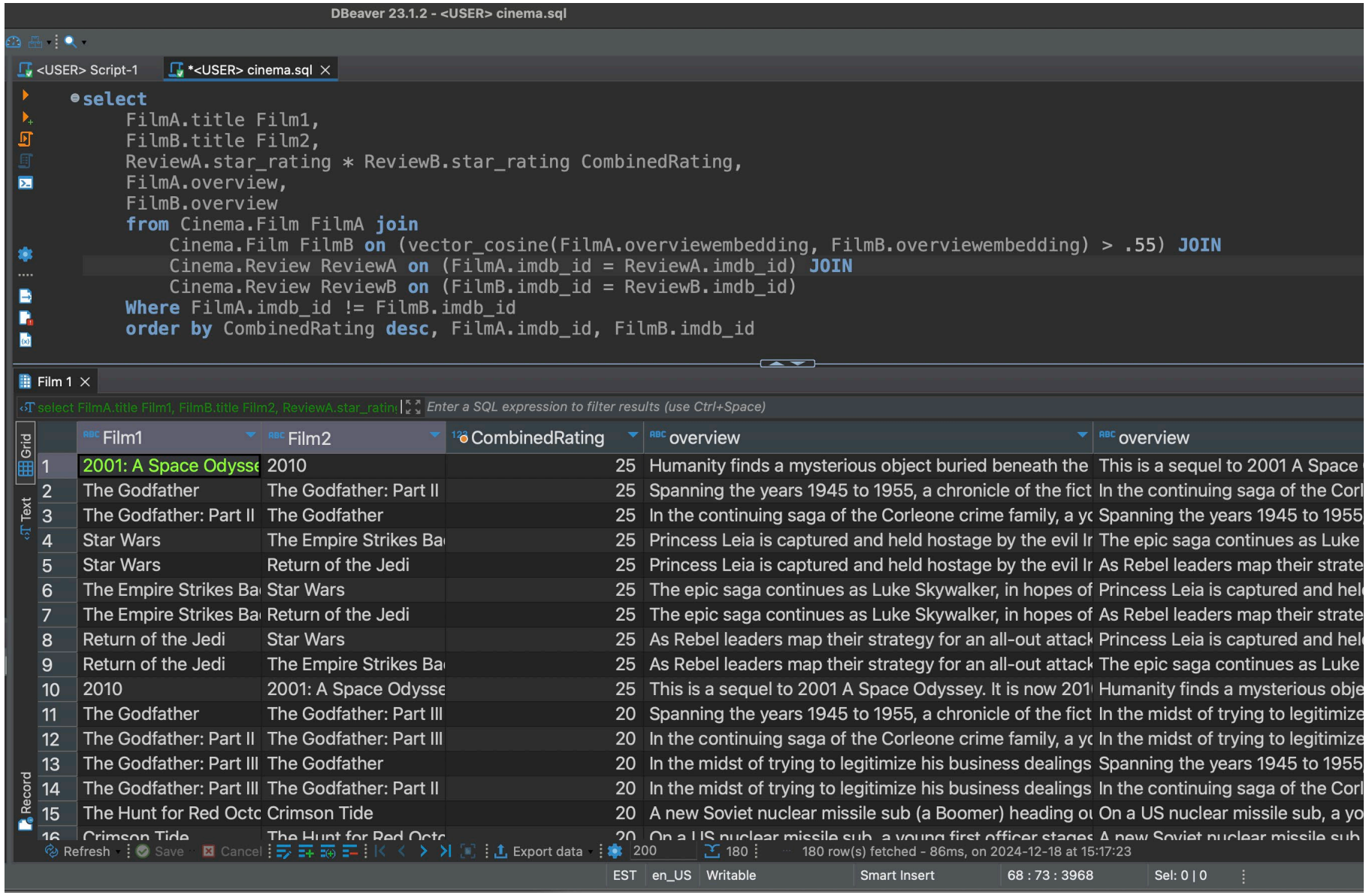
- Efficient integration of vector/embedding data type
- Example: Semantic Join
  - join with a similarity condition on word embeddings [1]
  - tolerates misspellings and different formats to deliver more join results

Reference: Dong Y, Xiao C, Nozawa T, Enomoto M, and Oyamada M **DeepJoin: joinable table discovery with pre-trained language models** Proc. VLDB Endow. Digital Library

Semantic Join in SQL

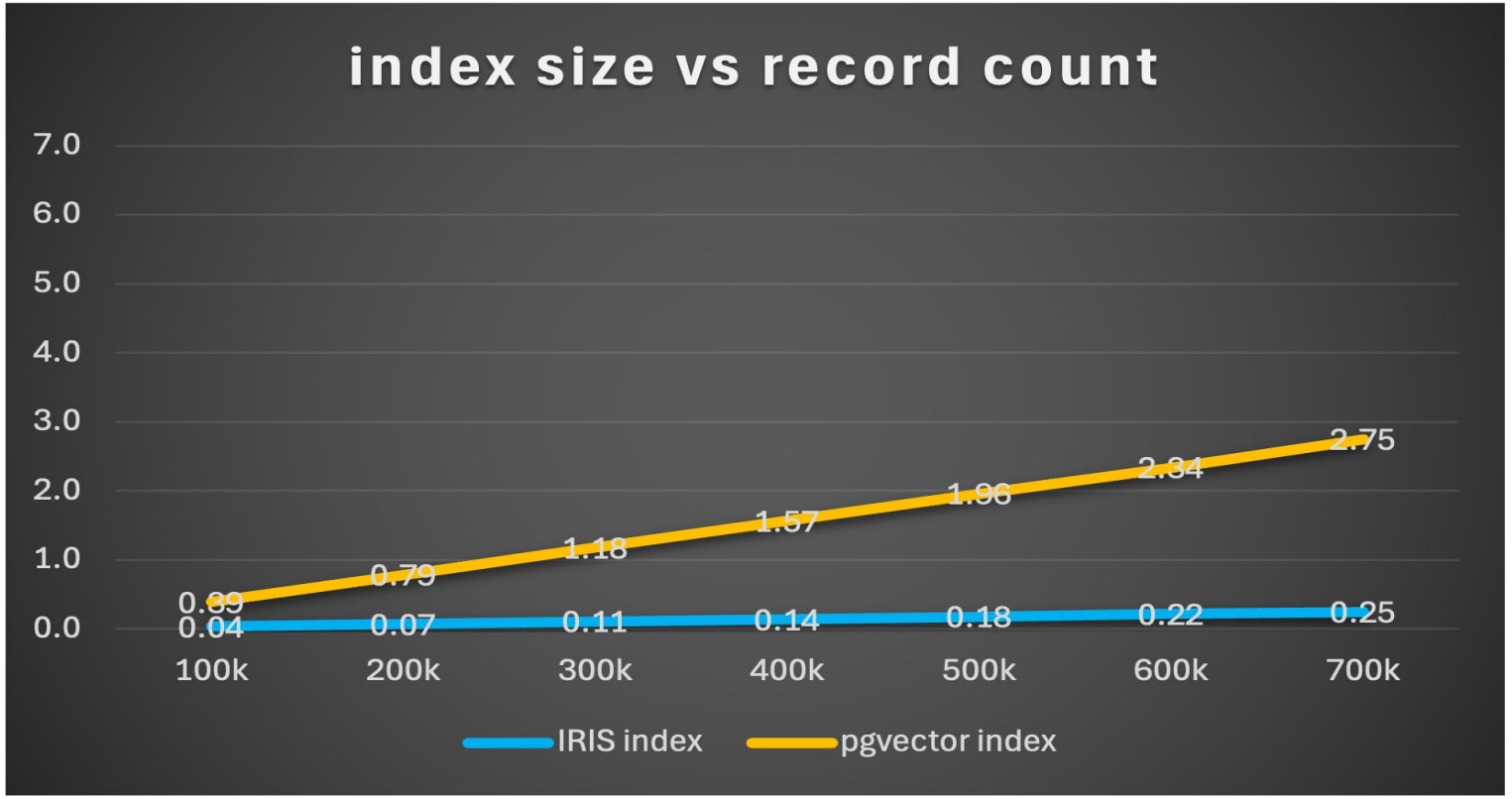
```
select
    FilmA.title Film1,
    FilmB.title Film2,
    ReviewA.star_rating * ReviewB.star_rating CombinedRating
from  Cinema.Film FilmA
join  Cinema.Film FilmB
    on (vector_cosine(FilmA.overview_embedding, FilmB.overview_embedding)>.55)
join  Cinema.Review ReviewA on (FilmA.imdb_id=ReviewA.imdb_id)
join  Cinema.Review ReviewB on (FilmB.imdb_id=ReviewB.imdb_id)
where FilmA.imdb_id<FilmB.imdb_id
    and FilmA.release_year>1950
    and FilmB.length>60
order by CombinedRating desc, FilmA.imdb_id, FilmB.imdb_id
```

Semantic Join Result



Query processing implementation: HNSW index

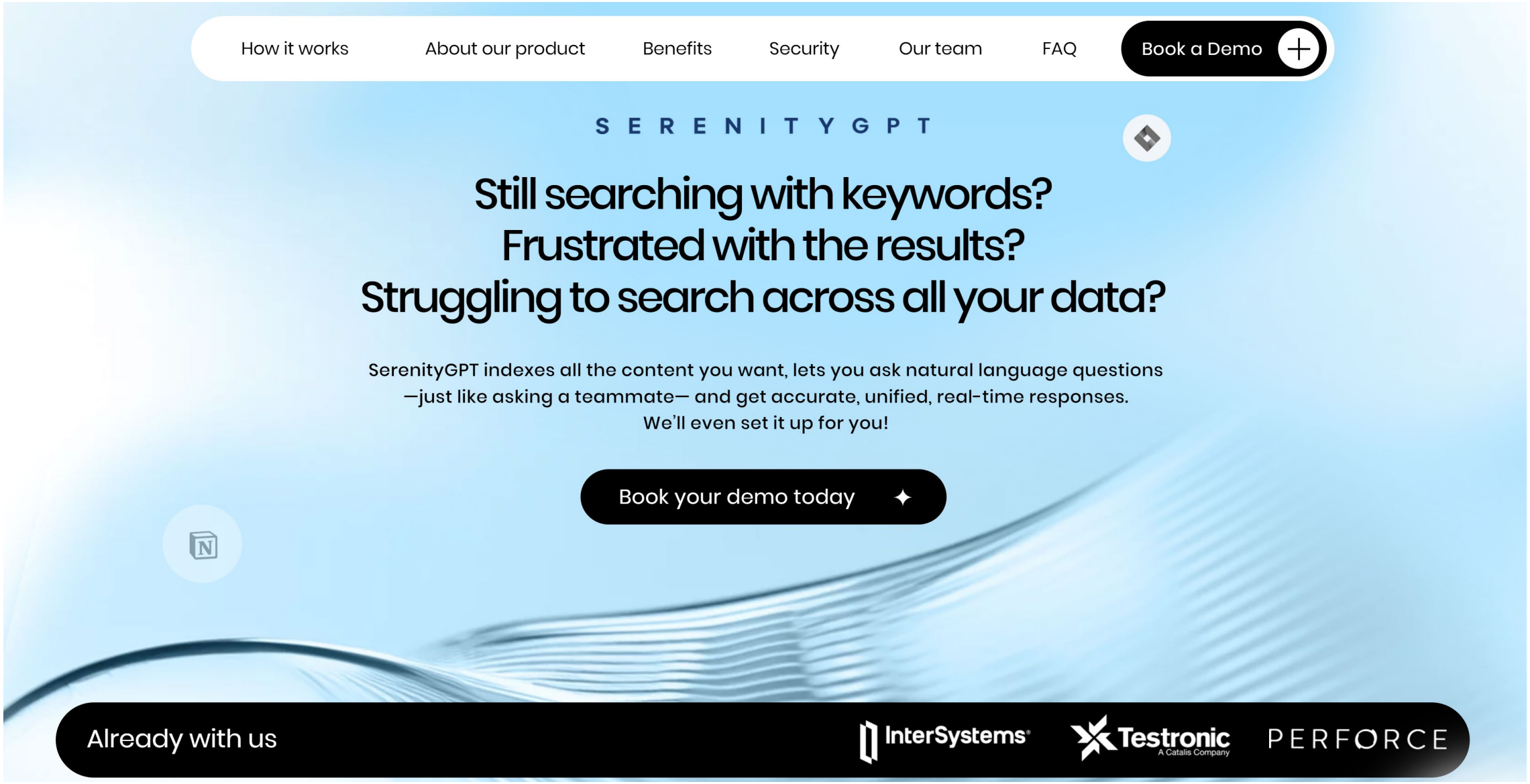
- Implemented HNSW index according to Malkov 2016
- No need to store the vectors in the index as the SQL engine can access the original vector field



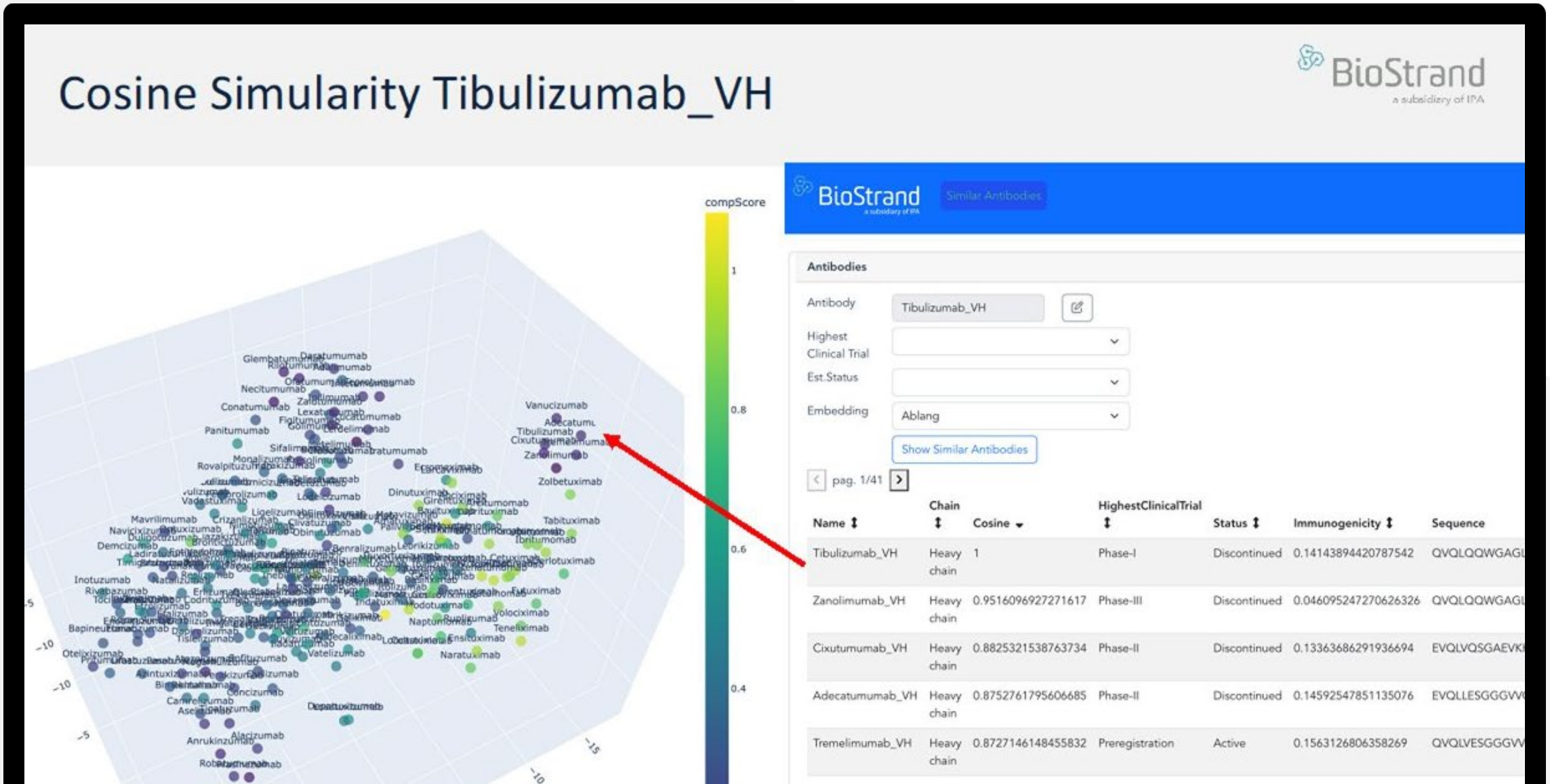
Reference: Yury Malkov, Dmitry A. Yashunin, **Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs**
IEEE TPAMI 2016 arXiv link

- Challenge: needs to be compatible with parallel INSERT

SerenityGPT – built on InterSystems IRIS & Vector Search



Biostrand: Complex Data Analysis With IRIS Vector Search



Contact:

Jeff.Fried@InterSystems.com

@jefffried

Thank you

InterSystems®
Creative data technology