Lance: Open Source Foundations for A Lakehouse for Multi-modal AI

DataCouncil 2024

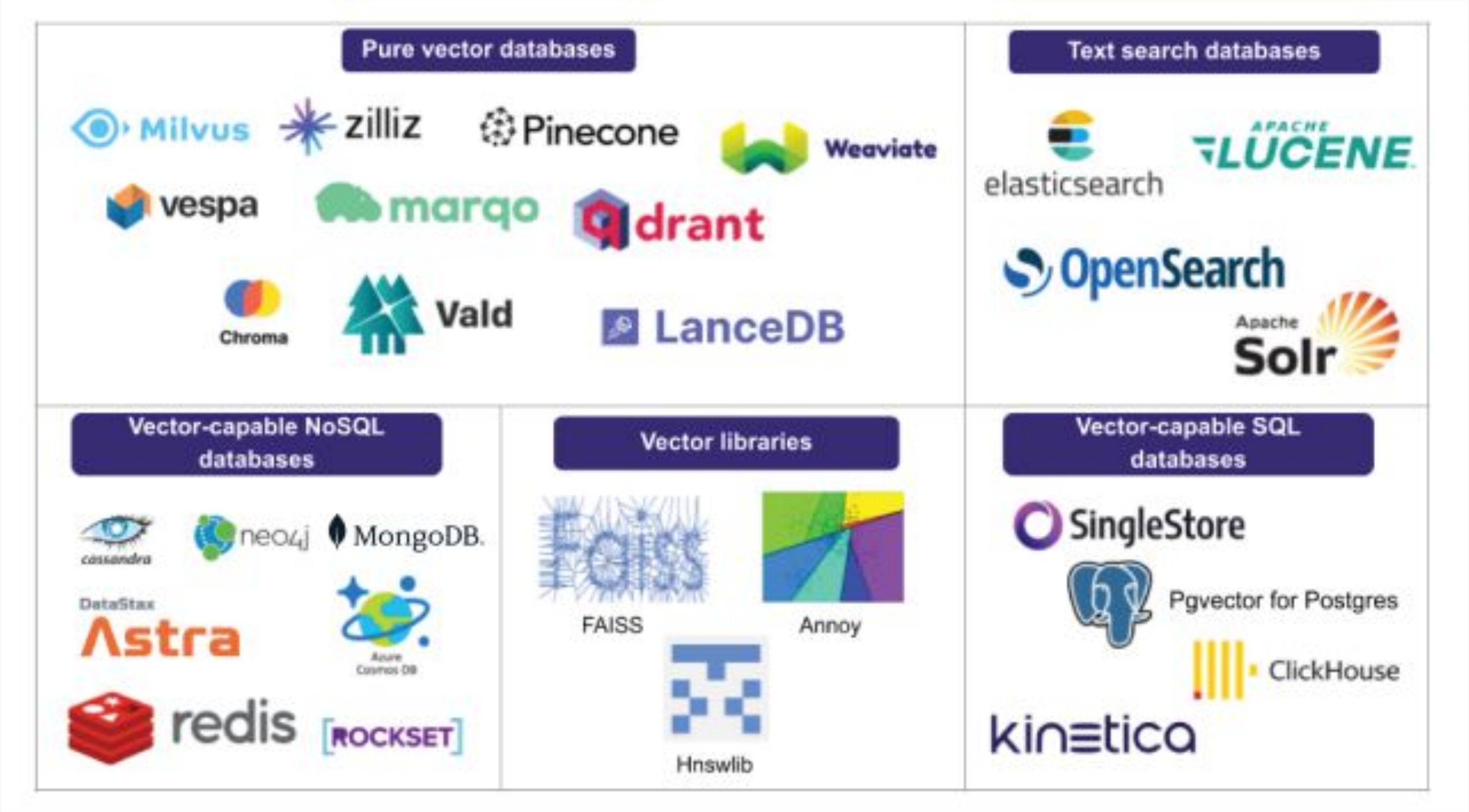
Chang She

- LanceDB co-founder
- Pandas co-author
- VP Eng @ TubiTV (MLOps / Experimentation)
- Twitter/Github: @changhiskhan

Vector databases won't exist in 3 years

ANN indices are a commodity

30+ Pure vdbs + libraries + databases adding ANN indices



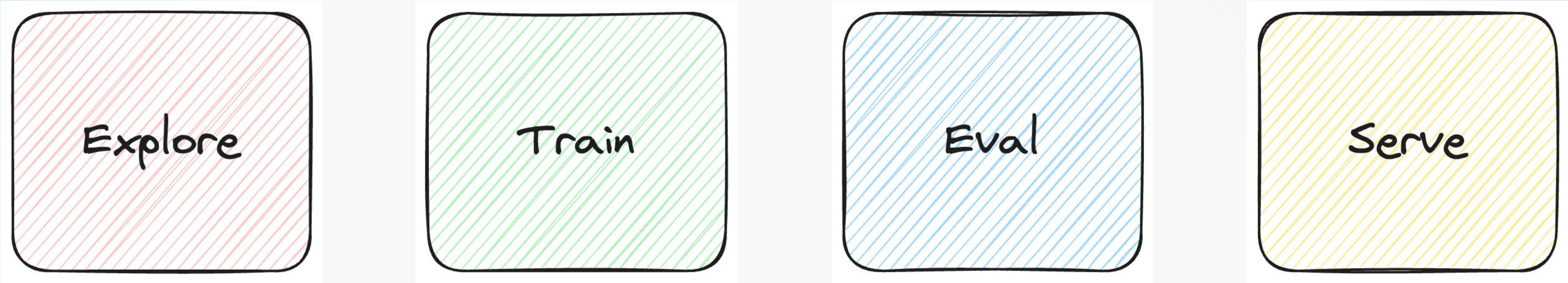
Is AI data infra solved?

Hopefully not or this will be a very short talk

- Scale / performance / cost ⇒ O(100m ~ B+) is still complicated and expensive
- Advanced retrieval is not yet solved
- Self-improving retrieval still doesn't exist yet
- **Working with the data itself, especially multimodal data, is hard**

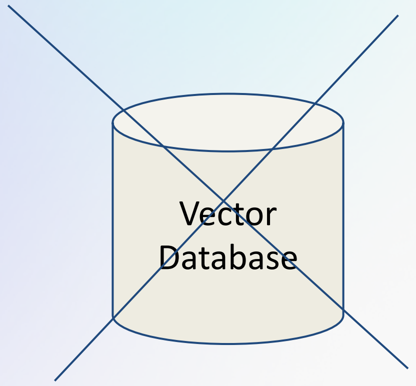
We need more than retrieval

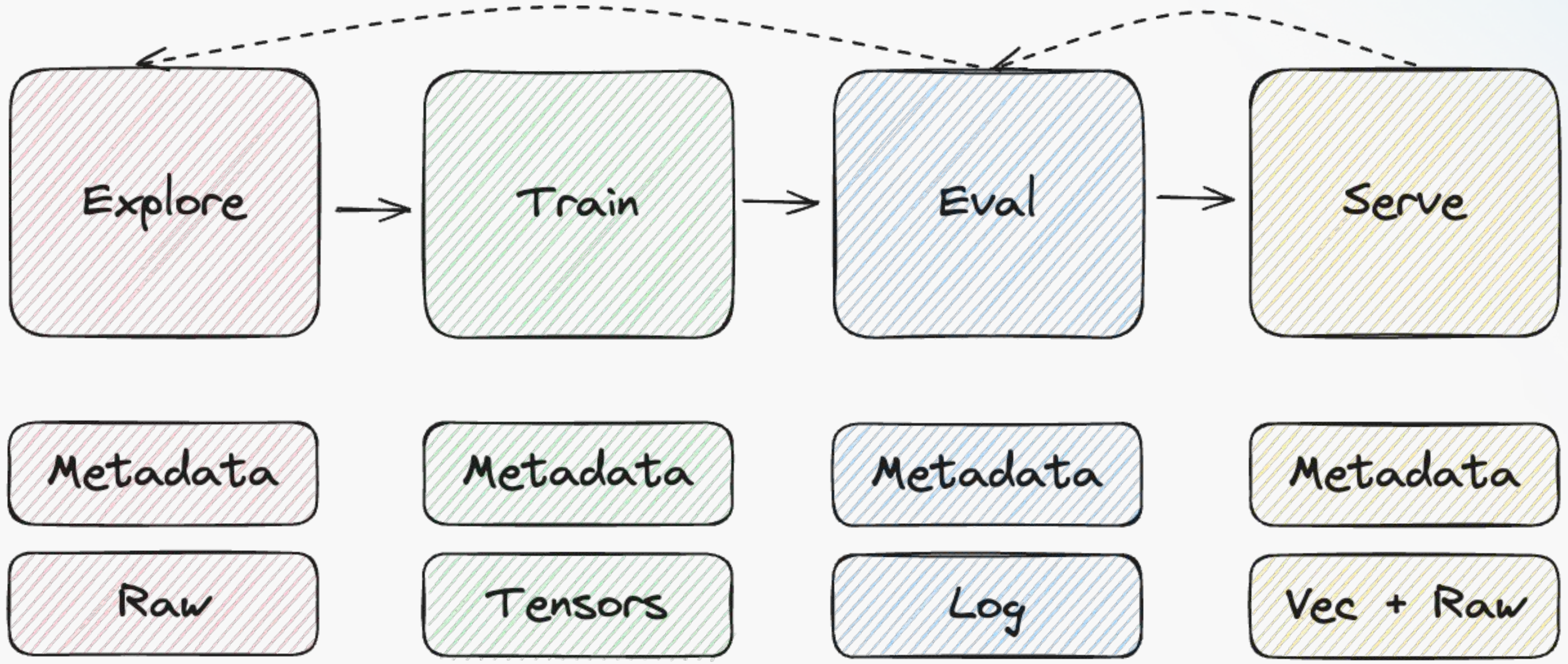
Especially for multi-modal, we need way more than just search



We need more than retrieval

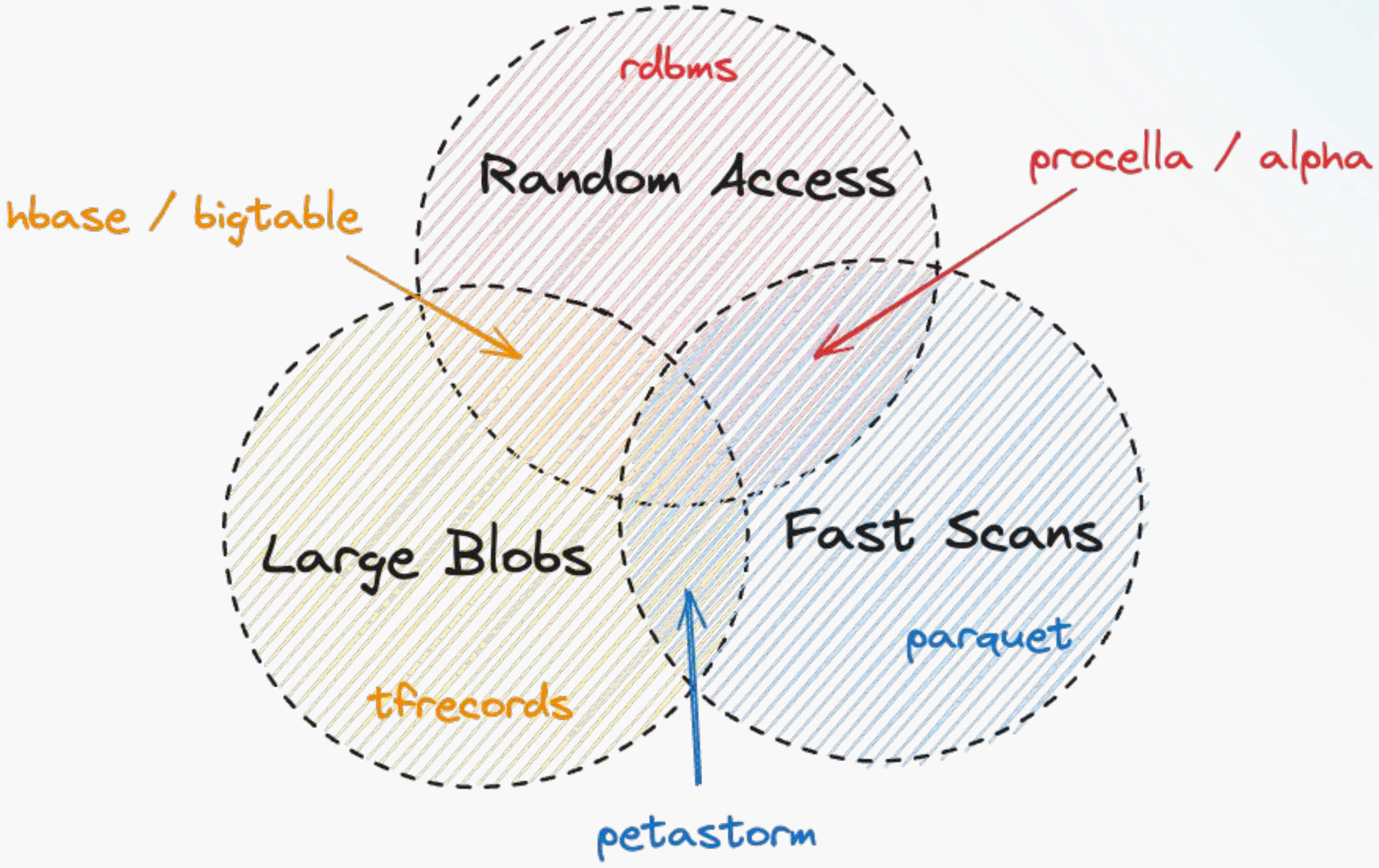
Especially for multi-modal, we need way more than just search





Vector search should be natively accessible anywhere within this workflow

What's the problem?



CAP Theorem for AI Data

What's the result?

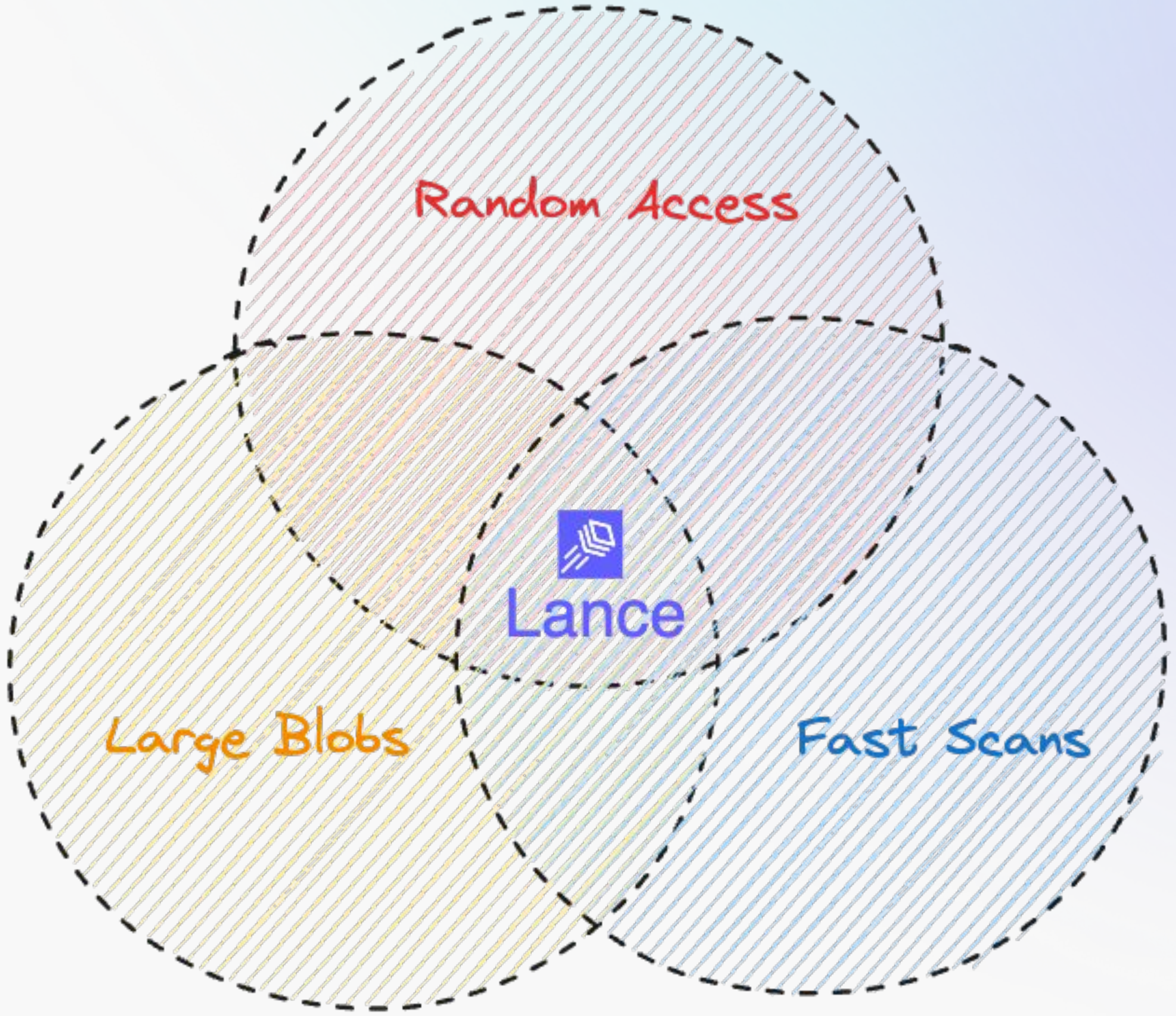
High cost, complex stack, bad performance

- Having multiple copies in different formats is expensive and slow
- Operating multiple systems is complicated
- Keeping GPU well-fed at scale means wasted GPU resources

What's the solution

Change the foundational data layer

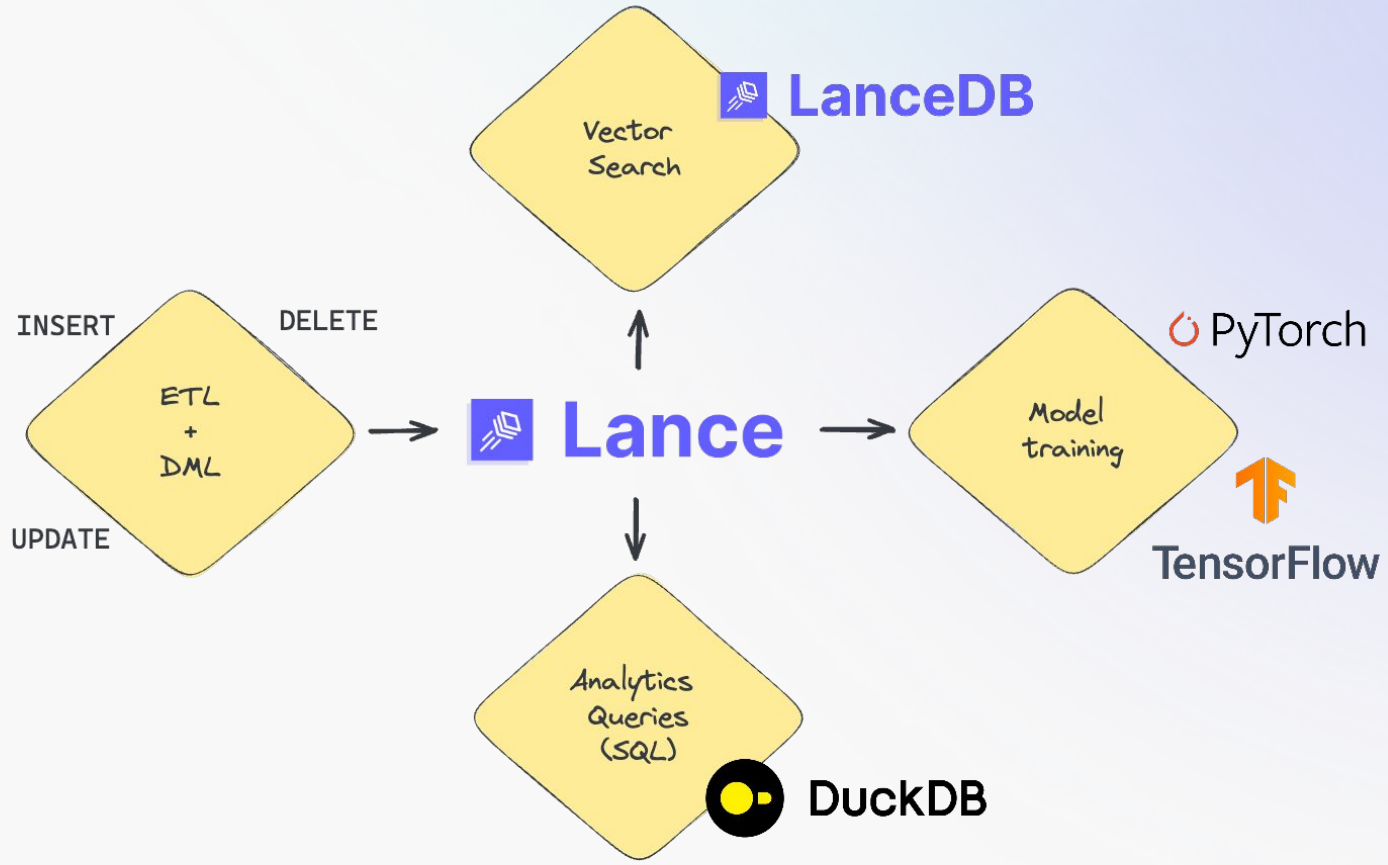
- Single source of truth for offline and online
- No wasted copies; faster ML experimentation
- SQL / DataFrames >>> custom scripts
- EDA, training, etc can all share the same data



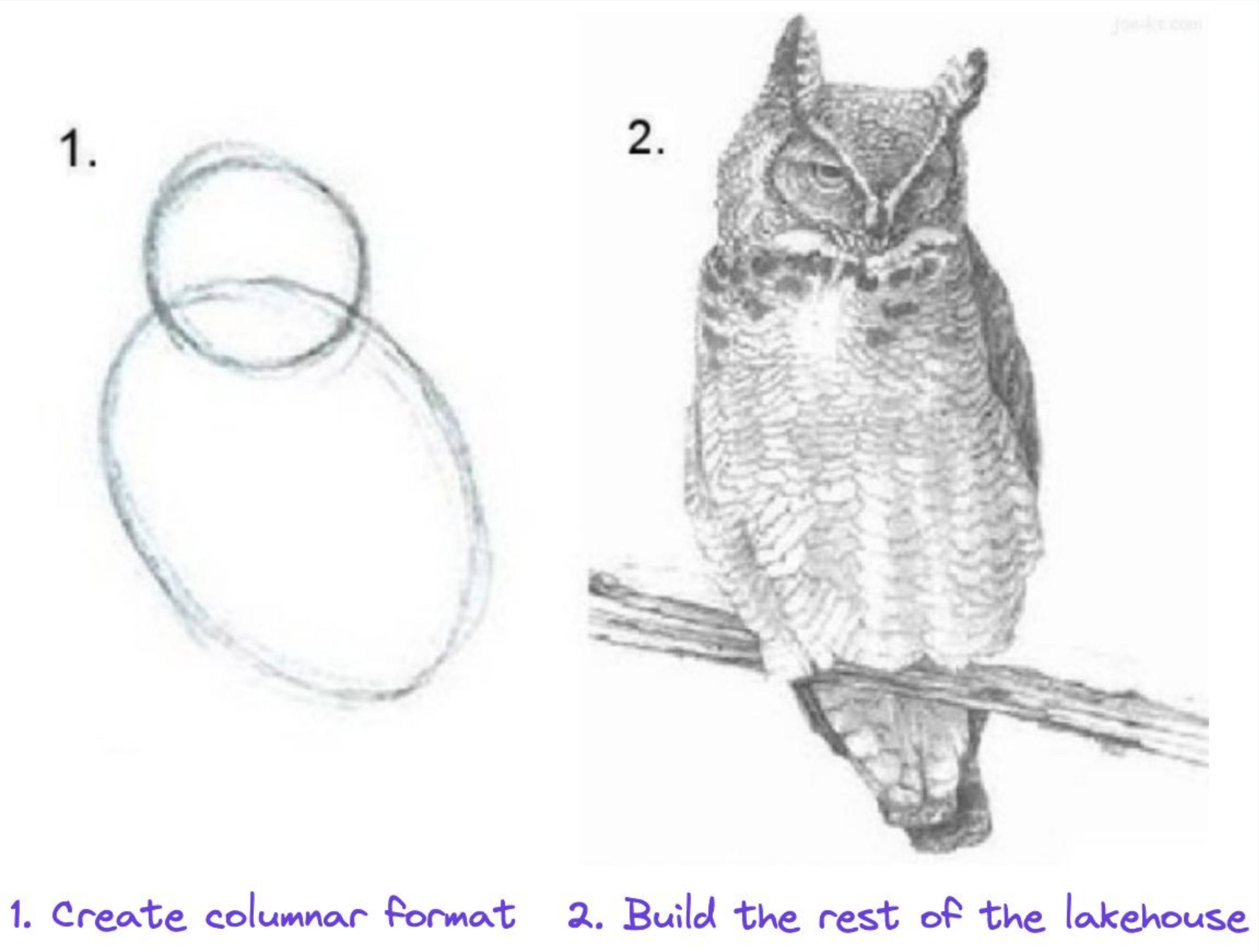
Lance format for AI

File format, table format, and indexing subsystem

- Columnar storage with
  - Different layout ⇒ fast scans & random access
  - IO exec optimized for large-blobs
  - (roadmap) advanced encodings
- Fast scans + random access ⇒ training, EDA
- Random access + indexing ⇒ ANN, filter / sample
- Table format ⇒ schema evolution, versioning, reproducibility

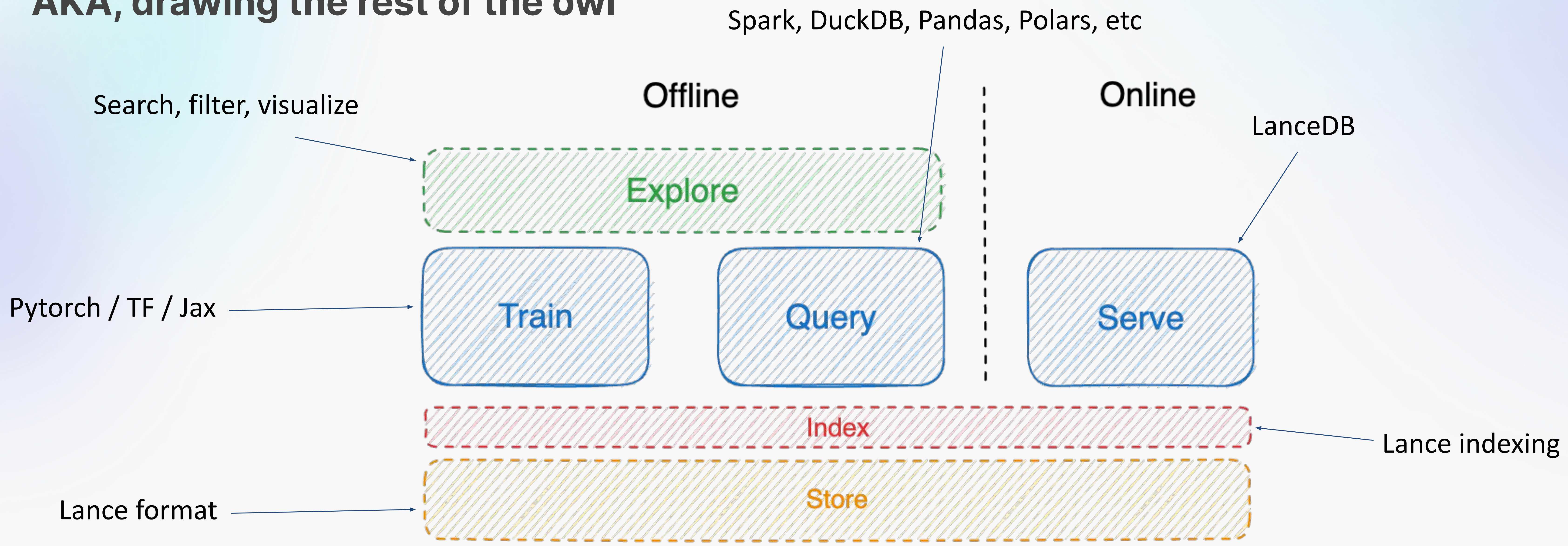


Composable Lakehouse for AI



Composable Lakehouse for AI

AKA, drawing the rest of the owl



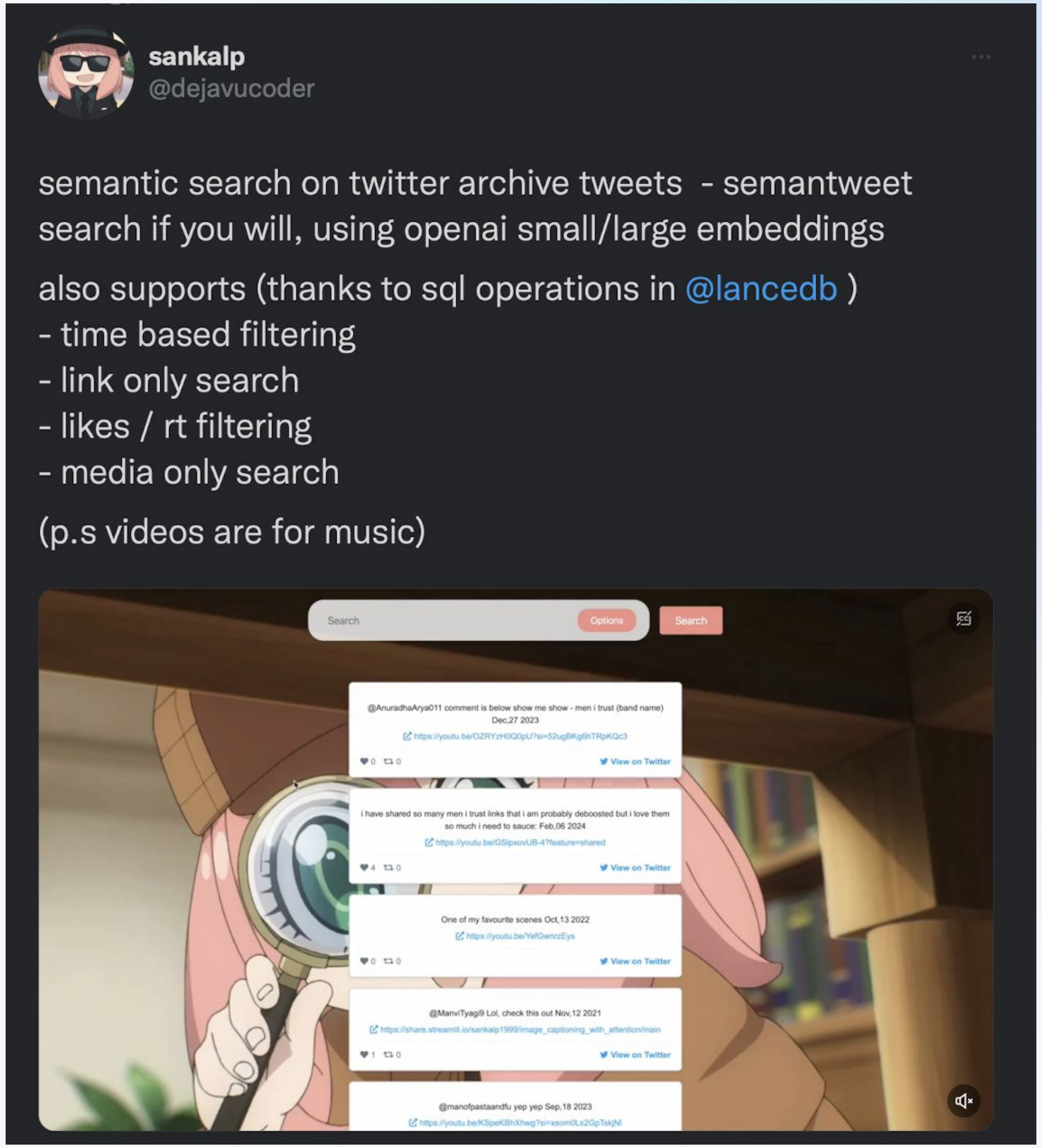
Online: self-optimizing RAG

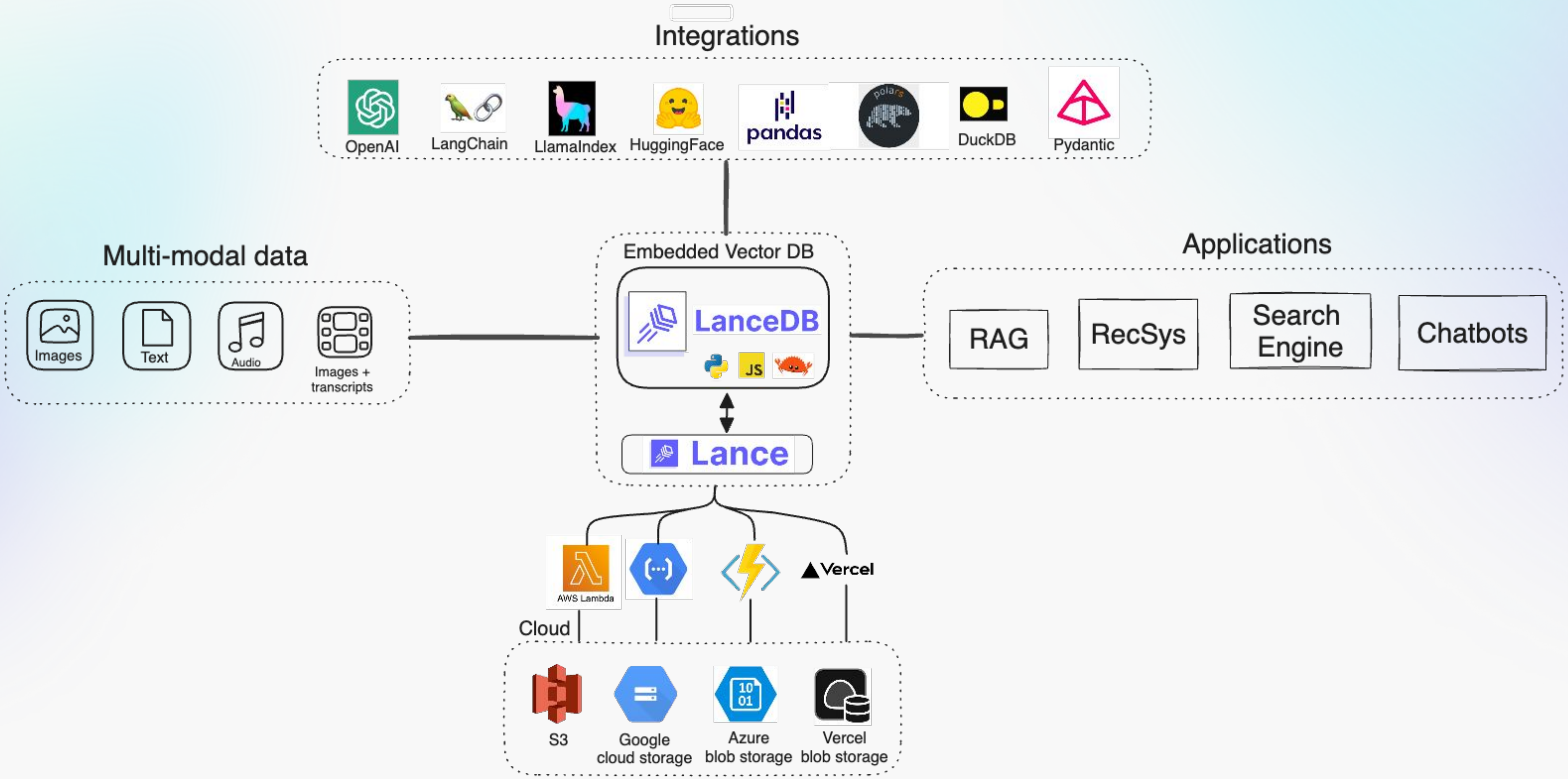
Not just vector search but production quality retrieval that automatically gets better as you use it

LanceDB

Real-time serving

- Lightweight ⇒ SQLite for vector search
- Hyper scalable ⇒ B+ vectors w/ simple infra at a fraction of the cost
- Rich features ⇒ Hybrid search, reranking, SQL filtering

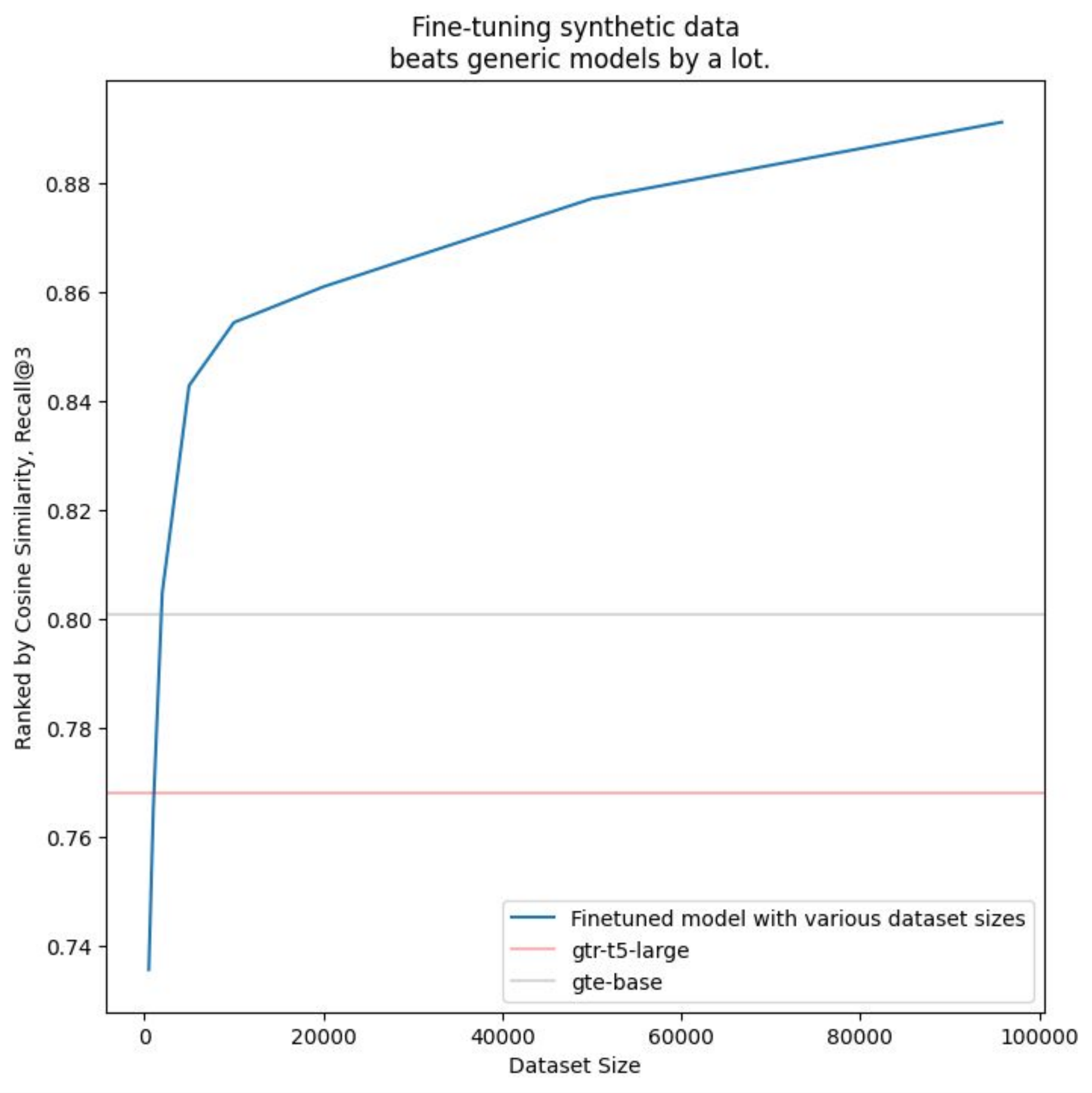



Integrations
OpenAI
LangChain
LlamaIndex
HuggingFace
pandas
polars
DuckDB
Pydantic
Multi-modal data
Images
Text
Audio
Images + transcripts
Embedded Vector DB
LanceDB
JS
Lance
Applications
RAG
RecSys
Search Engine
Chatbots
AWS Lambda
Vercel
Cloud
S3
Google cloud storage
Azure blob storage
Vercel blob storage

Fine-tuning

1st party data >>> big generic model

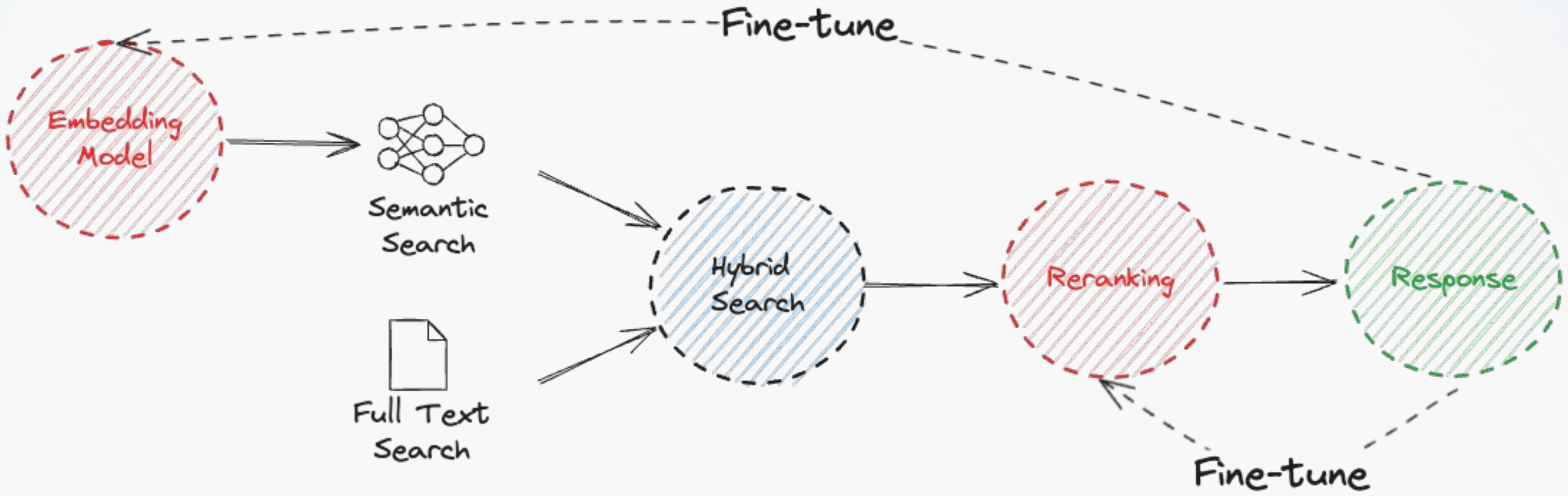
- Tuning embedding model with user feedback seems to be very effective
- Combining it with hybrid search and reranking makes it even more effective
- LanceDB used for both offline fine-tuning and online inference



Source: Chris Moody

RAG is just RecSys in Disguise

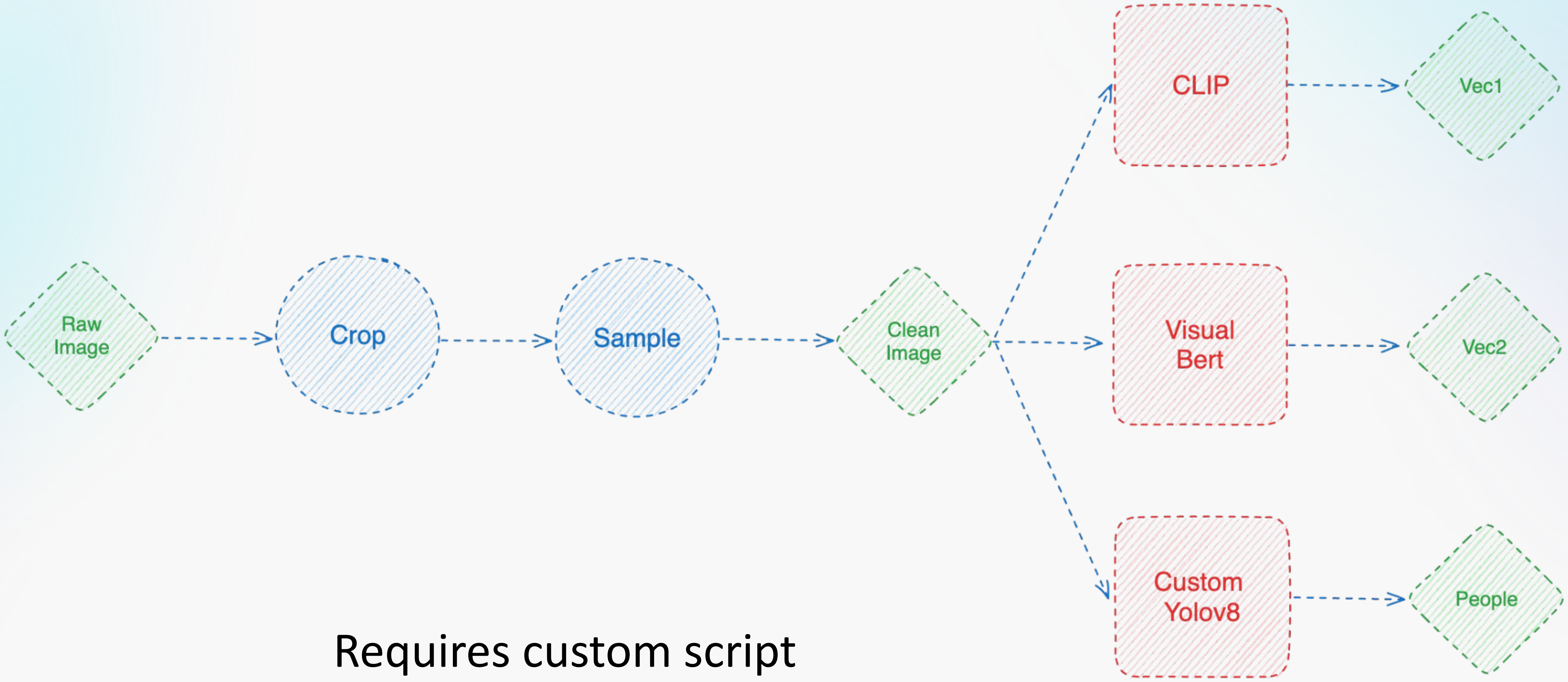
Multiple recallers + reranking + re-training ⇒ recsys



Offline: Declarative AI

One-stop shop for
managing multimodal AI data,
exploration, training, and
rapid experimentation

Declarative AI



Requires custom script

Declarative AI

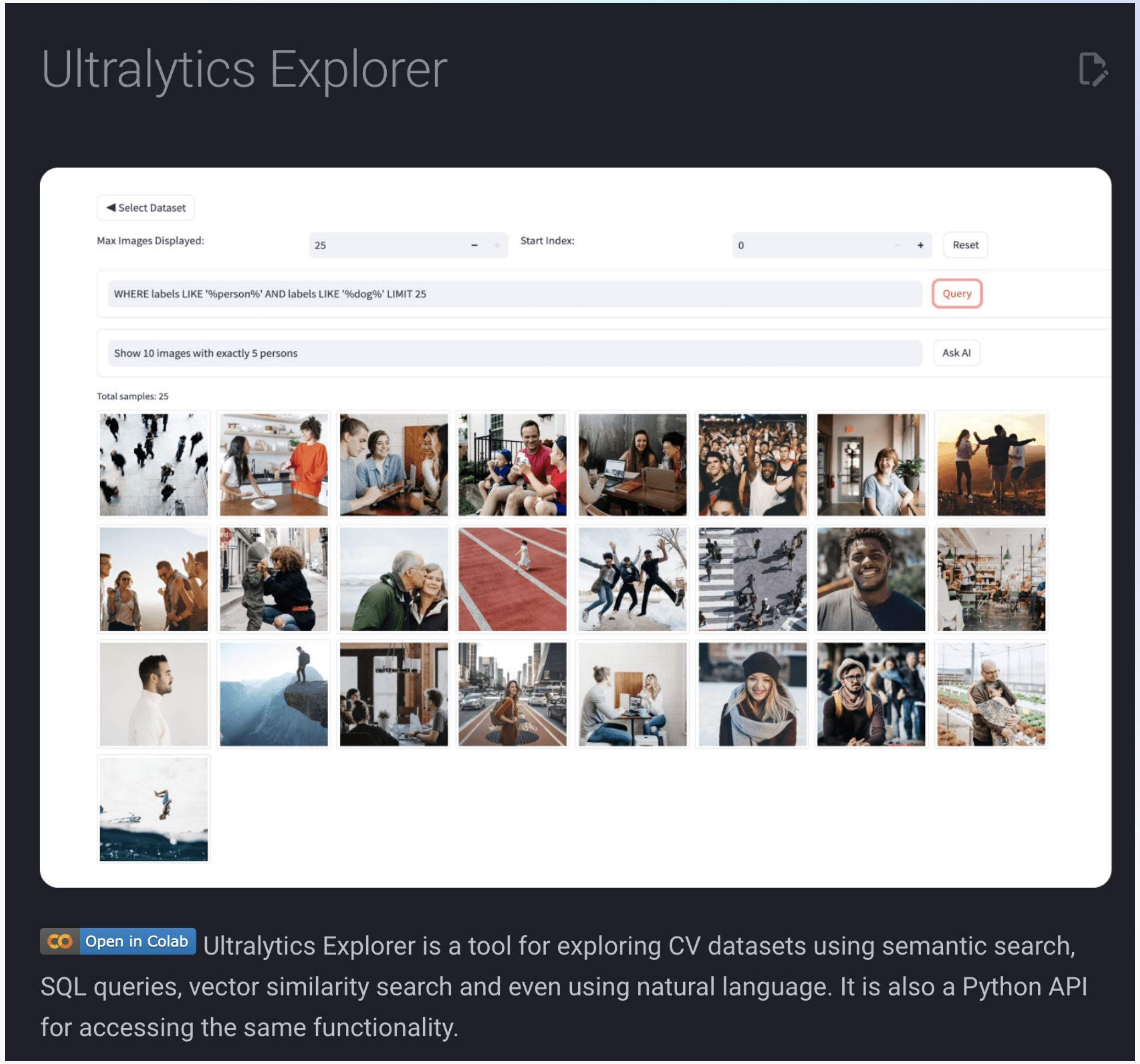
```
class ImageTable(pydantic.BaseModel):

    id: int
    raw_image: Image
    clean_image: Image = sample(crop(“raw_image”))
    image_vec1: Vector(512) = clip.VectorField(“clean_image”)
    image_vec2: Vector(512) = vbert.VectorField(“clean_image”)
    people: list = extract_objects(“yolov8_20240321_final_final”,“clean_image”)
```

Declaring schema is sufficient

Interactive EDA

Explore multimodal data at scale

- Ultralytics (yolov8) released the Explorer product for exploring CV datasets
- Filtering, semantic search, and "Ask AI" features
- Uses Lance/LanceDB under the hood for data storage, management, and OLAP on image datasets



Large scale data processing

SQL, DataFrames, Distributed Engines

- Local experimentation: DuckDB, pandas, polars
- Production: Spark, Slurm, Presto/Trino (in-progress)
- Bulk-ingestion made easy: no more daily dataload that takes 48-hours to finish loading data into service API

Training

Example: pre-train LLM with wikitext_500K

- pytorch / TF data loaders and samplers
- 95% average GPU utilisation
- Minimal CPU overhead
- Don’t need to spend time/effort converting between metadata for filtering / sampling and tensors / blobs for training


```python
# Define the dataset, sampler and dataloader
dataset = LanceDataset(dataset_path, block_size)
sampler = LanceSampler(dataset, block_size)
dataloader = DataLoader(
    dataset,
    shuffle=False,
    batch_size=batch_size,
    sampler=sampler,
    pin_memory=True
)

# Define the optimizer, training loop and train the model
model = model.to(device)
model.train()
optimizer = torch.optim.AdamW(model.parameters(), lr=lr)

for epoch in range(nb_epochs):
    print(f"========= Epoch: {epoch+1} / {nb_epochs} =========")
    epoch_loss = []
    for batch in dataloader:
        optimizer.zero_grad(set_to_none=True)
        batch['input_ids'] = batch['input_ids'].to(device)
        outputs = model(**batch)
        loss = outputs.loss

        loss.backward()
        optimizer.step()
```

Rapid experimentation

Speed of experimentation is king for AI

- Existing formats require making tons of unnecessary copies
- Lance table format supports
  - zero-copy schema evolution
  - Automatic versioning
  - Time-travel
- Perfect for running lots of experiments, tracking changes, and debugging in production

Roadmap

What's coming to Lance format in the next quarter

- Vector indexing optimizations
  - bf16 / f16
  - Additional indexing types
  - Clustering
- Format "V2"
  - Full nullability support
  - Advanced encodings
- Integrations
  - Spark DataSource
  - Ray integration

Thank you!

Check us out if you’re build multimodal AI

- Data format: https://github.com/lancedb/lance
- Vector database: https://github.com/lancedb/lancedb
- Join our community Discord for questions
- Contact: chang@lancedb.com

Appendix

What is the right db for retrieval

With so many options, it's often a confusing choice

Vector databases

- High cost
- Lack of data management
- Not full fledged databases

Traditional db's w/ vector index

- Limited scalability
- No advanced retrieval options
- Bolt-on rather than AI-native

Use cases change quickly, data infra shouldn't

Vector Databases

Pinecone, Weaviate, Qdrant, ChromaDB, etc

👍 Advantages

- Scale
- Feature rich
- Purpose-built APIs

👎 Disadvantages

- No data management
- High cost
- Not full fledged databases

Really just the index + service wrapper

Traditional databases

pgvector, elasticsearch / OpenSearch, etc

👍 Advantages

- No duplication of infra
- Data management already figured out
- Store other data along with vectors

👎 Disadvantages

- Scalability
- Advanced retrieval features
- Ease of use

Great for small scale and AI as a "side-feature"

Mo AI mo problems

- Vector databases:
  - Only deal with vector data
  - Only deal with vector search
- Pgvector does not scale
- FAISS requires you to manually build the rest of the database and stitch together multiple systems
- No effective storage solution at all for unstructured data

Mo data mo problems

- Parquet (Delta/Iceberg/Hudi) is only good for analytics
- TFRecords is only good for training
- JSON is often needed for debugging
- Now you have 3 copies of the data
    - Problem 1: Ballooning storage costs
    - Problem 2: Need different compute for each format
    - Problem 3: How do you know they're actually in sync?
    - Problem 4: It's still slow

Lance columnar format

Alternative to parquet for AI

💾 New layout

Analytics need fast scans. Training I/O need fast random access.

✨ **Ecosystem**

The main reader/writer interfaces of Lance is all through Arrow.

🔥 Rust implementation

Performance, safety, SIMD

🚀 Versioning

Zero-copy versioning, schema evolution, time travel

Lance columnar format

It's so awesome I need two slides

💾 Unified storage

Store and query embeddings, text, images, pdfs, videos, audio, point clouds, alongside tabular data

✨ **Plug-and-play**

Convert data with 2 lines of code. Compatible with pandas, polars, duckdb, spark, jupyter, and more

🔥 Performance

Reduce training time by up to 3x with faster filtering, shuffling, and data loading. Up to 2000x faster than parquet for AI

🚀 Compute storage separation

Store on cheap blob storage, stream data into accelerated compute for training

Encodings

Lance is designed to be good for both large scan and random access.

Support storing large blobs. Contrary to conventional wisdoms in OLAP and columnar store designs.

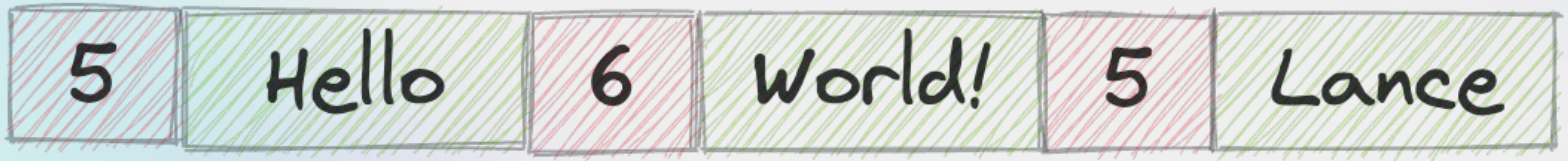
Encodings: Design Principles

- Two very simple design principles:
  - Scan: do not scan more data than Parquet / ORC
  - Point query:
    - Sub-linear time complexity to read one row
    - Amortize metadata overheads
- Revised storage optimizations in 2023

	PCIE NVME	S3
Block Size	4-16KB	32-256KB
Bandwidth	5000MB/s+	100 Gb/s (*EC2)
IOPS	1,000,000+ @ 32QD	5000 GET/HEAD

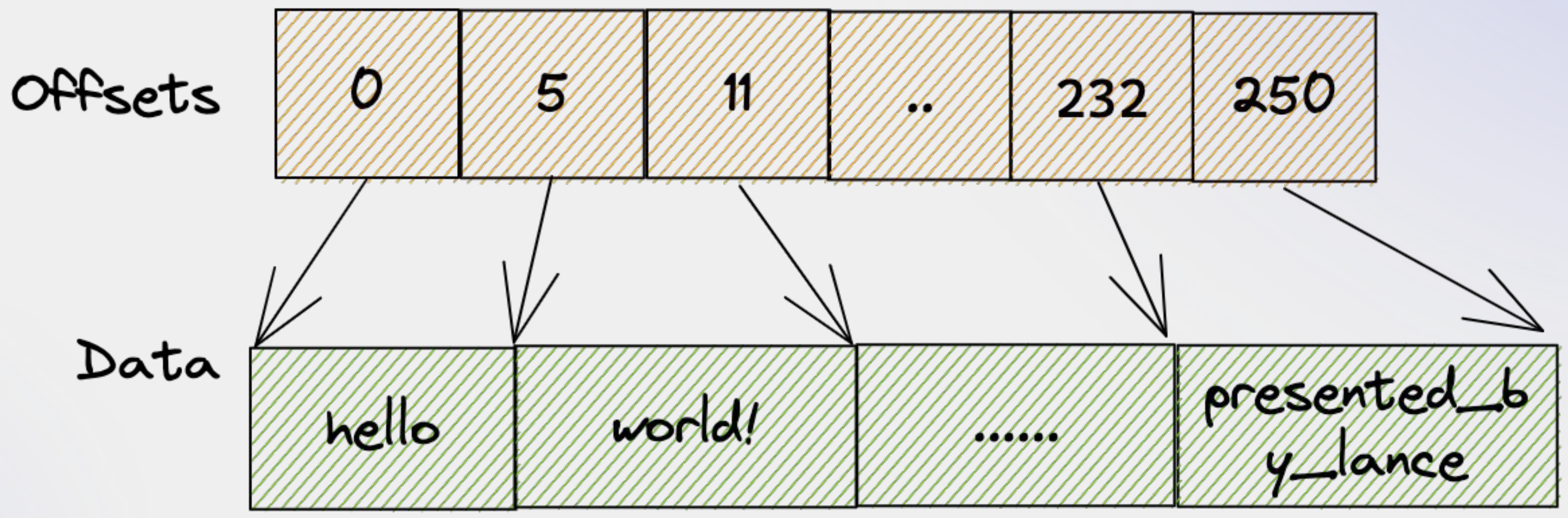
Encodings: Binary Encoding

- In Parquet, length and data are interleaving.
- Can not access one data point without deserializing all data in the group
- Used to store var-length bytes (Image / Lidar)



Encodings: Binary Encoding

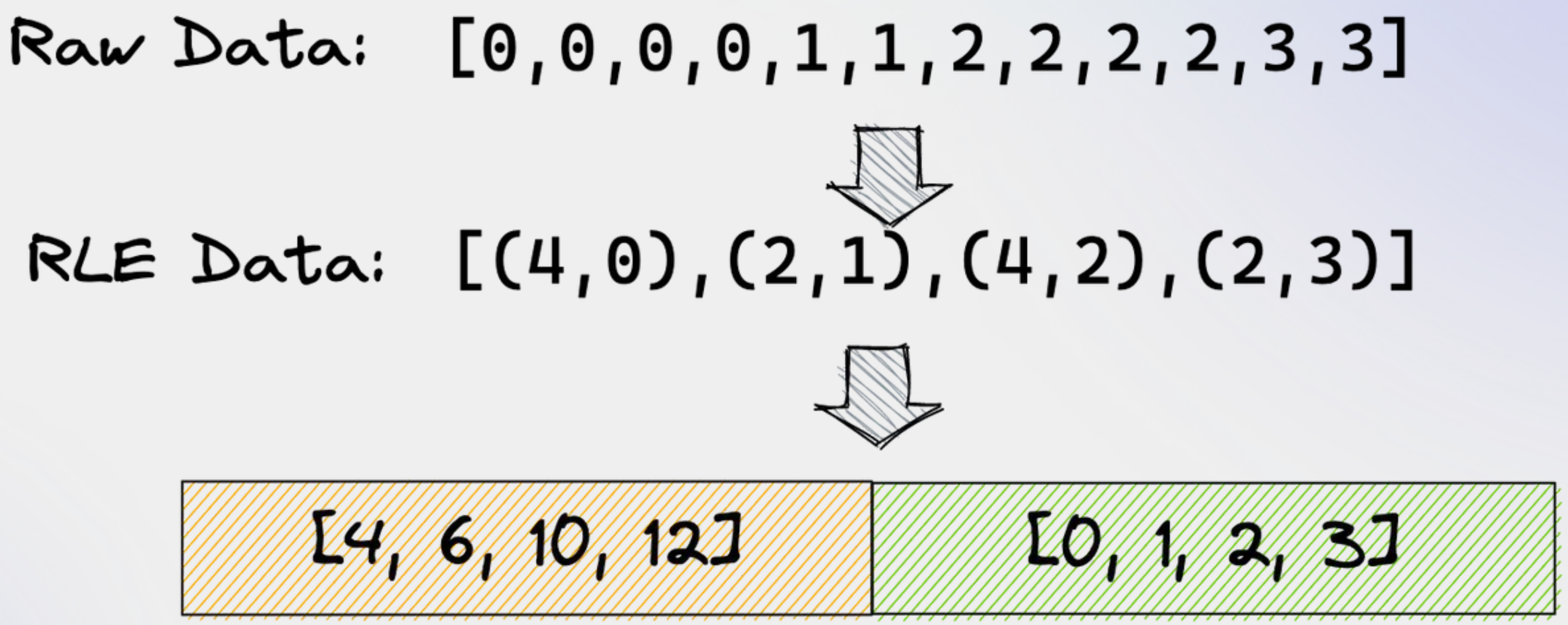
- Var-length Binary Encoding
  - String, Bytes, Image, Lidar PointCloud
- data array + offset array
- O(1) offset read + O(1) data read



Eto.ai

Encodings: RLE(*)

- Run-Length Encoding
- Cumulative run-length array + value array
- O(logn) offset lookup + O(1) value read
- Effective for sparse vectors



Lance vs Other Formats

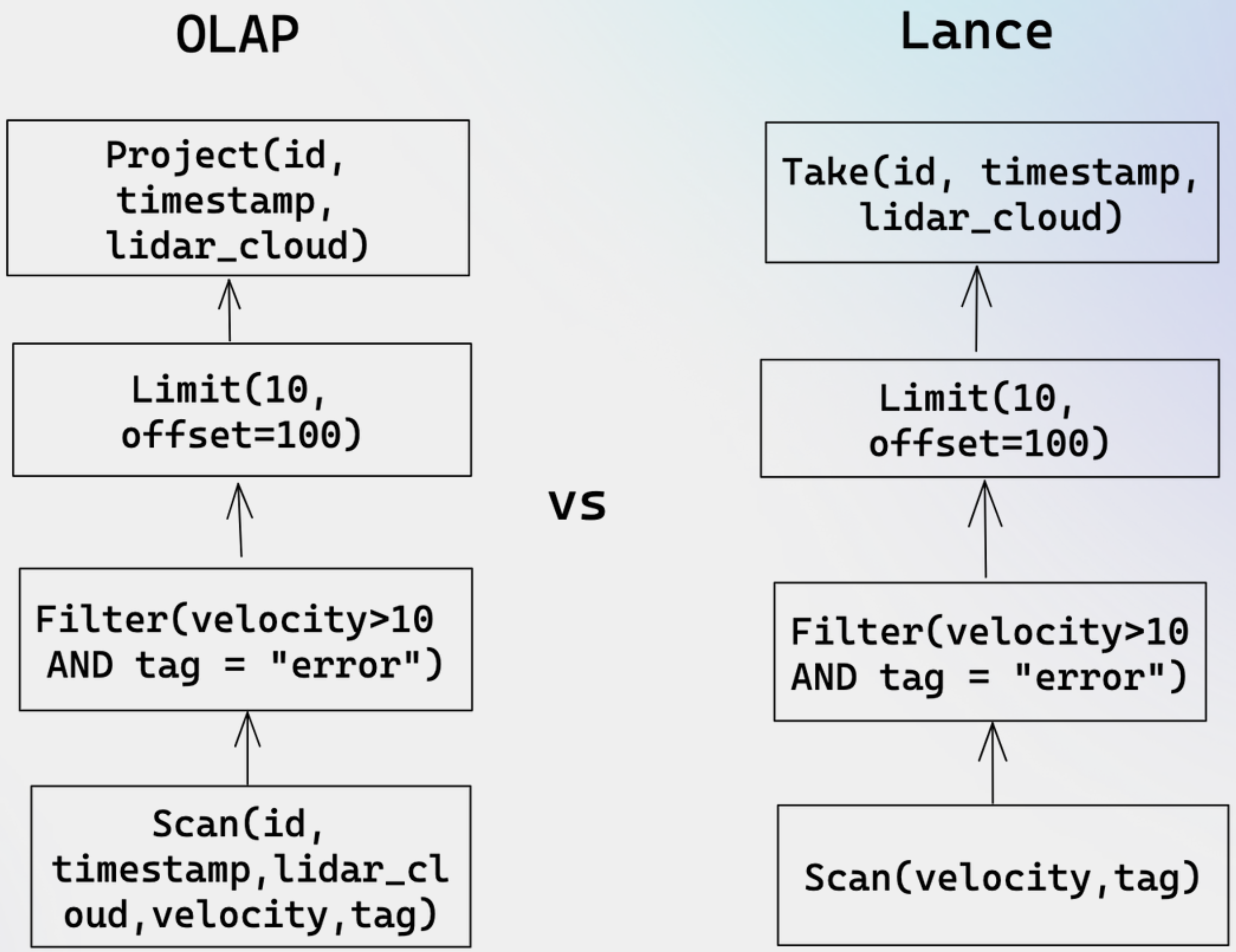
	Lance	Parquet / ORC	JSON / XML	TFRecord / HDF5	Database (Sqlite / Postgres)
Scan (Training / Mining)	Fast	Fast	Slow	Fast	Slow
Point Query (Debug / Shuffle)	Fast	Slow	Fast (small file contention)	Slow	Fast
Eco-system (Language / Library)	Good	Good	Good	Bad	Non-ML: Good ML: Bad

What tradeoff did I have to make here?

I/O Execution (1/2)

- Different to typical OLAP plan.
- Optimized for slicing and dicing with large blobs.

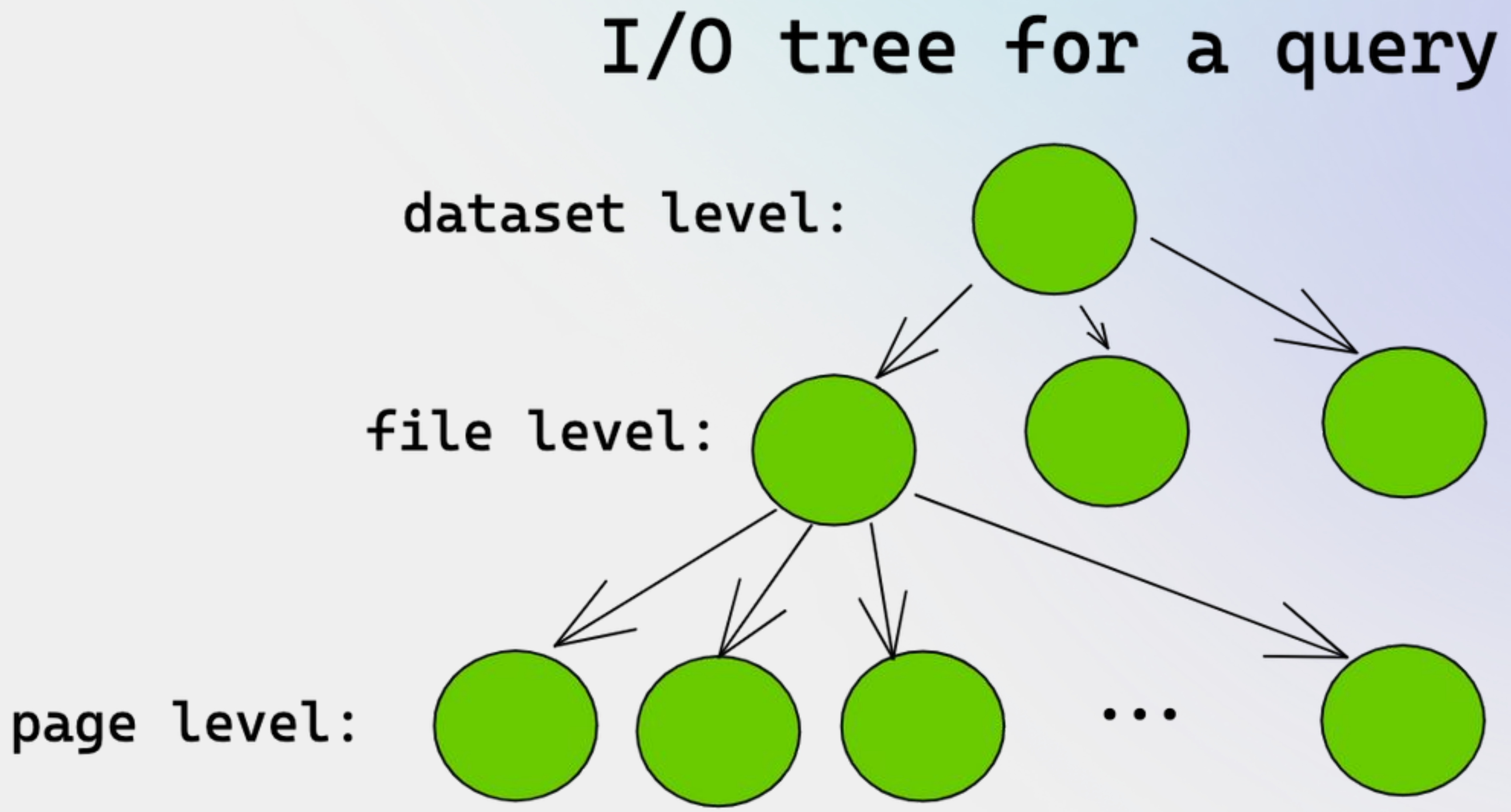
```
SELECT id, timestamp, lidar_cloud
FROM dataset
WHERE velocity > 10 and tag = "error"
LIMIT 10 OFFSET 100
```



Eto.ai

I/O Execution (2/2)

- Take advantages of
  - NVME's deep I/O queue
  - S3/GCS high parallelisms
- Eliminate data dependency between I/O requests when possible
- Issue large amount of parallel I/Os
- A short but wide I/O dependency



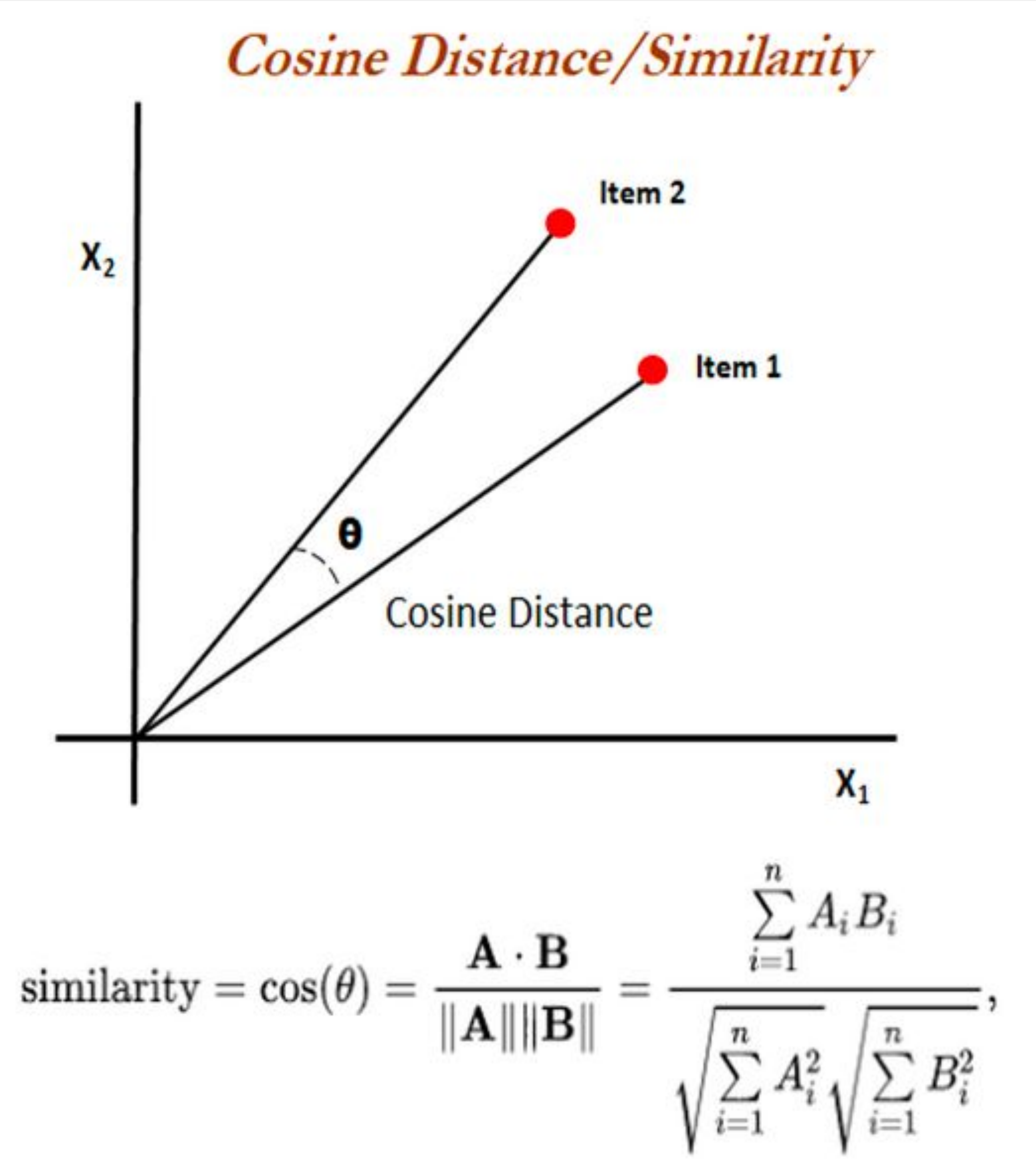
Secondary Indices

- Nice side-effect of Fast Point Query performance
- Support Approximate Nearest Neighbors (ANN) search on vectors
- Extensible to other index types

Why add indexing?

- Random access perf makes it worth it
- Mix SQL with FTS with vector search
- Store and query the data together - replace multiple tools with just one

Vector Distance



Anything can be turned into a vector

Vector (ANN) Index

- Approximate answers
- Hash-based approaches (LSH)
- Tree-based approaches (Annoy)
- Partitioning approaches (**IVF**)
- Graph based approaches (HNSW / **DiskANN**)
- Compression (**PQ**, **OPQ**, LOPQ, etc)

Latency vs recall

Vector Index

keyword search returns "customer"

Say you type "cust"

vector search returns "customer" and "user"

How is Lance vector index different?

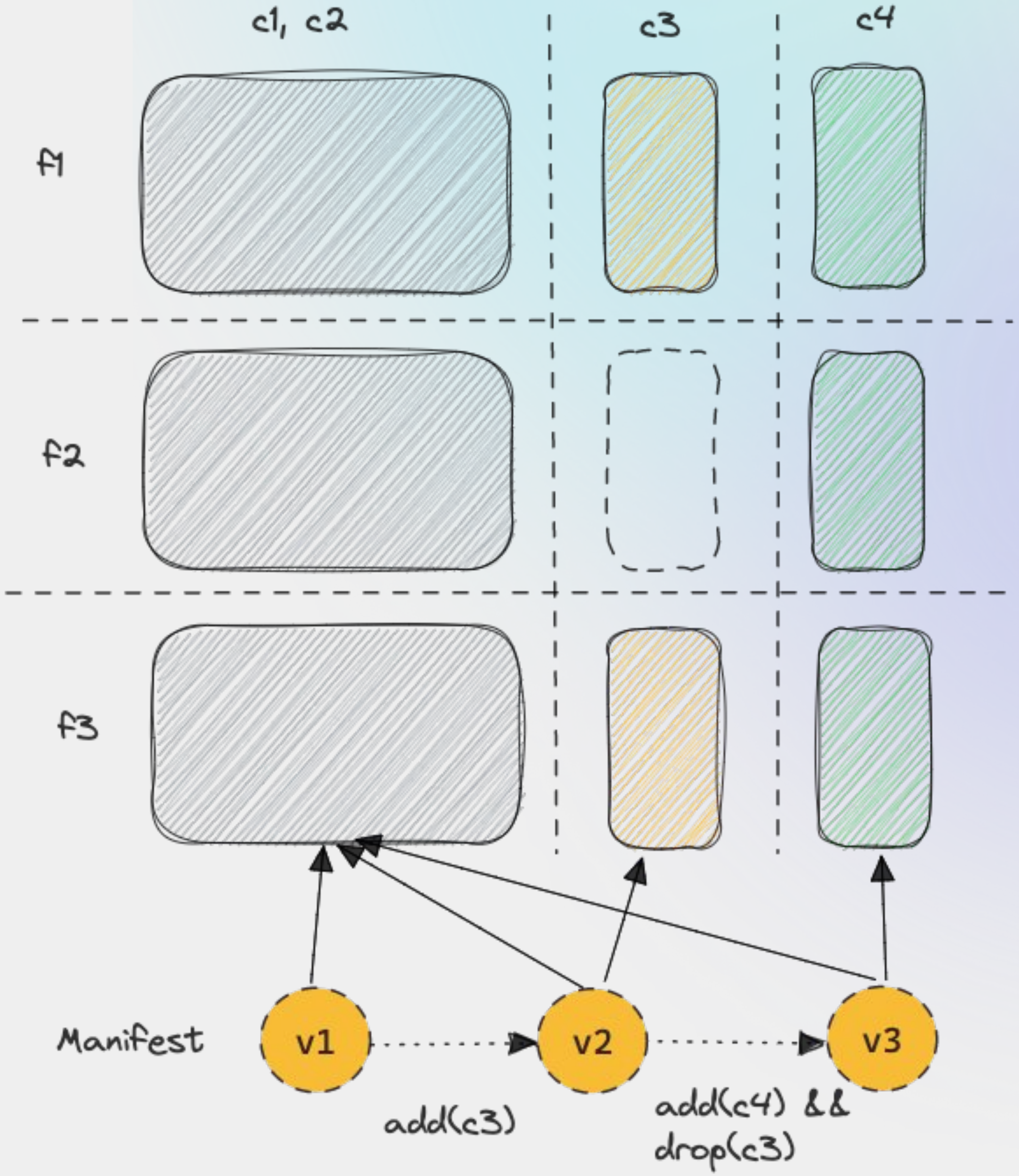
- Disk-based (easy and cheap to scale)
- Allow you to retrieve features together with vectors
- Low-level SIMD optimizations
- Supports B+ scale search on a single node

Versioning and Schema Evolution

- Zero-copy Append, Add & Remove Column, Snapshots
- Write-ahead log for Fast Data and Index
- Simple Data Management API

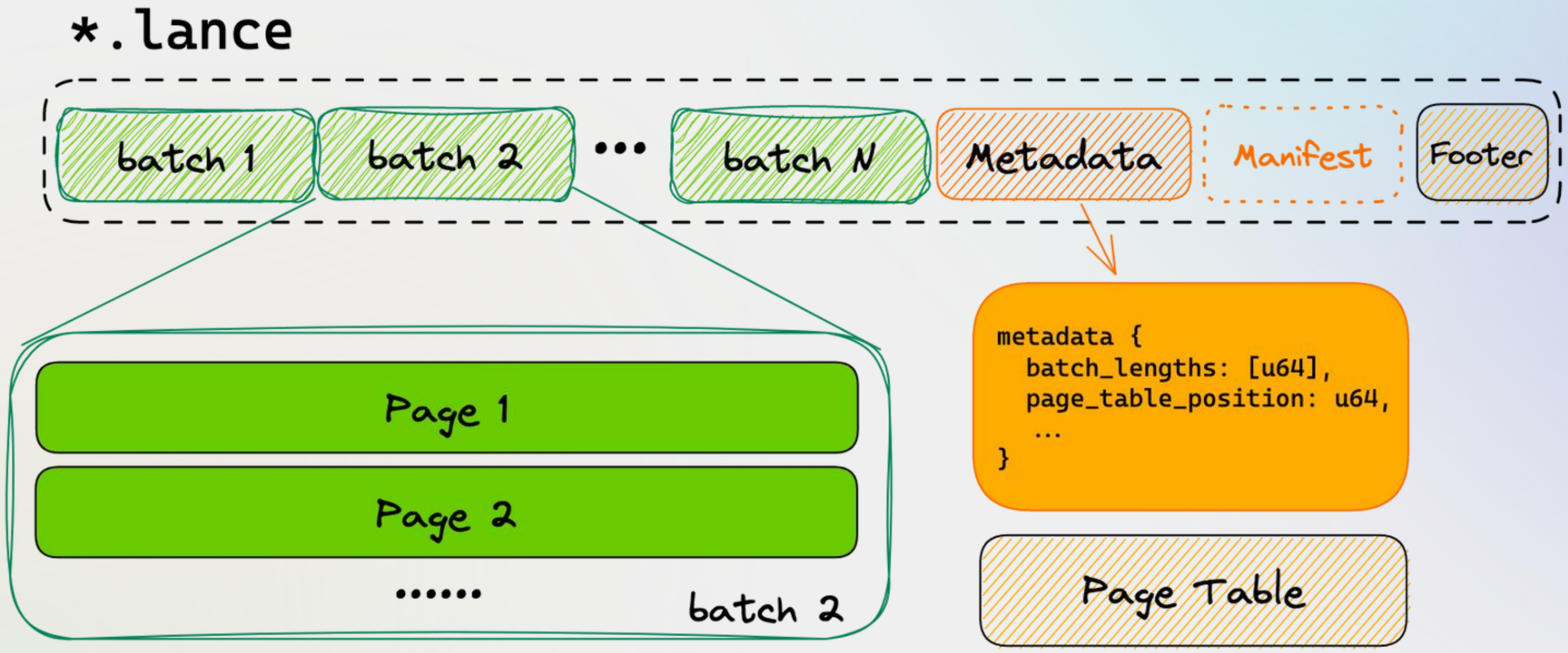
Versioning & Schema Evolution

- Manifest file tracks all metadata of one version.
- Fast version checkout
- Data fragment for partitioning and schema evolution
- Rich commit message with version.
  - Useful for lineage / commit tracking
- Lazy column materialization*



Eto.ai

File Layout



Roadmap

01 Partitioning Pruning, Row Group Pruning

02 DiskANN

03 Fast updates*

04 NA-handling

05 Data Compression, RLE

06 Spark integration (Scala)

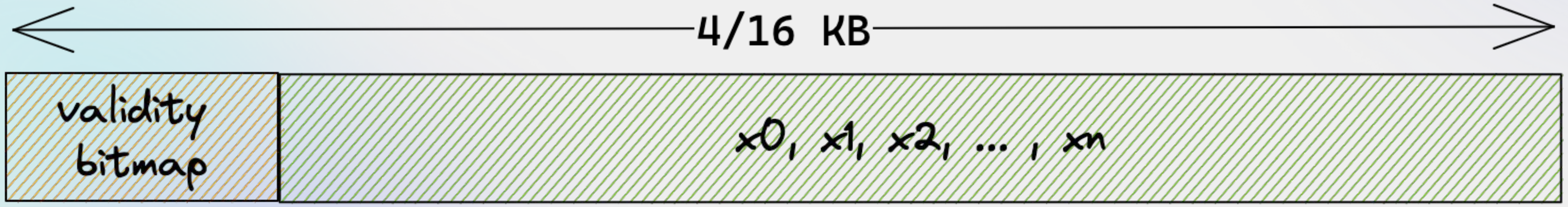
07 Semantic types

References

- Benchmarks vs parquet
- SIMD optimization
- Data format

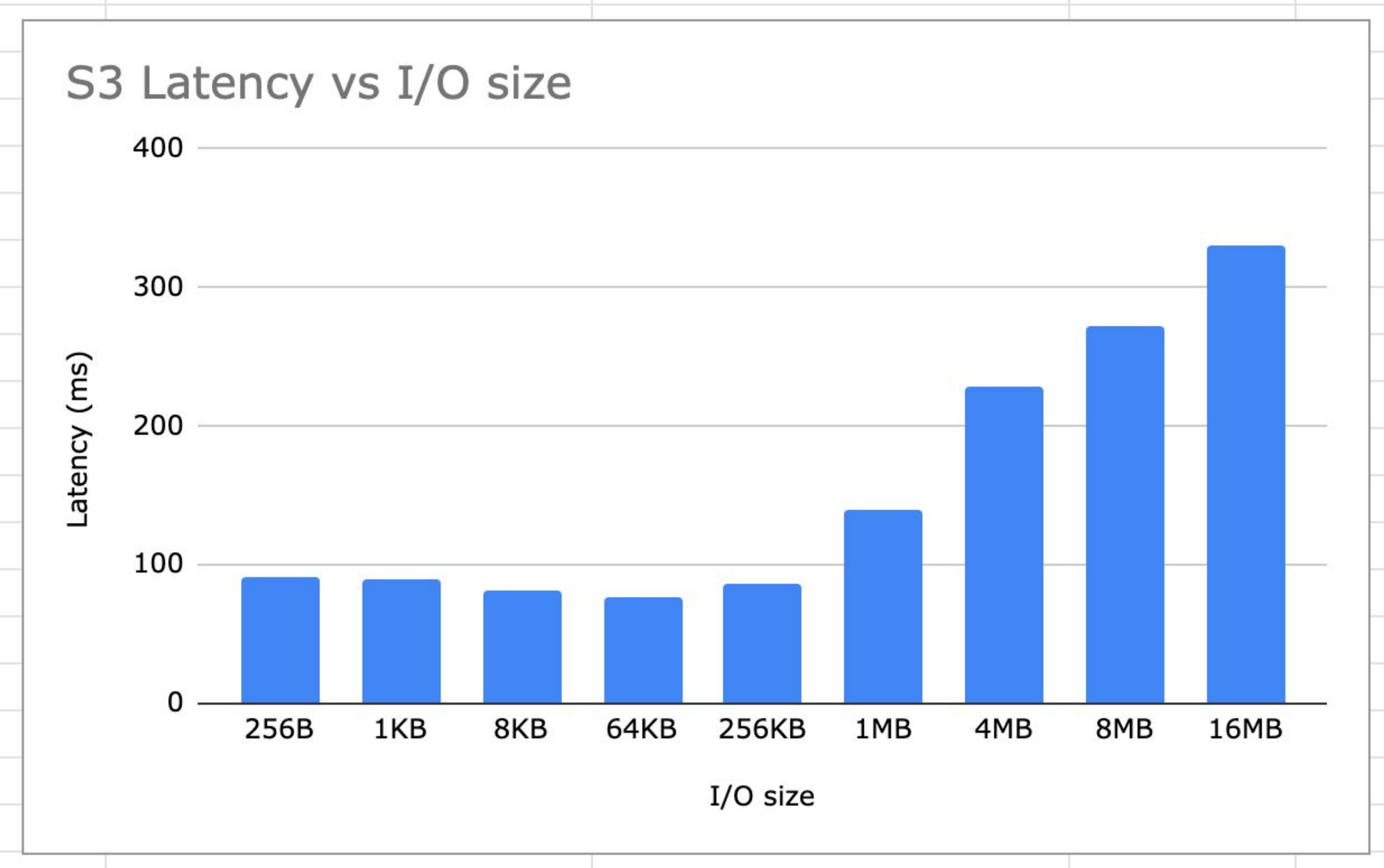
Encodings: Plain Encoding (Cont'd)

- Nullability and compression (*)
- Within I/O block size, does not negatively impact random I/O performance



Eto.ai

Optimal I/O Size on S3



Eto.ai

Nested Schema