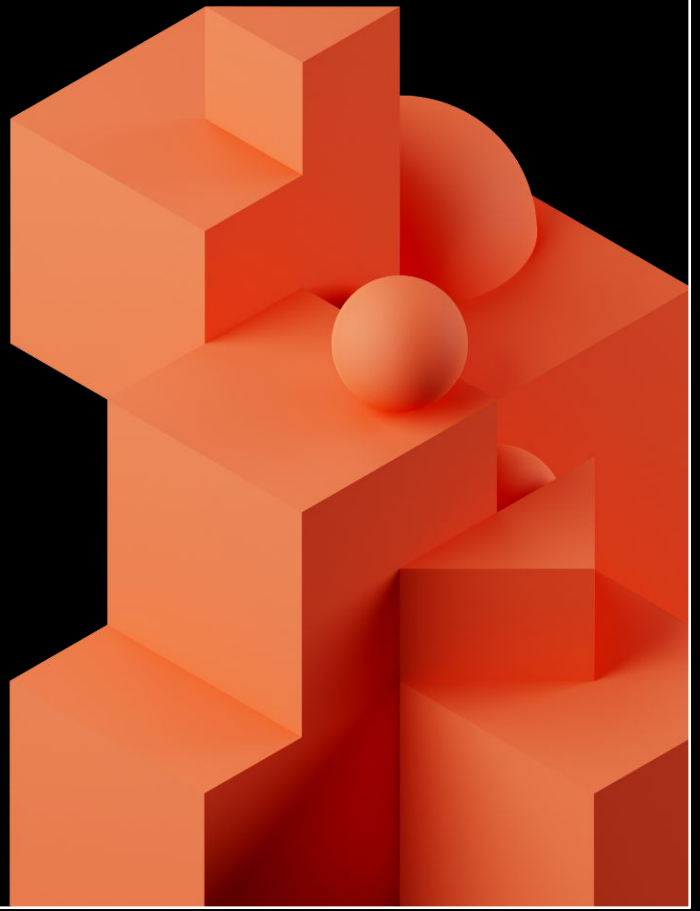
DATA+AI SUMMIT
BY databricks

Vector Data Lake

Do ***you*** need (more than) a vector database in 2023?



Databricks
2023

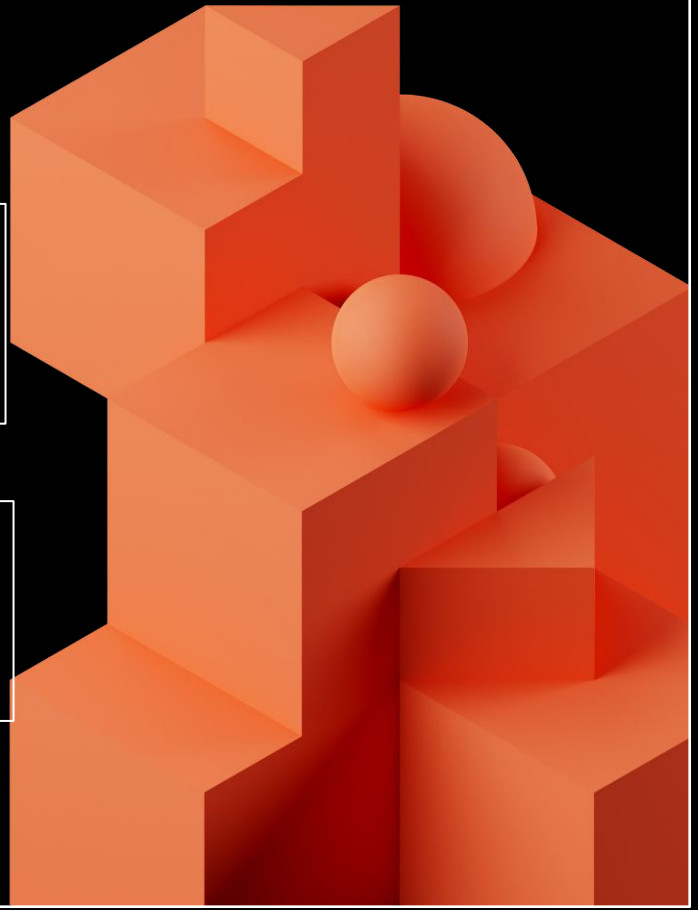
DATA+AI SUMMIT
BY databricks

PhD student at Stanford University

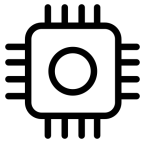
https://github.com/marsupialtail/quokka
@marsupialtail_2

Co-author Lance format
Co-author pandas

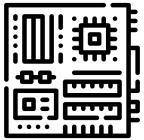
@changhiskhan



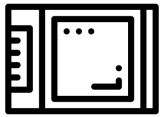
Generative AI is missing a storage layer

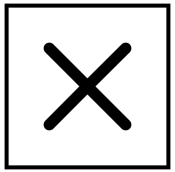


GPT-4, LLaMA, PaLM, Alpaca

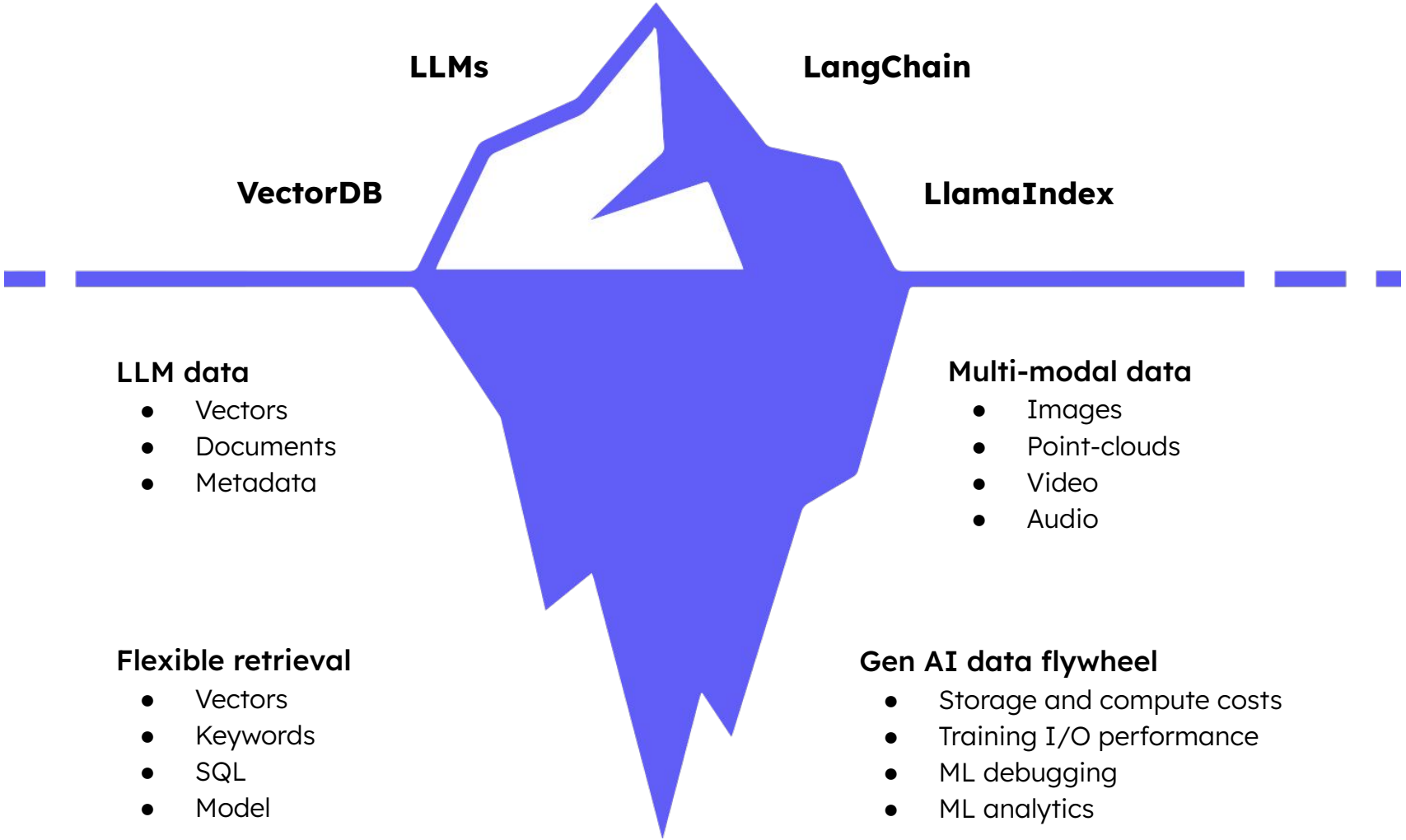


LangChain, LlamaIndex, AutoGPT





- Vector databases only deal with vectors
- Pgvector and similar does not scale
- No effective solution at all for multi-modal data



LLM data

- Vectors
- Documents
- Metadata

Multi-modal data

- Images
- Point-clouds
- Video
- Audio

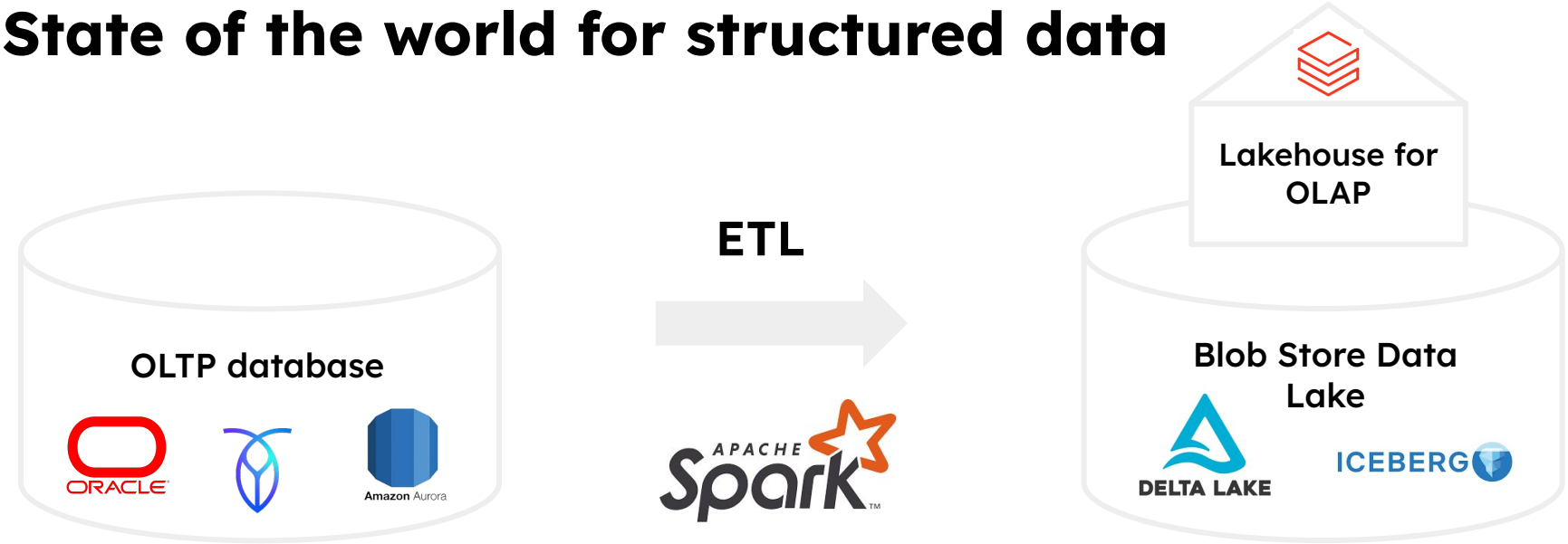
Flexible retrieval

- Vectors
- Keywords
- SQL
- Model

Gen AI data flywheel

- Storage and compute costs
- Training I/O performance
- ML debugging
- ML analytics

State of the world for structured data



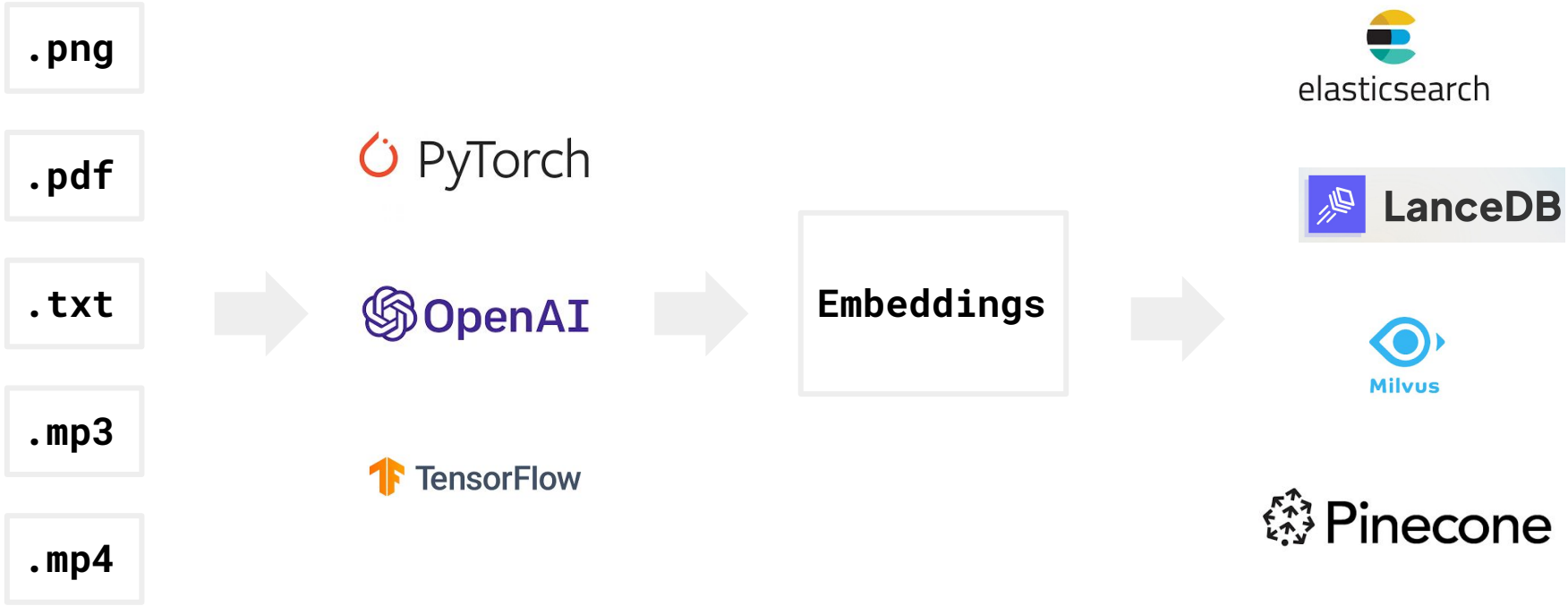
Desired Properties:

- Fast writes
- Strong consistency
- “Operational” SQL: e.g. selecting a row

Desired Properties:

- Efficient bulk updates
- Fast full scans
- Decouple compute/storage

State of the world for *unstructured* data



Our observations

TLDR: current cloud vector databases much resemble classic OLTP stores, optimized for operational workloads.

Strong focus on write TPS and consistency: ElasticSearch/Milvus both offer strong write consistency. All vector databases focus on write throughput.

Excels at point updates, low latency for point reads.

Integrated compute-storage: requires heavy indexing that needs to be kept live in always-on RAM/SSD, Trino-for-embeddings don't really exist.

But what about OLAP workloads?

What even are OLAP workloads for embeddings?

- **Recommendation models:** batch update embeddings, batch update recommendations
- **Data analytics:** which video genre contain the most inappropriate videos?
- **ML training on embeddings**.

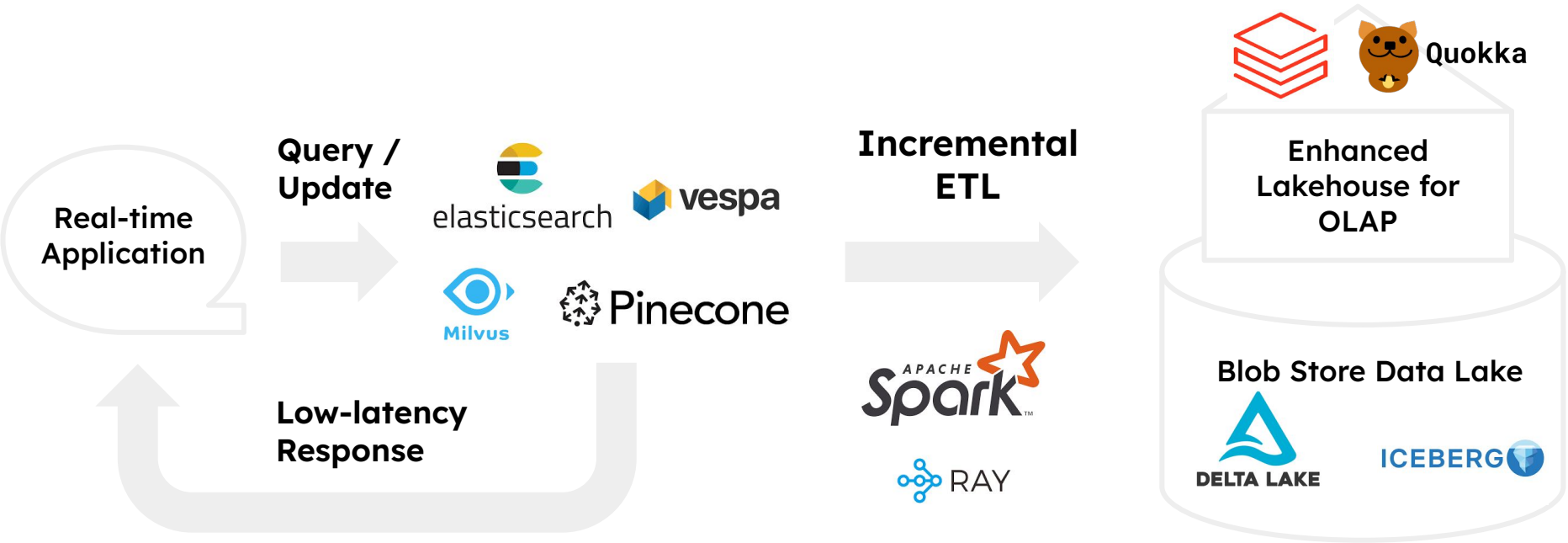
Characteristics of these workloads

Observations:

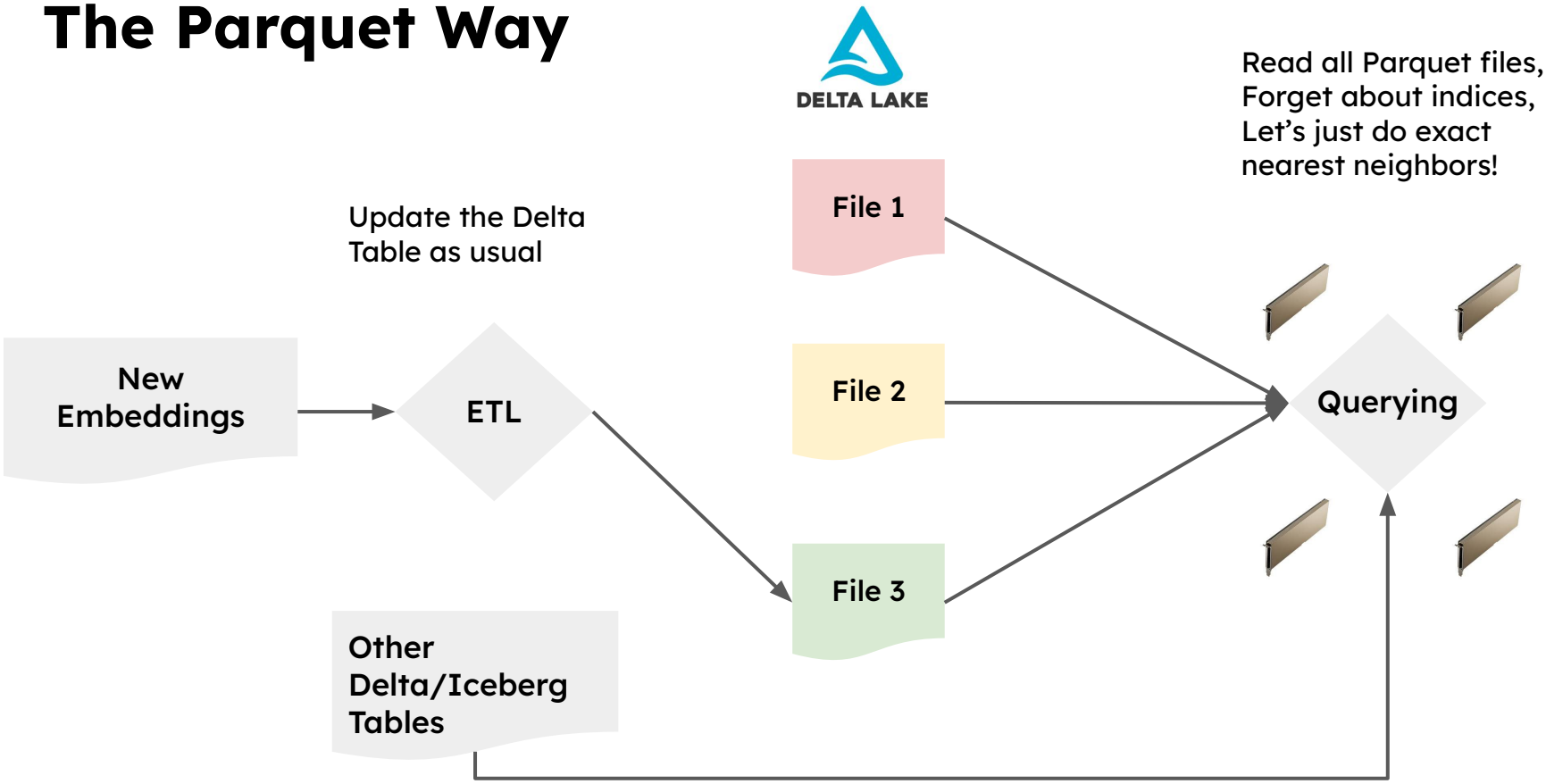
- (Very large) batched nearest neighbor lookup / range search
- Doesn’t care too much latency
- Data is usually immutable or updated in large batches
- Really want to decouple storage and compute.

Not quite the optimization target of current vector databases, but should remind you of Spark/Trino

Proposal - Do it in the data lake



The Parquet Way



Quokka

Distributed dataframe library that supports vector operations

Simple DataFrame API

```
vectors =
qc.read_parquet("s3://microsoft-turing-ann-p
arquet/*")

results = vectors.nn_probe(probe_df,
vec_column_left = "vector", vec_column_right
= "probe_vec", k = K)

results = results.filter( . . . )

results = results.join( . . . )
```

Runs on GPUs

NN search on 100M 100-dim vectors:

- Number of probes: 100 (< $0.10)
- Number of probes: 100k (estimate $10)

Storage cost: $1 / month (S3)

(Pinecone estimate: $320/month)

Introducing Lance format

Open source columnar format for AI

Unify AI data storage and reduce data lake TCO

🏹Arrow Interface

Compatible with pandas, polars, duckdb, spark and more

🔥AI data

Optimized for unstructured data types like images, audio, video and 3d point clouds

🚀Performance

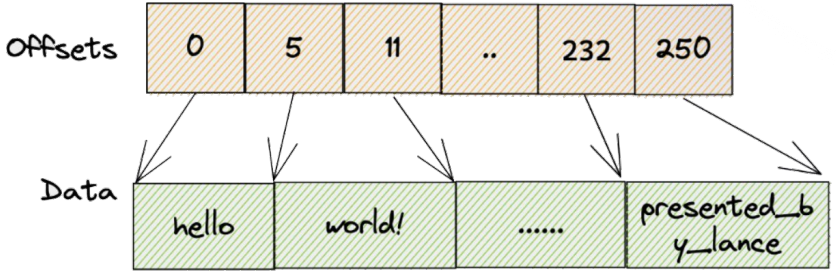
2000x faster than parquet for random access

🦀Indices

Vector index, fts index*, regular database indices*

https://github.com/lancedb/lance

Lance



Quokka

```
vectors = qc.read_lance("s3://...")

# Uses indexes automatically!
results = vectors.nn_probe(probe_df,
"vector","probe_vec", k = K)
```

Parquet

- Must read whole group to access 1 data point
- Matters a lot if storing large blobs like images, point clouds, etc

Lance

- Offsets determine exact byte range to read
- 2000x random access performance

Conclusion

- There are large scale vector workloads in the data lake
- Vector databases are not a good fit for those
- Parquet works but is still suboptimal
- Lance is a better format to support ANN in data lake
- Roadmap: Will look to benchmark vs parquet once batch ANN is in place