MySQL HeatWave Lakehouse

Analytics at the Speed of Thought

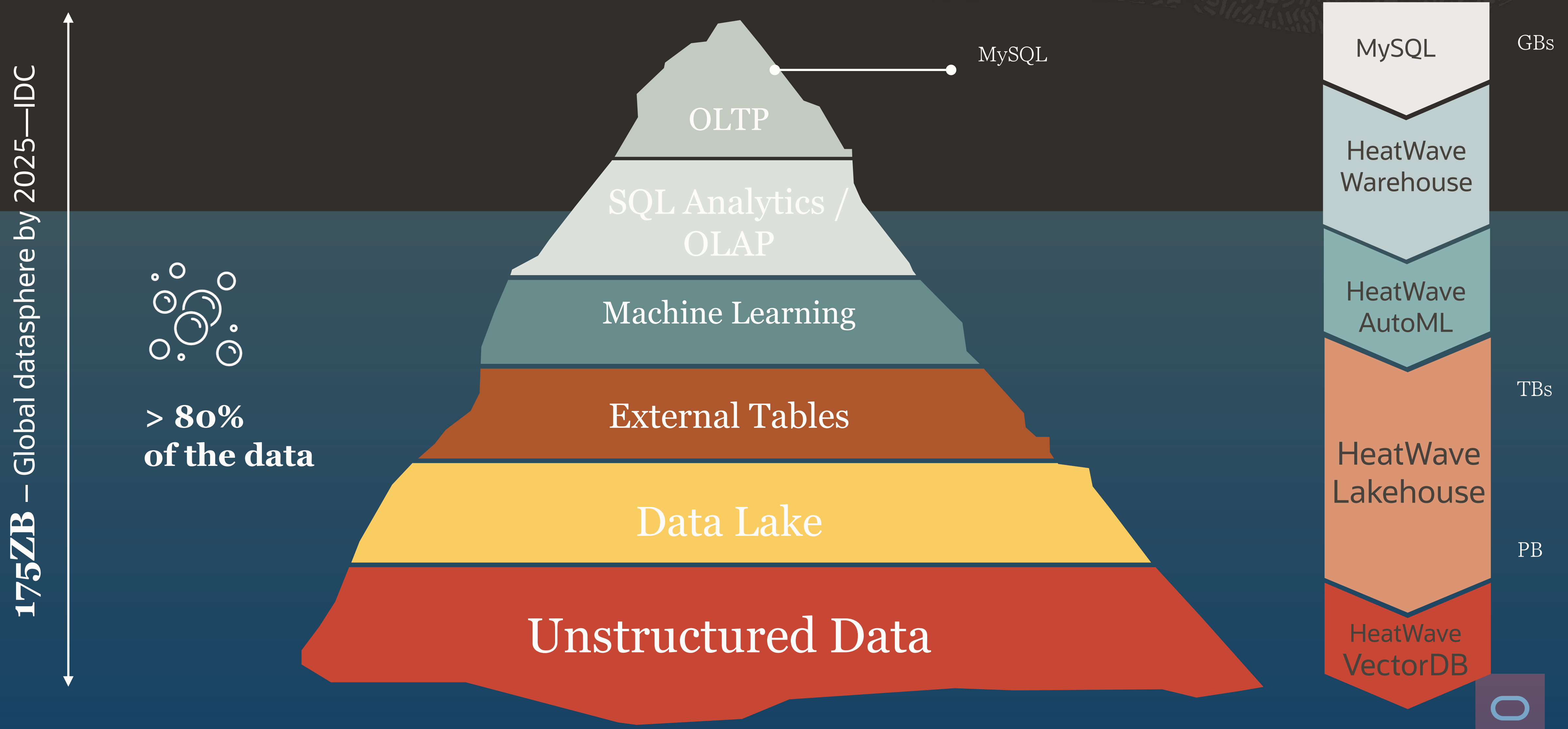
Cagri Balkesen, Ph.D.
Architect, MySQL HeatWave

January 31, 2025

Safe harbor statement

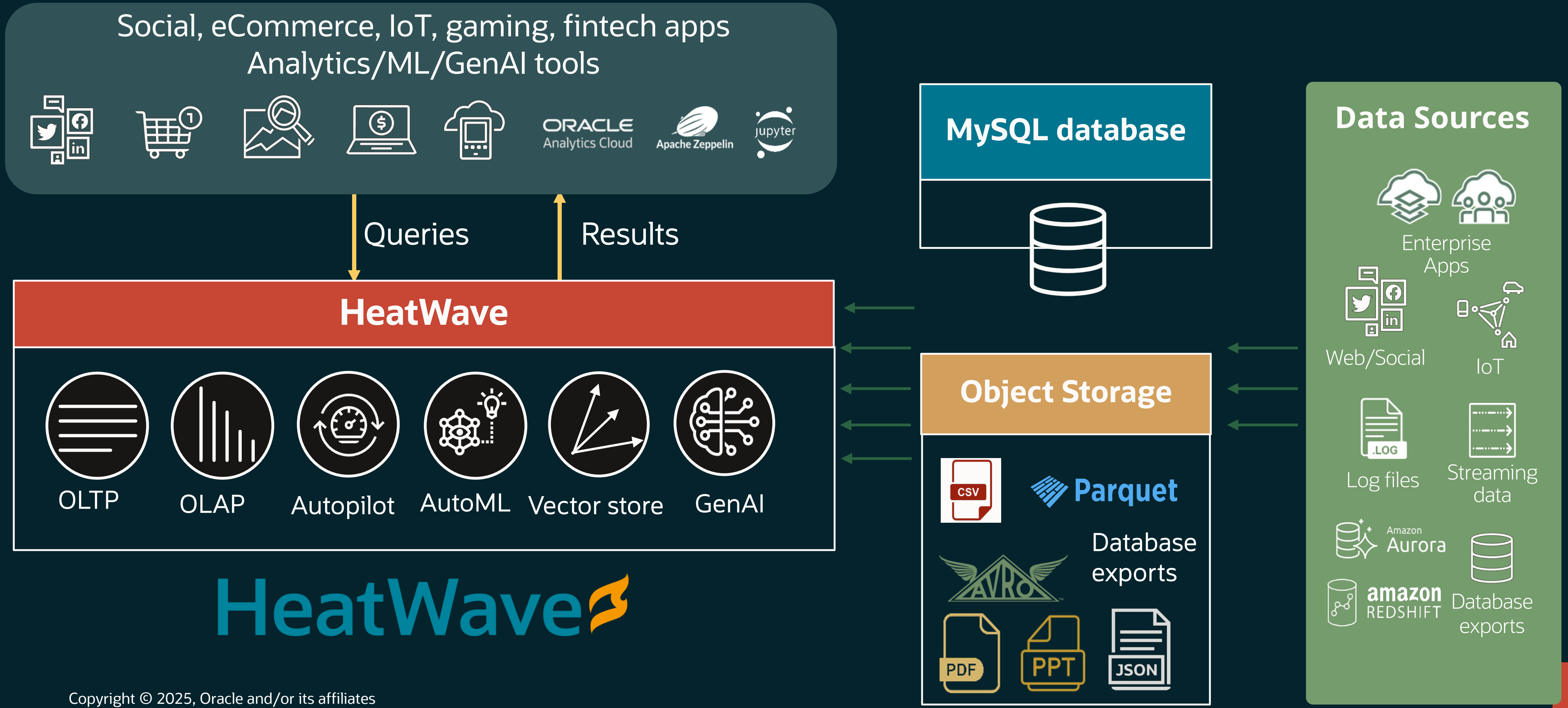
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

Unprecedented increase in volume of data

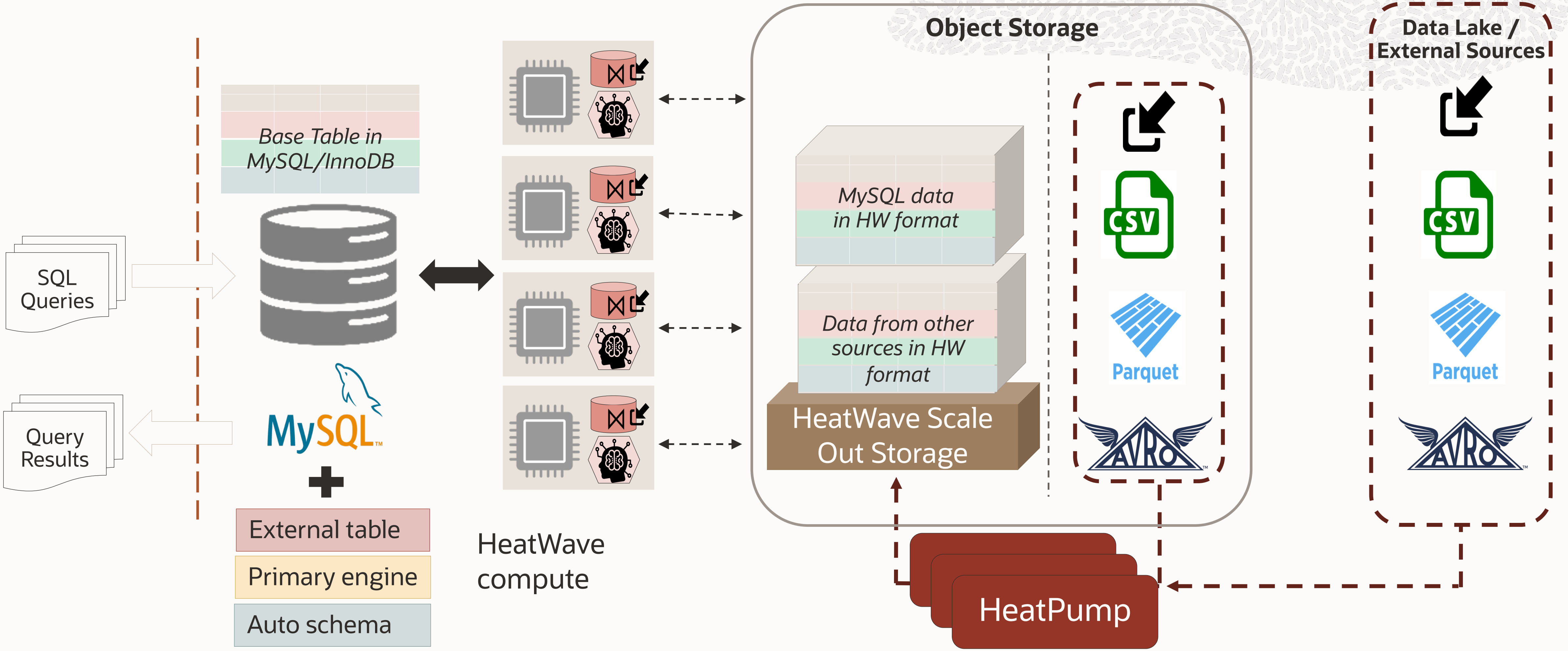


HeatWave —OLTP, OLAP, ML, **Lakehouse**, GenAI, Vector Store

Transactions, real-time analytics across data warehouse and data lake, machine learning, GenAI in one database service



HeatWave Lakehouse
Architecture Overview



- Introduction of external table concept
- HeatWave as the primary engine for data lake
- Automatic schema inference for external tables

- Fast & scalable data load from object store
- External source to internal HW format transformation
- Automatically collects statistics for query performance

HeatWave Lakehouse Table Interface

Extensible and MySQL compatible

A new primary engine Lakehouse serves as an interface for data in object store

```
> CREATE TABLE tbl_name <create_definition> ENGINE=LAKEHOUSE
  ENGINE_ATTRIBUTE='<engine_options>'
  SECONDARY_ENGINE=RAPID;
```

Provides Lakehouse-specific functionality with existing syntax and is extensible

- External source file locations specified in extensible JSON interface through the standard MySQL ENGINE_ATTRIBUTE
- Compatible with existing systems, such as Snowflake, Databricks to allow easy migration to Heatwave

```
ENGINE_ATTRIBUTE='{
 "location":{"object_storage_provider": default | aws | oci,
             "mode": "cloud_parameters" //default if object_storage_provider is OCI |
                   "uri" //default if object_storage_provider is AWS }
 "file": [file1, file2, ...],
 "dialect": { "format"= "csv"|"parquet"|"avro",
   "field_delimiter": ",",  "record_delimiter": "\n",
   "escape_character": "\","quotation_marks": "\"",  "skip_rows": "0",
   "encoding": "UTF-8", "date_format": "auto",  "time_format": "auto" }
}
```

cloud_parameters

“bucket”: “test-bucket”
“namespace”: “lrsrfayerklw”
“region”: “uk-London-1”
“prefix”: “src_data/customer.csv”

“par”: “https://objectstorage.uk-london-1.oraclecloud.com/n/lrsrfayerklw/b/baumeister-test-bucket,src_data/customer.csv”

```
s3://<bucket>/prefix
```

uri

Steps to query data in object store

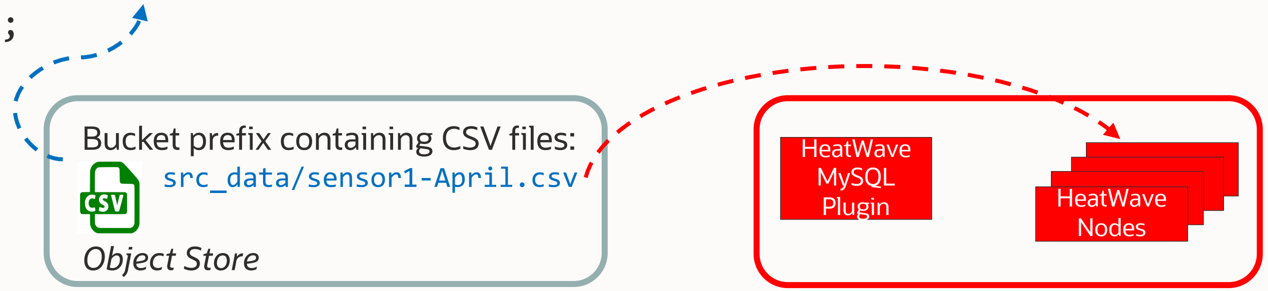
Fully compatible SQL syntax

1. Run MySQL Autopilot:

```
mysql> CALL sys.heatwave_load(@db_list, @options);
```

2. Execute generated DDLs:

```
mysql> CREATE TABLE `cust1DB`.`Sensor` (date DATE, degree INT) ENGINE=LAKEHOUSE SECONDARY_ENGINE=RAPID
    -> ENGINE_ATTRIBUTE = '{"file": [{"prefix": ”src_data/sensor1-April.csv", "par": "<PAR URL>"}]}';
mysql> ALTER TABLE `cust1DB`.`Sensor` SECONDARY_LOAD;
```




```
mysql> ALTER TABLE SALES SECONDARY_LOAD;
```

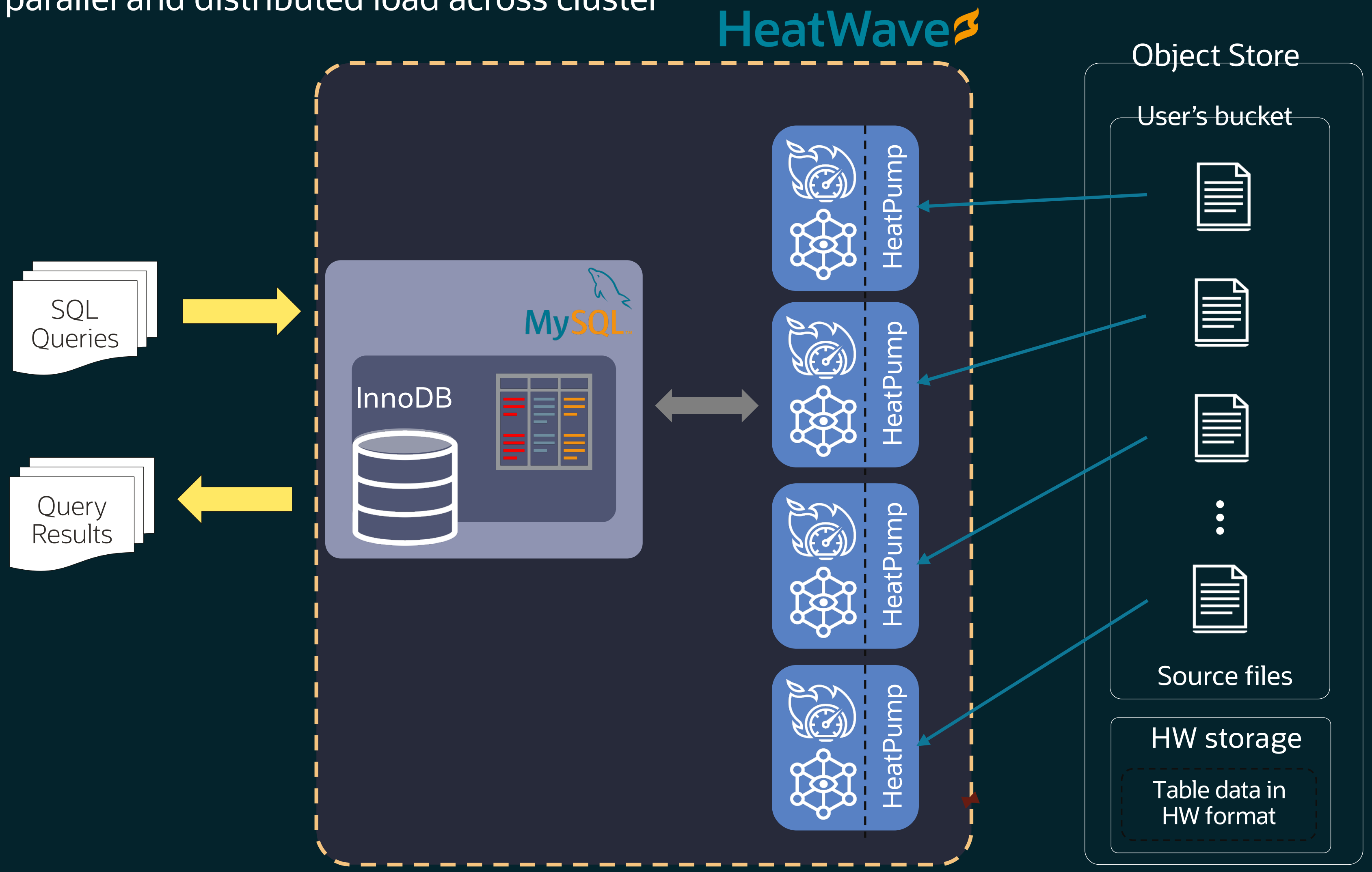
3. Execute Query:

```
mysql> SELECT count(*) FROM Sensor, SALES WHERE Sensor.degrees > 30 and Sensor.date = SALES.date;
```

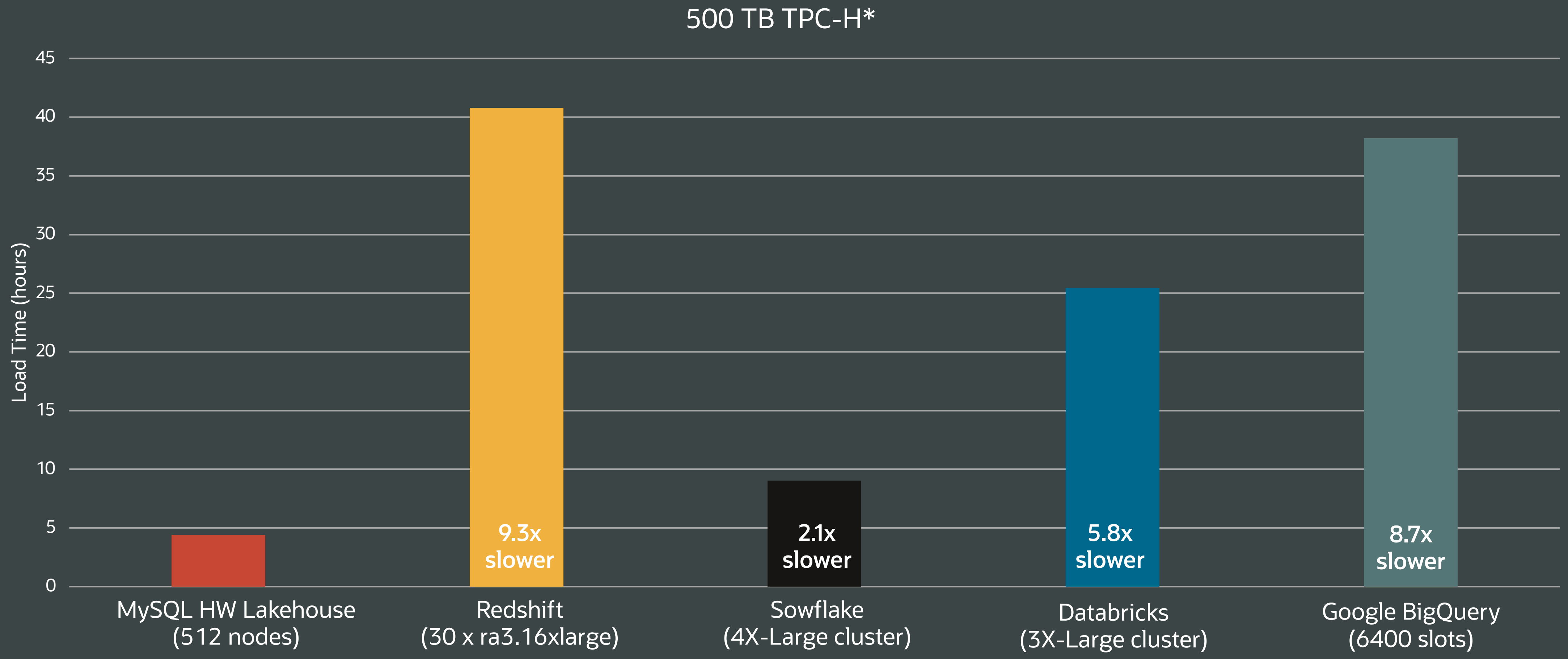
HeatWave Lakehouse: Scale-out load

Ingest and query data from data lakes in object store

Completely parallel and distributed load across cluster



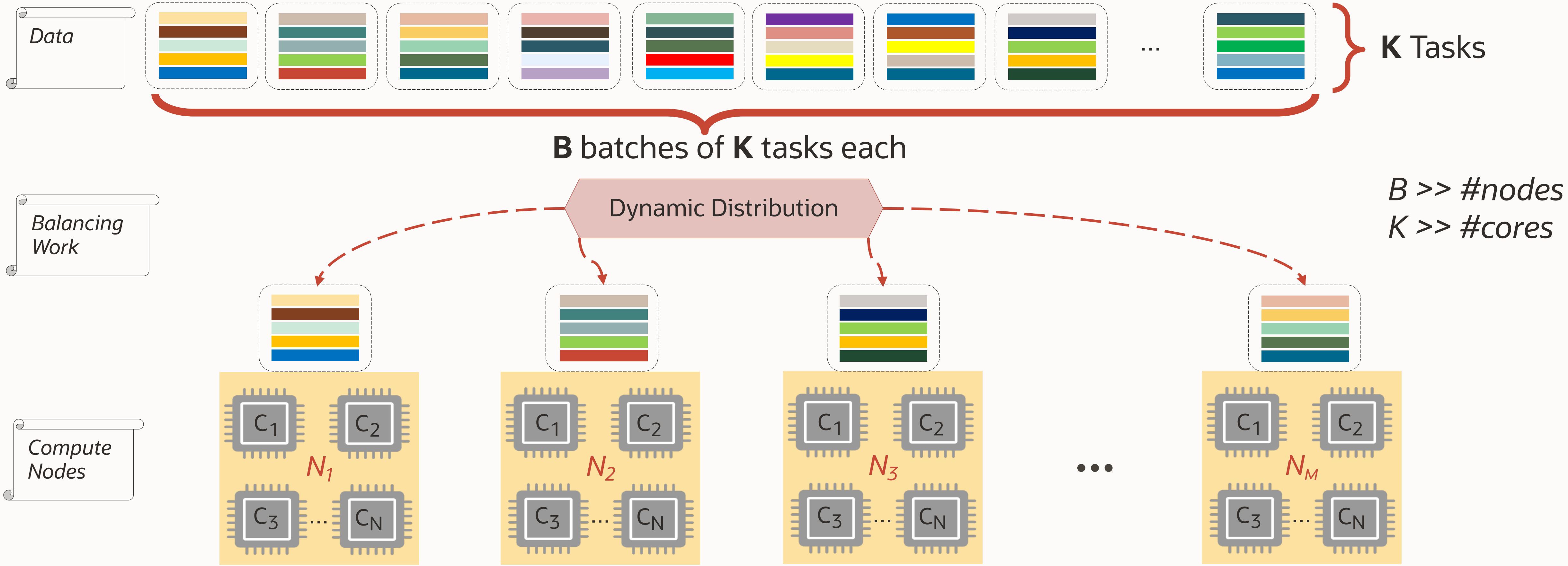
HeatWave Lakehouse scales all the way to 500 TB



*Benchmark data are derived from TPC-H benchmarks, but results are not comparable to published TPC-H benchmark results since these do not comply with TPC-H specifications

Scaling Load Performance

SuperChunking*: Dynamic task distribution technique to balance work across nodes & cores



The load processing scales and is balanced across the nodes

* patent in-progress

Lakehouse source data can various file formats

Key innovations maintain parity across file formats

	CSV	Parquet	Avro	DB Exports
Properties	Row major text format	Hybrid-columnar fmt	Row major binary fmt	Vary based on fmt
Data Load	Speculative distributed parsing	Distribution of work at row-group granularity	Distribution of work at block granularity	Depends on export format of the system (Redshift, Aurora, MySQL)
	Aggregation and validation of speculation with high success rate	Incremental processing and caching of column chunks	Parallelization by speculation for sync markers across blocks	Techniques apply based on format
Data Query	Converted into internal HeatWave format delivering identical performance			
Feature Set	Extensive configurability through dialect options	Support any practical row-group size	Support for blocks as large as 64 MB	Features apply based on format

Load and Query performance across file formats

100 TB TPCH	CSV	Parquet	Avro
Configuration	100 Nodes	100 Nodes	100 Nodes
Load Time (hrs)	2.8	2.8	2.74
Geomean Query Time (sec)	26.33	26.23	26.85
Total Query Time (sec)	921	920	933

Lakehouse support for semi-structured data

- JSON data in CSV, Parquet, and Avro file formats can also be processed by HeatWave

- Support extended to newline-delimited JSON files
  - Ease of parsing and streaming has made it the most popular JSON format

- NDJSON data ingestion and processing scales similarly to structured file formats

```
…
{ “name”: “Jane”, “academics”: { "undergraduate": "MIT", "graduate": "UT Austin” }, "age": 24 }
{ “name”: “Jill”, “academics”: { "undergraduate": ”Madison", "graduate": ”Stanford” }, "age": 27 }
…
```

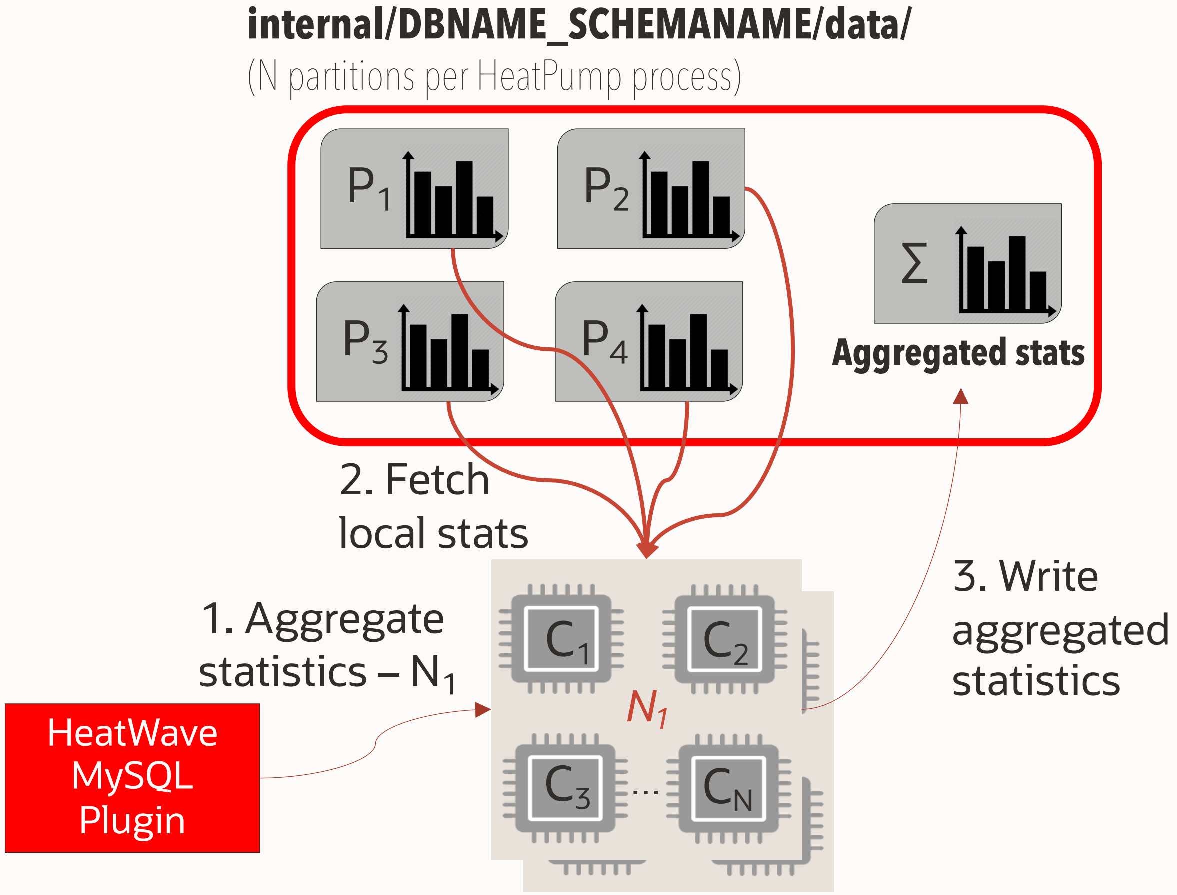
Example NDJSON file

Statistics is key for query performance

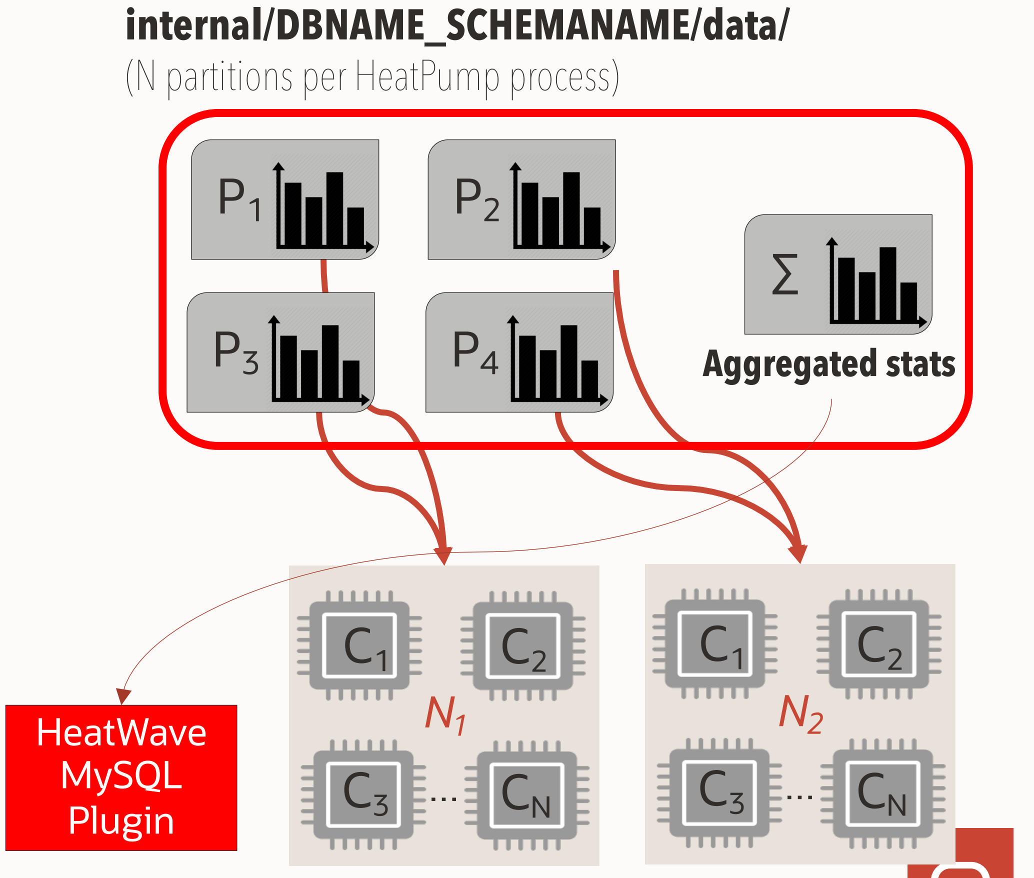
Lakehouse over object store performance within 5% of in-DB OLAP performance

1. Local statistics computed on-the-fly during transform

2. Statistics aggregated

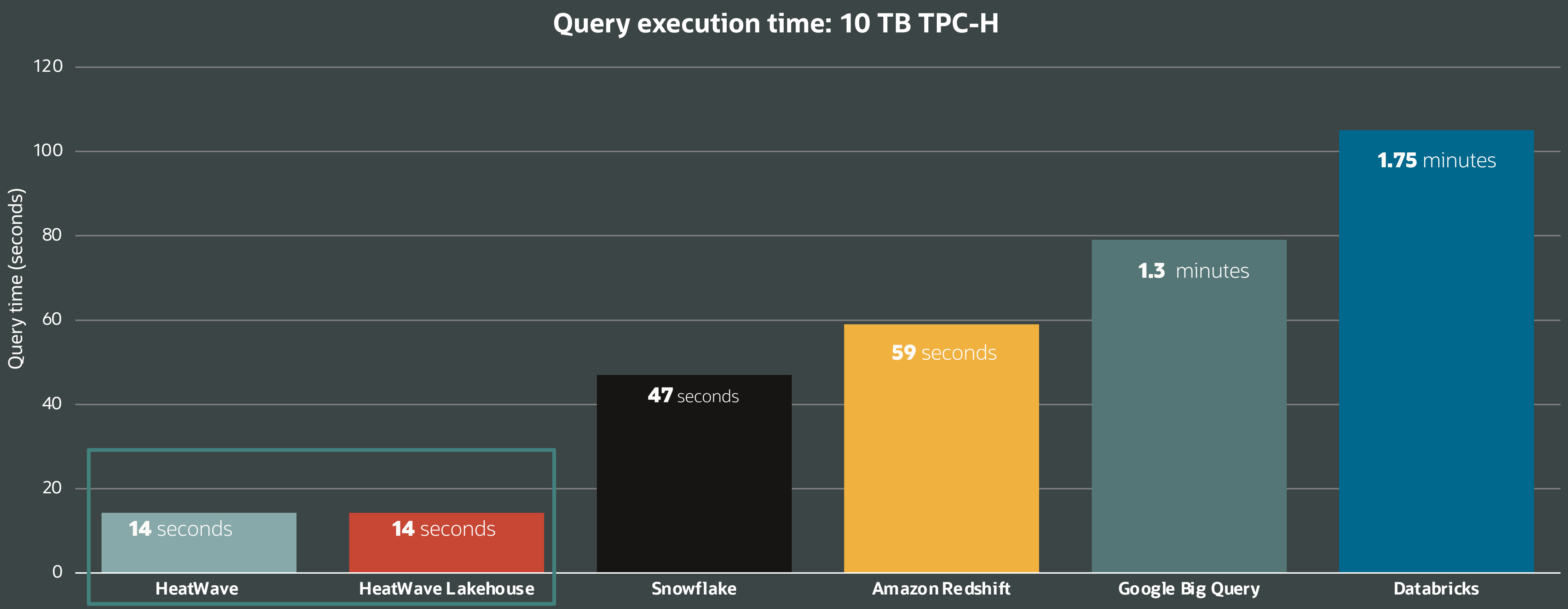


3. Statistics & data available for queries



Same performance for data in DB or in object store

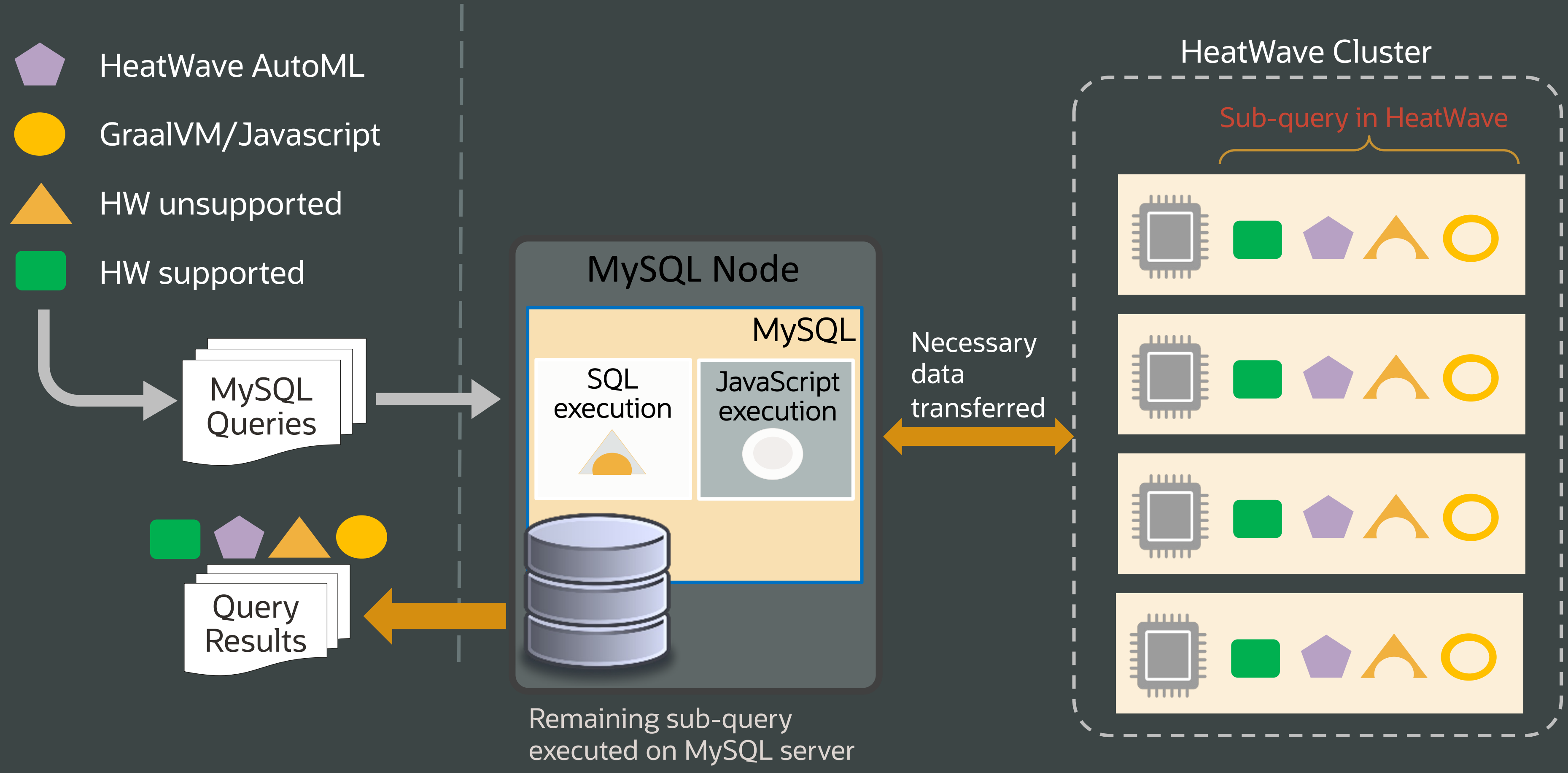
Develop applications with data on object store without any performance impact



Configuration: MySQL HeatWave Lakehouse: 512 nodes; Snowflake: 4X-Large Cluster; Databricks: 3X-Large Cluster; Amazon Redshift: 20-ra3.16xlarge; Google BigQuery: 6400 slots
Benchmark queries are derived from the TPC-H benchmarks, but results are not comparable to published TPC-H benchmark results since these do not comply with the TPC-H specifications.

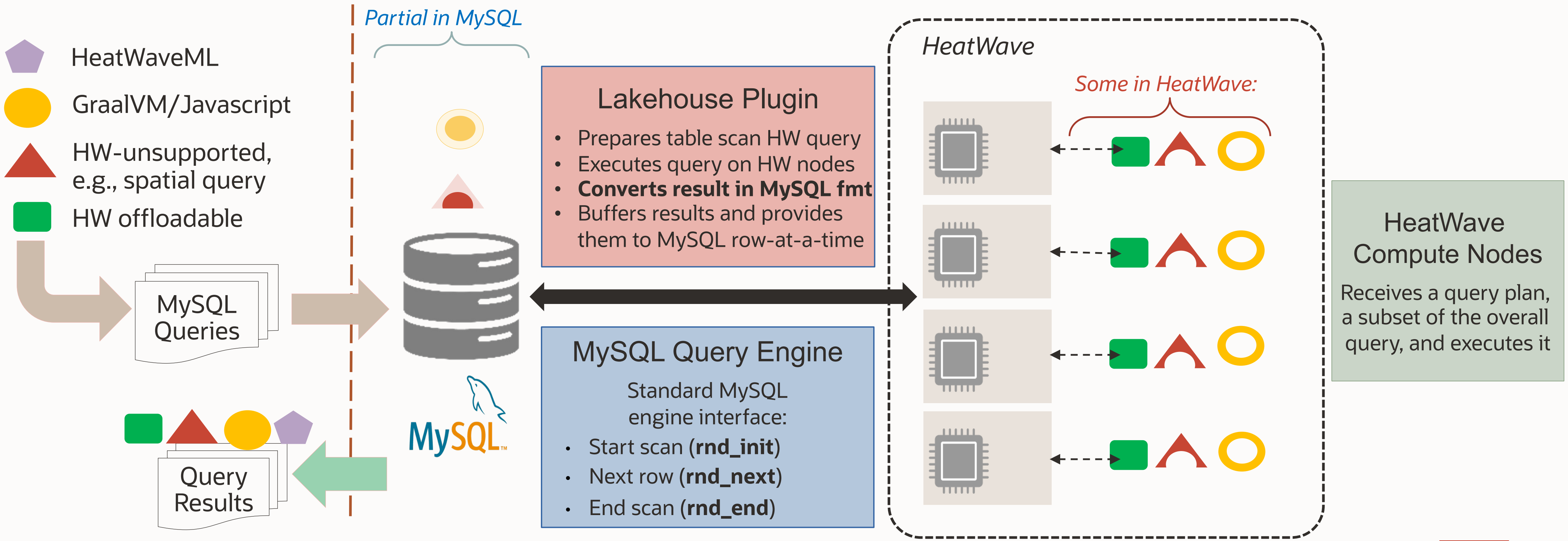
Partial query execution in HeatWave for data in object store

Execute part of the query in HeatWave, rest in MySQL

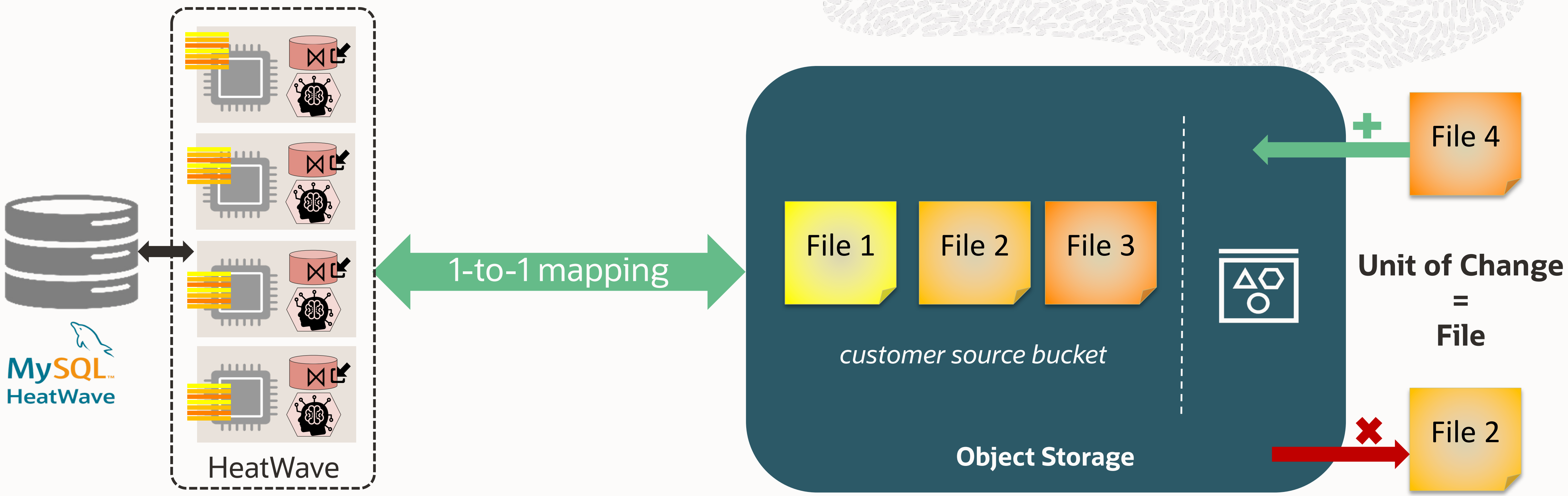


How do we achieve partial query execution with HeatWave Lakehouse

- When "*partial-execution*" is needed, the Lakehouse engine provides access to Lakehouse tables via standard MySQL storage engine interfaces .



Incremental data load in Lakehouse tables

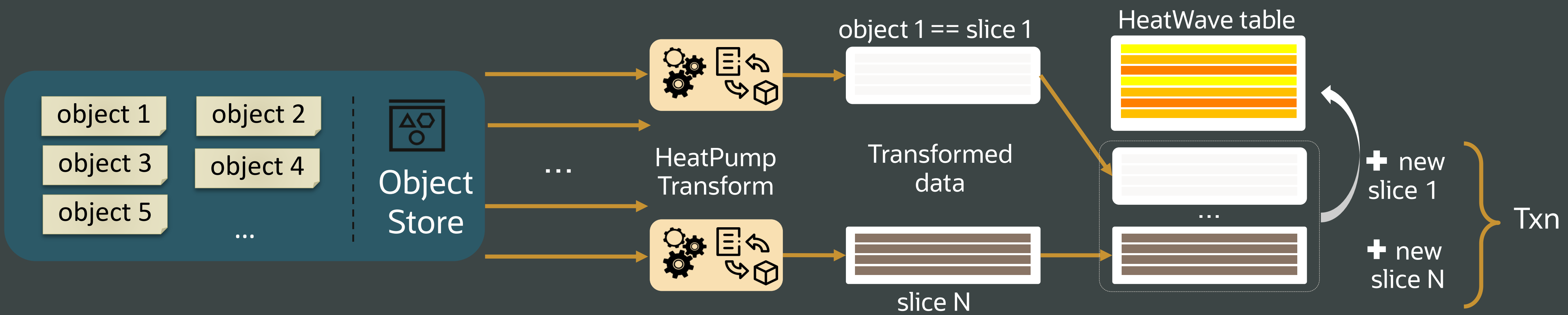


- **Feature**: Lakehouse table data is updated to reflect modifications in user data
  - Provides 1-to-1 mapping between user data and Lakehouse table data at any point in time
  - Only delta in user data is applied incrementally over existing table data

```
SET @options = JSON_OBJECT('mode', 'normal',
                           'refresh_external_tables', TRUE);
SET @heatwave_debug_output = TRUE;
CALL sys.heatwave_load(@db_list, @options);
```

Incremental data load in Lakehouse tables

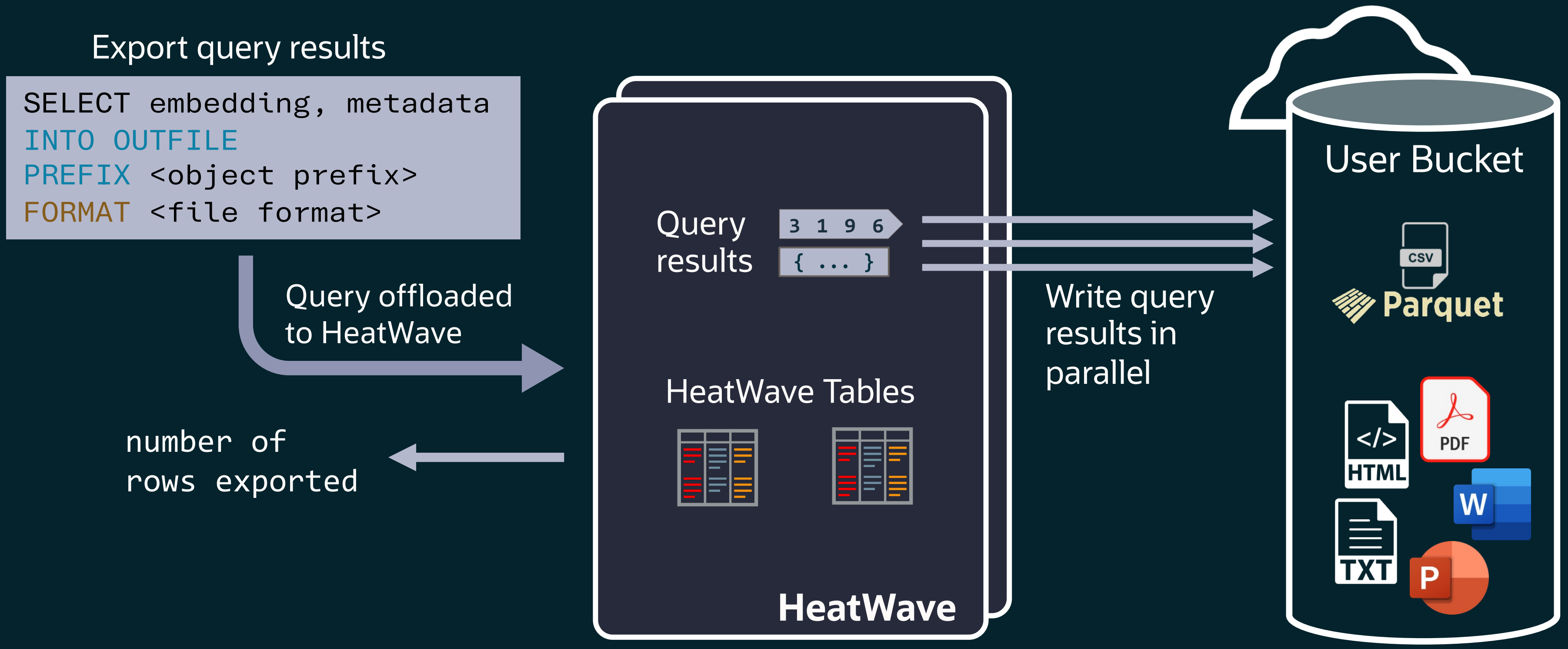
Scale-out delta ingestion



- **Granularity of data update** is an object corresponding to thousands of records
- **User data change detection**: On user-initiated SQL command, user data change is detected
  - Objects in user buckets can be **added**, **deleted**, or **updated**
  - Delta computed comparing current list of objects with the list from the last table load or incremental load
- **Delta apply design**: Treat each object as a new horizontal slice of the table
  - Objects added or updated are transformed and ingested in a scale-out manner across HeatWave cluster like table load
  - Bulk-inserts scale: HeatPump parallelism at inter-file & intra-file levels
  - Objects deleted – fast in-memory operation of dropping a table slice by updating table version

Export transformed data to Object Store

Massively parallel write to object store with the multi-node & multi-core parallelism of HeatWave



MySQL Autopilot - Auto Parallel Load in Action

Automatically generated schema from data in files by AutoPilot

```
# Load Script
CREATE DATABASE `tpch_500T`

CREATE TABLE `tpch_500T`.`lineitem`
(
    `col_1` int unsigned NOT NULL,          `col_2` mediumint unsigned NOT NULL,
    `col_3` mediumint unsigned NOT NULL,    `col_4` tinyint unsigned NOT NULL,
    `col_5` tinyint unsigned NOT NULL,      `col_6` decimal(8,2) NOT NULL,
    `col_7` decimal(3,2) NOT NULL,          `col_8` decimal(3,2) NOT NULL,
    `col_9` varchar(1) NOT,                 `col_10` varchar(1) NOT NULL,
    `col_11` date NOT NULL,                 `col_12` date NOT NULL,
    `col_13` date NOT NULL,                 `col_14` varchar(17),
    `col_15` varchar(7),                    `col_16` varchar(43),
    `col_17` varchar(0)
)

ENGINE=lakehouse
SECONDARY_ENGINE=RAPID
ENGINE_ATTRIBUTE='{"file":
    [{"name": "lineitem.tbl", "bucket": "tpch_500T", "region": "us-ashburn-1", "namespace": "mysql"}],
    "dialect": {"format": "csv", "field_delimiter": "|", "record_delimiter": "\\n"}}';

ALTER TABLE `tpch_500T`.`lineitem` SECONDARY_LOAD;
```

DDL to create non-existing DBs

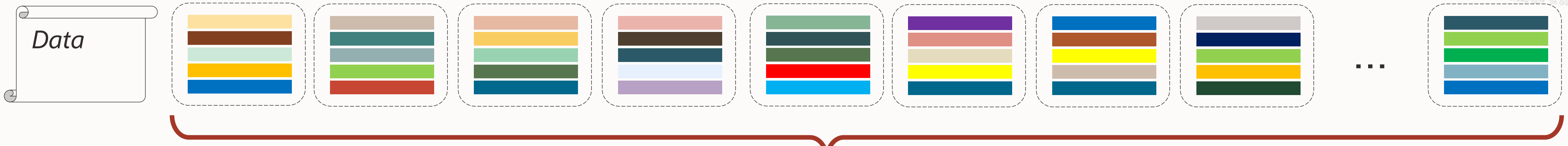
DDL to create non-existing tables

- Using inferred column types
    - Length
    - Precision
- Setting engines
- Setting engine attribute
- Can extract column names

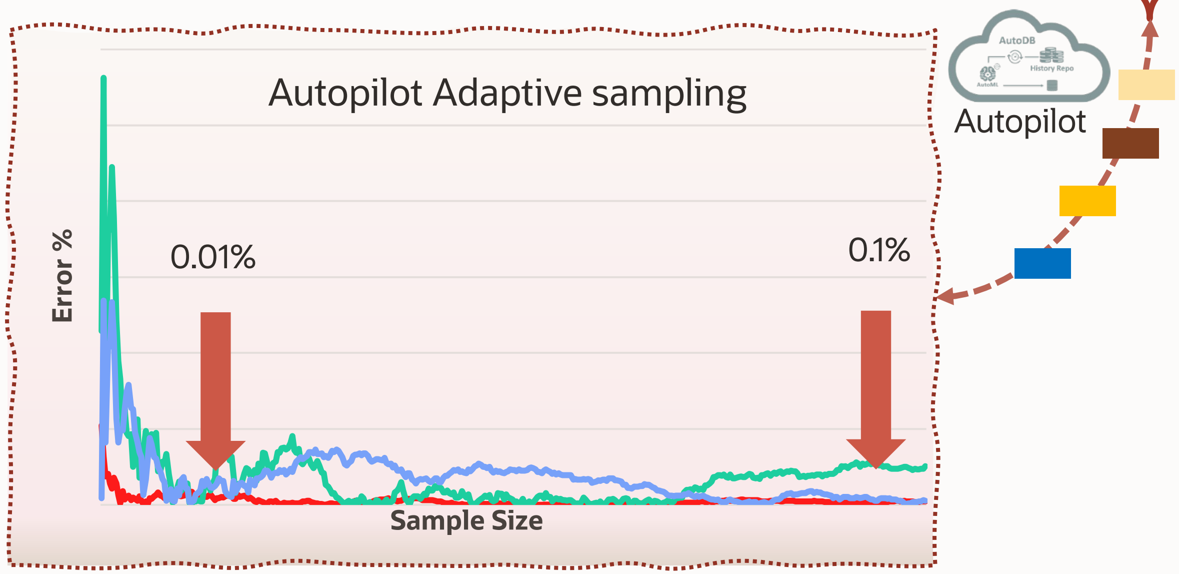
Load command

Scale out Autopilot with adaptive sampling

SAMPLE DATA UNTIL COLLECTED STATISTICS STABILIZE



Each color carries different statistics



TPCH Data set (TB)	Lineitem Autopilot schema inference time(s)
1 TB	8s
10 TB	8s
30 TB	13s
100 TB	15s
250 TB	25s
500 TB	47s

- Autopilot sampled processing is scaled out and balanced across the nodes, similar to actual load.
- If the data set is relatively uniform, a single node is enough to process 100s of TBs of data.

Native Vector Processing in MySQL HeatWave

Vector Datatype

- MySQL & HeatWave supports new Vector data type
- In-memory hybrid-columnar storage format for vector columns

Vector as first-class data type

```
mysql> CREATE TABLE wikipedia (
           title VARCHAR(1024),
           page_data TEXT,
           page_url TEXT,
           page_embedding VECTOR(1024));
```

Vector Processing

- Leverage SIMD instructions for vector processing
- Processes at near memory bandwidth

MySQL query syntax

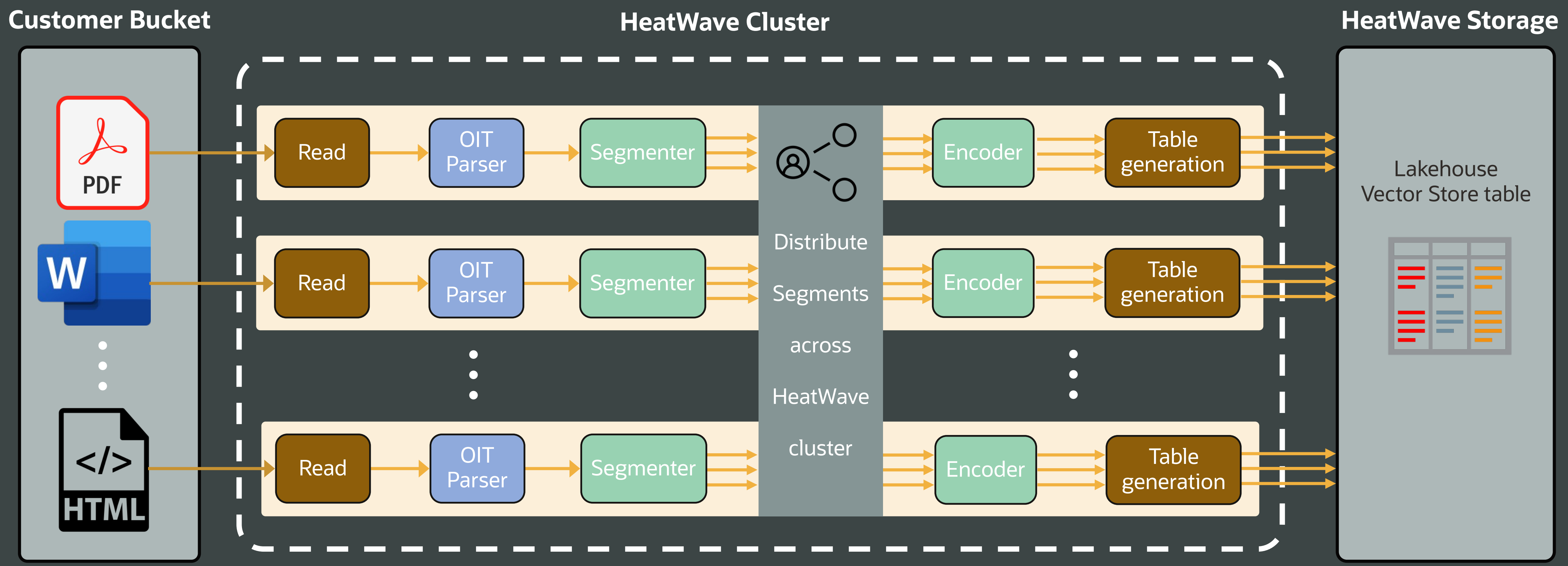
```
mysql> SELECT page_url,
       DISTANCE(page_embedding,
                @query_embedding, “COSINE")
                as distance
       FROM wikipedia
       ORDER by distance DESC LIMIT 10;
```

New distance function for similarity search

- L1/MANHATAN
- L2/EUCLIDIAN
- L1^2/MANHATAN_SQUARED
- L2^2/EUCLIDIAN_SQUARED
- COSINE
- DOT
- HAMMING

Scale out Vector Store creation with HeatWave Lakehouse

Parse source files with OutsideIn (OIT) and concurrent embedding generation across nodes



Conclusions

- HeatWave Lakehouse is part of a converged single system for OLTP, SQL Analytics, ML, Data Lake and Vector store

- The performance and cost advantages of the HeatWave analytics system is expanded for massive amounts of data (up to $^1/_2$ PBs of data, 512-nodes)
  - Data load performance is already ahead of the competition
  - Query performance is at par with HeatWave Data Warehouse

- Provides important differentiation from the competition
  - Query support across OLTP / Data Warehouse & Data Lake
  - Automatic schema inference for exploratory analysis
  - ML-based automation features via Autopilot
  - HeatWave AutoML & Vector Processing on object store resident files

Thank you