Dream the Stream

High Velocity Event Processing with a Converged Database

Shasank Chavan

Vice President, In-Memory Data Technologies

University of Wisconsin, Madison

October 24, 2022

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.

Agenda

What is Event Stream Processing

Converged vs. Specialized Databases

What does an Event Stream Processing Database Need

Demo Slides – DevOps Monitoring

What is Event Stream Processing

What is Event Stream Processing?

Database Use Case Characterized By

Continuous Ingestion

Continuous ingest of high frequency event data

Real-time Analytics

Unlike batch processing, event processing analytics is performed on data in motion

Data Reorganization

Event data is increasingly compressed and summarized as it ages before finally comes to rest as archived data

What is Event Data?

- Events are discrete data records generated by large farms of data sources
- Data sources are extremely diverse
  - Devices, sensors, meters, servers, desktops, smartphones
- An event typically includes the following information:

Time: 6/16/21:12:05pm
Meter ID: X45-123
Reading: {
Electricity KW-hrs: 0.4
Water Gallons: 5
Gas therms: 2
}

Time: 6/11/21:12:12pm
Phone ID: 1955ABC
Reading: {
Location: 37.6N/112.2W
Battery Level: 60%
}

Time: 6/17/21:1:0pm
Vehicle ID: WBG6108
Reading: {
Location: 37.6N/112.2W
Speed: 66.7 mph
Direction: 120.5 degrees
}

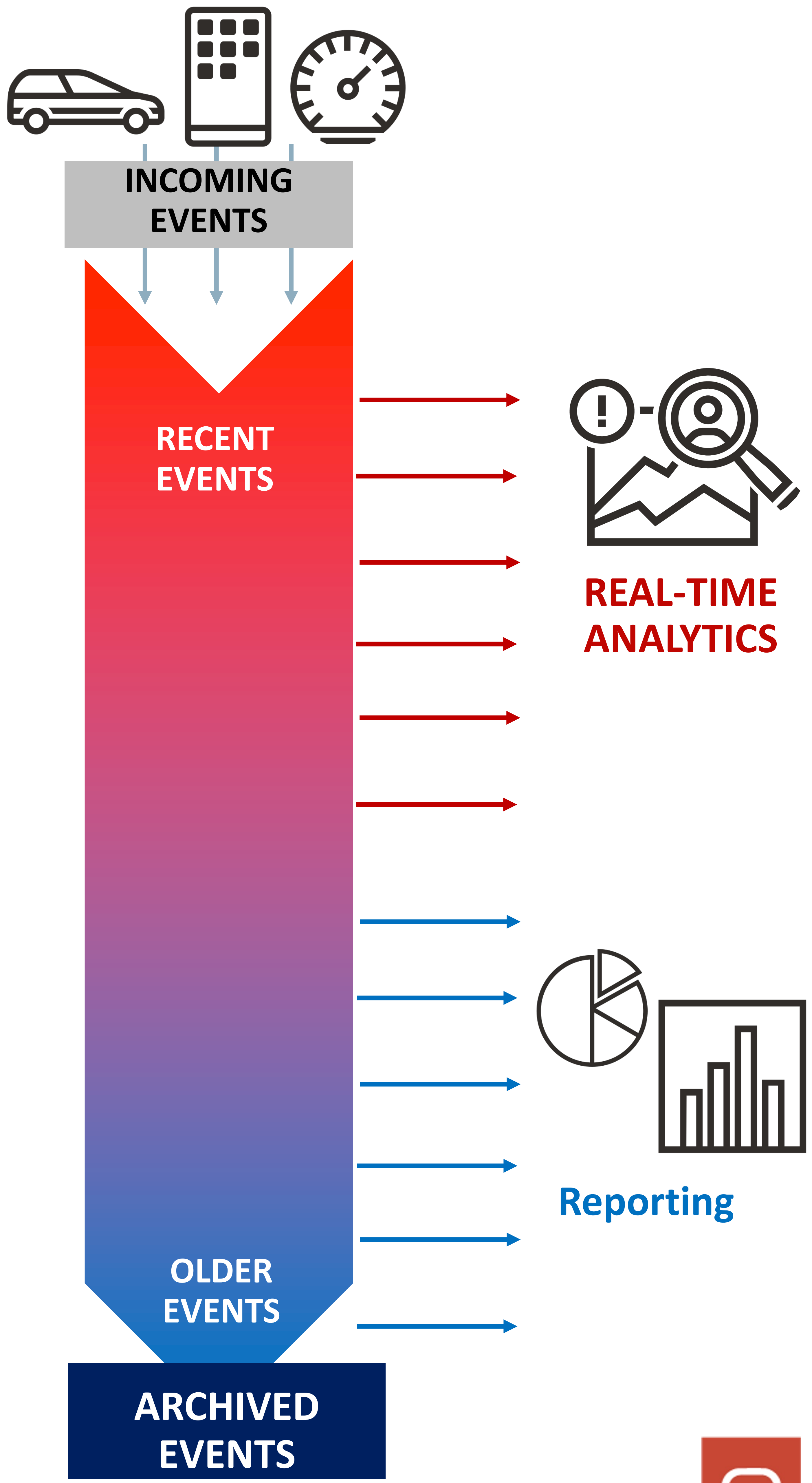
Properties of Event Data

High Arrival Rate

- Most event processing systems receive large numbers of events from many different sources
  - E.g. Billing systems receives millions of smart meter readings every few minutes.

High Obsolescence Rate

- Recent events are frequently queried for real time analytics while old events are used for historical reporting
- Events are often compressed and summarized at greater and greater levels of data and space reduction as they age
  - E.g. Per minute readings from smart meters converted to hourly summaries after a day and converted to daily summaries after a month



Requirements of an Event Stream Processing System

A

Flexible data model
Needs to handle heterogeneous event sources
For example, a new device type added to home network

B

High Speed Ingest
Must be able to sustain billions of events per day

C

Rich Analytic Query Capability
Requires advanced analytic functionality to filter, aggregate and summarize across moving windows of event stream data

D

Real-Time Analytics
Instantaneous reporting of timely actions on events
For example, detecting and reporting fraud, fire, leaks, etc.

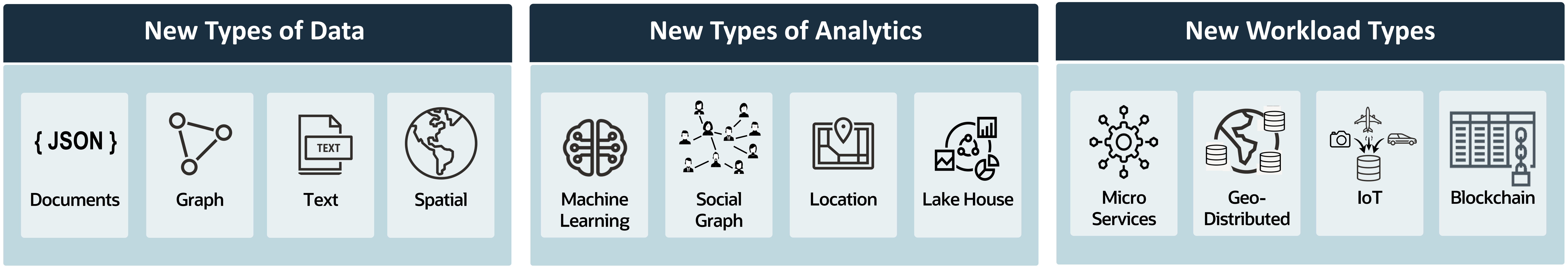
E

Automatic Data Lifecycle Management
Automatic data compression, summarization, archiving needed to avoid unbounded data growth

Converged vs. Specialized DBs

The Increasing Complexity of Modern Apps

- Modern Apps use a **new generation of data technologies:**



- Developing and running modern apps across these many engines become increasingly complicated – **bugs, security, upgrades, downtimes,** etc.

The Increasing Complexity of Modern Apps

- One approach to building modern apps is to use a specialized database for each application need
- Each specialized database excels at one aspect of the app's requirements

New Types of Data

{ JSON } Documents	Graph	TEXT Text	Spatial
MongoDB Documents	Neo4j Graph	Elastic Search	PostGIS Spatial

New Types of Analytics

Machine Learning	Social Graph	Location	Lake House
Spark MACHINE LEARNING Machine Learning	ArangoDB Social Graph	esri Location	Snowflake Lake House

New Workload Types

Micro Services	Geo-Distributed	IoT	Blockchain
kafka Micro Services	cassandra Geo-Distributed	DynamoDB IoT	AWS QLDB Blockchain

The Problems With Using Specialized Databases

- However, this approach inherently creates an application architecture that is heterogeneous and distributed
- Built from many moving parts that must be learned, synchronized, secured, maintained, and governed
- Fragments the data and app, which makes app dev more complex, and compromises security and QoS

New Types of Data

- MongoDB Documents
- Neo4j Graph
- Elastic Search
- PostGIS Spatial

New Types of Analytics

- Spark Machine Learning Machine Learning
- ArangoDB Social Graph
- esri Location
- Snowflake Lake House

New Workload Types

- kafka Micro Services
- cassandra Geo-Distributed
- DynamoDB IoT
- AWS QLDB Blockchain

- Specialized databases also provide limited ACID consistency requiring developers to code app level consistency
- Building apps using specialized databases forces developers to spend their time integrating instead of innovating

Event Stream Processing Solutions

influxdb

druid

ORACLE
Stream Analytics

TIMESCALE

Prometheus

Db2 Event Store
A complete solution for Fast Data
IBM

graphite

APACHE kafka

Spark Streaming

OPENTSDB

Amazon Timestream

amazon Kinesis

CONFLUENT

Flink

Oracle Converged Database

- Oracle is a Converged Database
  - Native support for all modern data types and the latest development paradigms built into one product
- New data management technologies are often implemented as separate products
  - With a converged database, you don't need to manage and maintain multiple systems
  - No need to worry about having to provide unified security across them.
- A good analogy is a smartphone
  - In the past, separate phone, camera, video recorder, gps, music device

What Does an Event Stream Processing Database Need

Flexible Data Model

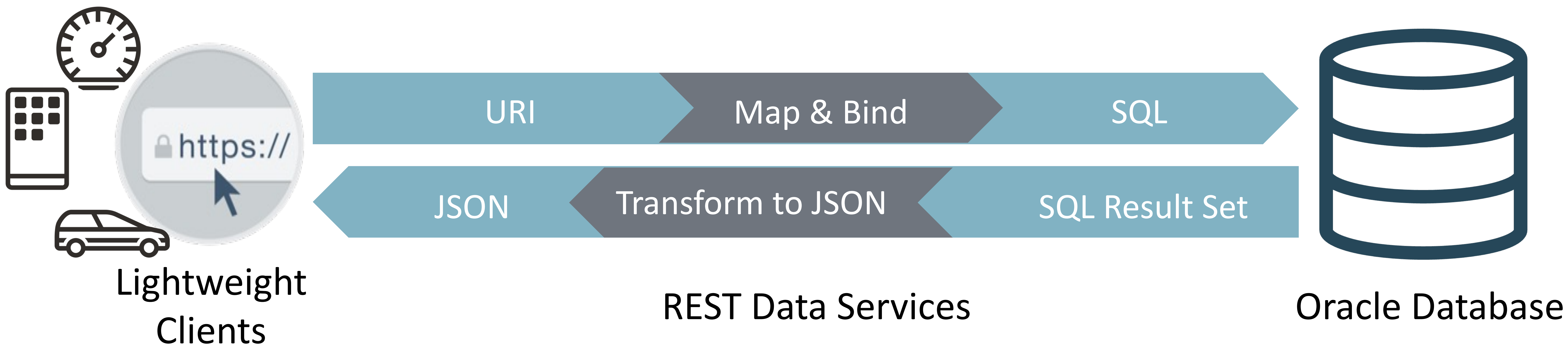
Why is JSON Necessary for Event Stream Processing


```
{
   "Time": "6/16/21:12:05:12"
   "HomeID": "BZ125"
   "Readings:" {
       "KWH":"0.4"
               }
}
```


Seq#	MeterID	Time	Readings
1005	BZ125	61621:1205:00	{“KWH”:”0.8”}
1006	BZ78	61621:1210:04	(“Gas Therms”:”1”}
1007	BZ123	61621 1215:12	{“Gallons”:”51”}

- Event Stream data is highly dynamic
- Formats can change constantly: between readings; after software update, after new type of device is added
- JSON allows applications to easily adapt to changes in data formats. e.g.:
  - The fixed part of this meter evet data (Meter ID, timestamp) could be stored in relational columns while the variable Readings data could be stored as JSON

Oracle REST Data Services

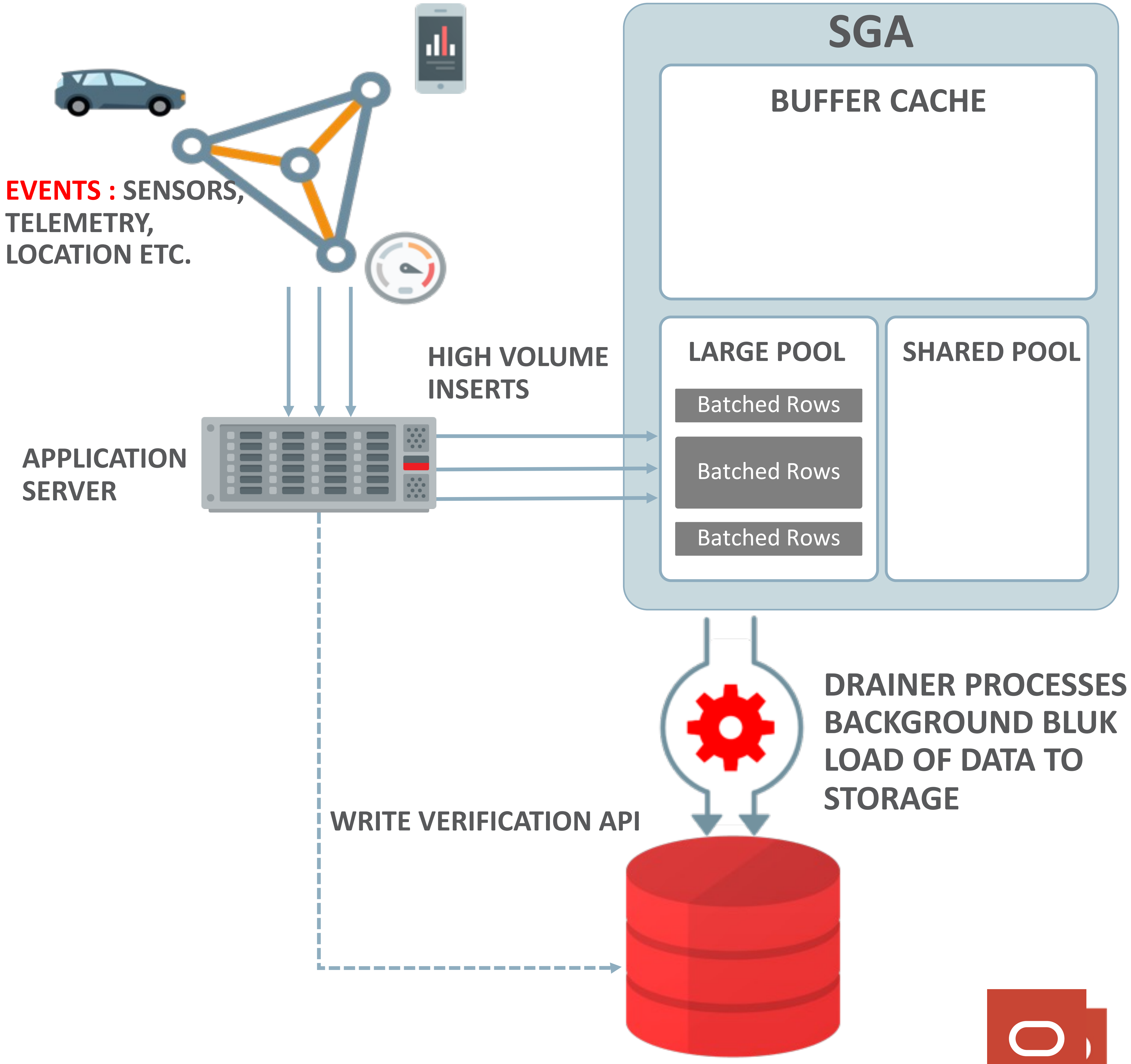


- REST is the ideal protocol for Event Stream Processing Ingest from lightweight clients
- Oracle Rest Data Services automatically generates REST endpoints for SQL statements
- Transforms SQL results into JSON or other formats (CSV, etc.)
- REST is stateless, all INSERTs/UPDATEs/DELETEs are auto-committed
- Applications access data like any other service via a REST API
- Simplifies and standardizes APIs to access data

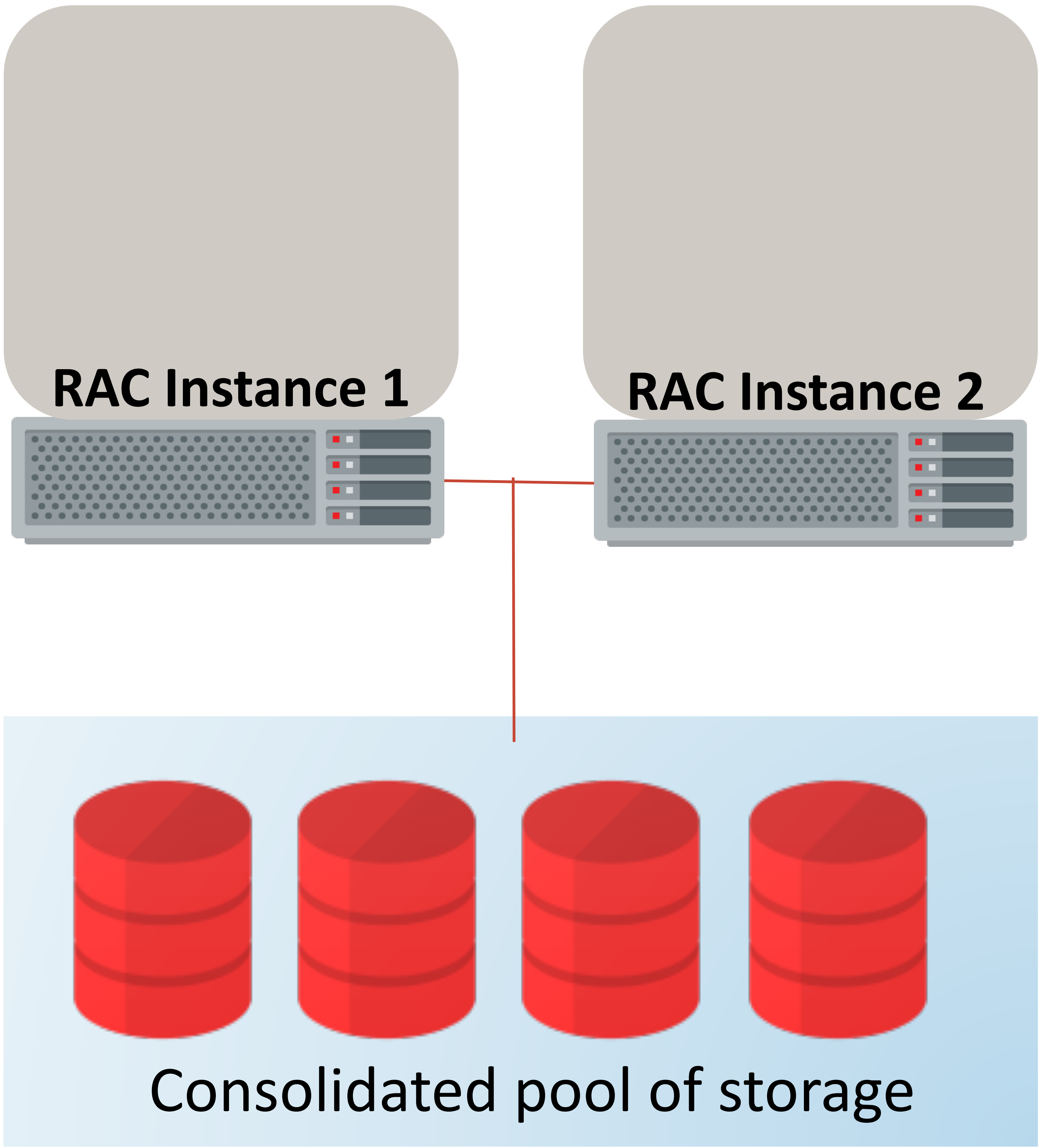
High-Speed Ingestion

Memoptimized Rowstore | Fast Ingest at the DB Tier

- A memory optimised mechanism for inserting data into the database
- Ideal for ingesting light weight events
- Event rows are buffered in memory and asynchronously drained to disk
- An API allows developers to check on the durability of their inserts
- Ultra-fast - 25 million inserts per second or 21 trillion per day on two socket server

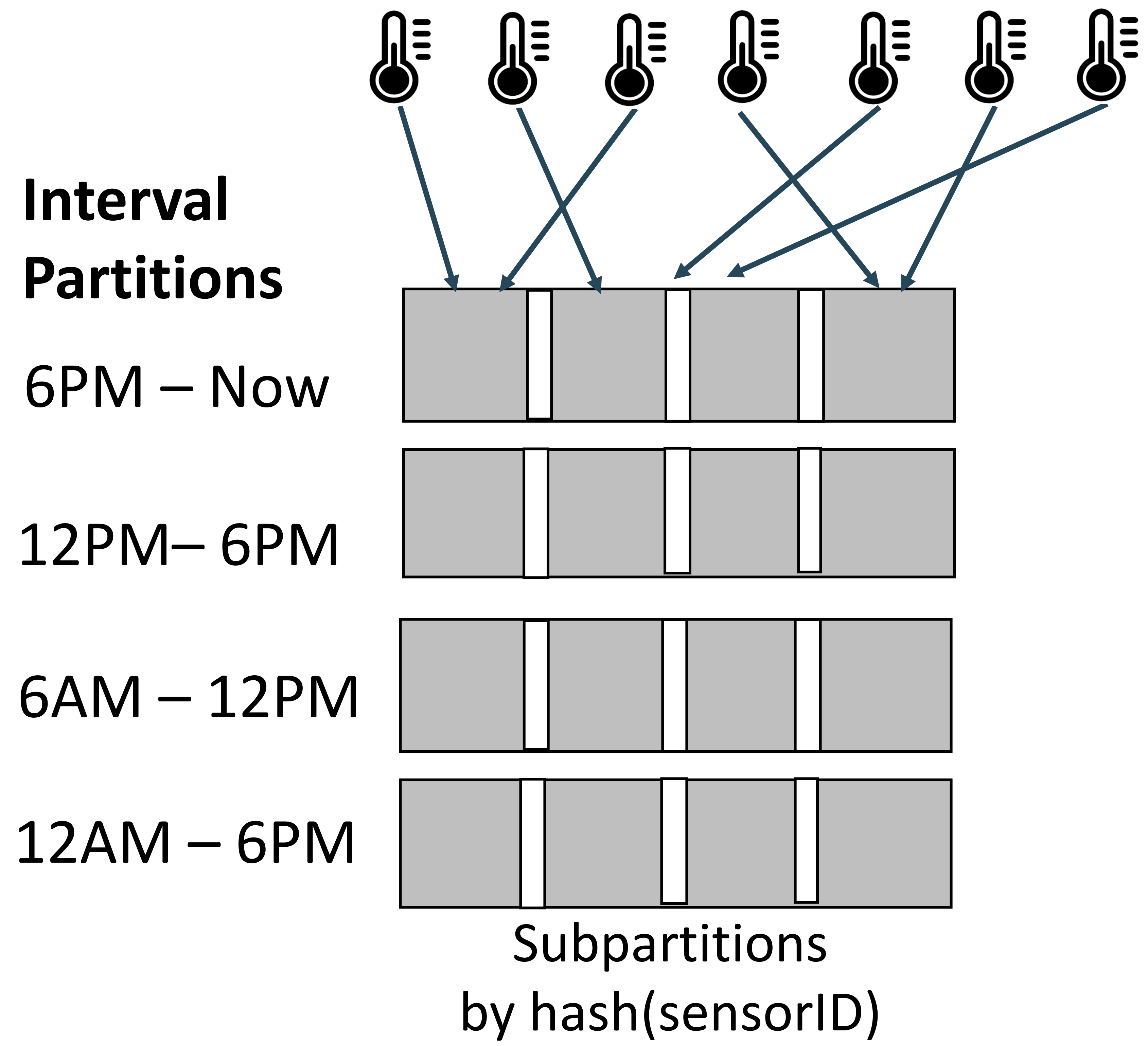


Real Application Clusters: Industry-Leading Scale-Out Compute



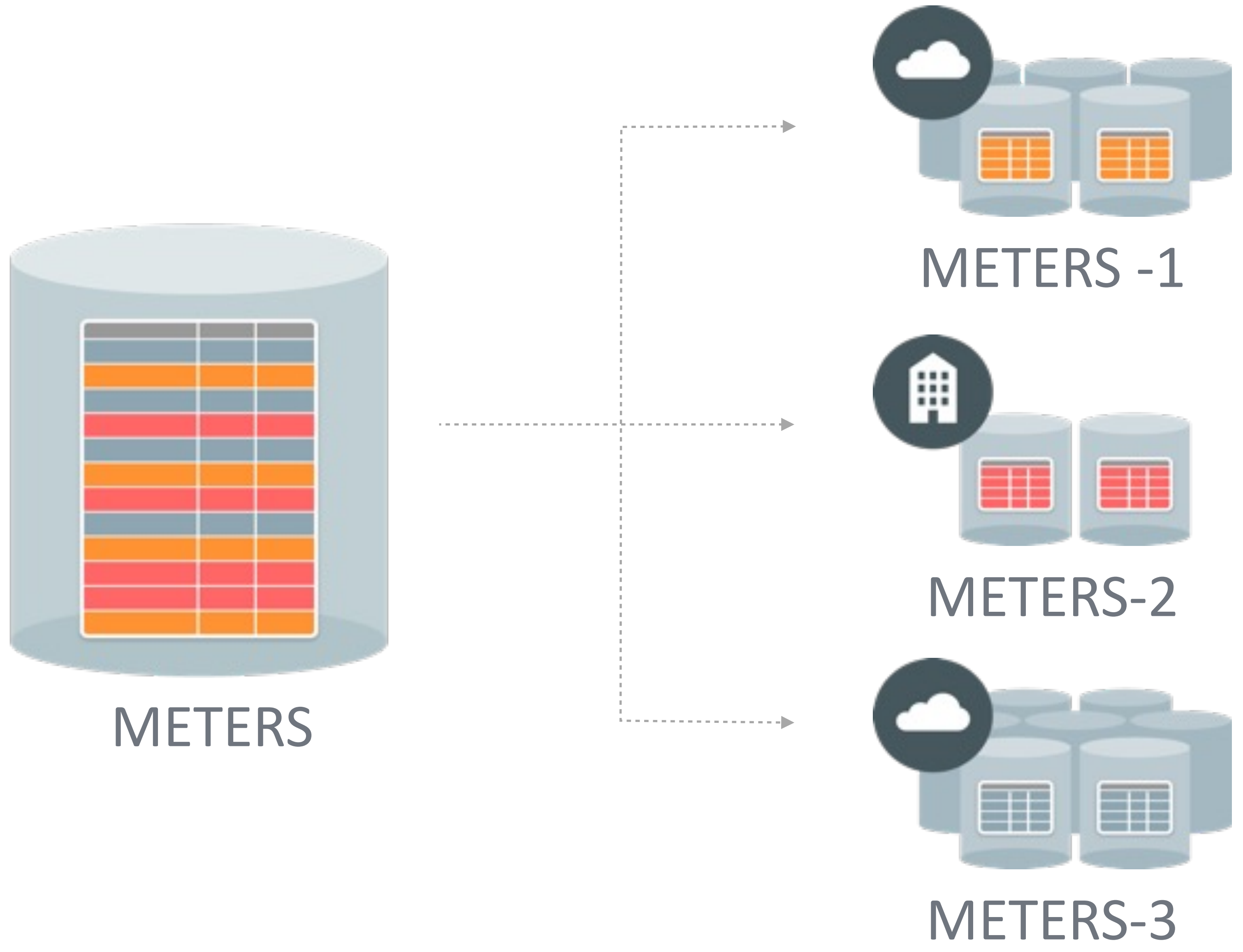
- Transparently scales out a database across a pool of hosts sharing the same storage pool
  - Only scale-out technology capable of running the world's most complex enterprise workloads
- **Scales performance**: more hosts imply more throughput
  - When more throughput is required, simply add a new host
- **Scales fault tolerance**: more hosts imply more availability
  - When a host goes down, the database remains available
- **Scales user experience**: Constant latency as system grows
- **Scales administration**: Larger clusters no harder to manage than smaller clusters, online upgrades and patching

Partitioning: Efficiency and Parallelism for Event Streams



- Partitioning avoids single table insert scaling limitations (E.g. Contention for space allocation)
- Divides large tables into multiple units for scalable ingest
  - Many different partitioning schemes exist for partitioning and for sub-partitioning
  - Event stream data typically **partitioned by time interval** and **sub-partitioned** by hash of source ID
- Very important scalability mechanism for event streams:
  - Sub-partitioning by source speeds up ingest
  - Partitioning by time reduces data access for analytics

Sharding: Globally Distributed Database Architecture

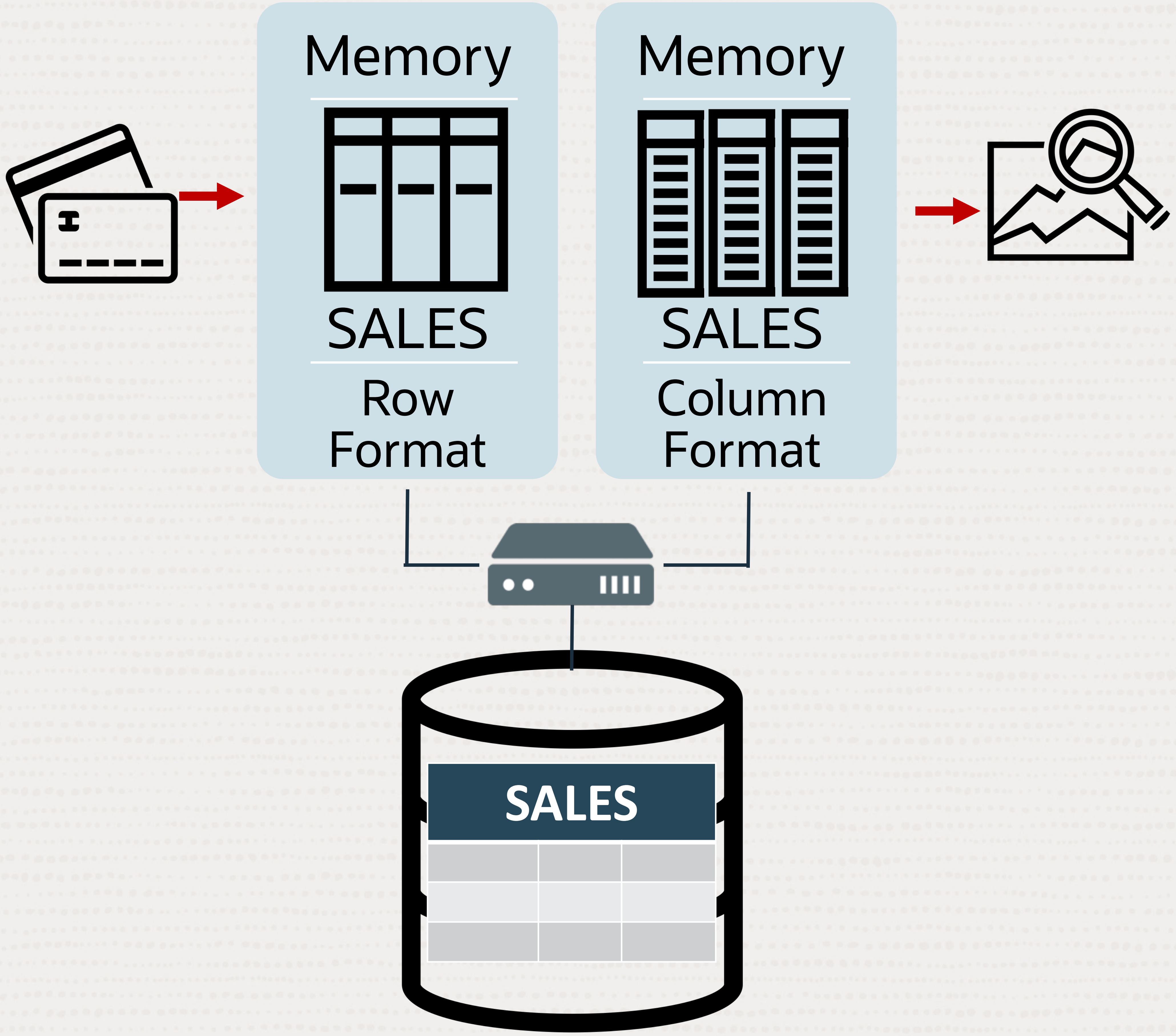


One giant database divided into several smaller databases (shards)

- Global-Scale applications may prefer to divide massive databases into a farm of smaller databases known as shards
  - Avoids scalability or availability issues with very large databases
  - Each shard can be replicated via Data Guard or Golden Gate
- Native SQL for sharding tables across up to 1000 Shards
  - Routing of SQL based on shard key, and cross shard queries
  - Online addition and reorganization of shards
- Sharding is the **Ultimate** scalability mechanism
  - Linear scalability of capacity, throughput, user population
  - Improves availability since shards are fault isolated
  - Scales user experience since shards are performance isolated
  - Scales administration since built-in shard management tools make managing of 100s of shards as simple as managing a single database

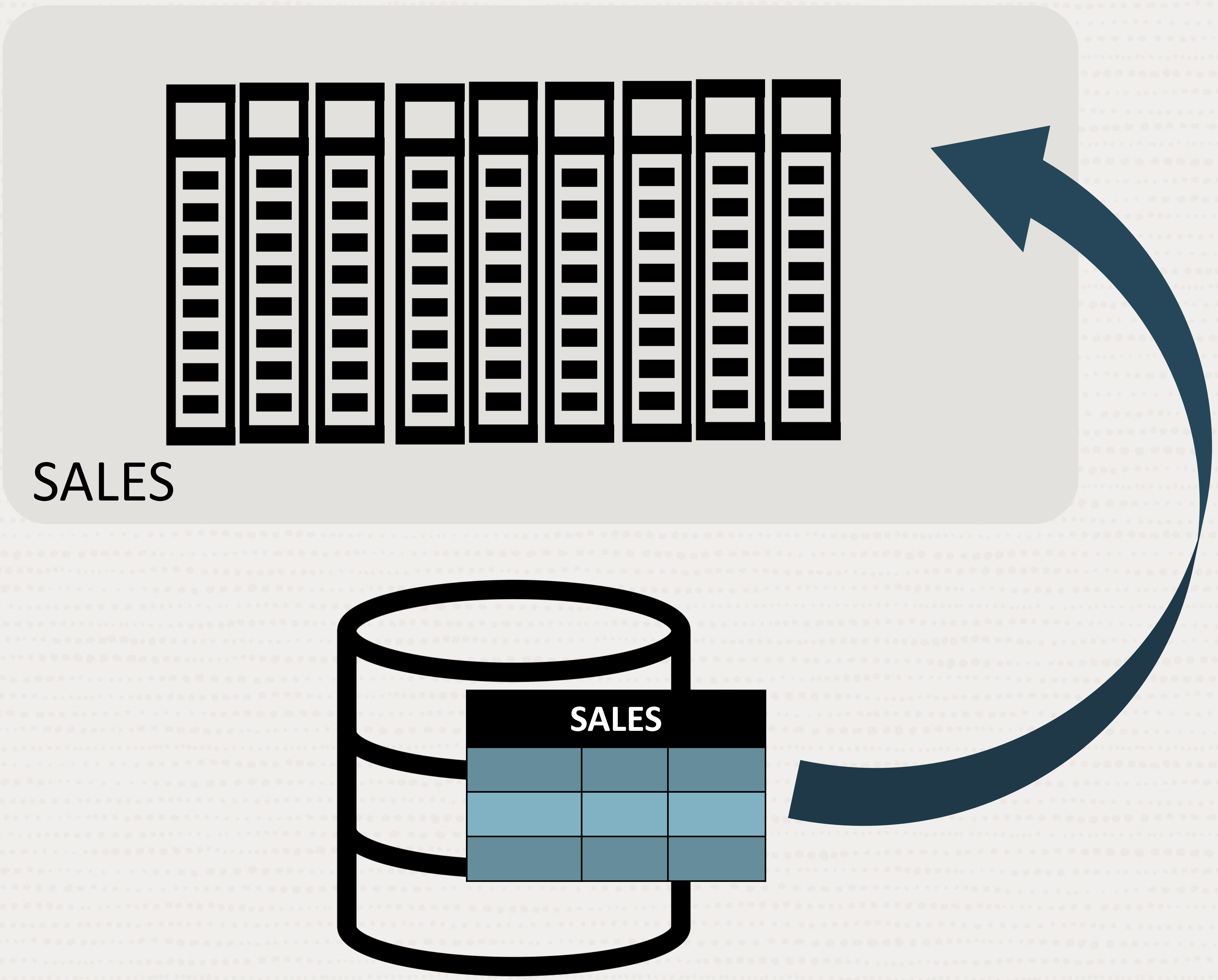
Real-Time Analytics

Database In-Memory | Real-Time Analytics with Fast OLTP



- **Single** copy of data, **Two** in-memory formats
- Both row and column format for the same table
  - Simultaneously active and consistent
- OLTP uses existing row format
- Analytics uses In-Memory column format
- Database In-Memory is seamlessly built into the Oracle Database – **not a separate engine**
- All enterprise features work : RAC, Dataguard, Flashback, etc.

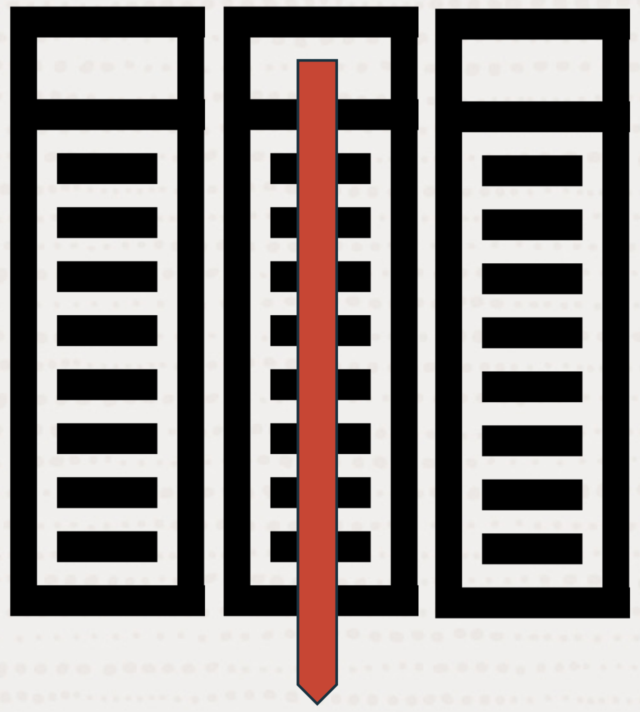
Database In-Memory | Columnar Format



- Pure In-Memory column format
  - In-Memory maintenance: **Fast OLTP**
  - No changes to disk format
  - All features (security, availability) work transparently
- Does not require whole database to be in-memory
  - Can be enabled for hot data, at tablespace, table, partition, level

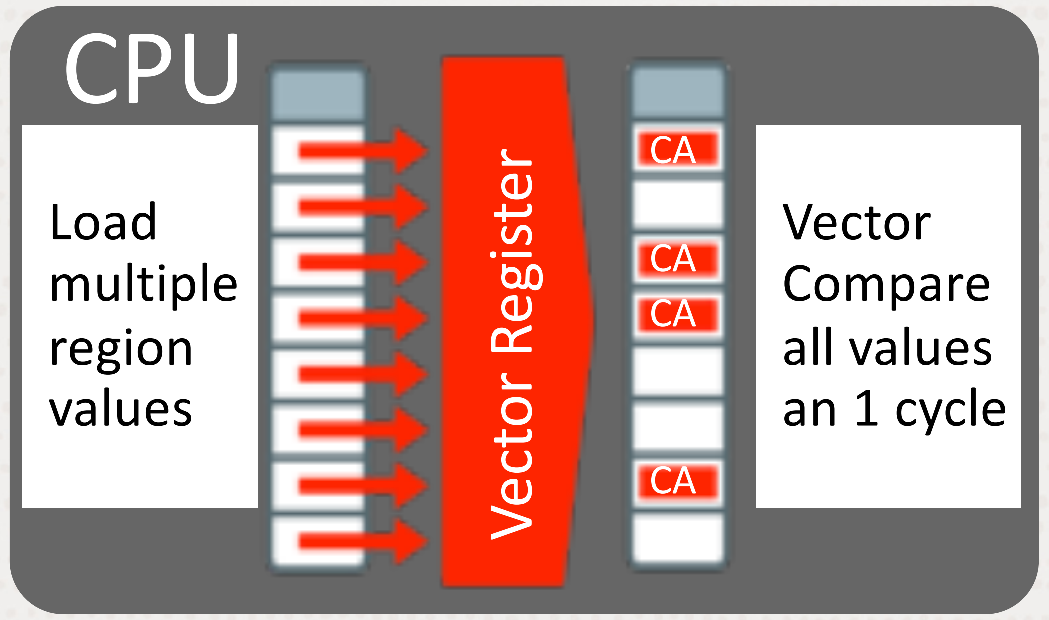
Database In-Memory | Technology

Columnar Format



Access only the columns you need

SIMD Vector Processing

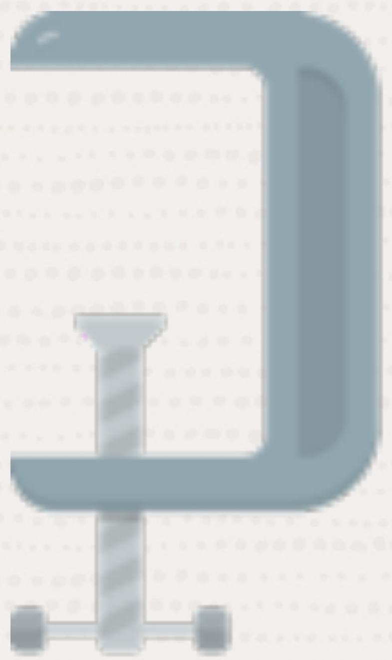


Process multiple column values in a single CPU instruction

Storage Indexes

Prune out any unnecessary data from the column

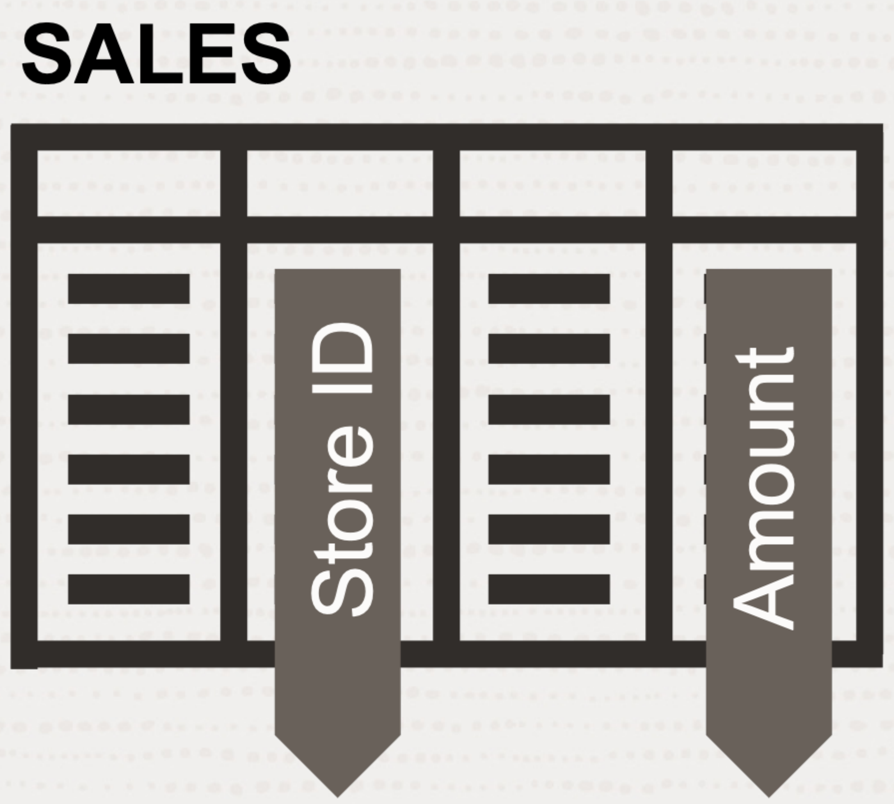
Compression



Scan & filter data in compressed format optimized for space and time

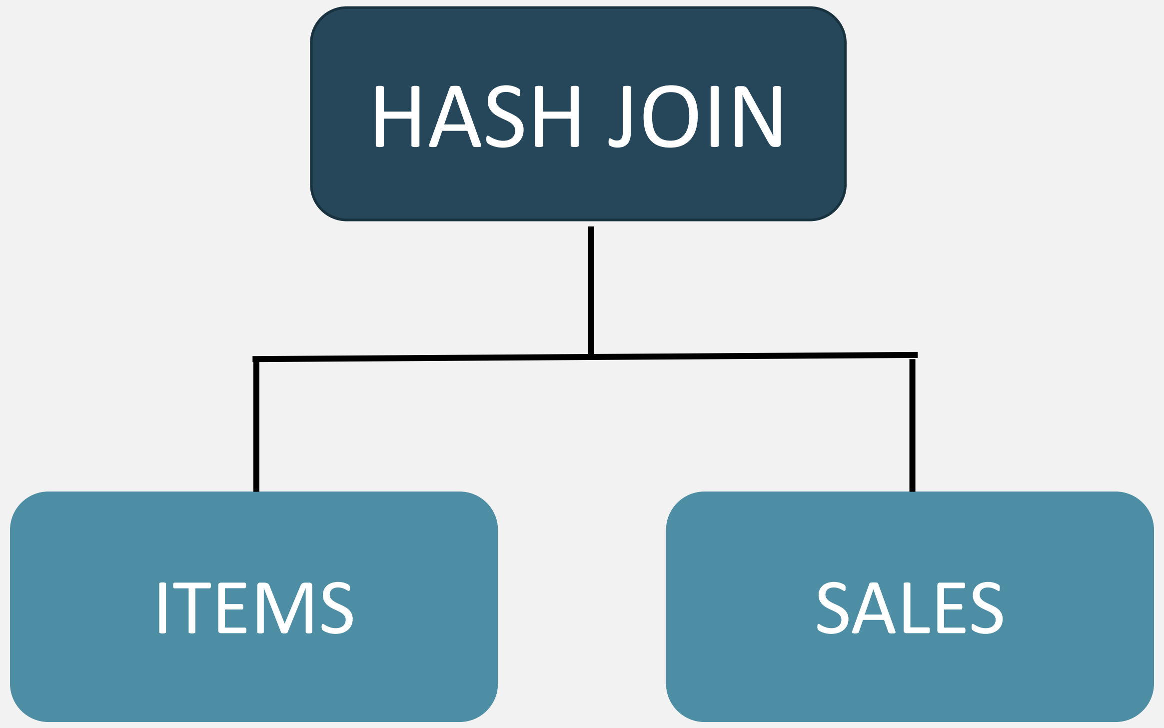
Database In-Memory | Improves All Aspects of Analytics

Scans



- Billions of Rows per second scans using SIMD Vectorization

Joins

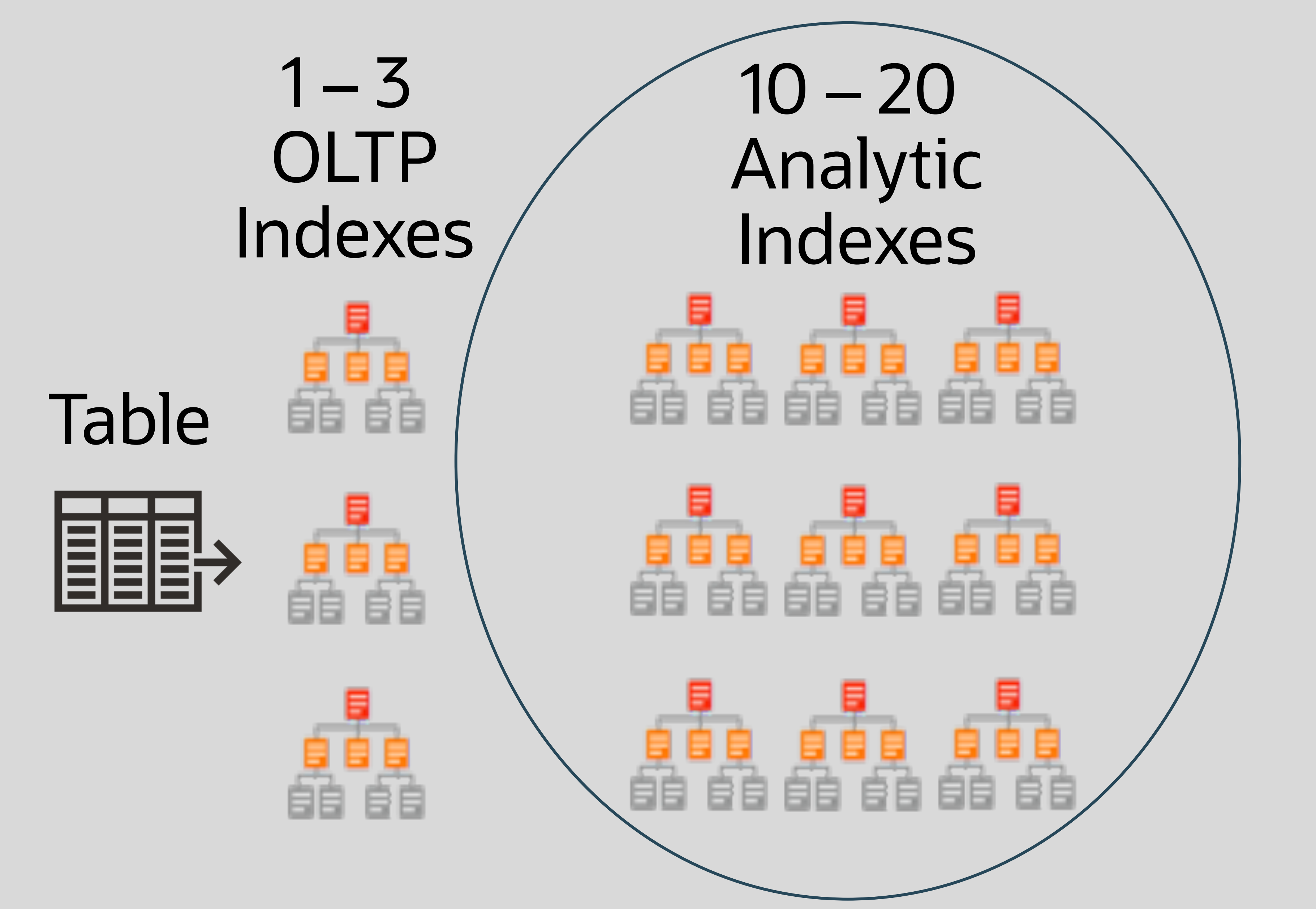


- Convert slower joins into 10x faster filtered column scans levering In-Memory Columnar Data formats

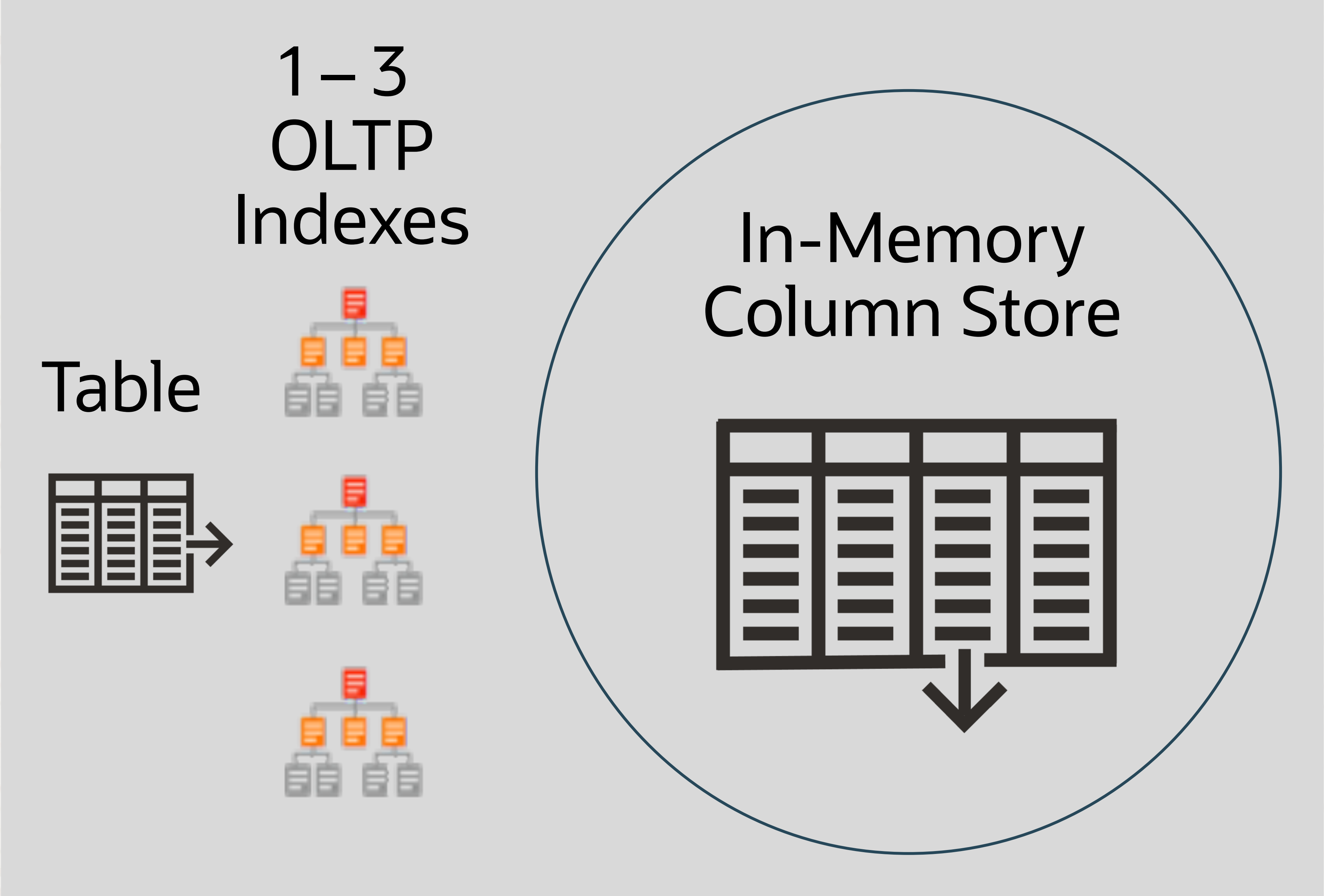
Reporting

- Run reports with aggregations and joins 10x faster using novel memory-optimized algorithms

Database In-Memory | Accelerate Mixed Workloads



REPLACE

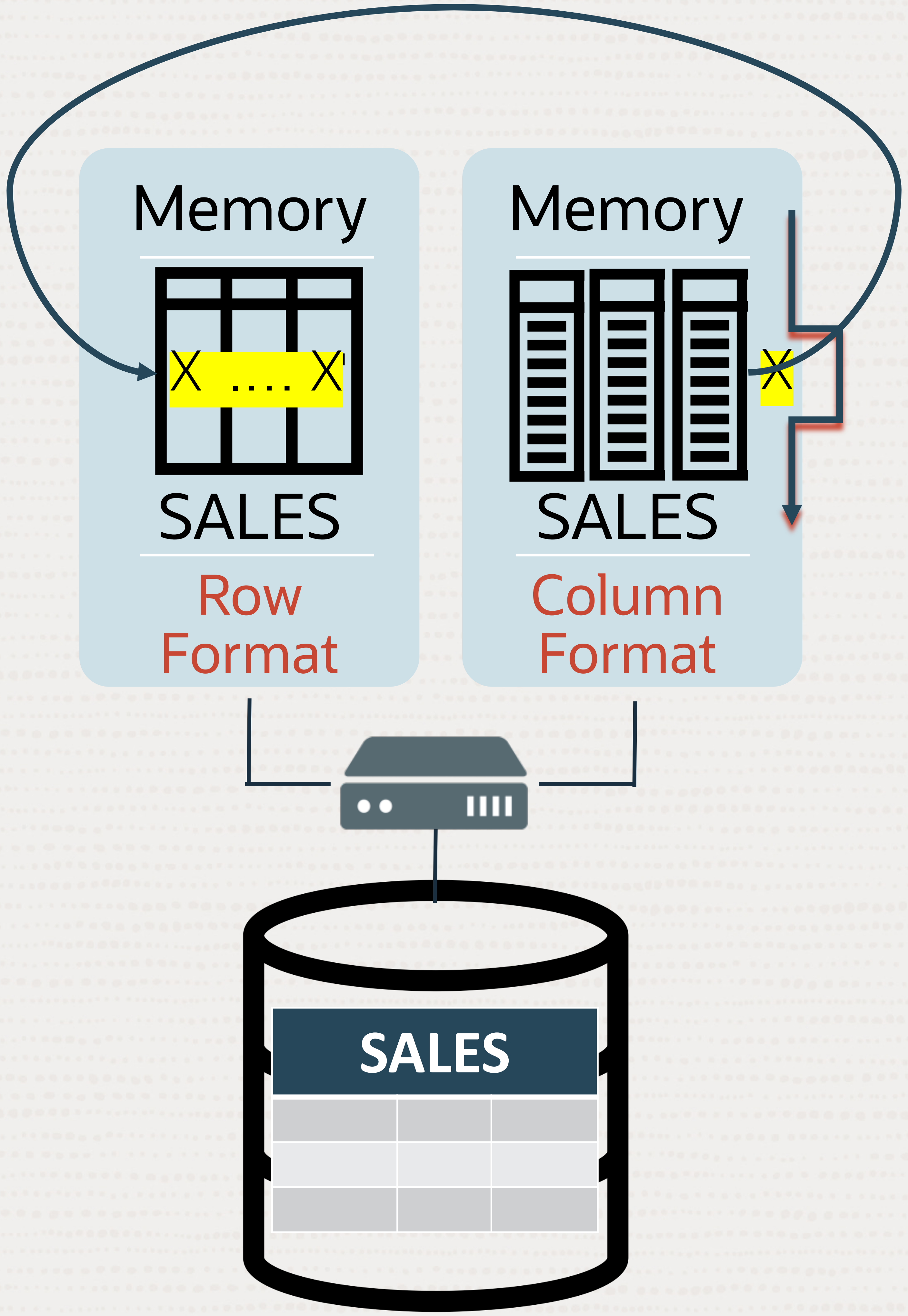


- Inserting one row into a table requires updating 10-20 analytic indexes: Slow!
- Fast analytics only on indexed columns
- Analytic indexes increase database size

- Column Store not persistent so updates are: Fast!
- Fast analytics on any columns
- No analytic indexes: Reduces database size

Mixed Workloads

- Dual-Format Architecture enables fast Mixed Workloads and faster Analytics
- Fast In-Memory DML because invalid row is logically removed from column store (just set a bit)
- Analytic query will ignore invalid rows in column store, and just vector process valid rows.
- Invalid rows are then processed.
- Mixed workload performance can suffer if the number of invalid rows accumulates in IMCUs
  - Additional techniques to refresh a dirty IMCU in the background

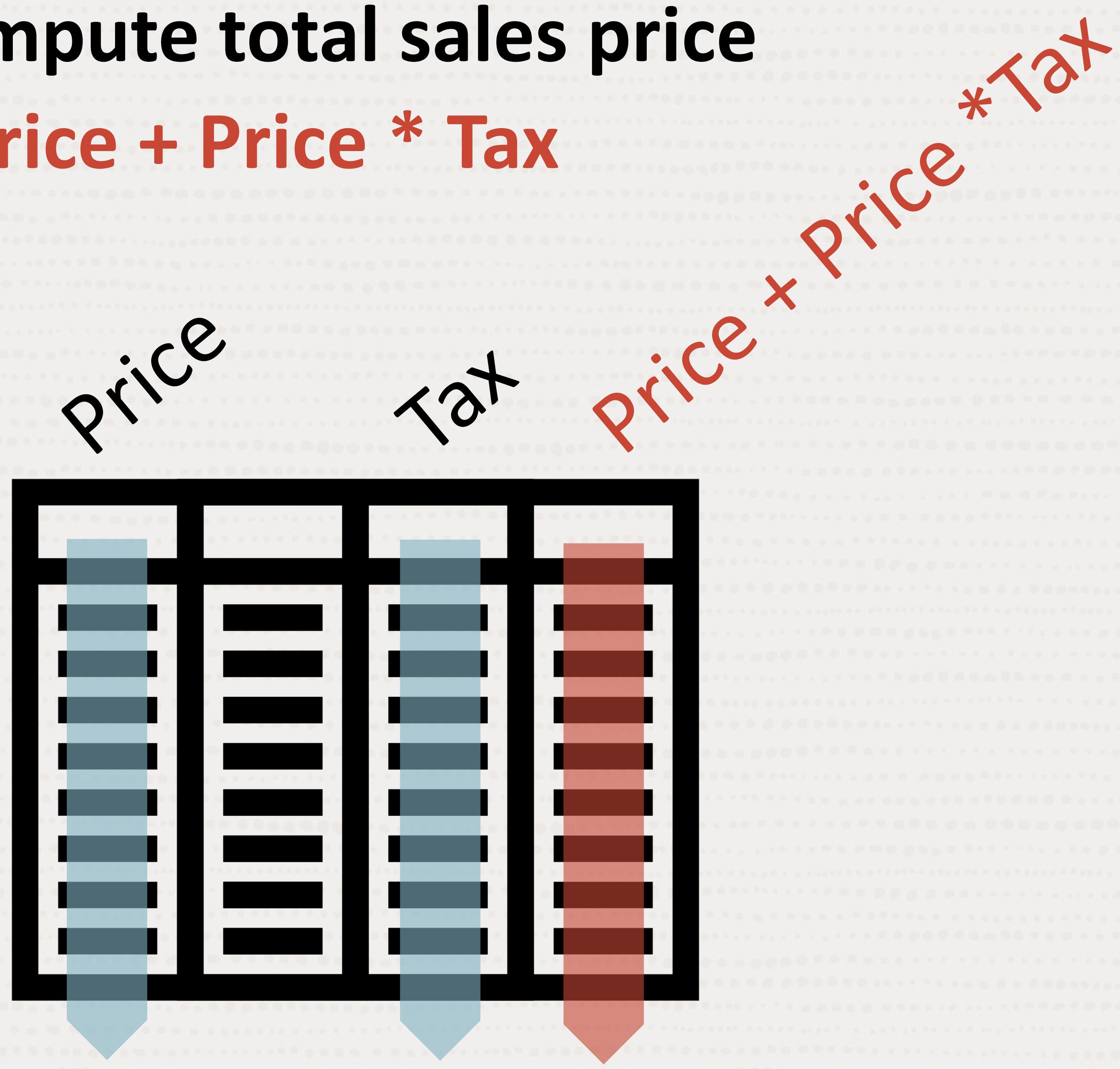


In-Memory Expressions

- Hot expressions can be stored as additional columns in memory
- All In-Memory optimizations apply to expression columns (e.g. Vector processing, storage indexes)
- Two modes:
  - **Manual:** Declare virtual columns for desired inmemory expressions
  - **Auto:** Auto detect frequent expressions
- 3-5x faster complex queries

Example: Compute total sales price

Net = Price + Price * Tax



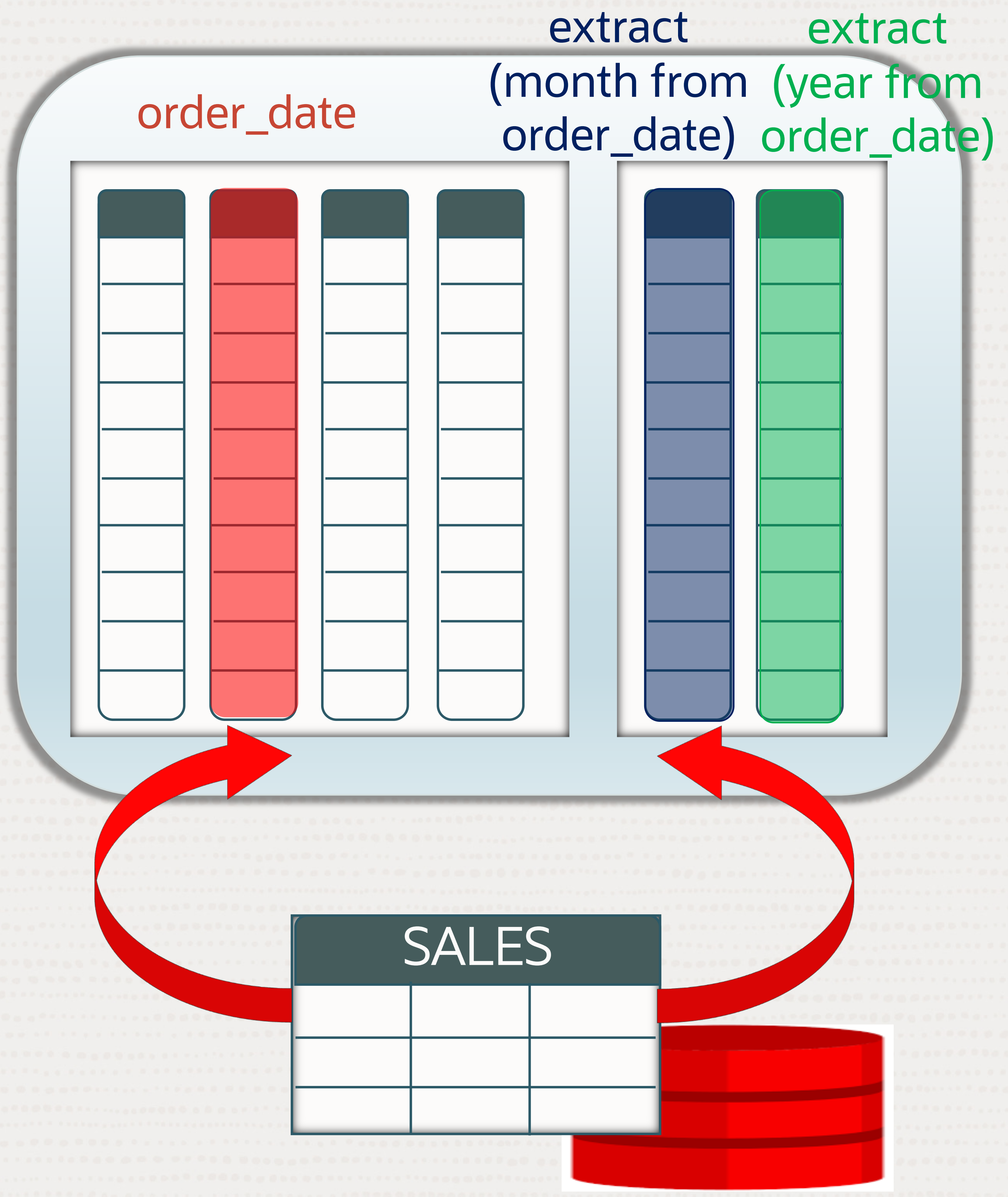

```
CREATE TABLE SALES (
        PRICE NUMBER, TAX NUMBER, …,
        NET AS (PRICE + PRICE * TAX)
        )
INMEMORY;
```

Database In-Memory | Accelerating DATE Queries

- Consider a query to find the *Total sales amount for every month in 2022*

```
select extract(month from order_date) MONTH,
sum(order_amount) TOTAL_SALES
from SALES
where extract(year from order_date) = 2022
group by extract(month from order_date);
```

- In-Memory can now run such queries by up-to **6X faster** by leveraging the In-Memory Expressions framework
  - Each extracted component (e.g. `MONTH`) for a DATE column adds only a **1B** per-row in-memory overhead
  - User can specify which `DATE` column component should be stored in-memory through a parameter



In-Memory Vectorized Joins

If we know ahead of time what tables will be joined, we can make the join fast

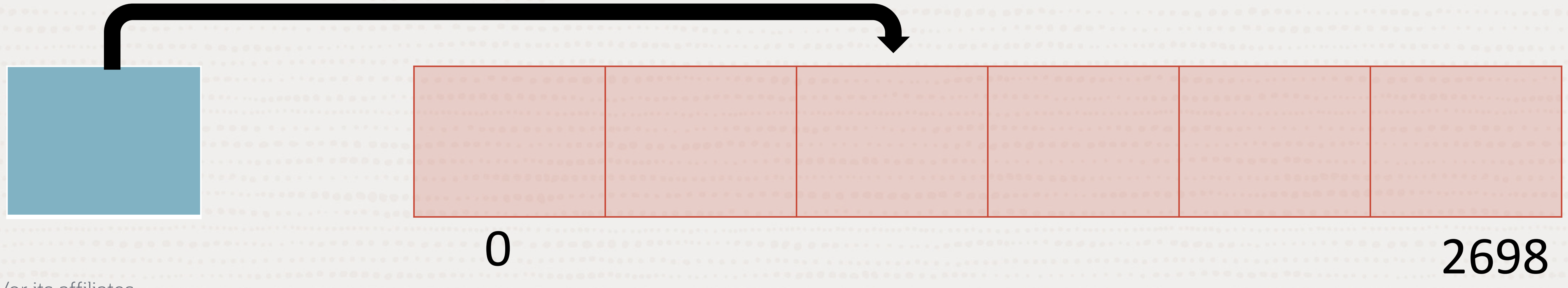
- `Create inmemory join group JG (Lineitem(l_orderkey), Orders(o_orderkey))`

Lineitem (l_orderkey)
151252
358159
695825
…
915238

Global Dictionary (L_orderkey and O_orderkey)	
105125	1
151252	2
…	
695825	1523
915238	2698

	Orders (o_orderkey)
0	105125
	…
	…
	…
2698	915238

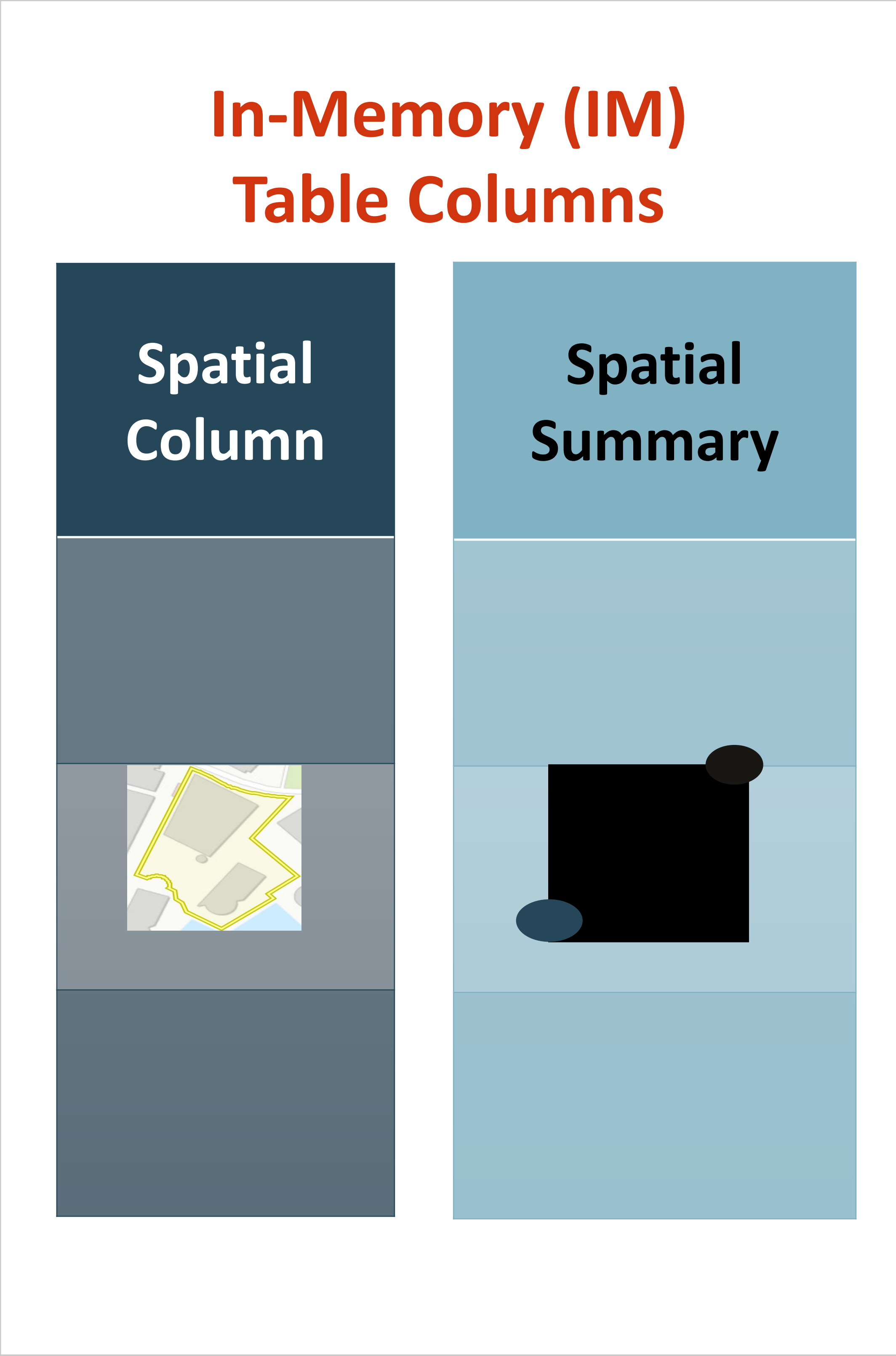
Hash Join is now changed into simple Array Lookup of codes:



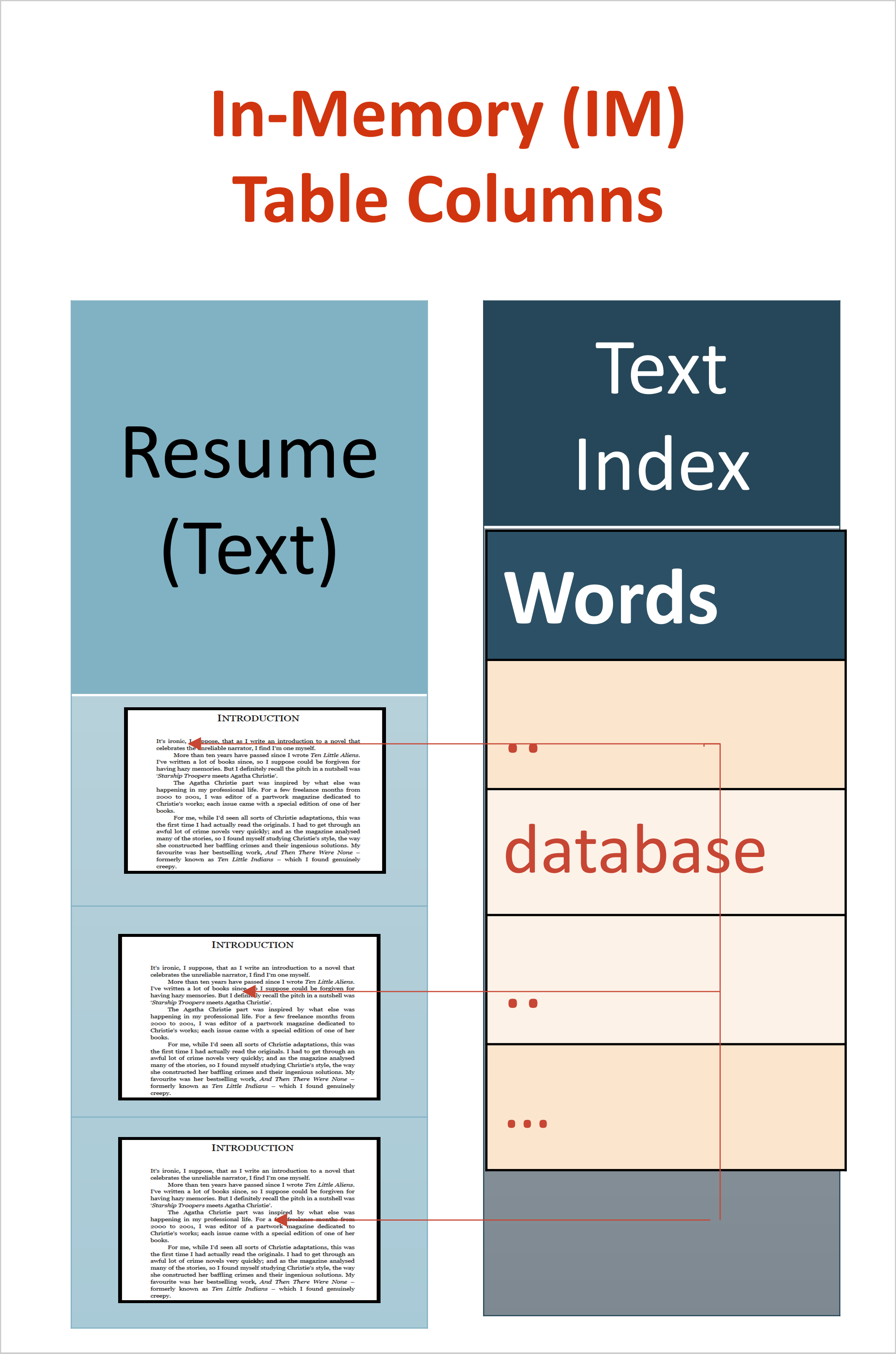
Converged Workloads

In-Memory Analytics on Spatial, Text, and JSON

1. Store **Spatial** Summaries in Column Store for Faster Filtering

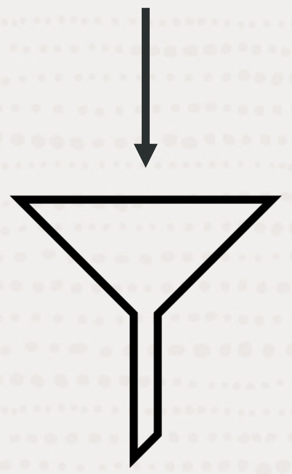


2. Store Optimized **Text** Index structure in Column Store for fast searches



3. Store **JSON** in optimized binary representation in Column Store

```
{
  "Theater":"AMC 15",
  "Movie":”Rogue One",
  "Time“:2017-01-09 18:45",
  "Tickets":{
     "Adults":2
  }
}
```



In-Memory Columnar JSON

- JSON documents stored in Database In-Memory (and in Cell Memory on Storage Nodes with Exadata) automatically get shredded into columns for faster key/value access:

jdoc


```
{
  "firstName": "John",
  "gender": "male",
  "age" : 34,
  "address": {"city": "Redwood City", "state": "CA"},
}
```

```
{
  "firstName": "Alan",
  "gender": "male",
  "age" : 24,
  "address": {"city": "New York", "state": "NY"},
}
```

```
{
  "firstName": "Clara",
  "gender": "female",
  "age" : 53,
  "address": {"city": "Dallas", "state": "TX"},
}
```


Path/Value Columns

name	gender	address.city	address.state	doc id
Alan	Male	New York	NY	2
Clara	Female	Dallas	TX	3
John	Male	Redwood City	CA	1

```
SELECT count(*) FROM  employee WHERE json_exists
(jdoc, '$.person?(@.age < 34 && @.name = 'John' &&
@.address.city = 'Redwood City')')
```

15X Faster Performance

Rich Analytics Query Functionality

SQL for Event Stream Processing
Analytic Window Functions

```
Select MAX(Energy)
(OVER PARTITION BY TIME_IN_HRS)
FROM MeterReadings;
```

- Oracle Database has the industry-leading portfolio of analytic functions for event steam processing:
  - Row Level functions: These are standard SQL functions returning a single value for each row of input (e.g. ROUND, TRUNC, UPPER, etc.) can be used for interpolation, smoothing, etc.
  - Aggregate functions: Return a single value for a group for rows (e.g. MAX, MIN, AVG, SUM etc.)
  - Window functions: Return a single value per row, depending on the group of rows (as specified by a window clause) that the row belongs to
- Window functions are especially useful for analyzing events across different time periods, e.g.
  - Max energy consumption within each 1 hour interval
  - Ranking of 10 minute energy consumption intervals within each day
  - Greatest change in consumption from prior interval

	MeterID	Time	KWhrs	
Window 3 8pm-9pm	1XC23	9:00pm	**2.0**	2
	1XC23	8:45pm	0.86	
	1XC23	8:30pm	0.56	
	1XC23	8:15pm	0.23	
Window 2 7-8pm	1XC23	8:00pm	0.4	1.5
	1XC23	7:45pm	0.5	
	1XC23	7:30pm	0.8	
	1XC23	7:15pm	**1.5**	
Window 1 6-7pm	1XC23	7:00pm	0.7	0.9
	1XC23	6:45pm	0.6	
	1XC23	6:30pm	**0.9**	
	1XC23	6:15pm	0.45	
Window 0 5-6pm	1XC23	6:00pm	0.86	1.34
	1XC23	5:45pm	**1.34**	
	1XC23	5:30pm	0.55	
	1XC23	5:15pm	1.02	

SQL for Event Stream Processing
Pattern Matching

```
Select * From MeterReadings
MATCH_RECOGNIZE(
  ORDER BY Time
  PATTERN(r r)
  ONE ROW PER MATCH
  DEFINE r as KWHrs > PREV(KWhrs))
```

- Event Stream data can be further analyzed using the MATCH_RECOGNIZE construct for SQL pattern matching
- MATCH_RECOGNIZE returns rows from a result set that match a specified pattern within a specified ordering of the result set
- Many use cases – detecting fraud, alerting on high usage, finding anomalies in IoT metrics, etc.:
  - Find meter readings which correspond to two successive periods of increased readings (shown here)
  - Detect a double-dip for a particular stock
  - Detect suspect pattern of credit card charges
- **Eliminates the need to write complex SQL with self joins and nested sub-queries**

MeterID	Time	KWhrs	
1XC23	**9:00pm**	**2.0**	**MATCH**
1XC23	**8:45pm**	**0.86**	**MATCH**
1XC23	8:30pm	0.56	
1XC23	8:15pm	0.23	
1XC23	8:00pm	0.4	
1XC23	7:45pm	0.5	
1XC23	7:30pm	0.8	
1XC23	**7:15pm**	**1.5**	**MATCH**
1XC23	7:00pm	0.7	
1XC23	6:45pm	0.6	
1XC23	6:30pm	0.9	
1XC23	6:15pm	0.45	
1XC23	6:00pm	0.86	
1XC23	5:45pm	1.34	
1XC23	5:30pm	0.55	
1XC23	5:15pm	1.02	

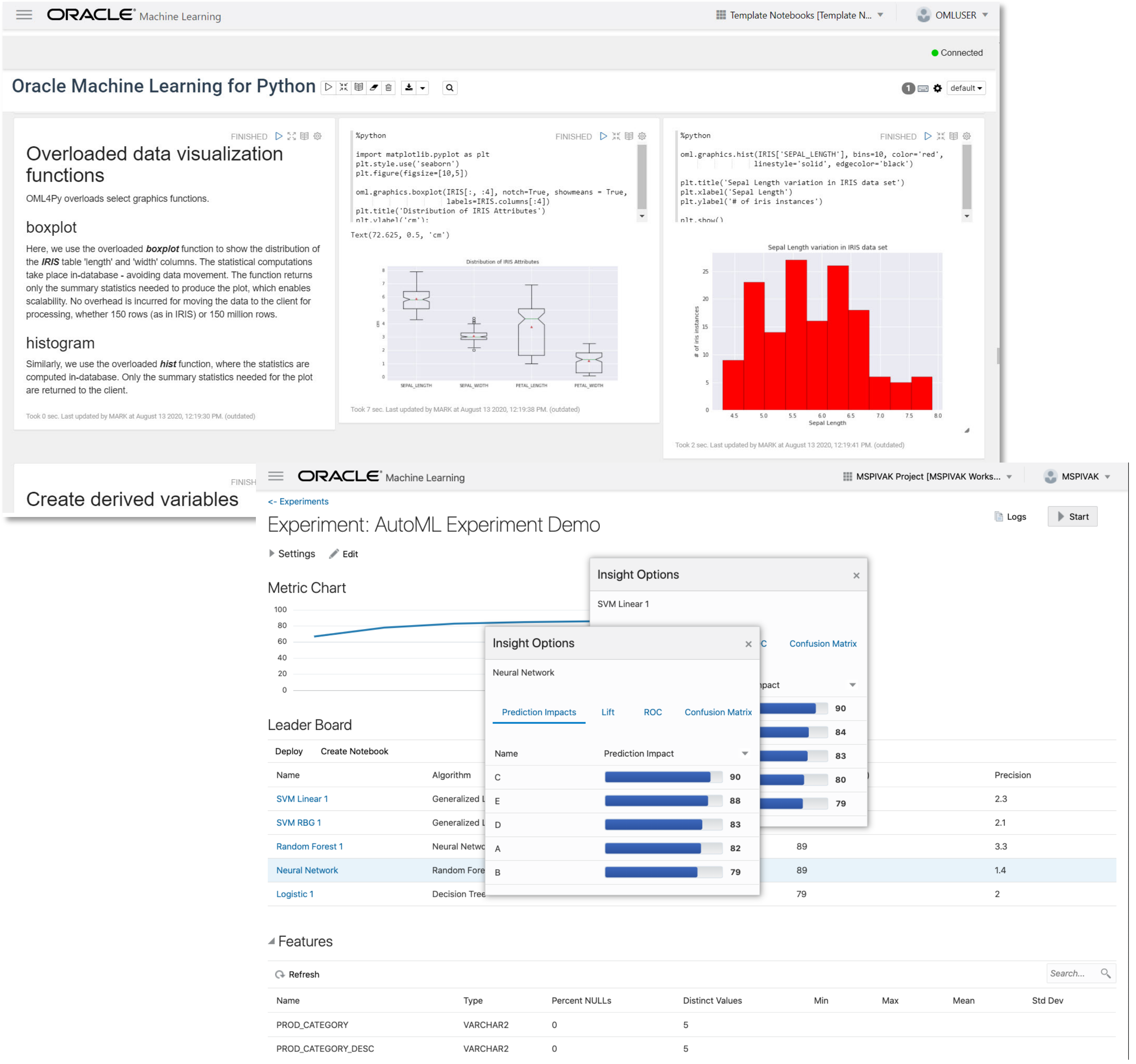
Oracle Machine Learning for Event Streams

- Typical applications of machine learning for event streams include failure prediction from sensor data, fraud detection in financial transactions, sentiment analysis from news feeds, spam filters, detection of correlated failures in event logs

- Oracle Database has a very rich portfolio Machine learning models for Event Streaming use-cases.

- Oracle Database also supports Auto ML which helps to select the ideal algorithms for a given data set that work best for provided data, settling on right data samples for the model, identifying features in data that provide good signal & minimize noise.

- Inferencing based on the models is easily done in real time without requiring any further data movement to a different store



https://oracle.com/machine-learning

Oracle Machine Learning | Summary

Scalable in-database algorithms and open source Python and R integration

Integrated APIs
SQL | Python | R | REST

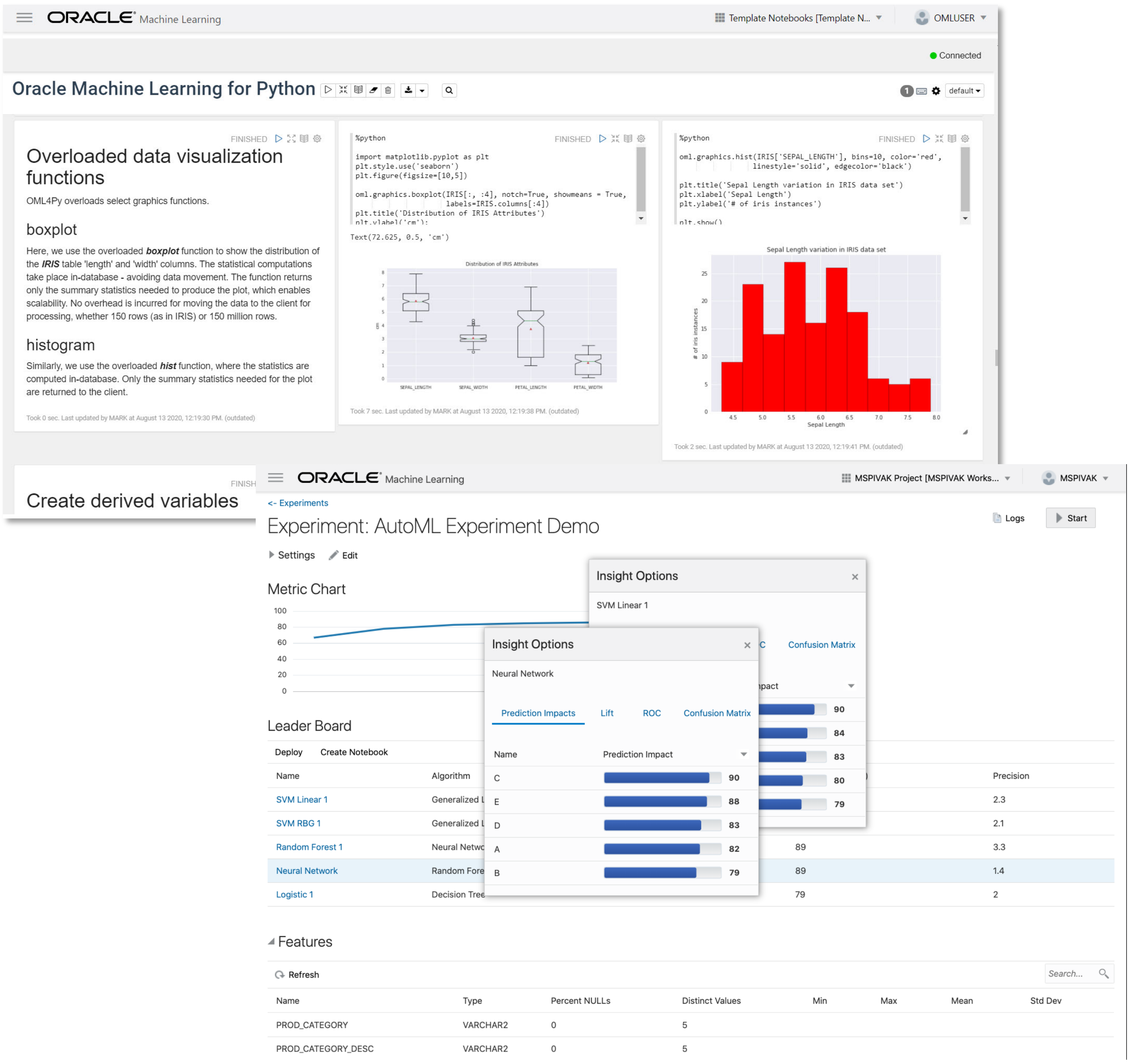
Interfaces
Zeppelin-based collaborative notebooks
AutoML UI – no-code ML modeling
Services – model management and deployment
Use 3rd party IDEs
SQL Developer plug-in Oracle Data Miner

ML techniques
classification | regression | clustering
anomaly detection | time series
feature extraction | attribute importance
ranking | row importance
30+ in-database algorithms

Automation
AutoML API and UI
Algorithm-specific data preparation
Integrated text mining
Partitioned model ensembles

Cloud and on premises
Oracle Database
Oracle Autonomous Database
Oracle Database Cloud Service
Oracle Big Data Service

Big Data
Native and Spark MLlib algorithms
Cloud SQL and Big Data SQL

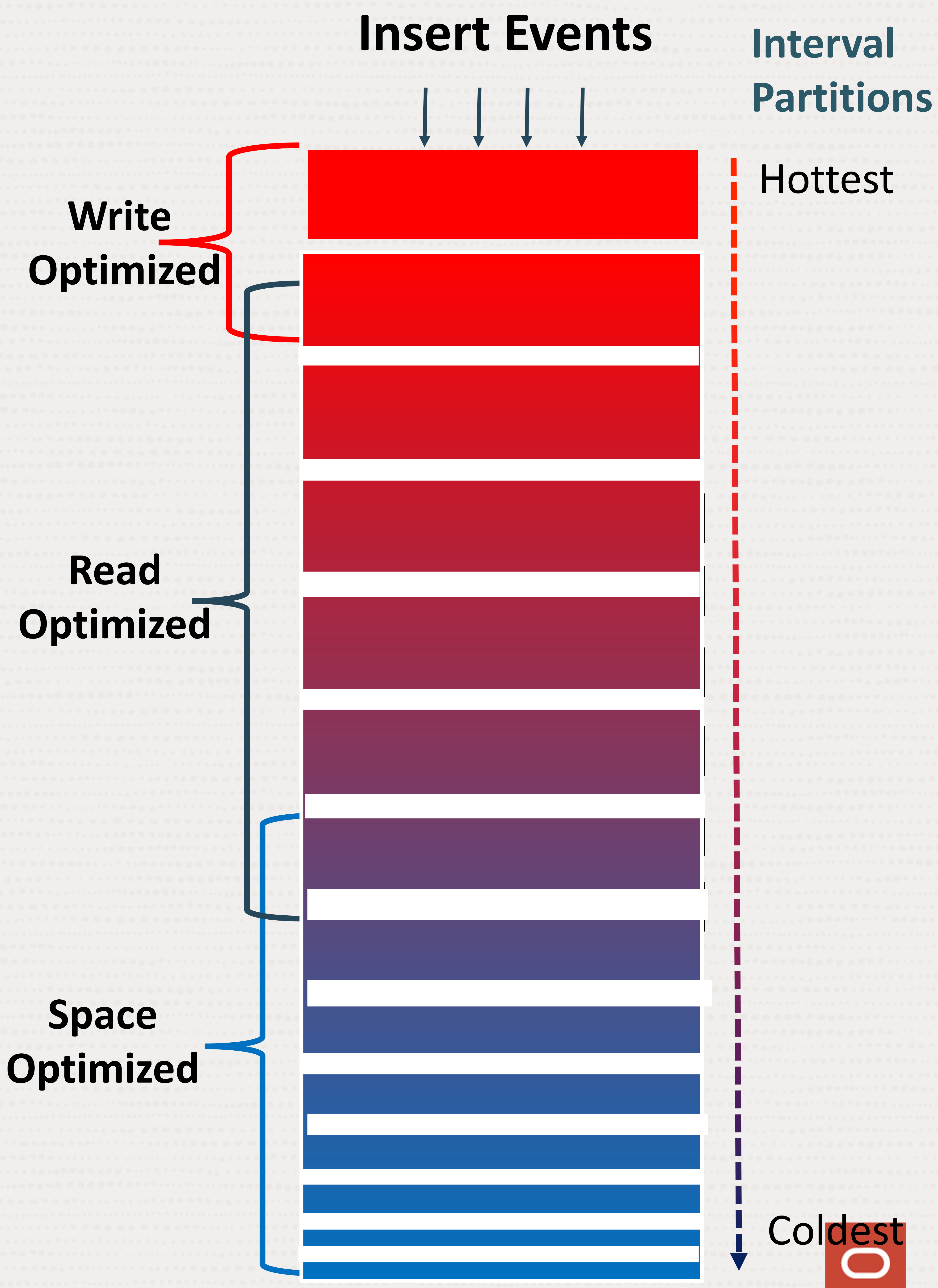


https://oracle.com/machine-learning

Automatic Data Life-Cycle Management

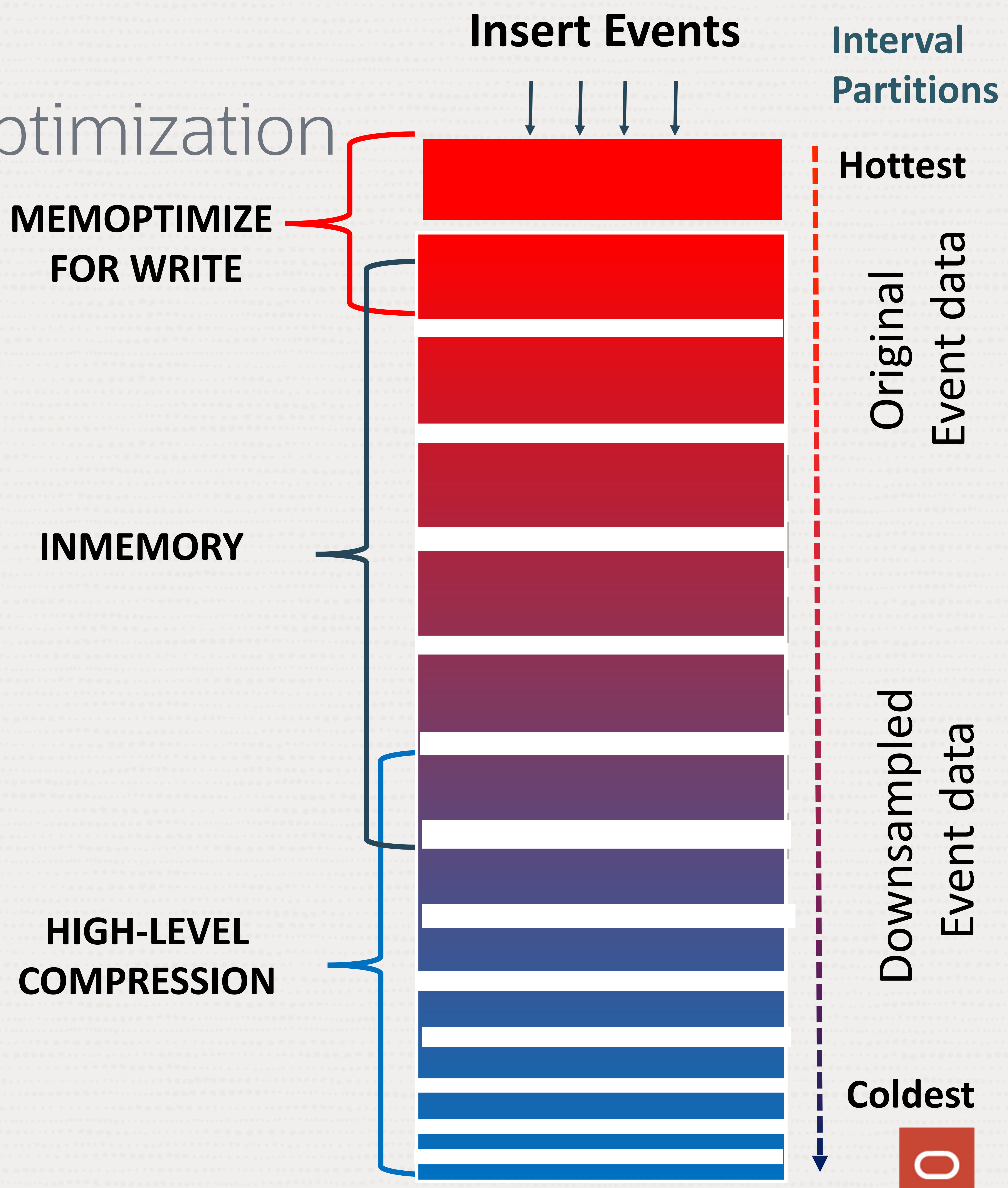
Event Stream Data Characteristics

- Event streams have high data arrival rates and a decaying rate of relevance
  - It is desirable to organize event data in a way that reflects this pattern of usage
- An ideally organized event stream should thus have three optimization zones
  - **Write Optimized Zone**: Organized for fast Ingest
  - **Read Optimized Zone:** Organized for Fast analytics
  - **Space Optimized Zone:** Organized for space savings
- These zones may overlap you may want high speed analytics on recently ingested data as well as on cooler longer term data



Achieving Read, Write & Space Optimization

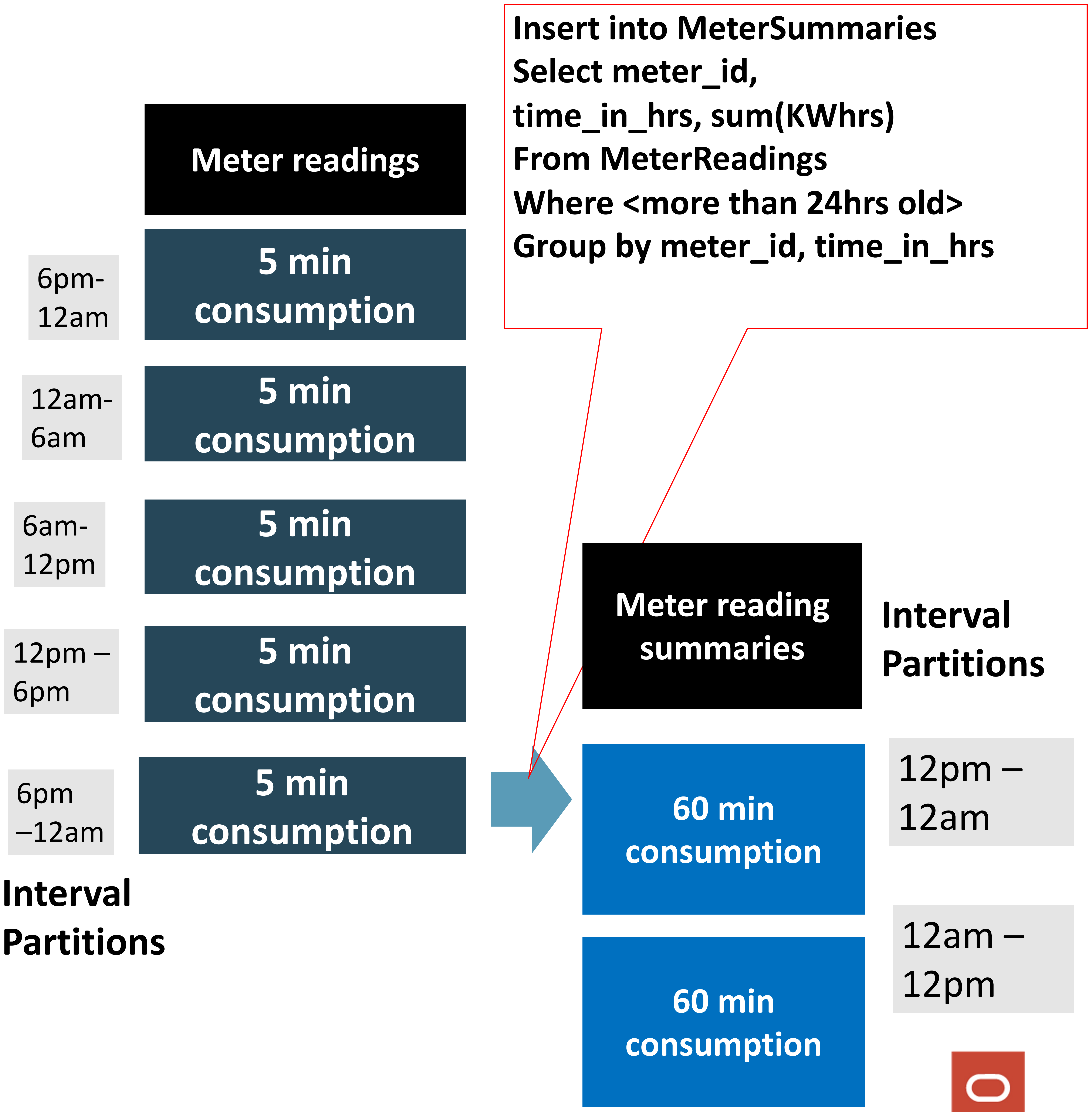
- Write optimized partitions should be declared MEMOPTIMIZED FOR WRITE
  - Recent partitions are typically uncompressed to achieve max ingest speed
- Read Optimized partitions should be declared INMEMORY in order to enable real-time analytics
- Space Optimized partitions should be compressed or downsampled
- This gradient of Write, Read, and Space optimization can be achieved with **Automatic Data Optimization and DBMS_SCHEDULER**



44

Downsampling Event Streams

- To save space and to accelerate reports on older data, events are often *downsampled* or *summarized* as they age
- This downsampling action can be performed via the DBMS_SCHEDULER package
  - E.g. A smart metering application may receive meter readings every five minutes
  - A DBMS_SCHEDULER job generates hourly summaries from intervals more than 24 hours old, inserts them into a summary table
  - Note: The summary table can have a different interval partitioning scheme
  - Can be used to feed other even more granular summaries (e.g. generate daily summaries after a month)



Oracle Database as an Event Stream Processing System

Many capabilities result in **class-leading** Event Stream Processing Support

Flexible data model:

Best-of-Breed Relational, JSON, Spatial, Text, etc.

Native Rest Services

High Speed Ingest:

TimesTen Application-Tier Cache

Memoptimized Tables

Partitioning

Real Application Clusters

Sharding

Exadata Persistent Memory Accelerator

Streaming Analytics Functionality:

Analytic Window Functions

Pattern Matching

Native Machine Learning

Real-Time Analytics:

Database In-Memory

Exadata In-Flash Column Store

Parallel Query

Attribute Clustering

Materialized Views

Automatic Event Lifecycle Management:

Advanced Compression

Automatic Data Optimization

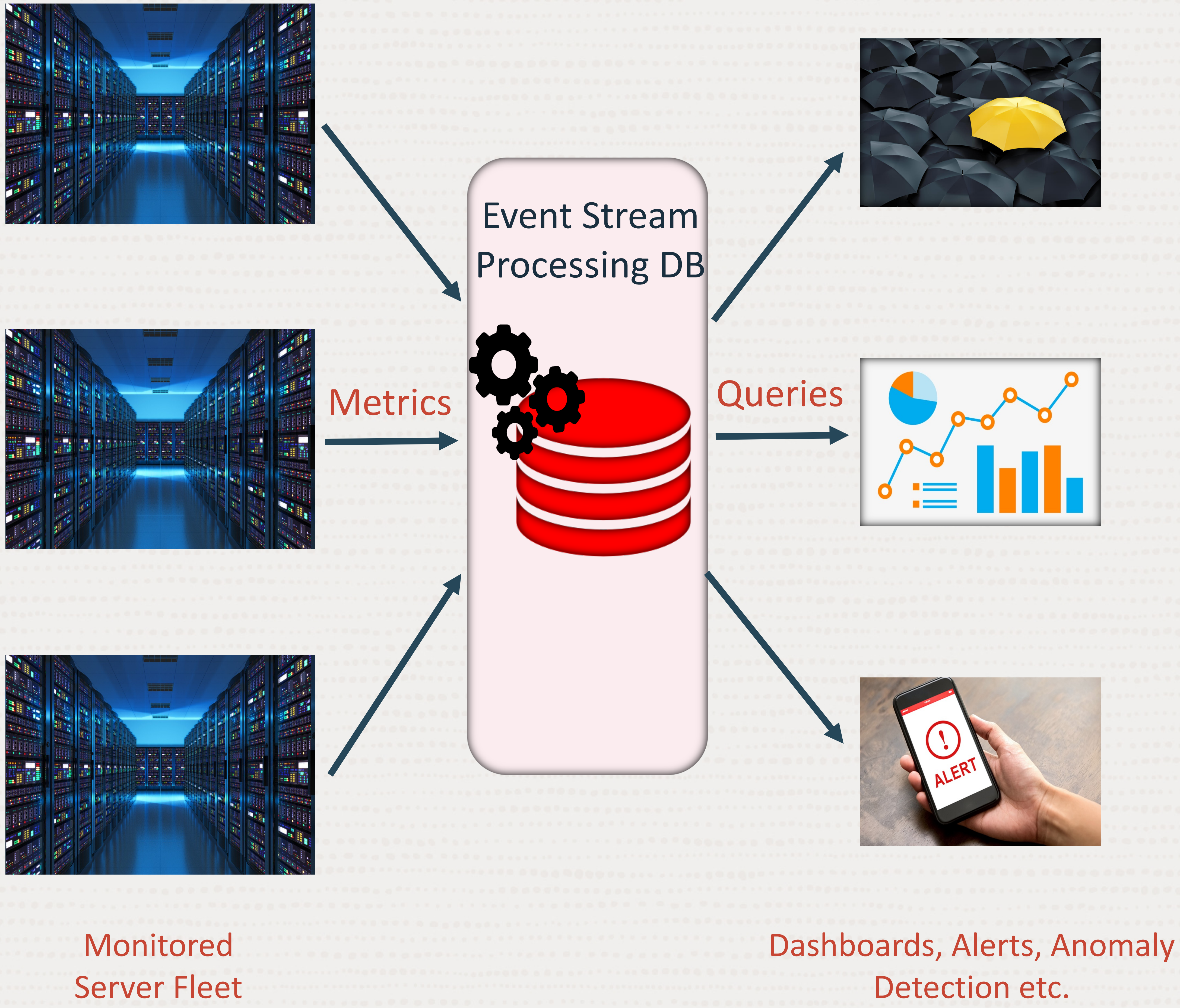
DBMS_SCHEDULER

Demo

DevOps Monitoring with Database In-Memory

DevOps Monitoring

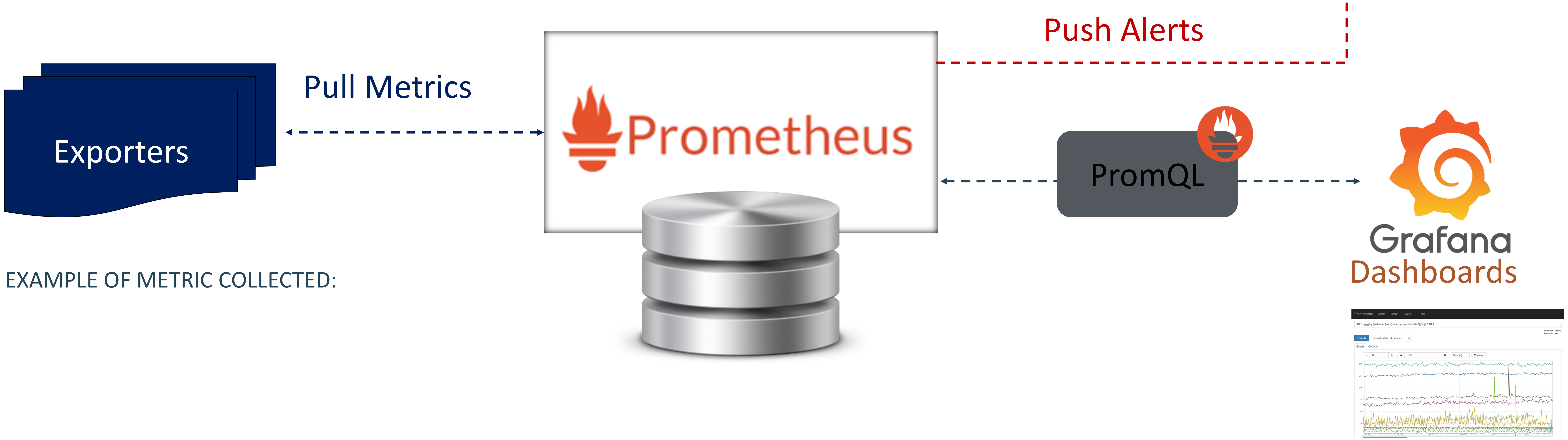
- DevOps has been adopted by companies of all sizes for rapid and continuous delivery of IT applications
- Event stream processing Databases allow businesses to:
  - **Rapidly ingest** streams of metrics emitted by DevOps toolchains
  - Analyze and **predict trends** based on recent and historical data in real-time.
  - Create custom **dashboards** to visualize fleet health
  - Detect **anomalous** behavior and raise alerts



Event Streams POC
Prometheus Ecosystem

Prometheus is a popular open-source systems monitoring and alerting solution.

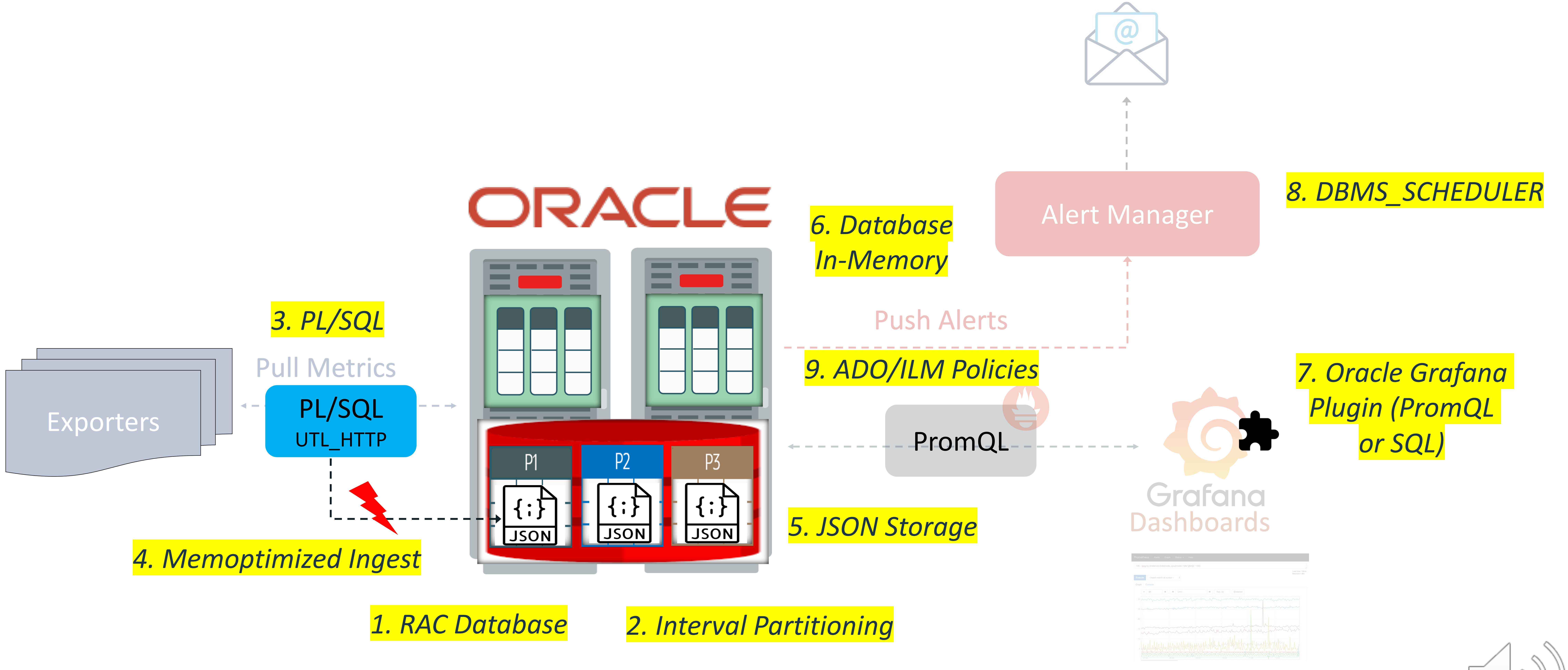
We thought, “Wouldn’t it be cool to see how easily the Oracle RDBMS can replace Prometheus’ storage engine to demonstrate it’s Event Stream Processing capabilities?”



EXAMPLE OF METRIC COLLECTED:

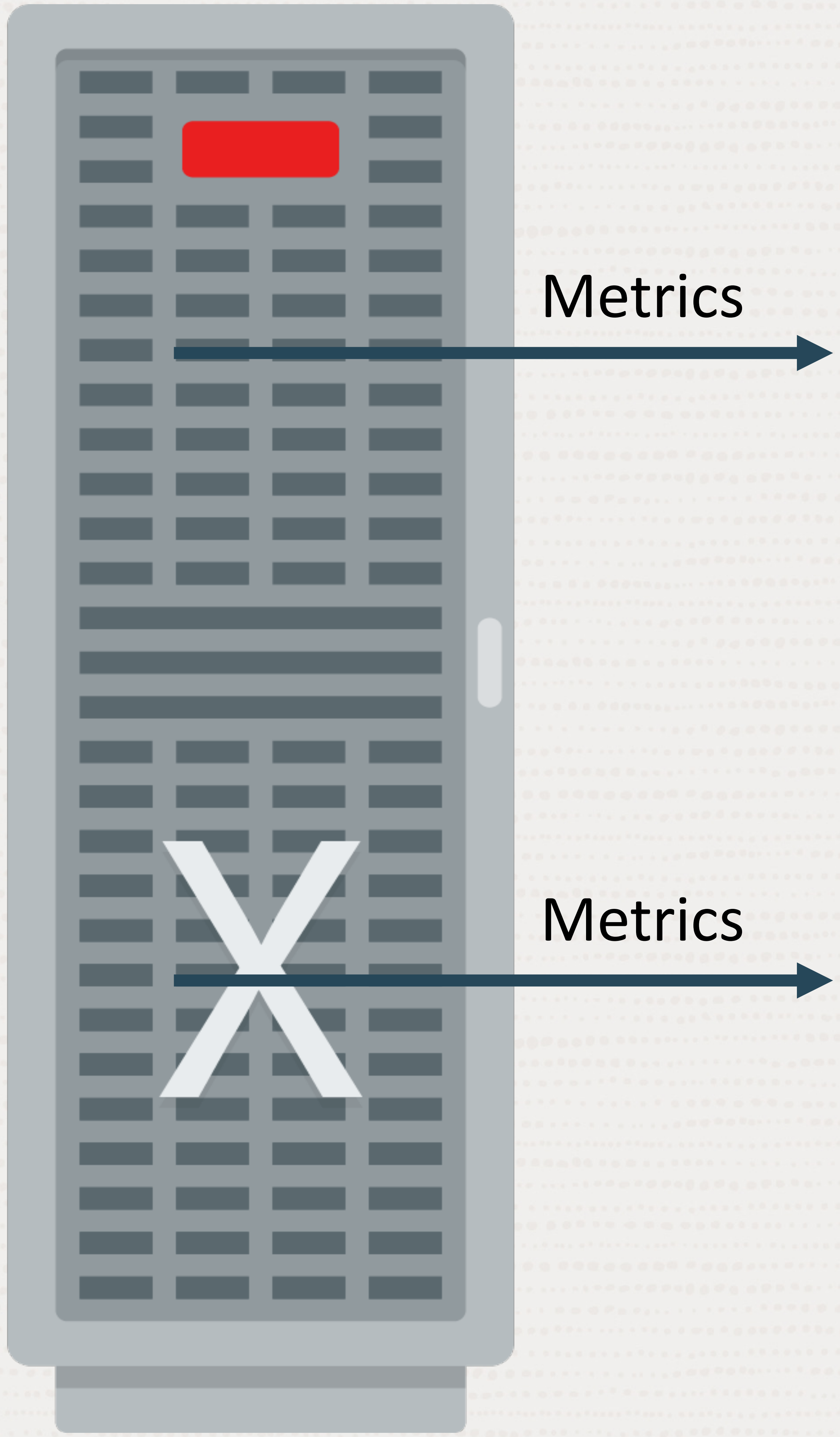
Event Streams POC

Prometheus Ecosystem with Oracle Database Server



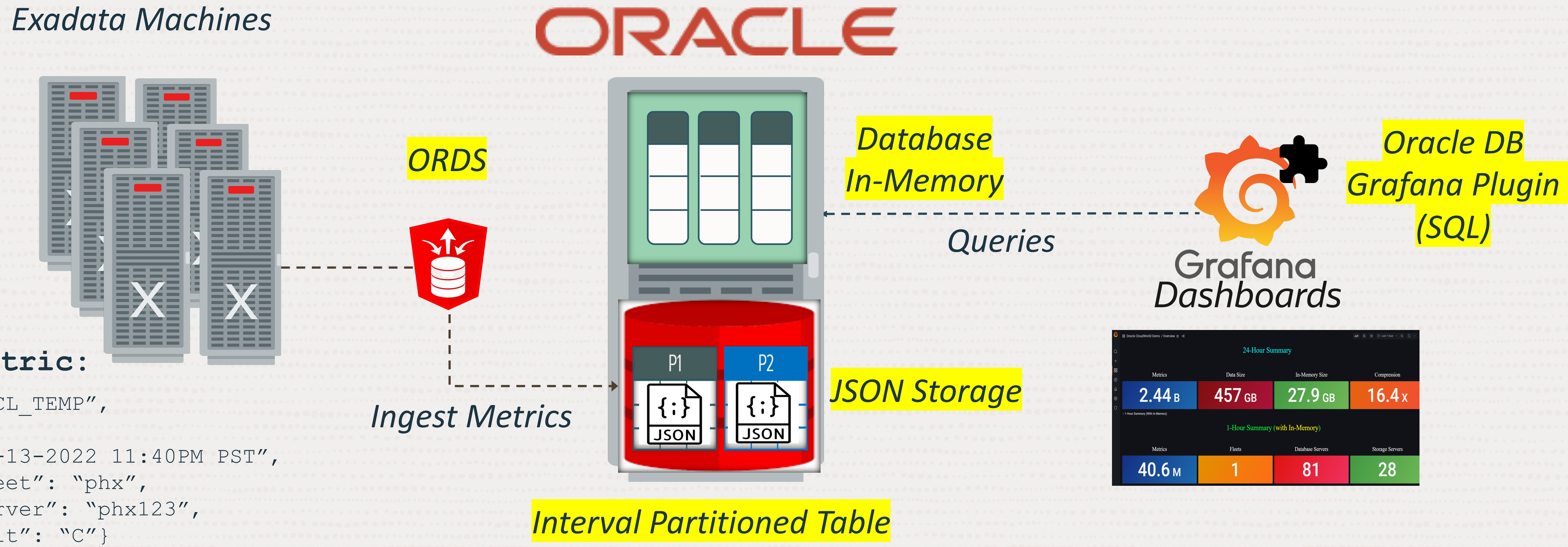
Exadata Real-Time Insights

- Exadata Real-Time Insights is a new and comprehensive monitoring solution for extracting detailed statistics/metrics across the Exadata Machine.
  - Over 2000 events are collected and streamed from Exadata Database and Storage Servers every second.
  - Metrics such as CPU utilization, Memory Utilization, Available Disk Space, etc.
- Real-Time Insights brings DevOps Monitoring to Exadata Systems
  - A monitoring dashboard can be built using an visualization tool such as Grafana to provide a single portal to observe Exadata metrics over time.



DevOps Monitoring Demo | Setup

- Monitor the health of over 500 Exadata Servers at an Oracle data center in real-time.
- Over 2.5 Billion Exadata metrics are captured in an Oracle Database over a 24h period.
- The demo will show how anomalous behavior can be detected instantly thanks to Database In-Memory.



Sample Metric:

```
{"metric": "CL_TEMP",
 "value": 25,
 "time": "10-13-2022 11:40PM PST",
 "tags":{"fleet": "phx",
         "server": "phx123",
         "unit": "C"}
}
```

DevOps Monitoring Demo | Data Model

- Exadata metrics are stored in an Hourly Interval Partitioned table
  - Partitioning the data by `TIME` enables partition pruning which accelerates queries by only scanning data in the desired time window
- Every metric has a `NAME`, `TAGS` describing the metric and its source as a `JSON`, `VALUE`, and `TIME` it was generated
  - Each TAG has 7 labels (e.g. the "server" label represents the name of the server that generated the metric)
- Every row is ~170B in size uncompressed

Schema

```
 Name           Type
 -----------    ------------
 METRIC_NAME    VARCHAR2(200)
 TAGS           JSON
 VALUE          NUMBER
 TIME           NUMBER
```

Sample Metric Row

```
CL_TEMP, /* Storage Server Temp */
{
  "objectName"  : "EDSCELL2",
  "unit"        : "C",
  "server"      : "phxdbfcm99",
  "nodeType"    : "STORAGE",
  "fleet"       : "phx",
  "pod"         : "phxdbfcm99",
  "cluster"     : "phxdbfcm"
}, /* Tags describing the metric */
42, /* Value of Temperature */
1666026000 /* Time in epoch secs */
```

Database In-Memory - DevOps Monitoring Demo

Detailed metrics/statistics from a fleet of Exadata Machines were collected, over a period of 1 day, through the Exadata Real-Time Insights monitoring solution, and were ingested into the Oracle Database.

We want to demonstrate how the health of this fleet can be monitored in real-time to quickly identify any anomalous behaviors across the servers being tracked - e.g. servers which are overheating, or servers low on memory or disk space.

This demo will show how the analytic queries used in this DevOps monitoring use-case are accelerated by orders of magnitude using Database In-Memory, which is essential for systems requiring immediate action in real-time.

Data Overview

Dashboard that details the storage specifics of the data being analyzed

Data Overview

Health Summary

Dashboard that monitors the health of various servers in the system

Health Summary

Performance Summary

Dashboard that compares performance with and without In-Memory

Performance Summary

Data Overview

This dashboard provides an overview of the storage specifics for metrics ingested from a fleet of 500 Exadata Machines located in Oracle's Phoenix Data Center over the last 24 hours.

Metric data is stored in the database using JSON documents for ultimate flexibility in a schema-less data model.

The queries are evaluated in real-time and involve a) JSON processing over metric data, b) Aggregation for analytic reporting, and c) Time-based filtering.

24-Hour Summary

24-Hour Summary

Metrics	Data Size	In-Memory Size	Compression
2.44 B	457 GB	27.9 GB	16.4 x


Oracle CloudWorld Demo / Overview
Last 1 hour
1-Hour Summary (With In-Memory)
1-Hour Summary (with In-Memory)
Metrics
Fleets
Database Servers
Storage Servers
60.1 M
1
85
28
1-Hour Summary (Without In-Memory)
1-Hour Summary (without In-Memory)
Metrics
Fleets
Database Servers
Storage Servers


Overview / Edit Panel
Table view
Fill
Actual
Last 1 hour
28
Query 1
Transform 0
Data source
demo2
Query options
MD = auto = 352
Interval = 10s
Query inspector
A
(demo2)
SQL
select count(distinct json_value(tags,'$.objectName')) value from metrics_data where (json_value(tags,'$.objectName') like '%cell%' or
json_value(tags,'$.objectName') like '%CELL%') and time > :start_time - $offset and time < :end_time - $offset
Convert Results
Time Bound
Legend
legend format
StepSize
Min 10, Default 10
PrefetchCount
Default 100
PROMQL To SQL
converted sql will appear hear
SQL Editor
1
Enter Sql Query here

Oracle CloudWorld Demo / Health Summary

Last 3 hours

Temperature 40 | CPU Usage 99 | Memory Usage 99 | Write Latency (s) 1

Health Summary

This dashboard monitors servers that are operating outside the normal window of temperature, CPU/Memory usage and I/O latency, with and without In-Memory

The thresholds can be configured by variables at the top of the dashboard.

This demo will show how with Database In-Memory we can quickly identify problematic servers in real-time

› **Health Summary with In-Memory** *(14 panels)*

› **Health Summary without In-Memory** *(14 panels)*

˅ Health Summary with In-Memory

Health Summary (In-Memory)

Servers exceeding thresholds

Temperature

1

SERVER	LAST_OCCURRENCE	TEMPE
phxdbfcm99	14-OCT-22 07.51.30...	42

CPU

64

SERVER	LAST_OCCURRENCE	CPU_U
phxdbfcb94	14-OCT-22 07.59.55...	99.7
phxdbfcb30	14-OCT-22 07.59.54...	99.7
phxdbfca83	14-OCT-22 07.59.50...	99.8
phxdbfcv68	14-OCT-22 07.59.35...	99.8

Memory

0

SERVERS	LAST_OCCURRENCE	MEMO

No data

I/O Writes

57

SERVER	LAST_OCCURRENCE	WRITE_
phxdbfca17	14-OCT-22 07.59.59...	2.19
phxdbfcq21	14-OCT-22 07.59.54...	1.14
phxdbfcq67	14-OCT-22 07.57.02...	1.26
phxdbfcc06	14-OCT-22 07.55.27...	1.15

Health Summary without In-Memory

Health Summary (without In-Memory)

Servers exceeding thresholds

Temperature	CPU	Memory	I/O Writes

⌄ Health Summary with In-Memory

Health Summary (In-Memory)

Servers exceeding thresholds

Temperature

1

SERVER	LAST_OCCURRENCE	TEMPE
phxdbfcm99	14-OCT-22 07.51.30...	42

CPU

64

SERVER	LAST_OCCURRENCE	CPU_U
phxdbfcb94	14-OCT-22 07.59.55...	99.7
phxdbfcb30	14-OCT-22 07.59.54...	99.7
phxdbfca83	14-OCT-22 07.59.50...	99.8
phxdbfcv68	14-OCT-22 07.59.35...	99.8

Memory

0

SERVERS	LAST_OCCURRENCE	MEMO

No data

I/O Writes

57

SERVER	LAST_OCCURRENCE	WRITE_
phxdbfca17	14-OCT-22 07.59.59...	2.19
phxdbfcq21	14-OCT-22 07.59.54...	1.14
phxdbfcq67	14-OCT-22 07.57.02...	1.26
phxdbfcc06	14-OCT-22 07.55.27...	1.15

Oracle CloudWorld Demo / Server Details

2022-10-14 07:41:30 to 2022-10-14 08:01:30

server phxdbfcm99

Anomalous Server Details

This dashboard is used to investigate various metrics for a server around the time of its anomalous behavior using In-Memory

Storage Server Temperature

Disk Controller Battery Temperature

Storage Server CPU Utilization

Storage Server Working Fans

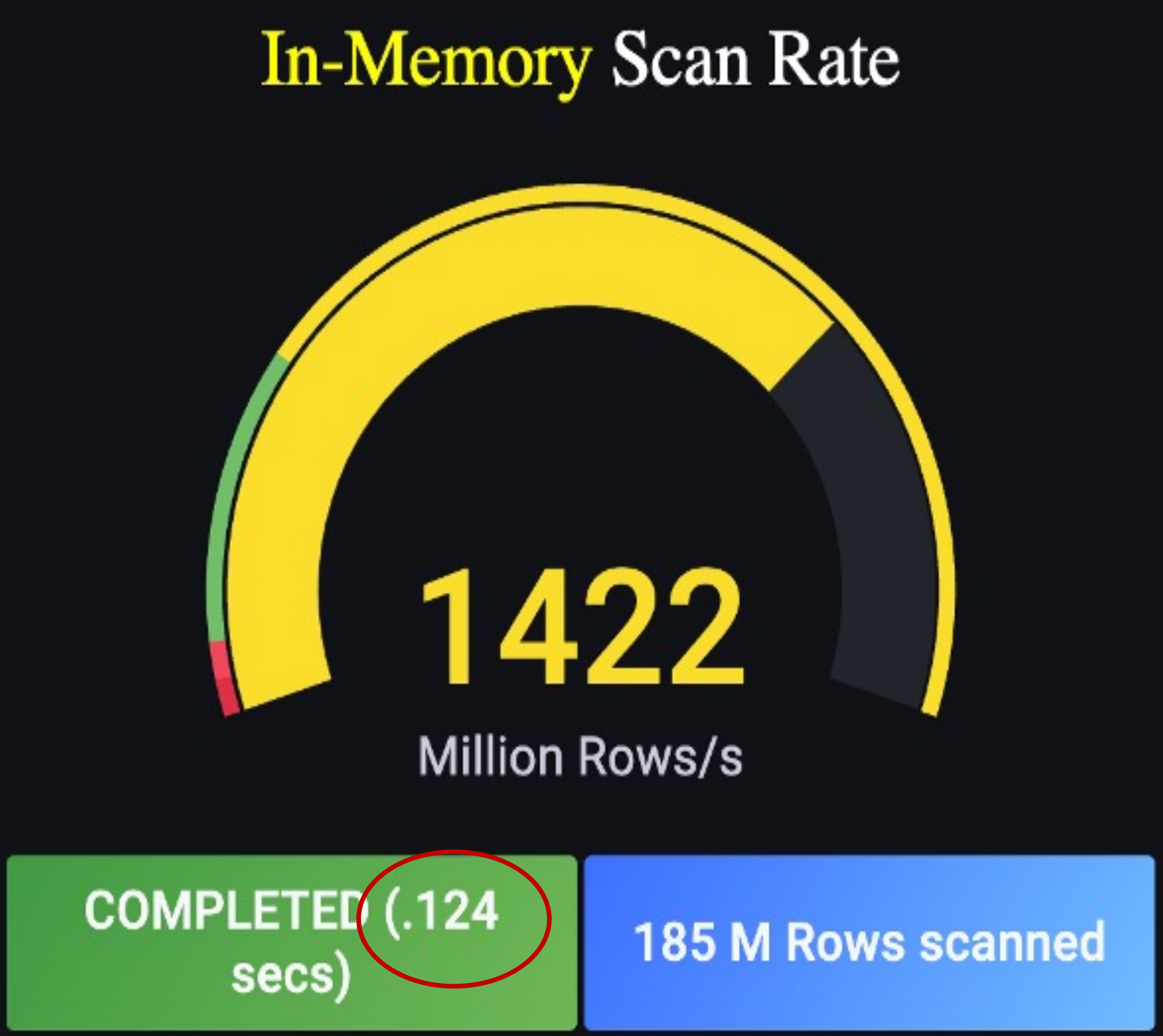

Health Summary without In-Memory
Health Summary (without In-Memory)
Servers exceeding thresholds
Temperature
CPU
Memory
I/O Writes
1
64
0
57
SERVER
LAST_OCCURRENCE
TEMPE
phxdbfcm99
14-OCT-22 07.51.30...
42


Oracle CloudWorld Demo / Performance Overview
Last 3 hours
5s
Ingest
DISABLE
Performance Overview
This dashboard tracks the performance of the health monitoring queries executed with and without In-Memory.
Database In-Memory improves these queries by 100x or more.
Query Performance
In-Memory Scan Rate
1422
Million Rows/s
COMPLETED (.124 secs)
185 M Rows scanned
X-Factor
384x
Without In-Memory Scan Rate
3.70
Million Rows/s
COMPLETED (47.62 secs)
185 M Rows scanned

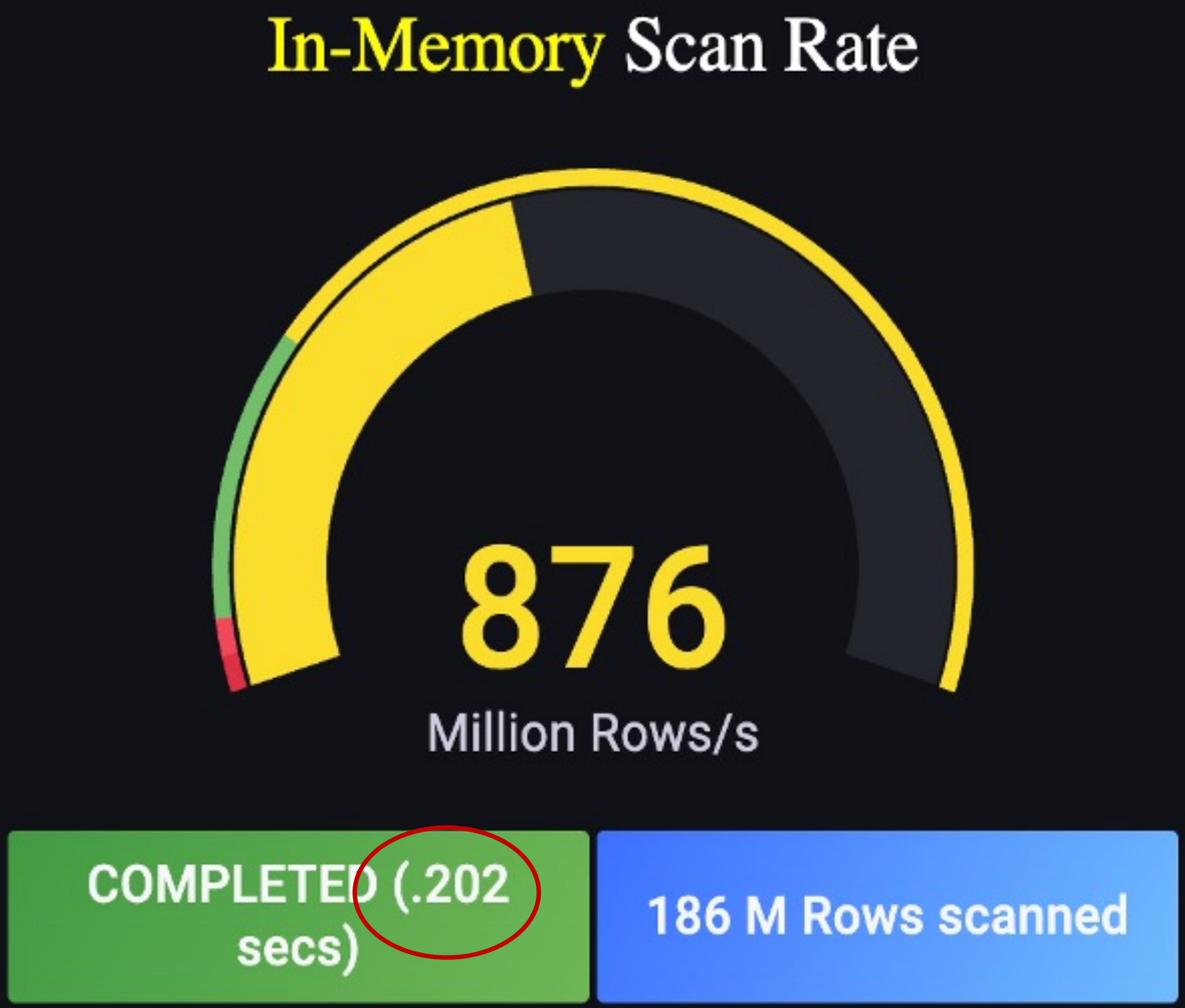

Oracle CloudWorld Demo / Performance Overview
Last 3 hours
5s
Ingest
ENABLE
Performance Overview
This dashboard tracks the performance of the health monitoring queries executed with and without In-Memory.
Database In-Memory improves these queries by 100x or more.
Query Performance
In-Memory Scan Rate
1422
Million Rows/s
COMPLETED (.124 secs)
185 M Rows scanned
X-Factor
384x
Without In-Memory Scan Rate
3.70
Million Rows/s
COMPLETED (47.62 secs)
185 M Rows scanned


Concurrent Ingestion
Ingest Rate
1.24 M
Rows Per Minute
ENABLED

Without Ingestion



With Ingestion



DevOps Monitoring Demo | Indexes

- Consider the query to find the *Number of distinct servers that have generated metrics in the last 1 hour*
  - A local-partitioned index on `(TIME, JSON_VALUE(TAGS, '$.SERVER'))` can achieve **10X faster** query execution over No In-Memory full table scans
  - However, it is still **10X-15X slower** than In-Memory query execution
  - In-Memory query execution is super-charged through Aggregation pushdown, optimized JSON evaluation, Min-max pruning, etc. which are not available in indexed execution
- Indexes occupy additional space, and the database needs a large buffer cache to avoid I/Os
  - For e.g., the index described above is 75GB in size (~18% of data size)
  - Different indexes need to be created to accelerate other dashboard queries (e.g. index on `(TIME, JSON_VALUE(TAGS, '$.NODETYPE'))` to find number of distinct storage servers)
- Further, indexes require maintenance which can slow down DMLs significantly
- Thus, In-Memory is the only solution that can provide instantaneous Real-Time analytics

Demo Summary

- Database In-Memory is essential for use-cases like DevOps Monitoring, where real-time anomaly detection and drill-down analysis is absolutely needed.
  - Any loss of time identifying and triaging irregularities in your data fleet can amount to customer dissatisfaction and loss of revenue.
- The demo showed how Database In-Memory could be used to speed up Exadata metrics monitoring by 400x compared to traditional buffer cache row-based processing.
  - With In-Memory enabled, we were able to detect and identify anomalous servers in the data center, and subsequently drill-down into the metric events to identify a potential root cause, **all before** the non In-Memory dashboard could even reveal that there *were* problem servers in the fleet.
- Database In-Memory would still be 10-15X faster than system with analytic indexes. But indexes are not practical because of a) high DML cost, b) high I/O due to space needed

Research and Development Opportunities

Event Stream Processing and more

- Compression Technology
- Hardware Acceleration
- Approximate Indexes
- Simplifying SQL
- Optimizing Algorithms for Database Operations and SQL Functionality
- Machine Learning and Expert Systems for Data Management
- Mixed Workload – Single Database for Operational Data and Reporting
- In-Memory Technology (Analytics and OLTP)

Thank You.

db_career_us_grp@oracle.com