ORACLE

Oracle Database MultiTenant/MultiModel/InMemory

Workshop

Guillermo Best
Oscar Sánchez
David Rodriguez
José Vázquez
Francisco Alvarez
Stephane Duprat

Madrid 14 de enero de 2020

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.

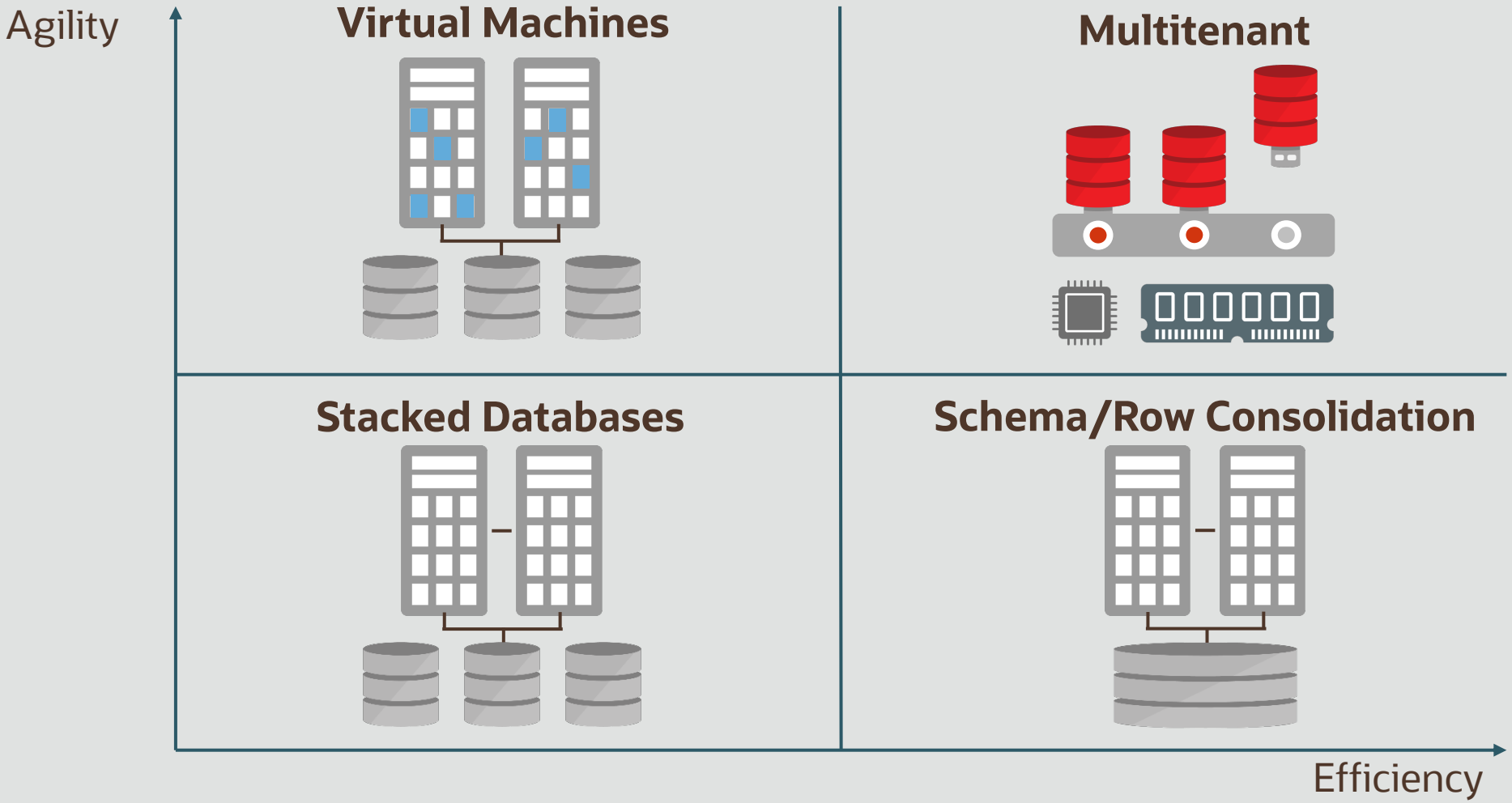
Overview

- MultiTenant
- MultiModel - JSON
- InMemory

Overview

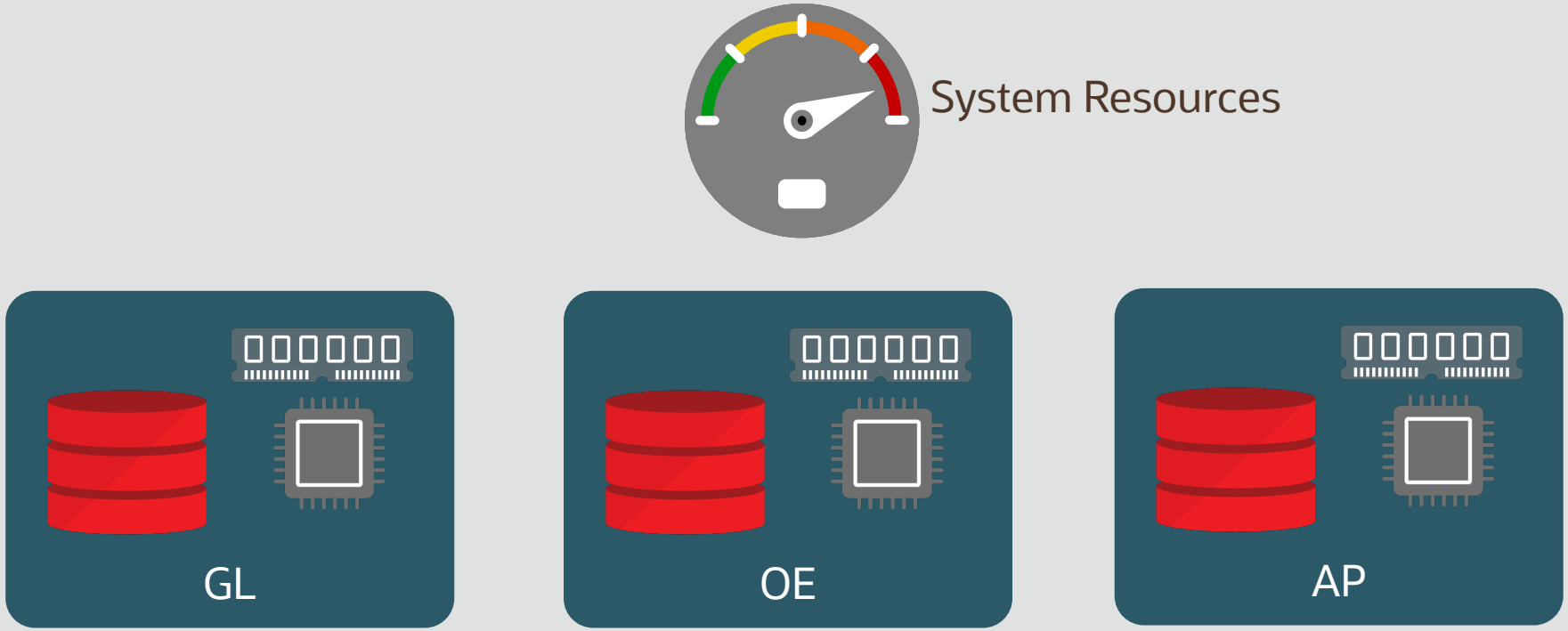
- MultiTenant
- Multimodel - JSON
- InMemory

Comparing Database Consolidation Architectures



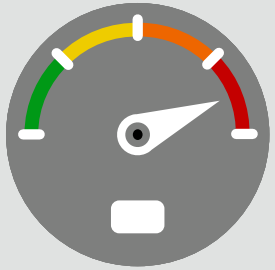
Classical Oracle Database Architecture

Requires memory, processes and database files

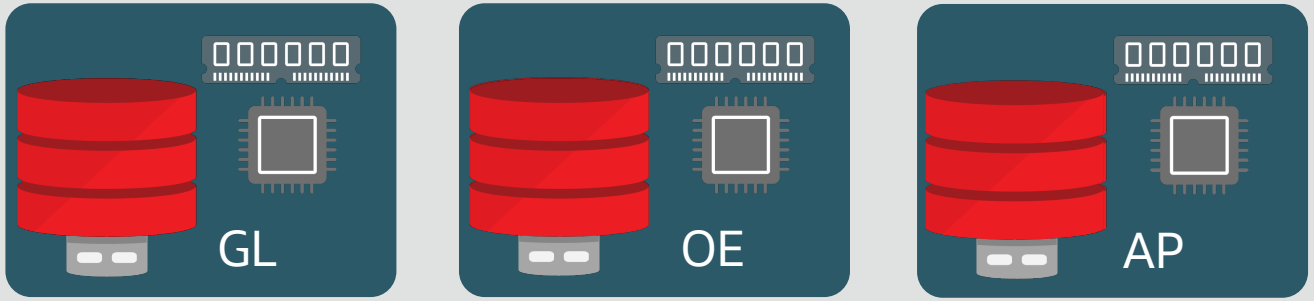


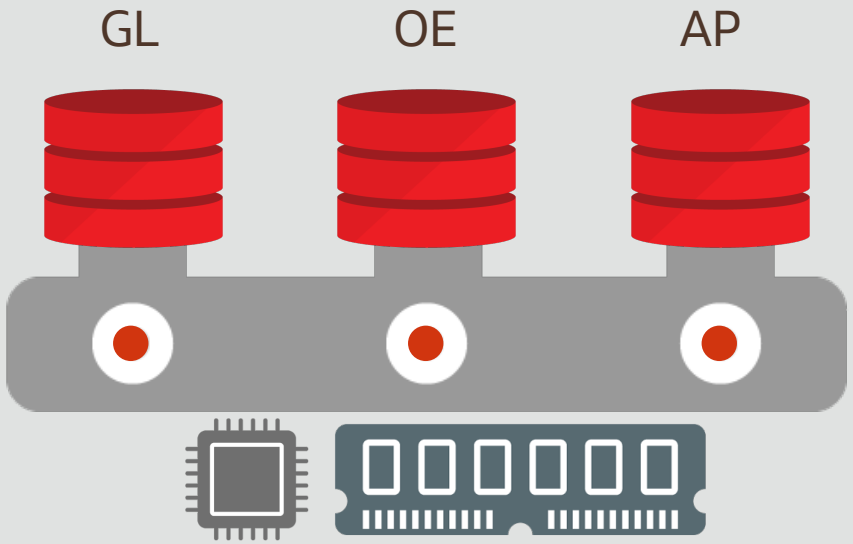
Multitenant Architecture

Memory and processes required at container level only



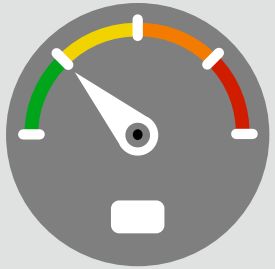
System Resources



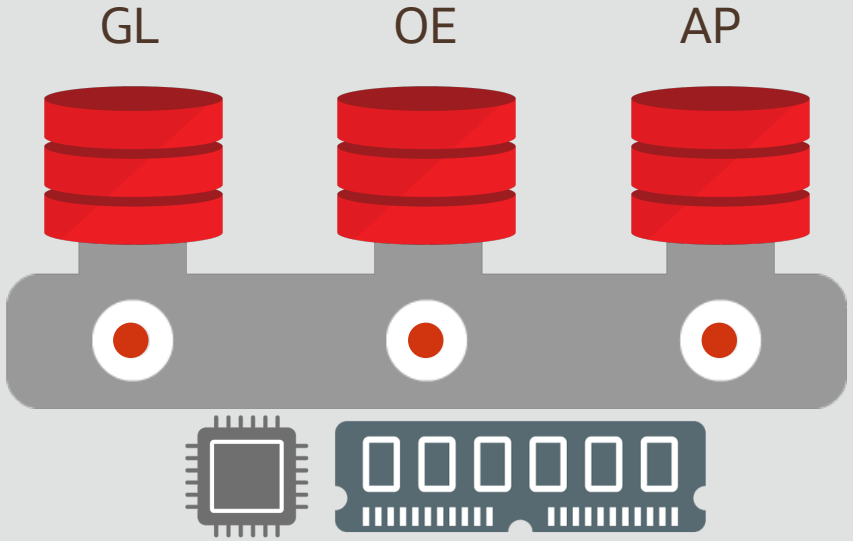


Multitenant Architecture

More efficient utilization of system resources



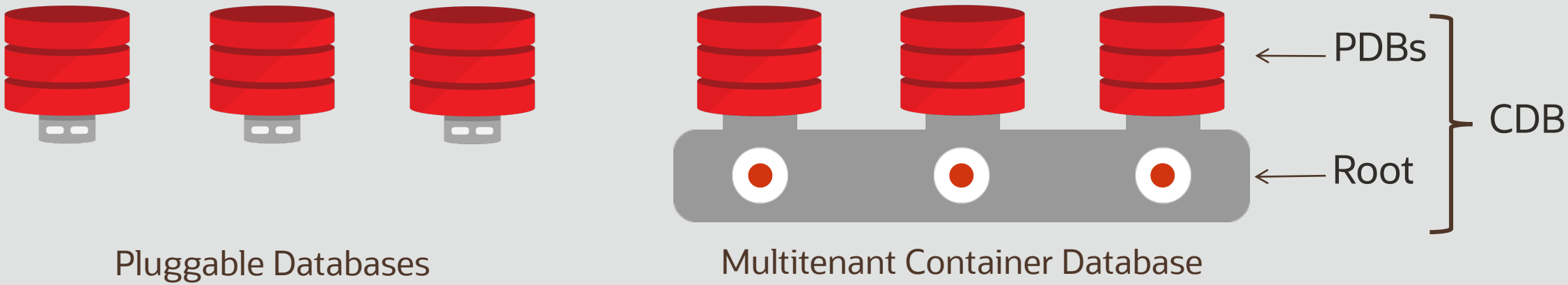
System Resources



Multitenant Architecture

Components of a Multitenant Container Database (CDB)

- Up to 4k PDBs per CDB

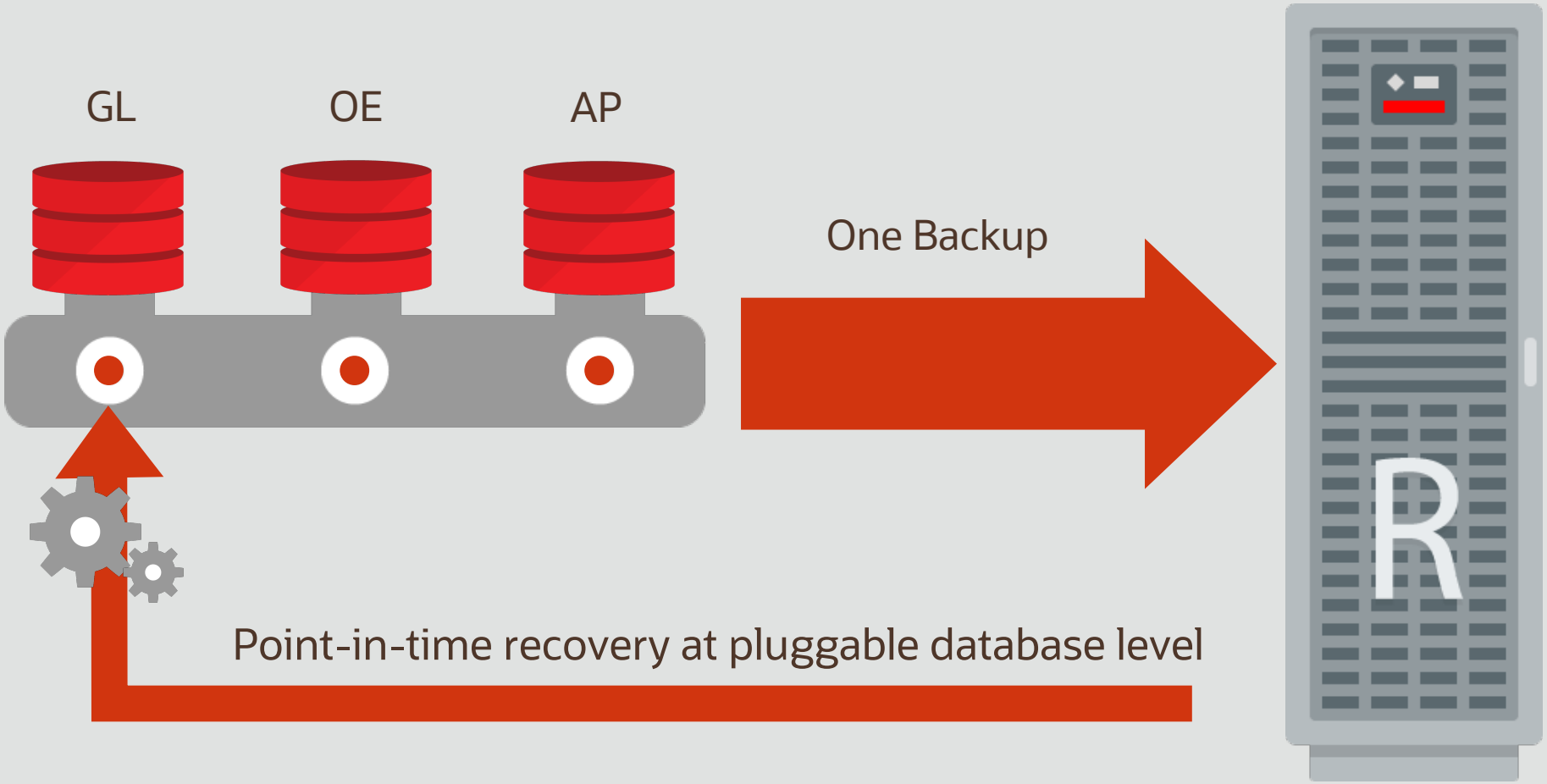


Pluggable Databases

Multitenant Container Database

Manage Many Databases as One

Backup databases as one; recover at pluggable database Level

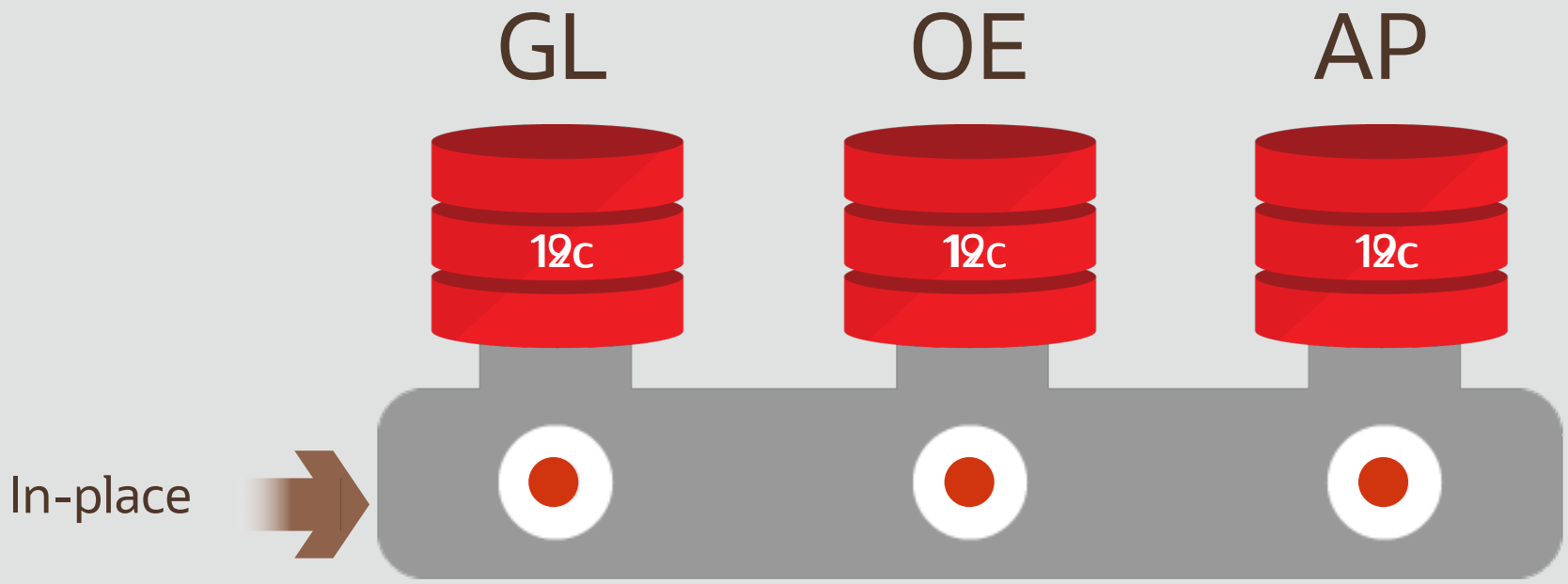


Manage Many Databases as One with Multitenant

One standby database covers all pluggable databases

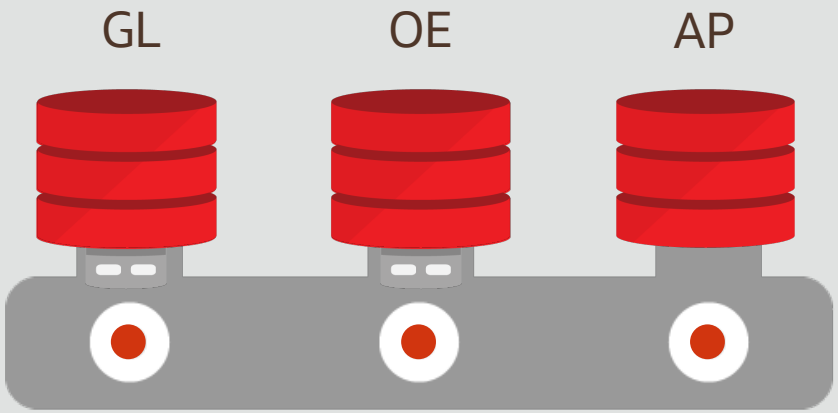
Simplified Patching and Upgrades

Apply changes once, all pluggable databases updated

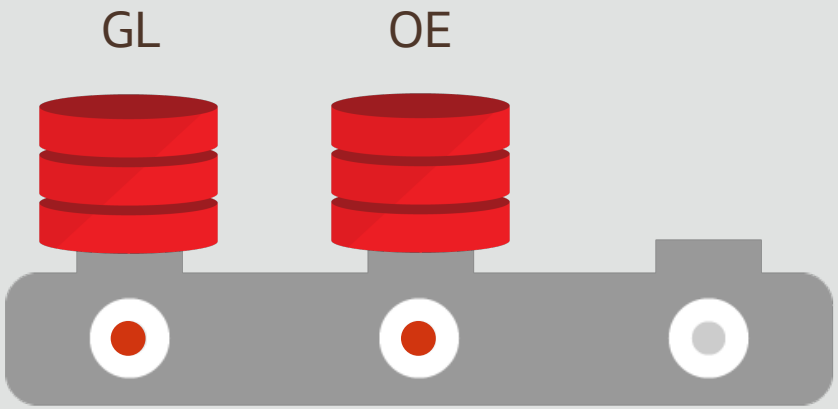


Simplified Patching and Upgrades

Flexible choice when patching & upgrading databases

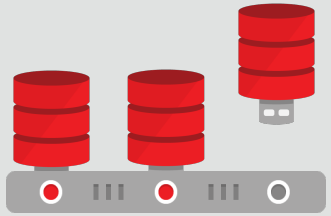


Original Container Database 19.3



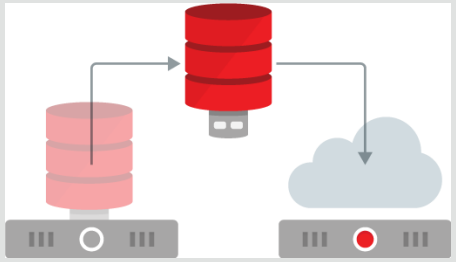
Upgraded Container Database 19.4

Multitenant



- Container managed database virtualization
- Manage Many as one
  - Patching, Backup, Security, Online Cloning, Online Relocation
- Software as Service
  - Shared metadata, Data location transparency

New in 12.2, 18c, 19c



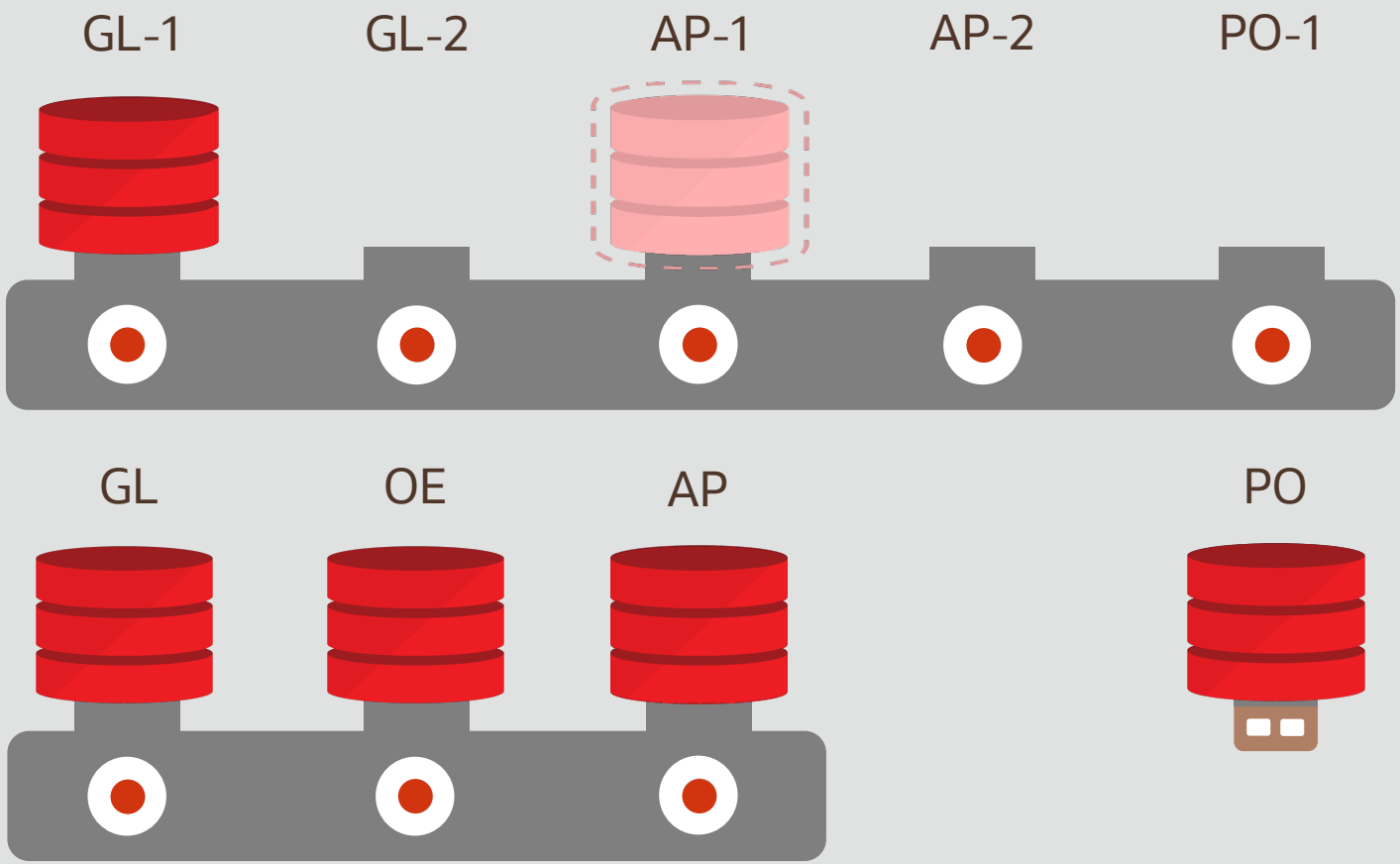
- 12.2
  - Online cloning & relocation
  - Incremental refresh of test/dev master
  - Application containers
- 18c
  - Transportable backups
  - Snapshot carousel
  - Refreshable PDB switchover
- 19c
  - RAT and ADDM at PDB level

Application Container

- Application Container comprises
  - Application Root (Master)
  - Application PDBs (for each Tenant)
  - Application Seed (for provisioning)
- PDBs share application objects
  - Code, metadata and data
- Further simplifies management
  - Apply updates to application container
  - Sync tenant PDBs from central master
- Suitable for all applications
  - SaaS, franchise, divisional, etc.

Multitenant for Provisioning

Fast cloning of PDBs

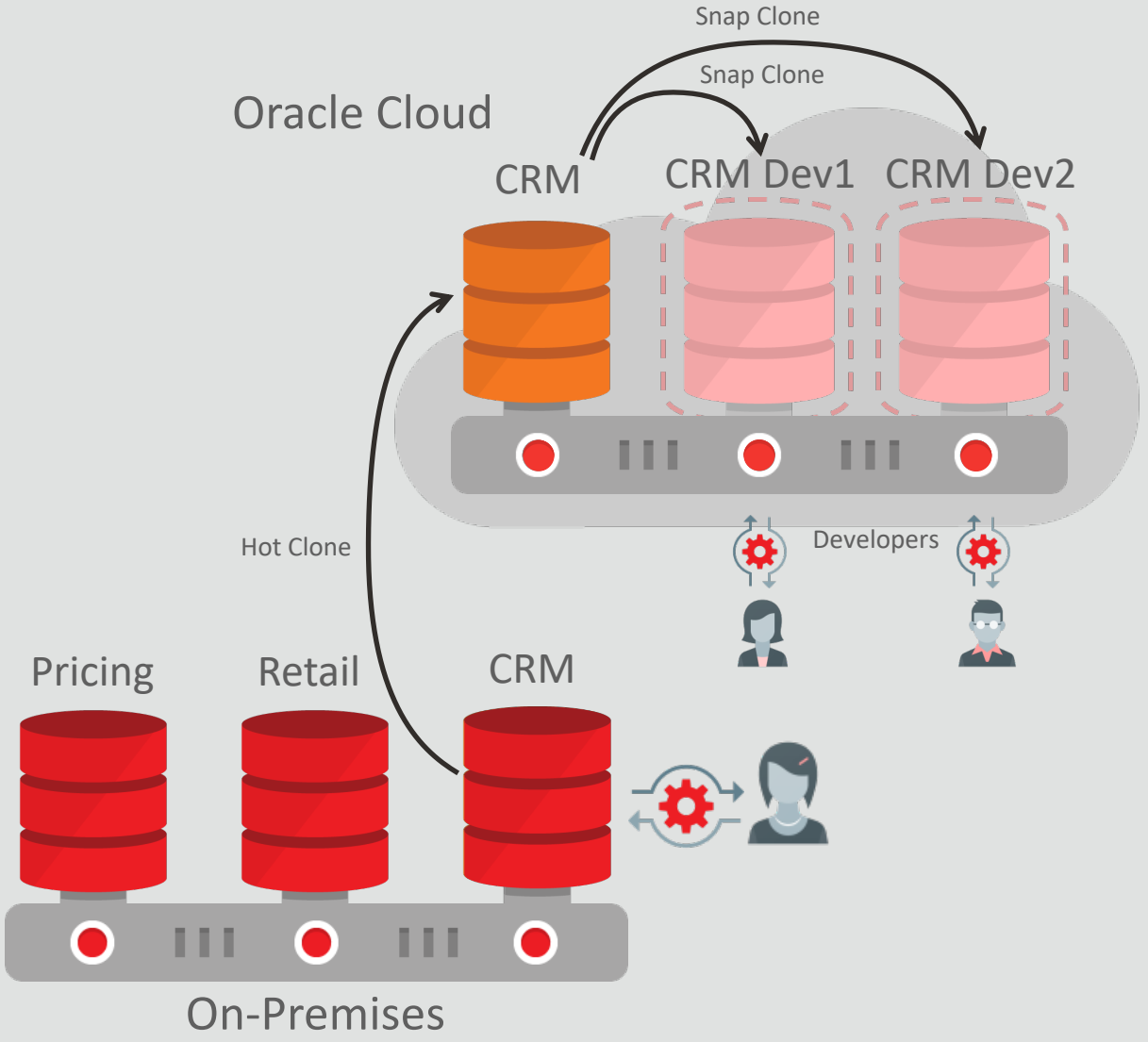


- PDBs can be cloned from within the same CDB
- PDBs can be cloned from remote CDBs
- PDBs can be cloned from non-CDBs
- Thinly provision *snapshot* clones in seconds

PDB Hot Clone

PDB Hot Clone

Online test master instantiation



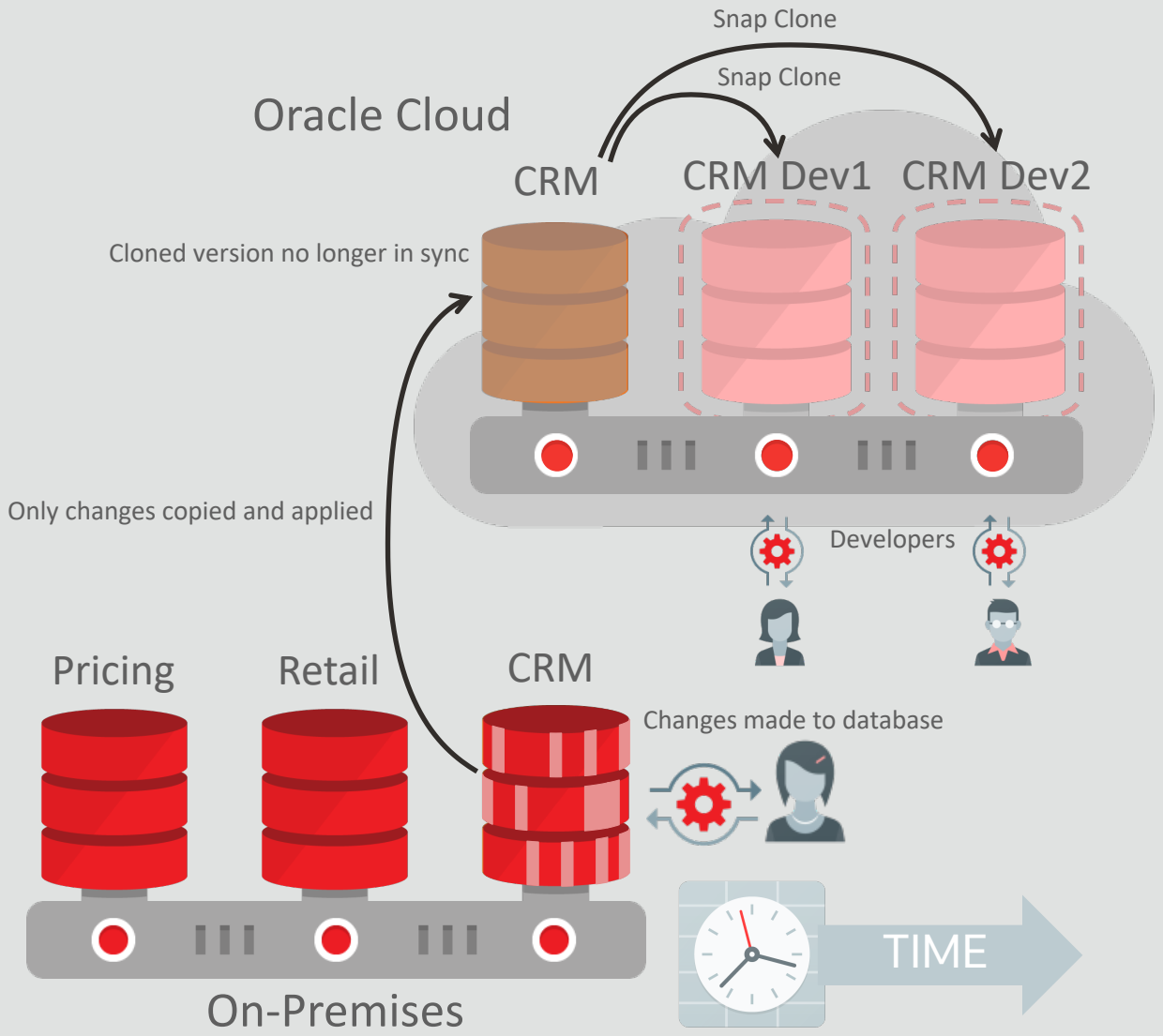
Refreshable PDB

PDB Hot Clone

- Online test master instantiation

Refreshable PDB

- Incremental refresh of clone with latest data



Online PDB Relocation

PDB Hot Clone

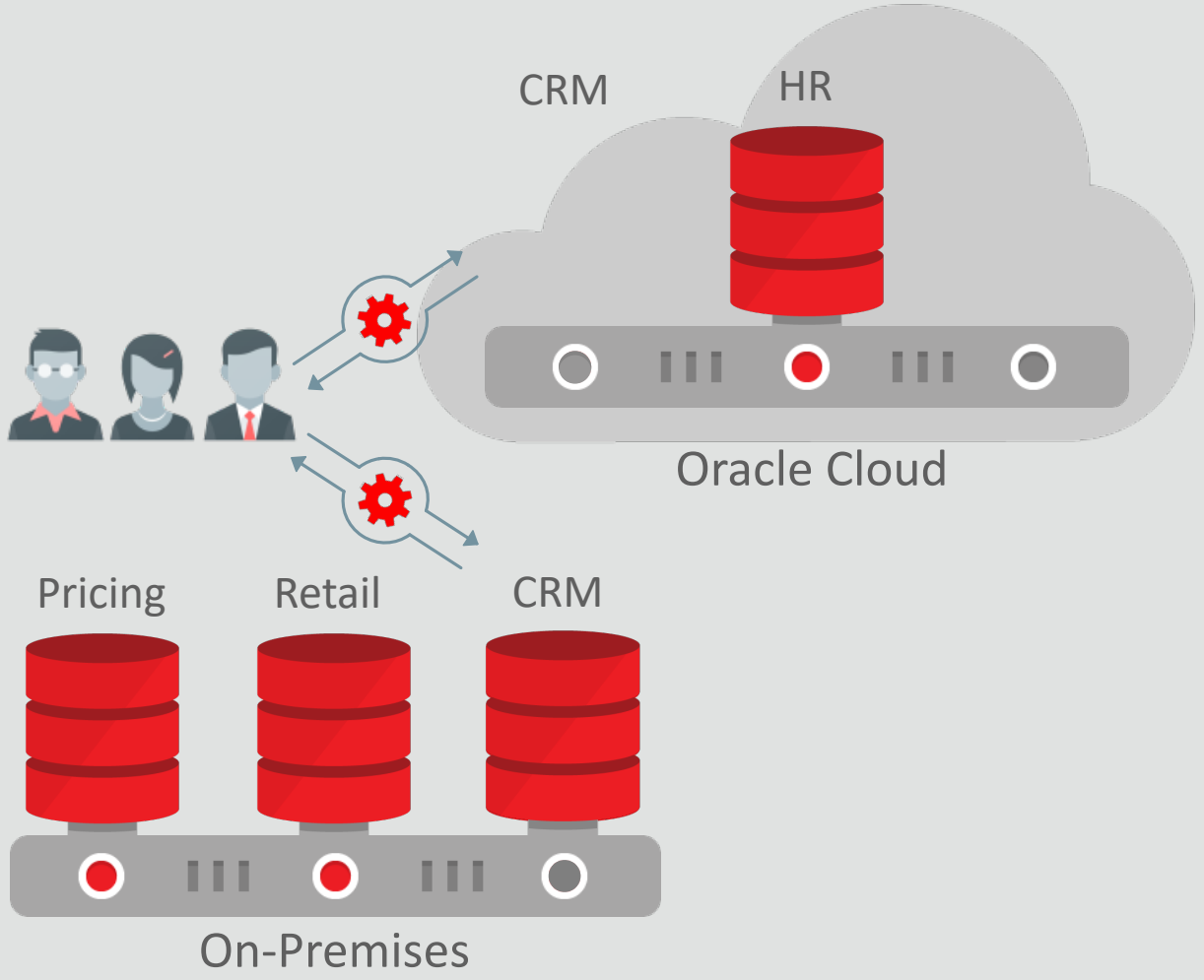
- Online test master instantiation

Refreshable PDB

- Incremental refresh of clone with latest data

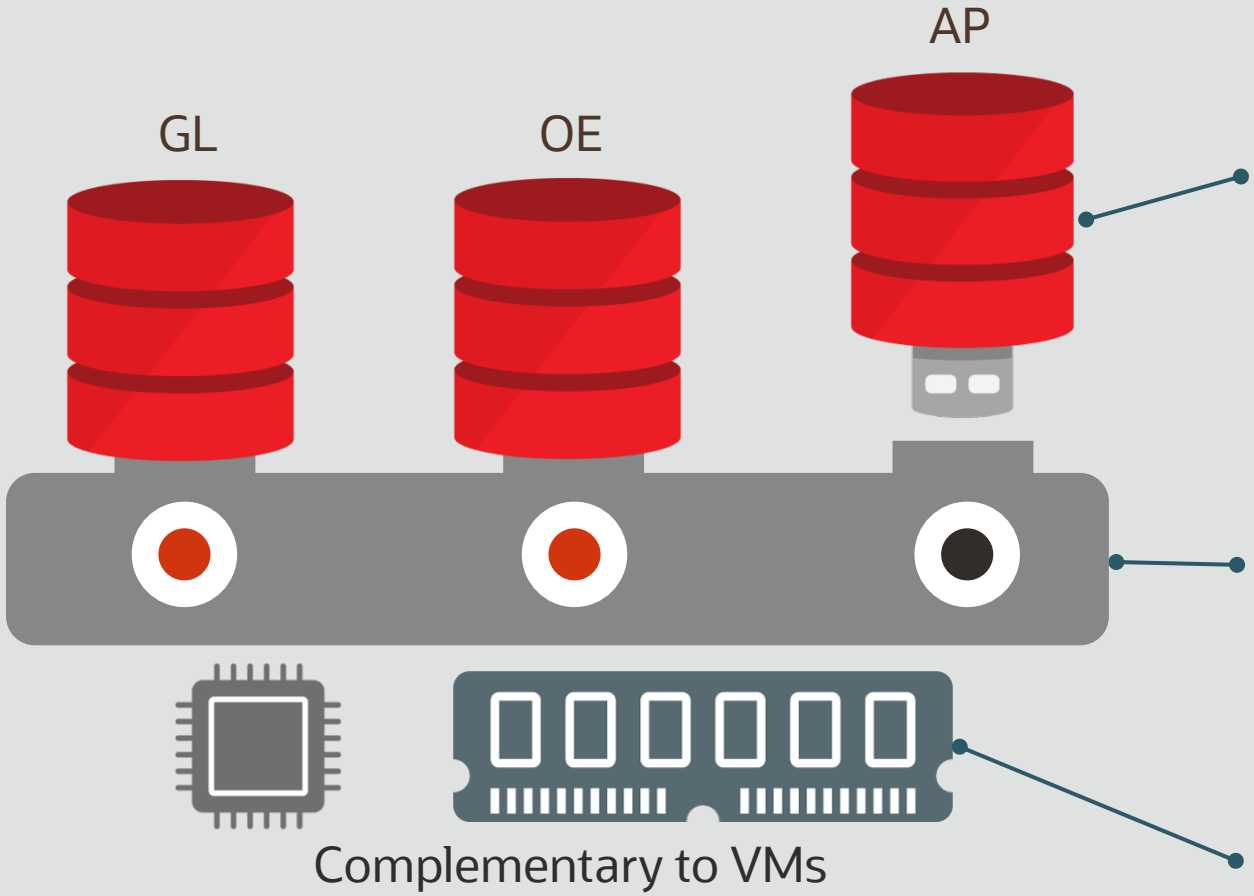
Online PDB Relocation

- Relocate with no downtime



Advantages of Multitenant Architecture

Isolation and agility with economies of scale



Self-contained PDB for each application

- Applications run unchanged
- Rapid provisioning (via clones)
- Portability (via pluggability)

Common operations performed at CDB level

- Manage many as one (upgrade, HA, backup)
- Granular control when appropriate

Shared memory and background processes

- More applications per server

Overview

- MultiTenant
- Multimodel - JSON
- InMemory

What is JSON?

JSON stands for JavaScript Object Notation

It's a "lightweight", readable data interchange format that's language independent

Most popular data format for new web applications

Instead of creating an entity relationship model to define all of the data the application needs and then mapping it to a set of relational tables

Storing JSON documents in the database greatly simplifies application development as the same schema-less data representation can be use in Application and the Database

What is JSON?


```
{"id":1,
 "name":"Century 16",
 "location":{"street":"Main St",
             "city":"Redwood",
             "zipCode":"94607",
             "state":"CA",
            "phoneNumber":null
             },
"ticketPrice":{"adultPrice":14.95,
               "childPrice":9.95,
               "seniorPrice":9.95
              }
}
```

- A data format that consists of one or more **name value pairs** enclosed in curly brackets
- The **name** is always a string and is separated from the value by a colon
- A **value** can be a number, string, true, false null, an object or array
  - E.g. location is an **object** as it has random set of name value pairs nested inside , enclosed in { }
  - An **array** is an ordered list of related items which could be JSON objects and is enclosed in [ ]
- Each pair is separated by a comma

Storing JSON in the Oracle Database

Table containing JSON documents

```
CREATE TABLE theater
 (
   theater_id     VARCHAR2(255),
   json_document  BLOB


 );
```

- Oracle stores JSON in table columns
  - No special data type
  - Can be VARCHAR2, BLOB or CLOB
- JSON supported by all Oracle features
  - Analytics, Encryption, In-Memory, RAC, Replication, Parallel SQL, …
  - Plus can index any JSON element

Storing JSON in the Oracle Database

Table containing JSON documents

```
CREATE TABLE theater
 (
   theater_id    VARCHAR2(255),
   json_document BLOB
   CONSTRAINT is_json CHECK
    (json_document IS JSON)
 );
```

```
{id:1,
 name:"Century 16",
}
```

```
{"id":1,
 "name":"Century 16",
}
```

- Oracle stores JSON in table columns
  - No special data type
  - Can be VARCHAR2, BLOB or CLOB
- JSON supported by all Oracle features
  - Analytics, Encryption, In-Memory, RAC, Replication, Parallel SQL, …
  - Plus can index any JSON element
- `IS JSON` check constraint enforces lax JSON syntax by default
  - Does not require NAME attributes to be in double quotes

Storing JSON in the Oracle Database

Which data type to pick?

VARCHAR2

- Best performance, easy to retrieve via SELECT
- Limited max size of 32k only (with MAX_STRING_SIZE=EXTENDED)

BLOB

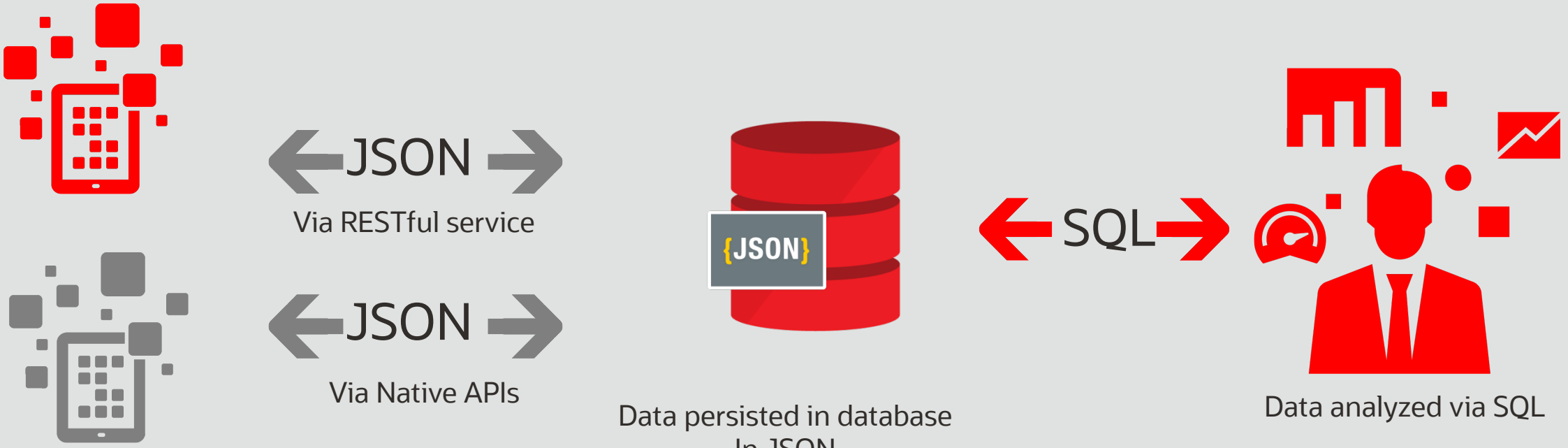
- Best LOB performance but not as fast as VARCHAR2
- Unlimited size, not as easy to retrieve via SELECT
- No potential characterset conversion

CLOB

- Unlimited size, easy to retrieve via SELECT
- Potential characterset conversion (from 1 byte UTF-8 to 2 byte USC-2, double space)
- Potential bigger size on disk

Oracle Database as a Document Store

Flexible Schema development



JSON Support in Oracle Database

Fast Application Development + Powerful SQL Access

{JSON}

Application developers:
Access JSON documents using RESTful API

```
PUT /my_database/my_schema/customers HTTP/1.0
Content-Type: application/json
Body:
{
 "firstName": "John",
 “lastName”: "Smith",
 "age": 25,
 "address": {
      "streetAddress": "21 2nd Street",
      "city": "New York",
      "state": "NY",
      "postalCode": "10021“,
      "isBusiness" : false  },
  "phoneNumbers": [
      {"type": "home",
       "number": "212 555-1234“ },
      {"type": "fax",
       "number": "646 555-4567“ }  ]
}
```

Oracle Database 12c

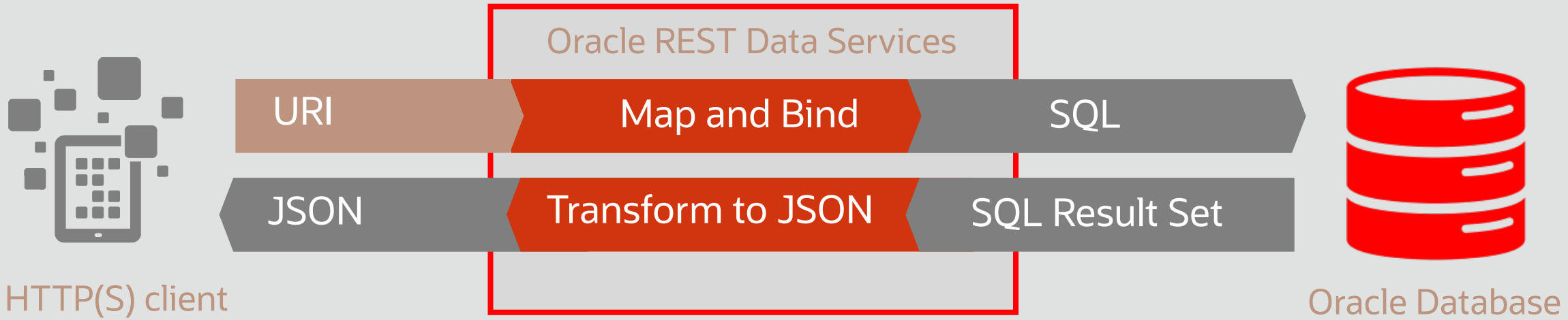
SQL Developers and Analytical tools:
Query JSON using SQL

```
select
  c.json_document.firstName,
  c.json_document.lastName,
  c.json_document.address.city
from customers c;

firstName    lastName     address.city
-----------  -----------  --------------
“John”       “Smith”      “New York”
```

Oracle REST Data Services

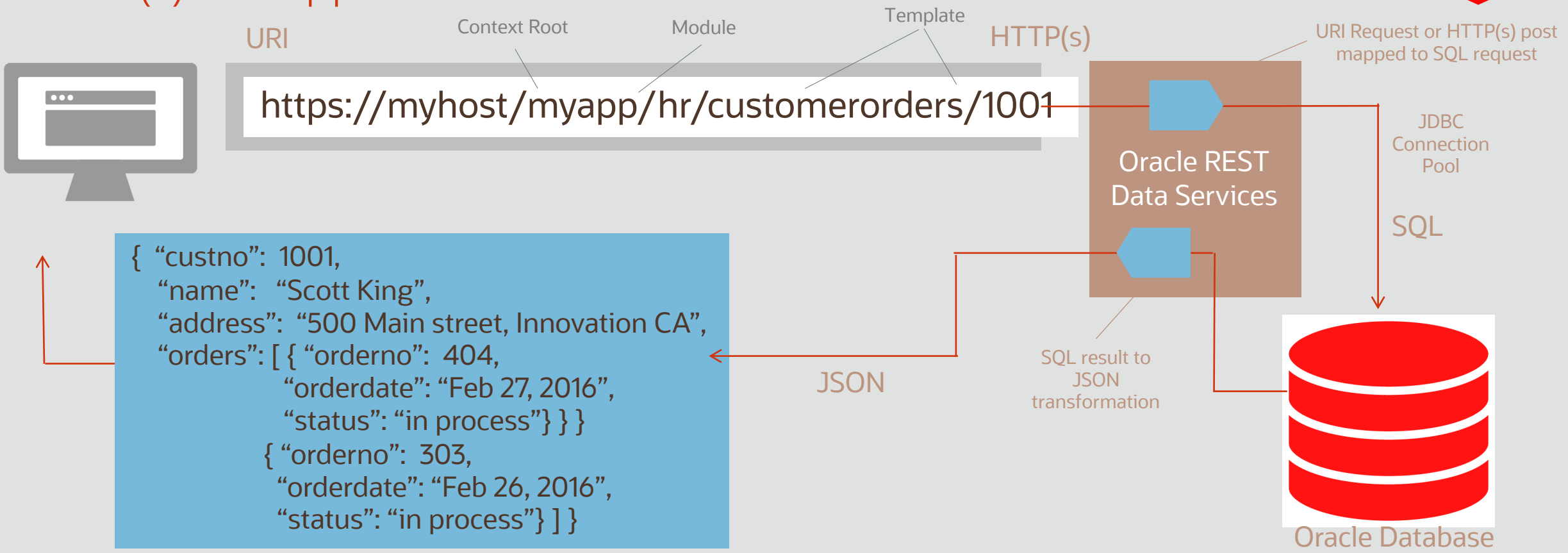
Serving JSON results from relational data



- Data stored in standard relational tables and columns
- Oracle REST Data Services (ORDS) Developer defines URI<>SQL mapping
- App Developer calls named URI over HTTP(S) gets and posts

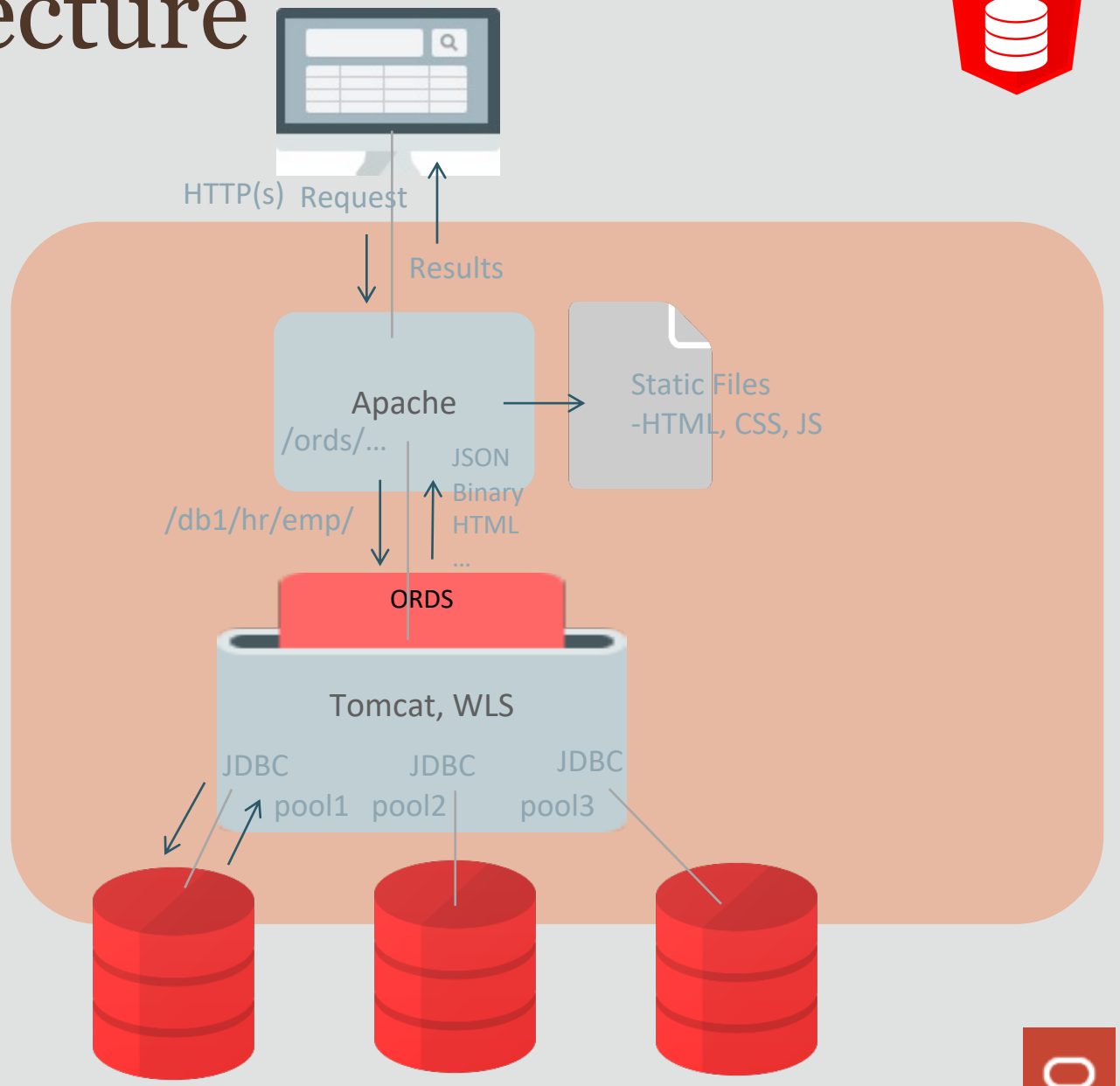
Oracle REST Data Services

HTTP(s) API App-Dev with Relational Tables in Oracle Database



ORDS maps standard URI requests to corresponding relational SQL (not schemaless): e.g. SQL SELECT from customers and orders table.
ORDS also transforms the SQL results into JavaScript Object Notation (JSON), other formats include HTML, binary and CSV.
Fully committed to supporting any and all standards required by Fusion / SaaS / FMW; we are actively engaged in the ongoing dialog.

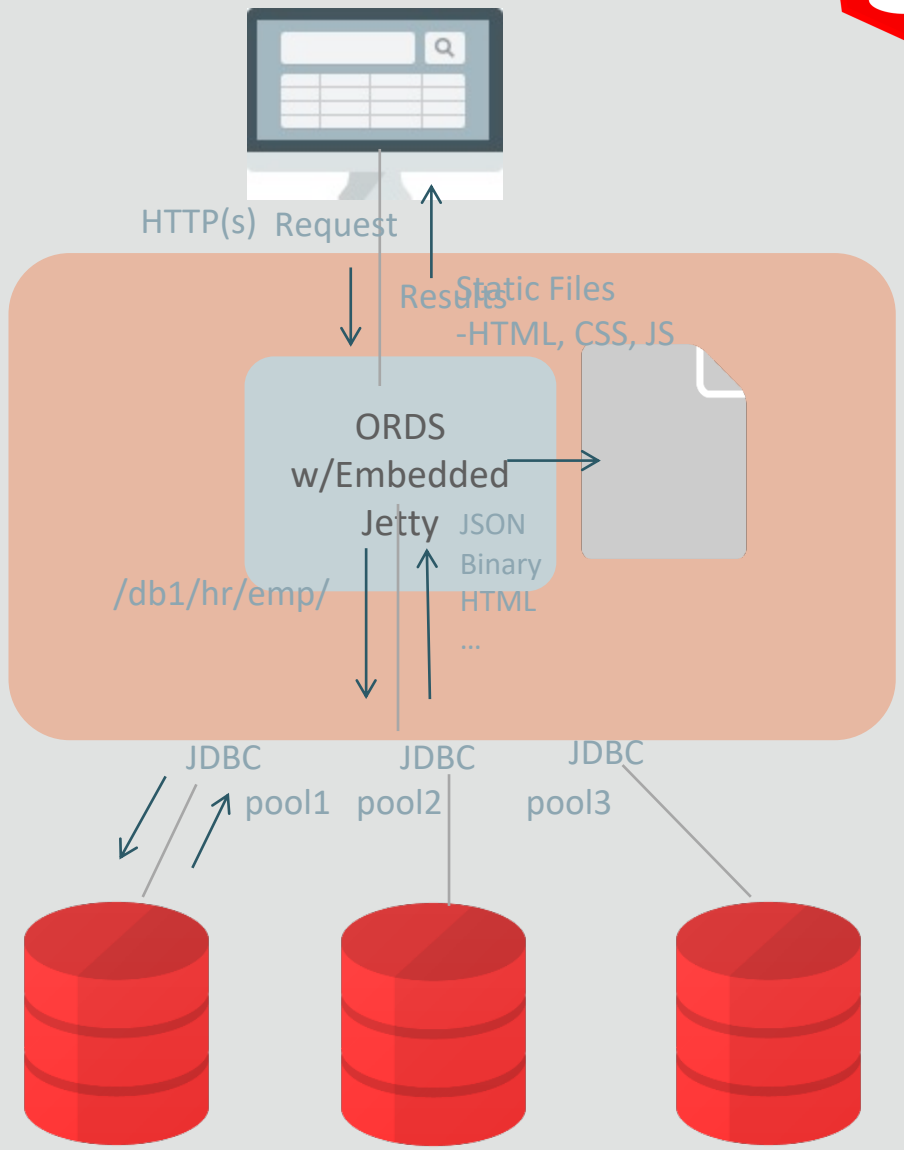
ORDS Typical Architecture



- Standard webserver layout
- Implements Java Servlet
- Deploys to Tomcat or WLS
- Also Supported:
  - Standalone mode (Jetty)

ORDS Standalone Deployments

- ORDS running as a OS process
- Eclipse Jetty Webserver & Servlet Container
- Supported for Production
- Offers much fewer control, configuration, and management features



SODA: Simple Oracle Document Access

- A simple NoSQL-style API for Oracle
  - Collection Management: Create and drop collections
  - Document Management: CRUD (Create, Retrieve, Update and Delete) operations
  - List and Search: (Query-by-Example) operations on collections
  - Utility and Control: Bulk Insert, index management
- Developers can work with Oracle without learning SQL or requiring DBA support
  - Same development experience as pure-play document stores

Overview

- MultiTenant
- MultiModel - JSON
- InMemory

Reduced Overhead

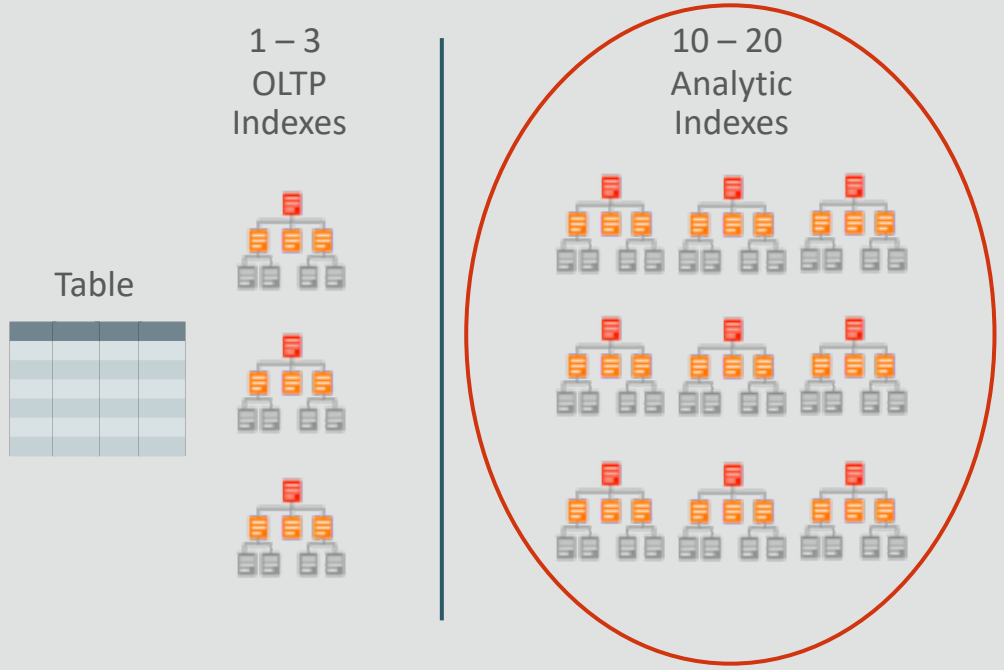
Faster Analytics -- Less Storage Overhead

Analytic indexes can slow down the performance of transactional applications

- Requires significantly more database storage (on costly tier 1 storage)
- Increases overhead due to index maintenance

Database In-Memory allows users to eliminate analytic reporting indexes – without impacting performance

Removing the need for analytic reporting indexes greatly simplifies tuning and reduces ongoing administration

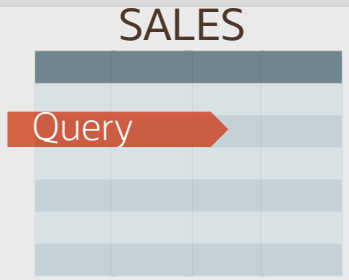


Using Database In-Memory resulted in:

- **Performance Gains: 1.8X to 12X**
- **Space savings and reduced contention on DML by dropping analytic indexes**

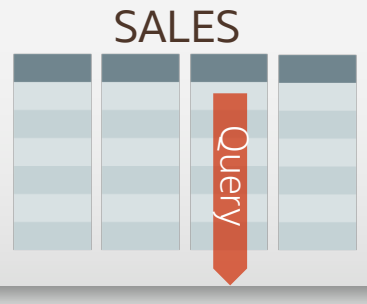
Row Format Databases vs. Column Format Databases

Rows Stored Contiguously



- Transactions run faster on row format
  - Example: Query or Insert a sales order
  - Fast processing few rows, many columns

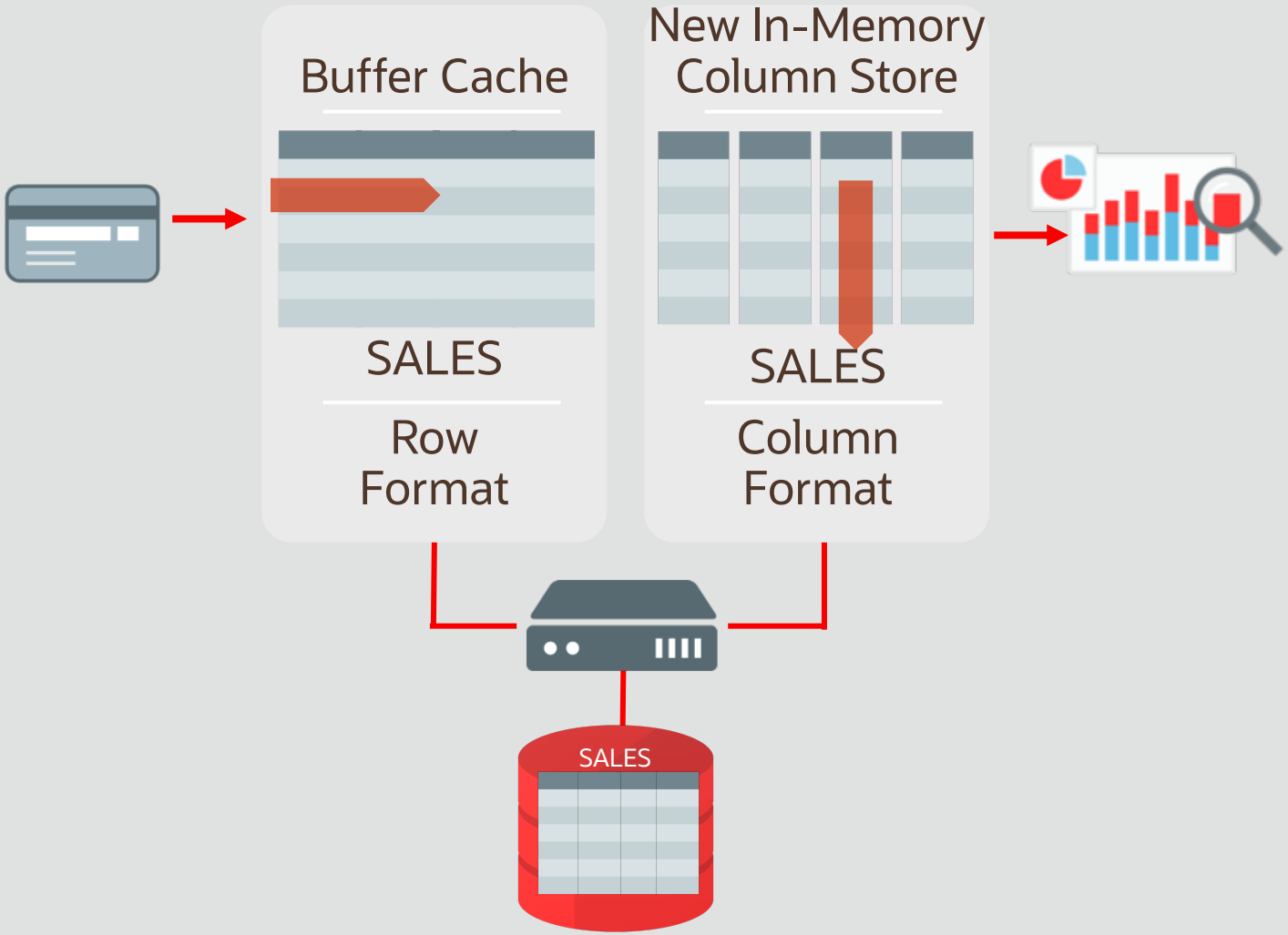
Columns Stored Contiguously



- Analytics run faster on column format
  - Example : Report on sales totals by region
  - Fast accessing few columns, many rows

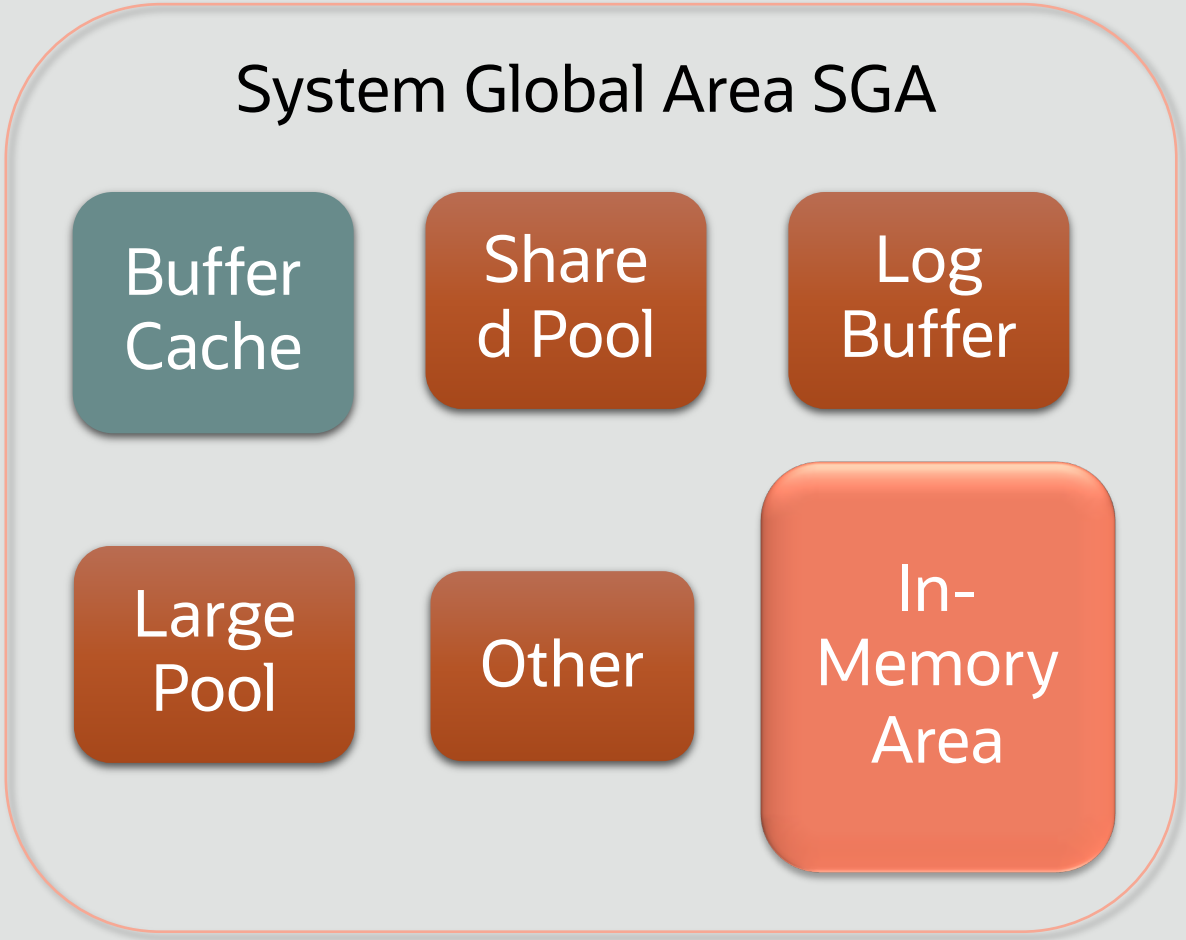
Until Now Must Choose One Format and Suffer Tradeoffs

Breakthrough: Dual Format Database



- BOTH row and column formats for same table
- Simultaneously active and transactionally consistent
- Analytics & reporting use new in-memory Column format
- OLTP uses proven row format

In-Memory Area: Static Area within SGA



- Contains data in the new In-Memory Columnar Format
- Controlled by INMEMORY_SIZE parameter
  - Minimum size of 100MB
- Can be re-sized larger while database is running (12.2)
- SGA_TARGET must be large enough to accommodate In-Memory area

Oracle In-Memory: Simple to Implement

1. Configure Memory Capacity

   ```
   inmemory_size = XXX GB
   ```

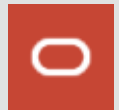
2. Configure tables or partitions to be in memory

   ```
   alter table | partition  …  inmemory;
   ```

3. Later drop analytic indexes to speed up OLTP

Conclusions

MultiTenant + MultiModel JSON + InMemory



Strategy: Oracle Database as a Document Store

Built on Foundation of Oracle Database

- Transactions and consistency
- Advanced SQL engine
- Enterprise-Grade High Availability
- Enterprise-Grade Security
- Scalability and Performance: Exadata and Real Application Clusters
- Oracle Public Cloud Infrastructure
- Sharding

Oracle Database: A multi-model platform

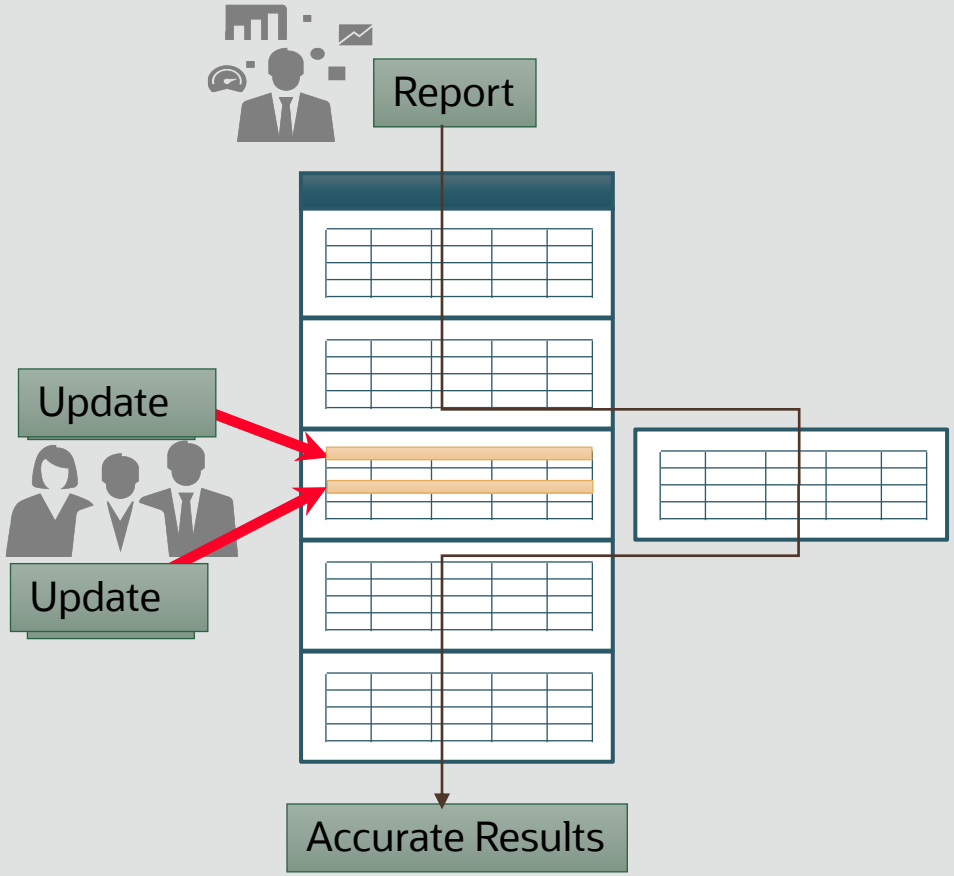
Oracle Database supports multi-model persistence

- Relational
- XML
- JSON
- Text
- Graph & Spatial

Oracle Database provides integrated access to all database objects

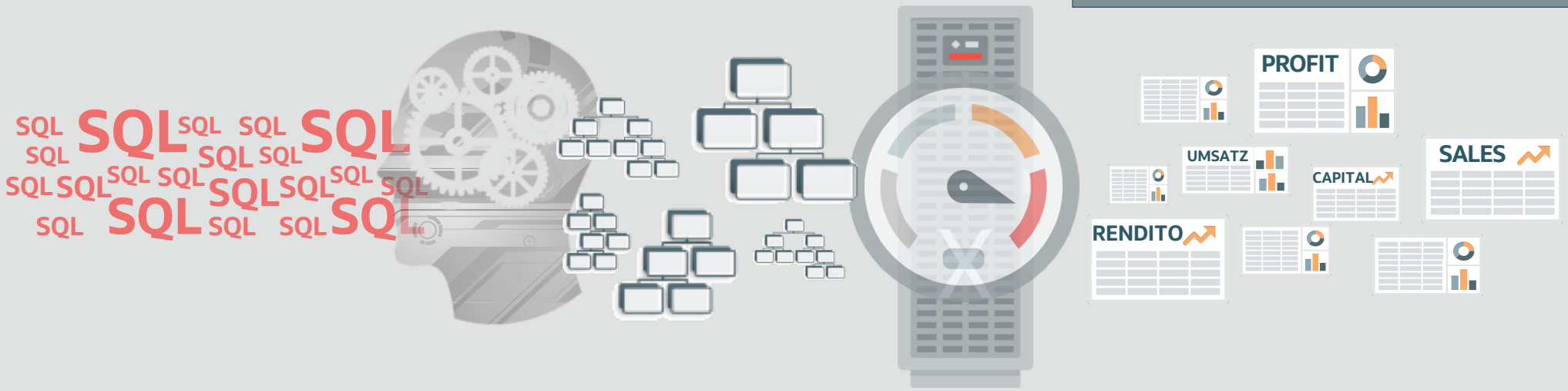
Transactions and consistency

- ACID transactions by default
  - Transactions across multiple documents and multiple collections
    - No hand-coding of two-phase commit required
- Read consistency by default
  - Unique multi-version read consistency model ensures that read queries and updates queries run simultaneously, without blocking and with consistent results



Advanced SQL Engine

Full-featured Query Processing Engine



Comprehensive SQL Language

Any data: Structured, Semi-structured

Sophisticated query optimization

Joins, aggregations, filters, indexes, query transformations

Smart processing

Parallel, In-memory

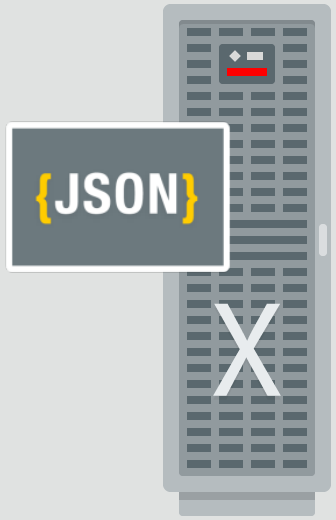
Any Answer

Operational, reporting, analytical, predictive

Query Optimizations for JSON

Exadata Smart Scans

- Exadata Smart Scans execute portions of SQL queries on Exadata storage cells
- JSON query operations 'pushed down' to Exadata storage cells
  - Massively parallel processing of JSON documents



In-Memory Columnar Store

Virtual columns, included those generated using JSON Data Guide loaded into In-Memory Virtual Columns

JSON documents loaded using a highly optimized In-Memory binary format (OSON)

Query operations on JSON content automatically directed to In-Memory