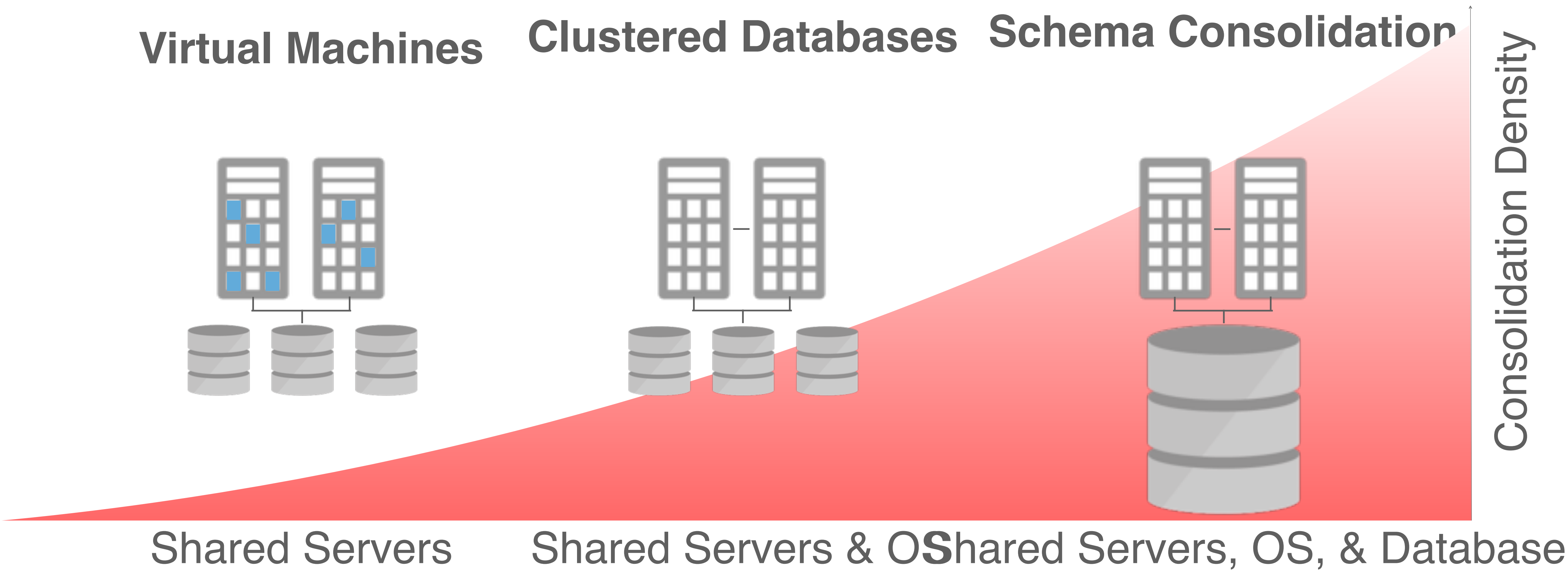
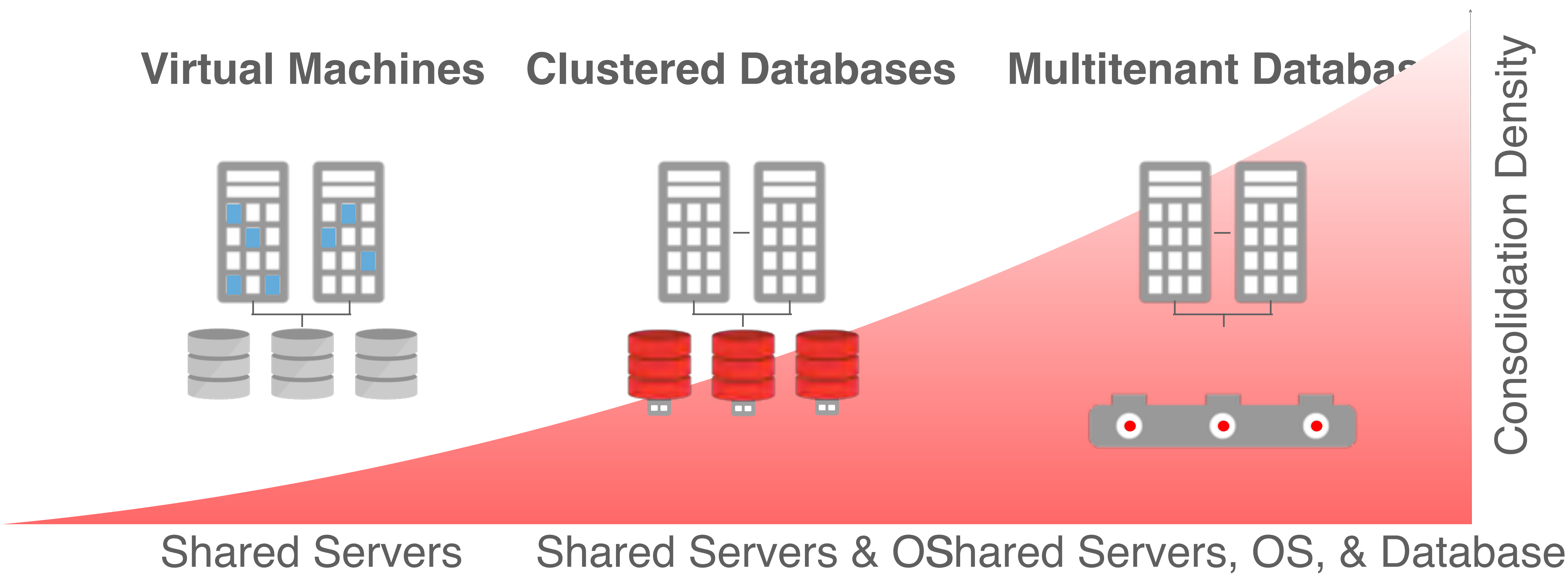
Multitenant Databases

Traditional Consolidation Methods



Consolidation through Multitenancy

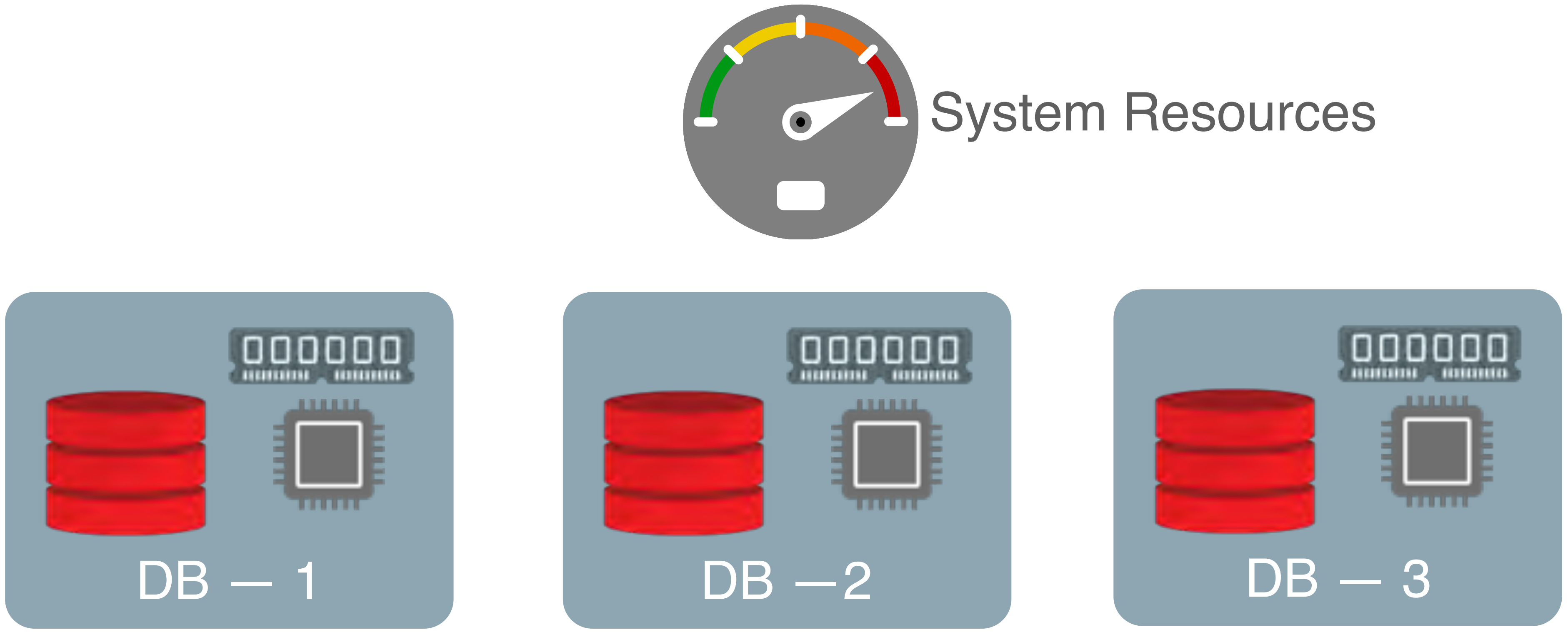
Simplifies consolidation; enables Database as a Service



Sohail Rafiqi

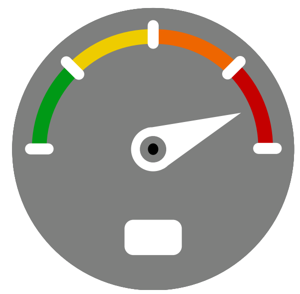
Traditional Database

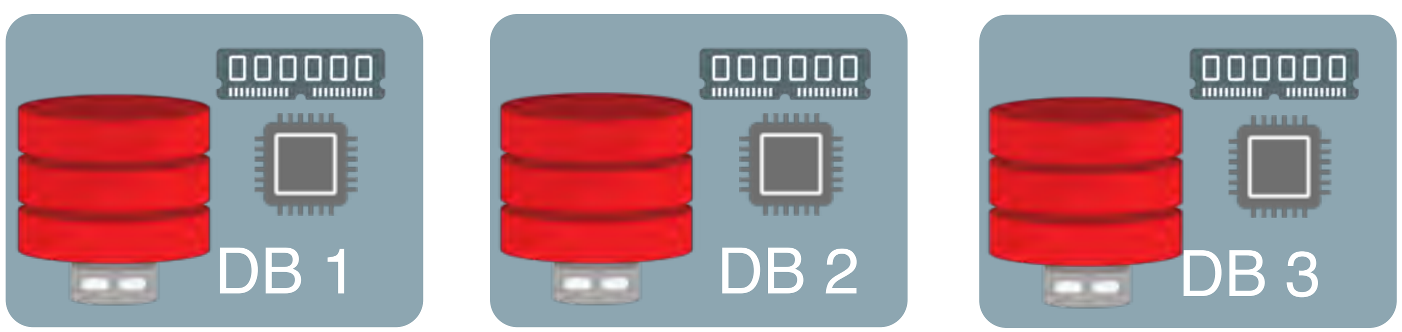
Requires memory, processes and database files

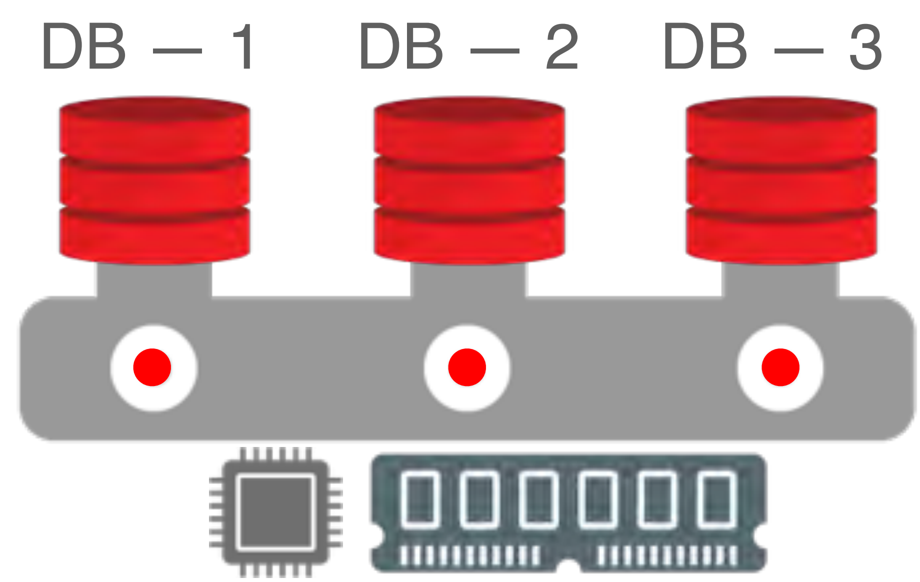


Multitenant

Maximize the resource utilization

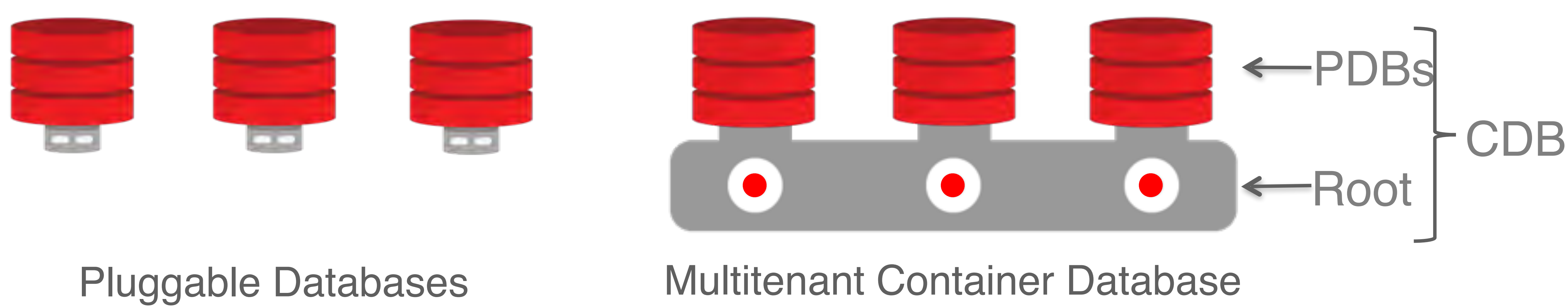

System Resources





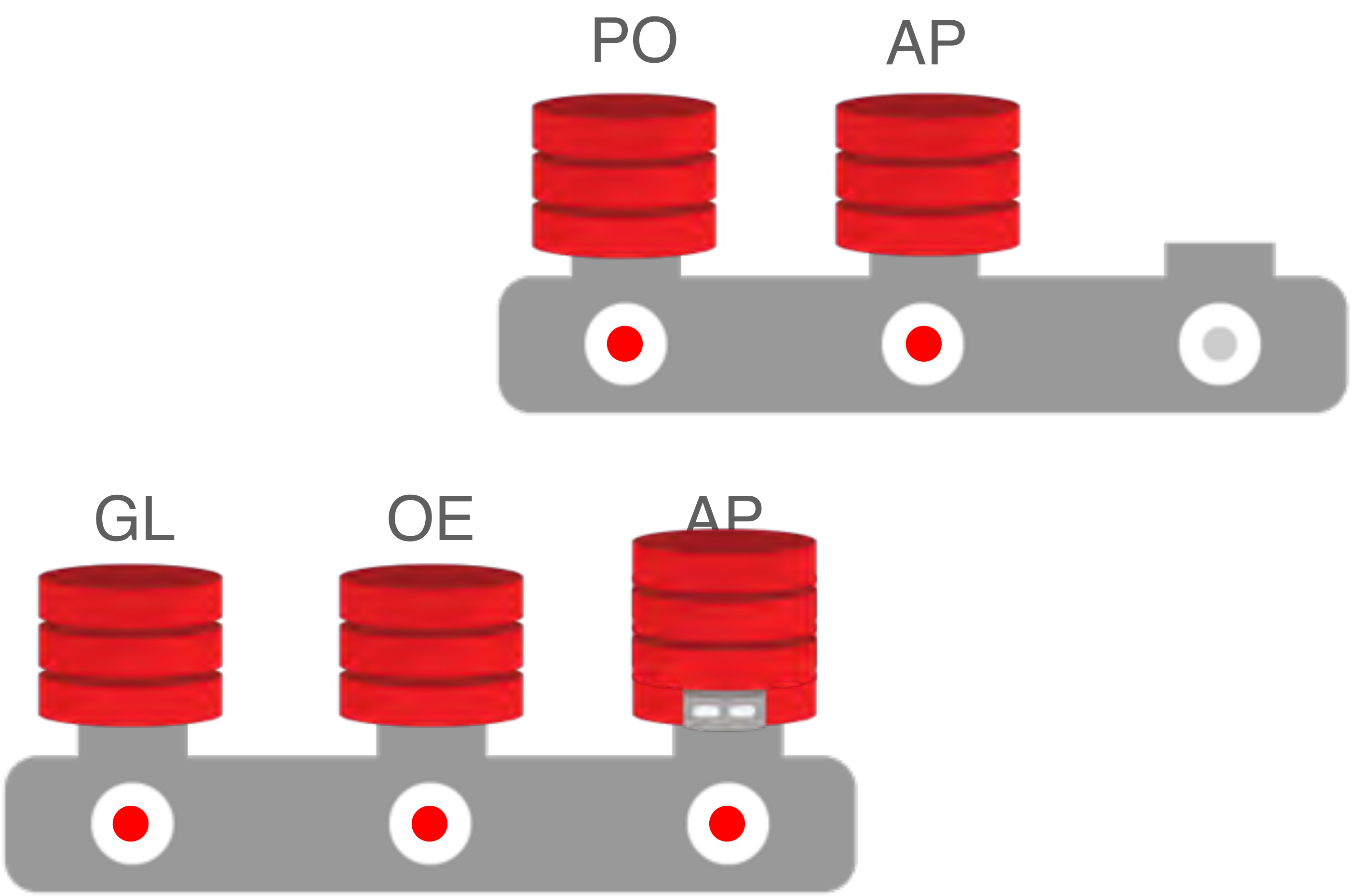
Architecture

Components of a Multitenant Container Database (CDB)



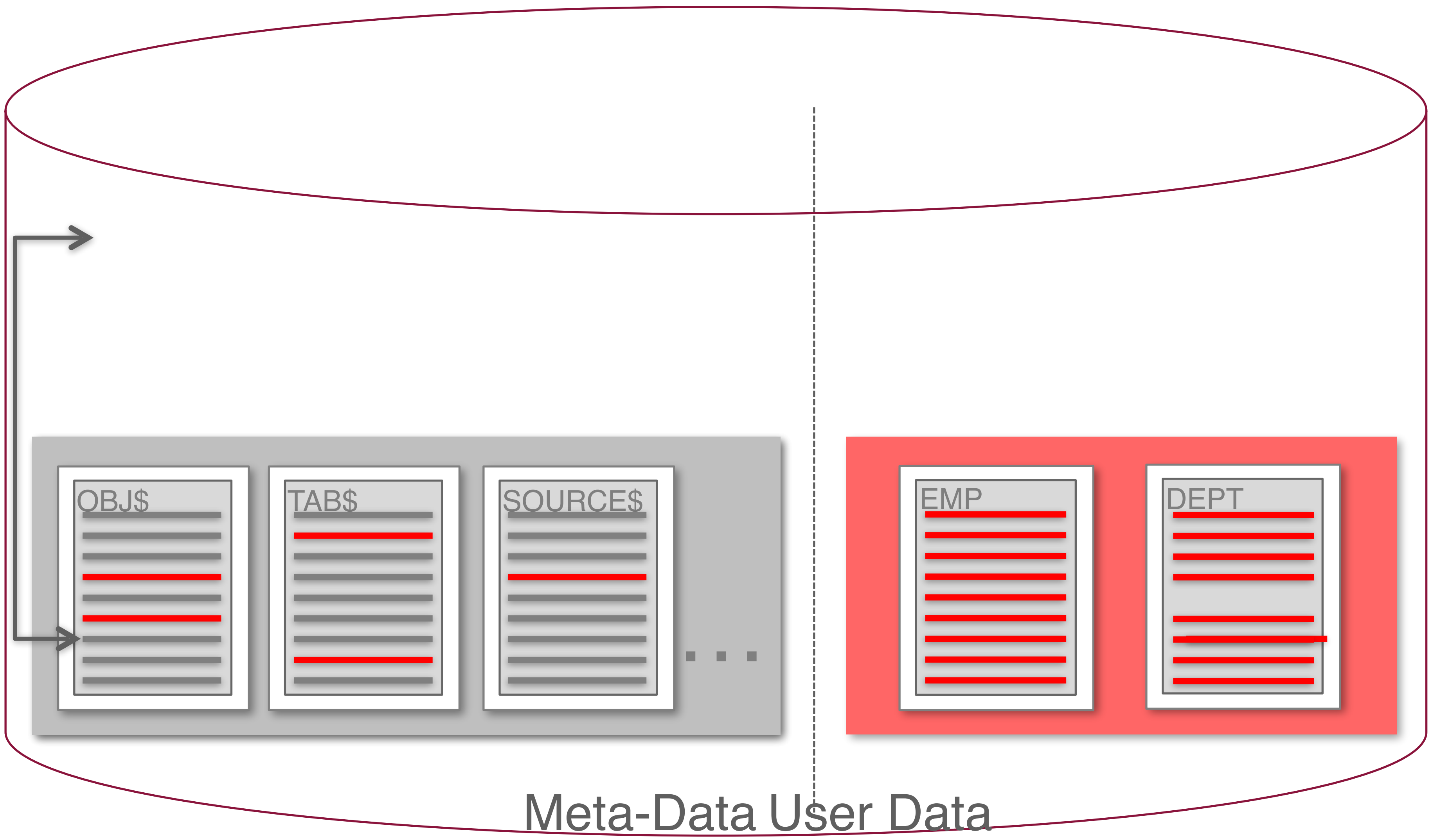
Pluggable Databababasese is a Portable Data

Unplugging a PDB and plugging it in



- Simply unplug from the old CDB…
- …and plug it into the new CDB
- Moving between CDBs is a simple case of moving a PDB's metadata
- An unplugged PDB carries with it lineage, opatch, encryption key info etc.

Data and User Data



- New database contains Oracle meta-data only
- Populate database with user data
  - Oracle and customer meta-data intermingled
  - Portability challenge!
- Multitenant fix: *Horizontally-partitioned data dictionary*
  - Only Oracle-supplied meta-data remains in root

Unplug/Plug using SQL

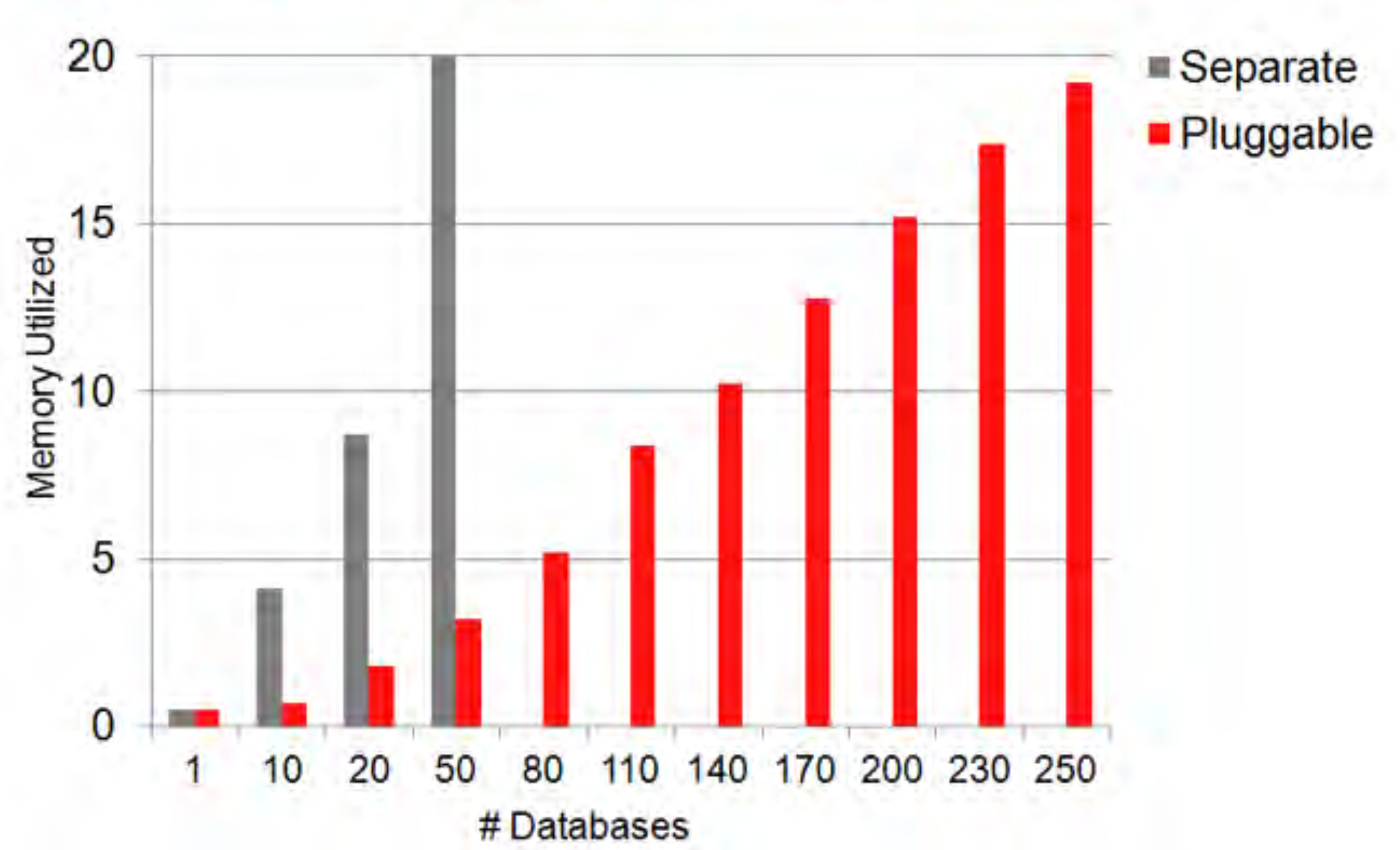
Unplug

```
alter pluggable database OE
unplug into '/u01/app/myDB/data1/…/
oe.xml'

create pluggable database My_PDB
using '/u01/app/data2/…/oe.xml'
```

Multitenant vs Separate Database

- OLTP benchmark comparison
- Only 3GB of memory vs. 20GB memory used for 50 databases
- Multitenant architecture scalable to large # of databases



Files in Container

- Each PDB has its own set of tablespaces including SYSTEM and SYSAUX
- PDBs share UNDO, REDO and control files, (s)pfile
- By default the CDB has a single TEMP tablespace but PDBs may create their own

Users

- Local users are the successors for customer-created users in a non-CDB
- A local user is defined only in a PDB
- A local user can administer a PDB
- A common user is defined in the root
  and is represented in every PDB
- A common user can log into any PDB
  where it has “Create Session” and can therefore administer a PDB
- The system is owned by common users

Common Users and Privileges

- A common user can be granted privileges locally in a PDB (or root) and therefore differently in each container

- A common user can, alternatively, be granted a system privilege commonly – the grant is made in root and every PDB, present and future

- You can create a common role

- A common role can be granted to a common user commonly

- Authorization is checked in the container where the SQL is attempted
  considering only the privileges that the user has in that container

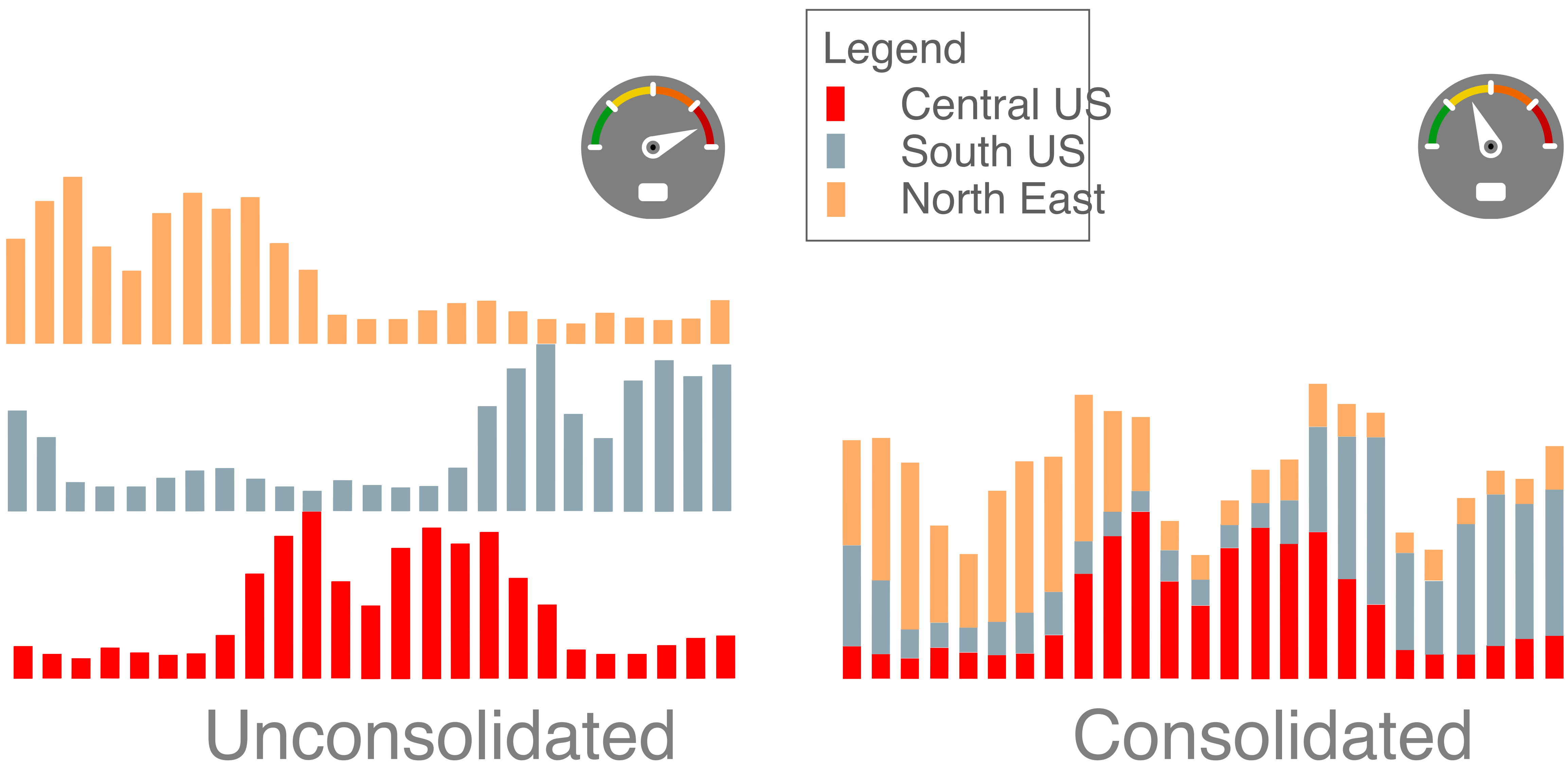
Granting the SysDBA privileges in a PDB

- When a user authorizes AS SYSDBA in a PDB, the effect is contained to within that PDB
- This does not enable the user to get to another container
- Getting to a new container depends only on the privileges
  that the user has in the new container

Creating Common Users

```
create user c##Über_Administrator
identified by pwd container=all
```

Benefits of Consolidation



Sohail Rafiqi

Consolidation Math

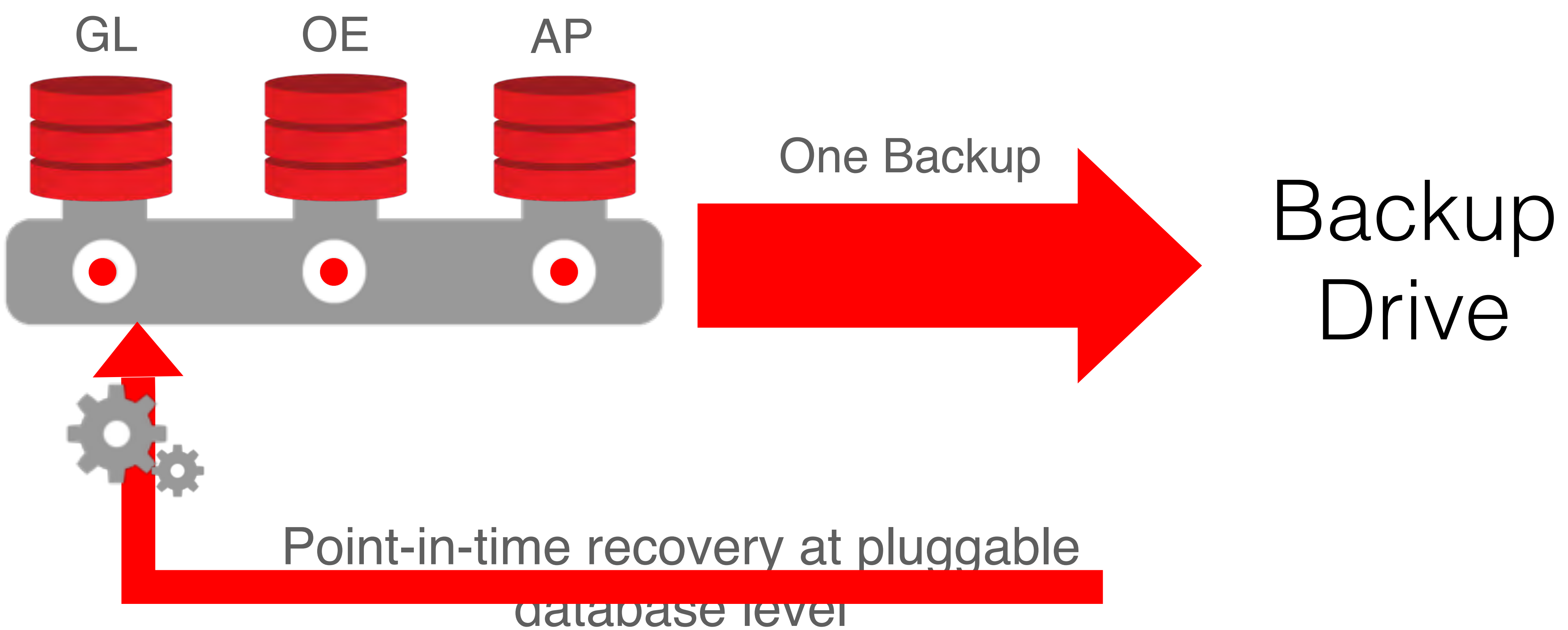
- Simple Example:
- 5-core server
- 60% steady load
  - 2 cores “unused”
- Background overhead 1 core
  - 2 cores required for application processing

Consolidation Math Cont.

- Consolidation Server
- 10 core
- Target utilization 70%
  - 3 cores unused
- Overhad 1 core per CDB
  - 6 core for app processing
  - 2 core required for each application (previous step)
- Multitenant Consolidation
  - 2x the core count
  - 3x the application

Manage Many Databases as One

Backup databases as one; recover at pluggable database Level



Manage Many Databases as One

Sohail Rafiqi

Simplified Patching and Upgrades

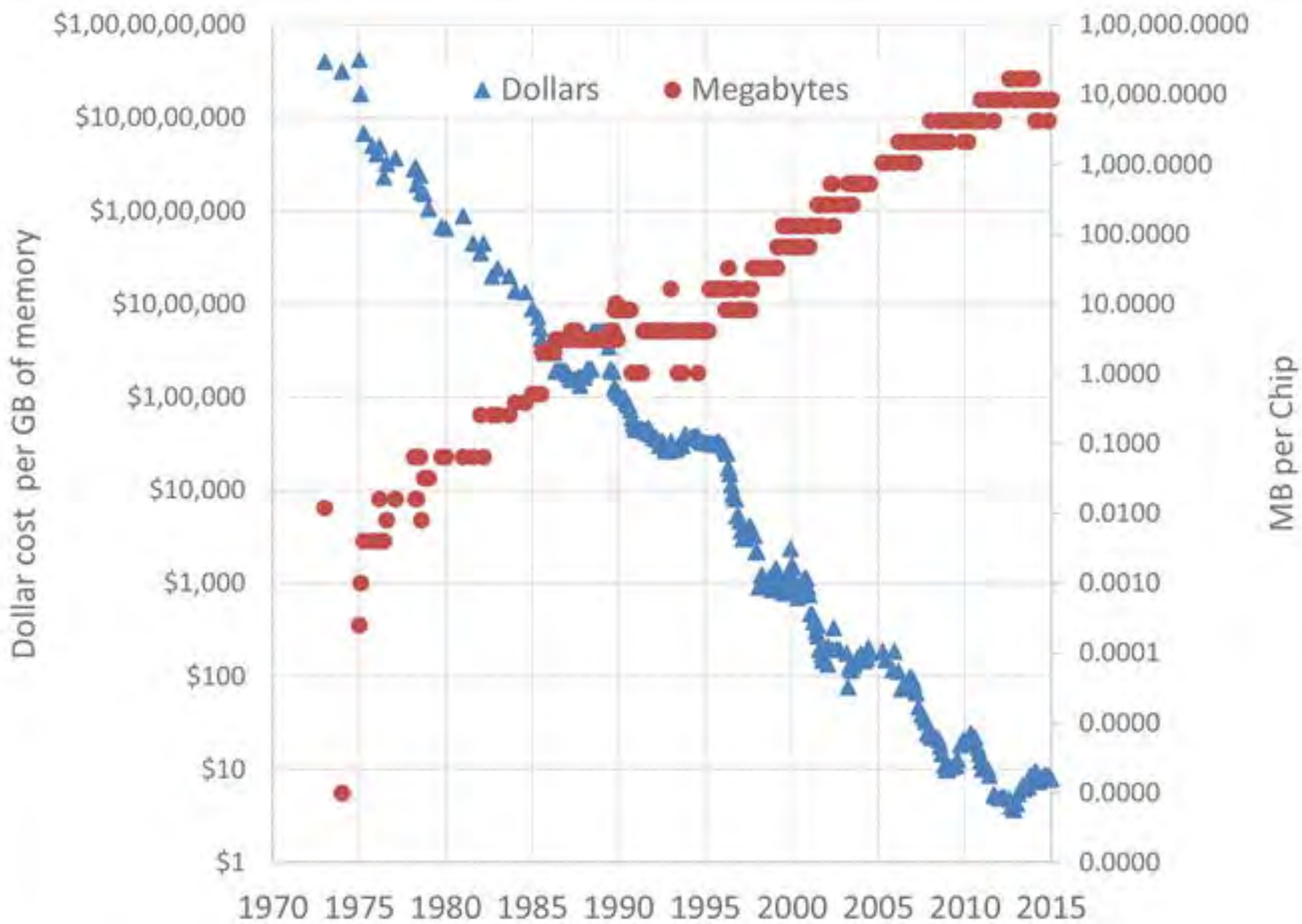
- Patches and upgrades are applied at multi tenant container level
- All PDBs in that containers are automatically upgraded
- Easy pathway to fallback as well

In-Memory Databases

Traditional Databases

- Traditional Database us memory to cache data stored on disk
    - Generally show significant performance improvements as the amount of memory increases
    - Some database operations must write to a persistent media
    - COMMIT requires a write to a transaction log
        - Periodically database writes checkpoint blocks from memory to disk
    - Taking advantage of large memory requires an architecture that is aware of memory residence of data

In-Memory Databases



Source: http://www.jcmit.com/memoryprice.htm

Sohail

Changes to Traditional Databases

- Cache-less architecture
  - There is no point caching in memory what is already stored in memory
- Alternative persistence model
  - Data in memory disappears when the power is turned off
  - Database must apply alternative mechanism for ensuring that no data loss occurs.

In-Memory Databases

- TimesTen
- SAP HANA
- Redis
- RDBMS with In-Memory options

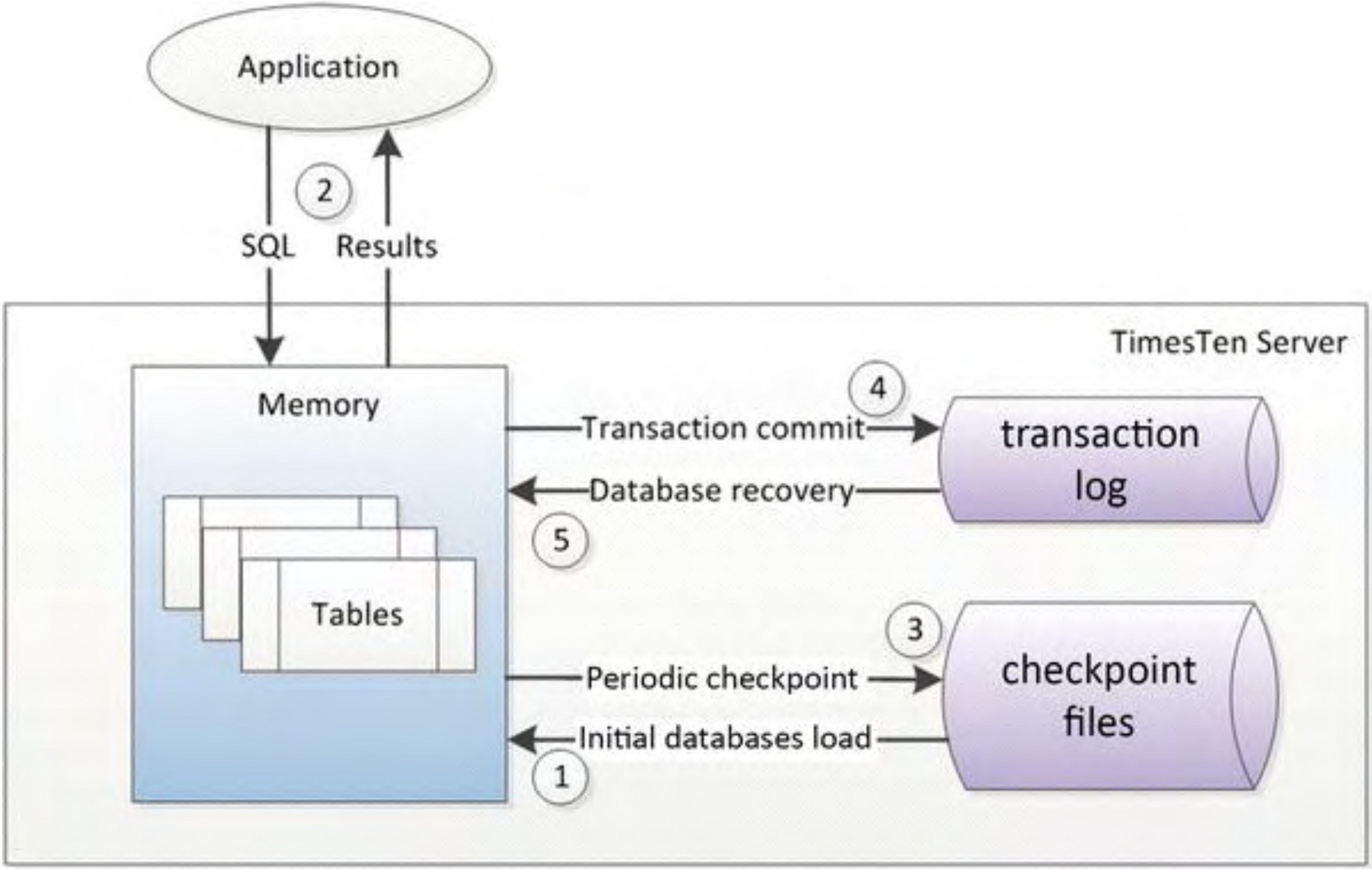
TimesTen

- One of the earliest implementation of in-memory database system
  - RDBMS compatible
- All data is memory resident
  - Persistence is achieved by periodic snapshots of memory to disks and
  - Writing to disk-based transaction log following a transaction commit.

TimesTen Architecture

- In default configuration
  - Disk writes are asynchronous
  - If the power fails between transaction commit & transaction log
    - Data could be lost — Transaction durability is not guaranteed
  - Synchronous writes — DB operation must wait on disk

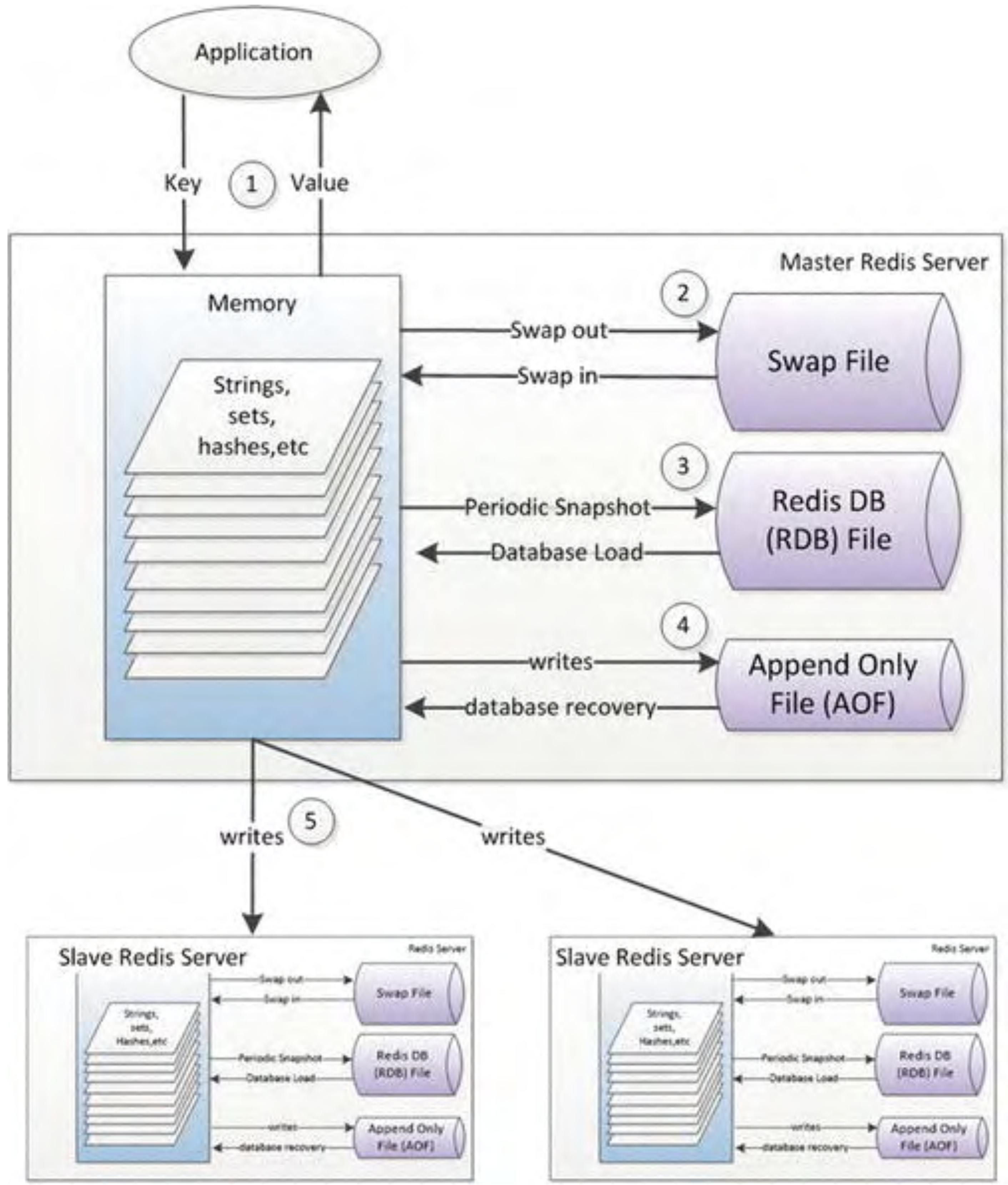
TimesTen Architecture



Redis

- In-Memory key-value pair store
- Possible to operate datasets larger than available memory
    - Virtual memory feature — Redis swap out older key value to a disk file
- Redis uses disk file for persistence:
    - Snapshot — stores copies of the entire Redisk system at a point in time
    - Append Only File (AOF) — Keeps journal of changes that can be applied in case of a failure

Redis Architecutre



Sohail Rafiqi

SAP HANA

- In-Memory designed for Business Intelligence
- Tables can be configured row-oriented or column-oriented
  - Data suited for BI —configure column-oriented
  - OLTP — Row oriented.
- All row store is guaranteed to be in memory
- Column store is by default loaded on demand
  - Can be configured for immediate loading on startup
- Persistence architecture of HANA uses snapshots and journal file patterns similar to TimeTen and Redis
- ACID is enabled by redo log
  - Redo log is written upon transaction commit
  - To speed up transaction — redo logs are placed on solid state disk

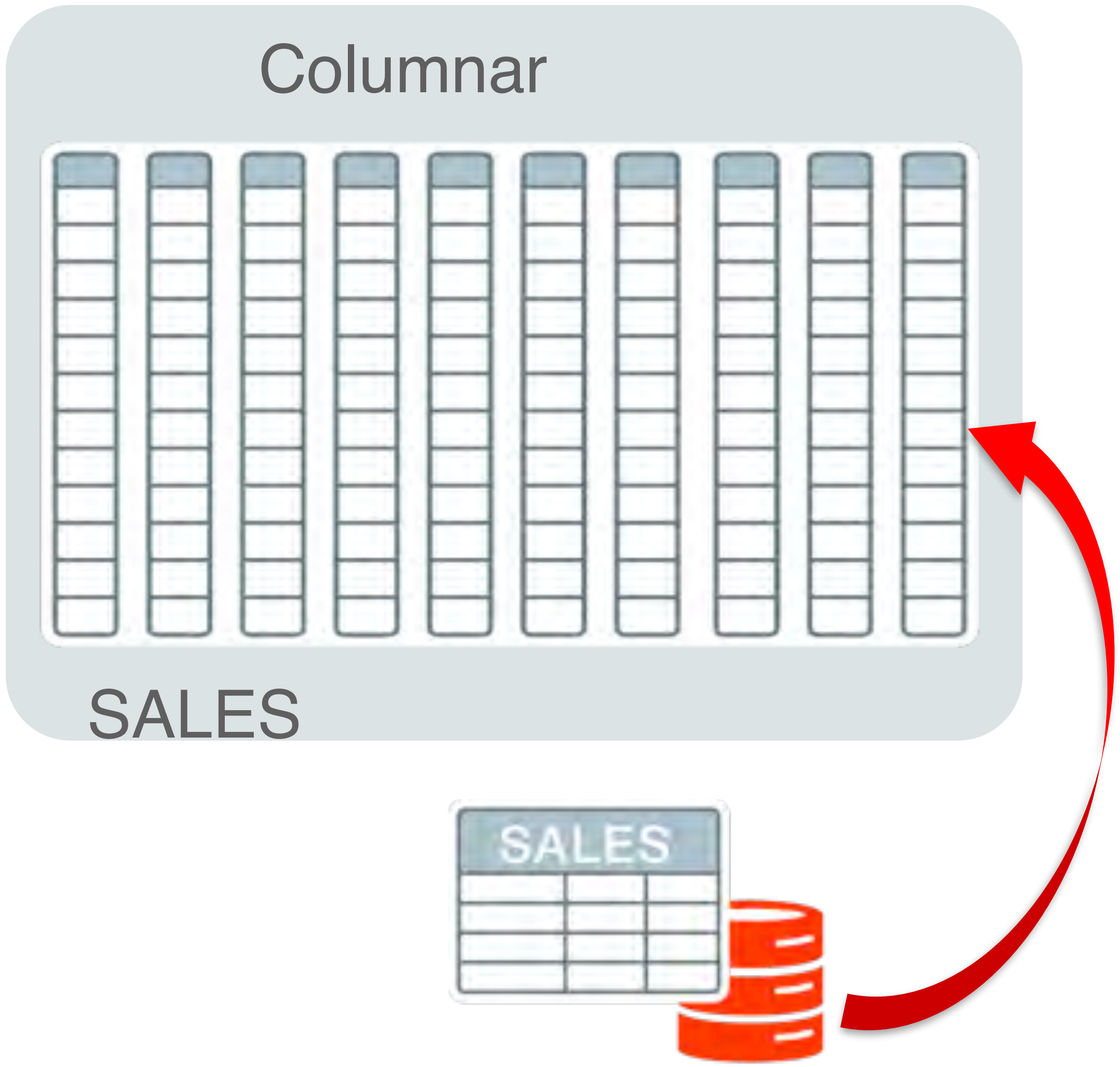
VoltDB

- Designed not to wait for disk IO
- Designed with explicit intention to not requiring disk IO
- Support aCID transactional mode through replication across multiple machines
    - Transaction commit only complete once data is successfully written to more than one physical machine

Column & Row-store

- OLTP
  - Row format —
- Analytic queries
  - Columnar format

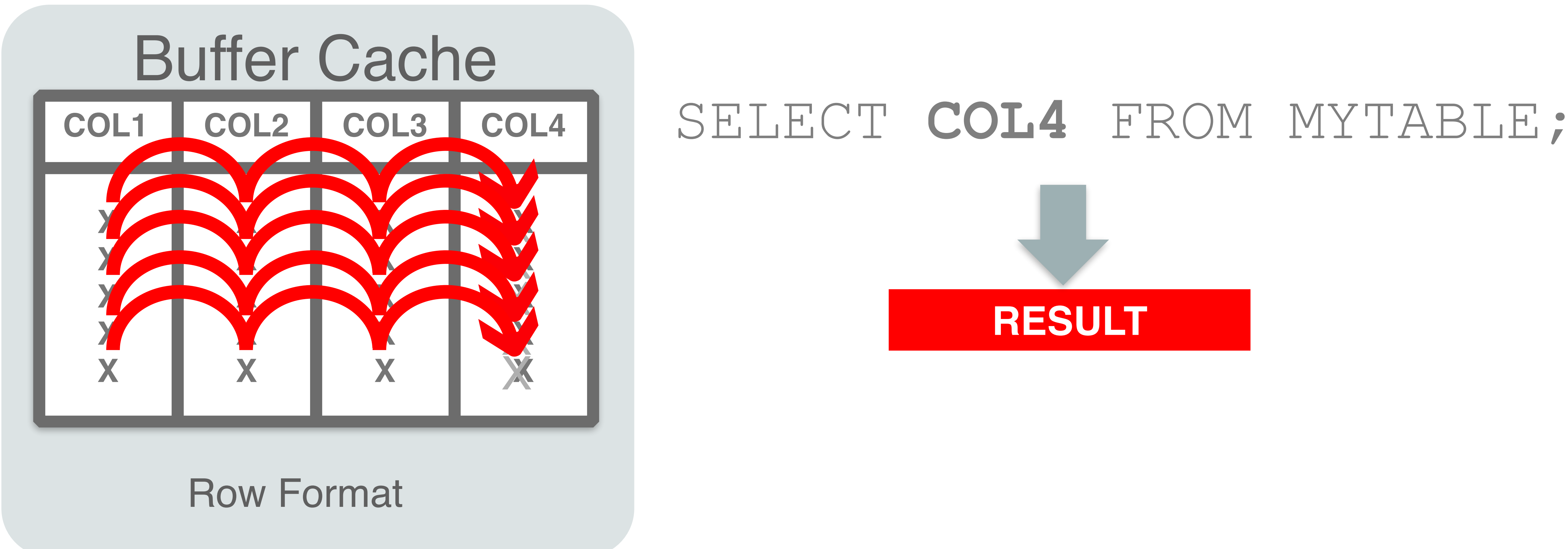
In-Memory Columnar Option



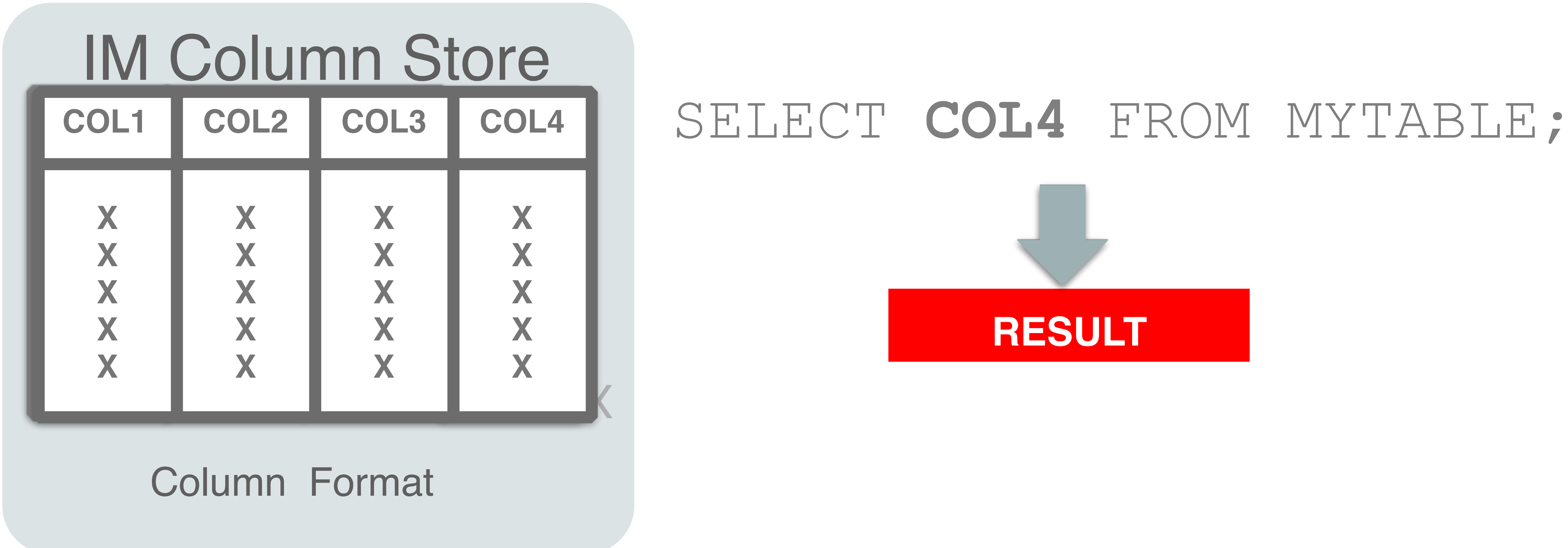
ALTER TABLE sales INMEMORY;

ALTER TABLE sales NO INMEMORY;

Why is In-Memory faster than buffer cache?

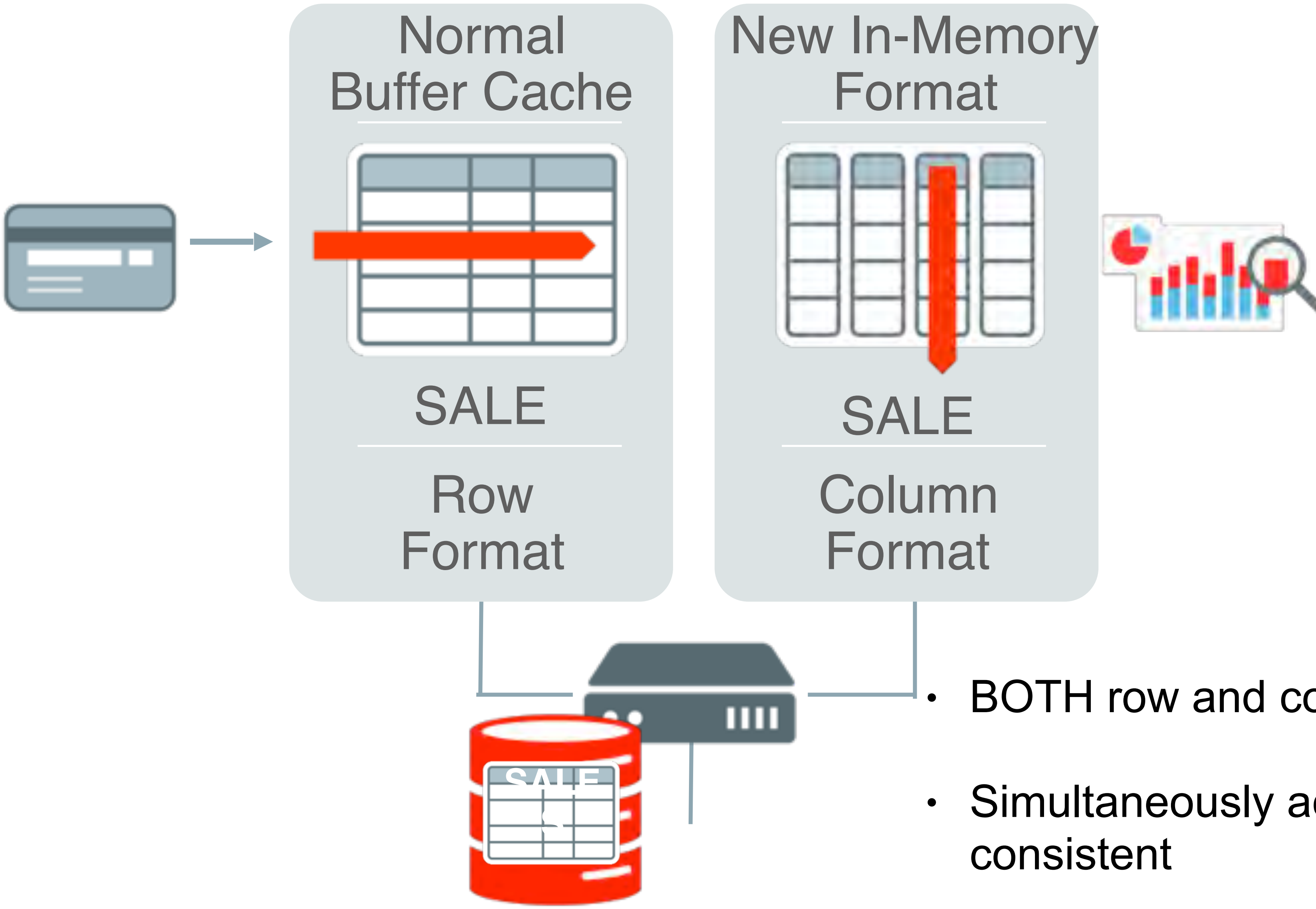


Why is In-Memory faster than buffer cache?



- In the column store the query behaves very differently
- Data is store is separate physical column structures
- Just go directly to the col4 structure and scan all the entries, one after the other.

Dual Format



- 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