
ORACLE®

Using Oracle Columnar Technologies

Across the Information Lifecycle

Roger MacNicol
Software Architect
Data Storage Technology

ORACLE

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, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Is the Terracotta Army in Rows or Columns?

Two Ways to Store Data

Row-Major: the Writer is King	Column-Major: the Reader is King
Small pages	Large pages
Data in rows	Data in columns
Many indexes	Few Indexes
Short transactions	Long transactions
High concurrency of writers	Low concurrency of writers
PK/Index based reads	Foundset based reads
Retrieve few rows, many columns	Retrieve many rows, few columns
Low compression heterogenous data	High compression homogenous data
Needs workarounds for DSS	Needs workarounds for row inserts

Oracle Columnar Technology

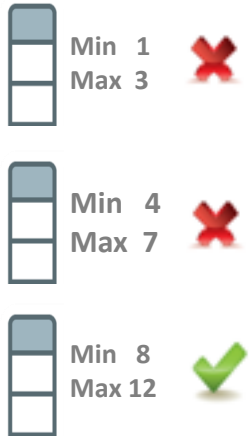
Columnar Format

Access only the columns you need

Compression

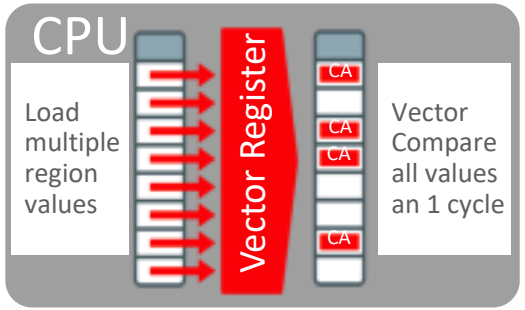
Scan & filter data in compressed format

Storage Indexes

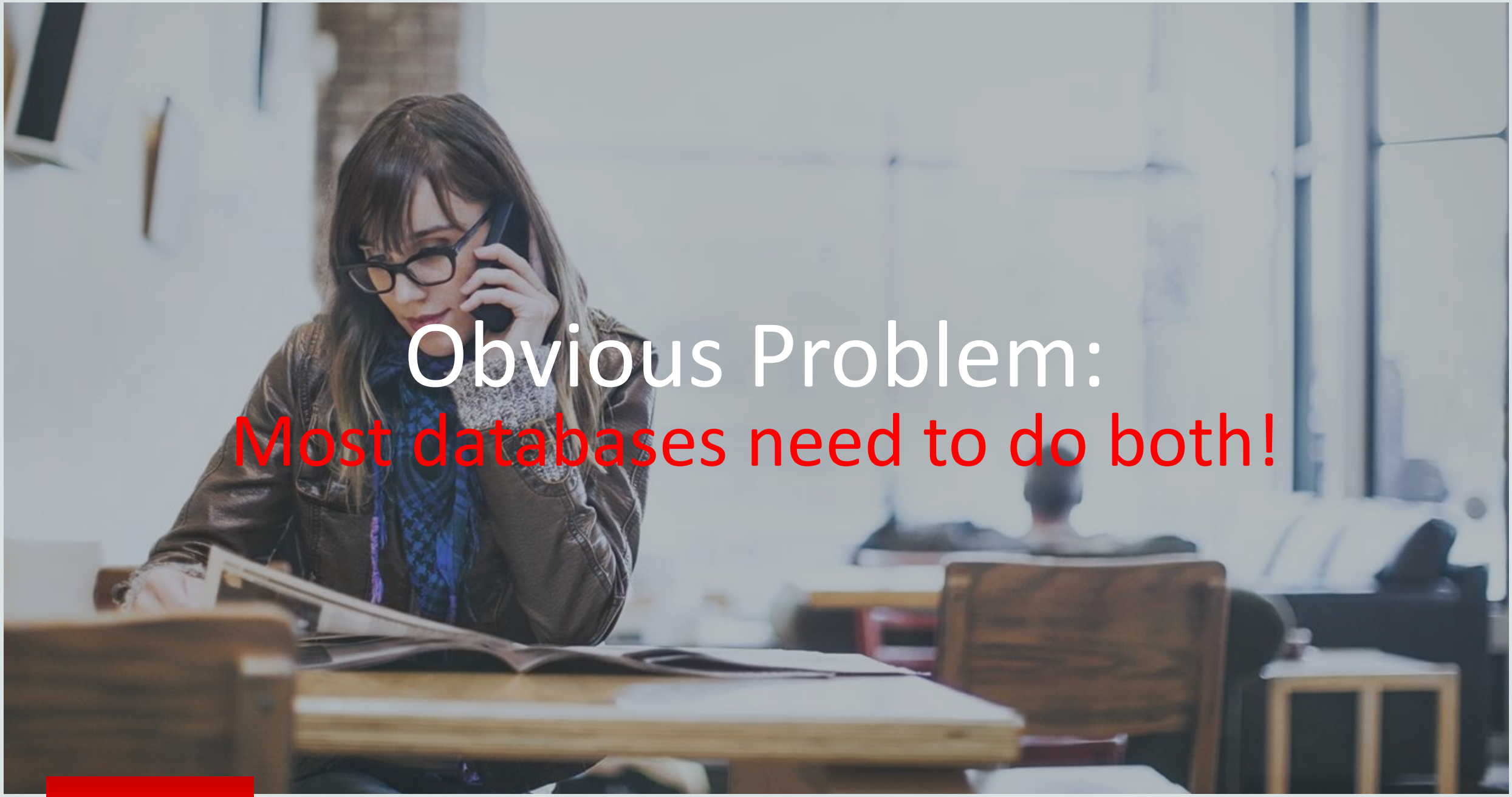


Prune out any unnecessary data from the column

SIMD Vector Processing

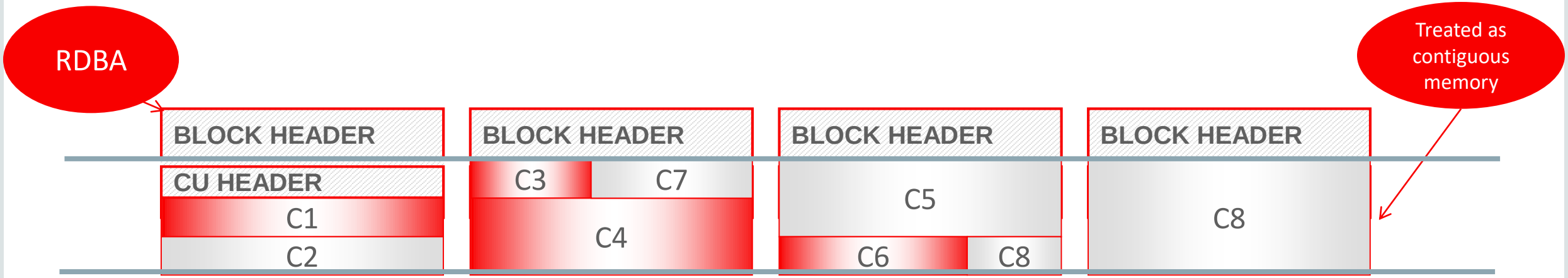


Process multiple column values in a single CPU instruction

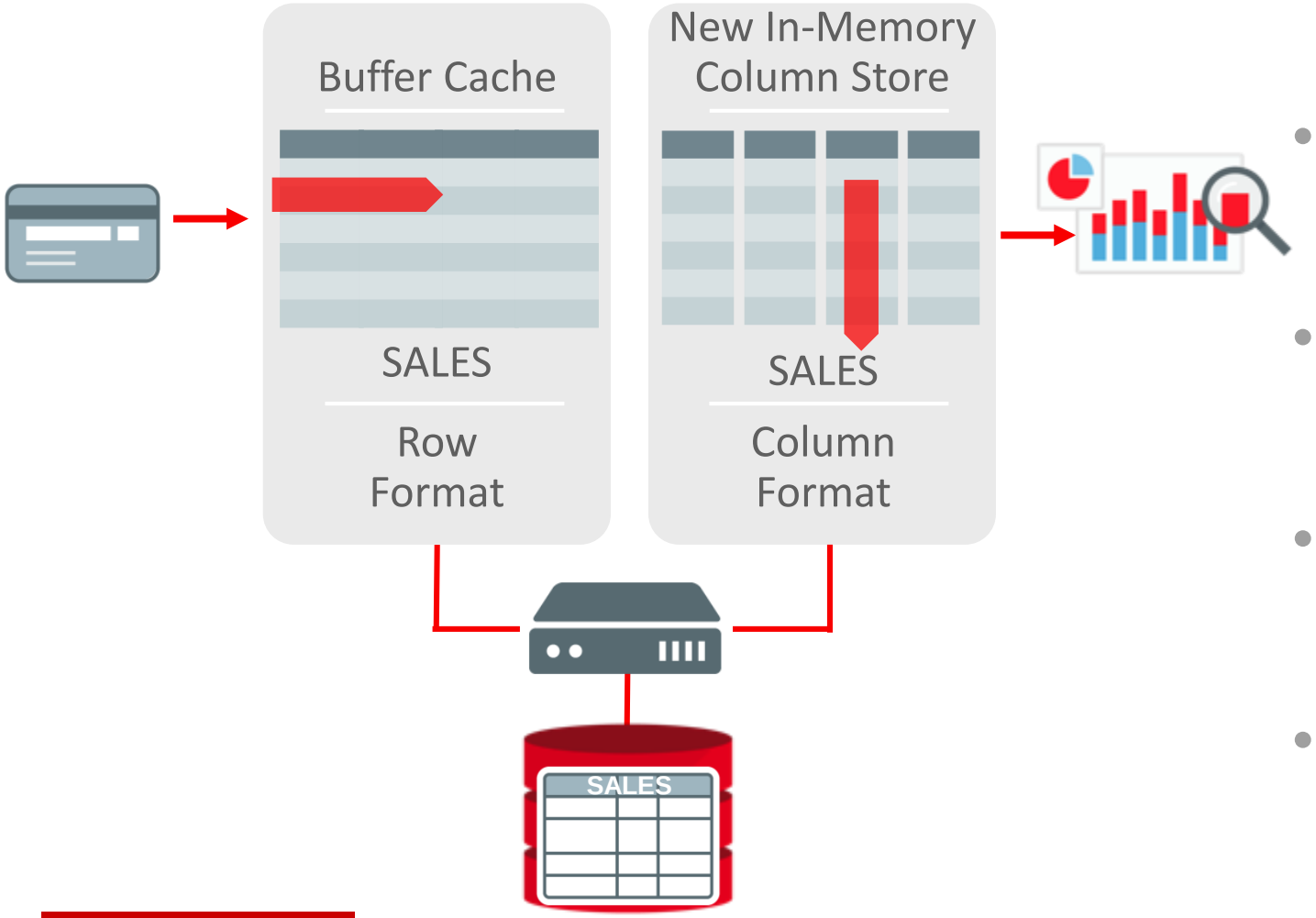


First Breakthrough: Dual Format Superblocks

- Pivoting data from several contiguous blocks to create a hybrid layout
  - Logical structure spanning multiple database blocks
  - Data organized by column and each column compressed separately
  - Only one CU header per RDBA (rowid)
  - Ranging from 64 to 256 KB
  - Oracle design philosophy: every block must be self-describing



Second Breakthrough: Dual Format Database



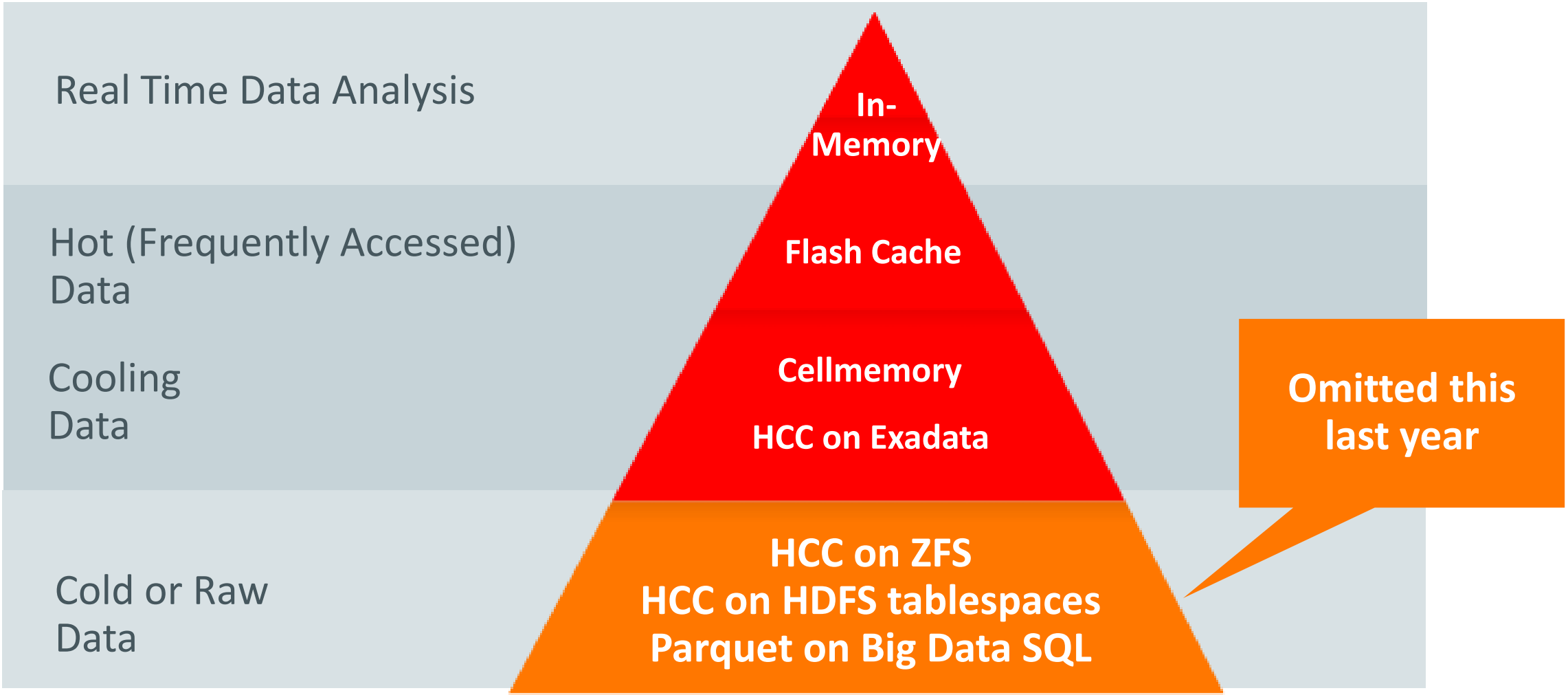
- **BOTH** row and column formats for same table
- Simultaneously active and transactionally consistent
- Analytics & reporting use new in-memory Column format
- OLTP uses proven row format

Only **analytic queries** benefit from accessing ~~the IM column store~~ **columnar data**

What is an analytic query?

- Our definition: Using aggregation to find patterns and trends in the data

Columnar Tiering: the complete columnar story



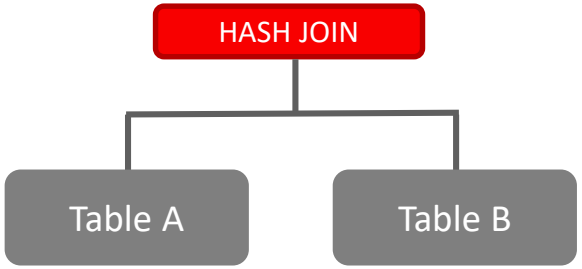
Database In-Memory: Real-time Analytics

Improves key aspects of analytic queries

Scans

- Speed of memory
- Scan and Filter only the needed Columns
- Vector Instructions

Semi-Joins

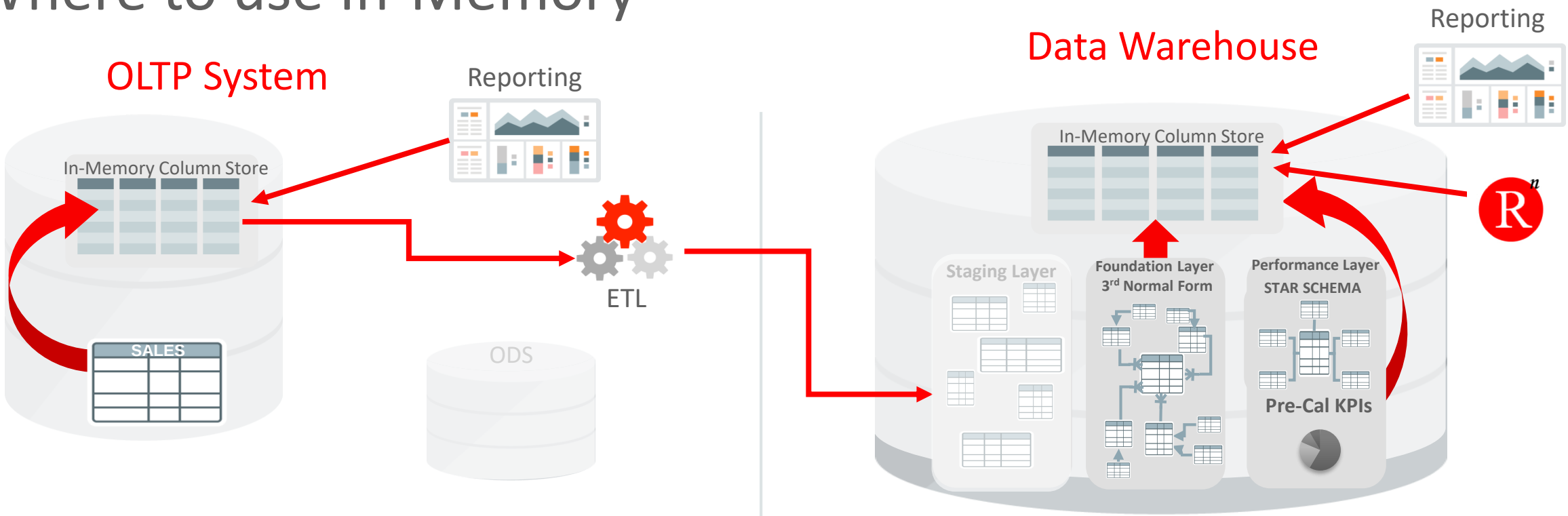


- Convert Star Joins into 10X Faster Column Scans
- Search large table for values that match small table

Scan Based Aggregation

- Pre-join the dimension tables and turn aggregation into a fast table scan

Where to use In-Memory



- Enables real-time reporting directly on OLTP data
- Speeds data extraction part of ETL process
- Removes need for separate Operational Data Stores
- Speeds up mixed workload

- Star-schema and pre-calculated Key Performance Indicators (KPIs)
  - Improves performance of dash-boards
- Note: Staging/ETL/Temp not good candidates
  - Why? Write once, read once

Get The Most From ~~In-Memory~~ Columnar Data

- Understand where it helps and where it doesn't
- Columnar processing speeds up analytic data access, not:
  - Network round trips, logon/logoff
  - Parsing, PL/SQL, complex functions
  - Data processing (as opposed to access)
    - Inserts, updates or deletes (but can help the searched part of DMLs)
    - Complex joins or aggregations where not much data is filtered before processing
  - Load and select once – Staging tables, ETL, temp tables

Know your bottleneck!

Managing the In-Memory Area
or what to do when my data doesn't fit

NEW IN 12.2

What to do when
Data Doesn't Fit In-Memory

- Use Dynamic Resizing of In-Memory Area
- Use Automatic Data Optimization to manage what's In-Memory
- Use Partial Indexing and table expansion
- Use In-Memory on Active Data Guard
- Use In-Memory Formats on Storage Flash

ORACLE

What happens to partially loaded segments?

- No automatic eviction (LRU) to make space
  - Only a refresh being able to get space
- Optimizer makes Cost-Based decision
  - Use DBIM and get the rest from Buffer Cache
  - Use DBIM and get the rest from Direct Read
    - Underscore param: IMCU percentage threshold to switch from DR to BC
    - Default 80%
  - Use Hybrid / Index-Access plan (Table Expansion)
- Out-standing enhancement request
  - Use DBIM and remaining from Smart Scan

Dynamic In-Memory Column Store

```
ALTER SYSTEM SET
  inmemory_size = 300m scope=both;
```

```
SQL> SELECT *
  2  FROM   v$inmemory_area;

POOL            ALLOC_BYTES USED_BYTES POPULATE_STATUS      CON_ID
--------------- ----------- ---------- ---------------- ----------
1MB POOL          124780544  102760448 DONE                     0
64KB POOL          16777216     851968 DONE                     0

SQL>
SQL> ALTER SYSTEM SET inmemory_size = 300M;

System altered.

SQL>
SQL> SELECT *
  2  FROM   v$inmemory_area;

POOL            ALLOC_BYTES USED_BYTES POPULATE_STATUS      CON_ID
--------------- ----------- ---------- ---------------- ----------
1MB POOL          216006656  102760448 DONE                     0
64KB POOL          32374784     851968 DONE                     0
```

- IM column store is now dynamic
  - Possible to increase the size of IM column store without database restart
  - The IM column store can not shrink
  - Only possible if free memory available within the SGA
  - Only possible if new size is 128MB greater than current INMEMORY_SIZE
  - Change reflected immediately in V$INMEMORY_AREA

Partial Indexing and Hybrid Plans with DBIM (12c)

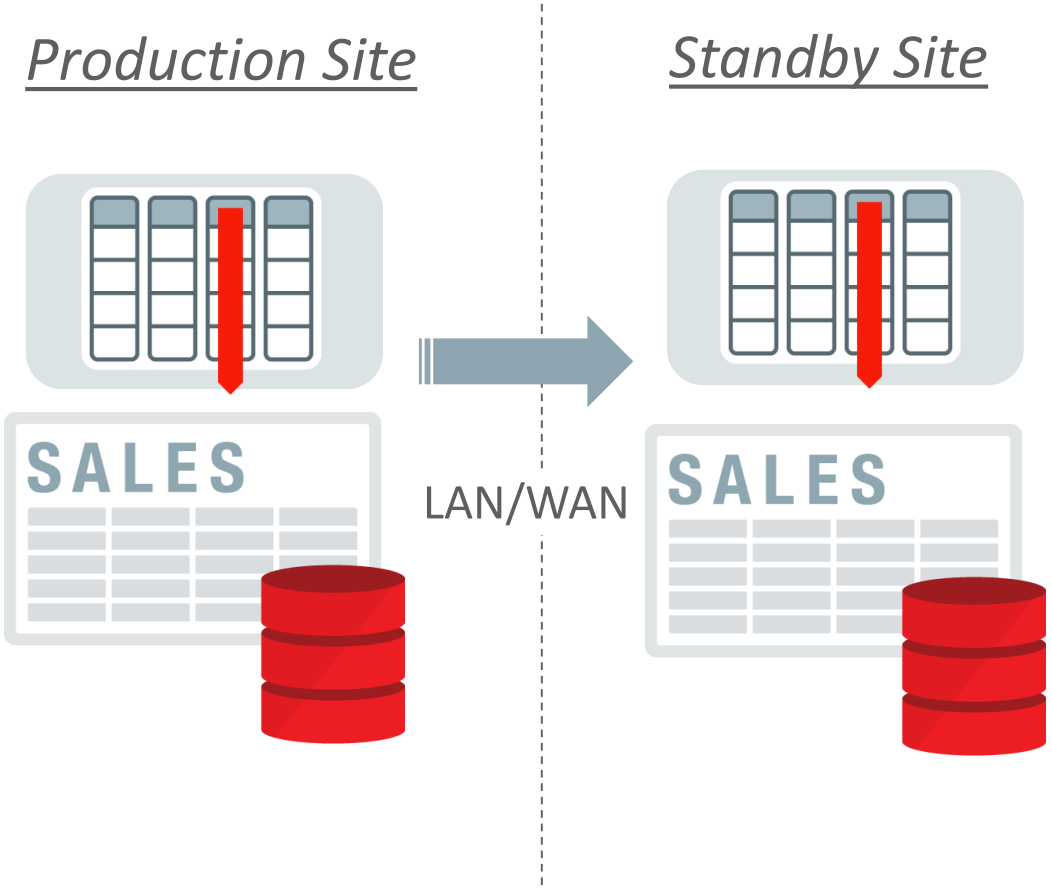
- In-Memory scans can find one or more values in a column fast
  - IMCU pruning
  - Dictionary Encoding
- When partitions age out of memory, we don't want a sudden drop in query performance
- One answer: use partial indexing:
  - Alter Table Modify Partition no inmemory;
  - Alter Table Modify Partition indexing on;
    - Not possible as an ADO policy

Partial Indexing and Hybrid Plans (12c)

- The INDEXING [FULL | PARTIAL] clause means only those table partitions with indexing on are to be included within the index.
- create index <index-name> on <table-name>(<column>) indexing partial;
- Partition by range or list to index only those values used in narrow range scans
  - subpartition template
    (subpartition statecode values ('MA', 'RI', 'CT') indexing off,
    subpartition statecode values ('VT', 'NH', 'ME') indexing on)
- alter table <table-name> modify partition <partition-name> indexing [on | off];
- Get a table expoansion plan
  - UNION-ALL
    - INDEX RANGE SCAN
    - TABLE ACCESS FULL

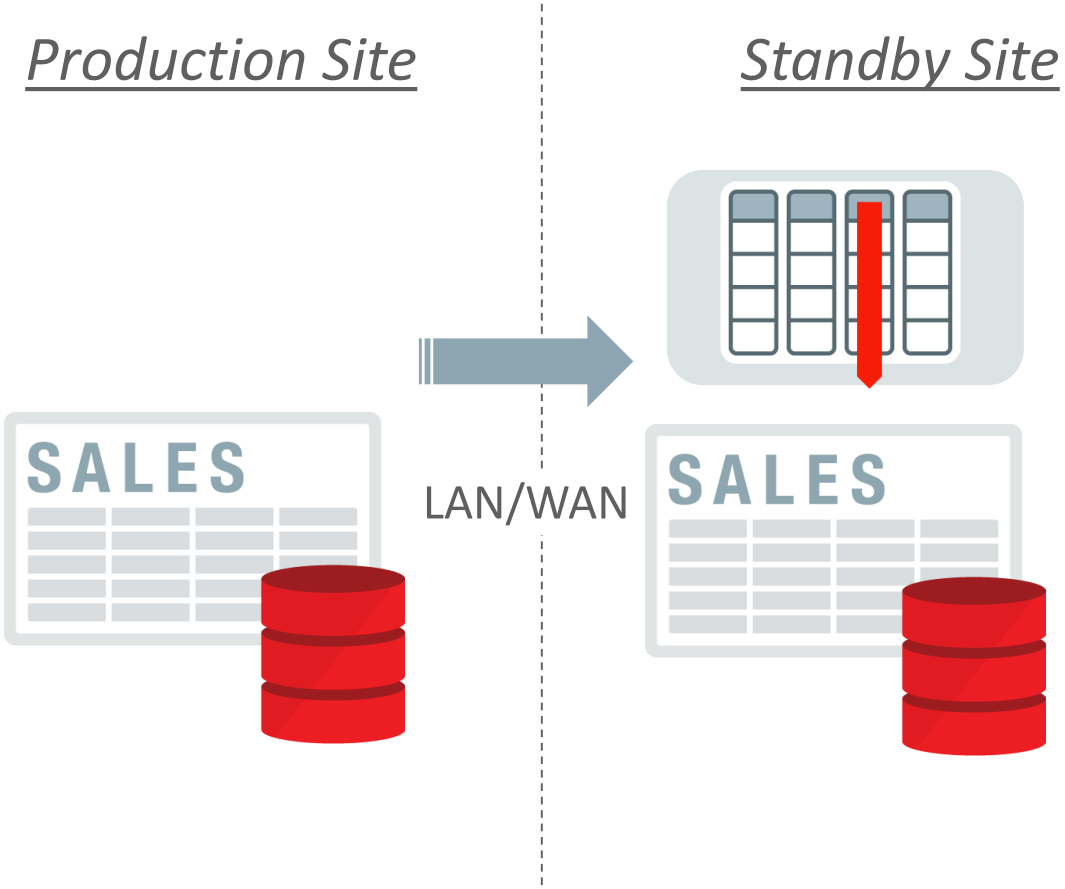
In-Memory on Active Data Guard

In-Memory on Active Data Guard: Identical Configurations



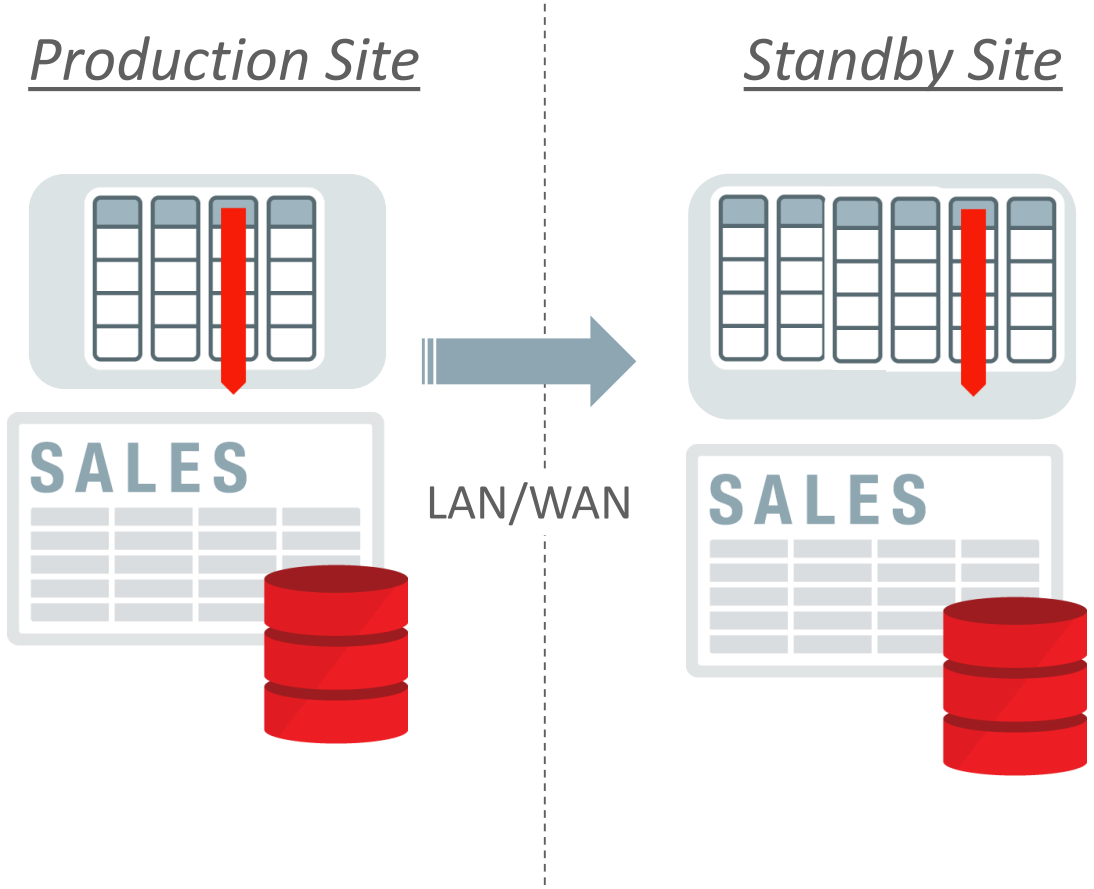
- INMEMORY_SIZE parameter
  - set on both Primary & Standby sites
- All objects marked INMEMORY
- Analytic queries
  - Connect to either system & access data In-Memory

In-Memory on Active Data Guard: In-Memory On Standby Only



- INMEMORY_SIZE parameter
  - set on Standby site only
  - Free up memory on Primary for applications to use
- All objects marked INMEMORY
- Analytic queries
  - Connect to secondary site & access data In-Memory

IM on Active Data Guard: Different Data In-Memory on Each Site

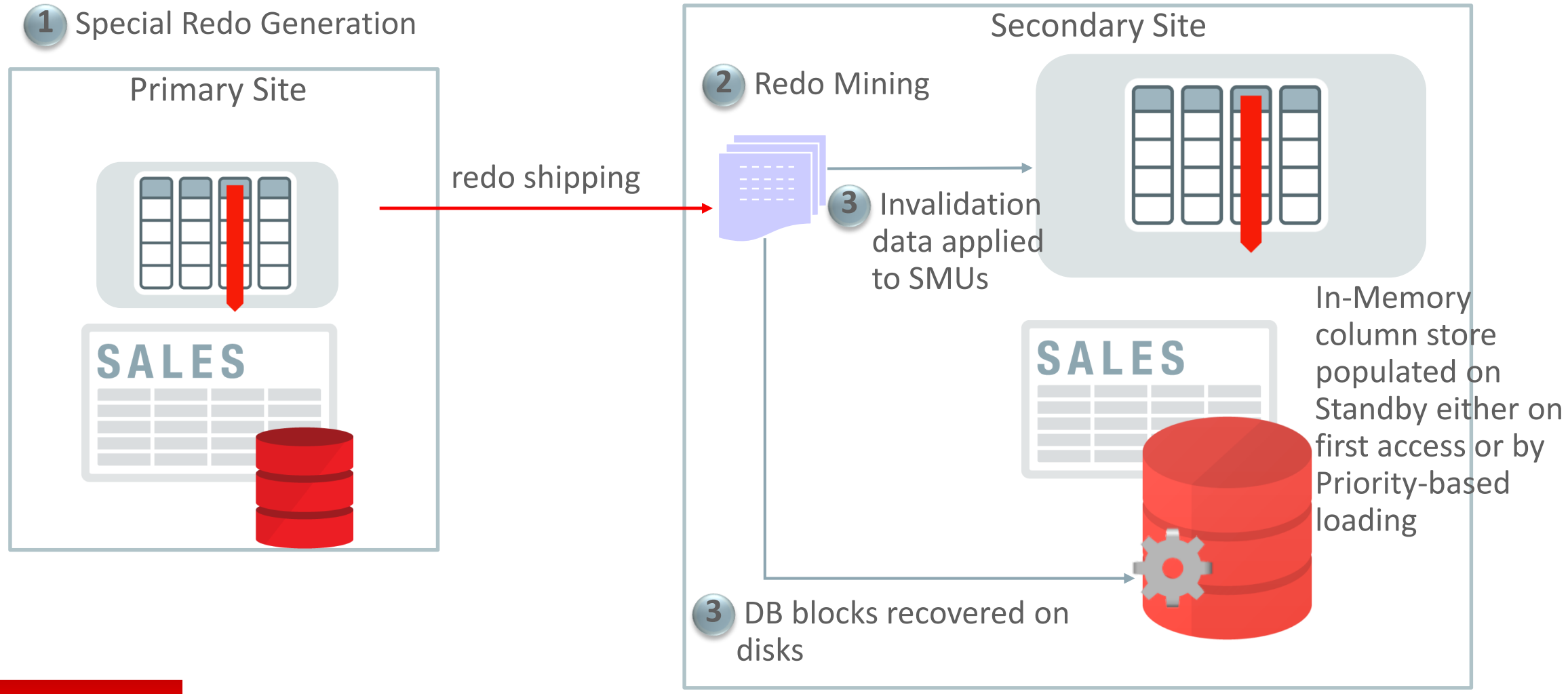


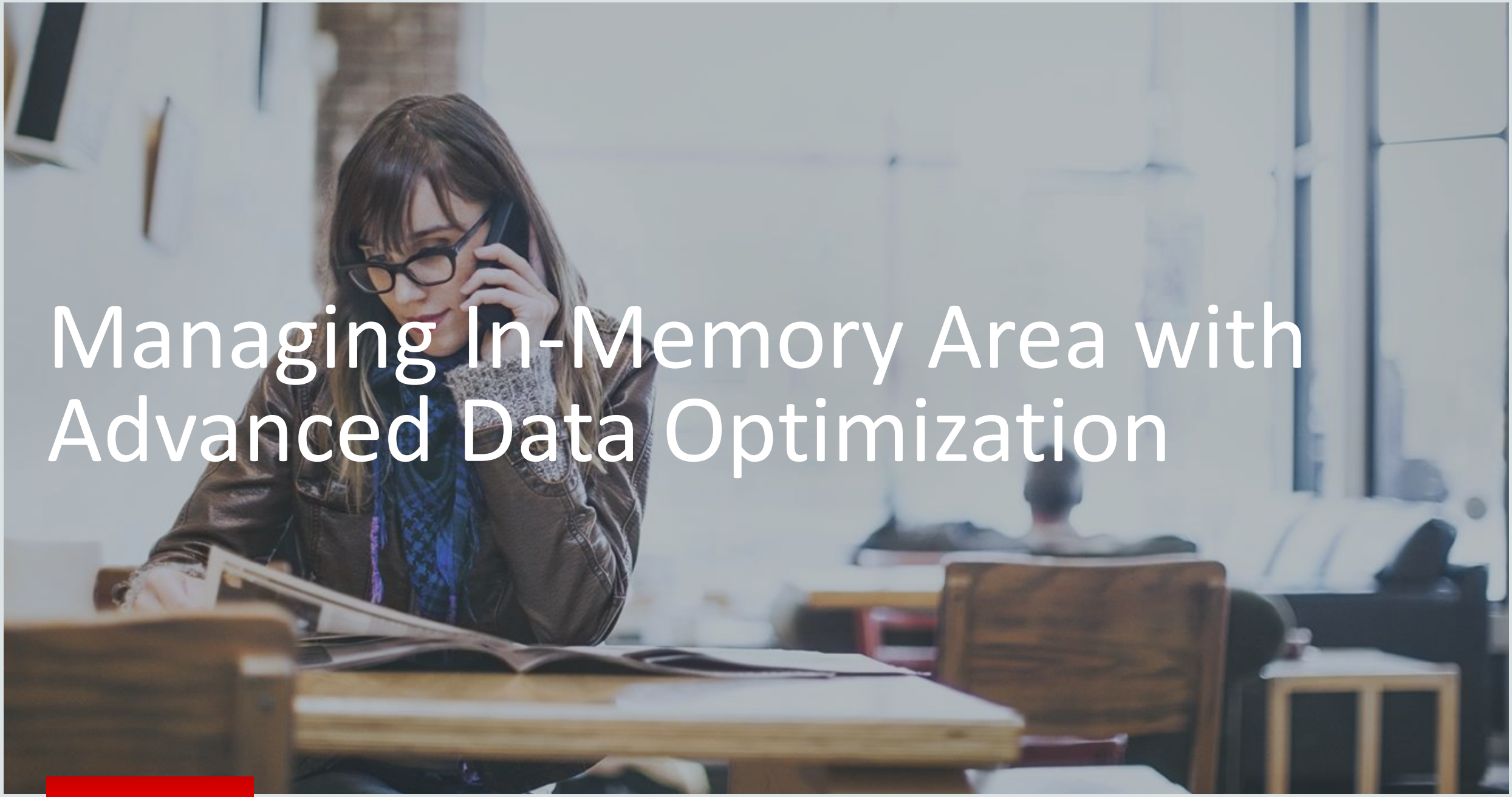
- `INMEMORY_SIZE` parameter
  - set on both Primary & Standby sites
- All objects marked `INMEMORY DISTRIBUTE FOR SERVICE`
  - Service defined for Primary
  - Service defined for Standby
  - Service defined for both
- Analytic queries
  - Connect to appropriate service

Redo Generation Modified

- Existing redo generated by DMLs extended to maintain information on whether change is to an in-memory object
- Existing Begin/End transaction redo has information indicating whether the transaction could have touched any IMC objects
- Special redo marker for certain DDLs on an in-memory object, which indicates invalidation of the whole object

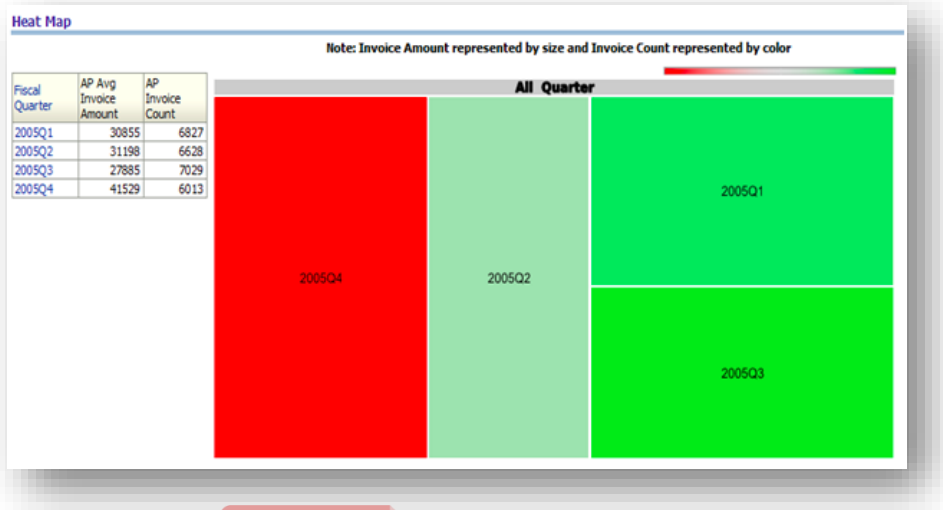
IMC-ADG High-Level Architecture





Managing In-Memory Area with Advanced Data Optimization

Automatic Data Optimization



- An in-memory heat map tracks disk based block and segment access
  - Heat map is periodically written to storage
  - Data is accessible by views or stored procedures
- Users can attach policies to tables to compress or tier data based on access
  - Tables, Partitions or Sub-partitions can be moved between storage tiers and compression levels
  - Online, no impact to data availability
  - Policies define automatic data tiering
- Part of the Advanced Compression Option

Automatic Data Optimization with Database In-Memory

NEW IN 12.2

ADO IM Policies

- Policy criteria
  - after <time spec> of no access
  - after <time spec> of creation
  - on <function_name>
- Successful policy completion results in policy being disabled
- Policies only run in the maintenance window
- Outside maintenance window we can run policies manually: dbms_ilm.execute_ilm procedure

Automatic Data Optimization with Database In-Memory

NEW IN 12.2

ADO IM Policy Examples

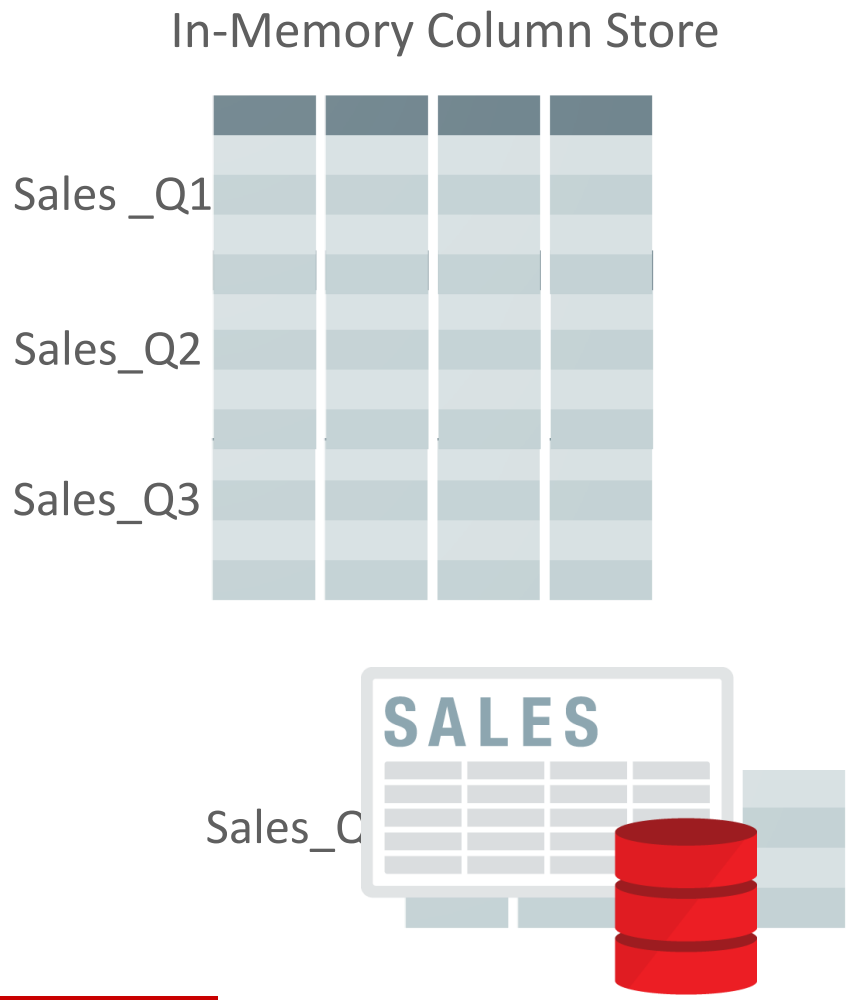
- Examples
  - alter table sales ilm add policy no inmemory after 10 days of no access;
  - alter table sales ilm add policy no inmemory after 45 days of creation;
  - alter table sales ilm add policy no inmemory after 3 days of no modification;
  - alter table sales ilm add policy no inmemory on sales_chk_im;
    - Where sales_chk_im is a Pl/SQL function that returns a boolean

ORACLE

Automatic Data Optimization with Database In-Memory

NEW IN 12.2

Policy Mode Example – No Access




```
ALTER TABLE sales ILM ADD POLICY
NO INMEMORY AFTER 45 days OF NO ACCESS;
```

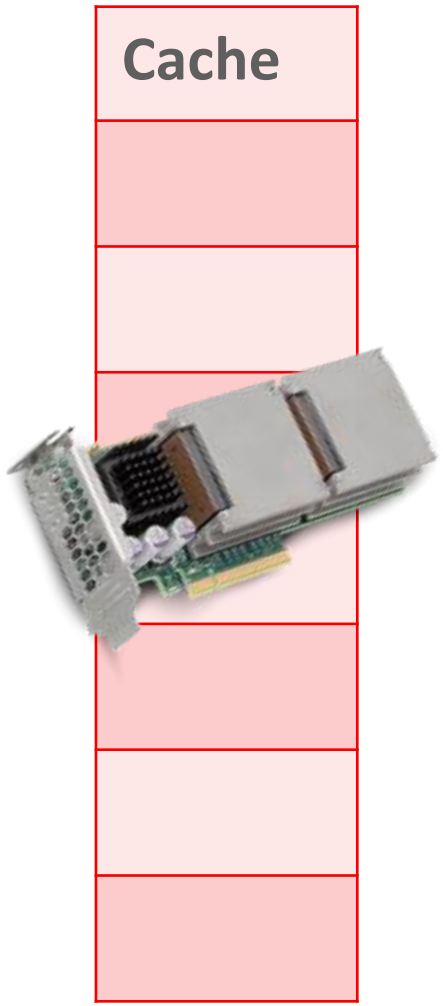
- "Cold" partitions are evicted based on no access
- Frees room in the IM column store
- New populations occur as they do today

ORACLE

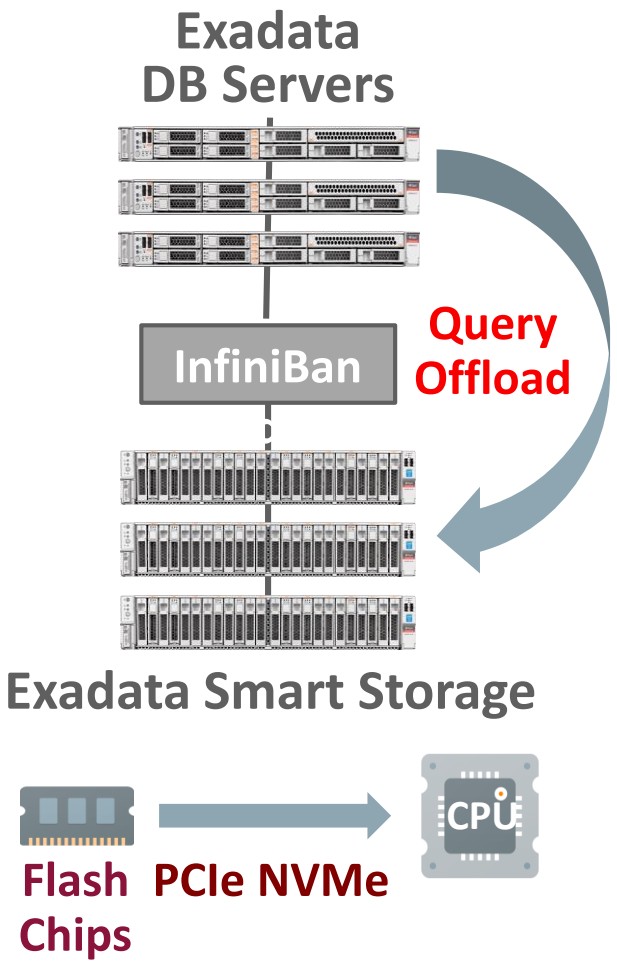
Next Generation Columnar Cache

Review: Exadata Smart Flash Cache Scan Awareness

- Exadata Smart Flash Cache is scan resistant
  - Ability to bring subset of the data into cache and not churn
  - OLTP and DW scan blocks can co-exist
  - Separate KEEP and AUTOKEEP areas
  - OLTP has absolute priority over 50%
  - Reads > 128 KB are eligible for AUTOKEEP
  - Tracks touch count for AUTOKEEP
  - Needs to see table scanned > once an hour
- Nested scans bring in repeated accesses
  - Repeat, For each item in large table, scan small table
- Happens automatically, no tuning or configuration needed
- Storage(cell_flash_cache keep) – not best practice



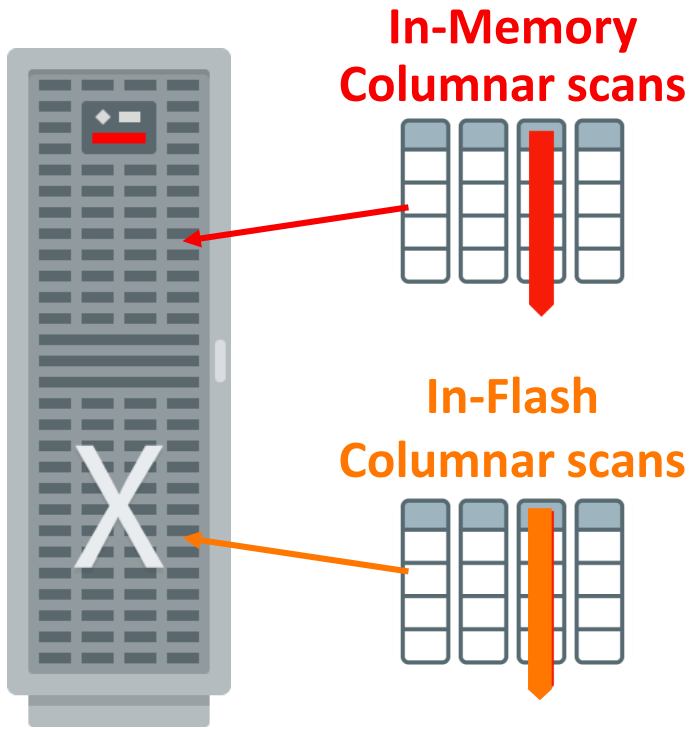
Exadata Achieves Memory Performance with Shared Flash



- **Exadata X6 delivers 300GB/sec flash bandwidth to any server**
  - Approaches 800GB/sec aggregate DRAM bandwidth of DB servers
- **Must move compute to data to achieve full flash potential**
  - Requires owning full stack, can't be solved in storage alone
- **Fundamentally, storage arrays can share flash capacity but not flash performance**
  - Even with next gen scale-out, PCIe networks, or NVMe over fabric
  - e.g. new EMC DSSD has **3-6 times lower** throughput than Exadata X6
- **Shared storage with memory-level bandwidth is a paradigm change in the industry**
  - Get near DRAM throughput, with the capacity of shared flash

Preview: Redesigning Scan Offload for Memory Throughput

- **With Exadata Flash throughput approaching memory throughput, SQL bottleneck moves from I/O to CPU**
- **Exadata will automatically transform table data into In-Memory DB columnar formats in Exadata flash cache**
  - Dual format architecture extended from DRAM to flash
- **Enables fast vector processing for storage server queries**
  - Smart Scan results sent to DB using In-Memory Columnar format to reduce DB CPU usage
- **Uniquely optimizes next generation flash as memory**



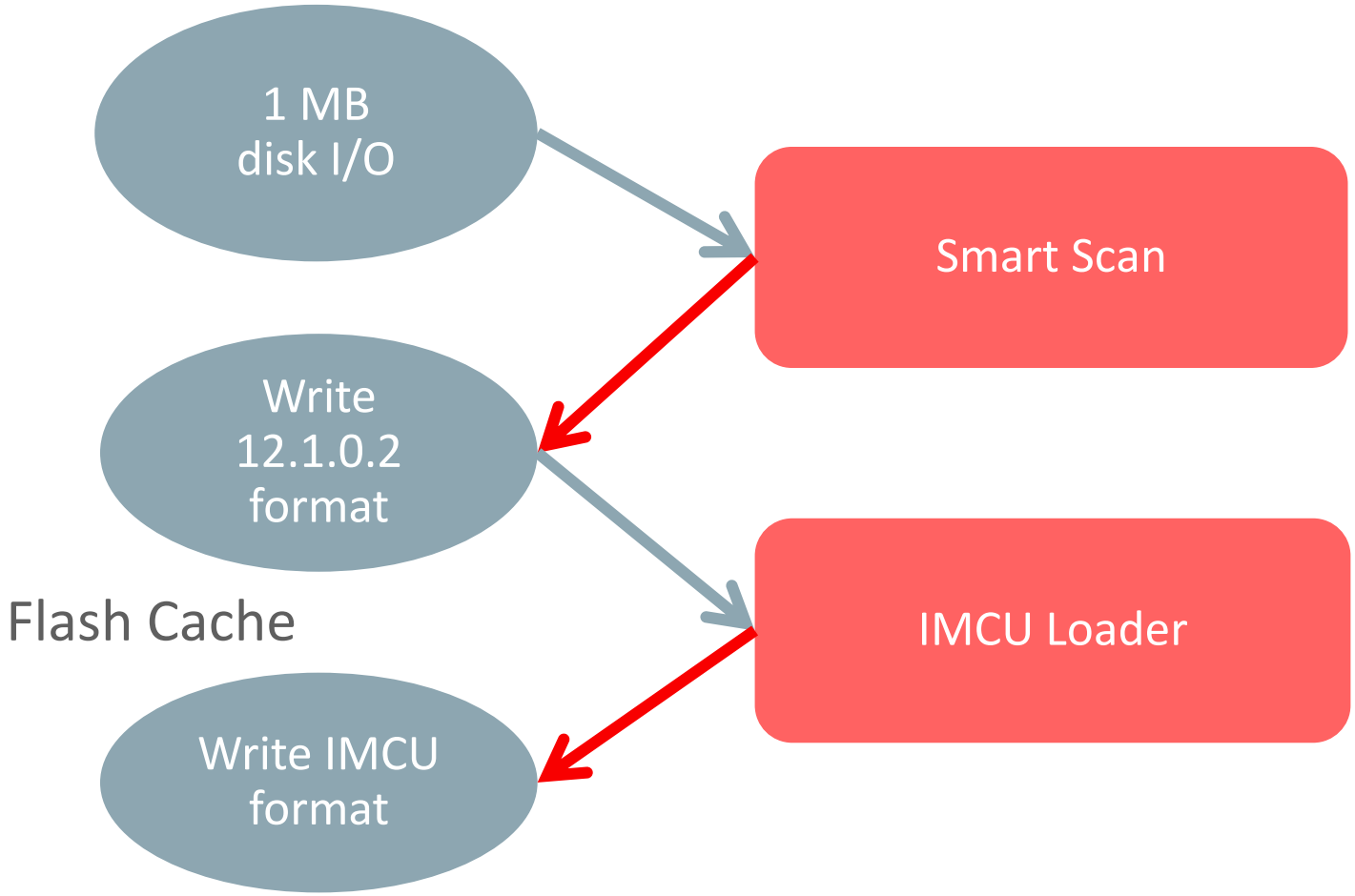
Production in 2016

Adds Explicit DDL to overrule the default behaviour

- Matches the INMEMORY [MEMCOMPRESS] clause
  - Requires INMEMORY_SIZE to be set
- Alter Table <T> No CELLMEMORY
- Alter Table <T> CELLMEMORY
  - Uses system default settings for columnar cache
- Alter Table <T> CELLMEMORY MEMCOMPRESS For Query
  - Equivalent to INMEMORY Query High
- Alter Table <T> CELLMEMORY MEMCOMPRESS For Capacity - **DEFAULT**
  - Equivalent to INMEMORY Capacity Low, default typically between 1X and 1.5X space of HCC Query High
- No DDL Specified
  - Use system default (Cellmemory Memcompress for Capacity)

Built by background threads

Same as DBIM



Smart Scan

- Checks eligible for Columnar Cache:
  - Every block HCC
  - Every block CR
- Gets one 1 MB buffer and returns rewrite in 12.1.0.2 format
- Supercluster: restore SPARC endianess
- Puts rewrite on queue

IMCU Loader

- Runs at a lower priority
- Rechecks eligibility
- Loads each column CU for a column into IMCU dictionary encoding
- Creates IMCUs and applies LZO encoding
- Requests one or more 1 MB buffers and returns rewrite in IMCU format
- Typical rewrite size with LZO: **1.2 MB**

What about row-major blocks?

- Several operations create row major blocks in HCC segments
  - DMLs
  - Inserts too small to compress
  - Blocks that pre-date making it HCC
- Currently any row-major blocks cause the 1 MB region to be ineligible for column caching
- Use Alter Table Move Compress to reload into pure HCC
- 12.2 Alter Table Space Shrink Compact reworked for offload of row-major blocks

Next Generation Column Cache

- New stats:
  - cellmemory IM scan CUs processed for capacity
  - cellmemory IM scan CUs processed for query
  - cellmemory IM scan CUs processed no memcompress
  - cellmemory IM scan CUs rejected for capacity
  - cellmemory IM scan CUs rejected for query
  - cellmemory IM scan CUs rejected no memcompress
  - cellmemory IM load CUs for capacity
  - cellmemory IM load CUs for query
  - cellmemory IM load CUs no memcompress
- New v$cell_state details

Tiering columnar cold data For Analytics and for Cost

Tiering columnar cold data
1. Using **HCC on ZFS** Tablespaces

Setting Up Storage Tiering Policy
for Automatic Data Optimization

View Combines Data Sources

Not best practice

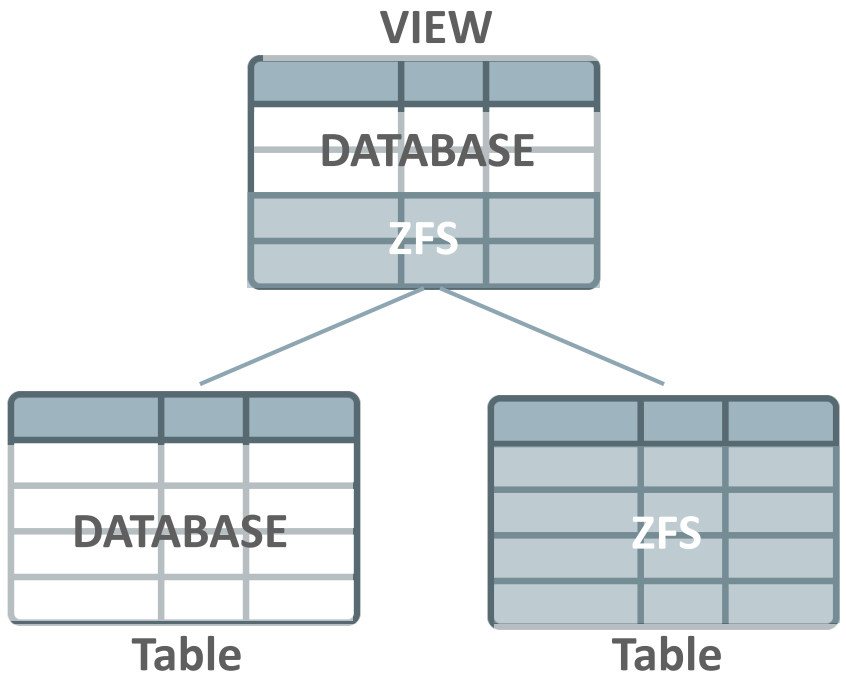
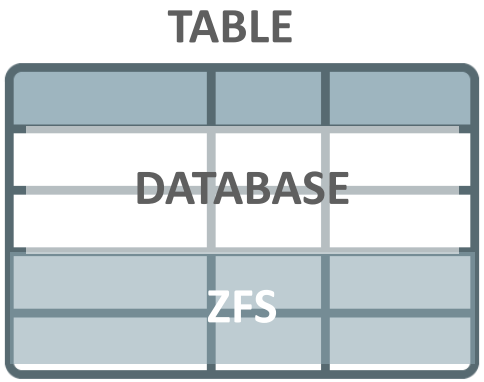


Table Storage Split Across Tier

Best practice

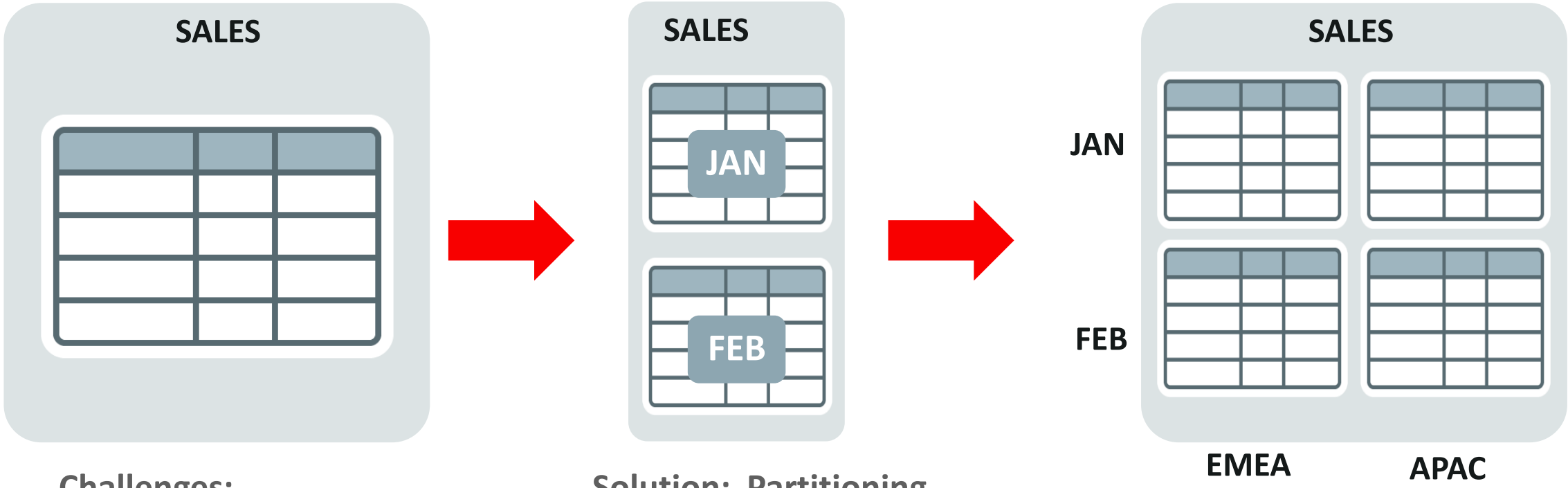


ALTER TABLE sales MODIFY PARTITION sales_2016_d100
ILM ADD POLICY
TIER TO my_zfs_tablespace;

NOTE: no time clause, triggered by TBS % free
dbms_ilm_admin.customize_ilm
(DBMS_ILM_ADMIN.TBS_PERCENT_USED, 90)
(DBMS_ILM_ADMIN.TBS_PERCENT_FREE, 75)

Oracle Partitioning

Enables large databases and indexes to be split into smaller, more manageable pieces

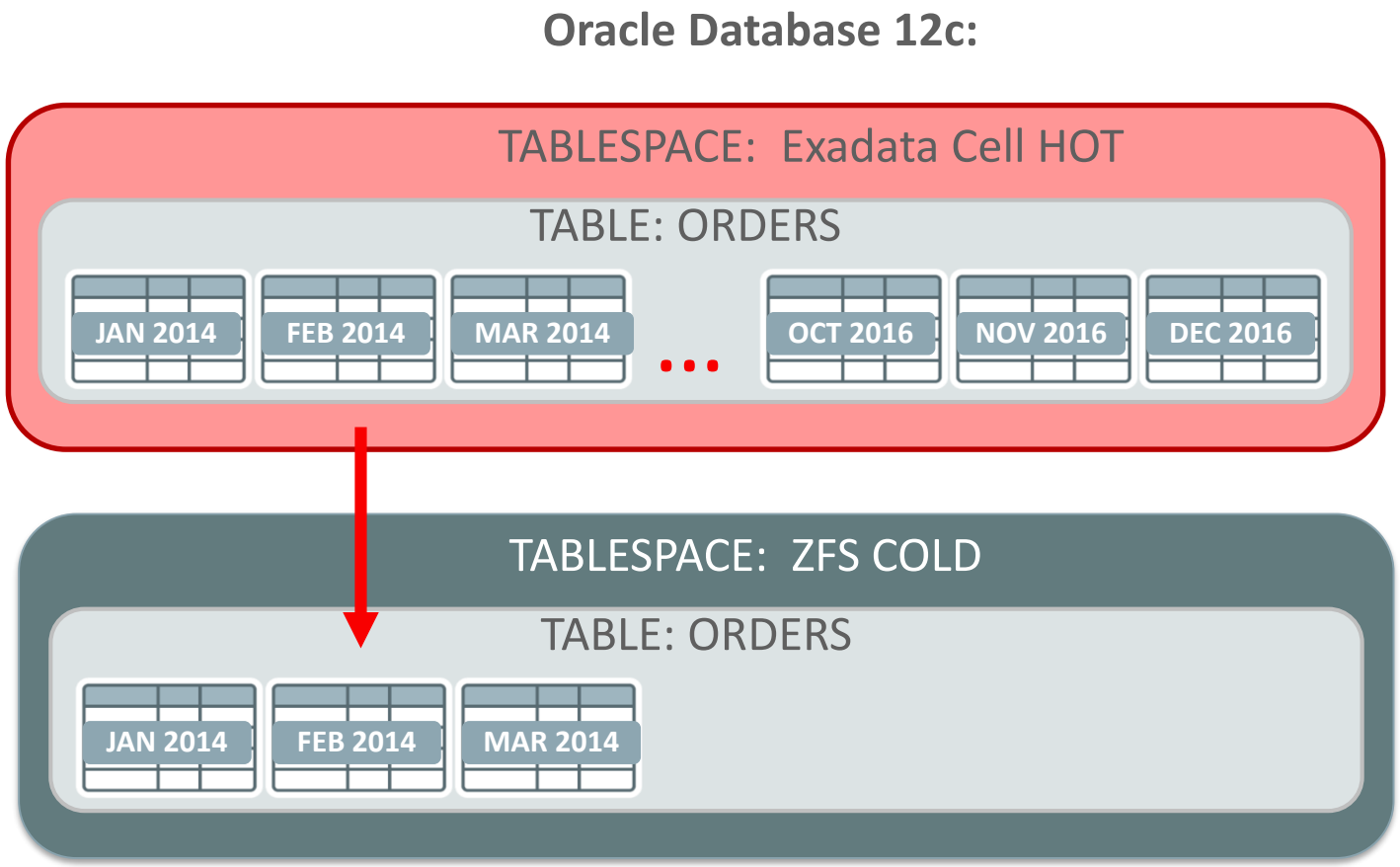


Challenges:
Large tables are difficult to manage

Solution: Partitioning
- Divide and conquer
- Easier data management
- Improve performance

Archive Data to “Cold” Partitions

Automatic Tiering kicks in



Oracle 12.2c *optimized by* ZFS Storage

ANALYTICS

- 107 real-time business variables
- 3X faster trouble-shooting
- Only 12c NAS storage analytics

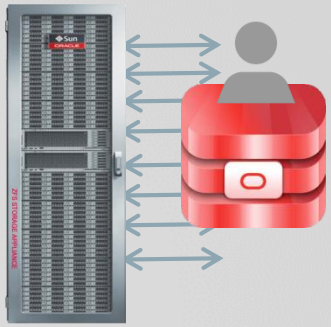
O.I.S.P. (Oracle Intelligent Storage Protocol)

- Automated NAS-12c Database I/O management
- Real-time database-storage optimization
- Faster 12c performance at 1/5 of the cost and time

Hybrid Columnar Compression (HCC)

- 20:1 OLAP database compression
- 5x faster database queries (this one customer, YMMV)
- Native 12c database functionality

OEM and SMU DB Mgmt.

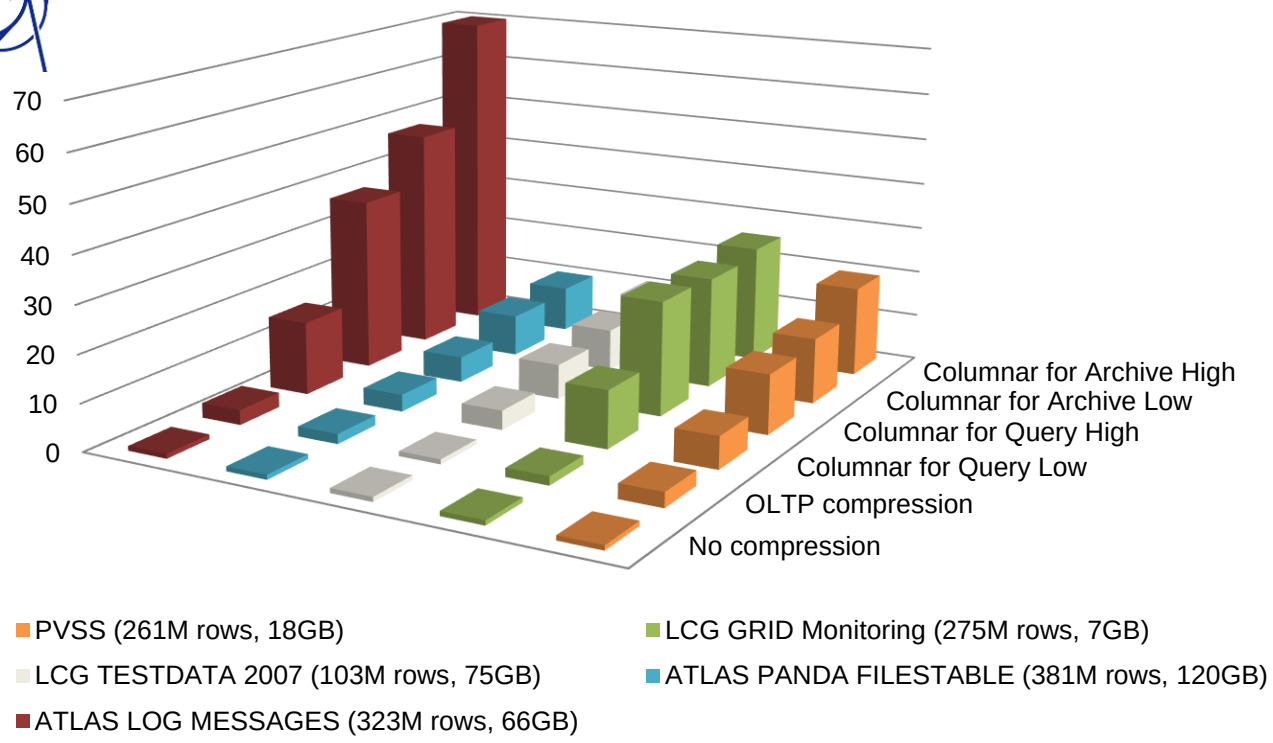


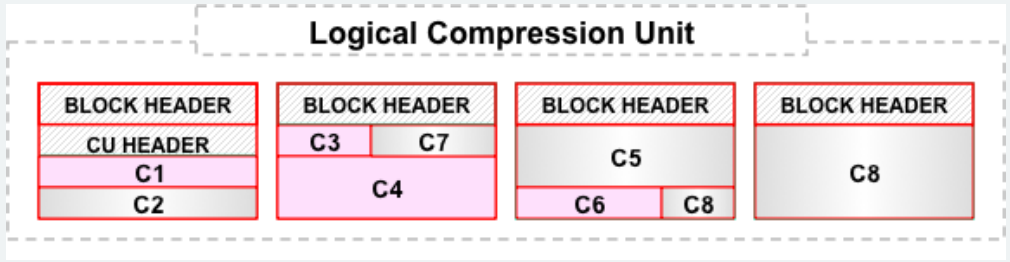
- DBAs self-provision storage
- 79% increase in DevOps efficiency
- Unlimited and instant snaps/clones
- Seamless device patching

HCC Hybrid Columnar Compression

Increase data warehousing performance by 5x, compression at up to 50x

CERN's Measured Compression Results





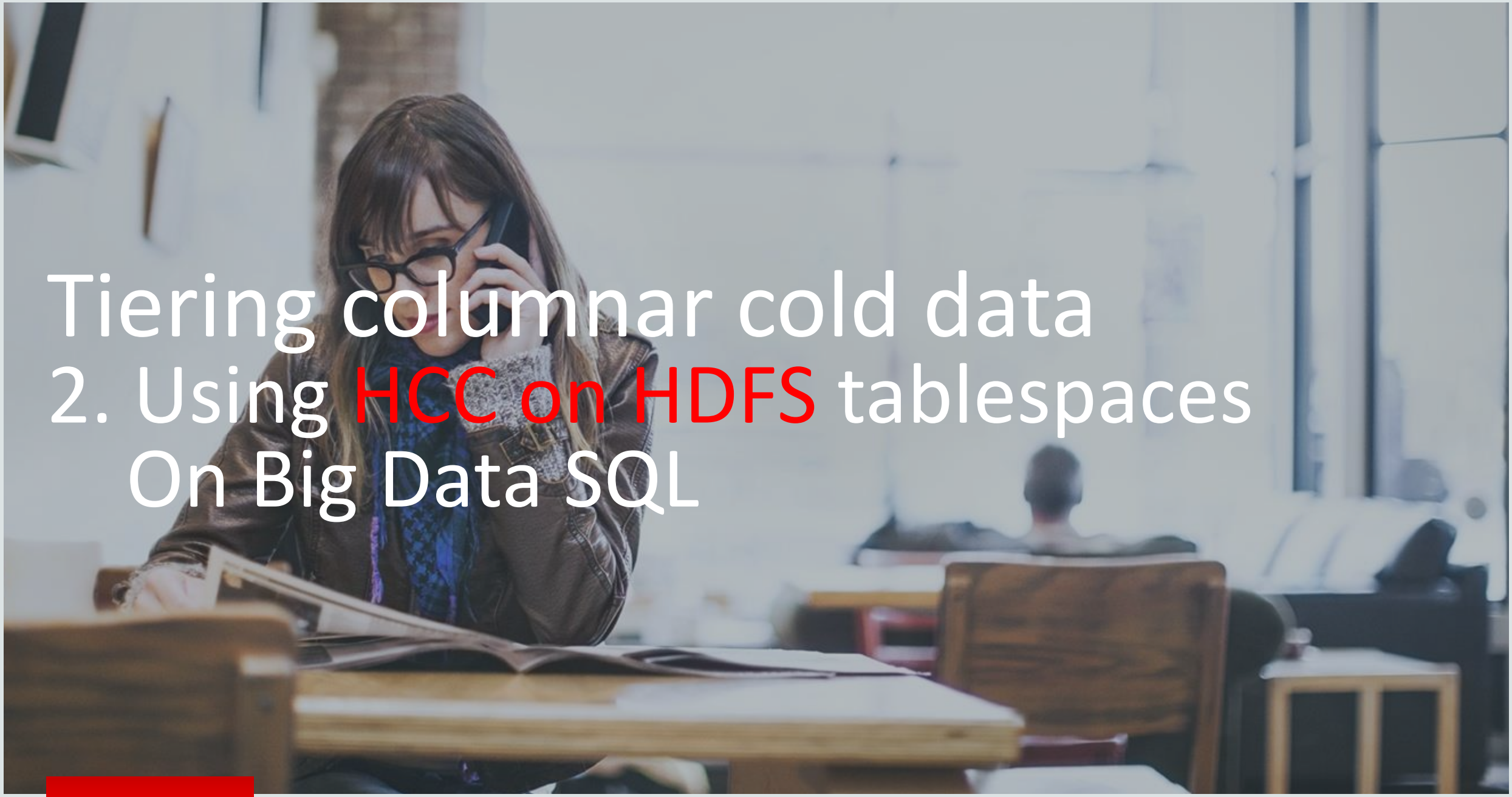
Tables are organized into Compression Unit

- Logical structure spanning multiple database blocks
- Data organized by column during data load
- Each column compressed separately
- Column organization brings similar values close together
- Typically 32K (4 blocks x 8k block size)

- **Re-evaluate compression levels!**
  - ZLIB decompression is CPU intensive

HCC on ZFS: how well will it perform with my data?

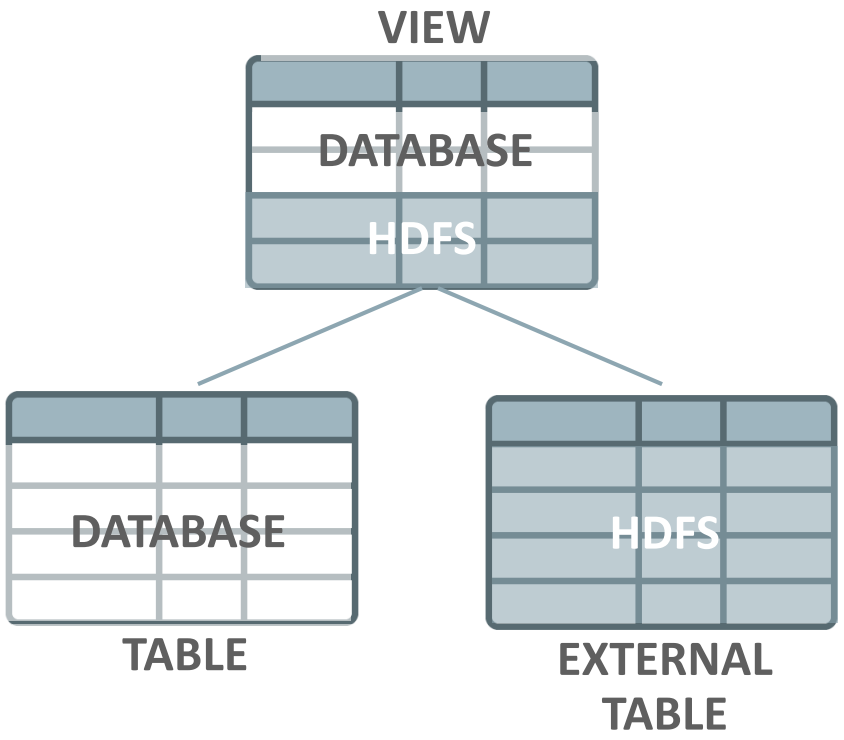
- Unlike Exadata, decompression must be done on RDBMS
  - competing for CPU resources: ZLIB is very expensive
- Emulate with In-Memory PQ
  - *.db_keep_cache_size=4000m
  - *.parallel_degree_policy=AUTO
  - alter table parts storage (buffer_pool keep);
  - exec dbms_stats.gather_table_Stats(USER, ‘PARTS');
- Emulate with no cell CPU help
  - alter session set cell_offload_processing=FALSE;
- Try using other compression levels: QL or AL



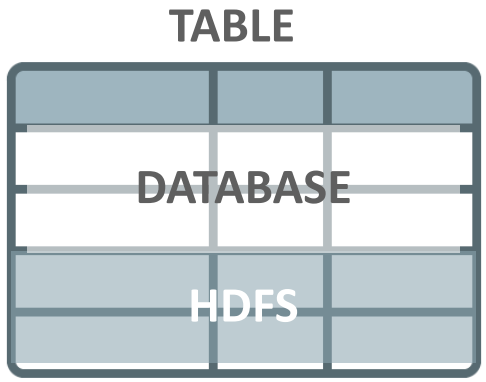
Tiering columnar cold data
2. Using HCC on HDFS tablespaces On Big Data SQL

Archive Data: Big Data SQL Implementation Options

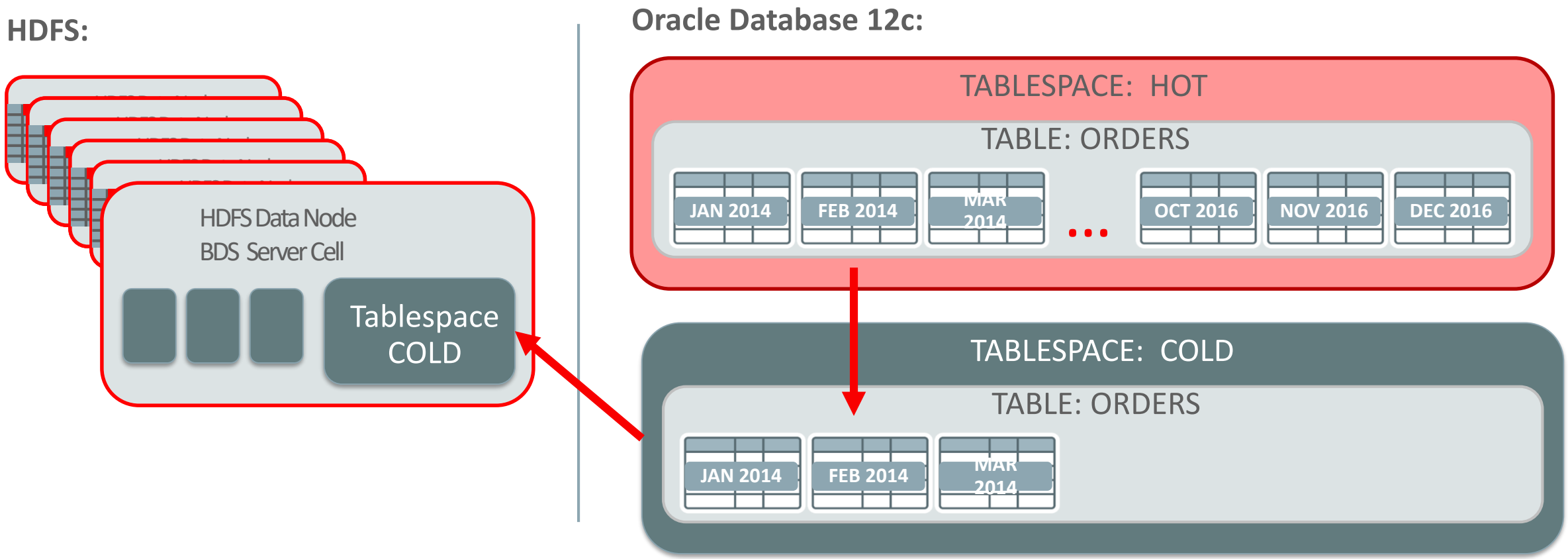
1. View Combines Data Sources



2. Table Storage Split Across Tiers



Archive Data to “Cold” Partitions



Steps: Detailed

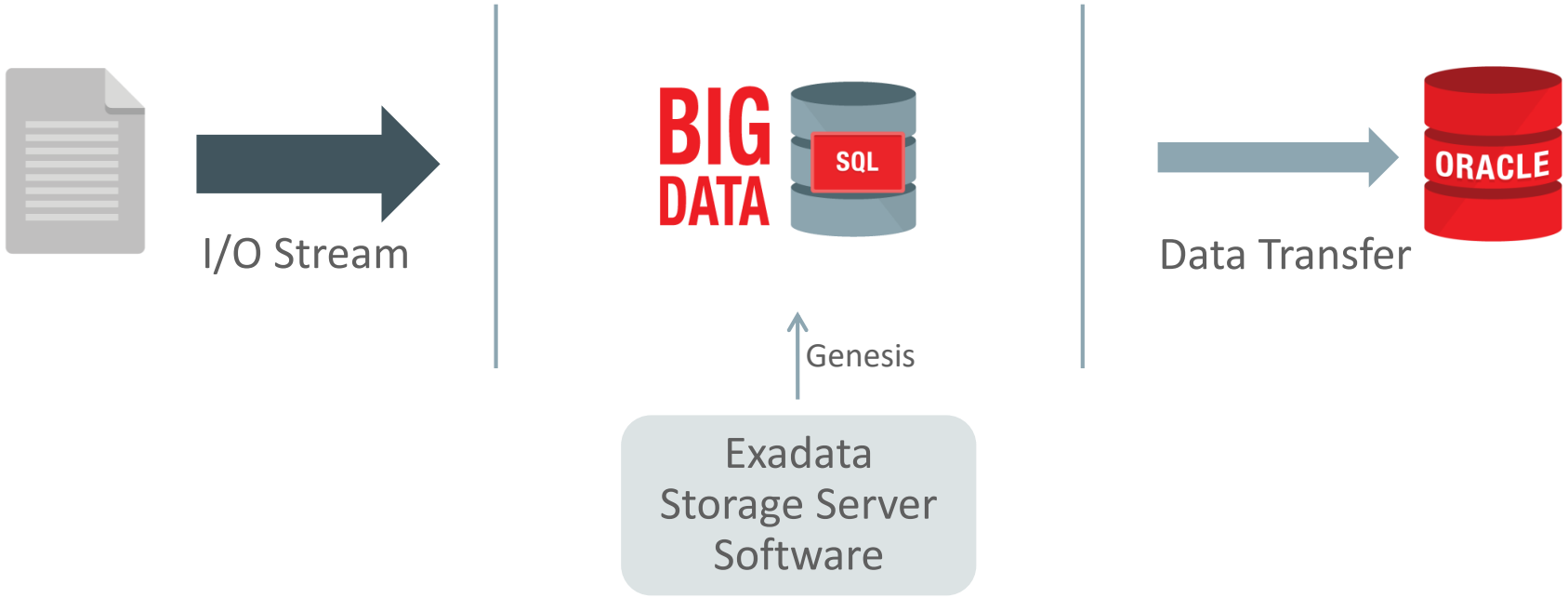
- Set up NFS Gateway or fuse-dfs
- Create a “cold tablespace”
- Move partitions (or tables, or whatever) to that tablespace
- Take the tablespace offline
- Copy the tablespace’s data files to hdfs
  - There is a specific directory where they should go
- Rename the tablespace’s data file to point to the hdfs location (NFS or fuse)
- Bring the tablespace online

Steps: Simplified

- Use **bds-copy-tbs-to-hdfs.sh** script to automate the process
  - Script will optionally install fuse-dfs
- Create a “cold tablespace”
- Move partitions (or tables, or whatever) to that tablespace
- Run single **bds-copy-tbs-to-hdfs.sh** command to do the rest. It will:
  - Take the tablespace offline
  - Copy the tablespace’s data files to hdfs
  - Rename the tablespace’s data file to point to the hdfs location (NFS or fuse)
  - Bring the tablespace online

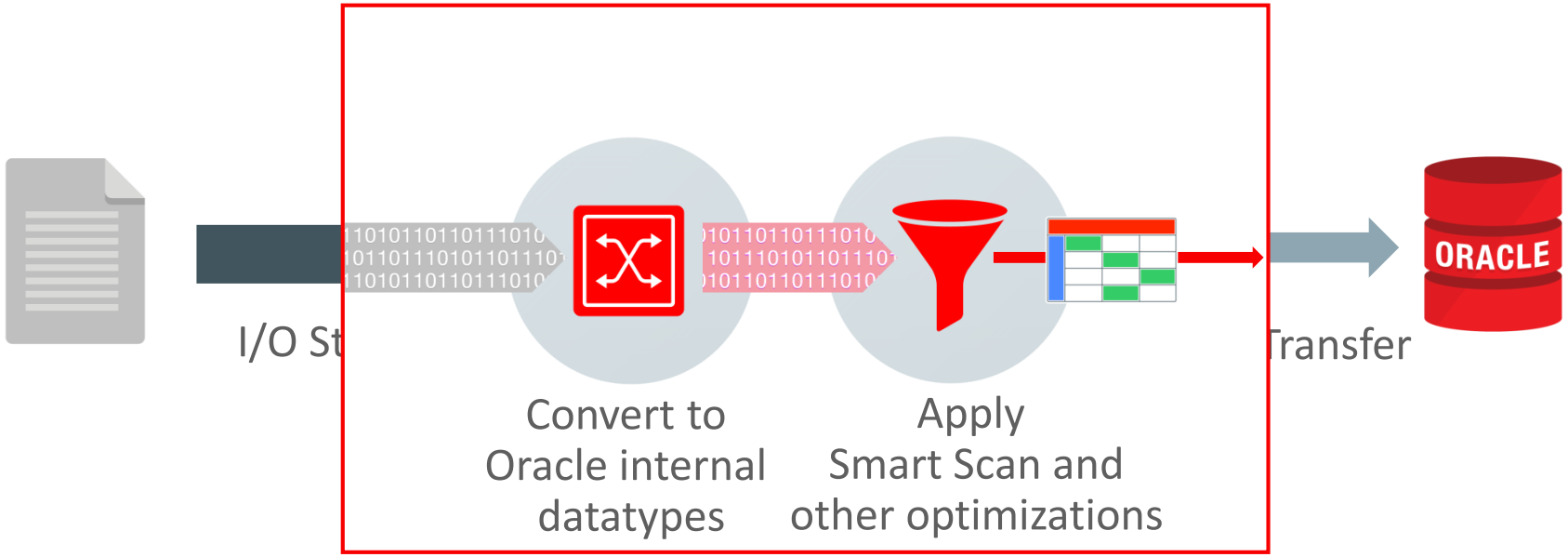
Analytics using Parquet on HDFS with Big Data SQL

Anatomy of a Big Data SQL Cell



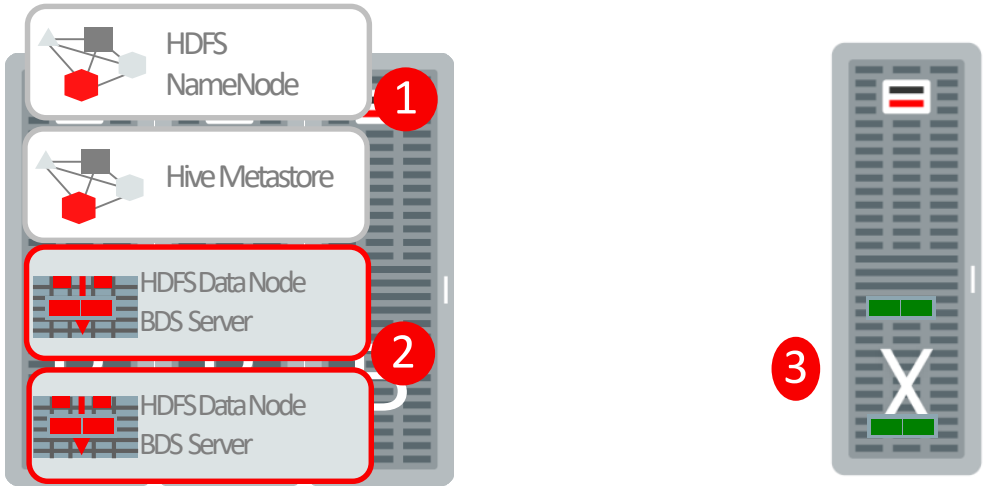
Anatomy of a Big Data SQL Cell

Smart Scan



Big Data SQL Query Execution

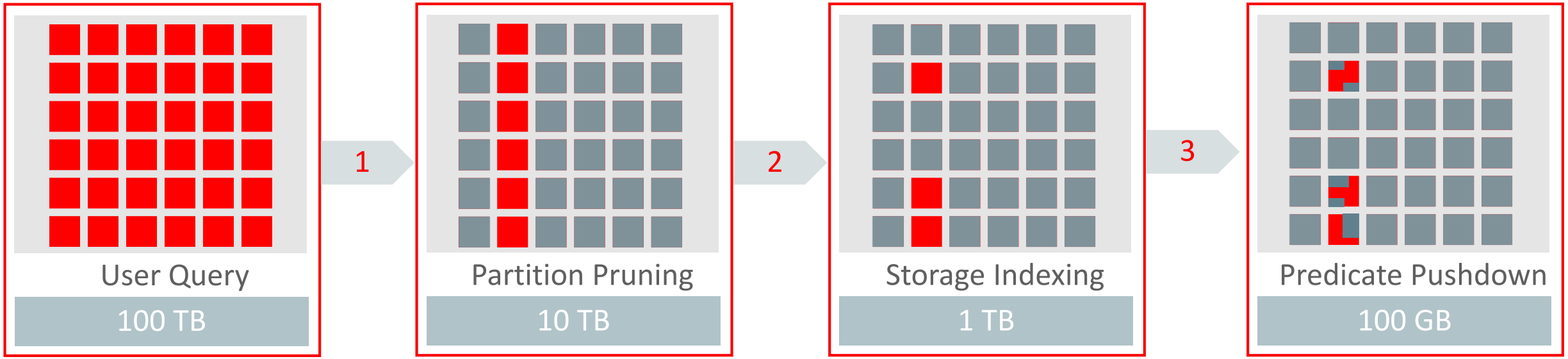
How do we query Hadoop?



1. Describe time determines:
   - Data locations (partition pruned)
   - Data structure
   - Parallelism
2. Fast reads using Big Data SQL Server
   - Schema-for-read using Hadoop classes
   - Predicate pushdown for intelligent sources
   - Smart Scan selects only relevant data
3. Process filtered result
   - Move relevant data to database
   - Join with database tables
   - Apply database security policies

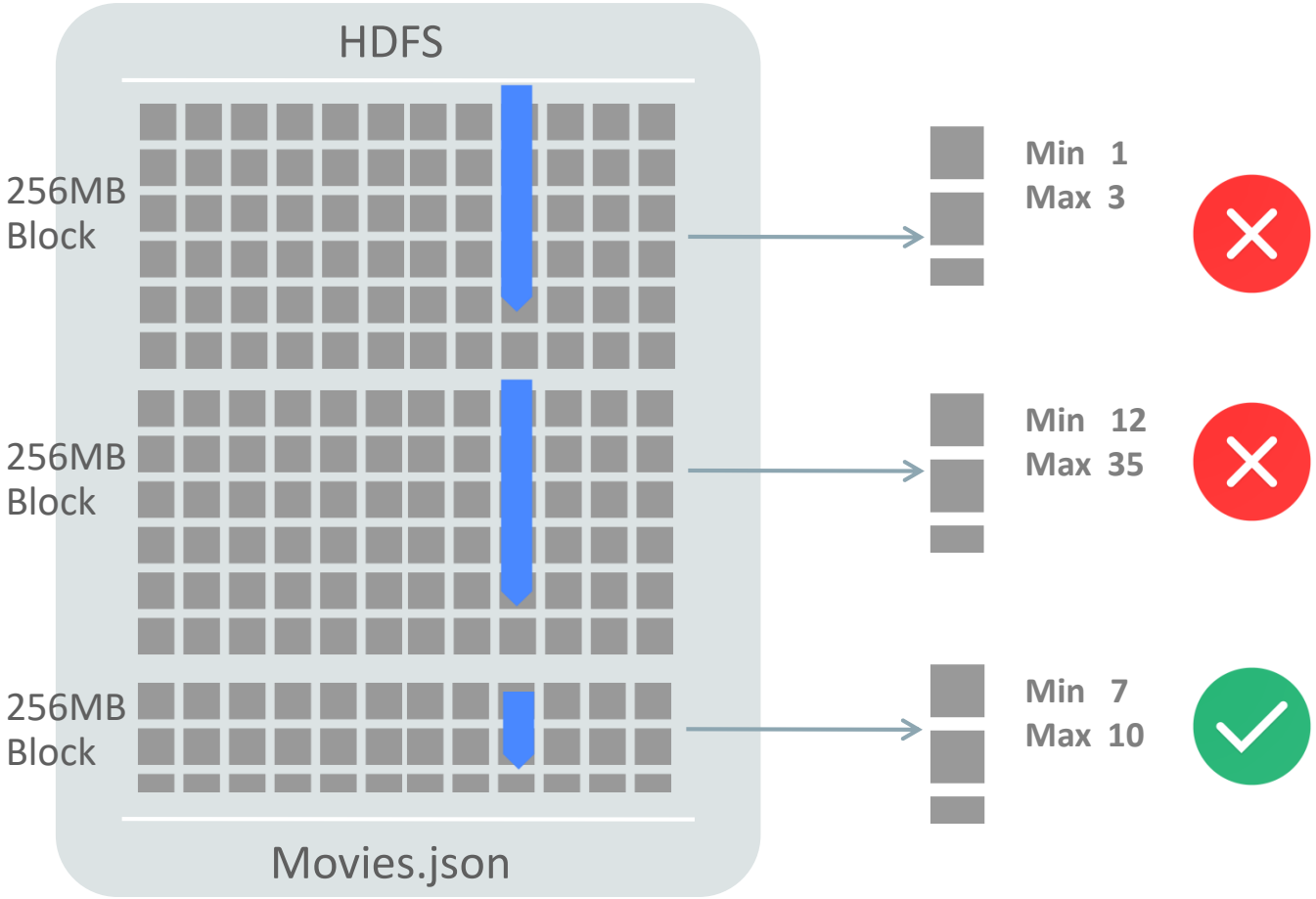
Big Data SQL Performance Features

IO Reduction Features Deliver Compound Results



Big Data SQL Storage Index

Example: Find revenue for movies in a category **9** (Comedy)



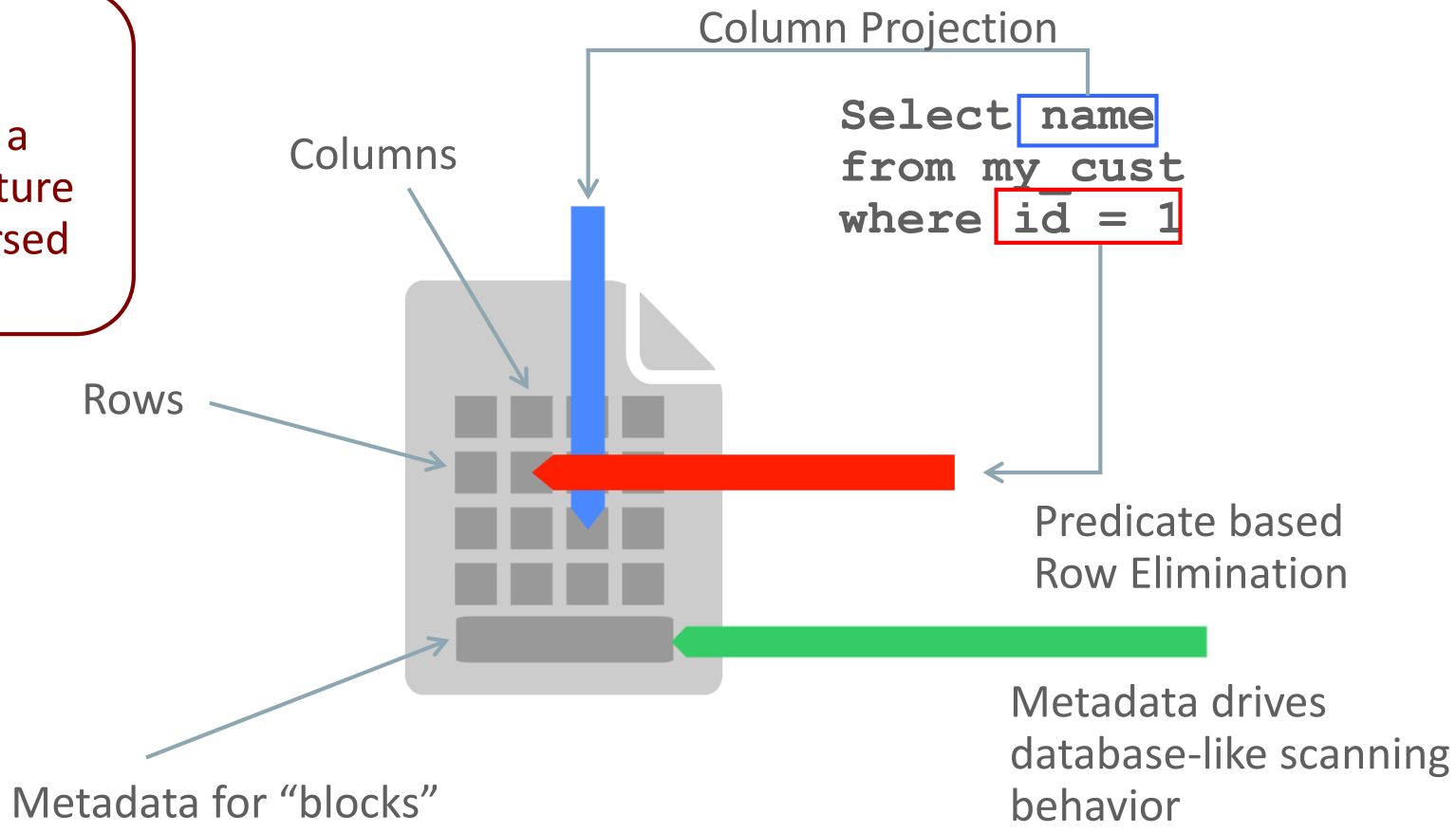
- Storage index provides query speed-up through transparent IO elimination of HDFS Blocks. It's a *negative index*
- Min / max value is recorded for columns included in a storage index (max # of colums = 32)
- Storage index provides partition pruning like performance for un-modeled data sets

How does Parquet Work?

Create and Query Parquet Files

Schema on Write

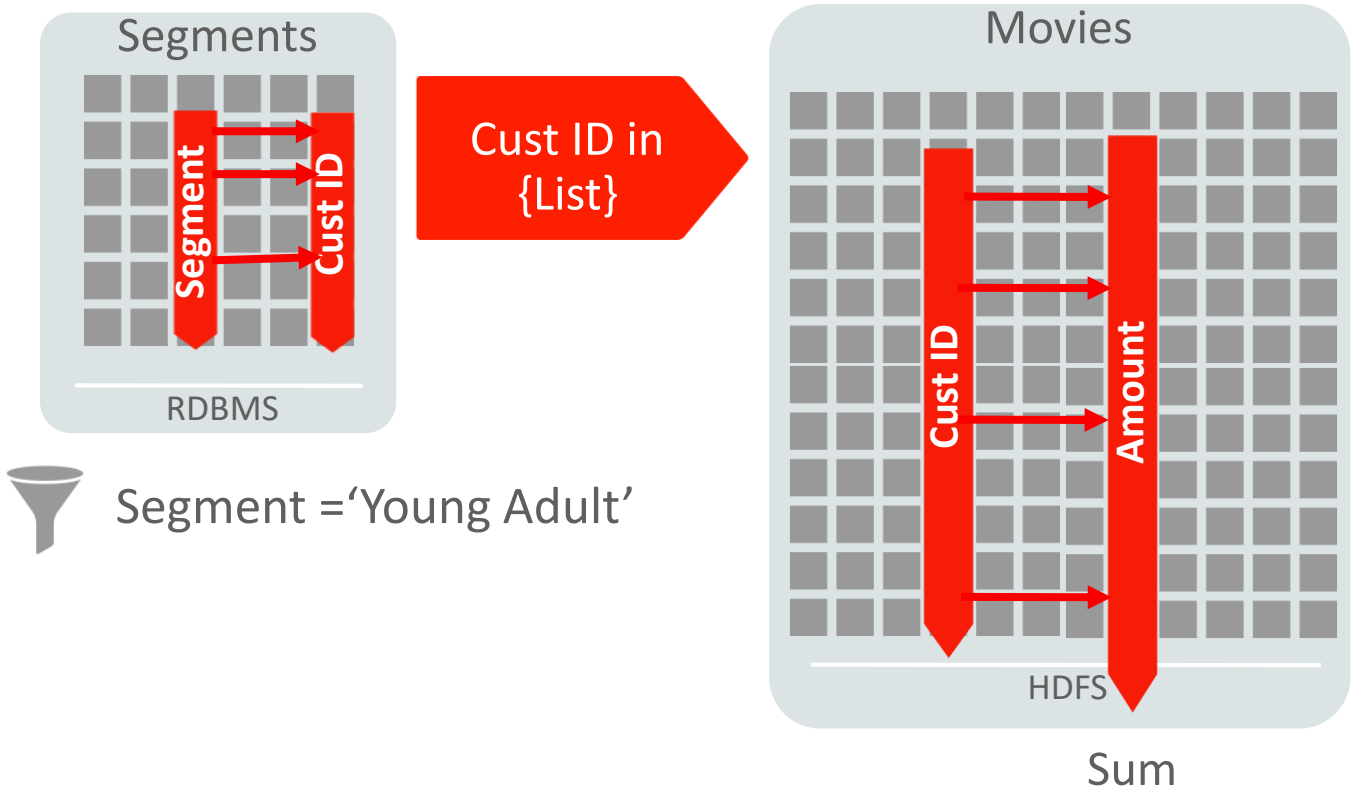
Parquet implements a database storage structure with metadata and parsed data elements



Big Data SQL Performance Features

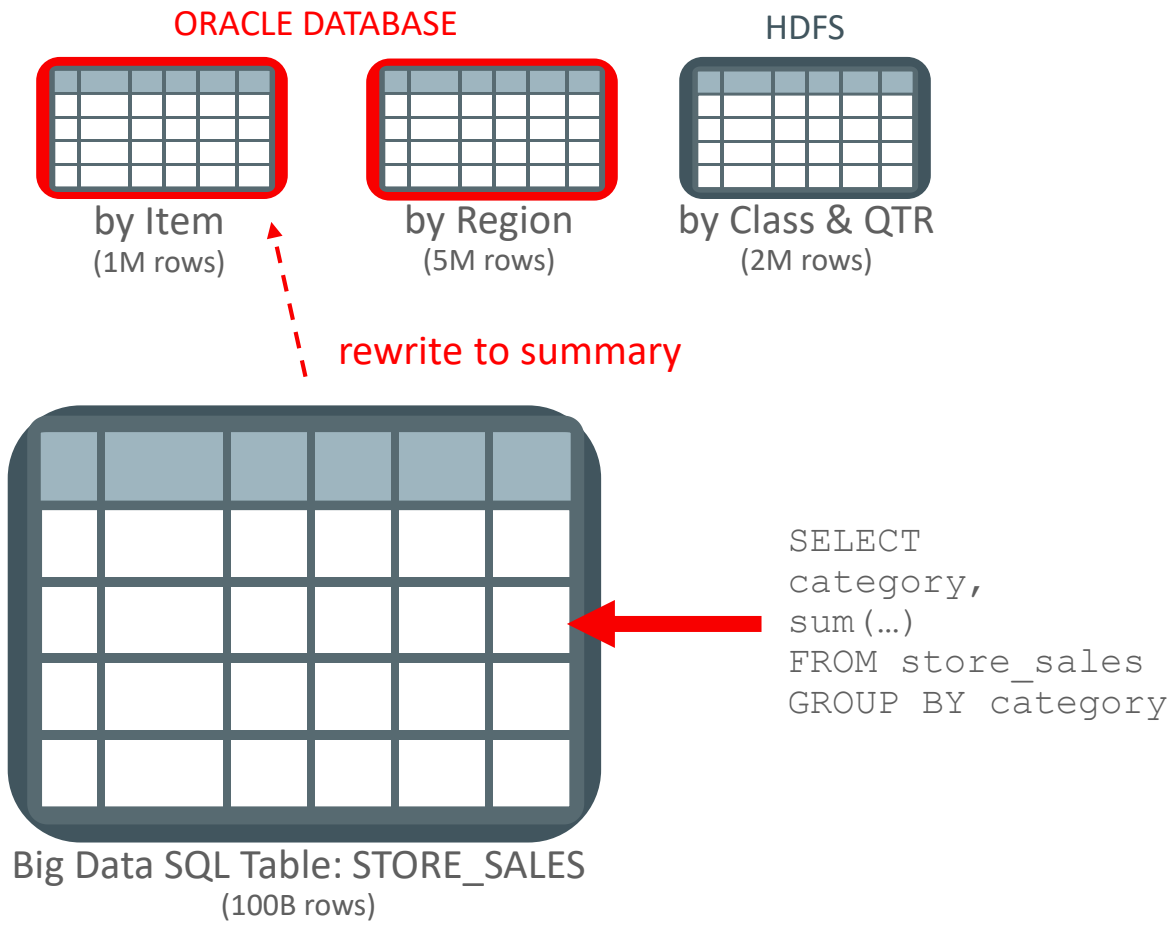
Smart Scan – Execute Joins as Bloom Filters on Hadoop Nodes

Example: Total movie sales for customer segment



- Converts joins of data in multiple tables into scans
- Result:
  - Scans are pushed down to Hadoop nodes and executed locally
  - No data moved to Database to process joins
  - Massive speed up of query
- Works with data spanning DB and Hadoop as well as data in two Hadoop data sets

Enhance Performance with Automatic Query Rewrite



- **Orders of magnitude** performance improvement
- **In-Memory Materialized view** query rewrite automatically redirects detail query to appropriate summary data
  - Store summaries in Oracle Database
  - If available, use existing summaries in HDFS
- No changes to query required

Additional Resources

Join the Conversation

- @TheInMemoryGuy
- https://blogs.oracle.com/in-memory/
- @ExadataPM
- @RogerMacNicol
- https://blogs.oracle.com/smartscan-deep-dive/

White Papers (otn.com)

- Oracle Database In-Memory White Paper
- Oracle Database In-Memory Aggregation Paper
- When to use Oracle Database In-Memory
- Oracle Database In-Memory Advisor

Additional Questions

- DBIM PM: andy.rivenes@oracle.com
- Exadata PM: gurmeet.goini@oracle.com
- ZFS PM: jason.schaffer@oracle.com
- BDS PM: martin.gubar@oracle.com
- My email: roger.macnicol@oracle.com

Q & A

If you have more questions later, feel free to ask


ORACLE®