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.

ORACLE

Oracle Database In-Memory Option

Powering the Real-Time Enterprise

Oracle Database In-Memory Goals

Real Time Analytics

100x

Accelerate OLTP

2x

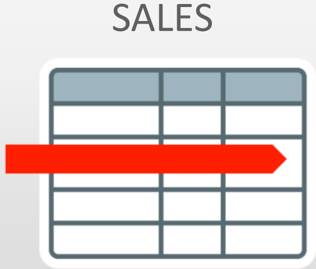
No Changes to Applications

Exploit latest generation hardware

CPU

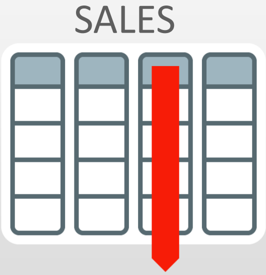
Row Format Databases vs. Column Format Databases

Row



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

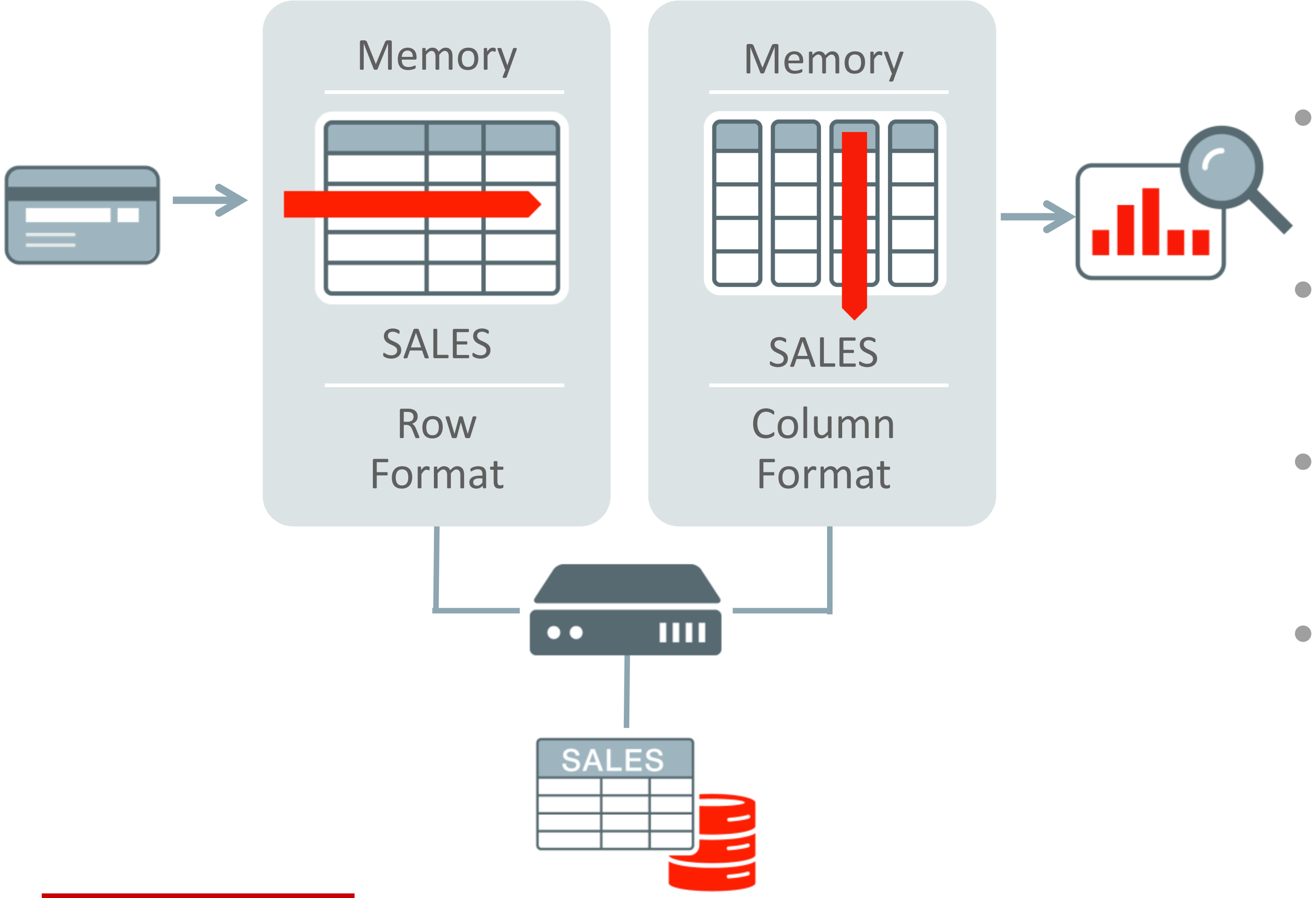
Column



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

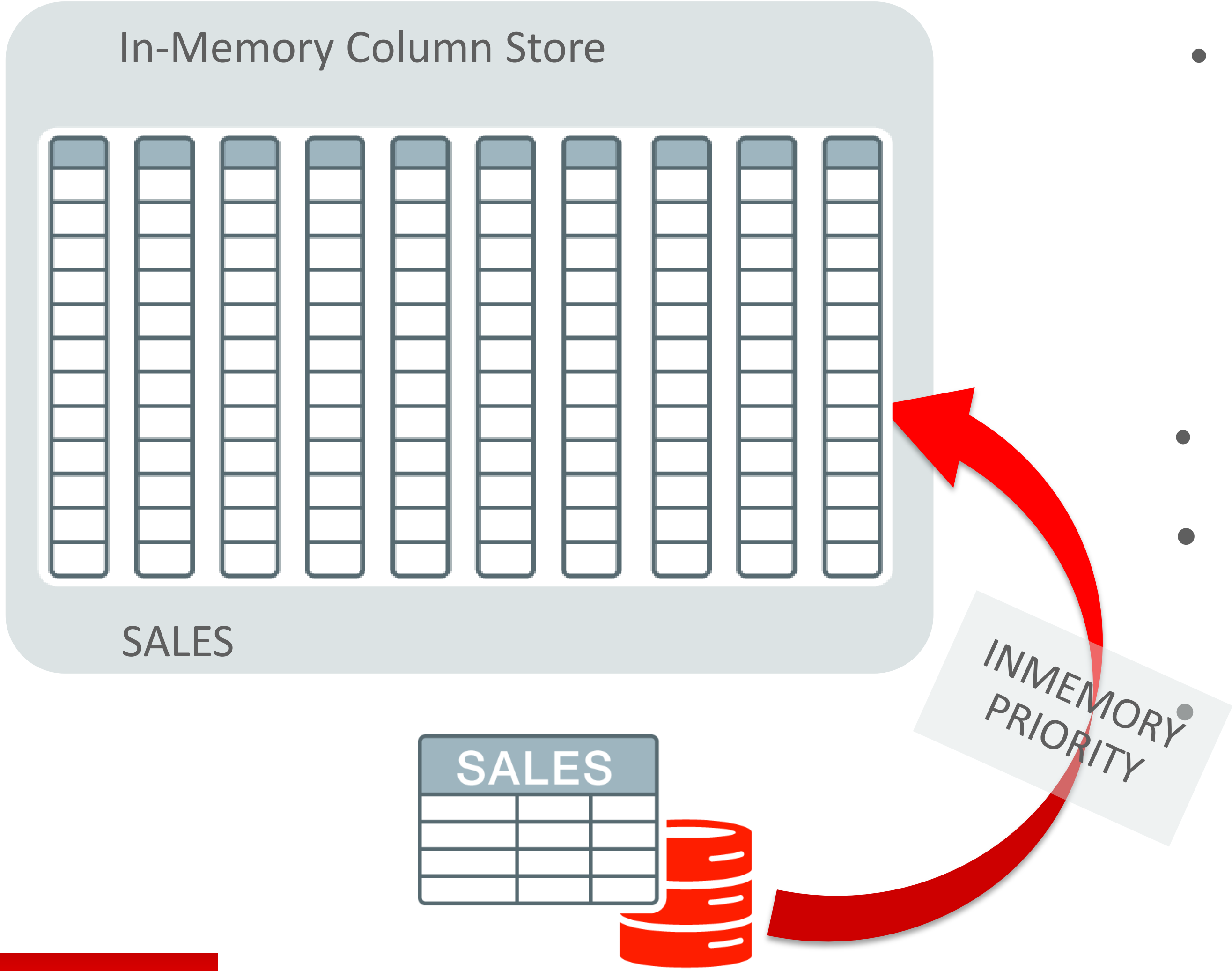
Until Now Must Choose One Format and Suffer Tradeoffs

Breakthrough: Dual Format Database



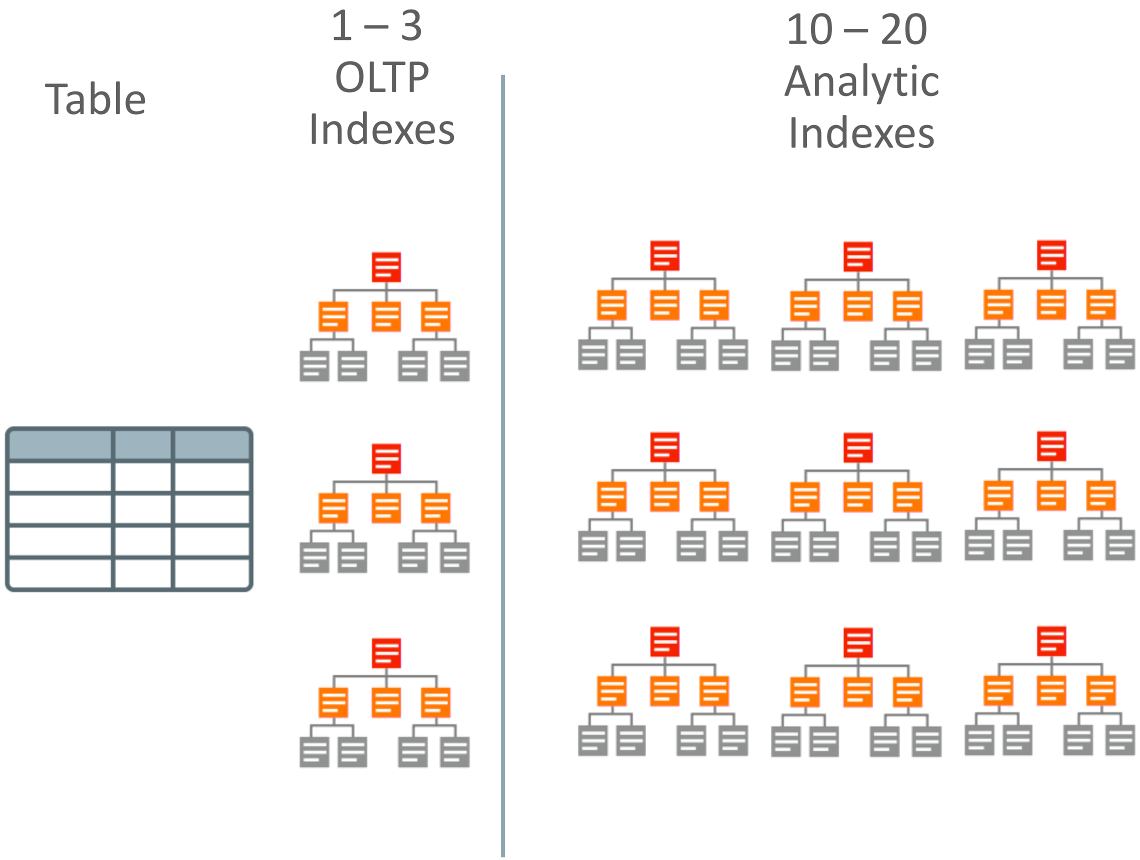
- **BOTH** row and column formats for same table
- Simultaneously active and transactionally consistent
- Analytics & reporting use new In-Memory Column format
- OLTP uses proven row format

Oracle In-Memory Columnar Technology



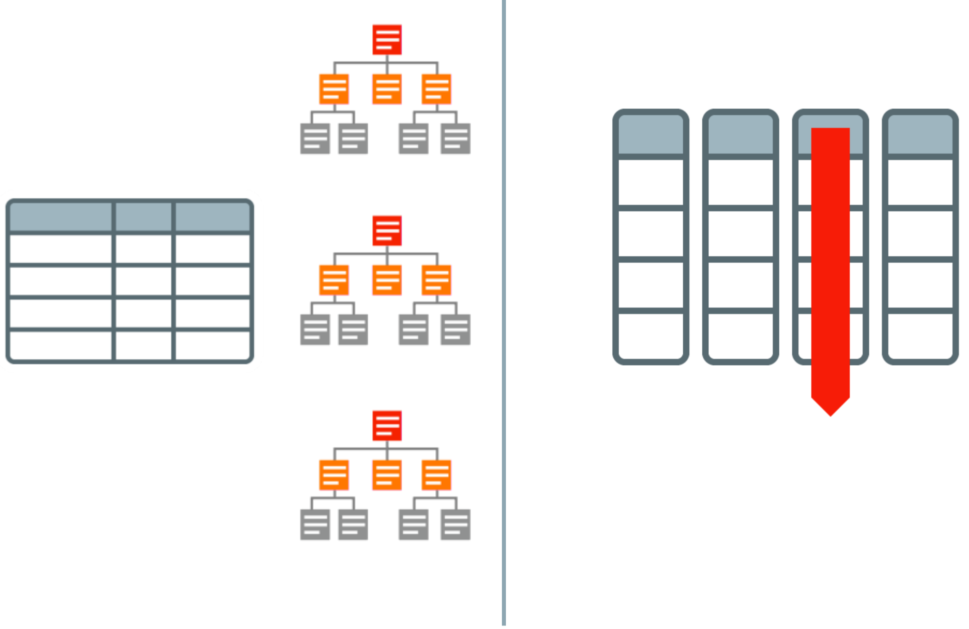
- Pure in-memory column format
  - Not persistent, no logging
  - Quick to change data: fast OLTP
- 2x to 20x compression
- Enabled at table, partition, MEV or tablespace level
- Available on all hardware platforms

Complex OLTP is Slowed by Analytic Indexes



- Most Indexes in complex OLTP (e.g. ERP) databases are only used for analytic queries
- Inserting one row into a table requires updating 10-20 analytic indexes: Slow!
- Indexes only speed up predictable queries & reports

Column Store Replaces Analytic Indexes



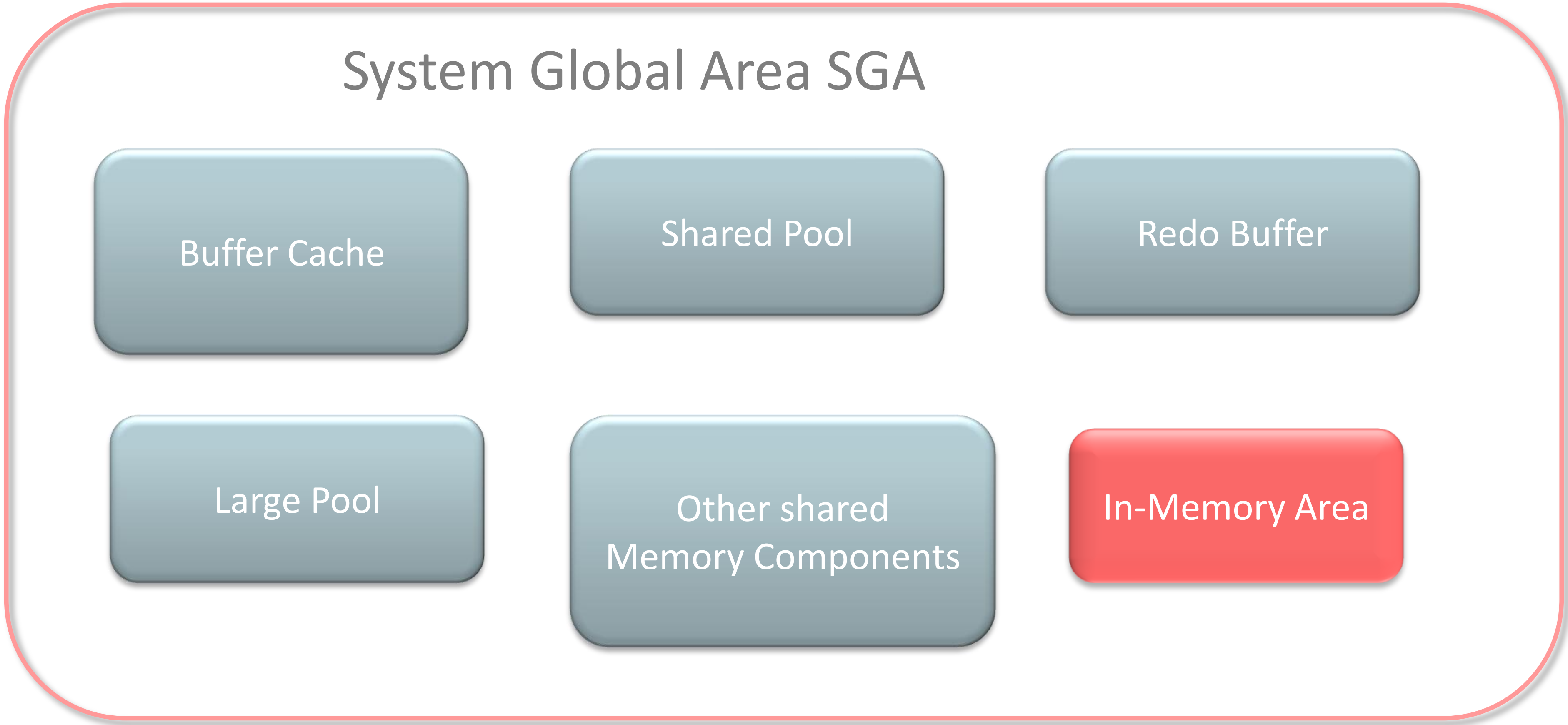
Table

1 – 3
OLTP
Indexes

In-Memory
Column Store

- Fast analytics on any columns
  - Better for unpredictable analytics
  - Less tuning & administration
- Column Store not persistent so update cost is much lower
  - OLTP & batch run faster

Configuring : In-Memory Column Store



Configuring : In-Memory Column Store

```
SELECT * FROM V$SGA;

NAME                  VALUE
------------------  ---------
Fixed Size            2927176
Variable Size       570426808
Database Buffers   4634022912
Redo Buffers         13848576
In-Memory Area     1024483648
```

```
select pool, alloc_bytes/1024/1024
allocated_GB, used_bytes/1024/1024
used_GB, populate_status
from V$INMEMORY_AREA;
```

- Controlled by INMEMORY_SIZE parameter
  - Minimum size of 100MB
  - Default 0
- SGA_TARGET must be large enough to accommodate
- Static Pool

Oracle In-Memory: In-Memory Column Store

New instance parameters:

- INMEMORY_SIZE   = integer [K | M | G]
- INMEMORY_CLAUSE_DEFAULT
  = [INMEMORY] [NO INMEMORY] [other-clauses]
- INMEMORY_FORCE   = { DEFAULT | OFF }
- INMEMORY_MAX_POPULATE_SERVERS
  = Half the effective CPU thread count
- INMEMORY_QUERY   = { ENABLE | DISABLE }

Populating : In-Memory Column Store

- Populate is the term used to bring data into the In-Memory column store
- Populate is used instead of load because load is commonly used to mean inserting new data into the database
- Populate doesn't bring new data into the database, it brings existing data into memory and formats it in an optimized columnar format
- Population is completed by a new set of background processes
  - ORA_W001_orcl
  - Number of processes controlled by INMEMORY_MAX_POPULATE_SERVERS

Populating : In-Memory Column Store

```
ALTER TABLE sales INMEMORY;

ALTER TABLE sales NO INMEMORY;
```

```
CREATE TABLE customers ……
PARTITION BY LIST
   (PARTITION p1 …… INMEMORY,
   (PARTITION p2 …… NO INMEMORY);
```

- New INMEMORY ATTRIBUTE
- Following segment types are eligible
  - Tables
  - Partitions
  - Subpartition
  - Materialized views
- Following segment types not eligible
  - IOTs
  - Hash clusters
  - Out of line LOBs

Pure OLTP Features

Populating : In-Memory Column Store

```
ALTER TABLE sales INMEMORY
NO INMEMORY (PROD_ID);
```

```
CREATE TABLE orders
   (c1 number,
    c2 varchar(20),
    c3 number)
INMEMORY PRIORITY CRITICAL
NO INMEMORY (c1);
```

- Possible to populate only certain columns from a table or partition
- Order in which objects are populated controlled by **PRIORITY** subclause
  - Critical, high, medium, low – populate after startup
  - Default none - populate on first access
  - Does not control the speed of population

Populating : In-Memory Column Store

```
ALTER MATERIALIZED VIEW mv1 INMEMORY
MEMCOMPRESS FOR QUERY;
```

```
CREATE TABLE trades
   (Name varchar(20),
    Desc varchar(200))
INMEMORY
MEMCOMPRESS FOR DML(desc);
```

- Objects compressed during population
- Queries execute directly against the compressed columns
- Compression ratios can vary from 2X – 20X
- Data is only decompressed when it is required for the result set
- Controlled by **MEMCOMPRESS** subclause
- Multiple levels of compression

ORACLE

Populating : In-Memory Column Store

```
CREATE TABLE ORDERS ......
PARTITION BY RANGE ......
  (PARTITION p1 ......
   INMEMORY NO MEMCOMPRESS
   PARTITION p2 ......
   INMEMORY MEMCOMPRESS FOR DML,
   PARTITION p3 ......
   INMEMORY MEMCOMPRESS FOR QUERY,
   :
  PARTITION p200 ......
   INMEMORY MEMCOMPRESS FOR CAPACITY
  );
```

- Different compression levels
  - FOR DML
    Use on tables or partitions with very active DML activity
  - FOR QUERY
    Default mode for most tables
  - FOR CAPACITY
    For less frequently accessed segments
- Possible to use a different level for different partitions in a table
- Easy to switch levels as part of ILM strategy

Identifying : Tables with INMEMORY Attribute

```
SELECT table_name, inmemory
FROM   USER_TABLES;

TABLE_NAME    INMEMORY
------------   --------
CHANNELS      DISABLED
COSTS
CUSTOMERS     DISABLED
PRODUCTS      ENABLED
SALES
TIMES         DISABLED
```

- New INMEMORY column in *_TABLES dictionary tables
- INMEMORY is a segment attribute
- USER_TABLES doesn't display segment attributes for logical objects
- Both COSTS & SALES are partitioned => logical objects
- INMEMORY attribute also reported in *_TAB_PARTITIONS

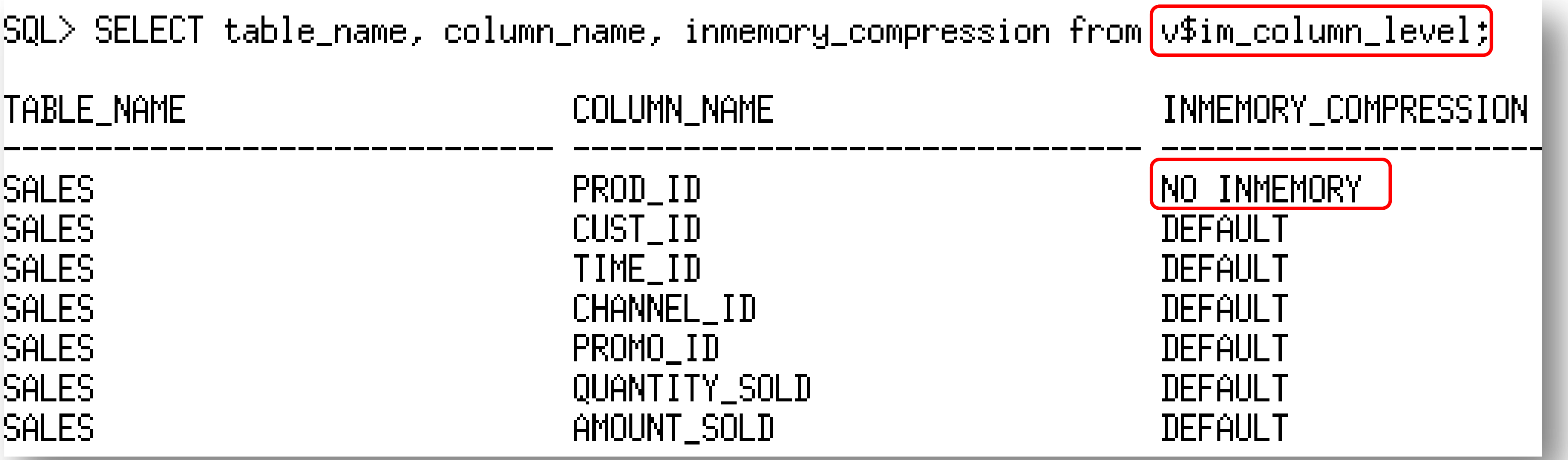
Identifying : Tables with INMEMORY Attribute

```
SELECT segment_name  name,
       population_status status
FROM   v$IM_SEGMENTS;

NAME          STATUS
------------  --------
PRODUCTS      COMPLETED
SALES         STARTED
```

- New view v$IM_SEGMENTS
- Indicate:
  - Objects populated in memory
  - Current population status
  - Can also be used to determine compression ratio achieved

Identifying : Columns without the INMEMORY Attribute



```
SQL> SELECT table_name, column_name, inmemory_compression from v$im_column_level;

TABLE_NAME                     COLUMN_NAME                    INMEMORY_COMPRESSION
------------------------------ ------------------------------ --------------------
SALES                          PROD_ID                        NO INMEMORY
SALES                          CUST_ID                        DEFAULT
SALES                          TIME_ID                        DEFAULT
SALES                          CHANNEL_ID                     DEFAULT
SALES                          PROMO_ID                       DEFAULT
SALES                          QUANTITY_SOLD                  DEFAULT
SALES                          AMOUNT_SOLD                    DEFAULT
```

Monitoring In-Memory

V$IM_SEGMENTS | V$IM_USER_SEGMENTS and DBA|ALL|USER_TABLES;

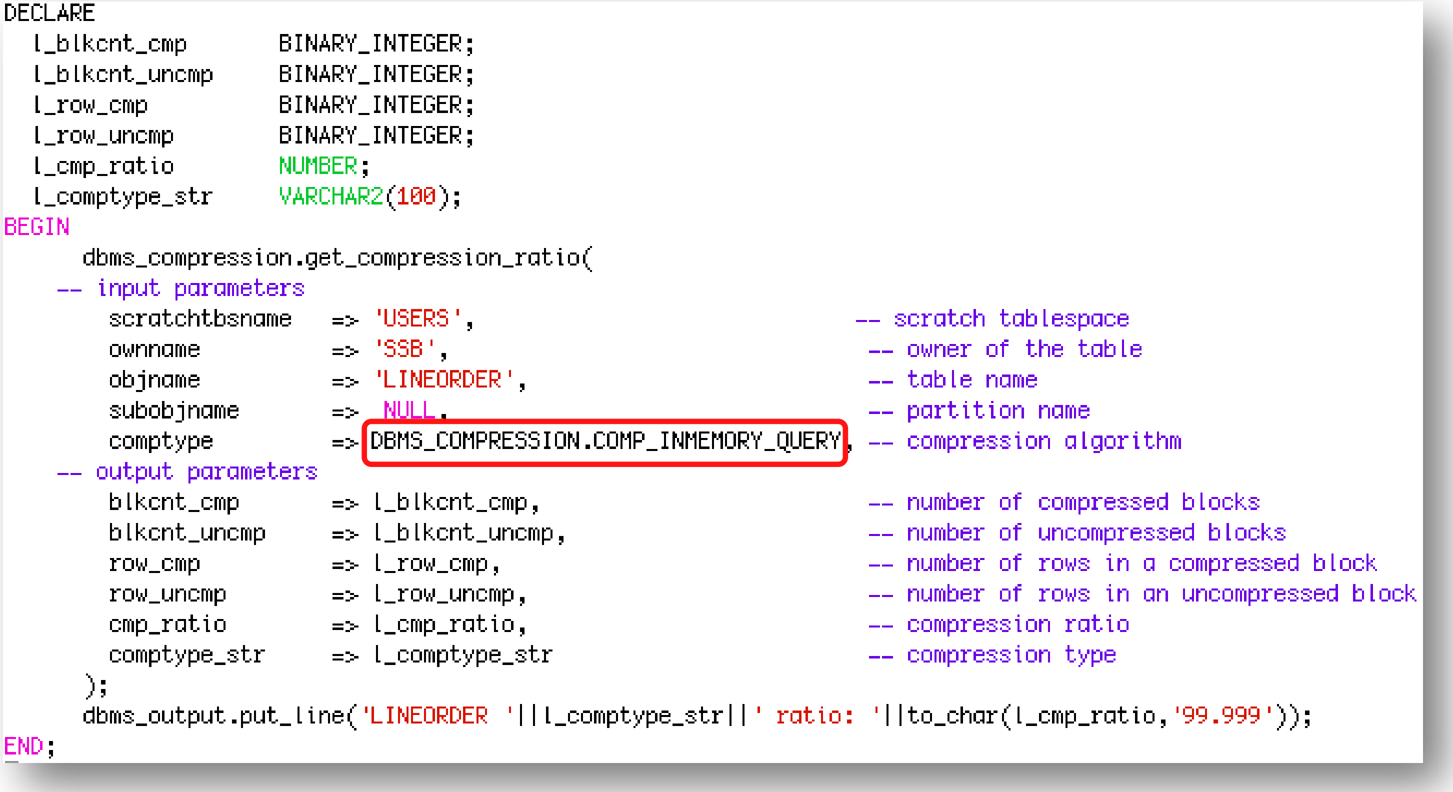
```
SELECT sum(bytes) as diskSize, sum(inmemory_size) as inMemSize,
sum(bytes_not_populated) as notInMemory
FROM v$im_segments;
```

```
SELECT sum(bytes), sum(inmemory_size), sum(bytes)/sum(inmemory_size)  as
compressRatio
FROM  v$im_segments;
```

```
SELECT  table_name, cache, inmemory_priority, inmemory_distribute,
inmemory_compression
FROM user_tables;
```

Population has completed when column BYTES_NOT_POPULATED = 0

Oracle Compression Advisor in 12.1.0.2+



```
DECLARE
  l_blkcnt_cmp         BINARY_INTEGER;
  l_blkcnt_uncmp       BINARY_INTEGER;
  l_row_cmp            BINARY_INTEGER;
  l_row_uncmp          BINARY_INTEGER;
  l_cmp_ratio          NUMBER;
  l_comptype_str       VARCHAR2(100);
BEGIN
      dbms_compression.get_compression_ratio(
    -- input parameters
        scratchtbsname   => 'USERS',                            -- scratch tablespace
        ownname          => 'SSB',                               -- owner of the table
        objname          => 'LINEORDER',                         -- table name
        subobjname       =>  NULL,                               -- partition name
        comptype         => DBMS_COMPRESSION.COMP_INMEMORY_QUERY, -- compression algorithm
    -- output parameters
        blkcnt_cmp       => l_blkcnt_cmp,                        -- number of compressed blocks
        blkcnt_uncmp     => l_blkcnt_uncmp,                      -- number of uncompressed blocks
        row_cmp          => l_row_cmp,                           -- number of rows in a compressed block
        row_uncmp        => l_row_uncmp,                         -- number of rows in an uncompressed block
        cmp_ratio        => l_cmp_ratio,                         -- compression ratio
        comptype_str     => l_comptype_str                       -- compression type
      );
      dbms_output.put_line('LINEORDER '||l_comptype_str||' ratio: '||to_char(l_cmp_ratio,'99.999'));
END;
```

- Easy way to determine memory requirements
- Use DBMS_COMPRESSION
- Applies MEMCOMPRESS to sample set of data from a table
- Returns estimated compression ratio

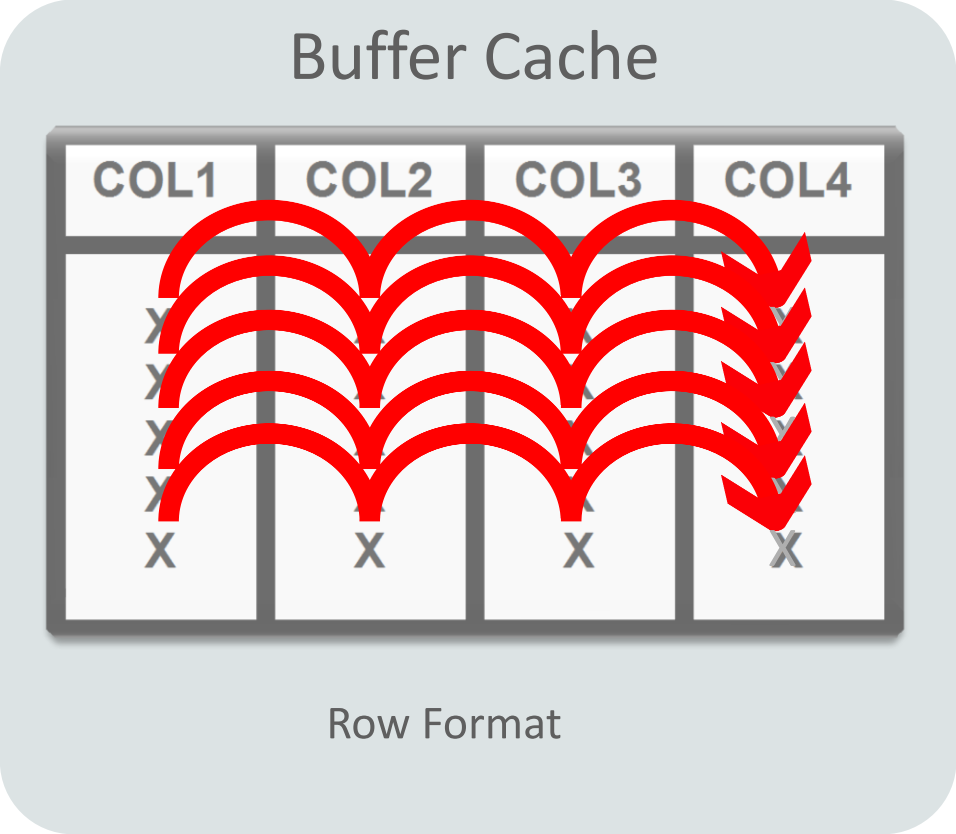
How to enable In Memory Column Store:

1. Ensure that the database is at 12.1.0 or higher compatibility level
2. Set the INMEMORY_SIZE initialization parameter to a non-zero value
3. Increase SGA Target → IMDB is a part of that
4. When you set this parameter in a server parameter file (SPFILE) using the ALTER SYSTEM statement, you must specify SCOPE=SPFILE * The minimum setting is 100M

   e.g. ALTER SYSTEM SET INMEMORY_SIZE=5G SCOPE=SPFILE;

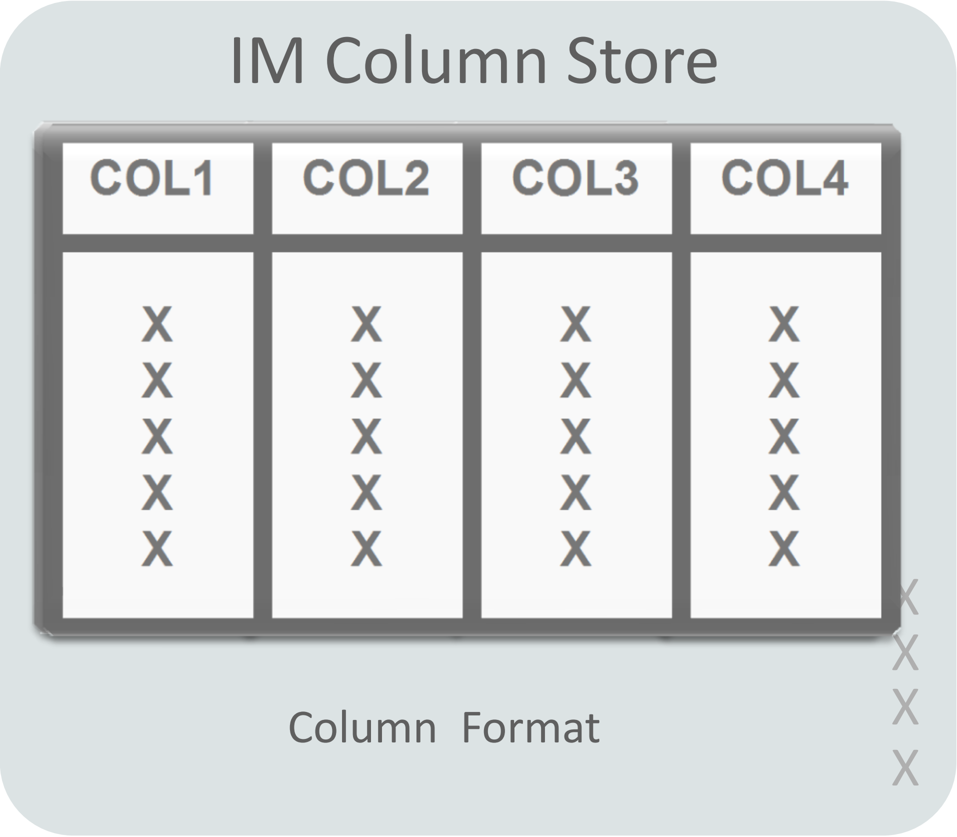
5. Increase PGA → To the SORT and GROUP BY operation did not reduce performance of IN-Memory→ Sort goes to disk if there's not enough PGA
6. Restart the database.
7. Populate tables/partitions/columns/tablespaces in the In-Memory Column Store
8. Make invisible/drop any analytic indexes that existed on the table to speed up OLTP

Why is an In-Memory scan faster than the buffer cache?



```
SELECT COL4 FROM MYTABLE;
```

Why is an In-Memory scan faster than the buffer cache?



IM Column Store

COL1	COL2	COL3	COL4
X	X	X	X
X	X	X	X
X	X	X	X
X	X	X	X
X	X	X	X

Column Format

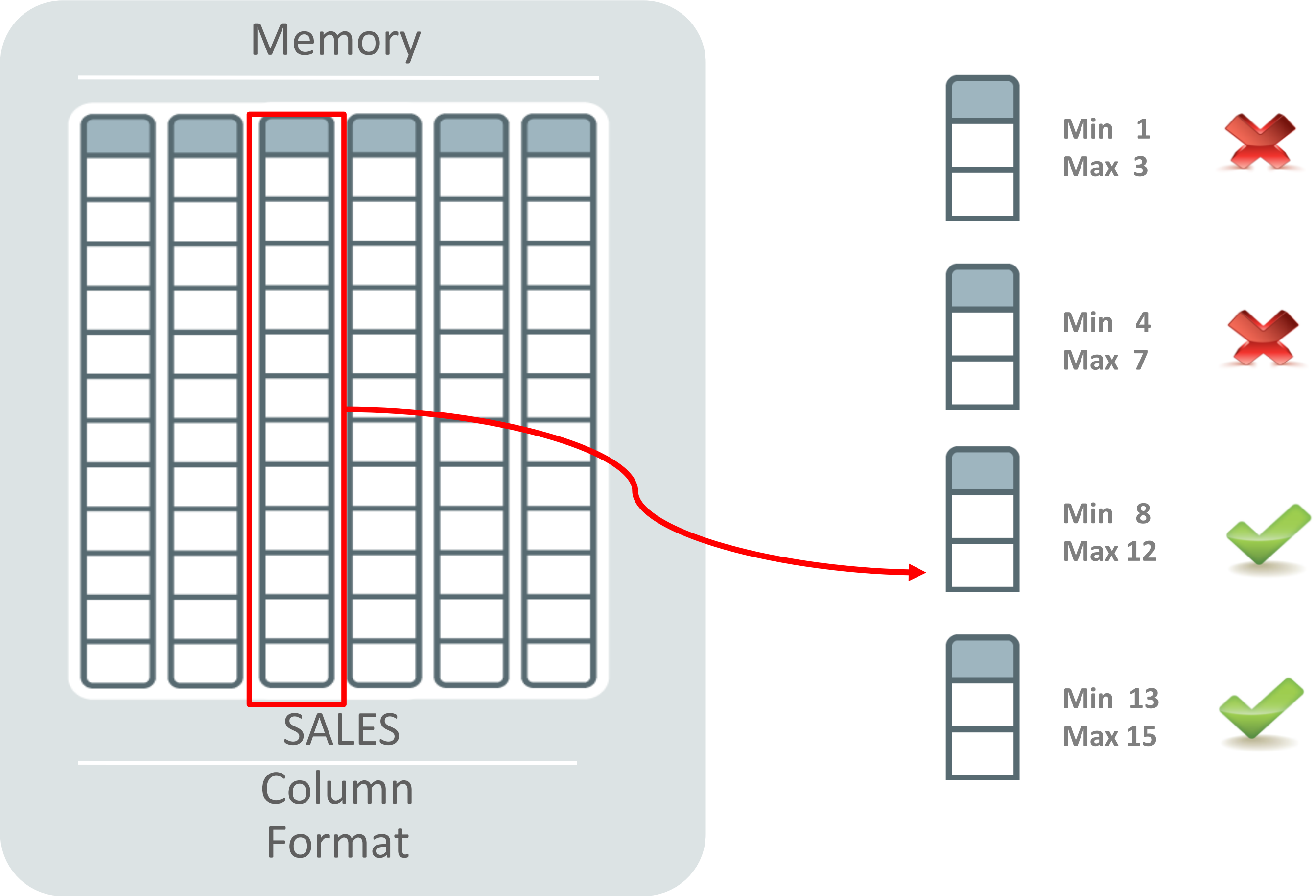
`SELECT **COL4** FROM MYTABLE;`

RESULT

Accesses only the columns needed by a query & applies any WHERE clause filter predicates to these columns directly without having to decompress them

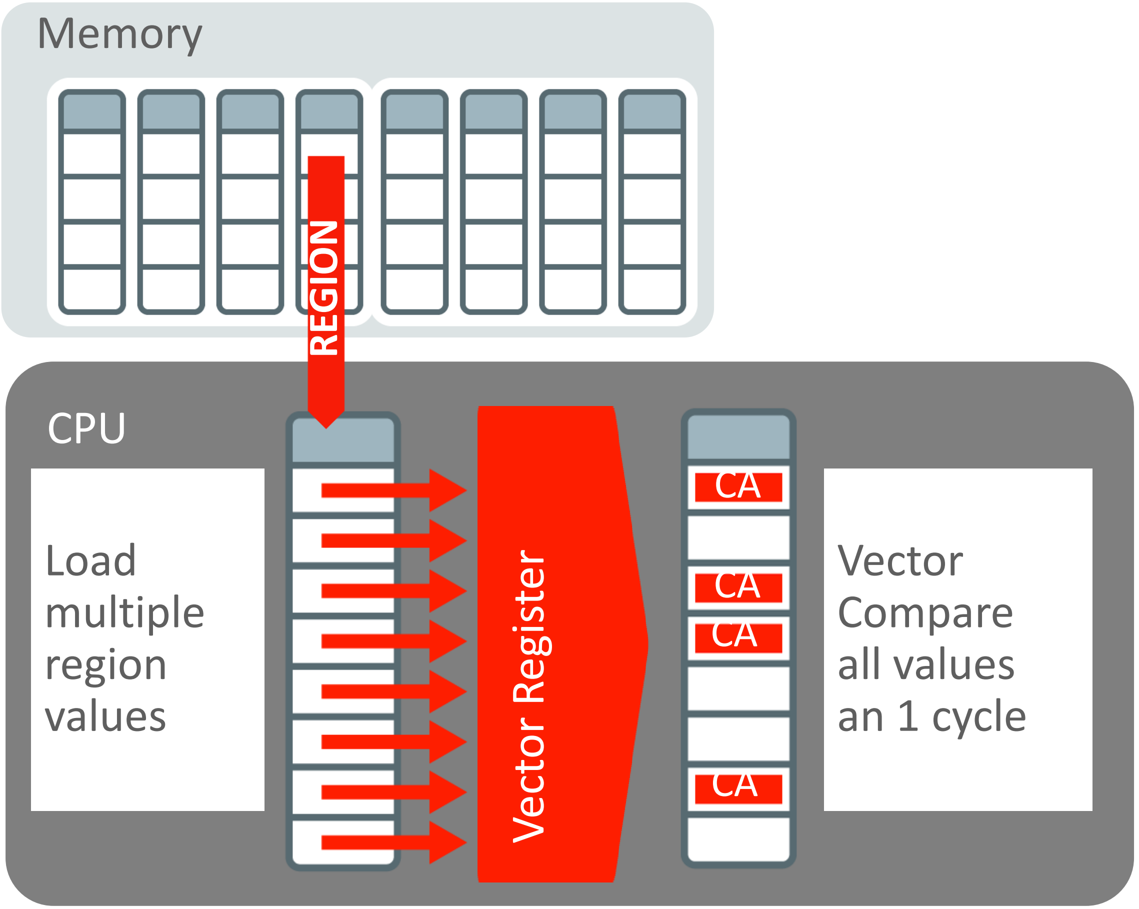
Oracle In-Memory Column Store Storage Index

Example: Find sales from stores with a store_id of 8 or higher



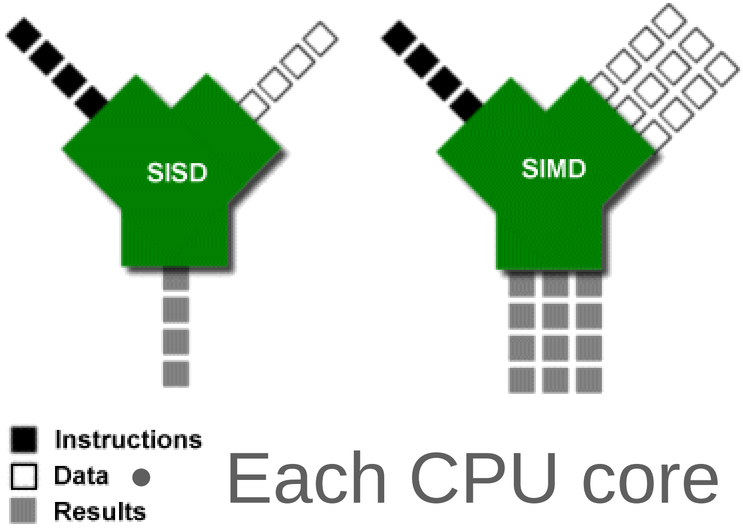
- Each column is the made up of multiple column units
- Min / max value is recorded for each column unit in a storage index
- Storage index provides partition pruning like performance for ALL queries

Orders of Magnitude Faster Analytic Data Scans


Memory
REGION
CPU
Load multiple region values
Vector Register
CA
CA
CA
CA
Vector Compare all values an 1 cycle

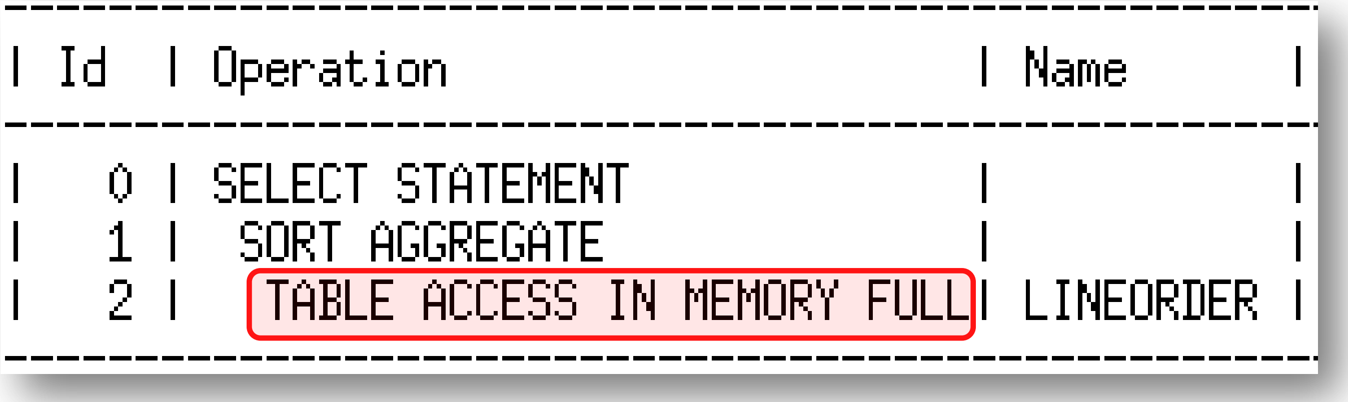
Example:
Find all sales in region of CA

> 100x Faster



- Each CPU core scans local in-memory columns
- Scans use super fast SIMD vector instructions
- **Billions of rows/sec** scan rate per CPU core
  - Row format is millions/sec

Identifying : INMEMORY Table Scan

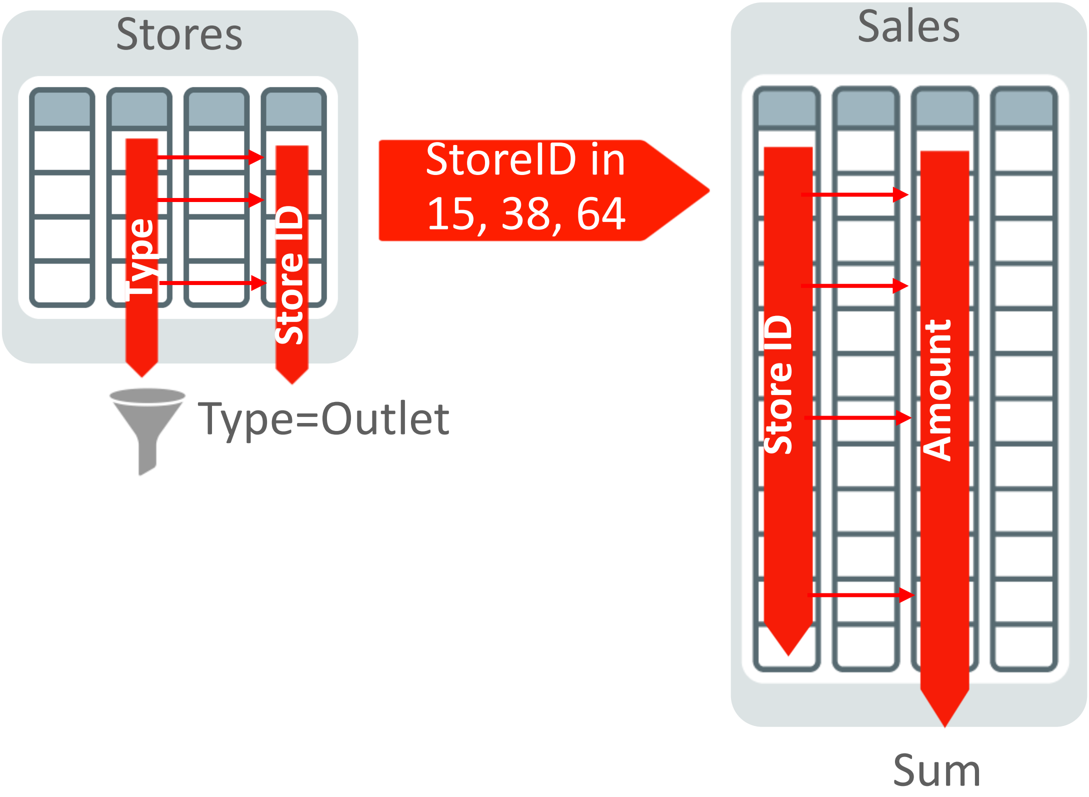


```
----------------------------------------------------
| Id  | Operation                     | Name      |
----------------------------------------------------
|   0 | SELECT STATEMENT              |           |
|   1 |  SORT AGGREGATE               |           |
|   2 |   TABLE ACCESS IN MEMORY FULL| LINEORDER |
----------------------------------------------------
```

- Optimizer fully aware
- Cost model adapted to consider INMEMORY scan
- New access method
  TABLE ACCESS IN MEMORY FULL
- Can be disabled via new parameter
  - INMEMORY_QUERY

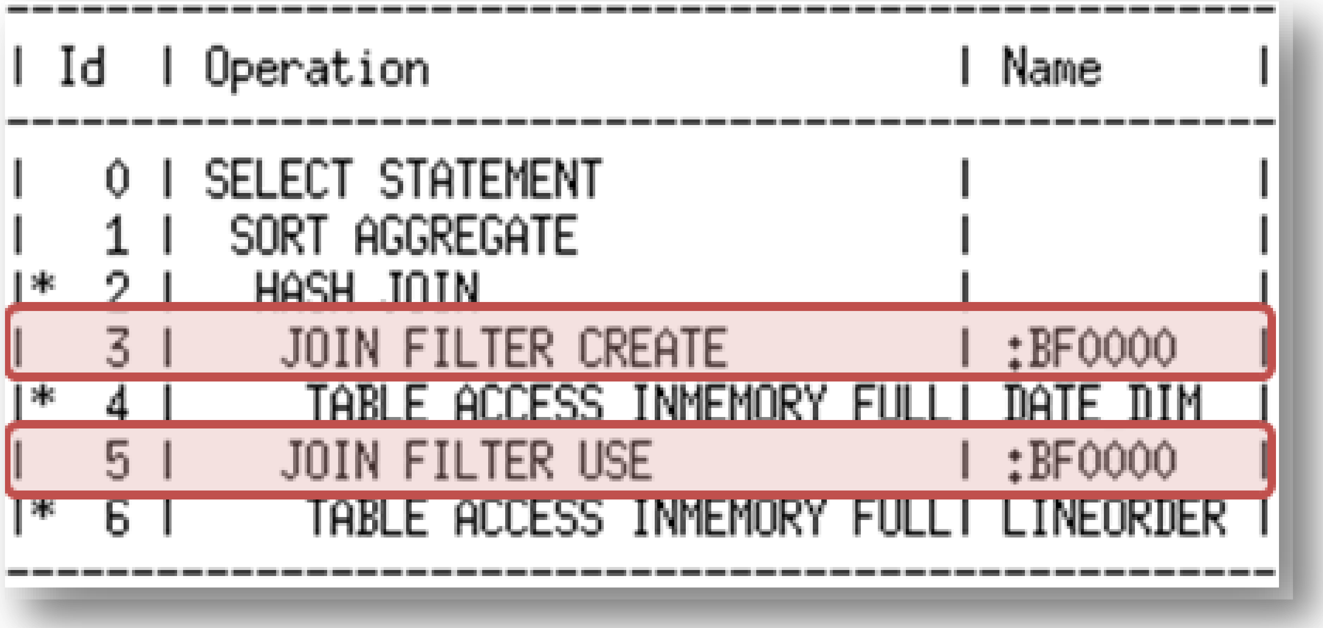
Joining and Combining Data Also Dramatically Faster

Example: Find total sales in outlet stores



- Converts joins of data in multiple tables into fast column scans
- Joins tables **10x** faster

Identifying : INMEMORY Joins

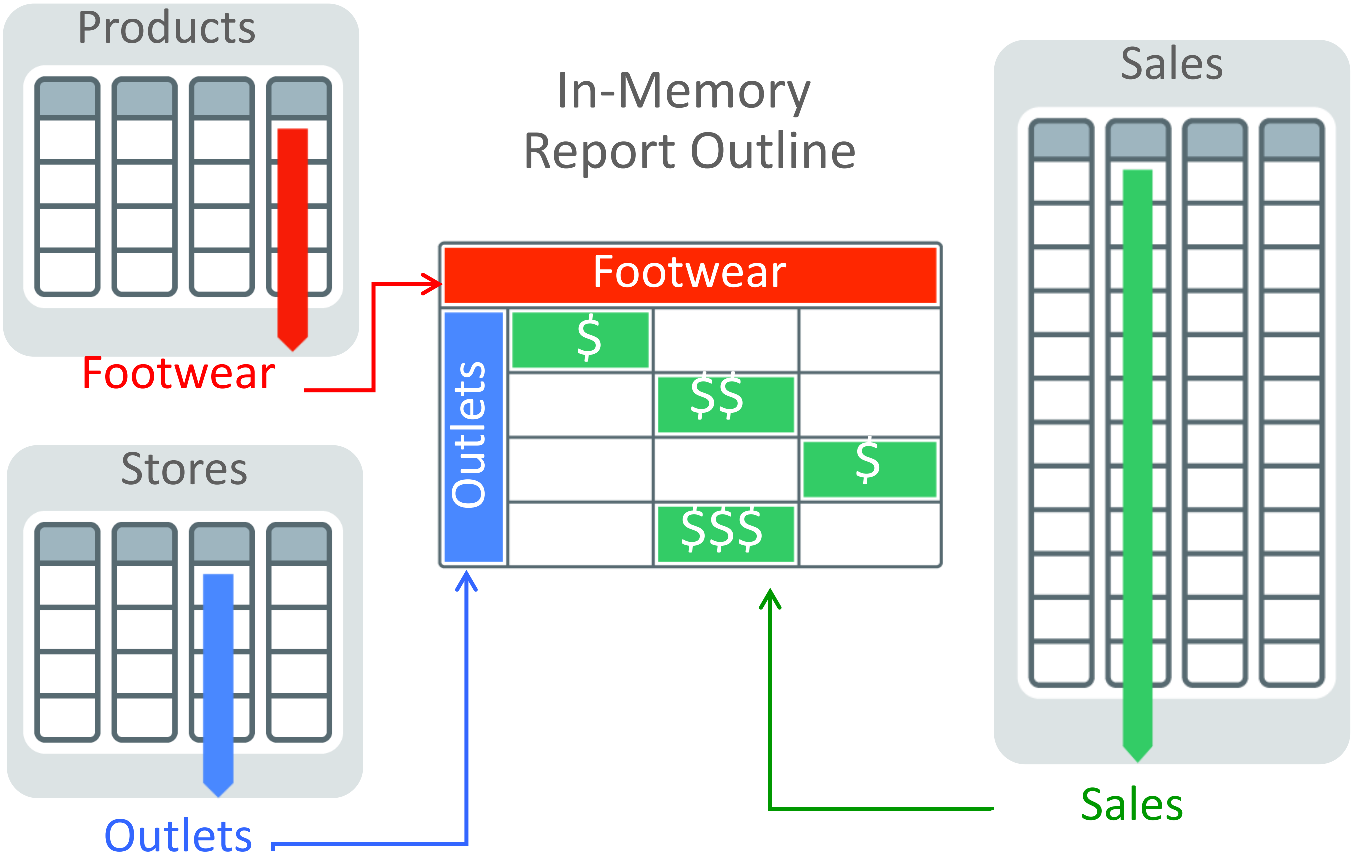


```
----------------------------------------------------------
| Id  | Operation                     | Name      |
----------------------------------------------------------
|   0 | SELECT STATEMENT              |           |
|   1 |  SORT AGGREGATE               |           |
|*  2 |   HASH JOIN                   |           |
|   3 |    JOIN FILTER CREATE         | :BF0000   |
|*  4 |     TABLE ACCESS INMEMORY FULL| DATE_DIM  |
|   5 |    JOIN FILTER USE            | :BF0000   |
|*  6 |     TABLE ACCESS INMEMORY FULL| LINEORDER |
----------------------------------------------------------
```

- Bloom filters enable joins to be converted into fast column scans
- Tried and true technology originally released in 10g
- Same technique used to offload joins on Exadata

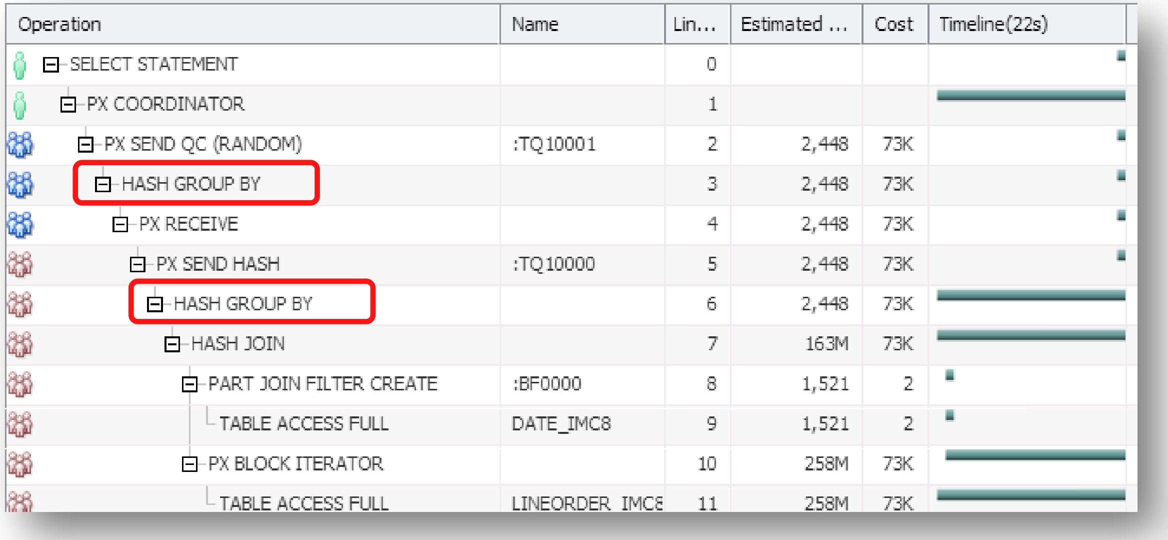
In-Memory Aggregation - Generates Reports Instantly

Example: Report sales of footwear in outlet stores



- Dynamically creates in-memory report outline
- Then report outline filled-in during fast fact scan
- Reports run much faster without predefined cubes

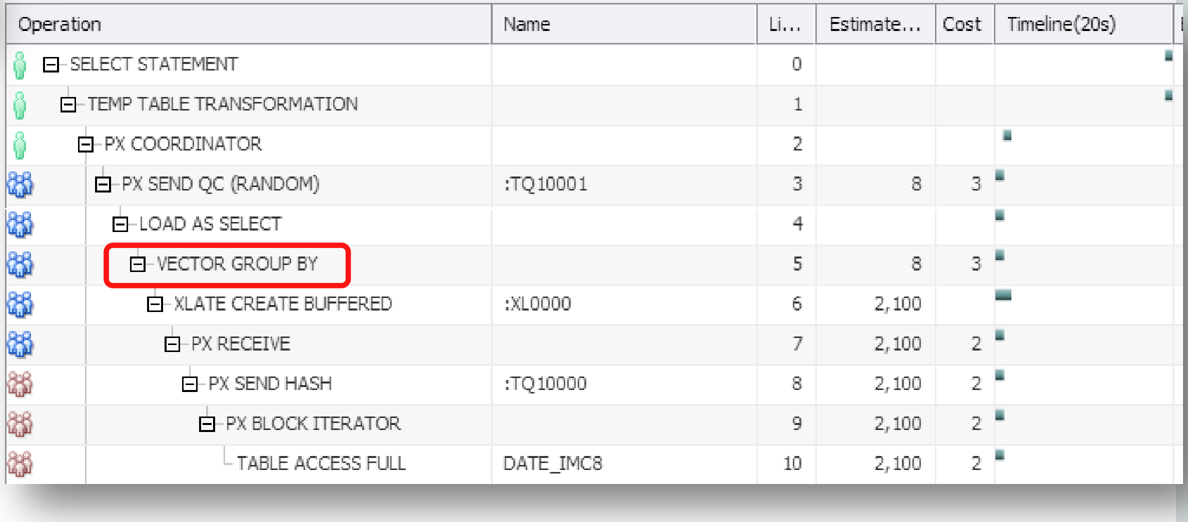
Identifying : INMEMORY Aggregation



Operation	Name	Lin...	Estimated ...	Cost	Timeline(22s)
SELECT STATEMENT		0			
PX COORDINATOR		1			
PX SEND QC (RANDOM)	:TQ10001	2	2,448	73K	
HASH GROUP BY		3	2,448	73K	
PX RECEIVE		4	2,448	73K	
PX SEND HASH	:TQ10000	5	2,448	73K	
HASH GROUP BY		6	2,448	73K	
HASH JOIN		7	163M	73K	
PART JOIN FILTER CREATE	:BF0000	8	1,521	2	
TABLE ACCESS FULL	DATE_IMC8	9	1,521	2	
PX BLOCK ITERATOR		10	258M	73K	
TABLE ACCESS FULL	LINEORDER_IMC8	11	258M	73K	

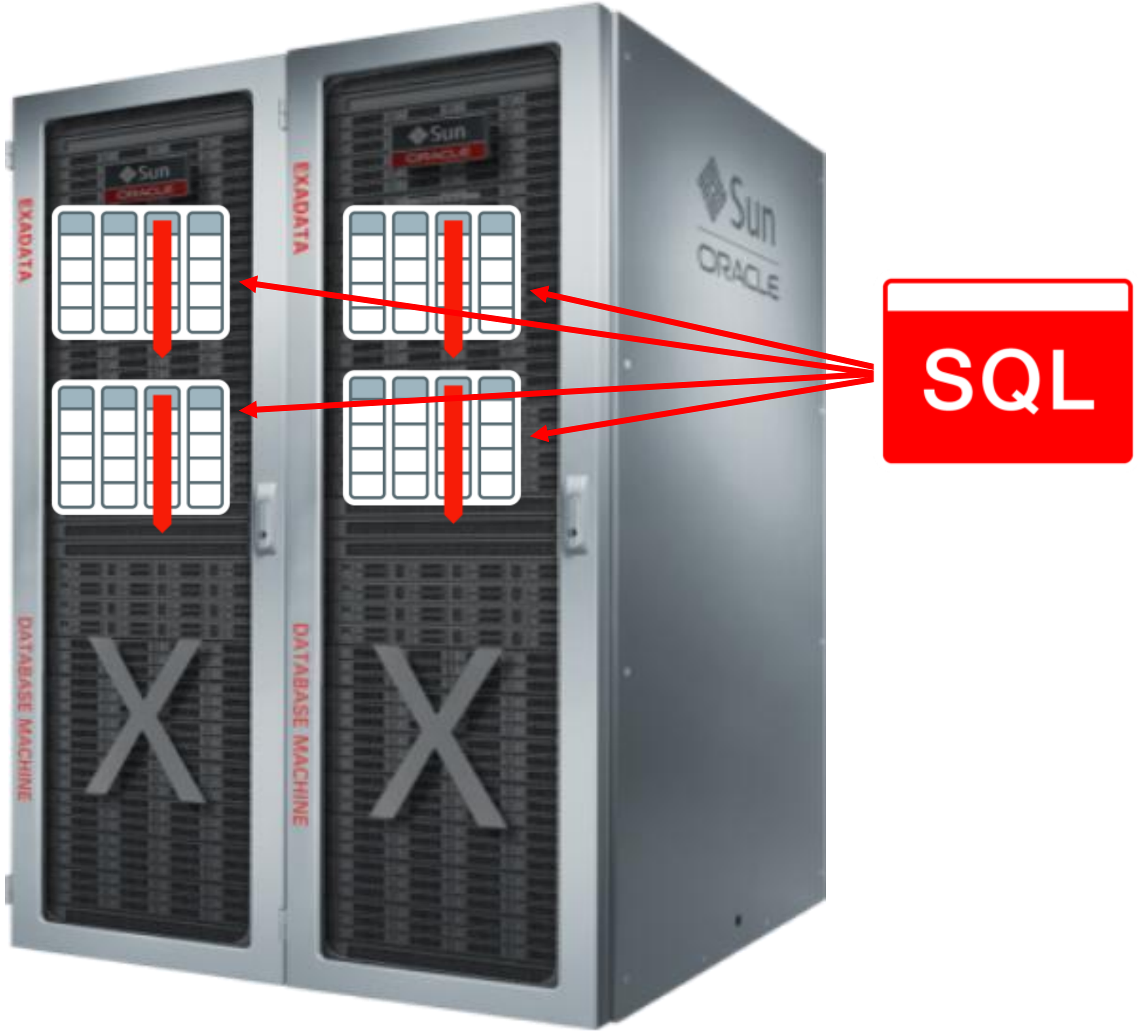
Hash Group By

Vector Group By



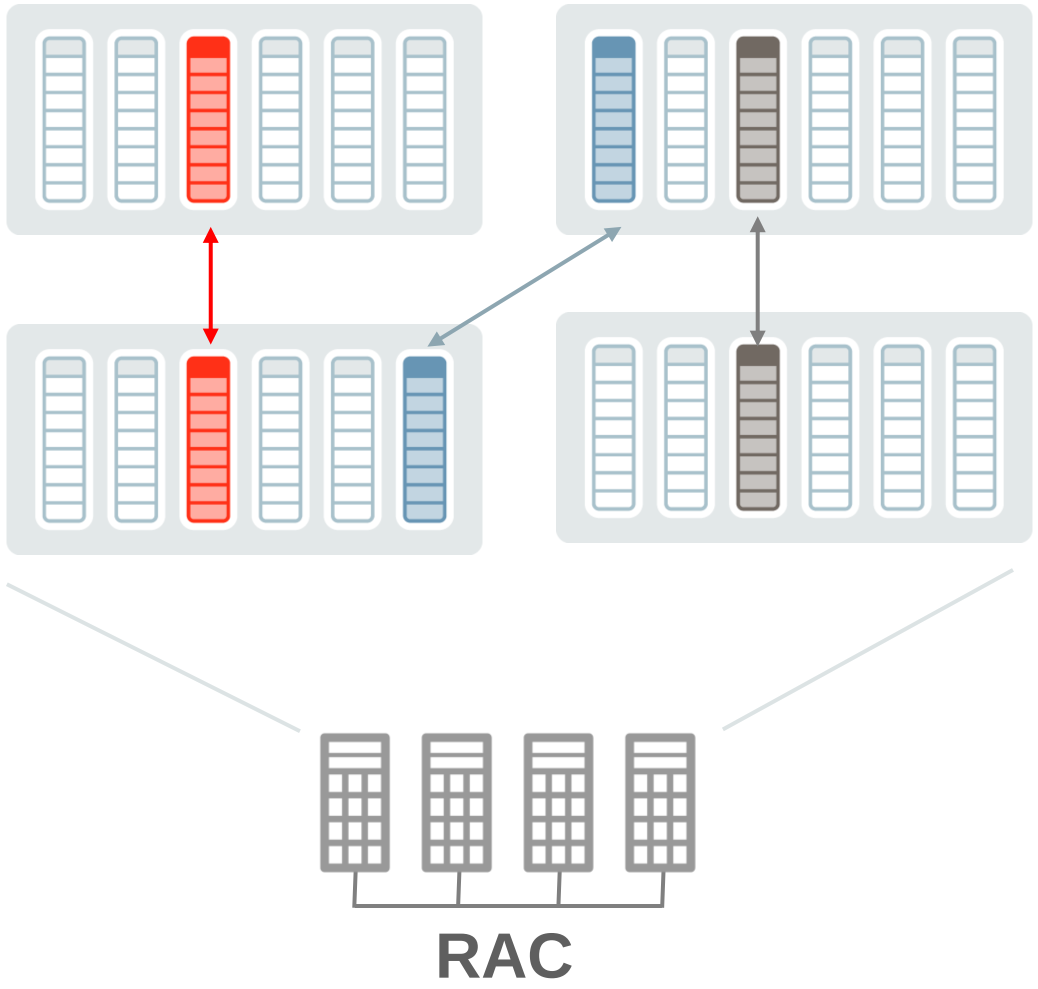
Operation	Name	Li...	Estimate...	Cost	Timeline(20s)
SELECT STATEMENT		0			
TEMP TABLE TRANSFORMATION		1			
PX COORDINATOR		2			
PX SEND QC (RANDOM)	:TQ10001	3	8	3	
LOAD AS SELECT		4			
VECTOR GROUP BY		5	8	3	
XLATE CREATE BUFFERED	:XL0000	6	2,100		
PX RECEIVE		7	2,100	2	
PX SEND HASH	:TQ10000	8	2,100	2	
PX BLOCK ITERATOR		9	2,100	2	
TABLE ACCESS FULL	DATE_IMC8	10	2,100	2	

Scale-Out In-Memory Database to Any Size



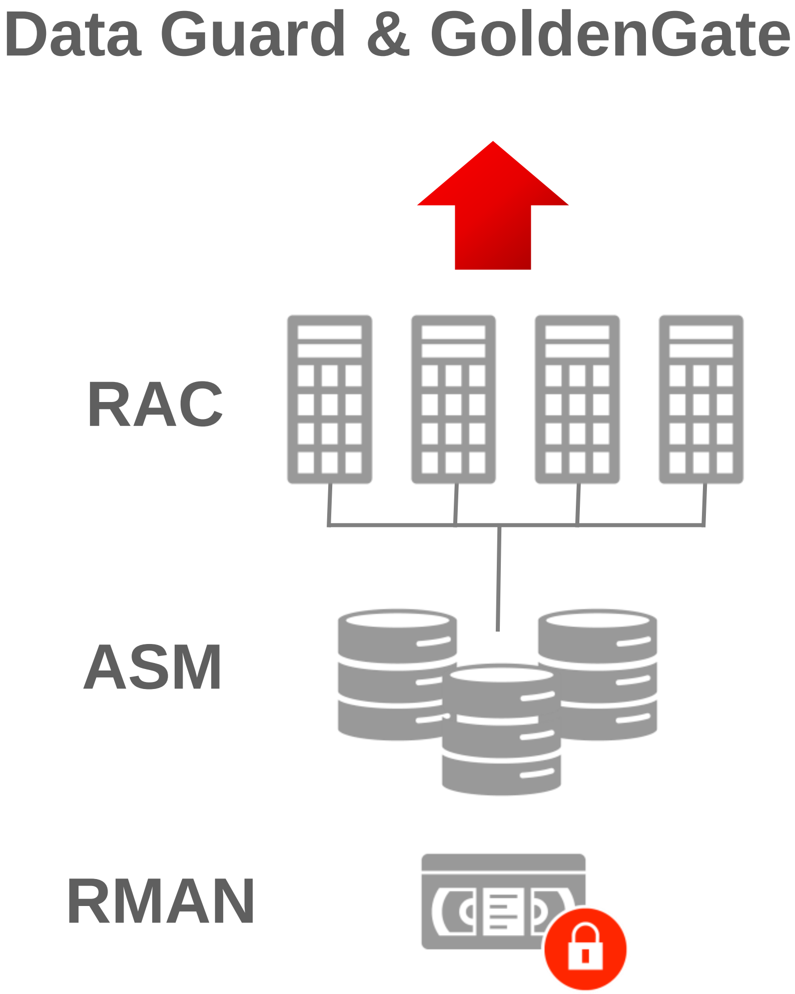
- Scale-Out across servers to grow memory and CPUs
- In-Memory **queries parallelized** across servers to access local column data
- Scale-out policy is defined at segment level (table, partition, sub partition) by DISTRIBUTE subclause
  - Distribute by rowid range
  - Distribute by partition
  - Distribute AUTO

Unique Fault Tolerance



- Similar to storage mirroring
- Duplicate in-memory columns on another or all nodes
  - Enabled per table/partition
  - Application transparent
- Downtime eliminated by using duplicate after failure
- **When you deploy Oracle RAC on a non-engineered system, the DUPLICATE clause is ignored**

Oracle In-Memory: Industrial Strength Availability



- Pure In-Memory format does not change Oracle's storage format, logging, backup, recovery, etc.
- All Oracle's proven availability technologies work transparently
- Protection from all failures
  - Node, site, corruption, human error, etc.

Oracle In-Memory Requires Zero Application Changes

Full Functionality	- No restrictions on SQL
Easy to Implement	- No migration of data
Fully Compatible	- All existing applications run unchanged
Fully Multitenant	- Oracle In-Memory is Cloud Ready

ORACLE E-BUSINESS SUITE

ORACLE FUSION APPLICATIONS

ORACLE JD EDWARDS

ORACLE PEOPLESOFT

ORACLE SIEBEL

Uniquely Achieves All In-Memory Benefits With No Application Changes

Demo

Hardware and Software

Engineered to Work Together

ORACLE®