Infrastructure at your Service.

In-Memory-Pläne für den 12.2-Optimizer: Teuer oder billig?

About me

Clemens Bleile

Senior Consultant

Oracle Certified Professional DB 11g, 12c
Oracle Certified Expert DB 12c Performance Management and Tuning

+41 78 677 51 09

clemens.bleile@dbi-services.com

Who we are
dbi services

Experts At Your Service
- 50 specialists in IT infrastructure
- Certified, experienced, passionate

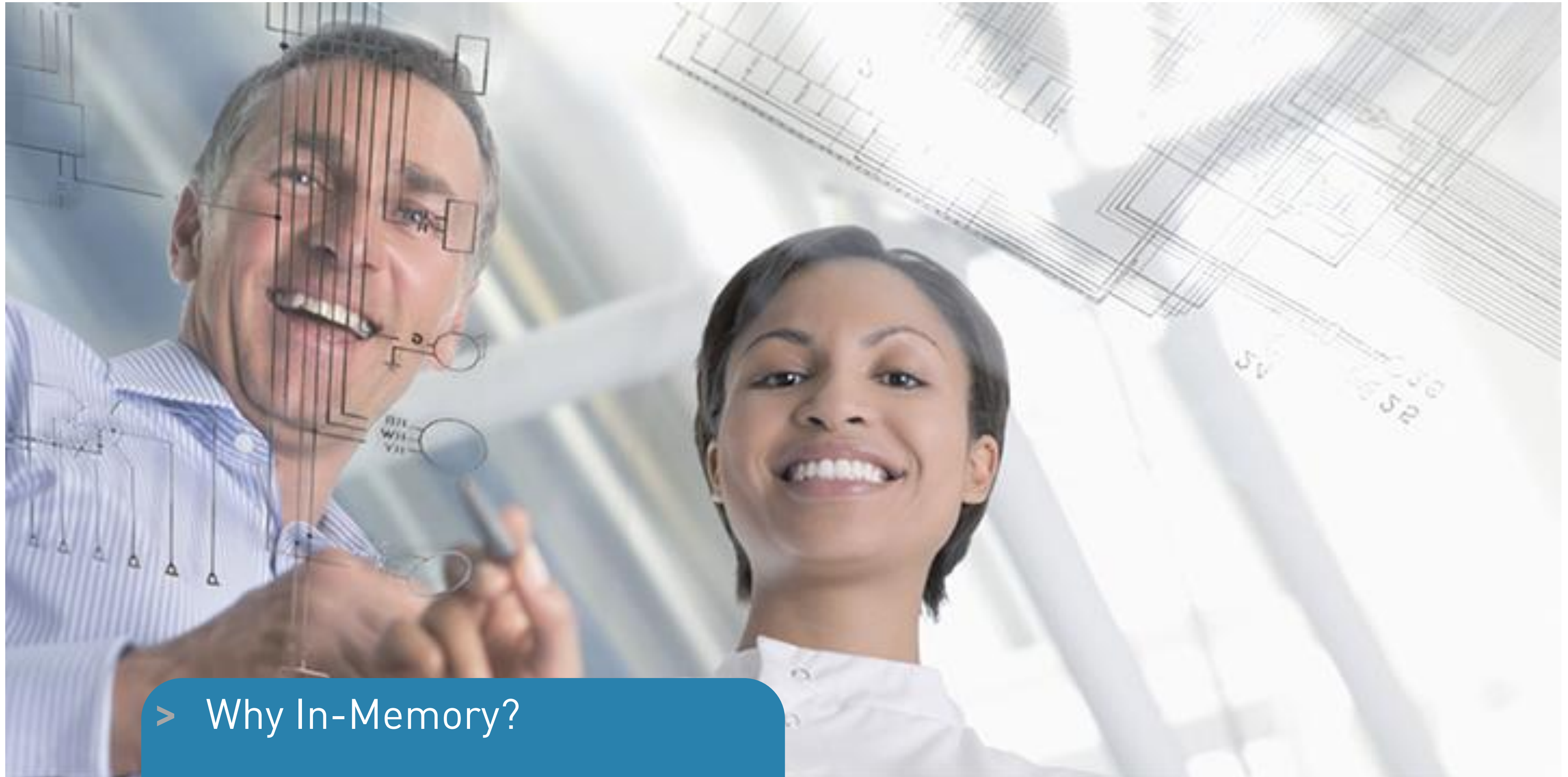
Based In Switzerland
- 100% self-financed Swiss company
- Over CHF6 mio. turnover

Leading In Infrastructure Services
- More than 120 customers in CH, D, & F
- Over 40 SLAs dbi FlexService contracted

Agenda

1. Oracle Database In-Memory
2. In-Memory and the Optimizer
3. New In-Memory features in 12cR2 related to the Optimizer
4. Test results with HammerDB TPC-H

Oracle Database In-Memory



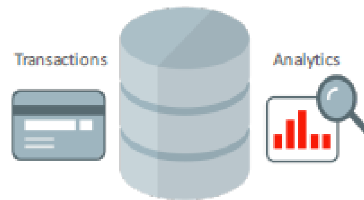
- Why In-Memory?
- In-Memory Goals
- Architecture

What is Database In-Memory?
In-Memory Goals?

Real-Time Analytics

Enable Real-Time
Business Decisions

Accelerate Mixed Workload

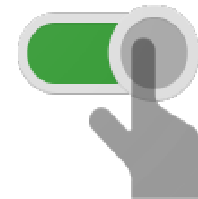


Run analytics on
Operational Systems

Risk-Free

Proven Scale-Out,
Availability, Security

Trivial to Implement

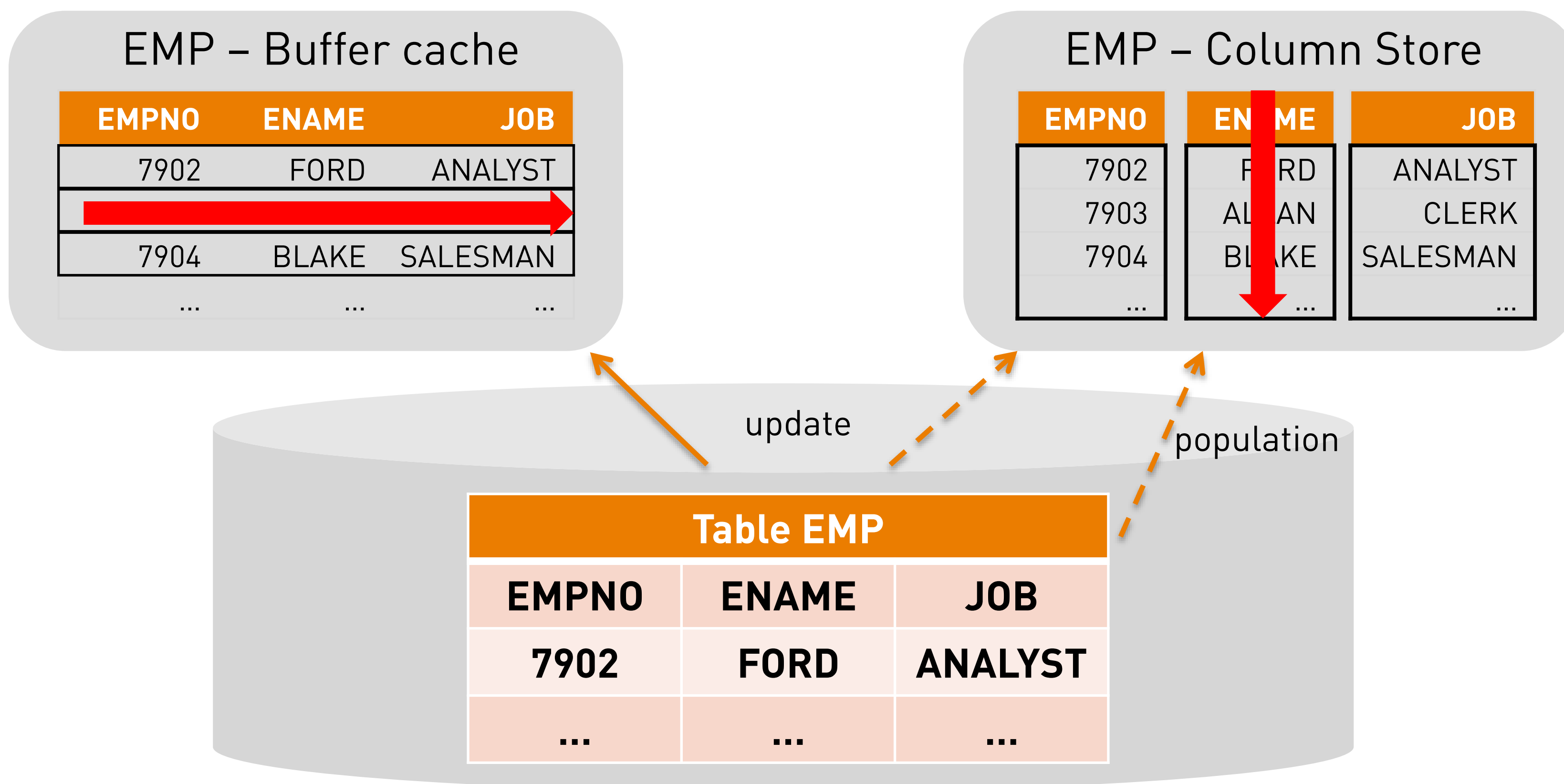


No Application Changes
Not Limited by Memory

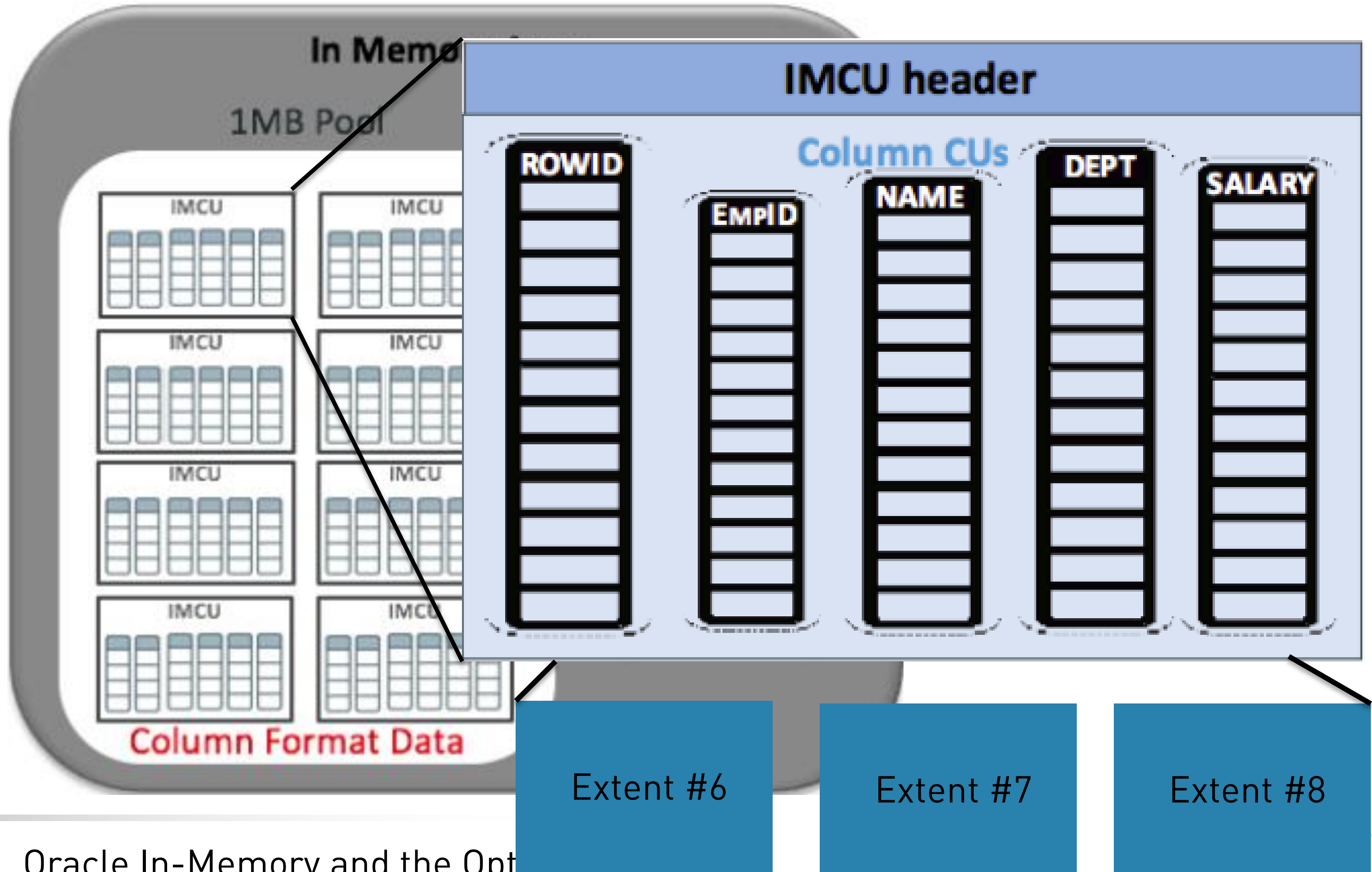
What is Database In-Memory?
Architecture

Oracle 12.1.0.2 introduced new columnar format

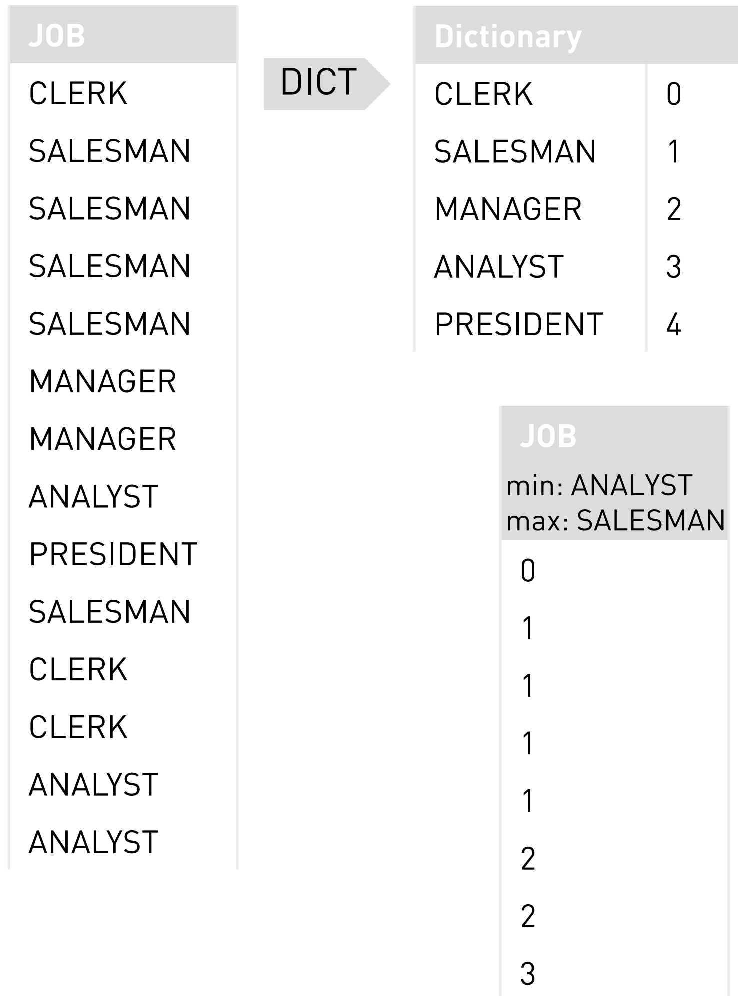
- New format does NOT replace existing row format
- Standard memory pools keeps row format and Oracle block size structure



What is Database In-Memory?
Architecture



What is Database In-Memory?
Architecture



Column Compression Unit (CU)

- Contiguous storage per column in an IMCU
- All CUs automatically store Min/Max values
- Multiple formats: depends on data and chosen compression level
- Most CUs have a "Dictionary"
- – Sorted list of distinct values in the CU
- – Column values replaced with dictionary IDs
- Compression Units are indexed (like Exadata Storage Index)

What is Database In-Memory?
Architecture – Query just what's needed

Access only needed columns

- In Buffer Cache all columns are scanned

EMP – Buffer Cache

EMPNO	ENAME	JOB
7902	FORD	ANALYST
7903	ALLAN	CLERK
7904	BLAKE	SALESMAN
...	...	...

EMP – Column Store

EMPNO	ENAME	JOB
7902	FORD	ANALYST
7903	ALLAN	CLERK
7904	BLAKE	SALESMAN
...	...	...

Scan and filter compressed data

- Data is populated in compressed format
- Compressed format allows WHERE-clause predicates to be evaluated against the compressed data (except for compression CAPACITY LOW/HIGH)

→ SCAN LESS BYTES THAN IN BUFFER CACHE

What is Database In-Memory?
Architecture – Query just what's needed

Storage Indexes

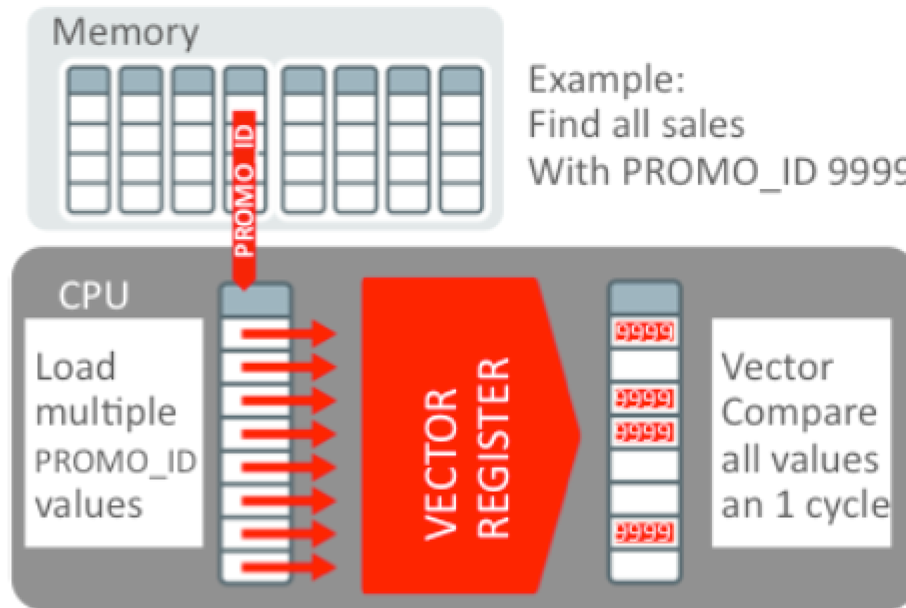
- Similar to Exadata Storage Indexes
- They are automatically maintained for each column in the Column Store
- Store the Min/Max values of each CU
- Purpose is to avoid accessing the IMCUs that do not contain relevant data
- Pruning happens at 2 levels:
  - predicate outside min/max → prune storage index
  - equally predicate > min and < max → prune if dictionary compressed and value is not in the list of distinct dictionary values

Demo

What is Database In-Memory?
Architecture

SIMD (Single Instruction processing Multiple Data values)

- Evaluate a set of data values in a single CPU instruction
- Scan rows in the range of billions/sec per core instead of millions/sec per core (buffer cache)



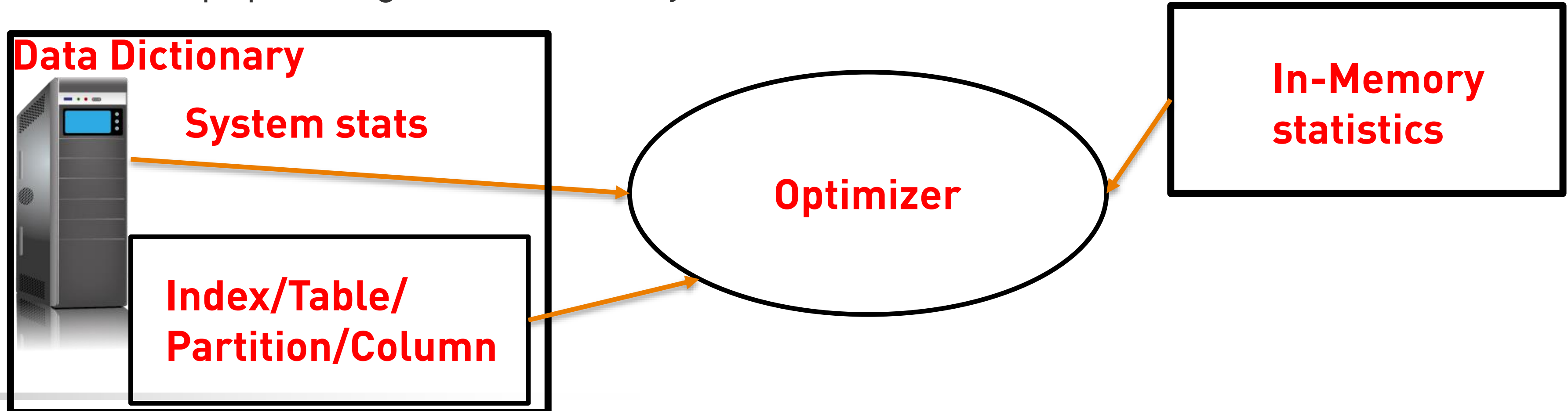
In-Memory and the Optimizer

- What does the optimizer know about IM?
- The 10053 Trace
- In-Memory Joins

In-Memory and the Optimizer
What does the optimizer know about IM?

Cost model has been expanded to consider in-memory operations

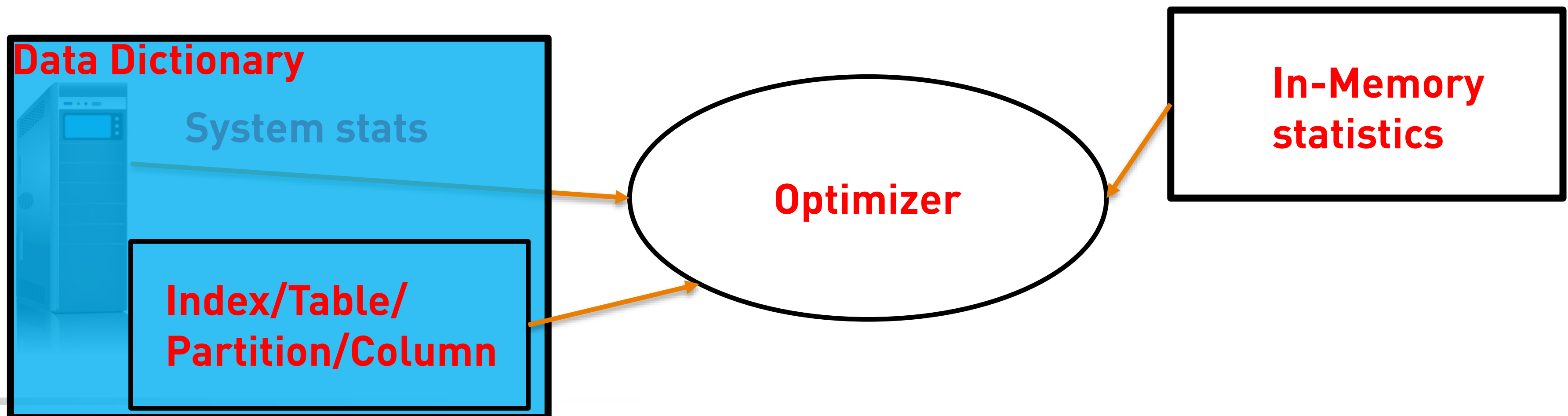
- Still considers statistics gathered with dbms_stats
  - Tables, columns, indexes, partitions
- Still considers system statistics
  - CPU-speed, IO-throughput for single and multiblock-reads
- In-memory specific statistics
  - Stats maintained about in-memory objects since starting populating the in-memory column store



In-Memory and the Optimizer
What does the optimizer know about IM?

The basic rules about statistics gathering still apply with IM:

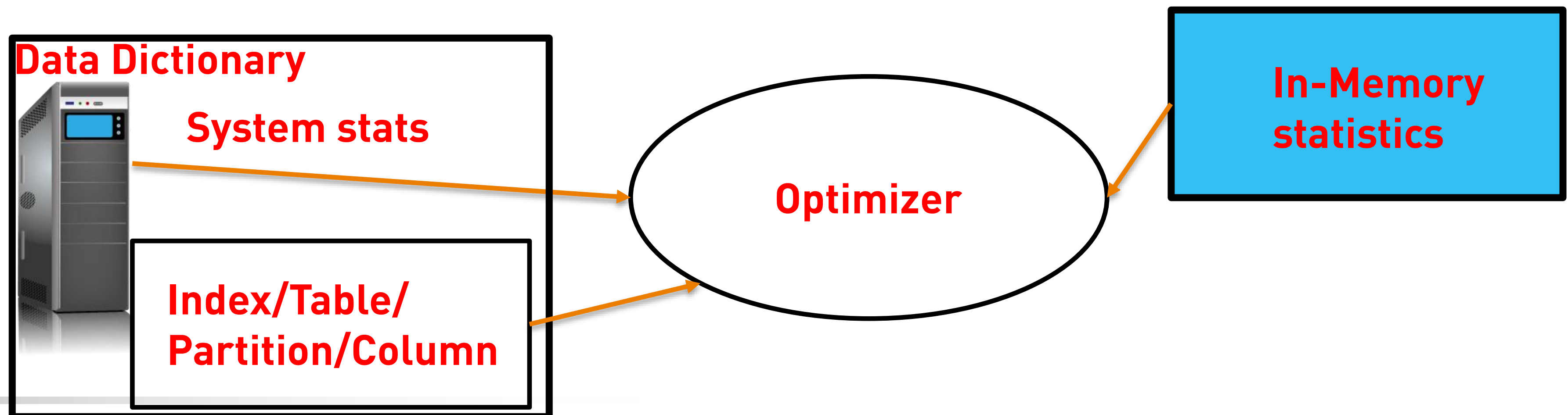
- Use DBMS_STATS.GATHER_..._STATS to gather statistics
- User Histograms in case of data skew
- Use extended statistics so that the optimizer knows about correlations: single table cardinality, joins, aggregations
- Use constraints (e.g. PK/FK, Not Null)



In-Memory and the Optimizer
What does the optimizer know about IM?

New In-Memory statistics

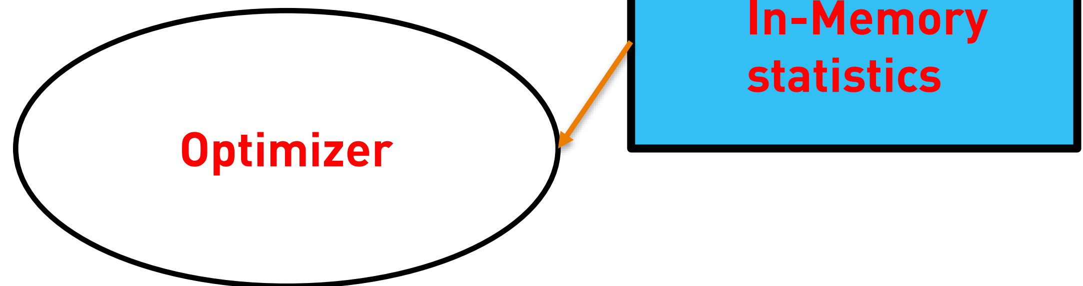
- Automatically computed during hard parse
  - In Memory Compression Units
  - IM Rows
  - Transaction Journal Rows
  - IM Blocks
  - IM Quotient
    - How much has been populated: 0-1



In-Memory and the Optimizer
What does the optimizer know about IM?

IM Costs

- IO costs
  - Read not populated/read consistent (invalid) rows
  - Extent Map
- CPU costs
  - Traverse IMCUs
  - Check storage indexes (pruning)
  - Decompressing
  - Predicate evaluation
  - Row stitch
  - Scanning the journal



In-Memory and the Optimizer
The 10053 trace

```
alter session set events '10053 trace name context forever, level 1';
alter session set events 'trace [SQL_optimizer.*]';
```

```
select /*+ FULL(t) */ count(*)
from part t where P_PARTKEY < 70000;
```

In-Memory and the Optimizer

The 10053 trace

```
select /*+ FULL(t) */ count(*)
from part t where P_PARTKEY < 70000;
```

```
***************************************
BASE STATISTICAL INFORMATION
***********************
Table Stats::
  Table: PART  Alias:  T
  #Rows: 2000000  SSZ: 0  LGR: 0  #Blks:  38657  AvgRowLen:
  #IMCUs: 4  IMCRowCnt: 2000000  IMCJournalRowCnt: 50000
  #IMCBlocks: 38657  IMCQuotient: 1.000000
Index Stats::
```

In-Memory and the Optimizer
The 10053 trace

```
select /*+ FULL(t) */ count(*)
from part t where P_PARTKEY < 70000;
```

```
Column (#1): P_PARTKEY(NUMBER)
  AvgLen: 6 NDV: 2000000 Nulls: 0 Density: 0.000000 Min: 1.0
Estimated selectivity: 0.035000 , col: #1
Table: PART  Alias: T
  Card: Original: 2000000.000000  Rounded: 69999  Computed:
Scan IO  Cost (Disk) =   0.000000
Scan CPU Cost (Disk) =   0.000000
Scan IO  Cost (IMC)  =   386.570000 (read invalid rows)
                       + 2.000000 (read extent map)
                     =   388.570000
Scan CPU Cost (IMC)  =   45000.000000 (overhead)
```

In-Memory and the Optimizer
The 10053 trace

```
Scan CPU Cost (IMC)  =   45000.000000 (overhead)
                       + 40000.000000 (traverse IMCU)
                       + 800.000000 (min-max eval) (= 200.000000 (min-max eval per cu) * 4 (#min-max CUs))
                       + 0.000000 (decompression) (= 0.000000 (per IMCU) * 4 (#IMCUs) * 0.500000 (prune ratio))
                       + 10000000.000000 (pred eval) (= 10.000000 (per row) * 2000000 (#IMCrows)
                         * 0.500000 (prune ratio))
                       + 100000000.000000 (row stitch) (= 100.000000 (per col) * 1 (#cols) * 2000000 (#IMCRows)
                         * 0.500000 (prune ratio))
                       + 90000000.000000 (scan journal) (= 1800.000000 (per row) * 50000 (#journal rows))
                     =   200085800.000000
Cost of predicates:
```

```
Total Scan CPU  Cost =   0.000000 (scan (Disk))
                       + 200085800.000000 (scan (IMC))
                       + 100000000.000000 (cpu filter eval)
                         (= 50.000000 (per row) * 2000000.000000 (#rows))
                     =   300085800.000000
Access Path: TableScan
```

```
                   =   300085800.000000
Access Path: TableScan
  Cost:  403.112071  Resp: 403.112071  Degree: 0
    Cost_io: 388.570000  Cost_cpu: 300085800
    Resp_io: 388.570000  Resp_cpu: 300085800
Best:: AccessPath: TableScan
```

In-Memory and the Optimizer
The 10053 trace

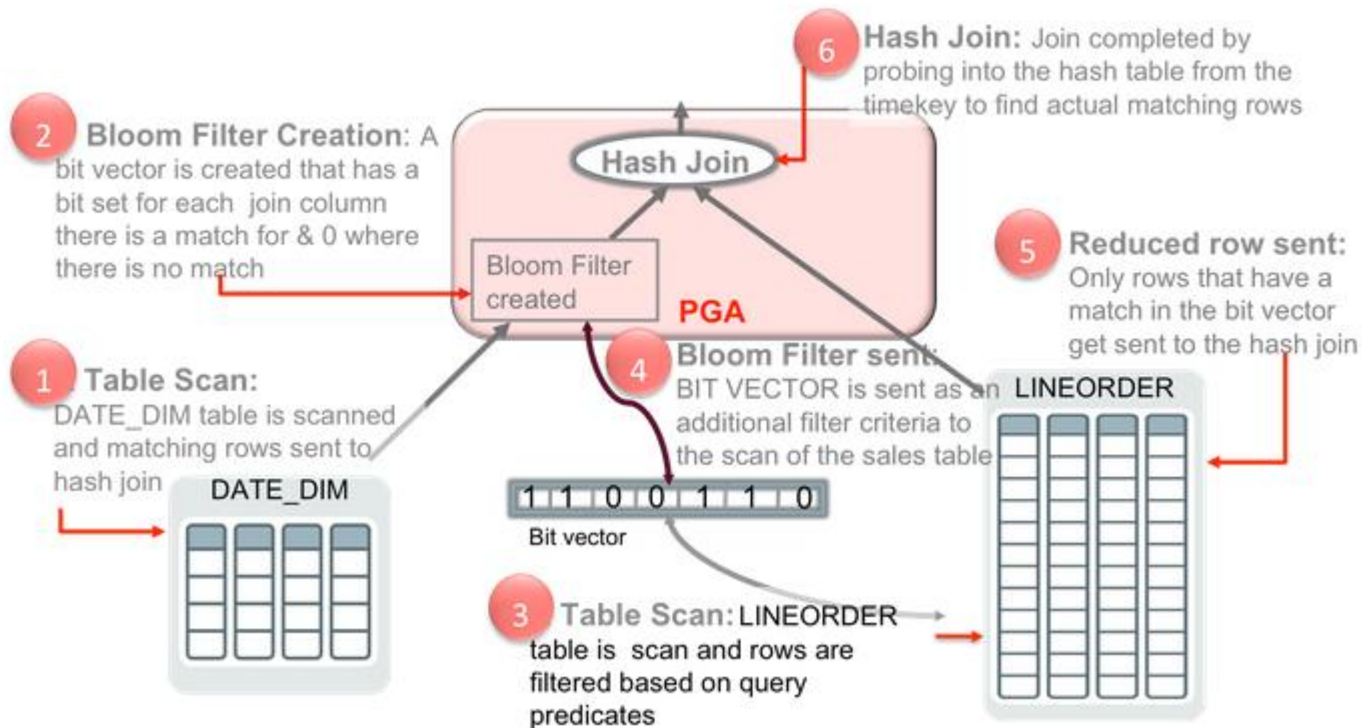
```
select /*+ FULL(t) */ P_PARTKEY,P_TYPE,P_SIZE,P_BRAND
from part t where P_PARTKEY < 70000;
```

```
Scan CPU Cost (IMC)  =   45000.000000 (overhead)
                       + 40000.000000 (traverse IMCU)
                       + 800.000000 (min-max eval) (= 200.000000 (min-max eval per cu)
                       + 0.000000 (decompression) (= 0.000000 (per IMCU) * 4 (#IMCUs)
                       + 10000000.000000 (pred eval) (= 10.000000 (per row) * 2000000
                       + 400000000.000000 (row stitch) (= 100.000000 (per col)
                         * 4 (#cols) * 2000000 (#IMCRows) * 0.500000 (prune ratio))
                       + 90000000.000000 (scan journal) (= 1800.000000 (per row) * 500
                     =   500085800.000000
Cost of predicates:
```

```
Access Path: TableScan
   Cost:   417.649985  Resp: 417.649985  Degree: 0
     Cost_io: 388.570000  Cost_cpu: 600085800
     Resp_io: 388.570000  Resp_cpu: 600085800
```

In-Memory and the Optimizer
In-Memory Joins

Joining In-Memory with Bloom Filters (12cR1)



In-Memory and the Optimizer

In-Memory Joins

```
------------------------------------------------------------------------------------------
| Id  | Operation                        | Name      | Starts | E-Rows | A-Rows |   A-Time   |
------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                 |           |      1 |        |      2 |00:00:00.90 |
|   1 |  SORT GROUP BY                   |           |      1 |      2 |      2 |00:00:00.90 |
|*  2 |   HASH JOIN                      |           |      1 |   474K|   311K|00:00:00.81 |
|   3 |    JOIN FILTER CREATE            | :BF0000   |      1 |   474K|   311K|00:00:00.44 |
|*  4 |     TABLE ACCESS INMEMORY FULL| LINEITEM |      1 |   474K|   311K|00:00:00.39 |
|   5 |    JOIN FILTER USE               | :BF0000   |      1 |    15M|  1282K|00:00:00.24 |
|*  6 |     TABLE ACCESS INMEMORY FULL| ORDERS    |      1 |    15M|  1282K|00:00:00.24 |
------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("O_ORDERKEY"="L_ORDERKEY")
   4 - inmemory(("L_RECEIPTDATE">=TO_DATE(' 1997-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss')
              "L_SHIPDATE"<"L_COMMITDATE" AND "L_COMMITDATE"<"L_RECEIPTDATE" AND "L_RECEIP
              'syyyy-mm-dd hh24:mi:ss') AND "L_SHIPDATE"<TO_DATE(' 1998-01-01 00:00:00', ':
              "L_COMMITDATE"<TO_DATE(' 1998-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
       filter(("L_RECEIPTDATE">=TO_DATE(' 1997-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') A
              "L_SHIPDATE"<"L_COMMITDATE" AND "L_COMMITDATE"<"L_RECEIPTDATE" AND "L_RECEIP
              'syyyy-mm-dd hh24:mi:ss') AND "L_SHIPDATE"<TO_DATE(' 1998-01-01 00:00:00', ':
              "L_COMMITDATE"<TO_DATE(' 1998-01-01 00:00:00', 'syyyy-mm-dd hh24:mi:ss') AND
   6 - inmemory(SYS_OP_BLOOM_FILTER(:BF0000,"O_ORDERKEY"))
       filter(SYS_OP_BLOOM_FILTER(:BF0000,"O_ORDERKEY"))
```

In-Memory and the Optimizer
In-Memory Joins

```
-------------------------------------------------------------------------------------------------
| Id  | Operation                           | Name         | Starts | E-Rows | A-Rows |   A-Time   |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |              |      1 |        |   114K|00:00:34.79 |
|   1 |  SORT ORDER BY                      |              |      1 |     74 |   114K|00:00:34.79 |
|*  2 |   HASH JOIN                         |              |      1 |     74 |   114K|00:00:34.65 |
|   3 |    JOIN FILTER CREATE               | :BF0000      |      1 |    256 |   567K|00:00:34.11 |
|*  4 |     HASH JOIN                       |              |      1 |    256 |   567K|00:00:33.83 |
|   5 |      VIEW                           | VW_GBC_10    |      1 |    256 |  8245K|00:00:11.82 |
|   6 |       HASH GROUP BY                 |              |      1 |    256 |  8245K|00:00:08.68 |
|*  7 |        TABLE ACCESS INMEMORY FULL   | LINEITEM     |      1 |    32M|  8315K|00:00:02.14 |
|*  8 |      TABLE ACCESS INMEMORY FULL     | ORDERS       |      1 |      1 |  7321K|00:00:01.86 |
|   9 |    JOIN FILTER USE                  | :BF0000      |      1 |   299K|   189K|00:00:00.03 |
|* 10 |     TABLE ACCESS INMEMORY FULL      | CUSTOMER     |      1 |   299K|   189K|00:00:00.03 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("C_CUSTKEY"="O_CUSTKEY")
   4 - access("ITEM_1"="O_ORDERKEY")
   7 - inmemory("L_SHIPDATE">TO_DATE(' 1995-03-19 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
       filter("L_SHIPDATE">TO_DATE(' 1995-03-19 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
   8 - inmemory("O_ORDERDATE"<TO_DATE(' 1995-03-19 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
       filter("O_ORDERDATE"<TO_DATE(' 1995-03-19 00:00:00', 'syyyy-mm-dd hh24:mi:ss'))
  10 - inmemory(("C_MKTSEGMENT"='FURNITURE' AND SYS_OP_BLOOM_FILTER(:BF0000,"C_CUSTKEY")))
       filter(("C_MKTSEGMENT"='FURNITURE' AND SYS_OP_BLOOM_FILTER(:BF0000,"C_CUSTKEY")))

Note
-----
   - statistics feedback used for this statement
   - this is an adaptive plan
```

In-Memory and the Optimizer

In-Memory Joins

```
-----------------------------------------------------------------------------------------------------
|  Id  | Operation                        | Name        | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------------
|    0 | SELECT STATEMENT                 |             |       |       | 15719 (100)|          |
|    1 |  SORT ORDER BY                   |             |    74 |  4292 | 15719  (27)| 00:00:01 |
|  * 2 |   HASH JOIN                      |             |    74 |  4292 | 15718  (27)| 00:00:01 |
|    3 |    JOIN FILTER CREATE            | :BF0000     |   256 | 10496 | 15349  (28)| 00:00:01 |
|  * 4 |     HASH JOIN                    |             |   256 | 10496 | 15349  (28)| 00:00:01 |
|-   5 |      NESTED LOOPS                |             |   256 | 10496 | 15349  (28)| 00:00:01 |
|-   6 |       NESTED LOOPS               |             |   256 | 10496 | 15349  (28)| 00:00:01 |
|-   7 |        STATISTICS COLLECTOR      |             |       |       |            |          |
|    8 |         VIEW                     | VW_GBC_10   |   256 |  4864 | 15092  (28)| 00:00:01 |
|    9 |          HASH GROUP BY           |             |   256 |  5376 | 15092  (28)| 00:00:01 |
|  * 10 |           TABLE ACCESS INMEMORY FULL| LINEITEM  |   32M |  643M | 13341  (19)| 00:00:01 |
|- * 11 |        INDEX UNIQUE SCAN         | ORDERS_PK   |     1 |       |     1   (0)| 00:00:01 |
|- * 12 |       TABLE ACCESS BY INDEX ROWID | ORDERS     |     1 |    22 |     2   (0)| 00:00:01 |
|  * 13 |      TABLE ACCESS INMEMORY FULL  | ORDERS      |     1 |    22 |     2   (0)| 00:00:01 |
|    14 |    JOIN FILTER USE               | :BF0000     |  299K| 4973K|   368  (14)| 00:00:01 |
|  * 15 |     TABLE ACCESS INMEMORY FULL   | CUSTOMER    |  299K| 4973K|   368  (14)| 00:00:01 |
-----------------------------------------------------------------------------------------------------
```

New In-Memory features in 12cR2

- In-Memory Expressions
- Join-Groups

New In-Memory features in 12cR2
In-Memory Expressions

What are In-Memory Expressions?

- Frequently computed expressions are "pre-computed" as In-Memory expressions (IME)
- IMEs can be created for
  - Virtual columns
  - Automatic capture
    - Frequently evaluated query expressions
    - Other internal computations (hash join values, predicate evaluations, data conversions)
- Repeated expression evaluation can be computationally expensive
- Very useful for large datasets
- Still supports other In-Memory optimizations (min/max pruning, SIMD, etc.)

New In-Memory features in 12cR2
In-Memory Expressions

Example of In-Memory Expressions

```
select l_returnflag, l_linestatus, sum(l_quantity) as sum_qty,
sum(l_extendedprice) as sum_base_price,
sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price,
avg(l_discount) as avg_disc, count(*) as count_order
from lineitem
where l_shipdate >= to_date ('2015-12-01','YYYY-MM-DD') - 90
group by l_returnflag, l_linestatus
order by l_returnflag, l_linestatus;
```

New In-Memory features in 12cR2
In-Memory Expressions

How do In-Memory Expressions work?

- In-Memory columns consisting of
  - User defined virtual columns

```
create table T1 (a number, b number, v as (a+b));
```

  - Automatic capture
    - Frequently evaluated (hot) query expressions

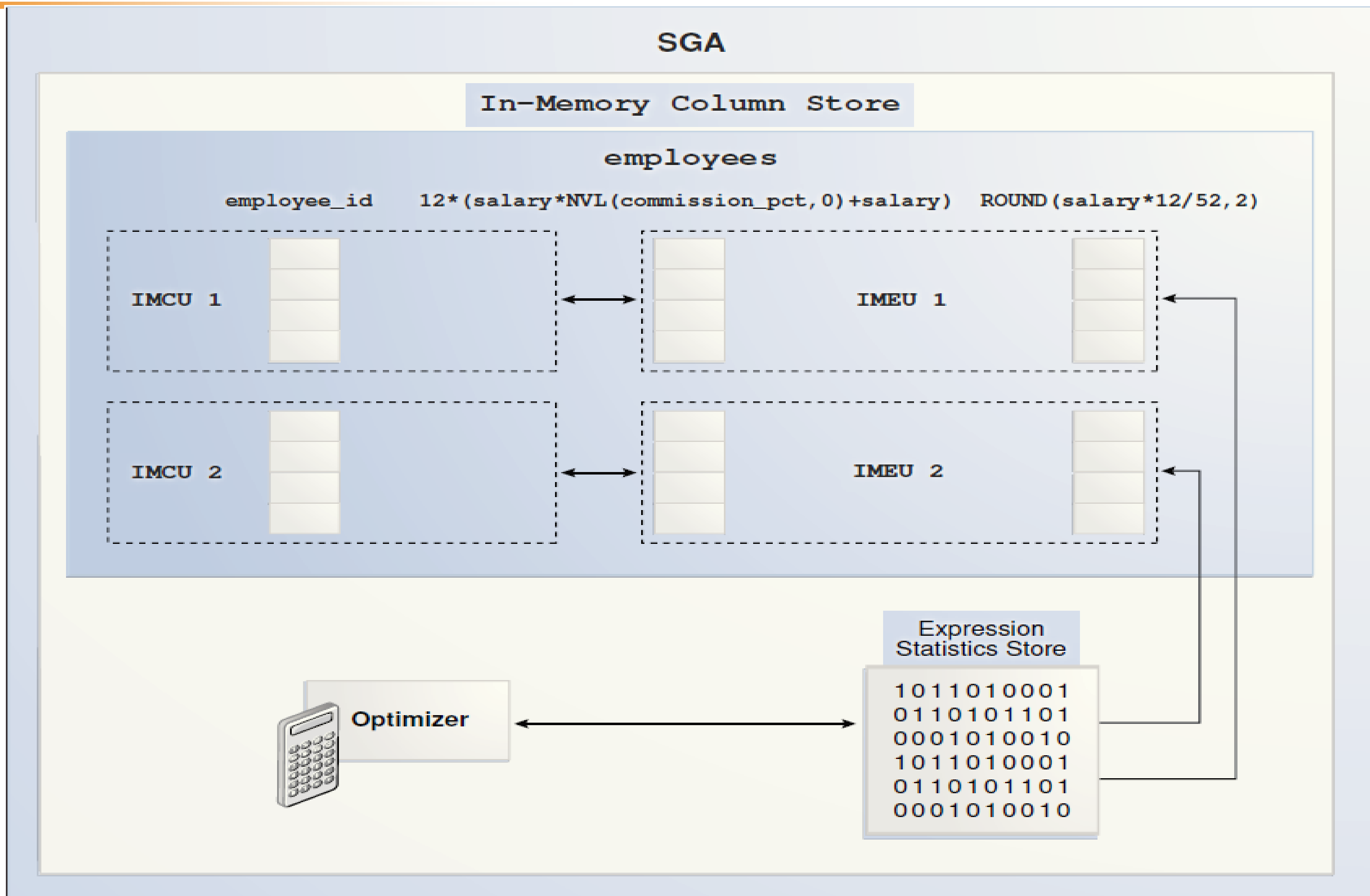
```
select a*b from T1 where a/b = 1;
```

→ One or more columns of a single row of a table, possibly some constants

→ Have a 1-to-1 mapping with the rows in the table

    - Other useful internal computations (e.g. hash join values)
  - Oracle JSON
    - Tree-based binary encoding of JSON

New In-Memory features in 12cR2
The IM Expression Statistics Store (ESS)



New In-Memory features in 12cR2
In-Memory Expressions

In-Memory Expressions

- Expressions Statistics Store (ESS)
  - Is maintained by the Optimizer layer
  - Identifies query expressions during hard parse
  - Maintains expression evaluation statistics
    - Frequency of execution
    - Cost of evaluation
    - Timestamp of evaluation
    - "Hotness" of score
  - Persisted to disk (and cached in SGA)
  - Exposed in the view all|dba|user_expression_statistics

New In-Memory features in 12cR2

In-Memory Expressions

```
-- Identify and capture the 20 most frequently accessed ('hottest')
-- expressions from the ESS in the specified time range. CURRENT=last
-- 24 hours. CUMULATIVE=since DB creation. Only expressions from
-- tables in the IM (at least partly populated) are considered.
BEGIN
   dbms_inmemory_admin.ime_capture_expressions('CURRENT');
END;
/

-- Check what expressions were captured
SELECT * FROM user_im_expressions;

-- Populate the captured expression in the IM column store.
-- Alternatively wait until the table gets repopulated.
BEGIN
   dbms_inmemory_admin.ime_populate_expressions;
END;
/
```

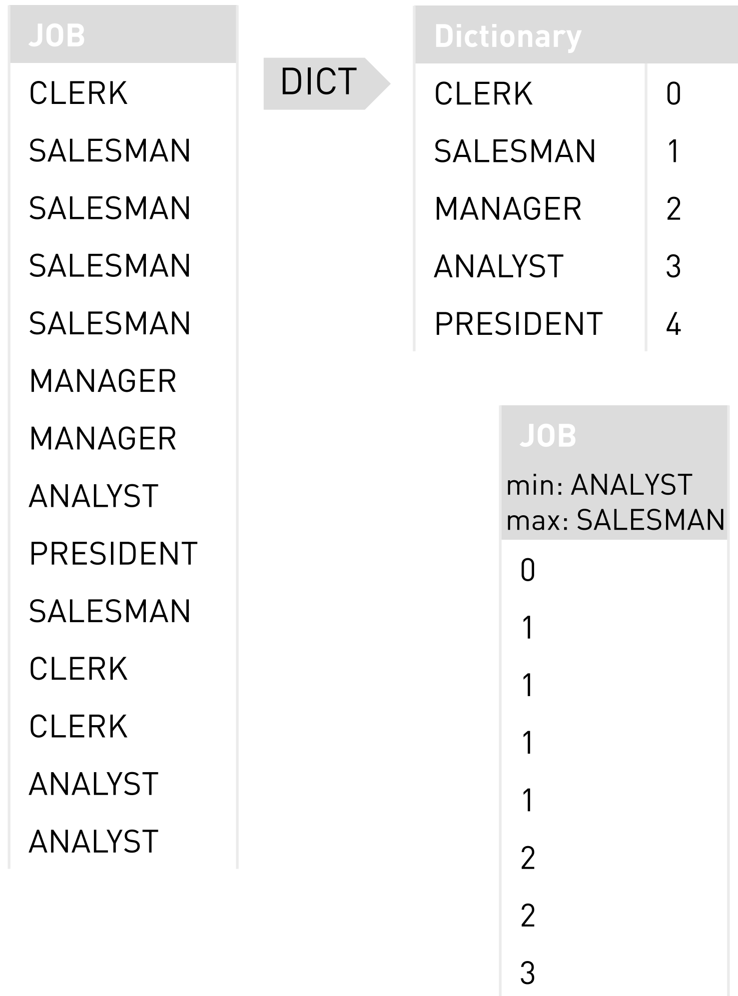
New In-Memory features in 12cR2
In-Memory Expressions

In-Memory Expressions

- IMEs are stored in In-Memory Expression Units → IMEU (like In-Memory Column Units (IMCUs) with subtle differences)
- 1-to-1 mapping with rows of IMCU
- Tied to parent IMCU lifespan
- Memory from IMCU 1MB pool
- Consider the space usage in IM for the IMEUs. It may consume considerable amount of space with default compression level
- Generally provides same statistics as for CUs:
  - IM populate EUs ...
  - IM prepopulate EUs ...
  - IM repopulate EUs ...
  - IM scan EUs ...
    - e.g. IM scan EU bytes in-memory, IM scan EU rows

Demo

New In-Memory features in 12cR2
Join-Groups



Column Compression Unit (CU)

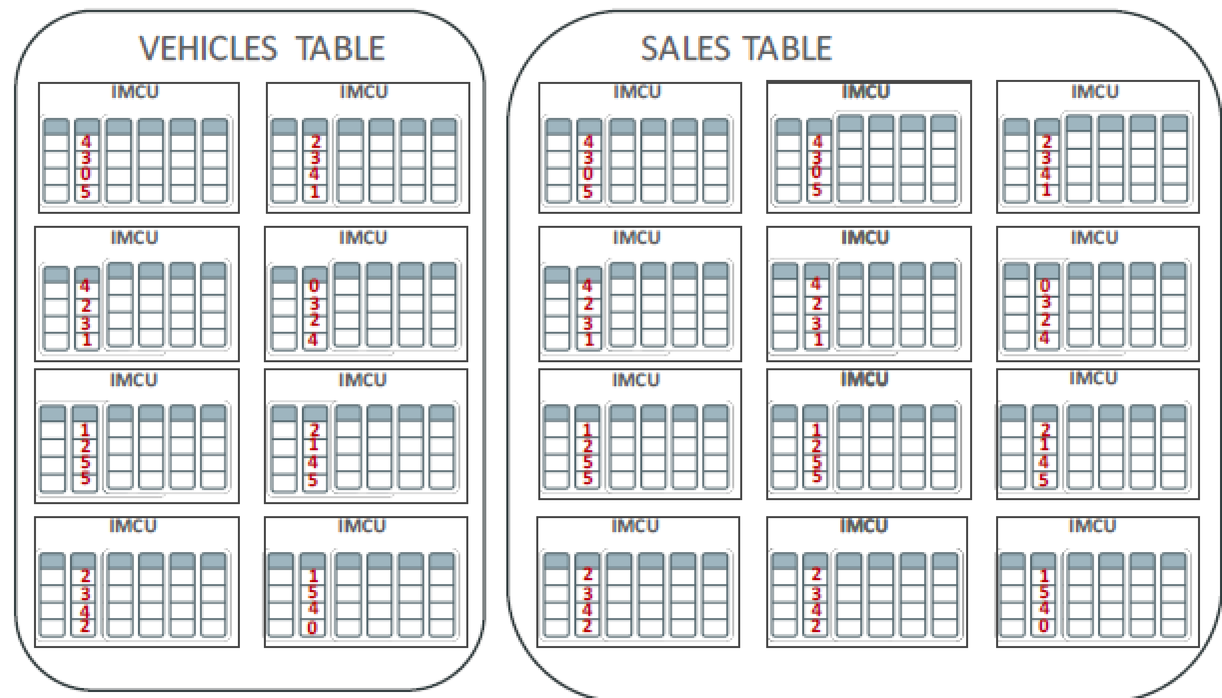
- Most CUs have a "Dictionary"
- – Sorted list of distinct values in the CU
- – Column values replaced with dictionary IDs
- Idea of join groups is to **compress the join columns in both tables based on the same dictionary**
- **Join occurs on dictionary values rather than on data**
  - Saves on decompressing data
  - Saves on hashing the data

New In-Memory features in 12cR2
Join-Groups

Join between Vehicles and Sales based on brand (NAME)

Global Dictionary

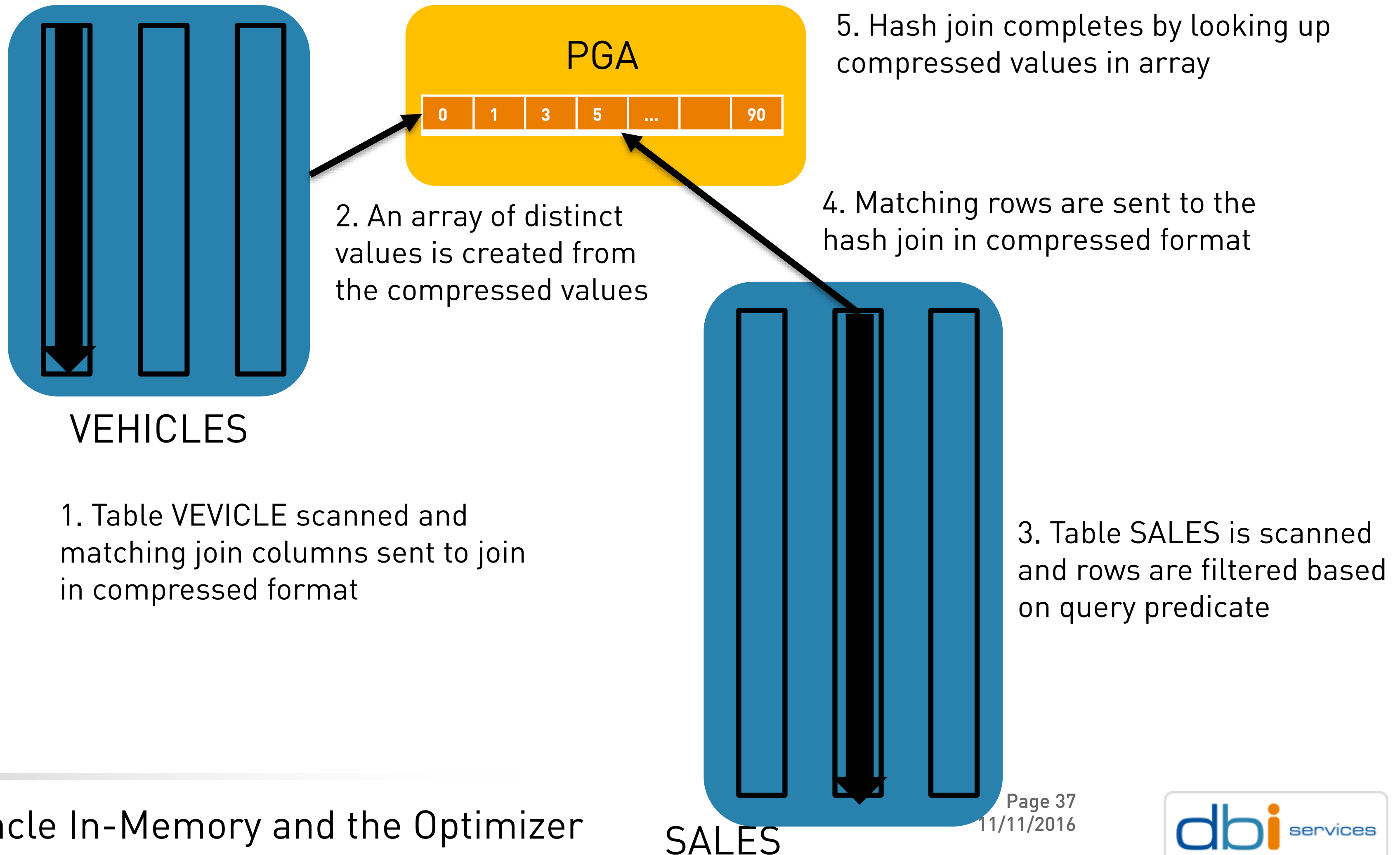
NAME	ID
AUDI	0
BMW	1
CADILLAC	2
PORSCHE	3
TESLA	4
VW	5



Global dictionary created when first table is populated and used for joining columns in both tables.

New In-Memory features in 12cR2
Join-Groups

Join between Vehicles and Sales based on brand (NAME)



dbi services

New In-Memory features in 12cR2
Join-Groups

Creating and managing a Join Group

```
CREATE INMEMORY JOIN GROUP vech_Sales_jg
(VEHICLES(NAME),SALES(NAME));

SELECT o.object_name table_name, c.column_name column_name,
gd.head_address "GD Address"
FROM user_objects o, user_tab_columns c, v$im_segdict gd
WHERE gd.objn = o.object_id
AND o.object_name = c.table_name
AND gd.column_number = c.column_id;
```

Conclusion
Oracle In-Memory and the Optimizer

- The optimizer is In-Memory aware
- In-Memory statistics are gathered/considered at parse time
- New optimizer transformations and join methods available
- Additional hints are available to (not) use In-Memory
- In-Memory is very good joining small datasets with large datasets
- Very good support of parallelism
- DDLs gain from IM as well: CTAS, Create Index
- Adaptive Stats read from IM

- License costs (Active Data Guard + In-Memory)
- Joining large datasets may not be faster with IM
- Some features available on engineered systems only

Any questions? Please do ask

Clemens Bleile
Senior Consultant

+41 78 677 51 09
clemens.bleile@dbi-services.com

We look forward to working with you!

Let's meet at booth 242