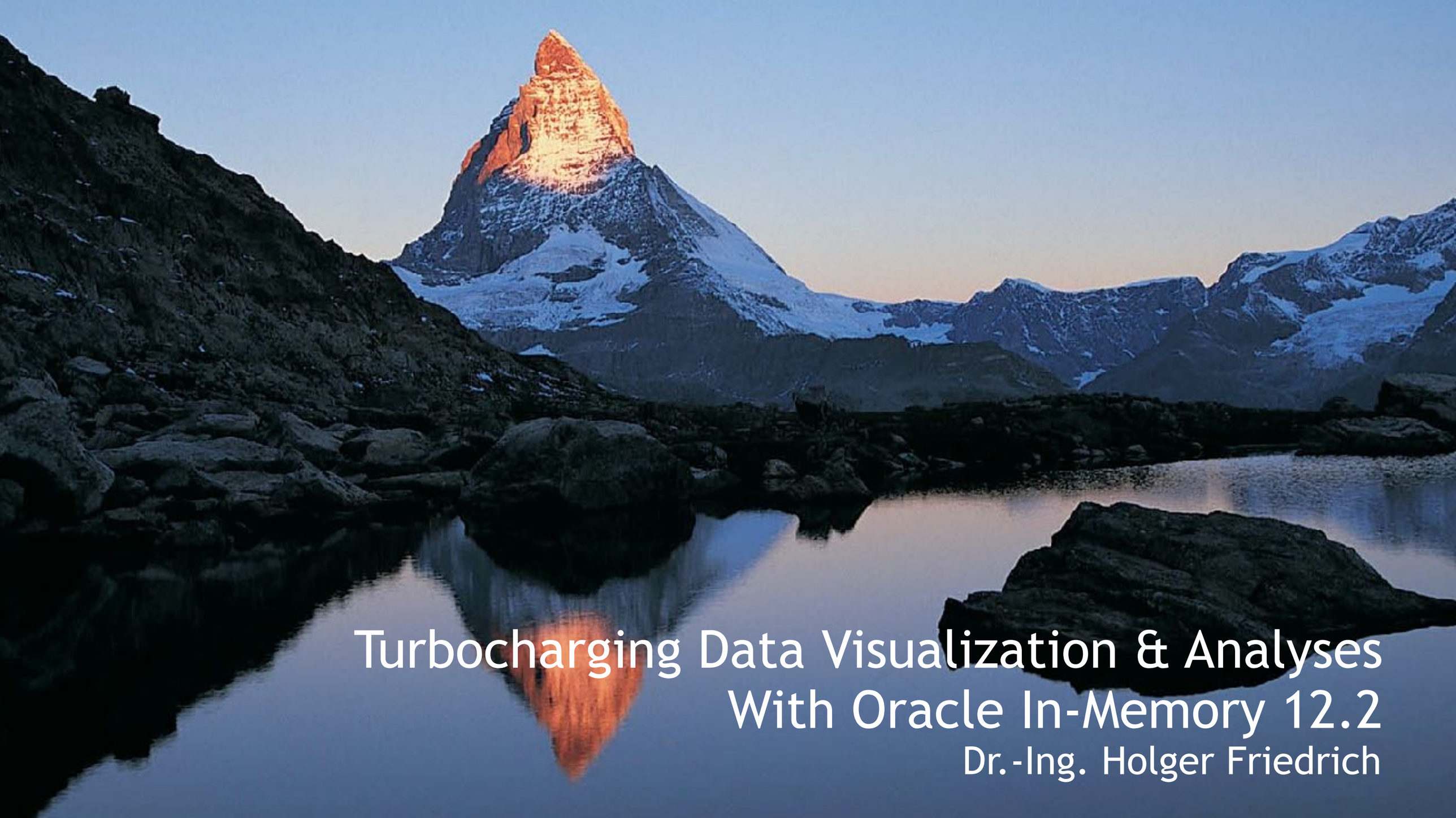


Turbocharging Data Visualization & Analyses With Oracle In-Memory 12.2

Dr.-Ing. Holger Friedrich

sumIT
get value from data

BIWA SIG
Better Information Better Results
www.oraclebiwa.org

Turbocharging Data Visualization and Analyses with Oracle In-Memory 12.2

Dr.-Ing. Holger Friedrich, CTO
sumIT AG

Agenda

- Introduction
- Analytic Workloads & Requirements
- Oracle In-Memory (12.2)
- Conclusions

sumIT AG

- Consulting and implementation services in Switzerland
- Experts for
  - Data Warehousing,
  - Business Intelligence,
  - Advanced Analytics
- Focussed on Oracle technology
- Oracle specialisations for BI Foundation, Database, Data Warehousing, and Data Integration
- Our motto: Get Value From Data
- Visit our web site: www.sumit.ch (in German)

ORACLE Gold Partner

ORACLE Gold Partner
Specialized
Oracle Business Intelligence Foundation

ORACLE Gold Partner
Specialized
Data Warehousing

Holger Friedrich

- Computer Science diploma of Karlsruhe Institute of Technology (KIT)
- Ph.D. in Robotics and Machine Learning
- More than 17 years experience with Oracle technology
- Expert for
  - Data Integration
  - Data Warehousing / Big Data,
  - Advanced Analytics and
  - Business Intelligence
- Technical Director of sumIT AG

Oracle ACE for DWH/BI

On-Premises


Cloud

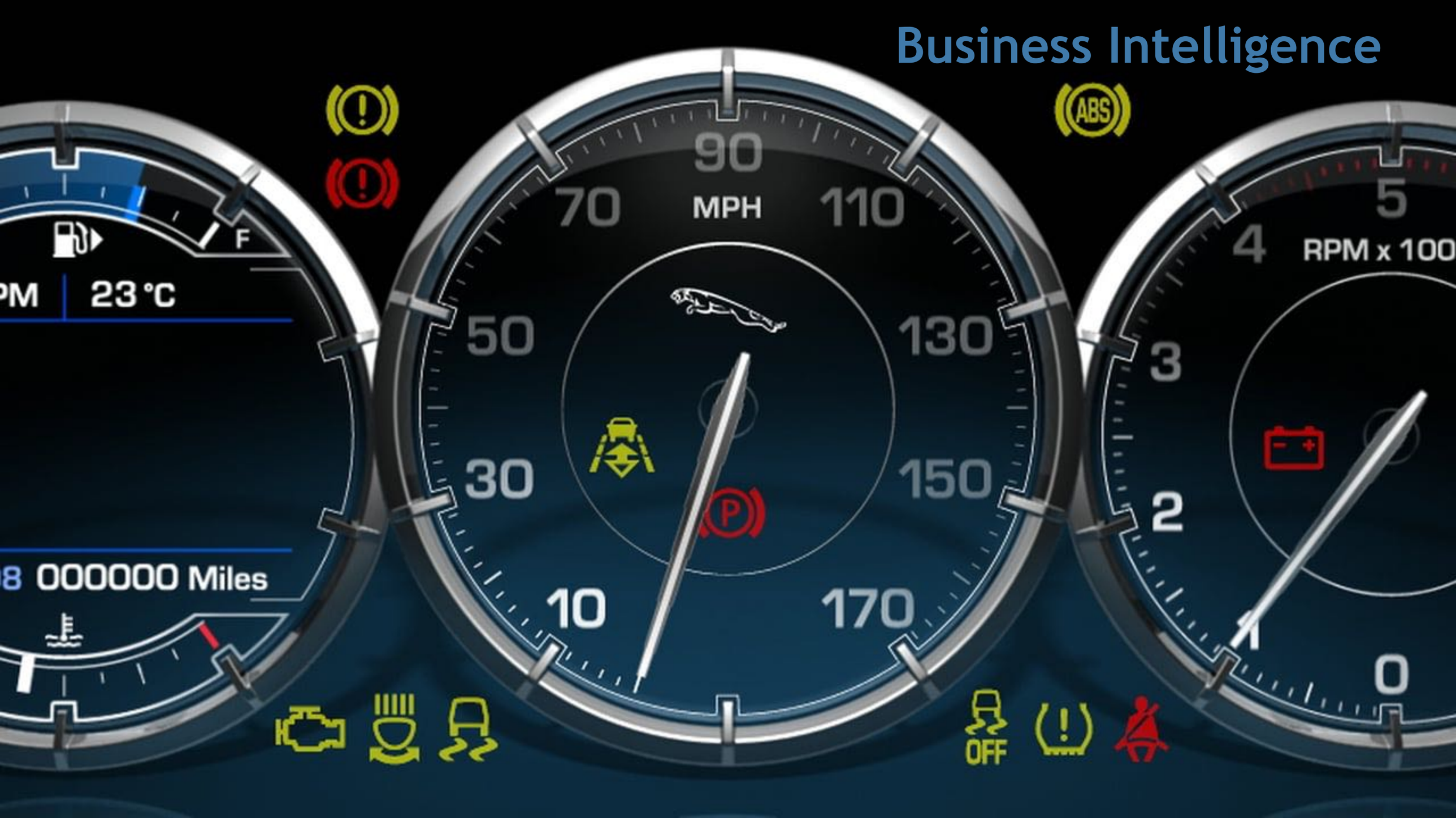
RDBMS

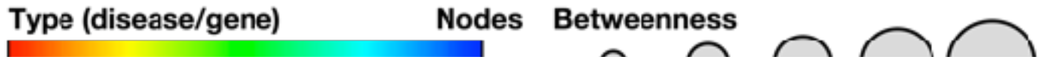
Big Data

Batch

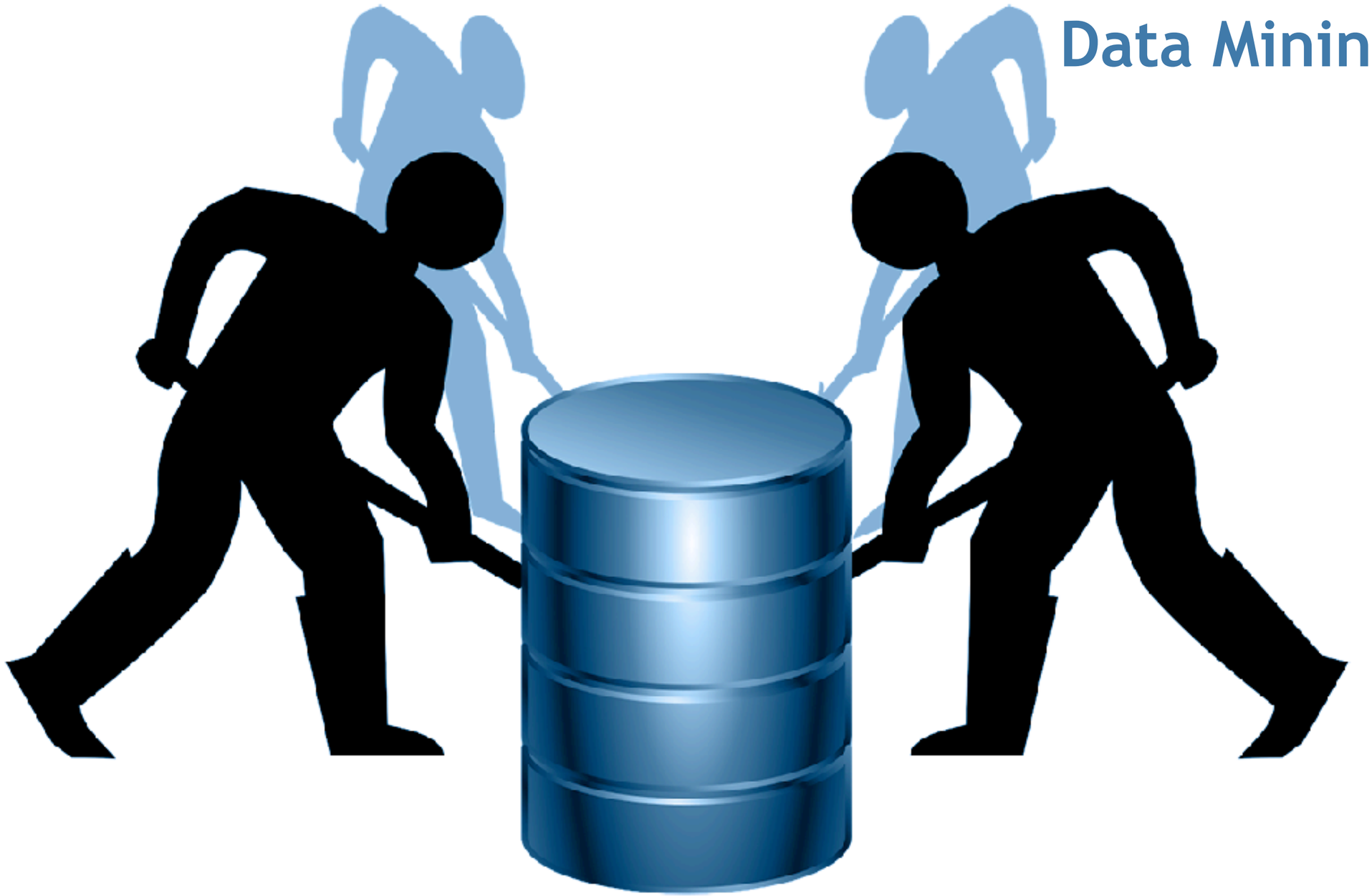
Streaming

Business Intelligence

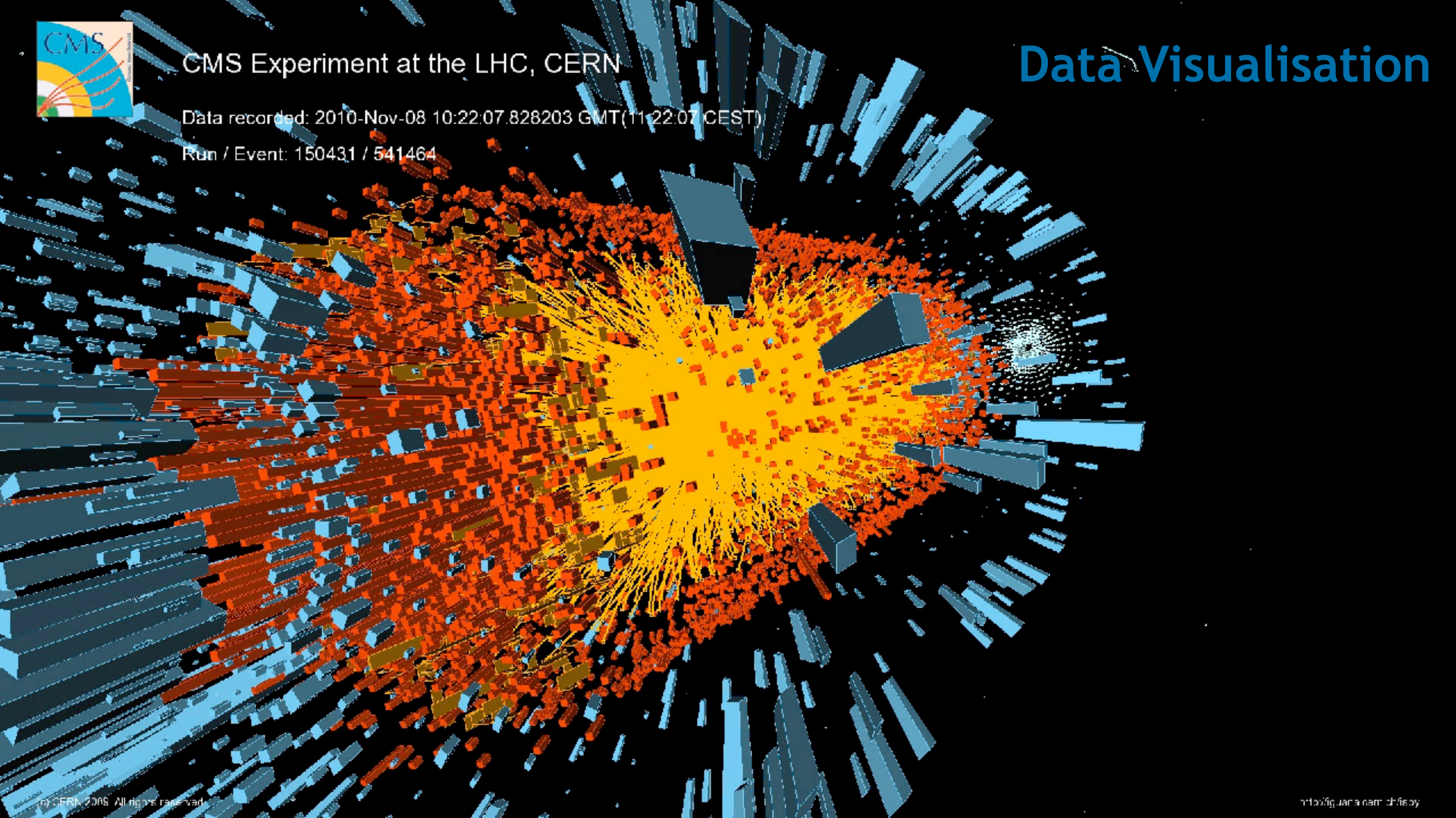




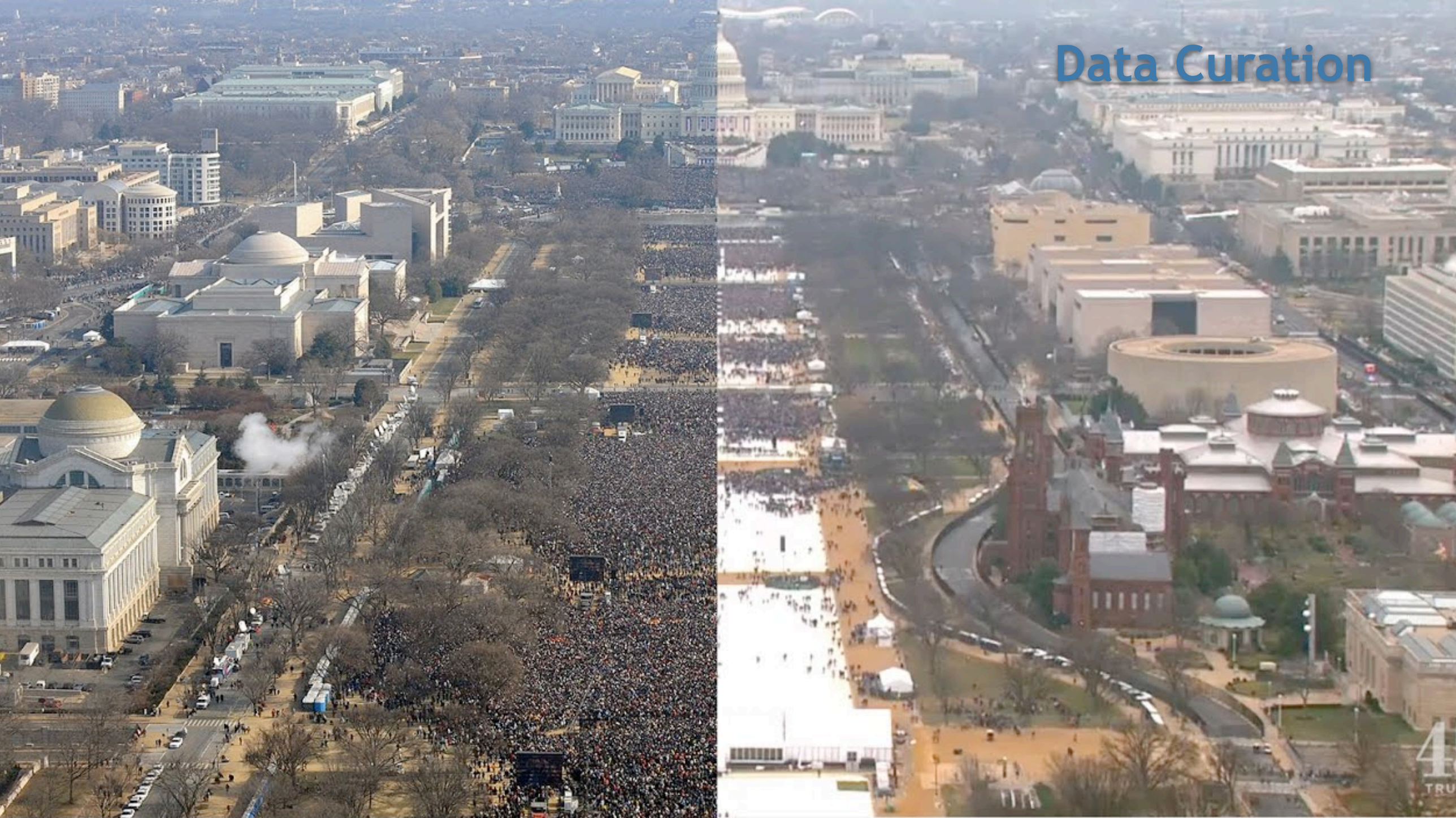
Data Mining



Data Visualisation



Data Curation



Single Source of Truth

(Non-Streaming) Workload Characteristics

- Scans of large amounts of data
- Apply functions to single columns
- Compute complex expressions on multiple attributes,
- Aggregate data
- Join data sets through known attribute pairs
- Flexibly join data sets through arbitrary attribute pairs
- Data sets do range from small to very large
- Query sensitive data sets
- Query new data sets (schema on read, JSON, etc.)

→ • All of the above with high speed

Columnar Databases

- Best for queries that
  - scan large quantities of data
  - compute aggregates on the results
- Efficient storage use, due to high compression benefits on most columns

Features Around Since First Release

1. In-memory storage indexes
2. Filtering on binary compressed data
3. Columnar storage of selected columns
4. Transparent querying across storage hierarchy
5. Real-time background synching of columnar store
6. Parallel query execution on the columnar store
7. SIMD vector processing
8. In-memory fault tolerance on RAC
9. In-memory aggregation

Scans & Single Column Computations in12.1

```
SELECT
  /*+  no_parallel(t) no_parallel_index(t) dbms_stats cursor_sharing_exact
       use_weak_name_resl dynamic_sampling(0) no_monitoring
       xmlindex_sel_idx_tbl no_substrb_pad  */
  COUNT(*),
  SUM(sys_op_opnsize("PROD_ID")),
  SUM(sys_op_opnsize("CUST_ID")),
  COUNT(DISTINCT "CHANNEL_ID"),
  SUM(sys_op_opnsize("CHANNEL_ID")),
  substrb(dump(MIN("CHANNEL_ID"),16,0,64),1,240),
  substrb(dump(MAX("CHANNEL_ID"),16,0,64),1,240),
  COUNT(DISTINCT "PROMO_ID"),
  SUM(sys_op_opnsize("PROMO_ID")),
  substrb(dump(MIN("PROMO_ID"),16,0,64),1,240),
  substrb(dump(MAX("PROMO_ID"),16,0,64),1,240),
  COUNT(DISTINCT "QUANTITY_SOLD"),
  SUM(sys_op_opnsize("QUANTITY_SOLD")),
  substrb(dump(MIN("QUANTITY_SOLD"),16,0,64),1,240),
  substrb(dump(MAX("QUANTITY_SOLD"),16,0,64),1,240),
  SUM(sys_op_opnsize("AMOUNT_SOLD"))
FROM "SH"."MY_SALES" t;
```

4. aggregations

1. scanning columns

3. post-processing scanned columns

2. single table scan

Scan Performance in 12.1

- In-Memory

```
-------------------------------------------------------------------------------
| Id  | Operation                    | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |           |     1 |    24 |  1070   (27)| 00:00:01 |
|   1 |  SORT GROUP BY               |           |     1 |    24 |            |          |
|   2 |   TABLE ACCESS INMEMORY FULL| MY_SALES  |   14M|   336M|  1070   (27)| 00:00:01 |
-------------------------------------------------------------------------------
```

1. In-Memory chosen automatically

3. Huge cost advantage due to IM

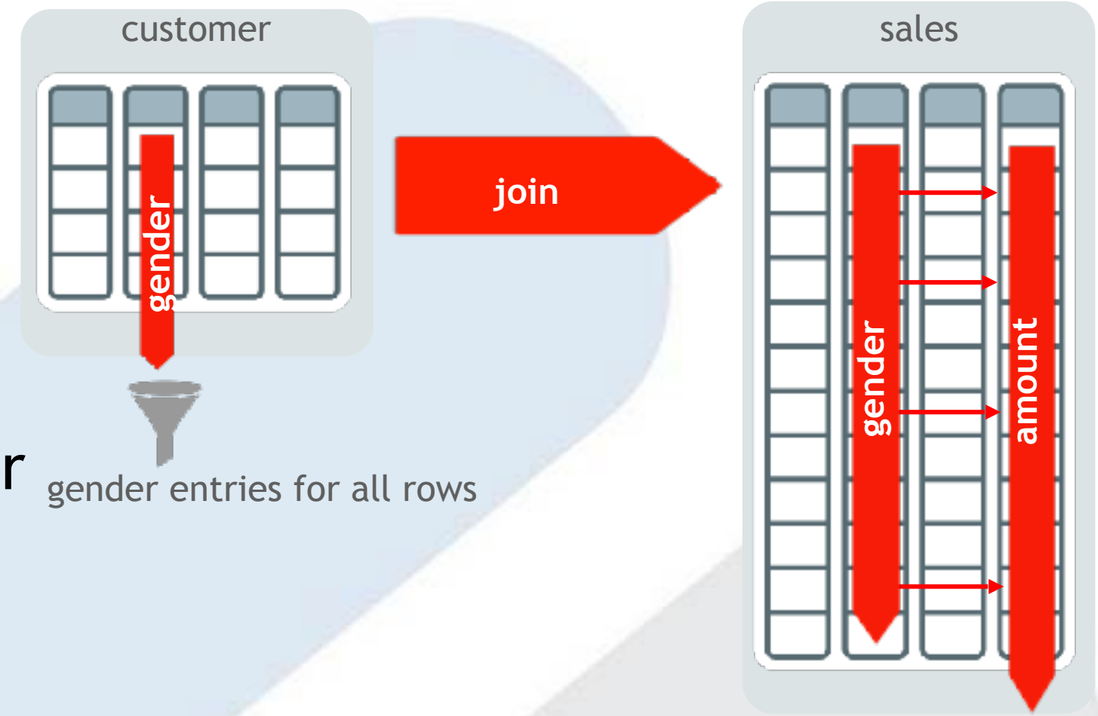
- With explicit NO_INMEMORY Hint

```
-------------------------------------------------------------------------------
| Id  | Operation          | Name      | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT   |           |     1 |    24 | 19350   (1)| 00:00:01 |
|   1 |  SORT GROUP BY     |           |     1 |    24 |            |          |
|   2 |   TABLE ACCESS FULL| MY_SALES  |   14M|   336M| 19350   (1)| 00:00:01 |
-------------------------------------------------------------------------------
```

2. Switched to row storage SGA, Flash, or Disk

Flexible Joins in 12.1

- Join tables on flexible criteria
- No special model required, e.g. star schema
- No indices required
- Speed up joins by use of bloom filters (not on all occasions)
- Small hash joins in PGA
- However data has to be decompressed for
  - bloom filter construction
  - hashing

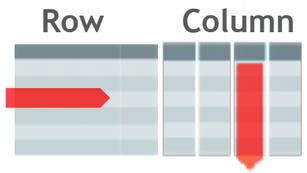


What's New in 12.2?

Real-Time Analytics

2X Faster Joins
5X Faster Expressions

Mixed Workload



Active Data
Guard Support

Massive Capacity

In-Memory on
Exadata Flash

Multi-model

Native support for
JSON Data type

Automation

Dynamic Data
Movement Between
Storage & Memory

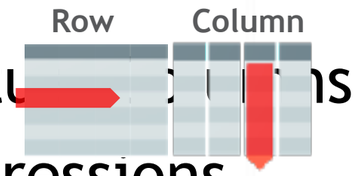
12.2 - Real-Time Analytics

Real-Time Analytics

- Join Groups
- In-Memory Virtual Columns
- In-Memory Expressions

2X Faster Joins
5X Faster Expressions

Mixed Workload



Active Data Guard Support

Massive Capacity

In-Memory on Exadata Flash

Multi-model

Native support for JSON Data type

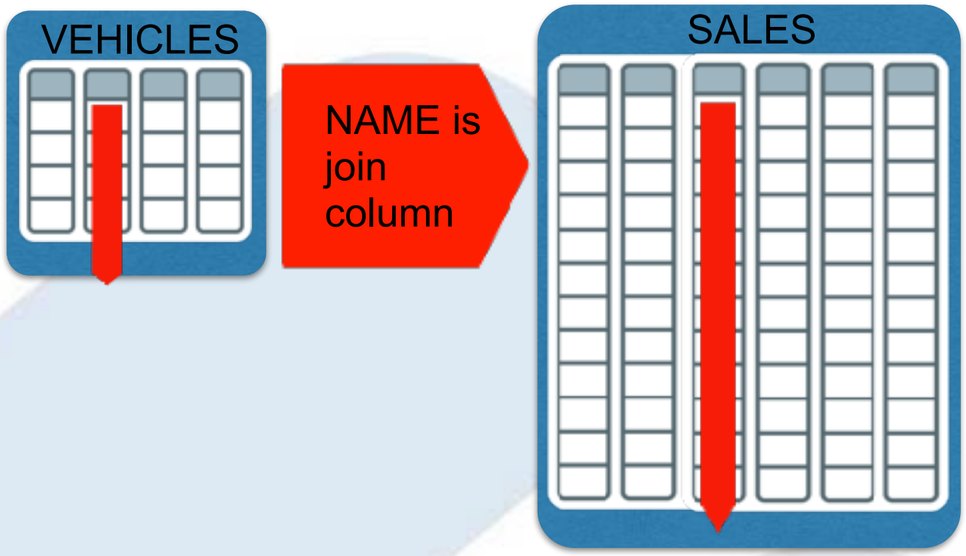
Automation

Dynamic Data Movement Between Storage & Memory

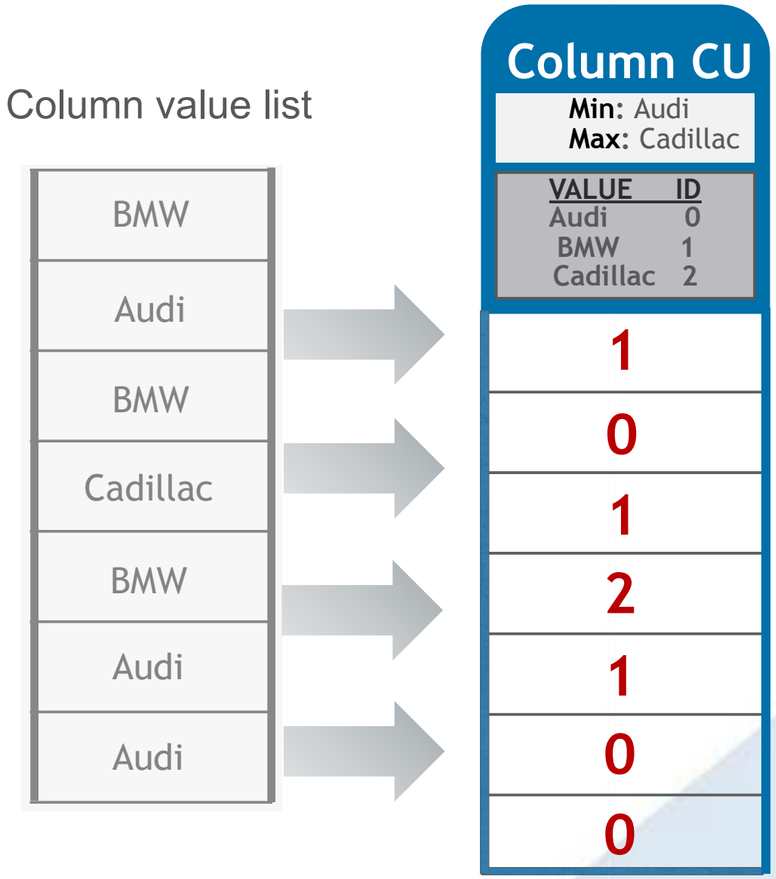
Join Groups: Faster Hash Joins

- Join columns in both tables are compressed using the same dictionary
- Joins occur on dictionary values rather than on data
  - Saves on decompression of data
  - Save on hashing the data

Example: Find sales price of each Vehicle



Local Dictionary at CU level

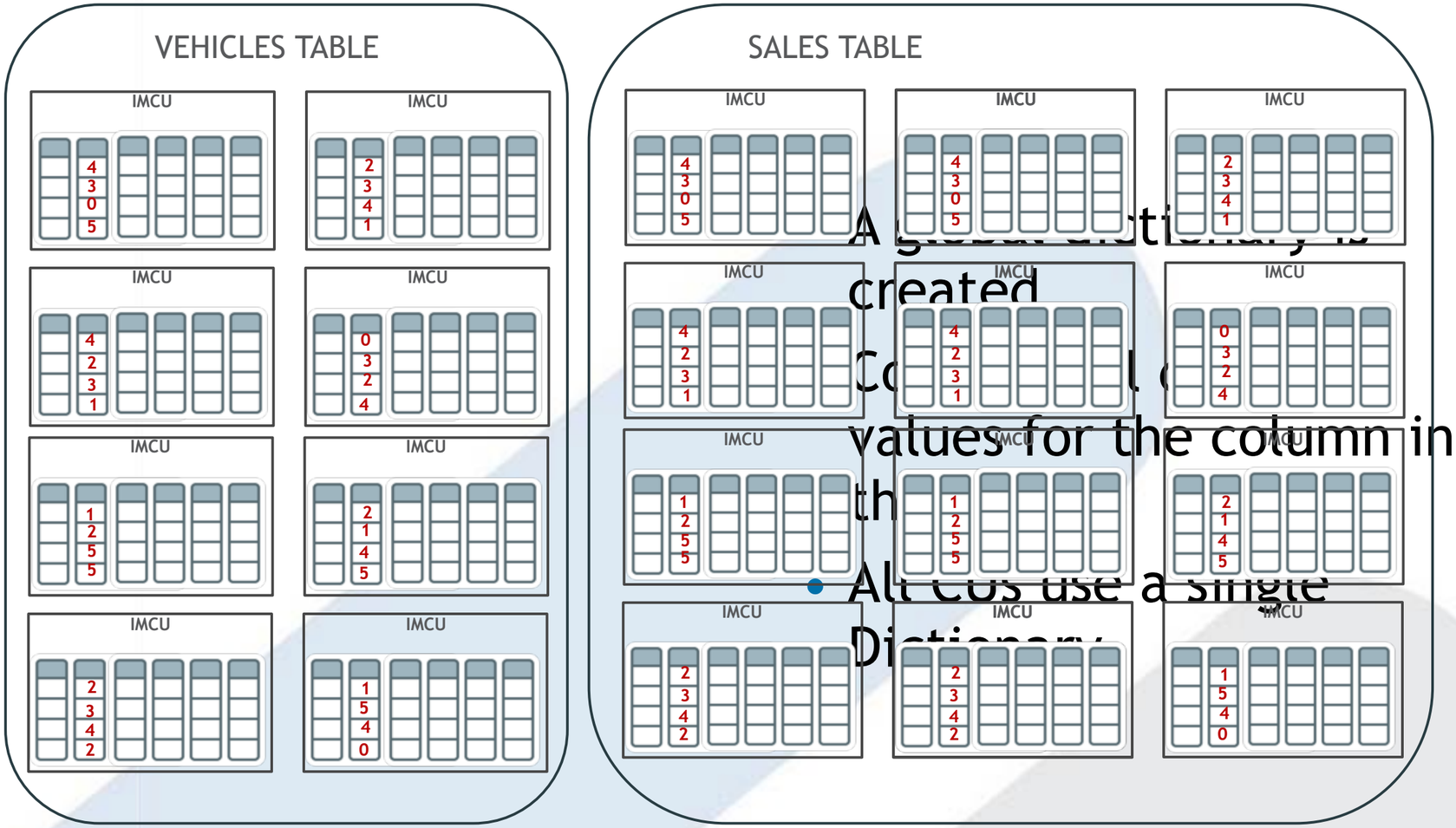


- Contiguous storage per column in an IMCU
- All CUs automatically store Min/Max values
- Multiple formats: depends on data and chosen compression level
- Most CUs have a “Dictionary”
  - Sorted list of distinct values in the CU
  - Column values replaced with dictionary IDs
- Additional compression of value list possible (e.g. Run Length Encoding, OZIP)

Same Global Dictionary used by Both Tables

Global Dictionary

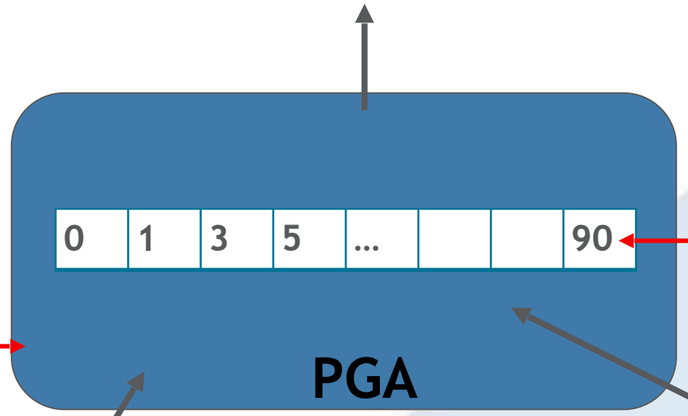
NAME	ID
AUDI	0
BMW	1
CADILLAC	2
PORSCHE	3
TESLA	4
VW	5



- Global Dictionary created when 1st table is populated & used for join column in both tables

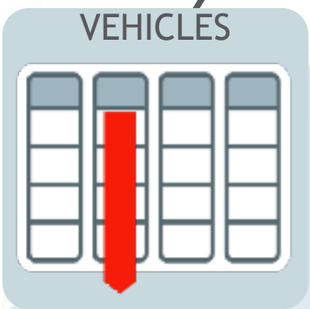
Hash Join With Join Group

2. Table Array of compressed values:
An array of distinct values is created from the compressed values

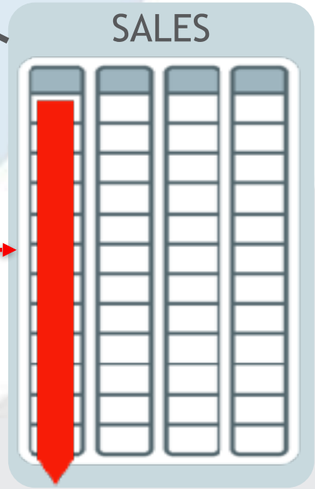


4. Hash Join: Join completed by looking up compressed values in the array

1. Table access inmemory full:
VEHICLES table is scanned and join column values sent to join in compressed format



3. Table access inmemory full:
SALES table is scanned and data in the name column is sent to the hash join in its compressed format



Managing and Monitoring (I)

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

- Create Join Group
- Populate tables into column store
- Make sure that the global dictionary exists
- Use dictionary view user_joingroup to monitor

```
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;
```

Managing and Monitoring (II)

Overview

General

SQL Text	SELECT --+ no_vector_transform monitor count("Custome ...
Execution Started	Thu Dec 10, 2015 7:03:15 PM
Last Refresh Time	Thu Dec 10, 2015 7:03:20 PM
Execution ID	16777232
User	VENDAVO
Fetch Calls	1

Time & Wait Statistics

Duration	5.0s
Database Time	4.8s
PL/SQL & Java	0s
Activity %	100

Details

Plan Statistics | Plan | Activity | Metrics

Plan Hash Value 2520514737 | Plan Note

Line ID	Operation	Name	Estim
0	SELECT STATEMENT		
1	SORT AGGREGATE		
2	HASH JOIN		
3	TABLE ACCESS FULL	V_PA_EXCHR_RT_JG	
4	PARTITION RANGE ALL		
5	TABLE ACCESS INMEMORY FULL	V_PA_TRANSACTIONS	

Other Plan Line Statistics

Build Size	983K
Build Row Count	380
Fan-out	8
Slot Size	123K
Total Build Partitions	8
Total Cached Partitions	8
Columnar Encodings Leveraged	1

OK

In-Memory Virtual Columns

- Virtual Columns
  - do represent expressions involving columns in a table
  - are computed based on values within a record of a table
  - are not represented on disk
  - can be queried like any regular columns

- Syntax:

```
Alter table Lineorder
        Add (net_price AS
              (l_extendedprice *
               (1 - l_discount)));
```

Rationale For In-Memory Virtual Columns

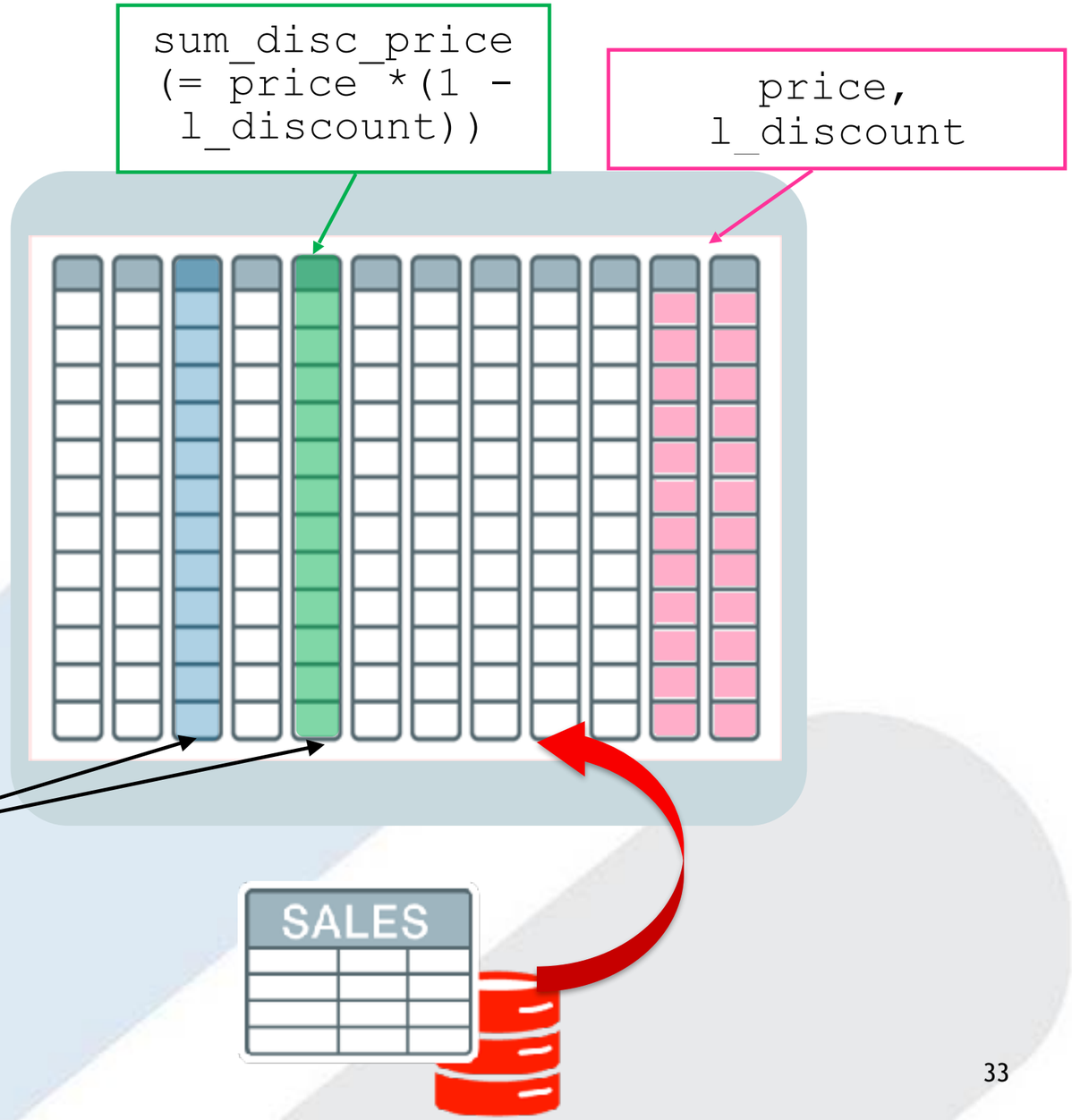
- Evaluating expressions repeatedly can be computationally expensive
- Repeated expressions are typical for analytic queries

```
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 charge,
         avg(l_quantity) as avg_qty, avg(l_extendedprice) as avg_price,
         avg(l_discount) as avg_disc,
         count(*) as count_order
From     lineorder
Where    l_shipdate <= to_date ('1998-12-01','YYYY-MM-DD')  -  90
group by l_returnflag,
         l_linestatus
order by l_returnflag, l_linestatus;
```

How IM Virtual Columns Work

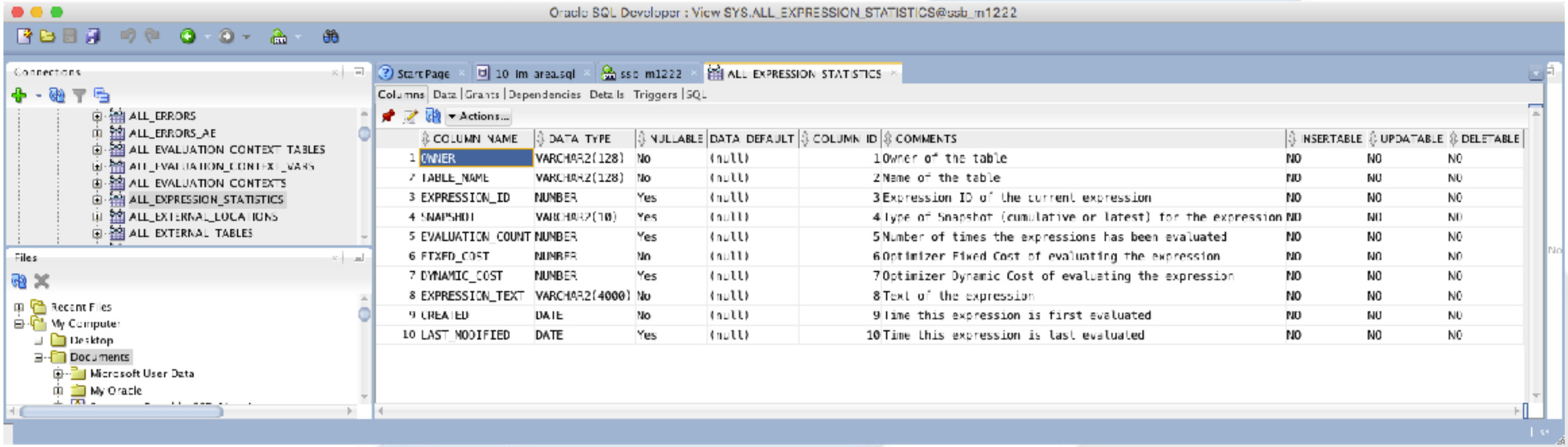
- Exist only in columnar store
- All performance features of In-Memory option are supported
  - IMCU pruning through storage indexes
  - SIMD
  - …

```
Select sum_disc_price
From Sales
Where region = 'CA';
```



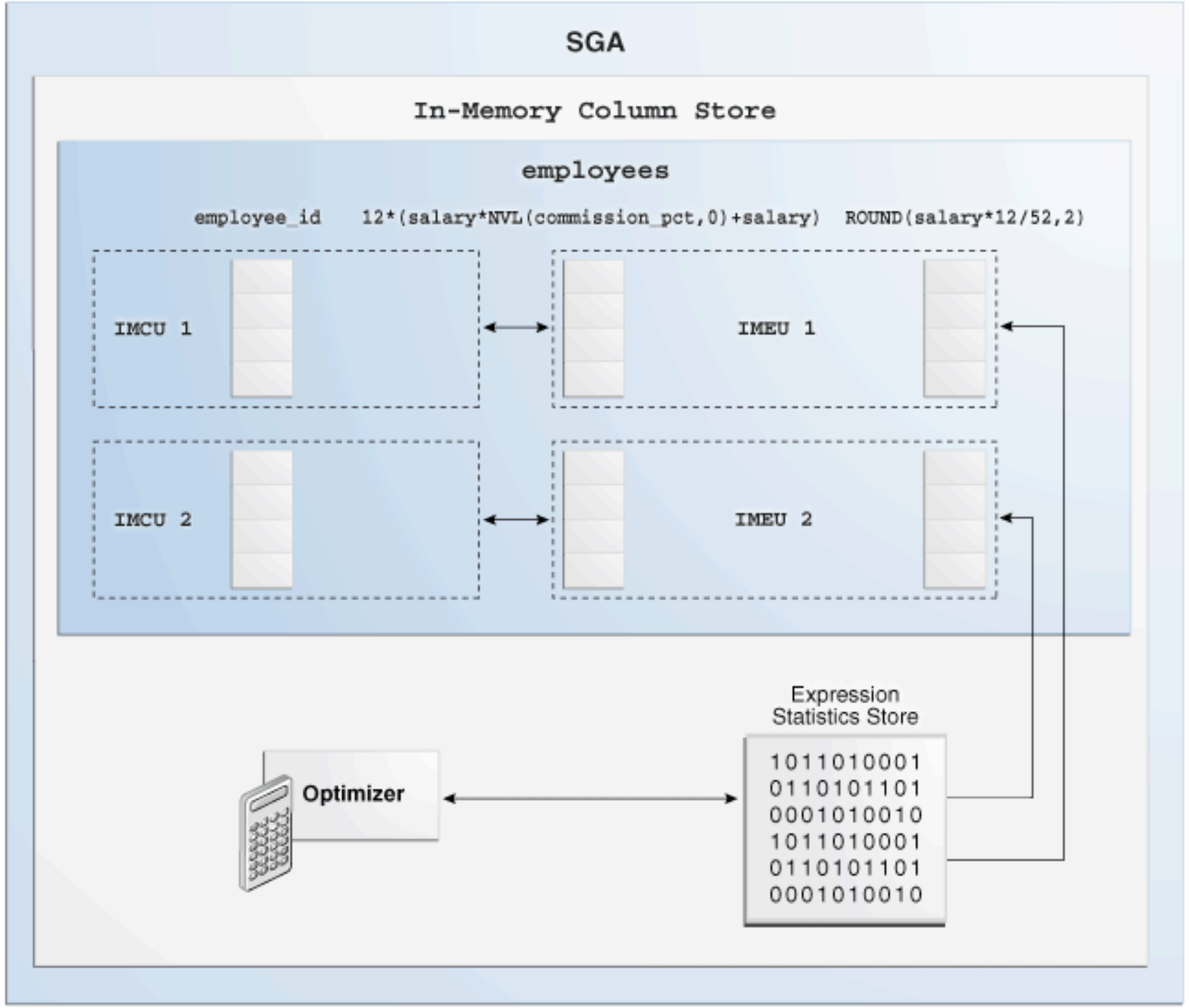
In-Memory Expressions

- Often used expressions that are automatically captured
- Capturing based on the Expression Statistics Store
- The database determines ‘hot’ expressions autonomously
- Top 50 expressions are added as virtual IM columns
- Populated automatically during next (re-)population of columnar store



	COLUMN_NAME	DATA_TYPE	NULLABLE	DATA_DEFAULT	COLUMN_ID	COMMENTS	INSERTABLE	UPDATABLE	DELETABLE
1	OWNER	VARCHAR2(128)	No	(null)	1	Owner of the table	NO	NO	NO
2	TABLE_NAME	VARCHAR2(128)	No	(null)	2	Name of the table	NO	NO	NO
3	EXPRESSION_ID	NUMBER	Yes	(null)	3	Expression ID of the current expression	NO	NO	NO
4	SNAPSHOT	VARCHAR2(10)	Yes	(null)	4	Type of Snapshot (cumulative or latest) for the expression	NO	NO	NO
5	EVALUATION_COUNT	NUMBER	Yes	(null)	5	Number of times the expressions has been evaluated	NO	NO	NO
6	FIXED_COST	NUMBER	No	(null)	6	Optimizer Fixed Cost of evaluating the expression	NO	NO	NO
7	DYNAMIC_COST	NUMBER	Yes	(null)	7	Optimizer Dynamic Cost of evaluating the expression	NO	NO	NO
8	EXPRESSION_TEXT	VARCHAR2(4000)	No	(null)	8	Text of the expression	NO	NO	NO
9	CREATED	DATE	No	(null)	9	Time this expression is first evaluated	NO	NO	NO
10	LAST_MODIFIED	DATE	Yes	(null)	10	Time this expression is last evaluated	NO	NO	NO

In-Memory Expression Units



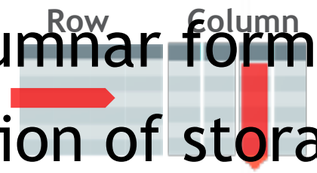
12.2 - Massive Storage Capacity

- In-Memory columnar format in flash cache
- Massive extension of storage capacity for the In-memory option

Real-Time Analytics

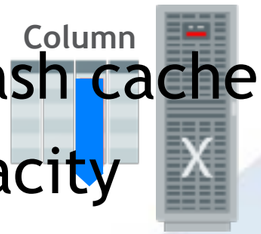
2X Faster Joins
5X Faster Expressions

Mixed Workload



Active Data Guard Support

Massive Capacity



In-Memory on Exadata Flash

Multi-model

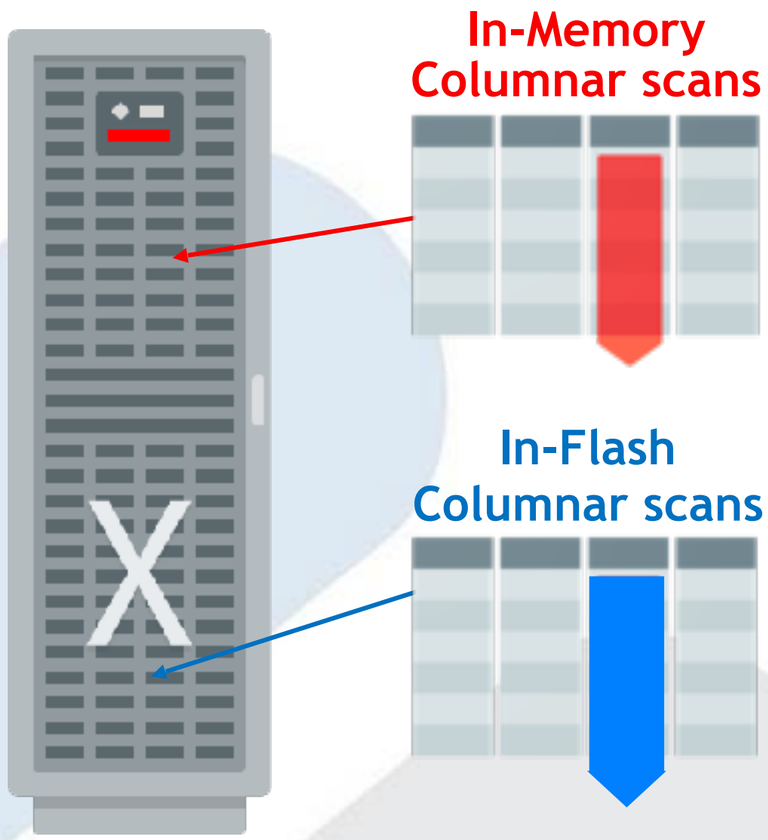
Native support for JSON Data type

Automation

Dynamic Data Movement Between Storage & Memory

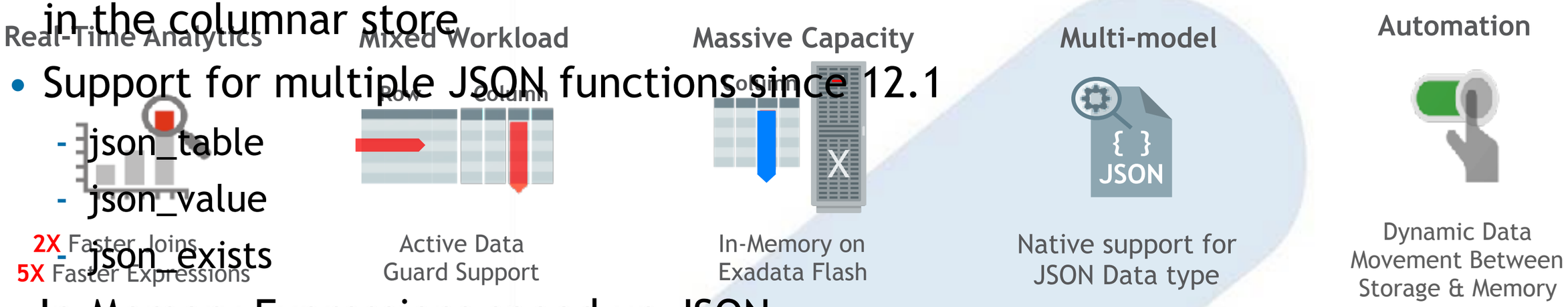
Use Of The Available Flash Cache

- New smart columnar flash cache format
- Available on Exadata
- Additional optimisations,
  e.g. evaluation of multiple column values
  in single vector instruction
- Optimiser spreads queries across DRAM & flash
- Extends available storage by factor 10
- Currently unique to Oracle In-Memory
- HCC objects are automatically populated
  into flash in IM-Columnar format



12.2 - Accelerate JSON Access

- Make use of JSON binary representation in the columnar store
- Support for multiple JSON functions since 12.1
  - json_table
  - json_value
  - json_exists
- In-Memory Expressions speed up JSON analytics further in IM 12.2



Conclusions

- Today's variety of anlaytic tools and tasks generate
  - more diverse workloads
  - require a very fast, flexible, agile, and robust query engine
- Oracle In-Memory boosts many of today's analytical workloads
- The new 12.2 features
  - In-memory Expressions
  - Join Groups
  - JSON optimisations
  - In-Memory Columnar Storage in Flash

  provide unprecedented performance
- Big Data SQL extends its use and integrates it with HADOOP clusters

SUMSIT
get value from data