ORACLE®

# Optimizing Table Scans in the Cloud

John Clarke
Software Development Director
Real-World Performance
Server Technologies

# 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.

# What is Real-World Performance in 2018?

## Bridging the Divide from Today's Performance to What is Possible

# Real-World Performance 2018

## Who We Are

- Part of the Database Development Organization
- Global Team located in USA, Europe, Asia
- 350+ combined years of Oracle database experience
- Innovate to achieve exceptional Database Performance
- Our methods:
    - Use the product as it was designed to be used
    - Numerical and logical debugging techniques
    - Educate others about the best performance methods and techniques
    - Avoid and eliminate “tuning” by hacking/guessing/luck

# We’ve Been Here Before

- How many rows do you need to find?
  a) One
  b) A few
  c) A lot
  **d) I don’t know**
- Do you scan or use an index?
- If you don’t know, what access method is the least risky?

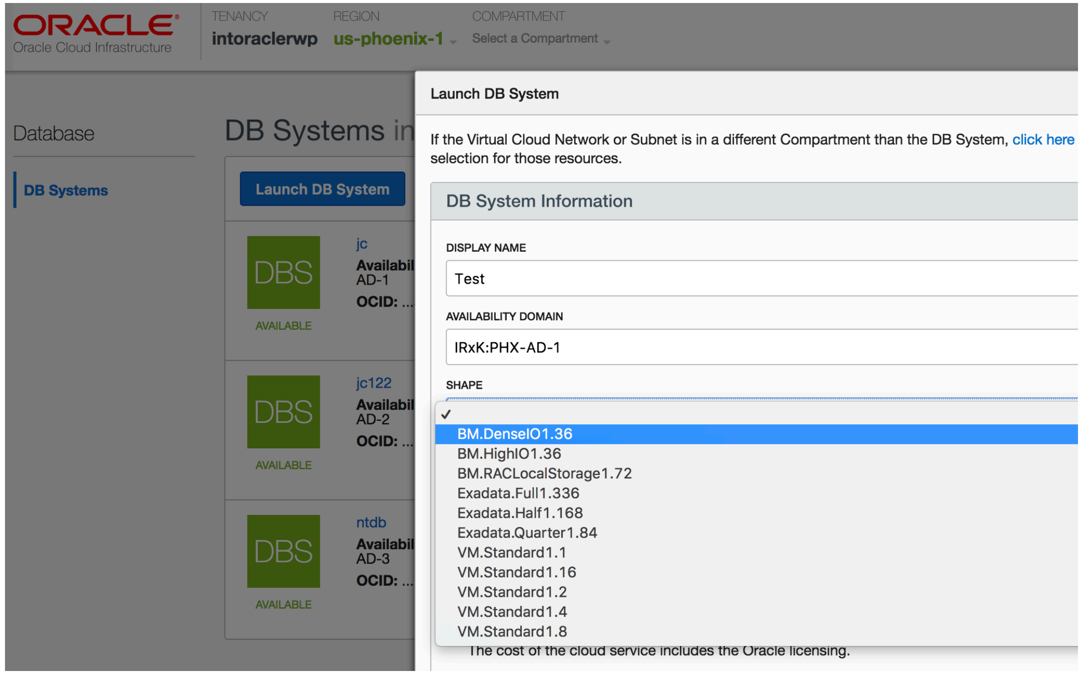
# The Question We’re Asking

```
SELECT d_year, d_sellingseason, c_region, SUM(lo_extendedprice),SUM(lo_supplycost)
       FROM    lineorder
                 JOIN    customer        ON lo_custkey = c_custkey
                 JOIN    date_dim        ON lo_orderdate = d_datekey
                 JOIN    part            ON lo_partkey = p_partkey
                 JOIN    supplier        ON lo_suppkey = s_suppkey
       WHERE  d_month IN (’June’,’July’,’August’)
       AND    p_mfgr  IN (’MFGR#1’,’MFGR#2’)
       AND    s_nation = ’China’
GROUP BY      d_year, d_sellingseason, c_region
ORDER BY      d_year, d_sellingseason, c_region
```

“Show me the price and cost by year, selling season, and customer region for all goods sold in June, July, and August for parts manufactured by MFGR#1 and MFGR#2 in China”

# Where We’re Asking the Question

## Oracle Cloud Infrastructure



Launch DB System

If the Virtual Cloud Network or Subnet is in a different Compartment than the DB System, click here
selection for those resources.

DB System Information

DISPLAY NAME: Test

AVAILABILITY DOMAIN: IRxK:PHX-AD-1

SHAPE

- ✓ BM.DenseIO1.36
- BM.HighIO1.36
- BM.RACLocalStorage1.72
- Exadata.Full1.336
- Exadata.Half1.168
- Exadata.Quarter1.84
- VM.Standard1.1
- VM.Standard1.16
- VM.Standard1.2
- VM.Standard1.4
- VM.Standard1.8

The cost of the cloud service includes the Oracle licensing.

- Oracle Database Cloud Service – Oracle Cloud Infrastructure
- Oracle Exadata Cloud Service - Oracle Cloud Infrastructure

ORACLE REAL-WORLD PERFORMANCE

# Agenda

1 Why Table Scans?

2 Making Scans Smaller

3 Where's Our Leverage?

4 Rearranging the Data

5 Things We Can Do to Speed Up Next Operation in Plan

# Agenda

1 Why Table Scans?

2 Making Scans Smaller

3 Where’s Our Leverage?

4 Rearranging the Data

5 Things We Can Do to Speed Up Next Operation in Plan

ORACLE

ORACLE

REAL-WORLD PERFORMANCE

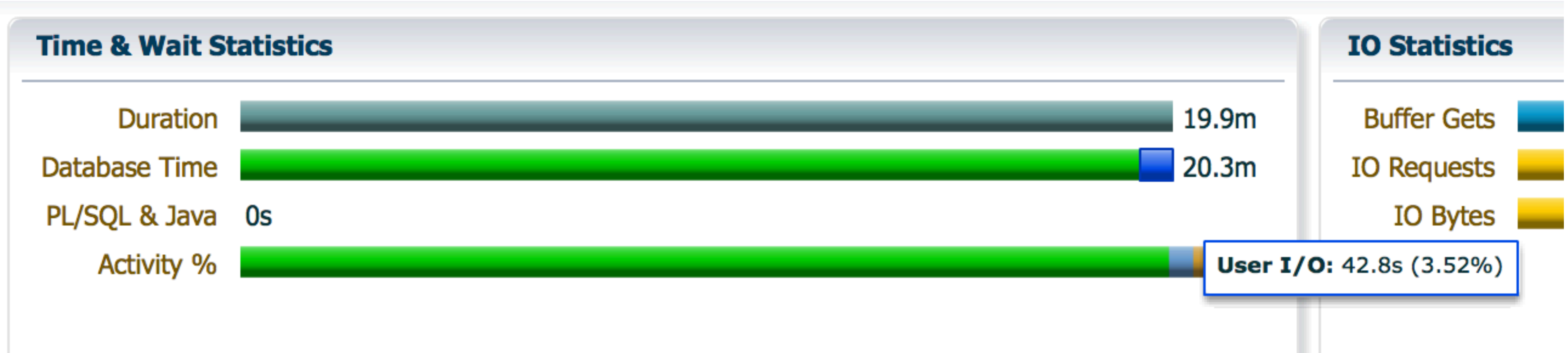
# Why Table Scans?

## Some History and Math

- Our query joins 4 dimension tables to a 1-billion row fact table
- Would you expect this query to be I/O bound?
- If we use indexes:
  - First join to fact table retrieves 42 million rows, or 4.2%
  - After completing additional joins, we end up doing ~ 320 million random reads
- 320m random reads @5ms/read =~ 1600 seconds
  - Is 5ms for a random I/O an "old tech number"?
  - If the "new tech number number" is 1ms, we're looking at ~5 minutes for I/O

# Why Table Scans?

## Index Access



- We spend 43 *seconds* on I/O, not 26 minutes or 5 minutes
- Our average random read is taking a fraction of a millisecond
- We're CPU-bound, not I/O bound

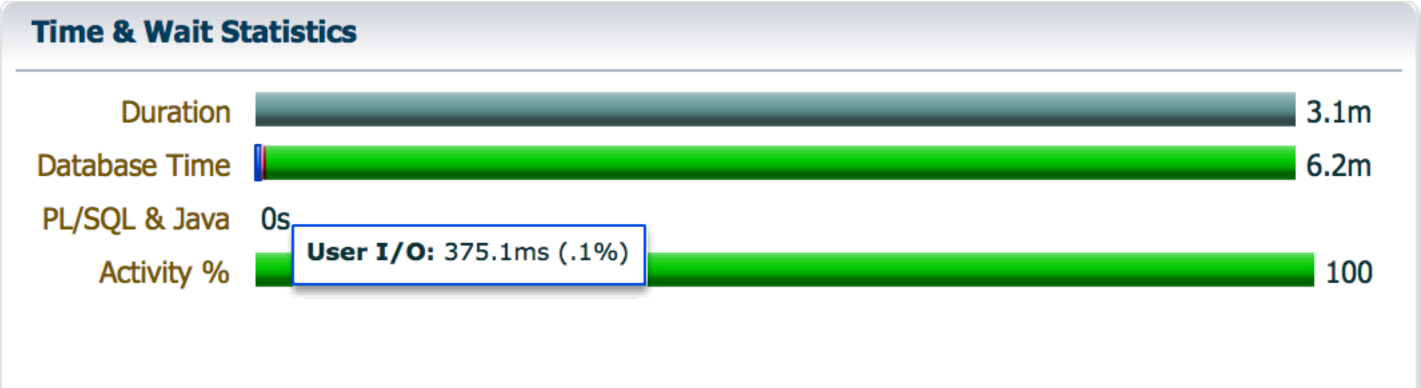
# Why Table Scans?

## Full Table Scans

- What about a full table scan?
- We have about 14 million blocks
- The “old tech number” for multi-block reads is about 6 or 7ms per MBR
- A multi-block read count of 128 = 109k multi-block reads
- 109k multi-block reads at 6.5ms per MBR means we’d spend under a second doing I/O
- Let’s see …

# Why Table Scans?

## Full Table Scans



**Time & Wait Statistics**

Duration 3.1m

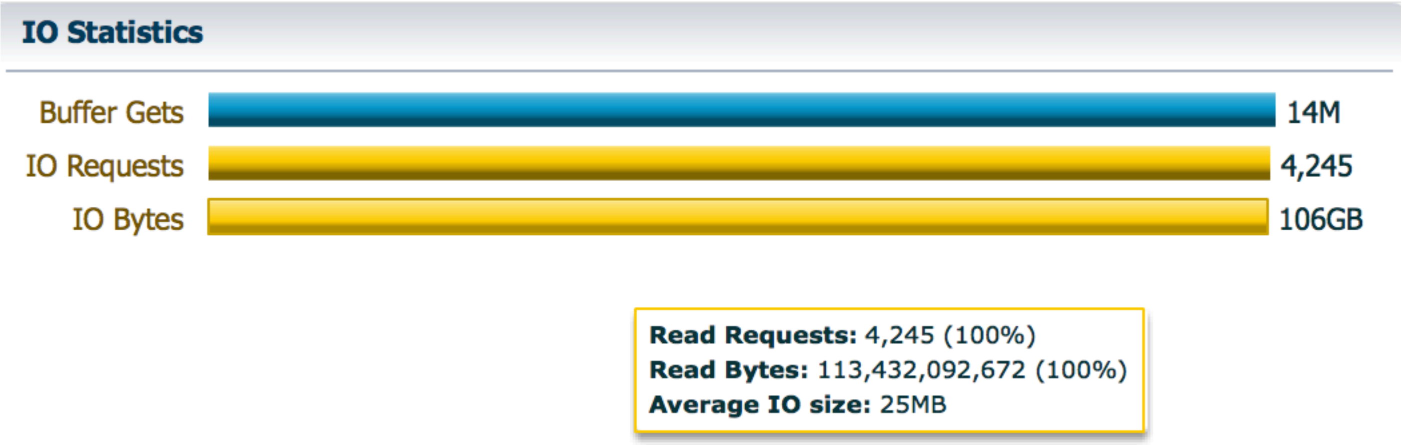
Database Time 6.2m

PL/SQL & Java 0s

Activity % 100

**User I/O:** 375.1ms (.1%)

- We 375ms doing I/O with an average I/O size of 25MB
- “Effective” MBRC much higher than 128
- We’re still CPU-bound



**IO Statistics**

Buffer Gets 14M

IO Requests 4,245

IO Bytes 106GB

**Read Requests:** 4,245 (100%)
**Read Bytes:** 113,432,092,672 (100%)
**Average IO size:** 25MB

# Why Table Scans?

## The New Math &  What We've Learned

- In Oracle's Cloud, random and sequential reads are much faster than the old numbers people think about
- For both index and table scan access, the queries are CPU-bound, not I/O bound
- In this case, scans were 6x faster.  Was this because of scans, joins, aggregation, or something else?
- Time to dig a bit deeper!

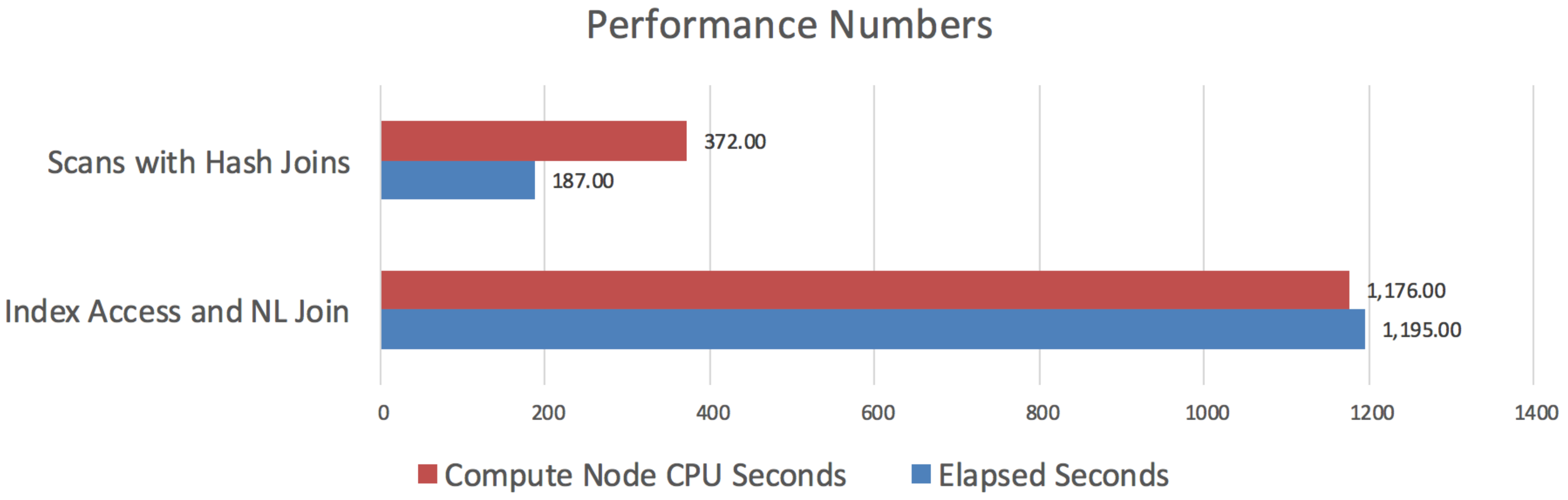
# Why Table Scans?

## Leverage Matrix

| Method | % of Time in Data Acquisition | % of Time in Joins | % of Time in Sort/Aggregate | Elapsed seconds |
|---|---|---|---|---|
| Scans with Hash Joins | 86% | 12.24% | 1.60% | 187 |
| Index access and NL Join | 99% | .08% | .17% | 1,195 |

# Why Table Scans?

## Numbers So Far



# Agenda

1. Why Table Scans?
2. Making Scans Smaller
3. Where’s Our Leverage?
4. Rearranging the Data
5. Things We Can Do to Speed Up Next Operation in Plan

ORACLE®

ORACLE® REAL-WORLD PERFORMANCE

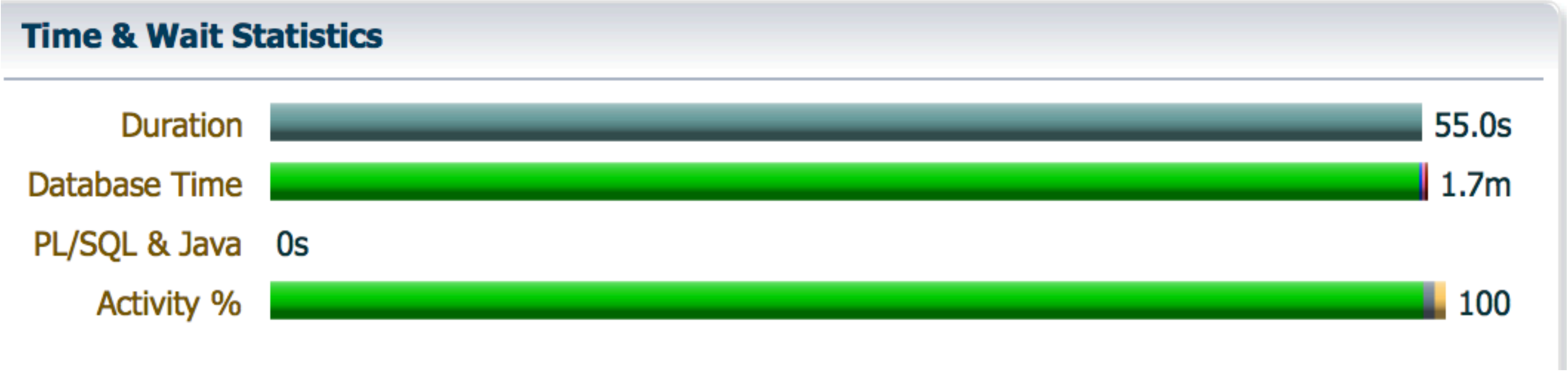
# Making Scans Smaller

## Partitioning and Compression

- Partitioning is a means to prune data and reduce I/O & CPU
- Compression is a means to reduce size of data on disk and reduce I/O
- Oracle Cloud Infrastructure supports Hybrid Columnar Compression
- Let's try it out

# Making Scans Smaller

## Partitioning



Time & Wait Statistics

Duration 55.0s

Database Time 1.7m

PL/SQL & Java 0s

Activity % 100

**Still CPU-bound, but uses a lot less CPU than scans without partitioning**

# Making Scans Smaller

## Partitioning

Table Scans *without* partitioning

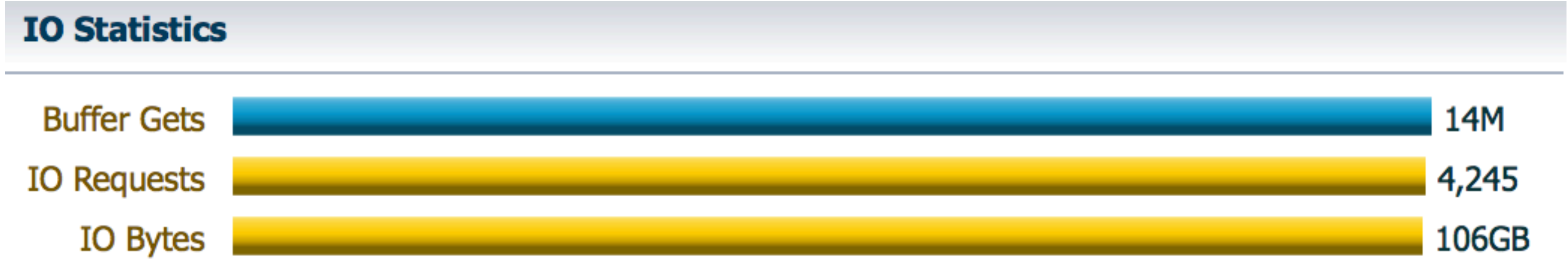
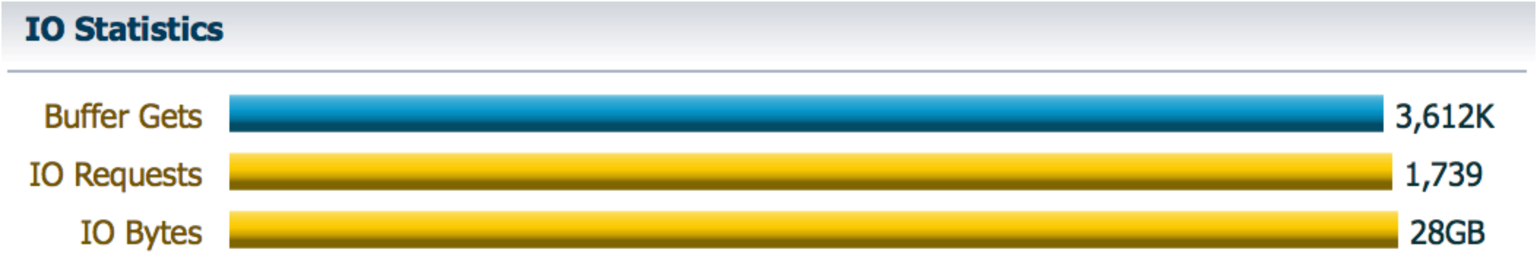


Table Scans *with* partitioning



We partitioned on date join key and our predicates filtered ¾ of the data

**Although CPU-bound, I/O bytes is a proxy for CPU consumption**

# Making Scans Smaller

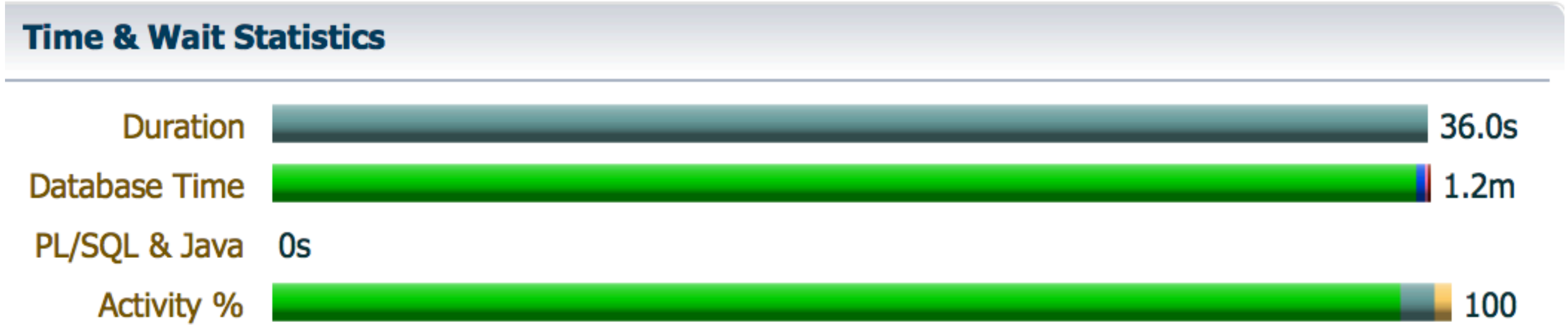
## Partitioning Warnings

- A common problem we see is over-partitioning
  - Proxy for indexes?
  - Used to avoid contention?
- Too many partitions can cause many problems:
  - Excessive time during parse & execute
  - High metadata cost
  - DDL more expensive due to data dictionary overheads
  - Exacerbated with RAC
  - Problems could reveal themselves in non-obvious ways

# Making Scans Smaller

## HCC Compression with Scans and Partitioning, Oracle Database Cloud Service

**Query is CPU-bound again but only ran marginally faster**



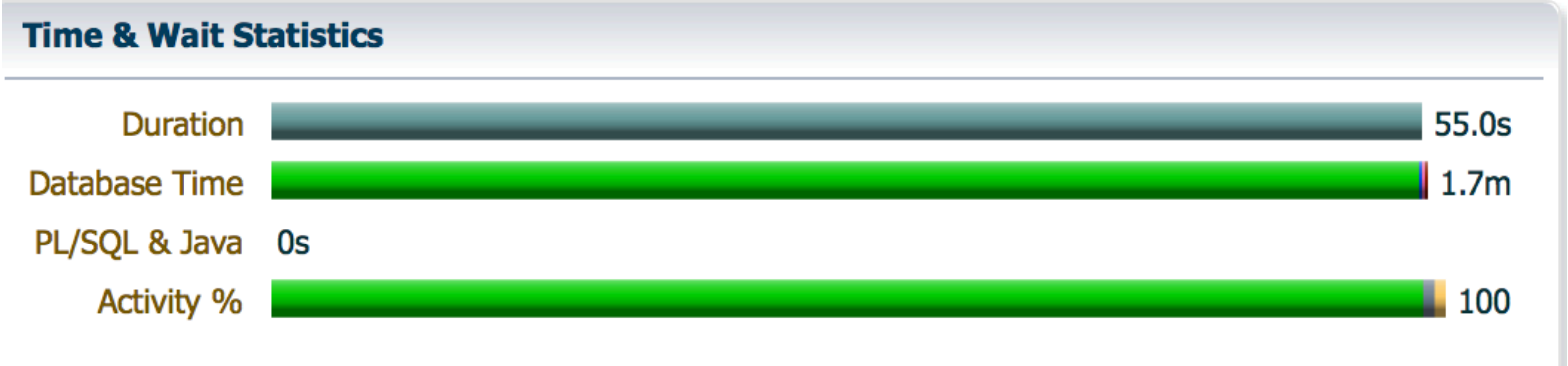
Time & Wait Statistics

Duration 36.0s

Database Time 1.2m

PL/SQL & Java 0s

Activity % 100



Time & Wait Statistics

Duration 55.0s

Database Time 1.7m

PL/SQL & Java 0s

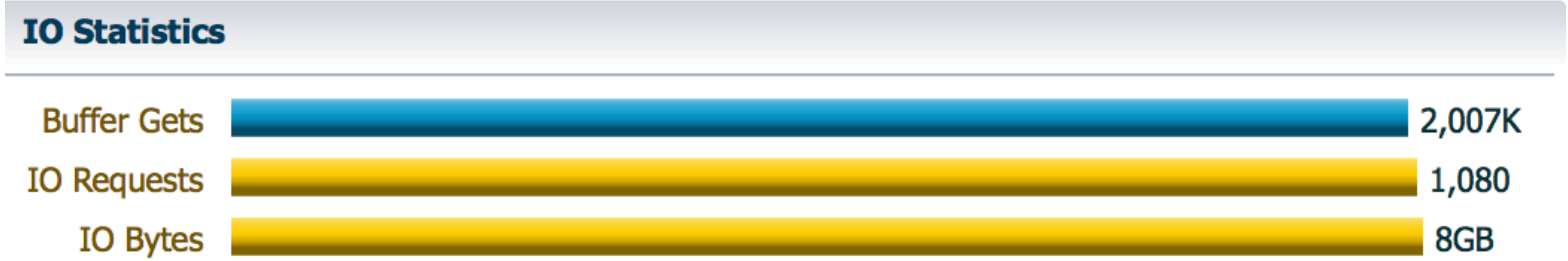
Activity % 100

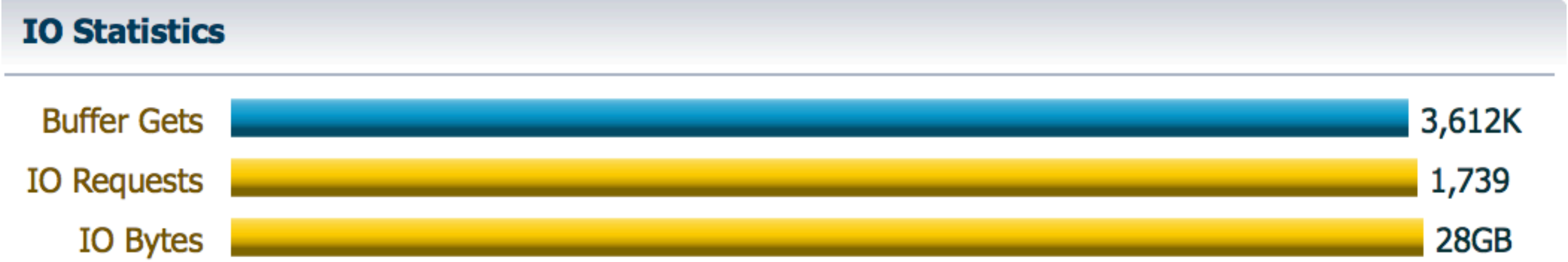
***Without*** **HCC**

# Making Scans Smaller

## HCC Compression with Scans and Partitioning, Oracle Database Cloud Service

**We scanned 3.5x less data but only improved query performance by 35%**





***Without*** **HCC**

# Making Scans Smaller

## HCC Compression with Scans and Partitioning, Oracle Database Cloud Service

- We weren't I/O bound to begin with, we were CPU-bound
- CPU & time to parse HCC blocks less than time to parse uncompressed blocks, but …
- We need CPU to decompress compressed data
- Querying smaller datasets doesn't yield linear performance gains
- Conventional mindset vs. modern capabilities

ORACLE REAL-WORLD PERFORMANCE

# Making Scans Smaller

## Leverage Matrix

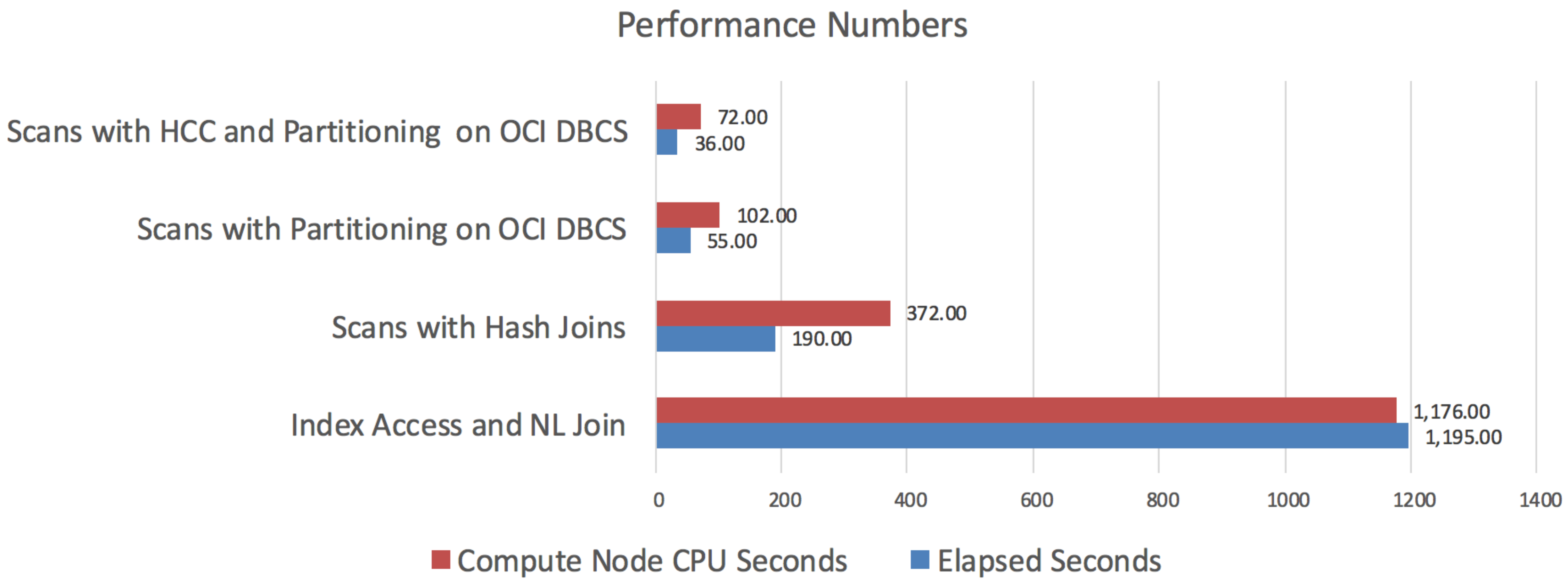
| Method | % of Time in Data Acquisition | % of Time in Joins | % of Time in Sort/Aggregate | Elapsed Seconds |
|---|---|---|---|---|
| **Partitioning with Compression (OCI DBCS)** | **81%** | **14.5%** | **5.8%** | **36** |
| **Partitioning on (OCI DBCS)** | **79%** | **17%** | **4%** | **55** |
| Scans with Hash Joins (OCI DBCS) | 86% | 12.24% | 1.60% | 187 |
| Index access and NL Join (OCI DBCS) | 99% | .08% | .17% | 1,195 |

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

# Making Scans Smaller

## Numbers So Far



# Agenda

1. Why Table Scans?
2. Making Scans Smaller
3. Where’s Our Leverage?
4. Rearranging the Data
5. Things We Can Do to Speed Up Next Operations` in Plan

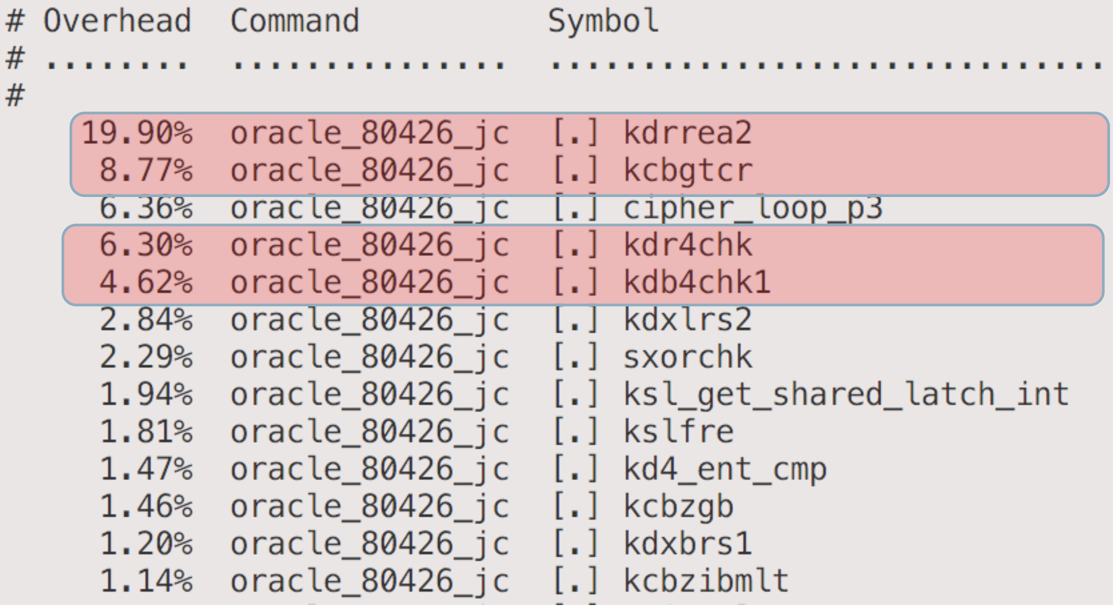
ORACLE

ORACLE REAL-WORLD PERFORMANCE

# Where’s Our Leverage?

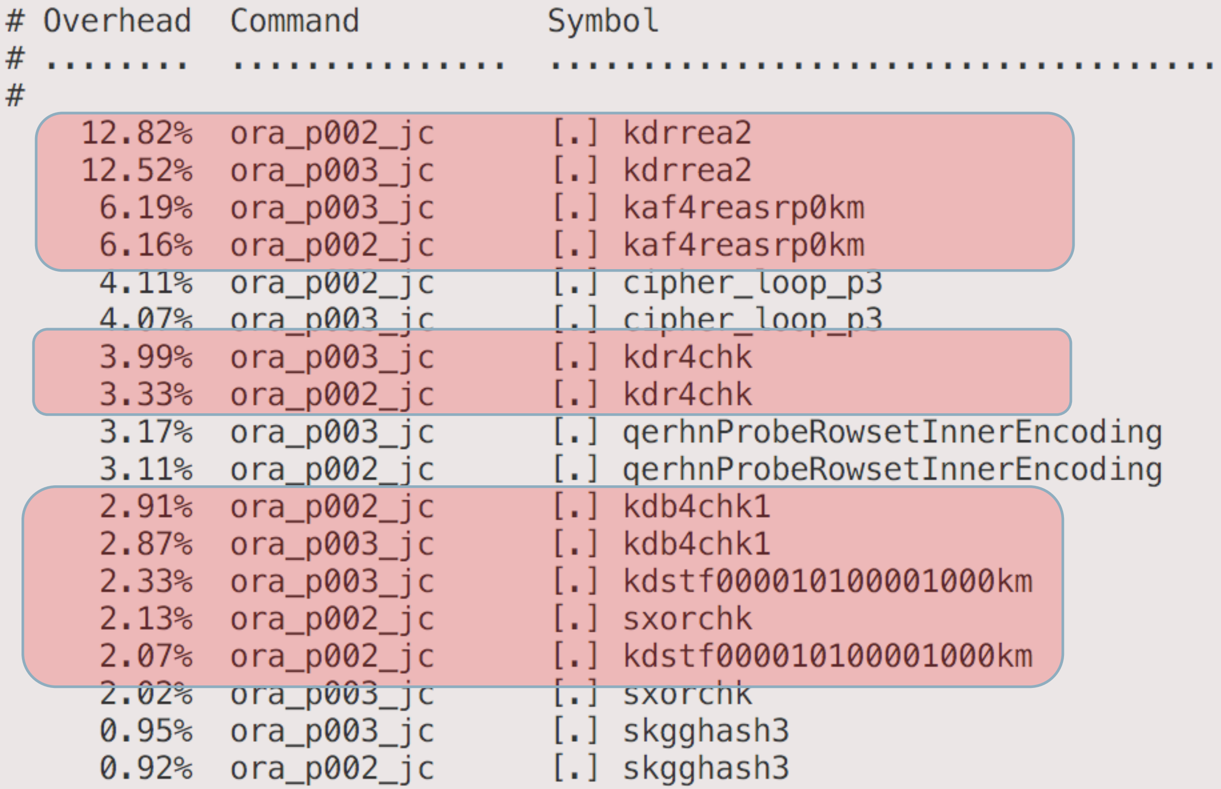
## Some Profiling Data

### Index Scans



```
# Overhead  Command          Symbol
# ........  ...............  ..............................
#
   19.90%  oracle_80426_jc  [.] kdrrea2
    8.77%  oracle_80426_jc  [.] kcbgtcr
    6.36%  oracle_80426_jc  [.] cipher_loop_p3
    6.30%  oracle_80426_jc  [.] kdr4chk
    4.62%  oracle_80426_jc  [.] kdb4chk1
    2.84%  oracle_80426_jc  [.] kdxlrs2
    2.29%  oracle_80426_jc  [.] sxorchk
    1.94%  oracle_80426_jc  [.] ksl_get_shared_latch_int
    1.81%  oracle_80426_jc  [.] kslfre
    1.47%  oracle_80426_jc  [.] kd4_ent_cmp
    1.46%  oracle_80426_jc  [.] kcbzgb
    1.20%  oracle_80426_jc  [.] kdxbrs1
    1.14%  oracle_80426_jc  [.] kcbzibmlt
```

### Table Scans

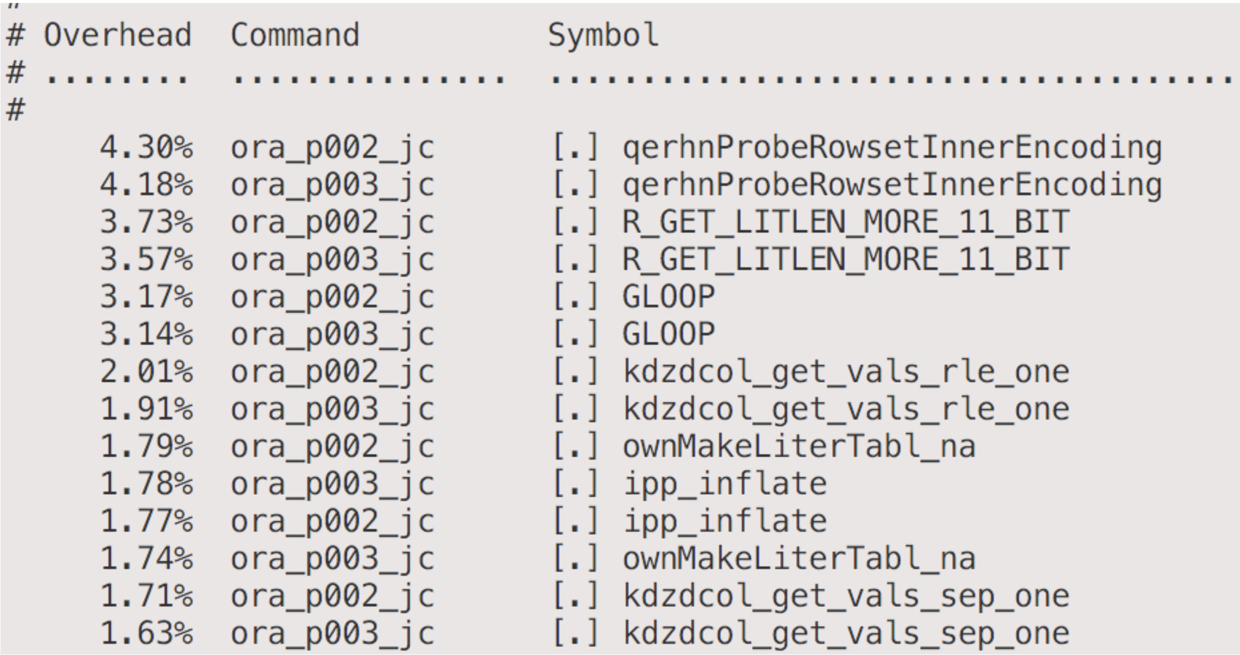


```
# Overhead  Command          Symbol
# ........  ...............  ...................................
#
    12.82%  ora_p002_jc      [.] kdrrea2
    12.52%  ora_p003_jc      [.] kdrrea2
     6.19%  ora_p003_jc      [.] kaf4reasrp0km
     6.16%  ora_p002_jc      [.] kaf4reasrp0km
     4.11%  ora_p002_jc      [.] cipher_loop_p3
     4.07%  ora_p003_jc      [.] cipher_loop_p3
     3.99%  ora_p003_jc      [.] kdr4chk
     3.33%  ora_p002_jc      [.] kdr4chk
     3.17%  ora_p003_jc      [.] qerhnProbeRowsetInnerEncoding
     3.11%  ora_p002_jc      [.] qerhnProbeRowsetInnerEncoding
     2.91%  ora_p002_jc      [.] kdb4chk1
     2.87%  ora_p003_jc      [.] kdb4chk1
     2.33%  ora_p003_jc      [.] kdstf000010100001000km
     2.13%  ora_p002_jc      [.] sxorchk
     2.07%  ora_p002_jc      [.] kdstf000010100001000km
     2.02%  ora_p003_jc      [.] sxorchk
     0.95%  ora_p003_jc      [.] skgghash3
     0.92%  ora_p002_jc      [.] skgghash3
```

**Most of our time is in parsing rows/blocks**

# Where’s Our Leverage?

## Profiling Data with HCC and Partitioning on Oracle Database Cloud Service



```
#
# Overhead  Command           Symbol
# ........  ................  ....................................
#
     4.30%  ora_p002_jc       [.] qerhnProbeRowsetInnerEncoding
     4.18%  ora_p003_jc       [.] qerhnProbeRowsetInnerEncoding
     3.73%  ora_p002_jc       [.] R_GET_LITLEN_MORE_11_BIT
     3.57%  ora_p003_jc       [.] R_GET_LITLEN_MORE_11_BIT
     3.17%  ora_p002_jc       [.] GLOOP
     3.14%  ora_p003_jc       [.] GLOOP
     2.01%  ora_p002_jc       [.] kdzdcol_get_vals_rle_one
     1.91%  ora_p003_jc       [.] kdzdcol_get_vals_rle_one
     1.79%  ora_p002_jc       [.] ownMakeLiterTabl_na
     1.78%  ora_p003_jc       [.] ipp_inflate
     1.77%  ora_p002_jc       [.] ipp_inflate
     1.74%  ora_p003_jc       [.] ownMakeLiterTabl_na
     1.71%  ora_p002_jc       [.] kdzdcol_get_vals_sep_one
     1.63%  ora_p003_jc       [.] kdzdcol_get_vals_sep_one
```

**Time spent parsing columns/rows in HCC format**

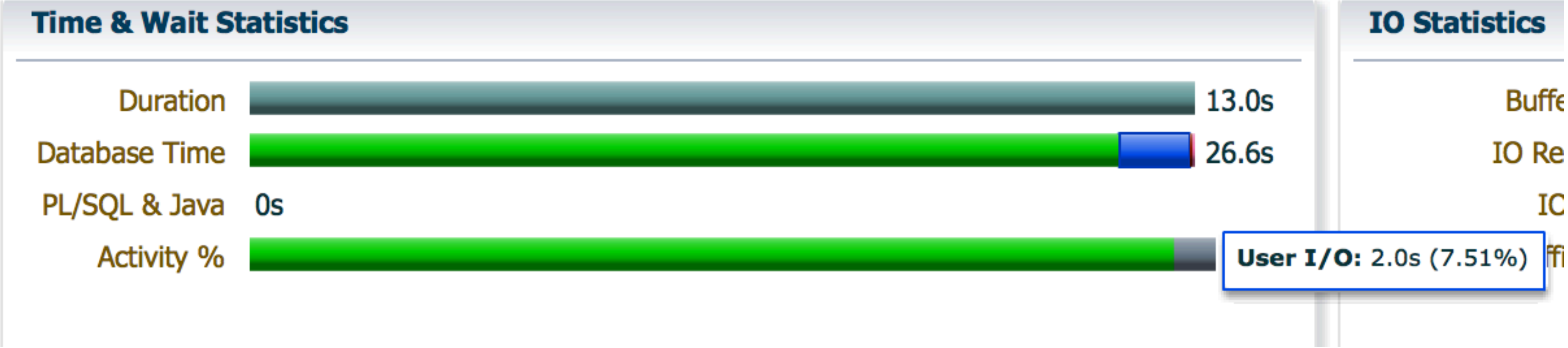
# Where's Our Leverage?

## Block Parsing

- Most of our time is being spent on data acquisition
- Data acquisition is CPU-bound on Oracle DB Cloud Service
- Profiling shows it's largely related to parsing blocks
- What if we could recruit more resources for block parsing, parse blocks in parallel, offload this work to different machines, and decrease wall clock time?
- Do we have any technology that does this?

# Where's Our Leverage?

## Exadata Cloud Service



**Time & Wait Statistics**

Duration 13.0s

Database Time 26.6s

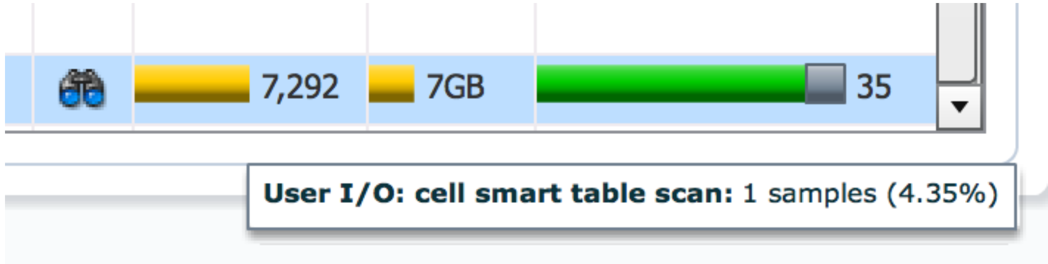
PL/SQL & Java 0s

Activity %

**User I/O:** 2.0s (7.51%)

**IO Statistics**

- **Query still CPU-bound but we see 7.5% (2 seconds) on I/O**
- **Remember we saw no I/O on DBCS**



7,292 7GB 35

**User I/O: cell smart table scan:** 1 samples (4.35%)

- **I/O in this case means anything in the I/O path, including CPU on storage cells. 4.35% of our time is on "smart scan"**

# Where’s Our Leverage?

## Exadata Cloud Service to Offload Block Parsing

- 4.35% of 13 seconds =~ .5 seconds, multiply by 2 slaves =~ **1** sec on cells
- On **Exadata** we use up to 10 parallel requests per slave
  - 2 slaves = up 20 parallel requests per cell
  - 7 cells =~ **140** parallel requests in total, which is **70x** more than DWCS
- In BMC we spend =~ **70** CPU seconds on scan
- Offloading allows us to parse blocks in parallel, reduce elapsed time, and reduce compute node CPU time
- Bonus question – if we do 10 requests per slave with 1MB I/O size, what is should our minimum partition size be?

REAL-WORLD PERFORMANCE

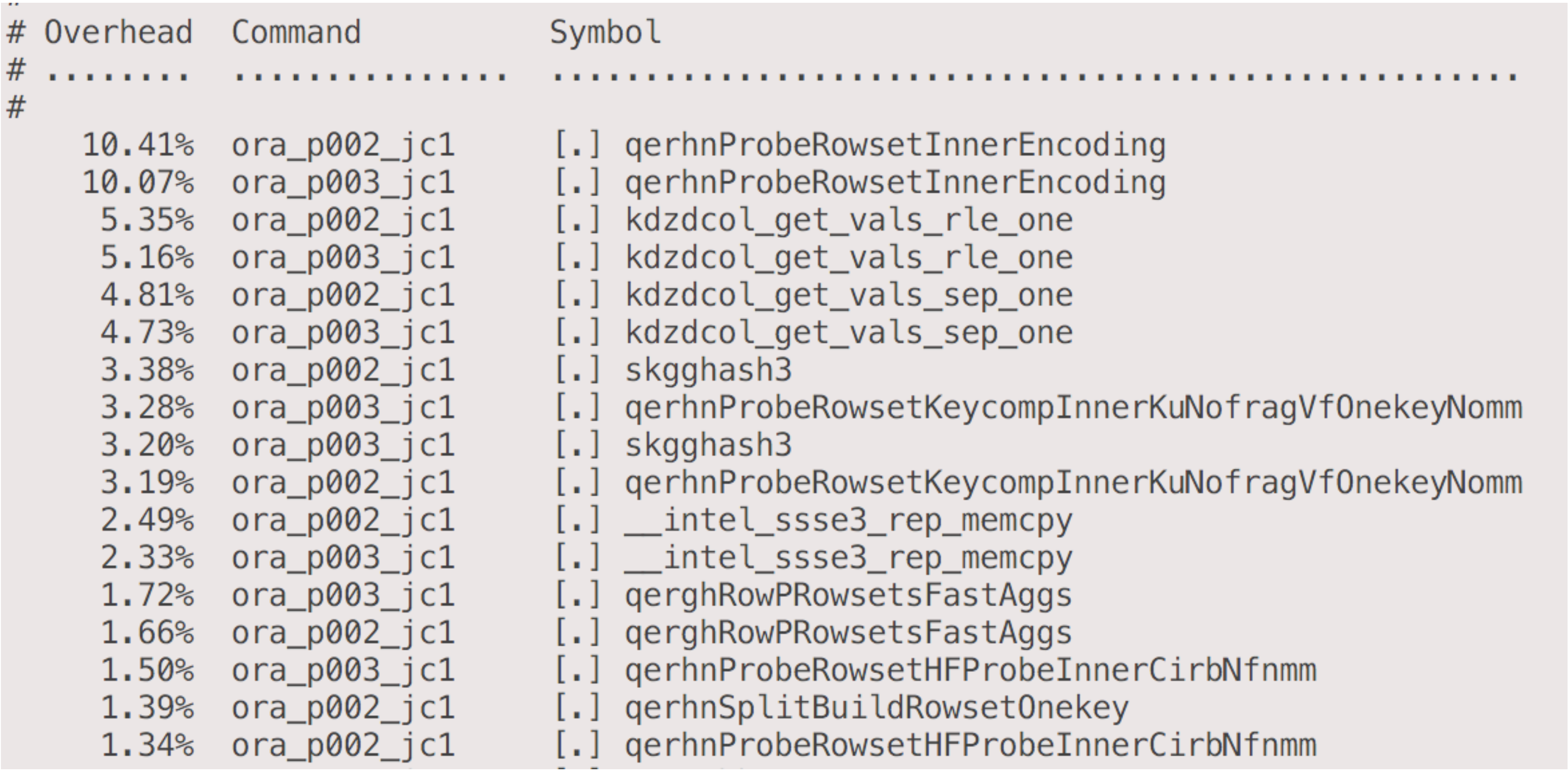
# Where's Our Leverage?

## Exadata Cloud Service to Offload Block Parsing

Did you notice we're still spending 24.5 CPU seconds *not doing* the scans?

# Where’s Our Leverage?

## Exadata Cloud Service to Offload Block Parsing Compute node Profiling



```
# Overhead  Command          Symbol
# ........  ...............  ....................................................
#
    10.41%  ora_p002_jc1     [.] qerhnProbeRowsetInnerEncoding
    10.07%  ora_p003_jc1     [.] qerhnProbeRowsetInnerEncoding
     5.35%  ora_p002_jc1     [.] kdzdcol_get_vals_rle_one
     5.16%  ora_p003_jc1     [.] kdzdcol_get_vals_rle_one
     4.81%  ora_p002_jc1     [.] kdzdcol_get_vals_sep_one
     4.73%  ora_p003_jc1     [.] kdzdcol_get_vals_sep_one
     3.38%  ora_p002_jc1     [.] skgghash3
     3.28%  ora_p003_jc1     [.] qerhnProbeRowsetKeycompInnerKuNofragVf0nekeyNomm
     3.20%  ora_p003_jc1     [.] skgghash3
     3.19%  ora_p002_jc1     [.] qerhnProbeRowsetKeycompInnerKuNofragVf0nekeyNomm
     2.49%  ora_p002_jc1     [.] __intel_ssse3_rep_memcpy
     2.33%  ora_p003_jc1     [.] __intel_ssse3_rep_memcpy
     1.72%  ora_p003_jc1     [.] qerghRowPRowsetsFastAggs
     1.66%  ora_p002_jc1     [.] qerghRowPRowsetsFastAggs
     1.50%  ora_p003_jc1     [.] qerhnProbeRowsetHFProbeInnerCirbNfnmm
     1.39%  ora_p002_jc1     [.] qerhnSplitBuildRowsetOnekey
     1.34%  ora_p002_jc1     [.] qerhnProbeRowsetHFProbeInnerCirbNfnmm
```

**Time on compute node no longer dominated by parsing blocks**

# Where’s Our Leverage?

## Leverage Matrix

| Method | % of Time in Data Acquisition | % of Time in Joins | % of Time in Sort/Aggregate | Elapsed Seconds |
|---|---|---|---|---|
| **Partitioning with Compression (OCI ExaCS)** | **48%** | **51%** | **1%** | **13** |
| Partitioning with Compression (OCI DBCS) | 81% | 14.5% | 5.8% | 36 |
| Partitioning on BMC (OCI DBCS) | 79% | 17% | 4% | 55 |
| Scans with Hash Joins (OCI DBCS) | 86% | 12.24% | 1.60% | 187 |
| Index access and NL Join (OCI DBCS) | 99% | .08% | .17% | 1,195 |

# Where’s Our Leverage?

## More You Can Do on Exadata

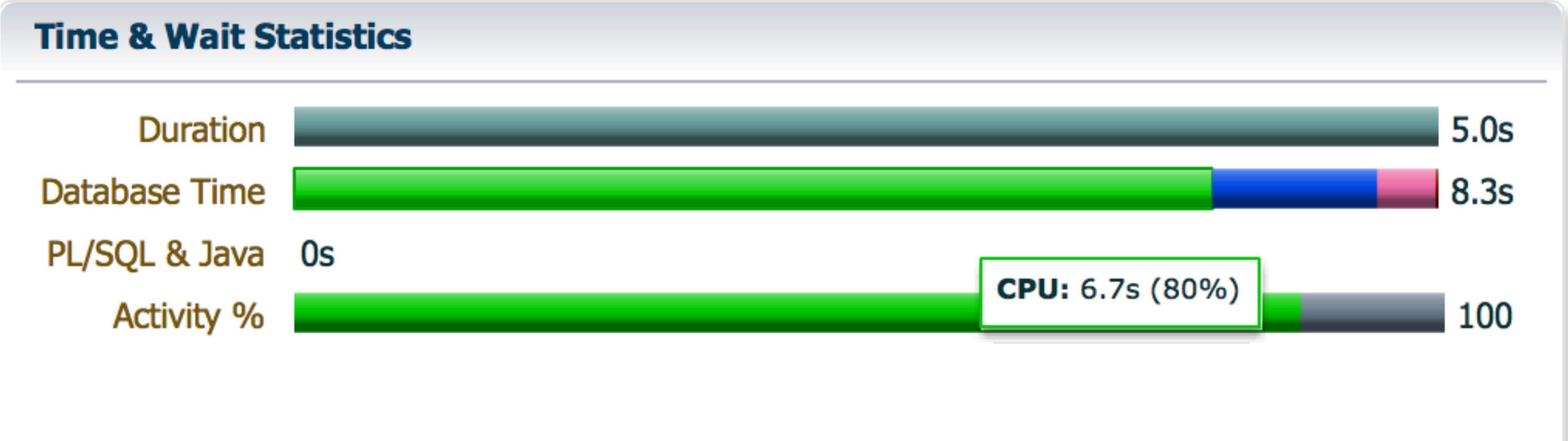
- Exadata provides some other interesting alternatives to explore
- Zone Maps with Attribute Clustering provide and additional means to prune I/O and reduce CPU
- In addition to partitioning on our date dimension’s join key, let’s implement Attribute Clustering with a Zone Map on our Supplier dimension’s join key

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

# Where’s Our Leverage?

## Exadata Cloud Service with Clustering and Zone Maps


Time & Wait Statistics

| | |
|---|---|
| Duration | 5.0s |
| Database Time | 8.3s |
| PL/SQL & Java | 0s |
| Activity % | 100 |

CPU: 6.7s (80%)

- Elapsed time reduced from 13 seconds to 5 seconds
- Compute node CPU reduced from 24.5 to 6.7 seconds
- Like partitioning, Zone Maps with Clustering means fewer calls to Exadata, with each call being more “row-rich”

ORACLE

ORACLE® REAL-WORLD PERFORMANCE

# Where’s Our Leverage?

## Leverage Matrix

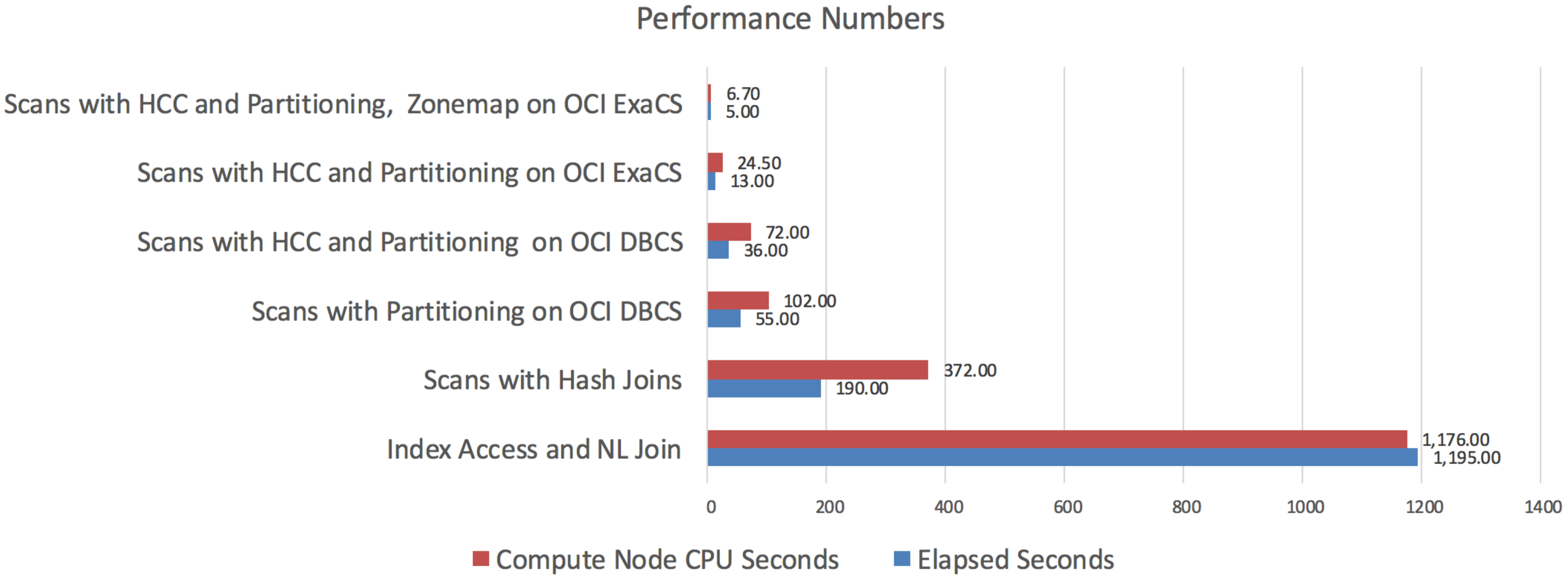
| Method | % of Time in Data Acquisition | % of Time in Joins | % of Time in Sort/Aggregate | Elapsed Seconds |
|---|---|---|---|---|
| **Partitioning with HCC & Zone Maps (OCI ExaCS)** | **57%** | **29%** | **14%** | **5** |
| **Partitioning with HCC (OCI ExaCS)** | **48%** | **51%** | **1%** | **13** |
| Partitioning with Compression (OCI DBCS) | 81% | 14.5% | 5.8% | 36 |
| Partitioning (OCI DBCS) | 79% | 17% | 4% | 55 |
| Scans with Hash Joins (OCI DBCS) | 86% | 12.24% | 1.60% | 187 |
| Index access and NL Join (OCI DBCS) | 99% | .08% | .17% | 1,195 |

ORACLE

ORACLE
REAL-WORLD PERFORMANCE

# Where’s Our Leverage?

## Numbers So Far



# Agenda

1. Why Table Scans?
2. Making Scans Smaller
3. Where’s Our Leverage?
4. Rearranging the Data
5. Things We Can Do to Speed Up Next Operation in Plan

ORACLE

ORACLE REAL-WORLD PERFORMANCE

# Rearranging the Data

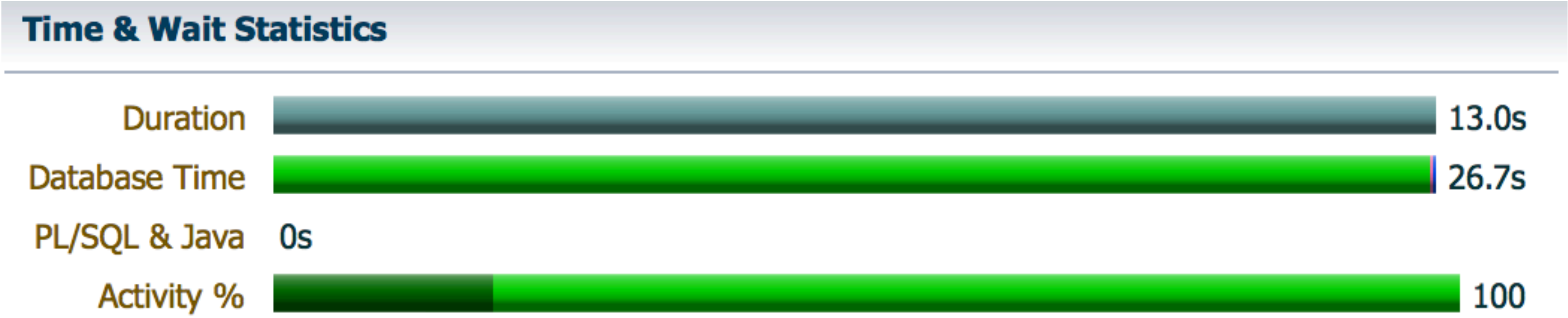
## In-Memory Columnar

- We’ve demonstrated that parsing blocks consumes CPU and contributes to query elapsed time during scans
- Offloading to Exadata provides us more CPUs to parse blocks
- How would In-Memory Columnar representation impact our results?
- Let’s test with Database In-Memory

# Rearranging the Data

## In-Memory Columnar

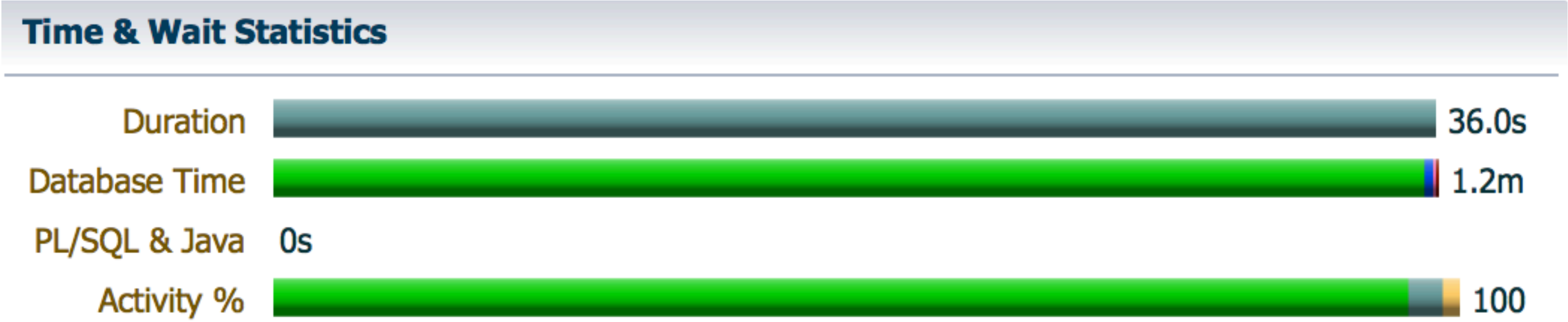
Scan with Partitioning and In-Memory (DB Cloud Service)



### Time & Wait Statistics

| | |
|---|---|
| Duration | 13.0s |
| Database Time | 26.7s |
| PL/SQL & Java | 0s |
| Activity % | 100 |

Scan with Partitioning (DB Cloud Service, row format)



### Time & Wait Statistics

| | |
|---|---|
| Duration | 36.0s |
| Database Time | 1.2m |
| PL/SQL & Java | 0s |
| Activity % | 100 |

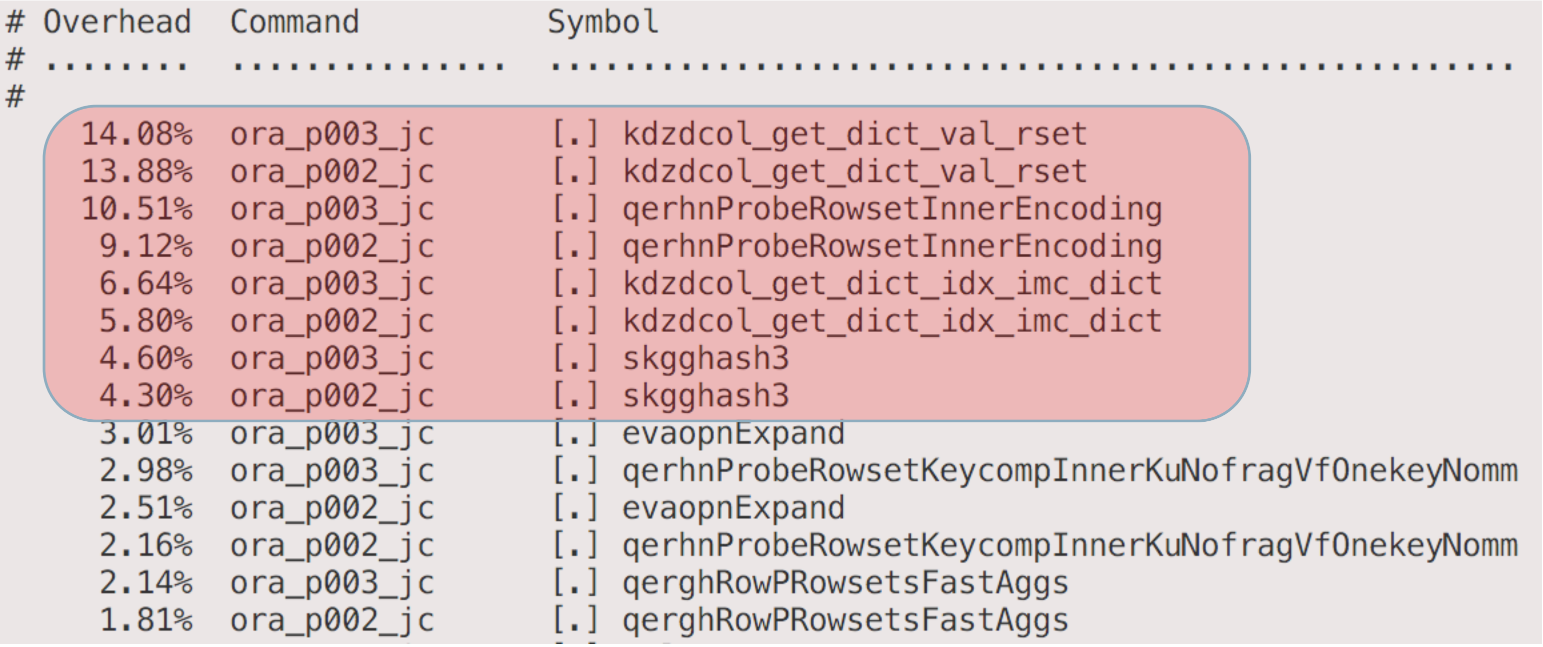
ORACLE

ORACLE
REAL-WORLD PERFORMANCE

# Rearranging the Data

## In-Memory Columnar

**Scan with Partitioning and DBIM**



```
# Overhead  Command          Symbol
# ........  ...............  ..................................................
#
    14.08%  ora_p003_jc      [.] kdzdcol_get_dict_val_rset
    13.88%  ora_p002_jc      [.] kdzdcol_get_dict_val_rset
    10.51%  ora_p003_jc      [.] qerhnProbeRowsetInnerEncoding
     9.12%  ora_p002_jc      [.] qerhnProbeRowsetInnerEncoding
     6.64%  ora_p003_jc      [.] kdzdcol_get_dict_idx_imc_dict
     5.80%  ora_p002_jc      [.] kdzdcol_get_dict_idx_imc_dict
     4.60%  ora_p003_jc      [.] skgghash3
     4.30%  ora_p002_jc      [.] skgghash3
     3.01%  ora_p003_jc      [.] evaopnExpand
     2.98%  ora_p003_jc      [.] qerhnProbeRowsetKeycompInnerKuNofragVf0nekeyNomm
     2.51%  ora_p002_jc      [.] evaopnExpand
     2.16%  ora_p002_jc      [.] qerhnProbeRowsetKeycompInnerKuNofragVf0nekeyNomm
     2.14%  ora_p003_jc      [.] qerghRowPRowsetsFastAggs
     1.81%  ora_p002_jc      [.] qerghRowPRowsetsFastAggs
```

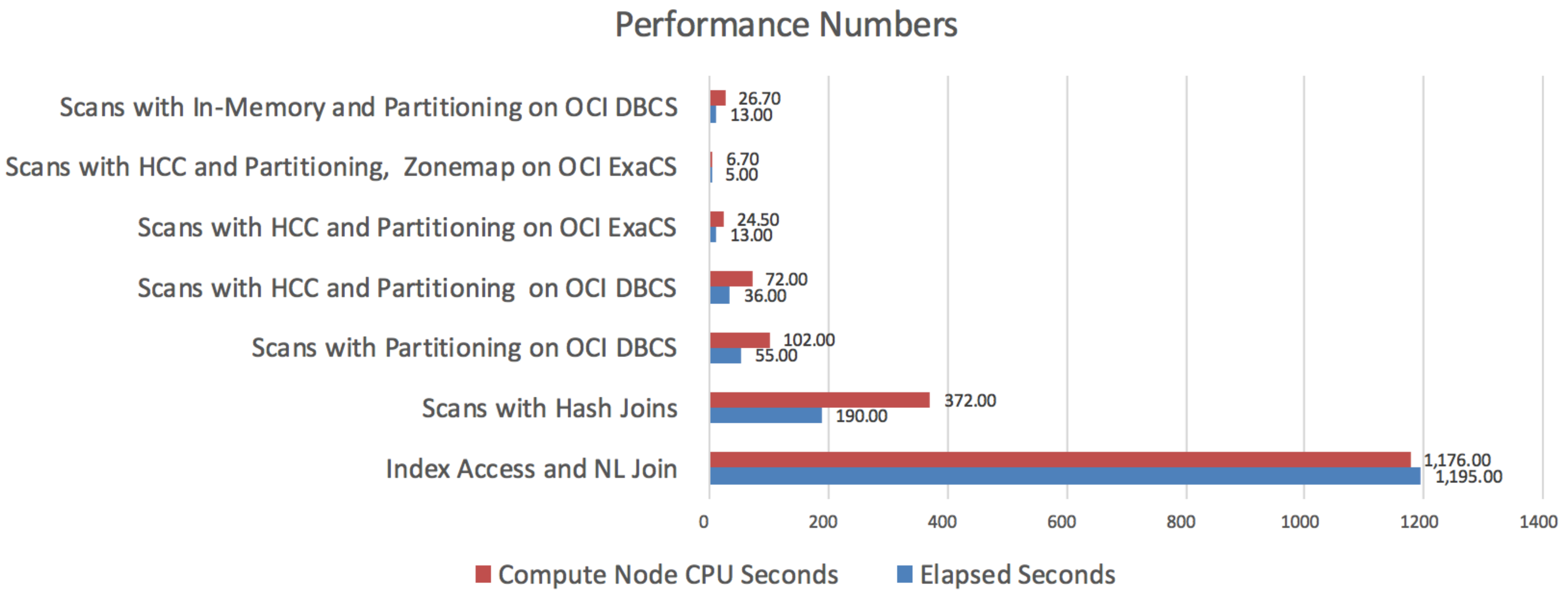
# Rearranging the Data

## In-Memory Columnar Leverage Matrix

| Method | % of Time in Data Acquisition | % of Time in Joins | % of Time in Sort/Aggregate | Elapsed Seconds |
|---|---|---|---|---|
| **DBIM with Partitioning (OCI DBCS)** | 18% | 64% | 18% | 13 |
| Partitioning with HCC & Zone Maps (OCI ExaCS) | 57% | 29% | 14% | 5 |
| Partitioning with HCC (OCI ExaCS) | 48% | 51% | 1% | 13 |
| Partitioning with Compression (OCI DBCS) | 81% | 14.5% | 5.8% | 36 |
| Partitioning (OCI DBCS) | 79% | 17% | 4% | 55 |
| Scans with Hash Joins (OCI DBCS) | 86% | 12.24% | 1.60% | 187 |
| Index access and NL Join (OCI DBCS) | 99% | .08% | .17% | 1,195 |

# Rearranging the Data

## Numbers So Far



ORACLE

ORACLE REAL-WORLD PERFORMANCE

# Agenda

1. Why Table Scans?
2. Making Scans Smaller
3. Where’s Our Leverage?
4. Rearranging the Data
5. Things We Can Do for Joins and Aggregation

# Things We Can Do for Joins and Aggregation

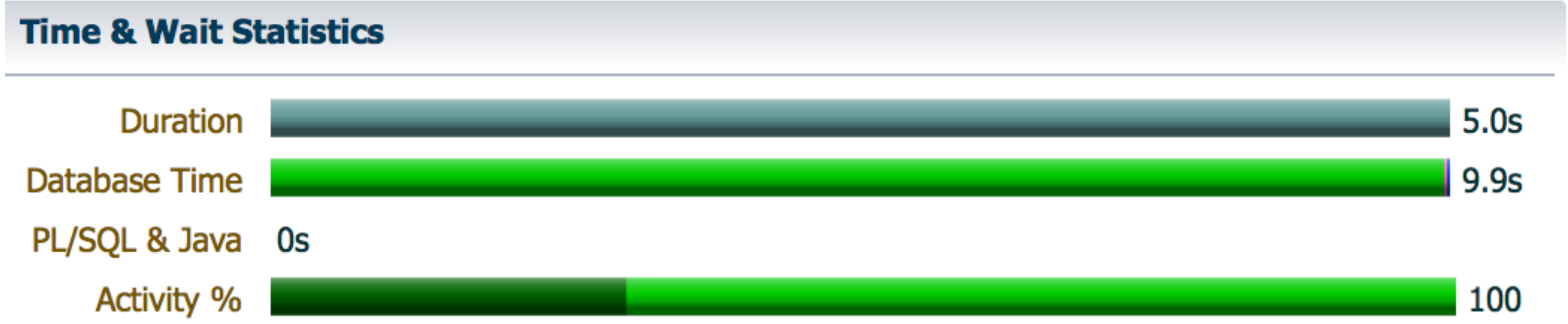
## Bloom Filters

- Bloom Filters provide means to efficiently filter data, reducing the volume of data for hash joins and distribution in subsequent plan steps
- Bloom Filter evaluation can be pushed down to Exadata so we can leverage storage cell CPUs
- Bloom Filter evaluation also pushed down to In-Memory column store and able to use different & more efficient algorithms
- Let’s test

# Things We Can Do for Joins ands Aggregation

## Bloom Filters

Scan with **Bloom Filters**, Partitioning and DBIM



**Time & Wait Statistics**

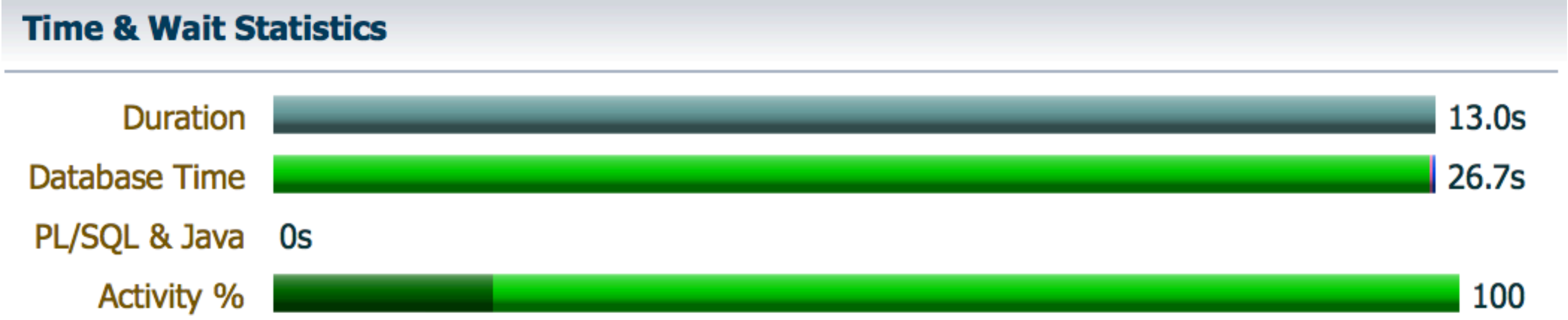
Duration 5.0s

Database Time 9.9s

PL/SQL & Java 0s

Activity % 100

Scan with Partitioning and DBIM



**Time & Wait Statistics**

Duration 13.0s

Database Time 26.7s

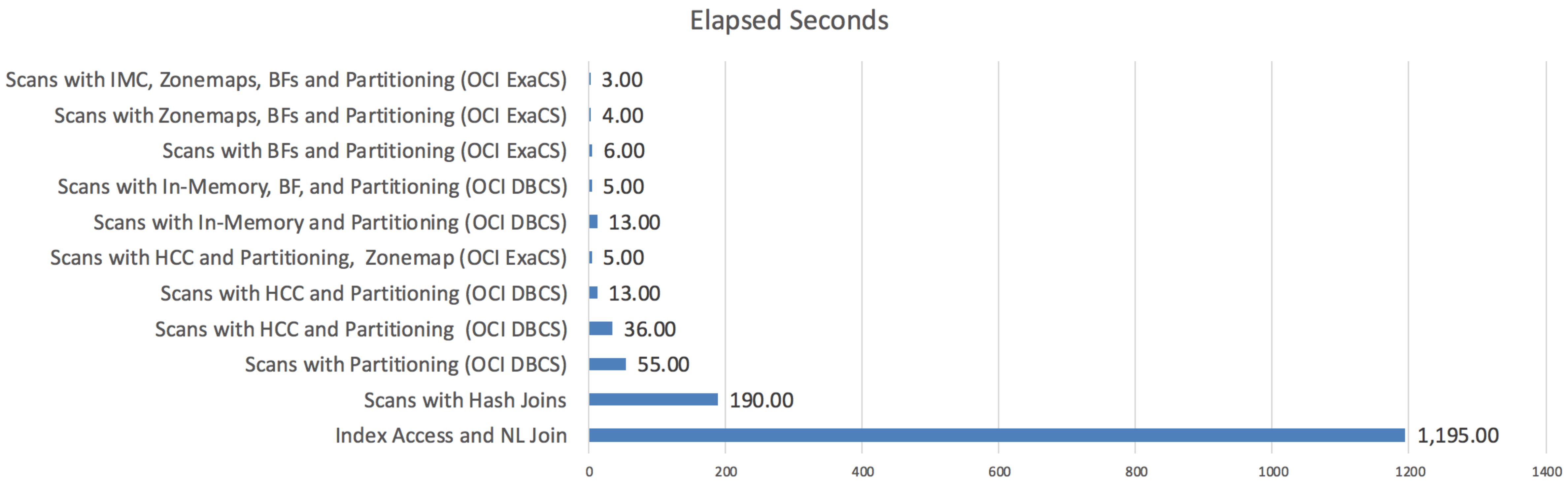
PL/SQL & Java 0s

Activity % 100

ORACLE REAL-WORLD PERFORMANCE

# Things We Can Do for Joins and Aggregation

## Numbers So Far with Bloom Filters on DBCS and ExaCS

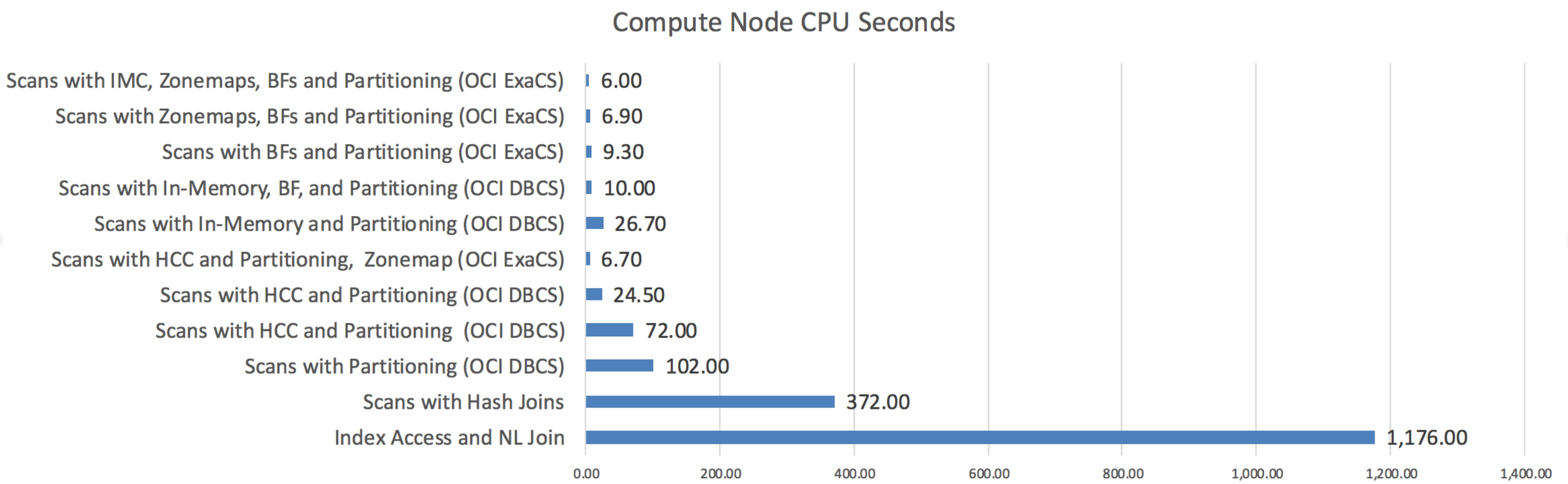


ORACLE

ORACLE
REAL-WORLD PERFORMANCE

# Things We Can Do for Joins and Aggregation

## Numbers So Far with Bloom Filters on DBCS and ExaCS



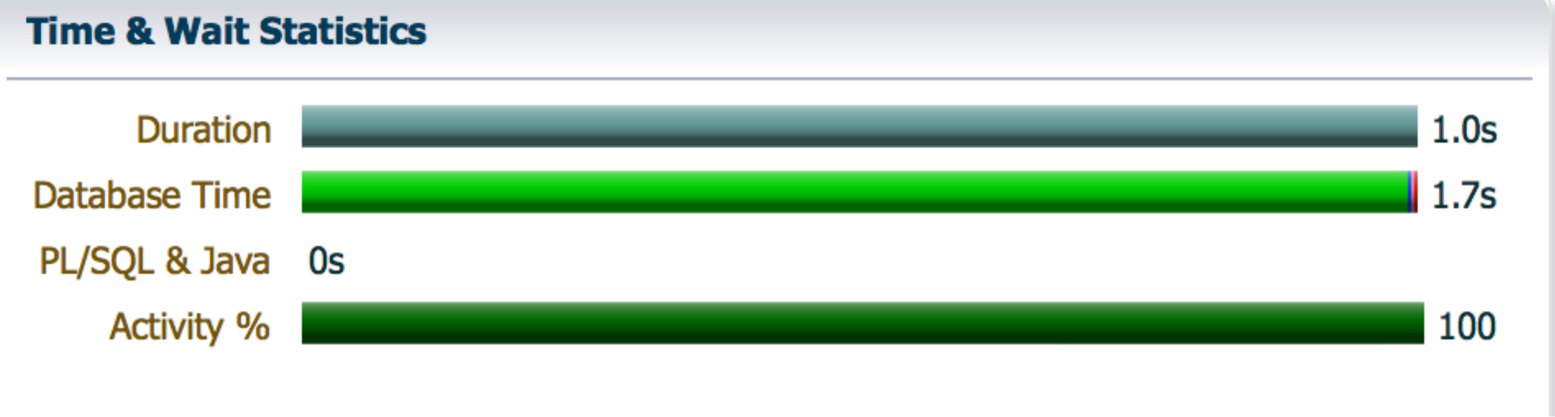
ORACLE

ORACLE
REAL-WORLD PERFORMANCE

# Things We Can Do for Joins and Aggregation

## In-Memory Aggregation

- Push down aggregation to scan
- *In-Memory Aggregation* performs aggregation during scan
- Let’s enable it and test



Time & Wait Statistics

Duration 1.0s

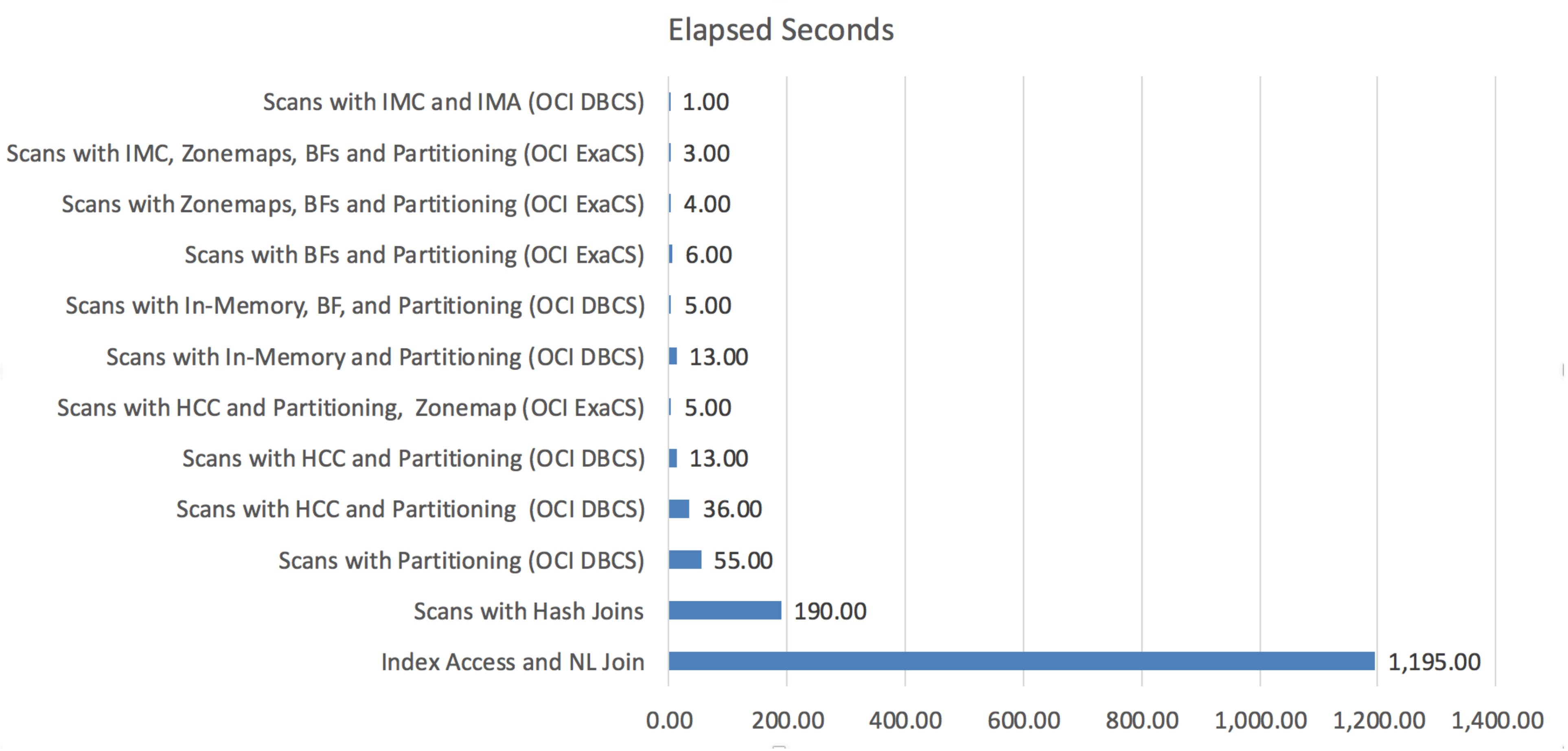
Database Time 1.7s

PL/SQL & Java 0s

Activity % 100

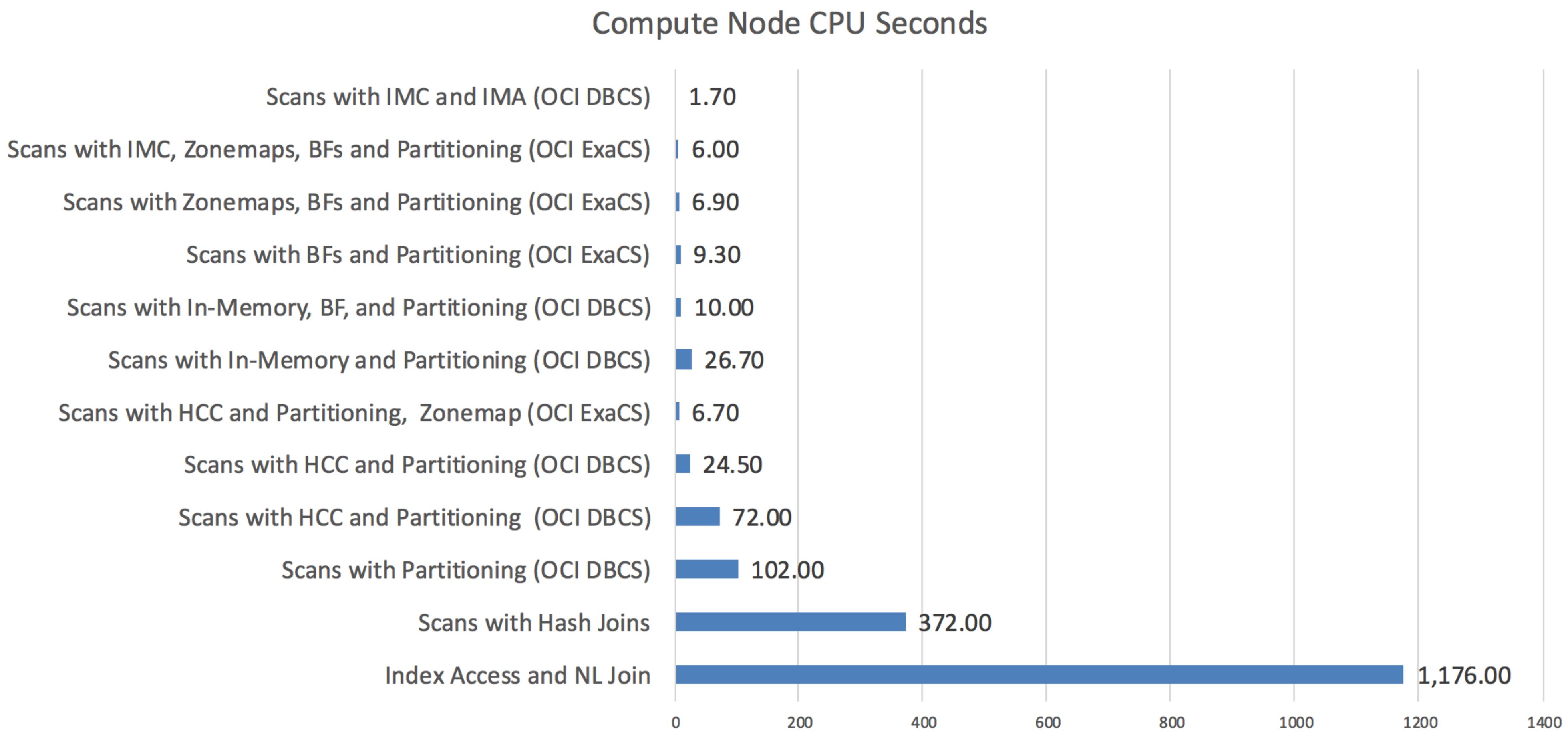
# Things We Can Do for Joins and Aggregation

## Elapsed Seconds



# Things We Can Do for Joins and Aggregation

## CPU Seconds



ORACLE

ORACLE
REAL-WORLD PERFORMANCE

# Things We Can Do for Joins and Aggregation

## Leverage Chart

| Method | % of Time in Data Acquisition | % of Time in Joins | % of Time in Sort/Aggregate | Elapsed Seconds |
|---|---|---|---|---|
| IMC with IMA and Partitioning (OCI DBCS) | 100% | 0% | 0% | 1 |
| IMC & Partitioning with BFs and Zone Maps (OCI ExaCS) | 28% | 58% | 14% | 3 |
| Partitioning with BFs and Zone Maps (OCI ExaCS) | 33% | 50% | 17% | 4 |
| Partitioning with BFs (OCI ExaCS) | 30% | 50% | 20% | 6 |
| IMC with BFs and Partitioning (OCI DBCS) | 30% | 40% | 10% | 5 |
| IMC with Partitioning (OCI DBCS) | 18% | 64% | 18% | 13 |
| Partitioning with HCC & Zone Maps (OCI ExaCS) | 57% | 29% | 14% | 5 |
| Partitioning with HCC (OCI ExaCS) | 48% | 51% | 1% | 13 |
| Partitioning with Compression (OCI DBCS) | 81% | 14.5% | 5.8% | 36 |
| Partitioning (OCI DBCS) | 79% | 17% | 4% | 55 |
| Scans with Hash Joins ((OCI DBCS) | 86% | 12.24% | 1.60% | 187 |
| Index access and NL Join (OCI DBCS) | 99% | .08% | .17% | 1,195 |

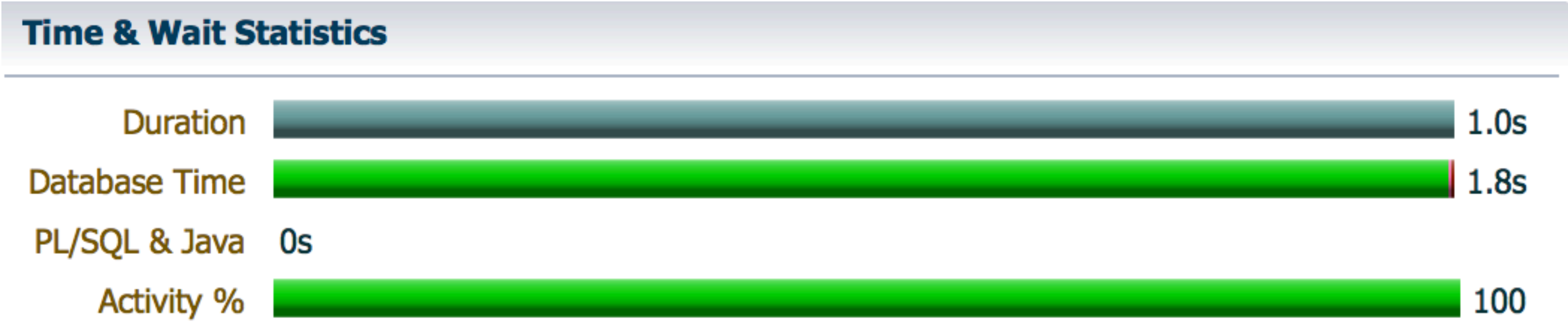
ORACLE

ORACLE
REAL-WORLD PERFORMANCE

# Bonus

## Sorting on Database Cloud Service, In-Memory, In-Memory Aggregation

- We can’t use Zone Maps with Attribute Clustering on non-Exadata, but here’s something (relatively) free in the Cloud
- Let’s manually sort the data to leverage In-Memory Min-Max pruning



**Look for ”IM scan CUs pruned” or “IM scan rows optimized” stats**

# Features Availability

| Feature | Oracle Cloud Infrastructure Database Cloud Service | Oracle Cloud Infrastructure Exadata Cloud Service |
|---|---|---|
| Partitioning | ✓ | ✓ |
| Hybrid Columnar Compression | ✓ | ✓ |
| Zone Maps and Attribute Clustering | | ✓ |
| In-Memory and In-Memory Aggregation | ✓ | ✓ |
| Bloom Filters | ✓ | ✓ |

# Summary

- These days, scan performance usually isn’t about reducing I/O, it’s about reducing CPU
- There are a number of ways to do this
- We reduced CPU from 1,176 seconds  to 1.7 seconds for the same query
- We reduced elapsed time from 1,195 seconds to 1 second for the same query
- Look for leverage!
- Don’t settle for “good enough”

# Summary

What can you do with your system?

# Safe Harbor Statement

The preceding 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.

# Integrated Cloud

## Applications & Platform Services

ORACLE®