Oracle Database In-Memory Option in Action

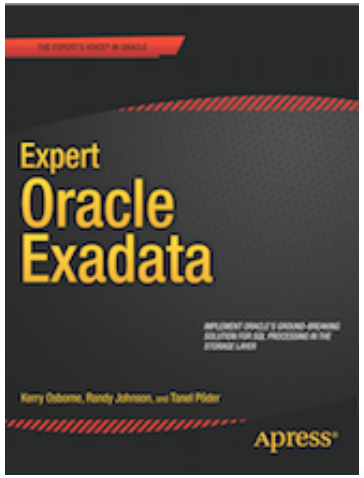
Tanel Põder & Kerry Osborne
Accenture Enkitec Group

http://www.enkitec.com

Intro: About

- Tanel Põder
  - Consultant, Trainer, Troubleshooter
  - Oracle Database Performance geek
  - Exadata Performance geek
  - In-Memory Columnar Cache perf. geek ;-)
  - http://blog.tanelpoder.com

- Professor Osborne
  - Nice Guy and All Around Prince of a Fellow ☺
  - Worked on Oracle since v2
  - Also a performance geek
  - http://kerryosborne.oracle-guy.com



Expert Oracle Exadata book
(with Kerry Osborne and Randy Johnson of Enkitec)

Intro: Goals

- Walkthrough of a few example queries that benefit from In-Memory and Oracle 12.1.0.2 features

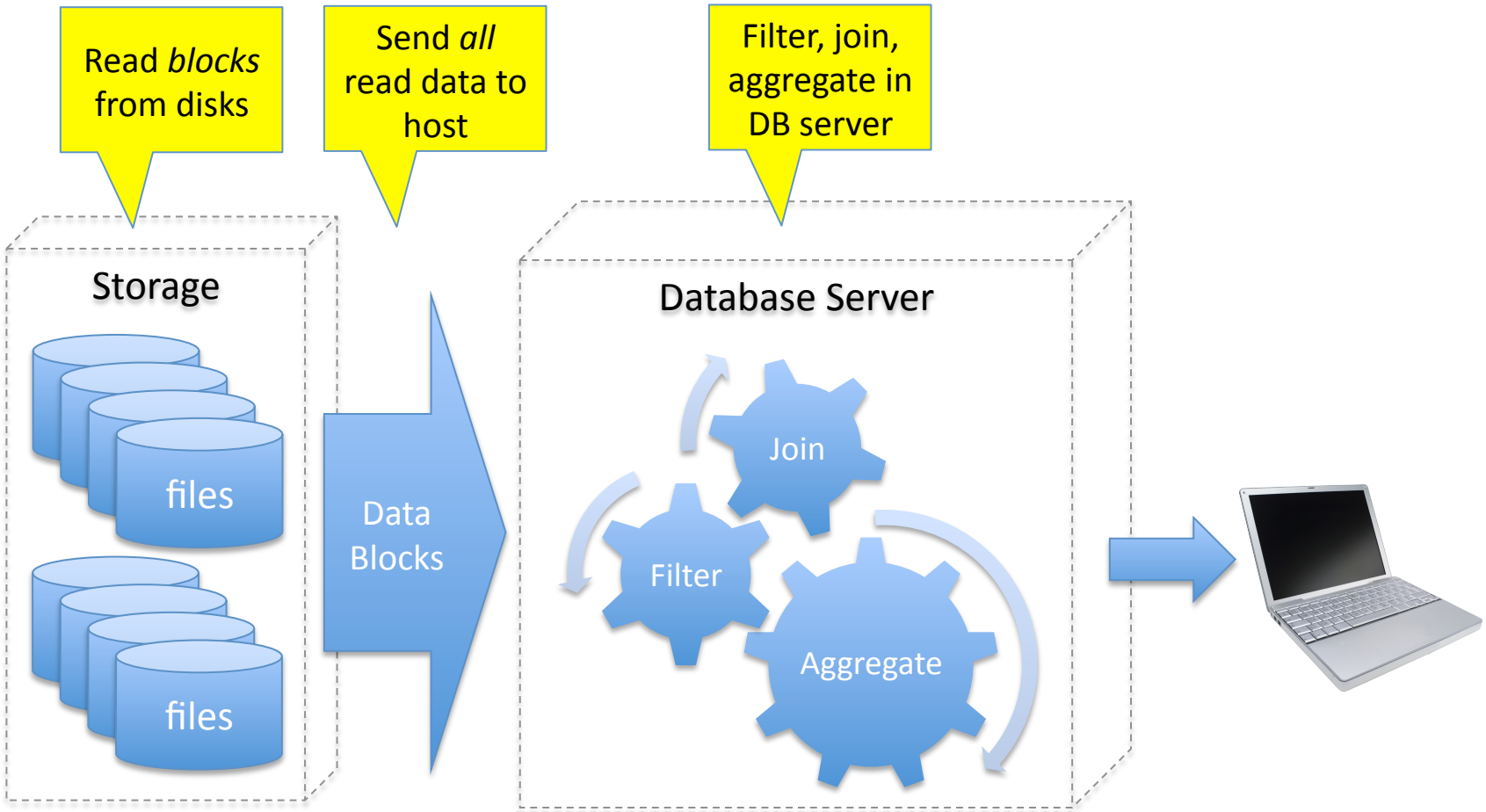
- First disable most of the performance features and then re-enable one by one (and show performance metrics)

Intro: What do databases do?

1. Retrieve data
2. Process data
3. Return results

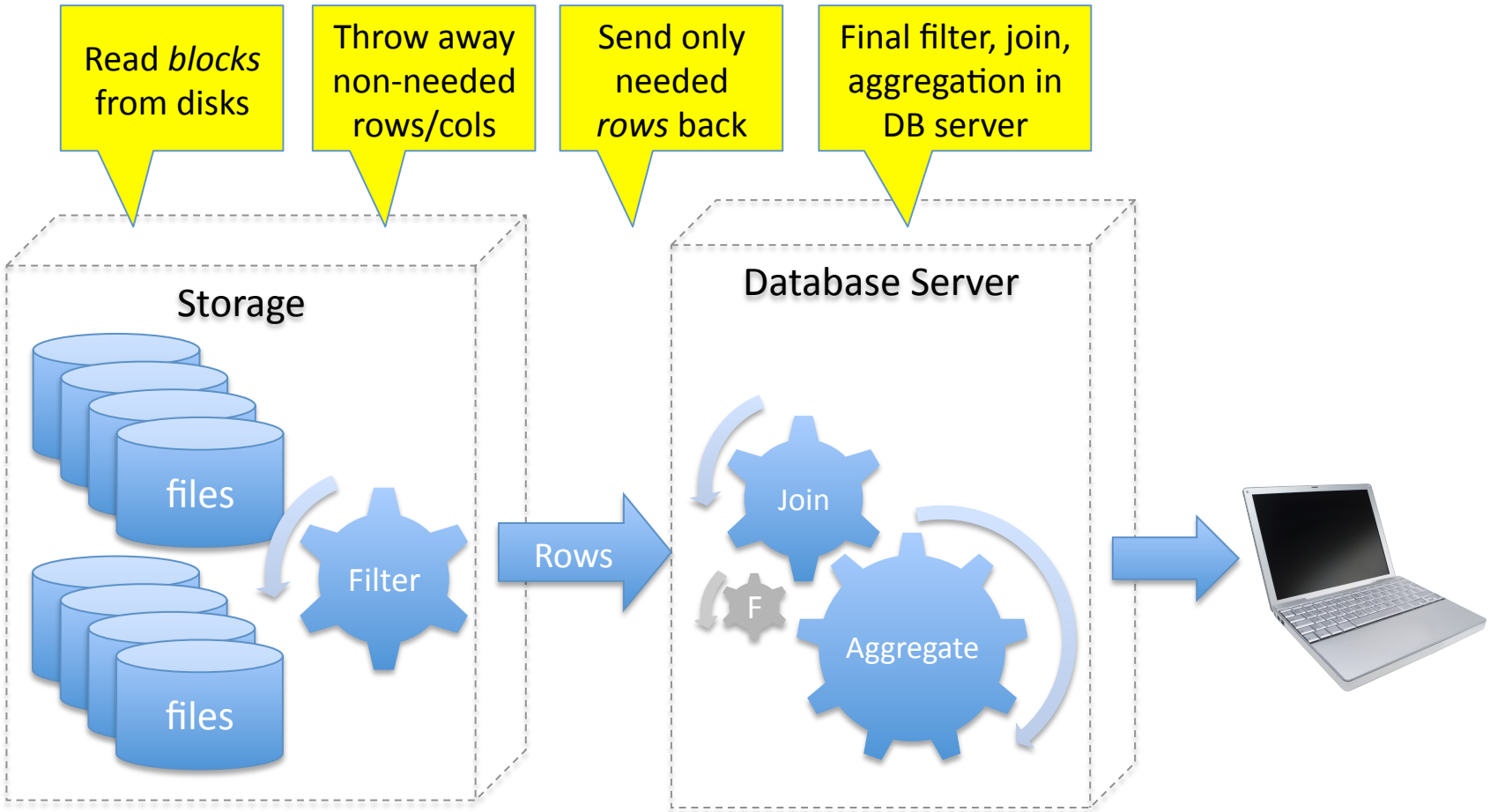
Traditional Database I/O flow

- Retrieve -> Process -> Return



Exadata I/O flow

- Retrieve -> Process -> Return



In-Memory Column Store I/O flow

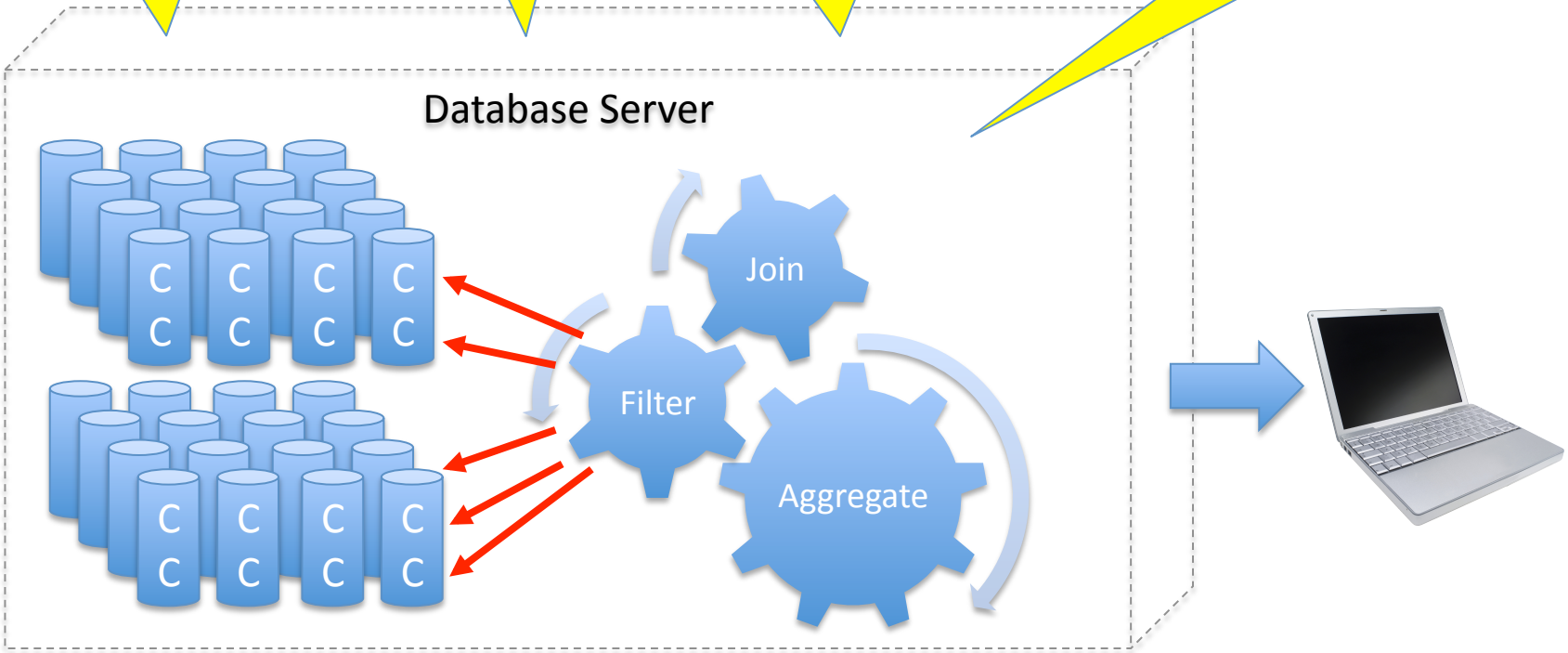
- Retrieve -> Process -> Return

Disk storage not in data *retrieval* critical path at all

Fast RAM access. No disk latency

Avoid RAM access with In-Memory "storage" indexes

Oracle 12c also brings data *processing* improvements (SIMD, vector joins, vector aggregation, approximate distinct)



Data Retrieval

A simple Data Retrieval test!

- Retrieve **1%** rows out of a **8 GB** table:

```
SELECT
    COUNT(*)
  , SUM(order_total)
FROM
    orders
WHERE
     warehouse_id BETWEEN 500 AND 510
```

Test data generated by SwingBench tool

The Warehouse IDs range between 1 and 999

Data Retrieval Test!

- Simulate a traditional Oracle database configuration:
  - The test *hardware* is still modern Exadata with flash enabled!

```
SQL> ALTER SESSION SET cell_offload_processing = FALSE;

Session altered.

SQL> ALTER SESSION SET "_serial_direct_read" = NEVER;

Session altered.

SQL> ALTER SESSION SET inmemory_query = DISABLE;

Session altered.
```

I will re-enable the settings in following tests

Data Retrieval: Index Range Scan

```
INDEX hint. Index lookups happen via buffer cache
```

Overview

General

SQL Text	SELECT /*+ MONITOR INDEX(o) test1: index range scan
Execution Started	Thu Sep 25, 2014 9:36:04 PM
Last Refresh Time	Thu Sep 25, 2014 9:40:29 PM
Execution ID	16777216
User	TANEL
Fetch Calls	1

Time & Wait Statistics

Duration	4.4m
Database Time	4.9m
PL/SQL & Java	0s
Activity %	100

IO Statistics

Buffer Gets	782K
IO Requests	511K
IO Bytes	4GB

User I/O: cell list of blocks physical read: 253 samples (95%)

Details

Plan Statistics | Plan | Activity | Metrics

Plan Hash Value 16715356 | Plan Note

Operation	Name	Li...	Estima...	Cost	Timeline(2...	Exe...	Actua...	Mem...	Temp...	O...	IO R...	IO...	Activity %
SELECT STATEMENT		0				1	1						
SORT AGGREGATE		1	1			1	1						
TABLE ACCESS BY IND...	ORDERS	2	859K	854K		1	787K				511K	4GB	100
INDEX RANGE SCAN	ORD_WAREI	3	859K	2,154		1	787K				180	1MB	

Data Retrieval: Full Table Scan - Buffered

```
ALTER SESSION SET "_serial_direct_read"=NEVER;
```

Overview

General

SQL Text	SELECT /*+ MONITOR FULL(o) test2: full buffered */ C
Execution Started	Thu Sep 25, 2014 9:40:32 PM
Last Refresh Time	Thu Sep 25, 2014 9:42:44 PM
Execution ID	16777216
User	TANEL
Fetch Calls	1

Time & Wait Statistics

Duration	2.2m
Database Time	2.5m
PL/SQL & Java	0s
Activity %	100

IO Statistics

Buffer Gets	1,014K
IO Requests	137K
IO Bytes	6GB

User I/O: cell multiblock physical read: 92 samples (70%)

Details

Plan Statistics | Plan | Activity | Metrics

Plan Hash Value 630573765 | Plan Note

User I/O: cell single block physical read: 11 samples (8.4%)

Operation	Name	Li...	Estima...	C...	Timeline(1...	Exe...	Actua...	Mem...	Temp...	O..	IO R...	IO...	Activity %
SELECT STATEMENT		0				1	1						
SORT AGGREGATE		1	1			1	1						
TABLE ACCESS STORAG...	ORDERS	2	859K	276K		1	787K				137K	6GB	100

Data Retrieval: Full Table Scan – Direct Path Reads

```
ALTER SESSION SET "_serial_direct_read"=ALWAYS;
```

Overview

General

SQL Text	SELECT /*+ MONITOR FULL(o) test3: full direct path */
Execution Started	Thu Sep 25, 2014 9:42:46 PM
Last Refresh Time	Thu Sep 25, 2014 9:43:02 PM
Execution ID	16777216
User	TANEL
Fetch Calls	1

Time & Wait Statistics

Duration	16.0s
Database Time	15.9s
PL/SQL & Java	0s
Activity %	100

IO Statistics

Buffer Gets	1,014K
IO Requests	7,944
IO Bytes	8GB

User I/O: direct path read: 5 samples (31%)

Details

Plan Statistics | Plan | Activity | Metrics

Plan Hash Value 630573765 | Plan Note

Operation	Name	Li...	Estima...	C...	Timeline(1...	Exe...	Actua...	Mem...	Temp...	O..	IO R...	IO...	Activity %
SELECT STATEMENT		0				1	1						
SORT AGGREGATE		1	1			1	1						
TABLE ACCESS STORAG...	ORDERS	2	859K	276K		1	787K				7,944	8GB	100

Data Retrieval: Full Table Scan – Smart Scan

```
ALTER SESSION SET cell_offload_processing = TRUE
```

Overview

General

SQL Text	SELECT /*+ MONITOR FULL(o) test4: full smart scan */
Execution Started	Thu Sep 25, 2014 9:43:03 PM
Last Refresh Time	Thu Sep 25, 2014 9:43:05 PM
Execution ID	16777216
User	TANEL
Fetch Calls	1

Time & Wait Statistics

Duration	2.0s
Database Time	2.3s
PL/SQL & Java	0s
Activity %	100

IO Statistics

Buffer Gets	1,014K
IO Requests	7,944
IO Bytes	8GB
Cell Offload Efficiency	100%

Details

Plan Statistics | Plan | Activity

User I/O: cell smart table scan: 2 samples (100%)

Plan Hash Value 630573765 | Plan Note

Operation	Name	Li...	Estim...	C...	Timeline(...	Ex...	Actu...	Mem...	Tem...	O..	IO ...	I...	Cell Offlo...	Activity %
SELECT STATEMENT		0				1	1							
SORT AGGREGATE		1	1			1	1							
TABLE ACCESS STOR...	ORDERS	2	859K	276K		1	787K	1MB			7,944	8GB	100	100

Data Retrieval: Full Table Scan – In-Memory Scan

```
ALTER SESSION SET inmemory_query = ENABLE
```

Overview

General

SQL Text	SELECT /*+ MONITOR FULL(o) test5: full inmemory sca
Execution Started	Thu Sep 25, 2014 9:43:06 PM
Last Refresh Time	Thu Sep 25, 2014 9:43:06 PM
Execution ID	16777216
User	TANEL
Fetch Calls	1

Time & Wait Statistics

Duration	0.1s
Database Time	0.1s
PL/SQL & Java	0s
Activity %	0

IO Statistics

Buffer Gets	8
IO Requests	0
IO Bytes	0

CPU: 147.0ms (100%)

Details

Plan Statistics | Plan | Activity

Plan Hash Value 630573765 | Plan Note

Operation	Name	Li...	Estima...	C...	Timeline(0....	Exe...	Actua...	Mem...	Temp...	O..	IO R...	IO...	Activity %
SELECT STATEMENT		0				1	1						
SORT AGGREGATE		1	1			1	1						
TABLE ACCESS INMEMO...	ORDERS	2	859K	11K		1	787K						

Data Retrieval: Test Results (from V$SQL)

- Remember, this is a very simple query operation
  - But complex queries are just a bunch of simple query ops in a loop ;-)

TESTNAME	PLAN_HASH	ELA_MS	CPU_MS	LIOS	BLK_READ
test1: index range scan *	16715356	265203	37438	782858	511231
test2: full buffered */ C	630573765	132075	48944	1013913	849316
test3: full direct path *	630573765	15567	11808	1013873	1013850
test4: full smart scan */	630573765	2102	729	1013873	1013850
test5: full inmemory scan	630573765	155	155	14	0

Eliminating the data *retrieval* IO component gave **1711x** speed improvement

CPU usage also dropped **241x**

Note that tests 1-4 all did physical IO as data working set didn't fit into cache

Comparing "in memory" with In-Memory

Cache all table *blocks* in buffer cache?

Data Retrieval: Full Table Scan – Buffer Cache Scan

```
ALTER TABLE orders CACHE;  ... SET inmemory_query = DISABLE
```

Overview

General

SQL Text	SELECT /*+ MONITOR FULL(o) test6: full buffered all */
Execution Started	Fri Sep 26, 2014 7:05:51 PM
Last Refresh Time	Fri Sep 26, 2014 7:05:59 PM
Execution ID	16777216
User	TANEL
Fetch Calls	1

Time & Wait Statistics

Duration	8.0s
Database Time	7.8s
PL/SQL & Java	0s
Activity %	100

IO Statistics

Buffer Gets	1,014K
IO Requests	0
IO Bytes	0

Cpu: 8 samples (100%)

Details

Plan Statistics | Plan | Activity | Metrics

Plan Hash Value 630573765 | Plan Note

Operation	Name	Li...	Estima...	C...	Timeline(8s)	Exe...	Actua...	Mem...	Temp...	O...	IO R...	IO...	Activity %
SELECT STATEMENT		0				1	1						
SORT AGGREGATE		1	1			1	1						
TABLE ACCESS STORAG...	ORDERS	2	859K	276K		1	787K						100

Data Retrieval: Test Results (Buffer Cache)

- Remember, this is a very simple query operation
  - But complex queries are just a bunch of simple query ops in a loop ;-)

TESTNAME	PLAN_HASH	ELA_MS	CPU_MS	LIOS	BLK_READ
test1: index range scan *	16715356	**265203**	**37438**	782858	511231
test2: full buffered */ C	630573765	132075	48944	1013913	849316
test3: full direct path *	630573765	15567	11808	1013873	1013850
test4: full smart scan */	630573765	2102	729	1013873	1013850
test5: full inmemory scan	630573765	**155**	155	**14**	0
test6: full buffer cache	630573765	**7850**	7831	**1014741**	0

50x difference in logical reads via buffer cache vs. IM processing

Your mileage will vary depending on hardware, dataset, filter % and predicates

"Secret Sauce"

- Columnar organization
  - Compression
  - Column data tightly packed together

SIMD would be useless if you waited on main memory all the time

- **Less memory traffic!**
  - Yes, RAM is the new disk (*slow compared to CPU speed!*)
  - Load only those memory lines where required columns reside
  - Decompression on-the-fly (probably) benefits from CPU L2/L3 cache

- SIMD
  - Reduce tight loops and branches in machine code
  - Get the CPU to simultaneously process multiple values in a vector

SIMD benefit (not Oracle-specific)

- Modern Intel CPUs have 16-32 SIMD registers

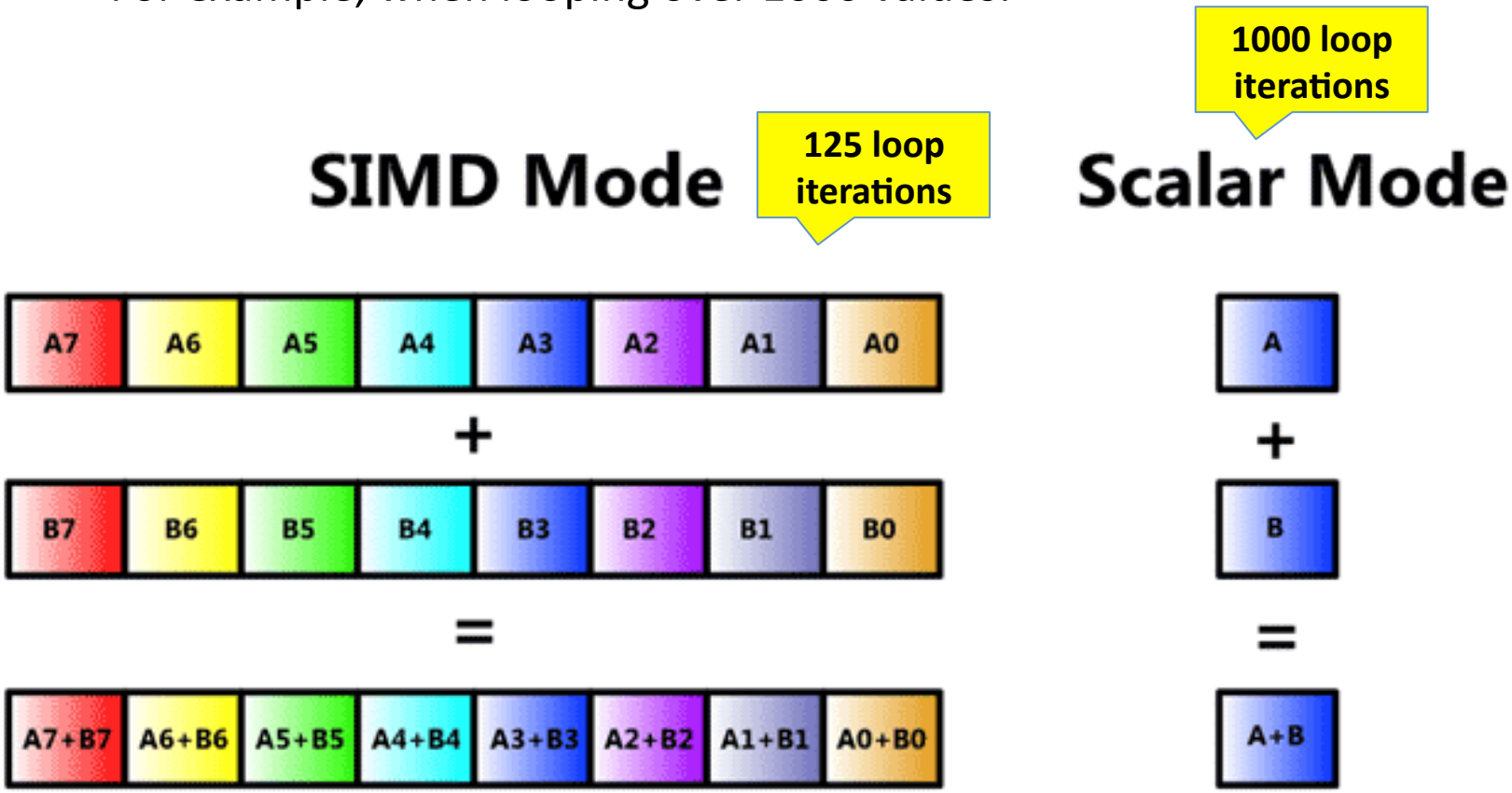
- Each register holds 128, 256 or soon 512 bits:
  - SSE/AVX, AVX2, AVX-512
  - A single register can hold many smaller-length values ***packed*** into it (depending on datatypes)

1. Vector load
2. Vector comparison
   - Filter predicates!
3. Masked Vector addition (etc)
4. Masked Vector store to RAM

Masking allows you to choose which packed values in register to process or ignore (no need to copy stuff around)

SIMD benefit (not Oracle-specific)

- Reduce the number of loops at low-level data operations
  - 2-16x on Intel CPUs, depending on HW & internal data types used
  - For example, when looping over 1000 values:



Data Processing

Data Processing

- Joins
- Aggregations / Group By
- Sorting
- etc…

Example 1: A Small Aggregation

```
SELECT /*+ MONITOR
           NO_VECTOR_TRANSFORM
           NO_PX_JOIN_FILTER(@"SEL$1" "S"@"SEL$1") */
    ch.channel_desc
  , SUM(s.quantity_sold)
FROM
    ssh.sales s
  , ssh.customers cu
  , ssh.channels ch
WHERE
    s.cust_id = cu.cust_id
AND s.channel_id = ch.channel_id
AND cu.cust_postal_code LIKE 'MMM%'
GROUP BY
    ch.channel_desc
```

No Bloom Filter Pushdown, No Vector Transformation

```
=========================================================================================
| Id |            Operation             |   Name    |  Rows   |Activity | Activity Detail |
|    |                                  |           |(Actual) |  (%)    |   (# samples)   |
=========================================================================================
|  0 | SELECT STATEMENT                 |           |       5 |         |                 |
|  1 |   HASH GROUP BY                  |           |       5 |         |                 |
|  2 |    HASH JOIN                     |           |      64 |         |                 |
|  3 |     HASH JOIN                    |           |      64 |   83.33 | Cpu (15)        |
|  4 |      PARTITION RANGE ALL         |           |       9 |         |                 |
|  5 |       TABLE ACCESS INMEMORY FULL | CUSTOMERS |       9 |         |                 |
|  6 |      PARTITION RANGE ALL         |           |    211M |         |                 |
|  7 |       TABLE ACCESS INMEMORY FULL | SALES     |    211M |   11.11 | in memory (1)   |
|    |                                  |           |         |         | Cpu (1)         |
|  8 |     TABLE ACCESS INMEMORY FULL   | CHANNELS  |       5 |         |                 |
=========================================================================================

   2 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
   3 - access("S"."CUST_ID"="CU"."CUST_ID")
   5 - inmemory("CU"."CUST_POSTAL_CODE" LIKE 'MMM%')
       filter("CU"."CUST_POSTAL_CODE" LIKE 'MMM%')
```

With Bloom Filter Pushdown, No Vector Transform

```
=========================================================================================
| Id |             Operation             |   Name    |  Rows   | Activity | Activity Detail |
|    |                                   |           |(Actual) |   (%)    |   (# samples)   |
=========================================================================================
|  0 | SELECT STATEMENT                  |           |       5 |          |                 |
|  1 |   HASH GROUP BY                   |           |       5 |          |                 |
|  2 |    HASH JOIN                      |           |      64 |          |                 |
|  3 |     HASH JOIN                     |           |      64 |          |                 |
|  4 |      JOIN FILTER CREATE           | :BF0000   |       9 |          |                 |
|  5 |       PARTITION RANGE ALL         |           |       9 |          |                 |
|  6 |        TABLE ACCESS INMEMORY FULL | CUSTOMERS |       9 |          |                 |
|  7 |      JOIN FILTER USE              | :BF0000   |   24753 |          |                 |
|  8 |       PARTITION RANGE ALL         |           |   24753 |          |                 |
|  9 |        TABLE ACCESS INMEMORY FULL | SALES     |   24753 |   100.00 | in memory (3)   |
| 10 |     TABLE ACCESS INMEMORY FULL    | CHANNELS  |       5 |          |                 |
=========================================================================================

   2 - access("S"."CHANNEL_ID"="CH"."CHANNEL_ID")
   3 - access("S"."CUST_ID"="CU"."CUST_ID")
   6 - inmemory("CU"."CUST_POSTAL_CODE" LIKE 'MMM%')
       filter("CU"."CUST_POSTAL_CODE" LIKE 'MMM%')
   9 - inmemory(SYS_OP_BLOOM_FILTER(:BF0000,"S"."CUST_ID"))
       filter(SYS_OP_BLOOM_FILTER(:BF0000,"S"."CUST_ID"))
```

With Vector Transformation

```
=========================================================================================
| Id |                Operation                |           Name |  Rows   |Activity | Activity Detail |
|    |                                         |                |(Actual) |  (%)    |   (# samples)   |
=========================================================================================
|  0 | SELECT STATEMENT                        |                |       5 |         |                 |
|  1 |   TEMP TABLE TRANSFORMATION             |                |       5 |         |                 |
|  2 |    LOAD AS SELECT                       |                |       2 |         |                 |
|  3 |     VECTOR GROUP BY                     |                |       1 |         |                 |
|  4 |      HASH GROUP BY                      |                |       0 |         |                 |
|  5 |       KEY VECTOR CREATE BUFFERED        | :KV0000        |       9 |         |                 |
|  6 |        PARTITION RANGE ALL              |                |       9 |         |                 |
|  7 |         TABLE ACCESS INMEMORY FULL      | CUSTOMERS      |       9 |         |                 |
|  8 |    LOAD AS SELECT                       |                |       2 |         |                 |
|  9 |     VECTOR GROUP BY                     |                |       5 |         |                 |
| 10 |      KEY VECTOR CREATE BUFFERED         | :KV0001        |       5 |         |                 |
| 11 |       TABLE ACCESS INMEMORY FULL        | CHANNELS       |       5 |         |                 |
| 12 |    HASH GROUP BY                        |                |       5 |         |                 |
| 13 |     HASH JOIN                           |                |       5 |         |                 |
| 14 |      MERGE JOIN CARTESIAN               |                |       5 |         |                 |
| 15 |       TABLE ACCESS FULL                 | SYS_TEMP_0FD...|       1 |         |                 |
| 16 |       BUFFER SORT                       |                |       5 |         |                 |
| 17 |        TABLE ACCESS FULL                | SYS_TEMP_0FD...|       5 |         |                 |
| 18 |      VIEW                               | VW_VT_0737CF93 |       5 |         |                 |
| 19 |       VECTOR GROUP BY                   |                |       5 |         |                 |
| 20 |        HASH GROUP BY                    |                |       0 |         |                 |
| 21 |         KEY VECTOR USE                  | :KV0001        |      64 |         |                 |
| 22 |          KEY VECTOR USE                 | :KV0000        |      64 |         |                 |
| 23 |           PARTITION RANGE ALL           |                |      64 |         |                 |
| 24 |            TABLE ACCESS INMEMORY FULL   | SALES          |      64 |  100.00 | in memory (2)   |
=========================================================================================
```

With Vector Transformation

(…continued…)

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

   7 - inmemory("CU"."CUST_POSTAL_CODE" LIKE 'MMM%')
       filter("CU"."CUST_POSTAL_CODE" LIKE 'MMM%')
  13 - access("ITEM_9"=INTERNAL_FUNCTION("C0") AND "ITEM_10"="C2"
              AND "ITEM_7"=INTERNAL_FUNCTION("C0") AND "ITEM_8"="C2")
  24 - inmemory((SYS_OP_KEY_VECTOR_FILTER("S"."CUST_ID",:KV0000) AND
                SYS_OP_KEY_VECTOR_FILTER("S"."CHANNEL_ID",:KV0001)))
       filter((SYS_OP_KEY_VECTOR_FILTER("S"."CUST_ID",:KV0000) AND
                SYS_OP_KEY_VECTOR_FILTER("S"."CHANNEL_ID",:KV0001)))

   - dynamic statistics used: dynamic sampling (level=2)
   - 1 Sql Plan Directive used for this statement
   - vector transformation used for this statement
```

Example 2: A bigger aggregation

```
SELECT /*+ MONITOR
           NO_VECTOR_TRANSFORM
           NO_PX_JOIN_FILTER(@"SEL$1" "S"@"SEL$1") */
    ch.channel_desc
  , p.promo_subcategory
  , SUM(s.quantity_sold)
FROM
    ssh.sales s
  , ssh.channels ch
  , ssh.promotions p
WHERE
    s.channel_id = ch.channel_id
AND s.promo_id = p.promo_id
AND p.promo_category = 'TV'
GROUP BY
    ch.channel_desc
  , p.promo_subcategory
```

No Bloom Filter Pushdown, No Vector Transformation

```
=========================================================================================
| Id |             Operation              |    Name    |  Rows   |Activity | Activity Detail |
|    |                                    |            |(Actual) |  (%)    |   (# samples)   |
=========================================================================================
|  0 | SELECT STATEMENT                   |            |      15 |         |                 |
|  1 |   HASH GROUP BY                    |            |      15 |         |                 |
|  2 |    HASH JOIN                       |            |      15 |         |                 |
|  3 |     TABLE ACCESS INMEMORY FULL     | CHANNELS   |       5 |         |                 |
|  4 |     VIEW                           | VW_GBC_10  |      15 |         |                 |
|  5 |      HASH GROUP BY                 |            |      15 |   35.00 | Cpu (7)         |
|  6 |       HASH JOIN                    |            |     48M |   50.00 | Cpu (10)        |
|  7 |        TABLE ACCESS INMEMORY FULL  | PROMOTIONS |     115 |         |                 |
|  8 |        PARTITION RANGE ALL         |            |    211M |         |                 |
|  9 |         TABLE ACCESS INMEMORY FULL | SALES      |    211M |   15.00 | in memory (3)   |
=========================================================================================

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

   2 - access("ITEM_1"="CH"."CHANNEL_ID")
   6 - access("S"."PROMO_ID"="P"."PROMO_ID")
   7 - inmemory("P"."PROMO_CATEGORY"='TV')
       filter("P"."PROMO_CATEGORY"='TV')
```

With Bloom Filter Pushdown, No Vector Transform

```
=========================================================================================
| Id |              Operation              |    Name    |  Rows   |Activity | Activity Detail |
|    |                                     |            |(Actual) |  (%)    |   (# samples)   |
=========================================================================================
|  0 | SELECT STATEMENT                    |            |      15 |         |                 |
|  1 |   HASH GROUP BY                     |            |      15 |         |                 |
|  2 |    HASH JOIN                        |            |      15 |         |                 |
|  3 |     TABLE ACCESS INMEMORY FULL      | CHANNELS   |       5 |         |                 |
|  4 |     VIEW                            | VW_GBC_10  |      15 |         |                 |
|  5 |      HASH GROUP BY                  |            |      15 |   71.43 | Cpu (10)        |
|  6 |       HASH JOIN                     |            |     48M |   28.57 | Cpu (4)         |
|  7 |        JOIN FILTER CREATE           | :BF0000    |     115 |         |                 |
|  8 |         TABLE ACCESS INMEMORY FULL  | PROMOTIONS |     115 |         |                 |
|  9 |        JOIN FILTER USE              | :BF0000    |     49M |         |                 |
| 10 |         PARTITION RANGE ALL         |            |     49M |         |                 |
| 11 |          TABLE ACCESS INMEMORY FULL | SALES      |     49M |         |                 |
=========================================================================================

   2 - access("ITEM_1"="CH"."CHANNEL_ID")
   6 - access("S"."PROMO_ID"="P"."PROMO_ID")
   8 - inmemory("P"."PROMO_CATEGORY"='TV')
       filter("P"."PROMO_CATEGORY"='TV')
  11 - inmemory(SYS_OP_BLOOM_FILTER(:BF0000,"S"."PROMO_ID"))
       filter(SYS_OP_BLOOM_FILTER(:BF0000,"S"."PROMO_ID"))
```

With Vector Transformation

```
=================================================================================================
| Id |                Operation                |       Name       |  Rows   |Activity |Activity Detail|
|    |                                         |                  |(Actual) |  (%)    | (# samples)   |
=================================================================================================
|  0 | SELECT STATEMENT                        |                  |      15 |         |               |
|  1 |   TEMP TABLE TRANSFORMATION             |                  |      15 |         |               |
|  2 |    LOAD AS SELECT                       |                  |       2 |         |               |
|  3 |     VECTOR GROUP BY                     |                  |       5 |         |               |
|  4 |      KEY VECTOR CREATE BUFFERED         | :KV0000          |       5 |         |               |
|  5 |       TABLE ACCESS INMEMORY FULL        | CHANNELS         |       5 |         |               |
|  6 |    LOAD AS SELECT                       |                  |       2 |         |               |
|  7 |     VECTOR GROUP BY                     |                  |       3 |         |               |
|  8 |      KEY VECTOR CREATE BUFFERED         | :KV0001          |     115 |         |               |
|  9 |       TABLE ACCESS INMEMORY FULL        | PROMOTIONS       |     115 |         |               |
| 10 |    HASH GROUP BY                        |                  |      15 |         |               |
| 11 |     HASH JOIN                           |                  |      15 |         |               |
| 12 |      HASH JOIN                          |                  |      15 |         |               |
| 13 |       TABLE ACCESS FULL                 | SYS_TEMP_0FD...  |       5 |         |               |
| 14 |       VIEW                              | VW_VT_0737CF     |      15 |         |               |
| 15 |        VECTOR GROUP BY                  |                  |      15 |   20.00 | Cpu (1)       |
| 16 |         HASH GROUP BY                   |                  |       0 |         |               |
| 17 |          KEY VECTOR USE                 | :KV0000          |     48M |         |               |
| 18 |           KEY VECTOR USE                | :KV0001          |     48M |         |               |
| 19 |            PARTITION RANGE ALL          |                  |     48M |         |               |
| 20 |             TABLE ACCESS INMEMORY FULL  | SALES            |     48M |   60.00 | in memory (3) |
| 21 |      TABLE ACCESS FULL                  | SYS_TEMP_0FD...  |       3 |         |               |
=================================================================================================
```

(…continued…)

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

   9 - inmemory("P"."PROMO_CATEGORY"='TV')
       filter("P"."PROMO_CATEGORY"='TV')
  11 - access("ITEM_10"=INTERNAL_FUNCTION("C0") AND "ITEM_11"="C2")
  12 - access("ITEM_8"=INTERNAL_FUNCTION("C0") AND "ITEM_9"="C2")
  20 - inmemory((SYS_OP_KEY_VECTOR_FILTER("S"."PROMO_ID",:KV0001) AND
                 SYS_OP_KEY_VECTOR_FILTER("S"."CHANNEL_ID",:KV0000)))
       filter((SYS_OP_KEY_VECTOR_FILTER("S"."PROMO_ID",:KV0001) AND
              SYS_OP_KEY_VECTOR_FILTER("S"."CHANNEL_ID",:KV0000)))

   - dynamic statistics used: dynamic sampling (level=2)
   - 1 Sql Plan Directive used for this statement
   - vector transformation used for this statement
```

A query bottlenecked by data *processing*, not *retrieval*

- A query bottlenecked by data *processing*, not retrieval
- Hash joins and a GROUP BY spilling to TEMP

Overview

SQL ID	7jsk74a56hm2x
Execution Started	Tue Mar 6, 2012 10:57:26 AM
Last Refresh Time	Tue Mar 6, 2012 11:07:58 AM
Execution ID	16777217
User	TANEL
Fetch Calls	0

Time & Wait Statistics

Duration	10.5m
Database Time	10.5m
PL/SQL & Java	0.0s
Wait Activity %	100

IO Statistics

Buffer Gets	1,352K
IO Requests	86K
IO Bytes	26GB
Cell Offload Efficiency	-14.94%

Details

Plan Statistics | Plan | Activity | Metrics

Plan Hash Value 147847724

TIP: Right mouse click on the table allows to toggle between IO Requests and IO Bytes

Operation	Name	Estim...	Cost	Timeline(63...	Ex...	Act...	Me...	Tem...	IO Bytes	Cell...	CPU Activity ...	Wait Activity...
CREATE TABLE STATEMENT					1	1						
LOAD AS SELECT					1	1	529KB		985MB		7.02	2.7
HASH GROUP BY		135M	2,45		1	26M	816ME	2GB	4GB		46	19
HASH JOIN		135M	727I		1	135M	648ME	5GB	9GB		32	52
HASH JOIN		45M	325I		1	45M	415ME	2GB	3GB		17	26
TABLE ACCESS STORAG...	ORDERS	45M	96K		1	45M			3GB	72	.67	
TABLE ACCESS STORAG...	CUSTOMERS	40M	105I		1	40M			3GB	59	.34	
TABLE ACCESS STORAGE ...	ORDER_ITEMS	135M	134I		1	135M			4GB	38	1.01	

Reducing PGA memory usage and TEMP IO?

- Classic SQL optimization techniques:
  - Filter early -> Sort and Join less rows
  - Group to fewer buckets
  - Partition wise joins (or "chunkify" workload)
  - Changing join orders
- Kill it with Hardware:
  - Increase PGA_AGGREGATE_TARGET
  - Increase PX degree
  - You'll use more CPU … and more memory!
- Not all operations are simply additive
  - They can't spread memory usage into "partitions" with more slaves!
  - **DISTINCT !**

Approximate Count Distinct (12.1.0.2)

```
-- traditional (and precise) way:

SELECT COUNT(DISTINCT cust_id)
FROM ssh.sales
WHERE amount_sold > 1;

-- new (approximate) way:

SELECT APPROX_COUNT_DISTINCT(cust_id)
FROM ssh.sales
WHERE amount_sold > 1;
```

- It looks like this feature utilizes the HyperLogLog algorithm:
  http://externaltable.blogspot.com/2014/08/scaling-up-cardinality-estimates-in.html

COUNT (DISTINCT column)

Overview

General

SQL Text	SELECT /*+ MONITOR */ COUNT(DISTINCT cust_id) F
Execution Started	Thu Oct 2, 2014 4:13:10 AM
Last Refresh Time	Thu Oct 2, 2014 4:14:43 AM
Execution ID	16777218
User	SYSTEM
Fetch Calls	1

Time & Wait Statistics

Duration	1.6m
Database Time	1.6m
PL/SQL & Java	0s
Activity %	100

IO Statistics

Buffer Gets	218
IO Requests	20K
IO Bytes	5GB

Cpu: 91 samples (98%)

Details

Plan Statistics | Plan | Activity | Metri

Plan Hash Value 917664585 | Plan Note

Operation	Name	Li...	Estima...	C...	Timeline(9...	Exe...	Actua...	Mem...	Temp...	O..	IO R...	IO...	Activity %
SELECT STATEMENT		0				1	1						
SORT AGGREGATE		1	1			1	1						
VIEW	VW_DAG_0	2	32M	619K		1	32M						
HASH GROUP BY		3	32M	619K		1	32M	139MB	2GB		20K	5GB	98
PARTITION RANGE ...		4	281M	158K		1	209M						
TABLE ACCESS IN...	SALES	5	281M	158K		30	209M						2.15

Write Requests: 9,926 (50%)
Write Bytes: 2,520,727,552 (50%)
Average IO size: 248KB

APPROX_COUNT_DISTINCT(column)

Overview

General

SQL Text	SELECT /*+ MONITOR */ APPROX_COUNT_DISTINC ...
Execution Started	Thu Oct 2, 2014 4:08:47 AM
Last Refresh Time	Thu Oct 2, 2014 4:09:10 AM
Execution ID	16777216
User	SYSTEM
Fetch Calls	1

Time & Wait Statistics

Duration	23.0s
Database Time	22.5s
PL/SQL & Java	0s
Activity %	100

IO Statistics

Buffer Gets	218
IO Requests	0
IO Bytes	0

Details

Plan Statistics | Plan | Activity | Metrics

Cpu: 12 samples (52%)

Plan Hash Value 3519235612 | Plan Note

Operation	Name	Li...	Estima...	Cost	Timeline(2...	Exe...	Actua...	Mem...	Temp...	O...	IO R...	IO...	Activity %
SELECT STATEMENT		0				1	1						
SORT AGGREGATE APPR...		1	1			1	1						57
PARTITION RANGE ALL		2	281M	158K		1	209M						
TABLE ACCESS INME...	SALES	3	281M	158K		30	209M						43

280,790,226 Rows

Thanks!

Tanel Põder & Kerry Osborne

Accenture Enkitec Group

http://www.enkitec.com

http://kerryosborne.oracle-guy.com

http://blog.tanelpoder.com