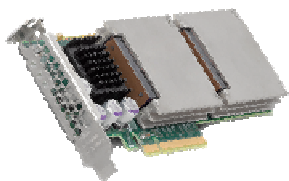
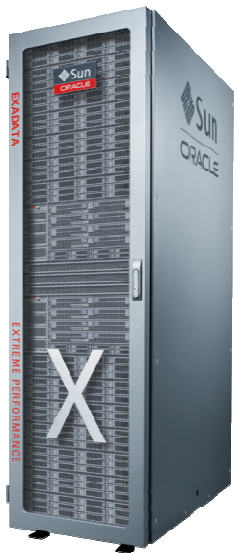




ORACLE®

Introducing Exadata X3

# Exadata X3 Hardware Overview



- Overall Exadata Architecture remains the same
  - Working great, no need for change
  - DB Machine names change from X2 to X3 (e.g. X3-2)
- X3 has
  - Dramatically more and faster flash memory
  - More DRAM memory
  - Faster CPUs
  - More connectivity
  - Lower power usage
  - Same price as X2
- New entry-level Eighth Rack Database Machine

# Exadata X3-2 Database Server Update

New Intel 8-core E5 “SandyBridge” CPUs, more memory, 10GbE



Database Server	X4170 M2 (current)	Sun Server X3-2 (new)	Improvement
CPU	6-core X5675 (3.06 GHz mid-bin)	<b>8-core</b> E5-2690 (2.9 GHz top-bin)	1.3X to 1.5X CPU performance
Memory	96GB (up to 144GB) 8 GB DIMMs	<b>128GB</b> (up to 256GB) 16 GB DIMMs	1.3X normal 1.7X w/expansion
Networking	4 x 1GbE copper 2 x 10GbE optical card	4 x 1 <b>or 10 GbE copper</b> 2 x 10GbE optical card	3X more 10GbE
PCIe Bus	Gen 2.0	Gen 3.0	Future, requires Gen 3 cards

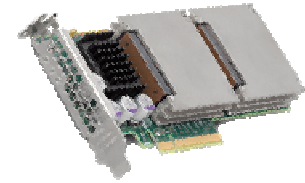
Xeon E5 (Sandy Bridge) - higher performance with lower clock speed

Disk Controller, Disks, and InfiniBand are Unchanged

Unlike X2, no performance penalty for memory expansion

# Exadata Storage Server Update

4X Flash Capacity



Storage Server	X4270 M2 (current)	Sun Server X3-2L (new)	Improvement
Flash	384 GB (4 x 96GB)	<b>1600 GB</b> (4 x 400GB)	4X
Disks	12 x 600GB or 12 x 3TB	No Change	No change
CPU	6-core L5640 (2.26 GHz)	6-core E5-2630L (2.0 GHz)	Similar Performance

Kept Storage CPU at 6-core low power processor.

Storage CPU requires much less throughput than DB CPU.

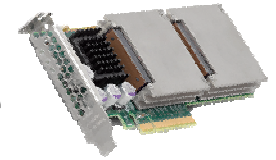
Disk Controller and InfiniBand card are unchanged.

Internal memory increased from 24GB to 64GB for managing large flash.

PCI 3.0 included in new servers but will not see benefit until cards redesigned for 3.0.

# New F40 Flash PCI Card in Storage

4X Capacity, Better Performance, Better Serviceability

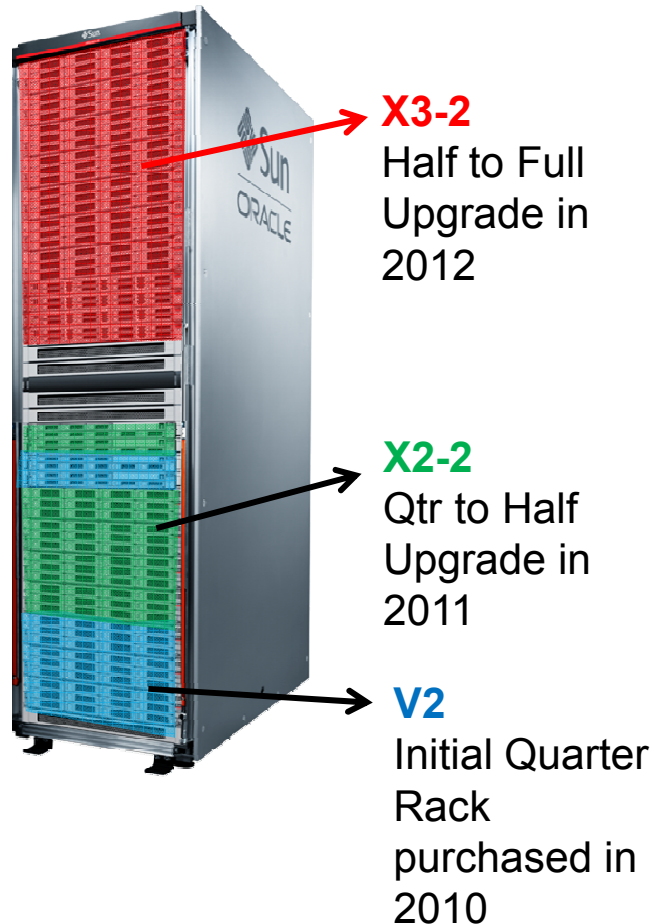


	Current F20 Card	New F40 Card	Improvement
Capacity*	96GB	400GB	4 X
Data Scan Rates	1 GB/s	>1.4 GB/s	1.4X

- New F40 eMLC card with 4X capacity
- eMLC is Enterprise grade Multi-level Cell
  - eMLC has excellent lifetime. Flash lifetime is guaranteed by Oracle. Any failed cards are replaced under Oracle Support contract
- Total read IOPS at the flash card level are much higher than quoted IOPS for DB Machine. We measure end-to-end SQL IOPs, not low level hardware IOPs.
- Read and Write latency improved **by 40%** or more
- Reduces Maintenance by replacing Energy Storage Module (ESM) with much longer lifetime conventional capacitors.

# Seamless Upgrades and Expansion

## Upgrade Example




- X2 or V2 systems can be expanded with X3 hardware
  - A single Database Machine can have servers from different generations
  - Databases and Clusters can span multiple generations of hardware
- As always we don't replace servers or components inside the servers
  - Expand by adding new servers, removing obsolete
- X3 hardware requires recent Exadata software release ( $\geq 11.2.3.2.0$ ) since servers require new drivers, firmware, etc.
  - X3 hardware does not require a Database, ASM, or Clusterware upgrade

# Exadata Eighth Rack



Ideal for  
Smaller Systems,  
Test, Dev, DR

- Lower cost entry configuration
  - 16 Database Cores, 54 TB Disk, 2.4 TB PCI Flash
  - Highly Available configuration with all Exadata features
- Exactly the same hardware as Quarter Rack
- Half the hardware in each server is disabled using software
  - Half the CPU cores in each DB server socket
  - Half the CPU cores in each Storage server socket
  - Half the disks and half the flash cards in each Storage server
    - DB server local disks are all enabled
  - Memory is not disabled - full memory capacity is available
- Upgrade eighth rack to quarter rack by running script
  - Capacity on Demand



# **New Exadata Software**

## **11.2.3.2.0**

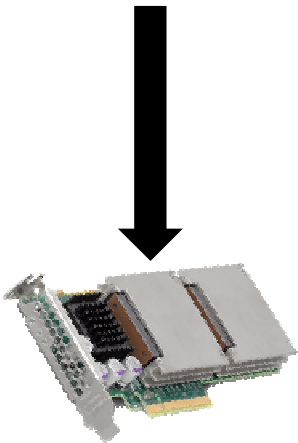
***Features work on all hardware generations***



# Exadata Smart Flash Cache Write-Back

Up to 20X write IOPS

**Writes I/Os**



**1M Flash Write IOPs**  
on X3 DB Machines

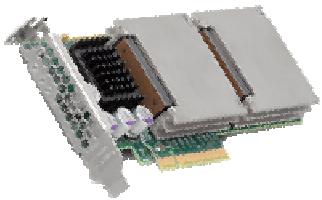
**500K Flash IOPs**  
on X2 DB Machines

- Caches Write I/Os in flash in addition to Read I/Os
- Accelerates write intensive workloads
  - Frequently updated tables and indexes
  - 20X more write IOPS than disk on X3
  - 10X more write IOPs than disk on V2 and X2
- Database writes go directly to flash cache
  - Block is kept in cache until it LRU's out
  - Could be months or years
  - While it is cached, reads or writes will be serviced from cache
  - If the block stops being accessed, block will eventually age out of cache and will be written to disk
- Smart Caching applies
  - For example RMAN Backup and Data Pump reads and writes are not cached in flash

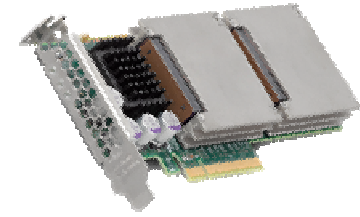
# Smart Flash Cache Write-Back Availability

- Write-back cache is persistent across reboots
  - Accelerates crash recovery
    - No need to reload or recover flash cache
  - Write-back speeds redo apply for Recovery and Data Guard
    - Redo apply is write IO intensive
- Flash Card failure is transparently and automatically handled by Exadata and ASM software
  - Writes are mirrored across cells, so a database write I/O becomes a flash write to 2 or 3 storage servers
  - No interruption to application on flash card failure
  - Contents of cache are recovered from mirrors on other cells
- No Administration is needed
  - Flash cache is transparent to Database, ASM, RMAN
  - As always, RMAN Backups are at the database level
  - Automatically backs up latest data on disk and on flash

**Writes I/Os**

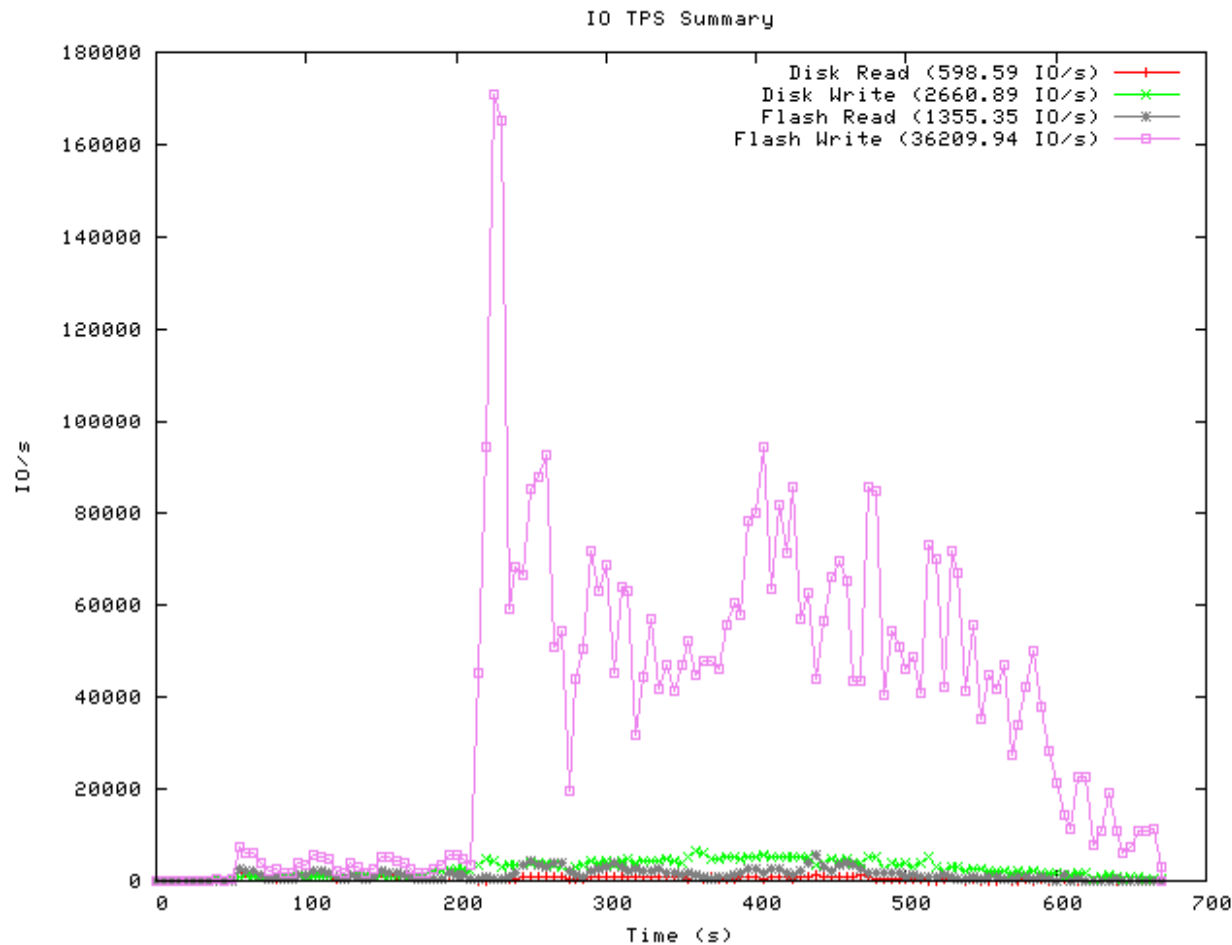


# Enabling Write-Back Cache



- Write-back flash cache is not enabled by default
  - Before enabling write-back cache, ASM must be upgraded
    - DB Patch for Exadata 11.2.0.3 BP 9 or later
  - Enabling write-back is an informed choice
    - Not required just because Exadata software update is applied
- Enable write-back cache on Exadata systems with correct ASM version that have write I/Os as a performance bottleneck
  - The best way to determine a write bottleneck is to look for “free buffer waits” in database wait event statistics
  - Can also check for high disk IO latencies and a large percentage of writes
  - Enable using: “Alter Cell flashCacheMode=WriteBack”

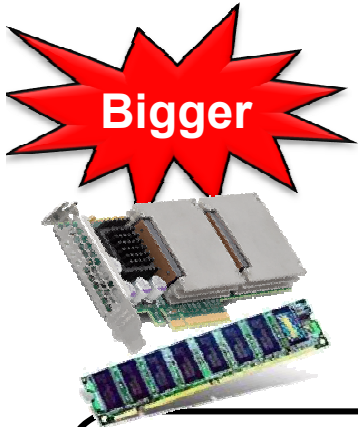
# Real Workload Write Cache Example



- Peoplesoft Batch Workload
- X3 Quarter Rack
- Flash writes often in the 60K to 80K IOPs range
- Peak at over 165K IOPs
- Would need more than 3 full racks without write-back flash cache
- IO bound job became CPU and application bound

# Summary

**Bigger**



- Same Architecture as Exadata X2
- Latest Technologies and Advances
- Same Price
- Lower Cost Entry Level Eighth Rack



**Faster**



## All X3 Database Machines

**4X Larger Flash Memory**

**22 TB of Flash Memory per Rack**

**20X More Write Performance**

**Exadata Smart Flash Write Caching**

**33% More Data Throughput**

**100 GB/sec running SQL**

**10% to 30% Lower Power**

**Up to 3 Kilowatt Reduction per Rack**

## X3-2 Database Servers

**33% Faster Database CPUs**

**8-Core Xeon® SandyBridge E5-2690**

**75% More Memory**

**1 TB to 2 TB per Rack**

**Full 10Gb Ethernet to Data Center**

**40X 10Gb ports per Rack**



**ORACLE®**

## Oracle Exadata Software Features

# Exadata Hybrid Columnar Compression



# About Hybrid Columnar Compression

## Compression Unit



10x to 15x  
Reduction



## Hybrid Columnar Compressed Tables

- For data that is **BULK loaded** and queried
- Designed for data that is **NOT frequently updated**
- Designed for **LOW concurrency** environments
  - Transaction modifying a single row in a CU locks the entire CU
- Compressed tables allow using conventional DML (INSERT/UPDATE/DELETE)
- All index types are supported (B-Tree, Bitmap)

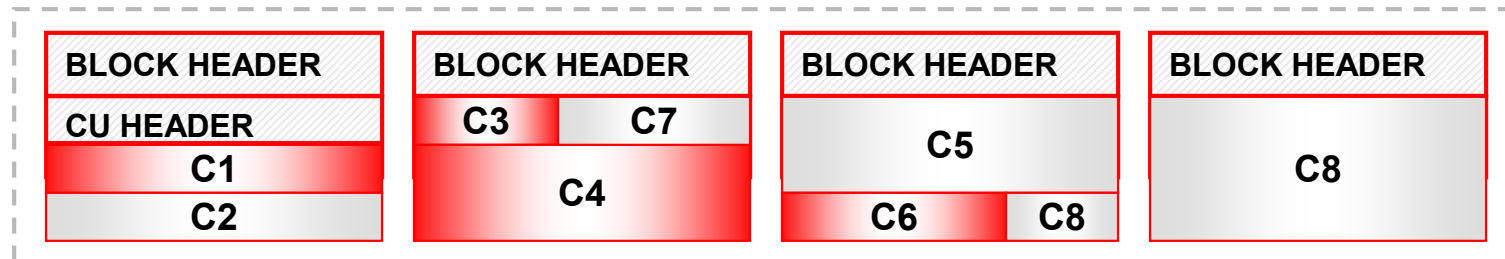


# Compression Units

- **Compression Unit**

- Logical structure spanning multiple database blocks
- Data organized by column during data load
- Each column compressed separately
- All column data for a set of rows stored in compression unit
- Column organization brings similar values close together, enhancing compression

## Logical Compression Unit



Hybrid Columnar Compression extended to Pillar Axiom and Sun ZFS Storage Appliance (ZFSSA) storage with DB 11.2.0.3

# Exadata Hybrid Columnar Compression

## Warehouse and Archive Compression

### Warehouse Compression

- 10x average storage savings
- 10x reduction in Scan IO

**Optimized for Speed**

**Smaller Warehouse  
Faster Performance**

### Archive Compression

- 15x average storage savings
  - Up to 70x on some data
- For cold or historical data

**Optimized for Space**

**Reclaim 93% of Disks  
Keep Data Online**

Can mix OLTP and hybrid columnar compression by partition for ILM



# Warehouse Compression

- **Warehouse Compression: LOW and HIGH**
  - HIGH typically provides a 10x reduction in storage
  - LOW typically provides a 6x reduction
- Both levels optimized to **increase scan query** performance by taking advantage of fewer number of blocks reads
- To maximize storage savings and query performance use **default level - HIGH**
  - LOW should be chosen for environments where **load times are more critical** than query performance



# Archive Compression

- **Archive Compression: LOW and HIGH**
  - HIGH typically provides a 15x reduction in storage
  - LOW typically provides a 10x reduction
- Best approach for **ILM and data archival**
  - Minimum storage footprint
  - Minimal access and update requirements
  - No need to move data to tape or less expensive disks
  - Data is always online and always accessible
    - Run queries against historical data (without recovering from tape)
    - Update historical data
    - Supports schema evolution (add/drop columns)



# EHCC DDL

- CTAS (create table as select)
  - create table foo **compress for query** as select \* from bar1;
- IAS (insert direct load)
  - create table foo **compress for archive low**;
  - insert /\*+APPEND\*/ into foo select \* from bar2;
- Compression can be specified at segment level
  - Each partition can have different compression type
  - create table orders (cid, pid, sid, price, discount, odate)  
partition by range (cid)  
(partition p1 values less than (100000) nocompress,  
partition p2 values less than (200000) **compress for archive low**,  
partition p3 values less than (300000) **compress for query high**,  
partition p4 values less than (maxvalue) **compress for query low**)  
enable row movement  
as select \* from prev\_orders;




# Efficient Data Storage

- EHCC is a feature of Exadata Storage
  - Decompression, selection and projection performed on storage
  - Data is stored compressed on disk and compressed in the Flash Cache, frequently accessed data cached on Flash
    - Table can be forced to be on Flash by setting *cell\_flash\_cache\_keep*
- EHCC is tightly integrated with Oracle DB 11gR2
  - Data is stored compressed in buffer cache (DRAM)
- With EHCC, entire databases can now run in memory
  - DRAM can hold 5TB of a compressed database
  - Flash can hold 50TB of a compressed database

# Exadata Hybrid Columnar Compression

**EXAMPLE**





```
SQL> alter session force parallel query;
SQL> alter session force parallel ddl;
SQL> alter session force parallel dml;
```

----- **CTAS PERFORMANCE**

```
SQL> create table sales_nocompress parallel 4 nologging
  2 as select * from sales_big;
Elapsed: 00:00:28.07 ←
```

```
SQL> create table sales_query compress for query high
  2 parallel 4 nologging as select * from sales_big;
Elapsed: 00:00:05.15 ←
```

```
SQL> create table sales_archive compress for archive high
  2 parallel 4 nologging as select * from sales_big;
Elapsed: 00:00:12.40 ←
```




--- COMPRESSION FACTOR

```
SQL> select table_name, compression, compress_for from user_tables
2 where table_name like 'SALES_%';
```

TABLE_NAME	COMPRESS	COMPRESS_FOR
SALES_ARCHIVE	ENABLED	ARCHIVE HIGH
SALES_QUERY	ENABLED	QUERY HIGH
SALES_NOCOMPRESS	DISABLED	

```
SQL> select segment_name, sum(bytes)/1024/1024 MB
2 from user_segments
3 where segment_name like 'SALES_%' and
4 segment_type='TABLE'
5 group by segment_name;
```

SEGMENT_NAME	MB
SALES_ARCHIVE	30,4 ← 16X
SALES_NOCOMPRESS	505
SALES_QUERY	36,9 ← 13X



----- **QUERY PERFORMANCE**

```
SQL> select count(amount_sold), sum(amount_sold) from sales_nocompress
  2  where time_id between to_date('01/01/1998','dd/mm/yyyy') and
  3  to_date('31/12/1999','dd/mm/yyyy')
  4  and quantity_sold = 1;
```

...

Elapsed: 00:00:00.66 ←

```
SQL> select count(amount_sold), sum(amount_sold) from sales_query
  2  where time_id between to_date('01/01/1998','dd/mm/yyyy') and
  3  to_date('31/12/1999','dd/mm/yyyy')
  4  and quantity_sold = 1;
```

...

Elapsed: 00:00:00.62 ←

```
SQL> select count(amount_sold), sum(amount_sold) from sales_archive
  2  where time_id between to_date('01/01/1998','dd/mm/yyyy') and
  3  to_date('31/12/1999','dd/mm/yyyy')
  4  and quantity_sold = 1;
```

...

Elapsed: 00:00:00.58 ←



---- **UPDATE PERFORMANCE**


```
SQL> update sales_nocompress set quantity_sold=quantity_sold+4
  2  where time_id between to_date('01/01/1998','dd/mm/yyyy') and
     to_date('31/12/1999','dd/mm/yyyy') and quantity_sold = 1;
```

426779 rows updated.

Elapsed: 00:00:00.82 


```
SQL> update sales_query set quantity_sold=quantity_sold+4
  2  where time_id between to_date('01/01/1998','dd/mm/yyyy') and
     to_date('31/12/1999','dd/mm/yyyy') and quantity_sold = 1;
```

426779 rows updated.

Elapsed: 00:00:06.04 

```
SQL> update sales_archive set quantity_sold=quantity_sold+4
  2  where time_id between to_date('01/01/1998','dd/mm/yyyy') and
     to_date('31/12/1999','dd/mm/yyyy') and quantity_sold = 1;
```

426779 rows updated.

Elapsed: 00:00:14.72 

----- UPDATE IMPACT ON COMPRESSION FACTOR

```
SQL> select segment_name, sum(bytes)/1024/1024 MB
       2  from user_segments
       3  where segment_name like 'SALES_%' and
       4  segment_type='TABLE'
       5  group by segment_name;
```

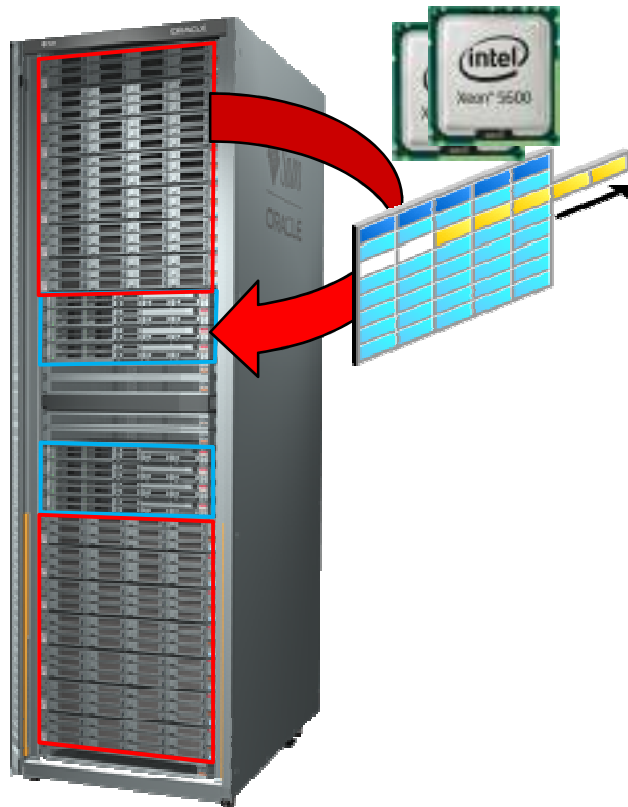
SEGMENT_NAME	MB	
-----	-----	
SALES_QUERY	80,9	← initially 30,4
SALES_NOCOMPRESS	505	
SALES_ARCHIVE	72,4	← initially 36,9



# Exadata Smart Storage Capabilities

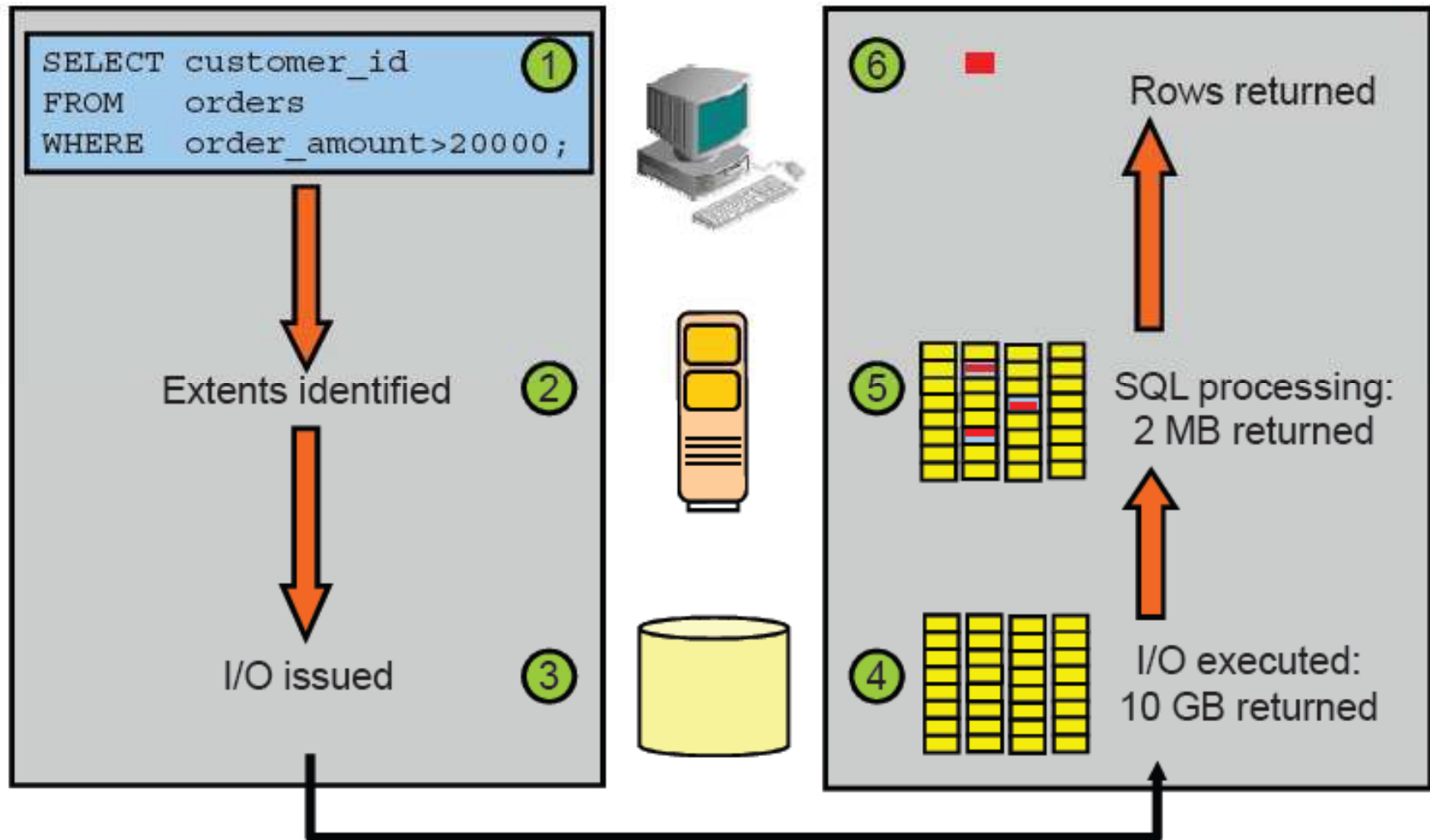
# Exadata Intelligent Storage

## Scalable Data Processing

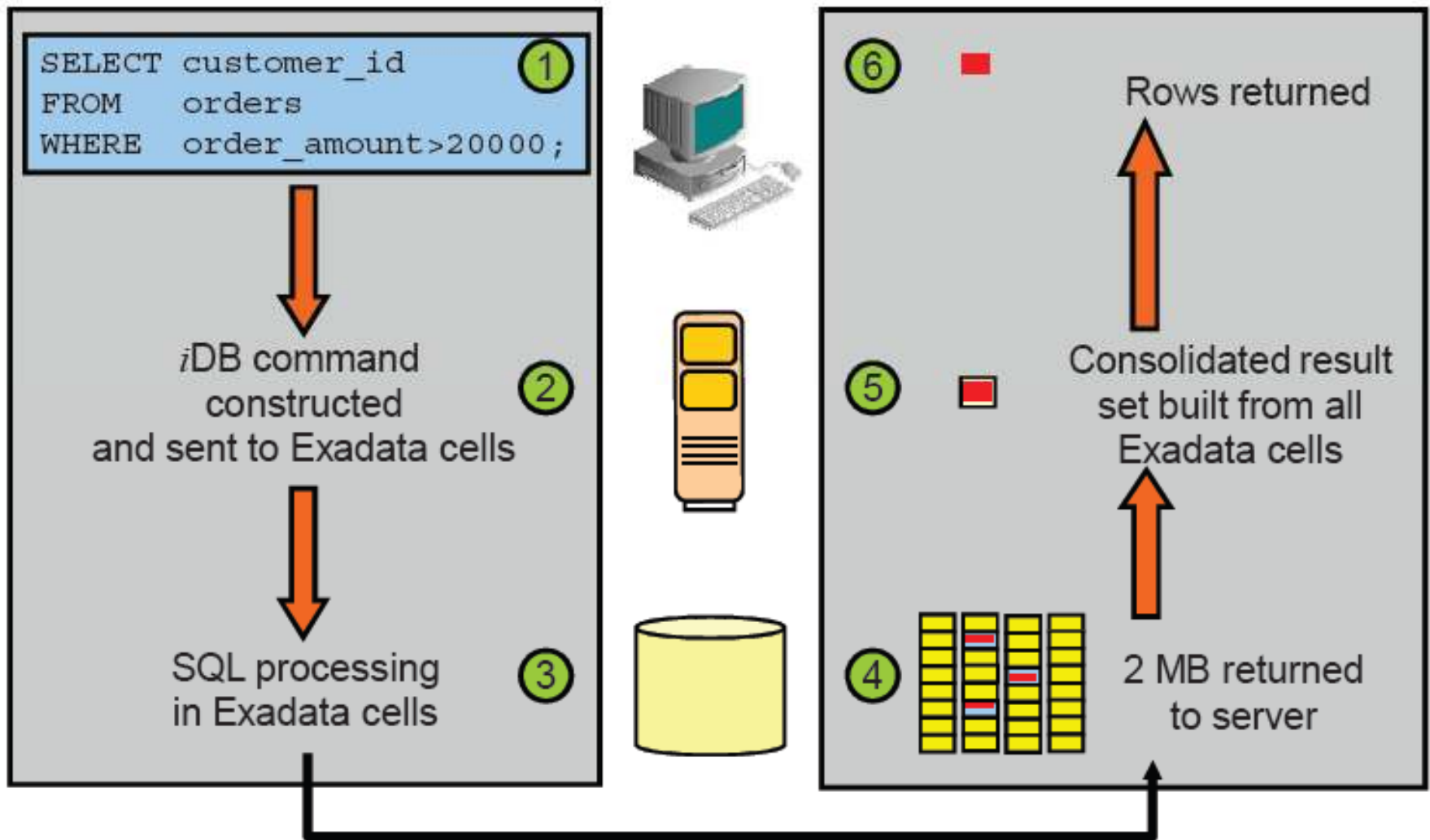


- Data Intensive processing runs in Exadata Storage Grid
  - Filter rows and columns as data streams from disks
- Example: How much product X sold last quarter
  - Exadata Storage Reads 10TB from disk
  - Exadata Storage Filters rows by Product & Date
  - Sends 100GB of matching data to DB Servers
- Scale-out storage **parallelizes** execution and removes bottlenecks

# Classic Database I/O & SQL Processing Model



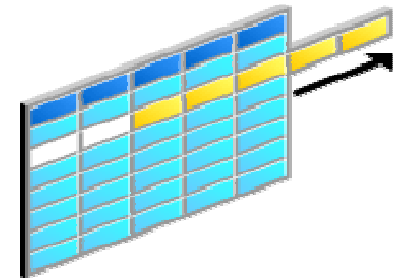
# Exadata Smart Scan Model





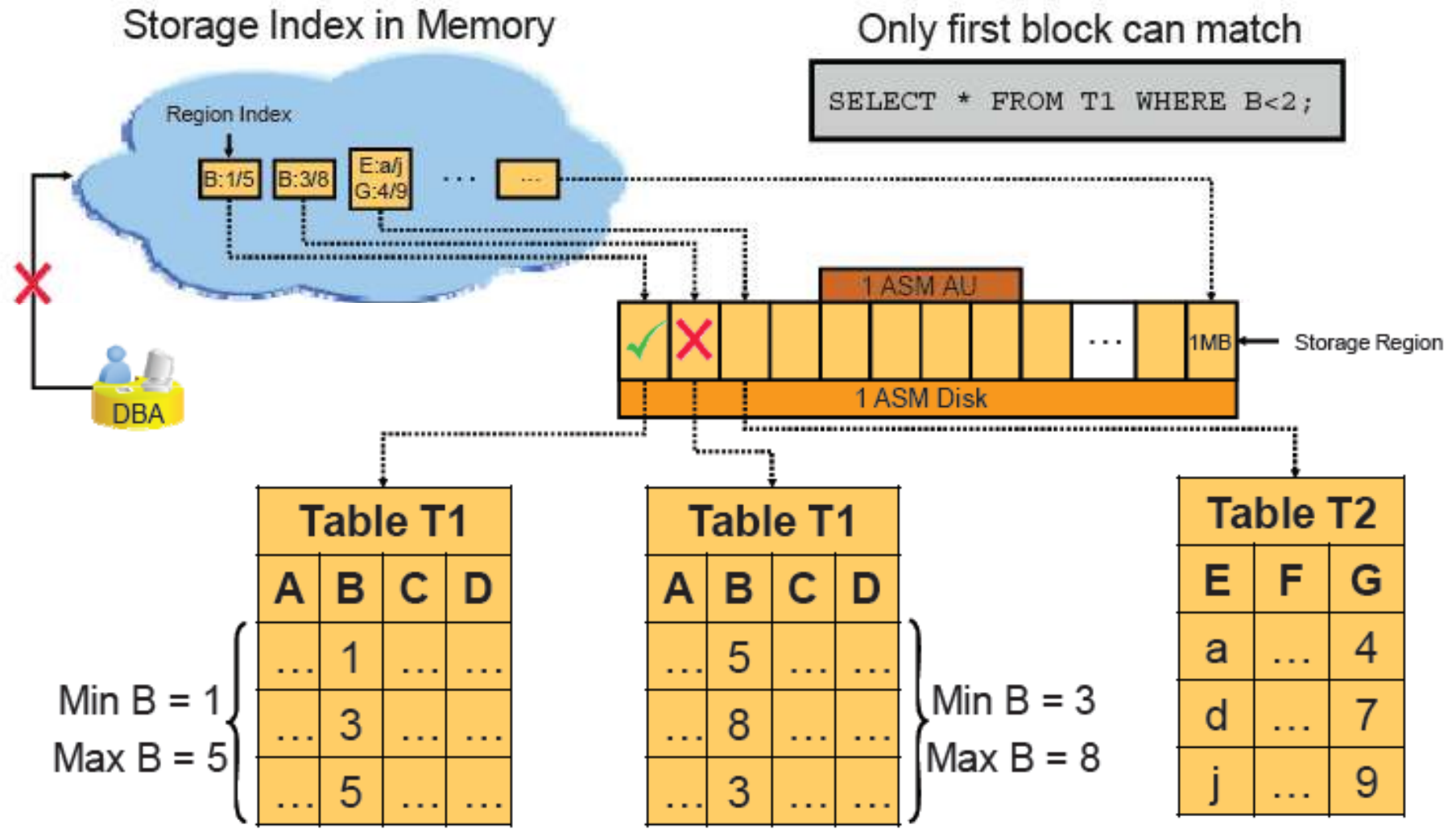
# Exadata Intelligent Storage

- Exadata implements data intensive processing in storage
  - Row filtering based on “where” predicate
  - Column filtering
  - Join filtering
  - Incremental backup filtering
  - Scans on Hybrid Columnar Compressed data
  - Scans on encrypted data
- 10x reduction in data sent to DB servers is common
- No application changes needed
  - Processing is automatic and transparent
  - Even if cell or disk fails during a query



# Exadata Storage Index

Transparent I/O Elimination with No Overhead





# Putting it all together

- Storage Index is a Filter (which prunes away unnecessary IOs)
- Is maintained **automatically** and is **transparent** to DB
- Collection of **in-memory** region indexes (RIDX)
  - *If cell restarts, SI is rebuilt by the next set of queries*
- One region index (RIDX) for every **1MB** of disk for **up to 8 columns** (Exadata determines which)
- Works with uncompressed tables, OLTP compression, HCC, tablespace encryption (not column level encryption)
- Would be most effective for clustered columns
  - *One option is to load data in sorted order*
- Works for most data-types (number, float, date, varchar2 ...)
- Works for simple predicates
  - Can evaluate <, <=, =, !=, >=, >, is NULL, is NOT NULL
  - column < 9 (literal)
  - column > :BINDVAR



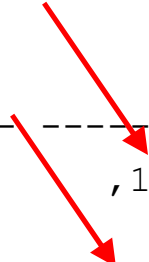
# Exadata Smart Scan & Storage Indexes


**EXAMPLE**

```
----- SMART SCAN DISABLED
----- SESSION IO STATS BEFORE QUERY
```

```
SQL> select a.name, b.value/1024/1024 MB
       2 from v$sysstat a, v$mystat b
       3 where a.statistic# = b.statistic# and
       4 (a.name in ('physical read total bytes',
       5 'physical write total bytes',
       6 'cell IO uncompressed bytes')
       7 or a.name like 'cell phy%');
```

NAME	MB
-----	-----
physical read total bytes	,109375
physical write total bytes	0
cell physical IO interconnect bytes	,109375
cell physical IO bytes sent directly to DB node to balanceCPU u	0
cell physical IO bytes saved during optimized file creation	0
cell physical IO bytes saved during optimized RMAN file restore	0
cell physical IO bytes eligible for predicate offload	0
cell physical IO bytes saved by storage index	0
cell physical IO interconnect bytes returned by smart scan	0
cell IO uncompressed bytes	0





```
SQL> -- Broj prodaja u 1998 za quantity_sold=1
SQL> -- ALTER SESSION SET CELL_OFFLOAD_PROCESSING = FALSE;
SQL>
SQL> select /*+ OPT_PARAM('cell_offload_processing' 'false') */ count(*)
  from sales_big
 2  where time_id between to_date('01/01/1998','dd/mm/yyyy') and
 3  to_date('31/12/1999','dd/mm/yyyy')
 4  and
 5  quantity_sold = 1;
```

```
      COUNT(*)
```

```
-----
      426779
```

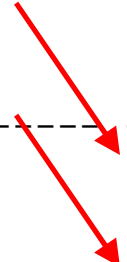
```
Elapsed: 00:00:04.63
```



----- SMART SCAN DISABLED  
----- SESSION IO STATS AFTER QUERY

```
SQL> select a.name, b.value/1024/1024 MB  
2 from v$sysstat a, v$mystat b  
3 where a.statistic# = b.statistic# and  
4 (a.name in ('physical read total bytes',  
5 'physical write total bytes',  
6 'cell IO uncompressed bytes')  
7 or a.name like 'cell phy%');
```

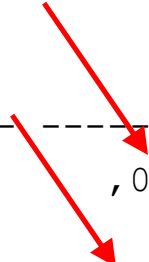
NAME	MB
-----	-----
physical read total bytes	172,703125
physical write total bytes	0
cell physical IO interconnect bytes	172,703125
cell physical IO bytes sent directly to DB node to balanceCPU u	0
cell physical IO bytes saved during optimized file creation	0
cell physical IO bytes saved during optimized RMAN file restore	0
cell physical IO bytes eligible for predicate offload	0
cell physical IO bytes saved by storage index	0
cell physical IO interconnect bytes returned by smart scan	0
cell IO uncompressed bytes	0




```
----- SMART SCAN ENABLED
----- SESSION IO STATS BEFORE QUERY
```

```
SQL> select a.name, b.value/1024/1024 MB
       2 from v$sysstat a, v$mystat b
       3 where a.statistic# = b.statistic# and
       4 (a.name in ('physical read total bytes',
       5 'physical write total bytes',
       6 'cell IO uncompressed bytes')
       7 or a.name like 'cell phy%');
```

NAME	MB
-----	-----
physical read total bytes	,015625
physical write total bytes	0
cell physical IO interconnect bytes	,015625
cell physical IO bytes sent directly to DB node to balanceCPU u	0
cell physical IO bytes saved during optimized file creation	0
cell physical IO bytes saved during optimized RMAN file restore	0
cell physical IO bytes eligible for predicate offload	0
cell physical IO bytes saved by storage index	0
cell physical IO interconnect bytes returned by smart scan	0
cell IO uncompressed bytes	0







```
SQL> -- Broj prodaja u 1998 za quantity_sold=1
SQL> select count(*) from sales_big
  2  where time_id between to_date('01/01/1998','dd/mm/yyyy') and
  3  to_date('31/12/1999','dd/mm/yyyy')
  4  and
  5  quantity_sold = 1;
```

```
      COUNT(*)
-----
      426779
```

1 row selected.

Elapsed: 00:00:00.28  **Initially 00:00:04.63**

----- SMART SCAN ENABLED  
----- SESSION IO STATS AFTER QUERY

```
SQL> select a.name, b.value/1024/1024 MB
  2  from v$sysstat a, v$mystat b
  3  where a.statistic# = b.statistic# and
  4  (a.name in ('physical read total bytes',
  5  'physical write total bytes',
  6  'cell IO uncompressed bytes')
  7  or a.name like 'cell phy%');
```

NAME	MB
-----	-----
physical read total bytes	172,46875
physical write total bytes	0
cell physical IO interconnect bytes	4,91526794
cell physical IO bytes sent directly to DB node to balanceCPU u	0
cell physical IO bytes saved during optimized file creation	0
cell physical IO bytes saved during optimized RMAN file restore	0
cell physical IO bytes eligible for predicate offload	172,453125
cell physical IO bytes saved by storage index	<b>12,328125</b>
cell physical IO interconnect bytes returned by smart scan	<b>4,89964294</b>
cell IO uncompressed bytes	160,125

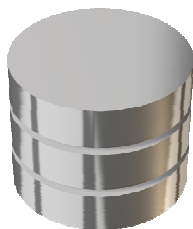
# Exadata Smart Flash Cache & Smart Flash Log



# Exadata Smart Flash Cache

## Breaks the Disk Random I/O Bottleneck

300 IOPS



10.000s IOPS



- Trade-off between disk and Flash
  - Disk - cheap, high capacity, low IOPS
  - Flash - expensive, lower capacity, tens of thousands IOPS
- Ideal Solution - Exadata Smart Flash Cache
  - Keep most data on disk for low cost
  - **Transparently move hot data to flash**
  - Use **flash cards** instead of flash disks to avoid disk controller limitations
- Flash cards in Exadata storage
  - 4 x 400GB PCI Express Flash Cards per Exadata Server

# Exadata Smart Flash Cache

## Intelligently manages flash

Understands different types of I/Os from database:

- Frequently accessed data and index blocks are cached
  - Control file reads and writes are cached
  - File header reads and writes are cached
  - DBA can influence caching priorities
- 
- I/Os to mirror copies are not cached
  - Backup-related I/O is not cached
  - Data Pump I/O is not cached
  - Data file formatting is not cached
  - Table scans do not monopolize the cache





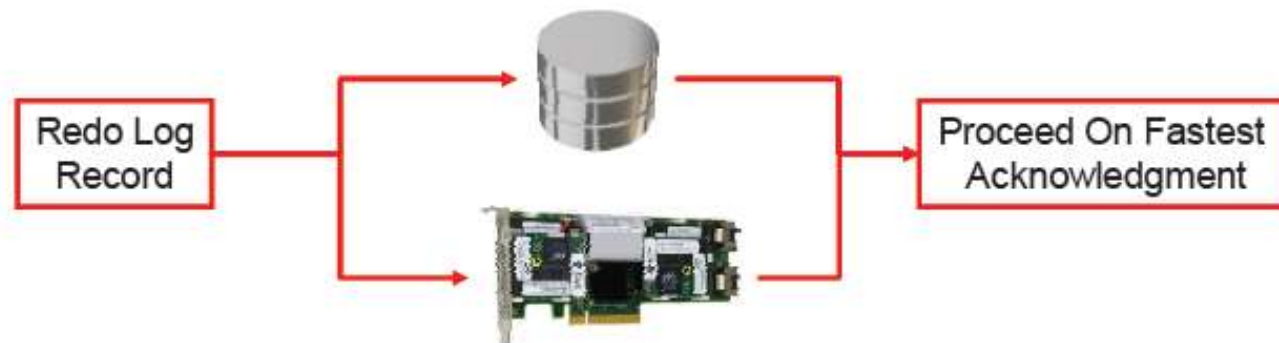
# Exadata Smart Flash Cache

- CELL\_FLASH\_CACHE setting for the object
  - DEFAULT – managed by LRU
  - KEEP - 80% of the total cache size
  - NONE
- Cache hint:
  - CACHE indicates that the I/O should be cached: I/O is for an index lookup.
  - NOCACHE indicates that the I/O should not be cached: I/O is for a mirrored block of data or is a log write.
  - EVICT
- Large I/Os on objects with CELL\_FLASH\_CACHE set to DEFAULT are not cached.
- Smart table scans are usually directed to disk. If object has a CELL\_FLASH\_CACHE KEEP, some reads may be satisfied from Flash

# Smart Flash Logging

High-performance, low-latency, reliable temporary store for redo log writes:

- Log writes are directed BOTH to disk and Exadata Smart Flash Log.
- Processing continues after fastest acknowledgement
- It is conceptually similar to multiplexed redo logs
- Exadata Storage Server automatically manages Smart Flash Log and ensures that all log entries are persisted to disk
- By default uses 32 MB on each flash-based cell disk, total of 512 MB on each Exadata Storage Server.



# Exadata Smart Flash Log

## Accelerate Transaction Response Times using Flash

Default (on left)

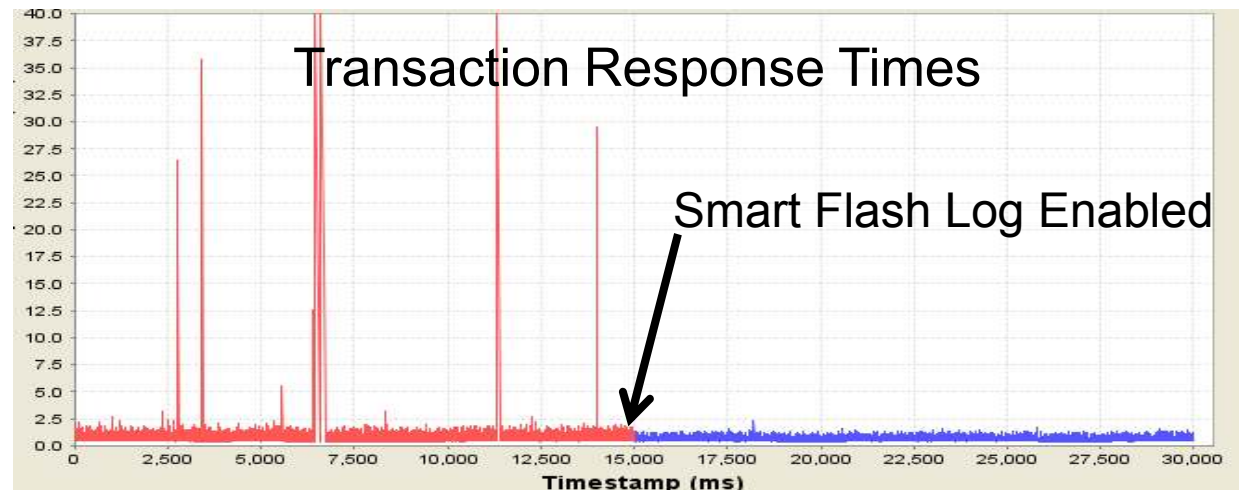
- Choppy response
- High Outliers

Smart Flash Log

- **3x faster** response
- Much lower outliers



**Automatic and  
Transparent**



- Transparently uses Flash as a **parallel write cache** to disk controller cache
  - Whichever write completes first wins (disk or flash)
- Uses almost no flash capacity (0.1% of capacity)
- Reduces response time and outliers
  - “log file parallel write” AWR histogram improves
  - Greatly improves “log file sync”





# What Smart Flash Logging is and is not

- It is solely a mechanism for providing low latency redo log writes.
- It is not a mirroring scheme because we don't keep a full copy of the entire redo log; we only mirror the tail/end of the redo log (this also enables us to use a relatively small amount of flash disk space, which allows the bulk of the remaining flash disk space to be used for Smart Flash Cache).
- It is not a cache because we don't satisfy read requests from flash disk; redo log reads are always done from hard disk.



# Exadata Smart Flash Cache

## EXAMPLE

----- SESSION IO STATS BEFORE WORKLOAD

```
SQL> select a.name, b.value from v$sysstat a, v$mystat b
  2  where a.statistic# = b.statistic# and
  3    (a.name like '%flash cache read hits'
  4  or a.name like 'cell phy%'
  5  or a.name like 'physical read tot%'
  6  or a.name like 'physical read req%');
```

NAME	VALUE
physical read total IO requests	58
physical read total multi block requests	0
physical read requests optimized	56
physical read total bytes optimized	917504
physical read total bytes	950272
cell physical IO interconnect bytes	950272
cell physical IO bytes sent directly to DB node to balanceCPU u	0
cell physical IO bytes saved during optimized file creation	0
cell physical IO bytes saved during optimized RMAN file restore	0
cell physical IO bytes eligible for predicate offload	0
cell physical IO bytes saved by storage index	0
cell physical IO interconnect bytes returned by smart scan	0
cell flash cache read hits	56





```
----- SIMULATE OLTP WORKLOAD
----- FLUSH BUFFER CACHE !!!
```

```
SQL> declare
  2  a number;
  3  s number := 0;
  4  begin
  5  for n in 1 .. 14000 loop
  6  select cust_credit_limit into a from customers
  7  where cust_id=n*2;
  8  s := s+a;
  9  end loop;
 10  dbms_output.put_line('Transaction total = '||s);
 11  end;
 12  /
```

```
Transaction total = 83939500
```

```
Elapsed: 00:00:01.17
```



----- SESSION IO STATS AFTER WORKLOAD

```
SQL> select a.name, b.value from v$sysstat a, v$mystat b
2  where a.statistic# = b.statistic# and
3  (a.name like '%flash cache read hits'
4  or a.name like 'cell phy%'
5  or a.name like 'physical read tot%'
6  or a.name like 'physical read req%');
```

NAME	VALUE
physical read total IO requests	800
physical read total multi block requests	0
physical read requests optimized	463
physical read total bytes optimized	7585792
physical read total bytes	13107200
cell physical IO interconnect bytes	13107200
cell physical IO bytes sent directly to DB node to balanceCPU u	0
cell physical IO bytes saved during optimized file creation	0
cell physical IO bytes saved during optimized RMAN file restore	0
cell physical IO bytes eligible for predicate offload	0
cell physical IO bytes saved by storage index	0
cell physical IO interconnect bytes returned by smart scan	0
cell flash cache read hits	463



----- SESSION IO STATS BEFORE WORKLOAD

```
SQL> select a.name, b.value from v$sysstat a, v$mystat b
  2  where a.statistic# = b.statistic# and
  3    (a.name like '%flash cache read hits'
  4  or a.name like 'cell phy%'
  5  or a.name like 'physical read tot%'
  6  or a.name like 'physical read req%');
```

NAME	VALUE
physical read total IO requests	14
physical read total multi block requests	0
physical read requests optimized	12
physical read total bytes optimized	196608
physical read total bytes	229376
cell physical IO interconnect bytes	229376
cell physical IO bytes sent directly to DB node to balanceCPU u	0
cell physical IO bytes saved during optimized file creation	0
cell physical IO bytes saved during optimized RMAN file restore	0
cell physical IO bytes eligible for predicate offload	0
cell physical IO bytes saved by storage index	0
cell physical IO interconnect bytes returned by smart scan	0
cell flash cache read hits	12





```
----- SIMULATE OLTP WORKLOAD
----- FLUSH BUFFER CACHE !!!
```

```
SQL> declare
  2  a number;
  3  s number := 0;
  4  begin
  5  for n in 1 .. 14000 loop
  6  select cust_credit_limit into a from customers
  7  where cust_id=n*2;
  8  s := s+a;
  9  end loop;
 10  dbms_output.put_line('Transaction total = '||s);
 11  end;
 12  /
```

```
Transaction total = 83939500
```

```
Elapsed: 00:00:00.98
```



----- SESSION IO STATS AFTER WORKLOAD

```
SQL> select a.name, b.value from v$sysstat a, v$mystat b
  2  where a.statistic# = b.statistic# and
  3    (a.name like '%flash cache read hits'
  4  or a.name like 'cell phy%'
  5  or a.name like 'physical read tot%'
  6  or a.name like 'physical read req%');
```

NAME	VALUE
physical read total IO requests	756
physical read total multi block requests	0
physical read requests optimized	733
physical read total bytes optimized	12009472
physical read total bytes	12386304
cell physical IO interconnect bytes	12386304
cell physical IO bytes sent directly to DB node to balanceCPU u	0
cell physical IO bytes saved during optimized file creation	0
cell physical IO bytes saved during optimized RMAN file restore	0
cell physical IO bytes eligible for predicate offload	0
cell physical IO bytes saved by storage index	0
cell physical IO interconnect bytes returned by smart scan	0
cell flash cache read hits	733







# **Hardware and Software Engineered to Work Together**