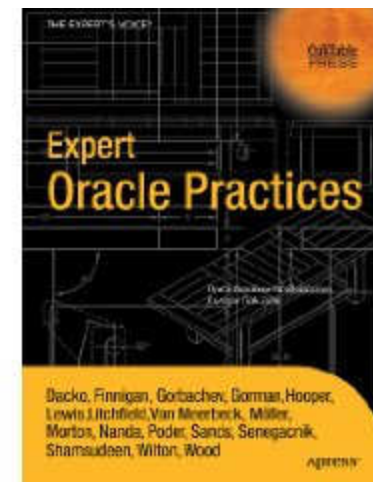

Advanced RAC troubleshooting

By
Riyaj Shamsudeen



Who am I?

- 17 years using Oracle products/DBA
- OakTable member
- Oracle ACE
- Certified DBA versions 7.0,7.3,8,8i &9i
- Specializes in RAC, performance tuning, Internals and E-business suite
- Chief DBA with OraInternals
- Co-author of “Expert Oracle Practices” ‘2010
- Email: rshamsud at gmail.com
- Blog : orainternals.wordpress.com



Disclaimer

These slides and materials represent the work and opinions of the author and do not constitute official positions of my current or past employer or any other organization. This material has been peer reviewed, but author assume no responsibility whatsoever for the test cases.

If you corrupt your databases by running my scripts, you are solely responsible for that.

This material should not be reproduced or used without the authors' written permission.

Agenda

- **Global cache performance**
- **Few important RAC wait events**
- **RAC background process tuning**
- **Interconnect issues, lost packets and network layer**
- **Network layer tuning**
- **Effective use of parallel query**
- **Troubleshooting locking issues**
- **Object re-mastering**

Types of packets

- Block oriented packets
 - Consistent Read blocks
 - Current Read blocks
- Message oriented packets
 - Single block grants
 - Multi block grants
- Service oriented packets
 - SCN generation
 - Row cache updates
 - GES layer packets

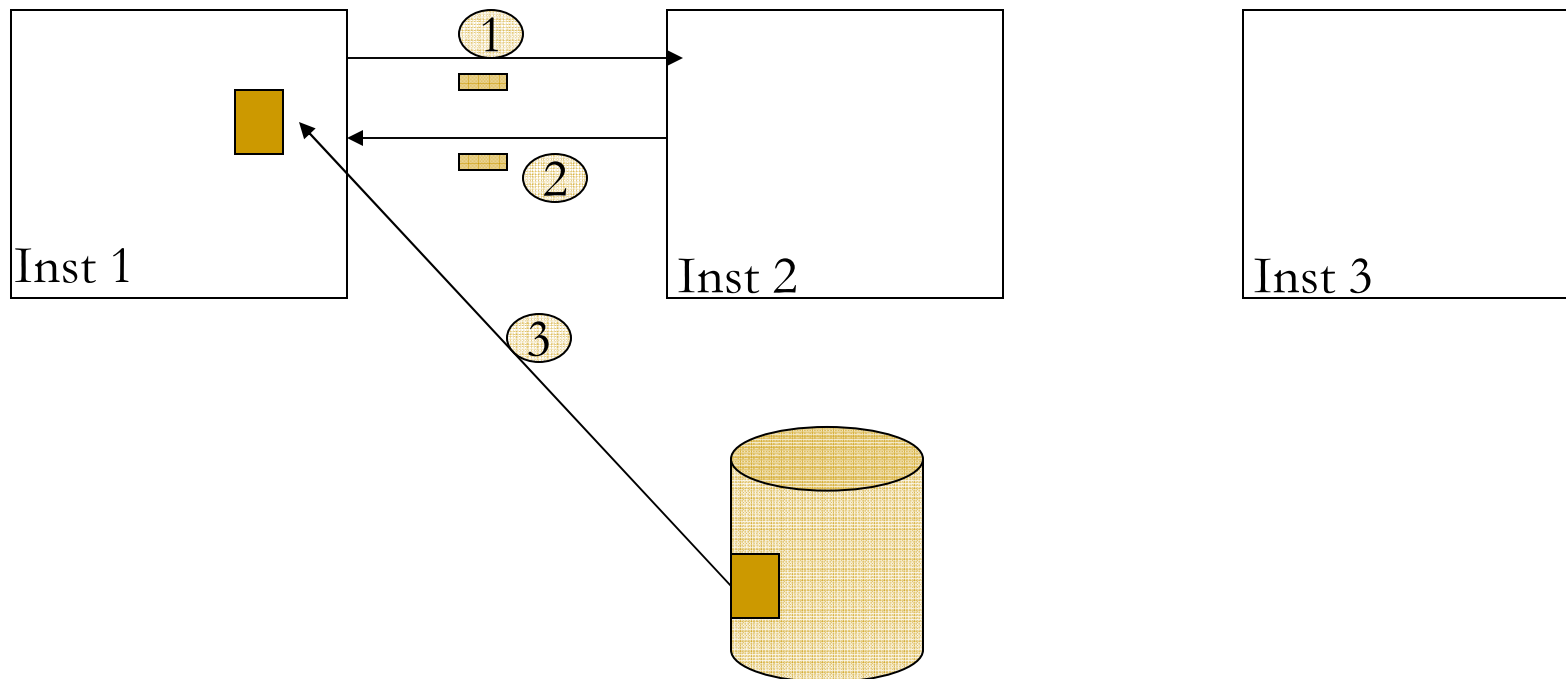
Block oriented packets

- Consistent Read (CR) blocks
 - Blocks requested by other nodes for mostly read operations.
 - User processes running in local node requests LMS processes running in the remote nodes for a block or set of blocks.
 - LMS might need to apply undo blocks to reconstruct requested version of the blocks.

CR – disk read

```
Select c1 from t1 where n1 =:b1;
```

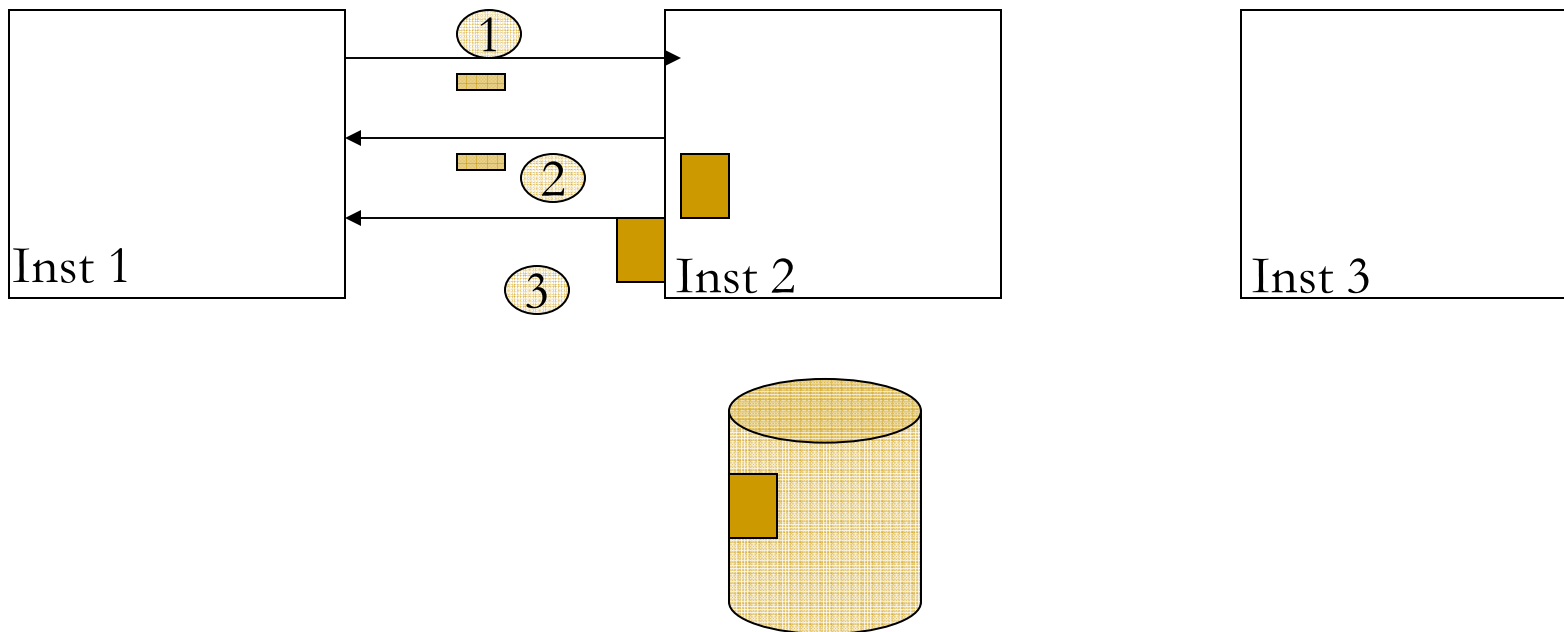
- ① User process in instance 1 requests master for a PR mode block.
- ② Assuming no current owner, master node grants the lock to inst 1.
- ③ User process in instance 1 reads the block from the disk and holds PR.



CR immediate 2-way

```
Select c1 from t1 where n1=:b1;
```

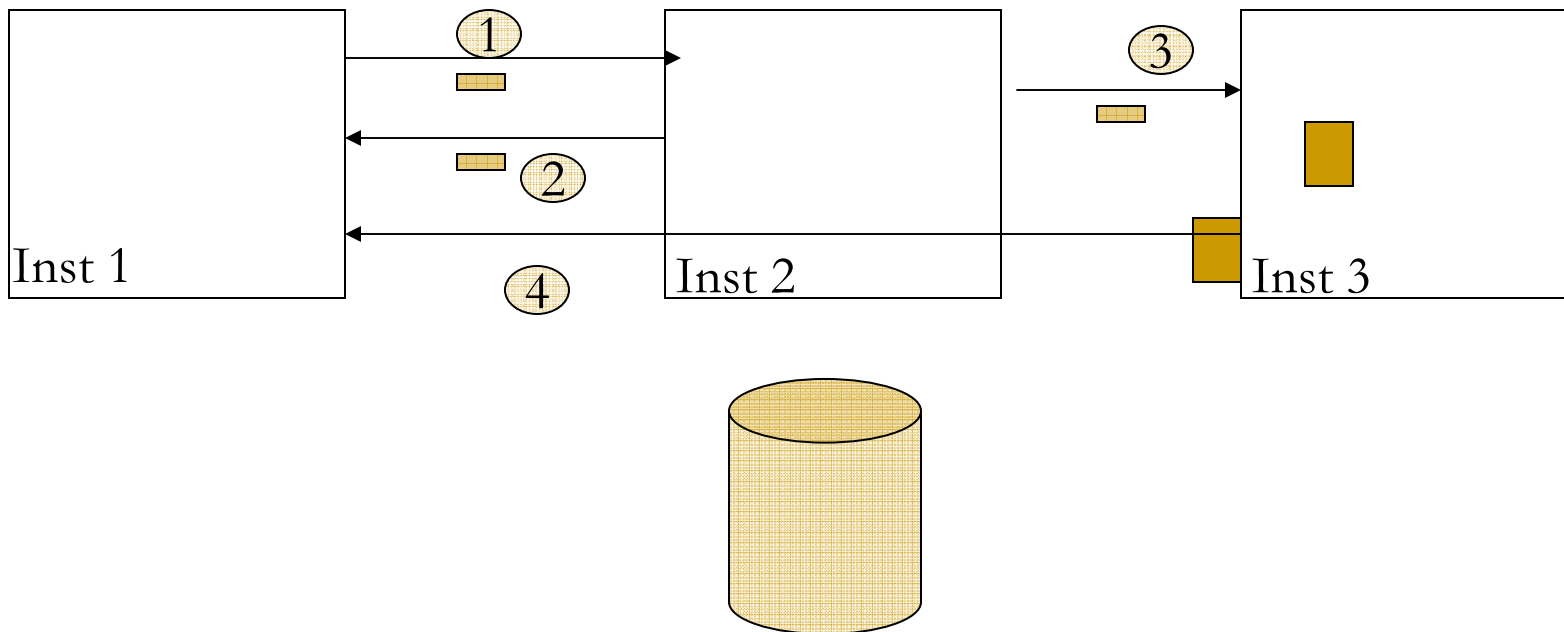
- ① User process in instance 1 requests master for the block in PR mode.
- ② Current owner (2) holds the block Protected Read (PR) and so master grants a PR to instance 1.
- ③ In this case master instance and owner instance are the same. Owner instance LMS process serves the block to the user process in instance 1.



CR immediate 3-way

Select c1 from t1 where
N1 =:b1;

- ① User process in instance 1 requests master for the block.
- ② Current owner holds Protected Read (PR) and so master grants a PR to instance 1.
- ③ Master sends the owner to send the block to instance 1.
- ④ Owner instance LMS process serves the block to the user process in instance 1



CR complications

- There are lot more complications then I covered here:
 - gcs log flush
 - What if the block has uncommitted transaction?
 - What if the block SCN is ahead the query environment SCN?
 - What if the block is undergoing changes and the buffer is pinned by another process?
 - What if the block is transaction table, undo block?
 - What if the block is index branch block that was just split?

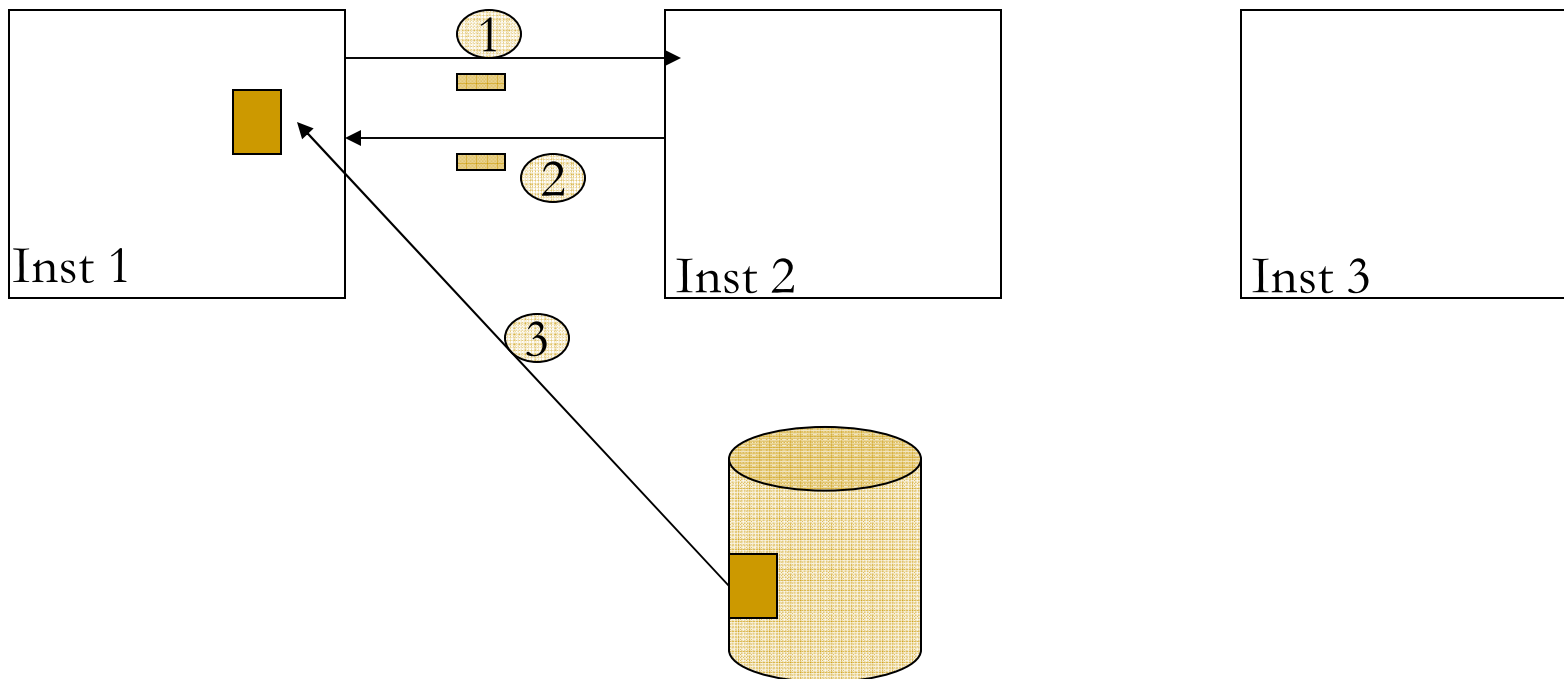
Block oriented packets

- CURRENT mode blocks
 - Blocks requested by other nodes to modify the blocks.
 - LMS will send the current mode block to the requesting process.
 - Converts the existing block to Past Image.

CUR – disk read

Update t1 set c1=:b1 where
T1 =:b2;

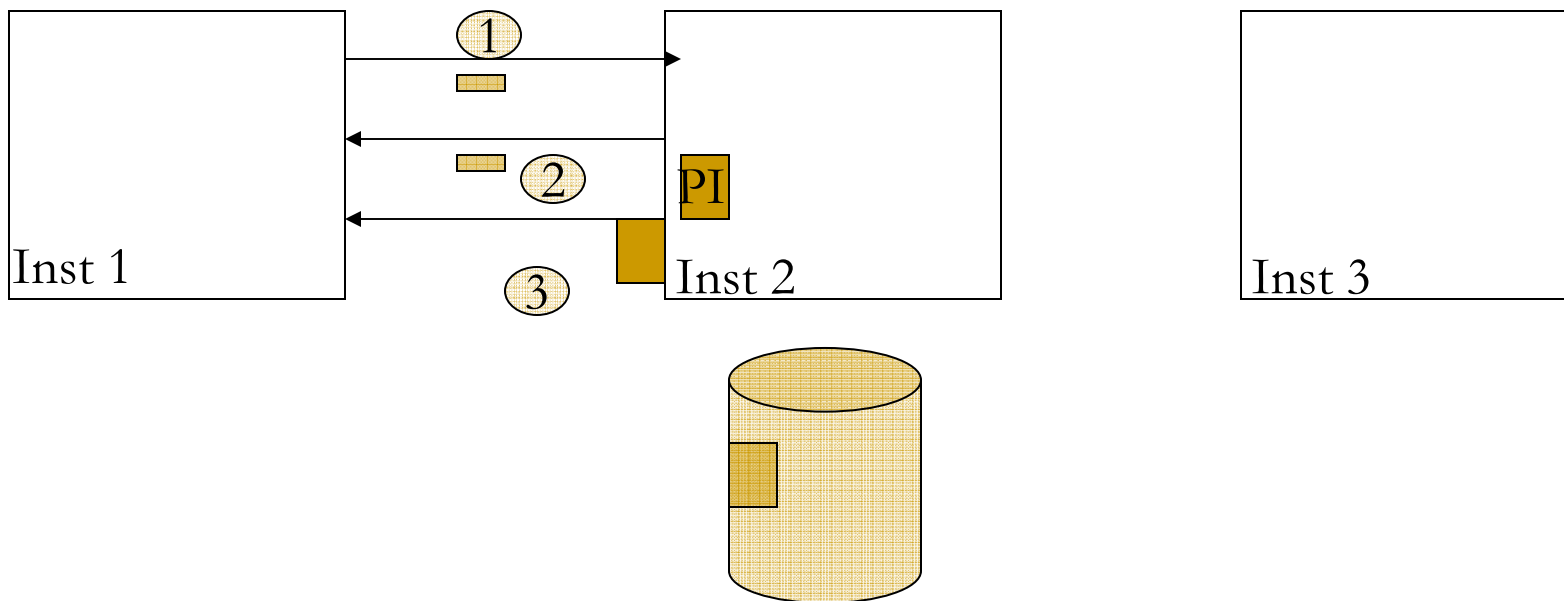
- ① User process in instance 1 requests master for a Exclusive grant.
- ② Assuming no current owner, master node grants the request.
- ③ User process in instance 1 reads the block from the disk.



CUR immediate 2-way

Update t1 set c1=:b1 where
T1 =:b2;

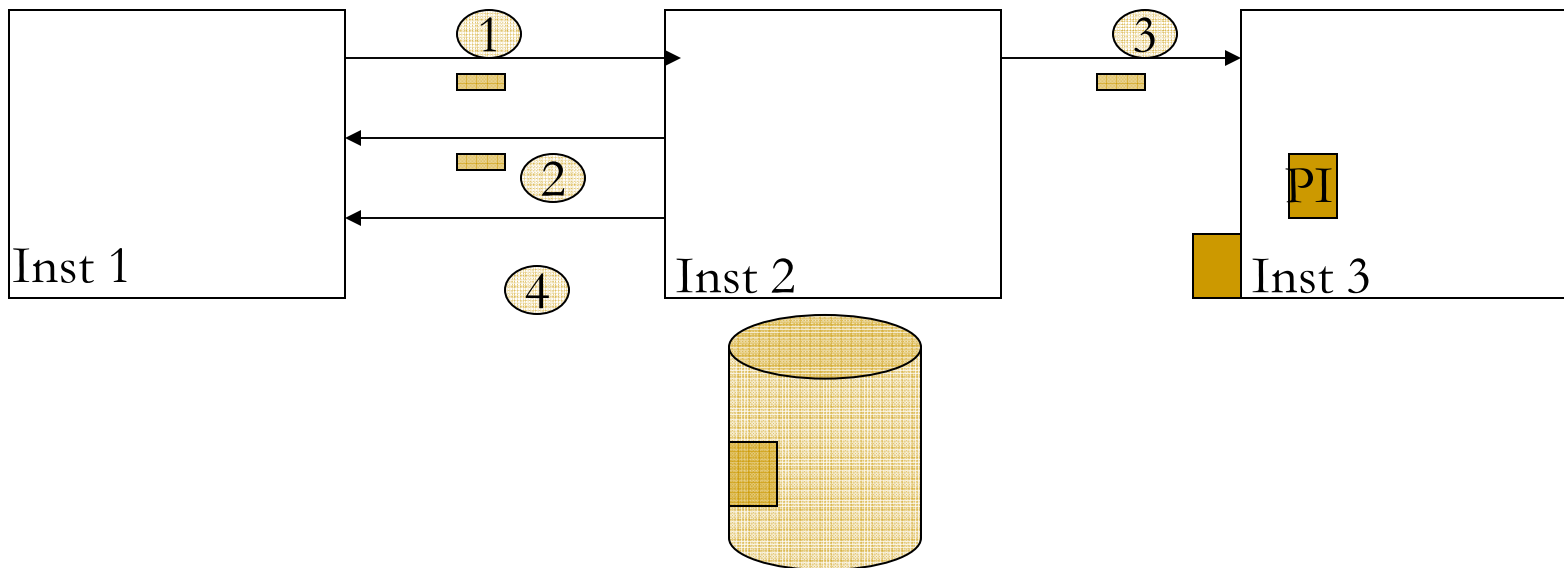
- ① User process in instance 1 requests master for Exclusive grant on the block.
- ② Current owner (2) holds Exclusive mode lock on that block.
- ③ In this case master instance and owner instance are the same. Owner instance LMS process serves the block to the user process in instance1. Block in instance 2 is marked as PI (Past Image).



CUR immediate 3-way

Update t1 set c1=:b1 where
T1 =:b2;

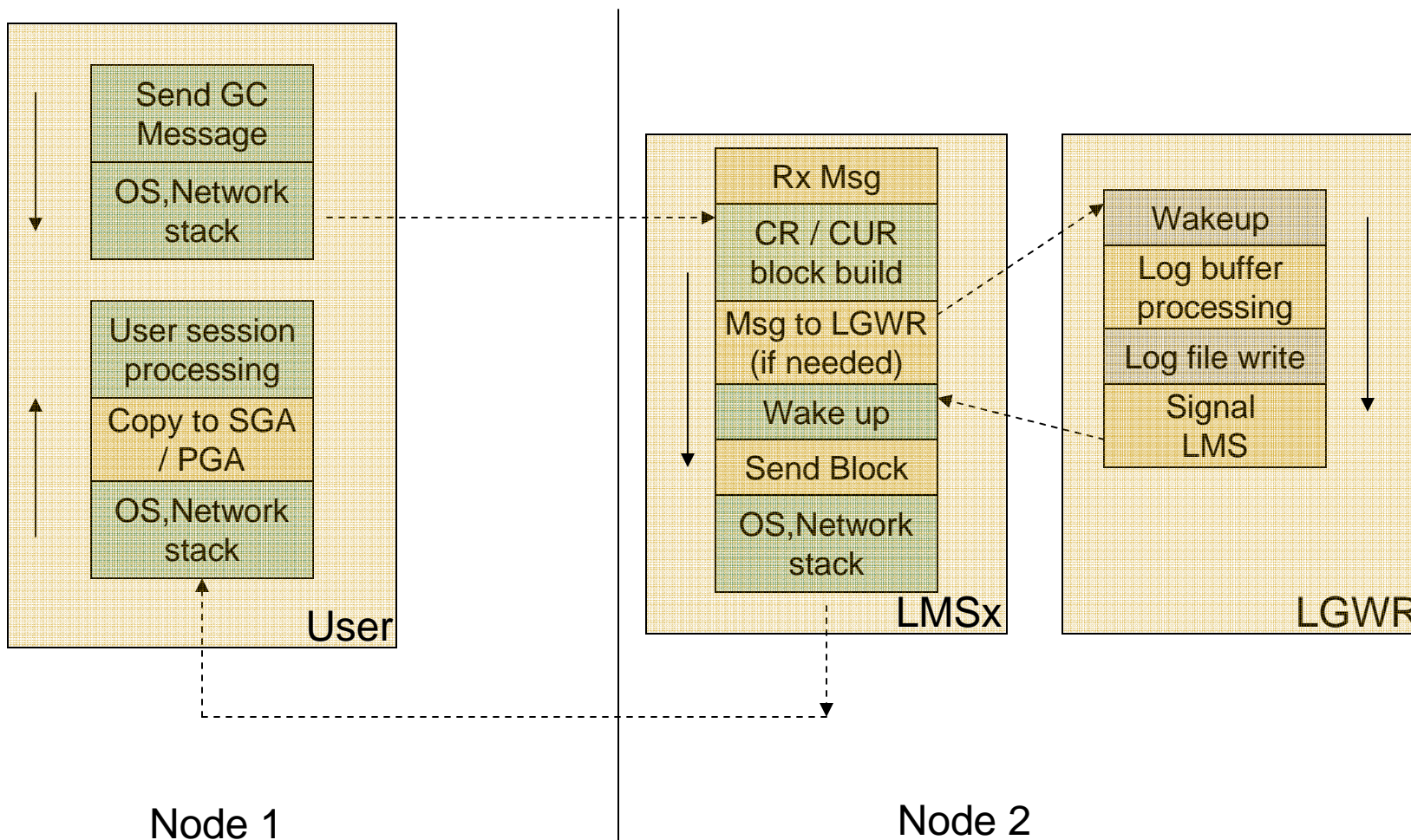
- ① User process in instance 1 requests master for Exclusive grant on the block.
- ② Current owner (3) holds Exclusive mode lock on that block.
- ③ Master (instance 2) requests instance 3 to ship the block to instance 1.
- ④ In this case master instance and owner instance are the same. Owner instance LMS process serves the block to the user process in instance 1. Block in instance 2 is marked as PI (Past Image).



Gcs log flush sync

- But, if the instances crash right after the block is transferred to other node, how does RAC maintain consistency?
- Actually, before sending a current mode block LMS process will request LGWR for a log flush.
- Until LGWR sends a signal back to LMS process, LMS process will wait on 'gcs log flush' event.
- CR block transfer might need log flush if the block was considered "busy".
- One of the busy condition is that if the block was constructed by applying undo records.

LMS Processing (over simplified)



GC CR latency

- GC CR latency \approx

Time spent in sending message to LMS +
LMS processing (building blocks etc) +
LGWR latency (if any) +
LMS send time +
Wire latency

Averages can be misleading. Always review both total time and average to understand the issue.

Breakdown latency

In this case, LGWR flush time
Need to be reduced to tune latency.

Avg global cache cr block receive time (ms): 6.2

Wait time	Node 1	Node 2	Node 3	Node 4	Total
gc cr block build time	402	199	100	227	1679
Gc cr block flush time	3016	870	978	2247	7111
Gc cr block send time	375	188	87	265	1290

GC CURRENT latency

- GC CUR latency \approx

Time spent in sending message to LMS +
LMS processing : (Pin and build block) +
LGWR latency: Log flush +
Wire latency

Statistics : gc current block flush time
gc current block pin time
gc current block send time

Caution

- Don't use gv\$views to find averages. Use AWR reports or custom scripts.
- gv\$views are aggregated data and persistent from the instance restart.
- For example this query can be misleading:

```
select b1.inst_id, b2.value "RECEIVED",  
       b1.value "RECEIVE TIME",  
       ((b1.value / b2.value) * 10) "AVG RECEIVE TIME (ms)"  
from gv$sysstat b1, gv$sysstat b2  
where b1.name = 'gc cr block receive time' and  
       b2.name = 'gc cr blocks received' and b1.inst_id = b2.inst_id
```

gc_traffic_print.sql

- You can use my script to print global cache performance data for the past minute. Download from scripts archive:

http://www.orainternals.com/scripts_rac1.php

Inst	CR blocks Rx	CR time	CUR blocks Rx	CUR time	CR blocks Tx	CUR blocks Tx	Tot blocks
1	40999	13.82	7827	4.82	25070	17855	91751
2	12471	5.85	8389	5.28	31269	9772	61901
3	28795	4.11	18065	3.97	28946	4248	80054
4	33105	4.54	12136	4.68	29517	13645	88403

- During the same time frame, output of the script from prior slide:

INST_ID	RECEIVED	RECEIVE TIME	AVG RECEIVE TIME (ms)
4	165602481	104243160	6.2947825
2	123971820	82993393	6.69453695
3	215681074	103170166	4.7834594
1	134814176	66663093	4.9448133

Very misleading!

Review all nodes.

- It is important to review performance data from all the nodes.
- It is easy to create AWR reports from all nodes using my script:
Refer `awrrpt_all_gen.sql`.
 - [Don't forget that access to AWR report needs license]
- Or use my script `gc_traffic_processing.sql` from my script archive.

Default collection period is 60 seconds.... Please wait for at least 60 seconds...

Inst	CR blk Tx	CR bld	CR fls tm	CR snd tm	CUR blk TX	CUR pin tm	CUR fls tm	CUR blk TX
2	67061	.08	.88	.23	34909	1.62	.2	.23
3	38207	.17	2.19	.26	28303	.61	.08	.26
4	72820	.06	1.76	.2	40578	1.76	.24	.19
5	84355	.09	2.42	.23	30717	2.69	.44	.25

Place holder events

- Few events are place holder events such as:
 - gc cr request
 - gc cr multiblock request
 - gc current request
- ...
- Sessions can be seen waiting for these wait events, but will not show up in AWR / ADDM reports.
- After sending the global cache block request, foreground process waits on these events.
- On receipt of the response, time is accounted for correct wait event.

2-way/3-way events

- As we saw in the prior section, there are 2-way and 3-way block transfer.
 - GC CR block 2-way
 - GC CR block 3-way
 - GC CUR block 2-way
 - GC CUR block 3-way
- Even if there are many instances, only three instances participate in a block transfer.
- But, flush messages can be sent to all instances in few cases.

Gc grants

- Wait events 'gc cr grant 2-way' and 'gc current grant 2-way' indicates
 - Block is not in any cache
 - Permission granted to read from the disk.

```
WAIT #6: nam='gc cr grant 2-way' ela= 567 p1=295 p2=770871 p3=1  
obj#=5153800 tim=817052932927
```

```
WAIT #6: nam='db file sequential read' ela= 11003 file#=295 block#=770871  
blocks=1 obj#=5153800 tim=817052943998
```

Congested..

- Congestion indicates that LMS processes were not able to service fast enough:
 - gc cr grant congested, gc current grant congested
 - gc cr block congested, gc current block congested
- Focus on LMS processes and usual culprits are load, SQL performance or longer CPU queue etc.

Histogram

89.4% of these waits are Under 4ms.

- Averages can be misleading. Use `v$event_histogram` to understand true performance metrics.
- It is better to take snapshots of this data and compare the differences.

INST_ID	EVENT	WAIT_TIME_MILLI	WAIT_COUNT	THIS_PER	TOTAL_PER
1	gc cr block 2-way	1	466345	.92	.92
1	gc cr block 2-way	2	23863264	47.58	48.51
1	gc cr block 2-way	4	20543430	40.96	89.47
1	gc cr block 2-way	8	4921880	9.81	99.29
1	gc cr block 2-way	16	329769	.65	99.95
1	gc cr block 2-way	32	17267	.03	99.98
1	gc cr block 2-way	64	2876	0	99.99
1	gc cr block 2-way	128	1914	0	99.99
1	gc cr block 2-way	256	1483	0	99.99
1	gc cr block 2-way	512	618	0	99.99
1	gc cr block 2-way	1024	83	0	99.99
1	gc cr block 2-way	2048	4	0	99.99
1	gc cr block 2-way	4096	3	0	99.99
1	gc cr block 2-way	8192	5	0	99.99
1	gc cr block 2-way	16384	3	0	100

GC event histograms

- Better yet, use my script `gc_event_histogram.sql` to understand current performance metrics.

```
Default collection period is sleep seconds. Please wait..  
Enter value for event: gc cr block 2-way  
Enter value for sleep: 60
```

Inst id	Event	wait time milli	wait cnt
1	gc cr block 2-way	1	37
1	gc cr block 2-way	2	4277
1	gc cr block 2-way	4	5074
1	gc cr block 2-way	8	1410
1	gc cr block 2-way	16	89
1	gc cr block 2-way	32	1
1	gc cr block 2-way	64	0
1	gc cr block 2-way	128	0
1	gc cr block 2-way	256	0

Gc buffer busy waits

- GC buffer busy waits are usually symptoms. In many instances, this event can show up the top most waited event.
- GC Buffer busy simply means that buffer is pinned by another process and waiting for a different global cache event.
- Understand why that 'buffer pin holder' is waiting. Resolving that will resolve global cache buffer busy waits.
- Segment header changes dues to insufficient freelist groups also can lead to longer 'gc buffer busy' waits.

Example analysis

Client had high Global Cache response time waits.

Global Cache and Enqueue Services - Workload Characteristics

```
~~~~~
```

Avg global enqueue get time (ms):	2.5
Avg global cache cr block receive time (ms):	18.2
Avg global cache current block receive time (ms):	14.6
Avg global cache cr block build time (ms):	0.3
Avg global cache cr block send time (ms):	0.2
Global cache log flushes for cr blocks served %:	25.1
Avg global cache cr block flush time (ms):	5.2
Avg global cache current block pin time (ms):	0.4
Avg global cache current block send time (ms):	0.2
Global cache log flushes for current blocks served %:	1.7
Avg global cache current block flush time (ms):	5.2

GC CR latency

- GC CR latency \approx

Time spent in sending message to LMS +
LMS processing (building blocks etc) +
LGWR latency (if any) +
LMS send time +
Wire latency

Statistics : gc cr block flush time
gc cr block build time
gc cr block send time

CR latency

- Three instances are suffering from CR latency, except instance 2!

Wait time	Node 1	Node 2	Node 3	Node 4
Avg. CR block receive time	18.2	6.7	20.0	17.3
Avg CUR block receive time	14.6	5.0	11.6	17.3

- In RAC, node suffering from chronic issues causes GC performance issues in other nodes. With that logic in mind, node 2 should be suffering from chronic issues.

Breakdown of latency

- Sum of flush time is higher, but it is comparable across the cluster.

But, notice the build time in node 2.

Statistics	Node 1	Node 2	Node 3	Node 4	Total
gc cr block build time	11,392	148,666	5,267	6,632	171,957
Gc cr block flush time	56,634	75,751	34,406	53,031	219,822
Gc cr block send time	9,153	7,779	4,018	7,905	28,855

Consistent reads

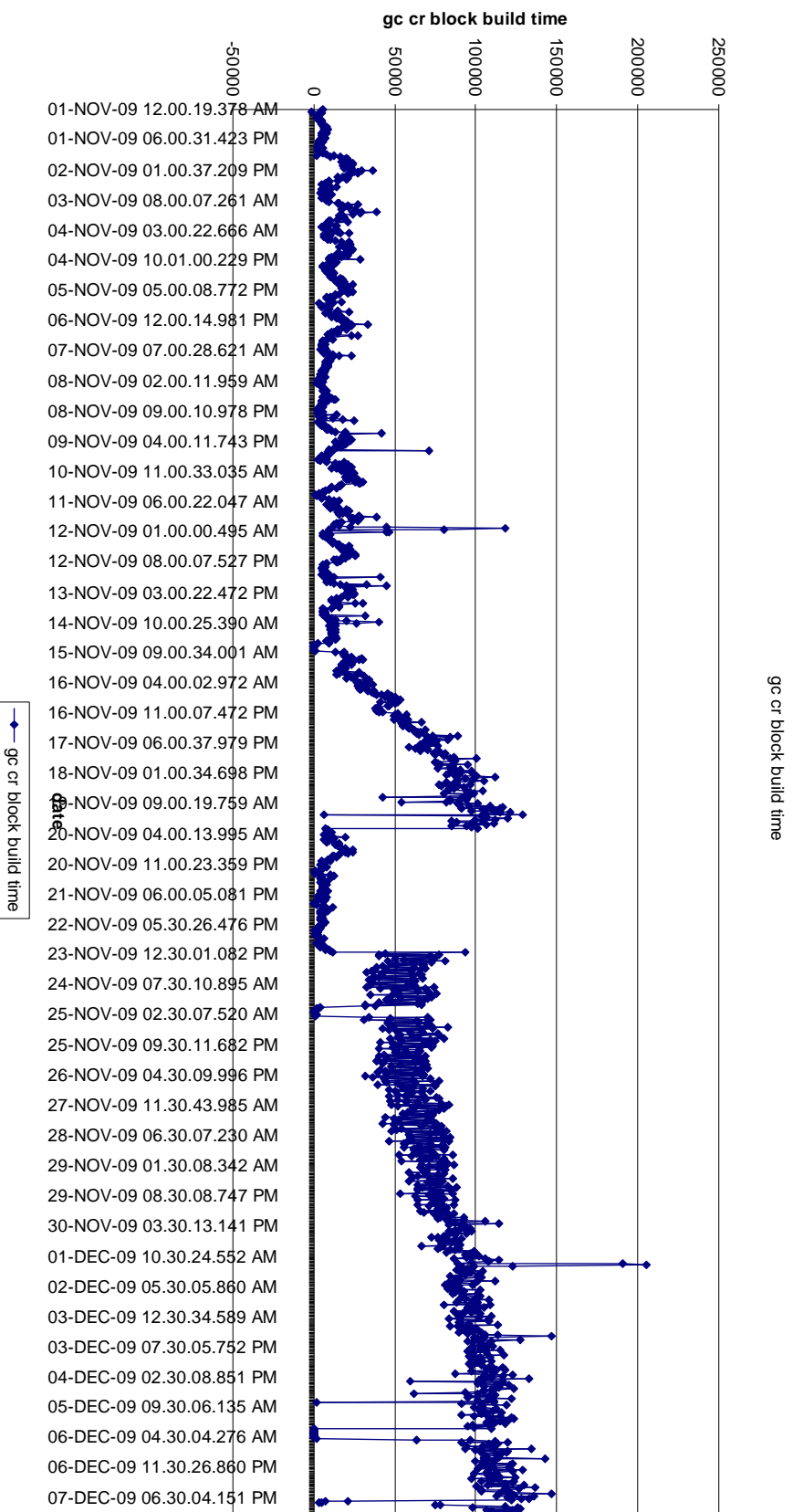
- For CR blocks, time is spent in building blocks, which indicates consistent block generation.

Very high value compared to other nodes.

Statistics	Node 1	Node 2	Node 3	Node 4
data blocks consistent Reads – undo records applied	2,493,242	86,988,512	3,090,308	7,208,575
db block changes	6,276,149	43,898,418	20,698,189	14,259,340

Time line

- We wanted to see when this problem started. Surprisingly, instance 2 had a pattern of increasing flush time.



Db block changes

Unfortunately, AWR report does not capture segments with high 'db block changes'.

```
with segstats as (  
  select * from (  
    select inst_id, owner, object_name, object_type , value ,  
          rank() over (partition by inst_id, statistic_name order by value desc  
    ) rnk , statistic_name  
    from gv$segment_statistics  
    where value >0  
  ) where rnk <11  
),  
sumstats as ( select inst_id, statistic_name, sum(value) sum_value from  
gv$segment_statistics group by statistic_name, inst_id)  
select a.inst_id, a.statistic_name, a.owner, a.object_name,  
a.object_type,a.value,(a.value/b.sum_value)*100 perc  
from segstats a , sumstats b  
where a.statistic_name = b.statistic_name  
and a.inst_id=b.inst_id  
and a.statistic_name ='db block changes'  
order by a.statistic_name, a.value desc  
/
```

INST_ID	STATISTIC_NAME	OWNER	OBJECT_NAME	TYPE	VALUE	PERC
2	db block changes	AR	CUSTOM_TABLE	TABLE	122949282400	81.39
4		INV	MTL_MATERIAL_TRANS_TEMP_N1	INDEX	1348827648	16.59
3		AR	RA_INTERFACE_LINES_N2	INDEX	791733296	9.77
3		AR	RA_CUSTOMER_TRX_LINES_N2	INDEX	715855840	8.83
1		INV	MTL_MATERIAL_TRANS_TEMP_N1	INDEX	652495808	12.44

...

Solution

- Finally, it boiled down to a custom code bug which was updating almost all rows in a table unnecessarily.
- Unfortunately, number of rows that fall in to that criteria was slowly increasing.
- So, GC CR response time was slowly creeping up and it wasn't easy to identify the root cause.
- After the code fix, GC CR time came down to normal range.

Agenda

- Global cache performance
- **Undo, redo and more**
- RAC background process tuning
- Interconnect issues, lost packets and network layer
- Network layer tuning
- Effective use of parallel query
- Troubleshooting locking issues
- Object re-mastering

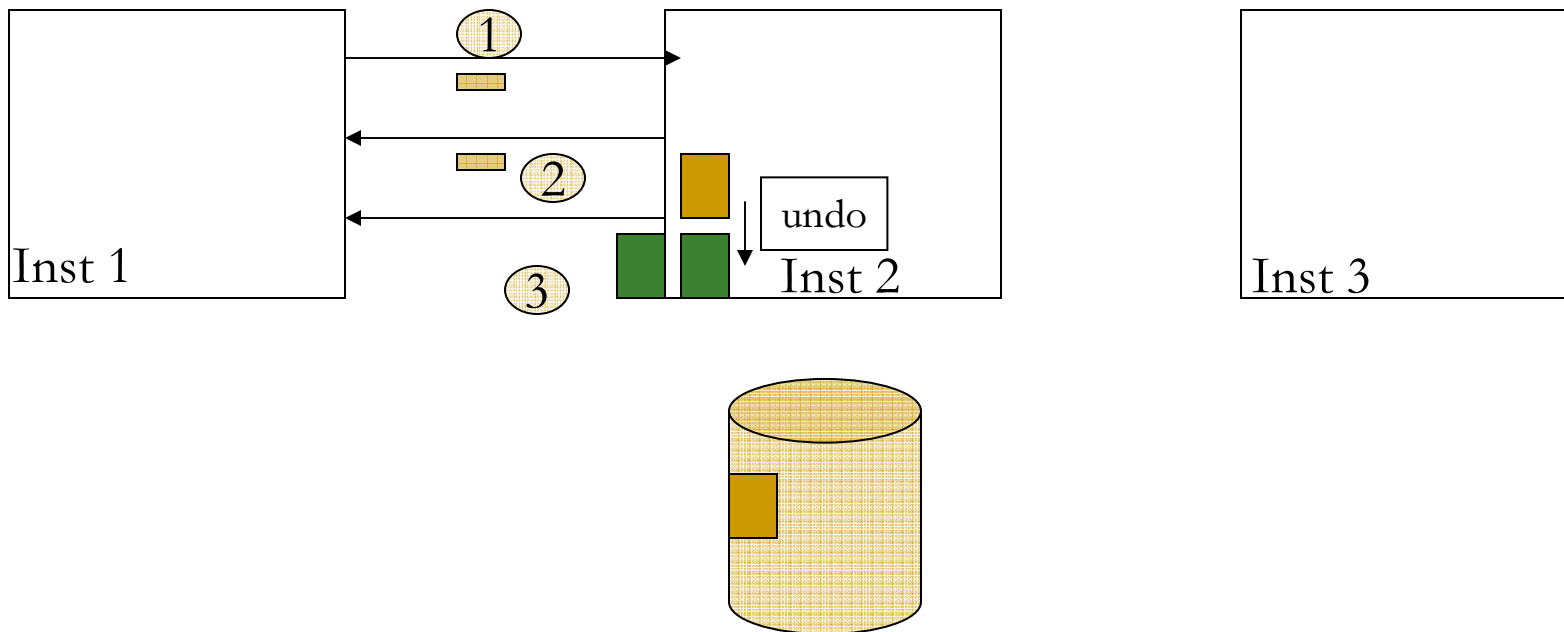
Question

- Does an instance access undo blocks allocated to another instance?

CR and undo

```
Select c1 from t1 where n1=:b1;
```

- ① User process in instance 1 requests master for the block in PR mode.
- ② Current owner (2) holds the block in Exclusive mode and so master grants a PR to instance 1. But, the block is ahead of time.
- ③ Instance 2 applies undo to create a version of the block consistent with SCN requested. Then ships the block to instance 1.



Light-works rule

- But, if one instance is a read only instance, then it might request another instance to generate CR copies applying undo blocks.
- This is avoided by light-works rule. If an instance is excessively serving a block by applying undo blocks, it will downgrade the block mode.
- Requesting instance then will apply undo blocks if needed reducing load on the prior owner.

Undo for CR

- A query can not read the block image with an SCN later than the query environment SCN.
- If a block is ahead of time, then undo blocks are applied to create a consistent version of the block.
- But, if the block was modified by a different instance, then undo blocks may need to be shipped from another instance.
- This can cause excessive cache transfers or excessive physical reads for undo blocks.
- Node affinity will be helpful to resolve this.

Commit cleanout

- ITL entries in the blocks may not be cleaned out immediately.
- Session reading the block next time will check if the pending transaction is committed or not.
- If the transaction table is cycled through, then the session will apply undo to find a transaction table version with that transaction.
- This can lead to vicious cycle of rolling back the transaction table accessing undo blocks excessively.
- In RAC, this problem is magnified, since the transaction(s) could be in a different instance. So, transaction table blocks and undo blocks from different instance need to be shipped from other node or read from the disk.

Redo and LGWR

- LGWR performance is important for global cache response time.
- Even for CR blocks LGWR must flush if the block is considered busy.

Statistics	Node 1	Node 2	Node 3	Node 4
Gc cr block flush time	129,970	12,289	11,556	27143

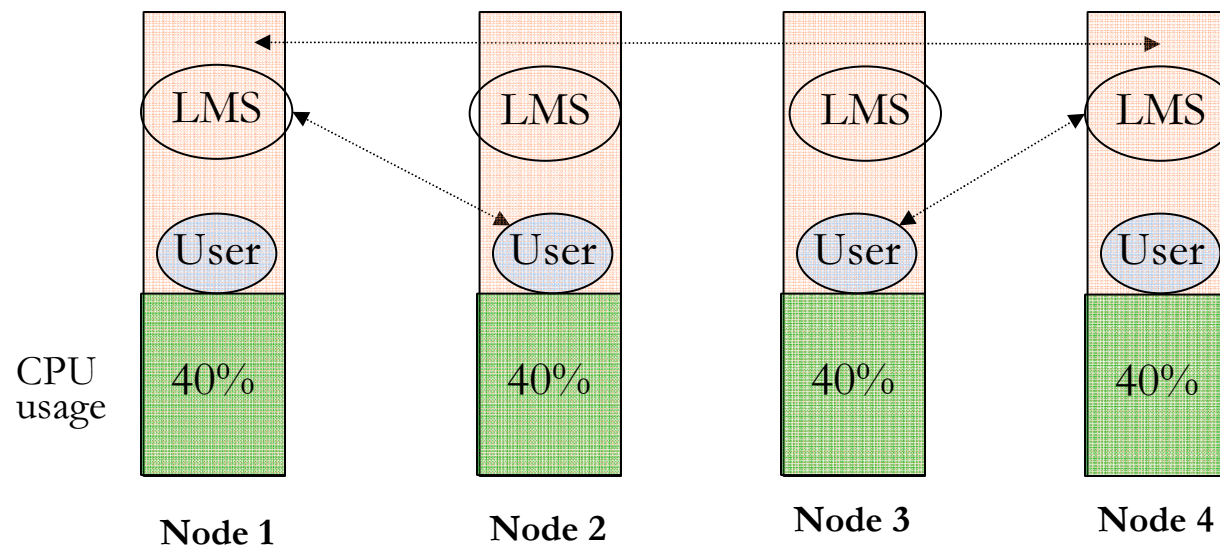
Statistics	Node 1	Node 2	Node 3	Node 4
Avg. gc cr block rx time	4.2	22.7	21.5	11.0

Agenda

- Global cache performance
- Undo, redo and more
- **RAC background process tuning**
- Interconnect issues, lost packets and network layer
- Network layer tuning
- Effective use of parallel query
- Troubleshooting locking issues
- Object re-mastering

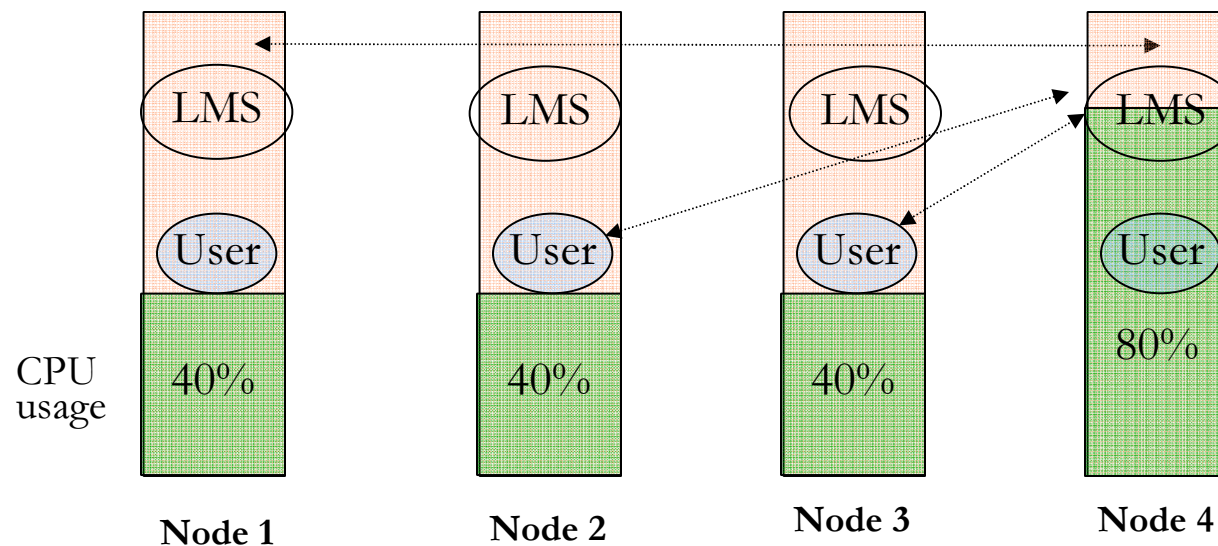
LMS processes – normal state

- During normal conditions, LMS processes are operating with no CPU latency.
- So, there is no Global cache latency either.



CPU latency

- If one node is suffering from CPU starvation then LMS process running in that node will suffer from CPU latency.
- This will result in Global cache latency in other nodes.



Global Cache waits

- Global Cache waits increases due to increase in LMS latency in the CPU starved node.
- Much of these GC waits are blamed on interconnect interface and hardware.
- In many cases, interconnect is performing fine, it is that GCS server processes are introducing latencies.

LMS & 10.2.0.3

- In 9i, increasing priority of LMS processes to RT helps (more covered later).
- From Oracle release 10.2.0.3 LMS processes run in Real Time priority by default.
- Two parameters control this behaviour:
 - `_high_priority_processes`
 - `_os_sched_high_priority`

Parameters in 10gR2

- `_high_priority_processes`:

Default value: LMS* | VKTM*

This parameter controls what background processes should get Real time priority. Default is all LMS processes and VKTM process.

- `_os_sched_high_priority` :

Default value: 1

This is a switch. If set to 0, no background process will run in high priority.

oradism

- Of course, bumping priority needs higher privileges such as root in UNIX.
- Oradism utility is used to increase the priority class of these critical background process in UNIX.
- Verify that LMS processes are using Real time priority in UNIX and if not, oradism might not have been configured properly.
- In Windows, oradism service is used to increase the priority.

More LMS processes?

- Typical response is to increase number of LMS processes adjusting `_lm_lms` (9i) or `gcs_server_processes`(10g).
- Increase in LMS processes without enough need increases `xcalls/migrates/tlb-misses` in massive servers.
- Further, LMS process runs in RT CPU priority and so, CPU usage will increase.

LMS & CPU usage

- In huge servers, by default, number of LMS processes might be quite high. It is possible to get up to 26 LMS processes by default.
- Typically, same number of LMS processes as interconnect or remote nodes is a good starting point.
- If there is enormous amount of interconnect traffic, then configure LMS processes to be twice the interconnect.

LGWR and CPU priority

- LGWR performance is akin to Global cache performance.
- If LGWR suffers from performance issues, it will reflect on Global cache performance.
- For example, If LGWR suffers from CPU latency issues, then LMS will have longer waits for 'gcs log flush sync' event
- This leads to poor GC performance in other nodes.

LGWR priority

- Method to increase priority for LGWR and LMS in 9i (Example for Solaris)]. If you don't want to increase priority to RT for LGWR, at least, consider FX priority.

```
priocntl -e -c class -m userlimit -p priority
```

```
priocntl -e -c RT -p 59 `pgrep -f ora_lgwr_${ORACLE_SID}`
```

```
priocntl -e -c FX -m 60 -p 60 `pgrep -f ora_lms[0-9]*_${ORACLE_SID}`
```

- In 10g, parameter `_high_priority_processes` can be used (needs database restart though)

```
alter system set "_high_priority_processes"="LMS*|LGWR*" scope=spfile sid='*';
```

```
alter system set "_high_priority_processes"="LMS*|VKTM*|LGWR*" scope=spfile  
sid='*'; (11g)
```

Pitfalls of RT mode

- Of course, there are few!
- LMS process can continuously consume CPU and can introduce CPU starvation in servers with few CPUs.
- A bug was opened to make LMS process sleep intermittently, but that causes LMS to be less active and can cause GC latency.
- Another undocumented parameter `_high_priority_process_num_yields_before_sleep` was introduced as a tunable. But, hardly a need to alter this parameter.
- Hyper-active LGWR can lead to latch contention issues.

Binding..

- Another option is to bind LGWR/LMS to specific processors or processor sets.
- Still, interrupts can pre-empt LMS processors and LGWR. So, binding LMS to processor set without interrupts helps (see psradm in solaris).
- But, of course, processor binding is useful in servers with higher number of CPUs such as E25K platforms.

CSSD/CRSD

- CSSD is a critical process. Few CSSD processes must be running with RT priority.

`crsctl set css priority 4`

- CPU starvation in the server can lead to missed network or disk heart beat. This can lead to node reboots.
- It is important to have good and consistent I/O performance to `ORA_CRSD_HOME` directories.
- If CSSD can't access those directories efficiently (i.e. due to NFS or other file system issues), then that can lead to node reboots too.

Summary

- In summary,
 - Use optimal # of LMS processes
 - Use RT or FX high priority for LMS and LGWR processes.
 - Configure decent hardware for online redo log files.
 - Tune LGWR writes and Of course, avoid double buffering and double copy using optimal file systems.
 - Of course, tune SQL statement to reduce logical reads and reduce redo size.

Agenda

- Global cache performance
- Few important RAC wait events and statistics
- RAC background process tuning
- Interconnect issues, lost packets and network layer
- Effective use of parallel query
- Troubleshooting locking issues
- Object re-mastering

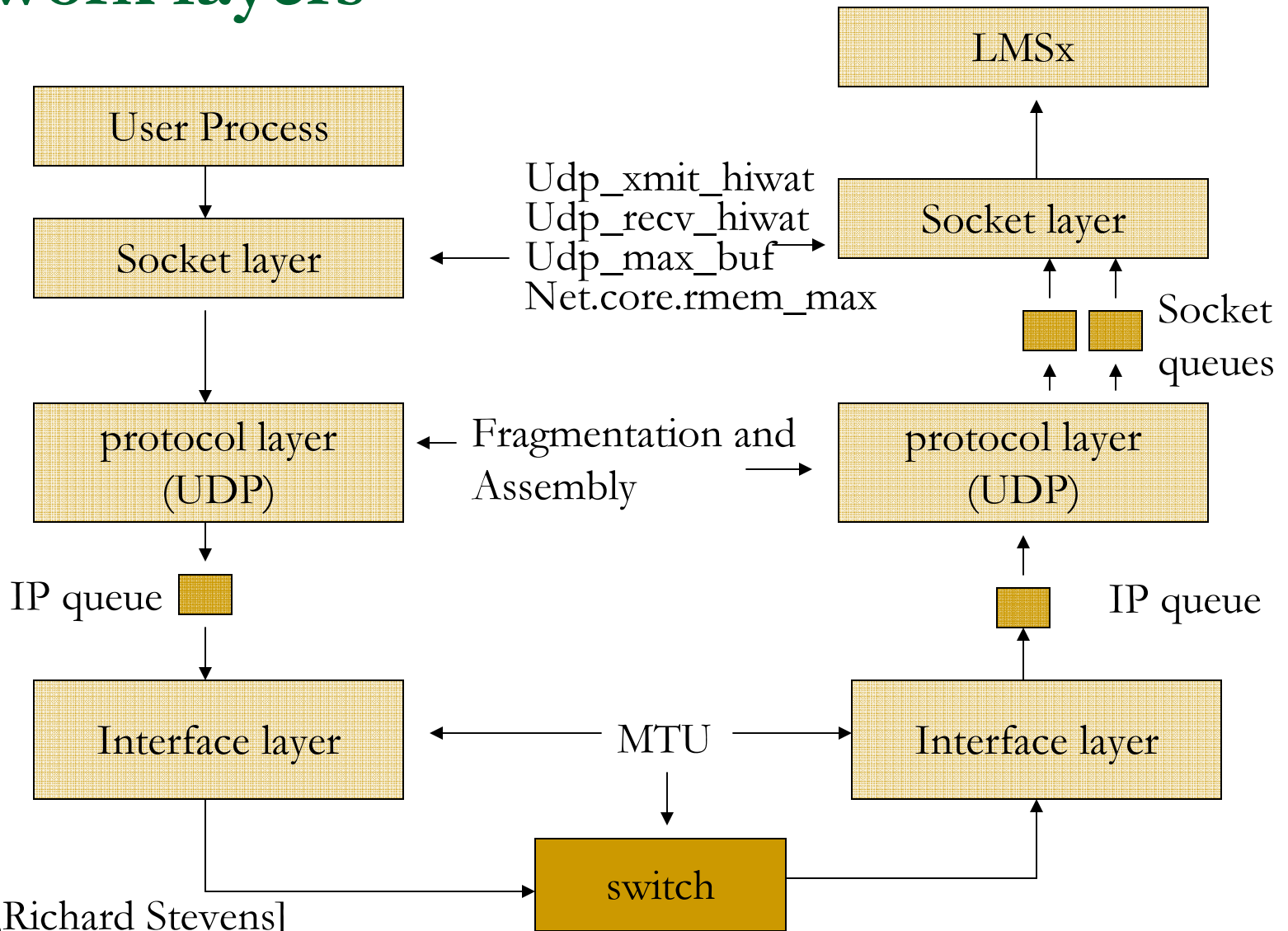
gc blocks lost

- Probably, the most critical statistics for interconnect issues.
- Consistent high amount of 'gc blocks lost' is an indication of problem with underlying network infrastructure. (Hardware, firmware etc).
- Need to understand which specific component is an issue. Usually, this is an inter-disciplinary analysis.
- Ideal value is near zero. But, only worry about this, if there are consistently higher values.

Effects of lost blocks

- Higher number of block loss can lead to timeouts in GC traffic wait events. Many processes will be waiting for place-holder events.
- Use total_timeouts column in v\$system_event to see if the timeouts are increasing.
- Percent of total_timeouts should be very small.

Network layers



Source: [8, Richard Stevens]

UDP buffer space

- UDP is a “send-and-forget” type protocol. Sending process does not get any acknowledgement.
- UDP Tx/Rx buffers are allocated per process.
- When the process executes CPU, it drains the UDP buffers. If the buffer is full, then incoming packets to that process are dropped.
- Default values for the UDP buffers are small for the bursty nature of interconnect traffic. Increase UDP buffer space to 128KB or 256KB.

CPU latency and UDP

- Due to CPU latency, process might not be able to acquire CPU quick enough.
- This can lead to buffer full conditions and lost packets.
- It is essential to keep CPU usage under 80% to avoid latencies and lost packets.

Agenda

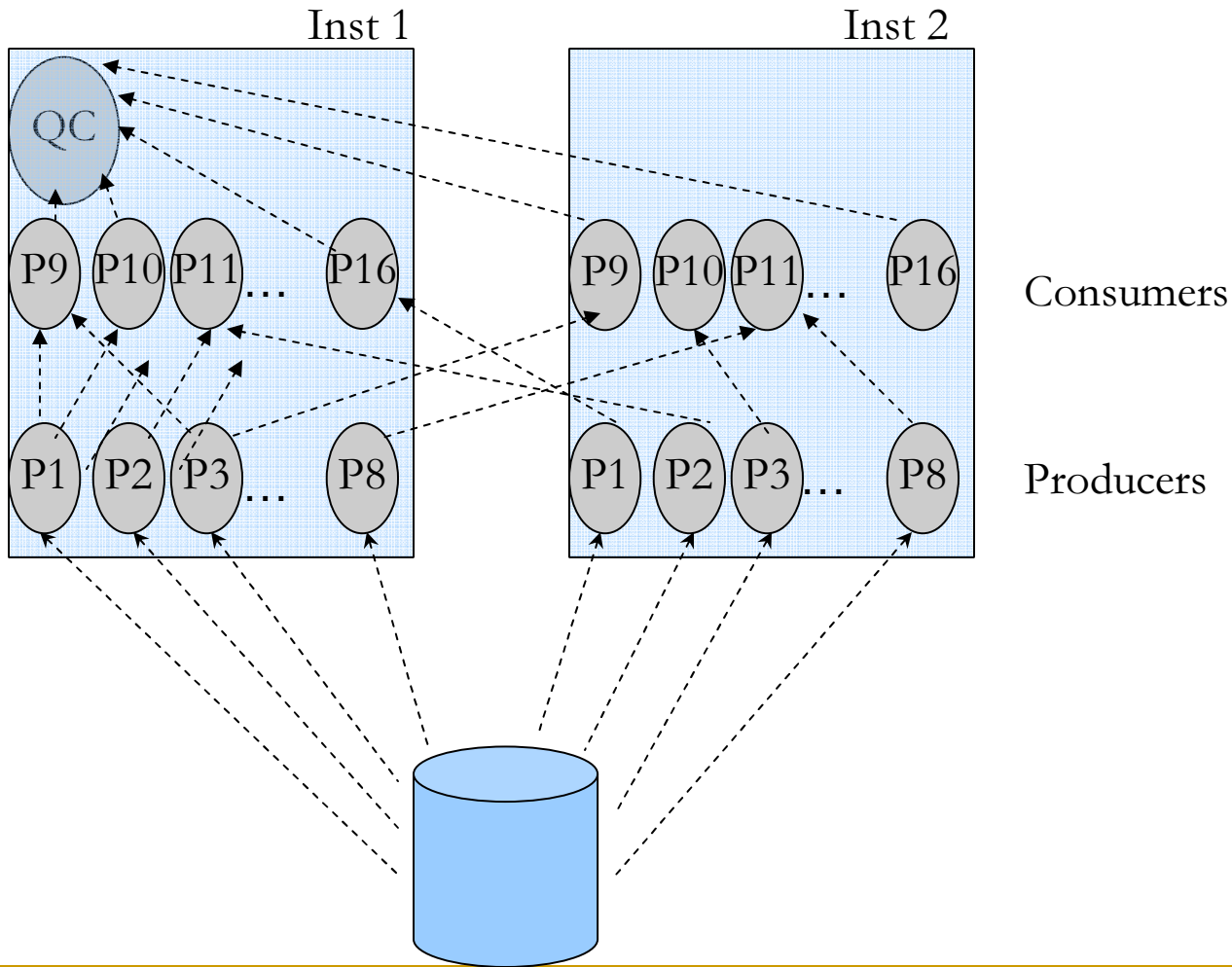
- Global cache performance
- Few important RAC wait events and statistics
- RAC background process tuning
- Interconnect issues, lost packets and network layer
- **Effective use of parallel query**
- Troubleshooting locking issues
- Object re-mastering

Parallel Query Setup

- Parallel Query slaves can be allocated from multiple instances for a query.
- It is imperative that PQ messages are transmitted between producers and consumers.
- Insufficient network bandwidth with PQ storm can cause higher GC latency and possible packet loss.

PQ Optimization

Communication between producers/consumers are Not limited to one node. Gigabytes of data flow Between node 1 and node 2.



Optimizations in 10g/11g

- PQ algorithms are optimized in Oracle versions 10g and 11g. Only few discussed here.
- In 11g, interconnect traffic due to PQ is also reported in the AWR reports.
- Oracle code tries to allocate all PQ slaves in one node, if possible. This minimizes PQ induced interconnect traffic.
- If it not possible to allocate all slaves from a node, then the least loaded node(s) are chosen for PQ slave allocation.

PQ and partitioning

- For partition-wise joins, PQ traffic is further minimized.
- Partitions are joined within the same node further reducing PQ induced interconnect traffic.
- Of course, partitioning strategy plays critical role in this localized partition-wise joins.
- Full partition-wise join can use all instance effectively, akin to shared-nothing systems. Full partition-wise joins does not induce spurious interconnect traffic.

PQ-Summary

- Inter instance parallelism need to be carefully considered and measured.
- For partition based processing, when processing for a set of partitions is contained within a node, performance will be better.
- Excessive inter instance parallelism will increase interconnect traffic leading to performance issues.
- http://www.oracle.com/technology/products/bi/db/11g/pdf/twp_bidw_parallel_execution_11gr1.pdf

“..inter-node parallel execution will not scale with an undersized interconnect”

Agenda

- Global cache performance
- Few important RAC wait events and statistics
- RAC background process tuning
- Interconnect issues, lost packets and network layer
- Effective use of parallel query
- **Troubleshooting locking issues**
- Object re-mastering

GES layer

- In RAC, Global Enqueue Services handles all global locks requests, converts.
- Almost many resources are globally co-ordinated.
- For example, in a single instance, library cache locks are not implemented as “enqueues”. But, In RAC, library cache locks are externalized as enqueues (LA-LZ) and globally co-ordinated.
- This means that DDL statement on an object in one node need to be globally co-ordinated so that parse locks can be invalidated in all instances.

Agenda

- Global cache performance
- Few important RAC wait events and statistics
- RAC background process tuning
- Interconnect issues, lost packets and network layer
- Effective use of parallel query
- Troubleshooting locking issues
- **Object re-mastering**

Object re-mastering

- Before reading the block, an user process must request master node of the block to access that block.
- Typically, a batch process will access few objects aggressively.
- If an object is accessed excessively from a node then re-mastering the object to that node reduces Global cache grants.
- Local grants (affinity locks) are very efficient compared to remote grants avoiding global cache messaging traffic.

Object based in 10gR2

- Dynamic remastering is file based in 10gR1. If a block need to be remastered, then every block in that data file must be remastered to an instance.
- In 10gR2, remastering is object based. If a block to be remastered, then all blocks associated with that object is remastered to an instance.
- Three background processes work together to implement dynamic remastering functionality.

High level overview 10gR2

- LCK0 process maintains object level statistics and determines if remastering must be triggered.
- If an object is chosen, a request is queued. LMD0 reads the request queue and initiates GES freeze. LMD0 trace file

```
*** 2010-01-08 19:41:26.726
```

```
* kjdrchkdrm: found an RM request in the request queue  
  Dissolve pkey 6984390
```

```
*** 2010-01-08 19:41:26.727
```

```
Begin DRM(189) - dissolve pkey 6984390 from 2 oscan 1.1  
  ftd received from node 1 (8/0.30.0)  
  ftd received from node 0 (8/0.30.0)  
  ftd received from node 3 (8/0.30.0)  
  all ftds received
```

- LMON performs reconfiguration.

```
*** 2010-01-08 19:41:26.793
```

```
Begin DRM(189)  
  sent syncr inc 8 lvl 5577 to 0 (8,0/31/0)  
  synca inc 8 lvl 5577 rcvd (8.0)
```

Parameters 10gR2

Three parameters control the behavior:

- `_gc_affinity_limit`
- `_gc_affinity_time`
- `_gc_affinity_minimum`
- `_gc_affinity_limit` default value is 50. Not documented well, but, it is number of times a node should access an object more than other nodes.
- `_gc_affinity_time` default value is 10. Frequency in seconds to check if remastering to be triggered or not.
- `_gc_affinity_minimum` determines number of DRM requests to enqueue and default is 600.

Defaults

- Default for these parameters may be too low in a very busy, high-end instances.
- If your database have higher waits for ‘gc remaster’ and ‘gcs drm server freeze’ then don’t disable this feature completely. Instead tune it.
- Some good starting points (for a very busy environment) are:
 - [YMMV]
 - `_gc_affinity_limit` to 250
 - `_gc_affinity_minimum` to 2500.

11g

- In 11g, these three parameters are completely removed.
- Three new parameters are introduced:
 - `_gc_affinity_locking`
 - `_gc_affinity_locks`
 - `_gc_affinity_ratio`
- Sorry, I have not tested these parameters thoroughly yet.

An example

Top 5 Timed Events					
Event	waits	Time (s)	Avg wait (ms)	%Total Call Time	wait Class
gc buffer busy	1,826,073	152,415	83	62.0	Cluster
CPU time		30,192		12.3	
enq: TX - index contention	34,332	15,535	453	6.3	Concurrenc
gcs drm freeze in enter server	22,789	11,279	495	4.6	Other
enq: TX - row lock contention	46,926	4,493	96	1.8	Applicatio

Global Cache and Enqueue Services - workload Characteristics

Avg global enqueue get time (ms):	16.8
Avg global cache cr block receive time (ms):	17.1
Avg global cache current block receive time (ms):	14.9

Views

- View `v$gcspfmaster_info` provides remastering details. For example, you can identify the object with high remastering count.

FILE_ID	OBJECT_ID	CURRENT_MASTER	PREVIOUS_MASTER	REMASTER_CNT
0	6983606	0	32767	1
0	5384799	2	1	2
0	6561032	3	2	2
0	5734002	0	2	2
0	6944892	2	0	2
0	5734007	2	0	4
0	6944891	2	0	5
0	6795604	2	0	5
0	6944894	2	0	5
0	6795648	2	0	6
0	5734006	2	0	6
0	4023250	2	0	6
0	5734003	0	2	7

Views

- View `x$object_object_affinity_statistics` provides current object affinity statistics.

```
select * from x$object_affinity_statistics order by opens
```

ADDR	INDX	INST_ID	OBJECT	NODE	OPENS
...					
FFFFFFFF7C04CB40	8	3	4740170	1	113
FFFFFFFF7C04CB40	109	3	1297745	1	127
FFFFFFFF7C04CB40	21	3	1341531	1	128
FFFFFFFF7C04CB40	2	3	2177393	1	135
FFFFFFFF7C04CB40	153	3	6942171	2	174
FFFFFFFF7C04CB40	108	3	1297724	1	237
FFFFFFFF7C04CB40	3	3	2177593	1	239
FFFFFFFF7C04CB40	106	3	1297685	1	337
FFFFFFFF7C04CB40	53	3	6984154	3	1162

Oradebug

- You can manually remaster an object with oradebug command

```
oradebug lkdebug -m pkey <object_id>
```

- This enqueues an object remaster request. LMD0 and LMON completes this request

```
*** 2010-01-08 23:25:54.948
```

```
* received DRM start msg from 1 (cnt 1, last 1, rmno 191)
```

```
Rcvd DRM(191) Transfer pkey 6984154 from 0 to 1 oscan 0.0
```

```
ftd received from node 1 (8/0.30.0)
```

```
ftd received from node 0 (8/0.30.0)
```

```
ftd received from node 3 (8/0.30.0)
```

Oradebug

- You can manually remaster an object with oradebug command. Current_master starts from 0.

```
1* select * from v$gcspfmaster_info where object_id=6984154
SQL> /
```

FILE_ID	OBJECT_ID	CURRENT_MASTER	PREVIOUS_MASTER	REMASTER_CNT
0	6984154	1	0	2

```
SQL> oradebug lkdebug -m pkey 6984154
Statement processed.
```

```
SQL> select * from v$gcspfmaster_info where object_id=6984154
2 /
```

FILE_ID	OBJECT_ID	CURRENT_MASTER	PREVIOUS_MASTER	REMASTER_CNT
0	6984154	2	1	3

References

1. Oracle support site. Metalink.oracle.com. Various documents
2. Internal's guru Steve Adam's website
www.ixora.com.au
3. Jonathan Lewis' website
www.jlcomp.daemon.co.uk
4. Julian Dyke's website
www.julian-dyke.com
5. 'Oracle8i Internal Services for Waits, Latches, Locks, and Memory'
by Steve Adams
6. Randolph Geist : <http://oracle-randolf.blogspot.com>
7. Tom Kyte's website
[Asktom.oracle.com](http://asktom.oracle.com)
8. Richard Stevens, Gary R Wright: TCP/IP Illustrated