

---

# RAC Performance myths

---

By  
Riyaj Shamsudeen



---

## Who am I?



- 16 years using Oracle products
- Over 15 years as Oracle DBA
- Certified DBA versions 7.0,7.3,8,8i &9i
- Specializes in performance tuning, Internals and E-business suite
- Principal at OraInternals – A performance consulting company.  
<http://www.orainternals.com>
- OakTable member
- Email: [rshamsud@orainternals.com](mailto:rshamsud@orainternals.com)
- Blog : <http://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.**

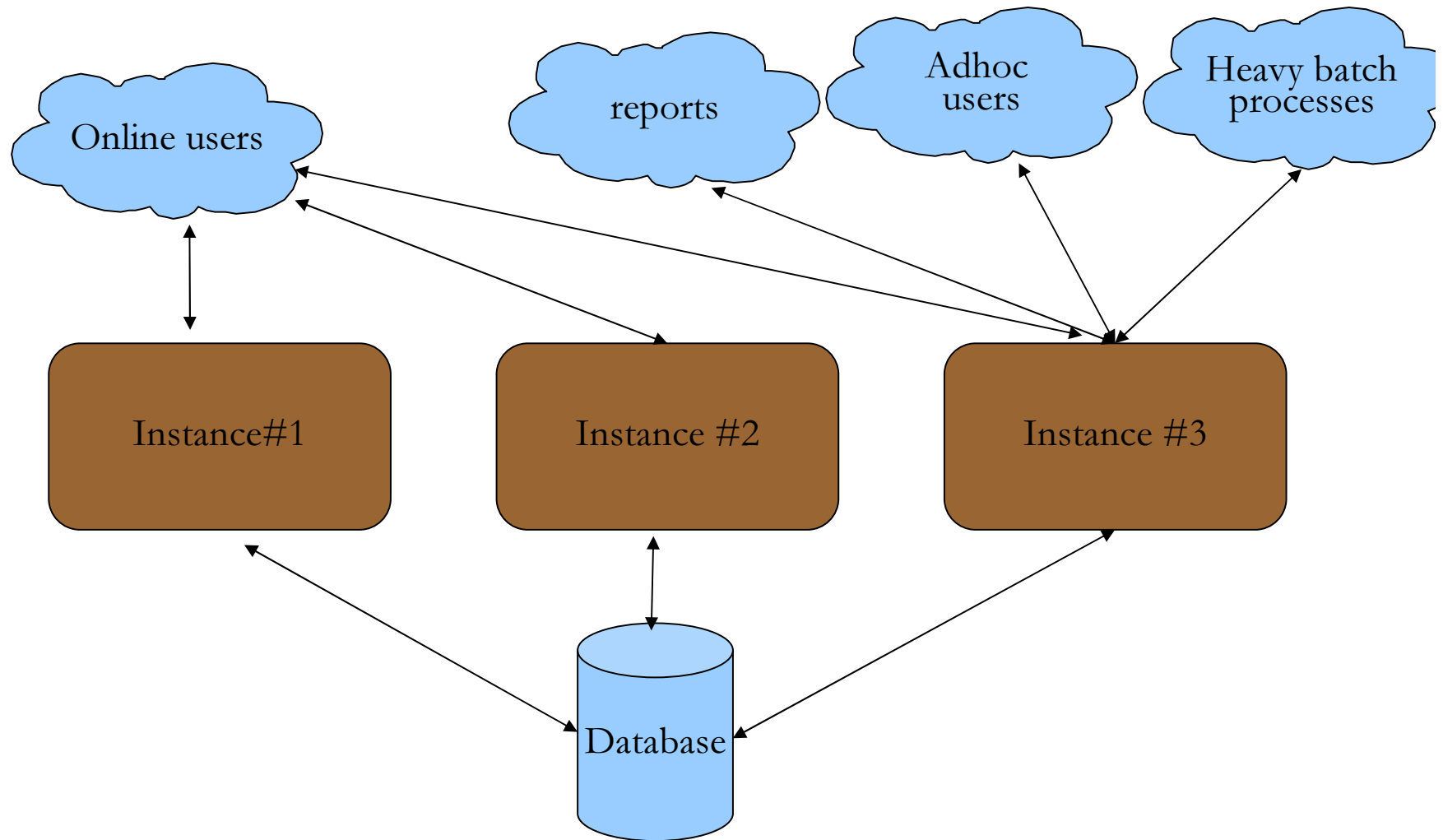
**If you corrupt your databases by running my scripts, you are solely responsible for that.**

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

# Agenda - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC.
- Bitmap index performance is worse compared to single instance.

# Typical RAC node setup

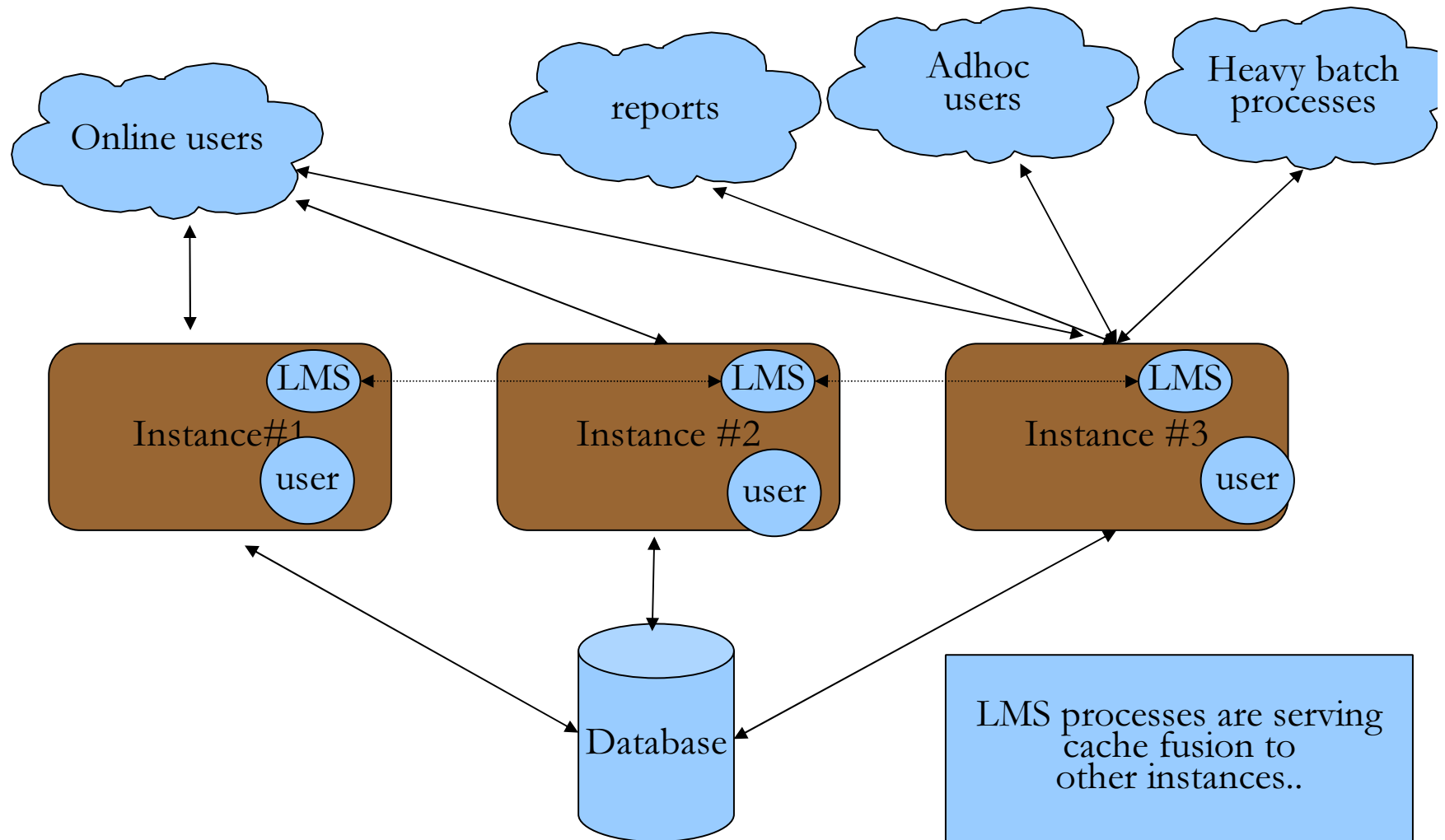


---

# Reporting node

- Idea here is to put online “money-paying” users to a all nodes and throw costly reports/adhoc SQL/batch in to one node.
  - Only a small part of online users are in batch node.
  - High CPU usage in the batch node shouldn't cause any issues to online users, right?
  - If SQL is bad, don't worry about tuning, let it run in report node. It wouldn't affect much online users performance, right?
  - If batch process is costly, no need to tune it, run that in batch node.
- 
- Not Exactly!

# What really happens?



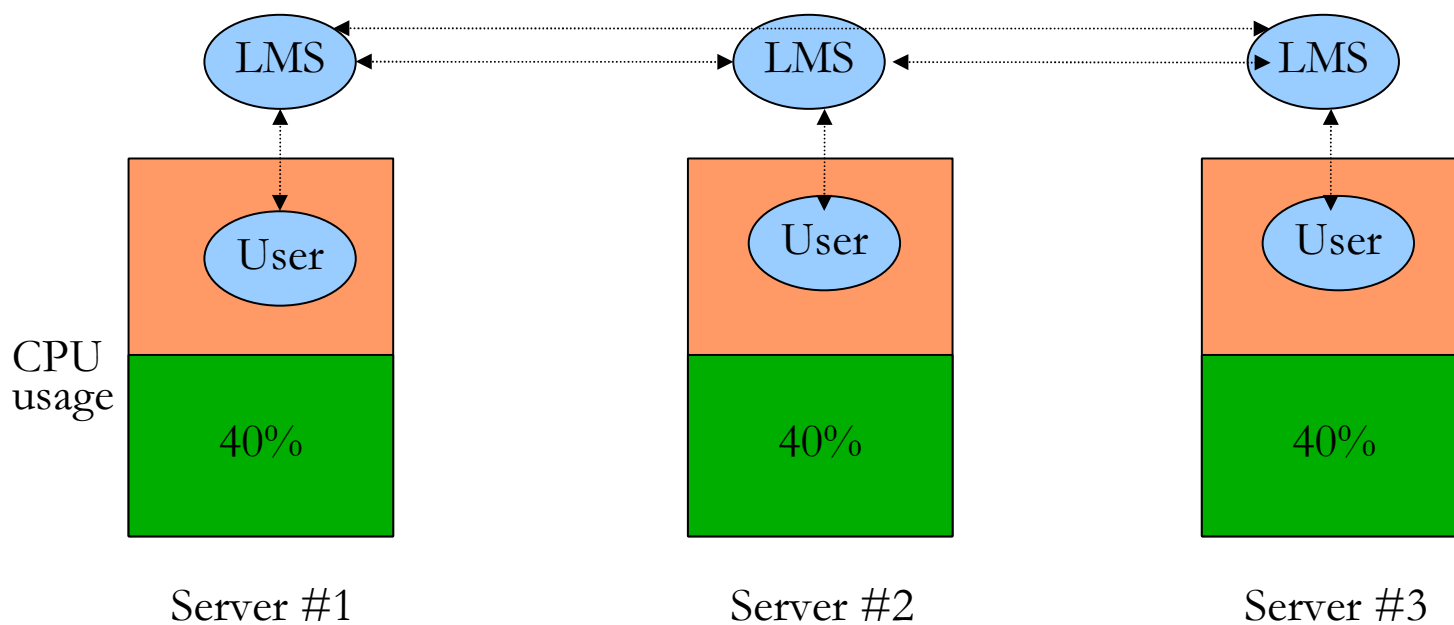
---

# Global cache transfer

- With cache fusion, blocks are transferred from remote cache if a suitable block is found in the remote cache avoiding costly disk reads.
- Block transfer between caches are done by LMS processes.
- Until 10.2.0.1, LMS processes are running in normal priority.
- If there is CPU starvation in any server, then all instances will be affected due to LMS latency.

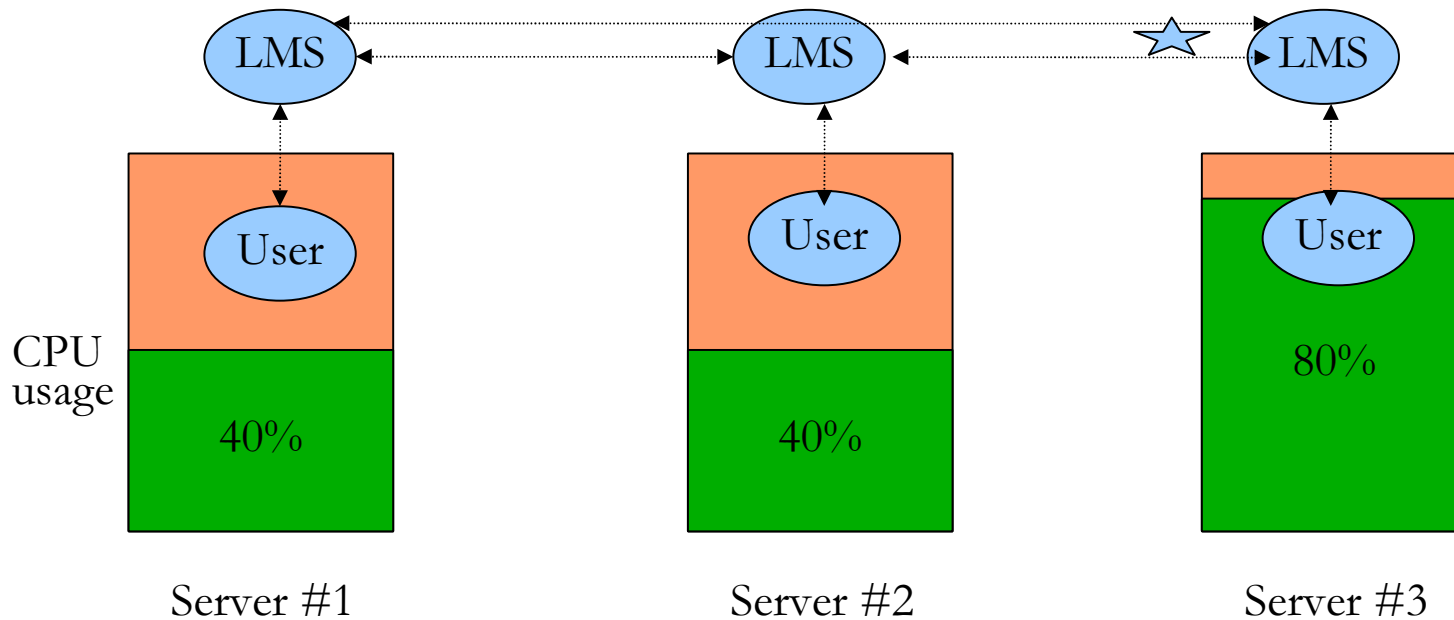
# LMS processes – normal state

In steady state, there is no message latency between LMS processes.



# LMS processes – one node is busy

But, if one node is busy, then LMS processes in that node starve for CPU and cause cache fusion latency.



# GC waits

- GC CR waits such as 'gc cr grant 2 way' (10g) and 'global cache cr request' (9i) latency increases due to global cache latencies.

Event	Waits	%Time -outs	Total Wait Time (s)	Avg wait (ms)	Waits /txn
gc cr grant 2-way	11,518	3.0	23	2	14.7

- 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.

---

## More LMS processes?

- Typical response from DBA to improve global cache performance is to increase # of LMS processes adjusting `_lm_lms` (9i) or `gcs_server_processes`(10g).
  - This has detrimental effect in performance.
  - More LMS processes increases latency due to TLB thrashing. From `mpstat/trapstat` outputs, it is visible that there is increased amount of `xcalls/migrates/tlb-misses`.
  - Few busy LMS processes are better than many quasi-busy LMS processes
-

---

## LMS & CPU usage

- Typically, same number of LMS processes as interconnect or remote nodes seems to be a good starting point.
- For e.g., in a four node cluster three LMS processes per node is a good starting point.
- Of course, Correct way to fix this issue is to reduce CPU usage by tuning SQL statements (or) add more CPUs if necessary.
- In real life, that is not always possible.

---

## 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. This is alleviating much of performance issues with LMS issues.
- Two parameters control this behaviour:
  - `_high_priority_processes` : “High Priority Process Name Mask” with a default value of LMS\*
  - `_os_sched_high_priority` : “OS high priority level” with a default value of 1.

# Agenda - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC.
- Bitmap index performance is worse compared to single instance.

# Node1 GC workload

Global Cache and Enqueue Services - workload Characteristics

Average of 63ms cr block receive time, quite high.

Avg global enqueue get time (ms):	8.9
Avg global cache cr block receive time (ms):	63.3
Avg global cache current block receive time (ms):	2.1
Avg global cache cr block build time (ms):	0.3
Avg global cache cr block send time (ms):	0.1
Global cache log flushes for cr blocks served %:	4.5
Avg global cache cr block flush time (ms):	51.5
Avg global cache current block pin time (ms):	0.0
Avg global cache current block send time (ms):	4.8
Global cache log flushes for current blocks served %:	0.1
Avg global cache current block flush time (ms):	30.0

---

## Could this be interconnect issue?

- Common reaction to any Global cache performance issue : It is an interconnect network problem.
- It could be, but not necessarily.
- Unless interconnect is flooded, interconnect latency is very small fraction of global cache latency.

---

# Interconnect performance

- Before LMS sends a block back to remote cache, LMS waits for Log flush to complete.
- Even CR block transfer suffer from this wait. Of course, CUR blocks needs to have log flush complete.

So,

Global cache latency  $\sim =$

Interconnect latency for message from & to LMS +  
LMS processing latency +  
LGWR log flush latency

# Node2 GC workload

In this specific case, log flush was very slow due to an hardware issue

## Global Cache and Enqueue Services - workload Characteristics

```
~~~~~
```

Avg global enqueue get time (ms):	0.3
Avg global cache cr block receive time (ms):	10.4
Avg global cache current block receive time (ms):	3.2
Avg global cache cr block build time (ms):	0.1
Avg global cache cr block send time (ms):	0.0
Global cache log flushes for cr blocks served %:	5.0
Avg global cache cr block flush time (ms):	4380.0
Avg global cache current block pin time (ms):	0.0
Avg global cache current block send time (ms):	0.1
Global cache log flushes for current blocks served %:	0.1
Avg global cache current block flush time (ms):	0.0

---

## LGWR priority

- LGWR processes should also run with higher priority, in addition to LMS processes.
- Better write throughput on redo log files is essential for overall RAC performance.
- High interconnect block transfer inevitably will result in hyper active LGWR.
- Increase priority for LGWR and LMS (Example for Solaris)

```
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}`
```

---

## 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 applicable to only servers with high # of CPUs such as E25K platforms.

---

# 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.

# Agenda - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC.
- Bitmap index performance is worse compared to single instance.

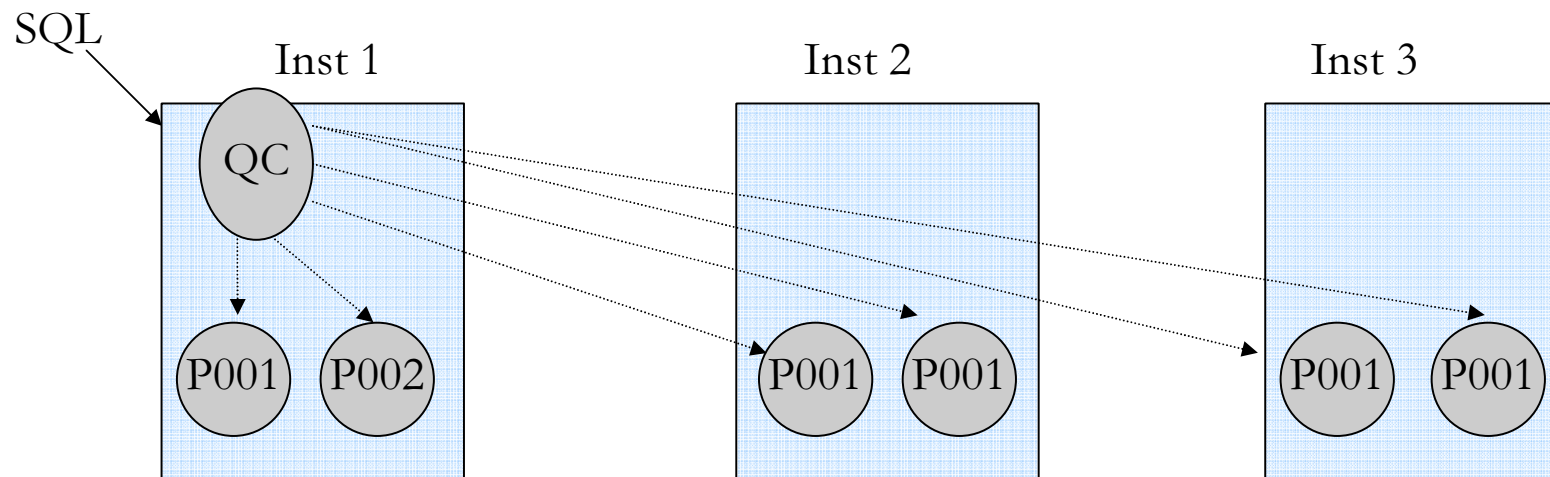
---

# Parallelism

- Few parameters controls this behaviour:
  - `parallel_min_servers`
  - `parallel_max_servers`
- Two more parameters, RAC specific:
  - `instance_group`
  - `parallel_instance_group`
- In a multi-instance RAC cluster, we can control parallelism to specific instances.

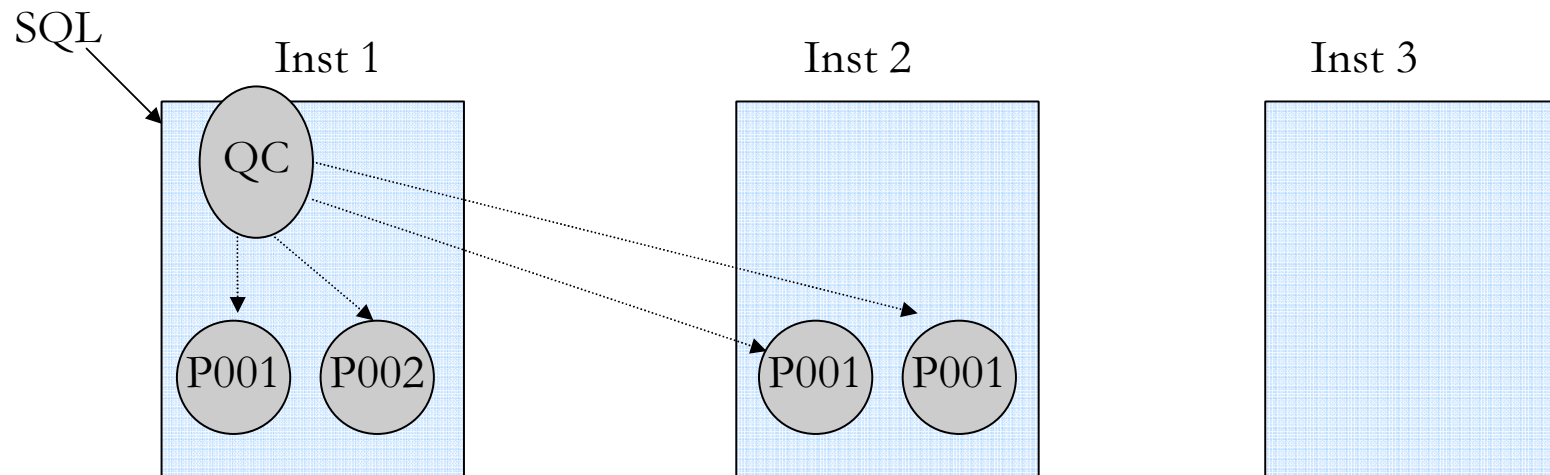
# Parallelism

- Let's say that there are three instances: inst1, inst2, inst3.
- To span slaves across all instances
  - `inst1.instance_groups='inst1','all'`
  - `inst2.instance_groups='inst2','all'`
  - `inst3.instance_groups='inst3','all'`
  - `inst1.parallel_instance_group='all'`



# Parallelism

- To span slaves across all instances inst1 and inst2 alone, parameters will be:
  - `inst1.instance_groups='inst1','all','inst12'`
  - `inst2.instance_groups='inst2','all','inst12'`
  - `inst3.instance_groups='inst3','all'`
  - `inst1.parallel_instance_group='inst12'`



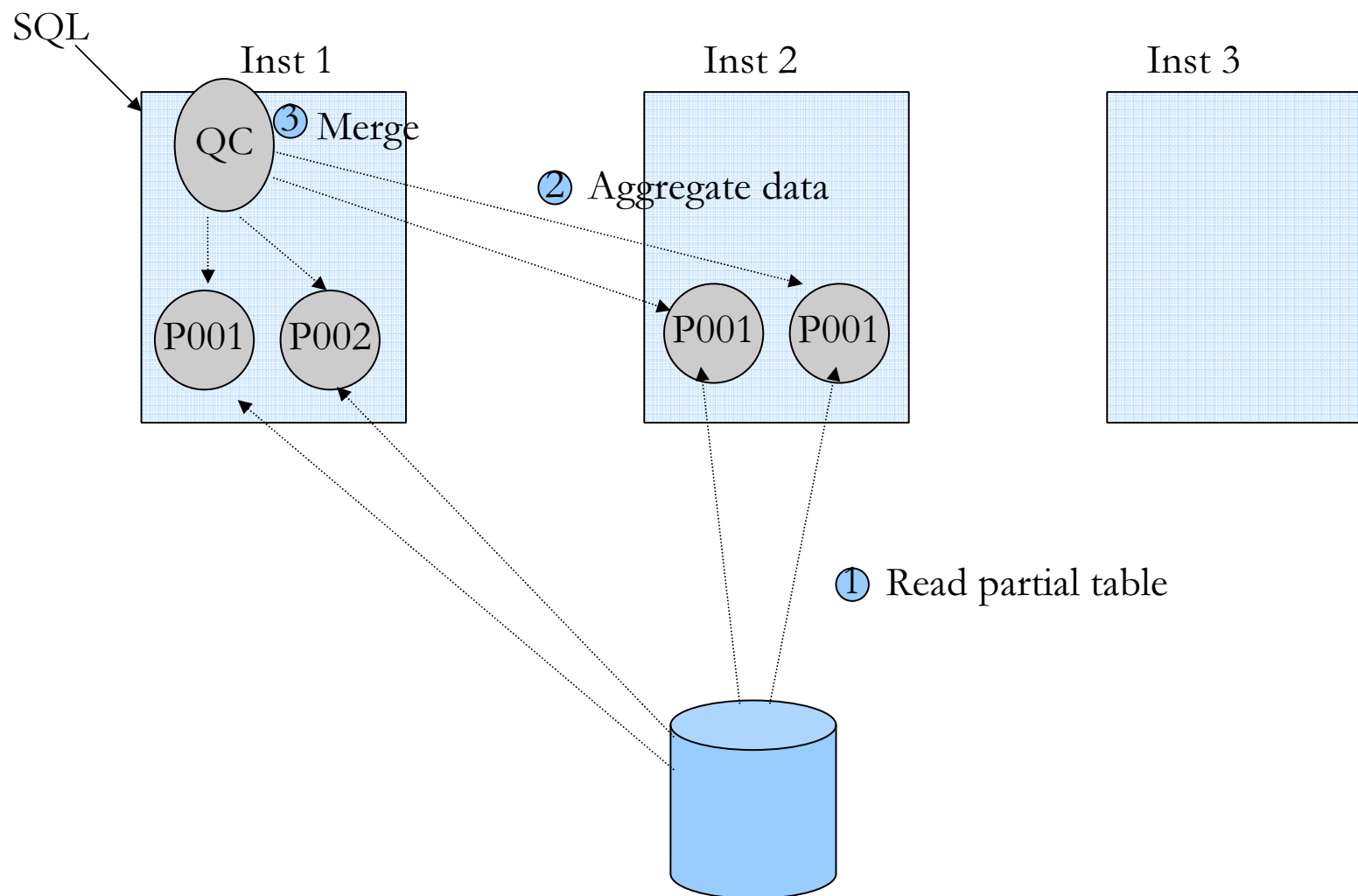
# Parallel Select

```
Alter session set parallel_instance_group='inst12';
select /*+ full(t1) parallel (t1,4) */
avg(n1), max(n1), avg(n2), max(n2), max(v1)
from t_large t1;
```

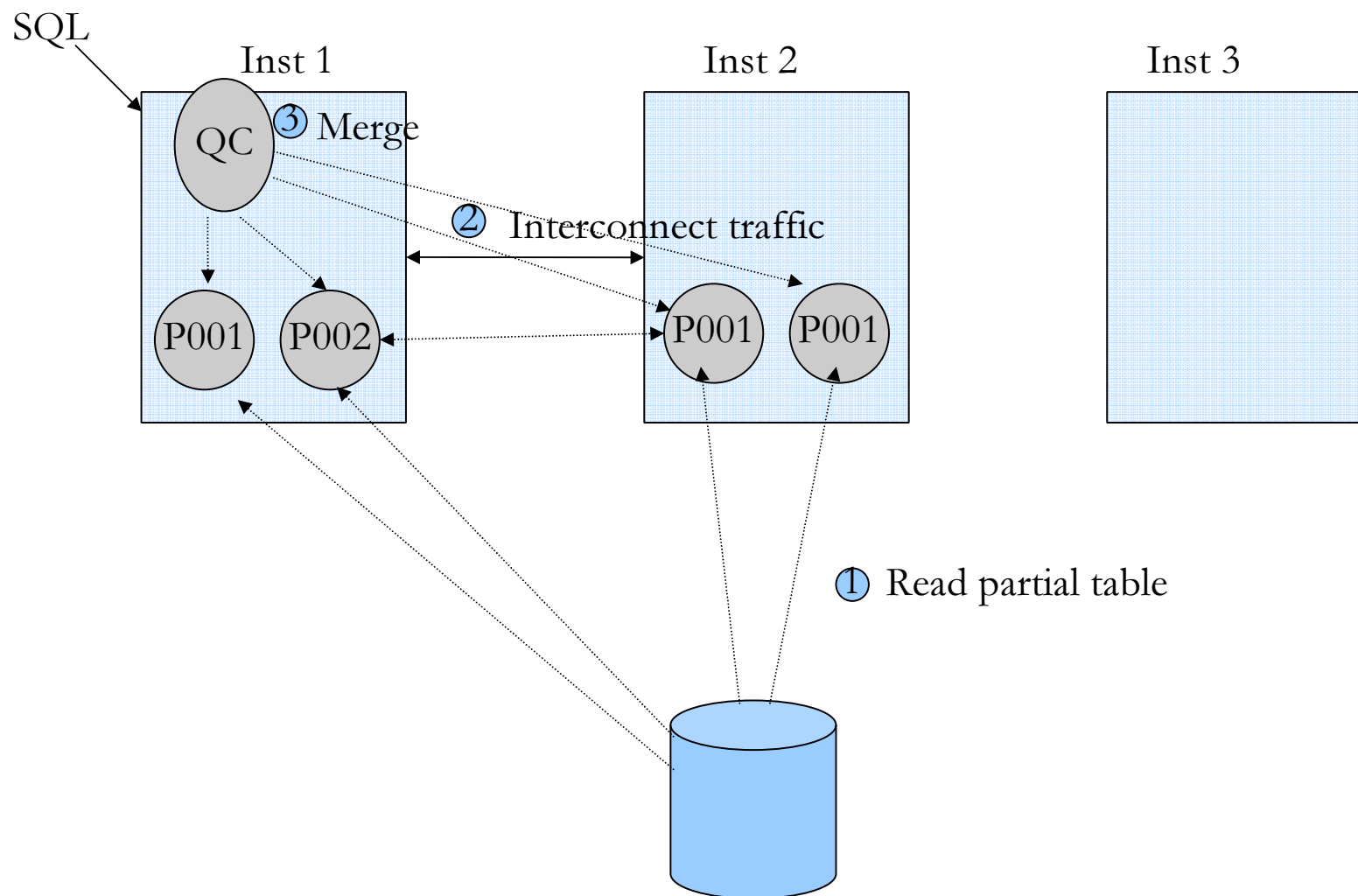
- Four slaves were allocated for above SQL statement.

Username	QC/Slave	Slave Set	SID	QC SID	Requested DOP	Actual DOP	INST_ID
CBQT	QC		140	140		1	
- p001	(Slave)	1	138	140	4	4	1
- p000	(Slave)	1	152	140	4	4	1
- p000	(Slave)	1	121	140	4	4	2
- p001	(Slave)	1	126	140	4	4	2

# PQ select – In ideal situation



# PQ select – actual processing



# Parallel Select

```
select /*+ full(t1) parallel (t1,4) */  
avg(n1), max(n1), avg(n2), max(n2), max(v1)  
from t_large t1;
```

Elapsed time reduced from 191  
seconds to 91 seconds.

```
Alter session set parallel_instance_group = 'ALL';
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	3	0.00	0.02	0	0	0	0
Execute	3	0.00	1.27	0	9	0	0
Fetch	6	69.90	189.92	0	0	0	3
total	12	69.91	191.22	0	9	0	3

```
Alter session set parallel_instance_group = 'ORCL1';
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.05	0	0	0	0
Execute	1	0.00	30.63	0	3	0	0
Fetch	2	7.48	60.69	0	0	0	1
total	4	7.50	91.38	0	3	0	1

---

# 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 may be better.
- Excessive inter instance parallelism will increase interconnect traffic leading to performance issues.

# Agenda - Myths

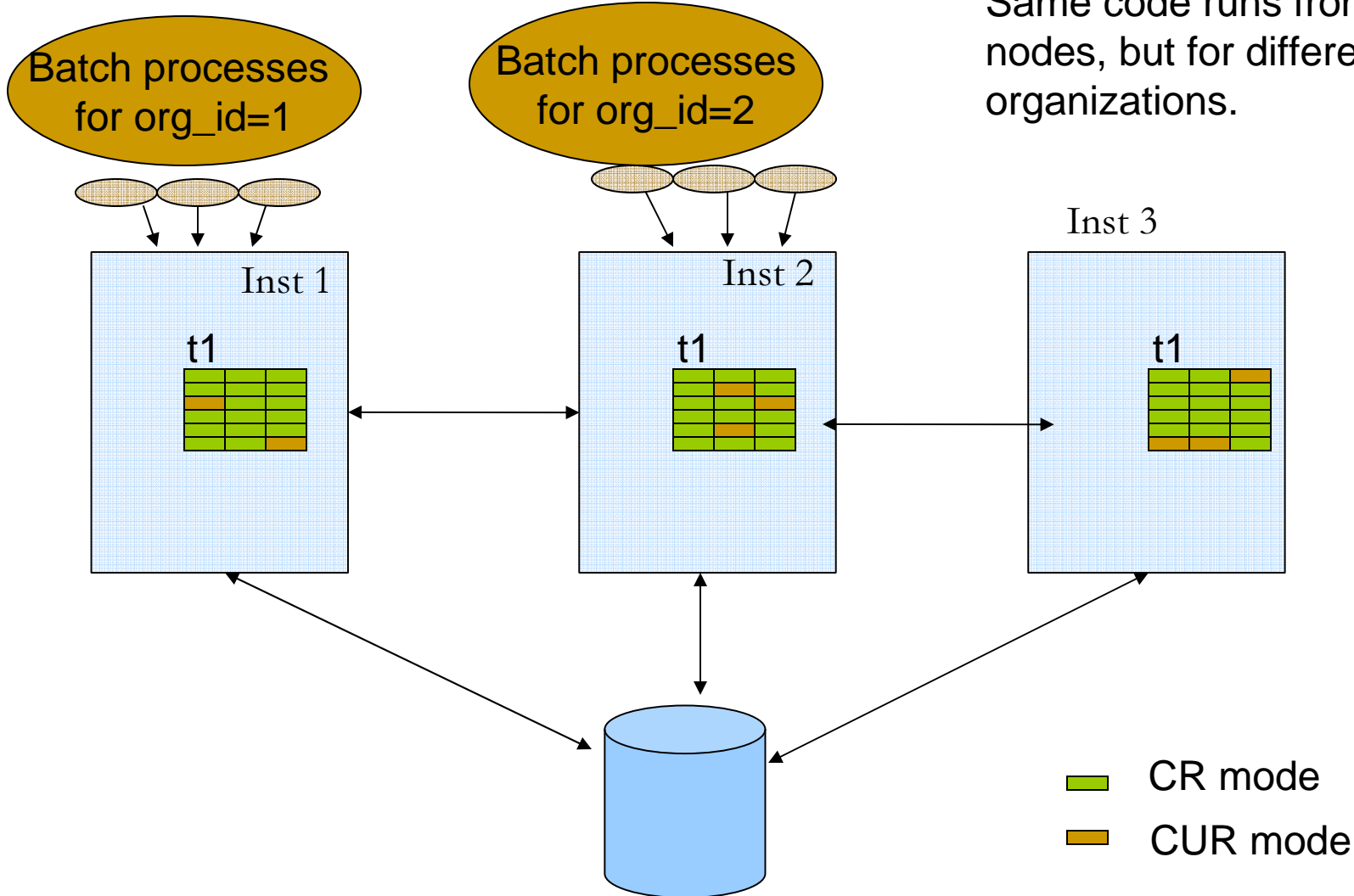
- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC.
- Bitmap index performance is worse compared to single instance.

---

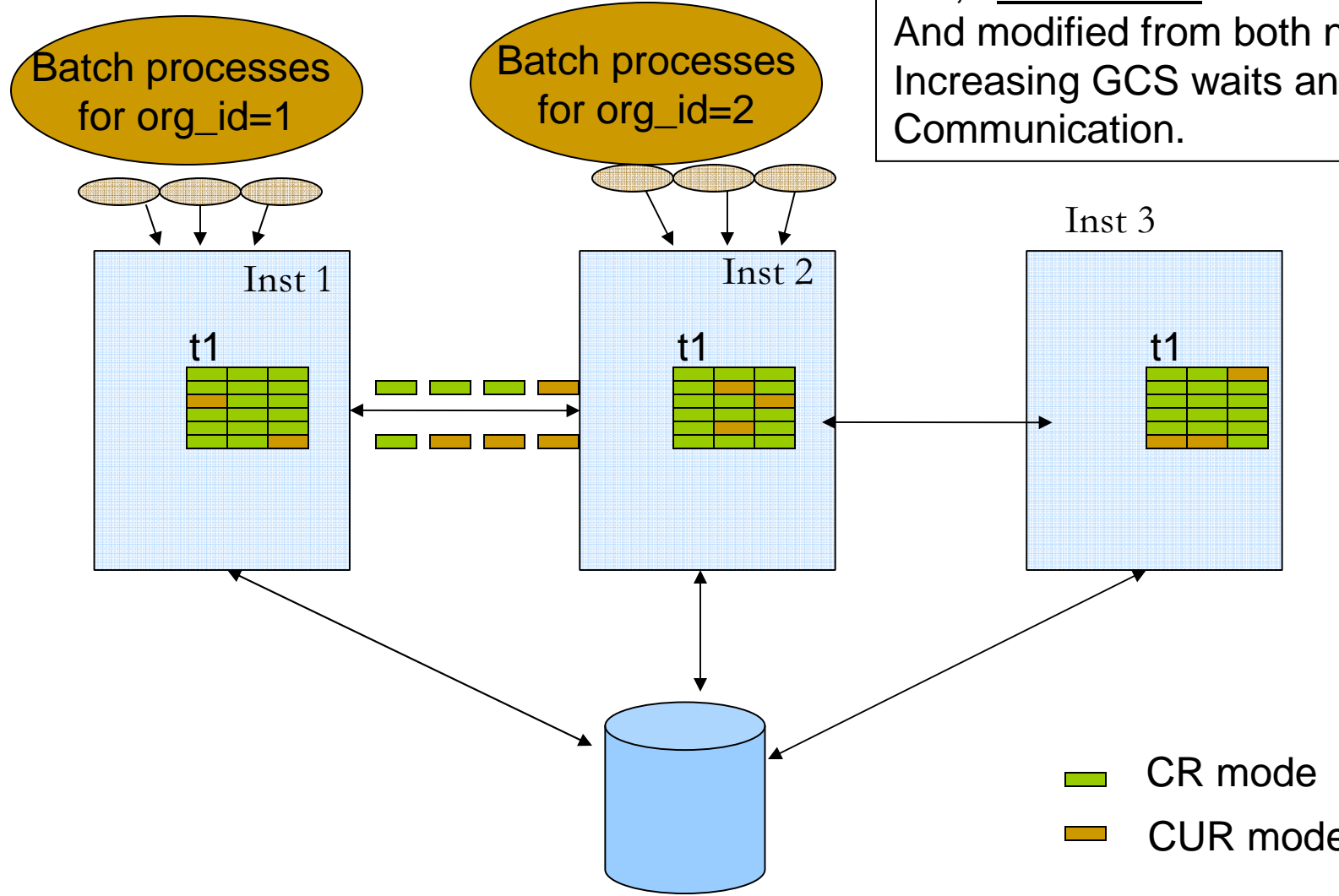
# Concurrent processing?

- A batch process was designed to accept `organization_id` as an argument, and spawns multiple threads for each organization.
- This is multi-node RAC cluster and so
  - Batch process was started in node 1 for `org_id=1`.
  - Same batch process was started in node 2 for `org_id=2`.
- But batch code accepts just single set of tables and they are not partitioned.

# Concurrent processing?



# Concurrent processing?



But, same blocks are accessed  
And modified from both nodes,  
Increasing GCS waits and GCS  
Communication.

---

## Are GCS waits cheap?

- Typical Global cache waits are 1-3ms for a decent hardware setup.
- Typical Single block or multi-block reads are 2-4ms for decent hardware.
- But typical access to the buffer in local SGA is just nano seconds (10-100ns).
- Higher global cache waits still introduces latency comparable to I/O based latencies.

# What happened?

- Enormous increase in Global cache waits!

(5 minutes statspack)

Top 5 Timed Events

~~~~~

| Event                   | Waits   | Time (s) | % Total<br>Ela Time |
|-------------------------|---------|----------|---------------------|
| -----                   | -----   | -----    | -----               |
| CPU time                |         | 3,901    | 46.17               |
| PL/SQL lock timer       | 80      | 2,316    | 27.42               |
| global cache cr request | 123,725 | 670      | 7.93                |
| global cache null to x  | 13,551  | 446      | 5.28                |
| buffer busy global CR   | 12,383  | 215      | 2.54                |

- If this has been a non-RAC, access to buffers would be in terms of nano seconds instead of milli-seconds.
- Point is that since these batch processes are accessing same physical blocks, they need to be grouped at physical level and scheduled to run from one node.
- Proper partitioning will help, but global indices needs special attention.

# What happened?

- Trace file analysis also shows excessive global cache waits.

Elapsed times include waiting on following events:

| Event waited on                     | Times<br>waited | Max. Wait | Total waited |
|-------------------------------------|-----------------|-----------|--------------|
| log file sync                       | 13              | 0.08      | 0.28         |
| latch free                          | 2185            | 0.17      | 32.76        |
| global cache cr request             | 21719           | 0.31      | 71.18        |
| db file sequential read             | 50              | 0.04      | 0.33         |
| global cache s to x                 | 791             | 0.06      | 5.31         |
| row cache lock                      | 113             | 0.02      | 0.41         |
| global cache null to x              | 4881            | 0.30      | 97.41        |
| buffer busy global CR               | 3306            | 0.23      | 15.26        |
| global cache open x                 | 903             | 0.06      | 5.83         |
| buffer busy waits                   | 1462            | 0.98      | 17.90        |
| global cache busy                   | 644             | 0.98      | 10.36        |
| buffer deadlock                     | 224             | 0.02      | 0.07         |
| global cache null to s              | 89              | 0.15      | 0.77         |
| buffer busy global cache            | 1066            | 0.31      | 27.62        |
| enqueue                             | 302             | 0.17      | 2.61         |
| KJC: wait for msg sends to complete | 102             | 0.00      | 0.00         |
| global cache open s                 | 62              | 0.01      | 0.13         |
| cr request retry                    | 4               | 0.00      | 0.00         |

# Agenda - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC.
- Bitmap index performance is worse compared to single instance.

# Sequence operation in RAC

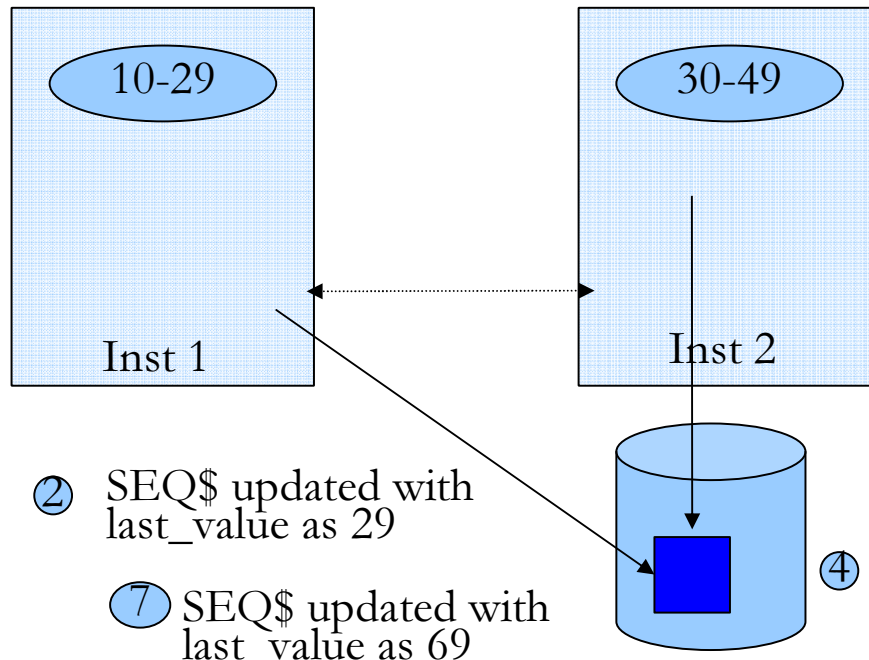
emp\_seq  
cache 20  
start with 10

⑥ After 29, values will be in 50-69 range.

⑤ Subsequent accesses returns values until value reaches 29

① First access to sequence caches values from 10 to 29

③ Second access caches value from 30-49



1. 60 access to sequence results in 3 changes to block.
2. These changes might not result in physical reads/writes.
3. Gaps in sequence values.
4. Still, log flush needed for cache transfer.

# Sequence operation in RAC

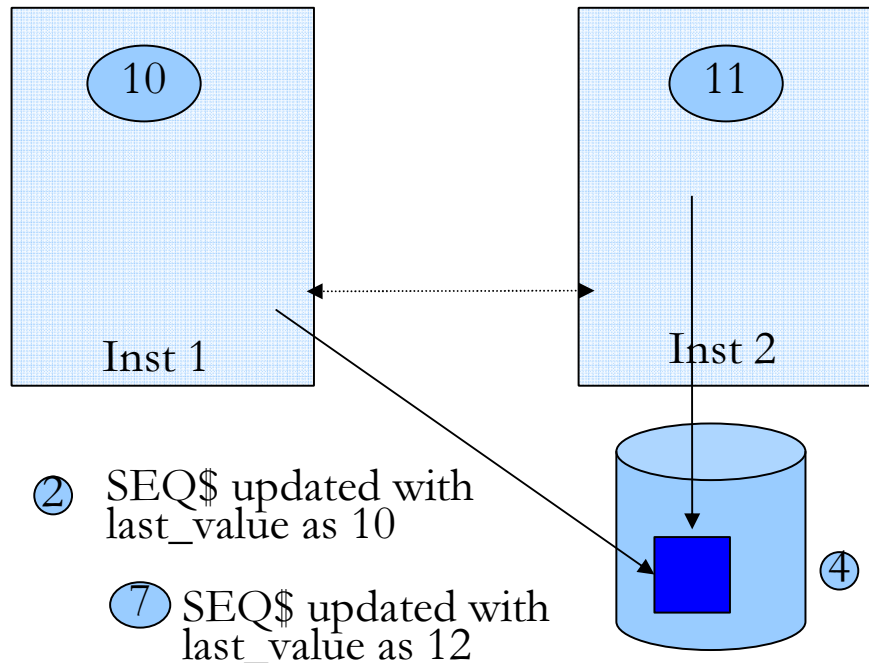
emp\_seq  
nocache  
start with 10

⑥ Due to nocache values,  
there will be no gaps.

⑤ Subsequent accesses returns  
value 12

① First access to sequence  
returns value 10

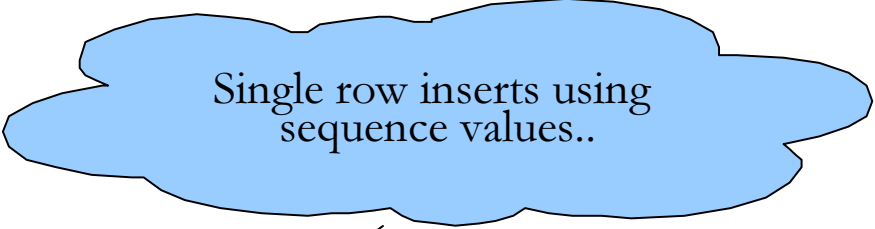
③ Second access returns  
value of 11



1. 3 access to sequence results in 3 block changes.
2. No gaps in sequence values.
3. But, SEQ\$ table blocks transferred back and forth.

# Sequences – Test case

```
set timing on
alter session set events '10046 trace name context forever, level 8';
declare
  l_v1 varchar2(512);
  l_n1 number :=0;
begin
  for loop_cnt in 1 .. 10000
  loop
    -- Random access
    -- Also making undo blocks to be pinged..
    insert into t1
    select t1_seq.nextval, lpad(loop_cnt, 500, 'x') from dual;
    if mod(loop_cnt, 1000) =0 then
      commit;
    end if;
  end loop;
end;
/
```



Single row inserts using  
sequence values..

# Code executions – one node

```
INSERT INTO T1 SELECT T1_SEQ.NEXTVAL, LPAD( :B1 , 500, 'x') FROM DUAL
```

| call    | count | cpu  | elapsed | disk | query | current | rows  |
|---------|-------|------|---------|------|-------|---------|-------|
| Parse   | 1     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| Execute | 10000 | 5.28 | 7.66    | 1    | 794   | 25670   | 10000 |
| Fetch   | 0     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| total   | 10001 | 5.29 | 7.66    | 1    | 794   | 25670   | 10000 |

```
update seq$ set increment$=:2,minvalue=:3,maxvalue=:4,cycle#=:5,order$=:6,  
cache=:7,highwater=:8,audit$=:9,flags=:10 where obj#=:1
```

| call    | count | cpu  | elapsed | disk | query | current | rows  |
|---------|-------|------|---------|------|-------|---------|-------|
| Parse   | 10000 | 0.32 | 0.30    | 0    | 0     | 0       | 0     |
| Execute | 10000 | 2.74 | 3.04    | 0    | 10000 | 20287   | 10000 |
| Fetch   | 0     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| total   | 20000 | 3.06 | 3.34    | 0    | 10000 | 20287   | 10000 |

# Code executions – two nodes

```
INSERT INTO T1 SELECT T1_SEQ.NEXTVAL, LPAD( :B1 , 500, 'x') FROM DUAL
```

| call    | count | cpu  | elapsed | disk | query | current | rows  |
|---------|-------|------|---------|------|-------|---------|-------|
| Parse   | 1     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| Execute | 10000 | 8.02 | 81.23   | 0    | 1584  | 27191   | 10000 |
| Fetch   | 0     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| total   | 10001 | 8.02 | 81.23   | 0    | 1584  | 27191   | 10000 |

Excessive row cache lock  
waits

Elapsed times include waiting on following events:

| Event waited on        | Times<br>Waited | Max. wait | Total waited |
|------------------------|-----------------|-----------|--------------|
| row cache lock         | 5413            | 2.93      | 62.86        |
| gc current block 2-way | 63              | 0.16      | 0.41         |
| gc cr block 2-way      | 46              | 0.00      | 0.06         |

# Code executions – two nodes

5000 blocks transferred  
between nodes..

```
update seq$ set increment$=:2,minvalue=:3,maxvalue=:4,cycle#=:5,order$=:6,  
  cache=:7,highwater=:8,audit$=:9,flags=:10
```

```
where obj#=:1
```

| call    | count | cpu  | elapsed | disk | query | current | rows  |
|---------|-------|------|---------|------|-------|---------|-------|
| Parse   | 10000 | 0.35 | 0.30    | 0    | 0     | 0       | 0     |
| Execute | 10000 | 4.08 | 11.18   | 0    | 10000 | 20290   | 10000 |
| Fetch   | 0     | 0.00 | 0.00    | 0    | 0     | 0       | 0     |
| total   | 20000 | 4.44 | 11.49   | 0    | 10000 | 20290   | 10000 |

| Event waited on | Times<br>waited | Max. wait | Total waited |
|-----------------|-----------------|-----------|--------------|
|-----------------|-----------------|-----------|--------------|

|                            |      |      |      |
|----------------------------|------|------|------|
| gc current block 2-way     | 5166 | 0.01 | 5.39 |
| log file switch completion | 3    | 0.16 | 0.22 |
| gc current grant busy      | 1    | 0.00 | 0.00 |

---

## Sequence- summary

- Nocache sequences increases 'row cache lock' waits.
- Increases interconnect traffic.
- Increases elapsed time.
- If no gaps are needed, control sequence access from just one node or use non-sequence based techniques.

# Agenda - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC.
- Bitmap index performance is worse compared to single instance.

---

## Small tables

- Even small tables must be indexed.
- Excessive full table scans on smaller tables will increase CPU usage.
- This guideline applies to RAC environments too.
- I think, this myth arises due to misunderstanding of the problem.

---

# Small tables

```
set timing on
drop table t_small2;
create table t_small2 (n1 number, v1 varchar2(10) ) tablespace users
;
insert into t_small2 select n1, lpad(n1,10,'x')
from (select level n1 from dual connect by level <=10001 );
commit;
```

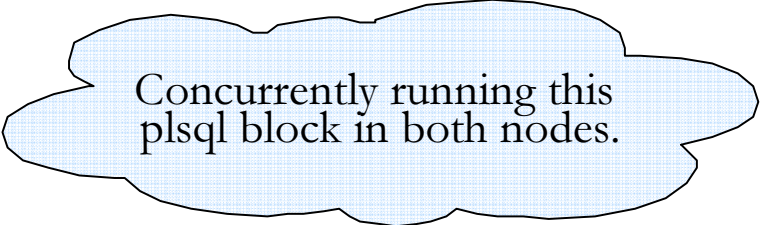
```
select segment_name, sum(bytes)/1024 from dba_segments where segment_name='T_SMALL2'
and owner='CBQT' group by segment_name
```

```
SQL> /
```

| SEGMENT_NAME | SUM(BYTES)/1024 |
|--------------|-----------------|
| T_SMALL2     | 256             |

# Test case

```
alter session set events '10046 trace name context forever , level 8';
set serveroutput on size 100000
declare
  v_n1 number;
  v_v1 varchar2(512);
  b_n1 number;
begin
  for i in 1 .. 100000 loop
    b_n1 := trunc(dbms_random.value (1,10000));
    select n1, v1 into v_n1, v_v1 from t_small2 where n1 =b_n1;
  end loop;
exception
  when no_data_found then
    dbms_output.put_line (b_n1);
end;
/
```



Concurrently running this  
plsql block in both nodes.

# Results from RAC nodes.

```
SELECT N1, V1
```

```
FROM
```

```
  T_SMALL2 WHERE N1 =:B1
```

| call    | count  | cpu   | elapsed | disk | query   | current | rows   |
|---------|--------|-------|---------|------|---------|---------|--------|
| Parse   | 1      | 0.00  | 0.00    | 0    | 0       | 0       | 0      |
| Execute | 100000 | 2.81  | 3.08    | 0    | 1       | 0       | 0      |
| Fetch   | 100000 | 62.72 | 63.71   | 0    | 3100000 | 0       | 100000 |
| total   | 200001 | 65.54 | 66.79   | 0    | 3100001 | 0       | 100000 |

63 seconds of CPUs consumed

```
Rows      Row Source Operation
```

```
100000    TABLE ACCESS FULL T_SMALL2 (cr=3100000 pr=0 pw=0 time=63391728 us)⌈
```

# Results with an index

REM adding an index and repeating test  
create index t\_small12\_n1 on t\_small12(n1);

CPU usage dropped from  
63 seconds to 3.5 seconds.

| call    | count  | cpu  | elapsed | disk | query  | current | rows   |
|---------|--------|------|---------|------|--------|---------|--------|
| Parse   | 1      | 0.00 | 0.00    | 0    | 0      | 0       | 0      |
| Execute | 100000 | 1.64 | 1.61    | 0    | 2      | 0       | 0      |
| Fetch   | 100000 | 1.79 | 1.78    | 23   | 300209 | 0       | 100000 |
| total   | 200001 | 3.43 | 3.40    | 23   | 300211 | 0       | 100000 |

Rows Row Source Operation

```
-----  
100000 TABLE ACCESS BY INDEX ROWID T_SMALL2 (cr=300209 pr=23 pw=0 time=1896719 us)⌈  
100000 INDEX RANGE SCAN T_SMALL2_N1 (cr=200209 pr=23 pw=0 time=1109464 us)(object id  
53783)⌈
```

# Agenda - Myths

- High CPU usage in one node doesn't affect other node performance.
- All global cache performance issues are due to interconnect performance.
- Inter instance parallelism is excellent, since CPUs from all nodes can be effectively used.
- Logical isolation is good enough, so, run same batch process concurrently from both nodes.
- Set sequence to nocache value in RAC environments to avoid gaps in sequence.
- Small tables should not be indexed in RAC.
- Bitmap index performance is worse compared to single instance.

---

# Bitmap index

- Bitmap indices are optimal for low cardinality columns.
- Bitmap indices are not suitable for table with massive DML changes.
- Bitmap index performance does not worsen because of RAC for select queries.
- Of course, having bitmap indices on columns with enormous DML changes is not optimal even in single instance databases.

---

# Test case - Select

```
Create bitmap index t_large2_n4 on t_large2(n4);

alter session set events '10046 trace name context forever , level 8';
set serveroutput on size 100000
declare
    v_n1 number;
    v_v1 varchar2(512);
    b_n1 number;
begin
    for i in 1 .. 100000 loop
        b_n1 := trunc(dbms_random.value (1,10000));
        select count(*) into v_n1 from t_large2 where n4 =b_n1;
    end loop;
exception
    when no_data_found then
        dbms_output.put_line (b_n1);
end;
/
```

---

# Result – Single thread

```
SELECT COUNT(*) FROM T_LARGE2 WHERE N4 =:B1
```

| call    | count  | cpu  | elapsed | disk | query  | current | rows   |
|---------|--------|------|---------|------|--------|---------|--------|
| Parse   | 1      | 0.00 | 0.00    | 0    | 0      | 0       | 0      |
| Execute | 100000 | 2.87 | 2.93    | 2    | 2      | 0       | 0      |
| Fetch   | 100000 | 1.86 | 2.03    | 78   | 200746 | 0       | 100000 |
| total   | 200001 | 4.73 | 4.97    | 80   | 200748 | 0       | 100000 |

| Rows   | Row Source Operation                                            |
|--------|-----------------------------------------------------------------|
| 100000 | SORT AGGREGATE (cr=200746 pr=78 pw=0 time=2854389 us)†          |
| 100000 | BITMAP CONVERSION COUNT (cr=200746 pr=78 pw=0 time=1766444 us)† |

# Result – From two nodes

```
SELECT COUNT(*)  
FROM  
  T_LARGE2 WHERE N4 =:B1
```

| call    | count  | cpu  | elapsed | disk | query  | current | rows   |
|---------|--------|------|---------|------|--------|---------|--------|
| Parse   | 1      | 0.00 | 0.01    | 0    | 0      | 0       | 0      |
| Execute | 100000 | 2.82 | 2.95    | 0    | 2      | 0       | 0      |
| Fetch   | 100000 | 1.90 | 1.94    | 3    | 200753 | 0       | 100000 |
| total   | 200001 | 4.73 | 4.90    | 3    | 200755 | 0       | 100000 |

Misses in library cache during parse: 1

---

# References

- Oracle support site. Metalink.oracle.com. Various documents
- Internal's guru Steve Adam's website

[www.ixora.com.au](http://www.ixora.com.au)

- Jonathan Lewis' website

[www.jlcomp.daemon.co.uk](http://www.jlcomp.daemon.co.uk)

- Julian Dyke's website

[www.julian-dyke.com](http://www.julian-dyke.com)

- 'Oracle8i Internal Services for Waits, Latches, Locks, and Memory'  
by Steve Adams

- Tom Kyte's website

Asktom.oracle.com