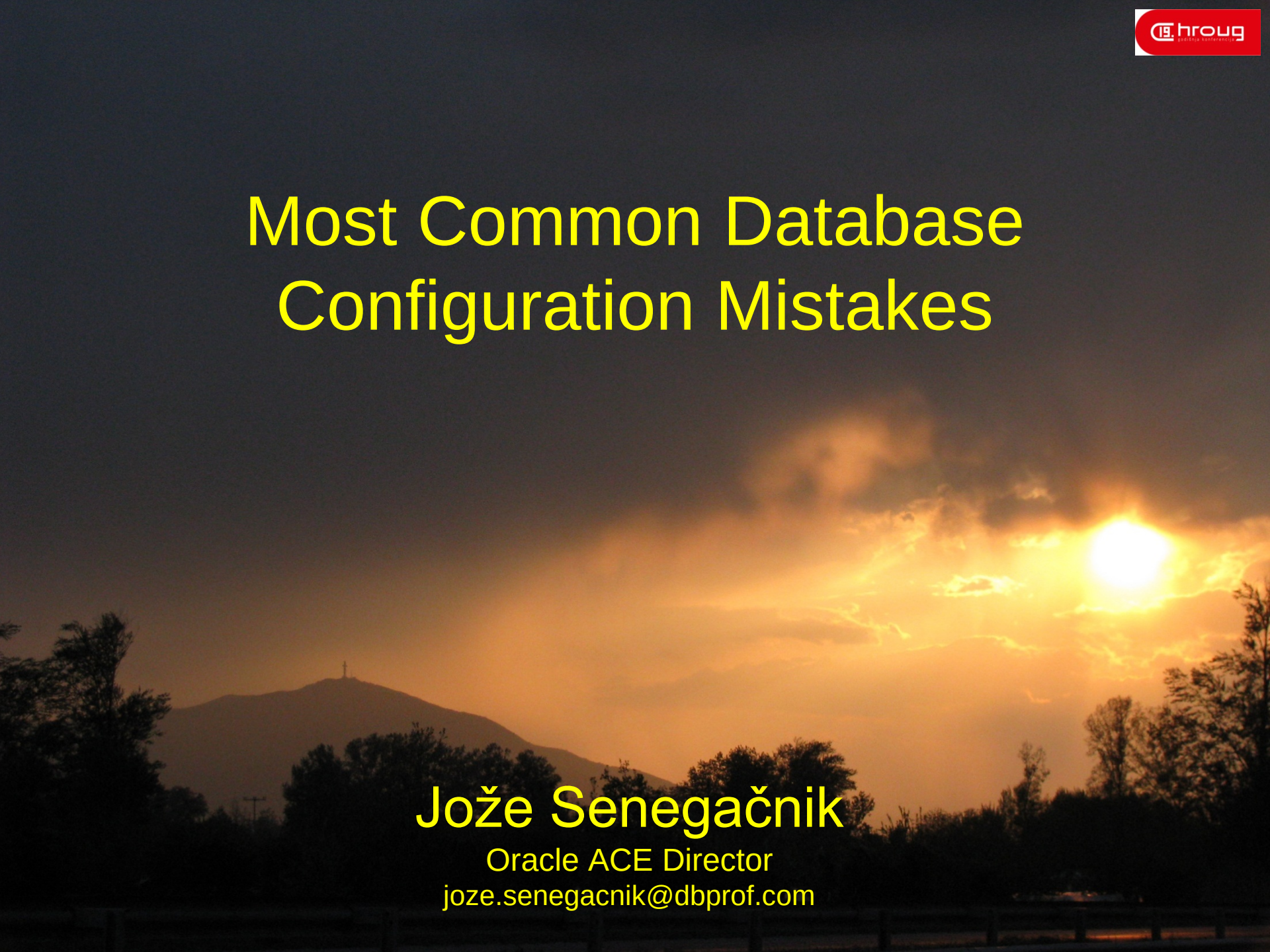


Most Common Database Configuration Mistakes

Jože Senegačnik
Oracle ACE Director
joze.senegacnik@dbprof.com

About the Speaker

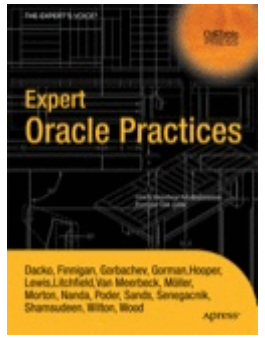
Jože Senegačnik

- First experience with Oracle Version 4 in 1988
- 26 years of experience with Oracle RDBMS.
- Proud member of the OakTable Network www.oaktable.net
- Oracle ACE Director
- Co-author of the OakTable book “Expert Oracle Practices” by Apress (Jan 2010)
- VP of Slovenian OUG (SIOUG) board
- CISA – Certified IS auditor
- Blog about Oracle: http://joze-senegacnik.blogspot.com

OakTable.net

ORACLE ACE Director

Oracle ACE



- PPL(A) – private pilot license PPL(A) / instrument rated IR/SE
- Blog about flying: http://jsenegacnik.blogspot.com
- Blog about Building Ovens, Baking and Cooking: http://senegacnik.blogspot.com

Agenda

- The growth of the size of databases, multi-tier architecture and big numbers of end users are creating new threat for proper configuration.
  - **Connection Configuration**
  - **Memory Configuration**
- The hardware is getting more and more powerful (a lot of CPUs), server's memory is measured in terabytes what makes people so comfortable about performance that they make huge mistakes in proper application development.
  - **Bad Application Design / Coding**

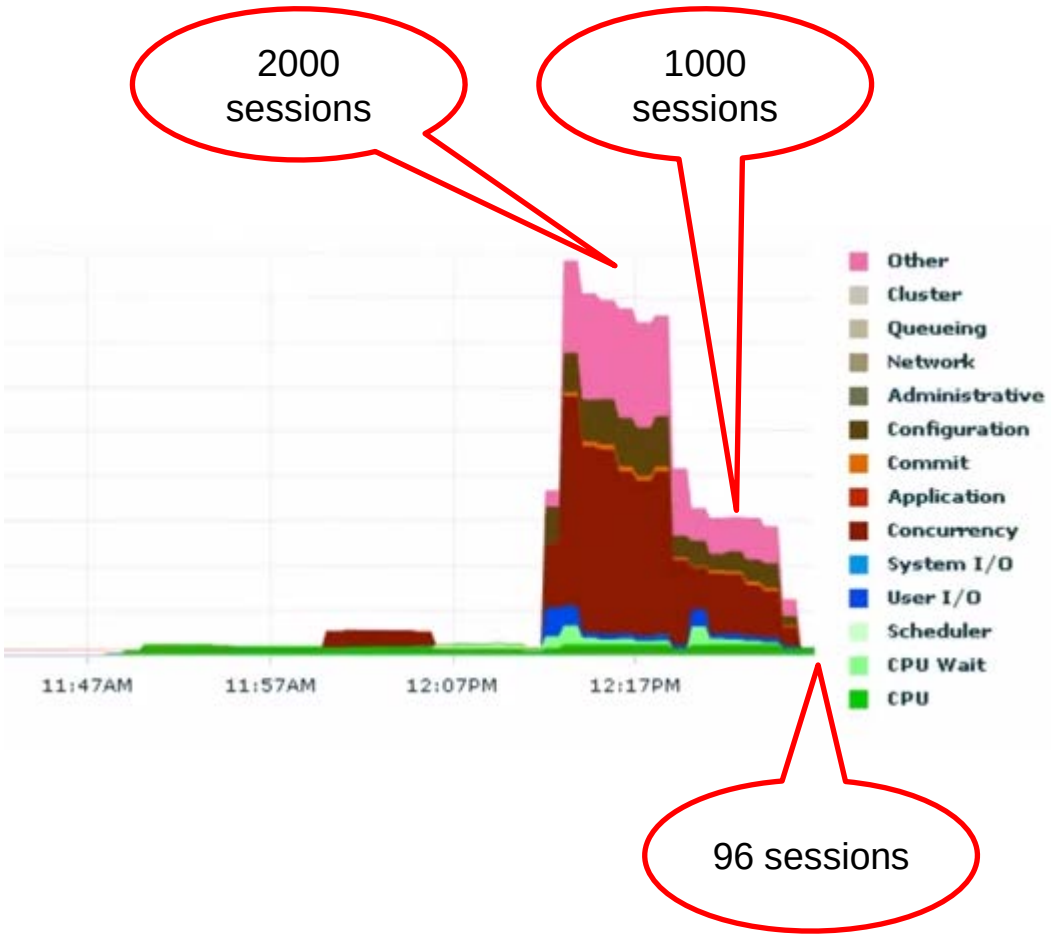
Connection Management

Inadequate Configured Middle Tier

I#	Logical Reads	Physical Reads	Physical Writes	Redo Size (k)	Block Changes	User Calls	Execs	Parses	Logons	Txns
1	242,705,688	120,829,400	818,867	2,266,319	9,340,200	9,425,782	5,802,705	1,510,228	34,683	767,741
2	392,159,540	304,830,326	3,354,542	5,222,460	21,823,043	8,821,932	9,467,687	1,807,293	34,237	,001,425
3	5,086,625,512	590,409,453	32,688,891	68,817,340	257,486,609	6,778,625	107,297,979	10,087,458	54,905	,609,989
4	4,895,970,684	550,659,841	30,405,360	88,305,236	335,134,092	6,428,450	123,959,286	9,161,039	110,292	,285,532
Sum	10,617,461,424	1,566,729,020	67,267,660	164,611,354	623,783,944	31,454,789	246,527,657	22,566,018	234,117	,664,687
Avg	2,654,365,356	391,682,255	16,816,915	41,152,839	155,945,986	7,863,697	61,631,914	5,641,505	58,529	,166,172
Std	2,700,269,210	220,356,453	17,065,936	43,938,695	165,228,222	1,482,750	62,737,809	4,615,996	35,830	363,827

- **Poor connection pool strategy**
  - High number connections to the database (1,000s)
  - Dynamic connection pool with a large number of logon/logoff to the database (>1/Sec)
  - Periods of unexplainable/undebuggable poor performance/availability
  - Inability to determine what is going wrong
- Logon process one of the most complex in Oracle database (disconnect is even more)

Concurrent Mid-Tier Connections



- Too many concurrent sessions introduce contention and decrease performance
- Significantly reduced number of sessions increases throughput
- Database level is not the right level for queueing.

Source: OLTP Performance - Concurrent Mid-Tier Connections – Oracle Real World performance group
http://www.youtube.com/watch?v=xNDnVOCdvQ0

Connection Pool Configuration

- Properly size the connection pools - the hardest thing to convince customers
- Minimum = Maximum
- More sessions mean potentially more contention for same resources.
- Don‘t allow the spiral dive by creating new connections due to bad database response time – this will just increase the response time.

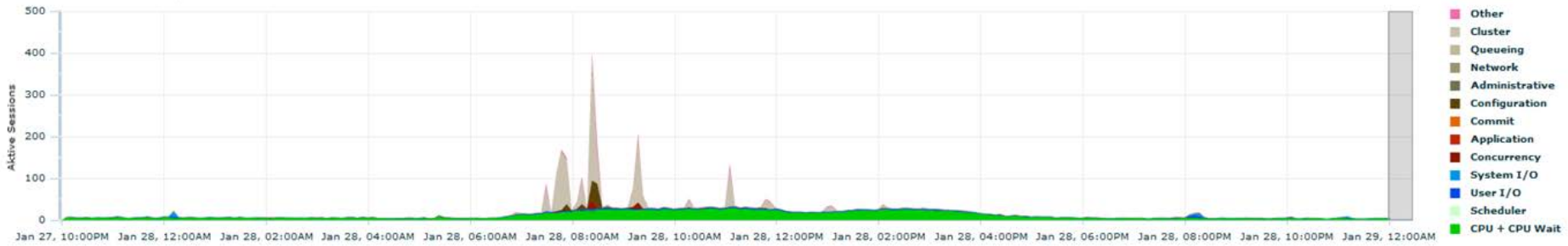
- Middle tier is the place for queueing, not database!

Questionable Values of Init. Parameters

- processes = 30,000 per Exadata node
- sessions = 45,024 per Exadata node

- Consequence:
  - Huge SGA internal tables (X$ tables are actually exposed C-structures)
  - Process/Session internal tables are fixed arrays – they are created at instance startup
  - The above numbers are totaly un-realistic

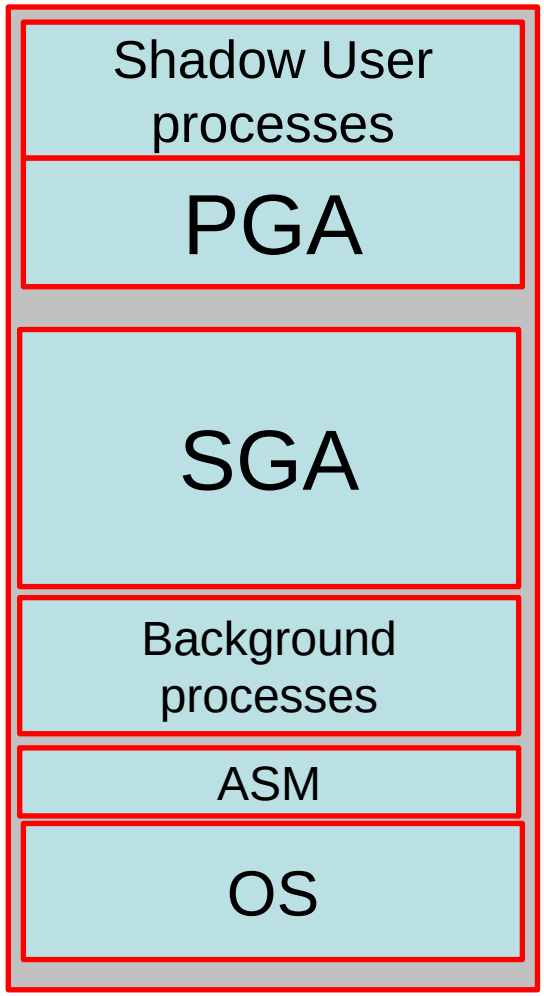
Another System With Same Problems



- > 40,000 online application users
- Middle tier ~100 application servers – Application written in Java
- Periods of performance problems with high wait times – most of them were not explainable
- 2-node RAC with 128 CPU-s per node
- 35,000 sessions per instance (70,000 overall)
- Spiral dive when response time was bad due to increased number of connections

Memory Confuguration

Memory Map

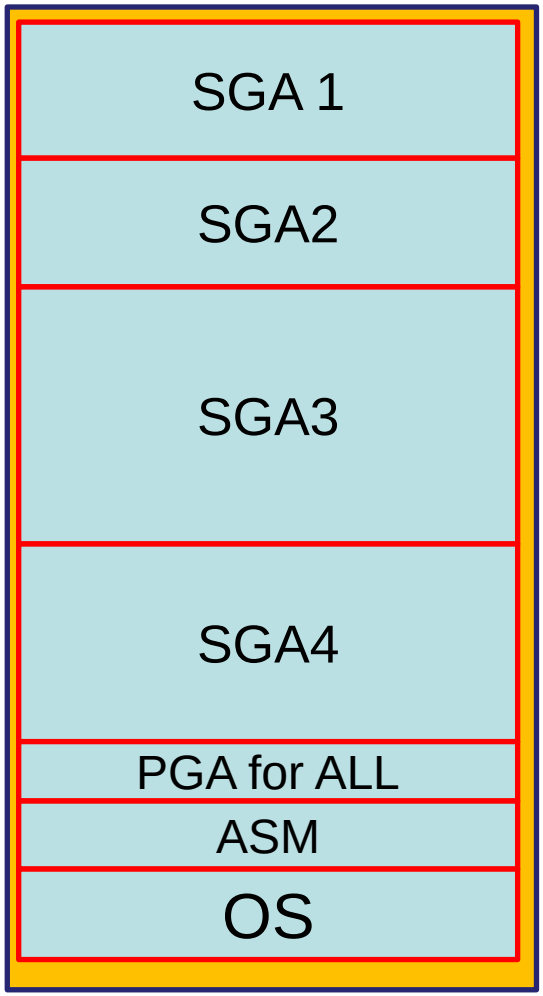


- Dedicated server mode – each session has its own server process
- Each process needs some memory space + all dynamic PGA allocations
- Each session (server process) needs some memory (PGA allocations are separate)
- Background processes need also some memory
- PGA_AGGREGATE_TARGET is only a target, not limit
- SGA (size easily detectable)

Are You Sure You Didn‘t Forget Something?

Component	Size GB	% of Total	User processes	PGA per process in MB
Total server memory	64			
SGA	26	40		
Background processes	1	2		
ASM	1	2		
User processes	30	47	2000	
PGA (aggregate)	6	9		3
	64	100		

Usually I Find Something like this:



- DBA: Oh, some of those databases are just opened, but actually nobody is using them!
- Obviously candidates for multitenant!
- Record: 80 instances on two sockets

Memory Usage Detection

- FAQ: How can I Investigate Memory Usage on my Unix/Linux Server (Doc ID 1447481.1)

- AIX: Determining Oracle Memory Usage On AIX (Doc ID 123754.1)
- How to Check and Analyze Solaris Memory Usage (Doc ID 1009500.1)
- Script for Checking the Grid Control Agent CPU, Memory & Threads Usage (Doc ID 464414.1)
- How to Calculate Memory Usage on Linux (Doc ID 1630754.1)

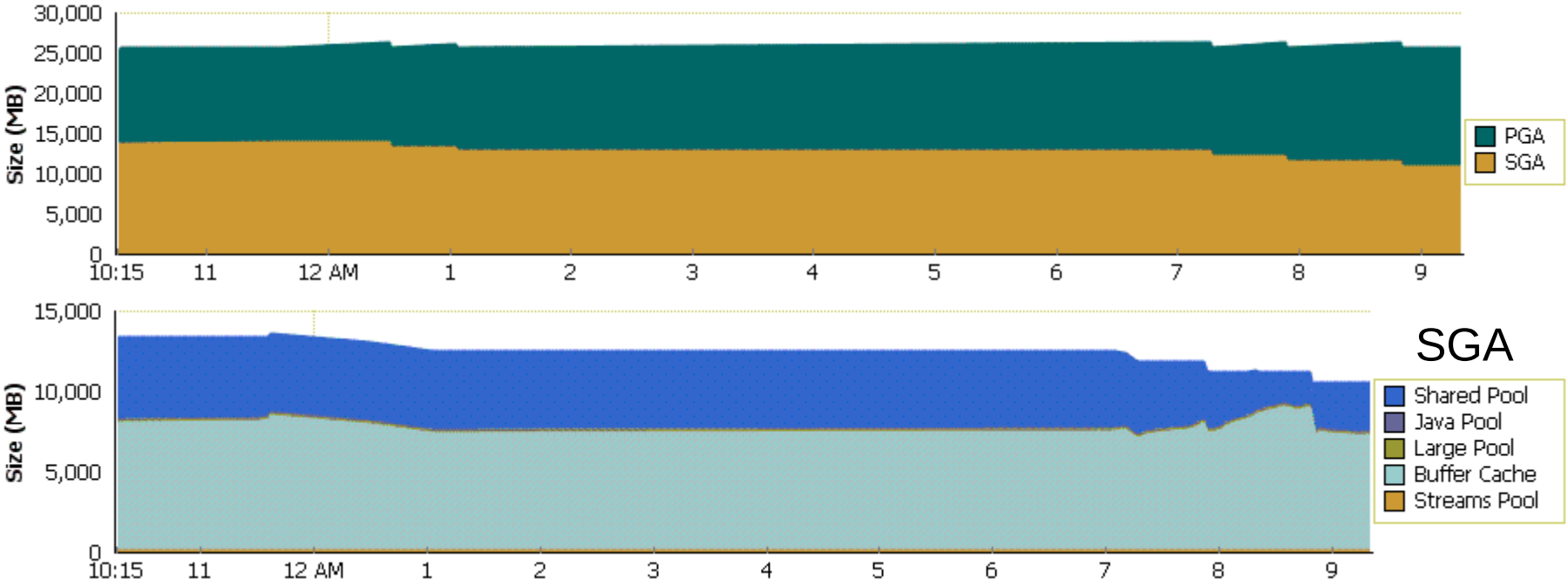
- **Tanel Poder - Using Process Memory Matrix script for understanding Oracle process memory usage**
  - http://tech.e2sn.com/oracle/performance/unix-performance-tools/process-memory-matrix
- **Oracle Instance Memory Usage**
  - http://www.pythian.com/blog/oracle-instance-memory-usage/

Finding Process Memory Usage

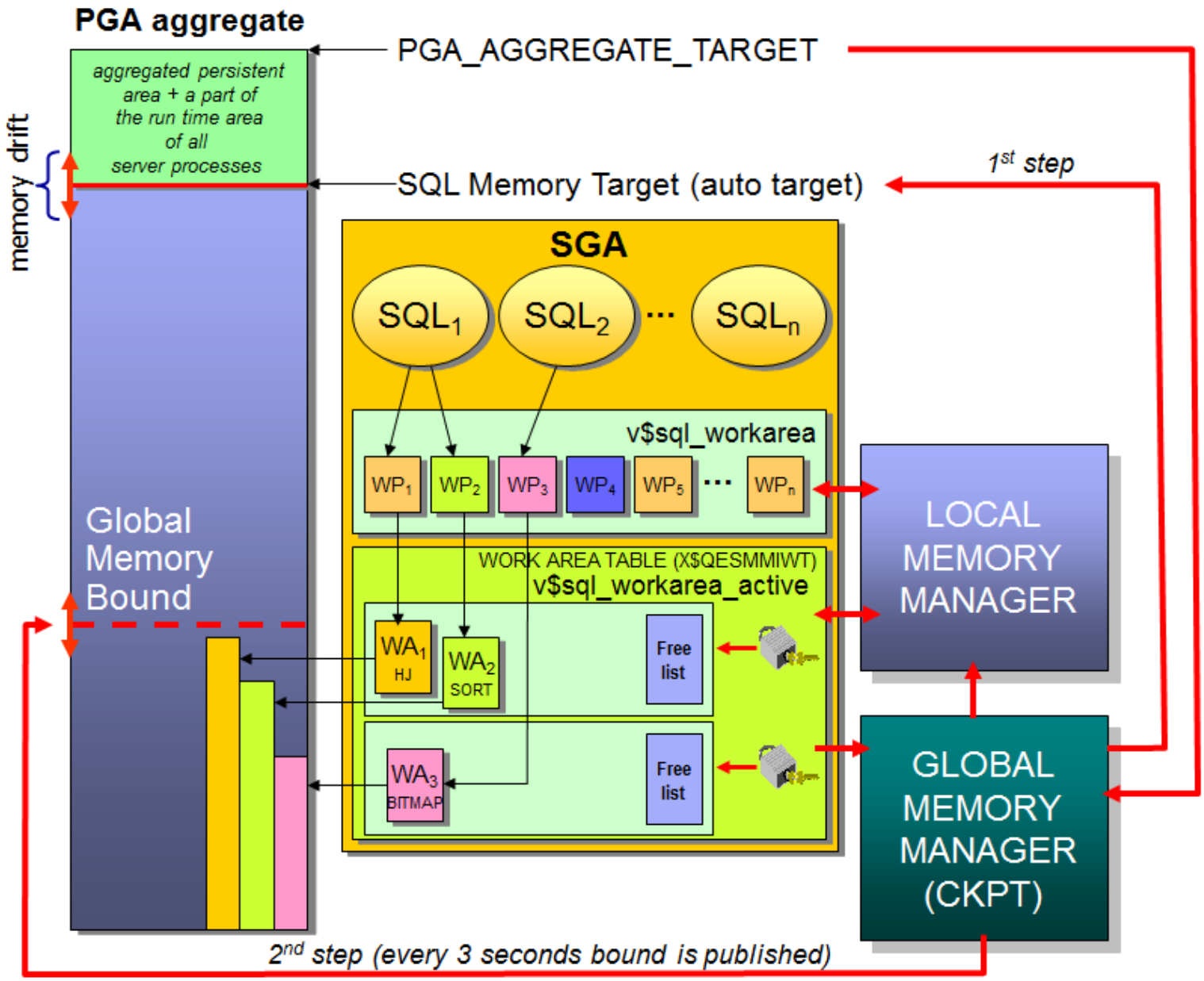
- pmap - on Linux
- procmap – on AIX

```
procmap <pid>

root# procmap 21430470

100000000           202626K  read/exec         oracle
110000070             6728K  read/write        oracle
9fffffff0000000         62K  read/exec         /usr/ccs/bin/usla64
9fffffff000fa7e          0K  read/write        /usr/ccs/bin/usla64
9000000004c6980         92K  read/exec         /usr/lib/libsrc.a[shr_64.o]
9001000a011f3e8         55K  read/write        /usr/lib/libsrc.a[shr_64.o]
90000000049dd00        157K  read/exec         /usr/lib/libcorcfg.a[shr_64.o]
9001000a03d9550         40K  read/write        /usr/lib/libcorcfg.a[shr_64.o]
9000000006be700        780K  read/exec         /usr/lib/liblvm.a[shr_64.o]
9001000a03a1188        221K  read/write        /usr/lib/liblvm.a[shr_64.o]
900000000486800         88K  read/exec         /usr/lib/libcfg.a[shr_64.o]
9001000a02b2028         29K  read/write        /usr/lib/libcfg.a[shr_64.o]
....
....
```

Automatic Memory Management



- Could introduce very turbulent behavior
- In the above case there was no need to increase PGA which was way oversized before
- Go manual after determining what would be reasonable minimum values to prevent ORA-4031 and ORA-4030 errors


PGA aggregate
PGA_AGGREGATE_TARGET
aggregated persistent area + a part of the run time area of all server processes
memory drift
SQL Memory Target (auto target)
1st step
SGA
SQL1
SQL2
SQLn
v$sql_workarea
WP1
WP2
WP3
WP4
WP5
WPn
Global Memory Bound
WORK AREA TABLE (X$QESMMIWT)
v$sql_workarea_active
WA1
HJ
WA2
SORT
WA3
BITMAP
Free list
Free list
LOCAL MEMORY MANAGER
GLOBAL MEMORY MANAGER (CKPT)
2nd step (every 3 seconds bound is published)

ORA-4030 – PGA allocation error

- ORA-4030 could mean:
  - System is out of memory
  - PGA per process reaches per process limit - usually this is 4GB
  - Solution:
    - On Linux vm.max_map_count = 65536 limiting the VM memory map table for the process.
    - Solution – increase memory map table

```
$ sysctl -w vm.max_map_count=262144 (16GB)
```

- Or at database level set hidden parameter higher, for example:
- _realfree_heap_pagesize_hint = 262144 (default 64K)
  -

PGA_AGGREGATE_LIMIT (12c)

- From documentation:
  - By default, PGA_AGGREGATE_LIMIT is set to the greater of 2 GB, 200% of PGA_AGGREGATE_TARGET, and 3 MB times the PROCESSES parameter.
  - It will not exceed 120% of the physical memory size minus the total SGA size.
- Imagine you defined 45,000 processes – the limit is then set to ~131GB
- Be aware that un-controlled memory allocations in PL/SQL by using huge arrays of records or collections (with millions of elements) will cause serious paging.

PGA_AGGREGATE_LIMIT (12c)

- When Exceeded
  - Parallel queries will be treated as a unit.
  - Sessions using the most of **untunable** memory will have their calls aborted.
  - If total PGA memory usage is still over the limit, the sessions using the most **untunable** memory will be terminated.
  - SYS processes and background processes other than job queue processes will not be subjected to limitations
- This should never happen – so configure appropriately.

About Statistics That Is Almost Never Set

Using User-defined Functions in SQLs

- A lot of developers like to use their own PL/SQL functions in SQL Statements.

- Nothing wrong with this if the function is really needed. However, many of them were developed instead of :
  - using simple decode statement
  - joining to lookup table(s)

- Executing functions means that database has to switch from SQL Engine to PL/SQL engine and then again to SQL Engine and .... what is very costly.

CBO's Default Costing For Functions

Selectivity	1% (0.01)
CPU cost	3000
IO cost	0
Network cost	0

Default Selectivity/Cost in PL/SQL

- The default selectivity is expressed in percentage between 0% and 100%.
- The **default cost** is defined as the **CPU, I/O** and the **NETWORK** cost. The latter is currently ignored by the CBO.

```
SQL> ASSOCIATE STATISTICS WITH FUNCTIONS
     MyFunc DEFAULT SELECTIVITY 25,
     DEFAULT COST (333064, 5, 0);
```

- We can define the default cost for functions, packages, types, indexes and indextypes.
- We can use SQL_TRACE and TKPROF to obtain actual numbers for I/O cost and execution time.

Estimating CPU Cost

- Use DBMS_ODCI.ESTIMATE_CPU_UNITS function to convert the CPU time per execution to the approximate number of CPU instructions.
- The returned number is defined in 1000 of instructions.
- To calculate CPU cost for a function that always executes in 0.002 second:

```
SQL> variable a number
SQL> begin :a := round(1000*DBMS_ODCI.ESTIMATE_CPU_UNITS(0.002),0);
     end;
 /

PL/SQL procedure successfully completed.

SQL> print a

         A
----------
    333064
```

Bad Application Design / Coding

Common Problems Seen So Many Times

- Observations shortly after being in production:
  - Some parts of batch process were running at too high degree of parallel sessions (this was not Oracle parallel query).
  - Some parts of code don‘t use bulk mechanism (array processing) – they **do single row inserts** or updates
  - Some of the steps are currently running in non efficient way and are recognizable as low load period.
  - Some IAS (insert as select) statements are using **parallel query for the select part**, but not for the **insert** part and are bottlenecked (PX qref latch)
  - Some of the processing in EOD could use MERGE statements instead of loops with select/update statements

Wait events from AWR report

Event	Wait Class	Waits	Time(s)	Avg Time(ms)	%DB time
CPU time			132,259.67		45.19
enq: TX - row lock contention	Application	116,248	38,181.97	328.45	13.04
resmgr:cpu quantum	Scheduler	7,109,318	25,650.18	3.61	8.76
gc buffer busy acquire	Cluster	3,684,126	18,749.70	5.09	6.41
cell single block physical read	User I/O	15,261,610	7,975.56	0.52	2.72
gc buffer busy release	Cluster	207,125	6,673.66	32.22	2.28
direct path read	User I/O	780,851	4,607.71	5.90	1.57
gc cr block busy	Cluster	1,388,690	4,302.12	3.10	1.47
gc cr block 2-way	Cluster	8,207,445	4,057.42	0.49	1.39
gc current block 2-way	Cluster	10,930,375	3,877.89	0.35	1.32
-gc current block 3-way	Cluster	6,924,819	3,558.07	0.51	1.22
-cell smart table scan	User I/O	1,001,315	3,060.46	3.06	1.05
-PX qref latch	Other	108,729,954	1,893.00	0.02	0.65
-db file parallel write	System I/O	2,494,129	1,443.46	0.58	0.49

Observations from AWR report

- enq: TX - row lock contention
  - Exagerated due to too high concurrency in processing
- resmgr:cpu quantum
  - Resource manager kicked in and limited CPU usage
- gc buffer busy acquire and gc buffer busy release
  - Blocks are held too long due to high concurrency
- PX qref latch
  - Non-balanced statements with parallel execution – parts of statements are executed with serial execution

After Reduced Parallel Sessions

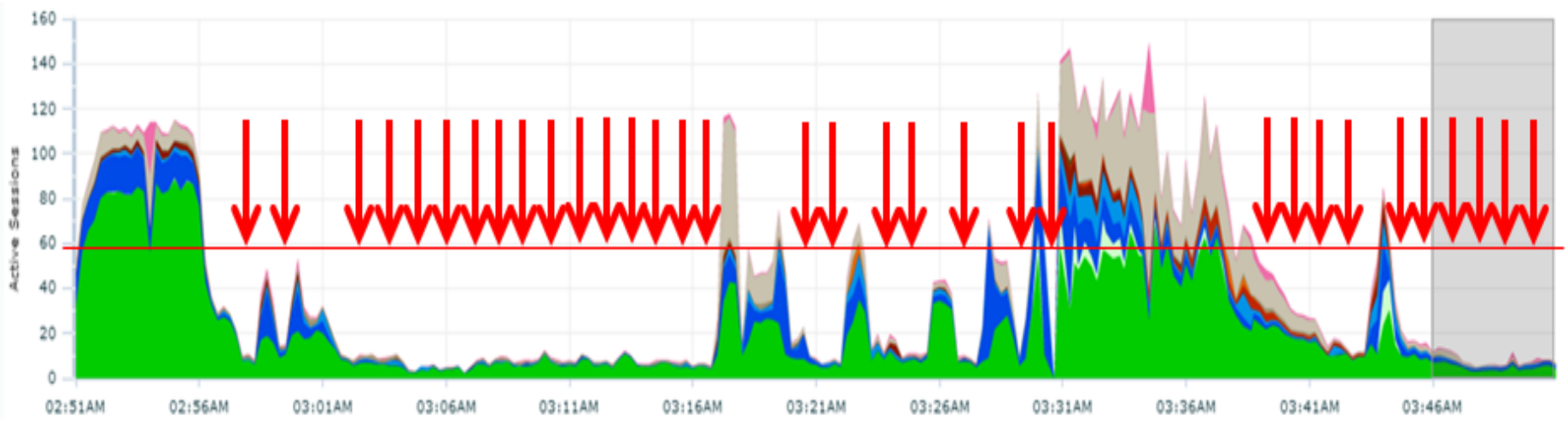
1st						2nd					
Event	Wait Class	Waits	Time(s)	Avg Time(ms)	%DB time	Event	Wait Class	Waits	Time(s)	Avg Time(ms)	%DB time
CPU time			132,259.67		45.19	CPU time			206,094.07		61.53
enq: TX - row lock contention	Application	116,248	38,181.97	328.45	13.04	cell single block physical read	User I/O	46,513,199	24,418.49	0.52	7.29
resmgr:cpu quantum	Scheduler	7,109,318	25,650.18	3.61	8.76	PX qref latch	Other	235,824,398	11,761.52	0.05	3.51
gc buffer busy acquire	Cluster	3,684,126	18,749.70	5.09	6.41	gc current block 2-way	Cluster	40,436,748	10,154.22	0.25	3.03
cell single block physical read	User I/O	15,261,610	7,975.56	0.52	2.72	gc current block 3-way	Cluster	26,130,317	9,341.10	0.36	2.79
gc buffer busy release	Cluster	207,125	6,673.66	32.22	2.28	db file parallel write	System I/O	5,807,283	6,893.70	1.19	2.06
direct path read	User I/O	780,851	4,607.71	5.90	1.57	cell smart table scan	User I/O	1,010,184	5,699.47	5.64	1.70
gc cr block busy	Cluster	1,388,690	4,302.12	3.10	1.47	gc cr block busy	Cluster	2,583,140	5,076.38	1.97	1.52
gc cr block 2-way	Cluster	8,207,445	4,057.42	0.49	1.39	gc cr block 2-way	Cluster	19,541,685	4,669.58	0.24	1.39
gc current block 2-way	Cluster	10,930,375	3,877.89	0.35	1.32	gc buffer busy acquire	Cluster	3,559,628	4,565.44	1.28	1.36
-gc current block 3-way	Cluster	6,924,819	3,558.07	0.51	1.22	-resmgr:cpu quantum	Scheduler	1,897,001	2,788.40	1.47	0.83
-cell smart table scan	User I/O	1,001,315	3,060.46	3.06	1.05	-gc buffer busy release	Cluster	373,414	2,227.63	5.97	0.67
-PX qref latch	Other	108,729,954	1,893.00	0.02	0.65	-enq: TX - row lock contention	Application	3,537	1,724.24	487.49	0.51
-db file parallel write	System I/O	2,494,129	1,443.46	0.58	0.49	-direct path read	User I/O	9,142	22.24	2.43	0.01

- **Amount of data between these two runs doubled!!!!**
- Number of sessions processing accounts in batches of 10,000 reduced from ~300 to 64
- CPU utilization increased / more work done
- Parallel execution still not balanced
- Significantly reduced cluster contention and TX-row lock contention

Problematic Application Design

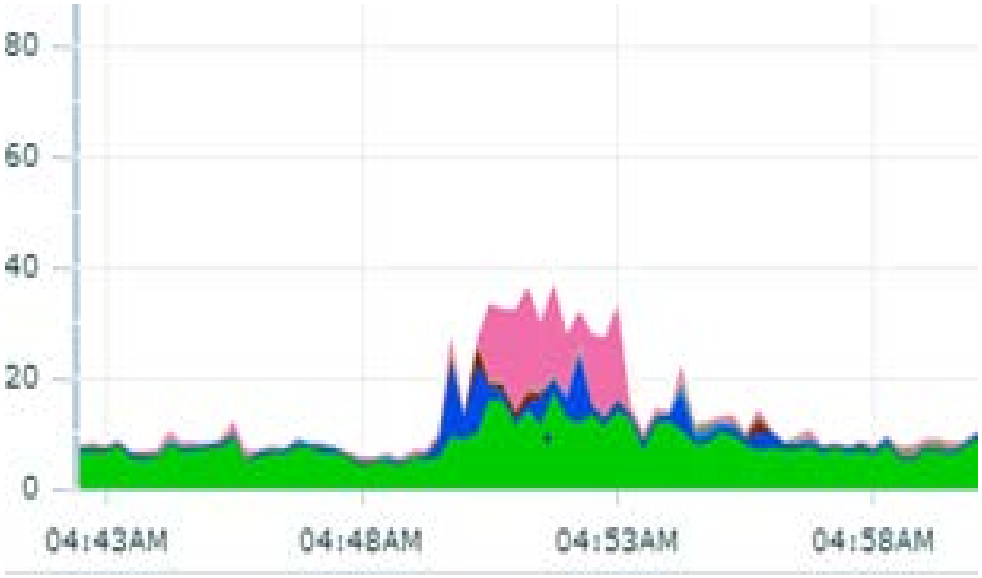
- Problematic batch process
- Processing of millions of accounts looks like:
  - Each session takes 10000 accounts and processes one by one in a loop
  - Run in parallel in „n“ sessions until all cards are processed
- As cards are processed one by one, index access is used for retrieving data (Exadata smart scan and other features practically not used)
- 20 milliseconds of CPU time used for processing one card account
- With available CPUs (without any wait time) SLA defined time for batch completion was not reachable.

Unbalanced Batch Processing



- Unbalanced batch processing – missed batch processing window
  - Periods of excessive activity experiencing contention
  - Periods of low activity not using the HW potential

Unbalanced Parallel Execution



- PX qref latch (pink color) wait

```
INSERT /*+ APPEND */ INTO … SELECT /*+ PARALELL(32) */ …

INSERT INTO … SELECT /*+ PARALELL(32) */ ….
```

Proposed Solutions

- Critical parts of application which are using parallel sessions for batch processing should be changed:
  - Instead of processing accounts one by one (10,000 per batch in one session) probably all accounts should be processed at once
    - Using Oracle parallel execution
    - Using array processing (bulk operations)
  - Improved processing can run in parallel on different nodes (using data segmentation) in order to reduce cluster contention.

“The Solution”

- Application Entry Form
- Table was constantly growing
- Whenever form was opened the below query was run:

```
select * from xxx
where check_if_record_should_be_shown( p1, p2 ) = 1
```

Bad Database/Application Design

- No constraints set besides PK and NOT NULL.
  - It is running faster (Really ???)
- NVARCHAR2 used for PK (2-byte chars – problems with histograms)
- Poor design of PL/SQL code – PL/SQL used when normal SQL would perform much better
- No special code written for operations repeated many times – just run the same code as many times as you need
- Write own PL/SQL functions when built-in PL/SQL functions would be sufficient
- Not using PL/SQL result cache.
- Store frequently used variables which don‘t change in PL/SQL variables and read them with SQL only once
- Don‘t use too big arrays or collections – beyond 200-300 elements will not improve performance

Conclusions

- Buy enough memory for your needs – it is not licensed.
- Processing in parallel sessions introduces contention
  - Massive batch processing does not mean just executing the same procedure million times!
  - Consider rewriting the code.
- Strive always for performance
  - Use CTAS or IAS way of doing – 12c gathers initial statistics for this
  - Always use bulk processing
  - Balance parallel execution and carefully plan degree of parallelism
- Write the code and test the performance, but not on „empty“ tables – use DBMS_RANDOM to populate tables

Thank you for your interest!

Q&A