Performance and Scalability Improvements in Oracle 10g and 11g

Tanel Põder

http://www.tanelpoder.com

ORACLE | CERTIFIED MASTER

Introduction

- About me:
  - Occupation: Consultant, researcher
  - Expertise: Oracle internals geek, *End-to-end* performance & scalability
  - Oracle experience: 10 years as DBA
  - Certification: OCM (2002) OCP (1999)
  - Professional affiliations: OakTable Network
- About the presentation:
  - This presentation is about low level kernel infrastructure, like redo/undo generation, memory management etc

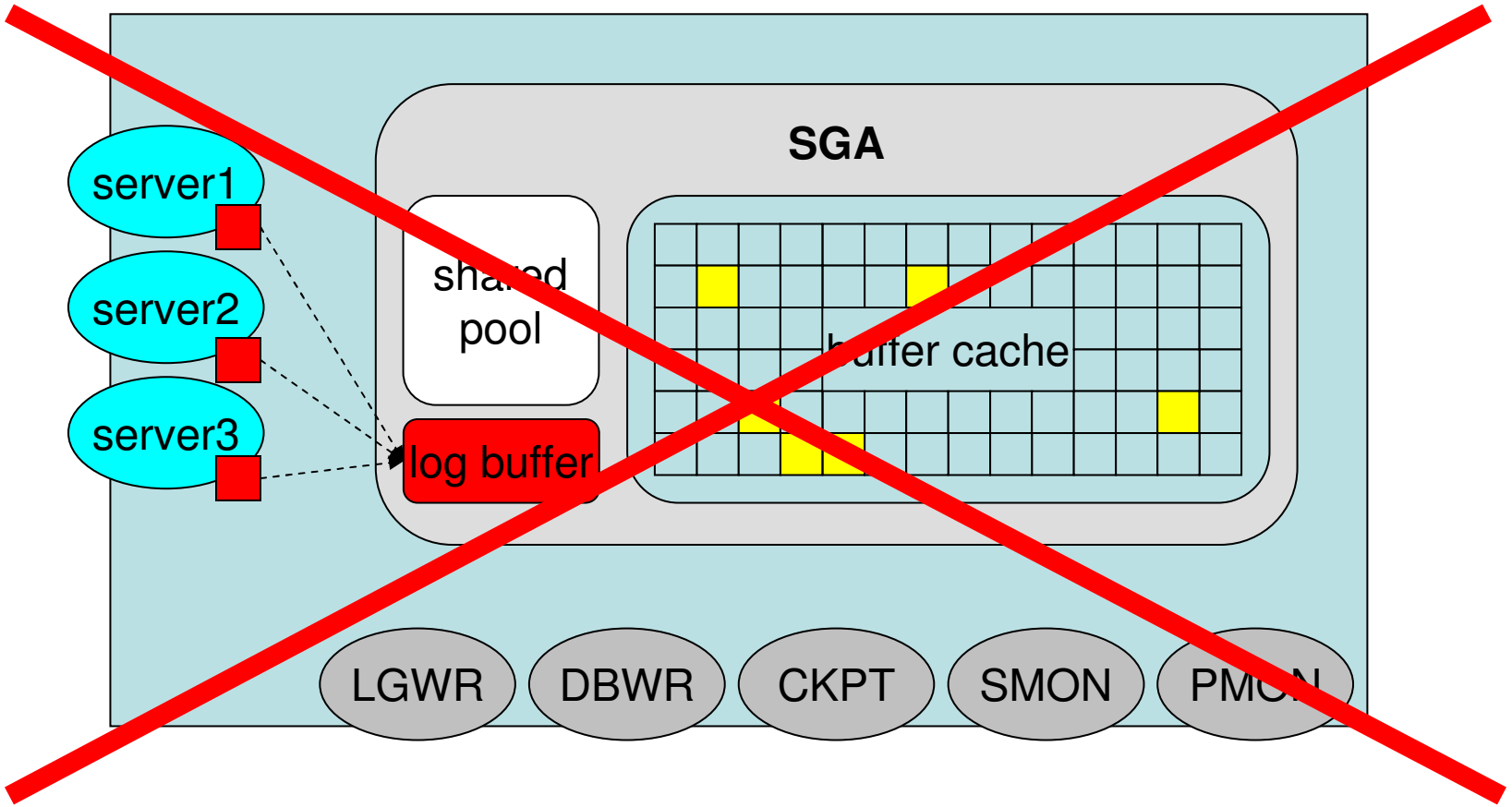
What are our CPUs really doing?

- Example: Update a row in a table
  - May need to update data block, index blocks, undo blocks, freelist or ASSM block, generate redo, pin structures, etc..
  - Each of those are complex data structures
  - Looked up via hash tables, arrays, linked lists, indexes
  - A LOT of pointer chasing within SGA
  - The “targets” for pointers might not be in CPU cache
- CPU Cache = FAST (roughly 2..20 CPU cycles)
- RAM = SLOOOW (hundreds of CPU cycles)
  - CPU’s STALL when waiting for RAM
  - Yes, CPU *service* time also includes *wait* time
  - In pointer-chasing scenarios CPUs can’t really *prefetch* nor do much *speculative execution*

Public redolog strands (_log_parallelism)

- Few shared redo log buffers
  - Yes, multiple shared redolog buffers available since 9i
- Each buffer protected by a redo allocation latch
  - These latches serialize the log buffer space preallocation
  - First, *any* redo copy latch must be taken
  - Redo copy latches help to ensure that noone is writing to log buffer while LGWR flushes redo to disk
  - When space is allocated, redo allocation latch is released
  - Then create redo change vectors in PGA
- Copy redo change vectors to log buffer
- Apply redo vectors to buffers in buffer cache
  - The same function is used for applying redo during recovery

Oracle Instance 101



Redo data

Undo data

Buffer cache

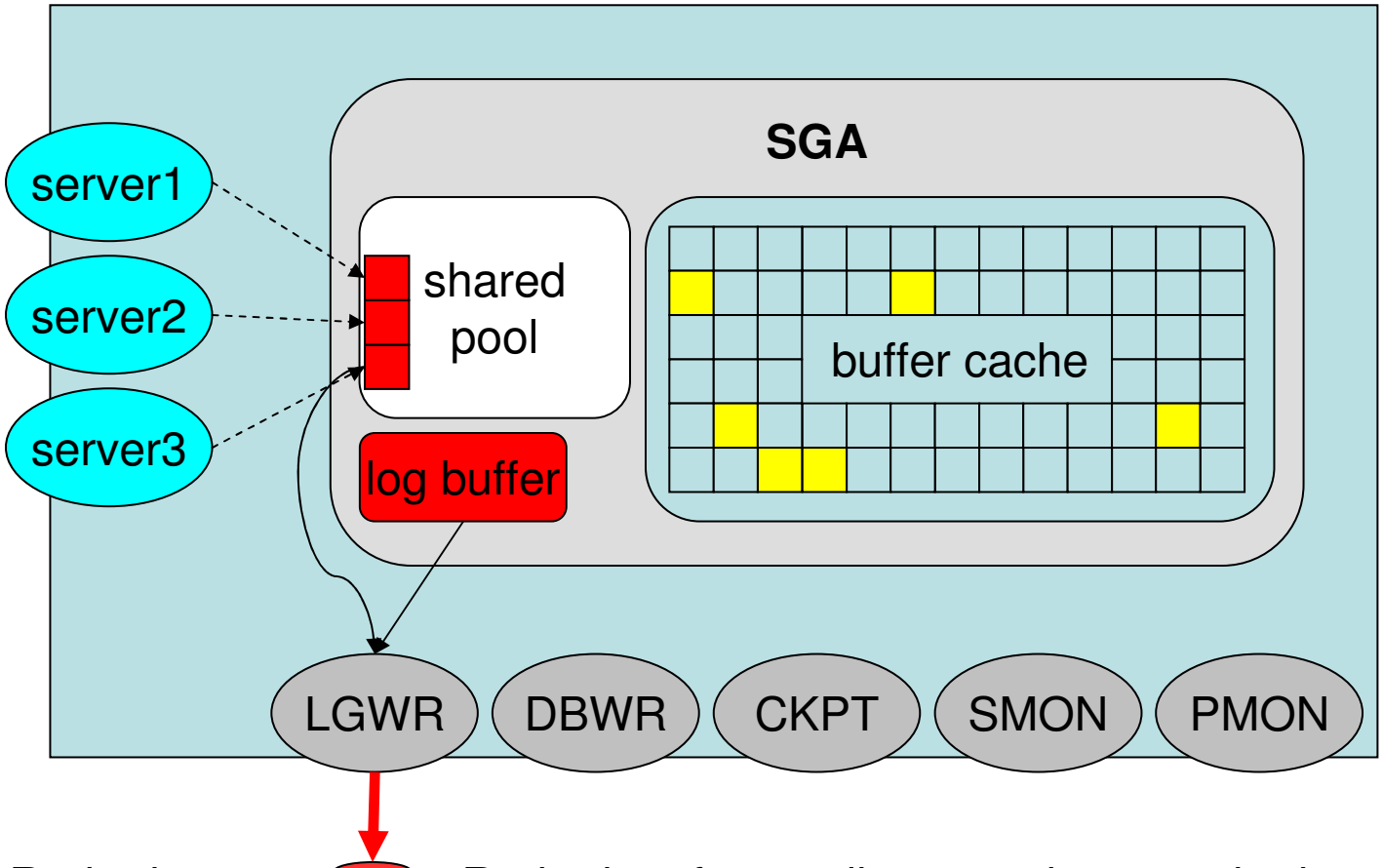
Shared pool

WRONG!

Private redolog strands

- Lots of private redolog buffers
  - Allocated from shared pool ('private strands' in v$sgastat)
  - Normally 64-128kB in size, each
  - Each protected by a separate redo allocation latch
  - New transaction is *bound* to a free private redo buffer
  - One buffer for one active transaction
- Redo generated directly to private buffer – not PGA
  - No extra memory copy operation needed
  - No need for redo copy latches (which were originally designed for relieving redo allocation latch contention)
  - On redo flush, all public redo allocation latches are taken
  - All redo copy latches for public strands are checked
  - And all private ones for *active* transactions are taken

Private redolog strands in Oracle instance



Redo data

Undo data

Buffer cache

Shared pool

Redo data for small transactions can be kept in preallocated memory locations in shared pool.

Less latching overhead, virtually no redo allocation latch collisions.

Private redolog strands

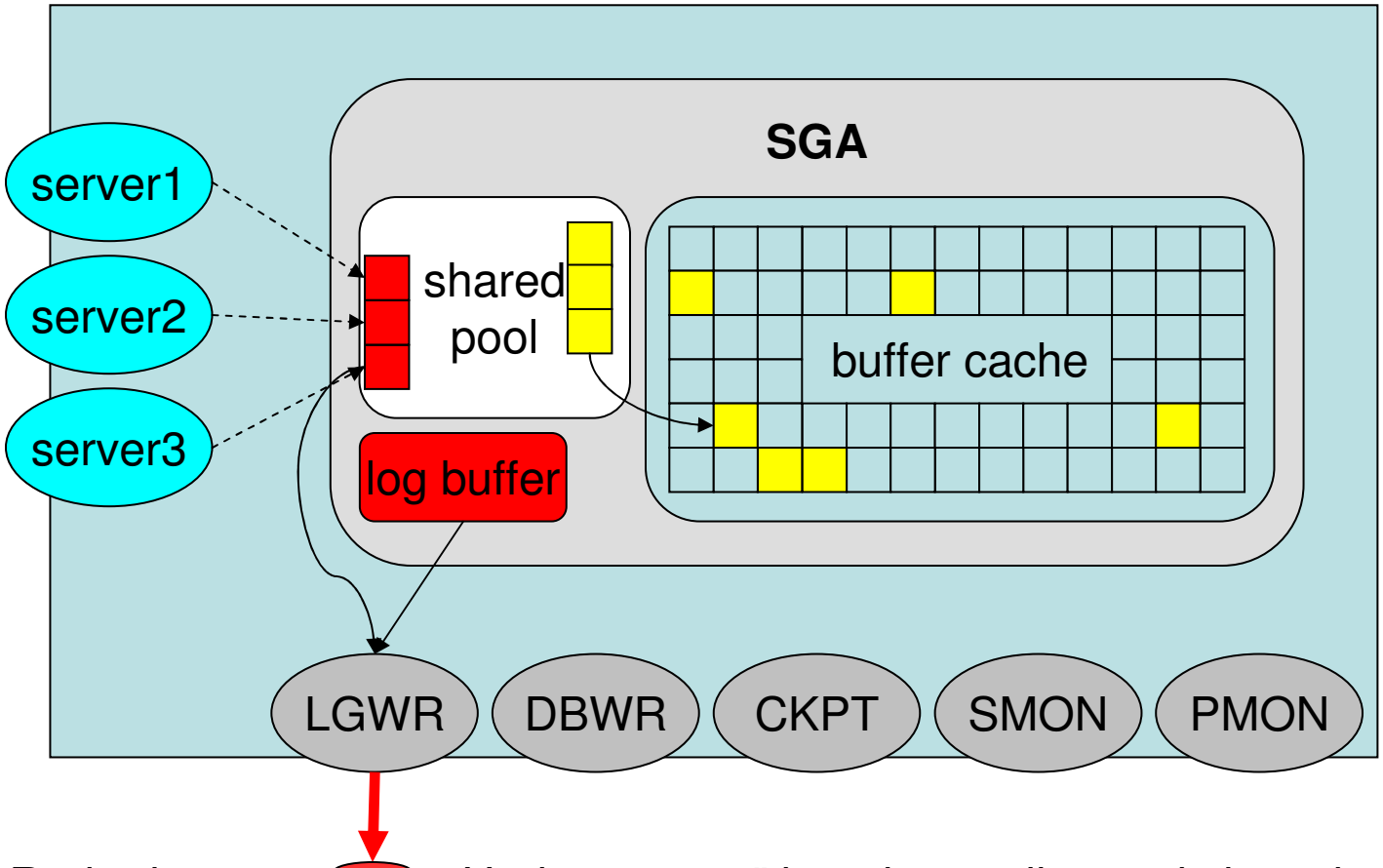
- Listing redolog strands
  - X$KCRFSTRAND
  - Includes both public and private strands
  - rs.sql (redo strand.sql)

```
select INDX,
  PNEXT_BUF_KCRFA_CLN nxtbufadr,
  NEXT_BUF_NUM_KCRFA_CLN nxtbuf#,
  BYTES_IN_BUF_KCRFA_CLN "B/buf",
  PVT_STRAND_STATE_KCRFA_CLN state,
  STRAND_NUM_ORDINAL_KCRFA_CLN strand#,
  PTR_KCRF_PVT_STRAND stradr,
  INDEX_KCRF_PVT_STRAND stridx,
  SPACE_KCRF_PVT_STRAND strspc,
  TXN_KCRF_PVT_STRAND txn,
  TOTAL_BUFS_KCRFA totbufs#,
  STRAND_SIZE_KCRFA strsz
from X$KCRFSTRAND
```

Private undo buffers (in-memory undo)

- In-memory undo (IMU) is tighly integrated with private redo strands
- IMU buffers are also allocated from shared pool
  - Called IMU pool
  - Around 64-128kB in size, each
- A new transaction is *bound* to a free IMU buffer
  - Acts as low cost undo buffer (no immediate datablock modifications needed)
  - Redo for undo data is generated into private redo strand
- Each IMU buffer protected by a separate latch
  - “In memory undo latch”
- IMU Flush happens if either IMU buffer or private redolog strand gets full (and for other reasons)

Private redolog strands in Oracle instance



Redo data

Undo data

Buffer cache

Shared pool

Undo “vectors” kept in preallocated shared pool locations.

Less latching overhead, can write to undo segments in batches on IMU flush.

Private undo buffers (in-memory undo)

```
SQL> select name, value from v$sysstat where name like 'IMU%';

NAME                                                  VALUE
---------------------------------------- ----------
IMU commits                                             890
IMU Flushes                                              92
IMU contention                                            0
IMU recursive-transaction flush                           0
IMU undo retention flush                                  0
IMU ktichg flush                                          0
IMU bind flushes                                          0
IMU mbu flush                                             0
IMU pool not allocated                                    0
IMU CR rollbacks                                         60
IMU undo allocation size                            6209336
IMU Redo allocation size                            1012524
IMU- failed to get a private strand                       0
```

Private undo buffers (in-memory undo)

- Listing in-use IMU buffers
  - Includes private redo strand usage info
  - X$KTIFP
  - im.sql

```
select ADDR,KTIFPNO, KTIFPSTA, KTIFPXCB xctaddr,
   to_number(KTIFPUPE, 'XXXXXXXXXXXXXXXX')-
   to_number(KTIFPUPB, 'XXXXXXXXXXXXXXXX') ubsize,
   (to_number(KTIFPUPB, 'XXXXXXXXXXXXXXXX')-
   to_number(KTIFPUPC, 'XXXXXXXXXXXXXXXX'))*-1 ubusage,
   to_number(KTIFPRPE, 'XXXXXXXXXXXXXXXX')-
   to_number(KTIFPRPB, 'XXXXXXXXXXXXXXXX') rbsize,
   (to_number(KTIFPRPB, 'XXXXXXXXXXXXXXXX')-
   to_number(KTIFPRPC, 'XXXXXXXXXXXXXXXX'))*-1 rbusage,
KTIFPPSI,KTIFPRBS,KTIFPTCN
from x$ktifp
where KTIFPXCB != hextoraw('00')
```

Even more shared pool subpools

- 9i introduced splitting shared pool into up to 7 pools
  - Meaning 7 subheaps, each having own LRU and free lists
  - Controlled by the _kghdsidx_count parameter
  - Each pool protected by separate shared pool latch
  - Library cache latch directory determined which objects belonged to which pool (which latch should be taken)
- In 10g R2 the number of heaps can be even larger
  - Still maximum 7 shared pool latches though
  - "Under" each latch there are 4 heaps for different allocation lifetimes (durations)
- select * from x$kghlu where kghlushrpool = 1;
  - Different *kghluidx* means protection by different latch
  - Different *kghludur* = same latch, different sub-sub-heap

Lightweight library cache mutexes (10gR2)

- Max 67 library cache latches available
  - Latches are assigned to KGL objects based on their *KGLNA* hash value (SQL_TEXT, object_name, etc)
- Collisions are inevitable, even with only 100 cursors
  - Child cursors of the same parent will always collide
- Library cache mutexes greatly relieve this problem
  - Each library child cursor has it's own mutex, no collisions
  - Small structure right in the child cursor handle
  - Used if _kks_use_mutex_pin = true (default in 10.2.0.2)
  - Are used for protecting other stuff as well (V$SQLSTATS)
  - Oracle mutexes have nothing to do with OS mutexes
  - Mutexes are just some memory locations in SGA

Lightweight library cache mutexes (10gR2)

- No separate PIN structure needed
  - Mutex itself acts as library cache object pin structure
  - Less pointer-chasing
  - If “non-zero” then corresponding KGL object is pinned
  - Helps only properly designed applications (no parsing)
  - …and apps using session_cached_cursors!
  - Eliminates the need for cursor_space_for_time
- No GET/MISS statistics maintained
  - Requires fewer CPU instructions for getting the mutex
  - SLEEP statistics are maintained, however
  - V$MUTEX_SLEEP
  - V$MUTEX_SLEEP_HISTORY

Incremental library cache cursor invalidation

- 10gR2 feature
- dbms_stats.gather_table_stats(…

  no_invalidate=>DBMS_STATS.AUTO_INVALIDATE

)
  - This is the default in 10.2.0.2
- _optimizer_invalidation_period
  - Time in seconds after which a new child cursor is created *if* such statement is parsed again
  - Default 18000 seconds, 5 hours
  - no_invalidate => FALSE causes immediate invalidation
  - no_invalidate => TRUE causes no invalidation at all
  - Less impact to library cache/shared pool on optimizer stats gathering – reduces the reparsing spike

Predicate selectivity-based conditionial parsing

- 11g new feature
- Measures the real *rows processed* statistic for bind predicates and stores it shared pool
  - Bind peeking is done for each execution
- If the *rows processed* counter for the cursor varies greatly over different executions, then:
  - Parent cursor is made bind aware and stored along its predicate rowcount statistics
  - I observed max 3 child cursors per bind variable (rowcounts on scale of 1…999, 1000-1M, 1M…)
  - On next execution, if hitting a bind aware cursor, the peeked bind and table statistics from dictionary are used for determining if an existing child cursor can be reused

Predicate selectivity-based conditionial parsing

- What does this mean for us?
  - SQL execution plans used will be even more dynamic
  - The same statement executed twice can change execution plans

- Some evidence already in 10gR2 oraus.msg file:

```
10507, 00000, "Trace bind equivalence logic"
// *Cause:
// *Action:
```

- In earlier versions you could use DBMS_RLS for it…
  - Which has scalability considerations though
  - See my presentation on Tuesday for details

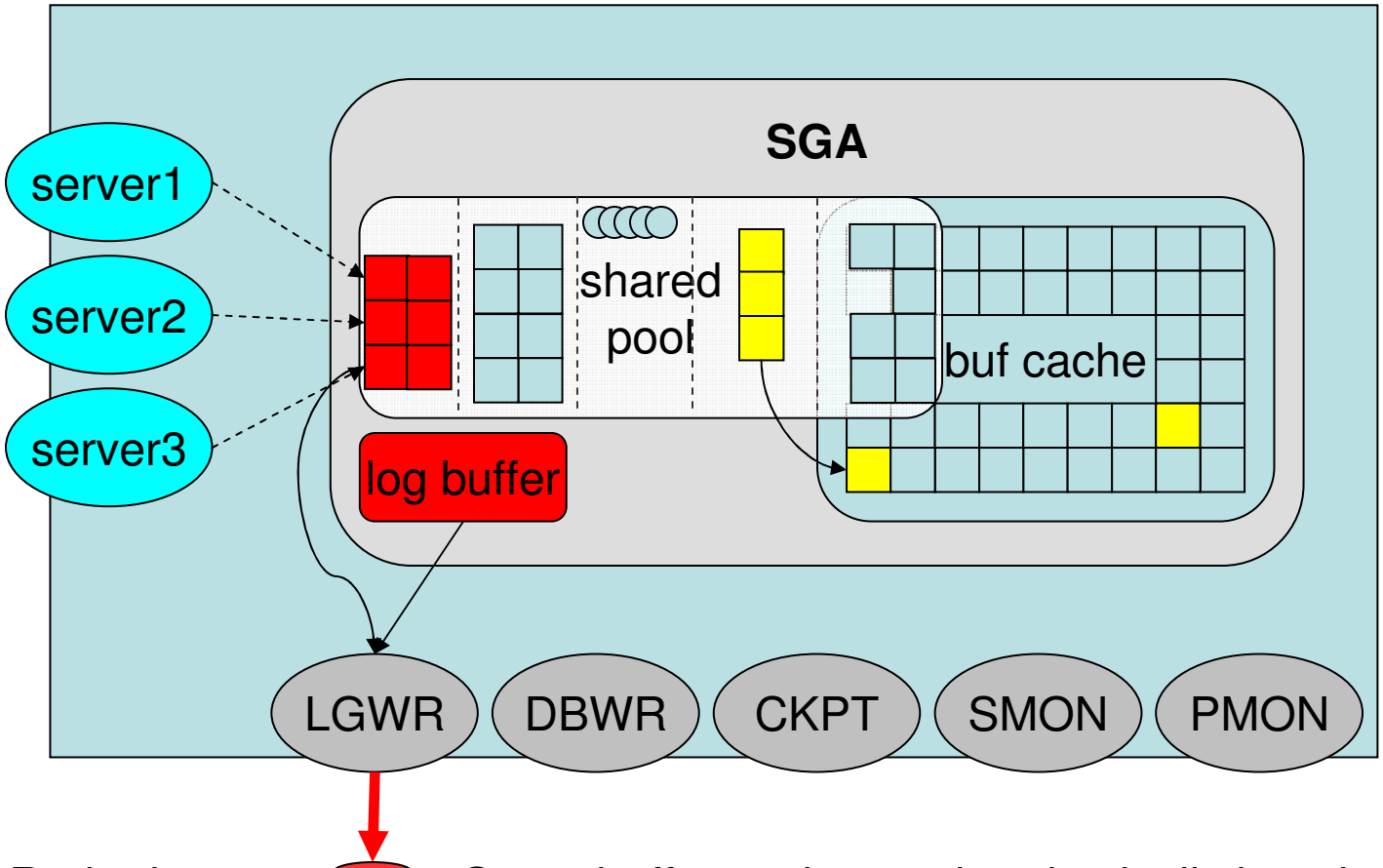
11g Query Result Cache

- Avoid LIOs from buffer cache by caching query results in shared pool
  - Shared pool - ResultCache statistic in V$SGASTAT
- How to use:
  - /*+ RESULT_CACHE */ hint
  - RESULT_CACHE_MODE=<force,manual,auto>
  - RESULT_CACHE_SIZE = …
- How to monitor:
  - V$RESULT_CACHE_DEPENDENCY
  - V$RESULT_CACHE_MEMORY
  - V$RESULT_CACHE_STATISTICS

Buffer cache within shared pool!

- Virtues of Automatic SGA Memory Management
  - When SGA_TARGET is set…
  - And buffer cache needs to expand…
  - If shared pool has to be reduced for making space…
  - No easy way to relocate in-use (pinned) memory chunks
- Easier to leave the granule into “partially transformed” mode
  - The chunks which were releasable, are marked as KGH: NO ACCESS in shared pool heap…
  - …and are used for buffer cache buffers
  - The granule is concurrently used by shared pool and buffer cache

The modern SGA



Redo data

Undo data

Buffer cache

Shared pool

Some buffer cache maybe physically kept in KGH: NO ACCESS permanent shared pool chunks.

Shared pool heap manager knows not to touch these chunks, as buffer cache is using them

Thank you!

Tanel Põder

http://www.tanelpoder.com

ORACLE | CERTIFIED MASTER