Inside the BUFFER CACHE

Yavor Ivanov

whoami

- 11 years experience with Oracle Database
- Oracle Database & Oracle RAC fanatic
- DBA at Moneybookers
- Oracle Certified Master
  - OCP DBA 9i/10g
  - OCE RAC
  - OCE SQL

Discalimer

The algorithms discussed in this presentation are not documented since Version 7/8 *(see MOS note 91062.1 dated May 18,1998)*. They changed several times form Oracle 7/8 to Oracle 11g and they will change in the future. Even more, since they are not documented, I cannot guarantee that they work the way I show you. So, use this presentation with caution.

A simple query

```
update EMP
   set SALARY=SALARY*2
 where FIRST_NAME='Yavor';
```

Steps we skip in this presentation

- Parsing the query; building a plan
- Data dictionary lookups
- Locking
- Transaction handling
- Consistent reads / Undo
- RAC inter-instance communication

I will talks only about the path of the data through the memory

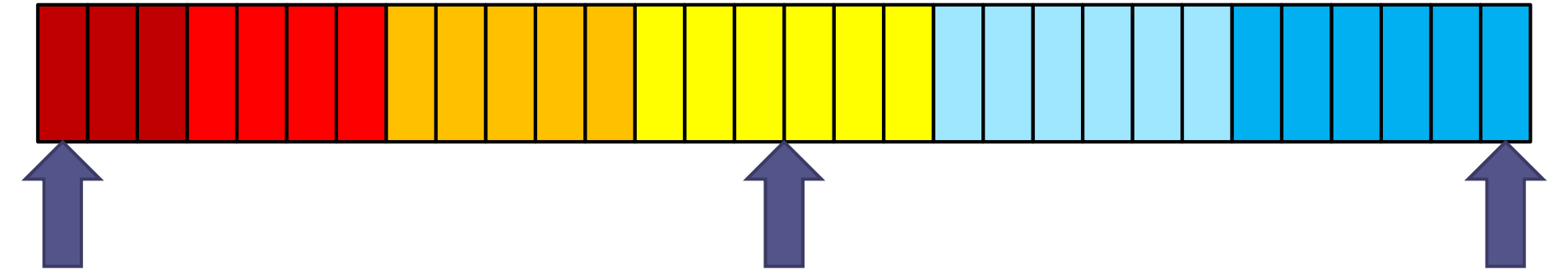
Steps discussed in this presentation

- Reading data from disk to memory
- Writing data from memory to disk

Basics

- When a session needs some data – be it for reading (`select`) or writing (`DML`) – the block, containing the data **should be** in the buffer cache*
  - If the block is already in the buffer cache, the session does not need to read it from disk – we have a *buffer cache hit* (Logical read, LIO)
  - If the block is not in the buffer cache, then it is read from the disk **by the session's server process** and put in the buffer cache (Physical read, PIO)
- The server process **never writes** to disk*
  The writing is done by LGWR and DBWR

The LRU discipline



- When a block arrives, it is put in the **midpoint***
- **Basically** when a block is touched it is moved to the head of the list
- The block **naturally** moves from the hot region into the cold region

- More details at http://www.bgoug.org/bg/events/details/74.html
  Oracle Touch Count Algorithm, Eduard Hajrabedian

Buffer cache terms

Buffers

- A **buffer** is a place in memory, which is as big as a single database block - no more, no less. One buffer is used to store the contents of a data block after it is read form disk

Buffer header

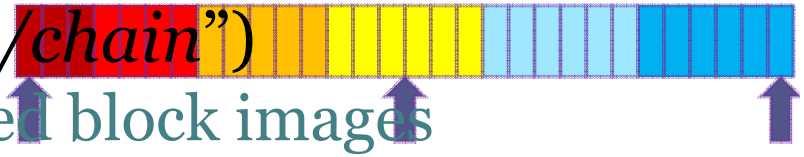
- For every buffer, there is a **buffer header** (only in memory)
- Buffer headers are used to organize the buffers
- Buffer headers contain metadata about the block/buffer:
  - the DBA of the block
  - the type of the block (e.g. data block, undo block, etc.)
  - the touch count
  - pointer from the header to the real buffer in memory
  - pointers to the prev and next buffer (header) in the cache buffer chain
  - pointers to the prev and next buffer (header) in the lists this buffer belongs to
  - etc.

Buffer statuses

- **Clean** – the data in the buffer is not modified, so it can be overwritten if needed
- **Dirty** – the data in the buffer is modified, it has to be written to disk before it can be resued
- **Pinned** – the buffer is being read/written by some process

Double-linked lists (queues) in buffer cache

- REPL-AUX list
  - Contains empty/clean buffers, ready for use
- REPL list (a.k.a. "*The LRU list/chain*")
  - Contains clean, dirty and pinned block images

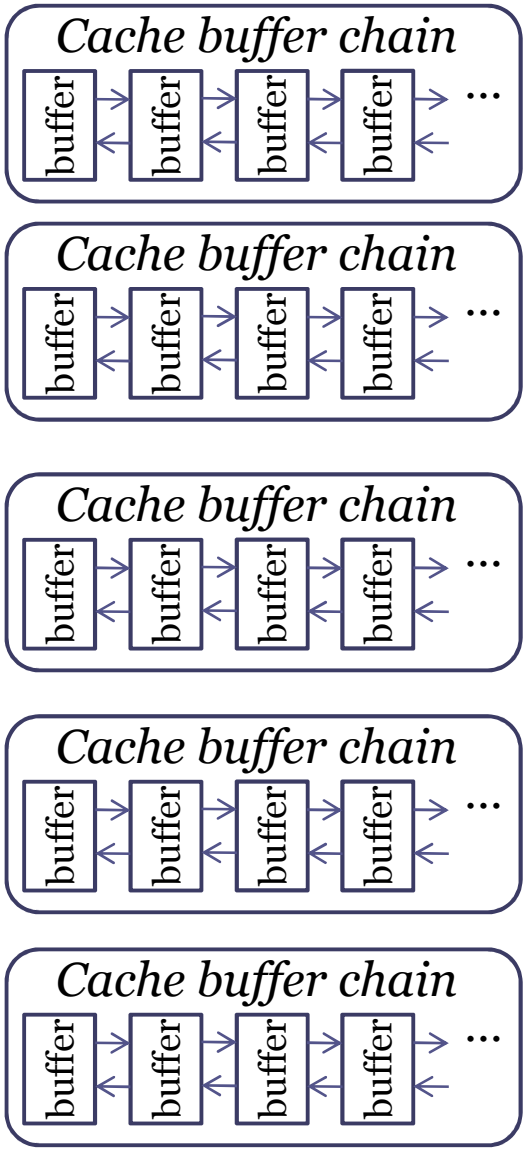


- CKPT-Q list
  - When a block is changed (dirtied), it is attached here
- WRITE list
  - Contains dirty buffers, which should be written by DBWR
- WRITE-AUX list
  - Contains blocks which are currently in process of writing to disk

Cache buffer chains (aka *Hash buckets*)

- Looking for a specific block by scanning through the whole buffer cache would be extremely slow
- The buffer cache is divided to small linked lists (buckets) of buffers – cache buffer chains (CBC)
  - Every block's buffers belongs only to one bucket
  - The bucket is found through hashing the block's DBA
- When a process want to scan for a block, it has to scan only through the relatively short chain (typically 1 buffer)

```
_db_block_hash_buckets Number of
      database block hash buckets
```



The value of _db_block_hash_buckets (speculation)

- In Oracle 7/8
  the next prime after **(db_block_buffers/4)** -
  documented in Metalink
- In Oracle 9.2
  the next prime after **(db_block_buffers*2)** -
  observed on HP-UX and IBM AIX
- In Oracle 10.2, 11.1, 11.2
  the next power of 2 after (**db_block_buffers*2)** -
  observed on UP-UX, IBM AIX and Linix x86-64

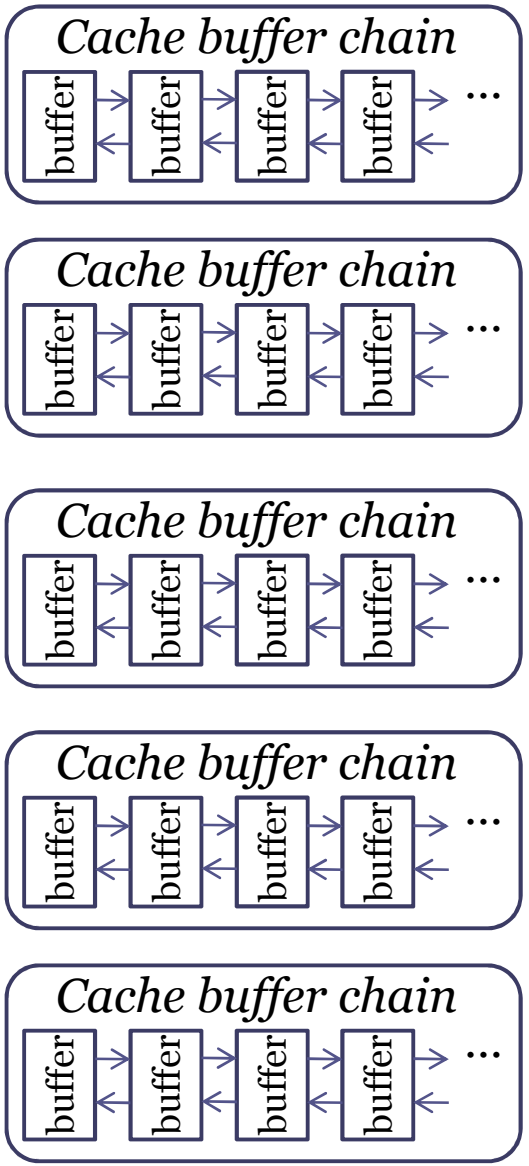
Reading a block

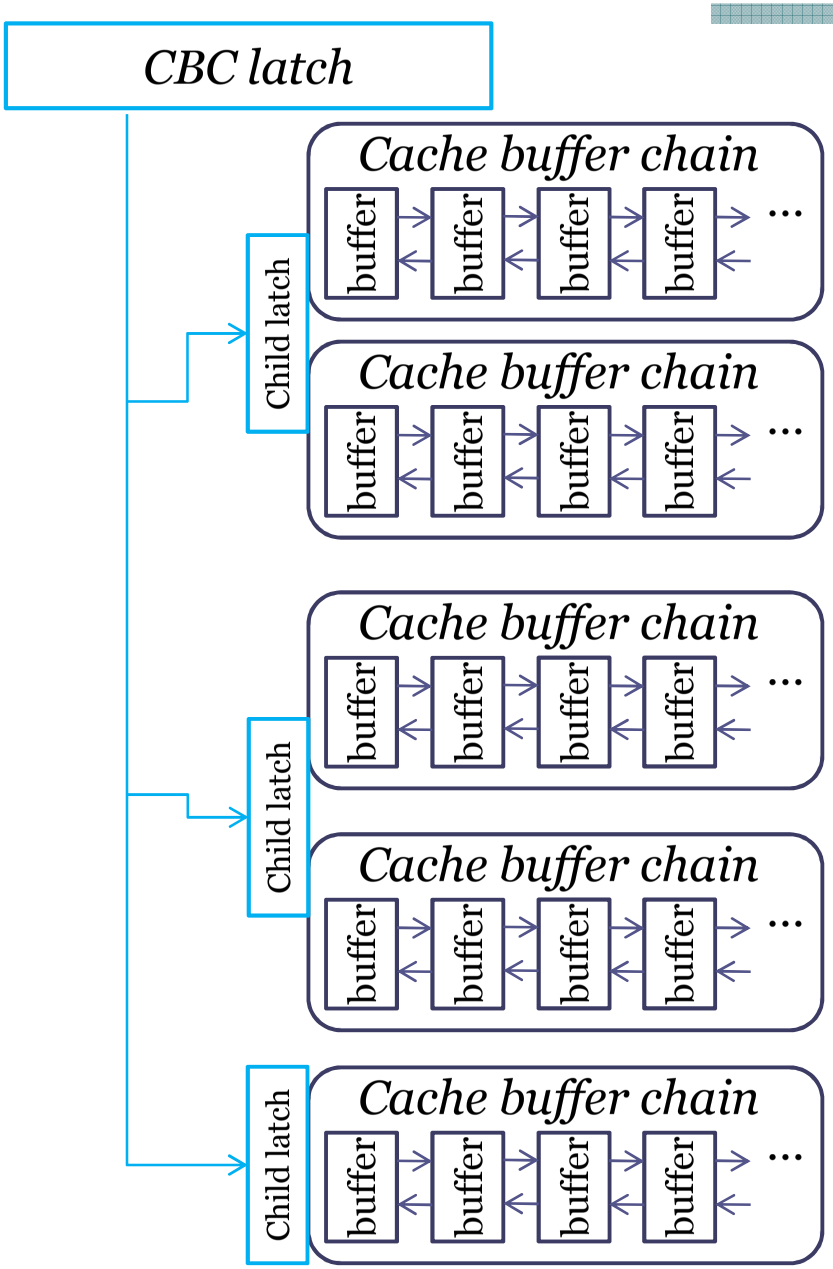
Case 1: LIO

- **The DBA of the block is hashed**. This hash tells us which hash bucket (CBC) will probably have our block
- Our process obtains **the CBC latch** relevant for the specific bucket; scans the bucket for a buffer with the given DBA and SCN; then releases the latch
  Possible contention: *cache buffers chains* latch
- When the block is found, it is pinned and used. Eventually, the **touch count** of the block is increased (no latching protects this)

  This is called LIO

```
_db_block_hash_latches Number of
  database block hash latches
```




CBC latch
Child latch
Child latch
Child latch
Cache buffer chain
Cache buffer chain
Cache buffer chain
Cache buffer chain
Cache buffer chain
buffer
buffer
buffer
buffer
...

The value of _db_block_hash_latches (speculation)

- In Oracle 10.2.0.1, 10.2.0.2
  _db_block_hash_buckets / 256
  - observed on HP-UX and IBM AIX

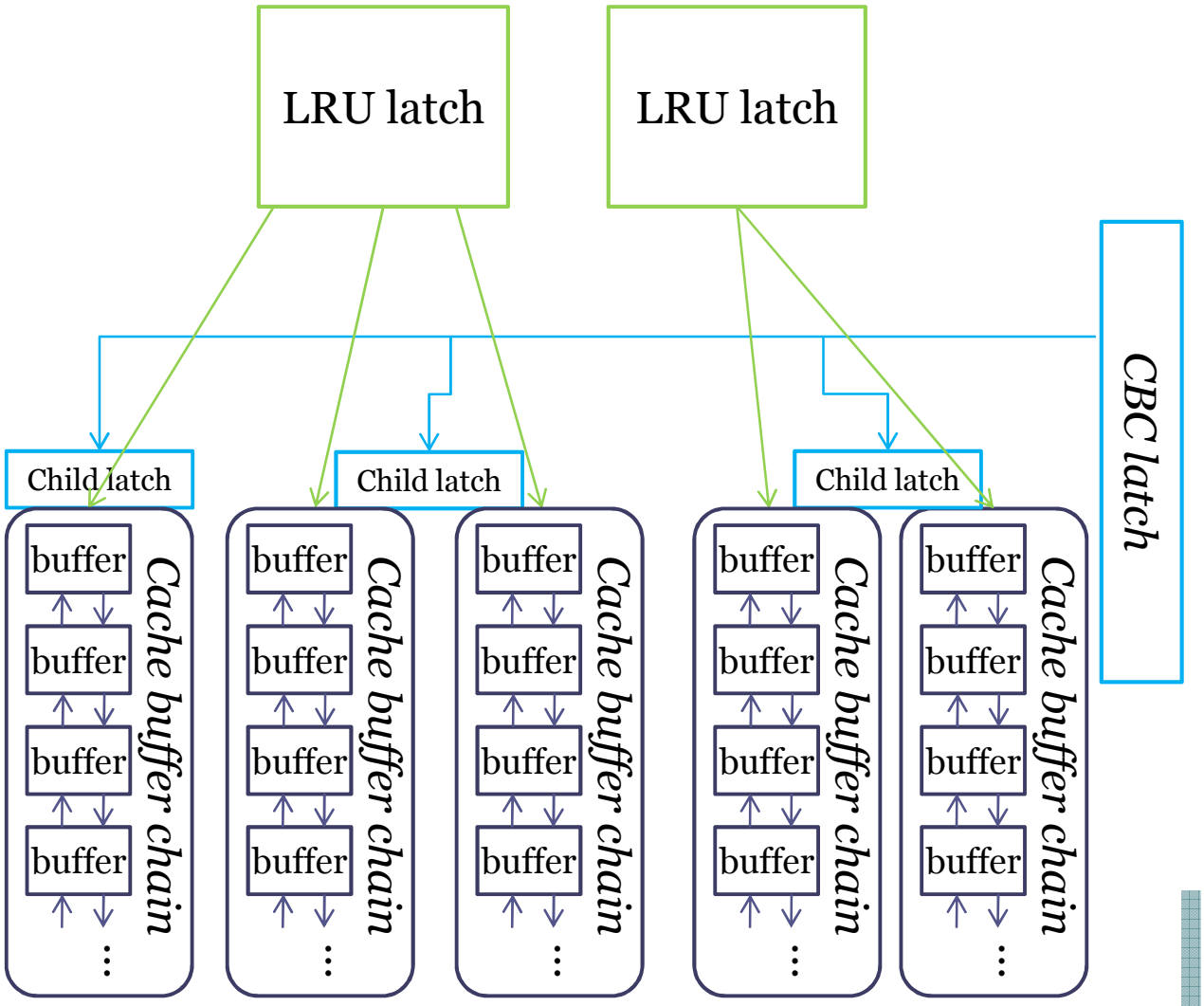
- In Oracle 10.2.0.3 - 11.2.0.2
  _db_block_hash_buckets / 32
  - observed on UP-UX, IBM AIX and Linix x86-64

Case 2: PIO

- The hash bucket is scanned, but the block is not there ☹
- We need to add an empty buffer in the bucket and read the block into it:
  - The LRU latch is obtained
    Possible contention: *cache buffer lru chain* latch
  - A free buffer is taken form the REPL-AUX list
  - The free buffer is put into the relevant CBC and pinned
  - A physical IO occurs. The contents of the block is put in the buffer
    - If another session tries to read the same block at the same time, we see the "*read by other session*" wait
  - The block is put at the midpoint of the LRU list (aka the REPL list), except for FTS. FTS puts the blocks at the "coldest" point of the LRU

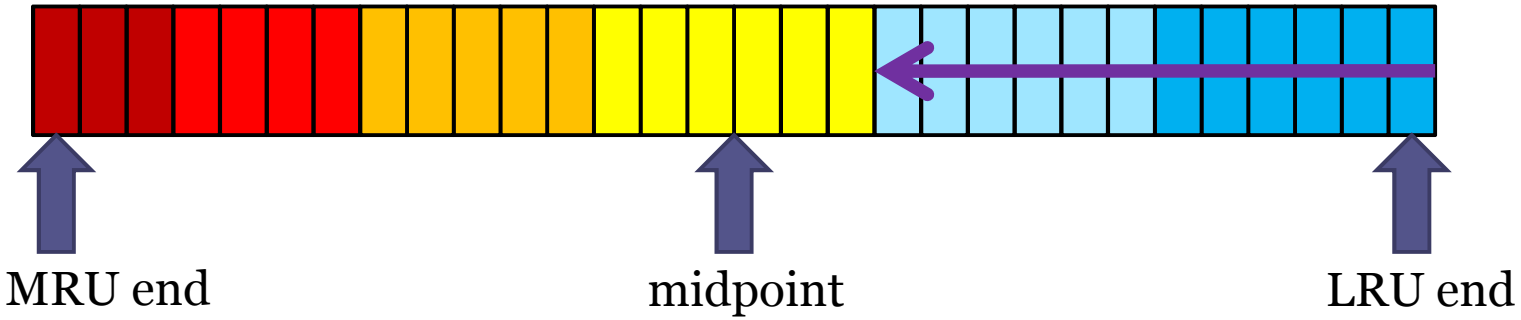
This is called PIO

```
_db_block_lru_latches  number of lru latches (e.g. 8,
  24, 32, 48, 80, 128, 192)
```


LRU latch
LRU latch
CBC latch
Child latch
Child latch
Child latch
buffer
Cache buffer chain

Special case: no free buffers in REPL-AUX

- The server process starts scanning the REPL list (**from the cold to the hot end!**)



  - If it finds a dirty buffer – it pits it in the WRITE list
  - If it finds a clean buffer – it puts it in the REPL_AUX list
- DBWR is poked to write all buffers from the WRITE list

```
_db_block_max_scan_pct = 40
Percentage of buffers to inspect when looking for free
```

Value observed on 9.2.0.6 – 11.2.0.2

Special case: we have the block in memory, but with newer SCN

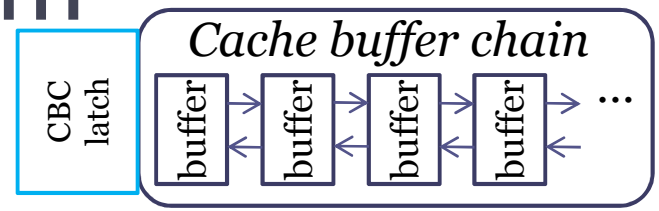
- A new buffer is taken from the REPL-AUX list and added to the CBC
- The old version of the block is reconstructed from undo and put in the buffer

We can have multiple versions of the same block in the CBC

`_db_block_max_cr_dba=6` **Maximum Allowed Number of CR buffers per dba**

Value observed on 9.2.0.6 – 11.2.0.2

The “hot block” problem



- If a block is rapidly changed
- And it is rapidly selected
- Multiple CR copies can exists in the buffer cache

- The problem is
  - All CR copies of the block have the same DBA
  - Therefore, they go to the same CBC
  - The CBC gets longer than usual
  - It takes longer to scan it…
  - (and the block is heavily used anyway)

High contention on *cache buffers chains* latch

(Solution: Avoid the hot blocks during DB/App design)

Changing the contents of a buffer (block)

Changing the buffer

- Once the buffer is found in the buffer cache, we can change it
  - The buffer is **pinned**
  - Undo image is generated
  - The data is changed
  - Redo data is generated and added to the redo log buffer
  - The buffer is marked as **dirty**
  - The buffer is added to the CKPT-Q list. However, it is not removed from the REPL list
- The block is dirtied at specific **RBA** – this is a point in the redo stream
- Our server process **never writes anything to disk** – neither the dirty block, nor the redo information

Writing dirty blocks to disk

Basic checkpoint rules

- Redo is always written (by LGWR) before the corresponding block image (DBWR)
- DBWR puts block which are being written to WRITE-AUX
- Once the block is safely on disk, DBWR removes it from CKPT-Q/WRITE/WRITE-AUX and attaches it to REPL-AUX. This is a clean buffer and can be reused.
  (however, the buffer is not emptied and stays in the REPL list with it's touch count)

Types of checkpoints

- Full checkpoint
- Thread checkpoint
- File checkpoint
- PQ checkpoint
- Object checkpoint
- Incremental checkpoint / Log switch checkpoint

More about checkpoints:
http://prutser.files.wordpress.com/2008/12/checkpointsukoug.pdf

Incremental checkpoint

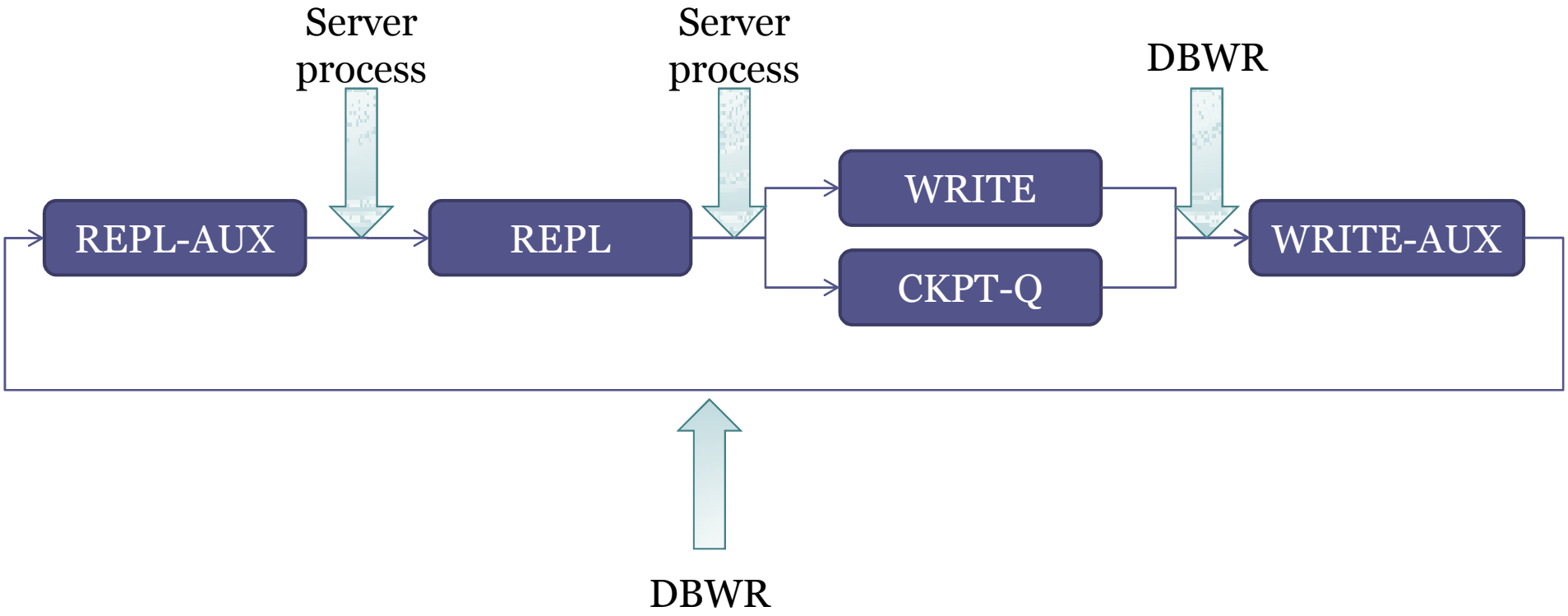
- Every 3 seconds CKPT calculates the checkpoint target RBA based on:
  - The most current RBA
  - log_checkpoint_timeout
  - log_checkpoint_interval
  - fast_start_mttr_target
  - fast_start_io_target
  - 90% of the size of the smallest online redo log file
- LGWR flushes the redo buffer up to the target RBA
- DBWR writes dirty buffers from CKPT-Q up to the target RBA
  - Moves batches of buffers to WRITE-AUX
  - Does the write
  - Moves the buffers to REPL-AUX. They can be safely reused
- CKPT updates the control file with the checkpoint progress

Log switch checkpoint

- Like the incremental checkpoint, but is triggered by a log switch
- Updates controlfile and datafile headers

That’s all!

Wrap up


Server process
Server process
DBWR
REPL-AUX
REPL
WRITE
CKPT-Q
WRITE-AUX
DBWR

Q & A ?

Thanks to
Harald van Breederode, Oracle http://prutser.wordpress.com/
Mihail Kalachev, Mobiltel

Yavor Ivanov
Oracle Certified Master
http://blog.yavor.info/