MySQL

ORACLE

MySQL Performance: Demystified Tuning & Best Practices

Dimitri KRAVTCHUK
MySQL Performance Architect @Oracle

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.

Are you Dimitri?.. ;-)

- Yes, it's me :-)
- Hello from Paris! ;-)
- Passionated by Systems and Databases Performance
- Previous 15 years @Sun Benchmark Center
- Started working on MySQL Performance since v3.23
- But during all that time just for “fun” only ;-)
- Since 2011 “officially” @MySQL Performance full time now
- http://dimitrik.free.fr/blog / @dimitrik_fr

Agenda

- Focus on Singe MySQL Server Instance Performance
- with a mix of and in any order :
  - Overview of MySQL Performance
  - Performance improvements up to MySQL 5.7 & Dev
  - What can be Tuned / and what should be Avoided
  - Pending Issues and Workarounds..
- Q & A
  - (and sorry in advance for many “smiles” in the slides ;-))

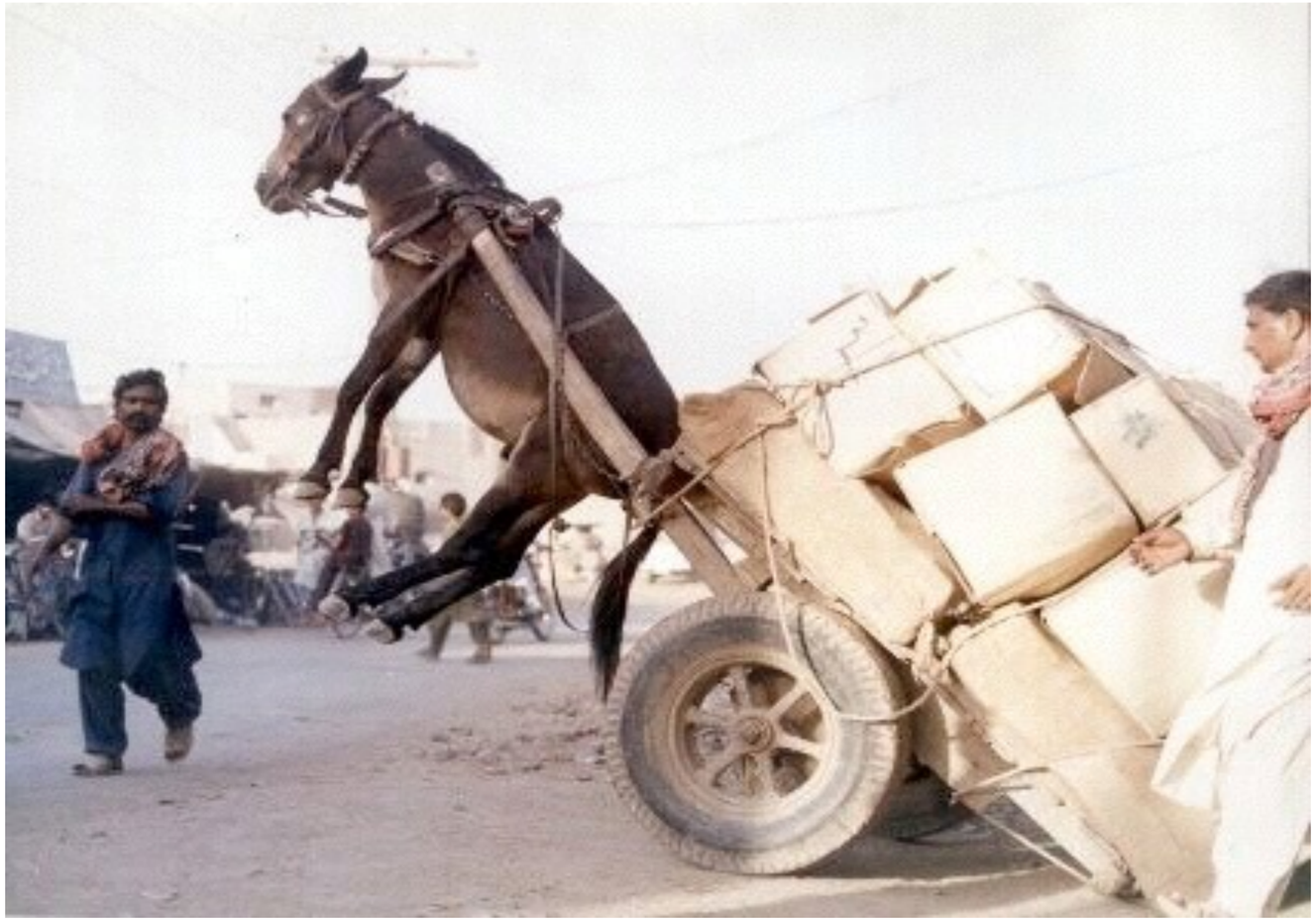
Why MySQL Performance ?...

Why MySQL Performance ?..

- Any solution may look “good enough”...

Why MySQL Performance ?..

- Until it did not reach its limit..



Why MySQL Performance ?..

- And even improved solution may not resist to increasing load..

Why MySQL Performance ?..

- And reach a similar limit..

Why MySQL Performance ?..

- Analyzing your workload performance and testing your limits may help you to understand ahead the resistance of your solution to incoming potential problems ;-)

Why MySQL Performance ?..

- However :
  - Even a very powerful solution but leaved in wrong hands may still be easily broken!... :-)

The Main MySQL Performance Best Practice #1 is... ???..

The Main MySQL Performance Best Practice #1 is... ???..

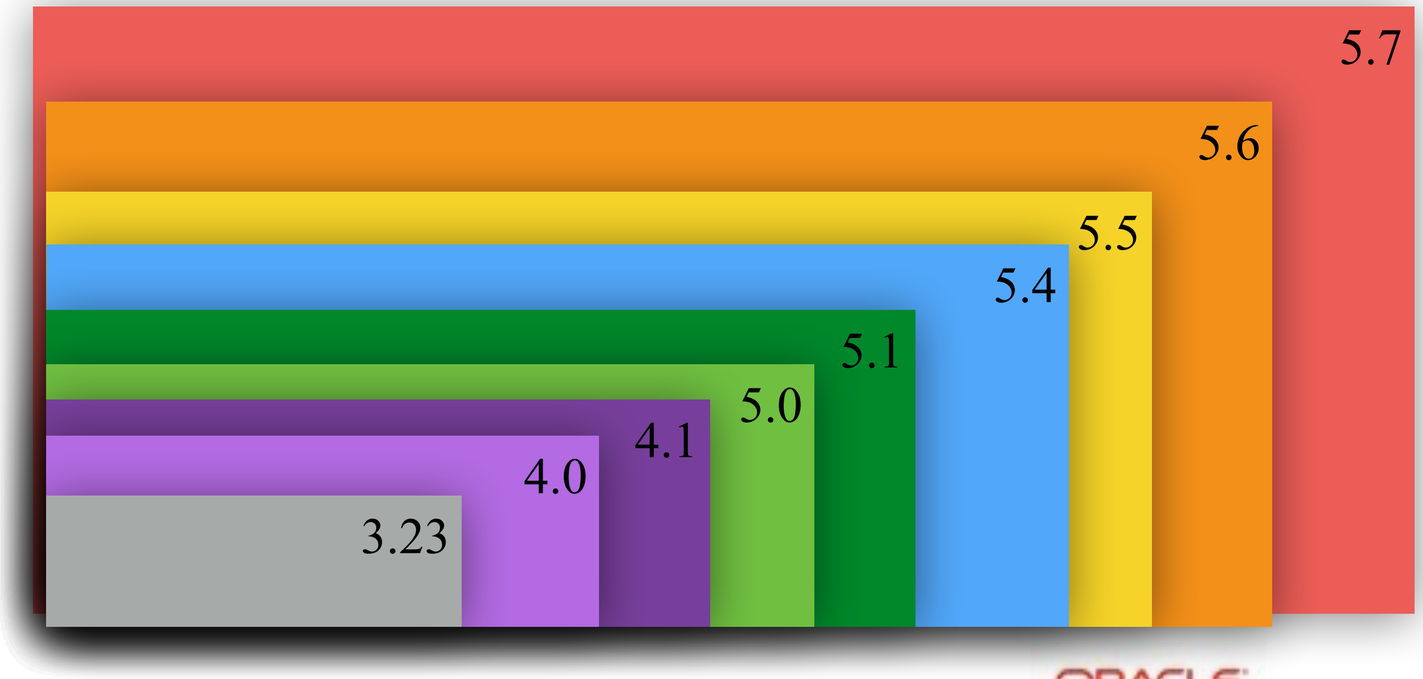
USE YOUR BRAIN !!!... ;-)

The Main MySQL Performance Best Practice #1 is... ???..

USE YOUR BRAIN !!!... ;-)

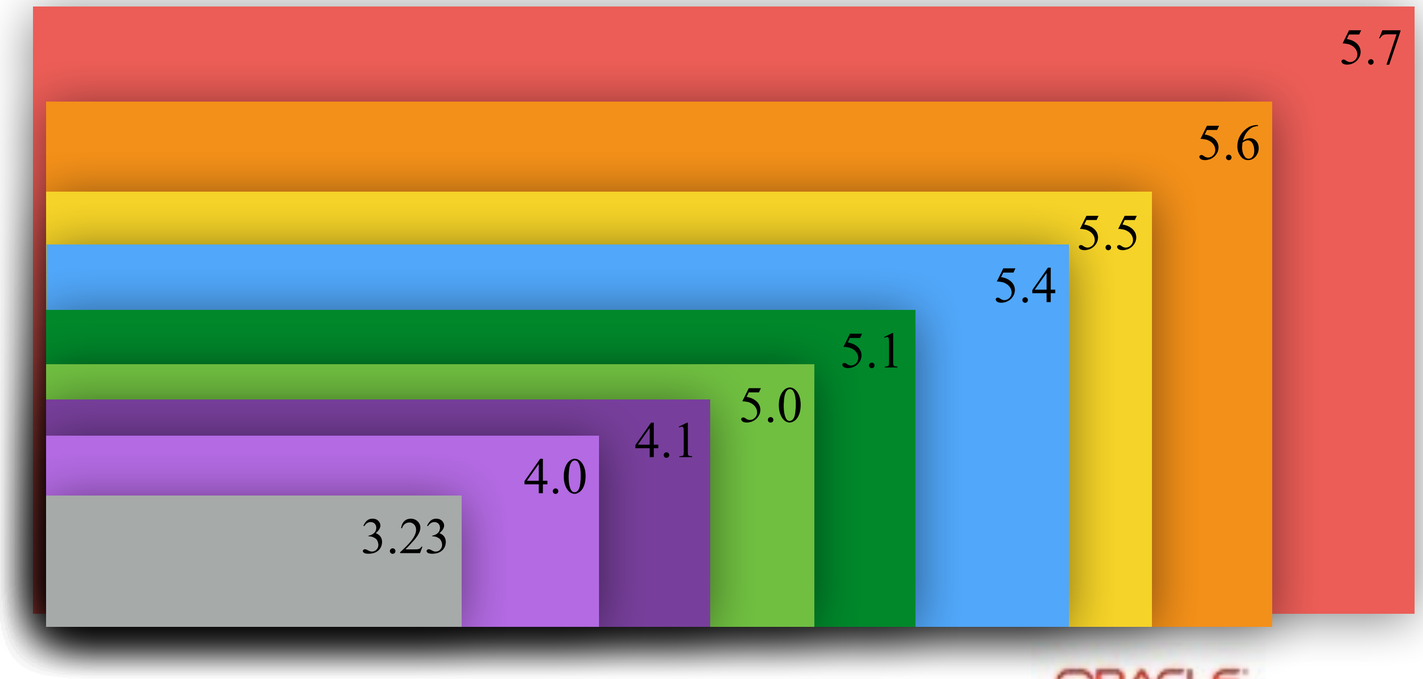
MySQL Performance Evolution

- From version-to-version :
  - 3.23 => 4.0 => 4.1 => 5.0 => 5.1 => 5.4 => 5.5 => 5.6 => 5.7 …
  - More features => longer code path.. (just google: “What is new in MySQL 5.7”)
  - MySQL/InnoDB code is very sensible to CPU cache(s)..
  - Going slower :
    - single-user..
    - low-load..
    - small-HW..
  - Going faster :
    - where scalability was improved
    - higher-load..
    - newer/bigger-HW..



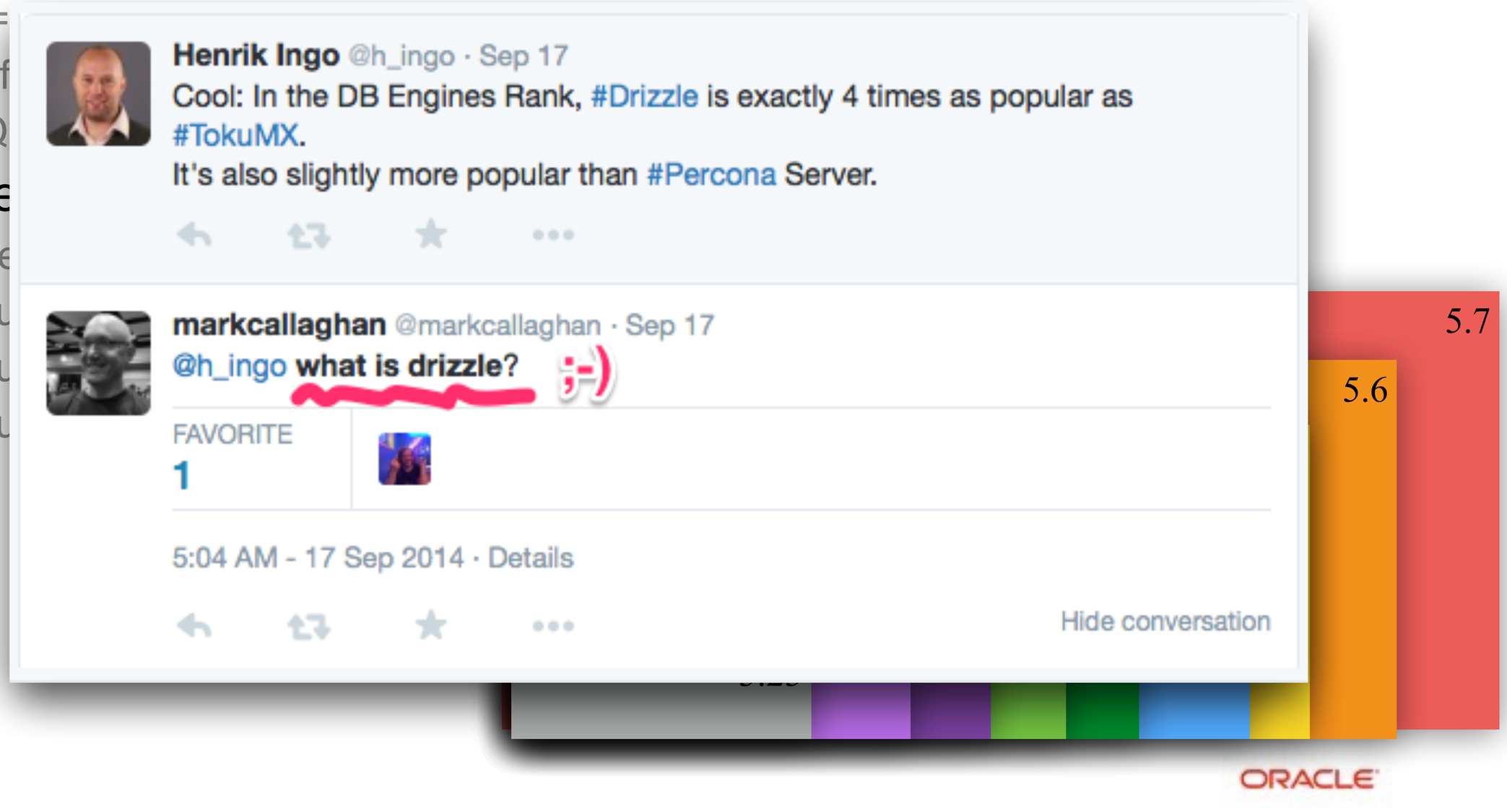
MySQL Performance Evolution

- From version-to-version :
  - 3.23 => 4.0 => 4.1 => 5.0 => 5.1 => 5.4 => 5.5 => 5.6 => 5.7 …
  - More features => longer code path.. (just google: “What is new in MySQL 5.7”)
  - MySQL/InnoDB code is very sensible to CPU cache(s)..
- Less featured MySQL ?
  - Drizzle !
  - do you know Drizzle ?
  - do you use Drizzle ?
  - do you run your production on ?



MySQL Performance Evolution

- From version-to-version :
  - 3.23 =
  - More f
  - MySQ
- Less fe
  - Drizzle
  - do you
  - do you
  - do you

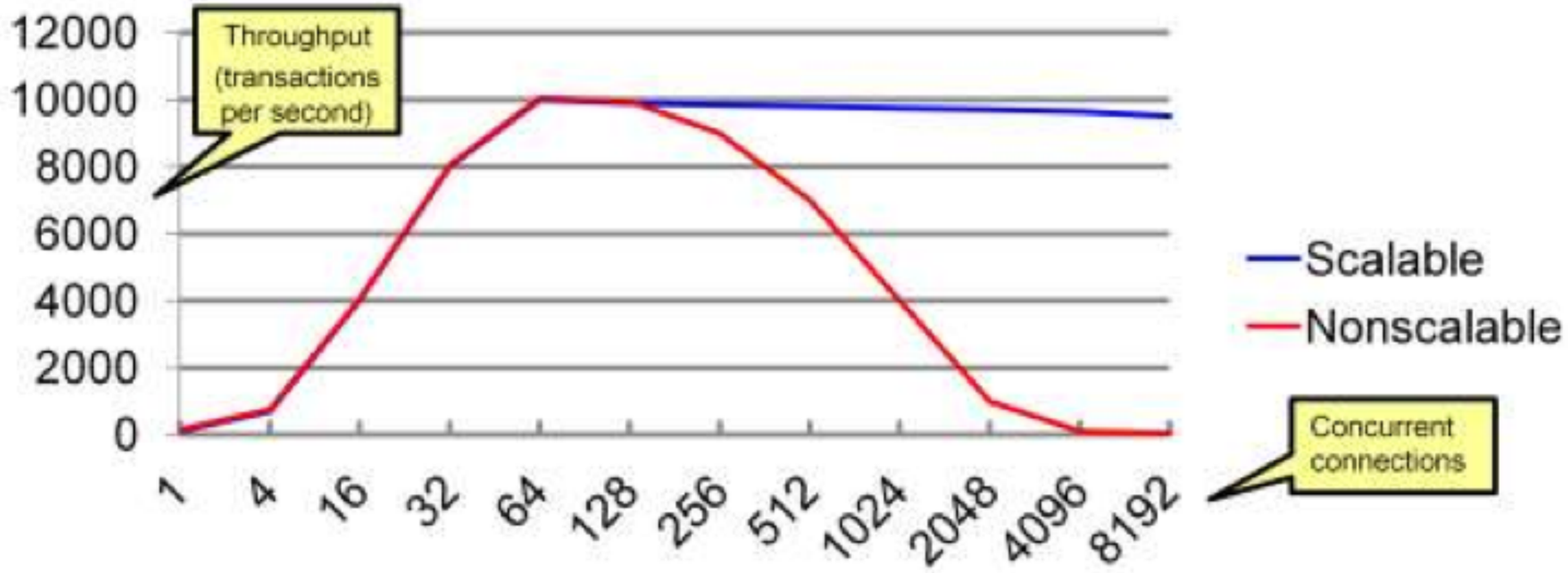


Why Scalability ?..

- CPU Speed : no more "free lunches" ;-)
  - will x2 times faster CPU increase your performance by x2 ?..
- CPU cores : more and more over year-to-year..
  - Intel 2CPU : 8cores-HT
  - Intel 2CPU : 12cores-HT
  - Intel 2CPU : 16cores-HT
  - Intel 2CPU : 20cores-HT
  - Intel 2CPU : 36cores-HT (2015)
  - Intel 2CPU : 44cores-HT (Mar.2016)
  - …
  - 2016: 4cores ==> “commodity HW” for a SmartWatch ;-)
- Scalability In Few Words :
  - your software is able to deliver a **higher** throughput if more HW resources are available..
  - (then, scaling it well or not is another story ;-))

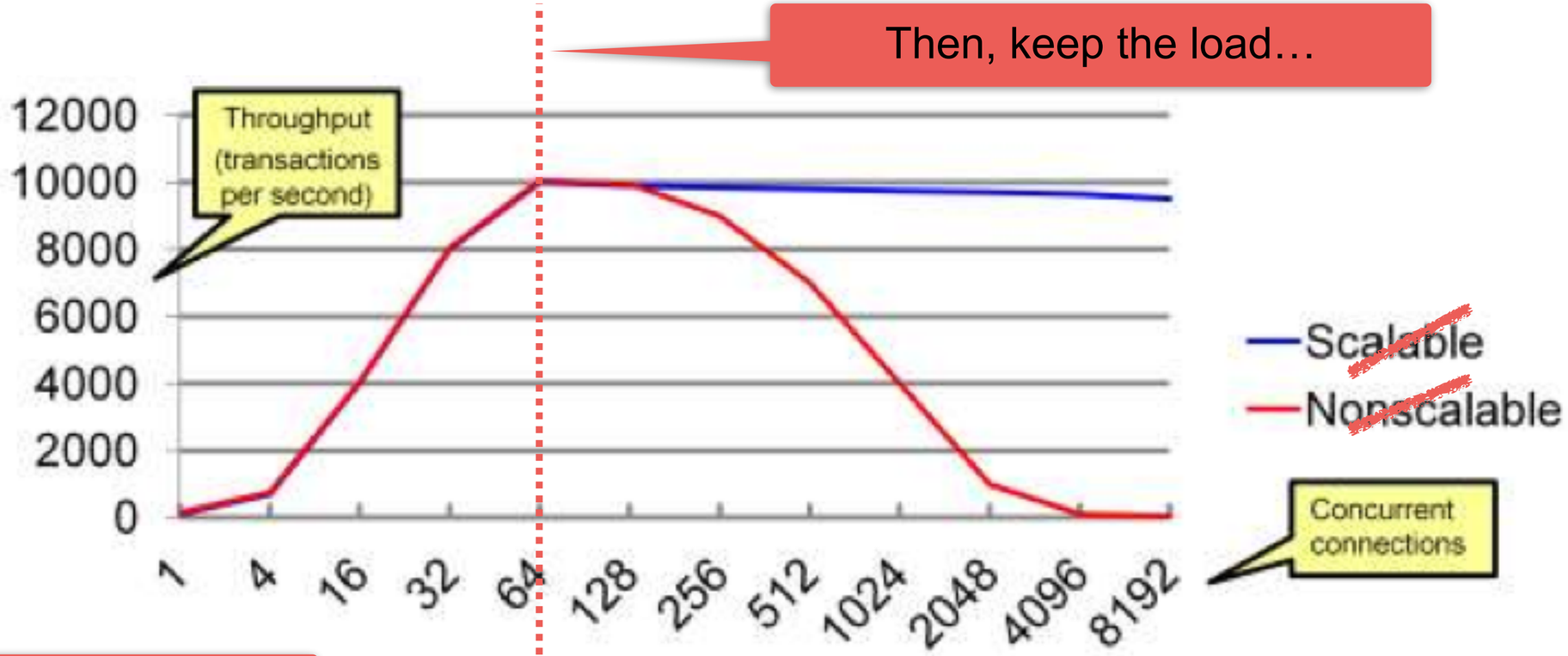
A B-shit Slide…

- Odd interpretation of Scalability…



A B-shit Slide… (2)

- Odd interpretation of Scalability…

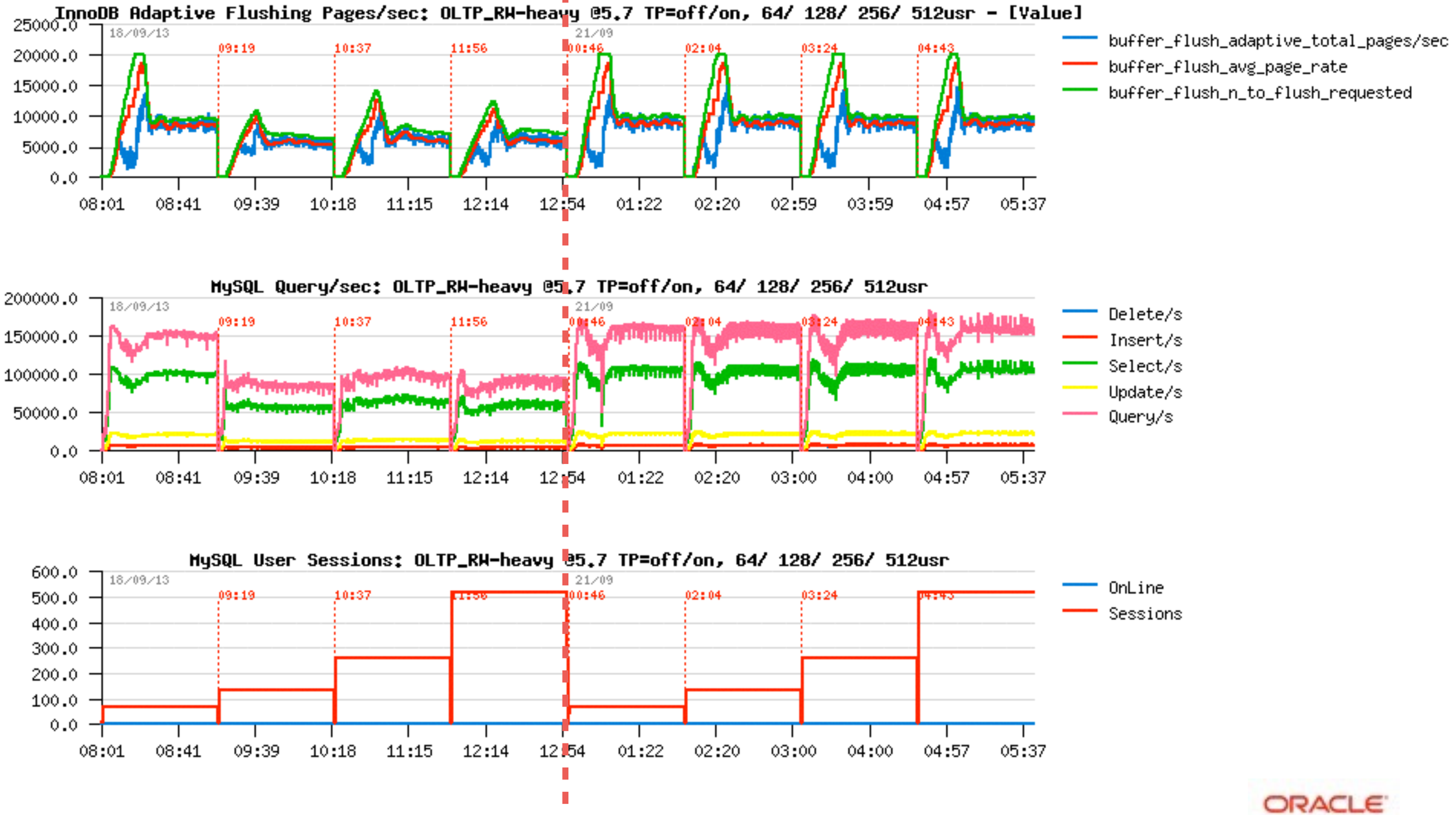


Both are scaling up to 64 connections,
but only one is able to keep a higher load..

MySQL on High Load

- Once you've reached your Max TPS on your system :
  - try to understand first what is limiting you? (I/O, CPU, Network, MySQL internals?)
  - the next goal then: to avoid a TPS "regression" on a higher load
- How to keep your Max TPS on a higher load too?
  - the dumb rule : avoid to have a load higher than you're able to keep ! ;-)
  - seriously :
    - usually all you need is to find a way to do not let you workload concurrency out-pass the levels you're reaching on the TPS Max, that's all..
    - InnoDB thread concurrency helps here (yet more improved in MySQL 5.7)
    - InnoDB spin wait delay tuning helps to lower impact of mutexes / rw-locks waits
    - ThreadPool / ProxySQL / etc..
  - **NOTE : there is no "magic" for response time :**
    - if your Max TPS you're reaching on N users
    - and able to keep the same Max TPS on N x2 users (or x3, x4, etc.)
    - your response time may only grow! (and be x2 times bigger (or x3, or x4, etc.))

Thread Pool in old MySQL 5.7 @Heavy OLTP_RW



Starting point : “Tuning” HW / OS / FS related choices

- Linux :
  - LD_PRELOAD MT-oriented malloc: **jemalloc**, tcmalloc, etc.
  - right IO scheduler (not cfq)
  - right FS/ mount options/ AIO/ O_DIRECT/ etc..
    - nobarriers,noatime,nodirtime,…
- Solaris :
  - LD_PRELOAD MT-oriented malloc: mtmalloc, **umem**
  - UFS/forcedirectio
  - ZFS
- why not shared storage / ZFS Appliance / etc..
- the main rule : **TEST before deploy !!!**
  - **know your HW / OS / FS limits !!!**

Test Before! - Only a real test gives you a real answer...

- Avoid to tweak on production systems ;-)
  - Rather try to reproduce your load on a similar, but dedicated to test server
  - Collect test cases for all the most critical parts..
- Want to simulate your production workload?..
  - Then just simulate it! (many SW available, not always OSS/free)
  - Hard to simulate? - adapt some generic tests
- Want to know capacity limits of a given platform?
  - Still try to focus on the test which are most significant for you!
- Want just to validate config settings impacts?
  - Focus on tests which are potentially depending on these settings
- Well, just **keep thinking** about what you're doing ;-)

“Generic” Test Workloads @MySQL

- Sysbench
  - OLTP, RO/RW, N-tables, lots test workload load options, deadlocks
- DBT2 / TPCC-like
  - OLTP, RW, very complex, growing db, no options, deadlocks
  - In fact using mostly only 2 tables! (thanks Performance Schema ;-))
- dbSTRESS
  - OLTP, RO/RW, several tables, one most hot, configurable, no deadlocks
- iiBench
  - pure INSERT (time series) + SELECT
- LinkBench (Facebook)
  - OLTP, RW, very intensive, IO-hungry..
- DBT3
  - DWH, RO, complex heavy query, loved by Optimizer Team ;-)

Side Note on “Generic” Test Workloads @MySQL

- For MySQL Dev Team :
  - each generic workload represents several real problems to fight
  - there is no “marketing” ;-)
  - just a hard work on investigation & fixing of problems..

Test Case / Workload Scenario

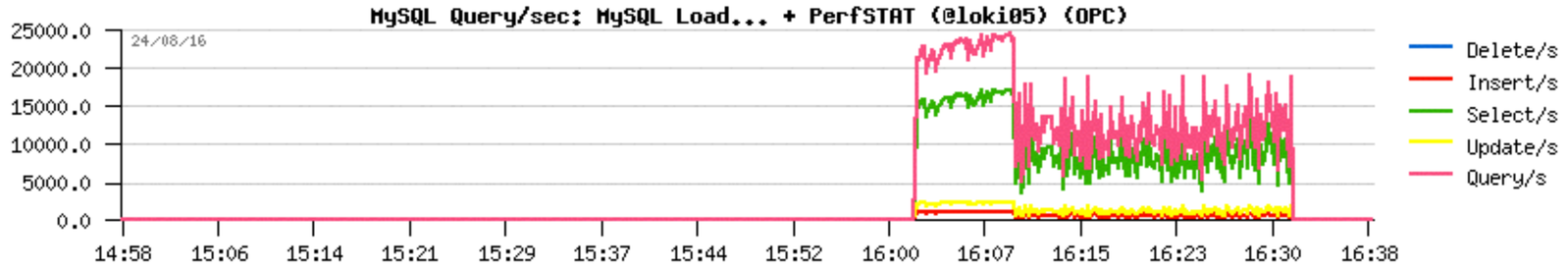
- Before to jump into something complex...
  - **Be sure first you're comfortable with “basic” operations!**
  - Single table? Many tables?
  - Short queries? Long queries?
- Remember: any complex load in fact is just a mix of simple operations..
  - So, try to split problems..
  - Start from as simple as possible..
  - And then increase complexity progressively..
- NB : **any** test case is important !!!
  - Consider the case rather reject it with “I’m sure you’re doing something wrong..” ;-))

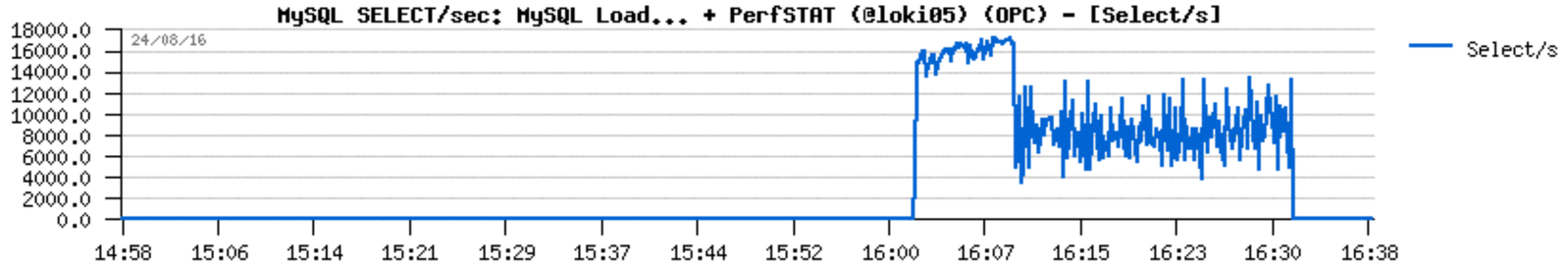
#2 - Monitoring is THE MUST !

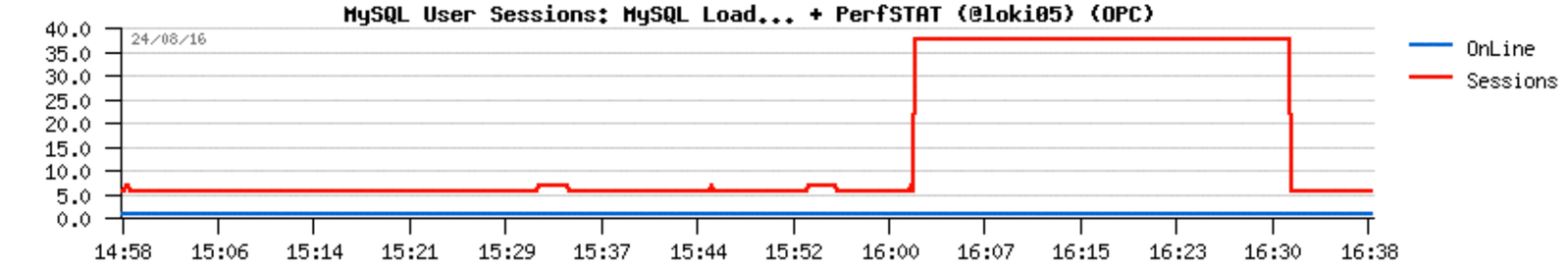
even don’t start to touch anything
without monitoring.. ;-)

You can be surprised but what you may discover..

- MySQL Activity :

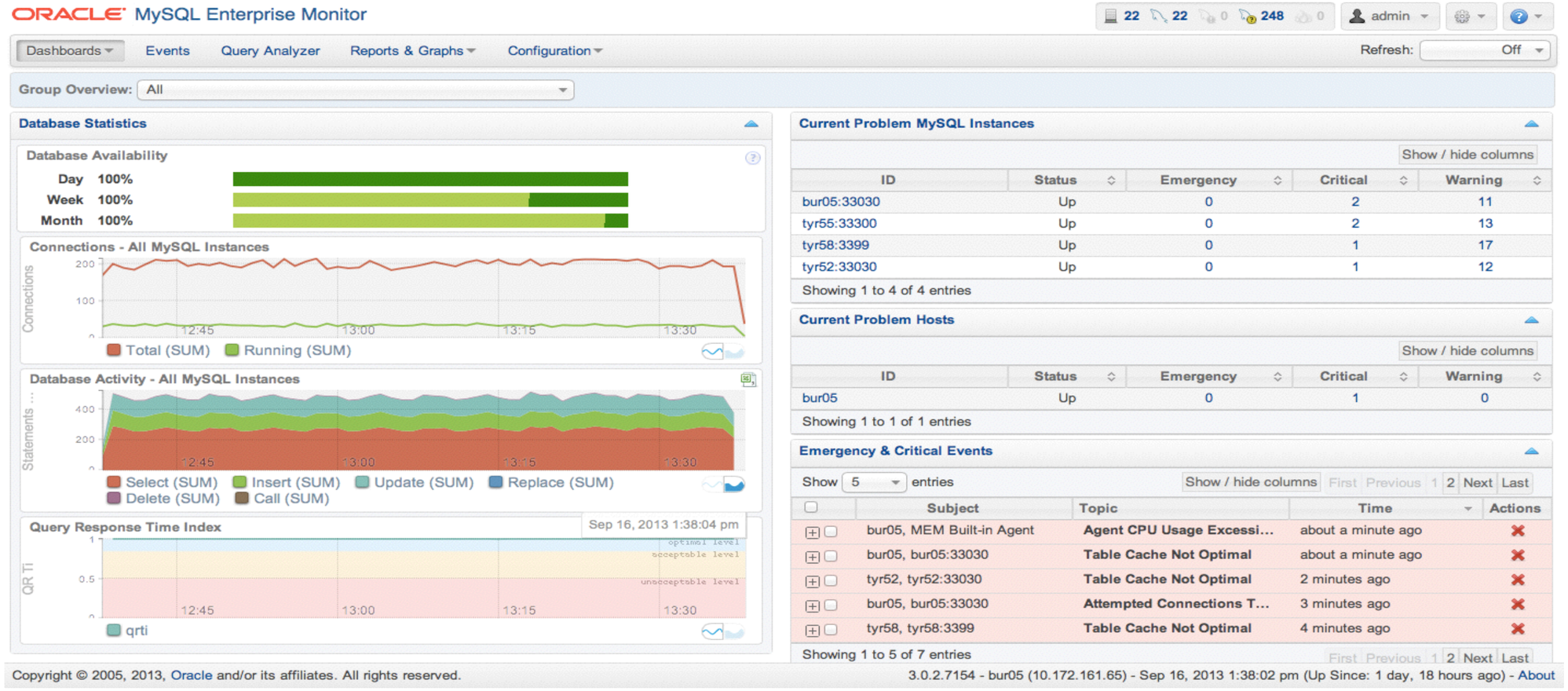






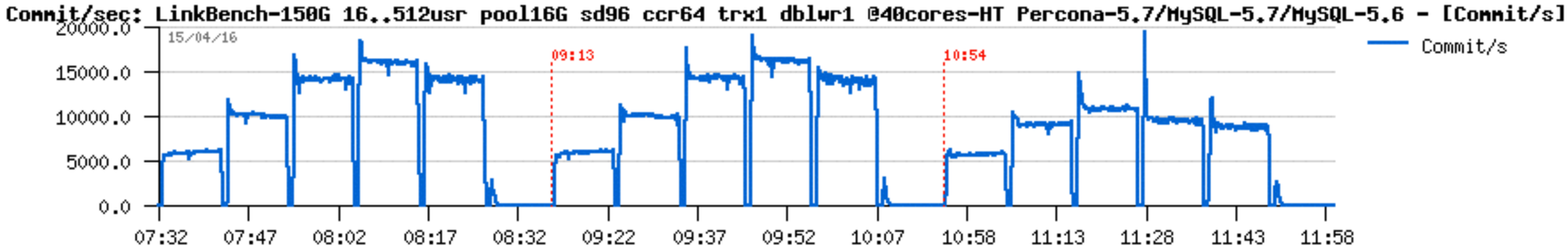
MySQL Enterprise Monitor

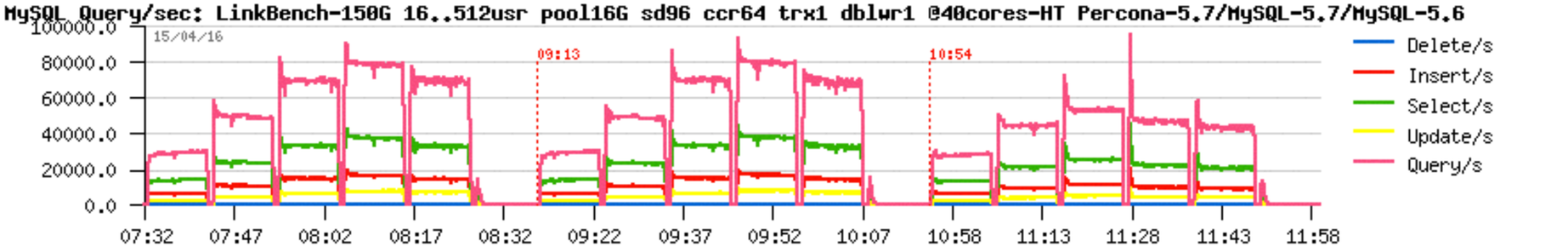
- Fantastic tool!
  - Did you already try it?.. Did you see it live?..



Other Monitoring Tools

- Cacti, Zabbix, Nagios, Solarwinds, etc…..
- *dim_STAT*
  - well, I'm using this one, sorry ;-)
  - all graphs within presentation were made with it
  - details are on the last slides of presentation..



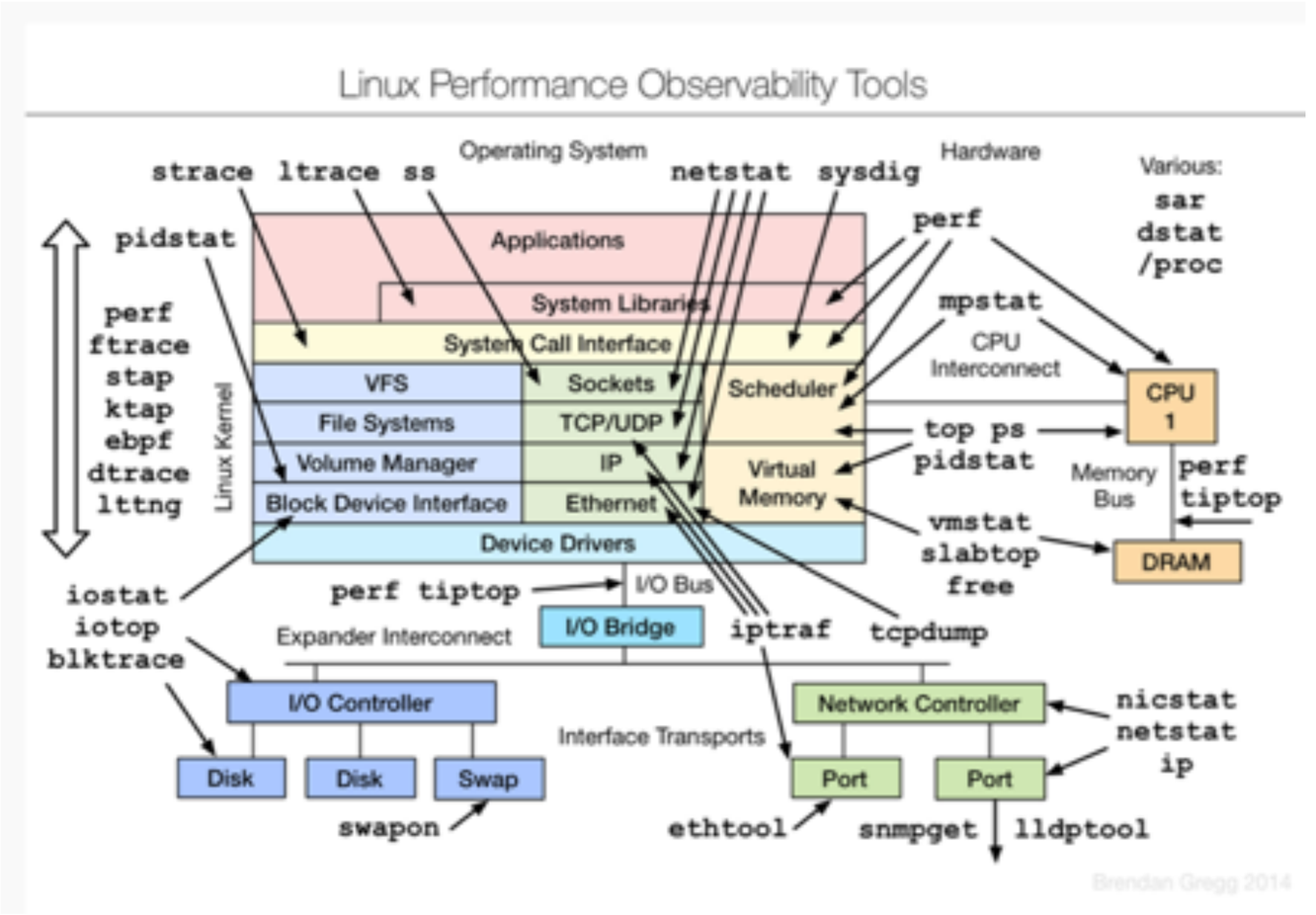


A Word about Monitoring…

- **always** validate the impact of your Monitoring on your Production ;-)
- taking 1sec measurements is fine, except :
  - if it's eating 100% CPU time on one or more CPU cores..
  - reducing your network traffic / latency..
  - eats your RAM, etc.
- avoid to be too much intrusive on MySQL/InnoDB internals..
  - you may easily create an additional overhead
  - as well you may add artificial locks on your workflow
    - for ex: run in loop "show processlist", etc..
- well, nothing is coming for free, so **think** about what you're doing !
- (#1 best practice once again ;-))

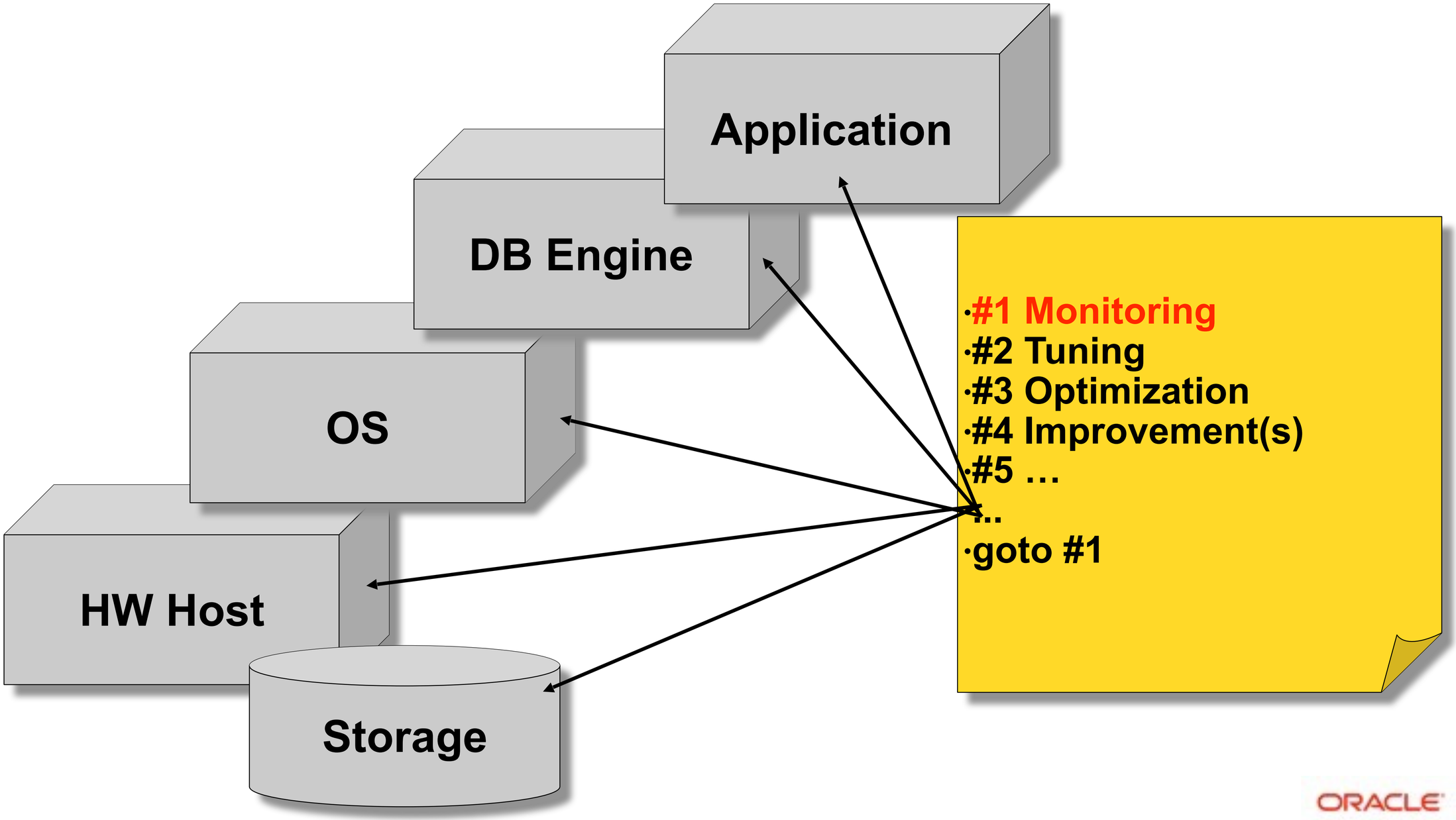
System Monitoring (Linux)

- Keep an eye on :
  - CPU Usage%
  - Run queue
  - RAM / swap
  - Top processes
  - I/O op/sec / MB/sec
  - Network traffic
  - etc..

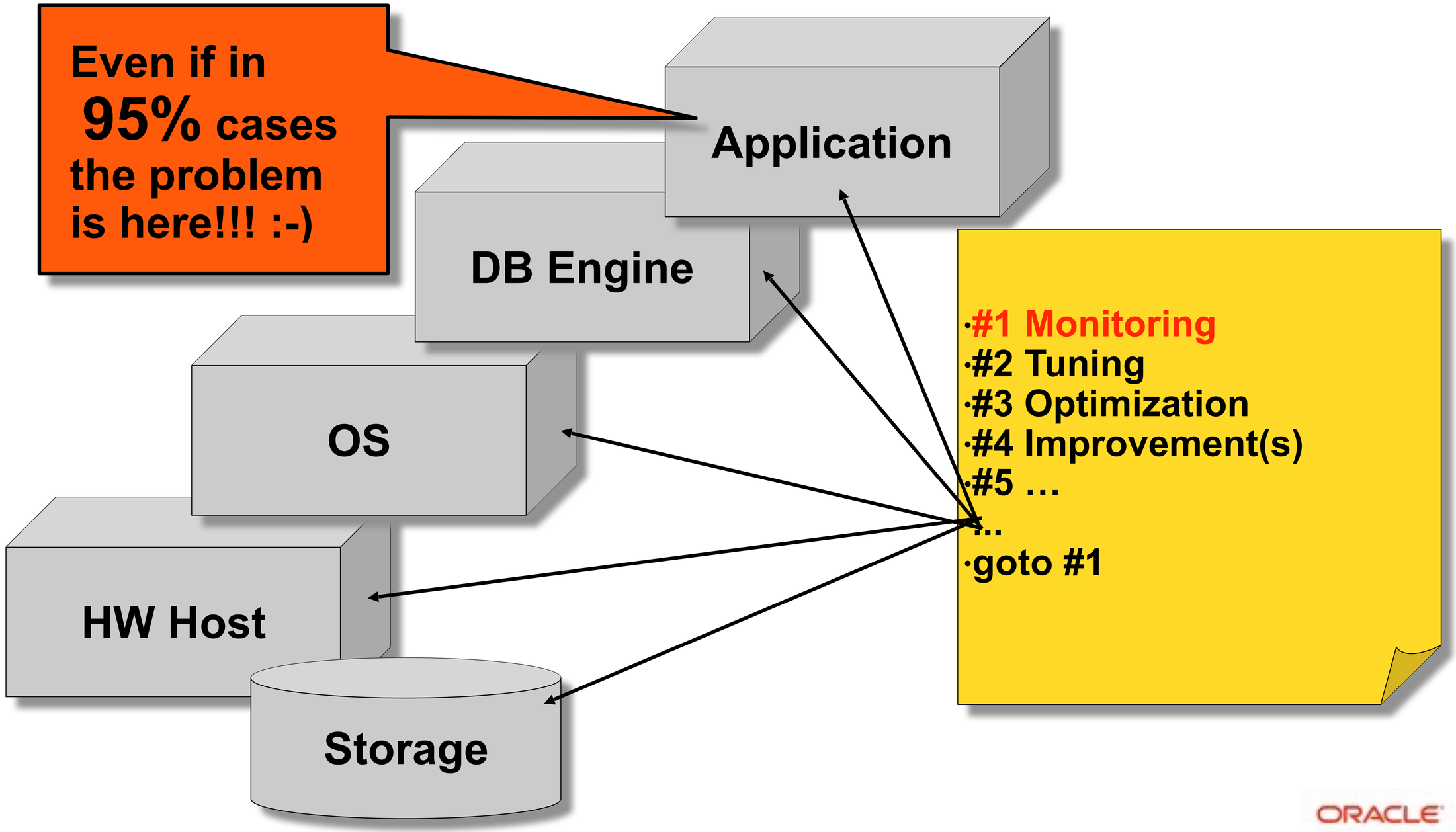


Credits image : Brendan GREGG (http://www.brendangregg.com)

The Infinitive Loop of Database Tuning...



The Infinitive Loop of Database Tuning...



What to Monitor ?..

- Everything ;-)
- The main goal of Monitoring :
  - to understand what is changed once you're hitting a performance problem..
  - (all the diff between "good" -vs- "bad")
  - otherwise all this is useless ;-))
- Then :
  - be sure the problem is coming from MySQL..
  - be sure you're not hitting any system limits !!
  - be sure you're not hitting MySQL internal limitations..

Using “perf” (Linux) — low impact profiler

- Use cases :
  - # perf top -z --stdio <== live monitoring
  - # perf record -a -g -f -F 99 -- sleep 20 <== record 20sec of data
  - # perf report | more <== report from collected data
  - # perf annotate <== jump to source code
- links :
  - https://perf.wiki.kernel.org <== main resource
  - http://www.brendangregg.com/perf.html <== the most fun stuff !!!
    - Thanks Brendan! ;-))

Profiling example: # perf top -z --stdio

- Observations :
  - nothing special..
  - mysqld is the top running process, fine..

```
PerfTop:  312195 irqs/sec  kernel:19.2%  exact:  0.0% [4000Hz cycles],  (all, 80 CPUs)
-------------------------------------------------------------------------------

   4.77%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] my_hash_sort_simple
   4.42%  libc-2.12.so                               [.] memcpy
   2.87%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] row_search_mvcc(unsigned char*, unsigned
   2.29%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] rec_get_offsets_func(unsigned char const*
   1.81%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] buf_page_get_gen(page_id_t const&, page_s
   1.59%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] my_strnxfrm_simple
   0.96%  libmysqlclient_r.so.16.0.0                 [.] 0x0000000000058710
   0.94%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] btr_cur_search_to_nth_level(dict_index_t*
   0.89%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] page_cur_search_with_match(buf_block_t co
   0.84%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] _ZL19rw_lock_s_lock_funcP9rw_lock_tmPKcm.
   0.74%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] cmp_dtuple_rec_with_match_low(dtuple_t co
   0.66%  libc-2.12.so                               [.] __memset_sse2
   0.60%  [kernel]                                   [k] copy_user_generic_string
```

Profiling example (2)

- Observations :
  - memcpy() is the most hot, called by mysqld (check call-stack)
  - nothing to do.. (check apps, SELECT ranges, etc..)

```
PerfTop:  286835 irqs/sec  kernel:20.2%  exact:  0.0% [4000Hz cycles],  (all, 80 CPUs)
-------------------------------------------------------------------------------

   8.49%  libc-2.12.so                                [.] memcpy
   4.90%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] row_search_mvcc(unsigned char*, unsigned
   3.33%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] rec_get_offsets_func(unsigned char const*
   1.95%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] buf_page_get_gen(page_id_t const&, page_s
   1.46%  libmysqlclient_r.so.16.0.0                  [.] 0x000000000005862f
   1.36%  [kernel]                                    [k] copy_user_generic_string
   1.15%  [kernel]                                    [k] native_write_msr_safe
   1.02%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] btr_cur_search_to_nth_level(dict_index_t*
   0.98%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] page_cur_search_with_match(buf_block_t co
   0.98%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] _ZL19rw_lock_s_lock_funcP9rw_lock_tmPKcm.
   0.94%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] evaluate_join_record(JOIN*, QEP_TAB*)
   0.90%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] ha_innobase::general_fetch(unsigned char*
   0.87%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] my_lengthsp_8bit
   0.86%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] row_sel_store_mysql_field_func(unsigned c
   0.83%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] row_sel_field_store_in_mysql_format_func(
```

Profiling example (3)

- Observations :
  - my_hash_sort_simple() is the most hot (mysqld)
  - nothing to do.. (check apps, memory temp tables usage, query plan, etc..)

```
PerfTop:  291110 irqs/sec  kernel:12.8%  exact:  0.0% [4000Hz cycles],  (all, 80 CPUs)
-------------------------------------------------------------------------------

  22.90%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] my_hash_sort_simple
   6.24%  libc-2.12.so                               [.] memcpy
   4.09%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] my_strnxfrm_simple
   2.57%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] row_search_mvcc(unsigned char*, unsigned
   1.71%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] rec_get_offsets_func(unsigned char const*
   1.59%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] hp_write_key
   1.15%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] void std::__merge_sort_with_buffer<unsign
   1.06%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] hp_rec_hashnr
   0.96%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] evaluate_join_record(JOIN*, QEP_TAB*)
   0.88%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] buf_page_get_gen(page_id_t const&, page_s
   0.81%  libmysqlclient_r.so.16.0.0                 [.] 0x000000000005881f
   0.71%  [kernel]                                   [k] copy_user_generic_string
   0.67%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] filesort(THD*, QEP_TAB*, Filesort*, bool,
   0.56%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] long long compare_between_int_result<unsi
   0.54%  [kernel]                                   [k] native_write_msr_safe
   0.52%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] page_cur_search_with_match(buf_block_t co
   0.51%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] heap_write
   0.50%  mysqld-576-withPFS-O3-Sep17-no_omit-futex  [.] ha_innobase::general_fetch(unsigned char*, unsign
```

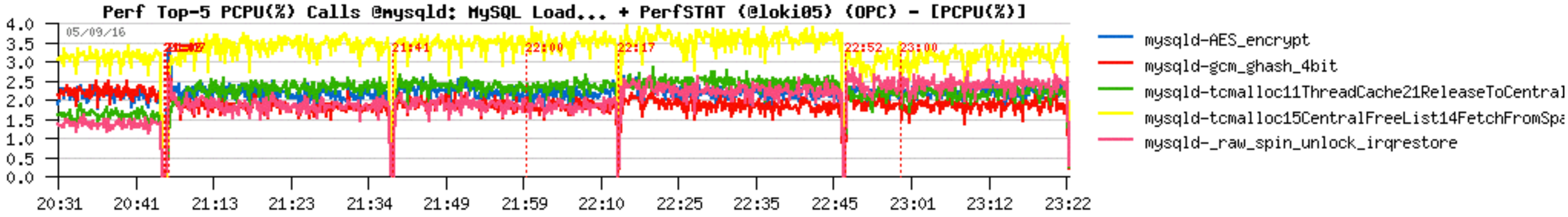
Profiling example (4)

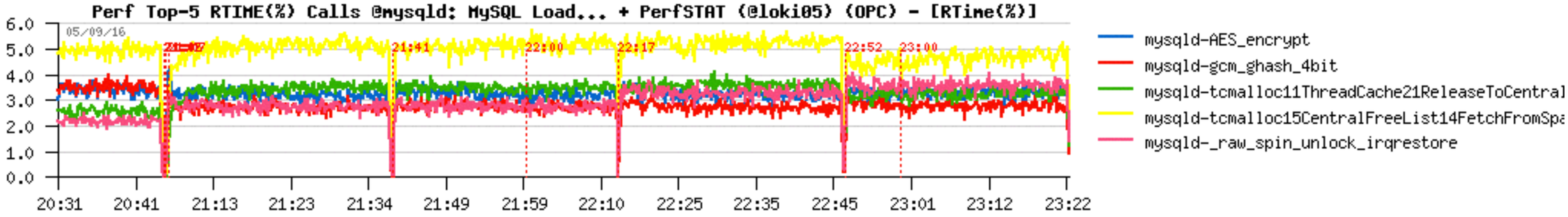
- Observations :
  - _spin_lock() is the most hot (or ut_delay, or rw_lock*, or *lock*, etc.)
  - you're hitting a lock contention! (MySQL or not)
  - if MySQL : analyze PFS waits, innodb status, mutex status, etc..

```
PerfTop:  296349 irqs/sec  kernel:44.9%  exact:  0.0% [4000Hz cycles],  (all, 80 CPUs)
-------------------------------------------------------------------------------

  29.04%  [kernel]                                    [k] _spin_lock
  15.39%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] PolicyMutex<TTASFutexMutex<NoPolicy> >::e
  10.18%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] mtr_t::Command::prepare_write()
   6.21%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] ut_delay(unsigned long)
   1.55%  [kernel]                                    [k] native_write_msr_safe
   1.52%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] log_write_up_to(unsigned long, bool)
   0.78%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] rw_lock_x_lock_func(rw_lock_t*, unsigned
   0.59%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] buf_page_get_gen(page_id_t const&, page_s
   0.44%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] btr_cur_search_to_nth_level(dict_index_t*,
   0.38%  [kernel]                                    [k] schedule
   0.35%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] lock_table(unsigned long, dict_table_t*, lo
   0.34%  mysqld-576-withPFS-O3-Sep17-no_omit-futex   [.] trx_undo_assign_undo(trx_t*, trx_undo_ptr_t
   0.27%  libjemalloc.so                              [.] free
   0.27%  libjemalloc.so                              [.] malloc
```

PerfSTAT example

- Observations :
  - tcmalloc is on top of CPU hot functions..
  - Action : try to use jemalloc instead
  - Result : x2 times better Max TPS (!)

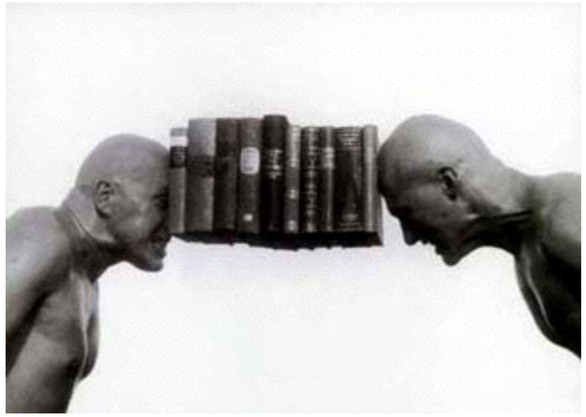




Things are progressing quickly !

- Constantly learn and improve your skills !
- Don’t delegate your production workload to googled “quick tips” ;-))

To Really Understand Something.. (The (dim) PLAN)



- Whatever Test workload you prepared :
  - **go with Read-Only scenario to begin !**
    - small data volume first (20M rows at least, in-memory)
    - then a bigger dataset, but still in-mamory
    - then IO-bound test (dataset is bigger than available memory)
  - **after what you can switch to Read+Write scenario**
    - on begin start with “relaxed” security (trx_commit=2, no binlog, no DBLWR, etc.)
    - same as RO, small data volume first (see how well your flushing will follow)
    - then bigger volume, still in-memory (see your flushing increased, higher pages activity)
    - then IO-bound (see your TPS diff vs RO, LRU flushing, IO reads, etc.)
    - then replay all again with progressively step-by-step increased data security :

      trx_commit = 1

      doublewrite = 1

      binlog & binlog sync = 100

      binlog & binlog sync = 1
    - always monitor & observe your flushing rate & times, Purge activity / History Length, Checkpoint Age, Dirty / Free pages levels, page waits / page scans, etc..

Analyzing Workloads “by pattern” : RO -vs- RW

- Read-Only (RO) :
  - Nothing more simple when comparing DB Engines, HW configs, etc..
  - RO In-Memory : data set fit in memory / BP / cache
  - RO IO-bound : data set out-passing a given memory / BP / cache
- Read+Write (RW) :
  - I/O is **ALWAYS** present ! - storage performance matters a lot !
  - may be considered as always IO-bound ;-)
  - RW In-Memory : same as RO, data set fit in memory, but :
    - small data set => small writes
    - big dataset => big writes ;-)
  - RW IO-bound : data set out-passing a memory
    - means there will be (a lot of?) reads !
- NOTE : Random Read (RR) operation is the main IO-bound killer !!!
  - so, **GO Flash !!!**

Read-Only Performance @MySQL / InnoDB

- Depends on a workload..
  - sometimes the limit is only within your memcpy() rate ;-)
- But really started to scale only since MySQL 5.7
  - due improved TRX list management, MDL, THR_lock, etc..
  - scaling up to 96 CPU cores-HT for sure, reported on more cores too..
  - Note : remind my “scalability” notes ;-))
  - Note : code path is growing with new features! (small HW may regress)

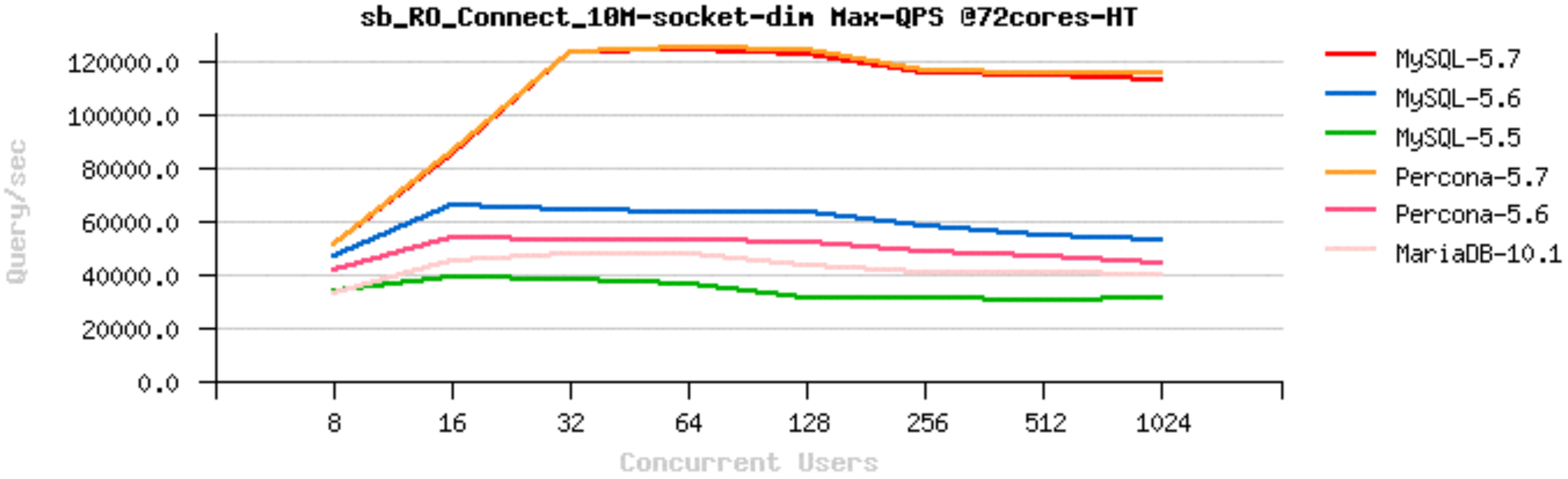
RO related starter configuration settings

- my.conf :

```
join_buffer_size=32K .. 2M
sort_buffer_size=32K .. 2M

table_open_cache = 8000
table_open_cache_instances = 16
query_cache_type = 0

innodb_buffer_pool_size= 64000M (2/3 RAM ?)
innodb_buffer_pool_instances = 32
innodb_thread_concurrency = 0 / 32 / 64
innodb_spin_wait_delay= 6 / 48 / 96

innodb_stats_persistent = 1
innodb_adaptive_hash_index= 0 / 1
innodb_monitor_enable = '%'
```

Entry Ticket : RO_Connect

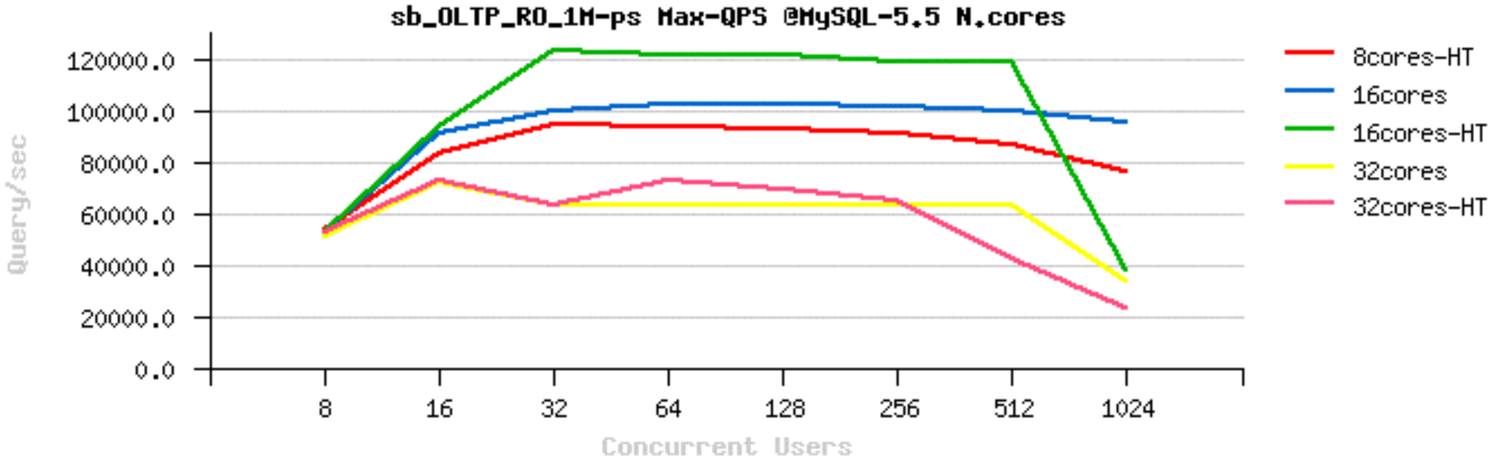
- Many web apps cannot use persistent connections
  - connect => Query(s) => disconnect

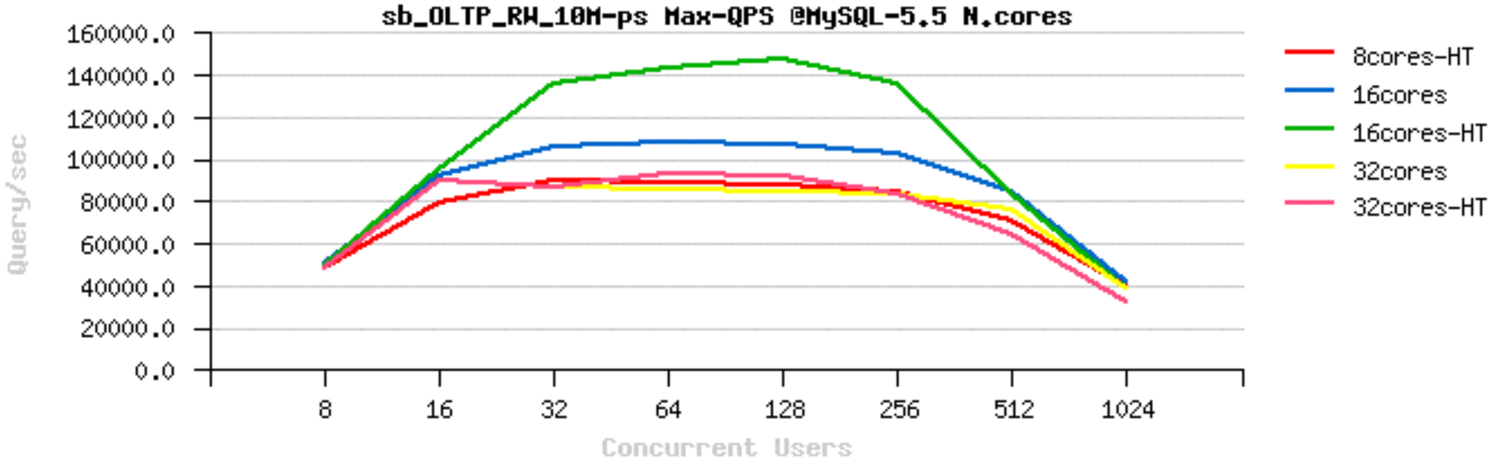


Why so much attention to RO Performance in MySQL 5.7 ?..

From where we're coming with MySQL 5.7 ?..

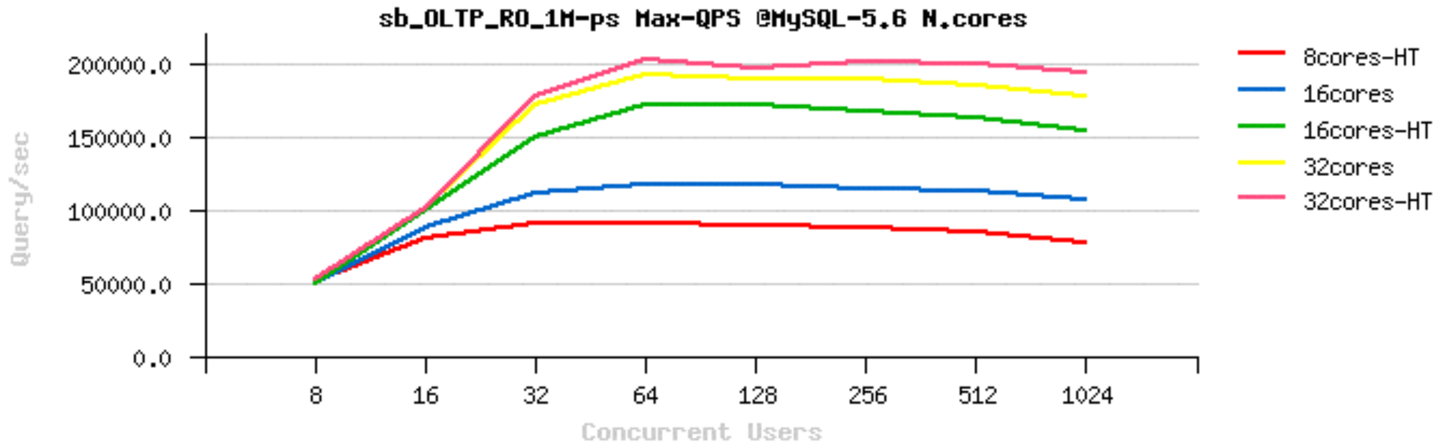
- MySQL 5.5 : RO & RW
  - QPS Max on 16cores
  - worse on 32cores
  - Note: RW out-pass RO!

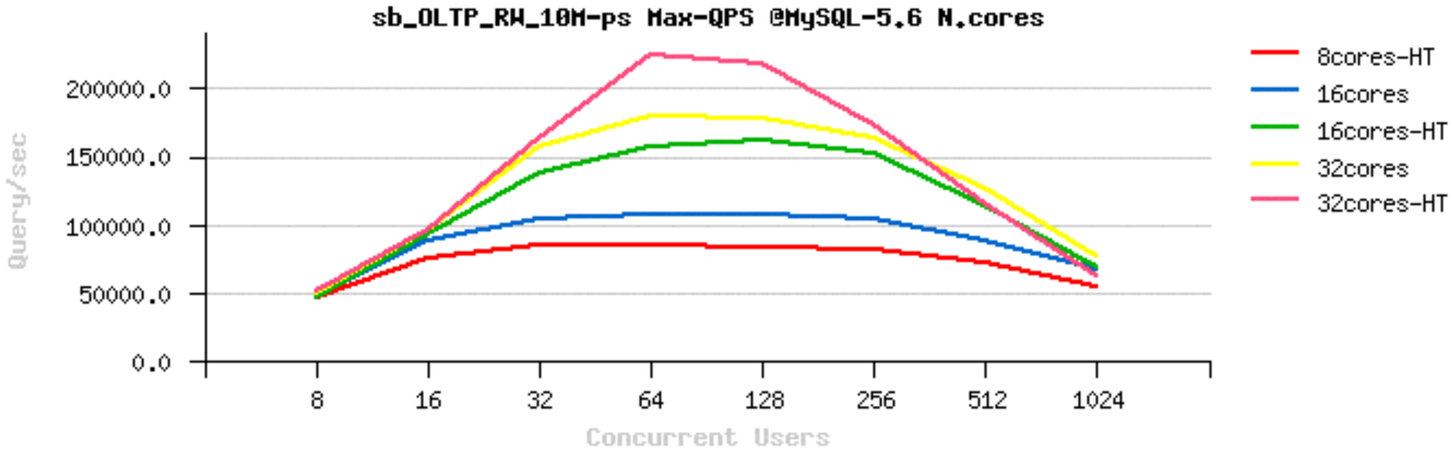




From where we're coming with MySQL 5.7 ?..

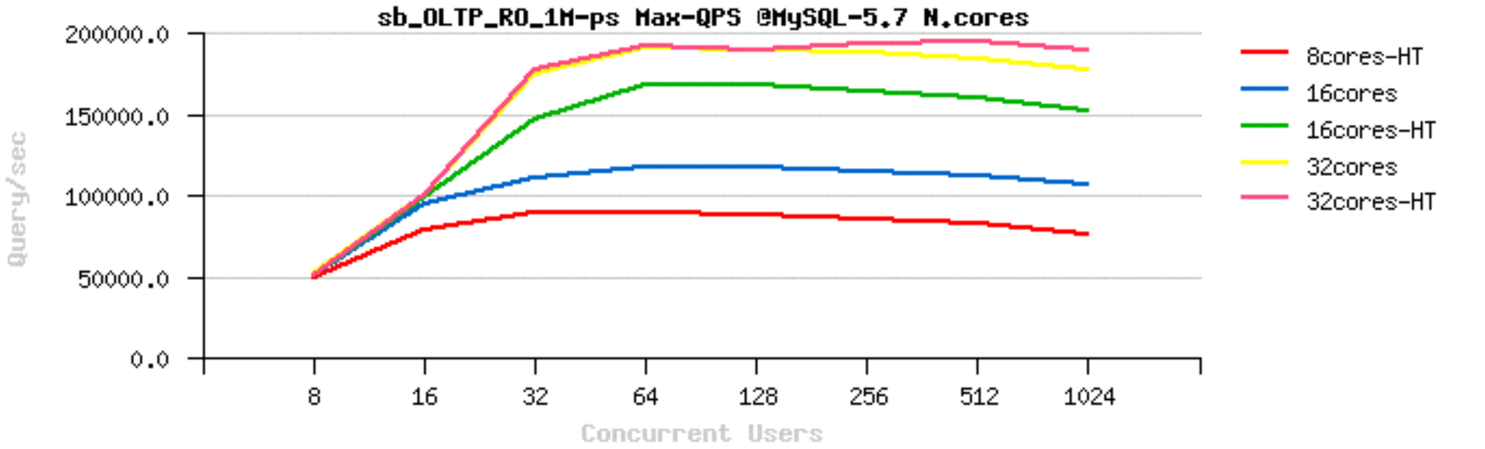
- MySQL 5.6 : RO & RW
  - not lower on 32cores!! ;-)
  - RW out-pass RO !!..??

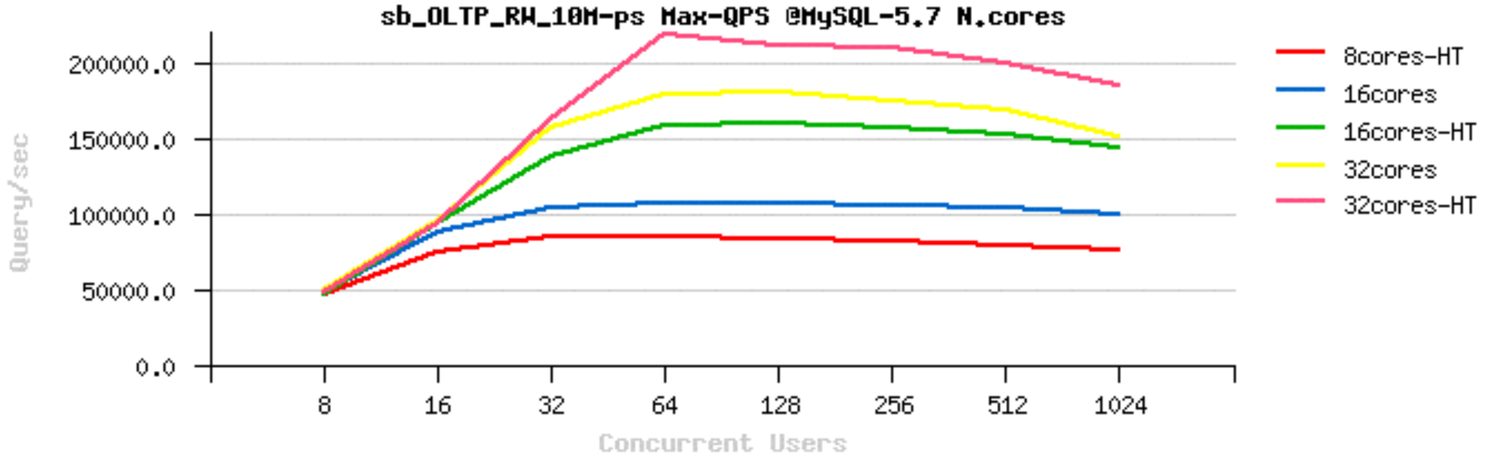




From where we're coming with MySQL 5.7 ?..

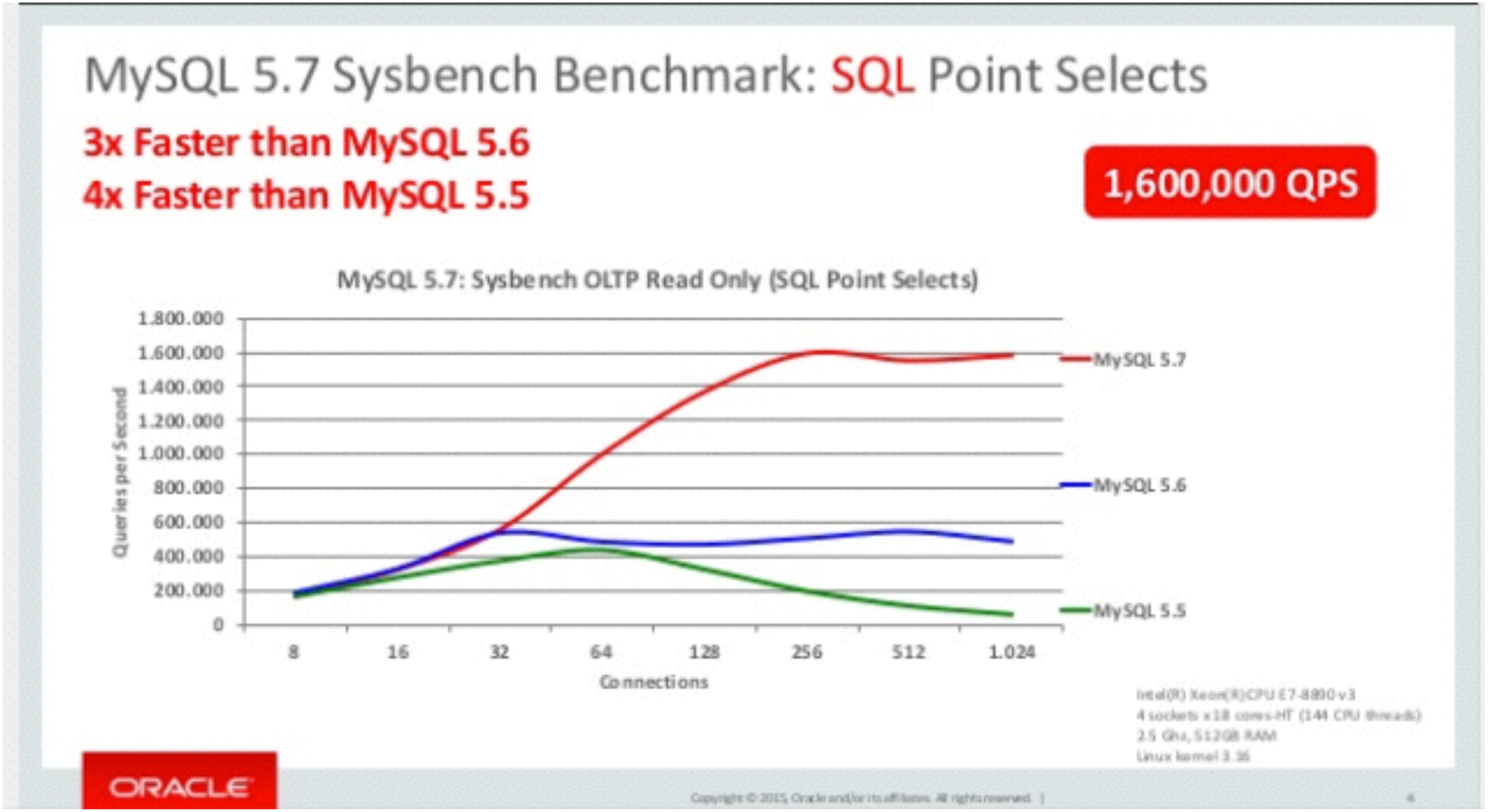
- MySQL 5.7.1 : RO & RW
  - more stable than 5.6
  - **RW** out-pass RO !!..





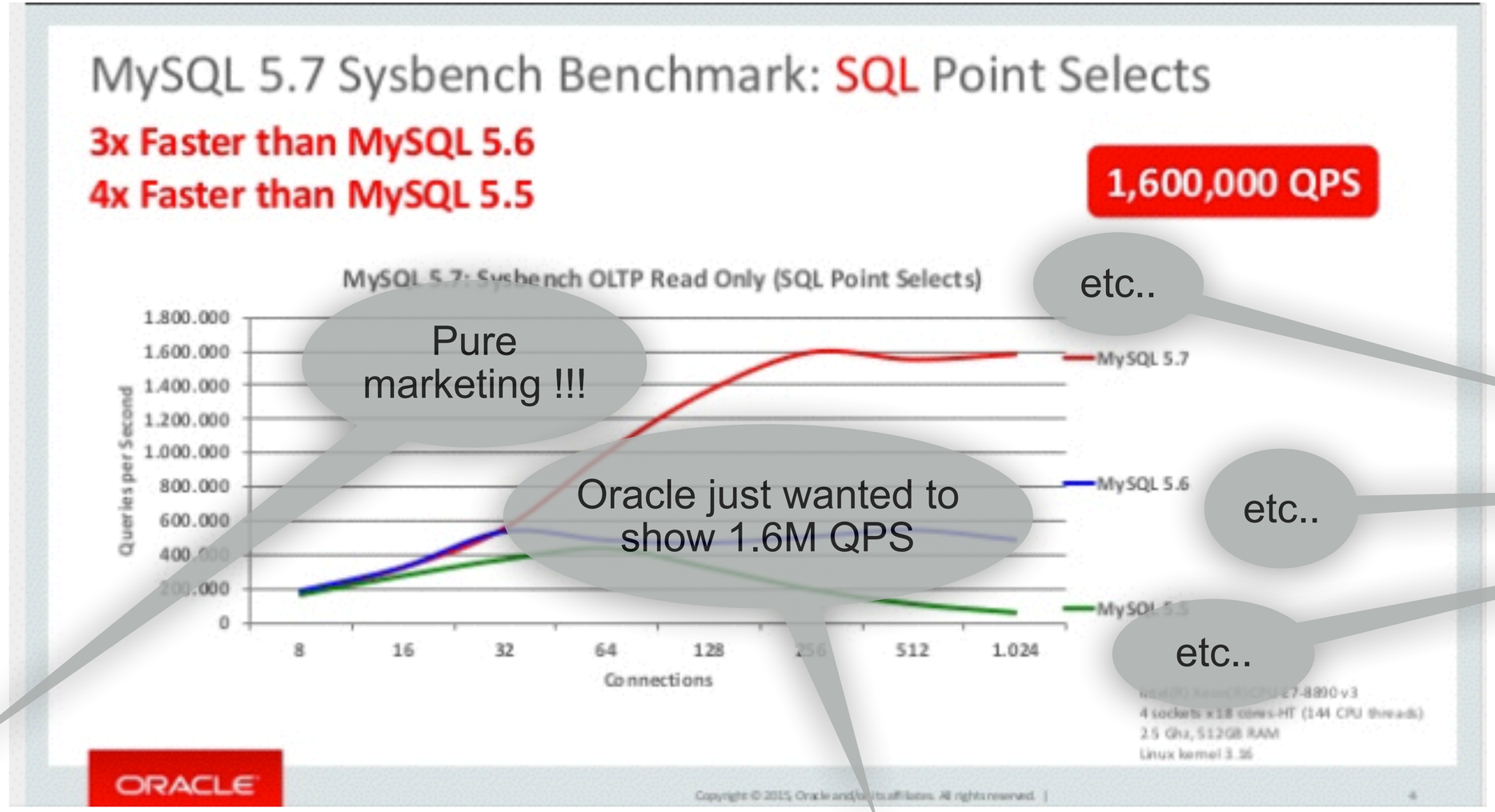
MySQL 5.7 : 1.6M QPS

- What is behind this number ?..



MySQL 5.7 : 1.6M QPS

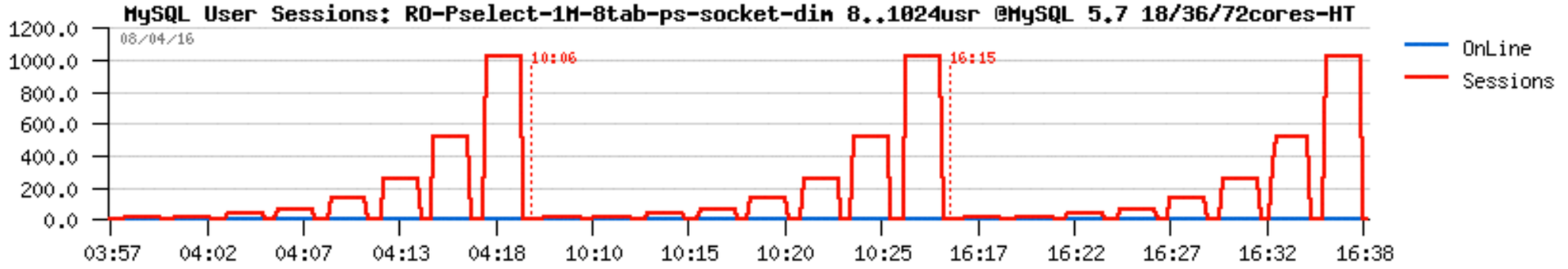
- What is behind this number ?..



Marketing ?..

- Why you did not hear then about **1.8M QPS** ?.. ;-))
  - same 72cores-HT server, same MySQL 5.7
  - we are not running after numbers ;-)
  - numbers are just reflecting what is behind..

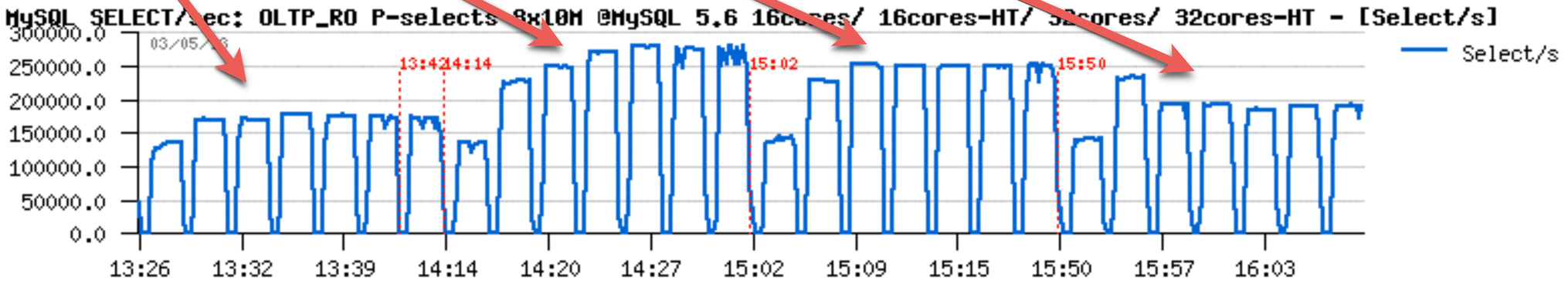
1.8M QPS

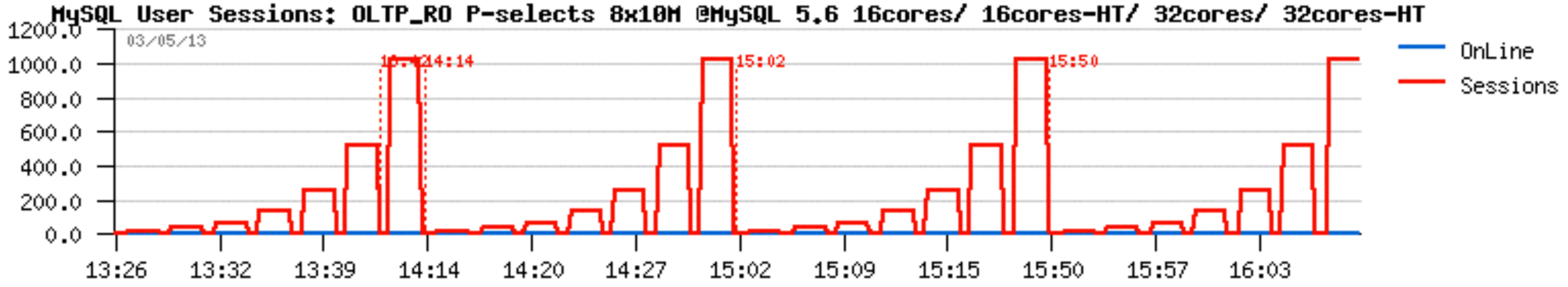


ORACLE

Behind the numbers...

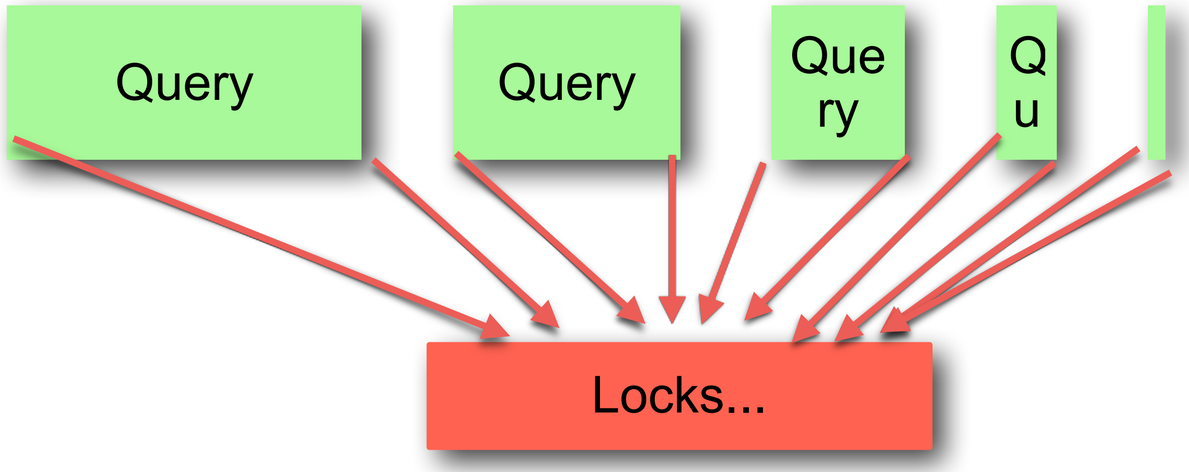
- MySQL 5.6, RO Point-Select Performance
  - 16cores, 16cores-HT, 32cores, 32cores-HT

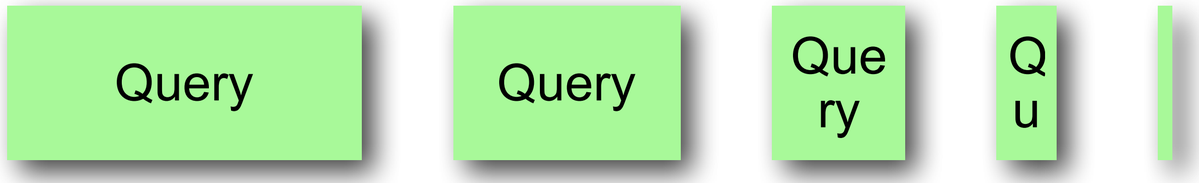




Behind the numbers...

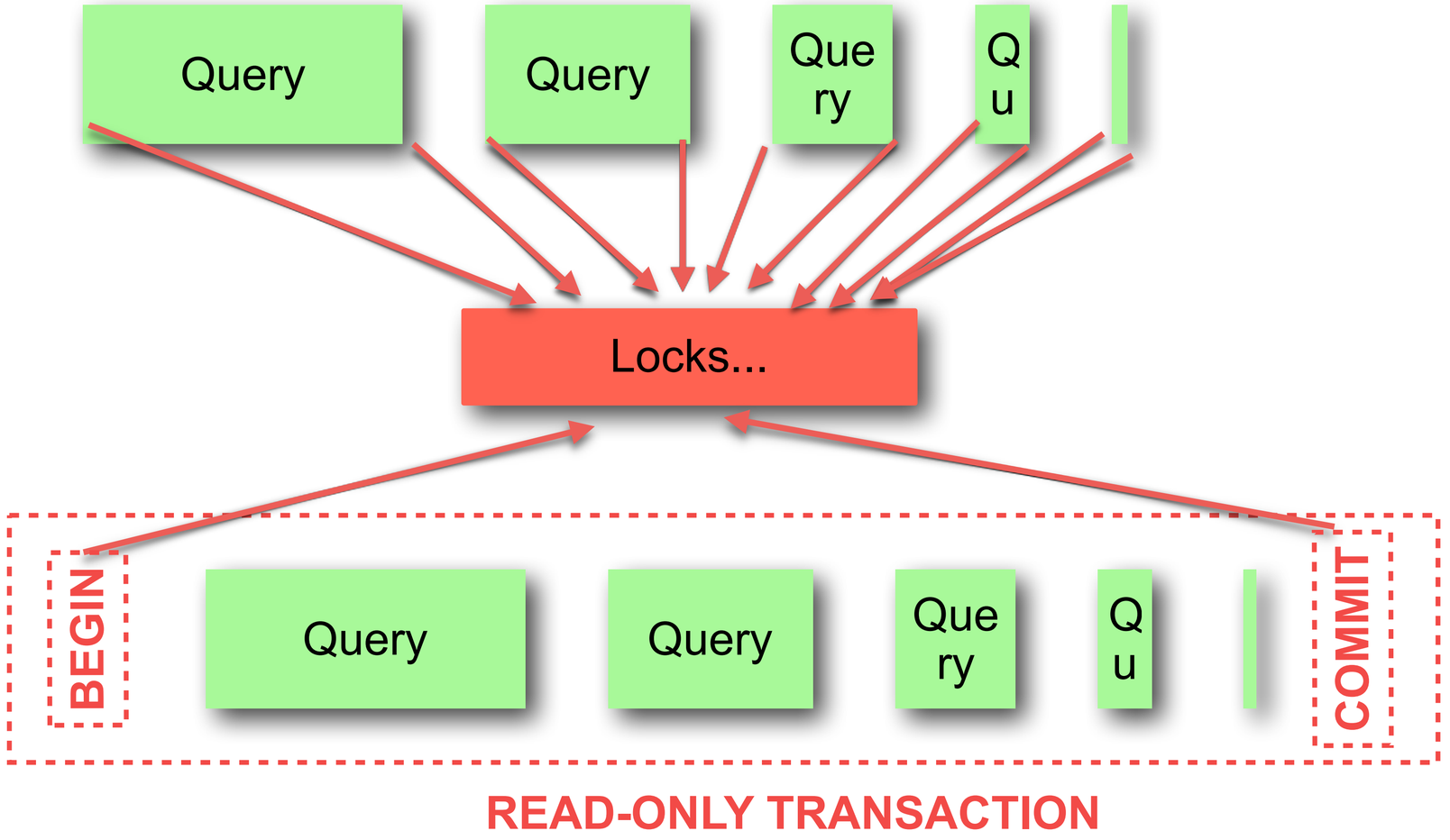
- Why ?..





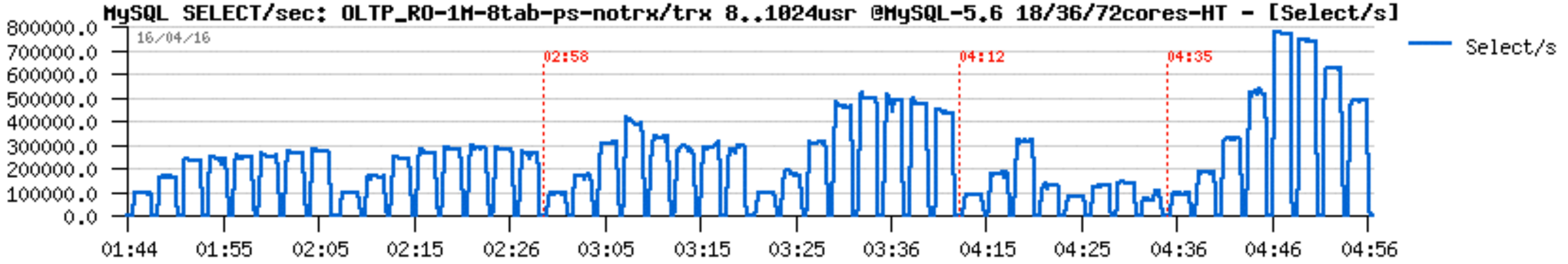
Behind the numbers...

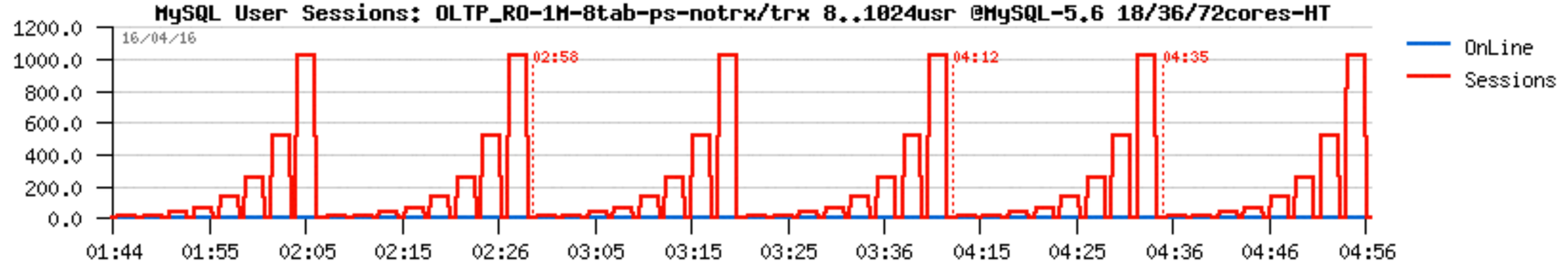
- MySQL 5.6 : Read-Only Transactions "workaround" :



Behind the numbers...

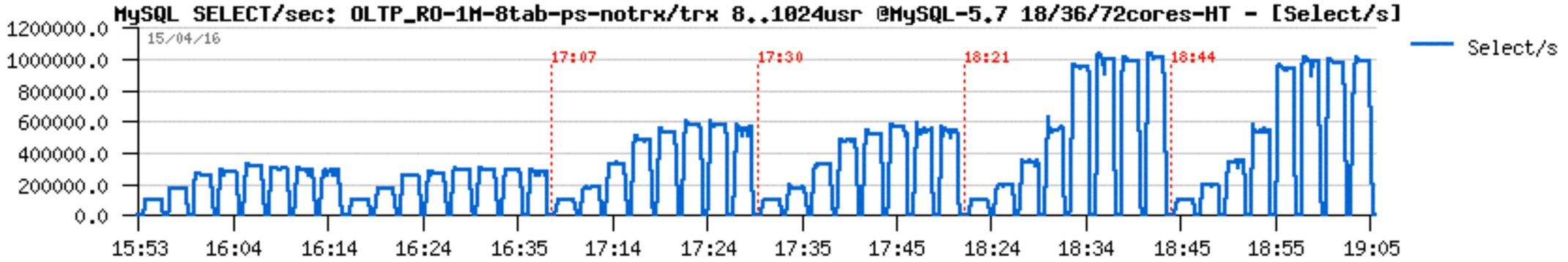
- MySQL 5.6, OLTP_RO-1Mx8-tables, 72cores-HT
  - OLTP_RO : [x14 SELECT Queries]
  - without / with transaction enclosure, 18/36/72cores-HT

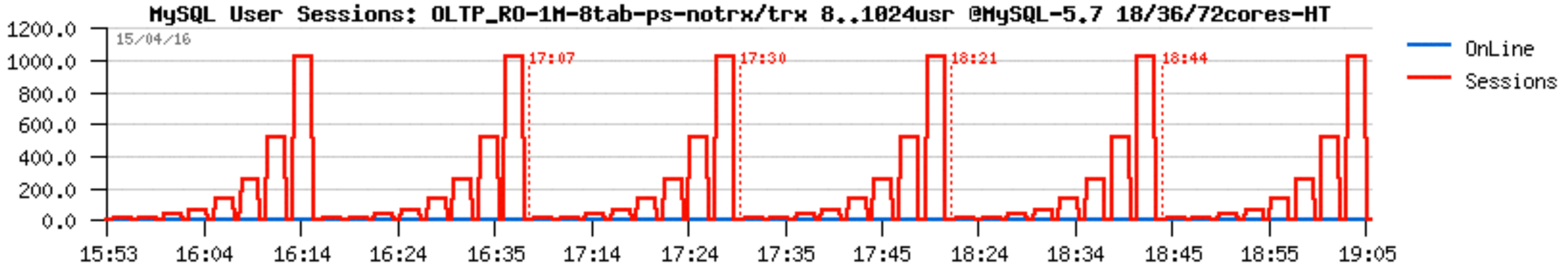




Behind the numbers...

- MySQL 5.7, OLTP_RO-1Mx8-tables, 72cores-HT
  - OLTP_RO : [x14 SELECT Queries]
  - without / with transaction enclosure, 18/36/72cores-HT



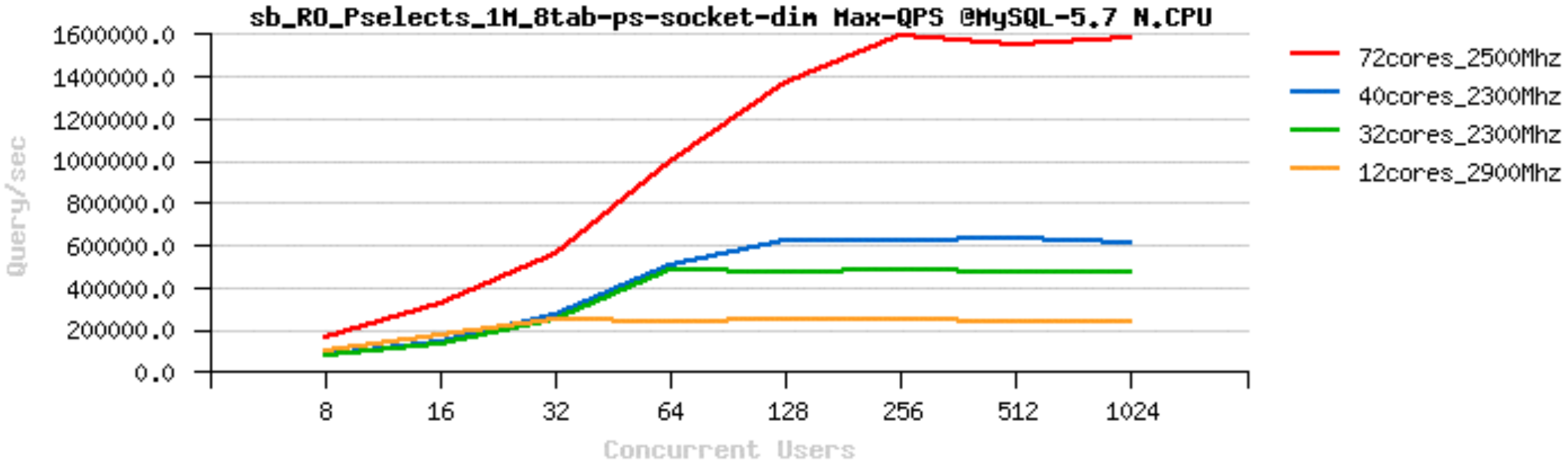


Behind the numbers...

- Up to you to decide what is less or more significant for you..
- If for ex. [x1000(!) Point-Select Queries] in transaction is OK
  - as was done by MariaDB to show their 1M QPS result..
  - hm.. and nobody called this Marketing ?..

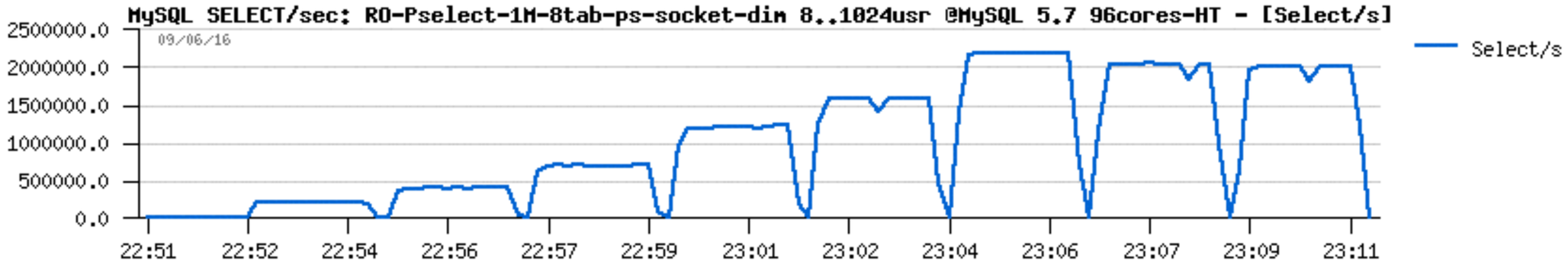
RO Point-Selects @MySQL 5.7

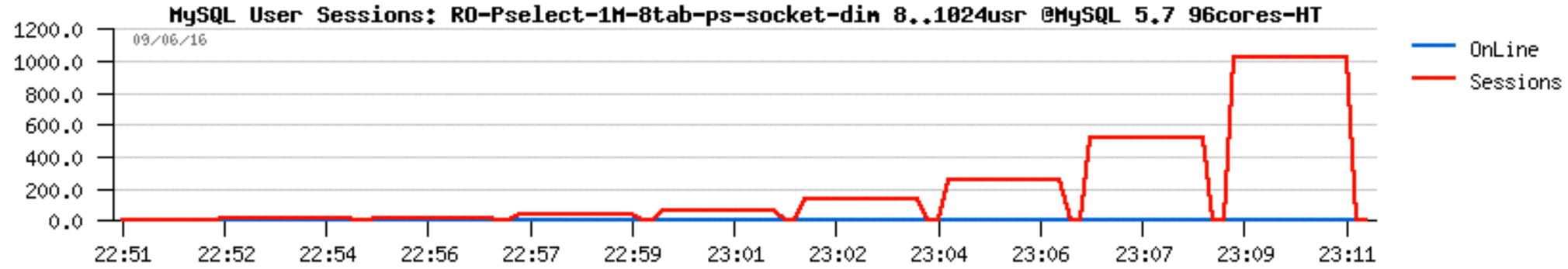
- Sysbench Point-Selects 8-tab => HW Progress :
  - new Intel CPU chips rock! (on 72cores-HT server here)



RO Point-Selects @MySQL 5.7 (One more thing..)

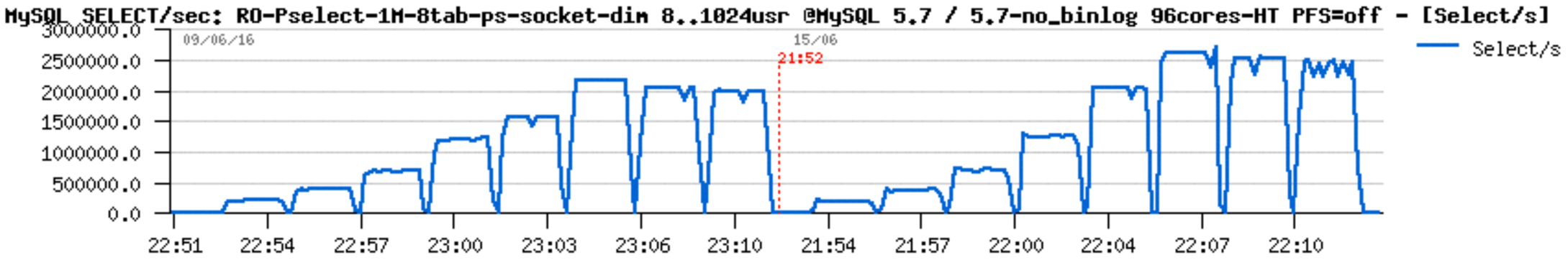
- Sysbench Point-Selects 8-tab => HW Progress : **2.2M QPS**
  - new Intel CPU chips rock! (the same 72cores-HT server upgraded to **96cores**-HT)

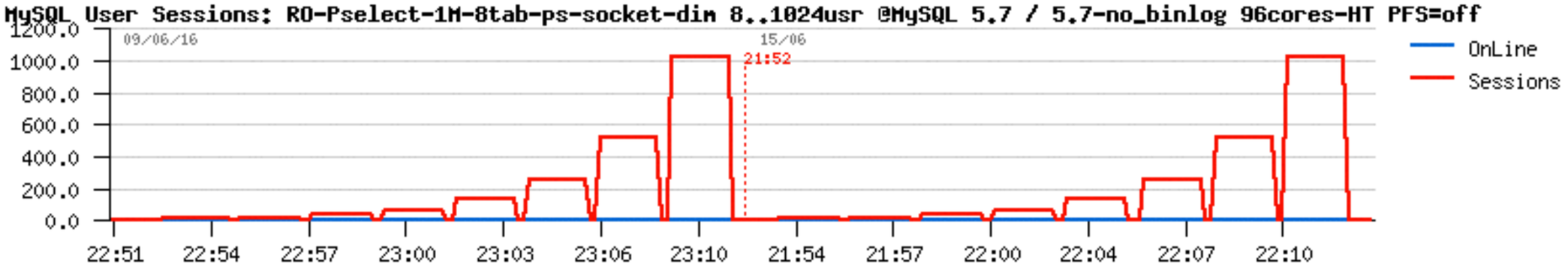




RO Point-Selects @MySQL 5.7 (One more thing..)

- Sysbench Point-Selects 8-tab => HW Progress : **2.5M QPS ;-))**
  - the same upgraded to **96cores**-HT server + some tweaking..





RO Pending Issues…

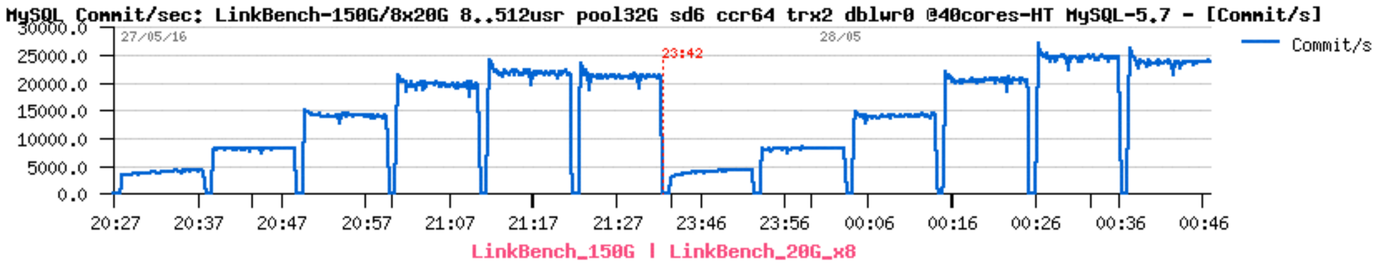
- PK vs Secondary Indexes lookup
  - lookups via PK are faster than via Secondary Indexes
  - sometimes x10 times faster (or even more)..
  - directly depending on the amount of rows to read..
  - AHI is helping here (but not solving)
  - “covering” indexes are solving, but needing more space + impacting writes
- InnoDB Adaptive Hash Index (AHI)
  - implemented with a global RW-lock
  - InnoDB RW-locks are not scaling by design (CPU cache syncs)
  - using table partitions helps to split indexes
  - using AHI partitions (5.7) helps to split RW-locks (coop. with Percona)
    - NOTE: and this is creating **20% regression** on DBT3 benchmark (single-thread)..
    - just to mention how the code is sensible today ;-))
  - yet far from fixed..

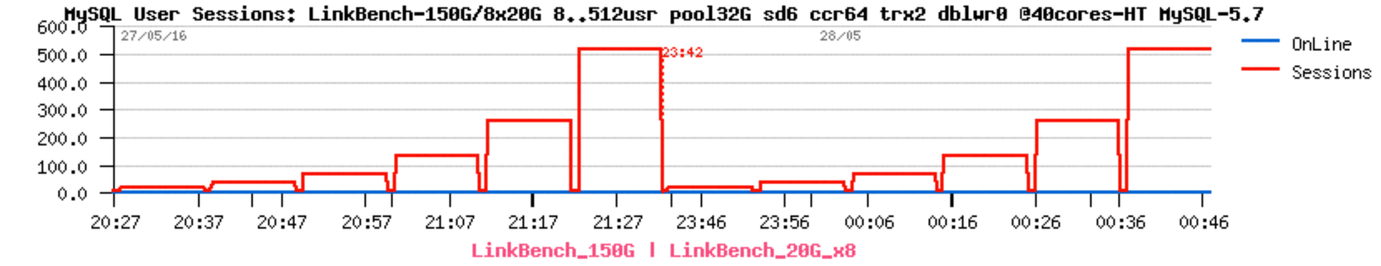
RO Pending Issues…

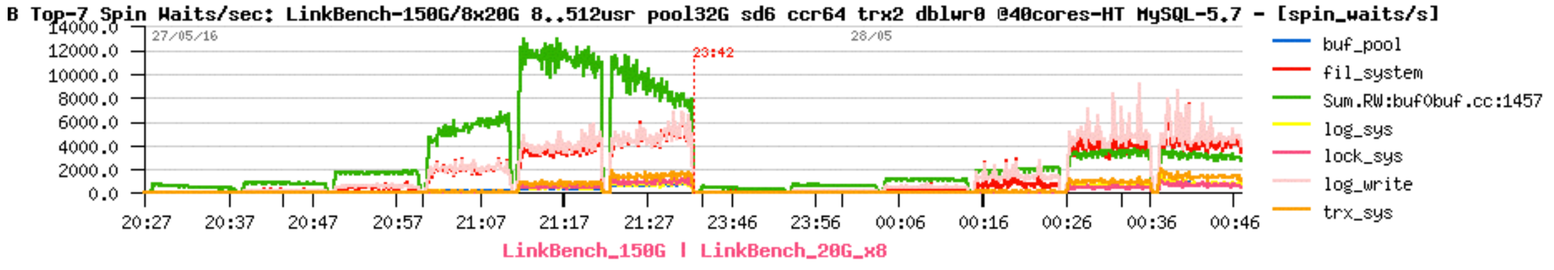
- InnoDB Block Locks
  - seen when the same pages are accessed concurrently..
  - how to see : “show mutex” is back ;-)
  - **workarounds :**
    - avoid such an access pattern, don’t do this ;-)
    - use a smart query cache (like ProxySQL), or row cache (memcached, etc.)..
  - expected to be fixed in v.xx.0 ..
    - requires a full page re-design..
    - so, nothing yet promised.. ;-)
  - **NOTE :**
    - impacts not only RO, but also RW ..

LinkBench-150GB

- IO-bound (150GB dataset with BP=32GB)
  - single 150GB database -vs- 8 x 20GB databases





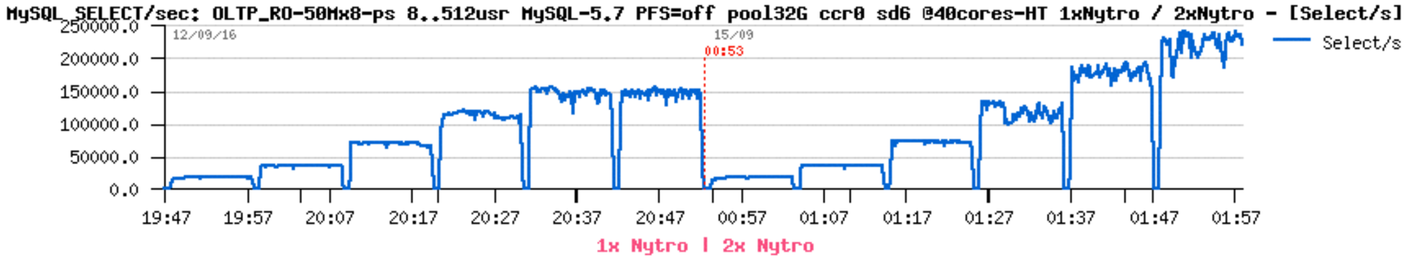


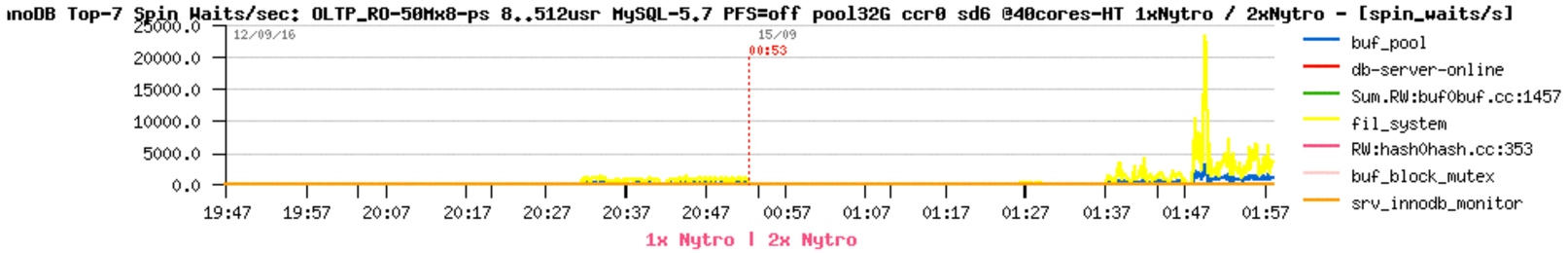
Read-Only : IO-bound @MySQL

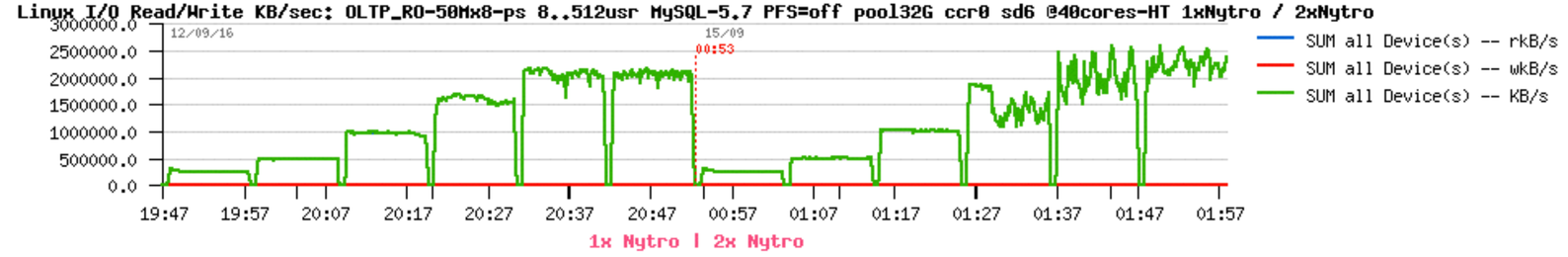
- 5.5 : hmm..
- 5.6 / 5.7 :
  - LRU driven : just page eviction, see METRICS stats
  - HDD : limited by your I/O layer..
  - SDD : limited by your I/O layer..
  - Really Fast Flash (LSI, Fusion-io, Seagate, etc.) :
    - avg load : follow I/O performance
    - high load: fil_sys mutex contention + kernel / FS locks !
  - also consider : innodb_old_blocks_time & innodb_old_blocks_pct
- 5.7 :
  - excessive page scan is fixed

Read-Only IO-bound @MySQL 5.7

- OLTP_RO 8x50M tables BP=32GB Seagate Flash “Nytro"
  - 1x flash card -vs- 2x flash cards

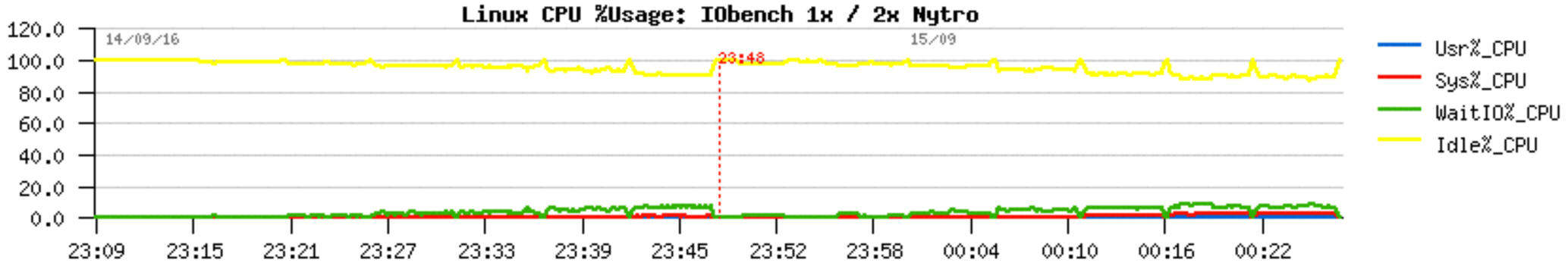


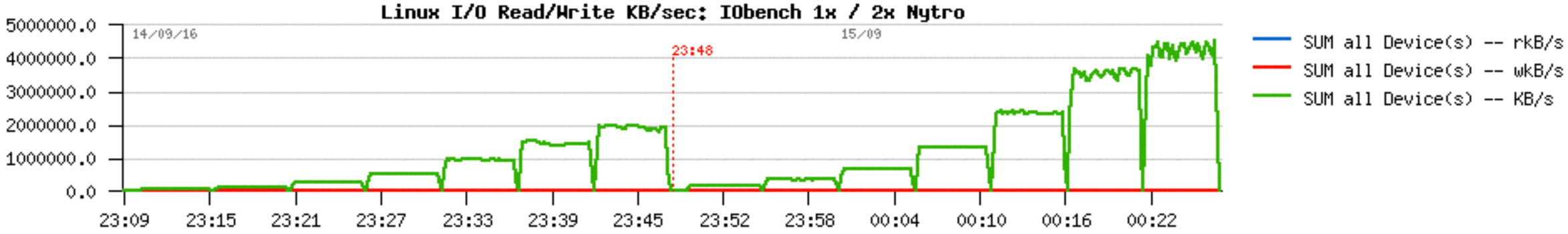




IO Reads Raw@Seagate Flash “Nytro" 1x -vs- 2x

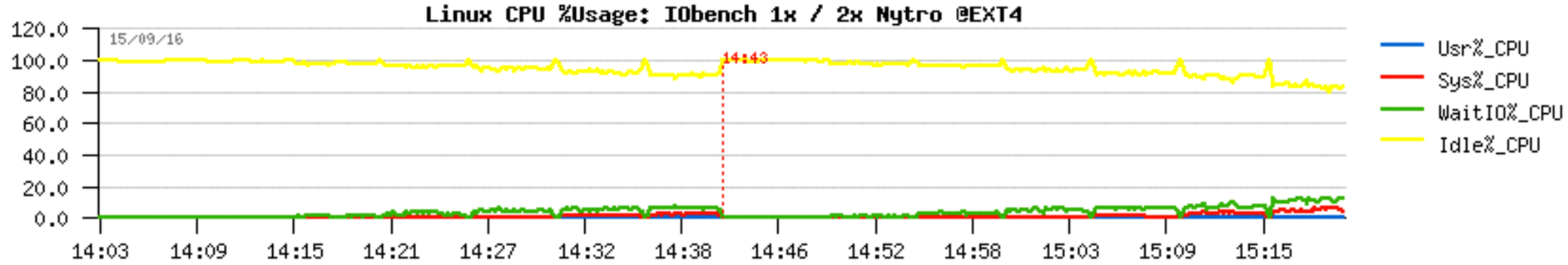
- IObench 16K Random-Reads from 128GB dataset
  - concurrency : 1,2,4 .. 64

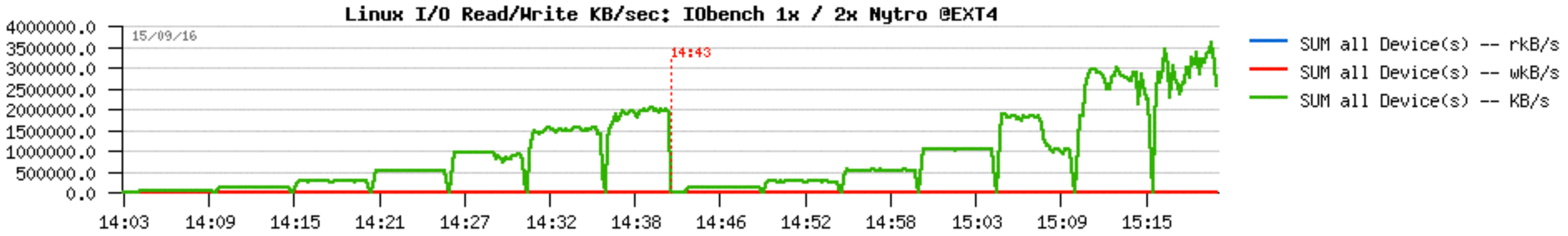




IO Reads EXT4@Seagate Flash “Nytro" 1x -vs- 2x

- IObench 16K Random-Reads from 128GB file(s)
  - concurrency : 1,2,4 .. 64





Hope it was not boring yet ?.. ;-)

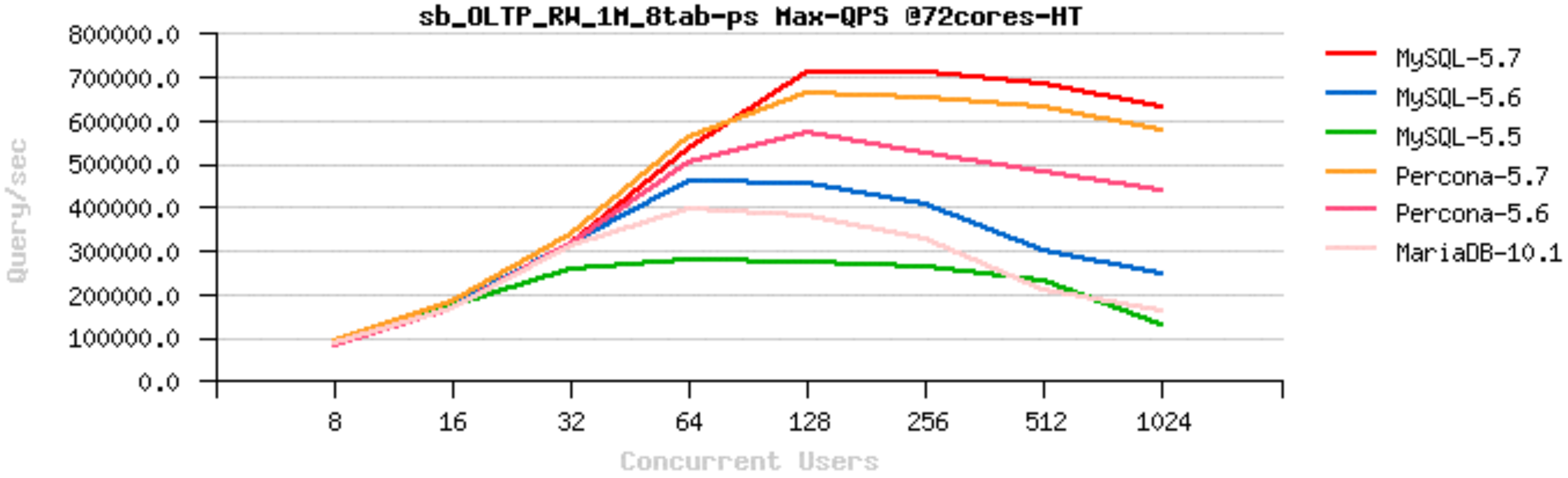
- Because I'll need all your full attention since now! ;-)

Read+Write (RW) Workloads Scalability @MySQL 5.7

- Huge progress is already here too!
  - improved index locking
  - reduced lock_sys mutex contention
  - parallel flushing + improved flushing design
  - much better observability of internals
  - etc..
- However, not yet as good as Read-Only..
  - Performance continues to increase with more CPU cores
  - But on move from 16 to 32cores-HT you may gain only 50% better
  - Better performance on a faster storage as well
  - **On OLTP_RW can use a full power of fast flash for today! (sometimes)**
  - **More work in progress ;-)**
  - Internal contentions & Design limitations are the main issues here..
  - still many things are in pipe & prototype..

OLTP_RW : 8-tables

- Sysbench OLTP_RW 1Mx8-tables
  - 72cores-HT
  - and the winner is: MySQL 5.7 !! (or Percona-5.7 + patch ;-))



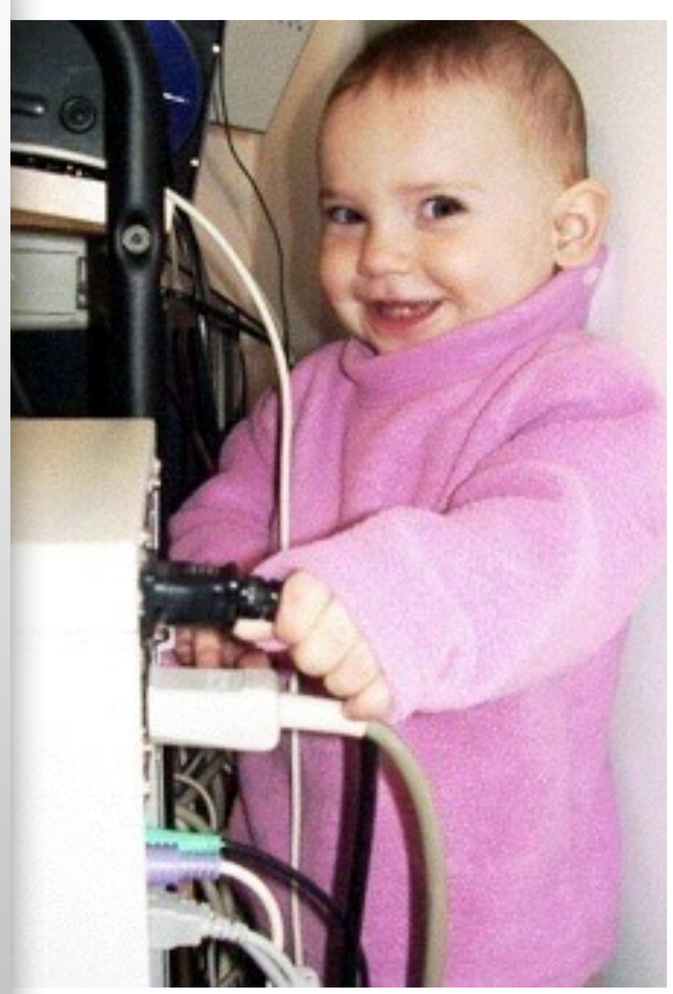
Read+Write Performance @MySQL / InnoDB

- Transactional processing
  - your CPU-bound transactional processing defines your Max possible TPS
  - with a bigger volume / more IO / etc. => Max TPS will not increase ;-)
- Pending issues :
  - same as RO + REDO (log_sys), locks (lock_sys), TRX (trx_sys), AHI=off, etc..
  - Purge lagging, more improved Adaptive Flushing
- Data Safety
  - binlog : overhead + bottleneck (be sure you have binlog group commit)
  - InnoDB checksums : overhead (reasonable since crc32 is used)
  - innodb_flush_log_at_trx_commit = 1 : overhead + bottleneck
  - **InnoDB double write buffer : KILLER ! overhead + huge bottleneck..**
    - Fusion-io atomic writes is one of (**true** support in MySQL 5.7)
    - **good news : the improved & more advanced DBLWR is coming for 5.7 & 8.0 ;-))**

RW related starter configuration settings

- my.conf :

```
innodb_file_per_table
innodb_log_file_size=1024M
innodb_log_files_in_group=3 / 12 / ...
innodb_checksum_algorithm= none / crc32
innodb_doublewrite= 0 / 1
innodb_flush_log_at_trx_commit= 2 / 1
innodb_flush_method=O_DIRECT_NO_FSYNC
innodb_use_native_aio=1
innodb_adaptive_hash_index=0

innodb_adaptive_flushing = 1
innodb_flush_neighbors = 0
innodb_read_io_threads = 16
innodb_write_io_threads = 16
innodb_io_capacity=15000
innodb_max_dirty_pages_pct=90
innodb_max_dirty_pages_pct_lwm=10
innodb_lru_scan_depth=4000
innodb_page_cleaners=4

innodb_purge_threads=4
innodb_max_purge_lag_delay=30000000
innodb_max_purge_lag= 0 / 1000000

binlog ??
```



RW Workloads : Checklist..

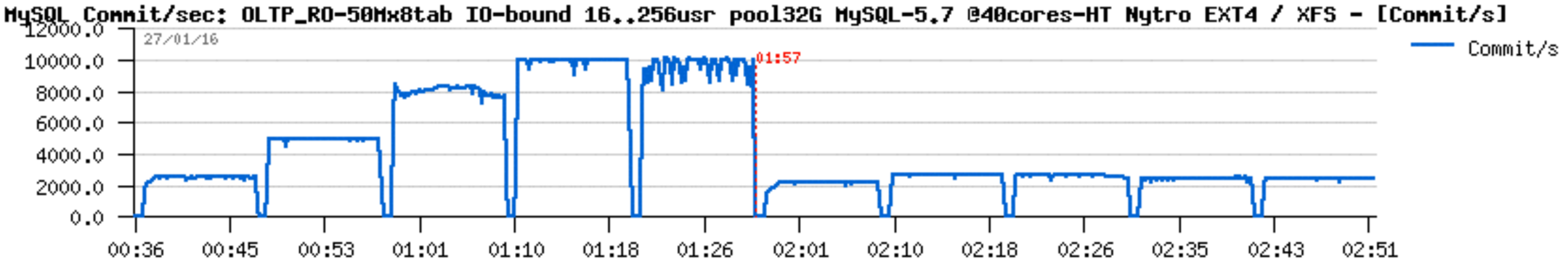
- To keep in mind :
  - FS choice
  - Flushing tuning
  - Double Write (DBLWR)
  - Purge
  - Massive INSERTs
  - UPDATE performance
  - etc..

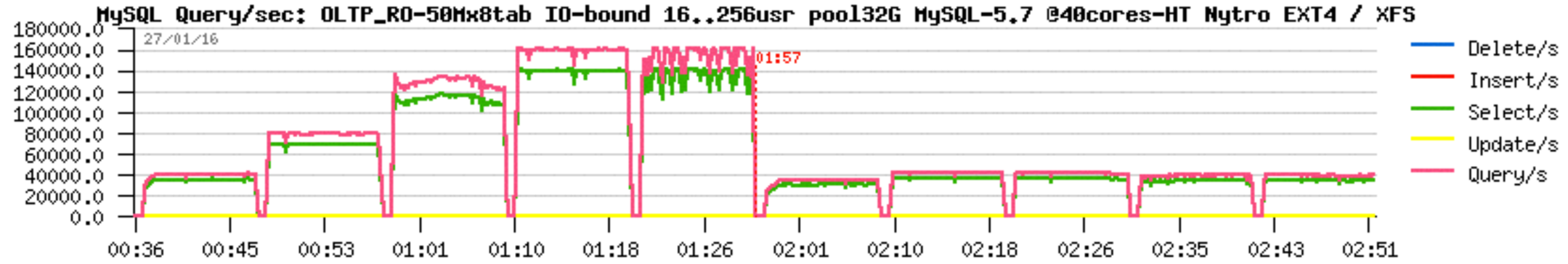
RW Workloads : Test your Filesystem before to deploy

- Do first a pure IO test to know your FS + Storage limits
  - RO first (random 16K IO reads)
  - RW next
  - compare several FS to be sure you did not miss something ;-)
- Then run on them MySQL workload
  - same, RO first
  - RW next
  - do your own conclusions..
  - **NOTE :** it's not because that something is good for others that the same thing will be good for you.. — **test yourself !!!**

RW IO-Bound : Test your Filesystem before to deploy

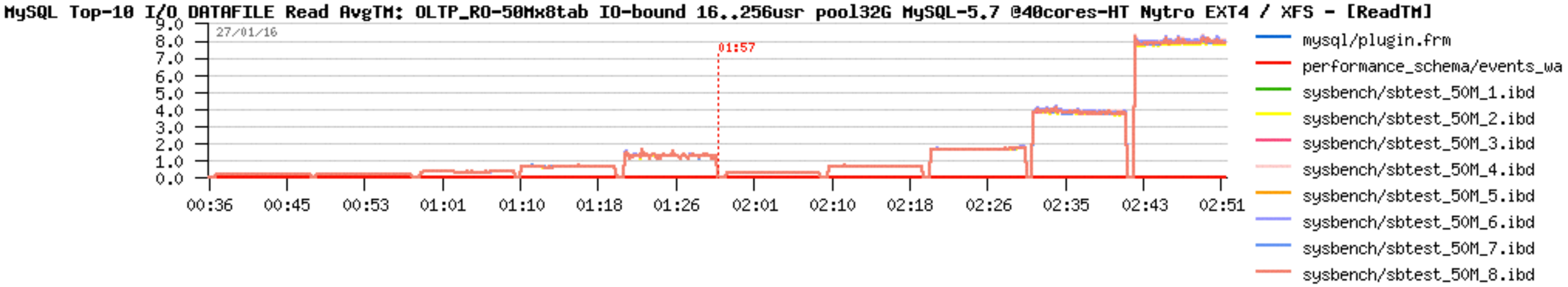
- EXT4 vs XFS
  - OLTP_RW 50M x8-tab OL6.5 @ 40cores-HT, pool 32GB, trx2 :

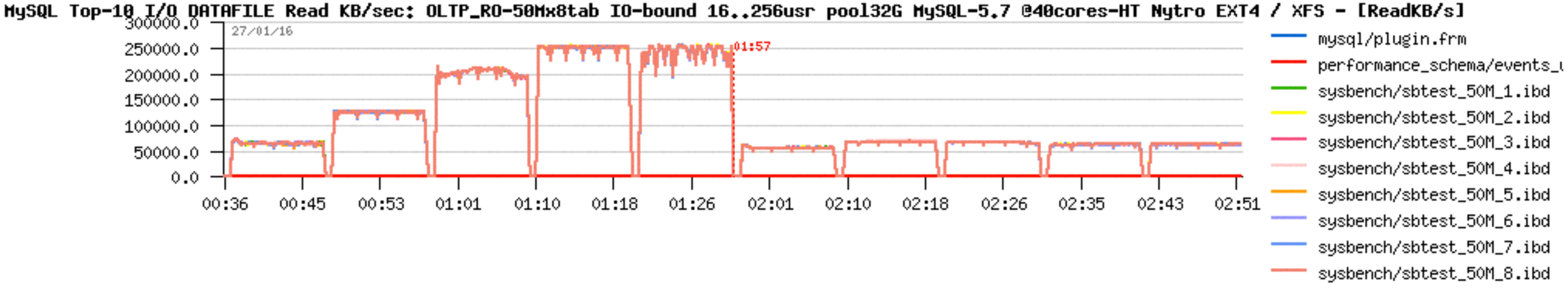




RW IO-Bound : Test your Filesystem before to deploy

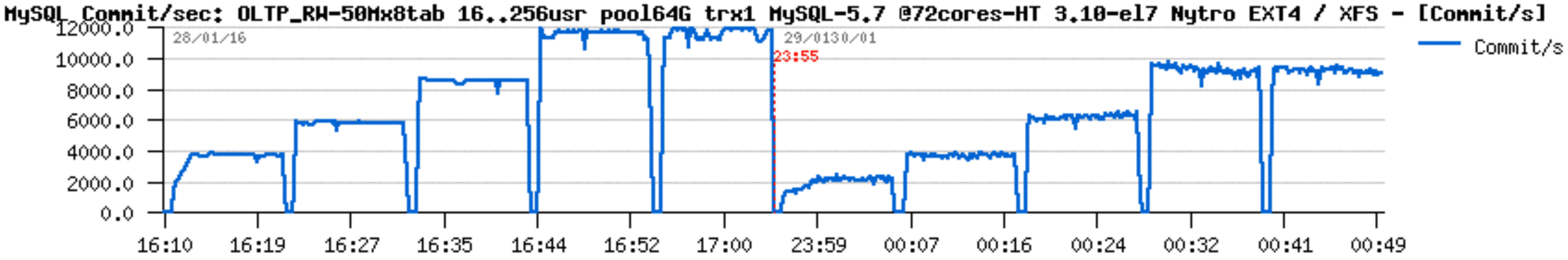
- EXT4 vs XFS
  - OLTP_RW 50M x8-tab OL6.5 @ 40cores-HT, pool 32GB, trx2 :

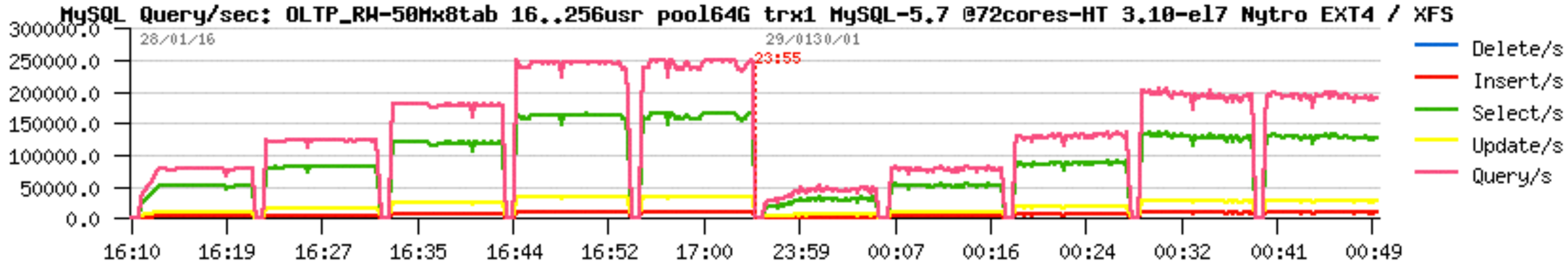




RW IO-Bound : Test your Filesystem before to deploy

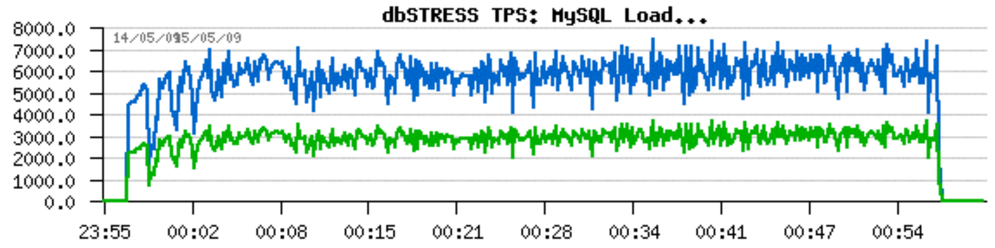
- EXT4 vs XFS
  - OLTP_RW 50M x8-tab OL7.2 @ 72cores-HT, pool 64GB, trx1 :

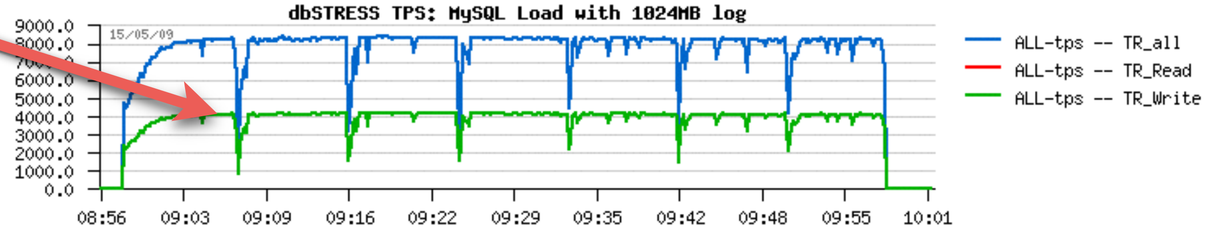


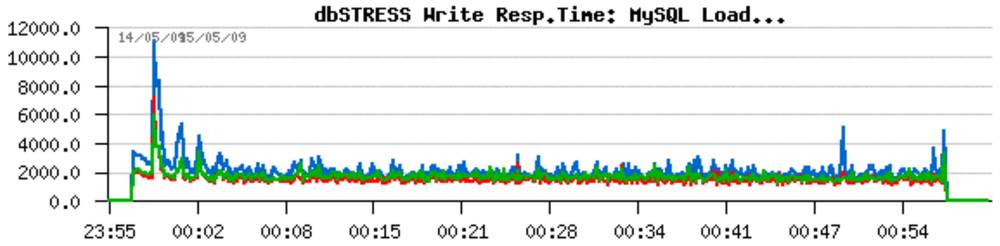


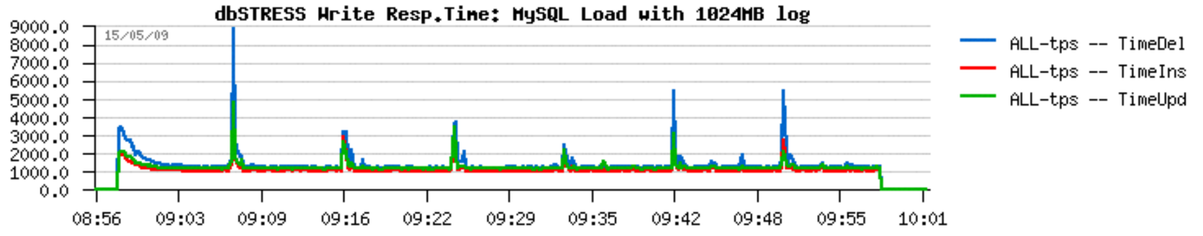
Read+Write Workloads : InnoDB REDO size

- REDO log size impact in the past (pre-MySQL-5.5) :
  - 128M vs 1024M
  - 6000 TPS => 8000 TPS and more stable
  - 2ms resp. time => 1ms
  - Why periodic TPS drops ?.. <= purge in Master Thread.. - fix: dedicated Purge Thread





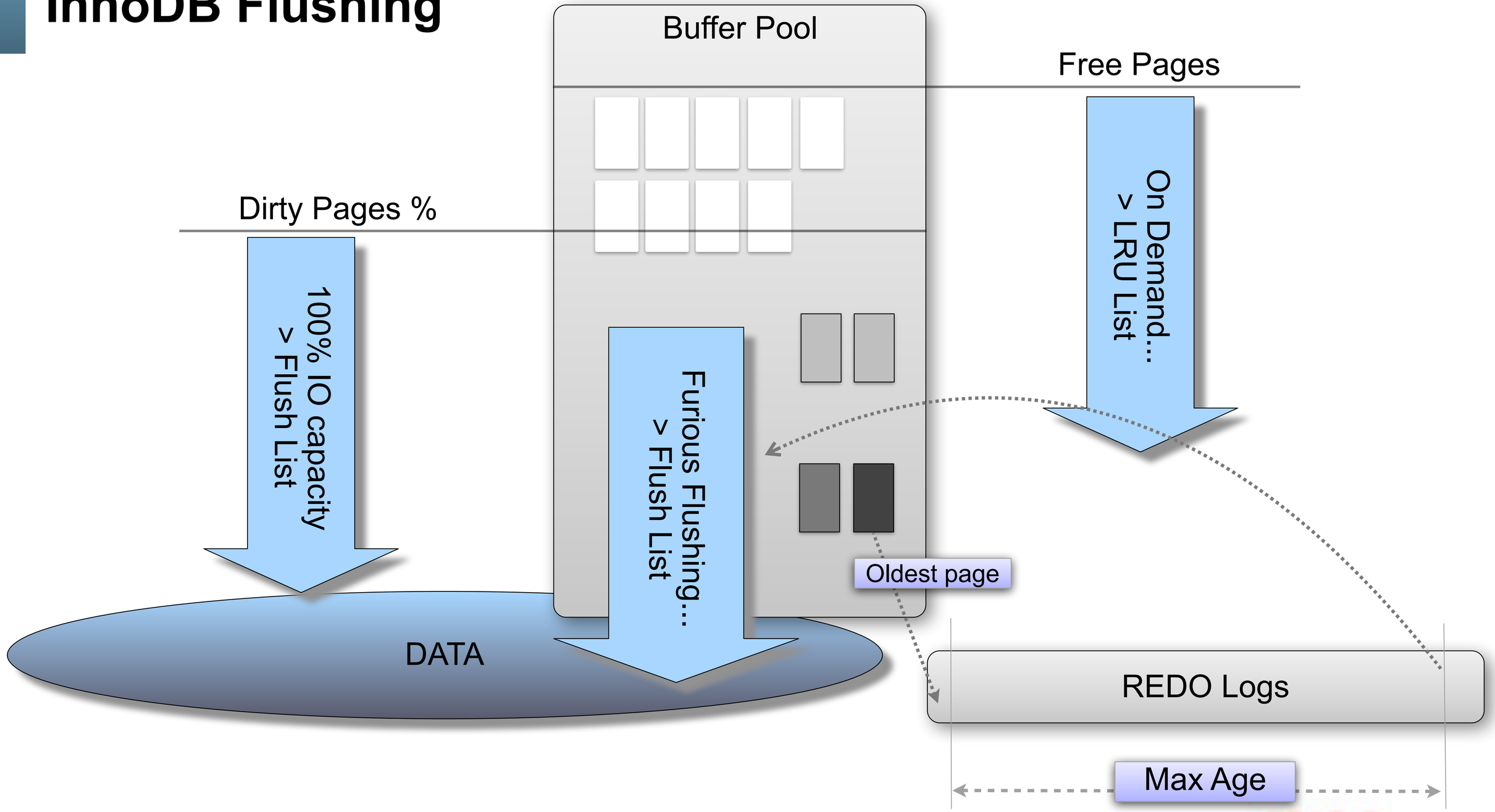




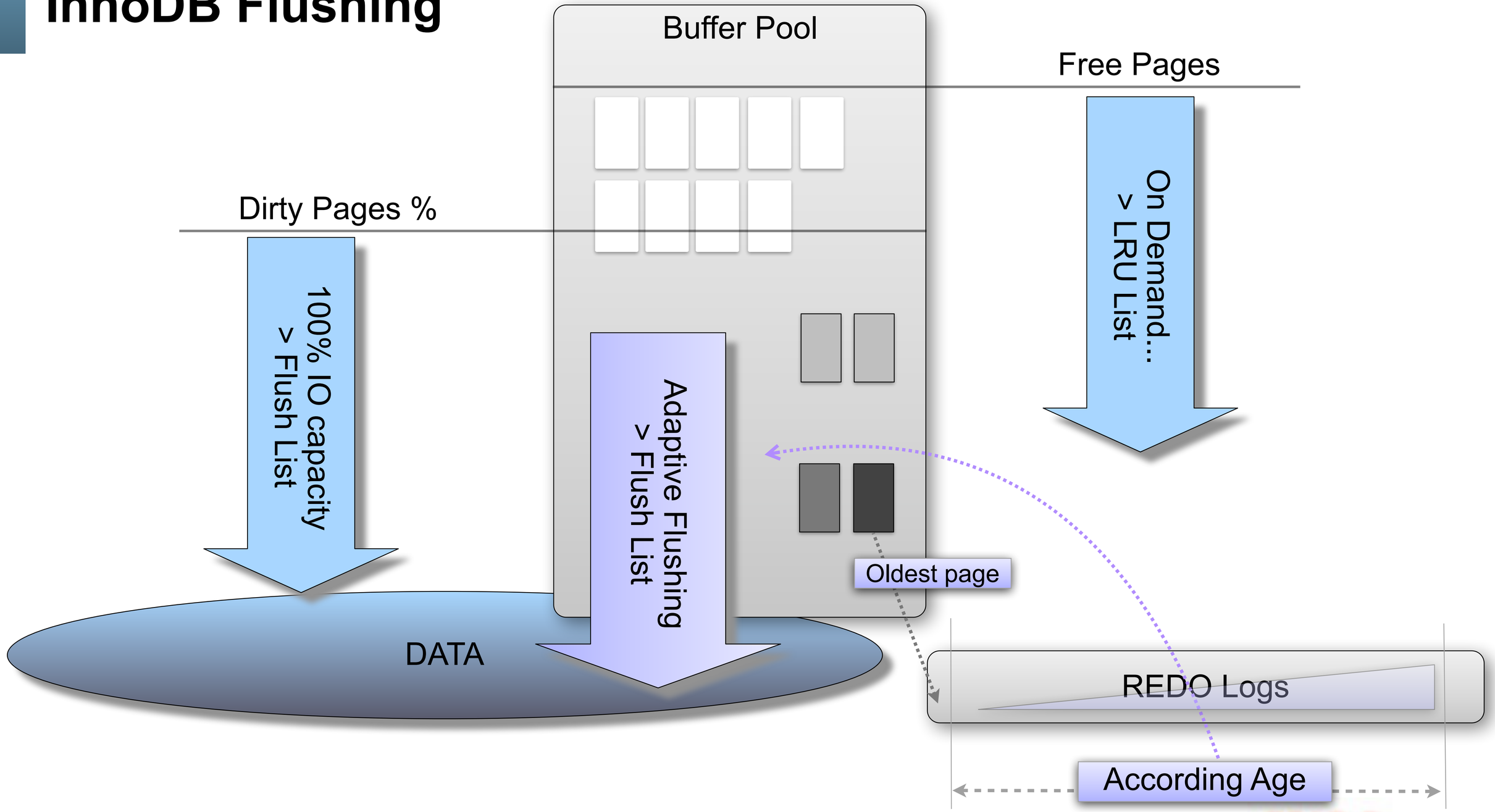
Read+Write Workloads : InnoDB Flushing

- InnoDB Flushing...
  - 5.5 : no comments.. ;-)
    - io capacity !!
  - 5.6 :
    - Improved Adaptive Flushing (step 1)
    - Cleaner Thread
    - io capacity max !!
    - LRU depth !!
  - 5.7 :
    - multiple Cleaner Threads
    - improved LRU flushing
    - improved Adaptive Flushing Design (step 2)

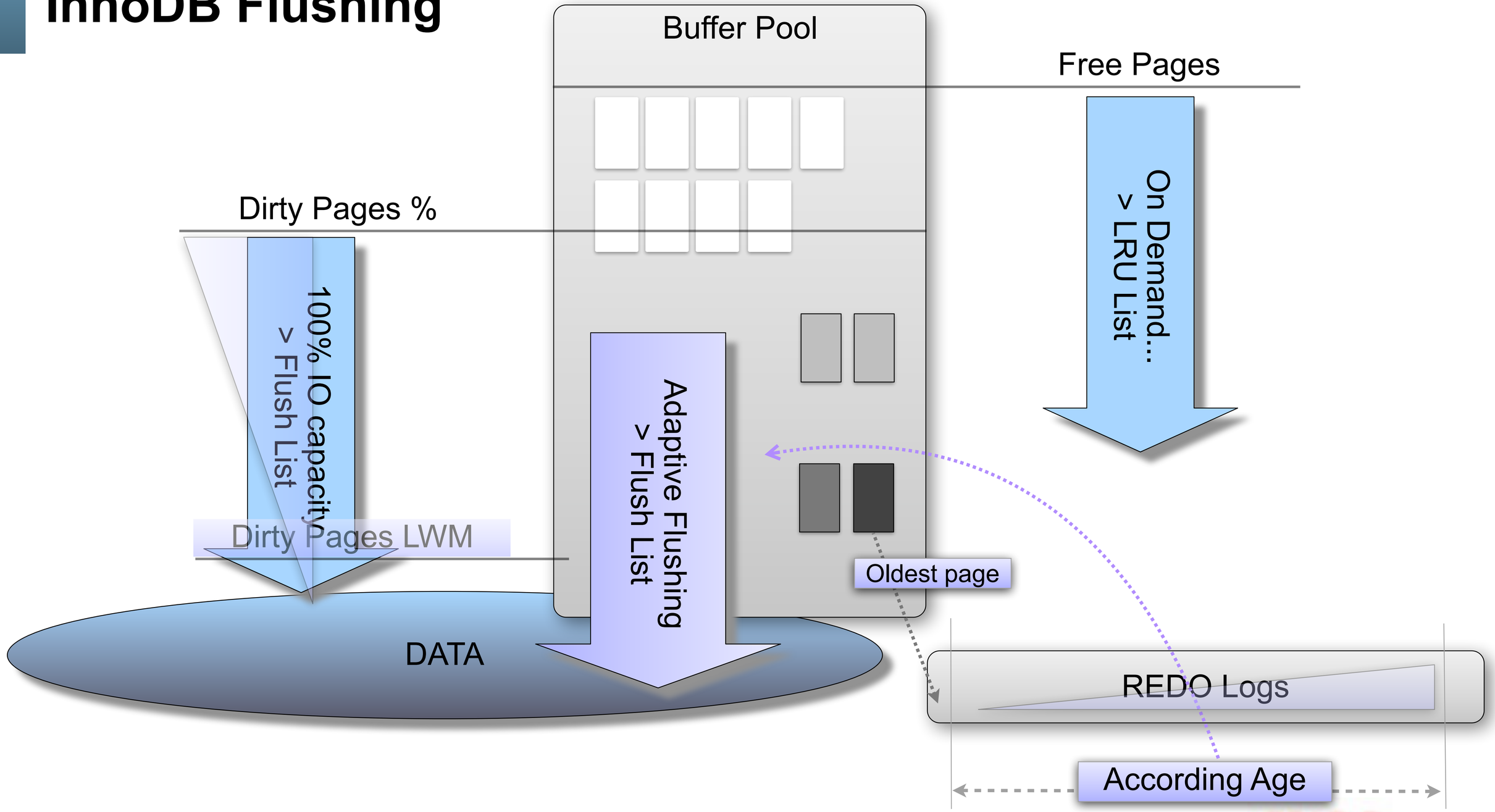
InnoDB Flushing



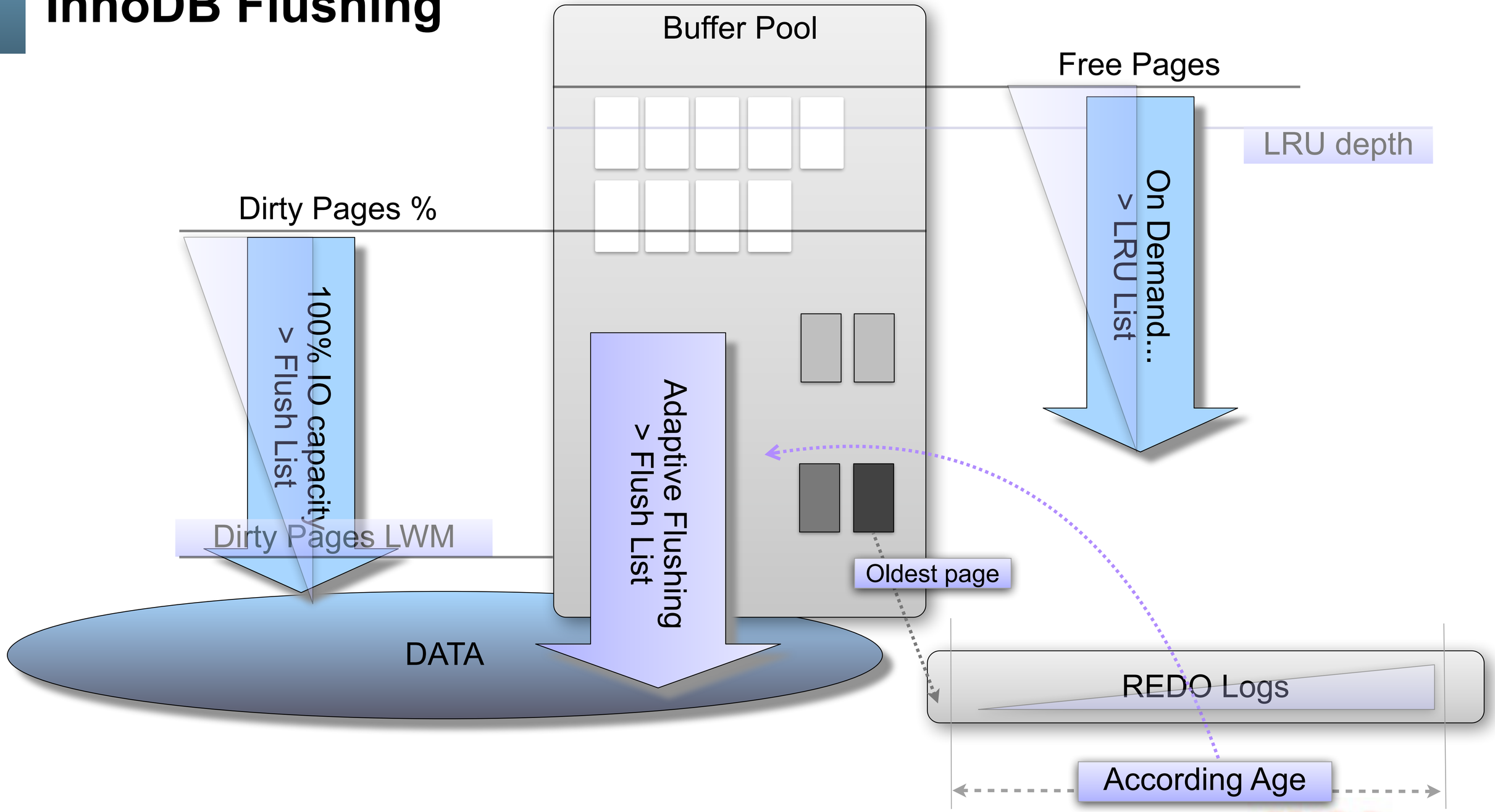
InnoDB Flushing



InnoDB Flushing

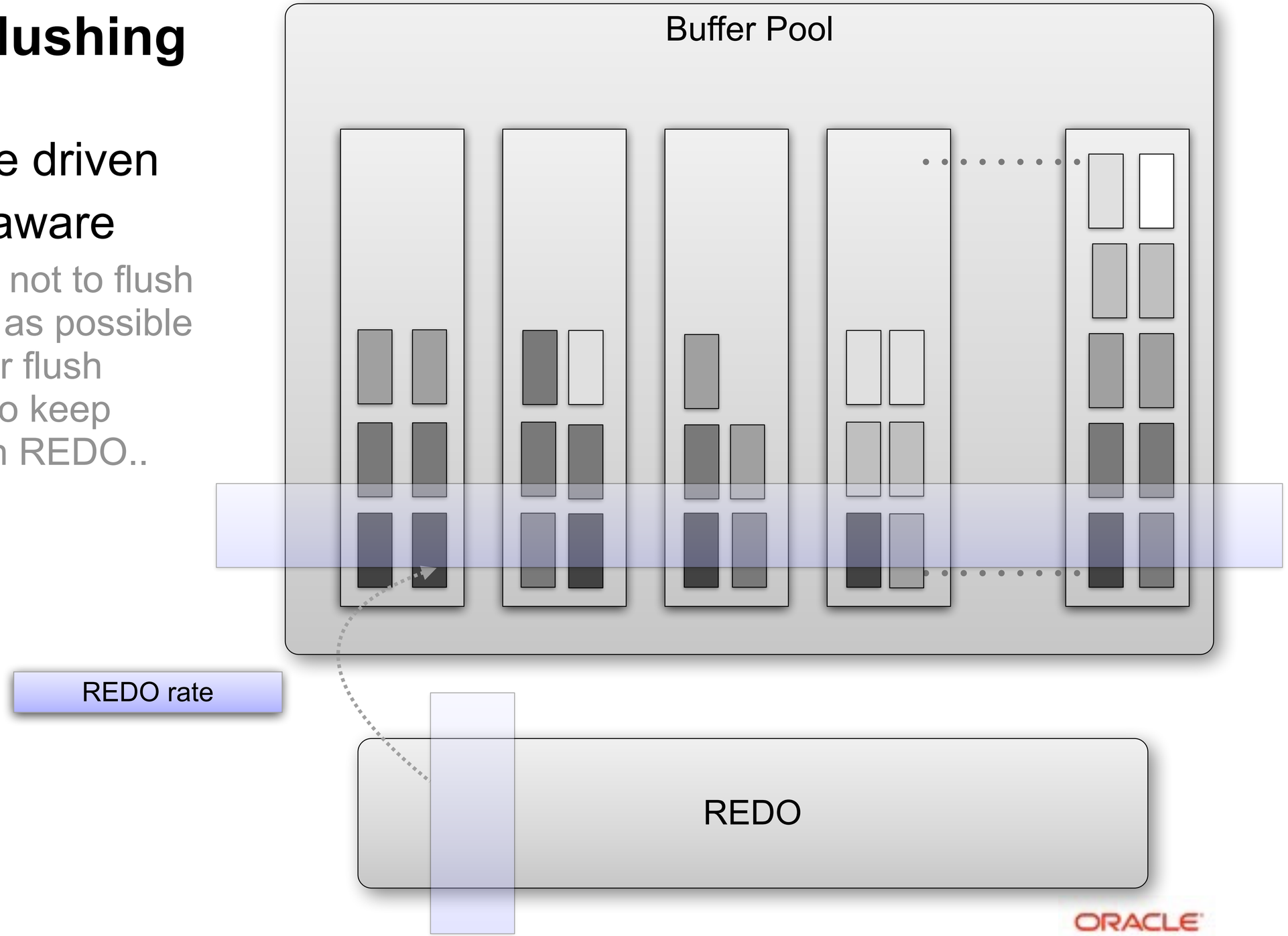


InnoDB Flushing



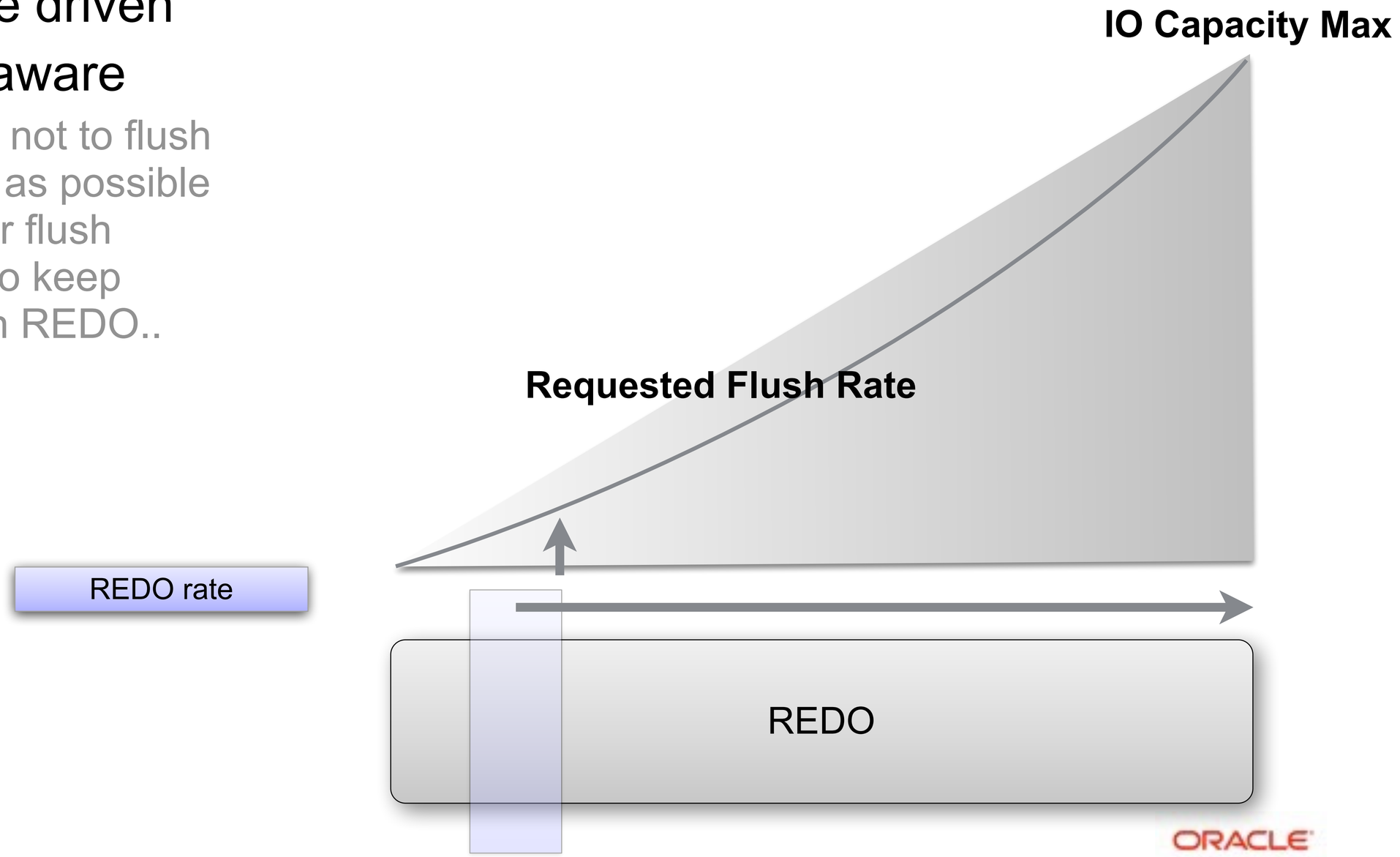
InnoDB Flushing

- REDO rate driven
- LSN Age aware
  - the goal is not to flush as much as possible but rather flush enough to keep a room in REDO..



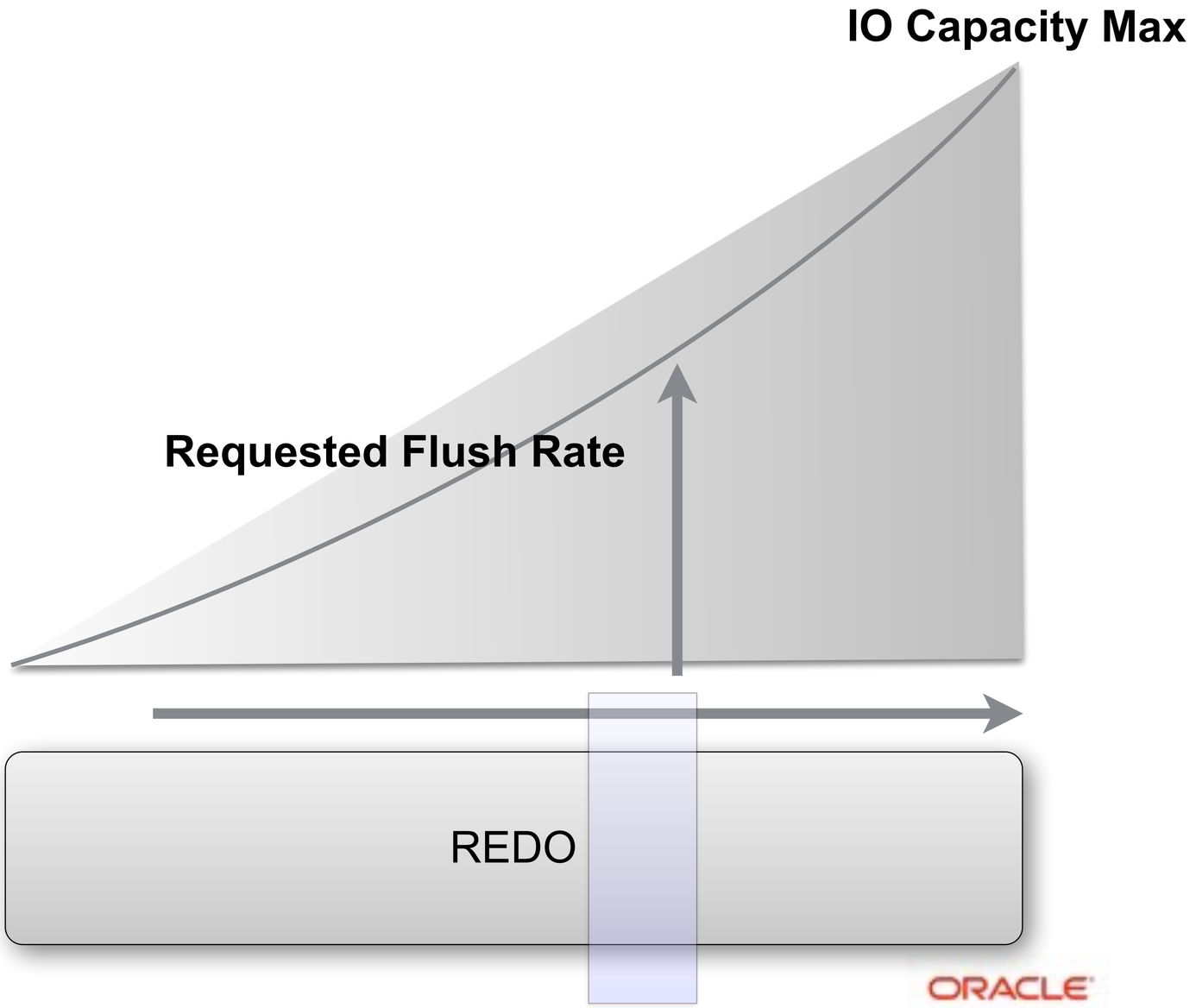
InnoDB Flushing

- REDO rate driven
- LSN Age aware
  - the goal is not to flush as much as possible but rather flush enough to keep a room in REDO..



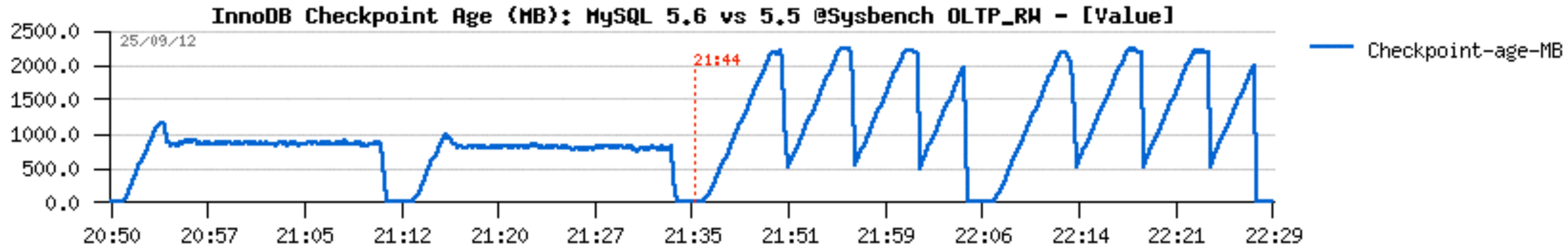
InnoDB Flushing

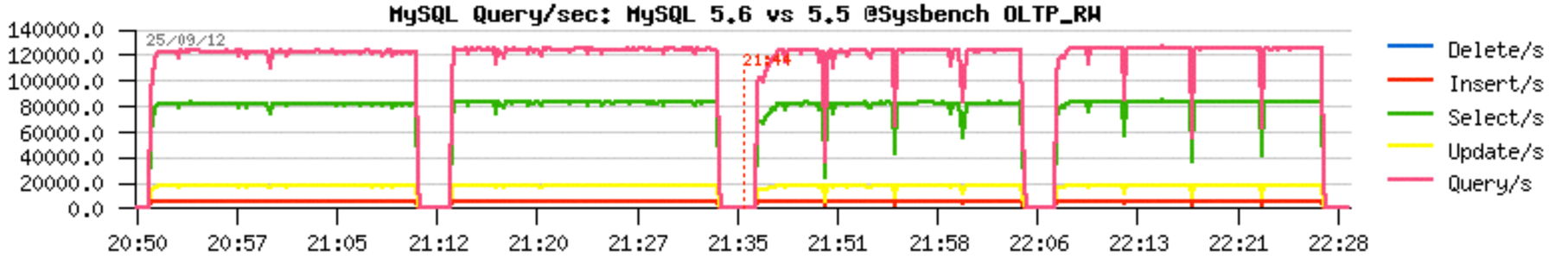
- REDO rate driven
- LSN Age aware
  - the goal is not to flush as much as possible but rather flush enough to keep a room in REDO..



Adaptive Flushing: MySQL 5.6 vs 5.5

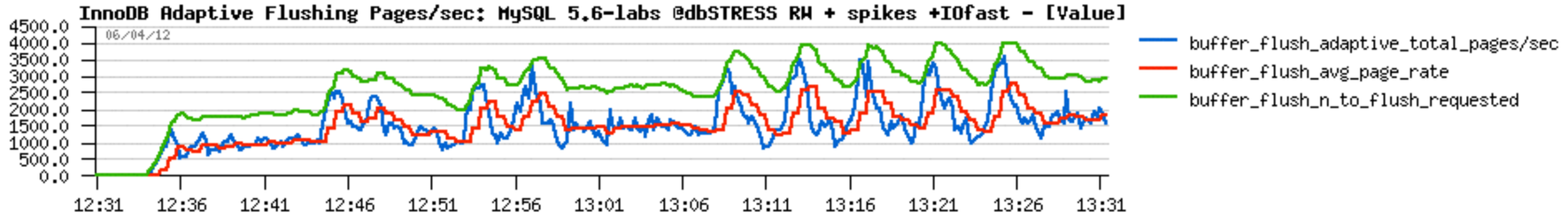
- OLTP_RW Workload:
  - Same IO capacity
  - Different logic..

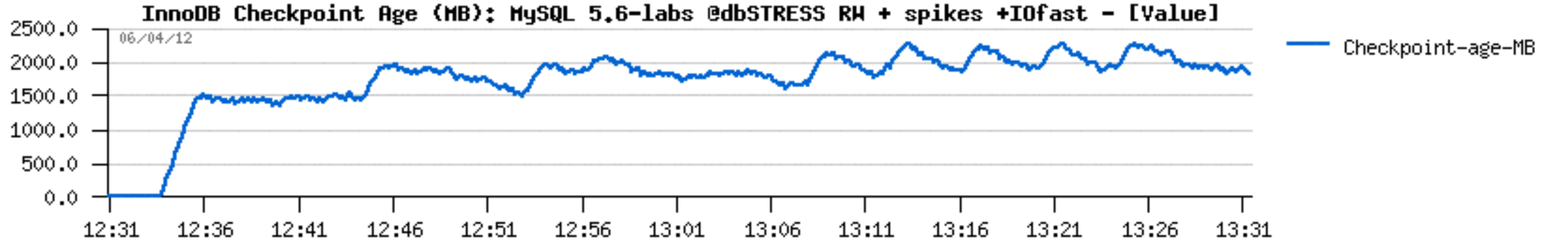


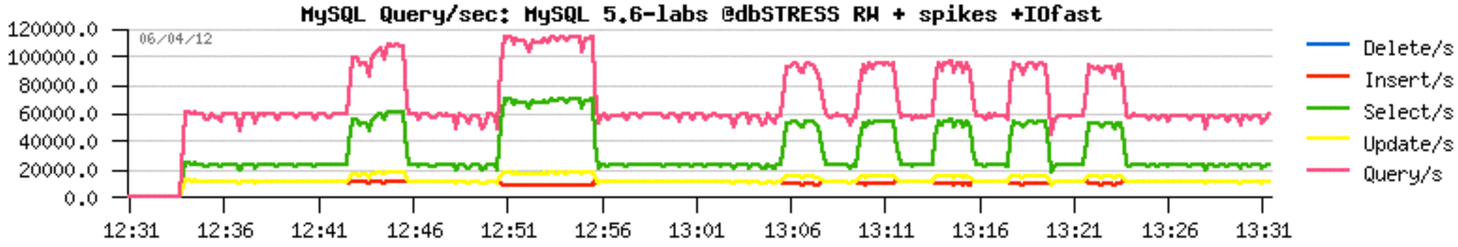


InnoDB : Resisting to activity spikes in 5.6

- dbSTRESS RW with spikes
  - having a big enough Checkpoint Age marge allowing to resist to spikes



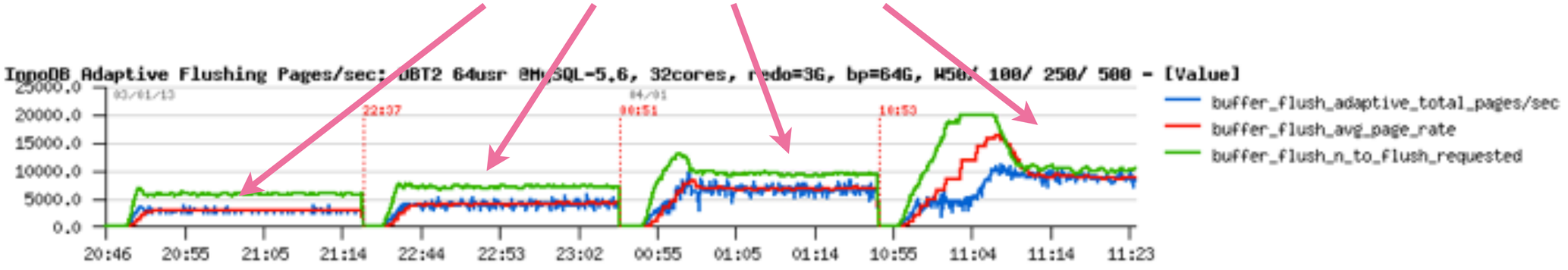




ORACLE

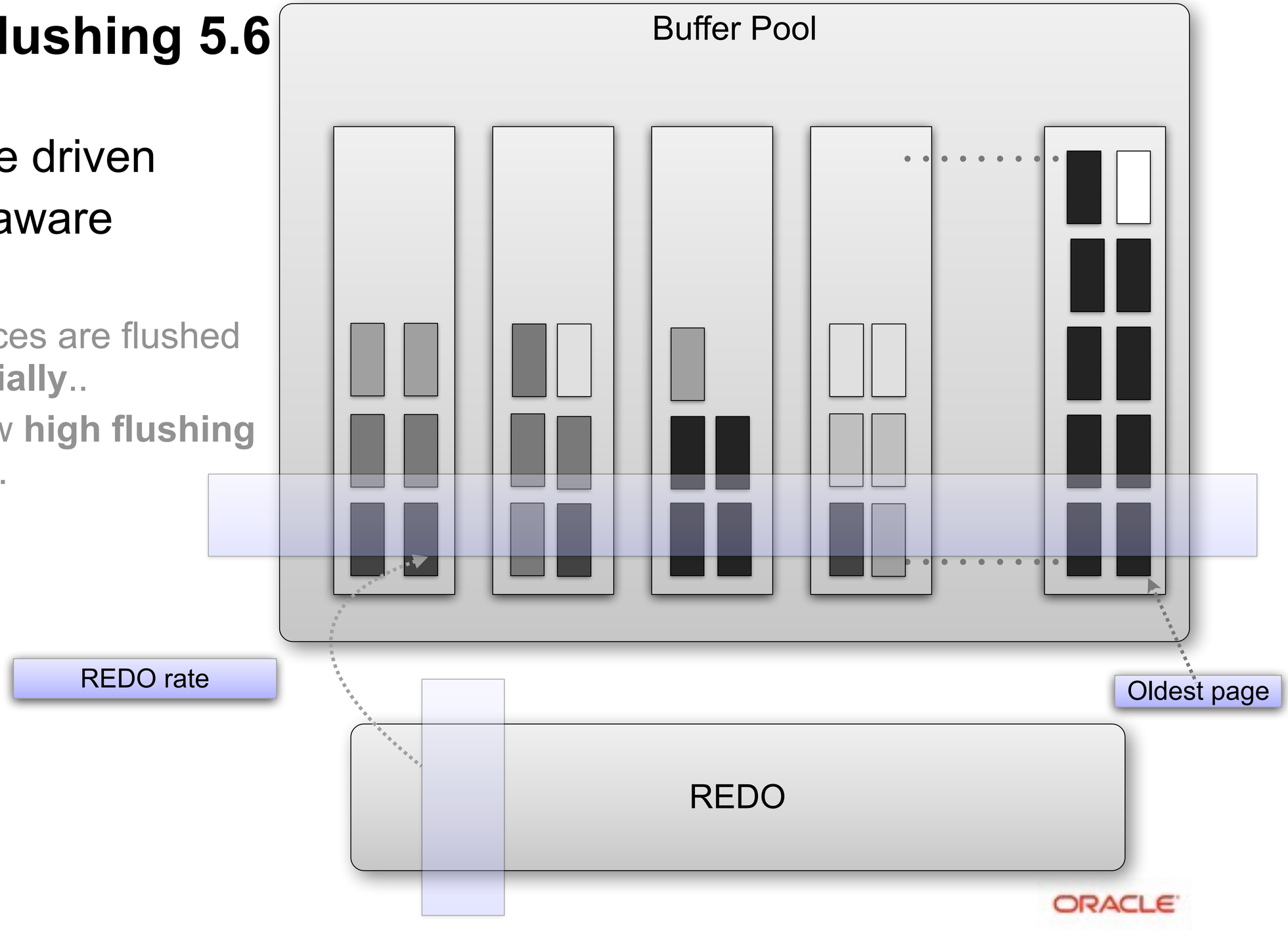
RW IO-bound “In-Memory”

- Impact of the database size
  - with a growing db size the TPS rate may be only the same or worse ;-)
  - and required Flushing rate may only increase.. <= **need parallel flushing** !
- DBT2 workload :
  - 64 users, db volume: 50W, 100W, 250W, 500W



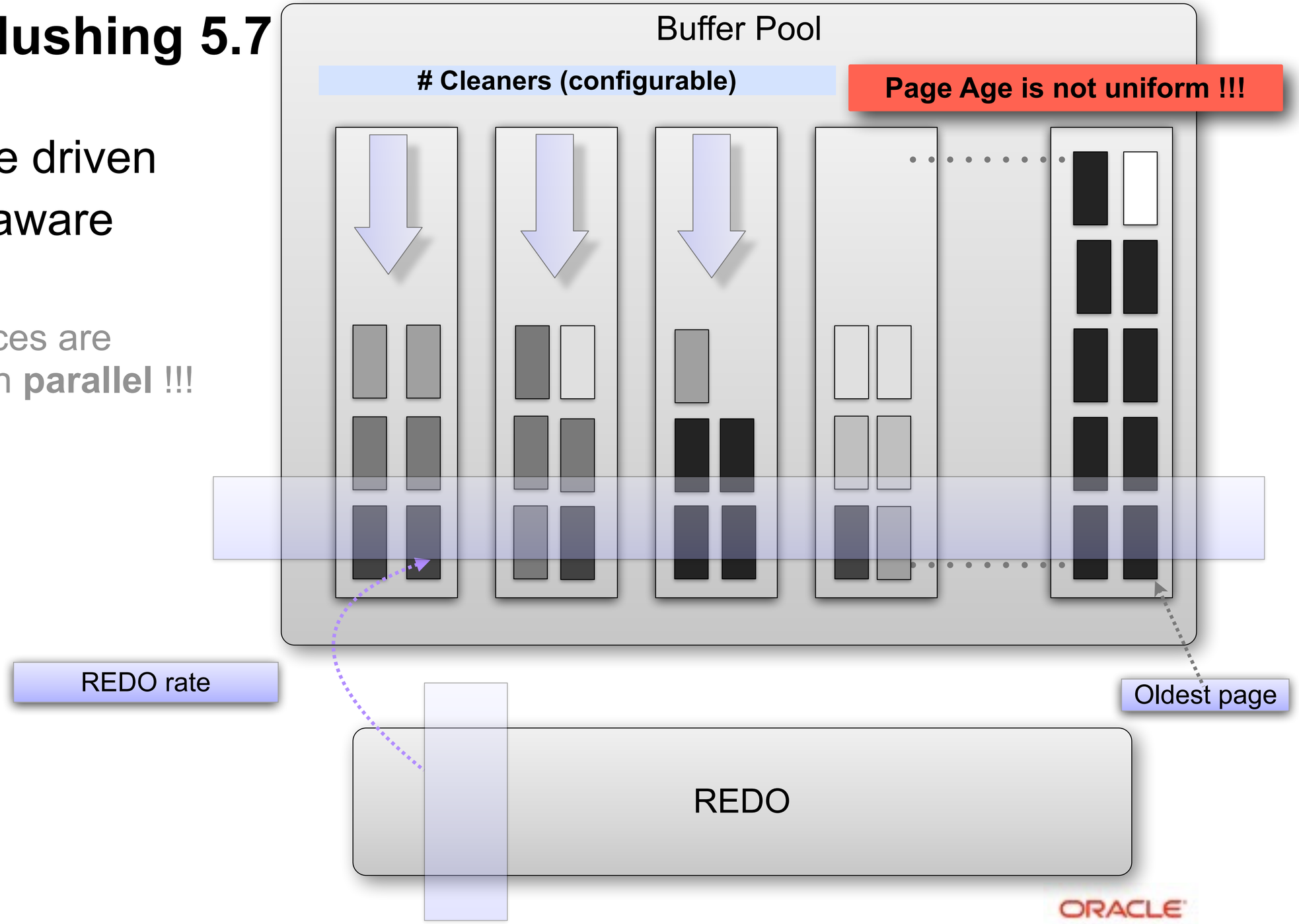
InnoDB Flushing 5.6

- REDO rate driven
- LSN Age aware
- 5.6 :
  - BP Instances are flushed **sequentially**..
  - can't follow **high flushing** demand..



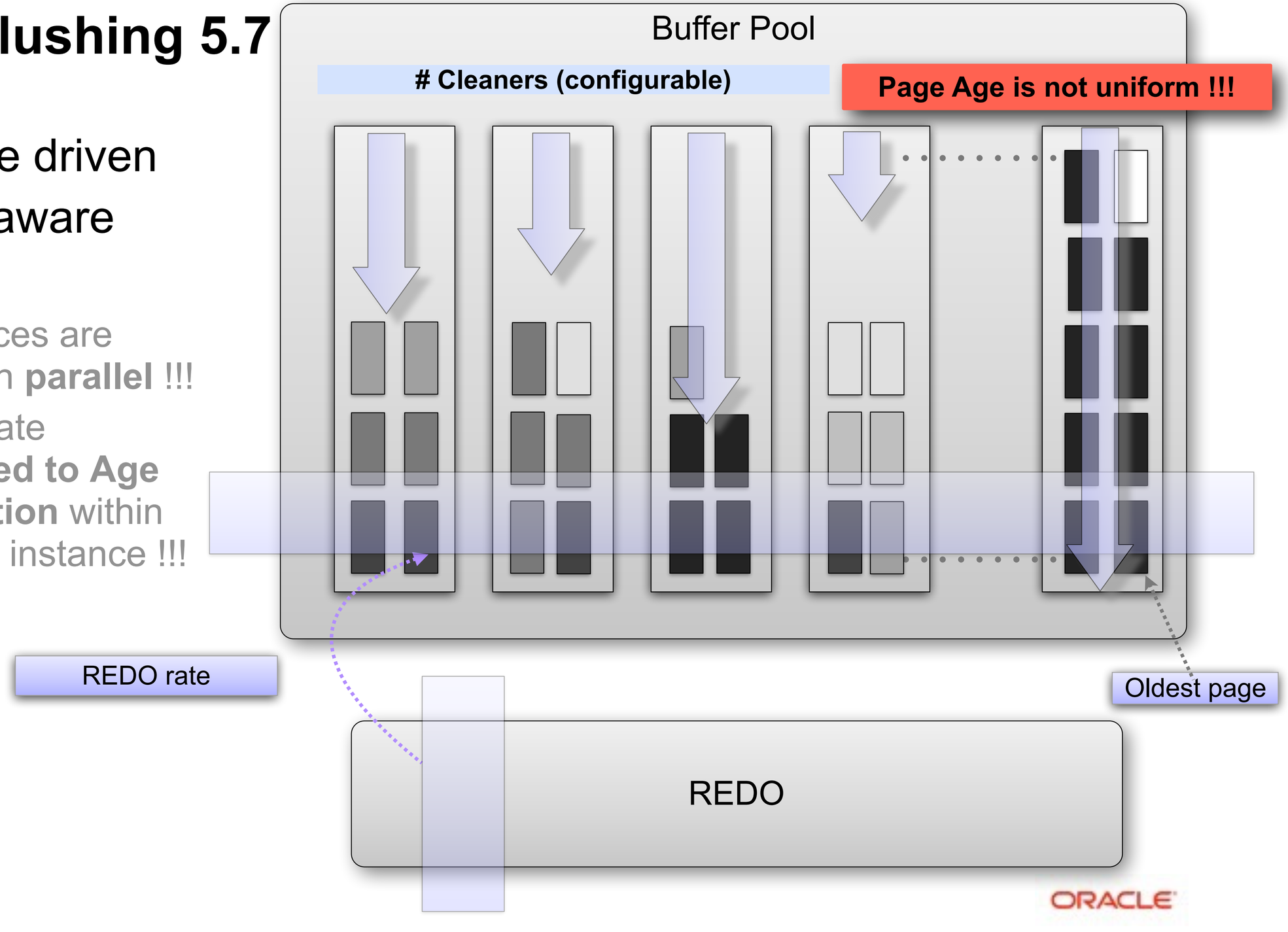
InnoDB Flushing 5.7

- REDO rate driven
- LSN Age aware
- 5.7 :
  - BP Instances are flushed in **parallel** !!!



InnoDB Flushing 5.7

- REDO rate driven
- LSN Age aware
- 5.7 :
  - BP Instances are flushed in **parallel** !!!
  - Flushing rate is **adapted to Age distribution** within **each** BP instance !!!

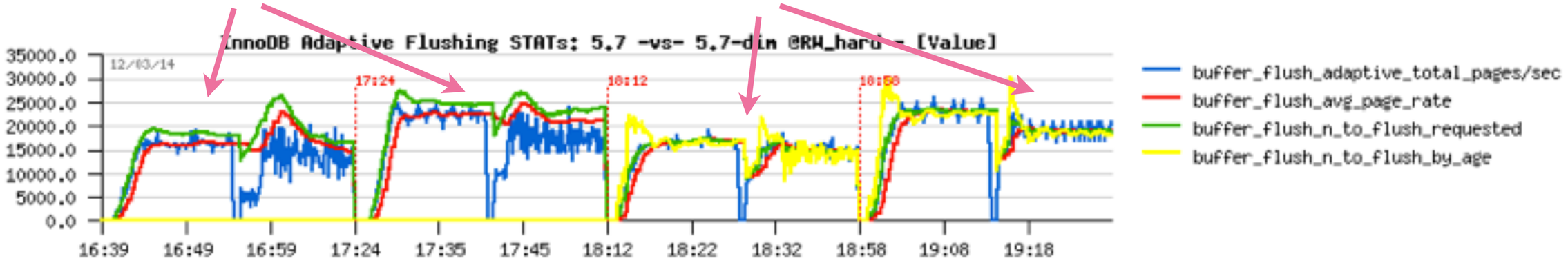


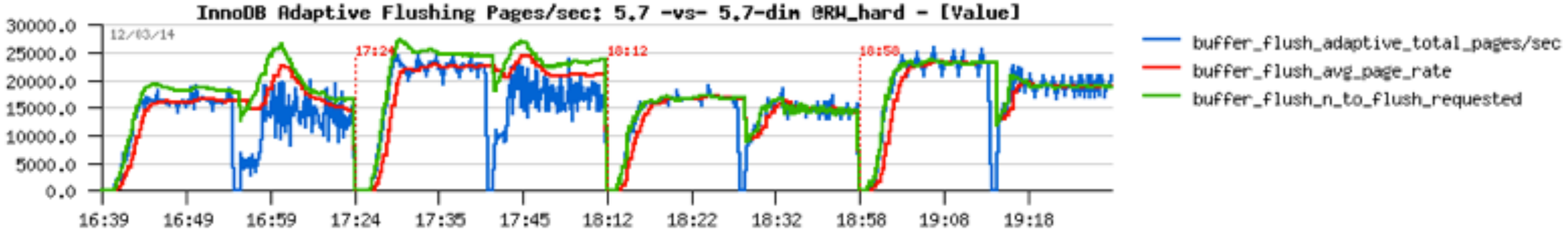
InnoDB Adaptive Flushing Tuning in 5.7

- Config :
  - innodb_adaptive_flushing = 1 (=> Linux: also allow cleaner threads priority !!)
  - REDO log size —> use big ;-) (ex: 12GB, 32GB)
  - innodb_page_cleaners = 4
  - innodb_io_capacity_max = … (max allowed (10000 ?))
  - innodb_io_capacity = 1/2 innodb_io_capacity_max (or according your needs)
  - innodb_max_dirty_pages_pct_lwm = 5
  - innodb_max_dirty_pages_pct = 90
- Monitor :
  - Checkpoint Age < REDO total size
  - buffer_flush_sync_waits && buffer_flush_sync_pages == 0
  - buffer_flush_avg_time < 1sec
  - buffer_flush_adaptive_avg_pass == 30 (def. avg loops)
  - buffer_flush_adaptive_total_pages/sec == buffer_flush_n_to_flush_requested

InnoDB Flushing in 5.7

- Considering Age distribution :
  - Parallel Only -vs- Parallel + Age aware





RW IO-bound

- **REDO**-driven : Still data In-Memory, but much bigger volume
  - more pages to flush for the **same** TPS rate
- **LRU**-driven : Data bigger or much bigger than Memory / cache / BP
  - the amount of free pages becomes short very quickly..
  - and instead of mostly only IO writes, you're starting to have IO reads too
  - these reads are usually random reads
  - if your storage is slow - reads will simply kill your TPS ;-)
  - if your storage can follow - once you're hitting fil_sys mutex you're done
  - as well LRU flushing may become very heavy..
- NOTE:
  - on Linux : **AIO + O_DIRECT_NO_FSYNC** seems to be the most optimal for RW IO-bound
  - but always check yourself ;-)

RW IO-bound “Out-of-Memory” LRU-driven

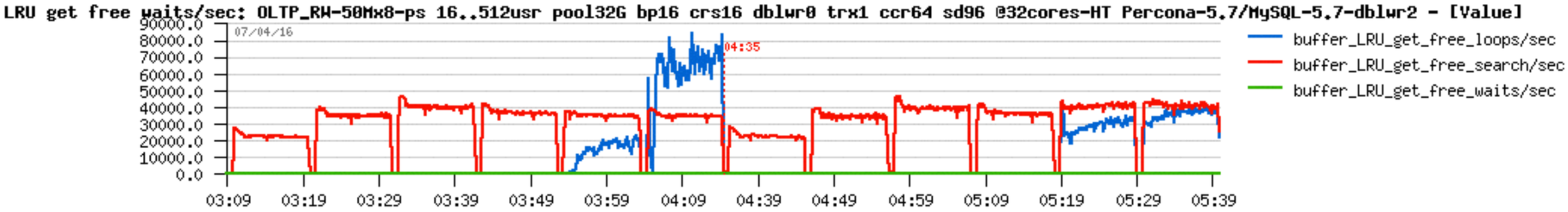
- The “entry” limit here is storage performance
  - as you’ll have a lot of IO reads..
  - => and to be able to read a new data you need a free page in BP
  - => if there no more free pages :
    - => you can evict a clean page from LRU tail
    - => you can flush & evict a dirty page from LRU tails
  - **=> e.g. to allow IO reads you must process first your IO writes**

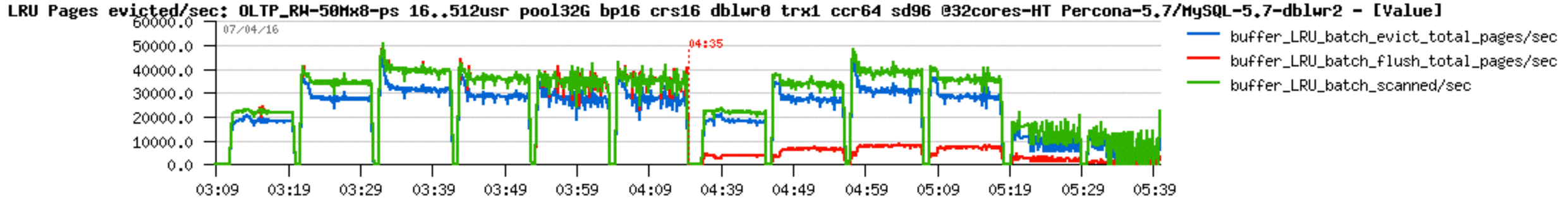
RW IO-bound “Out-of-Memory” LRU-driven

- Config :
  - **innodb_buffer_pool_instances** = 8 (16, 32..)
  - **innodb_page_cleaners** = 4 or 8 or eq. BP instances (depends on free page demand)
  - **innodb_lru_scan_depth** = 4K or more (according free page demand)
    - NOTE: innodb_lru_scan_depth is **per BP instance !!!**
    - NOTE: it also **defines your free pages target !!!**
- Tuning <=> Monitoring :
  - **buffer_LRU_get_free_search**/sec <== your free pages demand
    - so, align your LRU depth according this to match the demand
  - **buffer_LRU_get_free_loops**/sec <== loop waits on free pages…
  - **buffer_LRU_get_free_waits**/sec <== sleep waits on free pages..
  - **buffer_LRU_single_flush_num_scan**/sec <== single page flush/evict by no-cleaner..
  - **buffer_LRU_batch_evict_total_pages**/sec <== pages evicted by cleaner
  - **buffer_LRU_batch_flush_total_pages**/sec <== pages flushed by cleaner
  - **buffer_LRU_batch_scanned**/sec <== pages scanned by cleaner

RW IO-bound LRU-driven

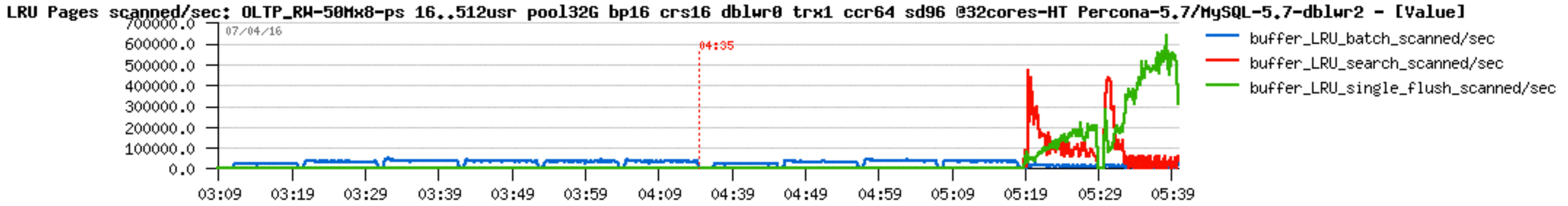
- OLTP_RW 50M x8-tab IO-bound LRU-driven
  - Percona-5.7 / MySQL-5.7
    - => the “optimal” solution is somewhere in the middle..

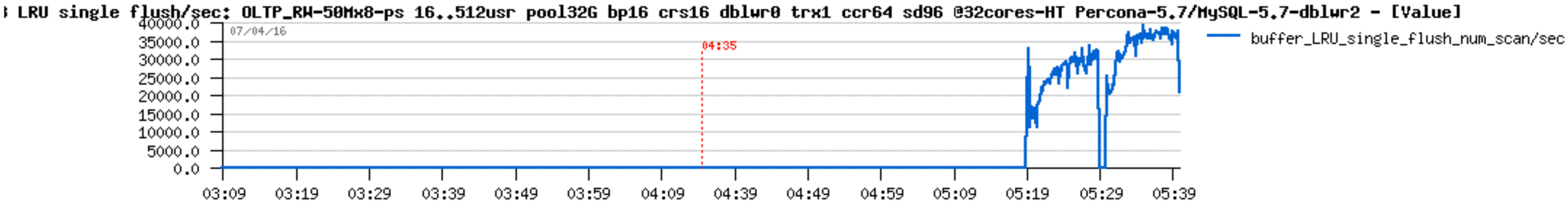




RW IO-bound LRU-driven

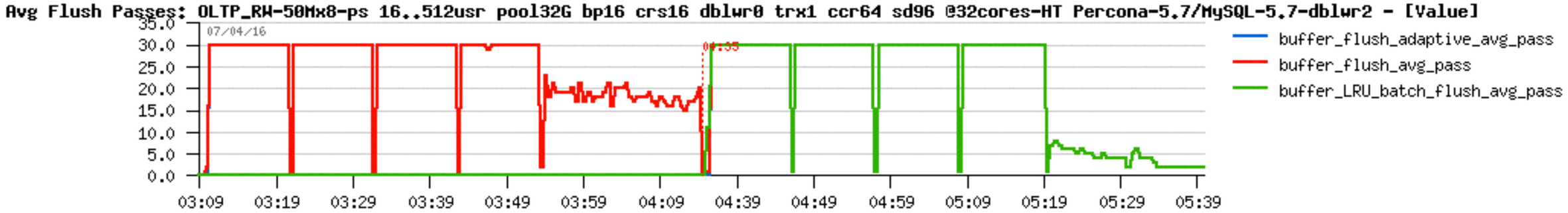
- OLTP_RW 50M x8-tab IO-bound LRU-driven
  - Percona-5.7 / MySQL-5.7
    - => the “optimal” solution is somewhere in the middle..

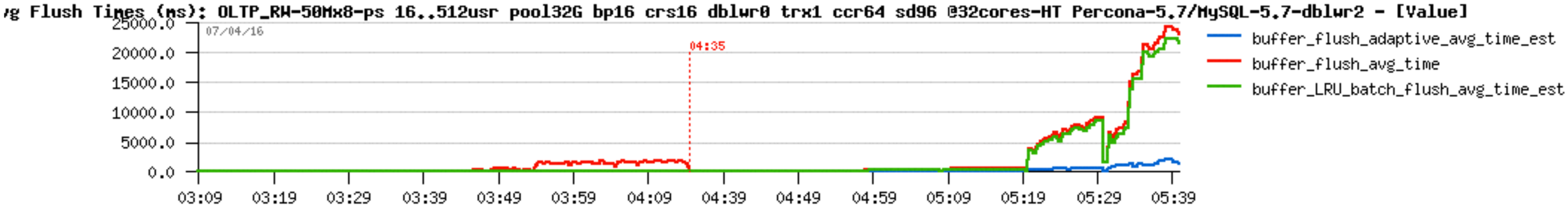




RW IO-bound LRU-driven

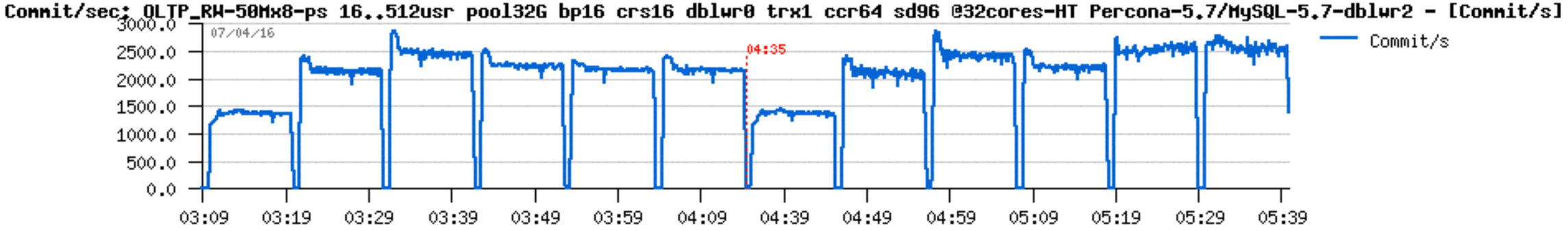
- OLTP_RW 50M x8-tab IO-bound LRU-driven
  - Percona-5.7 / MySQL-5.7
    - => the “optimal” solution is somewhere in the middle..

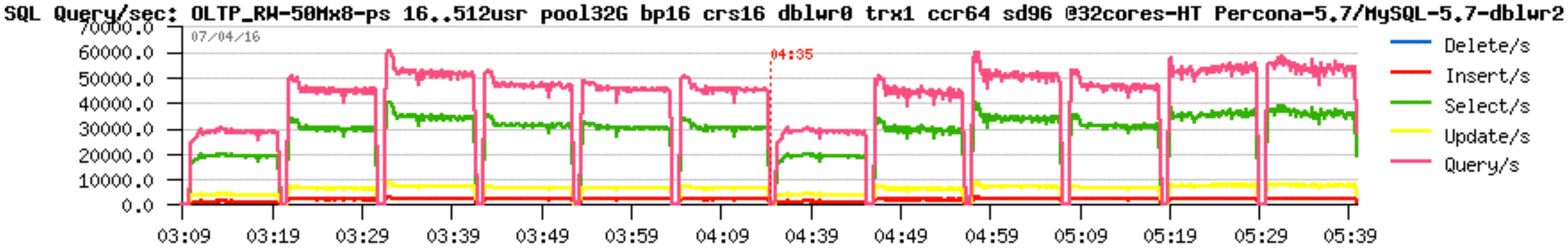




RW IO-bound LRU-driven

- OLTP_RW 50M x8-tab IO-bound LRU-driven
  - Percona-5.7 / MySQL-5.7
    - => the “optimal” solution is somewhere in the middle..
    - => as MySQL-5.7 “looked” bad, but delivered a higher TPS..



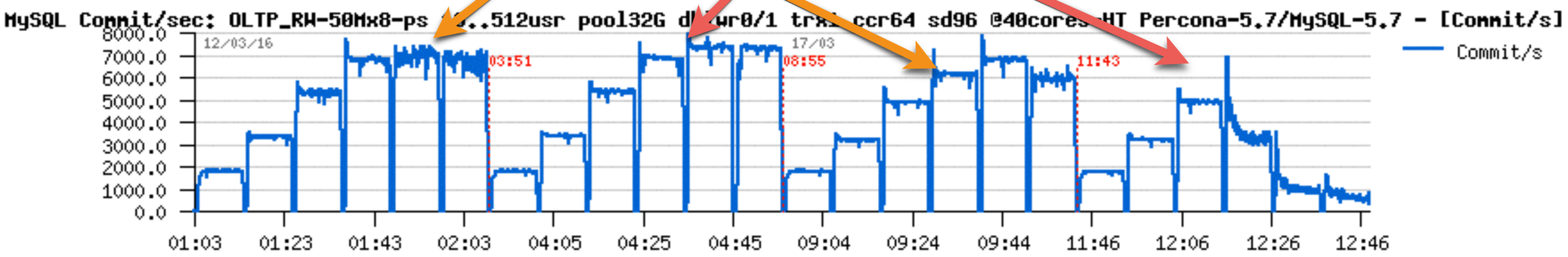


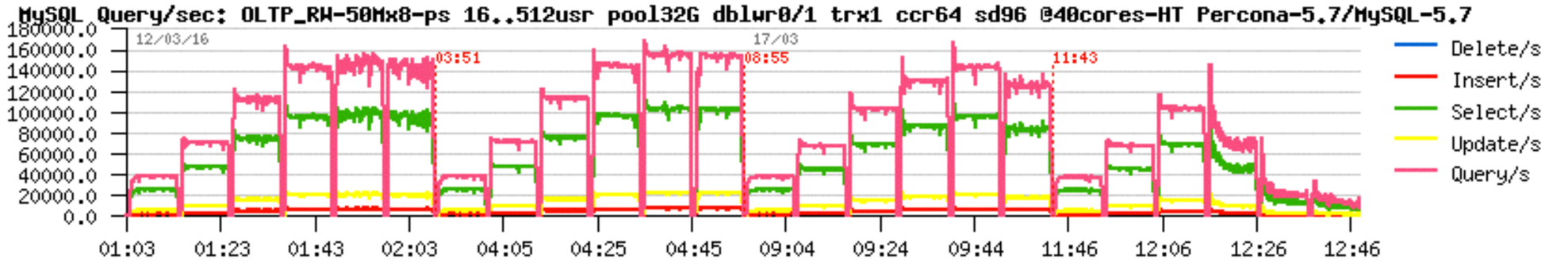
InnoDB Double-Write (DBLWR)

- Why ?
  - the only InnoDB feature to protect from partially written pages
  - each page is written **twice** (first into DBLWR zone, then to data file)
  - on recovery:
    - if corrupted page is detected => InnoDB is seeking DBLWR
    - if no “good” page image found => you’re in trouble ;-))
  - impact :
    - page write latency is growing at least x2 times..
    - flash storage life expectation becomes x2 times lower (due x2 more writes)
  - solution :
    - allow placing DBLWR to other storage (ex: $5 USB-stick / SD, $50 SSD, etc.)
    - allow more parallel writes to hide increased IO page write latency
    - => DBLWR path / size / threads config options (coming as 5.7+ fix)

InnoDB Double-Write (DBLWR)

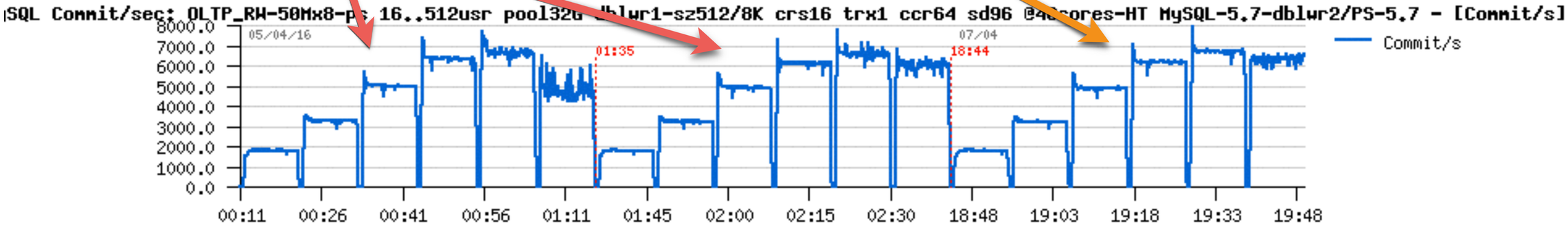
- OLTP_RW 50M x8-tables (120G dataset)
  - BP=32G, trx=1, dblwr=0/1, checksum=crc32, Flash “Nytro” Seagate-XP6500
  - Percona-5.7 / MySQL-5.7 (Jan.2016)

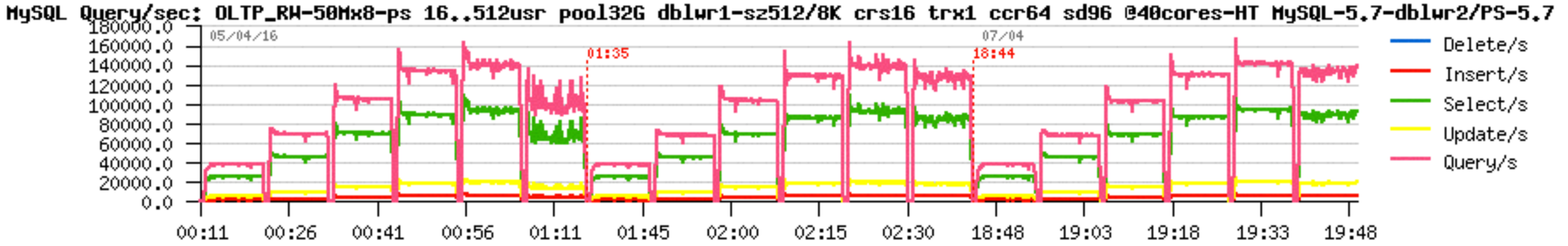




InnoDB Double-Write (DBLWR)

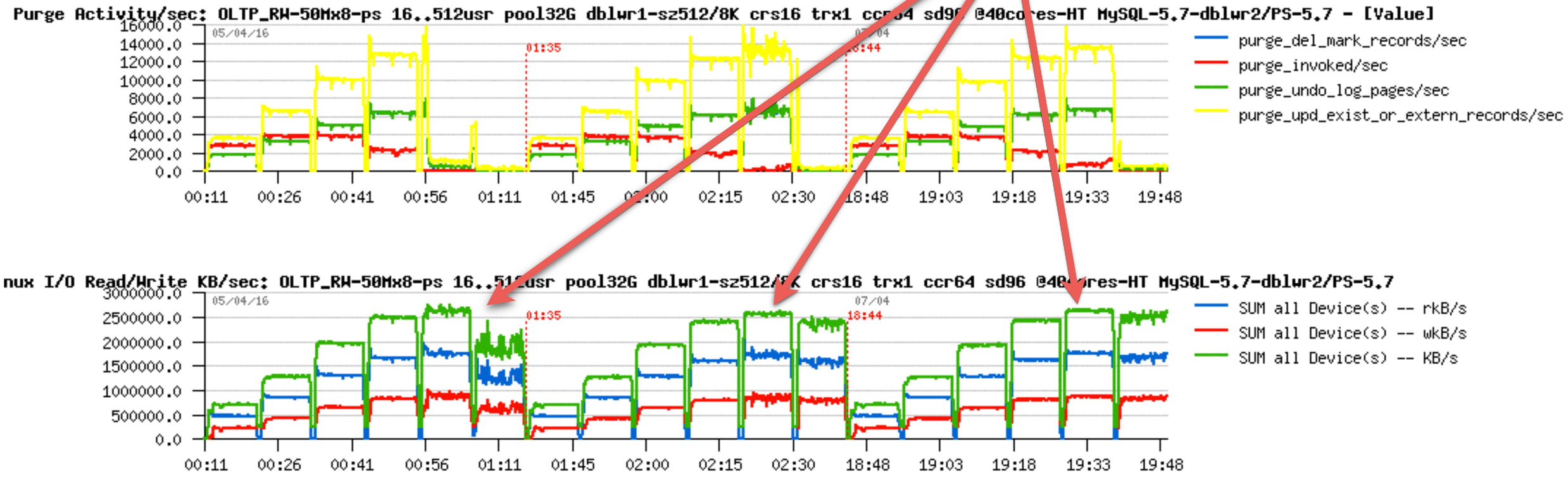
- OLTP_RW 50M x8-tables (120G dataset)
  - BP=32G, trx=1, dblwr=1, checksum=crc32, Flash “Nytro” Seagate-XP6500
  - MySQL-5.7-dblwr (work-in-progress) / Percona-5.7





InnoDB Double-Write (DBLWR) - Side Note..

- OLTP_RW 50M x8-tables (120G dataset)
  - Purge lagging can be a very serious issue..
  - 5.7 with Flash “Nytro” Seagate-XP6500 => **over 2500 MB/sec** (16K InnoDB pages)



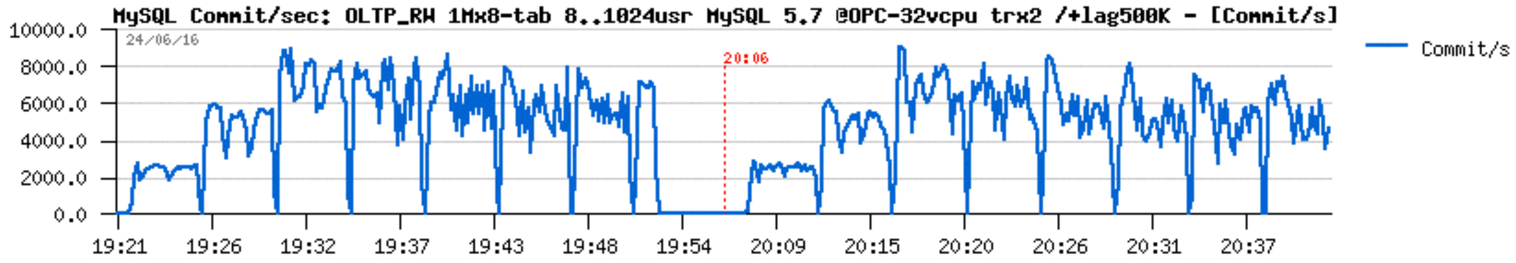
Read+Write Workloads : InnoDB Purge

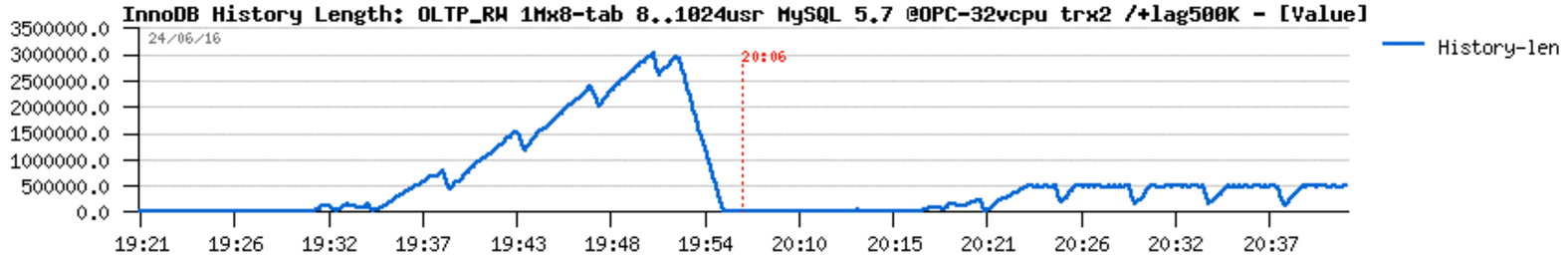
- InnoDB Purge...
  - 5.5 : Purge Thread !!! ;-)
  - 5.6 : Multi-Threaded Purge + fix for purge lag code !
  - 5.7 : UNDO space **can be auto-dropped** !! (RTFM..)
    - monitor InnoDB History Length **ALWAYS !** ;-)
    - if NO purge lagging : excellent! (& be happy! ;-))
    - if purge is lagging : use a purge lag config setting.. (write throttling)

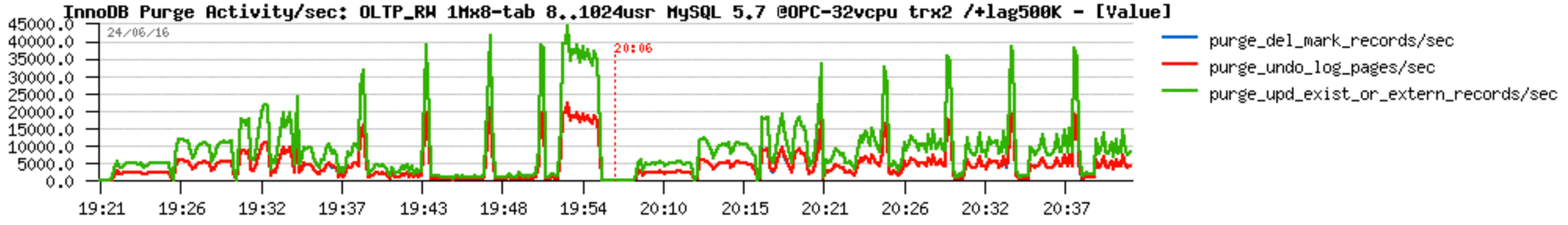
  - **Example of config for 5.6 and 5.7 to avoid purge lagging:**
    - innodb_max_purge_lag = 500000 (500K max, ex.)
    - innodb_max_purge_lag_delay = 30000000
    - innodb_purge_threads = 4

InnoDB Purge Lag

- OLTP_RW @MySQL 5.7
  - purge lag = 0 -vs- purge lag = 500K

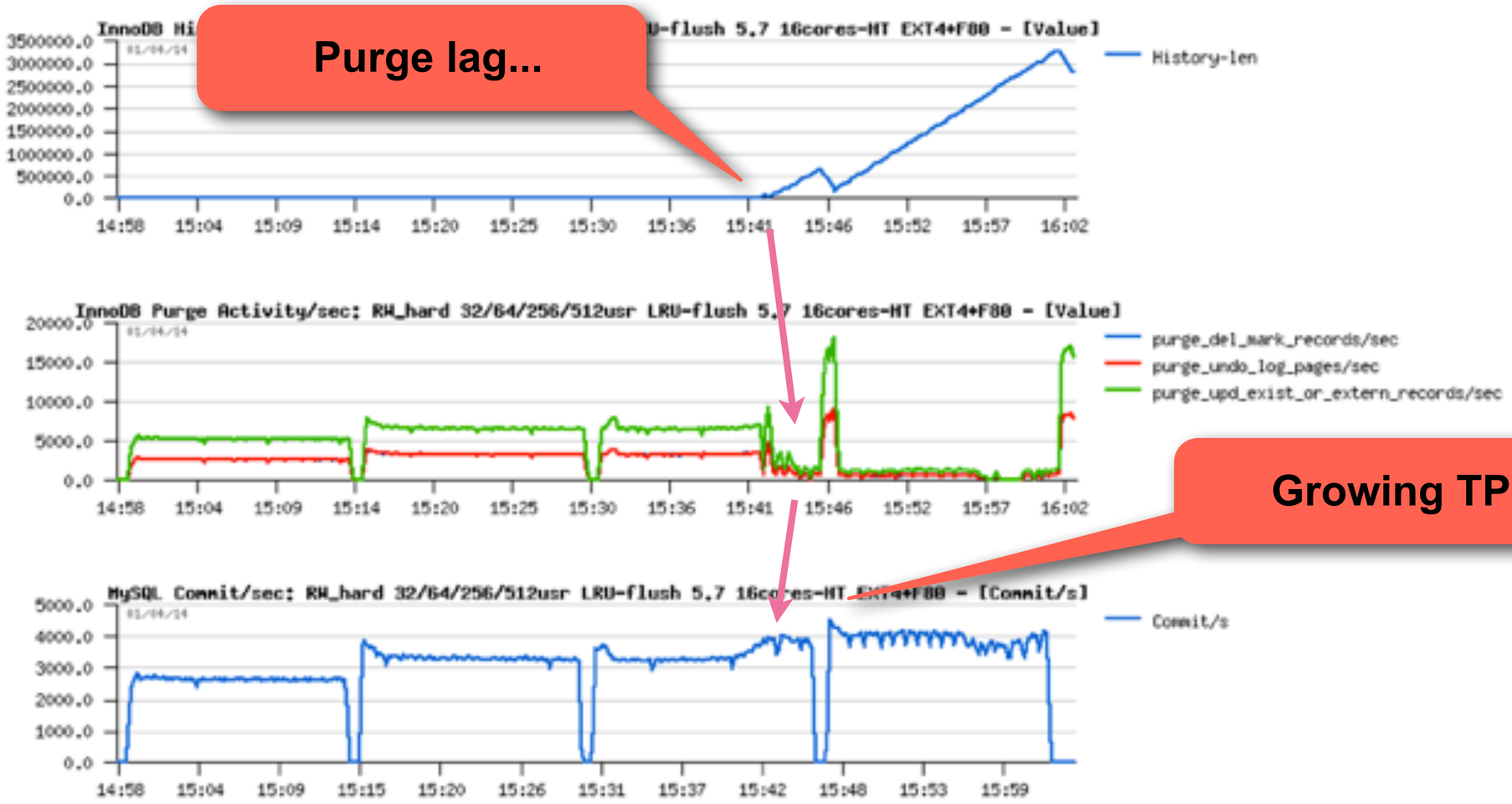






InnoDB : be sure your TPS is fair ;-)

- Purge lagging impact on IO-bound OLTP_RW 10Mx32-tab:
  - moving from 3200 to 4000 TPS... - cool, right? ;-) but not fair...



INSERT Performance in MySQL 5.7

- B-Tree impact + InnoDB data compactness..
  - over a time of INSERTS, B-Tree is growing & growing..
  - at some moment it'll be out of memory..
  - this will involve IO re-reads (mostly IO RR !!)
  - which will slowdown an overall performance..
- Workaround(s)
  - size a bigger memory for InnoDB Buffer Pool (BP)
  - use partitions :
    - this will keep an overall BTree(s) smaller
    - once you filled up a partition and switching INSERTs to the next one, the previous partition index data are no more required during INSERT, and BP will cache index pages mostly from the active partition..
- MySQL 8 : stay tuned ;-)

UPDATE Performance in MySQL 5.7

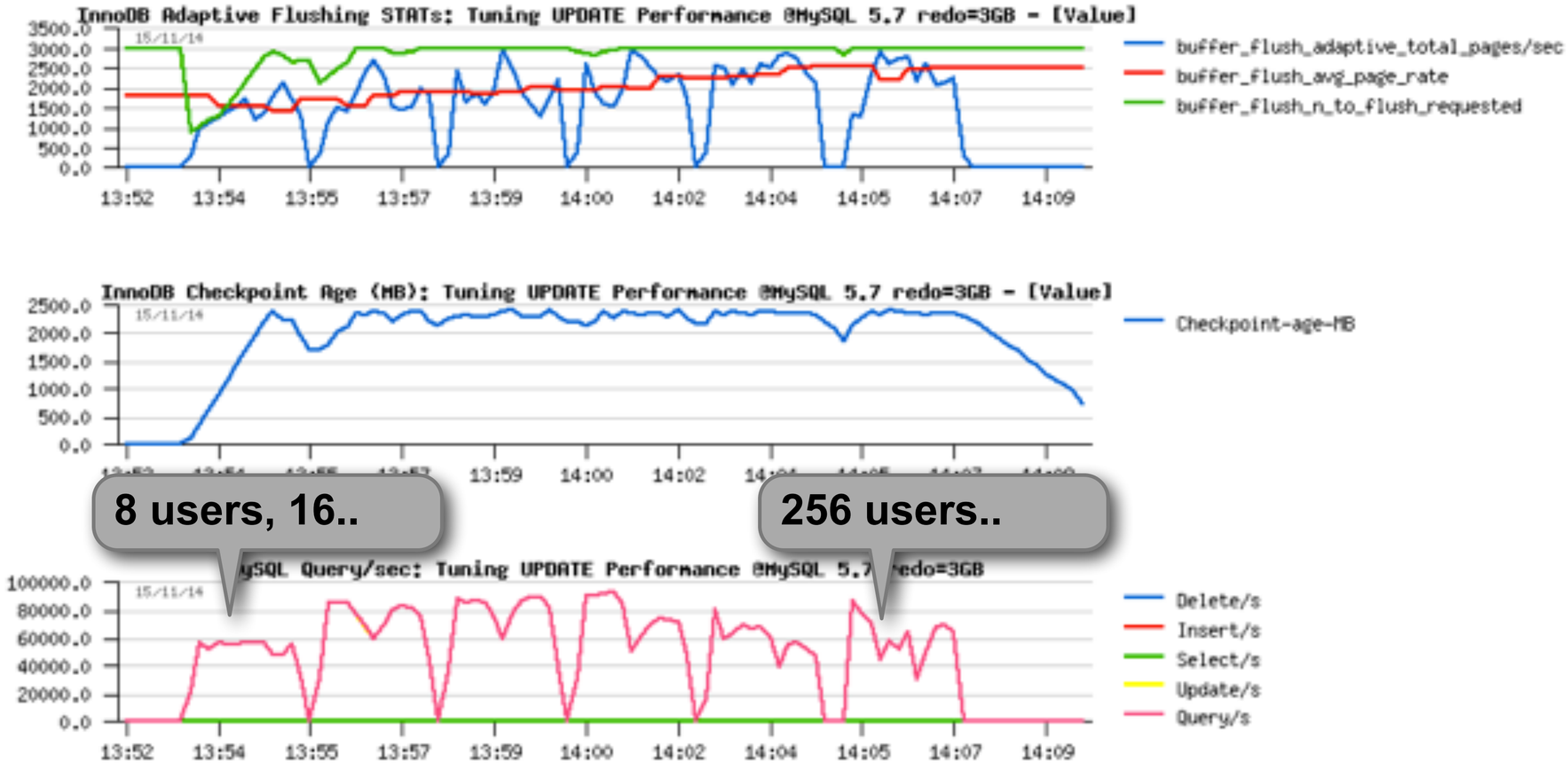
- Low load : slower than in MySQL 5.6
  - pure overhead in many functions due code changes..
- Higher load : much better than in MySQL 5.6
  - so, have to manage to do more and more stuff in parallel !!
  - and this is a general tendency…

Test Case: Tuning UPDATE Performance

- Test conditions :
  - Workload : Sysbench UPDATE
  - CPU config : 12cores-HT
  - IO subsystem : EXT4 on SSD
  - Users : 8, 16, 32 .. 256

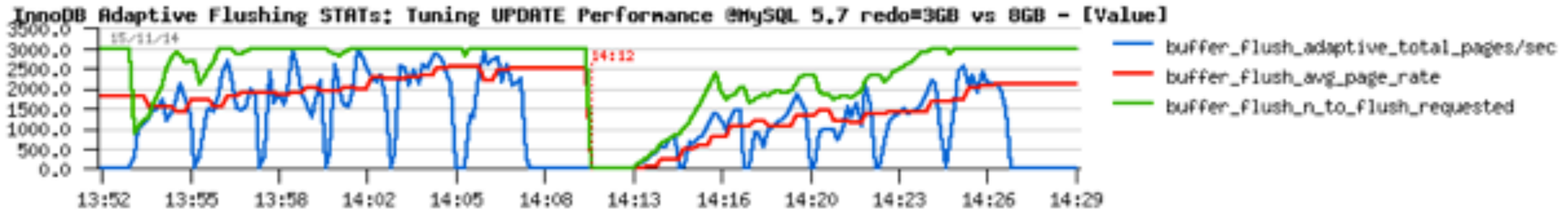
Test Case: Tuning UPDATE Performance (2)

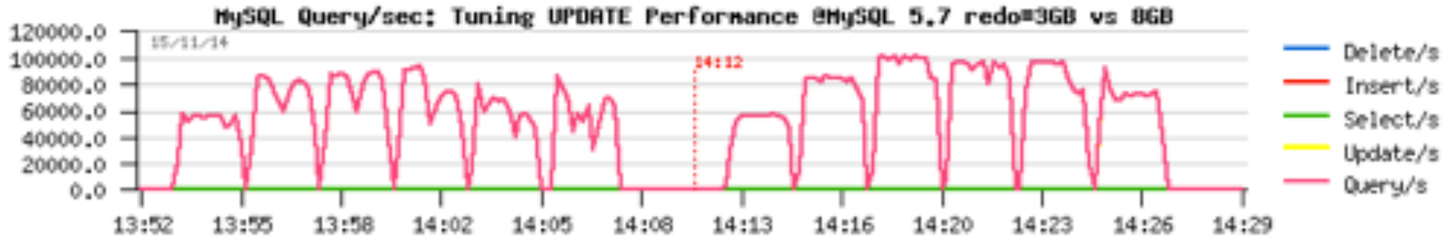
- Tuning :
  - starting with REDO size=3GB, io capacity max=3000
  - Performance: looks poor..



Test Case: Tuning UPDATE Performance (3)

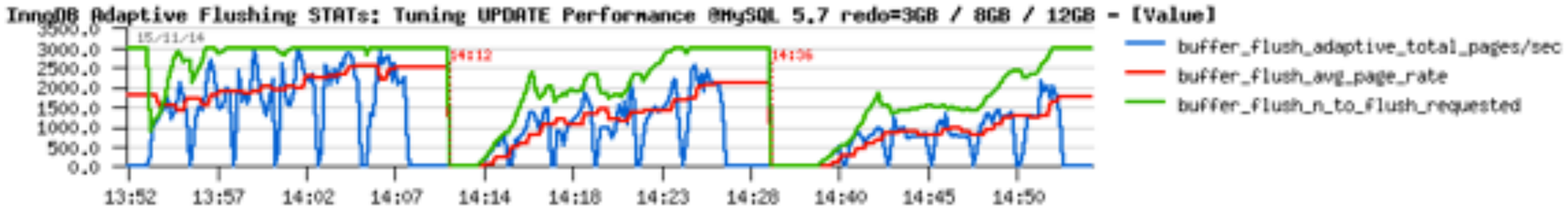
- Tuning :
  - moving to REDO size=8GB..
  - Performance: looks better, but still poor on a higher load..

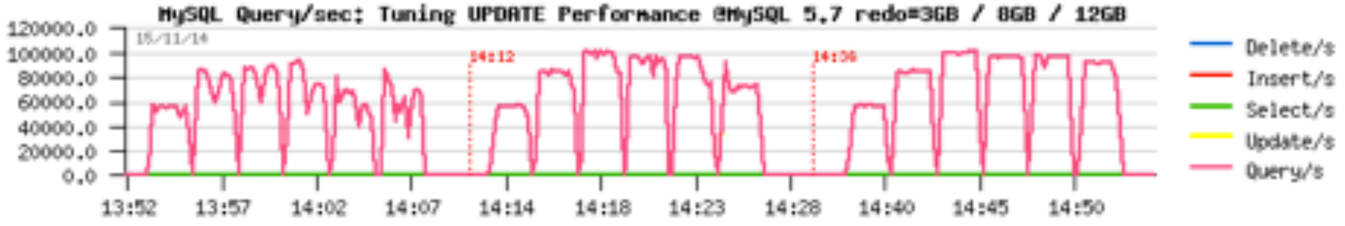




Test Case: Tuning UPDATE Performance (4)

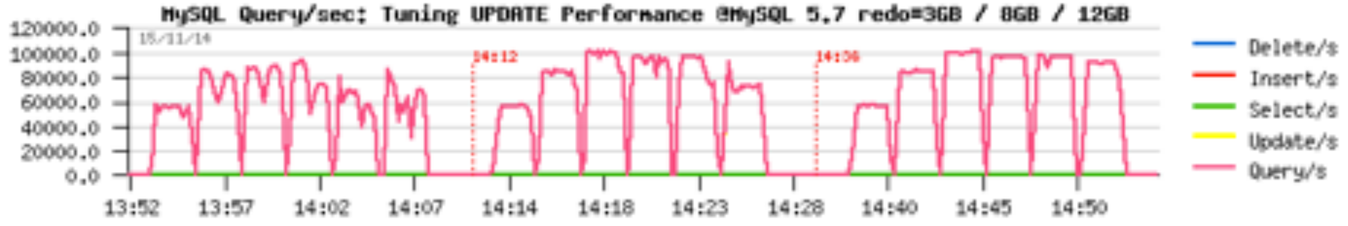
- Tuning :
  - moving to REDO size=12GB..
  - Performance: looks good, but Checkpoint Age continues to grow..





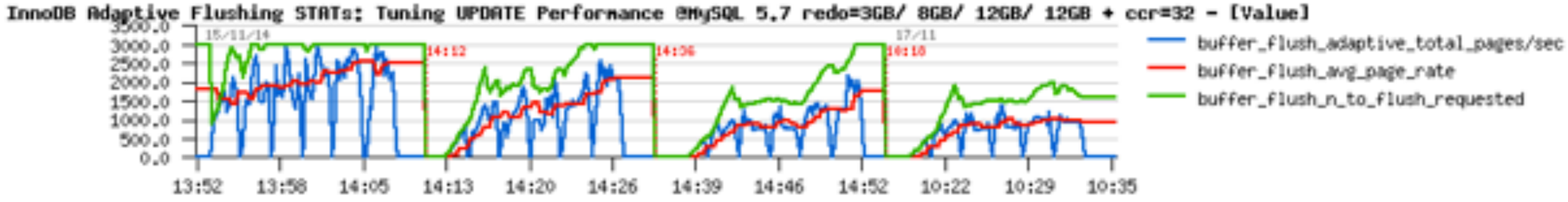
Test Case: Tuning UPDATE Performance (5)

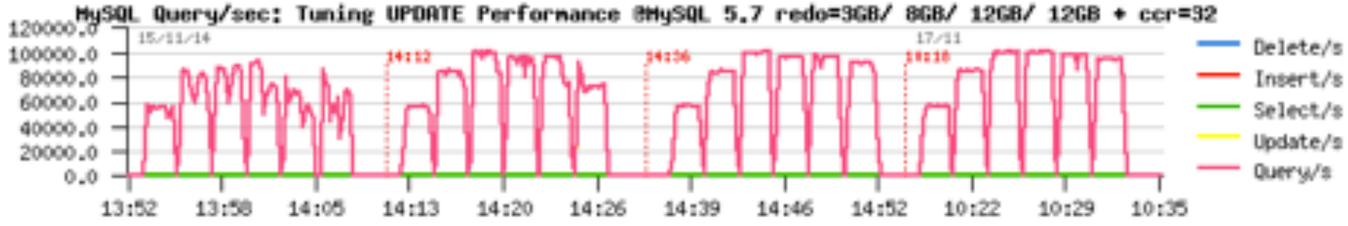
- Tuning :
  - moving to REDO size=12GB..
  - Performance: looks good, but Checkpoint Age continues to grow..
  - Analyze: up to 128 users all is going well..
  - So, we have to reduce the user's concurrency here



Test Case: Tuning UPDATE Performance (6)

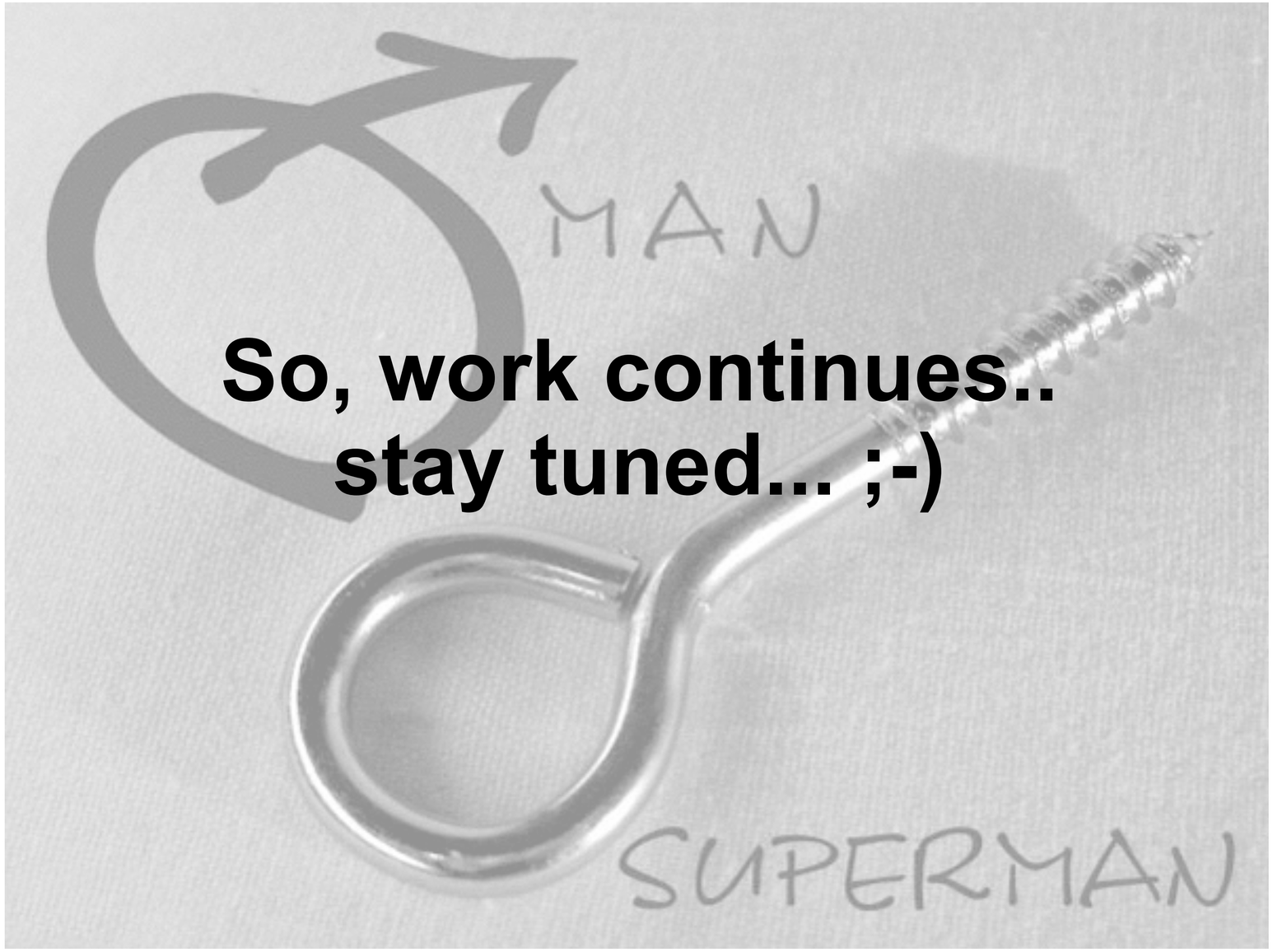
- Tuning :
  - REDO size=12GB + innodb thread concurrency=32
  - Performance: just fine! ;-)





Hope you’re seeing much more clear now ;-)

- And there is less mystery for you around MySQL Performance Tuning
- Most of stuff is available since MySQL 5.7 only..
- So, what do you wait to upgrade ?.. :-)


MAN
So, work continues..
stay tuned... ;-)
SUPERMAN

One more thing ;-)

- All graphs are built with dim_STAT (http://dimitrik.free.fr)
  - All System load stats (CPU, I/O, Network, RAM, Processes,...)
  - Manly for Linux, Solaris, OSX (and any other UNIX too :-)
  - Add-Ons for MySQL, Oracle RDBMS, PostgreSQL, Java, etc.
  - MySQL Add-Ons:
    - mysqlSTAT : all available data from “show status”
    - mysqlLOAD : compact data, multi-host monitoring oriented
    - mysqlWAITS : top wait events from Performance SCHEMA
    - InnodbSTAT : most important data from “show innodb status”
    - innodbMUTEX : monitoring InnoDB mutex waits
    - innodbMETRICS : all counters from the METRICS table
    - And any other you want to add! :-)
- Links
  - http://dimitrik.free.fr - dim_STAT, dbSTRESS, Benchmark Reports, etc.
  - http://dimitrik.free.fr/blog - Articles about MySQL Performance, etc.