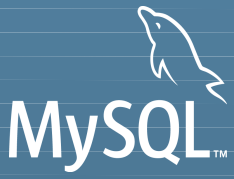

MySQL™ Connect



ORACLE

MySQL 5.6 Performance: Tuning and “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 last years officially @MySQL Performance full time now
- http://dimitrik.free.fr/blog / @dimitrik_fr

Agenda

- Overview
- Analyzing MySQL Workload
- Analyzing and Understanding of MySQL Internals
- Performance improvements in MySQL 5.6 (and 5.7)
- Benchmark results
- Pending issues..
- Q & A

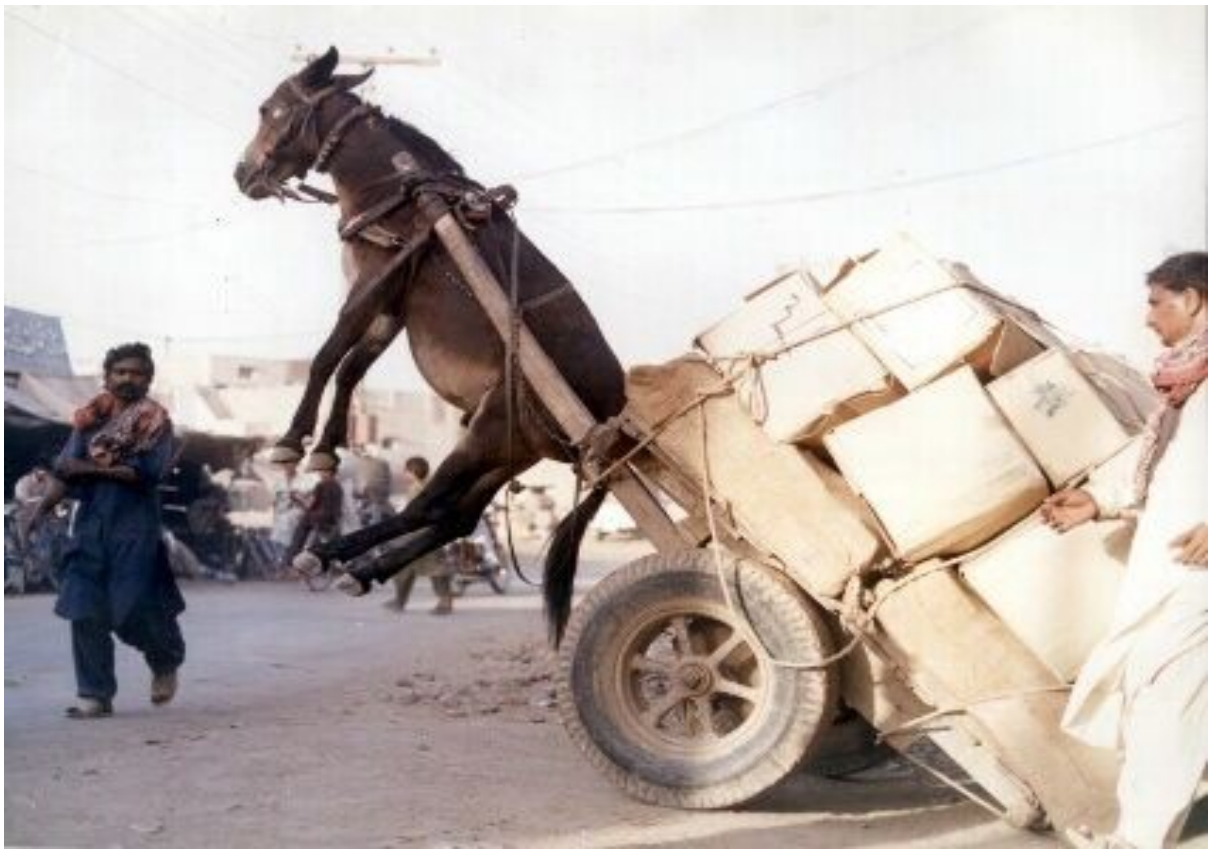
Why MySQL Performance ?...

Why benchmarking MySQL?..

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

Why benchmarking MySQL?..

- Until it did not reach its limit..



Why benchmarking MySQL?..

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

Why benchmarking MySQL?..

- And reach a similar limit..

Why benchmarking MySQL?..

- A good benchmark testing may help you understand ahead the resistance of your solution to incoming potential problems ;-)

Why benchmarking MySQL?..

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

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

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

USE YOUR BRAIN !!! :-)

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

USE YOUR BRAIN !!! :-)

Before we started..

- Please, keep in mind:
  - **NOBODY** knows everything ;-))
  - There is no absolute true in any topic around..
  - The best answer in most cases will be probably "**It depends..**" ;-))
  - So, again, "**USE YOUR BRAIN!**" is the best advice and the best option
  - Also, knowledge and understanding of problems are changing all the time..
  - And probably even what I'll tell you today is already obsolete. ;-))
  - Enjoy thinking and digging problems deeply ;-))
  - MySQL Performance is a very fun topic (specially current days ;-))

Different Approach for different problems

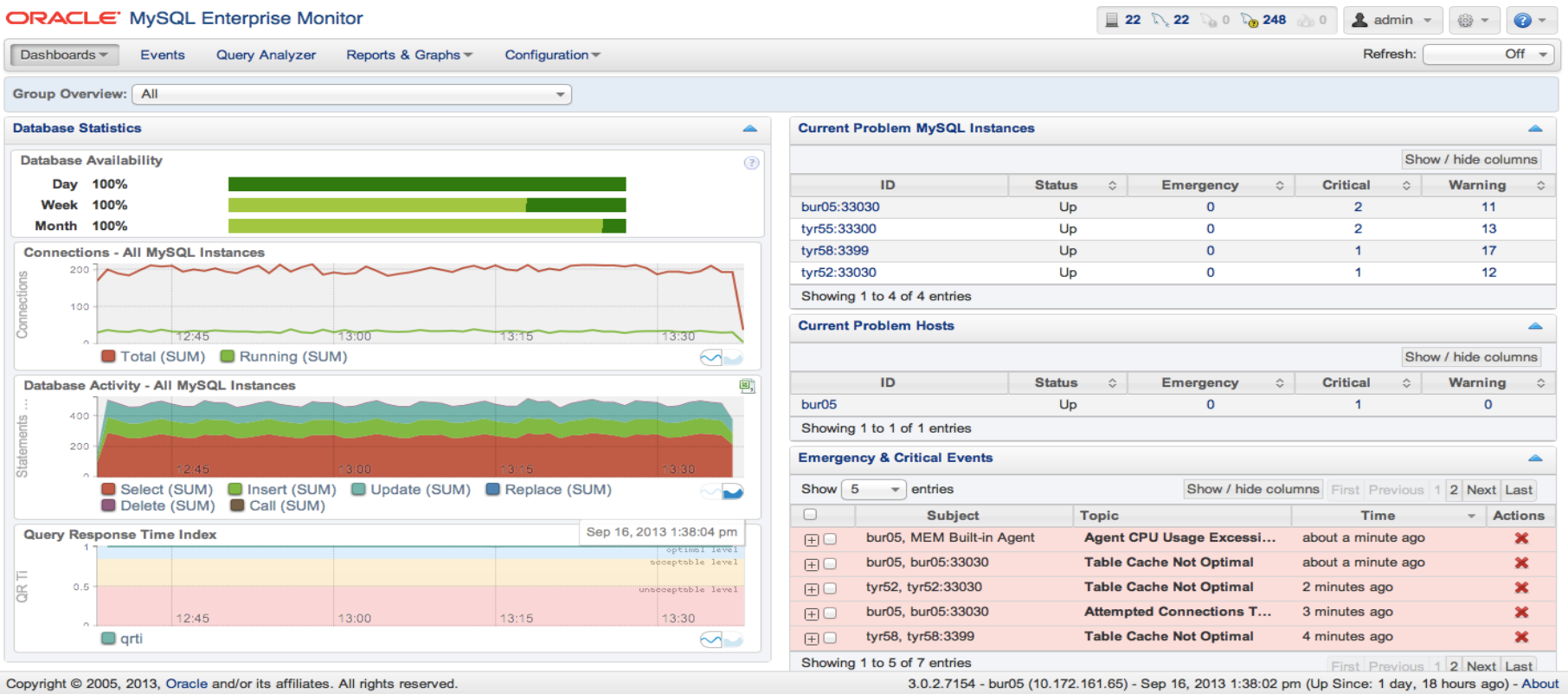
- You are discovering a production workload..
  - Full discovery..
- You are trying to understand why your production is running slower time to time..
  - Tracing, debugging, analyzing, discovering of new problems ;-)
- You are looking for a new platform for existing production workload (or new apps under dev.)..
  - Workload simulation, benchmarking, discovering of the next level issues..
- etc...

They all have something in common!

- Monitoring !..
  - Choose a tool you're familiar with (or install one and become familiar)
  - Use a tool you can completely trust ;-)
  - Keep in mind that sometimes you may need a 5-10sec interval measurements (or even less).. - not every tool is allowing..
  - Keep a history of your monitoring to be able to compare “good” and “bad” cases..
  - When something is starting to go wrong, usually it'll be not in the place which was always problematic, but in the place started to have a different behavior.. - and your goal is to find it ;-)
  - **Always** monitor your HW and OS !!!

MySQL Enterprise Monitor (MEM) v.3.0

- Absolutely fantastic product!
  - Try it! (and buy it if you like it! ;-) - improve your daily work experience!)



ORACLE

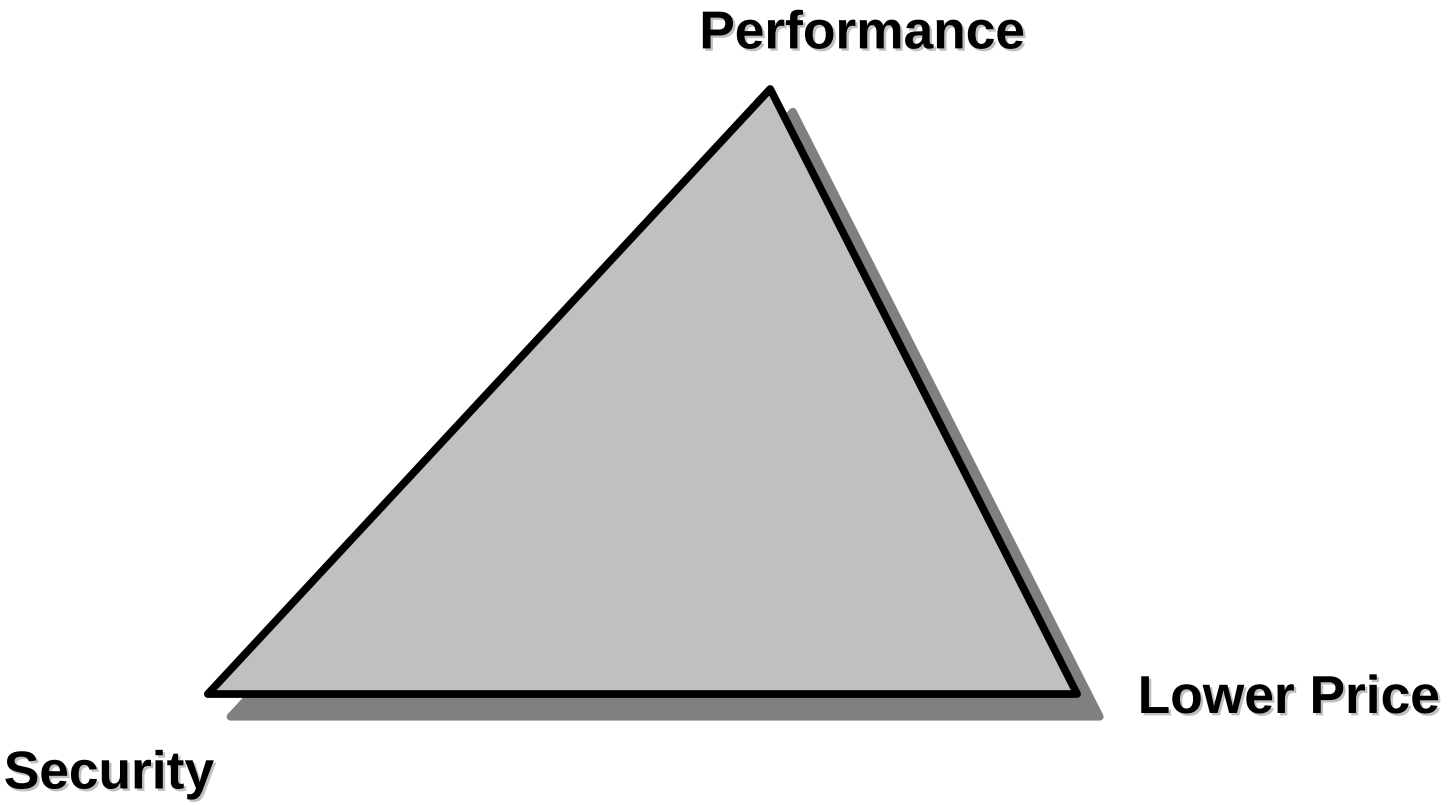
Monitoring & Analyzing with *dim_STAT* (as you ask ;)

- All my graphs are built with it (download: *http://dimitrik.free.fr*)
  - All System load stats (CPU, I/O, Network, RAM, Processes,...)
  - Manly for Solaris & Linux, but any other UNIX too :-)
  - Add-Ons for Oracle, MySQL, 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! :-)

Think “Database Performance” from the beginning!

- Server:
  - Having faster CPU is still better! 32 cores is good enough ;-)
  - OS is important! - Linux, Solaris, etc.. (and Windows too!)
  - Right malloc() lib!! (Linux: jemalloc, Solaris: libumem)
- Storage:
  - Don't use slow disks! (except if this is a test validation goal :-))
  - SSD helping random access! (index/data) more and more cheaper
  - FS is important! - ZFS, UFS, QFS, VxFS, EXT3, EXT4, XFS, etc..
  - O_DIRECT or not O_DIRECT, AIO or not AIO, and be aware of bugs! ;-)
  - Do some generic I/O tests first (Sysbench, IObench, iozone, etc.)
- Don't forget network !! :-) (faster is better, 10Gbit is great!)

Seek for your best option..



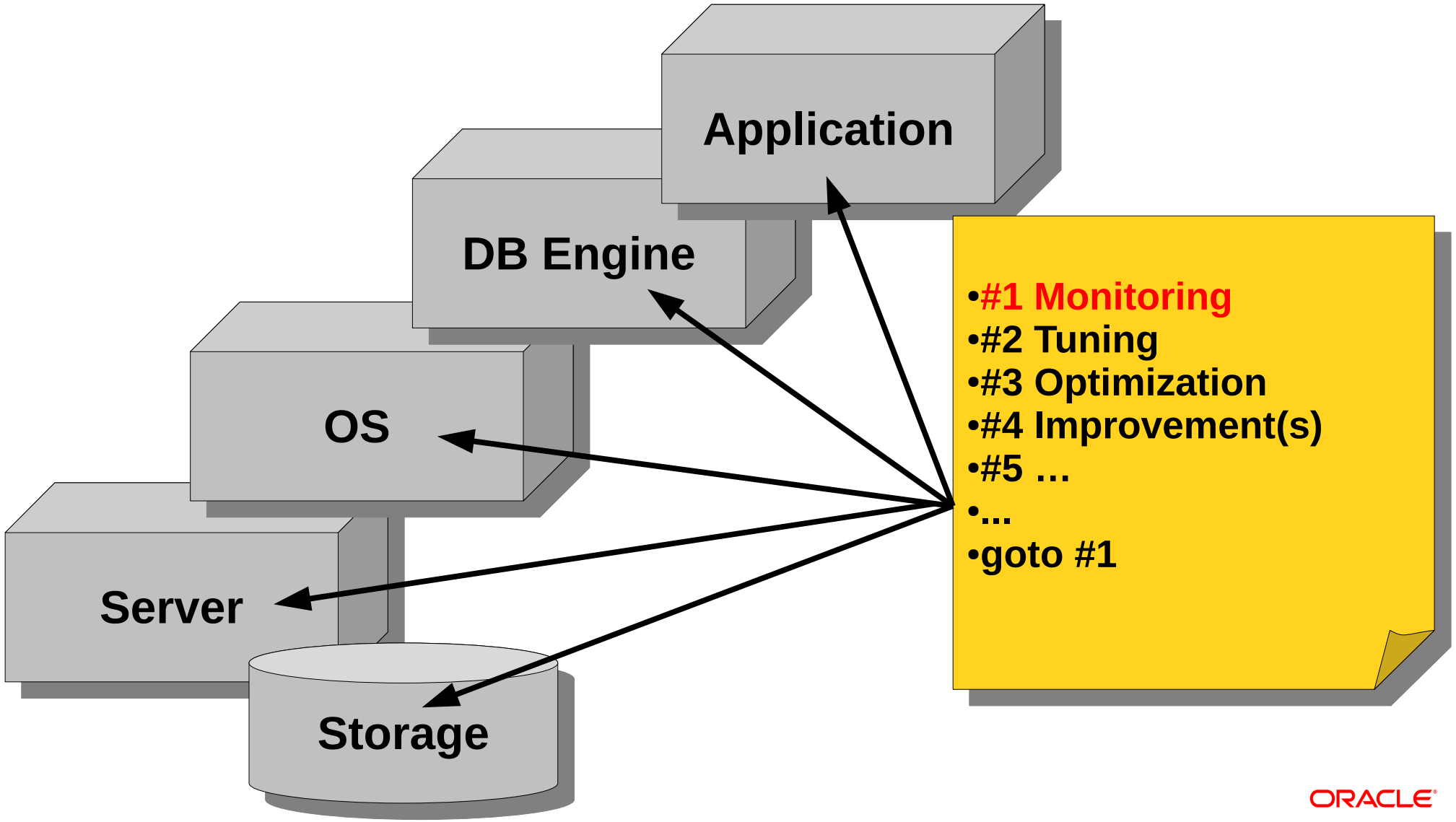
What to monitor on Linux?..

- First of all use the best Linux for you!
  - Or ORACLE Linux if you don't know which one to choose ;-)
  - Install & use “jemalloc”; if XFS has problems, use EXT4 (nobarrier!)
  - Use AIO + O_DIRECT, don't use “cfq” IO scheduler!..
- Always keep an eye on:
  - RunQueue(!), CPU, RAM, Swap in/outProcesses: vmstat, top, psSTAT
  - Storage level: iostat, ..
  - Network: netLOAD, nicstat, …
  - Overall system activity: # perf top -z
    - perf: excellent profiler!
  - **IMPORTANT** : system monitoring usually helps to dig DB issues!

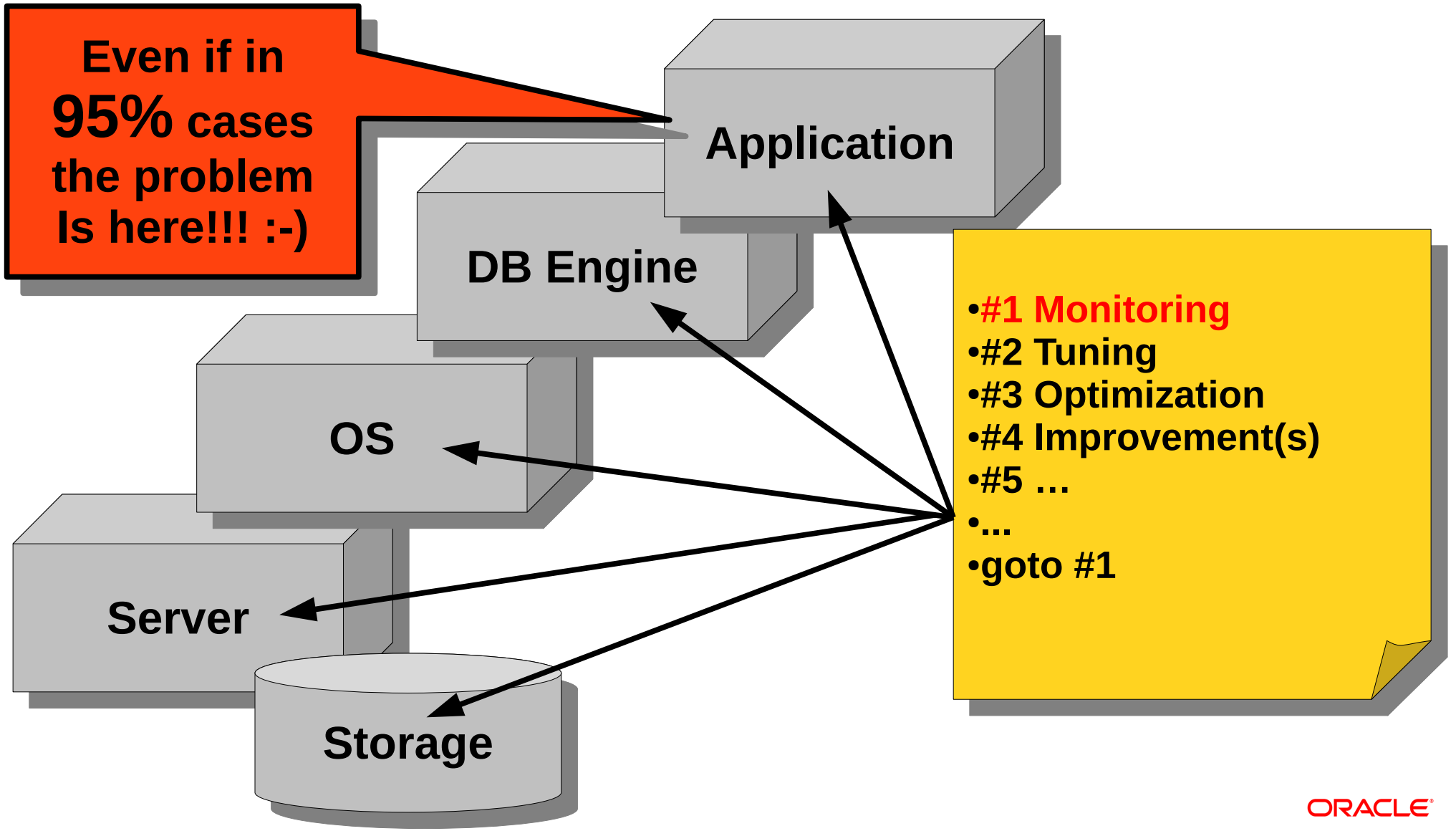
Know/ test/ check your platform limits / “features”..

- My backup is finished on Linux faster than on Solaris same HW
  - Be sure first there is **really** no more I/O activity once backup is “finished”
  - Keep in mind Linux buffering..
- Linux distro: MySQL Performance has x4 regression! Fix it!
  - How did you see it? – Our QA test is taking x4 times more time..
  - Which engine? – InnoDB..
  - What is innodb_flush_log_at_trx_commit value? – set to 1.. why?
  - Tried innodb_flush_log_at_trx_commit=2 ?.. – Oh! You fixed it!! Thanks!!
  - Wait, what did you “improve” recently in distro? – FS flushing, why?..
  - Well, the test in fact is proving that you did not “sync” on every fsync() before, that's all.. But now in your FS flushing you get it fixed ;-)

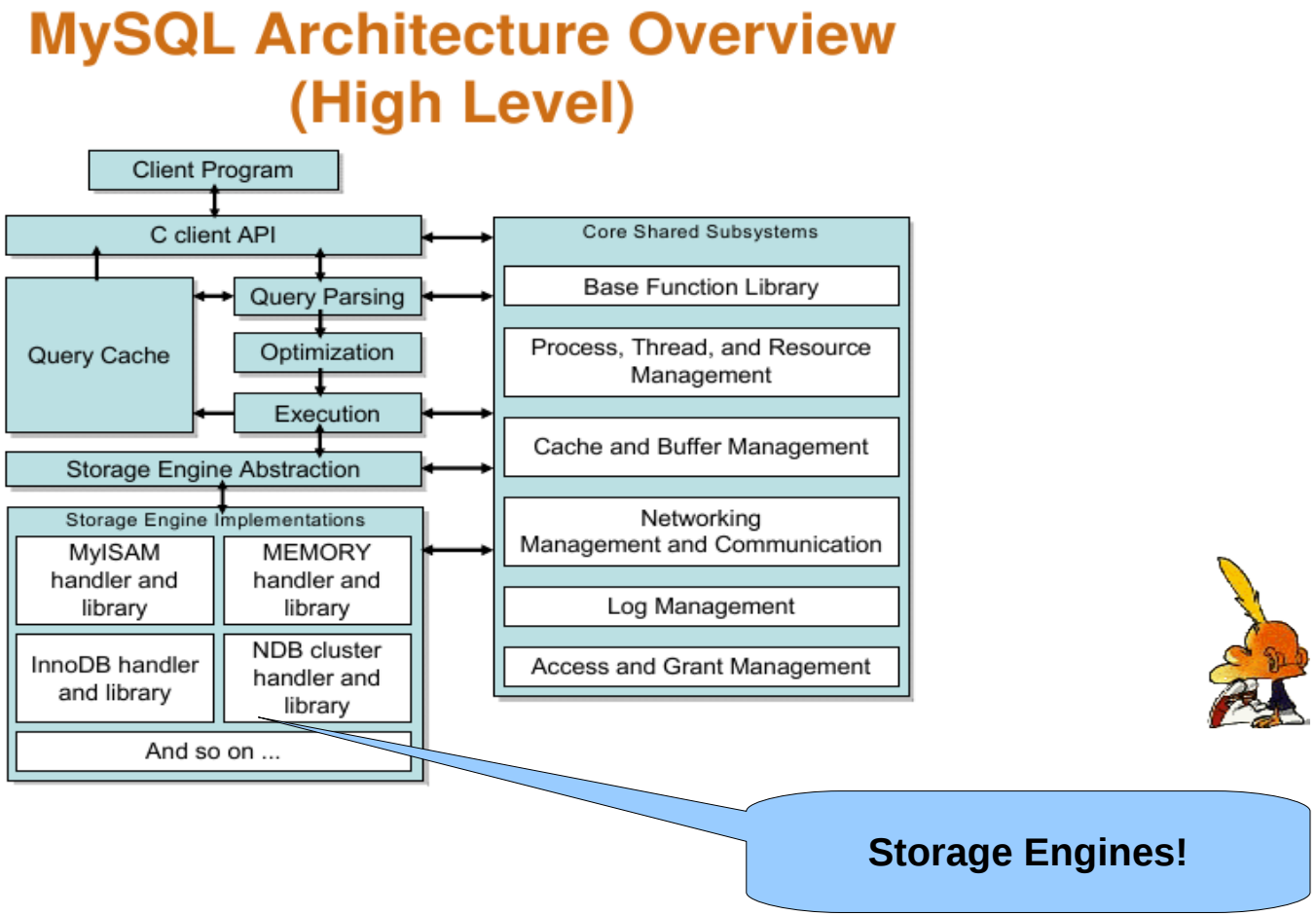
The Infinitive Loop of Database Tuning...



The Infinitive Loop of Database Tuning...



MySQL Design



MySQL Design

- Multi-Threaded database
  - Fast context switch!
  - Simplified data access!
  - Concurrent access?.. Scalability?..
- Storage Engines
  - Initially: MyISAM only
  - Then, with InnoDB: started to match expectations of a “true RDBMS” ;-)
  - Many other engines (MEMORY, CSV, NDB, PBXT, etc.)
  - CREATE TABLE ... ENGINE=<NAME_OF_ENGINE>
  - ALTER TABLE ... ENGINE=<NAME_OF_ENGINE>
  - **Did you choose a right Engine?..**

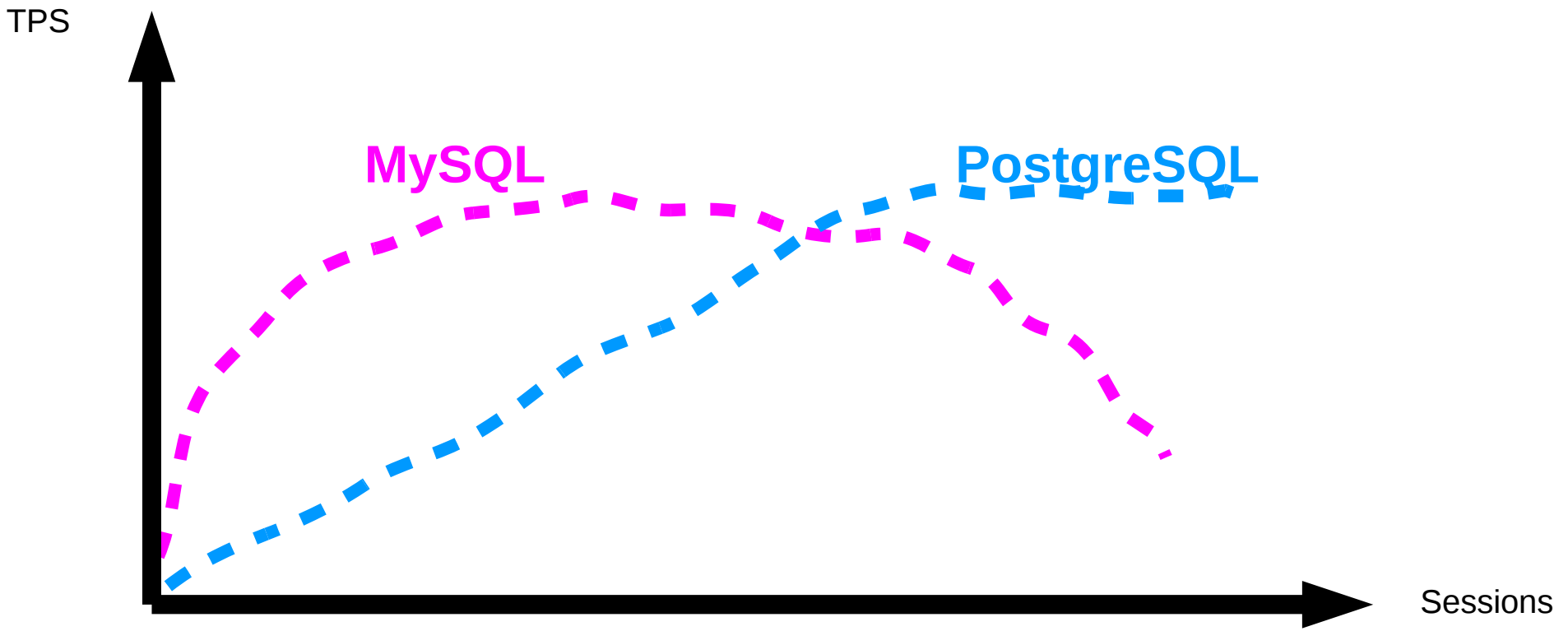
MyISAM Engine (since 1994)

- Non-transactional! / No fast recovery! :-)
- Cache
  - Index only
  - Data => FS cache
  - mysql> flush tables;
- Single Writer @Table
  - Main bottleneck! => single writer
  - Solutions: delayed inserts, low priority
- Query plan: Index forcing may be necessary (hint)
- Extremely simple and lightweight

Why MySQL + MyISAM was successful ?..

- Full Text search queries out-of-the-box!
- SELECT count(*) ... :-))
- Extremely SIMPLE!
  - my.conf => configuration parameters; mysql.server start / stop
  - Database => directory
  - Table => directory/Table.MYD, Table.MYI, Table.frm
  - $ cp Base1/Table.* /other/mysql/Base2
  - Data binary compatibility! (ex: reports via NFS)
  - Replication ready!
- Very FAST! (until some limit :-))
- RW workload is killing.. (but on 2CPU servers it was ok ;-))

RW Benchmark MyISAM vs PostgreSQL (in 2000)



InnoDB changing the game (since 2001)

- Row-level locking
- Index-only reads
- True transactions / UNDO
- Auto recovery
- Double write / Checksums
- Tablespaces or File-per-Table option
- Buffer pool
- Multi-threaded
- Currently the fastest transactional disk-based MySQL Storage Engine!

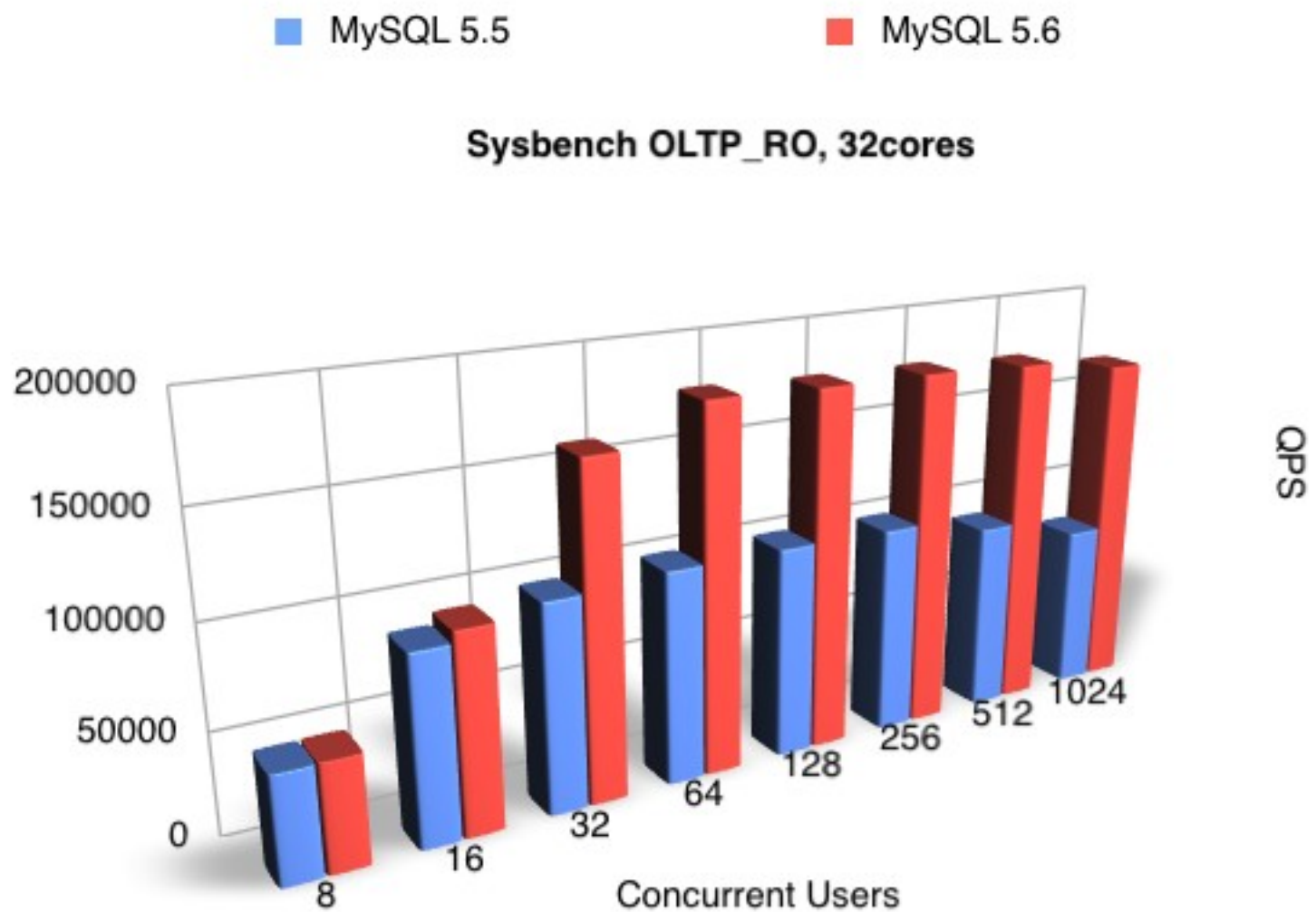
MySQL Performance (traditionally, in the past)

- Choose the right Engine for each of your table/database
  - Read-Only / Text search => MyISAM
  - Read+Write / Transactions => InnoDB
  - Short/Small Transactions + DB fits in RAM => NDB
- Tune / Optimize your queries
- Once scalability limit is reached => go for Distributed:
  - Sharding
  - Master / Slave(s) => role-based workload
  - Any other similar :-)
- Scalability = Main Performance Problem!...
  - But with Big Users on that time anyway: Google, Facebook, Amazon..

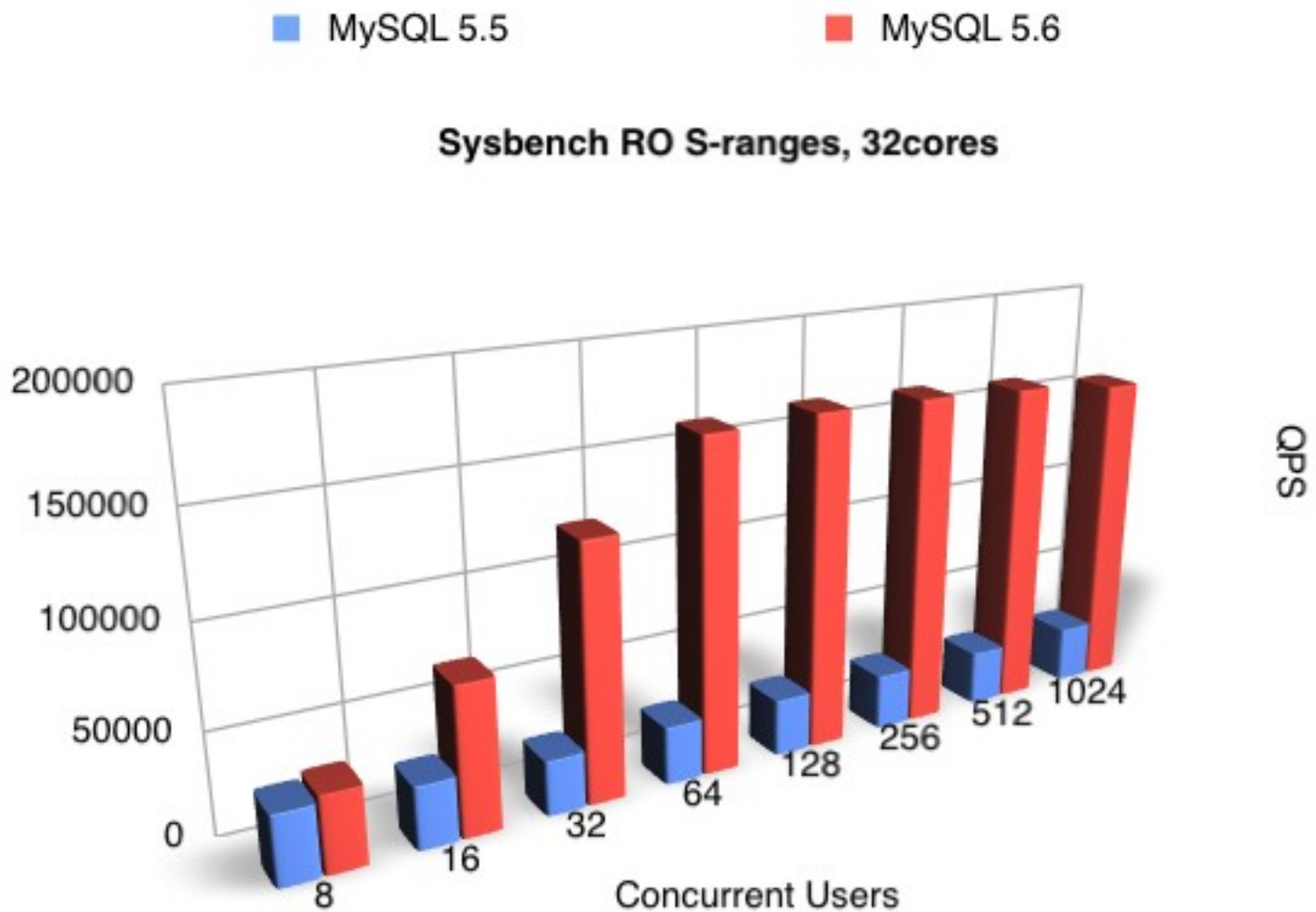
Things are changing constantly, stay tuned ;-)

- MySQL/InnoDB Scalability:
  - 2007 : up to 2CPU...
  - 2008 : up to 4CPU cores
  - 2009 : up to 16CPU cores (+Sun)
  - 2010 : up to 32CPU cores (+Oracle)
  - 2012 : up to 48CPU cores..
  - 2014 : …?? ;-)
  - NOTE: on the same HW performance is better from version to version!
- InnoDB today:
  - At least x4-8 times better performance than 2-3 years ago ;-)
  - **Capable** of over ~~100K~~ ~~300K~~ **500K** QPS(!) + FTS & Memcached

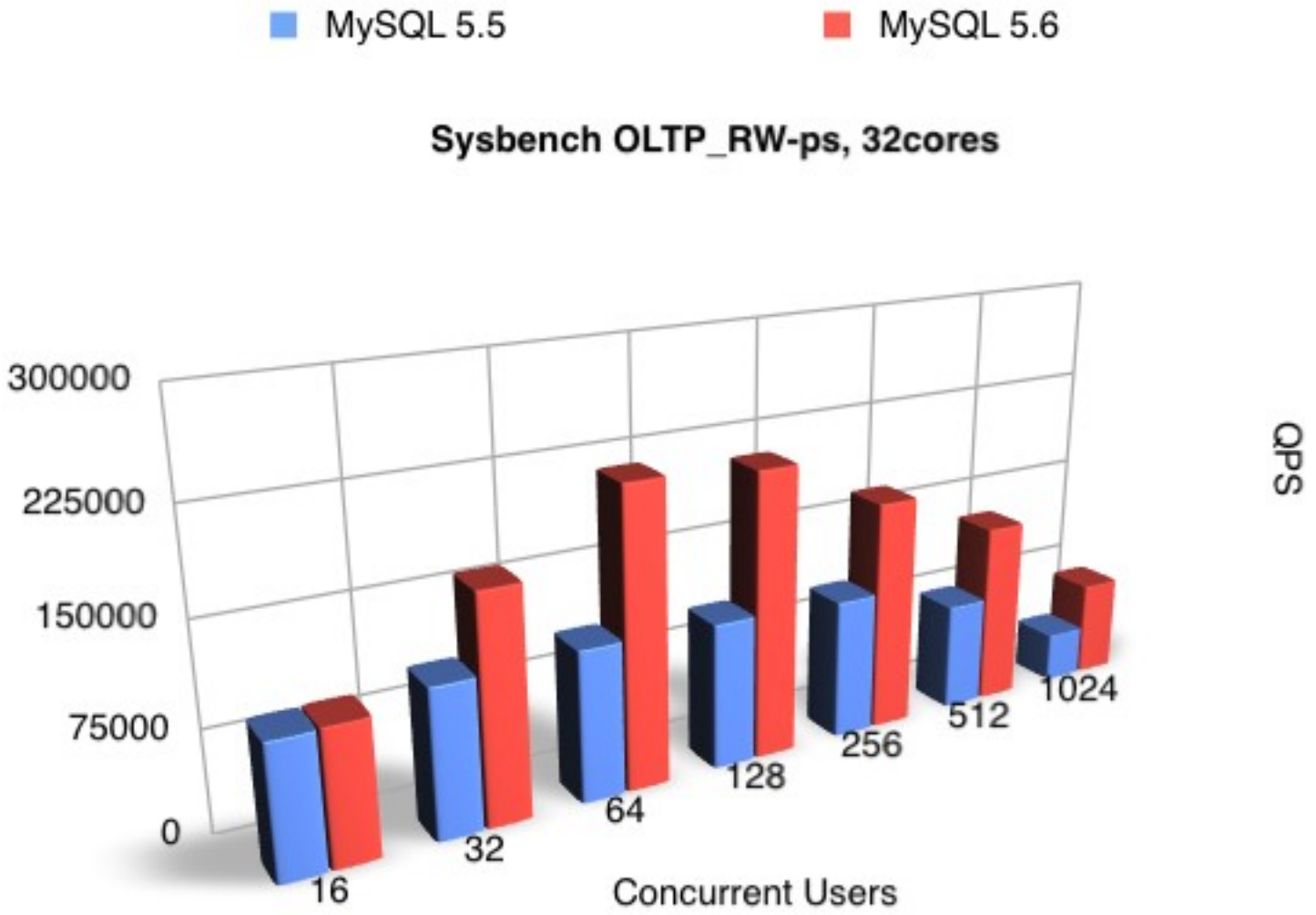
Hope you did not miss it ;-)



Hope you did not miss it ;-) (2)



Hope you did not miss it ;-) (3)



How easy is to see the same
in Production now?.. ;-)

Starting points

- What are your network limits?..
  - Latency? Max throughput? What CPU% is spent just for network?
  - Do you use prepared statements? (reducing traffic)
- Can you use persistent connections?
  - Connect / Disconnect has its limits..
  - Greatly improved in 5.6, yet more in 5.7 (**55K** Connect/s in 5.7 currently)
  - Higher QPS if more queries executed before disconnect!
  - Thread cache size matters!
- Do you use transactions on read-only requests?..
  - QPS is improved since 5.6 and yet more in 5.7
  - But you cannot get a rid from a traffic overhead due BEGIN / COMMIT exchanges

Analyzing MySQL Workload

- Understand the load first :
  - Hot queries <== could be improved?..
  - Hot tables / files <== storage ok? DB design?..
  - Bad query execution plans.. <== improve, force index, etc.
  - Row Lock contentions due Application Design <== will not scale..
  - Deadlocks due Application Logic.. <== will not scale..
  - **NOTE**: be sure you're not hitting some HW / OS limits (and MySQL is in fact out of scope ;-))

Performance Schema since MySQL 5.6: Gold Mine!

- Query digest (enabled by default) :
  - SELECT all queries with execution time > N ms
  - SELECT all queries having > N rows read
  - SELECT queries having table scans, not using indexes, etc..
- FILE_IO (enabled by default) :
  - Time spent on every IO operation for every database file
  - Amount of each kind of IO operations for every file
- Table Locks (enabled by default) :
  - See which tables are the most accessed
- ==> Just with these 3 metrics you already have an idea if things are still going well or not.. - and **MEM** is excellent here! ;-)

Classic MySQL Monitoring

- SHOW Commands:
  - mysql> status ;
  - mysql> show global status ;
  - mysql> show processlist ;
  - mysql> show engine innodb status ;
  - mysql> show engine innodb mutex ;
  - INFORMATION_SCHEMA.* , InnoDB METRICS table, etc..
- Important :
  - only PFS instrumentation / query is truly lock free..
  - every query during its execution uses 1 CPU core full time!
  - excessive requesting may significantly lower an overall performance!

So far, what do you have to look on?..

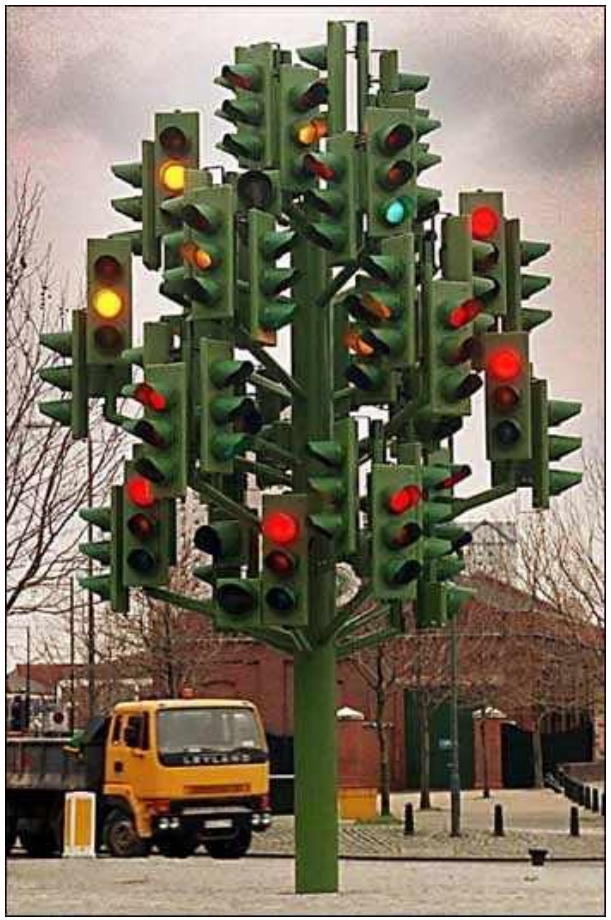
- MySQL Server general:
  - Query/sec, Select/sec, Commit/sec, Connect/sec, Connections, Abort...
- InnoDB:
  - BP usage/ dirty%/ page hit%
  - Checkpoint Age, REDO logs rates (MB/sec, Writes/sec, Sync time/sec)
  - Adaptive Flushing rates, Sync Flushing rates, Sync Flushing waits
  - LRU Flushing stats, User Threads LRU Flushing, ..
  - History List Length (purge)
  - Mutex Waits (InnoDB, PFS)
  - File IO Waits (PFS)
  - etc...

Suspecting a problem?.. - Benchmark!

- Have a clear goal!
  - Otherwise: I've obtained all these results, and now... so what?..
- 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
  - Or any, if the goal to prove there are not depending ;-)
- Well, just keep thinking about what you're doing ;-)

Test Workload

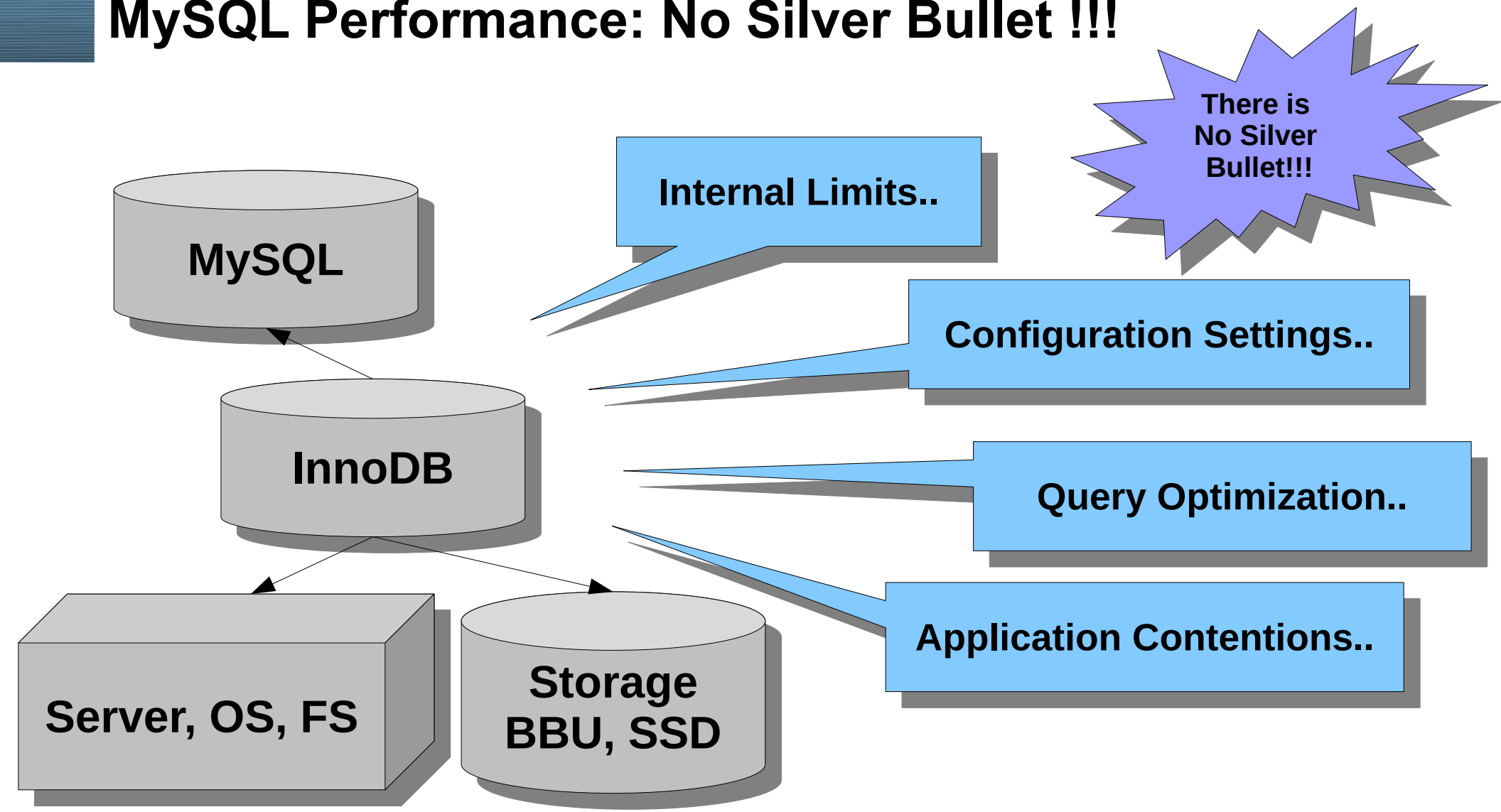
- Before to do something complex...
  - Be sure first you're comfortable with “basic” operations!
  - Single table?
  - Many tables?
  - Short queries?
  - Long queries?
- Remember: any complex load just represents a mix of simple operations..
  - So, start from as simple as possible..
  - And then increase complexity progressively..



Popular “Generic” Test Workloads @MySQL

- Sysbench
  - OLTP, RO/RW, 1-table, since v0.5 N-table, lots 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
- linkbench (Facebook)
  - OLTP, RW, very intensive
- DBT3
  - DWH, complex heavy query, loved by Optimizer Team ;-)

MySQL Performance: No Silver Bullet !!!

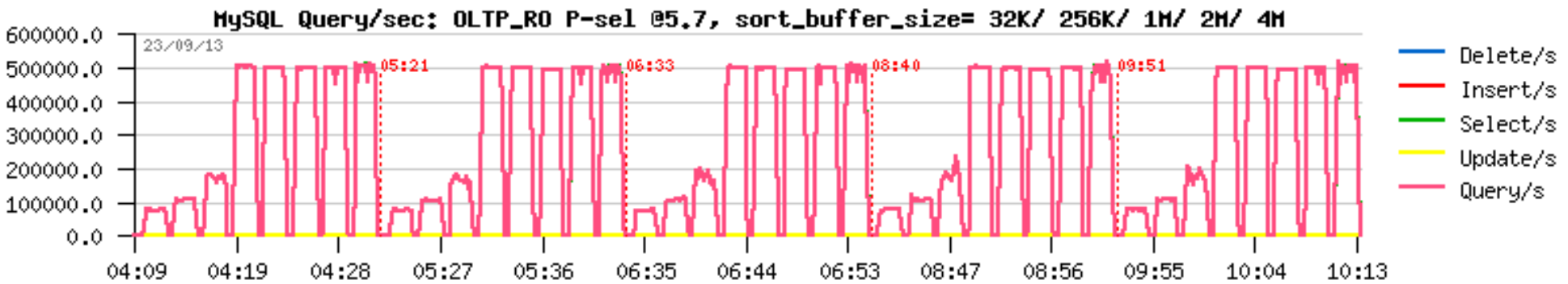


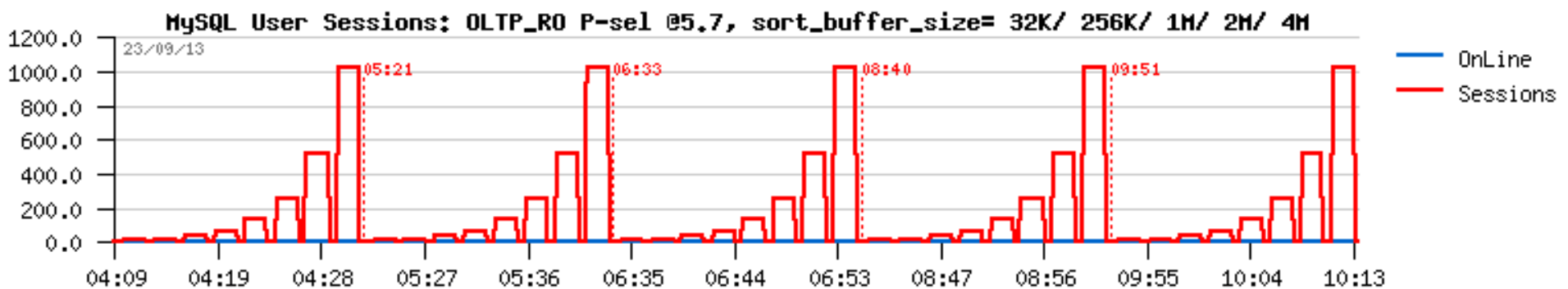
MySQL Config settings

- Ask yourself right questions and start with some basic params:
  - Buffer Pool size / Buffer Pool Instances
  - Double write buffer? Checksums?
  - innodb_flush_log_at_trx_commit= 1 / 2 ??
  - Flush Method = O_DIRECT + ASYNC
  - Binlog Sync? - binlog group commit is since MySQL 5.6 only!
  - File per table?
  - IO capacity = 2000
  - Etc..
- Then adapt then to discovered your HW/OS and MySQL Internal limits!..

Example: Sort Buffer Size

- OLTP_RO Point-Selects 8-tables
  - Sort Buffer Size: 32K, 256K, 1M, 2M, 4M

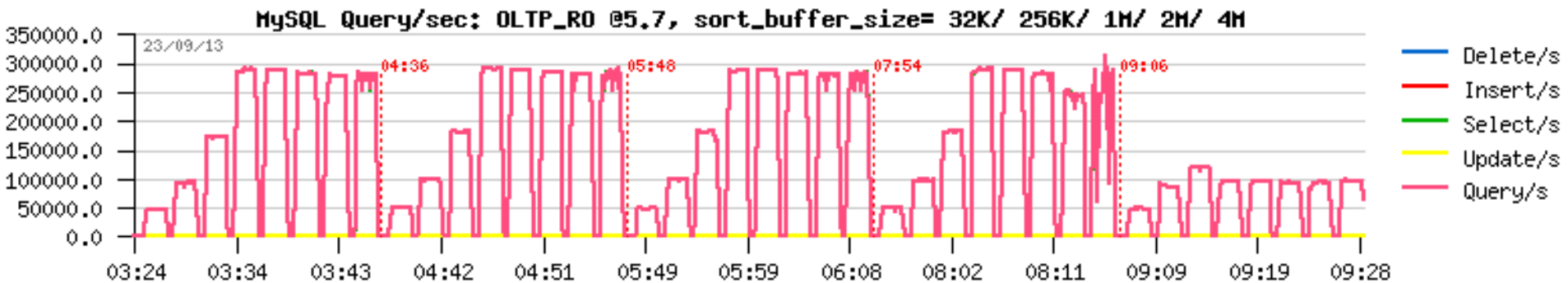


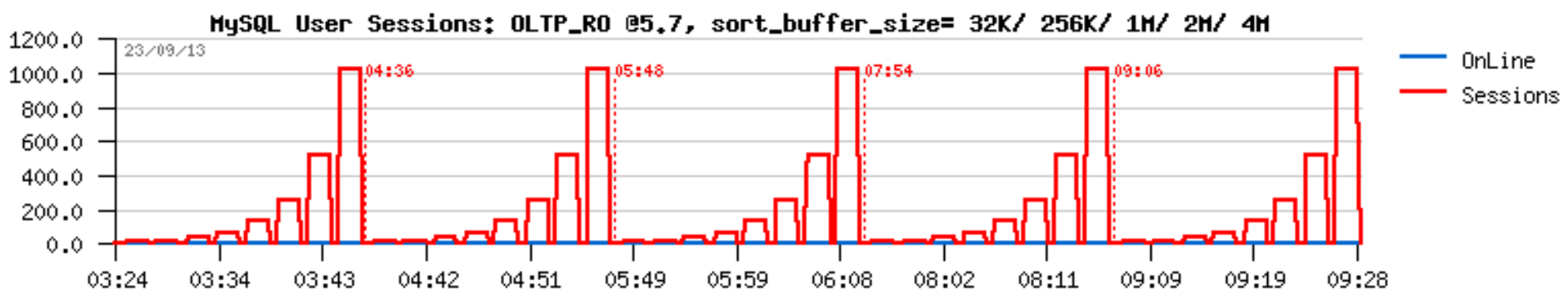


ORACLE

Example: Sort Buffer Size (2)

- OLTP_RO 8-tables
  - Sort Buffer Size: 32K, 256K, 1M, 2M, 4M

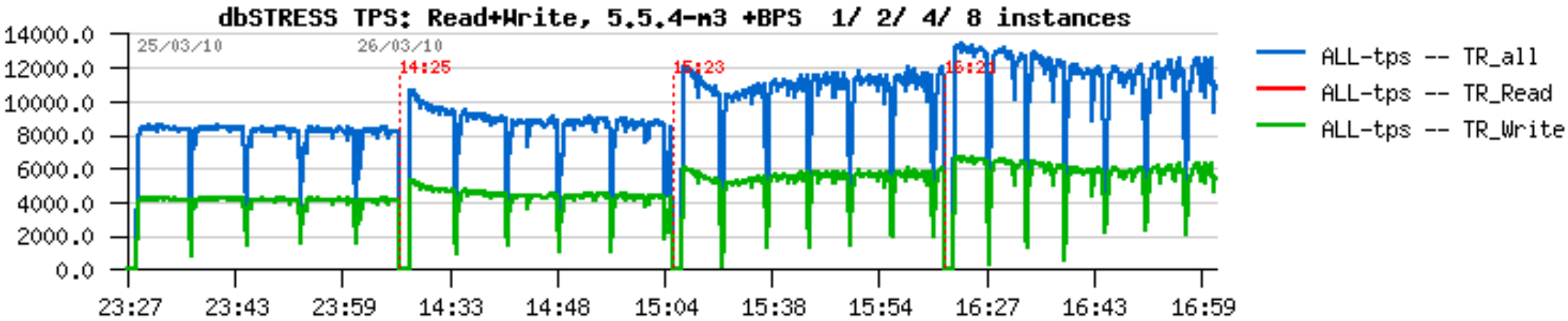




ORACLE

Example: Buffer Pool Instances

- RW intensive workload:
  - BP instances = 1/ 2/ 4/ 8

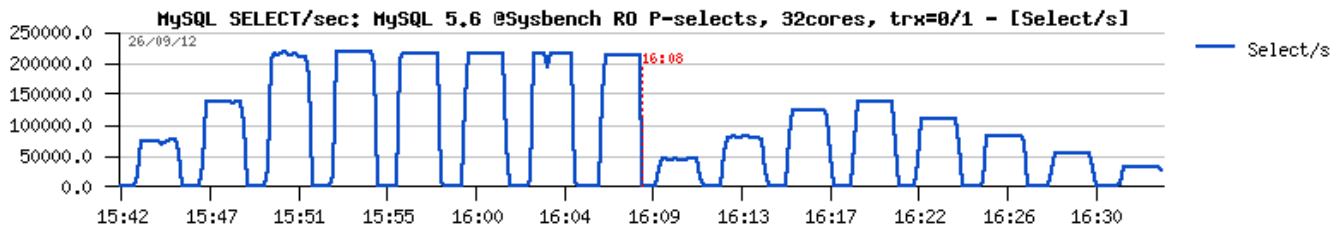


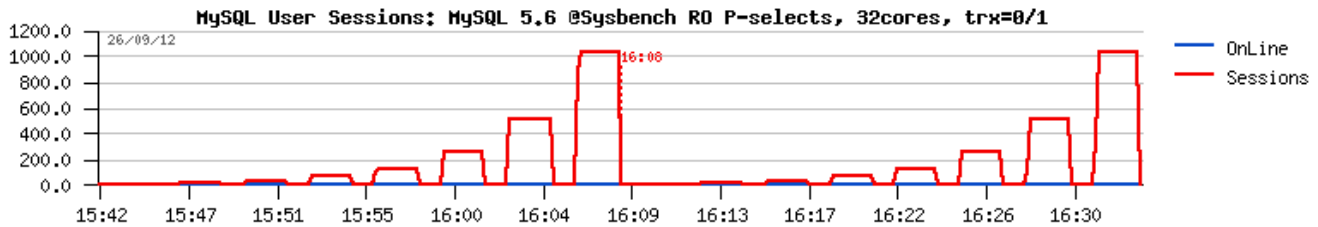
Workload: Read-Only oriented

- Bigger Buffer Pool (BP) is better
  - BP < dataset = IO-bound
- TRX list (kernel_mutex, since 5.6: trx_sys mutex)
- Read view
- Auto-commit or transactions?..
  - Grouping **many** queries within a single transaction may also largely reduce MDL locking, but still keep them **short** ! (check with PFS)
- Prepared statements
  - Observed 10% performance improvement in 5.6 (while Parser time is not more than 3% according to profiler)..
- Read-Only transactions!

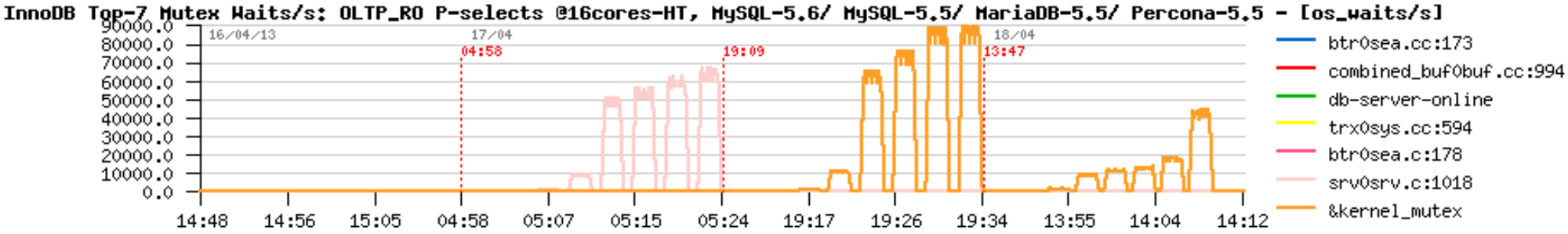
InnoDB: Read-Only Transactions in 5.6

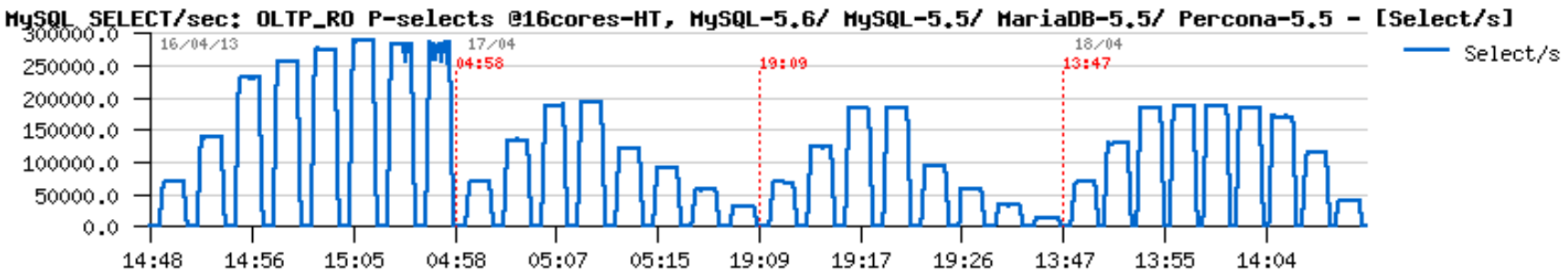
- Sysbench OLTP_RO Point-Selects:
  - Concurrent user sessions: 1, 2, 4 .. 1024
  - Using of transactions in sysbench = 0 / 1

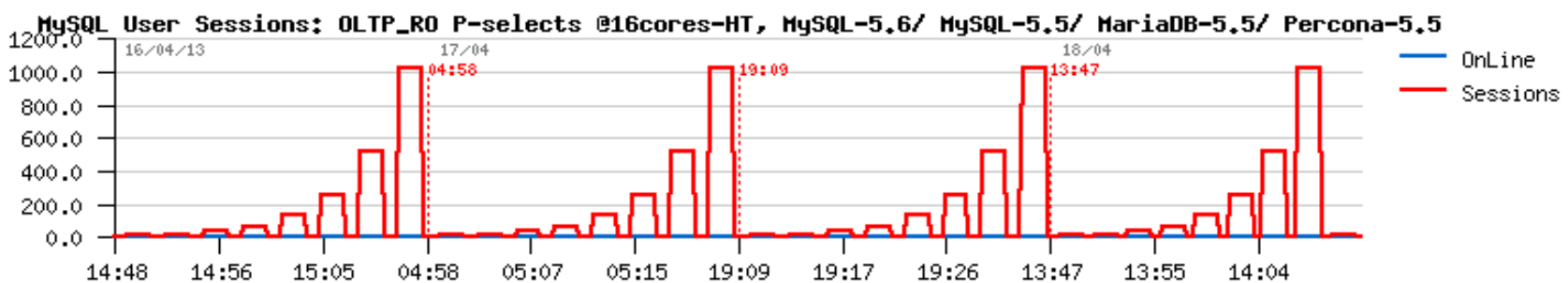




InnoDB: Read-Only Transactions in 5.6 (Apr.2013)



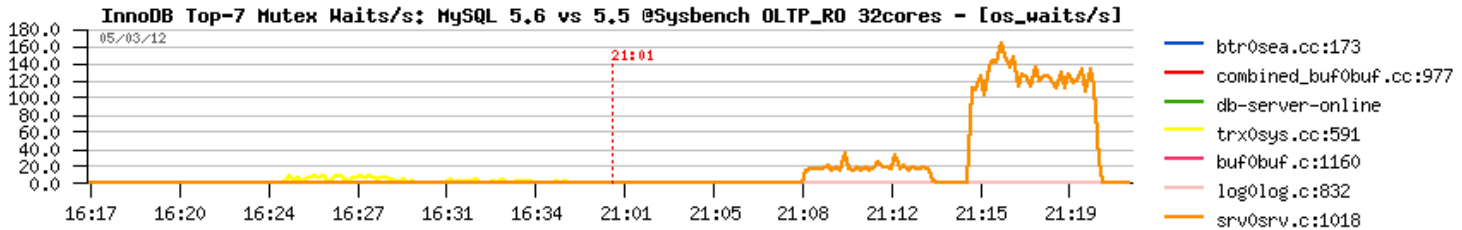


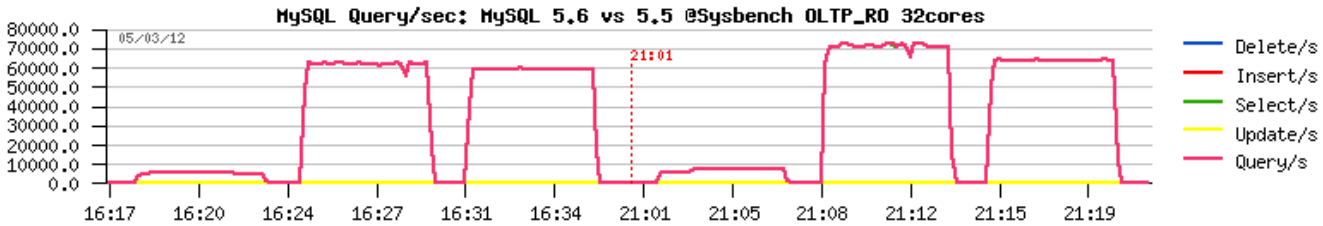


ORACLE

InnoDB : false sharing of cache-line = true killer

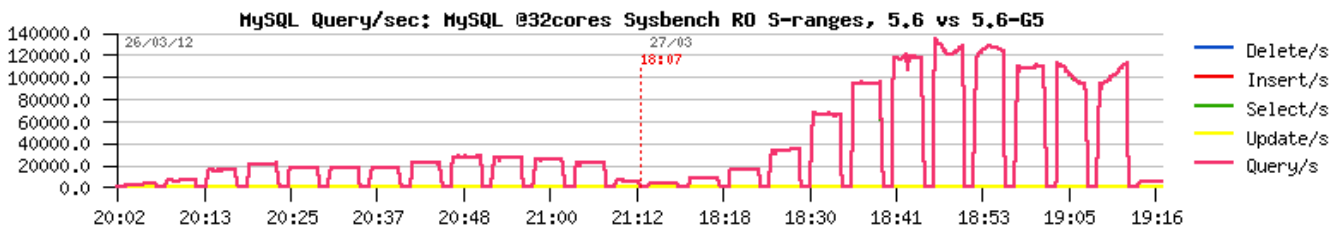
- RO or RW Workloads
  - Same symptoms in 5.5 & 5.6 : no QPS improvement between 16 and 32 user sessions:

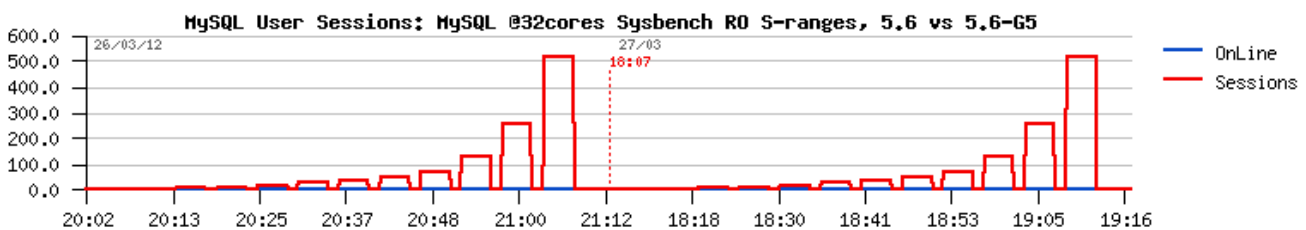




InnoDB : false sharing of cache-line fixed!

- RO or RW Workloads
  - “G5” patch! :-)
  - Over **x2(!)** times better on Sysbench OLTP_RO,
  - **x6(!)** times better on SIMPLE-Ranges!
  - NOTE: the fix is not applicable on 5.5..





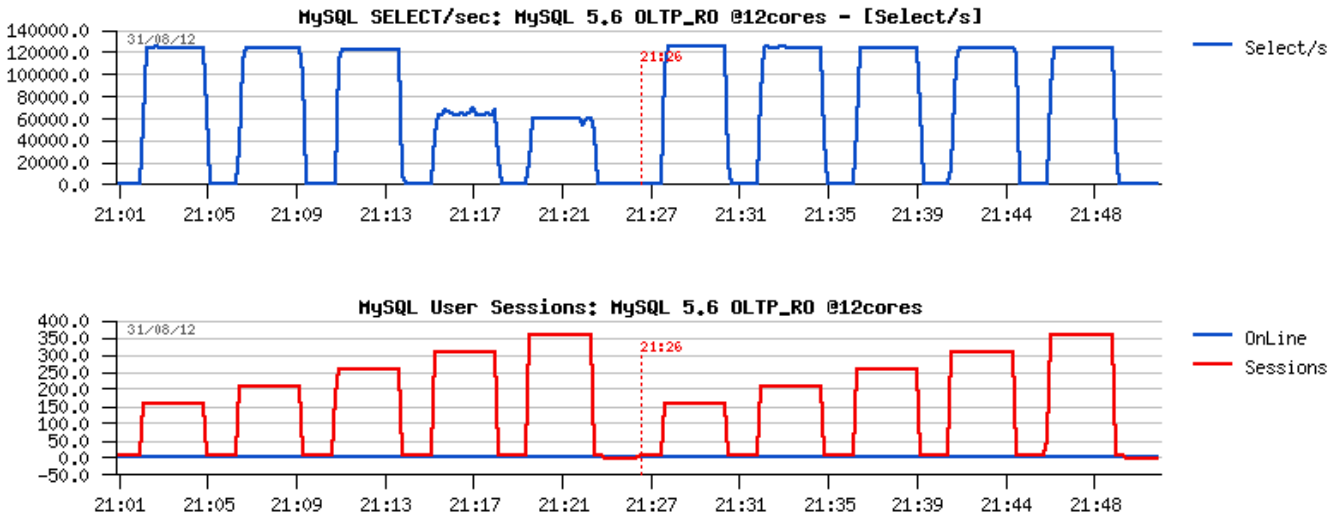
ORACLE

MySQL Internals: “killer” LOCK_open mutex

- MySQL 5.5 and before:
  - Keep “table_open_cache” setting big enough!
  - Monitor global status for '%opened%'
  - Once this contention become the most hot – well, time to upgrade to 5.6 ;-))
- Since MySQL 5.6:
  - Fixed: several table open cache instances
  - But it doesn't mean you can use a small “table_open_cache” either ;-)
  - Monitor PFS Waits!
  - Monitor “table_open_cache%” status variables!
  - Keep “table_open_cache_instances” at least bigger than 1

MySQL 5.6 Internals : low table_open_cache

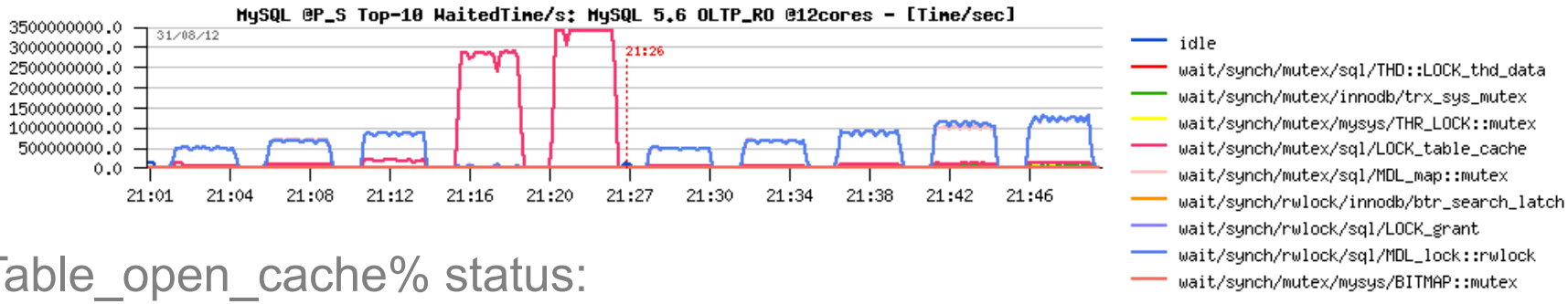
- MySQL 5.6 :
  - Not big enough “table_open_cache” setting



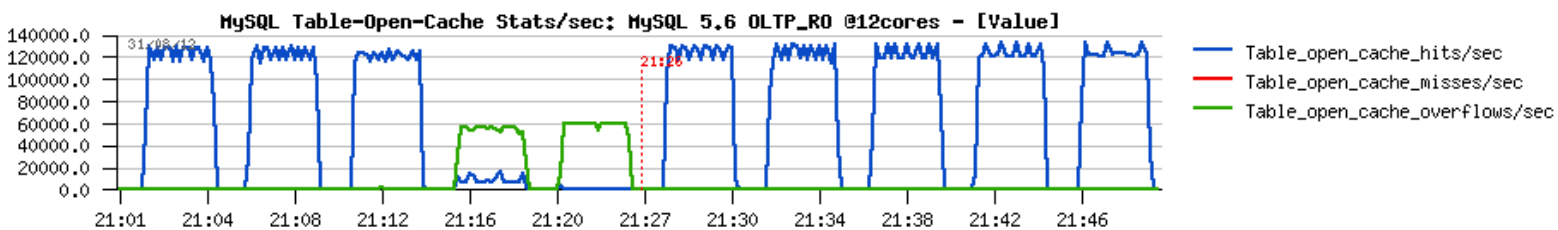
ORACLE

MySQL 5.6 Internals : low table_open_cache (2)

- MySQL 5.6 :
  - Not big enough “table_open_cache” setting
  - PFS Waits monitoring: LOCK_table_cache become the most hot:

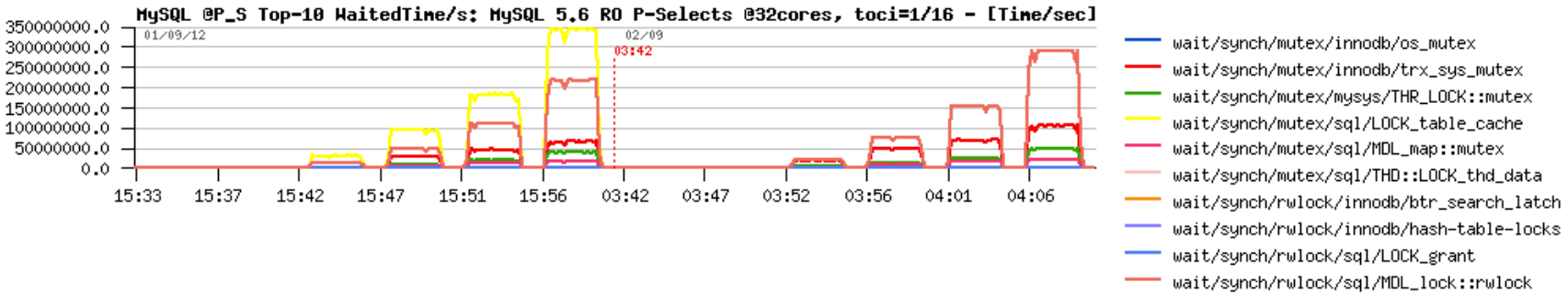


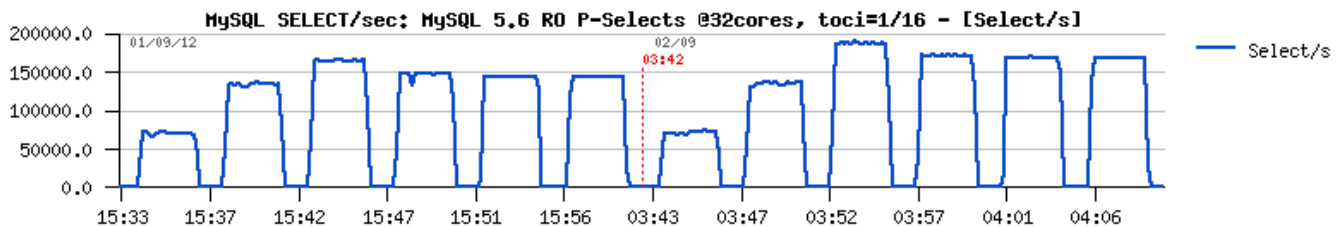
  - Table_open_cache% status:

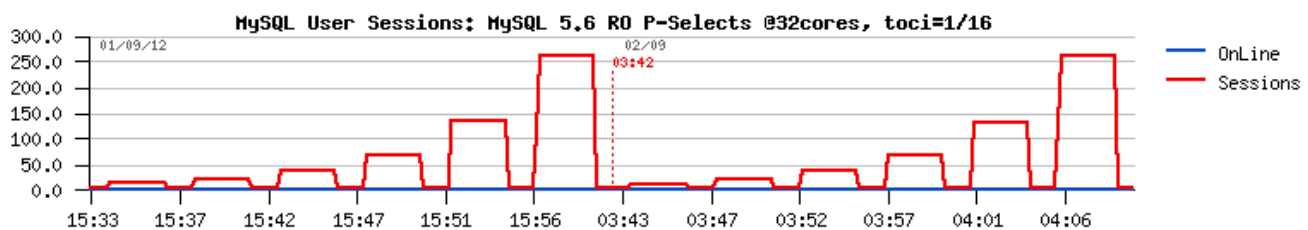


MySQL 5.6 Internals : table_open_cache_instances

- MySQL 5.6 :
  - When LOCK_table_cache wait is on top, the gain is usually well visible:







ORACLE

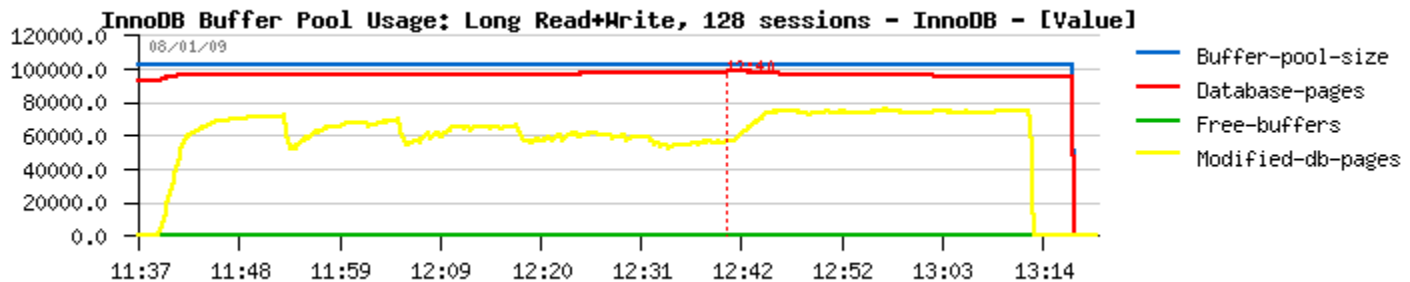
Workload: Read-Write

- RW activity
  - Updates only? Insert? Delete? R/W %ratio?
- Bigger Buffer Pool (BP) is still better
  - BP < dataset = IO-bound Reads(!) or R+W
  - BP > dataset = CPU-bound or IO-bound Writes(!)
- REDO size matters a lot! (up to 2TB in 5.6)
- Adaptive Flushing matters a lot!
- LRU flushing matters a lot as well!
- Tip: Neighbor Pages flushing = off / on

But let me tell you now the
whole story first! ;-)

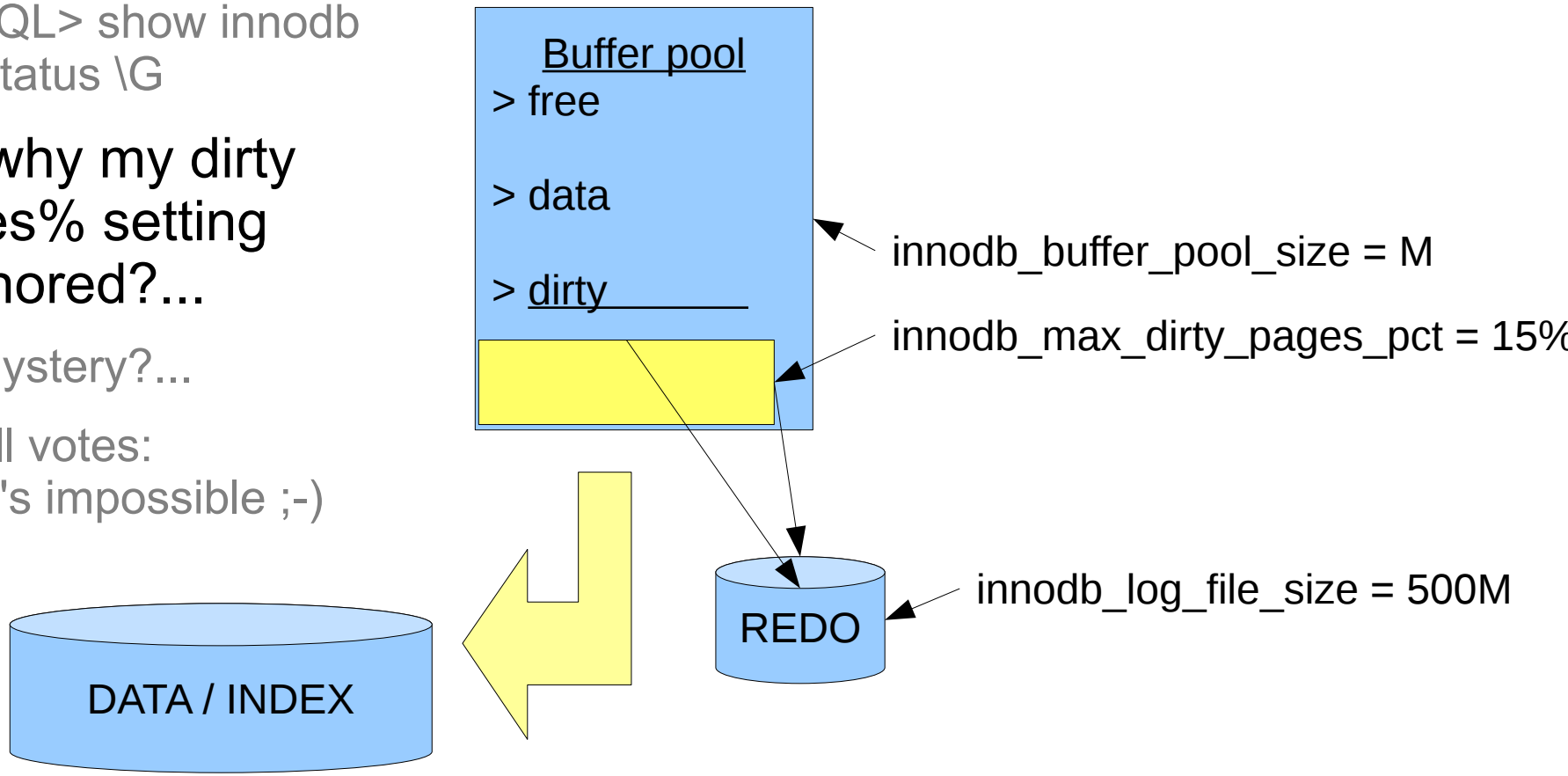
Jan.2009 : Long RW Intensive Test

- RW Workload:
  - 128 concurrent users, 500M REDO, dirty pages= 15%
  - But let's get a look on the real state of BP:



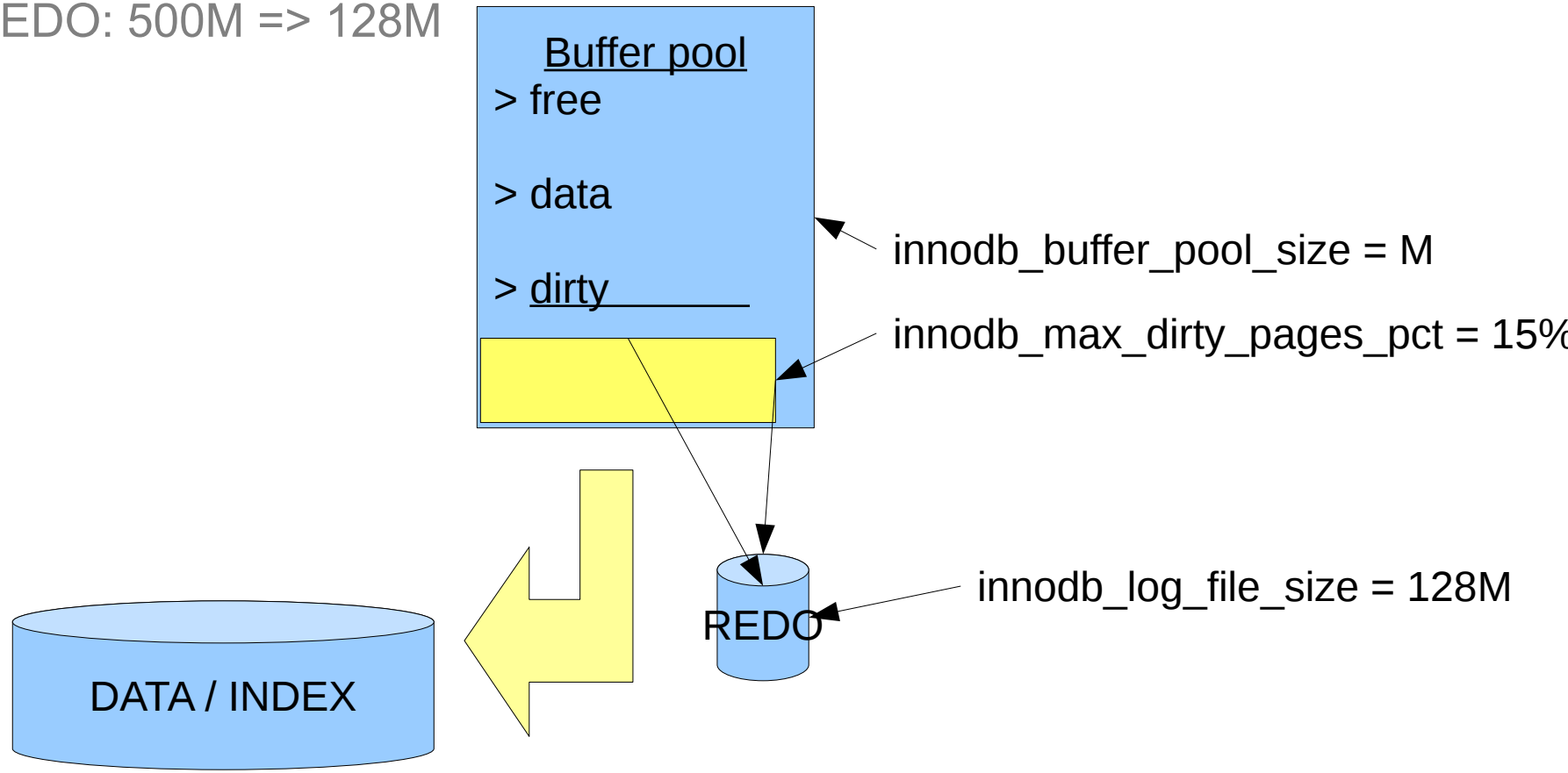
InnoDB Internals: Dirty pages

- How does it work?..
  - SQL> show innodb status \G
- But why my dirty pages% setting is ignored?...
  - Mystery?...
  - All votes: it's impossible ;-)



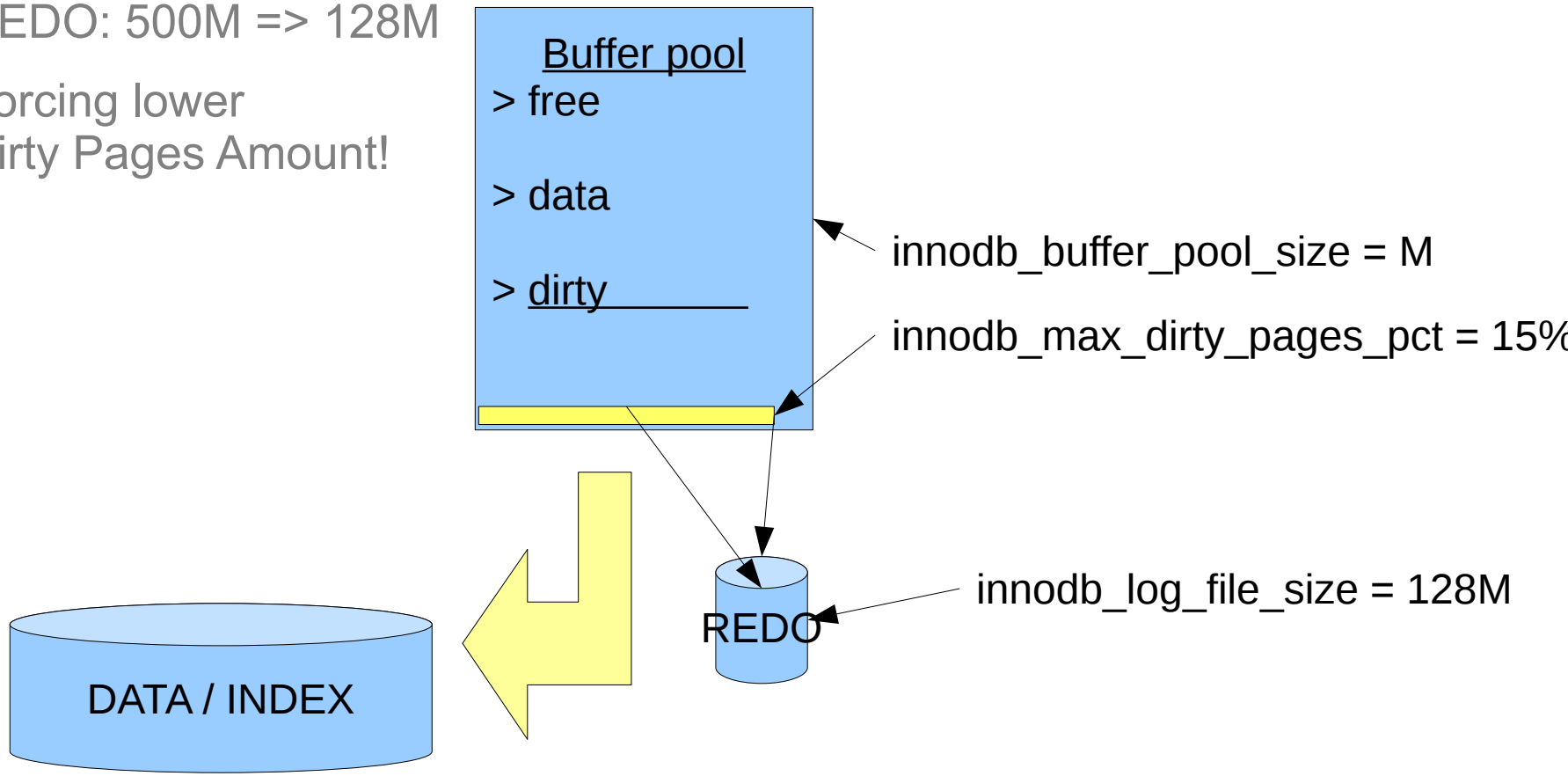
InnoDB Internals: Dirty pages and REDO?..

- What if I'll reduce REDO size now?..
  - REDO: 500M => 128M



InnoDB Internals: Dirty pages and REDO?..

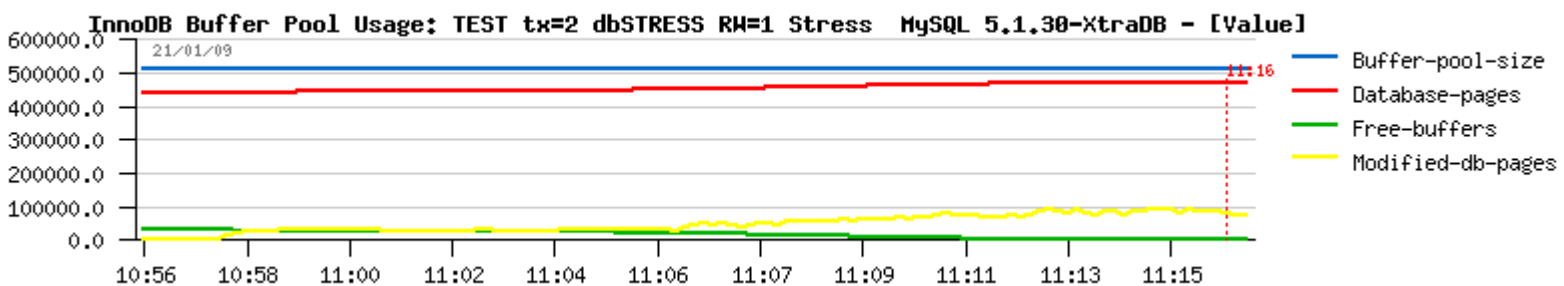
- What if I'll reduce REDO size now?..
  - REDO: 500M => 128M
  - Forcing lower Dirty Pages Amount!



Any Changes on RW Test now?..

- REDO = 500M

- REDO = 128M

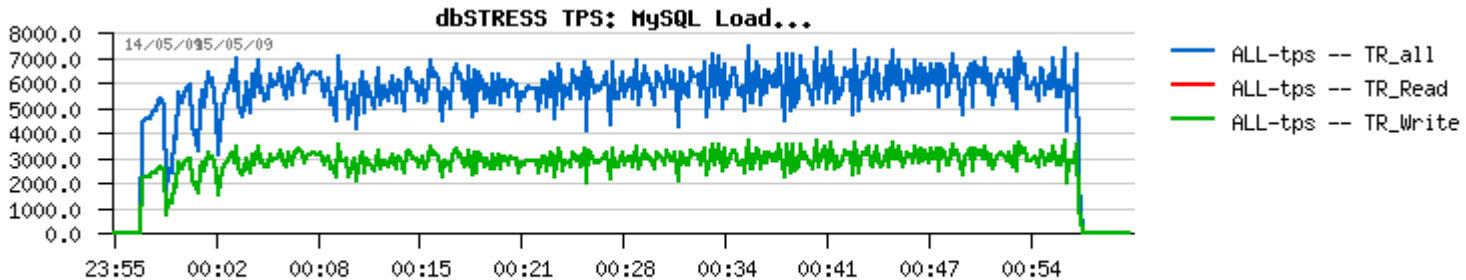


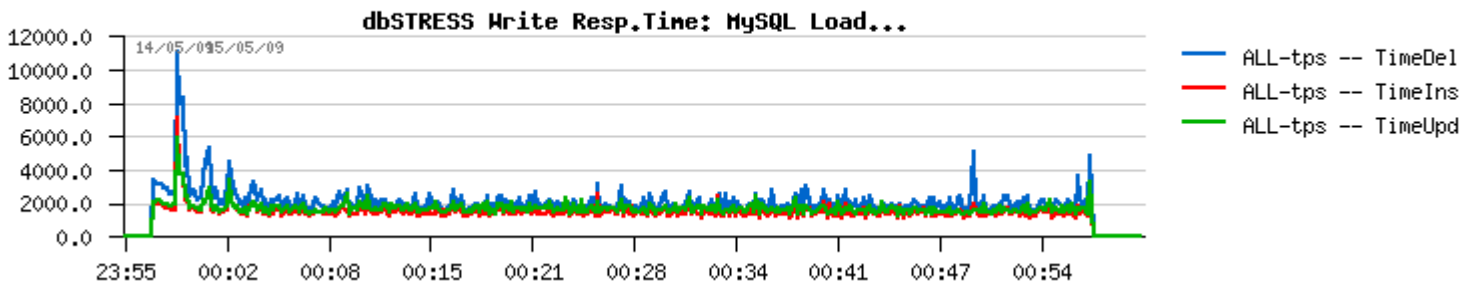
Fine, but..

- Remained questions:
  - Why finally Dirty Pages% setting is completely ignored?...
  - While, after all, any dangers to have many dirty pages?...
  - And what is the impact of REDO logs size?..

InnoDB Internals: Impact of REDO size

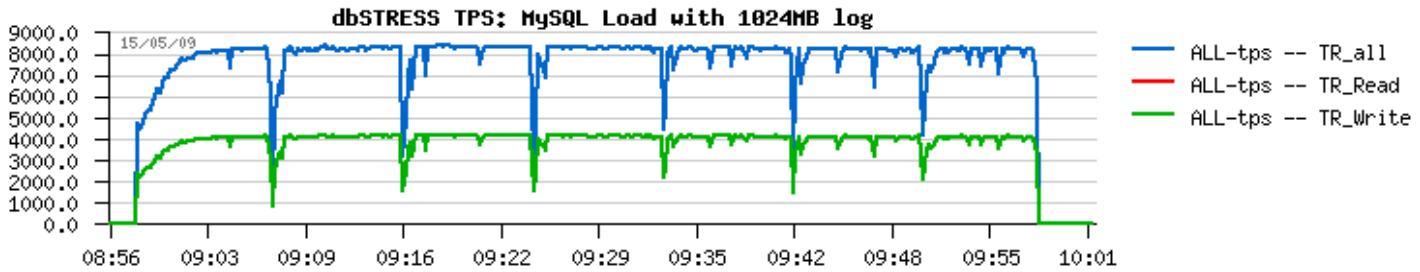
- RW Intensive Load
  - REDO size = 128M

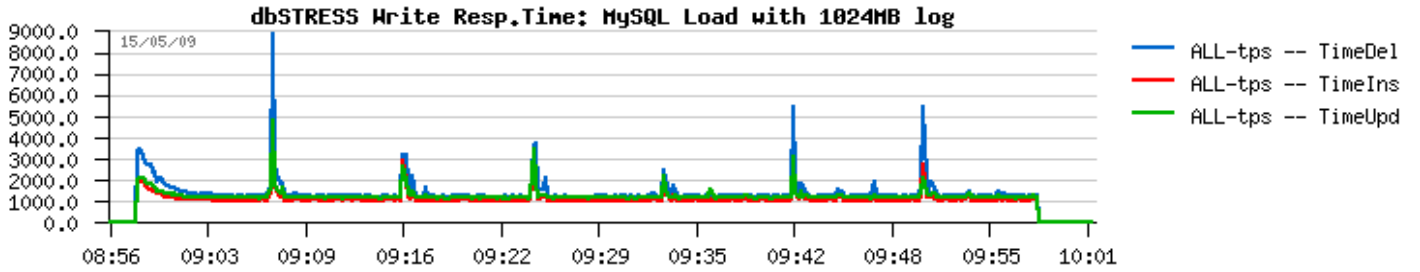




InnoDB Internals: Impact of REDO size

- RW Intensive Load
  - REDO size = 1024M





ORACLE

InnoDB Internals: Impact of REDO size

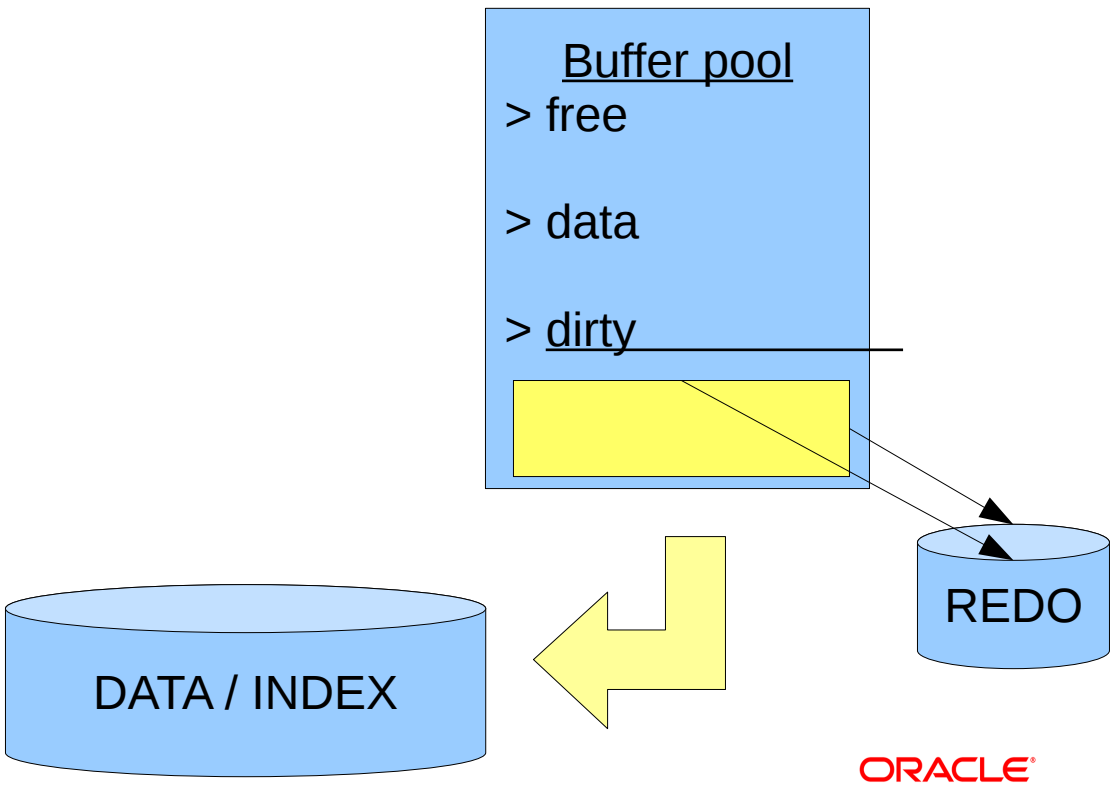
- RW Intensive Load
  - REDO size = 128M => 1024M
  - Result: 6000 TPS => 8000 TPS! **30%** better!!!
  - For such an improvement we may ignore Dirty Pages% ;-))
- But **:** **WHY** these TPS drops?...

InnoDB Internals: Analyzing the code..

- Master thread logic:

Master Thread

```
loop: //Main loop
...
if( dirty pct > limit)
  flush_batch( 100% IO);
...
do {
  pages= trx_purge();
  if( 1sec passed )  flush_log();
} while (pages);
...
goto loop;
```

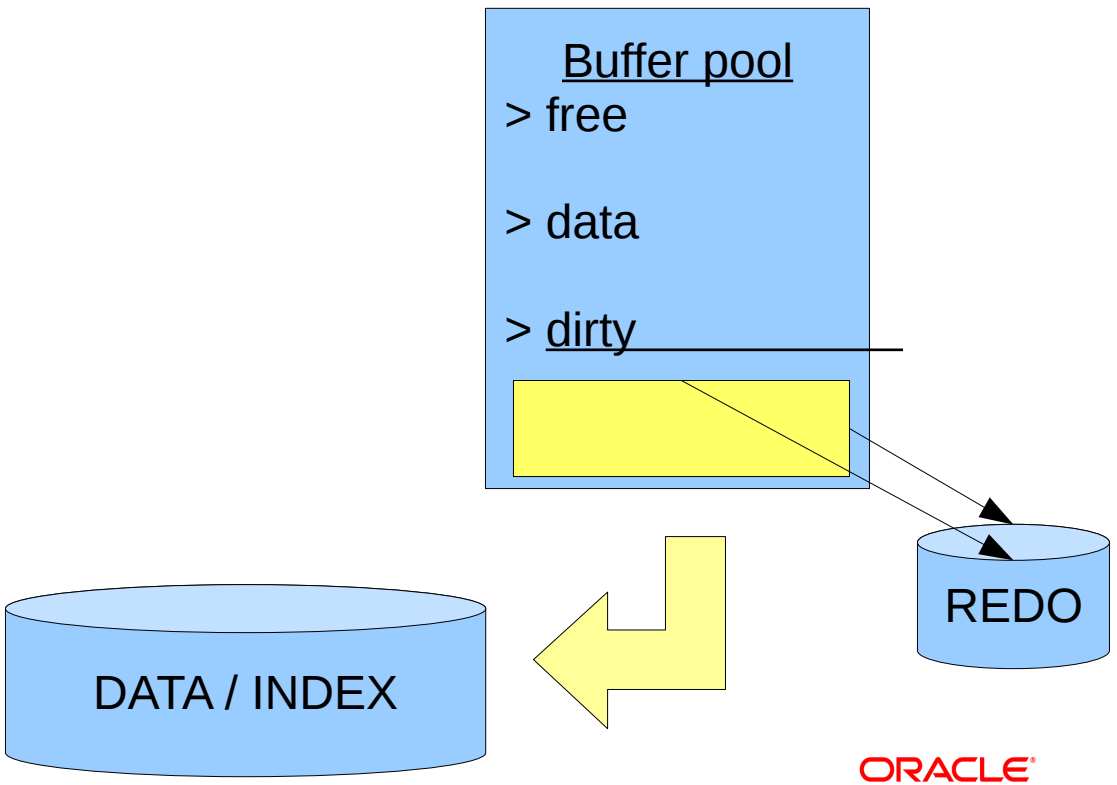


InnoDB Internals: Analyzing the code..

- Master thread may never leave purge loop!!!

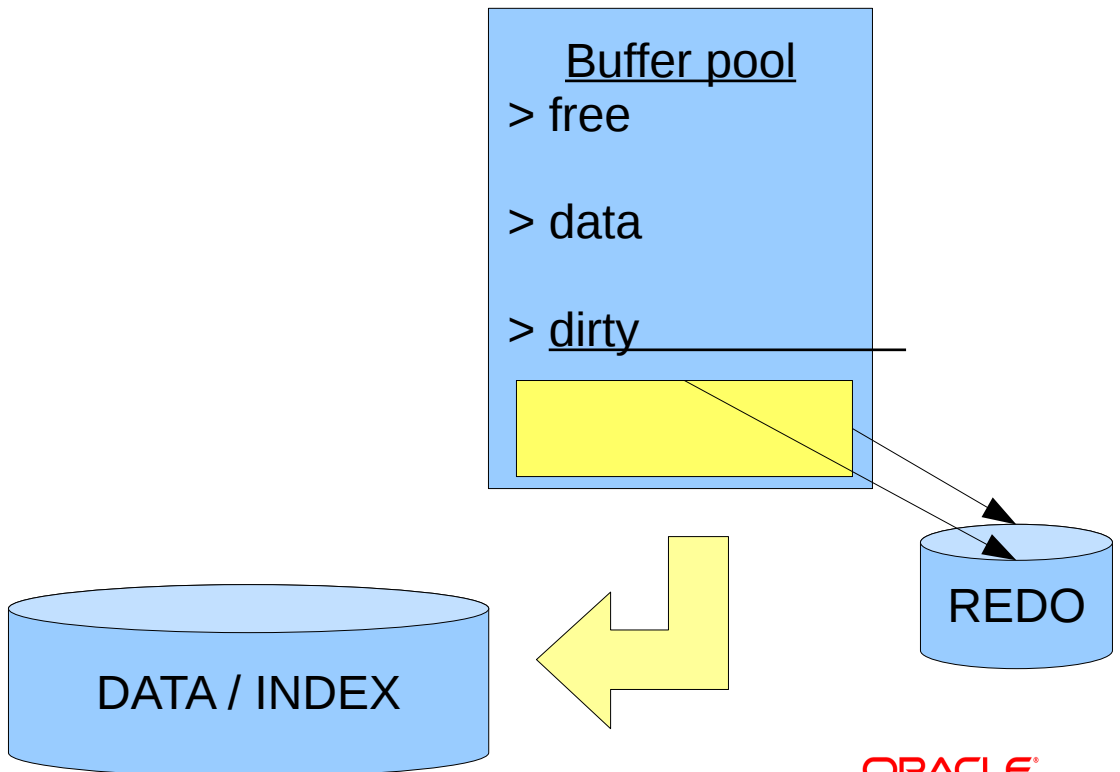
Master Thread

```
loop: //Main loop
...
if( dirty pct > limit)
  flush_batch( 100% IO);
...
do {
  pages= trx_purge();
  if( 1sec passed )  flush_log();
} while (pages);
...
goto loop;
```



InnoDB Internals: Analyzing the code..

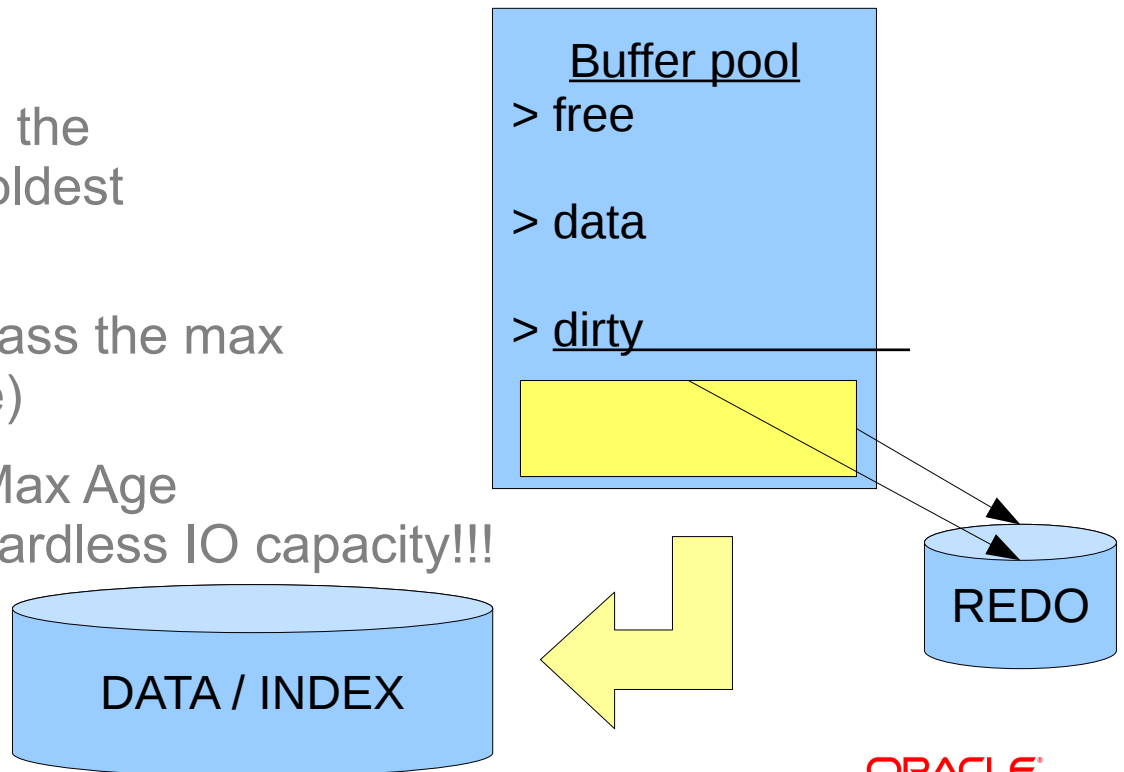
- But if Master thread is never leave purge loop...
- Who is then flushing Dirty Pages?...



InnoDB Internals: Analyzing the code..

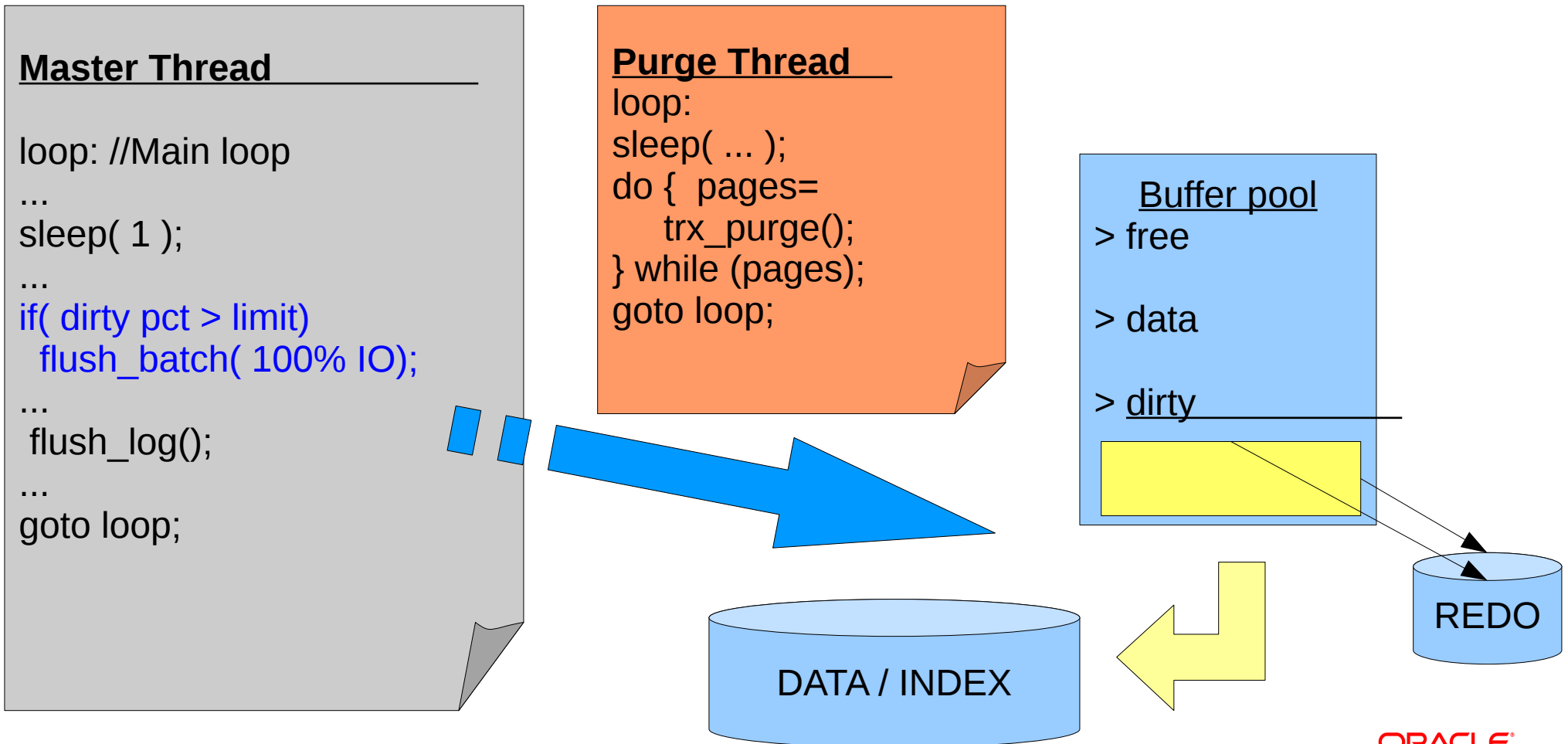
- But if Master thread is never leave purge loop...
  - Who is then flushing Dirty Pages?...
- Redo log constraints:
  - Cyclic, need free space
  - Checkpoint Age: diff between the current LSN in redo and the oldest dirty page LSN
  - Checkpoint Age cannot out-pass the max checkpoint age (redo log size)
  - If Checkpoint Age >= 7/8 of Max Age
    => Flush ALL dirty pages regardless IO capacity!!!

    ("*Furious Flushing*")



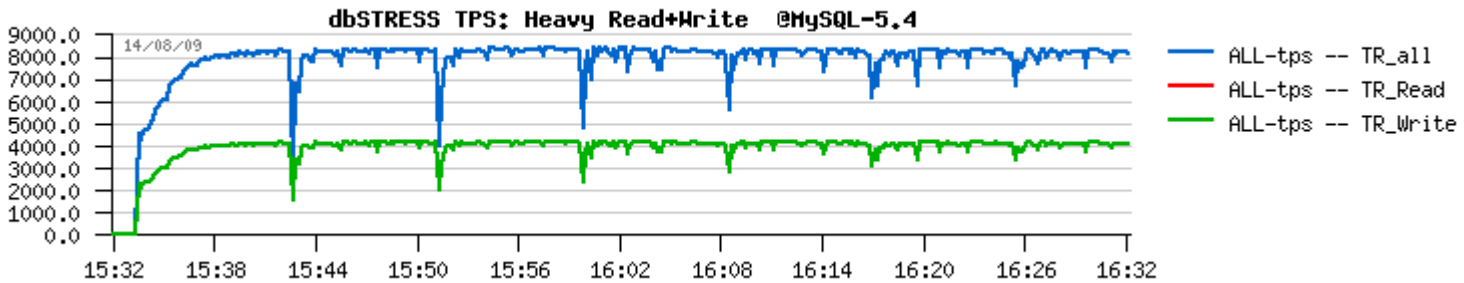
InnoDB Internals: Introducing Purge Thread

- Purge Thread is the MUST !!!

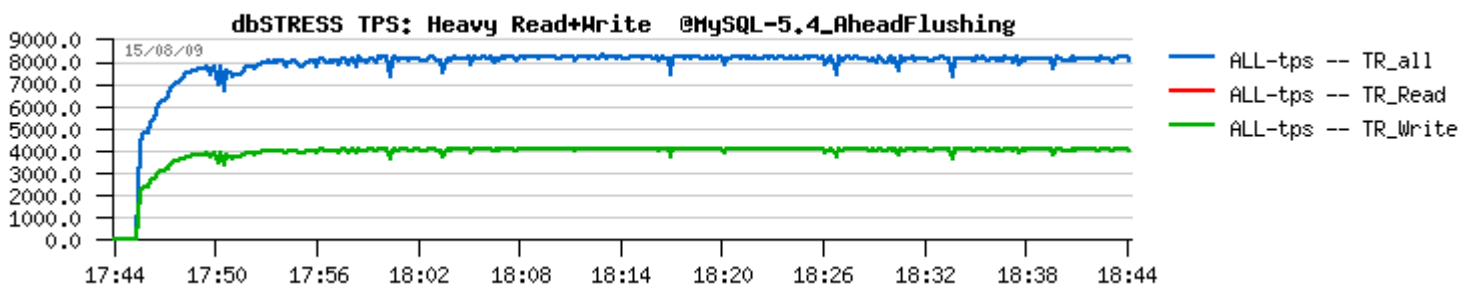


Performance with Purge Thread

- MySQL 5.4 :

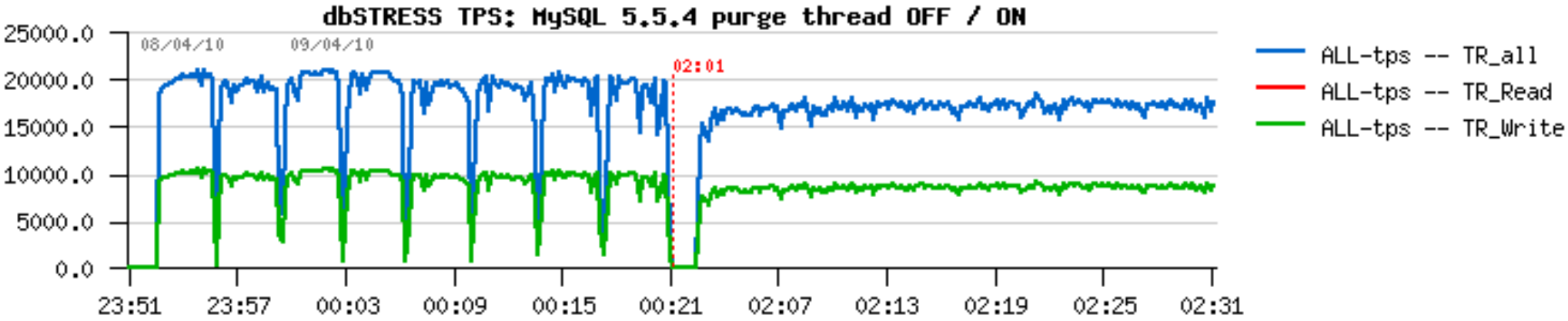


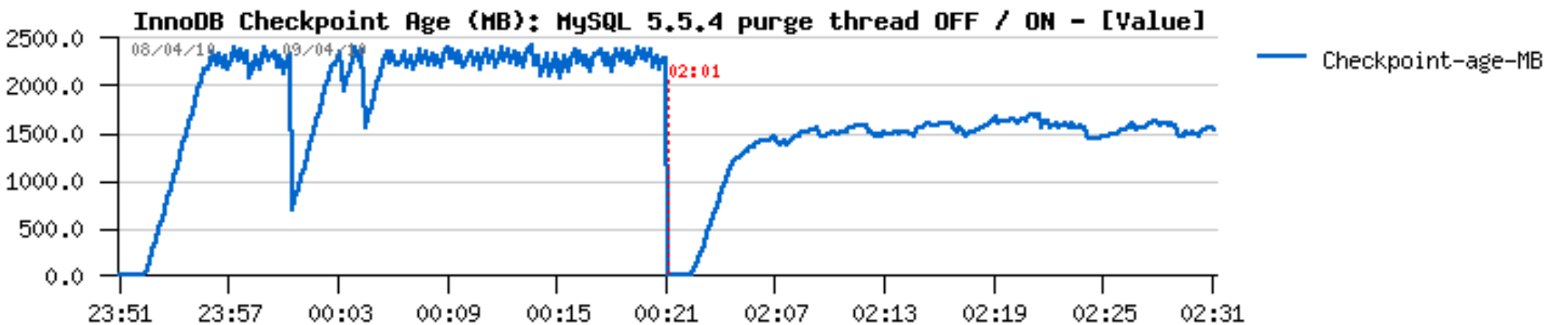
- MySQL 5.4 + purge fix :



Performance with Purge Thread

- MySQL 5.5 :



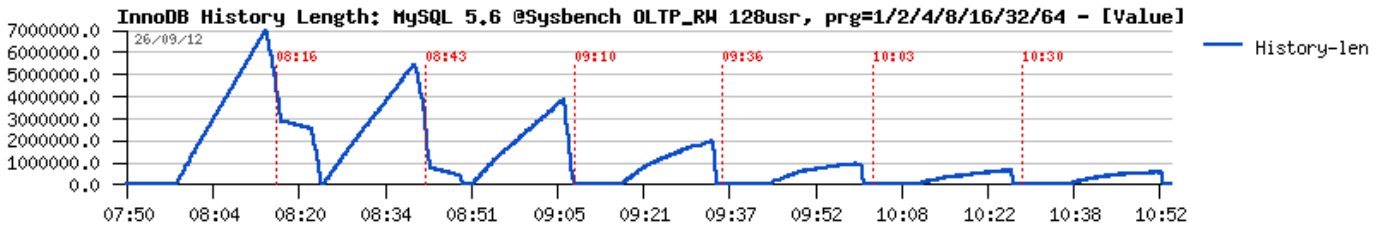


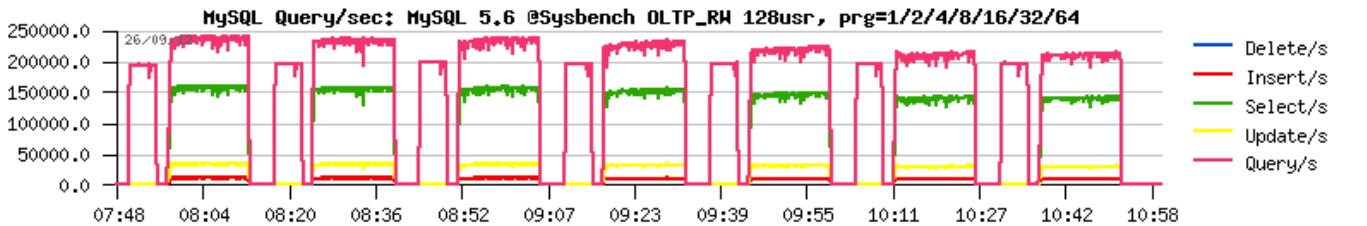
InnoDB Purge since MySQL 5.5

- Purging has a cost! (similar to Garbage Collecting)
  - Since MySQL 5.5: single purge thread (off by default)
  - Since MySQL 5.6: several purge thread(s) (up to 32)
- However, Purge may lag and do not follow workload..
  - This is very bad when happens...
  - Ex.: On aggressive RW got 400GB of undo records within few hours(!)
  - Then it took days to reach zero in History Length..
- The main problem is the past – how to dose purging?..
  - Since 5.6: with many threads, Purge become auto-stable itself
  - Still missing a dynamic config option to say how many purge threads to run in parallel right now (but it'll be fixed soon ;-))

InnoDB : Purge improvement in 5.6

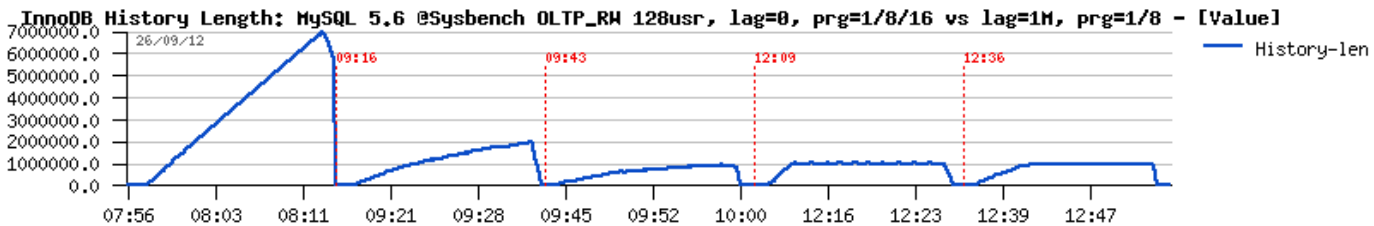
- Several Purge Threads :
  - NOTE: activation is auto-magical (I'm serious ;-))

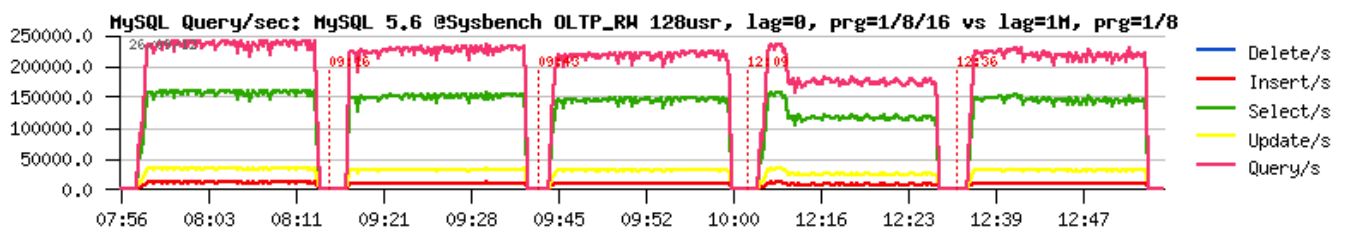




InnoDB : Purge improvement in 5.6

- Fixed max purge lag code!
  - innodb_max_purge_lag
  - innodb_max_purge_lag_delay <= configurable!
- Setting innodb_max_purge_lag=1M:

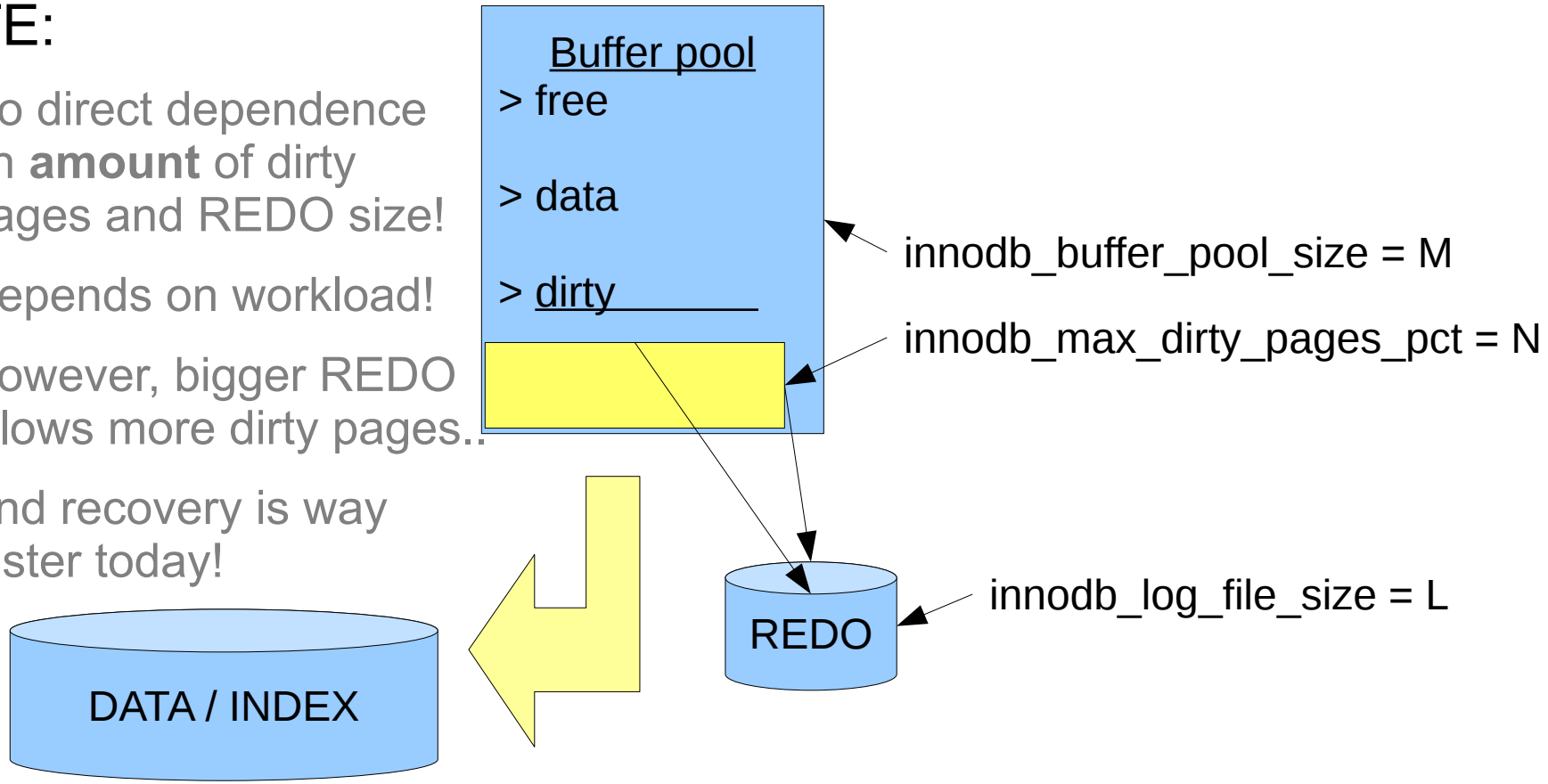




ORACLE

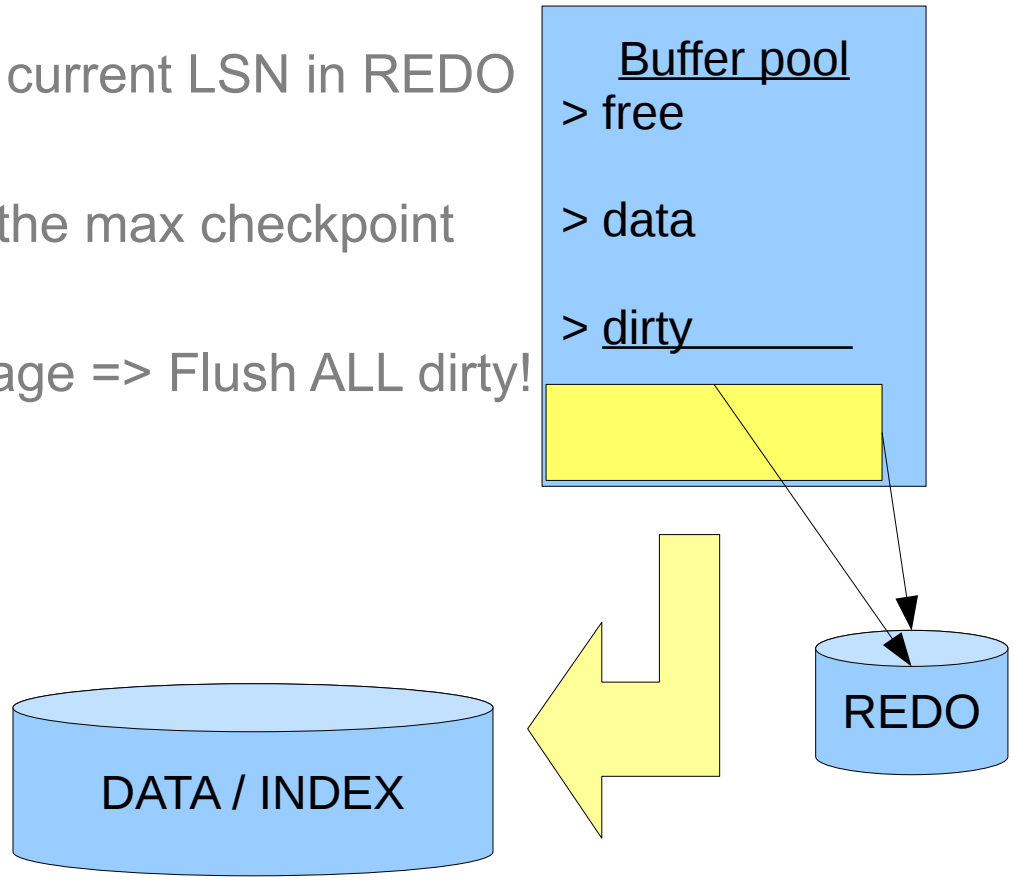
InnoDB Internals: “Furious Flushing”

- Direct dependence on REDO log size
- NOTE:
  - No direct dependence on **amount** of dirty pages and REDO size!
  - Depends on workload!
  - However, bigger REDO allows more dirty pages..
  - And recovery is way faster today!



InnoDB: REDO log constraints

- REDO log constraints: (Always monitor Checkpoint Age!!!)
  - Cyclic, need free space
  - Checkpoint age: diff between the current LSN in REDO and the oldest dirty page LSN
  - Checkpoint age cannot out-pass the max checkpoint age (redo log size)
  - If Checkpoint age >= 7/8 of Max age => Flush ALL dirty!
  - => AKA “furious flushing”...
- Adaptive Flushing:
  - Keep REDO under Max age
  - Respecting IO capacity limit

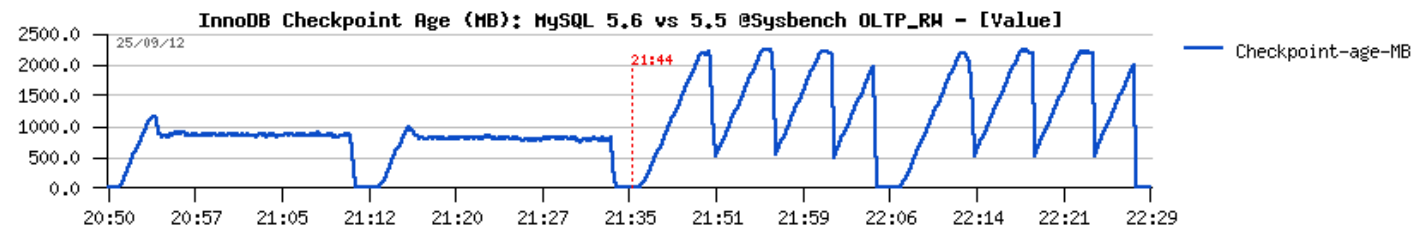


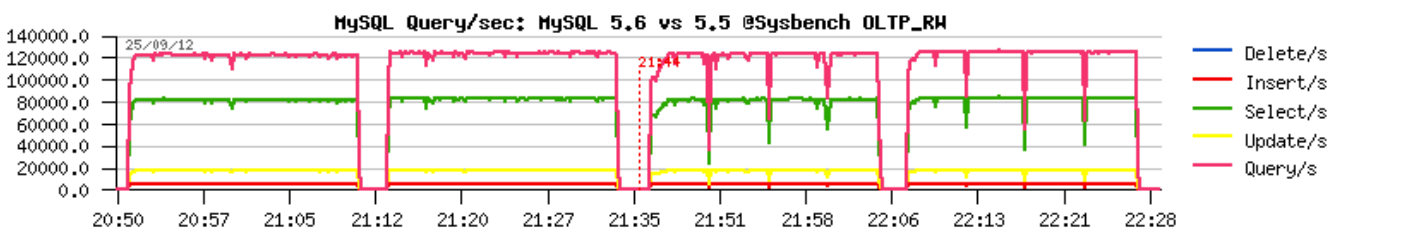
InnoDB: Adaptive Flushing

- MySQL 5.5:
  - Estimation based
  - Sometimes works ;-)
- MySQL 5.6 :
  - Based on REDO write rate + I/O capacity Max
  - Involving batch flushing with N pages to flush (progressive, depending on REDO %free) + page age limit (according REDO rate)
- Tuning:
  - innodb_io_capacity / innofb_io_capacity_max
  - innodb_adaptive_flushing_lwm / innodb_max_dirty_pages_pct_lwm
  - ALL are dynamic!
  - Monitor Checkpoint Age..

Adaptive Flushing: MySQL 5.6 vs 5.5

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

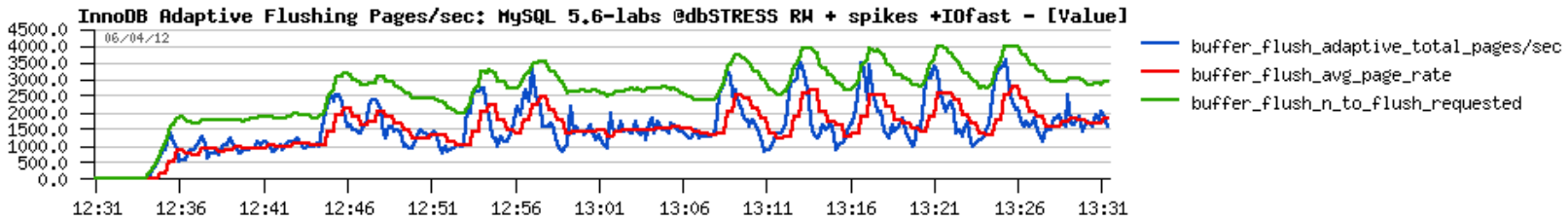


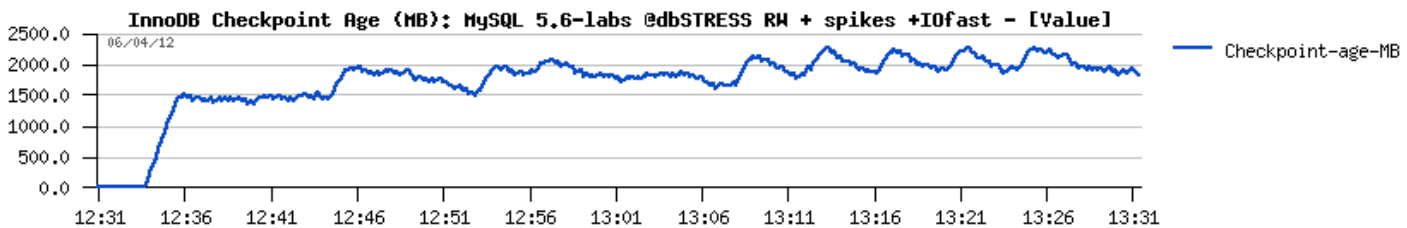


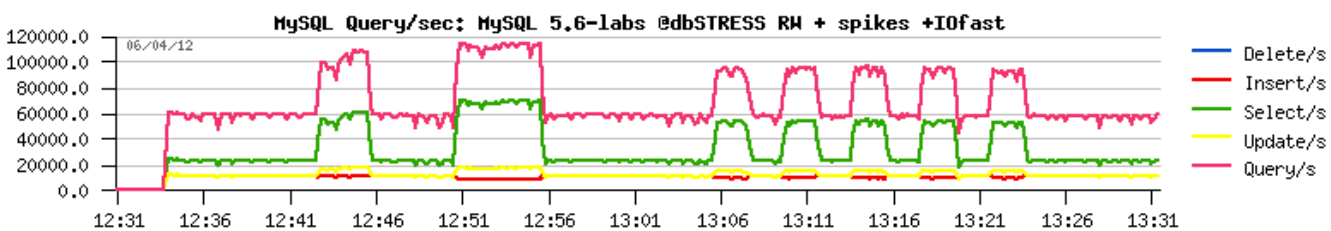
ORACLE

InnoDB : Resisting to activity spikes in 5.6

- dbSTRESS R+W with spikes



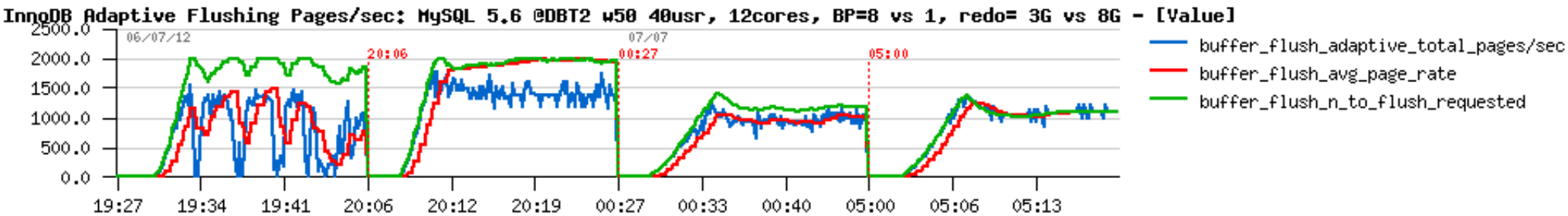


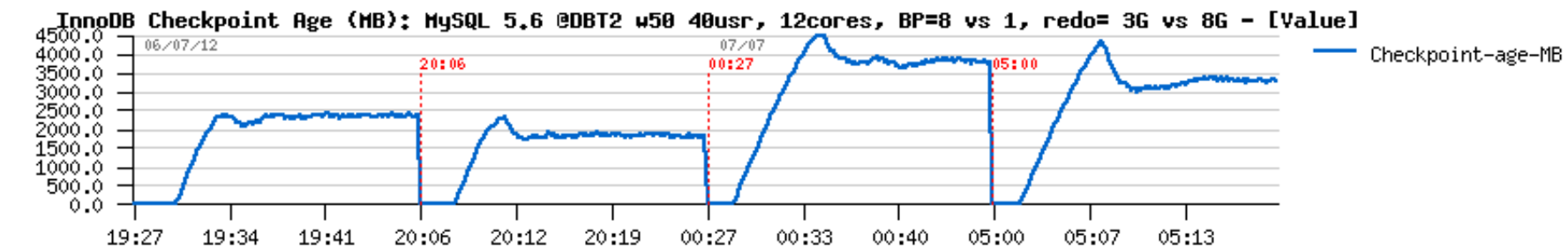


ORACLE

InnoDB Adaptive Flushing: Fine Tuning

- Monitor your Flushing rate / capabilities..
  - Adapt IO capacity and REDO size :

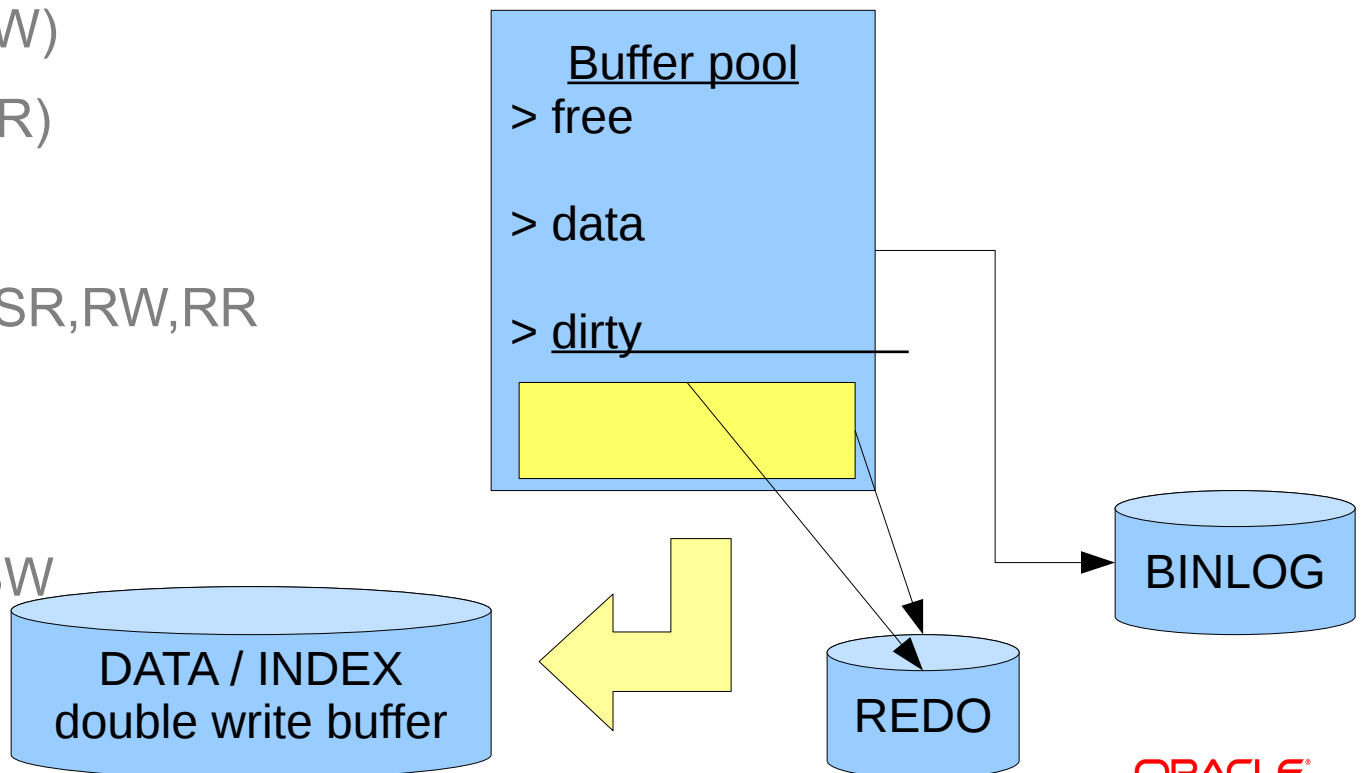




ORACLE

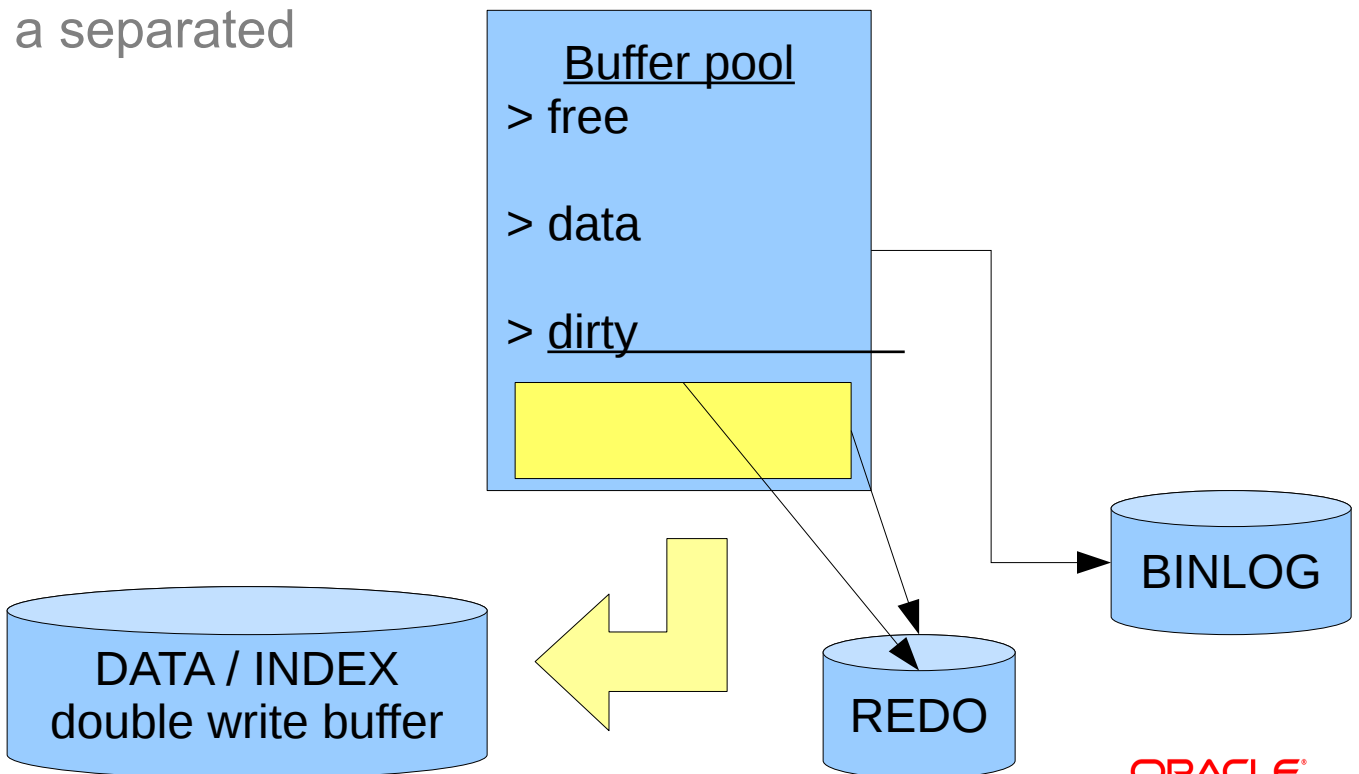
InnoDB and I/O Performance

- Keep in mind the nature of I/O operation!
  - Sequential Write (SW)
  - Sequential Read (SR)
  - Random Write (RW)
  - Random Read (RR)
- InnoDB
  - Data files <= SW,SR,RW,RR
  - Redo log <= SW
  - Bin log <= SW
  - Double write <= SW



InnoDB and I/O Performance

- Avoid a hot-mix of I/O operations!
  - Random Read (RR) <= most painful & costly!!!
  - Place REDO on different LUNs/disks
  - Place BINLOG on a separated storage array!
- I/O Settings
  - I/O capacity
  - I/O write threads
  - I/O read threads



InnoDB: Doublewrite Buffer

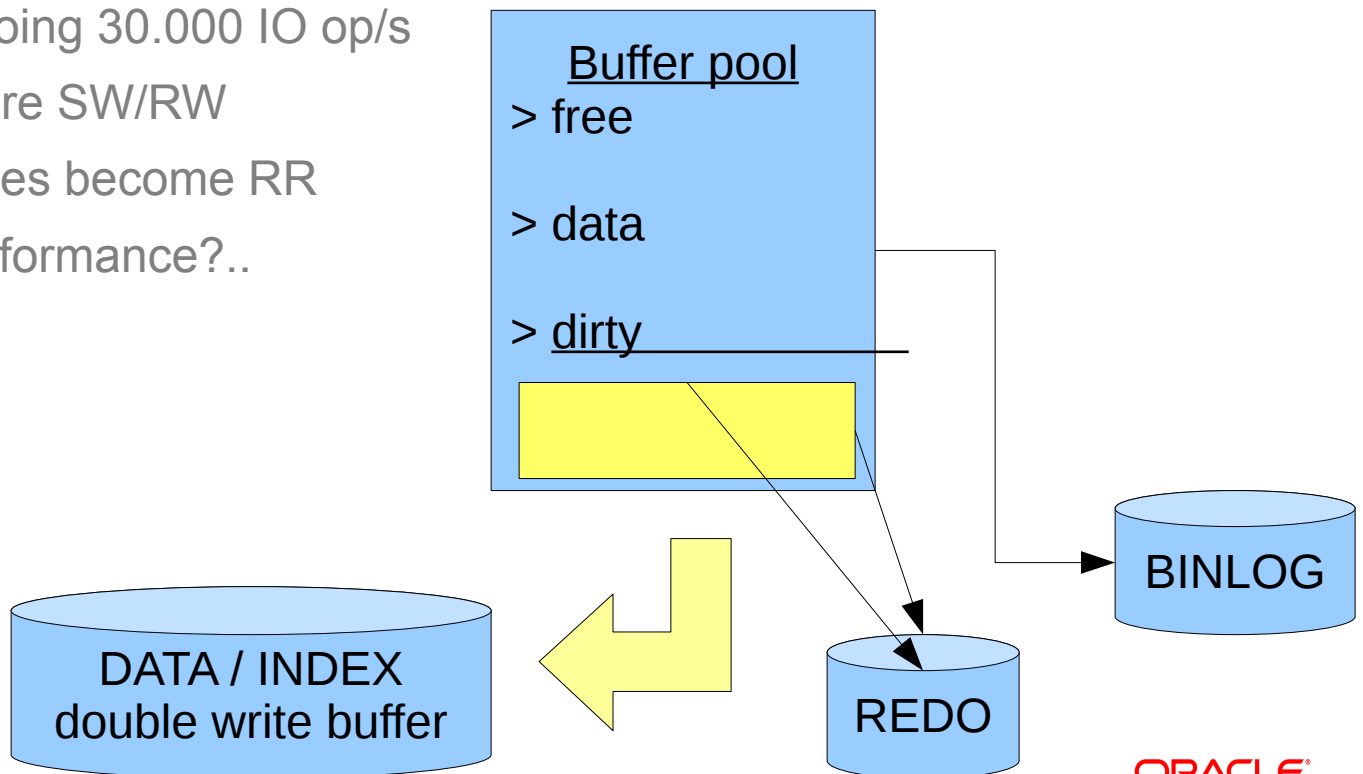
- Protecting from partially written pages
  - Data first written into Doublewrite buffer (sys.tablespace)
  - Then flushed to the datafiles
  - On recovery: if partially written page discovered => use its image from doublewrite buffer
- What is the cost?..
  - Doublewrite I/O is sequential, so should be fast
  - Writes will do less sync calls:
    - Instead of sync on every page write
    - Sync once on doublewrite buffer write
    - Then once on the datafile(s) for the same chunk of pages

InnoDB: Doublewrite buffer real impact?

- Usually:
  - performance remains the same (or better)
  - + recovery guarantee!
- In some cases:
  - Up to 30% performance degradation...
  - Why?...

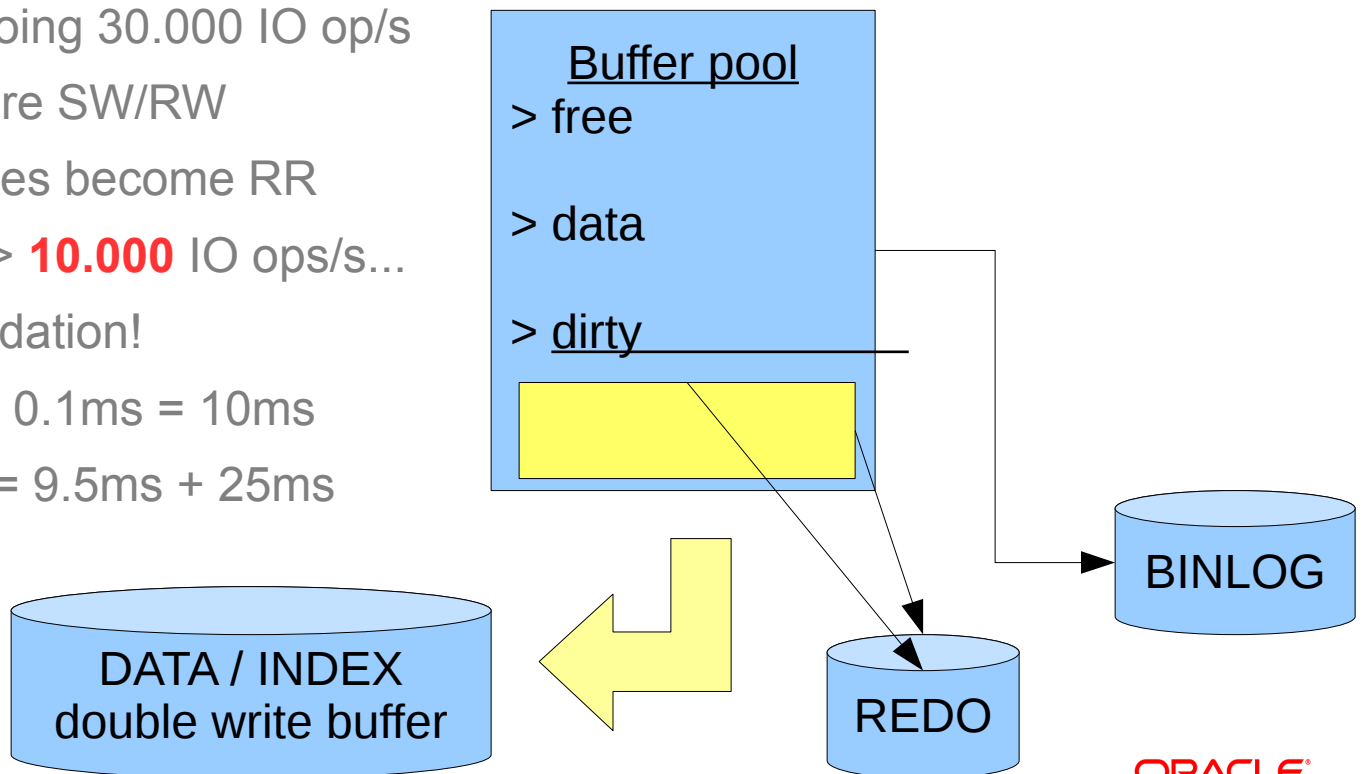
InnoDB: Doublebuffer and I/O dependency

- Random Reads are killing!
  - RR = ~5ms wait per operation on HD
  - Example:
    - Application is doing 30.000 IO op/s
    - All operations are SW/RW
    - Now 5% of Writes become RR
    - What about performance?..



InnoDB: Doublebuffer and I/O dependency

- Random Reads are killing!
  - RR = 5ms wait per operation
  - Example:
    - Application is doing 30.000 IO op/s
    - All operations are SW/RW
    - Now 5% of Writes become RR
    - Performance => **10.000** IO ops/s...
    - **x3 times** degradation!
    - 100 SW= 100 x 0.1ms = 10ms
    - 95 SW + 5 RR = 9.5ms + 25ms

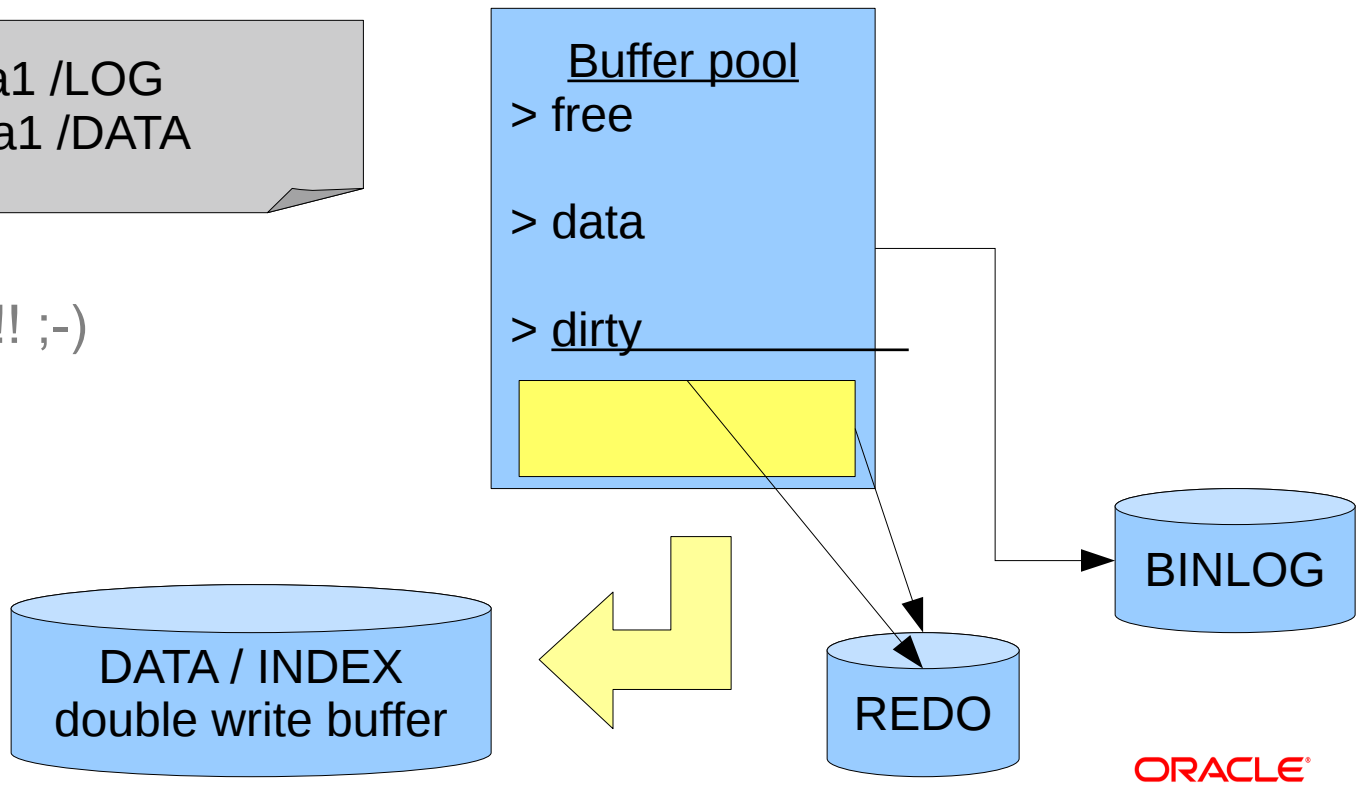


InnoDB: Doublebuffer and I/O dependency

- Workaround: move doublewrite buffer on REDO disks
  - Have to set innodb_file_per_table initially for DB
  - Move system tablespace on REDO disks:

```
$ mv /DATA/ibdata1 /LOG
$ ln -s /LOG/ibdata1 /DATA
```

  - Or just use SSD !!! ;-)



User Concurrency scenarios

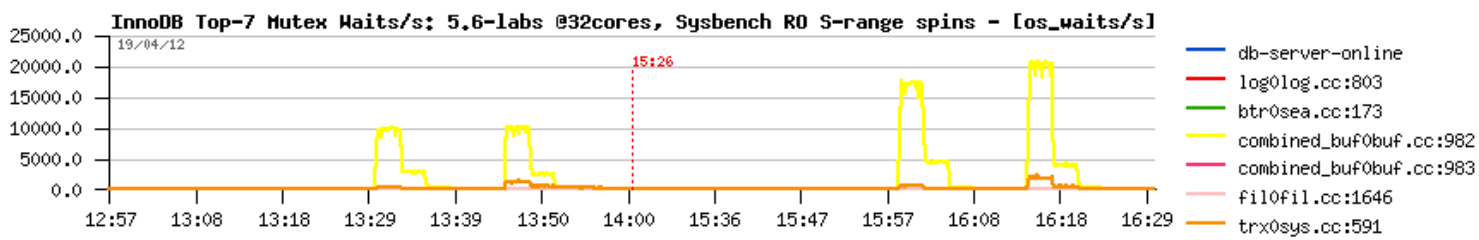
- Single user?..
  - With a bigger code path today 5.6 simply cannot be faster than 5.5
  - But then, why you're not considering Query Cache? ;-)
- More users?..
  - Up to 8-16 concurrent users all internal contention are not yet hot
  - So, 5.6 will not be better yet..
- More than 16 users?..
  - Then you'll feel a real difference, but if you have at least 16cores ;-)
  - Or if you have really a lot of concurrent users
- But don't forget other 5.6 improvements either!
  - On-line DDL, Binlog group commit, Memcached, etc..

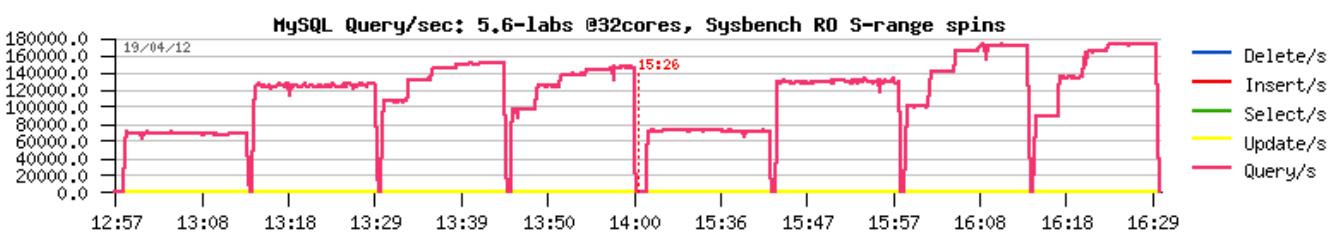
High Concurrency Tuning

- If bottleneck is due a concurrent access on the same data (due application design) – ask dev team to re-design ;-)
- If bottleneck is due MySQL/InnoDB internal contentions, then:
  - If you cannot avoid it, then at least don't let them grow ;-)
  - Try to increase InnoDB spin wait delay (dynamic)
  - Try innodb_thread_concurrency=N (dynamic)
  - CPU taskset / prcset (Linux / Solaris, both dynamic)
  - Thread Pool
  - NOTE: things with contentions may radically change since 5.7, so stay tuned ;-)

InnoDB Spin Wait Delay

- RO/RW Workloads:
  - With more CPU cores internal contentions become more hot..
  - Bind mysqld to less cores helps, but the goal is to use more cores ;-)
  - Using innodb_thread_concurrency is not helping here anymore..
  - So, **innodb_spin_wait_delay** is entering in the game:

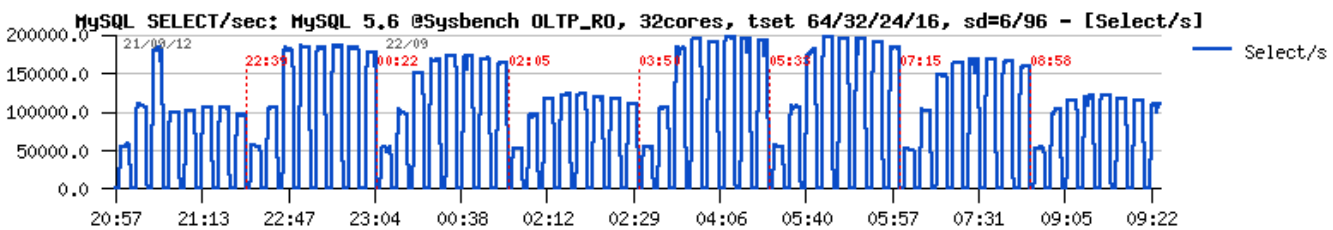


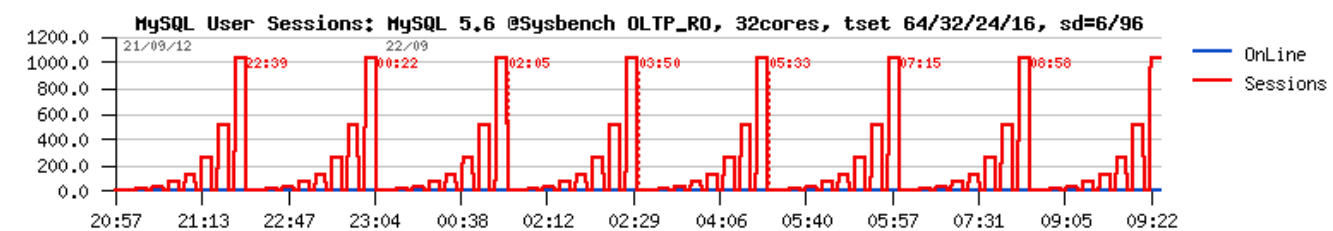


ORACLE

Tune InnoDB Spin Wait Delay

- Notes :
  - is the max random delay on “sleep” within a spin loop in wait for lock..
  - Ideally should be auto.. while the same tuning works for 5.5 as well ;-)
  - General rule: default is 6, may need an increase with more cores
  - Test: 32-HT/ 32/ 24/ 16cores, spin delay = 6 / 96 :





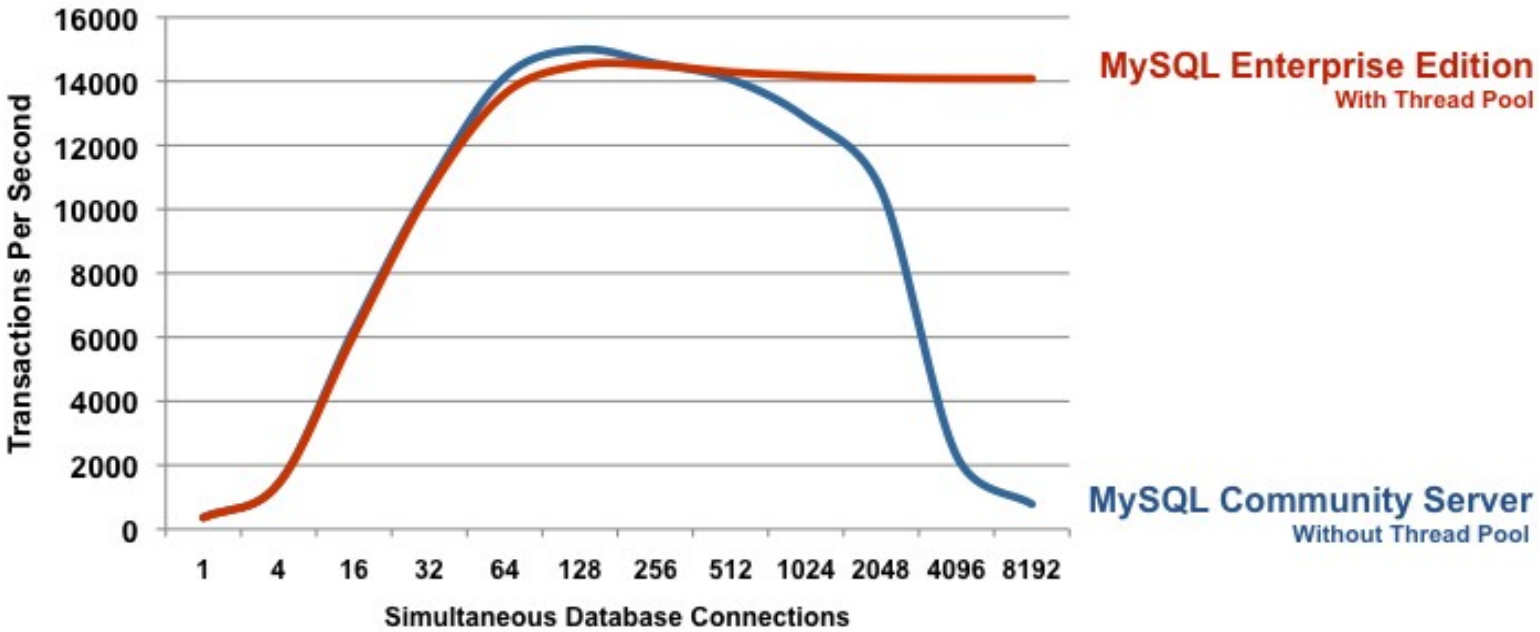
ORACLE

Thread Pool @MySQL

- None of these solutions will help to increase performance!
  - it'll just help to keep the peak level constant (and you yet need to discover on which level of concurrency you're reaching your peak ;-))
- ThreadPool in MySQL 5.5 and 5.6 is aware if I/O are involved!
  - So, better than innodb thread concurrency setting or taskset
  - May still require spin wait delay tuning!
  - The must for high concurrency loads!
  - May still start to show a difference since 32-128 concurrent users! (all depends on workload)..
  - Keep in mind that OS scheduler is not aware how to manage user threads most optimally, but ThreadPool does ;-)

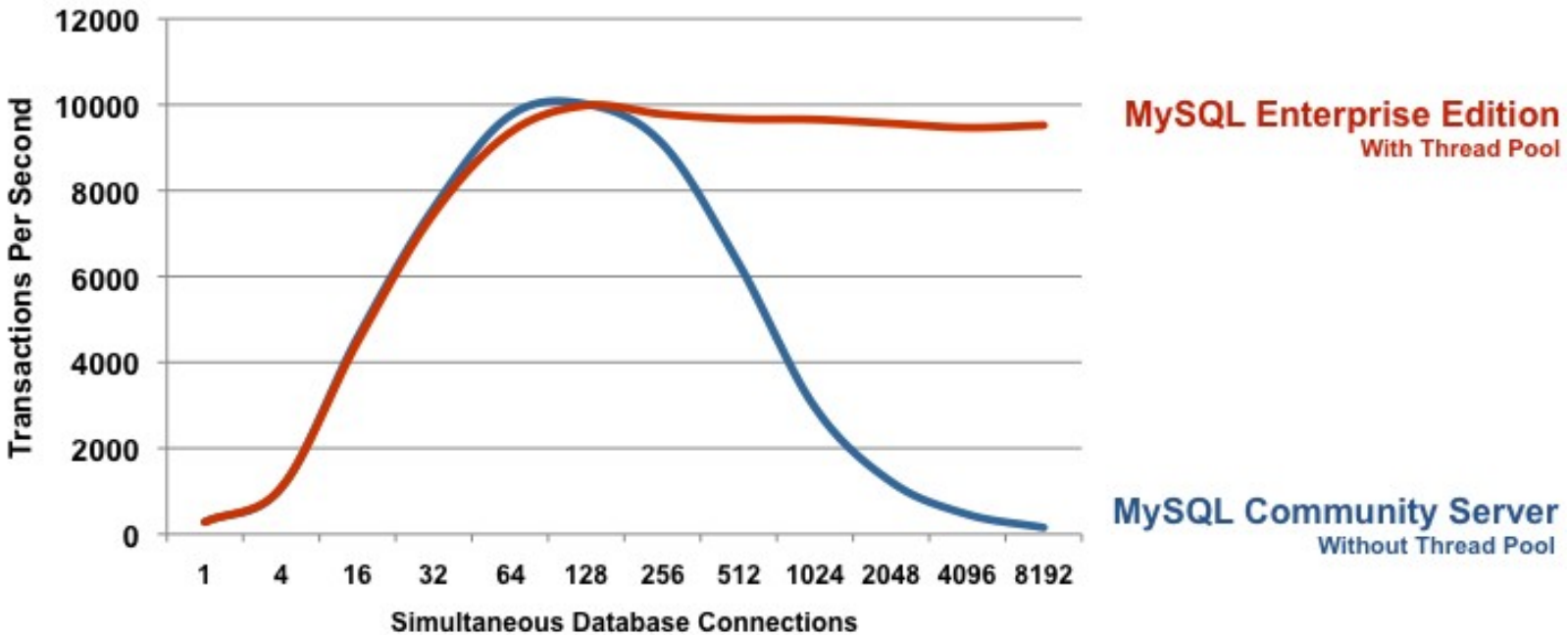
Thread Pool in MySQL 5.6

- OLTP_RO:

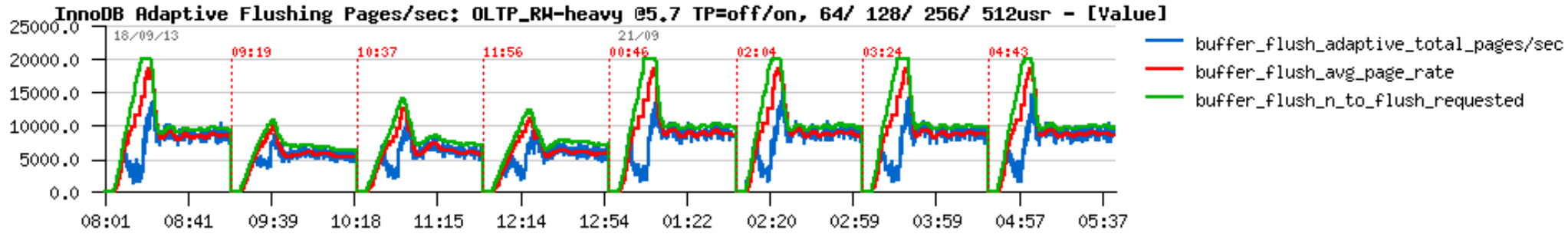


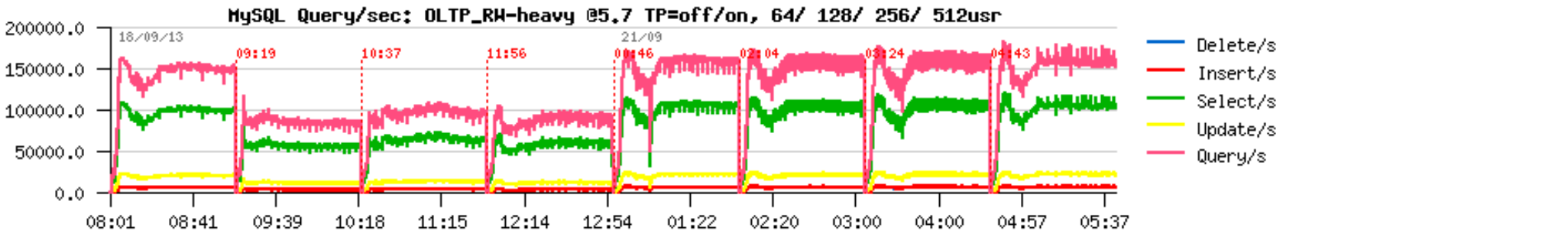
Thread Pool in MySQL 5.6

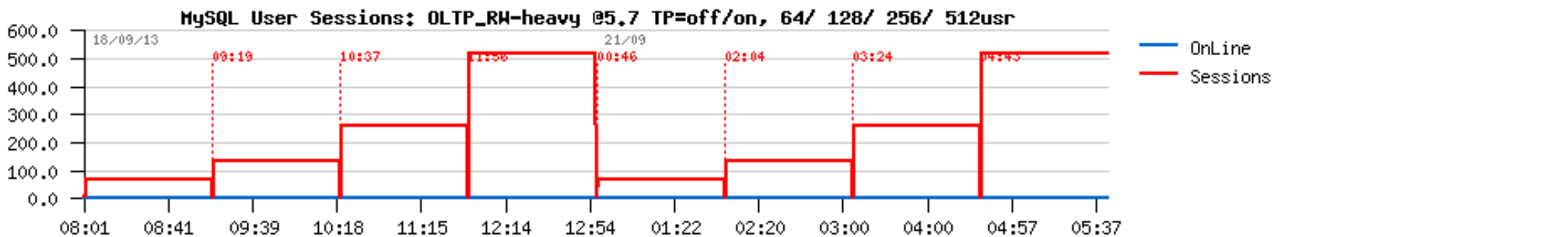
- OLTP_RW:



Thread Pool in MySQL 5.7 @Heavy OLTP_RW







ORACLE

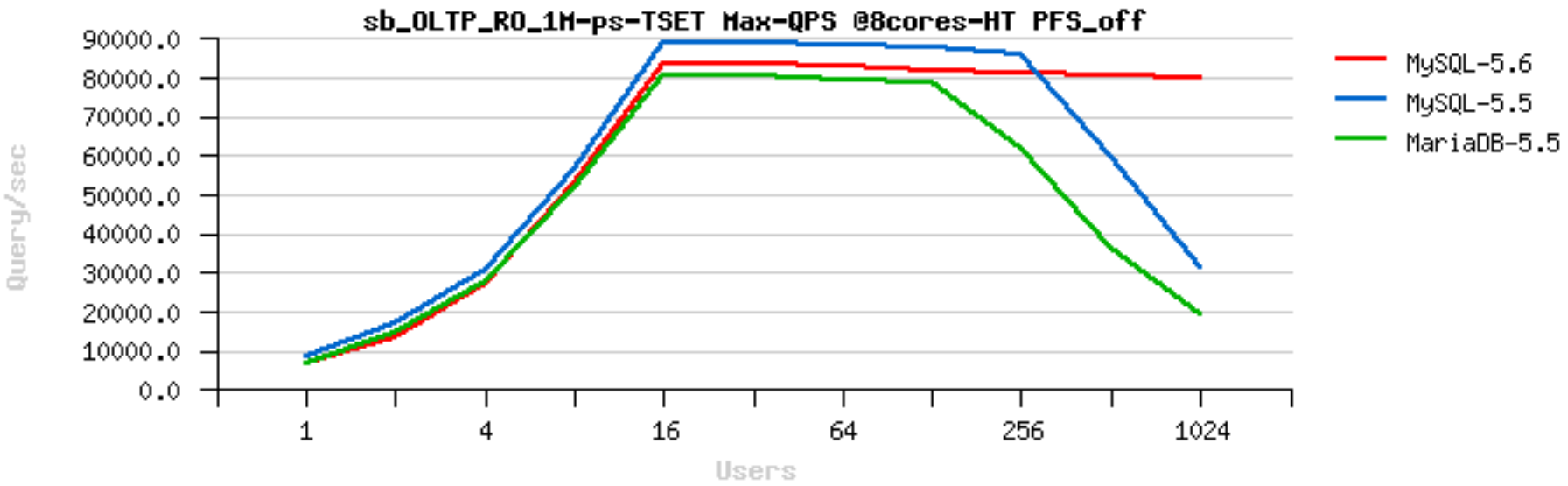
InnoDB High Concurrency: AHI

- Adaptive Hash Index (AHI)
  - Helps a lot on Read-Only workloads
  - In fact it helps always until itself become not actively modified
  - AHI contention is seen as its btr_search_latch RW-lock contetnion
  - So, on Read+Write become a huge bottleneck..
  - In many cases on RW the result is better with AHI=off..
  - NOTE: there is still a big mystery around AHI when it's having btr_search_latch contention even when there is no changes at all (pure RO in memory).. - expected to be fixed in 5.7 ;-)

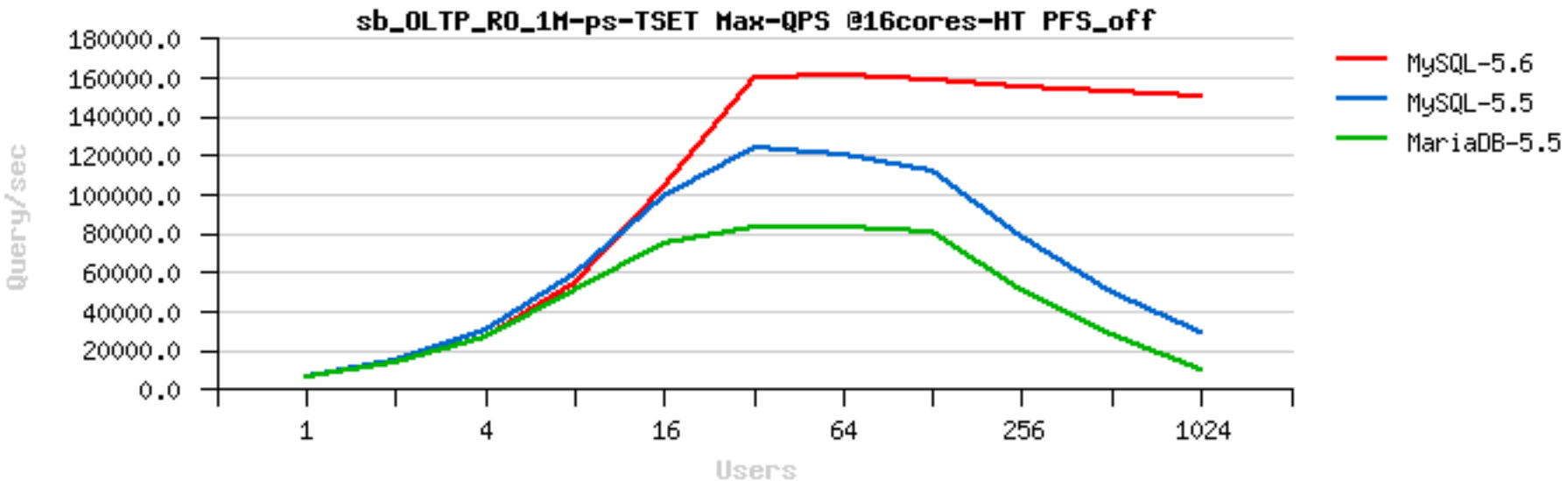
Testing Apples-to-Apples...

- Comparing MySQL 5.6 vs 5.5 :
  - Don't have G5: dead..
  - Don't have open table cache instances: bad..
  - Don't have improved Adaptive Flushing; bad..
  - Don't have fixed Purge & Lag: danger!..
  - Don't have binlog group commit and use binlog: dead..
  - Etc. etc. etc.
  - NOTE: some “improvement” are also fixes which are making stuff working properly, but coming with additional overhead (like Purge)..
  - NOTE: when comparing 5.6 and 5.5 keep in mind that Performance Schema is enabled by default in 5.6, and not in 5.5, so think to disable it in both (as 5.5 also has a way less PFS instrumentation)..

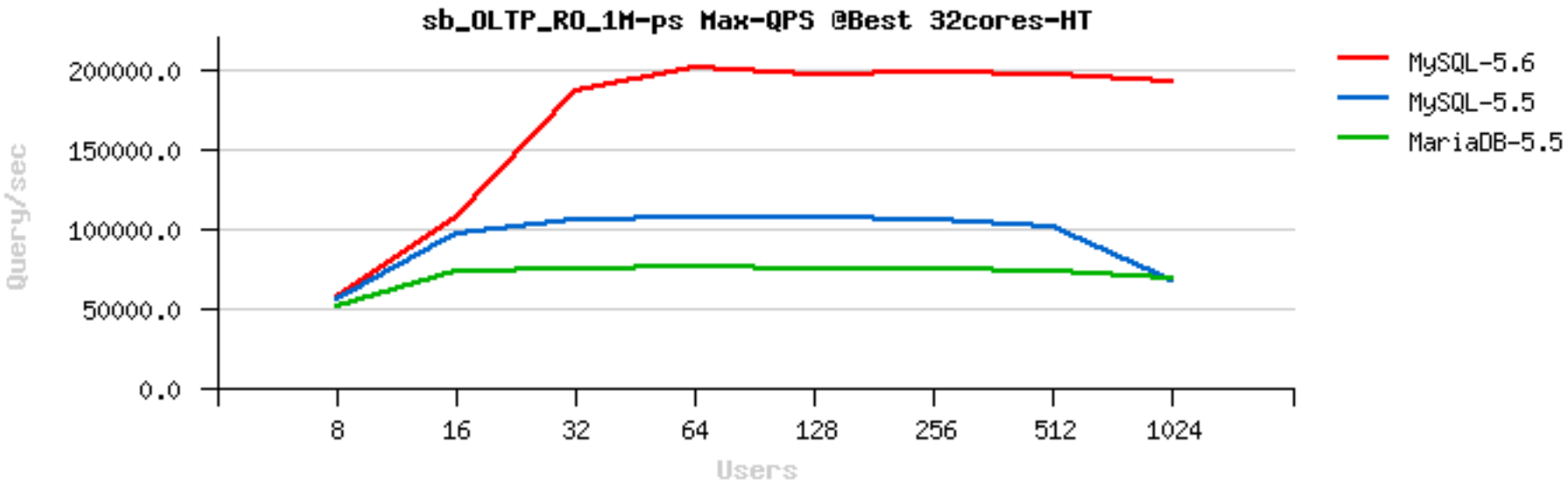
Sysbench OLTP_RO @8cores-HT (Apr.2013)



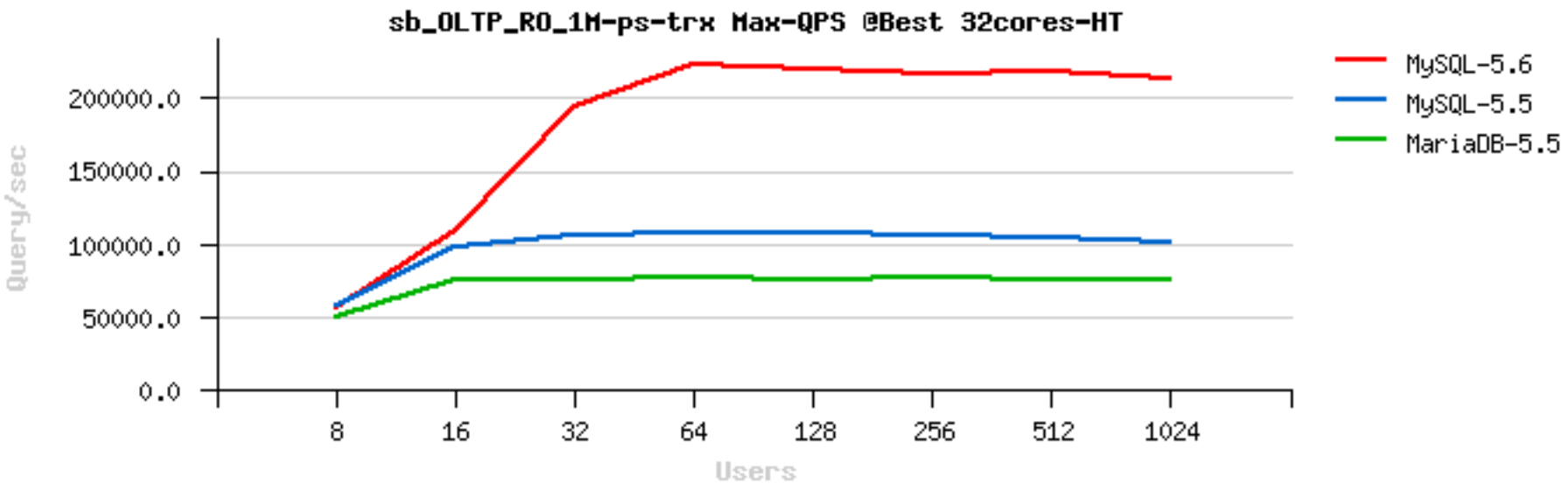
Sysbench OLTP_RO @16cores-HT (Apr.2013)



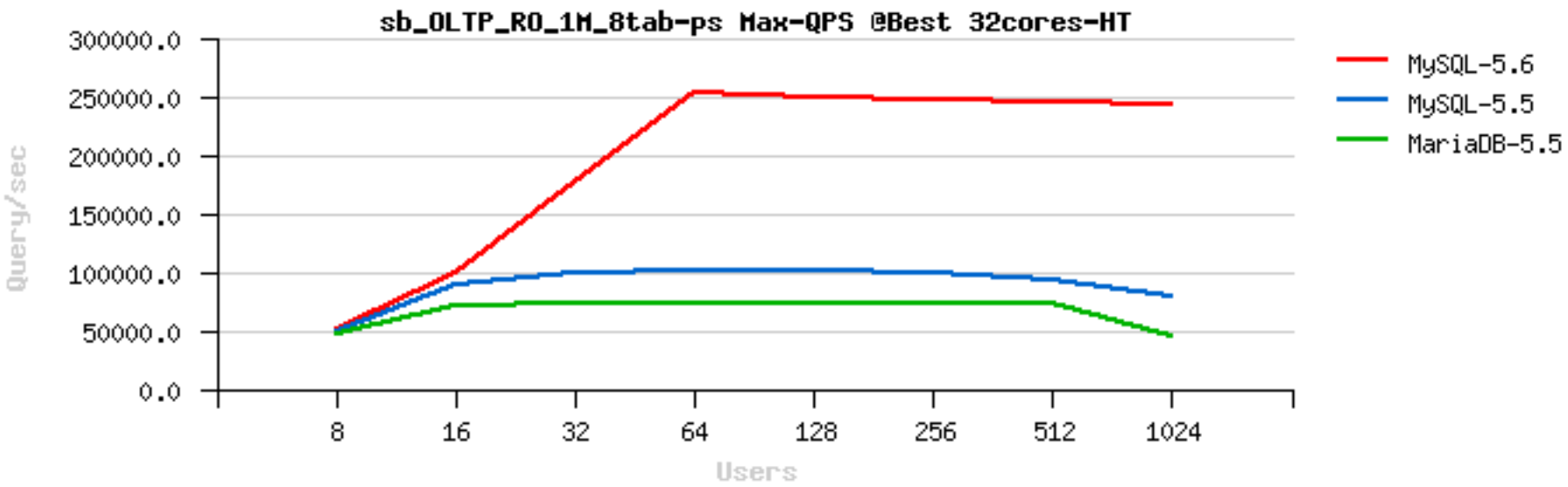
Sysbench OLTP_RO @32cores-HT (Apr.2013)



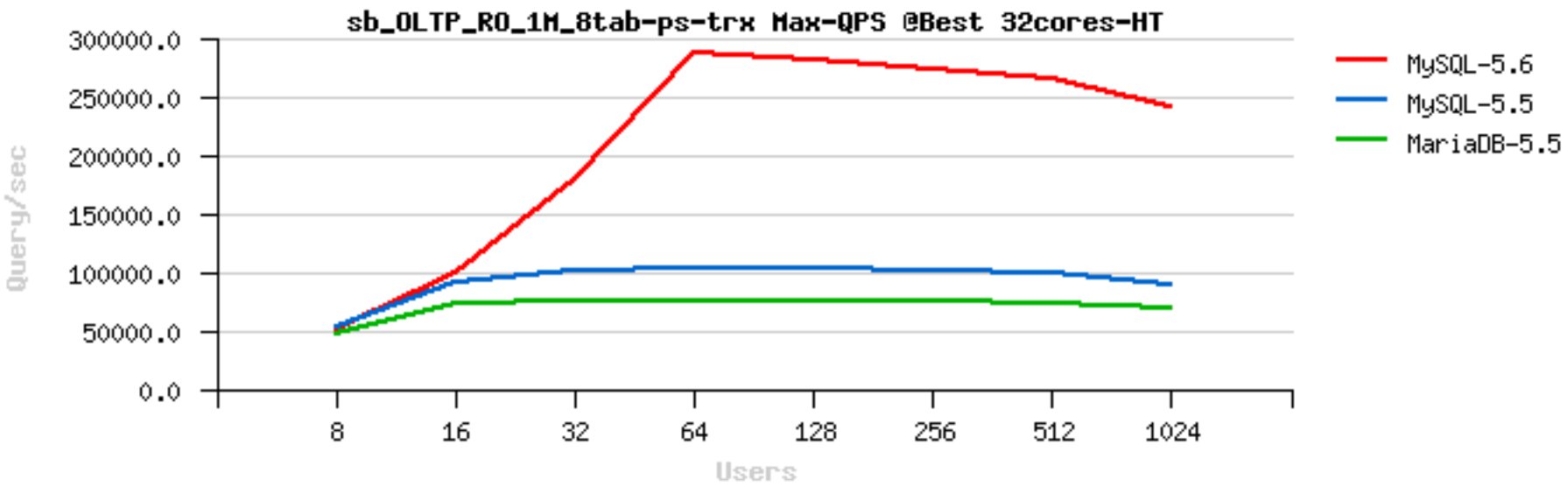
Sysbench OLTP_RO-trx @32cores-HT (Apr.2013)



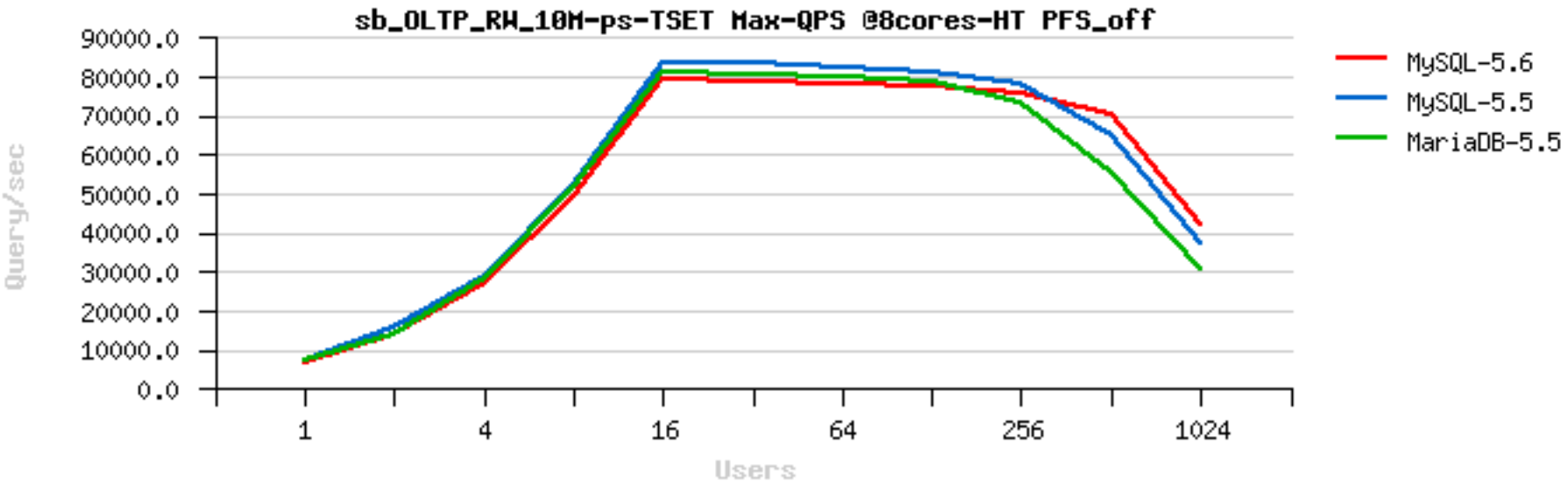
Sysbench OLTP_RO 8-tab @32cores-HT (Apr.2013)



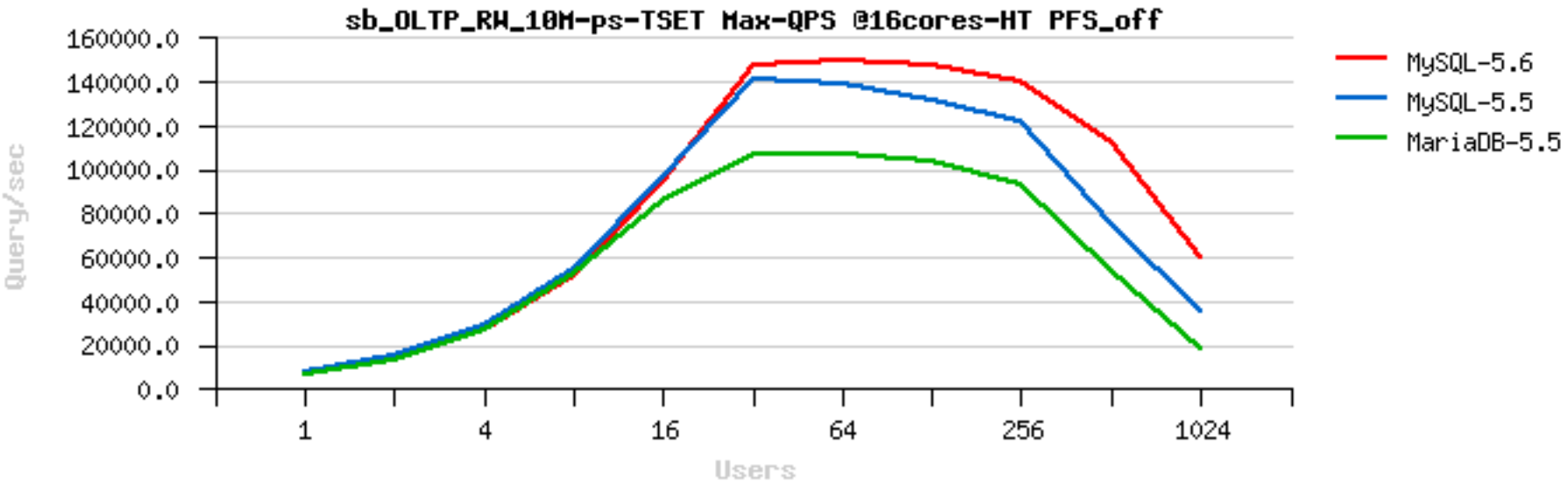
Sysbench OLTP_RO-trx 8-tab @32cores-HT (Apr.2013)



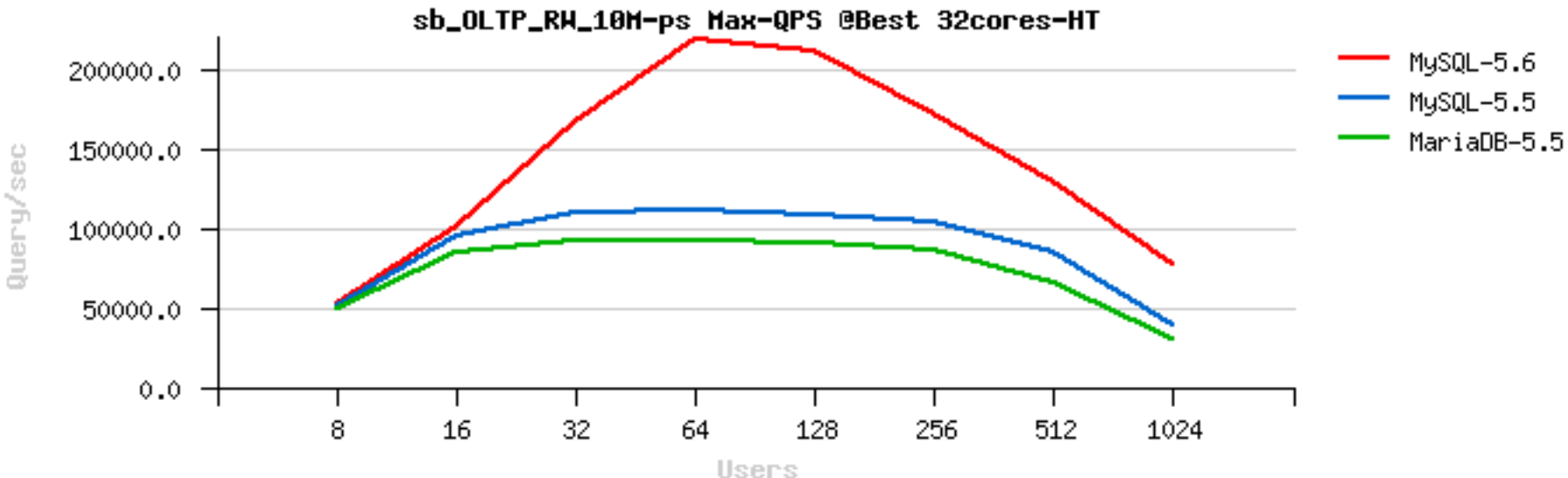
Sysbench OLTP_RW @8cores-HT (Apr.2013)



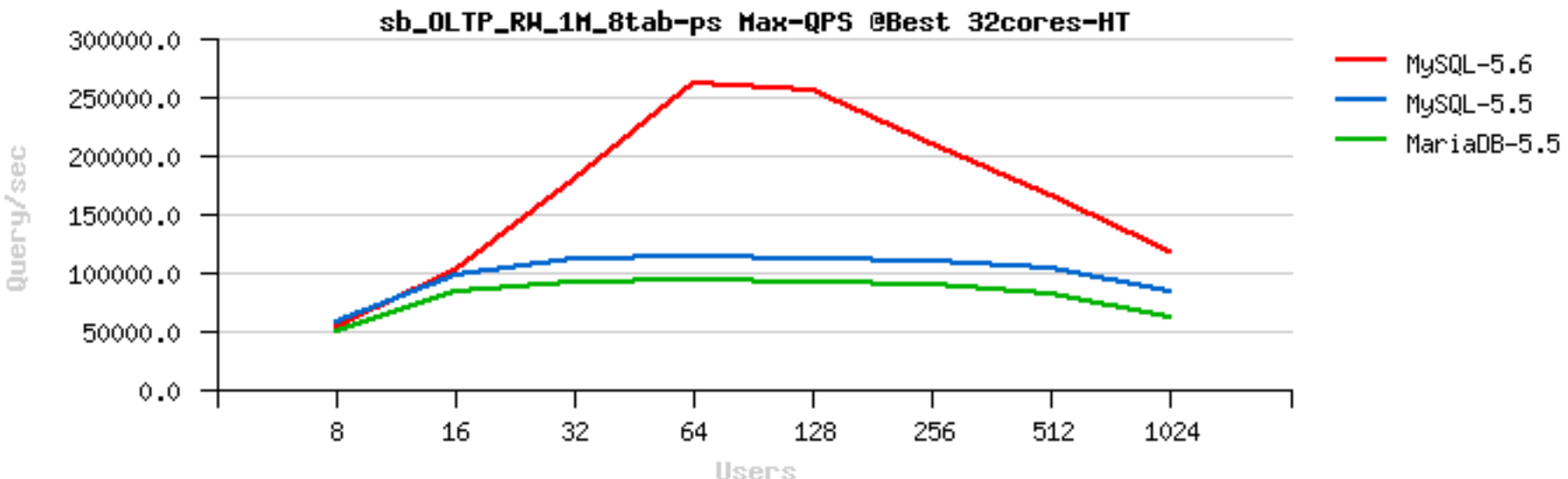
Sysbench OLTP_RW @16cores-HT (Apr.2013)



Sysbench OLTP_RW @32cores-HT (Apr.2013)



Sysbench OLTP_RW 8-tab @32cores-HT (Apr.2013)



MySQL 5.6: Pending issues

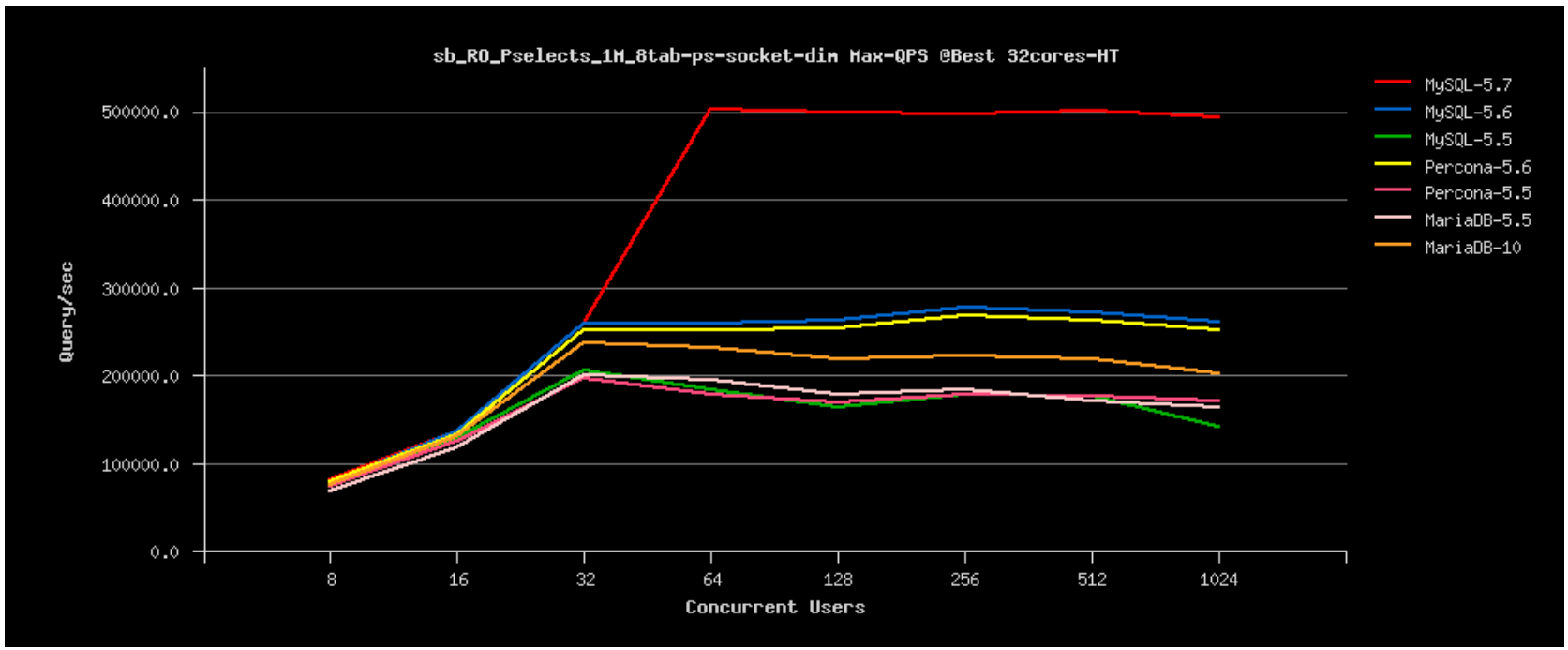
- Index lock..
- Lock_sys contention..
- Trx_sys contention..
- MDL scalability..
- Flushing limits..
- LRU flushing..
- Design bug on block locking.. (was here from the beginning)
- Not able yet to use 100% I/O capacity on a powerful storage..
- “Mysterious” contentions on dbSTRESS..
- etc..

MySQL 5.7: Work in progress.. ;-)

- Index lock.. <== fixed !
- Lock_sys contention.. <== lowered !
- Trx_sys contention.. <== improved a lot !!!
- MDL scalability.. <== in progress..
- Flushing limits.. <== in progress..
- LRU flushing.. <== in progress..
- Design bug on block locking.. (was here from the beginning)
- Not able yet to use 100% I/O capacity on a powerful storage..
- “Mysterious” contentions on dbSTRESS..
- Etc.. <== well, ALL in progress / investigation ;-)

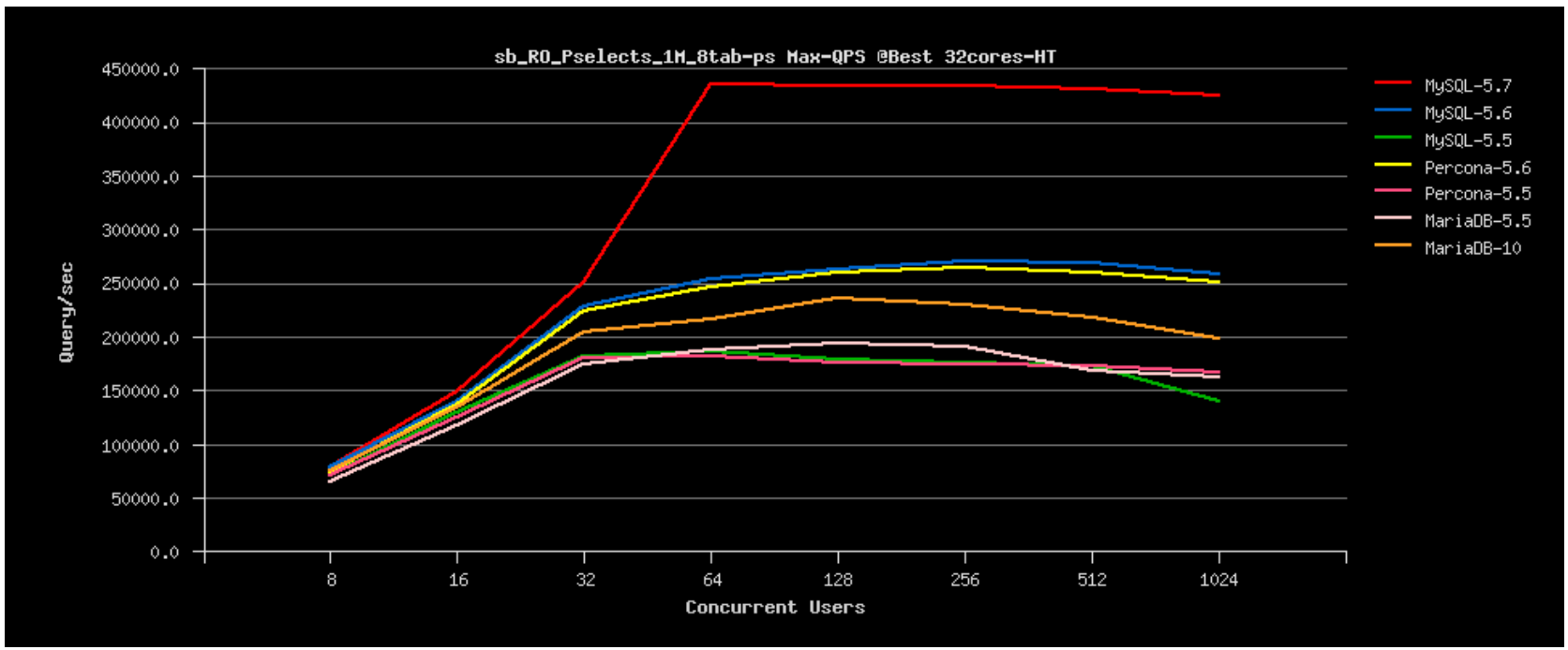
MySQL 5.7: DMR2 (Sep.2013)

- OLTP_RO Point-Selects 8-tables: **500K** QPS !!!
  - UNIX socket, sysbench 0.4.8 (older, using less CPU)



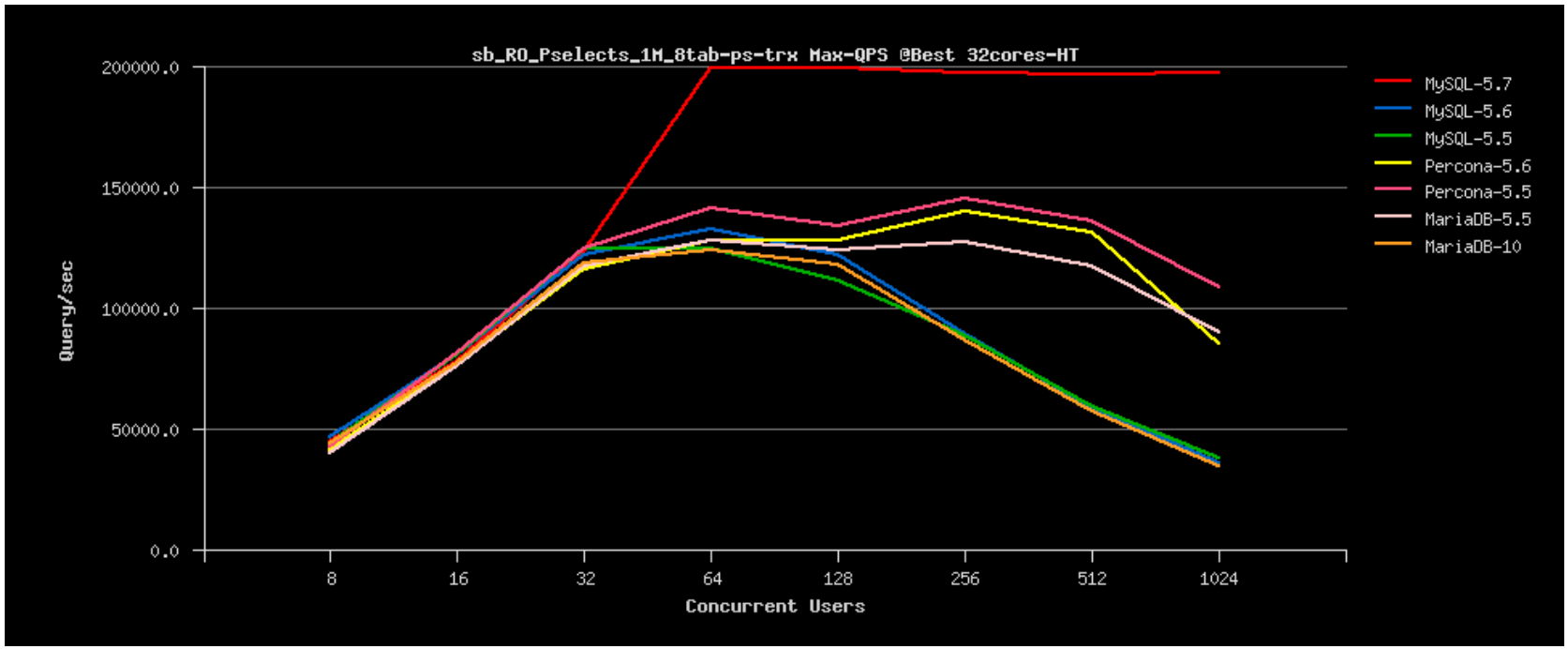
MySQL 5.7: DMR2 (Sep.2013)

- OLTP_RO Point-Selects 8-tables: **440K** QPS
  - IP port, sysbench 0.4.13 (“common”, using more CPU)



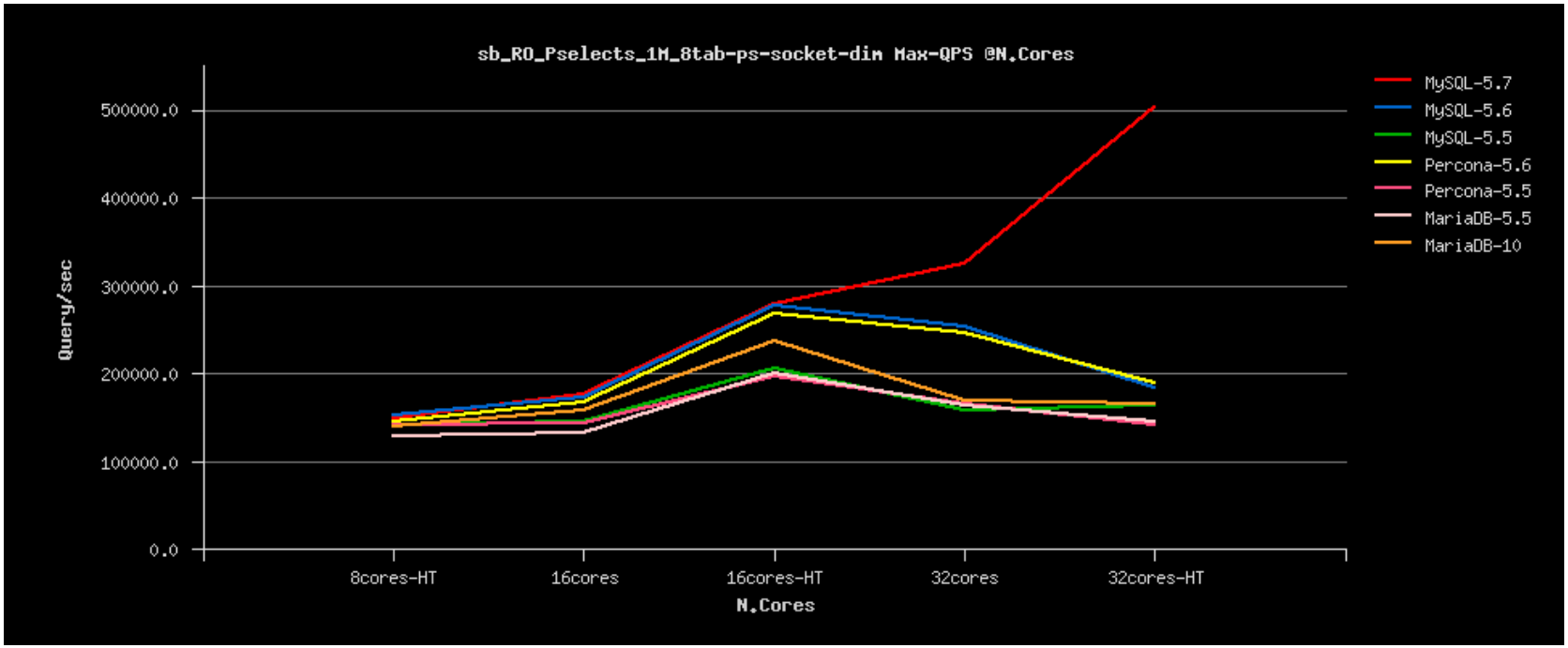
MySQL 5.7: DMR2 (Sep.2013)

- OLTP_RO Point-Selects-TRX 8-tables: **200K** QPS
  - IP port, sysbench 0.4.13 (“common”, using more CPU)



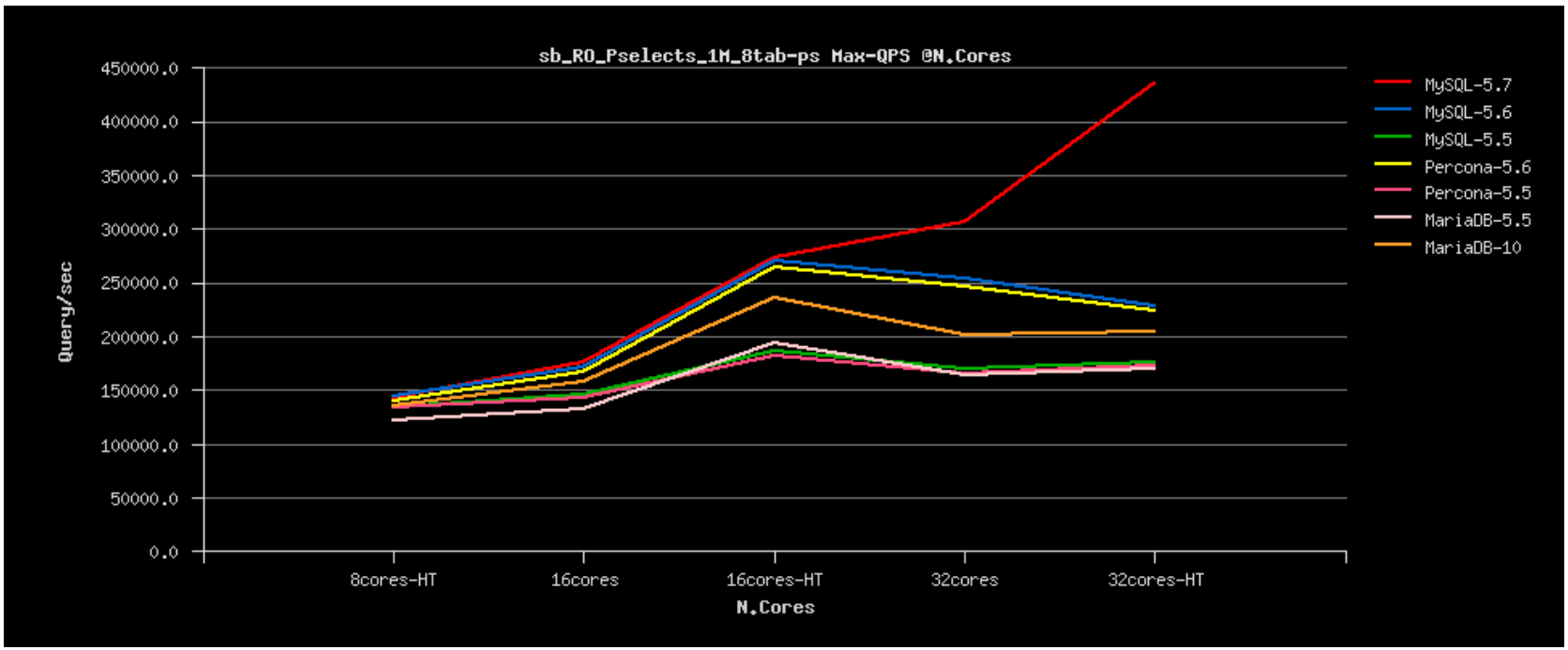
MySQL 5.7: DMR2 (Sep.2013)

- OLTP_RO Point-Selects 8-tables: Scalability..
  - UNIX socket, sysbench 0.4.8 (older, using less CPU)



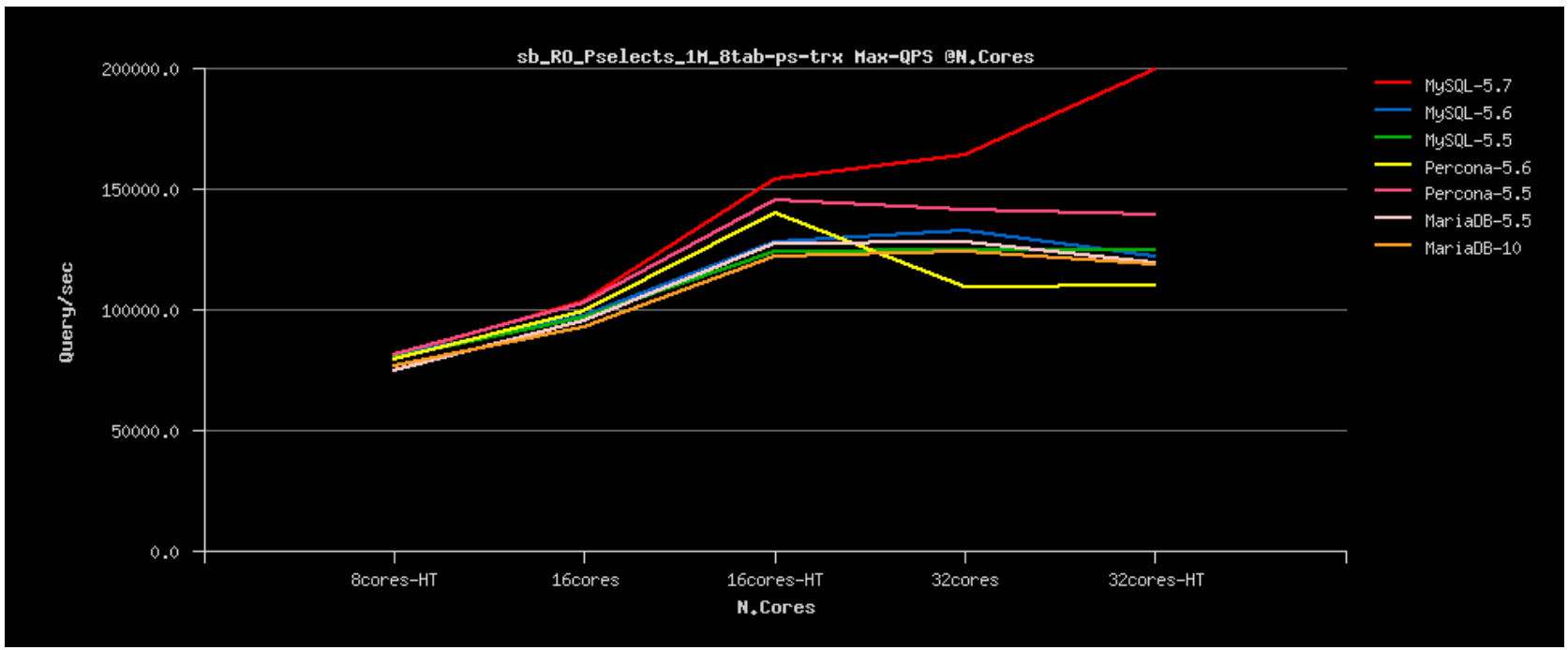
MySQL 5.7: DMR2 (Sep.2013)

- OLTP_RO Point-Selects 8-tables: Scalability..
  - IP port, sysbench 0.4.13 (using more CPU)



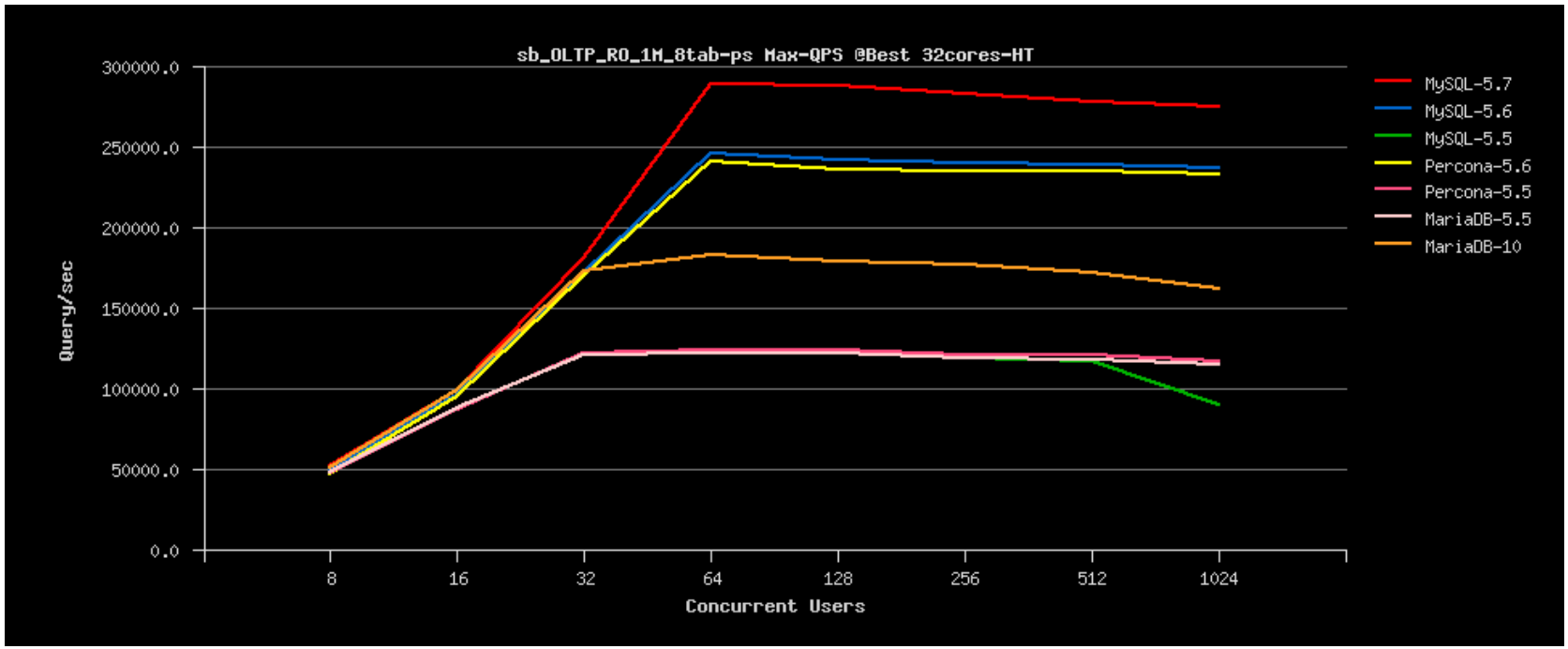
MySQL 5.7: DMR2 (Sep.2013)

- OLTP_RO Point-Selects-TRX 8-tables: Scalability..
  - IP port, sysbench 0.4.13 (using more CPU)



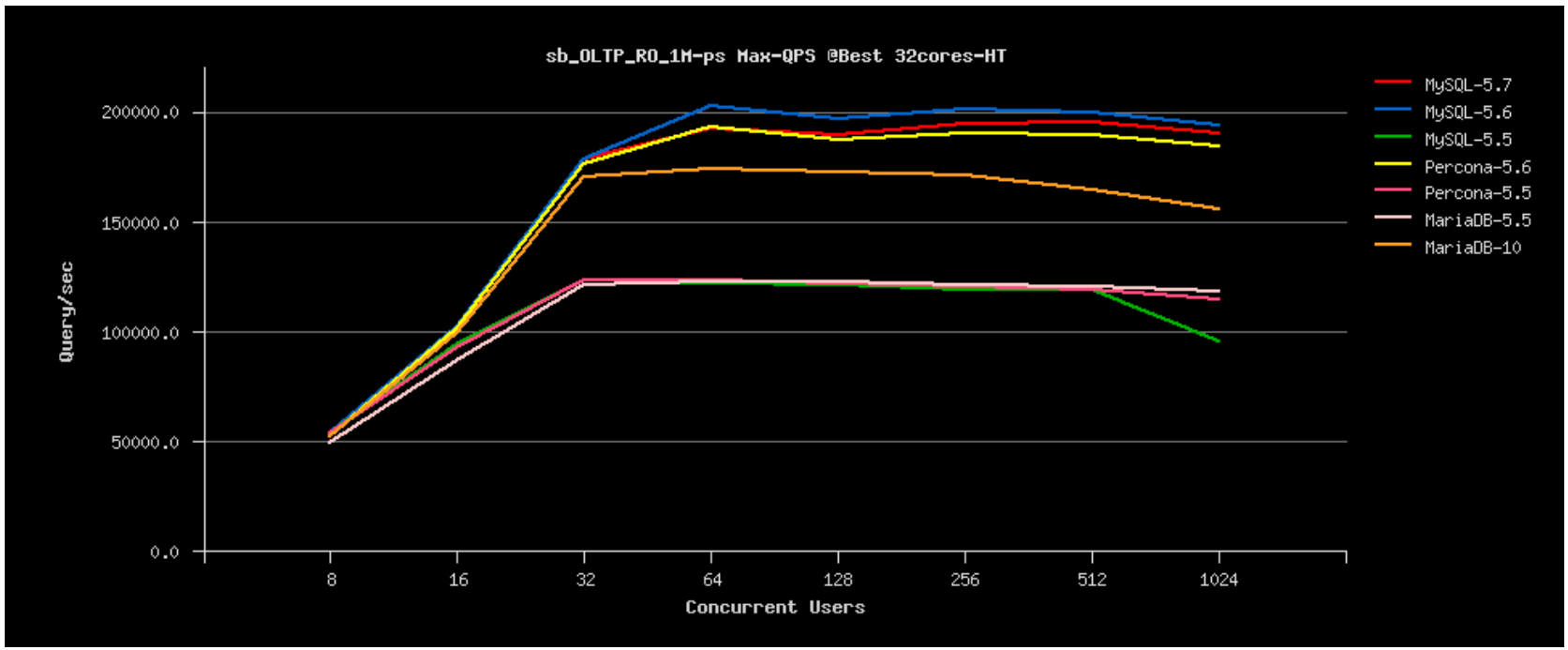
MySQL 5.7: DMR2 (Sep.2013)

- OLTP_RO 8-tables: **280K** QPS
  - IP port, sysbench 0.4.13 (“common”, using more CPU)



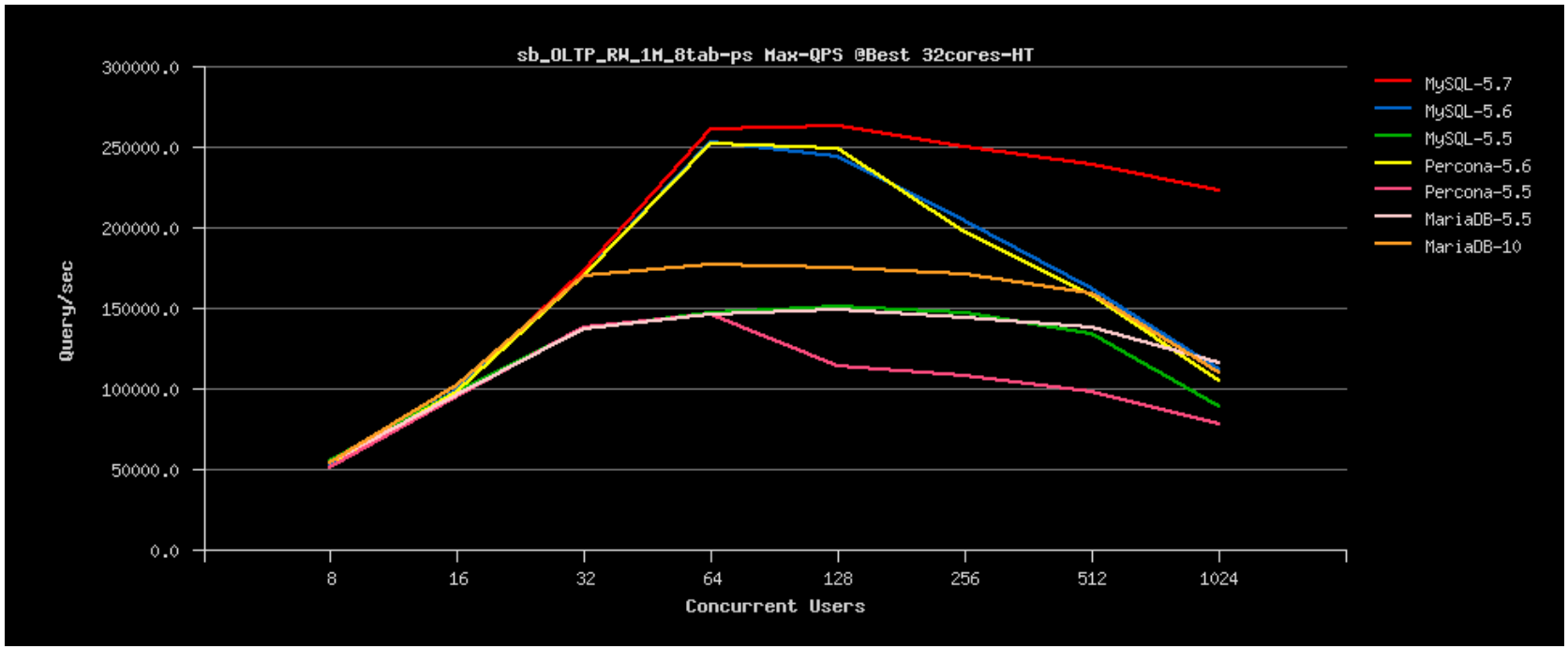
MySQL 5.7: DMR2 (Sep.2013)

- OLTP_RO 1-table: lower than 5.6...
  - Due higher MDL contentions, work in progress..



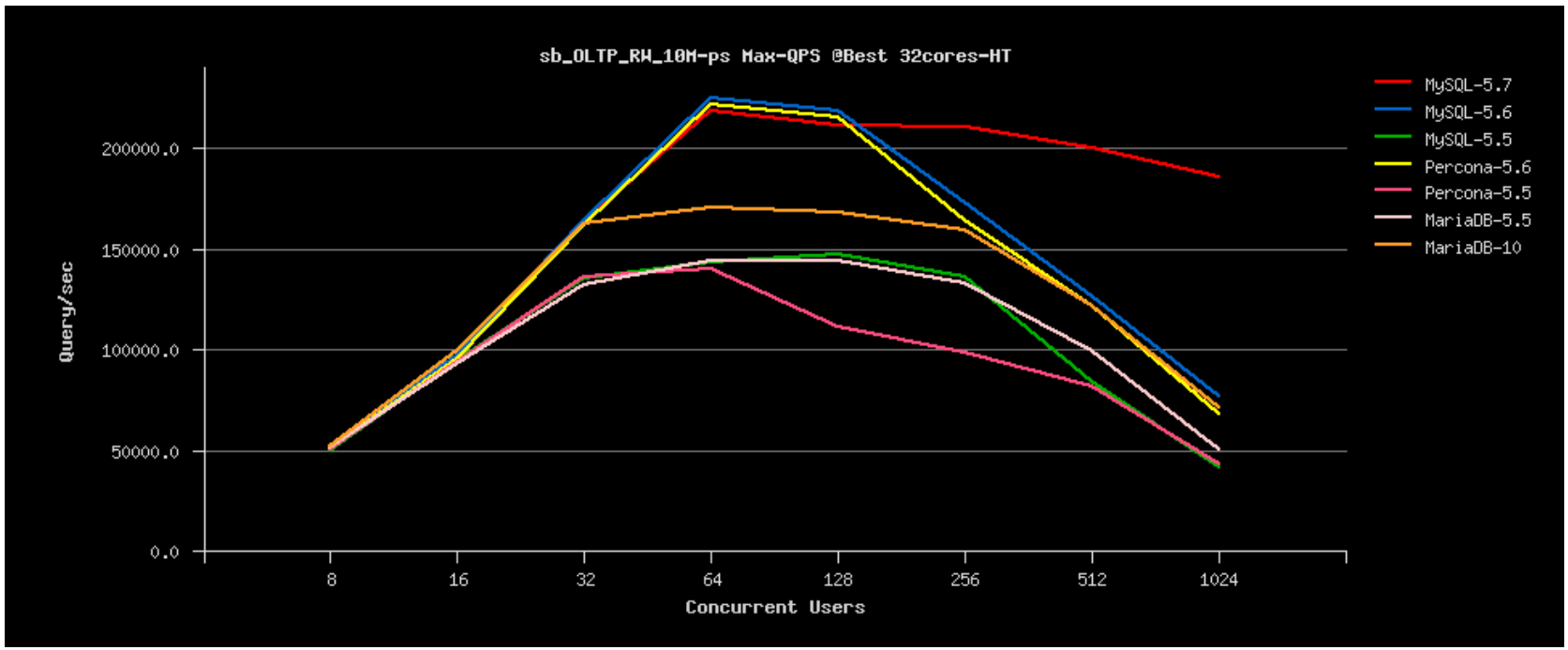
MySQL 5.7: DMR2 (Sep.2013)

- OLTP_RW 8-tables: **265K** QPS
  - IP port, sysbench 0.4.13 (“common”, using more CPU)



MySQL 5.7: DMR2 (Sep.2013)

- OLTP_RW 1-table: lower than 5.6
  - MDL contention, work in progress..



THANK YOU !!!

- All details about presented materials you may find on:
  - http://dimitrik.free.fr - dim_STAT, Benchmark Reports
  - http://dimitrik.free.fr/blog - Articles about MySQL Performance