MySQL

ORACLE

MySQL 8.0 Performance: Scalability & Benchmarks

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

- Overview of MySQL Performance
- Where we're with MySQL 8.0 & Benchmark results..
  - RO / Skylake / UTF8..
  - RW / new REDO / Resource Groups
  - IO-bound / Optane
  - Sysbench / TPCC-like / dbSTRESS
- Pending issues..
- Q & A

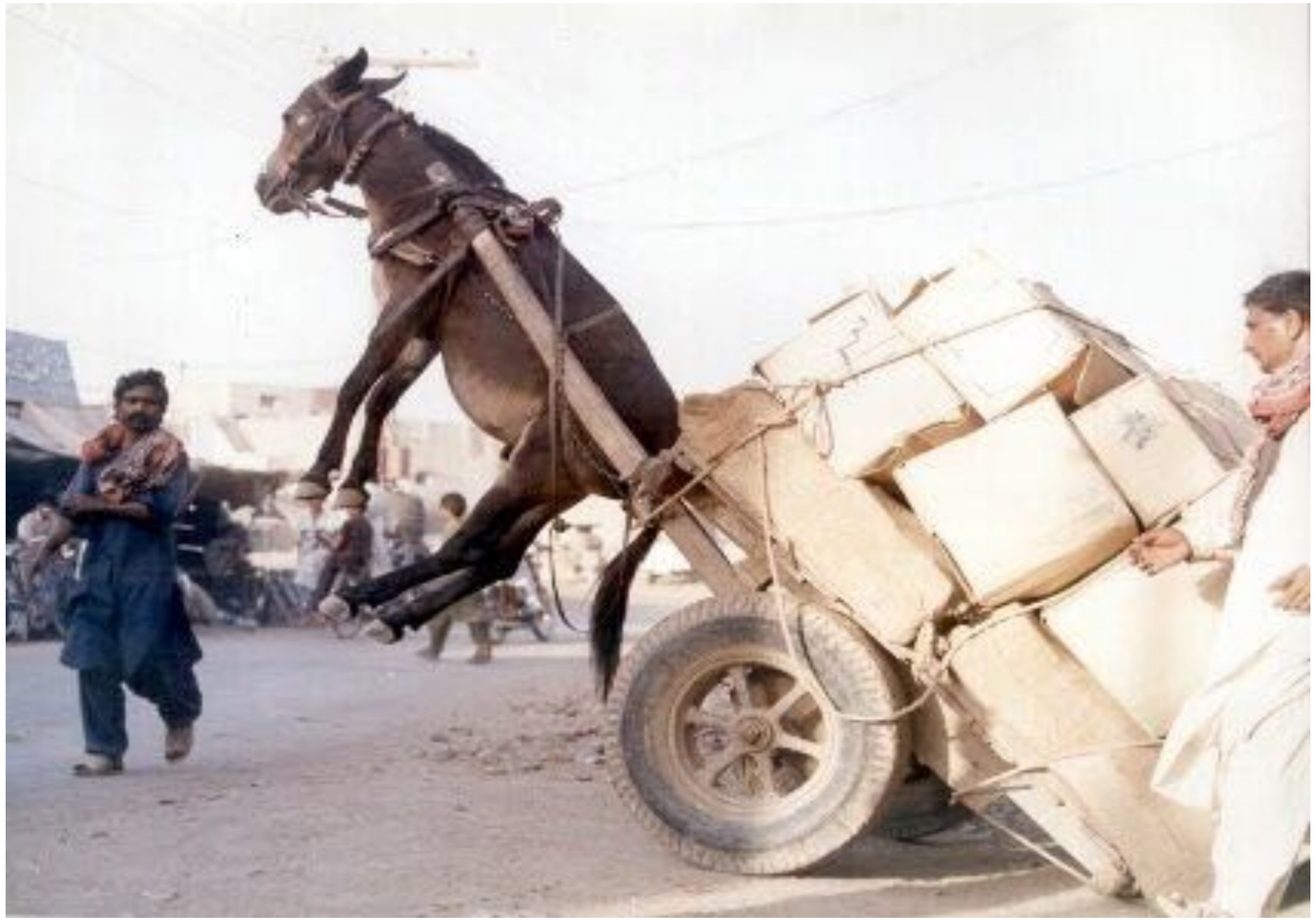
Why MySQL Performance ?...

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 ?..

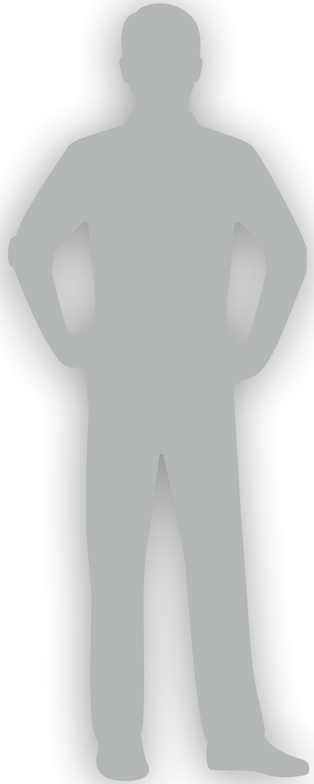
- Analyzing your workload performance and testing your limits may help you to understand ahead the resistance of your solution to incoming potential problems ;-)

Why MySQL Performance ?..

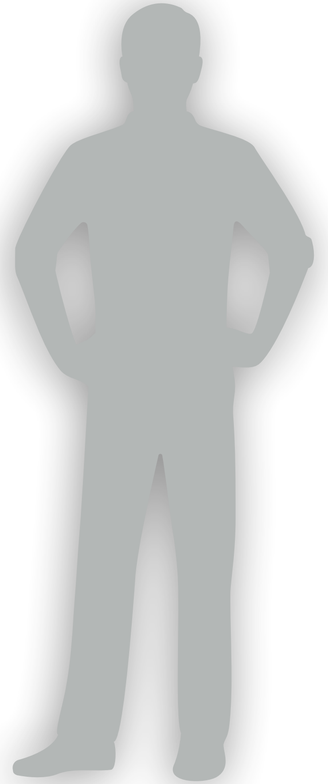
- However :
  - Even a very powerful solution but leaved in wrong hands may still be easily broken!... :-)

Why MySQL Performance ?

Is Performance your priority #1 ?



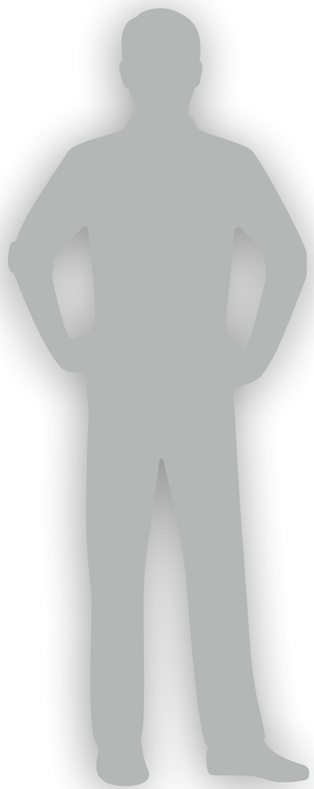
Dimitri



DB Guru

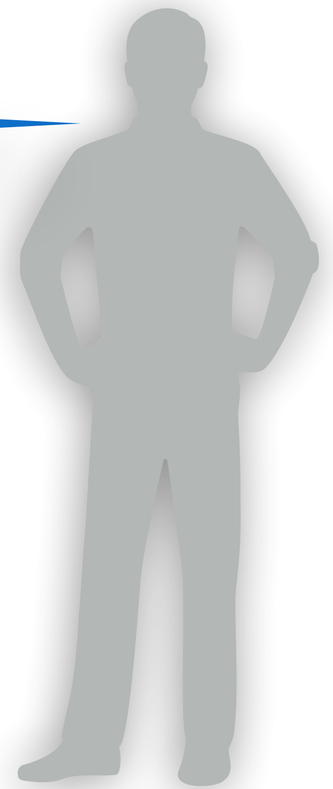
ORACLE

Why MySQL Performance ?



Is Performance your priority #1 ?

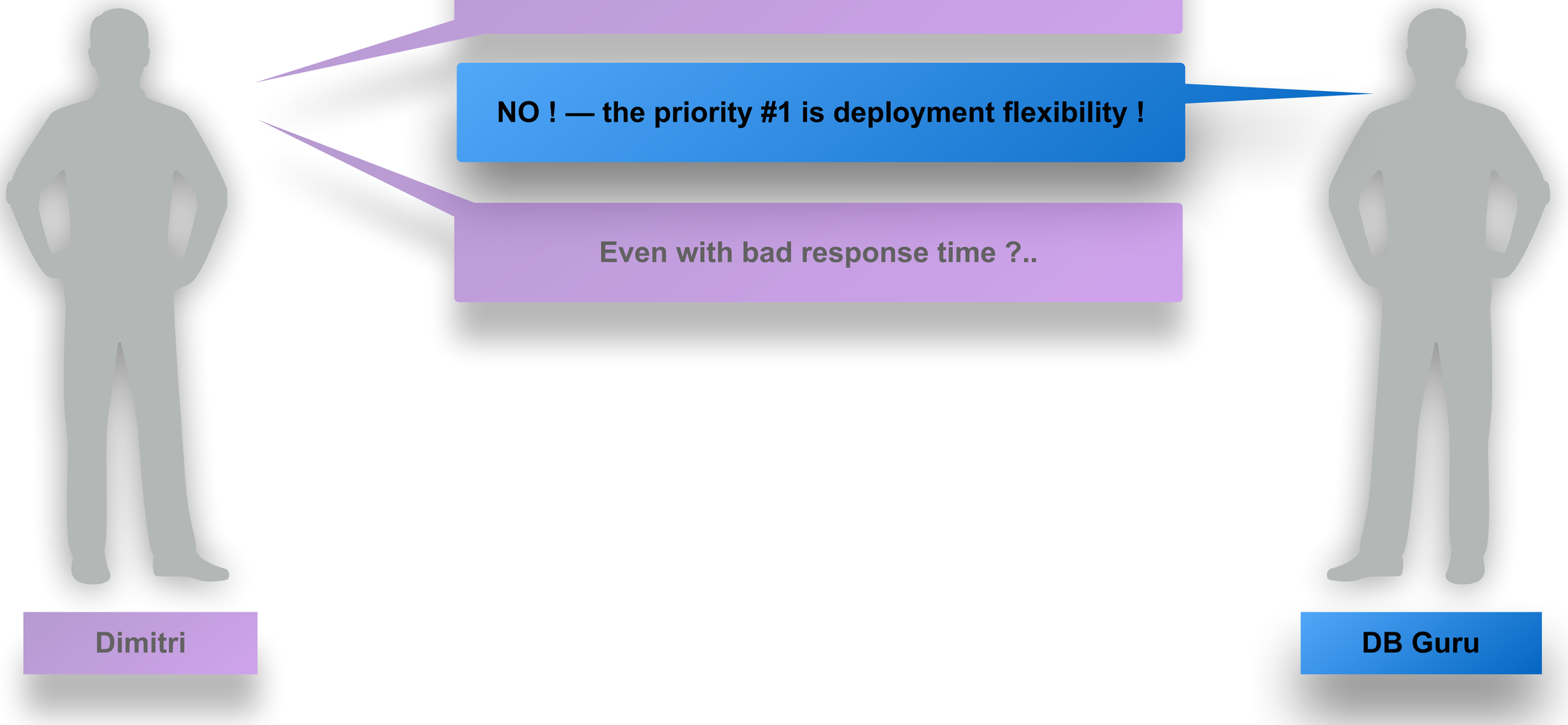
NO ! — the priority #1 is deployment flexibility !



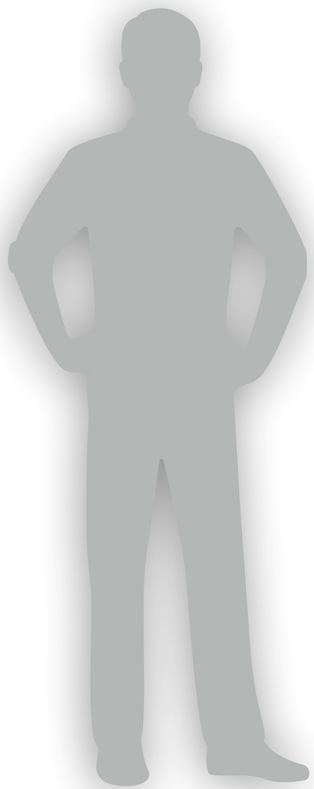
Dimitri

DB Guru

Why MySQL Performance ?



Why MySQL Performance ?



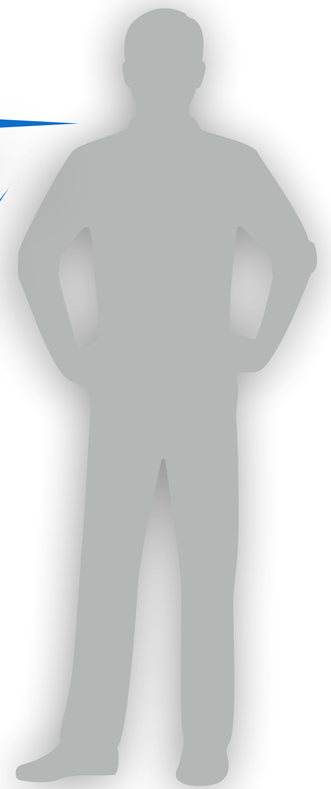
Dimitri

Is Performance your priority #1 ?

NO ! — the priority #1 is deployment flexibility !

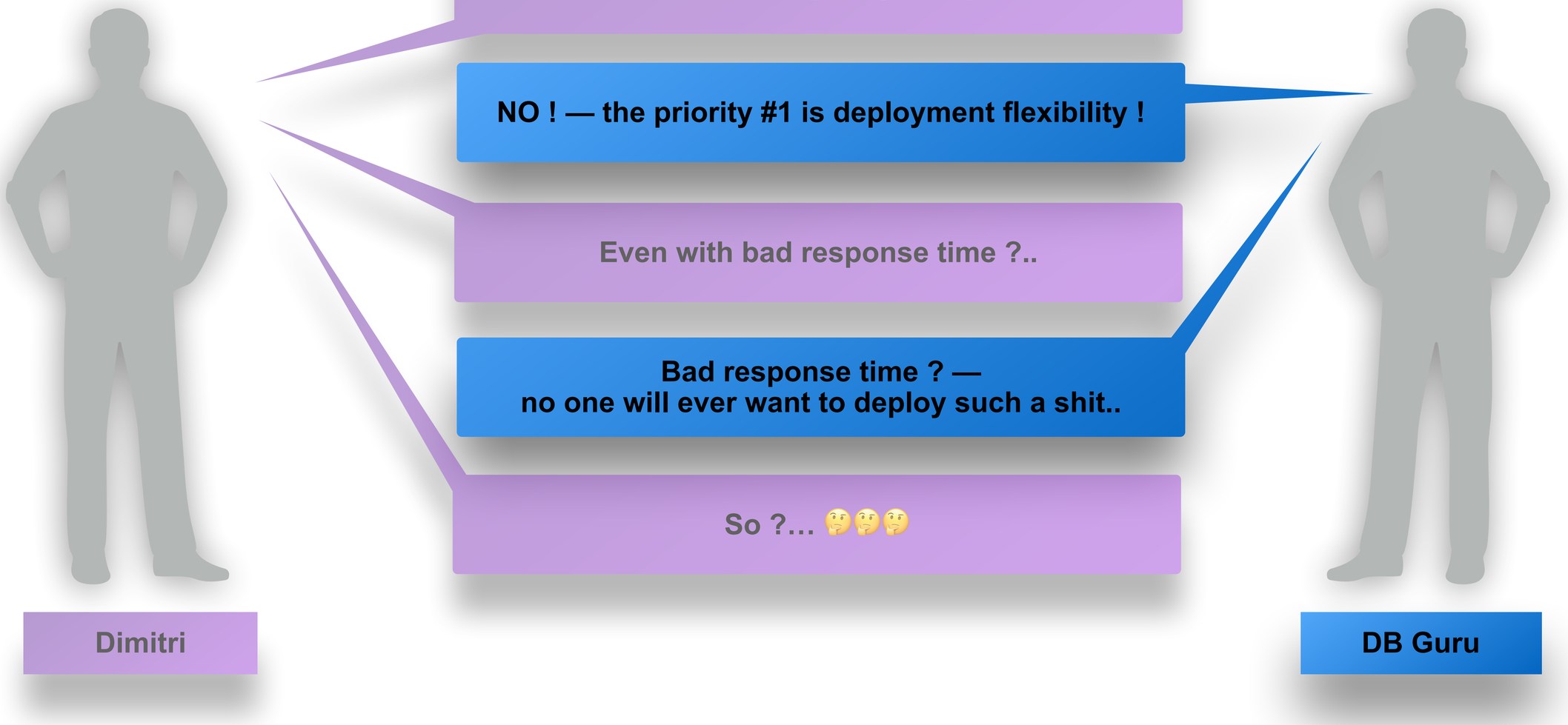
Even with bad response time ?..

Bad response time ? —
no one will ever want to deploy such a shit..



DB Guru

Why MySQL Performance ?



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

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

USE YOUR BRAIN !!! ;-)

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

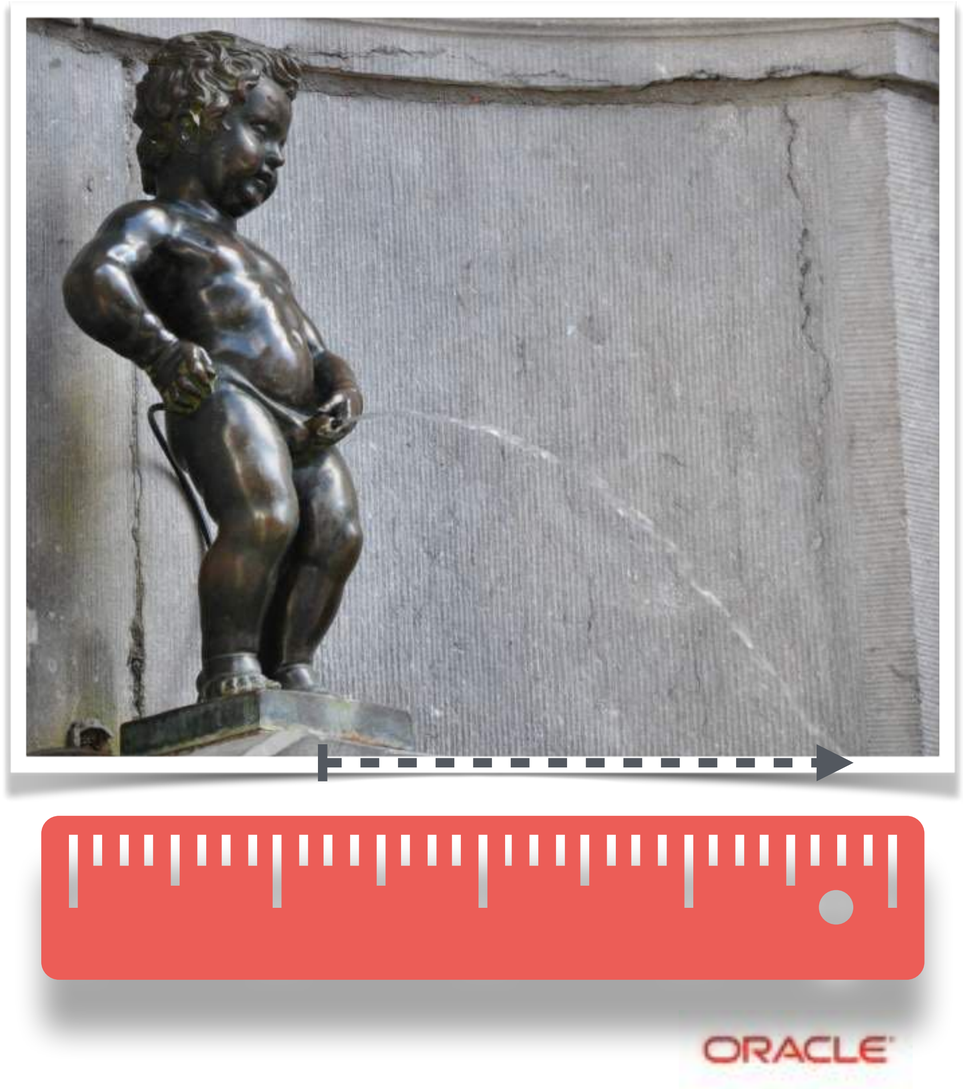
USE YOUR BRAIN !!! ;-)

Common Sources of MySQL Performance Problems..

- “Fixable” ones ;-)
  - DB Schema/ Indexes/ SQL query/ Optimizer plan/ Apps code/ etc. etc..
  - odd tuning/ wrong config setup/
  - e.g. generally can be fixed by => RTFM ! ;-)
- “By design” ones..
  - known ?..
  - workaround ?..
  - can be ever fixed ?..
  - heh…
  - work in progress.. <= and here is where we come ;-))

Why Benchmarks ?

- Common perception of benchmarks is often odd..
  - “not matching real world”…
  - “pure Marketing”..
  - “pure BenchMarketing”..
  - etc. etc. etc..
- well..
  - “it depends..” ©
  - get your own opinion by understanding of the tested workloads !
  - e.g. remind Best Practice #1 ;-))



Benchmarks & MySQL

- Every test workload is pointing to a problem to resolve !
  - evaluate & understand the problem(s)
  - then try to fix it
  - or propose a workaround
  - evaluate & confirm the fix / workaround
  - keep running in QA to discover any potential regression ON TIME !..

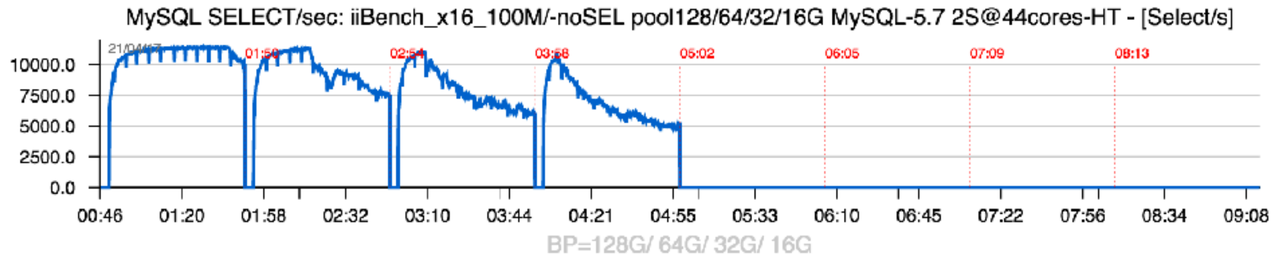
- As well :
  - kind of “reference” of what to expect
  - evaluate any new HW, systems, etc..

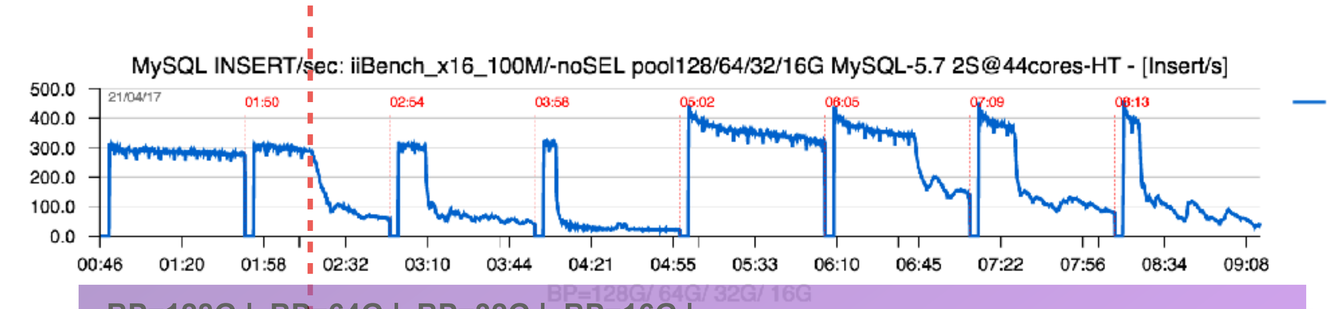
Example: iiBench (INSERT Benchmark)

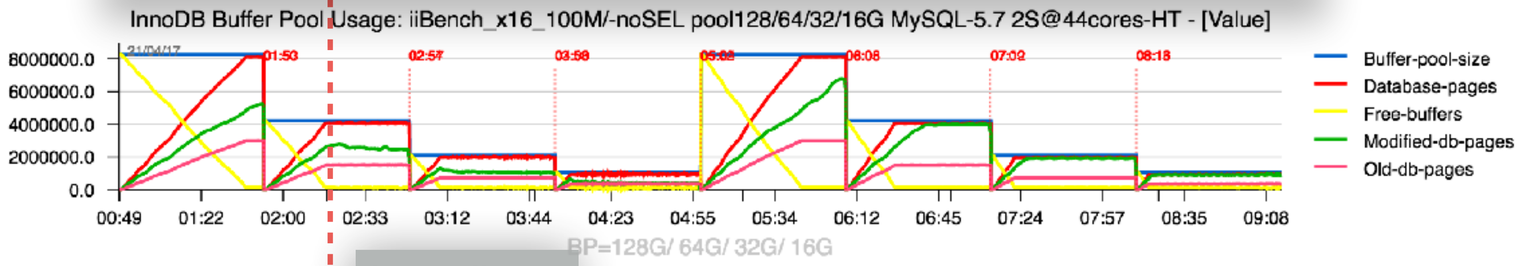
- Main claim :
  - InnoDB is xN times slower vs Write-oriented Engine XXX
  - so, use XXX, as it’s better
- Test Scenario :
  - x16 parallel iiBench processes running together during 1H
  - each process is using its own table
  - one test with SELECTs, another without..
- Key point :
  - during INSERT activity, B-Tree index in InnoDB growing quickly
  - as soon as index pages have no more place in BP and re-read from storage, performance is going down..
  - e.g. “by design” problem ;-))

iiBench 100M x16 : BP= 128G/ 64G/ 32G/ 16G

- Observations :
  - until B-Tree remains in BP => 300K INSERT/sec.. (and if not, QPS drop)



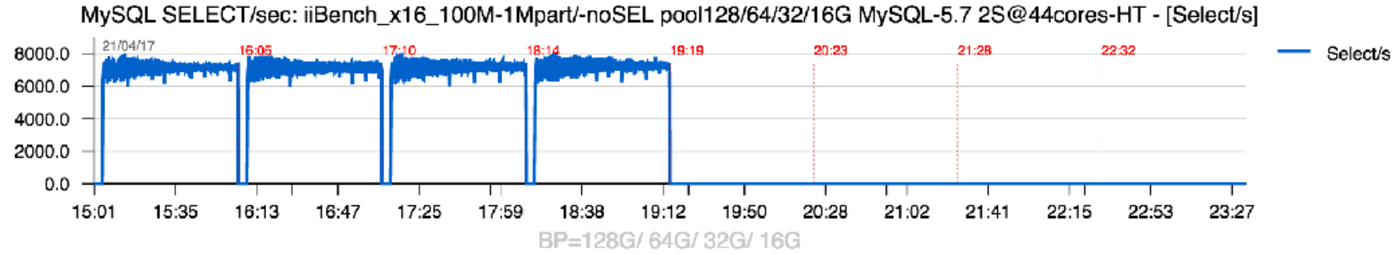


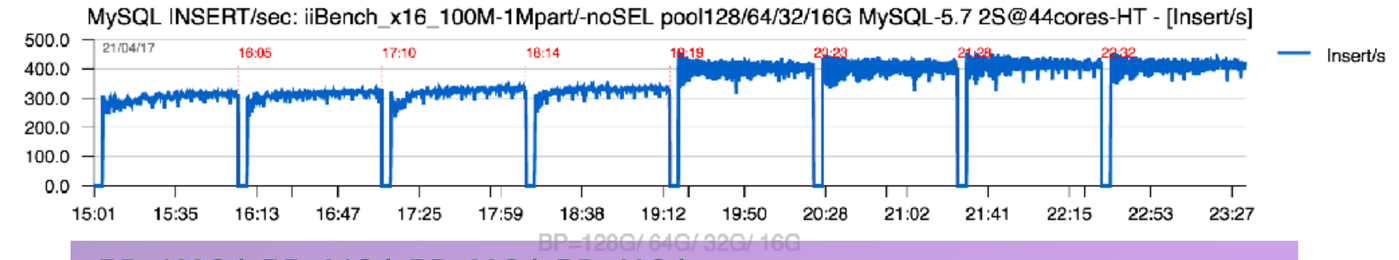


free buffers !!!

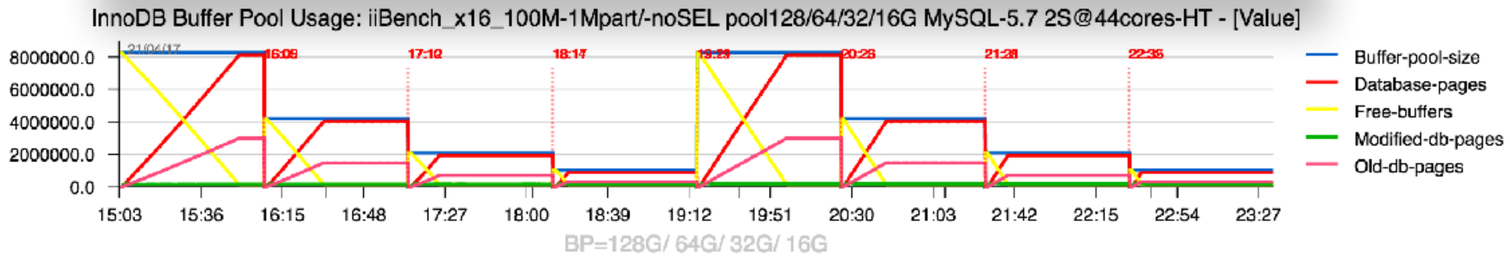
iiBench 100M x16 & 1M-parts : BP= 128G/ 64G/ 32G/ 16G

- Observations :
  - workaround : keep index cached in BP (via table partitions or other)





BP=128G | BP=64G | BP=32G | BP=16G |

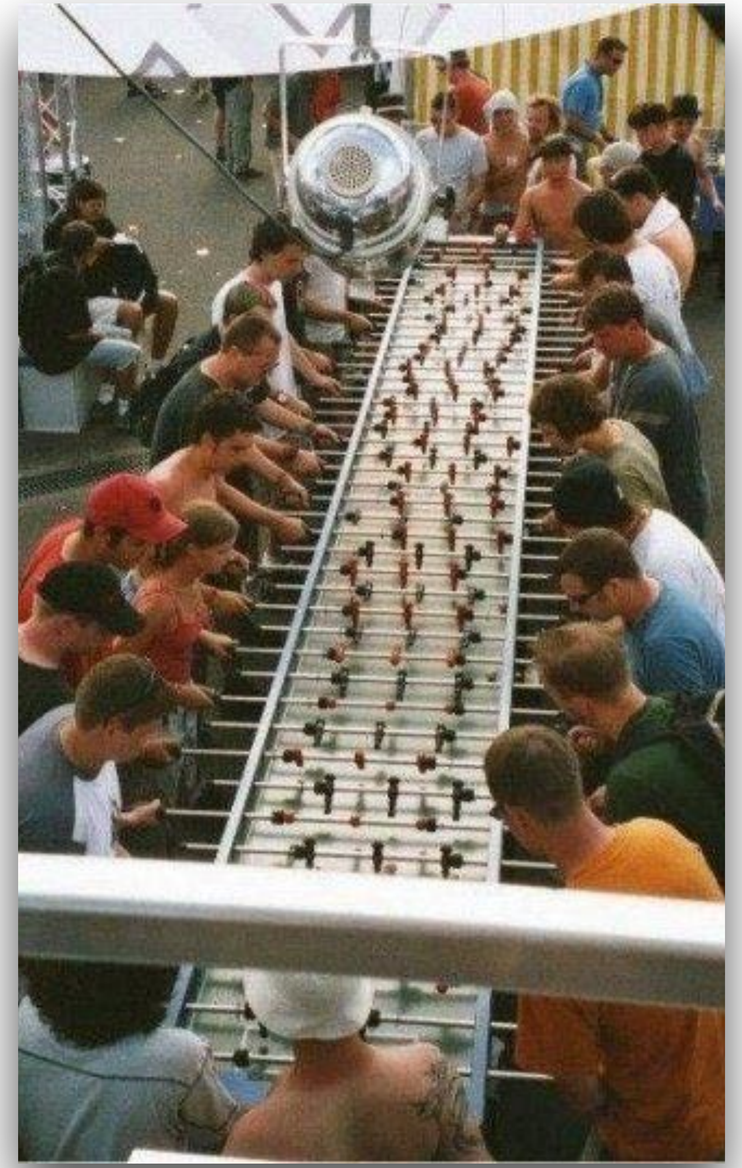


Test Workload

- 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..” ;-))
  - And even if you were doing something wrong, try to understand its impact..
  - (Best Practice #1 once again ;-))

The Best Test Workload

- For You :
  - workload simulating your Production !
  - JMetter (free)
  - LoadRunner ($$)
  - etc.
- otherwise :
  - use “generic” test workloads
  - change / adapt / extend..
  - **share** with us ! ;-))



“Generic” Test Workloads @MySQL

- Sysbench - #1
  - The “Entry Ticket” Workloads - looks simple, but still the most complete test kit !
  - OLTP, RO/RW, points on various RO and RW issues
- DBT2 / TPCC-like
  - OLTP, RW, pretty complex, growing db, no options, deadlocks
- DBT3
  - DWH, RO, complex heavy queries, loved by Optimizer Team ;-)
- dbSTRESS
  - OLTP, RO/RW, several tables, points on RW and Sec.IDX issues
- iiBench
  - pure INSERT bombarding + optionally SELECTs, points on B-Tree issues
- LinkBench (Facebook)
  - OLTP, RW, looks intensive and IO-hungry

Why Sysbench is #1 ?..

- Historically :
  - the most simple to install, to use, most lightweight
  - why entry ticket : covers most important “key workload cases” in MySQL performance

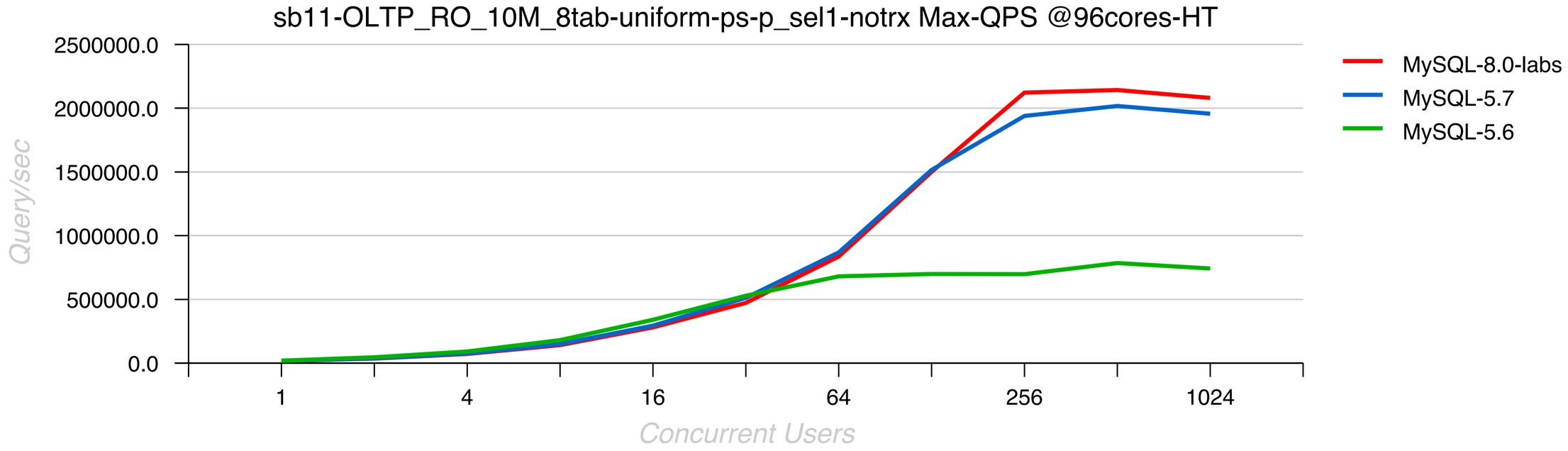
- New Sysbench :
  - https://github.com/akopytov/sysbench
  - have fixed all past issues
  - high flexibility for any test scenario with LUA scripts
  - integrated LUA JIT => high execution speed + lightweight !
  - more various test scenarios are expected to come
  - excellent opportunity to write your own test cases !
  - move and use it now ! ;-)

MySQL Scalability Milestones

- MySQL 5.5
  - delivered “already known” solutions (except BP instances and few other)..
- MySQL 5.6
  - first fundamental changes (kernel_mutex split, G5 patch, RO transactions, etc..)
  - but : RW workloads are faster than RO ! ;-))
- MySQL 5.7
  - finally fully unlocked Read-Only + no more contentions on the “Server” layer, etc..
  - and (finally) RO is faster than RW ;-))
- MySQL 8.0
  - main focus is on efficiency : do more on the same HW ;-))
  - main target HW : 2CPU Sockets systems
  - RW scalability.. & data security..
  - NOTE : Continuous Release Model !

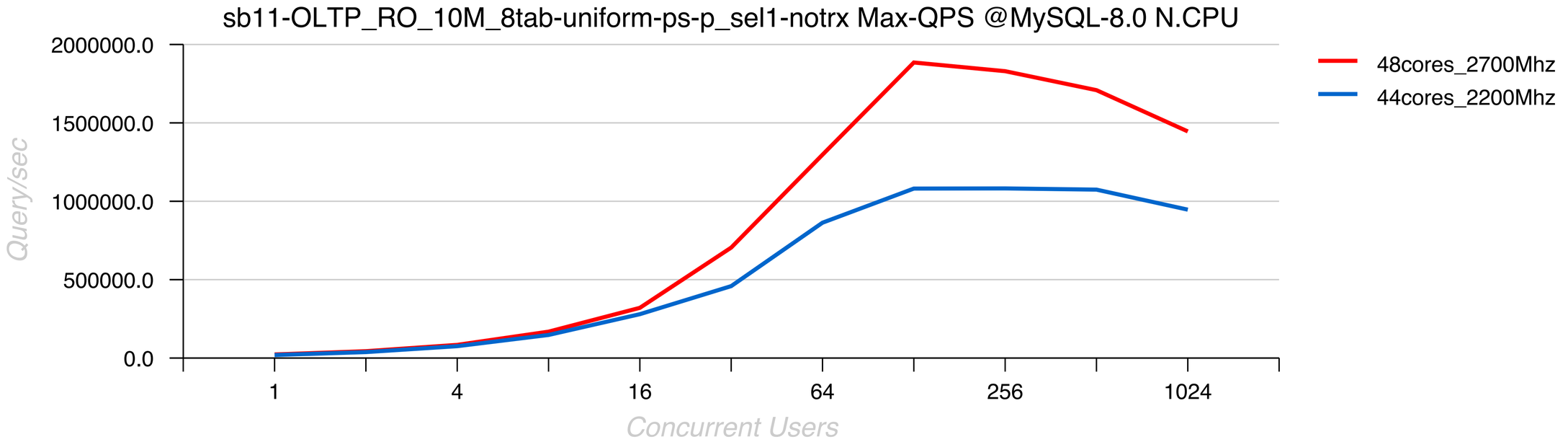
RO Point-Selects @MySQL 8.0-labs (Sep.2017)

- **2.1M (!!) QPS** Sysbench Point-Selects 10Mx8tab :
  - 4CPU Sockets (4S) : 96cores-HT Broadwell v4
  - the main gain : NO regression -vs- 5.7 !! (and I'm serious ;-))



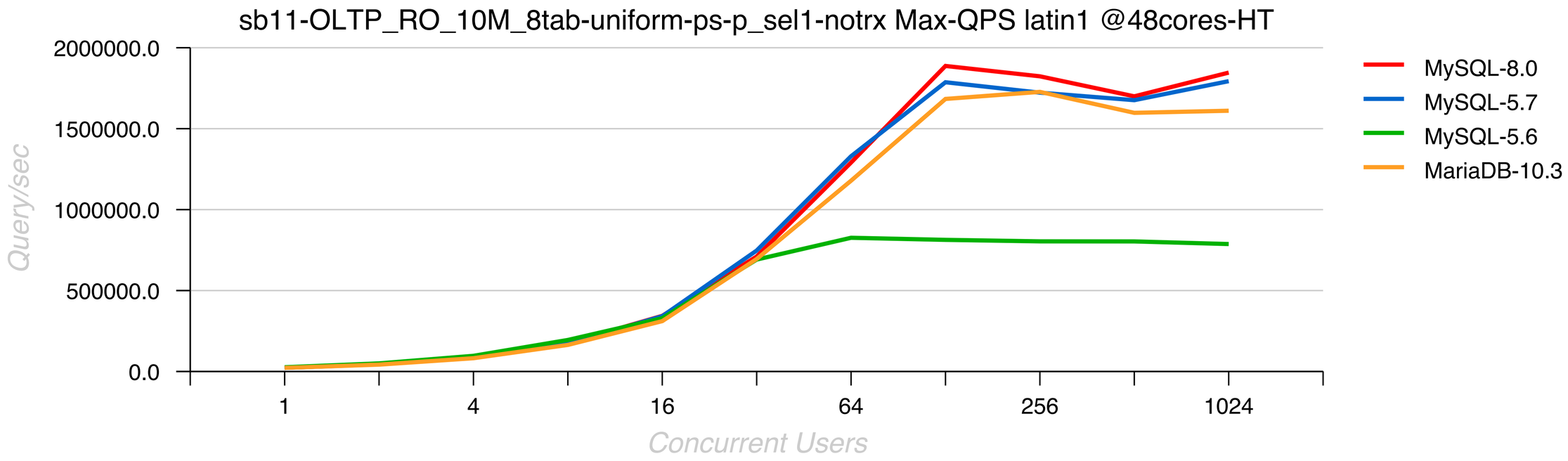
RO Point-Selects @MySQL 8.0 (Dec.2017)

- Sysbench Point-Selects 10Mx8tab @2CPU Sockets
  - 2CPU Sockets Broadwell v4 : 44cores-HT
  - 2CPU Sockets **Skylake** Platinum : 48cores-HT
  - => near **80**% gain in peak QPS !! already 50% on 32usr load !!



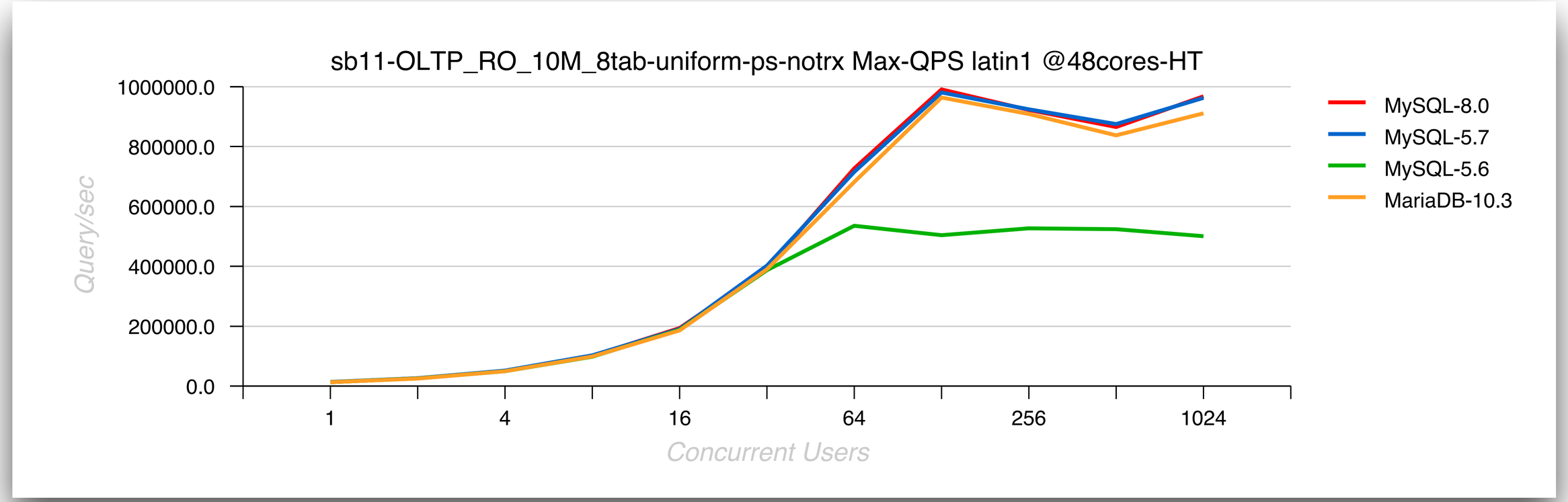
RO Point-Selects @MySQL 8.0 (Apr.2018)

- **1.8M+ (!!) QPS** Sysbench Point-Selects 10Mx8tab @2CPU Sockets
  - 2CPU Sockets **Skylake** Platinum : 48cores-HT
  - => currently our best results on 2S HW



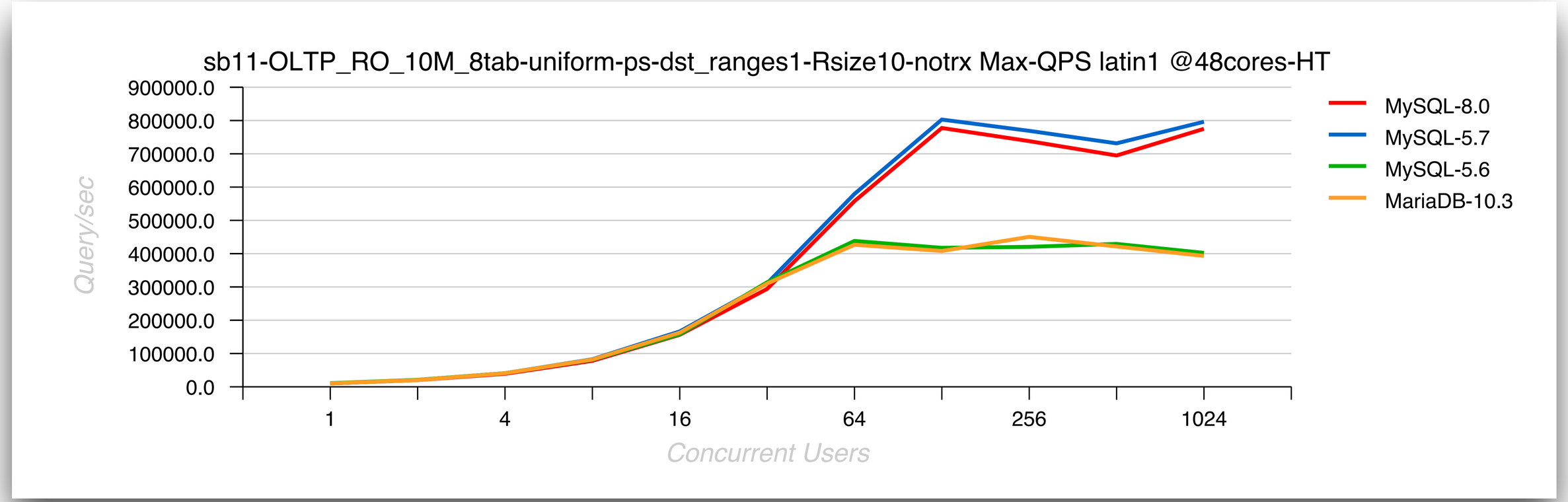
OLTP_RO @MySQL 8.0 (Apr.2018)

- **1M (!!) QPS** Sysbench OLTP_RO 10Mx8tab @2CPU Sockets
  - 2CPU Sockets **Skylake** Platinum : 48cores-HT



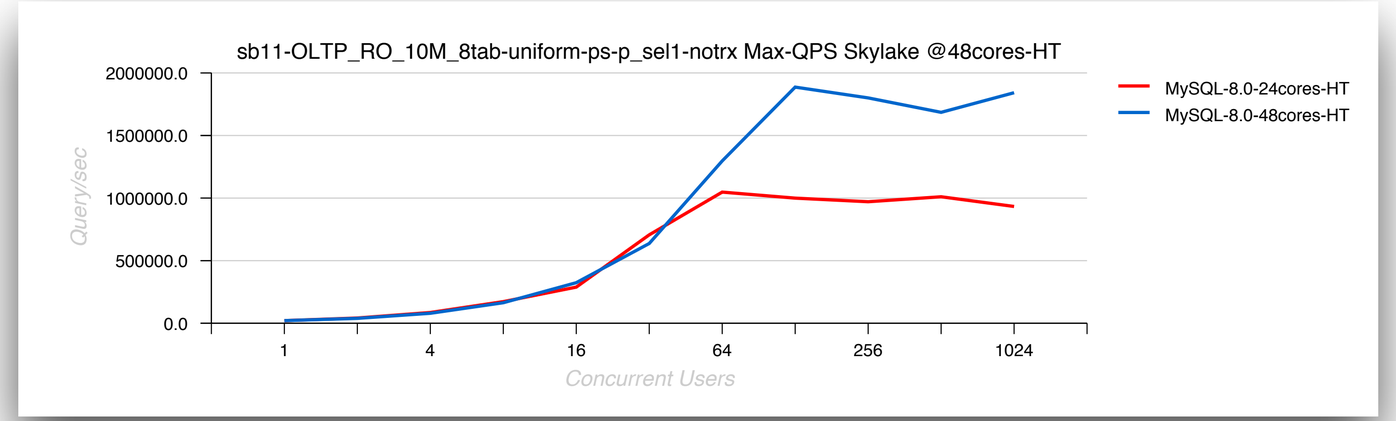
RO Distinct-Ranges @MySQL 8.0 (Apr.2018)

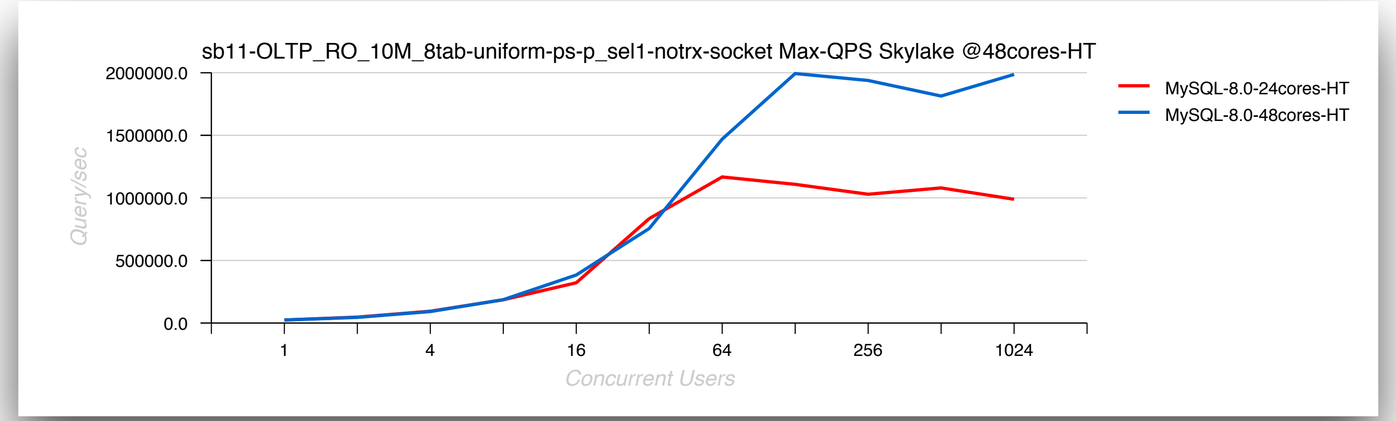
- **790K QPS** Sysbench OLTP_RO 10Mx8tab @2CPU Sockets
  - 2CPU Sockets **Skylake** Platinum : 48cores-HT
  - => first signs of regression in 8.0..
  - => no comments about MariaDB..



Point-Selects : IP port -vs- UNIX socket

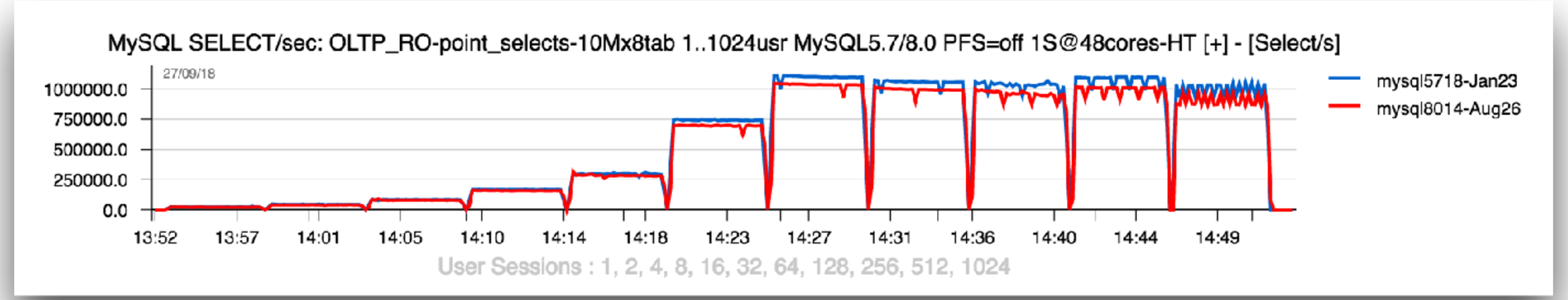
- Skylake 2S, 48cores-HT :
  - up to **19%** difference !
  - HW ? OS kernel ?
  - IP stack ?
  - under investigations…

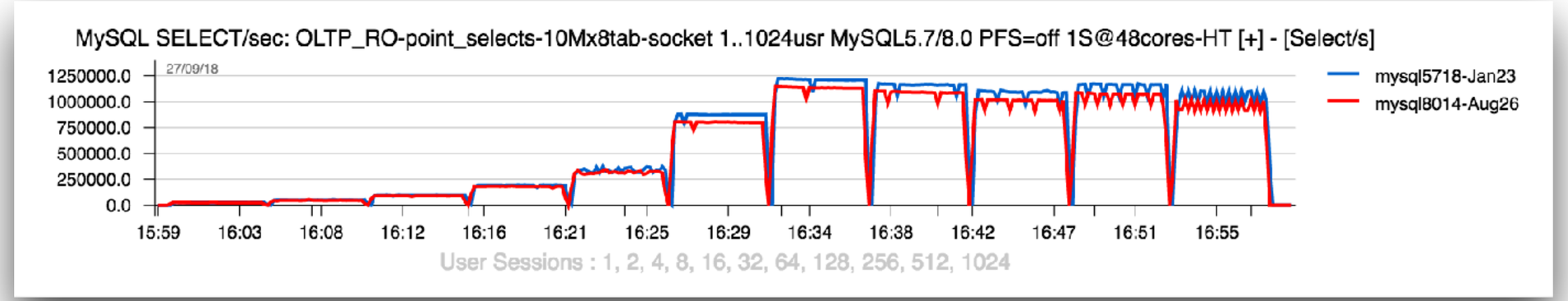




Point-Selects : MySQL 5.7 -vs- 8.0 (Oct.2018)

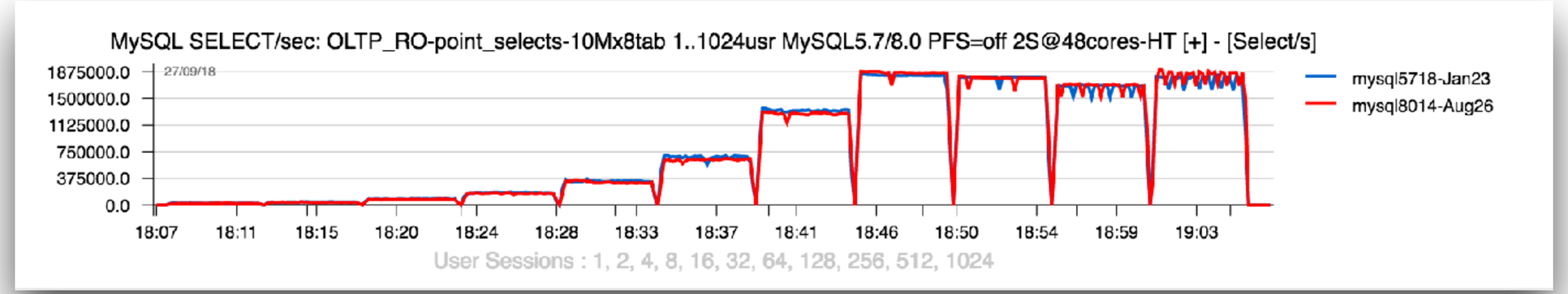
- Skylake 1S (24cores-HT)
  - 8.0 is near 10% lower..

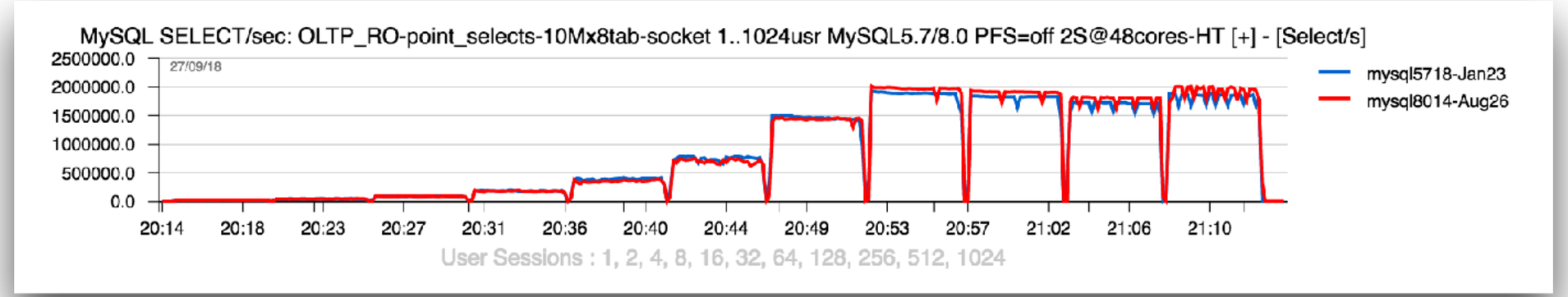




Point-Selects : MySQL 5.7 -vs- 8.0 (Oct.2018)

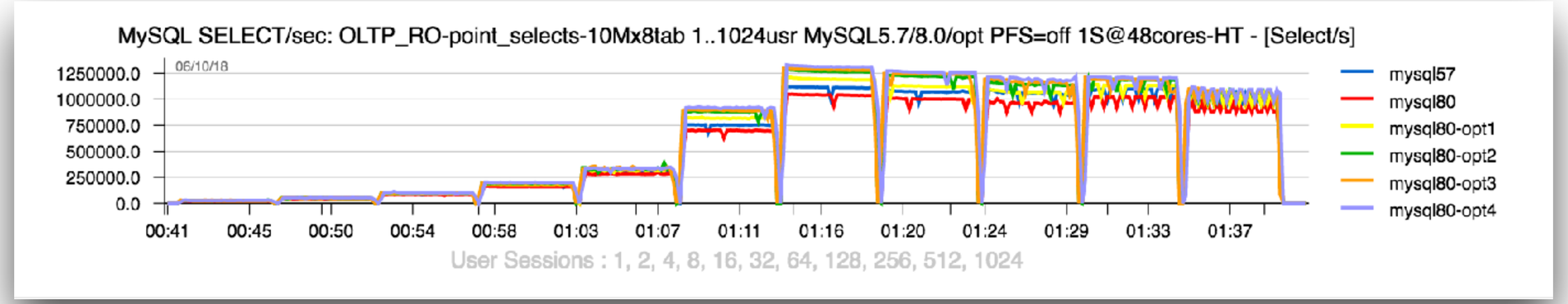
- Skylake 2S (48cores-HT)
  - 8.0 is little bit higher..

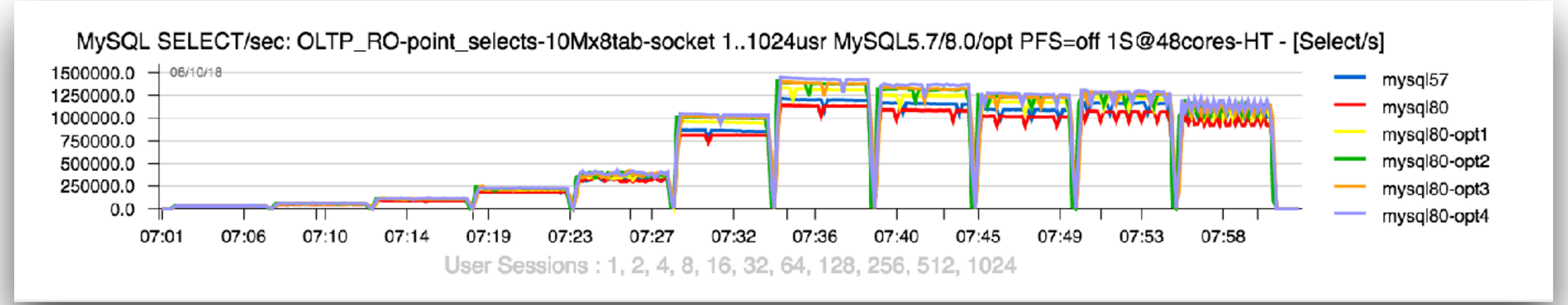




Point-Selects : MySQL 5.7 -vs- 8.0 (Oct.2018)

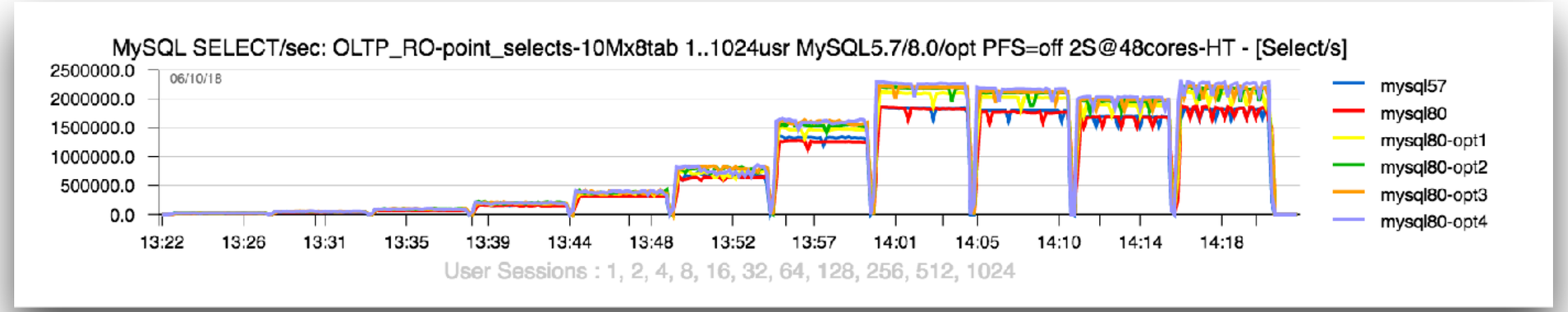
- Skylake 1S (24cores-HT) — the good news ;-))
  - 8.0 is largely higher..

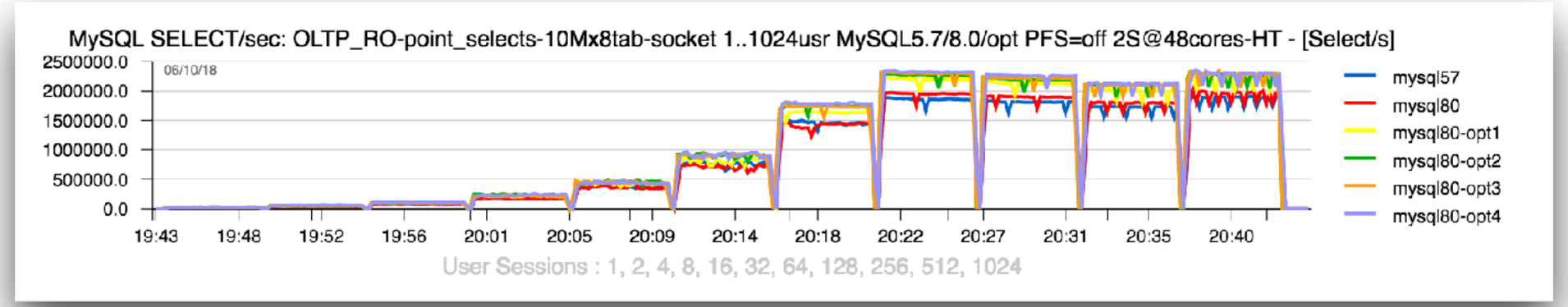




Point-Selects : MySQL 5.7 -vs- 8.0 (Oct.2018)

- Skylake 2S (48cores-HT) — the good news ;-))
  - 8.0 is largely higher..





ORACLE

Pending RO Scalability Issues after MySQL 5.7 GA..

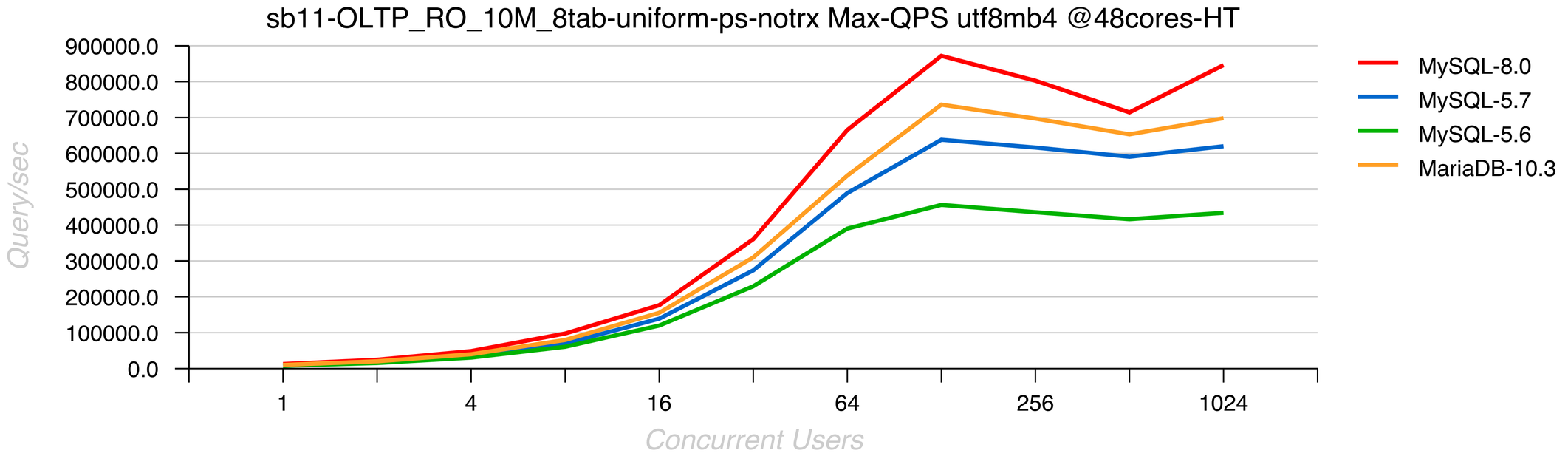
- RO :
  - Block Locks
  - Lookups via Sec.IDX
  - UTF8
  - Global lock on every IO read

Pending RO Scalability Issues after MySQL 5.7 GA..

- RO :
  - Block Locks <= workaround : ProxySQL Query Cache
  - Lookups via Sec.IDX <= possible workaround : use PK, AHI
  - UTF8 <= **use 8.0** ;-)
  - Global lock on every IO read <= **use 8.0** ;-)

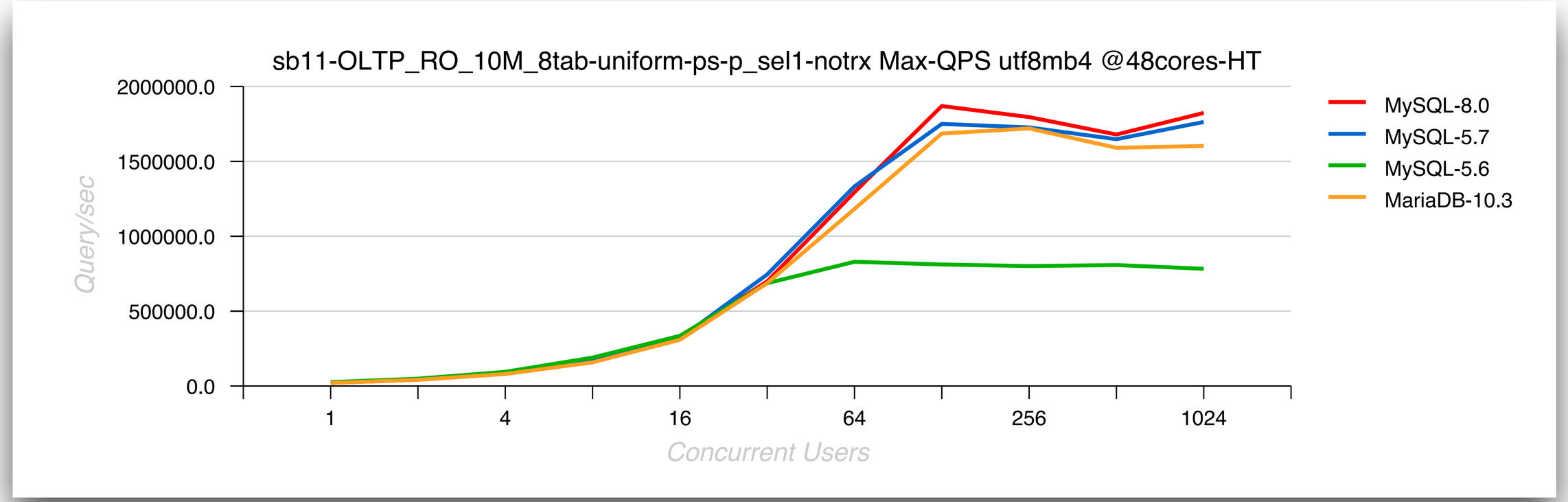
OLTP_RO UTF8mb4 @MySQL 8.0 (Apr.2018)

- **870K(!!) QPS** Sysbench OLTP_RO 10Mx8tab @2CPU Sockets
  - 2CPU Sockets **Skylake** Platinum : 48cores-HT
  - => up to 40% better than 5.7



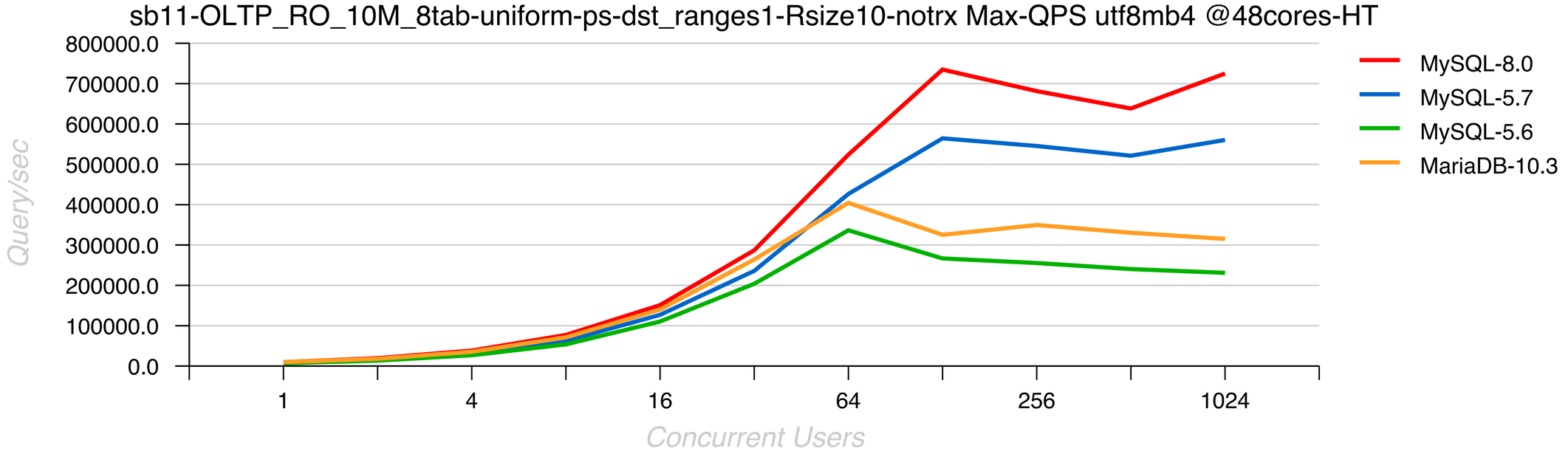
RO Point-Selects UTF8mb4 @MySQL 8.0 (Apr.2018)

- **Same QPS** Sysbench RO point-selects 10Mx8tab @2CPU Sockets
  - 2CPU Sockets **Skylake** Platinum : 48cores-HT



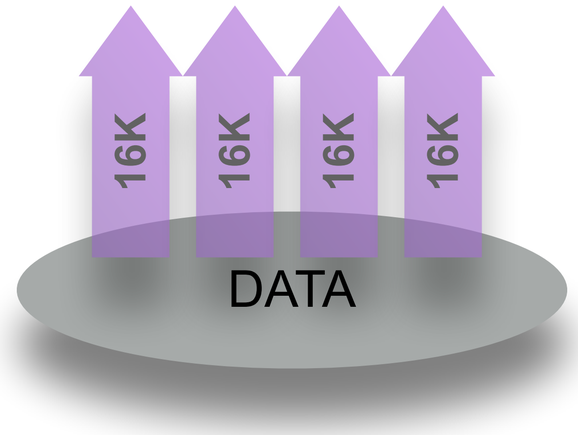
RO Distinct-Ranges UTF8mb4 @MySQL 8.0 (Apr.2018)

- **735K (!!) QPS** Sysbench RO dist-ranges 10Mx8tab @2CPU Sockets
  - 2CPU Sockets **Skylake** Platinum : 48cores-HT
  - => up to 30% better than 5.7



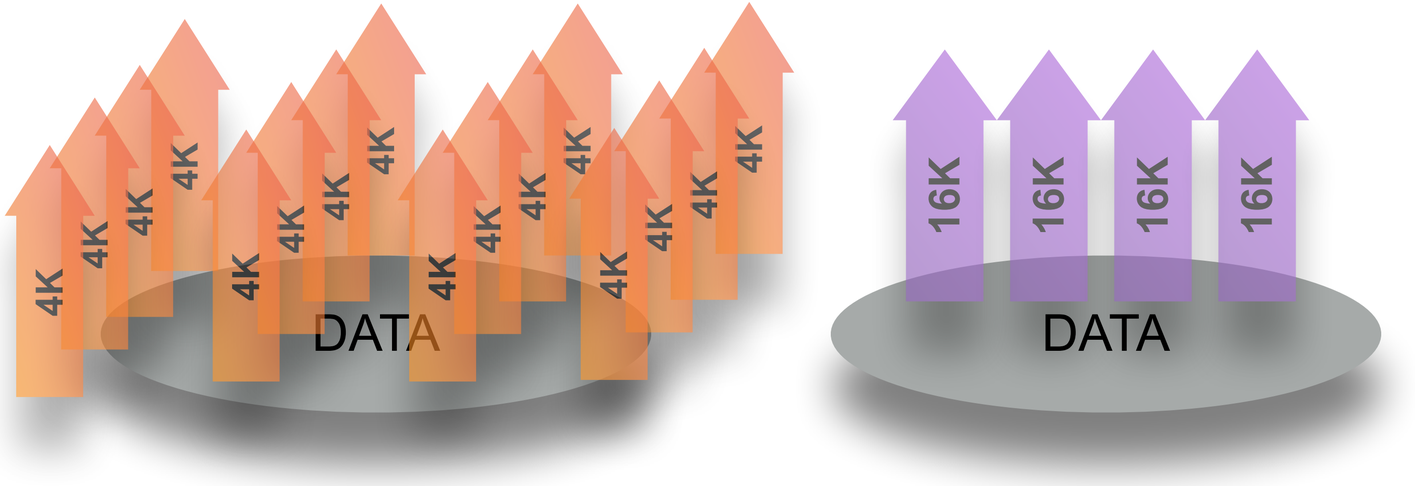
IO-bound Workloads : Game Changer..

- IO reads :
  - game changer : **FLASH** => goes faster / cheaper / more stable / living longer / etc..
  - e.g. no more “seek time” cost, the main IO limit : device throughput
  - supposing your max throughput is XXX MB/sec, what is the max IO-bound QPS possible ?
  - => driven by IO read **Operations/sec** …



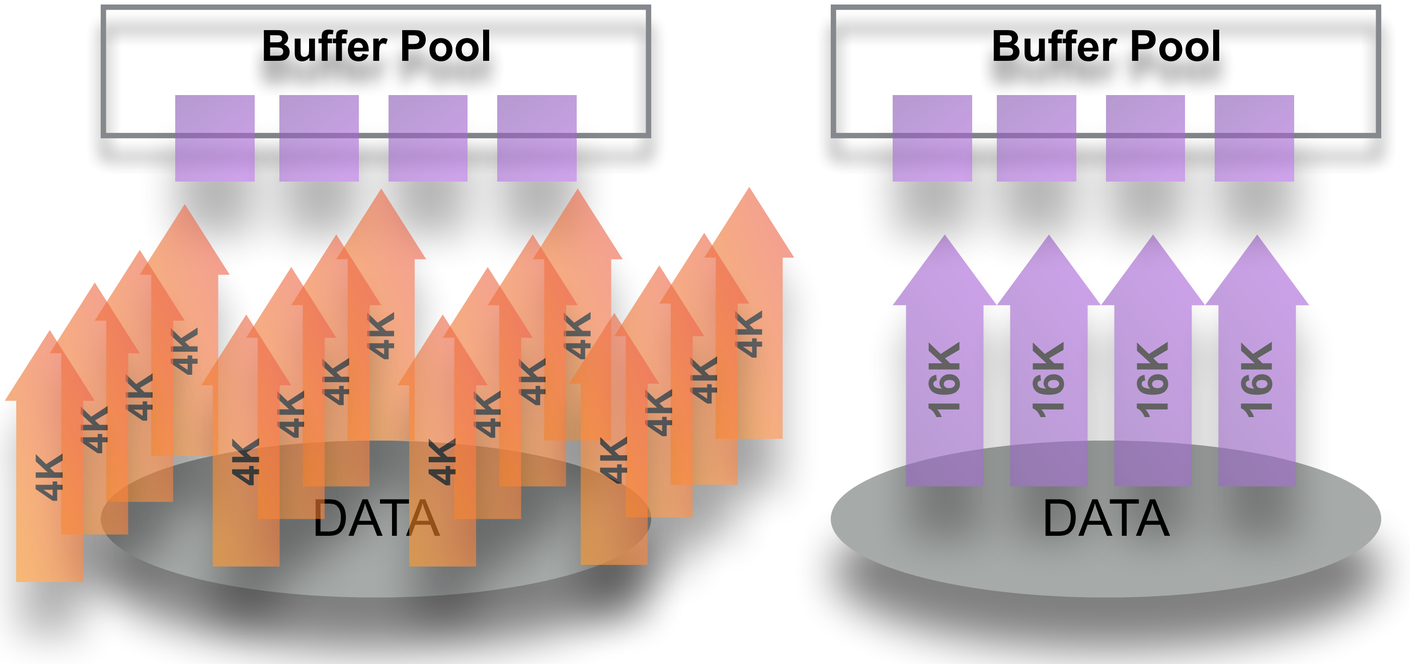
IO-bound Workloads : more in depth..

- IO reads :
  - game changer : FLASH => goes faster / cheaper / more stable / living longer / etc..
  - e.g. no more “seek time” cost, the main IO limit : device throughput
  - supposing your max throughput is XXX MB/sec, what is the max IO-bound QPS possible ?
  - => driven by IO read **Operations/sec** …
  - x4 Compression ? => x4 times more IO reads !!!



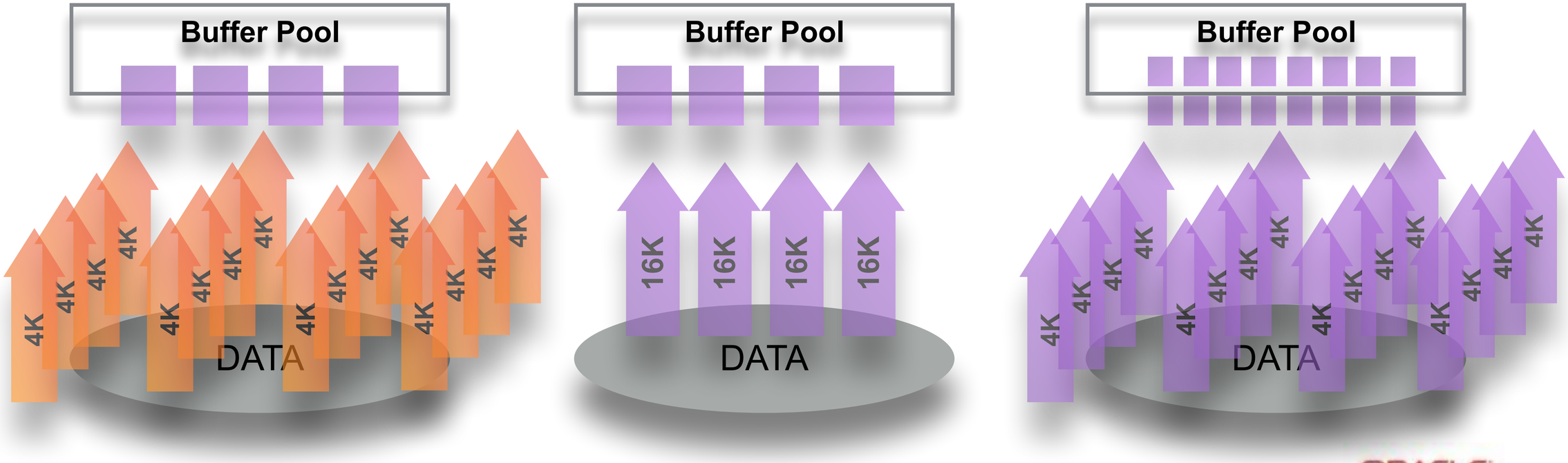
IO-bound Workloads : more in depth..

- IO reads :
  - game changer : FLASH => goes faster / cheaper / more stable / living longer / etc..
  - e.g. no more “seek time” cost, the main IO limit : device throughput
  - supposing your max throughput is XXX MB/sec, what is the max IO-bound QPS possible ?
  - => driven by IO read **Operations/sec** …
  - x4 Compression ? => x4 times more IO reads !!! => **and QPS** ?..



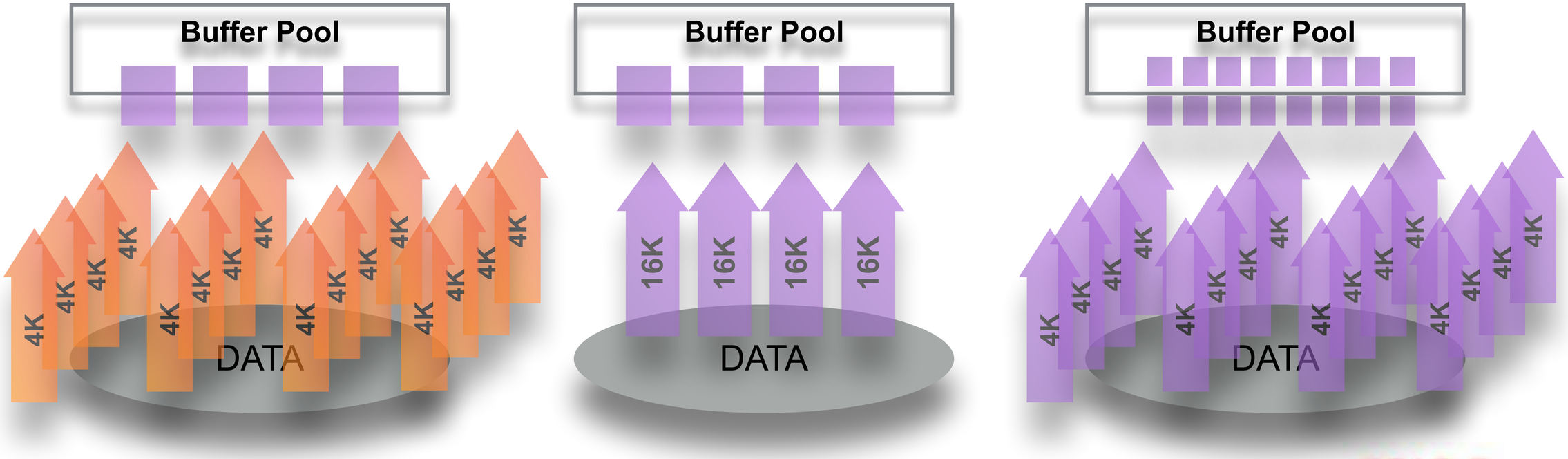
IO-bound Workloads : more in depth..

- IO reads :
  - game changer : FLASH => goes faster / cheaper / more stable / living longer / etc..
  - e.g. no more “seek time” cost, the main IO limit : device throughput
  - supposing your max throughput is XXX MB/sec, what is the max IO-bound QPS possible ?
  - => driven by IO read **Operations/sec** …
  - x4 Compression ? => x4 times more IO reads !!! => and QPS ?.. **and if 4K page ?**



IO-bound Workloads : more in depth..

- IO reads :
  - so, with fast FLASH + 4K page size => x4 times better RO performance vs default 16K ?
  - potentially YES ;-))
  - but.. => historically : **fil_system** global mutex lock on **every IO operation !!!**
  - good news : **fixed since MySQL 8.0 ! ;-))**



IO-bound Workloads : Test Case

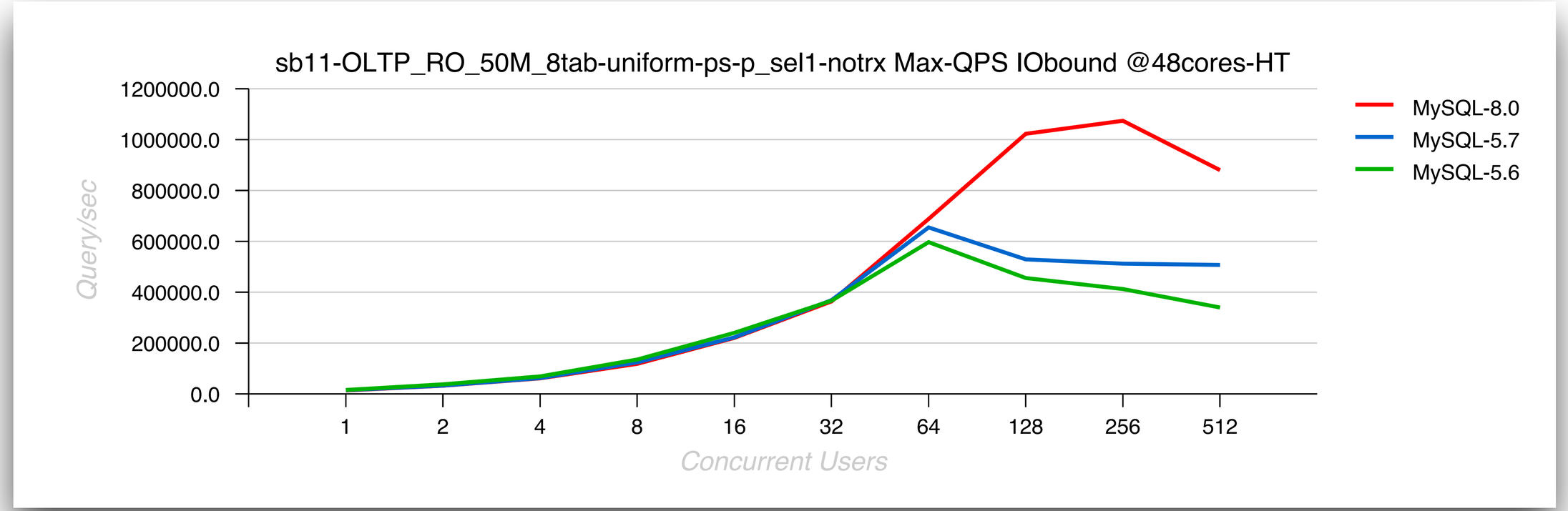
- Intel Optane drive :
  - IO read latency : 0,01ms (!!!)
  - 1 single process doing 16KB IO reads : ~65K reads/sec, 1000 MB/sec
  - however, the max throughput : 2000 MB/sec only (fix in progress by Intel)

- with x2 drives :
  - over 4000 MB/sec throughput
    - 16K page : ~260K IO reads/s
    - 8K page : over 500K IO reads/s
    - 4K page : **over 1M IO reads/s**

  - can MySQL get a profit of such an IO power ?..

IO-bound Point-Selects @MySQL 8.0 (Apr.2018)

- IO-bound Sysbench OLTP_RO Point-Selects
  - 50M x 8-tables, 48cores-HT, x2 Optane drives
  - NOTE : storage saturated & 100% CPU (new face of IO-bound ? ;-))
  - over **1M IO-bound QPS** with MySQL 8.0 !!!



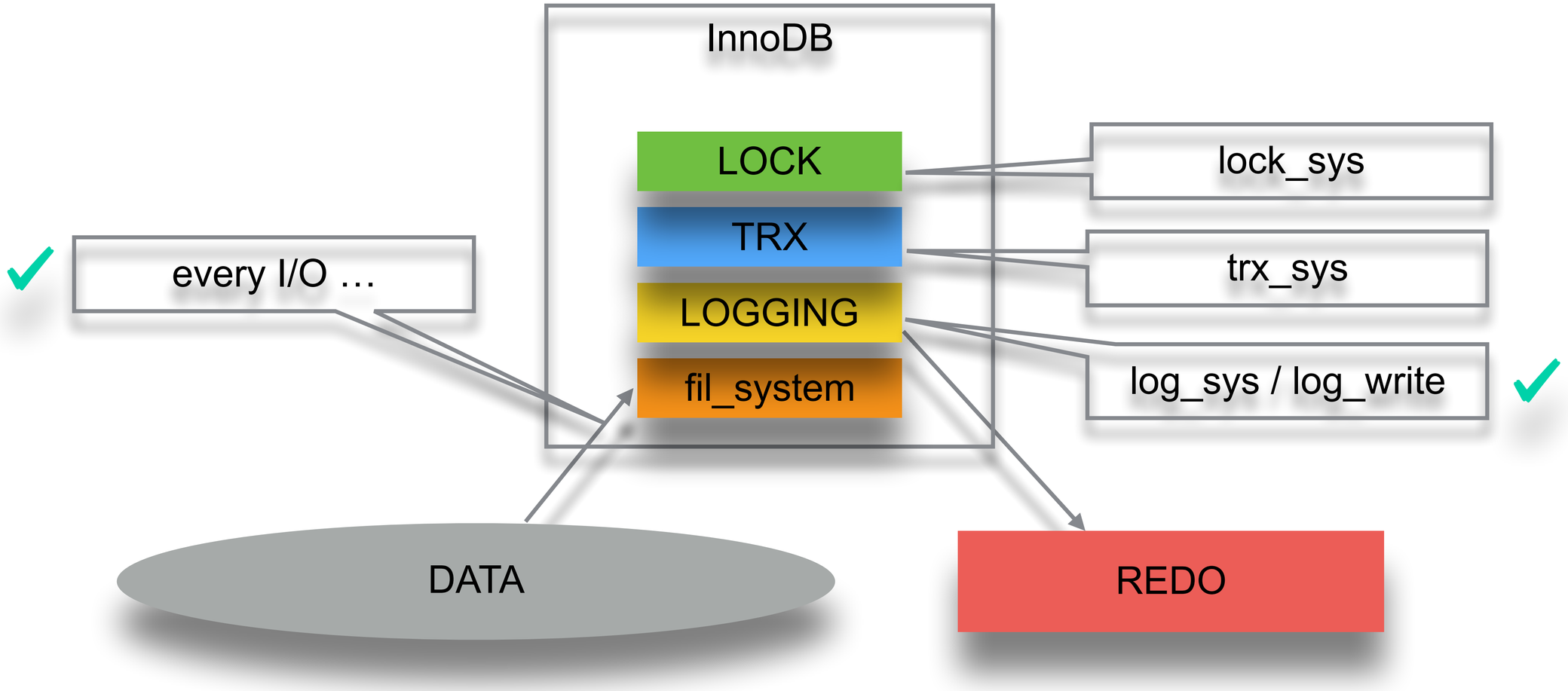
Pending Scalability Issues after MySQL 5.7 GA..

- RO :
  - Block Locks <= workaround : ProxySQL Query Cache
  - Lookups via Sec.IDX <= possible workaround : use PK, AHI
  - UTF8 <= **use 8.0** ;-)
  - Global lock on every IO read <= **use 8.0** ;-)
- RW :
  - REDO log related bottlenecks..
  - Double Write..
  - TRX management contentions..
  - LOCK management..
  - RR / RC isolation..
  - UPDATE Performance..
  - INSERT Performance..
  - Purge lagging..

Pending Scalability Issues after MySQL 5.7 GA..

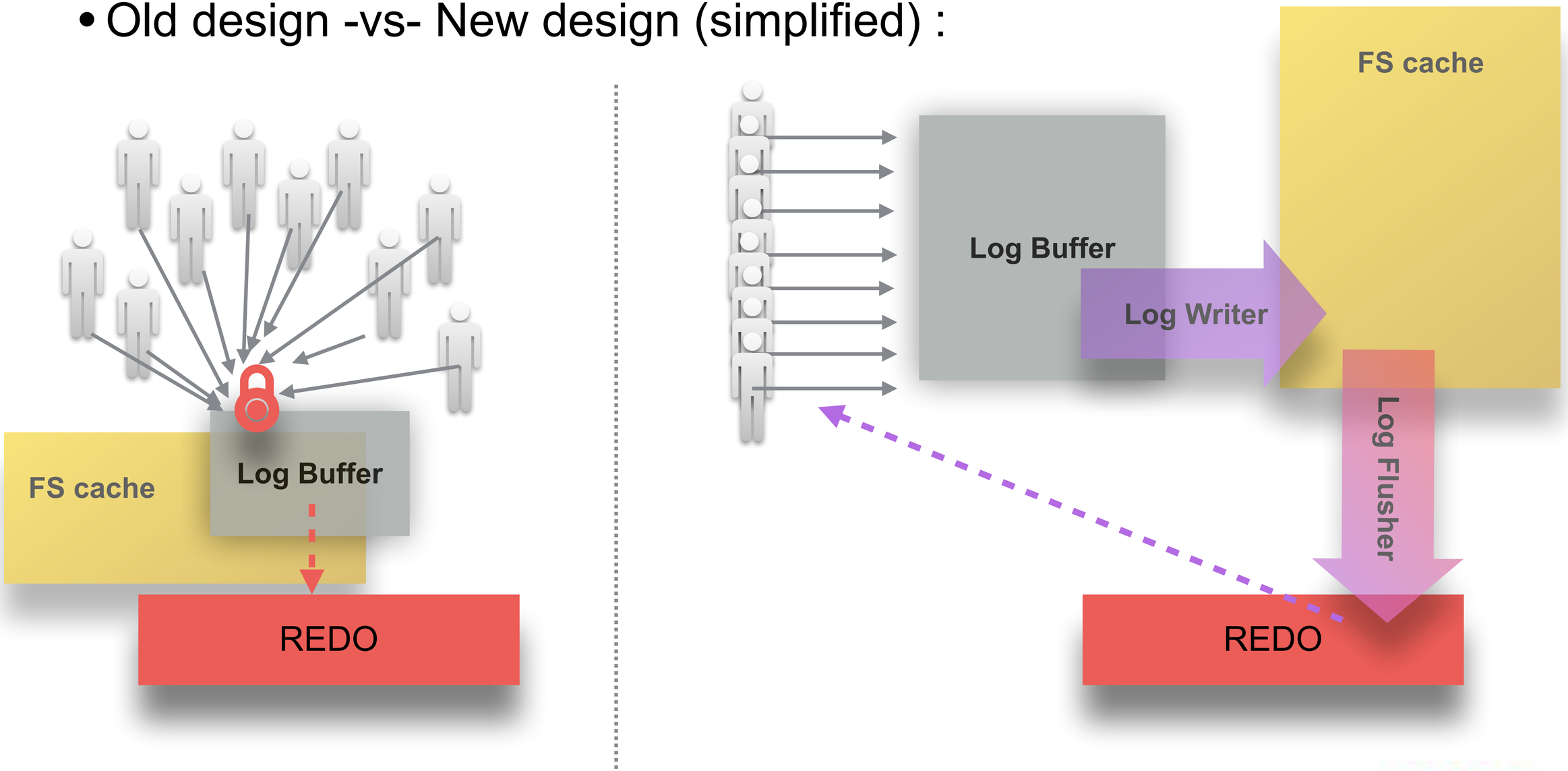
- RO :
  - Block Locks <= workaround : ProxySQL Query Cache
  - Lookups via Sec.IDX <= possible workaround : use PK, AHI
  - UTF8 <= **use 8.0** ;-)
  - Global lock on every IO read <= **use 8.0** ;-)
- RW :
  - REDO log related bottlenecks.. <= **new REDO since 8.0 !!**
  - Double Write.. <= **coming soon with 8.0 update (!!)**
  - TRX management contentions.. <= work-in-progress, prototyped..
  - LOCK management.. <= work-in-progress, prototyped..
  - RR / RC isolation.. <= work-in-progress, prototyped..
  - UPDATE Performance.. <= **use 8.0, and stay tuned ;-)**
  - INSERT Performance.. <= possible workaround : use partitions
  - Purge lagging.. <= not yet solved, but you can truncate UNDO

MySQL 8.0 : New Design for InnoDB Fundamentals..



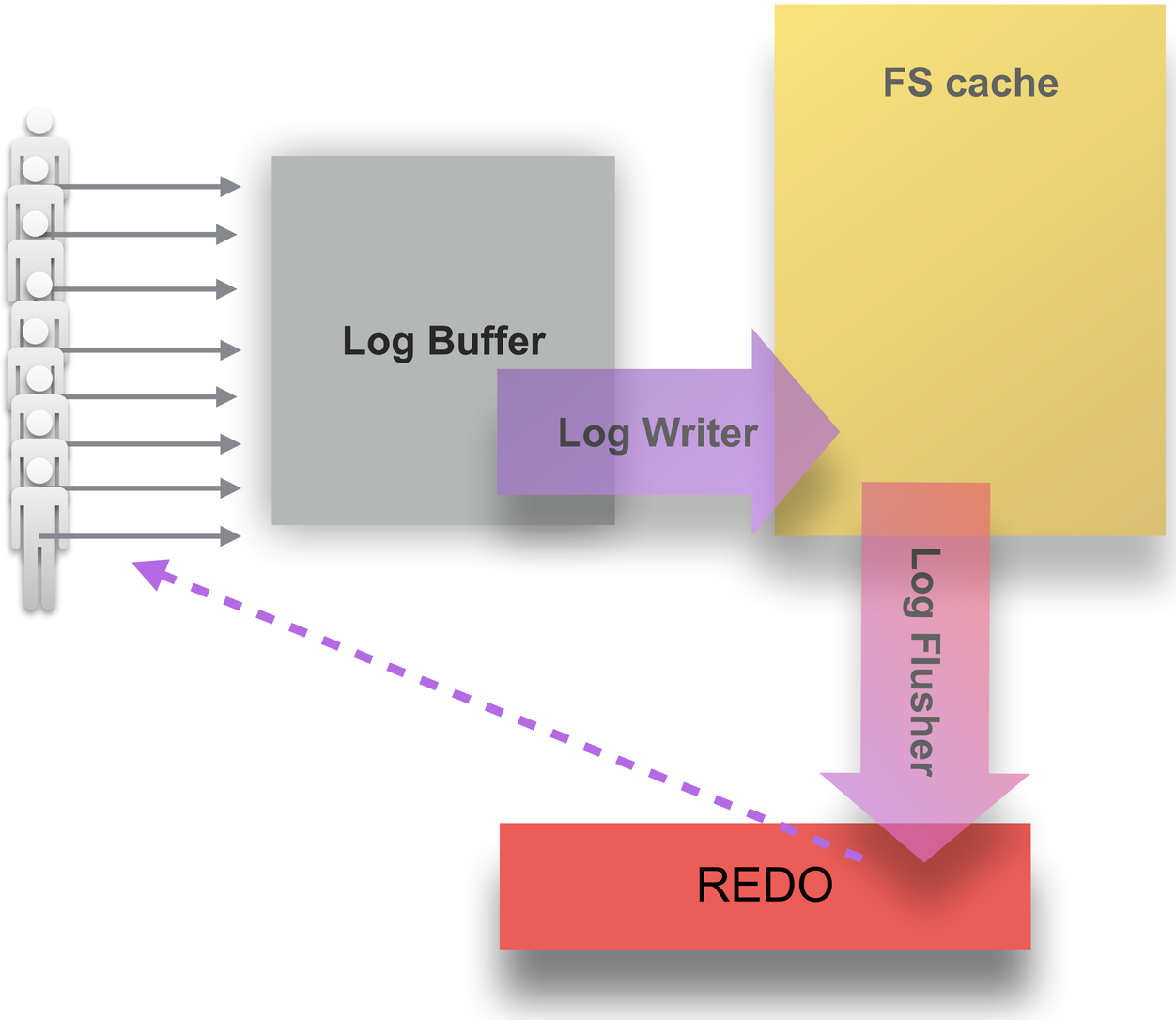
MySQL 8.0 : Re-Designed REDO

- Old design -vs- New design (simplified) :



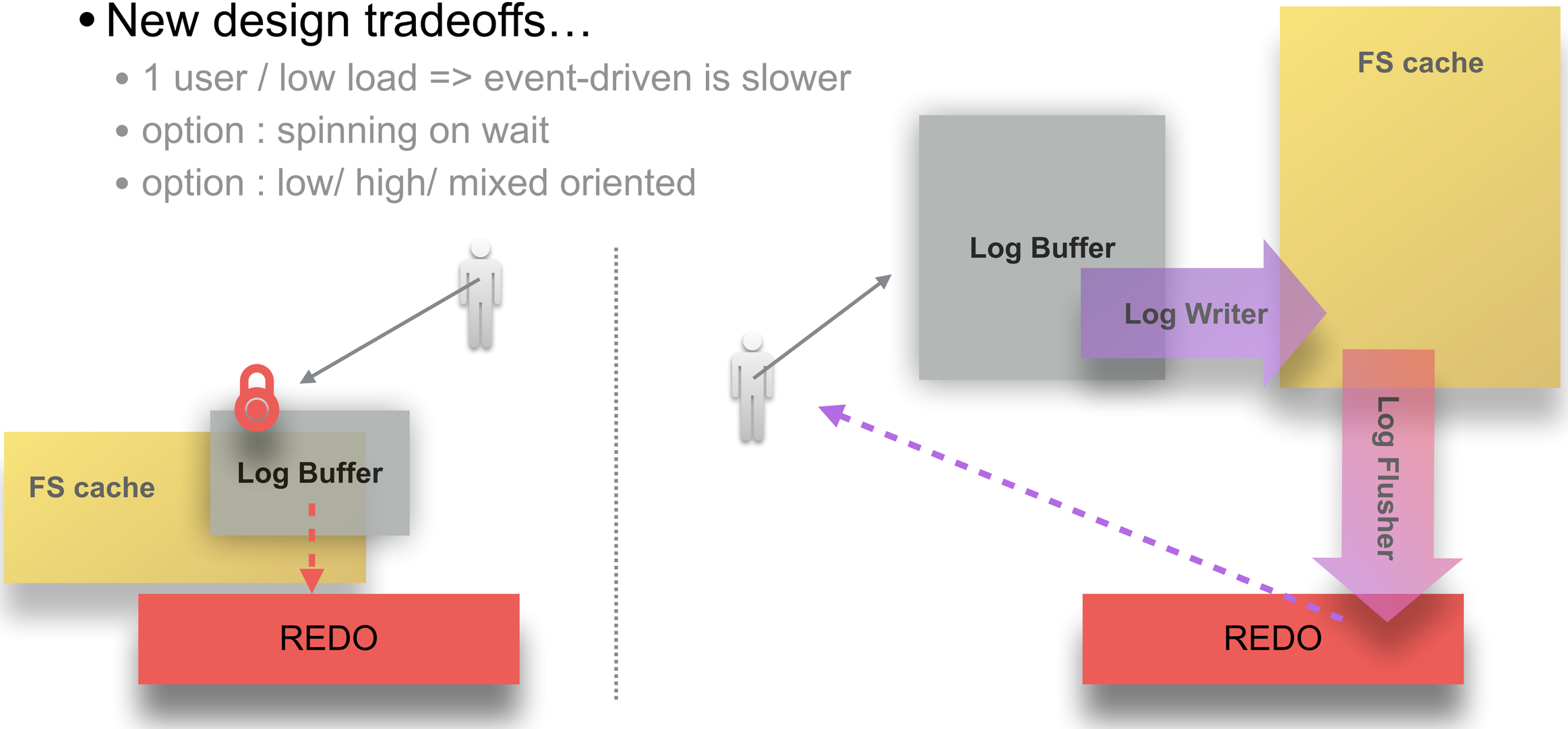
MySQL 8.0 : Re-Designed REDO

- New REDO design :
  - users are not fighting anymore !
  - self-driven processing..
  - self-driven by fsync() capacity
- Instrumented !
  - spins / waits
  - writer / flusher rates
  - max / avg flush times
  - etc..
- Configuration :
  - **mostly all dynamic !!!**
  - so you can play with it on-line ;-))



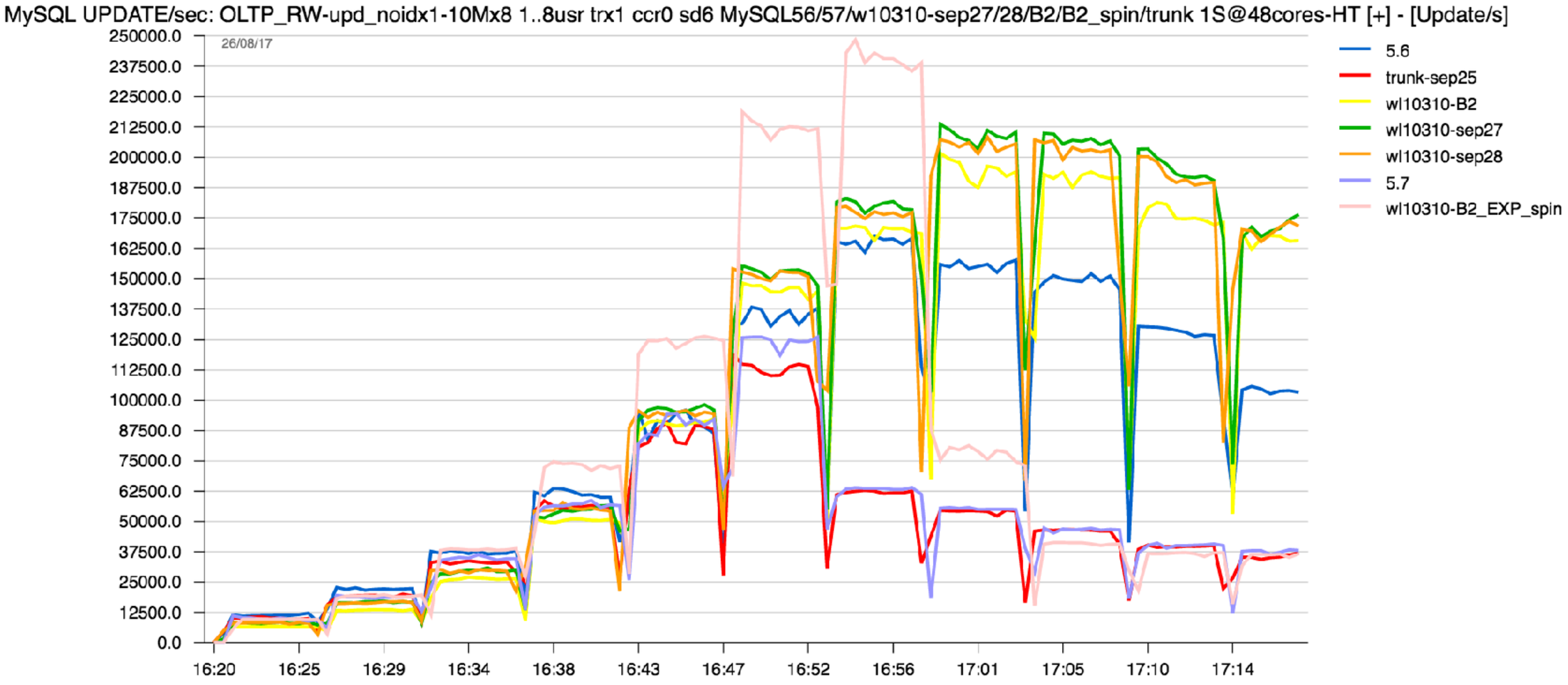
MySQL 8.0 : Re-Designed REDO

- New design tradeoffs…
  - 1 user / low load => event-driven is slower
  - option : spinning on wait
  - option : low/ high/ mixed oriented



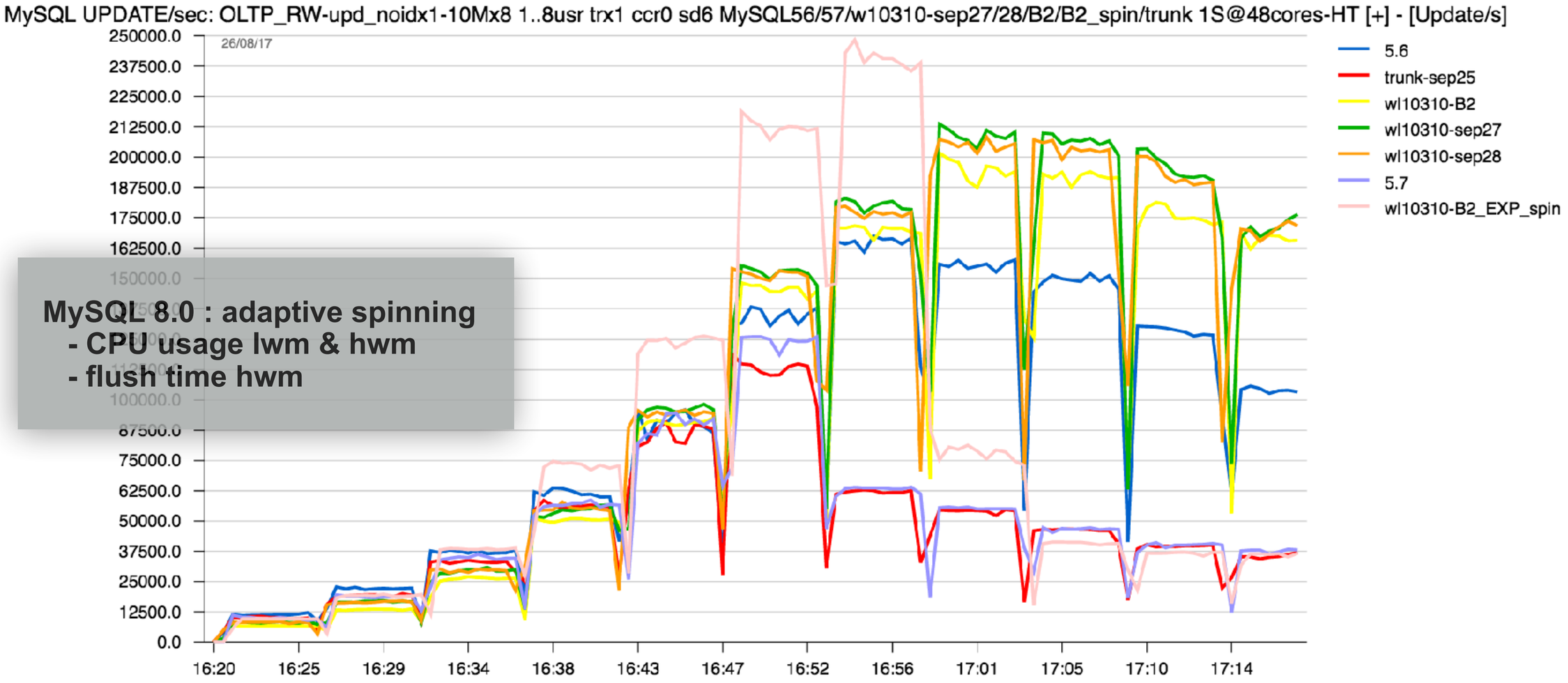
MySQL 8.0 : Re-Designed REDO

- New design tradeoffs…



MySQL 8.0 : Re-Designed REDO

- New design tradeoffs…

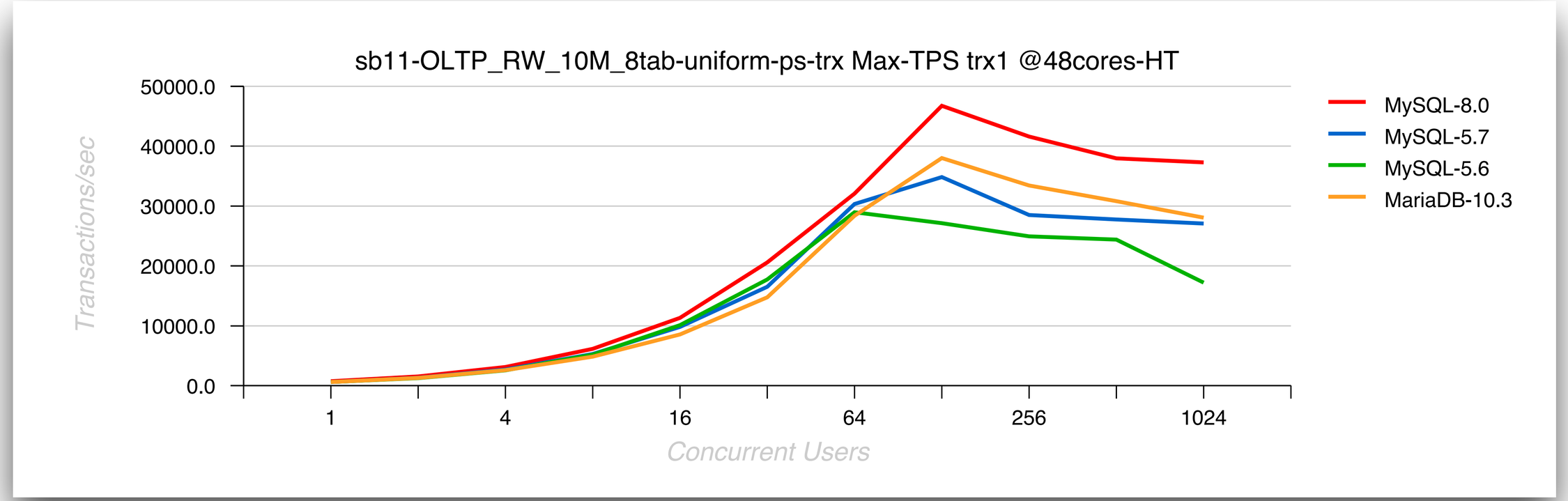


HW & my.conf

- for all the following tests :
  - Server : 2CPU Sockets (2S) 48cores-HT Skylake
  - Storage : x2 Optane NVMe, MDM-raid0, EXT4
  - OS : OL7.4 UEK4
  - Engines : mysql5.6, mysql5.7, mysql8.0, mariadb10.3.5
  - Concurrent Users : **1**, **2**, **4**, .. **1024**
  - charset : **latin1** (!! ;-))
  - **trx_commit=1**
  - REDO = 32GB
  - O_DIRECT
  - DBLWR=off, binlog=off, SSL=off, PFS=off, UNDO auto-truncate=off..
- NOTE :
  - we’re NOT running after “good numbers” !! ;-))
  - we just want to avoid “known overheads” to hide real problems..

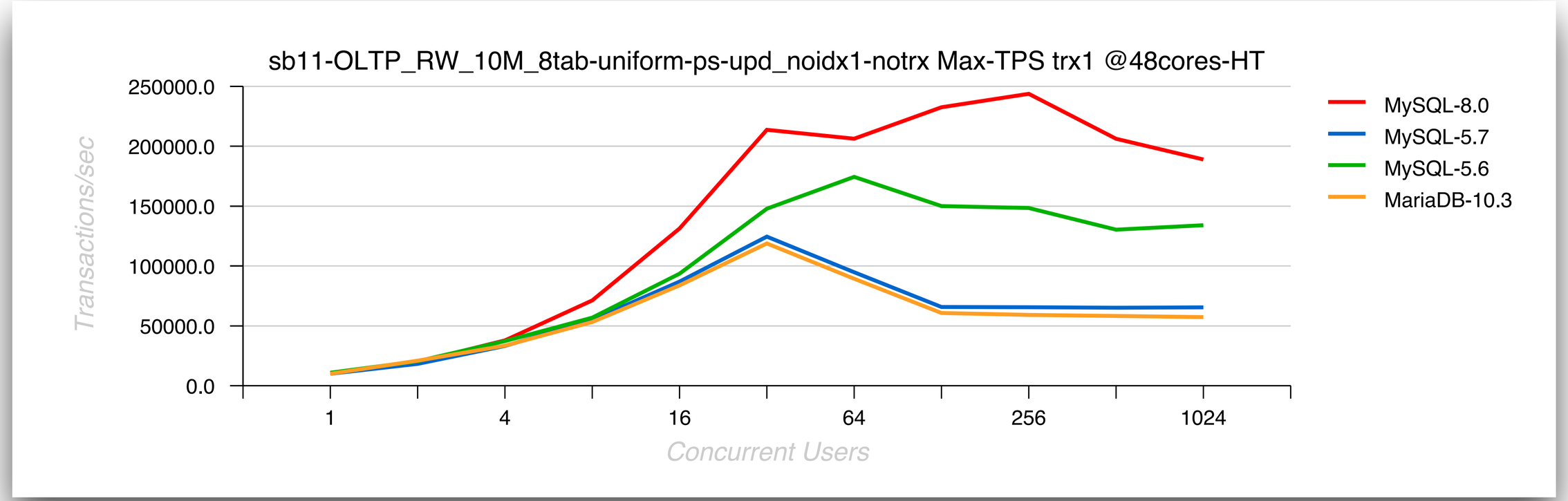
OLTP_RW latin1 @MySQL 8.0 (Apr.2018)

- **45K (!!) TPS** Sysbench OLTP_RW 10Mx8tab, trx_commit=1, 2S
  - **30**% gain vs MySQL 5.7
  - **50**% gain vs MySQL 5.6



Updates-NoKEY latin1 @MySQL 8.0 (Apr.2018)

- **250K (!!) QPS** Sysbench Updates-nokey 10Mx8tab, trx_commit=1, 2S
  - **100**% gain vs MySQL 5.7
  - **50**% gain vs MySQL 5.6 (and yes, 5.7 is bad here.. => fixed !! ;-))
  - and clearly seen MariaDB's adoption of InnoDB 5.7..

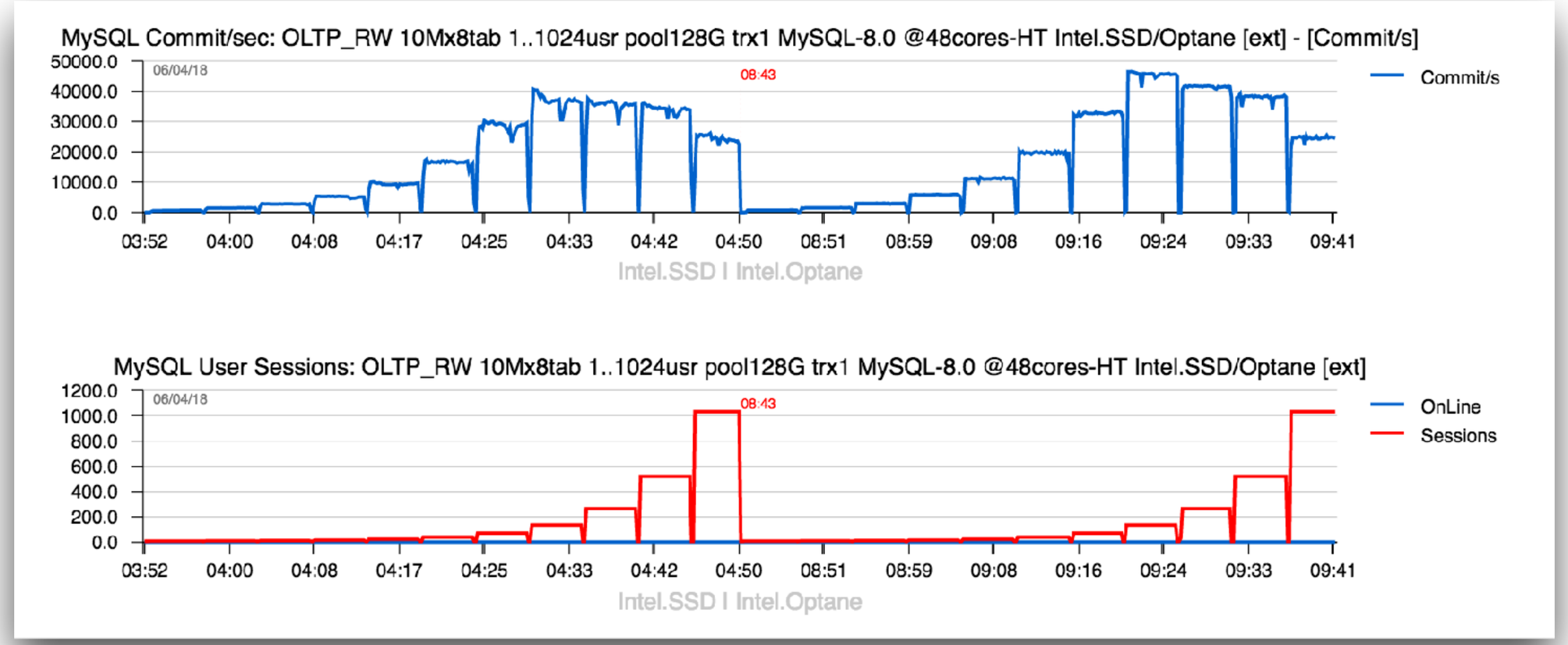


Storage Impact : “good” -vs- “very good” Flash

- So, you have :
  - “good” Intel SSD
  - “very good” Intel Optane
- Workload :
  - OLTP_RW
  - all data are cached in Buffer Pool (BP=128G)
- Question :
  - how different will be TPS if Optane is used instead of “good SSD” ?..

In-Memory OLTP_RW @MySQL 8.0 : Intel SSD -vs- Optane

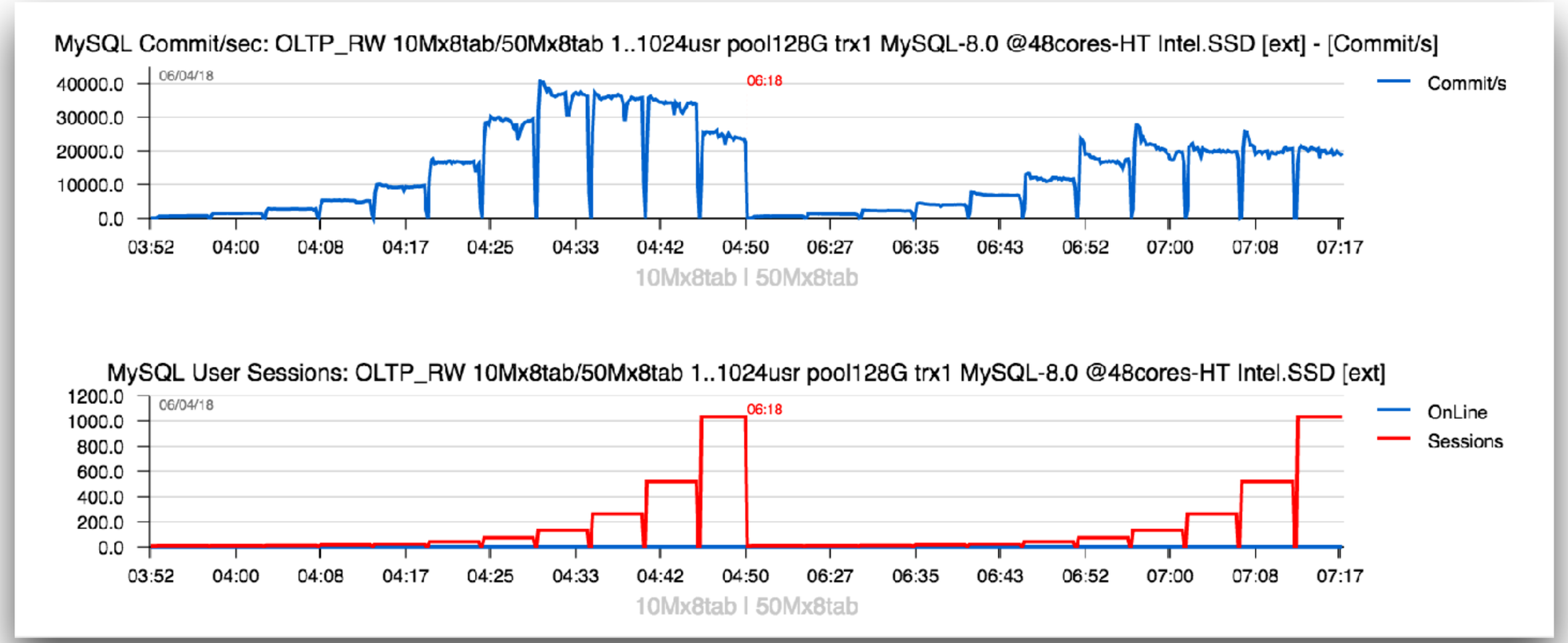
- OLTP_RW : 10Mx8tab (~20GB data):
  - SSD : Max 37K TPS
  - Optane : Max 45K TPS (20% gain — not that much, right ?)



ORACLE

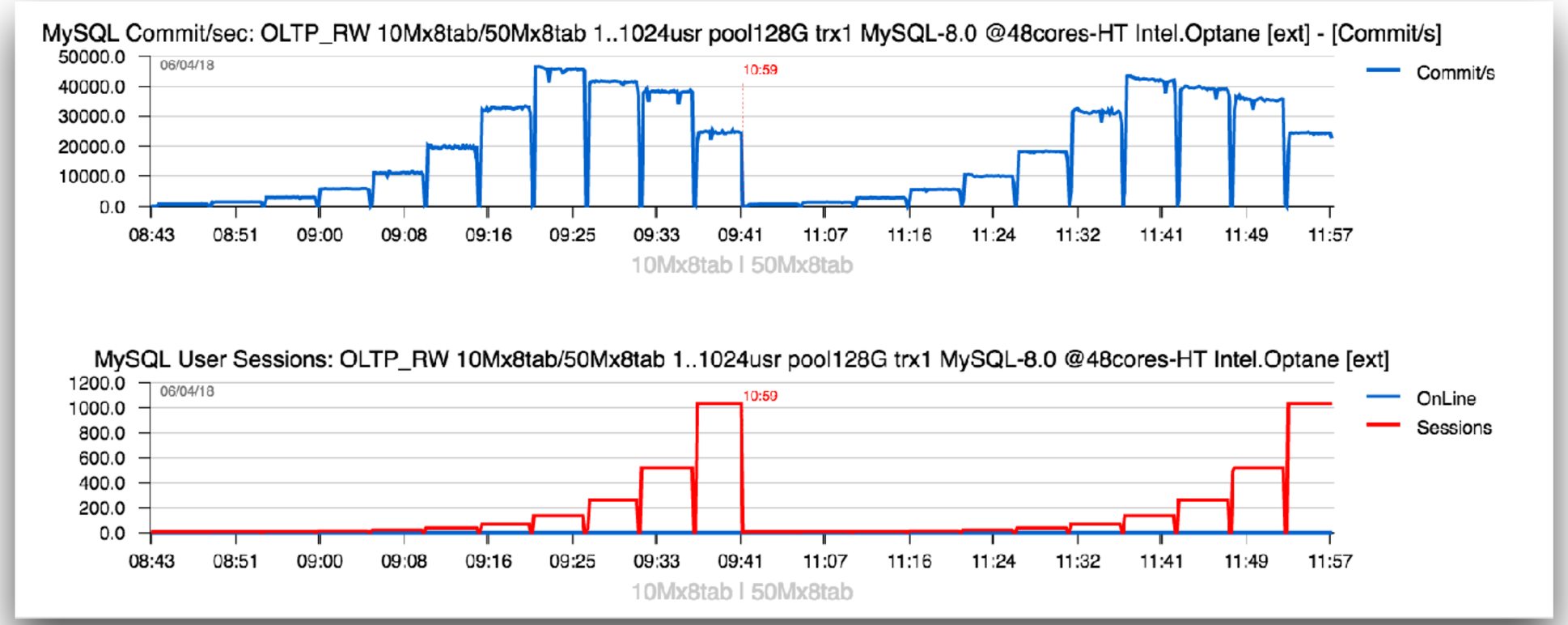
In-Memory OLTP_RW @MySQL 8.0 : Intel SSD

- OLTP_RW : 10Mx8tab (~20GB data) -vs- 50Mx8tab (100GB)
  - NOTE : data are still cached in BP !!!
  - Intel SSD : 37K TPS -vs- 20K TPS.. (near x2 TPS loss..— oh, shit ;-))



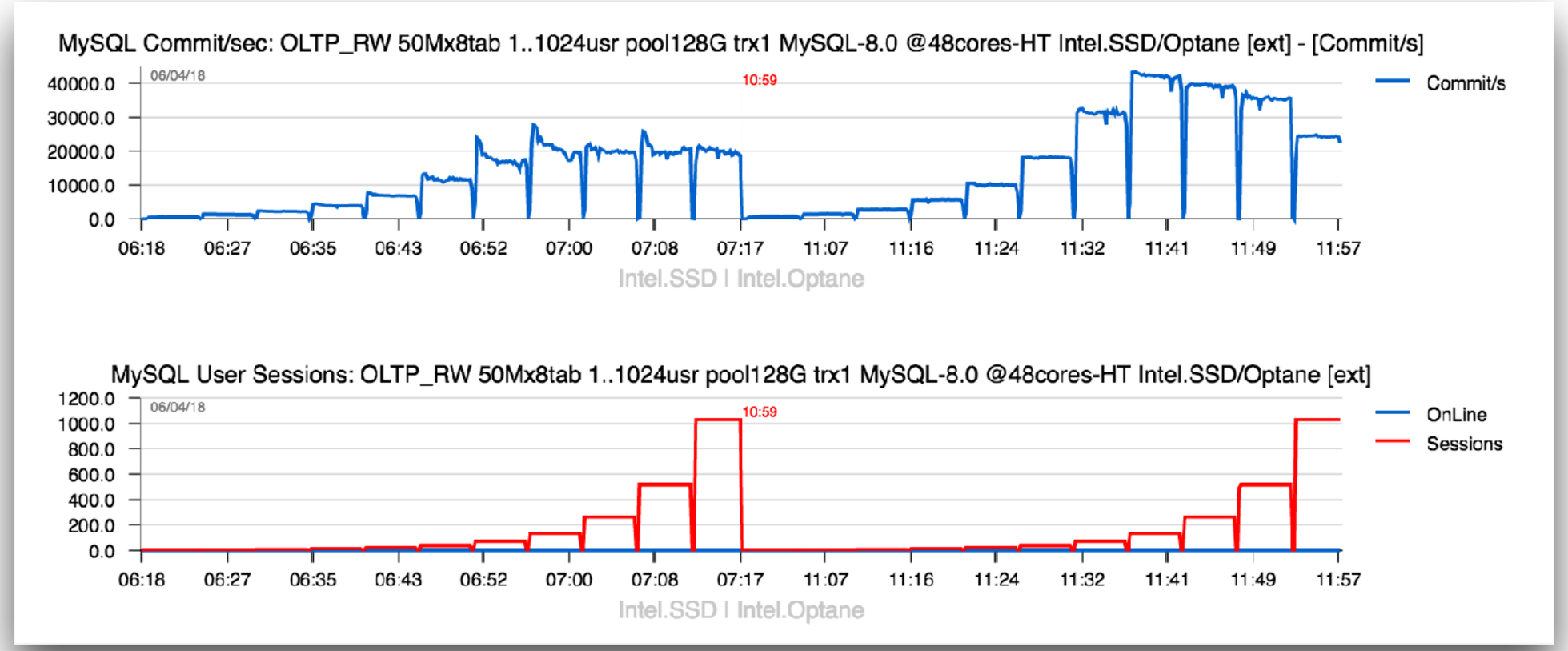
In-Memory OLTP_RW @MySQL 8.0 : Optane

- OLTP_RW : 10Mx8tab (~20GB data) -vs- 50Mx8tab (100GB)
  - NOTE : data are still cached in BP !!!
  - Optane : 45K TPS -vs- 41K TPS.. (9% TPS loss..— aha.. ;-))



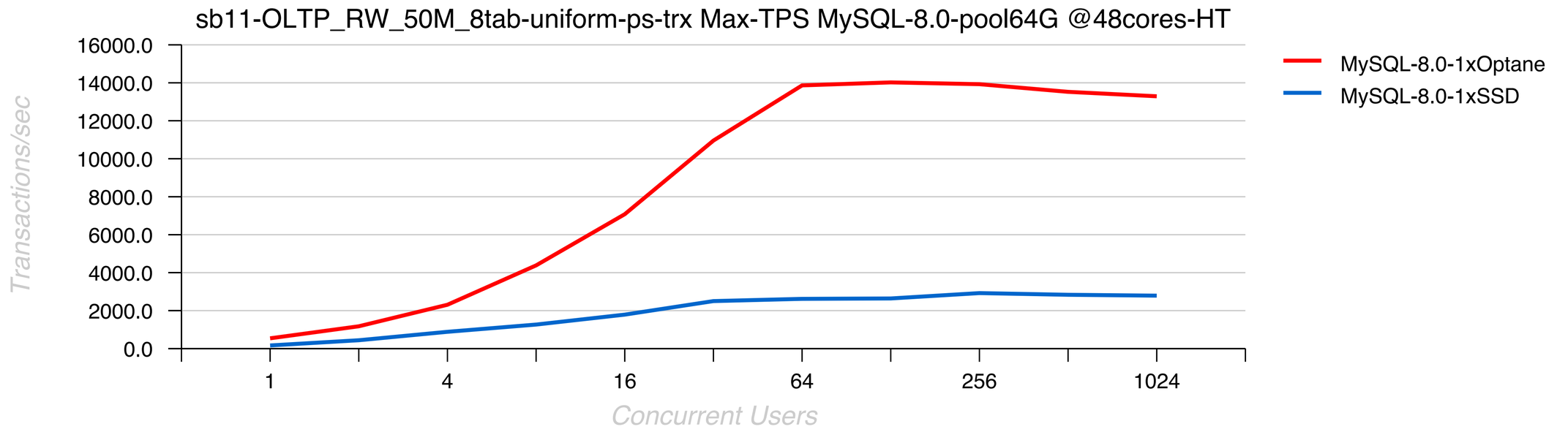
In-Memory OLTP_RW @MySQL 8.0 : Intel SSD -vs- Optane

- OLTP_RW : 50Mx8tab (100GB)
  - NOTE : data are still cached in BP !!!
  - SSD : 20K TPS -vs- Optane : 41K TPS (over x2 times diff..— hehe.. ;-))



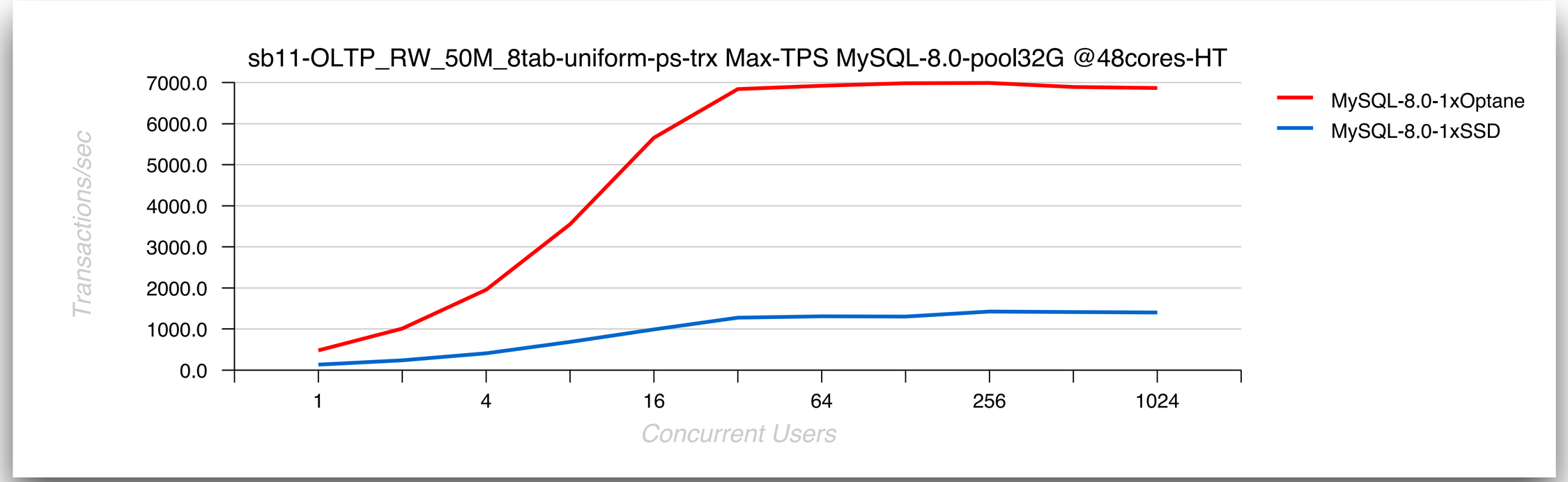
IO-bound OLTP_RW @MySQL 8.0 : Intel SSD -vs- Optane

- OLTP_RW : 50Mx8tab (100GB)
  - BP=64G — only 1/2 of data can be cached in BP
  - SSD : 2.8K TPS -vs- Optane : 14K TPS (x5 times diff..— hoho.. ;-))



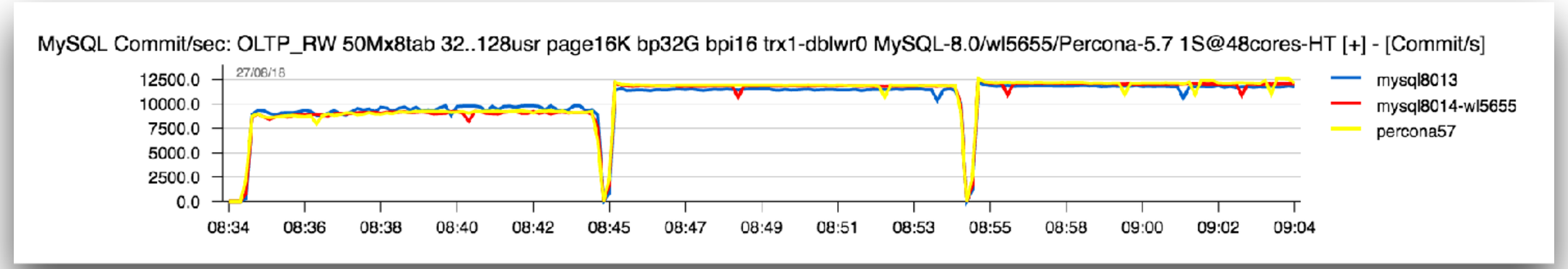
IO-bound OLTP_RW @MySQL 8.0 : Intel SSD -vs- Optane

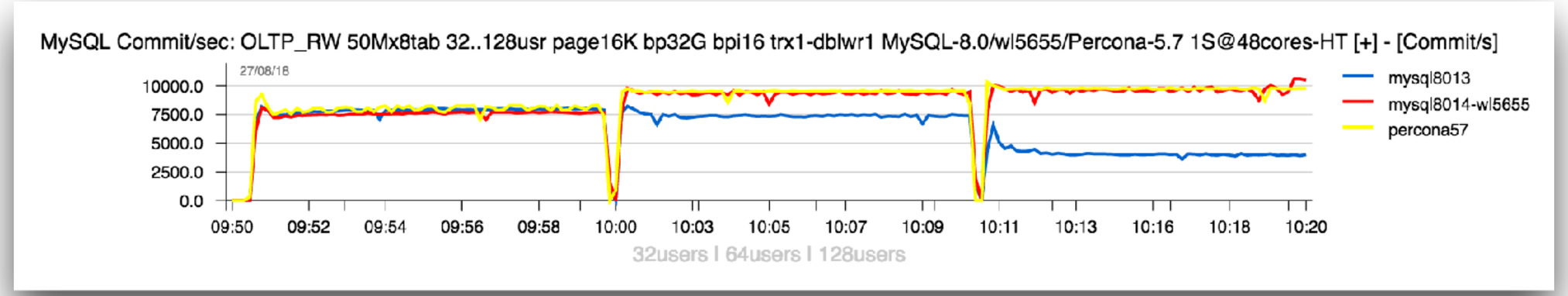
- OLTP_RW : 50Mx8tab (100GB)
  - BP=32G — only 1/3 of data can be cached in BP
  - SSD : 1.4K TPS -vs- Optane : 7K TPS (still x5 times diff..— still hoho.. ;-))
  - NOTE : we're hitting the MB/sec Optane limit (with x2 Optane you have x2 TPS more)



Double Write Impact

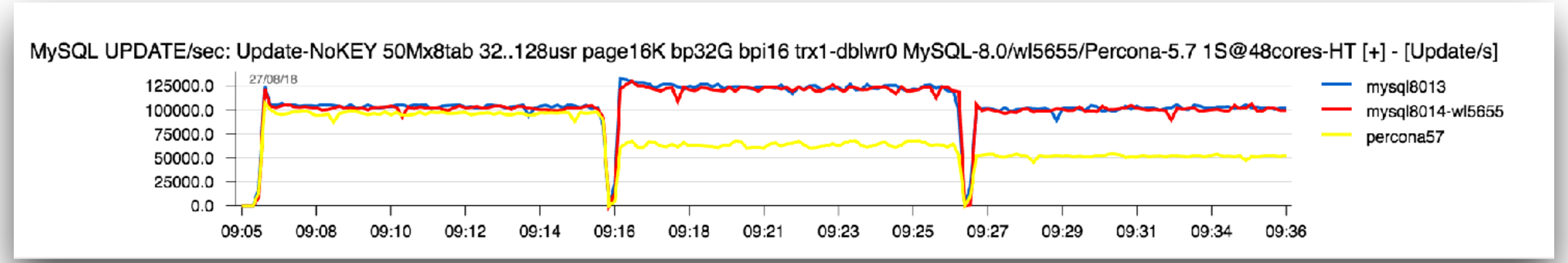
- IO-bound OLTP_RW 50Mx8tab
  - BP=32G
  - double write = OFF / ON

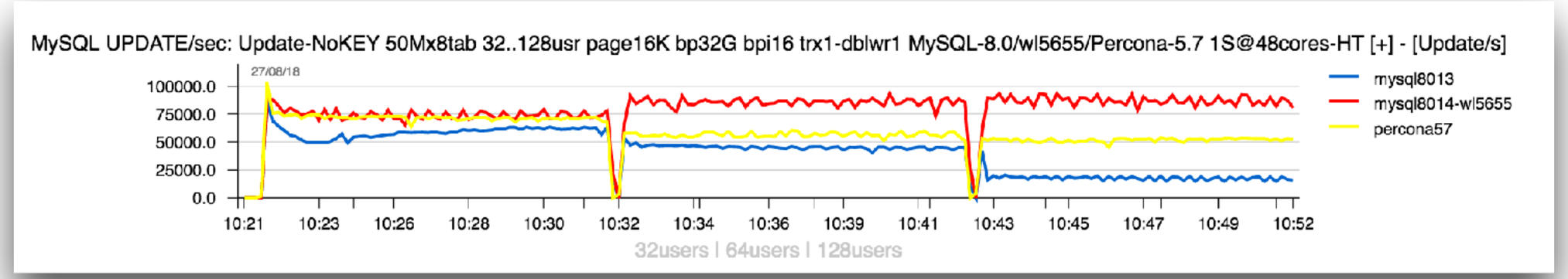




Double Write Impact

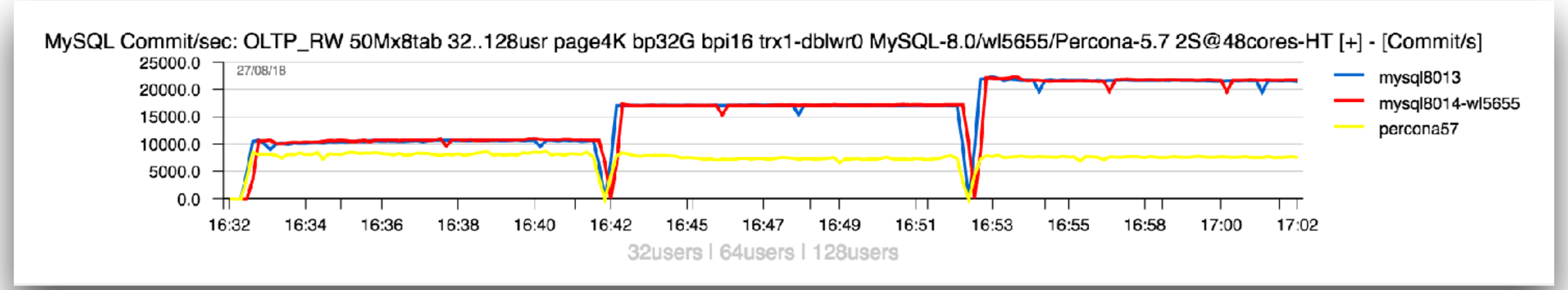
- IO-bound Update_NoKEY 50Mx8tab
  - BP=32G
  - double write = OFF / ON

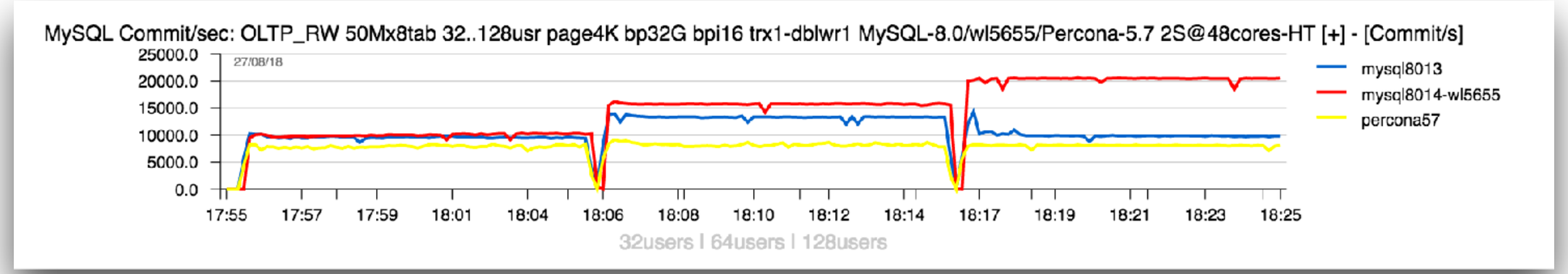




Double Write Impact

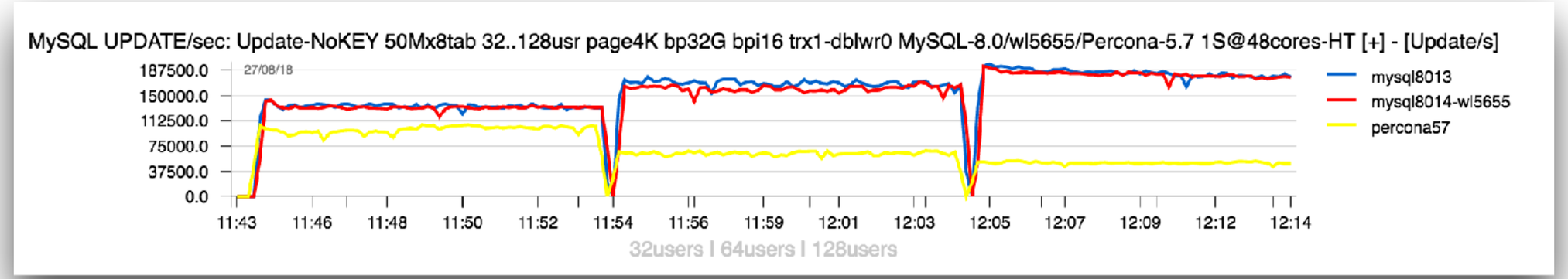
- IO-bound OLTP_RW 50Mx8tab
  - BP=32G, page size = 4K
  - double write = OFF / ON

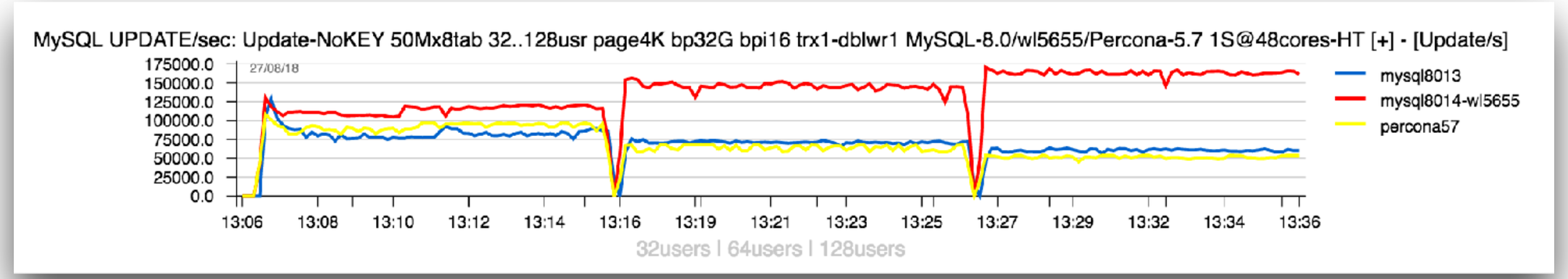




Double Write Impact

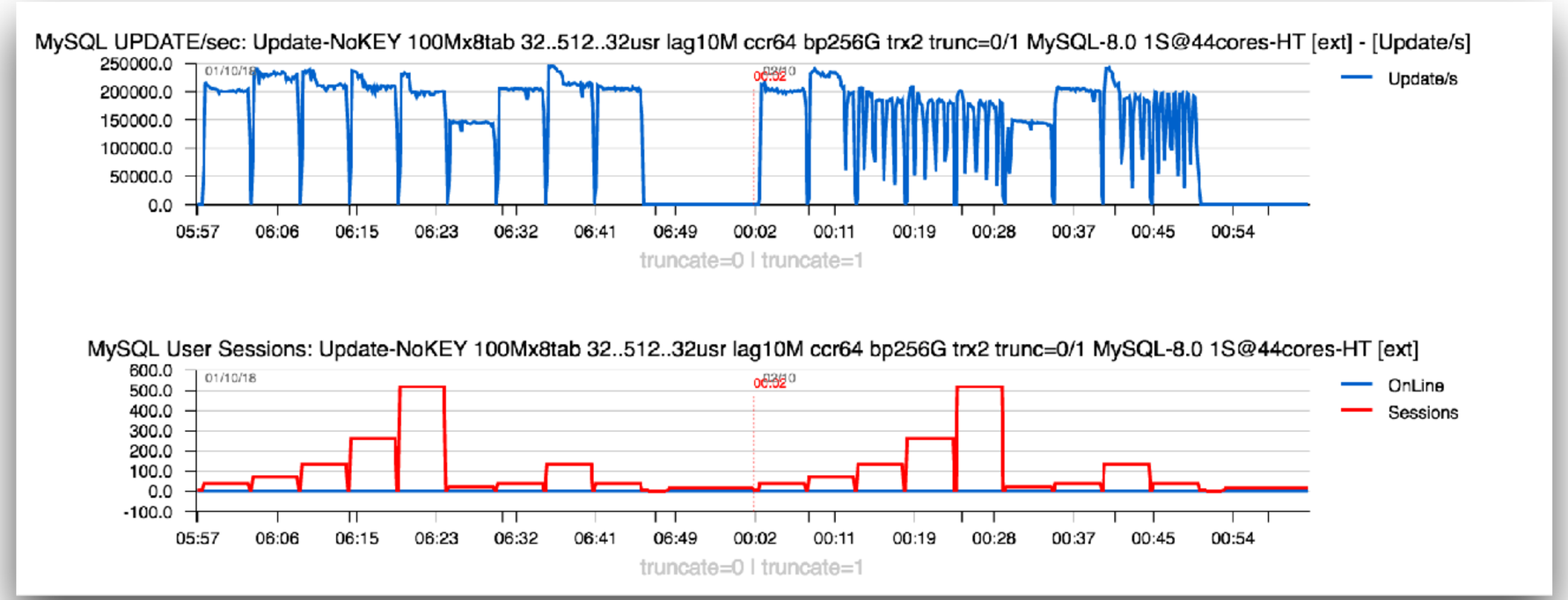
- IO-bound Update_NoKEY 50Mx8tab
  - BP=32G, page size = 4K
  - double write = OFF / ON





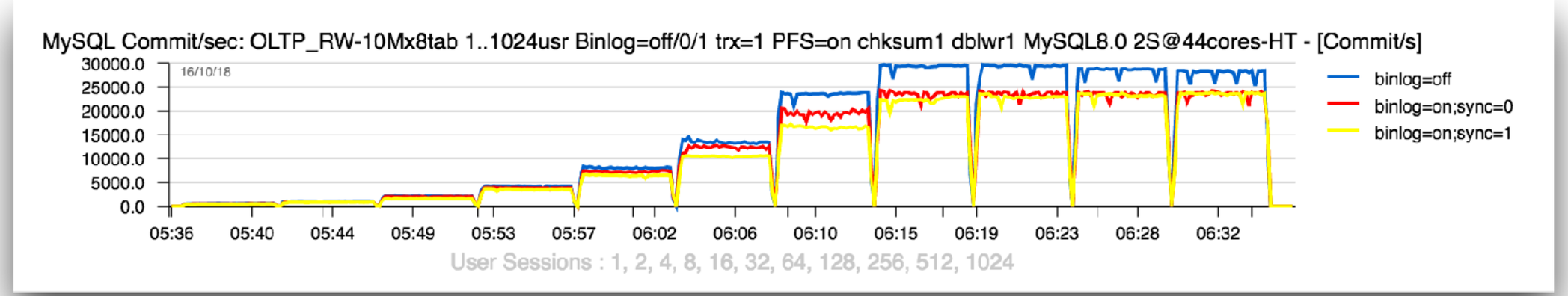
UNDO Auto-Truncate Impact

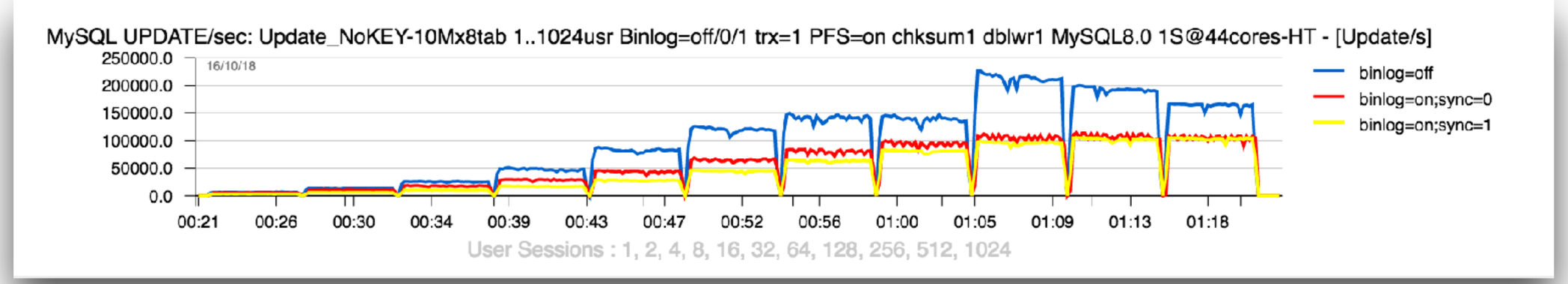
- Update_NoKEY 50Mx8tab, BP=256G
  - yes it sucks, yes we have to do something..
  - workaround :
    - innodb_undo_log_truncate=off
    - and truncate manually when no user activity



Binlog Impact

- OLTP_RW / Update_NoKEY 10Mx8tab, BP=32G
  - yes it sucks, yes we MUST do something..
  - workaround : … 🤔 …





MySQL 8.0 Writes Scalability

- IMPORTANT :
  - MySQL 8.0 overall WRITE performance is way better comparing to all we have before !
  - but : we're still NOT scaling !..
- Going from 1S => 2S (CPU Sockets) :
  - OLTP_RW : somewhat 50% better TPS only, and it's due RO scaling..
  - Update-NoKEY : just worse TPS..
- Why ?
  - 1) next-level bottlenecks (TRX / LOCK Management)
  - 2) + something else (yet to discover)..
  - so, still a lot of work ahead ;-))

New: MySQL Resource Groups

- What :
  - starting codebase for our future Resource Management solutions
  - flexible and proper thread / query isolation
  - dynamic, integrated, fun ! ;-))
- Why :
  - protect background threads, provide them optimal conditions for processing
  - run batches on low priority, OLTP on higher (and opposite on night)
  - isolate DDL orders from other activity
  - allow to move long running queries to low priority / isolate (live !! ;-))
  - apply particular execution conditions for any SQL query via Optimizer Hint
    - => Query Rewrite, ProxySQL, etc..
  - automatically assign RG to users / databases / workloads via ProxySQL
  - potential workaround for many CPU cache related issues
  - **huge opportunity to all kind of new tools !!!**

MySQL Resource Groups

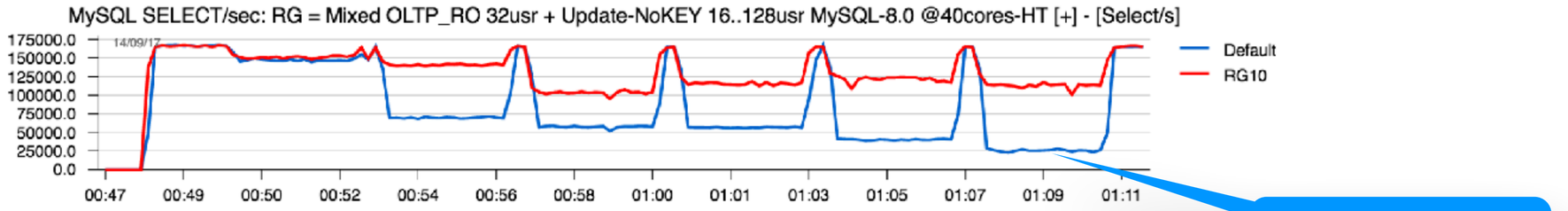
- Implementation Details :
  - currently : USER and SYSTEM groups
  - attributes : CPU (vcpu) affinity & thread priority
  - **thread priority** :
    - SYSTEM : [-19, 0] normal or **higher**
    - USER : [0, 20] normal or **lower**
- Admin :
  - permissions : none / can use / can use + admin
  - mysql> create RESOURCE GROUP RG10 type=user vcpu=0-9,40-49 thread_priority=0 ;
  - mysql> alter … ; drop … ; (also DISABLE / ENABLE / etc..)
- Using : only by name !
  - mysql> SET RESOURCE GROUP name ; (also for any THREAD ID)
  - SELECT /*+ RESOURCE_GROUP(name) */ … ; (query hint)

MySQL Resource Groups in Action

- Test case :
  - 40cores-HT 4S (Broadwell) server, OL7
  - 32 concurrent users are running SELECTs (Sysbench OLTP_RO)
  - other users are coming with UPDATEs (Sysbench Update-NoKEY)
    - 16 users, then 32, 64, 128, 256, 512
  - **Problem :** each workload is running well alone, but NOT together (yet)..
  - **Workaround :**
    - UPDATEs are not scaling and mixed with SELECTs creating yet more contentions
    - let's limit UPDATE queries to 10cores-HT only
    - mysql> create RESOURCE GROUP RG10 type=user vcpu=0-9,40-49 thread_priority=0 ;
    - and add a hint to UPDATE queries :
      UPDATE /*+ RESOURCE_GROUP(RG10) */ … ;

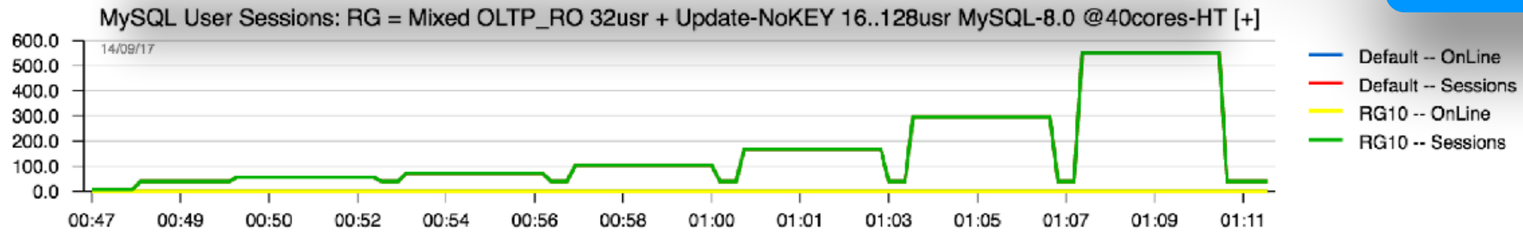
MySQL Resource Groups in Action

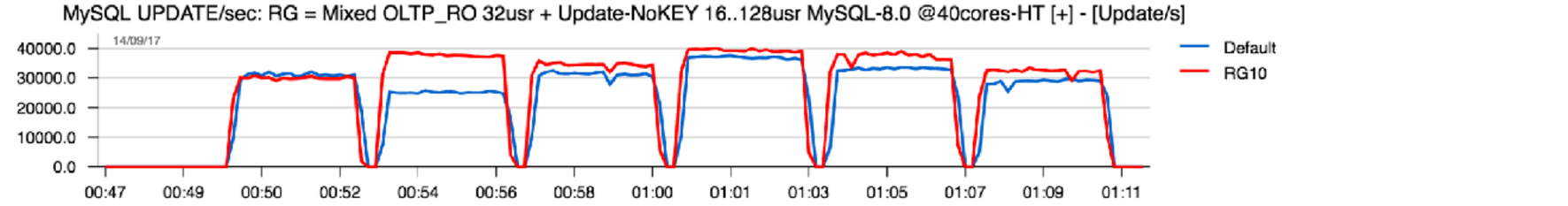
- Test case :
  - 32 concurrent users are running SELECTs
  - other users are coming with UPDATEs : 16 users, then 32, 64, 128, 256, 512..



none | 16usr | 32usr | 64usr | 128usr | 256usr | 512usr

Default

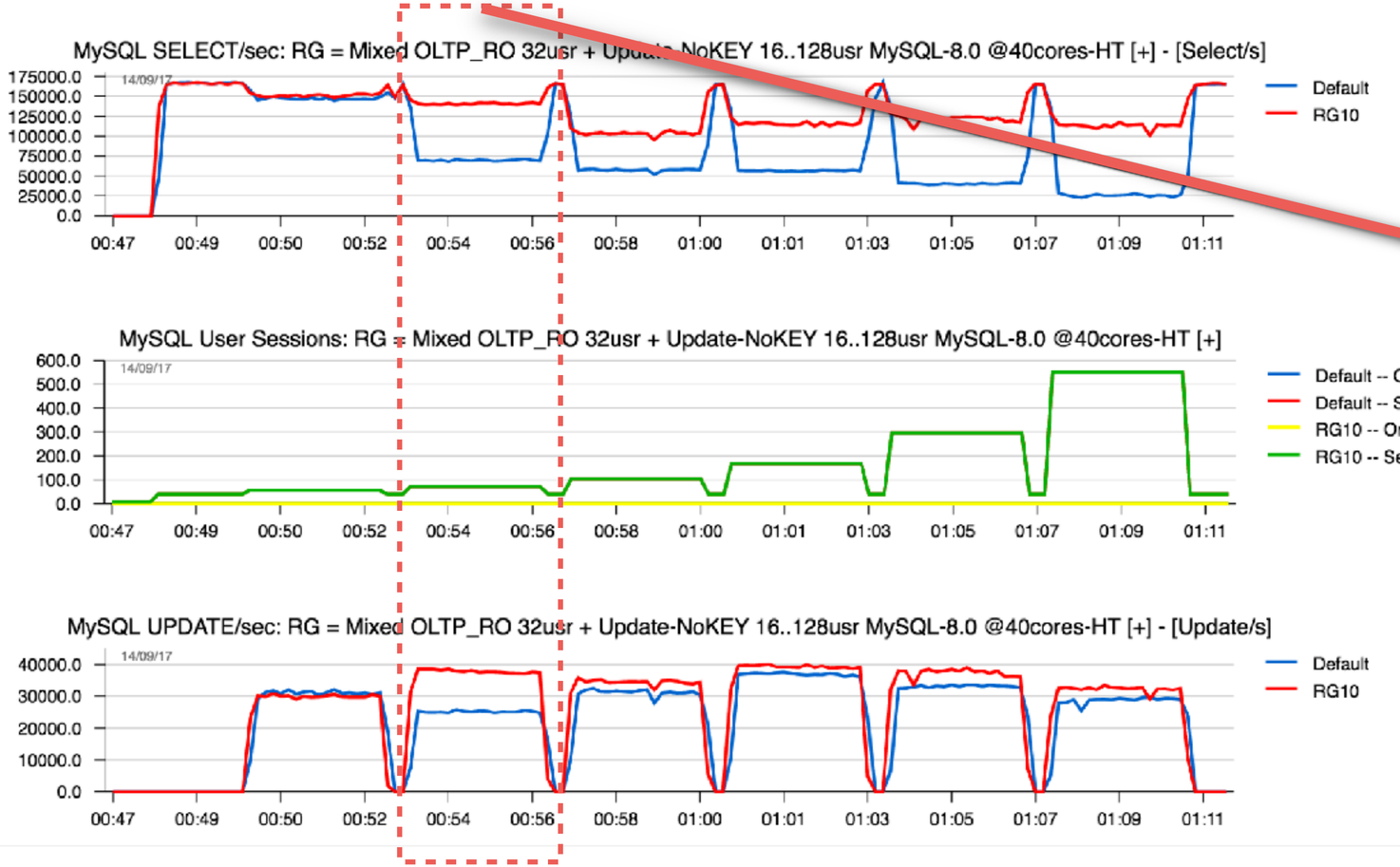


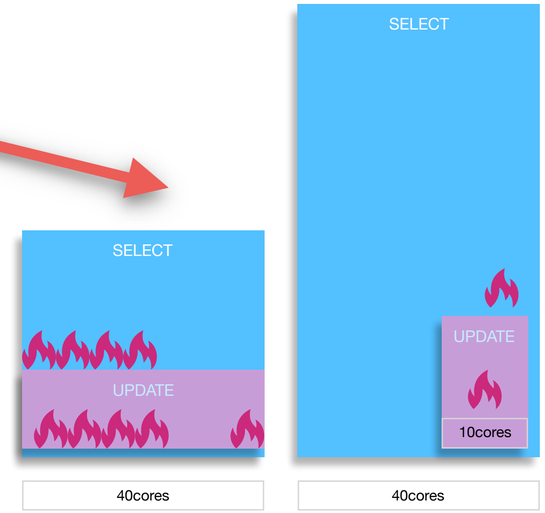


ORACLE

MySQL Resource Groups in Action

- Test case :
  - 32 concurrent users are running SELECTs
  - other users are coming with UPDATEs : 16 users, then 32, 64, 128, 256, 512..





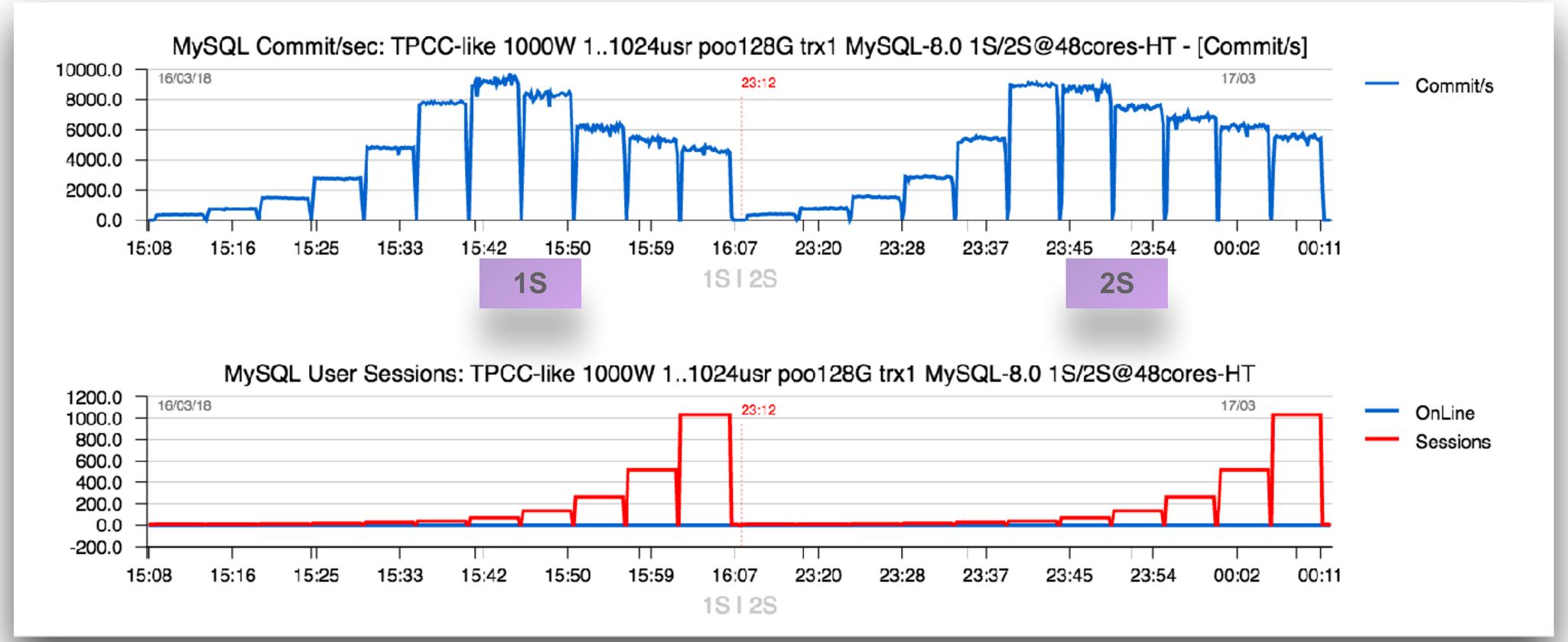
TPCC-like Workload

- Developed & Maintained by Percona
  - recently ported to Sysbench (**KUDOS Percona !!** ;-))
  - can be executed as x1 dataset
  - or xN datasets (similar to xN databases, except it's the same db)

- The following Test Scenarios :
  - Warehouses : 1000W or 10x100W (same ~100GB volume in total)
  - BP = 128GB (in-memory), = 32GB (IO-bound ?)

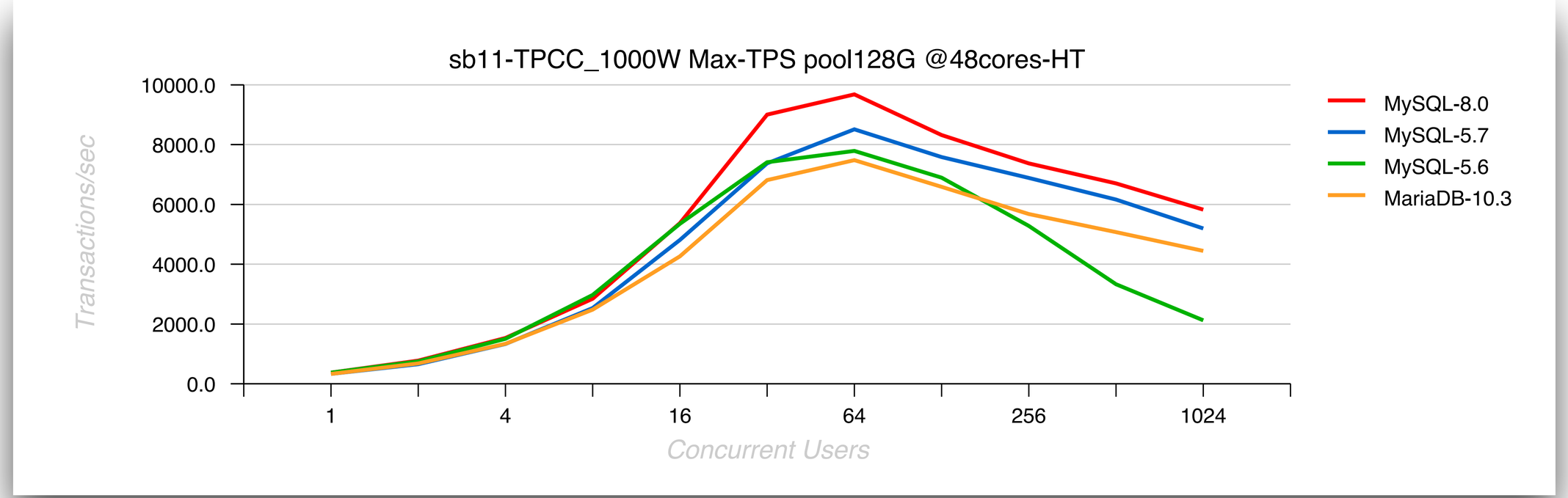
TPCC-like 1000W Workload

- MySQL 8.0 :
  - small or no gain between 1S -vs- 2S
  - scalability is totally blocked by “index RW-lock” contention..



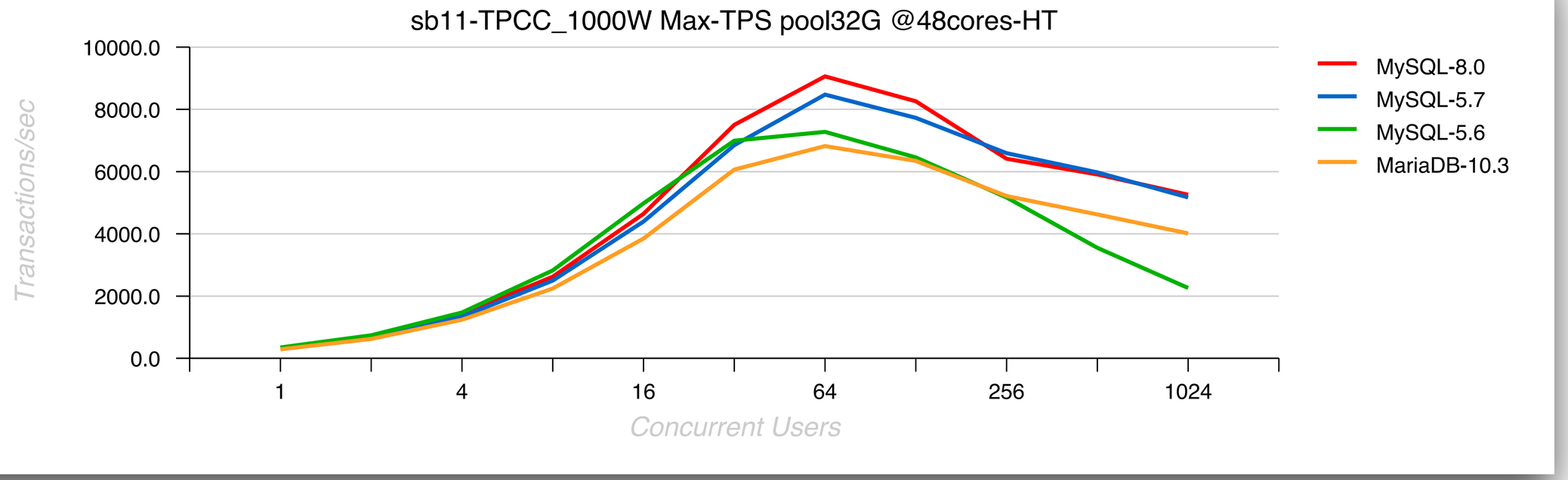
TPCC-like 1000W Workload

- BP = 128GB : in-memory
  - MySQL 8.0 is still doing better again
  - index RW-lock is dominating for all..



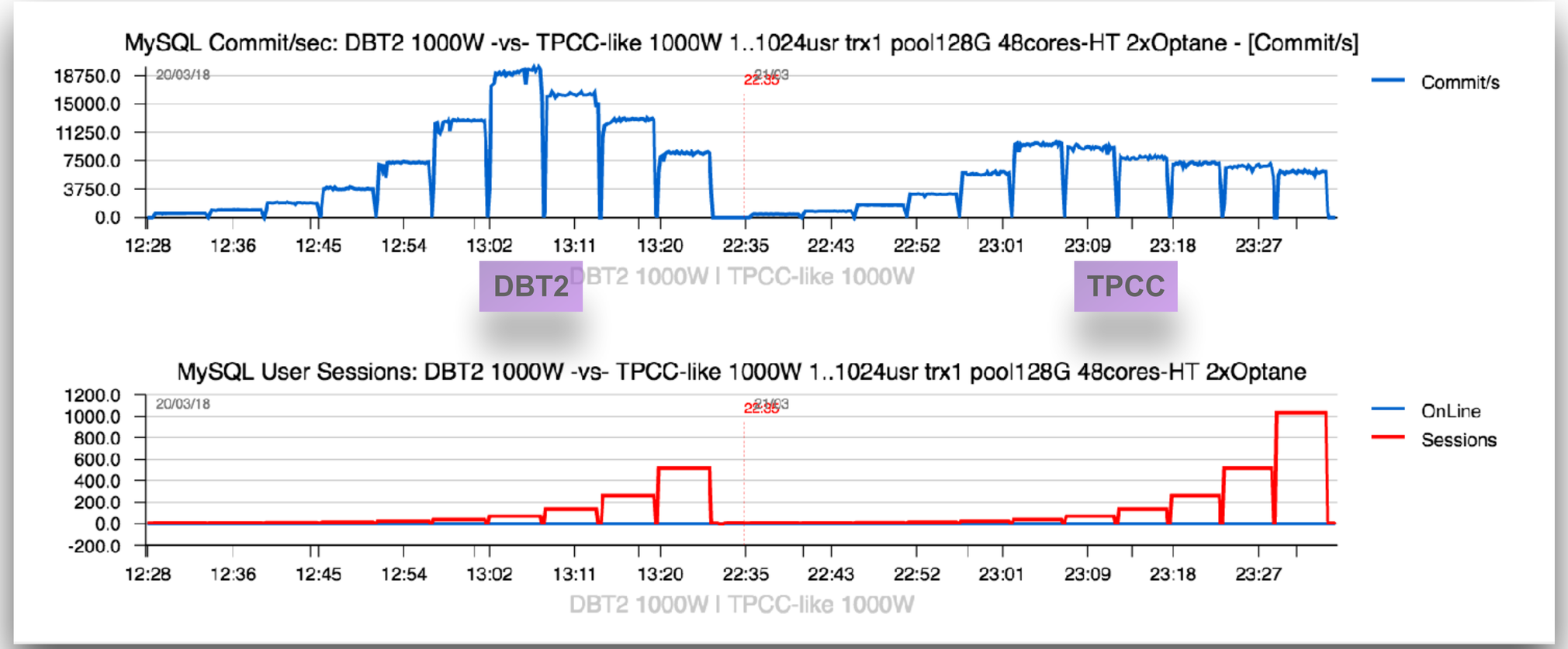
TPCC-like 1000W Workload

- BP = 32GB : IO-bound
  - MySQL 8.0 again is doing better
  - index RW-lock is dominating for all..



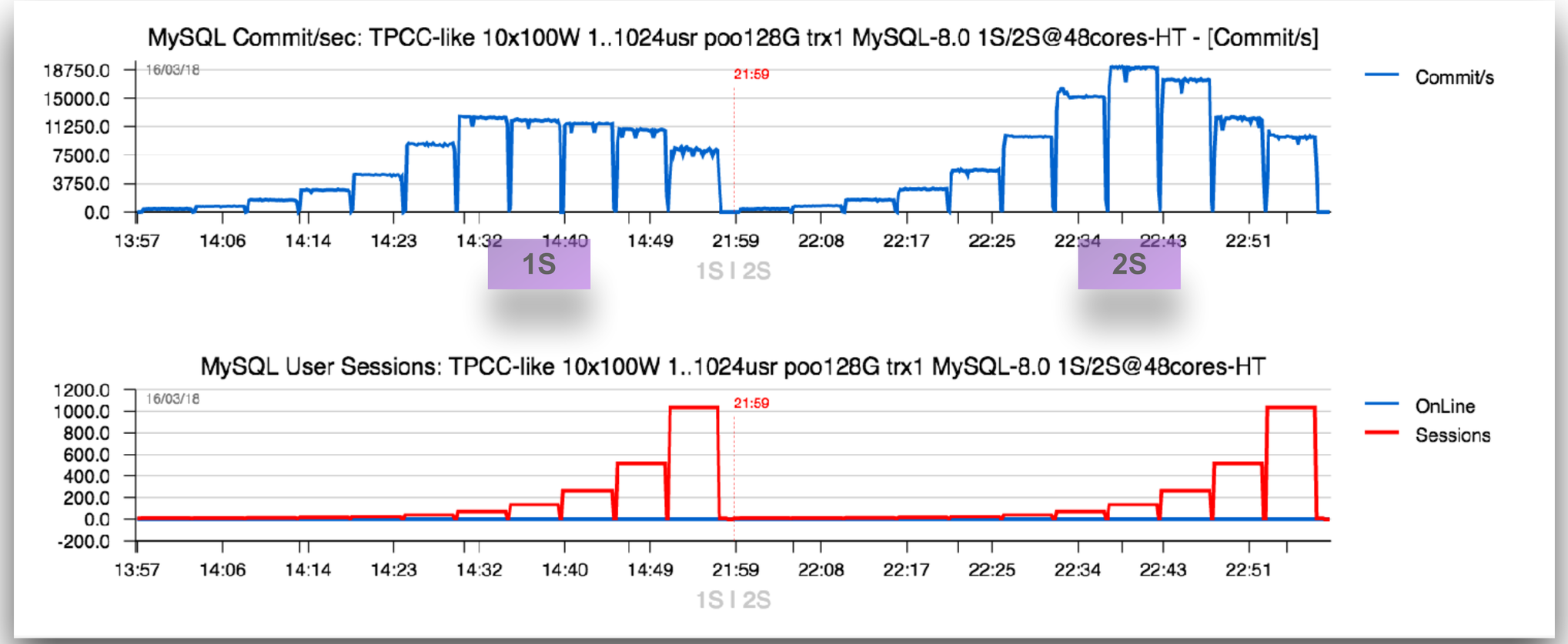
TPCC-like 1000W Workload -vs- DBT2 1000W

- BP = 128GB : in-memory
  - nearly the same workloads, but different “execution profile”
  - no index RW-lock contention in DBT2.. => to investigate



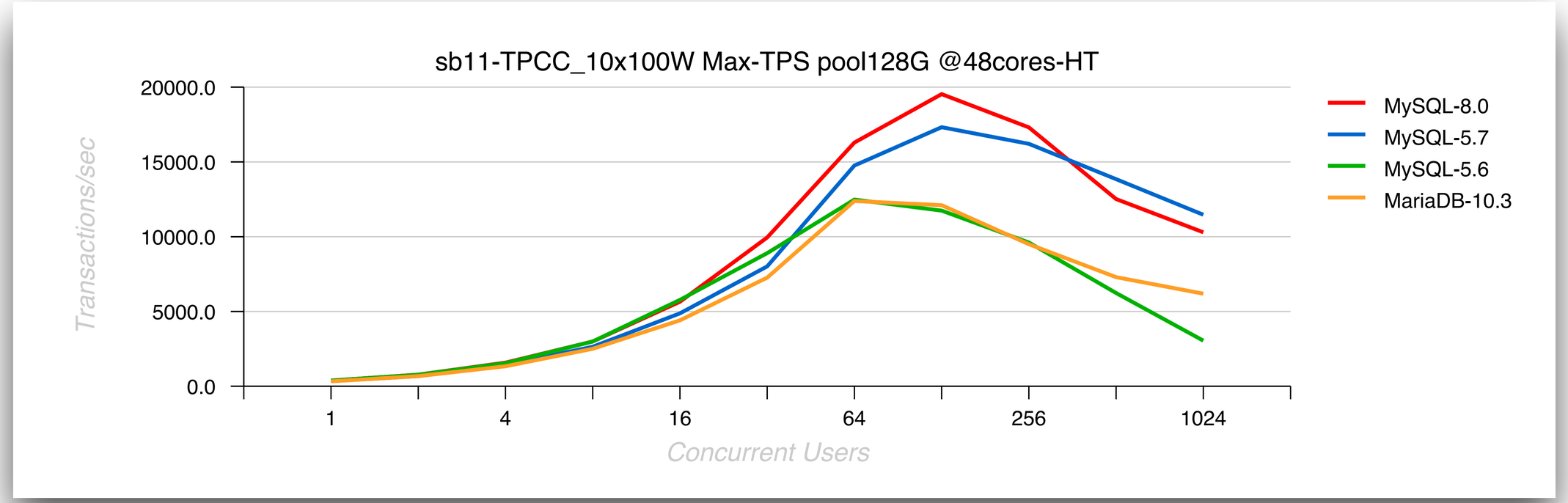
TPCC-like 10x100W Workload

- MySQL 8.0 :
  - over 50% gain between 1S -vs- 2S
  - scalability is limited by “lock_sys” contention (LOCK management in InnoDB)



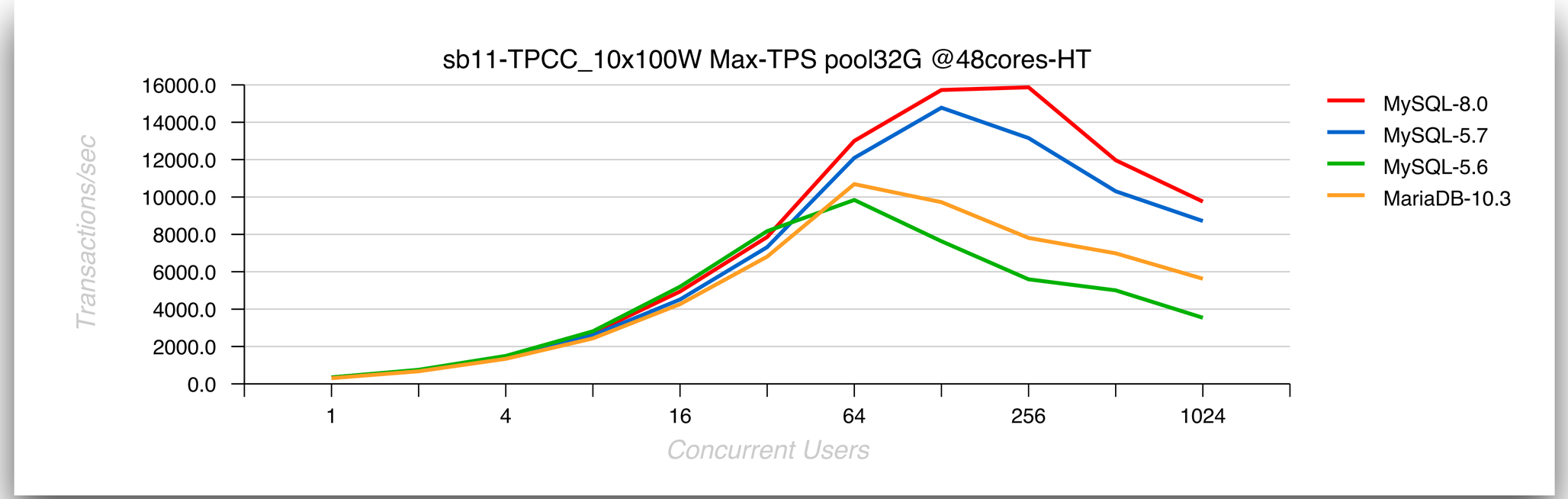
TPCC-like 10x100W Workload

- BP = 128GB : in-memory
  - MySQL 8.0 is doing better
  - drops on high load are due higher “lock_sys”, fix is in progress..



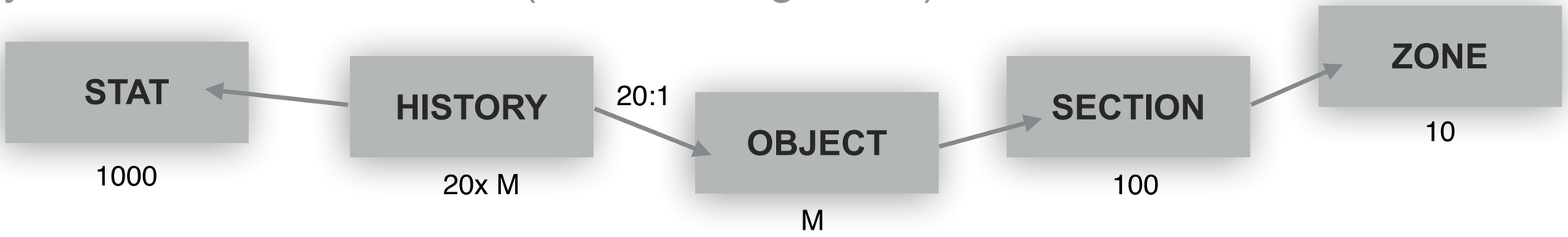
TPCC-like 10x100W Workload

- BP = 32GB : IO-bound ?
  - MySQL 8.0 is still doing better
  - drops on high load are still related to “lock_sys”



dbSTRESS Workload

- Developed & Maintained by me ;-))
  - in progress to be fully ported to Sysbench
  - can be executed as x1 dataset or xN datasets (similar to xN db, except it's the same db)
  - inspired by real customer workload (stock management)
  - schema :

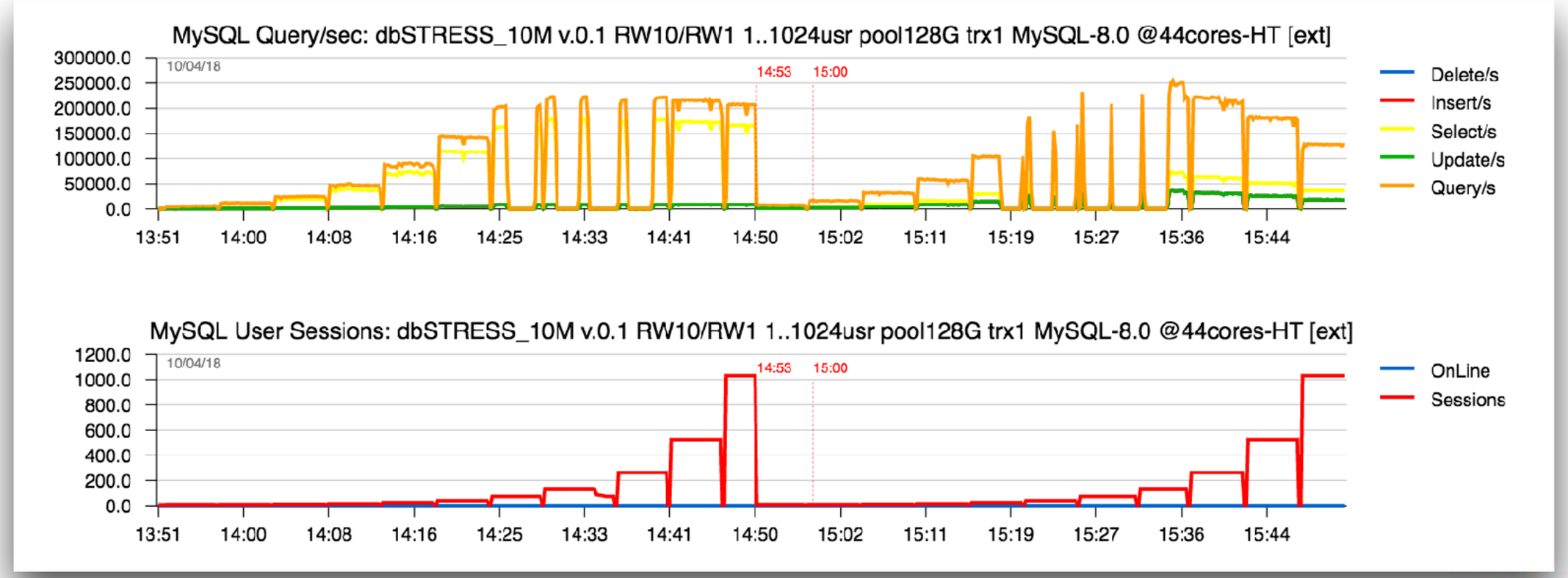


  - queries :
    - SEL1 : join of 3 tables by OBJECT REF
    - SEL2 : join of 2 tables by OBJECT REF
    - WRITE: DELETE/ INSERT/ UPDATE of single HISTORY record by OBJECT REF

- The following Test Scenarios :
  - Datasets : 10M or 10x1M (same ~100GB volume in total, 200M records in HISTORY)
  - BP = 128GB (in-memory)

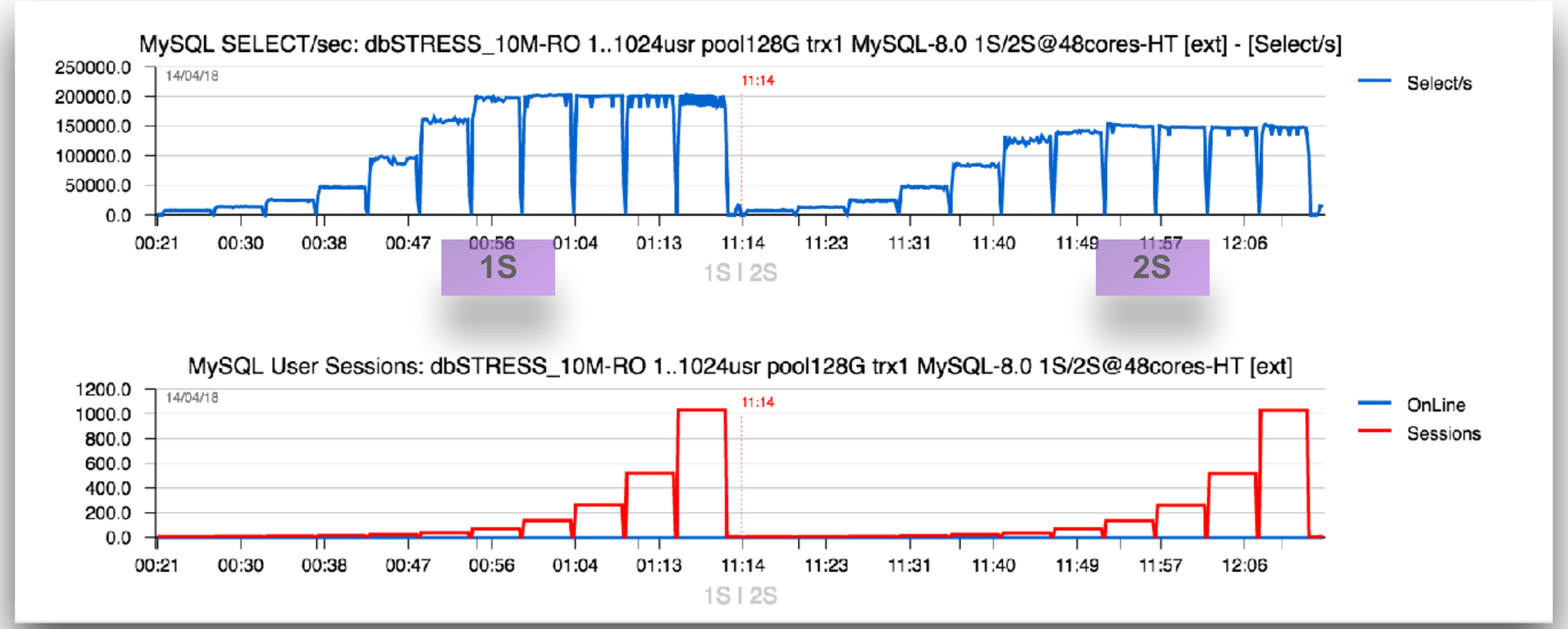
dbSTRESS Probe RW1 Test

- Dataset : 10M (~100GB)
  - surprise… => need a deep investigation..
  - NOTE : a good example how any “uncontrolled” app can kill your Production ;-))
  - Partial Workaround : attribute different REF values ranges for every thread



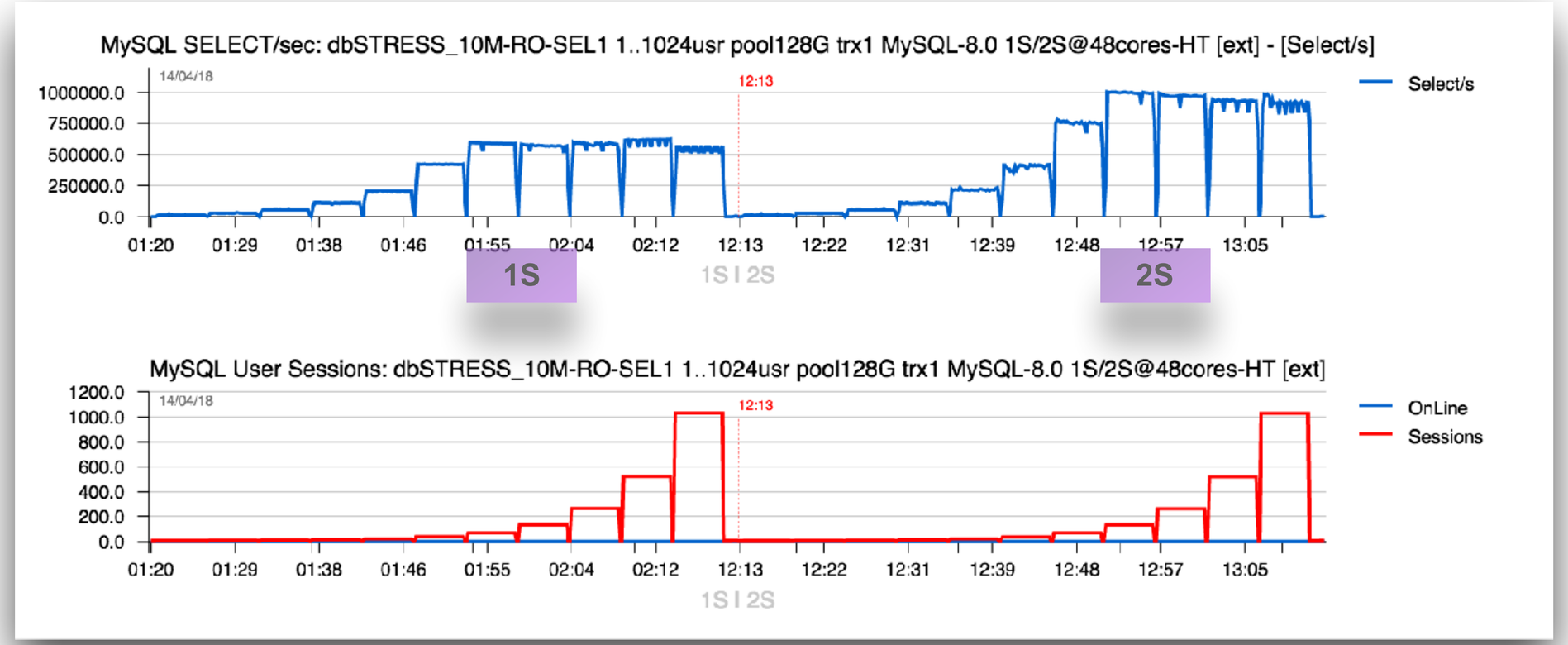
dbSTRESS 10M RO Test

- Dataset : 10M (200M HISTORY)
  - pure RO, but not scaling at all.. — 2S result is even worse than 1S..
  - RO = SEL1 + SEL2
  - so, both queries are not scaling or only one ?



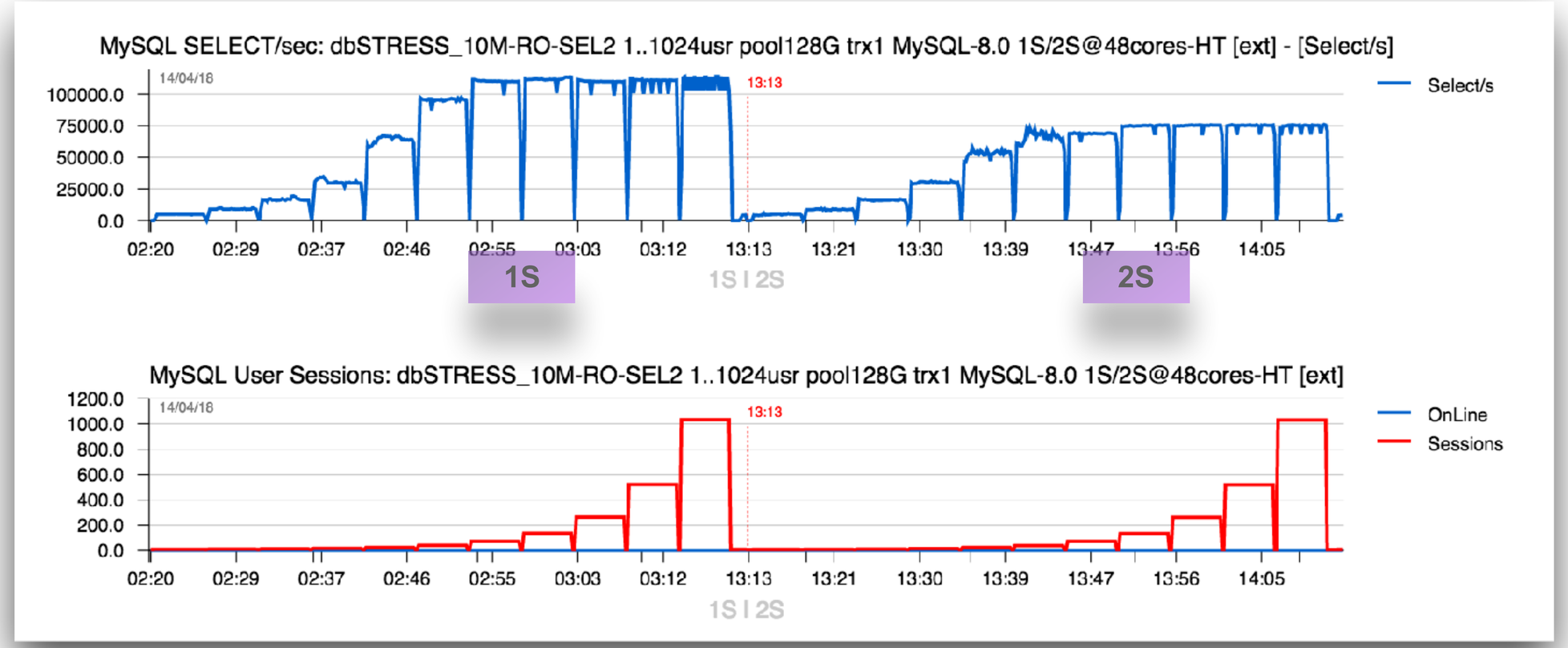
dbSTRESS 10M RO-SEL1 Test

- Dataset : 10M (200M HISTORY)
  - SEL1 is scaling pretty well..
  - what about SEL2 ?..



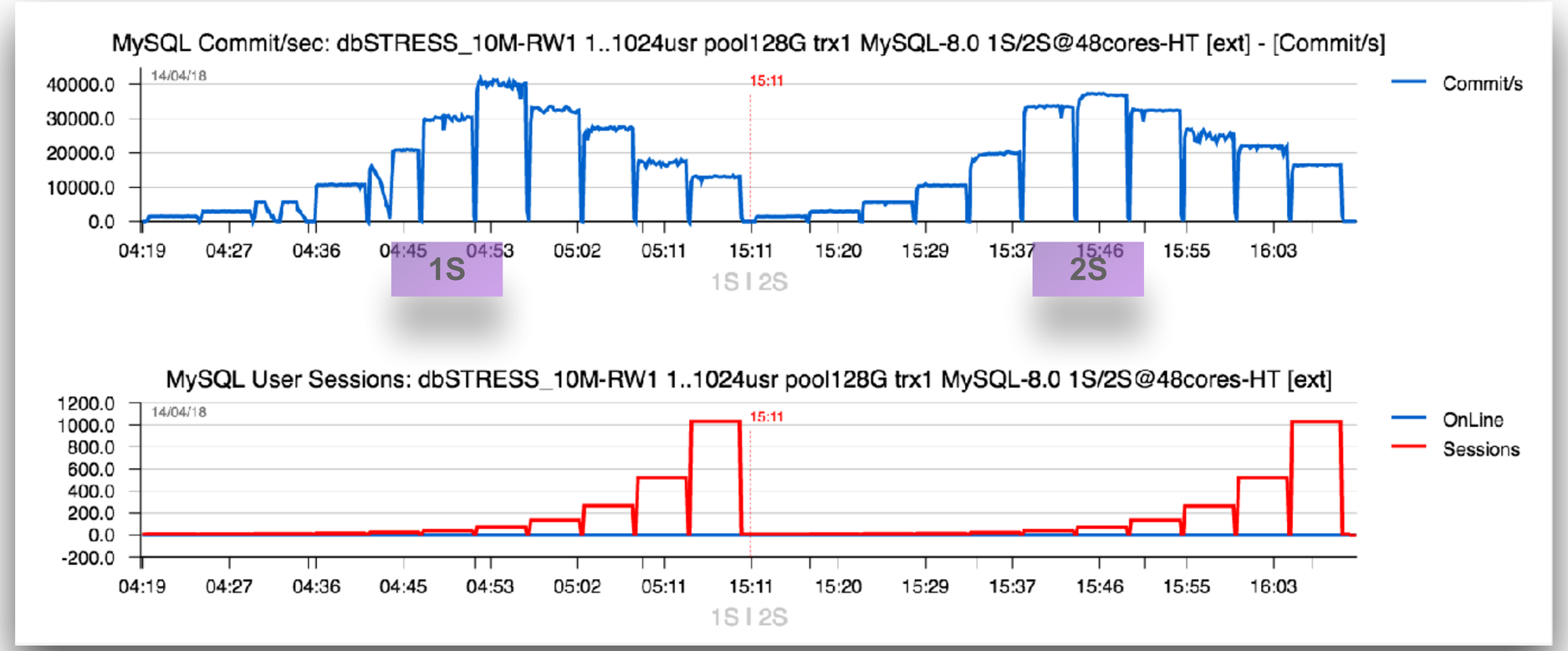
dbSTRESS 10M RO-SEL2 Test

- Dataset : 10M (200M HISTORY)
  - bingo ! — SEL2 is not scaling at all..
  - Why ? — Sec.IDX is part of problem + some more (by design)



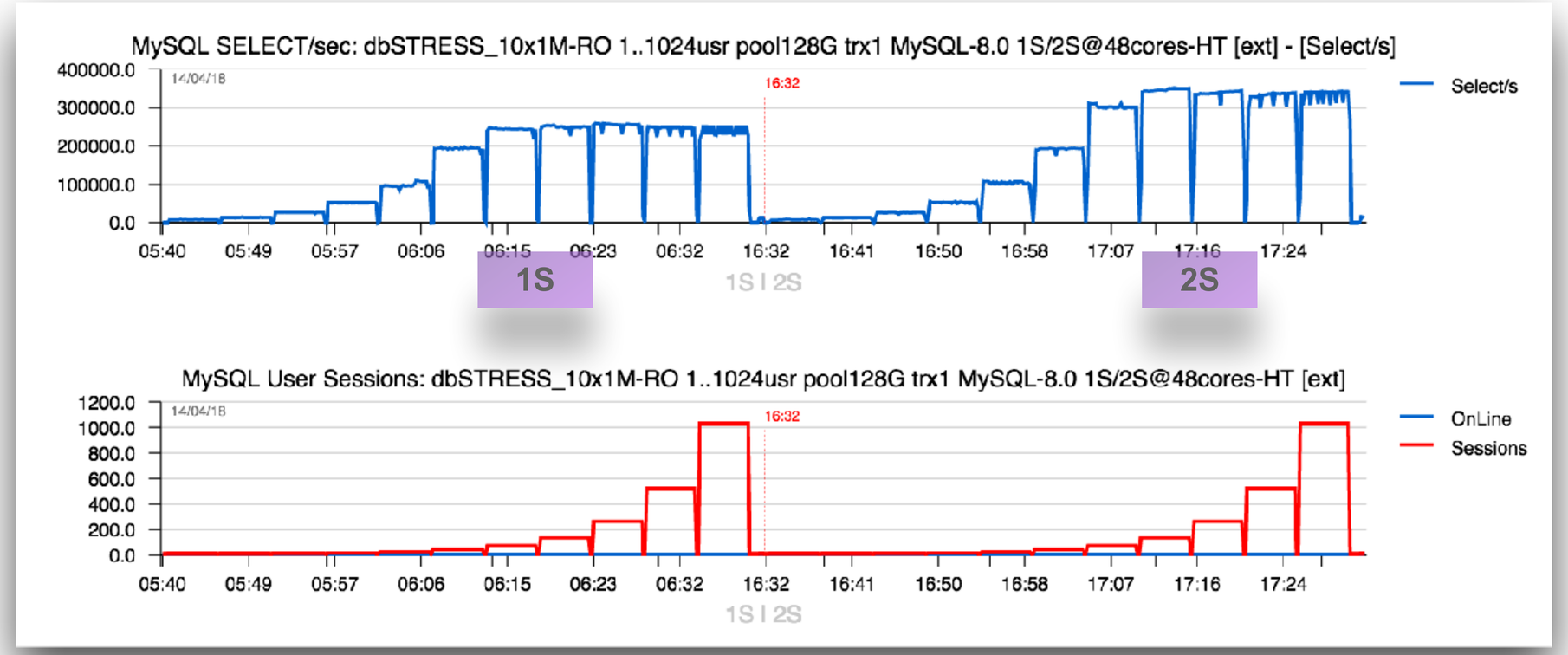
dbSTRESS 10M RW1 Test

- Dataset : 10M (200M HISTORY)
  - as the results, RW1 is not scaling either (RW1 = RW ratio 1:1)
  - still reaching 40K TPS Max anyway
  - but TPS on 2S is lower than 1S..



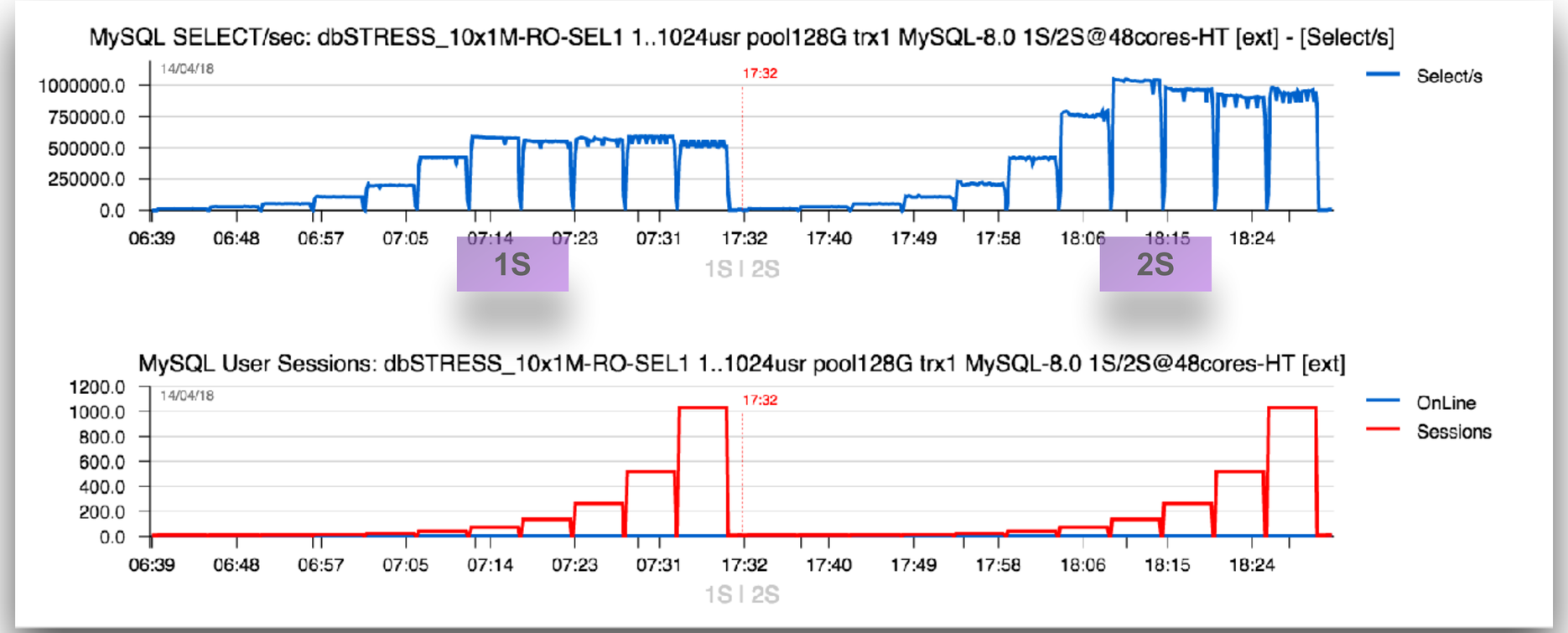
dbSTRESS 10x1M RO Test

- Dataset : 10x1M (10x20M HISTORY)
  - by using 10 datasets in parallel we divide the SEL2 bottleneck problem by 10 as well
  - as the result RO is getting better result on 1S than before, and yet more on 2S !
  - so, what about SEL1 and SEL2 ?



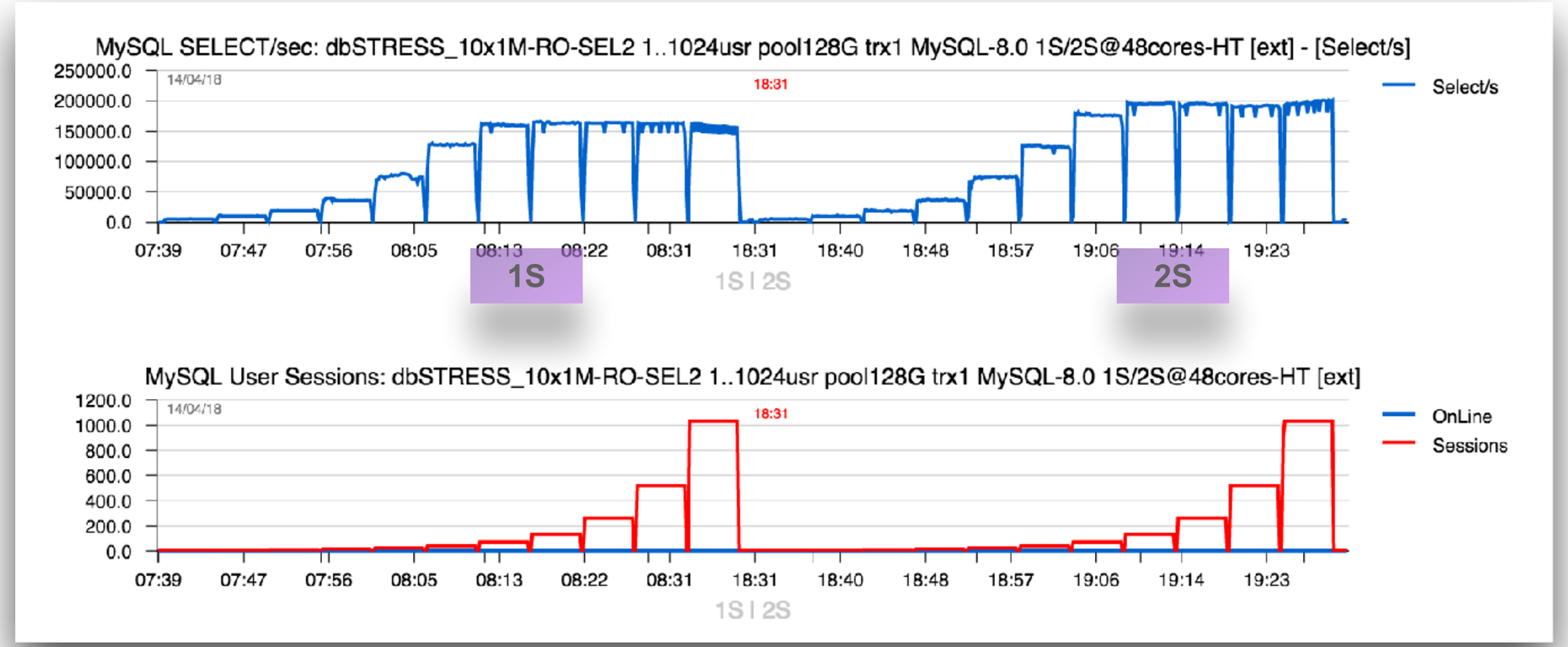
dbSTRESS 10x1M RO-SEL1 Test

- Dataset : 10x1M (10x20M HISTORY)
  - SEL1 was already well scaling on 10M Test, so no difference
  - not far from x2 times better performance on 2S, good !
  - so, what about SEL2 now ?



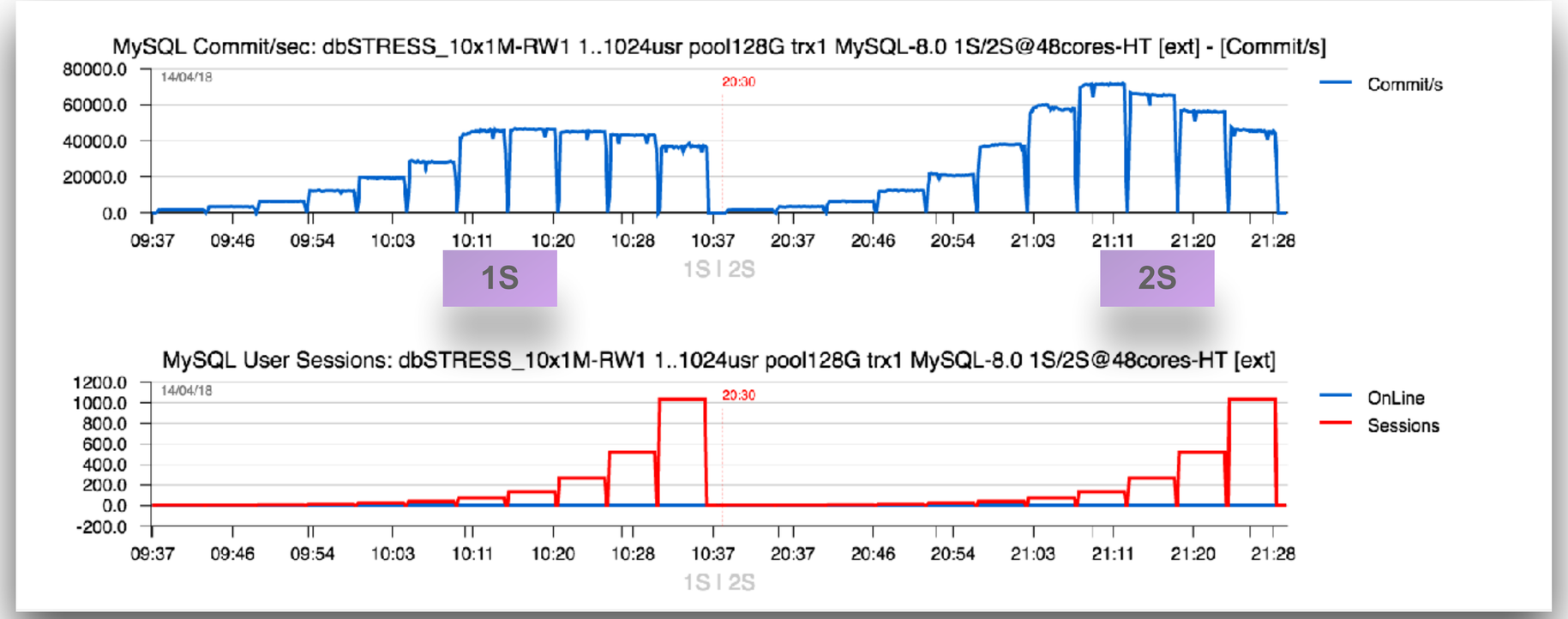
dbSTRESS 10x1M RO-SEL2 Test

- Dataset : 10x1M (10x20M HISTORY)
  - the result on 2S is only “slightly better” than on 1S..
  - which is pointing on a “bigger issue” than just a single bottleneck.. (Sec.IDX design)
  - but will this help RW1 workload in anyway ?



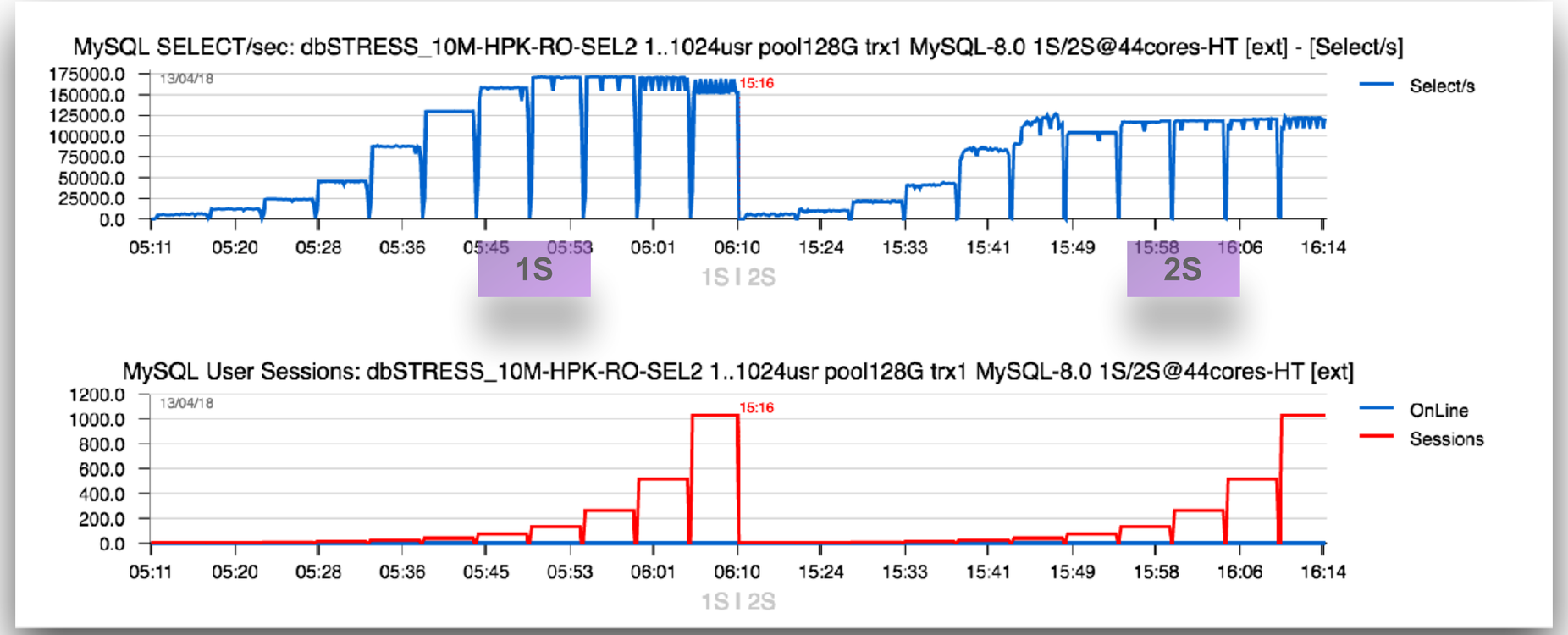
dbSTRESS 10x1M RW1 Test

- Dataset : 10x1M (10x20M HISTORY)
  - so far, reaching **70K** TPS on 2S (which is 75% more than 40K before on 10M Test)
  - however, we're just "hiding" the real problem by splitting the 10M Test into x10
  - if Sec.IDX design is part of the problem, could it help if we use PK instead ?



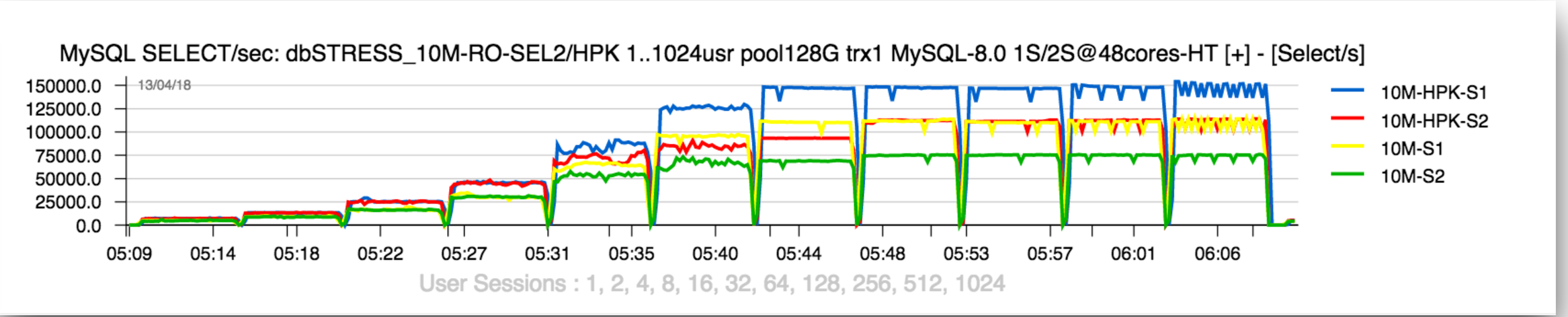
dbSTRESS 10M RO-SEL2 Test + HPK

- Dataset : 10M (200M HISTORY+PK (HPK))
  - no need to look on RO and RO-SEL1 Tests, as we know now RO results is driven by SEL2
  - so, still no scaling even with PK used for HISTORY table..
  - but what about numbers-to-numbers comparison with initial 10M Test ? (Sec.IDX)



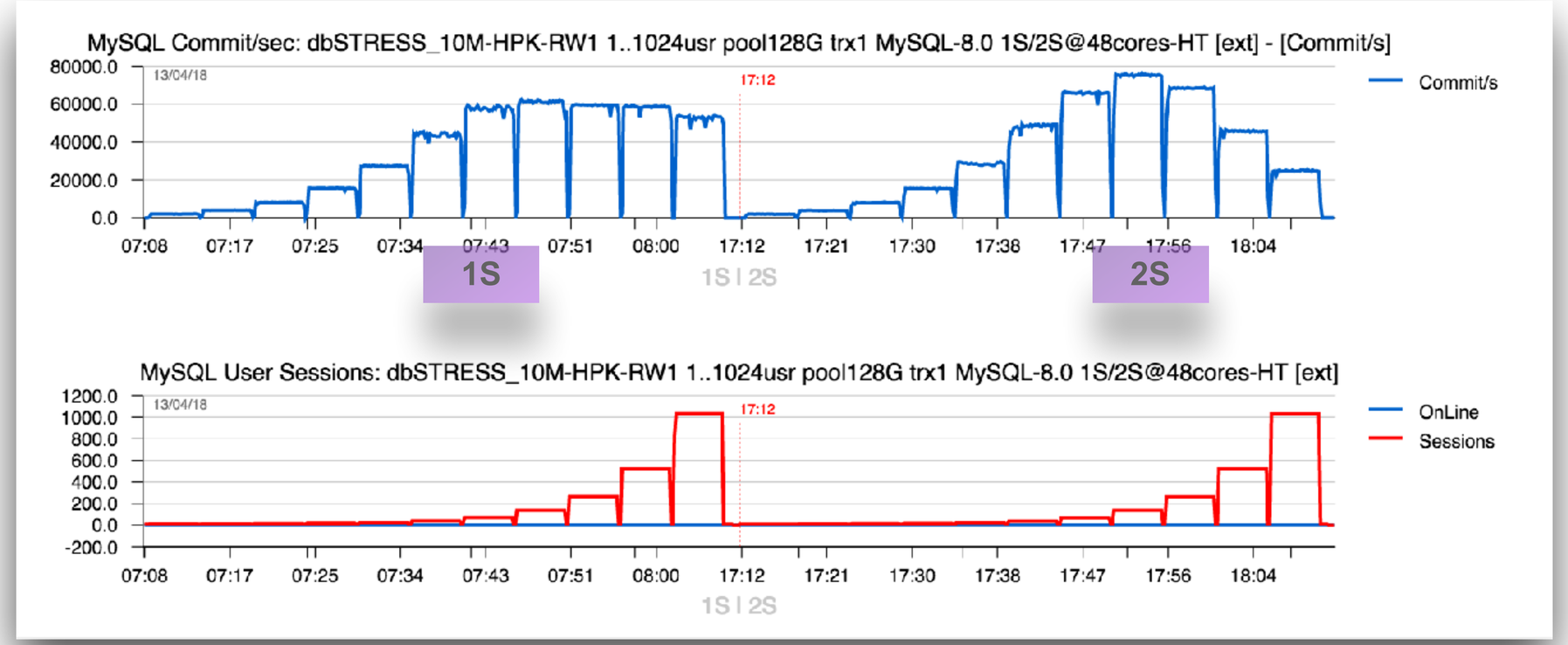
dbSTRESS 10M RO-SEL2 Test + HPK

- Dataset : 10M (200M HISTORY+PK (HPK))
  - the gain with PK over Sec.IDX is visible, but not that big as we could “expect”..
  - however, the real question also is it only impacting RO queries ?
  - what about RW1 now ?..



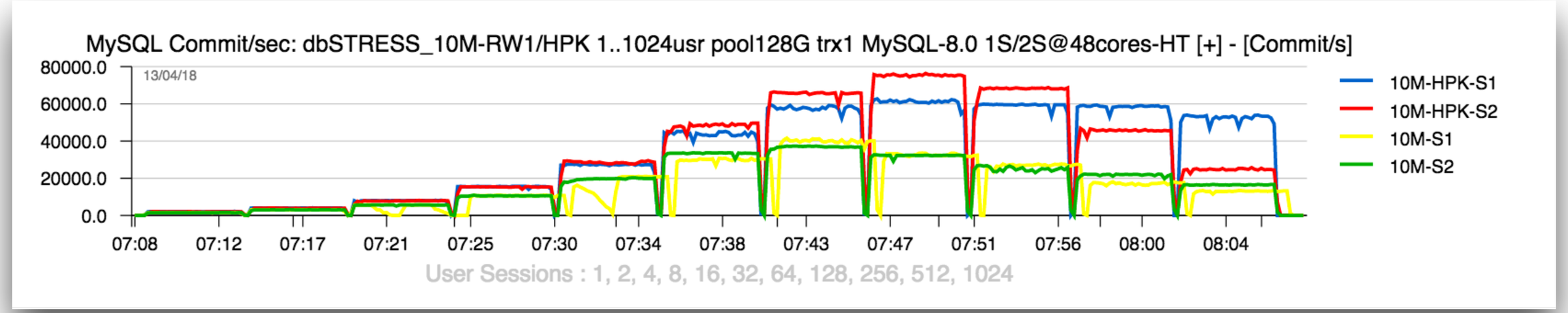
dbSTRESS 10M RW1 Test + HPK

- Dataset : 10M (200M HISTORY+PK (HPK))
  - over **60K** TPS on 1S and **75K** TPS on 2S !!
  - looks much better, but only 25% gain over 1S, so we're yet far from expected scalability..
  - so, PK is also greatly impacting WRITE operations, and how big the final difference ?



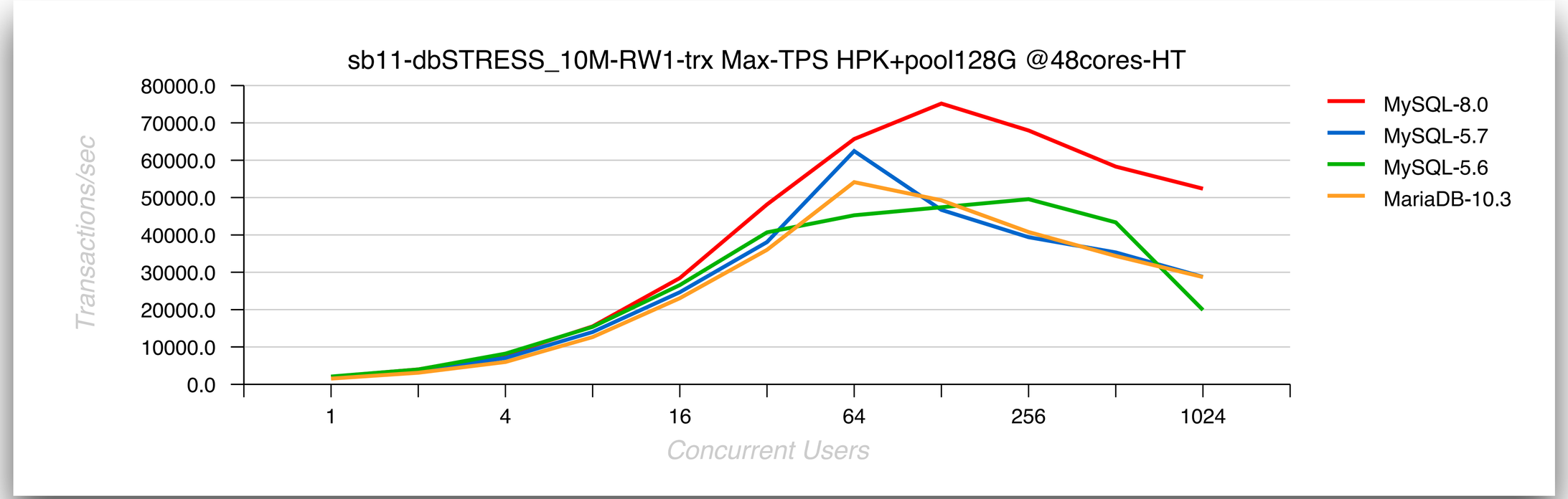
dbSTRESS 10M RW1 Test + HPK

- Dataset : 10M (200M HISTORY+PK (HPK))
  - as you can see, the impact of using PK over Sec.IDX is really huge..
  - near x2 times better Max TPS (75K vs 40K)
  - seems to be the most “favorable” test scenario to run with all other MySQL versions..



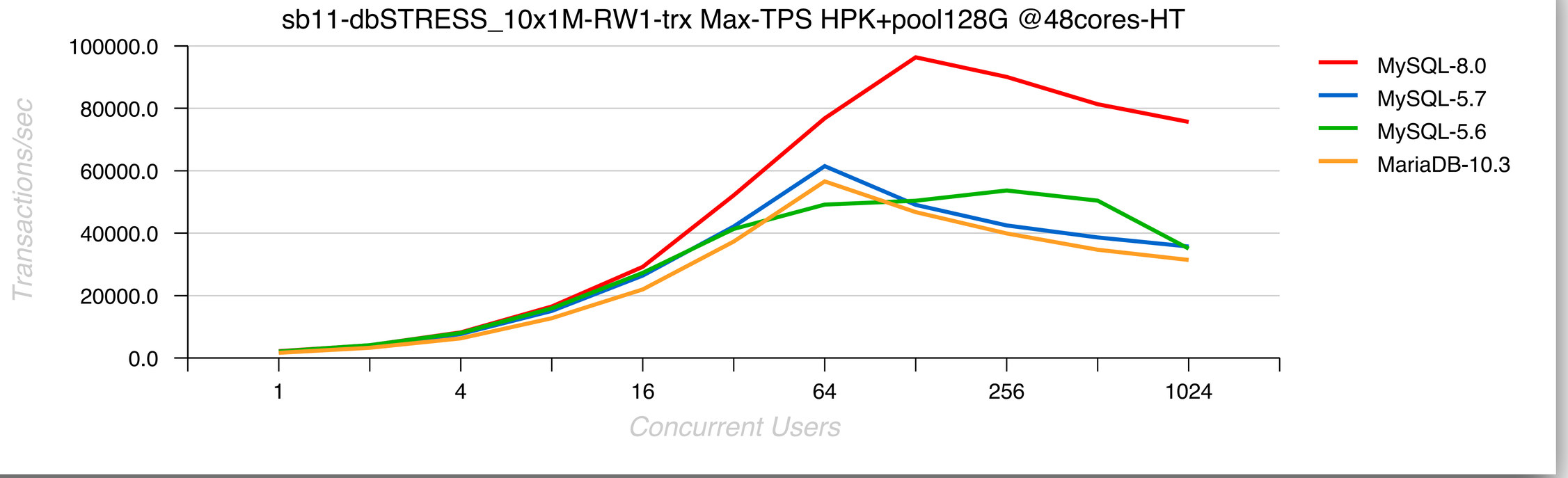
dbSTRESS 10M RW1 Test + HPK

- Dataset : 10M (200M HISTORY+PK (HPK))
  - as usual, “everything is relative”..
  - while we blame MySQL 8.0 for scalability limits, it’s still doing better than all we have ;-))
  - and what if we’ll combine this with 10x1M split ?..



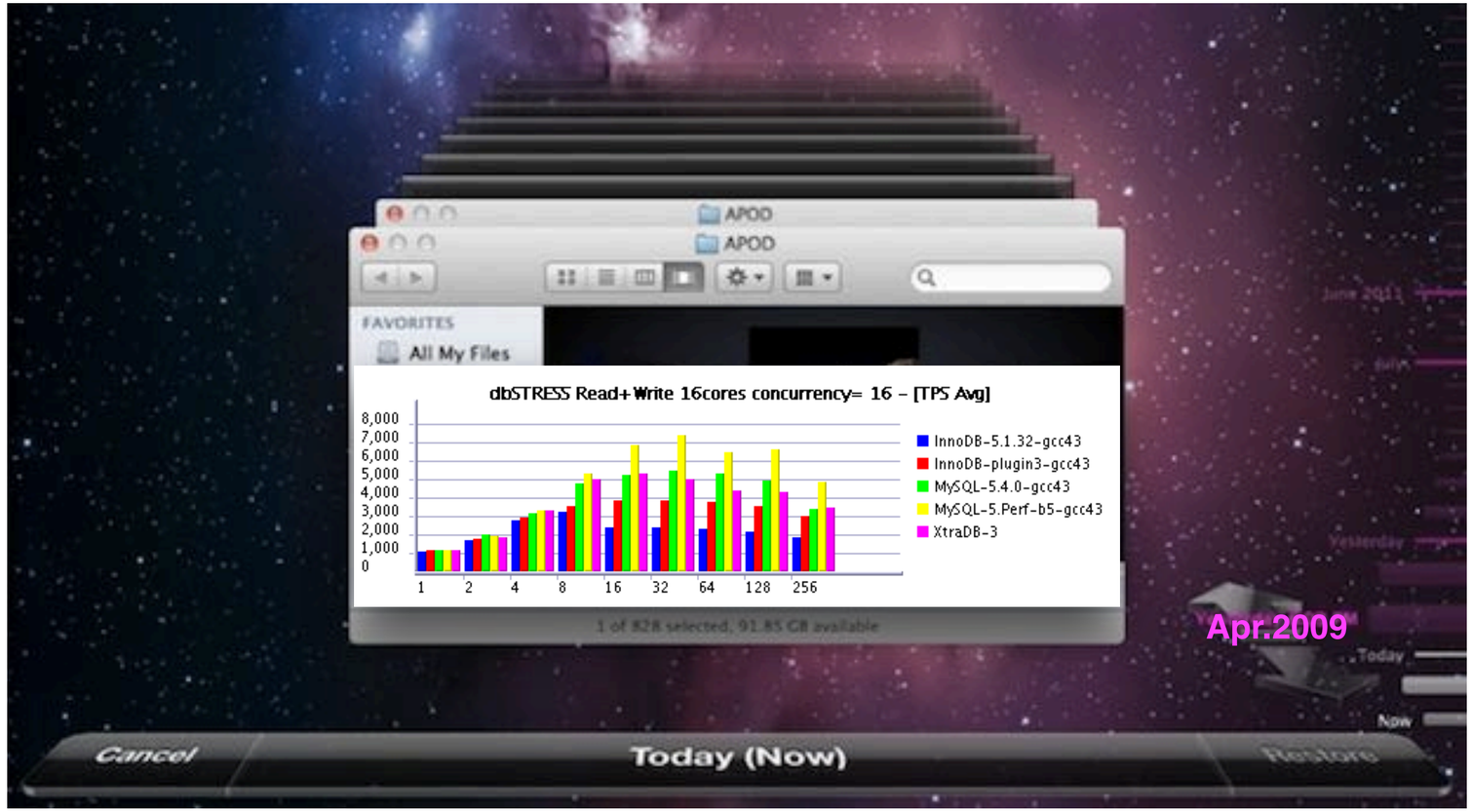
dbSTRESS 10x1M RW1 Test + HPK

- Dataset : 10x1M (10x20M HISTORY+PK (HPK))
  - even more once again — “everything is relative”..
  - reaching **95K** TPS with MySQL 8.0 !!
  - but still a lot of work ahead (if you don’t look on the numbers only, but under hood)..



dbSTRESS RW1 Test & Time Machine..

- Looking back with Time Machine :
  - even more once again — “everything is relative”..
  - **Apr.2009** : dbSTRESS 10M RW1 => **7.5K** TPS **only** as the best ever possible result !!
  - 9 years => and so huge jump in SW/HW + MySQL 8.0 !!! => x10 times diff..



Thank You !!! 🍾🍾🍾

ORACLE

One more thing ;-)

- All graphs are built with dim_STAT (http://dimitrik.free.fr)
  - All System load stats (CPU, I/O, Network, RAM, Processes,...)
    - Mainly for Linux, Solaris, OSX (and any other UNIX too :-)
    - Add-Ons for MySQL, Oracle RDBMS, PostgreSQL, Java, etc.
    - Linux : PerfSTAT (“perf” based), mysqlSTACK (quickstack based)
  - 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.