MySQL

ORACLE

MySQL Performance: Understanding & Tuning of CPU-Bound Workloads (part-1)

Overview 2026

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

- **Standalone** MySQL Performance & Scalability
  - part-1 : Understanding & Tuning of CPU-Bound Workloads (aka In-Memory)
  - part-2 : Understanding & Tuning of IO-Bound Workloads

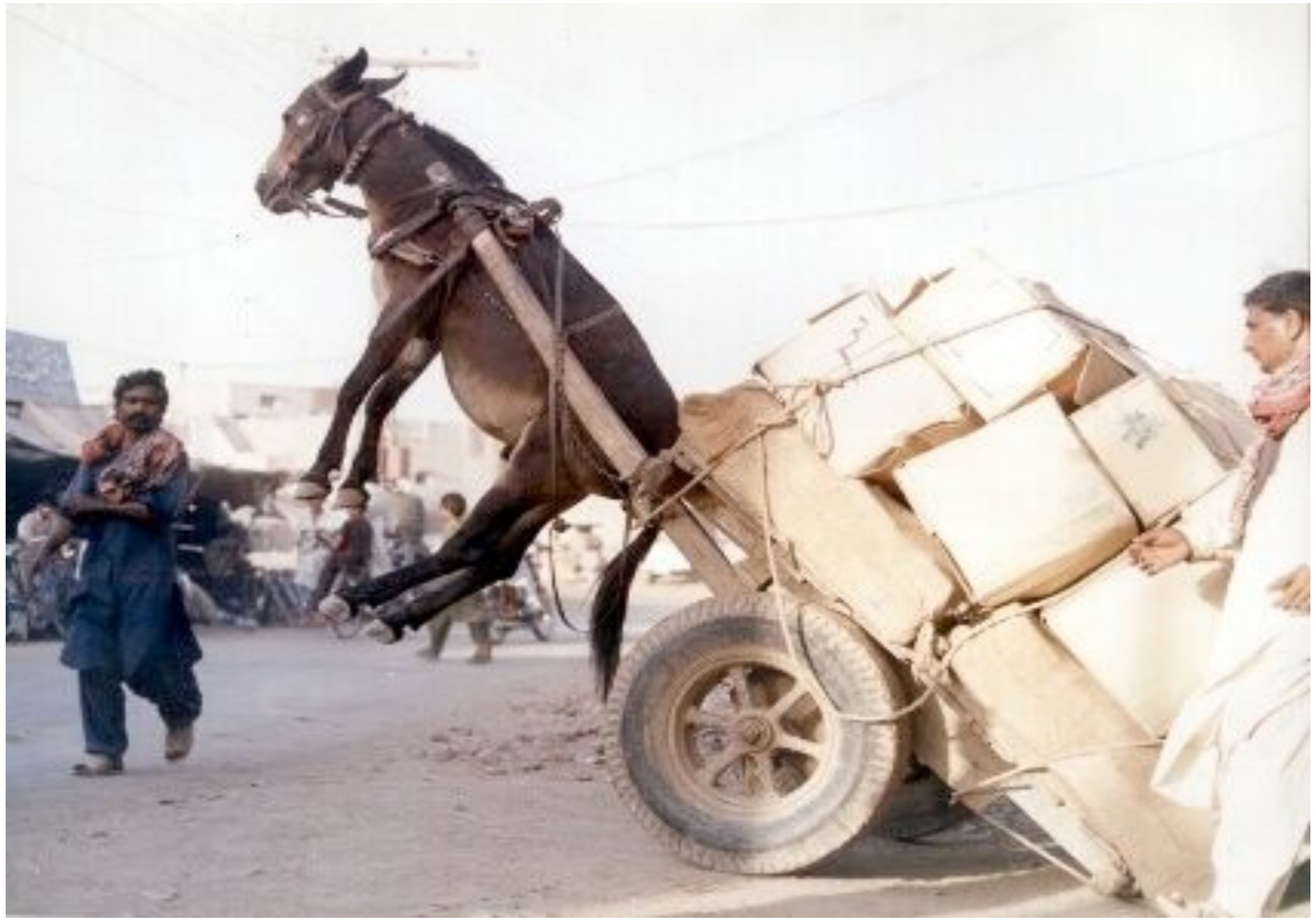
Why Performance & Scalability ?...

Why Performance & Scalability ?..

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

Why Performance & Scalability ?..

- Until it did not reach its limit..



Why Performance & Scalability ?..

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

Why Performance & Scalability ?..

- And reach a similar limit..

Why Performance & Scalability ?..

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

Why Performance & Scalability ?..

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

Why Performance & Scalability ?..

- World is changing constantly :
  - and what was considered rock solid yesterday, may become weak and smashable tomorrow..

But if you let “russian word” come to your Home..

- Final REMINDER :
  - Along with your WORK there is also REAL LIFE, don’t miss it..

MySQL Core Workload Target

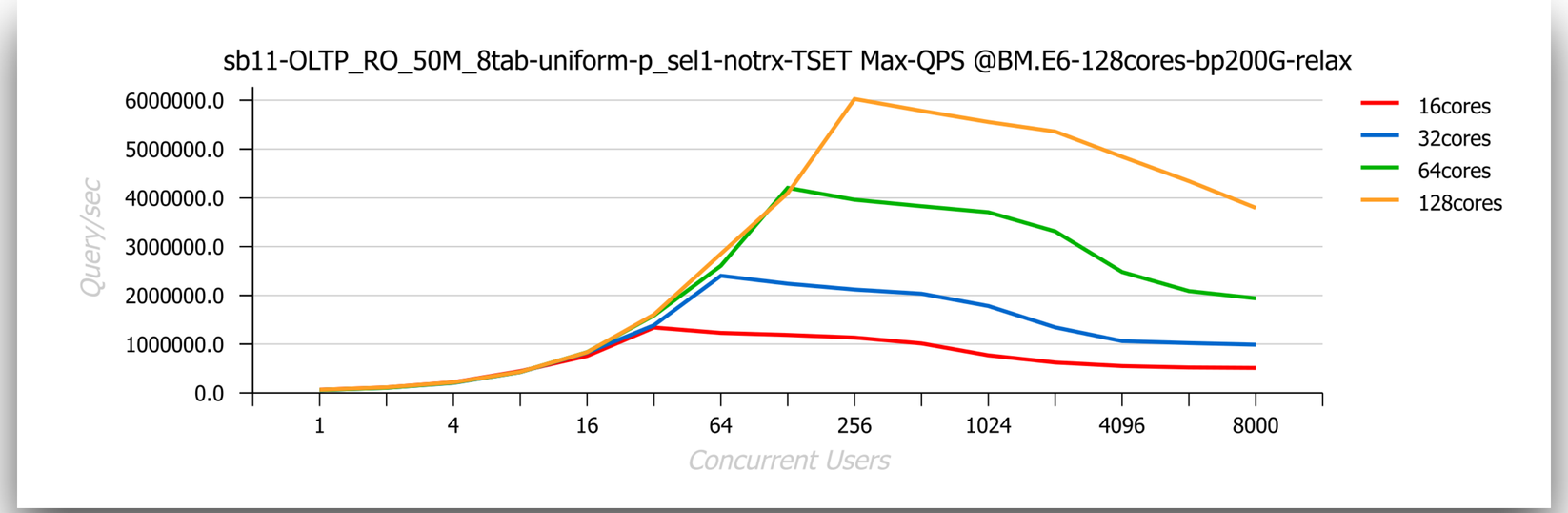
- Historically & Mainly => OLTP !
  - e.g. generally about fast short queries
  - do you recall LAMP ? Web ? Internet ? ;-))
- CPU-bound Workloads (aka In-Memory) :
  - no IO Reads from Storage
  - NOTE : except for TEMP tables (but not too much)
- IO-bound Workloads :
  - with IO Reads from Storage
  - generally active dataset is (much) bigger than RAM (or/and BP size)
- Side NOTE :
  - Hardware + Storage + Network + System setup choice = baseline for gains or pains !

CPU-bound : How Fast ?..

- **Q** : what is the fastest SQL Query ?
  - spoiler : please, don’t say “SELECT 1” ;-))

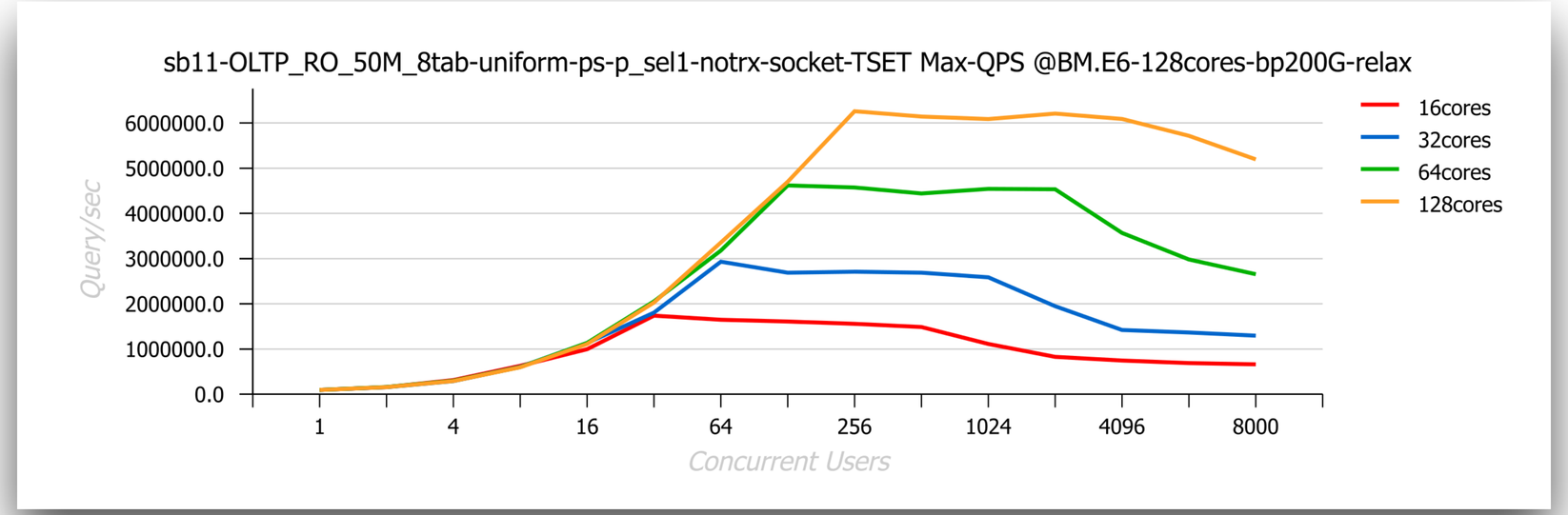
CPU-bound : How Fast ?..

- **A** : Lookup via Primary Key (aka PK-lookup)
  - ex.: “SELECT .. FROM table where ID = 1”
    - (where ID is PK)
    - Sysbench : point-selects workload
  - shorter PK => faster lookup
  - generally scales pretty well
  - NOTE : mind “clustered” nature of InnoDB tables



CPU-bound : Can Be Faster ?..

- PK-lookup Speed-Up :
  - using Prepared Statements => 10-20% boost
    - skipping Parser step, e.g. gain is only for short queries
  - connecting via UNIX-Socket => 10-100% boost (depends on the System)
  - enabling InnoDB Adaptive-Hash-Index (aka AHA) => helps, but it depends..
    - based on RW-locks
    - RW-locks are not scaling “by definition” (even for Read-Only)



Speed-UP with PGO

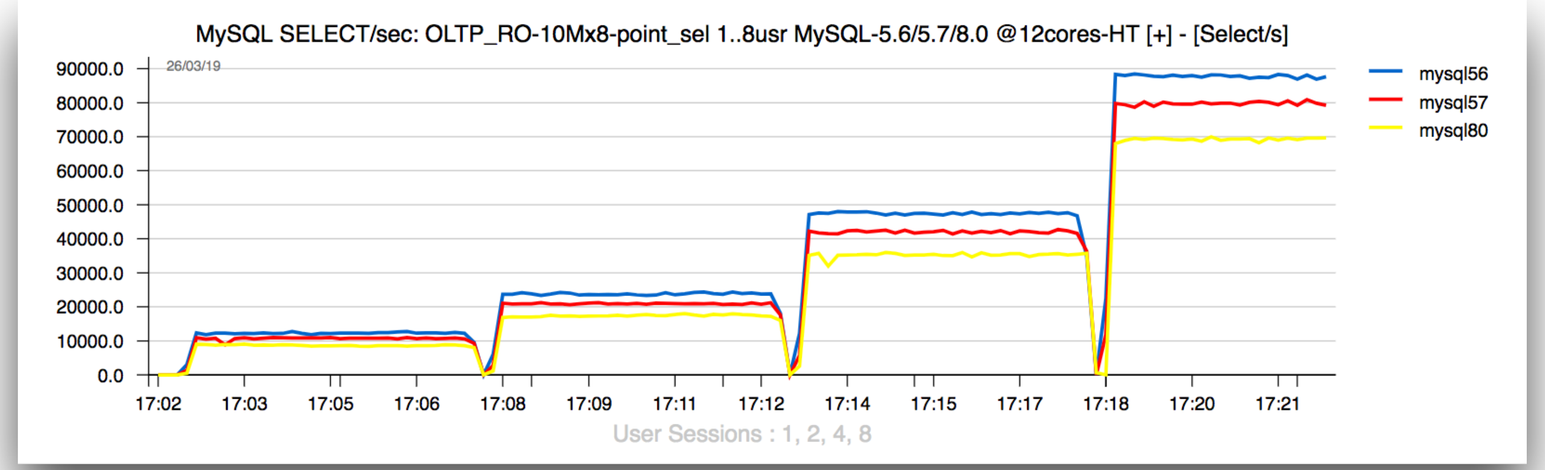
- 2019 : ready to deploy
- 2020 : …
- 2021 : …
- 2022 : …
- 2023 : …
- 2024 : …
- 2025 : …
- 2026 : finally OK to deploy !
  - expectation : to see all delivered MySQL binaries to be PGO-optimized since now

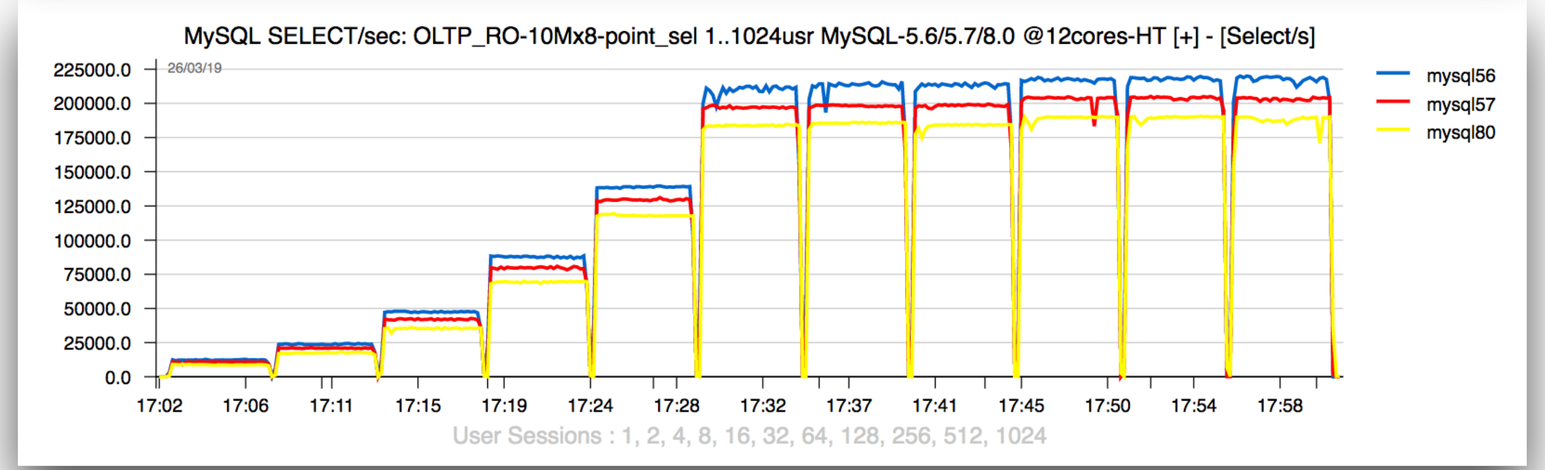
Small HW (May.2019 PLIVE Austin US)

- Unexpected Performance Regressions..
  - historically : single thread only (more new features => longer code path)
  - currently : **even on high load**..
- Looking closer :
  - Server : 12cores-HT Intel
  - Workloads (all in-memory) :
    - OLTP_RO
    - point-selects
    - OLTP_RW
    - Update_NoKEY
- So, **WHY** ???
  - the story is no more about longer code path..
  - the story is about bigger and bigger amount of instructions in binary (and icache miss) !
  - solution : optimize the MySQL 8.0 binary itself !
  - with strong collaboration with Oracle Linux Team ! (KUDOS Karsten !!! ;-))

Small HW (May.2019 PLIVE Austin US)

- OLTP_RO point-selects 10Mx8tab-uniform latin1 :

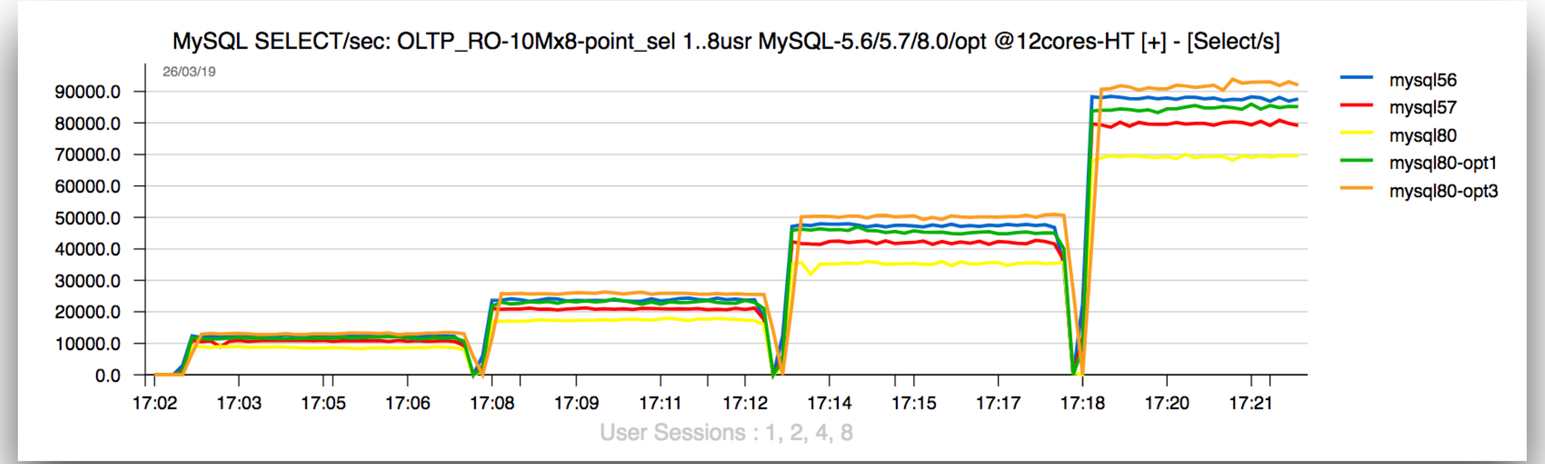


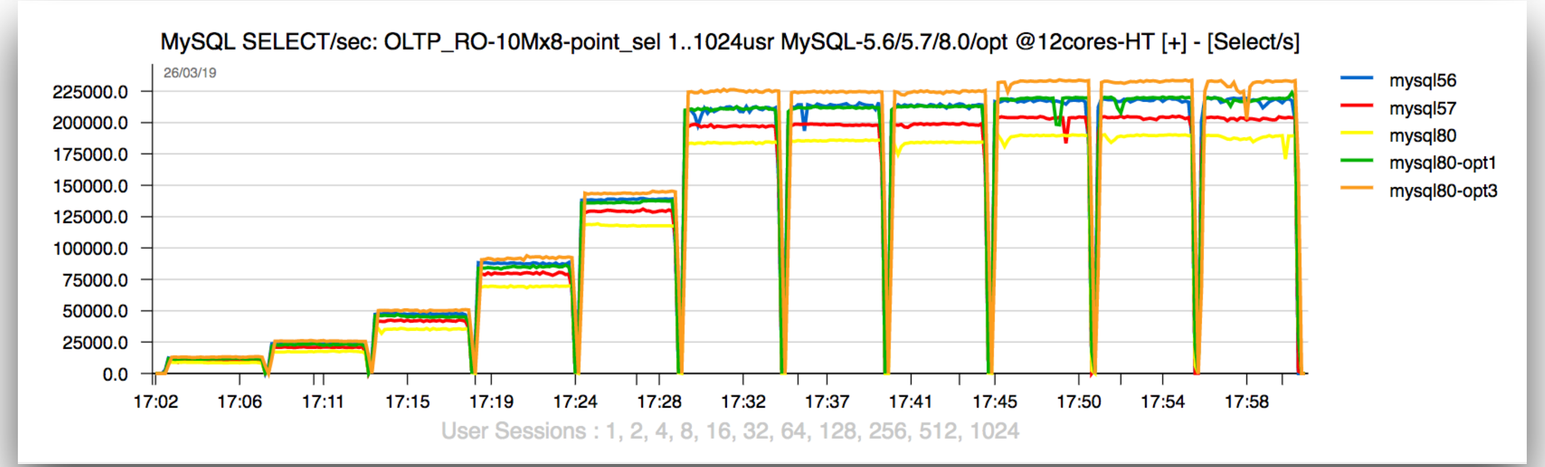


ORACLE

Small HW (May.2019 PLIVE Austin US)

- OLTP_RO point-selects 10Mx8tab-uniform latin1 + optimized :





PGO : Quick HOWTO (if you’re impatient)

- 3 steps :
  - compile
  - traine
  - re-compile

Compile with PGO enabled

```
$ cd mysql-sources
$ mkdir pgo-build
$ cd pgo-build
$ cmake3 .. -DFPROFILE_GENERATE=1 <other options as usual>
```

Run MTR tests

```
$ ./mysql-test/mtr --mem --force --max-test-fail=1000 --parallel=20
```

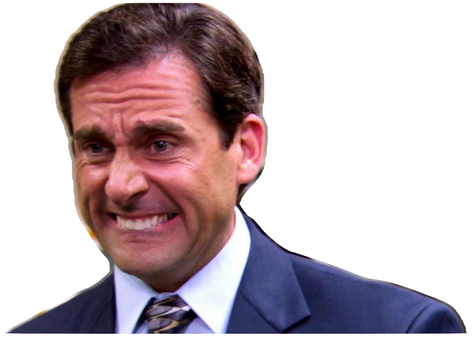
Compile final PGO optimized MySQL binary

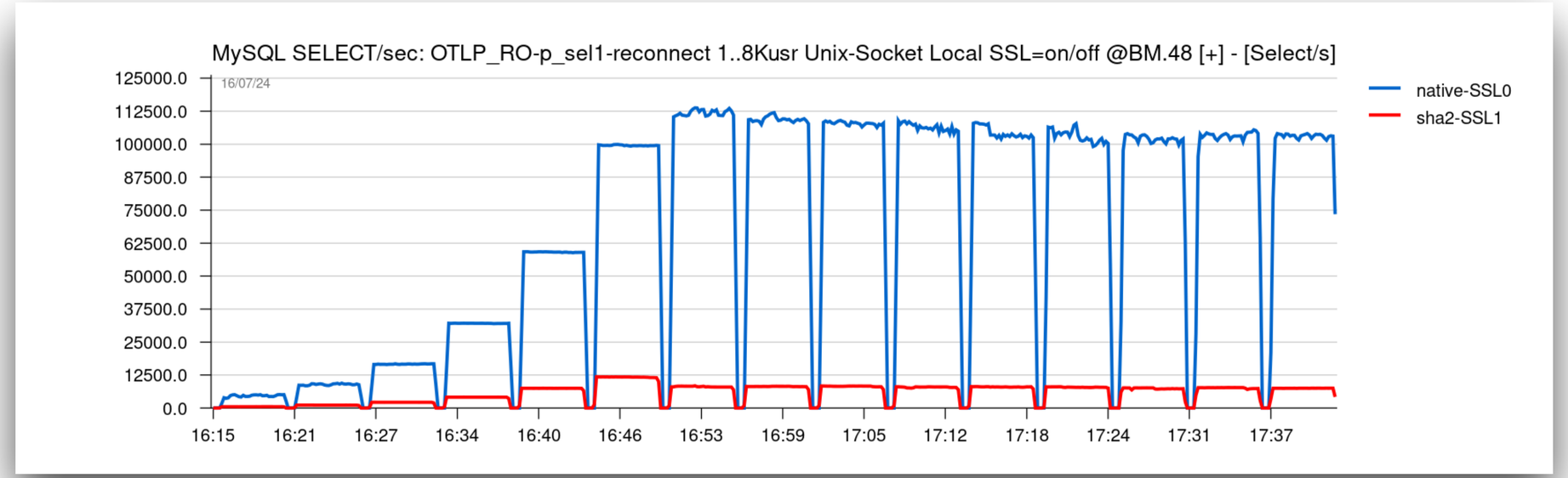
```
$ cd ..
$ mv pgo-build pgo-prepare
$ mkdir pgo-build
$ cd pgo-build
$ cmake3 .. -DFPROFILE_USE=1  <other options as usual>
$ cp runtime*/mysqld /path/to/mysqld-pgo
```

that's all ;-))

CPU-bound : Can be slower ?..

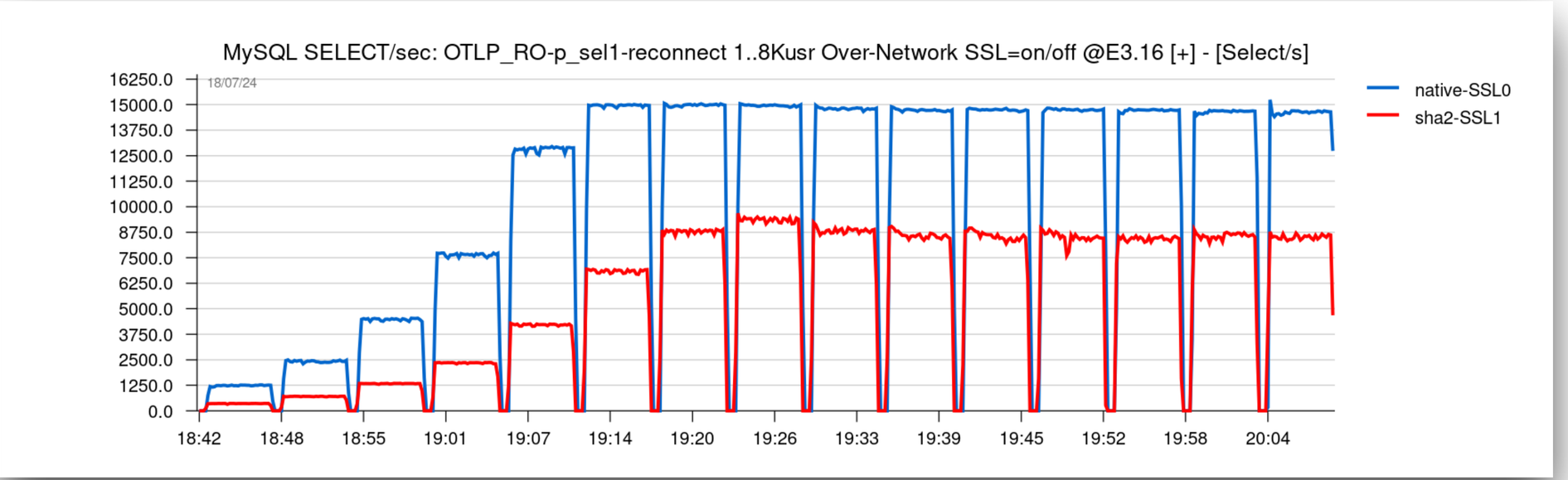
- Re-Connect on every query / transaction
  - aka PHP-based Web sites re-connecting on every Page refresh, etc..
  - BTW, “why” Oracle DB could not host PHP-based Web sites in 90’ ?..
- SSL = on
  - example of overhead with re-connect & SSL = 0/1 running on BM.48 server :
  - (with SSL = 0 and without re-connect it delivers 1M QPS)





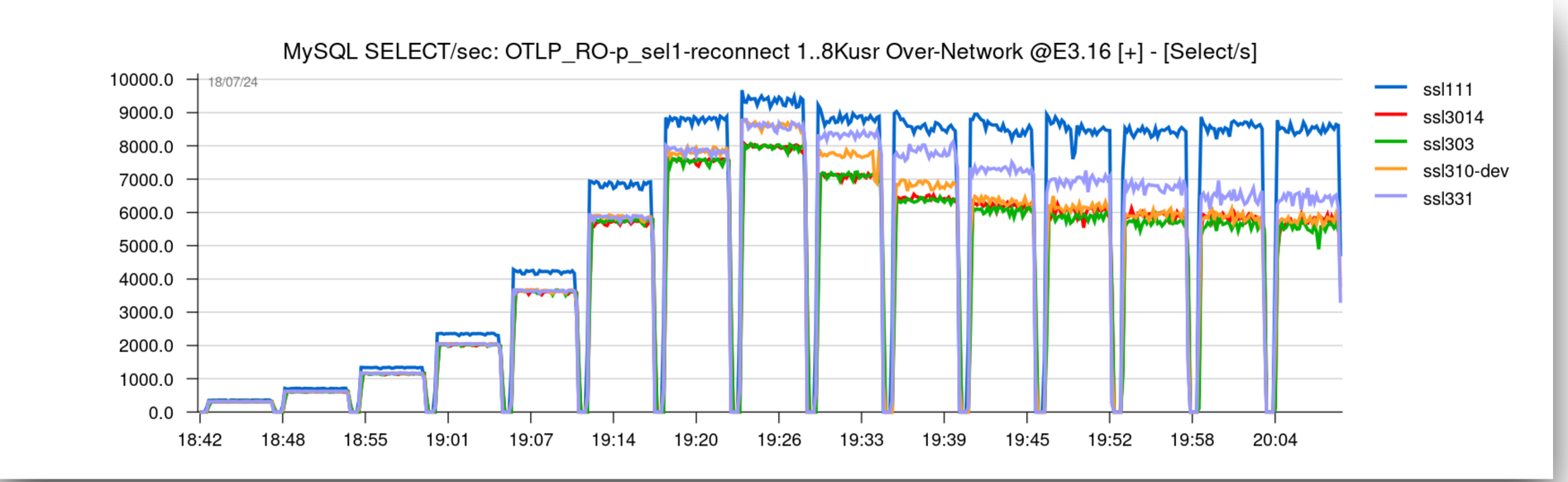
Re-Connect with SSL Over-Network

- Network is always adding latency
  - Network / NIC capacity is critical !
  - example from 16cores OCI VM with OpenSSL-1.1.1 :



Re-Connect with SSL Over-Network | OpenSSL versions

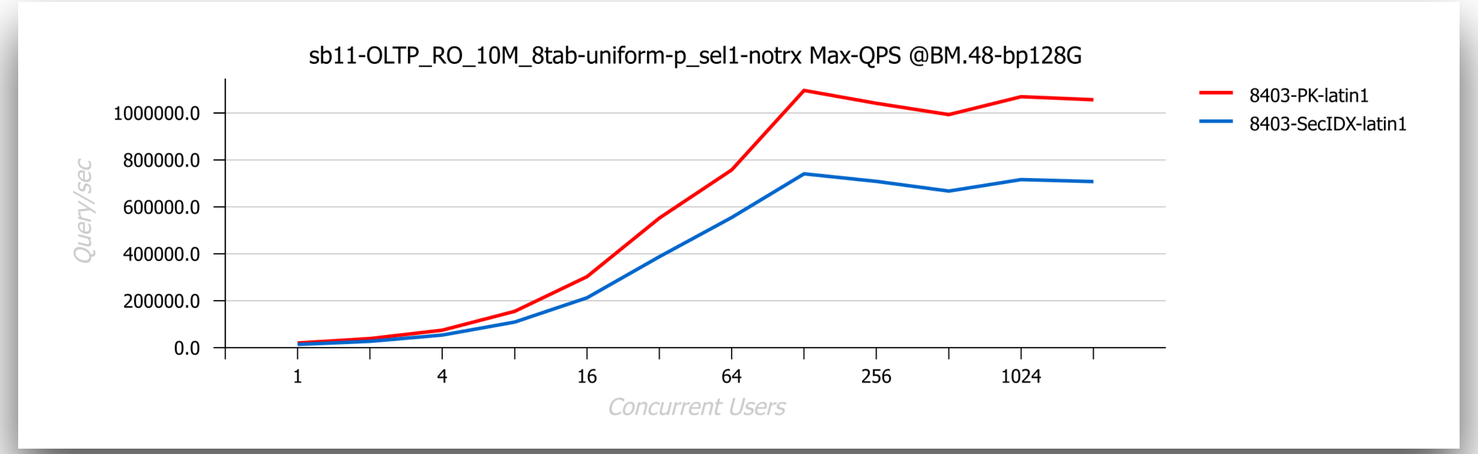
- OpenSSL regressions
  - OpenSSL-1.1.1 is still the best
  - example from 16cores OCI VM :



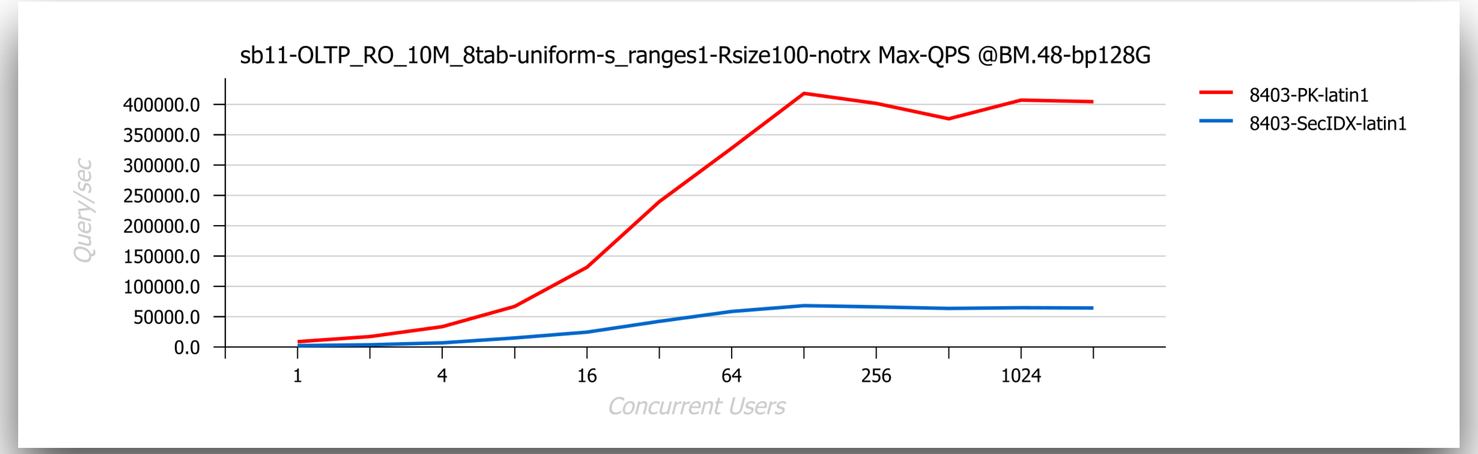
  - NOTE : evaluation of OpenSSL-3.5 is in my next TODO

Primary KEY or Secondary KEY ?..

- Point-SELECT
  - SELECT ..
    WHERE id = $v

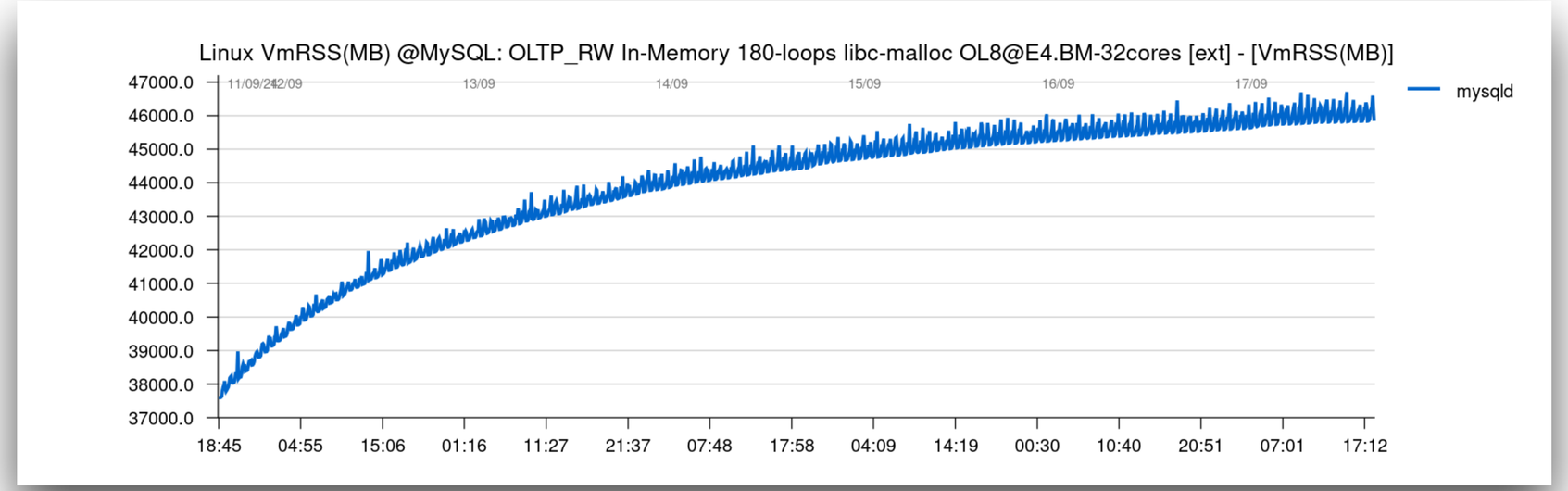


- Range-SELECT
  - SELECT ..
    WHERE id
    BETWEEN $v and $v + 100



Malloc Libs Today

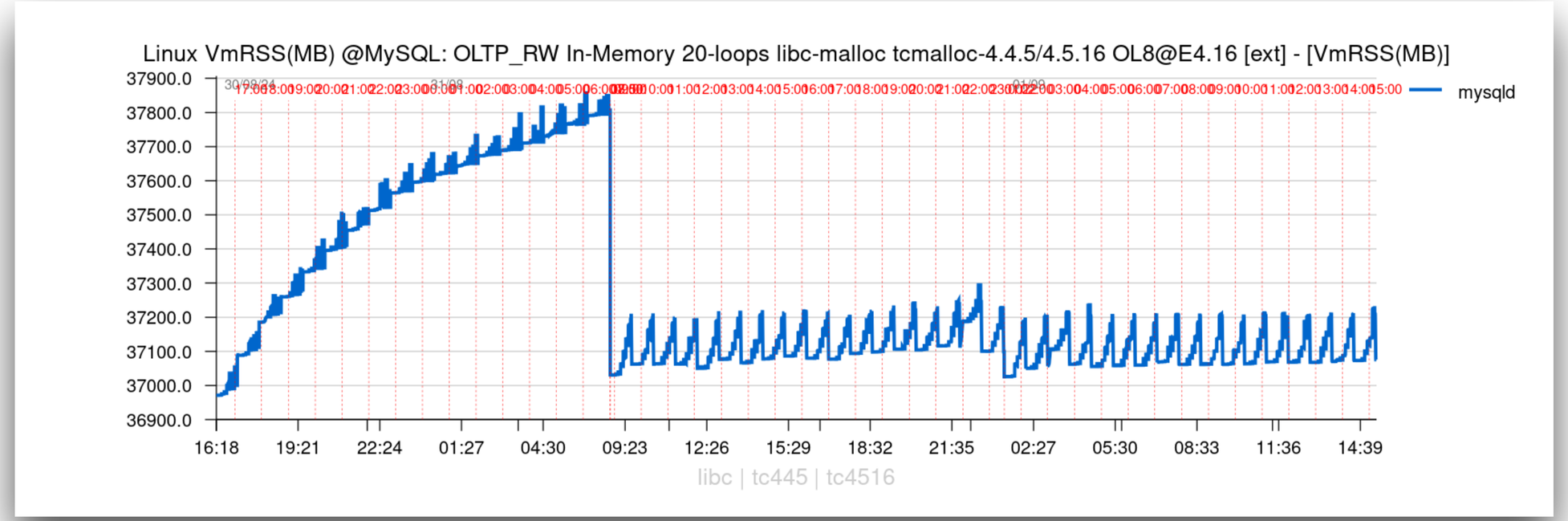
- glibc-malloc :
  - the latest glibc-malloc can be used with MT-apps today in most cases
  - but memory fragmentation !
- glibc-malloc : +10GB RSS usage over 1 week :



  - **NOTE** : Oracle Linux Team is pushing the fix to upstream !!!

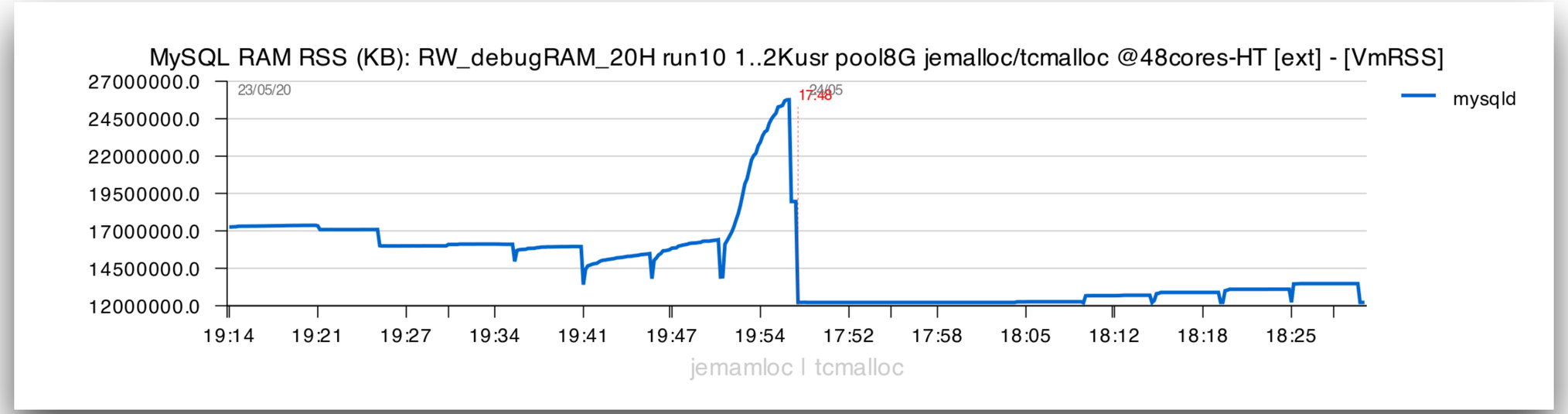
Malloc Libs Today

- glibc-malloc -vs- tcmalloc :



Malloc Libs Today

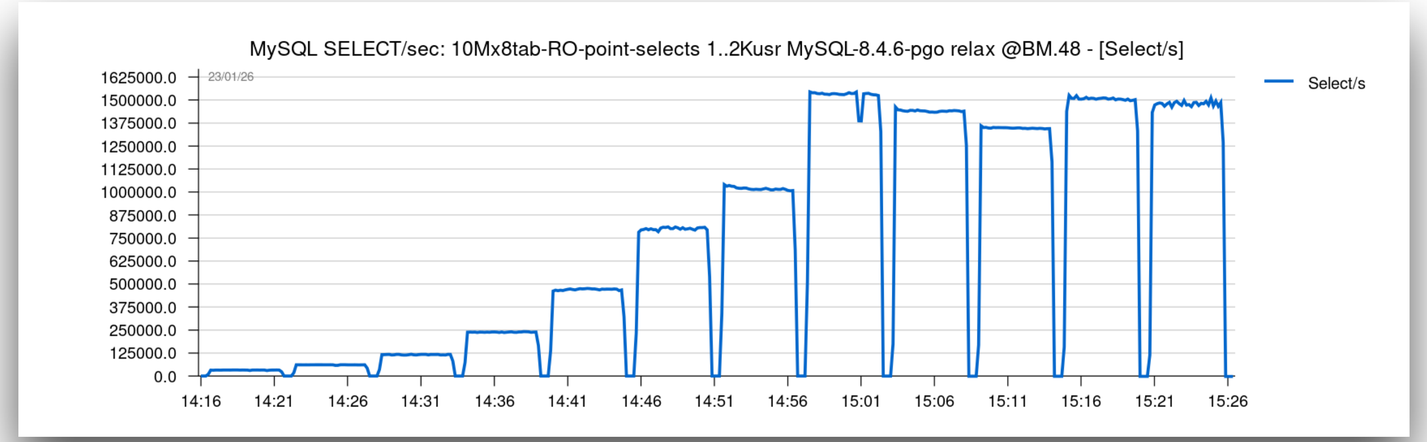
- jemalloc -vs- tcmalloc :



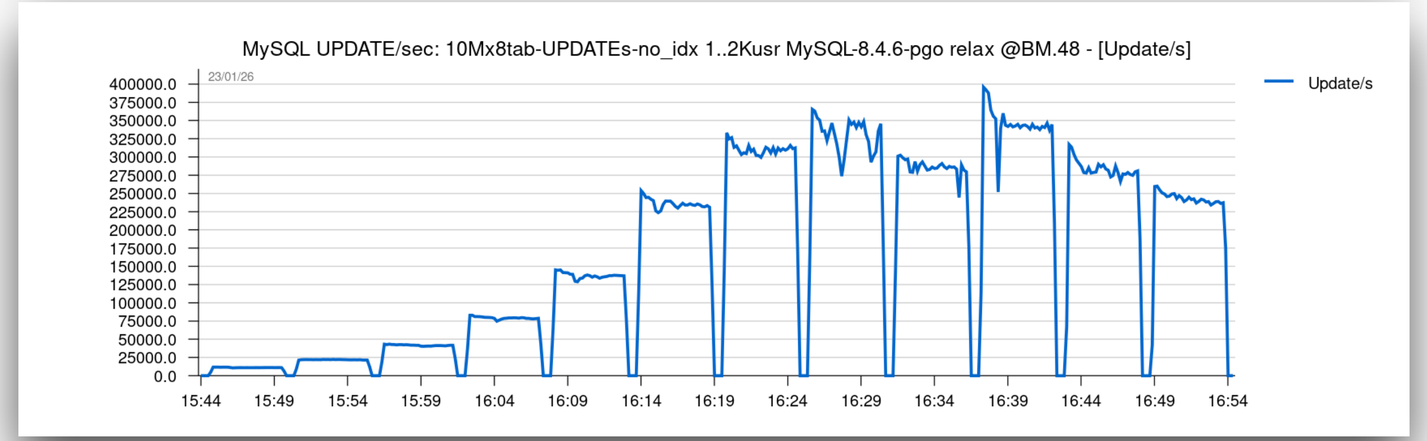
- NOTE :
  - **jemalloc** : EOL no more supported..
  - **tcmalloc** : latest evaluated stable version 4.5.16
    - (still supported within “gperftools” pkg, also shipped with MySQL source code)

CPU-bound : adding Writes

- SELECTs :
  - PK lookups
  - 1.5M QPS peak
  - 800K QPS on 32 users



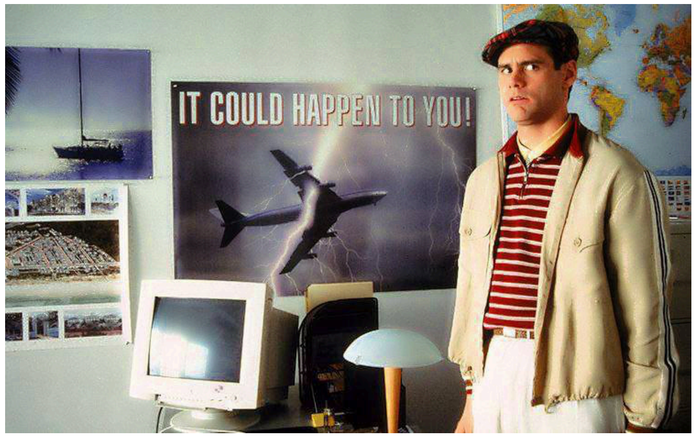
- UPDATEs :
  - non-index updates
  - over 350K QPS
  - over 225K QPS on 32 users

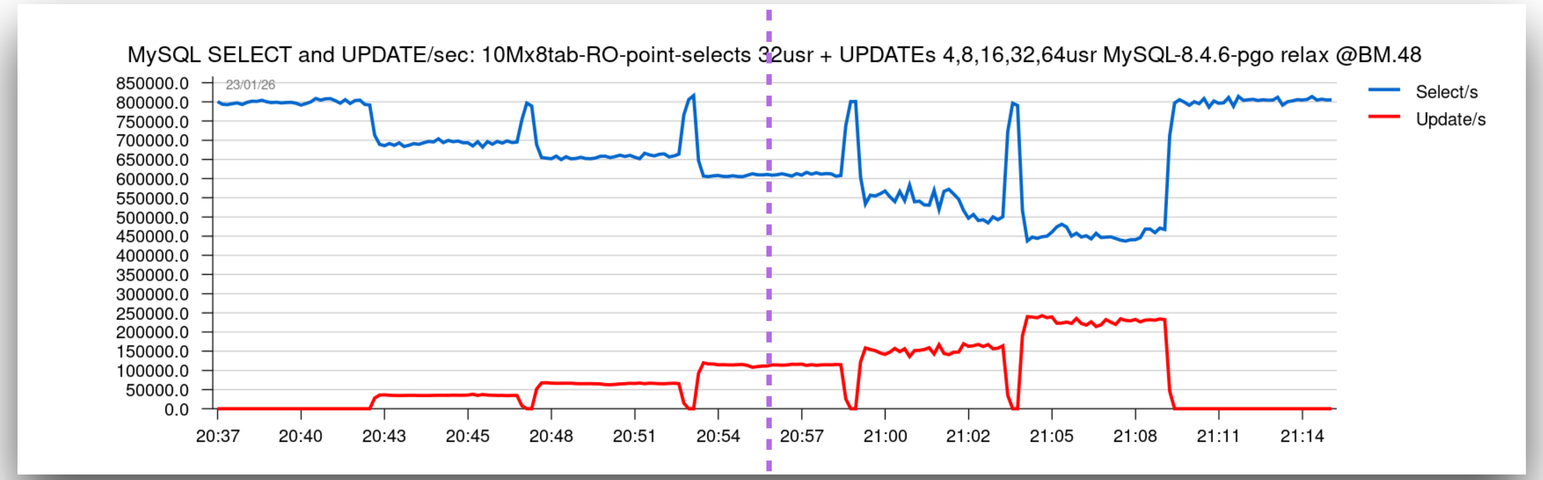


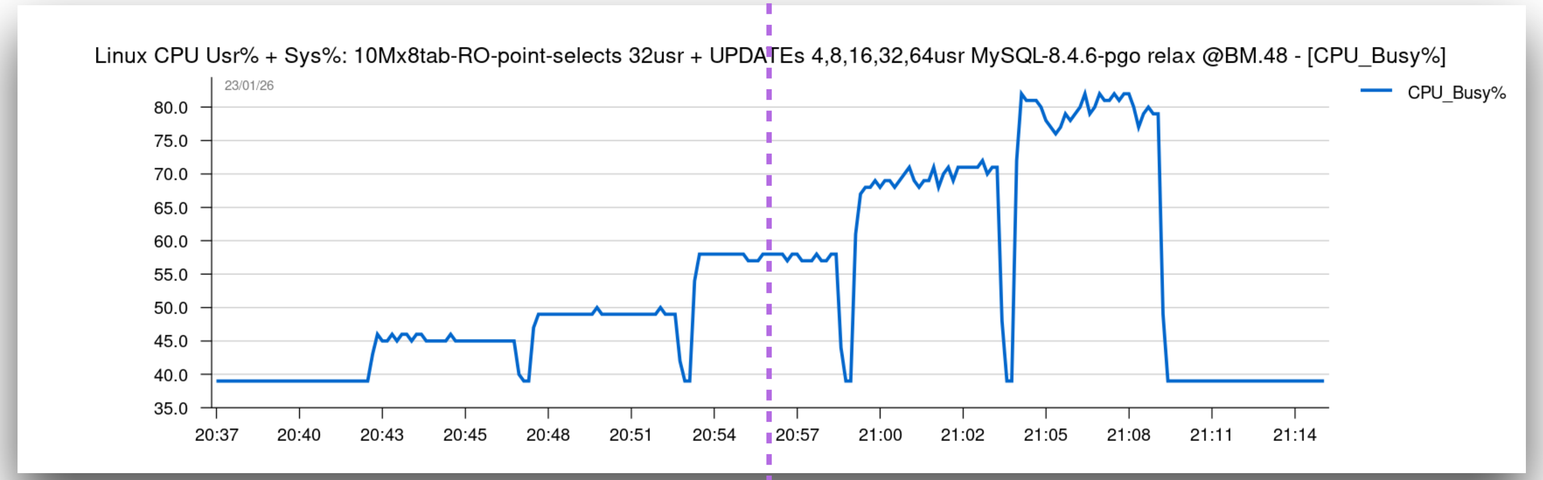
- Question :
  - what will happen if both will run on the same time ?..

SELECTs impacted by UPDATEs..

- SELECTs :
  - 32 users
  - running non-stop
- UPDATEs :
  - growing load
  - starting with 4 users
  - then 8, 16, 32, 64



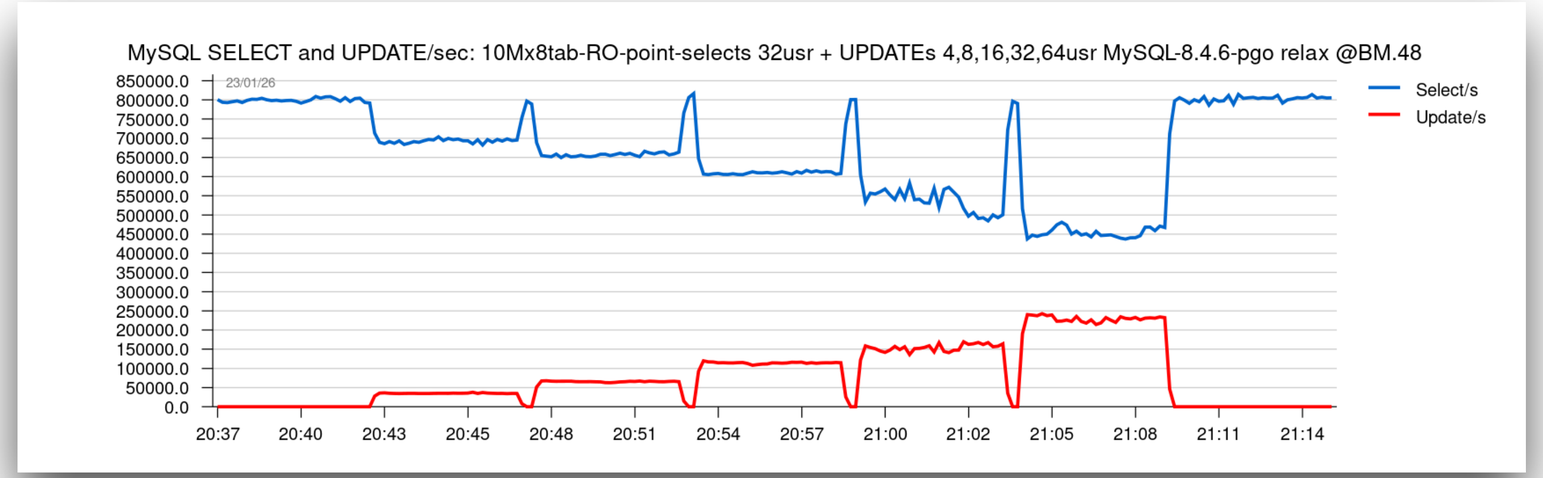


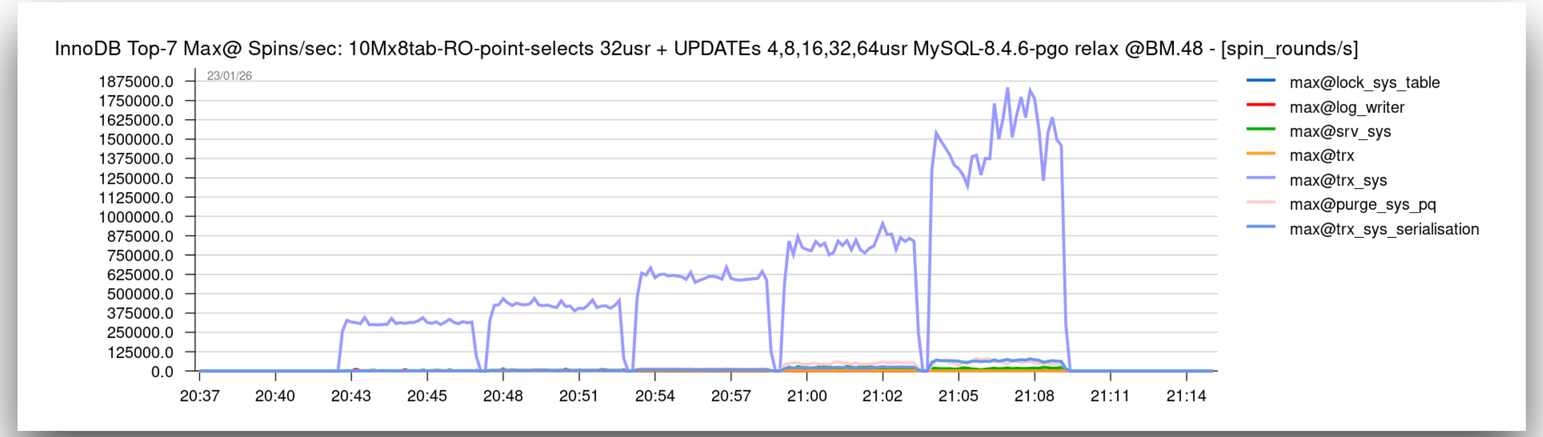


ORACLE

SELECTs impacted by UPDATEs..

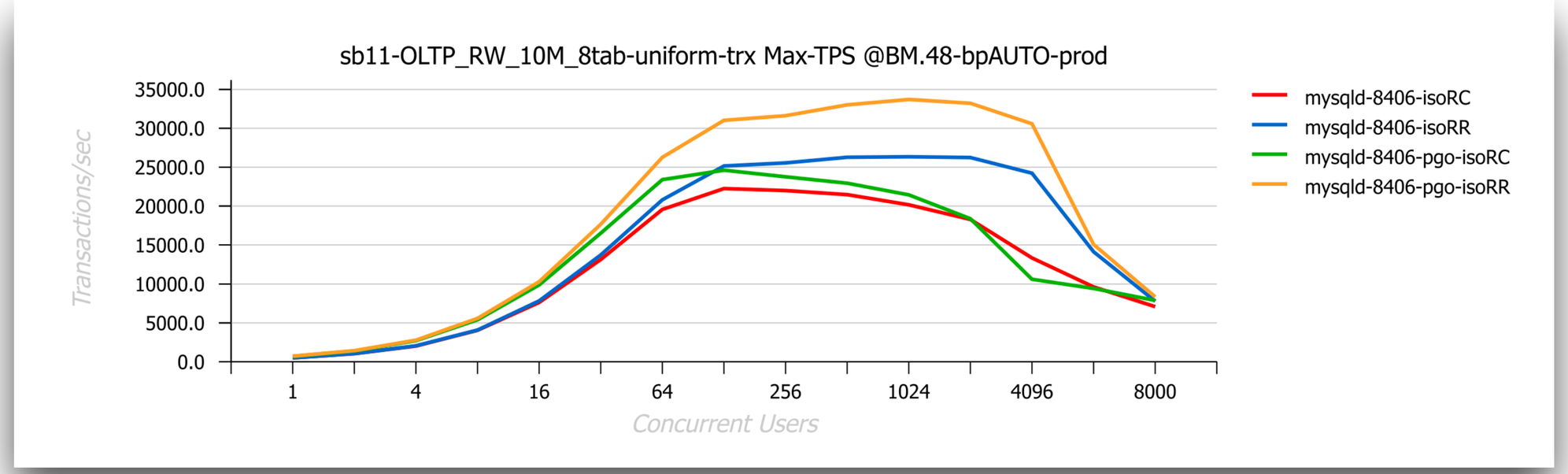
- SELECTs :
  - 32 users
  - running non-stop
- UPDATEs :
  - growing load
  - starting with 4 users
  - then 8, 16, 32, 64
- ReadView overhead
  - absent on RO (since 5.7)
  - present on RW
  - spin-rounds on trx_sys





ReadView & Transaction Isolation : RR or RC ?..

- REPEATABLE-READ (RR) used by default
  - ReadView is created **only** on BEGIN of transaction
- COMMITTED-READ (RC)
  - ReadView is created for **every** query !



- **NOTE** : work in progress to make RC as good as RR !

MySQL Config

- since MySQL 8.4 : **auto-adaptive** default config !!
- e.g. just use default config + InnoDB dedicated = ON
  - auto-adaptive : BP size, BP instances, Page Cleaners, REDO size, etc..
  - by default : AHI = off, IBUF = 5, DBLWR = 2x128, O_DIRECT, etc..
- example of your new my.conf :

```
[mysqld]
 innodb_dedicated_server = on
 max_connections = 4000
```

- data safety & security first !!!
  - trx_commit = 1, sync_binlog = 1
  - **NOTE** : and this is **mandatory** !! even if you use multi-nodes GR/Replication !!
  - (except if you don't care about your data and ok to loose them..)

NOTE to PeterZ :
please,
update your slides !!

REDO Log

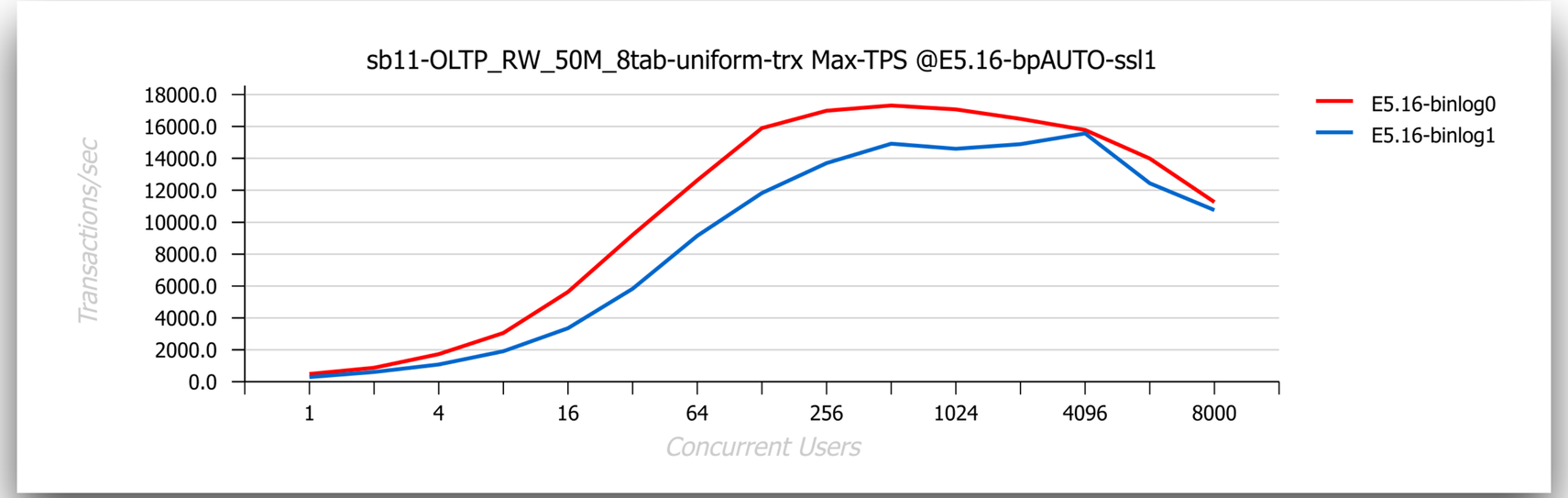
- Storage fsync latency
  - COMMIT => fsync
  - ex. : 1ms fsync latency = 1K COMMIT/s for single-user
  - InnoDB METRICS : log_flush_avg_time, log_flush_max_time (us)
  - check Storage :

```
$ fio --ioengine=psync --blocksize=512 --fsync=1 –rw=write \
 --numjobs=1 --iodepth=1 --name=TEST --filename=/path/to/file \
 --size=256m --runtime=60
```

- REDO log size
  - auto-defined according to your VCPUs number
  - online resize on-demand !
- REDO log threads
  - if Binlog=on : disable for less 16vcpu, enable if more
  - if Binlog=off : disable for less 4vcpu, enable if more
  - NOTE : auto-tuned since MySQL 9.x

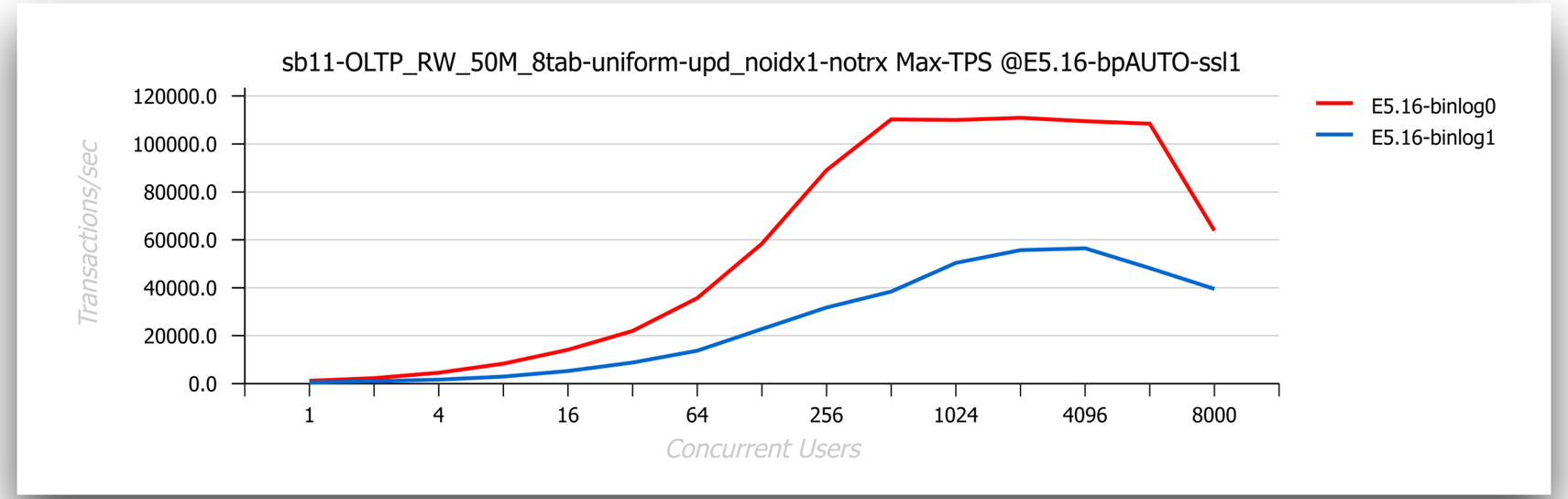
Binlog Overhead

- “By design” :
  - keeping only “**committed**” data !
  - do nothing until InnoDB COMMIT (e.g. no any parallel work)
  - so, every WRITE request is **twice** longer (“in theory”)
  - impact : directly depends on the READs / WRITEs ratio in your transaction
  - ex. : 14 SELECTs : 4 WRITEs => up to **30% lower** TPS :

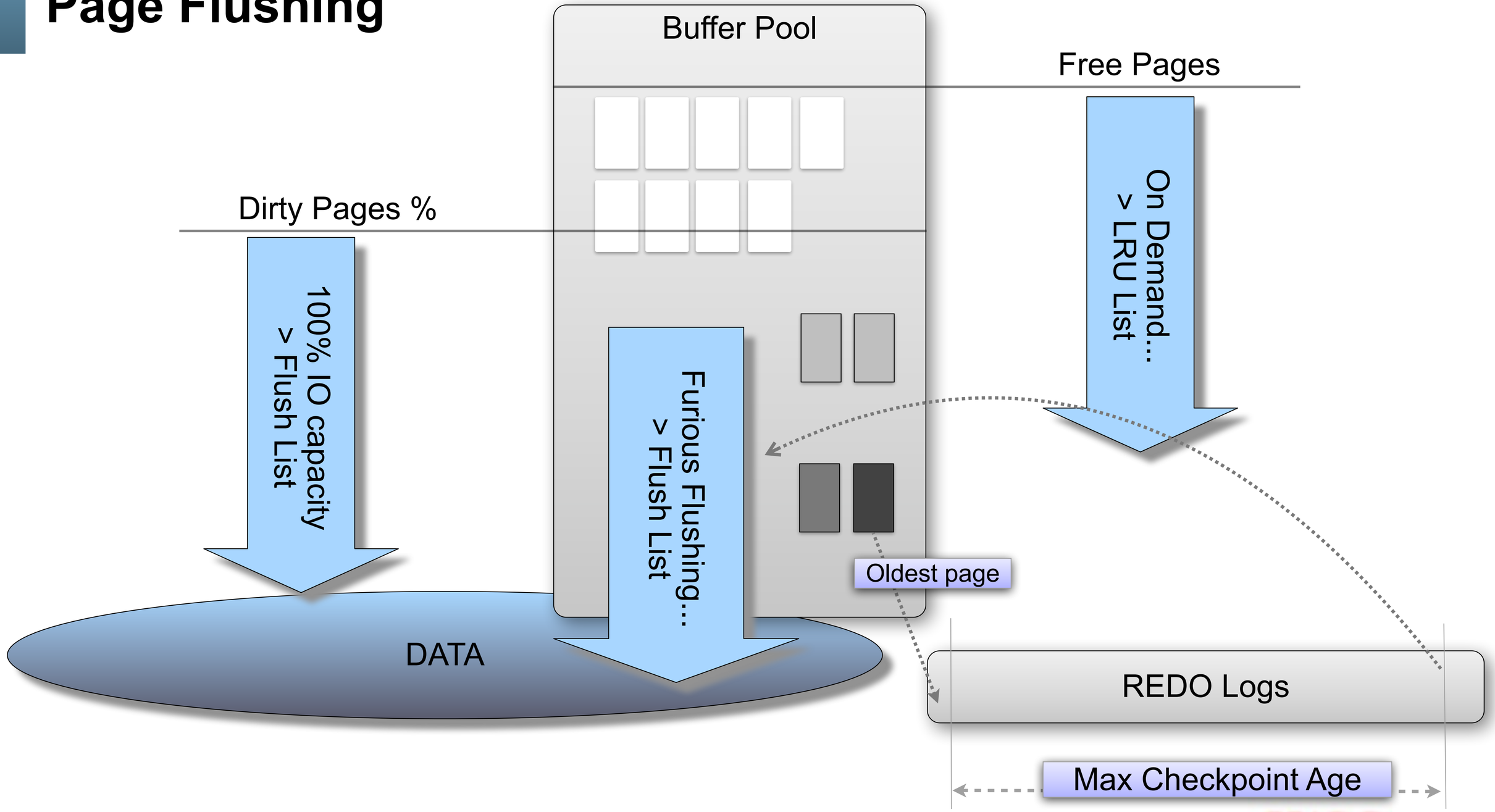


Binlog Overhead

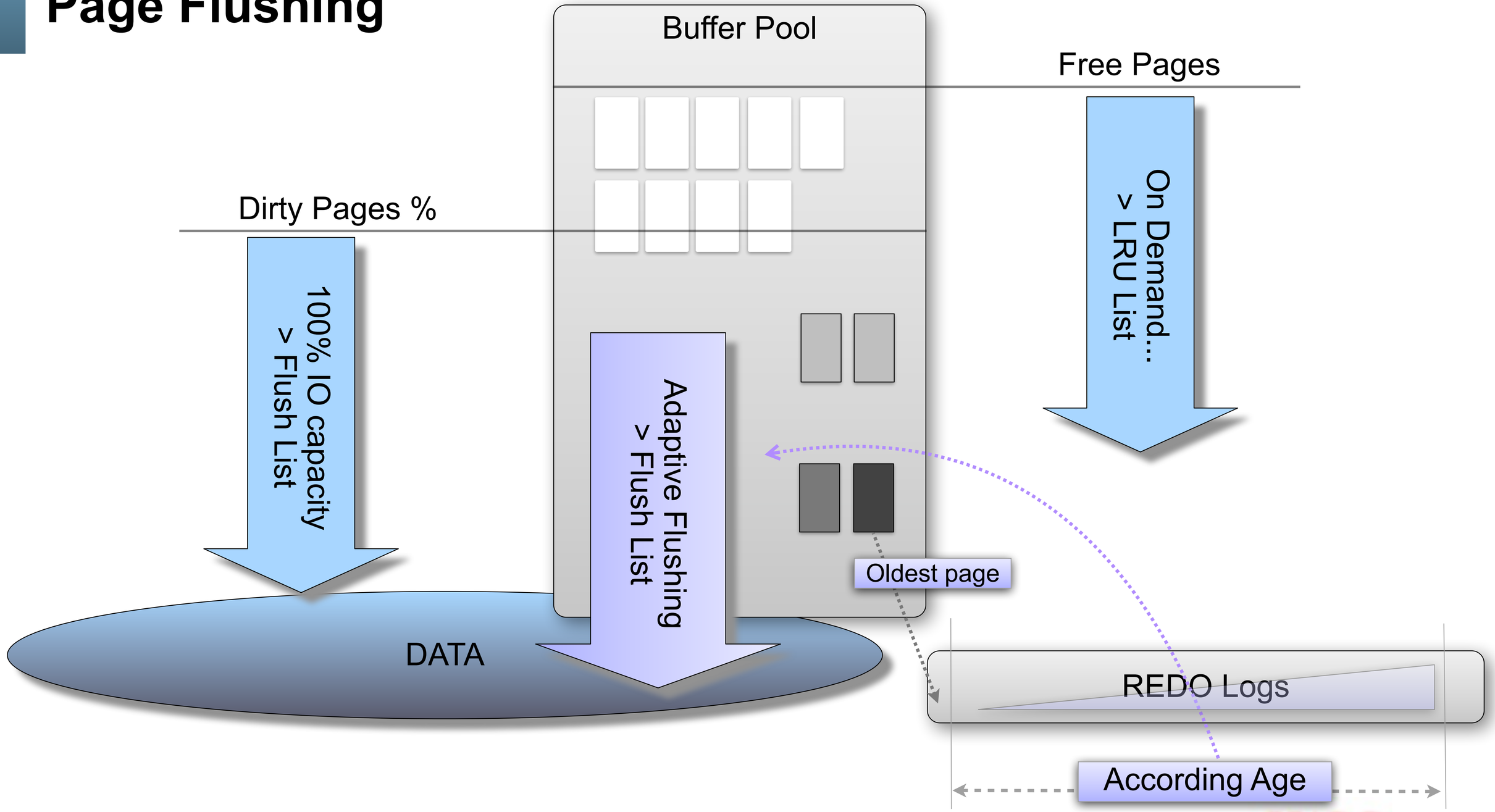
- “By design” :
  - keeping only “**committed**” data !
  - do nothing until InnoDB COMMIT (e.g. no any parallel work)
  - so, every WRITE request is **twice** longer (“in theory”)
  - impact : directly depends on the READs / WRITEs ratio in your transaction
  - ex. : WRITEs only (UPDATEs) => up to **x3 times lower** TPS :



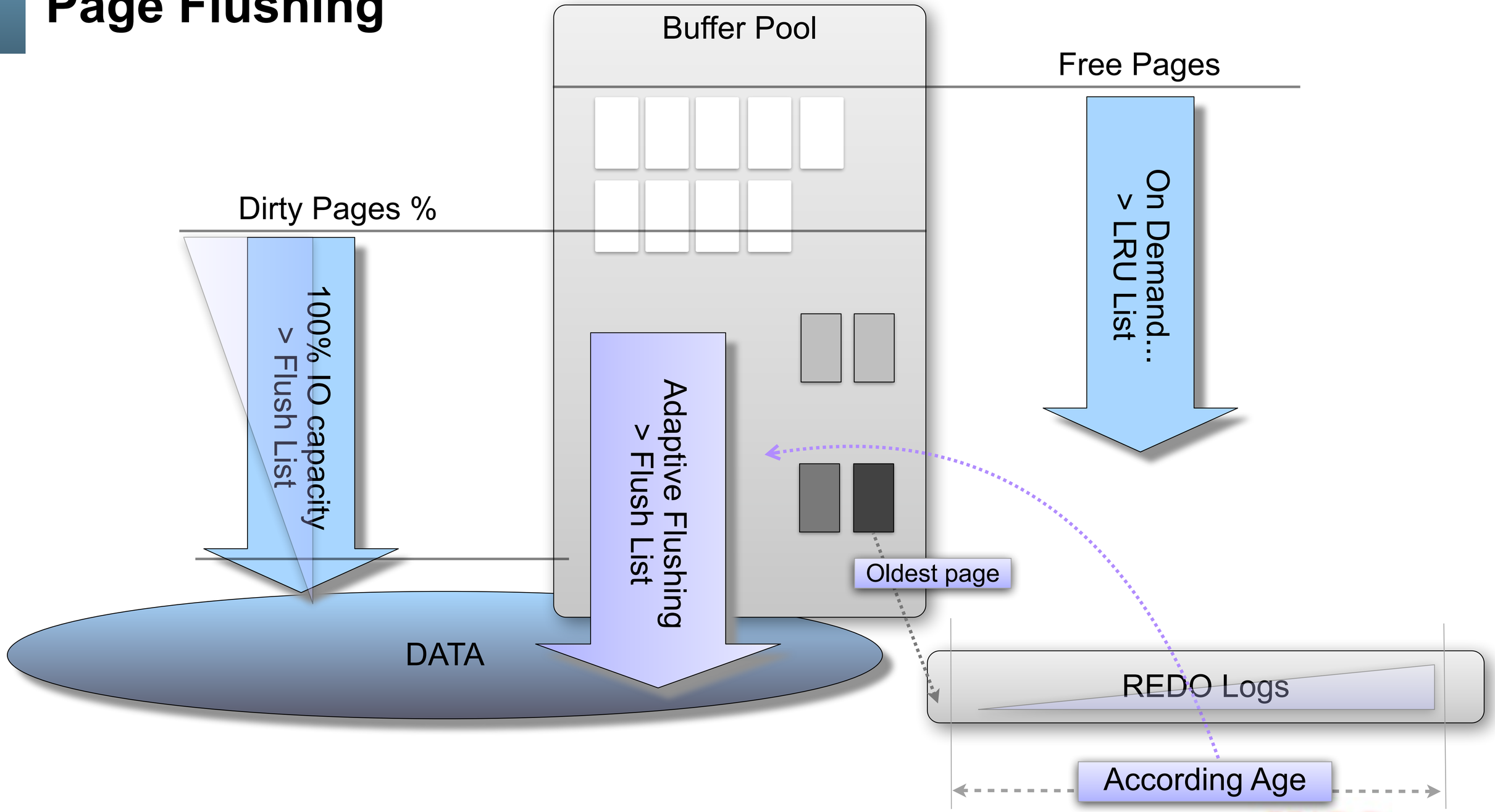
Page Flushing



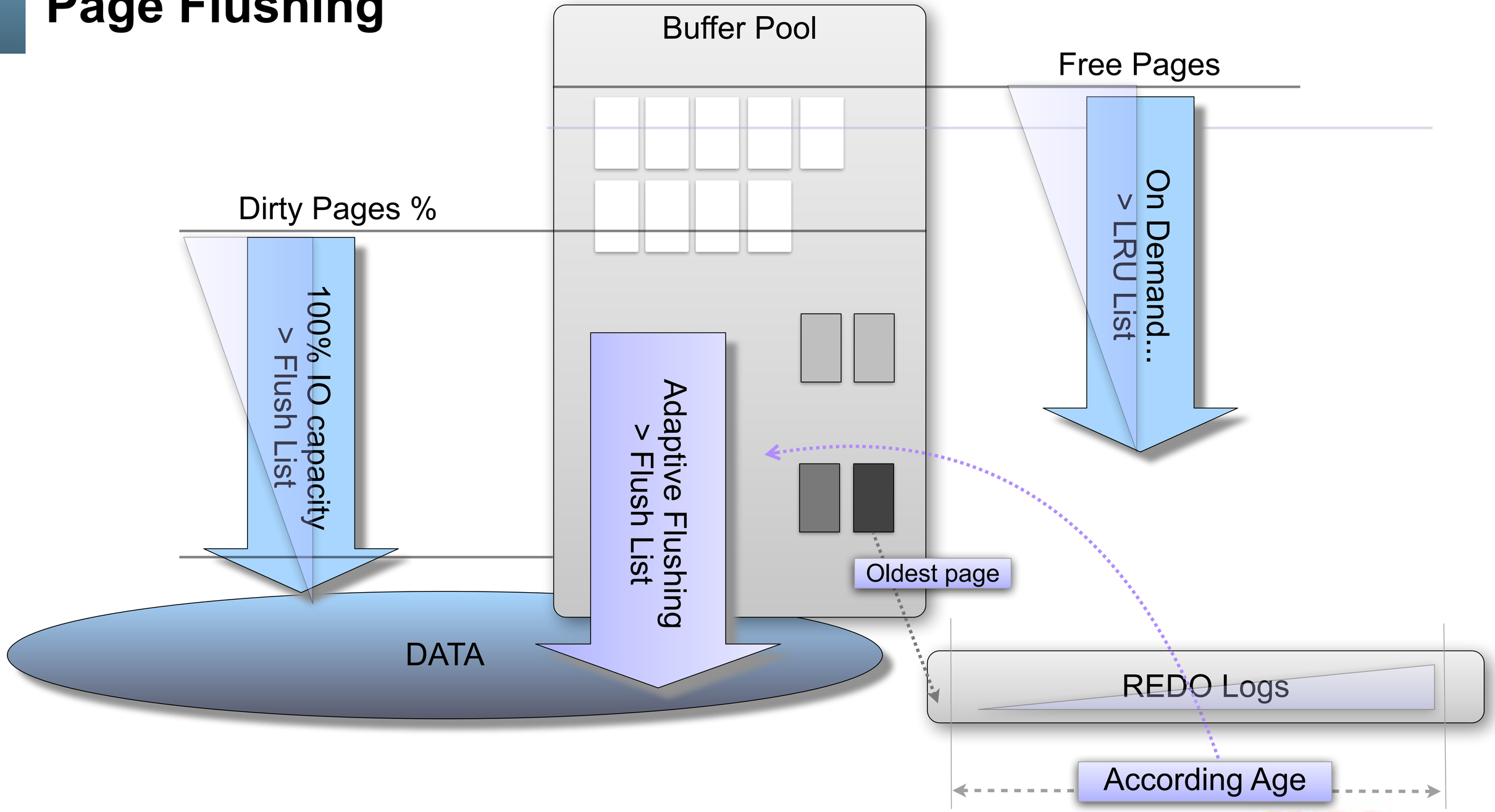
Page Flushing



Page Flushing

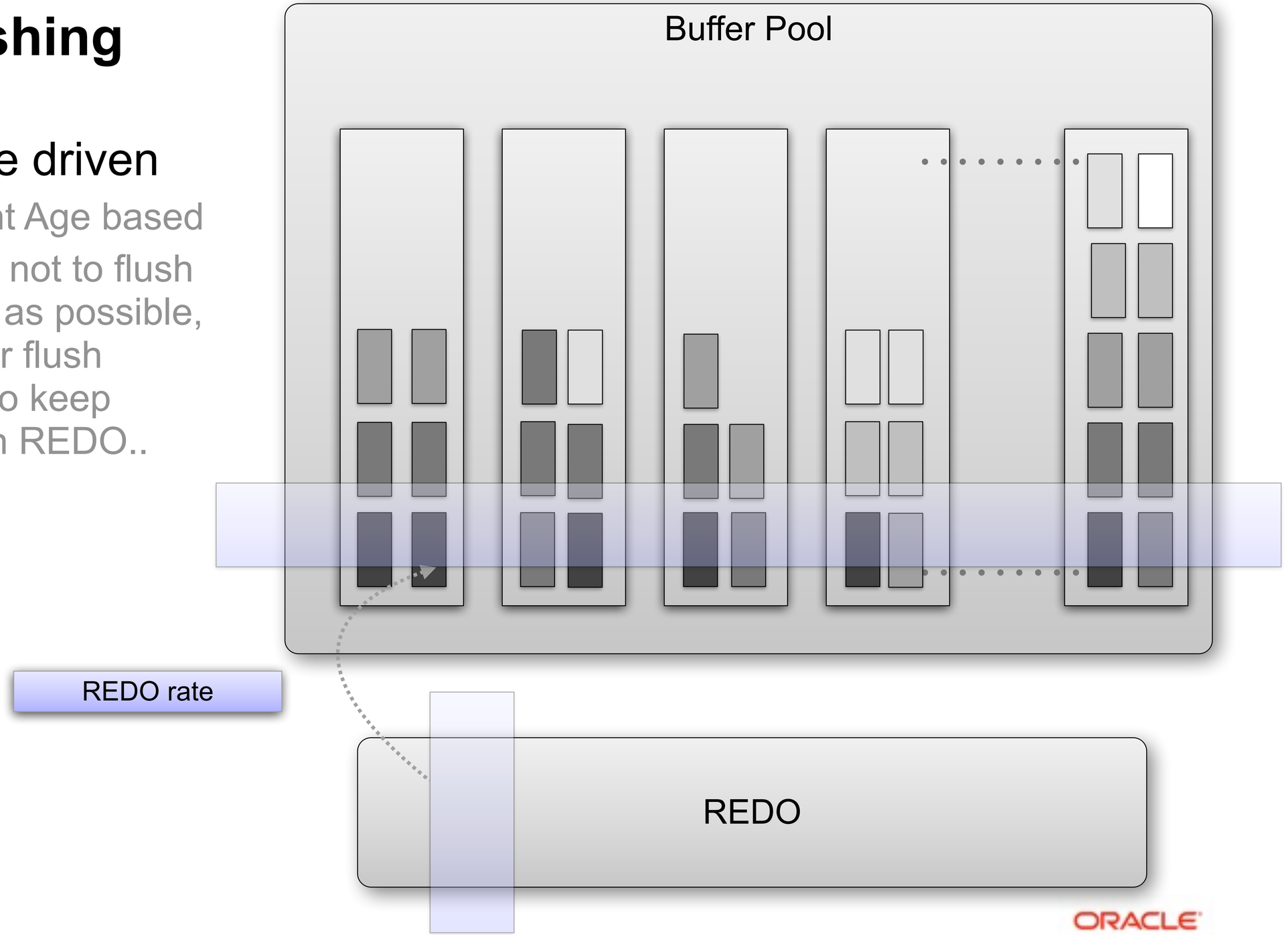


Page Flushing



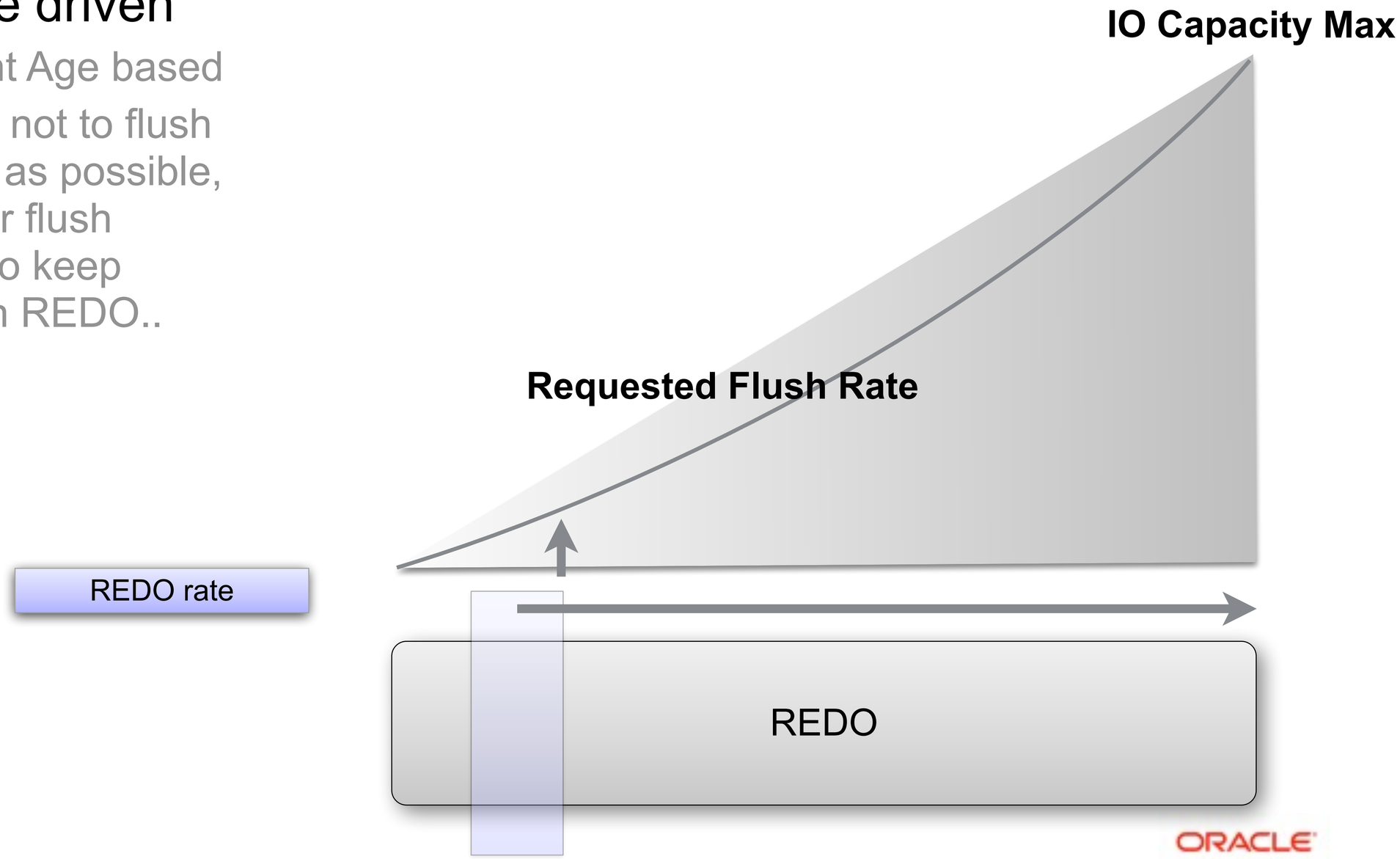
Page Flushing

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



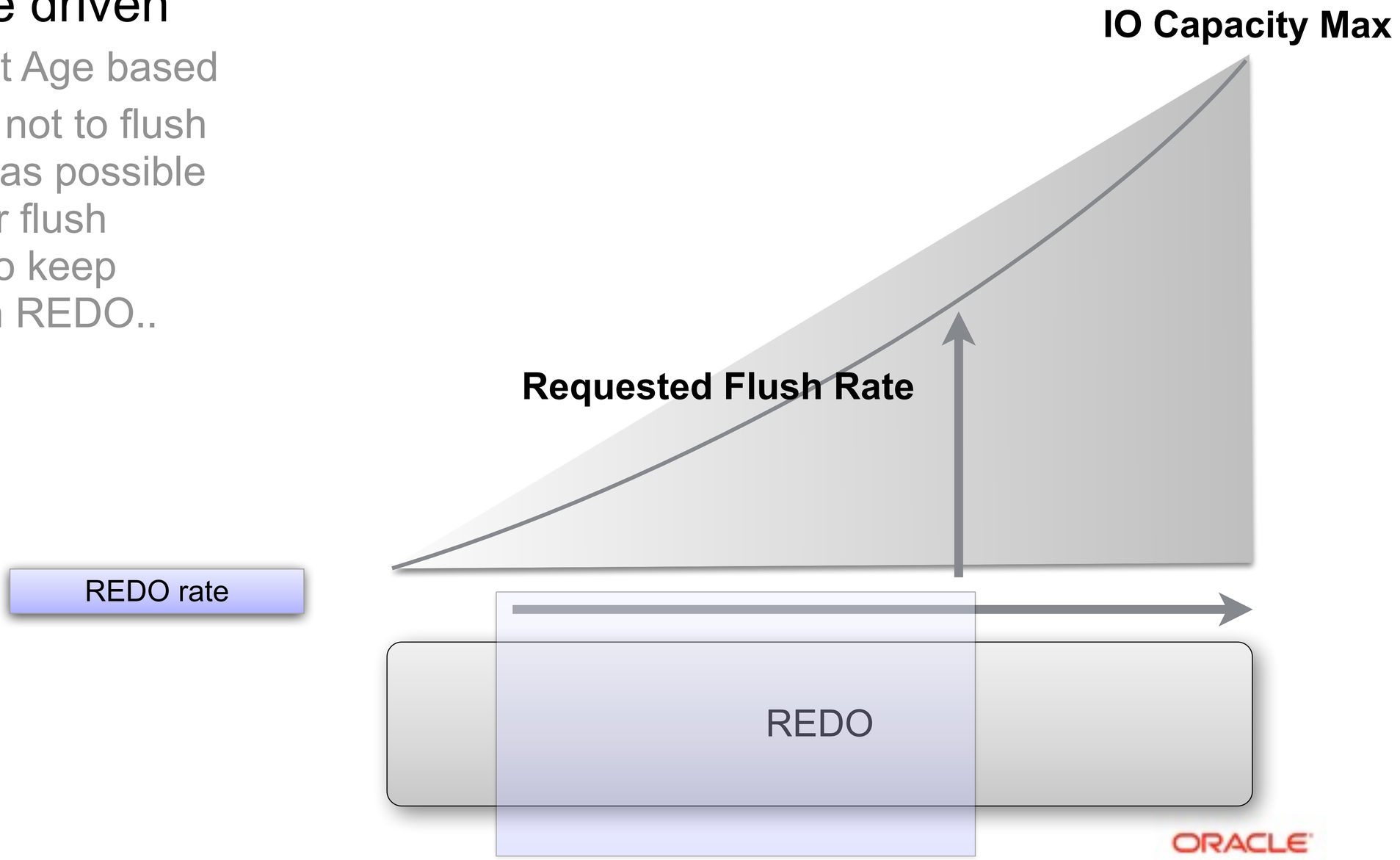
Page Flushing

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



Page Flushing

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

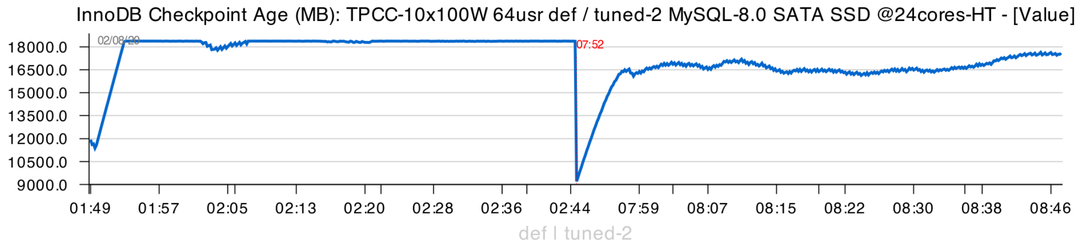


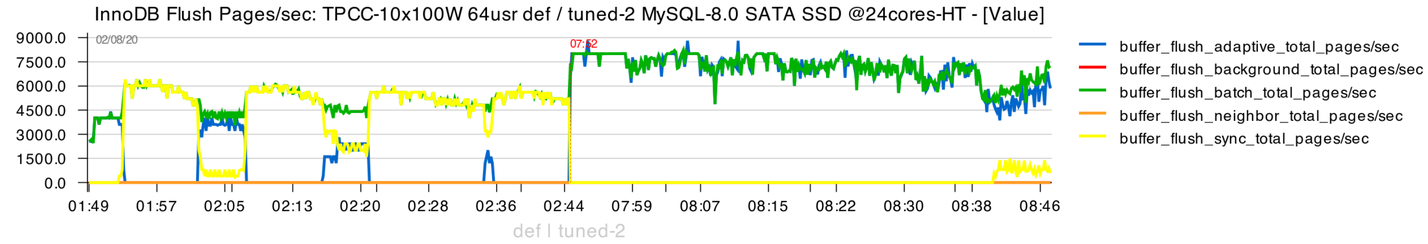
InnoDB Page Flushing : Tuning via Monitoring

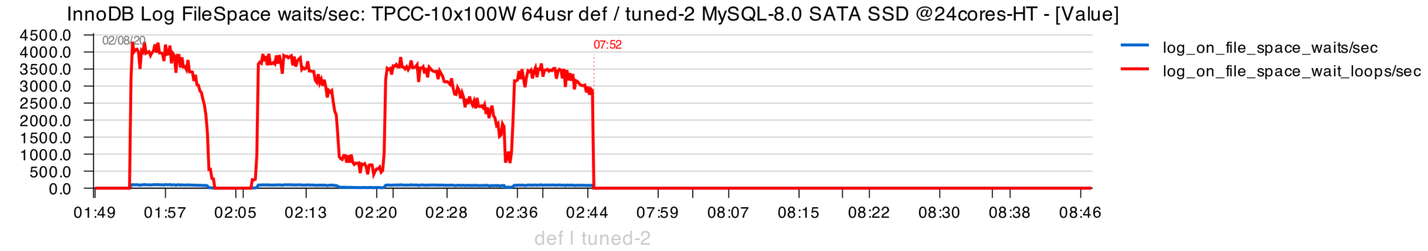
- Monitoring InnoDB METRICS :
  - Checkpoint Age < REDO total size
  - log_on_file_space_wait_loops/sec == 0
  - buffer_flush_sync_waits && buffer_flush_sync_pages == 0
  - buffer_flush_avg_time < 1sec
  - buffer_flush_adaptive_avg_pass == 30 (def. avg loops)
  - buffer_flush_adaptive_total_pages/sec ~= buffer_flush_n_to_flush_requested
    - this will tell if “IO capacity max” should be increased or not
- If Page Flushing is still lagging :
  - check your IO capacity max setting (can be changed **online**)
  - you can try higher number of BP Instances & Page Cleaners (need restart)
  - finally, increase REDO log size (can be changed **online** now)

InnoDB Page Flushing : Tuning via Monitoring

- Monitoring InnoDB METRICS :







InnoDB Page Flushing : Other Impacts

- DBLWR :
  - every Page is written twice => twice higher Page Write times !
  - if your Storage is saturated => may require additional tuning
    - since 8.4 default is : 2 files x 128 pages
    - if you use RAID on top of your Storage : use stripe size 128K or 256K
  - Oracle Linux Team => Atomic IO !!! (already pushed to upstream)
    - MySQL code : mostly ready
- Compression :
  - table compression : “in theory” may help
  - “transparent” compression (using punch holes) : DON’T USE ! (more later ;-))

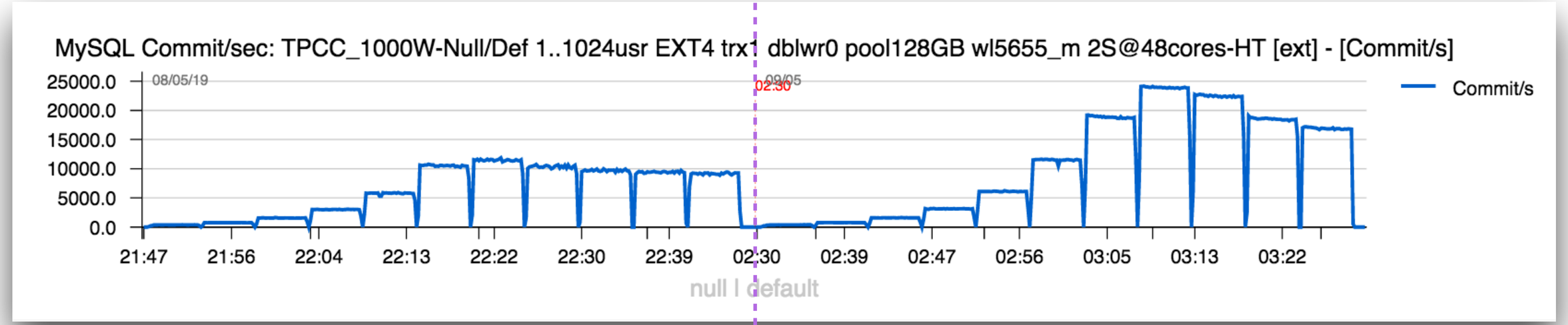
Using NULL -vs- DEFAULT

- When you use NULL as default value :
  - UPDATE NULL to real Value => Row size increase => Page split..
  - Page split => Lock contentions => slow down processing..
  - Page split => cascade increase of Dirty Pages => higher Page Flushing => stalls ?..

- More efficient approach :
  - instead of NULL use DEFAULT (and assign a “good size” value to DEFAULT)
  - then : INSERT values(…, DEFAULT, …) /* instead of NULL */
  - then : UPDATE … var = DEFAULT … /* instead of NULL */
  - then : SELECT … WHERE var != DEFAULT /* instead of not NULL */
  - and so on.. => no more Page split (or very minimal)

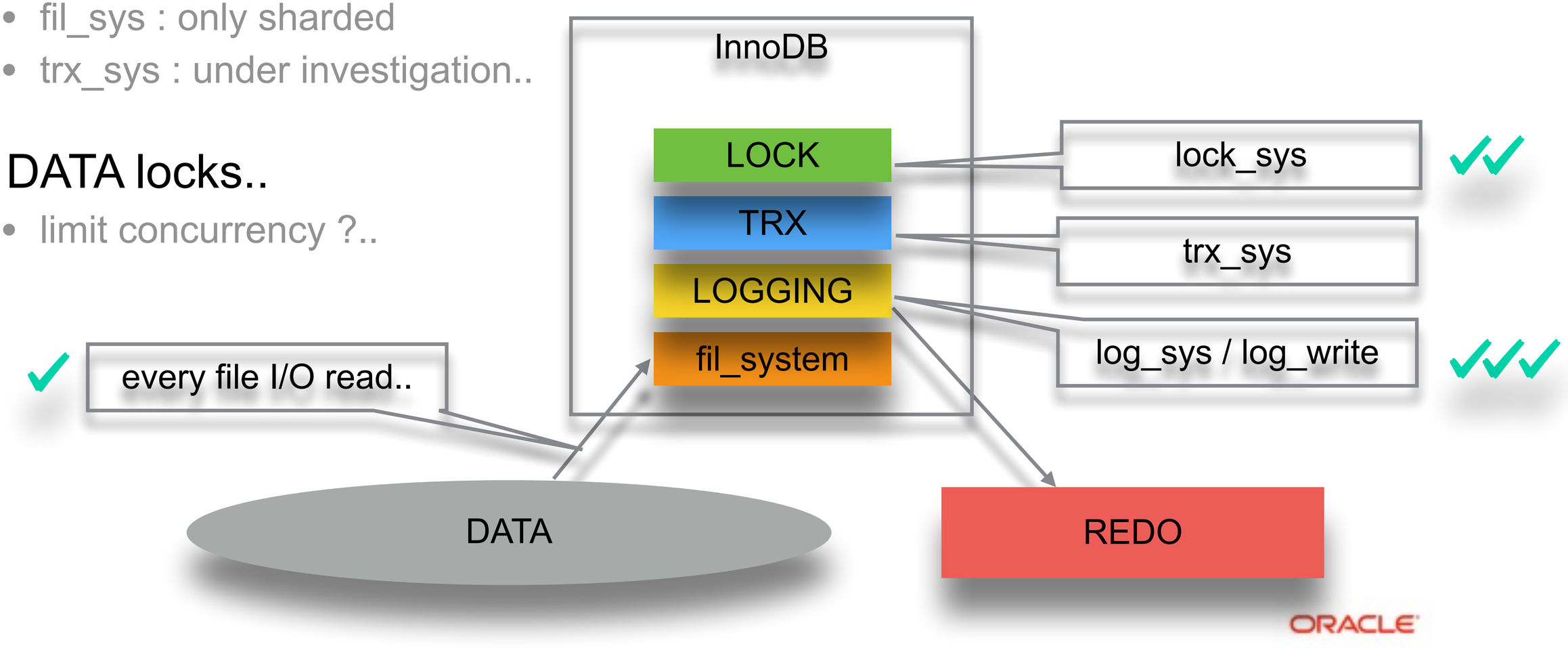
TPCC “mystery” : test results

- Sysbench-TPCC 1000W
  - NULL -vs- DEFAULT



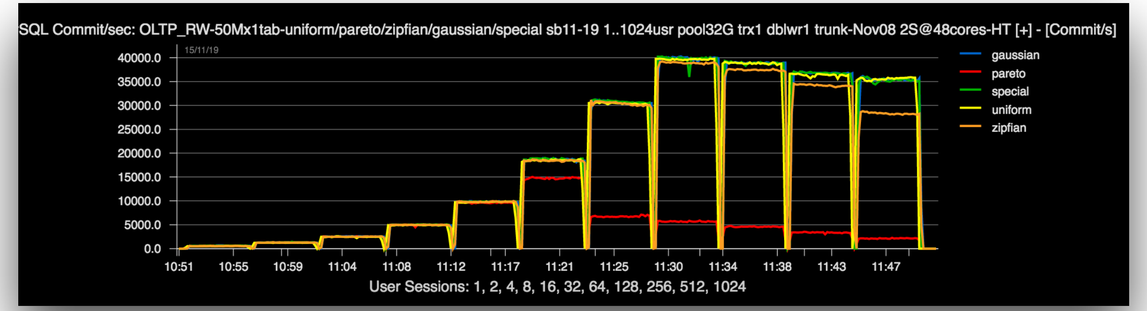
LOCK Contentions

- Many improvements since 8.0
  - log_sys : gone !!!
  - lock_sys : largely improved !!
  - fil_sys : only sharded
  - trx_sys : under investigation..
- DATA locks..
  - limit concurrency ?..



DATA Locking

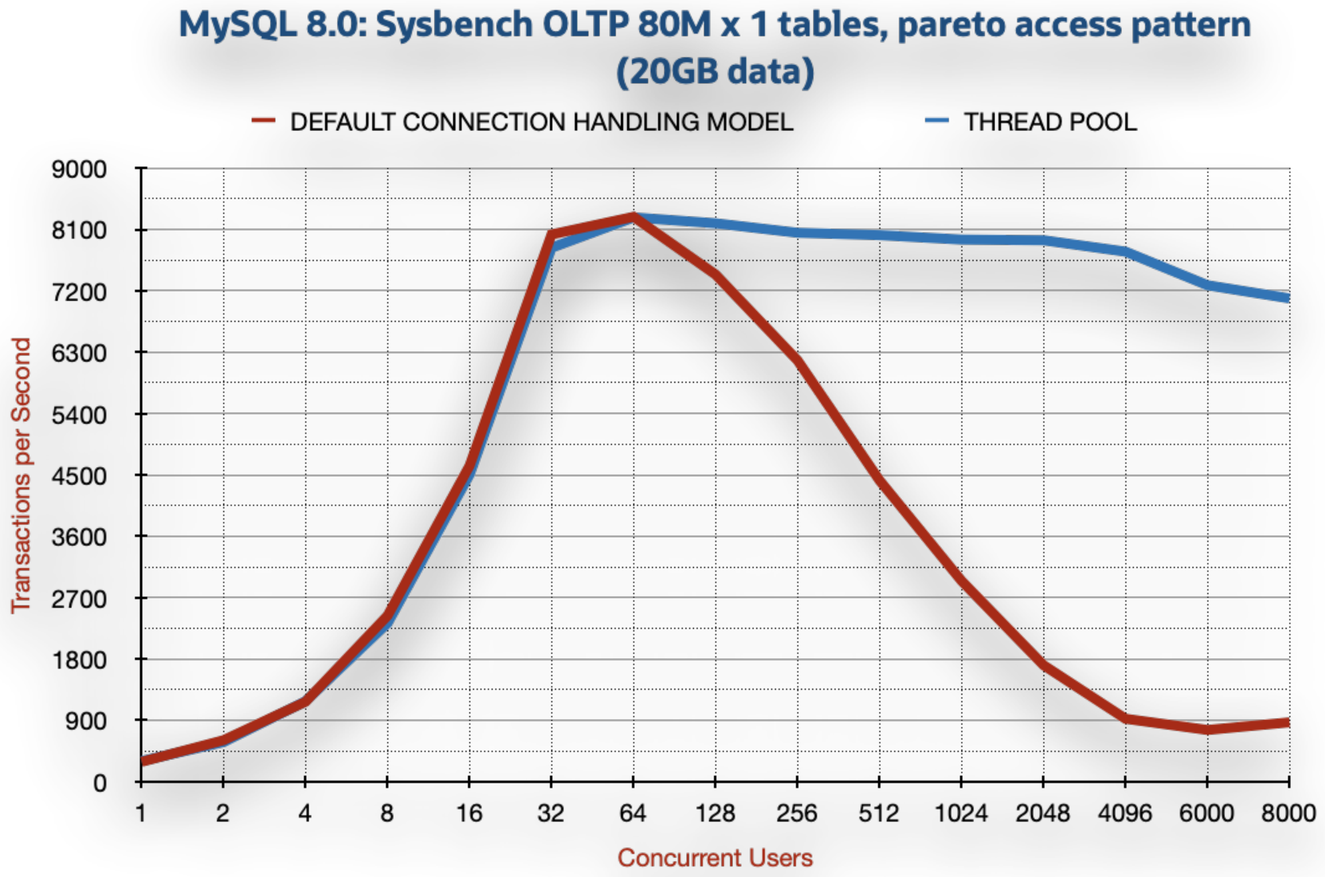
- DATA concurrent X-locking overload => TPS decrease
- InnoDB Thread Concurrency :
  - (+) historically most commonly advised solution
  - (+) limits number of concurrent user threads
  - (-) not considering thread's IO waits
  - (-) not considering transaction locking
  - (-) amplifies lock waits
  - (-) amplifies "deadlocks feeling" states..



- Most safe solution : Limit ~~Thread~~ **Transaction** Concurrency !
  - complicate to implement on InnoDB level
  - currently working solution : **New** MySQL ThreadPool

New MySQL ThreadPool

- Max Transaction Limit (MTL)
  - can be adapted online according to your workload



ref : https://blogs.oracle.com/mysql/the-new-mysql-thread-pool

InnoDB UNDO Logs & Purge

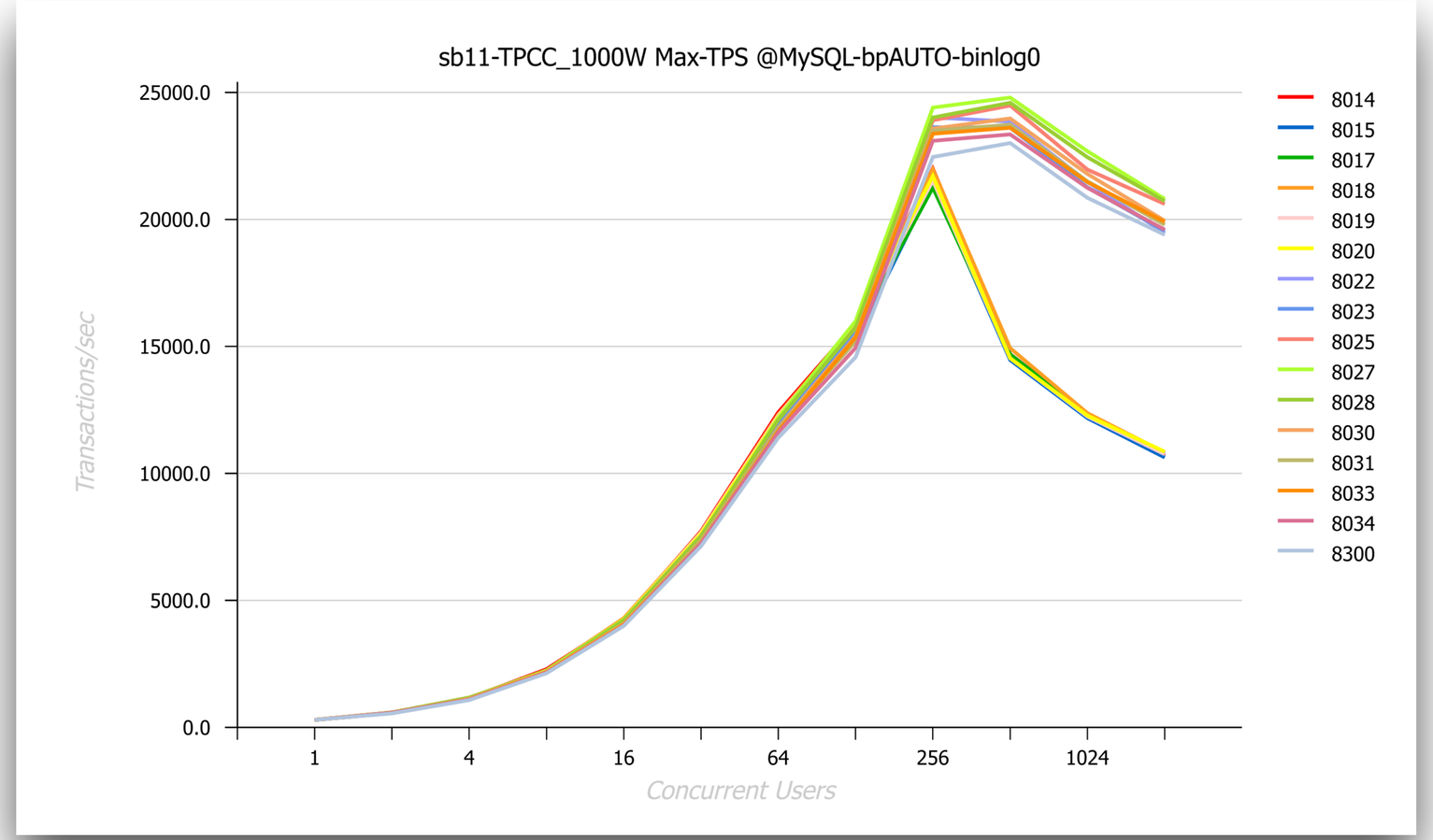
- InnoDB UNDO :
  - also using BP, can grow very quickly ! (performance / disk space / overall impact)
  - can be dropped manually / automatically (do with caution)

- InnoDB Purge :
  - cleaning-up UNDO records, but can be lagging
  - monitor InnoDB History Length **ALWAYS !**
    - if NO purge lagging : excellent! (& be happy! ;-))
    - if purge is lagging : use a purge lag config setting.. (write throttling)

  - **Example of config settings to avoid purge lagging :**
    - innodb_max_purge_lag = 1000000 (1M max for ex.)
    - innodb_max_purge_lag_delay = 30000 (30ms max for ex.)
    - innodb_purge_threads = 4 (check yourself if using more threads is helping)

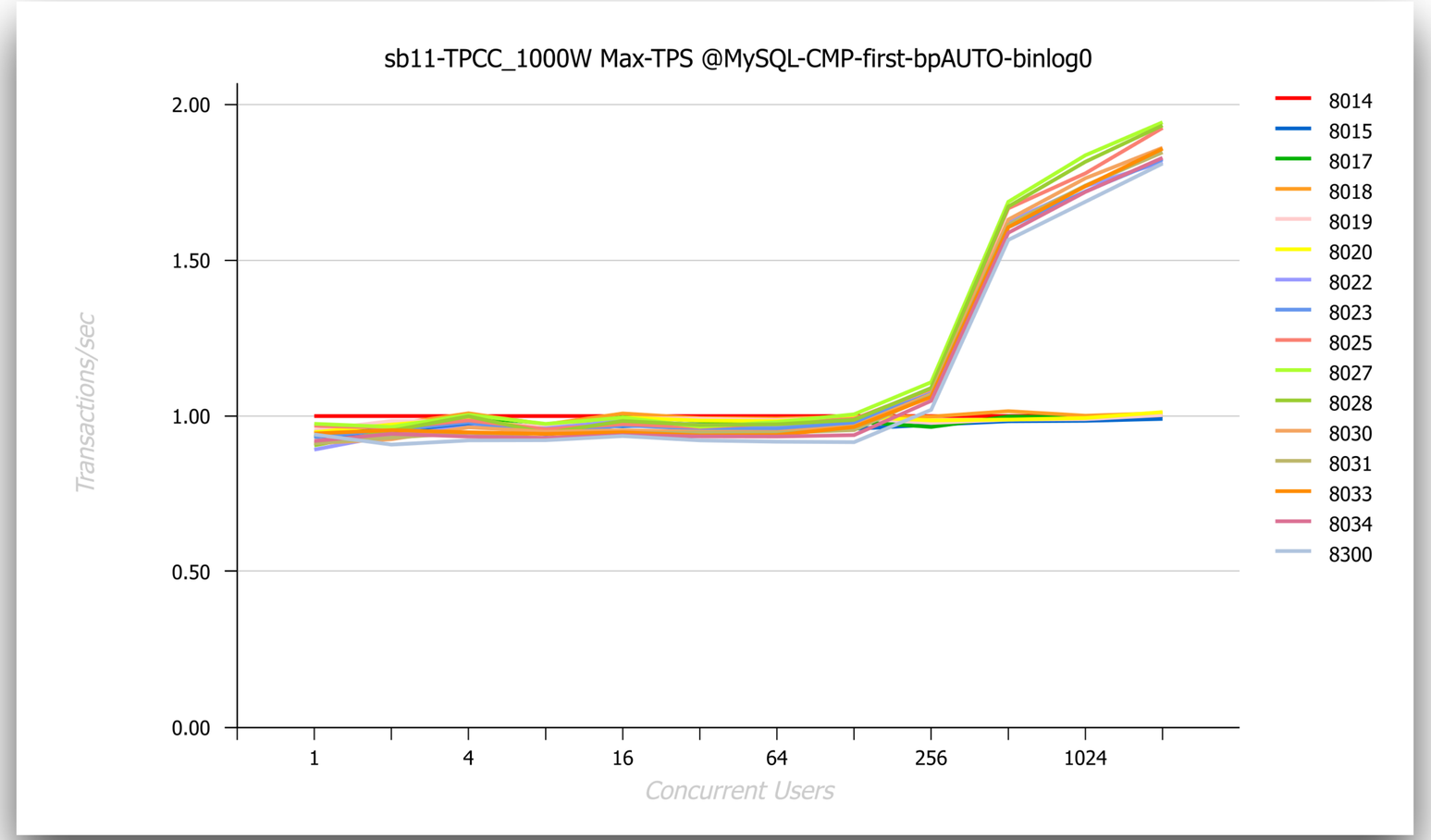
MySQL Regressions.. (in case you could think we don't care)

- TPCC example..
  - comparing TPS between 5 years back releases : clear regression on 8.3



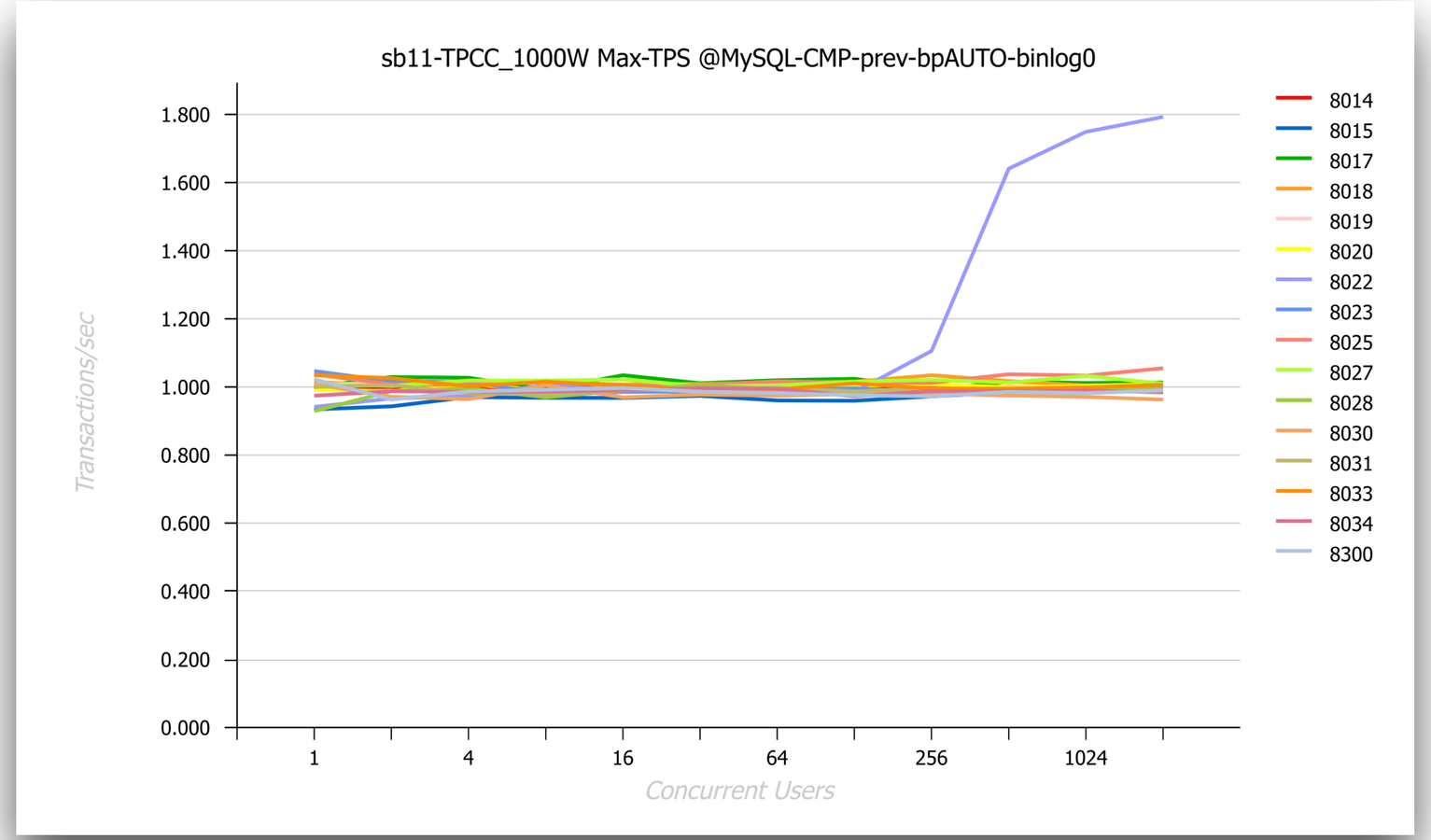
MySQL Regressions.. (in case you could think we don't care)

- TPCC example..
  - comparing relative ratio with 8.0.14 : up to 10% difference..



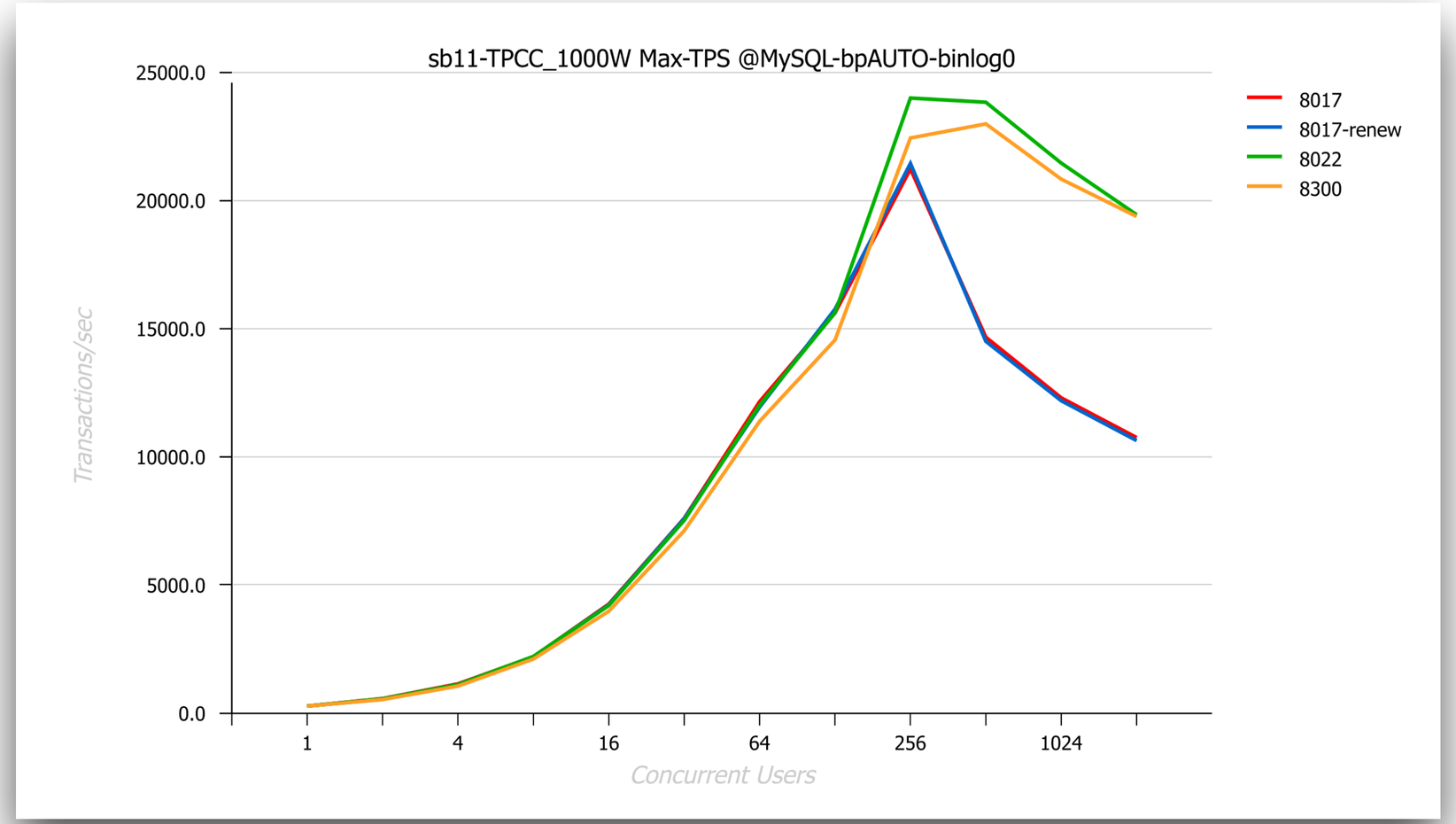
MySQL Regressions.. (in case you could think we don't care)

- TPCC example..
  - relative ratio for given release to its previous release : **less** 3% difference..



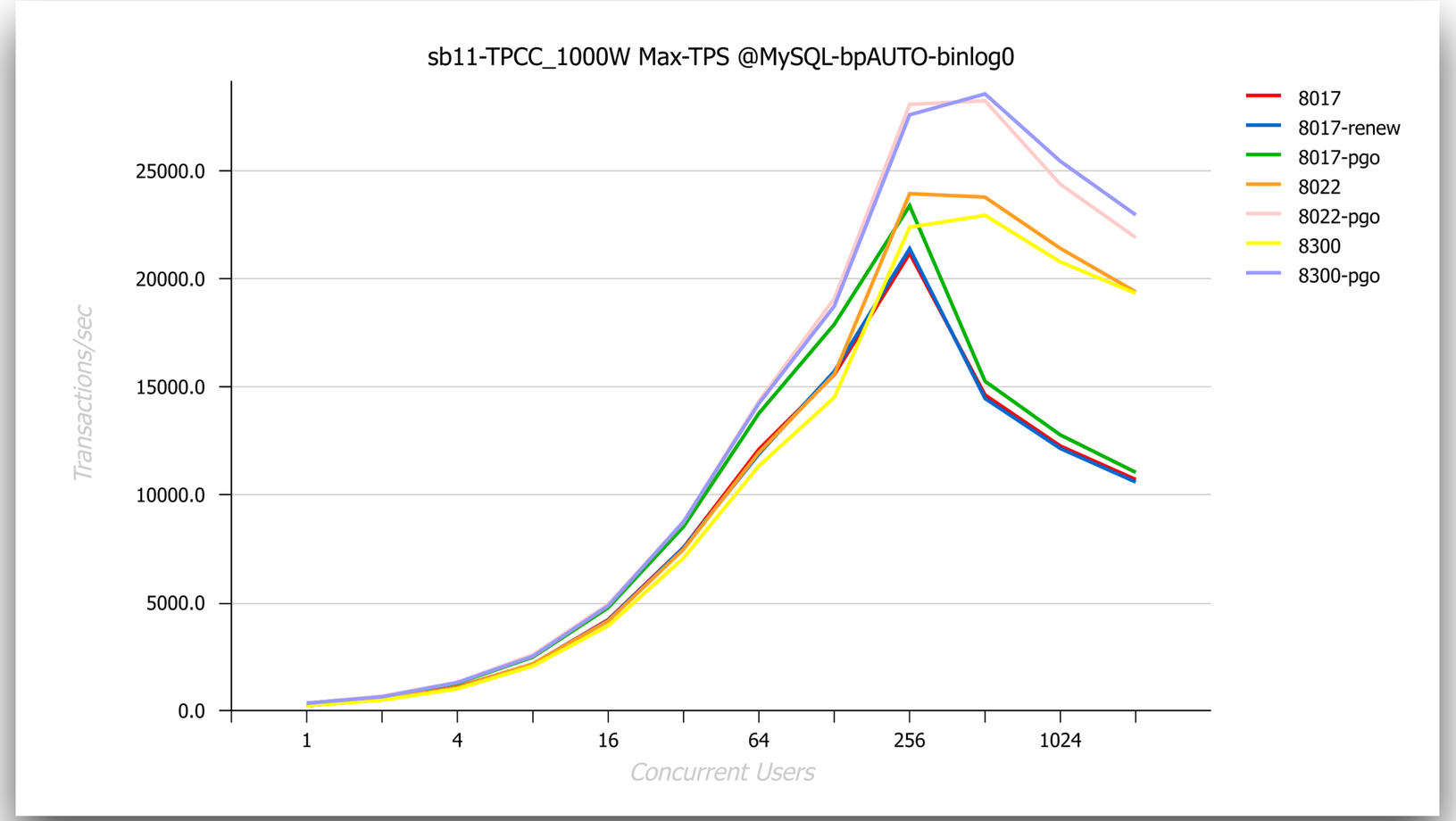
MySQL Regressions.. (in case you could think we don't care)

- TPCC example..
  - comparing main "milestone" releases : regression on 8.3



MySQL Regressions.. (in case you could think we don't care)

- TPCC example..
  - comparing main "milestone" releases + **all** PGO-optimized : no regression on 8.3 ?..



Thank you !

- Hope it was not boring ;-))

MySQL

ORACLE

MySQL Performance: Understanding & Tuning of IO-Bound Workloads (part-2)

Overview 2026

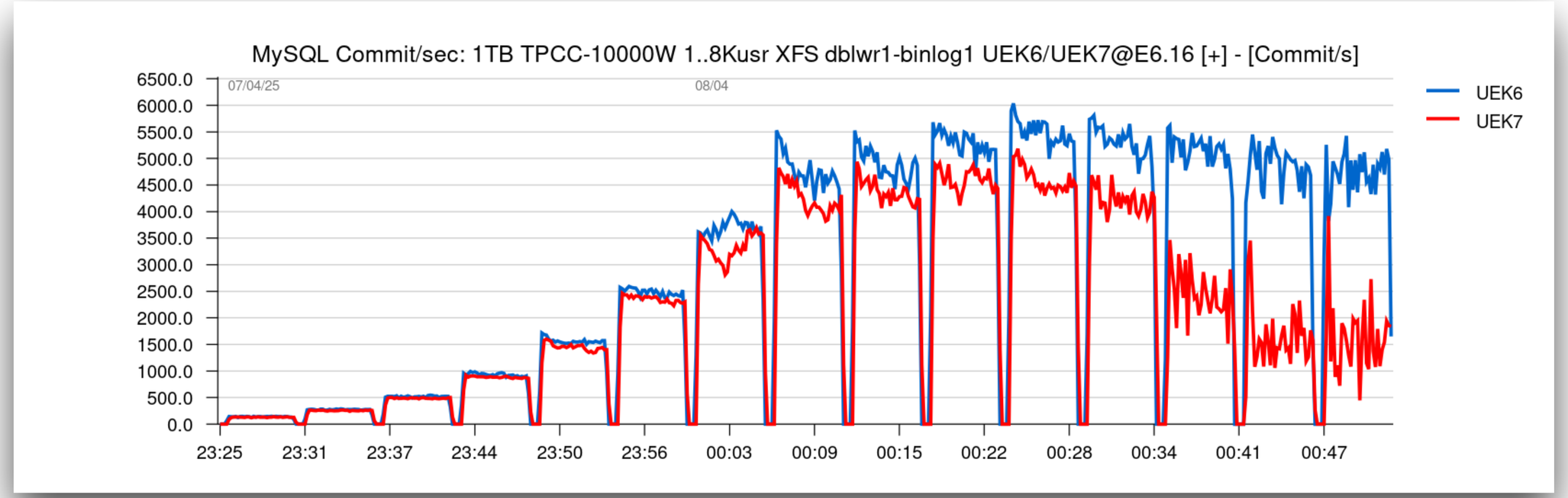
Dimitri KRAVTCHUK
MySQL Performance Architect @Oracle

IO-bound Workloads

- have IO Reads
- e.g. all complexity of CPU-bound + IO Reads !
- Storage = master piece !
  - both IOPS & MB/s capacities are critical !
  - fsync latency is **always** critical for REDO Writes ;-))
  - if you use RAID on top of Storage => use stripe size 128K/256K (for DBLWR)
- bigger RAM size always helps
- Linux kernel (see later)
- Filesystem choice :
  - XFS : most advanced
  - EXT4 : limited, but ok
  - ZFS : if you're ZFS expert
  - BTRFS : if you want to slowdown everything ;-))

Linux Kernel : Regression..

- Kernel 5.4 -vs- 5.15 (and any newer kernel) on 1TB TPCC :



- Oracle Linux Team has a potential fix with improved Scheduler !

Linux Kernel : Filesystem Cache

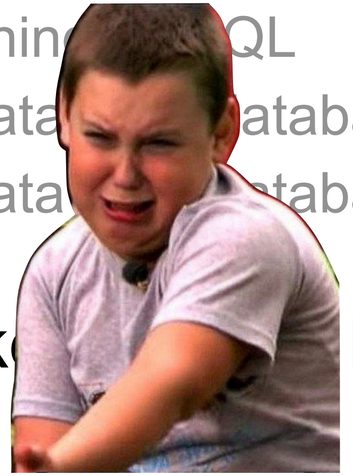
- No way to limit FS cache !
- Linux Kernel may give priority to FS cache
- e.g. for some strange reasons it’ll decide to make FS cache bigger !
- and then.. move MySQL process to SWAP ! ;-))
- currently the only workaround : switch SWAP = off
- **SR** : could we push Linux Kernel devs to implement FS cache limit ?..

IO-bound Read-Only

- Dilemma :
  - 256GB RAM on the server running MySQL
  - big batch-1 scanning all 1TB data from database-1 when runs alone = takes 1H
  - big batch-2 scanning another 1TB data from database-2 when runs alone = takes 1H
- Question : how long it’ll take if we run both batches on the same time ?..

IO-bound Read-Only

- Dilemma :
  - 256GB RAM on the server runnin QL
  - big batch-1 scanning all 1TB data atabase-1 when runs alone = takes 1H
  - big batch-2 scanning all 1TB data tabase-2 when runs alone = takes 1H



- Question : how long it'll tak both batches on the same time ?..
- Answer : **3H** (or more) !!
- Why ?…..

IO-bound Read-Only

- Dilemma :
  - 256GB RAM on the server running MySQL
  - big batch-1 scanning all 1TB data from database-1 when runs alone = takes 1H
  - big batch-2 scanning all 1TB data from database-2 when runs alone = takes 1H
- Question : how long it'll take if we run both batches on the same time ?..
- Answer : **3H** (or more) !!
- Why ?…..
  - batch-1 : brings its data to BP, removes data of batch-2 from BP..
  - batch-2 : brings its data to BP, removes data of batch-1 from BP..
  - e.g. run one-by-one = 2H total time ;-))

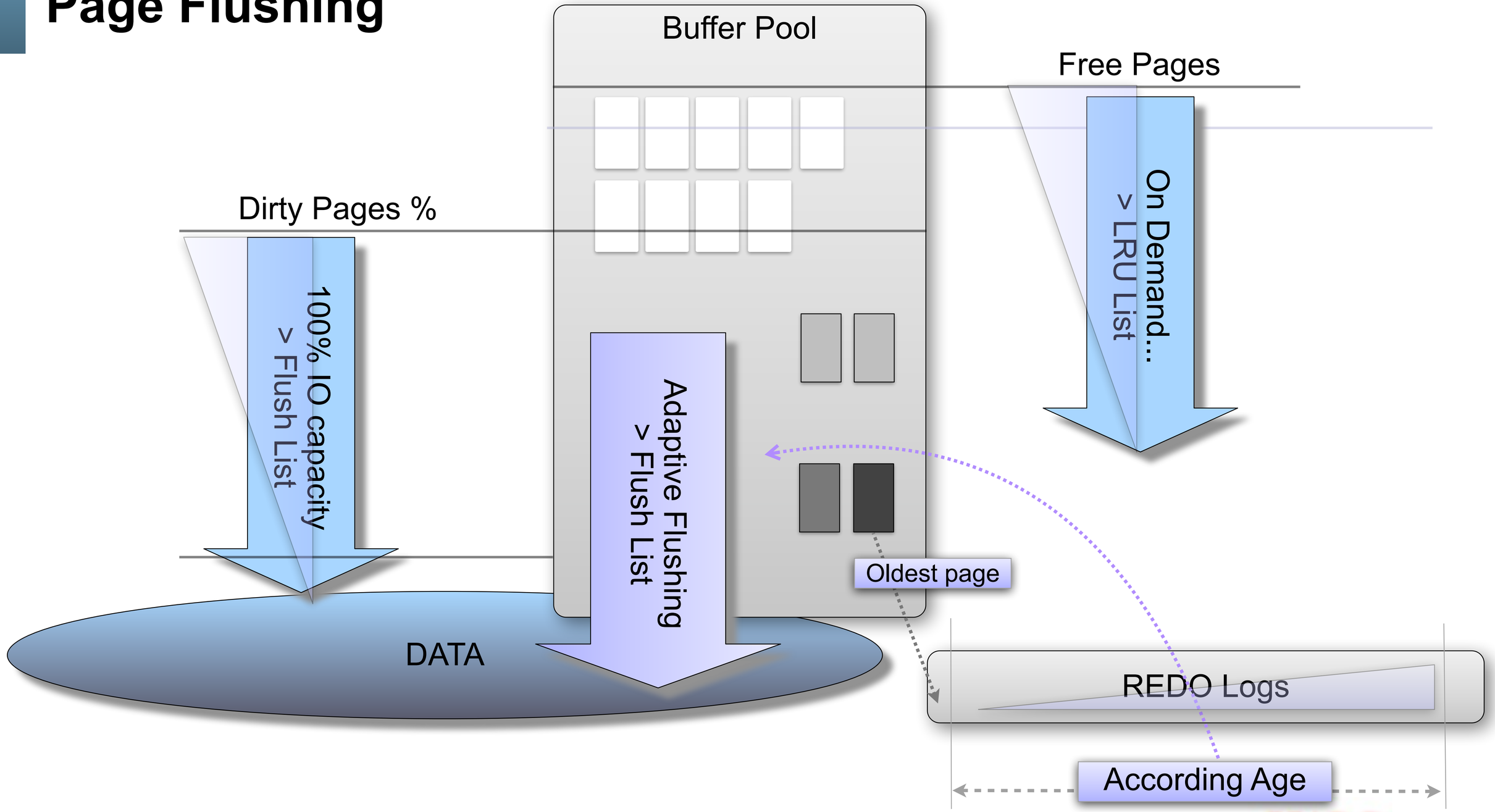
IO-bound Read-Only

- One of possible “tricks” to speed-up a single RO batch :
  - stop MySQL Server
  - drop FS cache (sync ; echo 3 > /proc/sys/vm/drop_caches)
  - start MySQL Server with “fsync” (instead of O_DIRECT)
  - run your batch-1 and leave Filesystem to do pre-fetch (read-ahead) for you !
  - drop FS cache again
  - run your batch-2
  - ….
  - restart MySQL Server with O_DIRECT => ready for RW OLTP ;-))

IO-bound Read-Write Overall Complexity

- To be able to Read a Page from Storage you need a Free Page in BP
- To get a Free Page in BP you need to Flush a Dirty Page from BP
- e.g. you need to do IO Write to be able to do IO Read
- but your Storage's IOPS capacity is limited
- if you do too many IO Reads => you have no more capacity for IO Writes
- if you do too many IO Writes => no more capacity for IO Reads..
- so, it's all about a right balance between IO Reads & Writes !
  - and this is the reason "why" InnoDB IO-Reads are not using AIO (will be disaster)
- NOTE :
  - Page Writes are most efficient with AIO + O_DIRECT (default since 8.4)
  - we don't use AIO for Page Reads (just O_DIRECT IO-Reads only)
  - same for DBLWR, but doing IO-Writes (O_DIRECT without AIO)
  - this all may add complexity to your Filesystem (ex.: EXT4 and 4.14 - 5.0 kernels)

Page Flushing

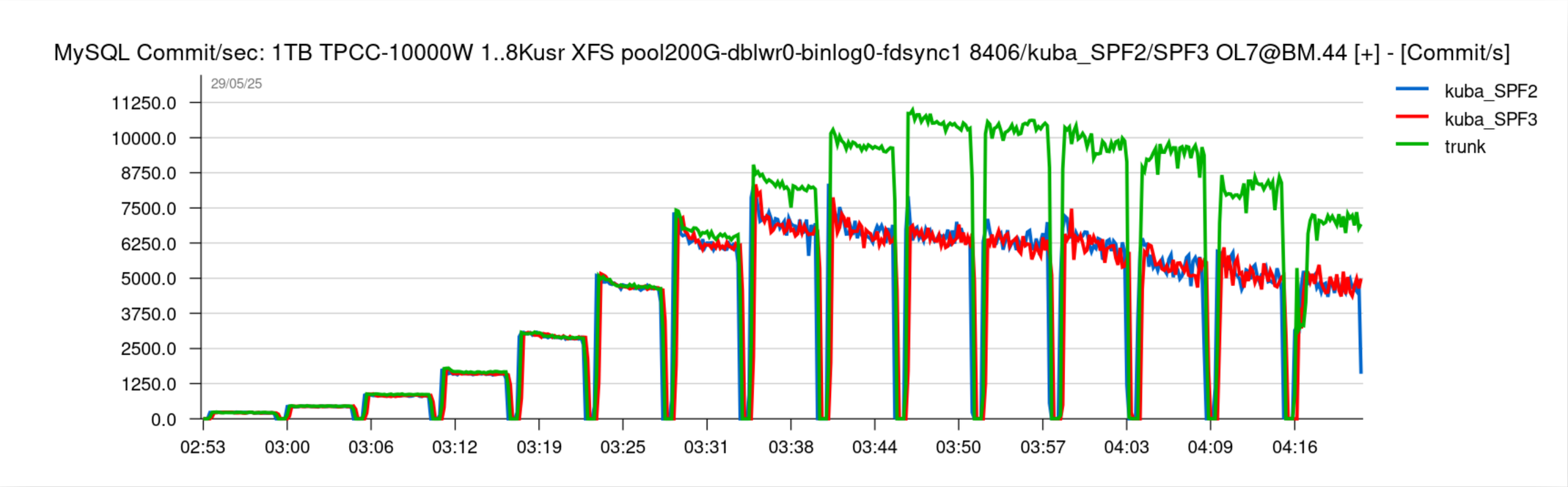


LRU Page Flushing

- Page Cleaners : Driven by Free Pages demand
  - involved when there is no more Free Pages
  - theory : scanning LRU up to “LRU depth” level
  - reality : LRU depth = amount of Free Pages to deliver ;-))
  - found not-Dirty Page => evict and move to Free Pages list
  - found Dirty Page => Page Flush first, then evict and move to Free Pages list
- User Thread :
  - 1) tries to get a Free Page => if ok, return
  - 2) if did not get => send signal to Page Cleaners and do a short wait loop
  - 3) if still did not get => involve Single Page Flush and goto 1)
- Single Page Flush :
  - user thread will scan LRU itself, and then evict or flush one Page
  - still better than do nothing !

Single Page Flush

- Historically often blamed
  - reality : blame is not really justified ;-))
  - tentatives to disable Single Page Flush -vs- enabled (green) on 1TB TPCC :

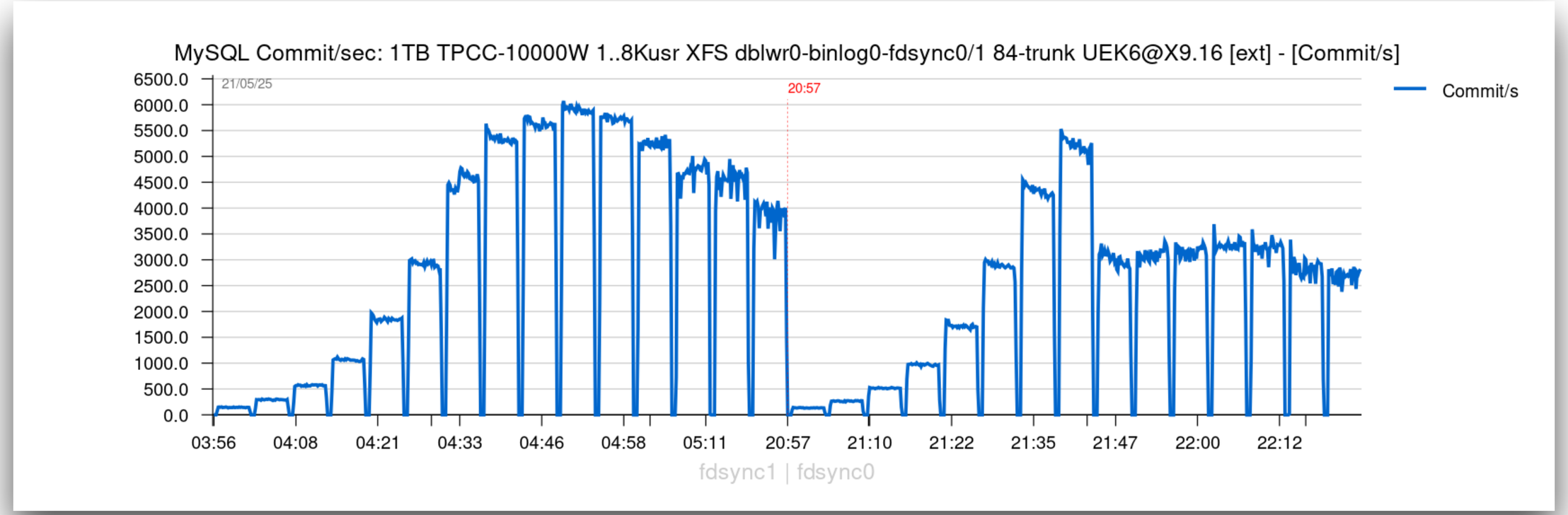


IO-bound Workloads : Tuning

- Config :
  - innodb_lru_scan_depth = 1K (default)
    - generally you don’t need a bigger value since Page Cleaners are waked-up on-demand
  - AIO + O_DIRECT : mandatory ! (O_DIRECT_NO_FSYNC : banned !!)
  - innodb_use_fdatasync = 1 : mandatory ! (new default)
  - IBUF : innodb_change_buffering = all, innodb_change_buffer_max_size = 5
  - DBLWR : try **default** settings first, tune further if Storage is saturated
- Monitoring :
  - buffer_LRU_get_free_search/sec <= Free Pages demand by Users
  - buffer_LRU_get_free_loops/sec <= User’s loop waits on Free Pages..
  - buffer_LRU_get_free_waits/sec <= User’s sleep waits on Free Pages..
  - buffer_LRU_single_flush_num_scan/sec <= Single Page Flush by User
  - buffer_LRU_batch_evict_total_pages/sec <= Pages Evicted by Cleaners
  - buffer_LRU_batch_flush_total_pages/sec <= Pages Flushed by Cleaners
  - buffer_LRU_batch_scanned/sec <= Pages Scanned by Cleaners

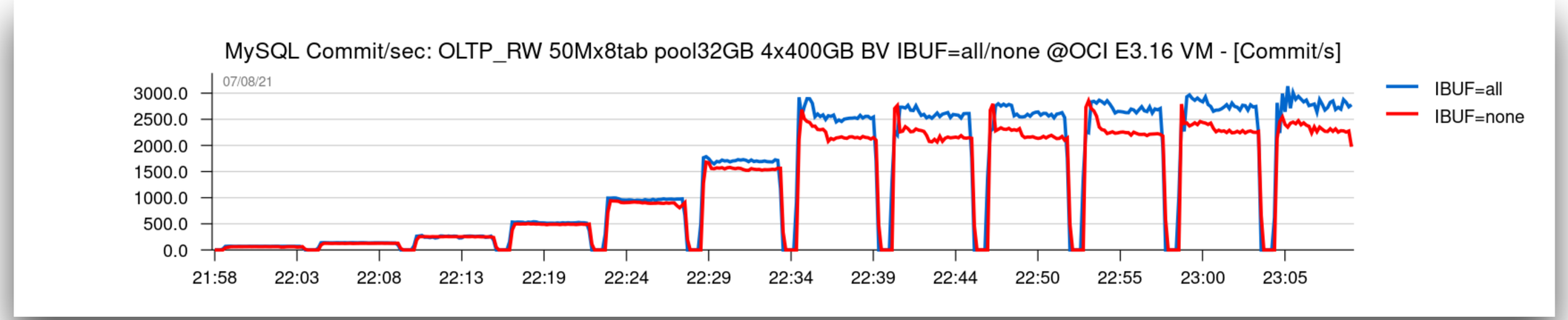
IO-bound : fdatasync Impact

- 1TB TPCC : innodb_use_fdatasync = 1/0

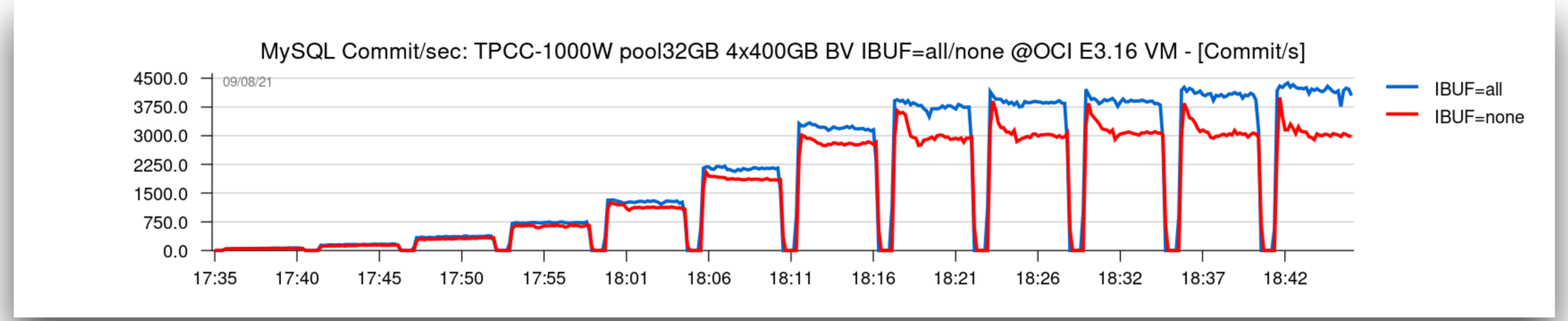


IO-bound : IBUF Impact (Change Buffer)

- IO-bound OLTP_RW :



- IO-bound TPCC :

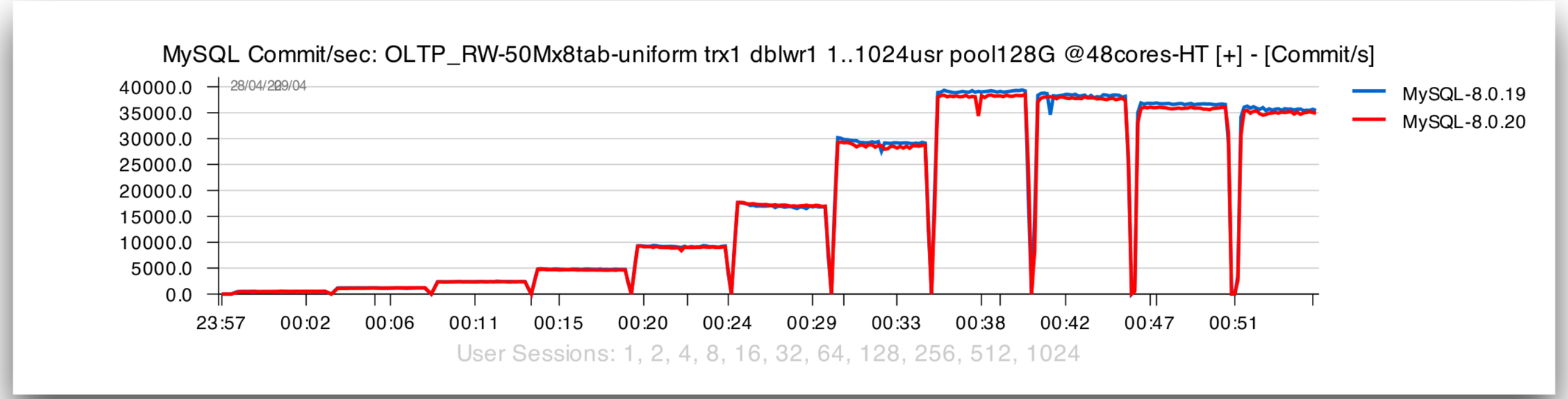


ORACLE

IO-bound : DBLWR Impact

- New DBLWR since MySQL 8.0.20 !!
  - just in case you forgot about innovations in 8.0 ;-))

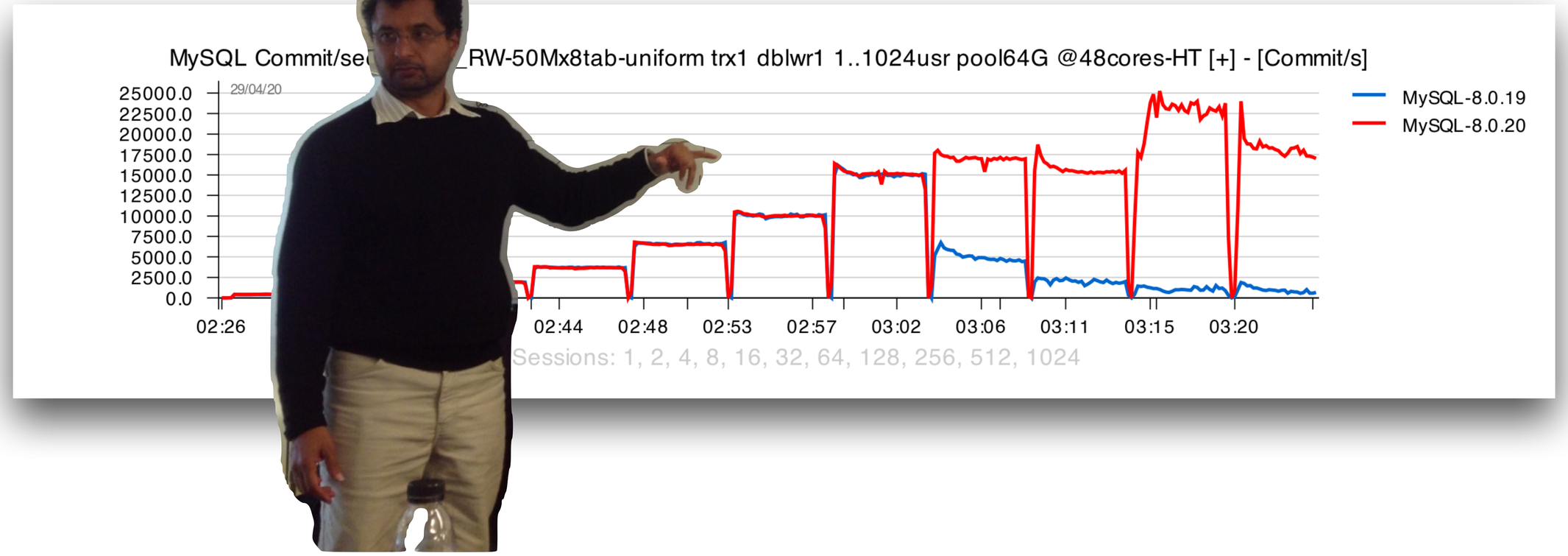
- CPU-bound :



IO-bound : DBLWR Impact

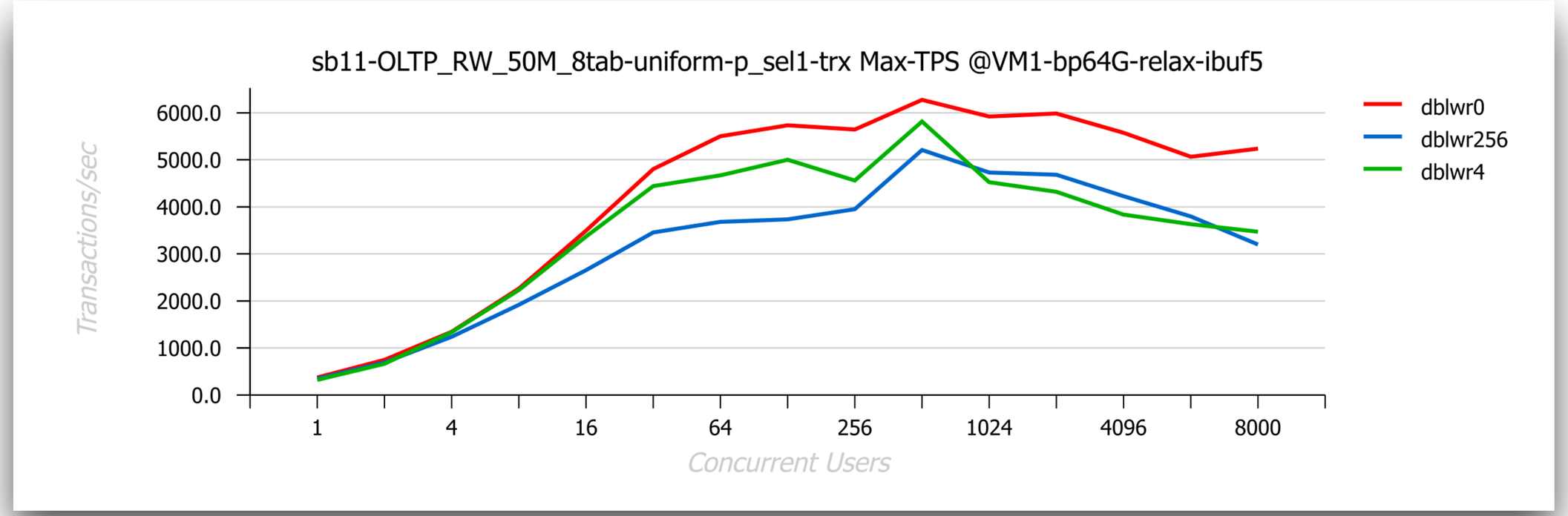
- New DBLWR since MySQL 8.0.20 !!
  - just in case you forgot about innovations in 8.0 ;-))

- IO-bound :



IO-bound : DBLWR Impact

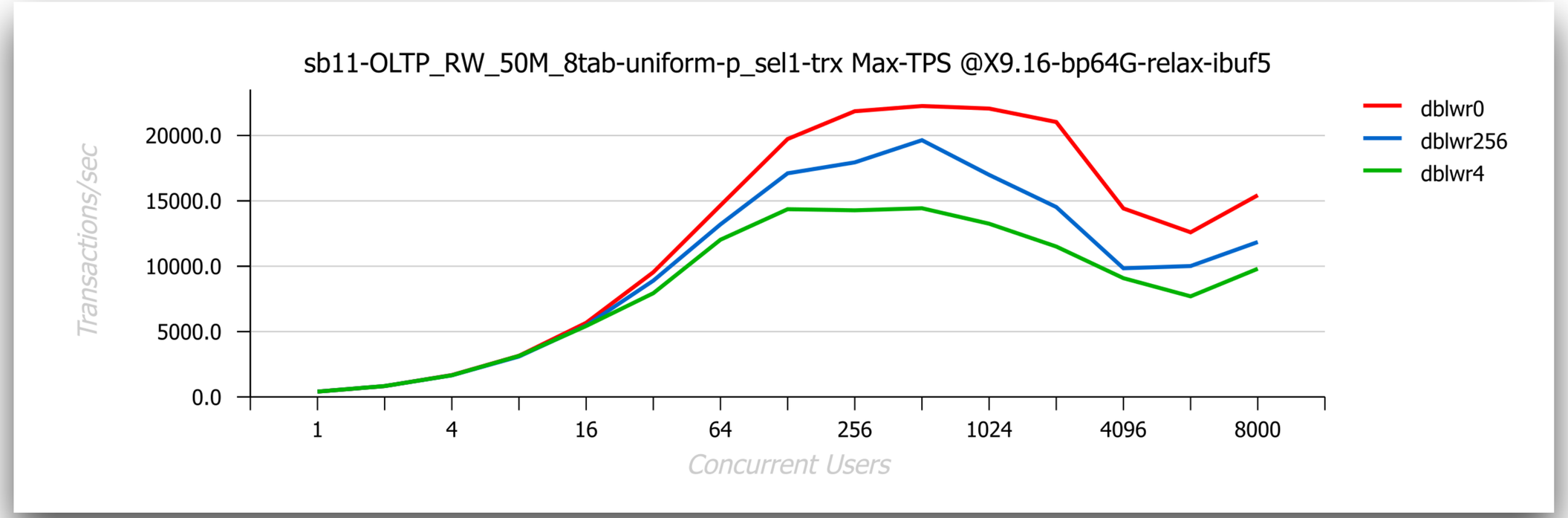
- OLTP_RW++ : Saturated Storage
  - dblwr0 : DBLWR=off (expectation for Atomic IO)
  - dblwr4 : DBLWR=on, 2 files x 2 pages (old default)
  - dblwr256 : DBLWR=on, 2 files x 128 pages (default since 8.4)



- Oracle Linux Team => Atomic IO !!! (already pushed to upstream)

IO-bound : DBLWR Impact

- OLTP_RW++ : Higher Capacity Storage
  - dblwr0 : DBLWR=off (expectation for Atomic IO)
  - dblwr4 : DBLWR=on, 2 files x 2 pages (old default)
  - dblwr256 : DBLWR=on, 2 files x 128 pages (default since 8.4)



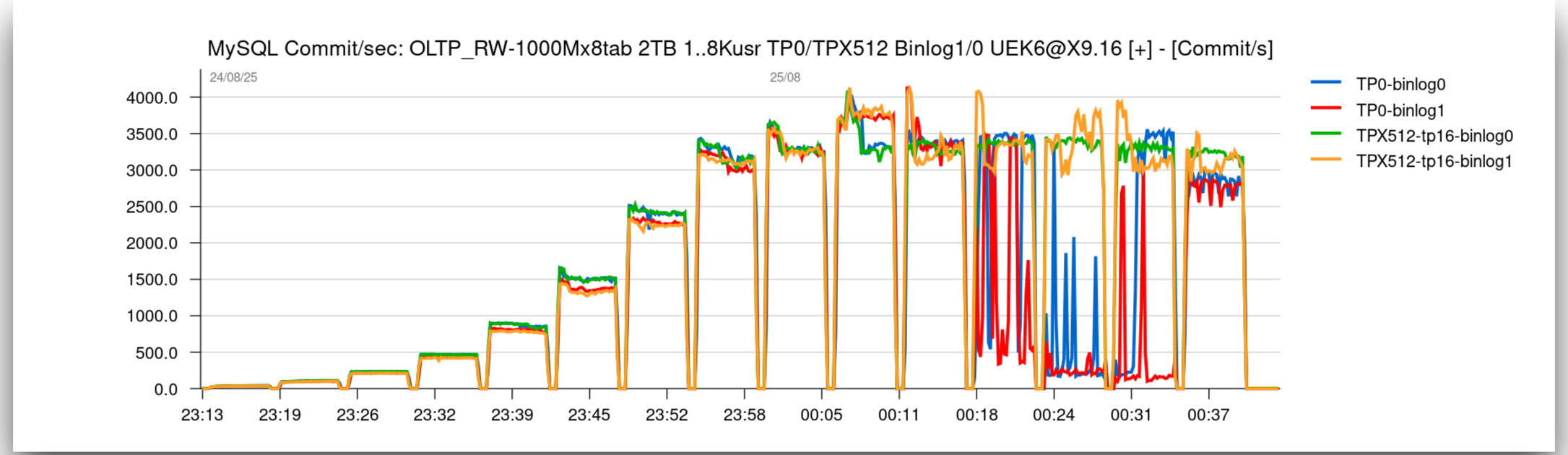
- Oracle Linux Team => Atomic IO !!! (already pushed to upstream)

Evaluation : 2TB OLTP_RW IO-bound Workload

- System :
  - 16cores VM, 256GB RAM, 8TB HighPerf OCI BV Storage, OL8, XFS
- MySQL Config :
  - BP size = auto (200GB) / 64GB
  - Binlog = 1 / 0
  - IBUF = 0 / 5
  - ThreadPool = on / off
- Test Workload :
  - OLTP_RW-uniform, 8 tables x 1000M rows each
  - growing load level : 1, 2, 4, 8, … 4K, 6000, 8000 users
  - InnoDB transparent compression : OFF / ON

2TB OLTP_RW : First Results

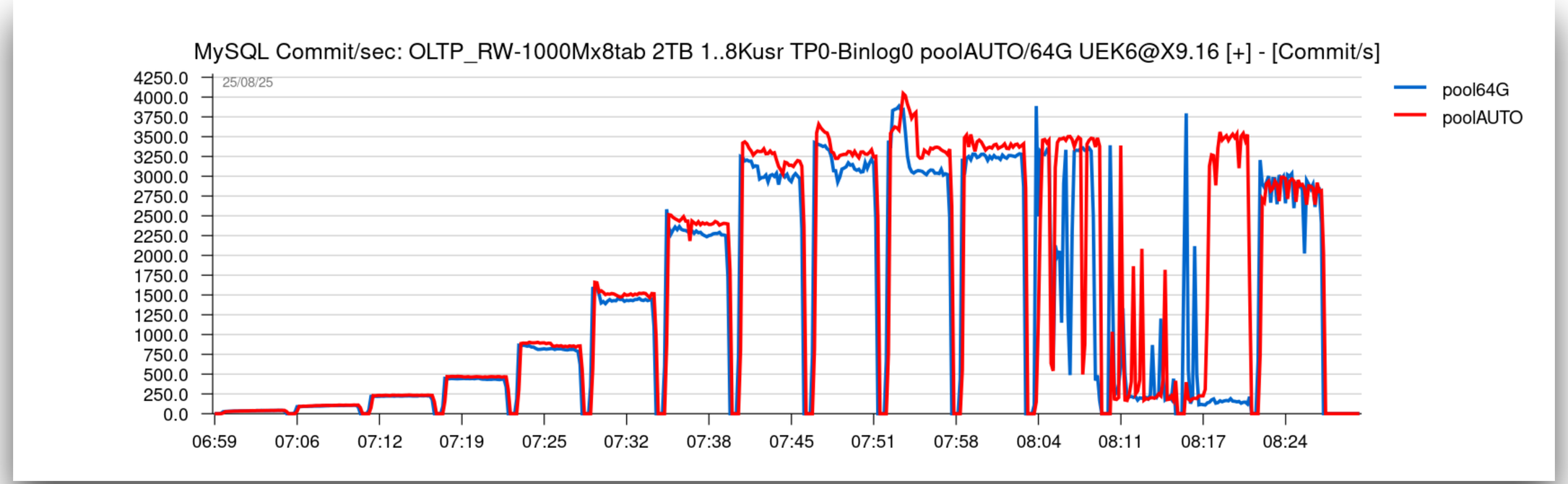
- Comparing :
  - Binlog = ON / OFF
  - ThreadPool = ON / OFF



- Binlog = ON / OFF => no difference
- TPS disaster since 2K users, but ThreadPool is able to solve this !

2TB OLTP_RW : First Results

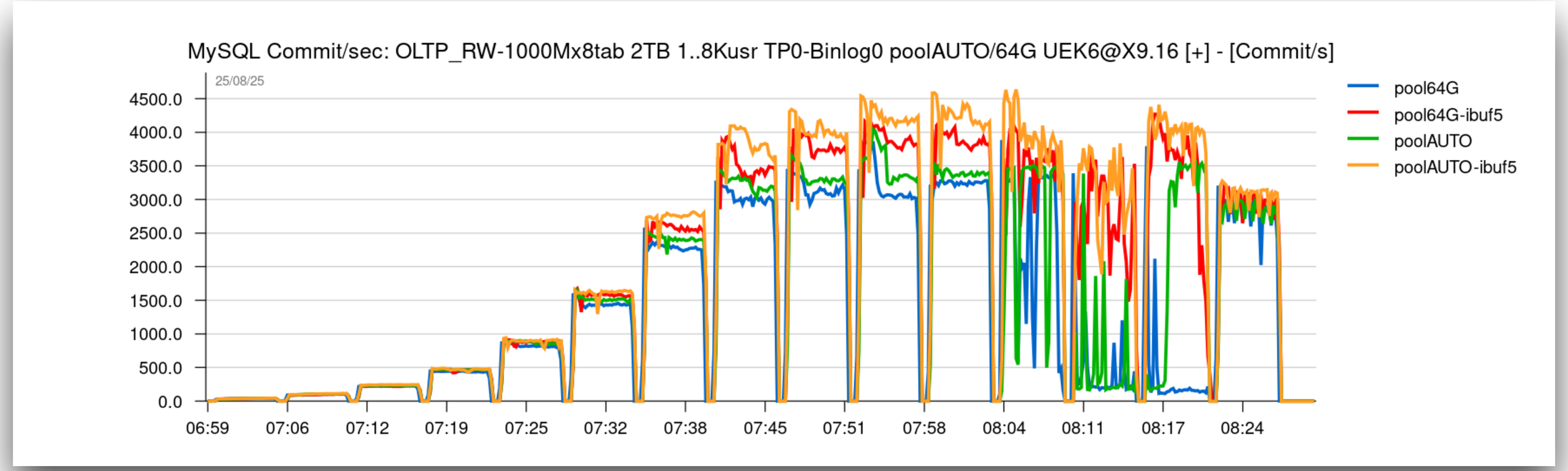
- Comparing :
  - BP size = auto (200G) / 64G
  - ThreadPool = OFF, Binlog = OFF



- small or bigger BP => no much difference, Storage capacity drives the results

2TB OLTP_RW : First Results

- Comparing :
  - BP size = auto (200G) / 64G
  - IBUF = 0 / 5



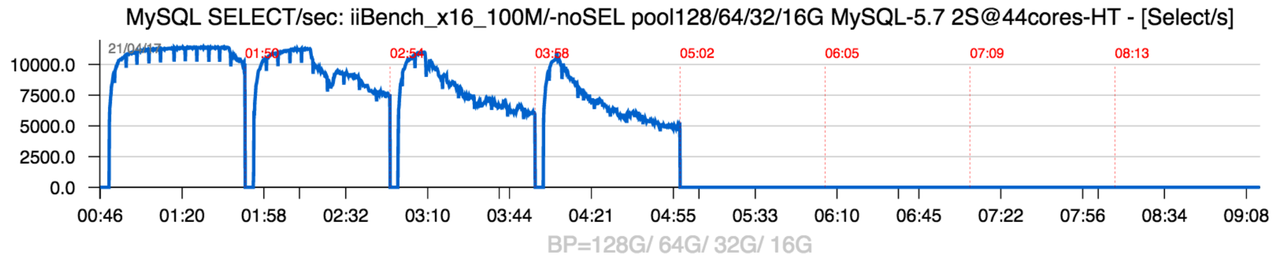
- IBUF helps (as expected), more help with a bigger BP (as expected)

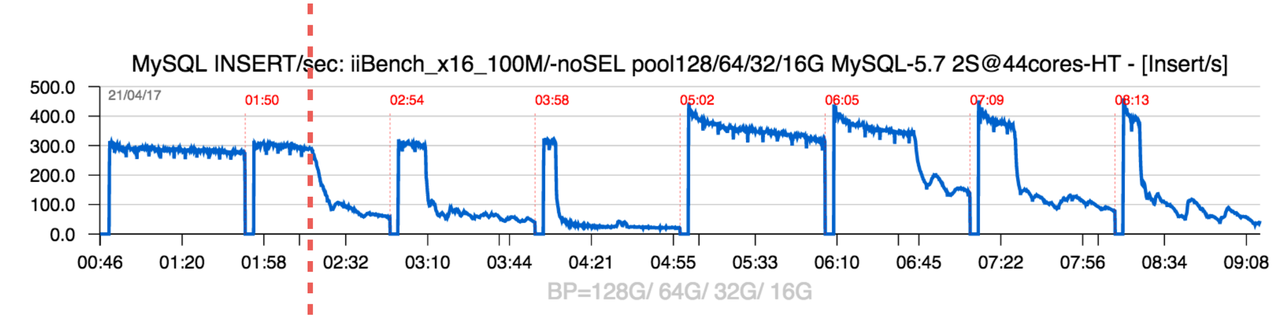
History from 2019 : iiBench (Index INSERT Benchmark)

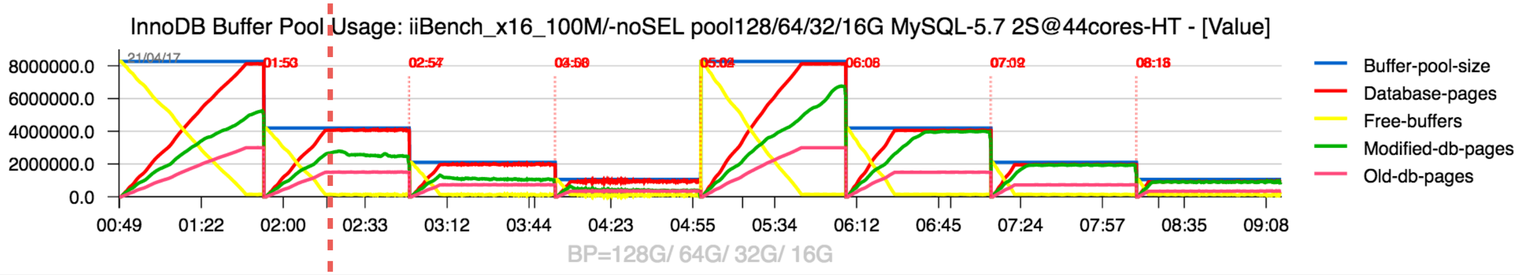
- Main claim :
  - InnoDB is xN times slower vs Write-oriented Engine XXX
  - so, use XXX, as it’s better
- Test Scenario :
  - x16 parallel iiBench processes running together during 1H
  - each process is using its own table
  - one test with SELECTs, another without..
- Key point :
  - during INSERT activity, B-Tree Index in InnoDB grows 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)

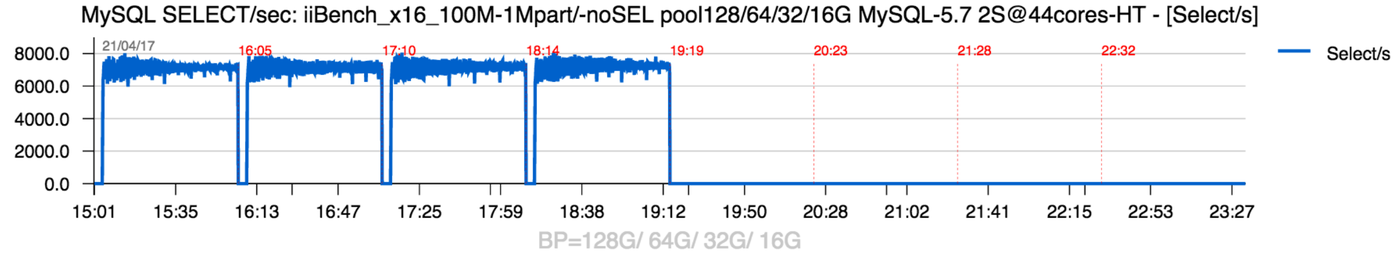


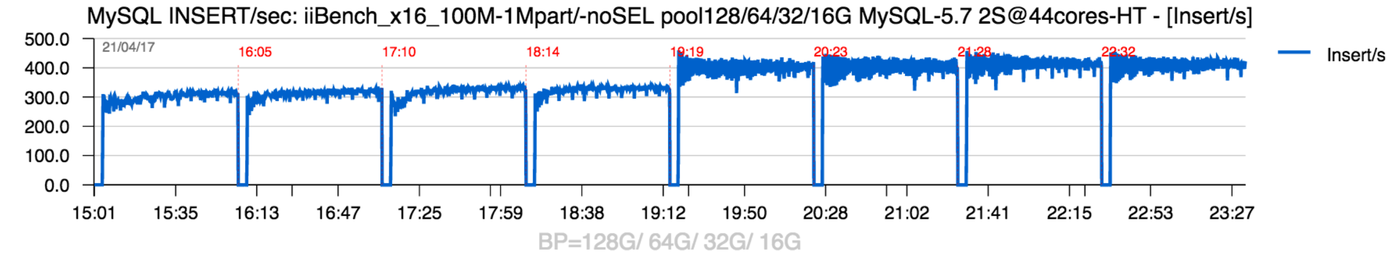


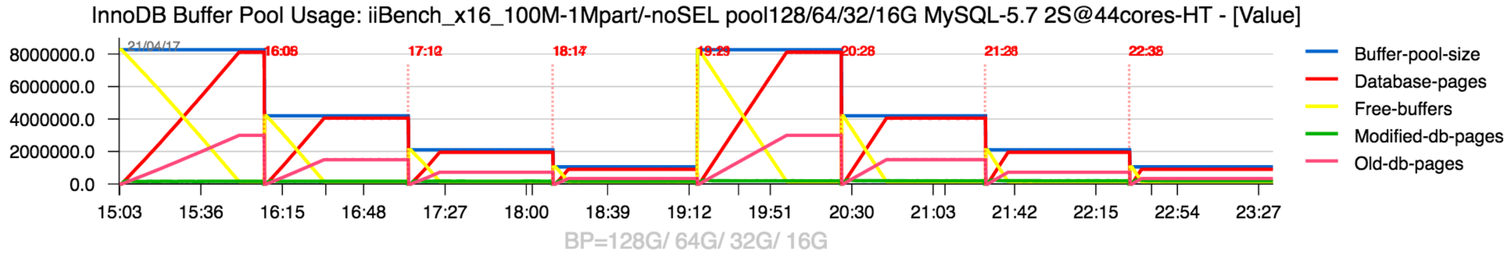


iiBench 100M x16 & 1M-parts : BP= 128G/ 64G/ 32G/ 16G

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

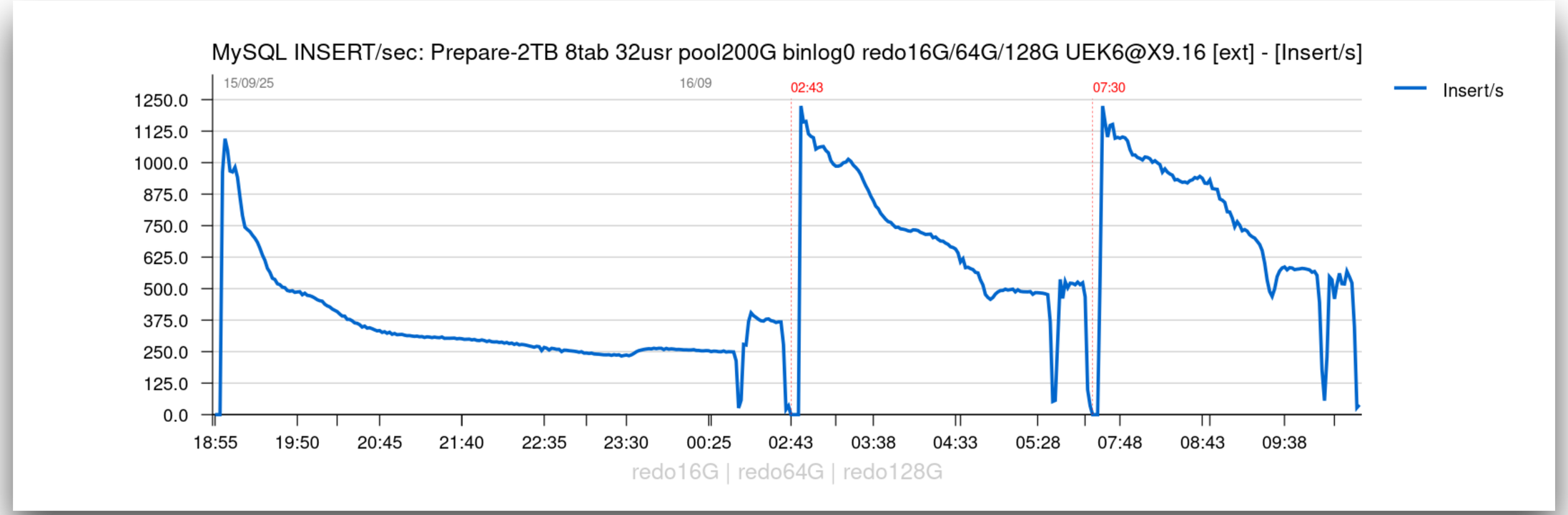






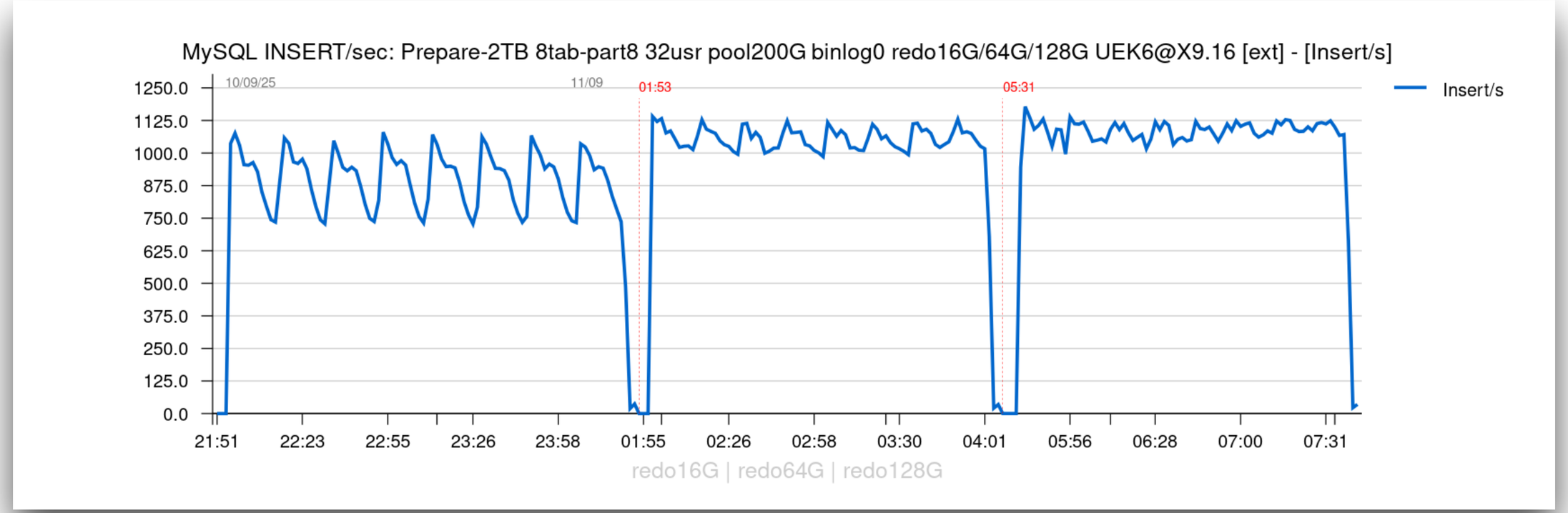
2TB OLTP_RW : Data Load with 32 connections

- Problem : progressive decrease of INSERT/sec rate
- Workaround #1 : increase REDO size
  - REDO = 16GB / 64GB / 128GB



2TB OLTP_RW : Data Load with 32 connections

- Problem : progressive decrease of INSERT/sec rate
- Workaround #2 : use Partitions
  - 8 Partitions + REDO = 16GB / 64GB / 128GB



2TB OLTP_RW : InnoDB Transparent Page Compression

- History background :
  - delivered since MySQL 5.7
  - based on Filesystem “punch holes” feature
  - **main problem** :
    - it takes a long time for FS to drop a big “sparse” file
    - InnoDB gets a signal on a long wait timeout and aborts..

- Today :
  - InnoDB has capability to drop data files in background (no abort anymore)
  - XFS is now dropping “sparse” files in background, mostly immediate ACK on unlink()
  - **main problem is gone !** sounds good ? — not really :
    - still some unexpected slowdowns in IO activity
    - side effects : IO activity on “sparse” file impacts IO on “normal” files, etc..
    - e.g. still not ready for Production

2TB OLTP_RW : Data Load Times & Transparent Compression

- Data Load :
  - 32 threads in parallel
  - using bulk INSERTs
  - 1K rows per INSERT

No Partitions

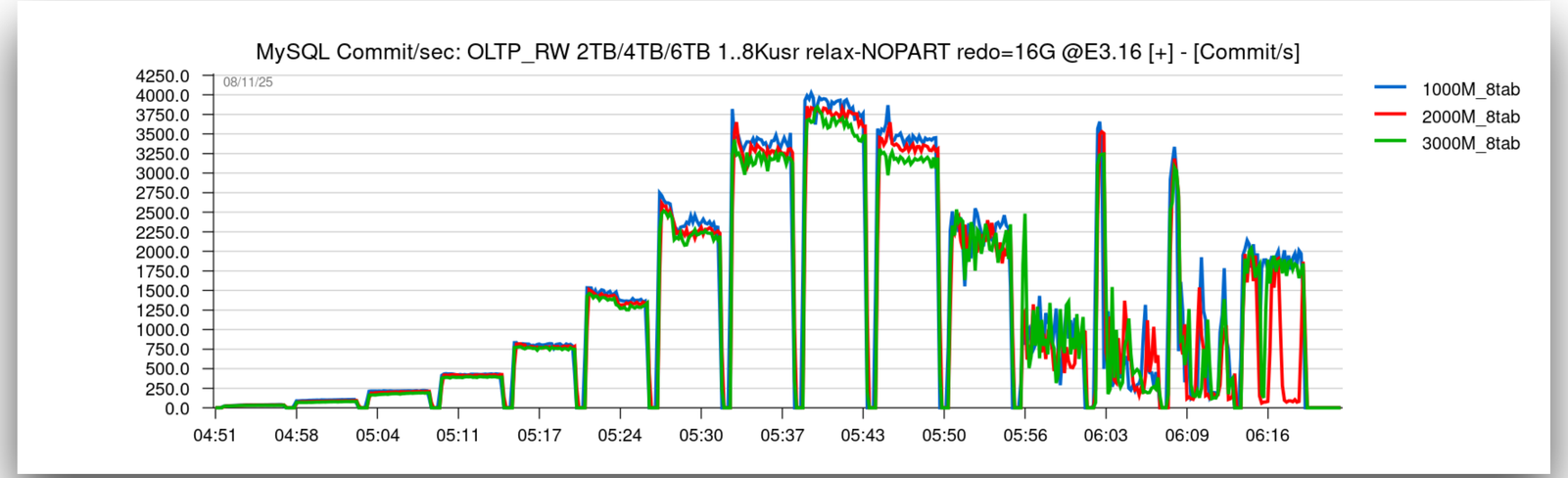
REDO size	compression=OFF	compression=ON
16G	6H20	15H15
64G	3H15	10H25
128G	2H54	9H52

8 Partitions

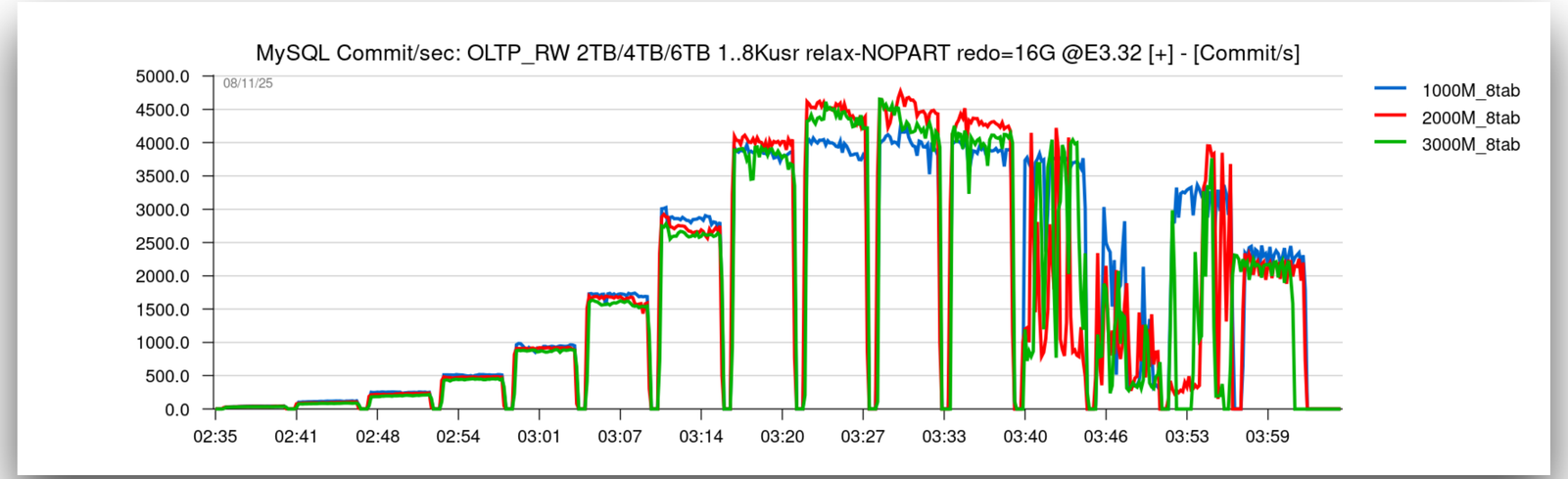
REDO size	compression=OFF	compression=ON
16G	2H30	5H42
64G	2H02	5H20
128G	2H08	5H00

IO-bound OLTP_RW : 2TB / 4TB / 6TB | REDO = 16GB

- E3.16 :
  - 16cores VM
  - 256GB RAM
  - 8TB BV Storage

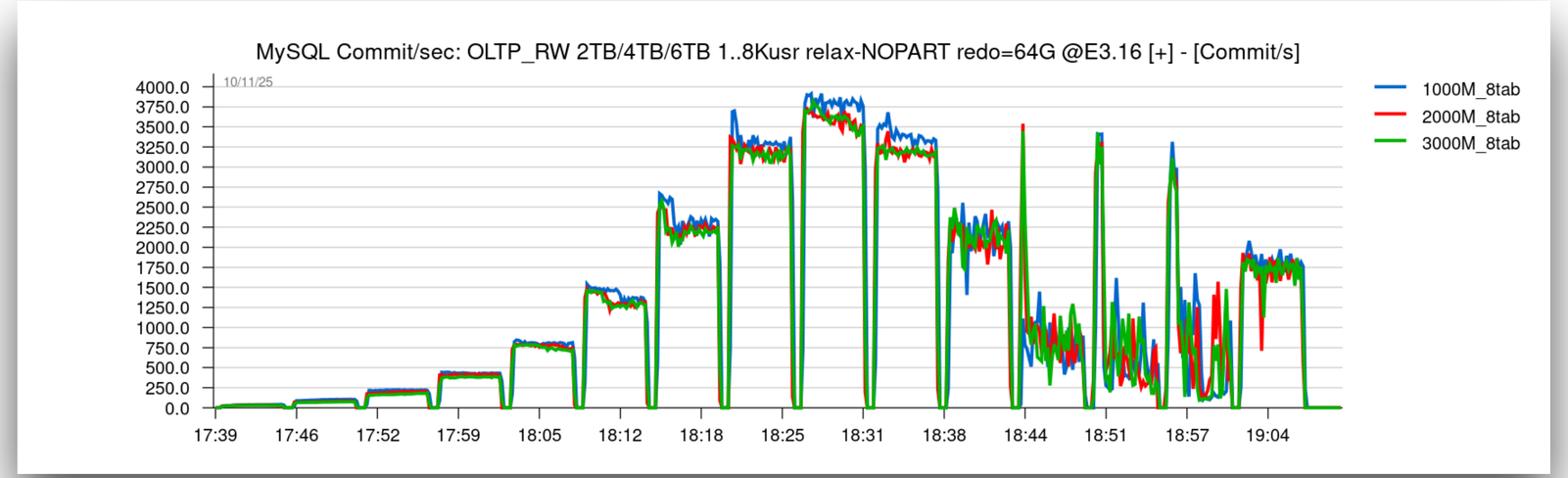


- E3.32 :
  - 32cores VM
  - 512GB RAM
  - 8TB BV Storage

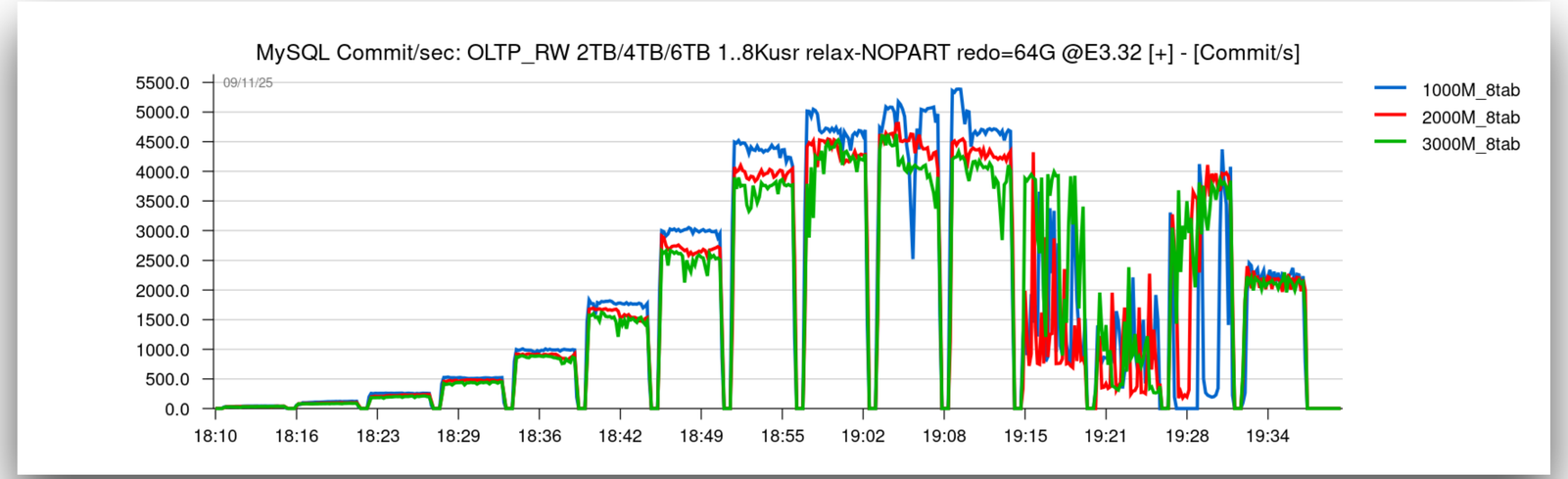


IO-bound OLTP_RW : 2TB / 4TB / 6TB | REDO = 64GB

- E3.16 :
  - 16cores VM
  - 256GB RAM
  - 8TB BV Storage

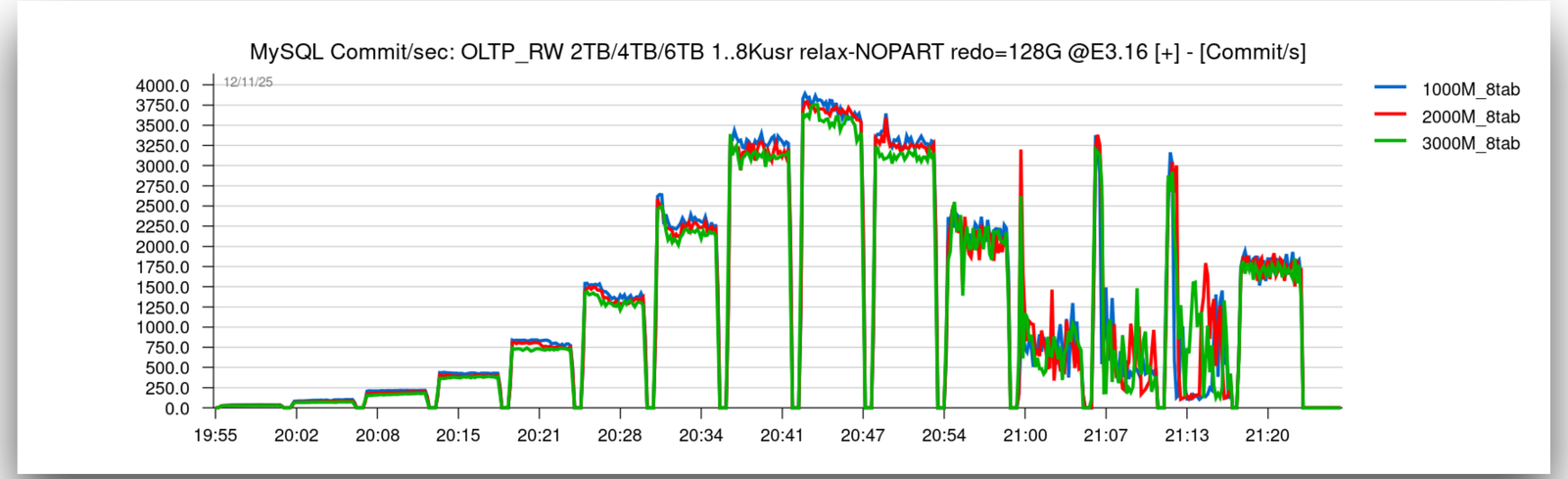


- E3.32 :
  - 32cores VM
  - 512GB RAM
  - 8TB BV Storage

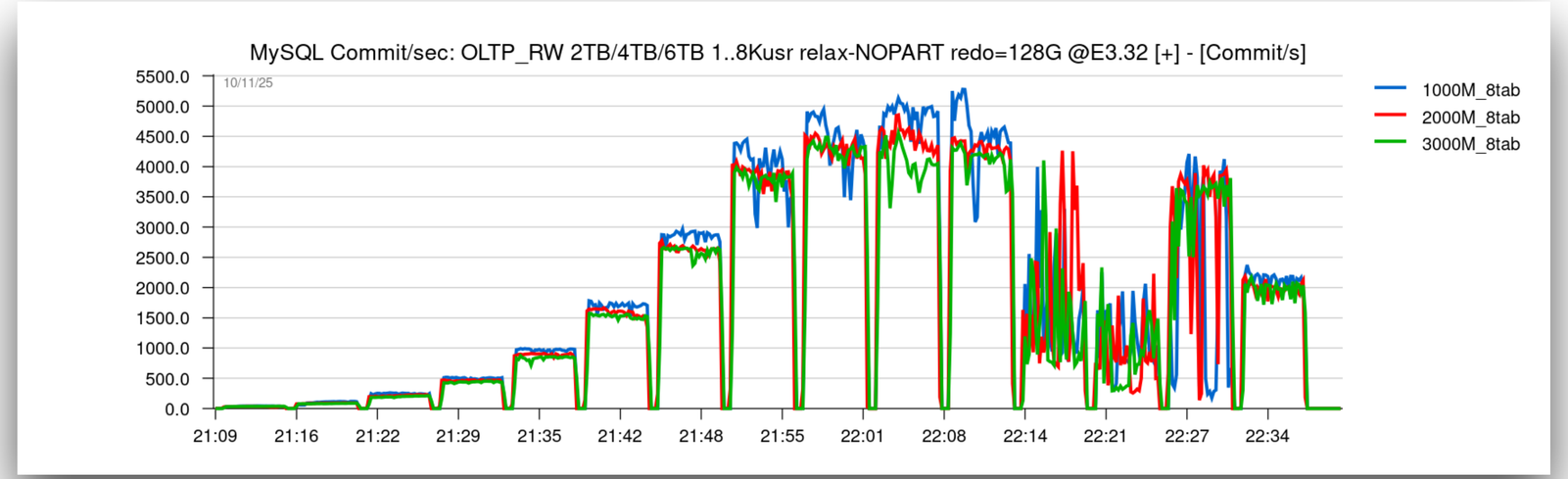


IO-bound OLTP_RW : 2TB / 4TB / 6TB | REDO = 128GB

- E3.16 :
  - 16cores VM
  - 256GB RAM
  - 8TB BV Storage



- E3.32 :
  - 32cores VM
  - 512GB RAM
  - 8TB BV Storage



IO-bound OLTP_RW : Data Load Times

- E3.16 :
  - 16cores VM
  - 256GB RAM
  - 8TB BV Storage

2TB data

REDO	NOPART	8-PART	NOPART-ibuf5	8-PART-ibuf5
16G	06H46	03H09	06H57	03H10
64G	03H59	02H45	04H03	02H47
128G	03H35	02H41	03H42	02H42

4TB data

REDO	NOPART	8-PART	NOPART-ibuf5	8-PART-ibuf5
16G	17H11	07H17	18H18	07H18
64G	12H24	05H48	13H41	05H52
128G	11H59	05H34	12H57	05H38

6TB data

REDO	NOPART	8-PART	NOPART-ibuf5	8-PART-ibuf5
16G	35H00	12H32	33H05	12H37
64G	29H36	09H02	26H38	09H10
128G	28H49	08H37	25H53	08H42

IO-bound OLTP_RW : Data Load Times

- E3.32 :
  - 32cores VM
  - 512GB RAM
  - 8TB BV Storage

2TB data

REDO	NOPART	8-PART	NOPART-ibuf5	8-PART-ibuf5
16G	04H30	02H43	04H20	02H44
64G	02H42	02H26	02H41	02H26
128G	02H39	02H23	02H35	02H23

4TB data

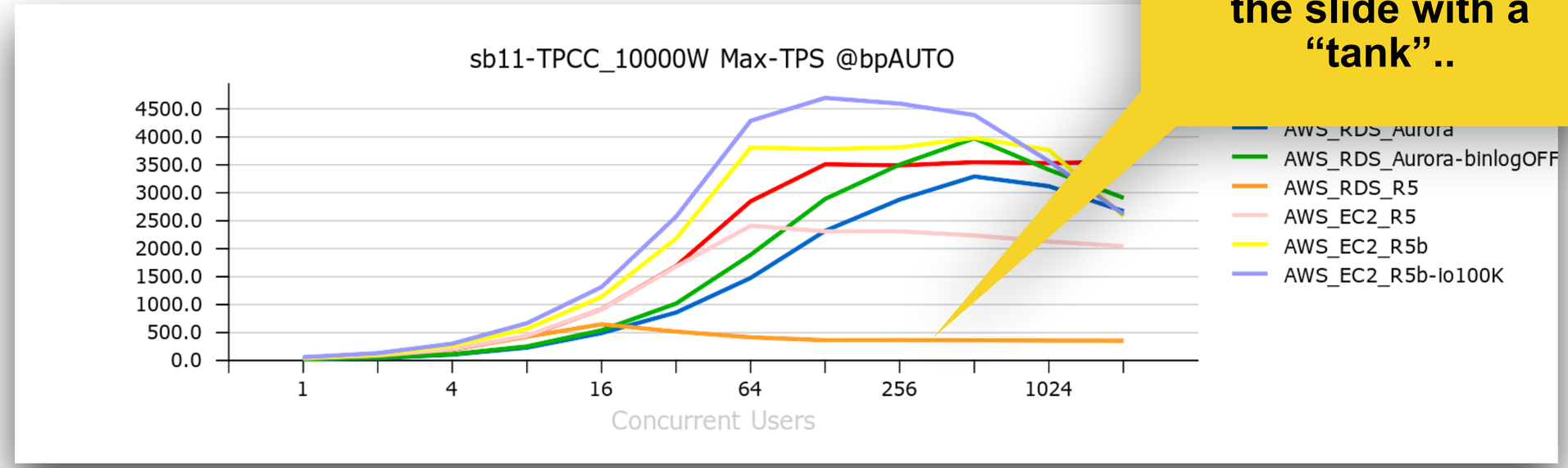
REDO	NOPART	8-PART	NOPART-ibuf5	8-PART-ibuf5
16G	11H58	05H47	12H19	05H50
64G	06H45	05H04	07H19	05H01
128G	06H43	04H57	06H59	04H55

6TB data

REDO	NOPART	8-PART	NOPART-ibuf5	8-PART-ibuf5
16G	20H14	09H10	22H57	09H16
64G	12H56	07H43	16H46	07H46
128G	12H07	07H35	16H09	07H41

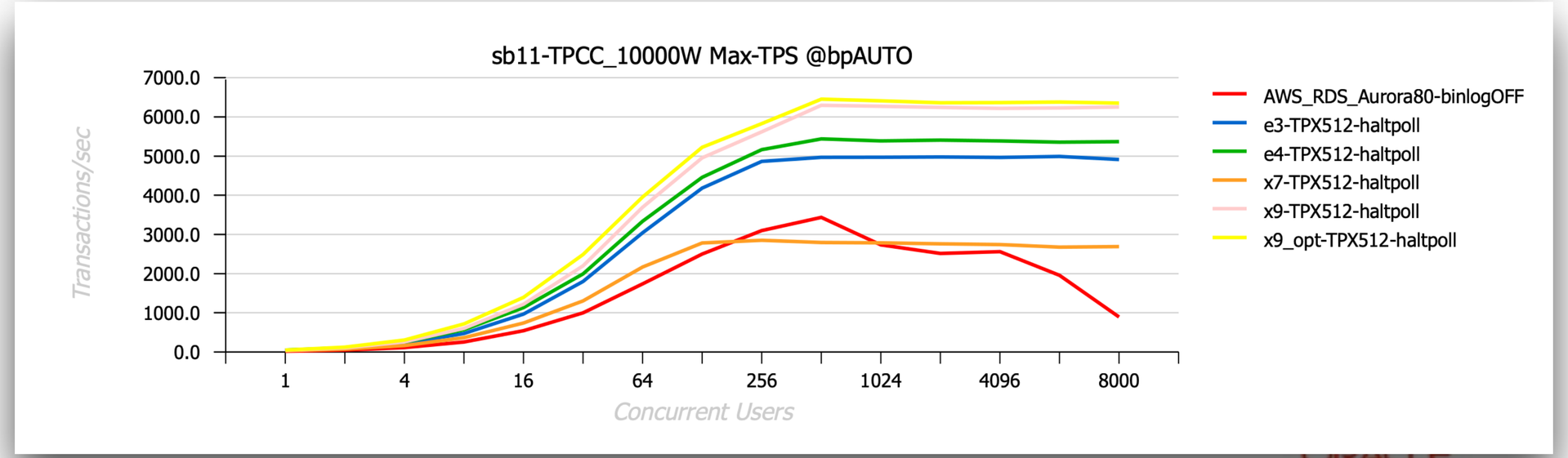
Few TPCC Results | 10000W (1TB), RAM 256G, 16cores

- Amazon
  - RDS
  - EC2
  - Aurora



If you recall the slide with a "tank"..

- MDS
  - E3
  - E4
  - X9



Hope everything is more clear now for you !

One more thing.. ;-))

- All graphs are built with ***dim_STAT*** (http://dimitrik.free.fr)
  - All System load stats (CPU, I/O, Network, RAM, Processes, etc..)
    - Mainly for Linux x64 / arm64 / MacOS (but can be any other UNIX too :-))
    - Add-Ons for MySQL, Oracle DB, PostgreSQL, Java, etc.
  - MySQL Add-Ons:
    - mysqlLOAD : compact stats data, multi-host monitoring oriented
    - mysqlWAITS : top wait events from Performance SCHEMA
    - InnodbSTAT : most important data from InnoDB status
    - innodbMUTEX : monitoring InnoDB mutex waits
    - innodbMETRICS : all counters from the METRICS table
    - Perf Profiling / DTrace call-stacks / etc..

- Links
  - http://dimitrik.free.fr - dim_STAT
  - http://dimitrik.free.fr/blog/posts/mysql-perf-bmk-kit.html - BMK-kit
  - http://dimitrik.free.fr/blog - Articles about MySQL Performance, etc.

Thank You !!!

ORACLE