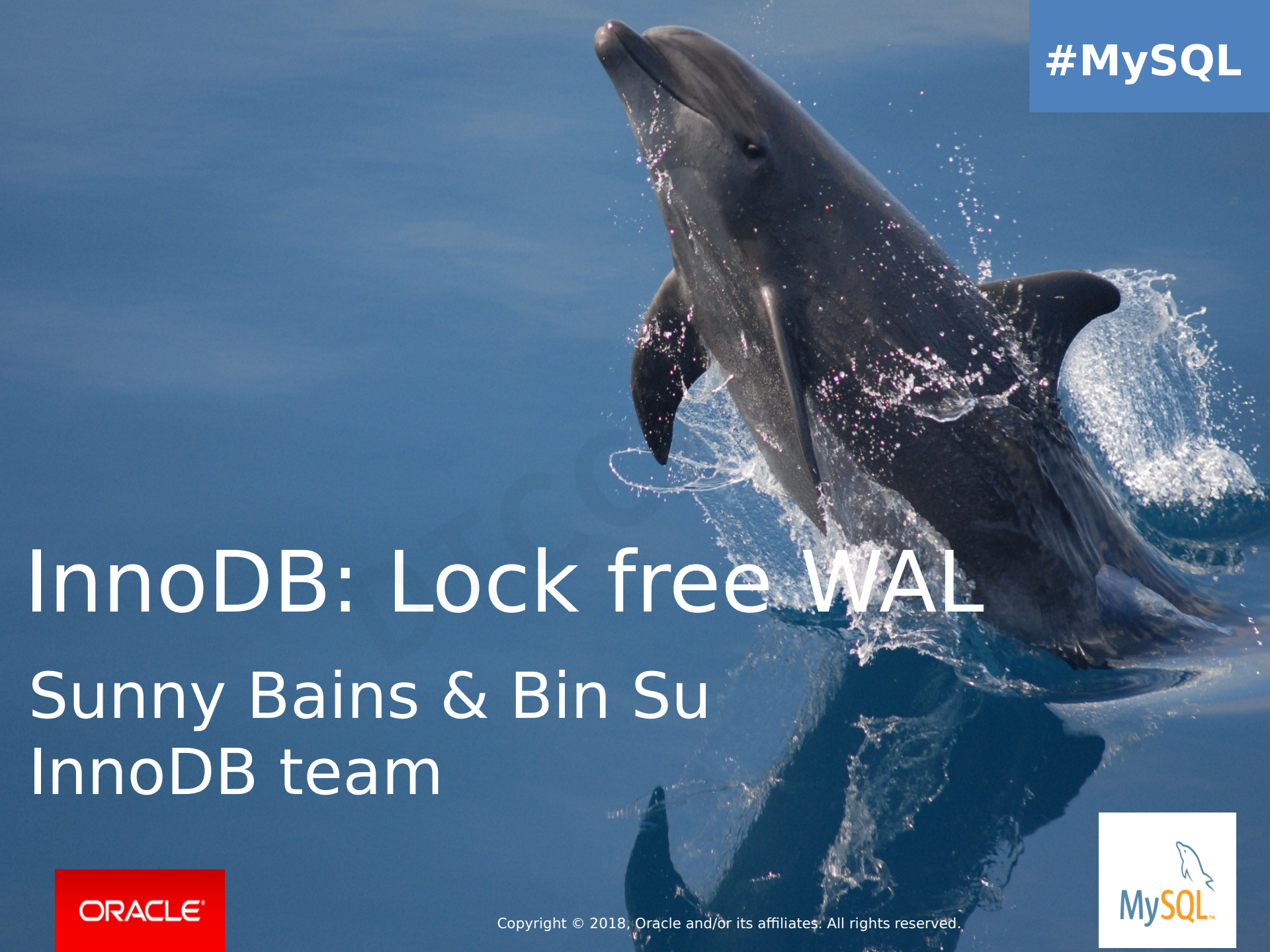
数领先机 智赢未来

DTCC 第九届中国数据库技术大会

DATABASE TECHNOLOGY CONFERENCE CHINA 2018

2018.05.10 – 12　北京国际会议中心



#MySQL

InnoDB: Lock free WAL

Sunny Bains & Bin Su
InnoDB team

ORACLE

MySQL

afe Harbor Statement

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.

Agenda

InnoDB concepts
New WAL design
Q&A

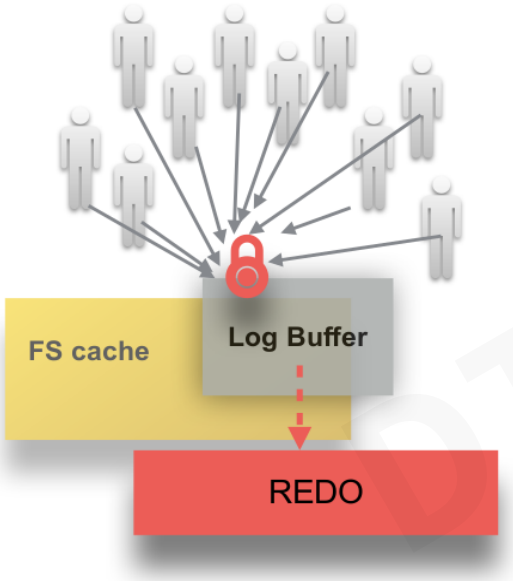
InnoDB concepts

- Mini-transaction (mtr)
  - Mechanism to log physical changes to data pages
  - Log Sequence Number (LSN) virtual offset within the redo log
  - mtr doesn’t have anything to do with ‘logical’ transactions
  - Releases the page latches too during mtr.commit().
  - LSN is generated after the changes to pages are added to the log buffer
- Flush list – dirty pages list
  - One list per buffer pool instances
  - Total order on log sequence number (LSN)
  - Oldest LSN in the list is the Low Water Mark (LWM)
    - All data pages with LSN < LWM **must have been flushed to disk**
  - LWM is the checkpoint LSN

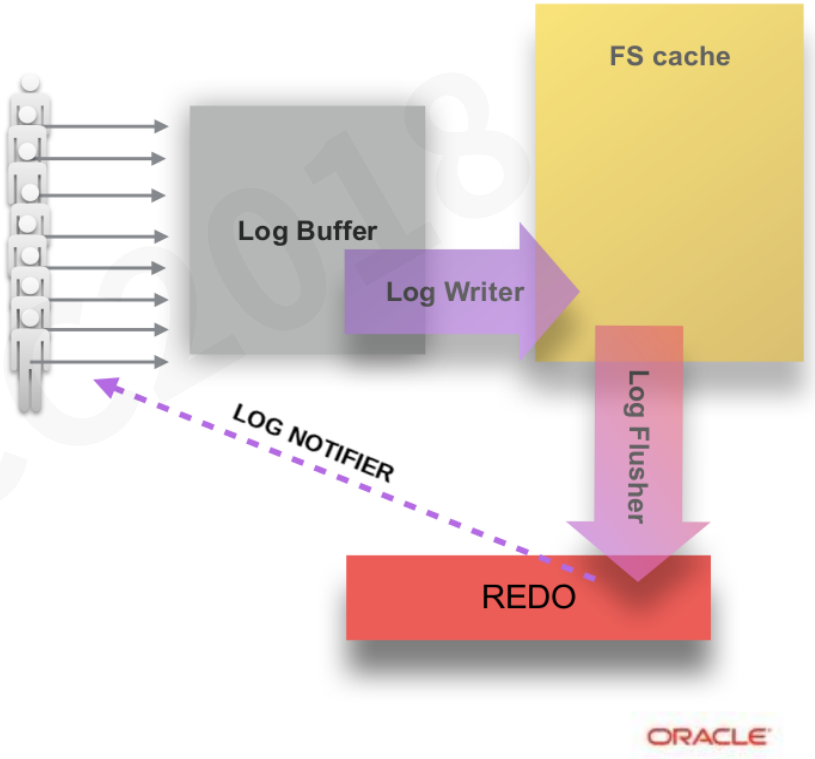
InnoDB concepts contd.

- The LSN is stamped on the data page header
- Redo is only written to when the server is running
- Redo is only read during crash recovery
- Redo log is circular
- Cannot wrap around and go past the Low Water Mark (LWM)
- Uses buffered IO on Unices
- Made up of multiple files of the same size
- Since 8.0 some logical changes too

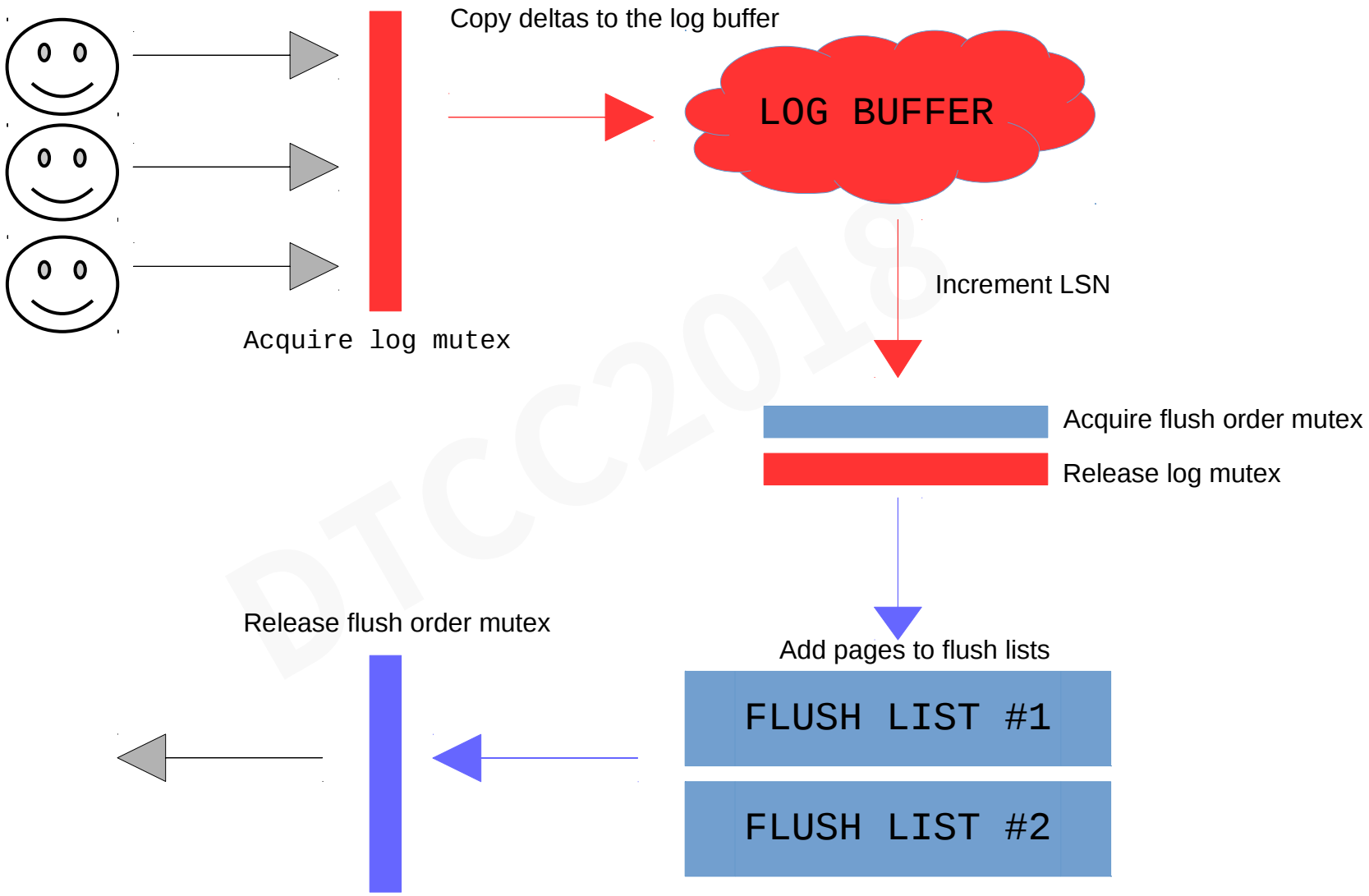
Old design, single mutex



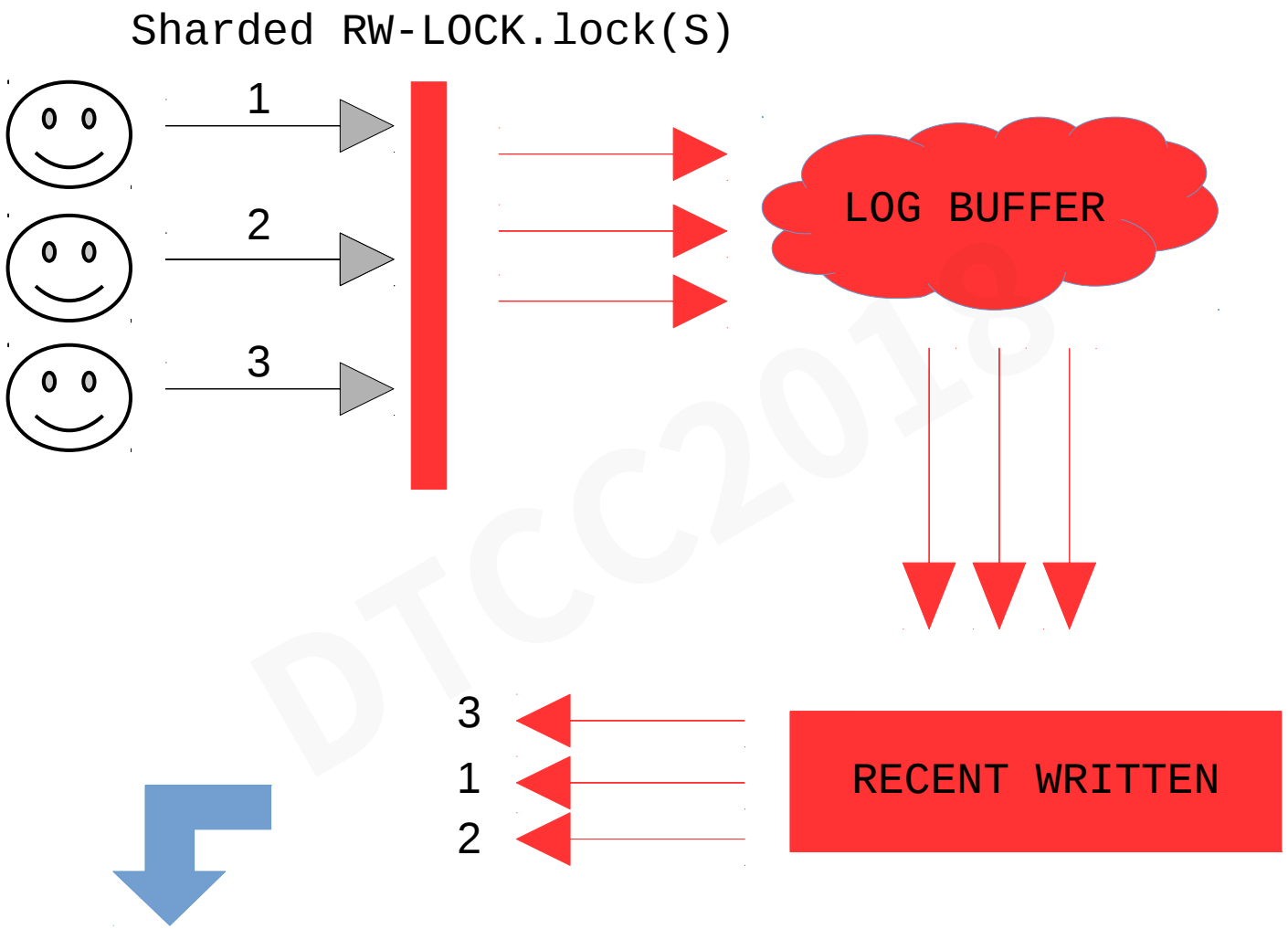
New design, lock free

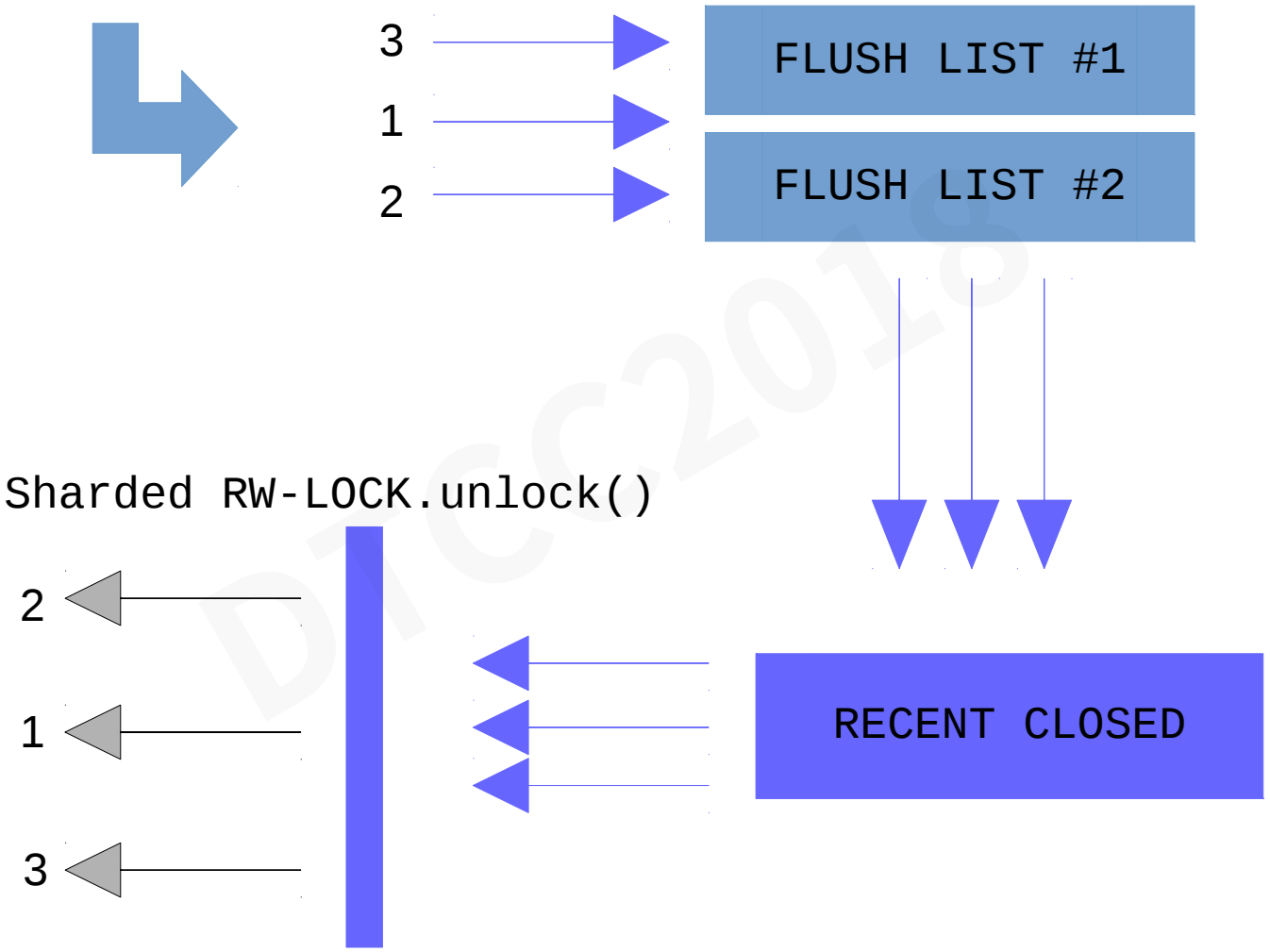


Old design – simpler but didn't scale well

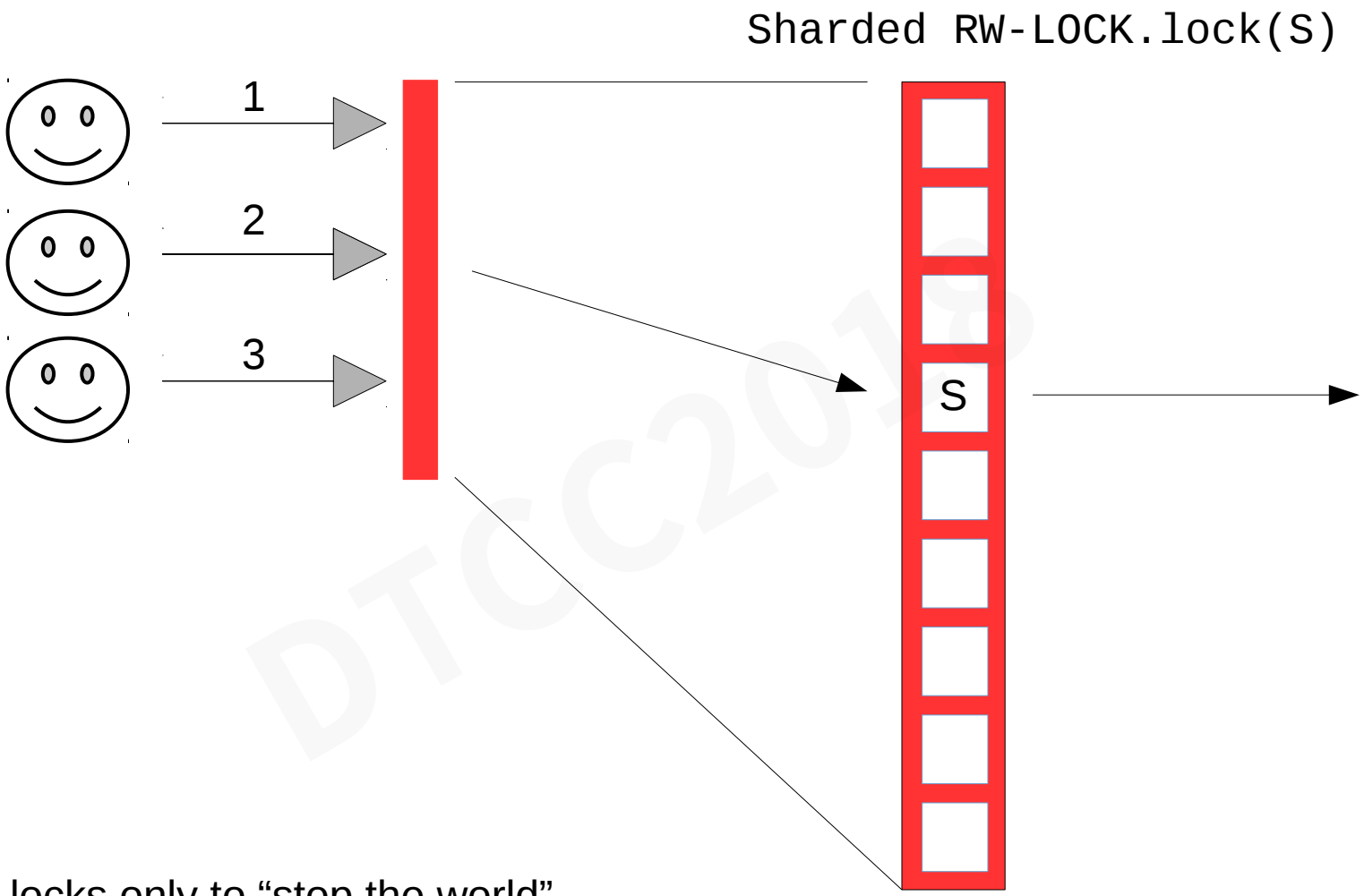


New design [1 / 2]



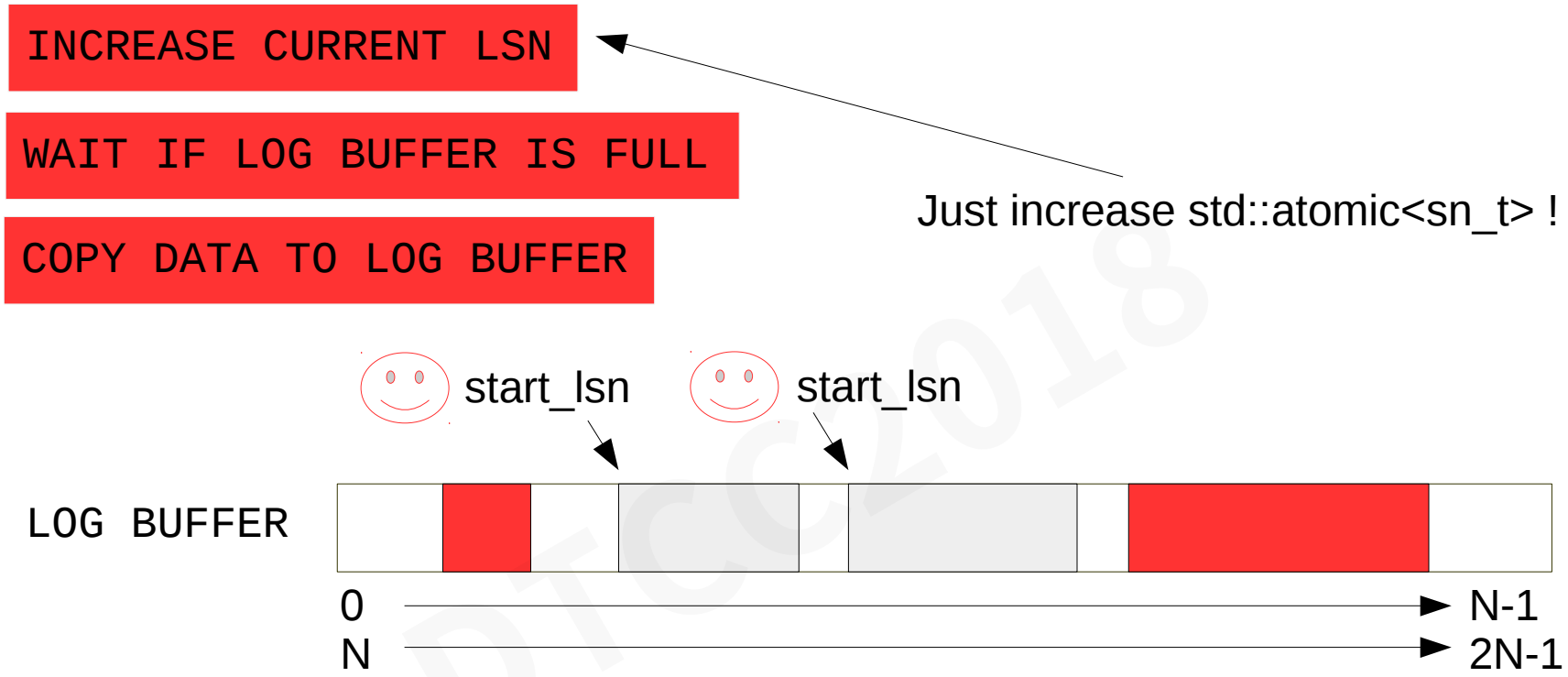


mtr.commit()



RW(S) locks only to “stop the world”

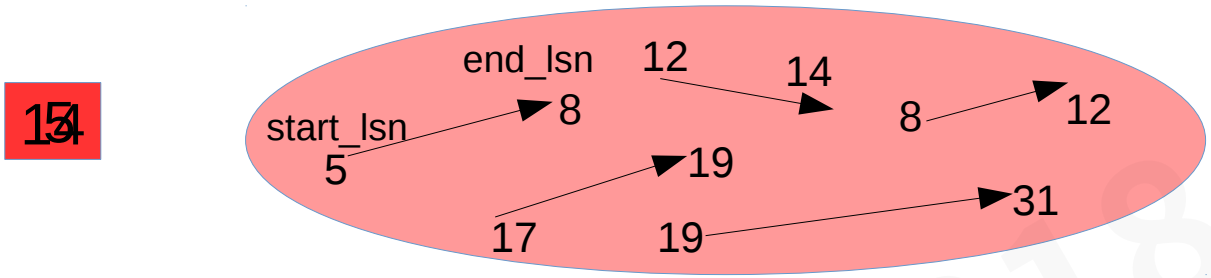
mtr.commit() - Write to log buffer



1. We no longer shift the log buffer when writing to log files.
2. Data for start_lsn..end_lsn now always starts at &log.buf[start_lsn % log.buf_size]
3. This allows concurrent writes to disjoint fragments (*there might be false sharing effect on the boundary between two consecutive fragments / two mtrs*).

1. Recall that we write data from log buffer to disk.
2. We have just allowed concurrent writes to the log buffer.
3. There is no guarantee that concurrent writes to the log buffer are finished in order of increasing LSN.
4. The reason we write to disk, is to advance LSN up to which all data is durable.
5. If we wanted to advance LSN to X, we must first ensure that all concurrent writes to the log buffer up to X are finished !
6. We need to track concurrent writes to be able to determine maximum such X, for which there are no holes in the log buffer (no pending concurrent writes to before X).

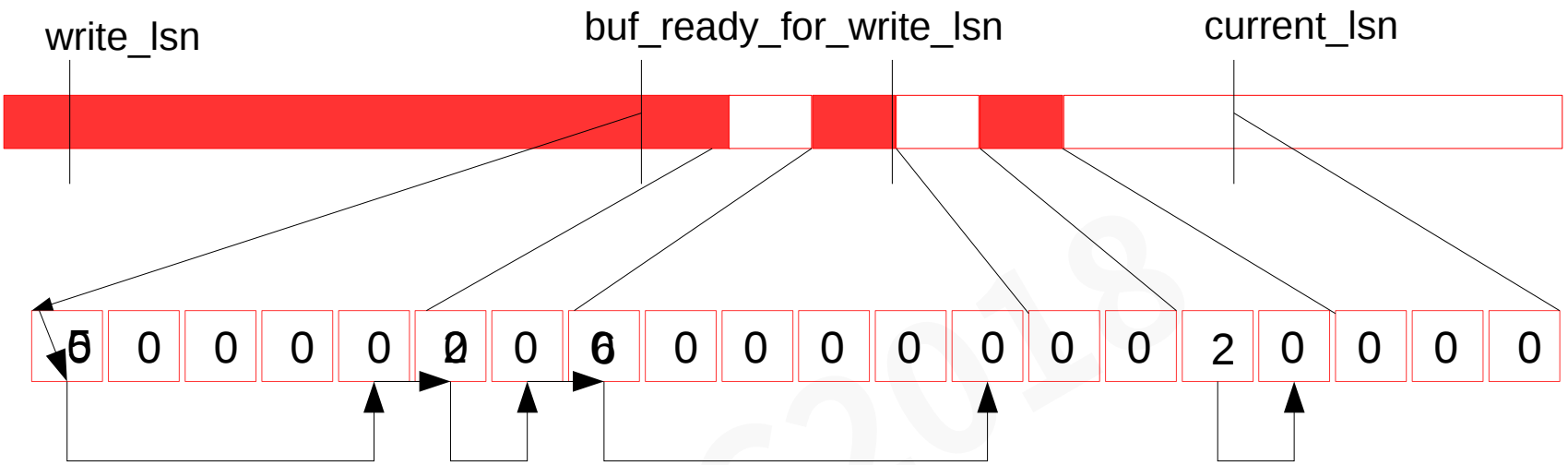
How could we efficiently track concurrent writes to buffer ?



NEW LOCK-FREE DATA STRUCTURE – RECENT_WRITTEN BUFFER CONCEPTUALLY:

- User thread add a link start_lsn→end_lsn, after it has finished copying log records to the log buffer (to start_lsn .. end_lsn).
- Different user threads add non-crossing links concurrently.
- Dedicated thread follows connected path of links, removes the links and updates the maximum LSN available for next write (*buf_ready_for_write_lsn)*.
- Data structure does not have to be big (1MB of recent redo is good enough) and can have limited space (disallowing bigger period with unfinished writes).

Details of the new lock-free data structure

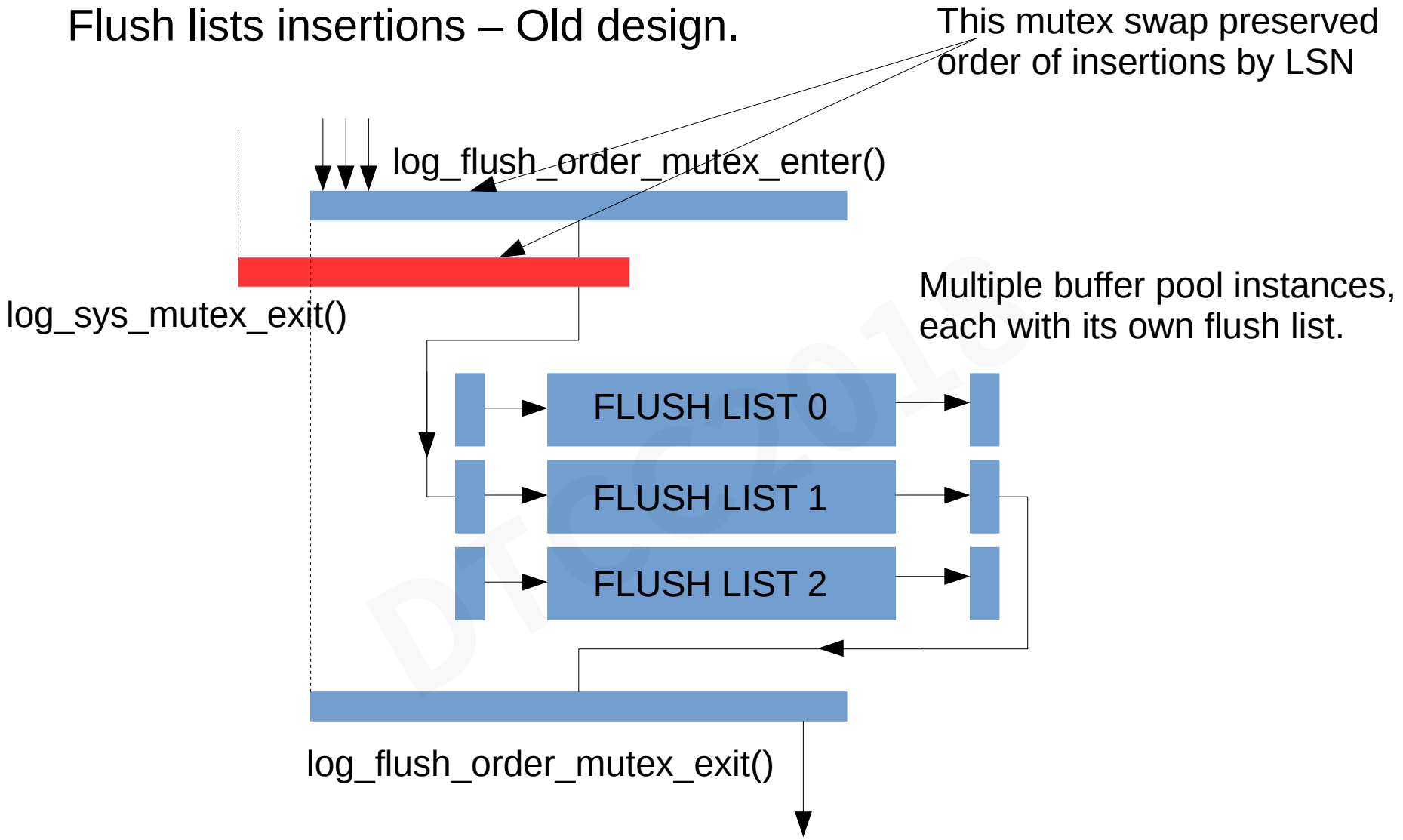


- Fixed size array of *M* slots with a std::atomic<lsn_t> in each slot.
- Link start_lsn → end_lsn is stored at start_lsn % *M* as value: end_lsn – start_lsn.
- User thread needs to wait if: start_lsn – buf_ready_for_write_lsn > M. Does not happen in practice.
- Each slot takes 8 bytes - cache-line is shared when mtr.commit() writes less than 8 bytes of data (8 * 8 = 64).
- Single thread removes the links and updates *buf_ready_for_write_lsn (log_writer)*.

The new data structure - reason for usage and gains:

- Allows to perform your own operations in out-of-order.
- Tracks which operations are finished.
- Limits window for pending operations up to size of the data structure.
- In other words – **relaxes the total order but limits how much it's relaxed.**
- That's enough in many of the cases for both performance and correctness.
- It allows to have concurrent log buffer…
- ...and it also allows to eliminate *log_flush_order_mutex,* and add dirty pages to different flush lists in parallel in concurrent mini transactions, and without dependency on order in which concurrent writes (to log buffer) are finished.

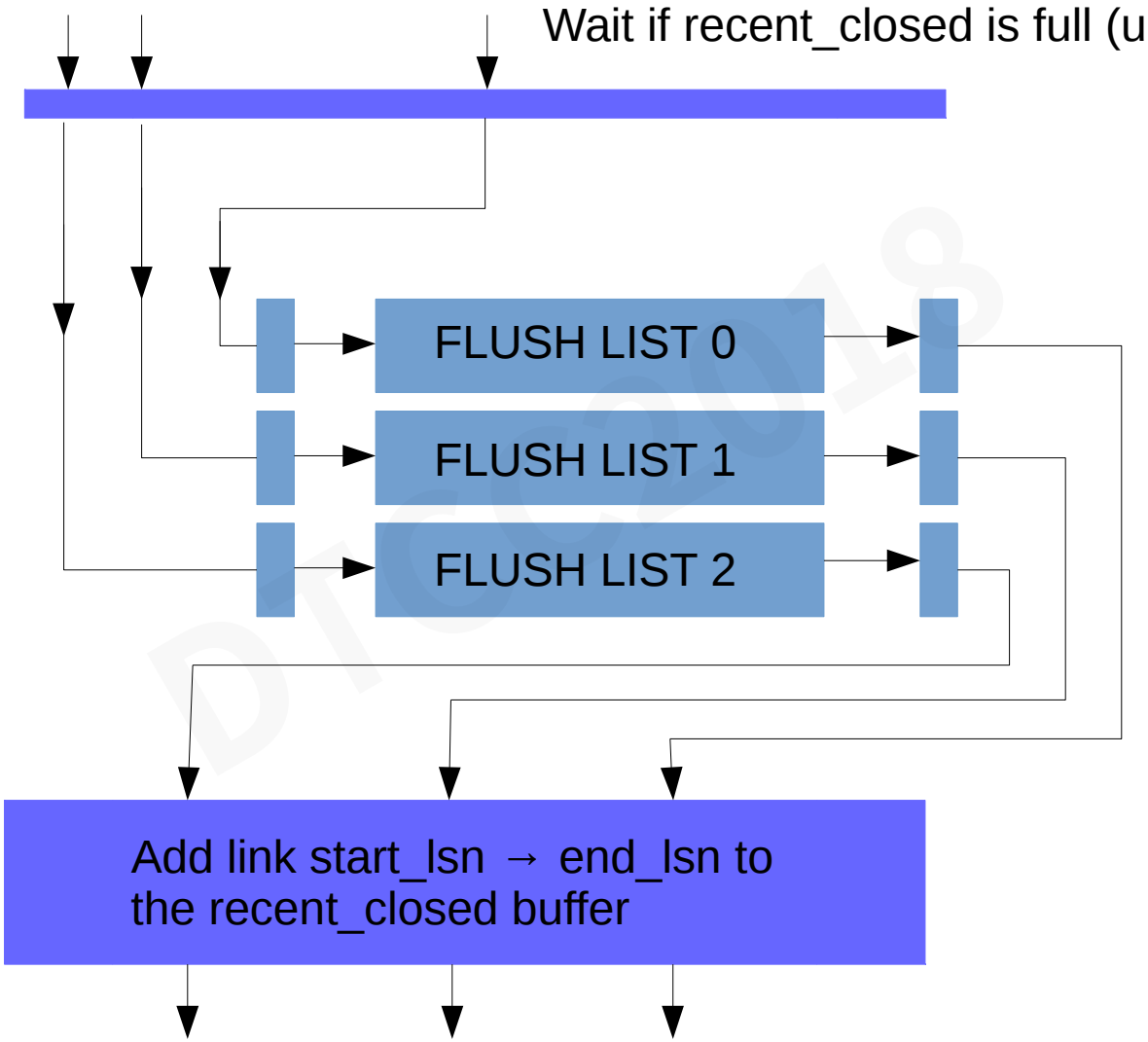
Flush lists insertions – Old design.



Flush list insertions – the new solution

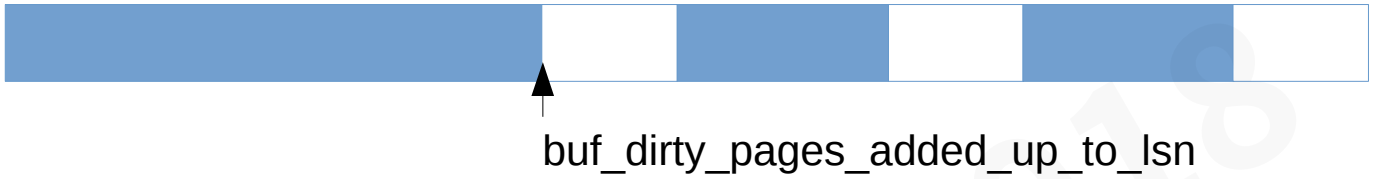
- Allow concurrent insertions to different flush lists.
- The order in flush lists is used to flush the pages starting from the oldest...
- … and now we do not try to preserve the total order of insertions by LSN...
- … but we allow the order to be disturbed only locally.
- Previously we had an invariant: if the last page has *oldest_modification* = *X*, then all pages in the flush list have *oldest_modification* >= *X*.
- Now we have relaxed property: if the last page has *oldest_modification* = *X,* then all pages in the flush list have *oldest_modification >= X – L*, where *L* is constant value (size of the recent closed buffer).
- Therefore the window of pending insertions to flush lists is limited by size of the recent_closed buffer.
- Writes to log buffer may finish in different order than insertions to flush lists.
- Single thread (*log_closer*) tracks up to which maximum *LSN* = *Y*, all pages with *oldest_modification* < *Y*, were added to flush lists.

Flush list insertions – new design

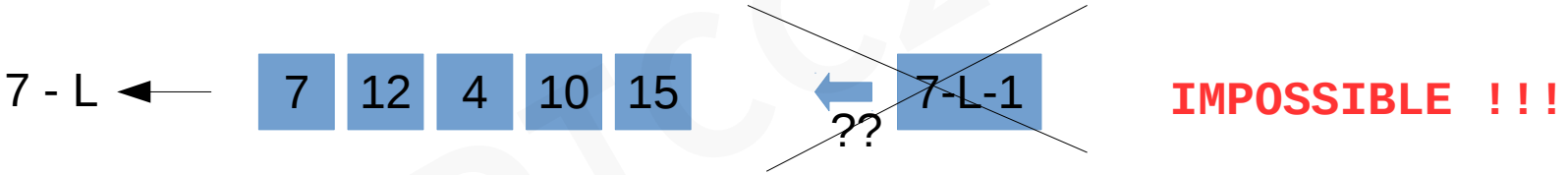


How to determine LSN available for next checkpoint ? (oldest_lsn)

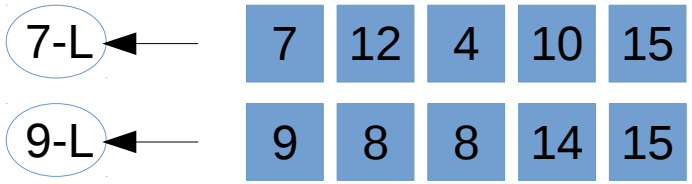
1. Check maximum LSN up to which all dirty pages have been added to flush lists – use the recent_closed buffer for that.



2. Take the first page from each flush list and subtract *L* from its *oldest_modification*



3. Take *X* ← minimum of these values and *buf_dirty_pages_added_up_to_lsn*.



4. Take maximum of: *X*, *last_checkpoint_lsn*.

Changes to redo recovery code.

- The *checkpoint_lsn* was previously chosen from the set of values assigned to *oldest_modification* of pages.
- These values are assigned to *start_lsn* during a first mtr which modifies the (non-dirty) page.
- Therefore the *checkpoint_lsn* pointed to the beginning of group of log records in that mini transaction.
- However, we subtract constant *L,* so this no longer holds.
- Actually the new *checkpoint_lsn* can point to the middle of some log record.
- The issue is resolved during recovery (thanks to *first_rec_group*).
- We need to find the first log block, which has a group of records, which starts after the *checkpoint_lsn* (or exactly at).
- Note, that this does not need to be the first group of records, which starts in a given log block.
- We need to parse redo log, starting from the *first_rec_group* in the first block which contains non-zero *first_rec_group* (at or after the block with *checkpoint_lsn*), but we must NOT re-apply records before checkpoint_lsn.

New redo background threads

Log_closer thread

- Traverses the recent_closed buffer.
- Removes the traversed links.
- Updates *buf_dirty_pages_added_up_to_lsn.*

Log_checkpointer thread

- Calculates LSN available for checkpoint (acquires flush list mutexes for that).
- Forces fsync() for all tablespace files.
- Writes checkpoint.
- Requests sync flush of dirty pages if needed.

Log_writer thread

- Traverses the recent_written buffer.
- Removes the traversed links.
- Updates *buf_ready_for_write_lsn.*
- Writes log buffer to files (OS buffer).

Log_write_notifier thread

- Notifies threads waiting for written log buffer (finished write).

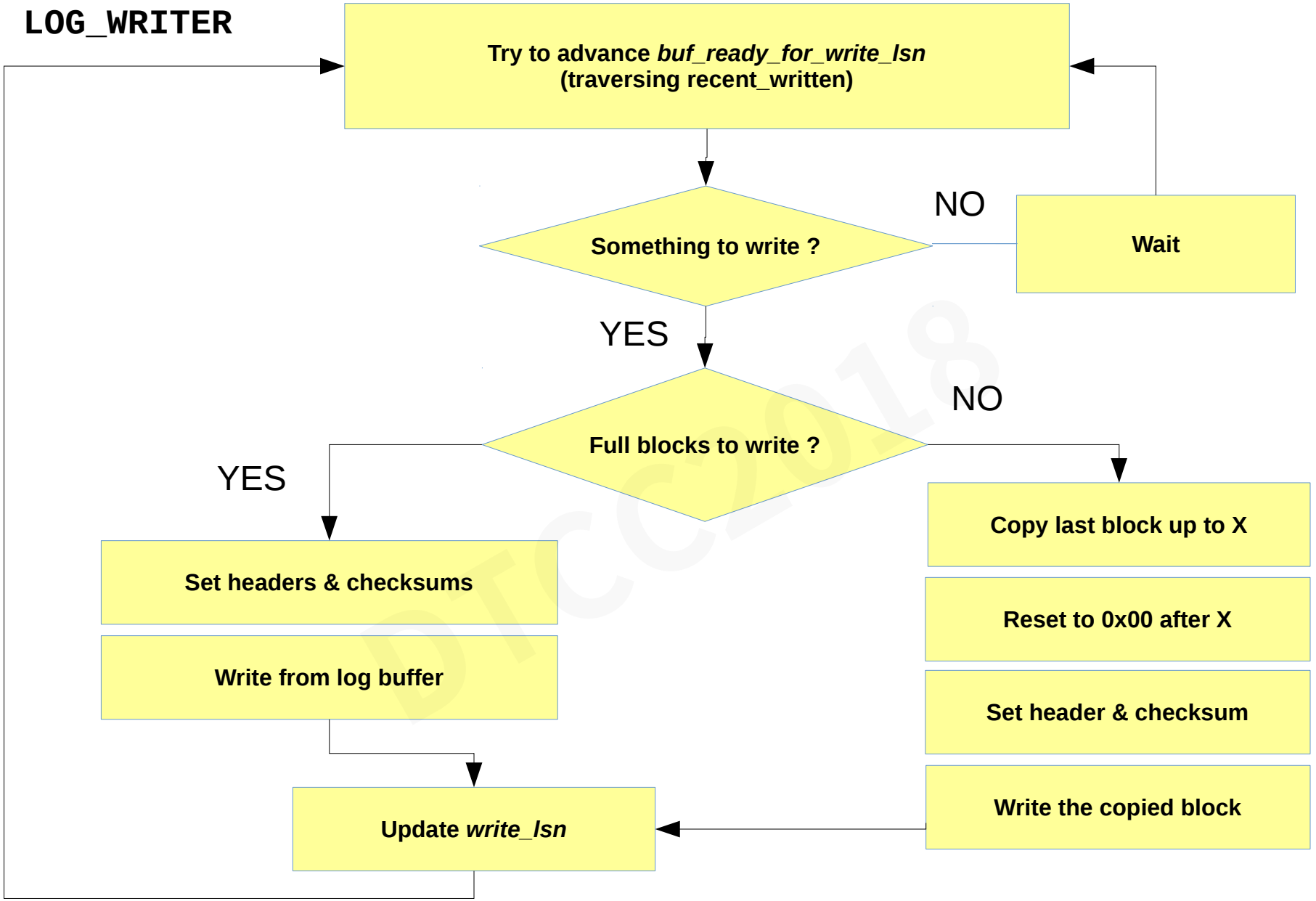
Log_flusher thread

- Executes fsync().

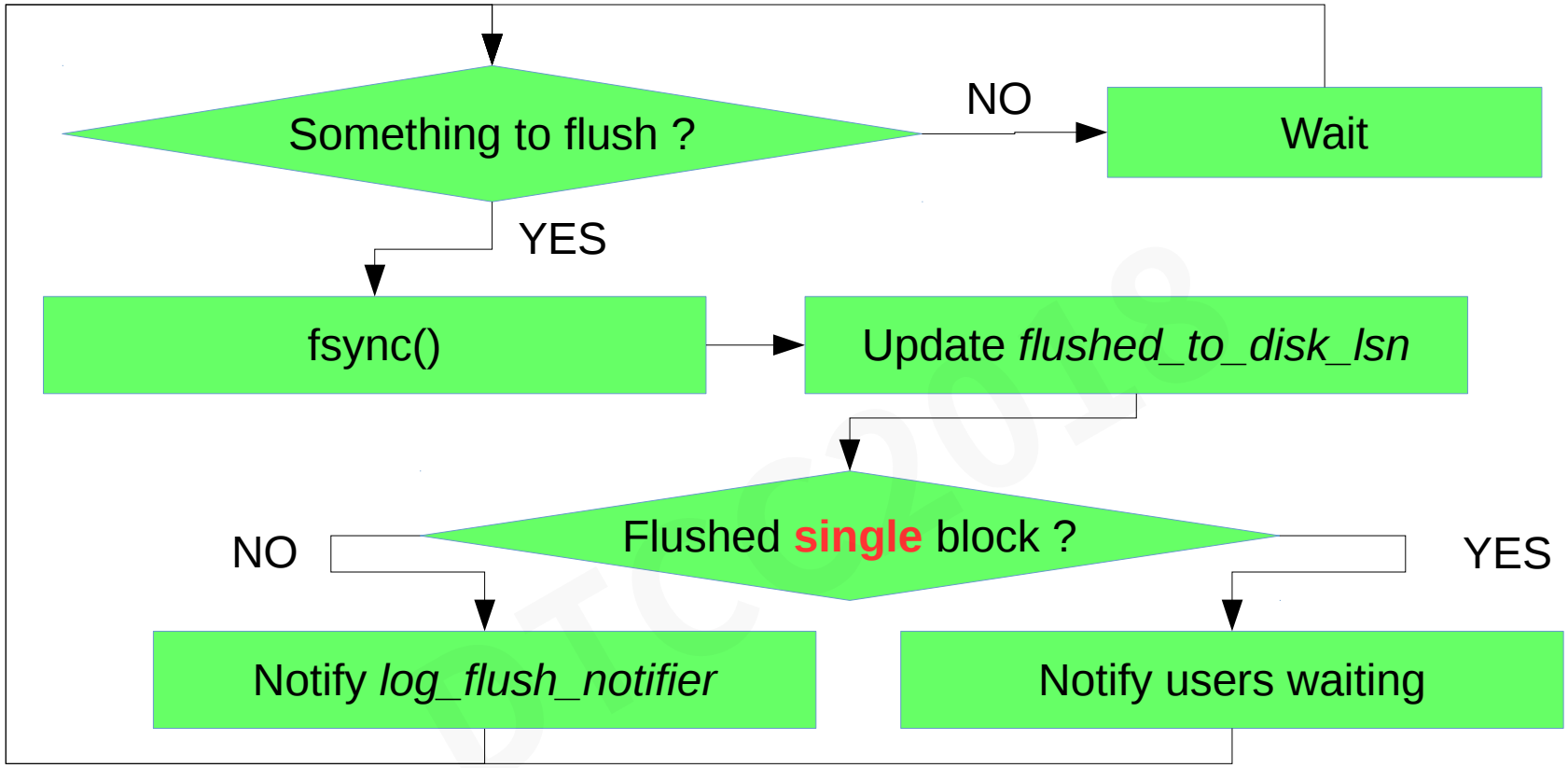
Log_flush_notifier thread

- Notifies threads waiting for flushed redo (finished fsync).

LOG_WRITER

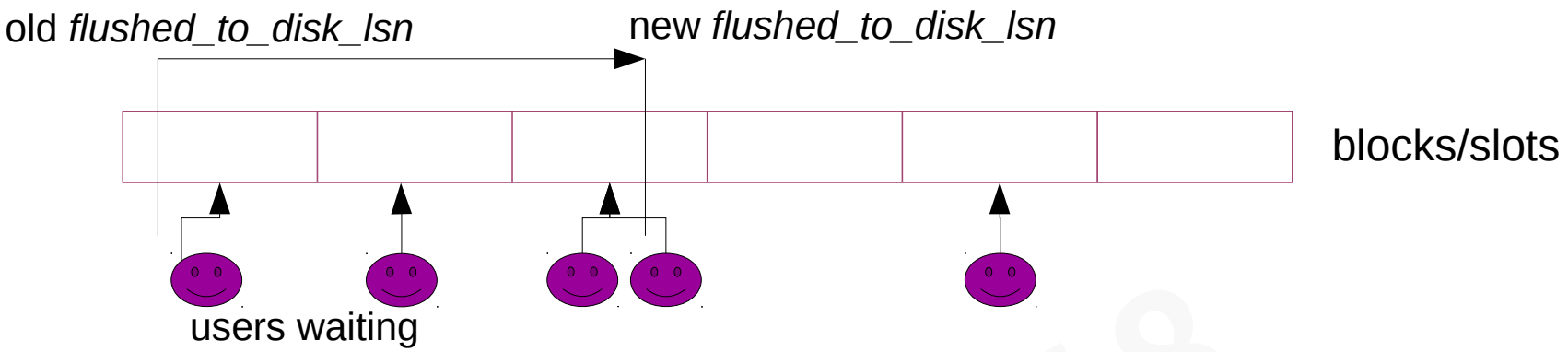


LOG_FLUSHER



For *innodb_log_flush_at_trx_commit* != 1, we ensure that 1sec left since previous *fsync()* before starting next *fsync()*.

LOG_FLUSH_NOTIFIER

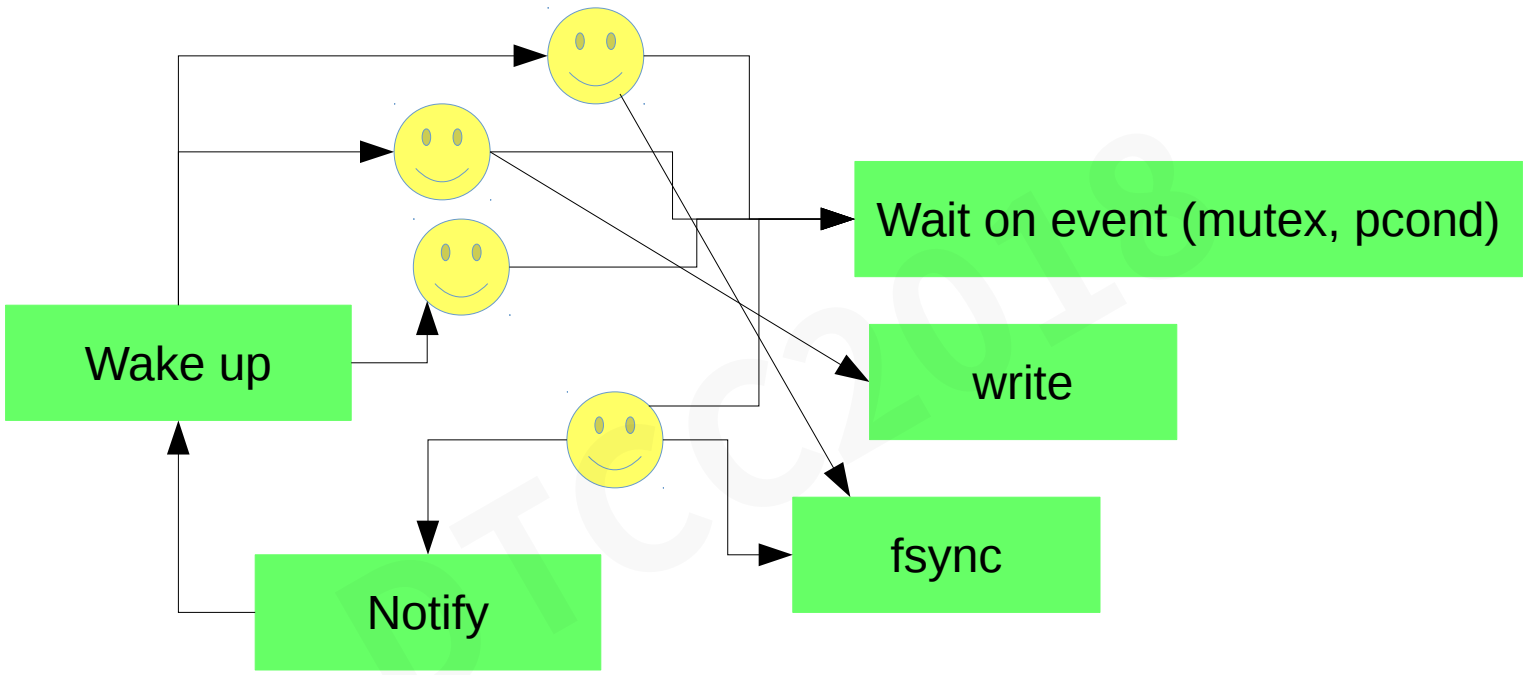


- Redo log is divided into blocks of 512 bytes.
- We assign consecutive blocks to consecutive slots (and wrap):

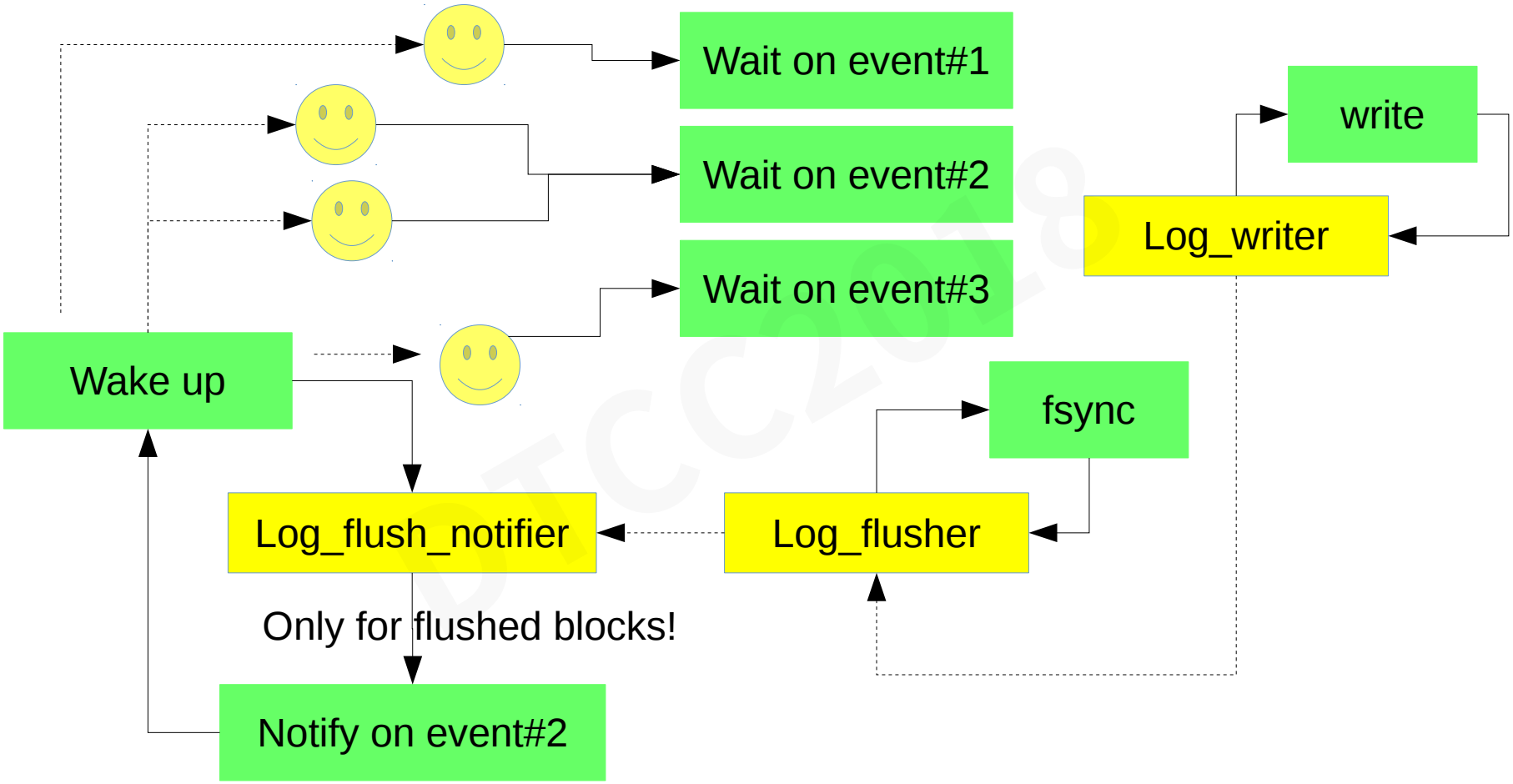
  slot = (lsn – 1) / 512 % N
- Each slot has single *os_event_t.*
- Users waiting for flushed redo, use proper slot to wait.
- When *flushed_to_disk_lsn* became advanced, slots corresponding to intermediate blocks are notified by log_flush_notifier.
- False wake-ups are allowed (user will retry wait), but we may not miss required wake-ups.
- Note, in the worst case, at most *N* slots have to be notified.

Commit of transaction – how it was before.

(innodb_flush_log_at_trx_commit = 1)



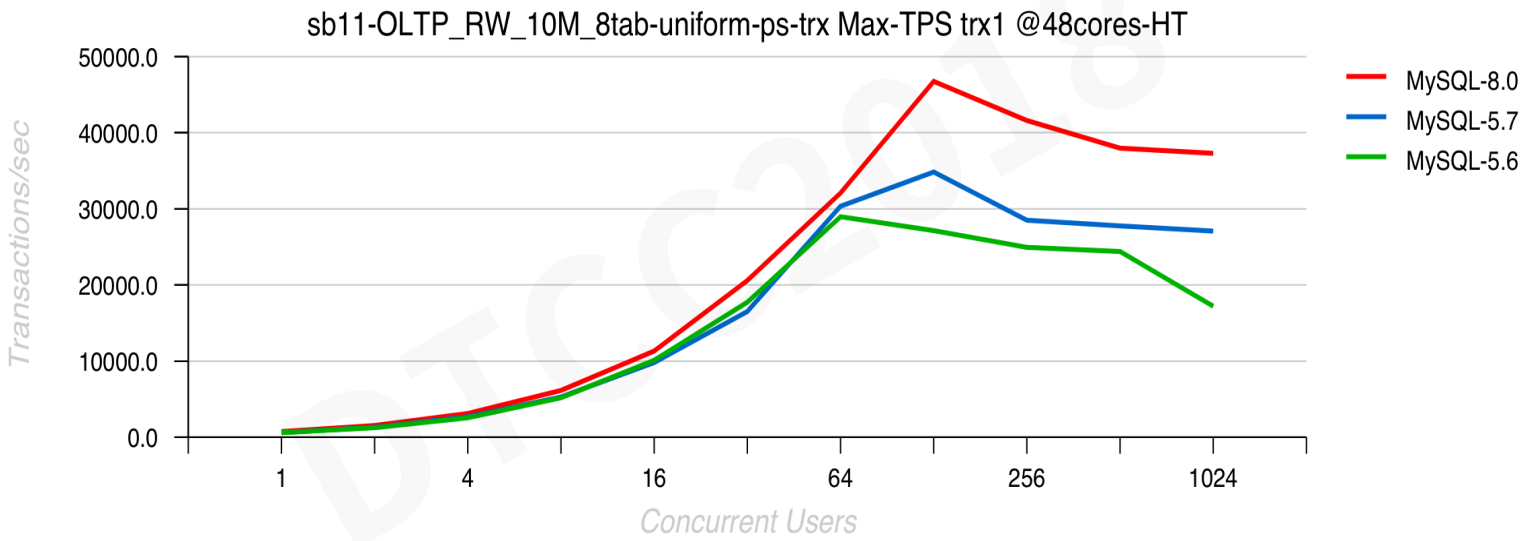
Commit of transaction – the new solution.



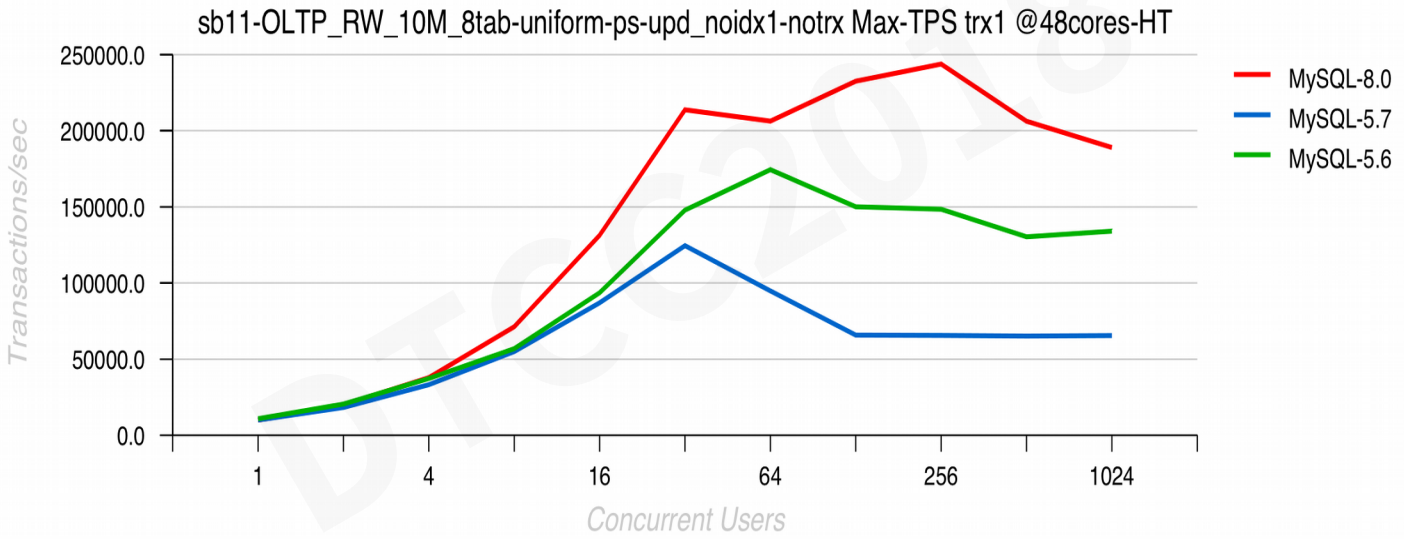
Server Configuration

- OS : Oracle Linux 7.4
- CPU : 48cores-HT Intel Skylake 2.7Ghz
- 2CPU sockets, Intel(R) Xeon(R) Platinum 8168 CPU
- RAM: 256GB
- Storage : x2 Intel (R) Optane (TM) SSD P4800X Series)
- Volume : RAID-0 via MDADM
- Filesystem : EXT4

Sysbench: OLTP RW 10m rows x 8 tables trx_commit=1



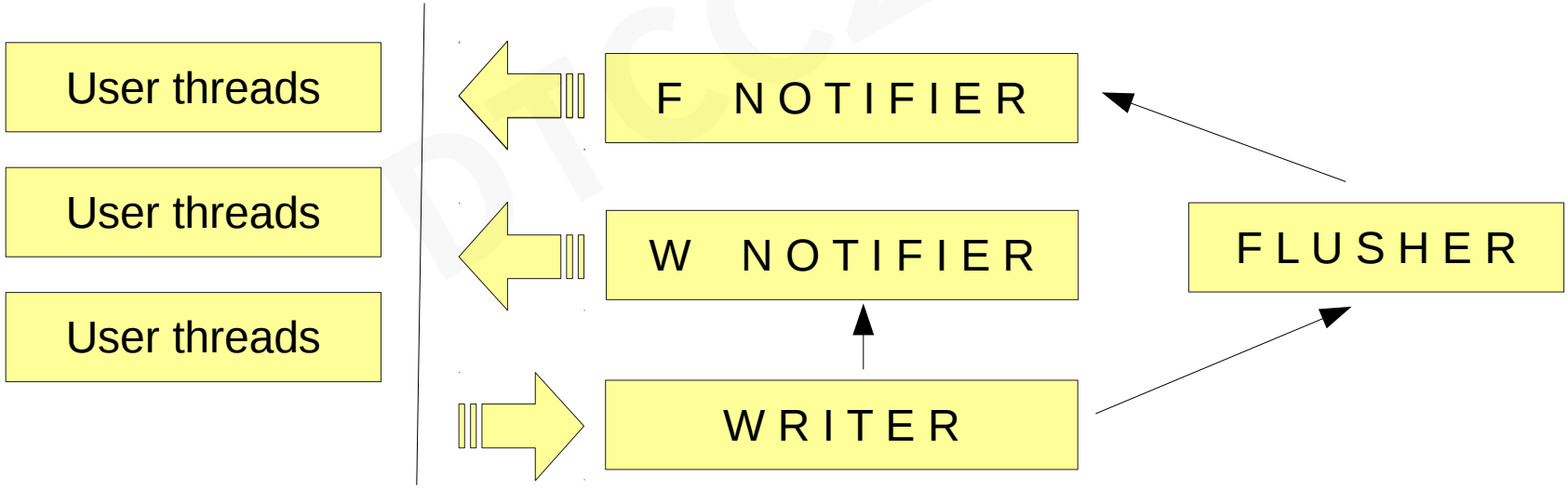
Sysbench: Update no index 10m rows x 8 tables trx_commit=1



LET'S SUMMARIZE

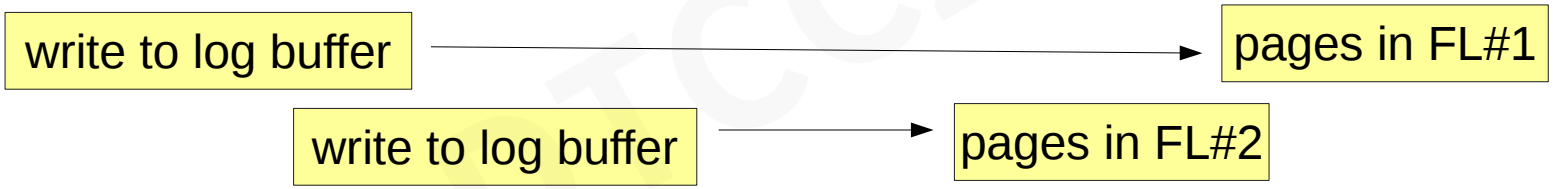
Dedicated redo log threads

- *log_writer* - Writes from log buffer to file
- *log_flusher* - Executes fsync()
- *log_write_notifier* - Notifies user threads about finished writes
- *log_flush_notifier* - Notifies user threads about finished fsync
- *log_checkpointer* - Writes checkpoints
- *log_closer* - Maintains limit for disorder in flush lists



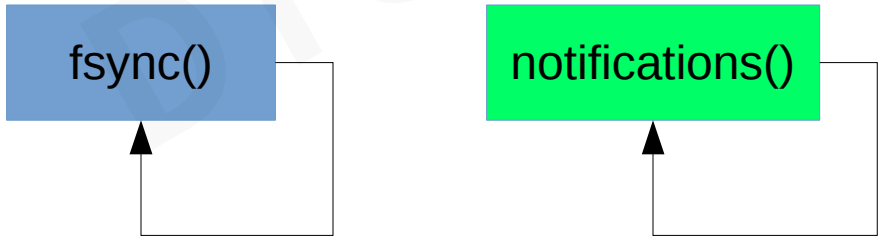
Increased concurrency of mtr commits

- Concurrent mtr commits may interleave and end in any order
- Order of dirty pages might be locally distorted
- Removed *log_flush_order* mutex
- Removed *log_sys* mutex



Decreased latency between: fsync()→trx committed

- Event per log block (instead of single global event)
- Less synchronization between user threads
- Only users waiting for LSN within flushed blocks are notified
- … others are not woken up ! 
- Dedicated thread for notifications
- … delay of the next *fsync()* start is decreased



Thank you!

To be continued…

讲师申请
联系电话（微信号）：18612470168

关注“ITPUB”更多
技术干货等你来拿~

与百度外卖、京东、魅族等先后合作系列分享活动

微学堂

让学习更简单

微学堂是以ChinaUnix、ITPUB所组建的微信群为载体，定期邀请嘉宾对热点话题、技术难题、新产品发布等进行移动端的在线直播活动。

截至目前，累计举办活动期数60+，参与人次40000+。

ITPUB学院

ITPUB学院是盛拓传媒IT168企业事业部（ITPUB）旗下
企业级在线学习咨询平台
历经18年技术社区平台发展
汇聚5000万技术用户
紧随企业一线IT技术需求
打造全方式技术培训与技术咨询服务
提供包括企业应用方案培训咨询（包括企业内训）
个人实战技能培训（包括认证培训）
在内的全方位IT技术培训咨询服务

ITPUB学院讲师均来自于企业
一些工程师、架构师、技术经理和CTO
大会演讲专家1800+
社区版主和博客专家500+

培训特色

无限次免费播放
随时随地在线观看
碎片化时间集中学习
聚焦知识点详细解读
讲师在线答疑
强大的技术人脉圈

八大课程体系

基础架构设计与建设
大数据平台
应用架构设计与开发
系统运维与数据库
传统企业数字化转型
人工智能
区块链
移动开发与SEO

联系我们

联系人：黄老师
电　话：010-59127187
邮　箱：edu@itpub.net
网　址：edu.itpub.net
培训微信号：18500940168