Innodb Architecture and Performance Optimization MySQL 5.7 Edition

Peter Zaitsev

PgDay Russia 2017, Saint Petersburg | July 5th, 2017

PERCONA LIVE

Percona Live Europe Call for Papers & Registration are Open!

Championing Open Source Databases

- MySQL, MongoDB, Open Source Databases
- Time Series Databases, PostgreSQL, RocksDB
- Developers, Business/Case Studies, Operations
- September 25-27th, 2017
- Radisson Blu Royal Hotel, Dublin, Ireland

Submit Your Proposal by July 17th!

www.percona.com/live/e17

Why Together ?

Advanced Performance Optimization Needs Architecture Knowledge

Right Level

Focus on Details What Matter

Wonder where Graphs are from ?

Graphs from Percona Monitorign and Management

Explore Online Demo at http://pmmdemo.percona.com

What about MySQL 8.0 ?

This is what was previously called MySQL 5.8

Only 2nd Milestone release is available

Innodb Features are still evolving

What Architecture ?

Data Structures

- On Disk
- In Memory

Details for

- Transactions
- MVCC
- Locking
- Latching

Background Activities

- Purging
- Checkpointing
- Flushing

Innodb Versions Covered

MySQL 5.7 as Baseline

Improvements in XtraDB / Percona Server

Changes in MariaDB

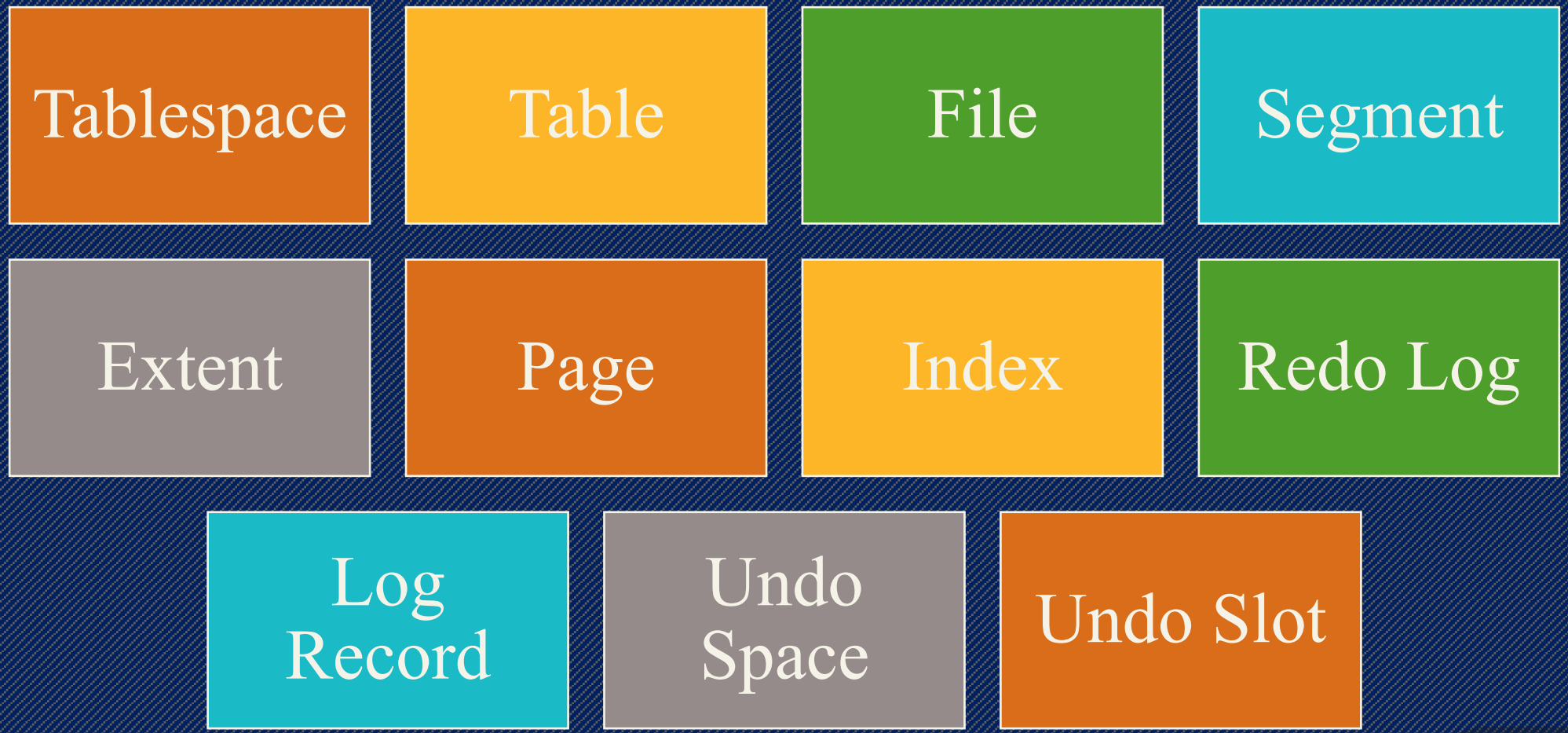
Innodb Basics

"Traditional" Storage Engine

- B+Tree Based
- ACID Transactions
- MVCC
- OLTP Optimized

Data Structures

Main "Objects"



Innodb Main Files

Tablespace Files

Log Files

Tablespace

- All Data Stored In Tablespaces
- System Tablespace
- Per-Table Tablespace
- Undo Tablespace(s) MySQL 5.6+
- Temporary Tablespace MySQL 5.7+
- General Tablespaces MySQL 5.7+

Performance Considerations

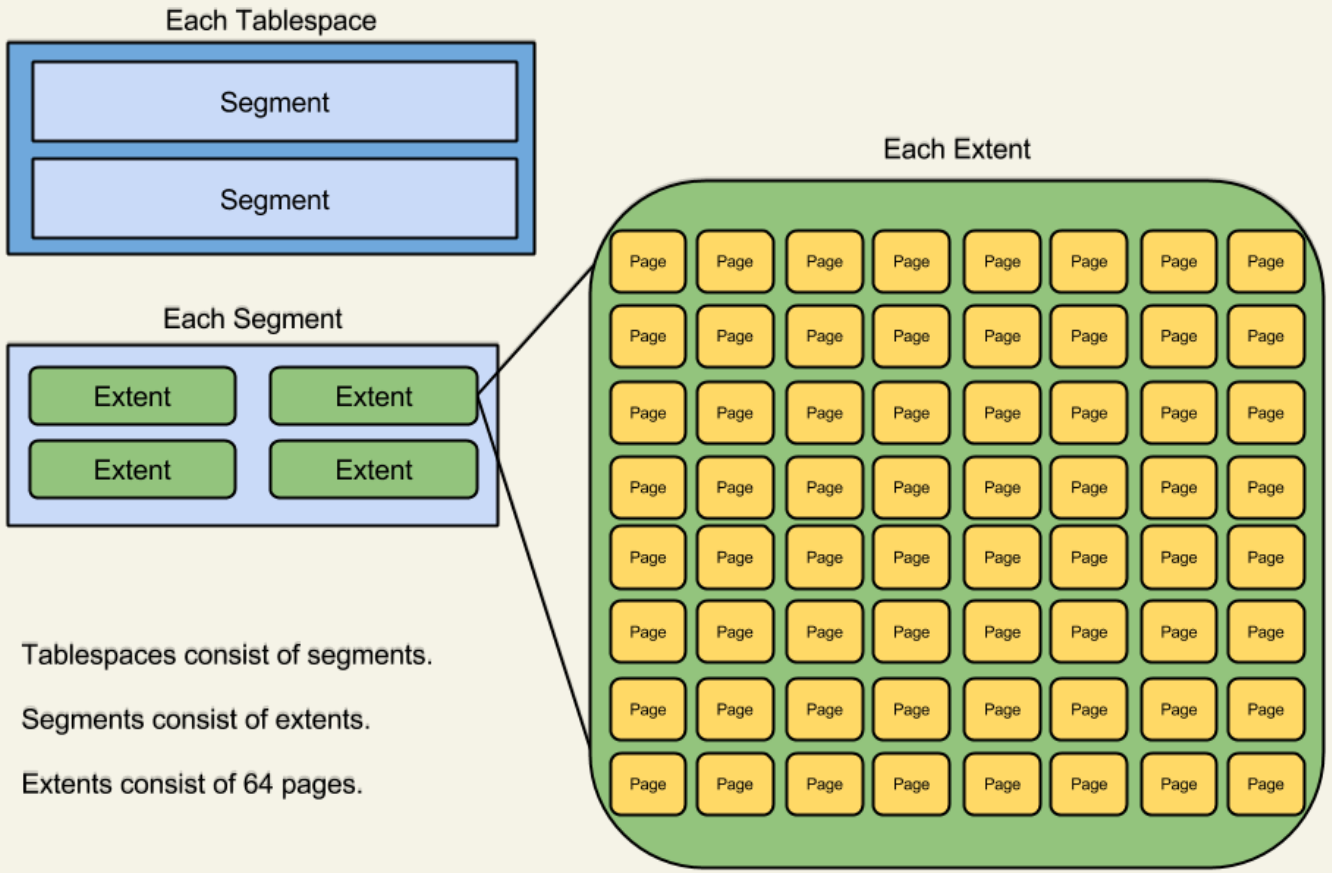
Innodb_file_per_table is default in MySQL 5.6+

Can cause problems with many tables

Can cause problems with many CREATE/DROP/TRUNCATE

Improved in MySQL 5.7

Tablespace: Physical Structure



15

Table

Stored In Tablespace

Consists of Indexes

May have many partitions

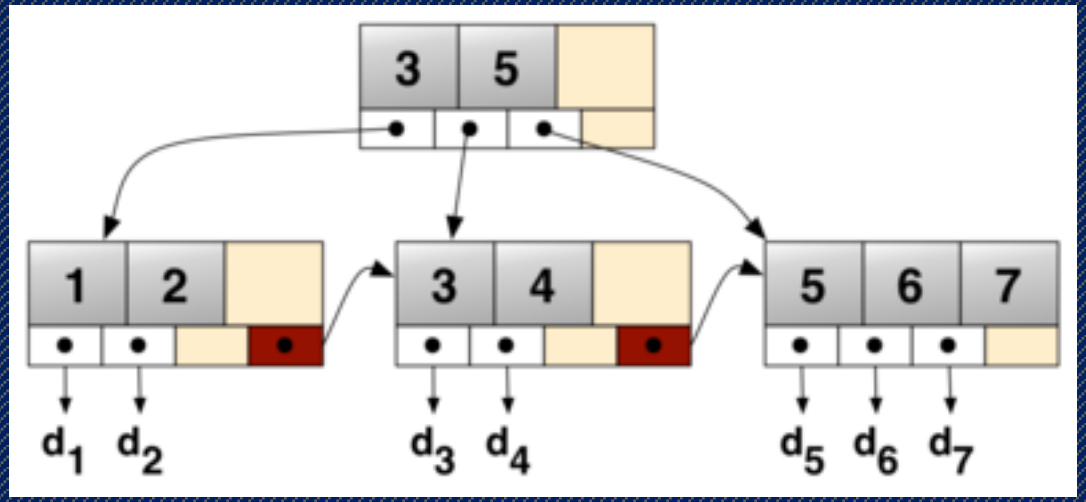
Table Consists of Indexes ?

Data is stored in CLUSTERED Index

PRIMARY KEY or hidden

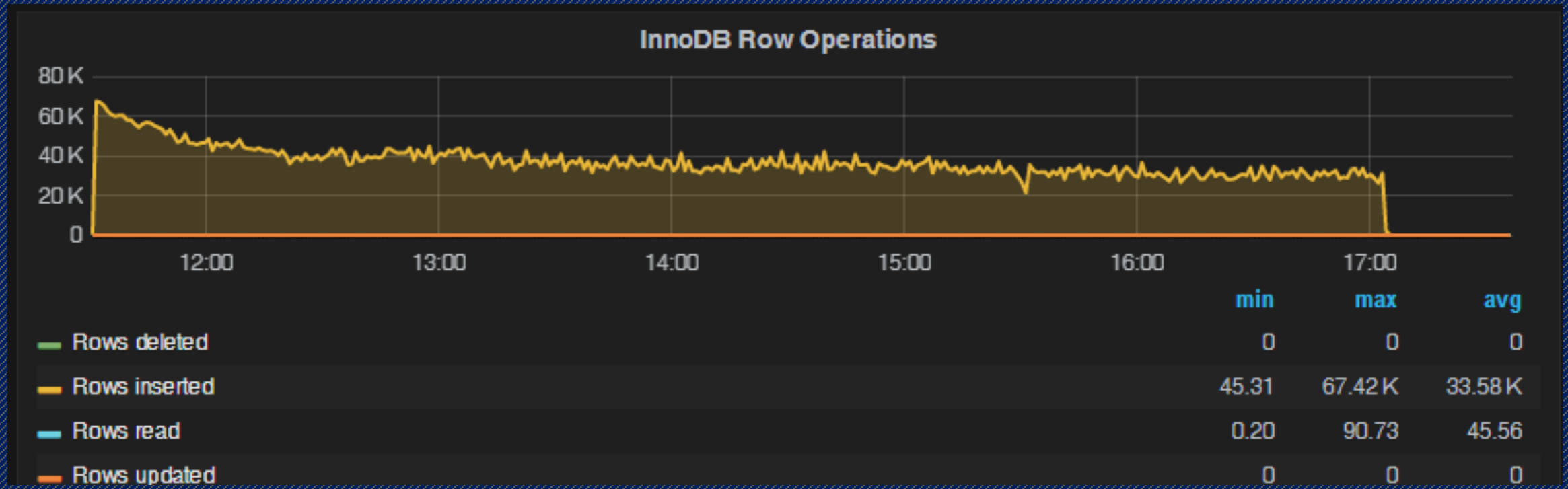
Secondary keys point to PRIMARY KEY

Index is B+Tree



File IO during INSERT

Row Operations During Insert



20

Physical and Logical Structure

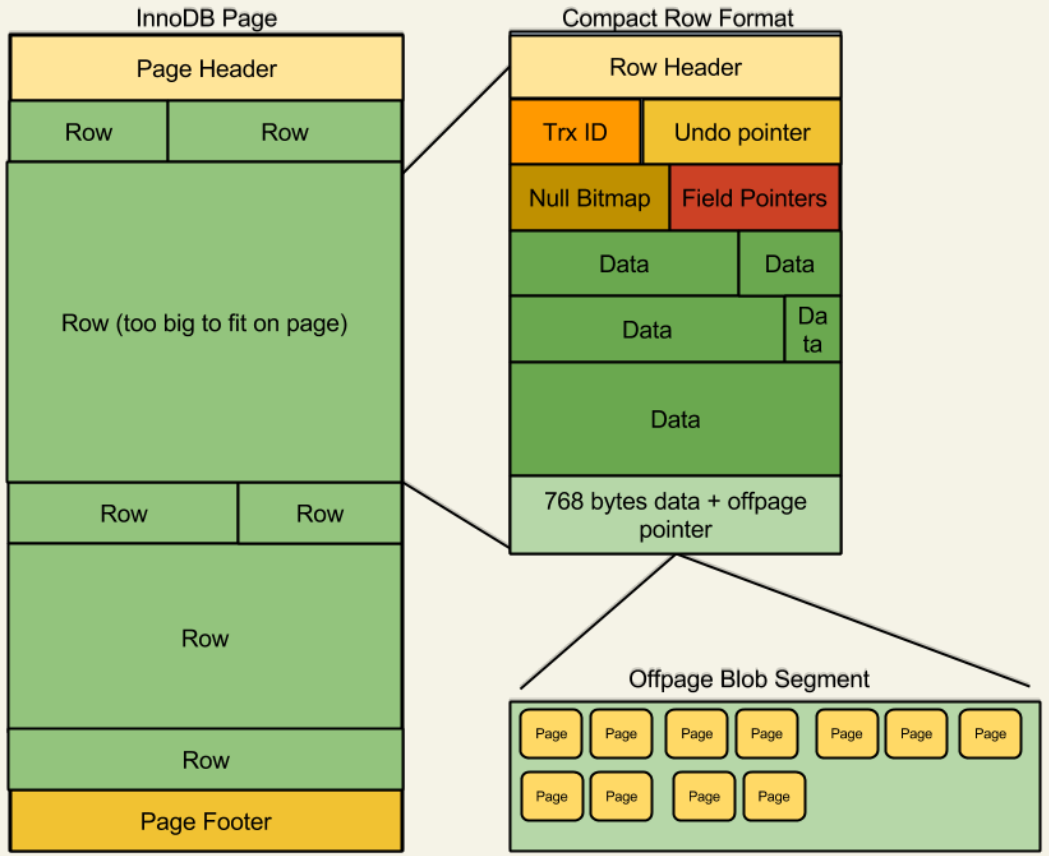
Segments are Similar to Files

Each Index has 2 segments

Leaf Page Segment

Non-Leaf Page Segment

Page Details



22

Performance Considerations

Have **PRIMARY KEY**

Short **PRIMARY KEYs** are better

Sequential inserts are much faster for **PRIMARY KEY**

Great to keep Secondary Keys in Memory

Redo Logs

2 (or more) files concatenated as redo log of fixed size

Log consists of records (512b aligned)

“Physio-logical” Redo record format

Every change to Tablespace must be recorded in redo log before it is done (except temporary tablespace in MySQL 5.7)

Redo log buffering

Buffered in Log Buffer

Optionally Flushed at Transaction Commit

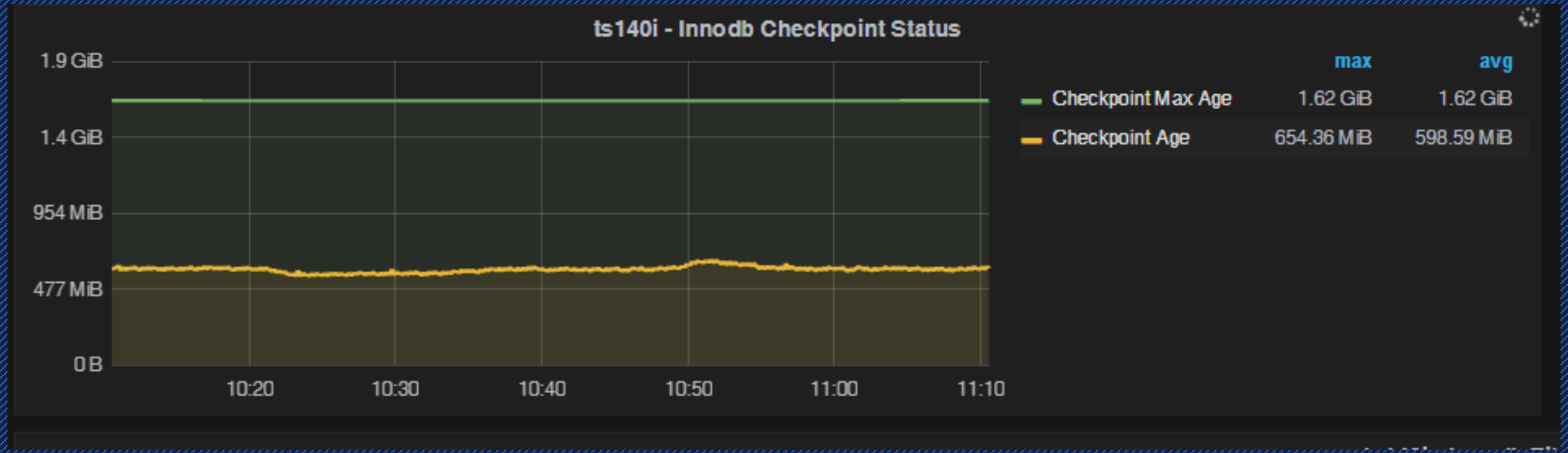
Flushed Periodically

Performance Considerations

Innodb_log_file_size	• to control size of redo log
innodb_log_write_ahead_size	• Avoid ‘read-around-write’ for large log files not in cache
Larger log size	• better write performance (less flushing) • longer recovery time

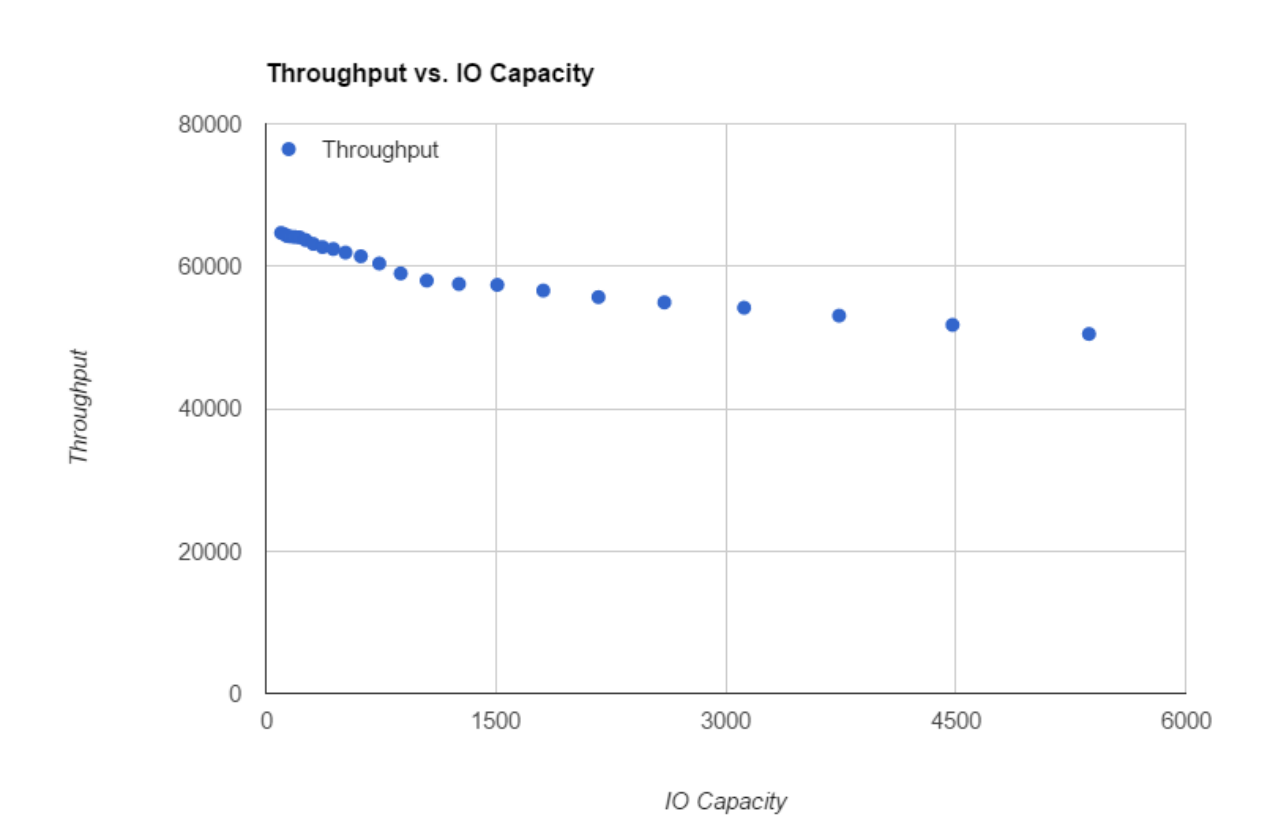
Innodb Checkpointing

How Much redo log space is used

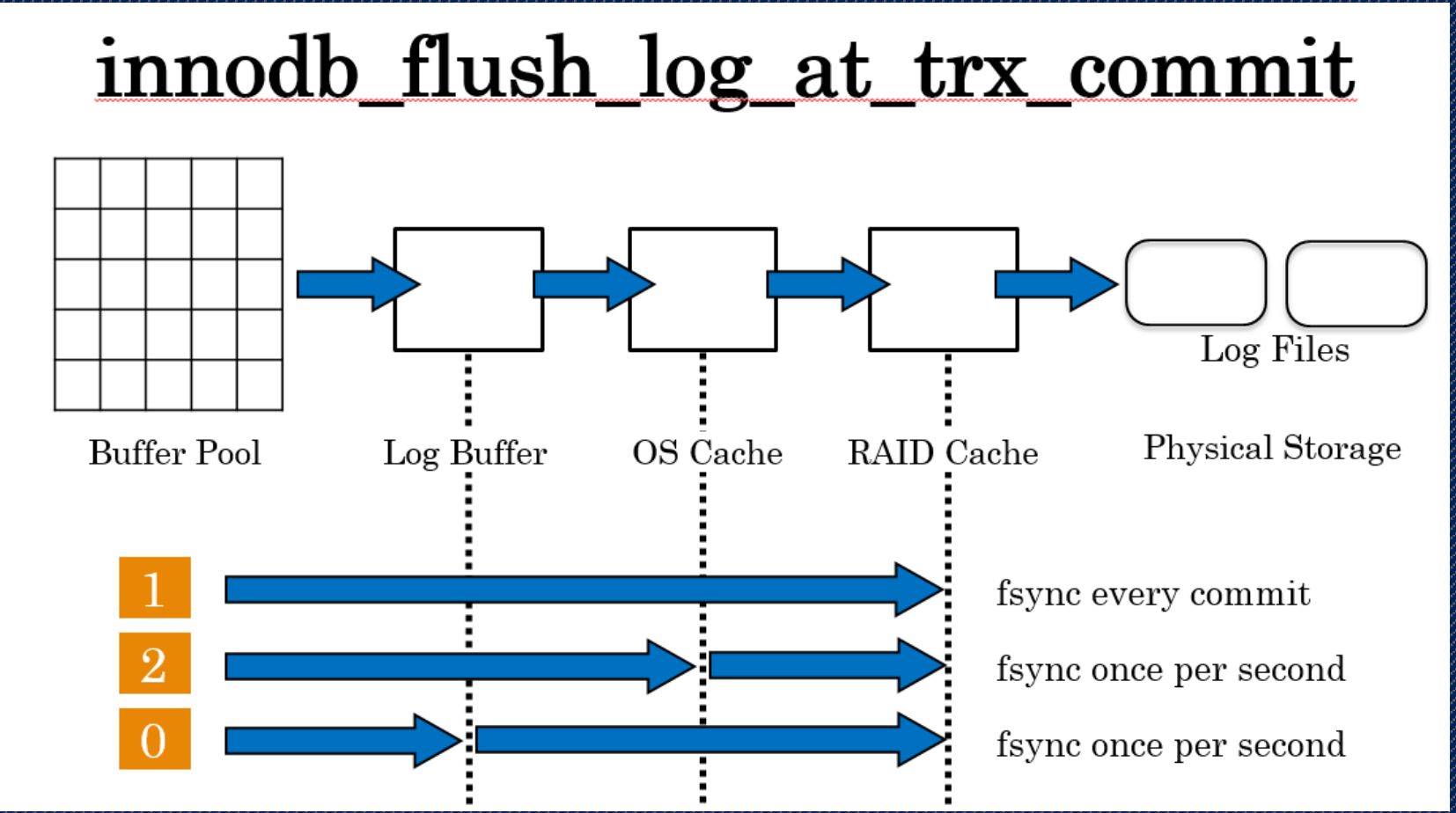


Mind innodb_io_capacity

- Recommended Setting can decrease performance

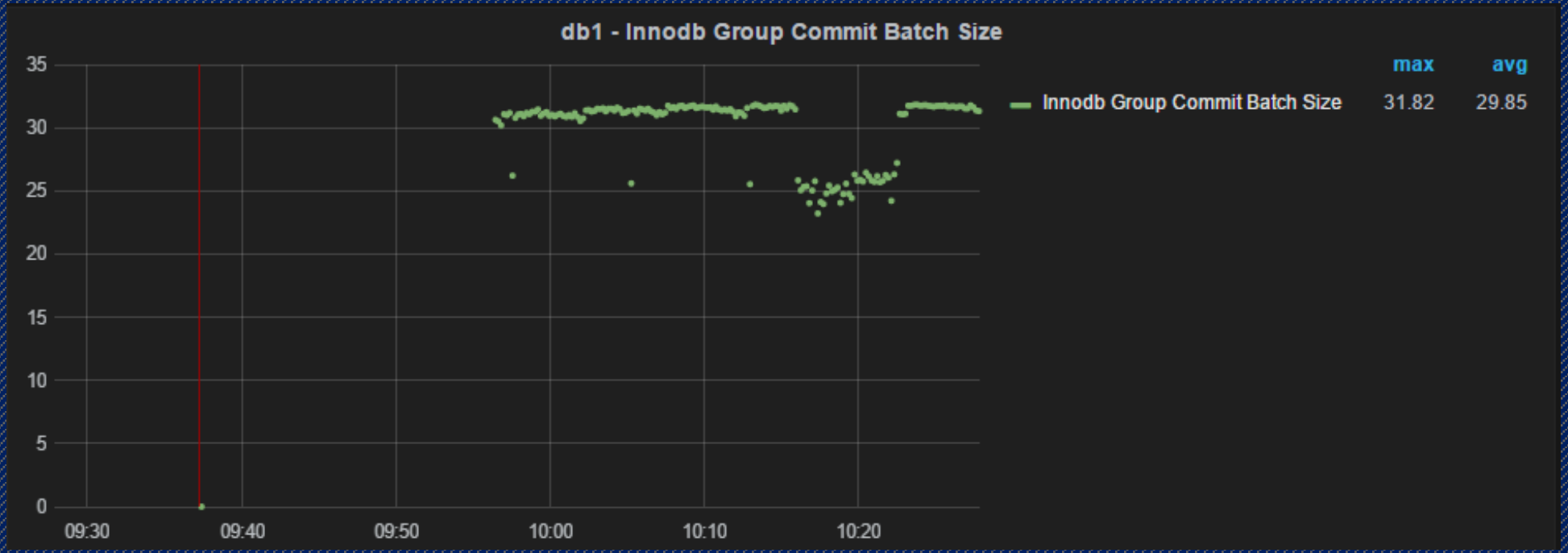


How Durable Transactions do you Need ?



Mind Group Committ

- Measuring transaction commits vs log writes



Double write Buffer

- Page Flushes are not guaranteed atomic
- Flush Pages to “double write buffer” and again in real locations
- Can get expensive (especially SSD)
- Can disable if file system guarantees atomicity
- MySQL 5.7 automatically does for Sandisk NVMFS

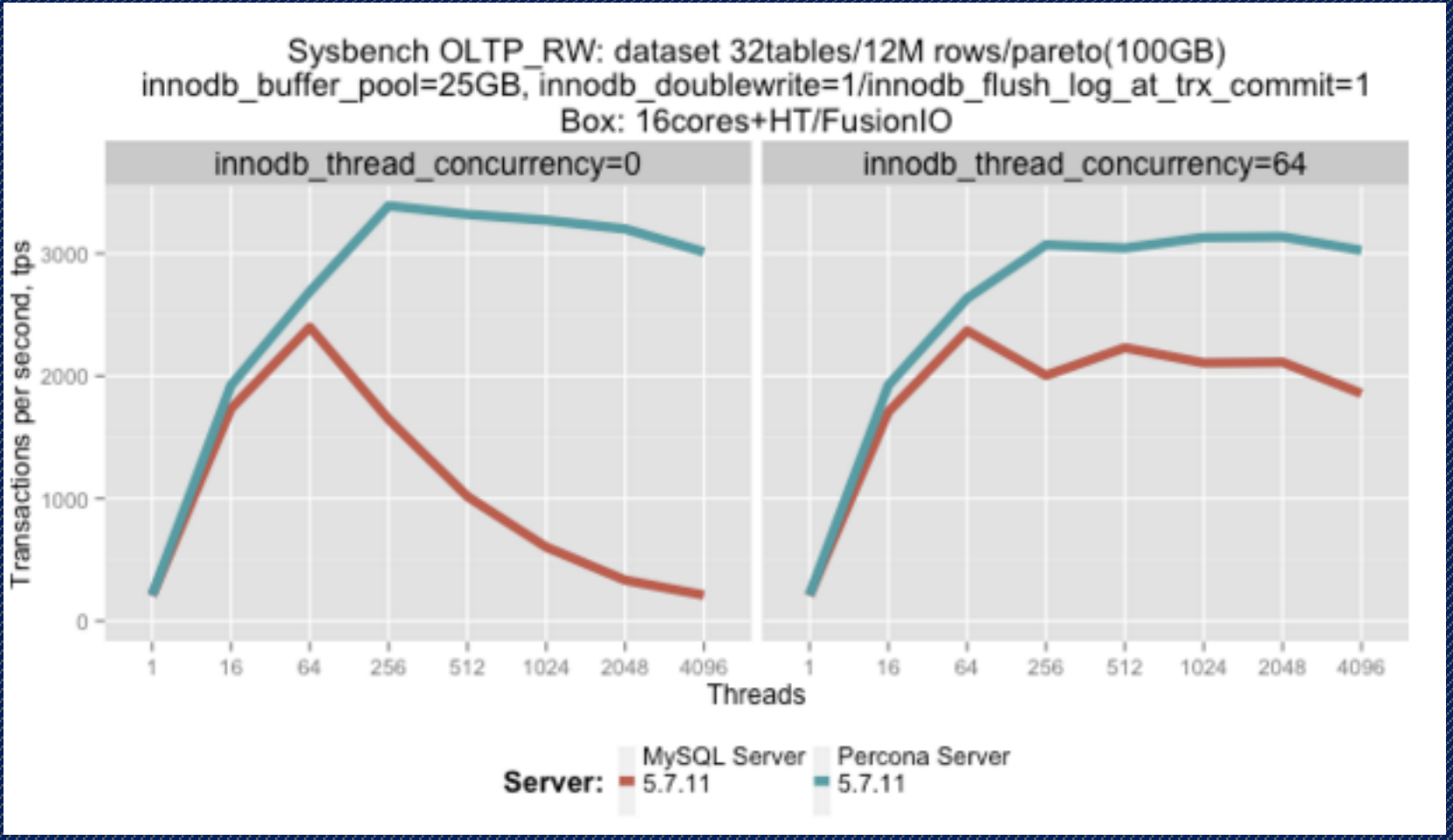
DoubleWrite in Percona Server 5.7

Multiple Doublewrite buffers (one per Buffer Pool)

Allows Parallel Contention Free Flushing

More Details http://bit.ly/1Xo6btL

Parallel Doublewrite Performance



33

Undo Space

Used to Implement MVCC and Rollbacks

Stored in System Tablespace by Default

Can be stored in separate Tablespace MySQL 5.6+

Undo Space Structure

Two Types of Records

- Inserts - no previous row version to store
- Update/Delete – store previous row version

Multiple Versions

Undo Row Record points to previous row version, if such exists

Chain can get really long for some workloads

Performance Considerations

Very long version history chains	• Avoid long running transactions if possible
Undo Space spilling out of cache	• http://bit.ly/1sI2IX9
Exploding System Tablespace due to runaway transaction	• MySQL 5.7 allows undo space purging

Enabling Undo Tablespace Purging

Requires Undo Tablespace to be In Separate Files

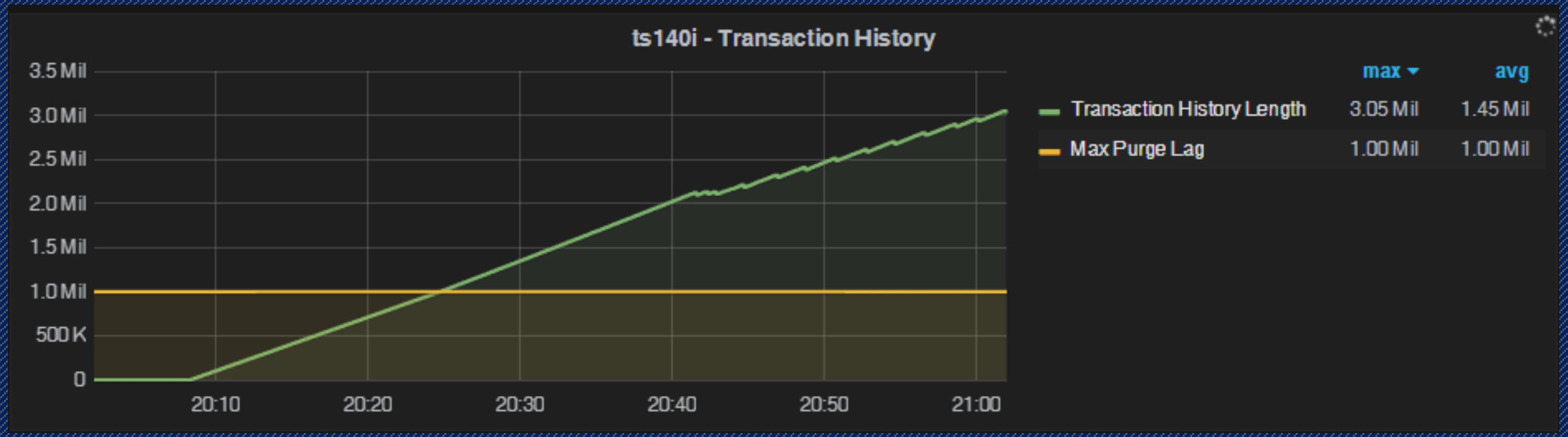
- innodb_undo_tablespaces=2
- innodb_undo_log_truncate=1

Purge Lag Prevention

Can help to prevent run away undo space

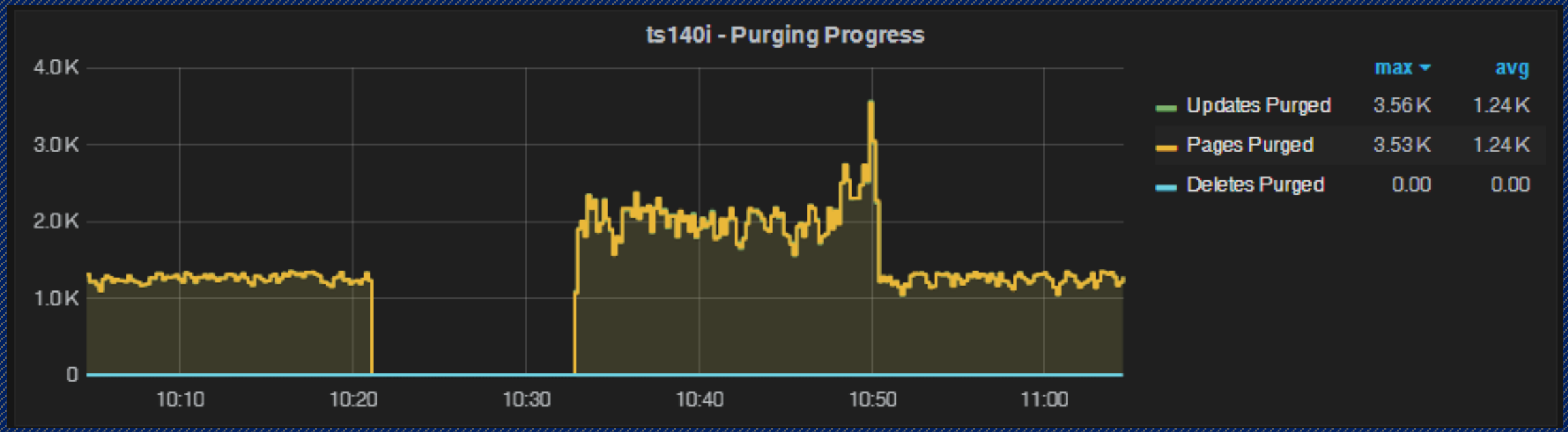
- **innodb_max_purge_lag=10000000**
- **innodb_max_purge_lag_delay=10000**

Transaction History



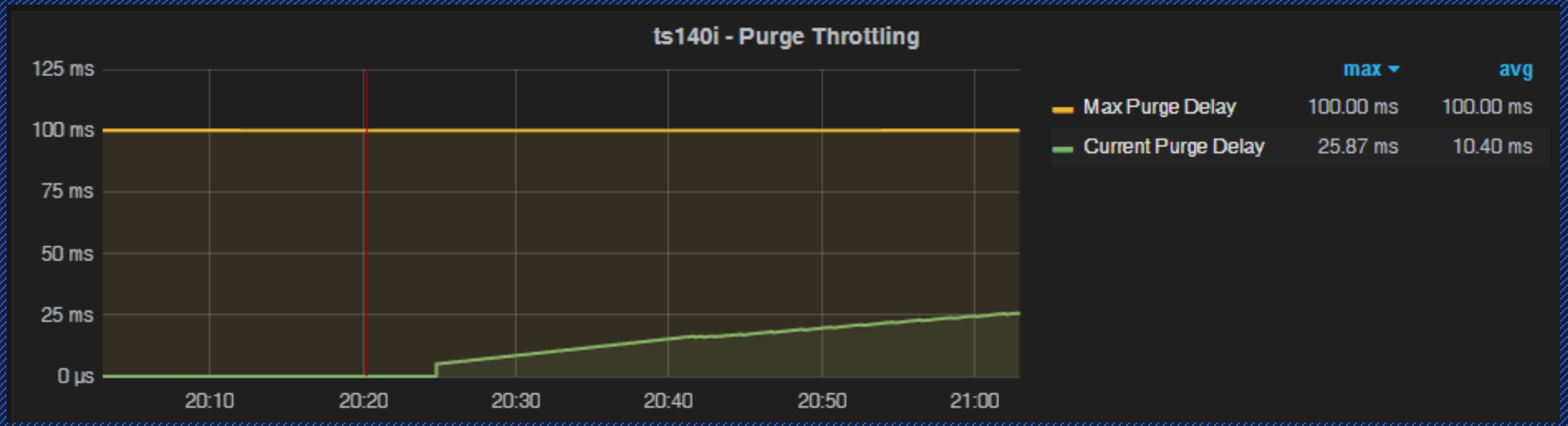
Purge Progress

Pause when long transaction is running



41

Purge Delay Management



Structures in Memory

Most Important Data Structures

Buffer Pool	Additional Memory Pool	Change Buffer	Adaptive Hash Index
Log Buffer	Double Write Buffer	Lock Structures	Dictionary Cache

Buffer Pool

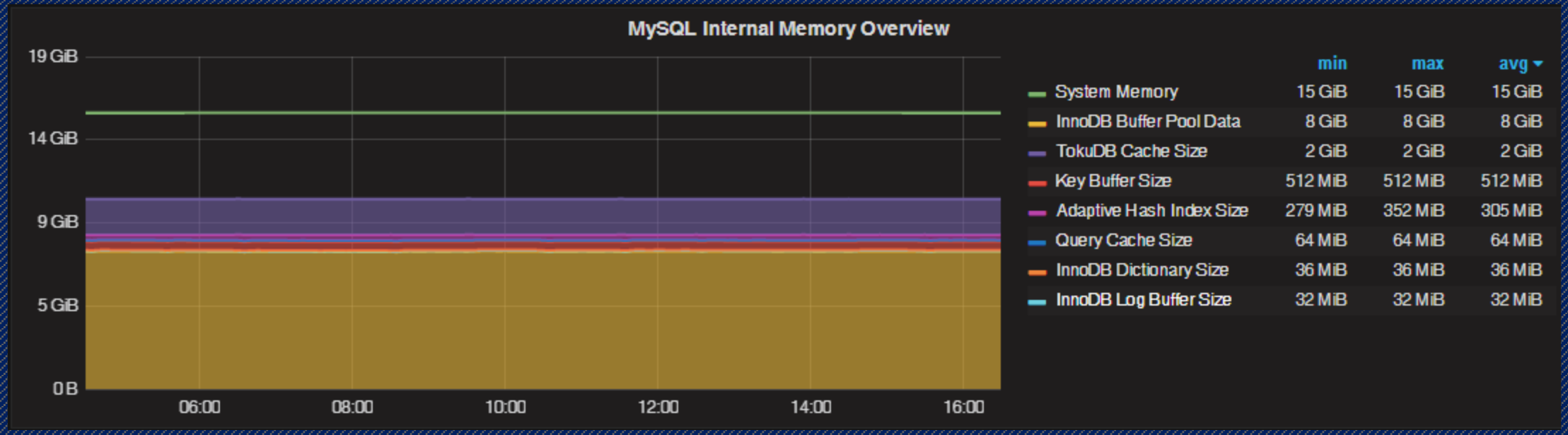
Cache (for IO to Tablespaces)

Storage for Lock Structures

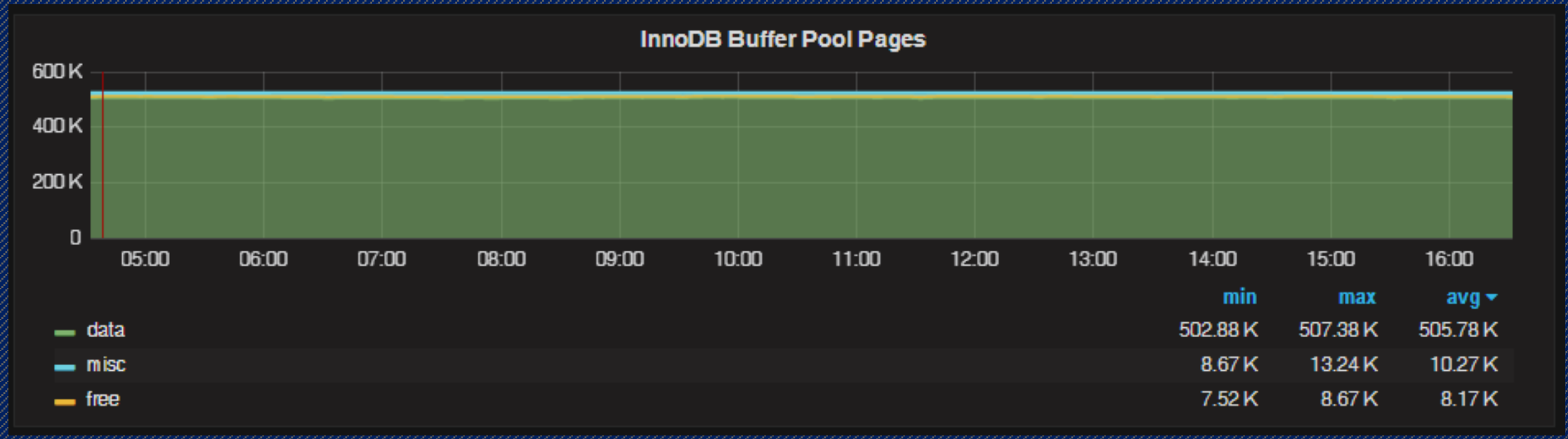
Storage for Adaptive hash index

Storage for Change Buffer

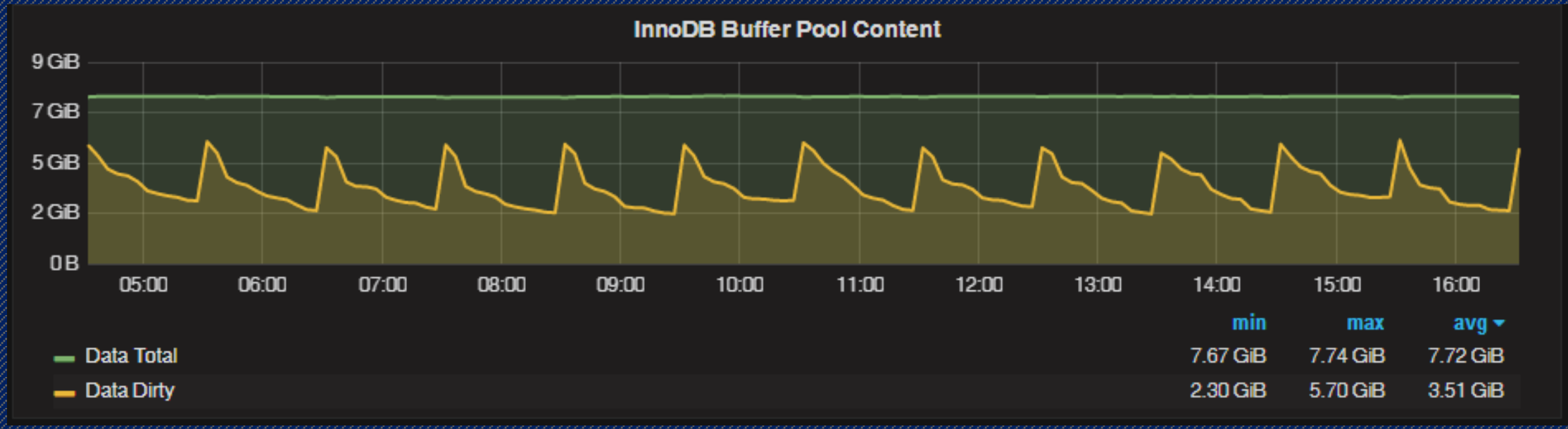
Typical Largest Memory Consumer



Will be mostly used to store data pages



Watch Data vs Dirty Data



Buffer Pool Configuration

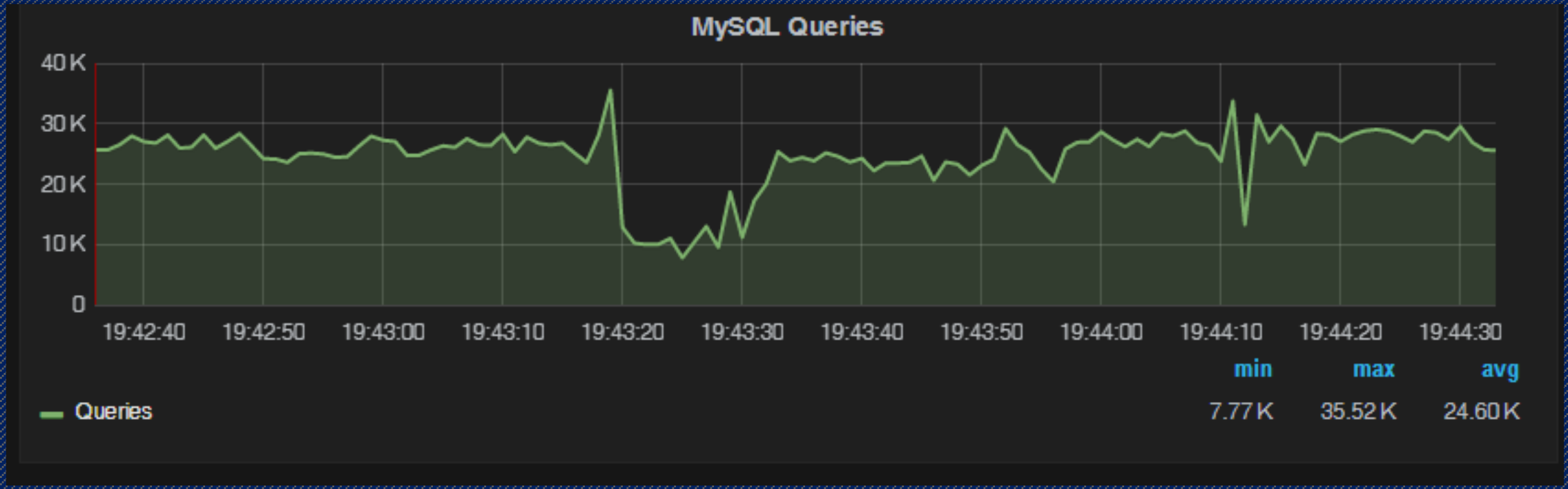
Can have Multiple Buffer pools

- Helps to reduce contention
- No mapping of tables to pools
- **Innodb_buffer_pool_instances**

Can Resize buffer pool online (MySQL 5.7)

- In case you picked the wrong size

Online Resizing has Performance Impact



50

Buffer Pool For Caching

Best Way to Cache Data

- **Innodb_flush_method=O_DIRECT** is recommended

Caching and Compression

- Both compressed and uncompressed copies
- May evict just uncompressed or both in case of cache pressure

Young and Old

- LRU based cache replacement policy
- Two main LRU list (Young and Old)
- Helps to make caching Full Table Scan Tolerant
- Page is placed to Young list first
- Moved to Old if accessed again after period of time

More on Caching

Background Flushing and “Page Cleaning”

Page Checksum validation on read

Integration with Change Buffer to merge outstanding changes on page read

Performance Consideration

Main use of Memory on Innodb system

- May allocate 80%+ memory for it

Beware of Swapping

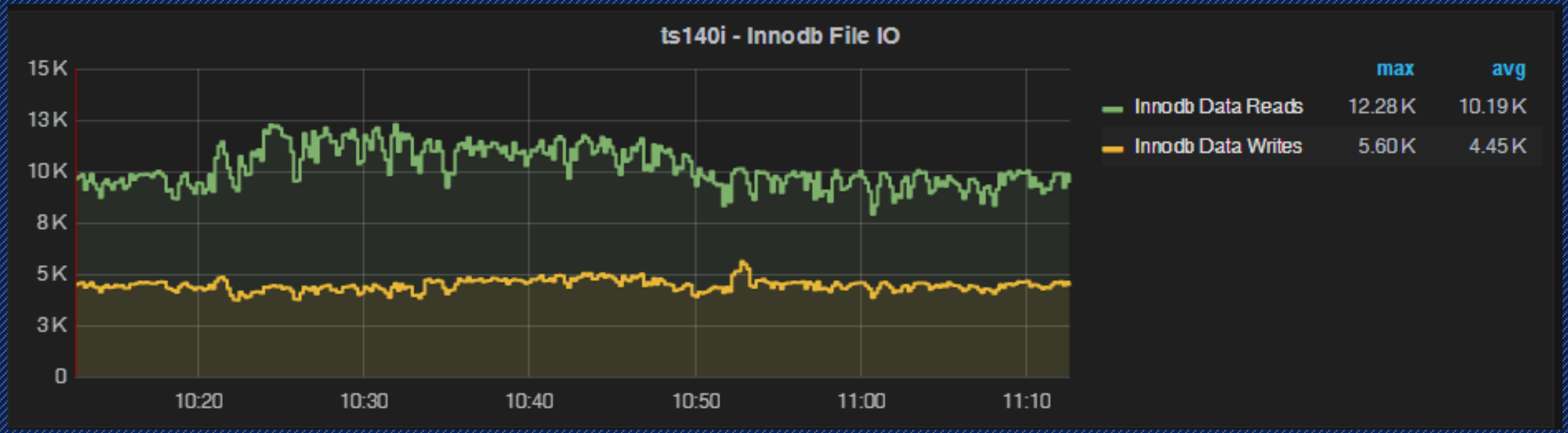
- Better few percent too little than few percent too much

More CPU Cores – more buffer pool instances might help

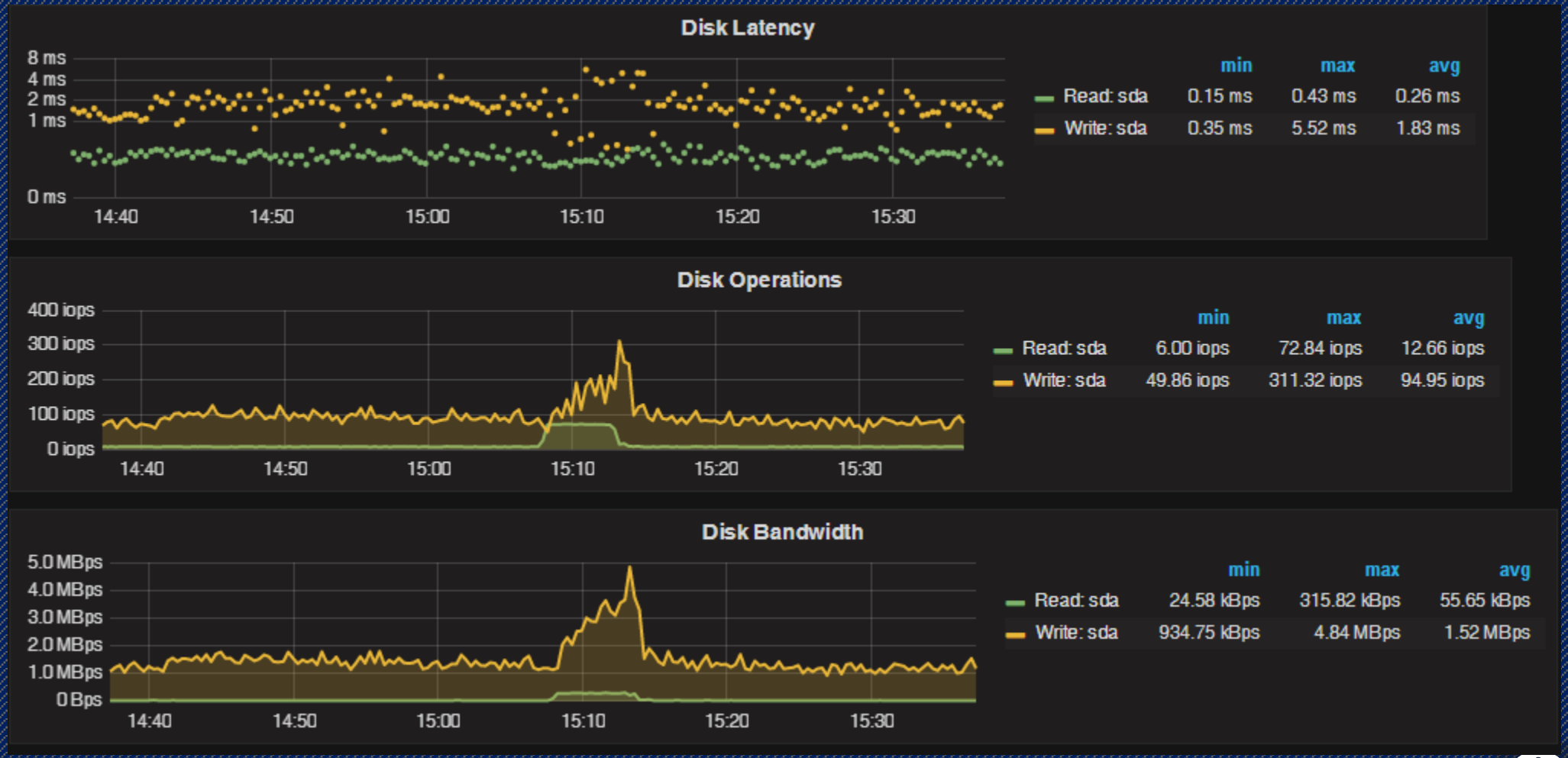
Innodb_flush_method=O_DIRECT best In most cases

Innodb IO

Watch number of IOs and Latency



IO Analyses Use Case



Good Idea to look at IO Usage Per Query

UPDATE sbtest D30AD7E3079ABCE7

Selected query class: 930.90 k Queries (258.58 QPS, 67.30%, 5.31 Load) | Total: 2.81 m Queries (780.82 QPS, 100.00%, 7.88 Load)

Metrics	Rate/Sec	Sum	Per Query Stats
Query Count	259.12 (per sec)	932.85 k 32.40% of total	
Query Time	5.31 load (67.16%)	19101.45 sec 67.16% of total	5.25 ms avg
Lock Time	1.38 (avg load)	4965.60 sec 51.50% of total 15.13% of query time	793.79 µs avg
Innodb Row Lock Wait	<0.01 (avg load)	28.14 sec 42.03% of total 1.44% of query time	75.79 µs avg
Innodb IO Read Wait	<0.01 (avg load)	35.85 sec 2.47% of total 10.93% of query time	573.60 µs avg
Innodb Read Ops	2.48 (per sec)	8.91 k 1.86% of total	0.00 avg
Innodb Read Bytes	39.61 KB (per sec)	139.25 MB 1.86% of total 16.00 KB avg io size	3.50 KB avg
Innodb Distinct Pages	-	-	6.03 avg
Bytes Sent	13.18 KB (per sec)	46.35 MB 1.72% of total	52.00 Bytes avg
Rows Examined	258.17 (per sec)	929.43 k 0.64% of total 0.00 per row sent	0.88 avg

Additional Memory Pool

Used to be used to speed up allocation

Now Innodb allocates directly from OS

Change Buffer

- Exists both in memory and on disk
- Stored Inside Buffer Pool
- Matters when data Is much larger than memory
- Buffers changes to be reflected in B-Tree later
- Transparent for the application

Things to Consider

Worse Than useless if full	• Takes away memory from cache
Can slow down reads to unmerged pages	• Merge needs to be performed before read
Can cause performance problems on restart	• As it will not be in memory any more

Performance Considerations

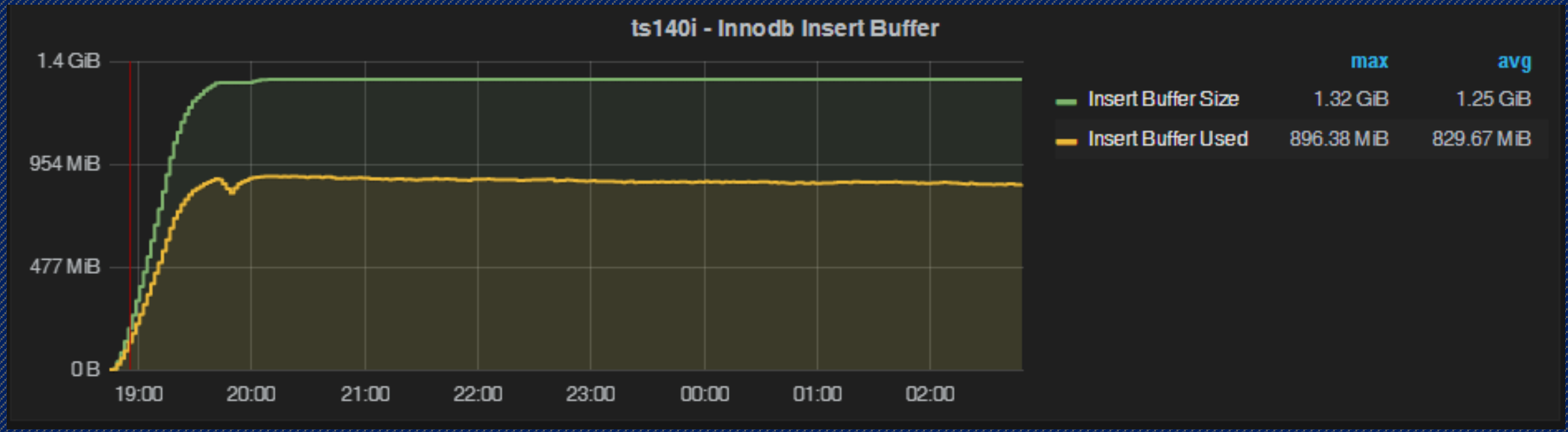
Can be disabled

Good to restrict size, especially on SSDs

innodb_change_buffer_max_size

Insert/Change Buffer Size

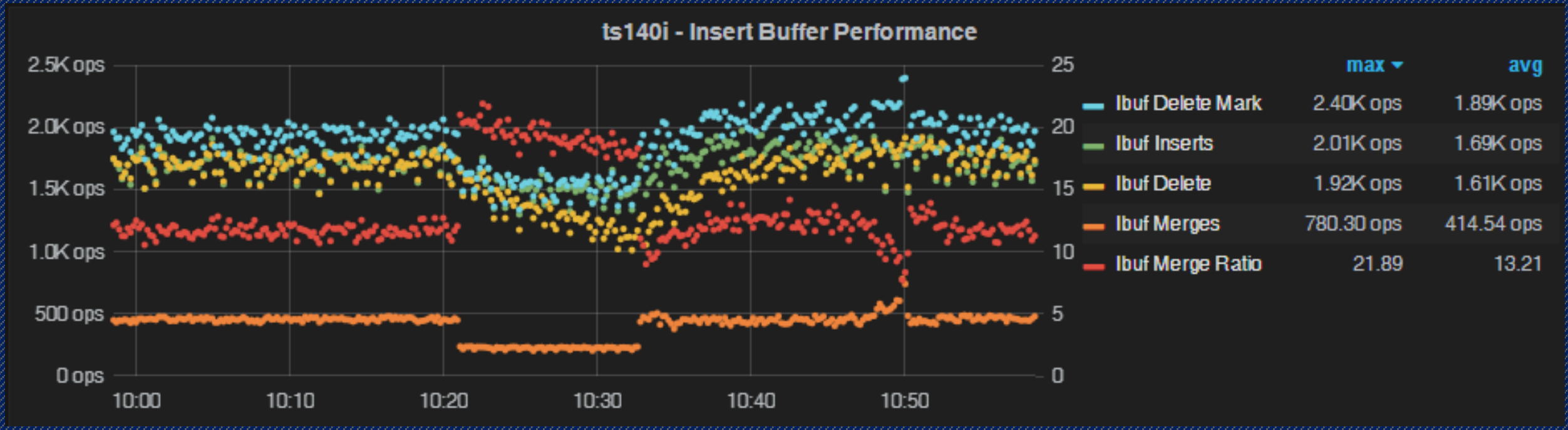
Takes a while to reach steady state



62

Insert Buffer Performance

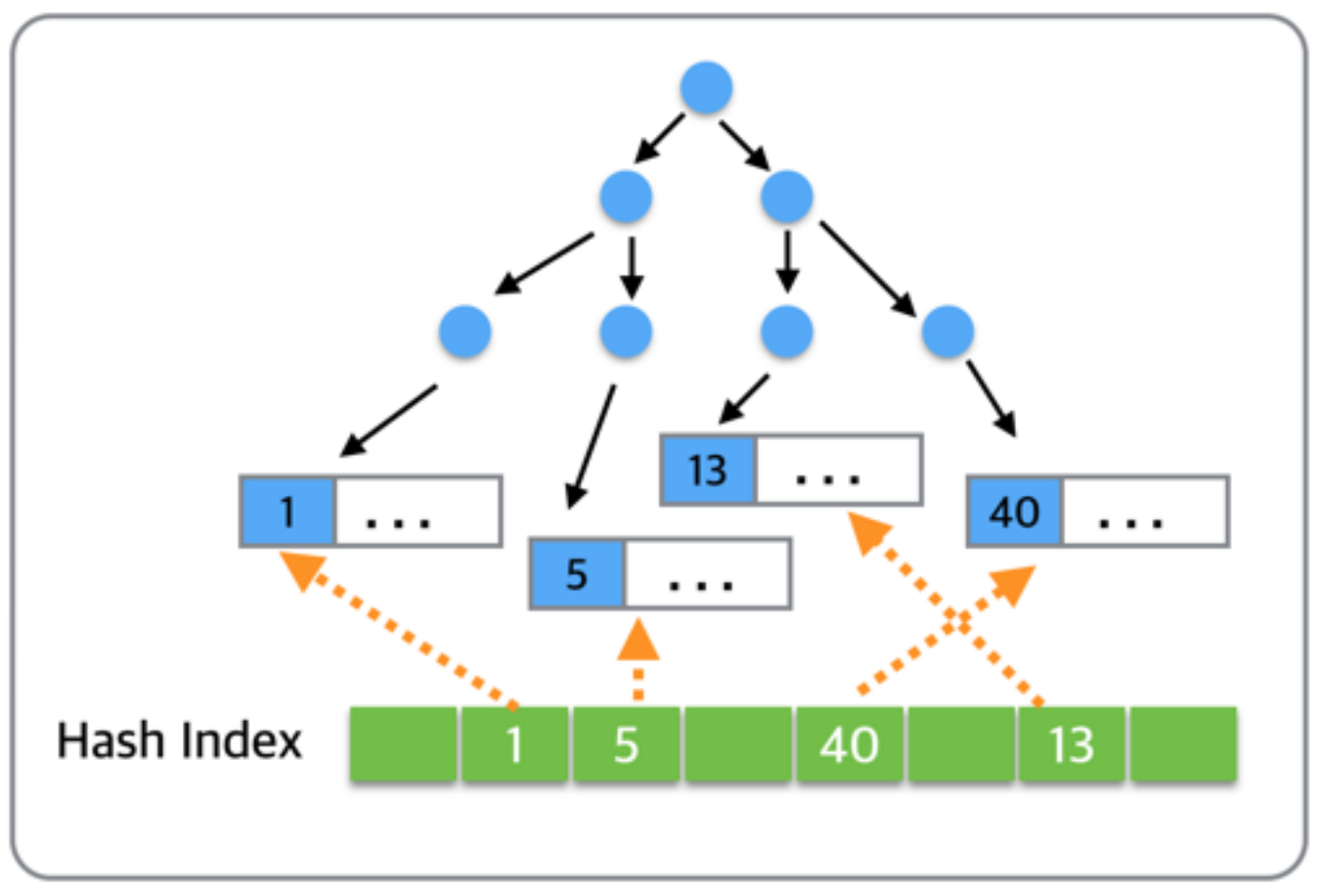
Watch Merge Ratio



Adaptive Hash Index

- Speed up access inside buffer pool
- B-Tree Descent to Hash Table Lookup
- Partial Index (most accessed pages)
- Built automatically by Innodb
- Works for PRIMARY and SECONDARY Keys
- For Key Prefixes and Full Keys

Adaptive Hash Index Illustration



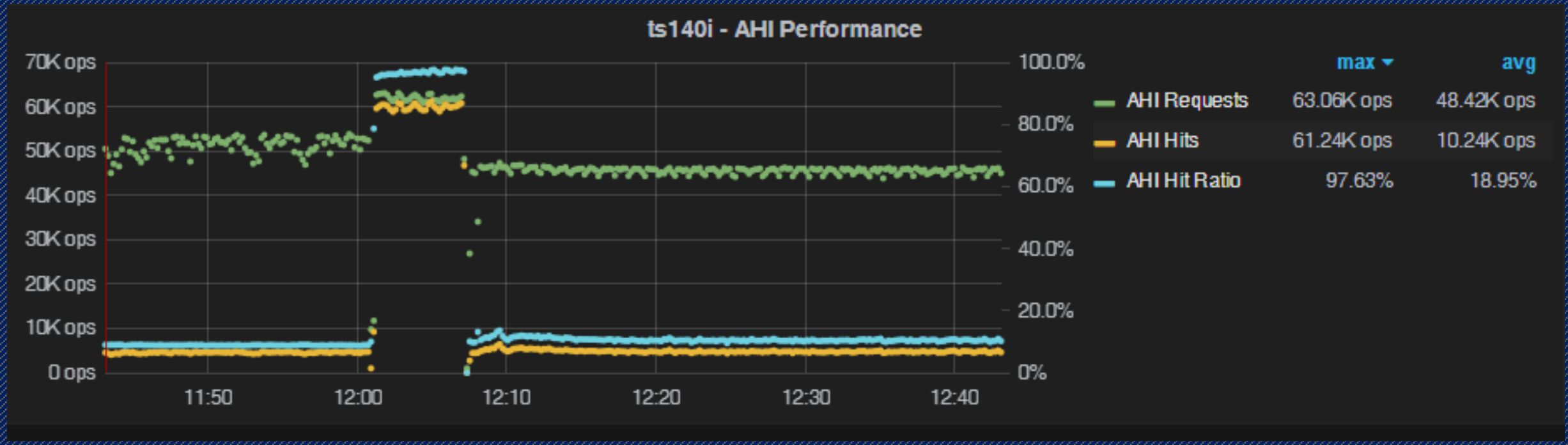
Performance Considerations

Can become contention hotspot	• Better Performance; Lower Concurrency
Disable	• **Innodb_adaptive_hash_index=0**
Partition (PS 5.6, MySQL 5.7)	• **innodb_adaptive_hash_index_parts=8**

66

AHI Performance

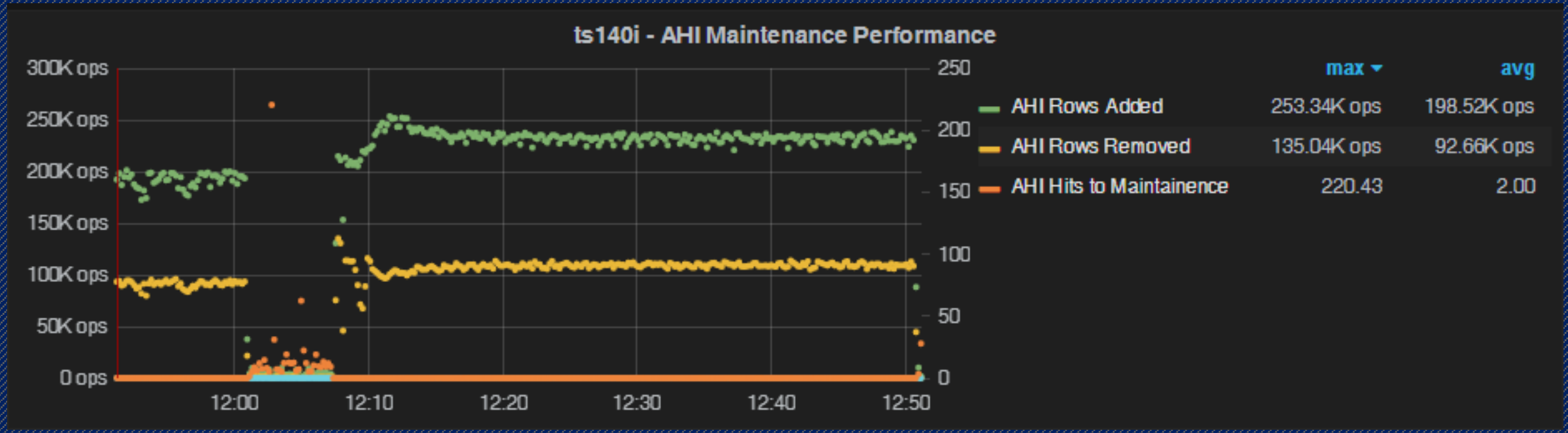
Consider AHI Hit Ratio



67

AHI Maintenance

And Maintenance Overhead vs Value



68

Log Buffer

Store Log Records before they are flushed to Log

Innodb_log_buffer_size

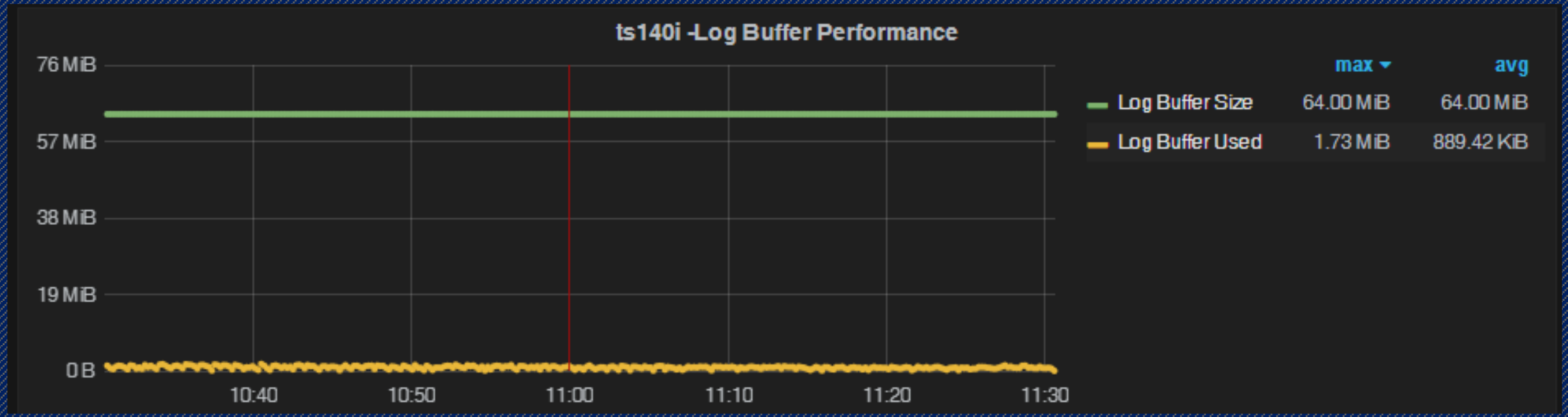
innodb_flush_log_at_timeout

Watch how much data is written to the logs

Higher Log Buffer Sizes reduce contention (up to 256Mb)

Innodb Log Buffer

More efficient than you would think



70

Double Write Buffer

Is it on disk or in memory ?

Both!

Size Can’t be tuned

PS 5.7 - Improved

Lock Structures

Allocated in the Buffer Pool

Very Efficient - few *bits* per lock

No Lock Escalation

For each page having lock small bitmap allocated indicating locked rows

Data Dictionary Cache

Information about open tables

- Structure; Index Statistics etc.
- Tied to **table_definition_cache_size**

Was not cache before MySQL 5.6

- All accessed tables were always kept in memory

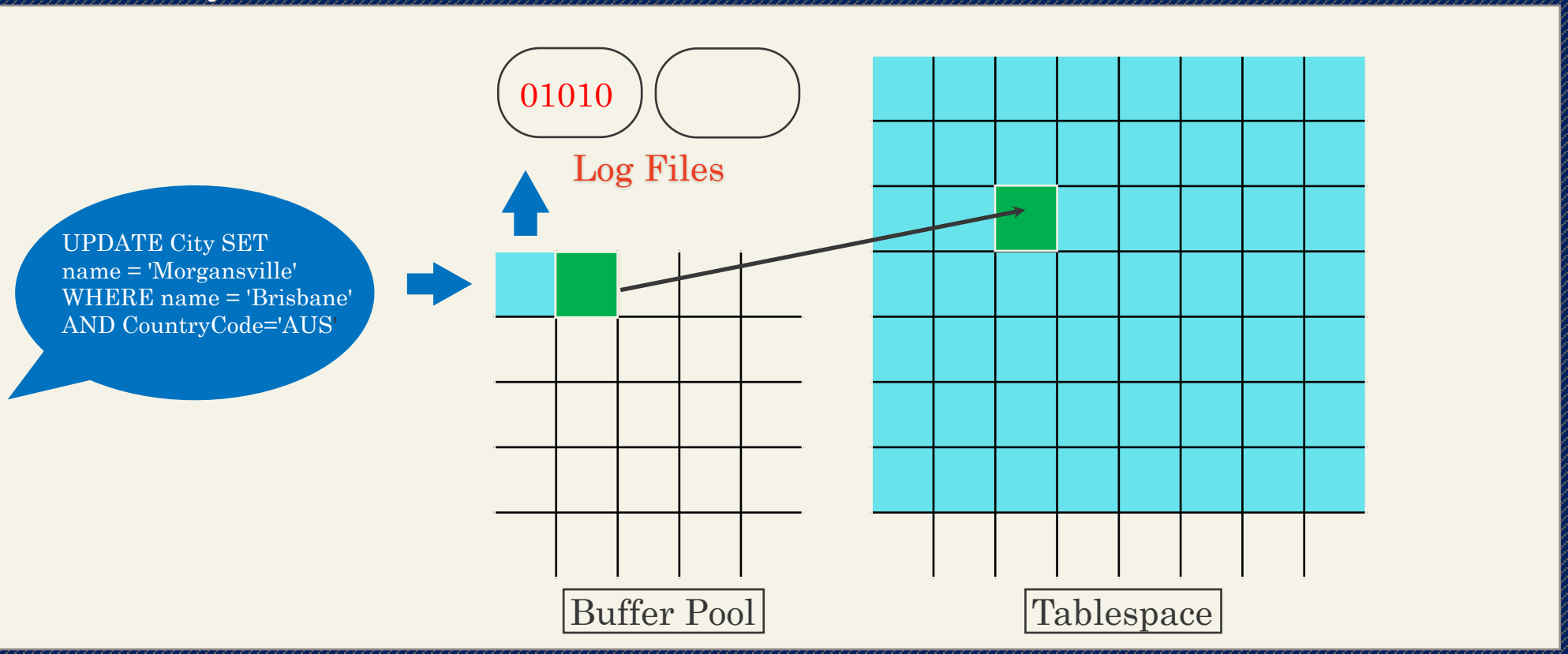
Operations Details

Transaction Control

How do Transactions “Work” In Innodb

Basic Operation

How Write Query is Handled



76

Database Operations

Reads

Writes

DDL (CREATE; Alter etc.)

Transaction Mode

No Transactions for DDLs

AUTOCOMMIT=1 by default

Every Statement is its own transaction

BEGIN/COMMIT typically used

AUTOCOMMIT=0 can also be used

Isolation Mode

How Transactions are Isolated from Each other

- **READ UNCOMMITTED**
- **READ COMMITTED**
- **REPEATABLE READS** (default)
- **SERIALIZABLE**

Performance Considerations

READ COMMITTED or **REPEATABLE READS** may be giving best performance

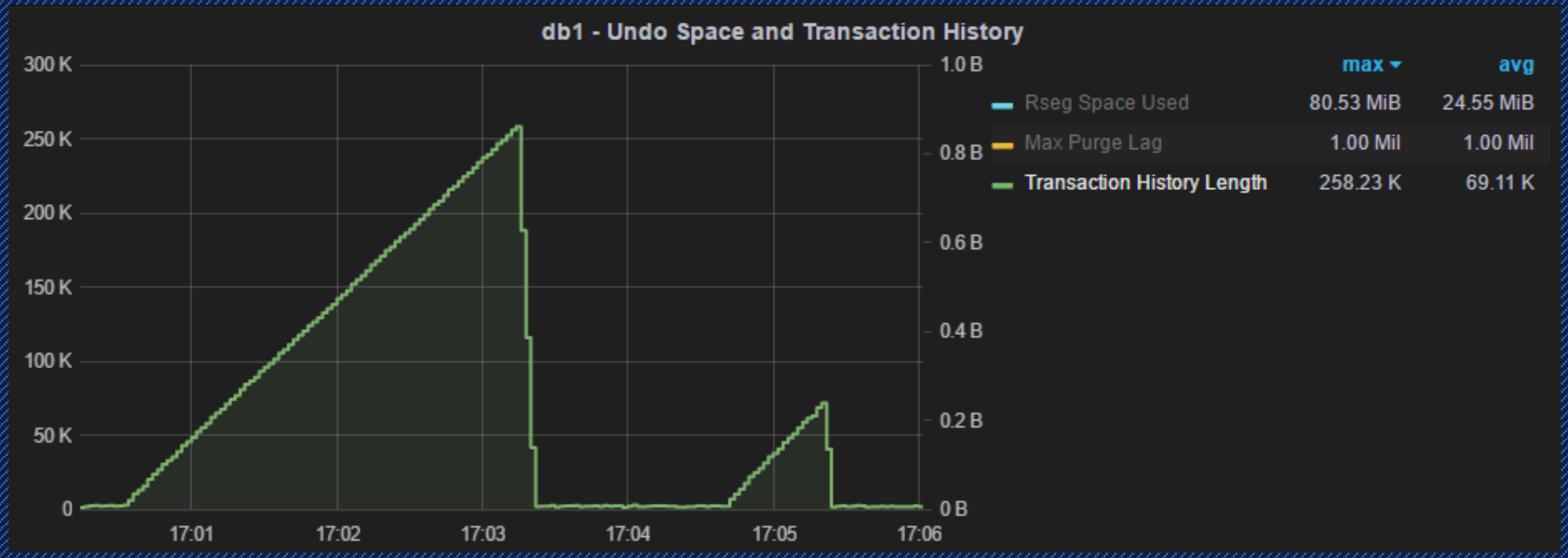
REPEATABLE READS reduces snapshot allocation overhead

READ COMMITTED reduces amount of history which need to be maintained

READ UNCOMMITTED for very long statistical queries

Better performance with READ-UNCOMMITTED

- Running Long Select while running SysBench updates



Pessimistic Locking

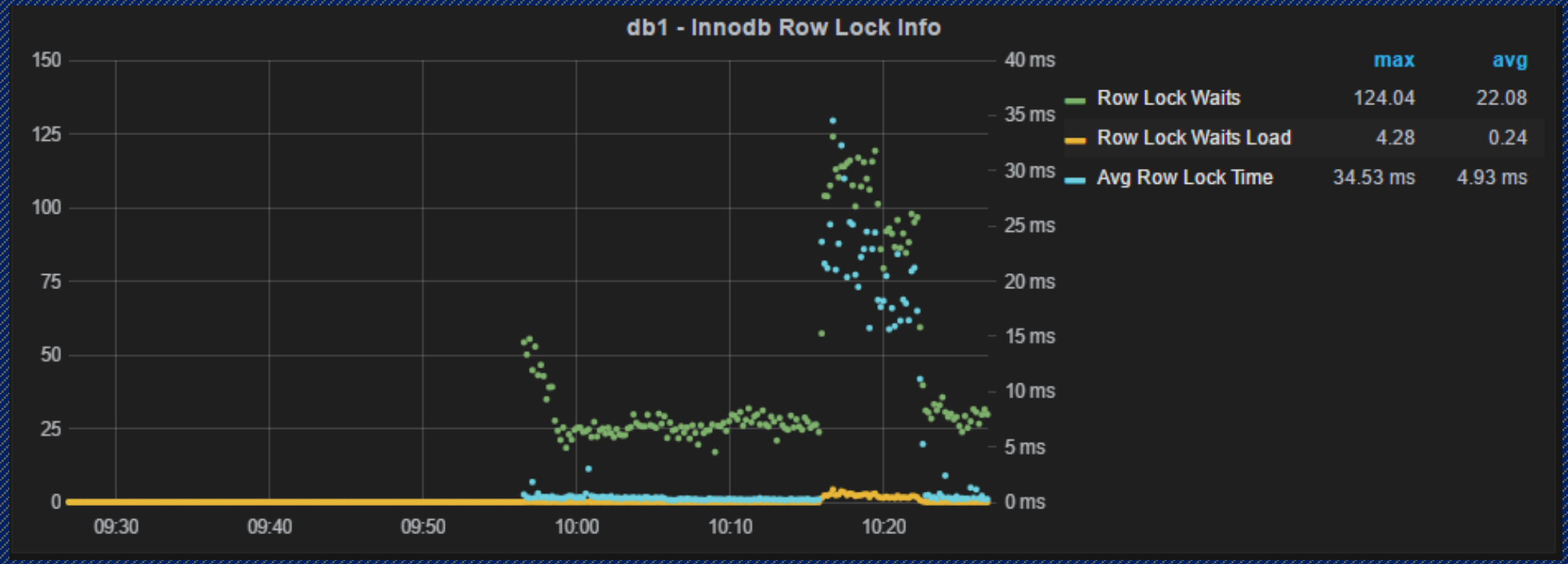
Take Locks as you go

Wait if row is currently locked

Detect Deadlocks almost instantly

How many Row Lock Waits are happening ?

- Wait more important than number



Innodb Reads

Non locking reads by default (MVCC)

Can be made locking with **FOR UPDATE** or **LOCK IN SHARE MODE** modifiers

SERIALIZABLE adds **LOCK IN SHARE MODE** to all reads

Long Read Transactions

Result in Concurrent writes creating a lot of undo space because Purging is blocked

May have to go far in undo chain to find data they are looking for

Performance Considerations

- Covering Indexes are great
- Will not read out-of page blobs if not needed by query
- Multiple columns faster than multiple rows
- Number of Columns impacts performance

86

Innodb Writes

Set Locks as they go

Bypass MVCC (you can't modify non current data)

Modify Actual Data as they go (COMMIT is trivial)

Innodb DDL

- Not Transactional but Atomic
- Commit Transaction when Executed
- Many DDL are allowed Online
- Metadata Locks used for Coordination with other operations

Long Write Transactions

- Create a lot of records in Undo Space
- Can produce waste in Indexes
- Reduce Concurrency due to Locks
- Have increased chance of Deadlocks
- Can Cause stalls in Replication

Performance Considerations

- Writes are more expensive than reads
- Behind every write there is the read
- Large amount of versions for the single row can be the problem
- Hot columns might be better in separate table

MVCC Details

How Does Multi Version Concurrency Control Works

MVCC Basics

Every transaction has a number

Innodb maintains list of active transactions

Each row stores transaction which has last touched it

Previous row versions are stored as chain in undo space

Delete is really “Delete Mark” and purge later

MVCC Basics

Transaction accessing a row can quickly check if it should "see" it or go look at old version

Updates done in place with previous copy migrated to undo space

Large BLOBs (out of page) are not updated in place

MVCC and Indexes

Indexes contain all current version

Key Value “5” will point to all rows which have key value 5 now or had it in the past

Index is marked with last transaction which modified it

Visibility can often be checked without reading row

MVCC Garbage Collection

- Transaction Commits
- Read-View Advances (READ COMMITTED)
- Some old version are no more needed
- This cleanup happens in background
- Purge Threads

Performance Considerations

Watch for MVCC Garbage grow unchecked

Due to Long Running Transactions

Due to Purge being unable to keep up

Monitor **Innodb_history_list_length**

Locking Basics

How Locking Works in Innodb

Types of Locks

- Row Locks
- Index Locks
- Gap Locks
- Meta Data Locks

Locking Modes

"S" – Shared Lock (Reads)

"X" – exclusive Lock (Writes)

"I" - Intention Lock (on higher level object)

"SIX" – Set on the table which has some words being updated

Locking Performance

Locking Reads can be up to 2x slower

Read Lock is set on Index records for active index

Write Locks are set on row and all index entries

Foreign Keys

Create complicated locking dependencies

Can be lock troubleshooting nightmare

Deadlock Detection

There is always chance for deadlocks

Immediate

Non-Recursive (5.6+)

May report false positives

Whole Transaction rolls back on deadlock

Lock Timeouts

Avoid waiting for locks forever

innodb_lock_wait_timeout

Can Rollback Transaction or Statement

Applies to Row Locks only (Meta Data Locks have their own Timeout)

Performance Considerations

Lock Less

Lock for Less time

Acquire Locks in the same order

Latching

Latches are Internal Locks – Mutexes, Read-Write-Locks

Latches vs Locks

Locks are driven by workload and transaction isolation semantics

Latches are based on internal implementations

Latches change a lot between versions

Why Important ?

Hot Latches frequent cause of Performance Problems

Innodb Latches

Does not use OS Primitives Directly

Implements its own Wrappers for Performance and Transparency

Understanding Latches

Performance Schema

- Need To additionally enable "sync" instrumentation

SHOW ENGINE INNODB STATUS

- Limited but always available

Where is my contention ?

Performance Schema is best way to look

- Check out **sys_schema** (Included with MySQL 5.7)

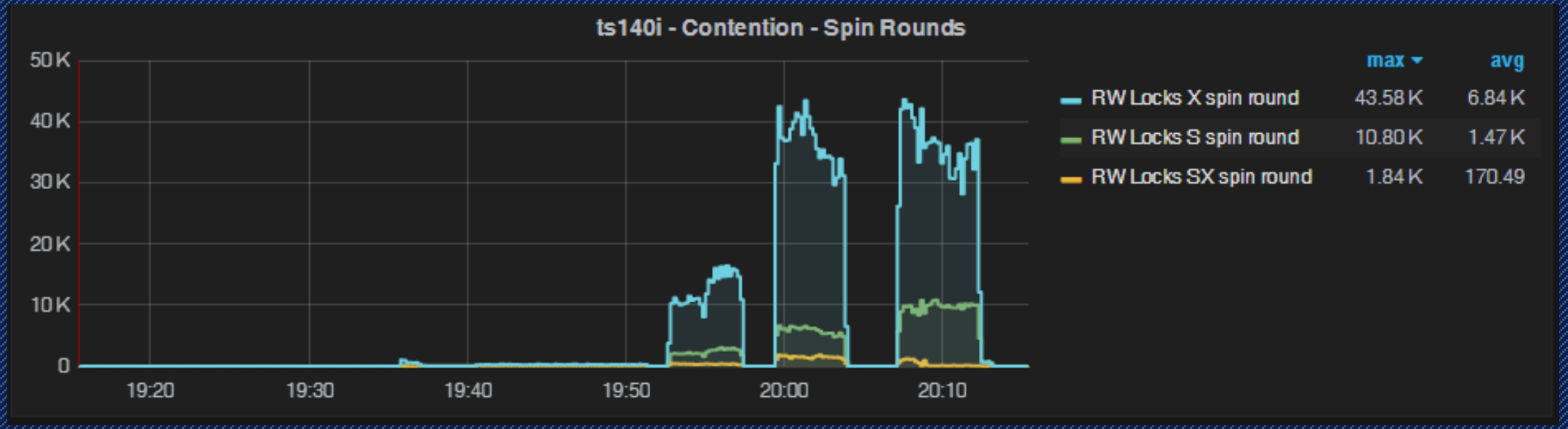
```
+-------------------------------------------+---------+---------+
| event_name                                |nsecs_per| seconds|
+-------------------------------------------+---------+---------+
| wait/synch/rwlock/innodb/index_tree_rw_lock | 19543.3 | 3456.1 |
| wait/synch/mutex/innodb/log_sys_mutex       |  2071.3 |  385.8 |
| wait/synch/rwlock/innodb/hash_table_locks   |   165.7 |  184.5 |
| wait/synch/mutex/innodb/fil_system_mutex    |   328.3 |  113.6 |
| wait/synch/mutex/innodb/redo_rseg_mutex     |  1766.4 |   84.9 |
| wait/synch/rwlock/sql/MDL_lock::rwlock      |   430.2 |   73.9 |
| wait/synch/mutex/innodb/buf_pool_mutex      |   264.5 |   72.7 |
| wait/synch/rwlock/innodb/fil_space_latch    | 27216.1 |   53.8 |
| wait/synch/mutex/sql/THD::LOCK_query_plan   |   167.0 |   50.7 |
| wait/synch/mutex/innodb/trx_sys_mutex       |   394.9 |   41.5 |
+-------------------------------------------+---------+---------+
```

InnoDB Latching in SHOW INNODB STATUS

```
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 13569
 OS WAIT ARRAY INFO: signal count 11421
--Thread 1152170336 has waited at ./../include/buf0buf.ic line 630 for 0.00 seconds the semaphore:
Mutex at 0x2a957858b8 created file buf0buf.c line 517, lock var 0
--Thread 1147709792 has waited at ./../include/buf0buf.ic line 630 for 0.00 seconds the semaphore:
Mutex at 0x2a957858b8 created file buf0buf.c line 517, lock var 0
 Mutex spin waits 5018, rounds 70800, OS waits 2033
 RW-shared spins 2736, rounds 84289, OS waits 2791
 RW-excl spins 904, rounds 111941, OS waits 3607
 Spin rounds per wait: 14.11 mutex, 30.81 RW-shared, 123.83 RW-excl
```

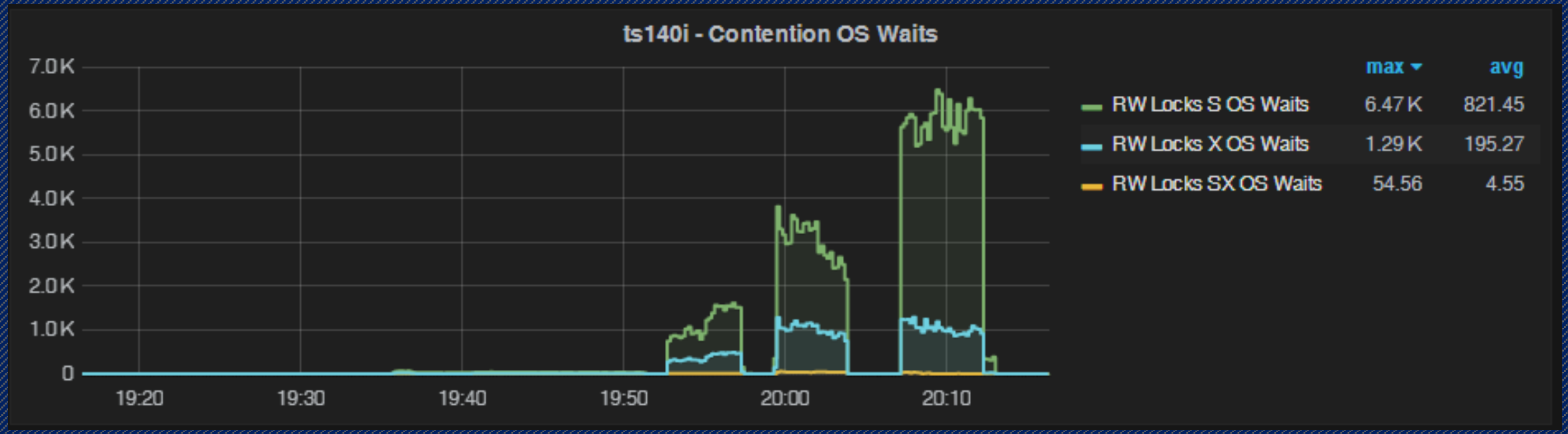
Some Lock Statistics in INNODB METRICS

- RW Locks Only; Not Mutexes
- Sysbench 1,8,64 and 512 connections



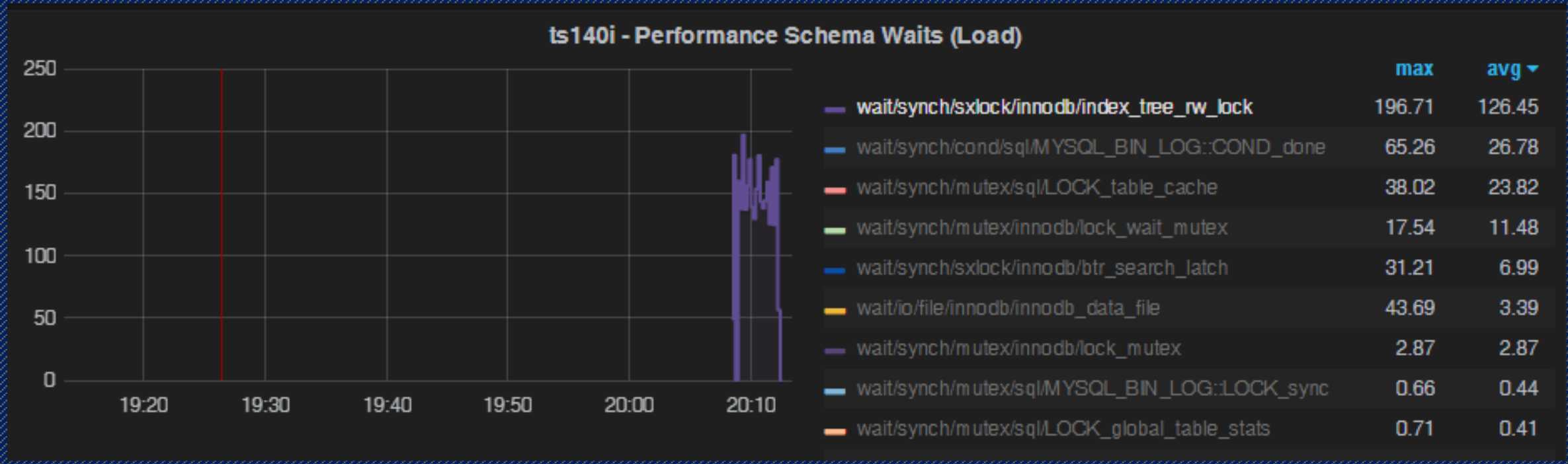
OS Waits

- When Spin Wait Failed



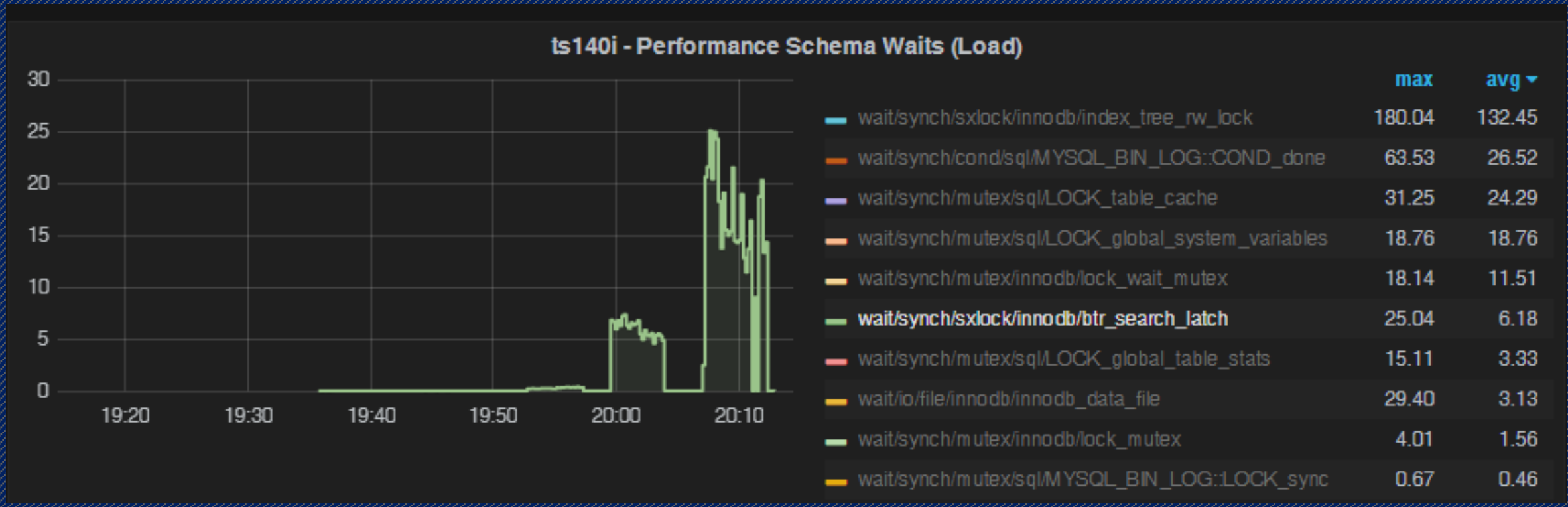
Index Tree rw_lock contention

- Only shows at high thread number



Adaptive Hash Index

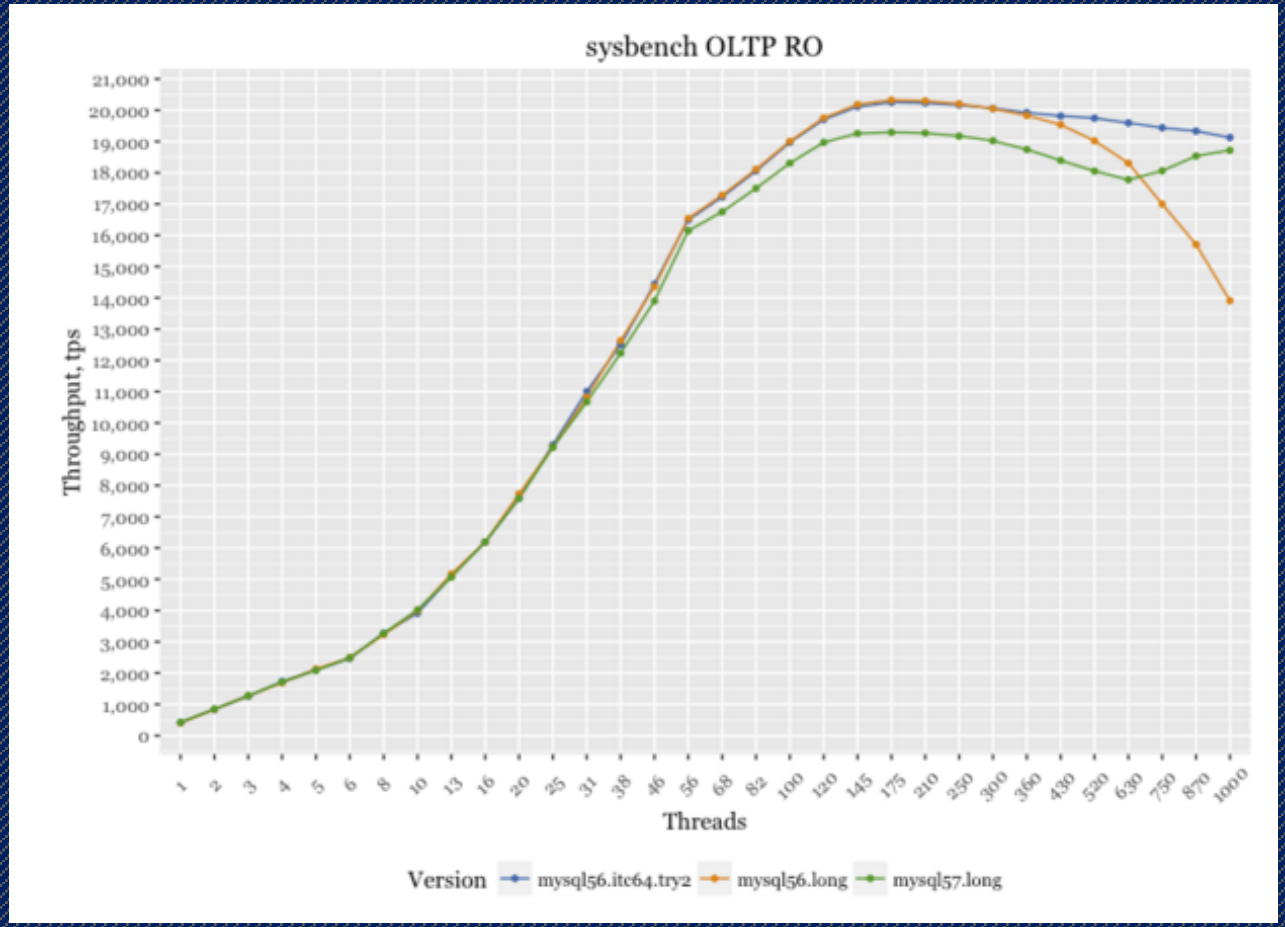
- Starts to become problem earlier



Performance Considerations

innodb_spin_wait_delay	• Balance wasting CPU vs. cost of context switch
Innodb_thread_concurrency	• Limit amount of threads in Innodb Kernel
Thread Pool	• MySQL Enterprise, Percona Server, MariaDB

Innodb_thread_concurrency still has its place



Bonus Material

(If we have time to get to it)

Background Operations

Why Important

If it does not have to happen synchronously it should not

A lot of is happening in Background!

Background Operations

Checkpointing/Flushing

Page Cleaning

Purging

Change Buffer Merging

Read-Ahead

Checkpointing Basics

Before record can be overwritten in the Log file corresponding page must be flushed from buffer pool.

Flush List

For each page in Buffer pool the LSN corresponding to last flush is kept

Flush List - list of all dirty buffer pool pages sorted by last flushed LSN

Checkpoint Age

The difference between current LSN and earliest LSN on the Flush List

Checkpoint Age must not reach combined Redo Log file size for Recovery to be successful

Flushing Challenge

Flush enough not to run out of log space

Do not flush too aggressively to impact performance

How much can we delay ? We do not know future workload

Users like uniform performance

Main Question

Flush Pages from the tail of the Flush List

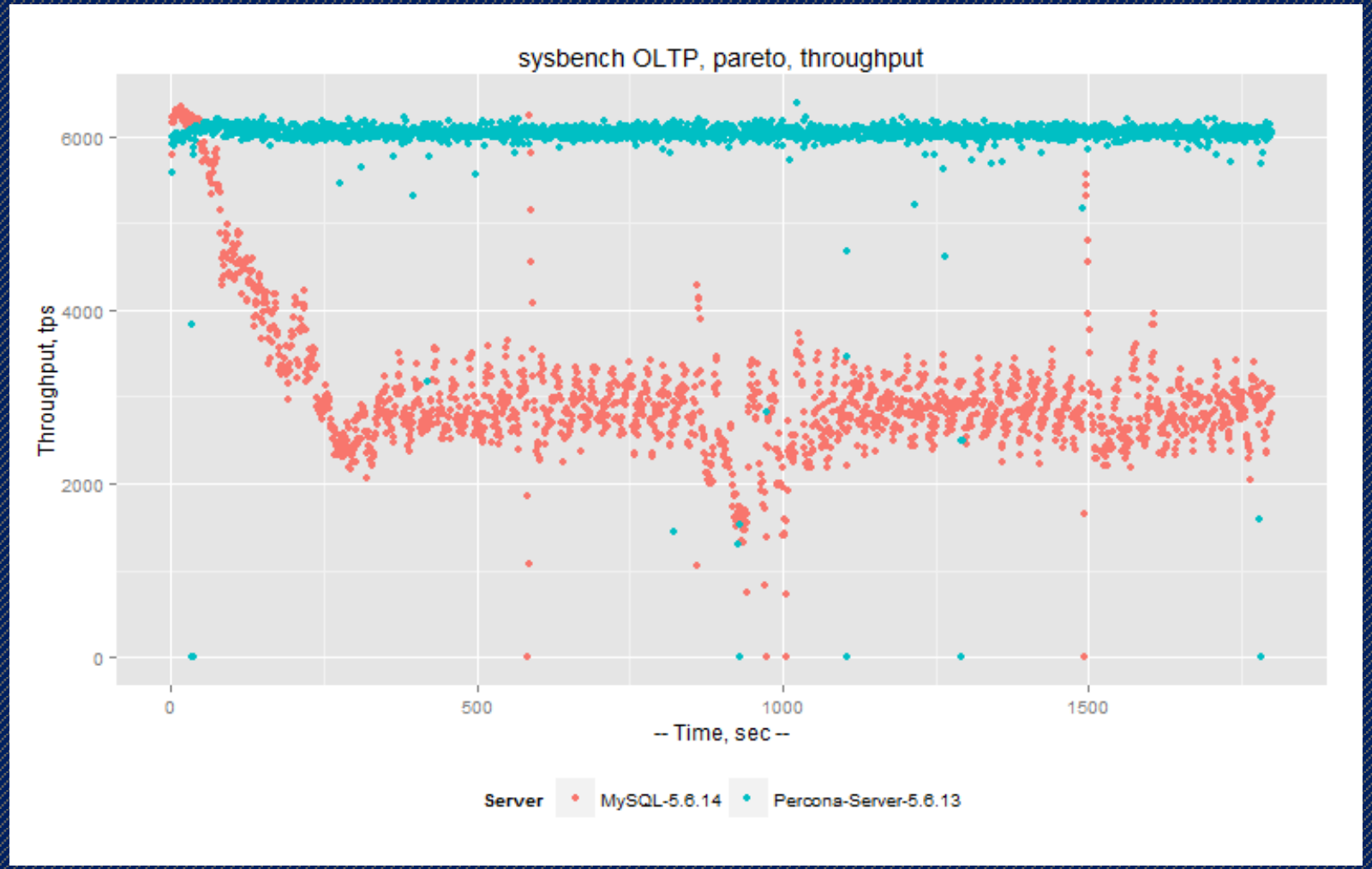
But how many ?

Not Completely Solved Problem

If pages can be made dirty much faster than the can be flushed, system will have huge stalls.

Gets better in every Major MySQL Release

Improvements in PS 5.6 vs MySQL 5.6



LRU Flushing Basics

When Read Happens need clean or free page to replace

Pages in the tail of LRU might not be clean

Excessive LRU scans can happen

Much more important page might be replaced

LRU Solution

Ensure pages in the LRU tail are always clean

So LRU Scans can be kept short

LRU Flushing by Page Cleaner does this

Page Cleaner(s)

Does Both Kinds of Flushing

Multiple Threads available in MySQL 5.7

Significant Optimizations in MySQL 5.7

Performance Considerations

innodb_max_dirty_pages_pct

innodb_flush_neighbors

innodb_lru_scan_depth

innodb_io_capacity

innodb_io_capacity_max

Purging

Keeping up with Garbage coming from MVCC

Purging Does

Remove old Row Versions in Undo Space

Cleanup unused Large Blobs

Remove Deleted rows from Tables

Remove old Index records from Indexes

Purge Thread(s)

One or More purge threads

- Innodb_purge_threads

May not be able to keep up

You can often insert/update rows faster then they can be purged

- Especially at high concurrency

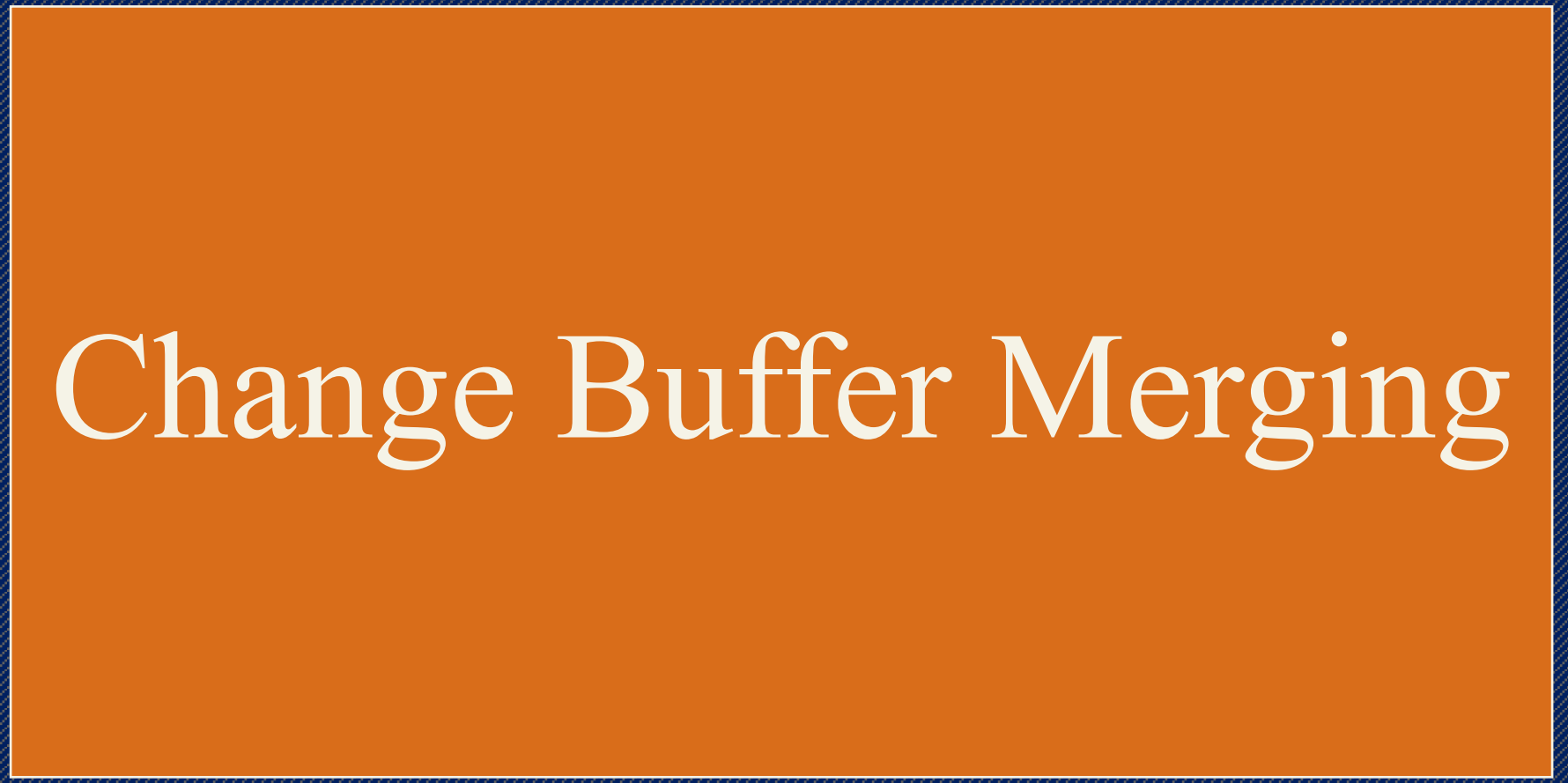
Make sure you watch **innodb_history_length**

- Units are Transactions

Consider limiting history length

- **innodb_max_purge_lag=1000000**
- **innodb_max_purge_lag_delay=10000**

Change Buffer Merging



Change Buffer

Change Buffer is delayed work which needs to be done at some point

The more records merged with single merge the better it is

Performance Considerations

Is background merge speed enough ?

Full change buffer is just overhead

Innodb_io_capacity

innodb_change_buffer_max_size

Read Ahead

How Innodb Does Read-Ahead

Types of Read-Ahead

Sequential Read-Ahead

Random Read-Ahead

Logical Read-Ahead

Sequential Read Ahead

Fetch the next Segment if a lot of pages are accessed from current one

Default for **innodb_read_ahead_threshold** is 56 making it not very aggressive

Does not help Fragmented tables

Can watch pages removed without access to see how effective it is

Random Read Ahead

Disabled by Default

Will fetch full extent if 13 or more pages from given extent are in buffer pool already

innodb_random_read_ahead

Logical Read Ahead

Available in WebScaleSQL

Looks at the Logical Order of Pages to be accessed for Prefetch

Can speed up full table scan 10x or more

Read Ahead Configuration and Status

Configuration

- innodb_read_ahead_threshold
- innodb_random_read_ahead

Status

- Innodb_buffer_pool_read_ahead
- Innodb_buffer_pool_read_ahead_evicted
- Innodb_buffer_pool_read_ahead_rnd

Advanced Features

Encryption

Hot Topic Nowadays

MySQL and MariaDB provide different implementations

Great Blog Post bit.ly/1Sr3R0H

Encryption in MySQL

5.7.11+

Only Innodb Tablespace at this point

Supports Key Rotation

Supported by Percona Xtrabackup

Table Rebuild Required to enable

Encryption in MariaDB

- Available MariaDB 10.1.3
- Options to encrypt tables, redo log, binary log, temporary files
- Support Multiple Keys
- General Log, Audit Log, Slow Log, Galera cache are still not encrypted
- Encrypted binlog can’t be read by mysqlbinlog

What's New in MySQL 5.7 Innodb

Brief Summary of Most Important Changes

Improved to Flushing

Tuned Adaptive Flushing (again)

More Efficient page cleaners

Multiple page cleaner threads

Read Only Transactions

Treat Transaction as Read-Only until proven otherwise

Improved Locking

Metadata locking optimized

"Index Lock" Contention Reduced

Index Building Optimized

Use “Bulk” Index build instead of record insertion one by one

Fast Temporary Tables

Do not use Dictionary for Meta Data

Store in Dedicated Tablespace

Optimized UNDO/REDO Logging

Innodb is used for Internal Temporary Tables

Buffer Pool Dump/Restore

Do Buffer Pool Pool Dump/Restore by Default

Specify % of hottest pages you want to preserve

Improved Online DDLs

Now can do Online OPTIMIZE TABLE for Innodb

Double Write Optimization

Automatically disable DoubleWrite Buffer if NVMFS is Detected

Transportable Partitioned Tablespaces

Move Partitioned Tables between servers in binary form

Online Buffer Pool Resize

Warm (not Hot) Online Innodb Buffer Pool resize

Faster Crash Recovery

No need to scan all .ibd files to see which have been affected

Undo Tablespace Truncation

Finally! Prevent forever huge undo space after runaway transaction

Native Partitioning for Innodb

Important having many partitions

Less overhead opening table

Much less Memory usage

General Table spaces Support

- Create named tablespace
- Specify Tables to Use that Tablespace
- Only one file per tablespace for now
- Can assign whole tables or partitions to tablespace
- Can’t assign indexes to different tablespaces

Welcome to Barracuda and CRC32

Finally “Barracuda” file format becomes default

CRC32 Becomes default checksum format

Index Merge Threshold configuration

Help to prevent split-merge thrashing for some workloads

```
ALTER TABLE t1
COMMENT='MERGE_THRESHOLD=40';
```

Tuning Innodb

Most important configuration settings

Do not obsess with tuning

Only Fraction of these will be important for your specific workload

Tuning Innodb

innodb_buffer_pool_size

innodb_buffer_pool_instances

innodb_log_file_size

innodb_log_buffer_size

innodb_flush_log_at_trx_commit

Tuning Innodb

innodb_flush_method

innodb_io_capacity

innodb_io_capacity_max

innodb_checksum_algorithm

innodb_adaptive_hash_index

Tuning Innodb

innodb_purge_threads

innodb_flush_neighbors

innodb_change_buffer_max_size=

innodb_flush_log_at_trx_commit

innodb_stats_on_metadata

Tuning Innodb

innodb_max_dirty_pages_pct

innodb_sync_array_size

innodb_max_purge_lag

innodb_max_purge_lag_delay

Tuning Innodb

innodb_file_per_table

Innodb_thread_concurrency

innodb_file_format

Innodb_page_size

innodb_spin_wait_delay

Further Reading

http://www.percona.com/blog

http://dimitrik.free.fr/blog

http://mysqlserverteam.com

http://blog.jcole.us/

http://www.planetmysql.org

Thank You!

pz@percona.com

https://www.linkedin.com/in/peterzaitsev

https://twitter.com/peterzaitsev