SanDisk®

NVM Compression: Hybrid Flash-Aware Application Level Compression

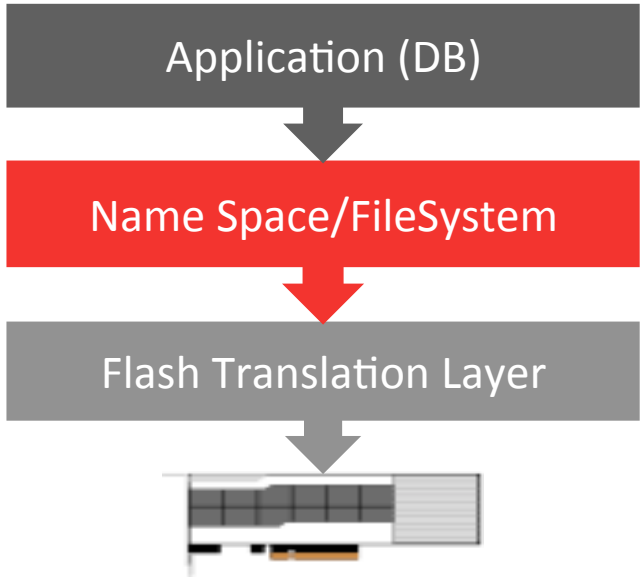
Dhananjoy Das

Sr. Architect, Advanced Development Group.

5th October 2014

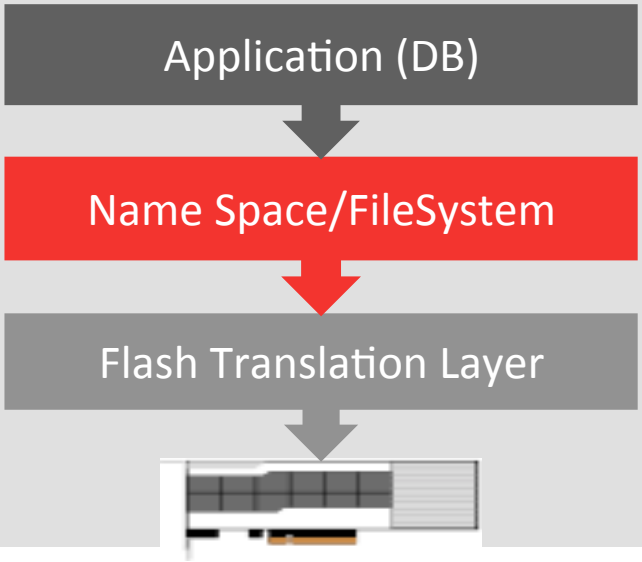
Presentation Outline

- Background
- Architecture
- Building blocks
- Evaluation/Benchmarks
- Summary
- Q & A



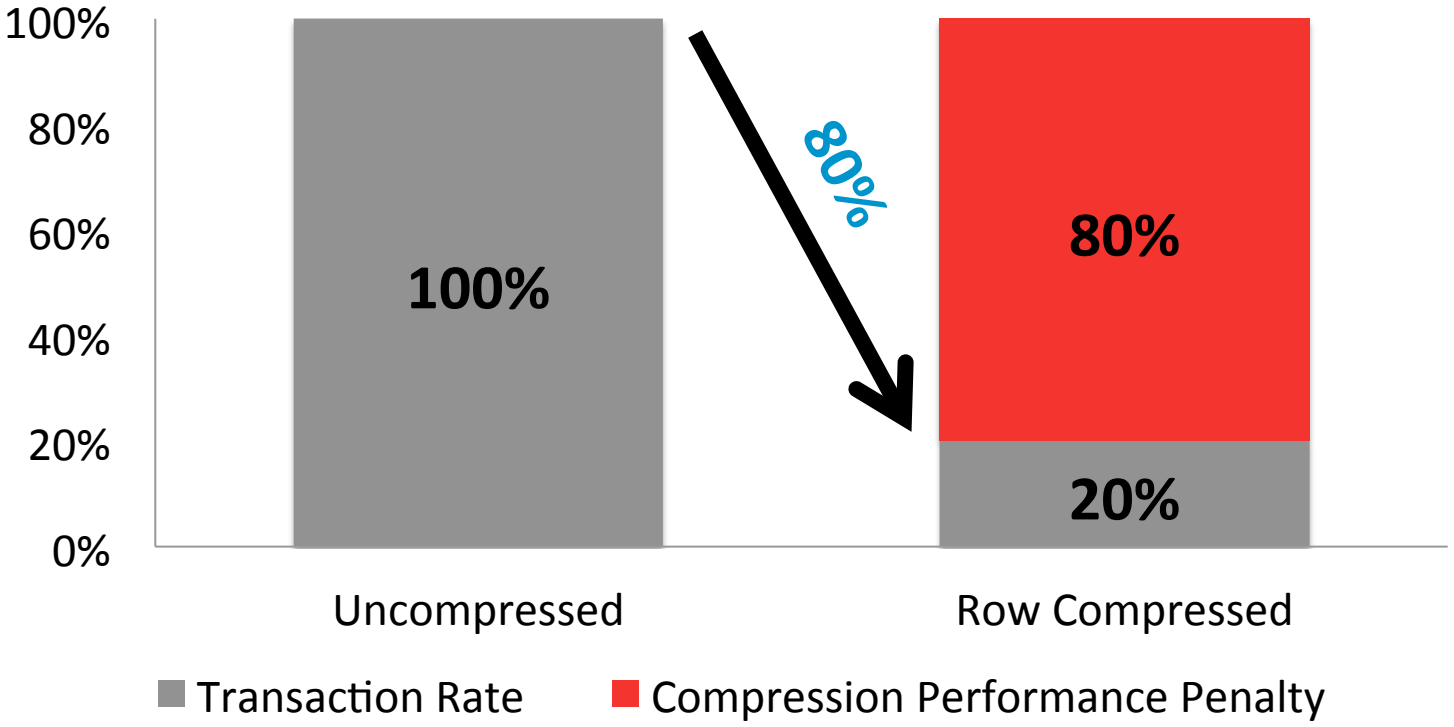
NVM-Compression solution stack

Background



SanDisk®

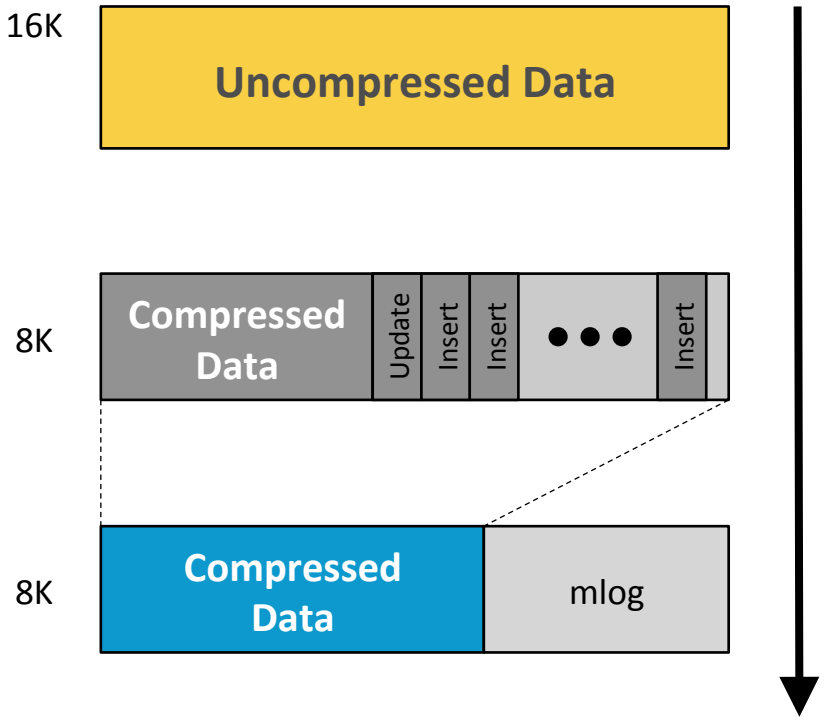
Problem: Existing MySQL ROW Compression



ROW-Compression

- Uncompressed data is stored in 16K pages
- 16K pages are compressed into a fixed compressed page size of 1K, 2K, 4K, 8K
  - Compressed page size is chosen at table creation

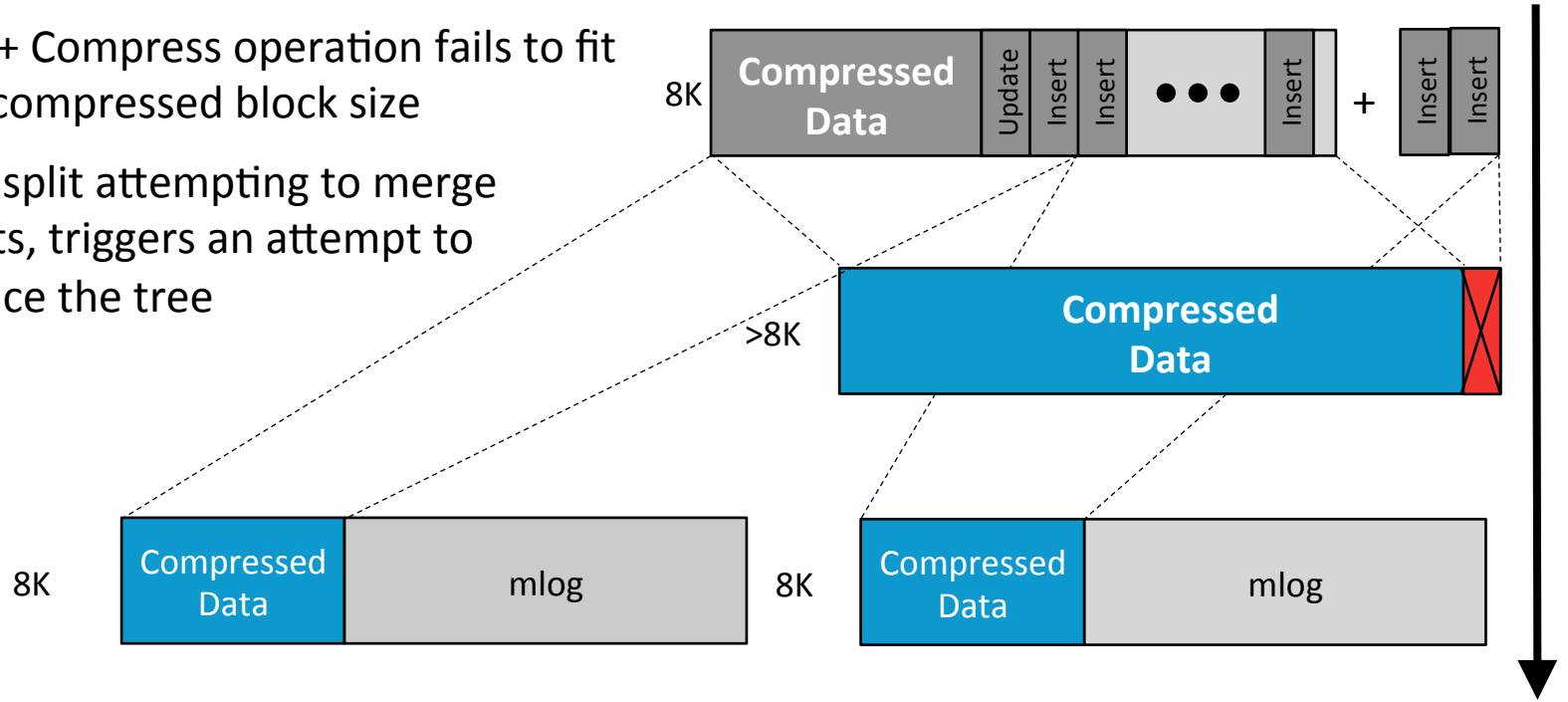
  (CREATE TABLE .. **KEY_BLOCK_SIZE=8/4/2**)
- Table updates appended to Page Modification Log (mlog) at the end of the compressed (8K) page
- When mlog gets full, page is recompressed



Row-Compression Insert Failures

Split – Rebalance – Recompress

- Merge + Compress operation fails to fit within compressed block size
- Page is split attempting to merge contents, triggers an attempt to rebalance the tree



Row-Compression Architectural Drawbacks

- Memory
  - Space: Uncompressed & Compressed pages stored in DRAM buffer pool
  - Access: Updates are applied to both copies in memory
- Capacity cap
  - Fixed compression page size - sets a upper bound on compression
- Outcome
  - Poor performance on standard benchmarks
  - Poor adoption (Less than 1% of MySQL users use compression)

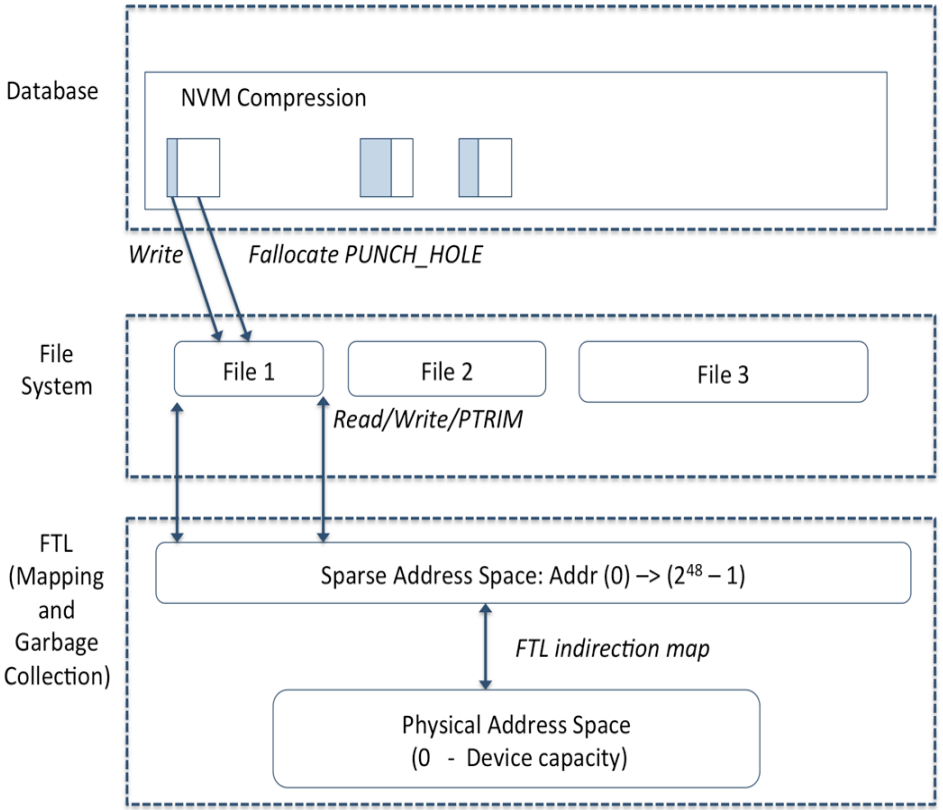
NVM Compression Architecture

SanDisk®

NVM Compression

High level approach

- Application operates on sparse address space which is always the size of uncompressed.
- Compressed data block is written in place at same virtual address as the un-compressed. Leaving a hole, empty space in the remainder of space allocated.
- FTL garbage collection naturally coalesces the addresses in physical space, allowing for re-provisioning of physical space.

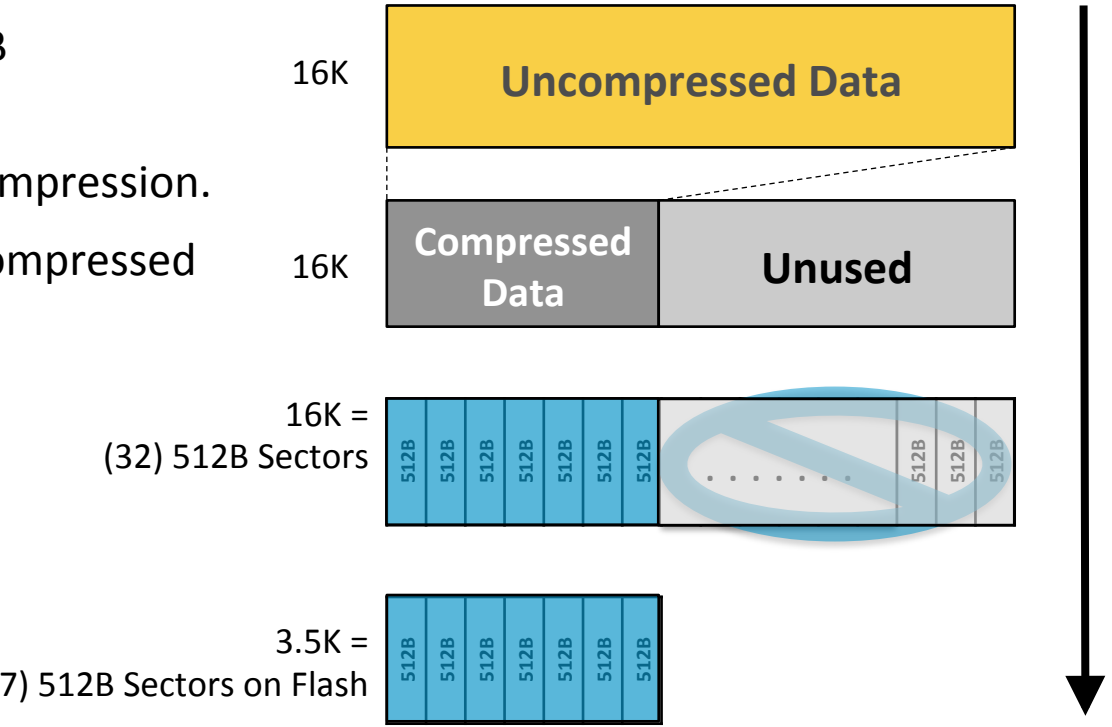


NVM Primitives

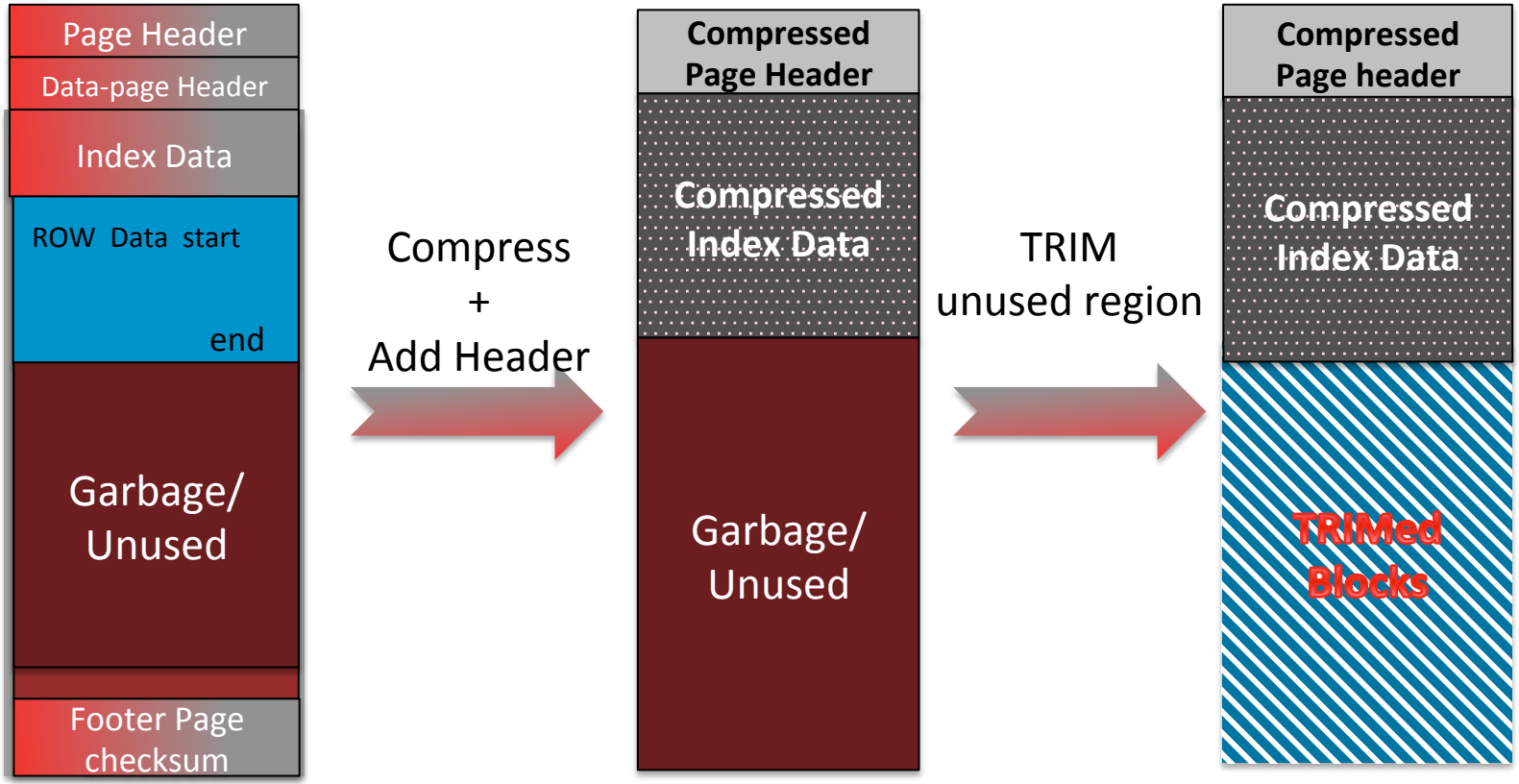
Primitives	Details
Sparse	Sparse address space, allows for mapping and allocation of blocks on FLASH only as required. Also termed as “Thin provisioning”.
PTRIMs	Persistent TRIMs, unlike regular TRIMs which are only hint to garbage-collector, are persistent and enforced across crashes to unmap block that can be reallocated.
Atomics	Atomic-write guarantees that no part of the buffer will be partially written.

NVM-Compression
(page-compression)

- Only store uncompressed 16KB pages in memory
- Update to storage results in compression.
- Trim unused 512B sectors in compressed page
- 16K uncompressed data is now compressed & “thin-provisioned” down to 3.5K on Flash



Flush operation

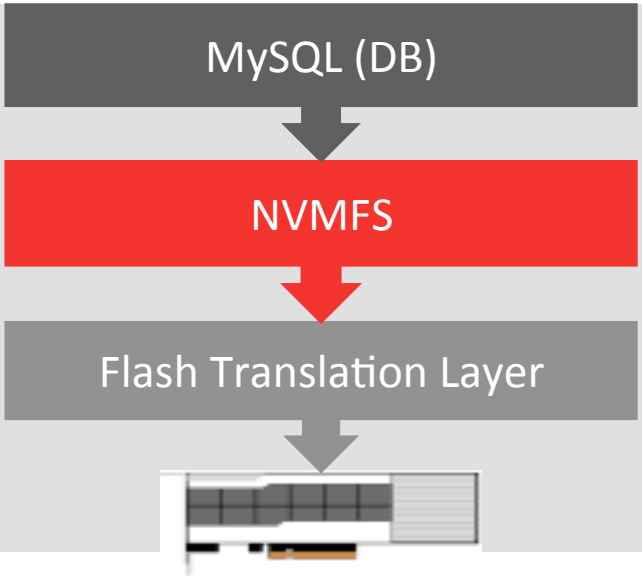


NVM-Compression Architectural Benefits

- Uses natural sparseness of the underlying storage to avoid packing of data
- Compression is performed only when dirty page is flushed
- Unused sectors are PTRIM(ed)/unmapped, for reuse
- Decompression is done on page read, and only uncompressed data is available in buffer-pool pages
- Multiple compression algorithms are pluggable

Modular design and simplified functionality

Building Blocks



SanDisk®

System Interfaces

NVM-Compression uses existing system Interfaces

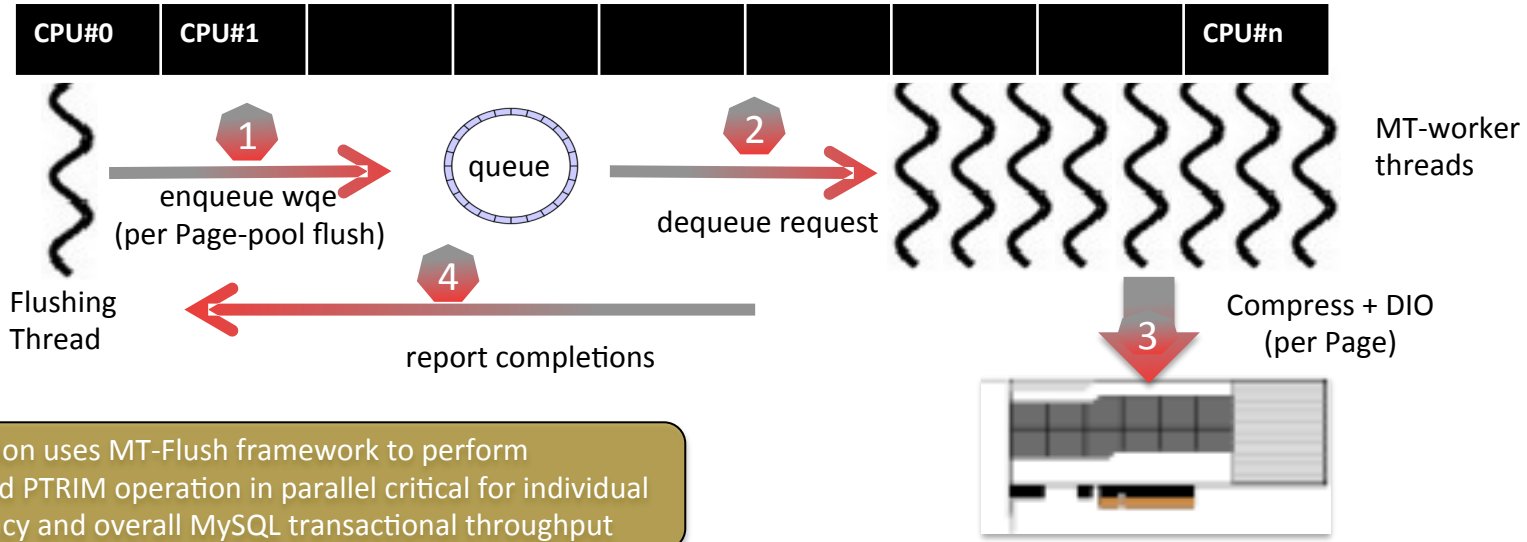
System Interface	Functionality
fallocate(offset, len)	Preallocation and extending files/table space
fallocate(**PUNCH_HOLE**)	Unmap/Punch hole operation. (issue Persistent TRIMs)
io_submit()	AIO Transparent Atomic writes

New: Multi-Threaded Flush Framework

SanDisk®

New: Multi-Threaded Flush Framework

- Producer/consumer framework for servicing scalable/distributed operation, can be used for purposes other than NVM-Compression
- Required by NVM-Compression, allows for scalable & efficient servicing of writes



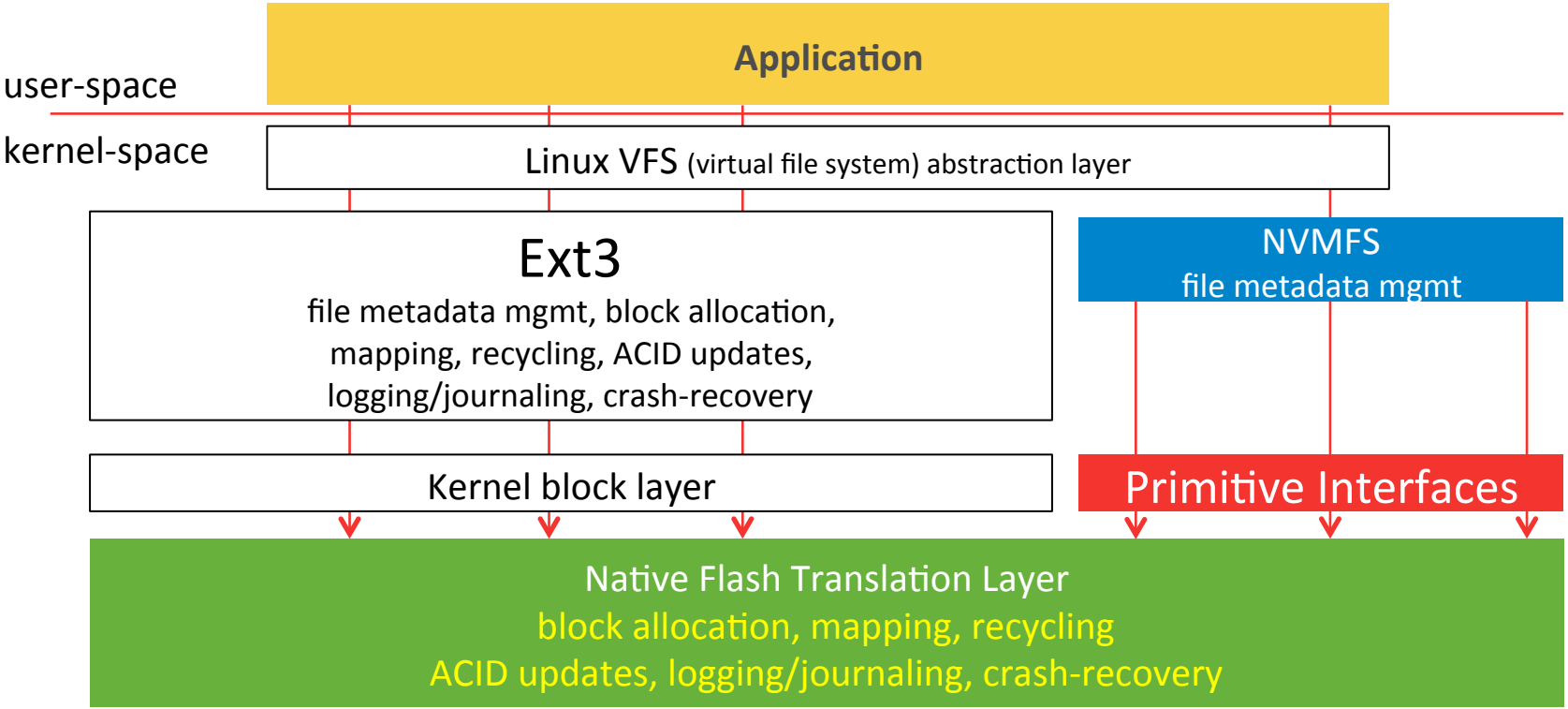
NVM Compression uses MT-Flush framework to perform compression and PTRIM operation in parallel critical for individual page-flush latency and overall MySQL transactional throughput

NVMFS

- **Non Volatile Memory FileSystem**
- It's a POSIX compliant filesystem, designed by SanDisk
- Designed to meet the needs of data intensive workloads

- **Strengths include**
  - Direct I/O on large, data intensive workloads (DB workload)
  - Pre-allocation of large files is efficient
  - No fragmentation/degradation with aging of filesystem
  - Re-exports features/**Primitives** of the underlying flash

NVMFS – Eliminating Duplicate Logic

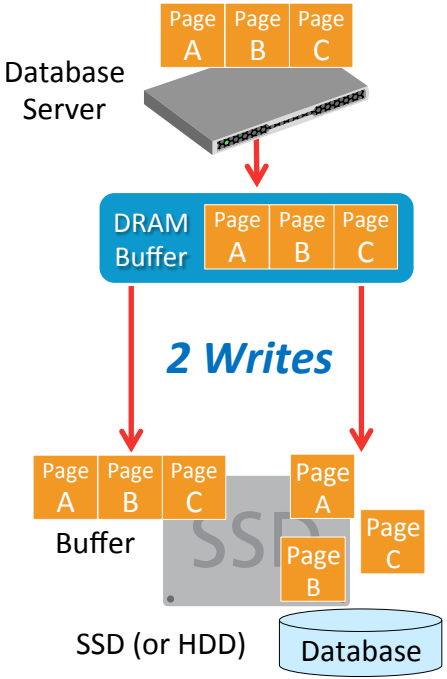


NVM Primitives

MySQL already uses Transparent Atomics

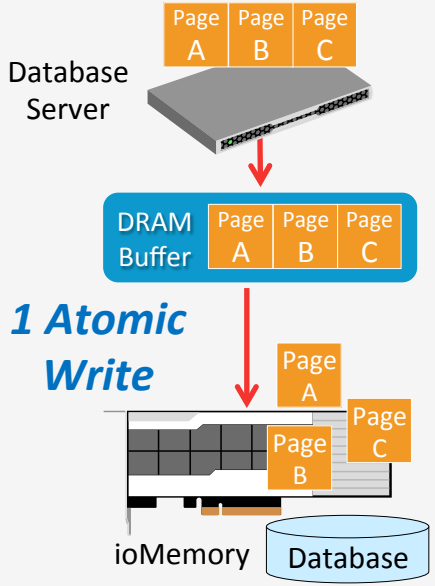
Atomic Writes (no double write)

Traditional MySQL Writes



1. Updates to pages A, B, C
2. Copied to memory buffer.
3. Writes to **double-write** buffer
4. Once acknowledged, commit to the database.

MySQL with Atomic Writes



1. Updates to pages A, B, C
2. Copied to memory buffer.
3. MySQL writes **ONCE** to database, committing the transaction with **inherent atomicity** through Atomic Writes API

Benchmarks

SanDisk®

Benchmarks Used

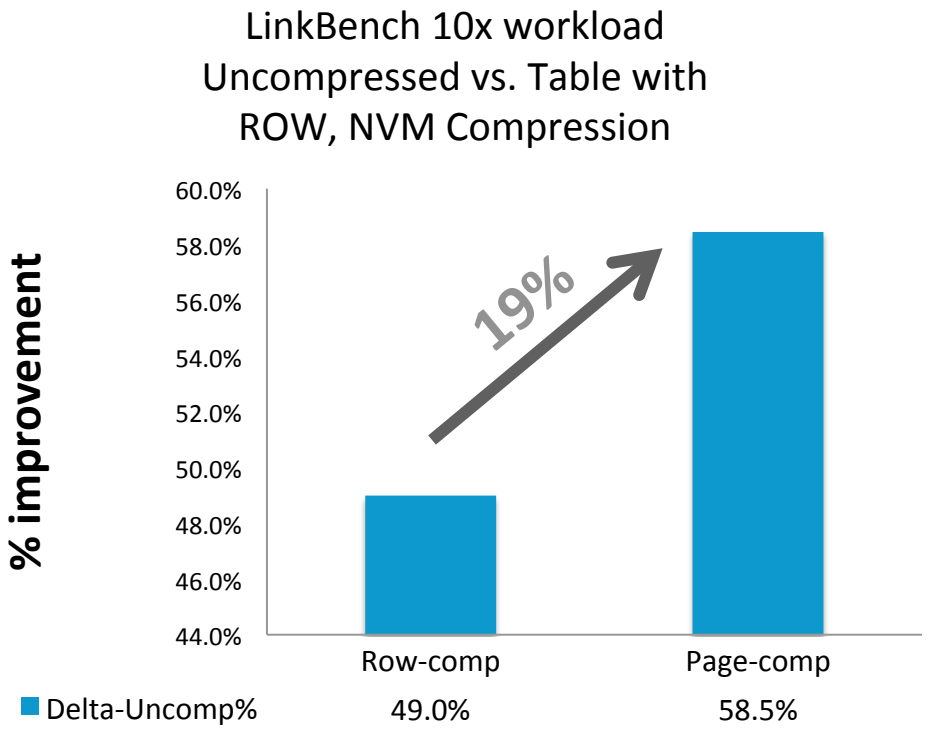
- LinkBench

  LinkBench is a database benchmark developed to evaluate database performance for workloads similar to those of Facebook's production MySQL deployment. LinkBench is highly configurable and extensible. It can be reconfigured to simulate a variety of workloads and plugins can be written for benchmarking additional database systems.

  LinkBench is released under the Apache License, Version 2.0.

- Percona tool tpcc-mysql (TPC-c like)
- Other Micro-benchmarks

Storage Savings

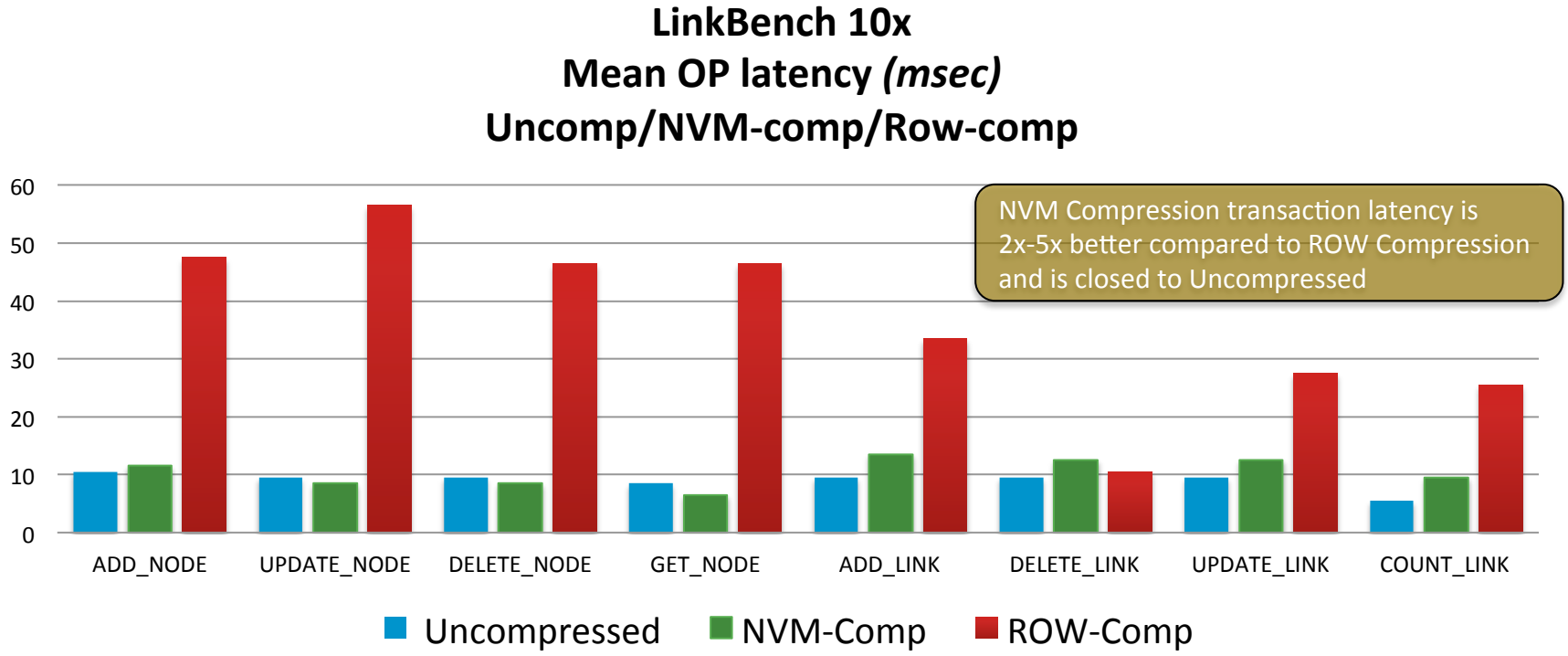


NVM Compression supports multiple pluggable compression methods including lz77, lz4, lzo

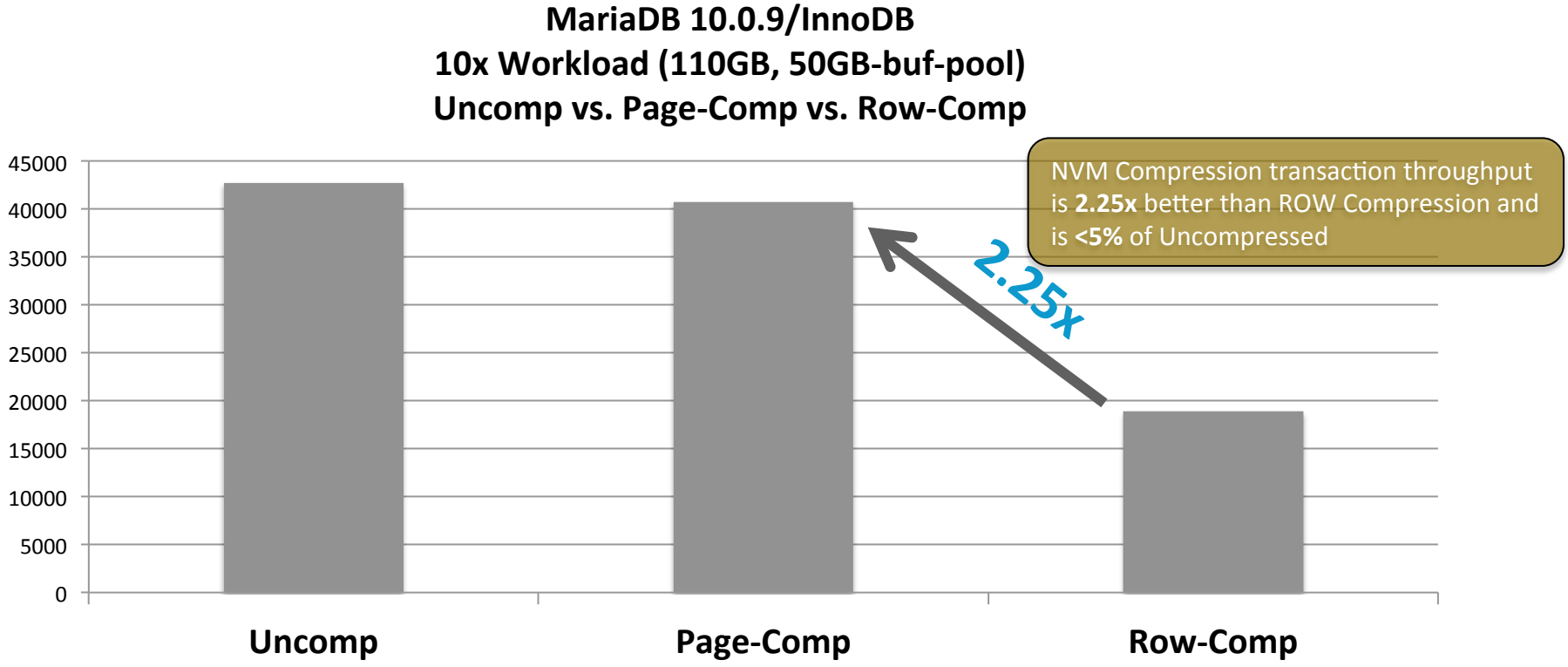
* The data show is using zlib(lz77), storage efficiency with lz4 are comparable delta ~3%

Caution:
Resulting compression ratio depends on Entropy of data along with the algorithm used

LinkBench 99% OP latencies

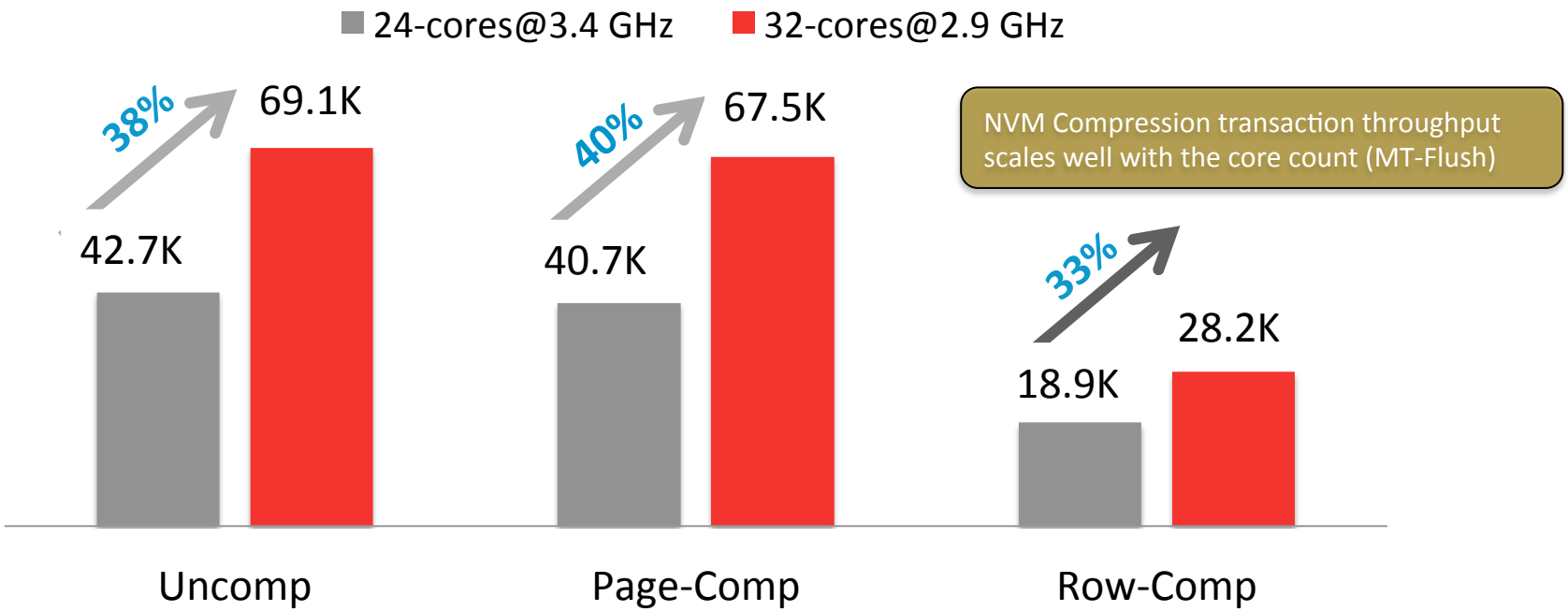


Benchmark: LinkBench

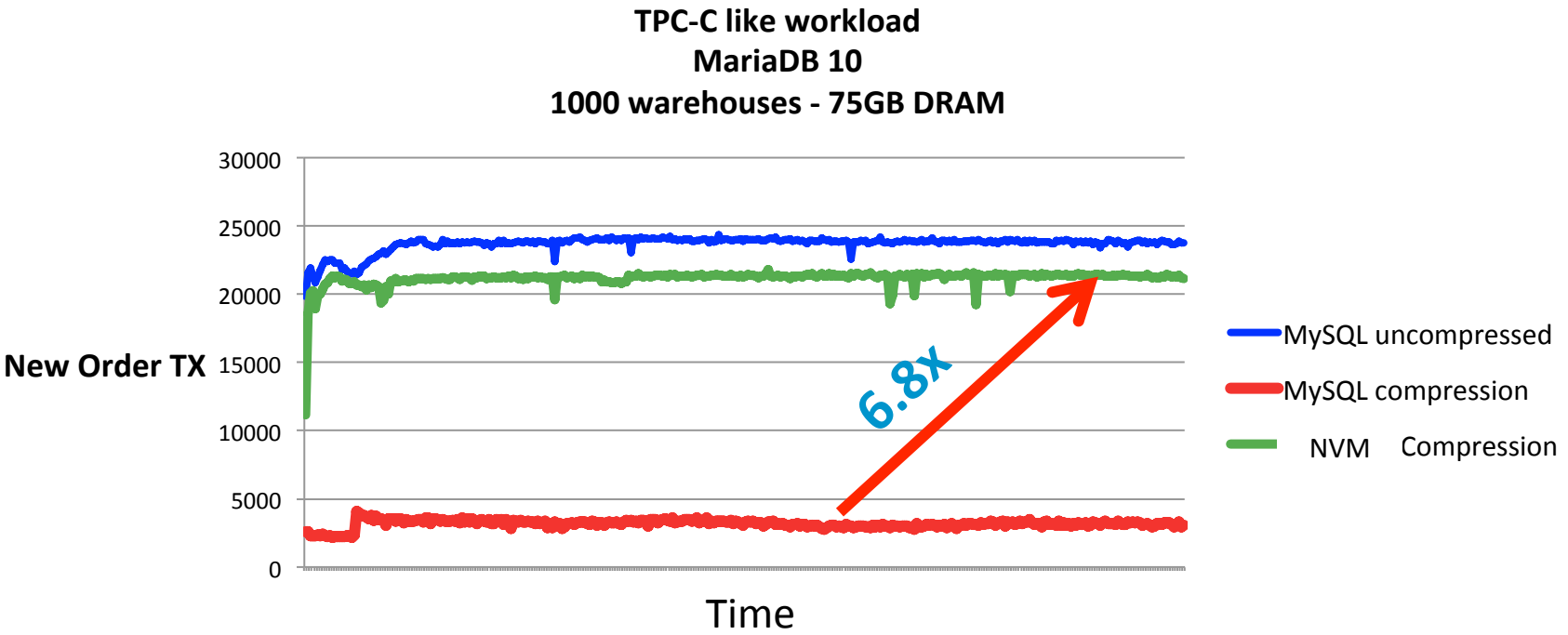


Scalability with Cores

LinkBench 10x workload (110GB, 50GB-buf-pool)
Uncomp vs. Page-Comp vs. Row-Comp



Benchmark: (TPCC-like) NVM-Compression



NVM Compression performance is **7x** better than ROW Compression and close to Uncompressed workload perf

NVMFS Performance - TRIM Handling

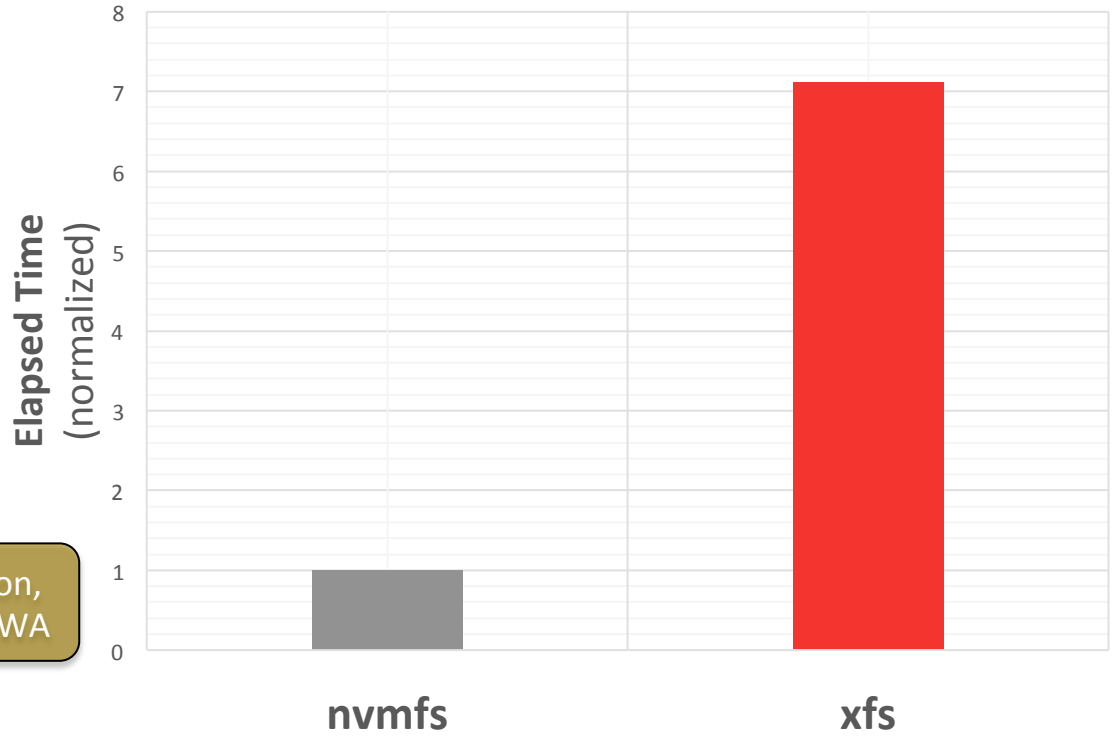
Micro-benchmark

- Trim after write
- 16 KiB Direct Write
  + 4KiB TRIM

Applications

- MySQL Page-compression

PTRIMs help with efficient garbage collection, reallocation of unused blocks and reduces WA

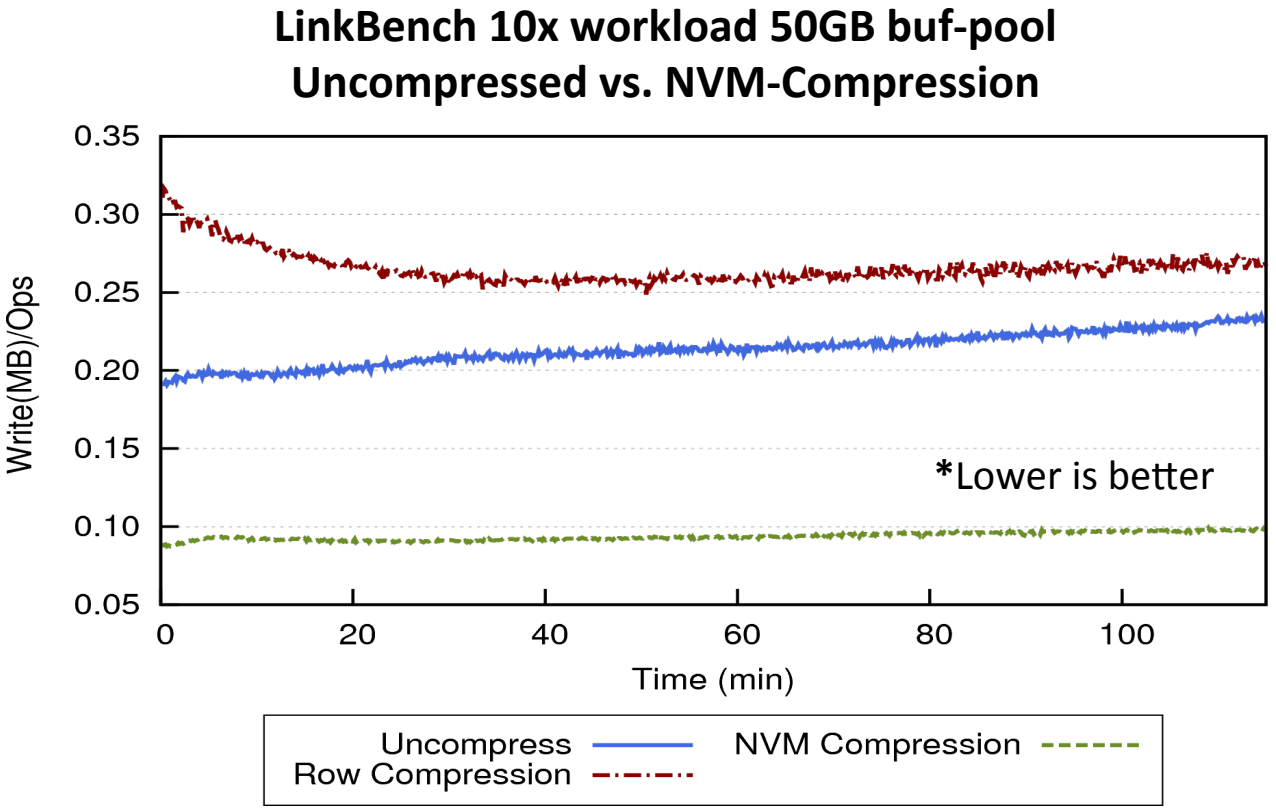


Flash Endurance

Power/Cooling

SanDisk®

Fewer Writes/OP

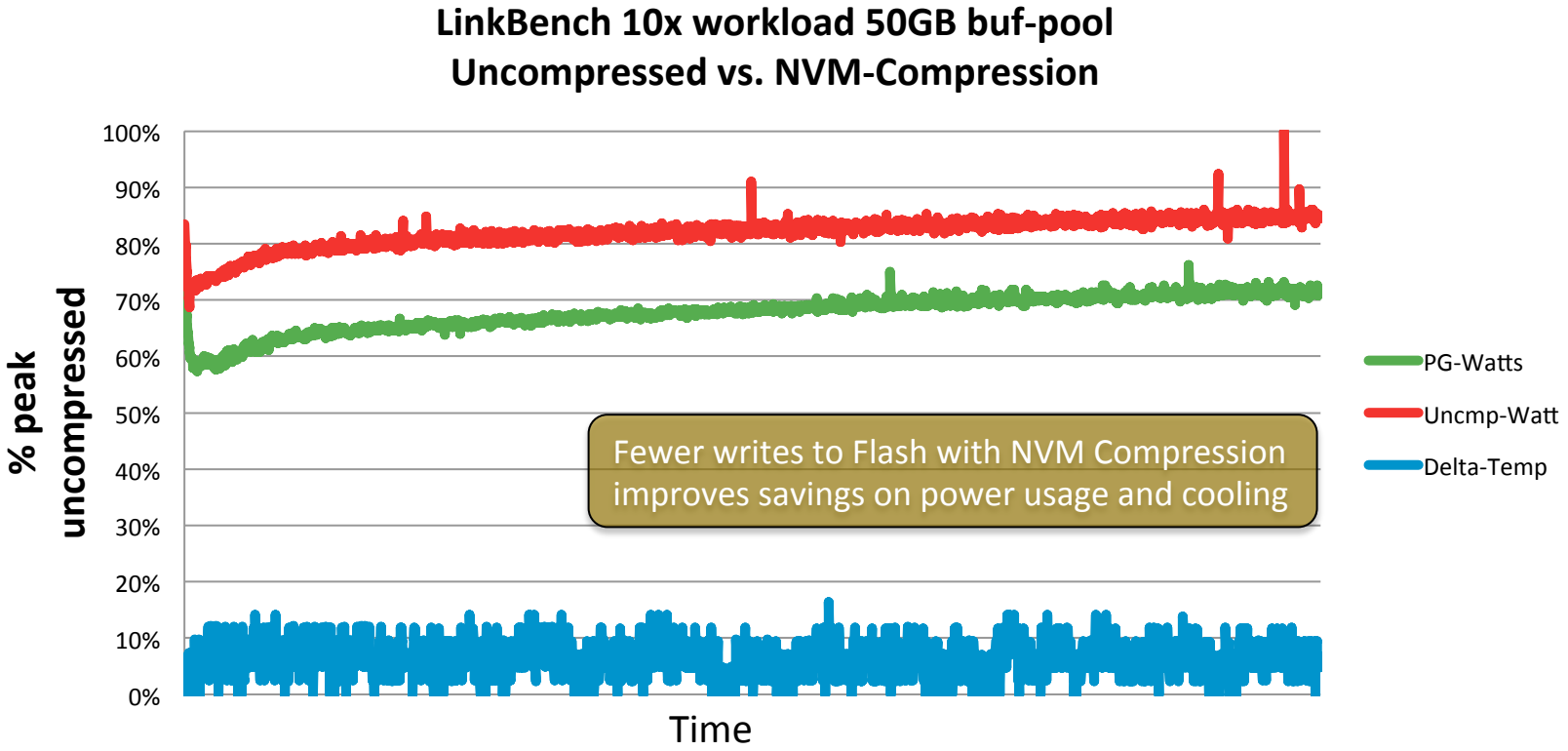


4x Better Flash Endurance

- **Compression**
  - 2x fewer writes to flash
- **Atomics**
  - 2x fewer writes by disabling double-writes
- **Persistent TRIMs**
  - Lower write amplification benefits for flash

Usable for flash with fewer write cycles, like TLC/3BPC

Power/Cooling



Summary

NVM-Compression

- Design combines application level compression with flash awareness
- Compression is performed by application at the point of page-flush
- Leverages FTL primitives for block management and garbage collection
- System standard interfaces (PTRIMs/Atomic-write)
- Pluggable compression libraries

Benchmark OLTP workload results

- Storage saving **2x+**
- Scalable & superior performance

Lessons learnt

- Reducing writes can have dramatic performance benefit – even if it comes at the cost of CPU based compression.

- Capabilities embedded in flash FTLs, like high performance GC, can be used to replace complex Application level block management very effectively.

- File system support is required to maintain manageability. File systems passing and implementing primitives efficiently is critical for performance.

Q & A

SanDisk®

Thank You

SanDisk®

Backup Slides

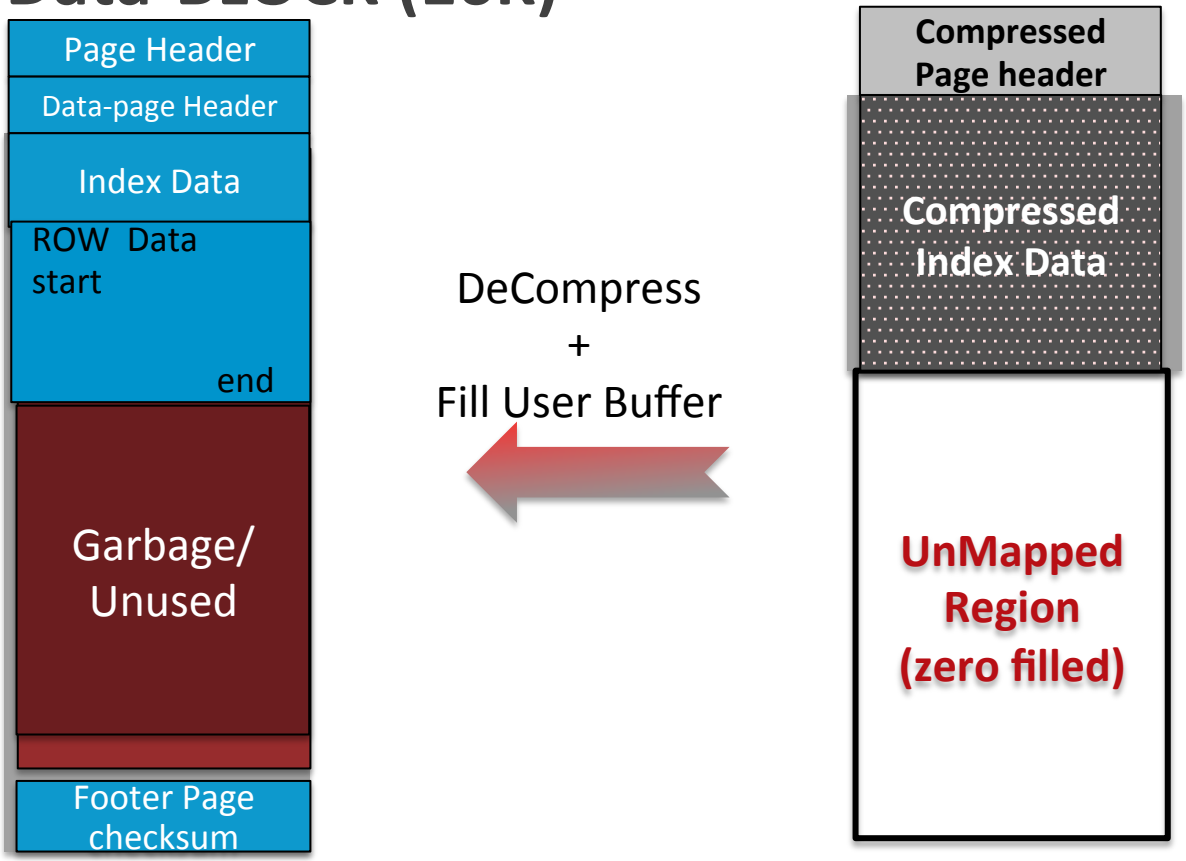
SanDisk®

Announcements

- Early access NVM-Compression solution
- Early access NVMFS
- Release by community

Vendor	Release	Download
SkySQL	MariaDB 10.0.9	http://bazaar.launchpad.net/~jplindst/maria/10.0-FusionIO
Percona	Percona Server 5.6	http://code.launchpad.net/~gl-az/percona-server/5.6-pagecomp_mtflush
Oracle	MySQL 5.7.4 DMR (Atomics)	http://dev.mysql.com
	MySQL Labs release (Compression)	http://labs.mysql.com/

Read Index-Data-BLOCK (16K)



How to NVM Compress

SanDisk®

How to NVM-Compress

- Setup MySQL (/etc/my.cnf)
- Create Table options
- Compression statistics

MySQL Setting /etc/my.cnf

```
#============================
# for atomics and fallocate
innodb_use_atomic_writes = 1
innodb_use_fallocate = 1
innodb_use_trim=1

#============================
# for MT-Flush setting
innodb_use_mtflush = 1
innodb_mtflush_threads=8
innodb_buffer_pool_instances=16

#============================
# for xtradb plugin
ignore-builtin-innodb
plugin-load=innodb=ha_xtradb.so
plugin_dir=/home/compression/10.0-FusionIO/storage/xtradb

#============================
#For lz4 you can also set
innodb-use-lz4=1

#============================
#LRU: io_capacity (DIV) buf_pool_instance
innodb_lru_scan_depth=1000
```

I/O engines supported

- Innodb
- XtraDB

Compression Protocols supported

- LZ4
- LZ77
- LZMO

Create Table
(MariaDB 10.0.9)

Sample # 1

Reserved Keywords to use for create table options

- **PAGE_COMPRESSION**
- **PAGE_COMPRESSION_LEVEL**

```
CREATE TABLE linktable (                                                \
id1 bigint(20) unsigned NOT NULL DEFAULT '0',                           \
..
..
data varchar(255) NOT NULL DEFAULT '',                                  \
PRIMARY KEY (id1,id2,link_type),                                        \
KEY id1_type (id1,link_type,visibility,time,version,data)               \
) ENGINE=InnoDB PAGE_COMPRESSED=1 PAGE_COMPRESSION_LEVEL=$compr CHARSET=latin1
PARTITION BY key(id1) PARTITIONS 16;"
```

Enable PAGE_COMPRESSION

Create Table
(MariaDB 10.0.9)

sample #2

Reserved Keywords to use for create table options

- **DIRECTORY**
- **ATOMICS_WRITES**
- **PAGE_COMPRESS options..**

```
CREATE TABLE linktable (                                    \
id1 bigint(20) unsigned NOT NULL DEFAULT '0',                \
..
..
data varchar(255) NOT NULL DEFAULT '',                       \
PRIMARY KEY (id1,id2,link_type),                             \
KEY id1_type (id1,link_type,visibility,time,version,data)    \
) ENGINE=InnoDB DIRECTORY='/mnt/nvmfs' PAGE_COMPRESSED=1     \
PAGE_COMPRESSION_LEVEL=$compr CHARSET=latin1                 \
PARTITION BY key(id1) PARTITIONS 16;"
```

Create Table under specified directory

Enable Page-compression + Atomic writes

NVM Compression: Statistics

- Show status like “Innodb_page_compression_%”
- Show status like “Innodb_n%”

```
+---------------------------------------+---------------+
| Variable_name                         | Value         |
+---------------------------------------+---------------+
| Innodb_page_compression_saved         | 6905823838720 |
| Innodb_page_compression_trim_sect512  | 20665471315   |
| Innodb_page_compression_trim_sect4096 | 1939752607    |
+---------------------------------------+---------------+
+-----------------------------------------+------------+
| Variable_name                           | Value      |
+-----------------------------------------+------------+
| Innodb_num_index_pages_written          | 1092477800 |
| Innodb_num_pages_page_compressed        | 877050078  |
| Innodb_num_page_compressed_trim_op      | 448698818  |
| Innodb_num_page_compressed_trim_op_saved | 429143865  |
| Innodb_num_pages_page_decompressed      | 873119717  |
+-----------------------------------------+------------+
```

LinkBench Setting (1x vs. 10x)

```
bin/linkbench -D dbid=linkdb -D host=127.0.0.1 -D user=root -D port=3306 -D password=xyz
-D maxid1=10000001 -c "$linkbench_mysqlconfig" -l --csvstats="$linkbench_csvstats" \
    --csvstream="$linkbench_csvstream" 2>&1 | tee "$linkbench_log")
```

1x workload

```
    bin/linkbench -c "$linkbench_mysqlconfig" -D requesters=100 -D dbid=linkdb -D host=127.0.0.1
-D user=root -D port=3306 -D password=xyz -D maxid1=10000001 -r --csvstats="$linkbench_csvstats"
 -D requests=50000000 -D maxtime=3600 -r --csvstream="$linkbench_csvstream" 2>&1 | tee "$linkbenc
h_log")
```

- Bump up the workload from default 1x (10Mil) to 10x using "**maxid**"
- Change runtime, using "**maxtime**"

OK What Happened with Latency ?

