facebook

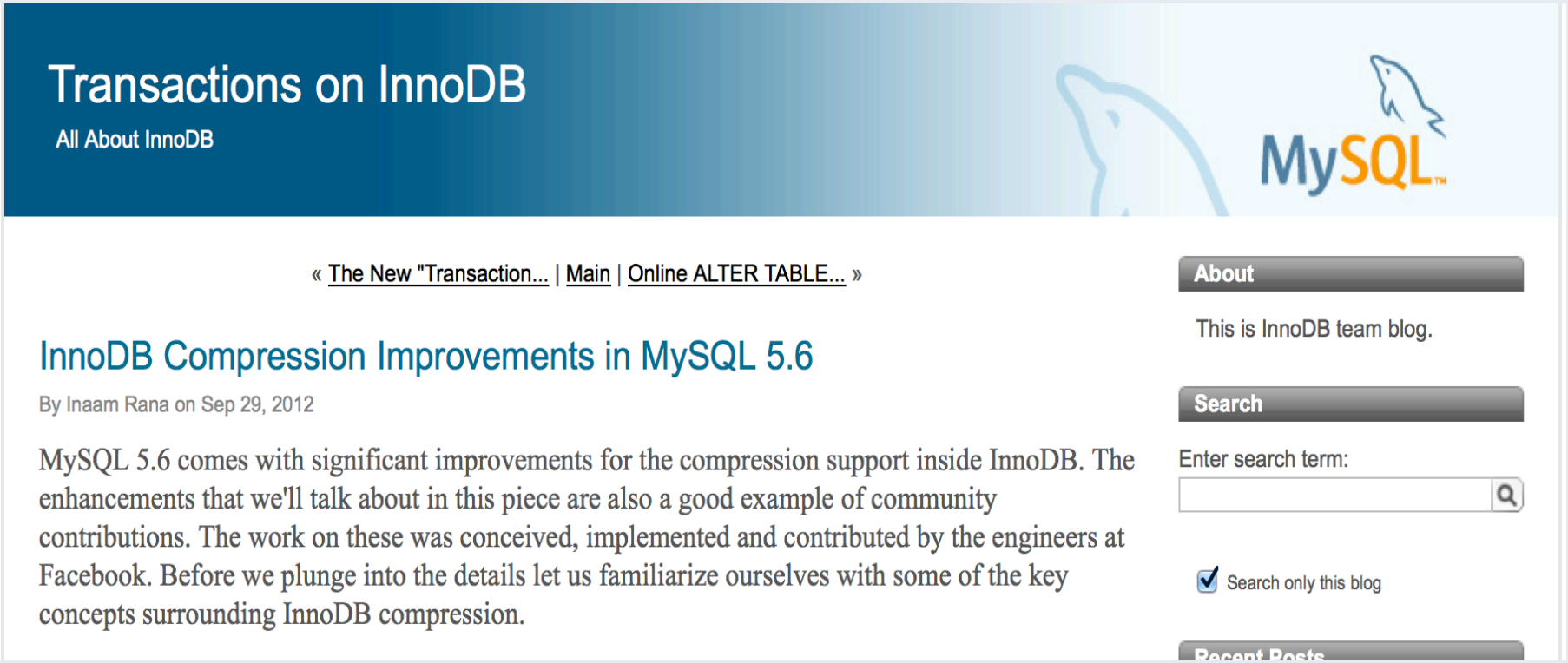
InnoDB Compression Present and Future

Nizameddin Ordulu nizam.ordulu@fb.com, Justin Tolmer jtolmer@fb.com
Database Engineering @Facebook

facebook

Agenda

- InnoDB Compression Overview
- Adaptive Padding
- Compression using 32K Pages
- Performance Issues with KEY_BLOCK_SIZE=4
- Work in Progress
- Questions



Transactions on InnoDB

All About InnoDB

MySQL

« The New "Transaction... | Main | Online ALTER TABLE... »

InnoDB Compression Improvements in MySQL 5.6

By Inaam Rana on Sep 29, 2012

MySQL 5.6 comes with significant improvements for the compression support inside InnoDB. The enhancements that we'll talk about in this piece are also a good example of community contributions. The work on these was conceived, implemented and contributed by the engineers at Facebook. Before we plunge into the details let us familiarize ourselves with some of the key concepts surrounding InnoDB compression.

About

This is InnoDB team blog.

Search

Enter search term:

Search only this blog

Recent Posts

facebook

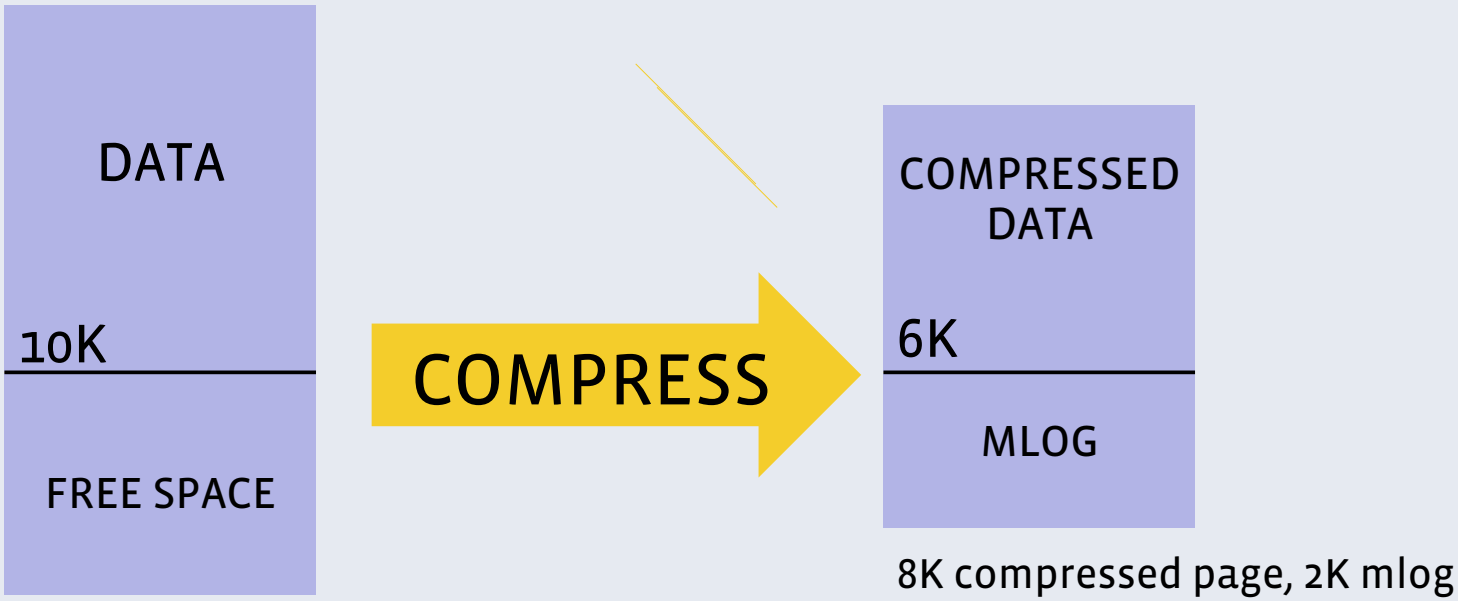
InnoDB Compression Overview

InnoDB Compression Overview

- Compression is enabled per table:
  - CREATE TABLE users(
    id int PRIMARY KEY,
    email varchar(255)
) ENGINE=InnoDB ROW_FORMAT=COMPRESSED KEY_BLOCK_SIZE=8;
- Uncompressed page is 16K by default.
- Compressed block size can be 8K, 4K, 2K, or 1K.
- Only the compressed pages are written to disk.

InnoDB Compression Overview

- Free space on the compressed page is called ***modification log (mlog)***.

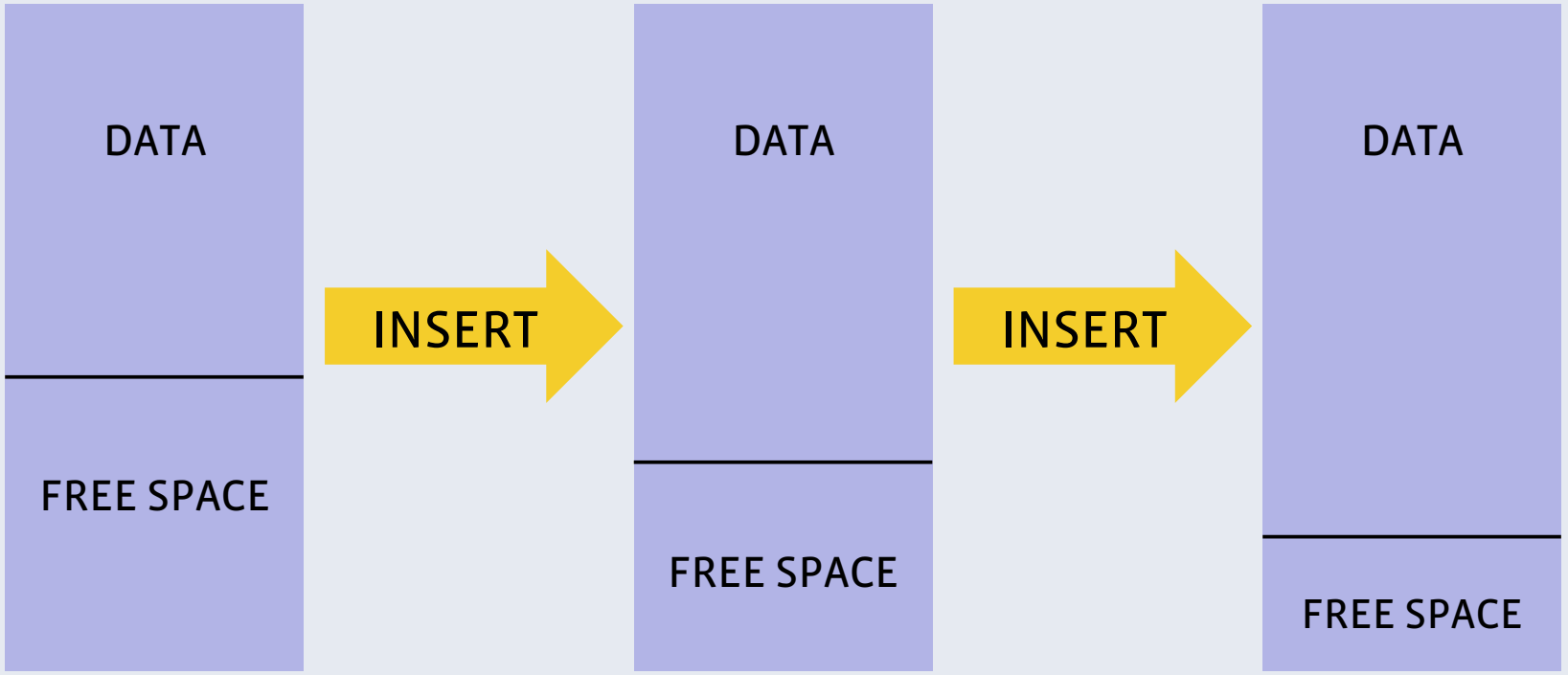


16K uncompressed page

8K compressed page, 2K mlog

InnoDB Compression Overview

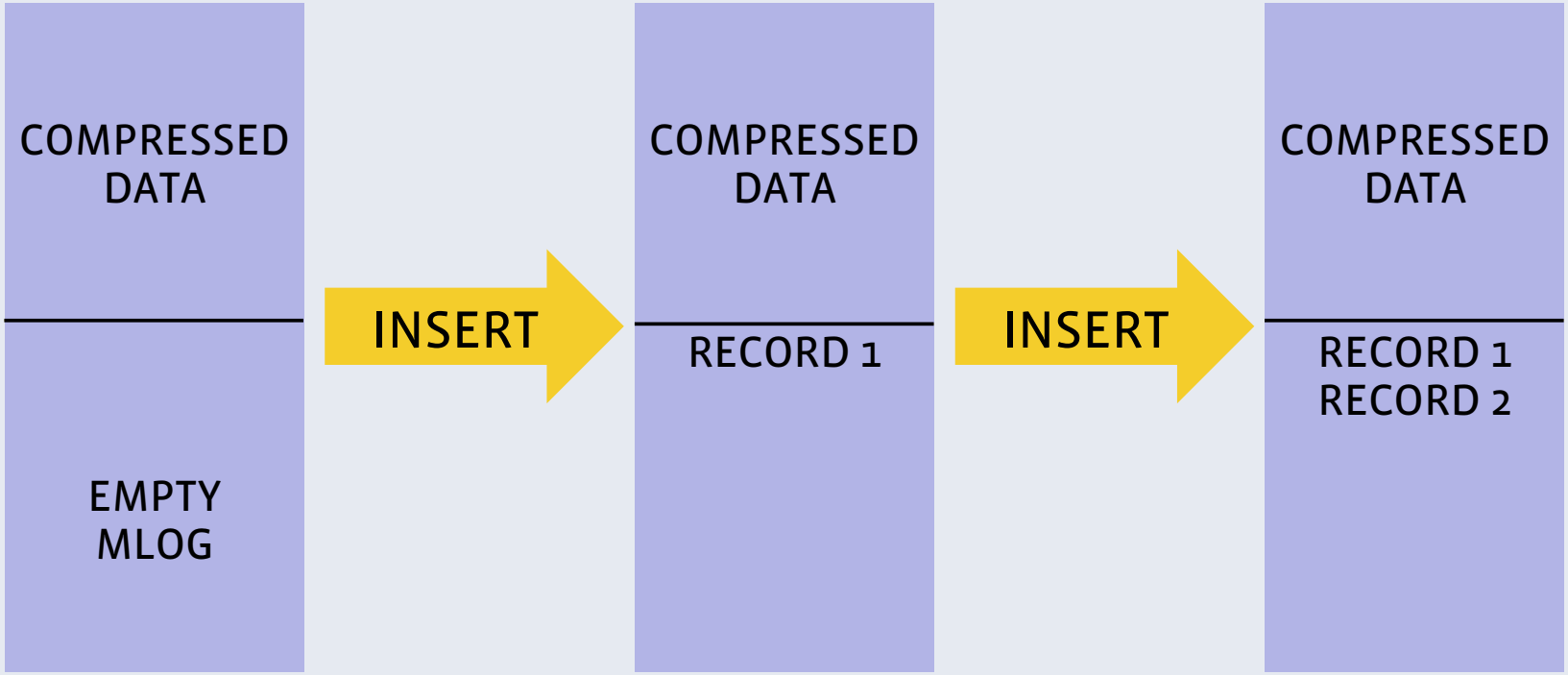
- Consecutive inserts to an uncompressed page use the free space on the uncompressed page.



Uncompressed Page

InnoDB Compression Overview

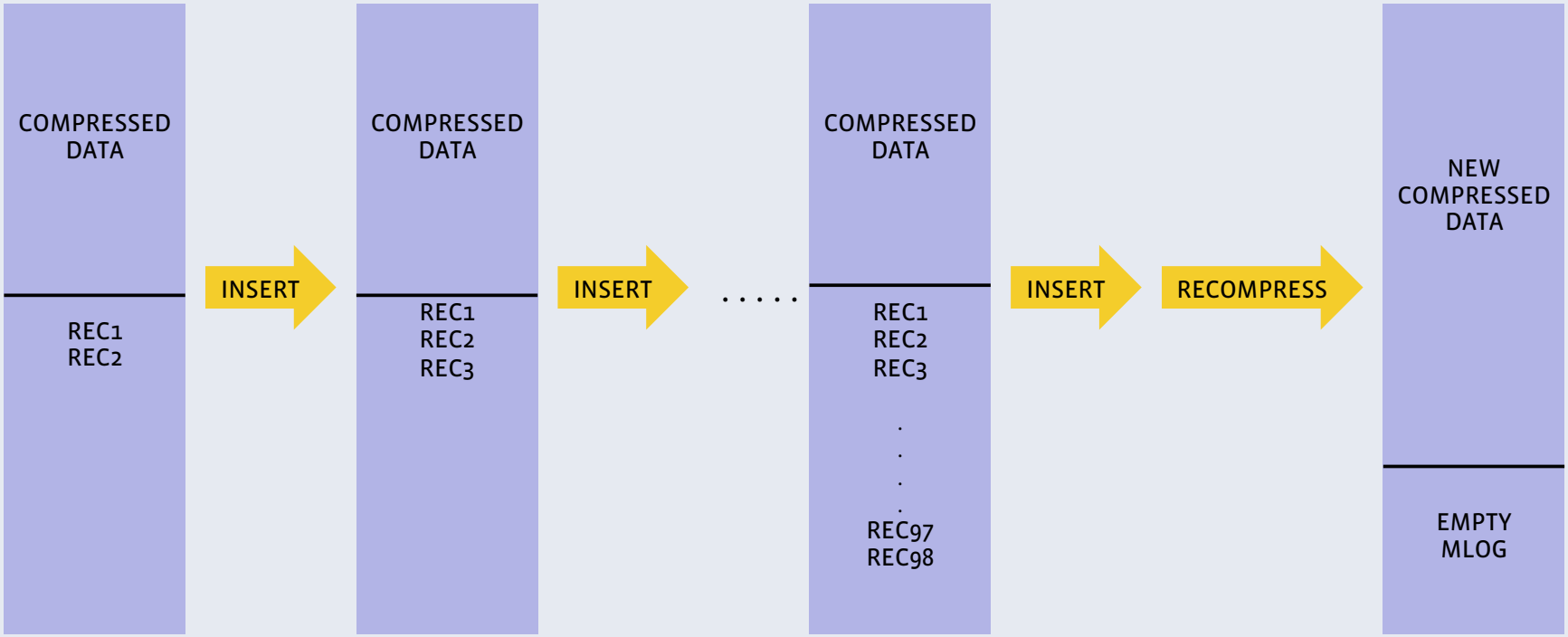
- Consecutive inserts to a compressed page use ***mlog***.
- Compressed pages and uncompressed pages are kept in sync.



Compressed Page

InnoDB Compression Overview

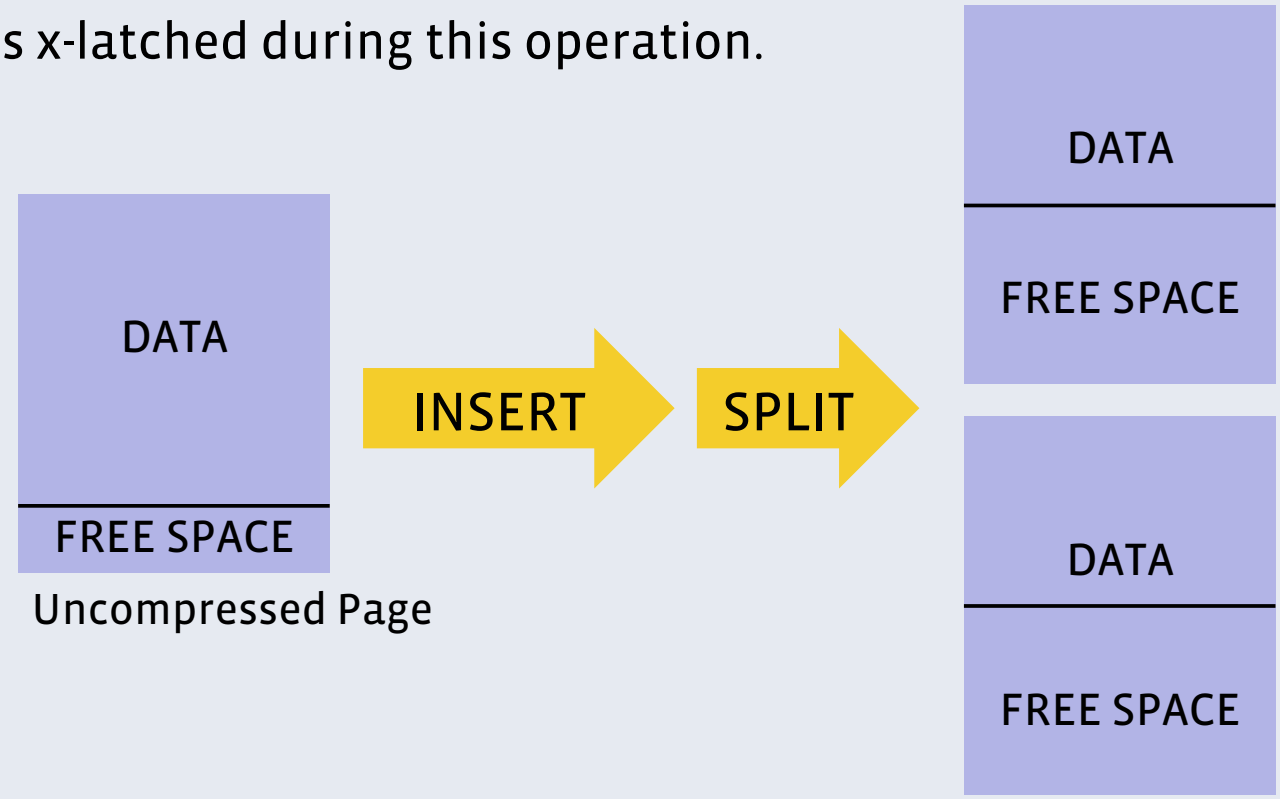
- When the ***mlog*** is full page is re-compressed, the new page will have empty ***mlog***. This is called a ***recompression.***



Compressed Page

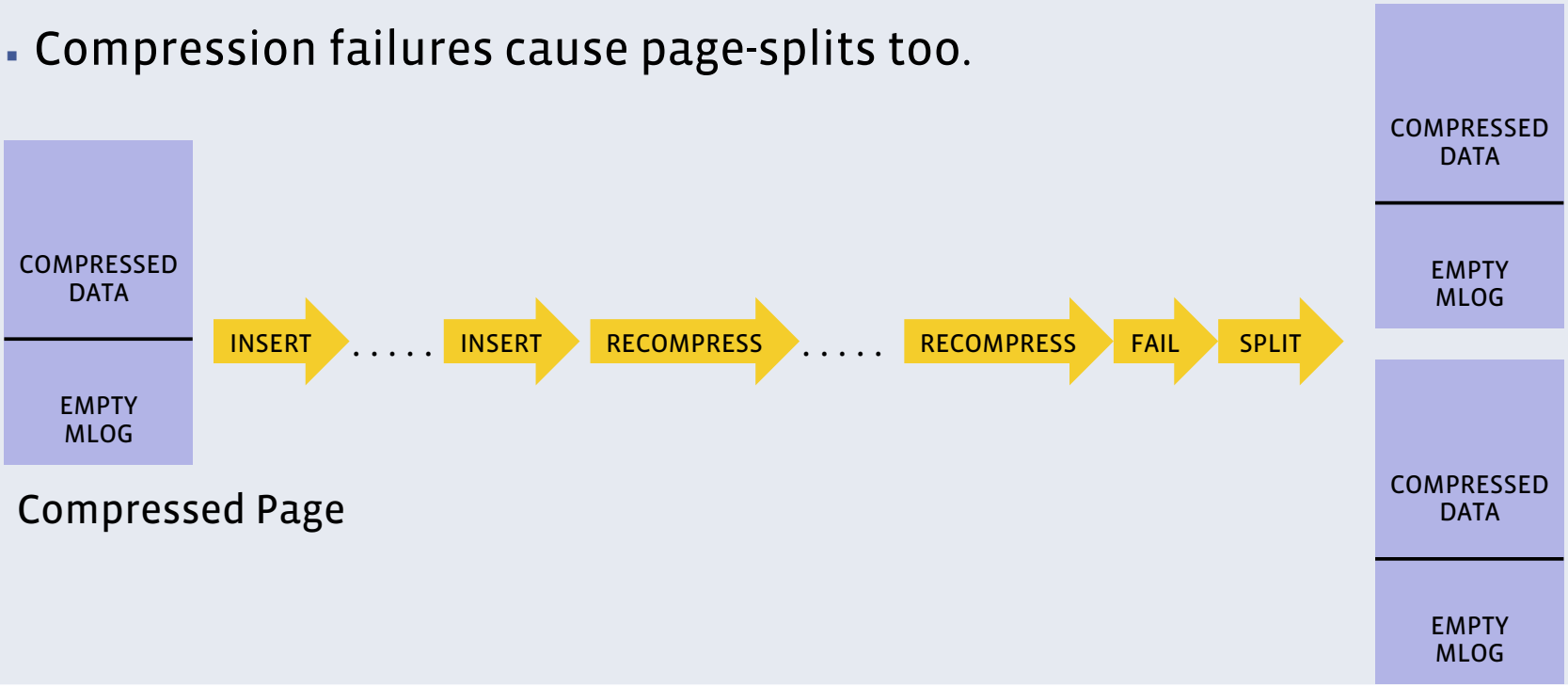
InnoDB Compression Overview

- A ***page-split*** happens when there is not enough free space on the uncompressed page.
- The B-tree lock is x-latched during this operation.



InnoDB Compression Overview

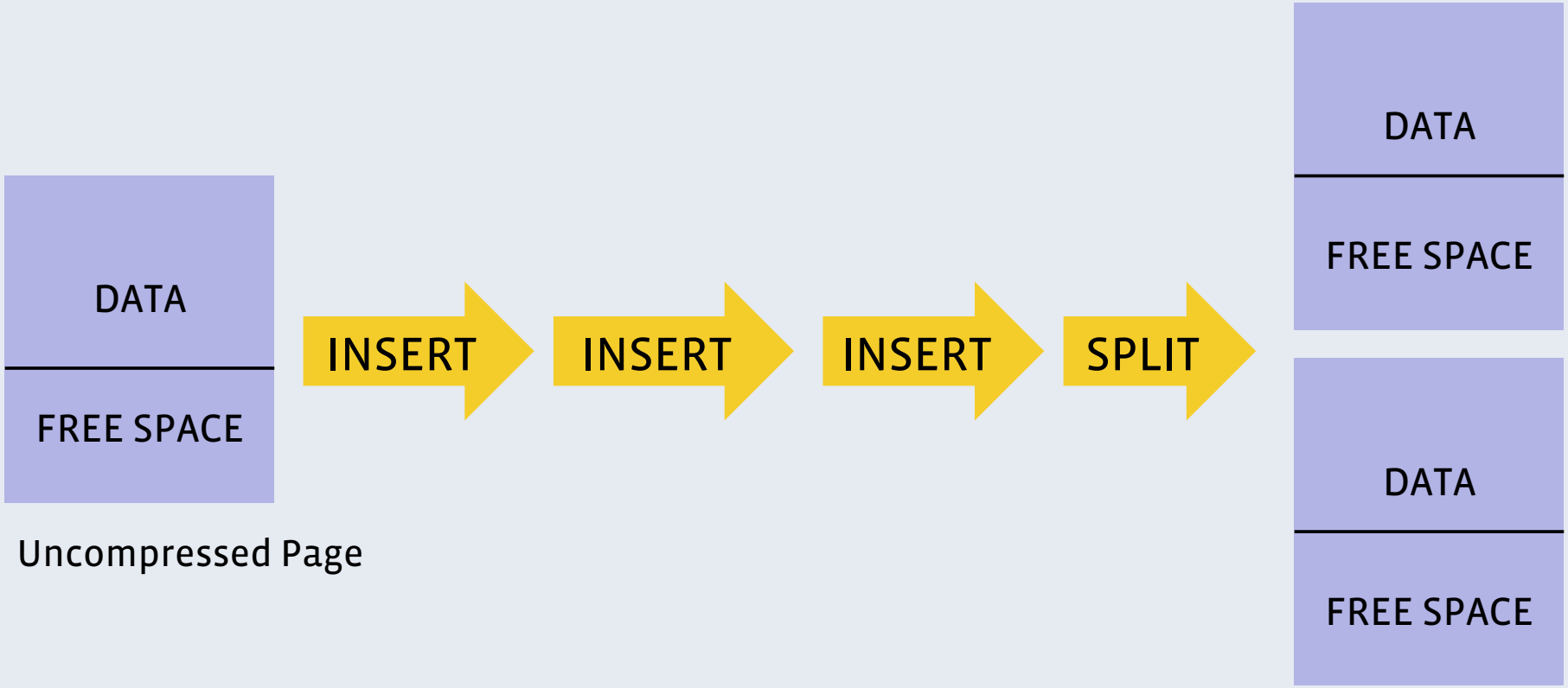
- Some recompressions will fail because of the lack of space on the compressed page.
- These are called ***compression failure***s.
- Compression failures cause page-splits too.



Compressed Page

InnoDB Compression Overview

- Compression failures do not happen if the table data is compressible enough.



Uncompressed Page

InnoDB Compression Overview

- Compressing a page is an expensive operation.
- Failure means we spent the precious cpu cycles for nothing.
- Compression failures while holding an x-lock on B-tree are even worse.
- The failure rate for compression operations was 40% on our test servers.

facebook

Adaptive Padding

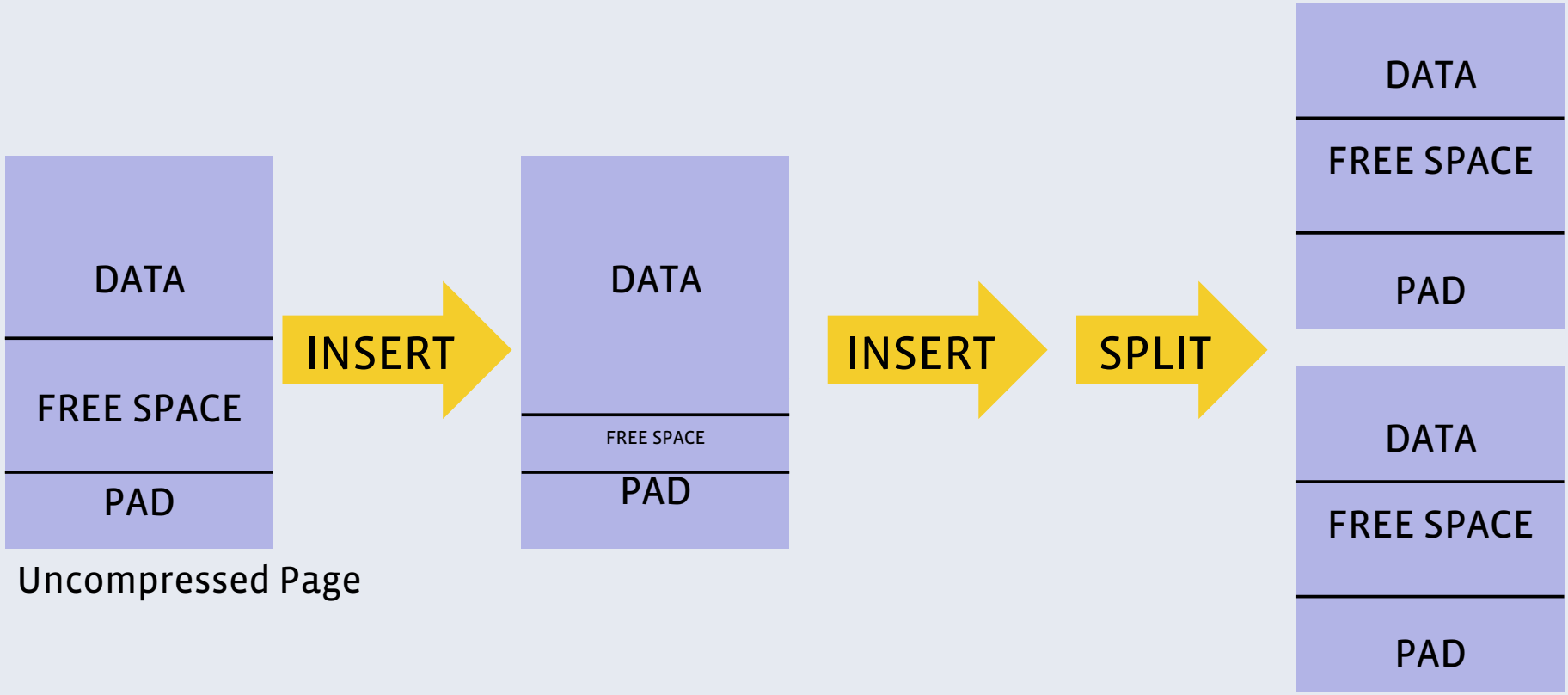
Adaptive Padding

- Idea: Restrict the amount of data on the uncompressed page to keep it compressible.
- Pseudocode for pad calculation:

```
pad = 0
while 1:
    wait for N compression operations
    if failure_rate > desired_failure_rate:
        pad += 128
    if failure_rate < desired_failure_rate:
        pad -= 128
```

Adaptive Padding

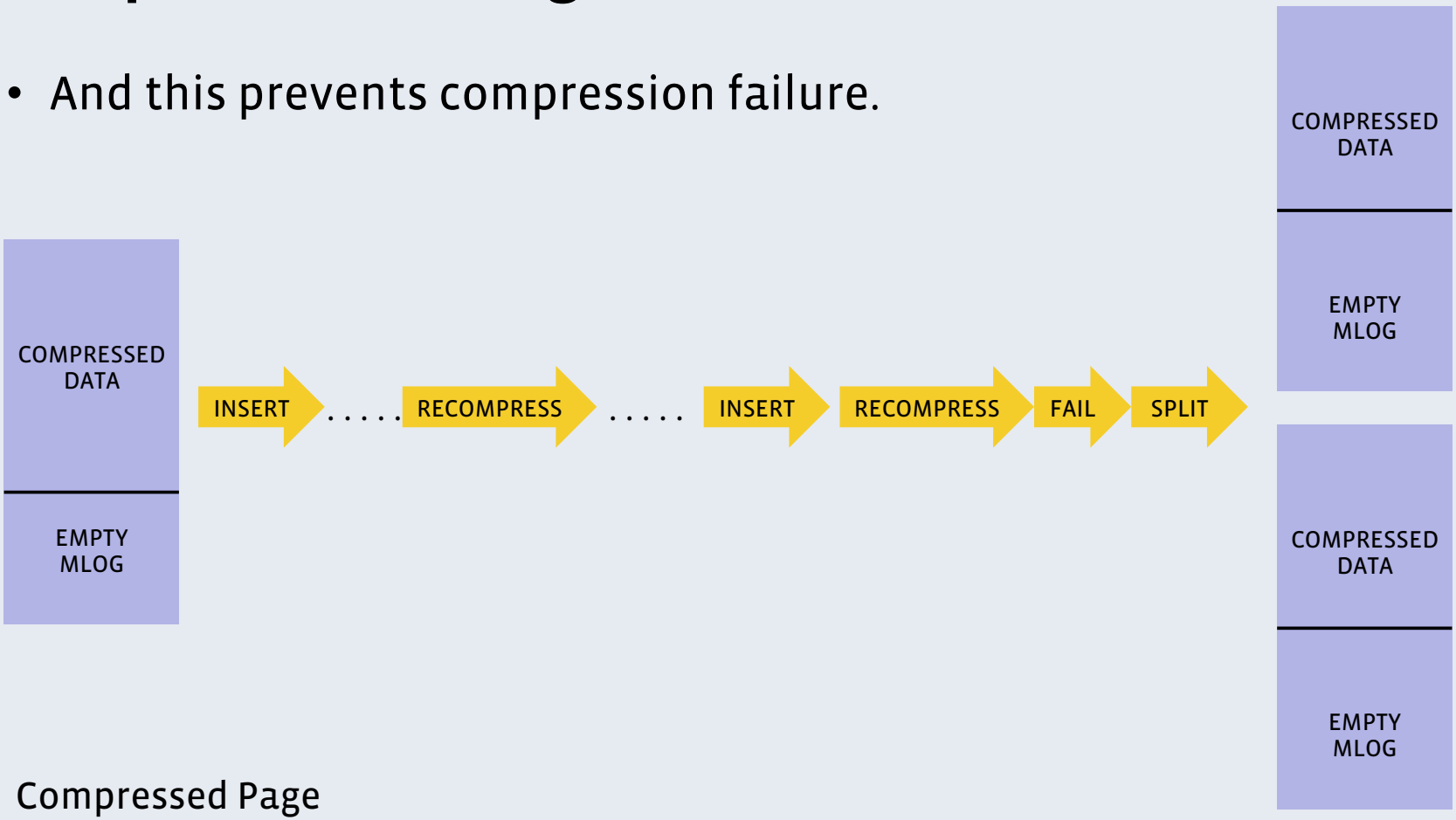
- Uncompressed page splits early.



Uncompressed Page

Adaptive Padding

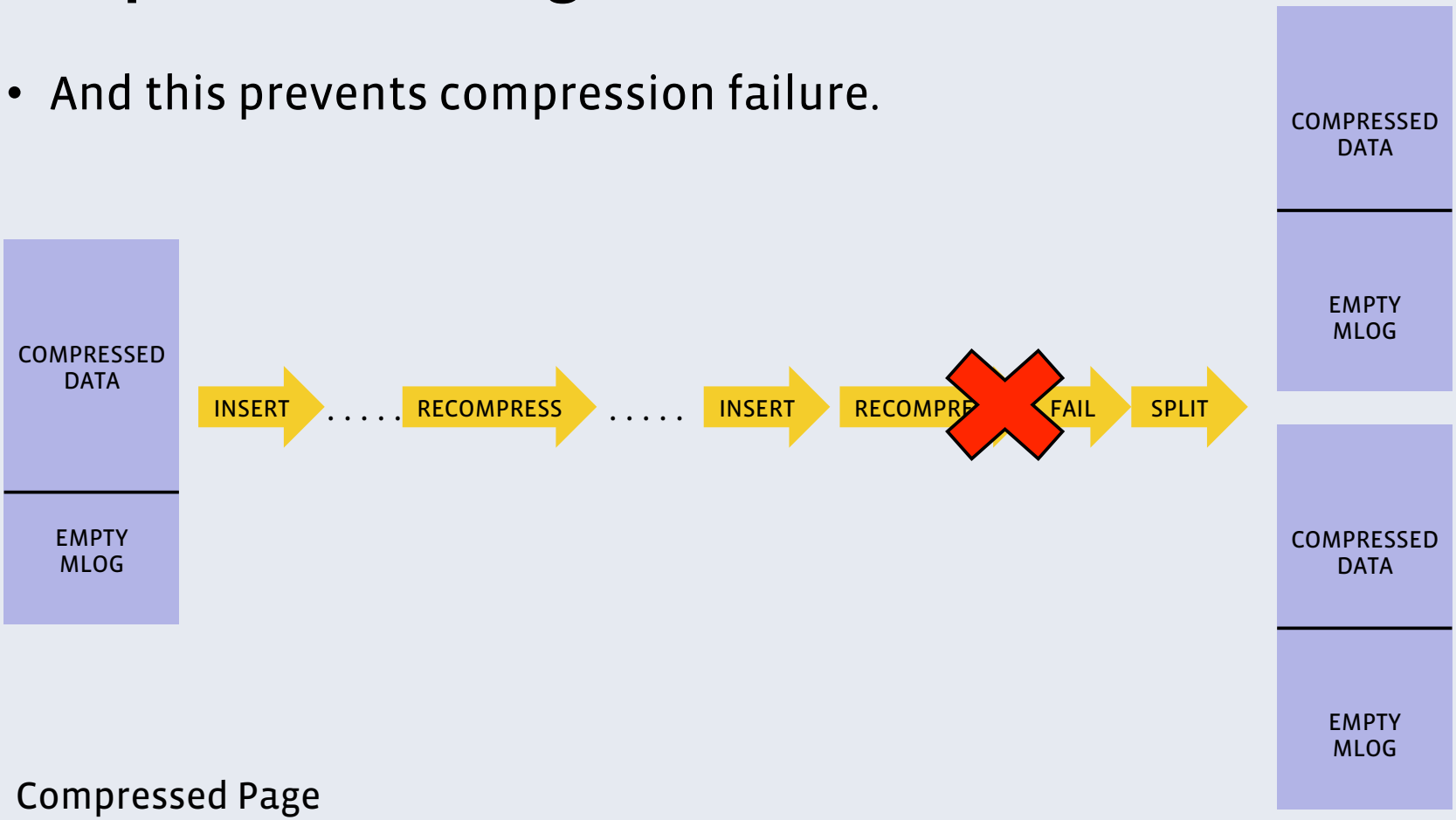
- And this prevents compression failure.



Compressed Page

Adaptive Padding

- And this prevents compression failure.

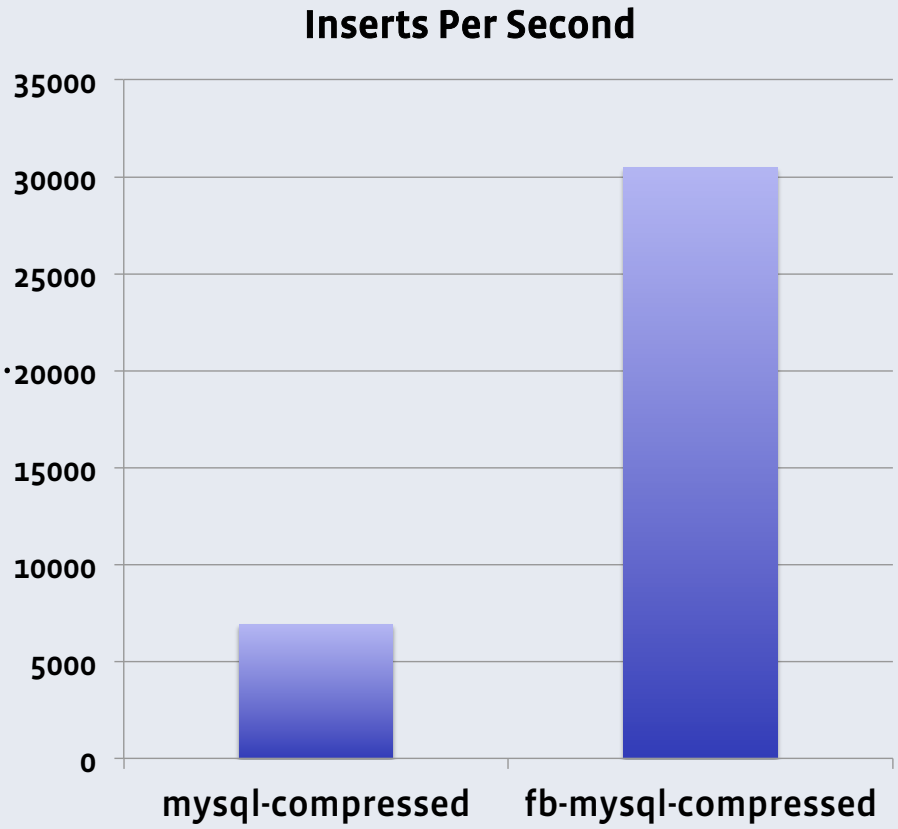


Compressed Page

Adaptive Padding

Insert Benchmark with Sysbench

- Compression failure rate:
  - mysql: 41%.
  - fb-mysql: 5%.
- No difference in on-disk table size.



Adaptive Padding

- Configuration variables for adaptive padding in 5.6:
  - innodb_padding_max_fail_rate: max compression failure rate.
  - innodb_padding_max: maximum padding value as a percentage of the page size.

facebook

32K InnoDB Pages?

Why 32K?

- Default InnoDB page size is 16K
- Compressed page format includes:
  - Page header
  - Compressed row data
  - Uncompressed modification log
  - Uncompressed metadata (transaction ID, rollback pointer, etc.)
- Possible benefits:
  - Smaller uncompressed data to page size ratio
  - Larger blocks may compress better

Implementation

- 5.6 includes --innodb-page-size
  - Max 16K
- Increase UNIV_PAGE_SIZE_SHIFT_MAX to 15
- Easy, right?

Problem #1

- Compilation error:

```
/home/jtolmer/mysql/56/storage/innobase/include/fsp0fsp.ic:287:4: error:
#error
```

- From code:

```
# if UNIV_PAGE_SIZE_MAX <= XDES_ARR_OFFSET                              \
                         + (UNIV_PAGE_SIZE_MAX / FSP_EXTENT_SIZE_MAX) \
                         * XDES_SIZE_MAX
#  error
# endif
```

- The extent descriptor array no longer fit on a single page!

Solution #1

- Increase extent size to 2M:

```
/** File space extent size in pages, one megabyte for pages sizes <= 16
kilobytes and two for larger page sizes */
#define  FSP_EXTENT_SIZE    ((UNIV_PAGE_SIZE <= (1 << 14) ?    \
          1048576U : 2097152U) / UNIV_PAGE_SIZE)
```

- Hard to track down bug in fil_node_open_file():

```
if (size_bytes >= 1024 * 1024) {
  /* Truncate the size to whole megabytes. */
  size_bytes = ut_2pow_round(size_bytes, 1024 * 1024);
}
```

Problem #2

- Dense directory on compressed page:

```
/** Size of an compressed page directory entry */
#define PAGE_ZIP_DIR_SLOT_SIZE  2
/** Mask of record offsets */
#define PAGE_ZIP_DIR_SLOT_MASK  0x3fff
/** 'owned' flag */
#define PAGE_ZIP_DIR_SLOT_OWNED 0x4000
/** 'deleted' flag */
#define PAGE_ZIP_DIR_SLOT_DEL   0x8000
```

- 14 bits too small to store 32K offset

Solution #2

- Increase slot size to 3 bytes:

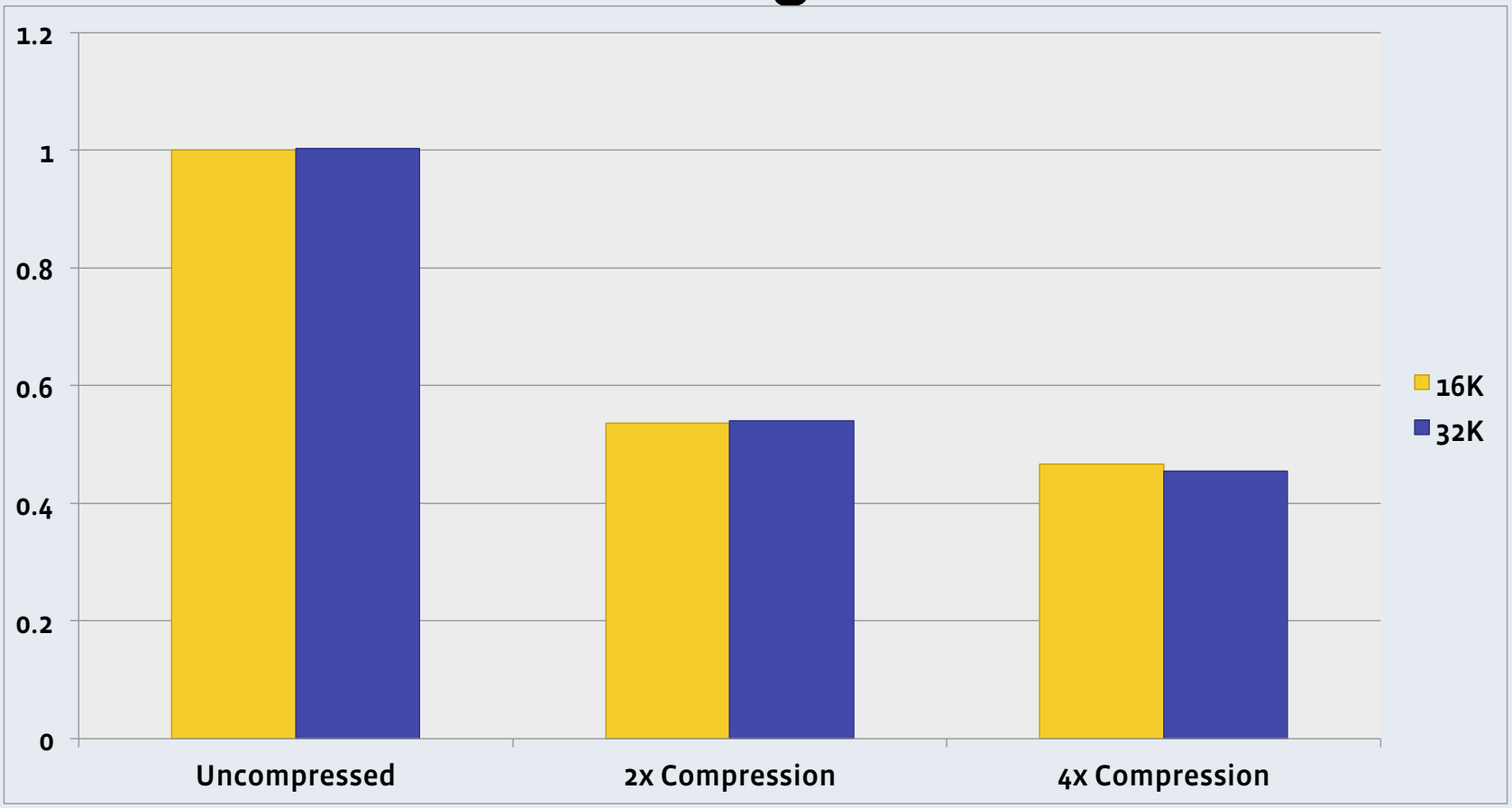
```
#define PAGE_ZIP_DIR_SLOT_MASK  (UNIV_PAGE_SIZE <= UNIV_PAGE_SIZE_DEF ? \
        0x3fff : 0x3fffff)
/** 'owned' flag */
#define PAGE_ZIP_DIR_SLOT_OWNED  (UNIV_PAGE_SIZE <= UNIV_PAGE_SIZE_DEF ? \
        0x4000 : 0x400000)
/** 'deleted' flag */
#define PAGE_ZIP_DIR_SLOT_DEL  (UNIV_PAGE_SIZE <= UNIV_PAGE_SIZE_DEF ? \
        0x8000 : 0x800000)
```

- Advantages:
  - Easy
  - Supports page sizes > 32K
- Disadvantage:
  - Needed only a single bit but used an entire byte (per record)

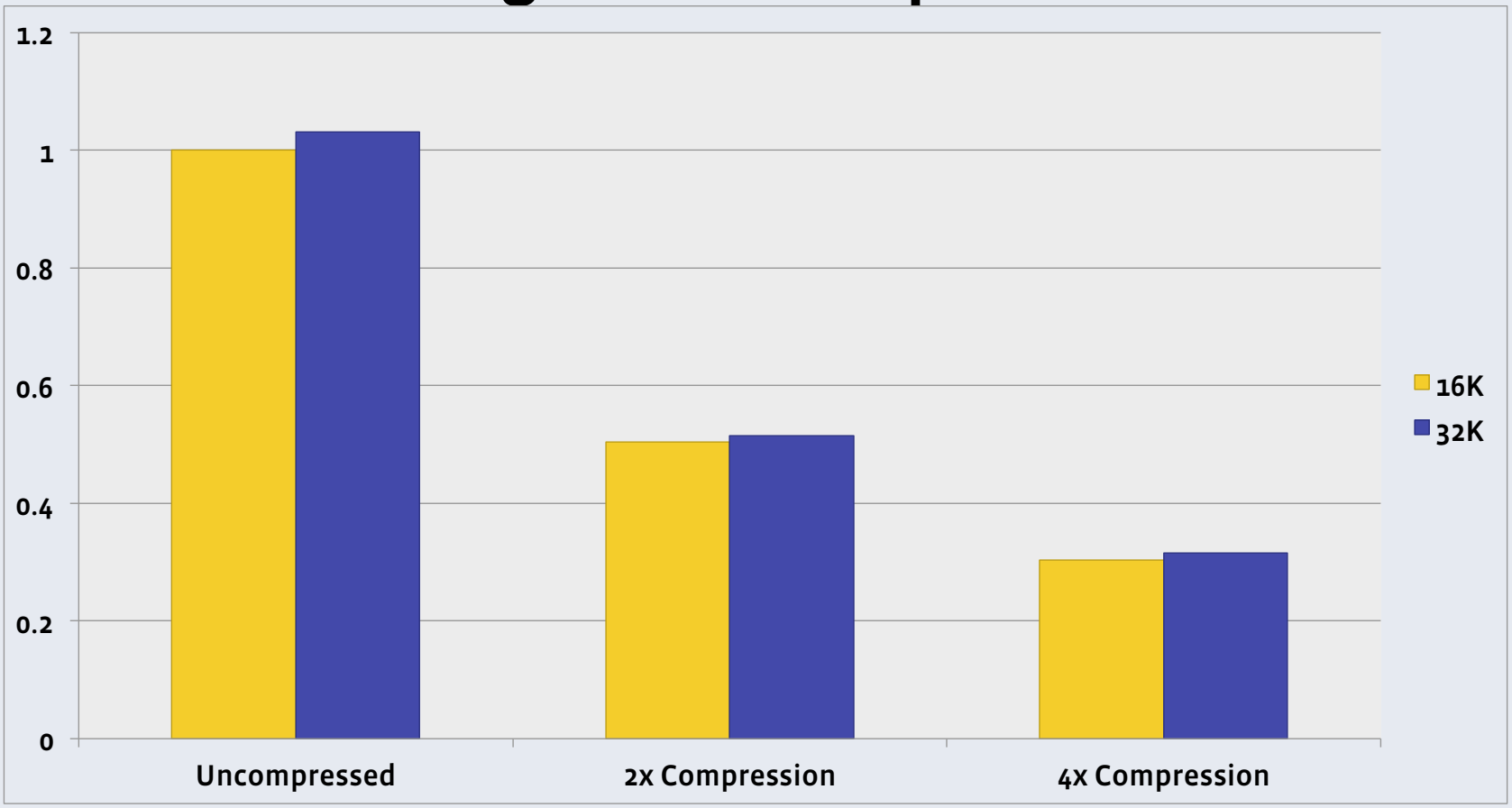
Testing Process

- Load_and_replay_binlog tool
  - Clones production instance
  - Exports with mysqldump
  - Transforms ROW_FORMAT and KEY_BLOCK_SIZE of CREATE TABLEs
  - Imports
  - Replays ~3 months of binary logs
    - Re-fragments tables

Results – Sum of 20 largest tables



One of Our Largest Uncompressed Tables



facebook

KEY_BLOCK_SIZE=4

Poor Server Throughput

- Excellent job by Yoshinori!
- Testing with KEY_BLOCK_SIZE=4
- rows_inserted / sec started ~100 K, dropped to ~500
- Many 8k pages in pages_free:

```
mysql> select pages_free from I_S.innodb_cmpmem;
+-----------+------------+
| page_size | pages_free |
+-----------+------------+
|      1024 |          0 |
|      2048 |          0 |
|      4096 |          1 |
|      8192 |      62170 |
|     16384 |          0 |
```

Poor Server Throughput (cont.)

- GDB shows thread at:

```
buf_buddy_free_low (…) at storage/innobase/buf/buf0buddy.cc:482
```

- Code:

```
for (bpage = UT_LIST_GET_FIRST(buf_pool->zip_free[i]); bpage; ) {
  ...
  bpage = UT_LIST_GET_NEXT(list, bpage);
}
```

- http://bugs.mysql.com/bug.php?id=68077
- Allocation of compressed pages based on a binary buddy system
- On free, buddy allocator will recombine if buddy page is free
- Scan of free list to find buddy was O(n)

Fixes

- Our fix in 5.1.63: replace list with tree using ut/utorbt.c.
- Oracle fixed in 5.6.11: http://bazaar.launchpad.net/~mysql/mysql-server/5.6/revision/4845.2.1.

Which Tables?

- Ran load_and_replay_binlog tool multiple times
  - No compression
  - KEY_BLOCK_SIZE=8
  - KEY_BLOCK_SIZE=4
- Collect statistics:
  - du -b *.ibd
  - SELECT COMPRESS_OPS, COMPRESS_OPS_OK, COMPRESS_USECS FROM I_S.TABLE_STATISTICS;
- Balance between disk space saved vs. CPU cost represented by:
  - Number compression failures
  - Number overall compressions

Results

- Deployed 5.1.63 fix to production
- Compressed best candidate table to 4x across all shards
- Harrison: “It is OUTRAGEOUS at the amount of space saved by the ‹table_name› compression!”

Remaining Work

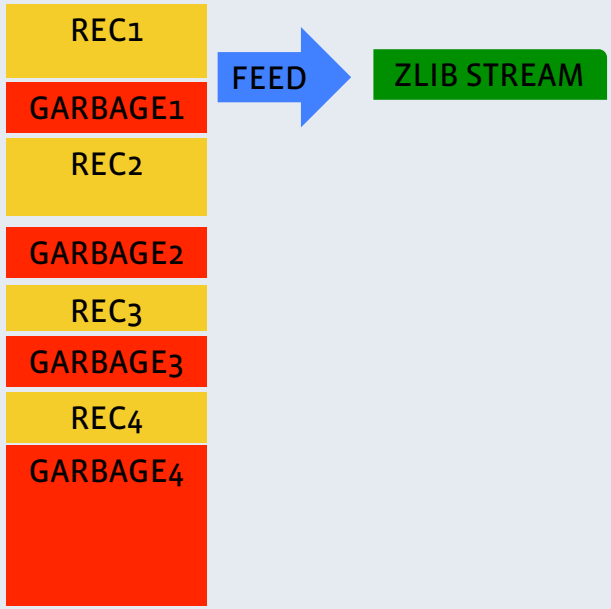
- Number of compression operations higher than we expected
- Nizam tackling that problem and more!

facebook

Rewriting Page Compression

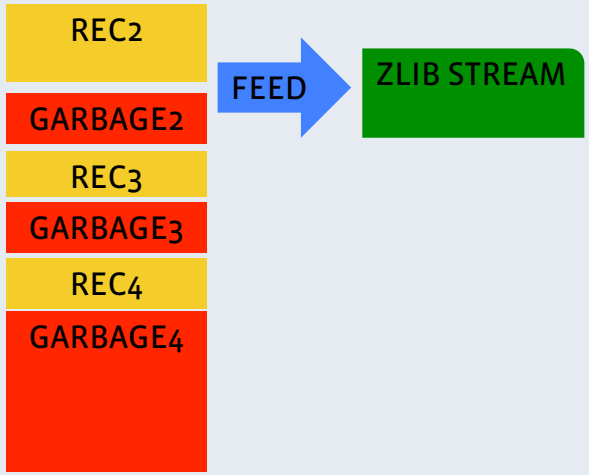
Rewriting Page Compression

- Current implementation feeds records to zlib one by one.
- Prevents use of other compression libraries.
- Garbage is also compressed.



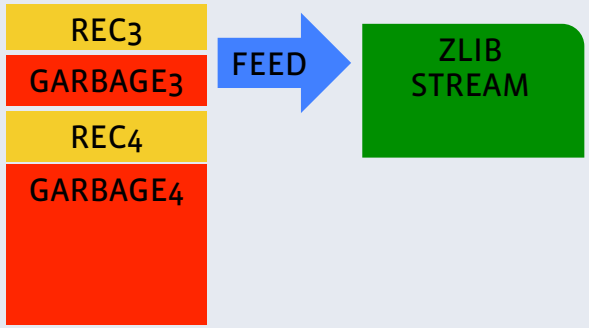
Rewriting Page Compression

- Current implementation feeds records to zlib one by one.
- Prevents use of other compression libraries.
- Garbage is also compressed.



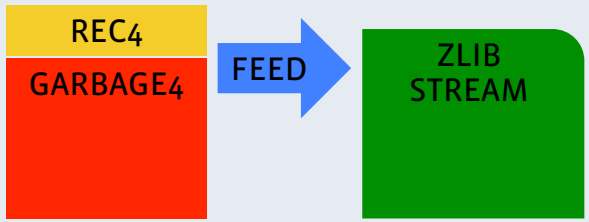
Rewriting Page Compression

- Current implementation feeds records to zlib one by one.
- Prevents use of other compression libraries.
- Garbage is also compressed.



Rewriting Page Compression

- Current implementation feeds records to zlib one by one.
- Prevents use of other compression libraries.
- Garbage is also compressed.

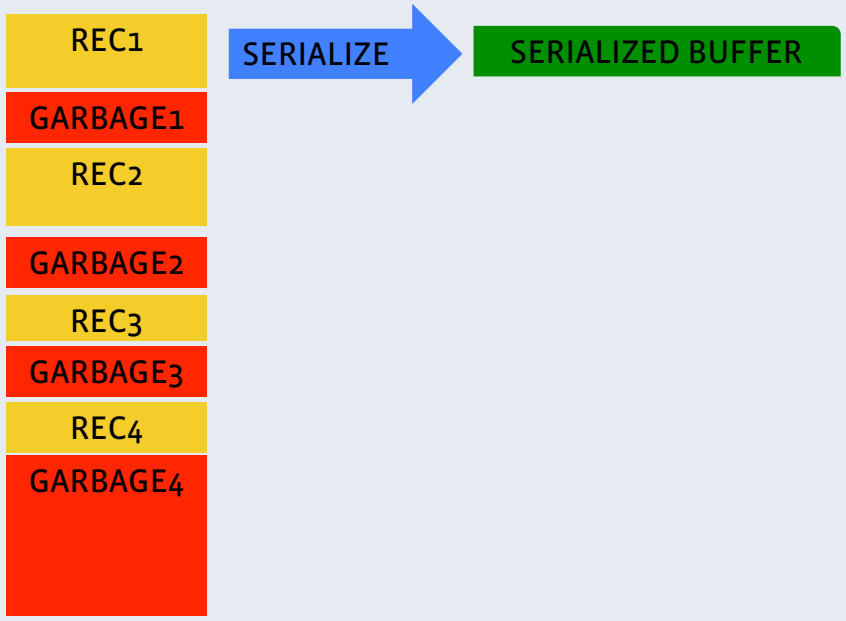


Rewriting Page Compression

- Current implementation feeds records to zlib one by one.
- Prevents use of other compression libraries.
- Garbage is also compressed.

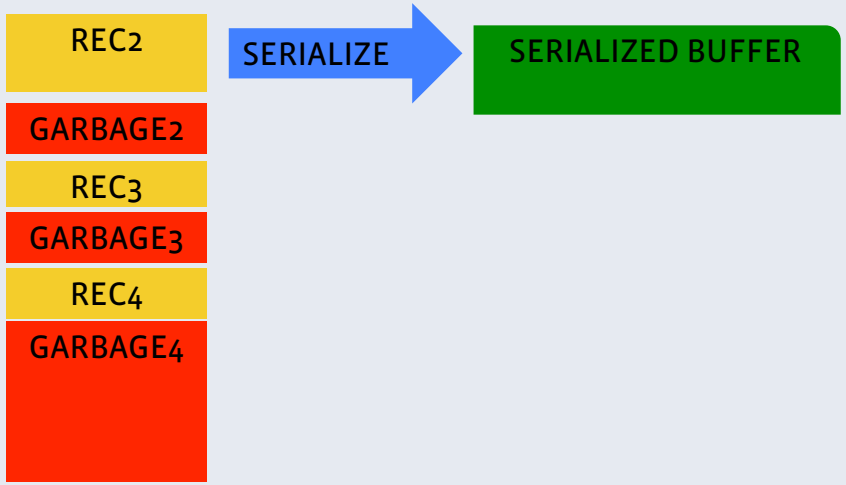
Rewriting Page Compression

- New implementation serializes the records into a buffer before passing data to compression.
- Does not compress garbage.



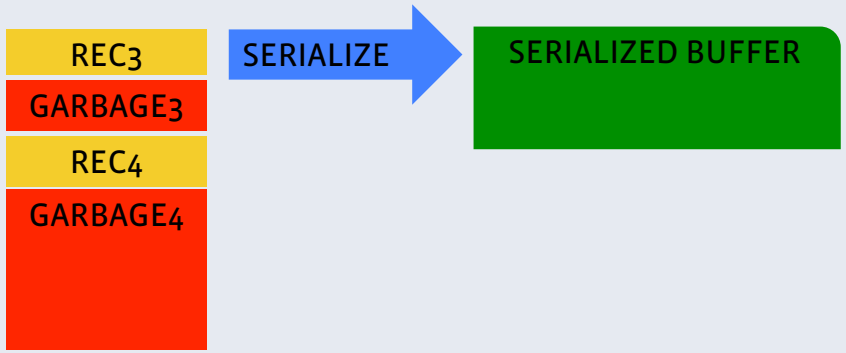
Rewriting Page Compression

- New implementation serializes the records into a buffer before passing data to compression.
- Does not compress garbage.



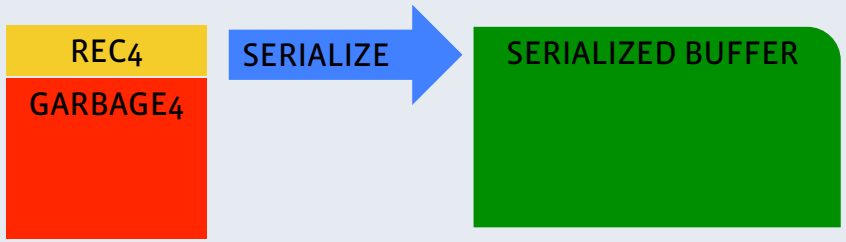
Rewriting Page Compression

- New implementation serializes the records into a buffer before passing data to compression.
- Does not compress garbage.



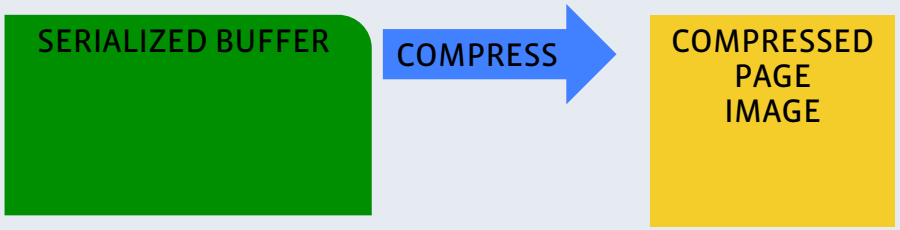
Rewriting Page Compression

- New implementation serializes the records into a buffer before passing data to compression.
- Does not compress garbage.



Rewriting Page Compression

- New implementation serializes the records into a buffer before passing data to compression.
- Does not compress garbage.
- Allows the use of other compression libraries.



Rewriting Page Compression

- We add the following headers to the compressed page:
  - PAGE_ZIP_N_RECS: number of records in the serialization stream.
  - PAGE_ZIP_SERIALIZED_LEN: the total serialized length for the page.
  - PAGE_ZIP_COMPRESSED_LEN: the length of the compressed page image.
- The values stored in these headers are used for monitoring purposes and page-based padding.

facebook

Compact Storage For Metadata

Compact Storage For Metadata

- Older versions of the records are stored in the undo log.
- Each version of the record contains the following metadata:
  - Last transaction id that modified the record (6 bytes).
  - A pointer to the previous version in the undo log (7 bytes).
- 13 bytes per record.
- Metadata is stored uncompressed even for compressed tables.

Compact Storage For Metadata

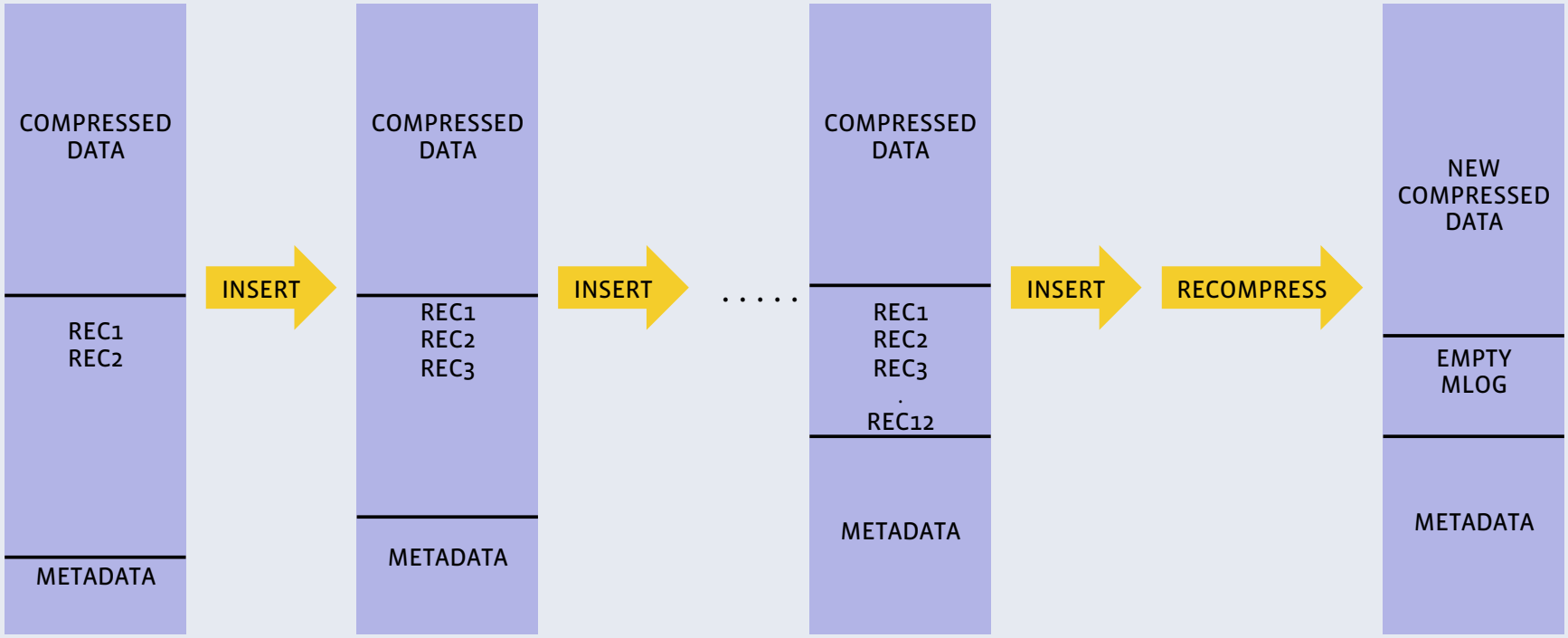
- The metadata for a record is needed as long as there are transactions that can see the older versions of the record.
- We modified the metadata storage so that only the metadata that may be used are stored.

Compact Storage For Metadata

- We keep the minimum transaction id that can still be undone in a global variable trx_id_undoable.
- trx_id_undoable is updated when the undo log is purged.
- Whenever a compressed page needs more space, we try reducing the space needed for metadata.
- We get rid of the metadata entries for which trx_id < trx_id_undoable.

Compact Storage For Metadata

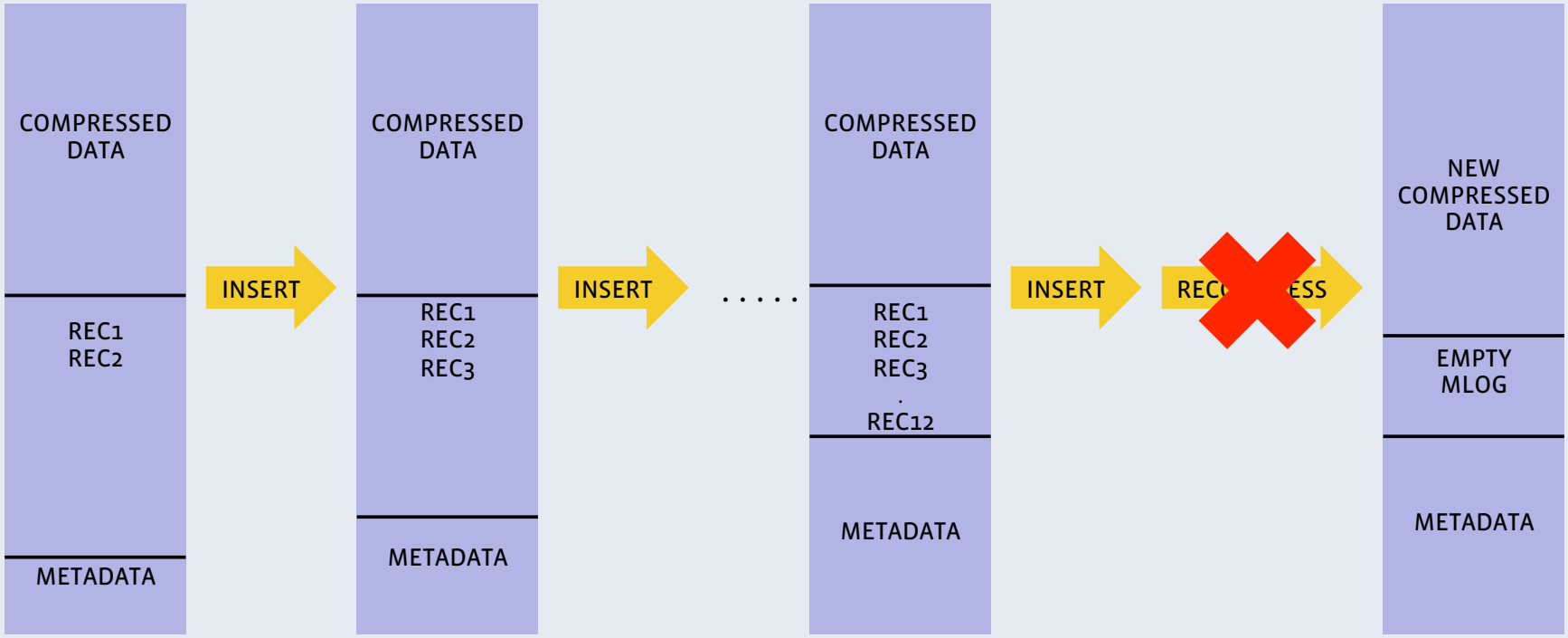
- Earlier:



Compressed Page

Compact Storage For Metadata

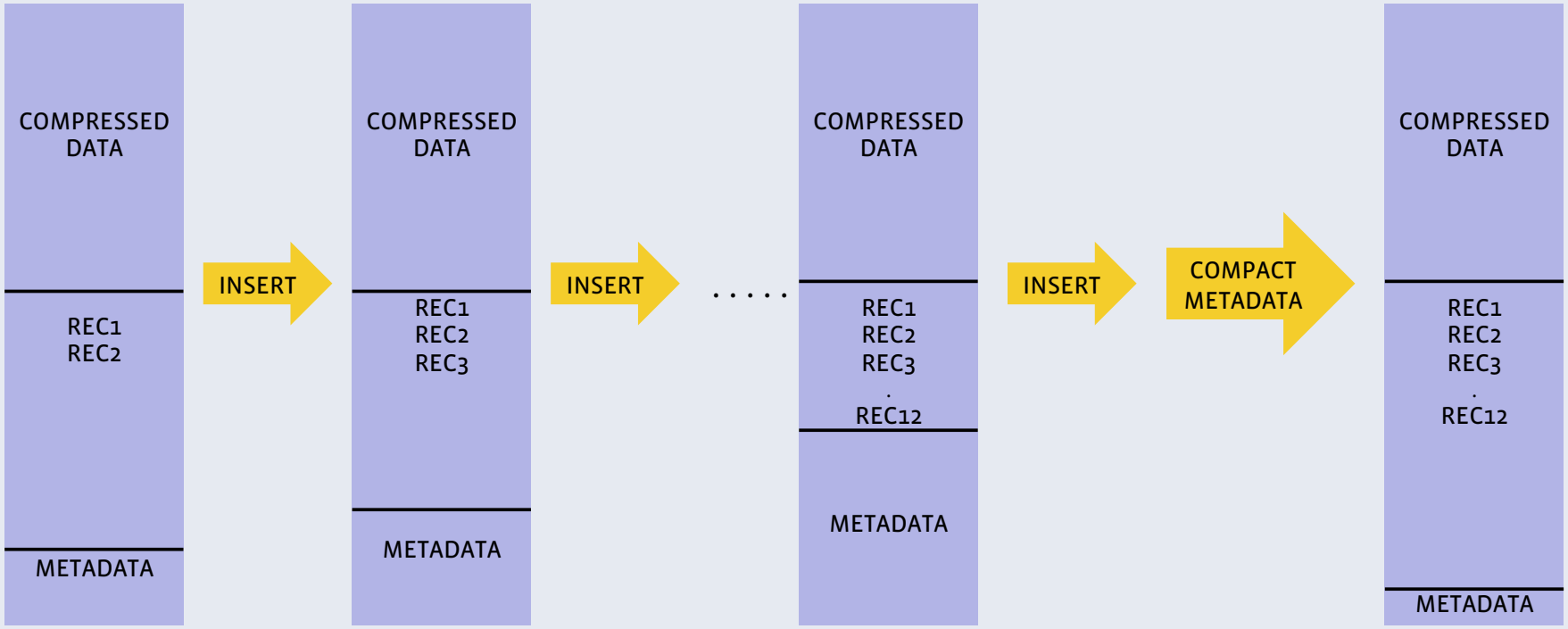
- Earlier:



Compressed Page

Compact Storage For Metadata

- Now:



Compressed Page

facebook

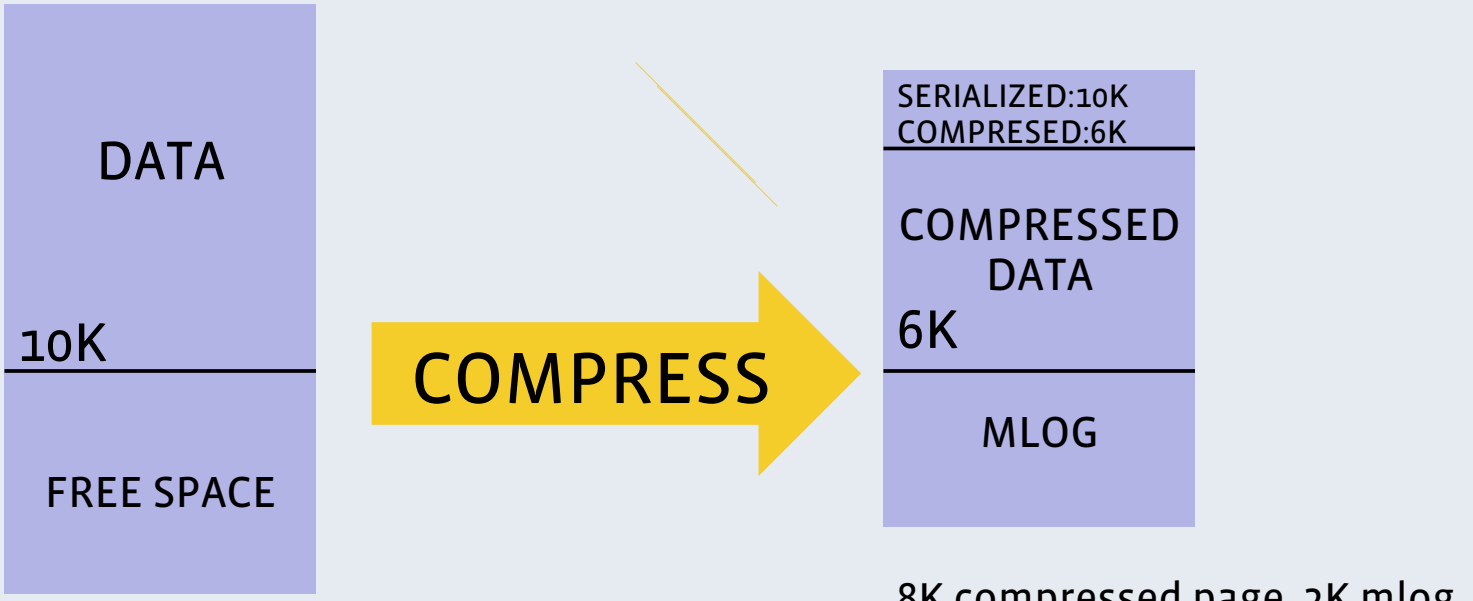
Page Based Padding

Page Based Padding

- Adaptive padding works great if the compressibility of the pages are similar.
- Some of the very incompressible pages may increase the padding value unnecessarily.
- Instead of having a padding value per table, we decide when to split a page based on the compressibility of the page.

Page Based Padding

- The compressibility of the page is determined by the values stored in compressed page headers.

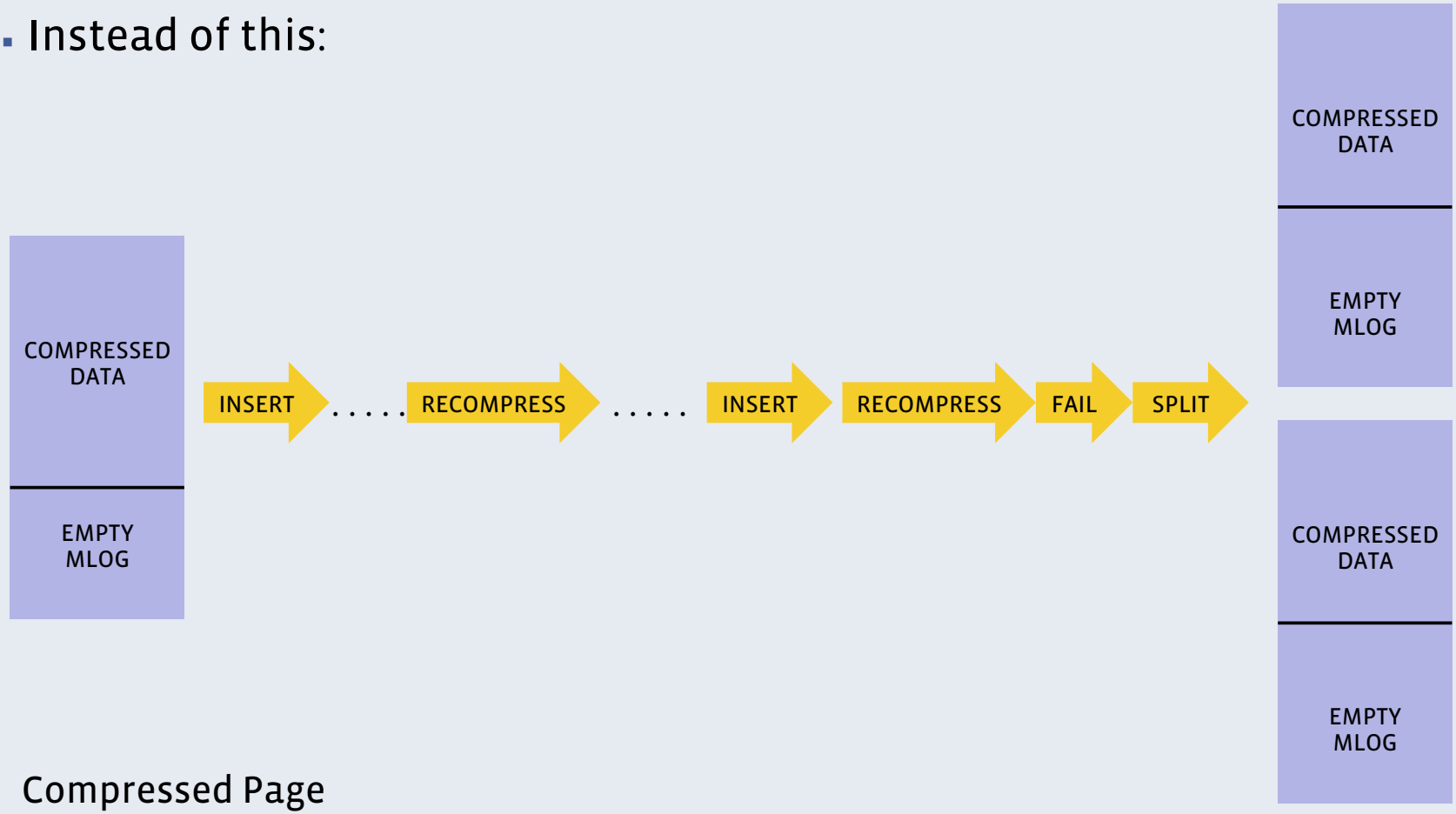


16K uncompressed page

8K compressed page, 2K mlog, compressibility: .6

Page Based Padding

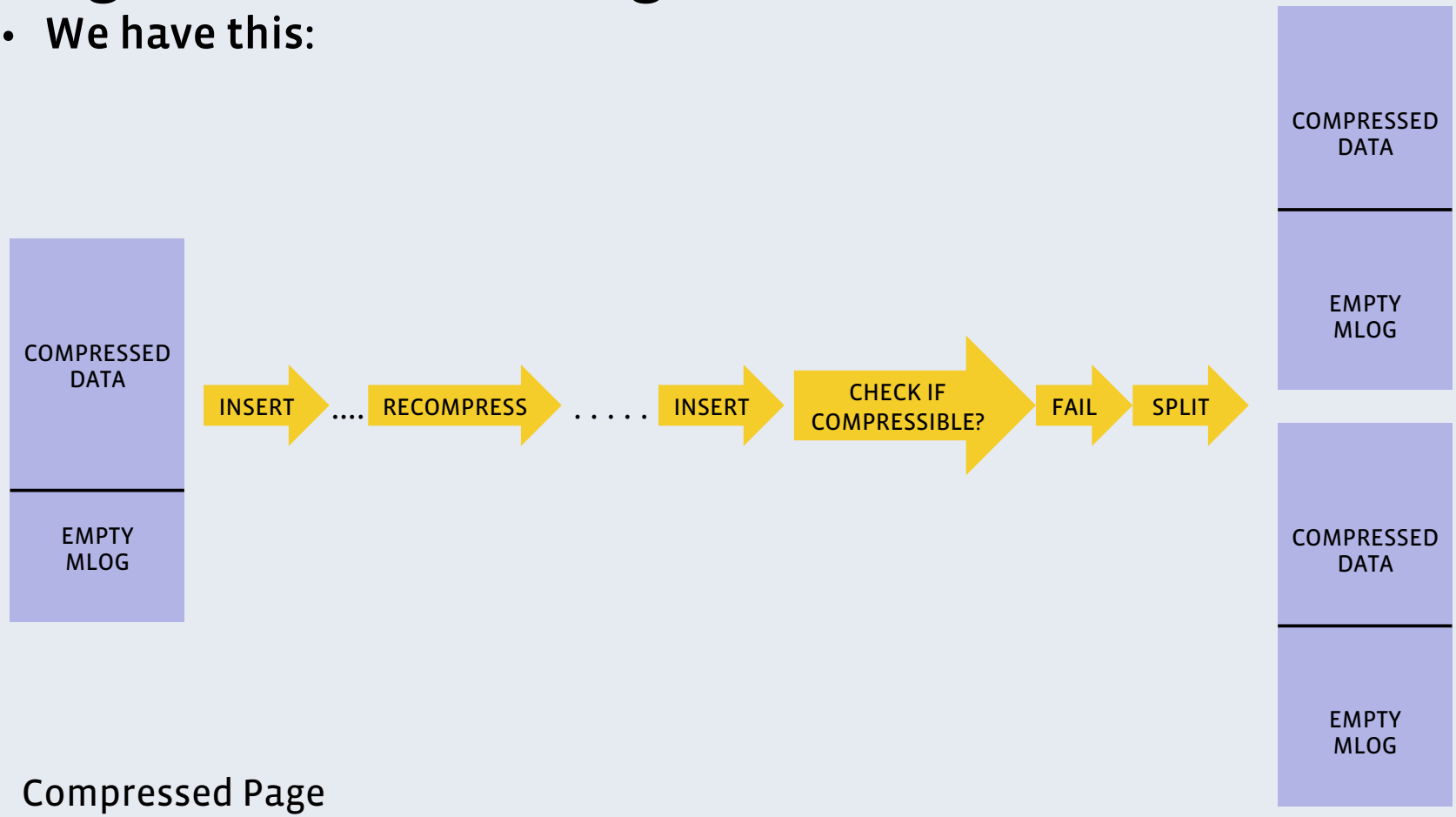
- Instead of this:



Compressed Page

Page Based Padding

- We have this:



Compressed Page

Page Based Padding

- Eliminates almost all of the compression failures for most of the tables.
- Less risk of wasting space in comparison to adaptive padding.

facebook

Efficient Use of Modification Log

Efficient Use of Modification Log

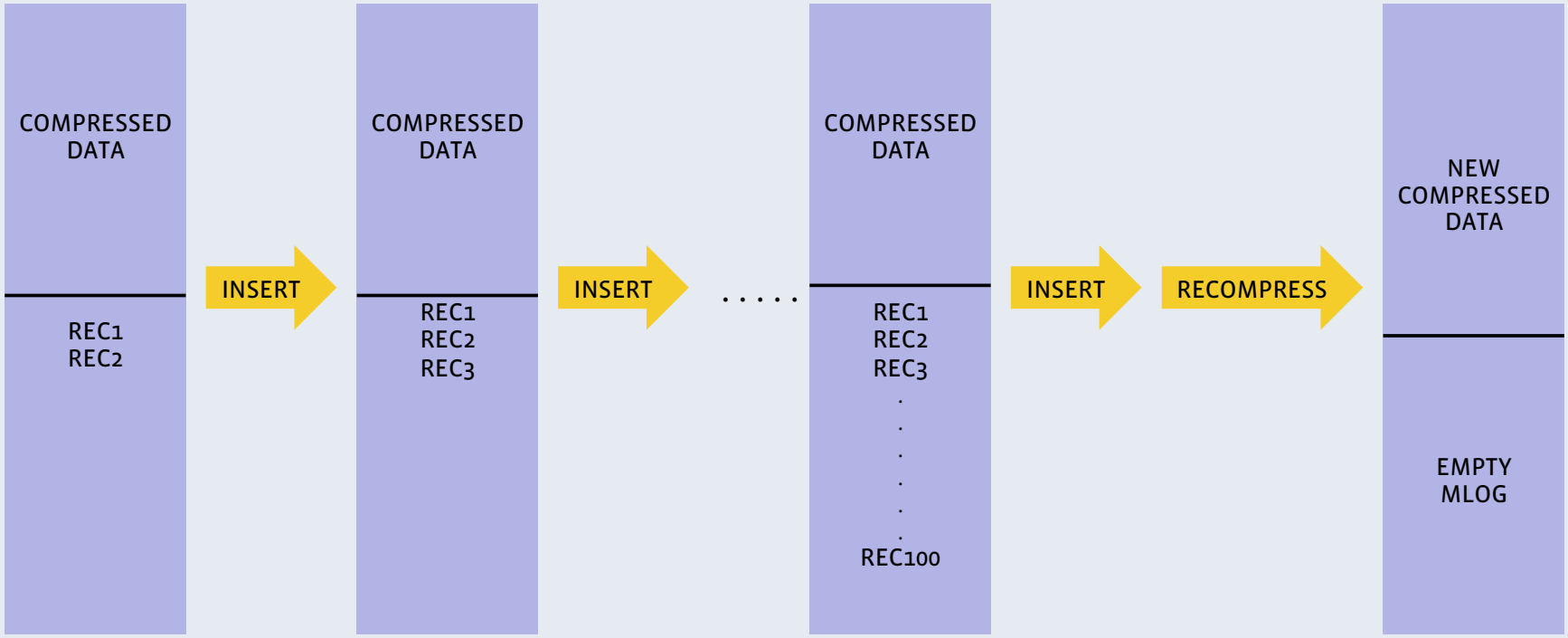
- Page compressions are expensive and modification log reduces the number of page compressions.
- With KEY_BLOCK_SIZE=4, we have very small space for modification log.
- Performance suffers because of the increase in the number of recompressions.

Efficient Use of Modification Log

- We compress the modification log when it becomes full.
- This makes room for new records in modification log.
- Cheaper than compressing the entire page.

Efficient Use of Modification Log

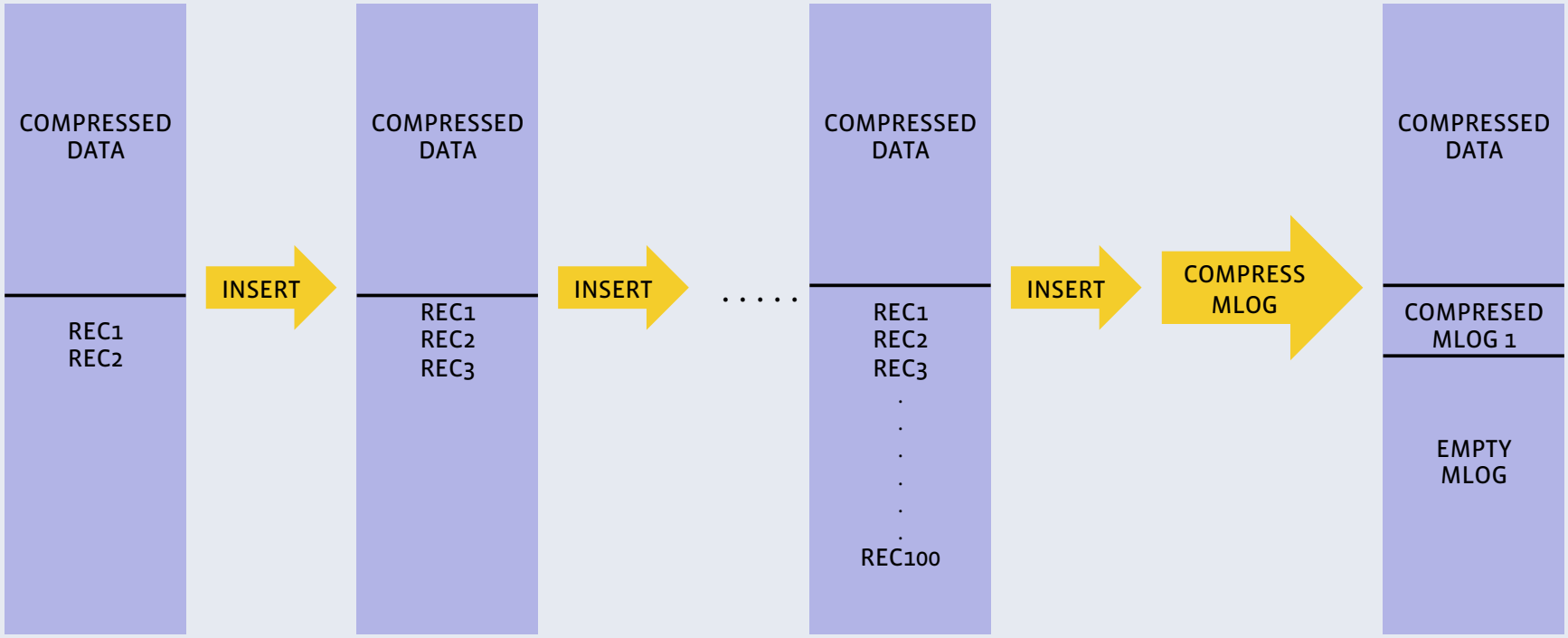
- Earlier:



Compressed Page

Efficient Use of Modification Log

- Now:



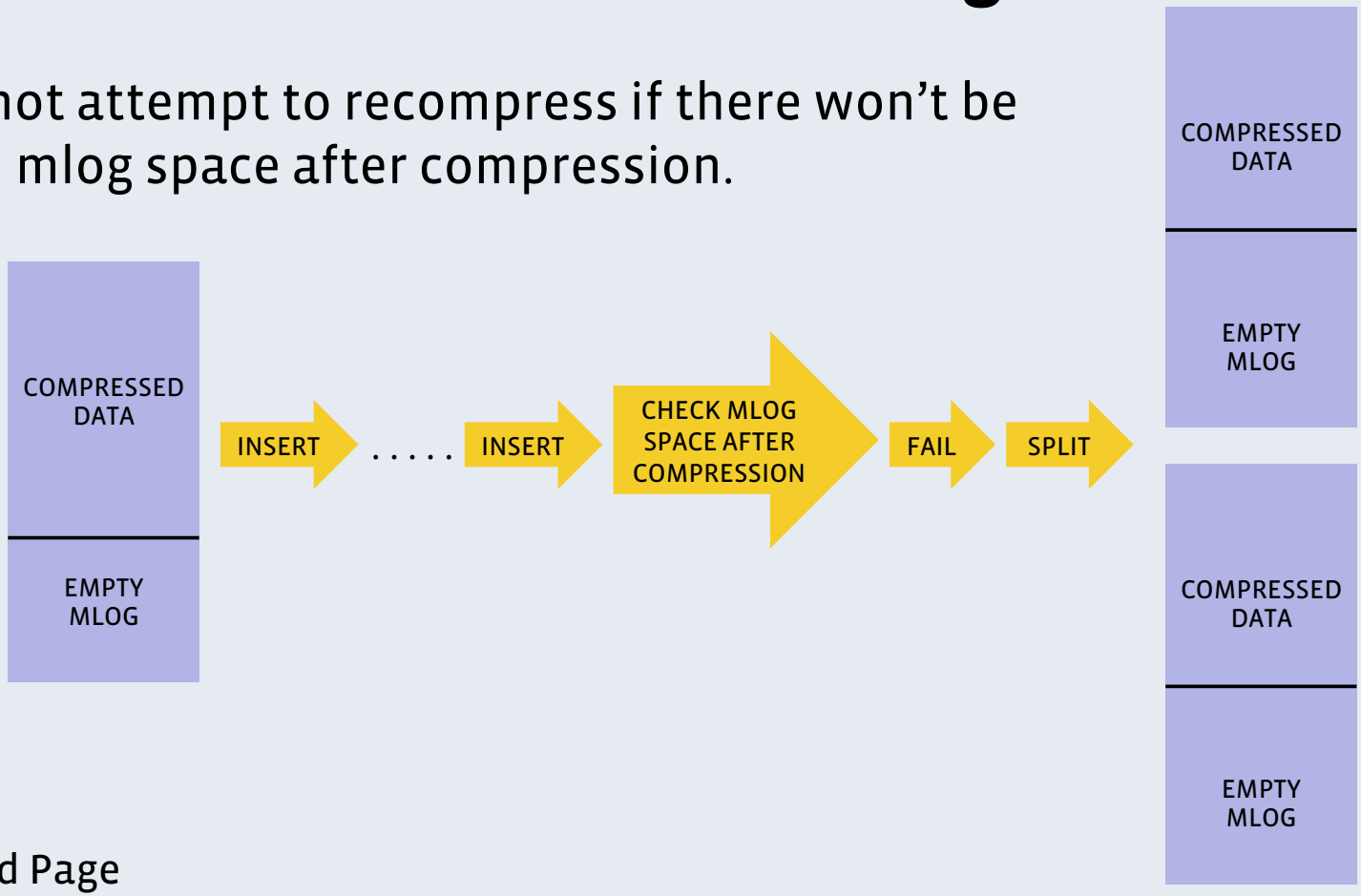
Compressed Page

Efficient Use of Modification Log

- When pages are compressed but they have very little mlog space, a recompression will happen soon.
- To prevent this we require a minimum amount of mlog on every compressed page.

Efficient Use of Modification Log

- We do not attempt to recompress if there won't be enough mlog space after compression.



Compressed Page

facebook

Preliminary Results

Preliminary Results

- Achieved 3x compression using KEY_BLOCK_SIZE=4 with zlib level=1.
- Performance was similar to using KEY_BLOCK_SIZE=8 with current compression.
- LZMA gave more space savings but was prohibitive in terms of CPU.

Questions

https://github.com/facebook/mysql-5.6