InnoDB:
A journey to the core III

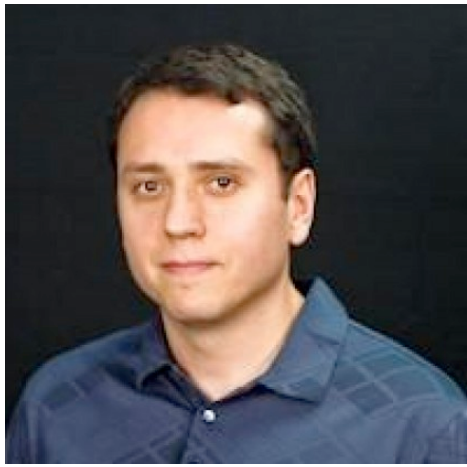
Jeremy Cole and Davi Arnaut

bit.ly/innodb_journey_iii

Jeremy Cole
@jeremycole

Making MySQL Awesome at Google
Worked at MySQL 2000-2004
Contributor since 3.23
16 years in the MySQL community
Code, documentation, research, bug reports
Yahoo!, Proven Scaling, Gazillion, Twitter

Davi Arnaut
@davi



MySQL Internals development at LinkedIn
Worked at MySQL 2007-2011
Designed and built Twitter and LinkedIn MySQL
Long time Open Source contributor: Apache, Linux kernel, etc.

About this work...

Not intended to be comprehensive
Not authoritative (it is based on research)
One of the best sources of documentation for InnoDB formats

Approach:

1. Read the C and C++ sources
2. Implement in Ruby
3. Refactor and correct until reasonable
4. Document!

Resources

blog.jcole.us/innodb/

github.com/jeremycole/innodb_ruby

groups.google.com/forum/#!forum/innodb_ruby

innodb_ruby@googlegroups.com

Overview of InnoDB Compression

The high-level idea

Each page is compressed using zlib to save space
Usage of compression is transparent to the user
Save disk space and IO size/throughput at the cost of CPU, memory usage, and complexity

Enabling compression

Use of innodb_file_per_table is required
Page sizes are 1, 2, 4, 8, or 16 KiB
Page size is the same for all pages in the space

```
CREATE TABLE ( ... )
ENGINE=InnoDB
ROW_FORMAT=COMPRESSED
KEY_BLOCK_SIZE=N
```

The implementation

Index usage in memory is only on uncompressed (16 KiB) pages
When a page is modified, both the compressed version of the page and the uncompressed version are modified
Only the compressed page is flushed to disk
The uncompressed page is discarded when no longer needed
The buffer pool can contain both the compressed and uncompressed version of each page at the same time
Uncompressed pages are evicted from the buffer pool first, compressed pages in the buffer pool act as a cache

The modification log

As modifications to a page are made, they are appended to a modification log inside the page

The modification log grows to consume the free space in a page

The page must be re-organized (re-compressed), or split when:

- The free space in the page is exhausted; or
- The page no longer compresses to the target page size

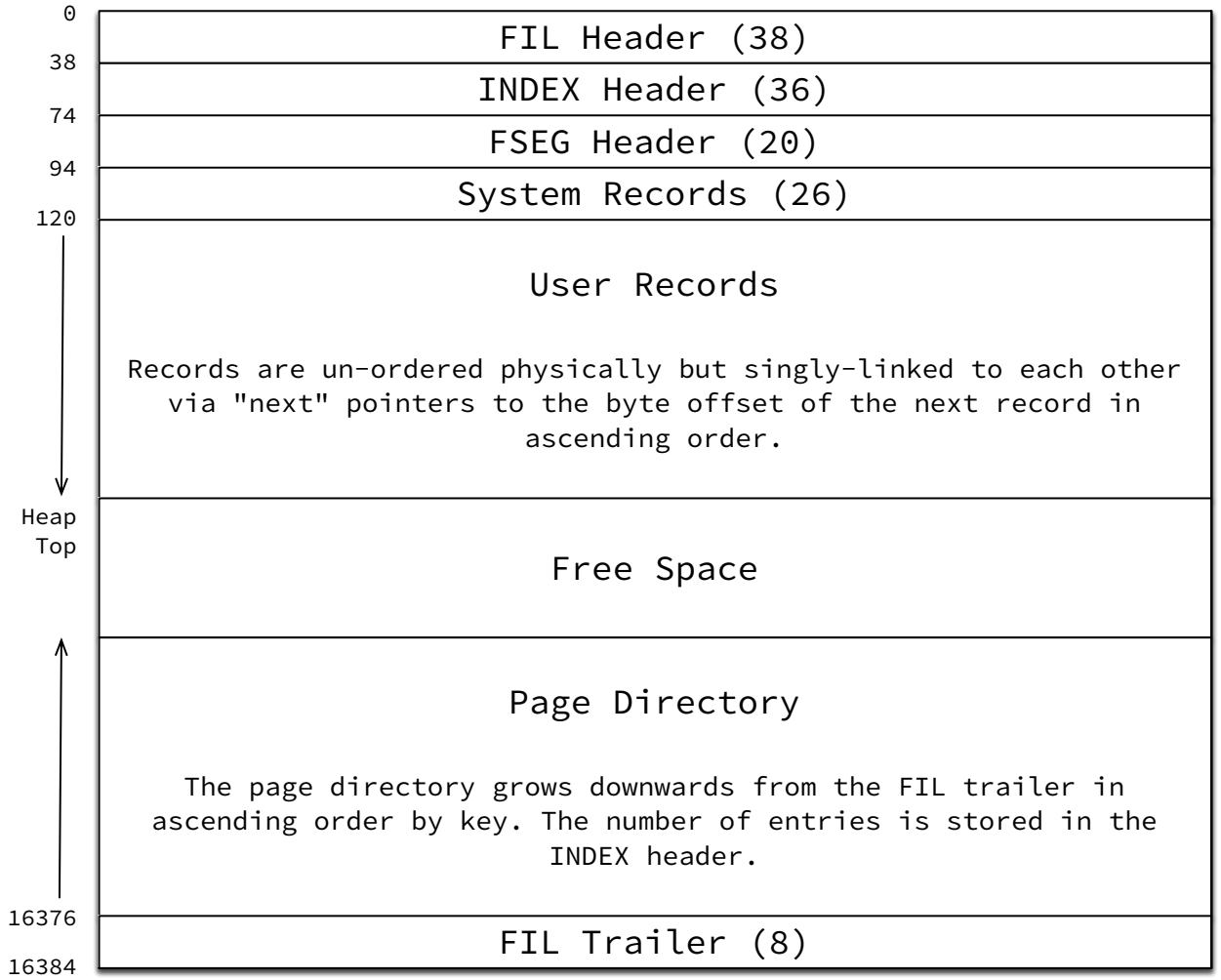
Pages initially start empty, with no compressed data; the modification log grows until full when the page is re-organized and that data is moved to the compressed data section

The buffer pool buddy system

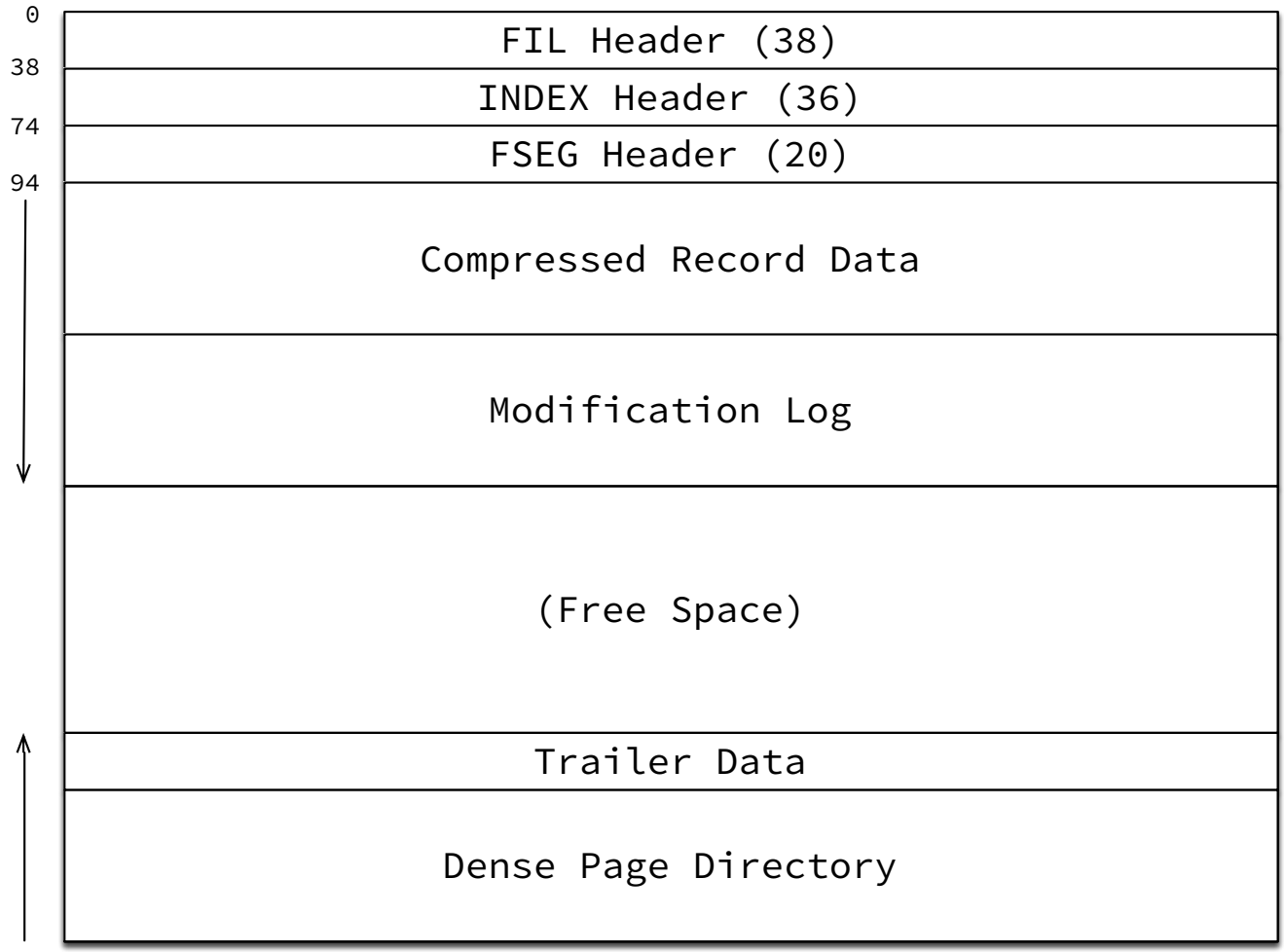
Pages in the buffer pool are always 16 KiB (`UNIV_PAGE_SIZE`)
Pages can be borrowed into the “buddy system” where they can be sub-divided into smaller (8, 4, 2, or 1 KiB) blocks
Management of buffer pool free space becomes a lot more complicated when multiple pages sizes are in use
For example, in the worst case, almost the entire buffer pool can be buddied as 8 KiB pages, of which half are free, but no two halves of a single 16 KiB page are free, so a page must be evicted to free a 16 KiB contiguous block

Physical Structures

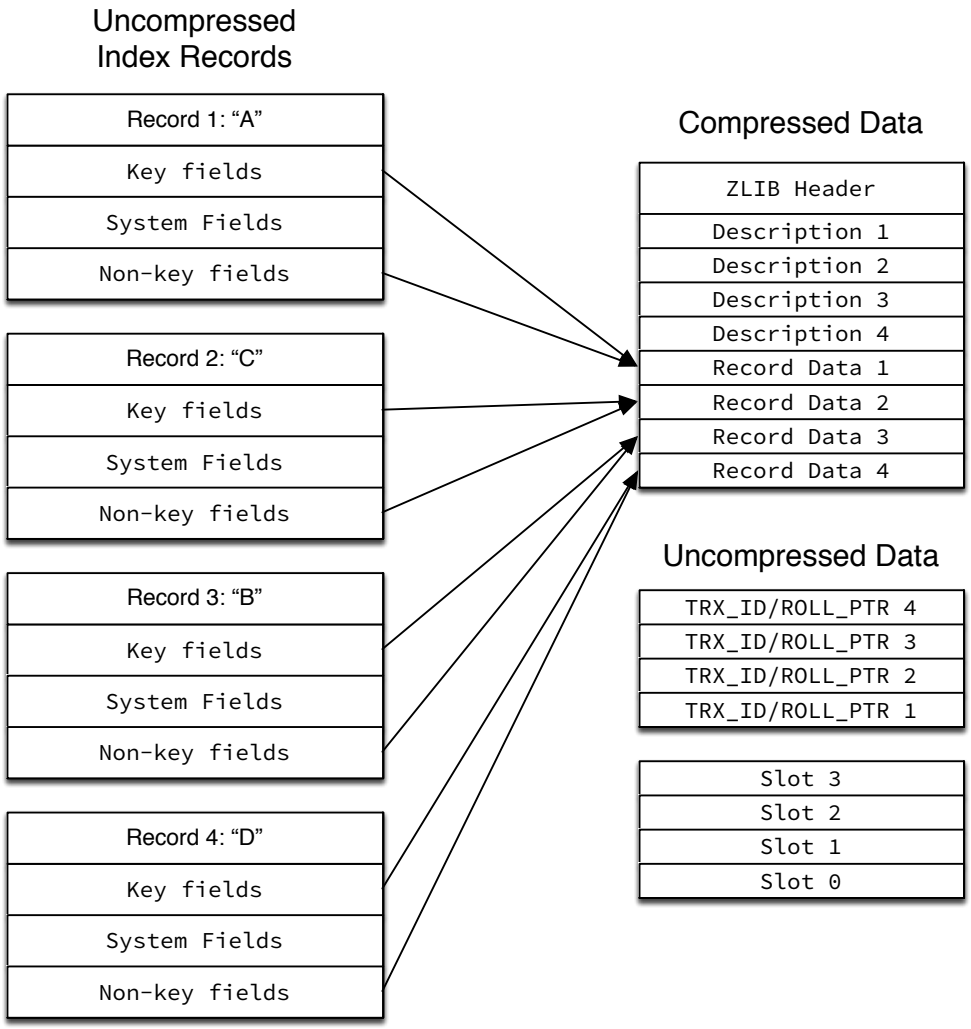
INDEX Overview



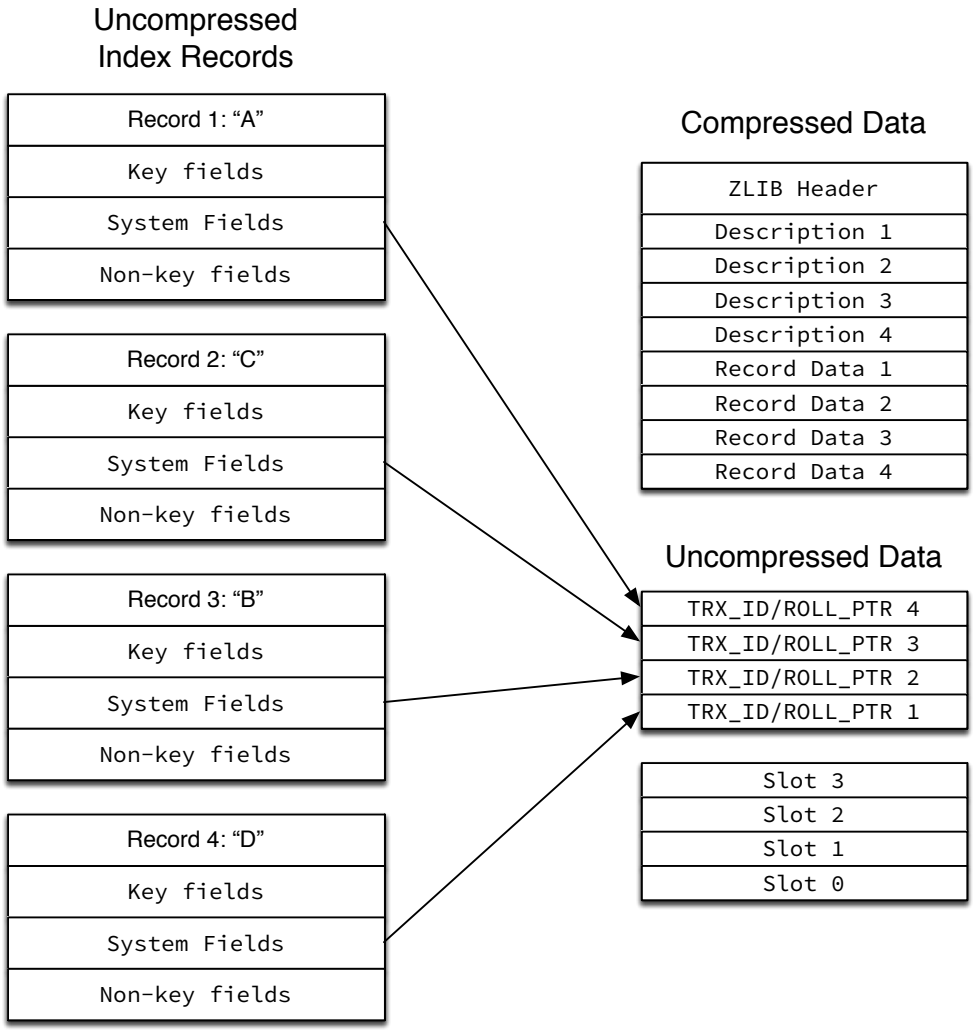
INDEX Overview



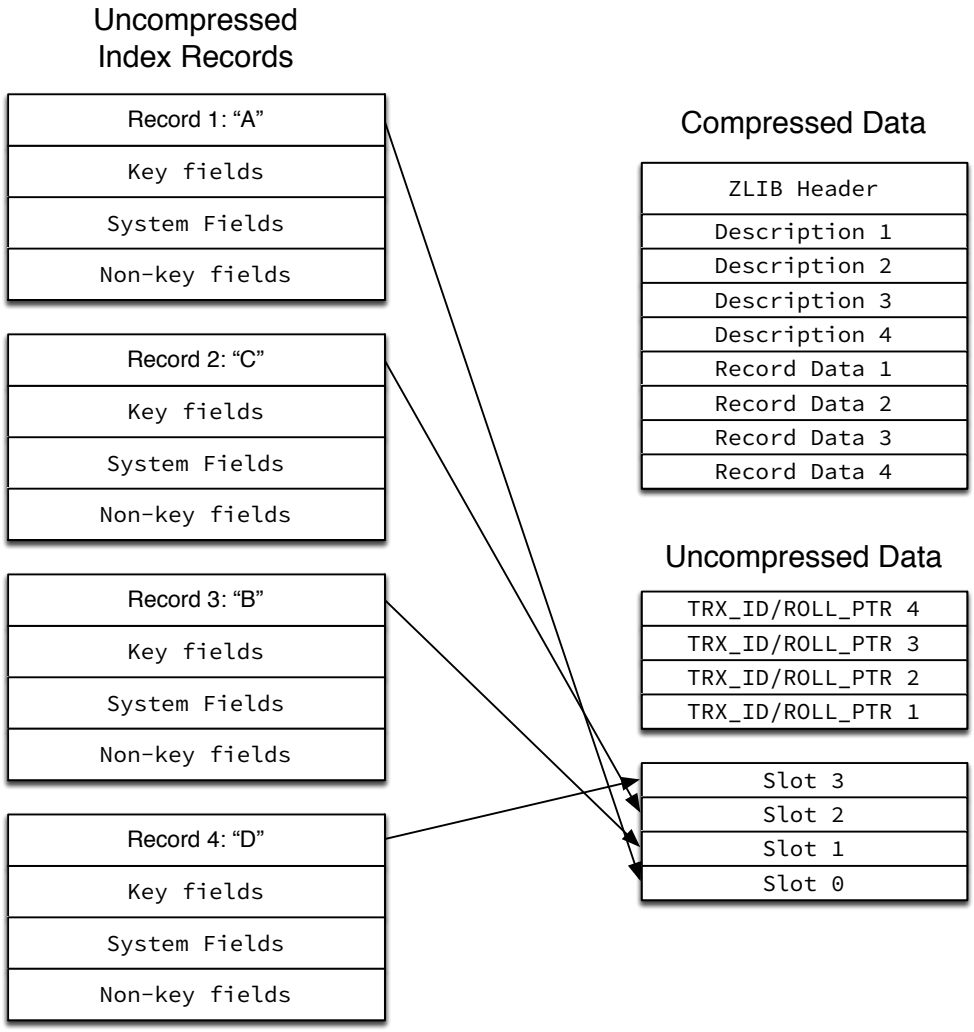
Compressed Index Data



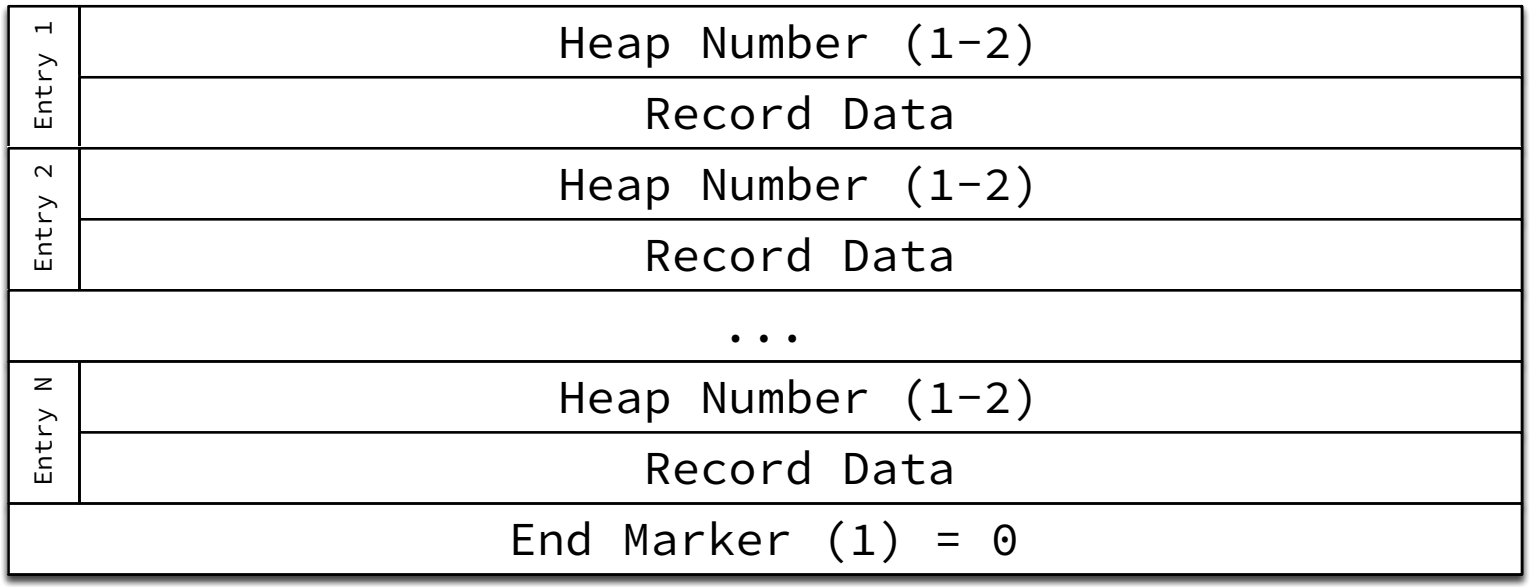
Compressed Index Data



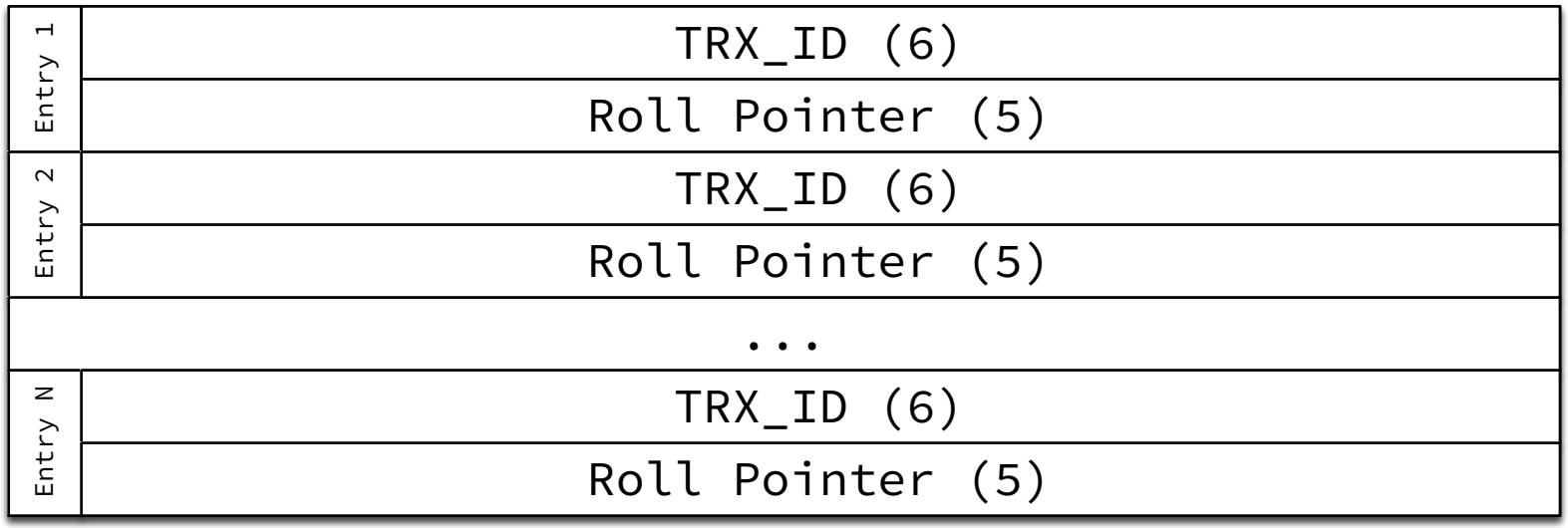
Compressed Index Data



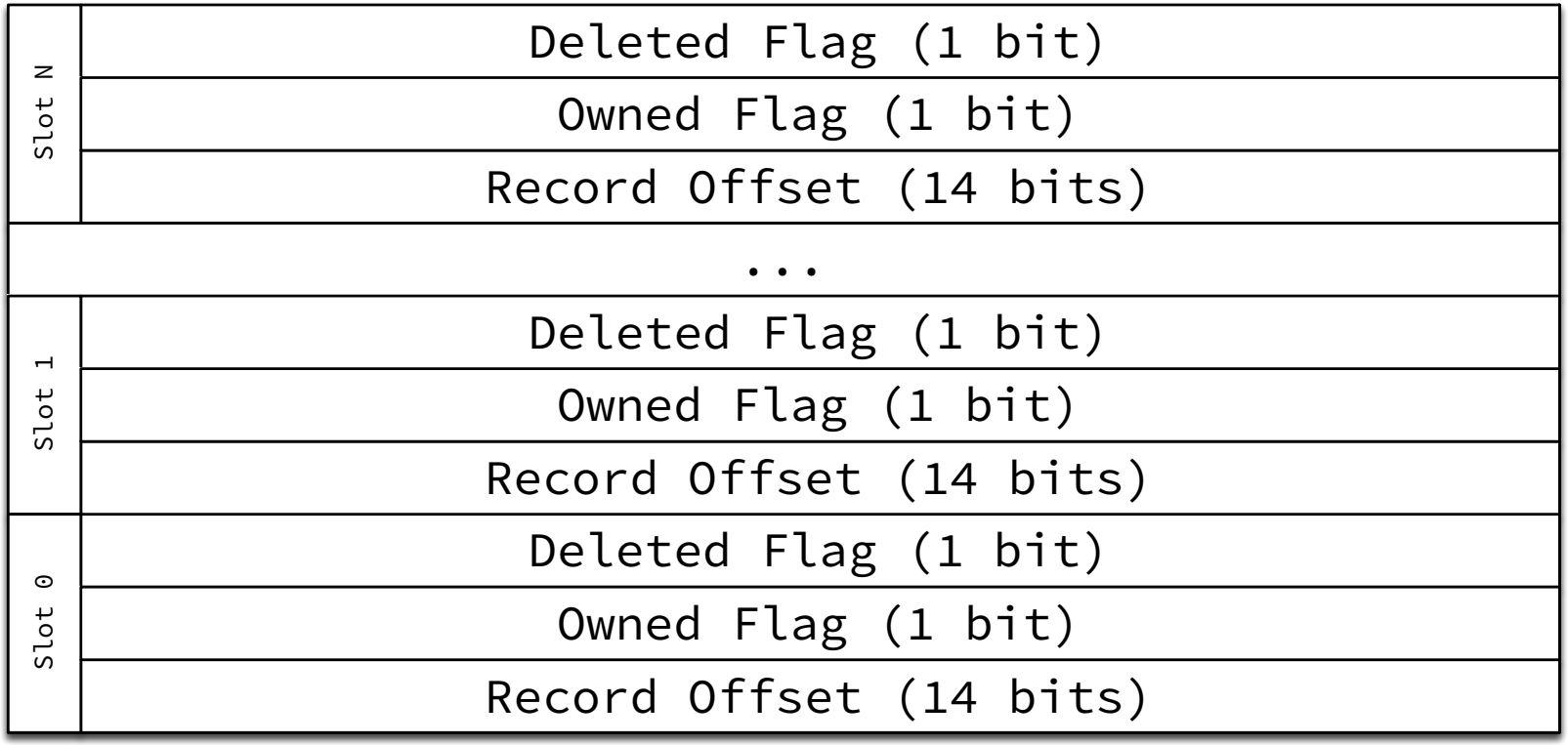
Modification Log



Trailer Data



Dense Page Directory



The dense page directory contains one entry per records, in the key's collation order.
All directory slots will own a minimum of 4 and maximum of 8 records.
The page directory grows "downwards" from the end of the page.

Problems with Compression

Many bugs, problems, inefficiencies

Facebook has worked for the past several years to fix

Probably should use at least MySQL 5.6 to get these fixes, older versions are not good

Quirks with the specific version of zlib used and upgrading; if the zlib version changes, compressed data may change size, so currently in some cases the entire compressed page is redo logged for safety

Compression is overly complicated :)

Lots of “if (page_zip)“ everywhere, lots of “_zip” variations of functions — compression support leaks all over the code base

It is difficult to monitor and to tell what it’s doing and why

Bad abstraction, tied to zlib implementation/design

Disk space waste with smaller page sizes

MySQL Bug #67963: “InnoDB wastes almost one extent out of every innodb_page_size pages”

Very little waste (0.37%) with regular uncompressed 16 KiB pages, but up to 24.8% waste in worst case (innodb_page_size=4k, key_block_size=1).

Typical waste 0.75% with default 16 KiB page size and key_block_size=8.


Start Page
0
256
512
768
1024
1280
1536
1792
2048
2304
2560
2816
3072
3328
3584
3840
4096
4352
4608
4864
5120
5376
5632
5888
6144
6400
6656
6912
7168
7424
7680
7936
8192

Onwards

Facebook's
InnoDB Compression Posts

facebook.com/MySQLatFacebook/notes

Facebook's "InnoDB Defragmentation"

Rongrong Zhong

Today @ 15:50 in Ballroom A

Q & A