InnoDB: Space Management

Mijin An
meeeeejin@gmail.com

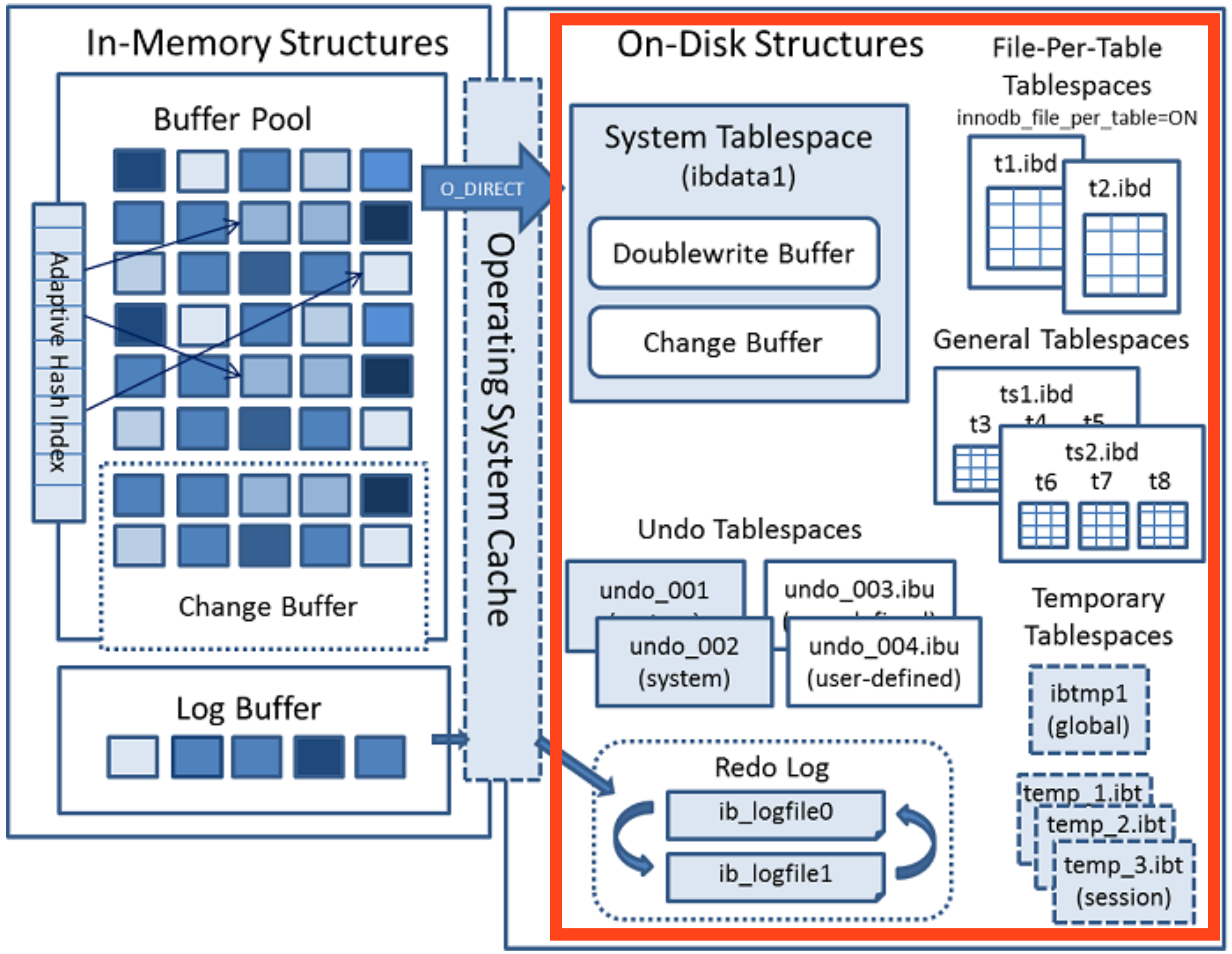
Five Types of Tablespaces

- The storage area for the doublewrite buffer and the change buffer (`ibdata1`)
  - **System Tablespace**

- The storage area for the user defined table and index data (`.ibd`)
  - **File-Per-Table Tablespace**
  - **General Tablespace**

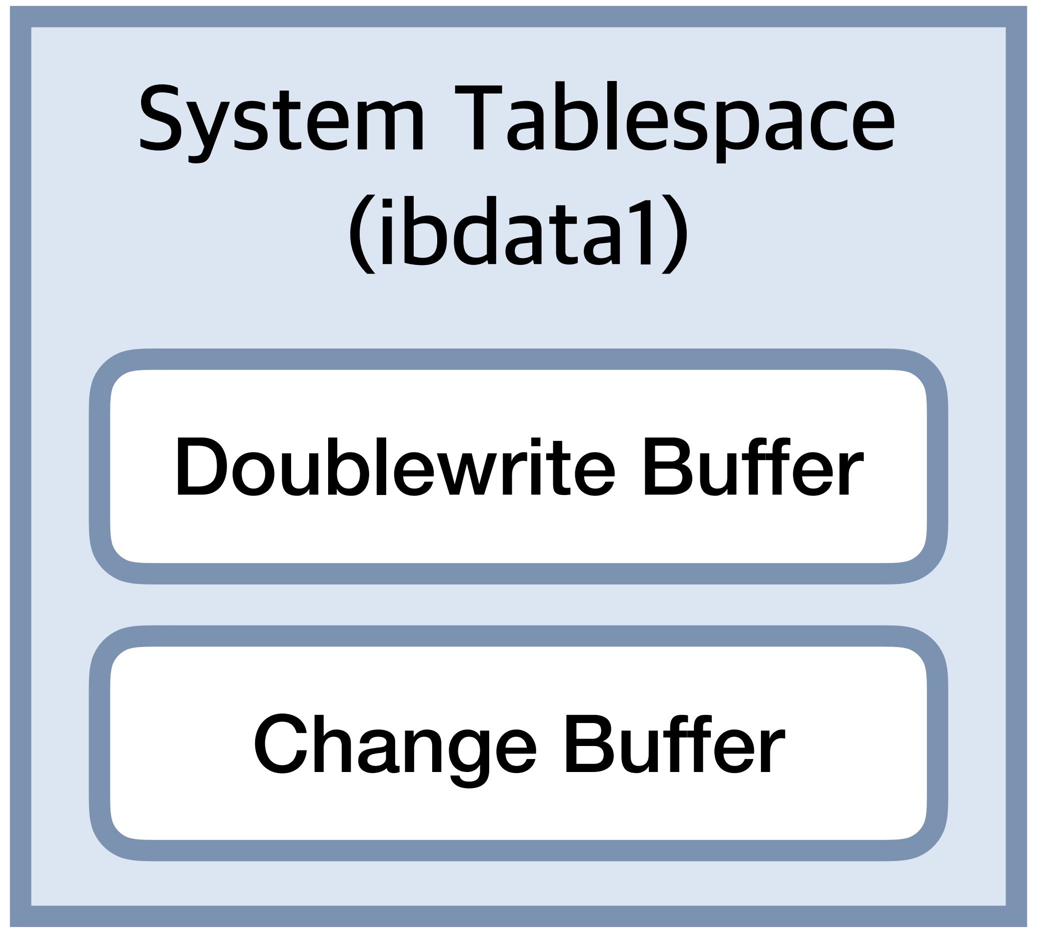
- The storage area for undo logs (`undo_001`)
  - **Undo Tablespace**

- The storage area for user/optimizer-created temporary tables and rollback segments for them (`.ibt`, `ibtmp1`)
  - **Temporary Tablespace**



System Tablespace (`ibdata1`)

- `innodb_file_per_table = OFF`
- The storage area for the **doublewrite buffer** and the **change buffer**
- The files are **logically concatenated** and there is no striping in use
  - You cannot define where within the system tablespace your tables are allocated
- Truncating or dropping a table only frees space and it can be **reused** for InnoDB data (i.e., not returned to the OS, as system tablespace data files never shrink)

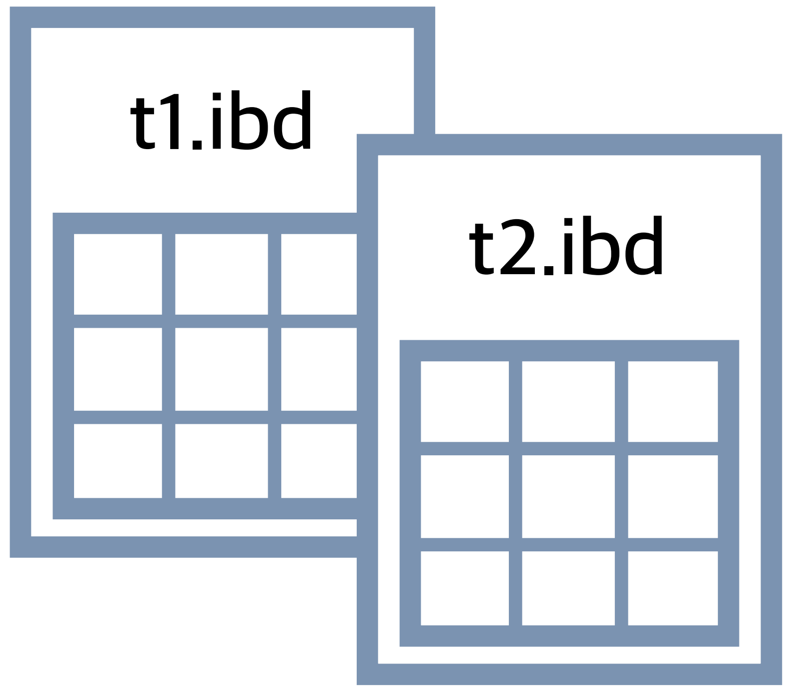


File-Per-Table Tablespace (`.ibd`)

- `innodb_file_per_table = ON` (default)
- Each newly created table is stored in a **separate** tablespace file with a extension `.ibd`
  - There is less fragmentation within the disk file
  - **The data of a single table and its indexes reside in one `.ibd` file**
- When a table is truncated or dropped, the freed space is **returned** to the OS

- **This slides will focus on the space management of file-per-tablespace**

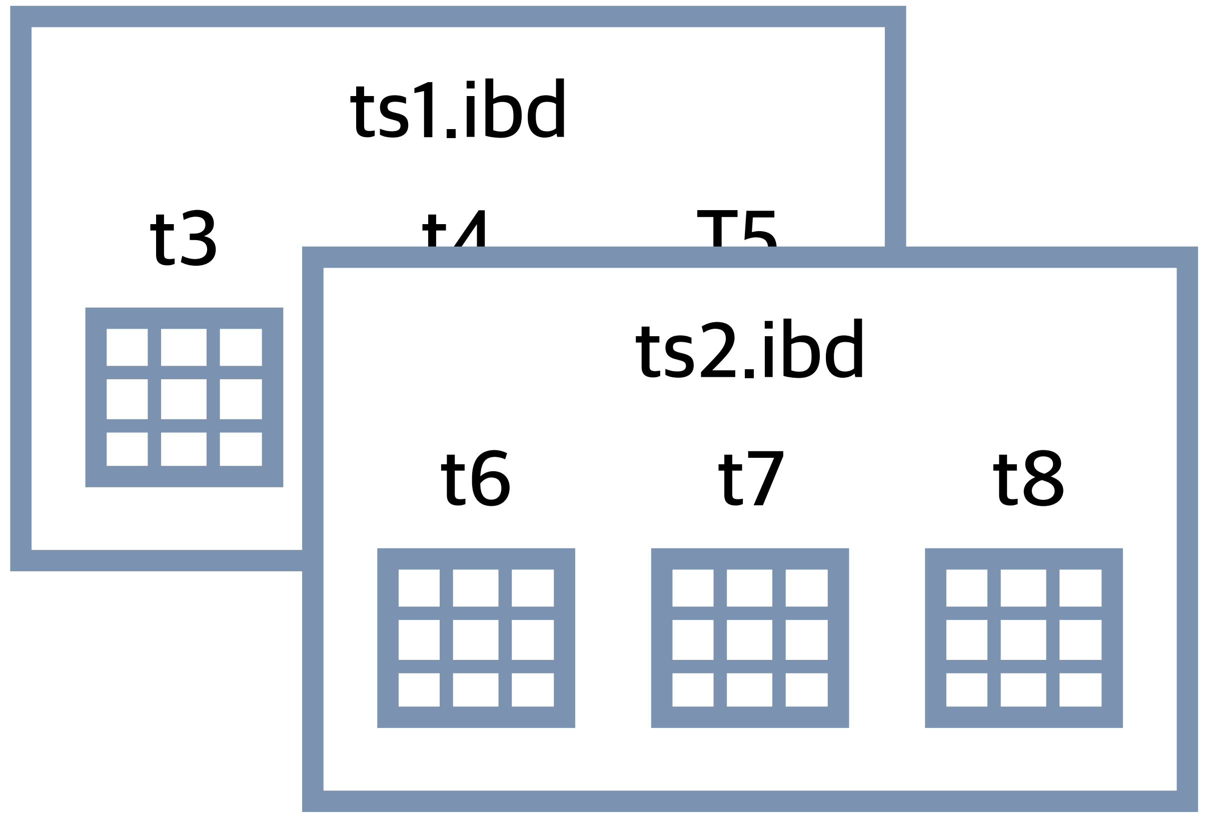
`innodb_file_per_table=ON`



General Tablespace (.ibd)

- **Shared** tablespaces created using `CREATE TABLESPACE` syntax
  - **The data from different tables and their indexes reside in one `.ibd` file**
- They can be created outside of the MySQL data directory
  - Capable of holding multiple tables
  - Support tables of all row formats

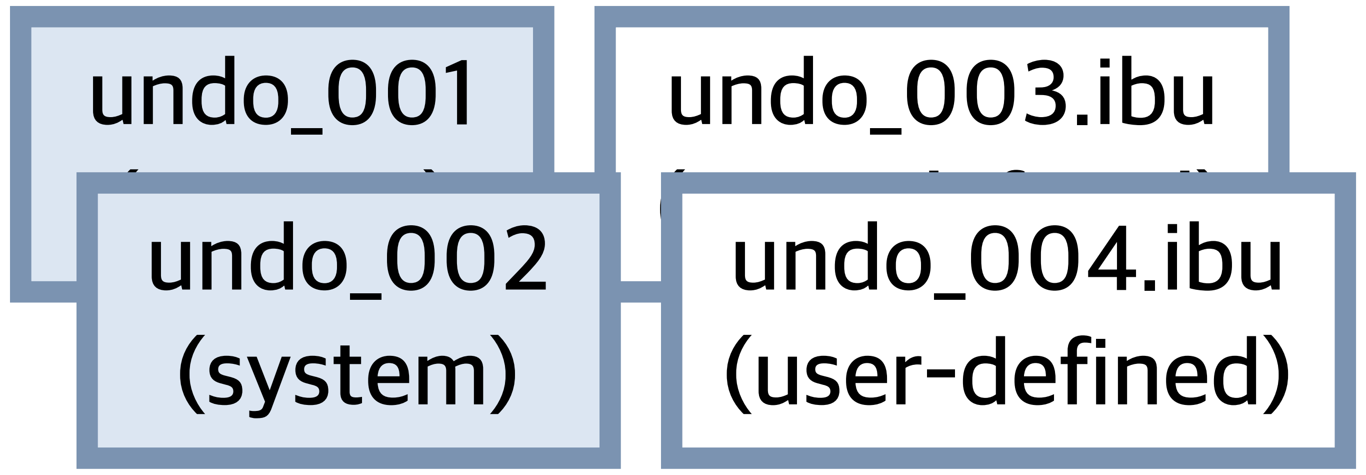
```
CREATE TABLESPACE tablespace_name
      [ADD DATAFILE 'file_name']
      [FILE_BLOCK_SIZE = value]
         [ENGINE [=] engine_name]
```



Undo Tablespace (`undo_00x`)

- **Two** default undo tablespaces are created when the MySQL instance is initialized
- Since MySQL 8.0.14, additional undo table spaces can be created at runtime
- The initial size of an undo tablespace data file depends on the `innodb_page_size` value

Page Size	Undo Tablespace File Size
4 KB	7 MiB
8 KB	8 MiB
16 KB	**10 MiB**
32 KB	20 MiB
64 KB	40 MiB

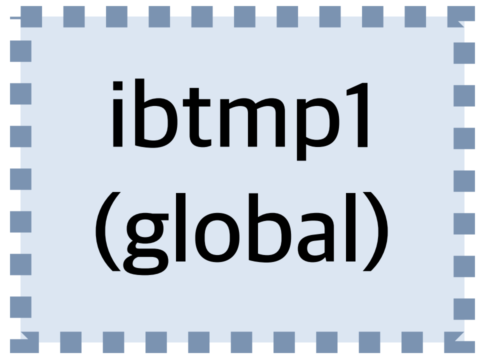


Temporary Tablespace (`.ibt, ibtmp1`)

- **Session Temporary Tablespace (`#innodb_temp/temp_x.ibt`)**
  - Store user-created temporary tables and internal temporary tables created by the optimizer
  - A pool of **10** temporary tablespaces (**5 pages in size**) is created when the server is started
  - When a session disconnects, its temporary table spaces are truncated and released back to the pool

temp_1.ibt
temp_2.ibt
temp_3.ibt
(session)

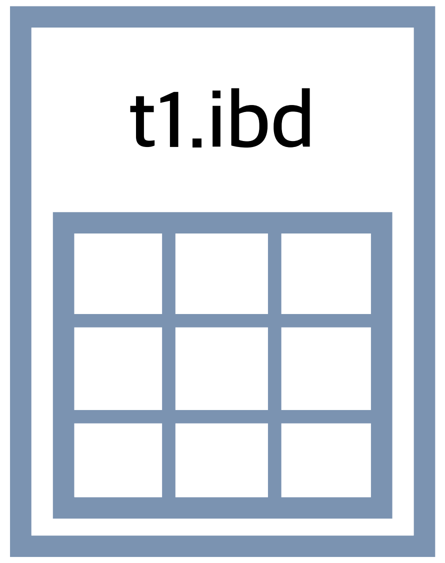
- **Global Temporary Tablespace (`ibtmp1`)**
  - Store rollback segments for changes made to user-created temporary tables
  - The initial file size is slightly larger than **12MB**
  - Removed on normal shutdown and recreated each time the server is started



.ibd Files

- Example: Create a table `test.t1`

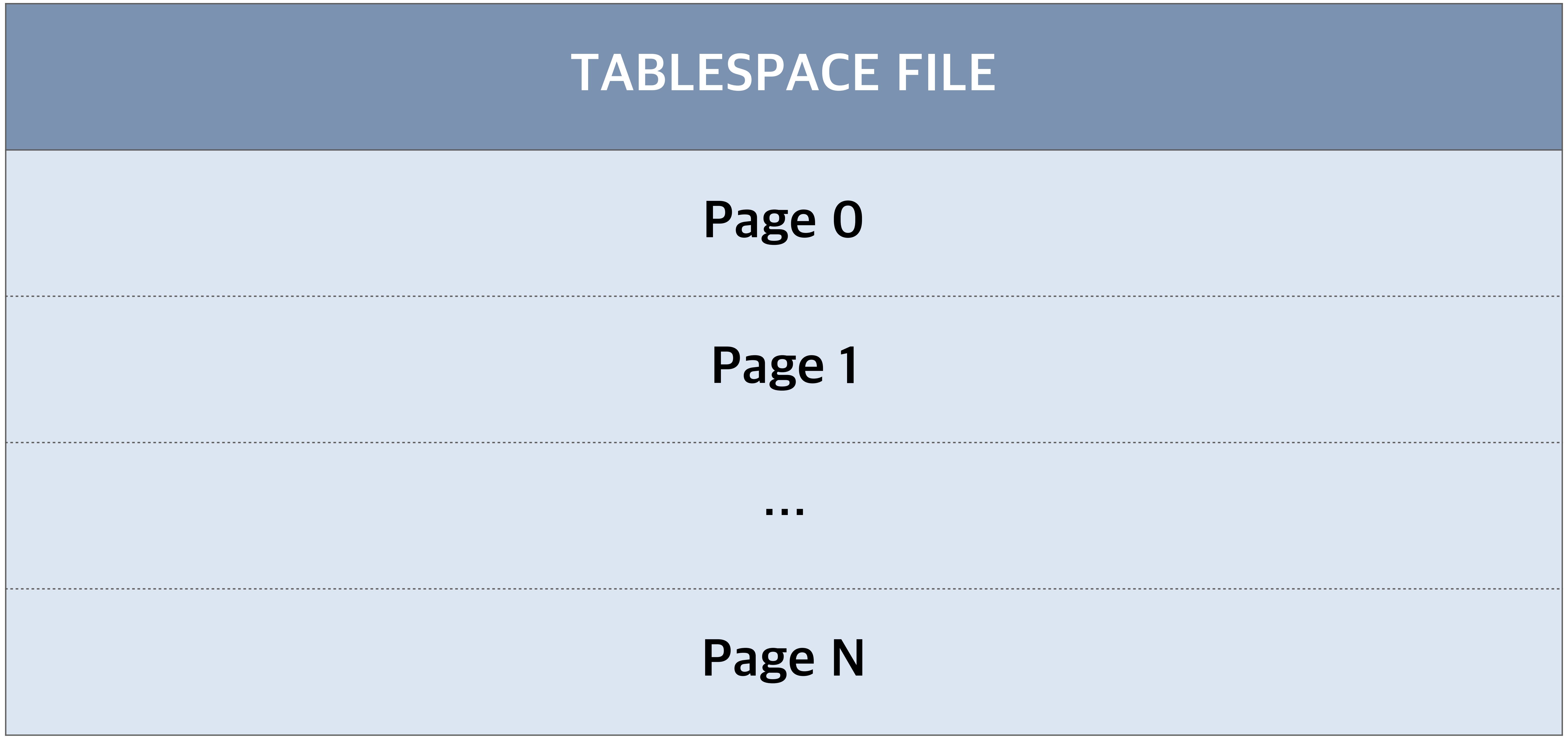
```
CREATE TABLE test.t1 (c INT) engine=InnoDB;

$ cd PATH_TO_DATA_DIR/test
$ ls
t1.ibd
```



- Table and index data related to the table `t1` will reside in the above file
- For a file-per-table tablespace, the tablespace name is the same as that of the file/table name
- Tablespaces are identified with a unique ID (i.e., **tablespace ID**)

Pages

- Each **tablespace** consists of number of **fixed size pages** (default: `innodb_page_size=16KB`)
- There are different type of pages to server different purpose



TABLESPACE FILE

Page 0

Page 1

...

Page N

Extents

- The **pages** are grouped into **extents** of size 1MB for pages up to 16KB in size
  - For a page size of 32KB, extent size is 2MB (for 64KB page, 4MB extent)

TABLESPACE FILE	
Extent 1	Extent 1 Page 0
	Extent 1 Page 1
	...
	Extent 1 Page 63
...	...
Extent N	Extent N Page 0
	Extent N Page 1
	...
	Extent N Page 63

Segments

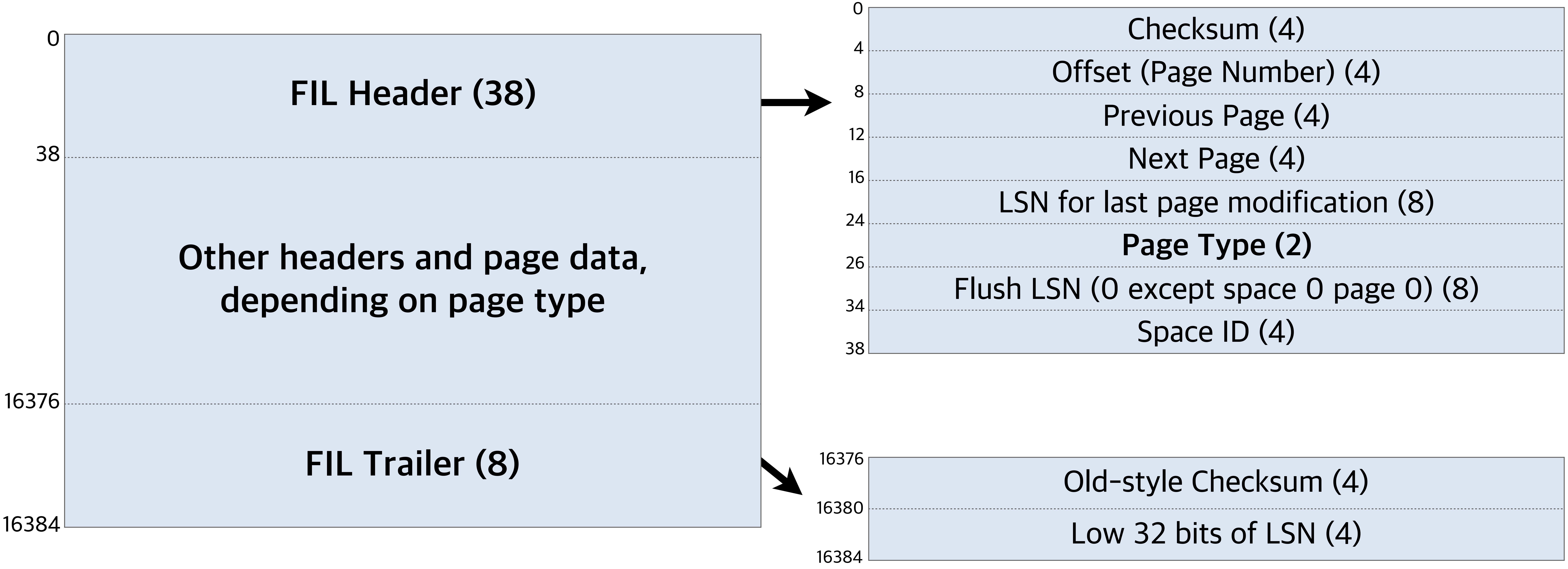
- The **files inside a tablespace** are called **segments** in InnoDB

- Within a **file-per-table tablespace**:
  - Table data: in **one** segment / Index: in its own segment
- The **system tablespace** contains **many different segments**:
  - It can hold many tables and their associated indexes
  - **Rollback segments is separated from the system tablespace since MySQL 8.0**

- When a segment needs more room, it is extended by **one extent (1MB)** at a time:
  - InnoDB can add up to **4 extents** at a time to ensure good **sequentiality** of data

Basic Page Overview

- Every page has a 38-byte FIL (file) header and 8-byte FIL trailer
- **The space ID and page number** is stored in the **header**



0

FIL Header (38)

38

Other headers and page data, depending on page type

16376

FIL Trailer (8)

16384

0

Checksum (4)

4

Offset (Page Number) (4)

8

Previous Page (4)

12

Next Page (4)

16

LSN for last page modification (8)

24

Page Type (2)

26

Flush LSN (0 except space 0 page 0) (8)

34

Space ID (4)

38

16376

Old-style Checksum (4)

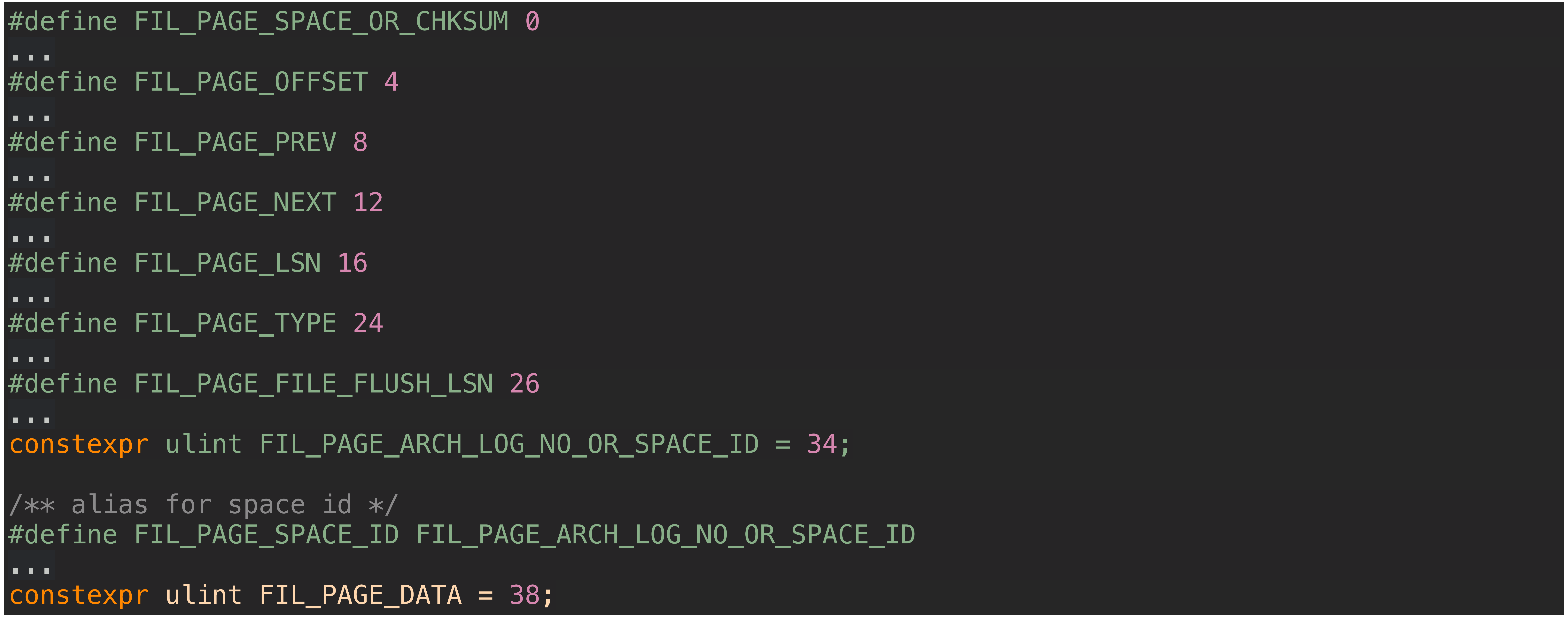
16380

Low 32 bits of LSN (4)

16384

Basic Page Overview: FIL Header

- include/fil0types.h



```
#define FIL_PAGE_SPACE_OR_CHKSUM 0
...
#define FIL_PAGE_OFFSET 4
...
#define FIL_PAGE_PREV 8
...
#define FIL_PAGE_NEXT 12
...
#define FIL_PAGE_LSN 16
...
#define FIL_PAGE_TYPE 24
...
#define FIL_PAGE_FILE_FLUSH_LSN 26
...
constexpr ulint FIL_PAGE_ARCH_LOG_NO_OR_SPACE_ID = 34;

/** alias for space id */
#define FIL_PAGE_SPACE_ID FIL_PAGE_ARCH_LOG_NO_OR_SPACE_ID
...
constexpr ulint FIL_PAGE_DATA = 38;
```

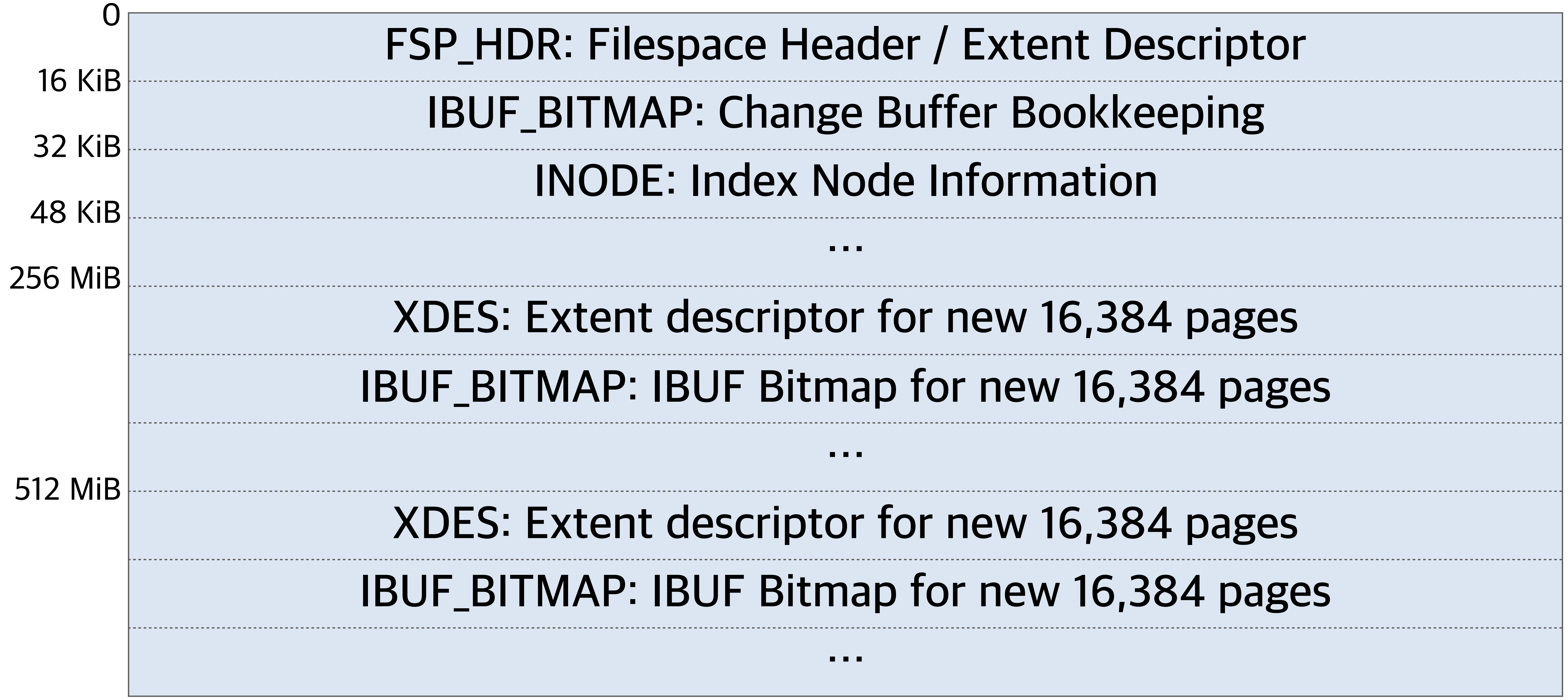
Basic Page Overview: FIL Trailer

- include/fil0types.h

```
/** File page trailer */
/** the low 4 bytes of this are used to store the page checksum, the
last 4 bytes should be identical to the last 4 bytes of FIL_PAGE_LSN */
constexpr ulint FIL_PAGE_END_LSN_OLD_CHKSUM = 8;

/** size of the page trailer */
constexpr ulint FIL_PAGE_DATA_END = 8;
```

Space File Overview

- A space file is just a concatenation of many pages
- The first page (**page 0**) in a space is ALWAYS an **FSP_HDR (file space header) page**
- An FSP_HDR page only store bookkeeping info for 256 extents (= 16,384 pages = 256 MiB)
  - Additional space must be reserved for bookkeeping info in the form of an **XDES page**



0

FSP_HDR: Filespace Header / Extent Descriptor

16 KiB

IBUF_BITMAP: Change Buffer Bookkeeping

32 KiB

INODE: Index Node Information

48 KiB

…

256 MiB

XDES: Extent descriptor for new 16,384 pages

IBUF_BITMAP: IBUF Bitmap for new 16,384 pages

…

512 MiB

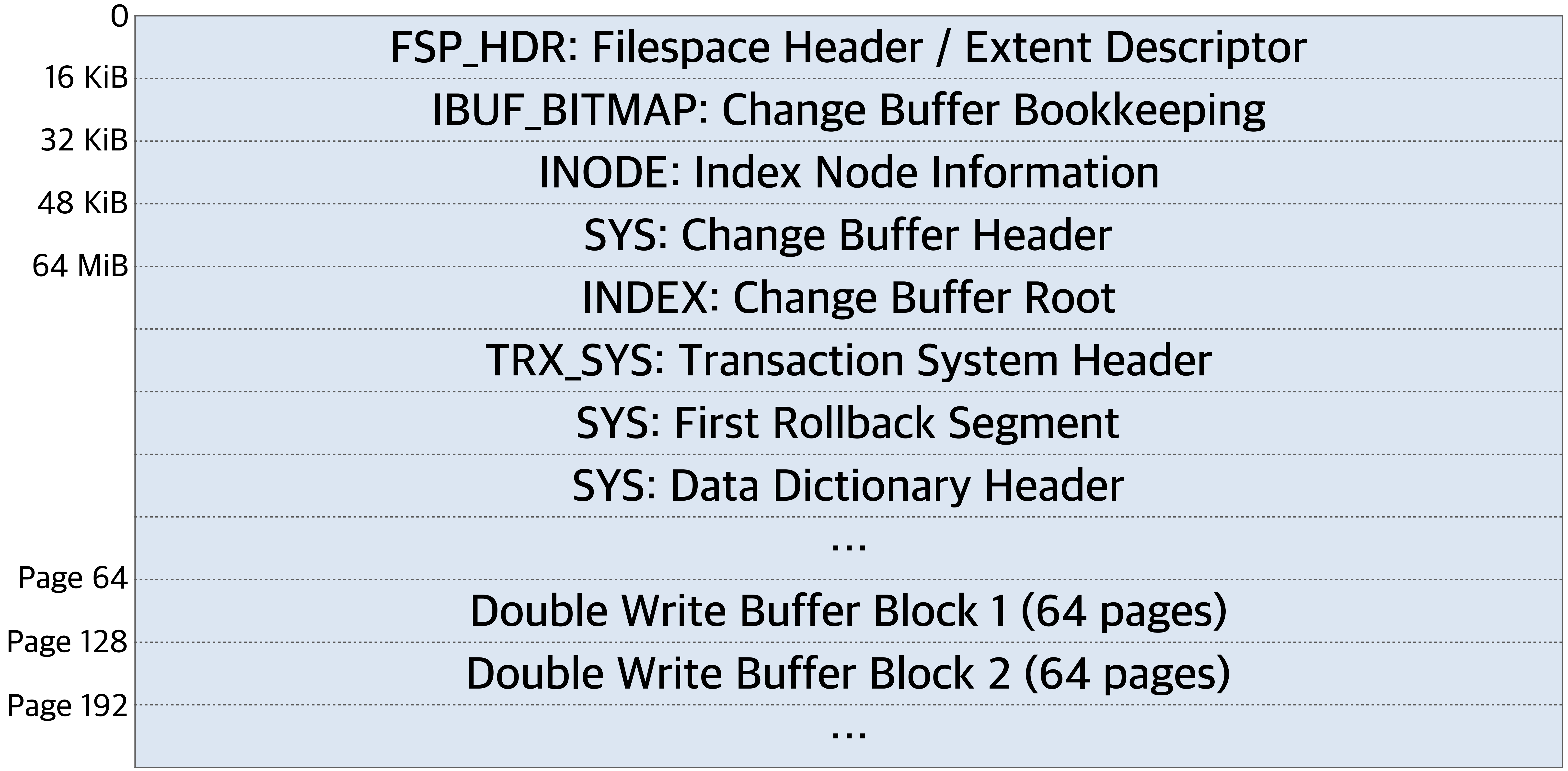
XDES: Extent descriptor for new 16,384 pages

IBUF_BITMAP: IBUF Bitmap for new 16,384 pages

…

`ibdata1` File Overview

- **The system tablesapce (space 0)** is special in InnoDB
  - To store a wide range of information critical to InnoDB’s operation



0

FSP_HDR: Filespace Header / Extent Descriptor

16 KiB

IBUF_BITMAP: Change Buffer Bookkeeping

32 KiB

INODE: Index Node Information

48 KiB

SYS: Change Buffer Header

64 MiB

INDEX: Change Buffer Root

TRX_SYS: Transaction System Header

SYS: First Rollback Segment

SYS: Data Dictionary Header

...

Page 64

Double Write Buffer Block 1 (64 pages)

Page 128

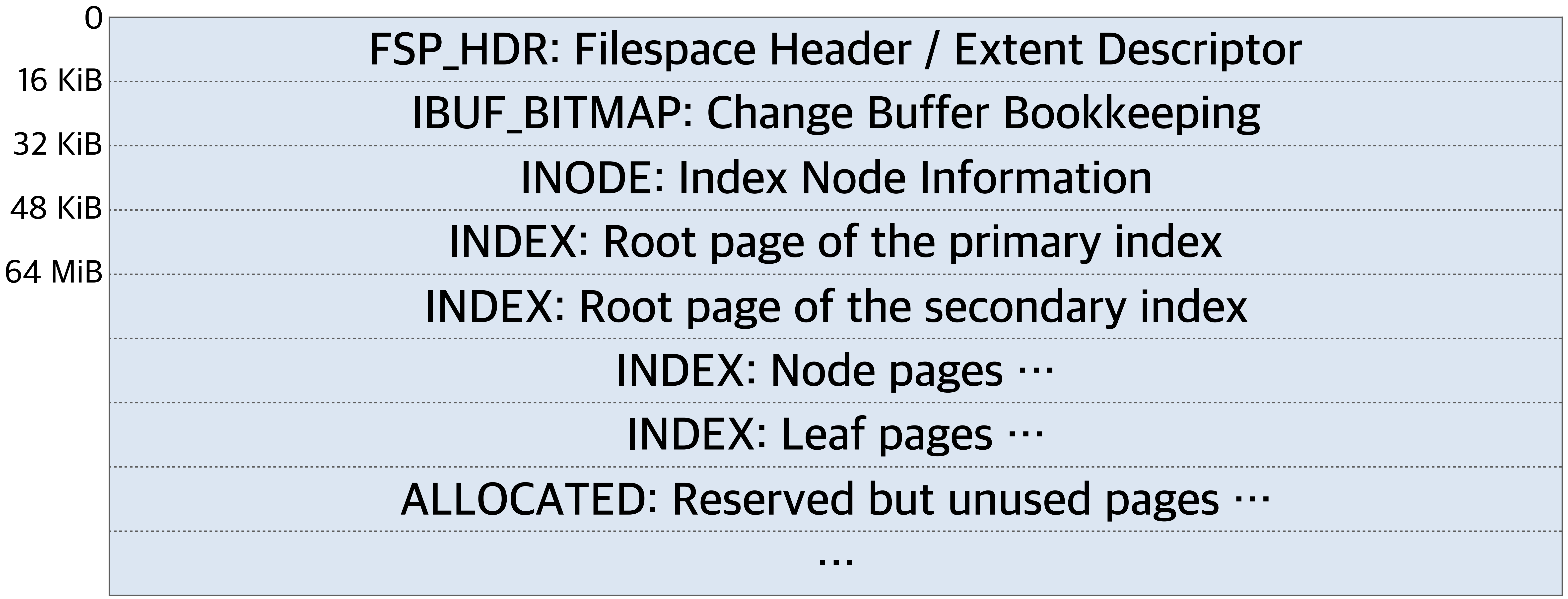
Double Write Buffer Block 2 (64 pages)

Page 192

...

`.ibd` File Overview

- **File-Per-Table** tablespace
  - The `.ibd` file created for each table has the typical space file structure



0

FSP_HDR: Filespace Header / Extent Descriptor

16 KiB

IBUF_BITMAP: Change Buffer Bookkeeping

32 KiB

INODE: Index Node Information

48 KiB

INDEX: Root page of the primary index

64 MiB

INDEX: Root page of the secondary index

INDEX: Node pages …

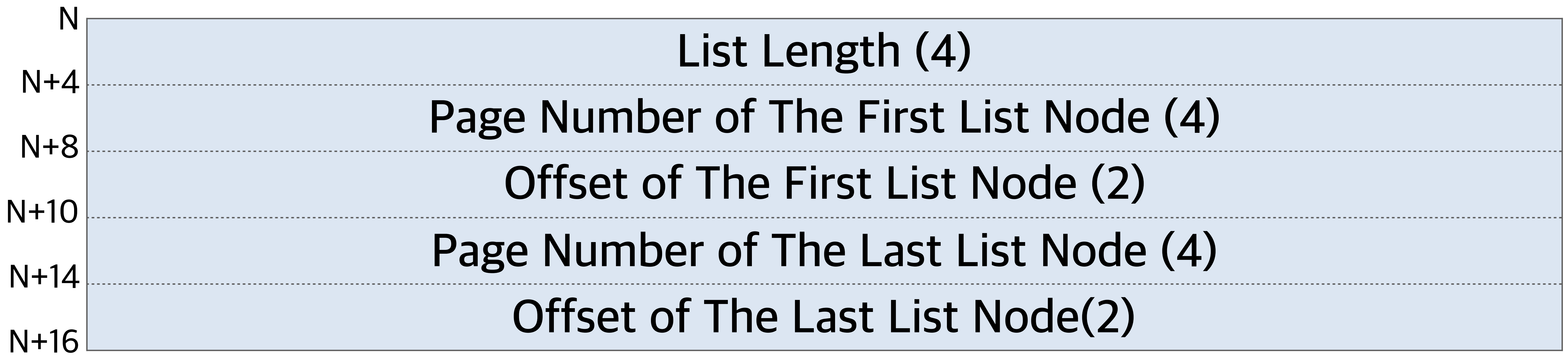
INDEX: Leaf pages …

ALLOCATED: Reserved but unused pages …

…

List Base Node

- Lists are fairly generic structure that allows linking multiple related structures together
- The base node is stored only once in some high level structure (e.g., FSP header)



N

List Length (4)

N+4

Page Number of The First List Node (4)

N+8

Offset of The First List Node (2)

N+10

Page Number of The Last List Node (4)

N+14

Offset of The Last List Node(2)

N+16

- include/fut0lst.h

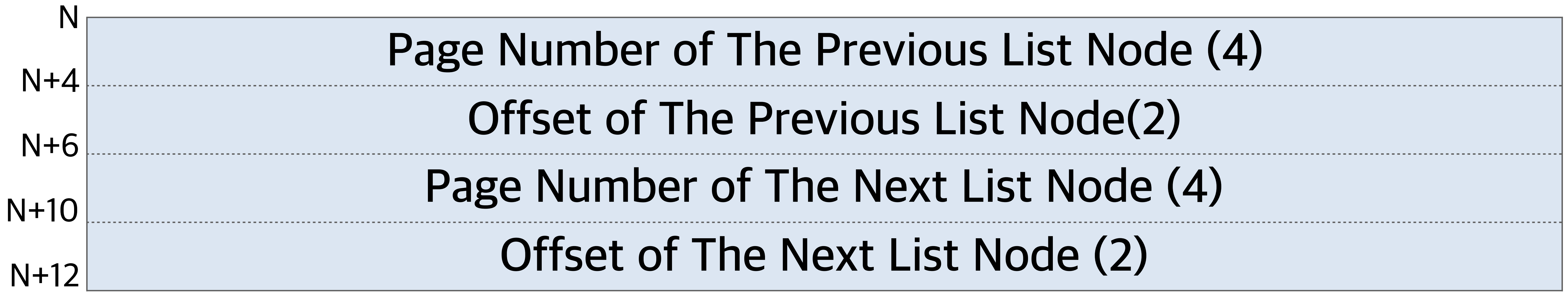
```
typedef byte flst_base_node_t;

/* The physical size of a list base node in bytes */            6
constexpr ulint FLST_BASE_NODE_SIZE = 4 + 2 * FIL_ADDR_SIZE;
```

- include/fut0lst.ic

```
#define FLST_LEN 0 /* 32-bit list length field */
#define FLST_FIRST 4 /* 6-byte address of the first element of the list */
#define FLST_LAST (4 + FIL_ADDR_SIZE) /* 6-byte address of the last element of the list */
```

List Node

- The list node stores previous and next pointers



N

Page Number of The Previous List Node (4)

N+4

Offset of The Previous List Node(2)

N+6

Page Number of The Next List Node (4)

N+10

Offset of The Next List Node (2)

N+12

- include/fut0lst.h

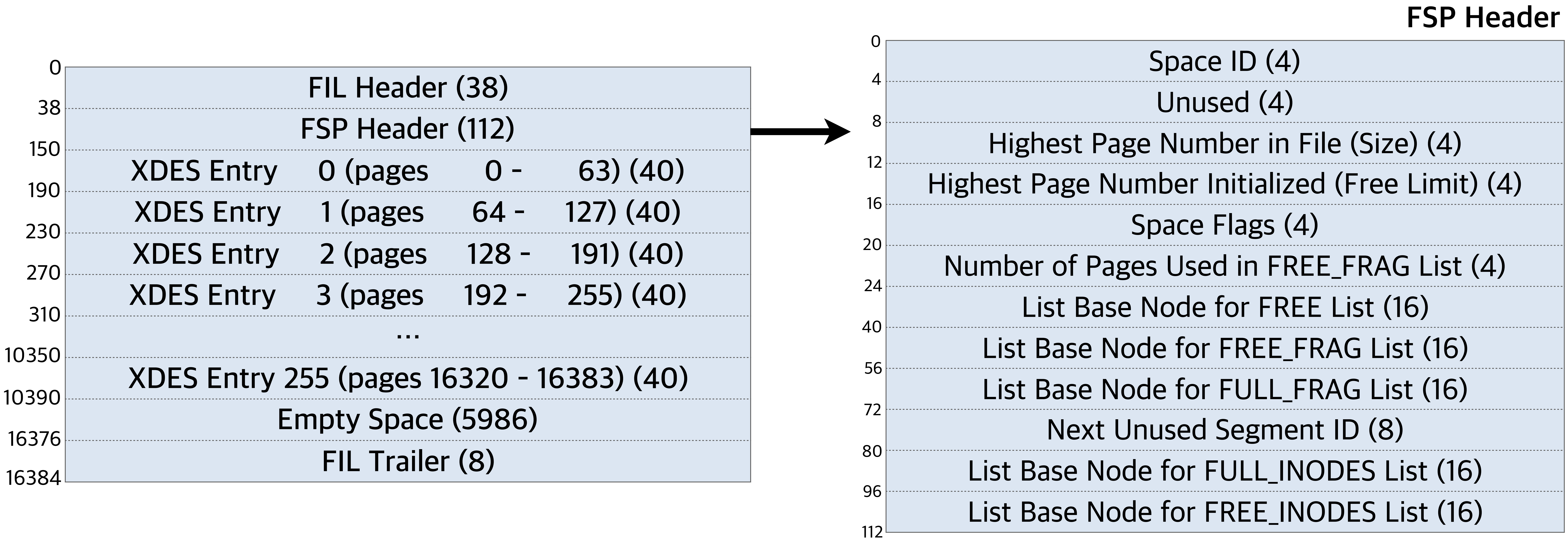
```
typedef byte flst_node_t;

/* The physical size of a list node in bytes */
constexpr ulint FLST_NODE_SIZE = 2 * FIL_ADDR_SIZE;
```

- include/fut0lst.ic

```
#define FLST_PREV 0 /* 6-byte address of the previous list element */
#define FLST_NEXT FIL_ADDR_SIZE /* 6-byte address of the next */
```

Header Page

- There is no separate storage for metadata information of tablespace
- It is stored in the same file in header page (**always page 0**)



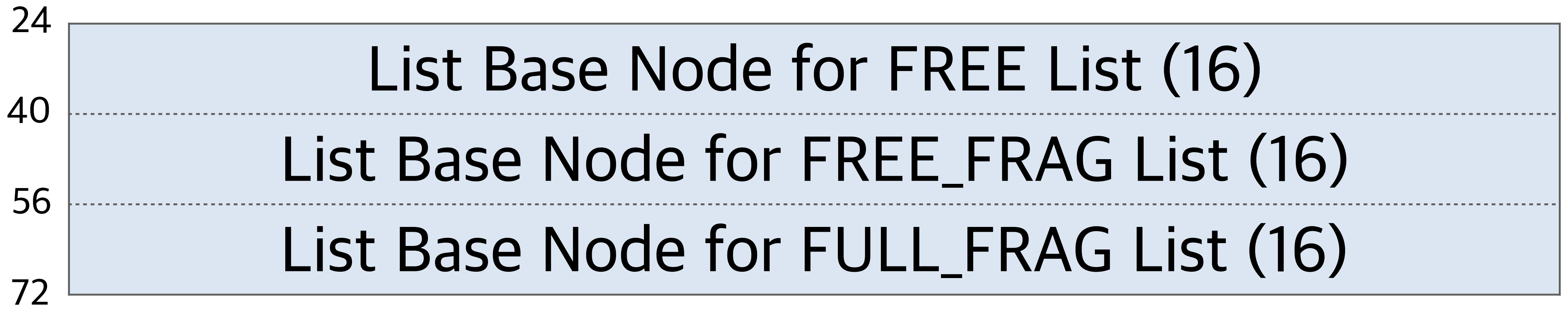
Offset	
0	FIL Header (38)
38	FSP Header (112)
150	XDES Entry 0 (pages 0 - 63) (40)
190	XDES Entry 1 (pages 64 - 127) (40)
230	XDES Entry 2 (pages 128 - 191) (40)
270	XDES Entry 3 (pages 192 - 255) (40)
310	…
10350	XDES Entry 255 (pages 16320 - 16383) (40)
10390	Empty Space (5986)
16376	FIL Trailer (8)
16384	

FSP Header

Offset	
0	Space ID (4)
4	Unused (4)
8	Highest Page Number in File (Size) (4)
12	Highest Page Number Initialized (Free Limit) (4)
16	Space Flags (4)
20	Number of Pages Used in FREE_FRAG List (4)
24	List Base Node for FREE List (16)
40	List Base Node for FREE_FRAG List (16)
56	List Base Node for FULL_FRAG List (16)
72	Next Unused Segment ID (8)
80	List Base Node for FULL_INODES List (16)
96	List Base Node for FREE_INODES List (16)
112	

Types of List Base node in FSP Header

- FREE List
  - Base node pointer of the linked-list of extents which are **free** to be allocated
  - An extent from this list could be allocated to a File Segment or FREE_FRAG List
- FREE_FRAG List
  - Base node pointer of the linked-list of extents which have **at least one free page** to be allocated
- FULL_FRAG List
  - Base node pointer of the linked-list of extents which have **no free page** left to be allocated



24

List Base Node for FREE List (16)

40

List Base Node for FREE_FRAG List (16)

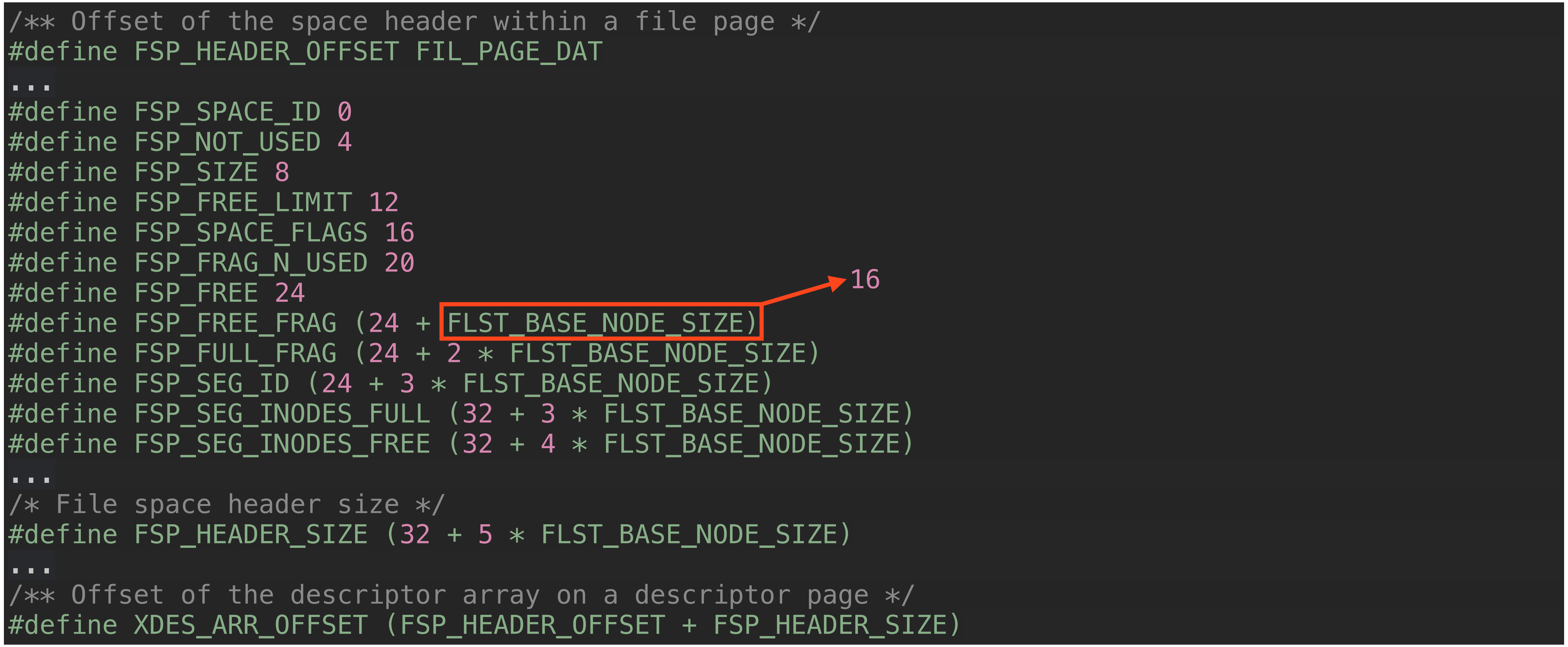
56

List Base Node for FULL_FRAG List (16)

72

FSP Header

- include/fsp0fsp.h

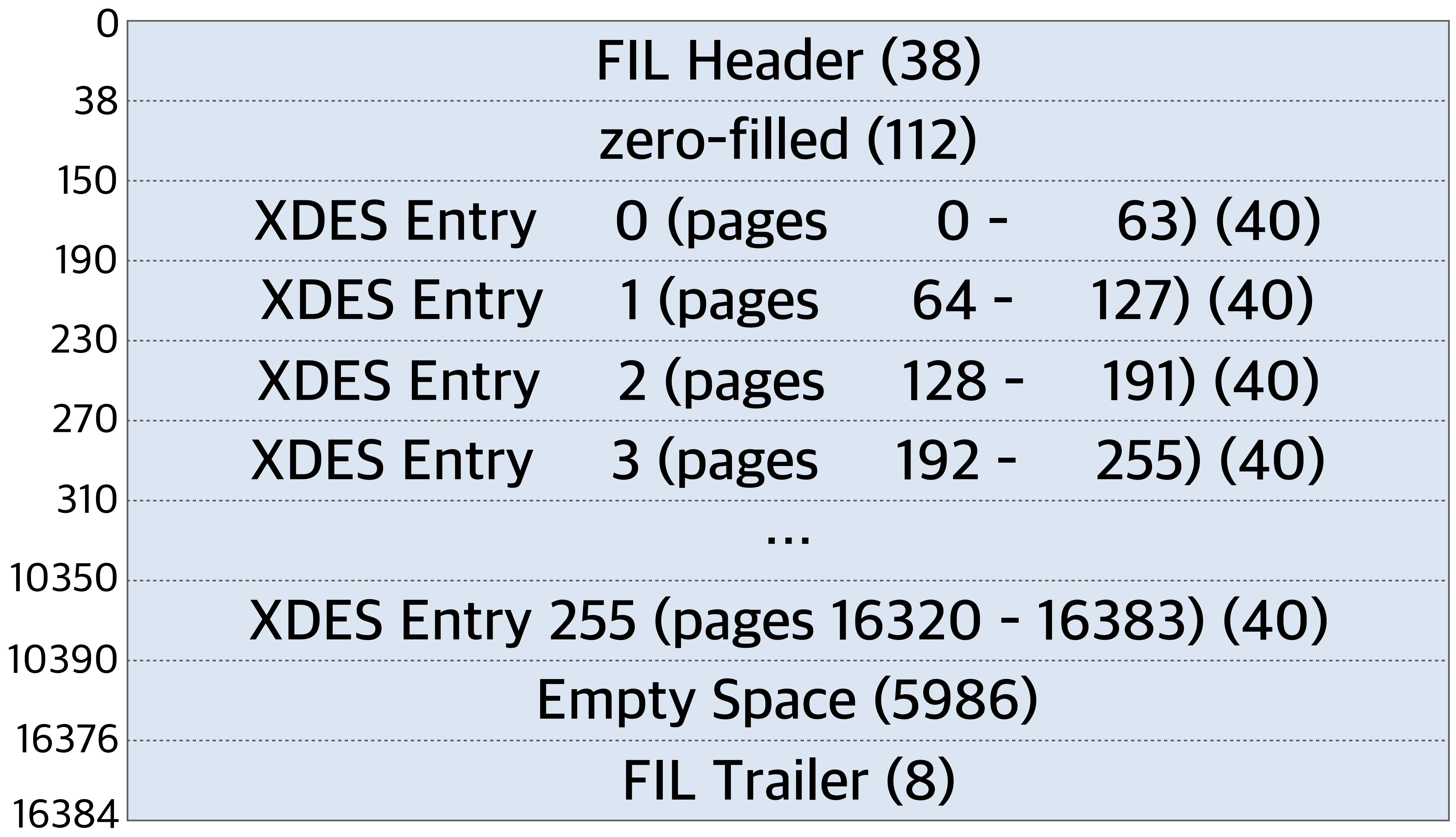


```
/** Offset of the space header within a file page */
#define FSP_HEADER_OFFSET FIL_PAGE_DAT
...
#define FSP_SPACE_ID 0
#define FSP_NOT_USED 4
#define FSP_SIZE 8
#define FSP_FREE_LIMIT 12
#define FSP_SPACE_FLAGS 16
#define FSP_FRAG_N_USED 20
#define FSP_FREE 24
#define FSP_FREE_FRAG (24 + FLST_BASE_NODE_SIZE)
#define FSP_FULL_FRAG (24 + 2 * FLST_BASE_NODE_SIZE)
#define FSP_SEG_ID (24 + 3 * FLST_BASE_NODE_SIZE)
#define FSP_SEG_INODES_FULL (32 + 3 * FLST_BASE_NODE_SIZE)
#define FSP_SEG_INODES_FREE (32 + 4 * FLST_BASE_NODE_SIZE)
...
/* File space header size */
#define FSP_HEADER_SIZE (32 + 5 * FLST_BASE_NODE_SIZE)
...
/** Offset of the descriptor array on a descriptor page */
#define XDES_ARR_OFFSET (FSP_HEADER_OFFSET + FSP_HEADER_SIZE)
```

16

XDES Page

- XDES pages store **metadata information related to pages which belongs to an Extent**
- Once the number of total extents is greater than what an XDES page can track:
  - a new XDES page is allocated which will be used to track the next set of extents
- **NOTE: For first set of extents, header page is used to store XDES entries**



0

FIL Header (38)

38

zero-filled (112)

150

XDES Entry 0 (pages 0 - 63) (40)

190

XDES Entry 1 (pages 64 - 127) (40)

230

XDES Entry 2 (pages 128 - 191) (40)

270

XDES Entry 3 (pages 192 - 255) (40)

310

...

10350

XDES Entry 255 (pages 16320 - 16383) (40)

10390

Empty Space (5986)

16376

FIL Trailer (8)

16384

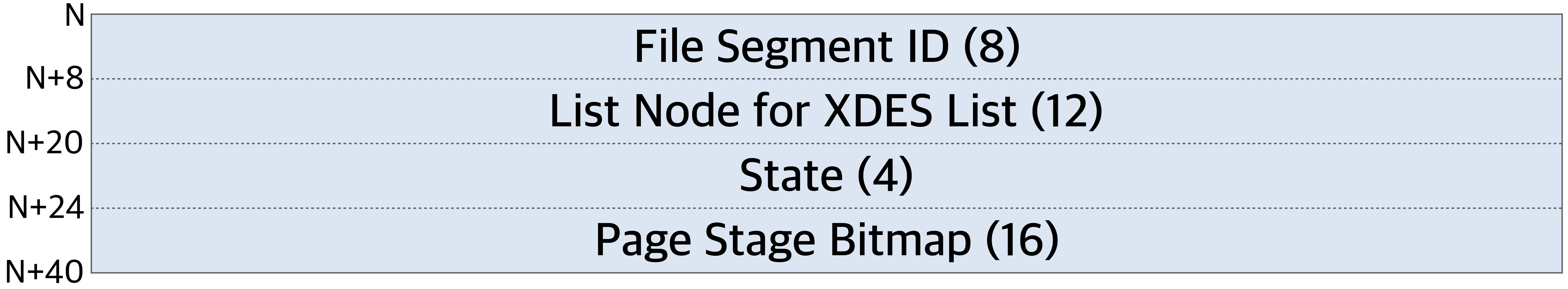
Pages Covered by One XDES Page

- For ease of implementation, **number of pages covered by one XDES page entries is equal to page size**

Page Size	Extent Size	Pages in An Extent	XDES Entry Size	Pages Covered in An XDES Page	XDES Entries in An XDES Page
4 K	1 M	256	88 B	4096 (4K)	16
8 K	1 M	128	56 B	8192 (8K)	64
16 K	1 M	64	40 B	16384 (16K)	256
32 K	2 M	64	40 B	32768 (32K)	512
64 K	4 M	64	40 B	65536 (64K)	1024

XDES Entry

- To keep track of which extents are in use, and which pages within each extent are in use
  - **List Node for XDES List**: Pointers to previous and next extents in a doubly-linked extent descriptor list
  - **State**: The current state of the extent (FREE, FREE_FRAG, FULL_FRAG, FSEG)
  - **Page State Bitmap**: A bitmap of 2 bits per page in the extent (64 * 2 = 128 bits = 16 bytes)
    - The first bit indicates whether the page is free
    - The second bit indicates whether the page is clean, but this bit is currently not used



N

File Segment ID (8)

N+8

List Node for XDES List (12)

N+20

State (4)

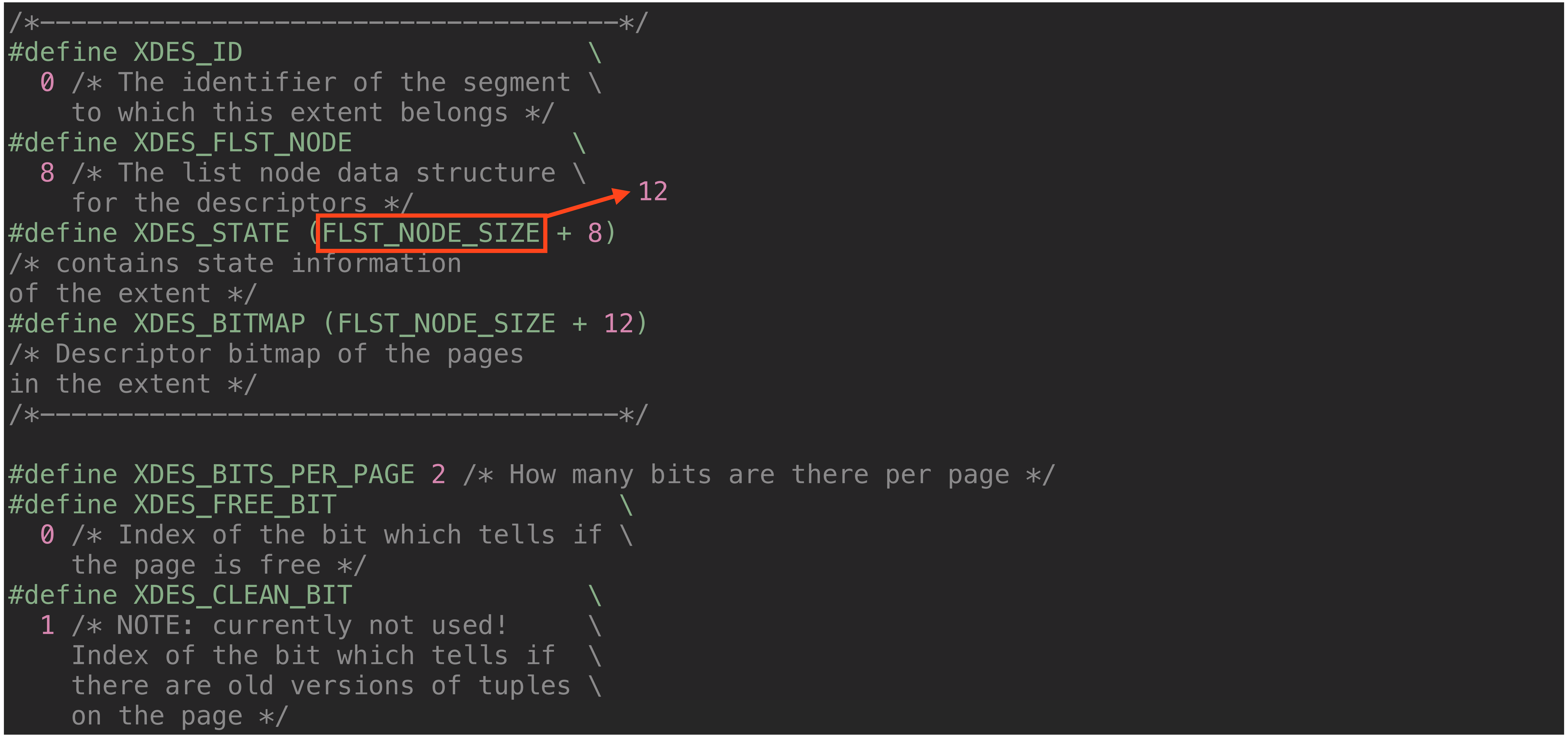
N+24

Page Stage Bitmap (16)

N+40

XDES Entry

- include/fsp0fsp.h



```
/*-------------------------------------*/
#define XDES_ID                      \
  0 /* The identifier of the segment \
    to which this extent belongs */
#define XDES_FLST_NODE              \
  8 /* The list node data structure \
    for the descriptors */
#define XDES_STATE (FLST_NODE_SIZE + 8)
/* contains state information
of the extent */
#define XDES_BITMAP (FLST_NODE_SIZE + 12)
/* Descriptor bitmap of the pages
in the extent */
/*-------------------------------------*/

#define XDES_BITS_PER_PAGE 2 /* How many bits are there per page */
#define XDES_FREE_BIT                  \
  0 /* Index of the bit which tells if \
    the page is free */
#define XDES_CLEAN_BIT               \
  1 /* NOTE: currently not used!     \
    Index of the bit which tells if  \
    there are old versions of tuples \
    on the page */
```

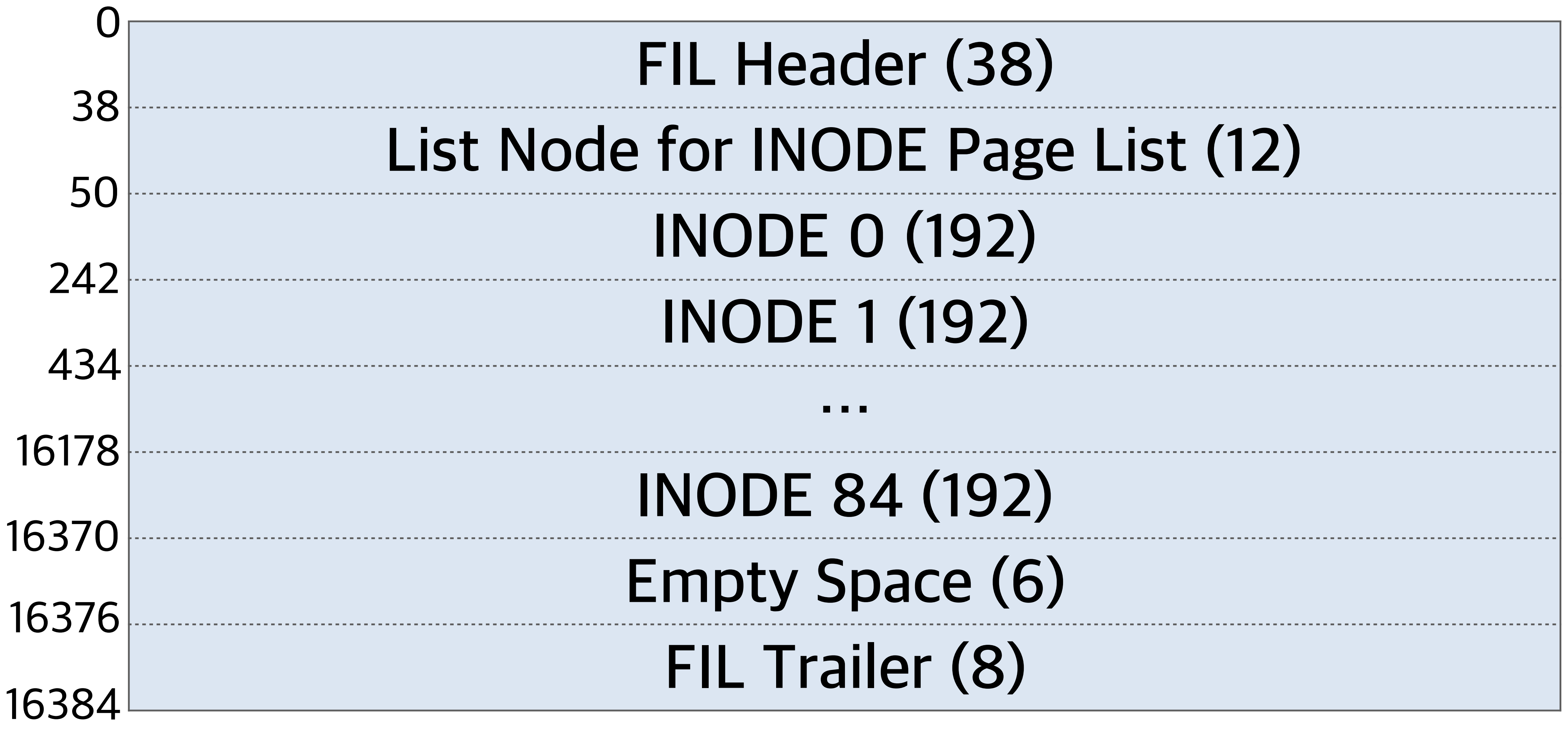
12

Three Types of State

- `FREE_FRAG`
  - **Extents with free pages** remaining that are allocated to be **used in “fragments**”, having Individual pages allocated to different purposes rather than allocating the entire extent
- `FULL_FRAG`
  - Exactly like `FREE_FRAG` but for extents with **no free pages remaining**
  - Extents are moved from `FREE_FRAG` to `FULL_FRAG` when they become full, and moved back to `FREE_FRAG` if a page is released so that they are no longer full
- `FREE`
  - Extents that are **completely unused** and available to be allocated in whole to some purpose
  - A `FREE` extent could be allocated to a file segment or moved to the `FREE_FRAG` list for individual page use

INODE Page Overview

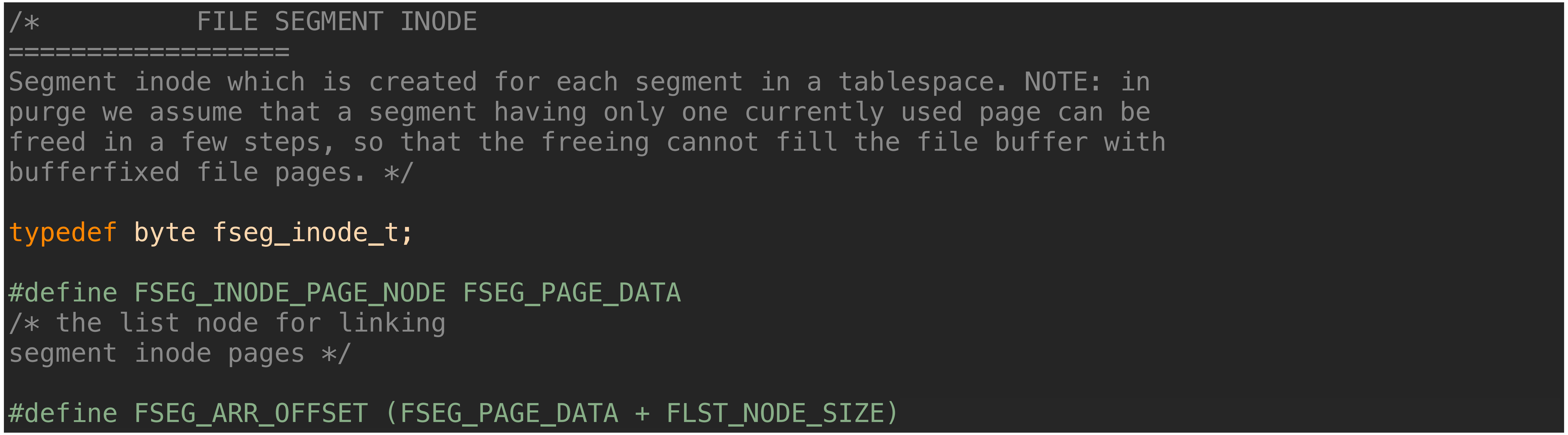
- An **INODE** entry in InnoDB merely **describes a file segment (FSEG)**
- Each INODE page contains 85 file segments INODE entries, each of which are 192 bytes
  - Also, the page contain a list node which is used in the following lists:
    - `FREE_INODES`: A list of INODE pages which have at least one free file segment INODE entry
    - `FULL_INODES`: A list of INODE pages which have zero free file segment INODE entries



0
FIL Header (38)
38
List Node for INODE Page List (12)
50
INODE 0 (192)
242
INODE 1 (192)
434
...
16178
INODE 84 (192)
16370
Empty Space (6)
16376
FIL Trailer (8)
16384

INODE Page Overview

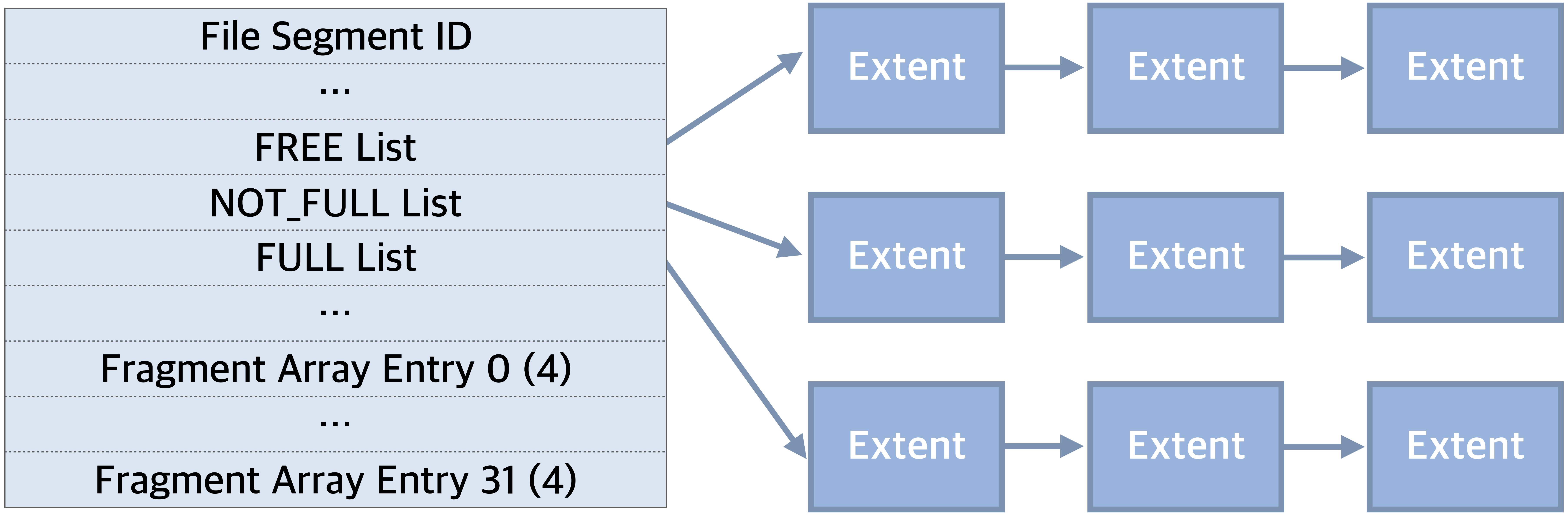
- include/fsp0fsp.h



```
/*          FILE SEGMENT INODE
==================
Segment inode which is created for each segment in a tablespace. NOTE: in
purge we assume that a segment having only one currently used page can be
freed in a few steps, so that the freeing cannot fill the file buffer with
bufferfixed file pages. */

typedef byte fseg_inode_t;

#define FSEG_INODE_PAGE_NODE FSEG_PAGE_DATA
/* the list node for linking
segment inode pages */

#define FSEG_ARR_OFFSET (FSEG_PAGE_DATA + FLST_NODE_SIZE)
```

File Segment

- A logical unit which is a collection of pages and extents:



File Segment ID

...

FREE List

NOT_FULL List

FULL List

...

Fragment Array Entry 0 (4)

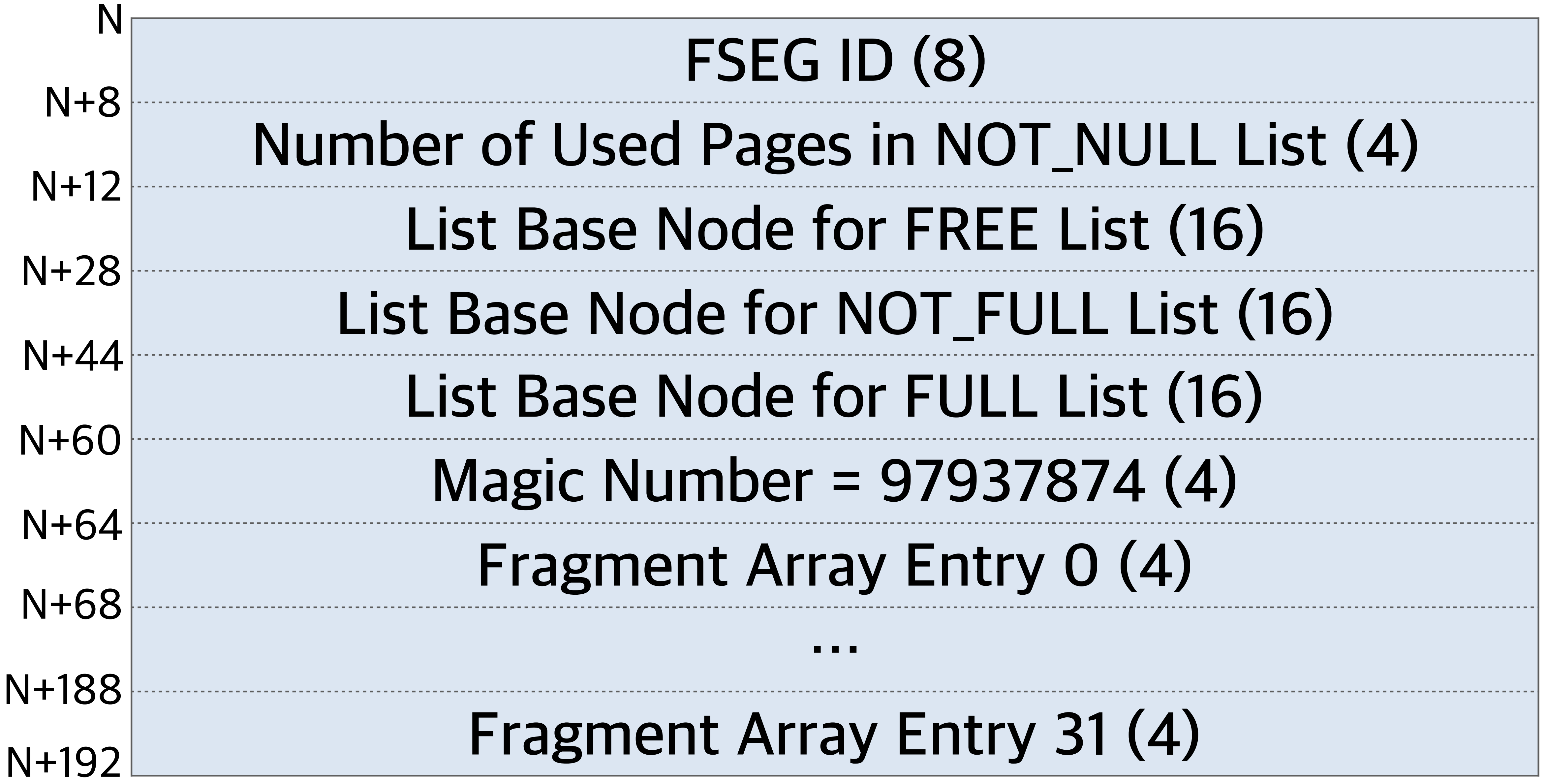
...

Fragment Array Entry 31 (4)

- File segment makes the page management easy
  - Once we delete the file segment, we know which all extents and pages are to be freed

INODE Entry

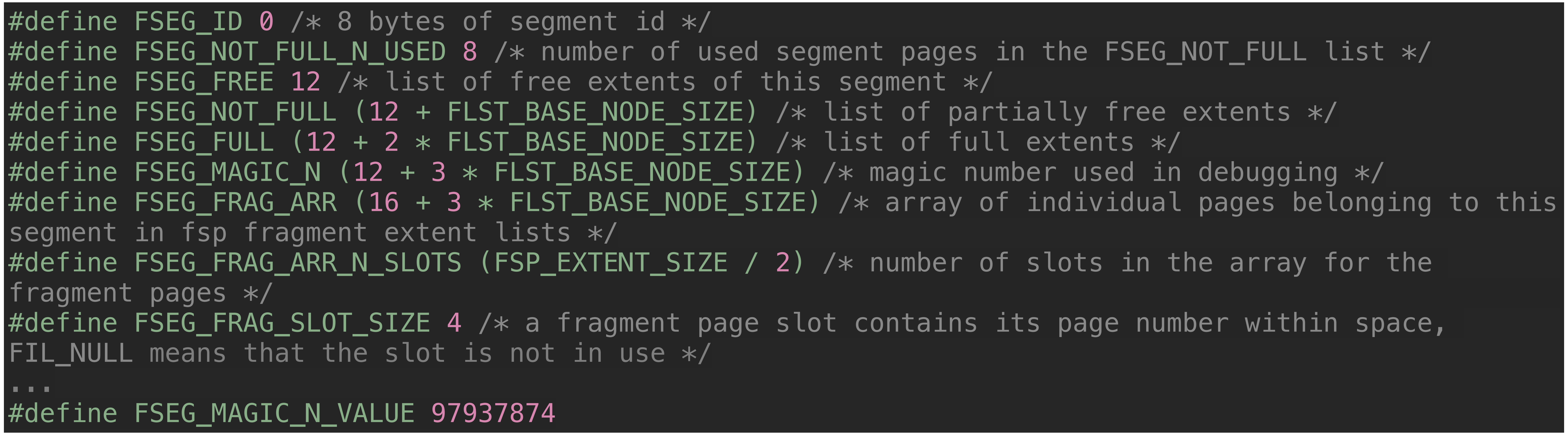
- **File Segment ID**: The ID of the file segment (FSEG) described by this file segment INODE entry
- **Magic Number**: A marker that this file segment INODE entry has been properly initialized
- **Fragment Array**: An array of 32 page numbers of pages allocated individually from extents in the space's FREE_FRAG or FULL_FRAG list of "fragment" extents



N	FSEG ID (8)
N+8	Number of Used Pages in NOT_NULL List (4)
N+12	List Base Node for FREE List (16)
N+28	List Base Node for NOT_FULL List (16)
N+44	List Base Node for FULL List (16)
N+60	Magic Number = 97937874 (4)
N+64	Fragment Array Entry 0 (4)
N+68	...
N+188	Fragment Array Entry 31 (4)
N+192	

INODE Entry

- include/fsp0fsp.h



```
#define FSEG_ID 0 /* 8 bytes of segment id */
#define FSEG_NOT_FULL_N_USED 8 /* number of used segment pages in the FSEG_NOT_FULL list */
#define FSEG_FREE 12 /* list of free extents of this segment */
#define FSEG_NOT_FULL (12 + FLST_BASE_NODE_SIZE) /* list of partially free extents */
#define FSEG_FULL (12 + 2 * FLST_BASE_NODE_SIZE) /* list of full extents */
#define FSEG_MAGIC_N (12 + 3 * FLST_BASE_NODE_SIZE) /* magic number used in debugging */
#define FSEG_FRAG_ARR (16 + 3 * FLST_BASE_NODE_SIZE) /* array of individual pages belonging to this
segment in fsp fragment extent lists */
#define FSEG_FRAG_ARR_N_SLOTS (FSP_EXTENT_SIZE / 2) /* number of slots in the array for the
fragment pages */
#define FSEG_FRAG_SLOT_SIZE 4 /* a fragment page slot contains its page number within space,
FIL_NULL means that the slot is not in use */
...
#define FSEG_MAGIC_N_VALUE 97937874
```

Three Types of List in File Segment

- `FREE`
  - Extents that are **completely unused** and are allocated to this file segment
- `NOT_FULL`
  - Extents **with at least one used page allocated** to this file segment
  - When the last free page is used the extent is moved to `FULL` list
- `FULL`
  - Extents **with no free pages allocated** to this file
  - If a page becomes free, the extent is moved to the `NOT_FULL` list

How Are File Segments Used in Index?

- **Two segments** are allocated for each **index** in InnoDB:
  - **Non leaf page segment**: to store **non-leaf** pages in B tree
  - **Leaf page segment**: to store **leaf** pages in B tree
- On a page, **FSEG HEADER** is the place where these two file segments INODE entries information is stored (detailed later)

Everything Is A Index in InnoDB

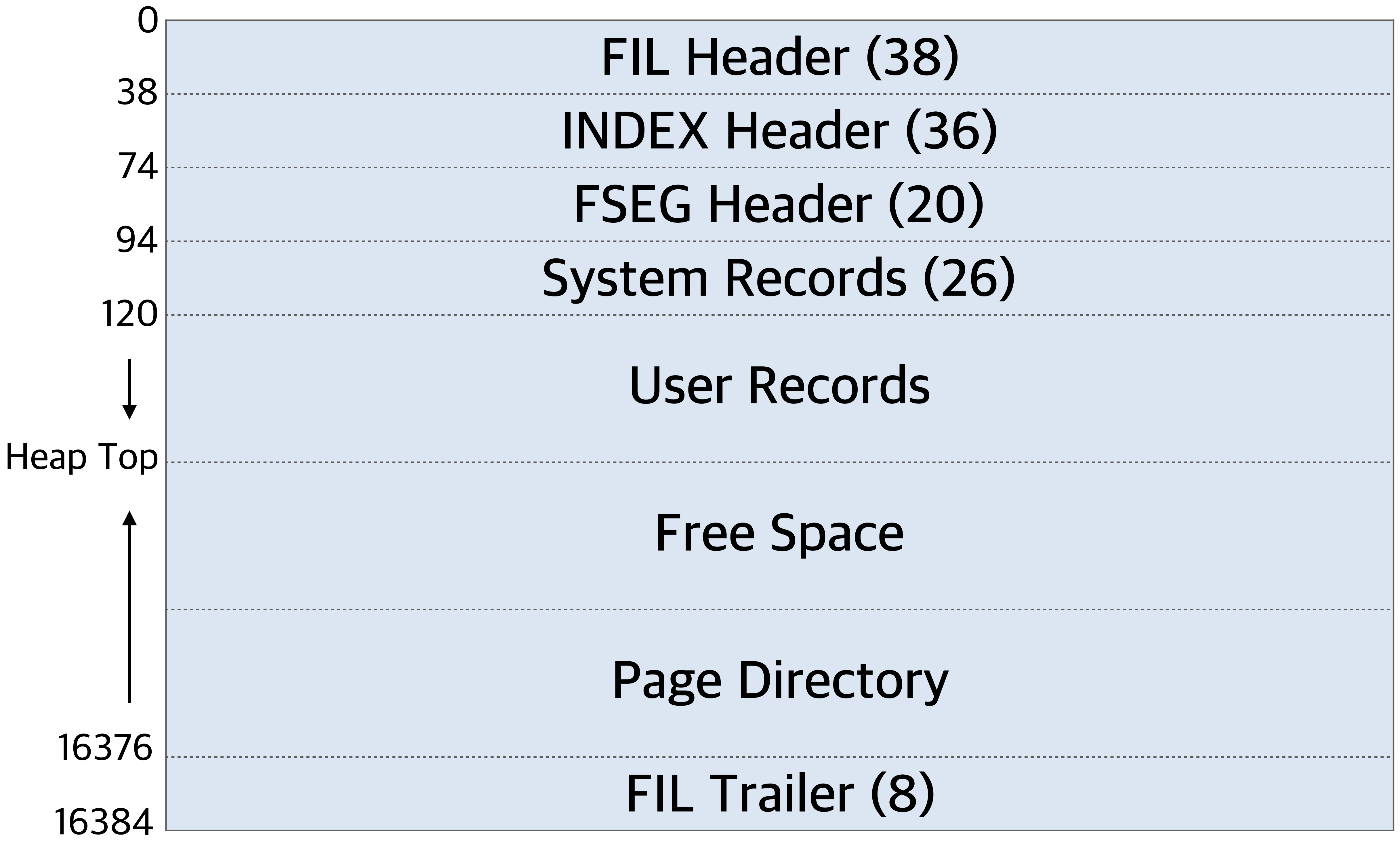
- **Every table has a primary key**
  - If the `CREATE TABLE` dose not specify one, the first **non-NULL unique key** is used
    - If failed, a **48-bit hidden “Row ID”** field is automatically added as a primary key

- **The row data are stored in the** `PRIMARY KEY INDEX` **structure (= clustered key)**
  - Key: `PRIMARY KEY` fields, Value: the row data

- **Secondary keys are stored in an identical index structure**
  - Key: `KEY` fields, Value: the primary key value

INDEX Page Overview

- **System Record**: InnoDB has two system records in each page, called `infimum` and `supremum`
- **User Record**: The actual data
  - Every record has a variable-length header with a “next record” pointer → Singly-linked list
- **Page Directory**: It contains pointers to some of the records in the page (every 4th to 8th record)



0

FIL Header (38)

38

INDEX Header (36)

74

FSEG Header (20)

94

System Records (26)

120

User Records

Heap Top

Free Space

Page Directory

16376

FIL Trailer (8)

16384

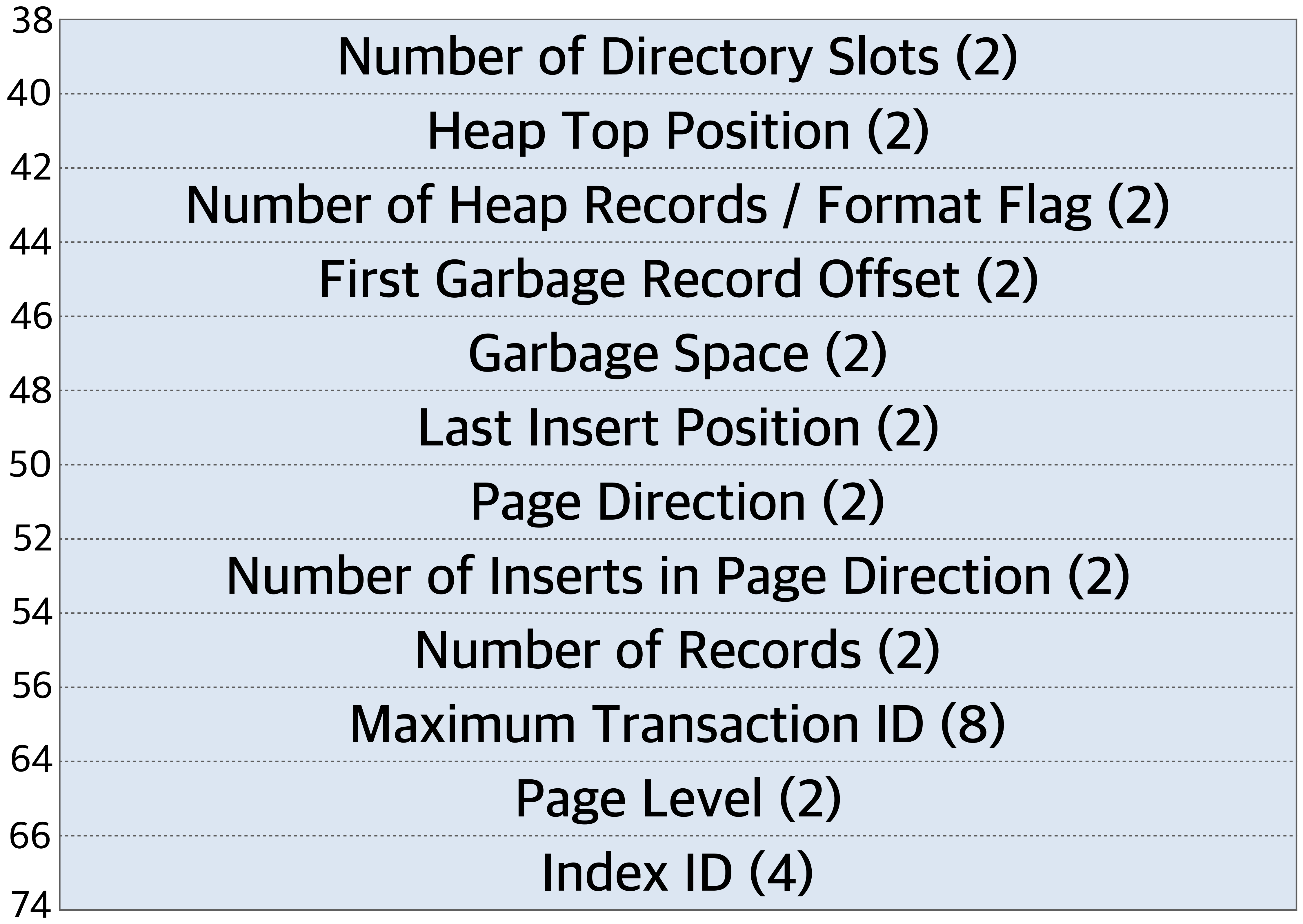
INDEX Page Overview

- include/

```
#define PAGE_DIR FIL_PAGE_DATA_END
...
#define PAGE_EMPTY_DIR_START (PAGE_DIR + 2 * PAGE_DIR_SLOT_SIZE)
```

INDEX Header

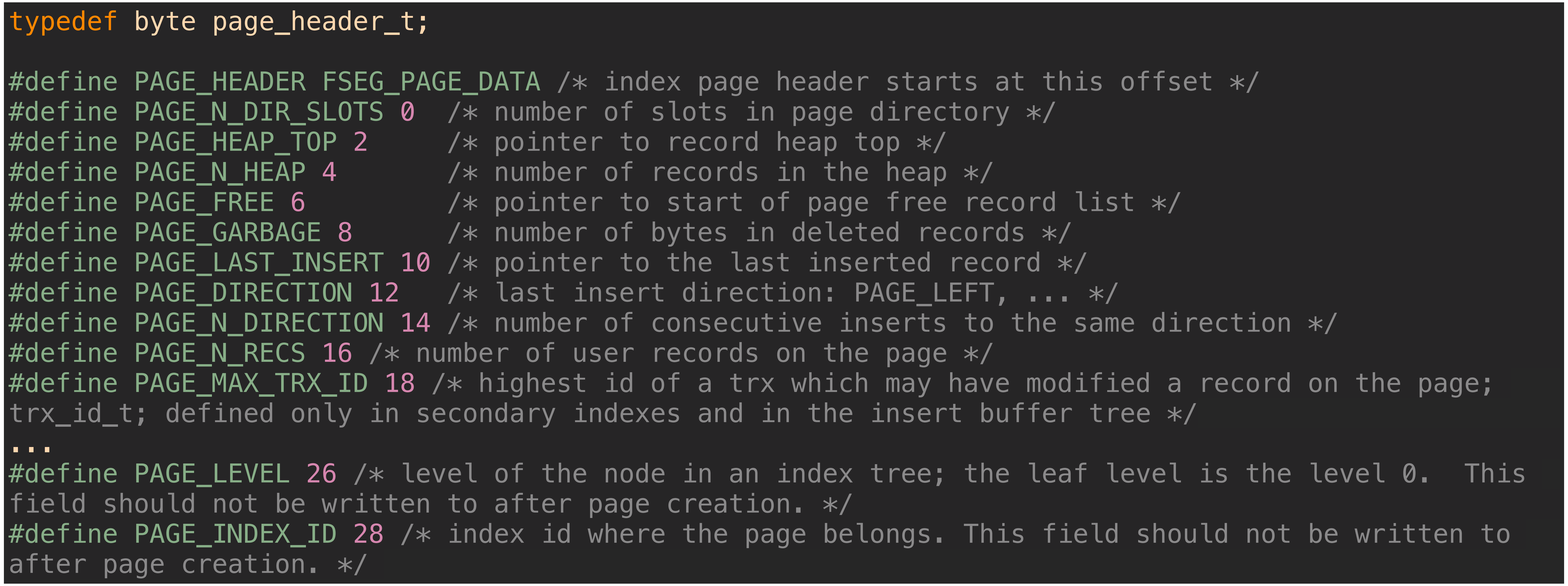
- The INDEX header in each INDEX page is **fixed-length**
- It contains many fields related to INDEX pages and record management



38	Number of Directory Slots (2)
40	Heap Top Position (2)
42	Number of Heap Records / Format Flag (2)
44	First Garbage Record Offset (2)
46	Garbage Space (2)
48	Last Insert Position (2)
50	Page Direction (2)
52	Number of Inserts in Page Direction (2)
54	Number of Records (2)
56	Maximum Transaction ID (8)
64	Page Level (2)
66	Index ID (4)
74	

INDEX Header

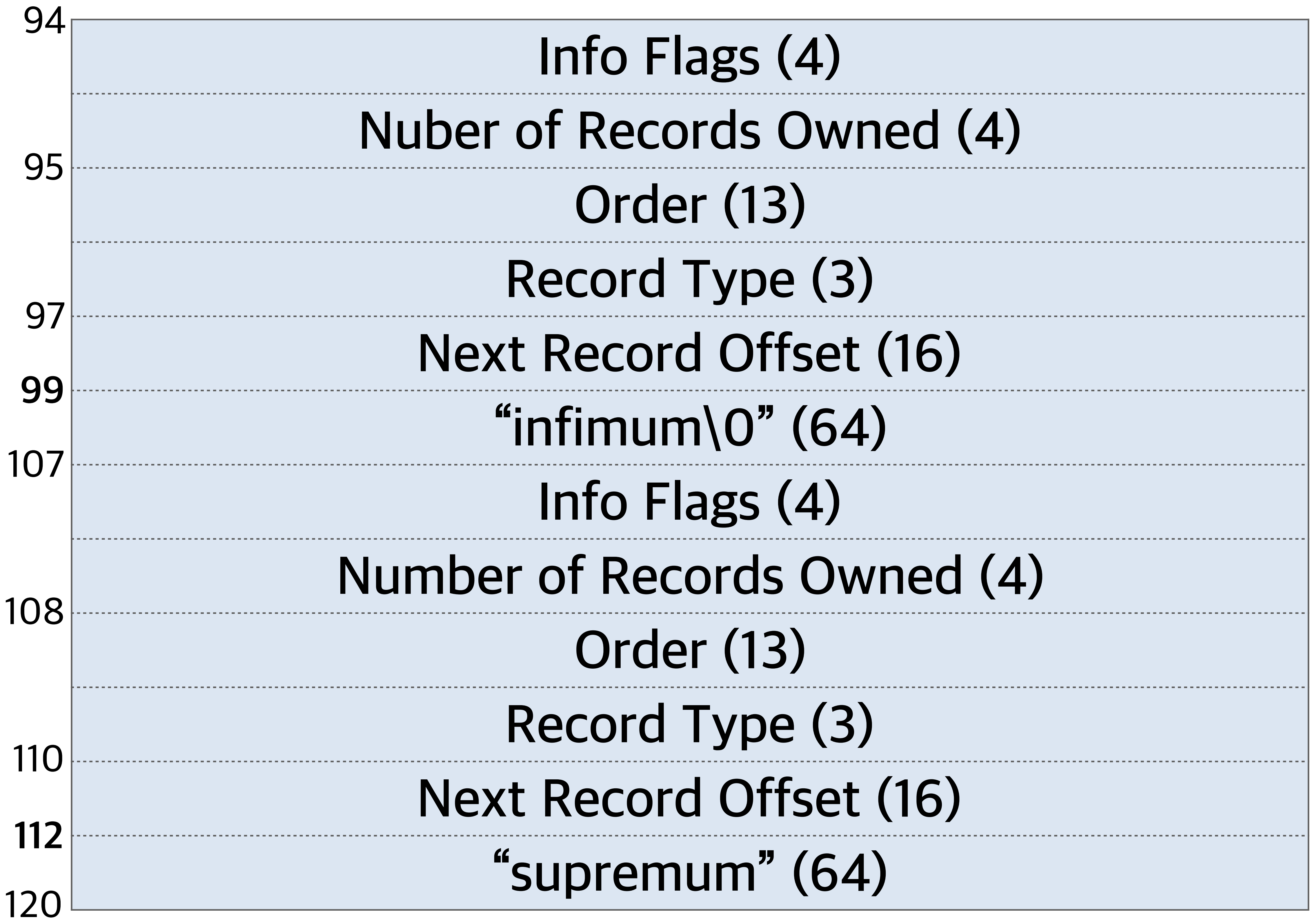
- include/page0types.h



```
typedef byte page_header_t;

#define PAGE_HEADER FSEG_PAGE_DATA /* index page header starts at this offset */
#define PAGE_N_DIR_SLOTS 0  /* number of slots in page directory */
#define PAGE_HEAP_TOP 2     /* pointer to record heap top */
#define PAGE_N_HEAP 4       /* number of records in the heap */
#define PAGE_FREE 6         /* pointer to start of page free record list */
#define PAGE_GARBAGE 8      /* number of bytes in deleted records */
#define PAGE_LAST_INSERT 10 /* pointer to the last inserted record */
#define PAGE_DIRECTION 12   /* last insert direction: PAGE_LEFT, ... */
#define PAGE_N_DIRECTION 14 /* number of consecutive inserts to the same direction */
#define PAGE_N_RECS 16 /* number of user records on the page */
#define PAGE_MAX_TRX_ID 18 /* highest id of a trx which may have modified a record on the page;
trx_id_t; defined only in secondary indexes and in the insert buffer tree */
...
#define PAGE_LEVEL 26 /* level of the node in an index tree; the leaf level is the level 0.  This
field should not be written to after page creation. */
#define PAGE_INDEX_ID 28 /* index id where the page belongs. This field should not be written to
after page creation. */
```

System Records

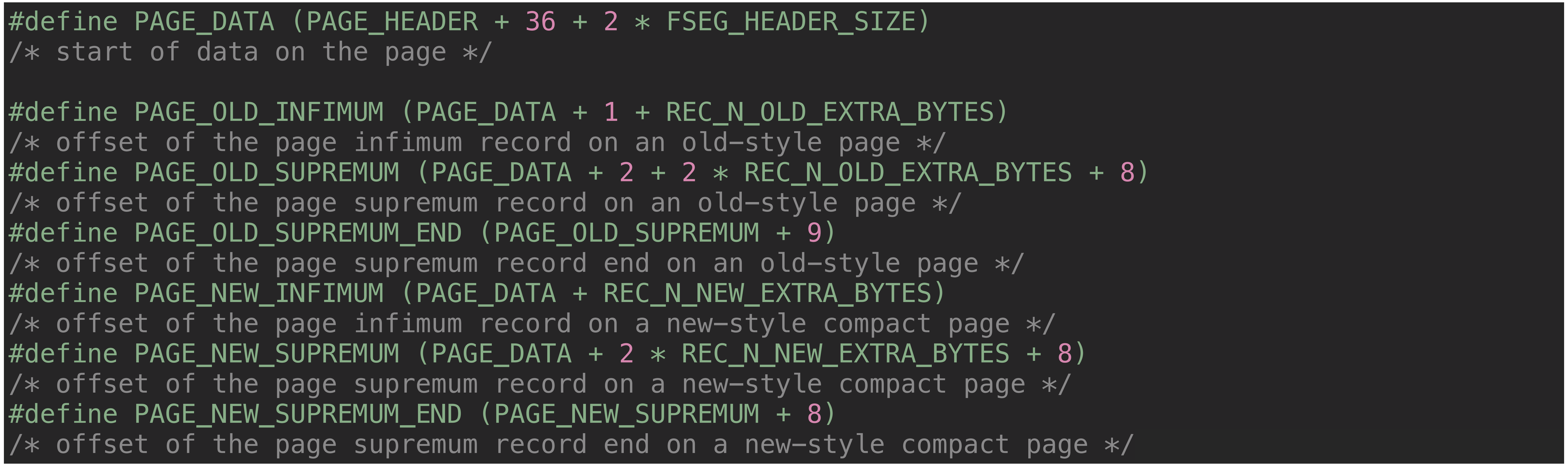
- Every INDEX page contains two system records, called `infimum` and `supremum`, at fixed locations (**offset 99 & offset 112**)
- The structure is as follows (in bits):



94
Info Flags (4)
Nuber of Records Owned (4)
95
Order (13)
Record Type (3)
97
Next Record Offset (16)
99
“infimum\0” (64)
107
Info Flags (4)
Number of Records Owned (4)
108
Order (13)
Record Type (3)
110
Next Record Offset (16)
112
“supremum” (64)
120

System Records

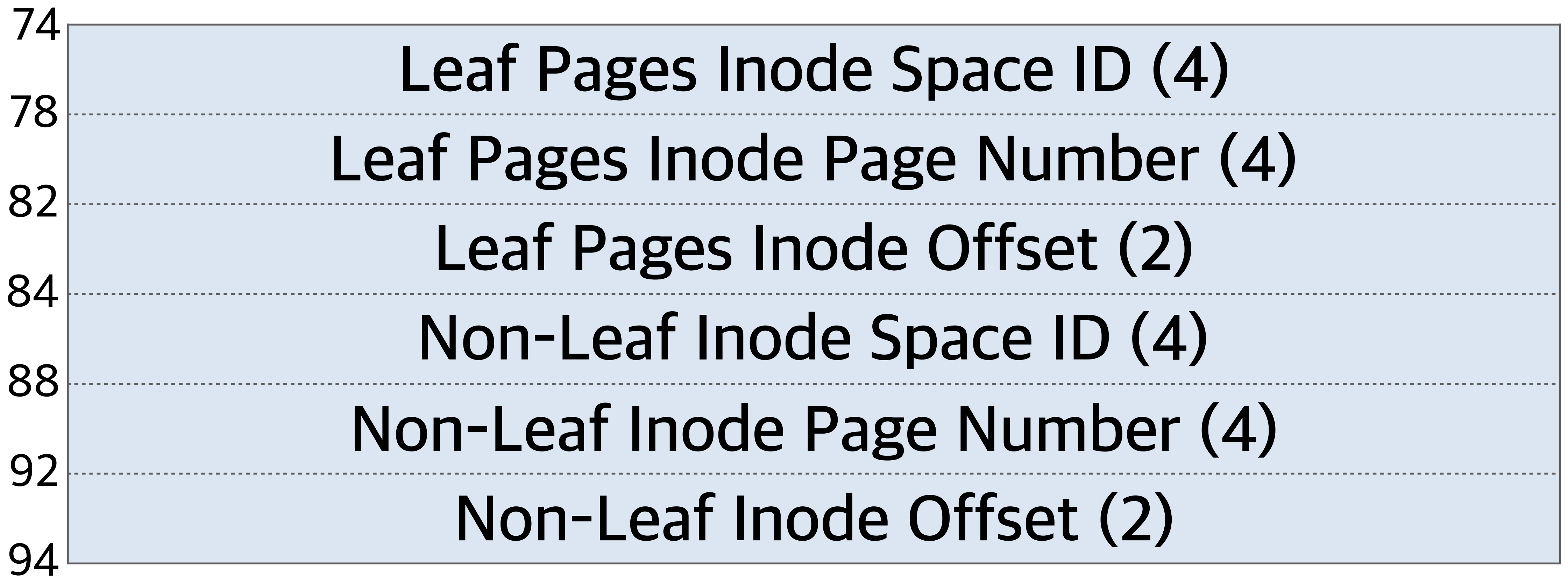
- include/page0types.h



```
#define PAGE_DATA (PAGE_HEADER + 36 + 2 * FSEG_HEADER_SIZE)
/* start of data on the page */

#define PAGE_OLD_INFIMUM (PAGE_DATA + 1 + REC_N_OLD_EXTRA_BYTES)
/* offset of the page infimum record on an old-style page */
#define PAGE_OLD_SUPREMUM (PAGE_DATA + 2 + 2 * REC_N_OLD_EXTRA_BYTES + 8)
/* offset of the page supremum record on an old-style page */
#define PAGE_OLD_SUPREMUM_END (PAGE_OLD_SUPREMUM + 9)
/* offset of the page supremum record end on an old-style page */
#define PAGE_NEW_INFIMUM (PAGE_DATA + REC_N_NEW_EXTRA_BYTES)
/* offset of the page infimum record on a new-style compact page */
#define PAGE_NEW_SUPREMUM (PAGE_DATA + 2 * REC_N_NEW_EXTRA_BYTES + 8)
/* offset of the page supremum record on a new-style compact page */
#define PAGE_NEW_SUPREMUM_END (PAGE_NEW_SUPREMUM + 8)
/* offset of the page supremum record end on a new-style compact page */
```

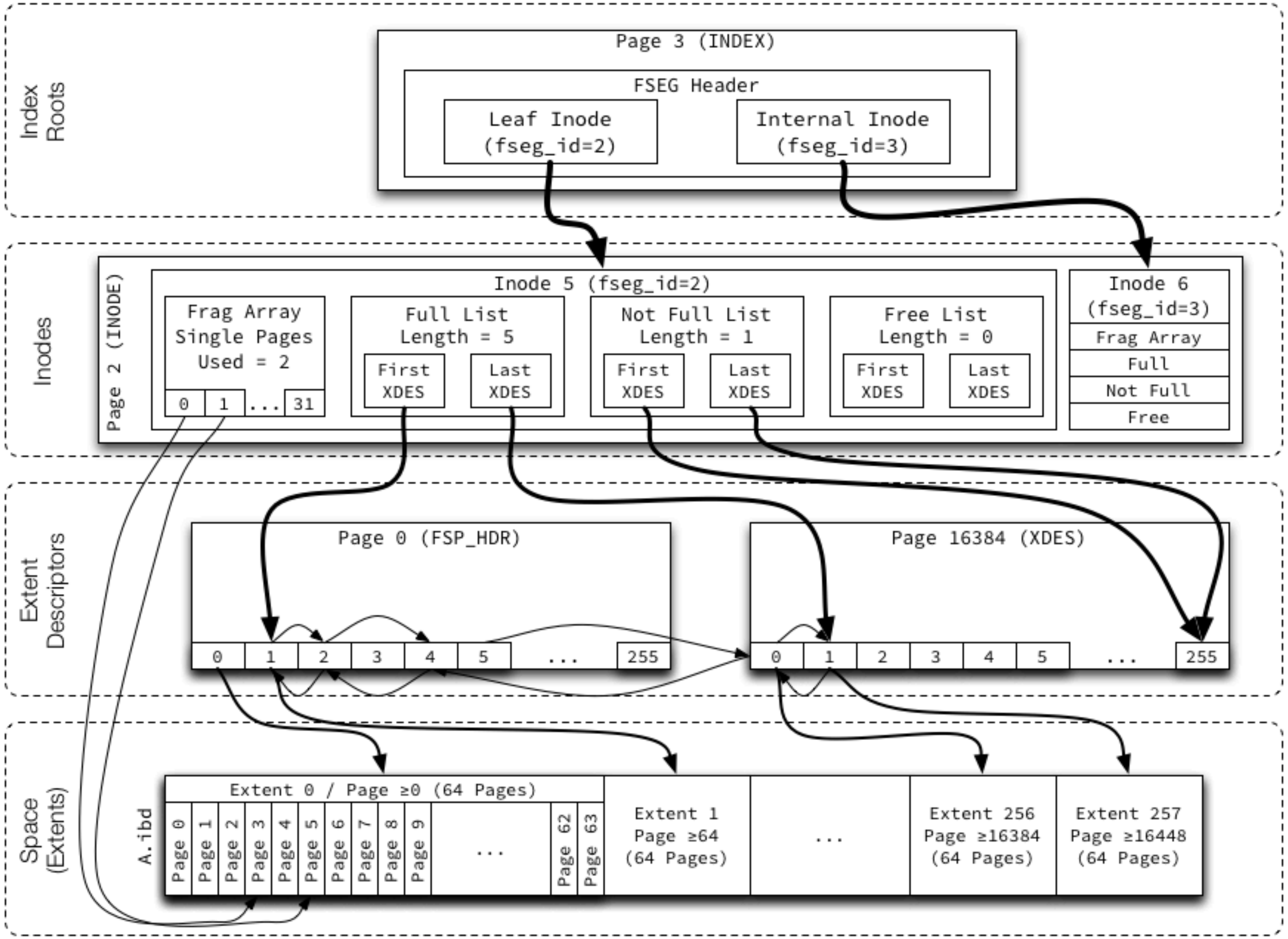
FSEG Header

- Each **index** uses one file segment for leaf pages and one for non-leaf (internal) pages
- This information is stored in the FSEG header structure in the INDEX **root** page
  - All other INDEX pages' FSEG headers are unused and zero-filled



74
Leaf Pages Inode Space ID (4)
78
Leaf Pages Inode Page Number (4)
82
Leaf Pages Inode Offset (2)
84
Non-Leaf Inode Space ID (4)
88
Non-Leaf Inode Page Number (4)
92
Non-Leaf Inode Offset (2)
94

Index File Segment Structure



How does it work when we CREATE/DROP an index?

- As soon as an index is created, **two file segments** will be allocated for the index:
  - One for **leaf** pages which will have **no page** as of now
  - One for **non-leaf** pages which will have only **one** single page allocate (**root page**)
- When B-Tree grows:
  - New pages are allocated in `Fragment Array`
  - Once demands cross 32 pages, an extent is allocated to segment and is moved to `FREE` list
  - Once pages of this new extent are used, it is moved to `NOT_FULL` list
  - Once all pages of this extent are used, it is moved to `FULL` list
- Once we drop the index:
  - From the **root** page, we go ahead and **mark all the extents in those two file segments as free**

Reference

[1] MySQL 8.0 Reference Manul: 15.6 InnoDB On-Disk Structures, MySQL, https://dev.mysql.com/doc/refman/8.0/en/innodb-tablespace.html

[2] MySQL 8.0 Reference Manul: 15.11 InnoDB Disk I/O and File Space Management, MySQL, https://dev.mysql.com/doc/refman/8.0/en/innodb-disk-management.html

[3] Innodb, Jeremy Cole, https://blog.jcole.us/innodb/

[4] InnoDB : Tablespace Space Management, Mayank Prasad, MySQL Server Blog, https://mysqlserverteam.com/innodb-tablespace-space-management/