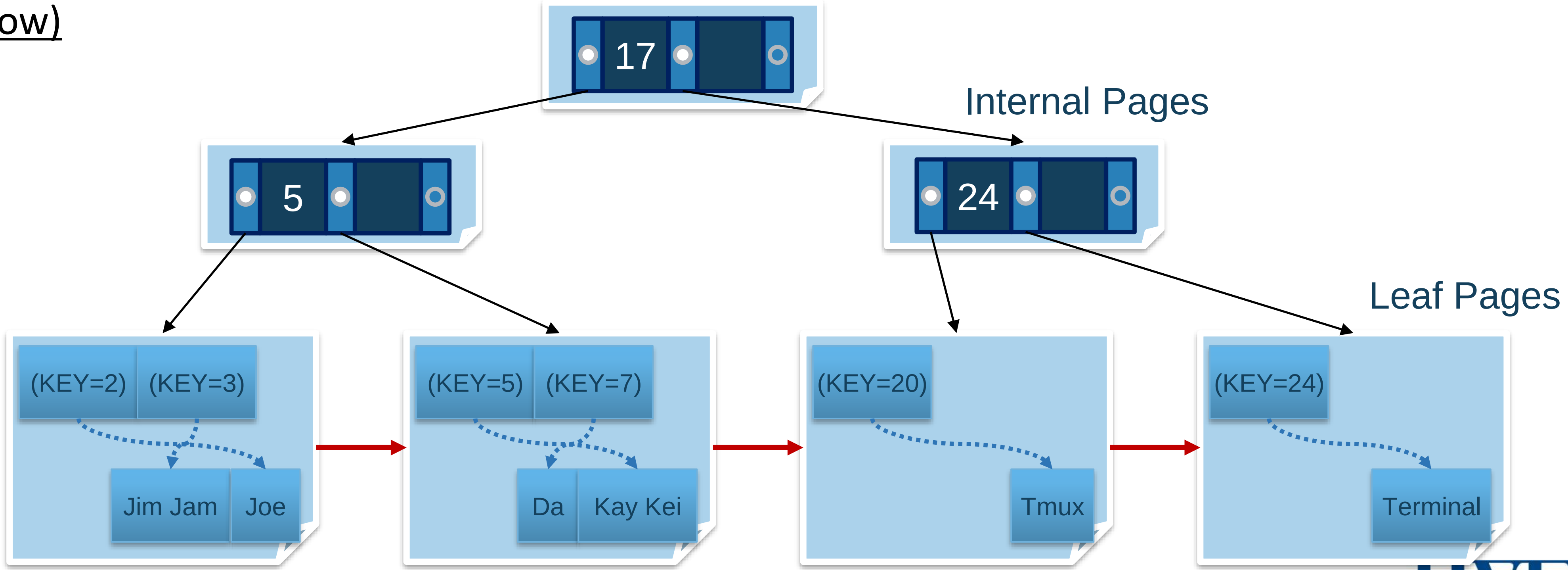
Project2 milestone2

Disk Based B+ Tree

Disk-based B+tree

- Note that the current design only considers *in-memory b+tree* and a *fixed size record*.
- Our goal:

  <u>Implement a **disk-based** b+ tree supporting a variable-length field. (like an example below)</u>



Project Specification

➢Your library (libdb.a) should provide the following APIs.

1. **int64_t open_table (char *pathname);**
   - Open existing data file using ‘pathname’ or create one if not existed.
   - If success, return the **unique table id**, which represents the own table in this database. Otherwise, return negative value.
2. **int db_insert (int64_t table_id, int64_t key, char * value, uint16_t val_size);**
   - Insert input ‘key/value’ (record) with its size to data file at the right place.
   - If success, return 0. Otherwise, return non-zero value.
3. **int db_find (int64_t table_id, int64_t key, char * ret_val, uint16_t * val_size);**
   - Find the record containing input ‘key’.
   - If found matching ‘key’, store matched ‘value’ string in ret_val and matched ‘size’ in val_size.
     If success, return 0. Otherwise, return non-zero value.
   - The “caller” should allocate memory for a record structure (ret_val).
4. **int db_delete (int64_t table_id, int64_t key);**
   - Find the matching record and delete it if found.
   - If success, return 0. Otherwise, return non-zero value.

Project Specification

- Your library (libdb.a) should provide the following APIs.

1. **int init_db ();**
   - Initialize your database management system.
   - Initialize and allocate anything you need.
   - The total number of tables is less than 20.
   - If success, return 0. Otherwise, return non-zero value.
2. **int shutdown_db();**
   - Shutdown your database management system**.**
   - Clean up everything.
   - If success, return 0. Otherwise, return non-zero value.

Project Specification

- All update operations (insert/delete) should be applied to your data file **as an operation unit.** That means one update operation should change the data file layout correctly.
- Note that your code **must work on** other students' data file. That means, **your code should handle open(), insert(), find() and delete() API with other students' data file as well**.
- So, **follow the data file layout** described from next slides.

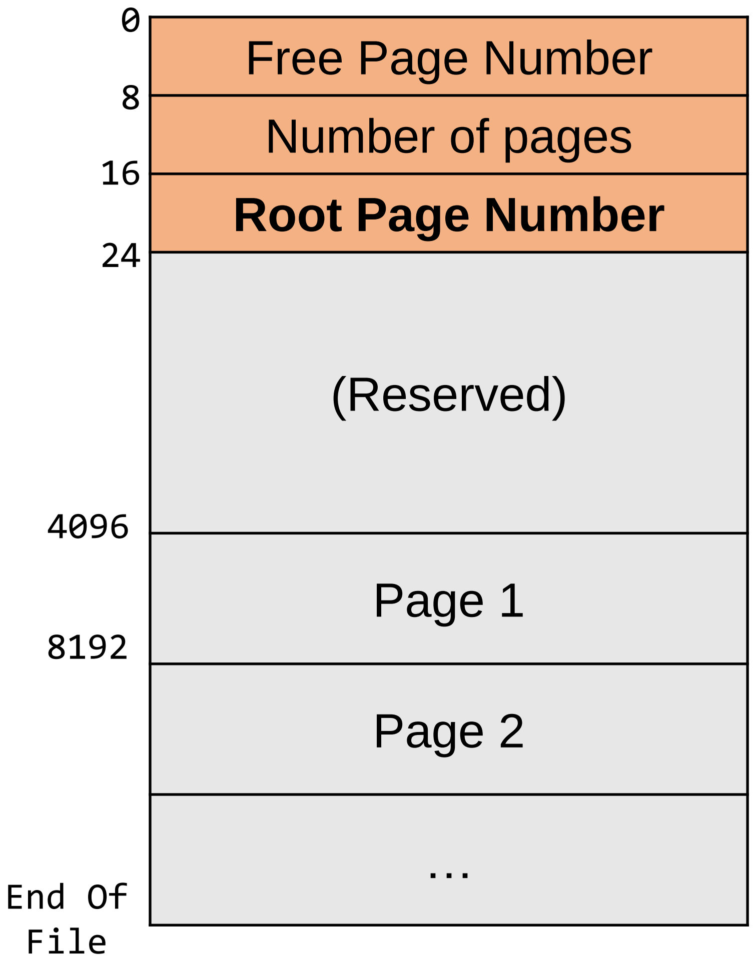
Project Specification

- We fixed the on-disk page size with **4096** Bytes.
- There are 4 types of page. (detail next slides..)
  1. **Header page** (special, containing metadata)
  2. **Free page** (maintained by free page list)
  3. **Leaf page** (containing records)
  4. **Internal page** (internal index pages)

Header Page (Special)

- ➢Header page is the **first page (offset 0-4095)** of a data file and contains metadata.
- ➢When we open the data file at first, initializing disk-based b+tree should be done using this header page.

- Free page number: [0-7]
  - points the first free page (head of free page list)
  - 0, if there is no free page left.
- Number of pages: [8-15]
  - how many pages exist in this data file now. (Count the header page itself as well)
- Root page number: [16-23]
  - pointing the root page within the data file.
  - 0, if there is no root page.



0

Free Page Number

8

Number of pages

16

Root Page Number

24

(Reserved)

4096

Page 1

8192

Page 2

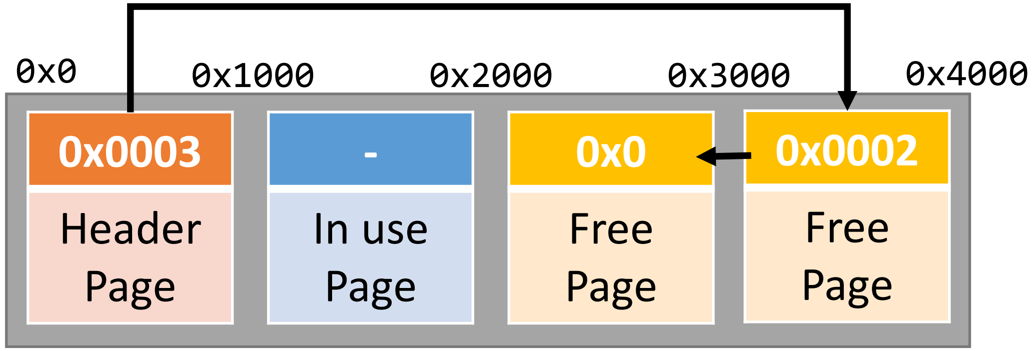
…

End Of File

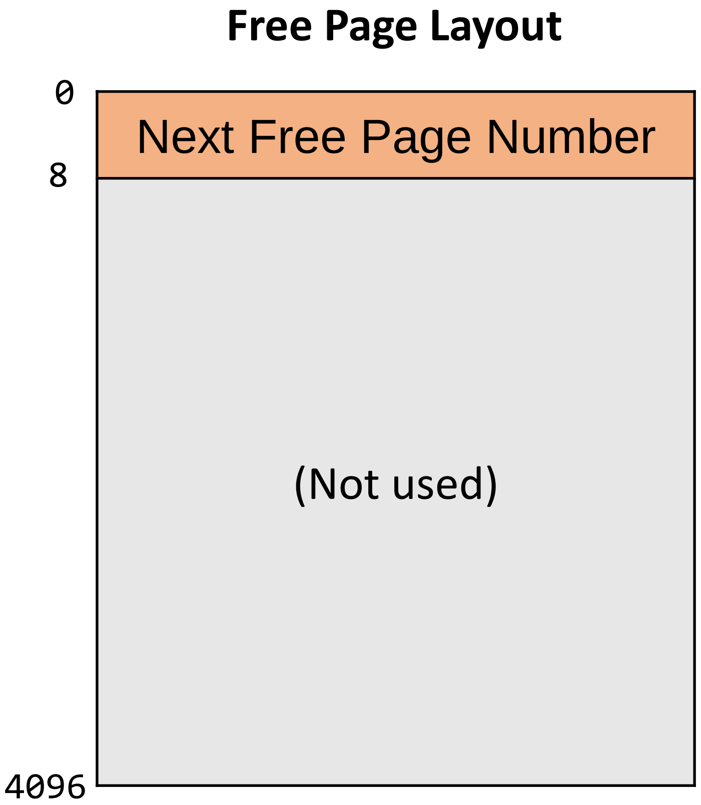
Free Page

- In the previous slide, header page contains the position of the *first free page*.
- Free pages are all linked and page allocation is managed by the free page list.

- Next free page Number: [0-7]
  - points the next free page.
  - 0, if end of the free page list.



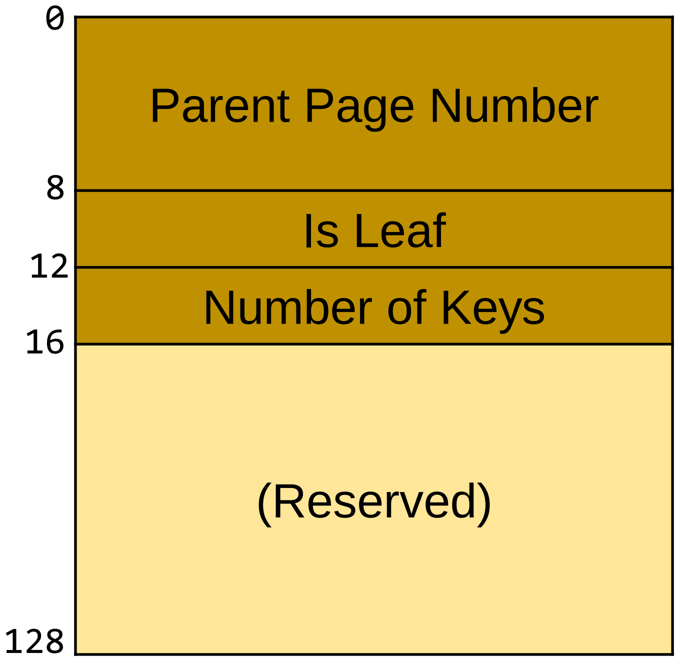
Data(Table) file example



Page Header

- ➢Internal/Leaf page have **first 128 bytes** as a page header.
- ➢Leaf/Internal page should contain those data (see the *node* structure in include/bpt.h)
  - Parent page Number [0-7]: If internal/leaf page, this field points the position of parent page.
    Set 0 if it is the root page.
  - Is Leaf [8-11] : 0 is internal page, 1 is leaf page.
  - Number of keys [12-15] : the number of keys within this page.

Page Header Layout



0

Parent Page Number

8

Is Leaf

12

Number of Keys

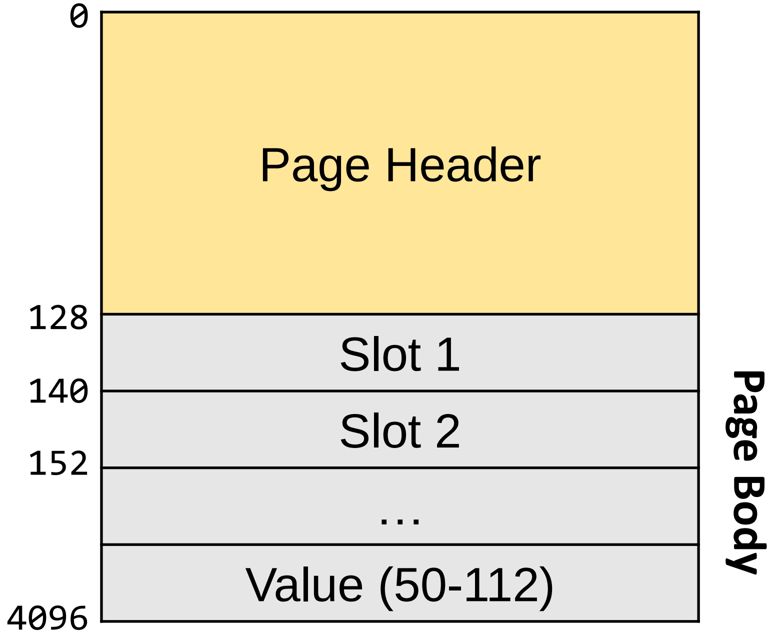
16

(Reserved)

128

Leaf Page

Leaf Page Layout



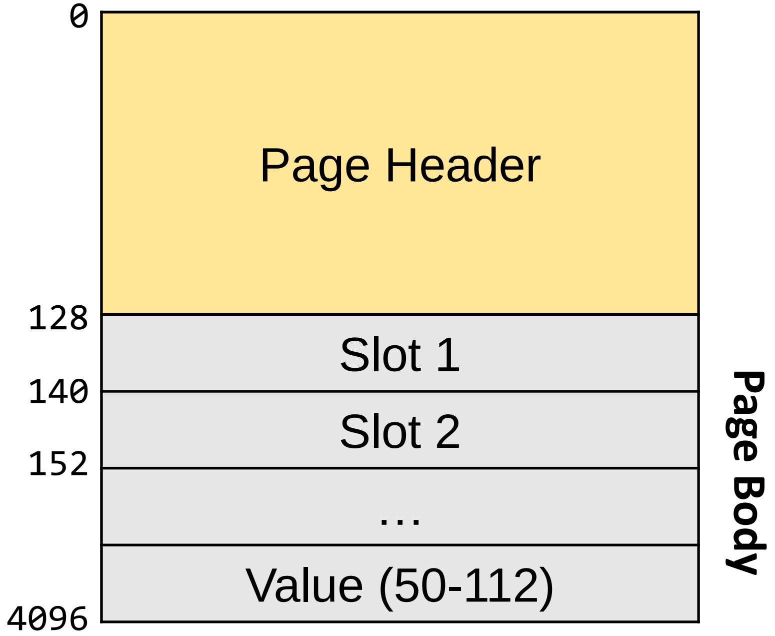
Slot format

Key (8)	Size (2)	Offset (2)

- Leaf page is a slotted page.
- Leaf page contains the key (fixed size) and value (variable size).
- First 128 bytes will be used as a page header.
- Each slot is 12bytes, and it consists of key (8 bytes), size of value (2 bytes), and in-page offset (2 bytes).

Leaf Page (cont.)

Leaf Page Layout



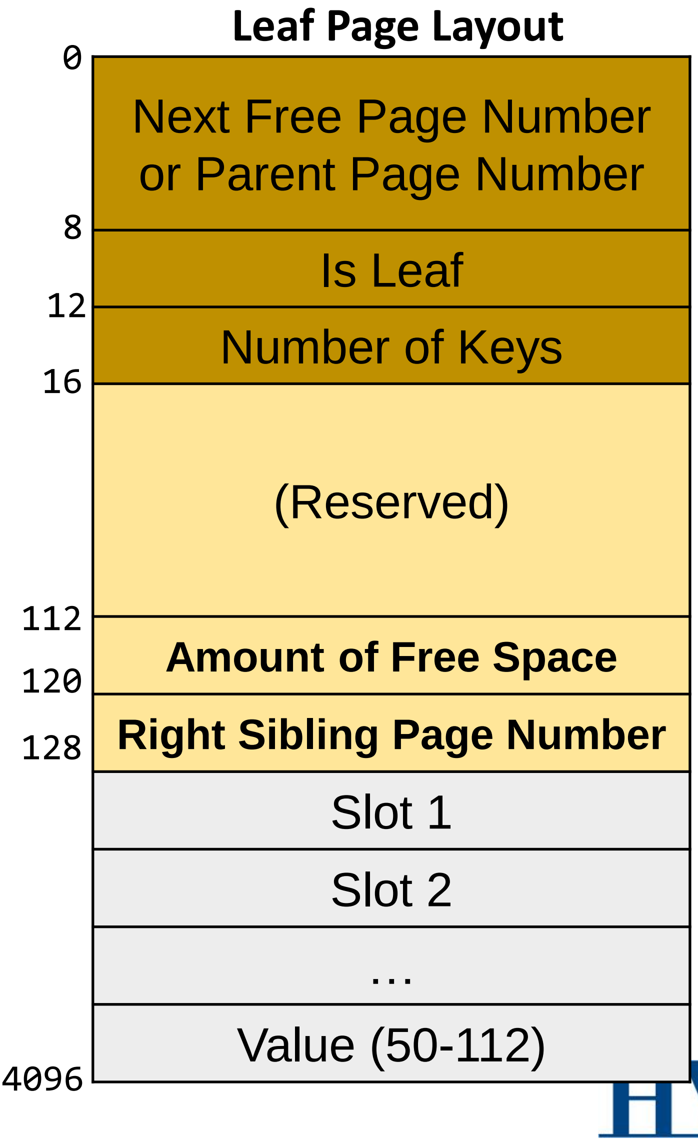
Slot format

Key (8)	Size (2)	Offset (2)

- ➢Keys are sorted in the page. (ascending)
- ➢Values do not need to be sorted.
- ➢Keys are inserted from the beginning of a page body, and corresponding values are inserted from the end. (slotted page)
- ➢ Size of Value : $50 \leq x \leq 112$ (bytes)
- ➢ # of slot : $32 \leq x \leq 64$

Leaf Page (cont.)

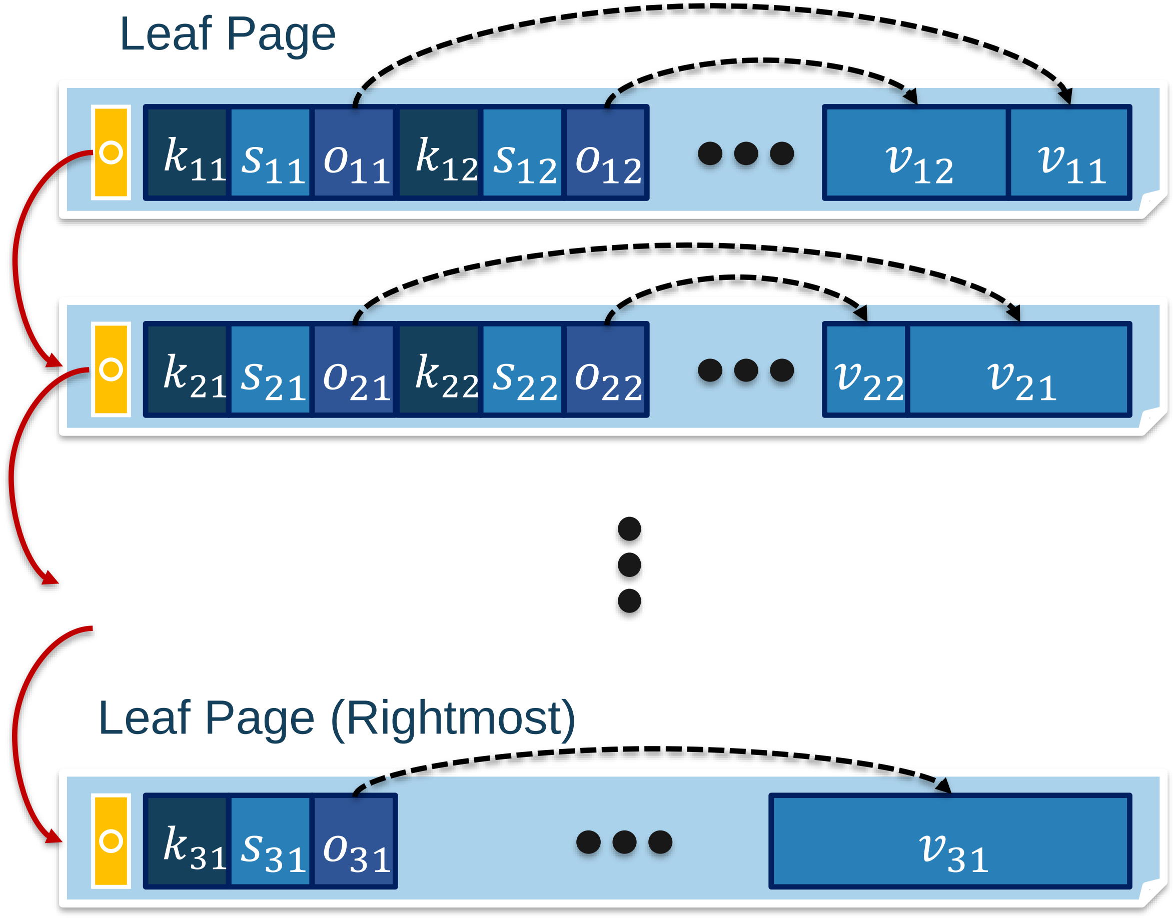
- There should be one more page number added to store right sibling page number for leaf page. (see the comments of *node* structure in include/bpt.h)
- The amount of free space helps to manage a leaf page.
- We define the amount of free space and the special page number at the end of page header.
- If rightmost leaf page, right sibling page number field is 0.
- The initial amount of free space should be 3968. (page size – page header size)



Leaf Page Layout

0
Next Free Page Number or Parent Page Number
8
Is Leaf
12
Number of Keys
16
(Reserved)
112
Amount of Free Space
120
Right Sibling Page Number
128
Slot 1
Slot 2
…
Value (50-112)
4096

Leaf Page (Cont.)

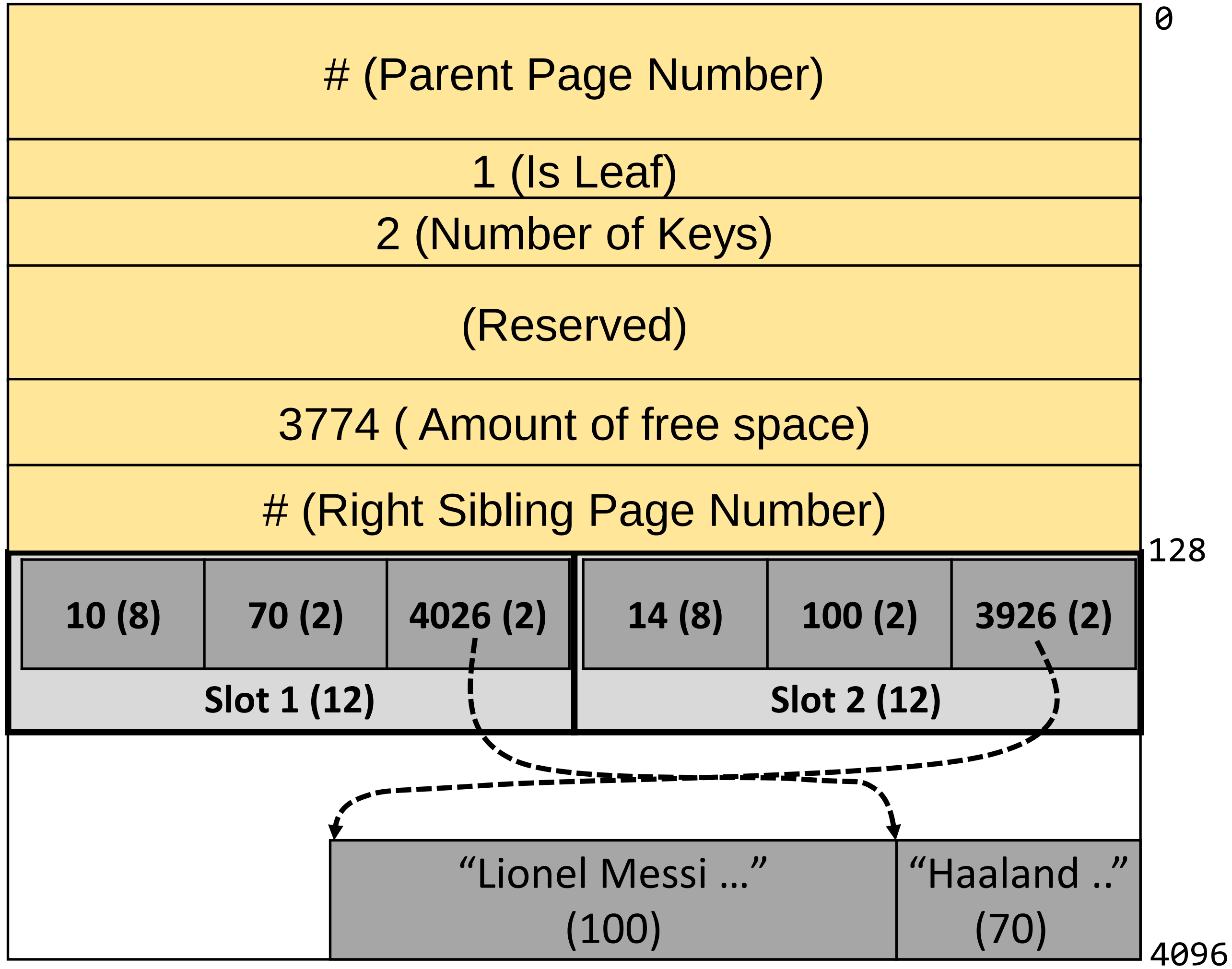

Leaf Page
k_{11} s_{11} o_{11} k_{12} s_{12} o_{12} ●●● v_{12} v_{11}
k_{21} s_{21} o_{21} k_{22} s_{22} o_{22} ●●● v_{22} v_{21}
Leaf Page (Rightmost)
k_{31} s_{31} o_{31} ●●● v_{31}

Leaf Page Layout

Offset	Field
0	Next Free Page Number or Parent Page Number
8	Is Leaf
12	Number of Keys
16	(Reserved)
112	Amount of Free Space
120	**Right Sibling Page Number**
128	Slot 1
	Slot 2
	…
	Value (50-112)
4096	

Leaf Page Example

Leaf Page Layout



# (Parent Page Number)

1 (Is Leaf)

2 (Number of Keys)

(Reserved)

3774 (Amount of free space)

# (Right Sibling Page Number)

10 (8) **70 (2)** **4026 (2)**

Slot 1 (12)

14 (8) **100 (2)** **3926 (2)**

Slot 2 (12)

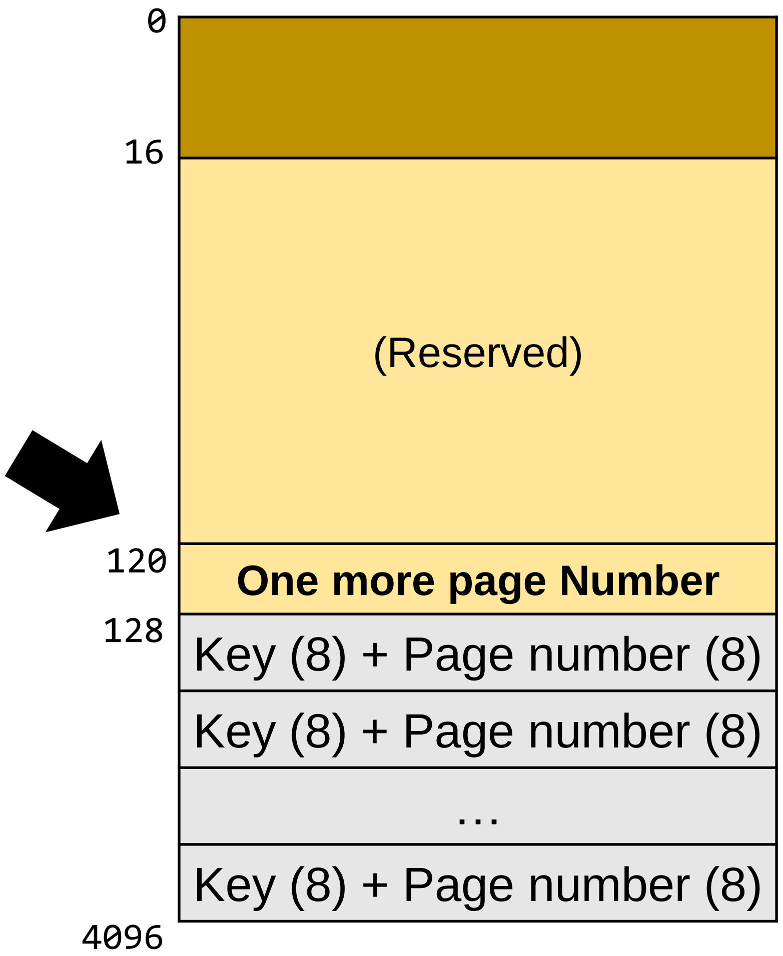
"Lionel Messi …" (100)

"Haaland .." (70)

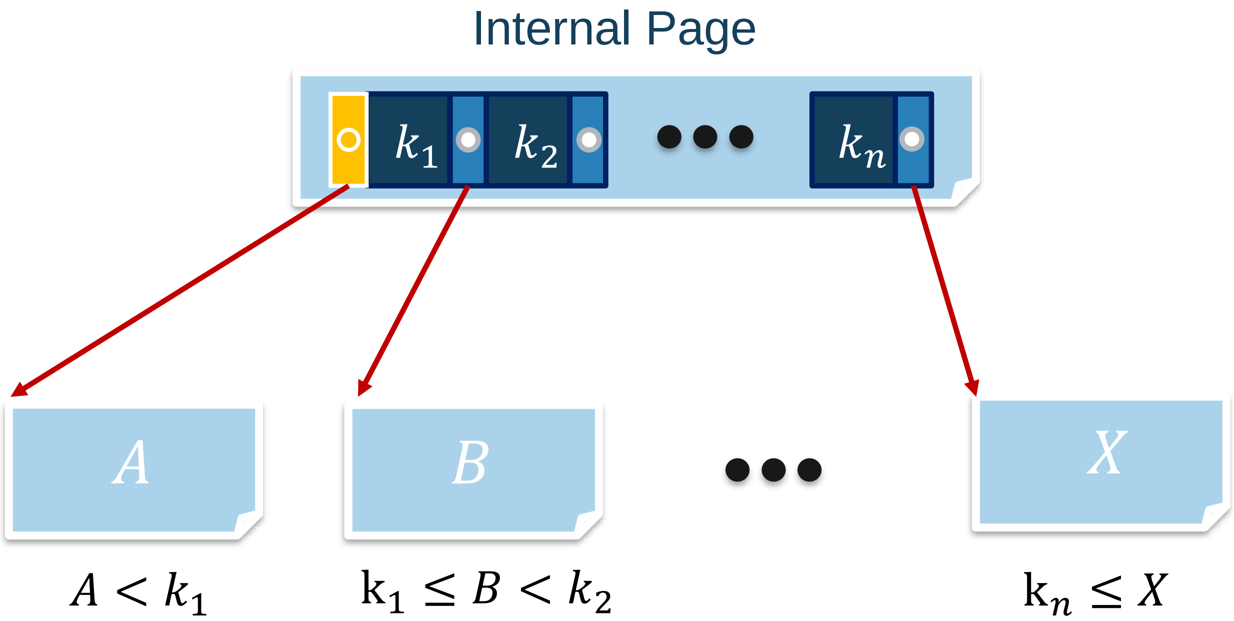
Internal Page

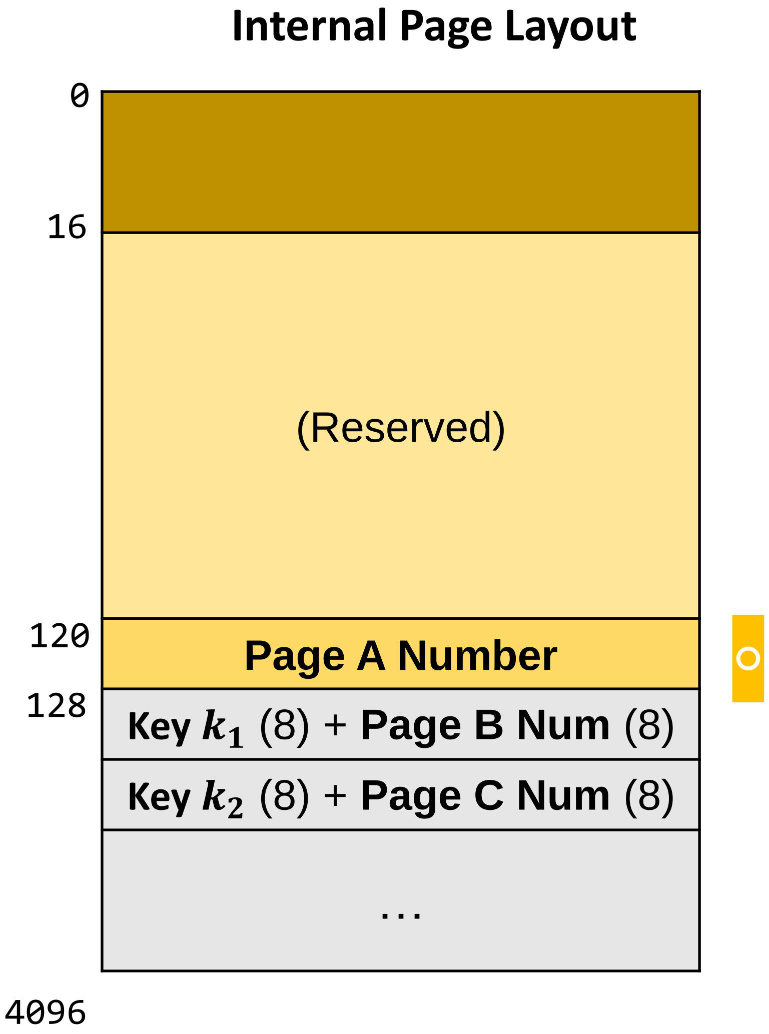
- Internal page is different from the leaf page, since it doesn't contain any variable length fields.
- Internal page needs one more page number to cover leftmost key range.
- Branching factor (2*order+1) = 249
  - Internal page can have maximum 248 entries, because 'key + page number' (8+8 bytes) can cover up to whole page (except page header) with the number of 248.
  - (4096 – 128) / (8+8) = 248

Internal Page Layout

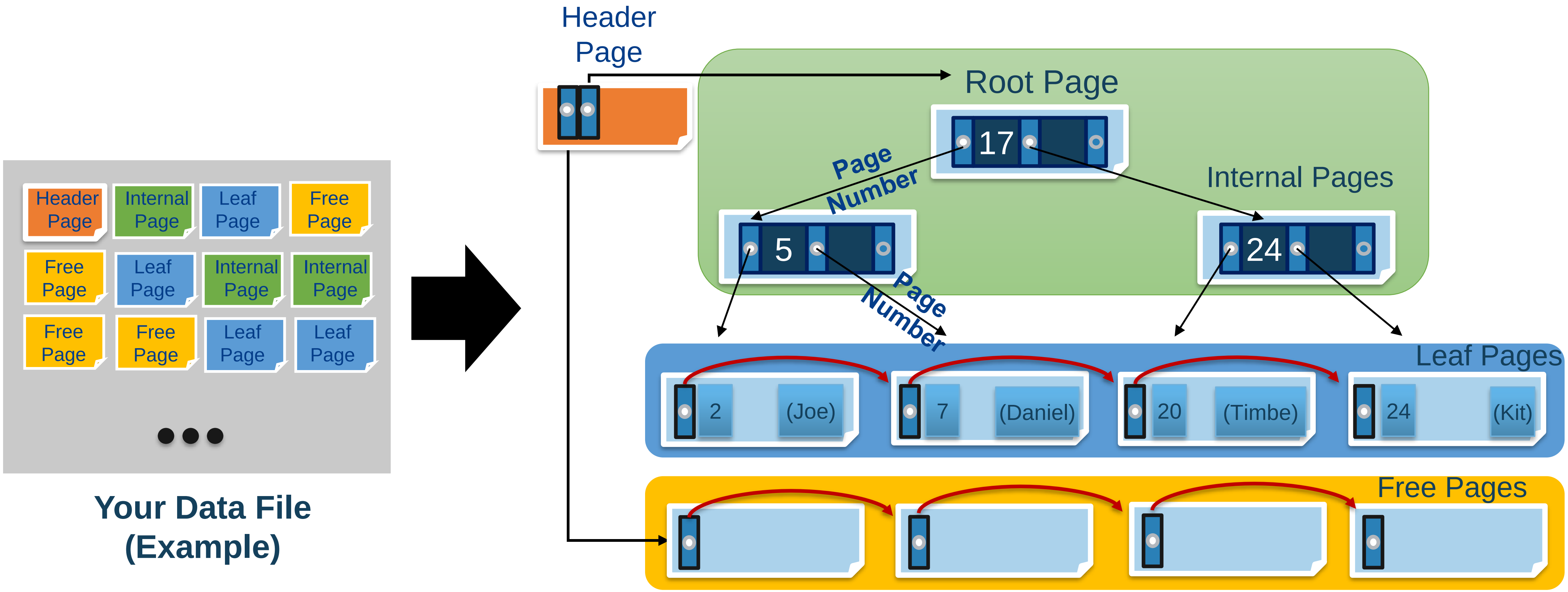

0
16
(Reserved)
120
One more page Number
128
Key (8) + Page number (8)
Key (8) + Page number (8)
…
Key (8) + Page number (8)
4096

Internal Page




Internal Page Layout
0
16
(Reserved)
120
Page A Number
128
Key k_1 (8) + Page B Num (8)
Key k_2 (8) + Page C Num (8)
…
4096

Disk-based B+tree Example



Your Data File (Example)

Insertion Rule for Leaf Pages

- Insert the new record into the target leaf page
  - Insert the new record as proper form and modify the value of the amount of free space in a page header
  - Case 1: Enough free space to insert
    - Nothing to do.
  - Case 2: No room for the new record
    - Need to split.
    - Set the first record that is equal to or greater than 50% (1984byte) of the total size on the Page Body as the point to split, and then move that record, and all records after that to the new leaf page.
- You must modify the value of the amount of free space in the page header in all cases if possible.

Deletion Rule For Leaf Pages

- Delete a record in the target leaf page
  - Delete a record and compact the target leaf page as possible as you can. And then modify the value of the amount of free space in a page header.
    - Compacting the leaf page every time could be inefficient but the efficiency of handling internal fragmentation is not the scope of this project.
  - Case 1: Free space < threshold (2500 bytes)
    - Nothing to do.
  - Case 2: Free space >= threshold
    - The target leaf page should be merged or redistributed. See the next slide.

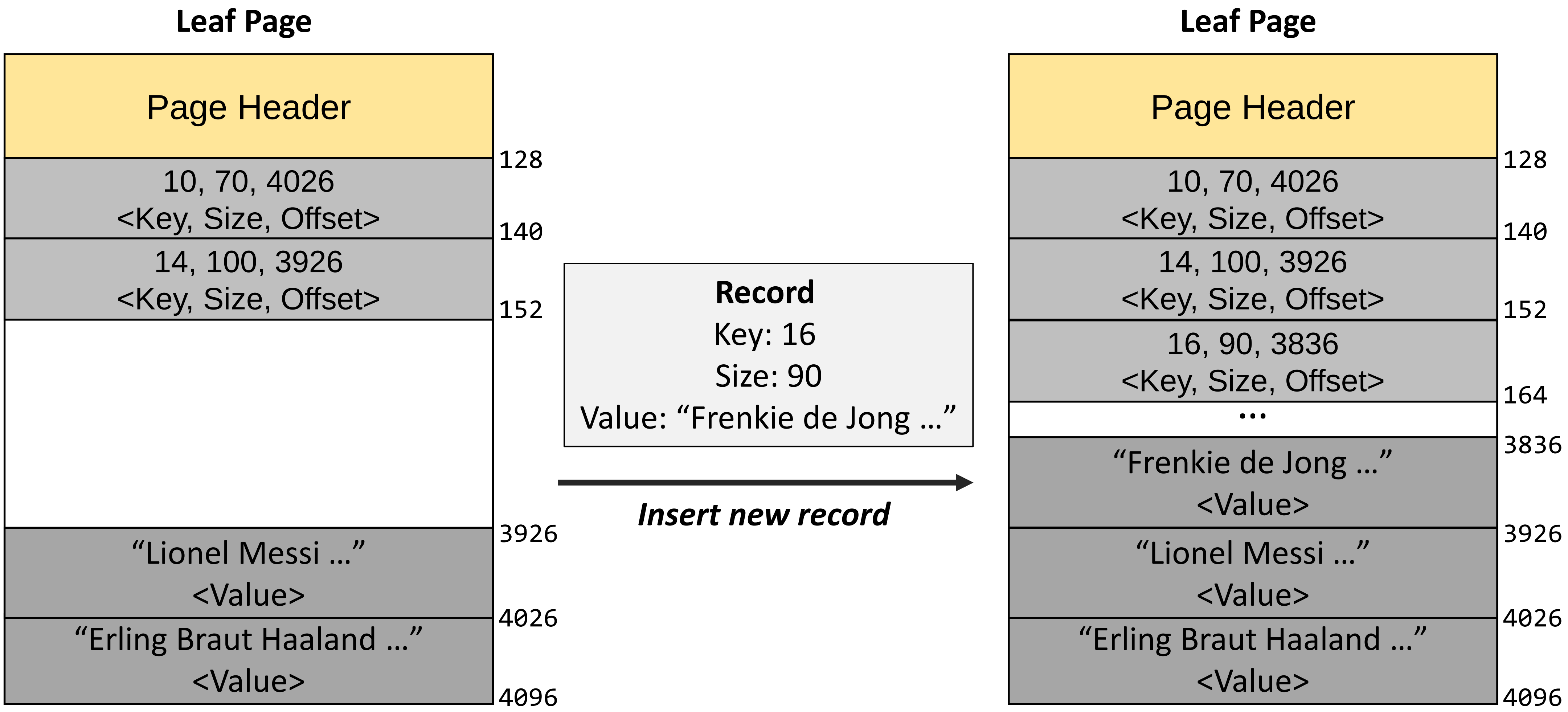
Deletion Rule For Leaf Pages(cont.)

- The method of selecting a sibling for merge or redistribution is the same as the existing logic. (see delete in bpt.cc)
- Case 2-1: Merge
  - Merge should occur only when the sibling has enough free space.
- Case 2-2: Redistribute
  - Redistribution should happen only when it cannot be merged.
  - Pull a record(s) from the sibling page until its free space becomes smaller than the threshold. (2500 bytes)
- You must modify the value of the amount of free space in the page header in all cases if possible.

Rules for Internal Pages

- There are no variable-length fields in internal pages, so all logics about insertion and deletion are same as the original one. (*see insert and delete in bpt.cc*)

Example1: Insert a Record into a Leaf Page



Leaf Page

Page Header

128

10, 70, 4026
<Key, Size, Offset>

140

14, 100, 3926
<Key, Size, Offset>

152

3926

"Lionel Messi …"
<Value>

4026

"Erling Braut Haaland …"
<Value>

4096

Record
Key: 16
Size: 90
Value: "Frenkie de Jong …"

Insert new record

Leaf Page

Page Header

128

10, 70, 4026
<Key, Size, Offset>

140

14, 100, 3926
<Key, Size, Offset>

152

16, 90, 3836
<Key, Size, Offset>

164

…

3836

"Frenkie de Jong …"
<Value>

3926

"Lionel Messi …"
<Value>

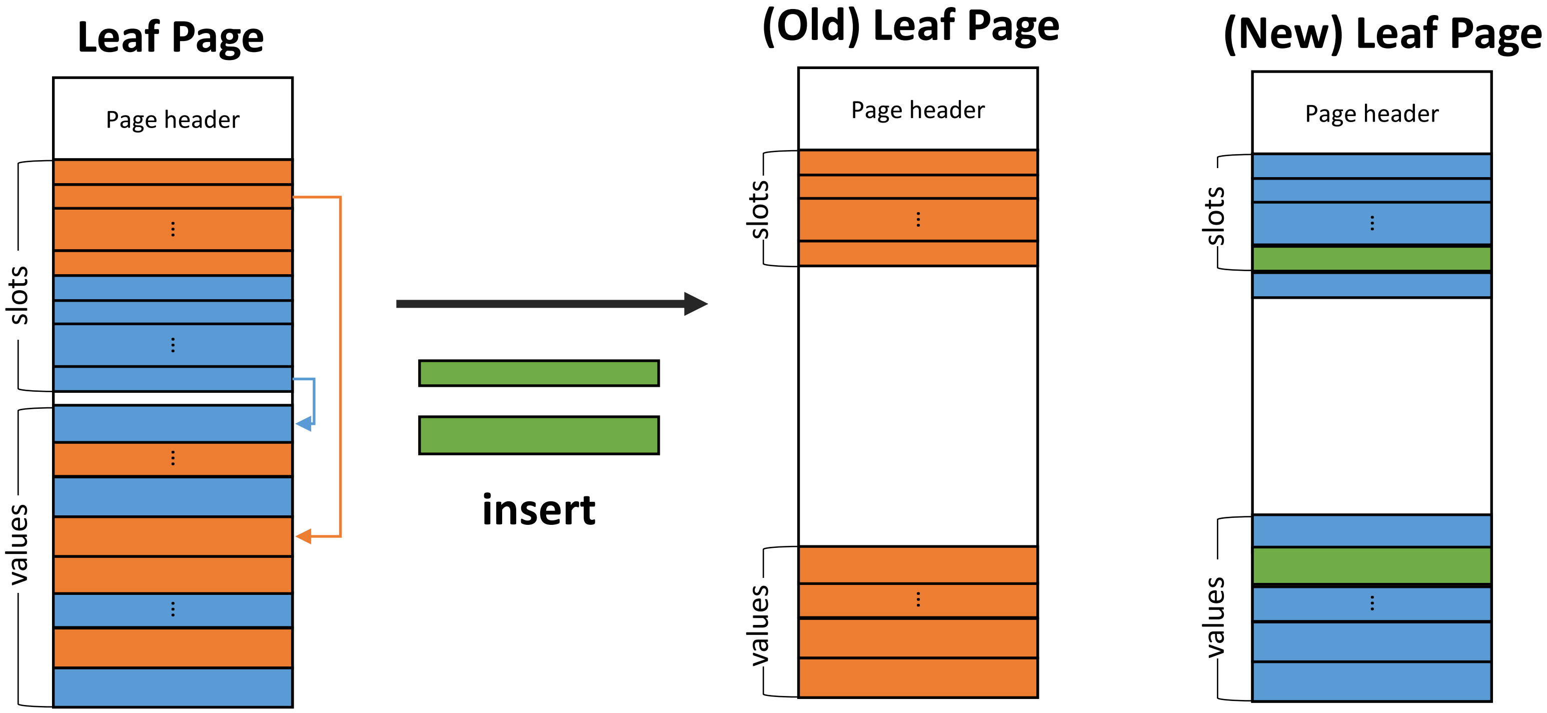
4026

"Erling Braut Haaland …"
<Value>

4096

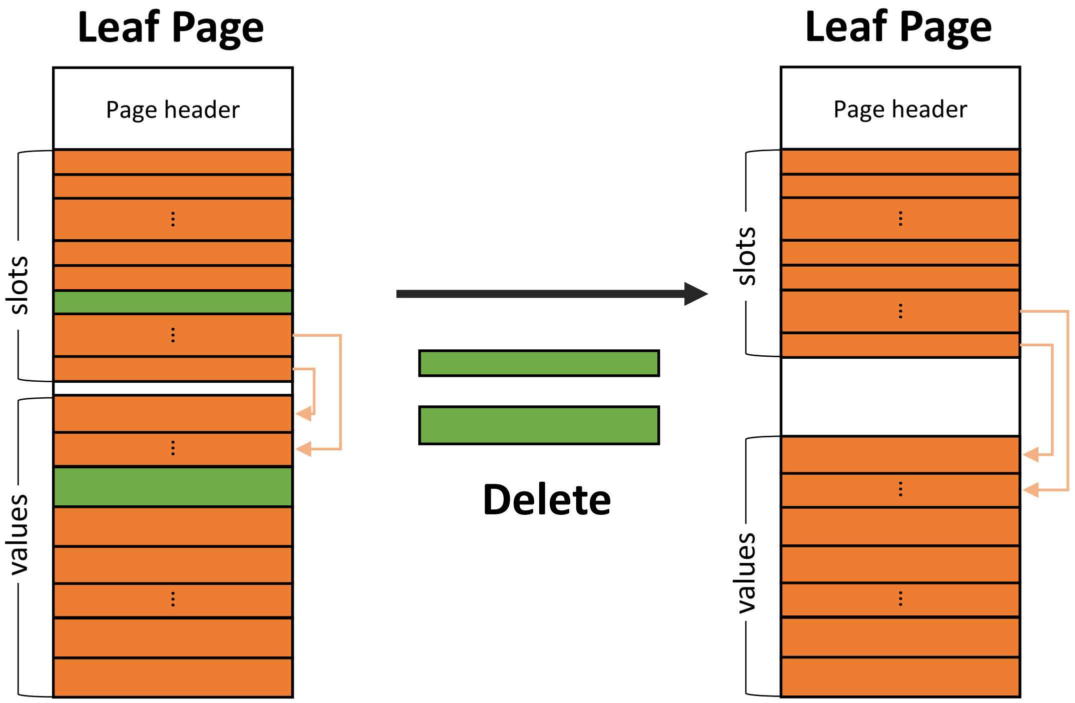
Example2: Insert a Record into a Leaf Page

🟧 : Total size of slots and values is 1983 bytes



Example3: Delete a Record in a Leaf Page

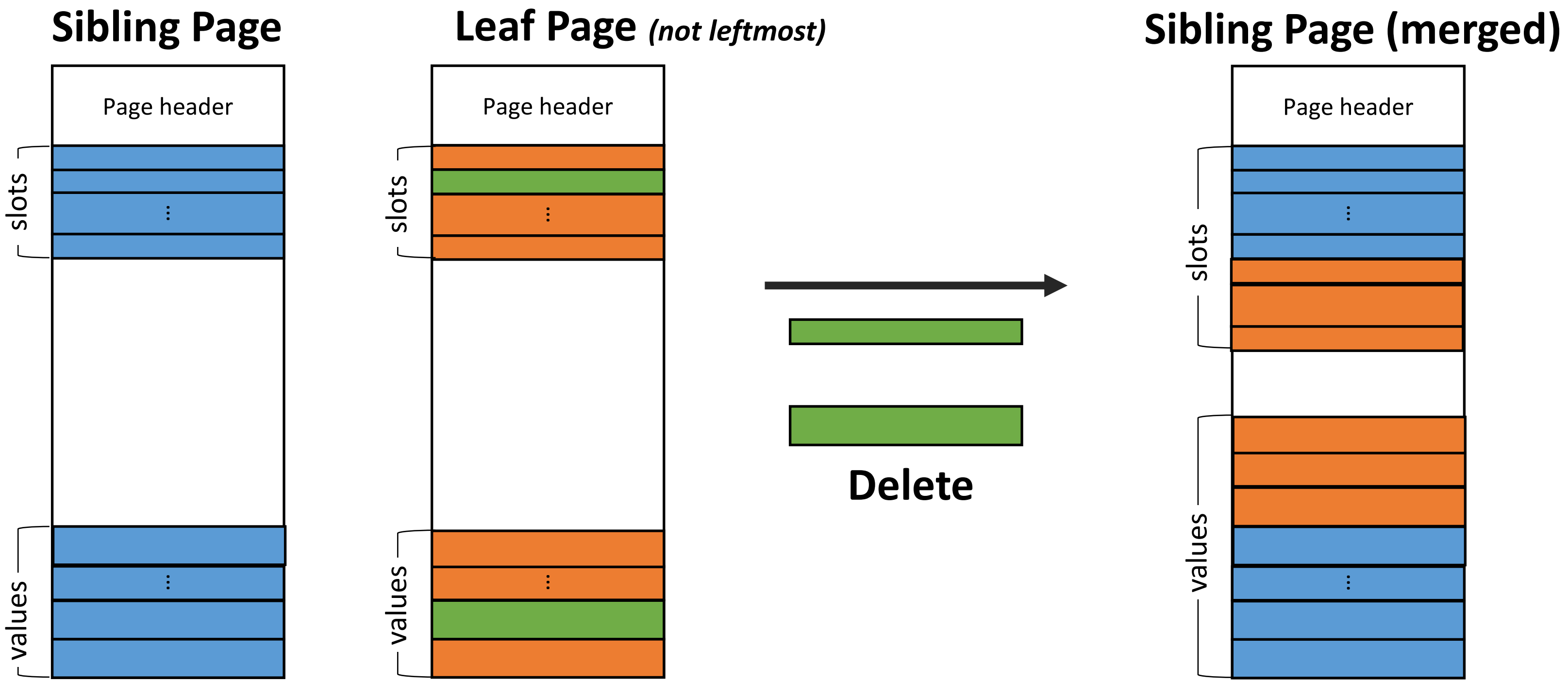
🟧 : Total size of slots and values is 3000 bytes



Example4: Delete a Record in a Leaf Page

🟧 : Total size of slots and values is 1400 bytes

🟩 : Total size of slot and value is 100 bytes

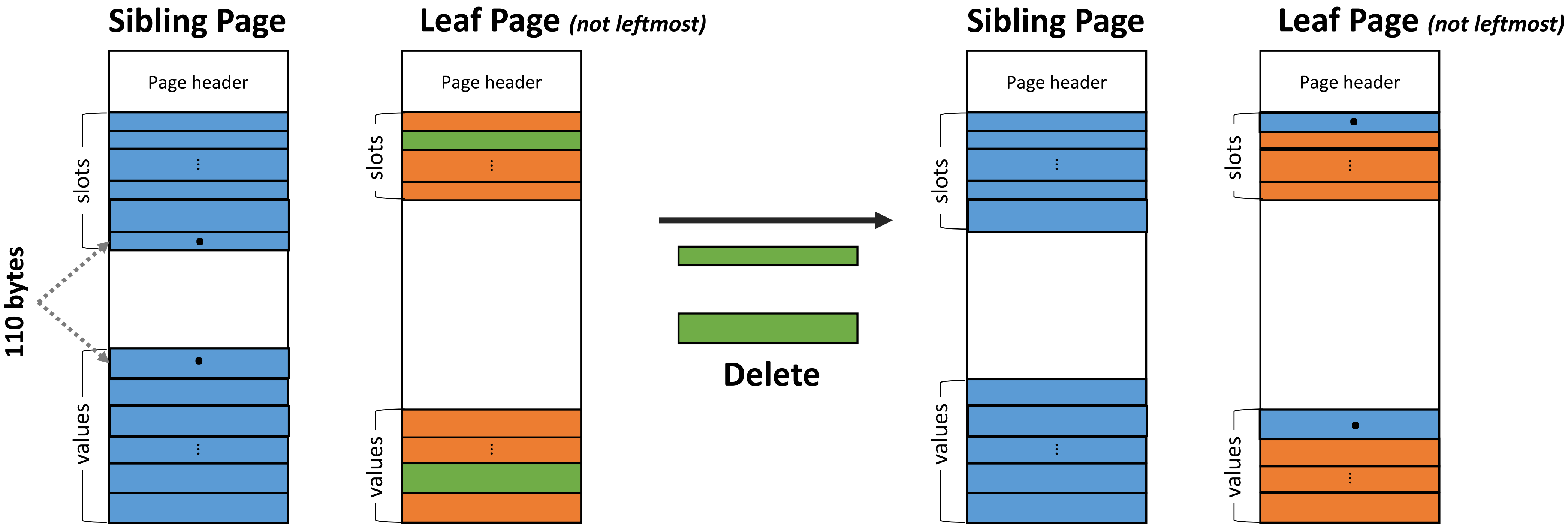


🟦 : Total size of slots and values is 2000 bytes

Example5: Delete a Record in a Leaf Page

🟧 : Total size of slots and values is 1400 bytes

🟩 : Total size of slot and value is 100 bytes



🟦 : Total size of slots and values is 3000 bytes

Disk Space Manager Specification

- **Modify six APIs** that should be exported to its upper layer :
  open table file / allocate page / free page / read page / write page / close table files
- **All I/O must be performed in a 4KiB (page) unit.**
- **Maintain the structure for mapping table id to file descriptor in Disk Space Manager**

1. **int64_t file_open_table_file (const char * pathname)**
   - Open the table file.
   - It opens an existing table file using 'pathname' or create a new file if absent.
   - If a new file needs to be created, the default file size should be **10 MiB**.
   - Then it returns the table id of the opened table file.
   - All other 5 commands below should be handled after open table file.
2. **uint64_t file_alloc_page (int64_t table_id);**
   - Allocate a page.
   - It returns a new page # from the free page list.
   - If the free page list is empty, then it should grow the table file and return a free page #.

Disk Space Manager Specification

3. **void file_free_page (int64_t table_id, uint64_t page_number);**
   - Free a page.
   - It informs the disk space manager of returning the page with ‘table id and page_number’ for freeing it to the free page list.
4. **void file_read_page (int64_t table_id, uint64_t page_number, char * dest);**
   - Read a page.
   - It fetches the disk page corresponding to ‘table id and page_number’ to the in-memory buffer (i.e., ‘dest’).
5. **void file_write_page (int64_t table_id, uint64_t page_number, const char * src);**
   - Write a page.
   - It writes the in-memory page content in the buffer (i.e., ‘src’) to the disk page pointed by ‘table id and page_number’.
6. **void file_close_table_files();**
   - Close all table files.

Disk Space Manager APIs

–exported to its upper layer-

```c
// page.h
#include <stdint.h>

typedef uint64_t pagenum_t;
struct page_t {
    // in-memory page structure
};

// Add any structures you need
```

```c
// file.h
#include <page.h>

// Open existing table file or create one if not existed.
int64_t file_open_table_file(const char* pathname);

// Allocate an on-disk page from the free page list
pagenum_t file_alloc_page(int64_t table_id);

// Free an on-disk page to the free page list
void file_free_page(int64_t table_id, pagenum_t pagenum);
```

Disk Space Manager APIs (cont.)
–exported to its upper layer-

```
// file.h

// Read an on-disk page into the in-memory page structure(dest)
void file_read_page(int64_t table_id,
        pagenum_t pagenum, page_t* dest);

// Write an in-memory page(src) to the on-disk page
void file_write_page(int64_t table_id, pagenum_t pagenum,
                     const page_t* src);

// Close all table files
void file_close_table_files();
```

Disk Space Manager APIs (cont.)

```
// file.cc

// Open existing table file or create one if not existed.
int64_t file_open_table_file(const char* pathname){
}

// Allocate an on-disk page from the free page list
pagenum_t file_alloc_page(int64_t table_id){
}

// Free an on-disk page to the free page list
void file_free_page(int64_t table_id, pagenum_t pagenum){
}
```

Disk Space Manager APIs (cont.)

```
// file.cc

// Read an on-disk page into the in-memory page structure(dest)
void file_read_page(int64_t table_id, pagenum_t pagenum,
                    page_t* dest){
}

// Write an in-memory page(src) to the on-disk page
void file_write_page(int64_t table_id, pagenum_t pagenum,
                     const page_t* src){
}

// Close all table files
void file_close_table_files(){
}
```

Milestone & DEADLINE

- Analyze the given b+ tree code and submit a report to the hconnect Wiki.
  - Your report should includes
    1. Possible call path of the insert/delete operation
    2. Detail flow of the structure modification (split, merge)
    3. (Naïve) designs or required changes for building on-disk b+ tree
- Implement on-disk b+ tree and submit a report(Wiki) including your design.
- Deadline: Oct 18 11:59pm
- We'll only score your commit before the deadline and your submission after that deadlines will not accepted.

Thank you